

C++ MythBusters Strike 2

Victor Ciura



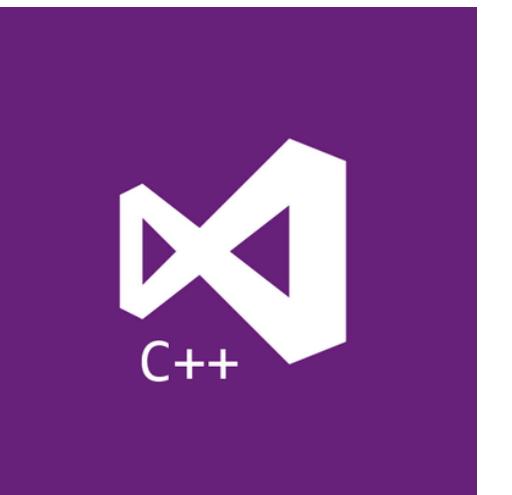


June 2023

Victor Ciura
Principal Engineer
Visual C++

 @ciura_victor

 @ciura_victor@hachyderm.io



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

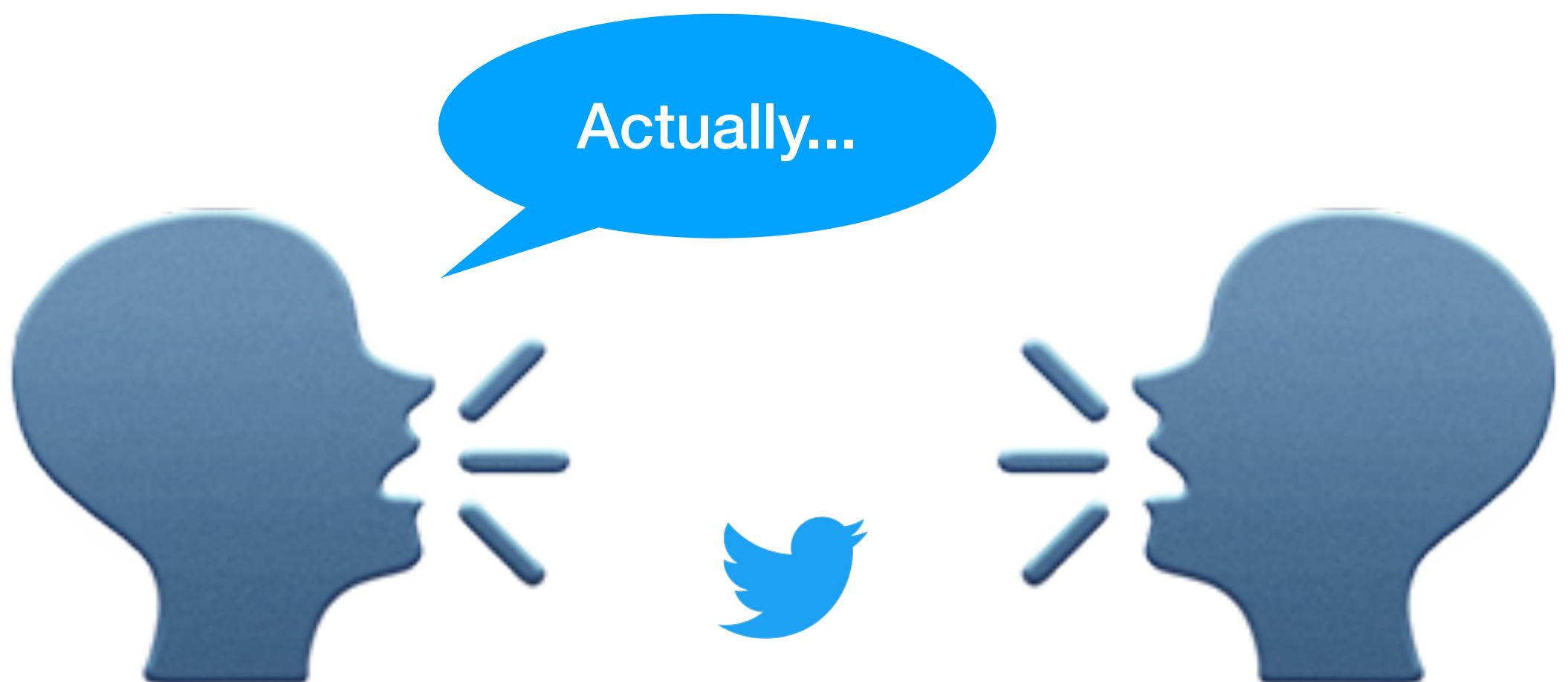


Do ask questions as we go along

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

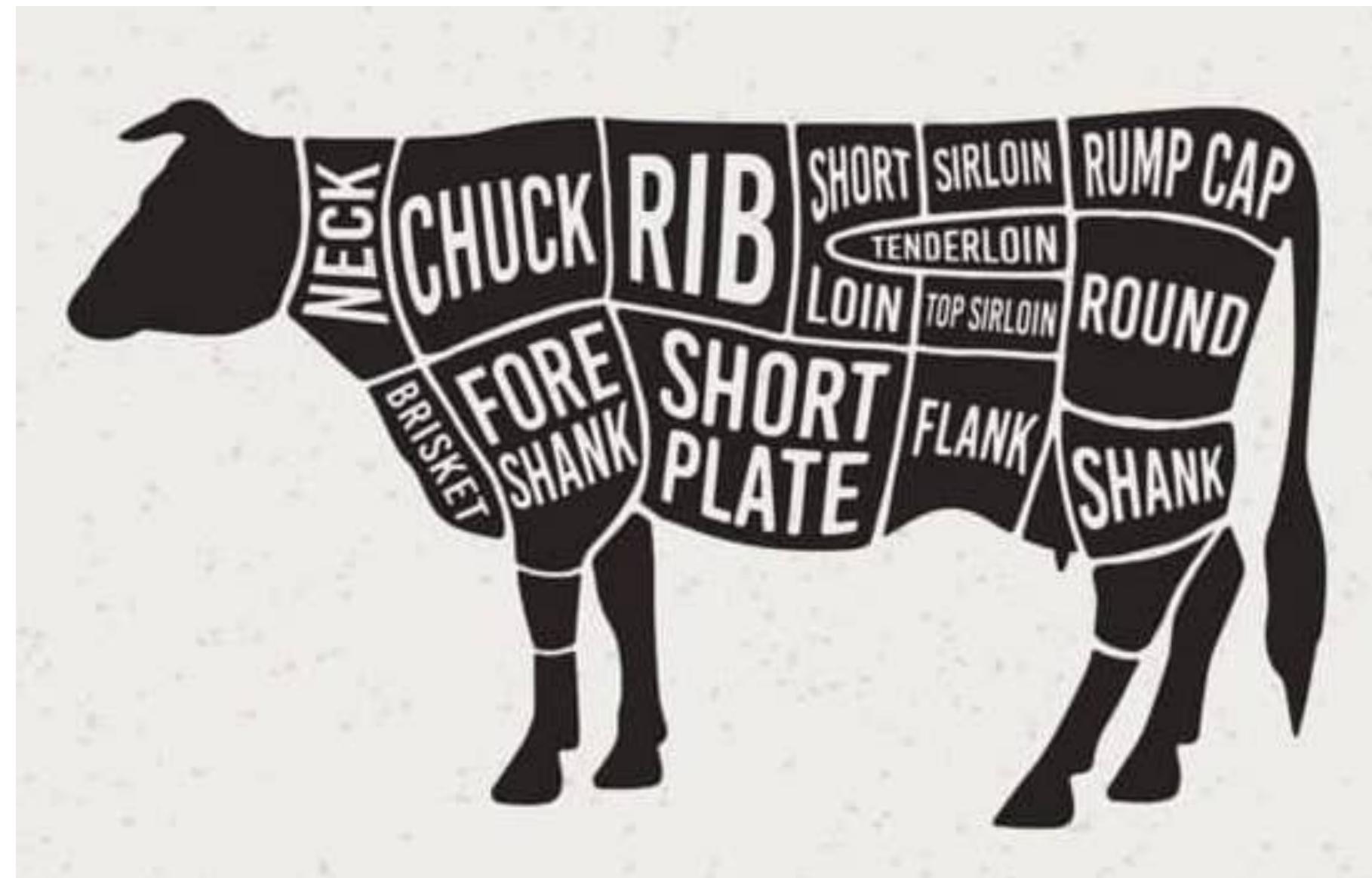
We're Different

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)



StackOverflow

How it started...

Mythbusting with Jason - unscripted improv (Pandemic edition)

21k views

youtube.com/watch?v=Bu1AEze14Ns

The video player displays two hosts in a video feed. The top host is Victor Ciura, and the bottom host is Jason Turner. The video title is "C++ Mythbusting with Victor and Jason". The video has 18,218 views and was streamed live on Jan 29, 2021. The video duration is 2:03:29. The video player interface includes a progress bar at 32:48 / 2:03:29, and controls for play, volume, and settings.

COMPILER EXPLORER

C++ source #1 x

A 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     if (something) {
11         return "Hello World";
12     } else {
13         return std::nullopt;
14     }
15 }
16
17 std::size_t get_optional_string_size(const bool something) {
18     const auto optional_str = get_optional_value(something);
19     if (optional_str) {
20         return optional_str->size();
21     } else {
22         return std::string::npos;
23     }
24 }
25 }
```

x86-64 gcc (trunk) | Editor #1, Compiler #1 C++ x

Sponsors intel PC-lint Solid Birds Share Other Policies

A 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
```

C Output (0/0) x86-64 gcc (trunk) - 3231ms (3198646)
#1 with x86-64 gcc (trunk) x
A 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 a

C++ Mythbusters - Season 1

The image shows a screenshot of a video player interface. At the top, there's a banner for "C++ MythBusters" with the year "2022" on the right. Below the banner, the title "Myth #24" is displayed. The main content area features a slide from Compiler Explorer. On the left, the C++ source code is shown:

```
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"; no more SSO
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 }
```

Annotations on the slide include a blue callout pointing to the return statement: "no more SSO" and another pointing to the "get_size" function: "compiler still sees through it and inlines it".

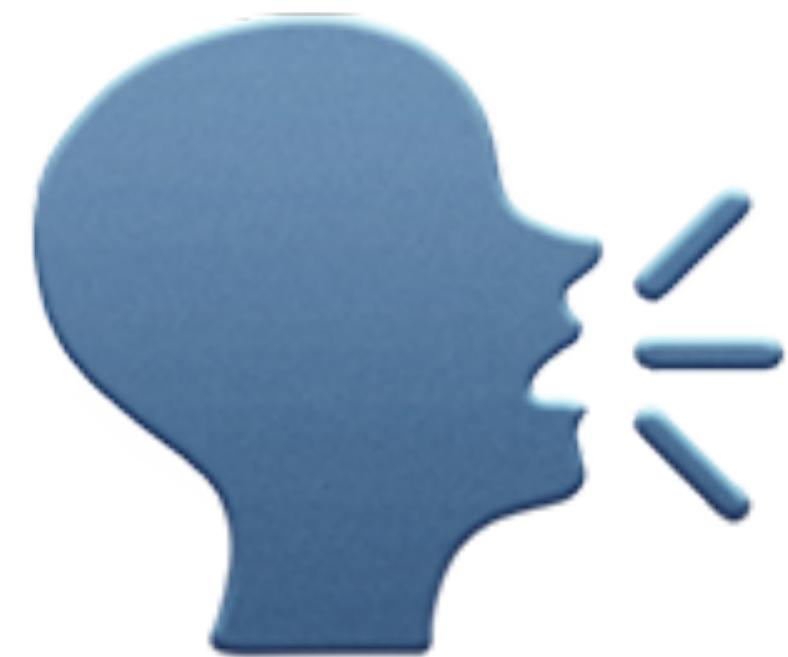
On the right side of the slide, the generated assembly code for x86-64 gcc 11.2 is shown:

```
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    movdqa 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
```

The video player interface includes a progress bar at 23:29 / 50:16, a timestamp of "35", and a name "Victor Ciura" at the bottom right. The video player has standard controls like play/pause, volume, and a DeepMind logo.

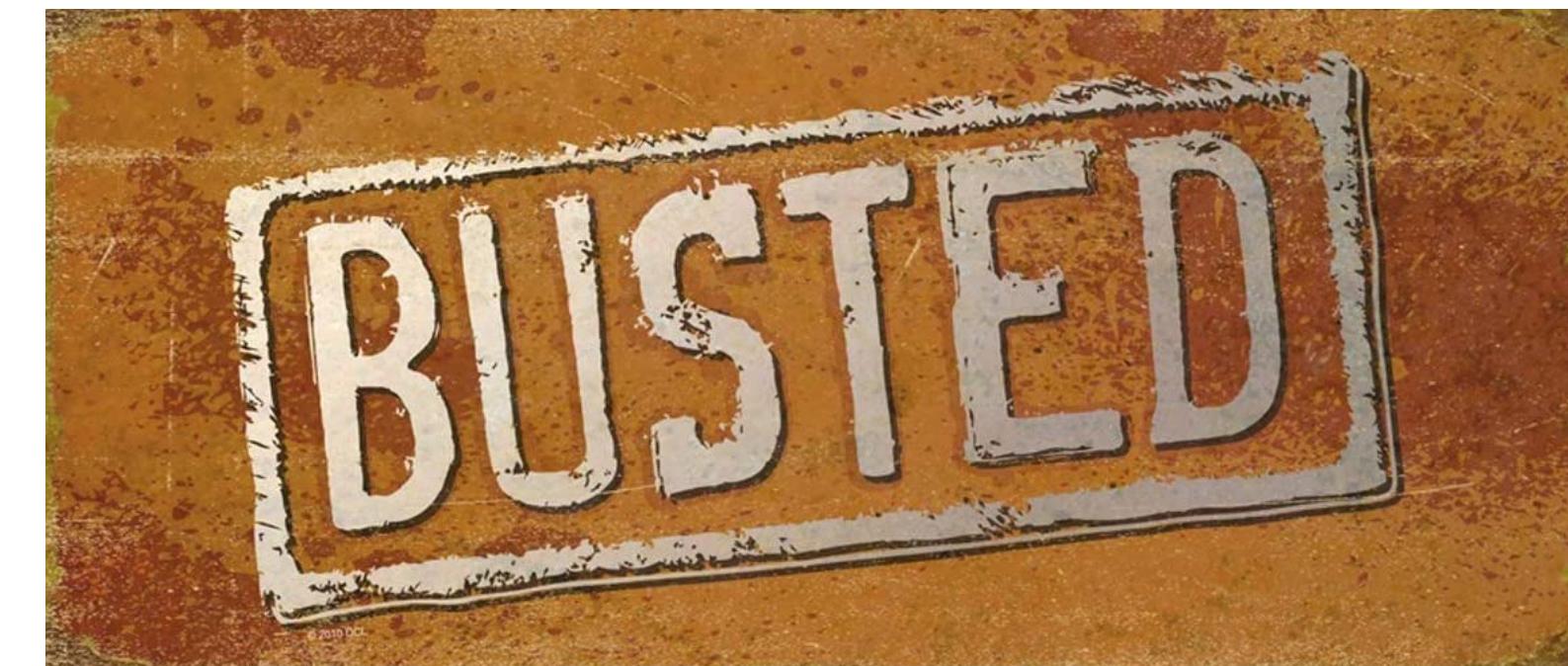
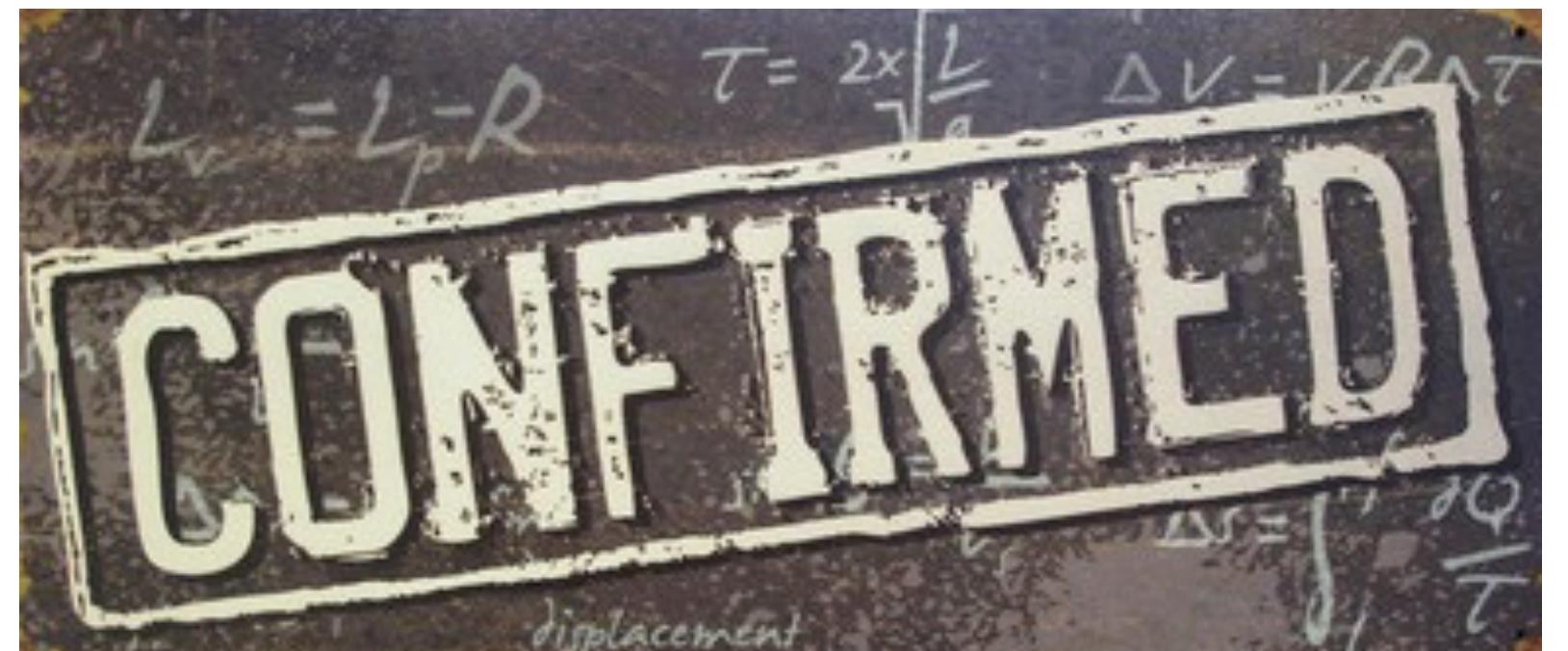
youtube.com/watch?v=ZGgrUhVNsSI

Motivation



I want to instigate a healthy dialog,
so speak up

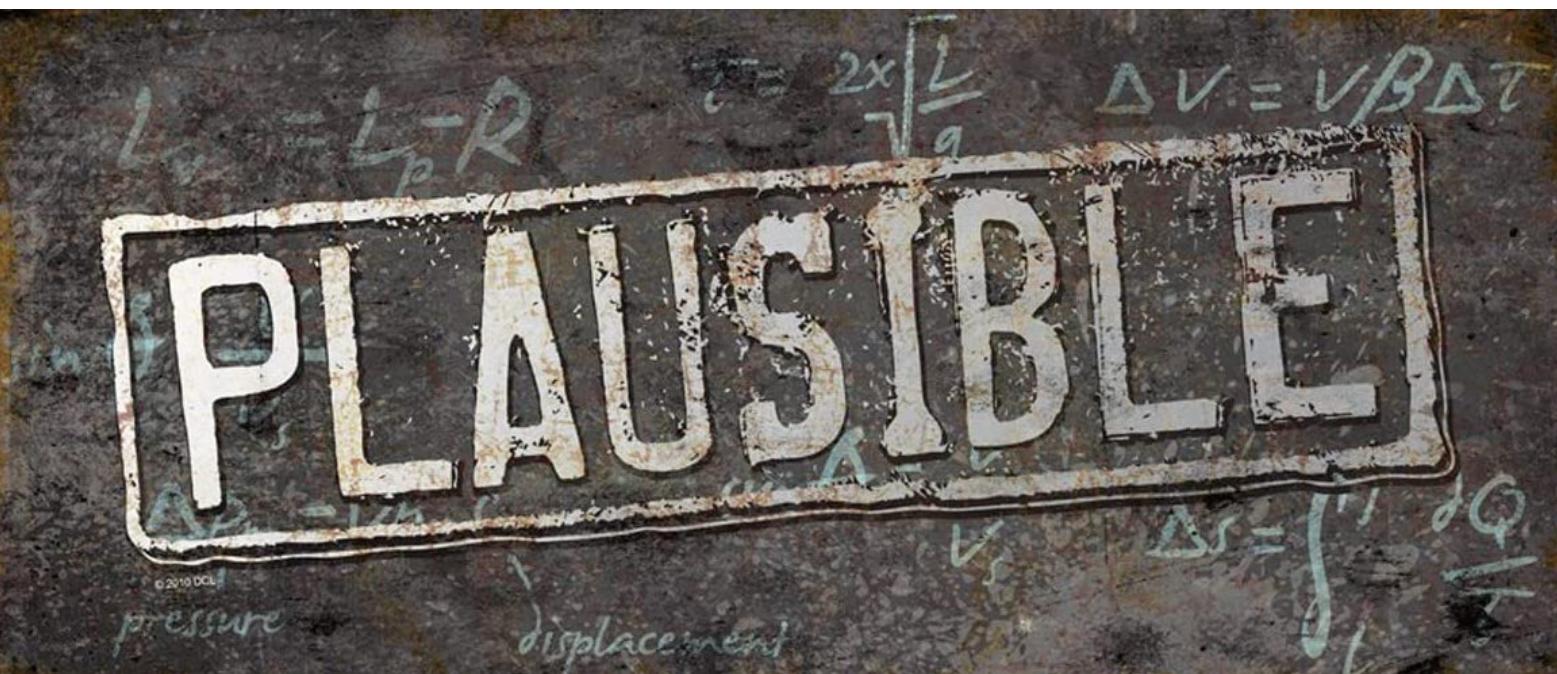
Verdict



Verdict

A programmer's staple response:

"*It depends...*" 😎



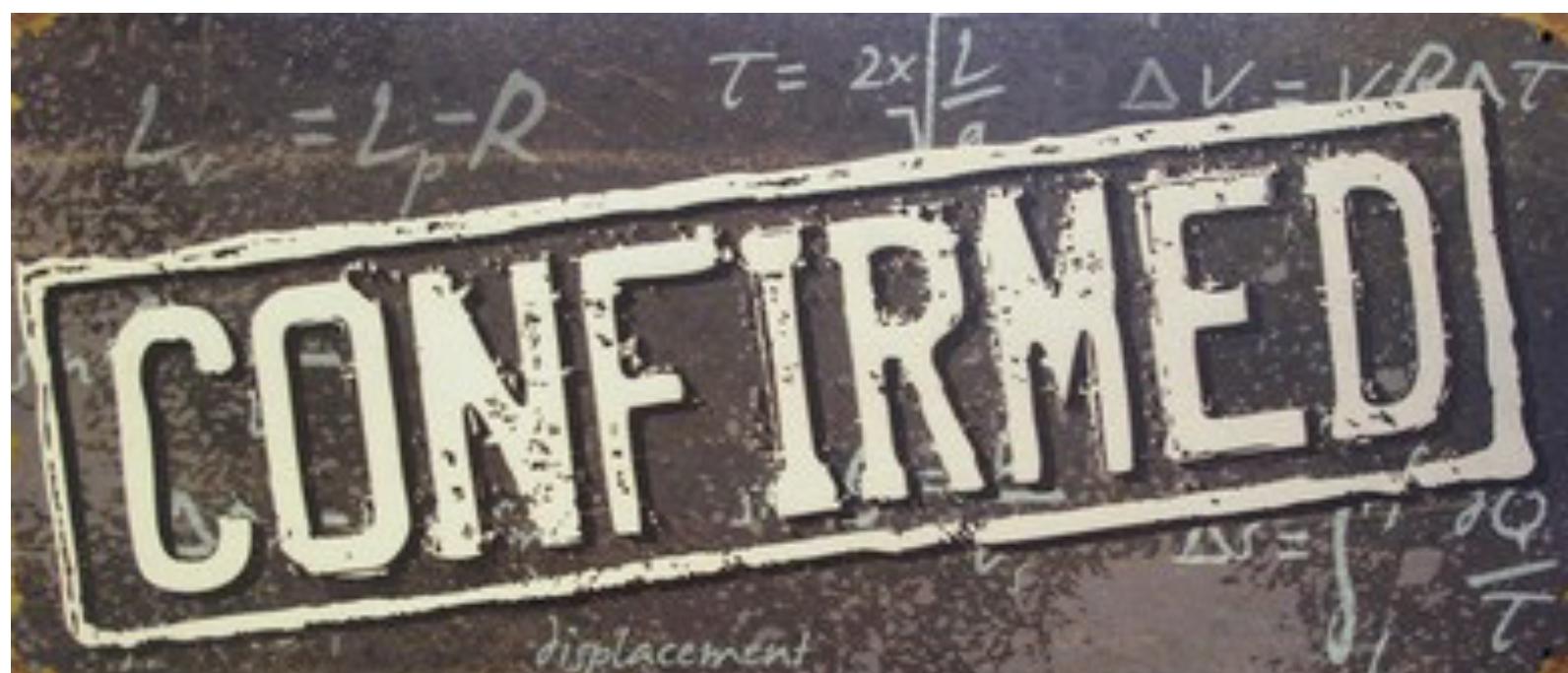
Verdict

Let's test this...



Test Myth

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

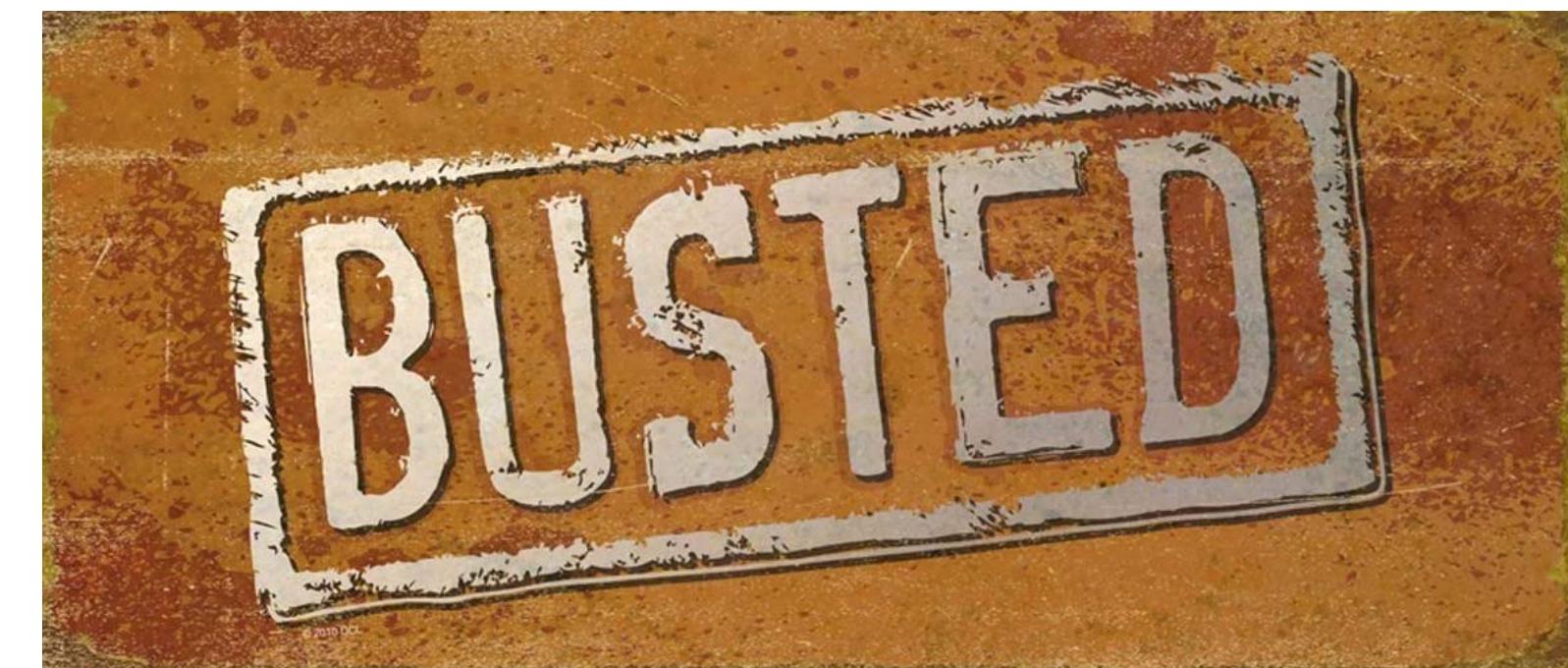


Just kidding 😊

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

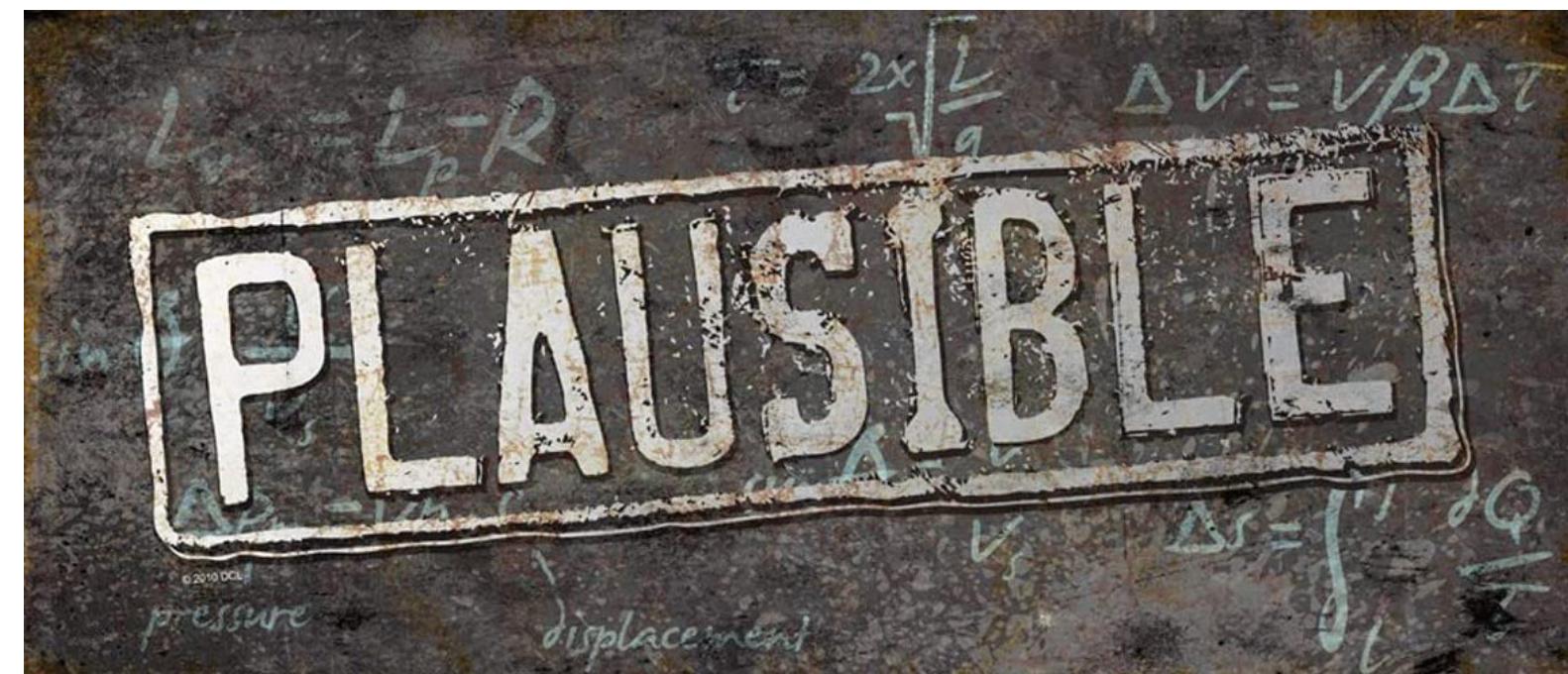
Test Myth

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



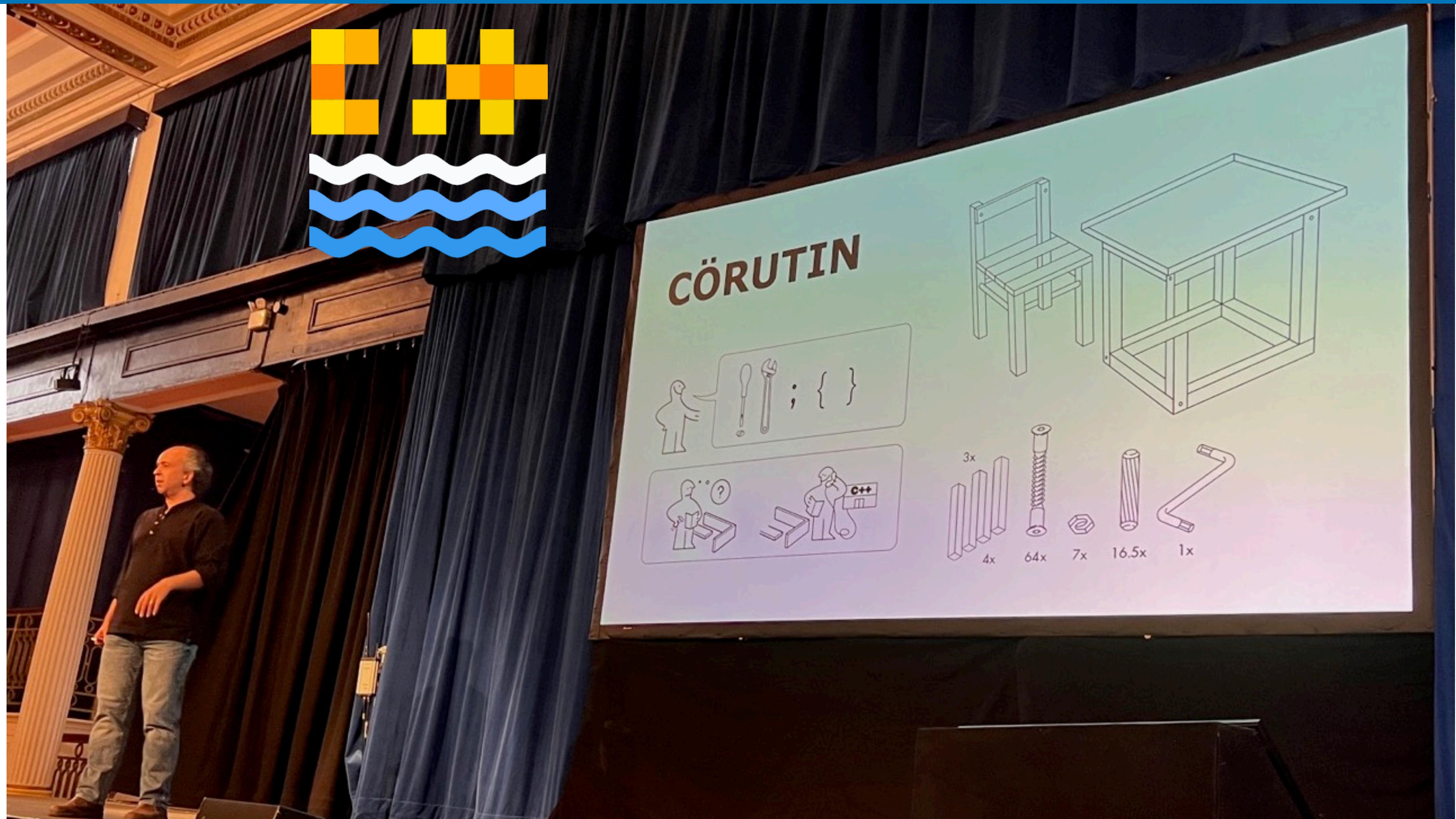
Kinda... 😞

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

```
#include <generator>
```



CÖRUTIN @ C++ On Sea



Verdict

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



Myth #14

C++ is not easily toolable



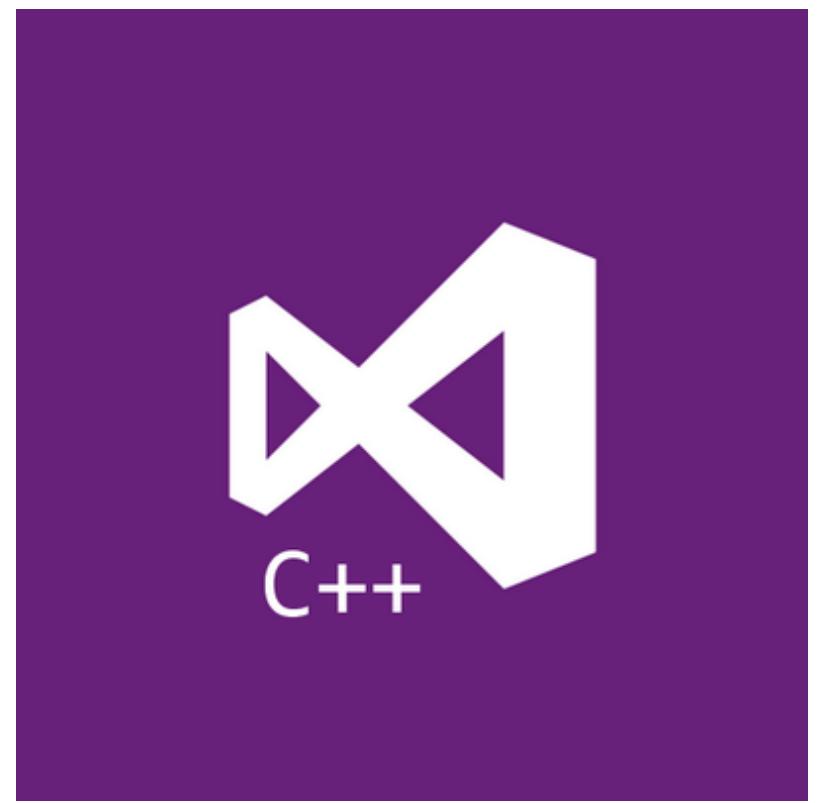
I'm a tool builder



[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 something here>
- 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_using.html
<https://learn.microsoft.com/en-us/cpp/standard-library/checked-iterators>
 - drmemory

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

youtube.com/watch?v=q7Gv4J3FyYE

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

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	filter column...	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_UTILITY_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
```

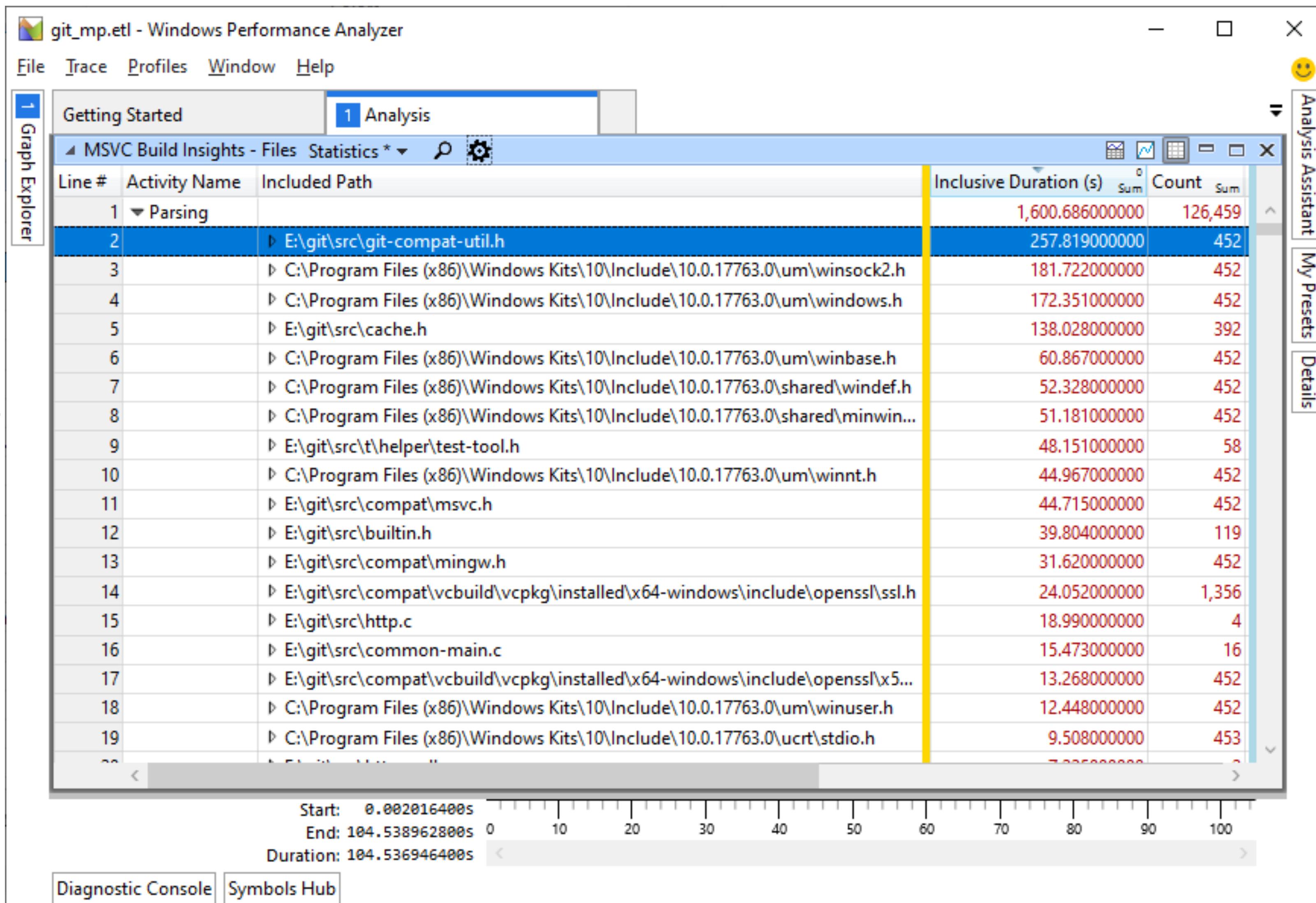
Myth #10



Tooling can help: `vcperf + WPA`

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

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



Myth #10



Tooling can help: Build Insights in Visual Studio

Included Files	Include Tree	File Path	Time [sec, %]	Parse Count	Project
Diagnostics Session: 75.462 seconds	Build: 72.59 seconds	▶ 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\ClrrDeviceWin32.h	0.012 (0.0%)	3	Irrlicht15.
C:\src\irrlicht_pch\source\Irrlicht\ClrrDeviceConsole.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

The image shows a YouTube video player interface. The main title of the video is "IMPROVING COMPILATION TIMES" in large bold letters, with the subtitle "Tools & Techniques" below it. The video is presented by "Vittorio Romeo". The video player includes standard controls like play, pause, volume, and a progress bar showing 0:06 / 1:43:50. The video content itself shows a man with glasses and a beard standing behind a podium, speaking. The background of the video frame features a pink and purple abstract design. The top of the video frame has the text "IMPROVING COMPILATION TIMES" and "TOOLS & TECHNIQUES" on the left, and "ACCU 2023" on the right. The bottom of the video frame has the ACCU.ORG logo and social media links for Vittorio Romeo. The overall theme is a technical presentation at a conference.

IMPROVING COMPILATION TIMES
Tools & Techniques

Vittorio Romeo
✉ mail@vittorioromeo.com
🐦 @supahvee1234

Bloomberg Engineering
TechAtBloomberg.com
Careers

ACCU.ORG

0:06 / 1:43:50

II CC HD ▶ ▶ 🔊 []

ACCU 2023

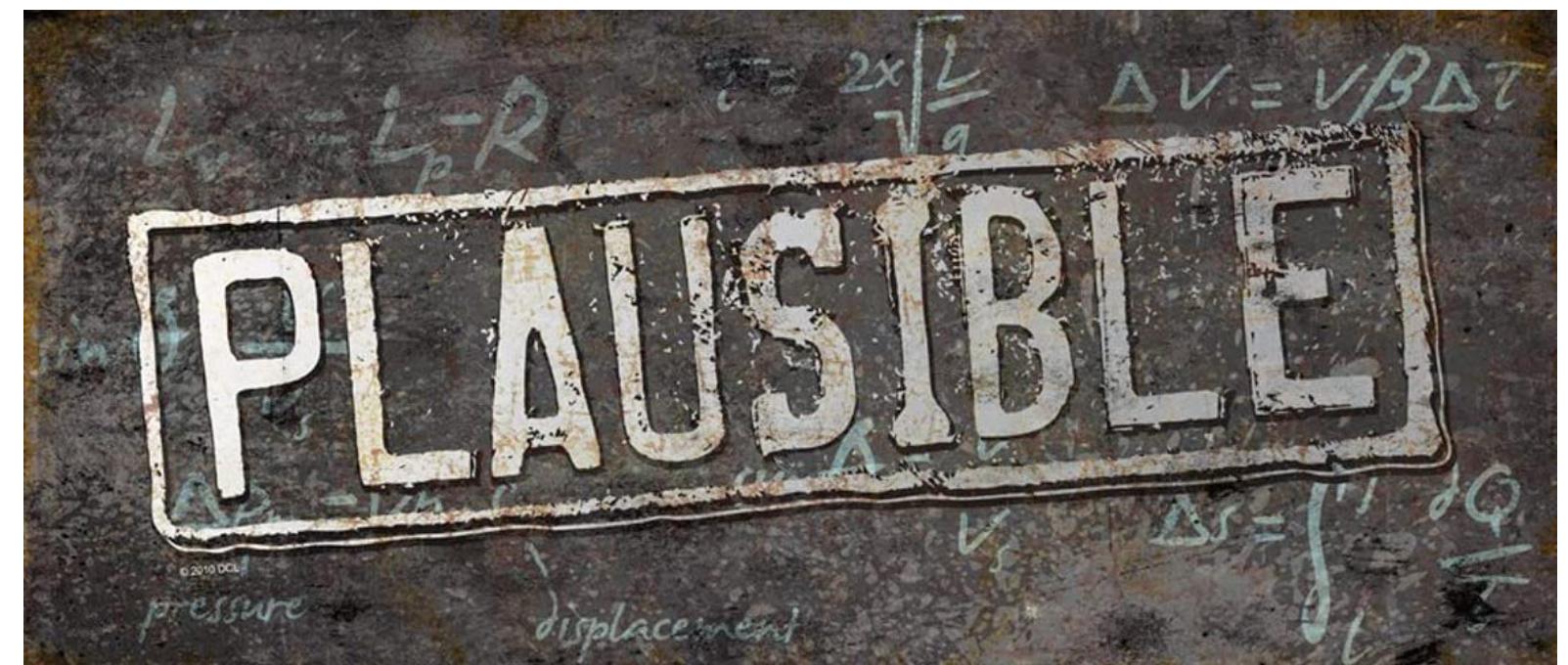
ACCU 2023

Vittorio Romeo

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

The screenshot displays four compiler outputs side-by-side:

- 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):**

```
x86-64 gcc 13.1 -std=c++20 -Wall -Wextra -Wpedantic
A 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
```
- x64 msvc v19.34 (Editor #1):**

```
x64 msvc v19.34 -std:c++latest
A Output... Filter... Libraries Overrides + Add new... Add tool...
1 i$ = 32
2 main PROC
3 $LN3:
4     sub    rsp, 56          ; 00000038H
5     mov    DWORD PTR i$[rsp], 0
6     lea    rax, DWORD 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):**

```
x86-64 clang 14.0.0 -std=c++20 -stdlib=libc++ -Wall -Wextra -Wpedantic
A Output... Filter... Libraries Overrides + Add new... Add tool...
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<int&>(int&)
```

The assembly code from all four compilers includes a call to `std::remove_reference<int&>::type&& std::move<int&>(int&)`. This call is highlighted with a red box in the x86-64 gcc 13.1 and x86-64 clang 14.0.0 outputs.

godbolt.org/z/Pj6xahP9j

Myth #12

The screenshot displays four windows side-by-side, illustrating the assembly output for different compilers given the same C++ code.

C++ Source Code:

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

x86-64 gcc (trunk) (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
```

A yellow sad face icon is placed next to the assembly line at address `rbp-4`, which corresponds to the `call` instruction.

x64 msvc v19.35 (Editor #1)

```
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
```

A blue rounded rectangle highlights the assembly lines from `sub rsp, 24` to `mov eax, DWORD PTR i$[rsp]`.

x86-64 clang 16.0.0 (Editor #1)

```
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
```

A blue rounded rectangle highlights the assembly lines from `mov dword ptr [rbp - 4], 0` to `mov eax, dword ptr [rbp - 8]`.

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

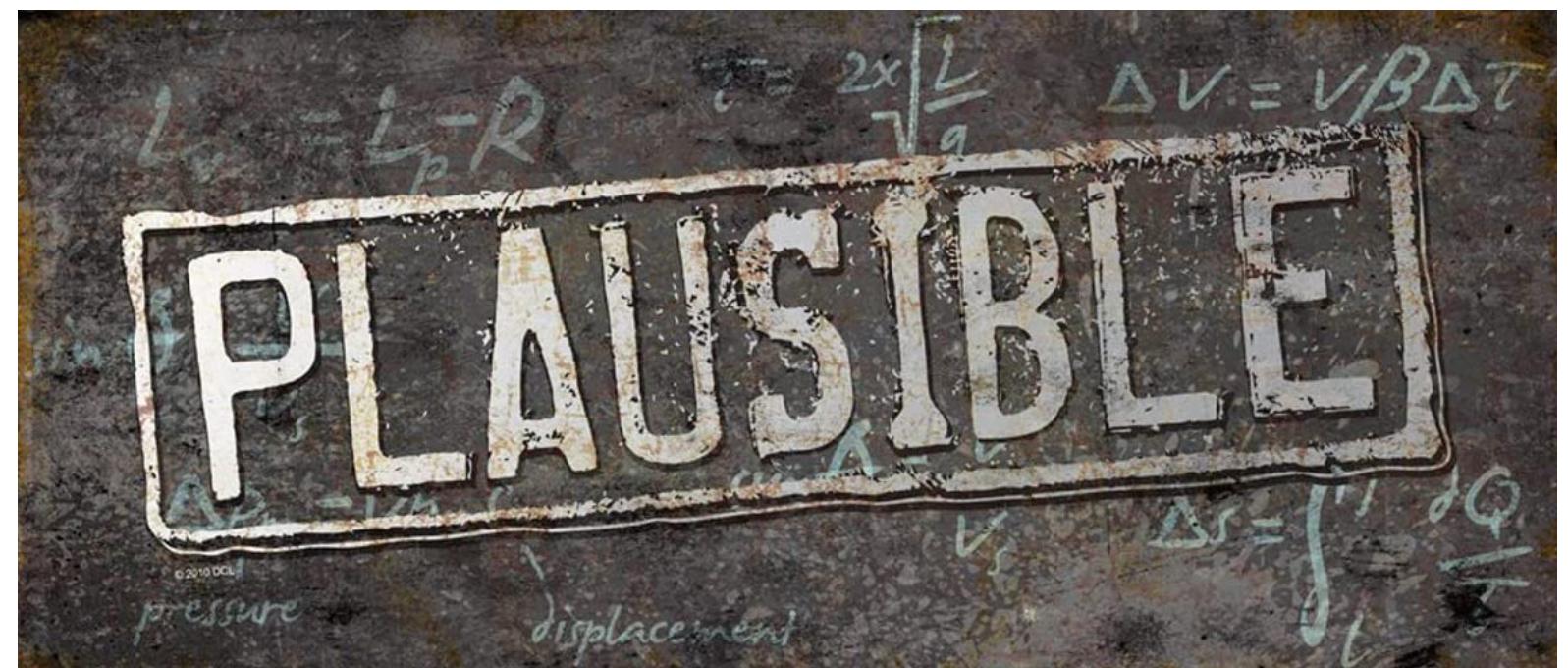
=> extensible to your own such utility functions

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

- ➊ type safety
- ➋ bounds safety
- ➌ lifetime safety
- ➍ initialization safety
- ➎ object access safety
- ➏ thread safety
- ➐ arithmetic safety

Myth #23

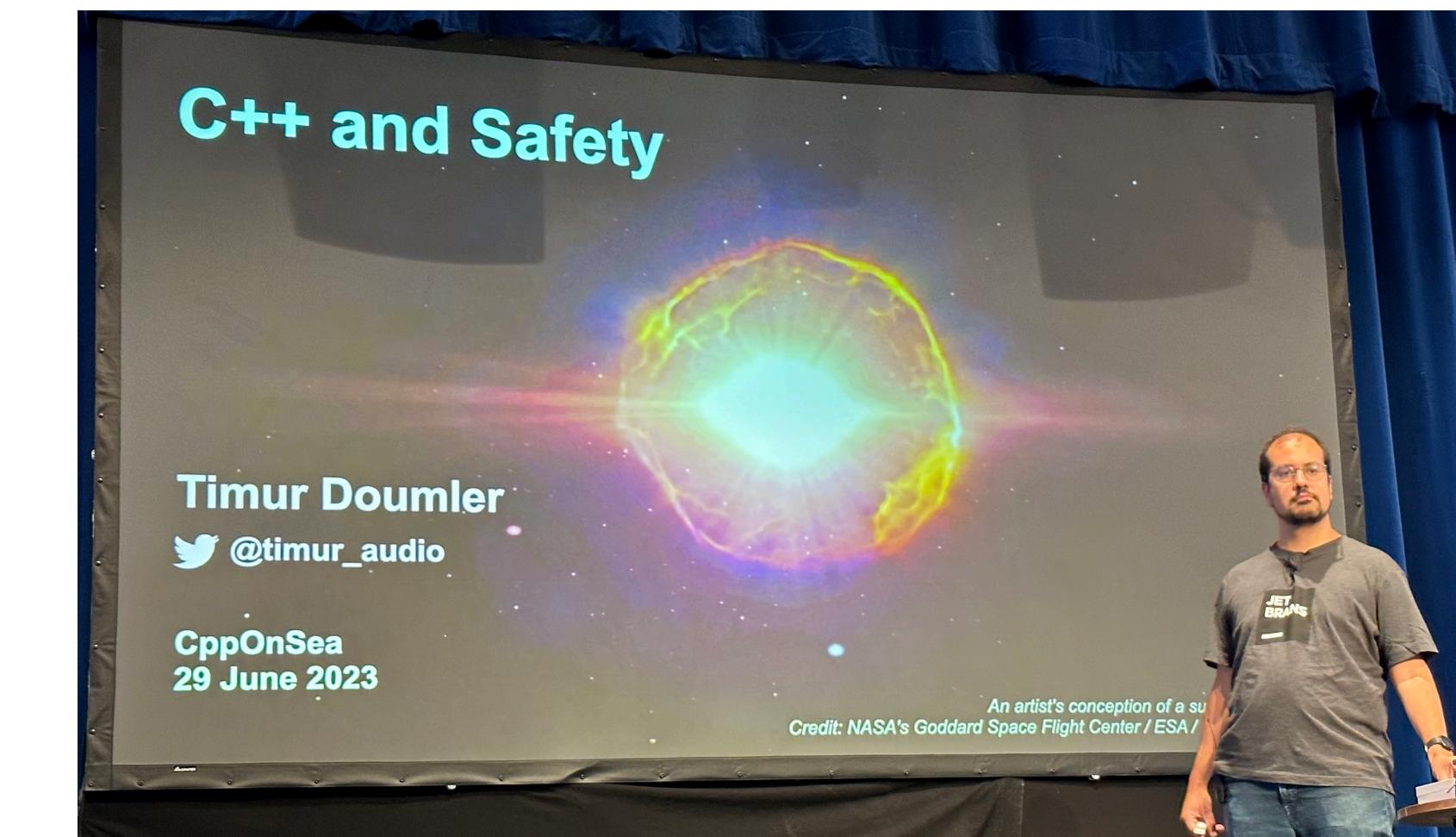
C++ is under attack... and the community is responding 🙌



ACCU 2023

REMOVING NEEDLESS UNDERFINED BEHAVIOR FOR A SAFER C++

ALISDAIR MEREDITH

A slide from Alisdair Meredith's presentation at ACCU 2023. It features a purple gradient background with white text. The title is "REMOVING NEEDLESS UNDERFINED BEHAVIOR FOR A SAFER C++". Below the title is the name "ALISDAIR MEREDITH". On the left, there is a small video frame showing Alisdair speaking.

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

134

A slide titled "What are we going to do?" listing a series of actions to address C++ safety concerns. The slide has a light blue gradient background. The "C++ now" logo is visible in the top right corner.

Myth #23

Tradeoffs need to be made...

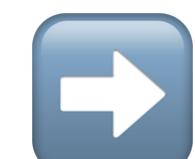


"To UB, or not to UB"

-- *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



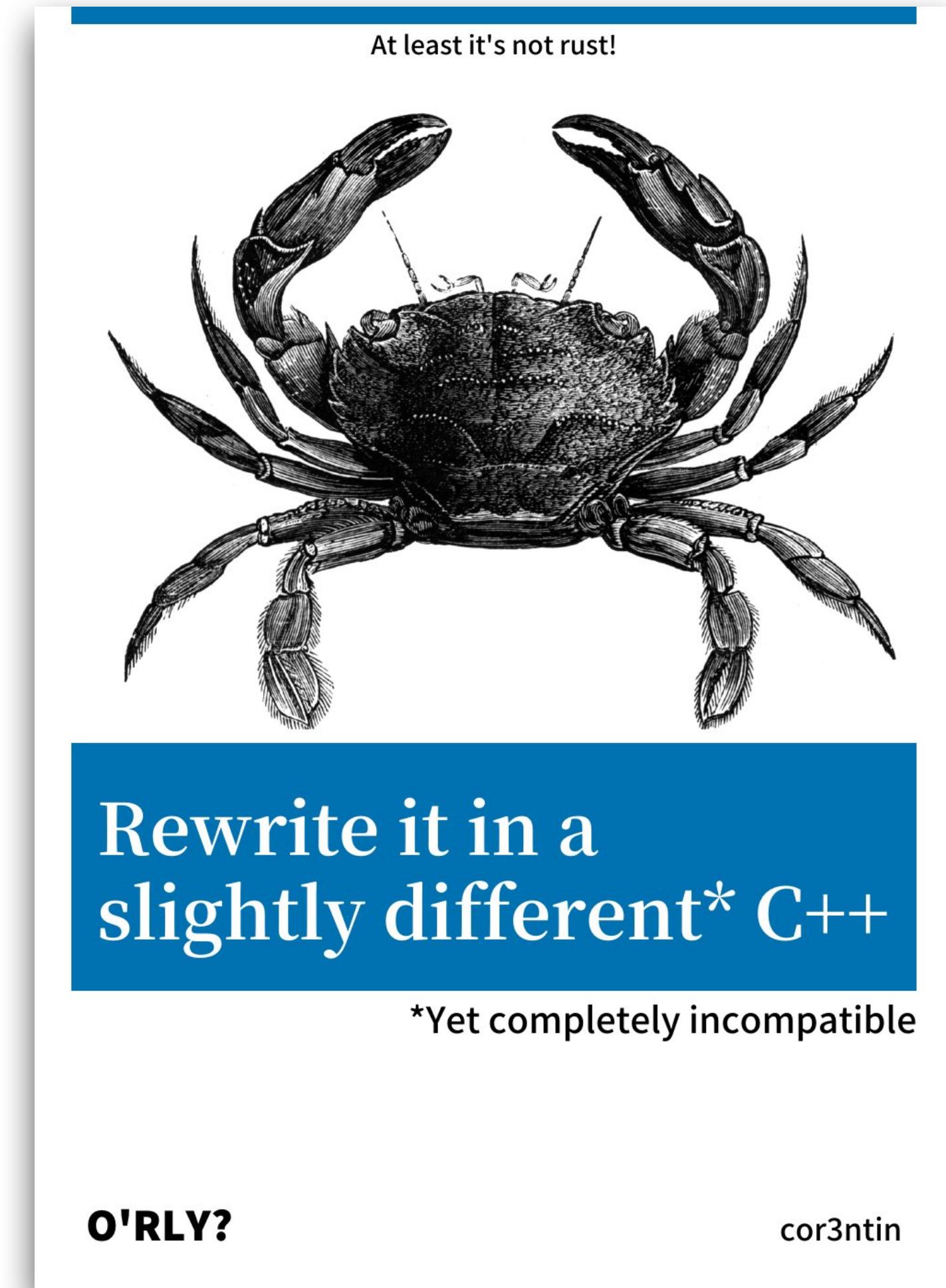
Myth #23

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

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:

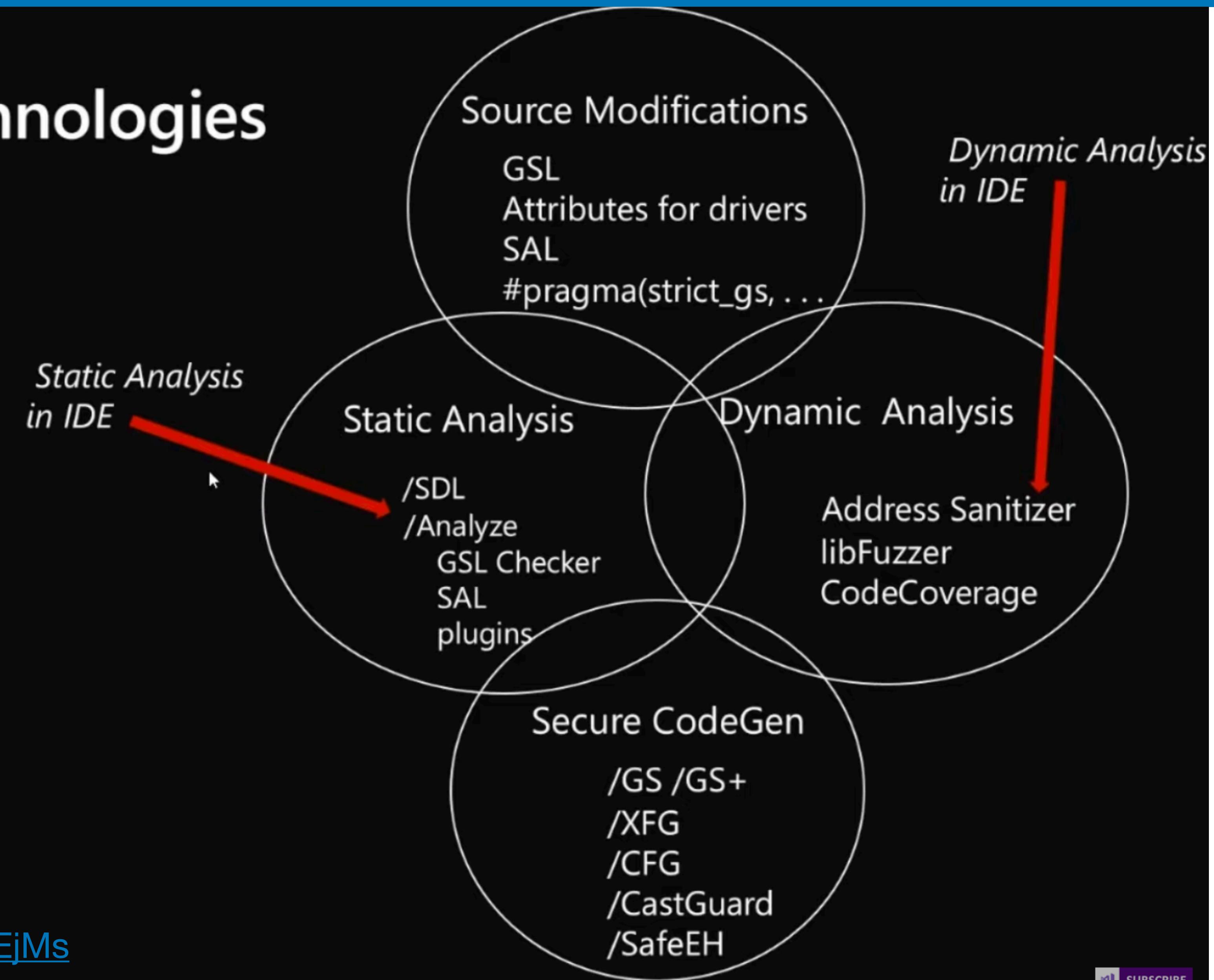
- garbage collector 😱
- dynamic memory analysis (**ASan**)
- statically enforce rules on references: **multiple immutable refs || unique mutable ref**
 - by compiler/language:
 - borrow checker (Rust)
 - mutable value semantics (Val)
 - no direct mutation (Haskell & other pure functional languages)
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

Myth #23

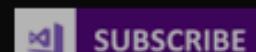
~~AAA (almost always auto)~~

AAA (almost always analyze)

C++ Security Technologies



youtube.com/watch?v=i8_RfDAEjMs



Myth #23

ASan FTW !!!

-fsanitize=address

{ Clang, gcc, MSVC }

youtube.com/watch?v=yJLyANPHNaA

The image shows a video player interface. On the left, there's a thumbnail of Victor Ciura speaking. To the right of the thumbnail, the Cppcon logo is displayed, followed by the text '2020 September 13-18 ONLINE GOING VIRTUAL'. Below this, the title '2020: The Year of Sanitizers?' is shown in large white text. In the bottom right corner of the slide, Victor Ciura's name and title are listed: 'Victor Ciura Principal Engineer'.

Cppcon | 2020 September 13-18
ONLINE GOING VIRTUAL

2020: The Year of Sanitizers?

Victor Ciura
Principal Engineer

@ciura_victor

CAPHYON

Myth #23

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

The slide is part of the "Pure Virtual C++ 2023" series. It features the Microsoft logo in the top right corner. The main title "Pure Virtual C++ 2023" is displayed prominently in white and orange text. Below the title, there is a large, bold, white text block that reads "Address Sanitizer ▶ continue_on_error". To the right of this text block is a circular portrait of Jim Radigan, a man with glasses and a white shirt, set against a dark background with some abstract shapes. At the bottom right, his name "Jim Radigan" is written in white, along with the text "Partner Software Architect, Microsoft".

Myth #23

Static Analysis lifetime annotations for C++

NEW:

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

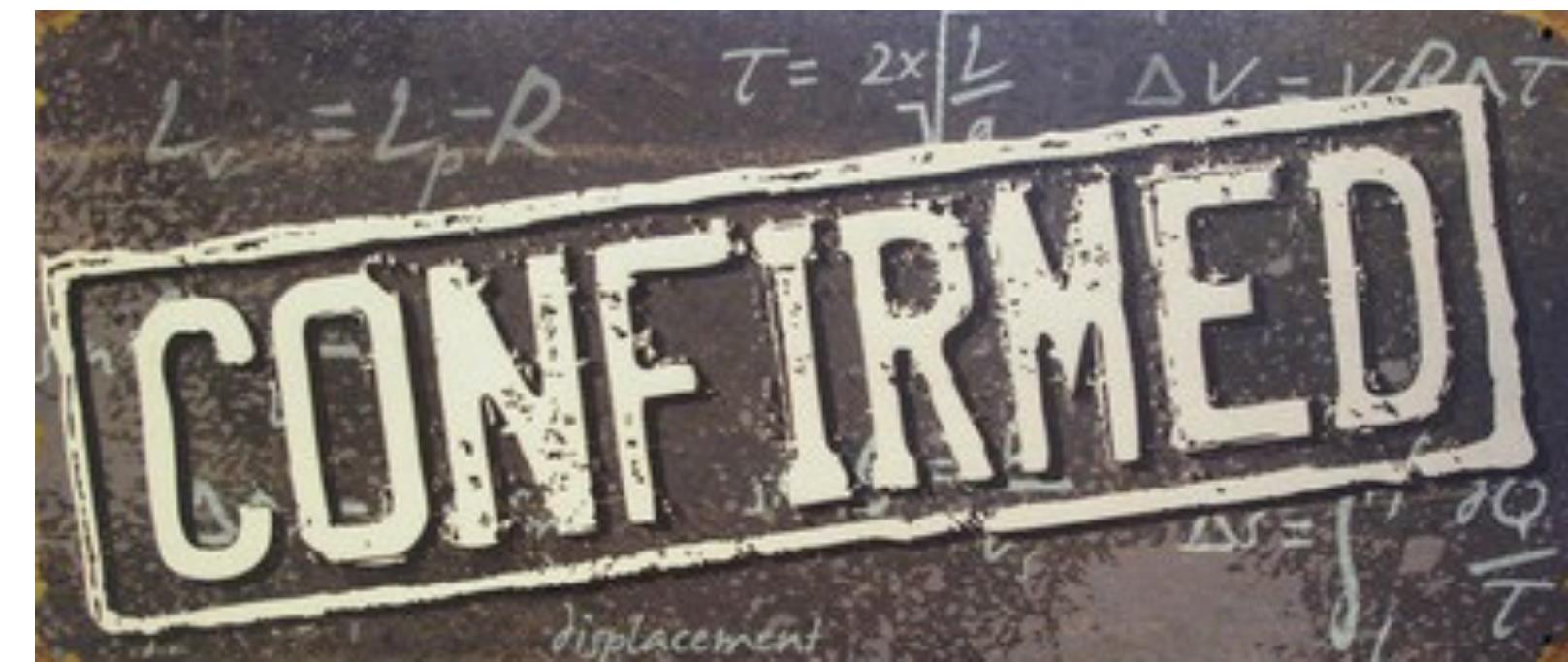
discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377

youtube.com/watch?v=fe6yu9AQIE4

The image shows a video thumbnail for a presentation titled "Pure Virtual C++ 2023". The title is displayed in large white and yellow text on a black background. Below the title, the subtitle "Lifetime Analysis Improvements" is shown in white text. To the right of the text, there is a circular profile picture of a man with short brown hair, wearing a dark shirt, speaking into a microphone. The background of the thumbnail features abstract geometric shapes like triangles and squares in yellow, orange, and white. In the top right corner, the Microsoft logo is visible. At the bottom right, the name "Gabor Horvath" is written in white, along with the title "Software Engineer, Microsoft".

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

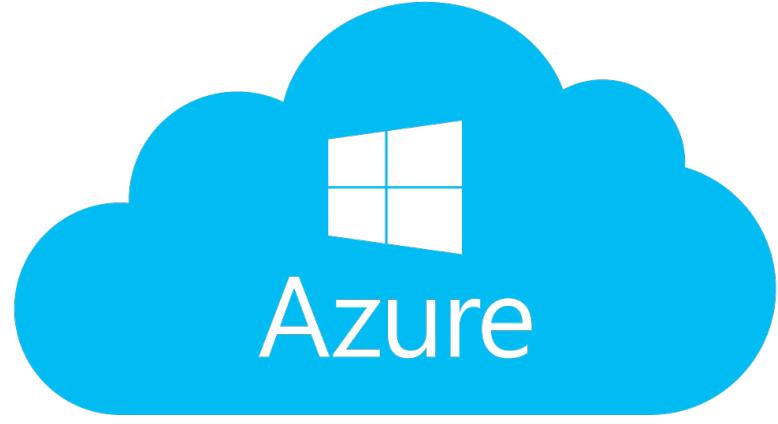
@markrussinovich · [Follow](#)



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

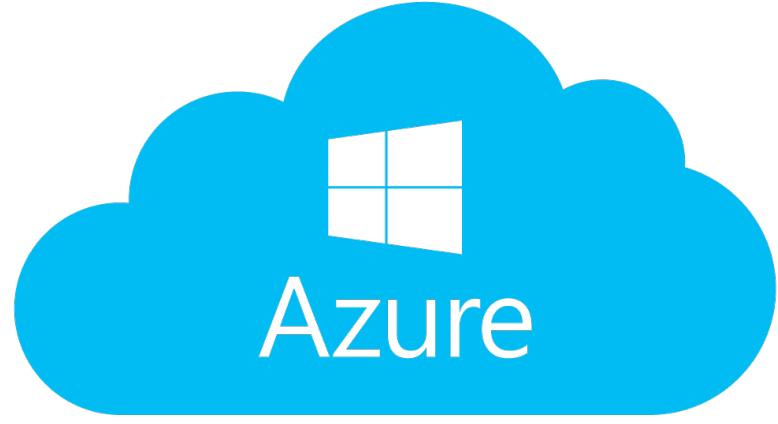




Microsoft Azure security evolution: Embrace secure multitenancy, Confidential Compute, and Rust

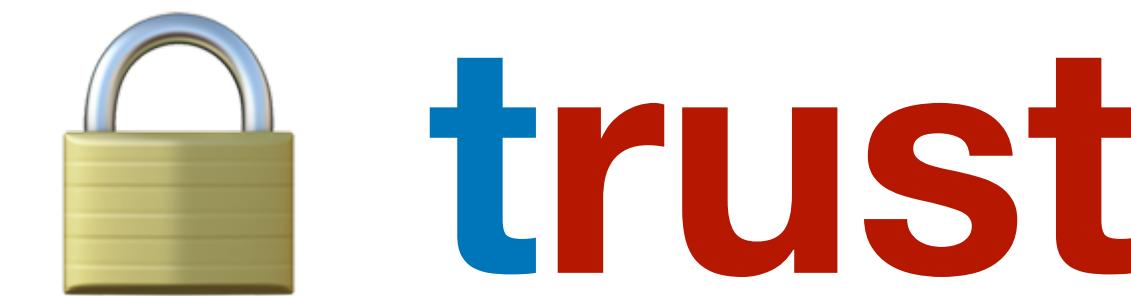
By [Jeffrey Cooperstein](#) Partner Software Architect, Azure Security

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



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

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
_rs = Rust!

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 win32krnl.sys
04/15/2023  09:49  PM          98,304 win32ksgd.sys

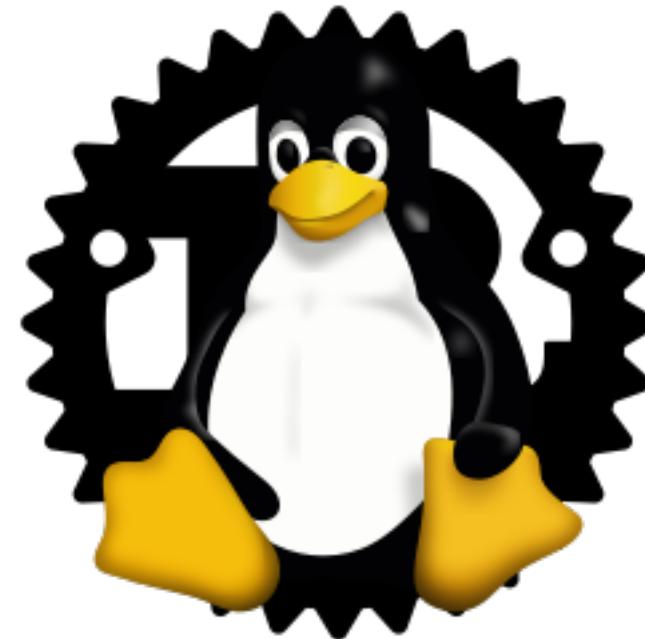
               7 File(s)      8,646,656 bytes
                0 Dir(s)  116,366,049,280 bytes free
```

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.



ubuntu

[wikipedia.org/wiki/Rust_for_Linux](https://en.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



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=8T6CIx-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 began in 2020
 - DWriteCore is now ~152 KLOC of Rust, ~96 KLOC of C++

1st
experiment

youtube.com/watch?v=8T6CIx-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.

```
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 {
    ...
}
```



youtube.com/watch?v=8T6CIx-y2AE&t=2703s

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

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 in completely safe code.

youtube.com/watch?v=8T6CIx-y2AE&t=2703s

Rust Fact vs. Fiction 5 Insights from Google's Rust journey in 2022

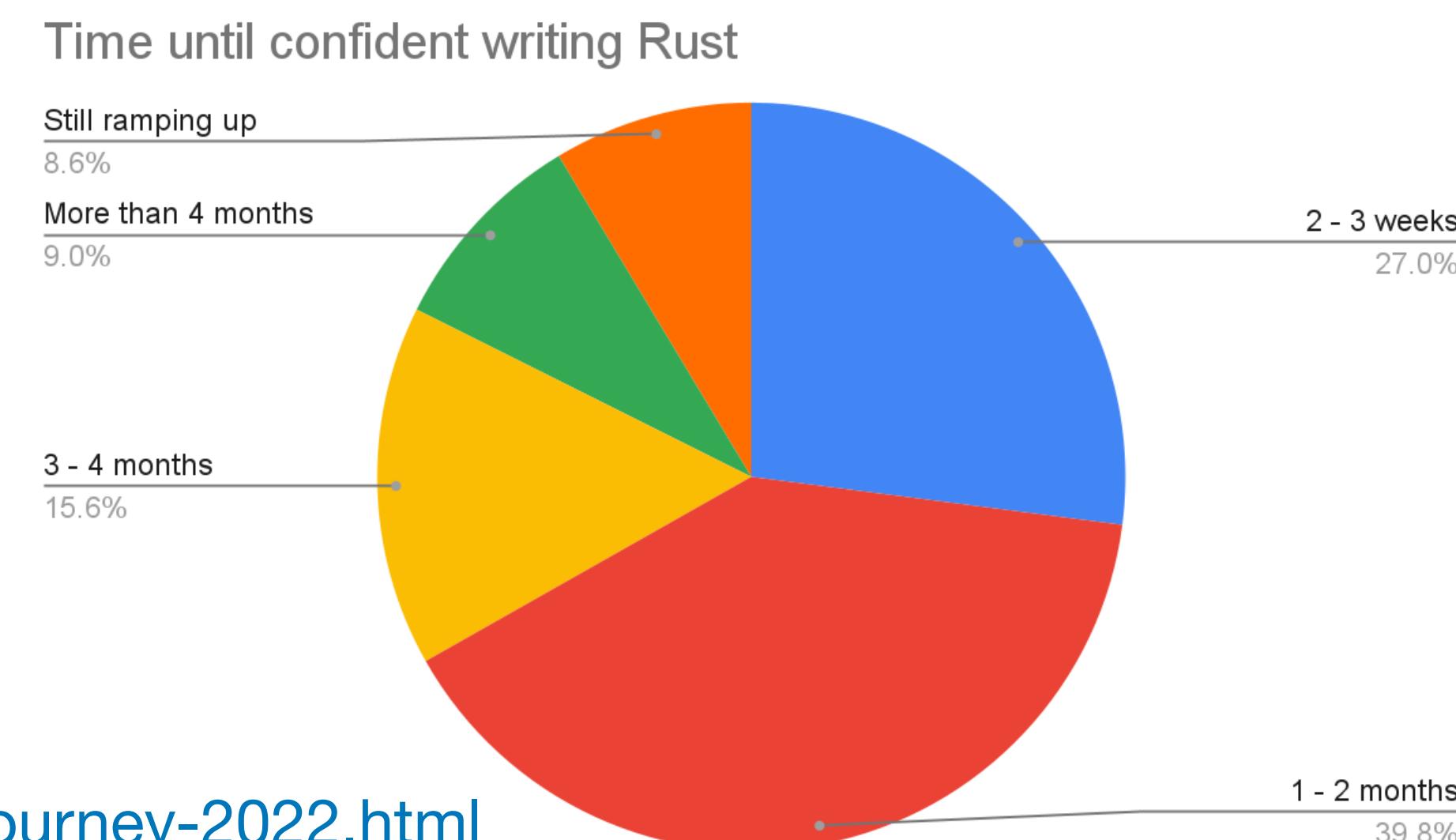
Rumor 1: Rust takes more than 6 months to learn – **Debunked**

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



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/

Myth #38

unsafe { 😱 }

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

The screenshot shows a video player interface with four panes. The top-left pane contains Rust code:

```
1 pub fn foo(x: *mut i32, y: *mut i32) -> i32 {
2     unsafe {
3         *x = 42;
4         *y = 99;
5         *x
6     }
7 }
```

The top-right pane shows the corresponding assembly output from the Rust compiler (rustc 1.63.0):

```
1 example::foo:
2     mov    dword ptr [rdi], 42
3     mov    dword ptr [rsi], 99
4     mov    eax, 42
5     ret
```

The bottom-left pane contains C code:

```
1 int foo(int *x, int *y) {
2     *x = 42;
3     *y = 99;
4     return *x;
5 }
```

The bottom-right pane shows the assembly output from the Clang compiler (x86-64 clang 14.0.0):

```
1 foo:                                # @foo
2     mov    dword ptr [rdi], 42
3     mov    dword ptr [rsi], 99
4     mov    eax, dword ptr [rdi]
5     ret
```

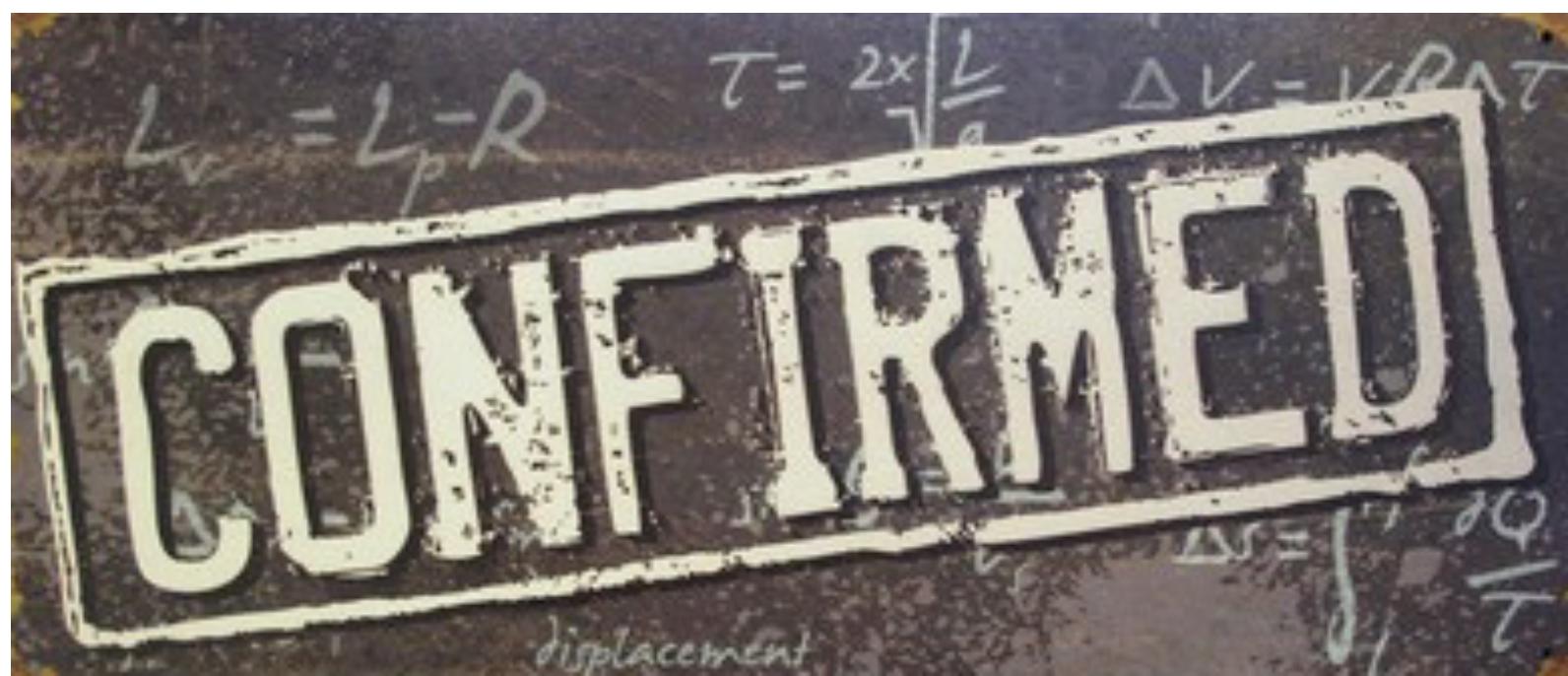
Below the video player, there is a caption: "Unsafe Rust is not C".

Unsafe Rust is not C

youtube.com/watch?v=DG-VLezRkYQ

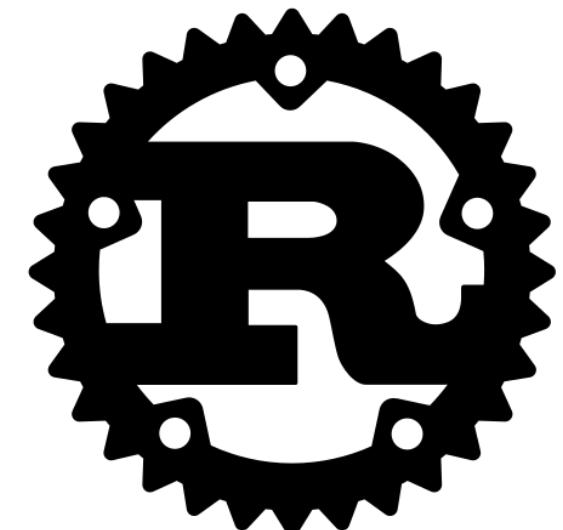
Myth #38

Just rewrite it in Rust 🦀



Myth #6

Successor languages are going to eat our lunch

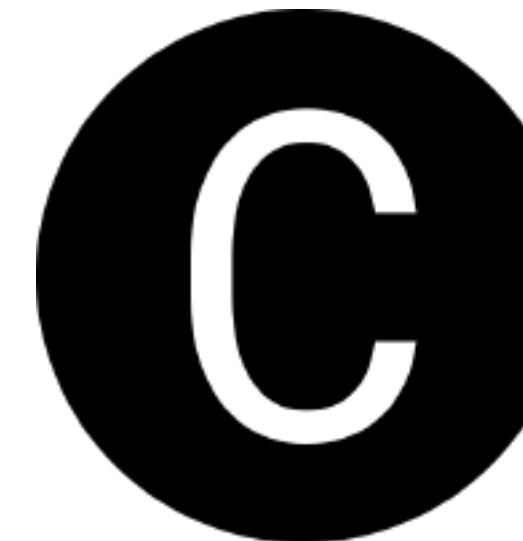


Myth #6



Val 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

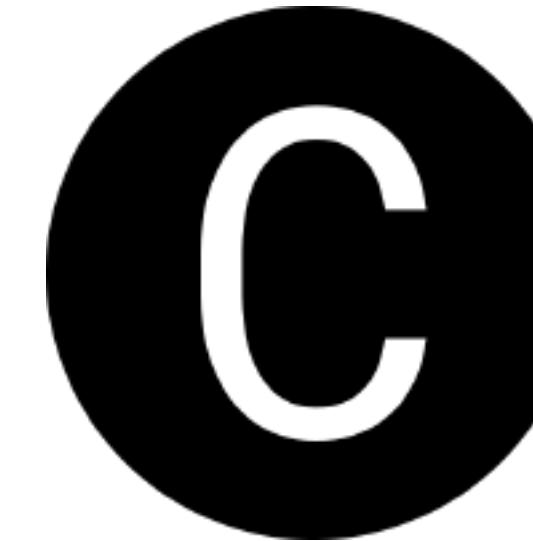
accu.org/journals/overload/30/172/teodorescu/

Myth #6



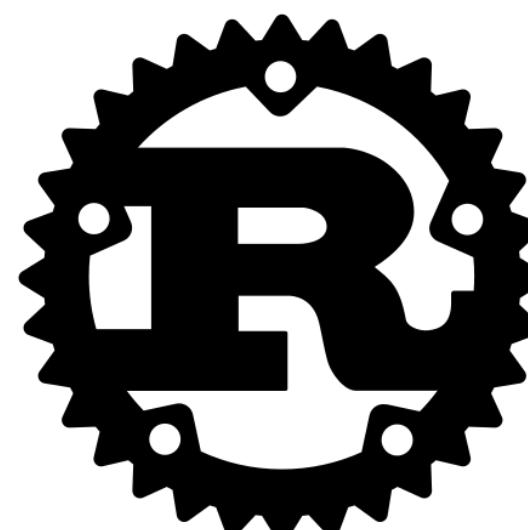
Val aims:

- 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 :)

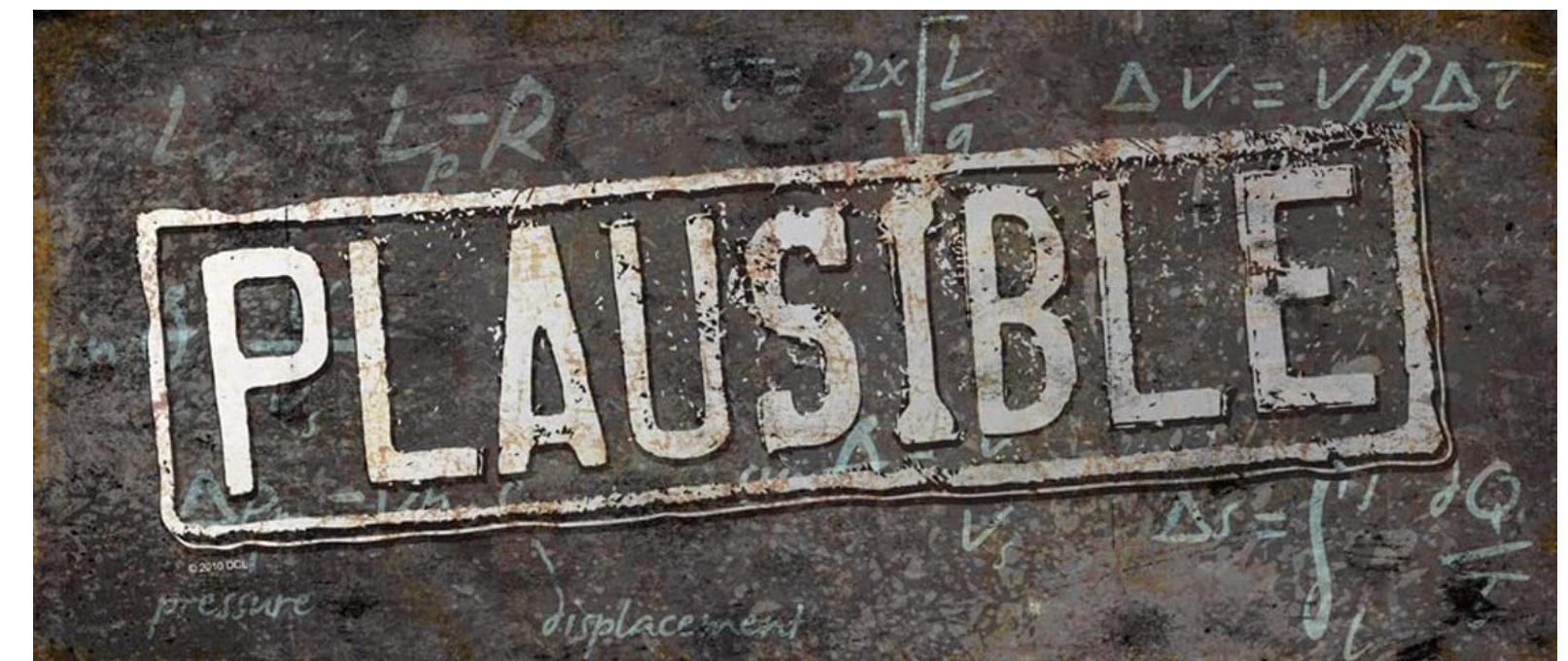


The Year of C++ Successor Languages
-- Lucian Radu Teodorescu

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

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

C++ STANDARD VIEWS

VIDEO SPONSORED BY

think-cell

ACCU
2023

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



Nico Josuttis

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.

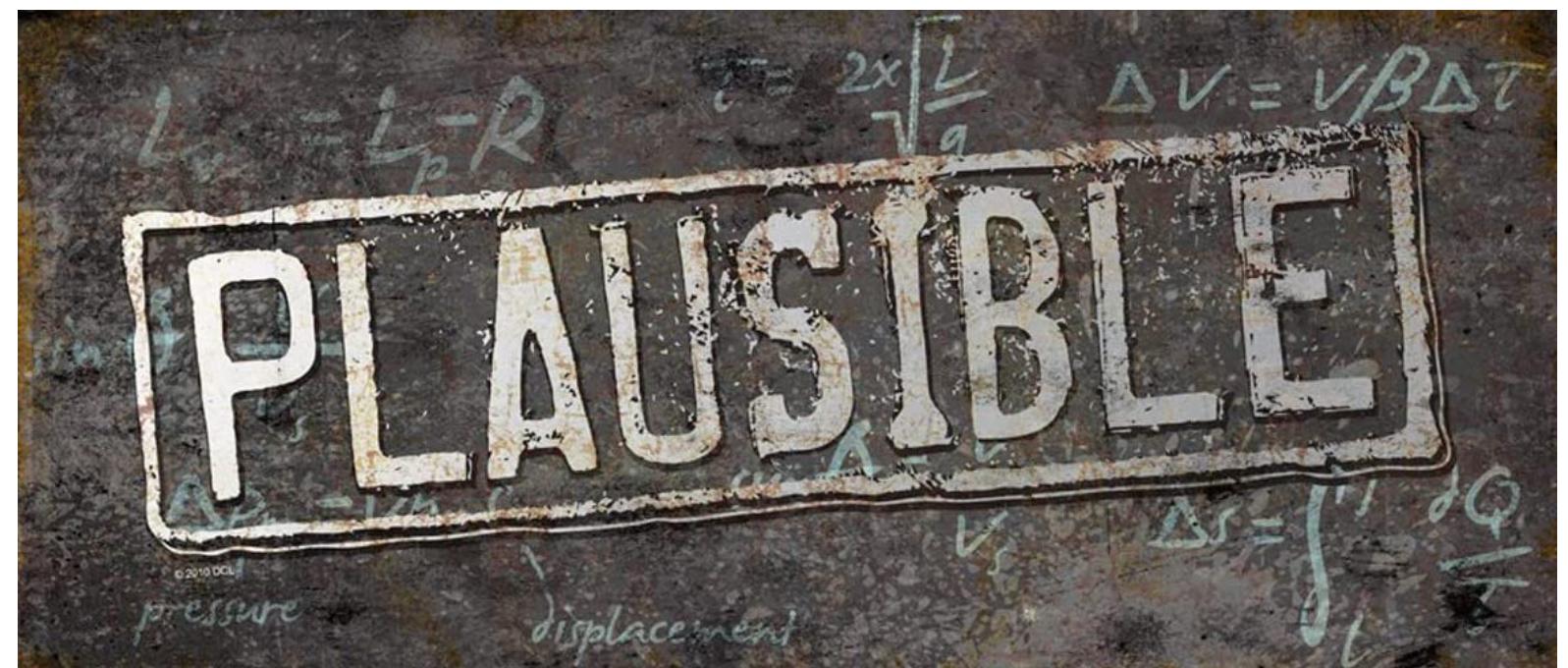
```
// as a shaman:  
for (auto& m : monsters | std::views::filter(isDead) ) {  
    m.resurrect(); // undefined behavior: because no longer dead  
    m.burn(); // OK (because it is still dead)  
}
```

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" 😊

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

CMake Debugger in Visual Studio and VSCode



youtube.com/watch?v=1eVJBEV9NTk

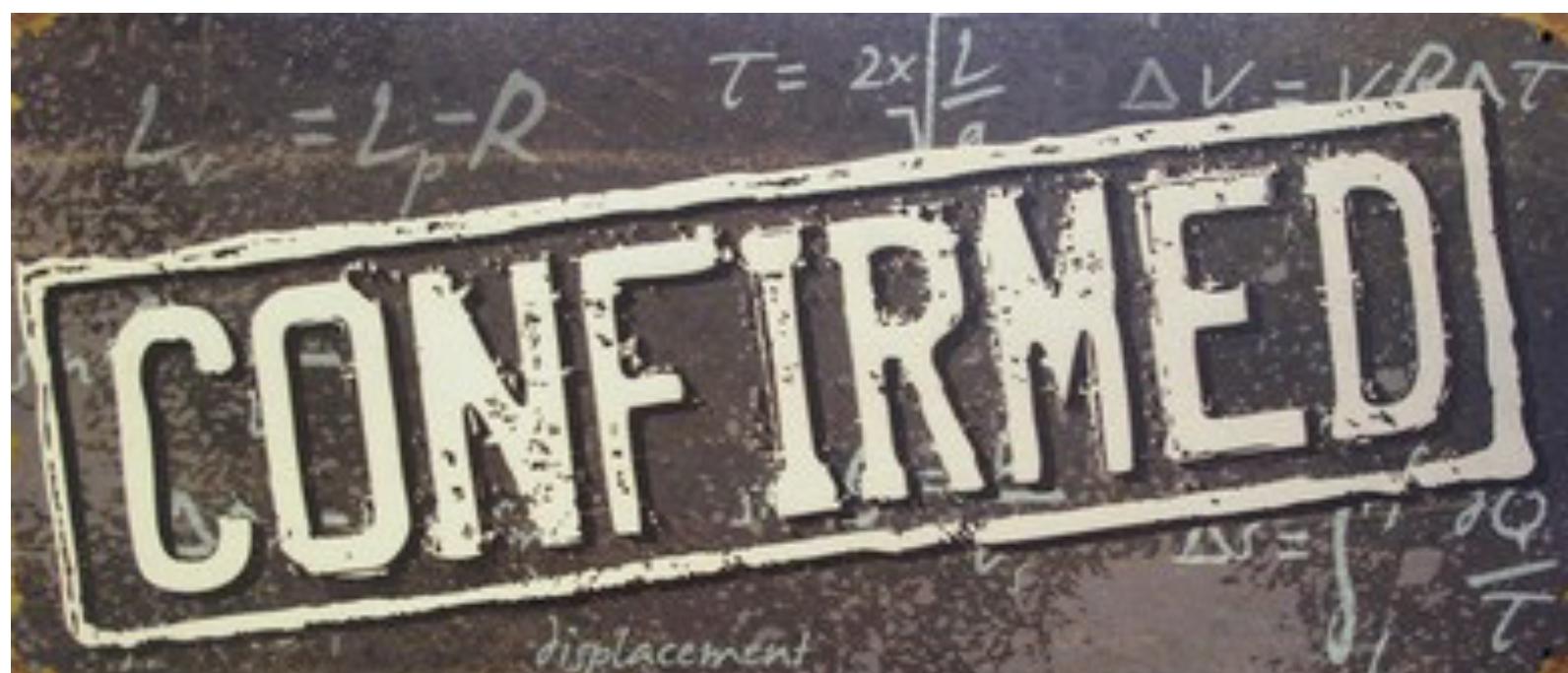
Myth #7

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



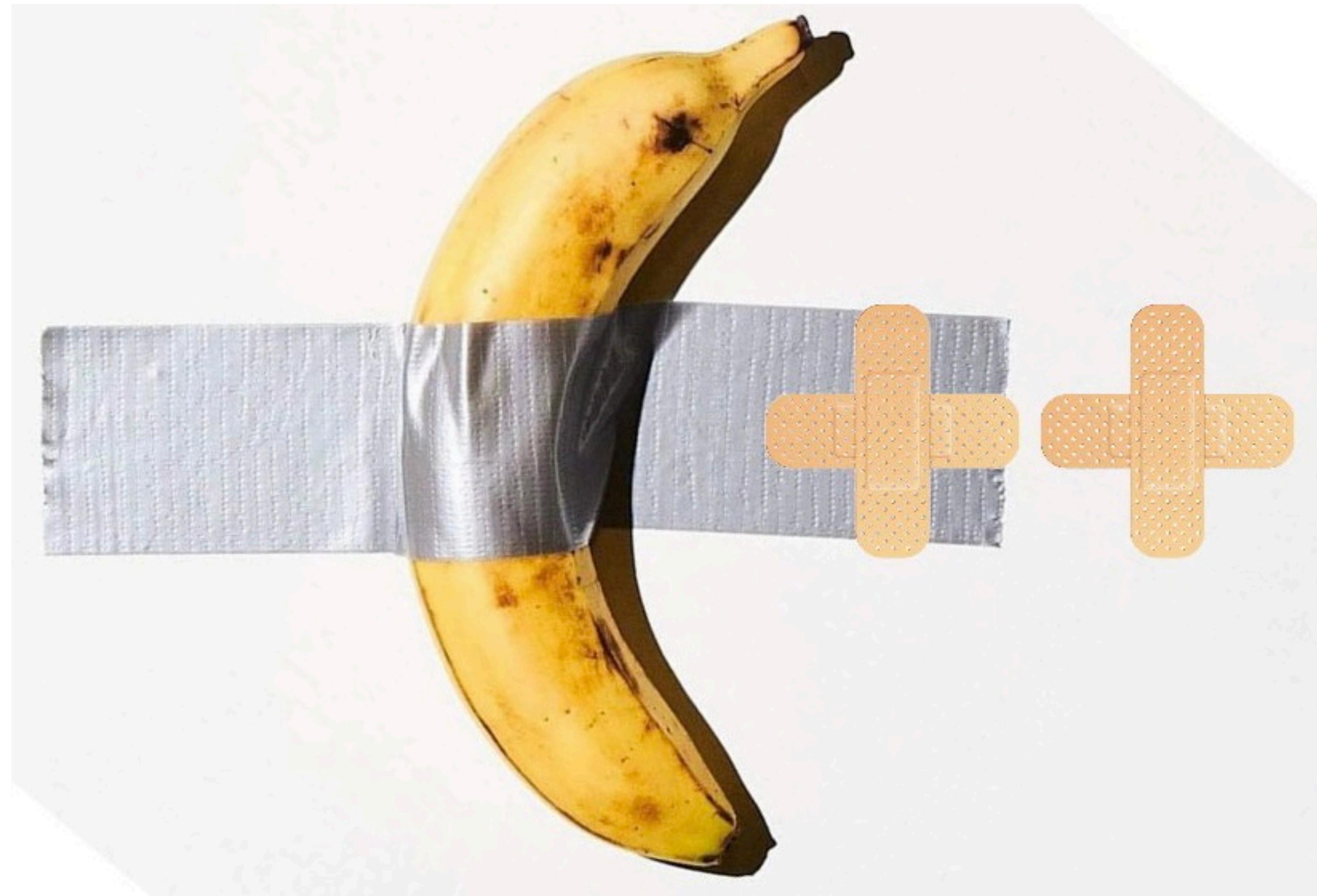
Myth #0

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

Other C++ Mythbusters

Other C++ Mythbusters

The image shows a video player interface. At the top right is the CppCon 2019 logo with the text "The C++ Conference" and "cppcon.org". The main title "Some Programming Myths Revisited" is displayed in large white text on an orange background. Below the title, the speaker's name "Patrice Roy" is shown, followed by two email addresses: "Patrice.Roy@USherbrooke.ca" and "Patrice.Roy@clg.qc.ca". The text "CeFTI, Université de Sherbrooke" and "Collège Lionel-Groulx" is also present. The video progress bar indicates it is at 0:25 / 1:00:50. The bottom right corner of the video frame contains the text "Video Sponsorship Provided By: ansatz". The video frame has rounded corners and a thin black border.

Some Programming Myths Revisited

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

0:25 / 1:00:50

II CC HD ansatz

© AURORA

Some Programming Myths Revisited - Patrice Roy - CppCon 2019

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/



June 2023

 @ciura_victor
 @ciura_victor@hachyderm.io

Victor Ciura
Principal Engineer
Visual C++

