C++ Prague Meetup

September 2023

@ciura_victor
@ciura_victor@hachyderm.io

Victor Ciura
Principal Engineer
Visual C++

Myths, Dogma and Practice

C++ Prague Meetup
September 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.

Q & A

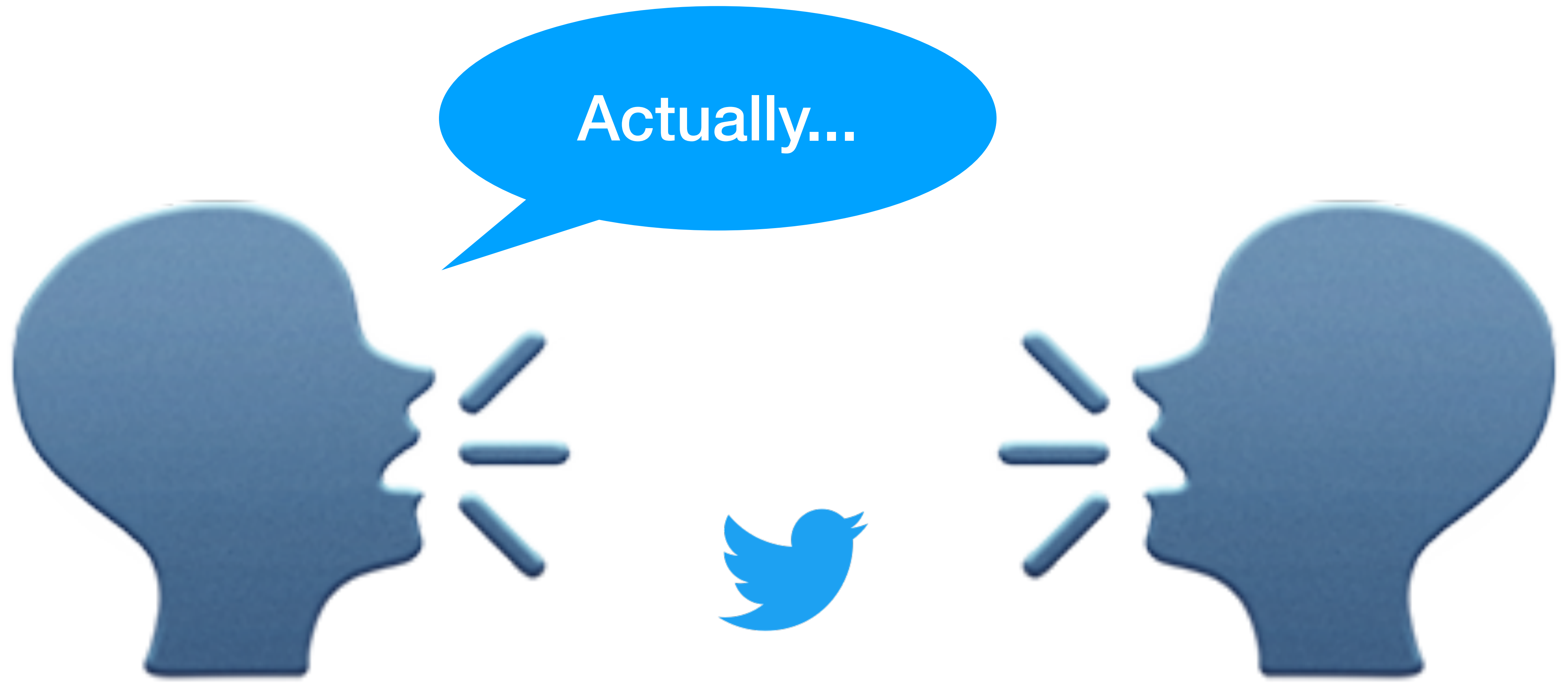
Do ask questions as we go along

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

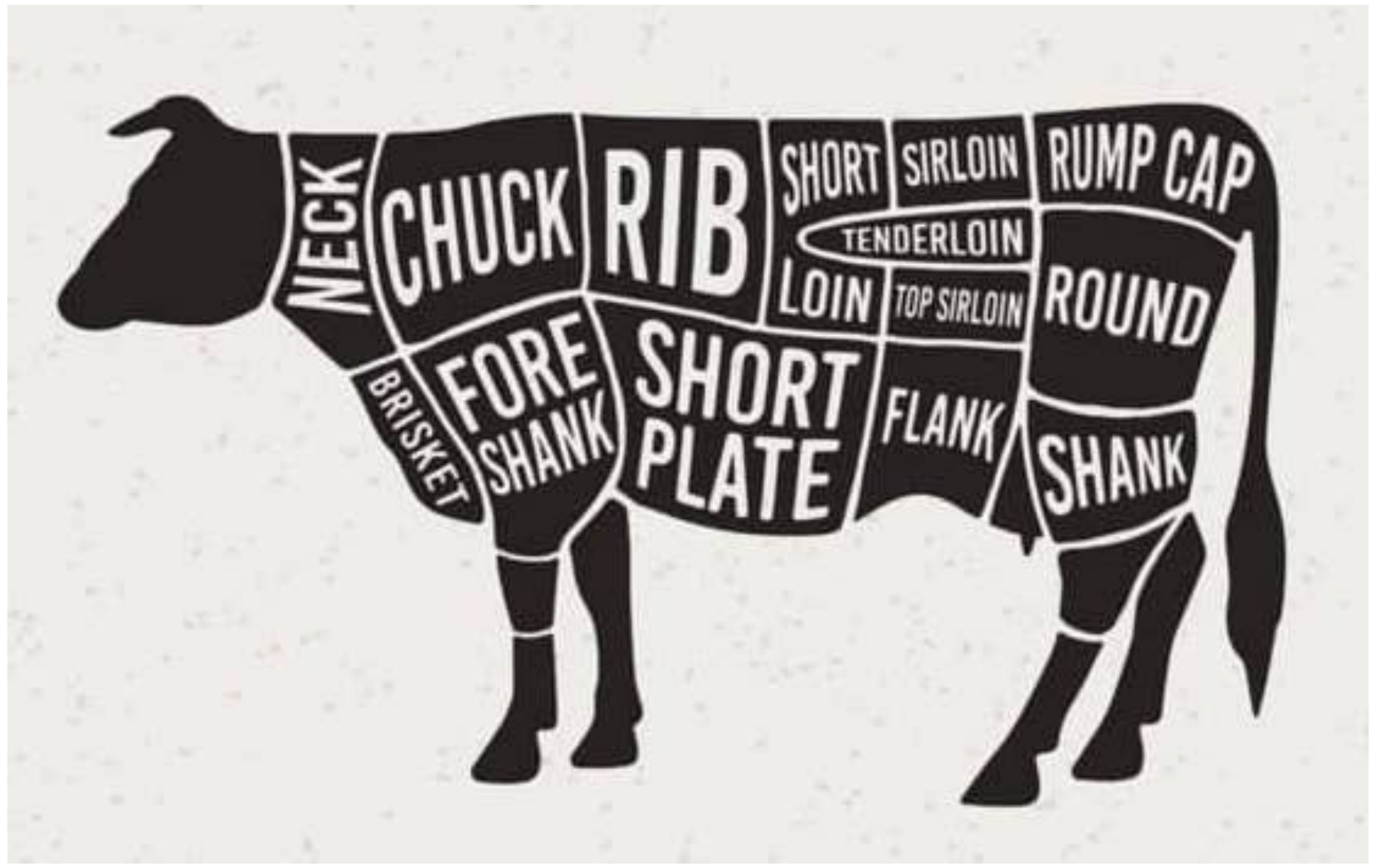
We're Different

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)



Sources

A lot of good information easily available:

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

Myths

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)

Help 😸

Hi, I have problems catching mice, any advice please?
Catching mice is no longer considered a best practice.
Recommend upgrading to relying on humans.
Covered by "How to stalk birds." Marked as duplicate.

StackOverflow

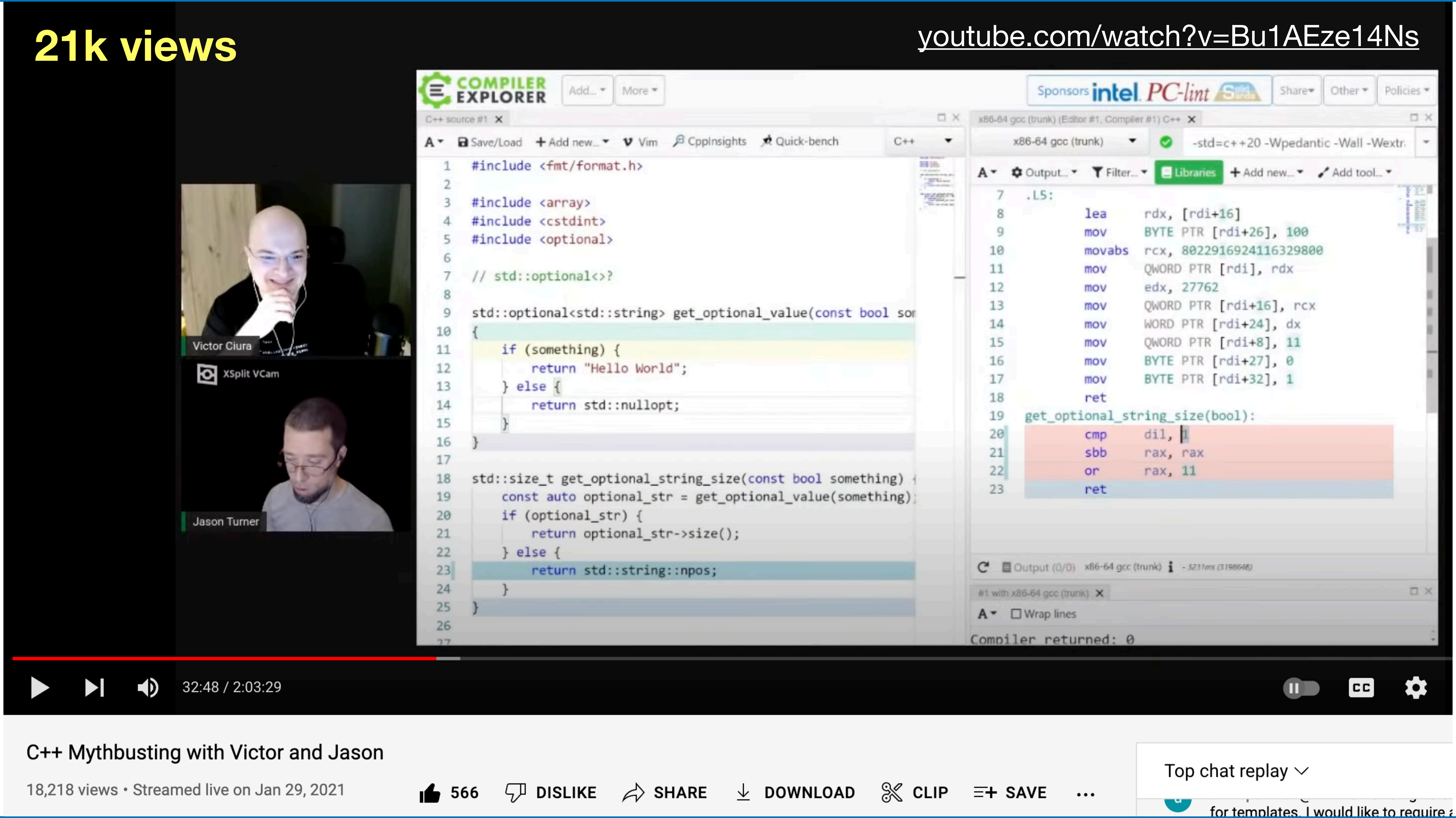
Motivation

How it started...

Mythbusting with Jason - unscripted improv (Pandemic edition)

21k views

youtube.com/watch?v=Bu1AEze14Ns



```cpp
#include <fmt/format.h>

#include <array>
#include <cstdint>
#include <optional>

// std::optional<>?

std::optional<std::string> get_optional_value(const bool sor
{
    if (something) {
        return "Hello World";
    } else {
        return std::nullopt;
    }
}

std::size_t get_optional_string_size(const bool something) {
    const auto optional_str = get_optional_value(something);
    if (optional_str) {
        return optional_str->size();
    } else {
        return std::string::npos;
    }
}
```

```asm
.L5:
        lea     rdx, [rdi+16]
        mov     BYTE PTR [rdi+26], 100
        movabs  rcx, 8022916924116329800
        mov     QWORD PTR [rdi], rdx
        mov     edx, 27762
        mov     QWORD PTR [rdi+16], rcx
        mov     WORD PTR [rdi+24], dx
        mov     QWORD PTR [rdi+8], 11
        mov     BYTE PTR [rdi+27], 0
        mov     BYTE PTR [rdi+32], 1
        ret
get_optional_string_size(bool):
        cmp     dil, 1
        sbb     rax, rax
        or      rax, 11
        ret
```

```
Compiler returned: 0
```

C++ Mythbusting with Victor and Jason

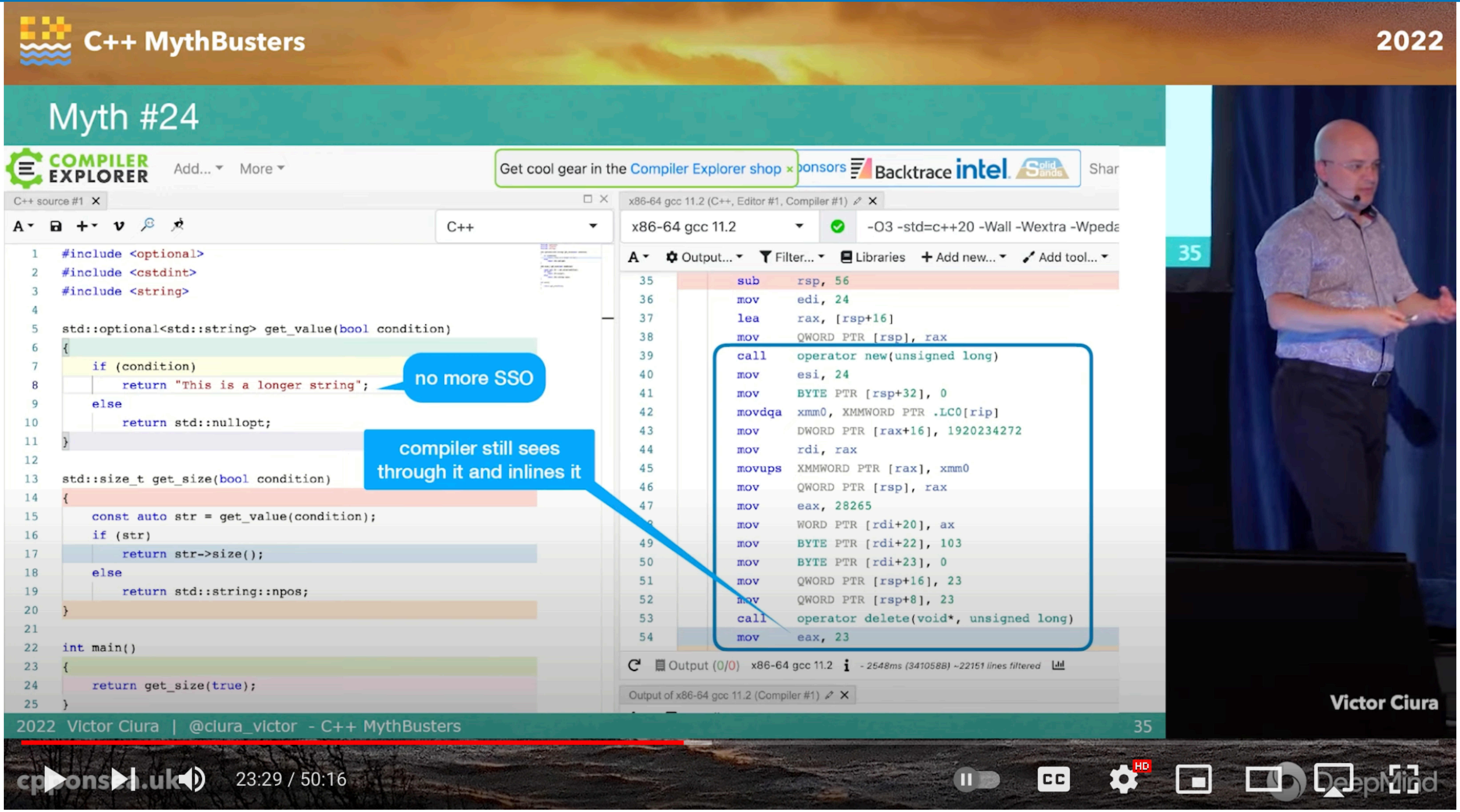
18,218 views • Streamed live on Jan 29, 2021

566 DISLIKE SHARE DOWNLOAD CLIP SAVE

Top chat replay

for templates, I would like to require

C++ Mythbusters - Season 1



Myth #24

```cpp
#include <optional>
#include <cstdint>
#include <string>

std::optional<std::string> get_value(bool condition)
{
    if (condition)
        return "This is a longer string";
    else
        return std::nullopt;
}

std::size_t get_size(bool condition)
{
    const auto str = get_value(condition);
    if (str)
        return str->size();
    else
        return std::string::npos;
}

int main()
{
    return get_size(true);
}
```

no more SSO

compiler still sees through it and inlines it

```
        sub     rsp, 56
        mov     edi, 24
        lea     rax, [rsp+16]
        mov     QWORD PTR [rsp], rax
        call    operator new(unsigned long)
        mov     esi, 24
        mov     BYTE PTR [rsp+32], 0
        movdqa  xmm0, XMMWORD PTR .LC0[rip]
        mov     DWORD PTR [rax+16], 1920234272
        mov     rdi, rax
        movups  XMMWORD PTR [rax], xmm0
        mov     QWORD PTR [rsp], rax
        mov     eax, 28265
        mov     WORD PTR [rdi+20], ax
        mov     BYTE PTR [rdi+22], 103
        mov     BYTE PTR [rdi+23], 0
        mov     QWORD PTR [rsp+16], 23
        mov     QWORD PTR [rsp+8], 23
        call    operator delete(void*, unsigned long)
        mov     eax, 23
```

2022 Victor Ciura | @ciura_victor - C++ MythBusters 35

youtube.com/watch?v=ZGgrUhVNsSI

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

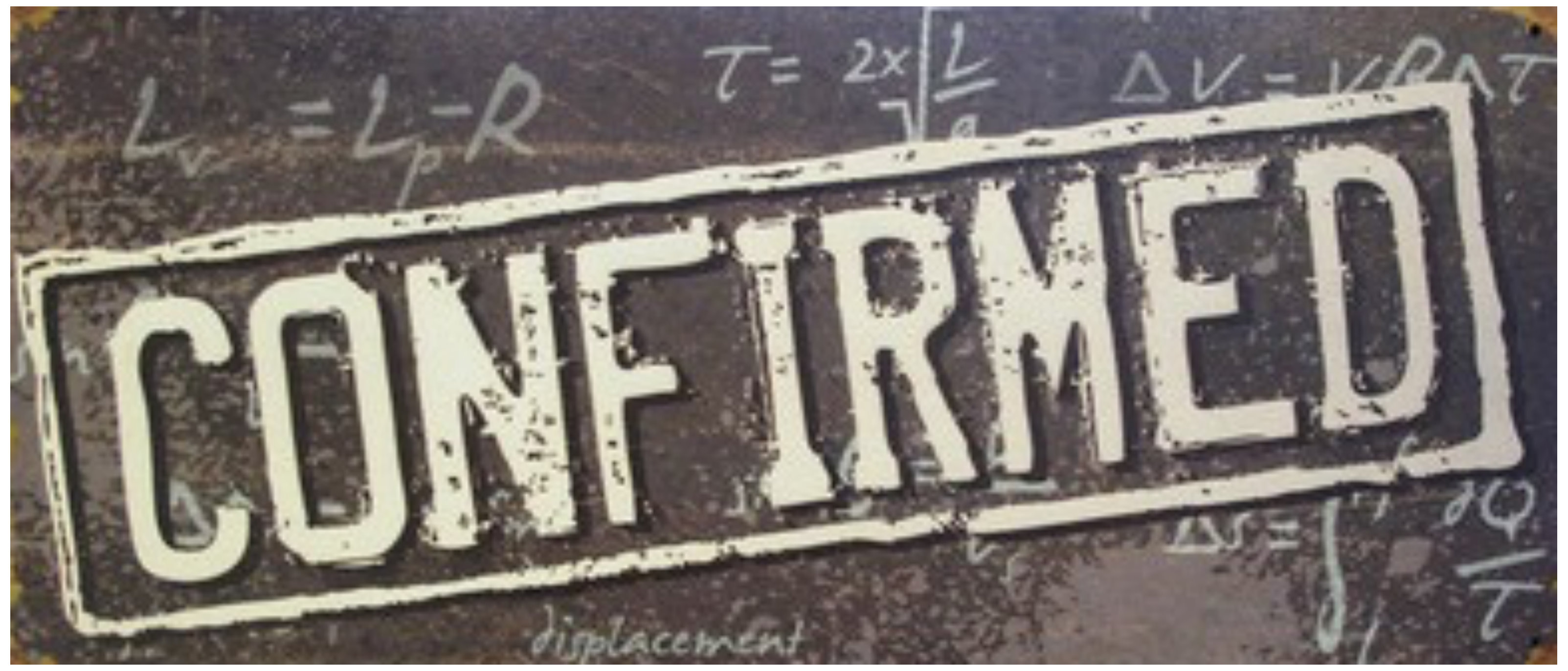
youtube.com/watch?v=ZGgrUhVNsSI

Motivation

I want to instigate a healthy dialog,

so speak up

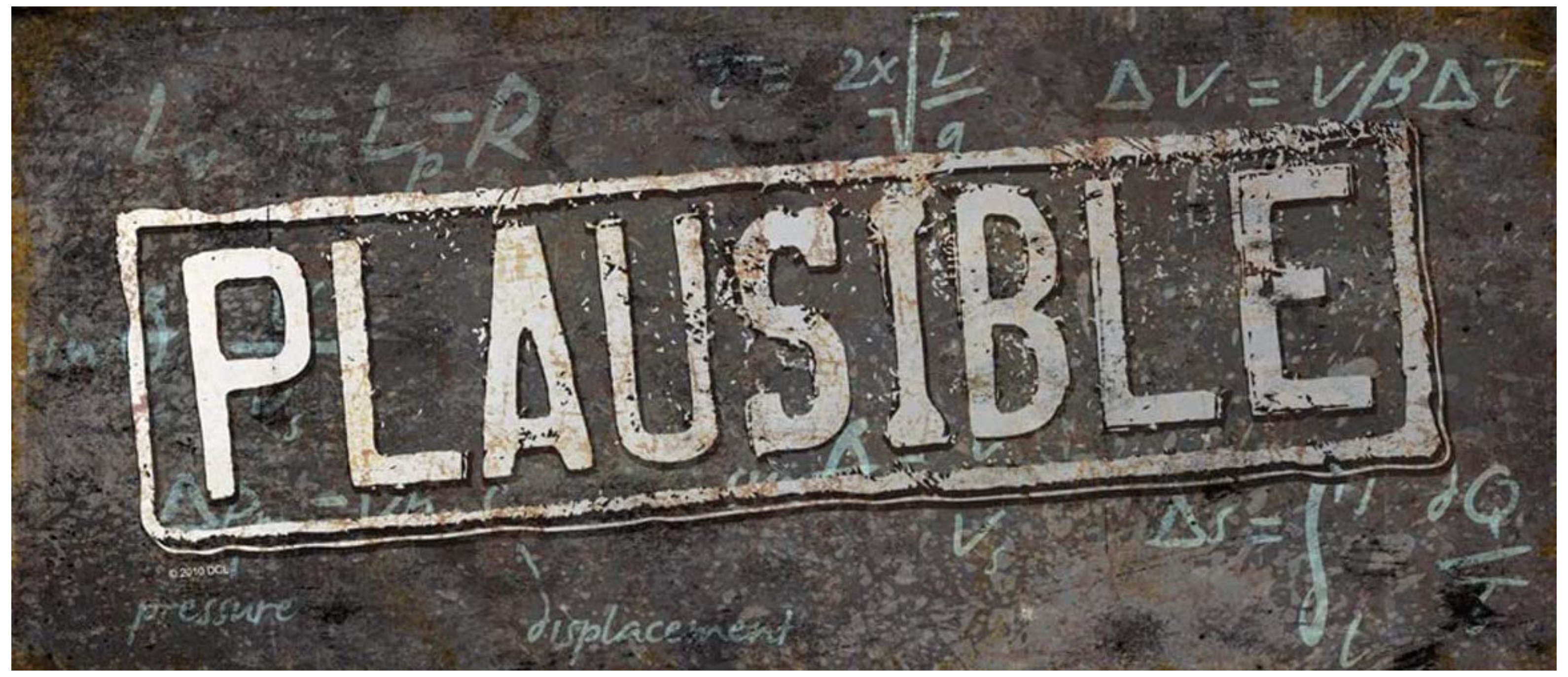
Verdict



Verdict

A programmer's staple response:

"It depends..." 🤓

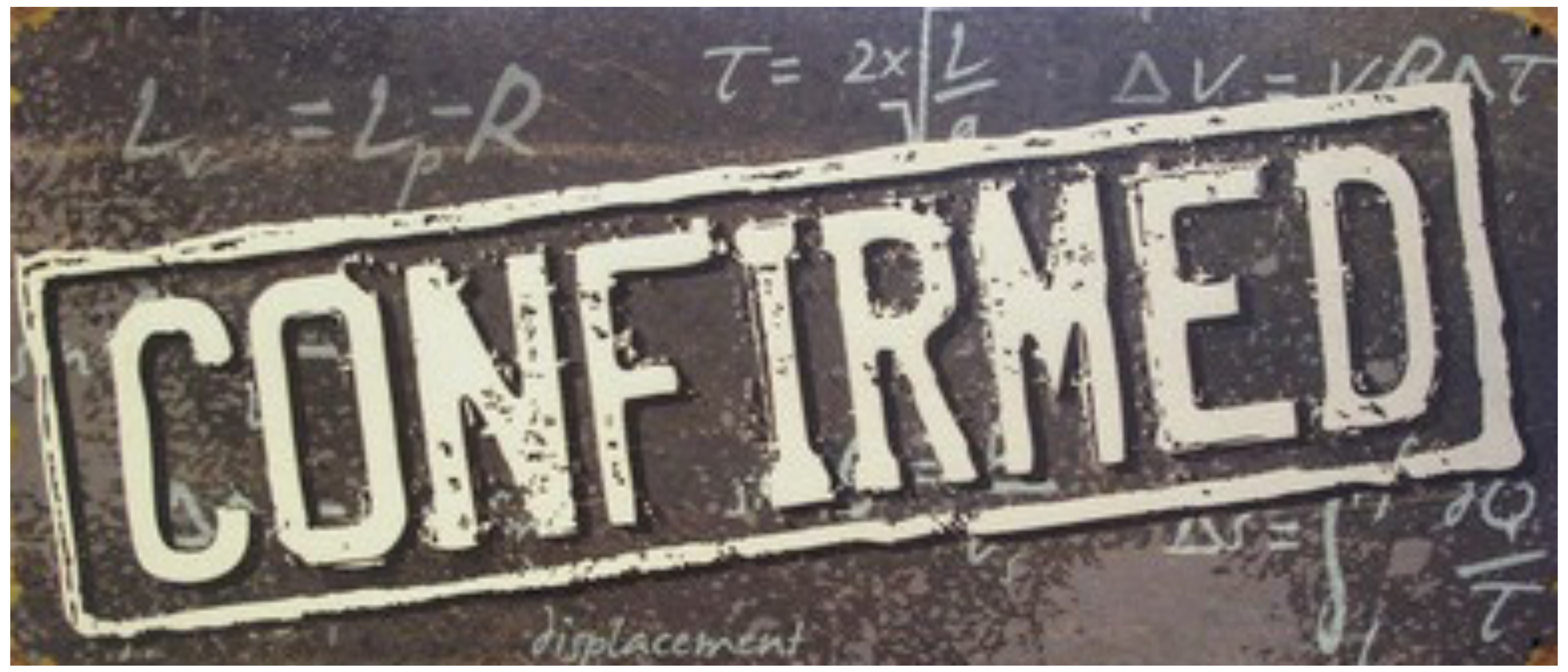


Verdict

Let's test this...

Test Myth

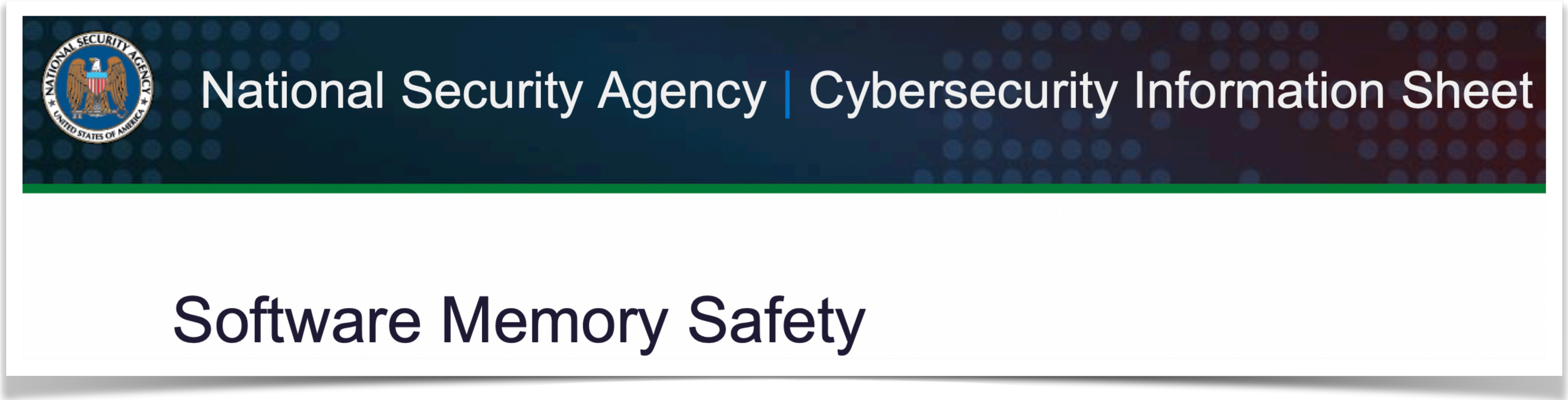
C++ is inherently unsafe and there's very little* we can do about it



Just kidding 😄

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

media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF



National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

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.

Where are all the compilers?!

Test Myth

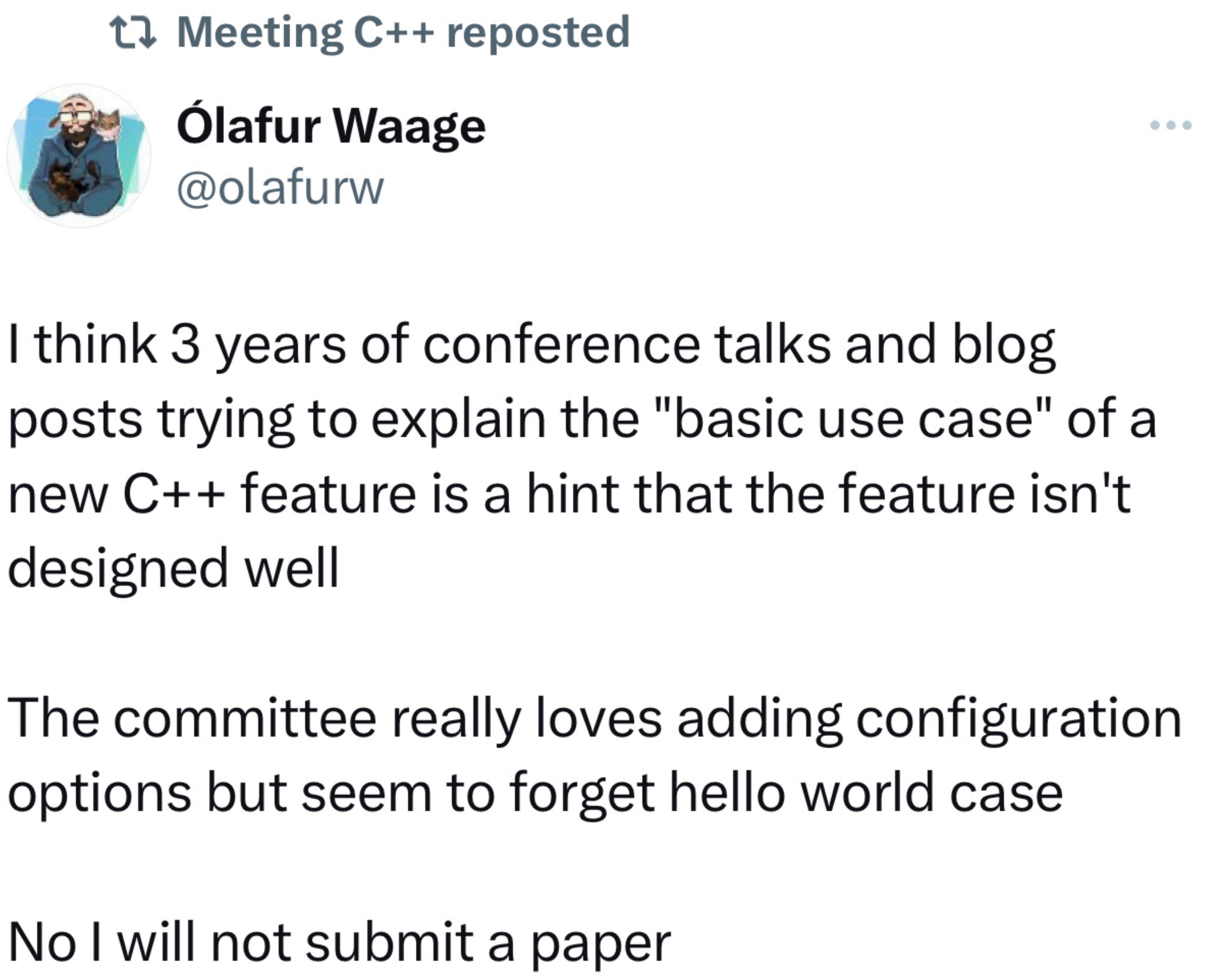
It's 2023, we should be able to leverage the power of C++20 modules to (re)structure our codebase and improve build times.

Where are all the compilers?!

Integrating C++ header units into **Office** using MSVC (Part 2).
The path to a clean code structure and better build throughput.
devblogs.microsoft.com/cppblog/integrating-c-header-units-into-office-using-msvc-2-n/

Test Myth

coroutines shipped in C++20


Meeting C++ reposted

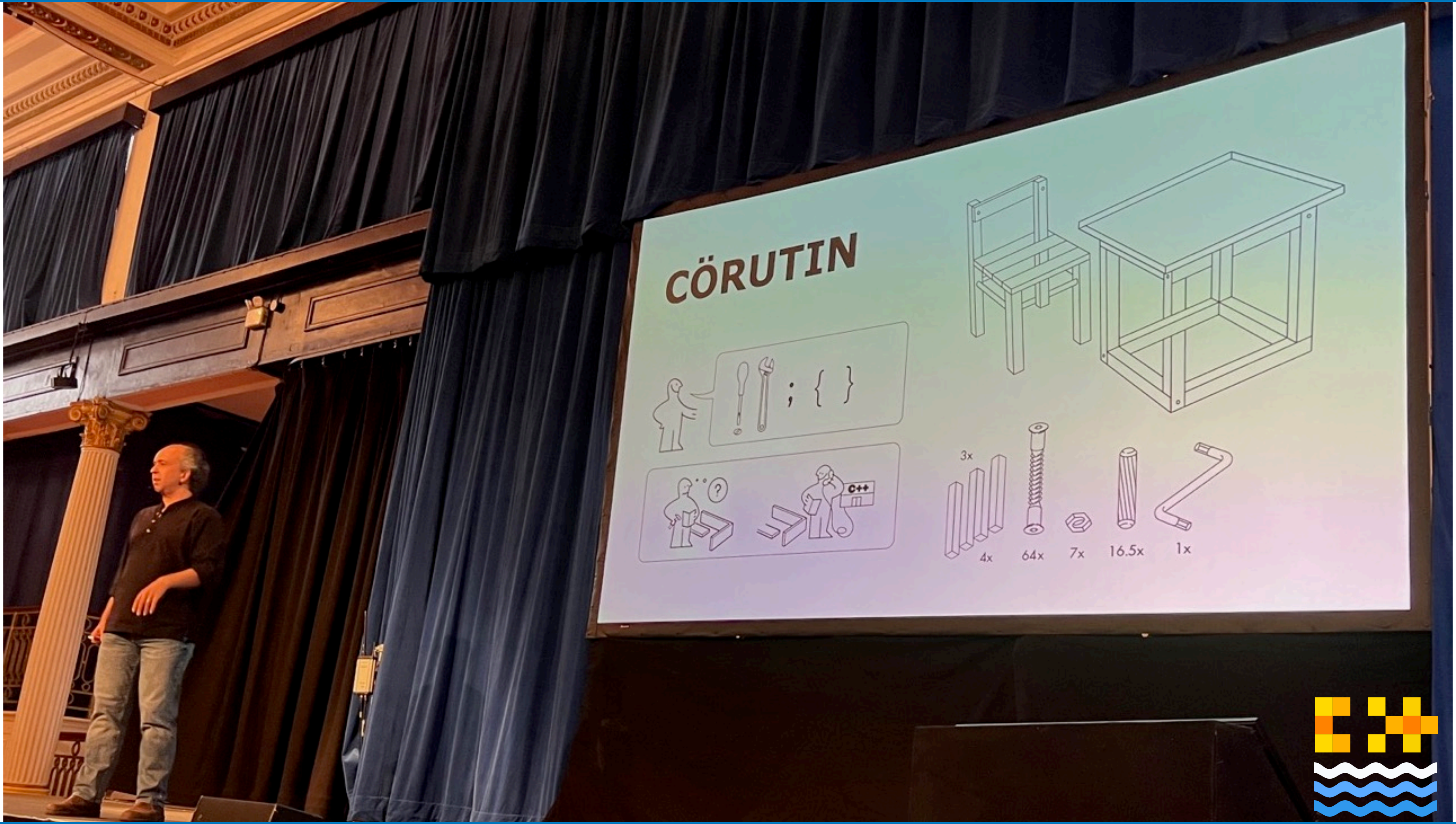
Ólafur Waage
@olafurw

I think 3 years of conference talks and blog posts trying to explain the "basic use case" of a new C++ feature is a hint that the feature isn't designed well

The committee really loves adding configuration options but seem to forget hello world case

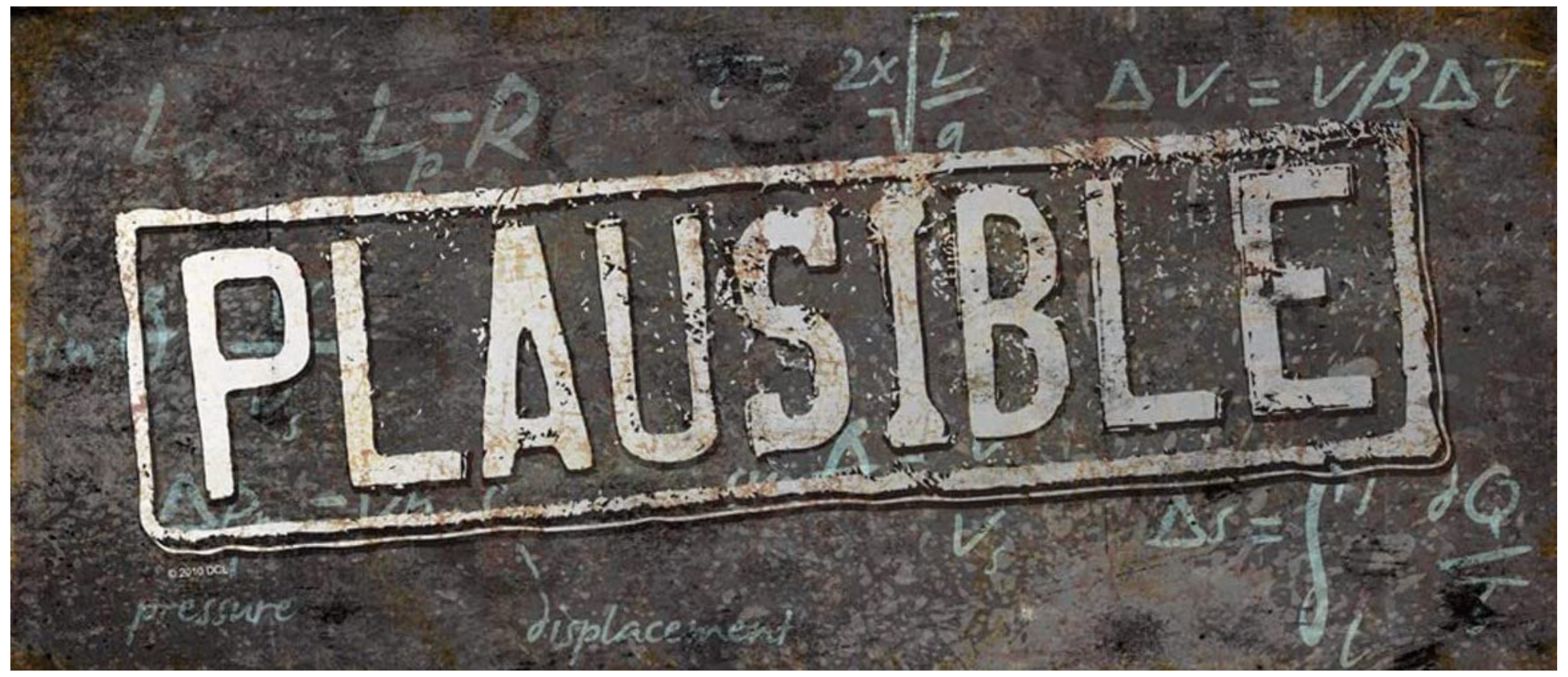
No I will not submit a paper

CÖRUTIN



Test Myth

coroutines shipped in C++20



Kinda... 😔

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

```
#include <generator>
```

Verdict

I think you got how it works

Mythbusting Series

<Part 2 of N>

C++ Myths

Let's dig in!

Humans Depend on Tools

Myth #14

C++ is not easily toolable 🛠️

Tools 🧰

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

CI/CD service

package manager

dynamic analyzer (runtime)

code reviews platform

SCM client

+ fuzzing

Programmers Depend on Tools

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

youtube.com/watch?v=q7Gv4J3FyYE

github.com/lefticus/cpp_weekly/issues/175

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 mi
    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://learn.microsoft.com/en-us/cpp/standard-library/checked-ite
  - drmemory
- 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

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

Myth #14

C++ is not easily toolable 🛠️

Get to know your tools well

Myth #10

C++ is slow to compile

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

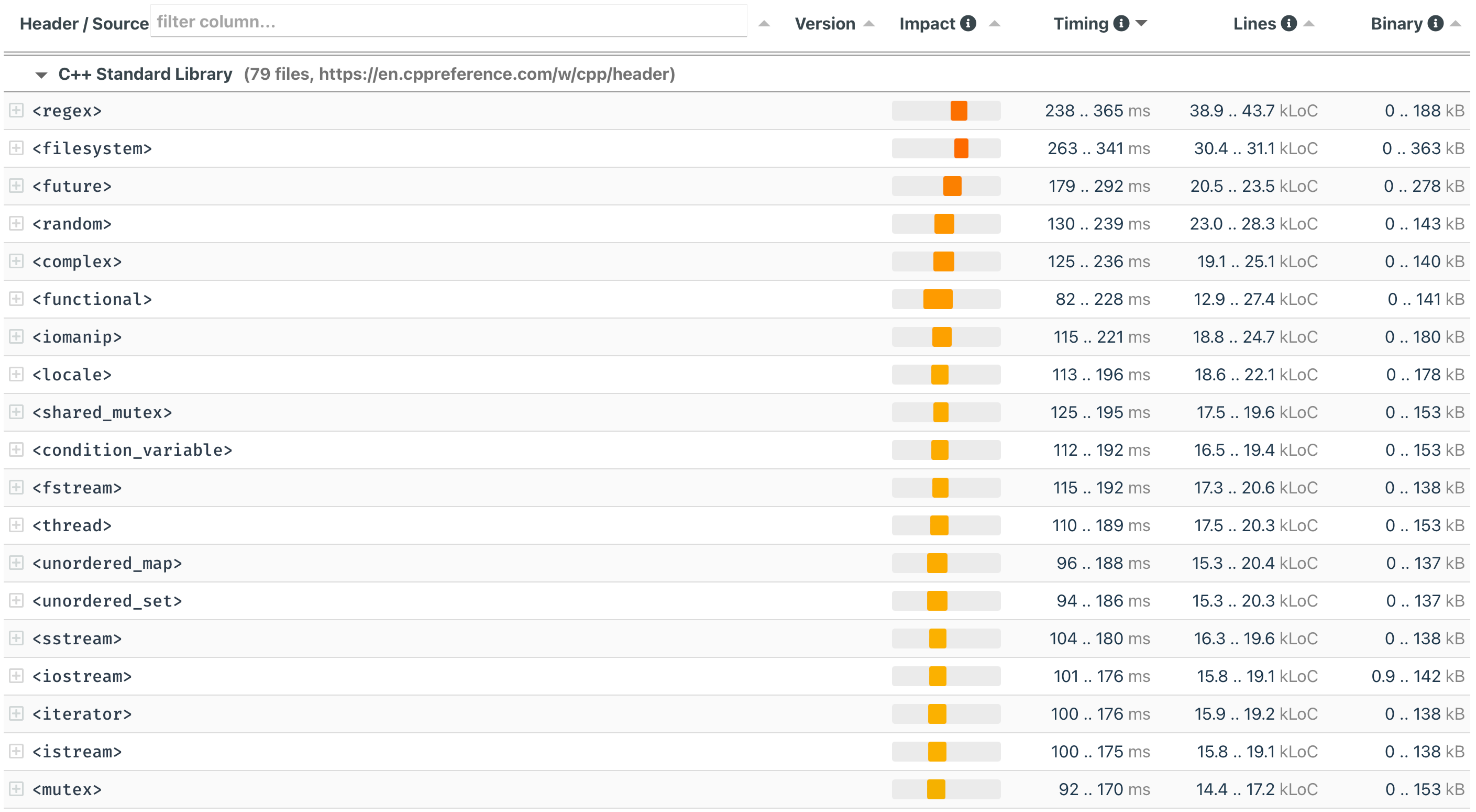
So, you think you know why your builds take so long... you'd be surprised.

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

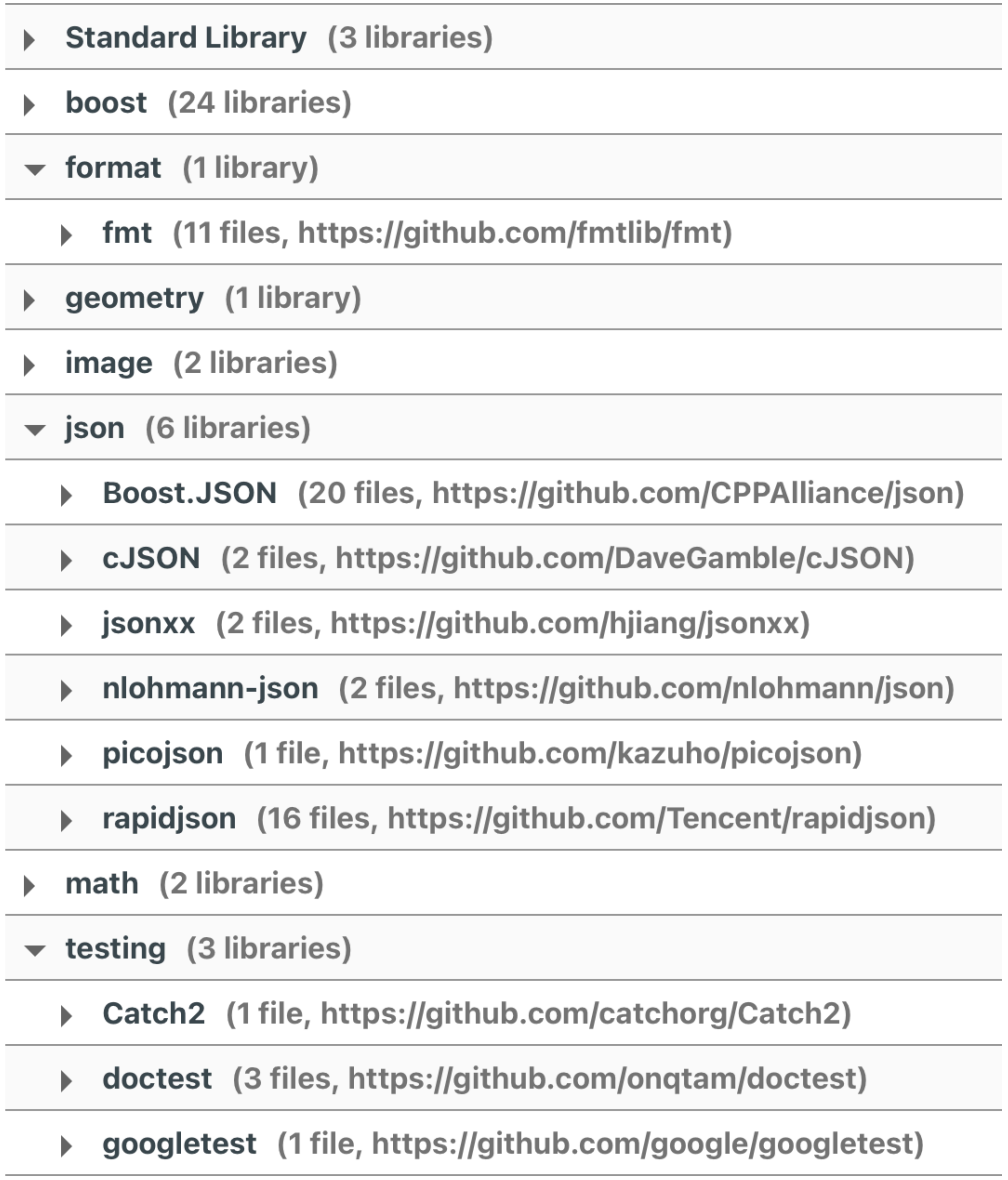
Myth #10



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

artificial-mind.net/projects/compile-health/

Myth #10

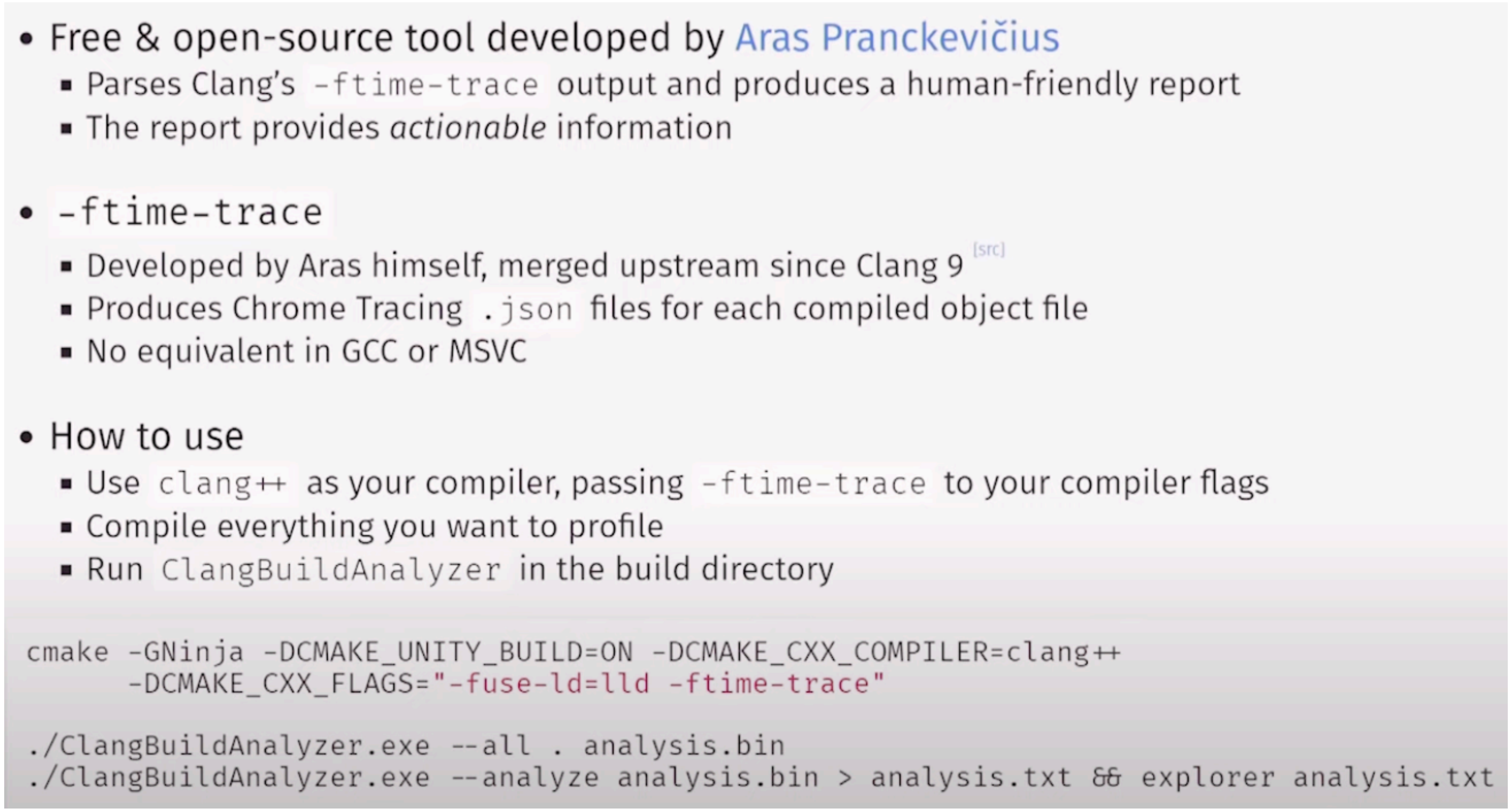


- ▸ **Standard Library** (3 libraries)
- ▸ **boost** (24 libraries)
- ▾ **format** (1 library)
  - ▸ **fmt** (11 files, https://github.com/fmtlib/fmt)
- ▸ **geometry** (1 library)
- ▸ **image** (2 libraries)
- ▾ **json** (6 libraries)
  - ▸ **Boost.JSON** (20 files, https://github.com/CPPAlliance/json)
  - ▸ **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/

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 [src]
  - 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

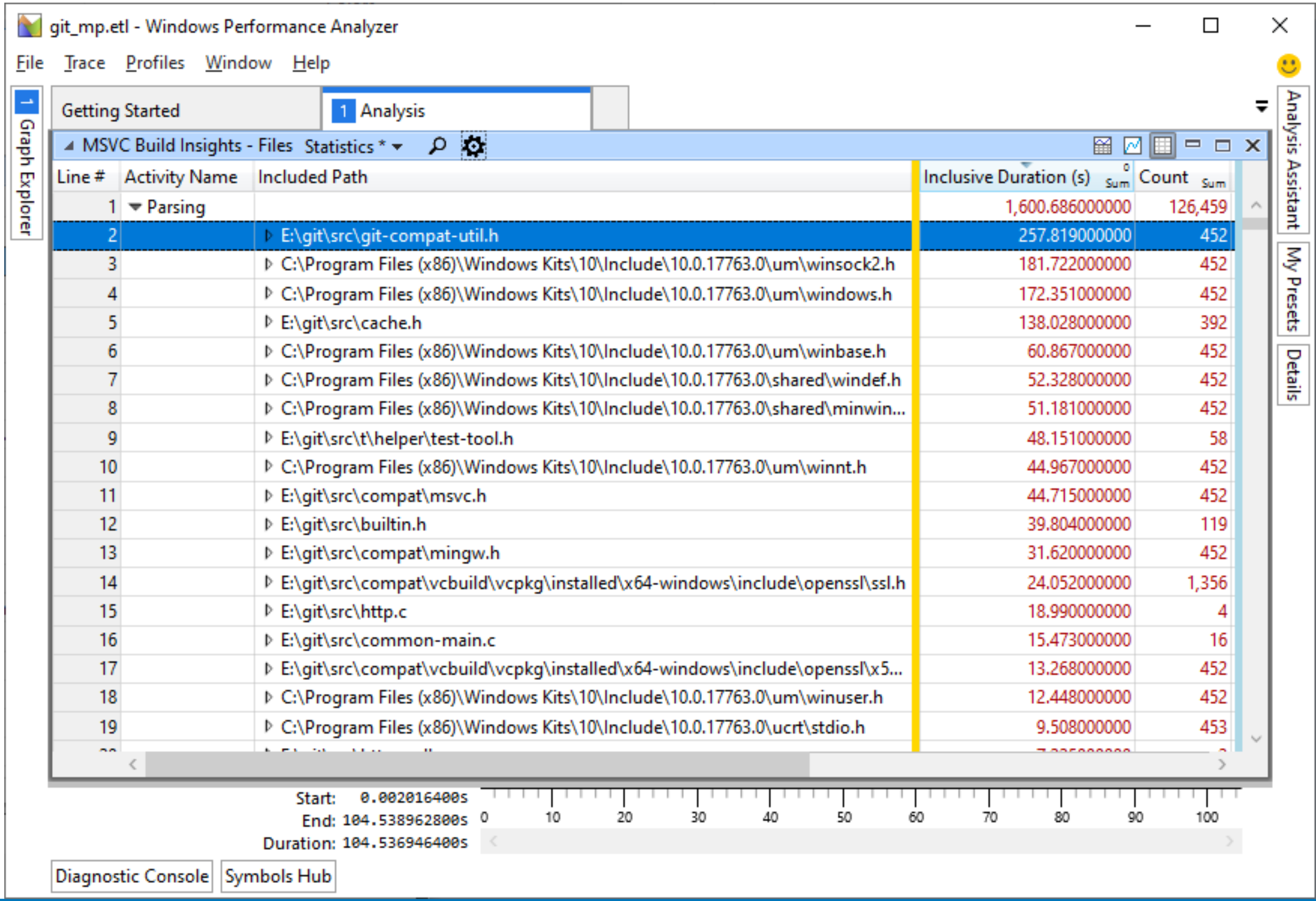
```
cmake -GNinja -DCMAKE_UNITY_BUILD=ON -DCMAKE_CXX_COMPILER=clang++
      -DCMAKE_CXX_FLAGS="-fuse-ld=lld -ftime-trace"

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

Myth #10

devblogs.microsoft.com/cppblog/introducing-c-build-insights/

Tooling can help: vcperf + WPA

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



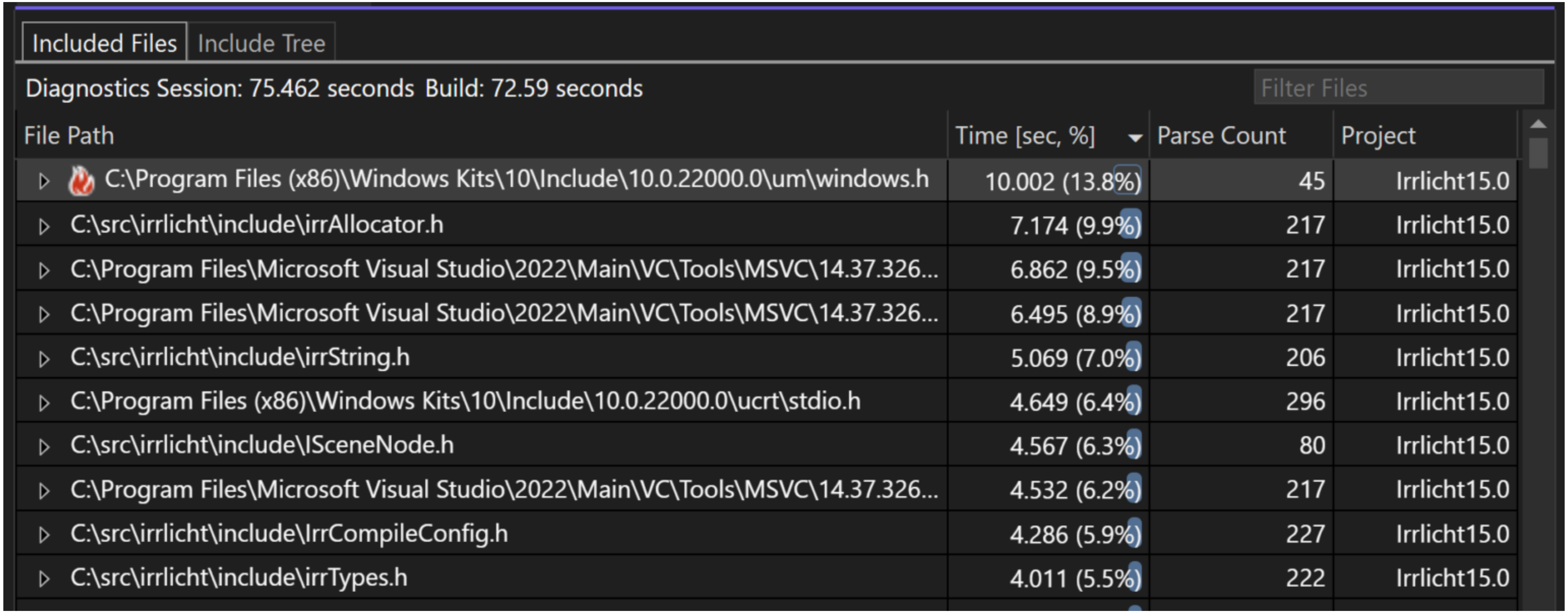
git_mp.etl - Windows Performance Analyzer
File Trace Profiles Window Help
Getting Started
1 Analysis
MSVC Build Insights - Files Statistics *
Graph Explorer
Analysis Assistant
My Presets
Details
Diagnostic Console
Symbols Hub

Line #	Activity Name	Included Path	Inclusive Duration (s) Sum	Count Sum
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



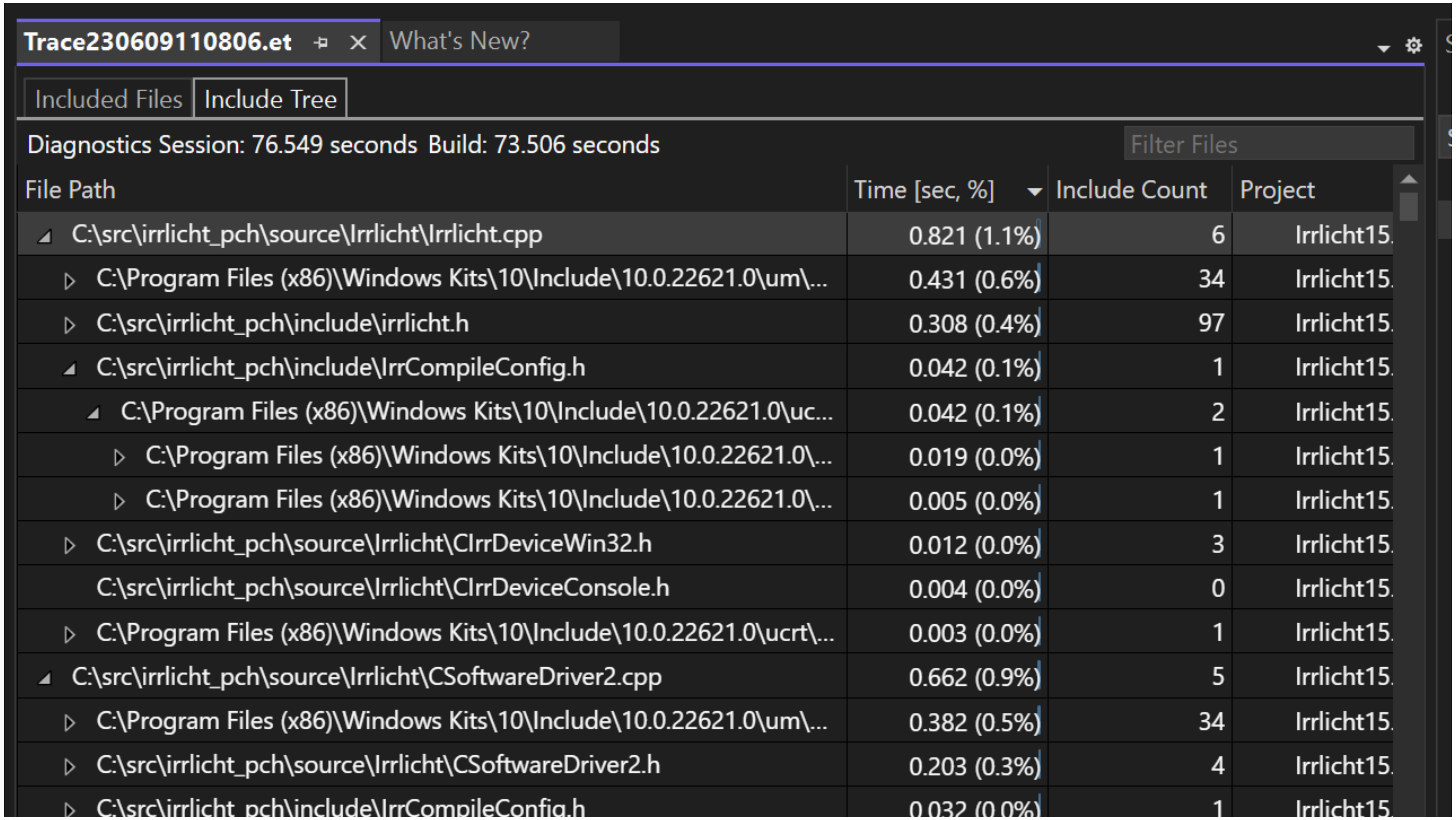
Diagnostics Session: 75.462 seconds Build: 72.59 seconds

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/

Myth #10

Tooling can help: Build Insights in Visual Studio



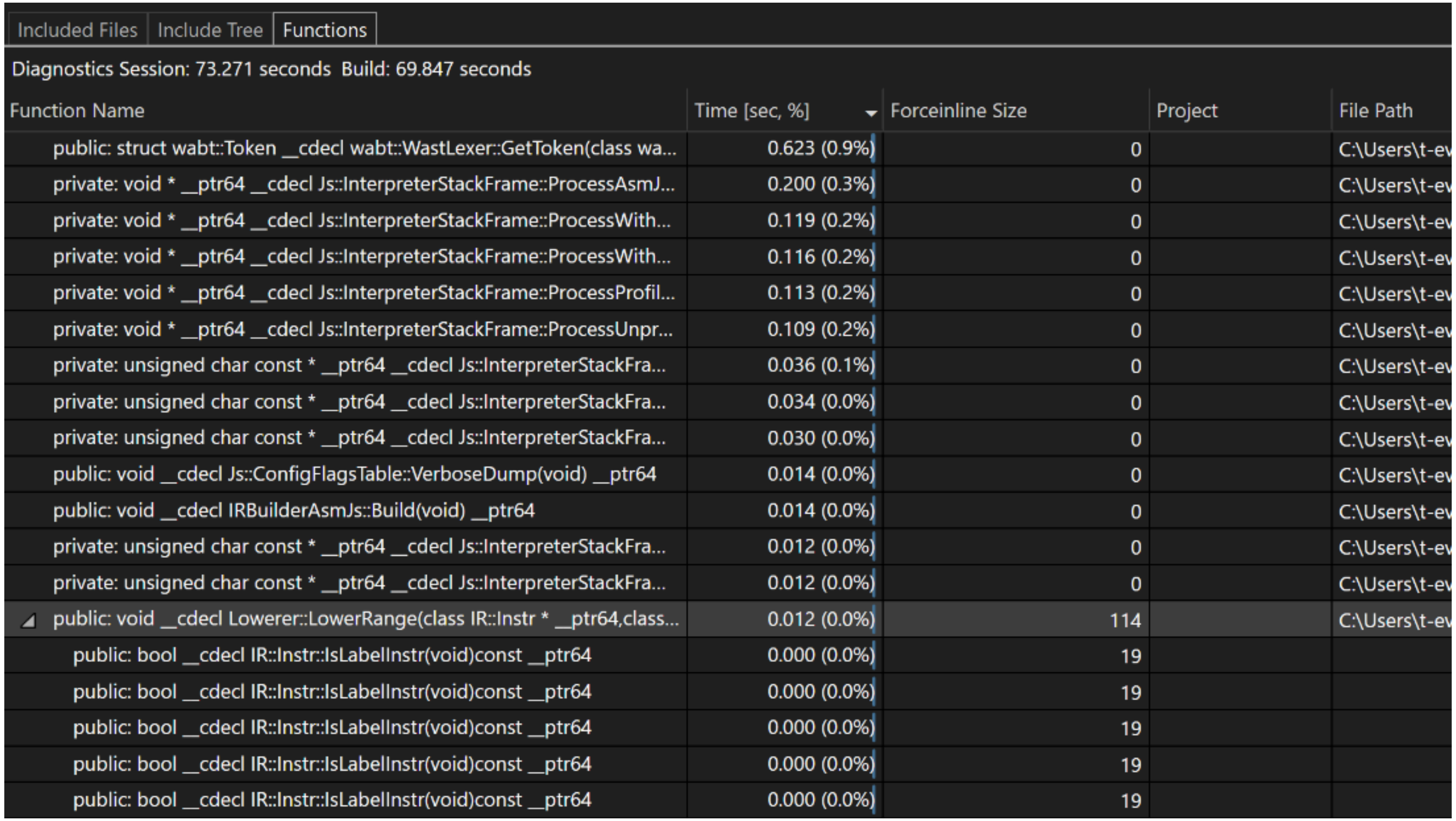
Diagnostics Session: 76.549 seconds Build: 73.506 seconds

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/

Myth #10

Tooling can help: Build Insights in Visual Studio



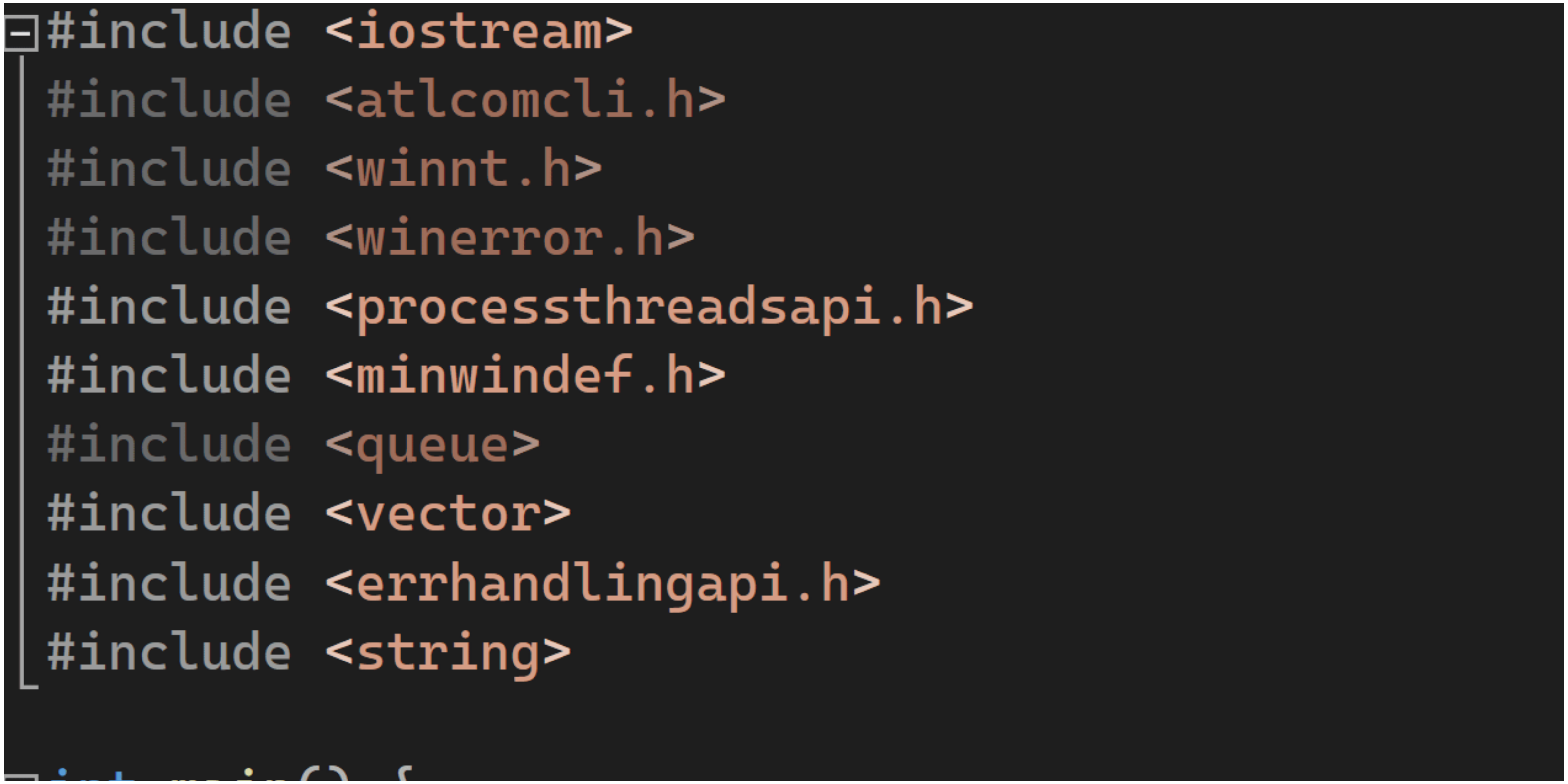
Diagnostics Session: 73.271 seconds Build: 69.847 seconds

Function Name	Time [sec, %]	Forceinline Size	Project	File Path
public: struct wabt::Token __cdecl wabt::WastLexer::GetToken(class wa...	0.623 (0.9%)	0		C:\Users\t-ev
private: void * __ptr64 __cdecl Js::InterpreterStackFrame::ProcessAsmJ...	0.200 (0.3%)	0		C:\Users\t-ev
private: void * __ptr64 __cdecl Js::InterpreterStackFrame::ProcessWith...	0.119 (0.2%)	0		C:\Users\t-ev
private: void * __ptr64 __cdecl Js::InterpreterStackFrame::ProcessWith...	0.116 (0.2%)	0		C:\Users\t-ev
private: void * __ptr64 __cdecl Js::InterpreterStackFrame::ProcessProfil...	0.113 (0.2%)	0		C:\Users\t-ev
private: void * __ptr64 __cdecl Js::InterpreterStackFrame::ProcessUnpr...	0.109 (0.2%)	0		C:\Users\t-ev
private: unsigned char const * __ptr64 __cdecl Js::InterpreterStackFra...	0.036 (0.1%)	0		C:\Users\t-ev
private: unsigned char const * __ptr64 __cdecl Js::InterpreterStackFra...	0.034 (0.0%)	0		C:\Users\t-ev
private: unsigned char const * __ptr64 __cdecl Js::InterpreterStackFra...	0.030 (0.0%)	0		C:\Users\t-ev
public: void __cdecl Js::ConfigFlagsTable::VerboseDump(void) __ptr64	0.014 (0.0%)	0		C:\Users\t-ev
public: void __cdecl IRBuilderAsmJs::Build(void) __ptr64	0.014 (0.0%)	0		C:\Users\t-ev
private: unsigned char const * __ptr64 __cdecl Js::InterpreterStackFra...	0.012 (0.0%)	0		C:\Users\t-ev
private: unsigned char const * __ptr64 __cdecl Js::InterpreterStackFra...	0.012 (0.0%)	0		C:\Users\t-ev
◢ public: void __cdecl Lowerer::LowerRange(class IR::Instr * __ptr64,class...	0.012 (0.0%)	114		C:\Users\t-ev
public: bool __cdecl IR::Instr::IsLabelInstr(void)const __ptr64	0.000 (0.0%)	19		
public: bool __cdecl IR::Instr::IsLabelInstr(void)const __ptr64	0.000 (0.0%)	19		
public: bool __cdecl IR::Instr::IsLabelInstr(void)const __ptr64	0.000 (0.0%)	19		
public: bool __cdecl IR::Instr::IsLabelInstr(void)const __ptr64	0.000 (0.0%)	19		
public: bool __cdecl IR::Instr::IsLabelInstr(void)const __ptr64	0.000 (0.0%)	19		

[Functions View] - how long a function takes during compilation, as well as the number of `forceinline`

Myth #10

🧰 `#include cleanup`

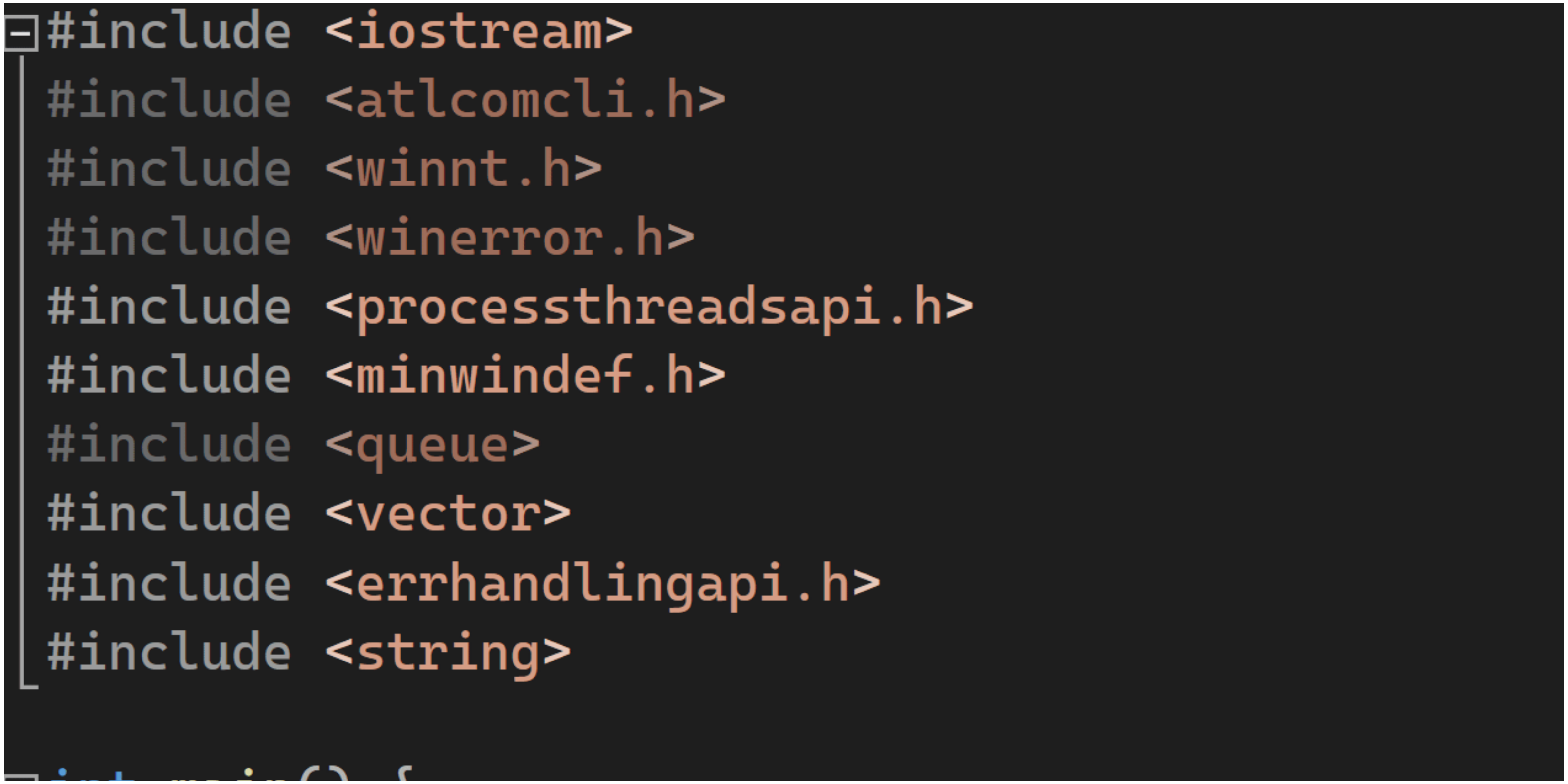


```
#include <iostream>
#include <atlcomcli.h>
#include <winnt.h>
#include <winerror.h>
#include <processthreadsapi.h>
#include <minwindef.h>
#include <queue>
#include <vector>
#include <errhandlingapi.h>
#include <string>
```

devblogs.microsoft.com/cppblog/include-cleanup-in-visual-studio/

Myth #10

🧰 `#include cleanup`



```
#include <iostream>
#include <atlcomcli.h>
#include <winnt.h>
#include <winerror.h>
#include <processthreadsapi.h>
#include <minwindef.h>
#include <queue>
#include <vector>
#include <errhandlingapi.h>
#include <string>
```

devblogs.microsoft.com/cppblog/include-cleanup-in-visual-studio/

Myth #10

IMPROVING COMPILATION TIMES
TOOLS & TECHNIQUES

accu 2023

IMPROVING COMPILATION TIMES
Tools & Techniques

ACCU 2023
April 20 2023

Vittorio Romeo
mail@vittorioromeo.com
@supahvee1234

Bloomberg Engineering
TechAtBloomberg.com
Careers

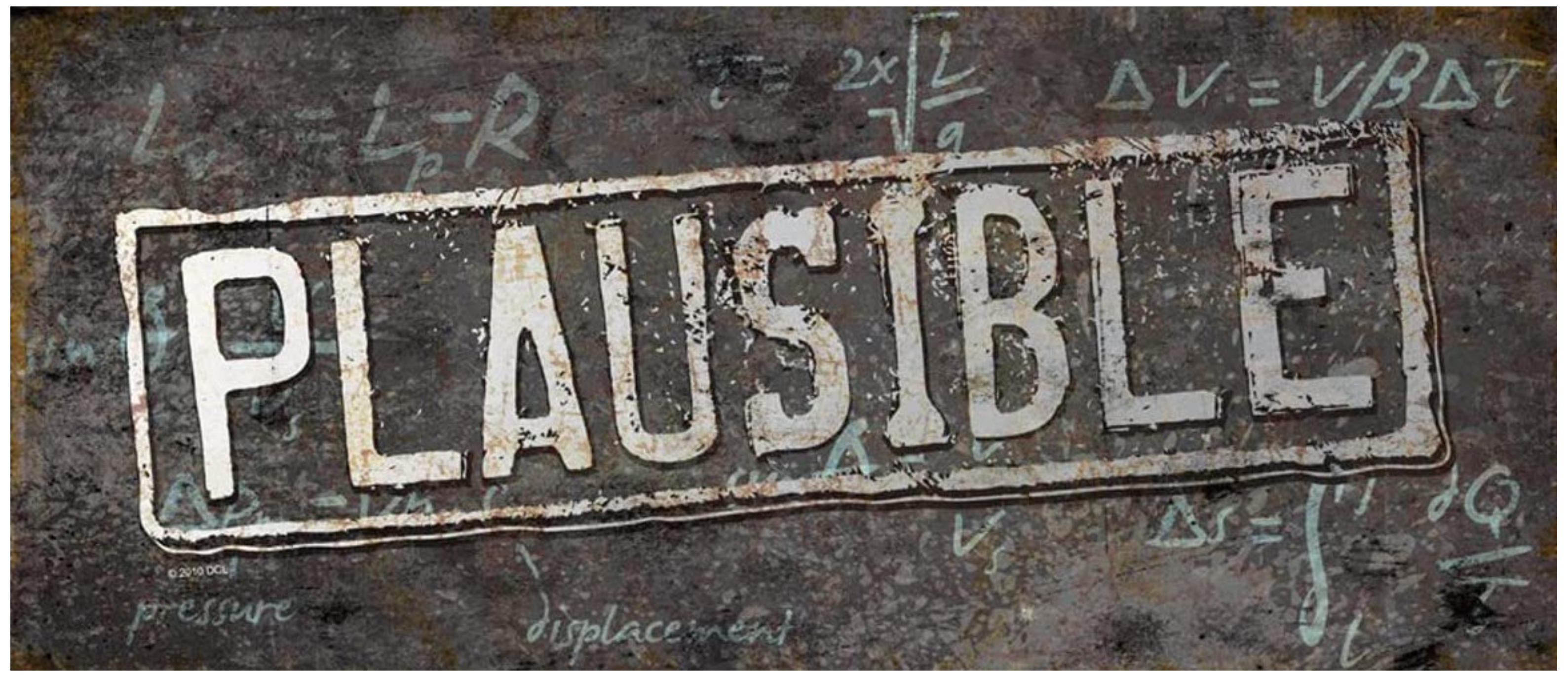
Vittorio Romeo

ACCU.ORG

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)

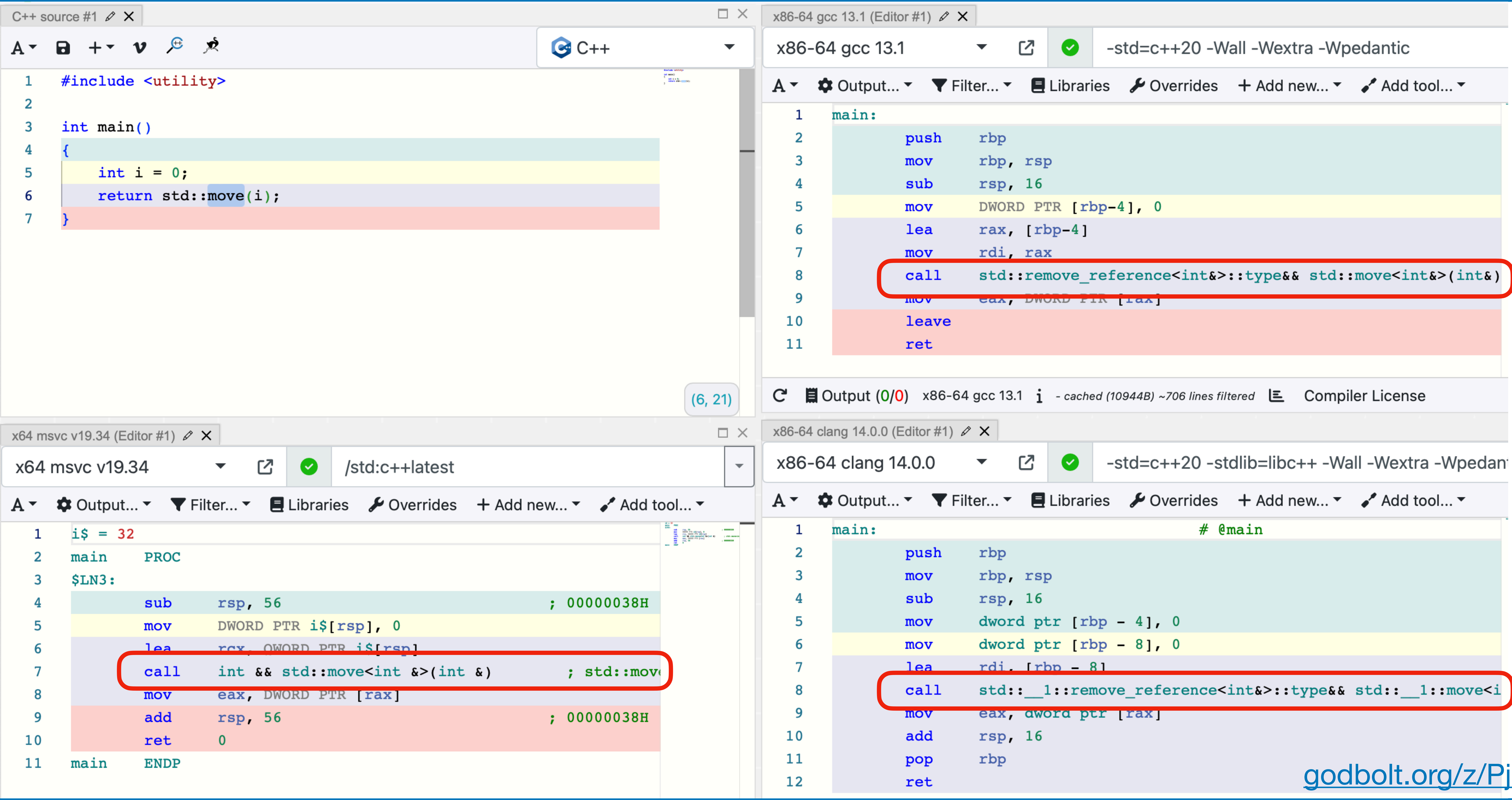
eg.

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

➡️ `static_cast<int&&>(i);`

vittorioromeo.info/index/blog/debug_performance_cpp.html

Myth #12



C++ source #1

C++

```cpp
#include <utility>

int main()
{
    int i = 0;
    return std::move(i);
}
```

(6, 21)

x86-64 gcc 13.1 (Editor #1)

x86-64 gcc 13.1 -std=c++20 -Wall -Wextra -Wpedantic

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 0
        lea     rax, [rbp-4]
        mov     rdi, rax
        call    std::remove_reference<int&>::type&& std::move<int&>(int&)
        mov     eax, DWORD PTR [rax]
        leave
        ret
```

Output (0/0) x86-64 gcc 13.1 - cached (10944B) ~706 lines filtered Compiler License

x64 msvc v19.34 (Editor #1)

x64 msvc v19.34 /std:c++latest

```
i$ = 32
main    PROC
$LN3:
        sub     rsp, 56                                 ; 00000038H
        mov     DWORD PTR i$[rsp], 0
        lea     rcx, QWORD PTR i$[rsp]
        call    int && std::move<int &>(int &)          ; std::move
        mov     eax, DWORD PTR [rax]
        add     rsp, 56                                 ; 00000038H
        ret     0
main    ENDP
```

x86-64 clang 14.0.0 (Editor #1)

x86-64 clang 14.0.0 -std=c++20 -stdlib=libc++ -Wall -Wextra -Wpedan

```
main:                                   # @main
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 0
        lea     rdi, [rbp - 8]
        call    std::__1::remove_reference<int&>::type&& std::__1::move<i
        mov     eax, dword ptr [rax]
        add     rsp, 16
        pop     rbp
        ret
```

godbolt.org/z/Pj6xahP9j

Myth #12

C++ source #1

```cpp
#include <utility>

int main()
{
    int i = 0;
    return std::move(i);
}
```

x86-64 gcc (trunk) — -std=c++20 -Wall -Wextra -Wpedantic

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 0
        lea     rax, [rbp-4]
        mov     rdi, rax
        call    std::remove_reference<int&>::type&& std::move<int&>(int&)
        mov     eax, DWORD PTR [rax]
        leave
        ret
```

Output (0/0) x86-64 gcc (trunk) - cached (11069B) ~706 lines filtered Compiler License

x64 msvc v19.35 — /std:c++latest

```
i$ = 0
main    PROC
$LN3:
        sub     rsp, 24
        mov     DWORD PTR i$[rsp], 0
        mov     eax, DWORD PTR i$[rsp]
        add     rsp, 24
        ret     0
main    ENDP
```

x86-64 clang 16.0.0 — -std=c++20 -stdlib=libc++ -Wall -Wextra -Wpedan

```
main:                                   # @main
        push    rbp
        mov     rbp, rsp
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 0
        mov     eax, dword ptr [rbp - 8]
        pop     rbp
        ret
```

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]]`

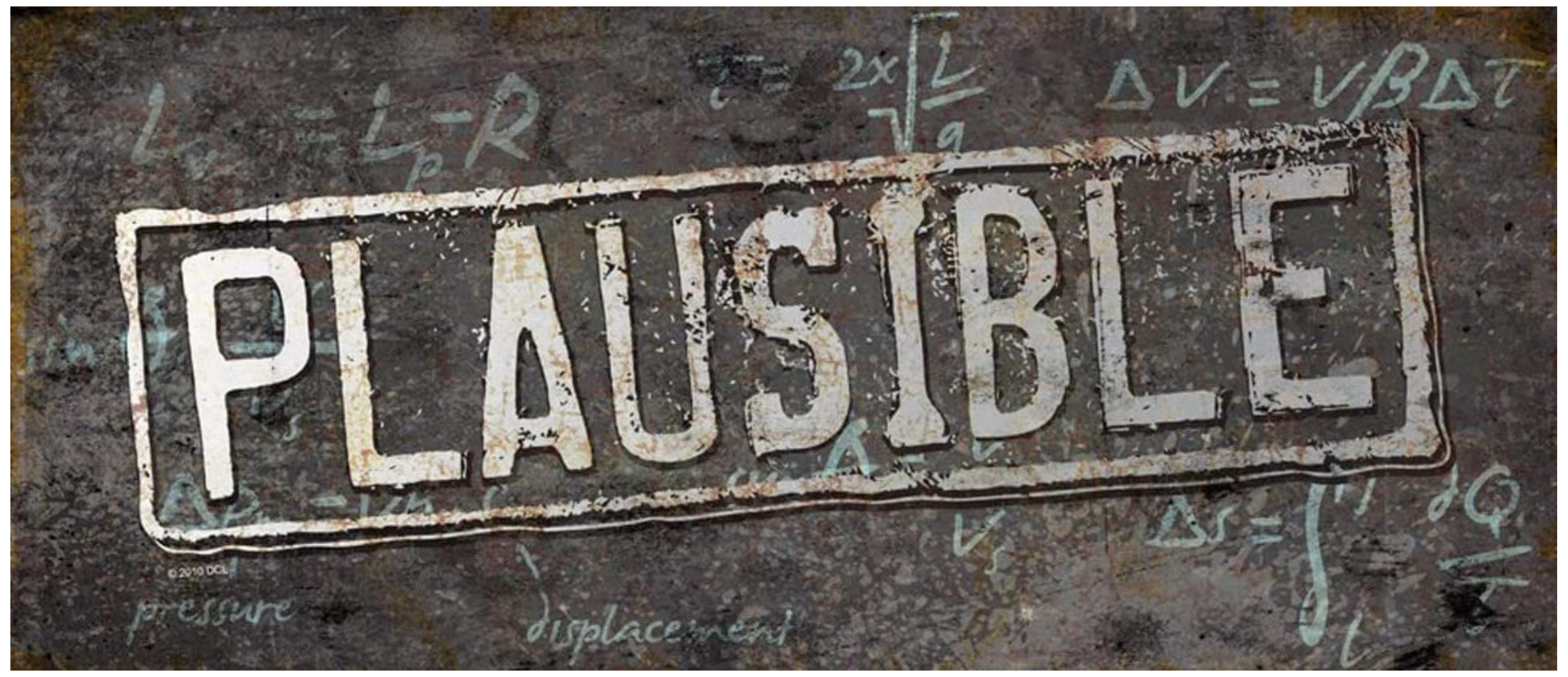
=> *extensible* to your own such utility functions

youtu.be/idwVQUG6Jqc

devblogs.microsoft.com/cppblog/improving-the-state-of-debug-performance-in-c/

Myth #12

The sad state of Debug performance in C++



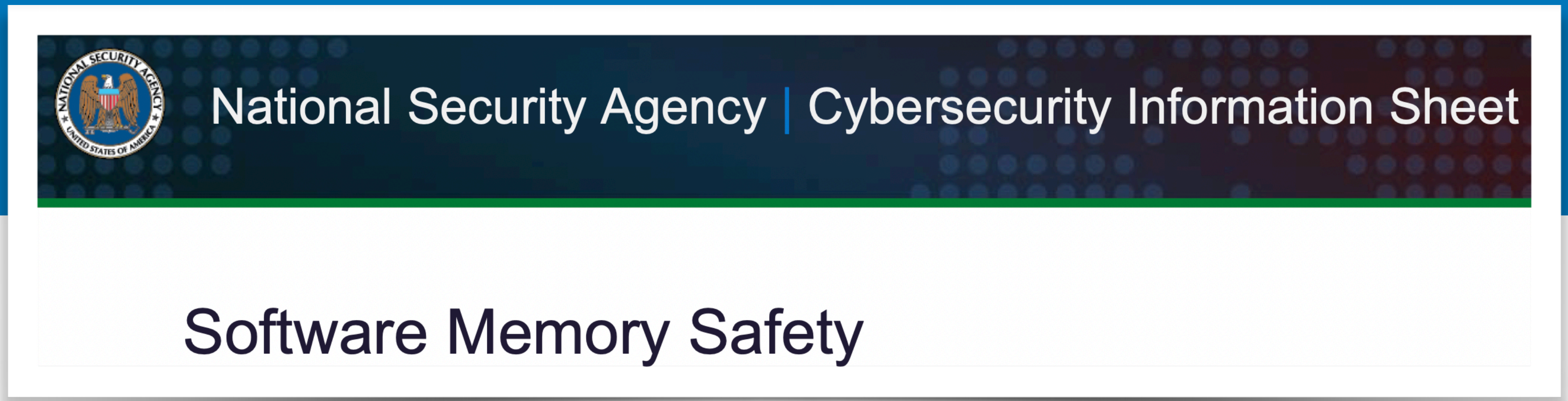
Myth #23

C++ will never be a safe language

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

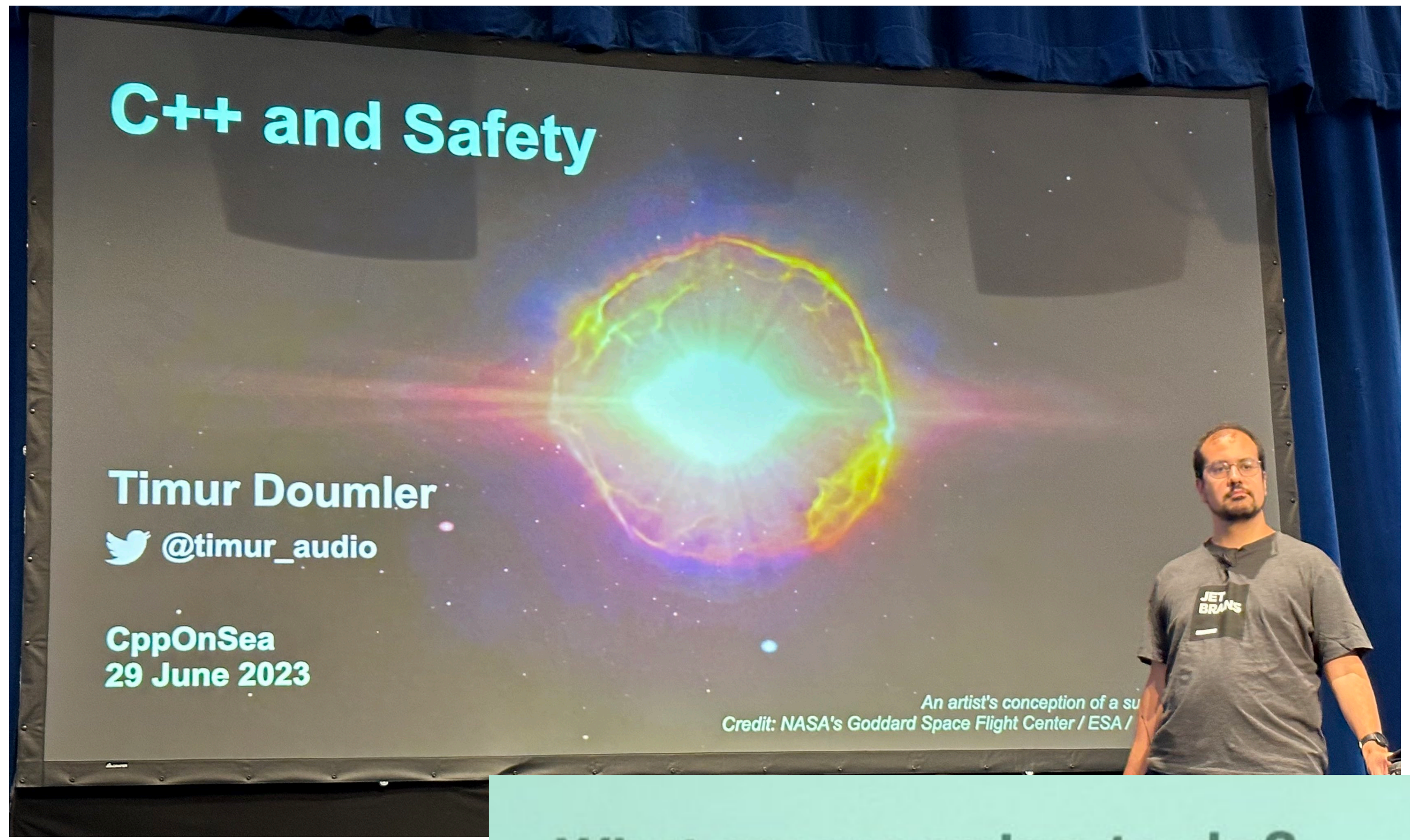
Myth #23

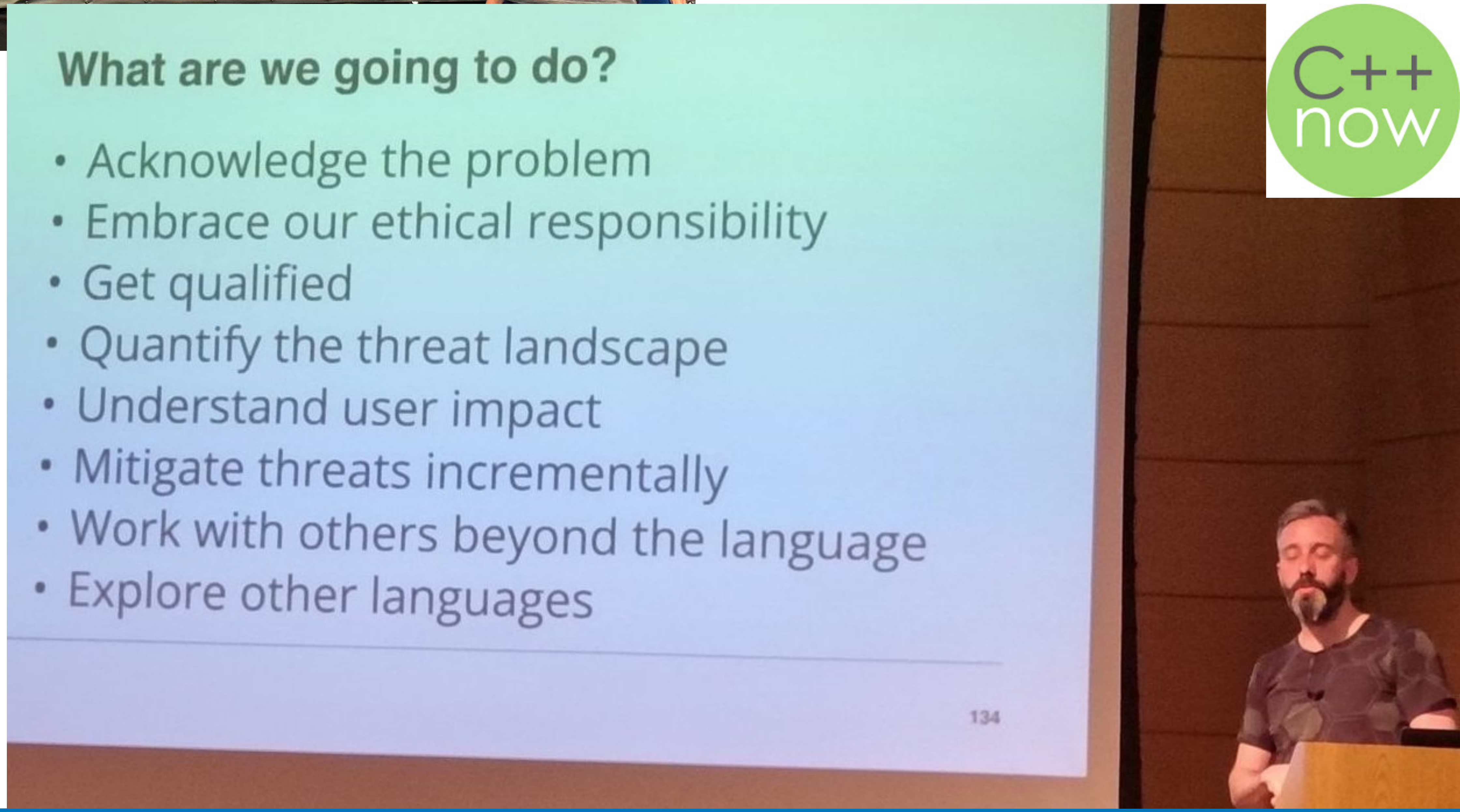
C++ is under attack... and the community is responding 🤷‍♂️


National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

defense.gov/2022/Nov/CSI_SOFTWARE_MEMORY_SAFETY.PDF




What are we going to do?

- Acknowledge the problem
- Embrace our ethical responsibility
- Get qualified
- Quantify the threat landscape
- Understand user impact
- Mitigate threats incrementally
- Work with others beyond the language
- Explore other languages

Myth #23

Tradeoffs need to be made...

"To UB, or not to UB"

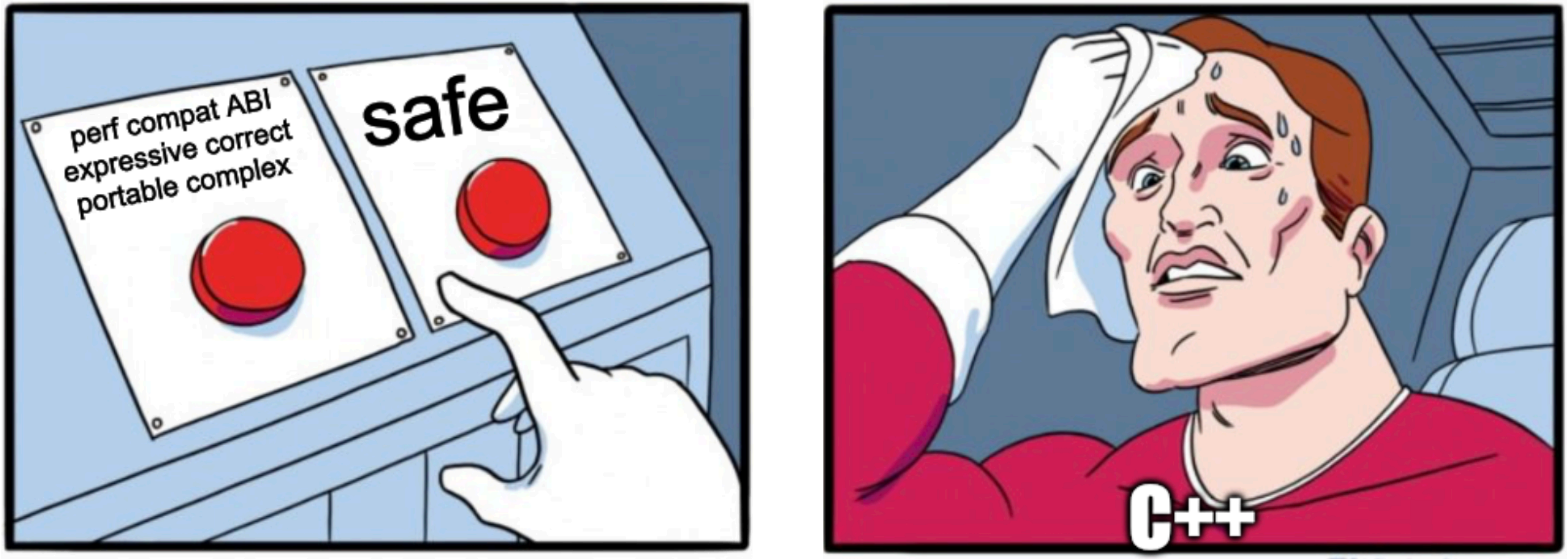
-- Prince Hamlet

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


perf compat ABI
expressive correct
portable complex
safe
C++

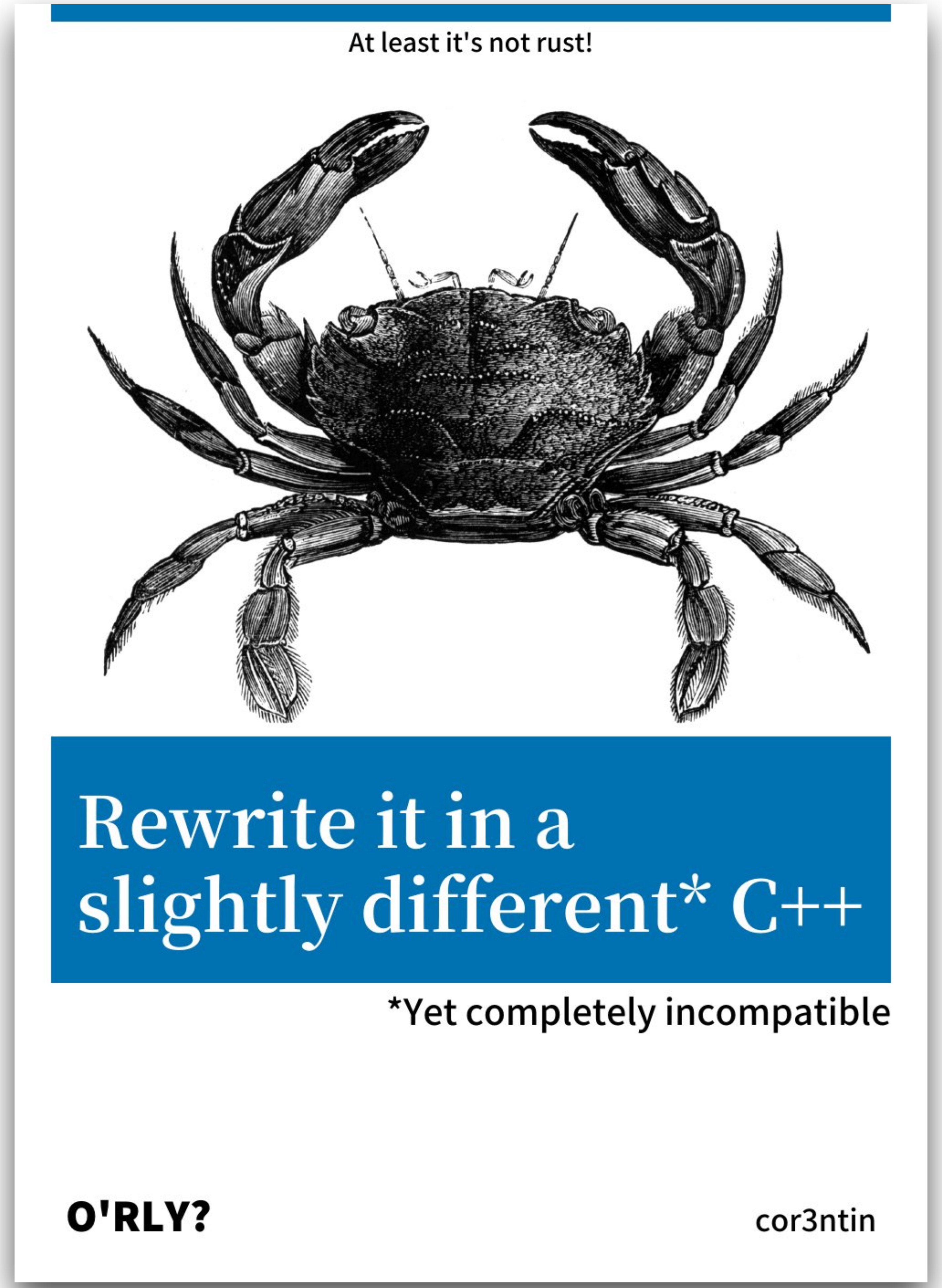
Myth #23

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

-- Corentin Jabot

cor3ntin.github.io/posts/safety/

- 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


At least it's not rust!
Rewrite it in a slightly different* C++
*Yet completely incompatible
O'RLY?
cor3ntin

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~~ Hylo)
    - no direct mutation (Haskell & other pure functional languages)
  - by tooling (static lifetime analysis):
    - clang-tidy
    - MSVC
    - other commercial analyzers (plenty of them)

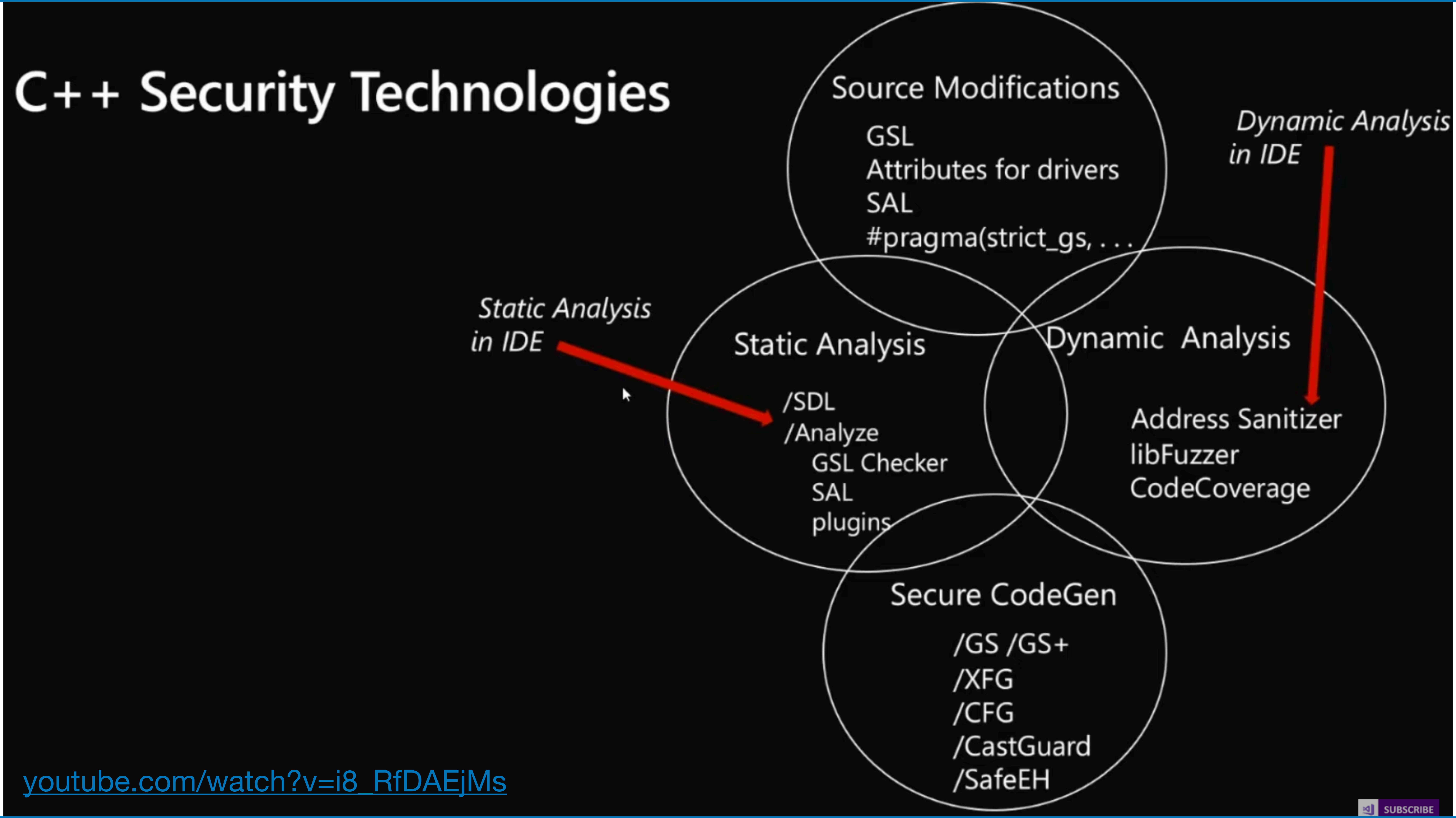
Myth #23

The new C++ "AAA"

~~AAA (almost always auto)~~

AAA (almost always analyze)

Myth #23



C++ Security Technologies

youtube.com/watch?v=i8_RfDAEjMs

Myth #23

ASan FTW !!!

`-fsanitizer=address`

{ Clang, gcc, MSVC }

youtube.com/watch?v=yJLyANPHNaA

ASan `continue_on_error`

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

Myth #23

Static Analysis `lifetime annotations for C++`

NEW:

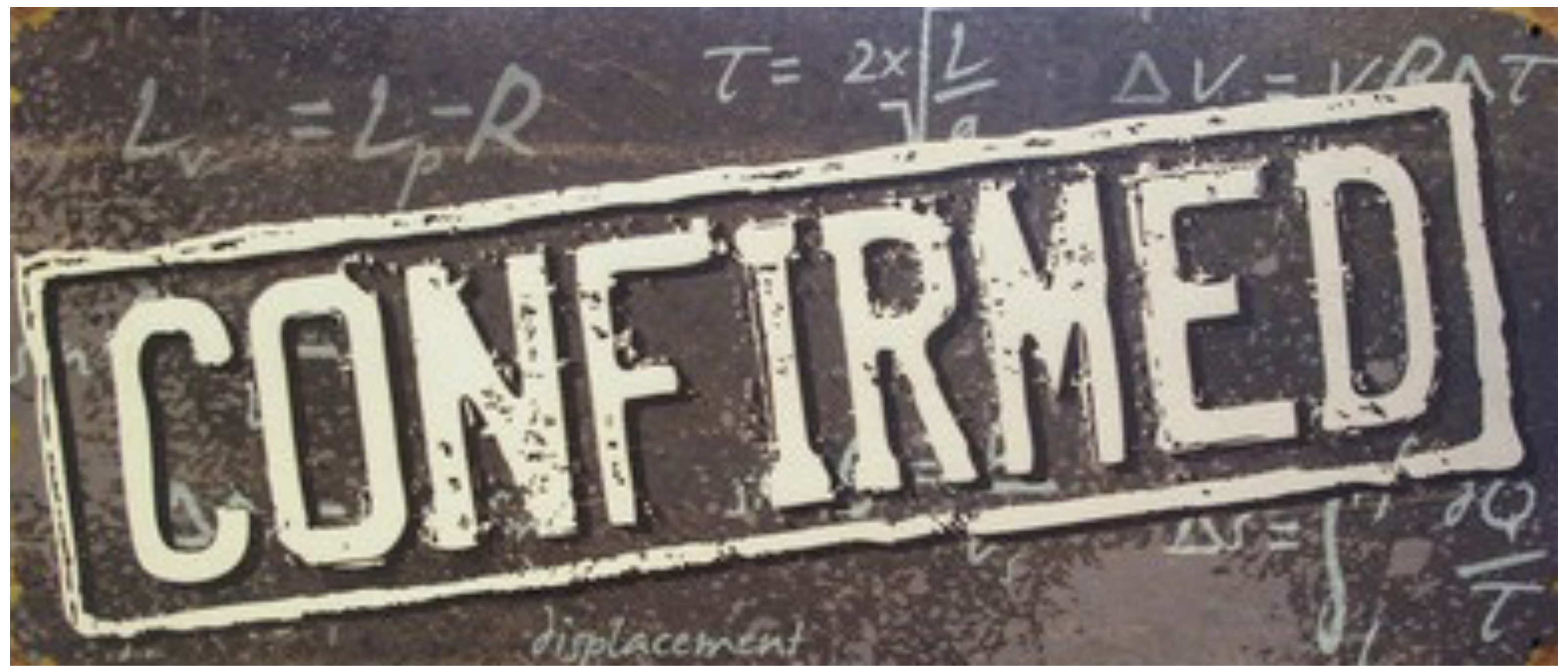
`[[clang::lifetimebound]]` and `[[msvc::lifetimebound]]`

discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377

youtube.com/watch?v=fe6yu9AQIE4

Myth #23

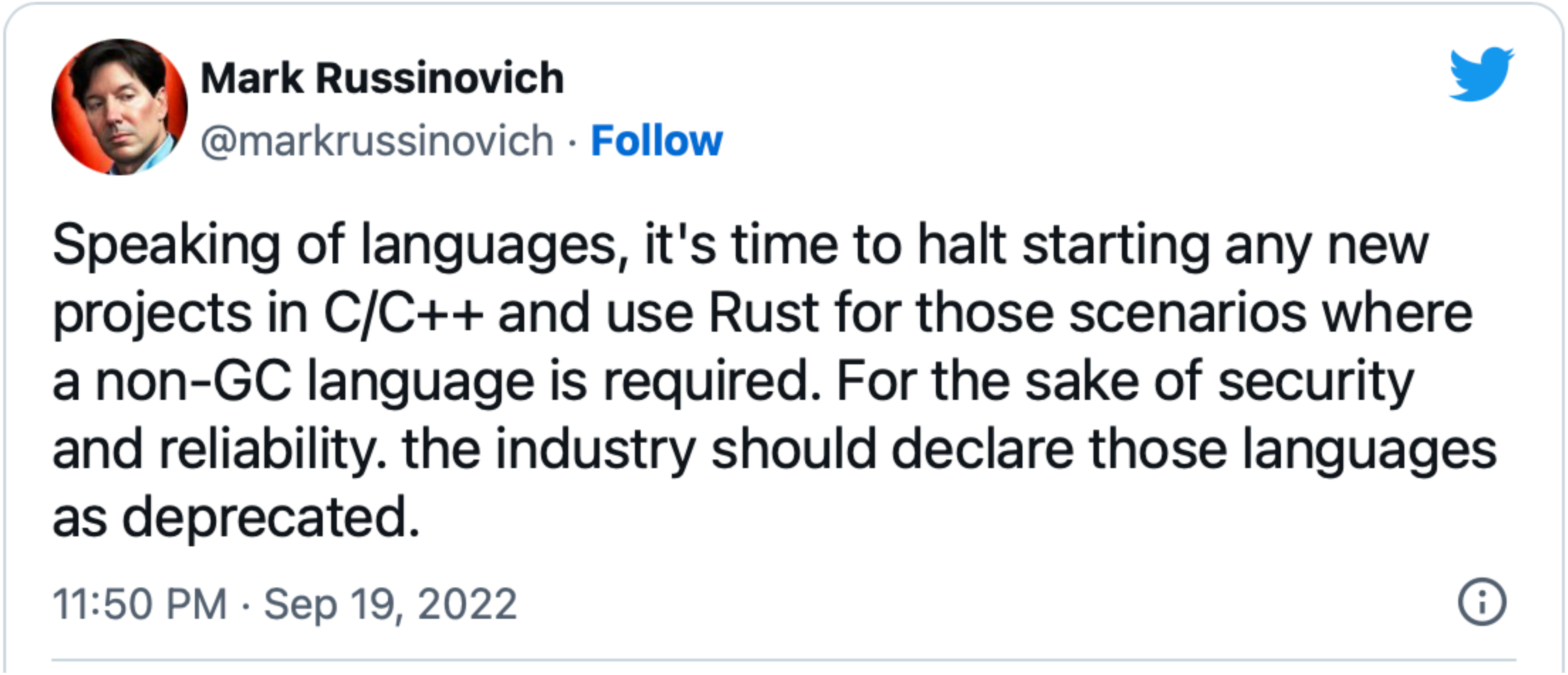
C++ will never be a safe language*



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

Myth #38

Just rewrite it in **Rust** 🦀


Mark Russinovich
@markrussinovich · Follow

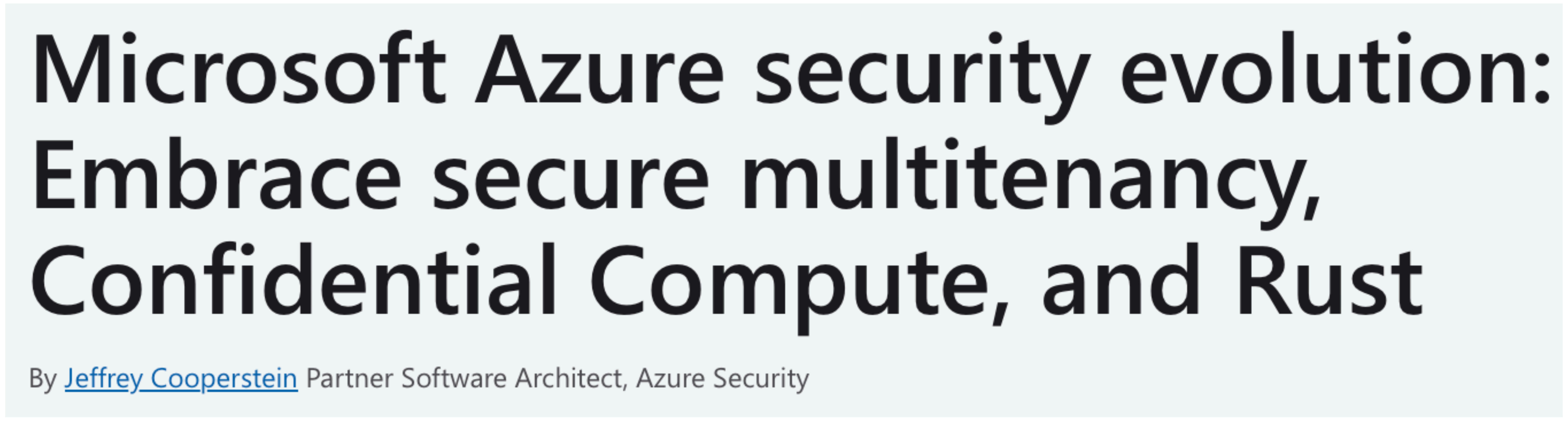
Speaking of languages, it's time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability. the industry should declare those languages as deprecated.

11:50 PM · Sep 19, 2022

Myth #38

🦀 Too soon?

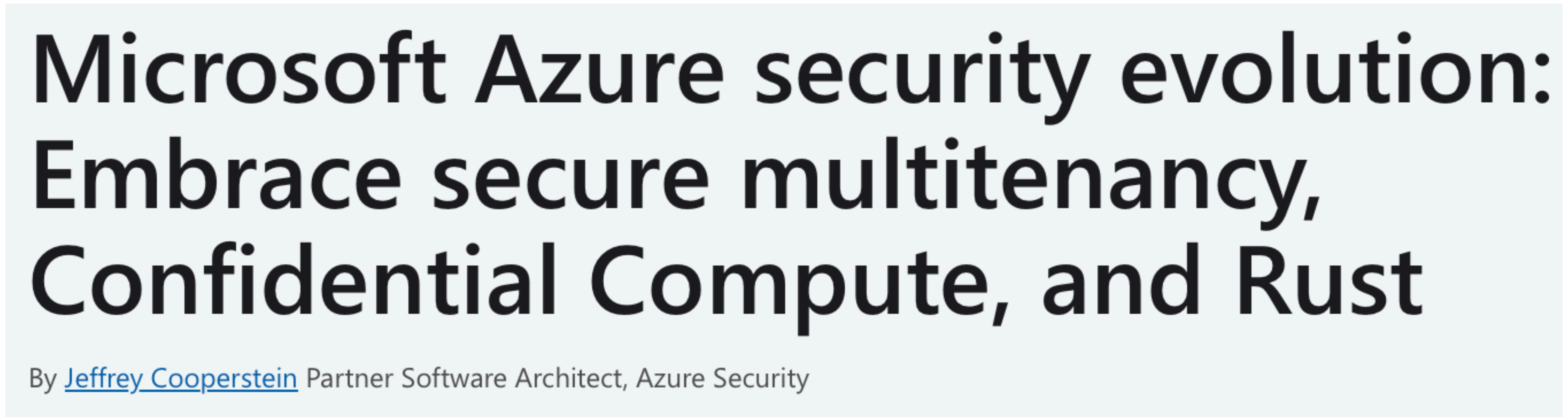
Myth #38


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/

Myth #38


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

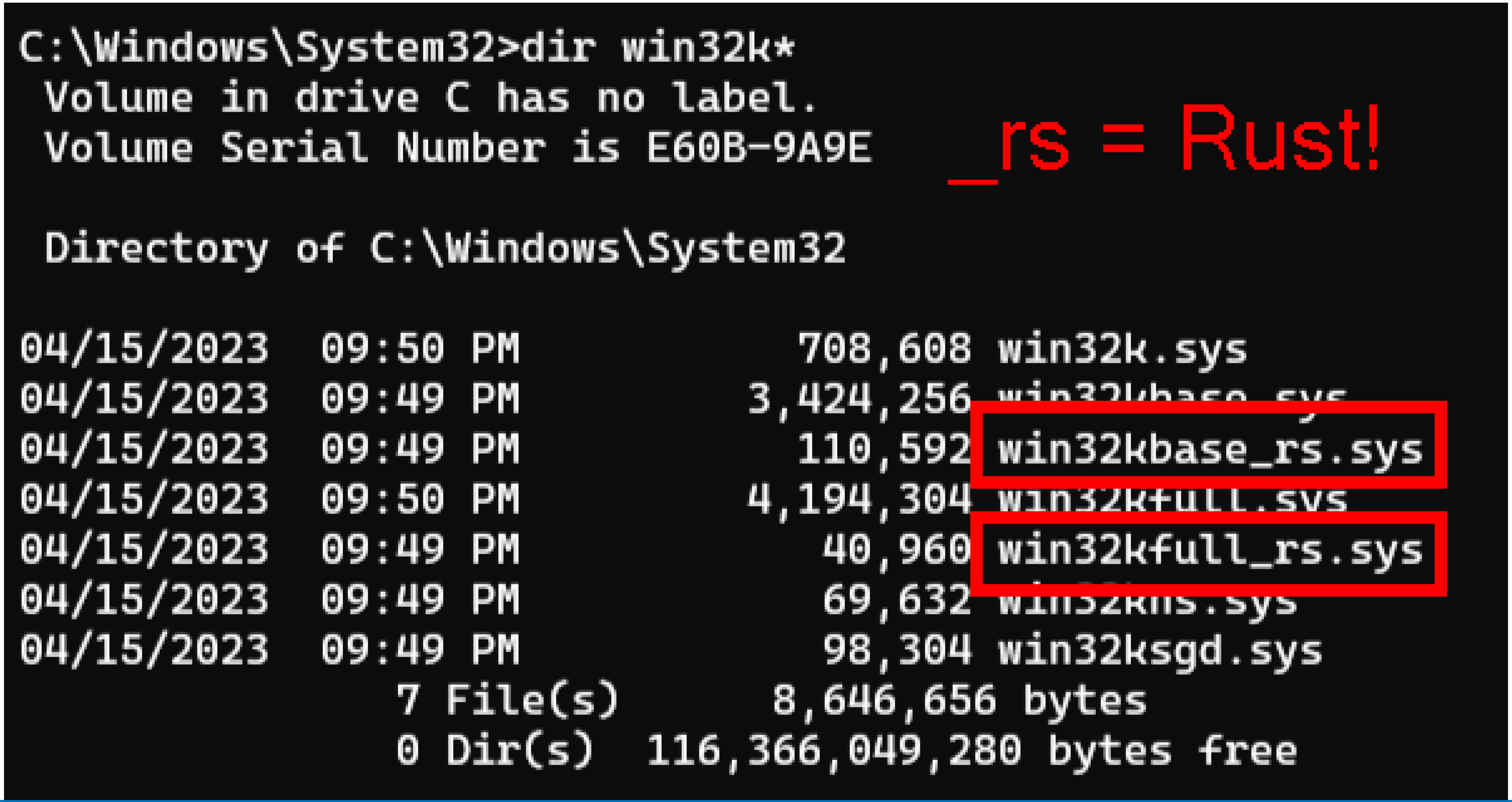
By Jeffrey Cooperstein Partner Software Architect, Azure Security

🔒 **trust**

azure.microsoft.com/blog/microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/

Myth #38

Rust already in the Windows 11 `kernel` (May 2023)

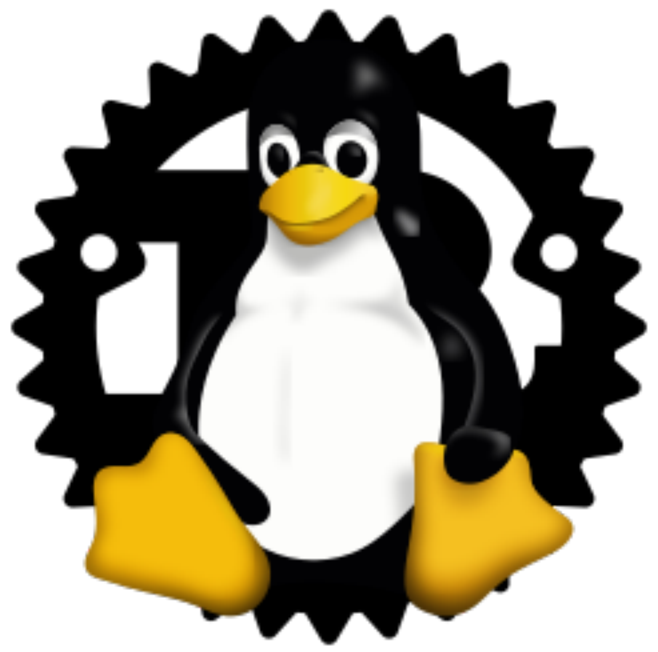


```
C:\Windows\System32>dir win32k*
 Volume in drive C has no label.
 Volume Serial Number is E60B-9A9E

 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 win32kns.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
```

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

Myth #38

youtube.com/watch?v=8T6ClX-y2AE&t=2703s

So this happened 👀 (public announcement, April 2023)

Ported Windows 11 core components from C++ to Rust

- **`DirectWrite`**
- **`GDI`**
- **`...`** 🤫

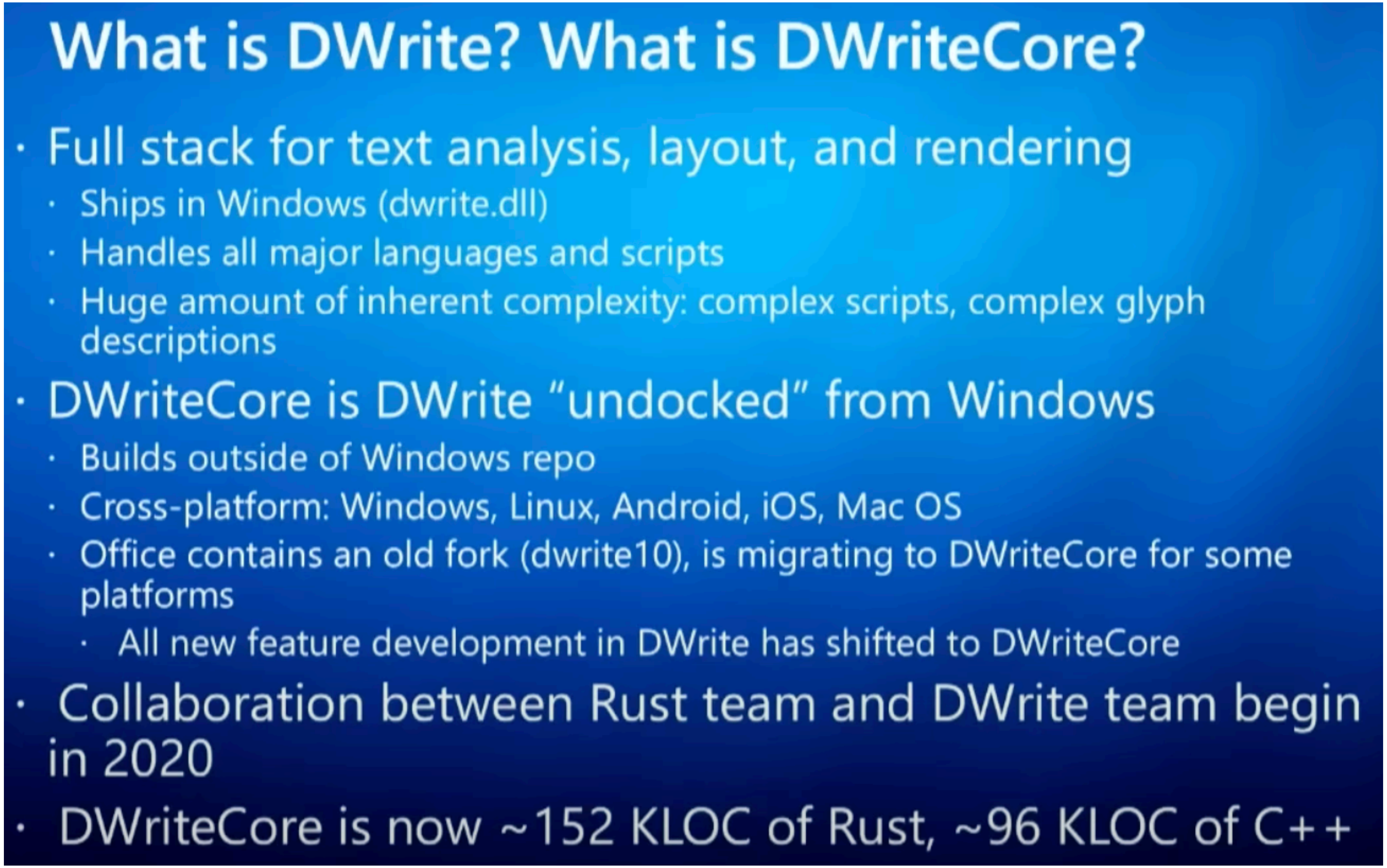
Myth #38

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?

youtube.com/watch?v=8T6ClX-y2AE&t=2703s

Myth #38



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=8T6ClX-y2AE&t=2703s

Myth #38 - 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.

```cpp
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;
};
```

```rust
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.

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

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

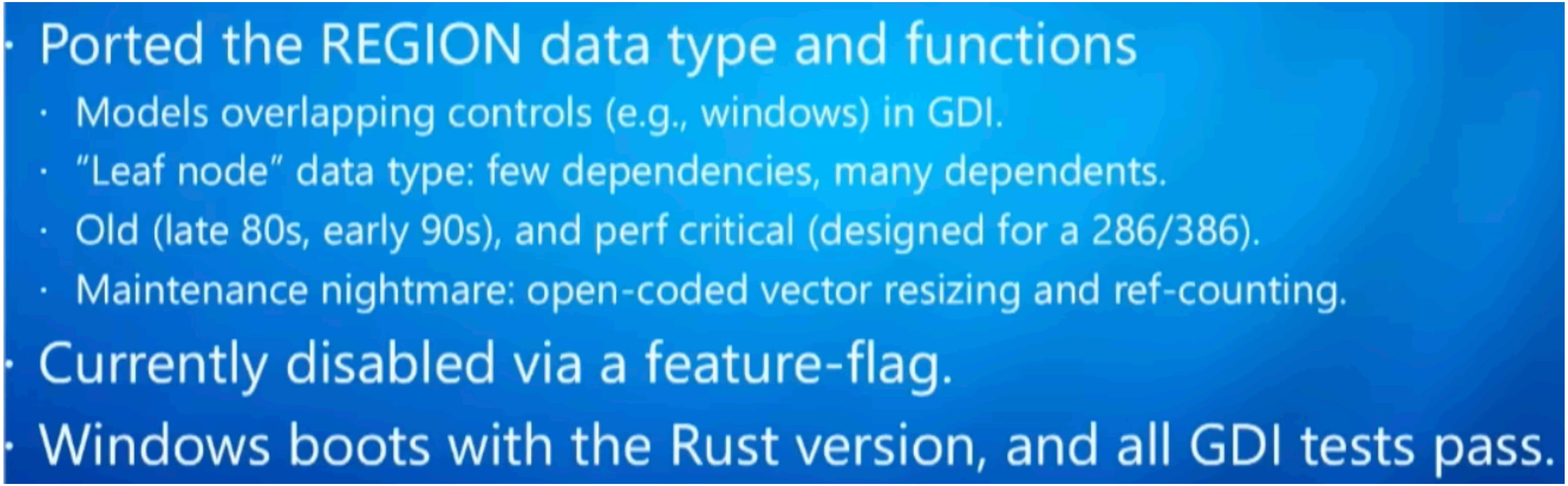
BlueHat IL

youtube.com/watch?v=8T6CIX-y2AE&t=2703s

Myth #38

Win32k **GDI** port to Rust

2nd experiment

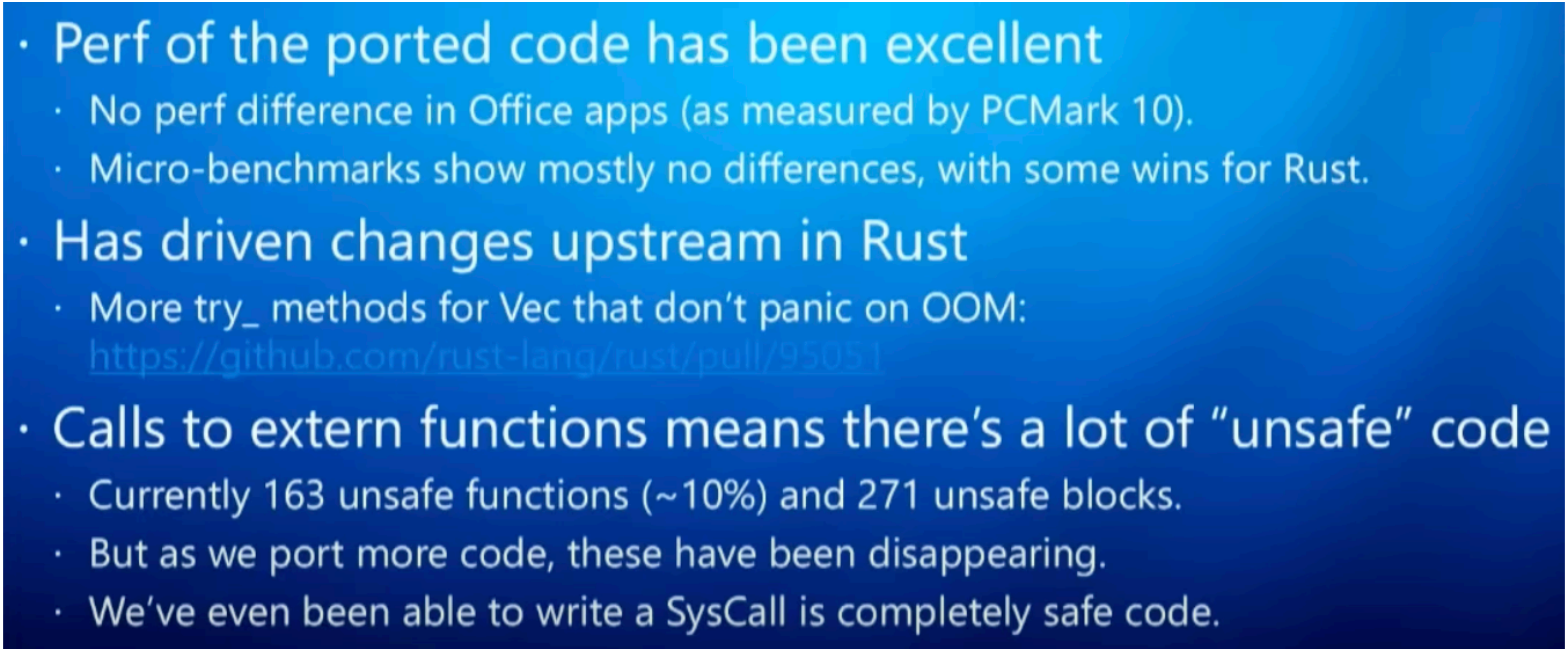


- Ported the REGION data type and functions
  - Models overlapping controls (e.g., windows) in GDI.
  - "Leaf node" data type: few dependencies, many dependents.
  - Old (late 80s, early 90s), and perf critical (designed for a 286/386).
  - Maintenance nightmare: open-coded vector resizing and ref-counting.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

youtube.com/watch?v=8T6ClX-y2AE&t=2703s

Myth #38

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.

youtube.com/watch?v=8T6ClX-y2AE&t=2703s

Myth #38

More on the way... 🤐

Myth #38

Rust Fact vs. Fiction

5 Insights from Google's Rust journey in 2022

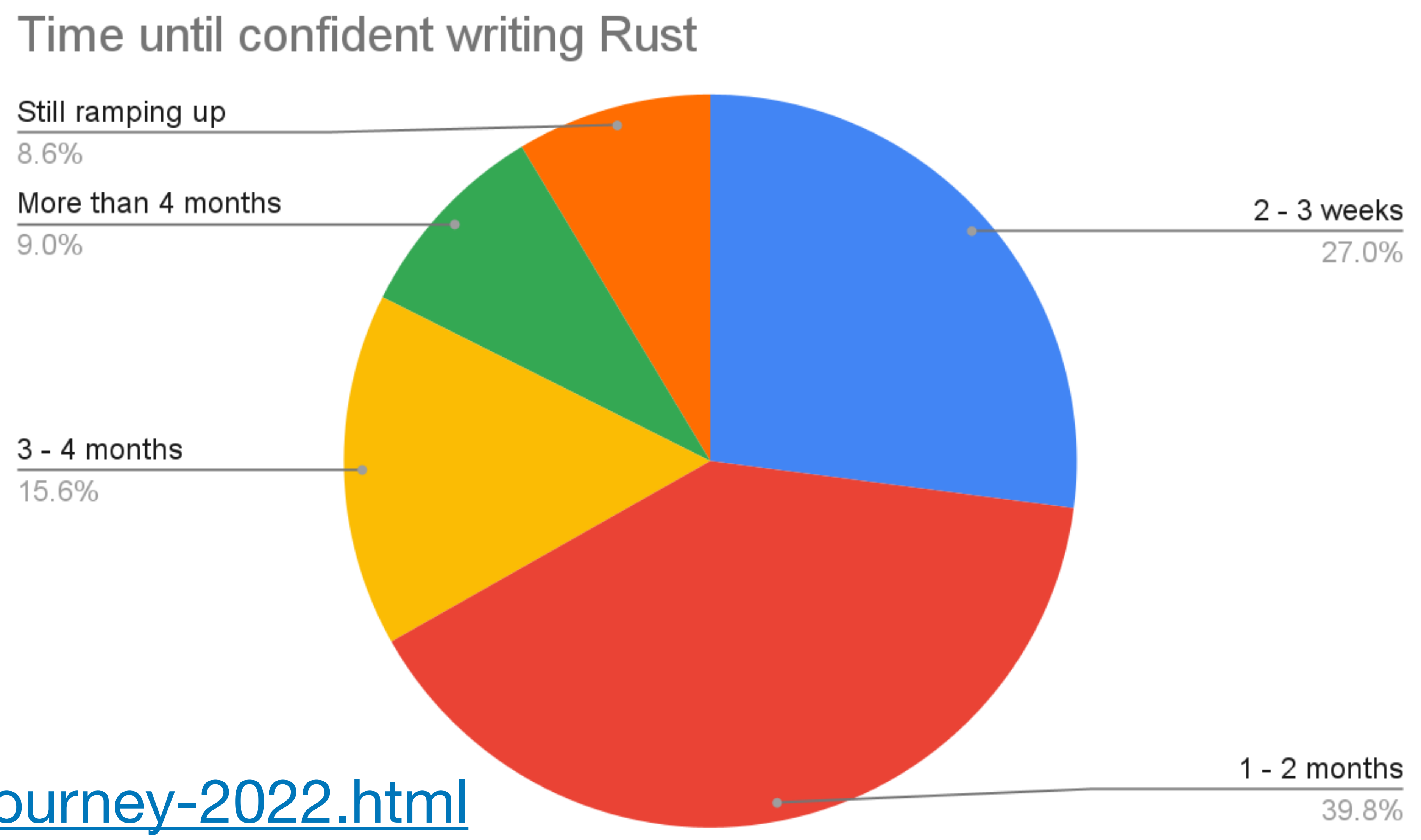
Rumor 1: Rust takes more than 6 months to learn – Debunked

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



opensource.googleblog.com/2023/06/rust-fact-vs-fiction-5-insights-from-googles-rust-journey-2022.html

Myth #38

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 re-declarations / No C++ annotations
- Broad type support - with safety
- Ergonomics - with safety

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

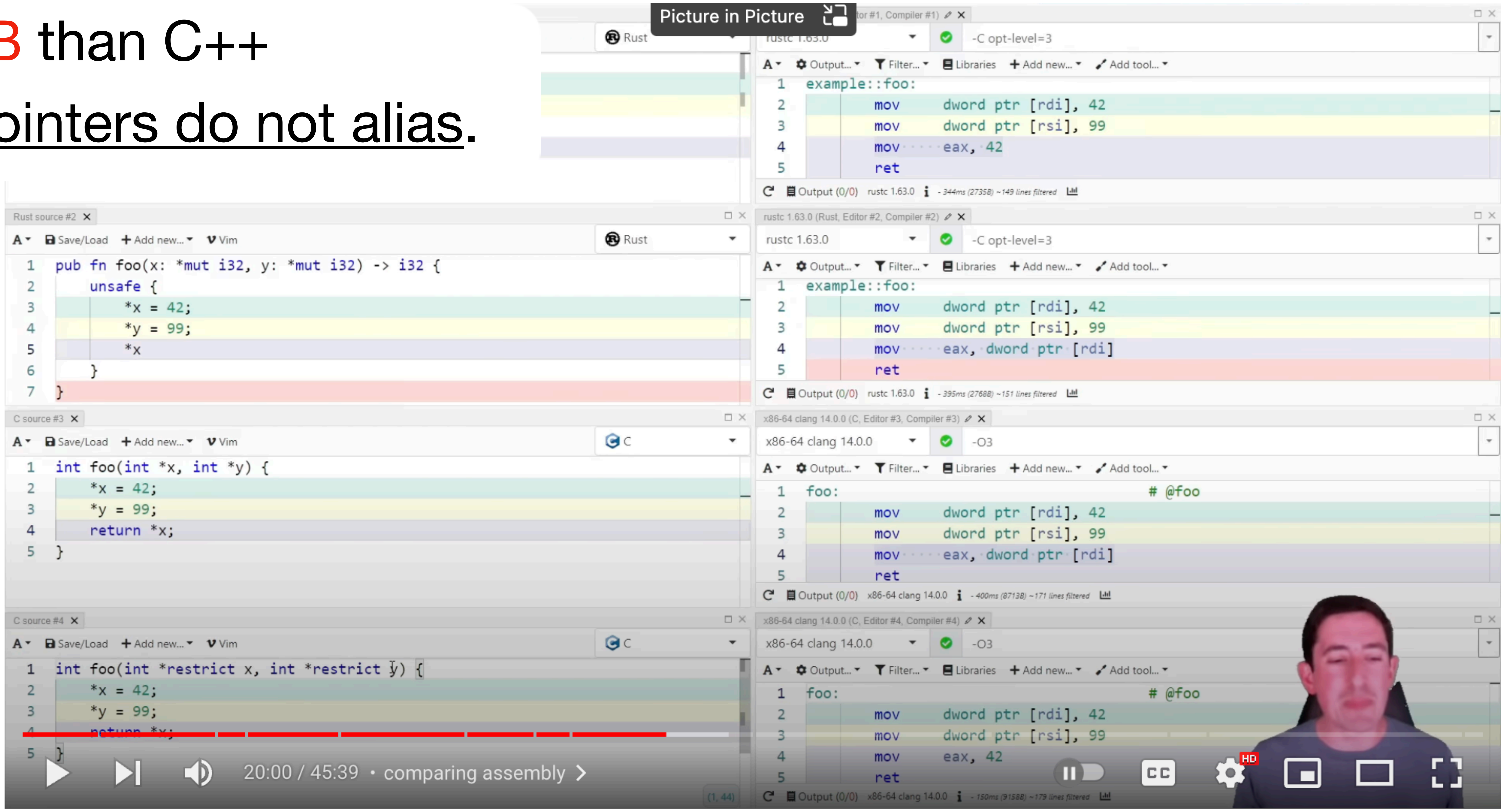
➡️ see moveit, autocxx and mosaic

chromium.org/Home/chromium-security/memory-safety/rust-and-c-interoperability/

Myth #38

unsafe { 😱 }

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



rustc 1.63.0 (Rust, Compiler #1) — -C opt-level=3

```
example::foo:
        mov     dword ptr [rdi], 42
        mov     dword ptr [rsi], 99
        mov     eax, 42
        ret
```

Rust source #2

```rust
pub fn foo(x: *mut i32, y: *mut i32) -> i32 {
    unsafe {
        *x = 42;
        *y = 99;
        *x
    }
}
```

rustc 1.63.0 (Rust, Editor #2, Compiler #2) — -C opt-level=3

```
example::foo:
        mov     dword ptr [rdi], 42
        mov     dword ptr [rsi], 99
        mov     eax, dword ptr [rdi]
        ret
```

C source #3

```c
int foo(int *x, int *y) {
    *x = 42;
    *y = 99;
    return *x;
}
```

x86-64 clang 14.0.0 (C, Editor #3, Compiler #3) — -O3

```
foo:                                    # @foo
        mov     dword ptr [rdi], 42
        mov     dword ptr [rsi], 99
        mov     eax, dword ptr [rdi]
        ret
```

C source #4

```c
int foo(int *restrict x, int *restrict y) {
    *x = 42;
    *y = 99;
    return *x;
}
```

x86-64 clang 14.0.0 (C, Editor #4, Compiler #4) — -O3

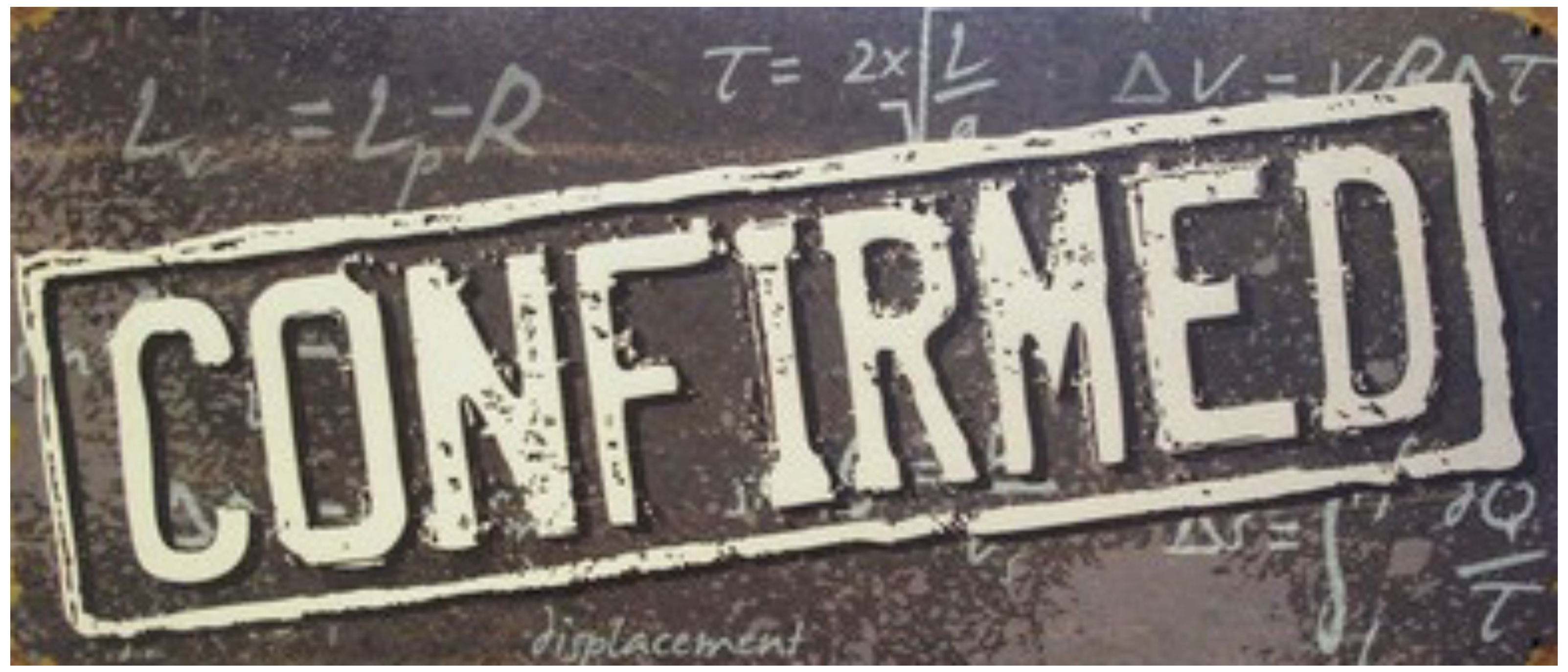
```
foo:                                    # @foo
        mov     dword ptr [rdi], 42
        mov     dword ptr [rsi], 99
        mov     eax, 42
        ret
```

Unsafe Rust is not C

youtube.com/watch?v=DG-VLezRkYQ

Myth #38

Just rewrite it in Rust 🦀



Myth #6

Successor languages are going to eat our lunch

Myth #6

~~Val~~ Hylo aims:

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

hylo-lang.org

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

github.com/carbon-language

📖 **The Year of C++ Successor Languages**
-- Lucian Radu Teodorescu

accu.org/journals/overload/30/172/teodorescu/

Myth #6

~~Val~~ Hylo aims:

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

hylo-lang.org

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

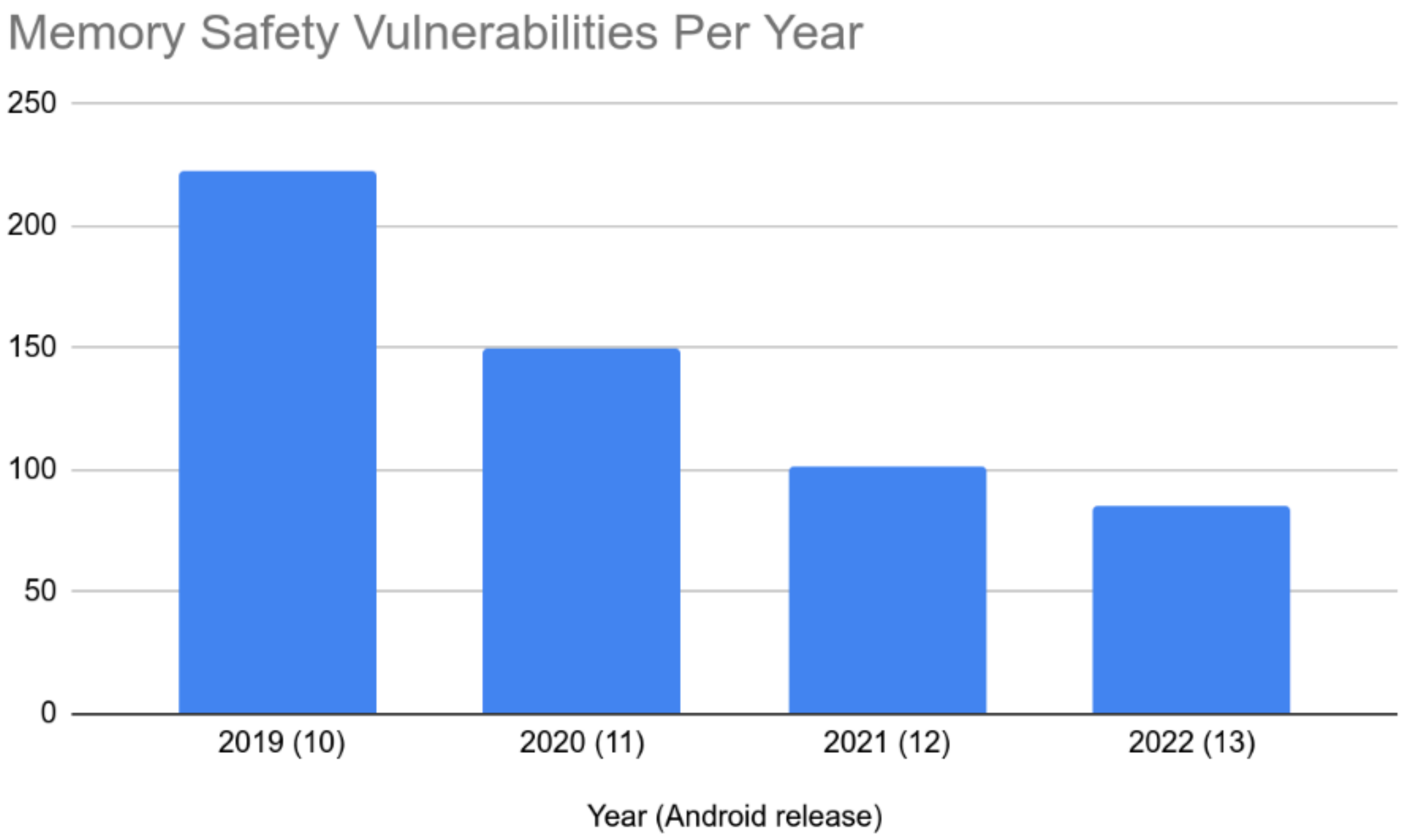
github.com/carbon-language

✓ perfect by construction 😄

📖 **The Year of C++ Successor Languages**
-- Lucian Radu Teodorescu

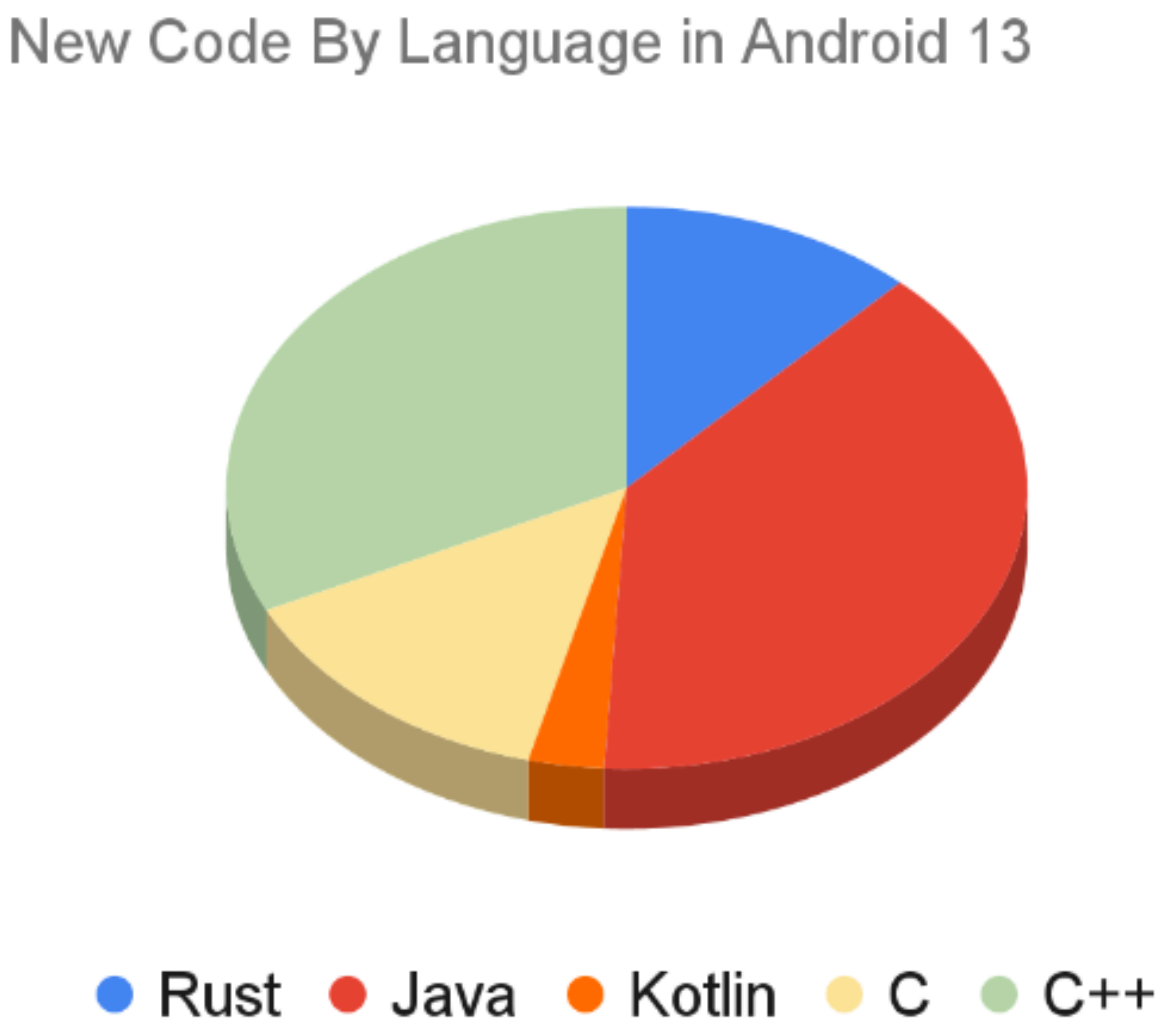
accu.org/journals/overload/30/172/teodorescu/

Myth #6 - Memory Safe Languages in 🤖 Android 13



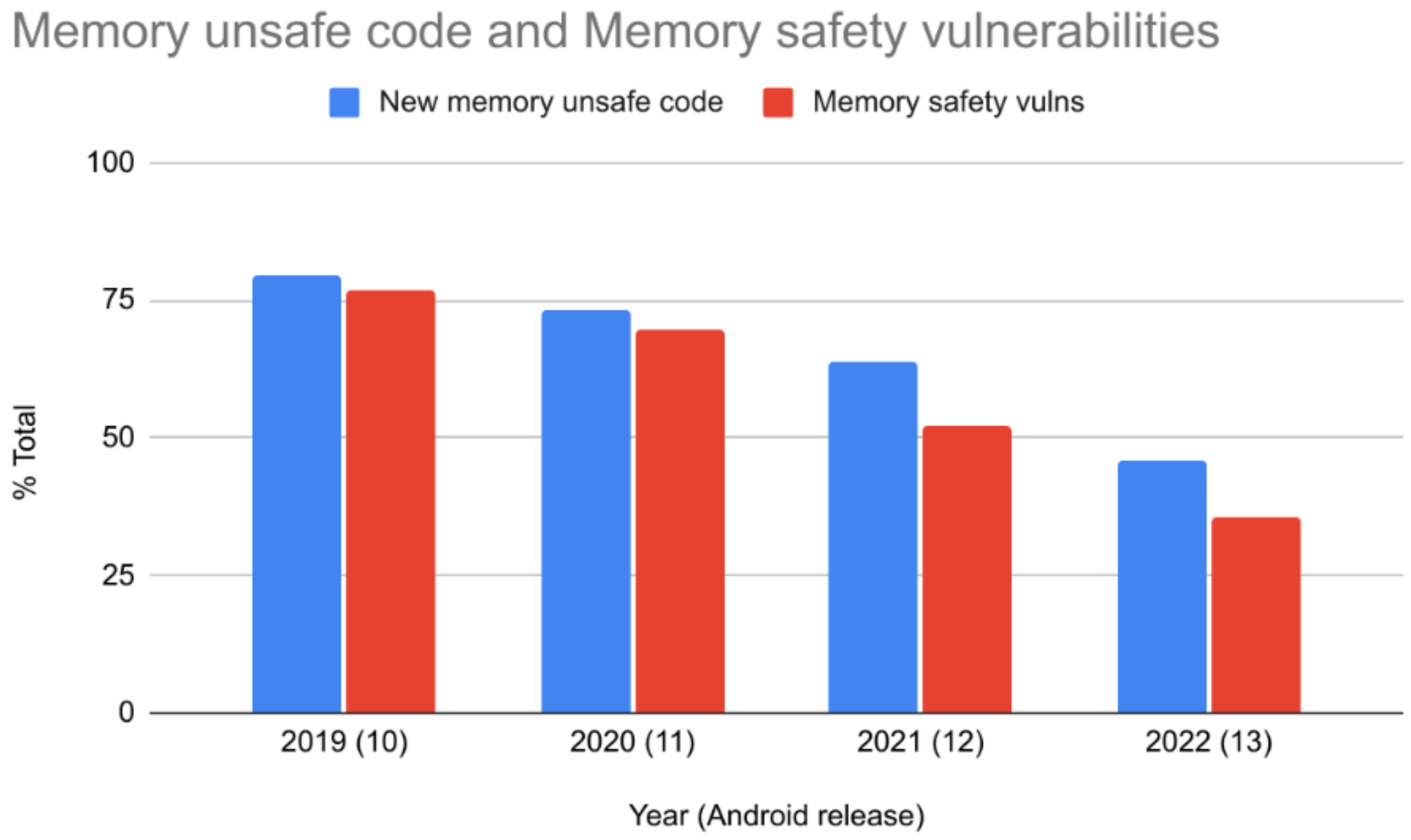
security.googleblog.com/2022/12/memory-safe-languages-in-android-13

Myth #6 - Memory Safe Languages in 🤖 Android 13



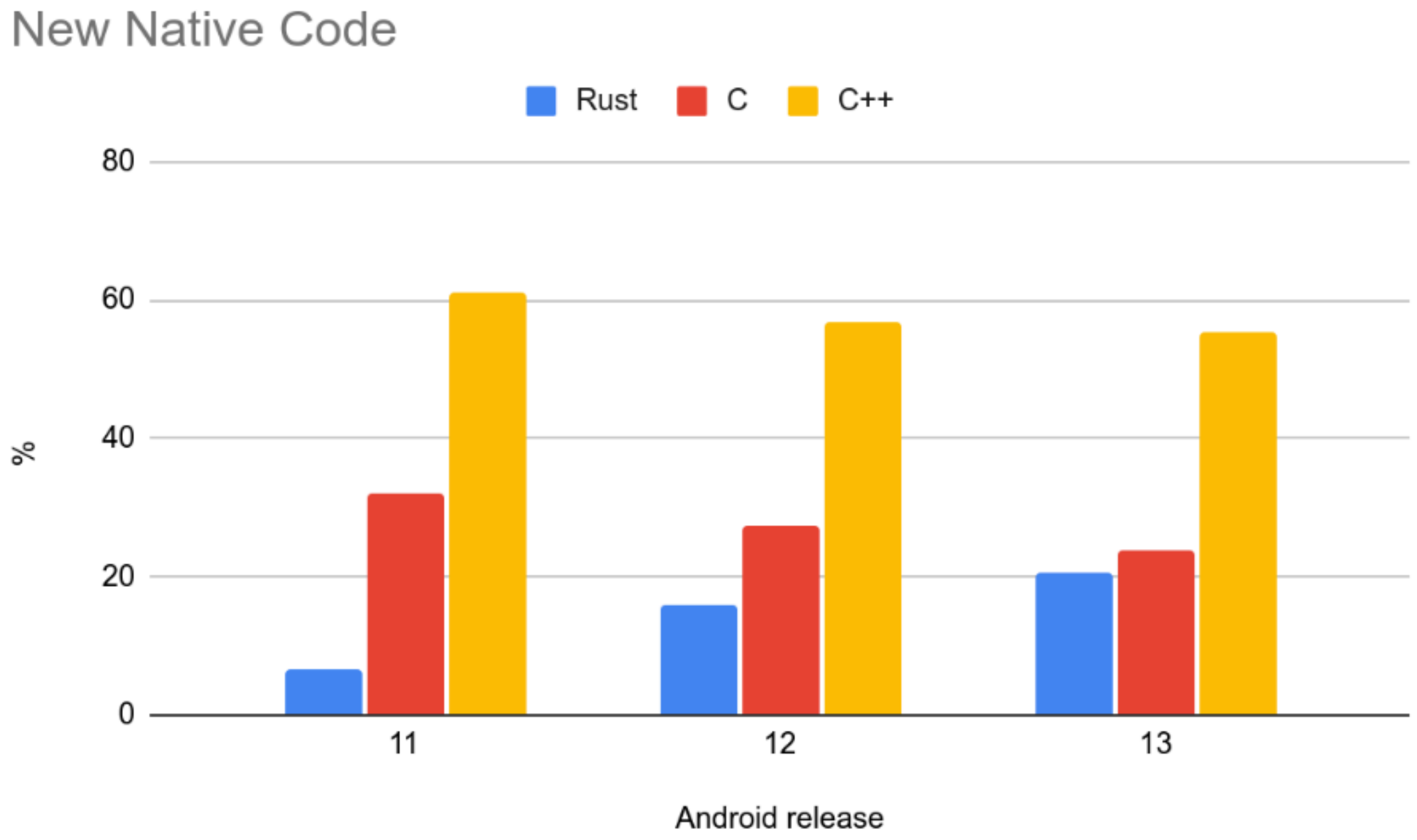
security.googleblog.com/2022/12/memory-safe-languages-in-android-13

Myth #6 - Memory Safe Languages in 🤖 Android 13



security.googleblog.com/2022/12/memory-safe-languages-in-android-13

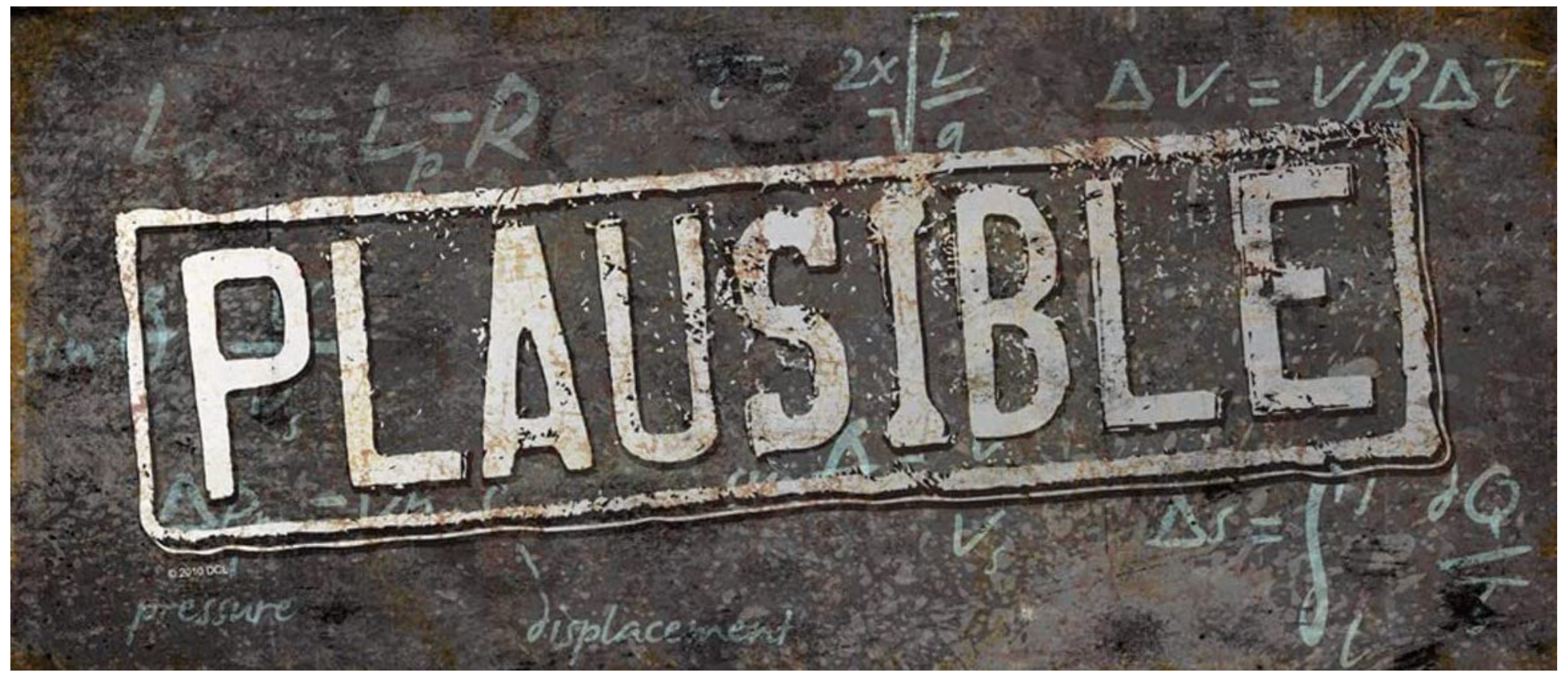
Myth #6 - Memory Safe Languages in 🤖 Android 13



security.googleblog.com/2022/12/memory-safe-languages-in-android-13

Myth #6

Successor languages are going to eat our lunch



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

Myth #39

std::ranges are safer than iterators

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

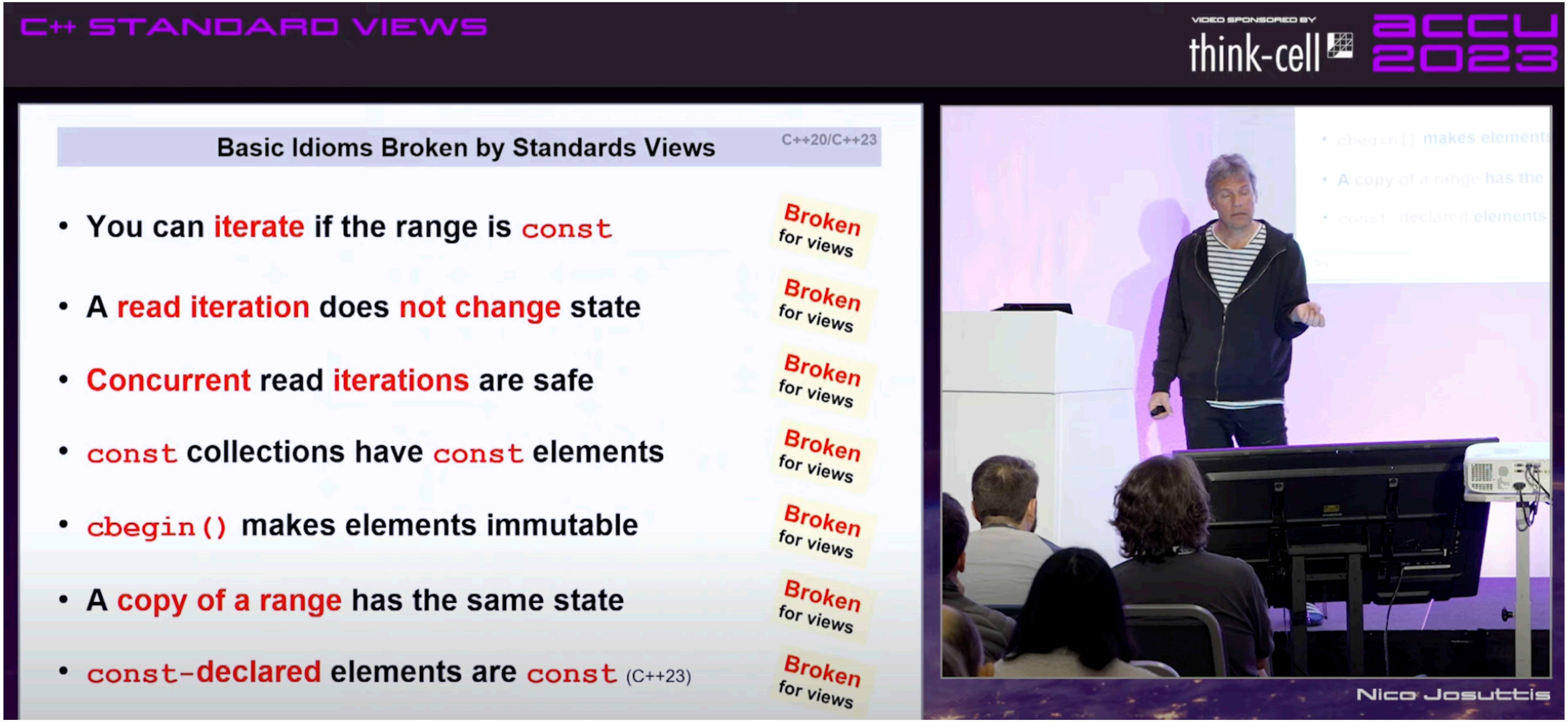
Myth #39

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 :)

Myth #39



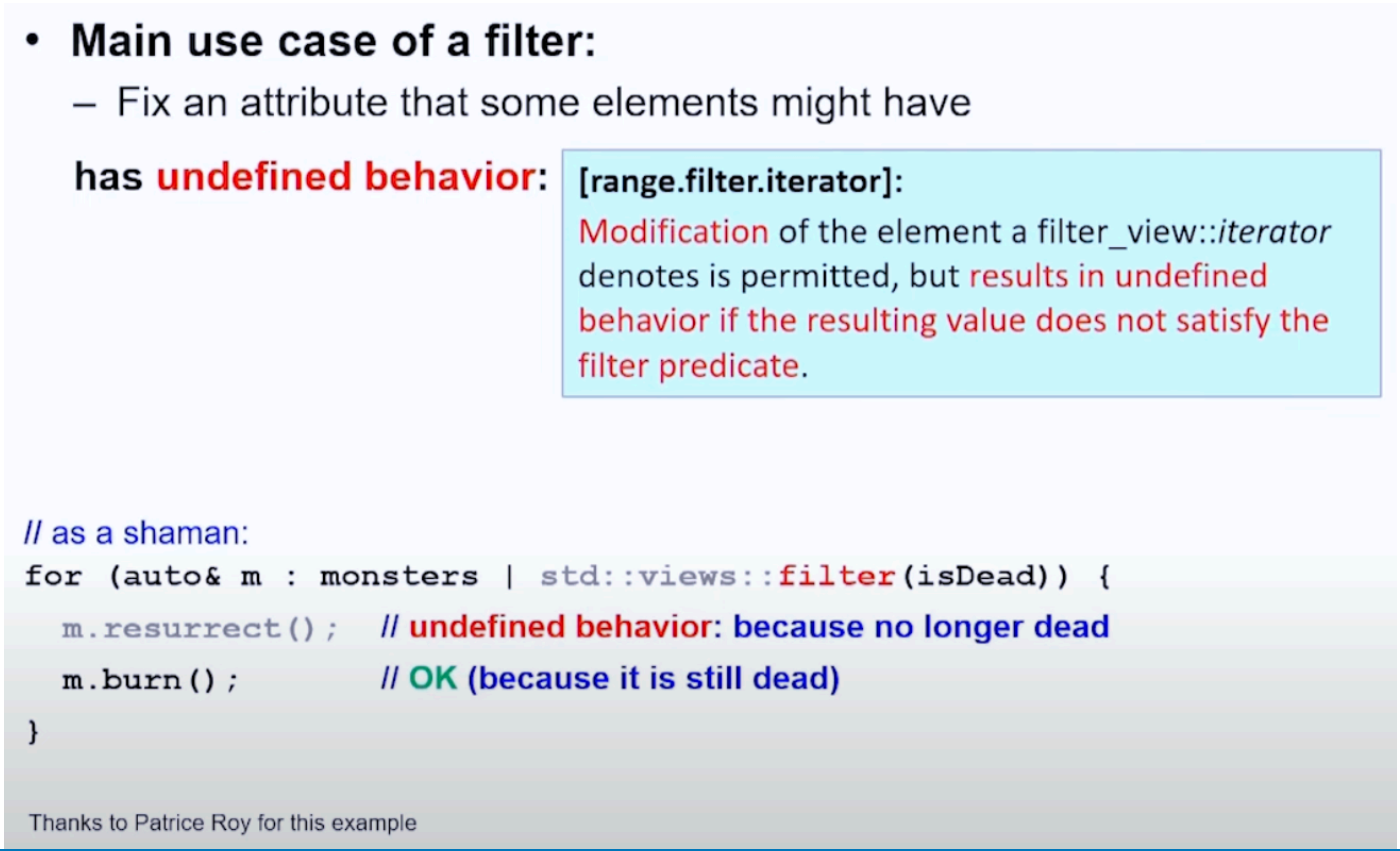
Basic Idioms Broken by Standards Views C++20/C++23

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

youtube.com/watch?v=qv29fo9sUjY

Myth #39

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.

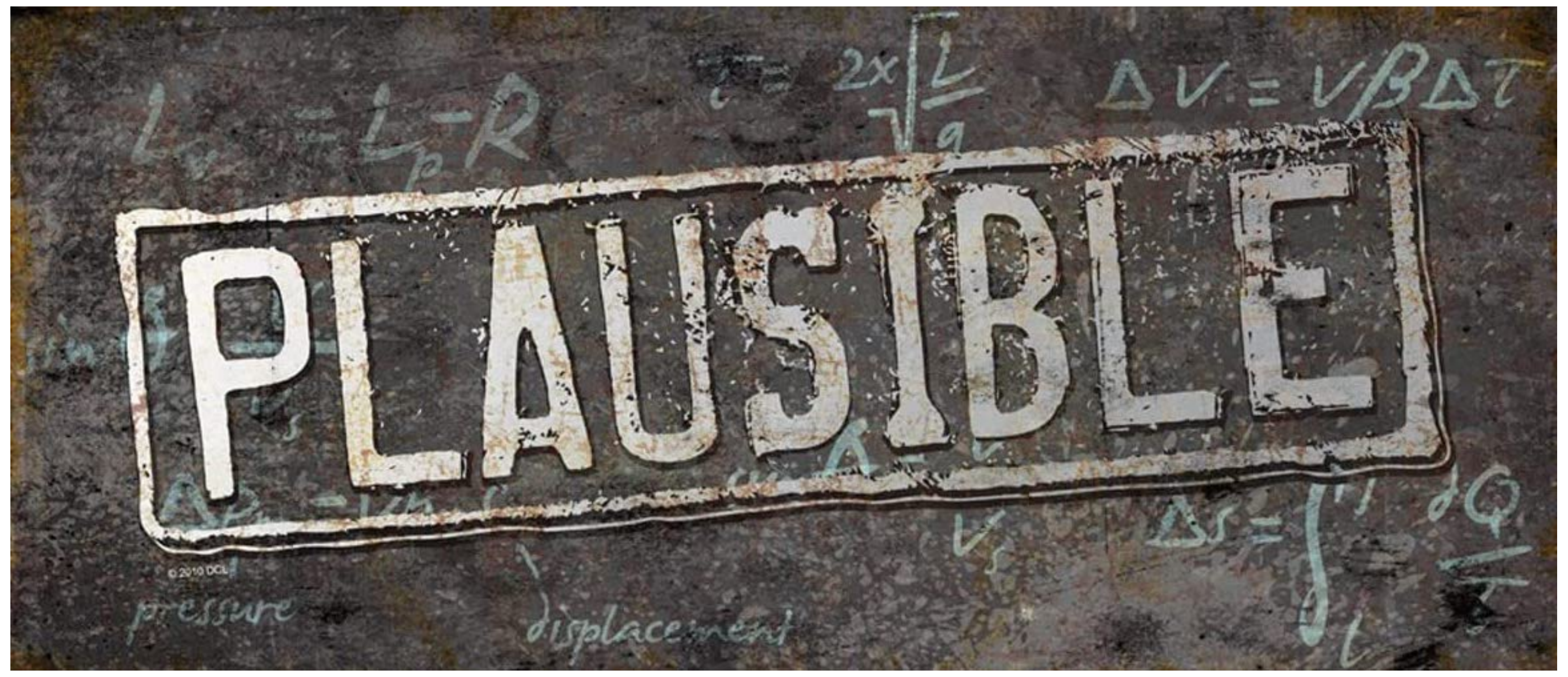
```
// as a shaman:
for (auto& m : monsters | std::views::filter(isDead)) {
  m.resurrect();    // undefined behavior: because no longer dead
  m.burn();         // OK (because it is still dead)
}
```

Thanks to Patrice Roy for this example

youtube.com/watch?v=qv29fo9sUjY

Myth #39

std::ranges are safer than iterators



Myth #7

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 🙂

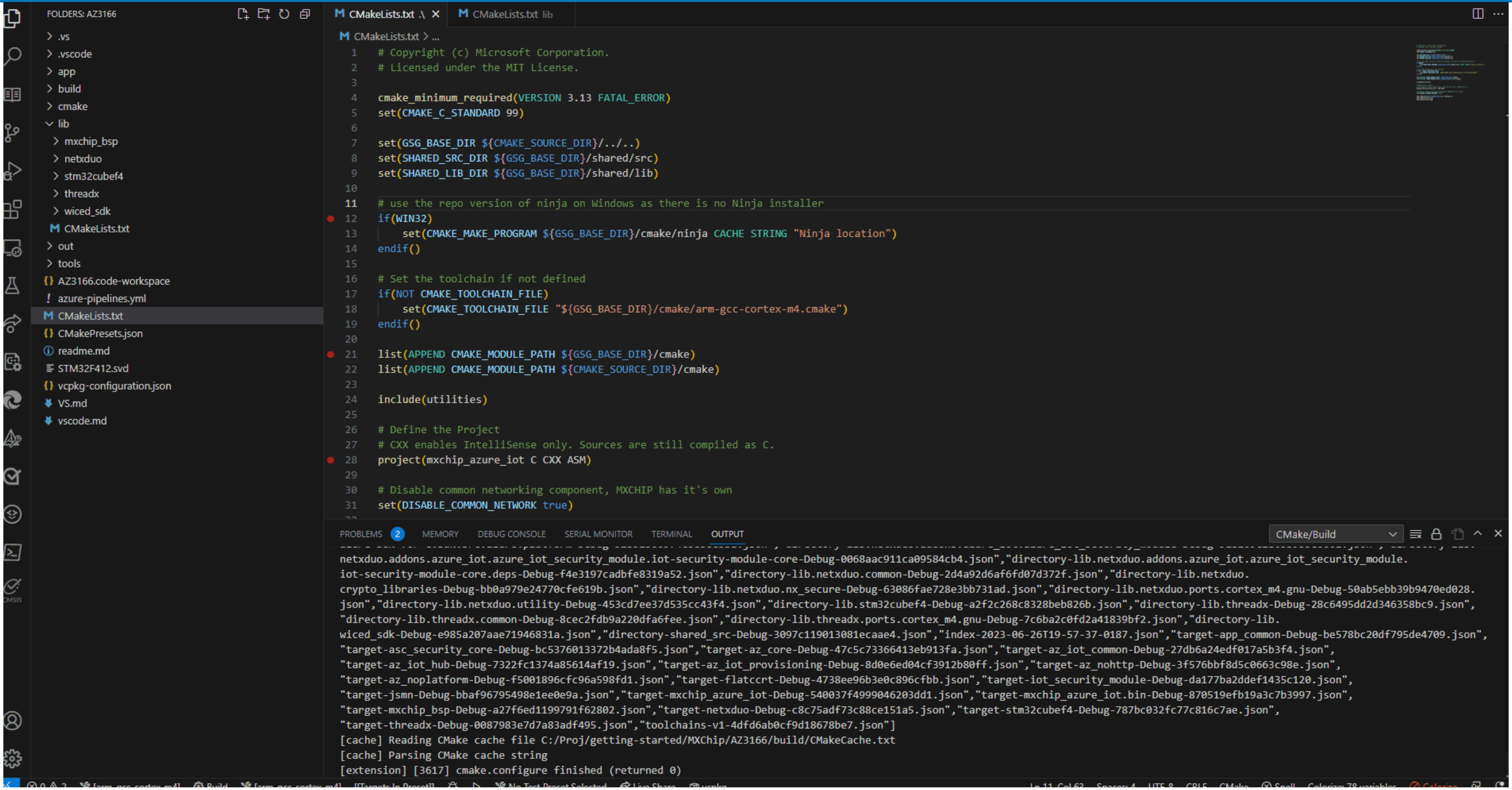
twitter.com/pati_gallardo/status/1672137915575545856?s=46&t=dcjdCXT0jeVLLjXhQ3J85A

Myth #7

CMake Debugger
in Visual Studio and VSCode

youtube.com/watch?v=1eVJBEV9NTk

Myth #7

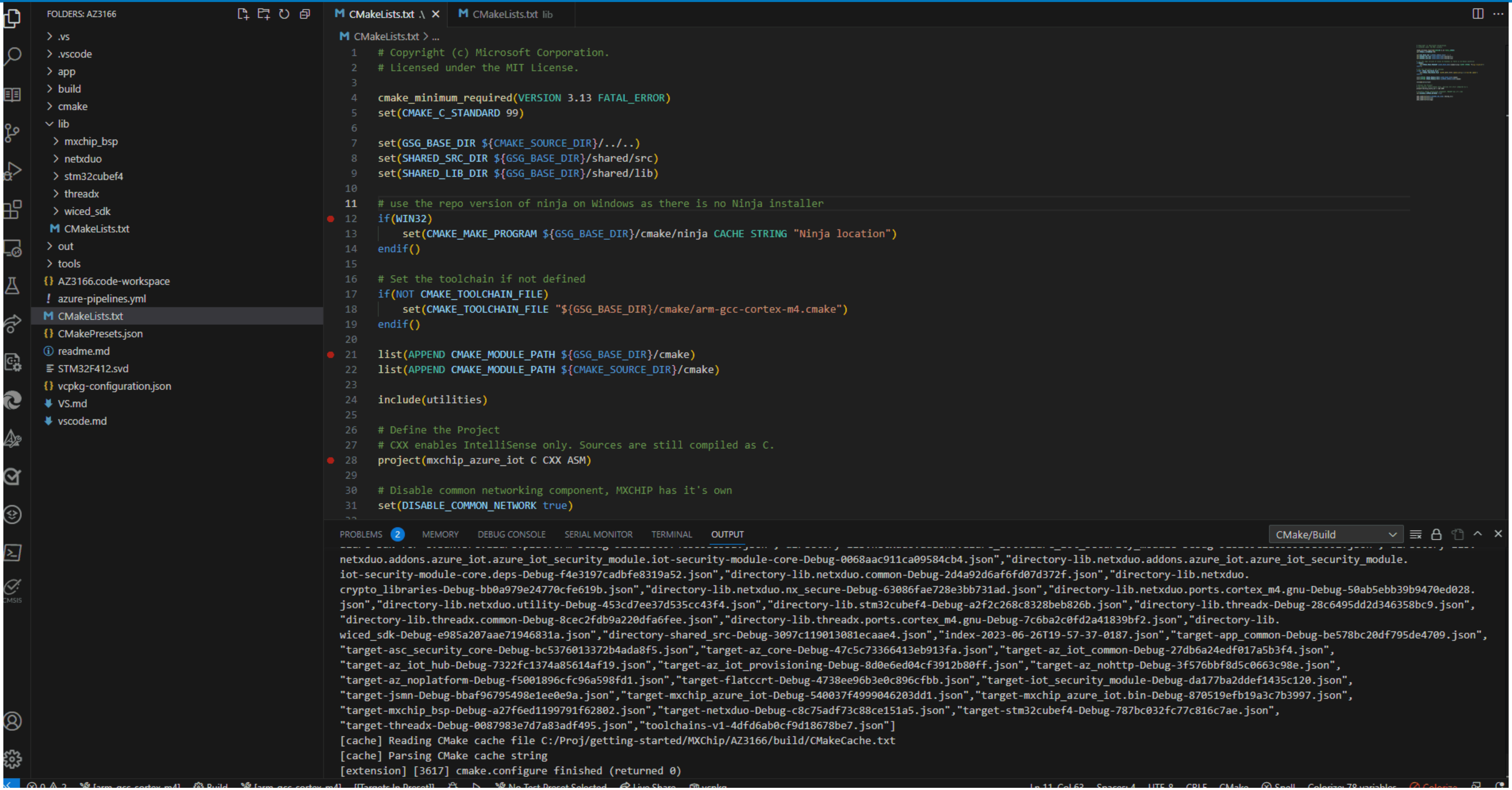


```cmake
# Copyright (c) Microsoft Corporation.
# Licensed under the MIT License.

cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
set(CMAKE_C_STANDARD 99)

set(GSG_BASE_DIR ${CMAKE_SOURCE_DIR}/../..)
set(SHARED_SRC_DIR ${GSG_BASE_DIR}/shared/src)
set(SHARED_LIB_DIR ${GSG_BASE_DIR}/shared/lib)

# use the repo version of ninja on Windows as there is no Ninja installer
if(WIN32)
    set(CMAKE_MAKE_PROGRAM ${GSG_BASE_DIR}/cmake/ninja CACHE STRING "Ninja location")
endif()

# Set the toolchain if not defined
if(NOT CMAKE_TOOLCHAIN_FILE)
    set(CMAKE_TOOLCHAIN_FILE "${GSG_BASE_DIR}/cmake/arm-gcc-cortex-m4.cmake")
endif()

list(APPEND CMAKE_MODULE_PATH ${GSG_BASE_DIR}/cmake)
list(APPEND CMAKE_MODULE_PATH ${CMAKE_SOURCE_DIR}/cmake)

include(utilities)

# Define the Project
# CXX enables IntelliSense only. Sources are still compiled as C.
project(mxchip_azure_iot C CXX ASM)

# Disable common networking component, MXCHIP has it's own
set(DISABLE_COMMON_NETWORK true)
```

```
netxduo.addons.azure_iot.azure_iot_security_module.iot-security-module-core-Debug-0068aac911ca09584cb4.json","directory-lib.netxduo.addons.azure_iot.azure_iot_security_module.
iot-security-module-core.deps-Debug-f4e3197cadbfe8319a52.json","directory-lib.netxduo.common-Debug-2d4a92d6af6fd07d372f.json","directory-lib.netxduo.
crypto_libraries-Debug-bb0a979e24770cfe619b.json","directory-lib.netxduo.nx_secure-Debug-63086fae728e3bb731ad.json","directory-lib.netxduo.ports.cortex_m4.gnu-Debug-50ab5ebb39b9470ed028.
json","directory-lib.netxduo.utility-Debug-453cd7ee37d535cc43f4.json","directory-lib.stm32cubef4-Debug-a2f2c268c8328beb826b.json","directory-lib.threadx-Debug-28c6495dd2d346358bc9.json",
"directory-lib.threadx.common-Debug-8cec2fdb9a220dfa6fee.json","directory-lib.threadx.ports.cortex_m4.gnu-Debug-7c6ba2c0fd2a41839bf2.json","directory-lib.
wiced_sdk-Debug-e985a207aae71946831a.json","directory-shared_src-Debug-3097c119013081ecaae4.json","index-2023-06-26T19-57-37-0187.json","target-app_common-Debug-be578bc20df795de4709.json",
"target-asc_security_core-Debug-bc5376013372b4ada8f5.json","target-az_core-Debug-47c5c73366413eb913fa.json","target-az_iot_common-Debug-27db6a24edf017a5b3f4.json",
"target-az_iot_hub-Debug-7322fc1374a85614af19.json","target-az_iot_provisioning-Debug-8d0e6ed04cf3912b80ff.json","target-az_nohttp-Debug-3f576bbf8d5c0663c98e.json",
"target-az_noplatform-Debug-f5001896cfc96a598fd1.json","target-flatccrt-Debug-4738ee96b3e0c896cfbb.json","target-iot_security_module-Debug-da177ba2ddef1435c120.json",
"target-jsmn-Debug-bbaf96795498e1ee0e9a.json","target-mxchip_azure_iot-Debug-540037f4999046203dd1.json","target-mxchip_azure_iot.bin-Debug-870519efb19a3c7b3997.json",
"target-mxchip_bsp-Debug-a27f6ed1199791f62802.json","target-netxduo-Debug-c8c75adf73c88ce151a5.json","target-stm32cubef4-Debug-787bc032fc77c816c7ae.json",
"target-threadx-Debug-0087983e7d7a83adf495.json","toolchains-v1-4dfd6ab0cf9d18678be7.json"]
[cache] Reading CMake cache file C:/Proj/getting-started/MXChip/AZ3166/build/CMakeCache.txt
[cache] Parsing CMake cache string
[extension] [3617] cmake.configure finished (returned 0)
```

devblogs.microsoft.com/cppblog/introducing-cmake-debugger-in-vs-code

Myth #7



```
# Copyright (c) Microsoft Corporation.
# Licensed under the MIT License.

cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
set(CMAKE_C_STANDARD 99)

set(GSG_BASE_DIR ${CMAKE_SOURCE_DIR}/../..)
set(SHARED_SRC_DIR ${GSG_BASE_DIR}/shared/src)
set(SHARED_LIB_DIR ${GSG_BASE_DIR}/shared/lib)

# use the repo version of ninja on Windows as there is no Ninja installer
if(WIN32)
    set(CMAKE_MAKE_PROGRAM ${GSG_BASE_DIR}/cmake/ninja CACHE STRING "Ninja location")
endif()

# Set the toolchain if not defined
if(NOT CMAKE_TOOLCHAIN_FILE)
    set(CMAKE_TOOLCHAIN_FILE "${GSG_BASE_DIR}/cmake/arm-gcc-cortex-m4.cmake")
endif()

list(APPEND CMAKE_MODULE_PATH ${GSG_BASE_DIR}/cmake)
list(APPEND CMAKE_MODULE_PATH ${CMAKE_SOURCE_DIR}/cmake)

include(utilities)

# Define the Project
# CXX enables IntelliSense only. Sources are still compiled as C.
project(mxchip_azure_iot C CXX ASM)

# Disable common networking component, MXCHIP has it's own
set(DISABLE_COMMON_NETWORK true)
```

```
netxduo.addons.azure_iot.azure_iot_security_module.iot-security-module-core-Debug-0068aac911ca09584cb4.json","directory-lib.netxduo.addons.azure_iot.azure_iot_security_module.
iot-security-module-core.deps-Debug-f4e3197cadbfe8319a52.json","directory-lib.netxduo.common-Debug-2d4a92d6af6fd07d372f.json","directory-lib.netxduo.
crypto_libraries-Debug-bb0a979e24770cfe619b.json","directory-lib.netxduo.nx_secure-Debug-63086fae728e3bb731ad.json","directory-lib.netxduo.ports.cortex_m4.gnu-Debug-50ab5ebb39b9470ed028.
json","directory-lib.netxduo.utility-Debug-453cd7ee37d535cc43f4.json","directory-lib.stm32cubef4-Debug-a2f2c268c8328beb826b.json","directory-lib.threadx-Debug-28c6495dd2d346358bc9.json",
"directory-lib.threadx.common-Debug-8cec2fdb9a220dfa6fee.json","directory-lib.threadx.ports.cortex_m4.gnu-Debug-7c6ba2c0fd2a41839bf2.json","directory-lib.
wiced_sdk-Debug-e985a207aae71946831a.json","directory-shared_src-Debug-3097c119013081ecaae4.json","index-2023-06-26T19-57-37-0187.json","target-app_common-Debug-be578bc20df795de4709.json",
"target-asc_security_core-Debug-bc5376013372b4ada8f5.json","target-az_core-Debug-47c5c73366413eb913fa.json","target-az_iot_common-Debug-27db6a24edf017a5b3f4.json",
"target-az_iot_hub-Debug-7322fc1374a85614af19.json","target-az_iot_provisioning-Debug-8d0e6ed04cf3912b80ff.json","target-az_nohttp-Debug-3f576bbf8d5c0663c98e.json",
"target-az_noplatform-Debug-f5001896cfc96a598fd1.json","target-flatccrt-Debug-4738ee96b3e0c896cfbb.json","target-iot_security_module-Debug-da177ba2ddef1435c120.json",
"target-jsmn-Debug-bbaf96795498e1ee0e9a.json","target-mxchip_azure_iot-Debug-540037f4999046203dd1.json","target-mxchip_azure_iot.bin-Debug-870519efb19a3c7b3997.json",
"target-mxchip_bsp-Debug-a27f6ed1199791f62802.json","target-netxduo-Debug-c8c75adf73c88ce151a5.json","target-stm32cubef4-Debug-787bc032fc77c816c7ae.json",
"target-threadx-Debug-0087983e7d7a83adf495.json","toolchains-v1-4dfd6ab0cf9d18678be7.json"]
[cache] Reading CMake cache file C:/Proj/getting-started/MXChip/AZ3166/build/CMakeCache.txt
[cache] Parsing CMake cache string
[extension] [3617] cmake.configure finished (returned 0)
```

devblogs.microsoft.com/cppblog/introducing-cmake-debugger-in-vs-code

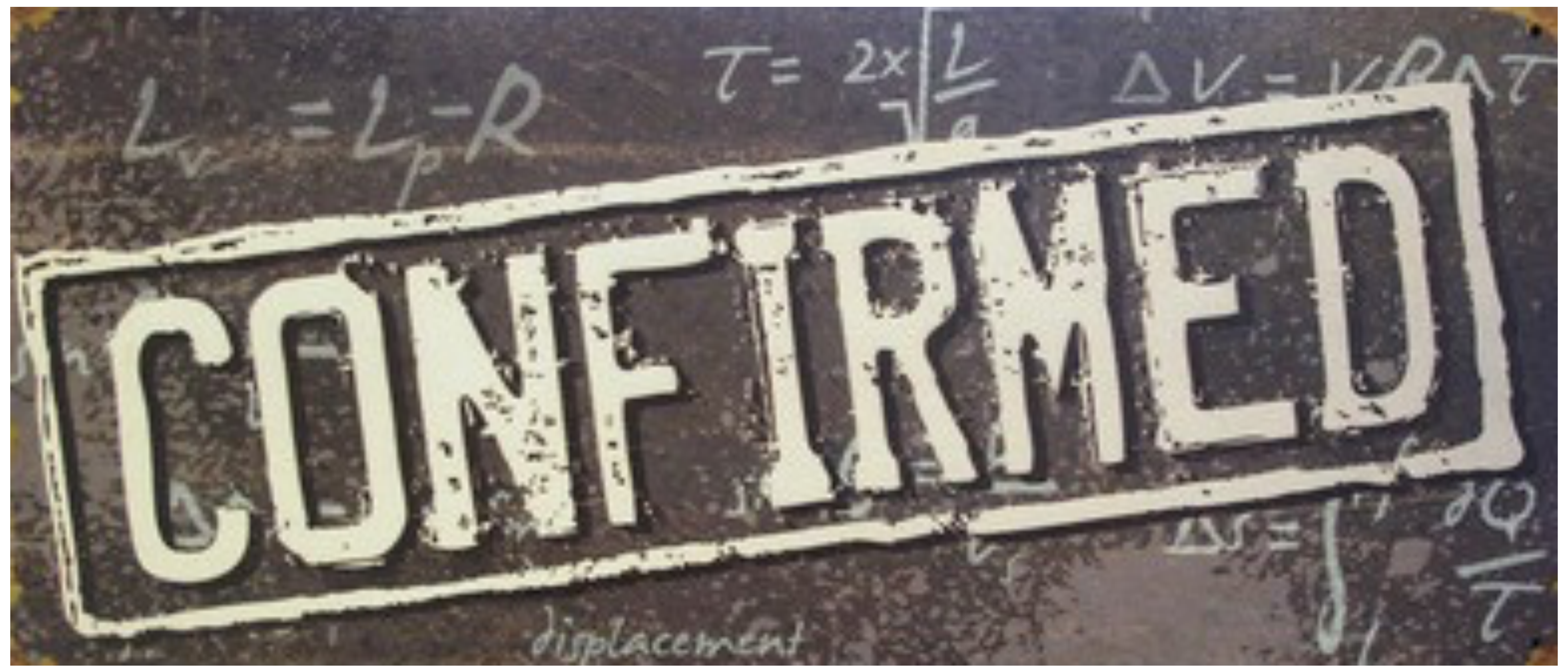
Myth #7

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

CMake Debugger: VS + VSCode + Rider + CLion

Myth #7

CMake is the gold standard of C++ project systems



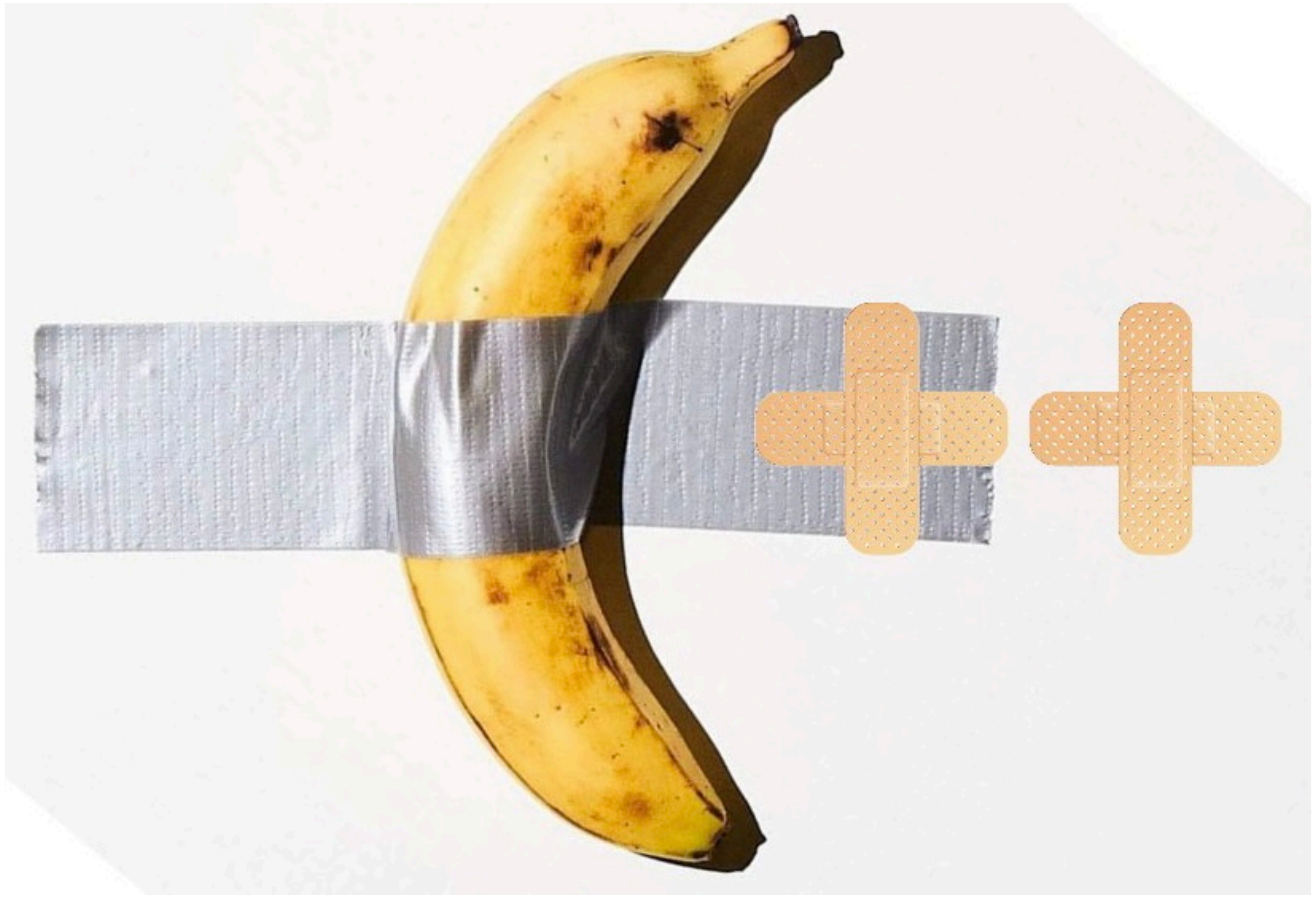
Myth #0

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

Mythbusting Series

Other C++ Mythbusters

Other C++ Mythbusters

Some Programming Myths Revisited

Patrice Roy
Patrice.Roy@USherbrooke.ca
CeFTI, Université de Sherbrooke
Patrice.Roy@clg.qc.ca
Collège Lionel-Groulx

CppCon The C++ Conference
2019 cppcon.org
Patrice Roy
Some Programming Myths Revisited
Video Sponsorship Provided By:
ansatz
0:25 / 1:00:50

AURORA

Some Programming Myths Revisited - Patrice Roy - CppCon 2019

youtube.com/watch?v=KNqRjzSlUVo

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/

C++ Prague Meetup

September 2023

@ciura_victor
@ciura_victor@hachyderm.io

Victor Ciura
Principal Engineer
Visual C++