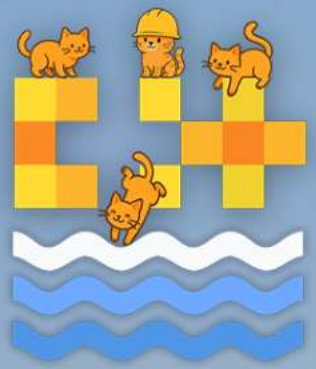
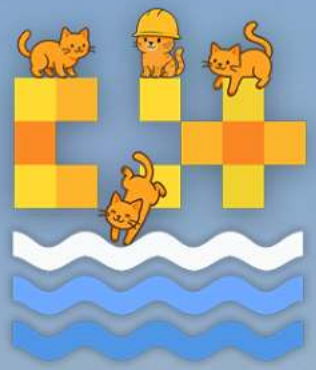


The 10 Essential Features for the Future of C++ Libraries

Mateusz Pusz

2025



The 18 Essential Features for the Future of C++ Libraries

Mateusz Pusz

2025

C++ Evolution

C++ Evolution

- C++11 - Cornerstone of Modern C++
  - Move Semantics, **`constexpr`**, lambda expressions, variable templates, memory model and threads, range-for, chrono, ...

C++ Evolution

- C++11 - Cornerstone of Modern C++
  - Move Semantics, **`constexpr`**, lambda expressions, variable templates, memory model and threads, range-for, chrono, ...
- C++20 - HUGE!!!
  - concepts, ranges, coroutines, class NTTPs, **`constexpr`** all the things, **`consteval`**, **`<=>`**, **`format`**, **`counting_semaphore/latch/barrier`**, modules 🤔, ...

C++ Evolution

- C++11 - Cornerstone of Modern C++
  - Move Semantics, **`constexpr`**, lambda expressions, variable templates, memory model and threads, range-for, chrono, ...
- C++20 - HUGE!!!
  - concepts, ranges, coroutines, class NTTPs, **`constexpr`** all the things, **`consteval`**, **`<=>`**, **`format`**, **`counting_semaphore/latch/barrier`**, modules 🤔, ...
- C++26 - AWESOME!!!
  - reflection, contracts, **`std::execution`**, even more **`constexpr`**, relocability, RCU, hazard pointers, SIMD, linalg, ...

C++ Evolution

- C++11 - Cornerstone of Modern C++
  - Move Semantics, **`constexpr`**, lambda expressions, variable templates, memory model and threads, range-for, chrono, ...
- C++20 - HUGE!!!
  - concepts, ranges, coroutines, class NTTPs, **`constexpr`** all the things, **`consteval`**, **`<=>`**, **`format`**, **`counting_semaphore/latch/barrier`**, modules 🤔, ...
- C++26 - AWESOME!!!
  - reflection, contracts, **`std::execution`**, even more **`constexpr`**, relocability, RCU, hazard pointers, SIMD, linalg, ...

Still, there are many features missing that would allow library developers to provide a better experience to the users.

I will...

1. Describe problems that library developers and their users face today

I will...

1. **Describe problems that library developers and their users face today**
   - Improving interfaces
   - Enabling compile-time code debugging
   - Better compile-time error messages
   - Improving safety
   - A need for additional text utilities
   - Improving distribution and consumption of libraries

I will...

1. **Describe problems that library developers and their users face today**
   - Improving interfaces
   - Enabling compile-time code debugging
   - Better compile-time error messages
   - Improving safety
   - A need for additional text utilities
   - Improving distribution and consumption of libraries

The feature selection is subjective and far from being complete...

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features
   - Recently accepted for C++26
     - *"C++26"* annotation next to a paper number in a title
     - most of the papers were not yet accepted when I was submitting this talk

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features
   - Recently accepted for C++26
     - *"C++26"* annotation next to a paper number in a title
     - most of the papers were not yet accepted when I was submitting this talk
   - In the WG21 pipeline that did not get to C++26
     - just a paper number in a title

I will...

1. **Describe problems that library developers and their users face today**
2. **Present how those problems can be addressed with future C++ language and library features**
   - Recently **accepted for C++26**
     - *"C++26"* annotation next to a paper number in a title
     - most of the papers were not yet accepted when I was submitting this talk
   - **In the WG21 pipeline** that did not get to C++26
     - just a paper number in a title
   - **Abandoned papers** that did not have a revision for a longer time
     - *"Abandoned"* annotation next to a paper number in a title
     - *Contributors are welcome to help progress them*

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features
3. NOT discuss big features in details (e.g., reflection, contracts)

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features
3. NOT discuss big features in details (e.g., reflection, contracts)
   - Not enough time
   - Many great talks about them already available

I will...

1. Describe problems that library developers and their users face today
2. Present how those problems can be addressed with future C++ language and library features
3. NOT discuss big features in details (e.g., reflection, contracts)
4. Present concrete examples of how they solve common problems in Modern C++ libraries

Finding a paper

- Get paper
  - https://wg21.link/nXXXX
  - https://wg21.link/pXXXXrX
  - https://wg21.link/pXXXX - latest version

Finding a paper

- Get paper
  - https://wg21.link/nXXXX
  - https://wg21.link/pXXXXrX
  - https://wg21.link/pXXXX - latest version
- Get GitHub tracking issue
  - https://wg21.link/pXXXX/github
  - https://wg21.link/pXXXX/status

Finding a paper

- Get paper
  - https://wg21.link/nXXXX
  - https://wg21.link/pXXXXrX
  - https://wg21.link/pXXXX - latest version
- Get GitHub tracking issue
  - https://wg21.link/pXXXX/github
  - https://wg21.link/pXXXX/status
- Get working draft
  - https://wg21.link/std{11,14,17,20,23}
  - https://wg21.link/std - latest released version

Finding a paper

- Get paper
  - https://wg21.link/nXXXX
  - https://wg21.link/pXXXXrX
  - https://wg21.link/pXXXX - latest version
- Get GitHub tracking issue
  - https://wg21.link/pXXXX/github
  - https://wg21.link/pXXXX/status
- Get working draft
  - https://wg21.link/std{11,14,17,20,23}
  - https://wg21.link/std - latest released version
- Usage info
  - https://wg21.link

IMPROVING INTERFACES

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

```
auto foo = [](auto&& t) { /* ... */ };
```

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

```
auto foo = [](auto&& t) { /* ... */ };
```

```
template<typename T> auto foo(T&& t) { /* ... */ }
```

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

```
auto foo = [](auto&& t) { /* ... */ };
```

```
auto foo(auto&& t) { /* ... */ }
```

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

```
auto foo = [](auto&& t) { /* ... */ };
```

```
auto foo(auto&& t) { /* ... */ }
```

```
template<typename T> class foo { /* ... */ };
```

How do you feel about such an interface?

```
void* foo(void* t) { /* ... */ }
```

```
auto foo = [](auto&& t) { /* ... */ };
```

```
auto foo(auto&& t) { /* ... */ }
```

```
template<typename T> class foo { /* ... */ };
```

Unconstrained template parameters are the **`void*`** of C++

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

- What should I pass as **R** and **Rep**?

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

- What should I pass as **R** and **Rep**?
- What if **Rep** can't be divided with **Rep2**?

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

- What should I pass as **R** and **Rep**?
- What if **Rep** can't be divided with **Rep2**?
- What is returned from a function?

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

- What should I pass as **R** and **Rep**?
- What if **Rep** can't be divided with **Rep2**?
- What is returned from a function?

If I will *manage to crash the compilation* of your code, then **it is your bug**!

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

Can you spot a bug?

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

NOT DEPENDENT

- Trivial Conversions
- Promotions
- Standard Conversions
- User-defined Conversions

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);

  // ...
};
```

NOT DEPENDENT

- Trivial Conversions
- Promotions
- Standard Conversions
- User-defined Conversions

DEPENDENT

- Parameter/Argument adjustments
- Consideration of alternatives (e.g., base classes)
- No additional conversions

Legacy template code

```
template<auto R, typename Rep = double>
class quantity {
public:
  template<auto R2, typename Rep2>
  [[nodiscard]] friend constexpr auto operator/(const quantity& lhs,
                                                const quantity<R2, Rep2>& rhs);
  // ...
};
```

Parameters of binary operators should be symmetrical

```
static_assert(std::is_same_v<decltype(a + b), decltype(b + a)>);
```

Concepts are great!

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs);
  // ...
};
```

Concepts are great!

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs);
  // ...
};
```

Challenge every "naked" **`typename`** and **`auto`** in your code.

Concepts are great!

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs);
  // ...
};
```

Both template parameters are dependent now.

It's only a half of the story 😳

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
  {




  }
  //...
};
```

It's only a half of the story 😳

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
  {
    MP_UNITS_EXPECTS_DEBUG(is_neq_zero(rhs));
    return ::mp_units::quantity{lhs.numerical_value_ref_in(unit) /
                                rhs.numerical_value_ref_in(rhs.unit),
                                R / R2};
  }
  //...
};
```

It's only a half of the story 😳

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
  {
    MP_UNITS_EXPECTS_DEBUG(is_neq_zero(rhs));
    return ::mp_units::quantity{lhs.numerical_value_ref_in(unit) /
                                rhs.numerical_value_ref_in(rhs.unit),
                                R / R2};
  }
  //...
};
```

- *Not visible in function declarations* and automatically generated *API reference* documentation

It's only a half of the story 😟

```cpp
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
  {
    MP_UNITS_EXPECTS_DEBUG(is_neq_zero(rhs));
    return ::mp_units::quantity{lhs.numerical_value_ref_in(unit) /
                                rhs.numerical_value_ref_in(rhs.unit),
                                R / R2};
  }
  //...
};
```

- *Not visible in function declarations* and automatically generated *API reference* documentation
- Often *checked only in Debug* builds

P2900 Contracts for C++ (C++26) 🎉

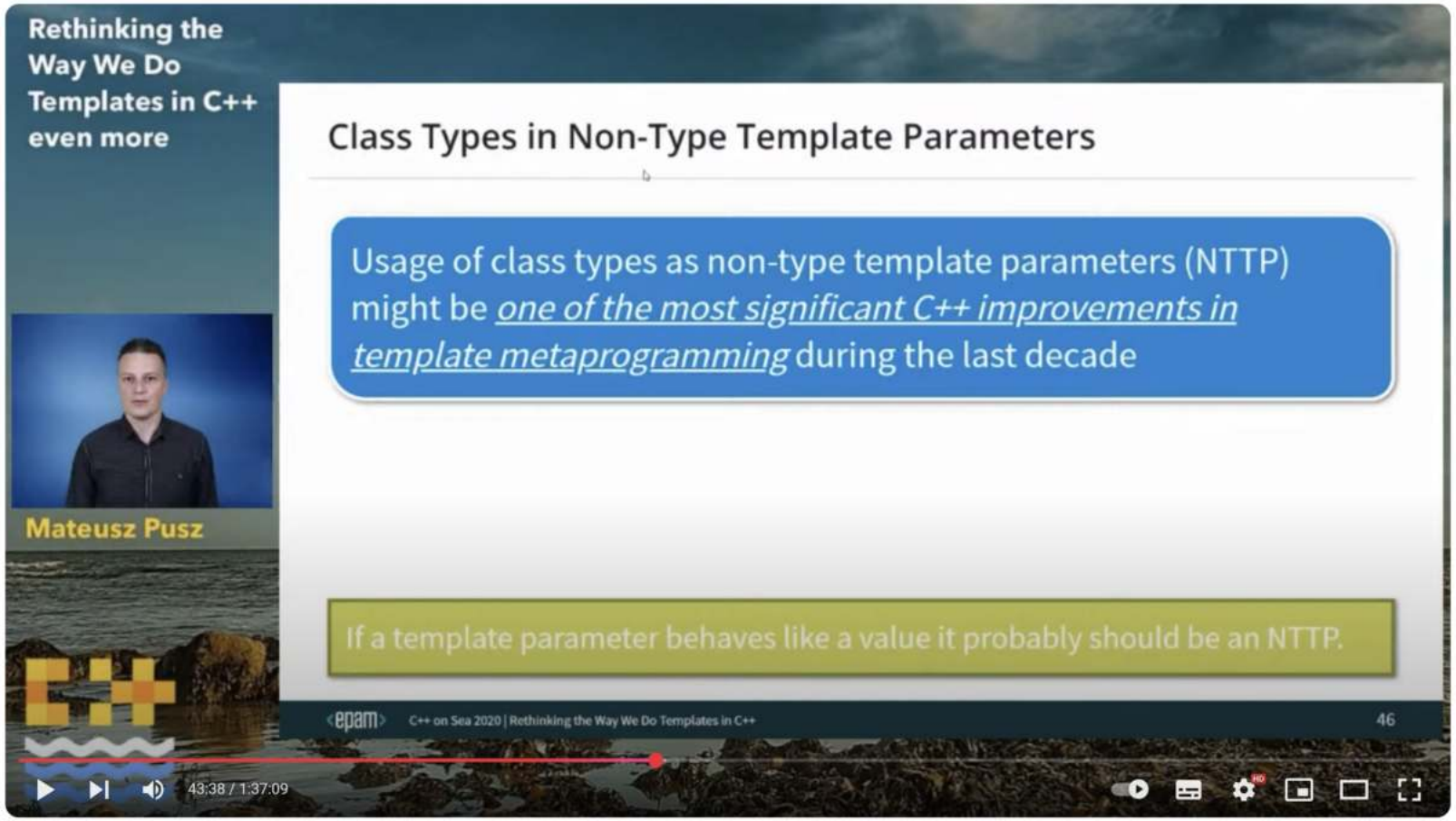
```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
      pre (is_neq_zero(rhs))
  {
    return ::mp_units::quantity{lhs.numerical_value_ref_in(unit) /
                                rhs.numerical_value_ref_in(rhs.unit),
                                R / R2};
  }
  //...
};
```

P2900 Contracts for C++ (C++26) 🎉

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<std::derived_from<quantity> Q, auto R2, typename Rep2>
    requires detail::InvocableQuantities<std::divides<>, quantity, quantity<R2, Rep2>>
  [[nodiscard]] friend constexpr Quantity auto operator/(const Q& lhs,
                                                         const quantity<R2, Rep2>& rhs)
      pre (is_neq_zero(rhs))
  {
    return ::mp_units::quantity{lhs.numerical_value_ref_in(unit) /
                                rhs.numerical_value_ref_in(rhs.unit),
                                R / R2};
  }
  //...
};
```

Being able to express compile-time and runtime contracts in declarations of our interfaces is AWESOME!

Supercharging Non-Type Template Parameters



Rethinking the Way We Do Templates in C++ even more - Mateusz Pusz [C++ on Sea 2020]

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

```
inline constexpr struct length final : quantity_spec<dim_length> {} length;
inline constexpr struct time final   : quantity_spec<dim_time> {} time;
inline constexpr struct speed final  : quantity_spec<length / time> {} speed;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

```
inline constexpr struct length final : quantity_spec<dim_length> {} length;
inline constexpr struct time final   : quantity_spec<dim_time> {} time;
inline constexpr struct speed final  : quantity_spec<length / time> {} speed;
```

```
inline constexpr struct metre final  : named_unit<"m", kind_of<isq::length>> {} metre;
inline constexpr struct second final : named_unit<"s", kind_of<isq::time>> {} second;
inline constexpr struct hour final   : named_unit<"h", mag<60> * minute> {} hour;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

```
inline constexpr struct length final : quantity_spec<dim_length> {} length;
inline constexpr struct time final   : quantity_spec<dim_time> {} time;
inline constexpr struct speed final  : quantity_spec<length / time> {} speed;
```

```
inline constexpr struct metre final  : named_unit<"m", kind_of<isq::length>> {} metre;
inline constexpr struct second final : named_unit<"s", kind_of<isq::time>> {} second;
inline constexpr struct hour final   : named_unit<"h", mag<60> * minute> {} hour;
```

```
template<PrefixableUnit U> struct kilo_ final : prefixed_unit<"k", mag_power<10, 3>, U{}> {};
template<PrefixableUnit auto U> constexpr kilo_<decltype(U)> kilo;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

```
inline constexpr struct length final : quantity_spec<dim_length> {} length;
inline constexpr struct time final   : quantity_spec<dim_time> {} time;
inline constexpr struct speed final  : quantity_spec<length / time> {} speed;
```

```
inline constexpr struct metre final  : named_unit<"m", kind_of<isq::length>> {} metre;
inline constexpr struct second final : named_unit<"s", kind_of<isq::time>> {} second;
inline constexpr struct hour final   : named_unit<"h", mag<60> * minute> {} hour;
```

```
template<PrefixableUnit U> struct kilo_ final : prefixed_unit<"k", mag_power<10, 3>, U{}> {};
template<PrefixableUnit auto U> constexpr kilo_<decltype(U)> kilo;
```

```
quantity<speed[kilo<metre>/hour]> v = 120 * kilo<metre> / (2 * hour);
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_length final : base_dimension<"L"> {} dim_length;
inline constexpr struct dim_time final   : base_dimension<"T"> {} dim_time;
```

```
inline constexpr struct length final : quantity_spec<dim_length> {} length;
inline constexpr struct time final   : quantity_spec<dim_time> {} time;
inline constexpr struct speed final  : quantity_spec<length / time> {} speed;
```

```
inline constexpr struct metre final  : named_unit<"m", kind_of<isq::length>> {} metre;
inline constexpr struct second final : named_unit<"s", kind_of<isq::time>> {} second;
inline constexpr struct hour final   : named_unit<"h", mag<60> * minute> {} hour;
```

```
template<PrefixableUnit U> struct kilo_ final : prefixed_unit<"k", mag_power<10, 3>, U{}> {};
template<PrefixableUnit auto U> constexpr kilo_<decltype(U)> kilo;
```

```
quantity<speed[kilo<metre>/hour]> v = 120 * kilo<metre> / (2 * hour);
```

Non-type template parameters (NTTP) (C++20)

One of the following (optionally cv-qualified) types

- a **structural type**
- a type that *contains a placeholder* (**auto**) type
- a placeholder for a *deduced class type* (CTAD)

Non-type template parameters (NTTP) (C++20)

One of the following (optionally cv-qualified) types

- a **structural type**
- a type that *contains a placeholder* (**auto**) type
- a placeholder for a *deduced class type* (CTAD)

Structural types

- a *scalar* type with a constant destruction
- an *lvalue reference* type
- a *literal class* where **all base classes and non-static data members are public** and non-mutable structural types

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_thermodynamic_temperature final :
           base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
           quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_thermodynamic_temperature final :
           base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
           quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

```
constexpr quantity delta = 273'150 * milli<kelvin>;
constexpr quantity_point point = absolute_zero + delta;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_thermodynamic_temperature final :
           base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
           quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

```
constexpr quantity delta = 273'150 * milli<kelvin>;
constexpr quantity_point point = absolute_zero + delta;
```

```
inline constexpr struct ice_point final : relative_point_origin<point> {} ice_point;

inline constexpr struct degree_Celsius final : named_unit<symbol_text{u8"℃", "'C"}, kelvin, ice_point> {} degree_Celsius;
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_thermodynamic_temperature final :
          base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
          quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

```
constexpr quantity delta = 273'150 * milli<kelvin>;
constexpr quantity_point point = absolute_zero + delta;
```

```
inline constexpr struct ice_point final : relative_point_origin<point> {} ice_point;

inline constexpr struct degree_Celsius final : named_unit<symbol_text{u8"℃", "'C"}, kelvin, ice_point> {} degree_Celsius;
```

```
quantity_point room_temp(21 * degree_Celsius);
```

Supercharging Non-Type Template Parameters

```
inline constexpr struct dim_thermodynamic_temperature final :
           base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
           quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

```
constexpr quantity delta = 273'150 * milli<kelvin>;
constexpr quantity_point point = absolute_zero + delta;
```

```
inline constexpr struct ice_point final : relative_point_origin<point> {} ice_point;

inline constexpr struct degree_Celsius final : named_unit<symbol_text{u8"℃", "'C"}, kelvin, ice_point> {} degree_Celsius;
```

```
quantity_point room_temp(21 * degree_Celsius);
```

Supercharging Non-Type Template Parameters

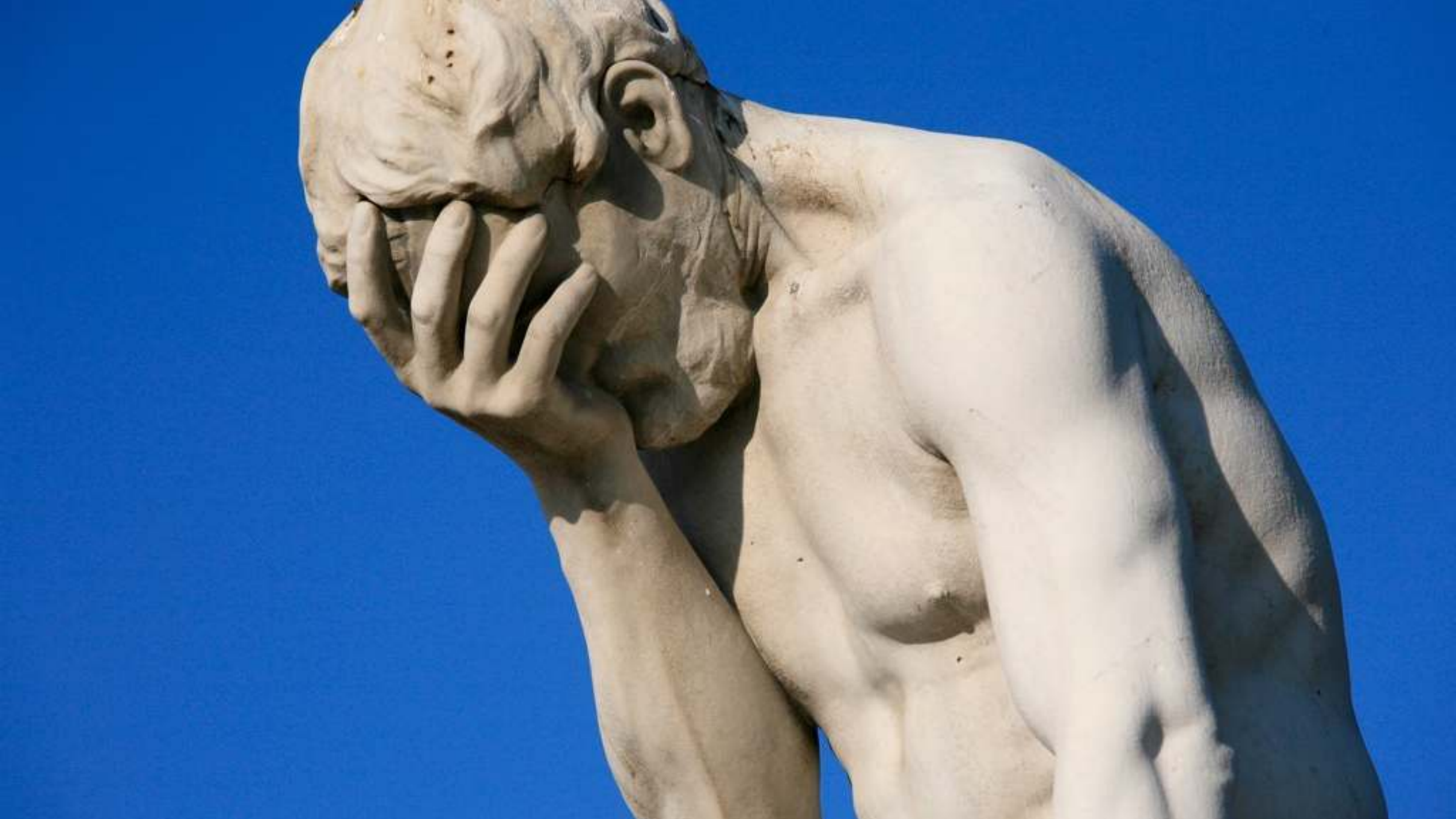
```
inline constexpr struct dim_thermodynamic_temperature final :
           base_dimension<symbol_text{u8"Θ", "O"}> {} dim_thermodynamic_temperature;

inline constexpr struct thermodynamic_temperature final :
           quantity_spec<dim_thermodynamic_temperature> {} thermodynamic_temperature;

inline constexpr struct absolute_zero final : absolute_point_origin<isq::thermodynamic_temperature> {} absolute_zero;

inline constexpr struct kelvin final : named_unit<"K", kind_of<isq::thermodynamic_temperature>, absolute_zero> {} kelvin;
```

```
constexpr quantity delta = 273'150 * milli<kelvin>;
constexpr quantity_point point = absolute_zero + delta;
```

```
inline constexpr struct ice_point final : relative_point_origin<point> {} ice_point;

inline constexpr struct degree_Celsius final : named_unit<symbol_text{u8"℃", "'C"}, kelvin, ice_point> {} degree_Celsius;
```

```
quantity_point room_temp(21 * degree_Celsius);
```

```
room_temp.quantity_from_origin_.numerical_value_ *= 2;  // !!!!!
```



Structural types

```
19      -   auto hours = units::quantity_cast<units::isq::si::hour>(minutes);
20      -   if(hours>0*h) {
21      -       if (hours < 10*h)
        13  +   auto hours = minutes.force_in(u_h);
        14  +   if(hours>0*u_h) {
        15  +       if (hours < 10*u_h)
22      16              result << "0";
23      -       result << std::to_string(hours.number());
        17  +       result << std::to_string(hours.numerical_value_);
```

Mateusz Pusz @mateusz.pusz · Oct 16, 2023

Please don't use member data directly. This is the implementation detail and should not be used directly.

Mateusz Pusz @mateusz.pusz · Oct 16, 2023

It seems I have to obfuscate the name more to discourage people from using it.

[redacted] Nov 09, 2023 Owner

Don't know if obfuscation would be effective. Maybe a short doxgen comment the discourages usage? Such a comment would pop up in must IDEs...

Structural types

```
template<Reference auto R,
         RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  Rep numerical_value_is_an_implementation_detail_;  ///< needs to be public for a structural type
  // ...
};
```

```
template<Reference auto R,
         PointOriginFor<get_quantity_spec(R)> auto PO = default_point_origin(R),
         RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity_point {
public:
  quantity_type quantity_from_origin_is_an_implementation_detail_;  ///< needs to be public for a structural type
  // ...
};
```

Impact of NTTP restrictions

- Authors of C++ classes that want them to be usable as NTTTPs have to sacrifice encapsulation

Impact of NTTP restrictions

- Authors of C++ classes that want them to be usable as NTTTPs have to sacrifice encapsulation
- Many types from the C++ Standard Library can't be passed as NTTPs today
  - e.g., **`std::optional`**, **`std::variant`**, **`std::tuple`**, **`std::span`**, **`std::vector`**, ...

Impact of NTTP restrictions

- Authors of C++ classes that want them to be usable as NTTTPs have to sacrifice encapsulation
- Many types from the C++ Standard Library can't be passed as NTTPs today
  - e.g., **`std::optional`**, **`std::variant`**, **`std::tuple`**, **`std::span`**, **`std::vector`**, ...
- NTTP-compatible wrapper types (e.g., **`quantity<R, Rep>`**) work as long as Rep is also a structural type
  - e.g., **`quantity<si::metre, uncertainty<double>>`** will not work if **`uncertainty<double>`** has private members

Impact of NTTP restrictions

- Authors of C++ classes that want them to be usable as NTTTPs have to sacrifice encapsulation
- Many types from the C++ Standard Library can't be passed as NTTPs today
  - e.g., **`std::optional`**, **`std::variant`**, **`std::tuple`**, **`std::span`**, **`std::vector`**, ...
- NTTP-compatible wrapper types (e.g., **`quantity<R, Rep>`**) work as long as Rep is also a structural type
  - e.g., **`quantity<si::metre, uncertainty<double>>`** will not work if **`uncertainty<double>`** has private members
- P3094: **`std::basic_fixed_string`** proposal not in C++26 because of hopes to get **`std::inplace_string`**
  - as of today **`std::inplace_string`** can't work properly as a structural type

P3380 Extending support for class types as non-type template parameters

```
template<Reference auto R,
         RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
  Rep numerical_value_;
  auto to_meta_representation(std::meta::serializer&) const = default;
  static auto from_meta_representation(std::meta::deserializer&) = default;
public:
// ...
};
```

P3380 Extending support for class types as non-type template parameters

```
template<Reference auto R,
         RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
  Rep numerical_value_;
  auto to_meta_representation(std::meta::serializer&) const = default;
  static auto from_meta_representation(std::meta::deserializer&) = default;
public:
// ...
};
```

```
template<Reference auto R,
         PointOriginFor<get_quantity_spec(R)> auto PO = default_point_origin(R),
         RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity_point {
  quantity_type quantity_from_origin_;
  auto to_meta_representation(std::meta::serializer&) const = default;
  static auto from_meta_representation(std::meta::deserializer&) = default;
public:
  // ...
};
```

What is wrong?

CONCEPTS

```
template<Quantity Q>
void foo(Q&& q);
```

What is wrong?

CONCEPTS

```
template<Quantity Q>
void foo(Q&& q);
```

`Q` from the forwarding reference may include **`&`**, **`const`**, and **`volatile`**.

Lack of template parameters composition

CONCEPTS

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

Lack of template parameters composition

CONCEPTS

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

```
template<std::ranges::range R>
  requires awaitable<std::ranges::range_value_t<R>>
awaitable auto when_all(const R& awaitables);
```

Lack of template parameters composition

CONCEPTS

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

```
template<std::ranges::range R>
  requires awaitable<std::ranges::range_value_t<R>>
awaitable auto when_all(const R& awaitables);
```

Often constraints terse or shorthand notations are not enough and additional **`requires`** clauses are needed to constrain template parameters.

Lack of template parameters composition

VARIABLE TEMPLATES

```
template<typename T>
inline constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<>
inline constexpr bool treat_as_floating_point<my_float> = true;
```

Lack of template parameters composition

VARIABLE TEMPLATES

```
template<typename T>
inline constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<>
inline constexpr bool treat_as_floating_point<my_float> = true;
```

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

Lack of template parameters composition

VARIABLE TEMPLATES

```
template<typename T>
inline constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<>
inline constexpr bool treat_as_floating_point<my_float> = true;
```

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

Only class template templates allowed in the C++ language.

Lack of template parameters composition

VARIABLE TEMPLATES

```
template<typename T>
inline constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<>
inline constexpr bool treat_as_floating_point<my_float> = true;
```

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

```
template<typename T>
struct treat_as_floating_point_t : std::bool_constant<treat_as_floating_point<T>> {};

static_assert(all_of<treat_as_floating_point_t, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, int, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, double, float, long, double>);
```

Lack of template parameters composition

VARIABLE TEMPLATES

```
template<typename T>
inline constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<>
inline constexpr bool treat_as_floating_point<my_float> = true;
```

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

```
template<typename T>
struct treat_as_floating_point_t : std::bool_constant<treat_as_floating_point<T>> {};

static_assert(all_of<treat_as_floating_point_t, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, int, double, float, long double>);
```

More to type, extraneous abstractions, slower to compile.

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

AFTER

```
template<typename T, template<typename> concept C>
concept decays_to = C<std::decay_t<T>>;
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<typename Q>
  requires Quantity<std::remove_cvref_t<Q>>
void foo(Q&& q);
```

AFTER

```
template<typename T, template<typename> concept C>
concept decays_to = C<std::decay_t<T>>;
```

```
void foo(decays_to<Quantity> auto&& q);
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<std::ranges::range R>
  requires awaitable<std::ranges::range_value_t<R>>
awaitable auto when_all(const R& awaitables);
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<std::ranges::range R>
  requires awaitable<std::ranges::range_value_t<R>>
awaitable auto when_all(const R& awaitables);
```

AFTER

```
template<typename T, template<typename> concept C>
concept range_of = std::ranges::range<T> && C<std::ranges::range_value_t<T>>;
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

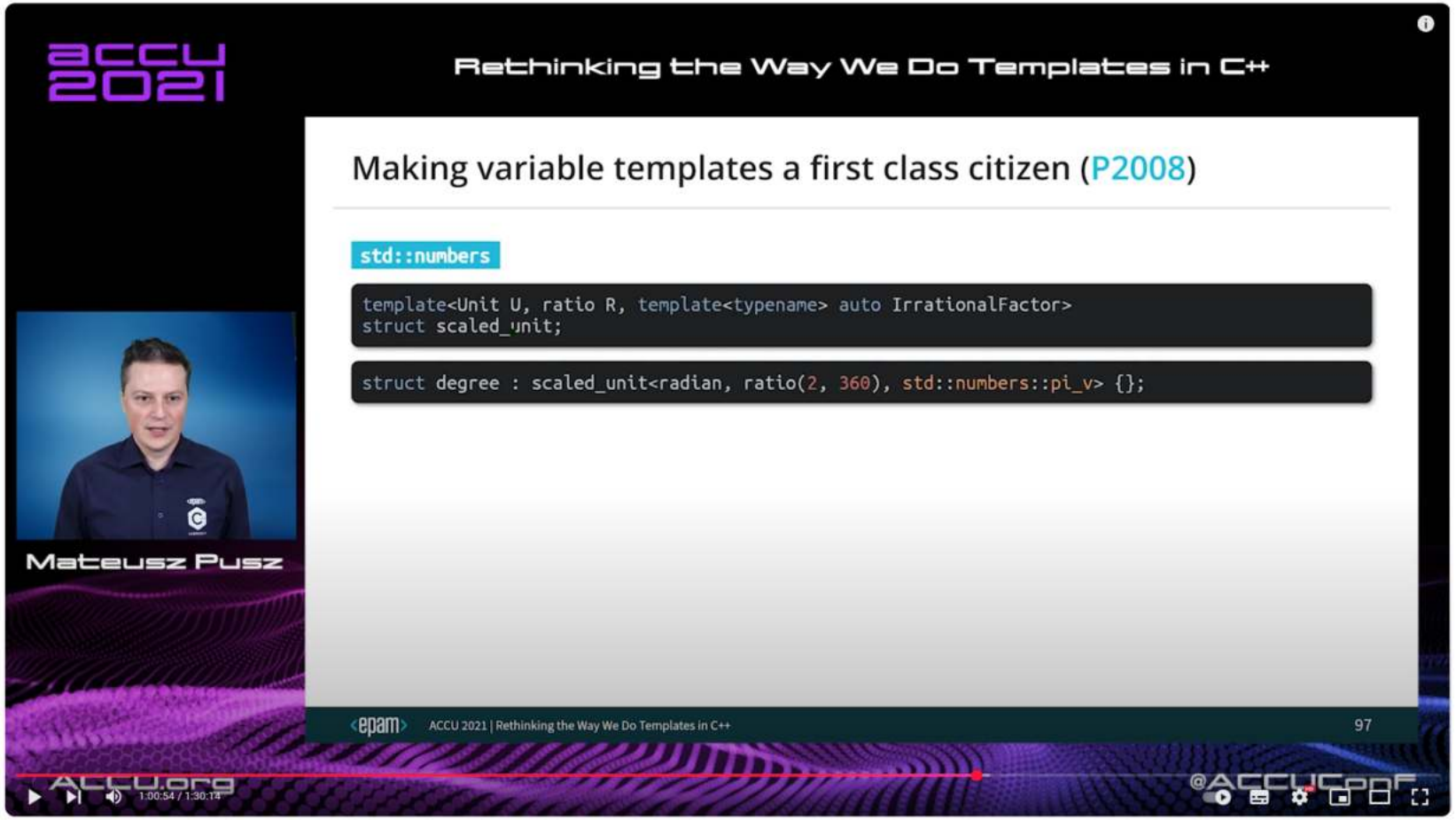
```
template<std::ranges::range R>
  requires awaitable<std::ranges::range_value_t<R>>
awaitable auto when_all(const R& awaitables);
```

AFTER

```
template<typename T, template<typename> concept C>
concept range_of = std::ranges::range<T> && C<std::ranges::range_value_t<T>>;
```

```
awaitable auto when_all(const range_of<awaitable> auto& awaitables);
```

Variable template template



Rethinking the Way We Do Templates in C++ - Mateusz Pusz [ACCU 2021]

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

```
template<typename T>
struct treat_as_floating_point_t : std::bool_constant<treat_as_floating_point<T>> {};

static_assert(all_of<treat_as_floating_point_t, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, int, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, double, float, long, double>);
```

P2841 Concept and variable-template template-parameters (C++26)

BEFORE

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

```
template<typename T>
struct treat_as_floating_point_t : std::bool_constant<treat_as_floating_point<T>> {};

static_assert(all_of<treat_as_floating_point_t, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, int, double, float, long double>);
static_assert(!all_of<treat_as_floating_point_t, double, float, long, double>);
```

AFTER

```
template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>);
```

```
static_assert(all_of<treat_as_floating_point, double, float, long double>);
static_assert(!all_of<treat_as_floating_point, int, double, float, long double>);
static_assert(!all_of<treat_as_floating_point, double, float, long, double>);
```

Template Parameter Kinds

BEFORE

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

AFTER

```
template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>);
```

Template Parameter Kinds

BEFORE

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>::value);
```

AFTER

```
template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of = (... && Pred<Ts>);
```

Why do I have to choose? Can't I have both?

Template Parameter Kinds

Templates are parameterized by *one* or *more* template parameters

Template Parameter Kinds

Templates are parameterized by *one* or *more* template parameters

- **type** template parameters

```
template<typename Ptr> class smart_ptr { /* ... */ };
smart_ptr<int> ptr;
```

Template Parameter Kinds

Templates are parameterized by *one* or *more* template parameters

- **type** template parameters

```
template<typename Ptr> class smart_ptr { /* ... */ };
smart_ptr<int> ptr;
```

- **non-type** template parameters

```
template<typename T, size_t N> class array { /* ... */ };
array<int, 5> a;
```

Template Parameter Kinds

Templates are parameterized by *one* or *more* template parameters

- **type** template parameters

```
template<typename Ptr> class smart_ptr { /* ... */ };
smart_ptr<int> ptr;
```

- **non-type** template parameters

```
template<typename T, size_t N> class array { /* ... */ };
array<int, 5> a;
```

- **template** template parameters

```
template<typename T> class my_deleter {};
template<typename T, template<typename> typename Policy> class handle { /* ... */ };
handle<FILE, my_deleter> h;
```

Template Parameter Kinds

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of_t_t = (... && Pred<Ts>::value);

template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of_v_t = (... && Pred<Ts>);

template<template<typename> concept Pred, typename... Ts>
constexpr bool all_of_c_t = (... && Pred<Ts>);
```

Template Parameter Kinds

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of_t_t = (... && Pred<Ts>::value);

template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of_v_t = (... && Pred<Ts>);

template<template<typename> concept Pred, typename... Ts>
constexpr bool all_of_c_t = (... && Pred<Ts>);
```

```
template<template<auto> typename Pred, auto... Vs>
constexpr bool all_of_t_v = (... && Pred<Vs>::value);

template<template<auto> auto Pred, auto... Vs>
constexpr bool all_of_v_v = (... && Pred<Vs>);

template<template<auto> concept Pred, auto... Vs>
constexpr bool all_of_c_v = (... && Pred<Vs>);
```

Template Parameter Kinds

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of_t_t = (... && Pred<Ts>::value);

template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of_v_t = (... && Pred<Ts>);

template<template<typename> concept Pred, typename... Ts>
constexpr bool all_of_c_t = (... && Pred<Ts>);
```

```
template<template<auto> typename Pred, auto... Vs>
constexpr bool all_of_t_v = (... && Pred<Vs>::value);

template<template<auto> auto Pred, auto... Vs>
constexpr bool all_of_v_v = (... && Pred<Vs>);

template<template<auto> concept Pred, auto... Vs>
constexpr bool all_of_c_v = (... && Pred<Vs>);
```

```
template<template<typename> typename Pred, auto... Vs>
constexpr bool all_of_t_tv = (... && Pred<decltype(Vs)>::value);

template<template<typename> auto Pred, auto... Vs>
constexpr bool all_of_v_tv = (... && Pred<decltype(Vs)>);

template<template<typename> concept Pred, auto... Vs>
constexpr bool all_of_c_tv = (... && Pred<decltype(Vs)>);
```

Template Parameter Kinds

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of_t_t = (... && Pred<Ts>::value);

template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of_v_t = (... && Pred<Ts>);

template<template<typename> concept Pred, typename... Ts>
constexpr bool all_of_c_t = (... && Pred<Ts>);
```

```
template<template<auto> typename Pred, auto... Vs>
constexpr bool all_of_t_v = (... && Pred<Vs>::value);

template<template<auto> auto Pred, auto... Vs>
constexpr bool all_of_v_v = (... && Pred<Vs>);

template<template<auto> concept Pred, auto... Vs>
constexpr bool all_of_c_v = (... && Pred<Vs>);
```

```
template<template<typename> typename Pred, auto... Vs>
constexpr bool all_of_t_tv = (... && Pred<decltype(Vs)>::value);

template<template<typename> auto Pred, auto... Vs>
constexpr bool all_of_v_tv = (... && Pred<decltype(Vs)>);

template<template<typename> concept Pred, auto... Vs>
constexpr bool all_of_c_tv = (... && Pred<decltype(Vs)>);
```

- Some user...

```
static_assert(all_of_oops<std::integral, int, 3, long, 7U>);
```

Template Parameter Kinds

```
template<template<typename> typename Pred, typename... Ts>
constexpr bool all_of_t_t = (... && Pred<Ts>::value);

template<template<typename> auto Pred, typename... Ts>
constexpr bool all_of_v_t = (... && Pred<Ts>);

template<template<typename> concept Pred, typename... Ts>
constexpr bool all_of_c_t = (... && Pred<Ts>);
```

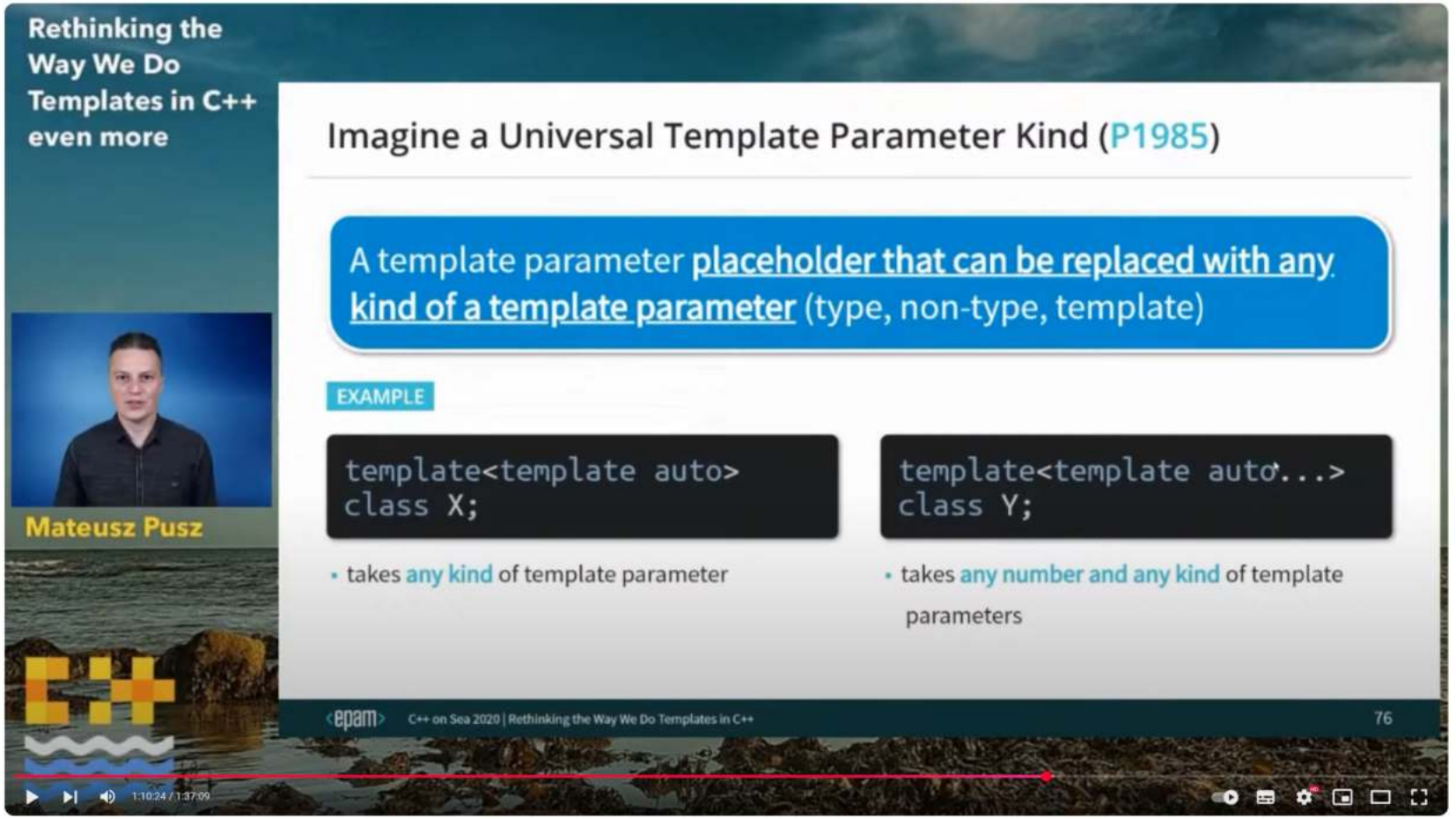
```
template<template<auto> typename Pred, auto... Vs>
constexpr bool all_of_t_v = (... && Pred<Vs>::value);

template<template<auto> auto Pred, auto... Vs>
constexpr bool all_of_v_v = (... && Pred<Vs>);

template<template<auto> concept Pred, auto... Vs>
constexpr bool all_of_c_v = (... && Pred<Vs>);
```

```
template<template<typename> typename Pred, auto... Vs>
constexpr bool all_of_t_tv = (... && Pred<decltype(Vs)>::value);

template<template<typename> auto Pred, auto... Vs>
constexpr bool all_of_v_tv = (... && Pred<decltype(Vs)>);

template<template<typename> concept Pred, auto... Vs>
constexpr bool all_of_c_tv = (... && Pred<decltype(Vs)>);
```

- Some user...

```
static_assert(all_of_oops<std::integral, int, 3, long, 7U>);
```

Universal Template Parameter Kind



Rethinking the Way We Do Templates in C++ even more - Mateusz Pusz [C++ on Sea 2020]

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check = check_impl<Pred, Arg>;



template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check = check_impl<Pred, Arg>;

template<template<typename> universal template Pred, auto Arg>
constexpr bool check<Pred, Arg> = check_impl<Pred, decltype(Arg)>;

template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check_impl = Pred<Arg>;



template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check = check_impl<Pred, Arg>;

template<template<typename> universal template Pred, auto Arg>
constexpr bool check<Pred, Arg> = check_impl<Pred, decltype(Arg)>;

template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check_impl = Pred<Arg>;

template<template<universal template> typename Pred, universal template Arg>
constexpr bool check_impl<Pred, Arg> = Pred<Arg>::value;

template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check = check_impl<Pred, Arg>;

template<template<typename> universal template Pred, auto Arg>
constexpr bool check<Pred, Arg> = check_impl<Pred, decltype(Arg)>;

template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

P2989 A Simple Approach to Universal Template Parameters

```
template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check_impl = Pred<Arg>;

template<template<universal template> typename Pred, universal template Arg>
constexpr bool check_impl<Pred, Arg> = Pred<Arg>::value;

template<template<universal template> universal template Pred, universal template Arg>
constexpr bool check = check_impl<Pred, Arg>;

template<template<typename> universal template Pred, auto Arg>
constexpr bool check<Pred, Arg> = check_impl<Pred, decltype(Arg)>;

template<template<universal template> universal template Pred, universal template... Args>
constexpr bool all_of = (... && check(Pred, Args));
```

```
static_assert(all_of<std::is_scalar, double, float, long double>);            // class template (type trait) taking types
static_assert(all_of<treat_as_floating_point, double, float, long double>);   // variable template taking types
static_assert(all_of<std::regular, double, std::pair<int, long>>);            // concept taking types
static_assert(all_of<non_negative, 1, 3.14, 2U>);                             // concept taking and testing values
static_assert(all_of<std::integral, int, 3, long, 7U>);                       // concept taking anything
static_assert(all_of<Unit, second, metre, metre / second>);                   // concept taking values and testing types
static_assert(all_of<Quantity, 42 * m, decltype(isq::height(42 * m))>);       // concept taking anything and testing types
```

specialization_of

```
template<typename T, template<universal template...> typename Templ>
constexpr bool is_specialization_of = false;
```

specialization_of

```cpp
template<typename T, template<universal template...> typename Templ>
constexpr bool is_specialization_of = false;

template<template<universal template...> typename Templ, universal template... Params>
constexpr bool is_specialization_of<Templ<Params...>, Templ> = true;
```

specialization_of

```
template<typename T, template<universal template...> typename Templ>
constexpr bool is_specialization_of = false;

template<template<universal template...> typename Templ, universal template... Params>
constexpr bool is_specialization_of<Templ<Params...>, Templ> = true;

template<typename T, template<universal template...> typename Templ>
concept bool specialization_of = is_specialization_of<T, Templ>;
```

specialization_of

```
template<typename T, template<universal template...> typename Templ>
constexpr bool is_specialization_of = false;

template<template<universal template...> typename Templ, universal template... Params>
constexpr bool is_specialization_of<Templ<Params...>, Templ> = true;

template<typename T, template<universal template...> typename Templ>
concept bool specialization_of = is_specialization_of<T, Templ>;
```

```
static_assert(specialization_of<some_range, std::vector>);
static_assert(specialization_of<my_array, std::array>);
```

derived_from_specialization_of

```
template<typename T, template<universal template...> typename Templ>
concept bool derived_from_specialization_of = is_derived_from_specialization_of<T, Templ>;
```

derived_from_specialization_of

```
template<typename T, template<universal template...> typename Templ>
constexpr bool is_derived_from_specialization_of =
      requires(T* t) { to_specialization_of<Templ>(t); };

template<typename T, template<universal template...> typename Templ>
concept bool derived_from_specialization_of = is_derived_from_specialization_of<T, Templ>;
```

derived_from_specialization_of

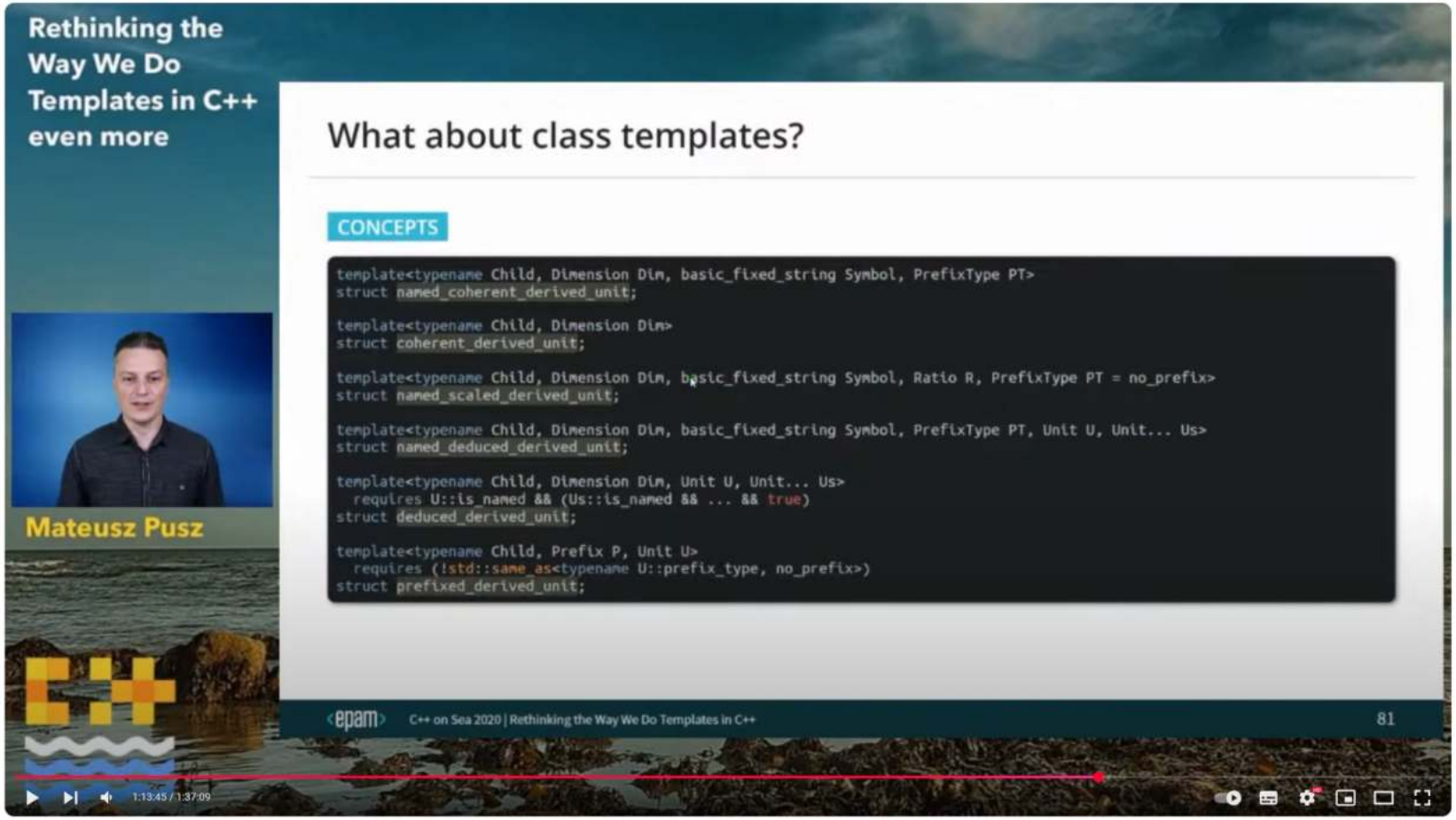
```
template<template<universal template...> typename Templ, universal template... Params>
void to_specialization_of(const volatile Templ<Params...>*);

template<typename T, template<universal template...> typename Templ>
constexpr bool is_derived_from_specialization_of =
      requires(T* t) { to_specialization_of<Templ>(t); };

template<typename T, template<universal template...> typename Templ>
concept bool derived_from_specialization_of = is_derived_from_specialization_of<T, Templ>;
```

derived_from_specialization_of

```
template<template<universal template...> typename Templ, universal template... Params>
void to_specialization_of(const volatile Templ<Params...>*);

template<typename T, template<universal template...> typename Templ>
constexpr bool is_derived_from_specialization_of =
      requires(T* t) { to_specialization_of<Templ>(t); };

template<typename T, template<universal template...> typename Templ>
concept bool derived_from_specialization_of = is_derived_from_specialization_of<T, Templ>;
```

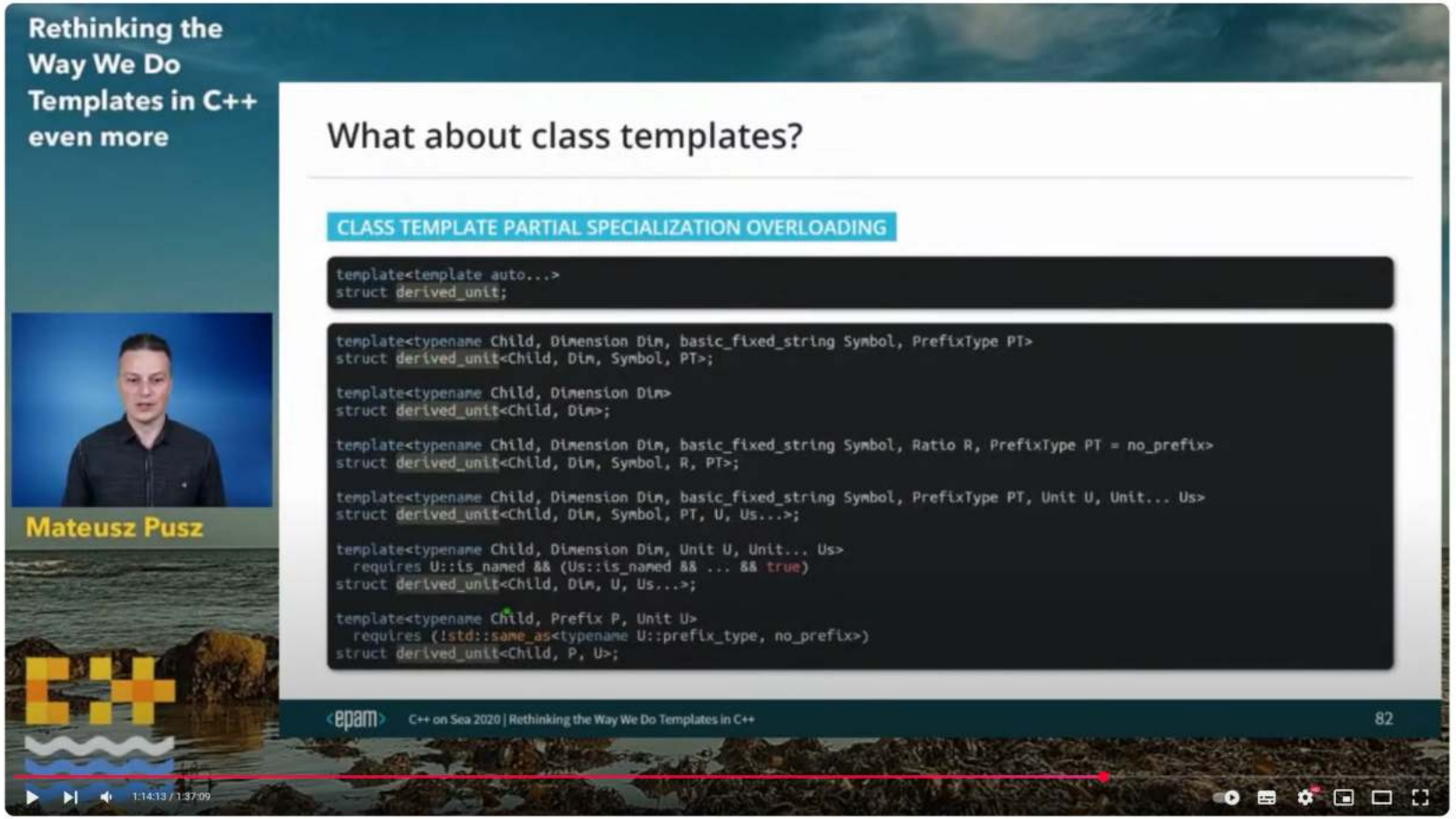
```
static_assert(derived_from_specialization_of<type, quantity>);
```

Derived unit without UTP



Rethinking the Way We Do Templates in C++ even more - Mateusz Pusz [C++ on Sea 2020]

Derived unit with UTP



Rethinking the Way We Do Templates in C++ even more - Mateusz Pusz [C++ on Sea 2020]

P2989 A Simple Approach to Universal Template Parameters

Universal Template Parameters are the cornerstone to building truly generic interfaces.

Variable templates

Customization points based on variable templates are often **faster to compile** and **less to type** than class templates, so their usage is highly recommended.

Variable templates

Customization points based on variable templates are often **faster to compile** and **less to type** than class templates, so their usage is highly recommended.

```
template<typename T>
constexpr bool treat_as_floating_point = std::is_floating_point_v<T>;

template<typename T>
constexpr bool treat_as_floating_point<Safe<T>> = treat_as_floating_point<T>;
```

Variable templates traits

```cpp
template<std::intmax_t Value>
  requires(0 <= Value) && (Value < 10)
constexpr fixed_u8string superscript_number = ???;

template<> constexpr fixed_u8string superscript_number<0> = u8"⁰";
template<> constexpr fixed_u8string superscript_number<1> = u8"¹";
template<> constexpr fixed_u8string superscript_number<2> = u8"²";
template<> constexpr fixed_u8string superscript_number<3> = u8"³";
template<> constexpr fixed_u8string superscript_number<4> = u8"⁴";
template<> constexpr fixed_u8string superscript_number<5> = u8"⁵";
template<> constexpr fixed_u8string superscript_number<6> = u8"⁶";
template<> constexpr fixed_u8string superscript_number<7> = u8"⁷";
template<> constexpr fixed_u8string superscript_number<8> = u8"⁸";
template<> constexpr fixed_u8string superscript_number<9> = u8"⁹";
```

Variable templates traits

```
template<std::intmax_t Value>
  requires(0 <= Value) && (Value < 10)
constexpr fixed_u8string superscript_number = ???;

template<> constexpr fixed_u8string superscript_number<0> = u8"⁰";
template<> constexpr fixed_u8string superscript_number<1> = u8"¹";
template<> constexpr fixed_u8string superscript_number<2> = u8"²";
template<> constexpr fixed_u8string superscript_number<3> = u8"³";
template<> constexpr fixed_u8string superscript_number<4> = u8"⁴";
template<> constexpr fixed_u8string superscript_number<5> = u8"⁵";
template<> constexpr fixed_u8string superscript_number<6> = u8"⁶";
template<> constexpr fixed_u8string superscript_number<7> = u8"⁷";
template<> constexpr fixed_u8string superscript_number<8> = u8"⁸";
template<> constexpr fixed_u8string superscript_number<9> = u8"⁹";
```

However, they are not the best fit if there is no good default value.

P2041 `template = delete` (Abandoned)

```
template<std::intmax_t Value>
  requires(0 <= Value) && (Value < 10)
constexpr fixed_u8string superscript_number = delete;

template<> constexpr fixed_u8string superscript_number<0> = u8"⁰";
template<> constexpr fixed_u8string superscript_number<1> = u8"¹";
template<> constexpr fixed_u8string superscript_number<2> = u8"²";
template<> constexpr fixed_u8string superscript_number<3> = u8"³";
template<> constexpr fixed_u8string superscript_number<4> = u8"⁴";
template<> constexpr fixed_u8string superscript_number<5> = u8"⁵";
template<> constexpr fixed_u8string superscript_number<6> = u8"⁶";
template<> constexpr fixed_u8string superscript_number<7> = u8"⁷";
template<> constexpr fixed_u8string superscript_number<8> = u8"⁸";
template<> constexpr fixed_u8string superscript_number<9> = u8"⁹";
```

Spherical distance

```
template<typename T = double>
using latitude = mp_units::quantity_point<mp_units::si::degree, equator, bounded<T, -90, 90>>;

template<typename T = double>
using longitude = mp_units::quantity_point<mp_units::si::degree, prime_meridian, bounded<T, -180, 180>>;

template<typename T>
struct position {
  latitude<T> lat;
  longitude<T> lon;
};
```

Spherical distance

```
template<typename T = double>
using latitude = mp_units::quantity_point<mp_units::si::degree, equator, bounded<T, -90, 90>>;

template<typename T = double>
using longitude = mp_units::quantity_point<mp_units::si::degree, prime_meridian, bounded<T, -180, 180>>;

template<typename T>
struct position {
  latitude<T> lat;
  longitude<T> lon;
};
```

```
template<typename T>
distance spherical_distance(position<T> from, position<T> to)
{
  using namespace mp_units;
  constexpr quantity earth_radius = 6'371 * isq::radius[si::kilo<si::metre>];
  const quantity from_lat = from.lat.quantity_from_zero();
  const quantity from_lon = from.lon.quantity_from_zero();
  const quantity to_lat = to.lat.quantity_from_zero();
  const quantity to_lon = to.lon.quantity_from_zero();

  using si::sin, si::cos, si::asin, si::acos;
  const quantity central_angle = acos(sin(from_lat) * sin(to_lat) + cos(from_lat) * cos(to_lat) * cos(to_lon - from_lon));
  return quantity_cast<isq::distance>((earth_radius * central_angle).in(earth_radius.unit));
}
```

Spherical distance

```
template<typename T = double>
using latitude = mp_units::quantity_point<mp_units::si::degree, equator, bounded<T, -90, 90>>;

template<typename T = double>
using longitude = mp_units::quantity_point<mp_units::si::degree, prime_meridian, bounded<T, -180, 180>>;

template<typename T>
struct position {
  latitude<T> lat;
  longitude<T> lon;
};
```

```
template<typename T>
distance spherical_distance(position<T> from, position<T> to)
{
  using namespace mp_units;
  constexpr quantity earth_radius = 6'371 * isq::radius[si::kilo<si::metre>];
  const quantity from_lat = from.lat.quantity_from_zero();
  const quantity from_lon = from.lon.quantity_from_zero();
  const quantity to_lat = to.lat.quantity_from_zero();
  const quantity to_lon = to.lon.quantity_from_zero();

  using si::sin, si::cos, si::asin, si::acos;
  const quantity central_angle = acos(sin(from_lat) * sin(to_lat) + cos(from_lat) * cos(to_lat) * cos(to_lon - from_lon));
  return quantity_cast<isq::distance>((earth_radius * central_angle).in(earth_radius.unit));
}
```

[basic.lookup.argdep]

- For each argument type **T** in the function call, there is a set of zero or more associated namespaces and a set of zero or more associated entities (other than namespaces) to be considered

[basic.lookup.argdep]

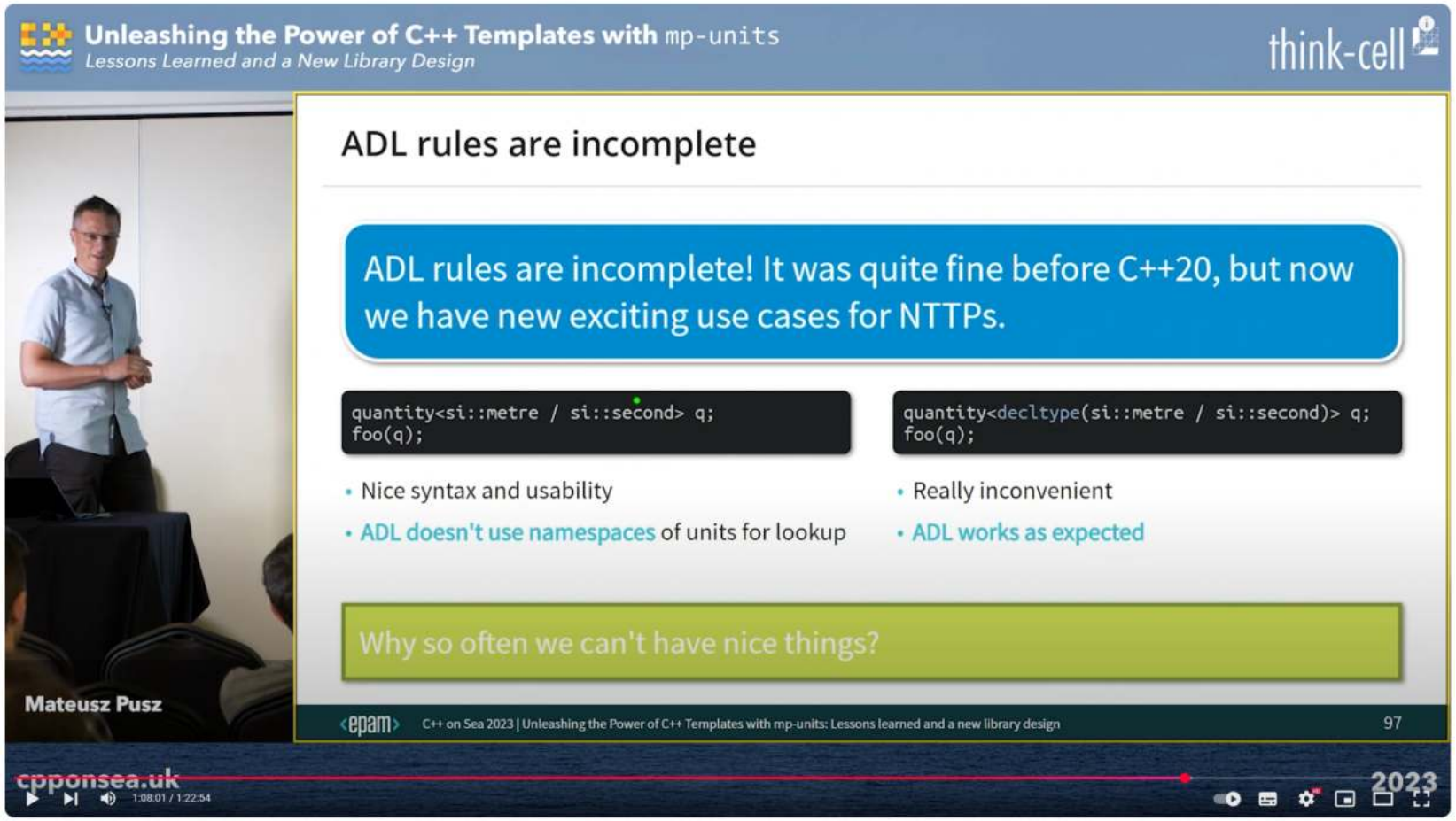
- For each argument type **T** in the function call, there is a **set of zero or more associated namespaces** and a set of zero or more associated entities (other than namespaces) **to be considered**
- The sets of namespaces and entities are **determined in the following way**
  - if **T** is a **class template specialization**, its associated namespaces and entities also include
    - the *namespaces* and entities *associated with the types of the template arguments* provided for **template type parameters** (excluding template template parameters)
    - the templates used as template template arguments
    - the namespaces of which any template template arguments are members
    - the classes of which any member templates used as template template arguments are members

[basic.lookup.argdep]

- For each argument type **T** in the function call, there is a **set of zero or more associated namespaces** and a set of zero or more associated entities (other than namespaces) **to be considered**
- The sets of namespaces and entities are **determined in the following way**
  - if **T** is a **class template specialization**, its associated namespaces and entities also include
    - the *namespaces* and entities *associated with the types of the template arguments* provided for **template type parameters** (excluding template template parameters)
    - the templates used as template template arguments
    - the namespaces of which any template template arguments are members
    - the classes of which any member templates used as template template arguments are members

[Note 1 : Non-type template arguments do not contribute to the set of associated namespaces. —end note]

ADL vs NTTPs



The Power of C++ Templates With mp-units: Lessons Learned & a New Library Design - Mateusz Pusz 2023

ADL workarounds for `std::constant_wrapper` (P2781)

ADL workarounds for `std::constant_wrapper` (P2781)

§ 13 Design

§ 13.1 Add `constant_wrapper`

```
namespace std {
  template<class T>
    struct cw-fixed-value;                                          // exposition only

  template<cw-fixed-value X, class = typename decltype(cw-fixed-value(X))::type>
    struct constant_wrapper;
```

ADL workarounds for `std::constant_wrapper` (P2781)

§ 13 Design

§ 13.1 Add `constant_wrapper`

```
namespace std {
  template<class T>
    struct cw-fixed-value;                                          // exposition only

  template<cw-fixed-value X, class = typename decltype(cw-fixed-value(X))::type>
    struct constant_wrapper;
```

```
auto f = std::cw<"foo">;
std::cout << f << "\n";
```

The stream insertion breaks without the unnamed parameter, which is `char const [4]`, and which in turn pulls the proper `operator<<` into consideration during ADL. Note that this ADL support is imperfect. An earlier version of the paper showed using `std::cw<strlit("foo")>` in a stream insertion operation, where `strlit` is a strucutral type that contains the bytes that comprise `"foo"`:

P2822 Providing user control of associated entities of class types

FIXING ADL FOR NTTPS

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity namespace(decltype(R));
```

P2822 Providing user control of associated entities of class types

FIXING ADL FOR NTTPS

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity namespace(decltype(R));
```

IMPROVING COMPILE-TIMES

```
template<typename Source,typename Func>
class then_sender namespace();
```

P2822 Providing user control of associated entities of class types

FIXING ADL FOR NTTPS

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity namespace(decltype(R));
```

IMPROVING COMPILE-TIMES

```
template<typename Source,typename Func>
class then_sender namespace();
```

NARROWING DOWN THE SET OF ASSOCIATED ENTITIES

```
template<typename T,typename Allocator = std::allocator<T>>
class array_map namespace(T);
```

Sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

Sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

Sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

```
derived_unit<si::kilo_<si::gram>, si::metre>
```

Sorting types at compile time

```
template<TypeList List, template<typename, typename> typename Pred>
using type_list_sort = type_list_sort_impl<List, Pred>::type;
```

Sorting types at compile time

```
template<TypeList List, template<typename, typename> typename Pred>
using type_list_sort = type_list_sort_impl<List, Pred>::type;
```

```
template<typename T>
[[nodiscard]] consteval std::string_view type_name();

template<typename Lhs, typename Rhs>
struct type_name_less : std::bool_constant<type_name<Lhs>() < type_name<Rhs>()> {};
```

Sorting types at compile time

```
template<TypeList List, template<typename, typename> typename Pred>
using type_list_sort = type_list_sort_impl<List, Pred>::type;
```

```
template<typename T>
[[nodiscard]] consteval std::string_view type_name();

template<typename Lhs, typename Rhs>
struct type_name_less : std::bool_constant<type_name<Lhs>() < type_name<Rhs>()> {};
```

```
static_assert(is_same_v<type_list_sort<type_list<B, A, C>, type_name_less>,
                        type_list<A, B, C>>);
```

Issues with sorting types at compile time

```cpp
template<typename T>
[[nodiscard]] consteval std::string_view type_name()
{
  std::string_view name, prefix, suffix;
#ifdef __clang__
  name = __PRETTY_FUNCTION__;
  prefix = "auto type_name() [T = ";
  suffix = "]";
#elif defined(__GNUC__)
  name = __PRETTY_FUNCTION__;
  prefix = "constexpr auto type_name() [with T = ";
  suffix = "]";
#elif defined(_MSC_VER)
  name = __FUNCSIG__;
  prefix = "auto __cdecl type_name<";
  suffix = ">(void)";
#endif
  name.remove_prefix(prefix.size());
  name.remove_suffix(suffix.size());
  return name;
}
```

Issues with sorting types at compile time

```cpp
template<typename T>
[[nodiscard]] consteval std::string_view type_name()
{
  std::string_view name, prefix, suffix;
#ifdef __clang__
  name = __PRETTY_FUNCTION__;
  prefix = "auto type_name() [T = ";
  suffix = "]";
#elif defined(__GNUC__)
  name = __PRETTY_FUNCTION__;
  prefix = "constexpr auto type_name() [with T = ";
  suffix = "]";
#elif defined(_MSC_VER)
  name = __FUNCSIG__;
  prefix = "auto __cdecl type_name<";
  suffix = ">(void)";
#endif
```

Non-C++ standard hack 🤓

```cpp
}
```

Issues with sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

```
derived_unit<si::kilo_<si::gram>, si::metre>
```

Issues with sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

```
derived_unit<si::kilo_<si::gram>, si::metre>
```

```
namespace {

inline constexpr struct cows final : named_unit<"cows", one> {} cows;

}
```

```
constexpr unit auto u3 = cows * si::metre;
```

Issues with sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

```
derived_unit<si::kilo_<si::gram>, si::metre>
```

```
namespace {

inline constexpr struct cows final : named_unit<"cows", one> {} cows;

}
```

```
constexpr unit auto u3 = cows * si::metre;
```

GCC

```
derived_unit<si::metre, {anonymous}::cows>
```

Issues with sorting types at compile time

```
constexpr unit auto u1 = si::kilo<si::gram> * si::metre;
constexpr unit auto u2 = si::metre * si::kilo<si::gram>;
```

```
static_assert(std::is_same_v<decltype(u1), decltype(u2)>);
```

```
derived_unit<si::kilo_<si::gram>, si::metre>
```

```
namespace {

inline constexpr struct cows final : named_unit<"cows", one> {} cows;

}
```

```
constexpr unit auto u3 = cows * si::metre;
```

GCC

```
derived_unit<si::metre, {anonymous}::cows>
```

CLANG

```
derived_unit<(anonymous namespace)::cows, si::metre>
```

P2830 Standardized Constexpr Type Ordering (C++26)

```
static_assert(is_same_v<type_list_sort<type_list<B, A, C>, std::type_order>,
                        type_list<A, B, C>>);
```

P2830 Standardized Constexpr Type Ordering (C++26)

```
static_assert(is_same_v<type_list_sort<type_list<B, A, C>, std::type_order>,
                        type_list<A, B, C>>);
```

- Canonicalized *type sets*
  - **`std::is_same_v<typeset<A, B>, typeset<B, A, A>>`**

P2830 Standardized Constexpr Type Ordering (C++26)

```
static_assert(is_same_v<type_list_sort<type_list<B, A, C>, std::type_order>,
                        type_list<A, B, C>>);
```

- Canonicalized *type sets*
  - **std::is_same_v<typeset<A, B>, typeset<B, A, A>>**
- Canonicalizing *policy-based libraries*
  - **using T1 = lib::dyn<ostreamable, iterable<int>, scalar_multiplicable<int>>;**

P2830 Standardized Constexpr Type Ordering (C++26)

```
static_assert(is_same_v<type_list_sort<type_list<B, A, C>, std::type_order>,
                        type_list<A, B, C>>);
```

- Canonicalized *type sets*
  - **std::is_same_v<typeset<A, B>, typeset<B, A, A>>**
- Canonicalizing *policy-based libraries*
  - **using T1 = lib::dyn<ostreamable, iterable<int>, scalar_multiplicable<int>>;**
- Canonicalized *variant*
  - **template<typename... Ts> using variant_for = canonicalized<std::variant, Ts...>;**

Always use constructor initializer lists

Always use constructor initializer lists

```cpp
template<typename T>
class measurement {
public:
  using value_type = T;

  measurement() = default;

  constexpr explicit measurement(value_type val, const value_type& err = {}) : value_(std::move(val))
  {
    using namespace std;
    uncertainty_ = abs(err);
  }

  // ...

private:
  value_type value_{};
  value_type uncertainty_{};
};
```

Using declaration can't be provided in a constructor initializer list.

Proper math for wrapping types

```
template<auto R, typename Rep>
  requires requires(Rep v) { abs(v); } || requires(Rep v) { std::abs(v); }
[[nodiscard]] constexpr Quantity auto abs(const quantity<R, Rep>& q) noexcept
{
  using std::abs;
  return quantity{abs(q.numerical_value_ref_in(q.unit)), R};
}
```

Using declaration can't be provided in constraints.

Should we have CPOs for mathematical functions?

Many people work now on libraries providing custom "smart" numeric representation types/wrappers.

Should we have CPOs for mathematical functions?

Many people work now on libraries providing custom "smart" numeric representation types/wrappers.

CPOs for math functions could increase interop between various numeric-related types.

Customization Point Object (CPO)

```cpp
template<typename T>
concept not_void = (!std::is_void_v<T>);
```

```cpp
namespace math {
namespace abs_impl {

void abs() = delete;  // poison pill

struct abs_t {
  [[nodiscard]] constexpr auto operator()(const auto& val) const
    requires requires { std::abs(val); } ||
             requires { { val.abs() } -> not_void; } || requires { { abs(val) } -> not_void; }
  {
    if constexpr (requires { val.abs(); }) return val.abs();
    else if constexpr (requires { abs(val); }) return abs(val);
    else return std::abs(val);
  }
};

}  // namespace abs_impl

inline constexpr abs_impl::abs_t abs;
}  // namespace math
```

Customization Point Object (CPO)

```cpp
template<typename T>
concept not_void = (!std::is_void_v<T>);
```

```cpp
namespace math {
namespace abs_impl {

void abs() = delete;  // poison pill

struct abs_t {
  [[nodiscard]] constexpr auto operator()(const auto& val) const
    requires requires { std::abs(val); } ||
             requires { { val.abs() } -> not_void; } || requires { { abs(val) } -> not_void; }
  {
    if constexpr (requires { val.abs(); }) return val.abs();
    else if constexpr (requires { abs(val); }) return abs(val);
    else return std::abs(val);
  }
};

}  // namespace abs_impl
```

More powerful than the previous simple checks.

Proper math for wrapping types

BEFORE

```
template<auto R, typename Rep>
  requires requires(Rep v) { abs(v); } || requires(Rep v) { std::abs(v); }
[[nodiscard]] constexpr Quantity auto abs(const quantity<R, Rep>& q) noexcept
{
  using std::abs;
  return quantity{abs(q.numerical_value_ref_in(q.unit)), R};
}
```

Proper math for wrapping types

BEFORE

```
template<auto R, typename Rep>
  requires requires(Rep v) { abs(v); } || requires(Rep v) { std::abs(v); }
[[nodiscard]] constexpr Quantity auto abs(const quantity<R, Rep>& q) noexcept
{
  using std::abs;
  return quantity{abs(q.numerical_value_ref_in(q.unit)), R};
}
```

AFTER

```
template<auto R, typename Rep>
  requires requires(Rep v) { math::abs(v); }
[[nodiscard]] constexpr Quantity auto abs(const quantity<R, Rep>& q) noexcept
{
  return quantity{math::abs(q.numerical_value_ref_in(q.unit)), R};
}
```

ENABLING COMPILE-TIME CODE DEBUGGING

Compile-time code debugging

Traditional debugging of **`constexpr`** code is hard. Debugging of **`consteval`** code is impossible.

Compile-time code debugging

Traditional debugging of **`constexpr`** code is hard. Debugging of **`consteval`** code is impossible.

```
template<QuantitySpec From, QuantitySpec To>
[[nodiscard]] consteval specs_convertible_result convertible_impl(From from, To to)
{
  using enum specs_convertible_result;

  if constexpr (From{} == To{})
    return yes;
  else if constexpr (From::dimension != To::dimension)
    return no;
  else if constexpr (QuantityKindSpec<From> || QuantityKindSpec<To>)
    return convertible_kinds(from, to);
  else if constexpr (NamedQuantitySpec<From> && NamedQuantitySpec<To>)
    return convertible_named(from, to);
  else
    return are_ingredients_convertible(from, to);
}
```

Compile-time code debugging

Traditional debugging of **`constexpr`** code is hard. Debugging of **`consteval`** code is impossible.

```
template<QuantitySpec From, QuantitySpec To>
[[nodiscard]] consteval specs_convertible_result convertible_impl(From from, To to)
{
  using enum specs_convertible_result;

  if constexpr (From{} == To{})
    return yes;
  else if constexpr (From::dimension != To::dimension)
    return no;
  else if constexpr (QuantityKindSpec<From> || QuantityKindSpec<To>)
    return convertible_kinds(from, to);
  else if constexpr (NamedQuantitySpec<From> && NamedQuantitySpec<To>)
    return convertible_named(from, to);
  else
    return are_ingredients_convertible(from, to);
}
```

P2758 Emitting messages at compile time

```cpp
template<QuantitySpec From, QuantitySpec To>
[[nodiscard]] consteval specs_convertible_result convertible(From from, To to)
{
  using enum specs_convertible_result;

  if constexpr (From{} == To{}) {
    std::constexpr_print_str("convertible: same");
    return yes;
  }
  else if constexpr (From::dimension != To::dimension) {
    std::constexpr_print_str("convertible: different dimension");
    return no;
  }
  else if constexpr (QuantityKindSpec<From> || QuantityKindSpec<To>) {
    std::constexpr_print_str("convertible: kind comparison");
    return convertible_kinds(from, to);
  }
  else if constexpr (NamedQuantitySpec<From> && NamedQuantitySpec<To>) {
    std::constexpr_print_str("convertible: named comparison");
    return convertible_named(from, to);
  }
  else {
    std::constexpr_print_str("convertible: ingredients comparison");
    return are_ingredients_convertible(from, to);
  }
}
```

P2758 Emitting messages at compile time

```cpp
template<QuantitySpec From, QuantitySpec To>
[[nodiscard]] consteval specs_convertible_result convertible(From from, To to)
{
  using enum specs_convertible_result;

  if constexpr (From{} == To{}) {
    std::constexpr_print_str("mp-units-debug-prints", "convertible: same");
    return yes;
  }
  else if constexpr (From::dimension != To::dimension) {
    std::constexpr_print_str("mp-units-debug-prints", "convertible: different dimension");
    return no;
  }
  else if constexpr (QuantityKindSpec<From> || QuantityKindSpec<To>) {
    std::constexpr_print_str("mp-units-debug-prints", "convertible: kind comparison");
    return convertible_kinds(from, to);
  }
  else if constexpr (NamedQuantitySpec<From> && NamedQuantitySpec<To>) {
    std::constexpr_print_str("mp-units-debug-prints", "convertible: named comparison");
    return convertible_named(from, to);
  }
  else {
    std::constexpr_print_str("mp-units-debug-prints", "convertible: ingredients comparison");
    return are_ingredients_convertible(from, to);
  }
}
```

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code*
  - in case of translations some **`format`** arguments may be unused

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱
  - in case of translations some **`format`** arguments may be unused

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳
  - in case of translations some **`format`** arguments may be unused

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳 🤓
  - in case of translations some **`format`** arguments may be unused

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳 🤓 🤣
  - in case of translations some **`format`** arguments may be unused

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳 🤓 🤣
  - in case of translations some **`format`** arguments may be unused
- Most users would probably like to *diagnose such issues* in their format strings

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳 🤓 🤣
  - in case of translations some **`format`** arguments may be unused
- Most users would probably like to *diagnose such issues* in their format strings

```
std::constexpr_warning_str("format-too-many-args",
                           "Format string consumed less arguments than were provided.");
```

P2758 Emitting messages at compile time

```
std::format("x={} and y=", x, y);
```

- *Valid but surprising code* 😱 😳 🤓 🤣
  - in case of translations some **`format`** arguments may be unused
- Most users would probably like to *diagnose such issues* in their format strings

```
std::constexpr_warning_str("format-too-many-args",
                           "Format string consumed less arguments than were provided.");
```

- Compilers may allow setting this warning tag from the *command line options or with pragmas*
  - enabled with **`-Wformat-too-many-args`**
  - disabled with **`-Wno-format-too-many-args`**

P2758 Emitting messages at compile time

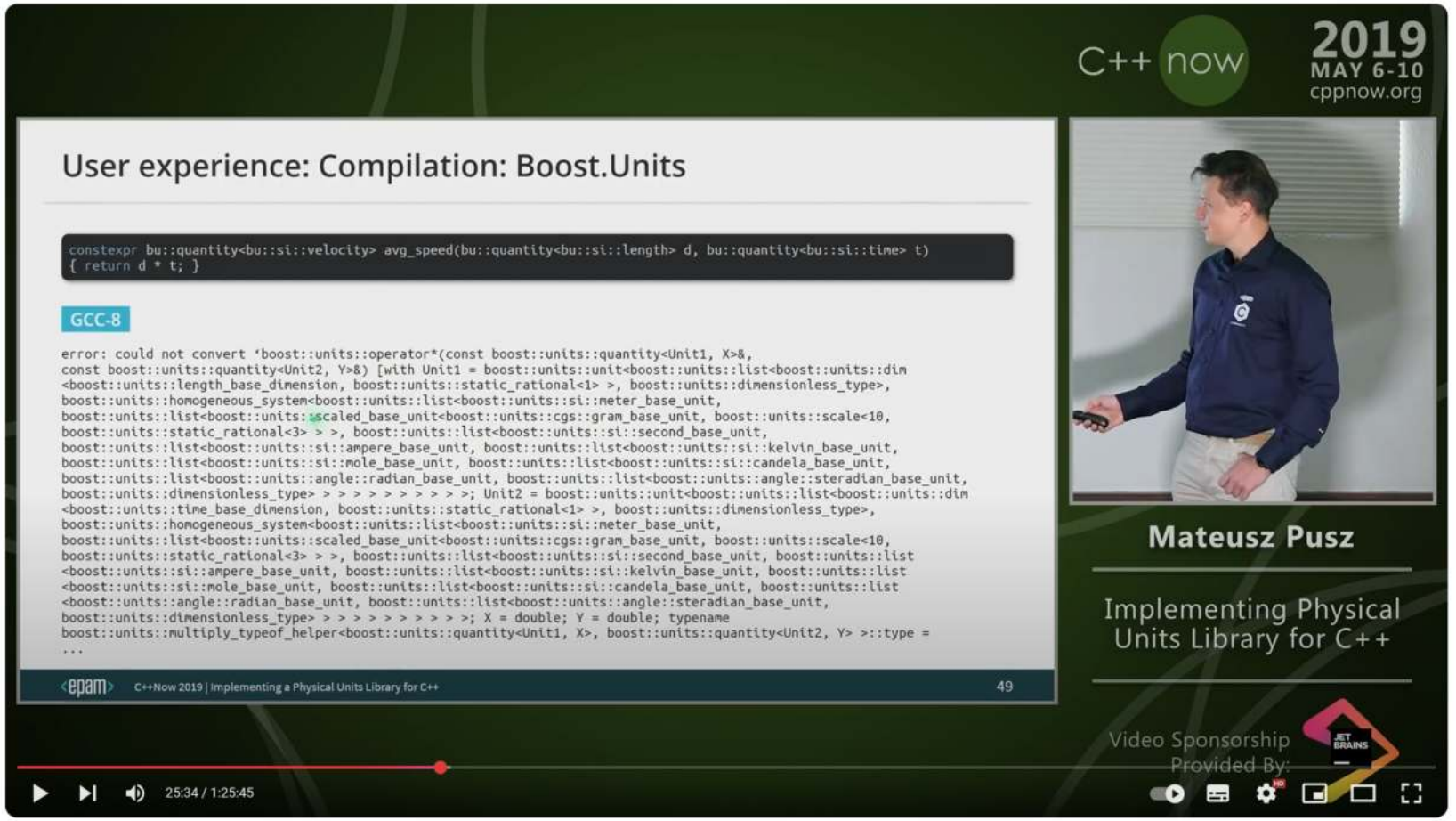
```
consteval int foo(int a)
{
  if (a == 0)
    std::constexpr_error_str("reject-zero", "can't call with a == 0");
  return a;
}

int x = foo(2); // OK
int y = foo(0); // error: reject-zero, can't call with a == 0
```

BETTER COMPILE-TIME ERROR MESSAGES

Question

Who thinks that the compilation error messages from generic C++ libraries are short, easy to read, and understand?

Unreadable error messages



C++Now 2019: Mateusz Pusz "Implementing Physical Units Library for C++"

Converting units

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  static constexpr Reference auto reference = R;
  static constexpr QuantitySpec auto quantity_spec = get_quantity_spec(reference);
  static constexpr Unit auto unit = get_unit(reference);
  using rep = Rep;









};
```

Converting units

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  static constexpr Reference auto reference = R;
  static constexpr QuantitySpec auto quantity_spec = get_quantity_spec(reference);
  static constexpr Unit auto unit = get_unit(reference);
  using rep = Rep;

  template<UnitOf<quantity_spec> ToU>
    requires ValuePreservingScaling<unit, ToU{}, rep>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Example: Invalid type (clang-20)

```
quantity q = 42 * m;
quantity q1 = q.in(isq::time);
```

Example: Invalid type (clang-20)

```
quantity q = 42 * m;
quantity q1 = q.in(isq::time);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q1 = q.in(isq::time);
      |                   ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct time]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::isq::time, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'mp_units::isq::time' does not satisfy 'Unit'
  123 |   Unit<U> && QuantitySpec<decltype(QS)> && ((!AssociatedUnit<U>) || UnitOf<U, QS>);
      |   ^
note: because 'std::derived_from<mp_units::isq::time, unit_interface>' evaluated to false
   46 | concept Unit = std::derived_from<T, unit_interface> && SymbolicConstant<T>;
      |                ^
note: because 'is_base_of_v<mp_units::unit_interface, mp_units::isq::time>' evaluated to false
   27 | concept derived_from = is_base_of_v<_Bp, _Dp> && is_convertible_v<const volatile _Dp*, const volatile _Bp*>;
      |                       ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Invalid type (clang-20)

```
quantity q = 42 * m;
quantity q1 = q.in(isq::time);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q1 = q.in(isq::time);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct time]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::isq::time, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'mp_units::isq::time' does not satisfy 'Unit'
  123 |   Unit<U> && QuantitySpec<decltype(QS)> && ((!AssociatedUnit<U>) || UnitOf<U, QS>);
      |   ^
note: because 'std::derived_from<mp_units::isq::time, unit_interface>' evaluated to false
   46 | concept Unit = std::derived_from<T, unit_interface> && SymbolicConstant<T>;
      |                ^
note: because 'is_base_of_v<mp_units::unit_interface, mp_units::isq::time>' evaluated to false
   27 | concept derived_from = is_base_of_v<_Bp, _Dp> && is_convertible_v<const volatile _Dp*, const volatile _Bp*>;
      |                       ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Invalid type (clang-20)

```
quantity q = 42 * m;
quantity q1 = q.in(isq::time);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q1 = q.in(isq::time);
      |                   ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct time]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::isq::time, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'mp_units::isq::time' does not satisfy 'Unit'`
  123 |   Unit<U> && QuantitySpec<decltype(QS)> && ((!AssociatedUnit<U>) || UnitOf<U, QS>);
      |   ^
note: because 'std::derived_from<mp_units::isq::time, unit_interface>' evaluated to false
   46 | concept Unit = std::derived_from<T, unit_interface> && SymbolicConstant<T>;
      |                ^
note: because 'is_base_of_v<mp_units::unit_interface, mp_units::isq::time>' evaluated to false
   27 | concept derived_from = is_base_of_v<_Bp, _Dp> && is_convertible_v<const volatile _Dp*, const volatile _Bp*>;
      |                       ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q2 = q.in(s);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct second]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::si::second, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'implicitly_convertible(get_quantity_spec(mp_units::si::second{}),
                                      kind_of_<decltype(struct length{{{{}}}})>{{{{}}}})' evaluated to false
  114 |   (implicitly_convertible(get_quantity_spec(U{}), QS) ||
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q2 = q.in(s);
      |                   ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct second]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::si::second, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'implicitly_convertible(get_quantity_spec(mp_units::si::second{}),
                                      kind_of_<decltype(struct length{{{{}}}})>{{{{}}}})' evaluated to false
  114 |   (implicitly_convertible(get_quantity_spec(U{}), QS) ||
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q2 = q.in(s);
      |                   ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct second]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::si::second, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'implicitly_convertible(get_quantity_spec(mp_units::si::second{}),
                                      kind_of_<length>{})' evaluated to false
  114 |   (implicitly_convertible(get_quantity_spec(U{}), QS) ||
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q2 = q.in(s);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct second]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::si::second, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'implicitly_convertible(get_quantity_spec(mp_units::si::second{}),
                                      kind_of_<length>{})' evaluated to false
  114 |   (implicitly_convertible(get_quantity_spec(U{}), QS) ||
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Unit of a different quantity kind (clang-20)

```
quantity q = 42 * m;
quantity q2 = q.in(s);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q2 = q.in(s);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = struct second]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'UnitOf<mp_units::si::second, quantity_spec>' evaluated to false
  247 |   template<UnitOf<quantity_spec> ToU>
      |            ^
note: because 'implicitly_convertible(kind_of_<time>{},
                                      kind_of_<length>{})' evaluated to false
  114 |   (implicitly_convertible(get_quantity_spec(U{}), QS) ||
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Example: Value truncation (clang-20)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

Example: Value truncation (clang-20)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = kilo_<decltype(metre{{{}}})>]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
                                      mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}})' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |
1 error generated.
```

Error messages don't have to be scarry

In many cases it is possible to provide user-friendly error messages in generic libraries already.

Error messages don't have to be scarry

In many cases it is possible to provide user-friendly error messages in generic libraries already.

Generic library authors should care more about producing readable error messages.

Example: Value truncation (gcc-15)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
In function ‘int main()’:
error: no matching function for call to 'mp_units::quantity<mp_units::si::metre(), int>::in(const mp_units::si::kilo_<mp_units::si::metre>&)'
   17 |   quantity q3 = q.in(km);
      |                 ~~~~^~~~
note: there are 3 candidates
note: candidate 1: 'template<class ToU>  requires (UnitOf<ToU, mp_units::quantity<R, Rep>::quantity_spec>) && (ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, {}, Rep>) constexpr auto
  [requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>] mp_units::quantity<R, Rep>::in(ToU) const [with auto [requires mp_units::Reference<<placeholder>, >]
  R = mp_units::si::metre(); Rep = int]'
  249 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                         ^~
note: template argument deduction/substitution failed:
note: constraints not satisfied
In substitution of 'template<class ToU>  requires (UnitOf<ToU, mp_units::quantity<R, Rep>::quantity_spec>) && (ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, {}, Rep>) constexpr auto
[requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>] mp_units::quantity<mp_units::si::metre(), int>::in(ToU) const [with ToU = mp_units::si::kilo_<mp_units::si::metre>]':
required from here
   17 |   quantity q3 = q.in(km);
      |                 ~~~~^~~~
  required for the satisfaction of 'ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, ToU{}, Rep>'
  [with ToU = mp_units::si::kilo_<mp_units::si::metre>; R = mp_units::si::metre{}; Rep = int]
note: no operand of the disjunction is satisfied
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit));
      |   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: set '-fconcepts-diagnostics-depth=' to at least 2 for more detail
note: candidate 2: 'template<class ToRep>  requires (RepresentationOf<ToRep, mp_units::quantity<R, Rep>::quantity_spec>) && (ValuePreservingConstruction<ToRep, Rep>)
constexpr auto [requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>] mp_units::quantity<R, Rep>::in() const [with auto [requires mp_units::Reference<<placeholder>, >]
R = mp_units::si::metre(); Rep = int]'
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                         ^~
note: candidate expects 0 arguments, 1 provided
note: candidate 3: 'template<class ToRep, class ToU>  requires (RepresentationOf<ToRep, mp_units::quantity<R, Rep>::quantity_spec>) && (UnitOf<ToU, mp_units::quantity<R, Rep>::quantity_spec>) &&
((ValuePreservingConstruction<ToRep, Rep>) && (ValuePreservingScaling2Reps<mp_units::quantity<R, Rep>::unit, Rep, {}, ToRep>)) constexpr auto
[requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>]
mp_units::quantity<R, Rep>::in(ToU) const [with ToU = ToRep; auto [requires mp_units::Reference<<placeholder>, >] R = mp_units::si::metre(); Rep = int]'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                         ^~
note: template argument deduction/substitution failed:
note:   couldn't deduce template parameter 'ToRep'
   17 |   quantity q3 = q.in(km);
```

Example: Value truncation (gcc-15)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
...
note: candidate 1: 'template<class ToU>  requires (UnitOf<ToU, mp_units::quantity<R, Rep>::quantity_spec>) &&
(ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, {}, Rep>) constexpr auto
  [requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>] mp_units::quantity<R, Rep>::in(ToU) const
  [with auto [requires mp_units::Reference<<placeholder>, >]
  R = mp_units::si::metre(); Rep = int]'
  249 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^~
note: template argument deduction/substitution failed:
note: constraints not satisfied
In substitution of 'template<class ToU>  requires (UnitOf<ToU, mp_units::quantity<R, Rep>::quantity_spec>) &&
(ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, {}, Rep>) constexpr auto
[requires mp_units::QuantityOf<<placeholder>, mp_units::quantity<R, Rep>::quantity_spec>]
mp_units::quantity<mp_units::si::metre(), int>::in(ToU) const [with ToU = mp_units::si::kilo_<mp_units::si::metre>]':
required from here
   17 |   quantity q3 = q.in(km);
      |                 ~~~~^~~~
  required for the satisfaction of 'ValuePreservingScaling<mp_units::quantity<R, Rep>::unit, ToU{}, Rep>'
  [with ToU = mp_units::si::kilo_<mp_units::si::metre>; R = mp_units::si::metre{}; Rep = int]
note: no operand of the disjunction is satisfied
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit));
      |   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: set '-fconcepts-diagnostics-depth=' to at least 2 for more detail
...
```

Converting units

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<UnitOf<quantity_spec> ToU>
    requires ValuePreservingScaling<unit, ToU{}, rep>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Converting units

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>, "incompatible unit");
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>, "not-value preserving");
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Example: Value truncation (clang-20)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: static assertion failed: not-value preserving
  253 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>, "not-value preserving");
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
          in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                   ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  253 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>, "not-value preserving");
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}}, mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}})'
      evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
...
2 errors generated.
```

P2741 user-generated `static_assert` messages (C++26)

```cpp
consteval std::string constexpr_format(auto fmt, auto&&... args)
{
  std::string text;
  fmt::format_to(std::back_inserter(text), fmt, std::forward<decltype(args)>(args)...);
  return text;
};
```

P2741 user-generated `static_assert` messages (C++26)

```cpp
consteval std::string constexpr_format(auto fmt, auto&&... args)
{
  std::string text;
  fmt::format_to(std::back_inserter(text), fmt, std::forward<decltype(args)>(args)...);
  return text;
};
```

```cpp
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>,
                  constexpr_format(FMT_COMPILE("Invalid unit '{}'"), unit_symbol(ToU{})));
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
                  constexpr_format(FMT_COMPILE("Scaling from '{}' to '{}' is not value-preserving"),
                                   unit_symbol(unit), unit_symbol(ToU{})));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Example: Value truncation (clang-20)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: static assertion failed: Scaling from 'm' to 'km' is not value-preserving
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
          in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                   ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
...
2 errors generated.
```

P3391 constexpr std::format

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>,
                  std::format("Invalid unit '{}'", unit_symbol(ToU{})));
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
                  std::format("Scaling from '{}' to '{}' is not value-preserving",
                              unit_symbol(unit), unit_symbol(ToU{})));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

P2996 Reflection for C++26

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>,
                  std::format("Unit '{}' is associated with quantity of kind '{}' "
                              "which is not convertible to the '{}' quantity",
                              unit_symbol(ToU{}),
                              display_string_of(get_quantity_spec(U{})._quantity_spec_),
                              display_string_of(QS)));
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
                  std::format("Scaling from '{}' to '{}' is not value-preserving for '{}' representation type",
                              unit_symbol(unit), unit_symbol(ToU{}), display_string_of(dealias(^^rep))));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Example: Value truncation (clang-20)

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: static assertion failed: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
           in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                   ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
...
2 errors generated.
```

Problems with `static_assert()`

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: static assertion failed: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
           in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                   ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
...
2 errors generated.
```

Problems with `static_assert()`

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: static assertion failed: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
          in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                   ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
error: no matching constructor for initialization of 'quantity<make_reference(quantity_spec, kilo_<metre>{}), int>'
  255 |     return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
      |            ^                                                 ~~~~~~~
129 lines more ...
2 errors generated.
```

Problems with `static_assert()`

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>,
                  std::format("Unit '{}' is associated with quantity of kind '{}' "
                              "which is not convertible to the '{}' quantity",
                              unit_symbol(ToU{}),
                              display_string_of(get_quantity_spec(U{})._quantity_spec_),
                              display_string_of(QS)));
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
                  std::format("Scaling from '{}' to '{}' is not value-preserving for '{}' representation type",
                              unit_symbol(unit), unit_symbol(ToU{}), display_string_of(dealias(^^rep))));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Problems with `static_assert()`

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    static_assert(UnitOf<ToU, quantity_spec>,
                  std::format("Unit '{}' is associated with quantity of kind '{}' "
                              "which is not convertible to the '{}' quantity",
                              unit_symbol(ToU{}),
                              display_string_of(get_quantity_spec(U{})._quantity_spec_),
                              display_string_of(QS)));
    static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
                  std::format("Scaling from '{}' to '{}' is not value-preserving for '{}' representation type",
                              unit_symbol(unit), unit_symbol(ToU{}), display_string_of(dealias(^^rep))));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Even when the **`static_assert`** fails, the compiler still tries to determine the return type of the function.

Problems with `static_assert()`

```
template<Quantity Q, Unit U>
constexpr Quantity auto try_convert_to(Q q, U u)
{
  if constexpr (requires { q.in(u); })
    return q.in(u);
  else
    return q;
}
```

```
quantity q = 42 * m;
quantity q4 = try_convert_to(q, km);
```

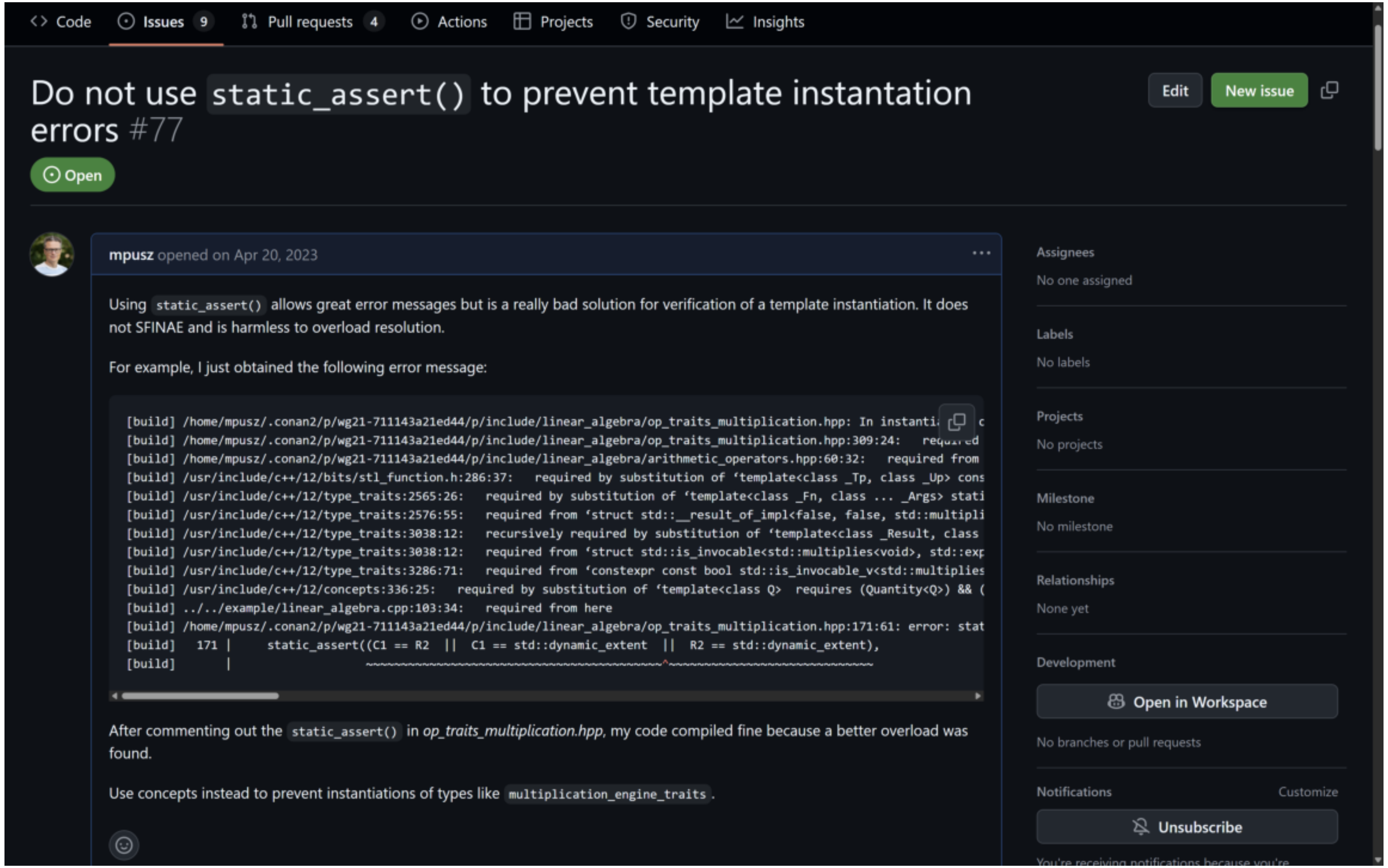
Problems with `static_assert()`

```
template<Quantity Q, Unit U>
constexpr Quantity auto try_convert_to(Q q, U u)
{
  if constexpr (requires { q.in(u); })
    return q.in(u);
  else
    return q;
}
```

```
quantity q = 42 * m;
quantity q4 = try_convert_to(q, km);
```

```
error: static assertion failed: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::
          in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
   10 |   quantity q3 = q.in(km);
      |                 ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  256 |     static_assert(ValuePreservingScaling<unit, ToU{}, rep>,
      |                   ^
note: because 'treat_as_floating_point<int>' evaluated to false
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |    ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}},
   82 |   (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit))
      |                                    ^
...
```

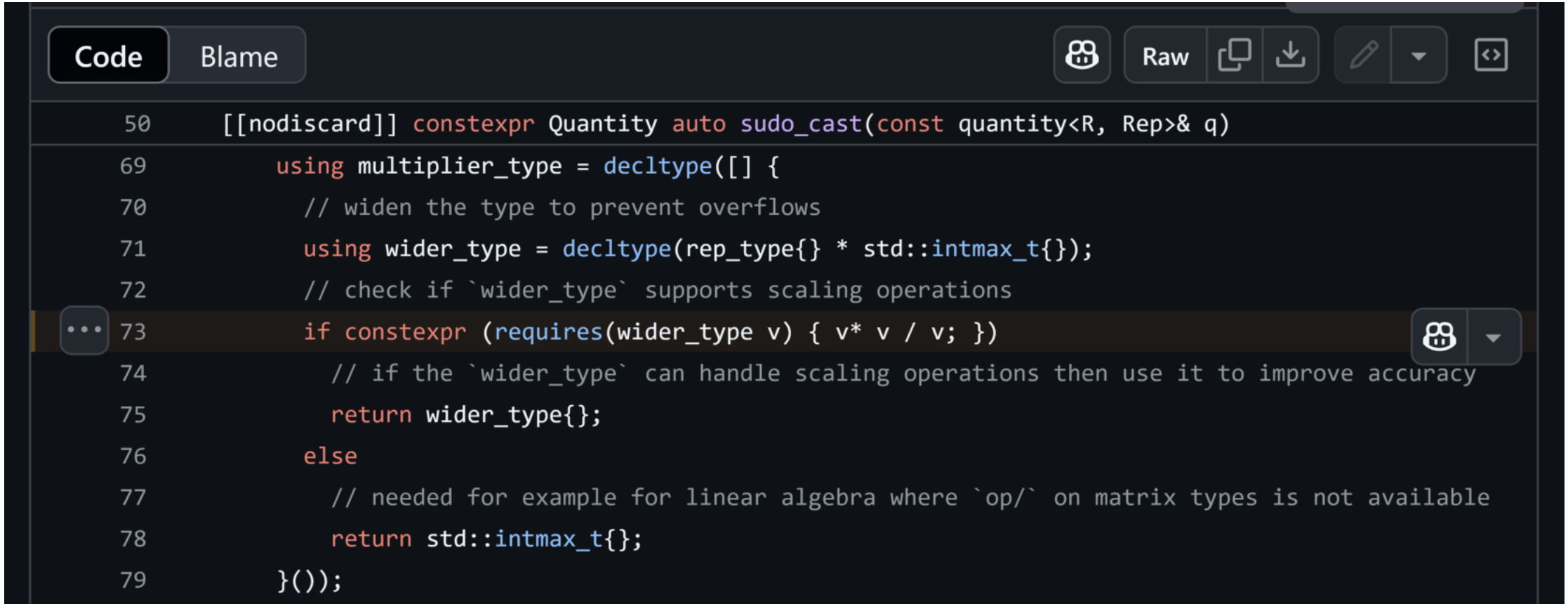
Problems with `static_assert()`



Problems with `static_assert()`

```
template<class COTR, class ET1, class ET2>
struct multiplication_engine_traits
{
  private:
    using element_type_1 = typename ET1::element_type;
    using element_type_2 = typename ET2::element_type;

    static constexpr size_t     R1 = engine_extents_helper<ET1>::rows();
    static constexpr size_t     C1 = engine_extents_helper<ET1>::columns();
    static constexpr size_t     R2 = engine_extents_helper<ET2>::rows();
    static constexpr size_t     C2 = engine_extents_helper<ET2>::columns();

    //- Validate the extents.
    //
    static_assert((C1 == R2  ||  C1 == std::dynamic_extent  ||  R2 == std::dynamic_extent),
                  "mis-matched/invalid number of rows and columns for multiplication");

    // ...
};
```

Problems with `static_assert()`



Problems with `static_assert()`

```
template<template<auto, typename> typename Q>
concept invalid_unit_conversion = requires {
  requires !requires { Q<isq::length[m], int>(2000 * m).in(km); };     // truncating conversion
  requires !requires { Q<isq::length[m], int>(2 * m).in(s); };         // unit of a different quantity & dimension
  requires !requires { Q<isq::frequency[Hz], int>(60 * Hz).in(Bq); };  // unit of a different kind (same dimension)
  requires !requires { Q<isq::length[m], int>(2 * m).force_in(s); };   // unit of a different quantity & dimension
  requires !requires {
    Q<isq::frequency[Hz], int>(60 * Hz).force_in(Bq);
  };  // unit of a different kind (same dimension)
};
static_assert(invalid_unit_conversion<quantity>);
```

Problems with `static_assert()`

```
template<template<auto, typename> typename Q>
concept invalid_unit_conversion = requires {
  requires !requires { Q<isq::length[m], int>(2000 * m).in(km); };     // truncating conversion
  requires !requires { Q<isq::length[m], int>(2 * m).in(s); };         // unit of a different quantity & dimension
  requires !requires { Q<isq::frequency[Hz], int>(60 * Hz).in(Bq); };  // unit of a different kind (same dimension)
  requires !requires { Q<isq::length[m], int>(2 * m).force_in(s); };   // unit of a different quantity & dimension
  requires !requires {
    Q<isq::frequency[Hz], int>(60 * Hz).force_in(Bq);
  };  // unit of a different kind (same dimension)
};
static_assert(invalid_unit_conversion<quantity>);
```

Usage of **`static_assert`** prevents unit testing of negative scenarios.

Problems with `static_assert()`

Introduction of requires-expressions in C++20 made it so easy to enable user-friendly development of smarter, safer, faster libraries.

Problems with `static_assert()`

Introduction of requires-expressions in C++20 made it so easy to enable user-friendly development of smarter, safer, faster libraries.

`static_assert` is not SFINAE-friendly. Prefer the usage of constraints and concepts to diagnose errors.

P2758 Emitting messages at compile time

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    if constexpr (!UnitOf<ToU, quantity_spec>)
      std::constexpr_error_str("invalid-unit-conversion",
                               std::format("Unit '{}' is associated with quantity of kind '{}' "
                                           "which is not convertible to the '{}' quantity",
                                           unit_symbol(ToU{}),
                                           display_string_of(get_quantity_spec(U{})._quantity_spec_),
                                           display_string_of(QS)));
    if constexpr (!ValuePreservingScaling<unit, ToU{}, rep>)
      std::constexpr_error_str("invalid-unit-conversion",
                               std::format("Scaling from '{}' to '{}' is not value-preserving "
                                           "for '{}' representation type",
                                           unit_symbol(unit), unit_symbol(ToU{}),
                                           display_string_of(dealias(^^rep))));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

Future extensions to P2758 with constexpr std::format

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    if constexpr (!UnitOf<ToU, quantity_spec>)
      std::constexpr_error("invalid-unit-conversion",
                           "Unit '{}' is associated with quantity of kind '{}' "
                           "which is not convertible to the '{}' quantity",
                           unit_symbol(ToU{}),
                           display_string_of(get_quantity_spec(U{})._quantity_spec_),
                           display_string_of(QS));
    if constexpr (!ValuePreservingScaling<unit, ToU{}, rep>)
      std::constexpr_error("invalid-unit-conversion",
                           "Scaling from '{}' to '{}' is not value-preserving for '{}' representation type",
                           unit_symbol(unit), unit_symbol(ToU{}), display_string_of(dealias(^^rep)));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

P2758 Emitting messages at compile time

```
template<Quantity Q, Unit U>
constexpr Quantity auto try_convert_to(Q q, U u)
{
  if constexpr (requires { q.in(u); })
    return q.in(u);
  else
    return q;
}
```

```
quantity q = 42 * m;
quantity q4 = try_convert_to(q, km);
```

P2758 Emitting messages at compile time

```
template<Quantity Q, Unit U>
constexpr Quantity auto try_convert_to(Q q, U u)
{
  if constexpr (requires { q.in(u); })
    return q.in(u);
  else
    return q;
}
```

```
quantity q = 42 * m;
quantity q4 = try_convert_to(q, km);
```

Not SFINAE-friendly 😥

C++ exceptions

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<Unit ToU>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    if constexpr (!UnitOf<ToU, quantity_spec>)
      throw invalid_unit_conversion(std::format("Unit '{}' is associated with quantity of kind '{}' "
                                                "which is not convertible to the '{}' quantity",
                                                unit_symbol(ToU{}),
                                                display_string_of(get_quantity_spec(U{})._quantity_spec_),
                                                display_string_of(QS)));
    if constexpr (!ValuePreservingScaling<unit, ToU{}, rep>)
      throw invalid_unit_conversion(std::format("Scaling from '{}' to '{}' is not value-preserving "
                                                "for '{}' representation type",
                                                unit_symbol(unit), unit_symbol(ToU{}),
                                                display_string_of(dealias(^^rep))));
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

C++ exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: no matching constructor for initialization of 'quantity<detail::make_reference(quantity_spec, kilo_<metre>{}), int>'
  281 |     return quantity<detail::make_reference(quantity_spec, ToU{}), Rep>{*this};
      |            ^                                                      ~~~~~~~~
132 lines more...
1 error generated.
```

C++ exceptions

```cpp
struct unsatisfied_constraints {
  std::string msg;
  [[nodiscard]] consteval const char* what() const noexcept { return msg.c_str(); }
};

template<mp_units::fixed_string Fmt, typename... Args>
[[nodiscard]] consteval bool unsatisfied([[maybe_unused]] Args&&... args)
{
  throw unsatisfied_constraints{std::format(Fmt.c_str(), std::forward<Args>(args)...)};
}
```

C++ exceptions

```
struct unsatisfied_constraints {
  std::string msg;
  [[nodiscard]] consteval const char* what() const noexcept { return msg.c_str(); }
};

template<mp_units::fixed_string Fmt, typename... Args>
[[nodiscard]] consteval bool unsatisfied([[maybe_unused]] Args&&... args)
{
  throw unsatisfied_constraints{std::format(Fmt.c_str(), std::forward<Args>(args)...)};
}
```

```
template<auto FromUnit, auto ToUnit, typename Rep>
concept ValuePreservingScaling =
  SaneScaling<FromUnit, ToUnit, Rep> &&
  (treat_as_floating_point<Rep> || integral_conversion_factor(FromUnit, ToUnit) ||
   unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
     unit_symbol(FromUnit), unit_symbol(ToUnit), type_name<Rep>()));
```

C++ exceptions

```
template<Reference auto R, RepresentationOf<get_quantity_spec(R)> Rep = double>
class quantity {
public:
  template<UnitOf<quantity_spec> ToU>
    requires ValuePreservingScaling<unit, ToU{}, rep>
  [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
  {
    return quantity<make_reference(quantity_spec, ToU{}), Rep>{*this};
  }
  // ...
};
```

C++ exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: substitution into constraint expression resulted in a non-constant expression
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   84 |      unit_symbol(FromUnit), unit_symbol(ToUnit), type_name<Rep>()));
      |      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: while checking the satisfaction of concept 'ValuePreservingScaling<mp_units::si::metre{{{}}}, mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}}, int>' requested here
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: while substituting template arguments into constraint expression here
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: while checking constraint satisfaction for template 'in<mp_units::si::metre{{{}}}, int>' required here
   25 |   quantity q3 = q.in(km);
      |                  ^~
note: in instantiation of function template specialization 'mp_units::quantity<mp_units::si::metre{{{}}}, int>::in<mp_units::si::kilo_<mp_units::si::metre>>' requested here
note: subexpression not valid in a constant expression
   66 |   throw unsatisfied_constraints{std::format(Fmt.c_str(), std::forward<Args>(args)...)};
      |   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: in call to 'unsatisfied<mp_units::basic_fixed_string<char, 79UL - 1>{"Scaling from \'{}\' to \'{}\' is not v[...]"}, std::string_view, std::string_view, std::string_view>(unit_symbol(mp_units::si:
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   84 |      unit_symbol(FromUnit), unit_symbol(ToUnit), type_name<Rep>()));
      |      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
error: no matching member function for call to 'in'
   25 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: failed template argument deduction
  249 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                          ^
```

P3068 Allowing exception throwing in constant-evaluation (C++26)

```
consteval auto hello(std::string_view input)
{
  if (input.empty())
    throw invalid_argument{"empty name provided"};  // BEFORE: compile-time error at throw expression site when reached
  return concat_into_a_fixed_string("hello ", input);
}
```

```
const auto a = hello("");                           // AFTER: compile-time error at call site "empty name provided"
const auto b = hello("Hana");                       // OK

try {
  const auto c = hello("");                         // AFTER: this exception is caught
} catch(const invalid_argument&) {
  // everything is fine
}
```

P3068 Allowing exception throwing in constant-evaluation (C++26)

```
consteval auto hello(std::string_view input)
{
  if (input.empty())
    throw invalid_argument{"empty name provided"};  // BEFORE: compile-time error at throw expression site when reached
  return concat_into_a_fixed_string("hello ", input);
}
```

```
const auto a = hello("");                          // AFTER: compile-time error at call site "empty name provided"
const auto b = hello("Hana");                      // OK

try {
  const auto c = hello("");                        // AFTER: this exception is caught
} catch(const invalid_argument&) {
  // everything is fine
}
```

Not SFINAE-friendly 😟

P3679 SFINAEable constexpr exceptions

- [temp.constr.atomic]

To determine if an atomic constraint is satisfied, the parameter mapping and template arguments are first substituted into its expression. If substitution results in an invalid type or expression in the immediate context of the atomic constraint ([temp.deduct.general]), the constraint is not satisfied. Otherwise, the lvalue-to-rvalue conversion is performed if necessary, and E shall be a constant expression of type bool. If an uncaught exception is thrown during constant evaluation ([expr.const]) of a constraint then the outermost constraint is evaluated as false. The constraint is satisfied if and only if evaluation of E results in true. If, at different points in the program, the satisfaction result is different for identical atomic constraints and template arguments, the program is ill-formed, no diagnostic required.

P3679 SFINAEable constexpr exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = kilo_<decltype(metre{{{}}})>]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^
note: because 'unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
              unit_symbol(mp_units::si::metre{{{}}}), unit_symbol(mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}}), type_name<int>())'
      throws an exception
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^
note: unhandled exception: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
   66 |   throw unsatisfied_constraints{constexpr_format(FMT_COMPILE(Fmt.c_str()), std::forward<Args>(args)...)};
      |   ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                          ^
1 error generated.
```

P3679 SFINAEable constexpr exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied [with ToU = kilo_<decltype(metre{{{}}})>]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to fal
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^
note: because 'unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
              unit_symbol(mp_units::si::metre{{{}}}), unit_symbol(mp_units::si::kilo_<mp_units::si::metre>{{{
      throws an exception
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^
note: unhandled exception: Scaling from 'm' to 'km' is not value-preserving for 'int' representation type
   66 |   throw unsatisfied_constraints{constexpr_format(FMT_COMPILE(Fmt.c_str()), std::forward<Args>(args)...
      |   ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                          ^
1 error generated.
```

https://godbolt.org/z/KhxTbnfKG

P3679 SFINAEable constexpr exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

```
error: no matching member function for call to 'in'
   10 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied with exception "Scaling from 'm' to 'km' is not value-preserving for 'int' representation type"
      [with ToU = kilo_<decltype(metre{{{}}})>]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^
note: because 'treat_as_floating_point<int>' evaluated to false
   73 |                                     (treat_as_floating_point<Rep> || (integral_conversion_factor(FromUnit, ToUnit)));
      |                                      ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}}, mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}})' evaluated to false
   73 |                                     (treat_as_floating_point<Rep> || (integral_conversion_factor(FromUnit, ToUnit)));
      |
note: and 'unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
               unit_symbol(mp_units::si::metre{{{}}}), unit_symbol(mp_units::si::kilo_<mp_units::si::metre>{{{{}, {}}}}), type_name<int>())'
      throws an exception
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                          ^
1 error generated.
```

P3679 SFINAEable constexpr exceptions

```
quantity q = 42 * m;
quantity q3 = q.in(km);
```

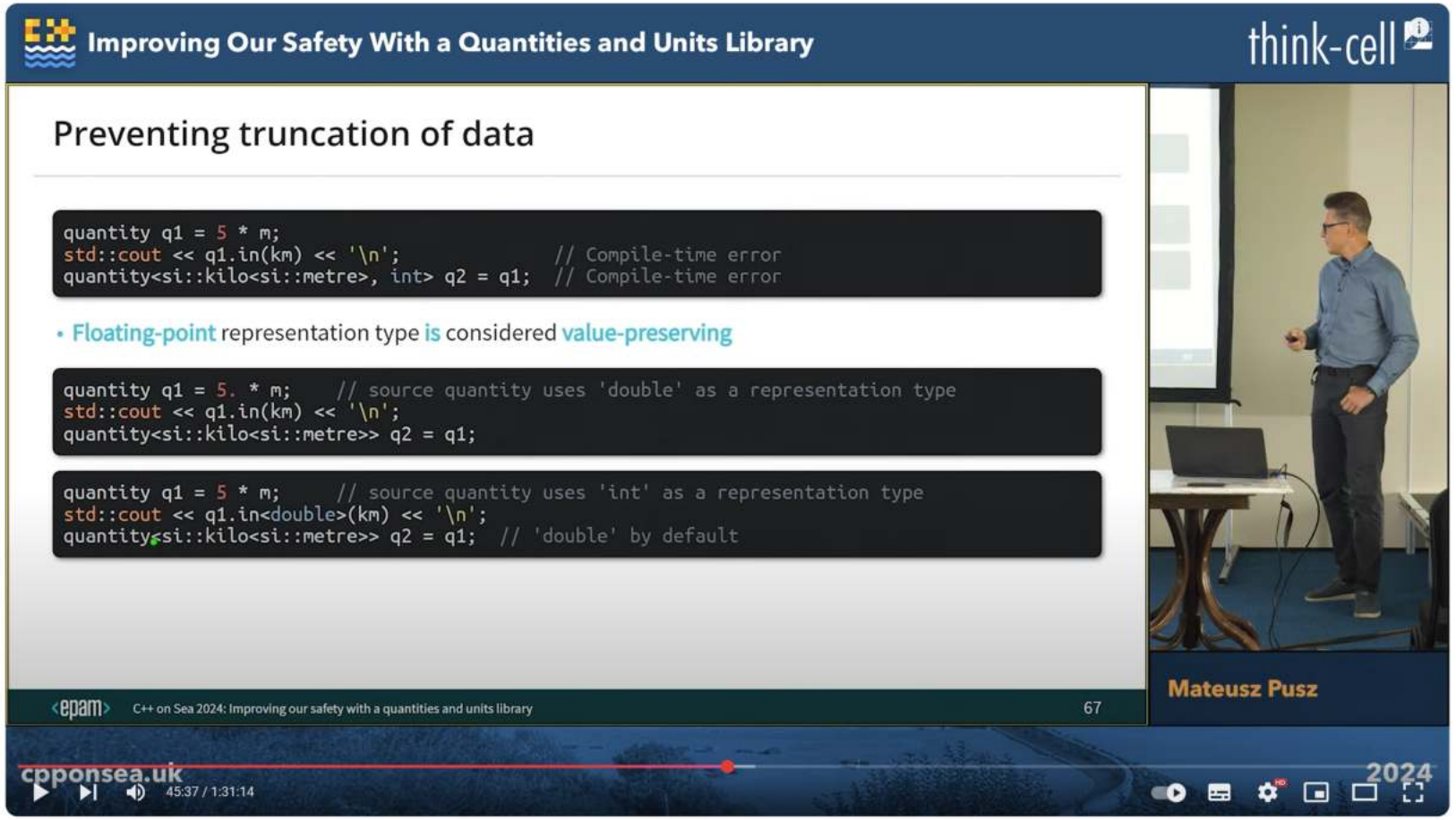
```
error: no matching member function for call to 'in'
   10 |   quantity q3 = q.in(km);
      |                 ~~^~
note: candidate template ignored: couldn't infer template argument 'ToRep'
  264 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: candidate template ignored: constraints not satisfied with exception "Scaling from 'm' to 'km' is not value-preserving for 'int' representation type"
      [with ToU = kilo_<decltype(metre{{{}}})>]
  244 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in(ToU) const
      |                                                          ^
note: because 'ValuePreservingScaling<unit, mp_units::si::kilo_<mp_units::si::metre>{}, rep>' evaluated to false
  248 |     requires ValuePreservingScaling<unit, ToU{}, rep>
      |              ^
note: because 'treat_as_floating_point<int>' evaluated to false
   73 |                                      (treat_as_floating_point<Rep> || (integral_conversion_factor(FromUni
      |                                       ^
note: and 'integral_conversion_factor(mp_units::si::metre{{{}}}, mp_units::si::kilo_<mp_units::si::metre>{{{{}, {
   73 |                                      (treat_as_floating_point<Rep> || (integral_conversion_factor(FromUni
      |
note: and 'unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
              unit_symbol(mp_units::si::metre{{{}}}), unit_symbol(mp_units::si::kilo_<mp_units::si::metre>{{{{},
      throws an exception
   83 |    unsatisfied<"Scaling from '{}' to '{}' is not value-preserving for '{}' representation type">(
      |    ^
note: candidate function template not viable: requires 0 arguments, but 1 was provided
  256 |   [[nodiscard]] constexpr QuantityOf<quantity_spec> auto in() const
      |                                                          ^
1 error generated.
```

P3378 constexpr exception types (C++26)

```
struct unsatisfied_constraints : std::logic_error {
  consteval explicit unsatisfied_constraints(const std::string& msg) : std::logic_error(msg)
  {
  }
};
```

IMPROVING SAFETY

Preventing truncation of data



Improving Our Safety With a Modern C++ Quantities and Units Library - Mateusz Pusz - C++ on Sea 2024

Preventing truncation of data

```
template<typename FromRep, typename ToRep, auto FromUnit, auto ToUnit>
concept ValuePreservingTo =
    Unit<decltype(FromUnit)> && Unit<decltype(ToUnit)> &&
    std::assignable_from<ToRep&, FromRep> &&
    (treat_as_floating_point<ToRep> ||
     (!treat_as_floating_point<std::remove_cvref_t<FromRep>> &&
      integral_conversion_factor(FromUnit, ToUnit) &&
      !overflows_non_zero_values<ToRep>(FromUnit, ToUnit)));
```

Klaus Iglberger: The Real Problem of C++

C++'s Superpower: Abstractions

```
class Second
{
 public:
   constexpr Second() = default;

   template< typename U > requires std::same_as<U,double>
   explicit constexpr Second( U value ) : value_{value} { /*possible checks*/ }

   template< typename U > requires std::same_as<U,double>
   constexpr Second& operator=( U value ) { /*Possible checks*/  value_ = value; return *this; }

   [[nodiscard]] constexpr double value() const { return value_; }

   // ...

 private:
   double value_{};
};
```

This prevents any kind of conversion!

Klaus Iglberger: The Real Problem of C++

C++'s Superpower: Abstractions

```
class Second
{
 public:
   constexpr Second() = default;

   template< typename U > requires no_narrowing_conversion<U,double>
   explicit constexpr Second( U value ) : value_{value} { /*possible checks*/ }

   template< typename U > requires no_narrowing_conversion<U,double>
   constexpr Second& operator=( U value ) { /*Possible checks*/  value_ = value; return *this; }

   [[nodiscard]] constexpr double value() const { return value_; }

   // ...

 private:
   double value_{};
};
```

YOU control the conversions!

`std::simd` (P1928)

(7.1) **29.10 Data-parallel types** **[simd]**

(7.1.1) **29.10.1 General** **[simd.general]**

8 The conversion from an arithmetic type `U` to a vectorizable type `T` is *value-preserving* if all possible values of `U` can be represented with type `T`.

P0870 A proposal for a type trait to detect narrowing conversions

```
template<auto FromUnit, typename FromRep, auto ToUnit, typename ToRep>
concept ValuePreservingConversion =
  SaneScaling<FromUnit, ToUnit, ToRep> &&
  std::assignable_from<ToRep&, FromRep> &&
  std::is_convertible_without_narrowing_v<FromRep, ToRep> &&
  (treat_as_floating_point<ToRep> ||
   (!treat_as_floating_point<FromRep> && integral_conversion_factor(FromUnit, ToUnit)) ||
   unsatisfied<"Scaling from '{}' as '{}' to '{}' as '{}' is not value-preserving">(
     unit_symbol(FromUnit), type_name<FromRep>(), unit_symbol(ToUnit), type_name<ToRep>()));
```

P0870 A proposal for a type trait to detect narrowing conversions

```
template<auto FromUnit, typename FromRep, auto ToUnit, typename ToRep>
concept ValuePreservingConversion =
  SaneScaling<FromUnit, ToUnit, ToRep> &&
  std::assignable_from<ToRep&, FromRep> &&
  std::is_convertible_without_narrowing_v<FromRep, ToRep> &&
  (treat_as_floating_point<ToRep> ||
   (!treat_as_floating_point<FromRep> && integral_conversion_factor(FromUnit, ToUnit)) ||
   unsatisfied<"Scaling from '{}' as '{}' to '{}' as '{}' is not value-preserving">(
     unit_symbol(FromUnit), type_name<FromRep>(), unit_symbol(ToUnit), type_name<ToRep>()));
```

```
quantity<si::metre, int> q1 = 3.14 * m;  // Compile-time error
quantity<si::metre> q2 = 42 * m;         // Compile-time error :-(
```

P2509 A proposal for a type trait to detect value-preserving conversions

```
template<auto FromUnit, typename FromRep, auto ToUnit, typename ToRep>
concept ValuePreservingConversion =
  SaneScaling<FromUnit, ToUnit, ToRep> &&
  std::assignable_from<ToRep&, FromRep> &&
  std::is_value_preserving_convertible_v<FromRep, ToRep> &&
  (treat_as_floating_point<ToRep> ||
   (!treat_as_floating_point<FromRep> && integral_conversion_factor(FromUnit, ToUnit)) ||
   unsatisfied<"Scaling from '{}' as '{}' to '{}' as '{}' is not value-preserving">(
     unit_symbol(FromUnit), type_name<FromRep>(), unit_symbol(ToUnit), type_name<ToRep>()));
```

P2509 A proposal for a type trait to detect value-preserving conversions

```
template<auto FromUnit, typename FromRep, auto ToUnit, typename ToRep>
concept ValuePreservingConversion =
  SaneScaling<FromUnit, ToUnit, ToRep> &&
  std::assignable_from<ToRep&, FromRep> &&
  std::is_value_preserving_convertible_v<FromRep, ToRep> &&
  (treat_as_floating_point<ToRep> ||
   (!treat_as_floating_point<FromRep> && integral_conversion_factor(FromUnit, ToUnit)) ||
   unsatisfied<"Scaling from '{}' as '{}' to '{}' as '{}' is not value-preserving">(
     unit_symbol(FromUnit), type_name<FromRep>(), unit_symbol(ToUnit), type_name<ToRep>()));
```

```
quantity<si::metre, int> q1 = 3.14 * m;  // Compile-time error
quantity<si::metre> q2 = 42 * m;         // OK (on most platforms)
```

P2509 A proposal for a type trait to detect value-preserving conversions

```
template<auto FromUnit, typename FromRep, auto ToUnit, typename ToRep>
concept ValuePreservingConversion =
  SaneScaling<FromUnit, ToUnit, ToRep> &&
  std::assignable_from<ToRep&, FromRep> &&
  std::is_value_preserving_convertible_v<FromRep, ToRep> &&
  (treat_as_floating_point<ToRep> ||
   (!treat_as_floating_point<FromRep> && integral_conversion_factor(FromUnit, ToUnit)) ||
   unsatisfied<"Scaling from '{}' as '{}' to '{}' as '{}' is not value-preserving">(
     unit_symbol(FromUnit), type_name<FromRep>(), unit_symbol(ToUnit), type_name<ToRep>()));
```

```
quantity<si::metre, int> q1 = 3.14 * m;  // Compile-time error
quantity<si::metre> q2 = 42 * m;         // OK (on most platforms)
```

Might be non portable!

A NEED FOR ADDITIONAL TEXT UTILITIES

A simple "legacy" C++ program

```
std::cout << "What is your name?\n";
std::string name;
std::cin >> name;
std::cout << "Hello " << name << "!\n";
```

A Modern C++ program

```
std::println("What is your name?");
std::string name;
std::cin >> name;
std::println("Hello {}!", name);
```

A Modern C++ program

```
std::println("What is your name?");
std::string name;
std::cin >> name;
std::println("Hello {}!", name);
```

NOT a Modern C++ program

```
std::println("What is your name?");
std::string name;
std::cin >> name;
std::println("Hello {}!", name);
```

NOT a Modern C++ program

```
std::println("What is your name?");
std::string name;
std::cin >> name;
std::println("Hello {}!", name);
```

- I/O streams
- Default construction followed by assignment
- No error handling

P1729 Text Parsing

WITH EXCEPTIONS

```
auto [name] = scn::prompt<std::string>("What is your name?\n", "{}")->values();
std::println("Hello {}!", name);
```

P1729 Text Parsing

WITH EXCEPTIONS

```
auto [name] = scn::prompt<std::string>("What is your name?\n", "{}")->values();
std::println("Hello {}!", name);
```

NO EXCEPTIONS

```
if (auto result = scn::prompt<std::string>("What is your name?\n", "{}")) {
  auto [name] = result->values();
  std::println("Hello {}!", name);
}
else
  std::println(stderr, "Error: {}", result.error().msg());
```

Internationalization

Isn't it embarrassing that in 2025 we do not have a standardized tools for Unicode or text internationalization?

IMPROVING DISTRIBUTION AND CONSUMPTION OF LIBRARIES

Question

Who likes writing CMake config files and installation scripts?

P2673 Common Description Format for C++ Libraries and Packages

BEFORE

- mp-unitsConfig.cmake

```
# mp-unitsConfig.cmake
include(CMakeFindDependencyMacro)

if(NOT MP_UNITS_API_FREESTANDING AND NOT MP_UNITS_API_STD_FORMAT)
    find_dependency(fmt)
endif()

if(MP_UNITS_API_CONTRACTS STREQUAL "GSL-LITE")
    find_dependency(gsl-lite)
elseif(MP_UNITS_API_CONTRACTS STREQUAL "MS-GSL")
    find_dependency(Microsoft.GSL)
endif()

include("${CMAKE_CURRENT_LIST_DIR}/mp-unitsTargets.cmake")
```

- CMakeLists.txt

```
# ...
configure_file("mp-unitsConfig.cmake" "." COPYONLY)
include(CMakePackageConfigHelpers)
write_basic_package_version_file(mp-unitsConfigVersion.cmake
                                 COMPATIBILITY SameMajorVersion)

install(EXPORT mp-unitsTargets
        DESTINATION ${CMAKE_INSTALL_LIBDIR}/cmake/mp-units
        NAMESPACE mp-units::)

install(FILES mp-unitsConfig.cmake
              ${CMAKE_CURRENT_BINARY_DIR}/mp-unitsConfigVersion.cmake
        DESTINATION ${CMAKE_INSTALL_LIBDIR}/cmake/mp-units)
```

P2673 Common Description Format for C++ Libraries and Packages

BEFORE

- mp-unitsConfig.cmake

```
# mp-unitsConfig.cmake
include(CMakeFindDependencyMacro)

if(NOT MP_UNITS_API_FREESTANDING AND NOT MP_UNITS_API_STD_FORMAT)
    find_dependency(fmt)
endif()

if(MP_UNITS_API_CONTRACTS STREQUAL "GSL-LITE")
    find_dependency(gsl-lite)
elseif(MP_UNITS_API_CONTRACTS STREQUAL "MS-GSL")
    find_dependency(Microsoft.GSL)
endif()

include("${CMAKE_CURRENT_LIST_DIR}/mp-unitsTargets.cmake")
```

- CMakeLists.txt

```
# ...
configure_file("mp-unitsConfig.cmake" "." COPYONLY)
include(CMakePackageConfigHelpers)
write_basic_package_version_file(mp-unitsConfigVersion.cmake
                                 COMPATIBILITY SameMajorVersion)

install(EXPORT mp-unitsTargets
        DESTINATION ${CMAKE_INSTALL_LIBDIR}/cmake/mp-units
        NAMESPACE mp-units::)

install(FILES mp-unitsConfig.cmake
              ${CMAKE_CURRENT_BINARY_DIR}/mp-unitsConfigVersion.cmake
        DESTINATION ${CMAKE_INSTALL_LIBDIR}/cmake/mp-units)
```

AFTER

- CMakeLists.txt

```
# ...
install(PACKAGE_INFO mp-units
        EXPORT mp-unitsTargets)
```

P2673 Common Description Format for C++ Libraries and Packages

BEFORE

```
class MPUnitsConan(ConanFile):
    # ...
    def package_info(self):
        compiler = self.settings.compiler
        # TODO remove the branch when Conan will learn to handle C++ modules
        if self.options.cxx_modules:
            # CMakeDeps does not generate C++ modules definitions for now
            # Skip the Conan-generated files and use the mp-unitsConfig.cmake bundled with mp-units
            self.cpp_info.set_property("cmake_find_mode", "none")
            self.cpp_info.builddirs = ["."]
        else:
            # handle contracts
            if self.options.contracts == "none":
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=0"
                )
            elif self.options.contracts == "gsl-lite":
                self.cpp_info.components["core"].requires.append("gsl-lite::gsl-lite")
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=2"
                )
            elif self.options.contracts == "ms-gsl":
                self.cpp_info.components["core"].requires.append("ms-gsl::ms-gsl")
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=3"
                )

            # handle API options
            self.cpp_info.components["core"].defines.append(
                "MP_UNITS_API_NO_CRTP=" + str(int(self.options.no_crtp == True))
            )
            self.cpp_info.components["core"].defines.append(
                "MP_UNITS_API_STD_FORMAT=" + str(int(self.options.std_format == True))
            )
            if not self.options.std_format:
                self.cpp_info.components["core"].requires.append("fmt::fmt")

            # handle hosted configuration
            if not self.options.freestanding:
                self.cpp_info.components["core"].defines.append("MP_UNITS_HOSTED=1")

            # handle import std
            if self.options.import_std:
                self.cpp_info.components["core"].defines.append("MP_UNITS_IMPORT_STD")
                if compiler == "clang" and Version(compiler.version) < 19:
                    self.cpp_info.components["core"].cxxflags.append(
                        "-Wno-deprecated-declarations"
                    )

            if compiler == "msvc":
                self.cpp_info.components["core"].cxxflags.append("/utf-8")

            self.cpp_info.components["systems"].requires = ["core"]
```

AFTER

```
class MPUnitsConan(ConanFile):
    # ...
    def package_info(self):
        from conan.cps import CPS
        self.cpp_info = CPS.load("cps/mp-units.cps").to_conan()
```

P2673 Common Description Format for C++ Libraries and Packages

BEFORE

```python
class MPUnitsConan(ConanFile):
    # ...
    def package_info(self):
        compiler = self.settings.compiler
        # TODO remove the branch when Conan will learn to handle C++ modules
        if self.options.cxx_modules:
            # CMakeDeps does not generate C++ modules definitions for now
            # Skip the Conan-generated files and use the mp-unitsConfig.cmake bundled with mp-units
            self.cpp_info.set_property("cmake_find_mode", "none")
            self.cpp_info.builddirs = ["."]
        else:
            # handle contracts
            if self.options.contracts == "none":
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=0"
                )
            elif self.options.contracts == "gsl-lite":
                self.cpp_info.components["core"].requires.append("gsl-lite::gsl-lite")
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=2"
                )
            elif self.options.contracts == "ms-gsl":
                self.cpp_info.components["core"].requires.append("ms-gsl::ms-gsl")
                self.cpp_info.components["core"].defines.append(
                    "MP_UNITS_API_CONTRACTS=3"
                )

            # handle API options
            self.cpp_info.components["core"].defines.append(
                "MP_UNITS_API_NO_CRTP=" + str(int(self.options.no_crtp == True))
            )
            self.cpp_info.components["core"].defines.append(
                "MP_UNITS_API_STD_FORMAT=" + str(int(self.options.std_format == True))
            )
            if not self.options.std_format:
                self.cpp_info.components["core"].requires.append("fmt::fmt")

            # handle hosted configuration
            if not self.options.freestanding:
                self.cpp_info.components["core"].defines.append("MP_UNITS_HOSTED=1")

            # handle import std
            if self.options.import_std:
                self.cpp_info.components["core"].defines.append("MP_UNITS_IMPORT_STD")
                if compiler == "clang" and Version(compiler.version) < 19:
                    self.cpp_info.components["core"].cxxflags.append(
                        "-Wno-deprecated-declarations"
                    )

            if compiler == "msvc":
                self.cpp_info.components["core"].cxxflags.append("/utf-8")

            self.cpp_info.components["systems"].requires = ["core"]
```

FUTURE

```python
class MPUnitsConan(ConanFile):
    # ...
```

P2673 Common Description Format for C++ Libraries and Packages

```
cmake --install . --config Release
```

BEFORE

```
$ tree lib
.
├── include
│   └── mp-units
│       └── ...
└── lib
    └── cmake
        └── mp-units
            ├── mp-unitsConfig.cmake
            ├── mp-unitsConfigVersion.cmake
            └── mp-unitsTargets.cmake
```

P2673 Common Description Format for C++ Libraries and Packages

```
cmake --install . --config Release
```

BEFORE

```
$ tree lib
.
├── include
│   └── mp-units
│       └── ...
└── lib
    └── cmake
        └── mp-units
            ├── mp-unitsConfig.cmake
            ├── mp-unitsConfigVersion.cmake
            └── mp-unitsTargets.cmake
```

AFTER

```
$ tree lib
.
├── include
│   └── mp-units
│       └── ...
└── lib
    └── cps
        └── mp-units
            └── mp-units.cps
```

P2673 Common Description Format for C++ Libraries and Packages

```
{
  "components" :
  {
    "core" :
    {
      "compile_definitions" :
      {
        "*" :
        {
          "MP_UNITS_API_NO_CRTP" : "1",
          "MP_UNITS_API_STD_FORMAT" : "0",
          "MP_UNITS_HOSTED" : "1"
        }
      },
      "compile_features" : [ "c++26" ],
      "compile_flags" : [ "-Wno-unused-result" ],
      "requires" : [ "fmt:fmt", ":mp-units-contracts" ],
      "type" : "interface"
    },
    "mp-units" :
    {
      "compile_features" : [ "c++26" ],
      "requires" : [ ":core", ":systems" ],
      "type" : "interface"
    },
```

```
    "mp-units-contracts" :
    {
      "compile_definitions" :
      {
        "*" :
        {
          "MP_UNITS_API_CONTRACTS" : "0"
        }
      },
      "type" : "interface"
    },
    "systems" :
    {
      "compile_features" : [ "c++26" ],
      "requires" : [ ":core" ],
      "type" : "interface"
    }
  },
  "cps_path" : "@prefix@/lib/cps/mp-units",
  "cps_version" : "0.12.0",
  "name" : "mp-units",
  "requires" :
  {
    "fmt" : null
  }
}
```

SUMMARY

C++ is awesome!

C++ is probably the best programming language to implement fast generic libraries.

C++ is awesome!

C++ is probably the best programming language to implement fast generic libraries.

- Zero-cost abstractions thanks to C++ templates
  - runtime performance
  - memory usage

C++ is awesome!

C++ is probably the best programming language to implement fast generic libraries.

- Zero-cost abstractions thanks to C++ templates
  - runtime performance
  - memory usage
- Constraints and concepts improve interfaces

C++ is awesome!

C++ is probably the best programming language to implement fast generic libraries.

- Zero-cost abstractions thanks to C++ templates
  - runtime performance
  - memory usage
- Constraints and concepts improve interfaces
- Contracts will improve safety and interfaces

C++ is awesome!

C++ is probably the best programming language to implement fast generic libraries.

- Zero-cost abstractions thanks to C++ templates
  - runtime performance
  - memory usage
- Constraints and concepts improve interfaces
- Contracts will improve safety and interfaces
- Reflection will change the C++ world

Please help!

Some features are still missing to make C++ even better.

Please help!

Some features are still missing to make C++ even better.

If you are an expert or just care about a specific subject, please help!


CAUTION
Programming
is addictive
(and too much fun)