

# Notes for PDF users

If you're reading these slides as a PDF, know that this is **not their intended format**. You shouldn't expect to be able to read the slides and understand the presentation. The biggest missing piece is the **speaker's notes**. Without them, the slides will **not all make sense**, and you could easily **misinterpret** some of them.

Additionally: some individual slides have rendered incorrectly, such as “Embedded Friendliness” and “Feature Inspirations”.

To view with the speaker's notes, you can view the [online slides](#), and press **s** to invoke the notes. You can also go to the repository itself, [chiphogg/cppcon-2023-au-units](https://github.com/chiphogg/cppcon-2023-au-units), either to host the slides locally, or to see how the talk is implemented or was made.

+ 23

# The Au Library: Handling Physical Units Safely, Quickly, and Broadly

CHIP HOGG



**Cppcon**  
The C++ Conference

20  
23



October 01 - 06

# Units library: basic concept

```
1 // No units library:  
2 double distance_m;  
3  
4  
5
```

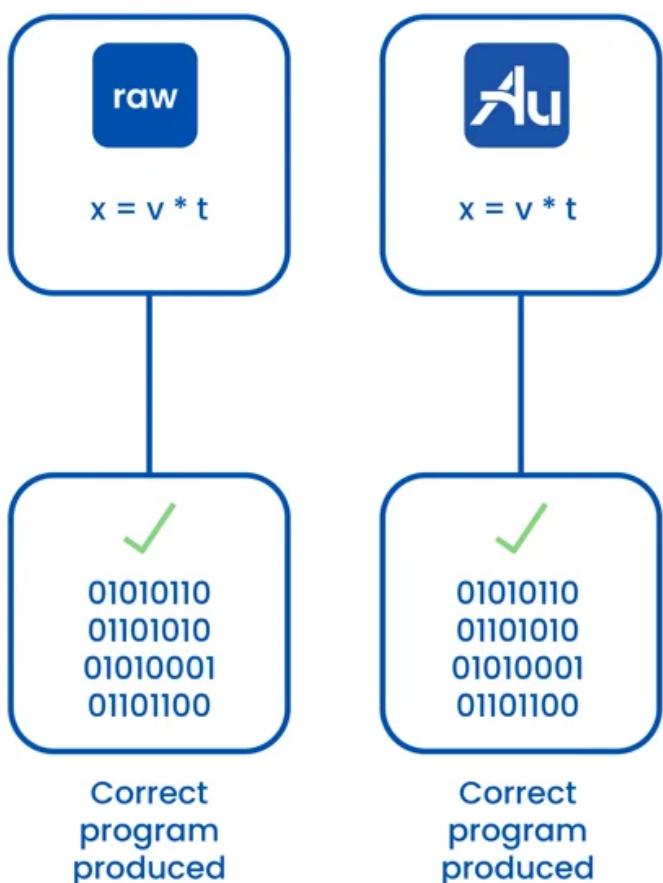
# Units library: basic concept

```
1 // No units library:  
2 double distance_m;  
3  
4 // With units library:  
5 Quantity<Meters, double> distance;
```

# Units library: basic concept

```
1 // No units library:  
2 double distance_m;  
3  
4 // With units library:  
5 Quantity<Meters, double> distance;
```

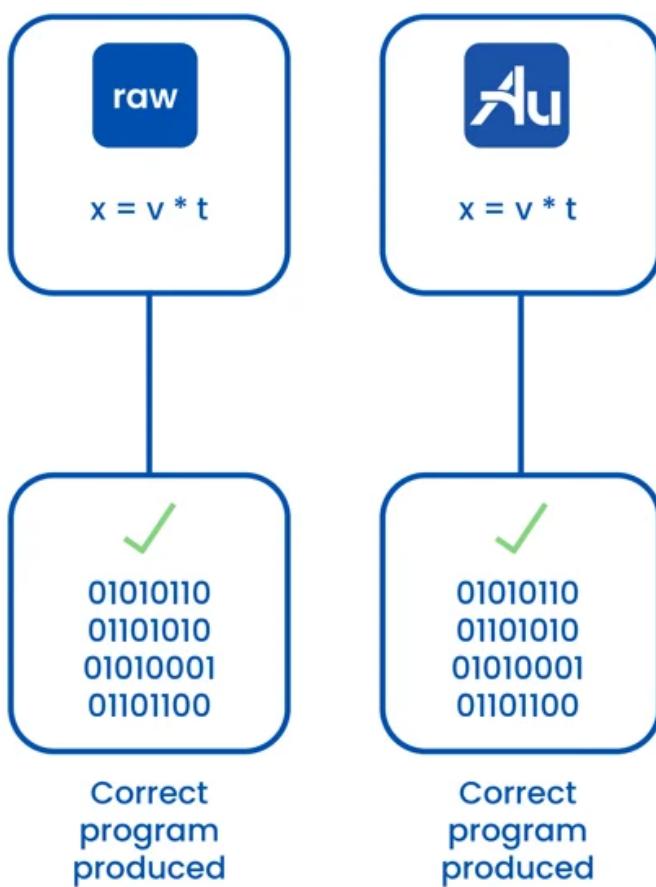
## Correct code



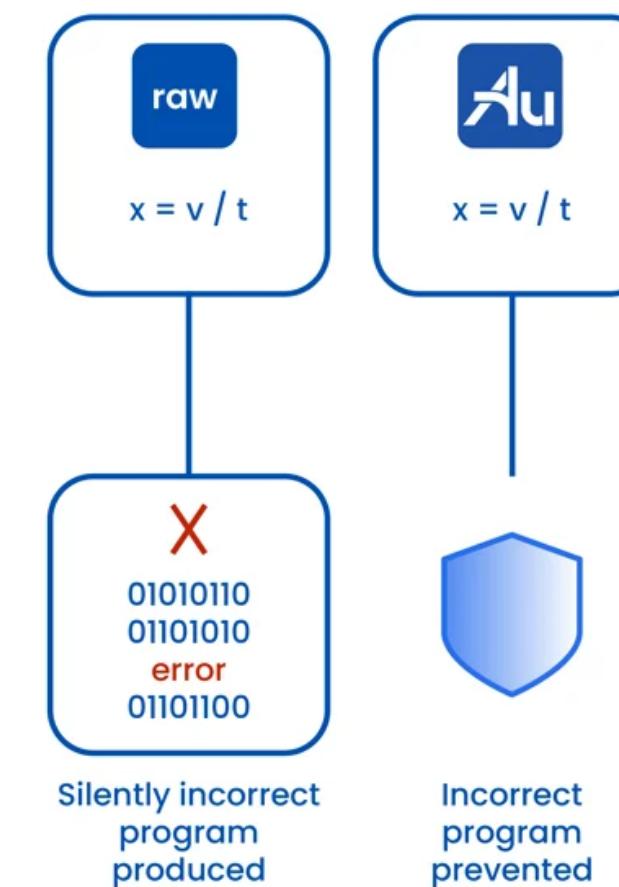
# Units library: basic concept

```
1 // No units library:  
2 double distance_m;  
3  
4 // With units library:  
5 Quantity<Meters, double> distance;
```

## Correct code



## Coding mistake



# Aurora Innovation



# Aurora Innovation

On a mission to deliver the  
benefits of self-driving—safely,  
quickly, and broadly



# Flashback: CppCon 2021

The slide features a dark blue background with yellow and white accents. At the top right is the CppCon 2021 logo. On the left, there's a large yellow plus sign icon. In the center, the title is displayed in yellow and white text. Below the title, the speaker's name is shown in yellow, followed by the year 2021 and the date October 24-29. A small image of the speaker, Chip Hogg, is shown on the right. The bottom right corner contains the title and speaker name again.

+

Cppcon 2021 | October 24-29

**Units Libraries and Autonomous Vehicles:  
Lessons from the Trenches**

**CHIP HOGG**

**2021** | October 24-29

**Chip Hogg**

---

**Units Libraries and  
Autonomous Vehicles:  
Lessons from the Trenches**

# A taste of Au

# Example: time to goal

# Example: time to goal

## No units library

```
const double dist_to_goal_m = 30.0;
const double speed_mph = 25.0;
const double speed_mps =
    speed_mph * MPS_PER_MPH;
const double time_to_goal_s =
    dist_to_goal_m / speed_mps;
```

# Example: time to goal

## No units library

```
const double dist_to_goal_m = 30.0;
const double speed_mph = 25.0;
const double speed_mps =
    speed_mph * MPS_PER_MPH;
const double time_to_goal_s =
    dist_to_goal_m / speed_mps;
```

```
// unit_conversions.hh
constexpr auto CM_PER_MI =
    2.54 * 12.0 * 5280.0;
constexpr auto M_PER_MI =
    CM_PER_MI / 100.0;
constexpr auto S_PER_H = 3600.0;

constexpr auto MPS_PER_MPH =
    M_PER_MI / S_PER_H;
```

# Example: time to goal

## No units library

```
const double dist_to_goal_m = 30.0;
const double speed_mph = 25.0;
const double speed_mps =
    speed_mph * MPS_PER_MP;
const double time_to_goal_s =
    dist_to_goal_m / speed_mps;
```

```
// unit_conversions.hh
constexpr auto CM_PER_MI =
    2.54 * 12.0 * 5280.0;
constexpr auto M_PER_MI =
    CM_PER_MI / 100.0;
constexpr auto S_PER_H = 3600.0;

constexpr auto MPS_PER_MP =
    M_PER_MI / S_PER_H;
```

## Au

```
const auto dist_to_goal = meters(30.0);
const auto speed = (miles / hour)(25.0);

const auto time_to_goal =
    (dist_to_goal / speed).as(seconds);
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     using NS_PER_TICK = std::ratio<1'000'000'000, CPU_CLOCK_HZ>;
3     return std::chrono::nanoseconds{
4         num_cpu_ticks * NS_PER_TICK::num / NS_PER_TICK::den
5     };
6 }
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     using NS_PER_TICK = std::ratio<1'000'000'000, CPU_CLOCK_HZ>;
3     return std::chrono::nanoseconds{
4         num_cpu_ticks * NS_PER_TICK::num / NS_PER_TICK::den
5     };
6 }
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     constexpr auto cpu_ticks = inverse(hertz * mag<CPU_CLOCK_HZ>());
3     return cpu_ticks(num_cpu_ticks).as(nano(seconds));
4 }
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     using NS_PER_TICK = std::ratio<1'000'000'000, CPU_CLOCK_HZ>;
3     return std::chrono::nanoseconds{
4         num_cpu_ticks * NS_PER_TICK::num / NS_PER_TICK::den
5     };
6 }
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     constexpr auto cpu_ticks = inverse(hertz * mag<CPU_CLOCK_HZ>());
3     return cpu_ticks(num_cpu_ticks).as(nano(seconds));
4 }
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     using NS_PER_TICK = std::ratio<1'000'000'000, CPU_CLOCK_HZ>;
3     return std::chrono::nanoseconds{
4         num_cpu_ticks * NS_PER_TICK::num / NS_PER_TICK::den
5     };
6 }
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     constexpr auto cpu_ticks = inverse(hertz * mag<CPU_CLOCK_HZ>());
3     return cpu_ticks(num_cpu_ticks).coerce_as(nano(seconds));
4 }
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds
```

```
1 std::chrono::nanoseconds
2     using NS_PER_TICK
3     return std::chrono::nanoseconds(
4         num_cpu_ticks
5     );
6 }
```

```
1 no_au::elapsed_time(unsigned long):
2     lea    rax, [rdi+rdi*4]
3     shr    rax
4     ret
5 au::elapsed_time(unsigned long):
6     lea    rax, [rdi+rdi*4]
7     shr    rax
8     ret
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     constexpr auto cpu_ticks = inverse(hertz * mag<CPU_CLOCK_HZ>());
3     return cpu_ticks(num_cpu_ticks).coerce_as(nano(seconds));
4 }
```

# Example: “CPU ticks” time units

```
1 constexpr uint64_t CPU_CLOCK_HZ = 400'000'000;
2
3 // API to implement:
4 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks);
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     using NS_PER_TICK = std::ratio<1'000'000'000, CPU_CLOCK_HZ>;
3     return std::chrono::nanoseconds{
4         num_cpu_ticks * NS_PER_TICK::num / NS_PER_TICK::den
5     };
6 }
```

```
1 std::chrono::nanoseconds elapsed_time(uint64_t num_cpu_ticks) {
2     constexpr auto cpu_ticks = inverse(hertz * mag<CPU_CLOCK_HZ>());
3     return cpu_ticks(num_cpu_ticks).coerce_as(nano(seconds));
4 }
```

# Au: Interfaces and Idioms

# Au: Interfaces and Idioms

```
Quantity<Meters, T>
```

```
T value_;
```

# Au: Interfaces and Idioms

```
T x = 1.75;
```

```
Quantity<Meters, T>
```

```
T value_;
```

# Au: Interfaces and Idioms

```
T x = 1.75;
```

```
Quantity<Meters, T>
```

```
-----  
meters(x) T value_ = 1.75;
```

# Au: Interfaces and Idioms

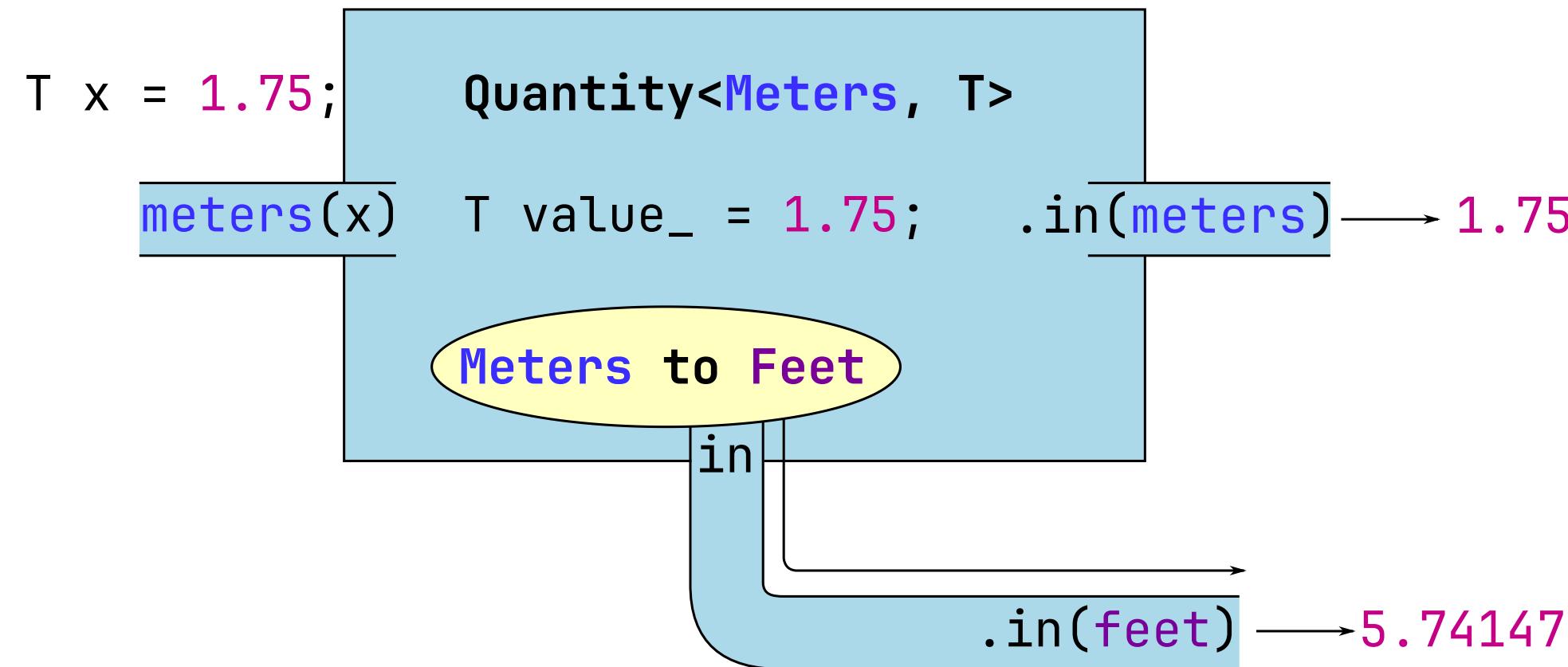
T x = 1.75;

Quantity<Meters, T>

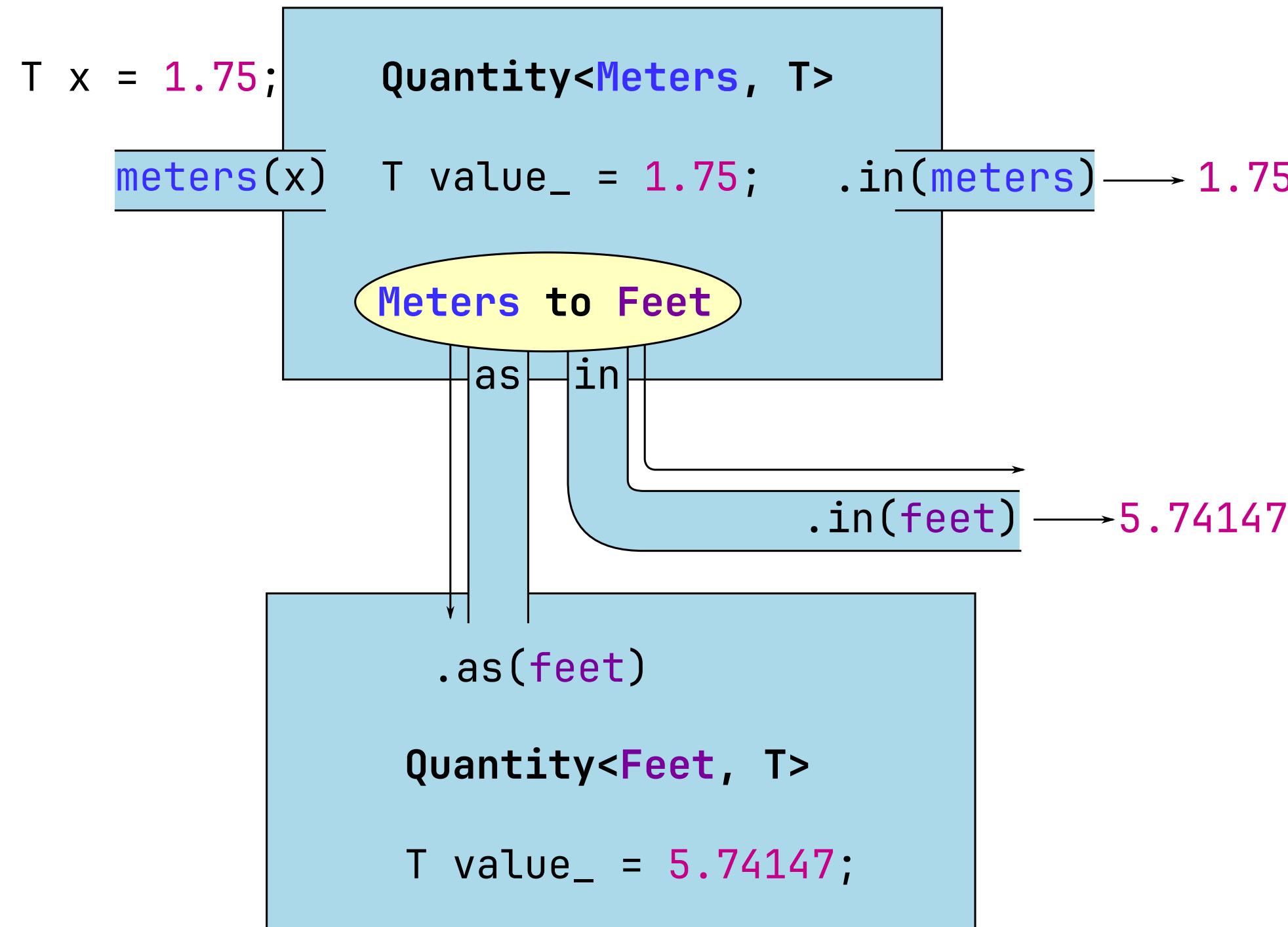
meters(x)

T value\_ = 1.75; .in(meters) → 1.75

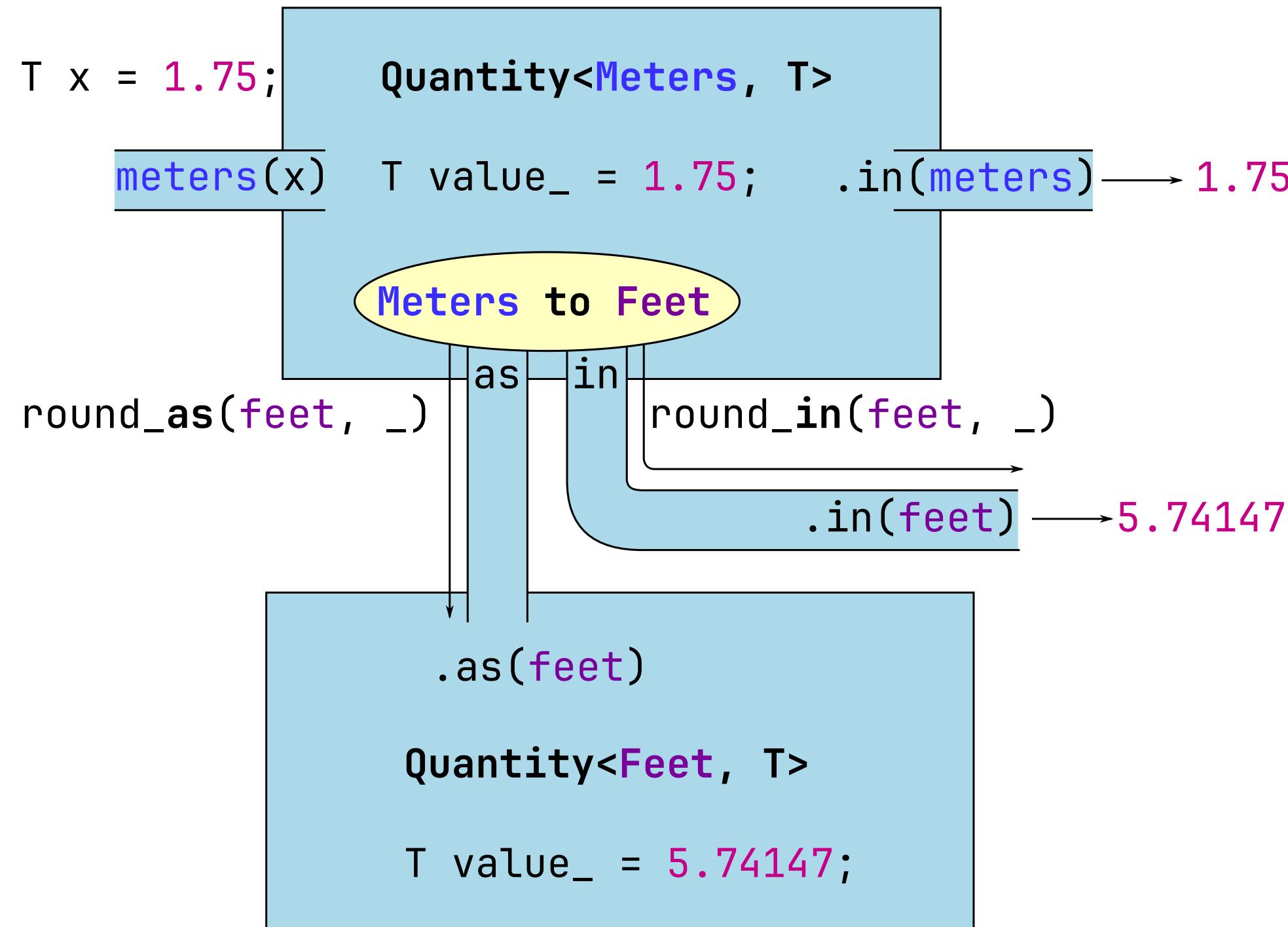
# Au: Interfaces and Idioms



# Au: Interfaces and Idioms



# Au: Interfaces and Idioms



# Au: learning more

# Au: learning more

The screenshot shows a GitHub page for 'The Au units library' with the URL <https://aurora-opensource.github.io/au/main/tutorial/>. The page title is 'Au 101: Quantity Makers'. It includes a search bar, a GitHub icon with '0.3.3', '109 stars', and '7 forks', and a navigation bar with 'The Au units library' and 'main'.

**Au 101: Quantity Makers**

This tutorial gives a gentle introduction to the Au library.

- **Time:** TBD.
- **Prerequisites:** Experience writing C++ code.
- **You will learn:**
  - The concept and importance of "unit safety".
  - How to store a numeric value in a *quantity*.
  - How to retrieve the stored numeric value.
  - Some basic operations you can perform with a quantity.

Status quo: no units library

Suppose you have a variable that represents a *physical quantity*. That variable has some *value*, but that value is meaningless unless you also know the *unit of measurement*. We usually indicate the unit with a suffix on the variable name. Here's a concrete example:

```
const double track_length_m = 100.0;
//           Unit suffix--^--  ^^^^^^--Value

const double best_time_s = 10.34;
//           Unit suffix--^--  ^^^^^^--Value
```

The first value is `100.0`. Since there's no such thing as a "length of 100", we add a `_m` suffix on the end of our variable name to make it clear that the value is the length *in meters*. We take a similar approach for our time *in seconds*.

This strategy works, in the sense that it can prevent unit errors, but it's labor intensive and error prone. The naming suffixes provide hints, but enforcement is basically on the honor system. Consider a function we might want to call:

```
double average_speed_mps(double length_m, double time_s);
```

With the above variables, our callsite might look like this:

```
const auto speed_mps = average_speed_mps(track_length_m, best_time_s);
```

It's time to consider a very important property:

**Definition**

**Unit correctness:** a program is *unit-correct* when every variable associated with physical units is used consistently with those units.

So: is this unit-correct? Yes:

- `track_length_m` gets passed as the parameter `length_m`: meters to meters ✓
- `best_time_s` gets passed as the parameter `time_s`: seconds to seconds ✓

<https://aurora-opensource.github.io/au/main/tutorial/>

# Au: learning more



## Au 101: Quantity Makers

This tutorial gives a gentle introduction to the Au library.

- **Time:** TBD.
- **Prerequisites:** Experience writing C++ code.
- **You will learn:**
  - The concept and importance of “unit safety”.
  - How to store a numeric value in a *quantity*.
  - How to retrieve the stored numeric value.
  - Some basic operations you can perform with a quantity.

### Status quo: no units library

Suppose you have a variable that represents a *physical quantity*. That variable has some *value*, but that value is meaningless unless you also know the *unit of measurement*. We usually indicate the unit with a suffix on the variable name. Here’s a concrete example:

```
const double track_length_m = 100.0;
//           Unit suffix--^--  ^^^^^^--Value

const double best_time_s = 10.34;
//           Unit suffix--^--  ^^^^^^--Value
```

The first value is `100.0`. Since there’s no such thing as a “length of 100”, we add a `_m` suffix on the end of our variable name to make it clear that the value is the length *in meters*. We take a similar approach for our time *in seconds*.

This strategy works, in the sense that it can prevent unit errors, but it’s labor intensive and error prone. The naming suffixes provide hints, but enforcement is basically on the honor system. Consider a function we might want to call:

```
double average_speed_mps(double length_m, double time_s);
```

With the above variables, our callsite might look like this:

```
const auto speed_mps = average_speed_mps(track_length_m, best_time_s);
```

It’s time to consider a very important property:

### Definition

**Unit correctness:** a program is *unit-correct* when every variable associated with physical units is used consistently with those units.

So: is this unit-correct? Yes:

- `track_length_m` gets passed as the parameter `length_m:meters` ✓
- `best_time_s` gets passed as the parameter `time_s:seconds` ✓

<https://aurora-opensource.github.io/au/main/tutorial/>



## Troubleshooting Guide

This page is a guide to the most commonly encountered types of error, what they mean, and how to fix them.

The intended use case is to help you interpret an *actual error in your code*, at the point where you encounter it. To use this page, copy some relevant snippets from your compiler error, and then search the text of this page using your browser’s Find function.

### Tip

To improve your chances of finding what you’re looking for, we include full compiler errors from both gcc and clang, inline with the text. Naturally, this makes this page very long, so it’s not meant to be read straight through. Stick with your browser’s Find function.

Each section below lists one category of compiler error you might encounter when using the library. It explains what it means and how to solve it, and gives specific snippets of erroneous code, along with the compiler errors that each would yield.

### Private constructor

**Meaning:** This means you passed a raw numeric value to an interface that expected a *Quantity*. It’s the “classic” error the units library aims to prevent.

**Solution:** Call the appropriate *Quantity maker*: instead of passing `x`, pass `meters(x)`, `(kilo(meters) / hour)(x)`, etc.

#### A note on quantity makers vs. constructors

### Example

#### Code

##### Broken      Fixed

```
void set_timeout(Quantity<Seconds> dt);

// A (BROKEN): passing raw number where duration expected.
set_timeout(0.5);

// B (BROKEN): calling Quantity constructor directly.
constexpr Quantity<Meters> length(5.5);
```

#### Compiler error (clang 14)

```
au/error_examples.cc:33:17: error: calling a private constructor of class 'au::Quantity<au::Seconds, double>'
    set_timeout(0.5);
               ^
./au/quantity.h:41:15: note: declared private here
constexpr Quantity<Rep value> : value_{value} {}
```

```
au/error_examples.cc:36:33: error: calling a private constructor of class 'au::Quantity<au::Meters, double>'
    constexpr Quantity<Meters> length(5.5);
               ^
./au/quantity.h:41:15: note: declared private here
constexpr Quantity<Rep value> : value_{value} {}
```

#### Compiler error (clang 11)

```
au/error_examples.cc:33:17: error: calling a private constructor of class 'au::Quantity<au::Seconds, double>'
    set_timeout(0.5);
               ^
```

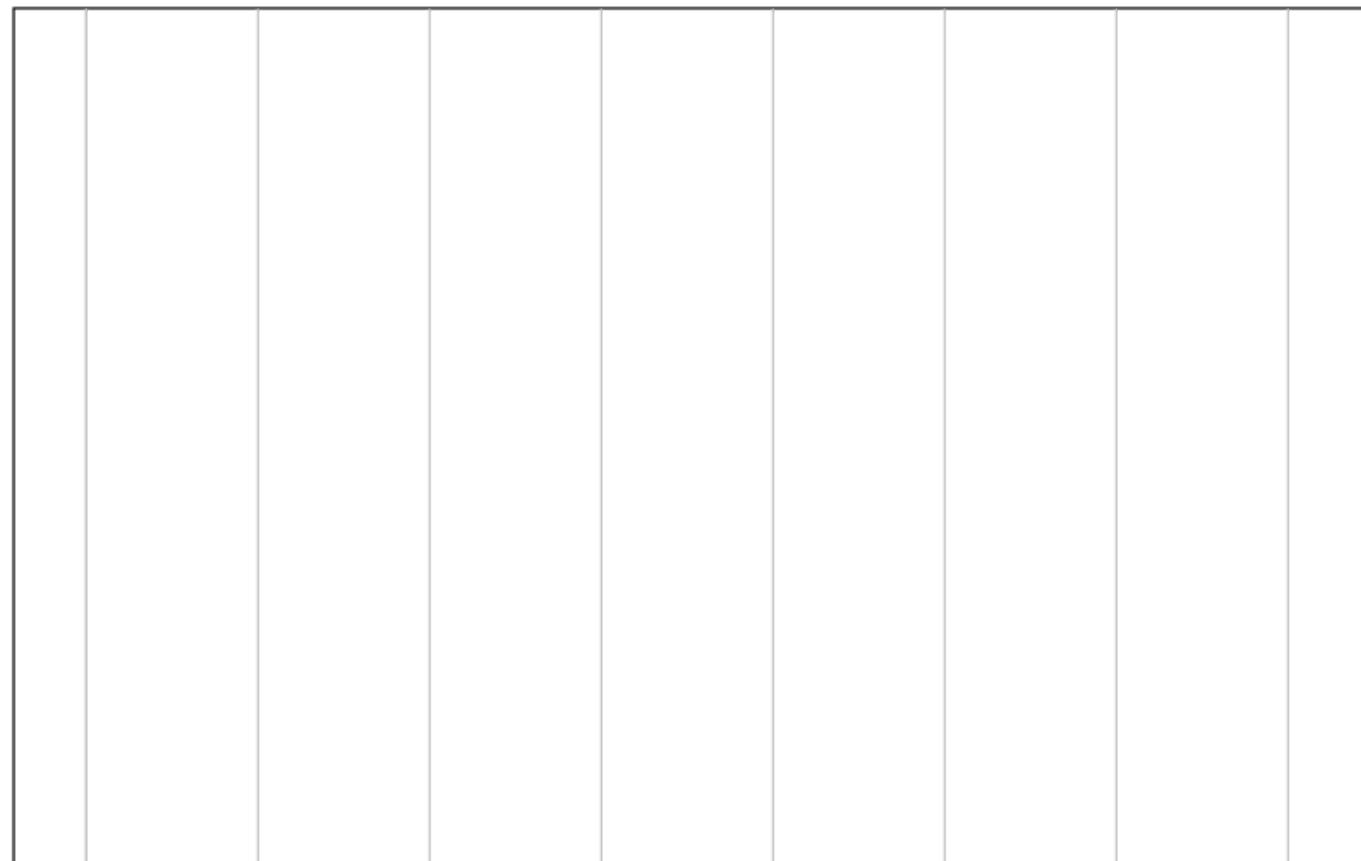
<https://aurora-opensource.github.io/au/main/troubleshooting/>

# C++ Units: the goal

# All of the people, all of the time

# All of the people, all of the time

## Robust Physical Units Support



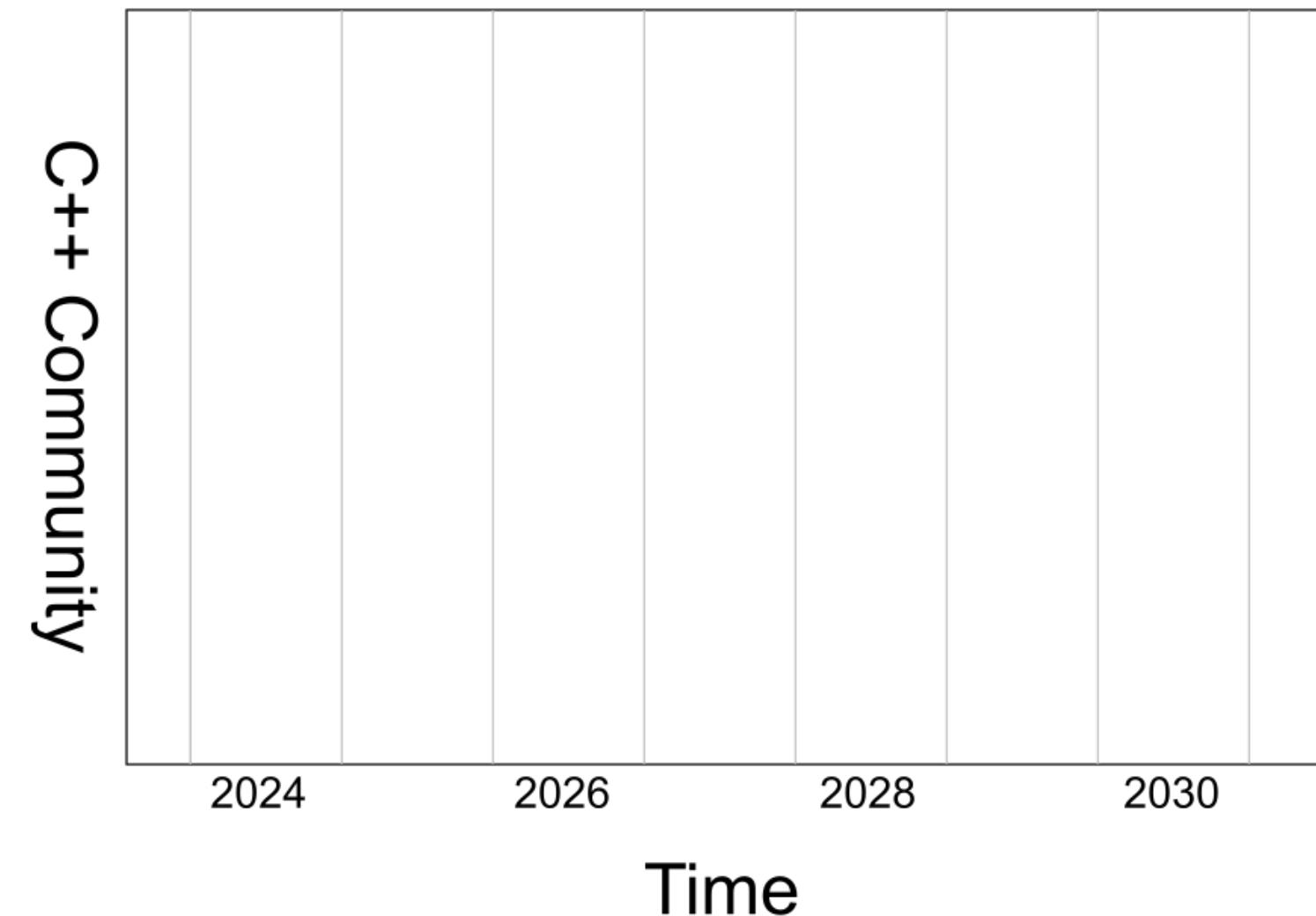
# All of the people, all of the time

## Robust Physical Units Support



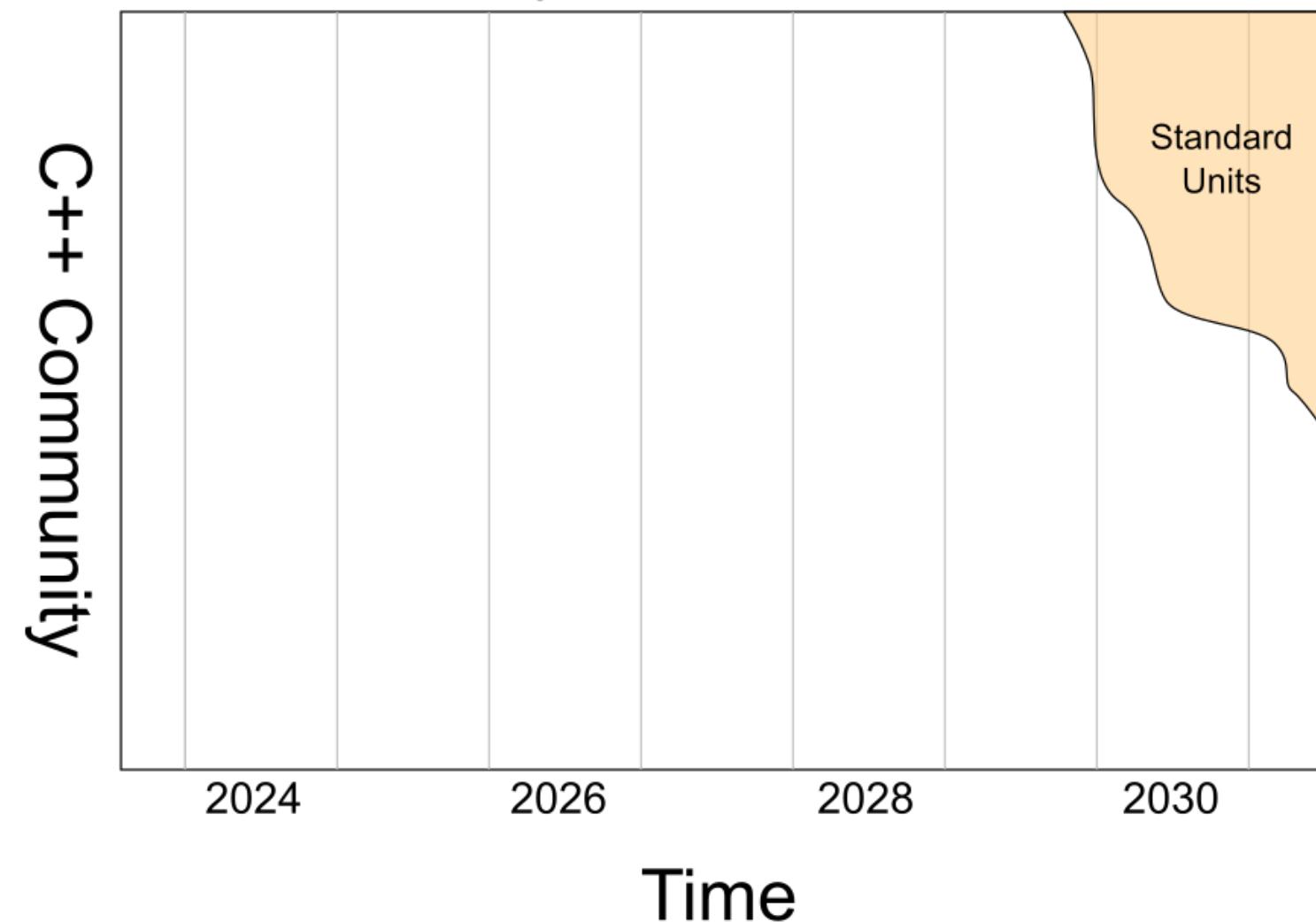
# All of the people, all of the time

## Robust Physical Units Support



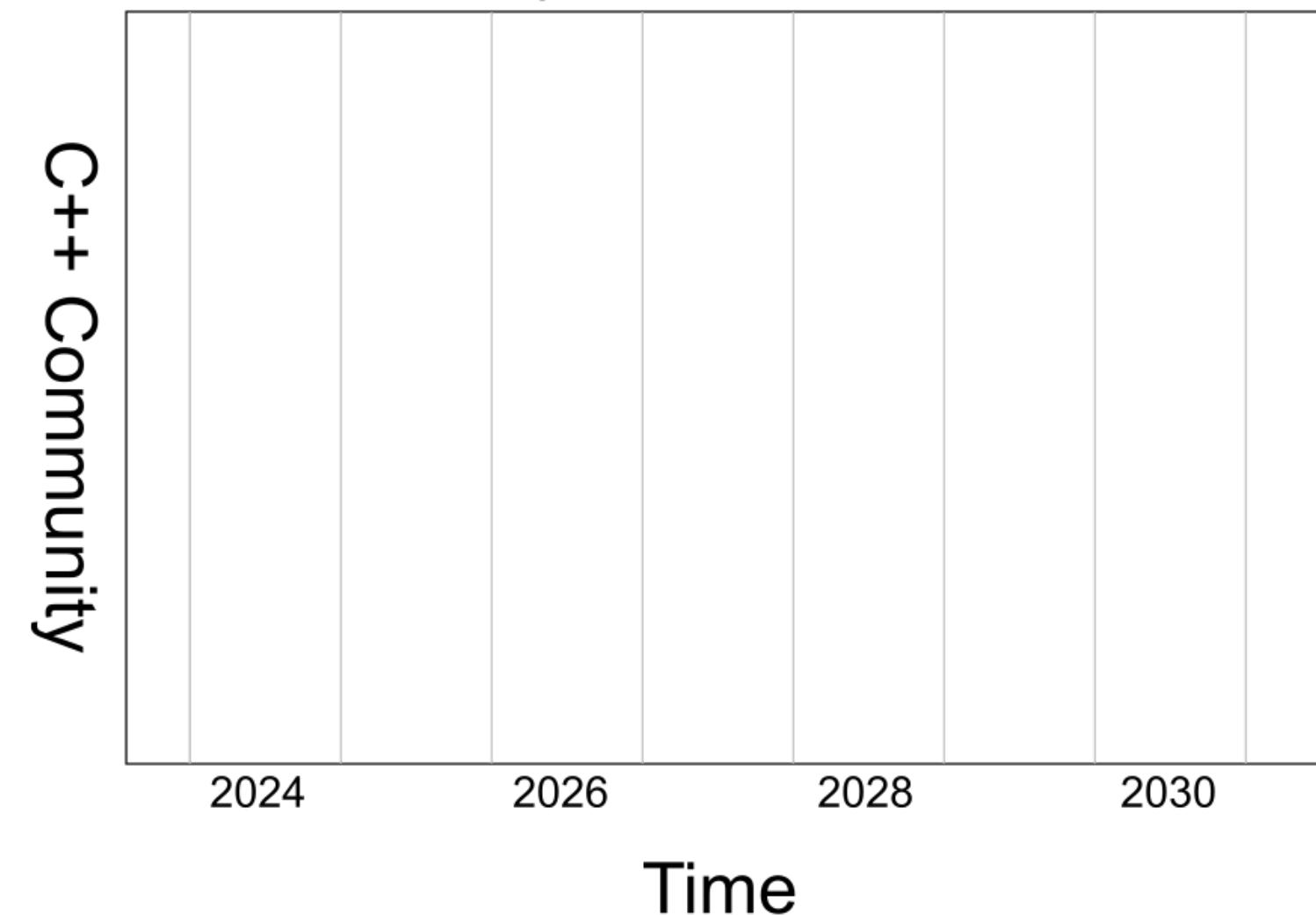
# Standard units library?

## Robust Physical Units Support



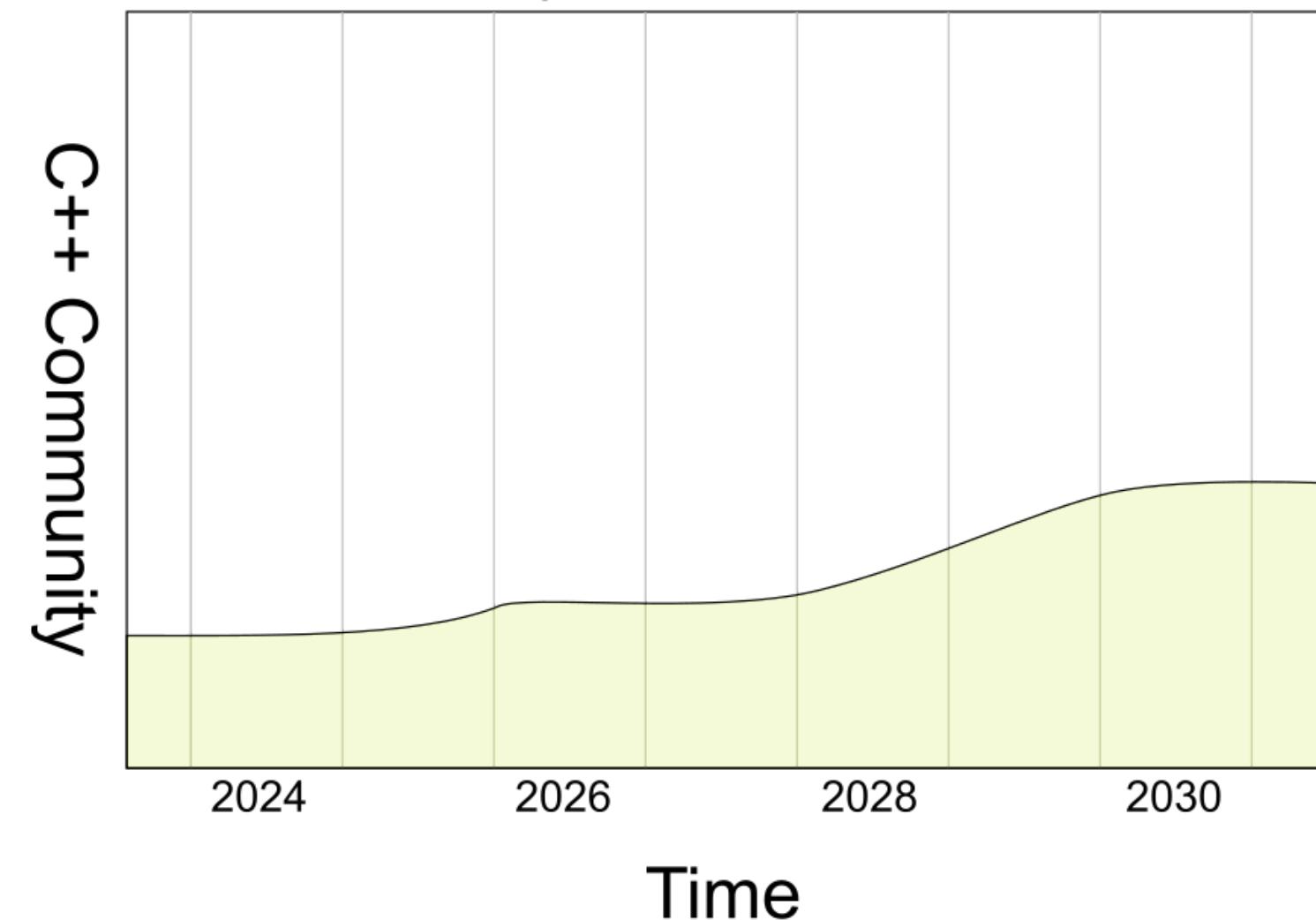
# The C++ Units Library Ecosystem

## Robust Physical Units Support



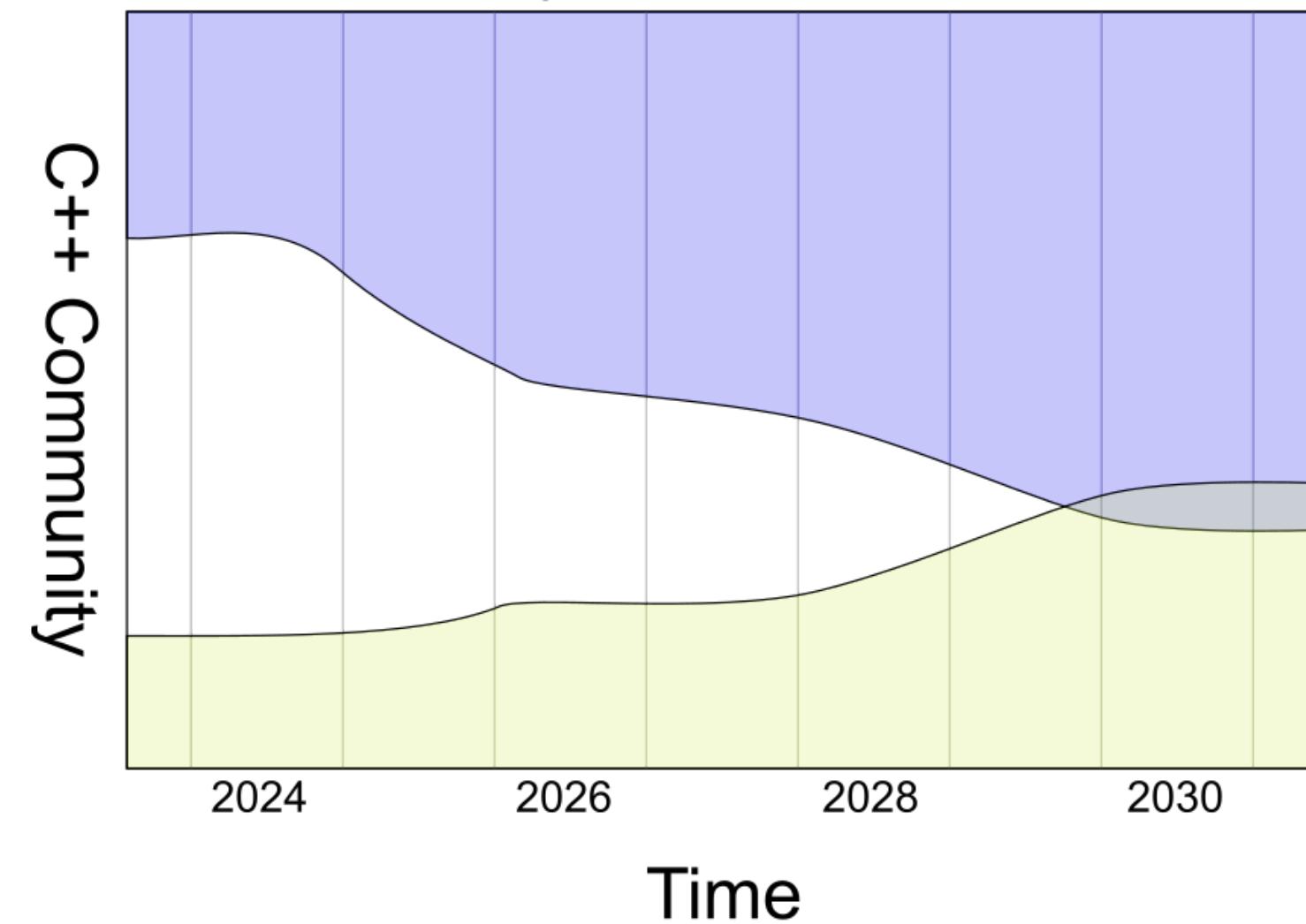
# The C++ Units Library Ecosystem

## Robust Physical Units Support



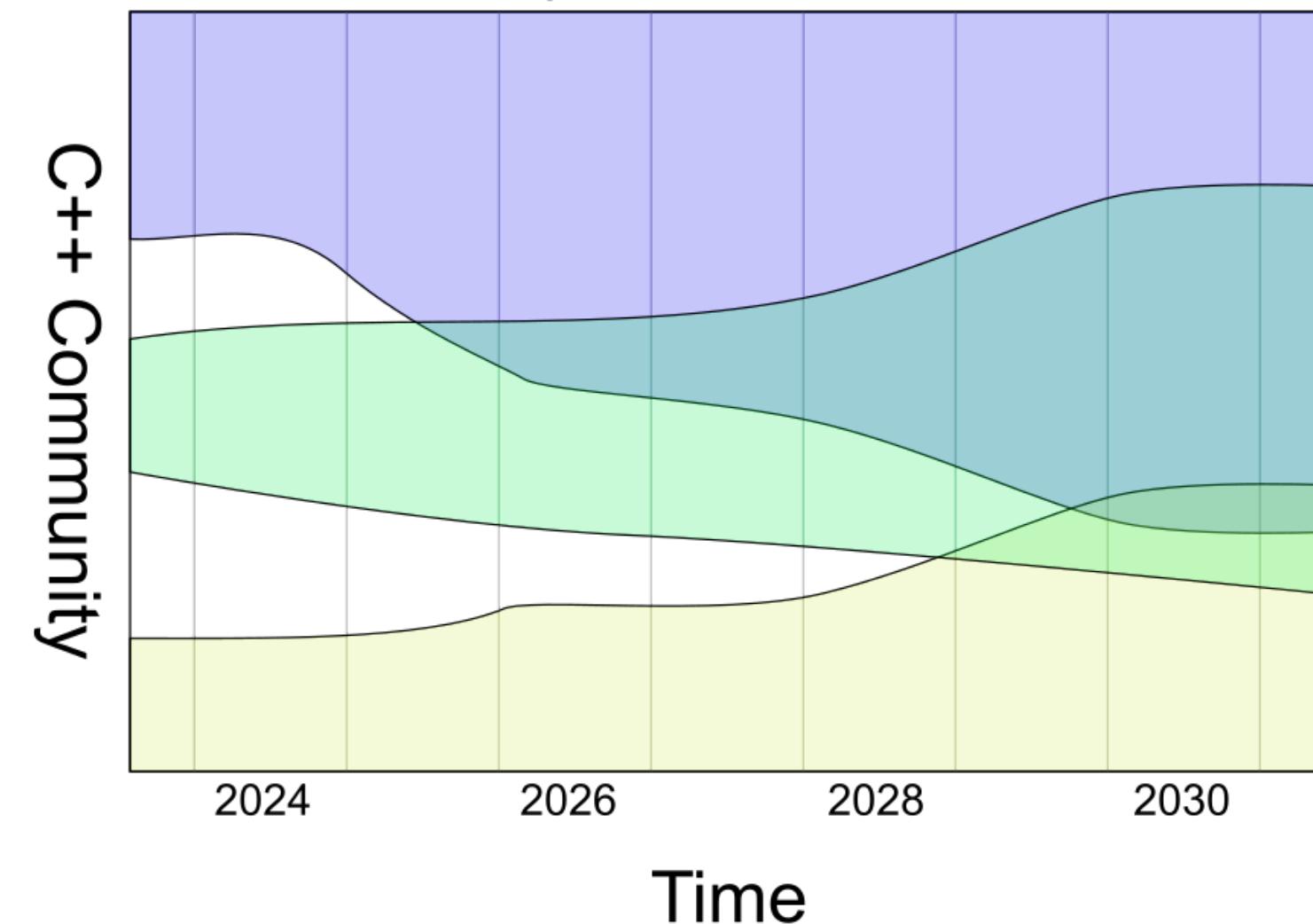
# The C++ Units Library Ecosystem

## Robust Physical Units Support



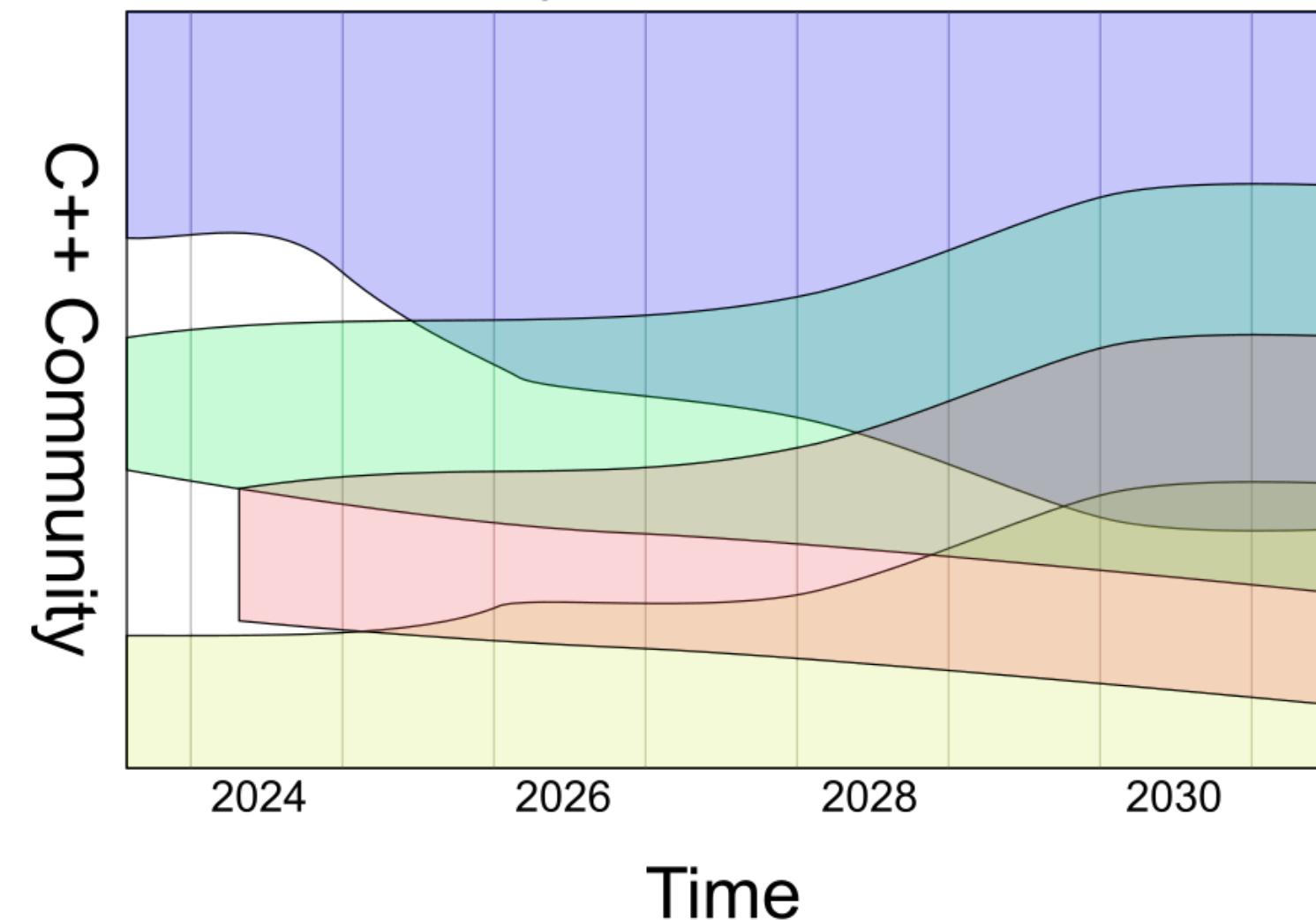
# The C++ Units Library Ecosystem

## Robust Physical Units Support



# The C++ Units Library Ecosystem

## Robust Physical Units Support



# Choosing a Units Library

# Framework for Choosing

# Framework for Choosing

1. Can you get it in your project?

# Framework for Choosing

1. **Can you get it in your project?**
2. **What does it cost, in terms of developer experience?**

# Framework for Choosing

1. **Can you get it in your project?**
2. **What does it cost, in terms of developer experience?**
3. **What "units library specific" features does it have?**

# Full comparison

<https://aurora-opensource.github.io/au/main/alternatives/>

# Full comparison

<https://aurora-opensource.github.io/au/main/alternatives/>

## 1. Can you get it?

	Boost	nholthaus	bernedom/SI	mp-units	Au
C++ Version Compatibility	C++98	C++14	C++17	C++20	C++14
Ease of Acquisition	Part of boost	Single, self-contained header	Available on conan	Available on conan and vcpkg	Supports single-header delivery, with features: <ul style="list-style-type: none"><li>✓ Easy to customize units and I/O support</li><li>✓ Version-stamped for full reproducibility</li></ul>

# Full comparison

<https://aurora-opensource.github.io/au/main/alternatives/>

## 1. Can you get it?

	Boost	nholthaus	bernedom/SI	mp-units	Au
C++ Version Compatibility	C++98	C++14	C++17	C++20	C++14
Ease of Acquisition	Part of boost	Single, self-contained header	Available on conan	Available on conan and vcpkg	Supports single-header delivery, with features: <ul style="list-style-type: none"><li>✓ Easy to customize units and I/O support</li><li>✓ Version-stamped for full reproducibility</li></ul>

## 2. What does it cost?

	Boost	nholthaus	bernedom/SI	mp-units	Au
Compilation Speed	(Not assessed)	Very slow, but can be greatly improved by removing I/O support and most units	(Not assessed)	(Not assessed)	Possibly "best", but will need to assess all libraries on the same code
Compiler Error Readability	Infamously challenging	Positional dimensions	Alias for unit template	<ul style="list-style-type: none"><li>✓ Pioneered strong typedefs for units</li><li>✓ Expression templates produce very readable errors</li></ul>	<ul style="list-style-type: none"><li>✓ Strong unit typenames appear in errors</li><li>✓ Short namespace minimizes clutter</li><li>✓ Detailed troubleshooting guide</li></ul>

# Full comparison

<https://aurora-opensource.github.io/au/main/alternatives/>

## 1. Can you get it?

	Boost	nholthaus	bernedom/SI	mp-units	Au
C++ Version Compatibility	C++98	C++14	C++17	C++20	C++14
Ease of Acquisition	Part of boost	Single, self-contained header	Available on conan	Available on conan and vcpkg	Supports single-header delivery, with features: <ul style="list-style-type: none"><li>✓ Easy to customize units and I/O support</li><li>✓ Version-stamped for full reproducibility</li></ul>

## 2. What does it cost?

	Boost	nholthaus	bernedom/SI	mp-units	Au
Compilation Speed	(Not assessed)	Very slow, but can be greatly improved by removing I/O support and most units	(Not assessed)	(Not assessed)	Possibly "best", but will need to assess all libraries on the same code
Compiler Error Readability	Infamously challenging	Positional dimensions	Alias for unit template	<ul style="list-style-type: none"><li>✓ Pioneered strong typedefs for units</li><li>✓ Expression templates produce very readable errors</li><li>✓ Detailed troubleshooting guide</li></ul>	<ul style="list-style-type: none"><li>✓ Strong unit typenames appear in errors</li><li>✓ Short namespace minimizes clutter</li><li>✓ Detailed troubleshooting guide</li></ul>

## 3. “Units library specific” features:

	Boost	nholthaus	bernedom/SI	mp-units	Au
Conversion Safety		Integer Reps unsafe	Integer Reps unsafe	Policy conversion with <code>std::numbers</code> library	Automatically adapts to level of overflow risk
Unit Safety					Only contains unit-safe interfaces
Low Friction	<ul style="list-style-type: none"><li>✗ Generally high learning curve</li><li>✗ No learning curve for implicit conversions</li><li>✗ Many headers hard to parse</li></ul>	<ul style="list-style-type: none"><li>✓ Single, short namespace</li><li>✓ User-friendly API references (<code>meter_t</code>, ...)</li><li>✗ Namespaces add unnecessary friction (for example, <code>nm::nm::namespace prevents #include</code>)</li></ul>	<ul style="list-style-type: none"><li>✗ Implicit conversions with pointers to objects</li><li>• Multiple headers, one per system</li><li>• Multiple headers, but easy to guess (one per dimension)</li></ul>	<ul style="list-style-type: none"><li>✓ Implict conversions with pointers to objects</li><li>• Multiple headers, one per system</li><li>• Long and more nested namespaces</li></ul>	<ul style="list-style-type: none"><li>✓ Namespaces: just one, and it's often <code>nm</code>, so the single header or easily guessable header per unit</li></ul>
Composability	Prefix only	No	No	Can compose units, prefixes, dimensions, and quantity types	QuantityMaker and PrefixApplier APIs
Unit-aware I/O	<ul style="list-style-type: none"><li>✗ Toggable <code>&lt;iostream&gt;</code> support</li><li>✗ Impressively configurable output (format mode, precision, etc.)</li><li>✗ No fmm support</li></ul>	<ul style="list-style-type: none"><li>✓ Toggable <code>&lt;iostream&gt;</code> support</li><li>✗ No fmm support</li></ul>	<ul style="list-style-type: none"><li>✗ Toggable <code>&lt;iostream&gt;</code> support</li><li>✗ Unit labels available even without <code>&lt;iostream&gt;</code></li><li>✗ No fmm support</li></ul>	<ul style="list-style-type: none"><li>✓ Supports <code>&lt;iostream&gt;</code> support</li><li>✓ Unit labels available even without <code>&lt;iostream&gt;</code></li><li>✗ Plan to add fmm support, see #140</li></ul>	<ul style="list-style-type: none"><li>✗ Toggable <code>&lt;iostream&gt;</code> support</li><li>✓ Unit labels available even without <code>&lt;iostream&gt;</code></li><li>✗ Plan to add fmm support, see #140</li></ul>
Mixed-Rep Support			Possible, but user-facing types use a global “preferred” Rep		
Unit-aware math	<ul style="list-style-type: none"><li>✗ Wide variety of functions</li><li>✗ <code>round</code>, <code>ceil</code>, and so on are not unit-safe</li></ul>	<ul style="list-style-type: none"><li>✗ Wide variety of functions</li><li>✗ <code>round</code>, <code>ceil</code>, and so on are not unit-safe</li></ul>	<ul style="list-style-type: none"><li>✗ Wide variety of functions</li><li>✗ Unit-safe APIs for <code>round</code>, <code>ceil</code>, and so on</li></ul>	<ul style="list-style-type: none"><li>✓ Wide variety of functions</li><li>✓ Unit-safe APIs for <code>round</code>, <code>ceil</code>, and so on</li></ul>	<ul style="list-style-type: none"><li>✓ Wide variety of functions</li><li>✓ Unit-safe APIs for <code>round</code>, <code>ceil</code>, and so on</li><li>✓ Smart, unit-aware inverse functions</li></ul>
Generic Dimensions	Generic templates, constrained with traits	Generic templates, constrained with traits	Generic templates, constrained with traits	Concepts excel here	Currently quirky. Could be better by adding concepts in extra C++14-only files, without complicating C++11 support.
Extensibility	Can add new units and dimensions	<ul style="list-style-type: none"><li>✓ One-line macro defines new units</li><li>✗ Can't add dimensions</li></ul>	Can add new units and dimensions	Can add new units and dimensions	Can add new units and dimensions
Ease of Migration	No interop with other units libraries	No interop with other units libraries	No interop with other units libraries	<ul style="list-style-type: none"><li>✗ No interop with other units libraries</li></ul>	Equivalent trait <sup>®</sup> feature gives more API compatibility
Point Types	<ul style="list-style-type: none"><li>✗ <code>absolute</code> wrapper for unit</li></ul>	<ul style="list-style-type: none"><li>Optional “offset” for units, but can't distinguish quantity from point</li></ul>	<ul style="list-style-type: none"><li>None, would be hard to add, since units conflated with quantity type</li></ul>		
Magnitudes	<ul style="list-style-type: none"><li>Close, but only instantiates units, and instance arithmetic is Ahead of its time!</li></ul>	<ul style="list-style-type: none"><li>“std::ratio” only with no solution for pi</li></ul>	Full support for Magnitudes	Formerly, Au alone was best, but we're in the Magnitudes with requests	
Embedded Friendliness	Assumed to be good, based on mixed Rep support	Can trim by excluding <code>&lt;iostream&gt;</code> , but integer Rep support is poor	<ul style="list-style-type: none"><li>✓ <code>&lt;iostream&gt;</code> not automatically included</li><li>✗ Supports integral Rep</li><li>✗ Integer Rep conversions unsafe</li></ul>	Assumed to be good, based on mixed Rep support	<ul style="list-style-type: none"><li>✗ Poor choice of all Reps (no “preferred” Rep, <code>&lt;absolute&gt;</code> handles unit label representation)</li><li>✓ Safe integer operations</li></ul>
Abbreviated construction		User-defined literals (UDLs)	User-defined literals (UDLs)	Quantity References	Planned to add #43
Linear algebra				Experimental support for Quantity character, can use matrix type as Rep	Planned to add #70
Rep Variety	Supports custom numeric types	Efficiently floating-point only (integer types unsafe)	<ul style="list-style-type: none"><li>✗ No “default” Rep</li><li>✗ Integer Rep unsafe</li></ul>	Well-defined Replication concept	<ul style="list-style-type: none"><li>✗ Return support for <code>&lt;iostream&gt;</code> Rep</li><li>✓ Experimental support for custom Rep</li><li>✗ No constraints yet (#12)</li></ul>
Zero	Guidance: use default constructor to specialize, but no special constructor for comparison	Supports <code>represents()</code> , but no special constructor or comparison	No special constructor or comparison	Holds <code>iszero()</code> member, but no special constructor or comparison	Can use <code>iszero()</code> to construct or compare any quantity
Angles	Curiously Imprecise pi value		<ul style="list-style-type: none"><li>✓ Supports degrees and radians</li><li>✗ pi represented as <code>std::ratio</code></li></ul>		
Physical constants	Includes built-in constants as quantities	Includes built-in constants as quantities		“Faster than lightning” constants	Plan to support someday, see #101
Non-linear scales (such as dB)					Plan to support someday, see #11
“Kil” Types	(Not assessed)				No plans at present to support.
Explicit Systems of Measurement		Single, implicit global system			Single, implicit global system (intentional design tradeoff reduces learning curve, and makes compiler analysis simpler)
Units/Dimensions as types	<ul style="list-style-type: none"><li>✗ Types exist, but confused with quantity names</li></ul>	<ul style="list-style-type: none"><li>✗ No separate types for units</li></ul>	<ul style="list-style-type: none"><li>✓ Types for units</li><li>✓ Types for dimensions</li></ul>		<ul style="list-style-type: none"><li>✗ Types for units</li><li>✓ Types for dimensions</li><li>✓ Can do arithmetic (compound units on the fly, abstract dimensional analysis)</li></ul>
Macro Usage	Present in user-facing APIs	Present in user-facing APIs	Very few and confined to implementation helpers	Very few and confined to implementation helpers	Macros

# Au and alternatives

# Libraries considered

Library Name	GitHub repo
 boost C++ LIBRARIES	Boost Units  <a href="https://github.com/boostorg/units">boostorg/units</a>

# Libraries considered

Library Name	GitHub repo
 boost C++ LIBRARIES	Boost Units  boostorg/units
 nholthaus	nholthaus units  nholthaus/units

# Libraries considered

Library Name	GitHub repo
 boost C++ LIBRARIES	Boost Units  boostorg/units
 nholthaus	nholthaus units  nholthaus/units
 SI	 bernedom/SI

# Libraries considered

Library Name

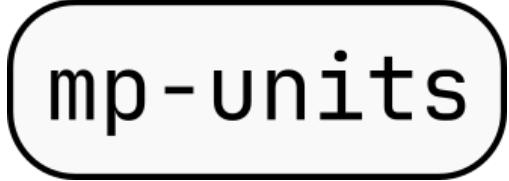
GitHub repo



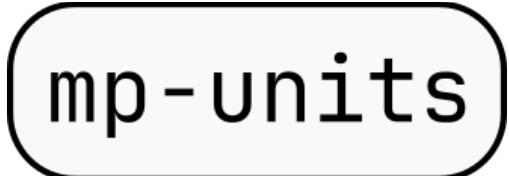
nholt



# Libraries considered

	Library Name	GitHub repo
	Boost Units	 <a href="https://github.com/boostorg/units">boostorg/units</a>
	nholthaus units	 <a href="https://github.com/nholthaus/units">nholthaus/units</a>
	SI	 <a href="https://github.com/bernedom/SI">bernedom/SI</a>
	mp-units	 <a href="https://github.com/mpusz/mp-units">mpusz/mp-units</a>

# Libraries considered

	Library Name	GitHub repo
	Boost Units	 <a href="https://github.com/boostorg/units">boostorg/units</a>
	nholthaus units	 <a href="https://github.com/nholthaus/units">nholthaus/units</a>
	SI	 <a href="https://github.com/bernedom/SI">bernedom/SI</a>
	mp-units	 <a href="https://github.com/mpusz/mp-units">mpusz/mp-units</a>
	Au	 <a href="https://github.com/aurora-opensource/au">aurora-opensource/au</a>

# 1. Can you get it?

## a) C++ standard compatibility

C++ Version	Example features
C++20	Concepts Non-type template parameters
C++17	Fold expressions <code>constexpr if</code>
C++14	More permissive <code>constexpr</code> <code>auto</code> return type
C++11	Primitive <code>constexpr</code> Variadic templates <code>static_assert</code>
C++98/03	

Legend: “C++ version is (fully / partially / not-at-all) supported in project.”

# 1. Can you get it?

## a) C++ standard compatibility

C++ Version	Example features
C++20	Concepts Non-type template parameters
C++17	Fold expressions <code>constexpr if</code>
C++14	More permissive <code>constexpr</code> <code>auto</code> return type
C++11	Primitive <code>constexpr</code> Variadic templates <code>static_assert</code>
C++98/03	



Legend: “C++ version is (fully / partially / not-at-all) supported in project.”

# 1. Can you get it?

## a) C++ standard compatibility

C++ Version	Example features
C++20	Concepts Non-type template parameters
C++17	Fold expressions <code>constexpr if</code>
C++14	More permissive <code>constexpr</code> <code>auto</code> return type
C++11	Primitive <code>constexpr</code> Variadic templates <code>static_assert</code>
C++98/03	

**nholthaus**



Legend: “C++ version is (fully / partially / not-at-all) supported in project.”

# 1. Can you get it?

## a) C++ standard compatibility

	C++ Version	Example features
	C++20	Concepts Non-type template parameters
 	C++17	Fold expressions <code>constexpr if</code>
	C++14	More permissive <code>constexpr</code> <code>auto</code> return type
	C++11	Primitive <code>constexpr</code> Variadic templates <code>static_assert</code>
	C++98/03	

Legend: “C++ version is (fully / partially / not-at-all) supported in project.”

# 1. Can you get it?

## a) C++ standard compatibility

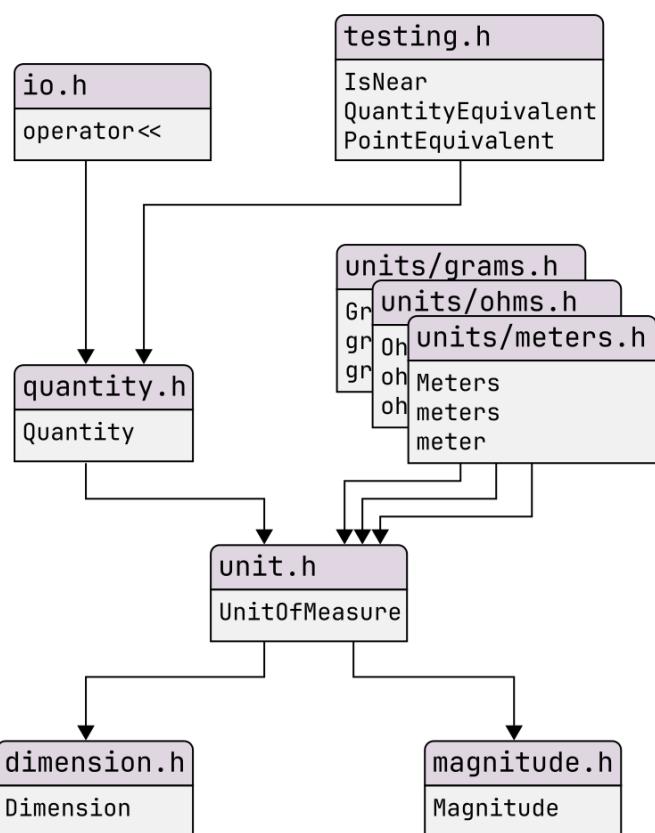
	C++ Version	Example features
<b>mp-units</b>	C++20	Concepts Non-type template parameters
	C++17	Fold expressions <code>constexpr if</code>
<b>nholthaus</b> 	C++14	More permissive <code>constexpr</code> <code>auto</code> return type
	C++11	Primitive <code>constexpr</code> Variadic templates <code>static_assert</code>
	C++98/03	

Legend: “C++ version is (fully / partially / not-at-all) supported in project.”

1. Can you get it?
  - b) Delivery mechanism

1. Can you get it?
  - b) Delivery mechanism

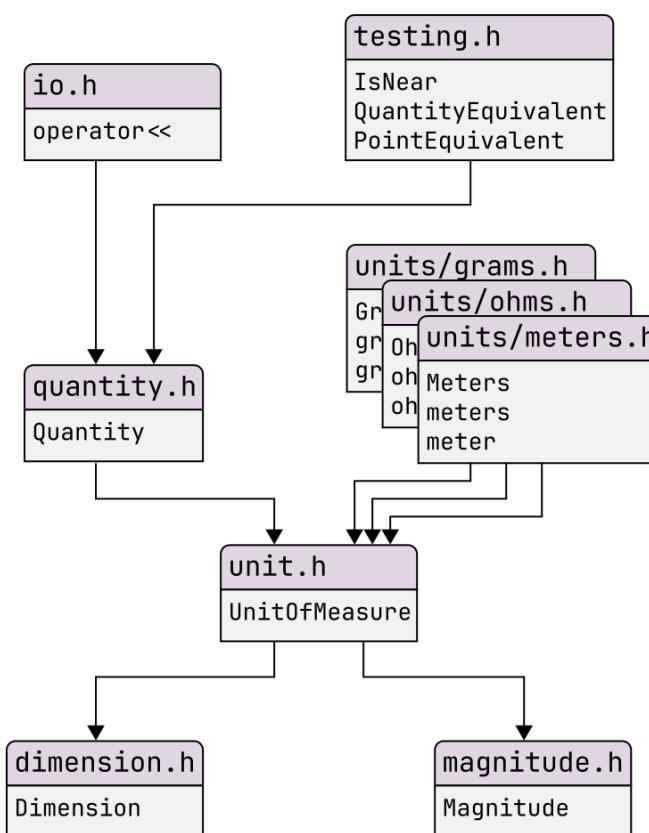
## Full Installation



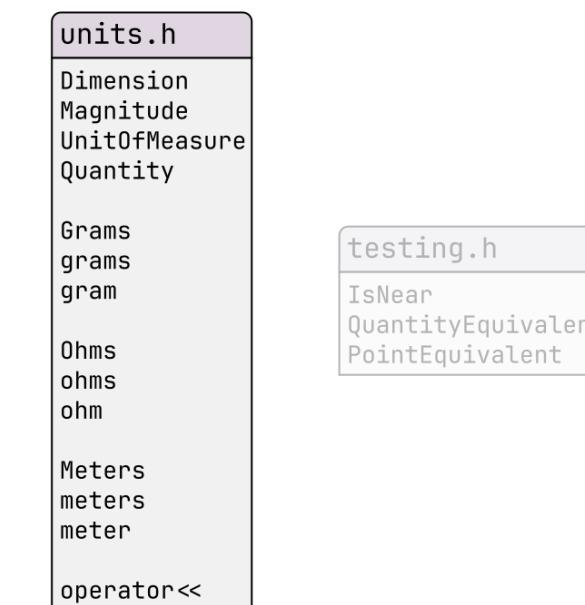
# 1. Can you get it?

## b) Delivery mechanism

Full Installation



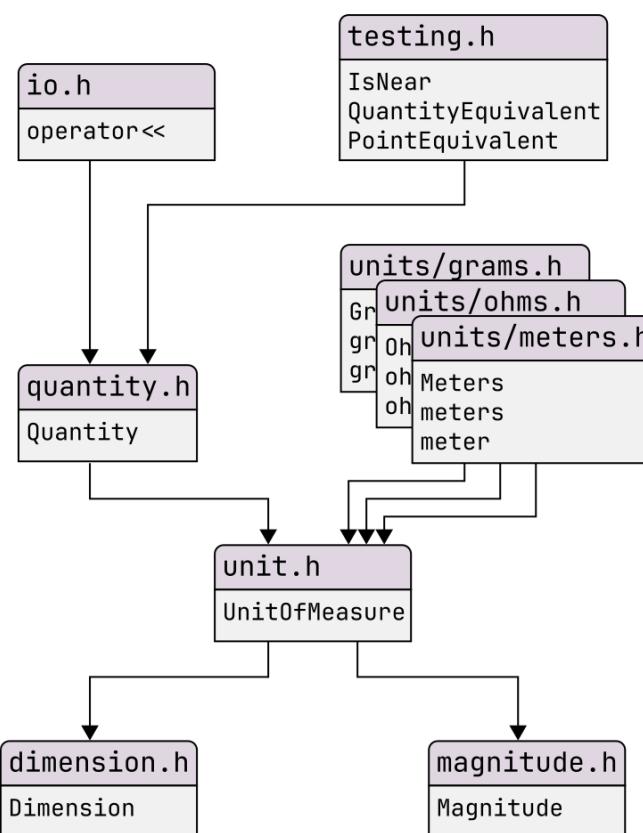
Single File



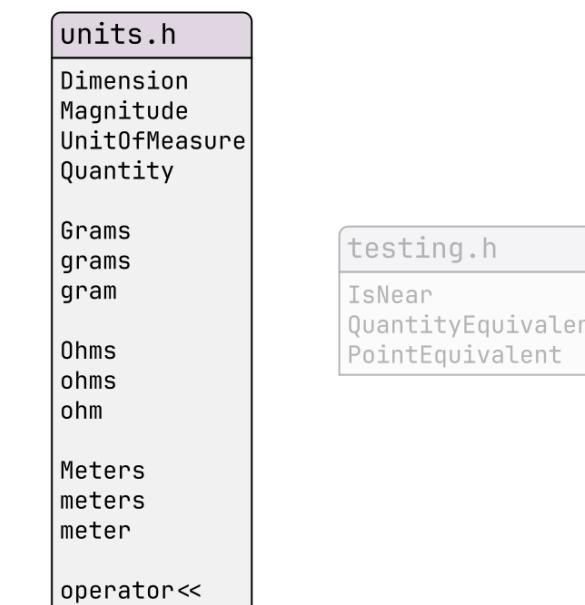
# 1. Can you get it?

## b) Delivery mechanism

Full Installation



Single File



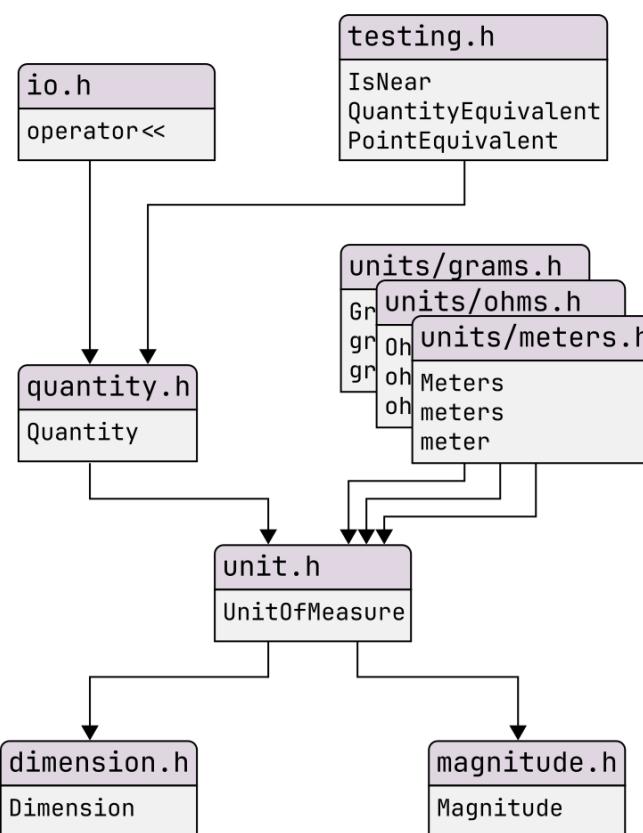
mp-units

nholthaus

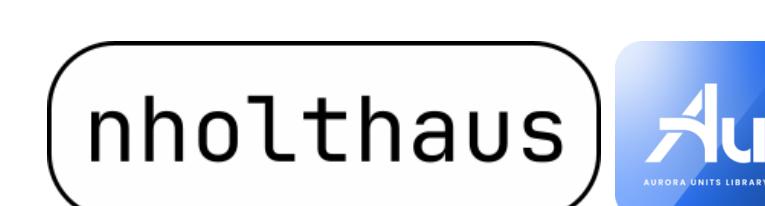
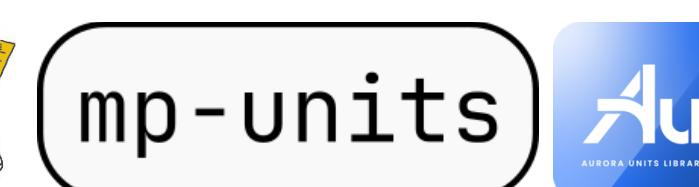
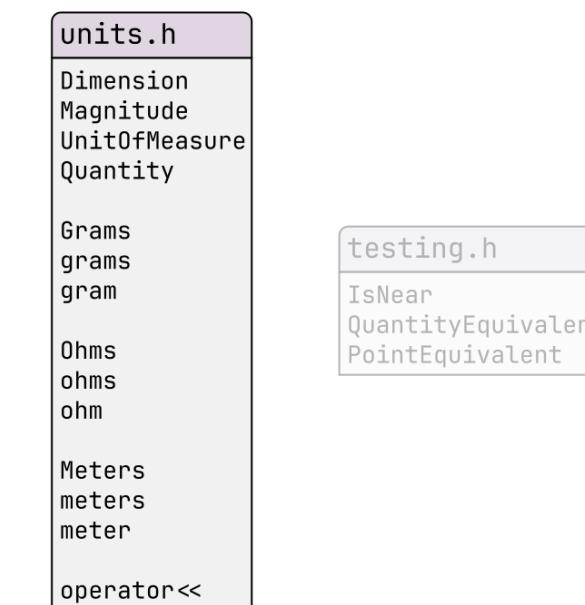
# 1. Can you get it?

## b) Delivery mechanism

Full Installation



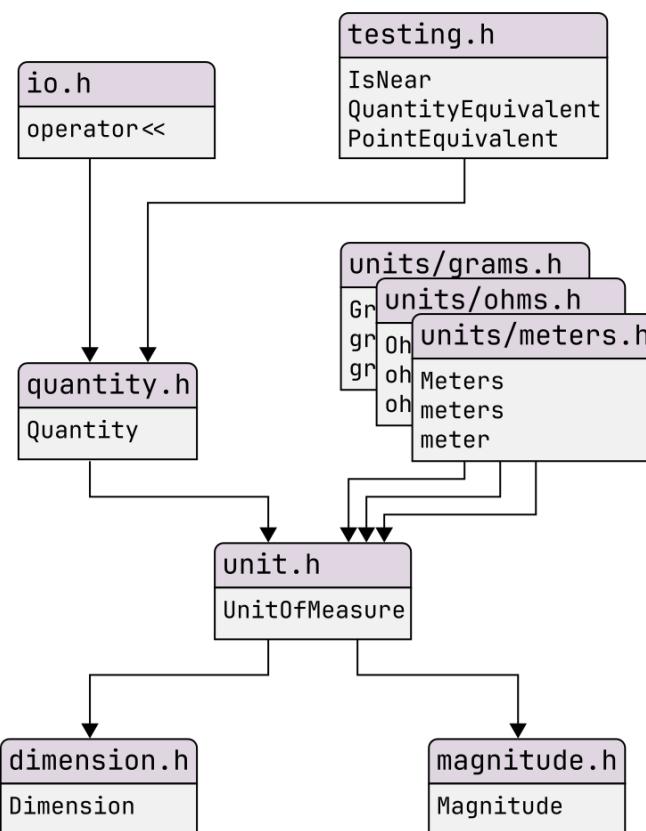
Single File



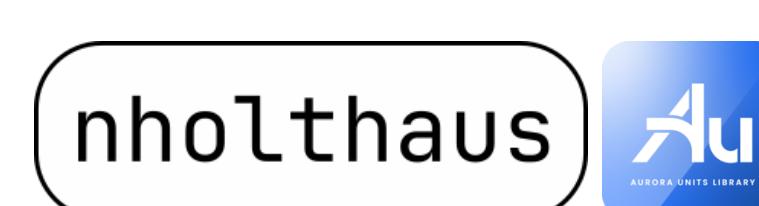
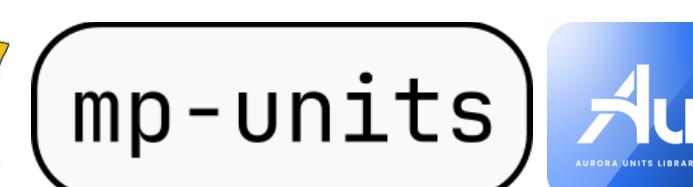
# 1. Can you get it?

## b) Delivery mechanism

### Full Installation

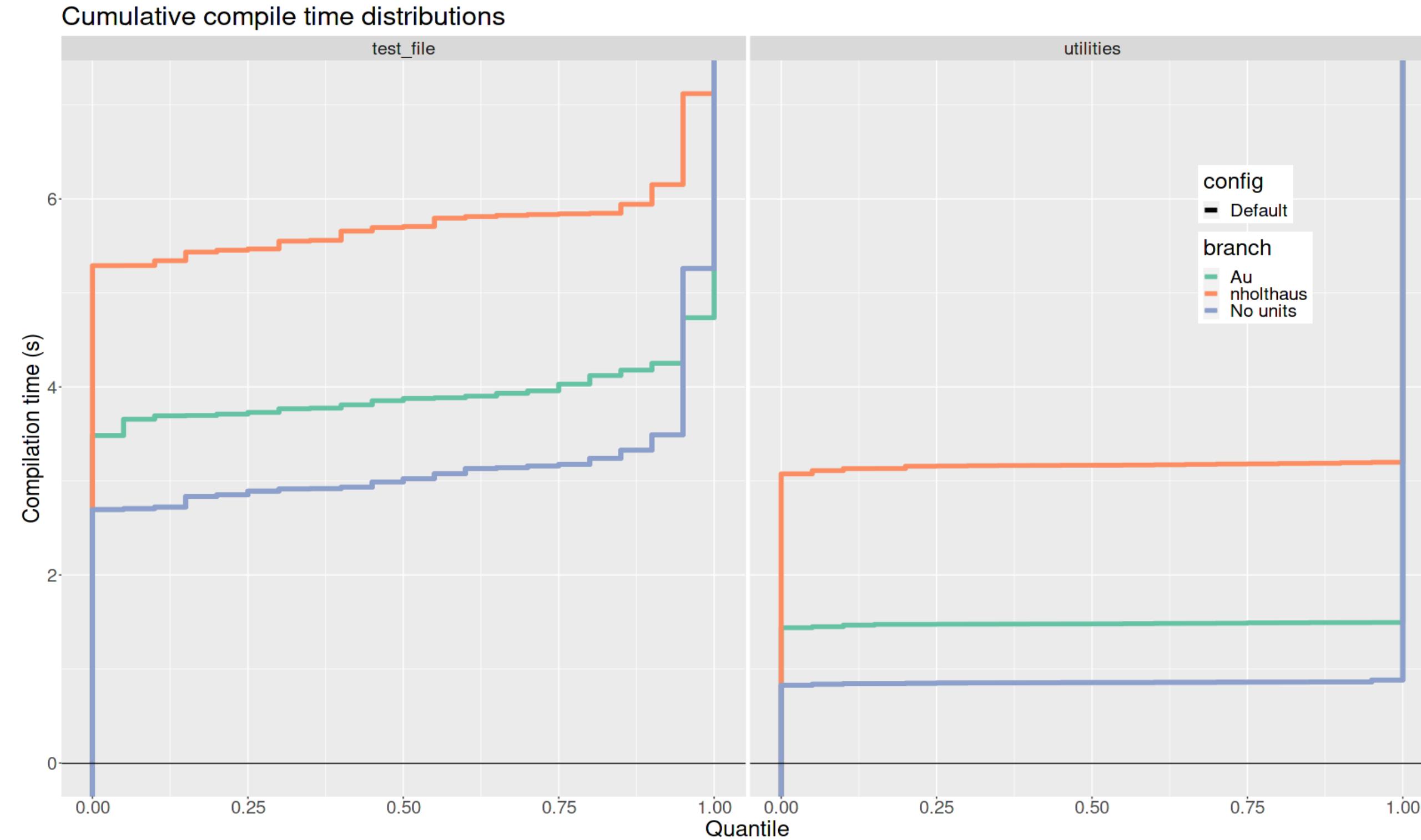


```
// Version identifier: 0.3.3
// <iostream> support: INCLUDED
// List of included units:
//   amperes
//   bits
//   candelas
//   grams
//   kelvins
//   meters
//   moles
//   radians
//   seconds
//   unos
```

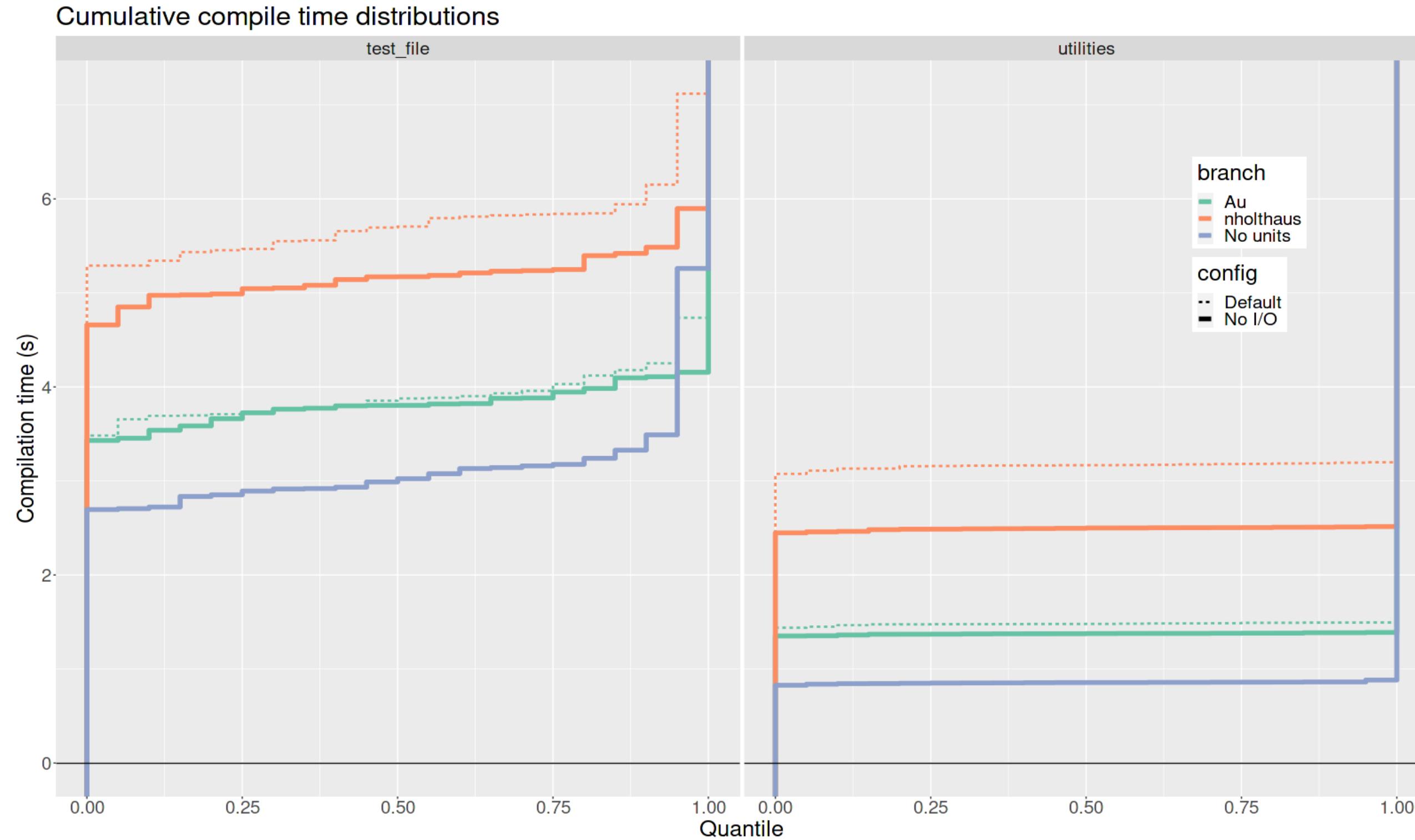


## 2. DevEx cost? a) Compile times

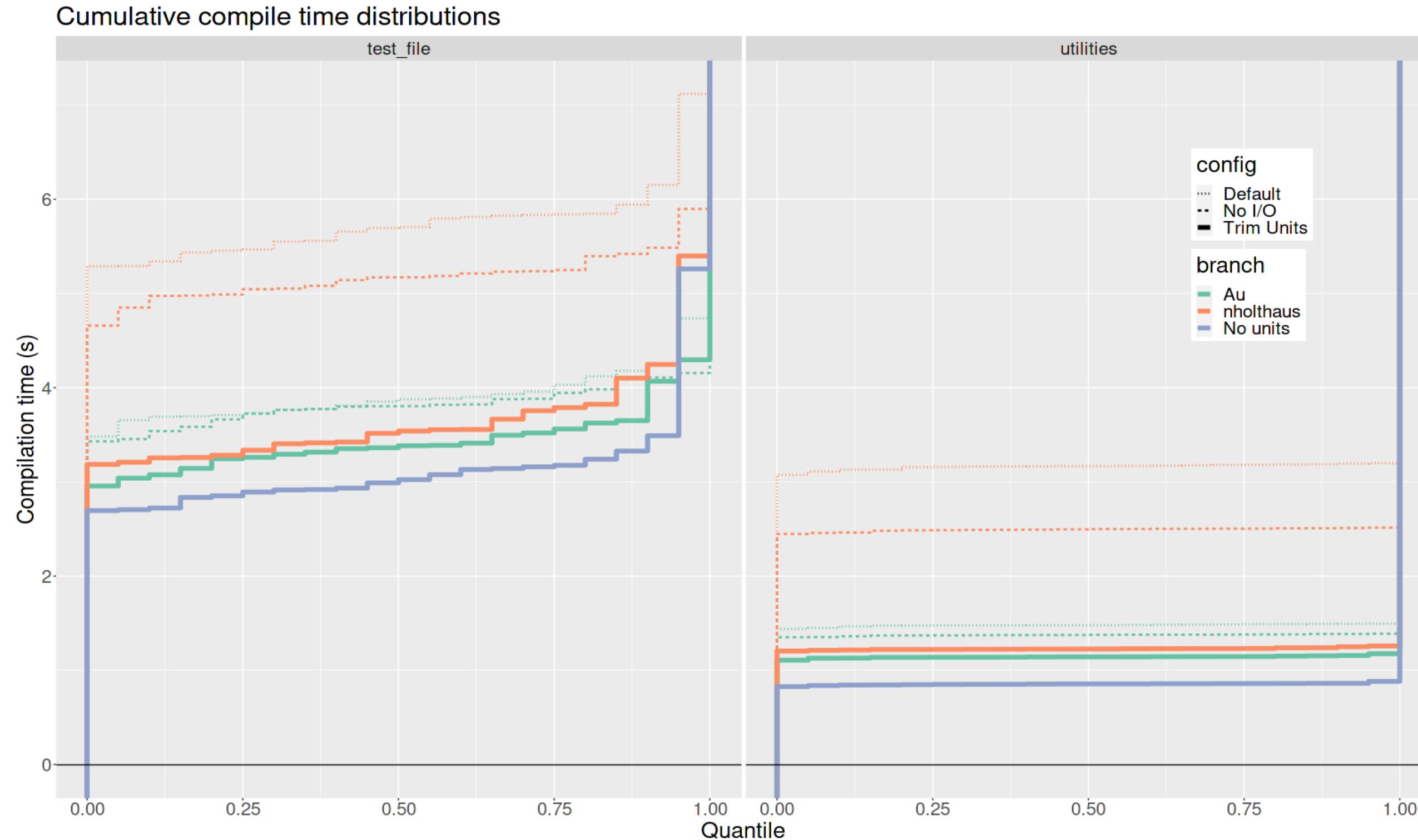
# 2. DevEx cost? a) Compile times



# 2. DevEx cost? a) Compile times



# 2. DevEx cost? a) Compile times



## 2. DevEx cost? b) Compiler errors

# 2. DevEx cost? b) Compiler errors



Code:

```
boost::units::quantity<boost::units::si::velocity> v =  
    (5 * boost::units::si::meters) * (1.0 * boost::units::si::seconds);
```

Error:

x86-64 gcc 7.1

```
<Compilation failed>  
  
<source>: In function 'int main(int, char**)':  
<source>:9:38: error: conversion from  
'boost::units::multiply_typeof_helper<boost::units::quantity<boost::units::unit<  
boost::units::list<boost::units::dim<boost::units::length_base_dimension,  
boost::units::static_rational<1> >, boost::units::dimensionless_type>,>  
boost::units::homogeneous_system<boost::units::list<boost::units::si::  
meter_base_unit,  
boost::units::list<boost::units::scaled_base_unit<boost::units::cgs::  
gram_base_unit,
```

# 2. DevEx cost? b) Compiler errors

nholthaus

Code:

```
units::velocity::meters_per_second_t v =
    units::length::meter_t{5.0} * units::time::second_t{1.0};
```

Error:

x86-64 gcc 7.1

<Compilation failed>

```
In file included from <source>:3:0:
/app/raw.githubusercontent.com/nholthaus/units/master/include/units.h: In
instantiation of 'constexpr T units::convert(const T&) [with UnitFrom
= units::unit<std::ratio<1>, units::base_unit<std::ratio<1>, std::ratio<0, 1>,
std::ratio<1>, std::ratio<0, 1>, std::ratio<0, 1>, std::ratio<0, 1>, std::ratio<0,
1>, std::ratio<0, 1>, std::ratio<0, 1> >, std::ratio<0, 1>, std::ratio<0, 1> >];
UnitTo = units::unit<std::ratio<1>, units::base_unit<std::ratio<1>, std::ratio<0,
1>, std::ratio<-1> > >; T = double]':
/app/raw.githubusercontent.com/nholthaus/units/master/include/units.h:1976:41:
required from 'constexpr units::unit_t<Units, T, NonLinearScale>::unit_t(const
```

# 2. DevEx cost? b) Compiler errors

mp-units

Code:

```
quantity<si::metre / si::second> v1 = (5.0 * m) * (1.0 * s);
```

Error:

x86-64 gcc 13.1

<Compilation failed>

```
<source>: In function 'int main()':  
<source>:8:49: error: conversion from 'quantity<mp_units::derived_unit<  
mp_units::si::metre, mp_units::si::second>(), [...]>'  
to non-scalar type 'quantity<mp_units::derived_unit<  
mp_units::si::metre, mp_units::per<mp_units::si::second> >(), [...]>' requested  
8 |   quantity<si::metre / si::second> v1 = (5 * m) * (1. * s);  
|           ~~~~~^~~~~~  
Compiler returned: 1
```

# 2. DevEx cost? b) Compiler errors



Code:

```
Quantity<decltype(Meters{} / Seconds{}) , double> v = meters(5.0) * seconds(1.0);
```

Error:

x86-64 gcc 7.1

<Compilation failed>

```
<source>: In function 'int main(int, char**)':
<source>:9:70: error: conversion from
'au::Quantity<au::UnitProduct<au::Meters, au::Seconds>, double>' to non-scalar type
'au::Quantity<au::UnitProduct<au::Meters, au::Pow<au::Seconds, -1> >, double>'
requested
    Quantity<decltype(Meters{} / Seconds{}) , double> v = meters(5.0) * seconds(1.0);
```

~~~~~^~~~~~

Compiler returned: 1

# Au: core features

# Unit safety

*A program is “unit safe” when the correct handling of physical units can be verified in each individual line, by inspection, in isolation.*

# Unit safety

A program is “unit safe” when the correct handling of physical units can be verified in each individual line, by inspection, in isolation.

```
1 auto height = meters(1.87);
2 //          ^^^^^^
3
4
5
6
7
8
9
```

# Unit safety

A program is “unit safe” when the correct handling of physical units can be verified in each individual line, by inspection, in isolation.

```
1 auto height = meters(1.87);
2 //           ^^^^^^
3
4 QuantityD<Joules> potential_energy = m * g * height;
5
6
7
8
9
```

# Unit safety

A program is “unit safe” when the correct handling of physical units can be verified in each individual line, by inspection, in isolation.

```
1 auto height = meters(1.87);
2 //          ^^^^^^
3
4 QuantityD<Joules> potential_energy = m * g * height;
5
6 const bool can_fit = (height < clearance);
7
8
9
```

# Unit safety

A program is “unit safe” when the correct handling of physical units can be verified in each individual line, by inspection, in isolation.

```
1 auto height = meters(1.87);
2 //           ^^^^^^
3
4 QuantityD<Joules> potential_energy = m * g * height;
5
6 const bool can_fit = (height < clearance);
7
8 proto.set_height_m(height.in(meters));
9 //           ^           ^^^^^^
```

# Same program, only safer

# Same program, only safer

(No units library)

```
int degrees_per_second_from_rpm(int rpm) {  
    return rpm * 6; // Magic number!  
}
```

# Same program, only safer

(No units library)

```
int degrees_per_second_from_rpm(int rpm) {  
    return rpm * 6; // Magic number!  
}
```

nholthaus

```
int degrees_per_second_from_rpm(int rpm) {  
    return degrees_per_second_t{revolutions_per_minute_t{rpm}}.to<int>();  
}
```

# Same program, only safer

(No units library)

```
int degrees_per_second_from_rpm(int rpm) {  
    return rpm * 6; // Magic number!  
}
```

nholthaus

```
int degrees_per_second_from_rpm(int rpm) {  
    return degrees_per_second_t{revolutions_per_minute_t{rpm}}.to<int>();  
}
```



```
int degrees_per_second_from_rpm(int rpm) {  
    return (revolutions / minute)(rpm).in(degrees / second);  
}
```

# Same program, only safer

(No units library)

```
int degrees_per_second_from_rpm(int rpm) {
    return rpm * 6; // Magic number!
}
```

```
no_units::degrees_per_second_from_rpm(int):
    lea    eax, [rdi+rdi*2]
    add    eax, eax
    ret
```

nholthaus

```
int degrees_per_second_from_rpm(int rpm) {
    return degrees_per_second_t{revolutions_per_minute_t{rpm}}.to<int>();
}
```

```
nholthaus::degrees_per_second_from_rpm(int):
    pxor   xmm0, xmm0
    cvtsi2sd      xmm0, edi
    mulsd  xmm0, QWORD PTR .LC0[rip]
    cvttsd2si    eax, xmm0
    ret
```



```
int degrees_per_second_from_rpm(int rpm) {
    return (revolutions / minute)(rpm).in(degrees / second);
}
```

```
with_au::degrees_per_second_from_rpm(int):
    lea    eax, [rdi+rdi*2]
    add    eax, eax
    ret
```

# Same program, only safer

(No units library)

```
int degrees_per_second_from_rpm(int rpm) {
    return rpm * 6; // Magic number!
}
```

nholthaus

```
int degrees_per_second_from_rpm(int rpm) {
    return degrees_per_second_t{revolutions_per_minute_t{rpm}}.to<int>();
}
```



```
int degrees_per_second_from_rpm(int rpm) {
    return (revolutions / minute)(rpm).in(degrees / second);
}
```

```
no_units::degrees_per_second_from_rpm(int):
    sll    $2,$4,1
    addu   $2,$2,$4
    jr     $31
    sll    $2,$2,1

nholthaus::degrees_per_second_from_rpm(int):
    addiu  $sp,$sp,-32
    sw     $31,28($sp)
    jal    __floatsidf
    nop

    lui    $4,%hi($L0)
    lw    $7,%lo($L0+4)($4)
    lw    $6,%lo($L0)($4)
    move   $5,$3
    move   $4,$2
    jal    __muldf3
    nop

    move   $5,$3
    move   $4,$2
    jal    __fixdfs
    nop

    lw    $31,28($sp)
    nop
    jr    $31
    addiu $sp,$sp,32
```

```
with_au::degrees_per_second_from_rpm(int):
    sll    $2,$4,1
    addu   $2,$2,$4
    jr     $31
    sll    $2,$2,1
```

# Conversion safety

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3  
4  
5  
6  
7  
8  
9
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5  
6  
7  
8  
9
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8  
9
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8 using chrono_minutes_double = std::chrono::duration<double, std::ratio<60>>;  
9
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8 using chrono_minutes_double = std::chrono::duration<double, std::ratio<60>>;  
9 auto dt_min = chrono_minutes_double{dt_s}; // 0.5 min
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8 using chrono_minutes_double = std::chrono::duration<double, std::ratio<60>>;  
9 auto dt_min = chrono_minutes_double{dt_s}; // 0.5 min
```

```
using nanos_u32 = std::chrono::duration<uint32_t, std::nano>;  
const auto five_sec = nanos_u32{std::chrono::seconds{5}}; // 0.705'032'704 s (!)
```

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8 using chrono_minutes_double = std::chrono::duration<double, std::ratio<60>>;  
9 auto dt_min = chrono_minutes_double{dt_s}; // 0.5 min
```

```
using nanos_u32 = std::chrono::duration<uint32_t, std::nano>;  
const auto five_sec = nanos_u32{std::chrono::seconds{5}}; // 0.705'032'704 s (!)
```



QuantityU32<Nano<Seconds>> dt = seconds(5);

# Conversion safety

```
1 auto dt_s = std::chrono::seconds{30};  
2  
3 auto dt_ms = std::chrono::milliseconds{dt_s}; // 30'000 ms  
4  
5 // Does not compile!  
6 // auto dt_min = std::chrono::minutes{dt_s};  
7  
8 using chrono_minutes_double = std::chrono::duration<double, std::ratio<60>>;  
9 auto dt_min = chrono_minutes_double{dt_s}; // 0.5 min
```

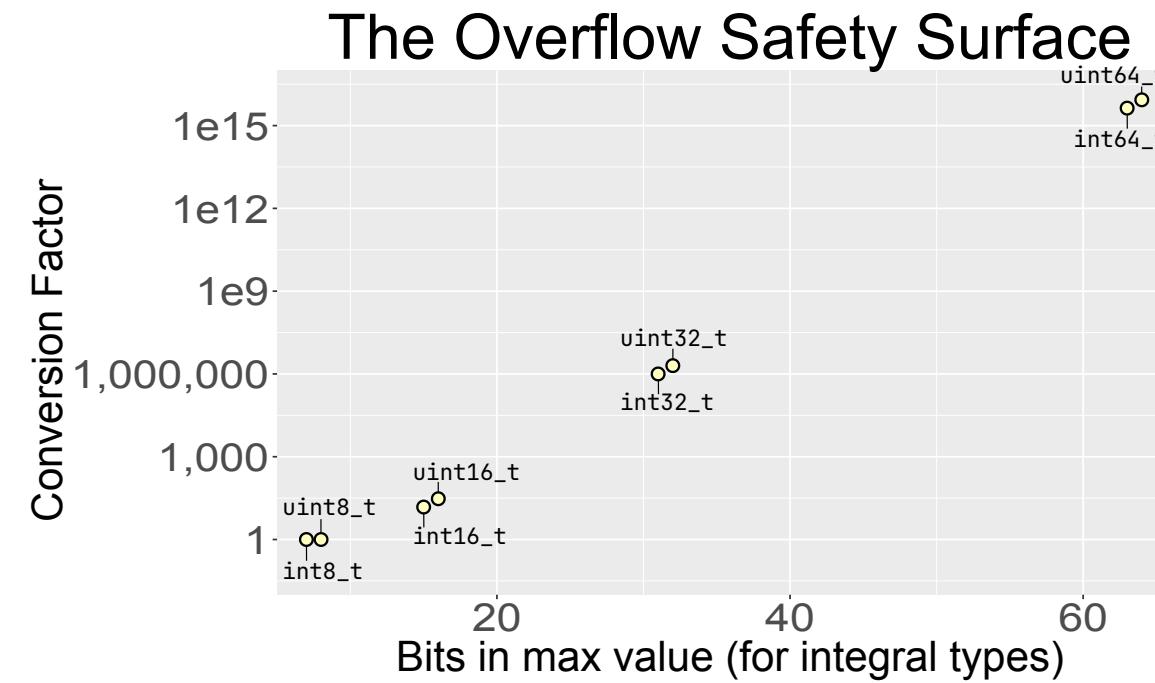
```
using nanos_u32 = std::chrono::duration<uint32_t, std::nano>;  
const auto five_sec = nanos_u32{std::chrono::seconds{5}}; // 0.705'032'704 s (!)
```



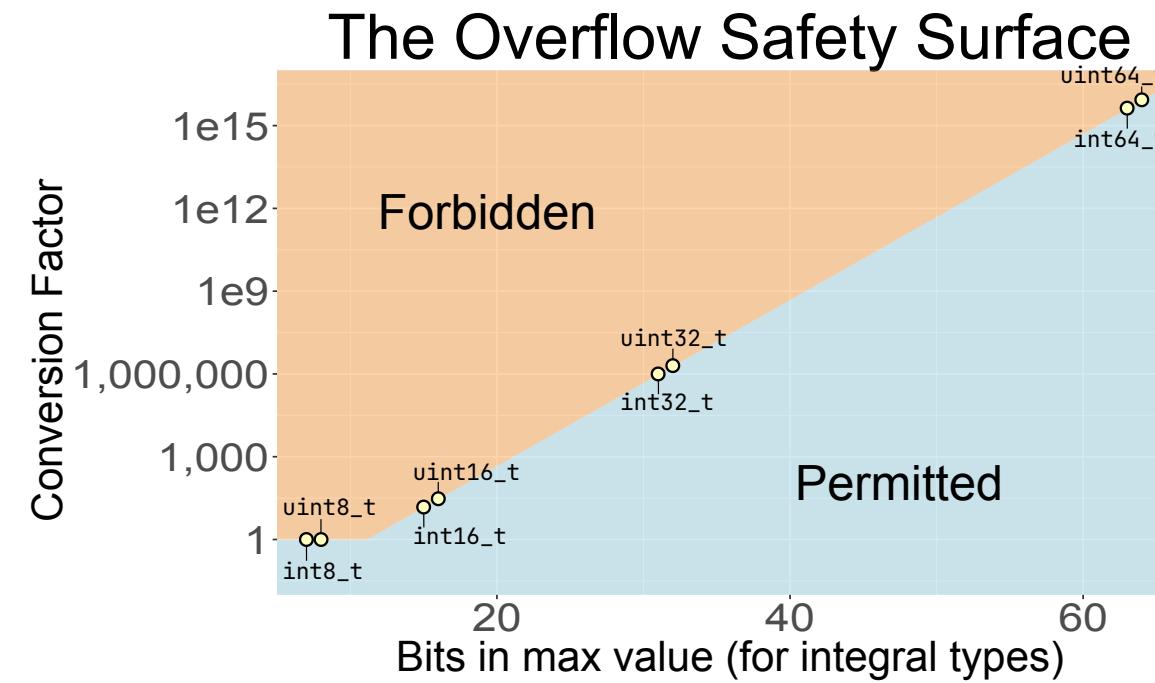
```
QuantityU32<Nano<Seconds>> dt = seconds(5);
```

```
error: conversion from 'Quantity<au::Seconds,int>' to non-scalar type  
'Quantity<au::Nano<au::Seconds>,unsigned int>' requested
```

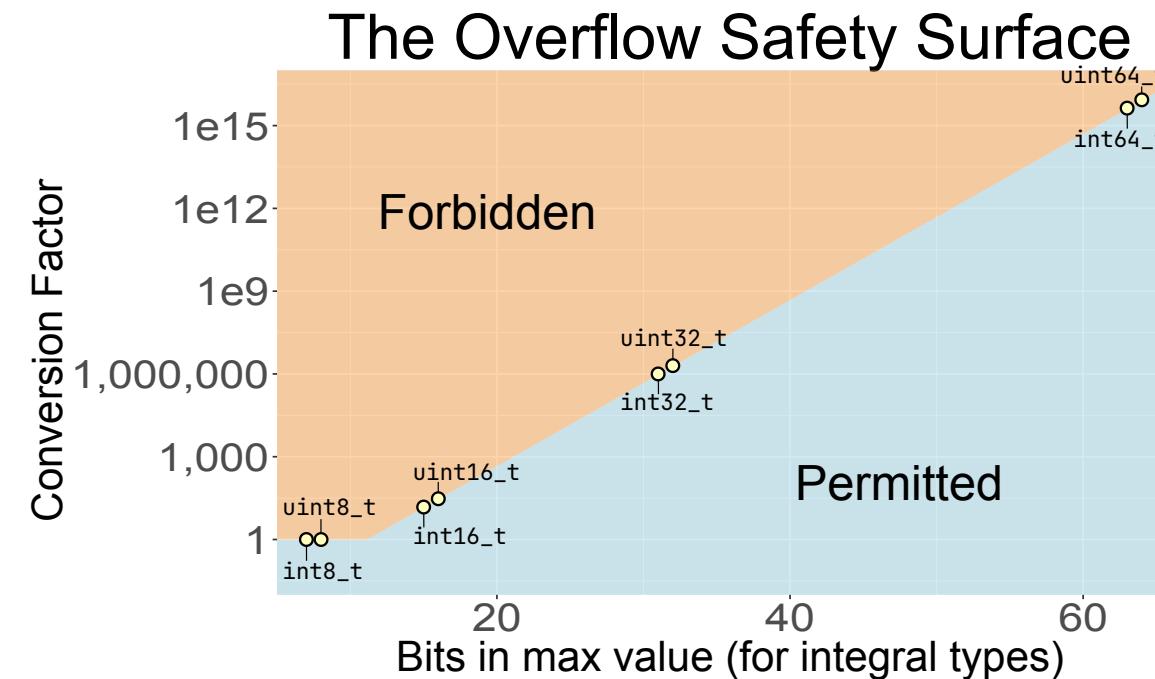
# The “Safety Surface”



# The “Safety Surface”



# The “Safety Surface”



## Protection

Truncation



nholthaus



mp-units



Overflow



# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and rational powers.

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

**std::ratio**:  $\frac{N}{D}$       Vector space magnitudes:  $\prod_i b_i^{p_i}$

---

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

**std::ratio**:  $\frac{N}{D}$       Vector space magnitudes:  $\prod_i b_i^{p_i}$

---

$$\left( \frac{\text{AU}}{\text{m}} \right) M = \frac{149\,597\,870\,700}{1} \quad M = 2^2 \cdot 3^1 \cdot 5^2 \cdot 73^1 \cdot 877^1 \cdot 7789^1$$

---

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

**std::ratio:**  $\frac{N}{D}$       Vector space magnitudes:  $\prod_i b_i^{p_i}$

---

$$\left(\frac{\text{AU}}{\text{m}}\right) \quad M = \frac{149\,597\,870\,700}{1} \quad M = 2^2 \cdot 3^1 \cdot 5^2 \cdot 73^1 \cdot 877^1 \cdot 7789^1$$

---

$$\left(\frac{\text{AU}}{\text{m}}\right)^2 \quad M = \text{🚫} \quad M = 2^4 \cdot 3^2 \cdot 5^4 \cdot 73^2 \cdot 877^2 \cdot 7789^2$$

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

**std::ratio:**  $\frac{N}{D}$       Vector space magnitudes:  $\prod_i b_i^{p_i}$

---

$$\left(\frac{\text{AU}}{\text{m}}\right) \quad M = \frac{149\,597\,870\,700}{1} \quad M = 2^2 \cdot 3^1 \cdot 5^2 \cdot 73^1 \cdot 877^1 \cdot 7789^1$$

---

$$\left(\frac{\text{AU}}{\text{m}}\right)^2 \quad M = \text{🚫} \quad M = 2^4 \cdot 3^2 \cdot 5^4 \cdot 73^2 \cdot 877^2 \cdot 7789^2$$

$\sqrt{2}$

$M =$  

$2\frac{1}{2}$

---

# Vector space magnitudes

[https://aurora-opensource.github.io/au/main/discussion/implementation/vector\\_space/#magnitude](https://aurora-opensource.github.io/au/main/discussion/implementation/vector_space/#magnitude)

**Requirement:** *magnitudes* must do everything *units* can do!

And units...

- ...are *closed* under products and **rational powers**.
- ...support **irrational** values such as  $\pi$ .

**std::ratio:**  $\frac{N}{D}$       Vector space magnitudes:  $\prod_i b_i^{p_i}$

---

$$\left(\frac{\text{AU}}{\text{m}}\right) \quad M = \frac{149\,597\,870\,700}{1} \quad M = 2^2 \cdot 3^1 \cdot 5^2 \cdot 73^1 \cdot 877^1 \cdot 7789^1$$

---

$$\left(\frac{\text{AU}}{\text{m}}\right)^2 \quad M = \text{🚫} \quad M = 2^4 \cdot 3^2 \cdot 5^4 \cdot 73^2 \cdot 877^2 \cdot 7789^2$$

$$\sqrt{2}$$

$$M = \text{ } \cancel{\text{ }}$$

$$2\frac{1}{2}$$

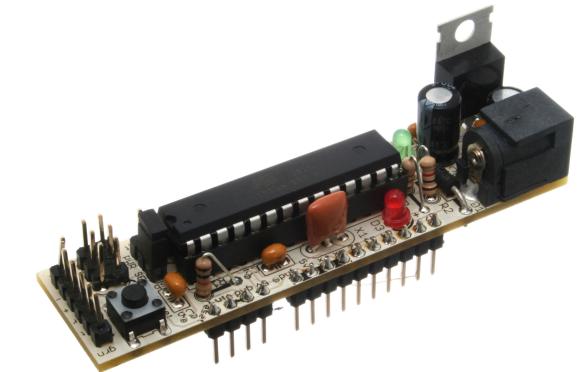
---

$$\frac{\text{deg}}{\text{rad}}$$

$$M = \text{ } \cancel{\text{ }}$$

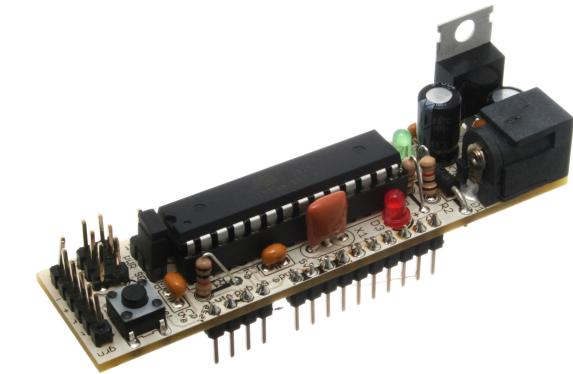
$$M = 2^{-2} \cdot 3^{-2} \cdot \pi^1 \cdot 5^{-1} (= \frac{\pi}{180})$$

# Embedded friendliness



By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

# Embedded friendliness

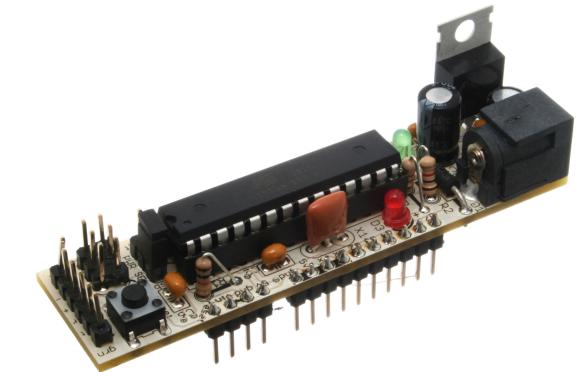


By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

Tired:

Wired:

# Embedded friendliness



By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

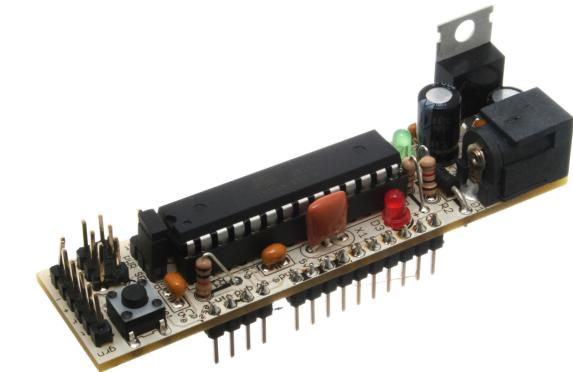
Tired:

- double, float

Wired:

- int64\_t, uint32\_t, ...

# Embedded friendliness



By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

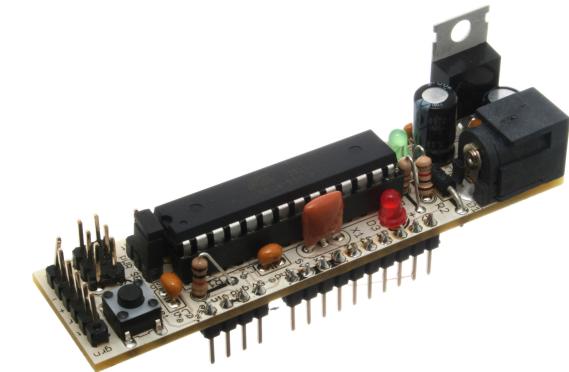
Tired:

- double, float
- <iostream> 😞 😱 😴

Wired:

- int64\_t, uint32\_t, ...
- const char[]

# Embedded friendliness



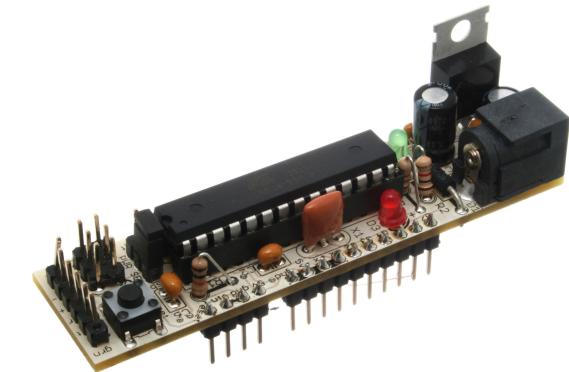
By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

Tired:

Wired:

```
const auto &label = unit_label(Meters{} * Kilo<Grams>{} / squared(Seconds{}));
```

# Embedded friendliness



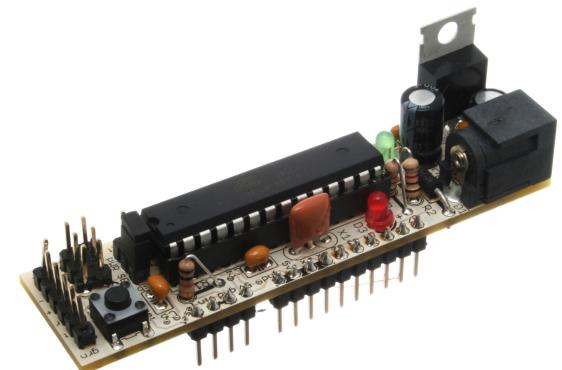
By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

Tired:

Wired:

```
const auto &label = unit_label(Meters{} * Kilo<Grams>{} / squared(Seconds{}));  
  
static_assert(std::is_same<decltype(label), const char(&)[15]>::value, "");  
// ^^^^^^
```

# Embedded friendliness



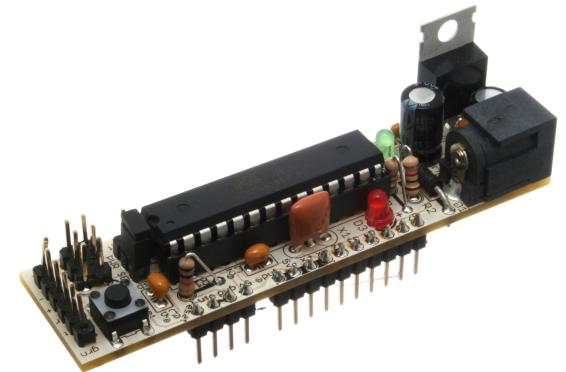
By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

# Tired.

# Wired:

```
const auto &label = unit_label(Meters{} * KiloGrams{} / squared(Seconds{}));  
  
static_assert(std::is_same<decltype(label), const char(&)[15]>::value, "");  
//  
  
printf("Label of length %d is: [%s]\n", sizeof(label), label);  
//
```

# Embedded friendliness



By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

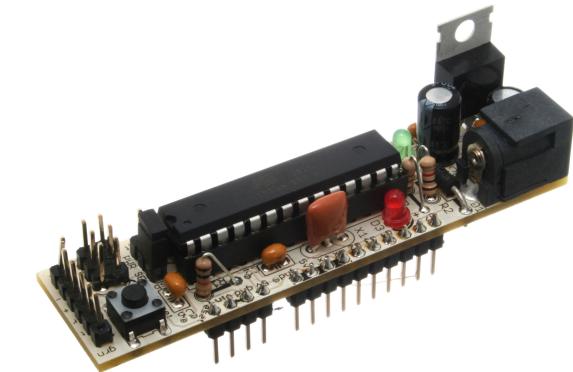
# Tired.

# Wired:

```
const auto &label = unit_label(Meters{} * KiloGrams{} / squared(Seconds{}));  
  
static_assert(std::is_same<decltype(label), const char(&)[15]>::value, "");  
//  
  
printf("Label of length %d is: [%s]\n", sizeof(label), label);  
//
```

Label of length 15 is: [ (m \* kg) / s^2]

# Embedded friendliness



By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

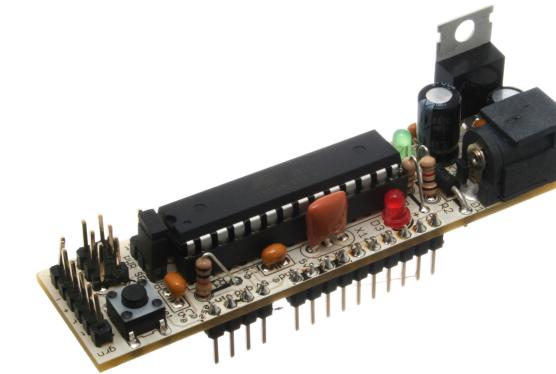
Tired:

- double, float
- <iostream> 😰 😱 😴

Wired:

- int64\_t, uint32\_t, ...
- const char[]

# Embedded friendliness



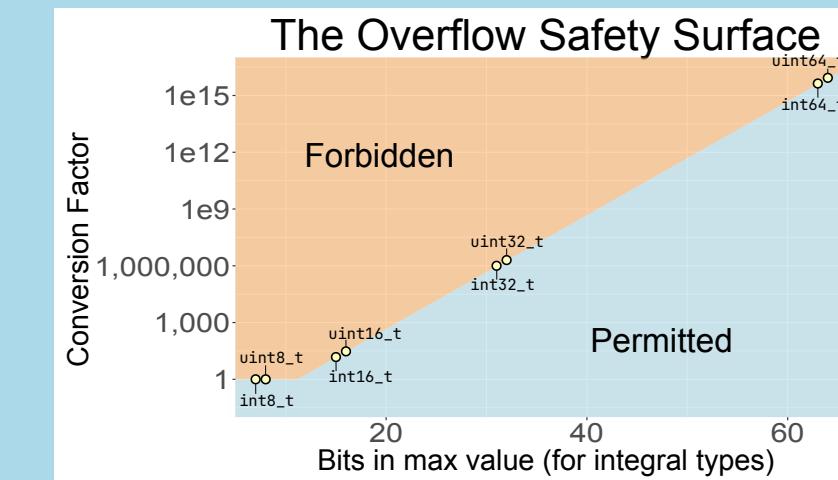
By oomlout (Flickr), CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:ADAF-03-BRD\\_\(14336910552\).jpg](https://commons.wikimedia.org/wiki/File:ADAF-03-BRD_(14336910552).jpg)

Tired:

- double, float
- <iostream> 😰 😱 😴

Wired:

- int64\_t, uint32\_t, ...
- const char[]



# Composability

Code:

Output:

# Composability

Code:

```
std::cout << meters(10) << std::endl;
```

Output:

```
10 m
```

# Composability

Code:

```
std::cout << meters(10) << std::endl;  
std::cout << (meters / second)(20) << std::endl;
```

Output:

```
10 m  
20 m / s
```

# Composability

Code:

```
std::cout << meters(10) << std::endl;
std::cout << (meters / second)(20) << std::endl;
std::cout << (kilo(meters) / hour)(30) << std::endl;
```

Output:

```
10 m
20 m / s
30 km / h
```

# Composability

Code:

```
std::cout << meters(10) << std::endl;
std::cout << (meters / second)(20) << std::endl;
std::cout << (kilo(meters) / hour)(30) << std::endl;
std::cout << (meters / squared(second))(40) << std::endl;
```

Output:

```
10 m
20 m / s
30 km / h
40 m / s^2
```

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s}$$

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

$$1 = Tf$$

$$= (2500 [\mu\text{s}])(400[\text{Hz}])$$

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

$$\begin{aligned} 1 &= Tf \\ &= (2500 [\mu\text{s}])(400[\text{Hz}]) \\ &= 1\,000\,000[\mu\text{s} \cdot \text{Hz}] \end{aligned}$$

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

$$\begin{aligned} 1 &= Tf \\ &= (2500 [\mu\text{s}])(400[\text{Hz}]) \\ &= 1\,000\,000[\mu\text{s} \cdot \text{Hz}] \\ &= 1\,000\,000[1 / 1\,000\,000] \end{aligned}$$

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

$$1 = Tf$$

$$= (2500 [\mu\text{s}])(400[\text{Hz}])$$

$$= 1000000[\mu\text{s} \cdot \text{Hz}]$$

$$= 1000000[1/1000000]$$

$$\therefore T_{\mu\text{s}}[\mu\text{s}] = (1000000[1/1000000])/(400[\text{Hz}])$$

— — —

# Unit-aware inverses

<https://aurora-opensource.github.io/au/main/reference/math/#inverse-functions>

$$f = 400 \text{ Hz}$$

$$T = 1/f = 0.0025 \text{ s} = 2500 \mu\text{s}$$

Solve for (units of) 1...

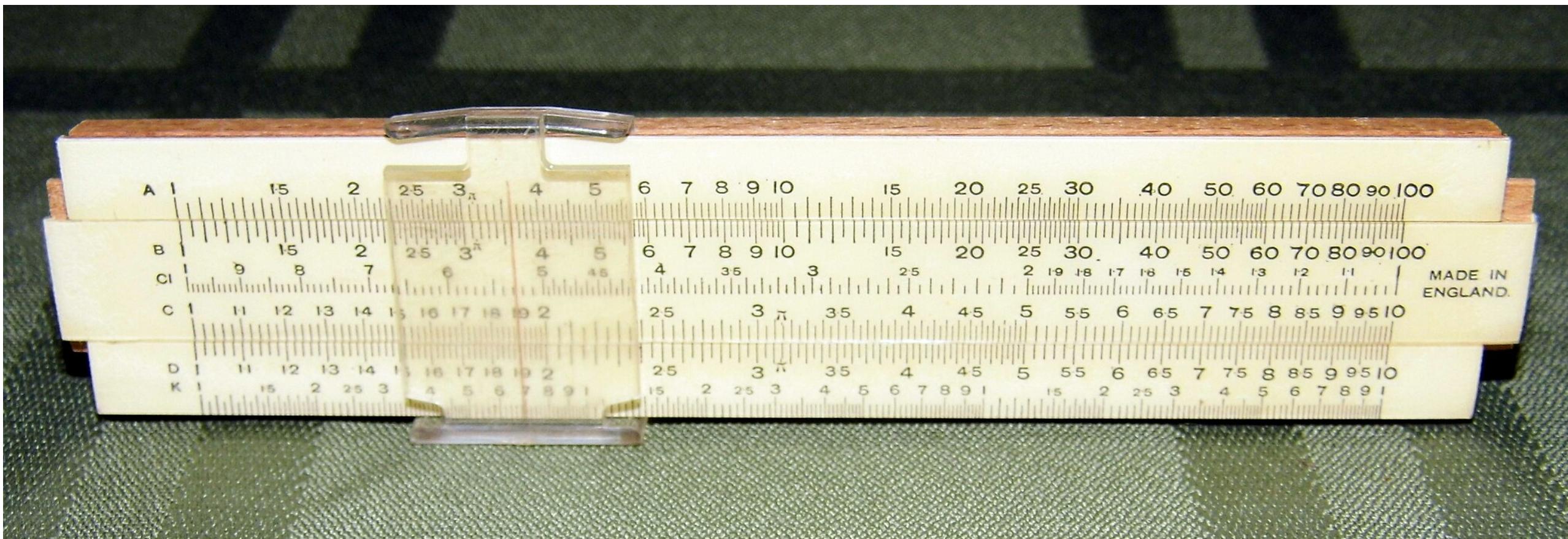
$$\begin{aligned} 1 &= Tf \\ &= (2500 [\mu\text{s}])(400[\text{Hz}]) \\ &= 1\,000\,000[\mu\text{s} \cdot \text{Hz}] \\ &= 1\,000\,000[1/1\,000\,000] \end{aligned}$$

$$\therefore T_{\mu\text{s}}[\mu\text{s}] = (1\,000\,000[1/1\,000\,000])/(400[\text{Hz}])$$

```
inverse_as(micro(seconds), hertz(400)); // micro(seconds)(2'500)
```

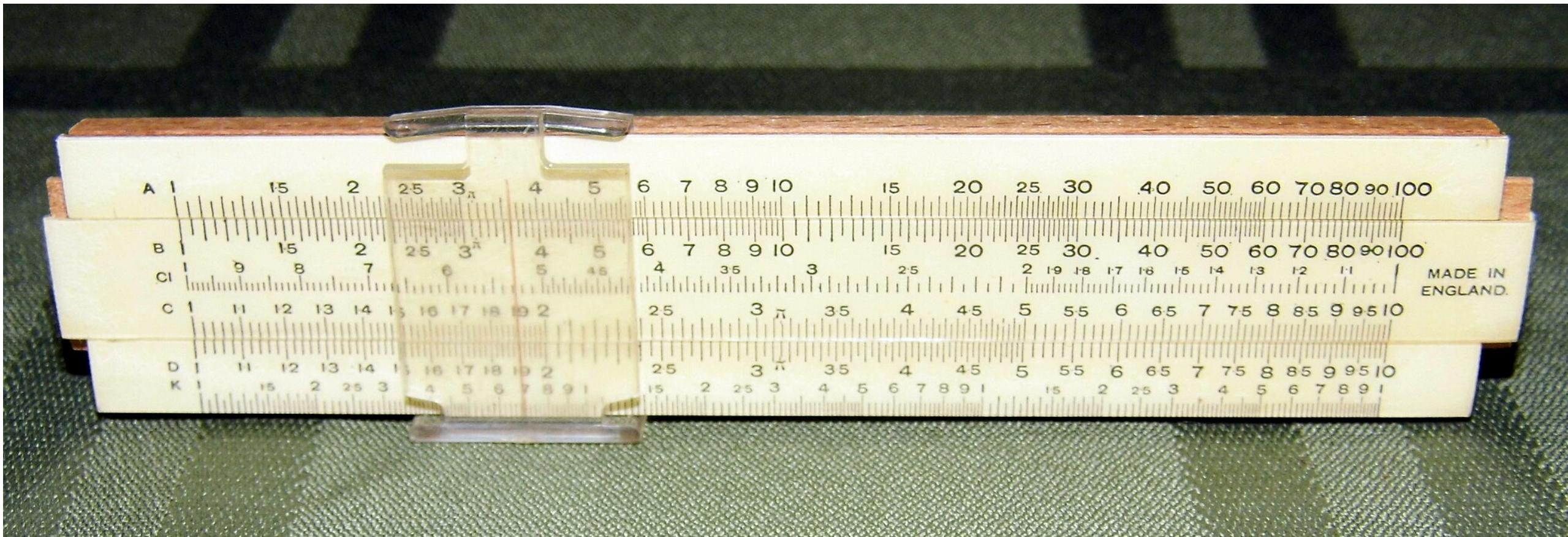
# Au: missing features

# Decibels



By Joe Haupt, CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:Vintage\\_Small\\_Slide\\_Rule,\\_4.75\\_Inches\\_in\\_Length,\\_Made\\_in\\_England\\_\(9610232930\).jpg](https://commons.wikimedia.org/wiki/File:Vintage_Small_Slide_Rule,_4.75_Inches_in_Length,_Made_in_England_(9610232930).jpg)

# Decibels



By Joe Haupt, CC BY-SA 2.0 DEED, [https://commons.wikimedia.org/wiki/File:Vintage\\_Small\\_Slide\\_Rule,\\_4.75\\_Inches\\_in\\_Length,\\_Made\\_in\\_England\\_\(9610232930\).jpg](https://commons.wikimedia.org/wiki/File:Vintage_Small_Slide_Rule,_4.75_Inches_in_Length,_Made_in_England_(9610232930).jpg)

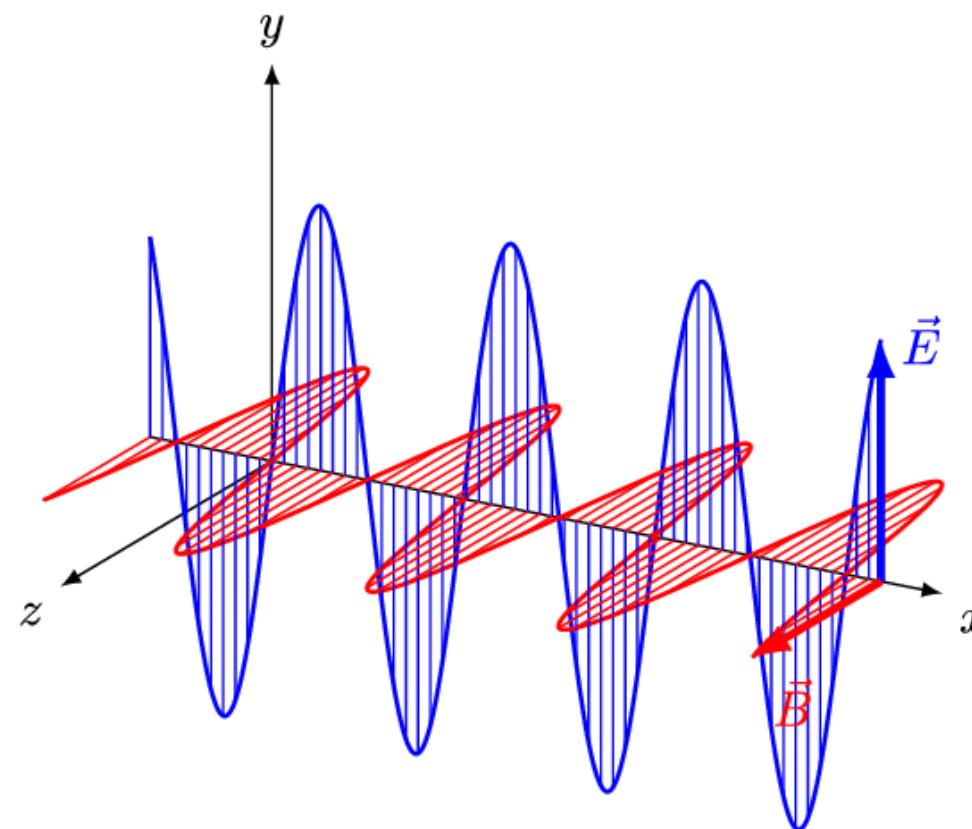
“Power” quantity:

“Root power” quantity:

$$L = \textcolor{red}{10} \log_{10} \left( \frac{X}{X_0} \right) \text{dB}$$

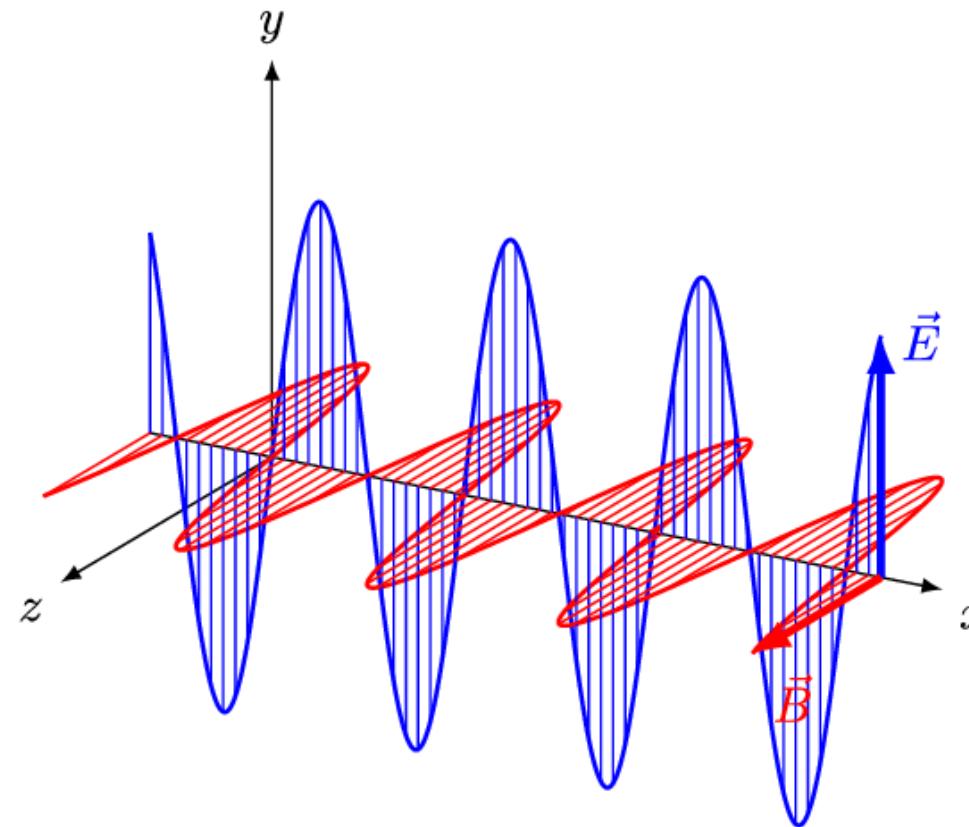
$$L = \textcolor{red}{20} \log_{10} \left( \frac{X}{X_0} \right) \text{dB}$$

# Explicit systems of quantities



By And1mu - Own work, CC BY-SA 4.0, [https://commons.wikimedia.org/wiki/File:Plane\\_electromagnetic\\_wave.svg](https://commons.wikimedia.org/wiki/File:Plane_electromagnetic_wave.svg)

# Explicit systems of quantities

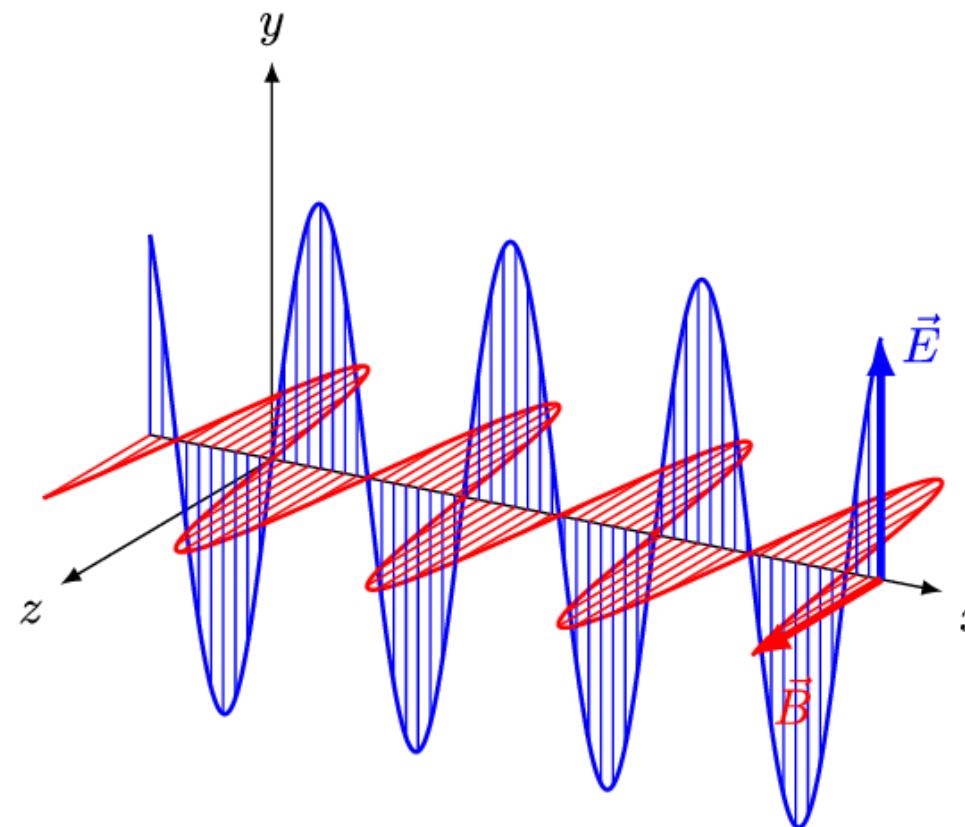


By And1mu - Own work, CC BY-SA 4.0, [https://commons.wikimedia.org/wiki/File:Plane\\_electromagnetic\\_wave.svg](https://commons.wikimedia.org/wiki/File:Plane_electromagnetic_wave.svg)

SI Units:

$$E = cB$$

# Explicit systems of quantities



By And1mu - Own work, CC BY-SA 4.0, [https://commons.wikimedia.org/wiki/File:Plane\\_electromagnetic\\_wave.svg](https://commons.wikimedia.org/wiki/File:Plane_electromagnetic_wave.svg)

SI Units:

$$E = cB$$

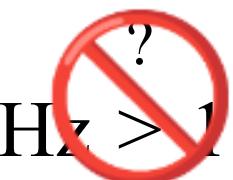
Gaussian Units:

$$E = B$$

# Quantity “kind”

$$1 \text{ Hz} \stackrel{?}{>} 1 \text{ Bq}$$

# Quantity “kind”

1 Hz  > 1 Bq

# Quantity “kind”

1 Hz  $\cancel{>} 1$  Bq



```
constexpr bool b = (hertz(1) > becquerel(1)); // Not meaningful, but compiles
```

# Quantity “kind”

1 Hz > 1 Bq



```
constexpr bool b = (hertz(1) > becquerel(1)); // Not meaningful, but compiles
```

mp-units

```
const bool b = (1 * si::hertz) > (1 * si::becquerel);
```

```
<source>:15:30: error: no match for 'operator>' (operand types are
'mp_units::quantity<mp_units::si::hertz(), int>' and
'mp_units::quantity<mp_units::si::becquerel(), int>')
15 |   const bool b = (1 * si::hertz) > (1 * si::becquerel);
|   ~~~~~^ ~~~~~
|       |
|           quantity<mp_units::si::becquerel(), [...]>
|           quantity<mp_units::si::hertz(), [...]>
```

# Aside: mistakes in the real world

**Tier**

---

**e.g.**

---

**Real  
World  
Errors**

# Aside: mistakes in the real world

**Tier**      Wrong dimension

---

e.g.

```
dist_m = time_s;
```

---

**Real  
World  
Errors**

# Aside: mistakes in the real world

Tier

Wrong dimension

Right dimension,  
Wrong magnitude

e.g.

```
dist_m = time_s;
```

```
dist_m = dist_cm;
```

Real  
World  
Errors

# Aside: mistakes in the real world

| Tier | Wrong dimension | Right dimension,<br>Wrong magnitude | Right dimension,<br>Right magnitude,<br>Wrong "kind" |
|------|-----------------|-------------------------------------|------------------------------------------------------|
|------|-----------------|-------------------------------------|------------------------------------------------------|

e.g.

```
dist_m = time_s;
```

```
dist_m = dist_cm;
```

```
freq_hz = act_bq
```

Real  
World  
Errors

# Aside: mistakes in the real world

| Tier | Wrong dimension | Right dimension,<br>Wrong magnitude | Right dimension,<br>Right magnitude,<br>Wrong "kind" |
|------|-----------------|-------------------------------------|------------------------------------------------------|
|------|-----------------|-------------------------------------|------------------------------------------------------|

e.g.

```
dist_m = time_s;
```

```
dist_m = dist_cm;
```

```
freq_hz = act_bq
```

## Real World Errors

### Quantity errors from P2981:

Mars Climate

Columbus

Vasa

Gimli glider

Black Sabbath

Korean Air 6316

Moorpark Zoo

Space Mountain

Wild rice

Med dose

# Aside: mistakes in the real world

| Tier | Wrong dimension | Right dimension,<br>Wrong magnitude | Right dimension,<br>Right magnitude,<br>Wrong "kind" |
|------|-----------------|-------------------------------------|------------------------------------------------------|
|------|-----------------|-------------------------------------|------------------------------------------------------|

e.g.

```
dist_m = time_s;
```

```
dist_m = dist_cm;
```

```
freq_hz = act_bq
```

Real  
World  
Errors

Mars Climate      Columbus  
Vasa      Gimli glider  
Black Sabbath  
Korean Air 6316  
Moorpark Zoo  
Space Mountain      Wild rice  
Med dose

Quantity errors from P2981:

# Unit symbol APIs (e.g., literals)

## Functions

```
meters(3.0)
```

## UDLs

```
3.0_m
```

# Unit symbol APIs (e.g., literals)

## Functions

```
meters(3.0)
```

```
(meters / second)(8.0)
```

## UDLs

```
3.0_m
```

```
8.0_mps // New manual definition 😞
```

# Unit symbol APIs (e.g., literals)

## Functions

```
meters(3.0)
```

```
(meters / second)(8.0)
```

```
meters(3.0f)
```

## UDLs

```
3.0_m
```

```
8.0_mps // New manual definition 😞
```

```
???
```

# Unit symbol APIs (e.g., literals)

## Functions

```
meters(3.0)
```

```
(meters / second)(8.0)
```

```
meters(3.0f)
```

```
meters(dist_m)
```

## UDLs

```
3.0_m
```

```
8.0_mps // New manual definition 😞
```

```
???
```

```
???
```

# Unit symbol APIs (e.g., literals)

## Functions

```
meters(3.0)
```

```
(meters / second)(8.0)
```

```
meters(3.0f)
```

```
meters(dist_m)
```

## UDLs

```
3.0_m
```

```
8.0_mps // New manual definition 😞
```

```
???
```

```
???
```

## Unit Symbols

```
3.0 * m
```

```
8.0 * m / s
```

```
3.0f * m
```

```
dist_m * m
```

# Inter-library interactions

# Feature inspirations

nholthaus



mp-units

- Directly Shared
- Inspired

# Feature inspirations



→ Directly Shared  
- - - → Inspired

# Feature inspirations



→ Directly Shared  
→ Inspired

# Feature inspirations



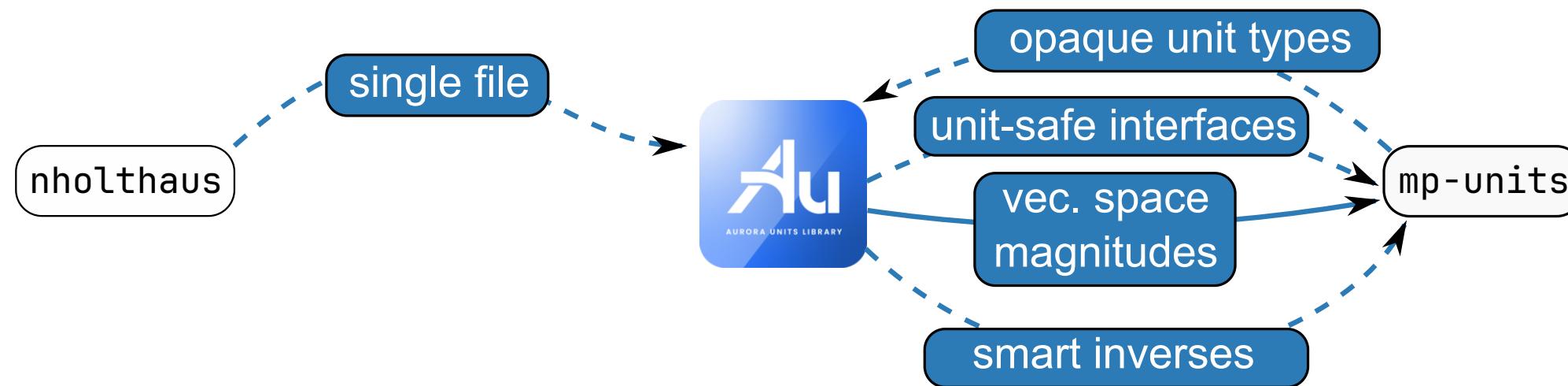
→ Directly Shared  
→ Inspired

# Feature inspirations



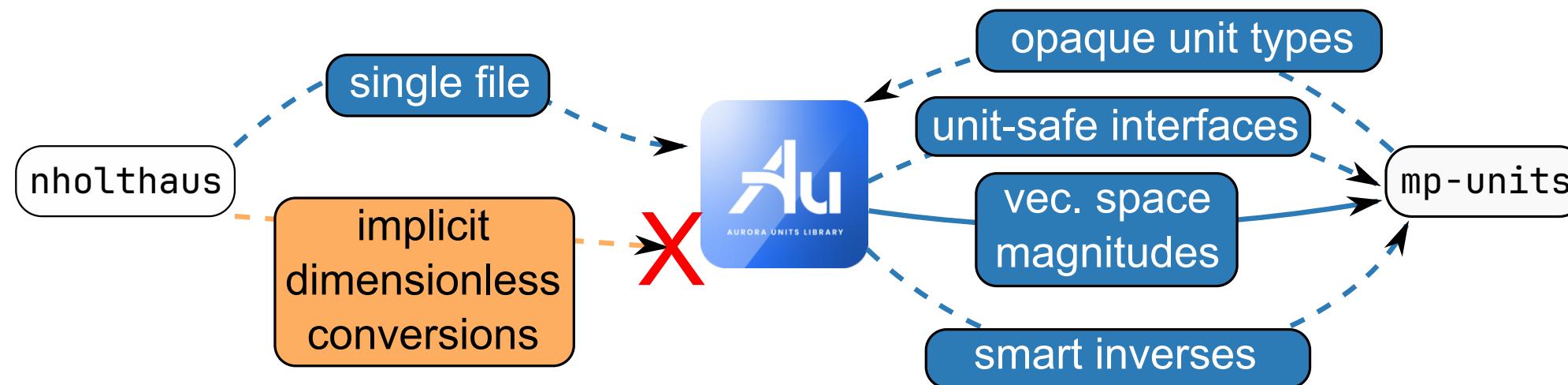
→ Directly Shared  
→ Inspired

# Feature inspirations



→ Directly Shared  
→ Inspired

# Feature inspirations

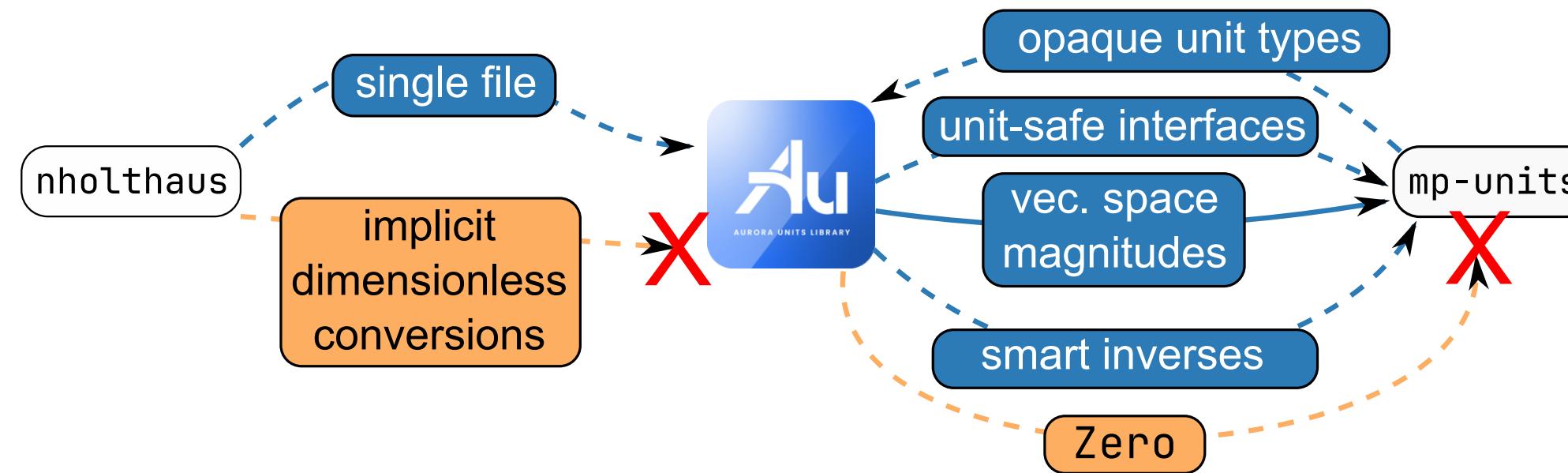


→ Directly Shared

→ Inspired

→ "Negatively" Inspired

# Feature inspirations



→ Directly Shared

→ Inspired

→ "Negatively" Inspired

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14 };  
15 } // namespace au
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4         using Unit = Meters;  
5     };  
6     // ...  
7     // ...  
8     // ...  
9     // ...  
10    // ...  
11    // ...  
12    // ...  
13    // ...  
14};  
15} // namespace au
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4         using Unit = Meters;  
5         using Rep = uint32_t;  
6     };  
7     struct CorrespondingQuantity<Quantity<MyMeters, uint32_t>> {  
8         using Unit = Meters;  
9         using Rep = uint32_t;  
10    };  
11    struct CorrespondingQuantity<Quantity<Quantity<MyMeters, uint32_t>, uint32_t>> {  
12        using Unit = Meters;  
13        using Rep = uint32_t;  
14    };  
15 } // namespace au
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4         using Unit = Meters;  
5         using Rep = uint32_t;  
6  
7         // Support Quantity construction from MyMeters.  
8         static constexpr Rep extract_value(MyMeters x) { return x.val; }  
9  
10  
11  
12  
13  
14     };  
15 } // namespace au
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

**<=>**

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4         using Unit = Meters;  
5         using Rep = uint32_t;  
6  
7         // Support Quantity construction from MyMeters.  
8         static constexpr Rep extract_value(MyMeters x) { return x.val; }  
9  
10        // Support Quantity conversion to MyMeters.  
11        static constexpr MyMeters construct_from_value(Rep x) {  
12            return {.value = x};  
13        }  
14    };  
15 } // namespace au
```

# Corresponding quantity mechanism

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

**<=>**

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

```
1 namespace au {  
2     template<>  
3     struct CorrespondingQuantity<MyMeters> {  
4         using Unit = Meters;  
5         using Rep = uint32_t;  
6  
7         // Support Quantity construction from MyMeters.  
8         static constexpr Rep extract_value(MyMeters x) { return x.val; }  
9  
10        // Support Quantity conversion to MyMeters.  
11        static constexpr MyMeters construct_from_value(Rep x) {  
12            return {.value = x};  
13        }  
14    };  
15 } // namespace au
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3  
4  
5
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4  
5
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters)(3'000)  
5
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters)(3'000)  
5 Quantity<Nano<Meters>, uint32_t> q3 = x; //==> Compiler error! Overflow safety
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters)(3'000)  
5 Quantity<Nano<Meters>, uint32_t> q3 = x; //==> Compiler error! Overflow safety
```

API passing:

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters)(3'000)  
5 Quantity<Nano<Meters>, uint32_t> q3 = x; //==> Compiler error! Overflow safety
```

API passing:

```
// Legacy interfaces:  
MyMeters measure_length(const Object& o);  
void evaluate(const MyMeters& m);
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

## Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters)(3'000)  
5 Quantity<Nano<Meters>, uint32_t> q3 = x; //==> Compiler error! Overflow safety
```

## API passing:

```
// Legacy interfaces:  
MyMeters measure_length(const Object& o);  
void evaluate(const MyMeters& m);
```

```
// Client code:  
MyMeters m = measure_length(my_object);  
evaluate(m);
```

# Cross-library interop

<https://aurora-opensource.github.io/au/main/howto/interop/>

```
struct MyMeters {  
    uint32_t val;  
};
```

<=>

```
class Quantity<Meters, uint32_t> {  
    uint32_t value_;  
};
```

## Initialization:

```
1 const MyMeters x{3};  
2  
3 Quantity<Meters, uint32_t> q1 = x; //==> meters(3)  
4 Quantity<Milli<Meters>, uint32_t> q2 = x; //==> milli(meters) (3'000)  
5 Quantity<Nano<Meters>, uint32_t> q3 = x; //==> Compiler error! Overflow safety
```

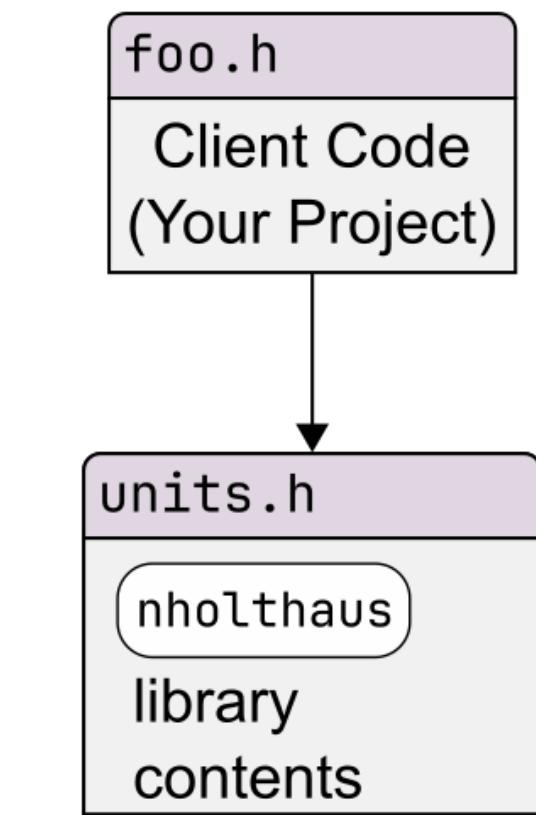
## API passing:

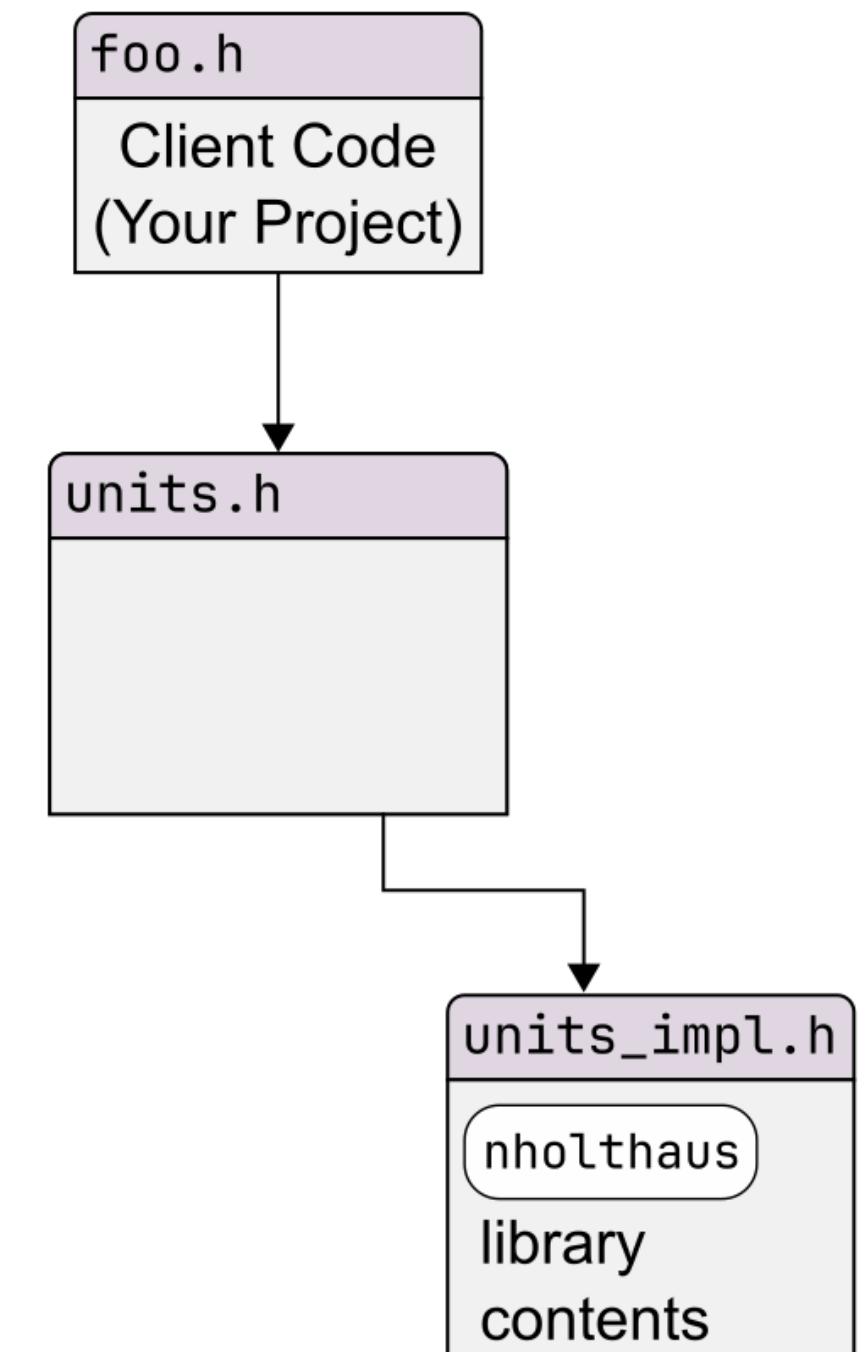
```
// After refactoring:  
Quantity<Meters, uint32_t> measure_length(const Object& o);  
void evaluate(const Quantity<Meters, uint32_t>& m);
```

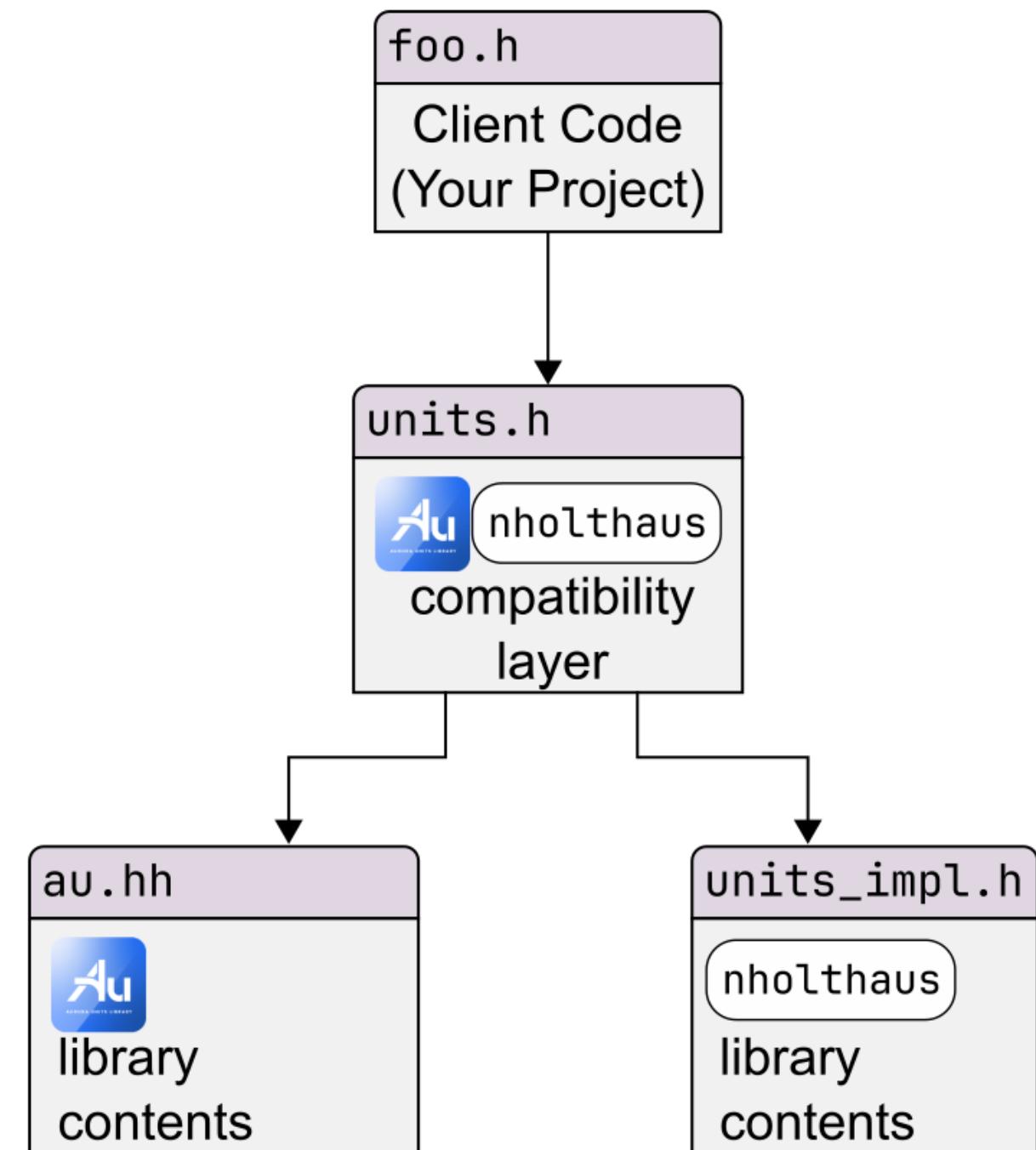
```
// Client code:  
MyMeters m = measure_length(my_object);  
evaluate(m);
```

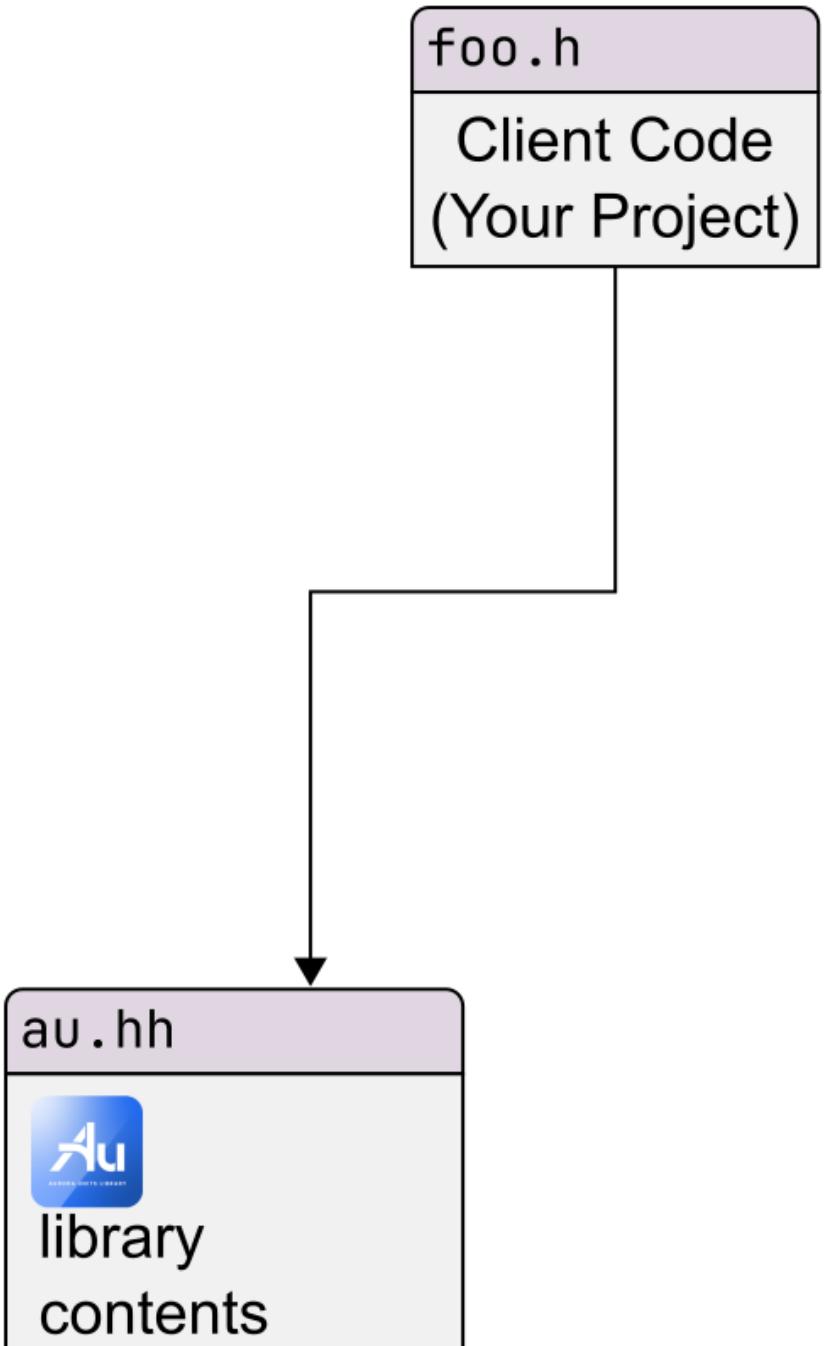
# nholthaus compatibility layer

<https://aurora-opensource.github.io/au/main/howto/interop/nholthaus/>











aurora-opensource/au