# std::units

LIBRARY IN A WEEK 2019

Mateusz Pusz

May 9, 2019

# What is possible in 3 days of C++Now?

# What is possible in 3 days of C++Now?

1. Delivered a Physical Units talk on Tuesday
   - plus one more on Monday
   - slides: https://github.com/train-it-eu/conf-slides

# What is possible in 3 days of C++Now?

1. Delivered a Physical Units talk on Tuesday
   - plus one more on Monday
   - slides: https://github.com/train-it-eu/conf-slides
2. Got a lot of words of encouragement
   - people confirmed that my design choices are right and that they solve their problems with `Boost.Units`

# What is possible in 3 days of C++Now?

1. Delivered a Physical Units talk on Tuesday
   - plus one more on Monday
   - slides: https://github.com/train-it-eu/conf-slides
2. Got a lot of words of encouragement
   - people confirmed that my design choices are right and that they solve their problems with `Boost.Units`
3. Got some feedback regarding the implementation
   - some of it is implemented already

# What is possible in 3 days of C++Now?

1. Delivered a Physical Units talk on Tuesday
   - plus one more on Monday
   - slides: https://github.com/train-it-eu/conf-slides
2. Got a lot of words of encouragement
   - people confirmed that my design choices are right and that they solve their problems with `Boost.Units`
3. Got some feedback regarding the implementation
   - some of it is implemented already
4. Progressed a lot with the implementation of the library

# MY UNITS LIBRARY (WIP!!!)

HTTPS://GITHUB.COM/MPUSZ/UNITS

# Requirements

- The best possible user experience
  - compiler errors
  - debugging

# Requirements

- The best possible user experience
  - compiler errors
  - debugging
- Safety and performance
  - strong types
  - template metaprogramming
  - `constexpr` all the things

# Requirements

- The best possible user experience
  - compiler errors
  - debugging
- Safety and performance
  - strong types
  - template metaprogramming
  - `constexpr` all the things
- No macros in the user interface
- No external dependencies
- Easy extensibility

# Requirements

- The best possible user experience
  - compiler errors
  - debugging
- Safety and performance
  - strong types
  - template metaprogramming
  - `constexpr` all the things
- No macros in the user interface
- No external dependencies
- Easy extensibility
- Possibility to be standardized as a freestanding part of the C++ Standard Library

# A simple example

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

# A simple example

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
const auto kmph = avg_speed(220._km, 2._h);
std::cout << kmph.count() << " km/h\n";
```

```
const auto mph = avg_speed(140._mi, 2._h);
std::cout << mph.count() << " mph\n";
```

# A simple example

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
const auto kmph = avg_speed(220._km, 2._h);
std::cout << kmph.count() << " km/h\n";
```

```
const auto mph = avg_speed(140._mi, 2._h);
std::cout << mph.count() << " mph\n";
```

No intermediate conversion to SI base units and back

# A simple example

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
const auto kmph = avg_speed(length<kilometer>(220), time<hour>(2));
std::cout << kmph.count() << " km/h\n";
```

```
const auto mph = avg_speed(length<mile>(140), time<hour>(2));
std::cout << mph.count() << " mph\n";
```

# A simple example

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
length<kilometer> d(220);
time<hour> t(2);
const auto kmph = avg_speed(d, t);
std::cout << kmph.count() << " km/h\n";
```

```
length<mile> d(140);
time<hour> t(2);
const auto mph = avg_speed(d, t);
std::cout << mph.count() << " mph\n";
```

# User experience: Debugging

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

# User experience: Debugging

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
Breakpoint 1, avg_speed<units::quantity<units::dimension_length, units::kilometer, double>,
                        units::quantity<units::dimension_time, units::hour, double> > (d=..., t=...)
13        return d / t;
```

# User experience: Debugging

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
Breakpoint 1, avg_speed<units::quantity<units::dimension_length, units::kilometer, double>,
                        units::quantity<units::dimension_time, units::hour, double> > (d=..., t=...)
13        return d / t;

(gdb) ptype d
type = class units::quantity<units::dimension_length, units::kilometer, double>
       [with D = units::dimension_length, U = units::kilometer, Rep = double] {
...
```

# User experience: Debugging

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d / t;
}
```

```
Breakpoint 1, avg_speed<units::quantity<units::dimension_length, units::kilometer, double>,
                        units::quantity<units::dimension_time, units::hour, double> > (d=..., t=...)
13        return d / t;

(gdb) ptype d
type = class units::quantity<units::dimension_length, units::kilometer, double>
       [with D = units::dimension_length, U = units::kilometer, Rep = double] {
...
```

Achieved via **`units::upcasting_traits`** engine

# User experience: Bad equation

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d * t;
}
```

# User experience: Bad equation

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d * t;
}
```

```
velocity.cpp:23:37:   required from here
velocity.cpp:13:16: error: placeholder constraints not satisfied
   13 |     return d * t;
      |                ^
In file included from velocity.cpp:1:
include/units/velocity.h:34:16: note: within ‘template<class T> concept const bool units::Velocity<T>
    [with T = units::quantity<units::dimension<units::exp<units::base_dim_length, 1>, units::exp<units::base_dim_time, 1> >,
                              units::unit<units::dimension<units::exp<units::base_dim_length, 1>,
                                                           units::exp<units::base_dim_time, 1> >,
                                          std::ratio<3600000, 1> >,
                              double>]’
   34 |   concept bool Velocity = Quantity<T> && std::Same<typename T::dimension, dimension_velocity>;
      |                ^~~~~~~~
...
```

# User experience: Bad equation

```
using namespace units;

constexpr Velocity auto avg_speed(Length auto d, Time auto t)
{
  return d * t;
}
```

```
...
In file included from include/experimental/ranges/concepts:12,
include/stl2/detail/concepts/core.hpp:37:15: note: within ‘template<class T, class U> concept const bool std::Same<T, U>
    [with T = units::dimension<units::exp<units::base_dim_length, 1>, units::exp<units::base_dim_time, 1> >;
          U = units::dimension_velocity]’
   37 |  META_CONCEPT Same = meta::Same<T, U> && meta::Same<U, T>;
      |               ^~~~
In file included include/experimental/ranges/concepts:12,
include/meta/meta_fwd.hpp:224:18: note: within ‘template<class T, class U> concept const bool meta::Same<T, U>
    [with T = units::dimension<units::exp<units::base_dim_length, 1>, units::exp<units::base_dim_time, 1> >;
          U = units::dimension_velocity]’
  224 |     META_CONCEPT Same =
      |                  ^~~~
include/meta/meta_fwd.hpp:224:18: note: ‘meta::detail::bool_’ evaluated to false
include/meta/meta_fwd.hpp:224:18: note: within ‘template<class T, class U> concept const bool meta::Same<T, U>
    [with T = units::dimension_velocity;
          U = units::dimension<units::exp<units::base_dim_length, 1>, units::exp<units::base_dim_time, 1> >]’
include/meta/meta_fwd.hpp:224:18: note: ‘meta::detail::bool_’ evaluated to false
```

# Implementation feedback

```
template<class Rep1, class Rep2, class Period>
duration<typename std::common_type<Rep1,Rep2>::type, Period>
constexpr operator*(const Rep1& s, const duration<Rep2,Period>& d);
```

# Implementation feedback

```
template<class Rep1, class Rep2, class Period>
duration<typename std::common_type<Rep1,Rep2>::type, Period>
constexpr operator*(const Rep1& s, const duration<Rep2,Period>& d);
```

Some scientific numeric types return different types for their arithmetic operators

# Implementation feedback

```
template<class Rep1, class Rep2, class Period>
duration<typename std::common_type<Rep1,Rep2>::type, Period>
constexpr operator*(const Rep1& s, const duration<Rep2,Period>& d);
```

```
template<Scalar Rep1, Dimension D, Unit U, Scalar Rep2>
[[nodiscard]] constexpr Quantity operator*(const Rep1& v, const quantity<D, U, Rep2>& q)
{
  using common_rep = decltype(v * q.count());
  using ret = quantity<D, U, common_rep>;
  return ret(v * ret(q).count());
}
```

# Implementation update

# Implementation update

- Got rid of *`std::common_type_t`* in most places

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*
- Experiments with *classes as non-type template parameters*

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*
- Experiments with *classes as non-type template parameters*
- Fighting with *gcc-9 bugs*
  - classes as non-type template parameters
  - various issues with the Concepts TS support

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*
- Experiments with *classes as non-type template parameters*
- Fighting with *gcc-9 bugs*
  - classes as non-type template parameters
  - various issues with the Concepts TS support
- Pushed ***`units_compare`*** *repo* to GitHub

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*
- Experiments with *classes as non-type template parameters*
- Fighting with *gcc-9 bugs*
  - classes as non-type template parameters
  - various issues with the Concepts TS support
- Pushed ***`units_compare`*** *repo* to GitHub
- Lots of *code cleanup*

# Implementation update

- Got rid of ***`std::common_type_t`*** in most places
- Introduced ***`Scalar`*** *concept*
- Experiments with *classes as non-type template parameters*
- Fighting with *gcc-9 bugs*
  - classes as non-type template parameters
  - various issues with the Concepts TS support
- Pushed ***`units_compare`*** *repo* to GitHub
- Lots of *code cleanup*
- *CI and packaging* improvements

# Let's join forces!

We really need physical units and dimensional analysis support in the C++ Standard Library

# Let's join forces!

We really need physical units and dimensional analysis support in the C++ Standard Library

WHY TO JOIN?

- C++ community and industry really need it
- Great opportunity to learn C++20
- An interesting and hard challenge to solve ;-)

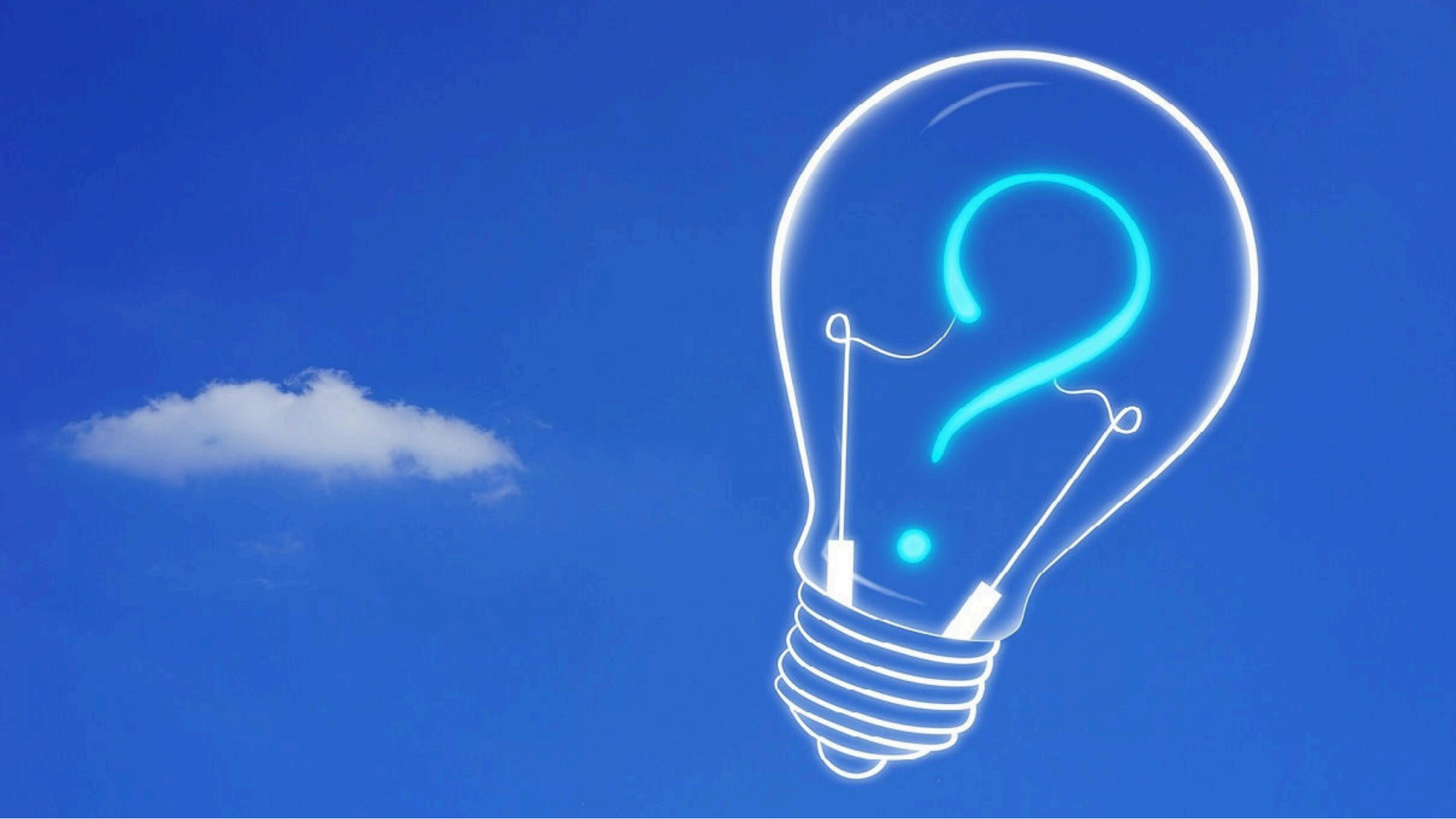
# Let's join forces!

We really need physical units and dimensional analysis support in the C++ Standard Library

WHY TO JOIN?

- C++ community and industry really need it
- Great opportunity to learn C++20
- An interesting and hard challenge to solve ;-)

Please, help...



# CAUTION
# Programming
# is addictive
# (and too much fun)