



code::dive 

November 2022



@ciura_victor

Victor Ciura
Senior SW Engineer
Visual C++



Abstract

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

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

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

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

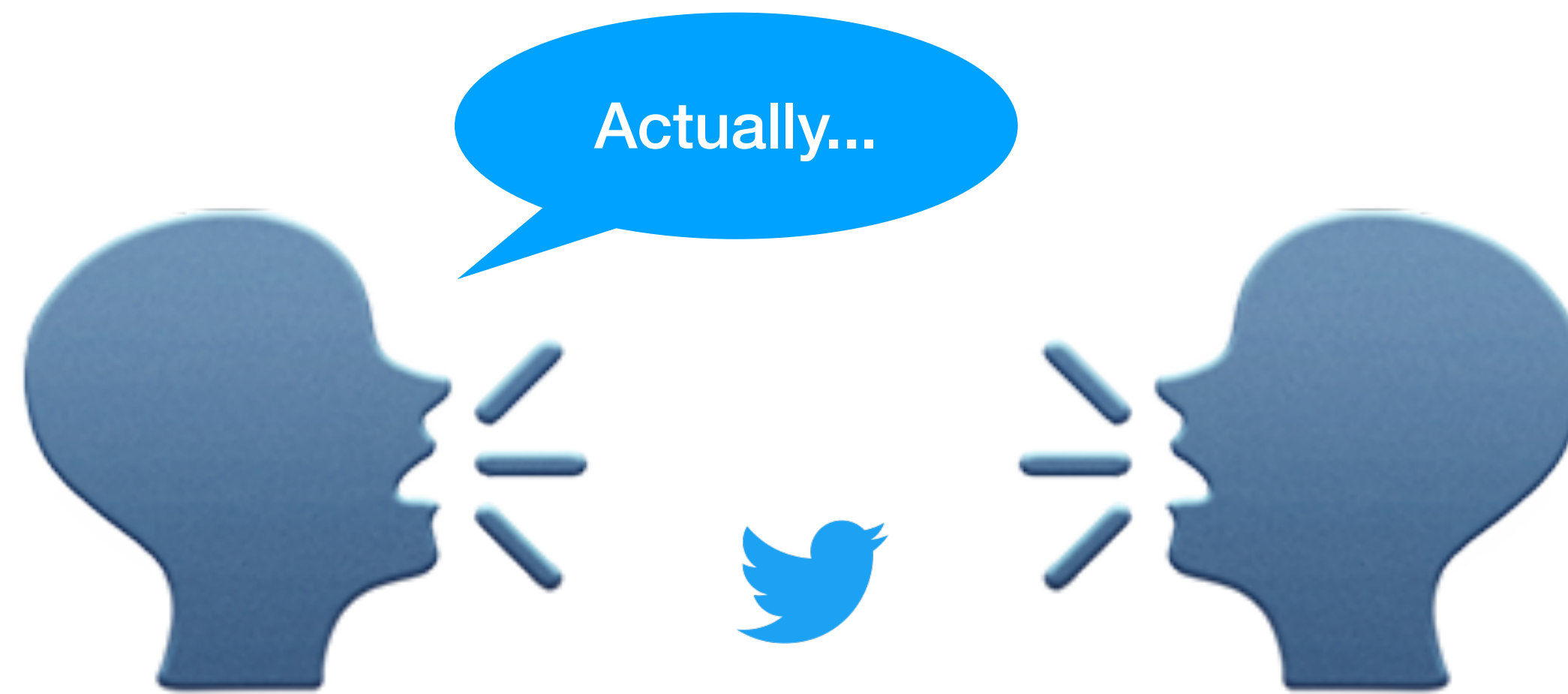


Do ask questions as we go along

Comments are welcome, too

Actually, ...

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



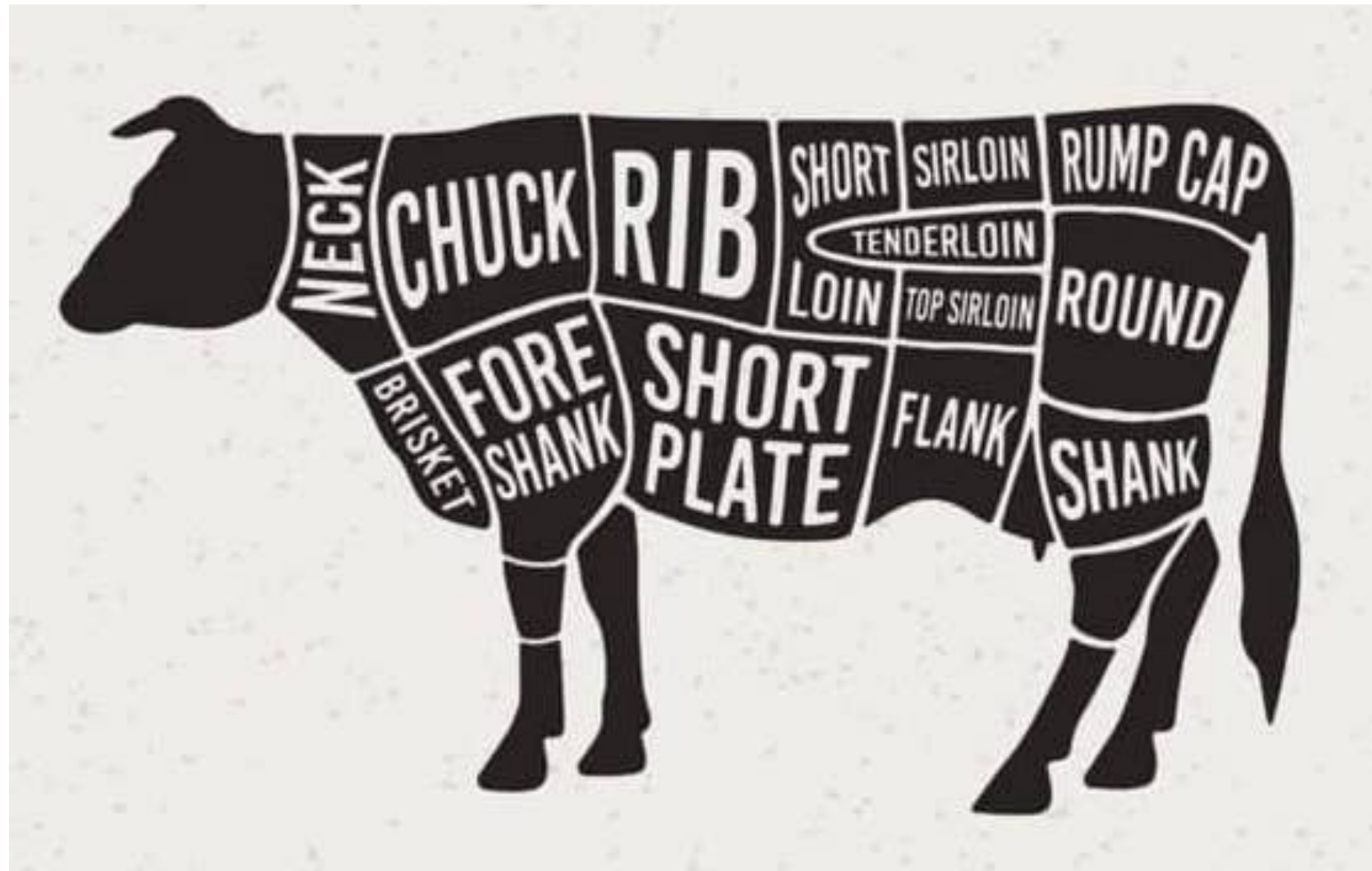
We're Different

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

🔥 Hot take typing

If it looks like a hot take, if it feels like a hot take... it probably is 😈



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

- some myths stem from **obsolete** information
- some from bad **teaching** materials

Let's talk about teaching C++

Let's talk about teaching C++

Note that a few items might seem silly/obvious to you, but I've heard them and had to discuss them with [junior devs](#) or [interns](#) we hired fresh out of college.

Or questions I usually get when [lecturing](#) at my alma mater.

The video player shows a presentation slide with the following content:

- Top left: ACCU 2021 logo
- Top right: C+ UNlverse
- Center: ACCU 2021 VIRTUAL EVENT
- Below center: C++ UNlverse
- Bottom center: Victor Ciura

The video player interface includes a video thumbnail of Victor Ciura, a progress bar at 0:35 / 26:18, and standard YouTube controls (play, volume, full screen, etc.).

#Programming #Cpp #AccuConf

C++ UNlverse - Victor Ciura [ACCU 2021]

youtube.com/watch?v=q0NJDr5hIWA



StackOverflow

How it started...

Mythbusting with Jason - unscripted improv

20K views

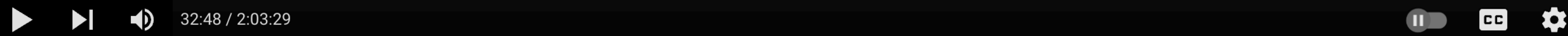
youtube.com/watch?v=Bu1AEze14Ns



```
COMPILER EXPLORER
C++ source #1
x86-64 gcc (trunk) (Editor #1, Compiler #1) C++
x86-64 gcc (trunk) -std=c++20 -Wpedantic -Wall -Wextr.
Output (0/0) x86-64 gcc (trunk) - 3231ms (3198646)
#1 with x86-64 gcc (trunk)
Wrap lines
Compiler returned: 0

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

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
20 get_optional_string_size(bool):
21    cmp    dil, 1
22    sbb   rax, rax
23    or    rax, 11
24    ret
```



C++ Mythbusting with Victor and Jason

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

566 DISLIKE SHARE DOWNLOAD CLIP SAVE ...

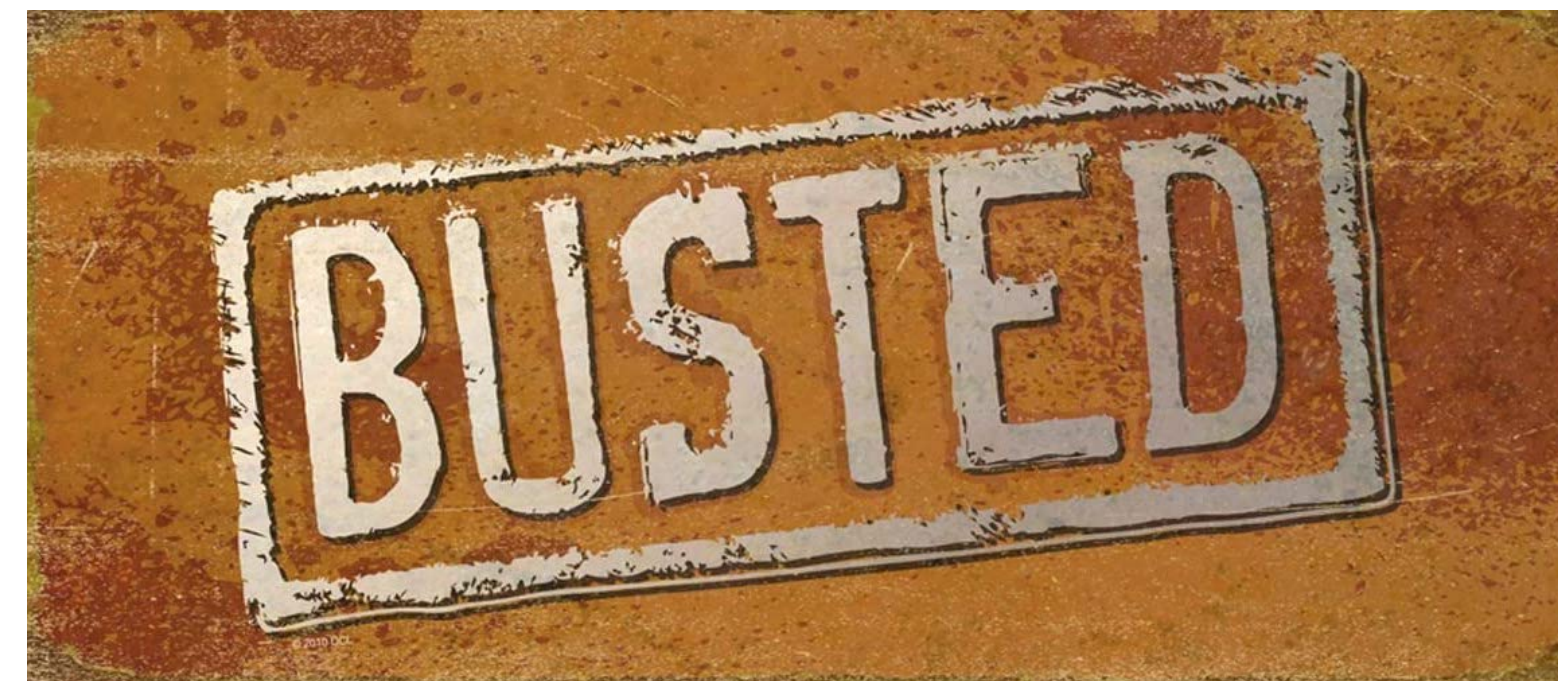
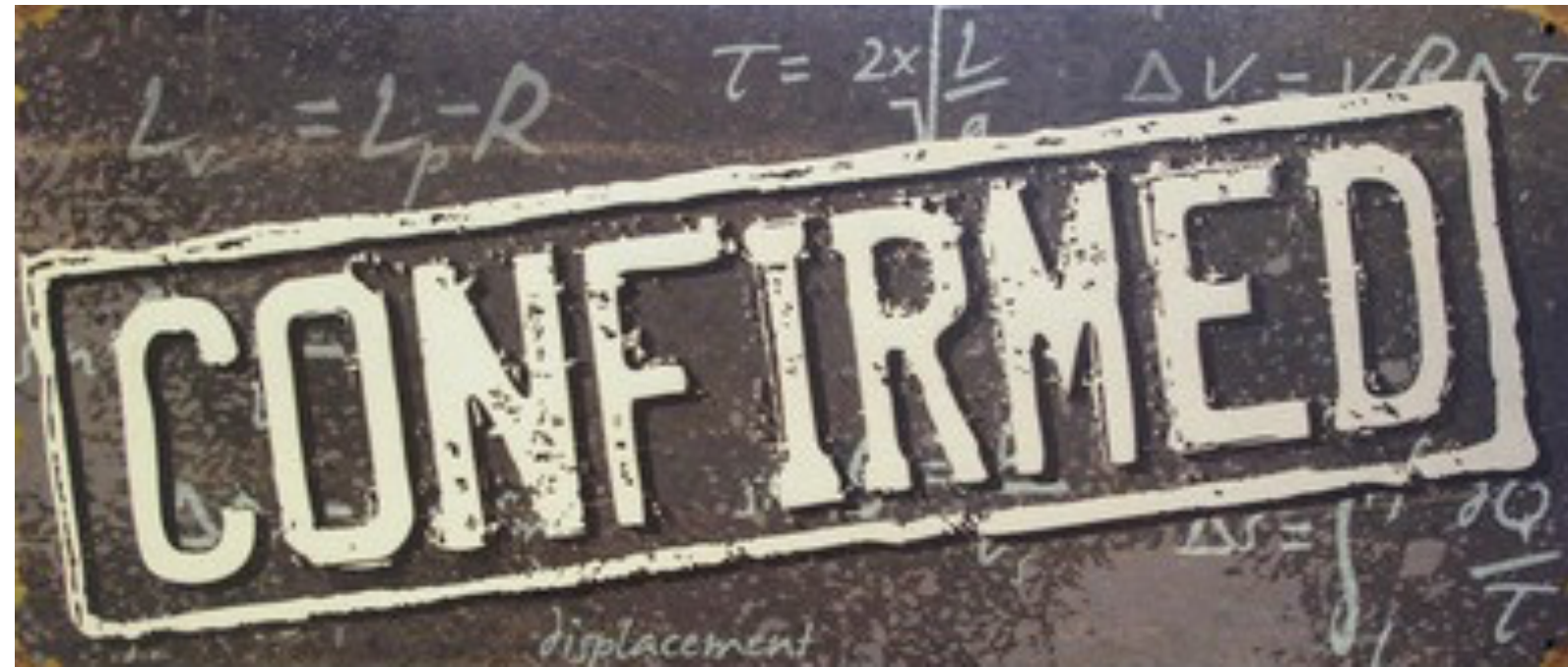
Top chat replay

for templates. I would like to require



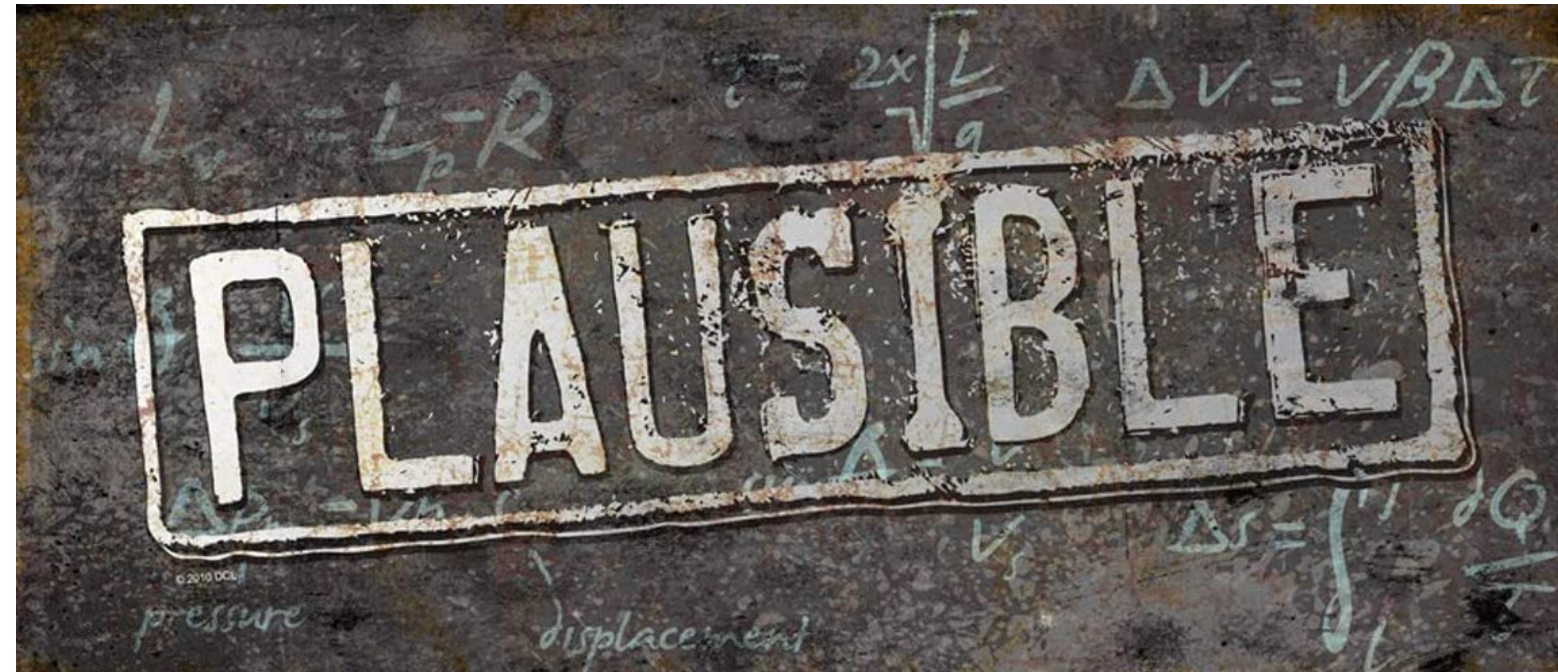
I want to instigate a healthy **dialog**,
so speak up

Verdict



A programmer's staple response:

"It depends..." 🧐



Let's test this...



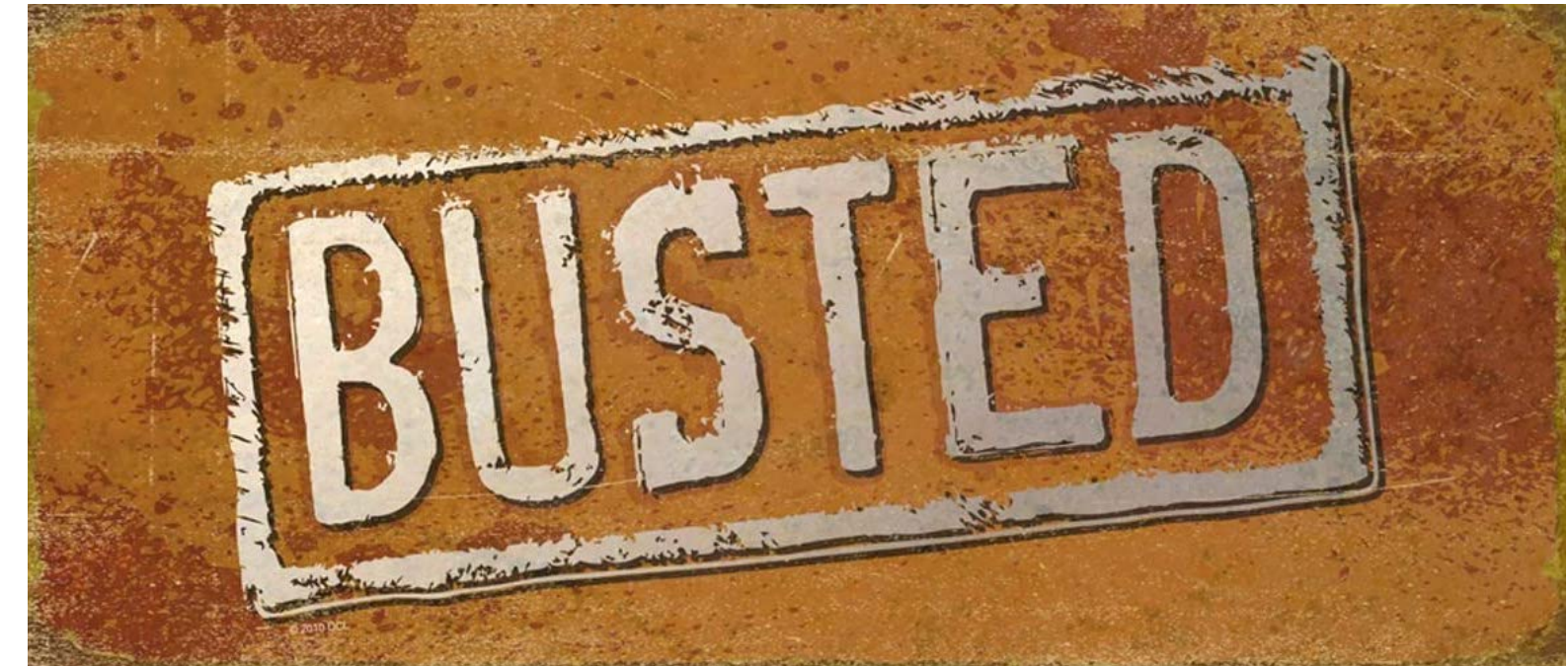
`iostreams` are slow



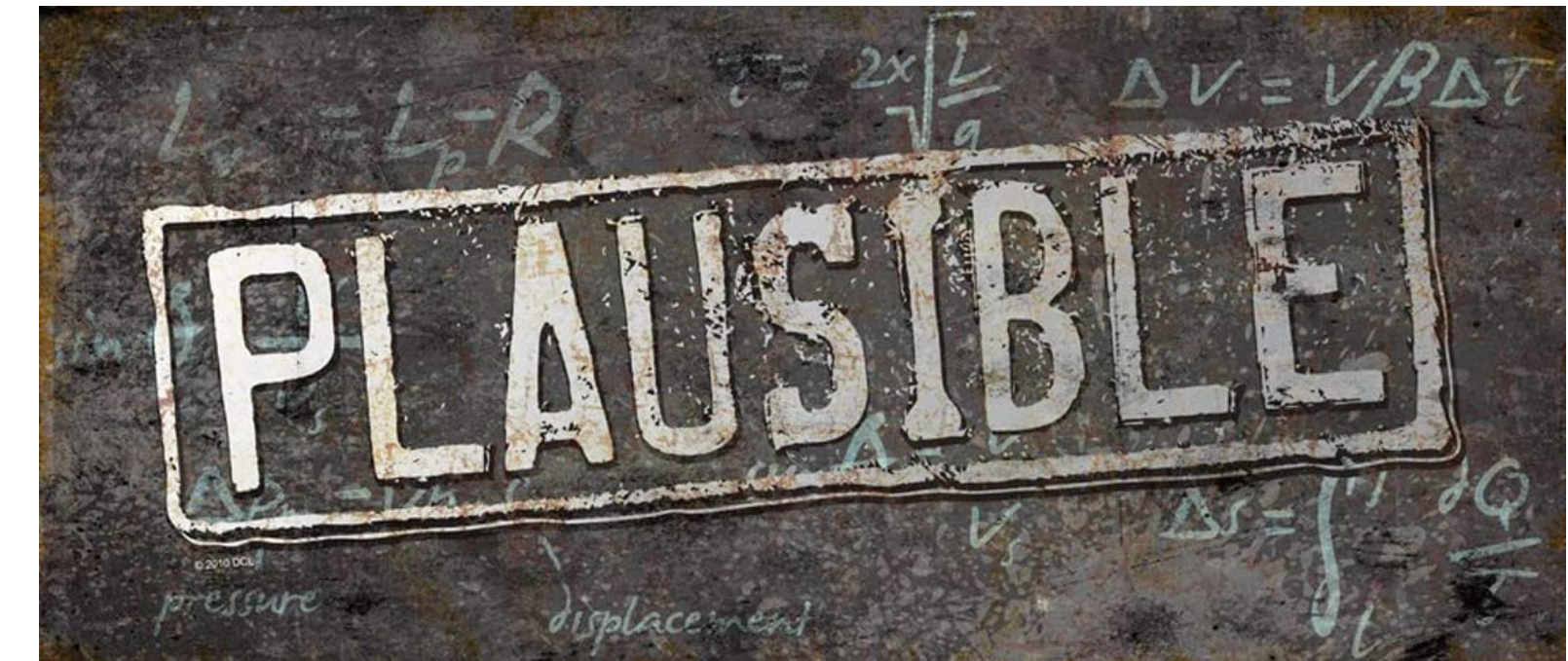
Just kidding 😊

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

C++20 **modules** will solve all C++ problems



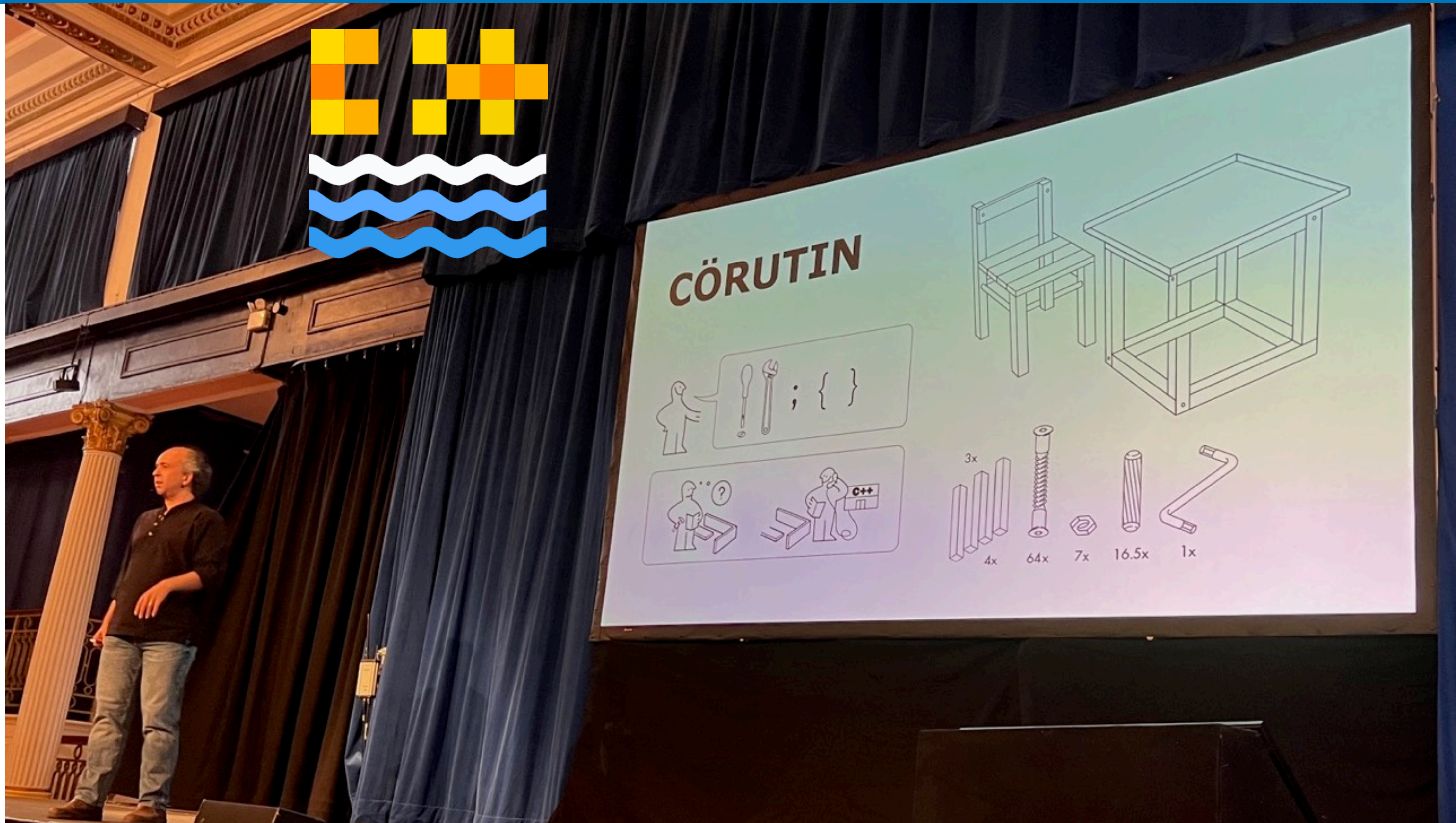
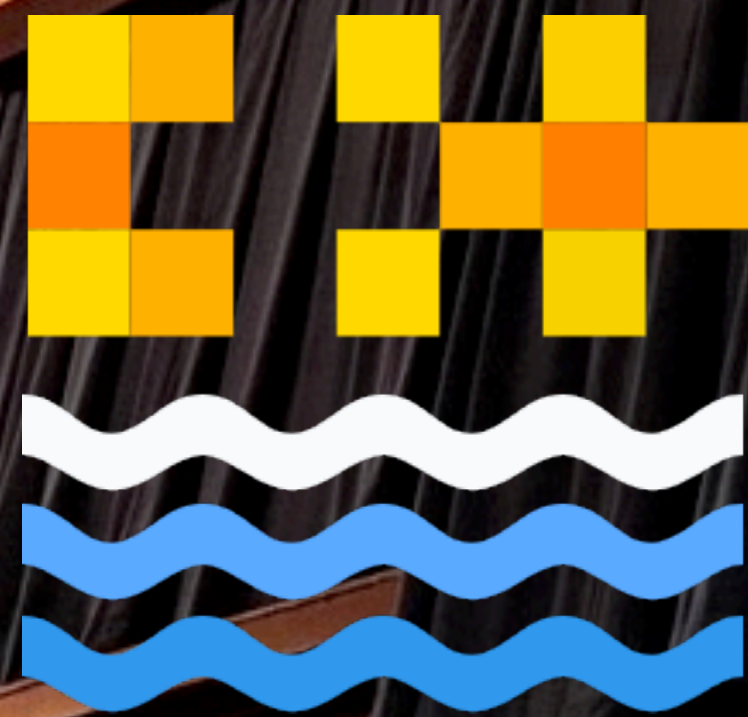
`coroutines` shipped in C++20



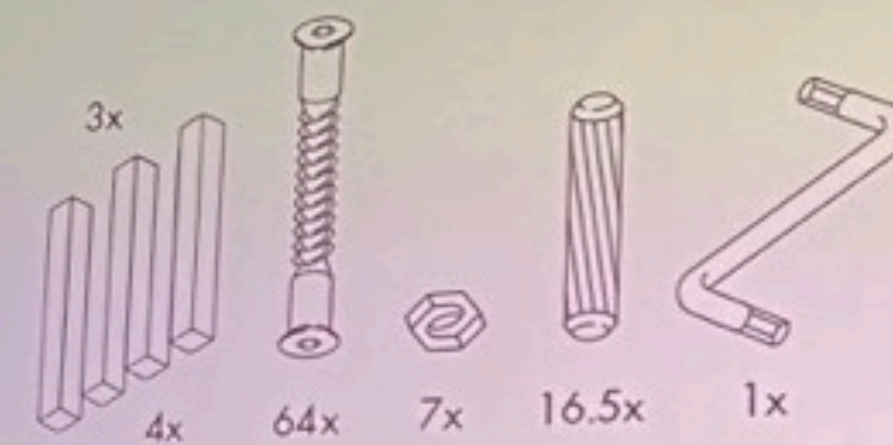
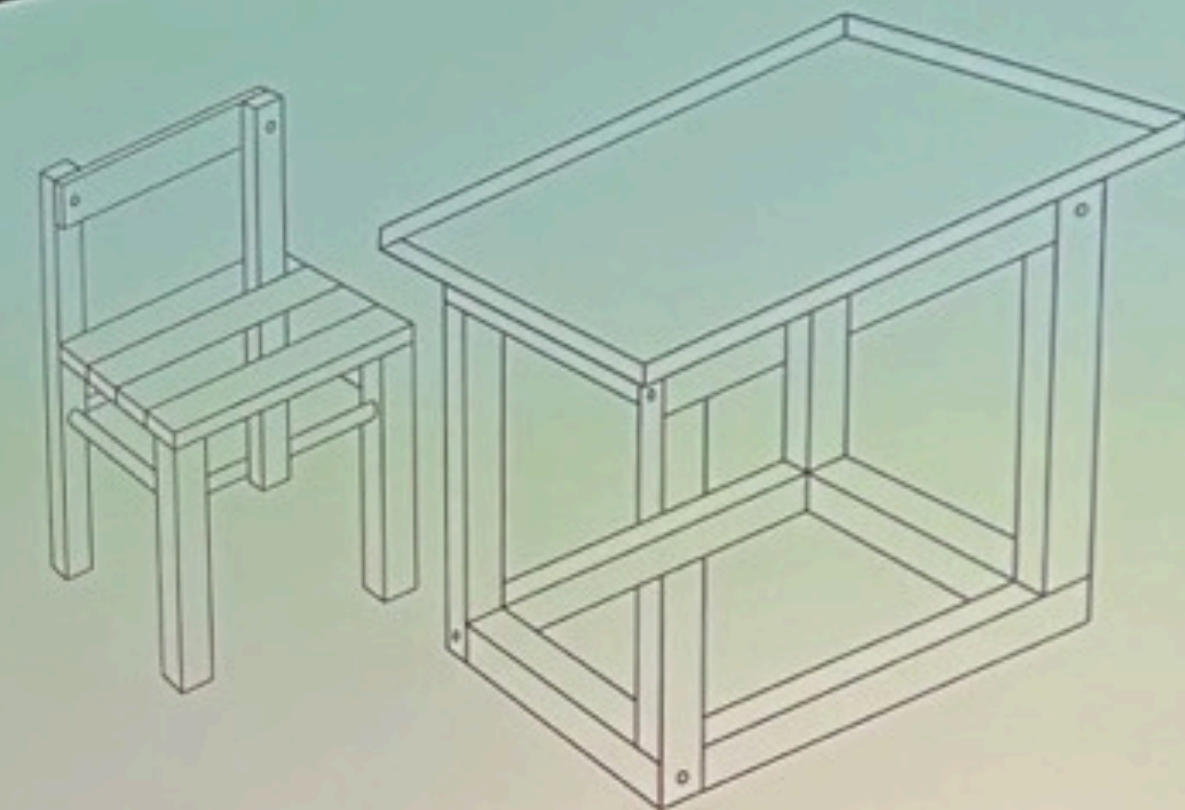
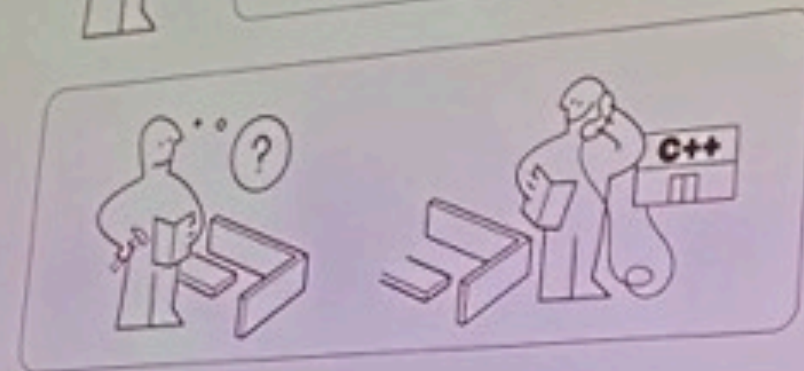
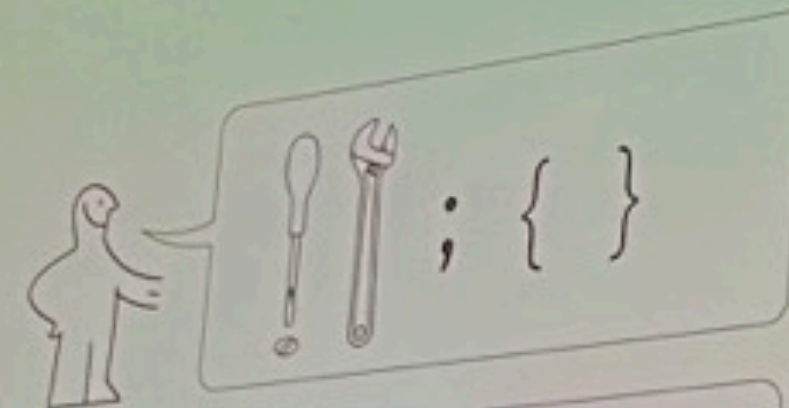
No library support 🙄

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





CÖRUTIN



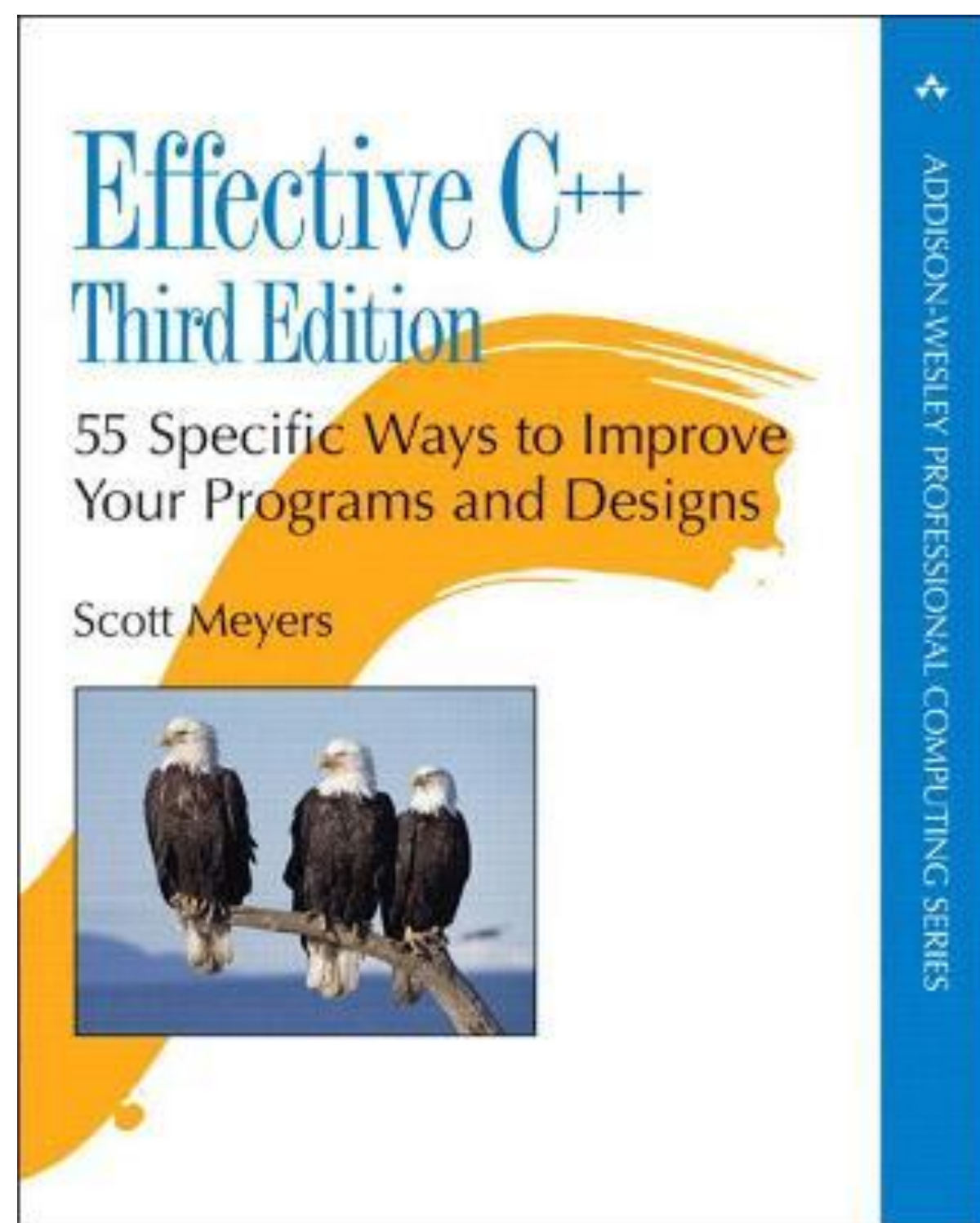
I think you got how it works



<Part 1 of N> presentation

<Part 1 of N> presentation

I invite other speakers to **join** this C++ Mythbusting series



66 Item 14

vector and string

The upshot of all this is simple. If you're dynamically allocating arrays, you're probably taking on more work than you need to. To lighten your load, use vectors or strings instead.

Item 14: Use reserve to avoid unnecessary reallocations.

One of the most marvelous things about STL containers is that they automatically grow to accommodate as much data as you put into them, provided only that you don't exceed their maximum size. (To discover this maximum, just call the aptly named `max_size` member function.) For vector and string, growth is handled by doing the moral equivalent of a `realloc` whenever more space is needed. This `realloc`-like

GotW

Let's dig in!

Myth #11

`printf/sprintf` are very fast

Myth #11

`printf/sprintf` are very fast

`sprintf` uses the **global locale**

=> **mutex lock** 😞

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

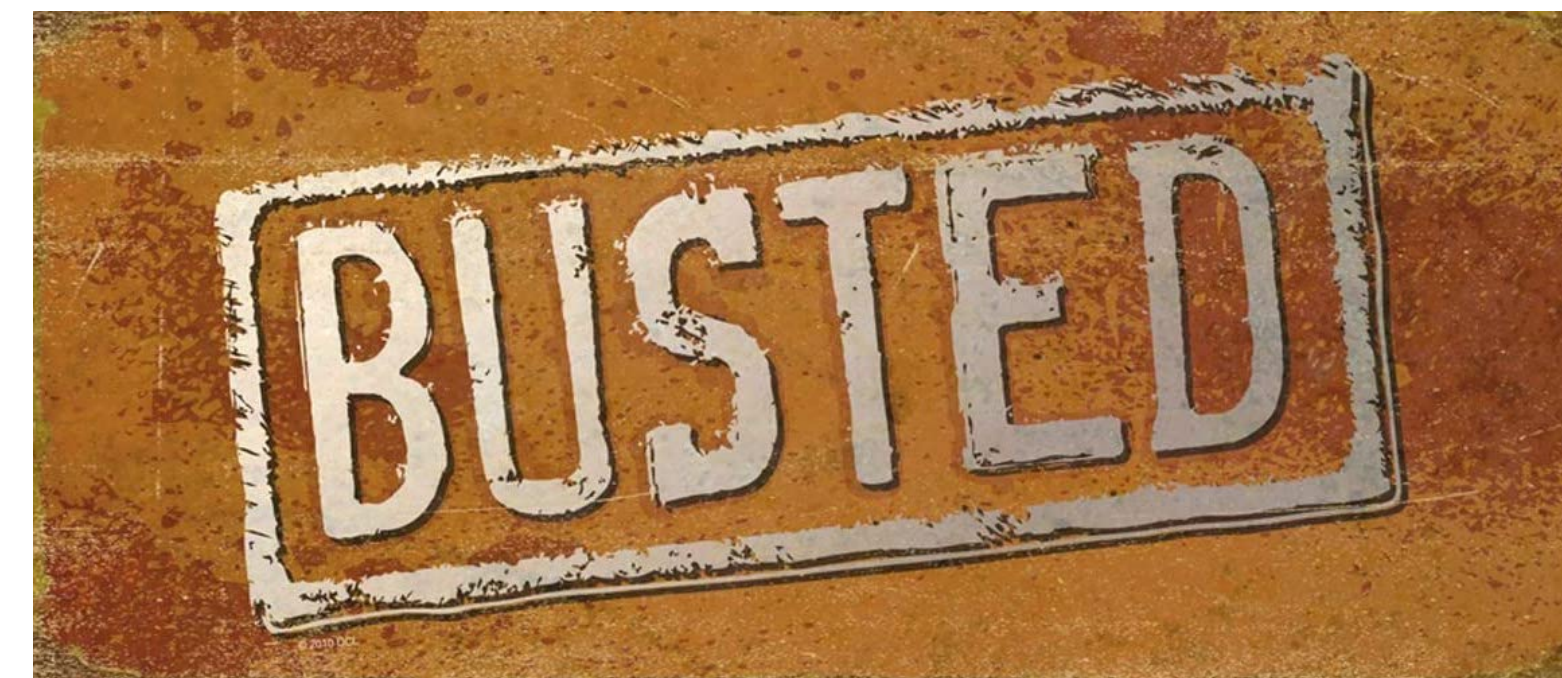
Blender case study: `sprintf` => `{fmt}`

- 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



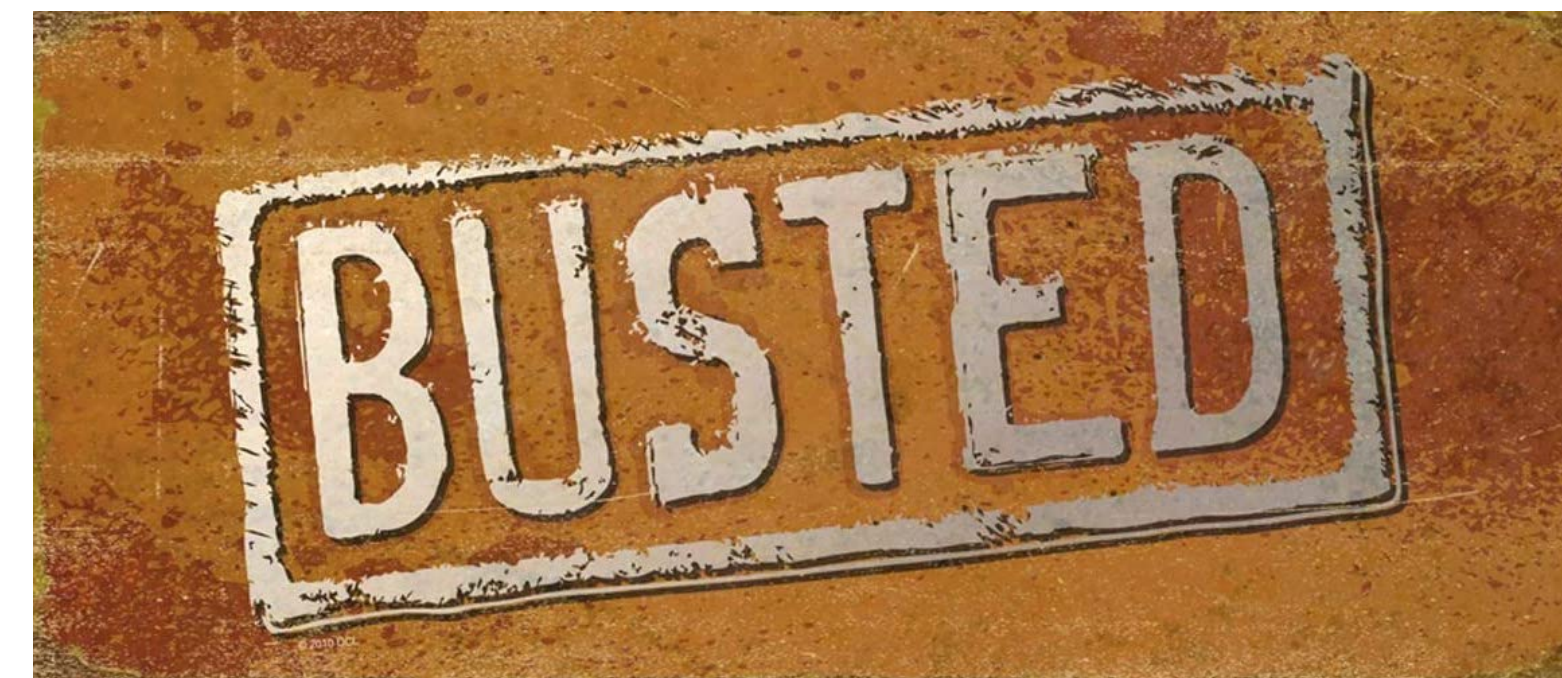
Blender case study: `sprintf` => `{fmt}`

- 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

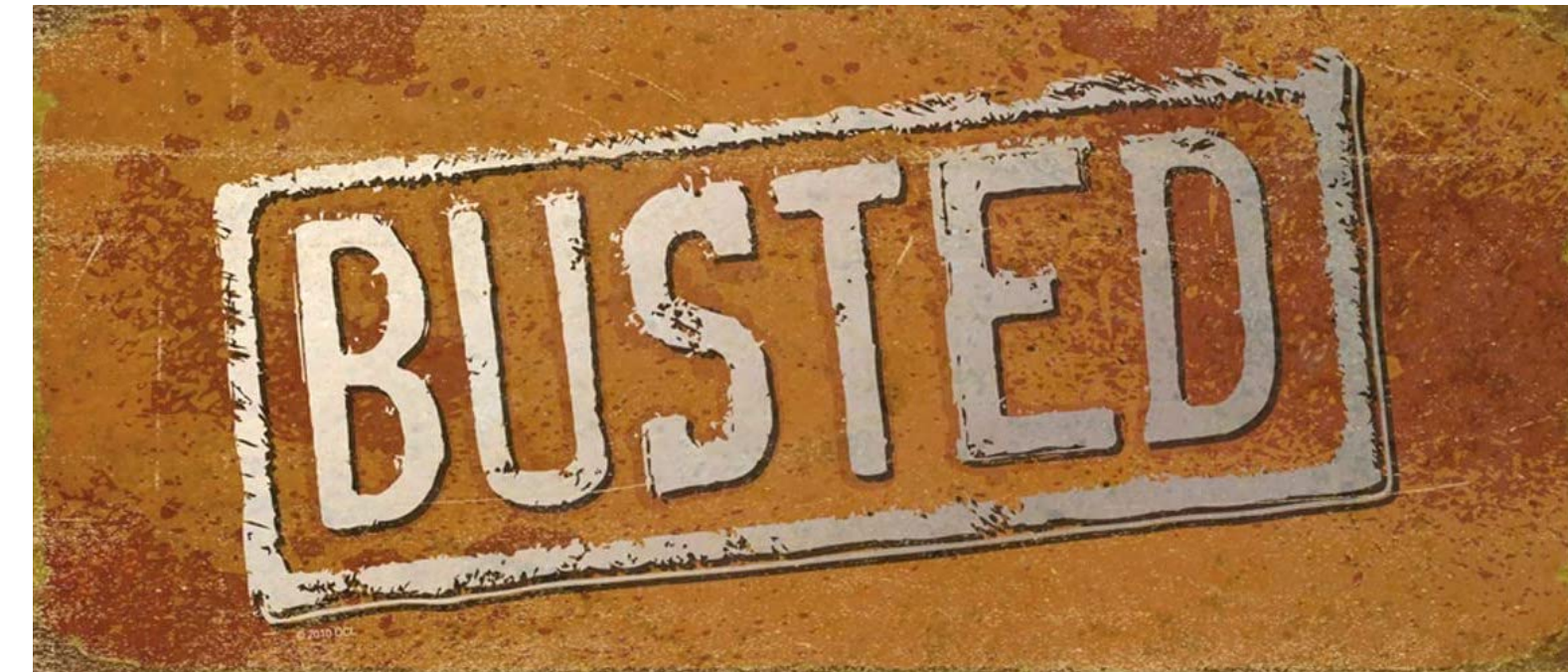


Use C++20 `std::format` or `{fmt}` library - `fmt::format()`

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

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

Humans Depend on Tools



Myth #14

C++ is not easily toolable 

I'm a tool builder



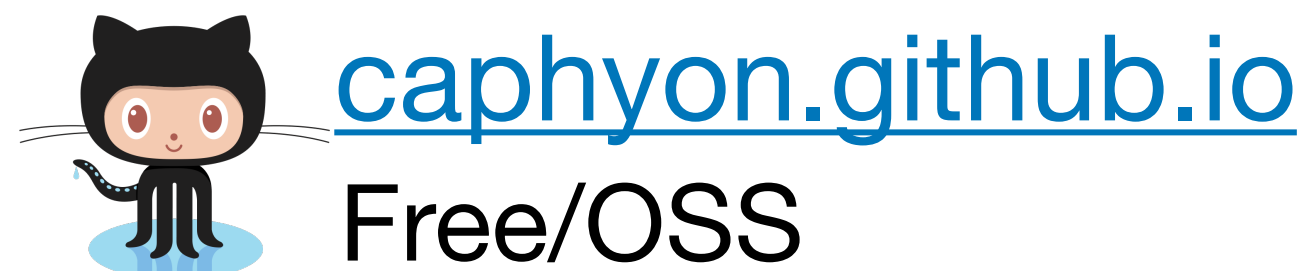
[Advanced Installer](#)



[Clang Power Tools](#)



Visual C++



Programmers Depend on Tools



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 in 2022

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.

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

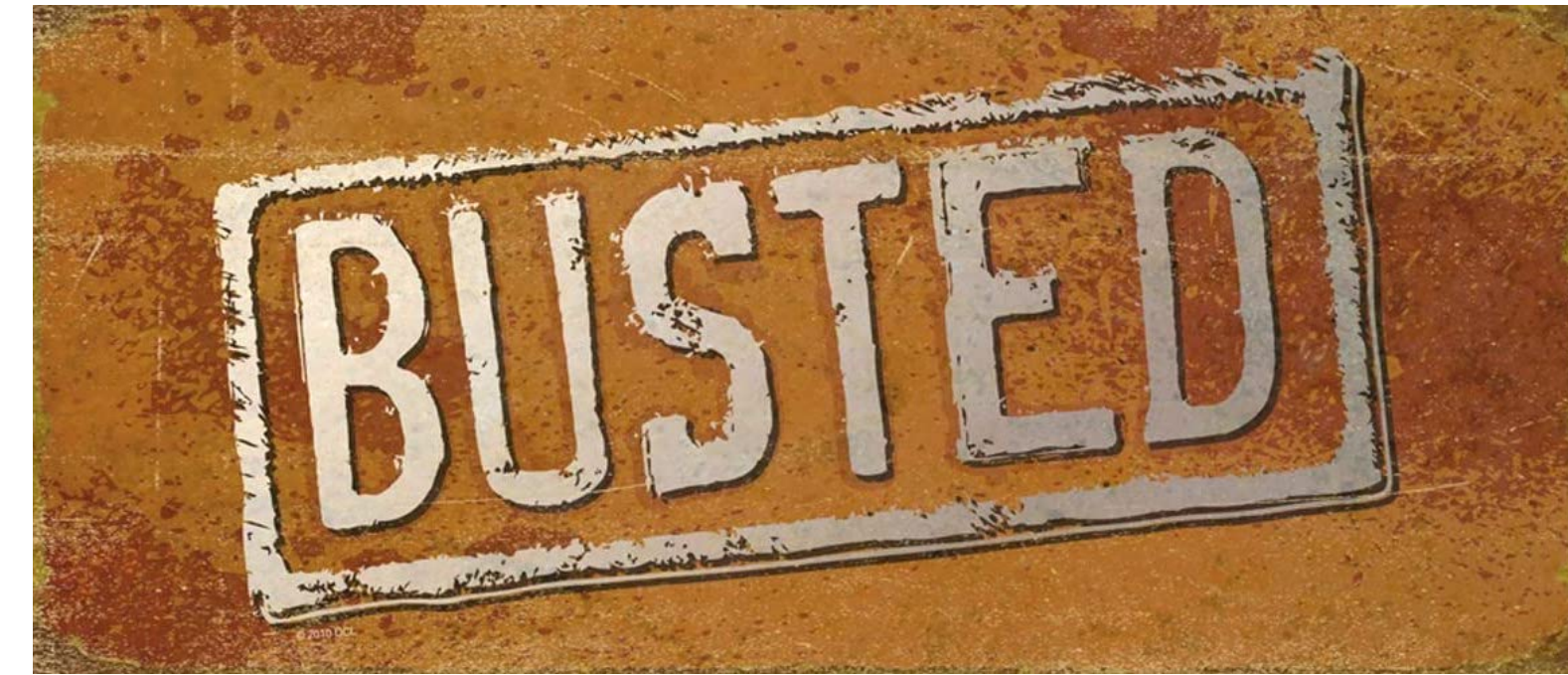
dynamic analyzer
(runtime)

SCM client

code reviews platform

+ fuzzing

C++ is not easily toolable 🛠️



Get to know your tools
well

Myth #19

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

```
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

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
and complicates APIs

Let's see...

Myth #24

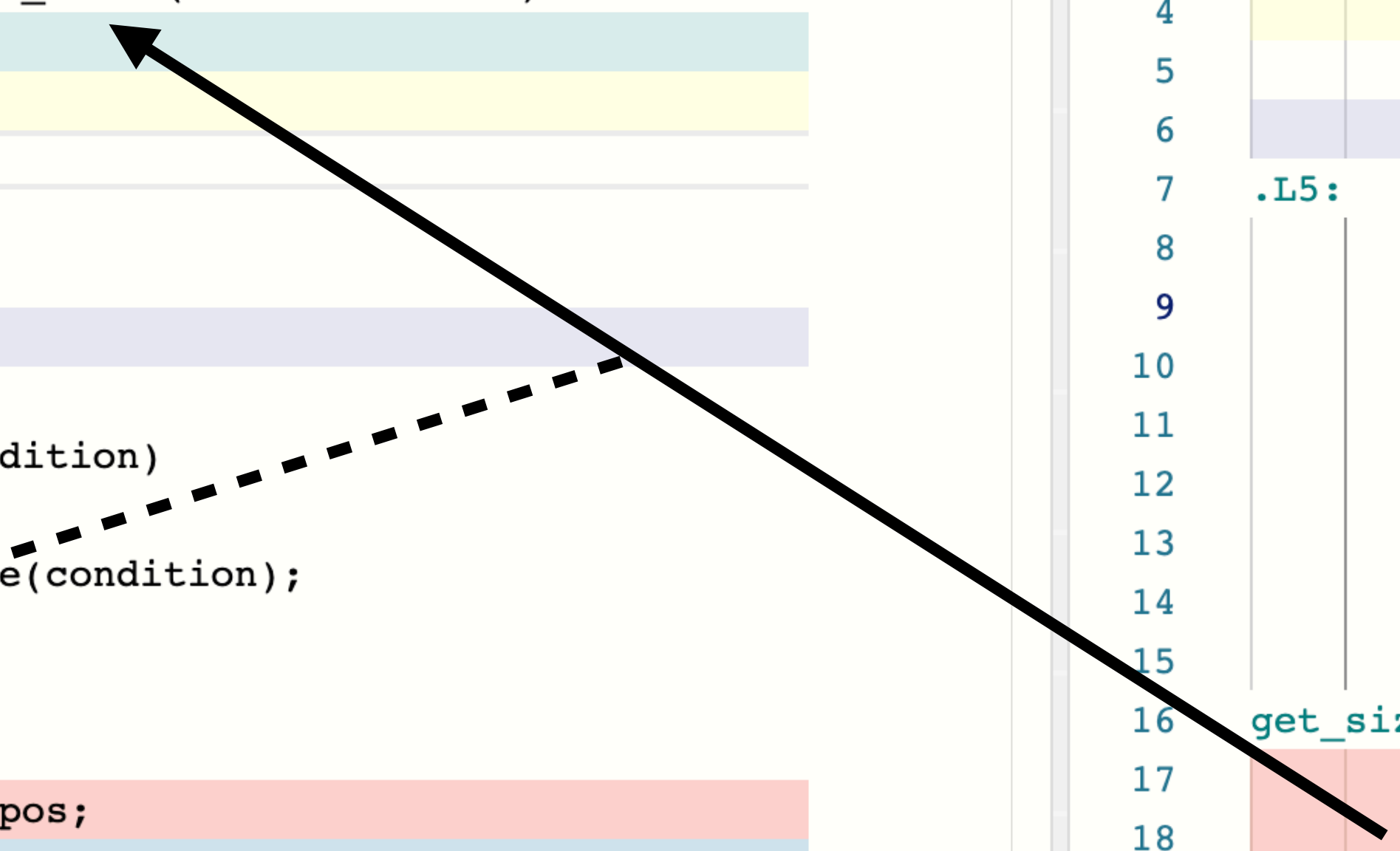
```
C++ source #1 x
A [ + v [ C++
1 #include <optional>
2 #include <cstdint>
3 #include <string>
4
5 std::optional<std::string> get_value(bool condition)
6 {
7     if (condition)
8         return "Hello";
9     else
10        return std::nullopt;
11 }
12
13 std::size_t get_size(bool condition)
14 {
15     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
```


Myth #24

```
C++ source #1 x
A [ ] + v [ ] [ ]
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 [ ] -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 [ ] - 3272ms (298483B) ~19264 lines filtered [ ]
Output of x86-64 gcc 11.2 (Compiler #1) x
A [ ] [ ] Wrap lines
```

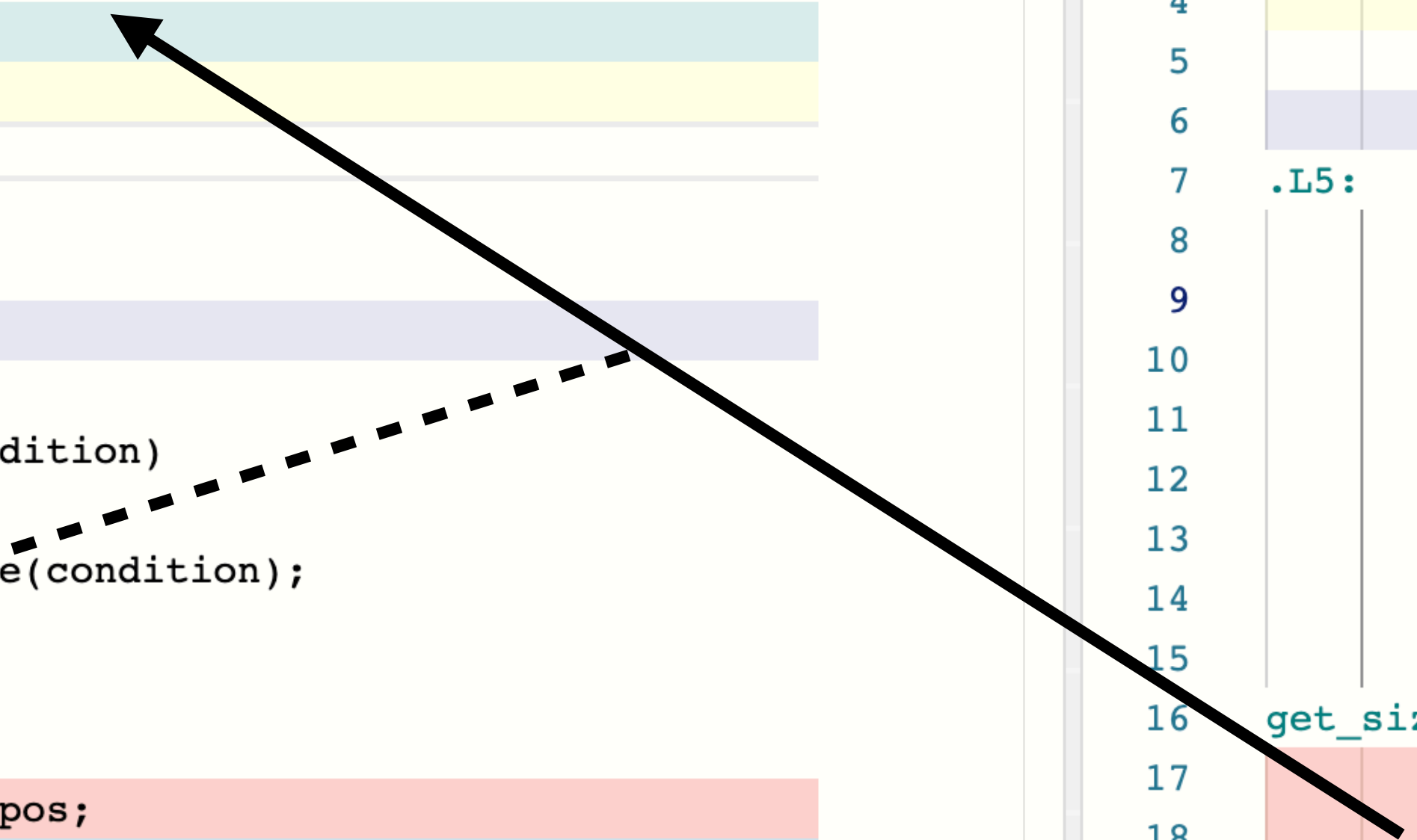


1819043144 = 0x6C6C6548

Myth #24

```
C++ source #1 x
A [ ] + v [ ] [ ]
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 [ ] -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 [ ] - 3272ms (298483B) ~19264 lines filtered [ ]
Output of x86-64 gcc 11.2 (Compiler #1) x
A [ ] [ ] Wrap lines
```



Myth #24

```
C++ source #1 x
A [ ] + [ ] [ ] [ ] [ ]
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 [ ] -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 [ ] - 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"

Myth #24

```
C++ source #1 x
A [ ] + v [ ] [ ]
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 [ ] -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 [ ] - 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



Share

C++ source #1



C++

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

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

```
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)

Myth #24



Add... More

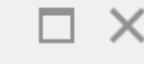
Get cool gear in the [Compiler Explorer shop](#) x [Sponsors](#)

Share

C++ source #1 X



C++



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

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

```
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



Add... More

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

C++ source #1 X



C++

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

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

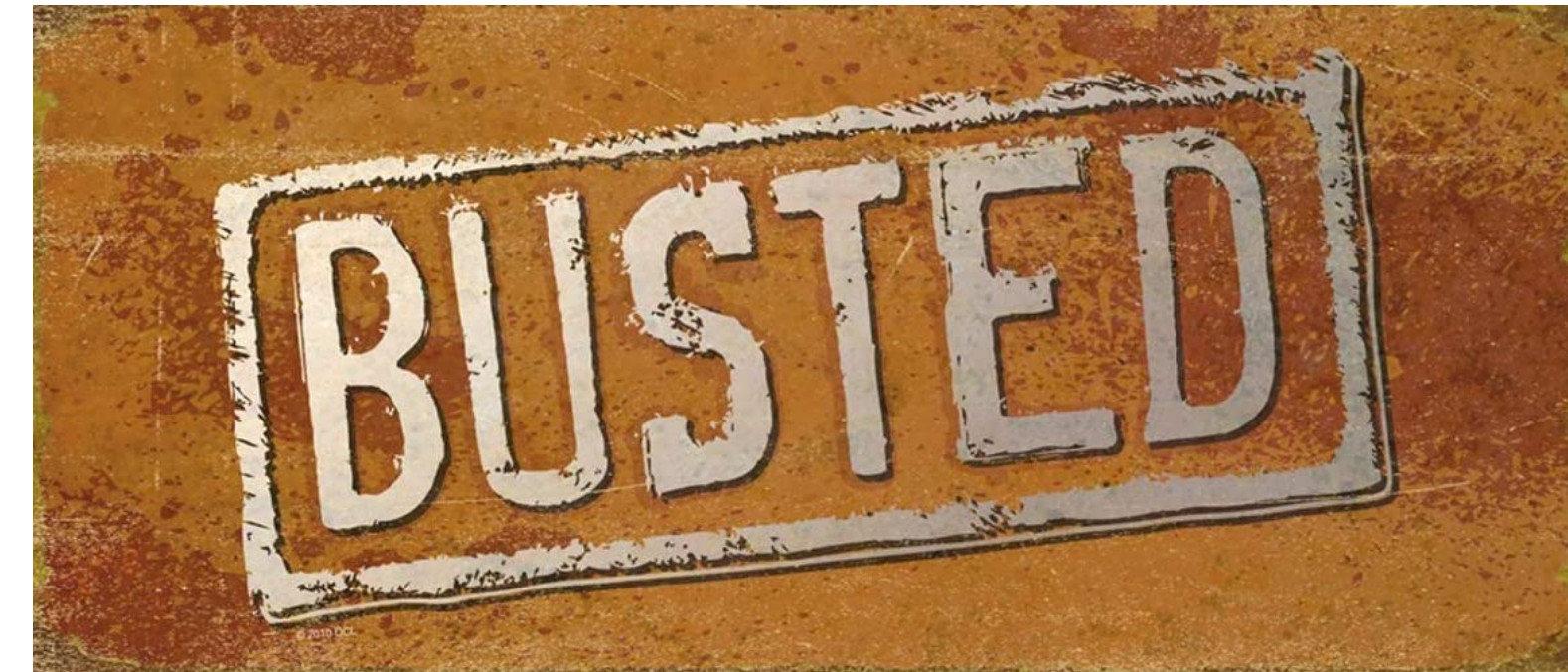
```
35 sub    rsp, 56
36 mov    edi, 24
37 lea   rax, [rsp+16]
38 mov   QWORD PTR [rsp], rax
39 call  operator new(unsigned long)
40 mov   esi, 24
41 mov   BYTE PTR [rsp+32], 0
42 movdqa xmm0, XMMWORD PTR .LC0[rip]
43 mov   DWORD PTR [rax+16], 1920234272
44 mov   rdi, rax
45 movups XMMWORD PTR [rax],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

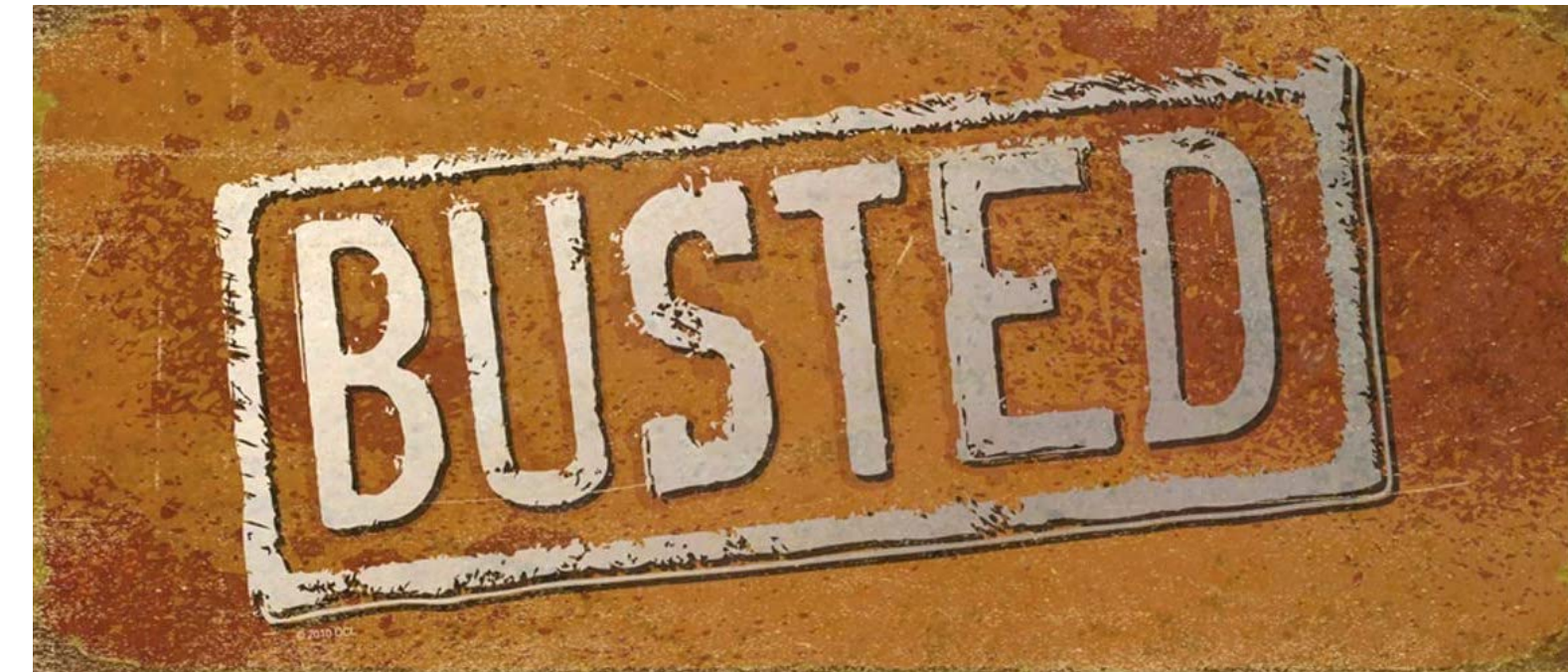
Myth #24

`std::optional` inhibits optimizations



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 }
  
```

x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)

x86-64 gcc 11.2 -O3 -std=c++20 -Wall -Wextra -Wpedantic

A [Icons] 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
  
```

```

Diff Viewer x86-64 gcc 11.2 vs x86-64 gcc 11.2
A [Icons] Left: x86-64 gcc 11.2 -O3 -std=c++20 Assembly
Right: x86-64 gcc 11.2 -O3 -std=c++20 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
80  ret
56  ret
57 main:
58  sub     rsp, 8
59  mov     edi, 1
60  call   get_size(bool)
61  add     rsp, 8
62  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+ .LC1:
91  .quad  2338328219631577172
92  .quad  8243108416984981601
  
```



```

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 }
  
```

x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)

x86-64 gcc 11.2 -O3 -std=c++20 -Wall -Wextra -Wpedantic

A 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
  
```

```

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++20 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
63 .LC0:
64-  .quad  2338328219631577172
65-  .quad  8243108416984981601

80-  ret
81 main:
82-  sub     rsp, 8
83-  mov     edi, 1
84-  call   get_size(bool)
85-  add     rsp, 8
86-  ret
87 .LC0:
88+  .quad  23
89+  .quad  23
90+ .LC1:
91-  .quad  2338328219631577172
92-  .quad  8243108416984981601
  
```



```

C++ source #1 x C++ source #2 x
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 }

```

x86-64 gcc 11.2 (C++, Editor #2, Compiler #2)

x86-64 gcc 11.2 -O3 -std=c++20 -Wall -Wextra -Wpedantic

```

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

```

```

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++20 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

```

~40% more instructions


```

C++ source #1 X C++ source #2 X
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

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*


`std::optional` complicates APIs

`std::optional` complicates APIs

- don't look inside the  `box`
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:
Haskell (`Maybe`) or Rust (`Option<T>`)

Myth #24


`std::optional` complicates APIs

- don't look inside the  `box`
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:
Haskell (`Maybe`) or Rust (`Option<T>`)



Myth #24

`std::optional` complicates APIs

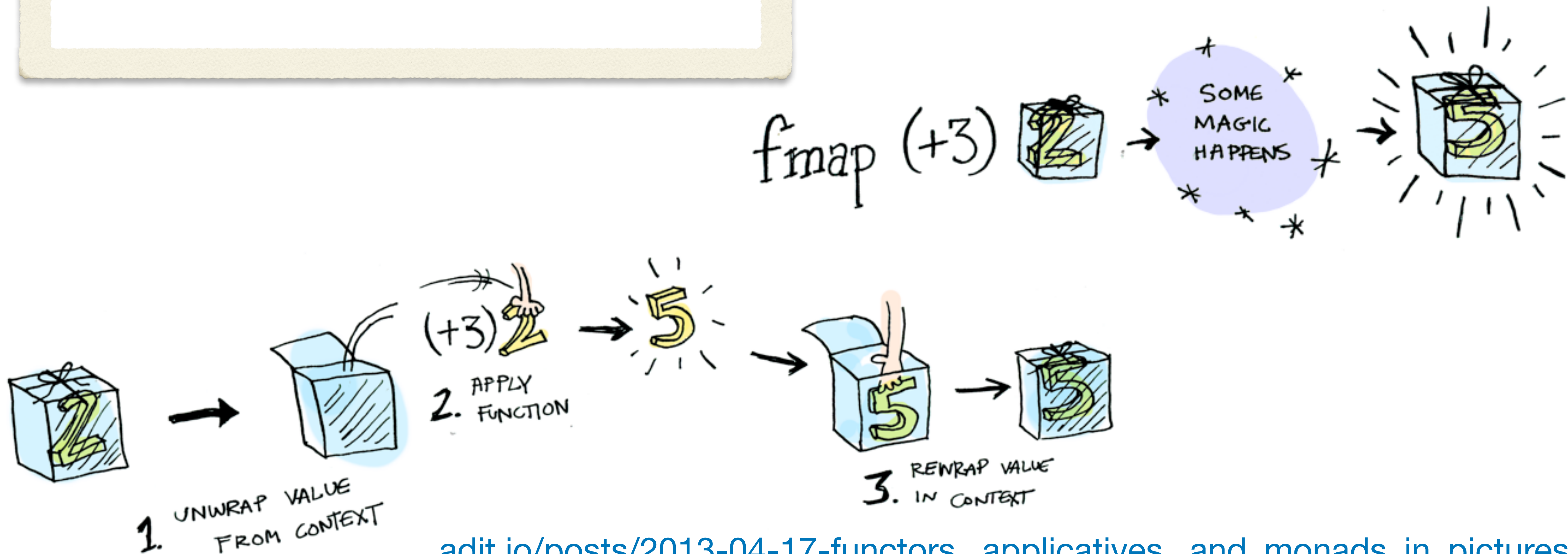
- don't look inside the  `box`
- don't use optional for error handling
- when in doubt, draw inspiration from other APIs:
Haskell (`Maybe`) or Rust (`Option<T>`)



adit.io/posts/2013-04-17-functors, applicatives, and monads in pictures

Myth #24

`std::optional` complicates APIs



adit.io/posts/2013-04-17-functors, applicatives, and monads in pictures

Myth #24

```
optional<T> f()
```

if / else

```
optional<T> g(optional<T> in)
```

if / else

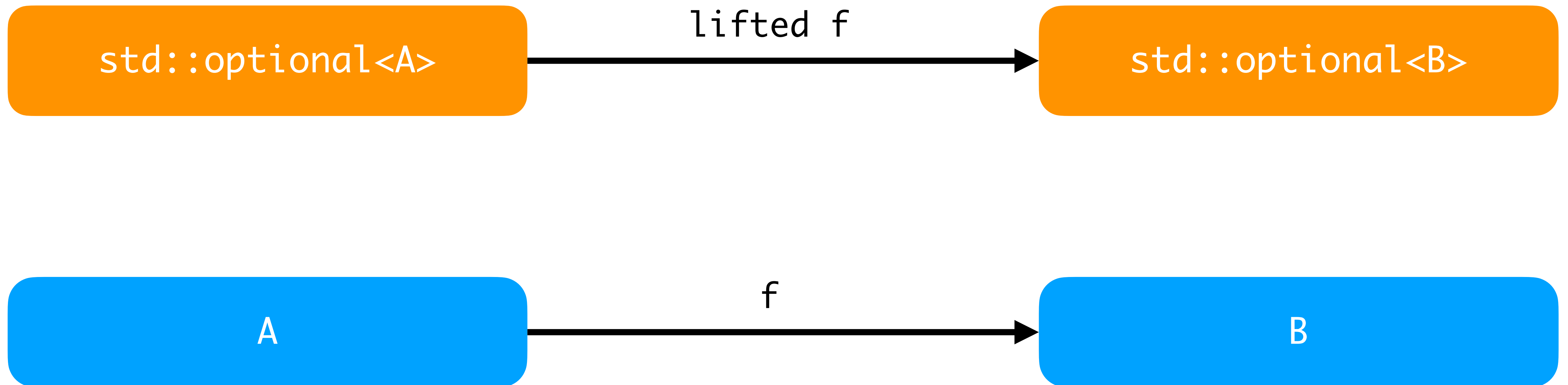
```
optional<T> h(optional<T> in)
```

 don't look inside the  box

Boxes 

Lift 

Lifting any function



Lifting any function

Lifted `f` operates on `optional<A>` and produces `optional`

```
template<class A, class B>
optional<B> fmap(function<B(A)> f, const optional<A> & o)
{
    optional<B> result;
    if (o)
        result = f(*o); // wrap a <B>

    return result;
}
```

Lifting any function

Lifted `f` operates on `optional<A>` and produces `optional`

```
template<typename T, typename F>
auto fmap(const optional<T> & o, F f) -> decltype( f(o.value()) )
{
    if (o)
        return f(o.value());
    else
        return {}; // std::nullopt
}
```




The Imperatives Must Go

VICTOR CIURA

Cppcon
The C++ Conference

2022
September 12th-16th

#cppcon #programming #cpp

The Imperatives Must Go! [Functional Programming in Modern C++] - Victor Ciura - CppCon 2022

The image shows a YouTube video player interface. The video title is "The Imperatives Must Go" by Victor Ciura. The video is from CppCon 2022, which took place from September 12th to 16th. The video player shows a play button in the center, a progress bar at 0:00 / 57:33, and various control icons like volume, closed captions, settings, and full screen. The video player is set to HD quality. The video player is embedded in a slide with a blue header and footer.

youtube.com/watch?v=M5HuOZ4sgJE

Monadic `std::optional` (C++23 P0798)

```
optional<int> string_view_to_int(string_view sv)
{
    const auto first = sv.data();
    const auto last  = first + sv.size();

    int val = -1;
    const auto result = std::from_chars(first, last, val);

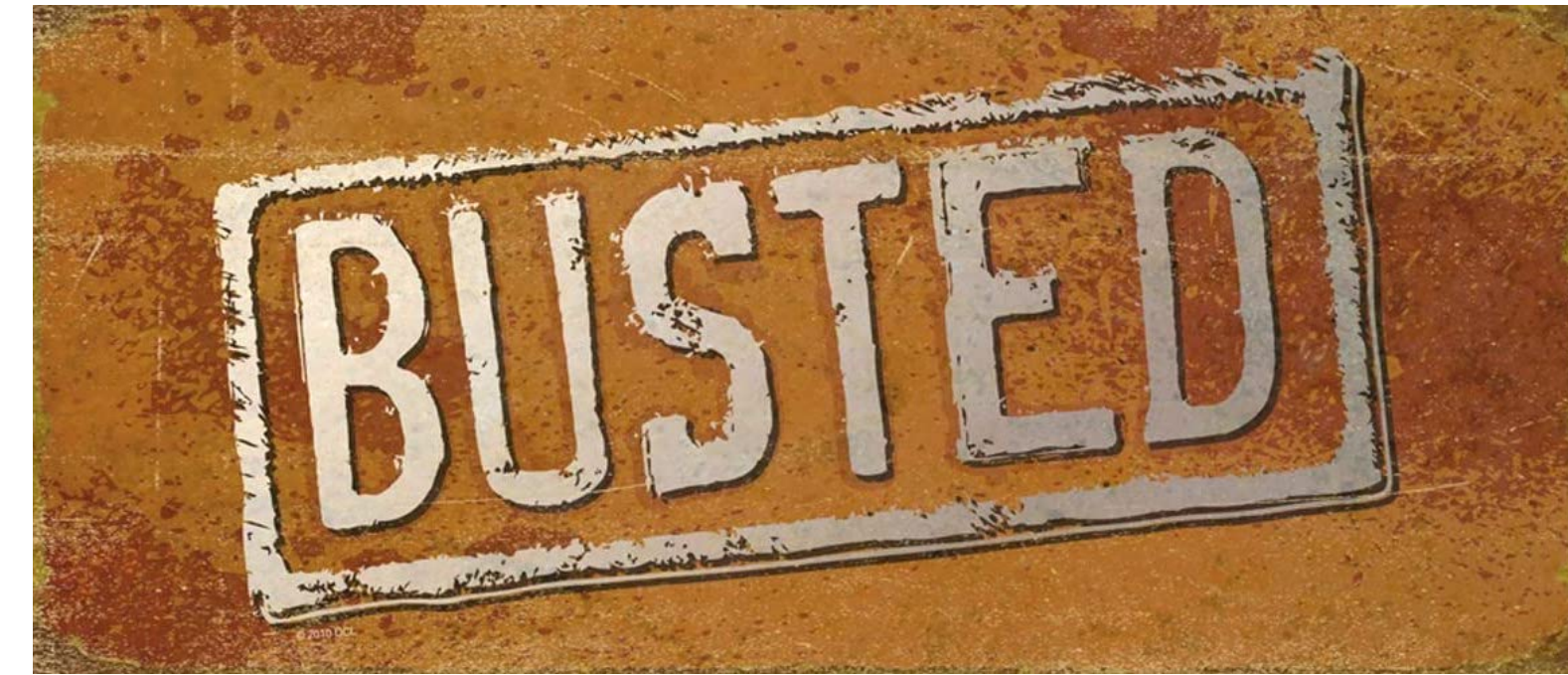
    if (result.ec == errc{} && result.ptr == last)
        return val;
    else
        return nullopt;
}
```


Monadic `std::optional` (C++23 P0798)

```
cout << string_view_to_int(sv)
    .and_then( [=](int val) -> optional<int> {
        const int logs = clamp(val, 0, max_logs);
        if (logs > 0)
            return logs;
        else
            return std::nullopt;
    })
    .transform( [](int val) {
        return std::format("Collecting in {} logs.", val);
    })
    .or_else( [] {
        return optional<string>{"Log error"};
    })
    .value()
```

Myth #24

`std::optional` complicates APIs



`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 ?

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);
}
```

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);
}
```

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));
}
```



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));
}
```



```
'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));
}
```


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));
}
```

```
'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))`

`'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));
}
```

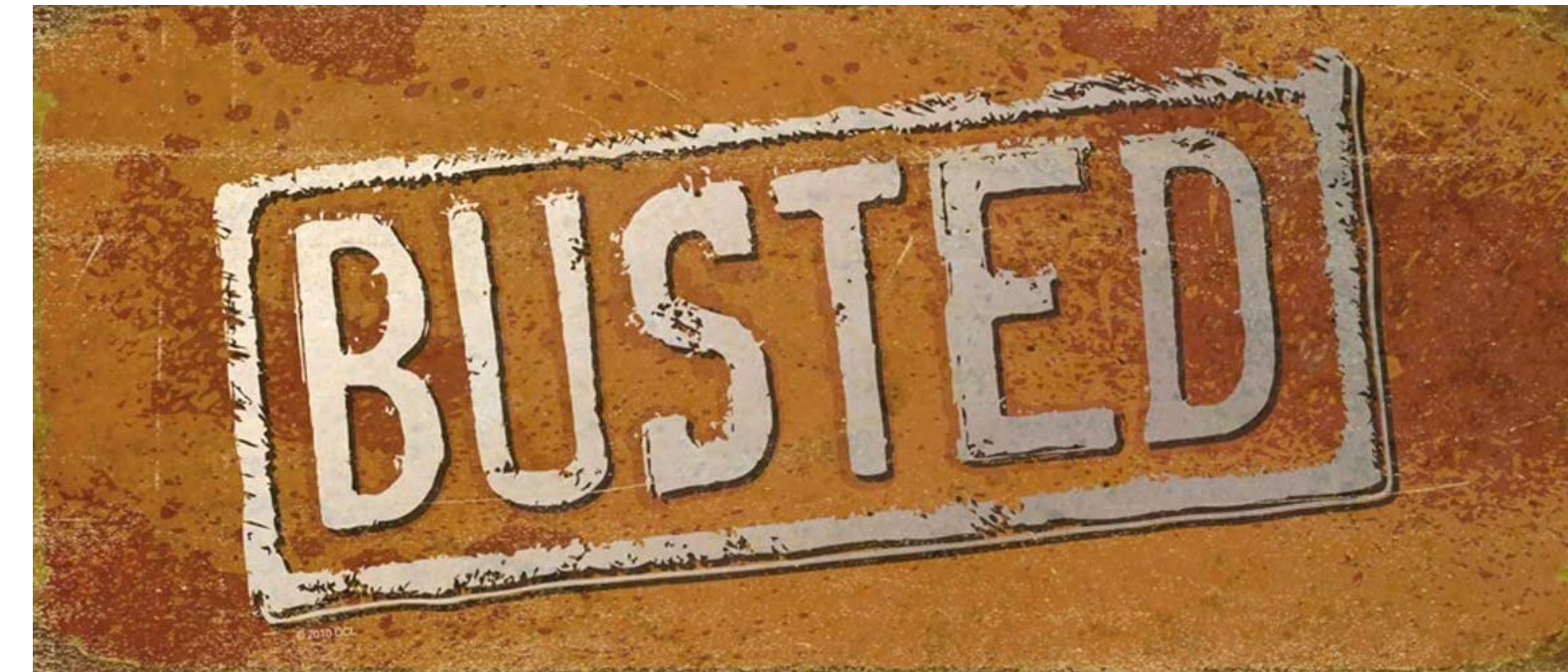
`string(std::move(greeting))`

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);
```

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

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);
    }
};
```

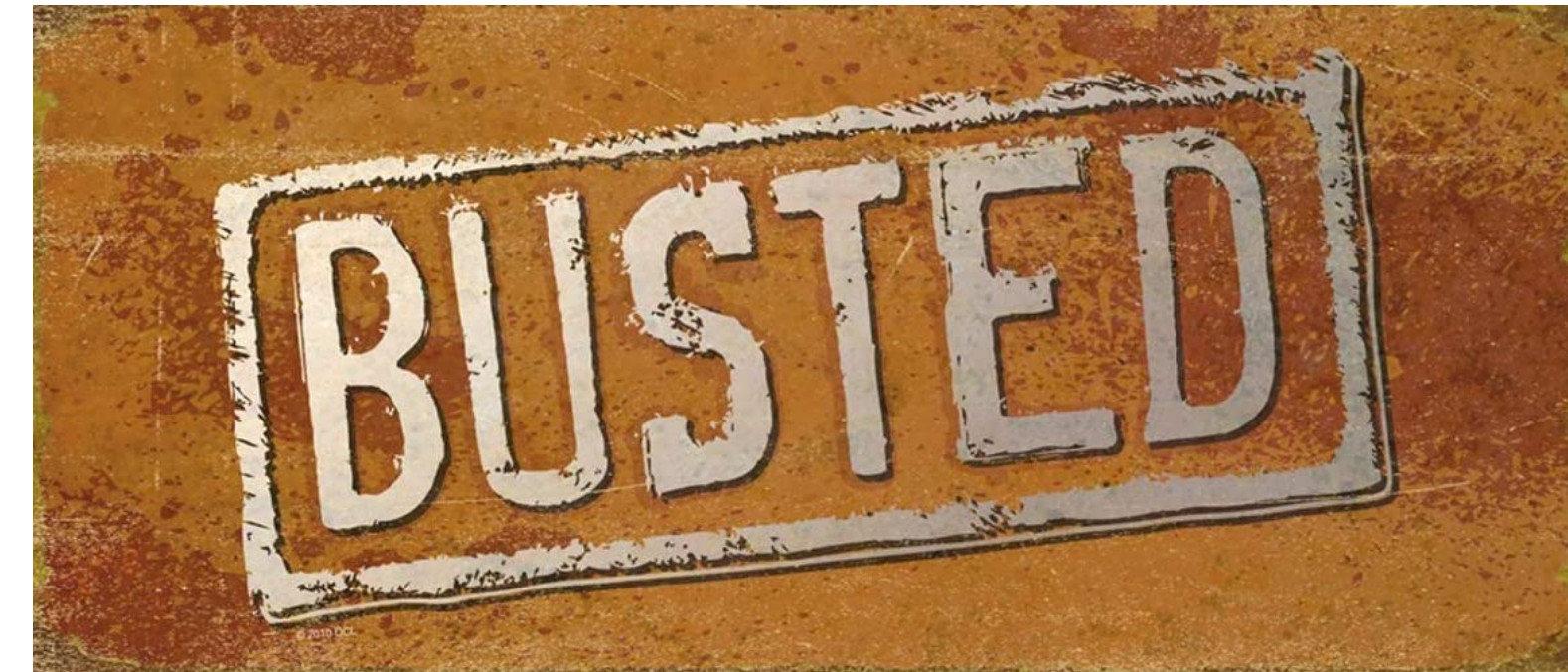
```
Widget w;
std::string name{"Hello from a long string"};
w.set_name(name);
```

Technically, more efficient
(one less move operation)

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

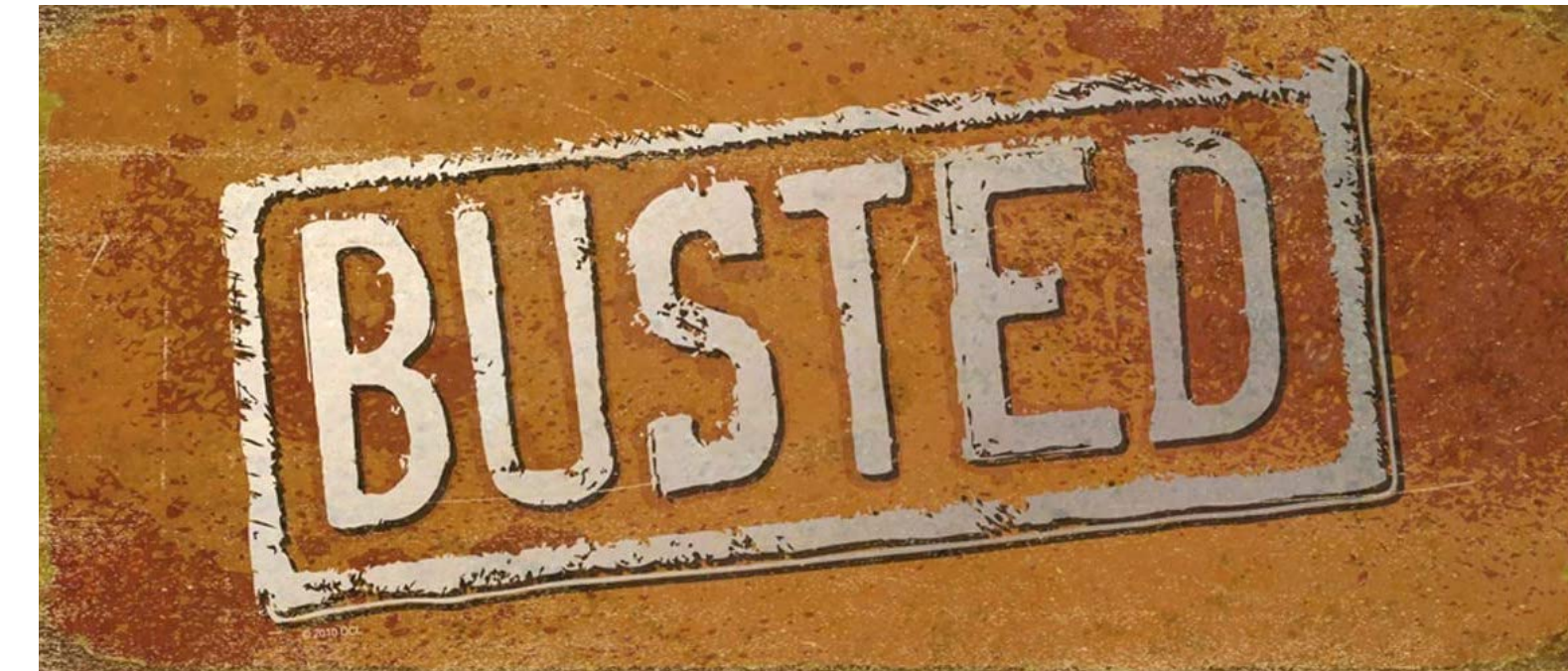
Myth #36

Always pass input arguments
by const reference



Myth #36

Always pass input arguments
by const reference

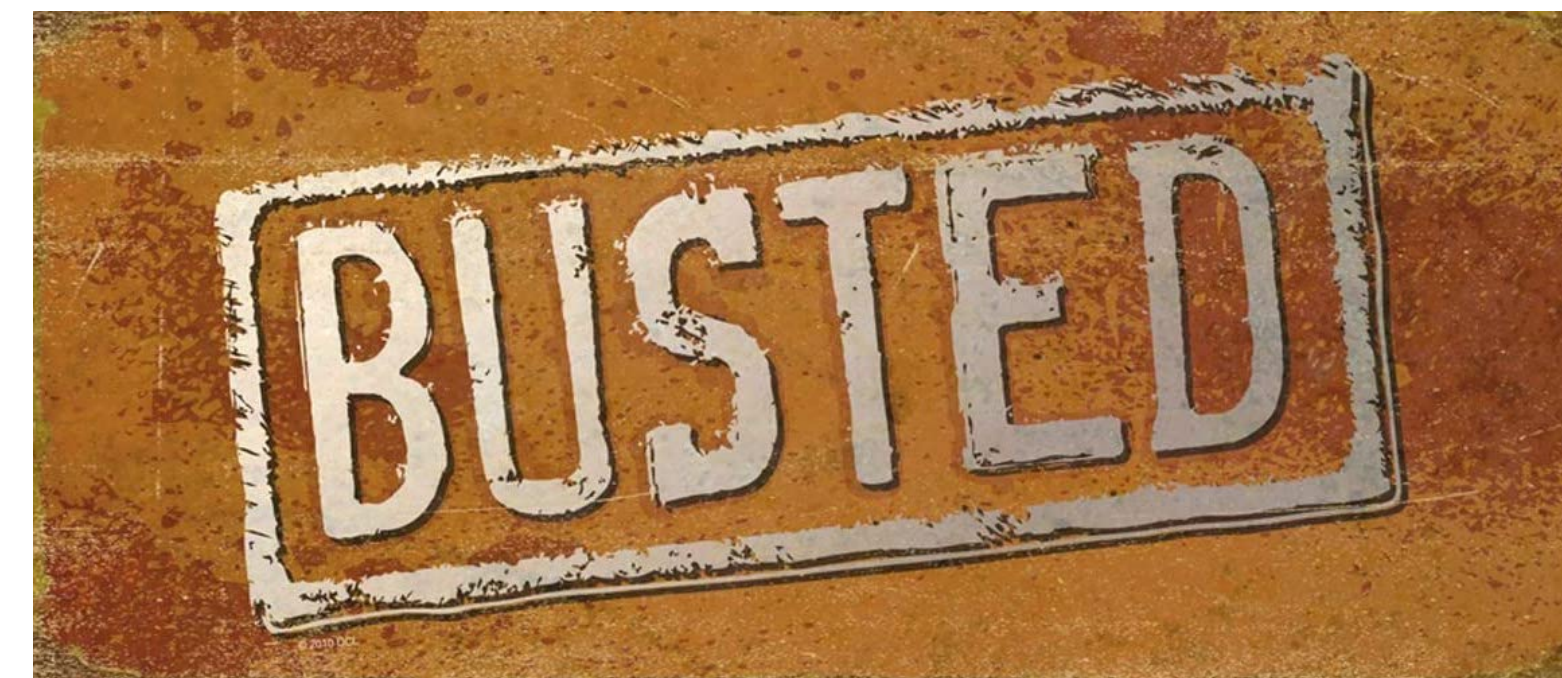


There's even a [clang-tidy modernizer](#) check to perform this transformation [automatically](#), at scale

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

Myth #36

Always pass input arguments
by const reference



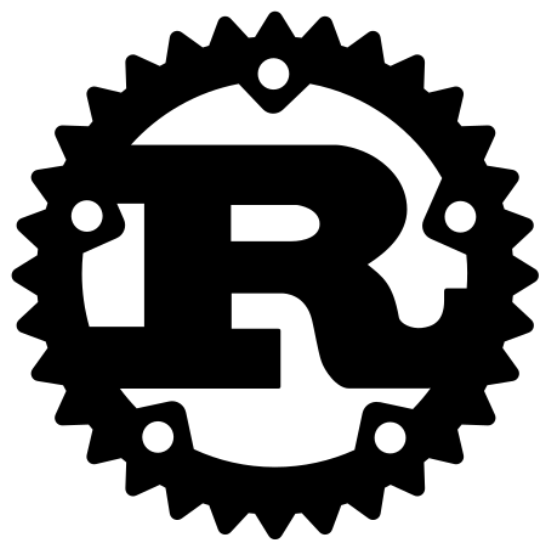
There's even a [clang-tidy modernizer](#) check to perform
this transformation [automatically](#), at scale



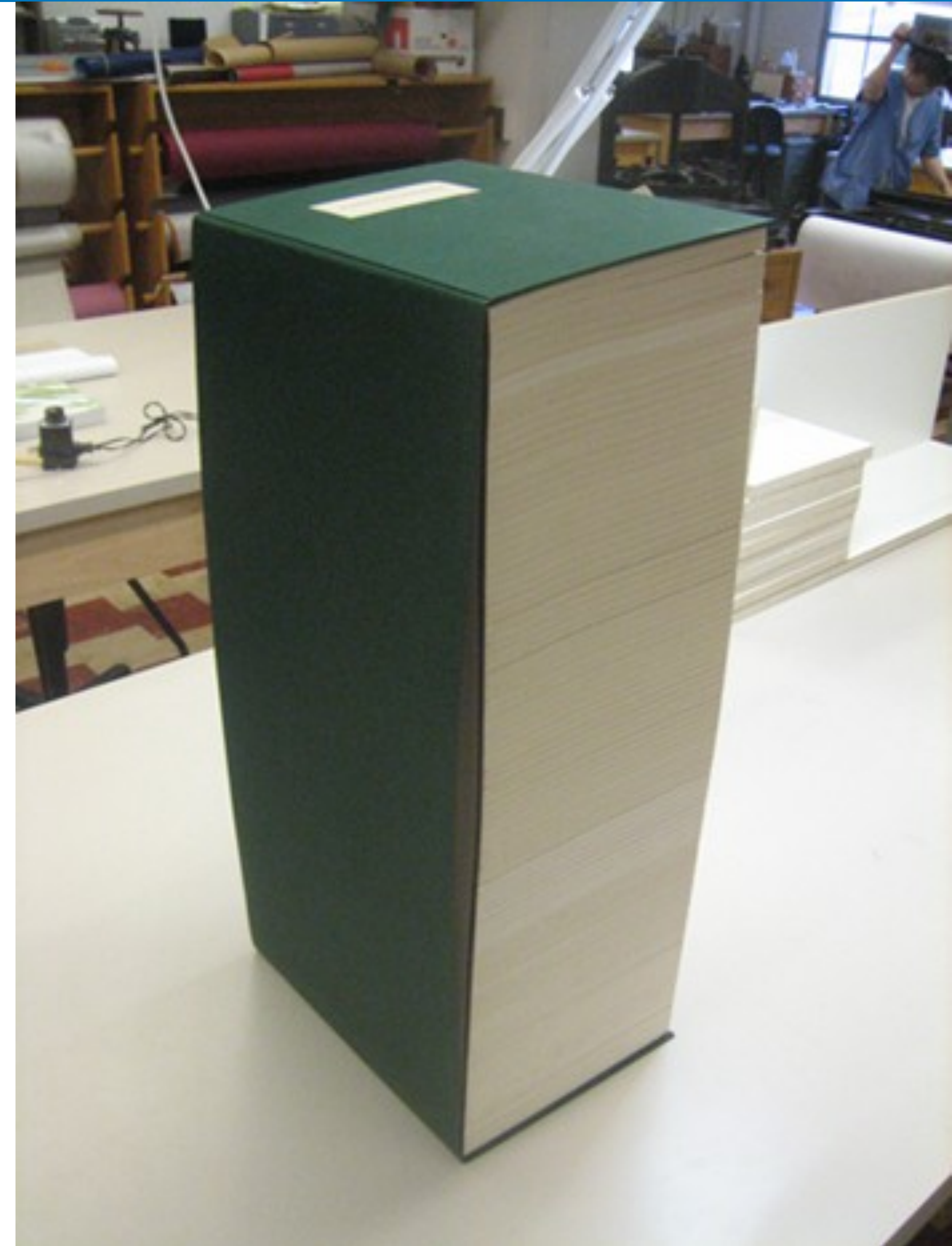
clang.llvm.org/extra/clang-tidy/checks/modernize-pass-by-value

C++ Move Semantics

only 2600 pages 😄💧



Borrow checker
baked the [lifetime rules](#) into the compiler



Myth #5

Adding **const** always helps

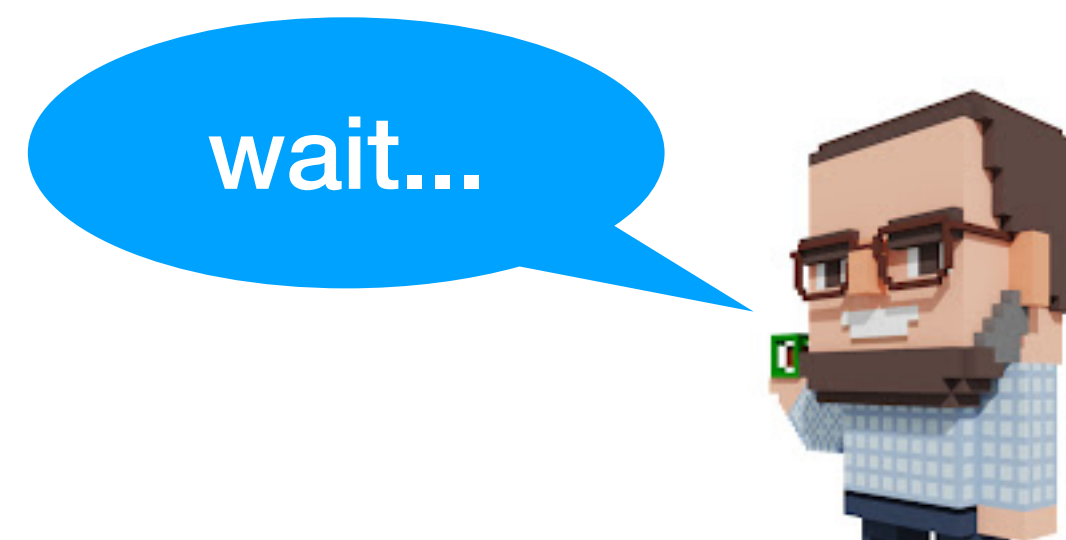
const all the things!



Myth #5

Adding **const** always helps

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



The image is a screenshot of a YouTube video player. The video title is "C++ Weekly With Jason Turner Episode 322" and the subtitle is "Top 4 Places To Never Use `const`". The video content shows a C++ code snippet in a compiler explorer, followed by the compiler output which shows 33948 warnings generated. The video player interface includes a play button, a progress bar at 0:00 / 18:52, and various control icons. The video player also shows a "Thank You To My Patrons!" message and a contact link for Klaim Music.

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

Myth #5

compiler-explorer.com/z/9Wcc54r9x

Myth #5

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

compiler-explorer.com/z/9Wcc54r9x

Myth #5

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

compiler-explorer.com/z/9Wcc54r9x

Myth #5

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

- don't **const** non-reference return types

compiler-explorer.com/z/9Wcc54r9x

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)

compiler-explorer.com/z/9Wcc54r9x

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

compiler-explorer.com/z/9Wcc54r9x

Myth #5

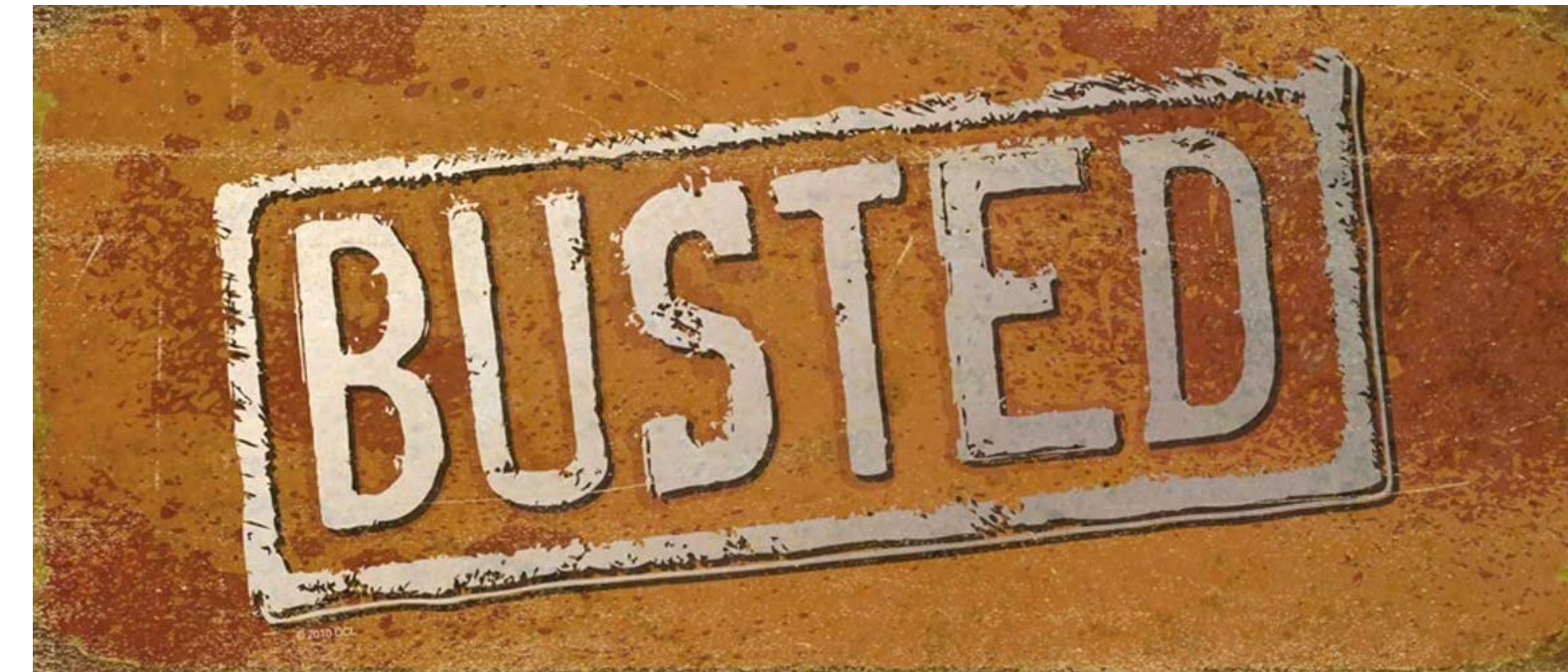
Top 4 places to **never use const**:

- 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 ?

Make All Data Members Private ?

- typically seen as **good practice**

Make All Data Members Private ?

- typically seen as **good practice**
- enforces **encapsulation**: the object is in control of its internal states

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)

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**

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*")

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

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 { }


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` 😊



C++
2022

youtube.com/watch?v=Y3wxJD3Bpql

ABSTRACTION PATTERNS:

*MAKING CODE RELIABLY BETTER
WITHOUT DEEP UNDERSTANDING*

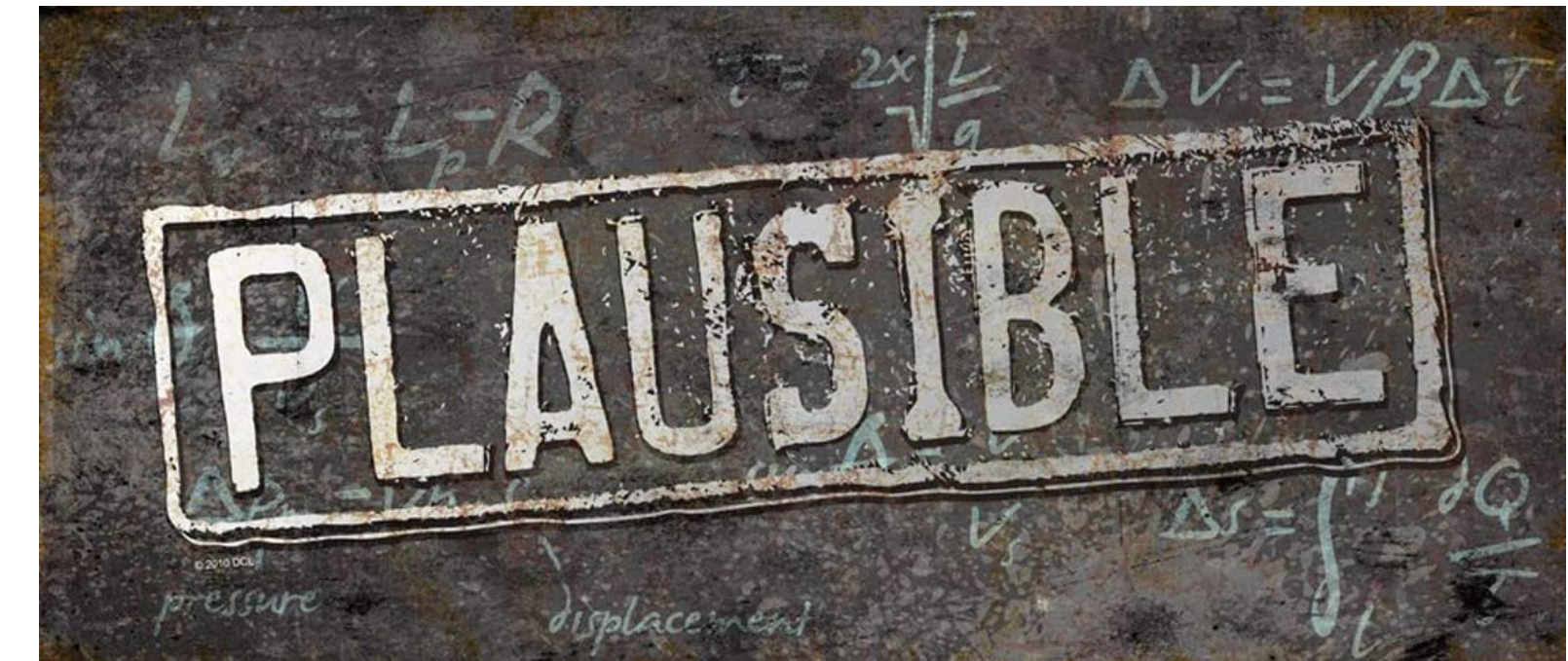
KATE GREGORY

00:02:39 / 01:24:13

Speed CC

Myth #37

Make All Data Members Private ?



Sometimes `structs` just wanna be `structs` 😊

Sometimes `structs` just wanna be `structs` 😊

While we're on the subject of `structs` ...

Recommended

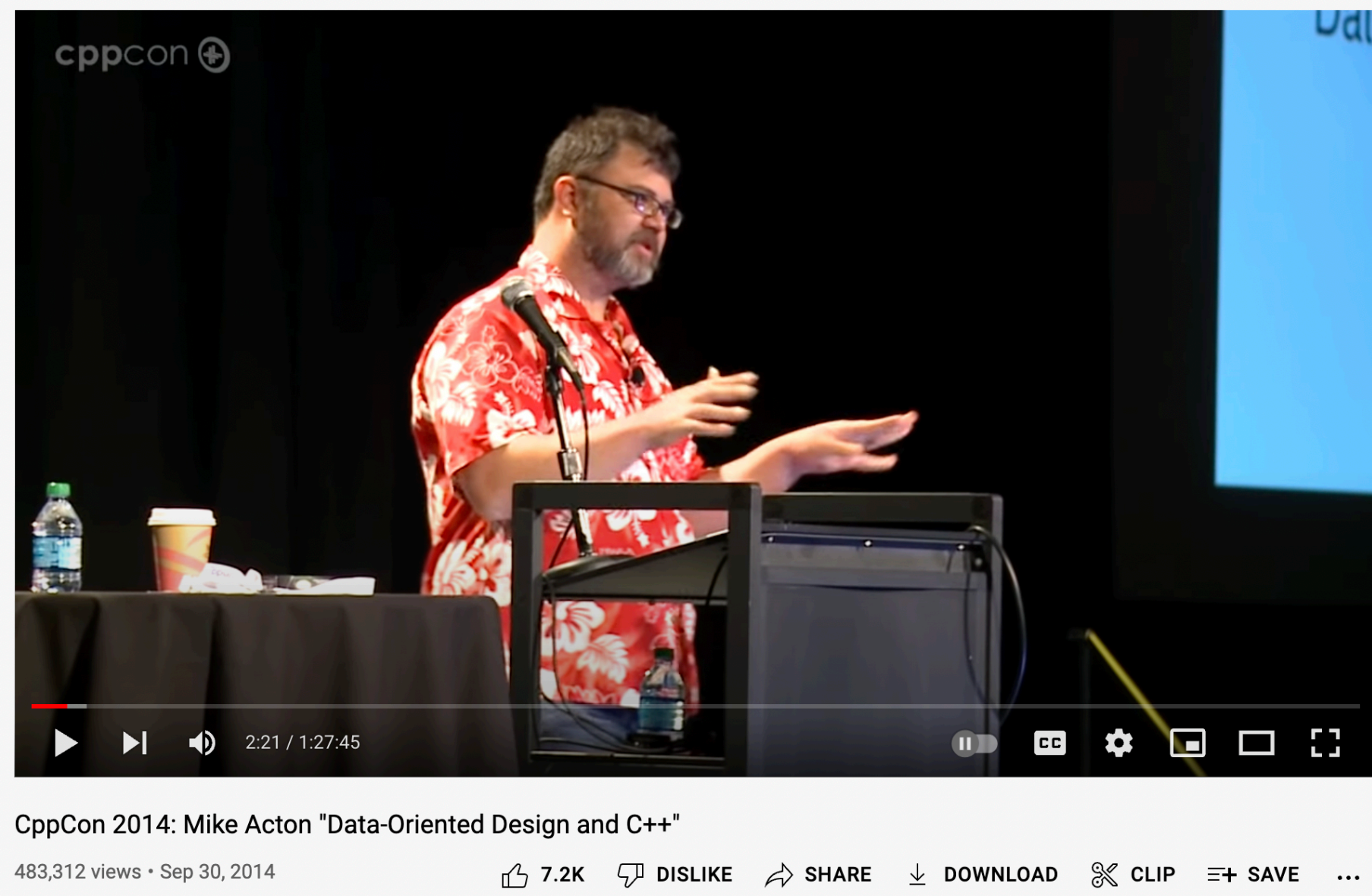


CppCon 2014: Mike Acton "Data-Oriented Design and C++"

483,312 views • Sep 30, 2014

👍 7.2K 👎 DISLIKE ➦ SHARE ⬇️ DOWNLOAD ✂️ CLIP ≡+ SAVE ...

Recommended



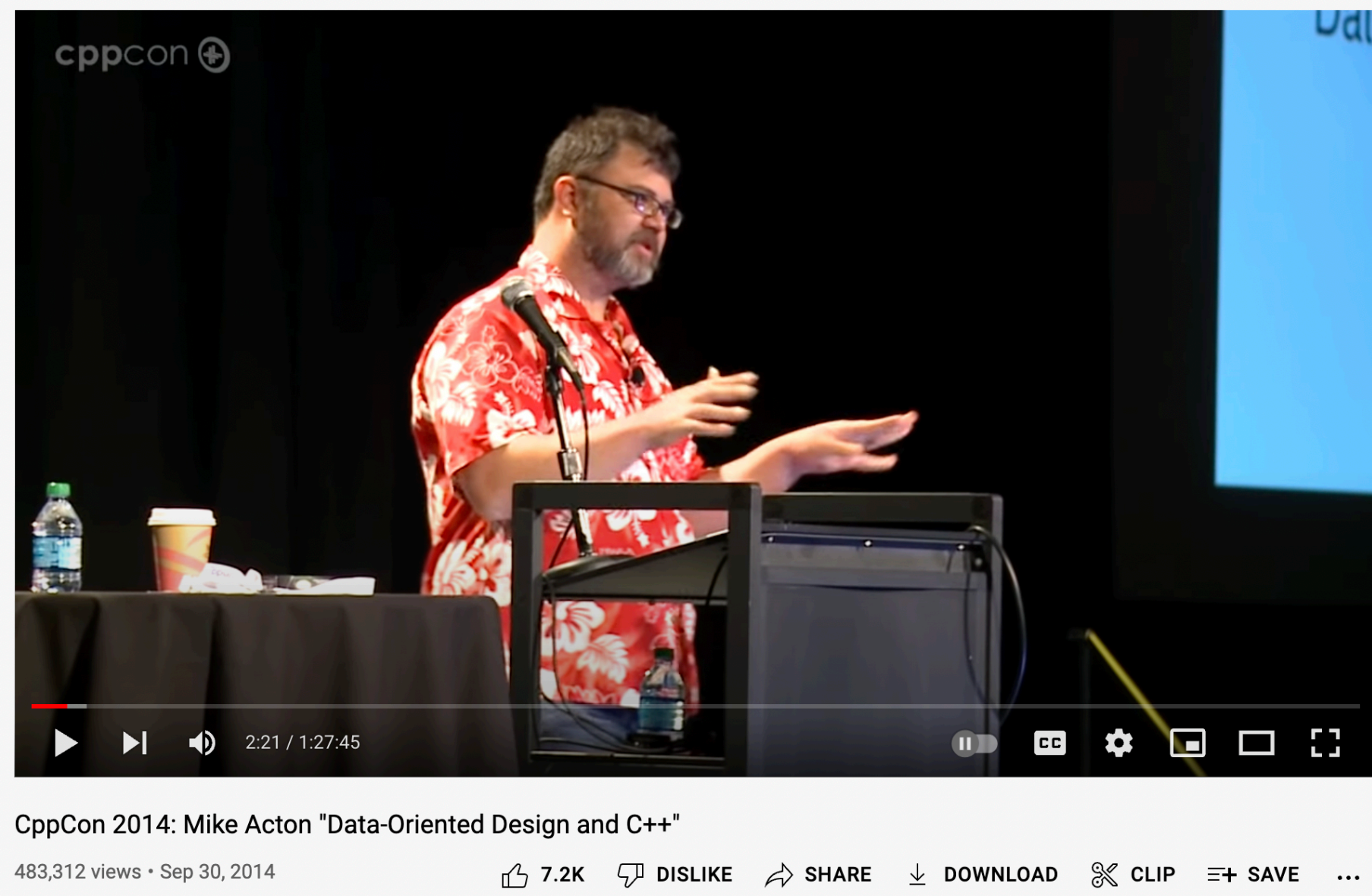
Unpopular opinion (in the C++ community):

But I still think [Mike Acton](#)'s keynote at CppCon 2014 is one of the best CppCon keynotes ever.

I still remember the audience gasps 🤪

youtube.com/watch?v=rX0ltVEVjHc

Recommended



Unpopular opinion (in the C++ community):

But I still think [Mike Acton](#)'s keynote at CppCon 2014 is one of the best CppCon keynotes ever.

I still remember the audience gasps 🤪

youtube.com/watch?v=rX0ItVEVjHc

Software = **Data** ➡ xForm ➡ **Data**

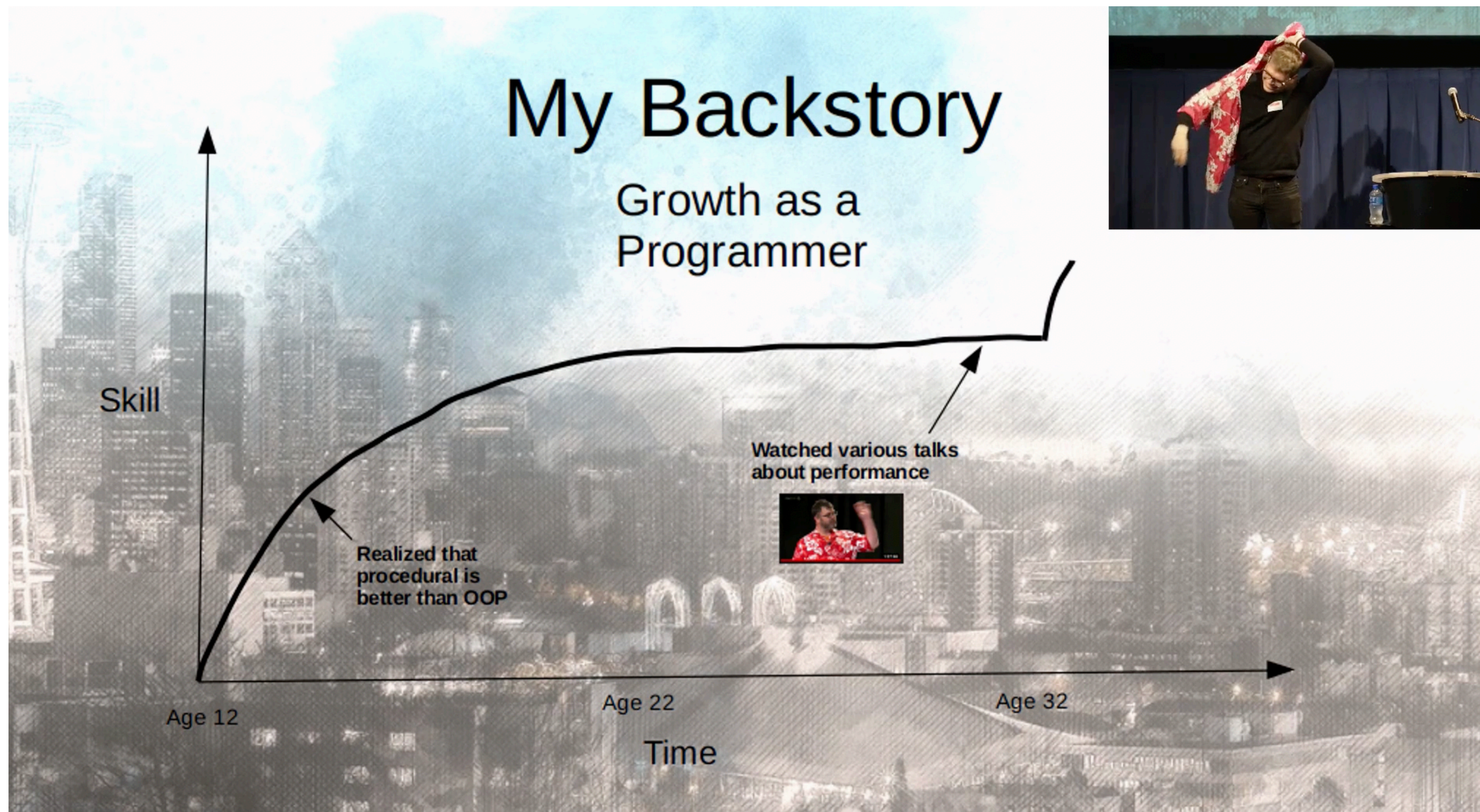
Performance = smart data layout

Memory access patterns

Recommended

A Practical Guide to Applying **Data-Oriented Design**

vimeo.com/649009599



Andrew Kelley



Myth #40

Iterators must go!

BoostCon 2009



"What's up with that?"

"Destroy!" 😊

I finally found a recording of this:

archive.org/details/AndreiAlexandrescuKeynoteBoostcon2009

Iterators must go!

BoostCon 2009



A. Stepanov: *"Alexandrescu is on a crusade to eliminate iterators [...]"*

Programming Conversations Lecture - A9

youtube.com/playlist?list=PLHxtyCq_WDLXFAEA-IYoRNQIezL_vaSX-

Myth #40

Iterators must go!

BoostCon 2009



Someone did properly "*Destroy*", but after ~12 years...

Iterators and Ranges

youtube.com/watch?v=95uT0RhMGwA



Iterators and Ranges: Comparing C++ to D, Rust, and Others

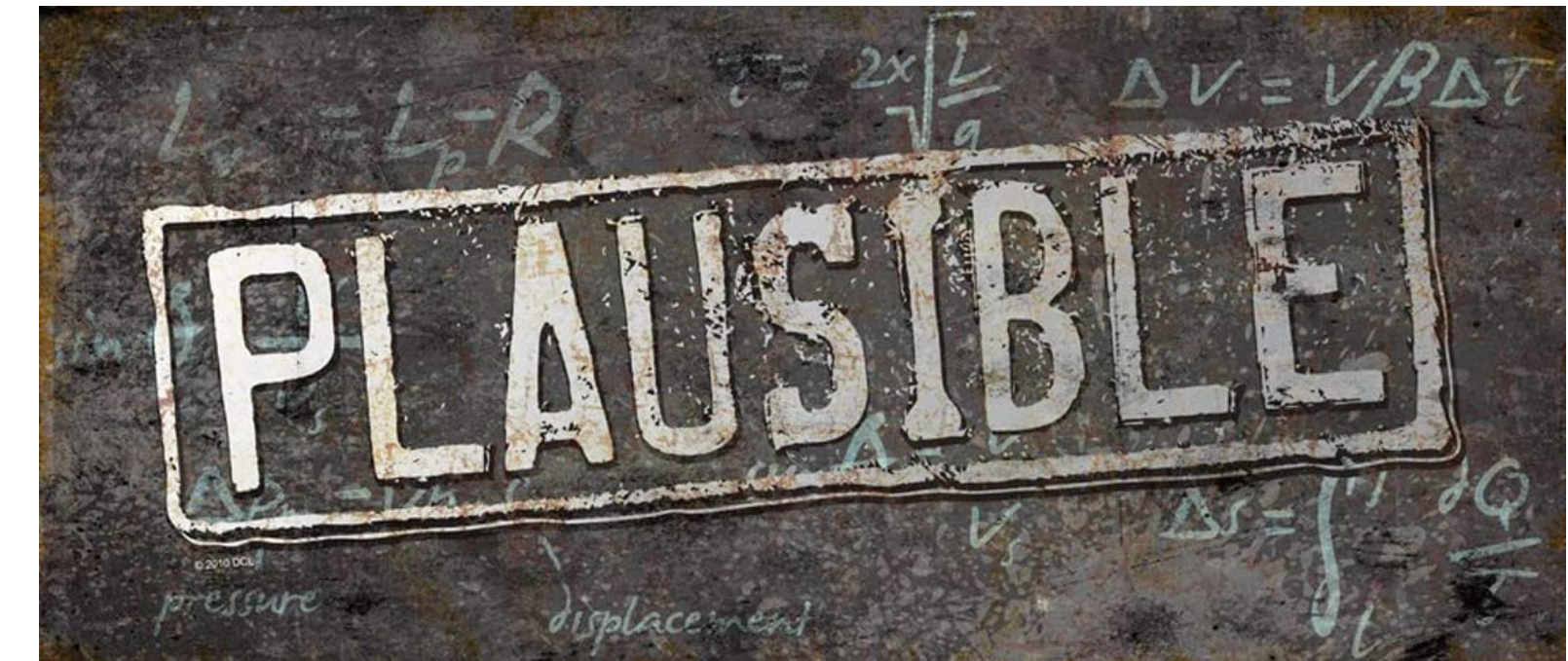
Barry Revzin



Keynote: Iterators and Ranges: Comparing C++ to D, Rust, and Others - Barry Revzin - CPPP 2021

Myth #40

Iterators must go!

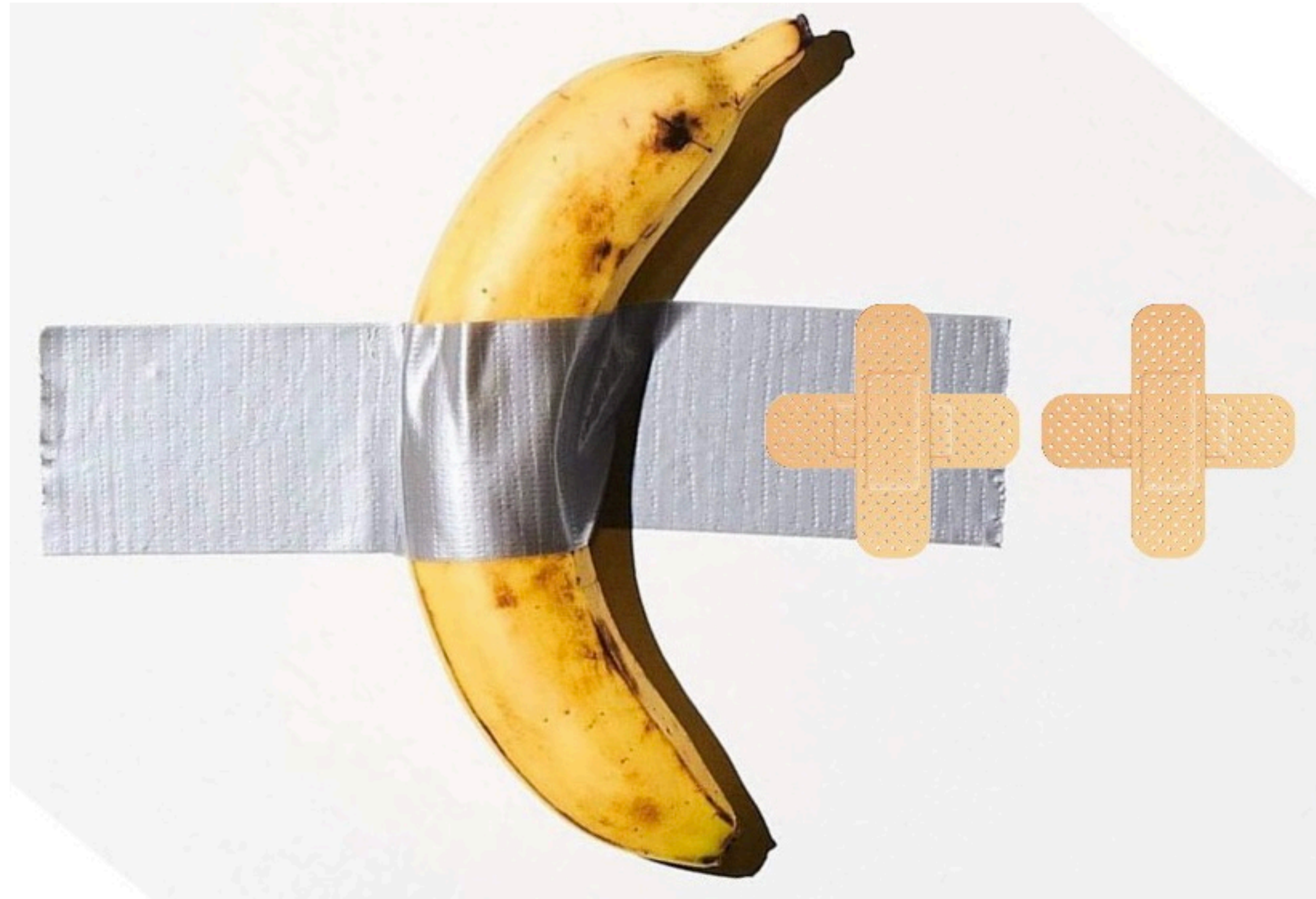


New (C++) is the enemy of the old

"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



<Part 1 of **N**> presentation

I invite other speakers to **join** this C++ Mythbusting series

The image shows a video player interface. The main content is a presentation slide with a white background and an orange header bar. The header bar contains the text "Some Programming Myths Revisited". Below the header, the presenter's name "Patrice Roy" is listed, along with two email addresses: "Patrice.Roy@USherbrooke.ca" and "Patrice.Roy@clg.qc.ca", and two affiliations: "CeFTI, Université de Sherbrooke" and "Collège Lionel-Groulx". To the right of the slide is a video thumbnail showing Patrice Roy at a podium. The thumbnail includes the Cppcon 2019 logo at the top, the text "Patrice Roy" on a black box, and the title "Some Programming Myths Revisited". Below the thumbnail, it says "Video Sponsorship Provided By:" followed by the "ansatz" logo. At the bottom of the video player, there are playback controls (play, next, volume, progress bar showing 0:25 / 1:00:50, and a pause button) and icons for Creative Commons, settings, HD, and full screen.

📍 AURORA

Some Programming Myths Revisited - Patrice Roy - CppCon 2019

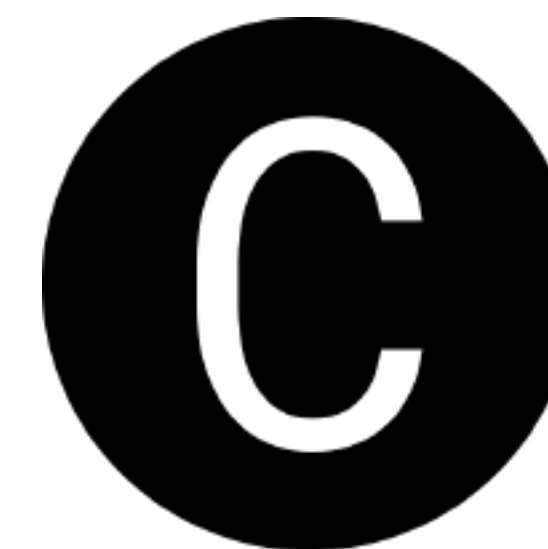
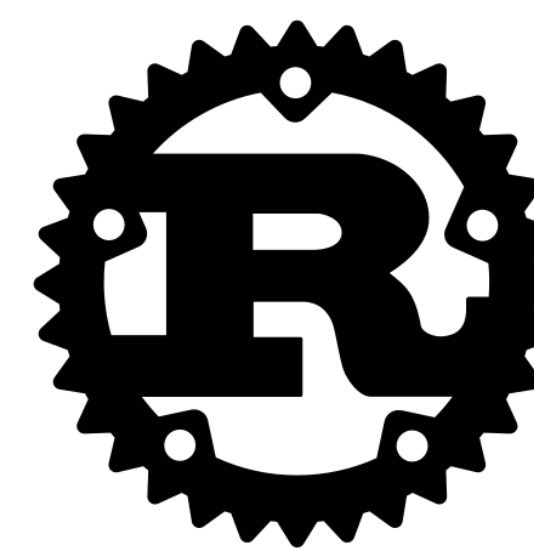
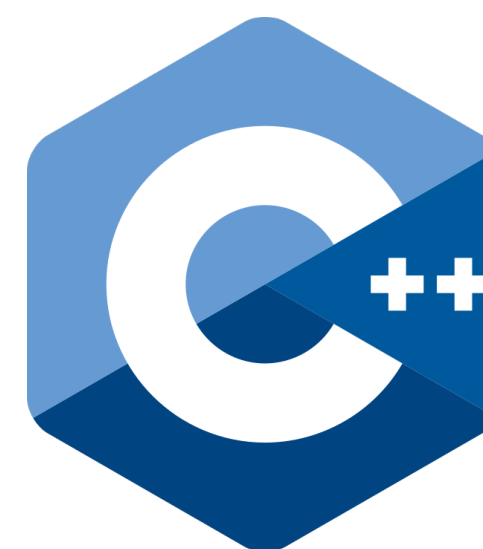
youtube.com/watch?v=KNqRjzSIUVo

In future episodes:

- strength reduction is so '90s, compilers are much smarter today
- `shared_ptr` is cheap (to copy)
- C++ templates produce code bloat
- C++ uniform initialization `{}` is to be preferred, always
- always capture `[=]` by value to avoid dangling
- `std::string` generates a lot of waste because of SSO (COW could be better)
- (N)RVO is always guaranteed
- concept checking increases build times
- function default argument values are evil
- ...

In future episodes:

- Featuring guest appearances by **Rust**
- Borrowing Trouble: the difficulties of a C++ borrow checker
- Rust and C++ interoperability
- Memory safety in Chromium
- Lifetime annotations for C++ (LLVM RFC)
- ...





code::dive 

November 2022



@ciura_victor

Victor Ciura
Senior SW Engineer
Visual C++

