



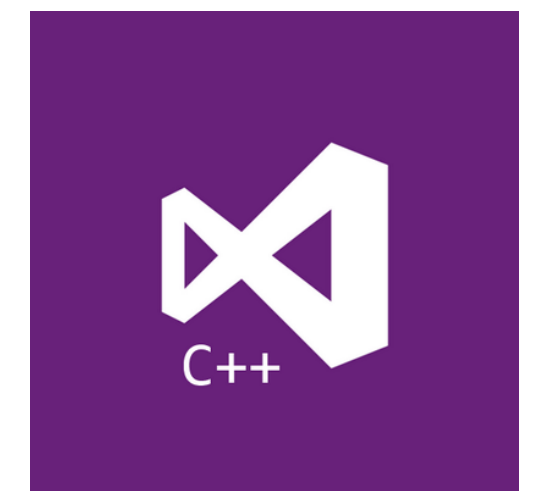
Myths, Dogma and Practice

~2023();

 @ciura_victor

 @ciura_victor@hachyderm.io

Victor Ciura
Principal Engineer
Visual C++



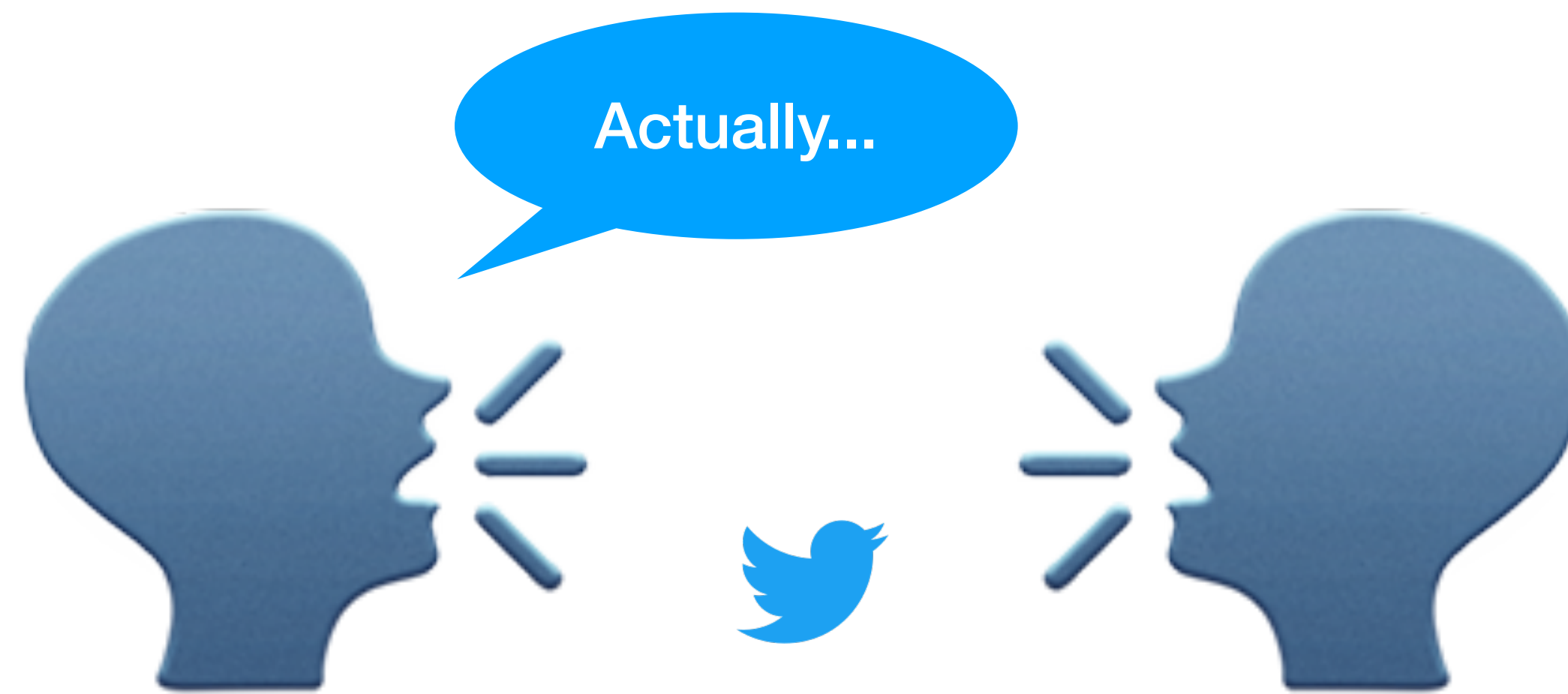


Do ask questions as we go along

Comments are welcome, too

Actually, ...

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



Your opinion...



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

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

-- David Fowler

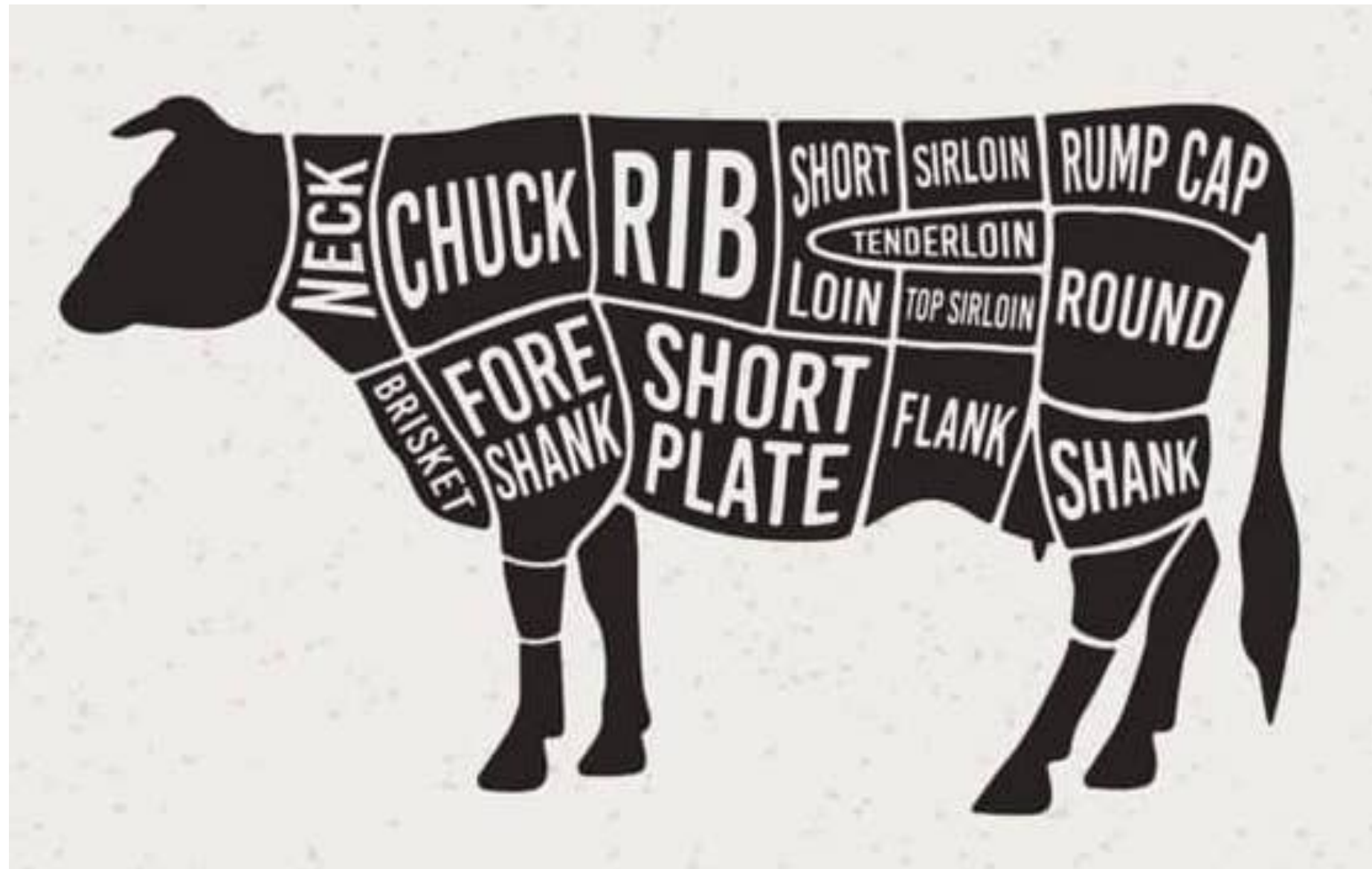
We're Different

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

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

We're Different

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



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

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

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

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



StackOverflow

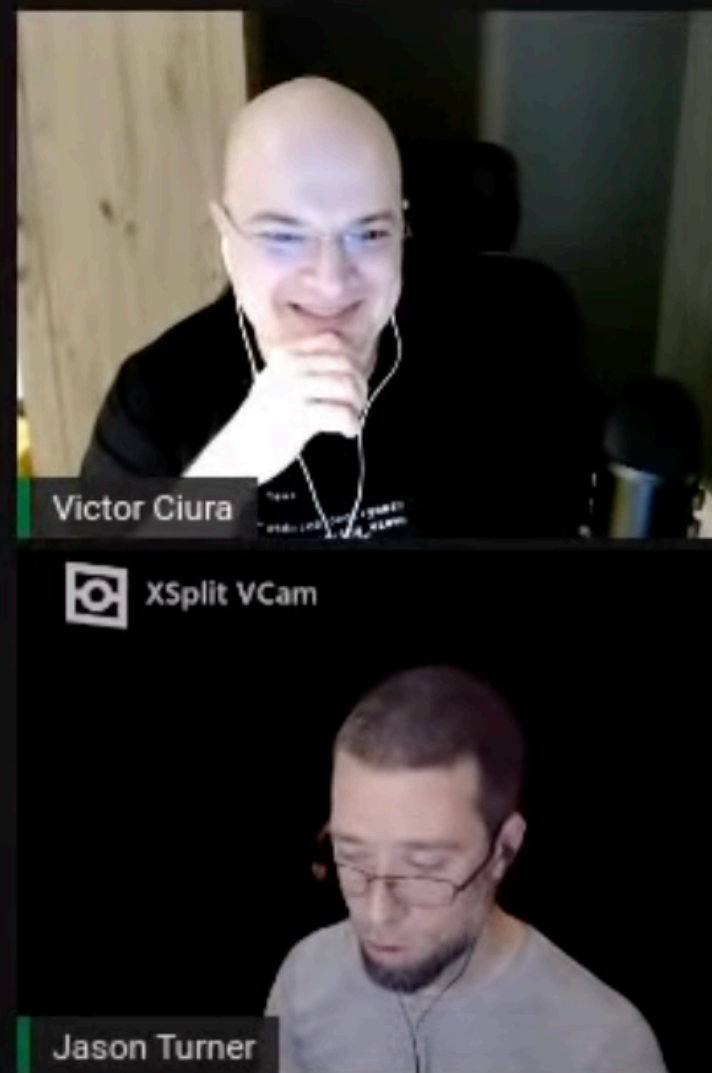


How it started...

Mythbusting with Jason - unscripted improv (Pandemic edition)

21k views

youtube.com/watch?v=Bu1AEze14Ns



```
COMPILER EXPLORER
C++ source #1 x
x86-64 gcc (trunk) (Editor #1, Compiler #1) C++ x
x86-64 gcc (trunk) -std=c++20 -Wpedantic -Wall -Wextr.
Output... Filter... Libraries + Add new... Add tool...
.L5:
7
8 lea rdx, [rdi+16]
9 mov BYTE PTR [rdi+26], 100
10 movabs rcx, 8022916924116329800
11 mov QWORD PTR [rdi], rdx
12 mov edx, 27762
13 mov QWORD PTR [rdi+16], rcx
14 mov WORD PTR [rdi+24], dx
15 mov QWORD PTR [rdi+8], 11
16 mov BYTE PTR [rdi+27], 0
17 mov BYTE PTR [rdi+32], 1
18 ret
19 get_optional_string_size(bool):
20 cmp dil, 1
21 sbb rax, rax
22 or rax, 11
23 ret
Output (0/0) x86-64 gcc (trunk) i - 3231ms (3198646)
#1 with x86-64 gcc (trunk) x
Wrap lines
Compiler returned: 0
```

```
1 #include <fmt/format.h>
2
3 #include <array>
4 #include <cstdlib>
5 #include <optional>
6
7 // std::optional<>?
8
9 std::optional<std::string> get_optional_value(const bool something)
10 {
11     if (something) {
12         return "Hello World";
13     } else {
14         return std::nullopt;
15     }
16 }
17
18 std::size_t get_optional_string_size(const bool something) {
19     const auto optional_str = get_optional_value(something);
20     if (optional_str) {
21         return optional_str->size();
22     } else {
23         return std::string::npos;
24     }
25 }
26
27
```

32:48 / 2:03:29

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

C++ MythBusters 2022

Myth #24

COMPILER EXPLORER

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

no more SSO

compiler still sees through it and inlines it

```
35 sub    rsp, 56
36 mov    edi, 24
37 lea   rax, [rsp+16]
38 mov   QWORD PTR [rsp], rax
39 call  operator new(unsigned long)
40 mov   esi, 24
41 mov   BYTE PTR [rsp+32], 0
42 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
```

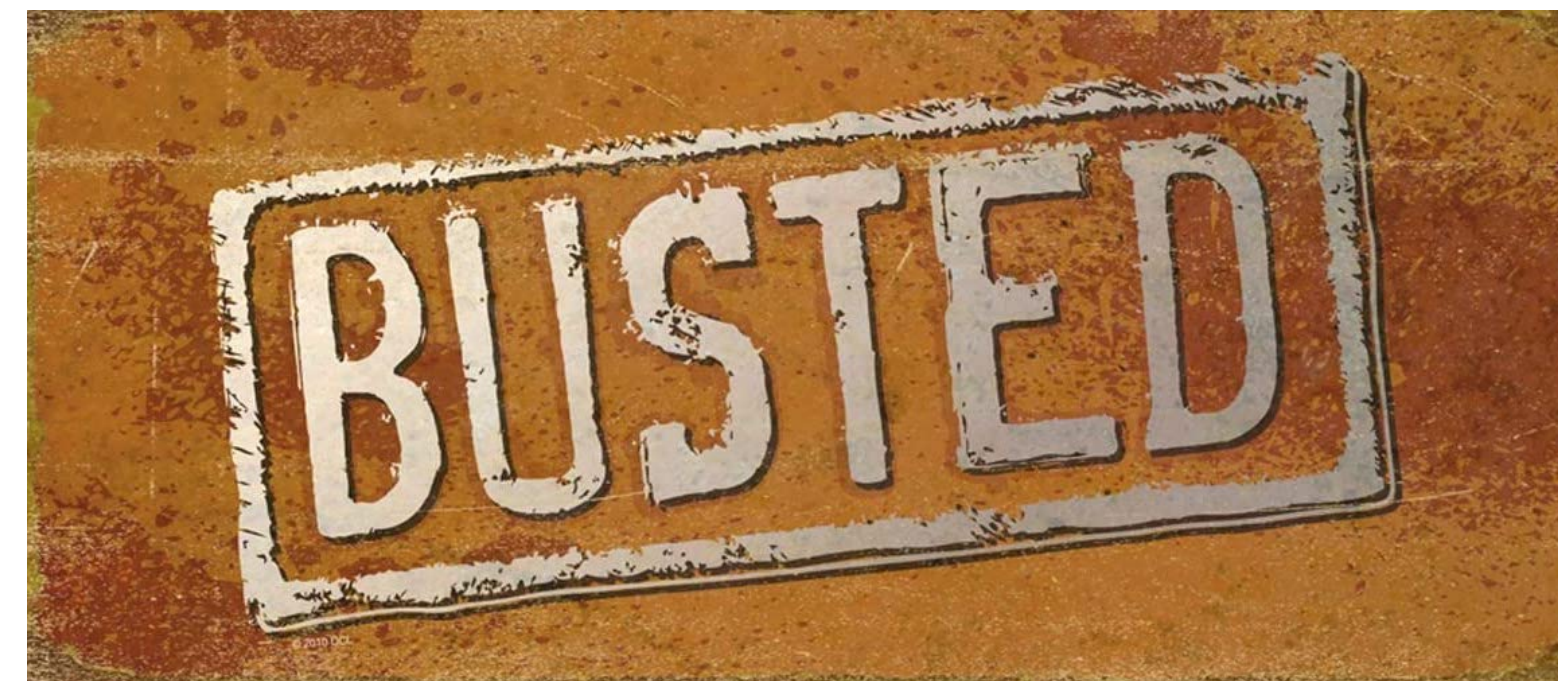
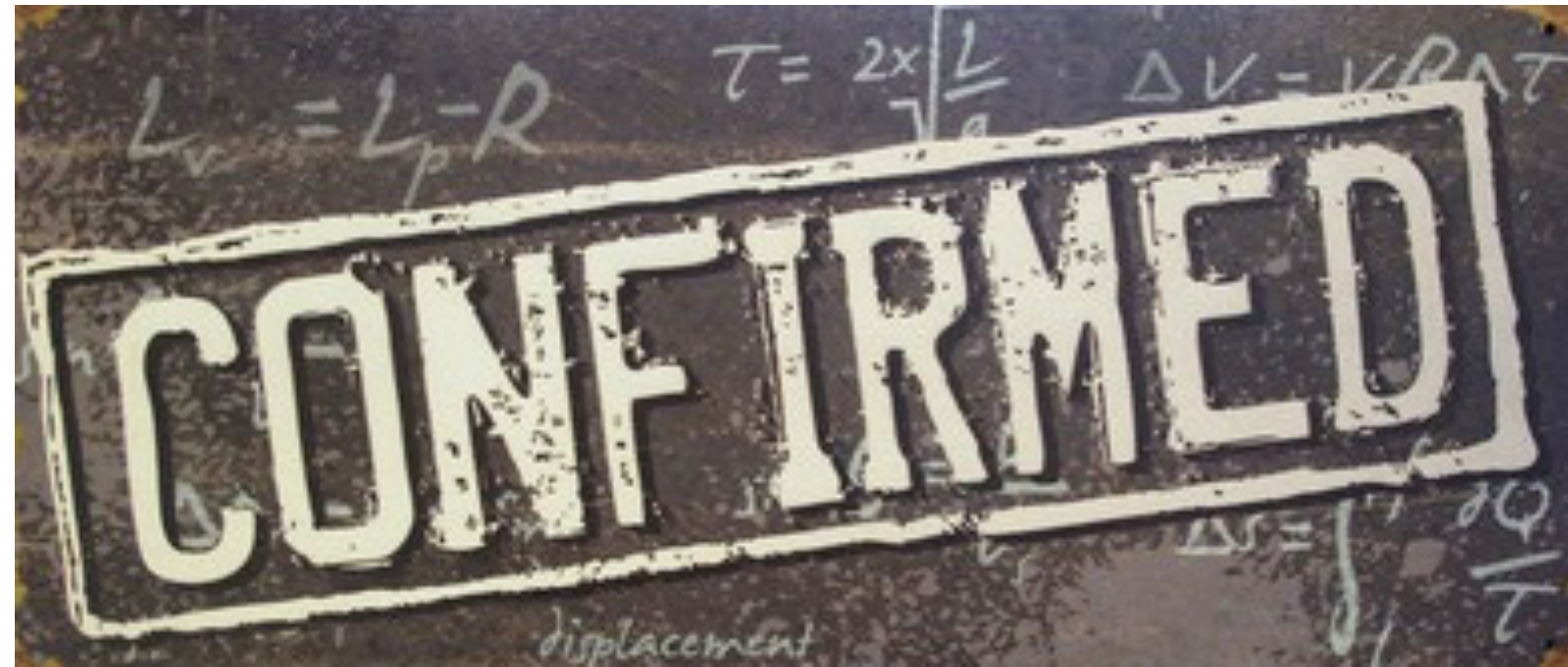
Victor Ciura

2022 Victor Ciura | @ciura_victor - C++ MythBusters

23:29 / 50:16

youtube.com/watch?v=ZGgrUhVNsSI

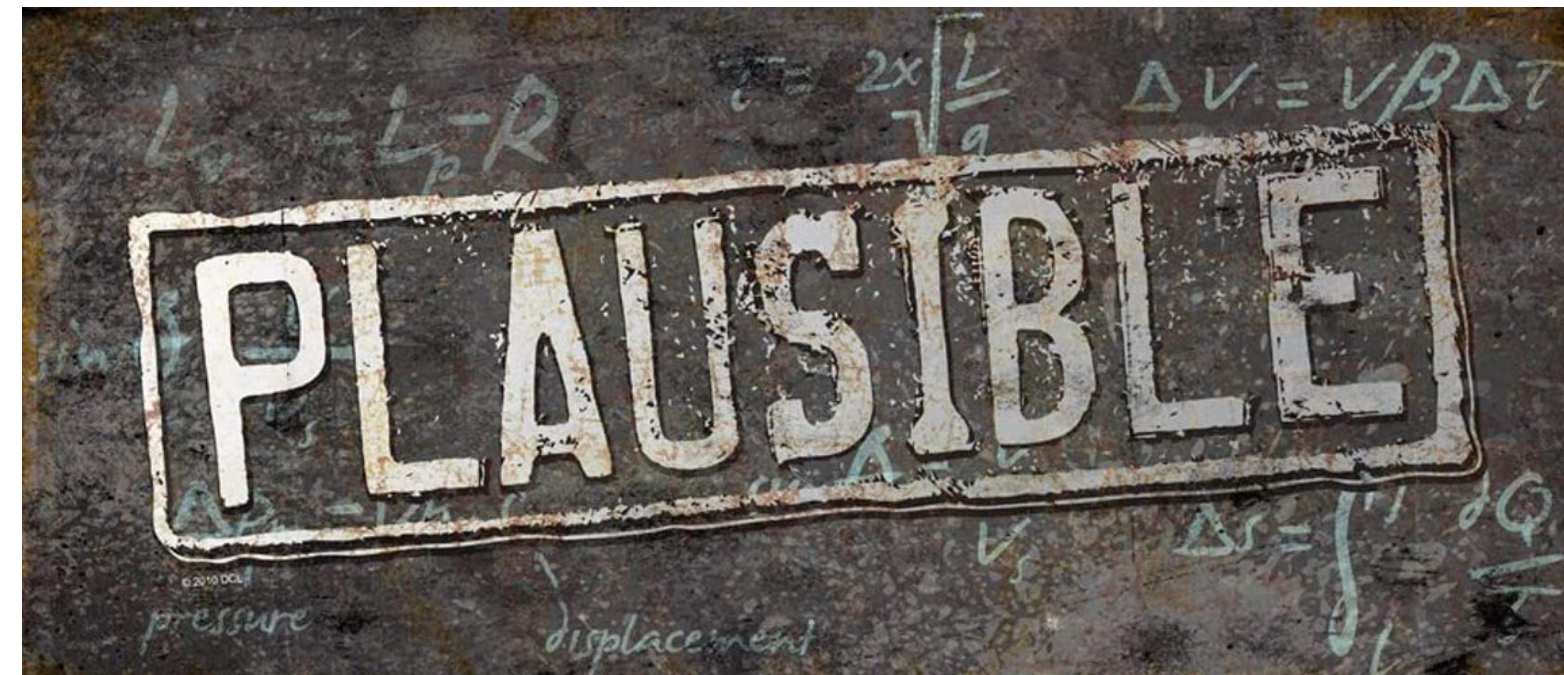
Verdict



Verdict

A programmer's staple response:

"It depends..." 🧐



Let's test this...



Test Myth

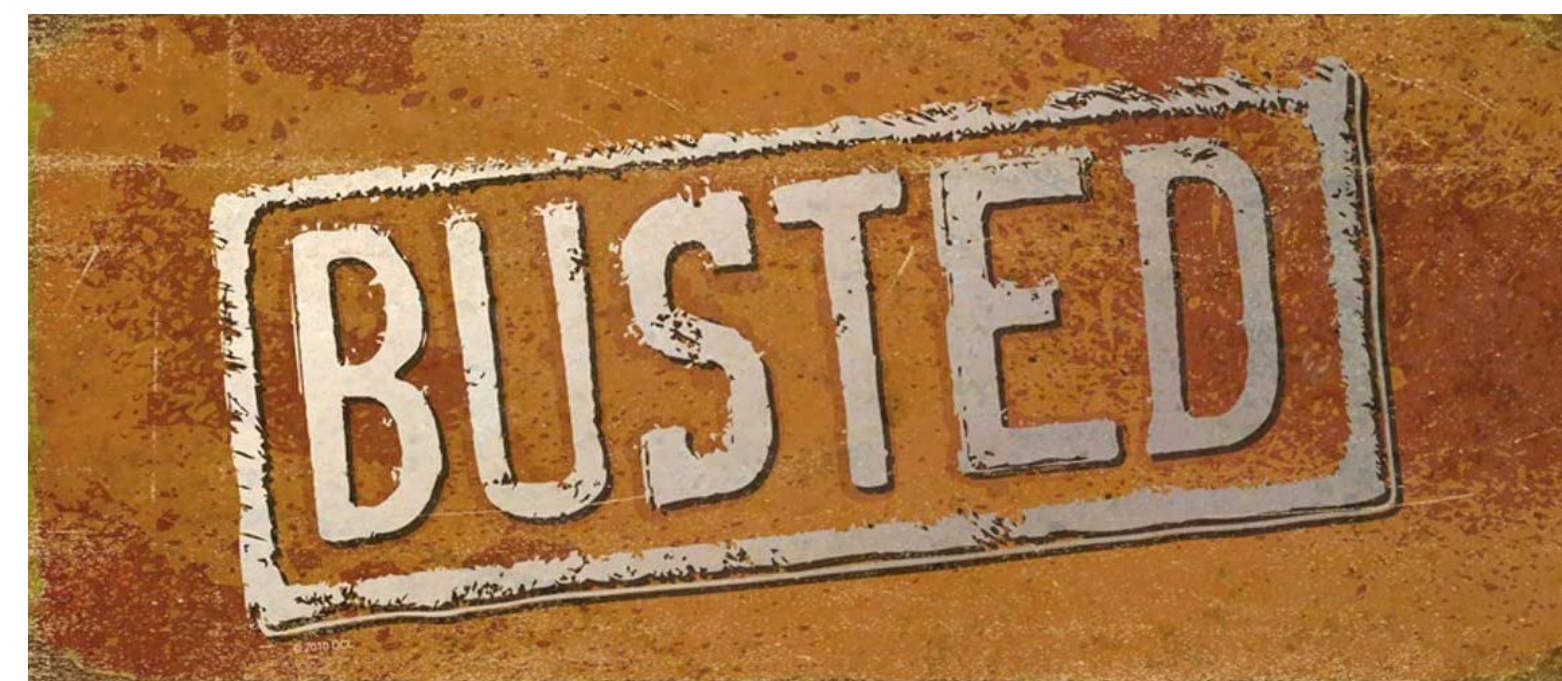
`iostreams` are slow



Just kidding 😊

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

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/

coroutines shipped in C++20

↻ Meeting C++ reposted



Ólafur Waage

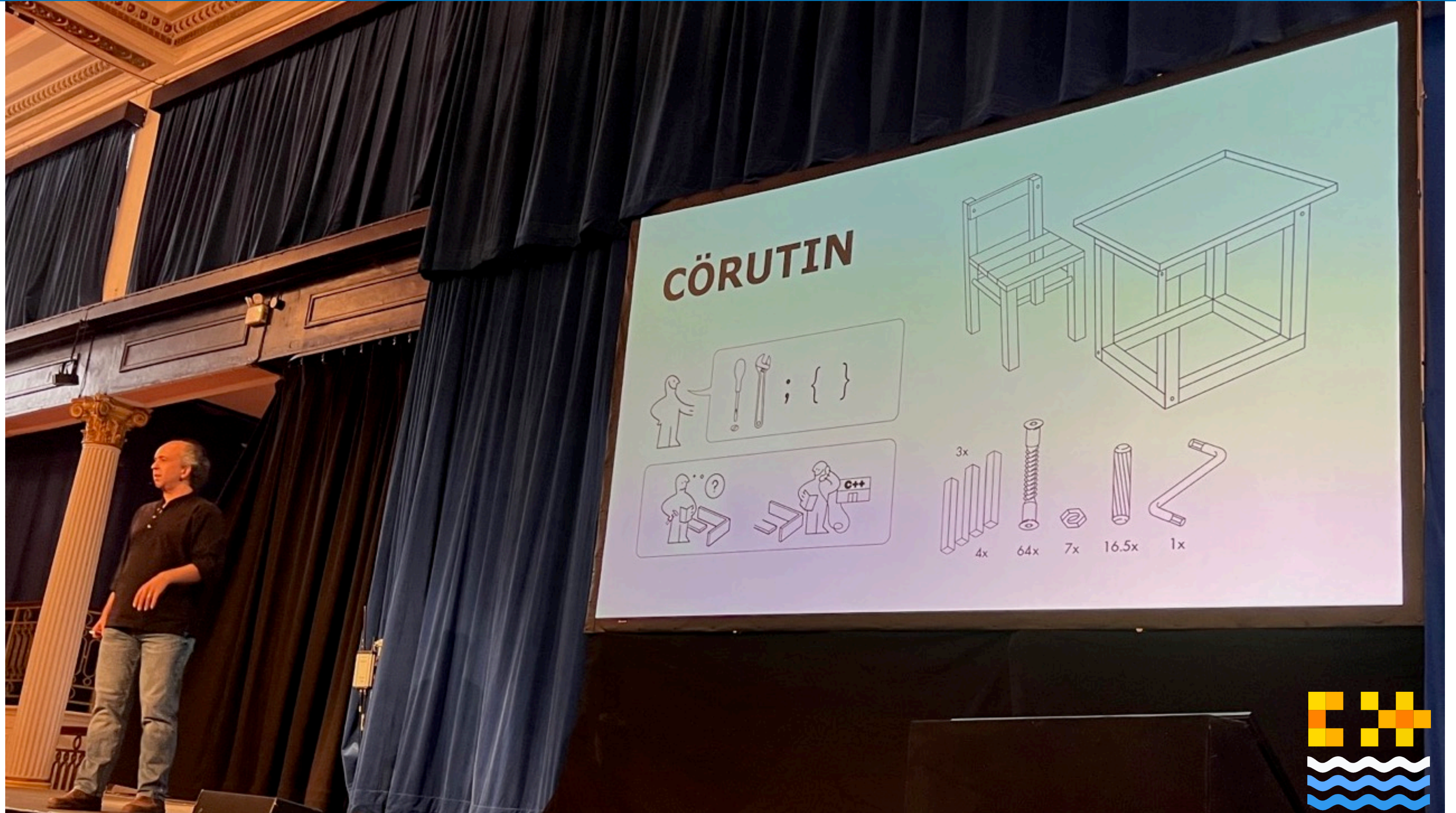
@olafurw



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

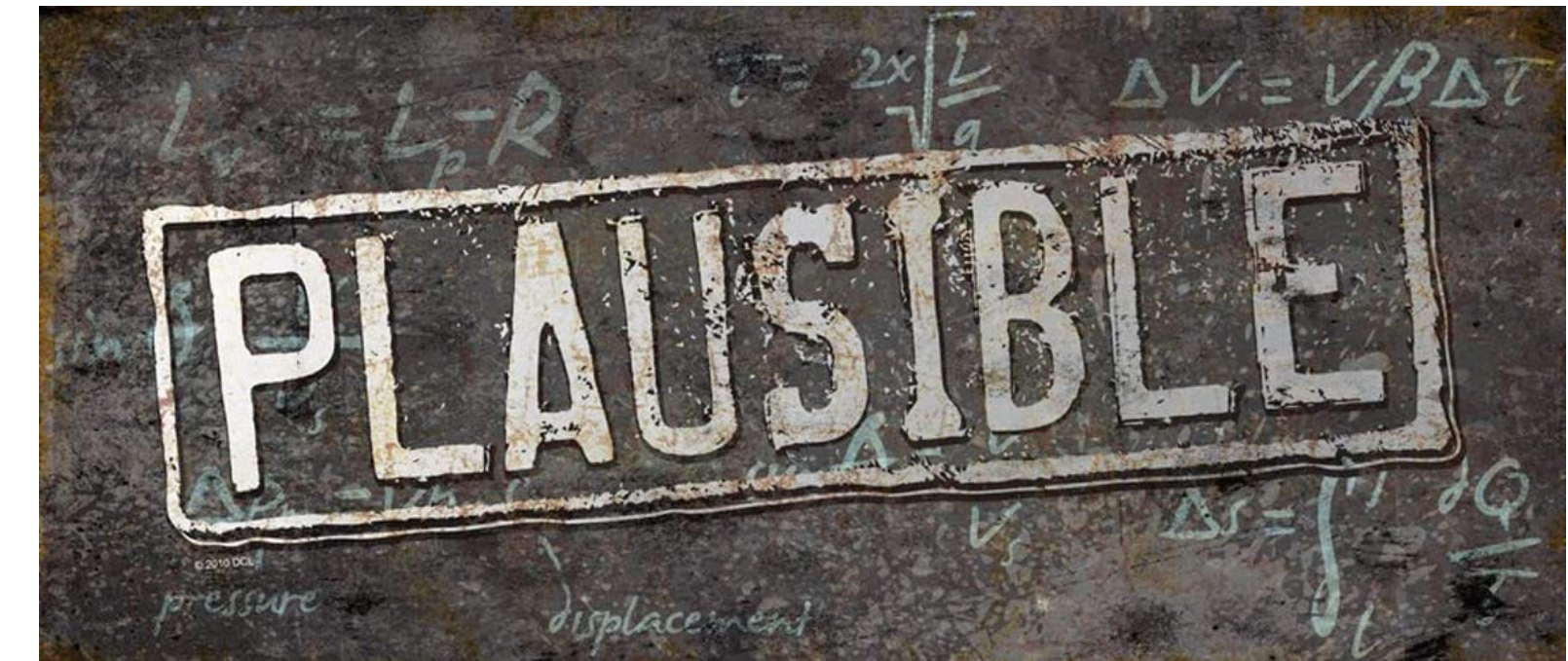
The committee really loves adding configuration options but seem to forget hello world case

No I will not submit a paper



Test Myth

coroutines shipped in C++20



Kinda... 🙄

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

```
#include <generator>
```



I think you got how it works

Let's dig in!

Humans Depend on Tools



Myth #14

C++ is not easily toolable 

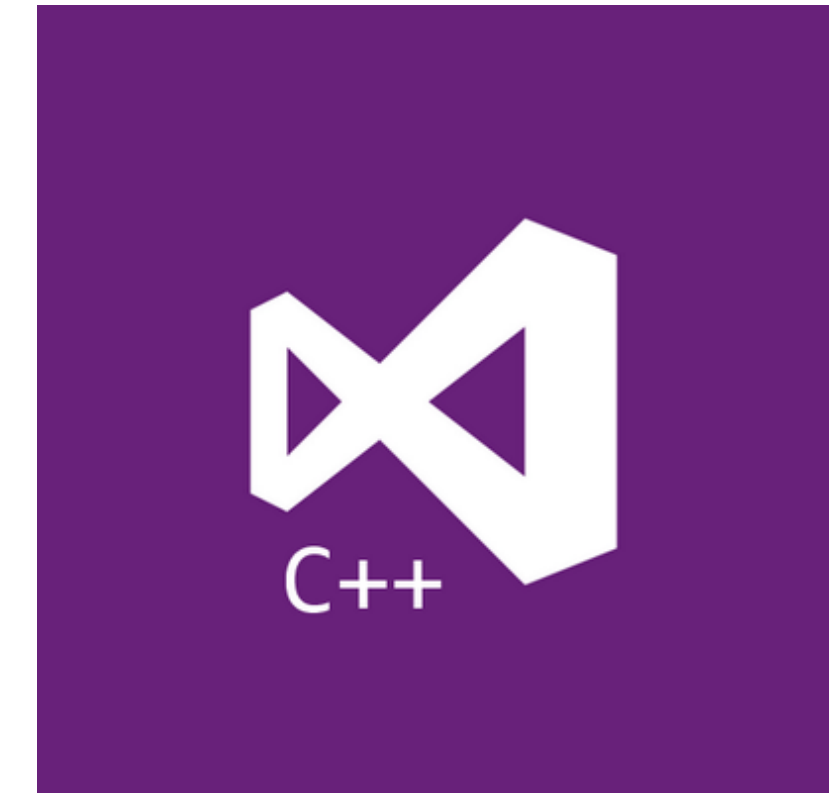
I'm a tool builder



[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 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

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

youtube.com/watch?v=q7Gv4J3FyYE

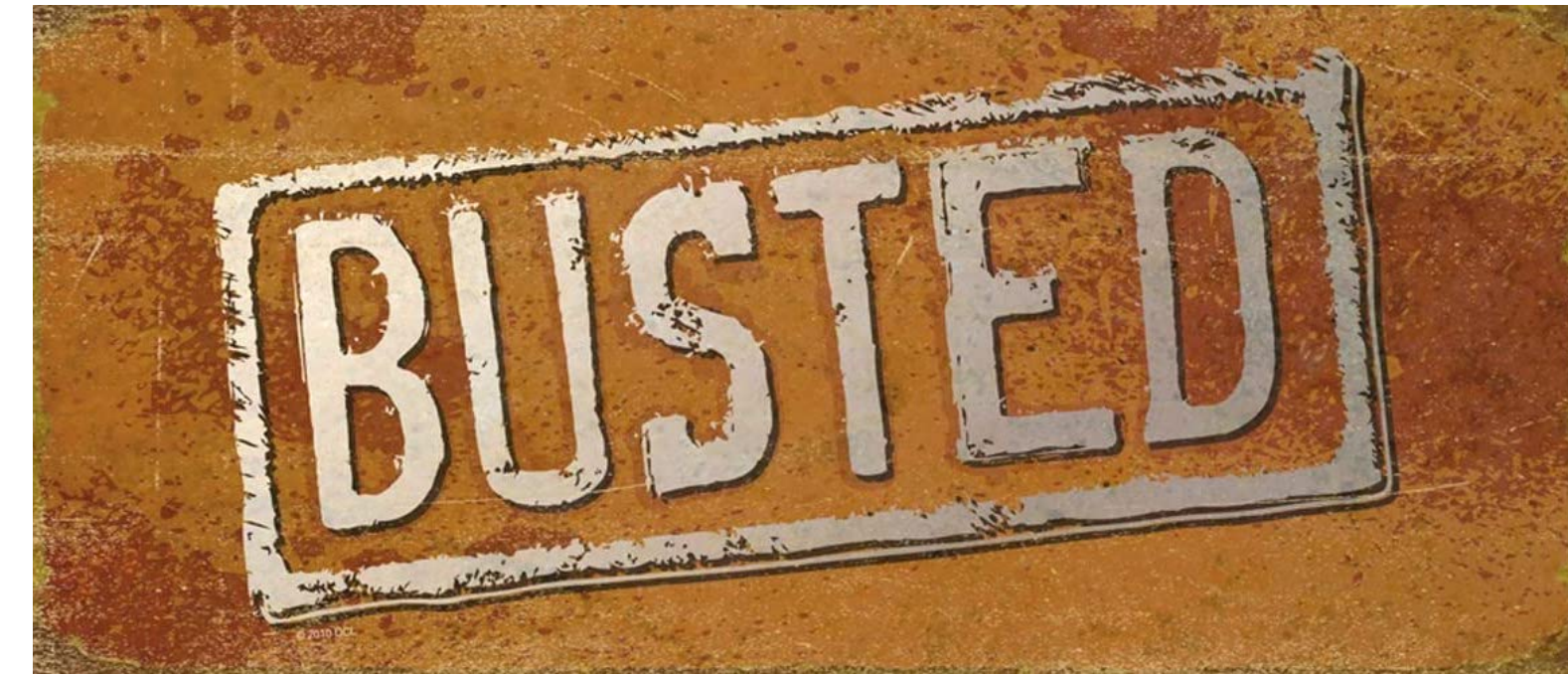
github.com/lefticus/cpp_weekly/issues/175

- 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.

C++ is not easily toolable 🛠️



Get to know your tools well

Myth #11

`printf/sprintf` are very fast

`sprintf` uses the **global locale**

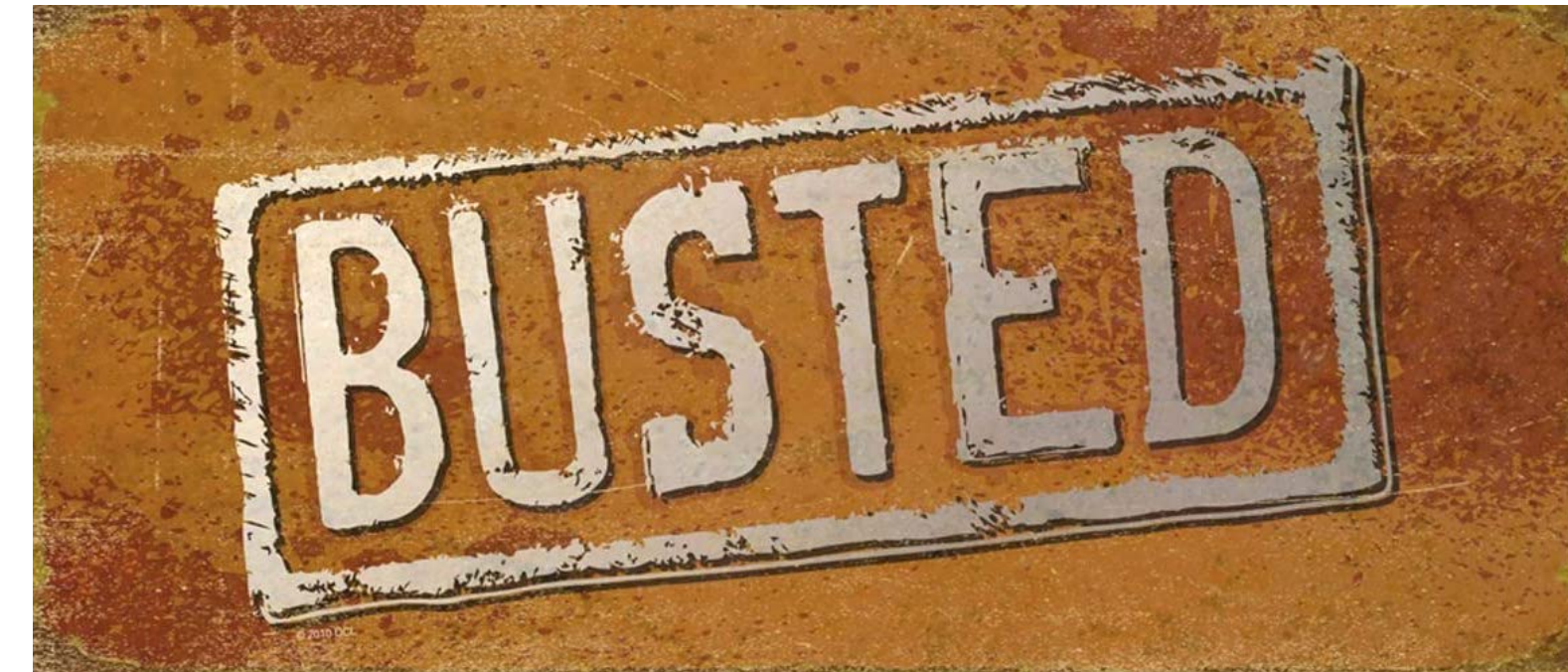
=> **mutex lock** 😞

On macOS, `sprintf` - that is in system libraries

ends up spending almost all the time inside a locale-related mutex lock 🔥

Myth #11

`printf/sprintf` are very fast



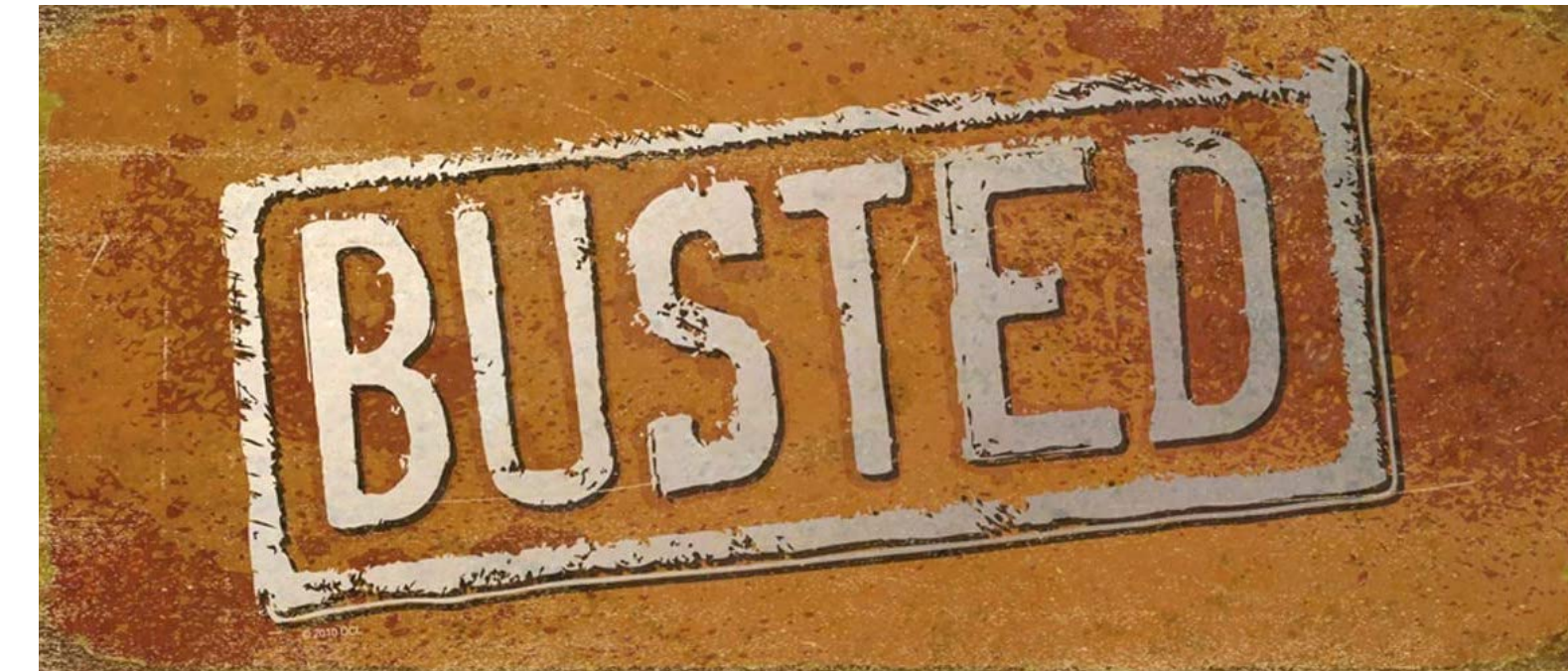
Blender case study: `sprintf` => `{fmt}`

- on macOS: **3-4x** speedup
- on Windows: **20%** speedup (due to a faster float/int formatter)

developer.blender.org/D13998

Myth #11

`printf/sprintf` are very fast



Use C++20 `std::format` or `{fmt}` library - `fmt::format()`

Use C++23 `std::print` or `{fmt}` library - `fmt::print()`

⚠ Beware of standard functions that use `locale`

Beware of **locale**



Josh Simmons
@dotstdy



Well tickle me surprised, after [@aras_p](#)'s locale adventures in blender, I yolo made an issue on the Microsoft STL github repo. Today it was closed as...
FIXED!!!1

microsoft/STL

#3030 **<locale>**: **Unnecessary locking around classic locale**



 2 comments



jsimmons opened on August 13, 2022



github.com

: Unnecessary locking around classic locale · Issue #3030 · microsoft/STL

When retrieving the classic locale the current code calls `_Init` which takes a lock regardless of initialization state. Expanded here, the code looks like the ...

twitter.com/dotstdy/status/1585530722751811584

Beware of **locale**

```
_MRTIMP2_PURE const locale& __CLRCALL_PURE_OR_CDECL locale::classic() { // get reference to "C" locale
    // _Init();
    {
        locale::_Locimp* ptr = nullptr;

        _BEGIN_LOCK(_LOCK_LOCALE) // prevent double initialization

        // this function just returns a global variable
        // ptr = _Getgloballocale();
        ptr = global_locale;

        if (ptr == nullptr) { // create new locales
            _Setgloballocale(ptr = _Locimp::_New_Locimp());
            ptr->_Catmask = all; // set current locale to "C"
            ptr->_Name     = "C";

            _Locimp::_Clocptr = ptr; // set classic to match
            _Locimp::_Clocptr->_Increref();
            ::new (&classic_locale) locale(_Locimp::_Clocptr);
        }

        // this is always false in the classic() codepath
        //if (_Do_increref) {
        //    ptr->_Increref();
        //}

        _END_LOCK()
        //return ptr;
    }
    return classic_locale;
}
```

github.com/microsoft/STL/issues/3030

Myth #19

`std::regex` is too slow for production use

This short snippet is so slow to compile, it will actually `timeout` in CompilerExplorer 😊

```
const auto r = std::regex(R"((\S+)\s*=\s*(\S+))");  
  
std::cmatch results;  
const auto success = std::regex_match("x = 5", results, r);  
  
fmt::print("Matched: {} '{}='{}'", success,  
           string(results[0].first, results[0].second),  
           string(results[1].first, results[1].second));
```


Myth #19

`std::regex` is too slow for production use

- difficult to use API
- very slow to compile
- very slow at runtime
- perf gotchas: regex c-tor, cmatch expensive



Myth #19

`std::regex` is too slow for production use

Use **CTRE** library instead:

- very fast to compile
- much cleaner API
- supports `string_view`
- builds regular expressions automata at compile time
- github.com/hanickadot/compile-time-regular-expressions



Myth #24

`std::optional` inhibits optimizations

Let's see...

Myth #24

```
C++ source #1 X
A [ + v [ C++
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 "Hello";
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 }
26
```

```
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1) X
x86-64 gcc 11.2 [x] -O3 -std=c++20 -Wall -Wextra -Wpedanti
A [ Output... [ Filter... [ Libraries + Add new... [ Add tool...
1 get_value[abi:cxx11](bool):
2     mov     rax, rdi
3     test   sil, sil
4     jne    .L5
5     mov    BYTE PTR [rdi+32], 0
6     ret
7 .L5:
8     lea   rdx, [rdi+16]
9     mov   DWORD PTR [rdi+16], 1819043144
10    mov   QWORD PTR [rdi], rdx
11    mov   BYTE PTR [rdi+20], 111
12    mov   QWORD PTR [rdi+8], 5
13    mov   BYTE PTR [rdi+21], 0
14    mov   BYTE PTR [rdi+32], 1
15    ret
16 get_size(bool):
17    cmp   dil, 1
18    sbb  rax, rax
19    or   rax, 5
20    ret
Output (0/0) x86-64 gcc 11.2 [i] - 3272ms (298483B) ~19264 lines filtered [
Output of x86-64 gcc 11.2 (Compiler #1) X
A [ [ ] Wrap lines
```

1819043144 = 0x6C6C6548

48 65 6c 6c
"Hell"

0x6F = 'o'

Myth #24



Add... More

Get cool gear in the [Compiler Explorer shop](#) x [Sponsors](#)

Share

C++ source #1 x x86-64 gcc 11.2 (C++, Editor #1, Compiler #1) x

A B + v 🔍 🐞 C++ x86-64 gcc 11.2 -O3 -std=c++20 -Wall -Wextra -Wpeda

Output... Filter... Libraries + Add new... Add tool...

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

no more SSO

compiler still sees through it and inlines it

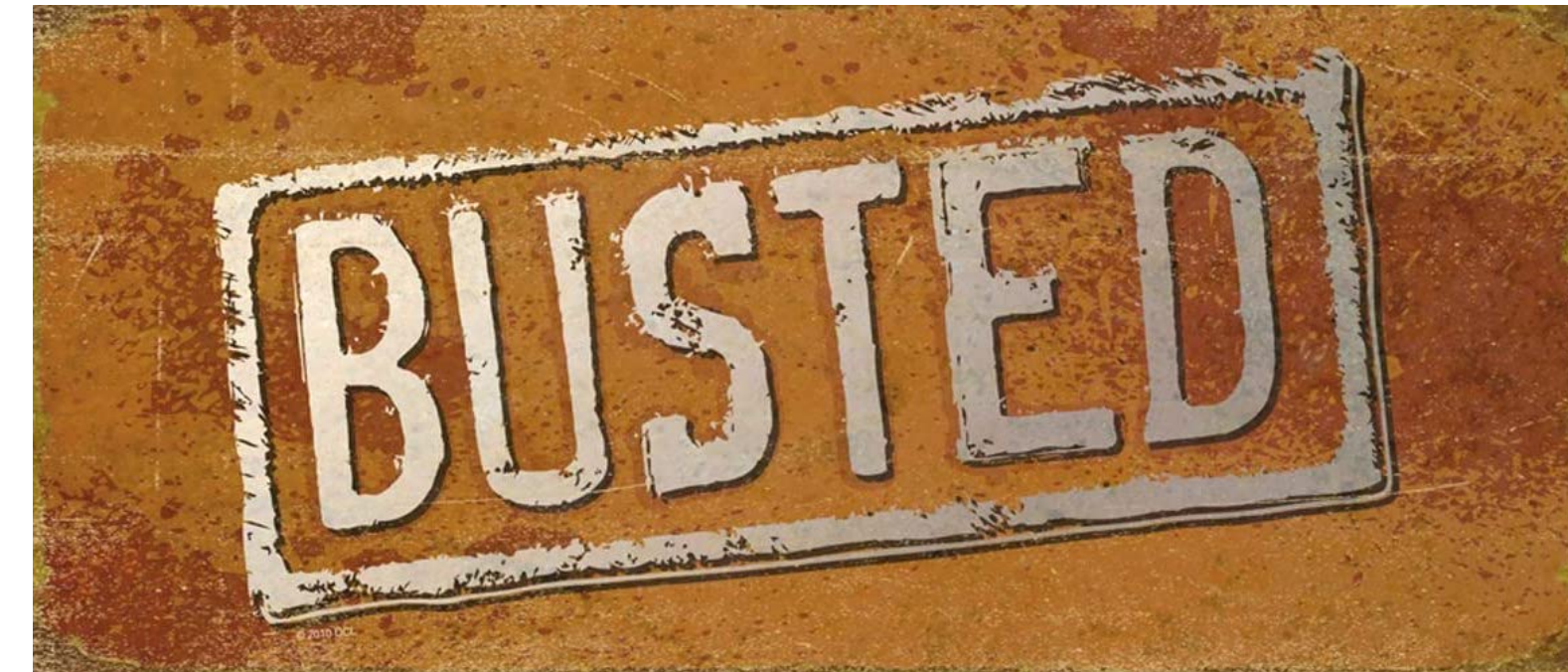
```
35 sub    rsp, 56
36 mov    edi, 24
37 lea   rax, [rsp+16]
38 mov   QWORD PTR [rsp], rax
39 call  operator new(unsigned long)
40 mov   esi, 24
41 mov   BYTE PTR [rsp+32], 0
42 movdqa xmm0, XMMWORD PTR .LC0[rip]
43 mov   DWORD PTR [rax+16], 1920234272
44 mov   rdi, rax
45 movups XMMWORD PTR [rax], 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
```

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

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

Myth #24

`std::optional` inhibits optimizations



However...


```

C++ source #1 X C++ source #2 X
A [Icons] C++
5 std::optional<std::string> get_value(bool condition)
6 {
7     std::string value = "This is a longer string";
8     if (condition)
9         return value;
10    else
11        return std::nullopt;
12 }
13
14 std::size_t get_size(bool condition)
15 {
16     const auto str = get_value(condition);
17     if (str)
18         return str->size();
19     else
20         return std::string::npos;
21 }
22
23 int main()
24 {
25     return get_size(true);
26 }

```

```

Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
Left: x86-64 gcc 11.2 -O3 -std=c++20 Assembly
Right: x86-64 gcc 11.2 -O3 -std=c++17 Assembly

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*, uns
54-  mov     eax, 23
55-  add     rsp, 56

66-  call   operator delete(void*, uns
67+  add     rsp, 88
68+  mov     rax, r12
69+  pop     rbx
70+  pop     r12
71+  ret
72+ .L10:
73+  mov     esi, 24
74+  mov     r12d, 23
75+  call   operator delete(void*, uns
76+  add     rsp, 88
77+  mov     rax, r12
78+  pop     rbx
79+  pop     r12

56-  ret
57 main:
58-  sub     rsp, 8
59-  mov     edi, 1
60-  call   get_size(bool)
61-  add     rsp, 8
62-  ret

80-  ret
81 main:
82-  sub     rsp, 8
83-  mov     edi, 1
84-  call   get_size(bool)
85-  add     rsp, 8
86-  ret

63- .LC0:
64-  .quad  2338328219631577172
65-  .quad  8243108416984981601

87- .LC0:
88+  .quad  23
89+  .quad  23
90+  .quad  23
91-  .quad  2338328219631577172
92-  .quad  8243108416984981601

```

copy constructing a string

~40% more instructions

```

x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)
x86-64 gcc 11.2 -O3 -std=c++20 -Wall -Wextra -Wper
Output... Filter... Libraries Add new... Add tool...
1 get_value[abi:cxx11](bool):
2   push   r12
3   mov    r12, rdi
4   test  sil, sil
5   ine   .L6

```


Myth #24

```
template <class U = T>  
constexpr optional(U && value);
```

Constructs an optional object that contains a value, initialized **as if** direct-initializing (but not direct-list-initializing) an object of type T with `std::forward<U>(value)`

- this constructor does not participate in overload resolution unless `std::is_constructible_v<T, U&&>` is *true* and `std::remove_cvref_t<U>` is neither `std::in_place_t` nor `std::optional<T>`
- this constructor is `explicit` iff `std::is_convertible_v<U&&, T>` is *false*

Good names

`std::move` doesn't move

`std::forward` doesn't forward

`std::remove` doesn't remove

`std::function` is not a function

...

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
    fmt::print("{} ', '{}", first, second);
}
```

```
int main()
{
    std::string greeting{"Hello from a long string"};
    echo(greeting, greeting);
}
```

```
'Hello from a long string', 'Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
    fmt::print("{} ', '{}'", first, second);
}
```

```
int main()
{
    std::string greeting{"Hello from a long string"};
    echo(std::move(greeting), greeting);
}
```

```
'Hello from a long string', 'Hello from a long string'
```


Myth #31

`std::move()` moves ?

```
void echo(const std::string & first, const std::string & second)
{
    fmt::print("{} ', '{}", first, second);
}
```

```
int main()
{
    std::string greeting{"Hello from a long string"};
    echo(std::move(greeting), std::move(greeting));
}
```

`string && => const string &`



```
'Hello from a long string', 'Hello from a long string'
```

Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
    fmt::print("{} ', '{}", first, second);
}
```

```
int main()
{
    std::string greeting{"Hello from a long string"};
    echo(std::move(greeting), std::move(greeting));
}
```

`string(std::move(greeting))`

clang

```
'Hello from a long string', ''
```


Myth #31

`std::move()` moves ?

```
void echo(const std::string first, const std::string second)
{
    fmt::print("{} ', {}'", first, second);
}

int main()
{
    std::string greeting{"Hello from a long string"};
    echo(std::move(greeting), std::move(greeting));
}
```

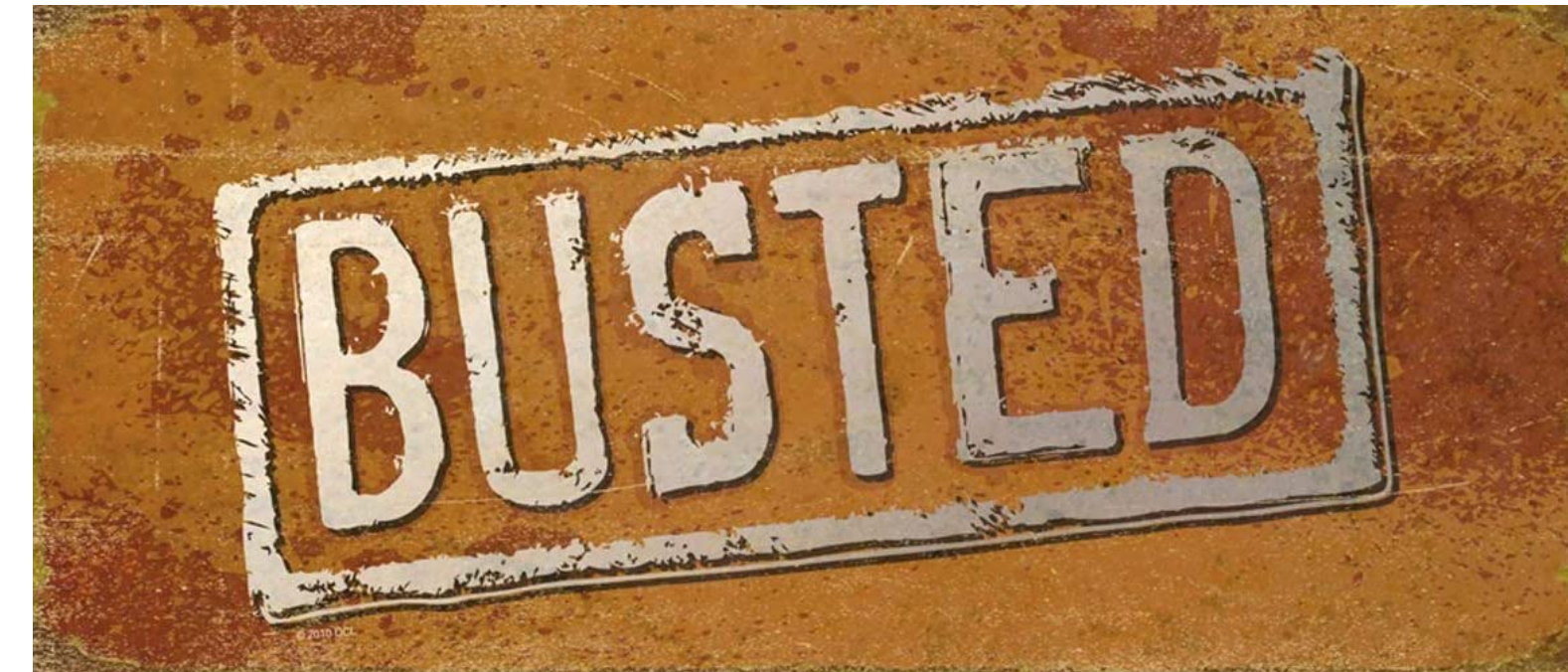
`string(std::move(greeting))`

gcc

```
' ', 'Hello from a long string'
```

Myth #31

`std::move()` moves ?



Myth #36

Always pass input arguments
by const reference

```
void echo(const std::string & first, const std::string & second);
```

Myth #36

```
class Widget
{
    std::string id;

public:
    Widget(const std::string & new_id)
        : id(new_id) {}

    Widget(std::string && new_id)
        : id(std::move(new_id)) {}
};
```


Myth #36

```
class Widget
{
    std::string id;
    std::string name;

public:

    Widget(const std::string & new_id, const std::string & new_name)
        : id(new_id), name(new_name) {}

    Widget(std::string && new_id, std::string && new_name)
        : id(std::move(new_id)), name(std::move(new_name)) {}

    Widget(const std::string & new_id, std::string && new_name)
        : id(new_id), name(std::move(new_name)) {}

    Widget(std::string && new_id, const std::string & new_name)
        : id(std::move(new_id)), name(new_name) {}

};
```



Myth #36

```
class Widget
{
    std::string id;
    std::string name;

public:
    Widget(std::string new_id, std::string new_name)
        : id(std::move(new_id)), name(std::move(new_name)) {}
};
```

when we take ownership (sink)

by value

Myth #36

```
class Widget
{
    std::string id;
    std::string name;
```

```
public:
```

```
void set_name(std::string new_name)
{
    name = std::move(new_name);
}
```

by value

```
};
```

when we take ownership (sink)

Myth #36

```
class Widget
{
    std::string id;
    std::string name;

public:

    void set_name(std::string new_name)
    {
        name = std::move(new_name);
    }
};

Widget w;
w.set_name("Hello from a long string");
```

- create the string with the literal value
- move assignment into data member

Myth #36

```
class Widget
{
    std::string id;
    std::string name;

public:

    void set_name(std::string new_name)
    {
        name = std::move(new_name);
    }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

- create the string with the literal value
- make a copy of the string
- move assignment into data member

Myth #36

```
class Widget
{
    std::string id;
    std::string name;

public:

    void set_name(std::string new_name)
    {
        name = std::move(new_name);
    }
};

Widget w;
std::string name{"Hello from a long string"};
w.set_name(std::move(name));
```

- create the string with the literal value
- move construct the string
- move assignment into data member

Myth #36

```
class Widget
{
    std::string name;

public:

    void set_name(const std::string & new_name)
    {
        name = new_name;
    }
    void set_name(std::string && new_name)
    {
        name = std::move(new_name);
    }
};
```

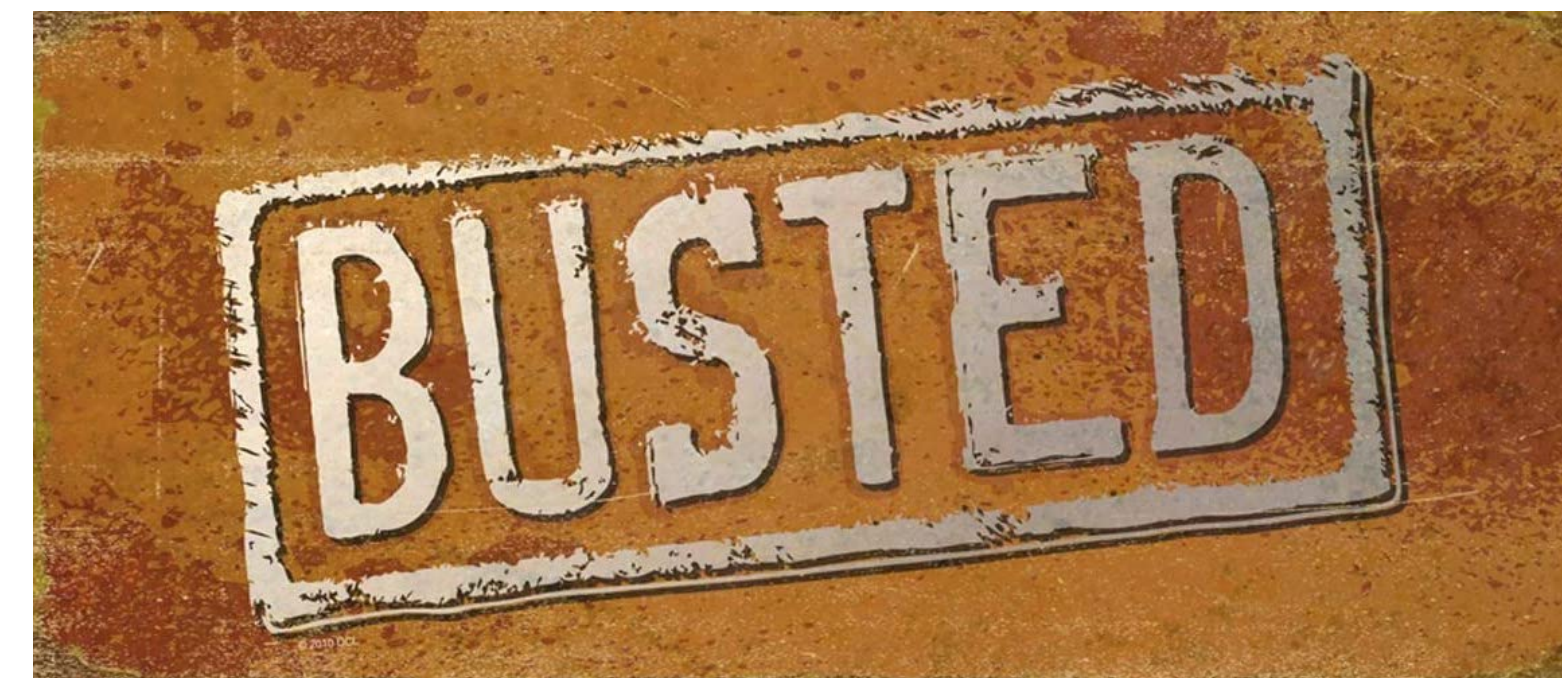
```
Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

Technically, more efficient
(one less move operation)

- create the string with the literal value
- make a copy of the string

Myth #36

Always pass input arguments
by const reference



There's even a [clang-tidy modernizer](#) check to perform
this transformation [automatically](#), at scale



clang.llvm.org/extra/clang-tidy/checks/modernize-pass-by-value

Myth #5

Adding **const** always helps

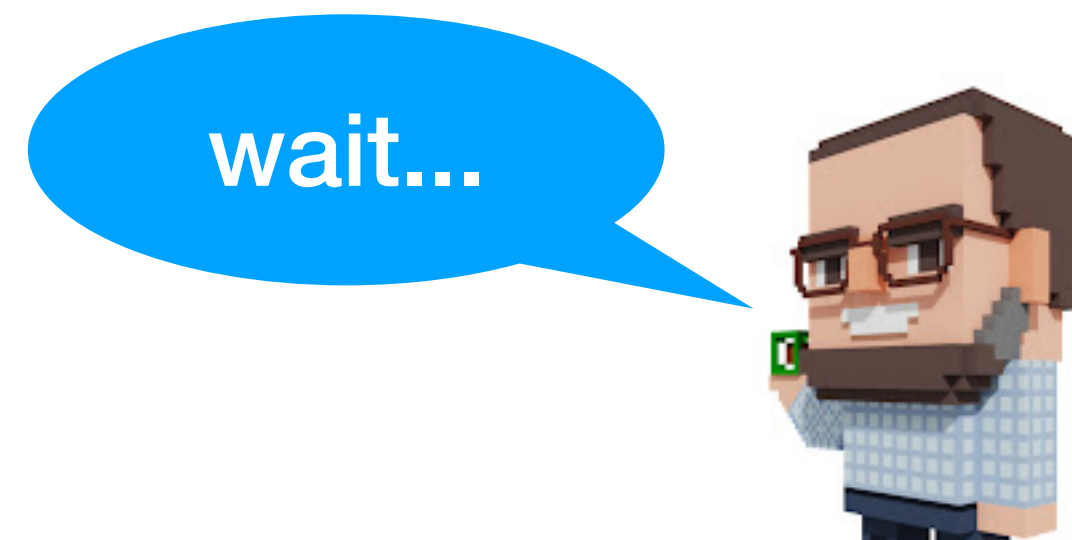
const all the things!



Myth #5

Adding **const** always helps

<https://www.youtube.com/watch?v=dGCxMmGvocE>



C++ Weekly
With Jason Turner
Episode 322
Top 4 Places To Never Use `const`

```
1 #include <string>
2 #include <string_view>
3
4 struct S
5 {
6     S() { puts("S()"); }
7     S(const S &) { puts("S(const S &)"); }
8     S(S&&) noexcept { puts("S(S&&)"); }
9     ~S() { puts("~S()"); }
10    S &operator=(const S &) { puts("operator=(const S &)"); return *this; }
11    S &operator=(S &&) noexcept { puts("operator=(S &&)"); return *this; }
12 };
13
14
```

Compiler Explorer Output:

```
Output of x86-64 gcc (trunk) (Compiler #1)
Compiler returned: 0
```

clang-tdy (trunk) #1 with x86-64 gcc (trunk)

```
33948 warnings generated.
Suppressed 33948 warnings (33948 in non-user code).
Use -header-filter=.* to display errors from all non-system headers. Use -system-headers to display
```

Thank You To My Patrons!

Music: Klain! (contact@klain-music.com)

C++ Weekly - Ep 322 - Top 4 Places To Never Use `const`

Myth #5

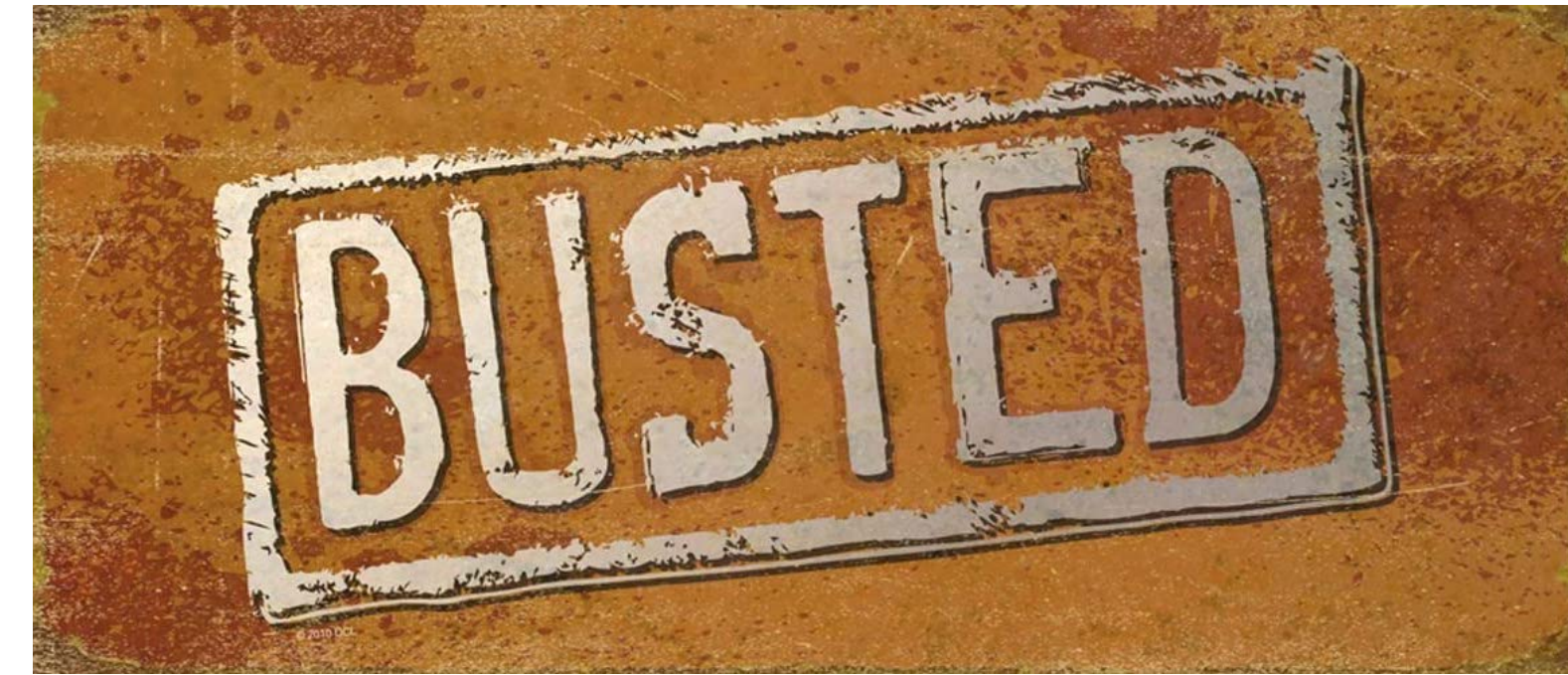
Top 4 places to **never use const**:

- don't **const non-reference return types**
- don't **const local values** that need to take advantage of implicit **move-on-return** operations (even if you have multiple different objects that might be returned)
- don't **const non-trivial value parameters** that you might need to **return directly** from the function
- don't **const any member data**
 - it breaks implicit and explicit moves
 - it breaks common use cases of assignment

compiler-explorer.com/z/9Wcc54r9x

Myth #5

Adding **const** always helps



Make All Data Members Private ?

- typically seen as **good practice**
- enforces **encapsulation**: the object is in control of its internal states
- not all types have **invariants** to enforce (document invariants)
- narrow/wide **contracts**
- added **complexity** (YAGNI - "*You aren't gonna need it*")
- write **simpler** classes
- maybe you don't need **constructors** either { }
- **refactoring** concerns

Myth #37

Make All Data Members Private ?

Sometimes `structs` just wanna be `structs` 😊

ACCU
2022

youtube.com/watch?v=Y3wxJD3Bpql

ABSTRACTION PATTERNS:

*MAKING CODE RELIABLY BETTER
WITHOUT DEEP UNDERSTANDING*

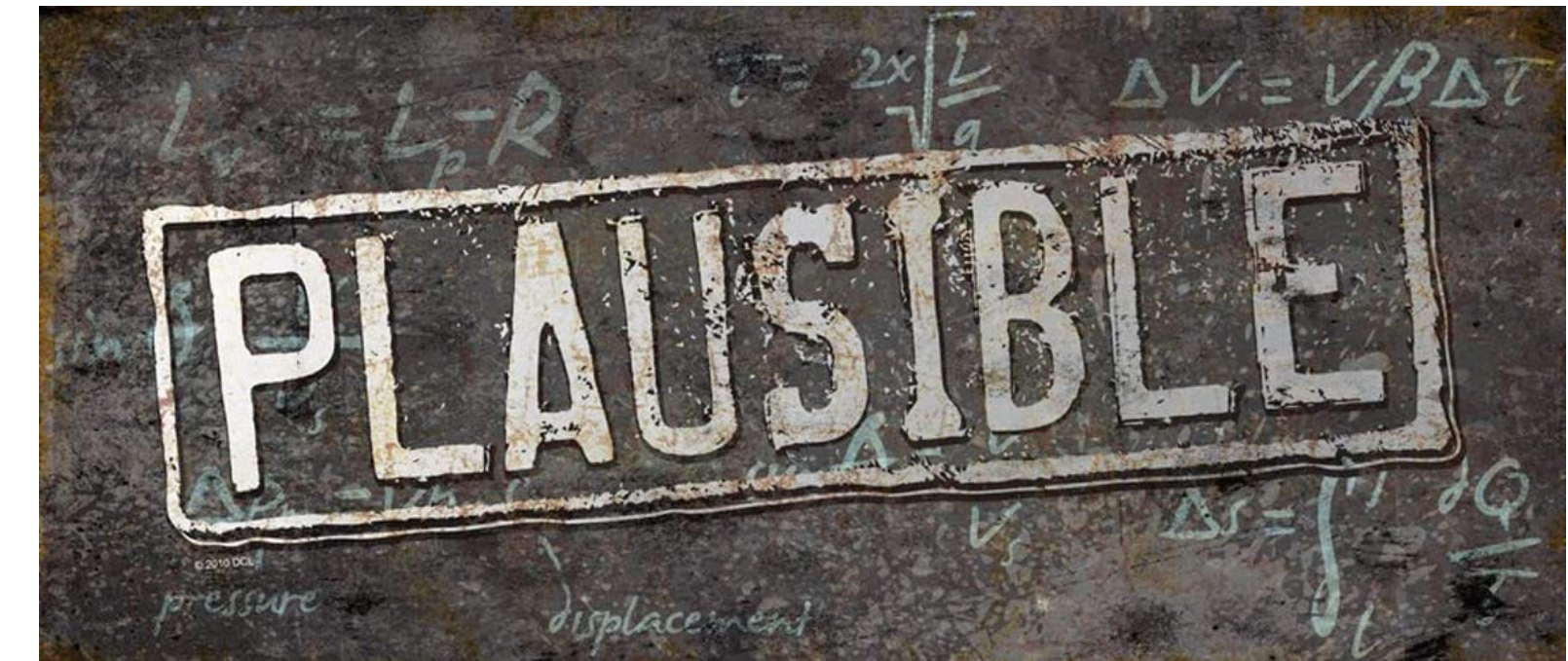
KATE GREGORY

00:02:39 / 01:24:13

Speed CC

Myth #37

Make All Data Members Private ?



Sometimes `structs` just wanna be `structs` 😊

std::ranges are safer than iterators

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

Myth #39

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

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

* the Nico slide :)

Basic Idioms Broken by Standards Views

C++20/C++23

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

Broken
for views

Broken
for views

Broken
for views

Broken
for views

Broken
for views

Broken
for views

Broken
for views



Nico Josuttis

youtube.com/watch?v=qv29fo9sUjY

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:

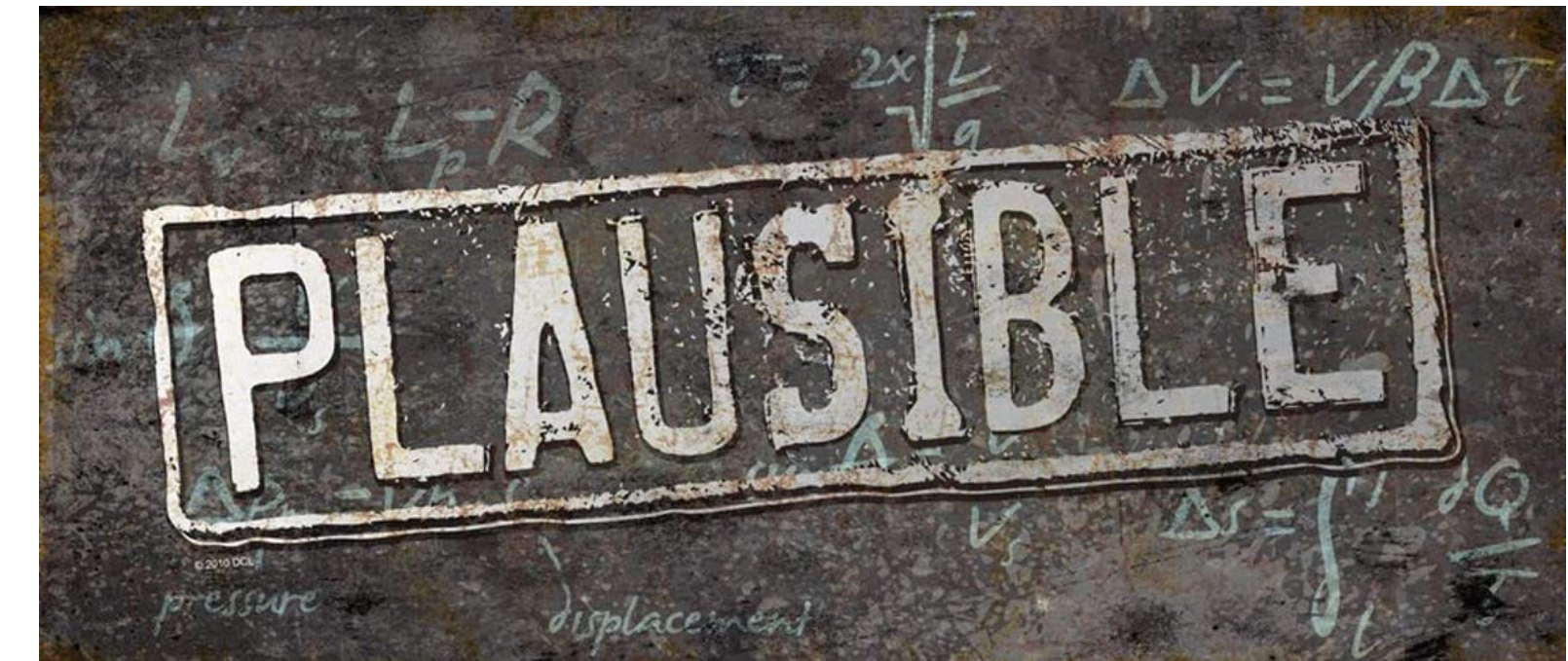
```
for (auto& m : monsters | std::views::filter(isDead)) {  
    m.resurrect(); // undefined behavior: because no longer dead  
    m.burn();     // OK (because it is still dead)  
}
```

Thanks to Patrice Roy for this example

youtube.com/watch?v=qv29fo9sUjY

Myth #39

`std::ranges` are safer than iterators



CMake is the gold standard of C++ project systems

“ CMake:

When it works, it's great;

when it doesn't, you're regretting your life decisions 😊

twitter.com/pati_gallardo/status/1672137915

CMake Debugger in Visual Studio and VSCode



youtube.com/watch?v=1eVJBEV9NTk

Myth #7

The screenshot displays the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with folders like .vs, .vscode, app, build, cmake, lib, mxchip_bsp, netxduo, stm32cubef4, threadx, wiced_sdk, and CMakeLists.txt. The main editor window shows the CMakeLists.txt file with the following content:

```
1 # Copyright (c) Microsoft Corporation.
2 # Licensed under the MIT License.
3
4 cmake_minimum_required(VERSION 3.13 FATAL_ERROR)
5 set(CMAKE_C_STANDARD 99)
6
7 set(GSG_BASE_DIR ${CMAKE_SOURCE_DIR}/../..)
8 set(SHARED_SRC_DIR ${GSG_BASE_DIR}/shared/src)
9 set(SHARED_LIB_DIR ${GSG_BASE_DIR}/shared/lib)
10
11 # use the repo version of ninja on Windows as there is no Ninja installer
12 if(WIN32)
13     set(CMAKE_MAKE_PROGRAM ${GSG_BASE_DIR}/cmake/ninja CACHE STRING "Ninja location")
14 endif()
15
16 # Set the toolchain if not defined
17 if(NOT CMAKE_TOOLCHAIN_FILE)
18     set(CMAKE_TOOLCHAIN_FILE "${GSG_BASE_DIR}/cmake/arm-gcc-cortex-m4.cmake")
19 endif()
20
21 list(APPEND CMAKE_MODULE_PATH ${GSG_BASE_DIR}/cmake)
22 list(APPEND CMAKE_MODULE_PATH ${CMAKE_SOURCE_DIR}/cmake)
23
24 include(utilities)
25
26 # Define the Project
27 # CXX enables IntelliSense only. Sources are still compiled as C.
28 project(mxchip_azure_iot C CXX ASM)
29
30 # Disable common networking component, MXCHIP has it's own
31 set(DISABLE_COMMON_NETWORK true)
```

Below the editor, the Output window shows the CMake/Build output, including the following text:

```
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-f4e3197cadbf8319a52.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-3097c119013081eca4e4.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_noplatfrom-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



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



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



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`

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

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

Myth #10



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

Included Files		Include Tree	
Diagnostics Session: 75.462 seconds Build: 72.59 seconds			
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](#)

Trace230609110806.et What's New?

Included Files Include Tree

Diagnostics Session: 76.549 seconds Build: 73.506 seconds Filter Files

File Path	Time [sec, %]	Include Count	Project
▲ C:\src\irrlicht_pch\source\Irrlicht\Irrlicht.cpp	0.821 (1.1%)	6	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\um\...	0.431 (0.6%)	34	Irrlicht15.
▷ C:\src\irrlicht_pch\include\irrlicht.h	0.308 (0.4%)	97	Irrlicht15.
▲ C:\src\irrlicht_pch\include\IrrCompileConfig.h	0.042 (0.1%)	1	Irrlicht15.
▲ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\uc...	0.042 (0.1%)	2	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\...	0.019 (0.0%)	1	Irrlicht15.
▷ C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\...	0.005 (0.0%)	1	Irrlicht15.
▷ C:\src\irrlicht_pch\source\Irrlicht\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



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

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/

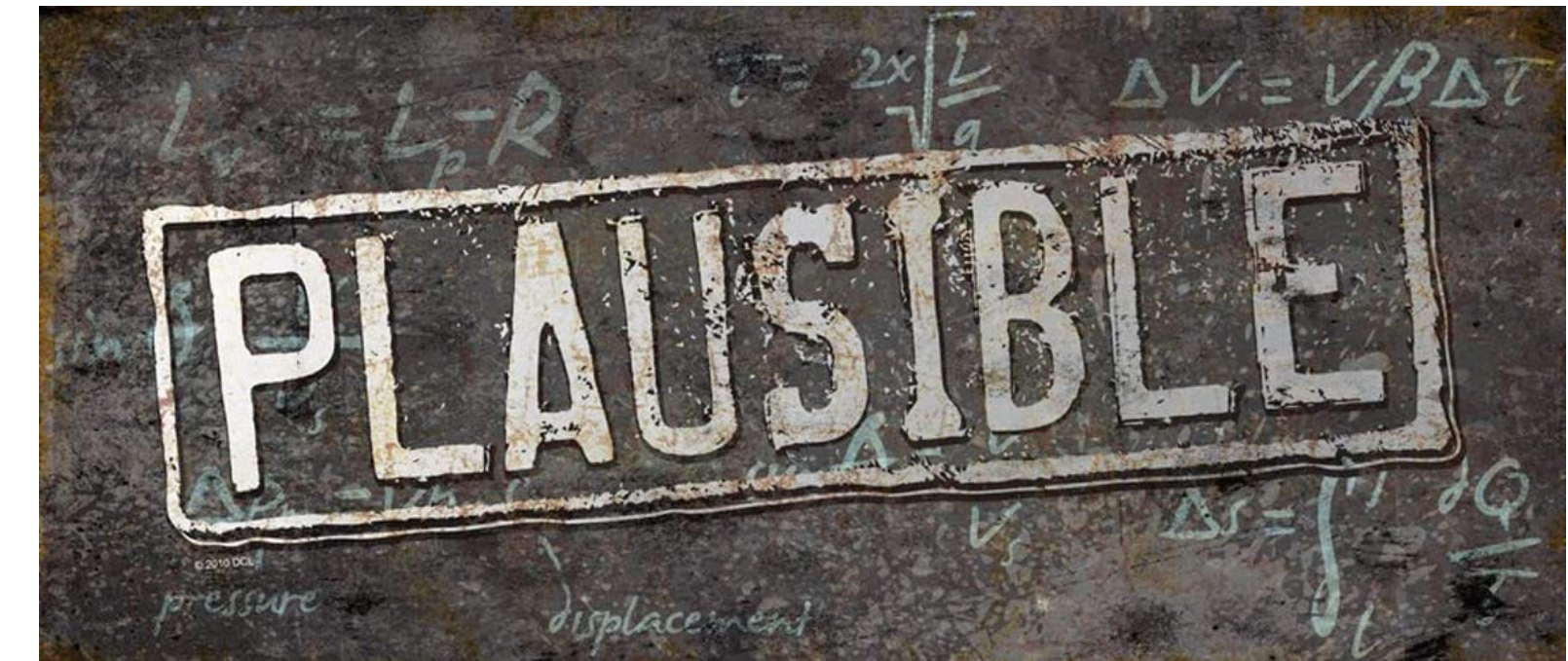
Myth #10

The image shows a YouTube video player interface. At the top left, the text reads "IMPROVING COMPILATION TIMES TOOLS & TECHNIQUES". At the top right, the ACCU 2023 logo is displayed. The main video area features a title slide with the text "IMPROVING COMPILATION TIMES" and "Tools & Techniques". The event information "ACCU 2023 April 20 2023" is shown in the top right of the slide. The speaker's name "Vittorio Romeo" is at the bottom left, with contact information: "mail@vittorioromeo.com" and "@supahvee1234". The Bloomberg Engineering logo and "TechAtBloomberg.com Career" are at the bottom right. A small inset video on the right shows the speaker at a podium. The video player controls at the bottom include a play button, a progress bar showing "0:06 / 1:43:50", a volume icon, a pause button, a closed captions icon, a settings gear, a full screen icon, and a share icon. The ACCU.ORG logo is centered at the bottom of the video area.

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

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]]`

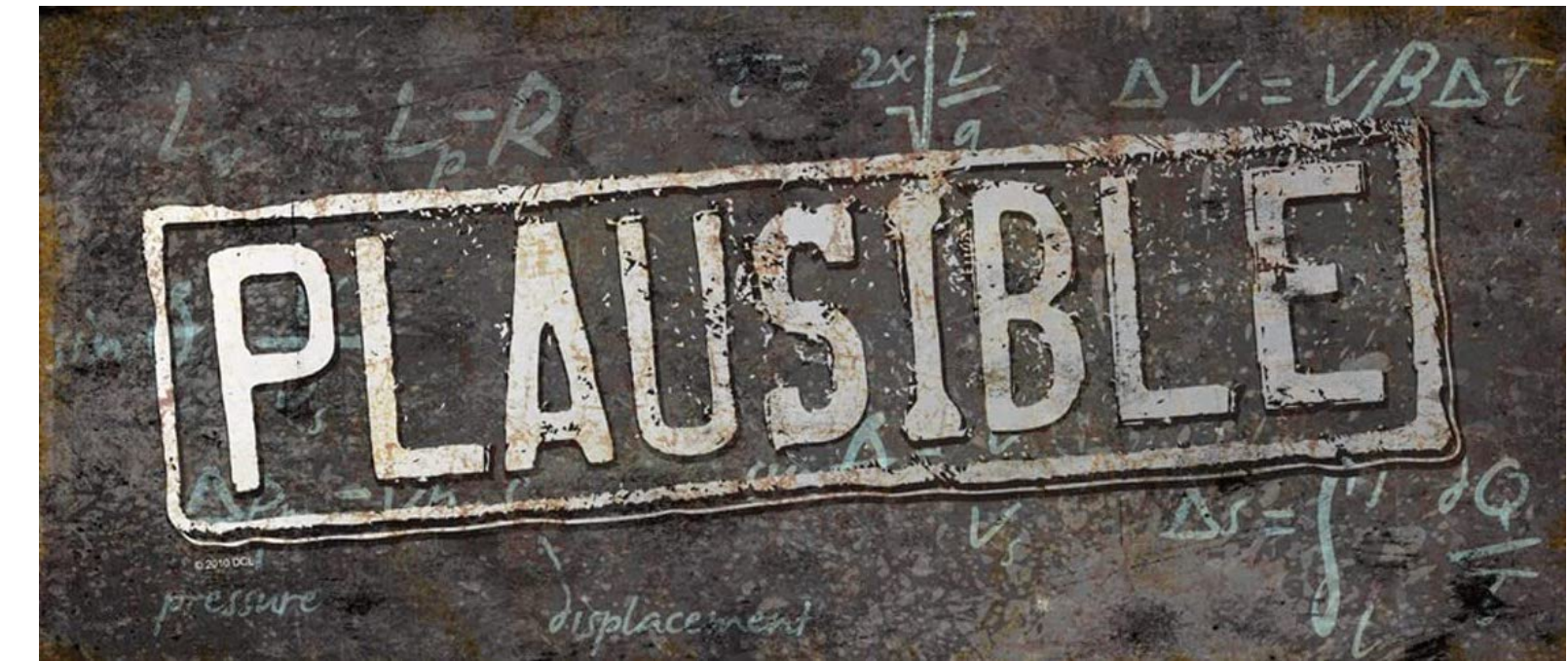
=> *extensible* to your own such utility functions

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++



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

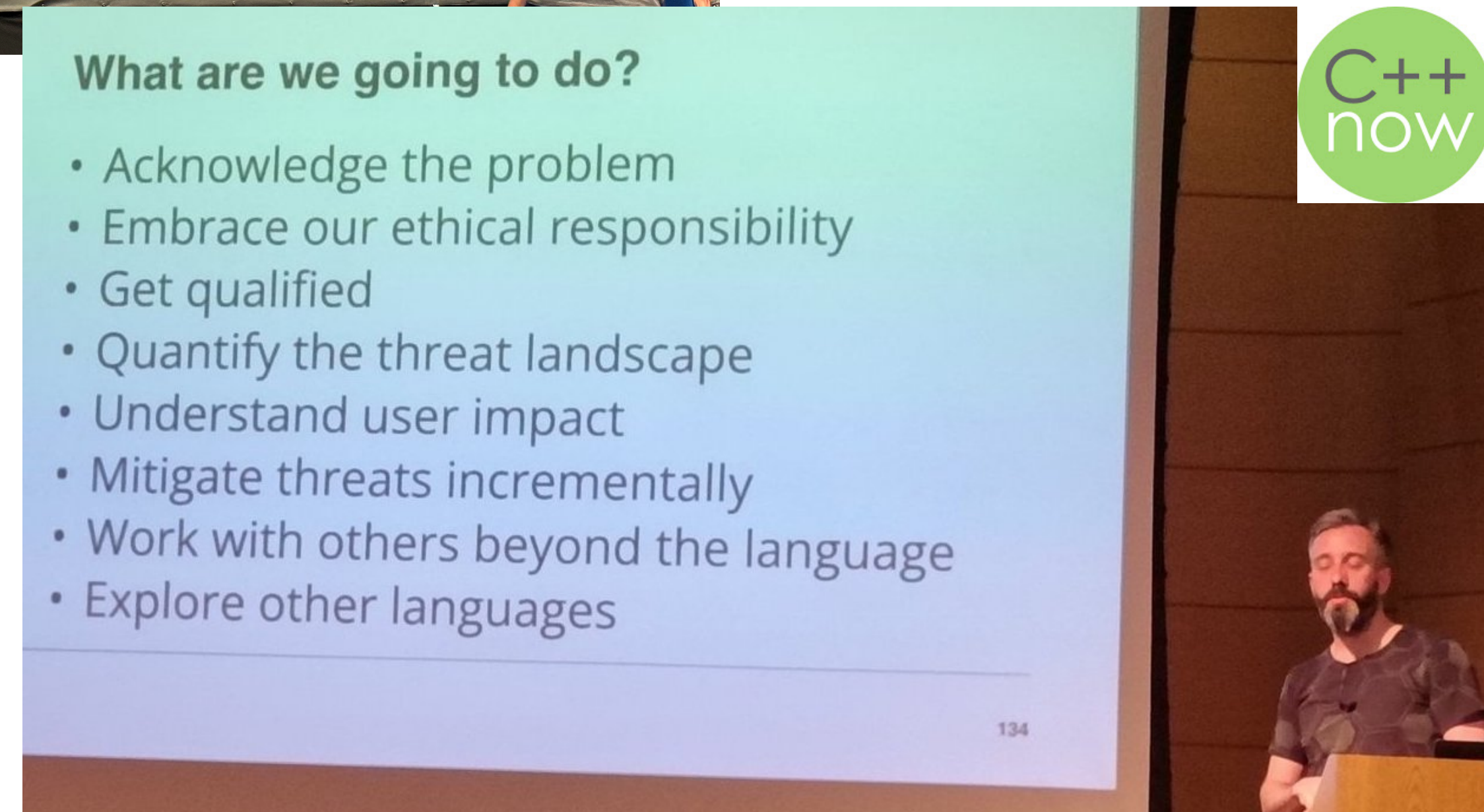
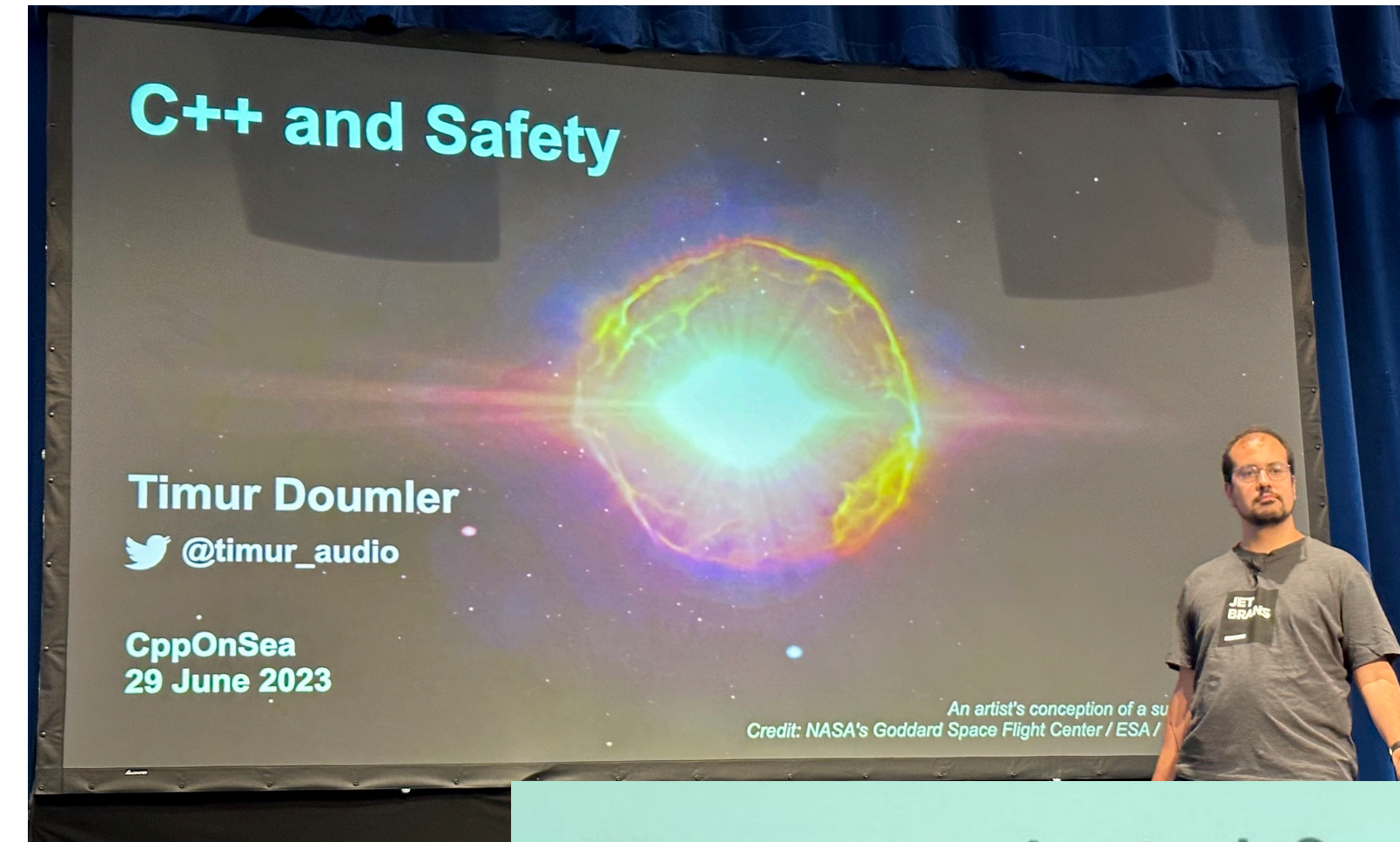
- type safety
- bounds safety
- lifetime safety
- initialization safety
- object access safety
- thread safety
- arithmetic safety

Myth #23

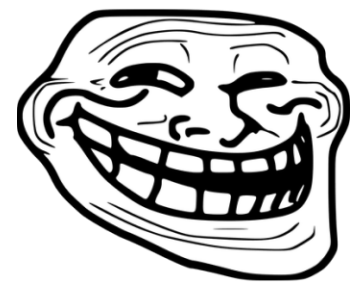
C++ is under attack... and the community is responding 🙋

Software Memory Safety

defense.gov/2022/Nov/CSI_SOFTWARE_MEMORY_SAFETY.PDF



Tradeoffs need to be made...

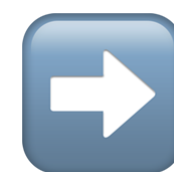


"To UB, or not to UB"

-- *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:

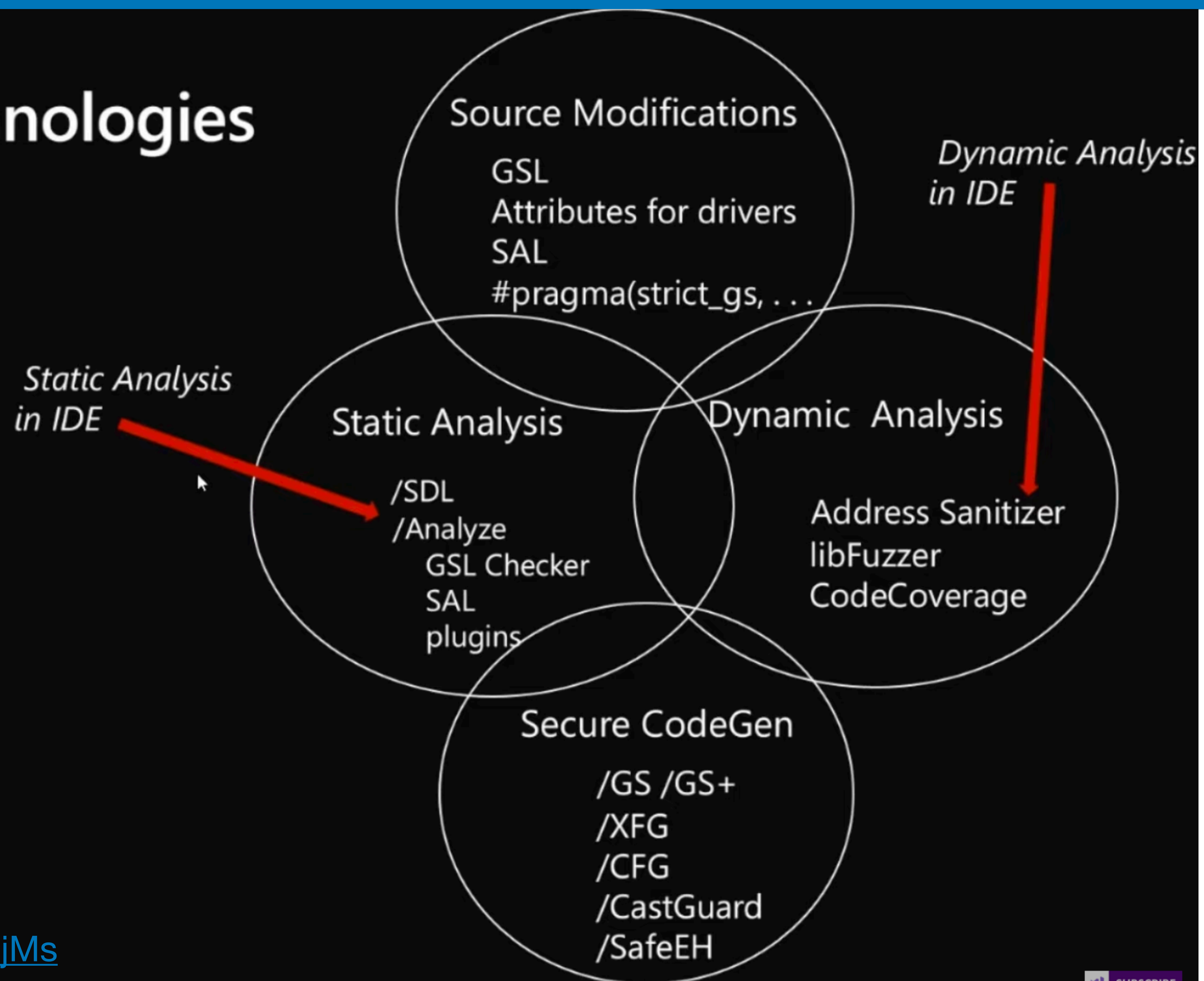
- garbage collector 🤖
- *dynamic* memory analysis (ASan)
- *statically* enforce rules on references: **multiple immutable refs** | | **unique mutable ref**
 - **by compiler/language:**
 - borrow checker (Rust)
 - mutable value semantics (Val Hylo)
 - no direct mutation (Haskell & other pure functional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

The new C++ "AAA"

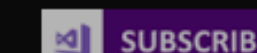
~~AAA (almost always auto)~~

AAA (almost always analyze)

C++ Security Technologies



youtube.com/watch?v=i8_RfDAEjMs



Myth #23

ASan FTW !!!

`-fsanitizer=address`

{ Clang, gcc, MSVC }

youtube.com/watch?v=yJLyANPHNaA

Victor Ciura

2020: The Year of Sanitizers?

Cppcon
The C++ Conference

2020
September 13-18
ONLINE
GOING VIRTUAL

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 image is a screenshot of a presentation slide. At the top left, there is a logo consisting of three curved lines in yellow and white. To its right, the text 'Pure Virtual' is written in white, with 'C++ 2023' below it in yellow. In the top right corner, the Microsoft logo is visible. The main content of the slide is 'Address Sanitizer' in white, with 'continue_on_error' below it in white. To the right of this text is a circular portrait of a man with white hair and glasses, identified as 'Jim Radigan'. Below his name, it says 'Partner Software Architect, Microsoft'. There are also some faint yellow symbols on the slide, including a code symbol and a plus sign.

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 is a video thumbnail for a presentation. On the left, there is a logo consisting of three curved lines in yellow and white. To the right of the logo, the text 'Pure Virtual' is in white, 'C++ 2023' is in yellow, and 'Lifetime Analysis Improvements' is in white. A play button icon is positioned to the right of the title. In the top right corner, the Microsoft logo is visible. On the right side, there is a circular portrait of Gabor Horvath, a man with short dark hair, wearing a black shirt. Below the portrait, his name 'Gabor Horvath' is written in white, followed by 'Software Engineer, Microsoft' in a smaller white font. The background of the thumbnail is dark with some faint yellow and white geometric shapes.

Myth #23

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



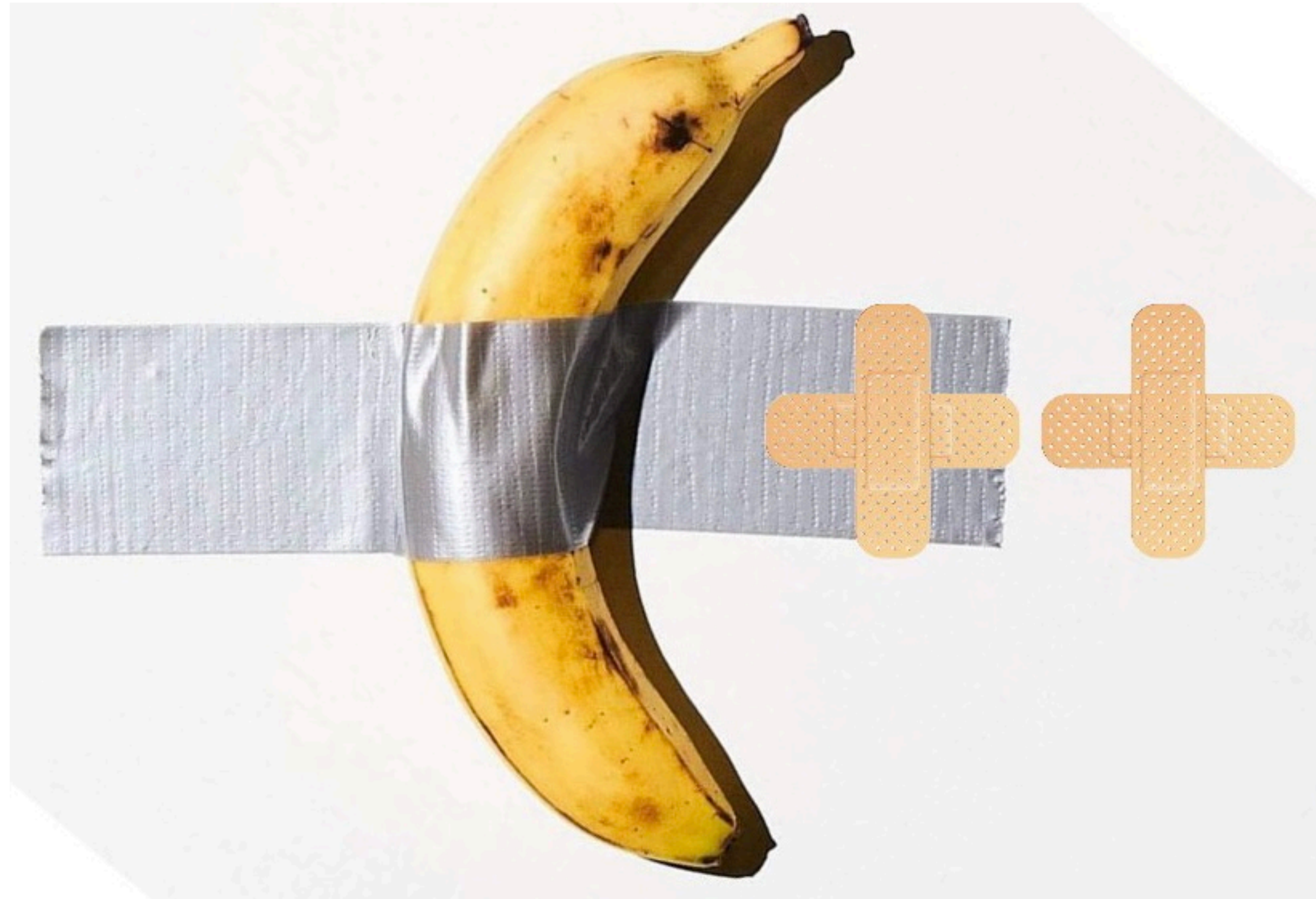
* but it can be much **safe(r)** with some effort and **good tooling** 

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



Myths, Dogma and Practice

~2023();

 @ciura_victor

 @ciura_victor@hachyderm.io

Victor Ciura
Principal Engineer
Visual C++

