

# C++ MythBusters Strike 2

Victor Ciura



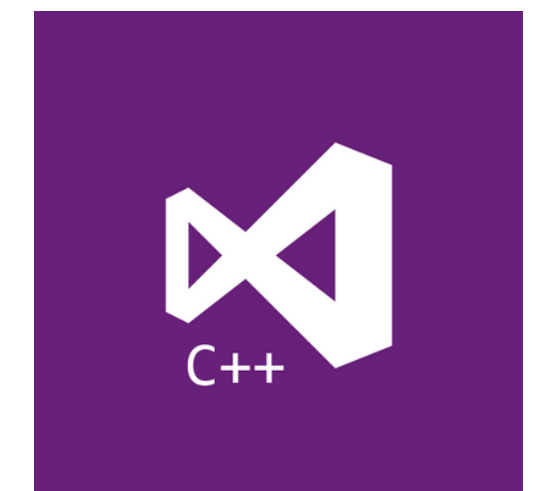


June 2023

 @ciura\_victor

 @ciura\_victor@hachyderm.io

**Victor Ciura**  
Principal Engineer  
Visual C++



# Abstract

The C++ community is very large and quite vocal when it comes to controversial issues. We're very fragmented on many topics, based on the breadth of the C++ ecosystem and the background/experience we each bring from our C++ niche.

From CppCoreGuidelines to opinionated best practices to established idioms, there's a lot of good information easily available. Mixed up with all of this there are also plenty of myths. Some myths stem from obsolete information, some from bad teaching materials.

In this presentation, I will dissect a few of the most popular C++ myths to a level of detail not possible on Twitter... and without the stigma of newb/duplicate/eyeroll one might experience when asking these questions on StackOverflow.

Expect the familiar “Busted”, “Plausible”, or “Confirmed” verdicts on each myth and come prepared to chat about these.

This is **Part 2** of the Mythbusters series.

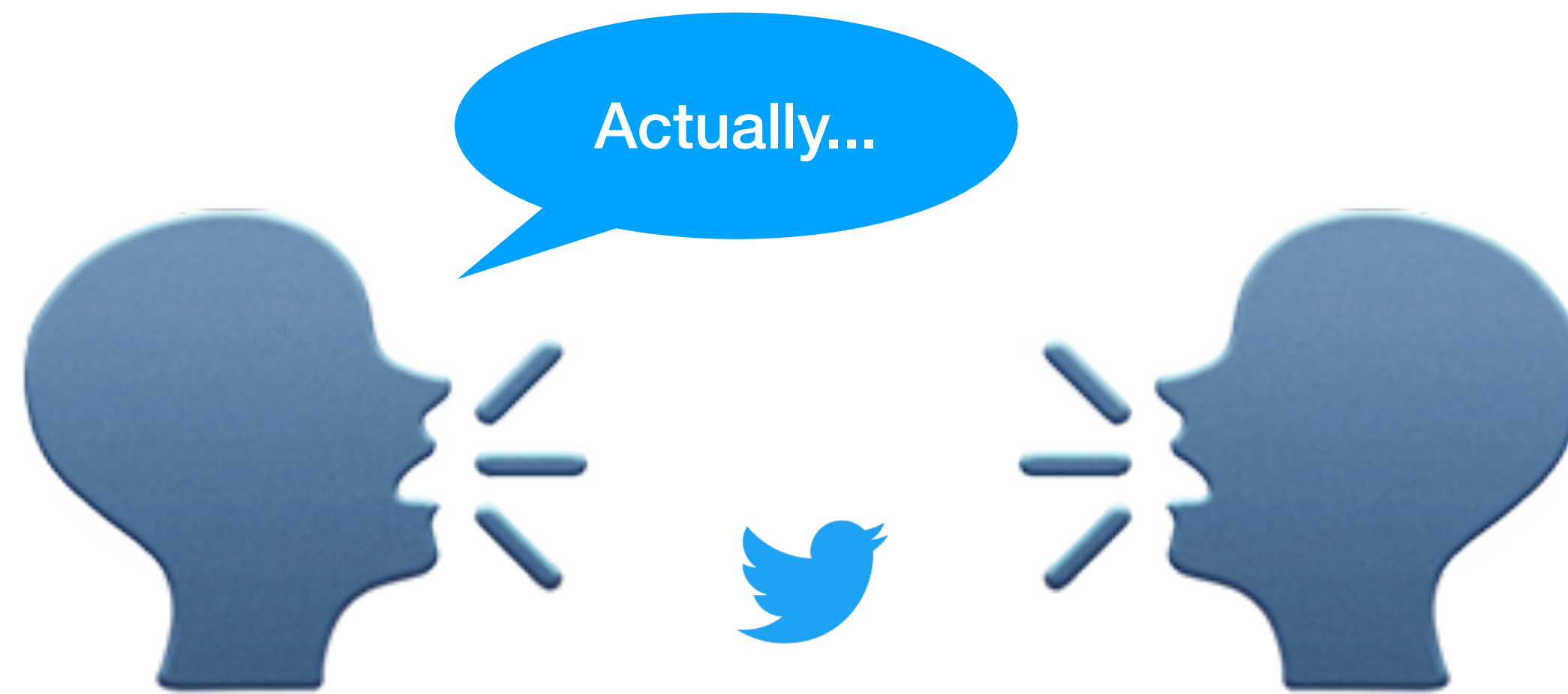


Do ask questions as we go along

Comments are welcome, too

# Actually, ...

The C++ community is very large and quite vocal when it comes to controversial issues



# Your opinion...



Developers love to treat their **opinions** like **facts**: "*This is the right way*"

No, that's just another way, with a different set of pros and cons.

-- David Fowler

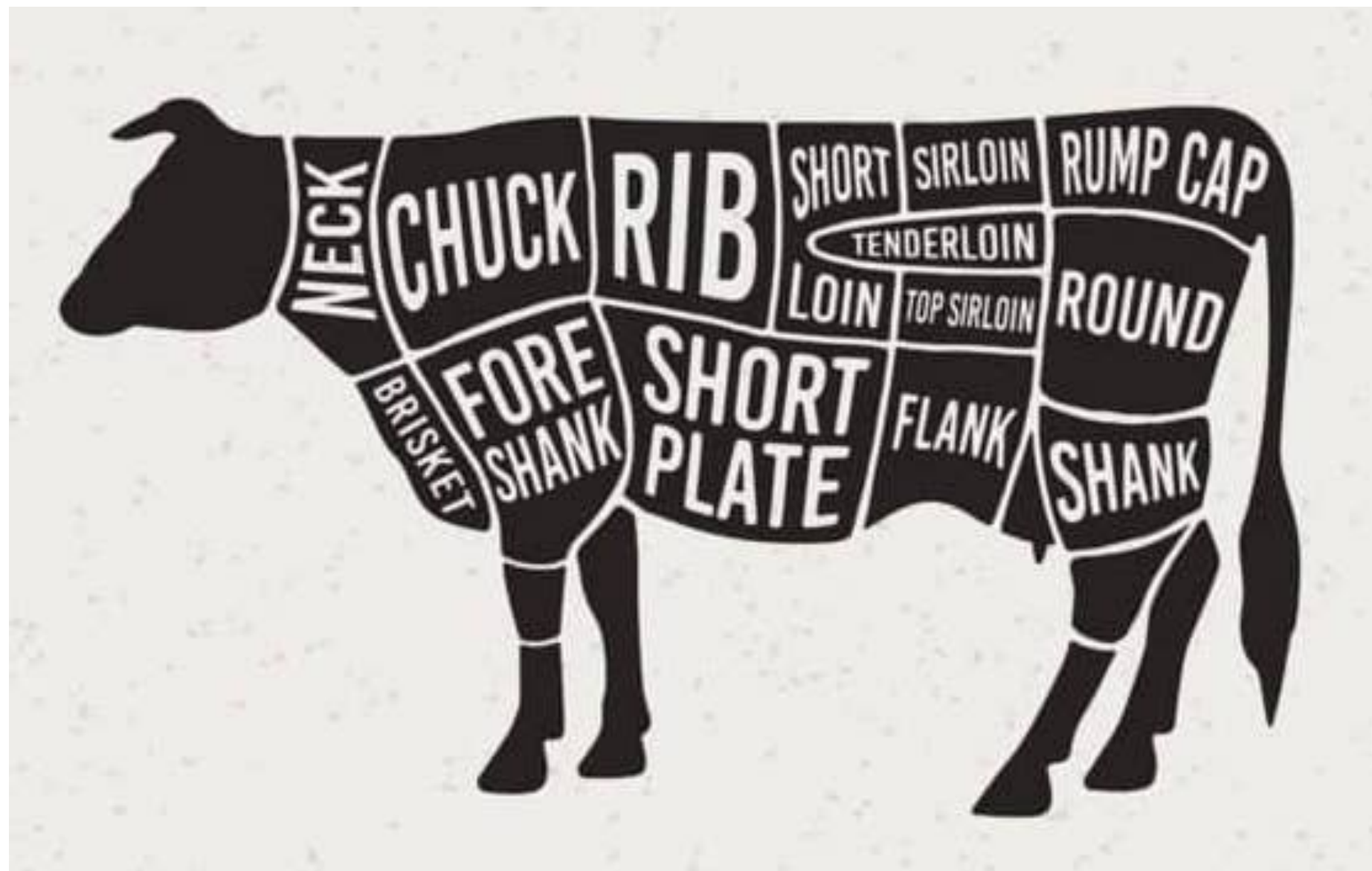
# We're Different

We're very **fragmented** on many topics

- based on the **breadth** of the C++ ecosystem
- background/experience we each bring from our C++ **niche**

# We're Different

We're very **fragmented** on many topics (Bjarne Stroustrup's 🐘 elephant metaphor)





A lot of **good** information easily available:

- CppCoreGuidelines
- (opinionated) best practices
- established idioms
- books
- conference presentations
- StackOverflow

Mixed up with all of this, there are also plenty of myths

- some myths stem from **obsolete** information
- some from bad **teaching** materials
- **old coding guidelines** in some projects
- onboarding C++ beginners on **legacy** C++ codebases (bad habits by example)



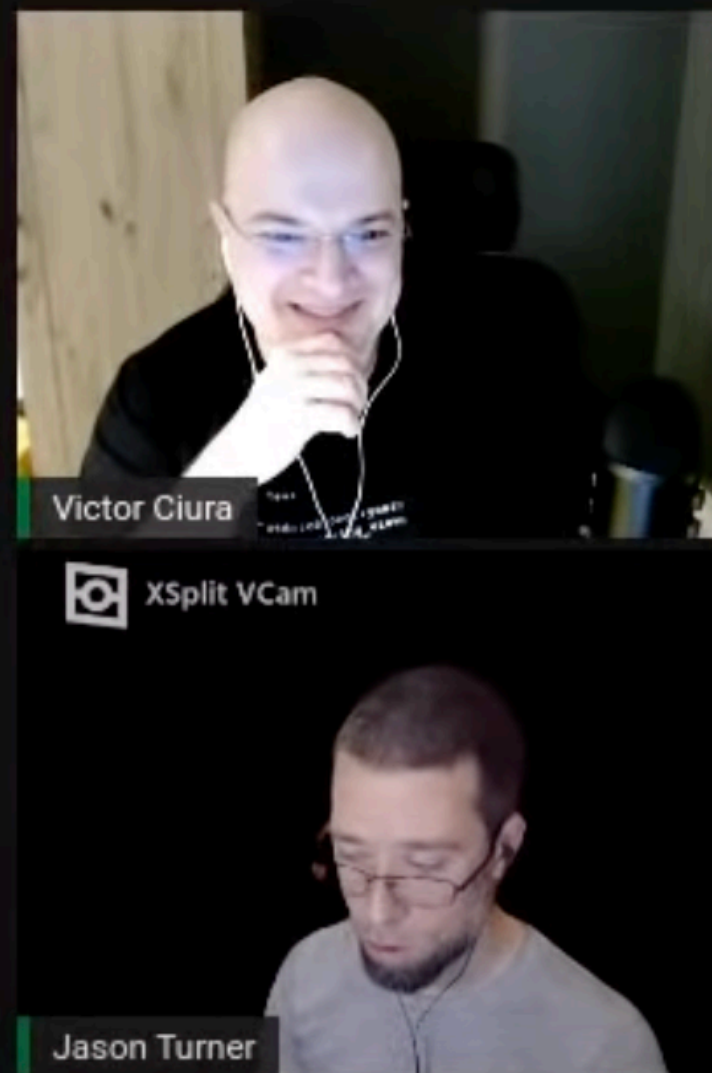
StackOverflow

How it started...

# Mythbusting with Jason - unscripted improv (Pandemic edition)

21k views

[youtube.com/watch?v=Bu1AEze14Ns](https://youtube.com/watch?v=Bu1AEze14Ns)



```
COMPILER EXPLORER
C++ source #1
Save/Load Add new... Vim CppInsights Quick-bench C++
1 #include <fmt/format.h>
2
3 #include <array>
4 #include <cstdint>
5 #include <optional>
6
7 // std::optional<>?
8
9 std::optional<std::string> get_optional_value(const bool something)
10 {
11     if (something) {
12         return "Hello World";
13     } else {
14         return std::nullopt;
15     }
16 }
17
18 std::size_t get_optional_string_size(const bool something) {
19     const auto optional_str = get_optional_value(something);
20     if (optional_str) {
21         return optional_str->size();
22     } else {
23         return std::string::npos;
24     }
25 }
26
27
x86-64 gcc (trunk) (Editor #1, Compiler #1) C++
x86-64 gcc (trunk) -std=c++20 -Wpedantic -Wall -Wextr.
Output... Filter... Libraries + Add new... Add tool...
7 .L5:
8     lea    rdx, [rdi+16]
9     mov    BYTE PTR [rdi+26], 100
10    movabs rcx, 8022916924116329800
11    mov    QWORD PTR [rdi], rdx
12    mov    edx, 27762
13    mov    QWORD PTR [rdi+16], rcx
14    mov    WORD PTR [rdi+24], dx
15    mov    QWORD PTR [rdi+8], 11
16    mov    BYTE PTR [rdi+27], 0
17    mov    BYTE PTR [rdi+32], 1
18    ret
19 get_optional_string_size(bool):
20    cmp    dil, 1
21    sbb   rax, rax
22    or    rax, 11
23    ret
Output (0/0) x86-64 gcc (trunk) - 3231ms (3198646)
#1 with x86-64 gcc (trunk)
Wrap lines
Compiler returned: 0
```



C++ Mythbusting with Victor and Jason

18,218 views • Streamed live on Jan 29, 2021

566 DISLIKE SHARE DOWNLOAD CLIP SAVE ...

Top chat replay  
for templates. I would like to require

# C++ Mythbusters - Season 1

**C++ MythBusters** 2022

## Myth #24

COMPILER EXPLORER

Get cool gear in the [Compiler Explorer shop](#) x [sponsors](#) Backtrace intel Solid Sands

```
1 #include <optional>
2 #include <cstdint>
3 #include <string>
4
5 std::optional<std::string> get_value(bool condition)
6 {
7     if (condition)
8         return "This is a longer string";
9     else
10        return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15     const auto str = get_value(condition);
16     if (str)
17         return str->size();
18     else
19         return std::string::npos;
20 }
21
22 int main()
23 {
24     return get_size(true);
25 }
```

no more SSO

compiler still sees through it and inlines it

```
35 sub    rsp, 56
36 mov    edi, 24
37 lea   rax, [rsp+16]
38 mov   QWORD PTR [rsp], rax
39 call  operator new(unsigned long)
40 mov   esi, 24
41 mov   BYTE PTR [rsp+32], 0
42 movdq xmm0, XMMWORD PTR .LC0[rip]
43 mov   DWORD PTR [rax+16], 1920234272
44 mov   rdi, rax
45 movups XMMWORD PTR [rax], xmm0
46 mov   QWORD PTR [rsp], rax
47 mov   eax, 28265
48 mov   WORD PTR [rdi+20], ax
49 mov   BYTE PTR [rdi+22], 103
50 mov   BYTE PTR [rdi+23], 0
51 mov   QWORD PTR [rsp+16], 23
52 mov   QWORD PTR [rsp+8], 23
53 call  operator delete(void*, unsigned long)
54 mov   eax, 23
```

Output (0/0) x86-64 gcc 11.2 - 2548ms (341058B) ~22151 lines filtered

Output of x86-64 gcc 11.2 (Compiler #1)

Victor Ciura

2022 Victor Ciura | @ciura\_victor - C++ MythBusters

23:29 / 50:16

cpsons.uk

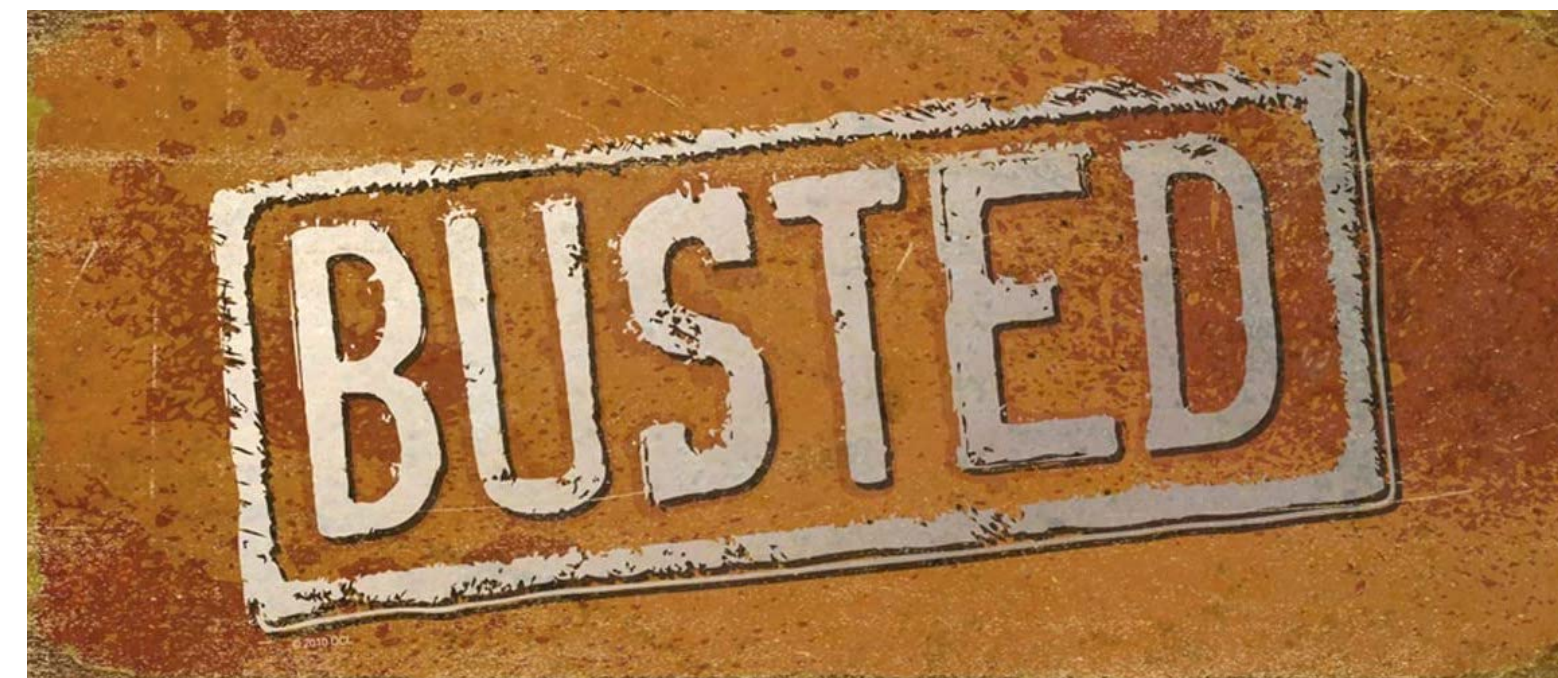
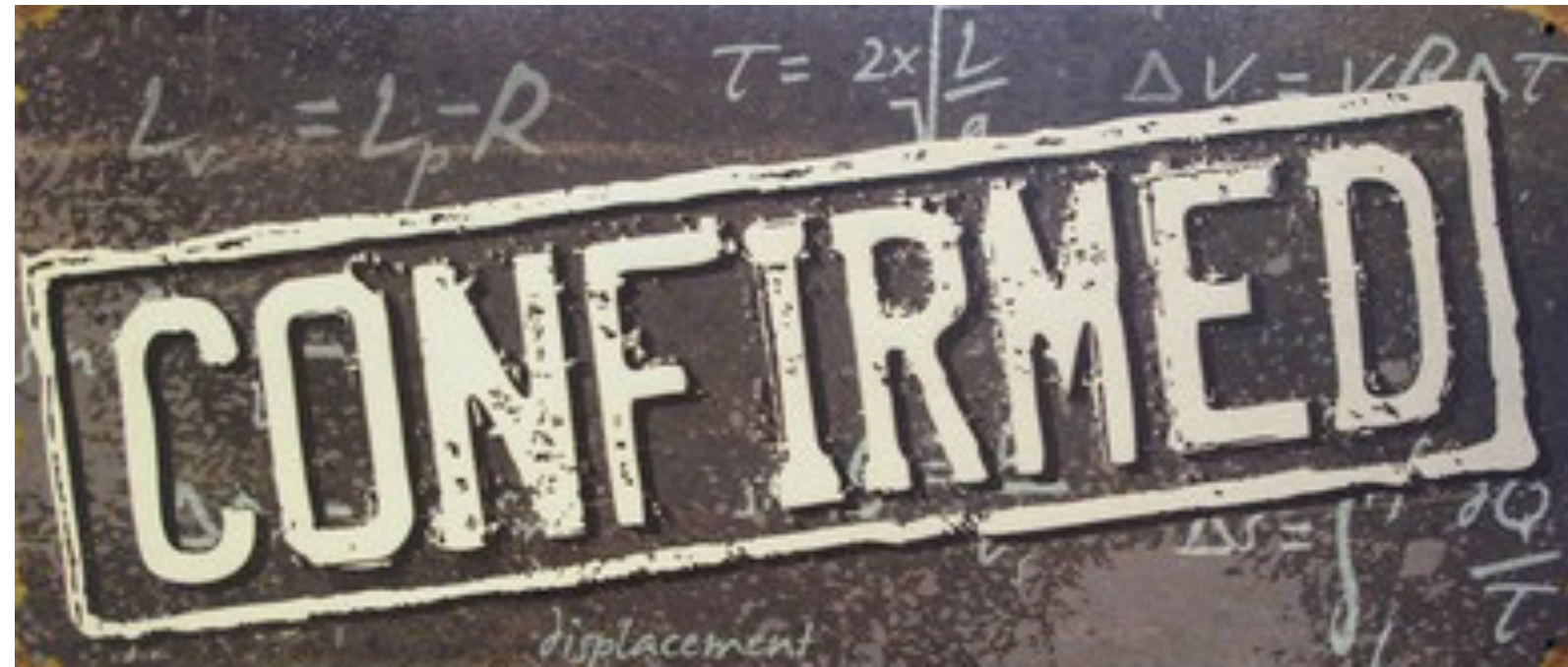
DeepMind

[youtube.com/watch?v=ZGgrUhVNsSI](https://youtube.com/watch?v=ZGgrUhVNsSI)



I want to instigate a healthy **dialog**,  
so speak up

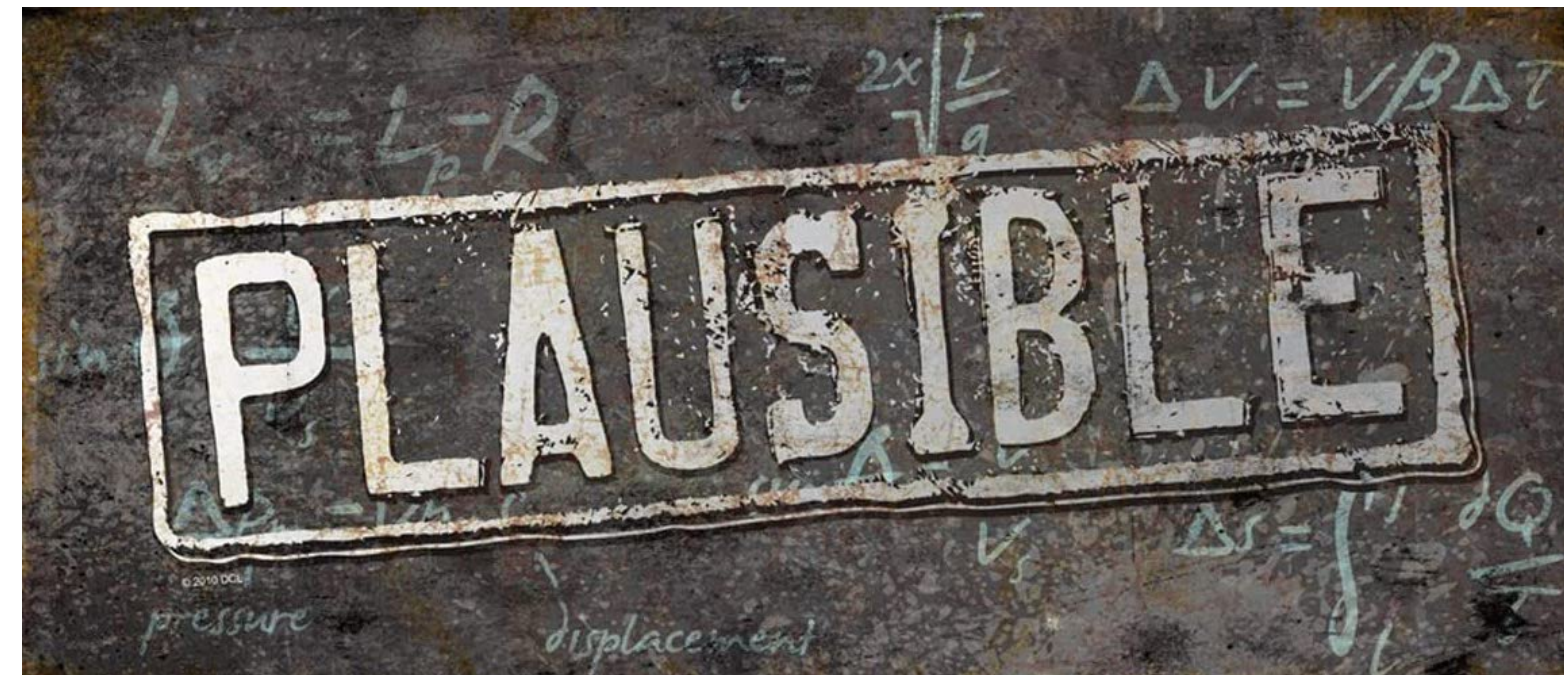
# Verdict





A programmer's staple response:

*"It depends..."* 🧐



Let's test this...



# Test Myth

C++ is inherently **unsafe** and there's very little\* we can do about it

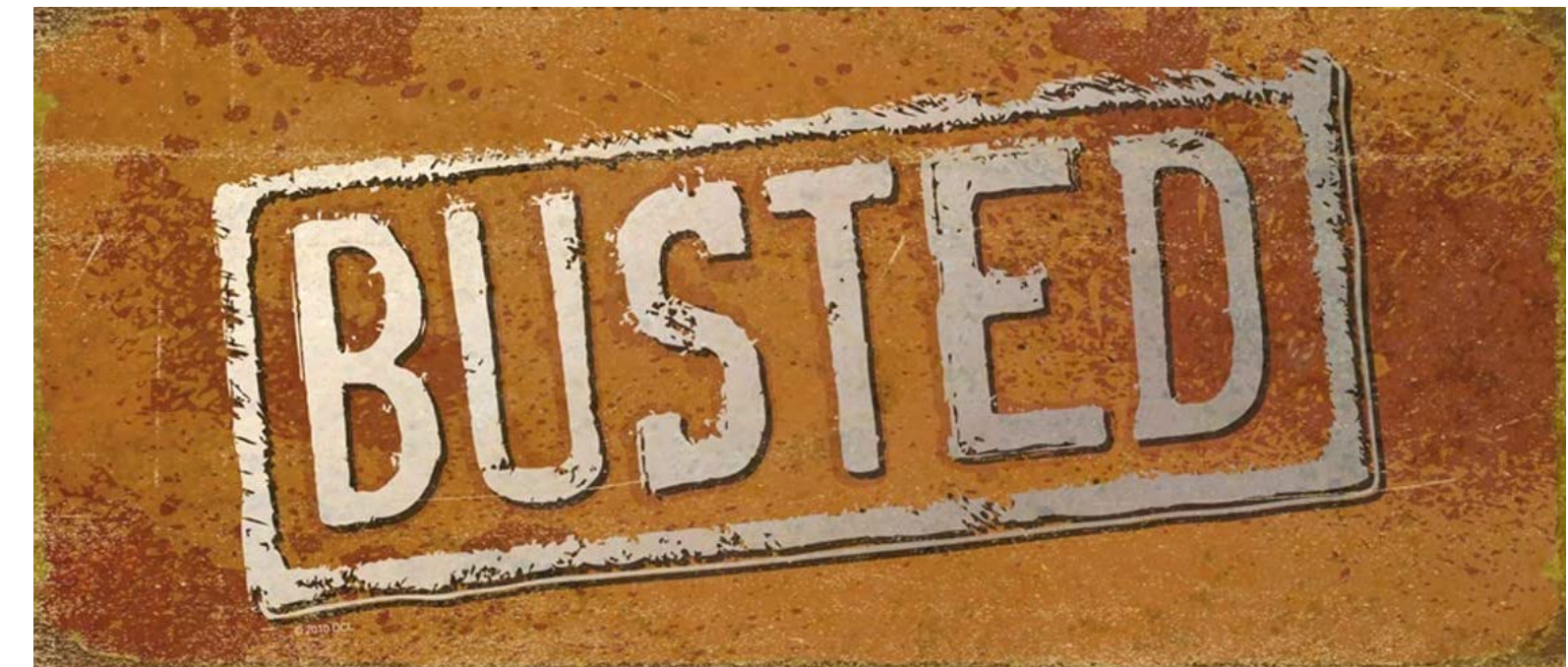
Just kidding 😊

It's not a myth, we've known this for years before **NSA**.



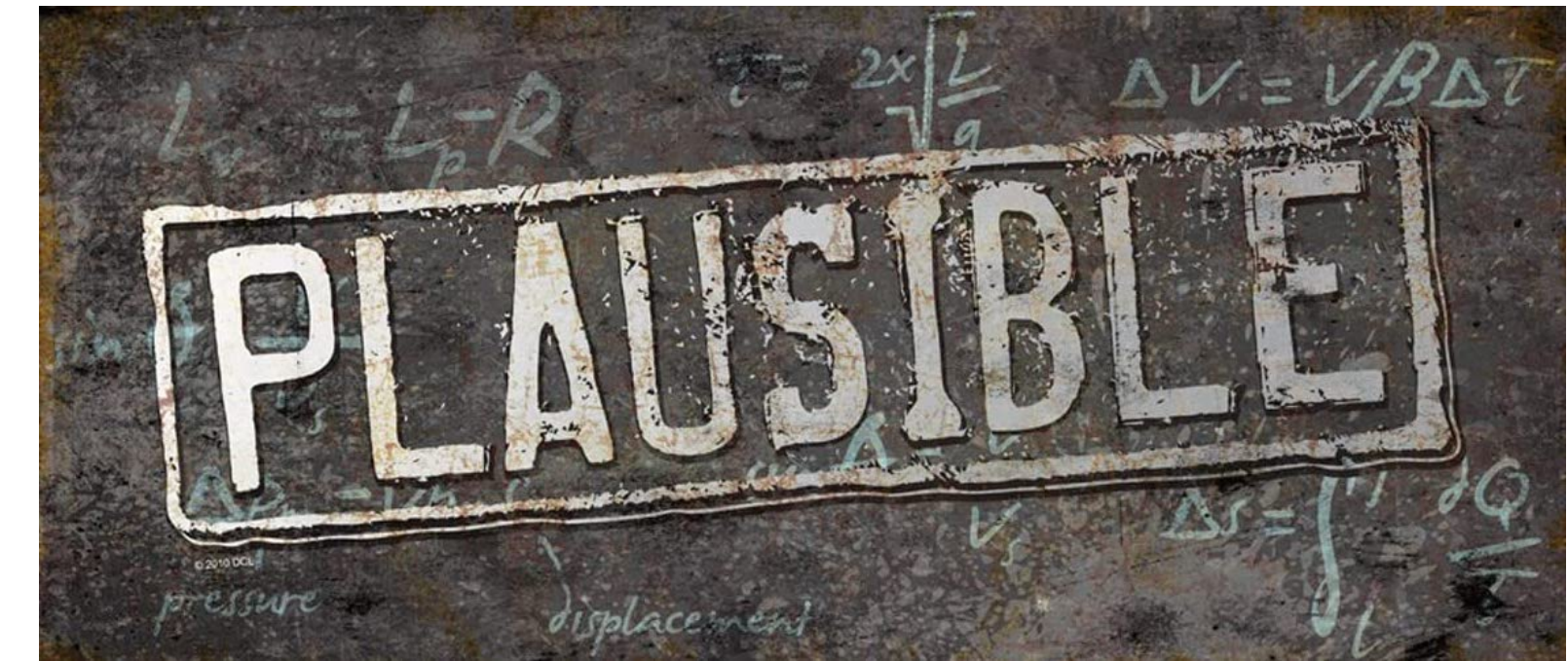
# Test Myth

It's 2023, we should be able to leverage the power of C++20 **modules** to (re)structure our codebase and improve build times.



# Test Myth

coroutines shipped in C++20

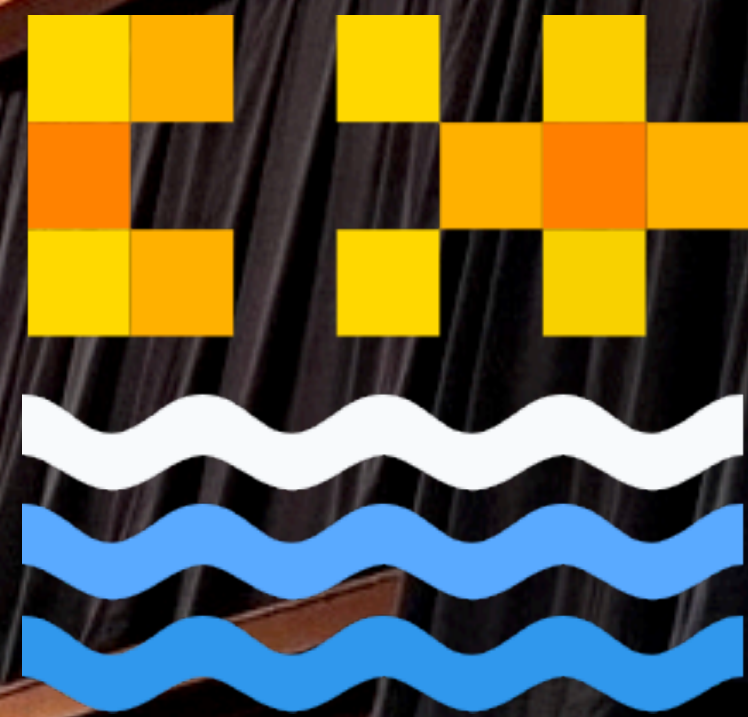


Kinda... 🙄

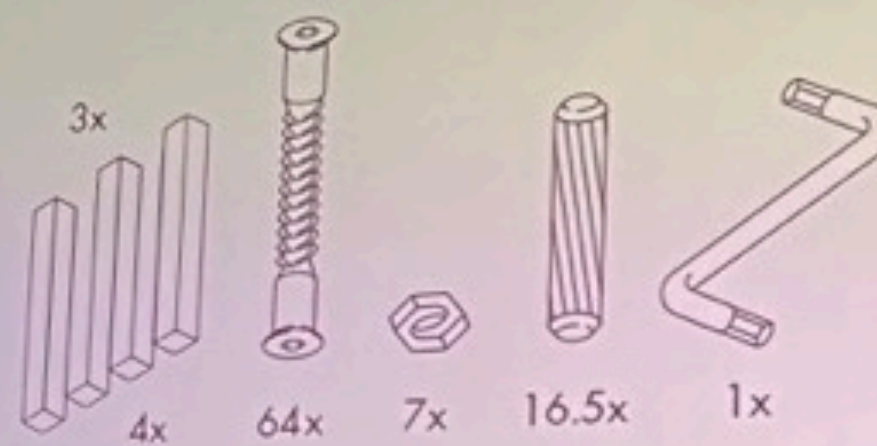
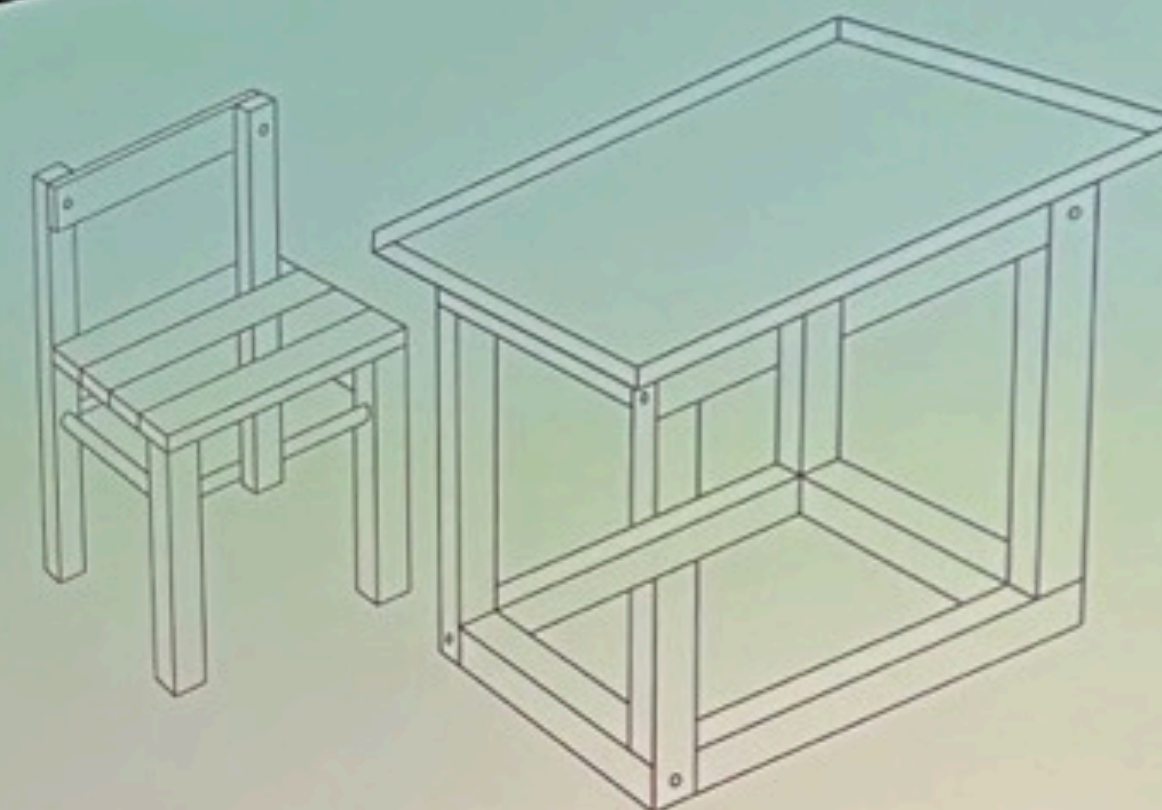
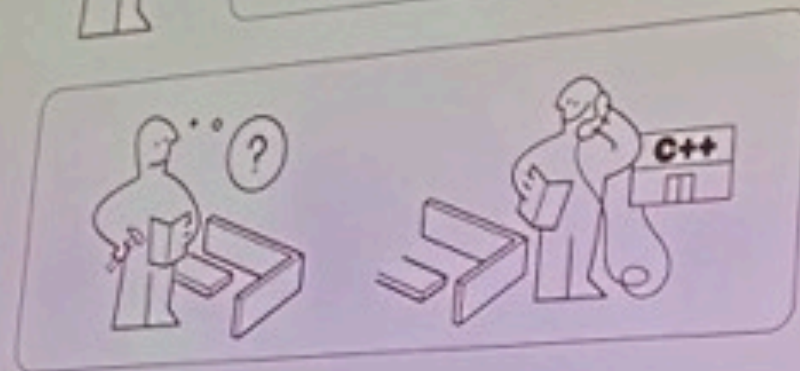
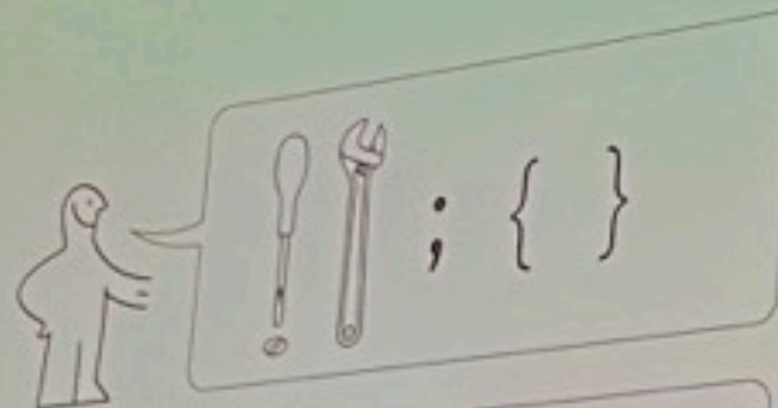
We're going to get a `generators` library in C++23 (ranges library)

```
#include <generator>
```





## CÖRUTIN



I think you got how it works




<Part 2 of N> presentation



# Season 1 Recap

## What we covered so far (Part 1):

- #11 printf/sprintf are very fast
- #14 C++ is not easily toolable 
- #19 std::regex is too slow for production use
- #24a std::optional inhibits optimizations
- #24b std::optional complicates APIs (boxes, lifting, continuation monads)
- #31 std::move() moves
- #36 Always pass input arguments by const reference (move, sinks)
- #5 Adding `const` always helps (places where not to use `const`)
- #37 Make All Data Members Private? (abstraction, structs, perf, DOD)
- #40 Iterators must go!
- #0 New (C++) is the enemy of the old

Let's dig in!

# Humans Depend on Tools



C++ is not easily toolable 

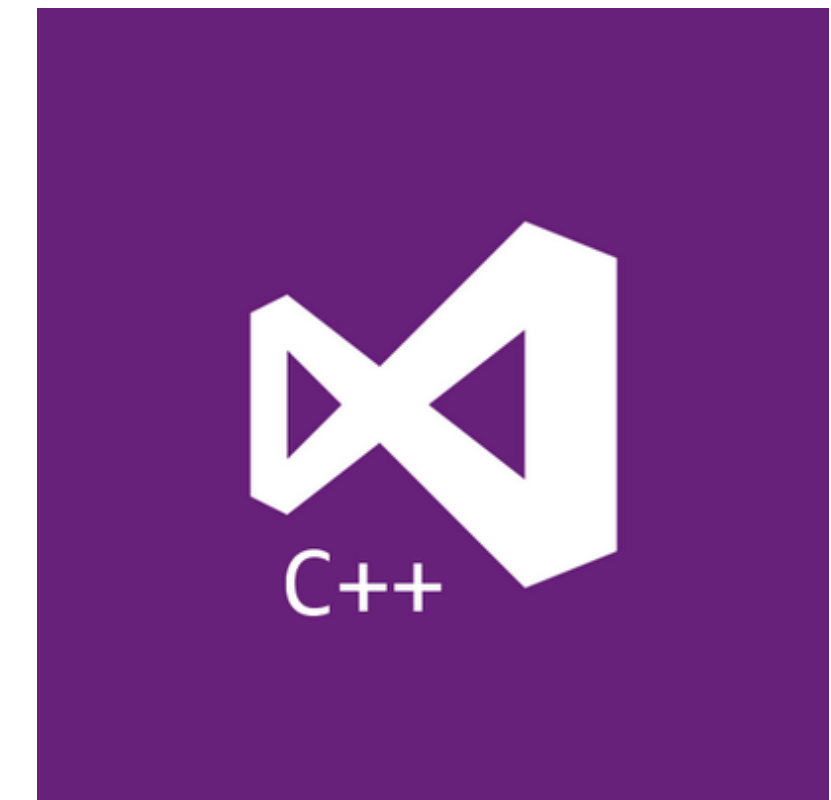
# I'm a tool builder



[Advanced Installer](#)



[Clang Power Tools](#)



Visual C++

# Programmers Depend on Tools

code editor/IDE

IntelliSense

recent compiler(s)  
[conformant/strict]

linter/formatter

perf profiler

(visual) debugger

test framework

(automated) refactoring tools

build system

static analyzer

package manager

CI/CD service

dynamic analyzer  
(runtime)

SCM client

code reviews platform

+ fuzzing

# Programmers Depend on Tools



lefticus commented 26 days ago

Owner Author 😊 ...

## We are in a golden age of C++ tools

If you are developing blindly, without any tool guidance, you are doing C++ wrong. Think of these tools like a backup camera in your car. Certainly you can back up without a camera, but having one gives you a second set of eyes, deeper into the action than is possible with your human eyes.

You need:

- Continuous build environment
  - github
  - gitlab
  - jenkins
  - <what's your favorite, did I leave it out?>
- As many compilers as you can
  - GCC
  - Clang
  - cl (visual studio)
  - clang-cl (clang's msvc compatibility)
- An organized testing framework
  - doctest
  - catch
  - gtest
  - boosttest
  - <what's your favorite, did I leave it out?>
- test coverage analysis, reporting and tracking (you need to know if your test rate is decreasing!)
  - coveralls
  - codecov
  - <what else am I missing here?>
- As much static analysis as you can (most are free or have free options)
  - *at least* `-Wall -Wextra -Wshadow -Wconversion -Wpedantic -Werror` and `-W4` on Windows
  - gcc `-fanalyzer` - <https://gcc.gnu.org/onlinedocs/gcc/Static-Analyzer-Options.html>
  - `cl.exe /analyze`
  - cppcheck
  - clang-tidy
  - pvs studio <https://pvs-studio.com/en/>
  - sonar's tools
  - <countless many options, I expect many of you to tell me that I'm missing don't work with C++>
- Runtime analysis during testing
  - address sanitizer (<https://clang.llvm.org/docs/index.html>)
  - undefined behavior sanitizer
  - thread sanitizer
  - valgrind (if you can tolerate it)
  - debug checked iterators  
[https://gcc.gnu.org/onlinedocs/libstdc++/manual/debug\\_mode\\_usin](https://gcc.gnu.org/onlinedocs/libstdc++/manual/debug_mode_usin)  
<https://learn.microsoft.com/en-us/cpp/standard-library/checked-ite>
  - drmemory

## C++ Weekly - The Right Way to Write C++ Code

[youtube.com/watch?v=q7Gv4J3FyYE](https://youtube.com/watch?v=q7Gv4J3FyYE)

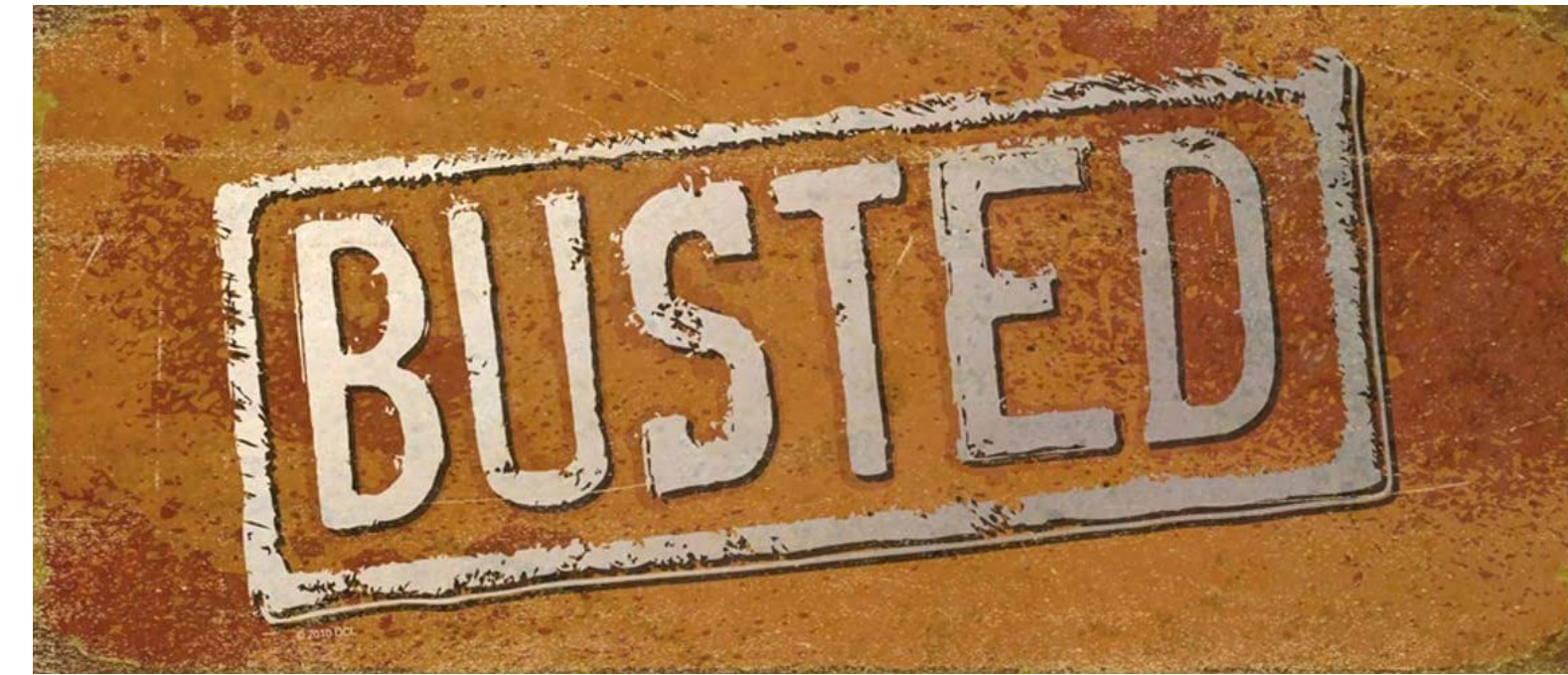
[github.com/lefticus/cpp\\_weekly/issues/175](https://github.com/lefticus/cpp_weekly/issues/175)

- Fuzz Testing
  - More on this coming, but every library should be fuzz tested
  - It generates novel / unique inputs for your library in an attempt to generate 100% code coverage
  - Should be used in conjunction with runtime analysis, to hard-catch any bug
- Ship with hardening enabled
  - Control Flow Guard - <https://learn.microsoft.com/en-us/cpp/build/reference/guard-enable-control-flow-guard?view=msvc-170>
  - `_FORITFY_SOURCE` - <https://developers.redhat.com/articles/2022/09/17/gccs-new-fortification-level>
  - Stack Protector - <https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html>
  - UBSan "Minimal runtime" mode - <https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html#minimal-runtime>

See more info about tools and specific compiler options and flags here: [https://github.com/cpp-best-practices/cppbestpractices/blob/master/02-Use\\_the\\_Tools\\_Available.md](https://github.com/cpp-best-practices/cppbestpractices/blob/master/02-Use_the_Tools_Available.md)

Using an IDE or plugin for your IDE can help integrate many of these things as well.

C++ is not easily toolable 🛠️



**Get to know your tools well**



C++ is slow to compile




It's all about the structure & build configuration you have.

So, *you think you know* why your builds take so long... you'd be surprised.

# Myth #10

Multiple ways to improve (or screw up) your build:

- build configuration
- project dependencies (graph)
- header usage (compilation firewalls)
- unity builds
- PCH
- C++ modules/header units
- build caches
- build accelerators
- vfs
- ... use ranges 

# Myth #10

Header / Source <input type="text" value="filter column..."/>	Version ▲	Impact ⓘ ▲	Timing ⓘ ▼	Lines ⓘ ▲	Binary ⓘ ▲
▼ C++ Standard Library (79 files, <a href="https://en.cppreference.com/w/cpp/header">https://en.cppreference.com/w/cpp/header</a> )					
<input type="checkbox"/> <regex>			238 .. 365 ms	38.9 .. 43.7 kLoC	0 .. 188 kB
<input type="checkbox"/> <filesystem>			263 .. 341 ms	30.4 .. 31.1 kLoC	0 .. 363 kB
<input type="checkbox"/> <future>			179 .. 292 ms	20.5 .. 23.5 kLoC	0 .. 278 kB
<input type="checkbox"/> <random>			130 .. 239 ms	23.0 .. 28.3 kLoC	0 .. 143 kB
<input type="checkbox"/> <complex>			125 .. 236 ms	19.1 .. 25.1 kLoC	0 .. 140 kB
<input type="checkbox"/> <functional>			82 .. 228 ms	12.9 .. 27.4 kLoC	0 .. 141 kB
<input type="checkbox"/> <iomanip>			115 .. 221 ms	18.8 .. 24.7 kLoC	0 .. 180 kB
<input type="checkbox"/> <locale>			113 .. 196 ms	18.6 .. 22.1 kLoC	0 .. 178 kB
<input type="checkbox"/> <shared_mutex>			125 .. 195 ms	17.5 .. 19.6 kLoC	0 .. 153 kB
<input type="checkbox"/> <condition_variable>			112 .. 192 ms	16.5 .. 19.4 kLoC	0 .. 153 kB
<input type="checkbox"/> <fstream>			115 .. 192 ms	17.3 .. 20.6 kLoC	0 .. 138 kB
<input type="checkbox"/> <thread>			110 .. 189 ms	17.5 .. 20.3 kLoC	0 .. 153 kB
<input type="checkbox"/> <unordered_map>			96 .. 188 ms	15.3 .. 20.4 kLoC	0 .. 137 kB
<input type="checkbox"/> <unordered_set>			94 .. 186 ms	15.3 .. 20.3 kLoC	0 .. 137 kB
<input type="checkbox"/> <sstream>			104 .. 180 ms	16.3 .. 19.6 kLoC	0 .. 138 kB
<input type="checkbox"/> <iostream>			101 .. 176 ms	15.8 .. 19.1 kLoC	0.9 .. 142 kB
<input type="checkbox"/> <iterator>			100 .. 176 ms	15.9 .. 19.2 kLoC	0 .. 138 kB
<input type="checkbox"/> <istream>			100 .. 175 ms	15.8 .. 19.1 kLoC	0 .. 138 kB
<input type="checkbox"/> <mutex>			92 .. 170 ms	14.4 .. 17.2 kLoC	0 .. 153 kB

[artificial-mind.net/projects/compile-health/](https://artificial-mind.net/projects/compile-health/)

# Myth #10

- ▶ **Standard Library** (3 libraries)
- ▶ **boost** (24 libraries)
- ▼ **format** (1 library)
  - ▶ **fmt** (11 files, <https://github.com/fmtlib/fmt>)
- ▶ **geometry** (1 library)
- ▶ **image** (2 libraries)
- ▼ **json** (6 libraries)
  - ▶ **Boost.JSON** (20 files, <https://github.com/CPPAlliance/json>)
  - ▶ **cJSON** (2 files, <https://github.com/DaveGamble/cJSON>)
  - ▶ **jsonxx** (2 files, <https://github.com/hjiang/jsonxx>)
  - ▶ **nlohmann-json** (2 files, <https://github.com/nlohmann/json>)
  - ▶ **picojson** (1 file, <https://github.com/kazuho/picojson>)
  - ▶ **rapidjson** (16 files, <https://github.com/Tencent/rapidjson>)
- ▶ **math** (2 libraries)
- ▼ **testing** (3 libraries)
  - ▶ **Catch2** (1 file, <https://github.com/catchorg/Catch2>)
  - ▶ **doctest** (3 files, <https://github.com/onqtam/doctest>)
  - ▶ **googletest** (1 file, <https://github.com/google/googletest>)

[artificial-mind.net/projects/compile-health/](https://artificial-mind.net/projects/compile-health/)

# Myth #10



## Tooling can help: [ClangBuildAnalyzer](#) `-ftime-trace`

- Free & open-source tool developed by [Aras Pranckevičius](#)
  - Parses Clang's `-ftime-trace` output and produces a human-friendly report
  - The report provides *actionable* information
- `-ftime-trace`
  - Developed by Aras himself, merged upstream since Clang 9 <sup>[src]</sup>
  - Produces Chrome Tracing `.json` files for each compiled object file
  - No equivalent in GCC or MSVC
- How to use
  - Use `clang++` as your compiler, passing `-ftime-trace` to your compiler flags
  - Compile everything you want to profile
  - Run `ClangBuildAnalyzer` in the build directory

```
cmake -GNinja -DCMAKE_UNITY_BUILD=ON -DCMAKE_CXX_COMPILER=clang++  
      -DCMAKE_CXX_FLAGS="-fuse-ld=lld -ftime-trace"
```

```
./ClangBuildAnalyzer.exe --all . analysis.bin  
./ClangBuildAnalyzer.exe --analyze analysis.bin > analysis.txt && explorer analysis.txt
```

# Myth #10



Tooling can help: **vcperf + WPA**

[devblogs.microsoft.com/cppblog/introducing-c-build-insights/](https://devblogs.microsoft.com/cppblog/introducing-c-build-insights/)

- `vcperf /start MySession`
- build your C++ project
- `vcperf /stop MySession outputFile.etl`

Line #	Activity Name	Included Path	Inclusive Duration (s)	Count
1	▼ Parsing		1,600.686000000	126,459
2	▶ E:\git\src\git-compat-util.h		257.819000000	452
3	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\um\winsock2.h		181.722000000	452
4	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\um\windows.h		172.351000000	452
5	▶ E:\git\src\cache.h		138.028000000	392
6	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\um\winbase.h		60.867000000	452
7	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\shared\windef.h		52.328000000	452
8	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\shared\minwin...		51.181000000	452
9	▶ E:\git\src\t\helper\test-tool.h		48.151000000	58
10	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\um\winnt.h		44.967000000	452
11	▶ E:\git\src\compat\msvc.h		44.715000000	452
12	▶ E:\git\src\builtin.h		39.804000000	119
13	▶ E:\git\src\compat\mingw.h		31.620000000	452
14	▶ E:\git\src\compat\vcbuild\vcpkg\installed\x64-windows\include\openssl\ssl.h		24.052000000	1,356
15	▶ E:\git\src\http.c		18.990000000	4
16	▶ E:\git\src\common-main.c		15.473000000	16
17	▶ E:\git\src\compat\vcbuild\vcpkg\installed\x64-windows\include\openssl\x5...		13.268000000	452
18	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\um\winuser.h		12.448000000	452
19	▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.17763.0\ucrt\stdio.h		9.508000000	453

Start: 0.002016400s  
End: 104.538962800s  
Duration: 104.536946400s

# Myth #10



Tooling can help: [Build Insights in Visual Studio](#)

Included Files | Include Tree

Diagnostics Session: 75.462 seconds Build: 72.59 seconds

Filter Files

File Path	Time [sec, %]	Parse Count	Project
▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22000.0\um\windows.h	10.002 (13.8%)	45	Irrlicht15.0
▶ C:\src\irrlicht\include\irrAllocator.h	7.174 (9.9%)	217	Irrlicht15.0
▶ C:\Program Files\Microsoft Visual Studio\2022\Main\VC\Tools\MSVC\14.37.326...	6.862 (9.5%)	217	Irrlicht15.0
▶ C:\Program Files\Microsoft Visual Studio\2022\Main\VC\Tools\MSVC\14.37.326...	6.495 (8.9%)	217	Irrlicht15.0
▶ C:\src\irrlicht\include\irrString.h	5.069 (7.0%)	206	Irrlicht15.0
▶ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22000.0\ucrt\stdio.h	4.649 (6.4%)	296	Irrlicht15.0
▶ C:\src\irrlicht\include\ISceneNode.h	4.567 (6.3%)	80	Irrlicht15.0
▶ C:\Program Files\Microsoft Visual Studio\2022\Main\VC\Tools\MSVC\14.37.326...	4.532 (6.2%)	217	Irrlicht15.0
▶ C:\src\irrlicht\include\IrrCompileConfig.h	4.286 (5.9%)	227	Irrlicht15.0
▶ C:\src\irrlicht\include\irrTypes.h	4.011 (5.5%)	222	Irrlicht15.0

[devblogs.microsoft.com/cppblog/build-insights-now-available-in-visual-studio-2022/](https://devblogs.microsoft.com/cppblog/build-insights-now-available-in-visual-studio-2022/)

# Myth #10



Tooling can help: [Build Insights in Visual Studio](#)

Trace230609110806.et What's New?

Included Files Include Tree

Diagnostics Session: 76.549 seconds Build: 73.506 seconds Filter Files

File Path	Time [sec, %]	Include Count	Project
▲ C:\src\irrlicht_pch\source\Irrlicht\Irrlicht.cpp	0.821 (1.1%)	6	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\um\...	0.431 (0.6%)	34	Irrlicht15.
▷ C:\src\irrlicht_pch\include\irrlicht.h	0.308 (0.4%)	97	Irrlicht15.
▲ C:\src\irrlicht_pch\include\IrrCompileConfig.h	0.042 (0.1%)	1	Irrlicht15.
▲ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\uc...	0.042 (0.1%)	2	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\...	0.019 (0.0%)	1	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\...	0.005 (0.0%)	1	Irrlicht15.
▷ C:\src\irrlicht_pch\source\Irrlicht\CIrrDeviceWin32.h	0.012 (0.0%)	3	Irrlicht15.
C:\src\irrlicht_pch\source\Irrlicht\CIrrDeviceConsole.h	0.004 (0.0%)	0	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\ucrt\...	0.003 (0.0%)	1	Irrlicht15.
▲ C:\src\irrlicht_pch\source\Irrlicht\CSoftwareDriver2.cpp	0.662 (0.9%)	5	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\um\...	0.382 (0.5%)	34	Irrlicht15.
▷ C:\src\irrlicht_pch\source\Irrlicht\CSoftwareDriver2.h	0.203 (0.3%)	4	Irrlicht15.
▷ C:\src\irrlicht_pch\include\IrrCompileConfig.h	0.032 (0.0%)	1	Irrlicht15.

[devblogs.microsoft.com/cppblog/build-insights-now-available-in-visual-studio-2022/](https://devblogs.microsoft.com/cppblog/build-insights-now-available-in-visual-studio-2022/)



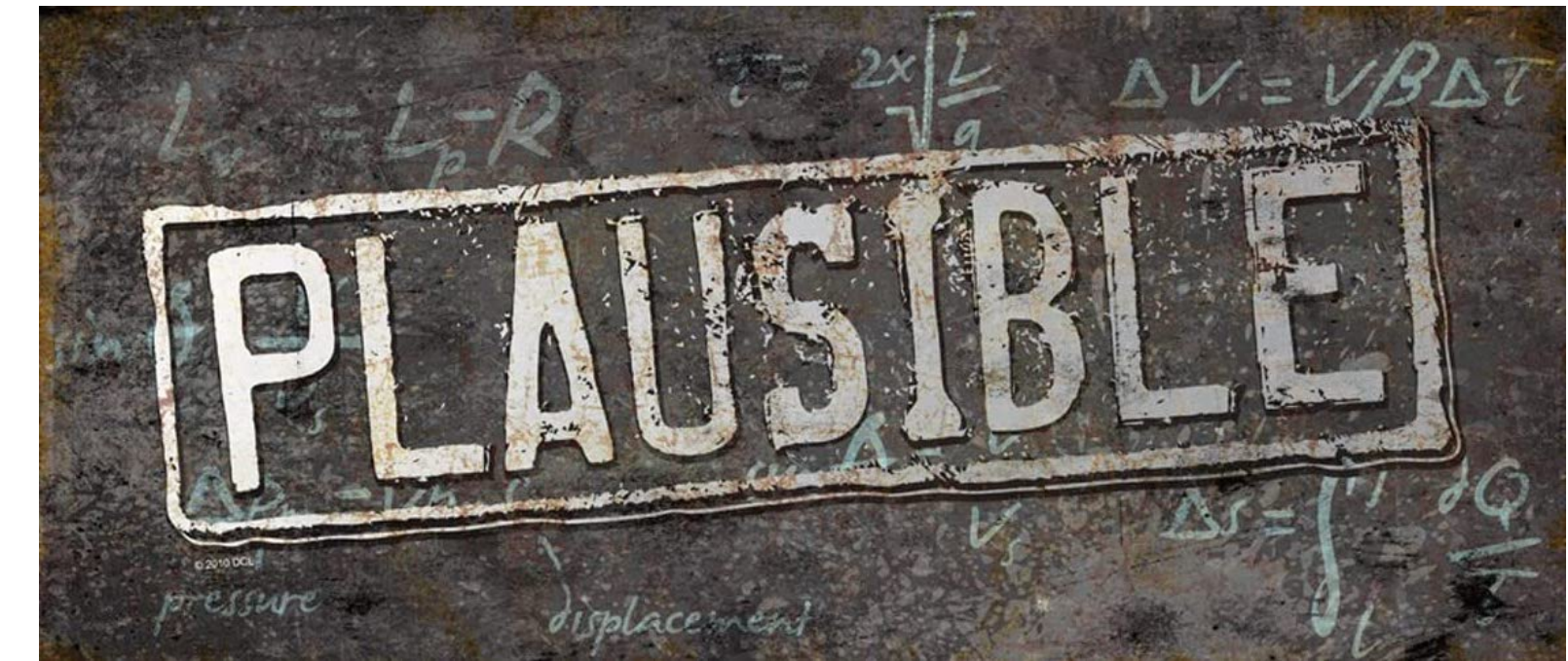
# Myth #10

The image shows a YouTube video player interface. At the top left, the text "IMPROVING COMPILATION TIMES" is written in a stylized purple font, with "TOOLS & TECHNIQUES" underneath. In the top right corner, the "ACCU 2023" logo is displayed in purple. The main video area has a light grey background with the title "IMPROVING COMPILATION TIMES" in large black letters and the subtitle "Tools & Techniques" in a smaller, italicized font. To the right of the title, it says "ACCU 2023" and "April 20 2023". On the right side of the video frame, there is a vertical inset showing a man with glasses and a beard, identified as Vittorio Romeo, standing at a podium with a laptop. Below the video frame, there is a dark blue control bar with various icons for play, volume, and settings. The video progress is shown as "0:06 / 1:43:50". In the bottom right corner of the video frame, there is a "Bloomberg Engineering" logo and the text "TechAtBloomberg.com Career 2". The "ACCU.ORG" logo is visible in the bottom center of the video frame.

[youtube.com/watch?v=PfHD3BsVsAM](https://youtube.com/watch?v=PfHD3BsVsAM)

# Myth #10

C++ is slow to compile



It can be, but if you work on it ([good tooling](#)) you can drastically improve it.

# Myth #12

The sad state of **Debug** performance in C++

“*zero cost abstraction*” is a kind of a lie - for sure on Debug builds (no optimizations)

eg.

```
int i = 0;  
std::move(i);  
std::forward<int&>(i);
```

➔ `static_cast<int&&>(i);`

[vittorioromeo.info/index/blog/debug\\_performance\\_cpp.html](http://vittorioromeo.info/index/blog/debug_performance_cpp.html)

# Myth #12

C++ source #1

```
1 #include <utility>
2
3 int main()
4 {
5     int i = 0;
6     return std::move(i);
7 }
```

x86-64 gcc 13.1 (Editor #1)

```
1 main:
2     push    rbp
3     mov     rbp, rsp
4     sub    rsp, 16
5     mov     DWORD PTR [rbp-4], 0
6     lea    rax, [rbp-4]
7     mov     rdi, rax
8     call   std::remove_reference<int&>::type&& std::move<int&>(int&)
9     mov     eax, DWORD PTR [rax]
10    leave
11    ret
```

x64 msvc v19.34 (Editor #1)

```
1 i$ = 32
2 main PROC
3 $LN3:
4     sub    rsp, 56                ; 00000038H
5     mov     DWORD PTR i$[rsp], 0
6     lea    rcx, QWORD PTR i$[rsp]
7     call   int && std::move<int &>(int &) ; std::move
8     mov     eax, DWORD PTR [rax]
9     add    rsp, 56                ; 00000038H
10    ret     0
11 main ENDP
```

x86-64 clang 14.0.0 (Editor #1)

```
1 main:                                # @main
2     push    rbp
3     mov     rbp, rsp
4     sub    rsp, 16
5     mov     dword ptr [rbp - 4], 0
6     mov     dword ptr [rbp - 8], 0
7     lea    rdi, [rbp - 8]
8     call   std::__1::remove_reference<int&>::type&& std::__1::move<i
9     mov     eax, dword ptr [rax]
10    add    rsp, 16
11    pop    rbp
12    ret
```

[godbolt.org/z/Pj6xahP9j](https://godbolt.org/z/Pj6xahP9j)

# Myth #12

```
C++ source #1 X
A [Icons] C++
1 #include <utility>
2
3 int main()
4 {
5     int i = 0;
6     return std::move(i);
7 }
(6, 21)
```

```
x64 msvc v19.35 (Editor #1) X
x64 msvc v19.35 /std:c++latest
A [Icons]
1 i$ = 0
2 main PROC
3 $LN3:
4     sub     rsp, 24
5     mov     DWORD PTR i$[rsp], 0
6     mov     eax, DWORD PTR i$[rsp]
7     add     rsp, 24
8     ret     0
9 main ENDP
```

```
x86-64 gcc (trunk) (Editor #1) X
x86-64 gcc (trunk) -std=c++20 -Wall -Wextra -Wpedantic
A [Icons] Output... Filter... Libraries Overrides + Add new... Add tool...
1 main:
2     push   rbp
3     mov    rbp, rsp
4     sub   rsp, 16
5     mov   DWORD PTR [rbp-4], 0
6     lea  rax, [rbp-4]
7     mov  rdi, rax
8     call std::remove_reference<int&>::type&& std::move<int&>(int&)
9     mov  eax, DWORD PTR [rax]
10    leave
11    ret
Output (0/0) x86-64 gcc (trunk) i - cached (11069B) ~706 lines filtered Compiler License
```

```
x86-64 clang 16.0.0 (Editor #1) X
x86-64 clang 16.0.0 -std=c++20 -stdlib=libc++ -Wall -Wextra -Wpedan
A [Icons] Output... Filter... Libraries Overrides + Add new... Add tool...
1 main: # @main
2     push   rbp
3     mov    rbp, rsp
4     mov   dword ptr [rbp - 4], 0
5     mov   dword ptr [rbp - 8], 0
6     mov   eax, dword ptr [rbp - 8]
7     pop   rbp
8     ret
```

[godbolt.org/z/5vEhrnPbK](https://godbolt.org/z/5vEhrnPbK)

# Myth #12

Compilers can implement some mechanism to acknowledge meta functions like `std::move` and `std::forward` as compiler intrinsics - in the *compiler front-end*

MSVC took an alternative approach and implemented this new inlining ability using a C++ attribute: `[[msvc::intrinsic]]`

The new attribute will semantically replace a function `call` with a `cast` to that function's return type if the function definition is decorated with `[[msvc::intrinsic]]`

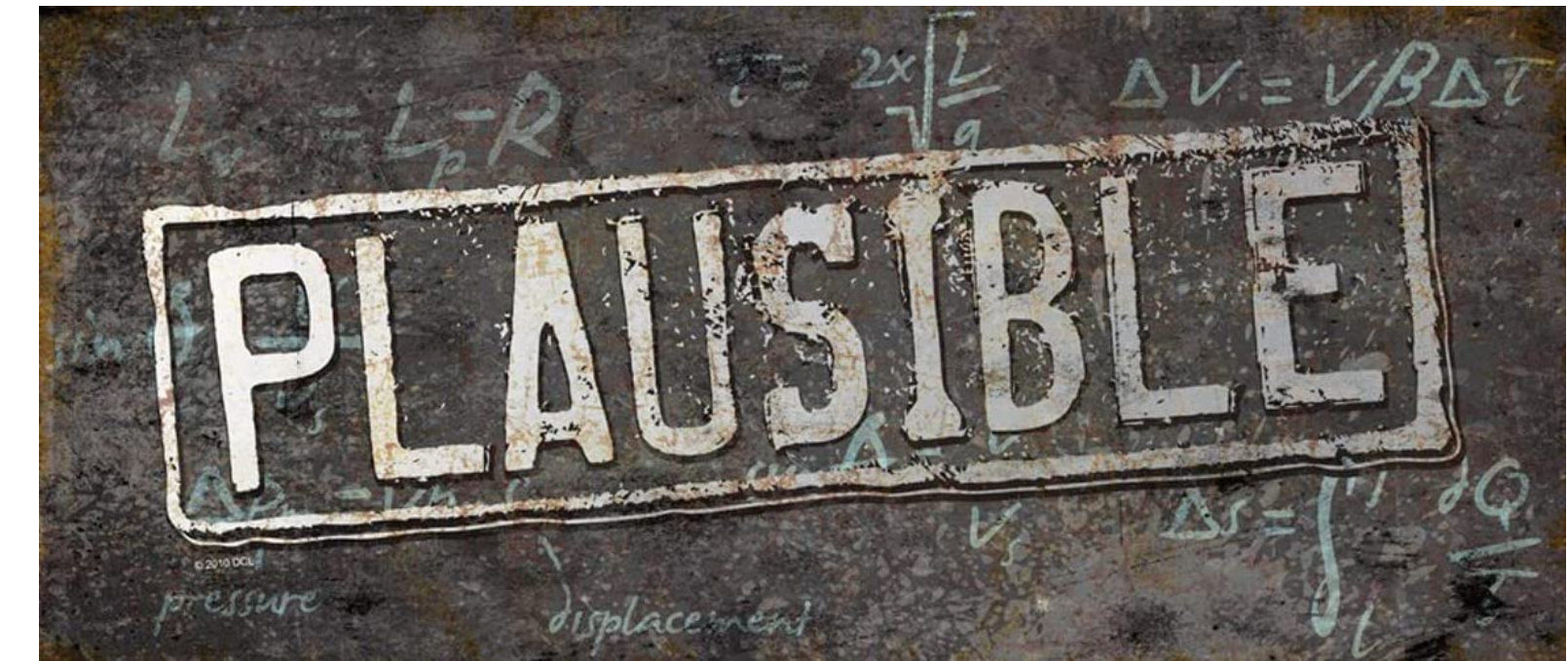
=> *extensible* to your own such utility functions

[youtu.be/idwVQUG6Jqc](https://youtu.be/idwVQUG6Jqc)

[devblogs.microsoft.com/cppblog/improving-the-state-of-debug-performance-in-c/](https://devblogs.microsoft.com/cppblog/improving-the-state-of-debug-performance-in-c/)

# Myth #12

The sad state of **Debug** performance in C++



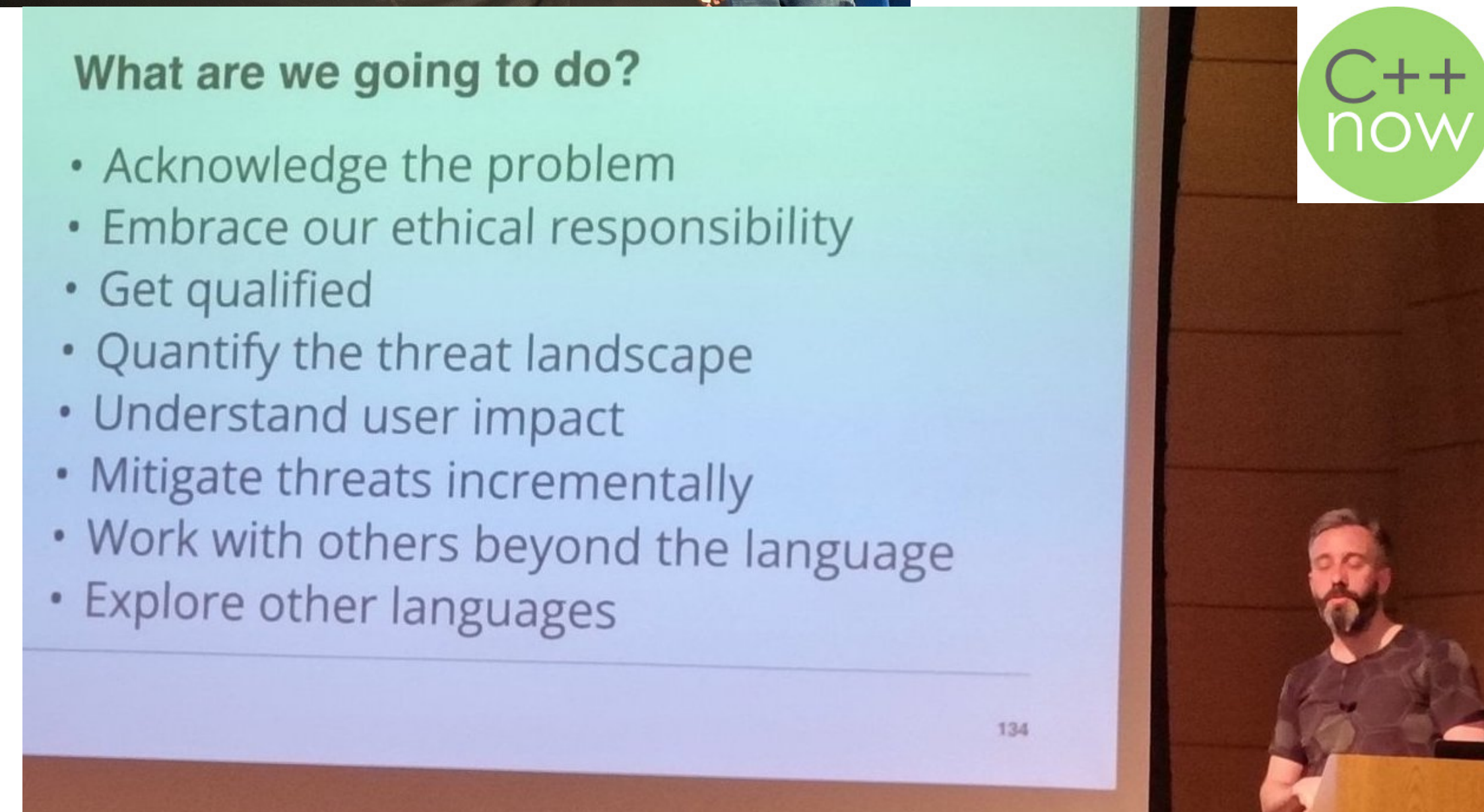
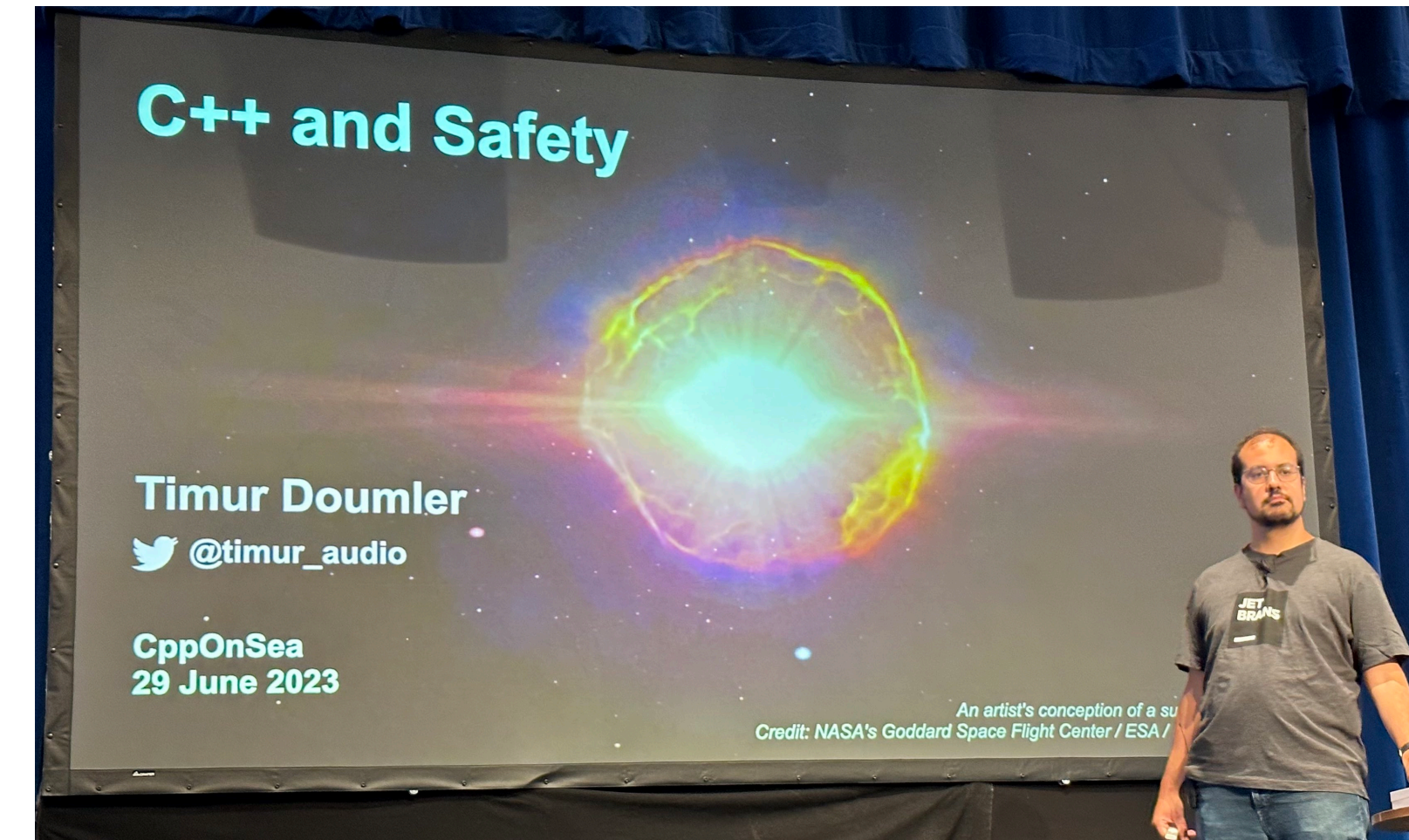
C++ will never be a **safe** language

- type safety
- bounds safety
- lifetime safety
- initialization safety
- object access safety
- thread safety
- arithmetic safety



# Myth #23

C++ is under attack... and the community is responding 🙋



Tradeoffs need to be made...



"To UB, or not to UB"

-- *Prince Hamlet*

We have not addressed C++ safety until we have eliminated **all** UB.

We can't **completely** eliminate UB from C++ (for good reasons\*).



C++ will never be a **safe** language

# Myth #23



# Myth #23

An excellent essay on the subject of safety: "*If we must, let's talk about safety*"

[cor3ntin.github.io/posts/safety/](https://cor3ntin.github.io/posts/safety/)

-- Corentin Jabot

- A cakewalk and eating it too
- Borrowing the borrow checker
- But we care about safety, right?
- Dogma
- Down with Safety!
- UB
- Correct by confusion
- ++(C++) / Rust



# Myth #23

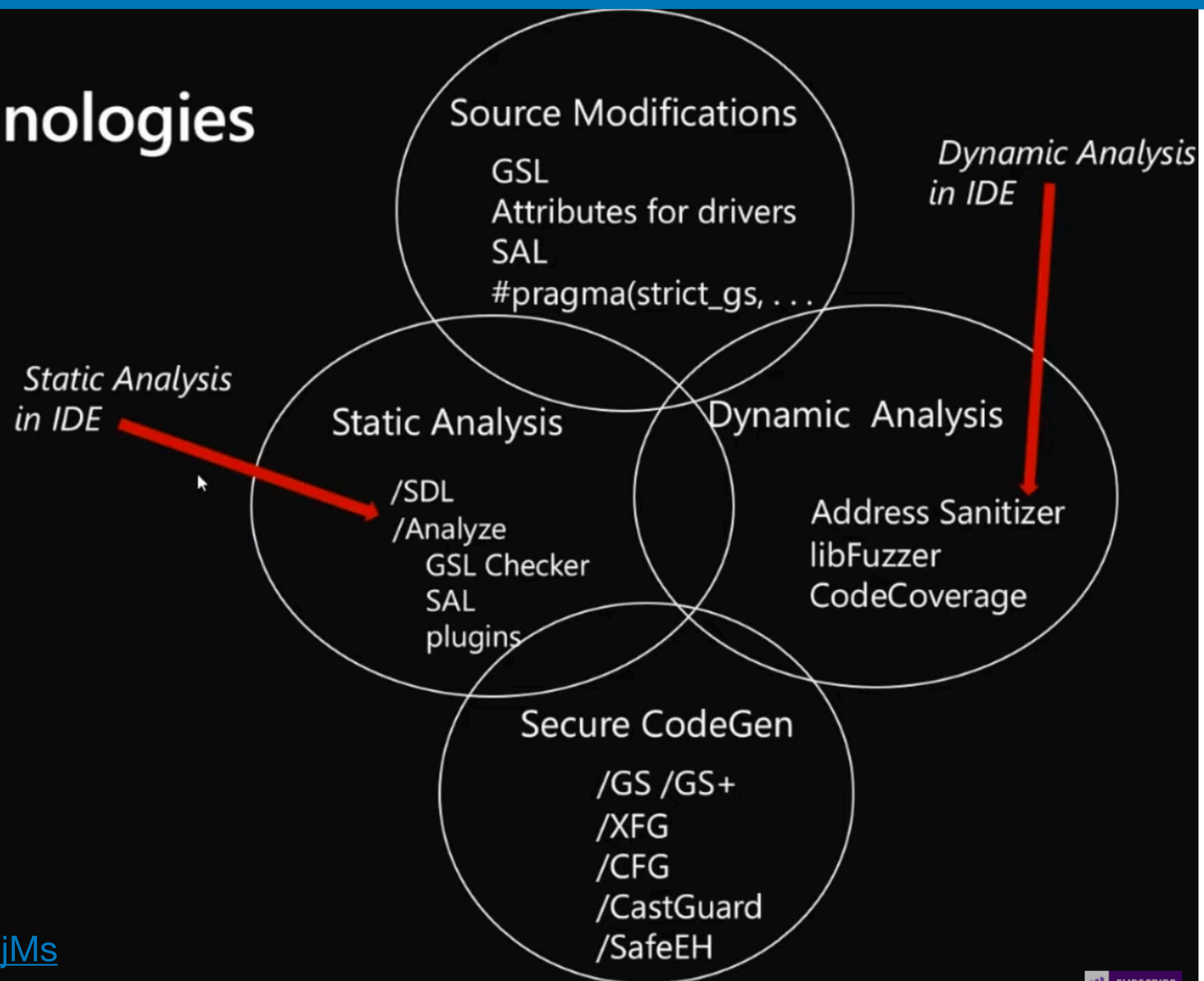
Guarantee **lifetime** safety:

- garbage collector 🤖
- *dynamic* memory analysis (ASan)
- *statically* enforce rules on references: **multiple immutable refs** | | **unique mutable ref**
  - **by compiler/language:**
    - borrow checker (Rust)
    - mutable value semantics (Val)
    - no direct mutation (Haskell & other pure functional languages)
  - **by tooling (static lifetime analysis):**
    - clang-tidy
    - MSVC
    - other commercial analyzers

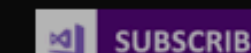
~~AAA (almost always auto)~~

AAA (almost always analyze)

## C++ Security Technologies



[youtube.com/watch?v=i8\\_RfDAEjMs](https://youtube.com/watch?v=i8_RfDAEjMs)



# Myth #23

ASan FTW !!!

-fsanitizer=address

{ Clang, gcc, MSVC }

[youtube.com/watch?v=yJLyANPHNaA](https://youtube.com/watch?v=yJLyANPHNaA)

The image is a promotional banner for the Cppcon 2020 conference. On the left, there is a video thumbnail of Victor Ciura, a man with glasses wearing a maroon polo shirt with a Cppcon logo. Above the thumbnail is the name 'Victor Ciura' and below it is the title '2020: The Year of Sanitizers?'. The main banner features the Cppcon logo (a stylized plus sign in a circle) and the text 'Cppcon The C++ Conference'. To the right, it says '2020 September 13-18 ONLINE GOING VIRTUAL'. At the bottom right, it identifies 'Victor Ciura Principal Engineer' and includes a Twitter handle '@ciura\_victor' and the CAPHYON logo.



## ASan `continue_on_error`


[devblogs.microsoft.com/cppblog/addresssanitizer-continue\\_on\\_error/](https://devblogs.microsoft.com/cppblog/addresssanitizer-continue_on_error/)

**NEW:** (Visual Studio 2022 v17.6)

Address Sanitizer runtime which provides a new “checked build”.

This new runtime mode diagnoses and reports hidden memory safety errors, with zero false positives, as your app runs.

[youtube.com/watch?v=i8\\_RfDAEjMs](https://youtube.com/watch?v=i8_RfDAEjMs)



The image is a screenshot of a presentation slide. At the top left, there is a logo consisting of three curved lines in yellow and white. To the right of the logo, the text 'Pure Virtual' is written in white, and 'C++ 2023' is written in yellow. In the top right corner, the Microsoft logo is visible. The main content of the slide is 'Address Sanitizer' in white, followed by 'continue\_on\_error' in white with a small white triangle pointing to the right. Below this, there is a circular portrait of a man with white hair and glasses, identified as 'Jim Radigan'. Underneath his name, it says 'Partner Software Architect, Microsoft'. The background of the slide is dark with some faint yellow and white geometric shapes.

## Static Analysis lifetime annotations for C++

**NEW:**

`[[clang::lifetimebound]]` and `[[msvc::lifetimebound]]`

[discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377](https://discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377)

[youtube.com/watch?v=fe6yu9AQIE4](https://youtube.com/watch?v=fe6yu9AQIE4)



The image is a video thumbnail with a dark background. In the top left, there is a logo consisting of three curved lines in yellow and white. To its right, the text 'Pure Virtual' is in white, and 'C++ 2023' is in yellow. In the top right corner, the Microsoft logo and the word 'Microsoft' are visible. The main title 'Lifetime Analysis Improvements' is in white, with a play button icon to its right. Below the title, there is a circular portrait of Gabor Horvath, a man with short dark hair, wearing a black shirt. Below the portrait, his name 'Gabor Horvath' is written in white, followed by his title 'Software Engineer, Microsoft' in a smaller white font. There are also some faint yellow symbols like '<>' and '+' scattered on the background.

# Myth #23

C++ will never be a **safe** language\*



\* but it can be much **safe(r)** with some effort and **good tooling** 

# Myth #38

Just rewrite it in Rust 🦀



**Mark Russinovich**

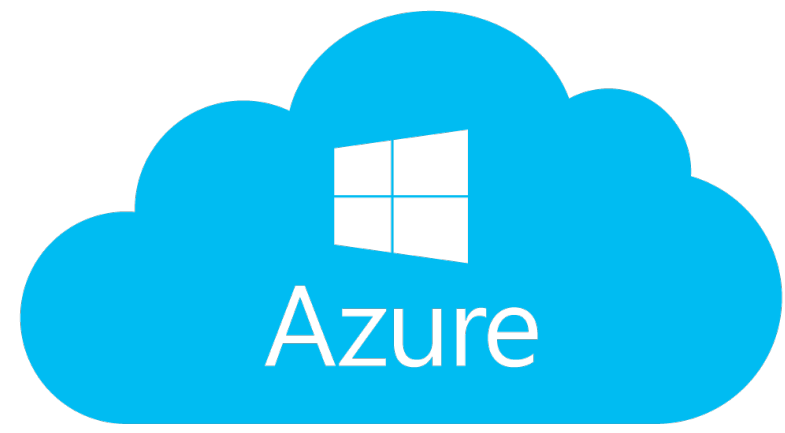
@markrussinovich · [Follow](#)



Speaking of languages, it's time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability, the industry should declare those languages as deprecated.

11:50 PM · Sep 19, 2022

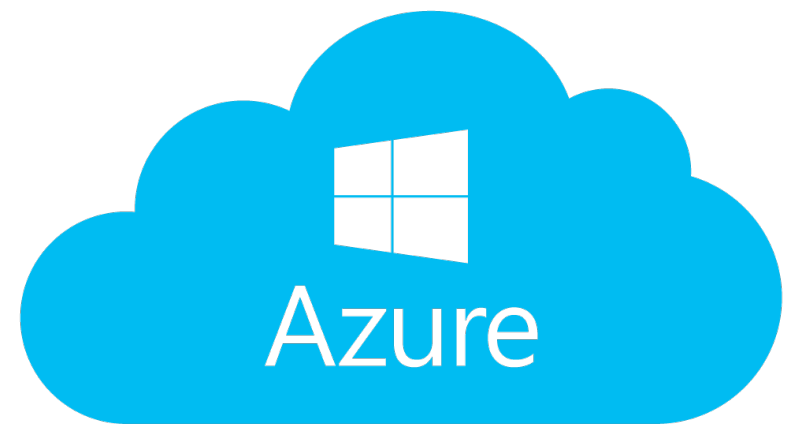




## Microsoft Azure security evolution: Embrace secure multitenancy, Confidential Compute, and Rust

By [Jeffrey Cooperstein](#) Partner Software Architect, Azure Security

[azure.microsoft.com/blog/microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/](https://azure.microsoft.com/blog/microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/)



## Microsoft Azure security evolution: Embrace secure multitenancy, Confidential Compute, and Rust

By [Jeffrey Cooperstein](#) Partner Software Architect, Azure Security



**trust**

[azure.microsoft.com/blog/microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/](https://azure.microsoft.com/blog/microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/)

# Myth #38

Rust already in the [Windows 11](#) kernel (May 2023)

```
C:\Windows\System32>dir win32k*
Volume in drive C has no label.
Volume Serial Number is E60B-9A9E

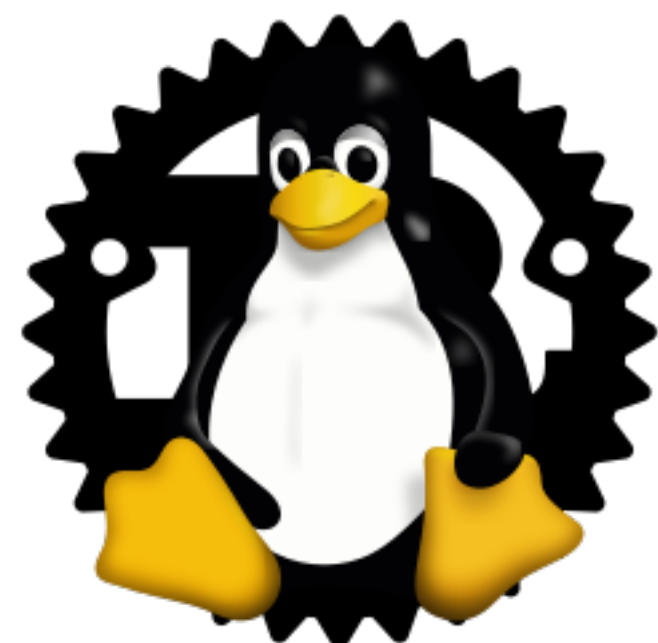
Directory of C:\Windows\System32

04/15/2023  09:50 PM                708,608 win32k.sys
04/15/2023  09:49 PM            3,424,256 win32kbase.sys
04/15/2023  09:49 PM            110,592 win32kbase_rs.sys
04/15/2023  09:50 PM            4,194,304 win32kfull.sys
04/15/2023  09:49 PM             40,960 win32kfull_rs.sys
04/15/2023  09:49 PM             69,632 win32kuis.sys
04/15/2023  09:49 PM             98,304 win32ksgd.sys
              7 File(s)            8,646,656 bytes
              0 Dir(s)  116,366,049,280 bytes free
```

**\_rs = Rust!**

# Myth #38

Rust in the Linux kernel (since 6.1)  
-- with Linus Torvalds' blessing



The first Rust modules start to make their way into the Linux kernel (6.3+)

Ubuntu has done all the work to provide the right toolchain in the distro and custom kernel patches (SAUCE) that allow easier acquisition and build of Rust modules.



[wikipedia.org/wiki/Rust\\_for\\_Linux](https://wikipedia.org/wiki/Rust_for_Linux)



# Myth #38

So this happened 🤖 (April 2023)

Porting **Windows 11** core components from C++ to Rust

- **DirectWrite**
- **GDI**

[youtube.com/watch?v=8T6CIX-y2AE&t=2703s](https://youtube.com/watch?v=8T6CIX-y2AE&t=2703s)



## Rust in Windows: Crawl

- Learn by doing: Exploration → Flighting → Production (crawl → walk → run)
- Direct impact: Improve security
- Indirect impact: Gain experience with transitioning to Rust in production
  - Costs of learning Rust?
  - Costs of porting Rust?
  - Costs of writing new Rust?
  - Is the full pipeline of Rust tooling ready? Debugging, perf, cross-platform, POGO, etc.
  - Costs of maintaining a hybrid C++/Rust codebase?

## What is DWrite? What is DWriteCore?

- Full stack for text analysis, layout, and rendering
  - Ships in Windows (dwrite.dll)
  - Handles all major languages and scripts
  - Huge amount of inherent complexity: complex scripts, complex glyph descriptions
- DWriteCore is DWrite “undocked” from Windows
  - Builds outside of Windows repo
  - Cross-platform: Windows, Linux, Android, iOS, Mac OS
  - Office contains an old fork (dwrite10), is migrating to DWriteCore for some platforms
    - All new feature development in DWrite has shifted to DWriteCore
- Collaboration between Rust team and DWrite team begin in 2020
- DWriteCore is now ~152 KLOC of Rust, ~96 KLOC of C++

1st  
experiment

[youtube.com/watch?v=8T6CIX-y2AE&t=2703s](https://youtube.com/watch?v=8T6CIX-y2AE&t=2703s)

## Interop Rust and C++

- DWriteCore internally uses COM-like interfaces. These were a good integration point for C++/Rust, and provided natural boundaries for incremental porting.
- DWriteCore public APIs are all COM. In some cases, Rust code is directly callable from app code, through COM interfaces.

```
DWRITE_BEGIN_INTERFACE(INumberSubstitution,  
    "9d5d67e0-7bde-4f6d-a073-360c5c381dd6") : IDWriteNumberSubstitution  
{  
    virtual NumberSubstitutionMode GetMode() const = 0;  
    virtual NumberSubstitutionChars const& GetChars() const = 0;  
    virtual uint32_t GetScript() const = 0;  
};
```

```
com::interfaces! {  
    #[uuid("9d5d67e0-7bde-4f6d-a073-360c5c381dd6")]  
    pub unsafe interface INumberSubstitution : IDWriteNumberSubstitution {  
        pub fn GetMode(&self) -> NumberSubstitutionMode;  
        pub fn GetChars(&self) -> *const NumberSubstitutionChars;  
        pub fn GetScript(&self) -> u32;  
    }  
}
```

◆ In other places, we statically link Rust and C++ code.

```
extern "C" IDWriteInlineObject* Rust_Layout_CreateInlineObject(  
    IDWriteTextLayout *layout,  
    InlineLayoutBoundMode boundMode,  
    bool adjustBaseline);
```

```
#[no_mangle]  
pub extern "C" fn Rust_Layout_CreateInlineObject(  
    layout: IDWriteTextLayout,  
    bound_mode: InlineLayoutBoundMode,  
    adjust_baseline: bool,  
) -> IDWriteInlineObject {  
    ...  
}
```

## Win32k **GDI** port to Rust

2nd  
experiment

- Ported the REGION data type and functions
  - Models overlapping controls (e.g., windows) in GDI.
  - “Leaf node” data type: few dependencies, many dependents.
  - Old (late 80s, early 90s), and perf critical (designed for a 286/386).
  - Maintenance nightmare: open-coded vector resizing and ref-counting.
- Currently disabled via a feature-flag.
- Windows boots with the Rust version, and all GDI tests pass.

[youtube.com/watch?v=8T6CIX-y2AE&t=2703s](https://youtube.com/watch?v=8T6CIX-y2AE&t=2703s)

## Win32k **GDI** port to Rust

- Perf of the ported code has been excellent
  - No perf difference in Office apps (as measured by PCMark 10).
  - Micro-benchmarks show mostly no differences, with some wins for Rust.
- Has driven changes upstream in Rust
  - More try\_ methods for Vec that don't panic on OOM:  
<https://github.com/rust-lang/rust/pull/95051>
- Calls to extern functions means there's a lot of "unsafe" code
  - Currently 163 unsafe functions (~10%) and 271 unsafe blocks.
  - But as we port more code, these have been disappearing.
  - We've even been able to write a SysCall is completely safe code.

## Rust Fact vs. Fiction

### 5 Insights from Google's Rust journey in 2022

Rumor 1: Rust takes more than 6 months to **learn** – **Debunked**

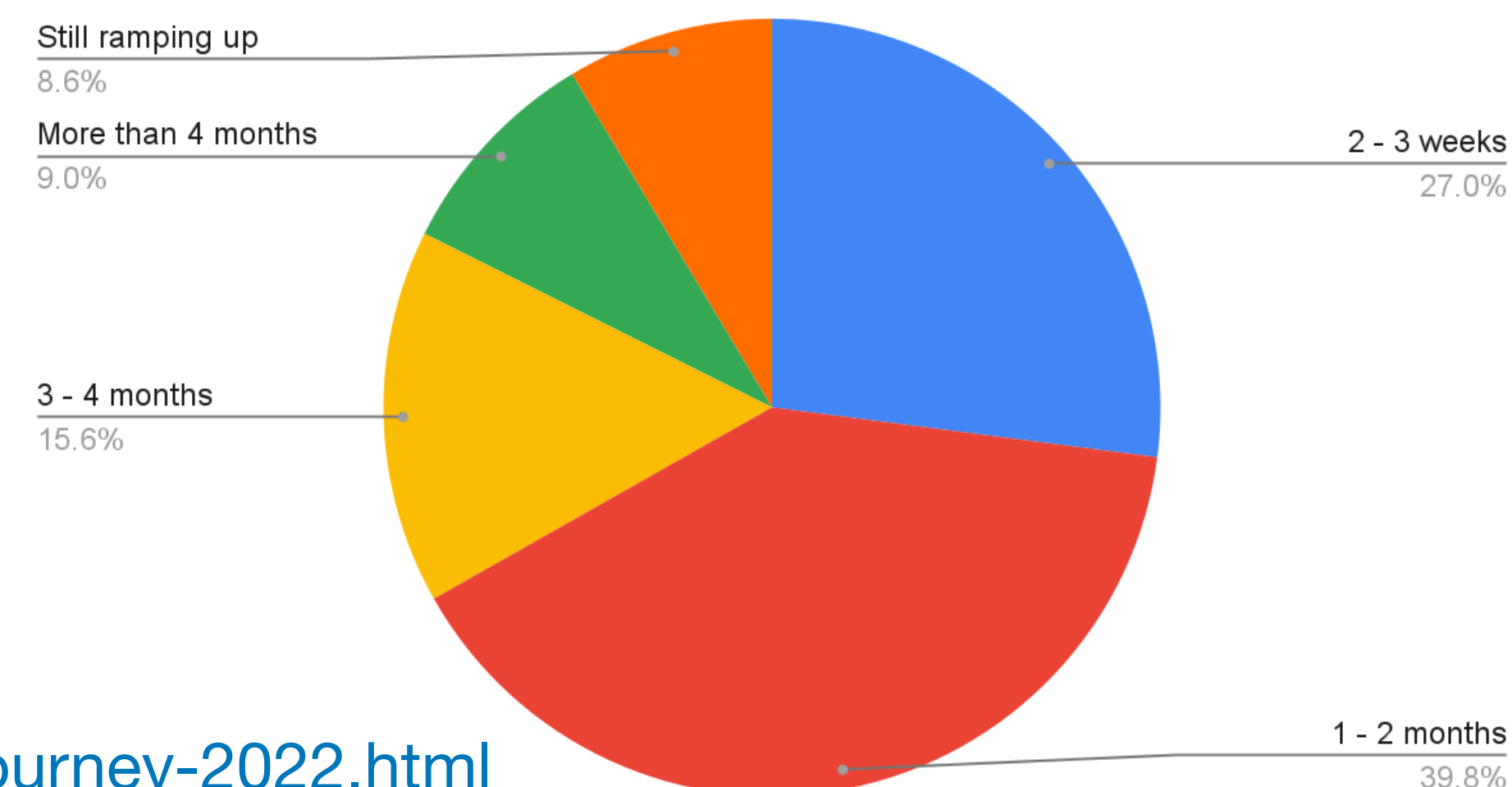
Rumor 2: The Rust **compiler is not as fast** as people would like – **Confirmed**

Rumor 3: **Unsafe** code and **interop** are always the biggest challenges – **Debunked**

Rumor 4: Rust has amazing compiler **error messages** – **Confirmed**

Rumor 5: Rust code is **high quality** – **Confirmed**

Time until confident writing Rust





## Chromium: Rust and C++ interoperability

It's important for Rust to be able to call C++ functions in a way that meets the following criteria:

- No need for `unsafe` keyword
- No overhead in the general case
- No boilerplate or redeclarations / No C++ annotations
- Broad type support - with safety
- Ergonomics - with safety

There's progress in Rust community in solving some of these problems:

➔ see [moveit](#), [autocxx](#) and [mosaic](#)

[chromium.org/Home/chromium-security/memory-safety/rust-and-c-interoperability/](https://chromium.org/Home/chromium-security/memory-safety/rust-and-c-interoperability/)



# Myth #38

unsafe { 😱 }

Rust has **more UB** than C++ in `unsafe{}` because it always assumes pointers do not alias.

The screenshot displays a video player interface with four panels. The top-left panel shows Rust source code for a function `foo` that takes two mutable pointers and updates their values. The top-right panel shows the assembly output for this Rust code, where the second pointer is not used to update its value. The bottom-left panel shows the equivalent C code using `restrict` pointers. The bottom-right panel shows the assembly output for the C code, where the second pointer is used to update its value. A red line highlights the difference in pointer aliasing between the two versions. A video player control bar at the bottom shows the video is at 20:00 / 45:39.

Unsafe Rust is not C

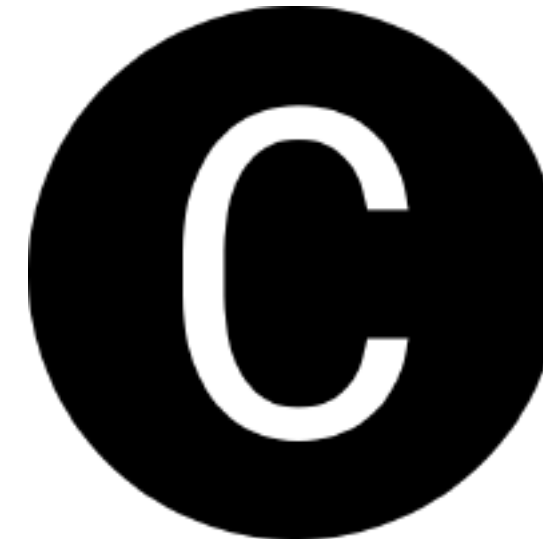
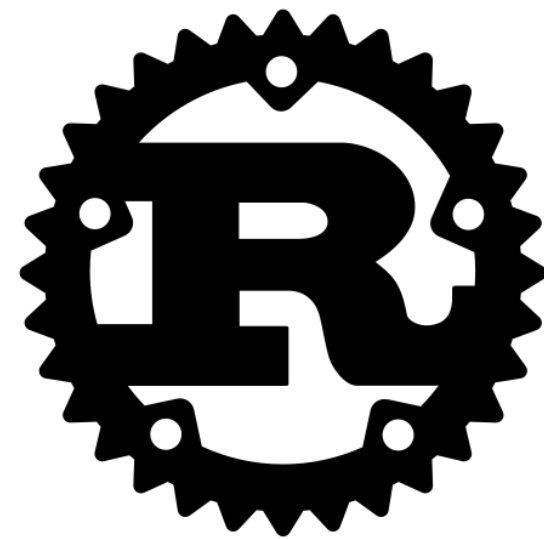
[youtube.com/watch?v=DG-VLezRkYQ](https://youtube.com/watch?v=DG-VLezRkYQ)

# Myth #38

Just rewrite it in Rust 🦀



Successor languages are going to eat our lunch

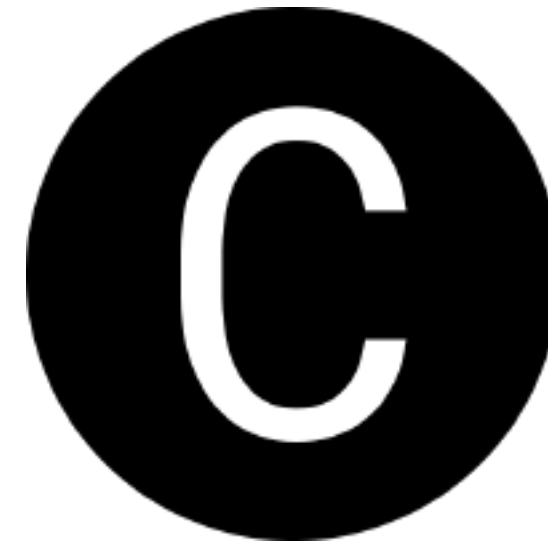


# Myth #6



## Val aims:

- fast by definition
- safe by default
- simple
- interoperable with C++
- whole/part relationships
- mutable value semantics
- **Swift**, as it should have been



## Carbon aims:

- interoperability with C++
- better defaults than C++
- no function overloading
- no exception handling
- no multiple inheritance
- doesn't handle raw pointers
- doesn't have constructors



**The Year of C++ Successor Languages**

-- Lucian Radu Teodorescu

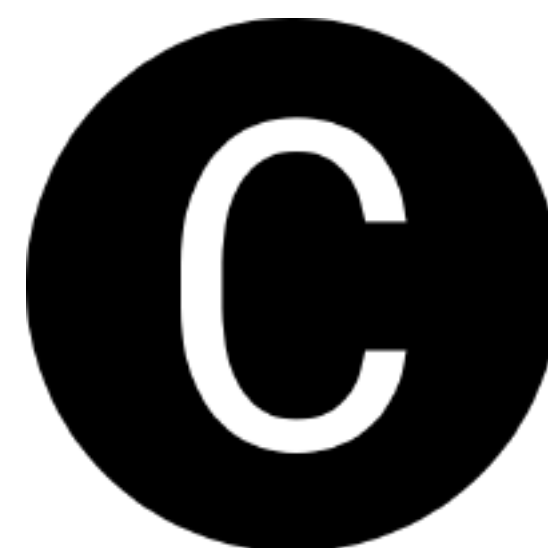
[accu.org/journals/overload/30/172/teodorescu/](https://accu.org/journals/overload/30/172/teodorescu/)

# Myth #6



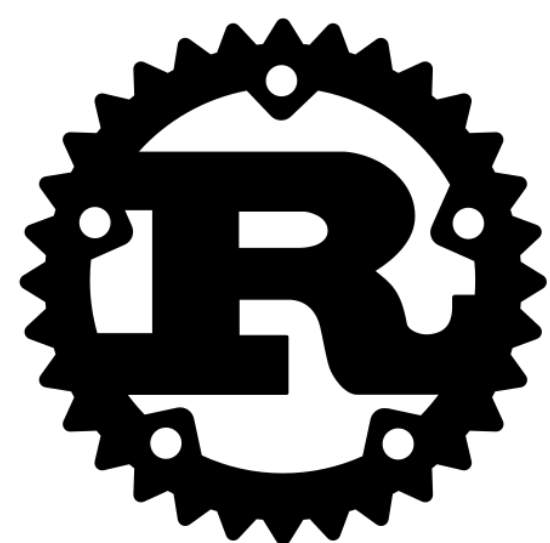
## Val aims:

- fast by definition
- safe by default
- simple
- interoperable with C++
- whole/part relationships
- mutable value semantics
- **Swift**, as it should have been



## Carbon aims:

- interoperability with C++
- better defaults than C++
- no function overloading
- no exception handling
- no multiple inheritance
- doesn't handle raw pointers
- doesn't have constructors



✓ perfect by construction :)



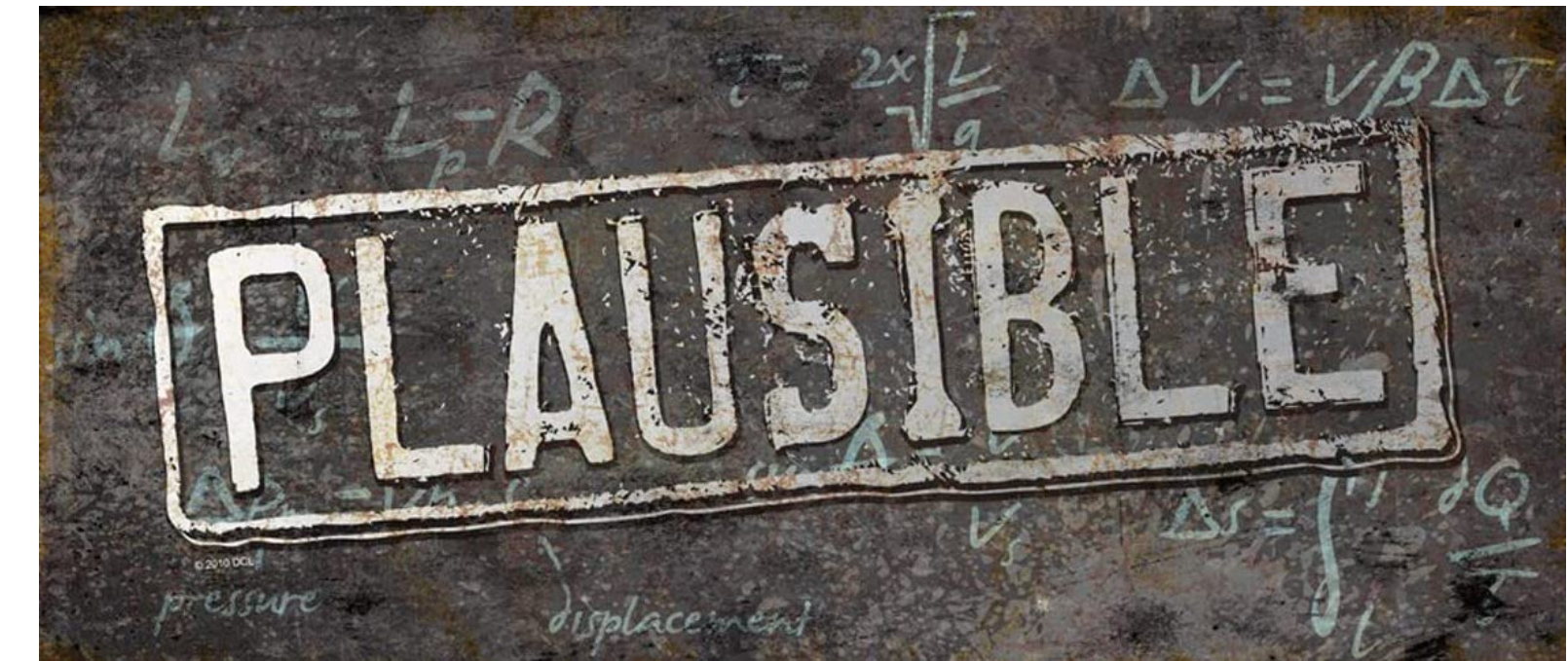
**The Year of C++ Successor Languages**

-- Lucian Radu Teodorescu

[accu.org/journals/overload/30/172/teodorescu/](https://accu.org/journals/overload/30/172/teodorescu/)

# Myth #6

Successor languages are going to eat our lunch



\* it's not a zero sum game - there will be enough food for everyone

`std::ranges` are safer than iterators

All our experience with iterators since the 90s, tells us they should be 😊

# Myth #39

C++20 ranges library is fantastic tool, but watch out for gotchas ⚠

- views have *reference* semantics => all the reference gotchas apply
- as always with C++, **const** is *shallow* and doesn't propagate (as you might expect)
- some functions do *caching*, eg. `begin()`, `empty()`, `| filter | drop`
- don't hold on to *views* or try to reuse them
  - safest to use them *ad-hoc*, as temporaries
  - if needed, better "copy" them (cheap) for reuse

\* the Nico slide :)



### Basic Idioms Broken by Standards Views

C++20/C++23

- You can **iterate** if the range is **const**
- A **read iteration** does **not change** state
- **Concurrent** read **iterations** are safe
- **const** collections have **const** elements
- **cbegin()** makes elements immutable
- A **copy of a range** has the same state
- **const-declared** elements are **const** (C++23)

Broken  
for views

Broken  
for views

Broken  
for views

Broken  
for views

Broken  
for views

Broken  
for views

Broken  
for views



Nico Josuttis

[youtube.com/watch?v=qv29fo9sUjY](https://youtube.com/watch?v=qv29fo9sUjY)

## Ranges & filter predicate invariant

- **Main use case of a filter:**

- Fix an attribute that some elements might have

has **undefined behavior**: [range.filter.iterator]:

**Modification** of the element a `filter_view::iterator` denotes is permitted, but **results in undefined behavior** if the resulting value does not satisfy the filter predicate.

// as a shaman:

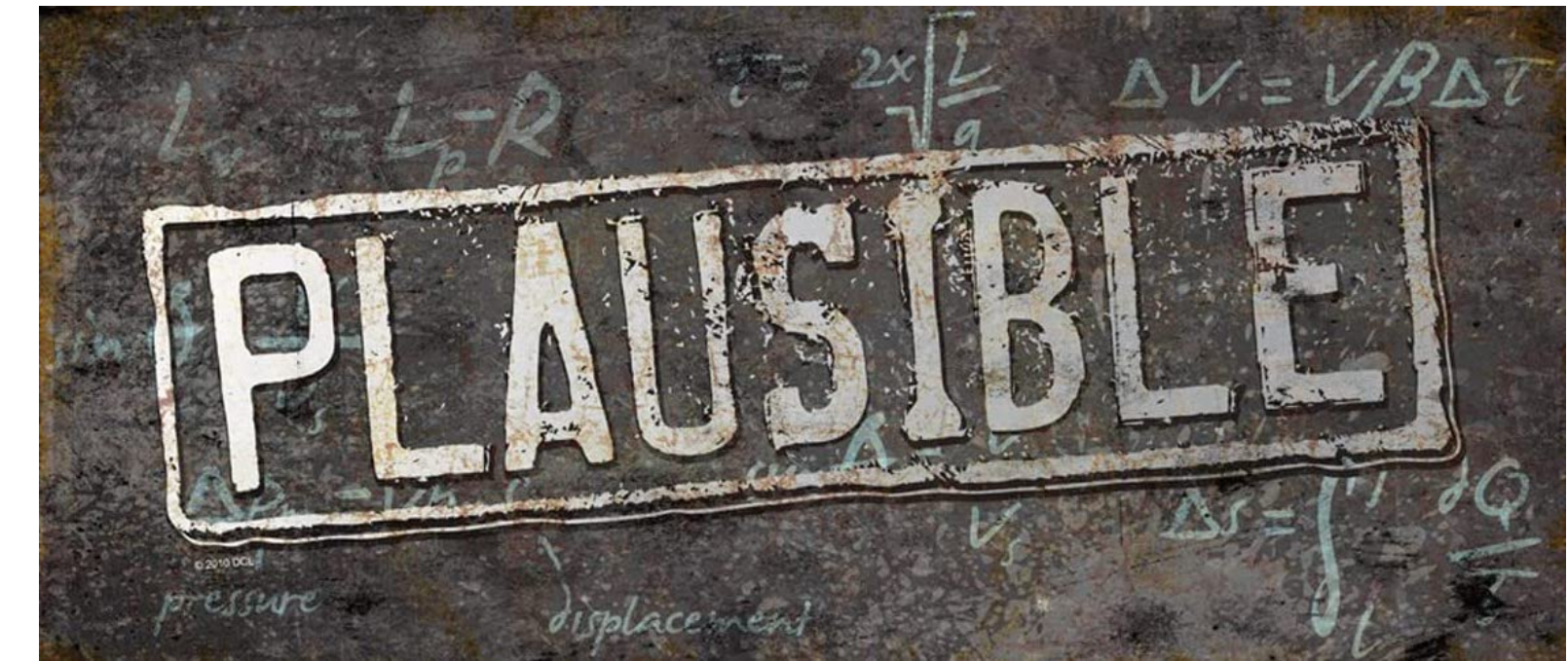
```
for (auto& m : monsters | std::views::filter(isDead)) {  
    m.resurrect(); // undefined behavior: because no longer dead  
    m.burn();     // OK (because it is still dead)  
}
```

Thanks to Patrice Roy for this example

[youtube.com/watch?v=qv29fo9sUjY](https://youtube.com/watch?v=qv29fo9sUjY)

# Myth #39

`std::ranges` are safer than iterators



CMake is the gold standard of C++ project systems

# Myth #7

CMake:

"When it works, it's great; when it doesn't you're regretting your life decisions" 😊

[https://twitter.com/pati\\_gallardo/status/1672137915575545856?s=46&t=dcjdCXT0jeVLLjXhQ3J85A](https://twitter.com/pati_gallardo/status/1672137915575545856?s=46&t=dcjdCXT0jeVLLjXhQ3J85A)

## CMake Debugger in Visual Studio and VSCode



[youtube.com/watch?v=1eVJBEV9NTk](https://youtube.com/watch?v=1eVJBEV9NTk)

# Myth #7

The CMake debugger has now been implemented in VS Code and merged upstream to Kitware.

CMake Debugger: [VS](#) + [VSCode](#) + [Rider](#) + [CLion](#)

# Myth #7

CMake is the gold standard of C++ project systems



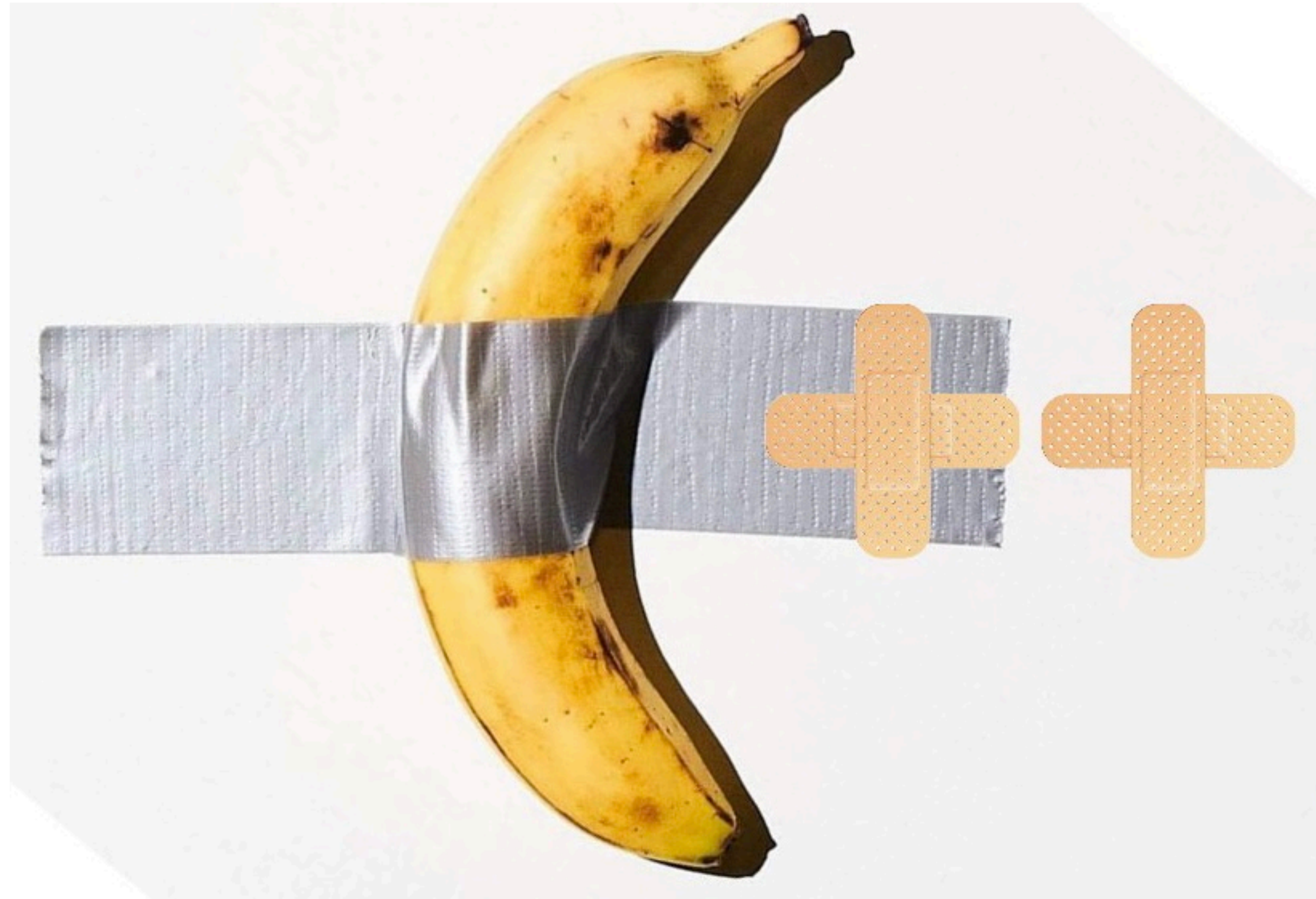


**New** (C++) is the enemy of the old

"Before we had [\[feature\]](#), we were nonetheless able to program in C++"

- *Pablo Halpern, ACCU Conf 2022 (via Kate Gregory)*

# New (C++) is the enemy of the old



[twitter.com/tvaneerd/status/1387](https://twitter.com/tvaneerd/status/1387)

## Other C++ Mythbusters

# Other C++ Mythbusters

The image shows a video player interface. The main content area displays a presentation slide with the following text:

Some Programming Myths Revisited

Patrice Roy  
[Patrice.Roy@USherbrooke.ca](mailto:Patrice.Roy@USherbrooke.ca)  
CeFTI, Université de Sherbrooke  
[Patrice.Roy@clg.qc.ca](mailto:Patrice.Roy@clg.qc.ca)  
Collège Lionel-Groulx

The video player includes a progress bar at the bottom showing 0:25 / 1:00:50. To the right of the main content is a video thumbnail showing Patrice Roy at a podium. The thumbnail includes the Cppcon 2019 logo (The C++ Conference, cppcon.org) and the text "Patrice Roy" and "Some Programming Myths Revisited". Below the thumbnail, it says "Video Sponsorship Provided By: ansatz".

📍 AURORA

Some Programming Myths Revisited - Patrice Roy - CppCon 2019

[youtube.com/watch?v=KNqRjzSIUVo](https://youtube.com/watch?v=KNqRjzSIUVo)

# Other C++ Mythbusters



Andrey Karpov

May 30 2023

## 60 terrible tips for a C++ developer

- Terrible tip N1. Only C++
- Terrible tip N2. Tab character in string literals
- Terrible tip N3. Nested macros
- Terrible tip N4. Disable warnings
- Terrible tip N5. The shorter the variable name is, the better
- Terrible tip N6. Invisible characters
- Terrible tip N7. Magic numbers
- Terrible tip N8. int, int everywhere
- Terrible tip N9. Global variables
- Terrible tip N10. The abort function in libraries
- Terrible tip N11. The compiler is to blame for everything
- Terrible tip N12. Feel free to use argv
- Terrible tip N13. Undefined behavior is just a scary story
- Terrible tip N14. double == double
- Terrible tip N15. memmove is a superfluous function
- Terrible tip N16. sizeof(int) == sizeof(void \*)
- Terrible tip N17. Don't check what the malloc function returned
- Terrible tip N18. Extend the std namespace
- Terrible tip N35. Declaring variables at the beginning of a function
- Terrible tip N36. Add everything, it might come in handy
- Terrible tip N37. Create your own h-quest
- Terrible tip N38. C-style cast
- Terrible tip N39. Versatility is cool
- Terrible tip N40. You are the lord of pointers — do what you want
- Terrible tip N41. const is a redundant entity
- Terrible tip N42. Vintage is cool
- Terrible tip N43. Don't initialize
- Terrible tip N44. Trust everyone
- Terrible tip N45. Don't worry about naming variables
- Terrible tip N46. Write your code as if you are training for the IOCCC
- Terrible tip N47. Have fun when writing code
- Terrible tip N48. Everyone has their own style
- Terrible tip N49. Overload everything
- Terrible tip N50. Don't believe in the efficiency of std::string
- Terrible tip N51. For as long as possible, resist using the new C++ standard
- Terrible tip N52. Variables Reuse
- Terrible tip N53. Answer the question "what?" in code comments
- Terrible tip N54. More multithreading
- Terrible tip N55. The fewer .cpp files, the better
- Terrible tip N56. More classes!
- Terrible tip N57. Reading books is no longer relevant
- Terrible tip N58. printf(str);
- Terrible tip N59. Virtual functions in constructors and destructors
- Terrible tip N60. No time to think, copy the code!
- Terrible tip N61. You can look beyond the array

[pvs-studio.com/en/blog/posts/cpp/1053/](https://pvs-studio.com/en/blog/posts/cpp/1053/)



June 2023

 @ciura\_victor

 @ciura\_victor@hachyderm.io

**Victor Ciura**  
Principal Engineer  
Visual C++

