C++ Club Meeting Notes

Gleb Dolgich

2018-08-02

Follow-up: Four Habit-Forming Tips to Faster C++ (1/4)

Post by KDAB

“Cache member-variables and reference-parameters”

The original version in the article doesn’t even work without aliasing:

```
template<typename T>
class complex {
    complex& operator*=(const complex<T>& a) {
        real = real * a.real - imag * a.imag;
        // BUG: real is changed, then used in subsequent calculation
        imag = real * a.imag + imag * a.real;
        return *this;
    }
};
```

Follow-up: Four Habit-Forming Tips to Faster C++ (2/4)

Post by KDAB

“Cache member-variables and reference-parameters”

The recommended version from the article:

```
template<typename T>
class complex {
    complex& operator*=(const complex<T>& a) {
        T a_real = a.real, a_imag = a.imag;
        T t_real = real, t_imag = imag;
        real = t_real * a_real - t_imag * a_imag;
        imag = t_real * a_imag + t_imag * a_real;
        return *this;
    }
};
```

Follow-up: Four Habit-Forming Tips to Faster C++ (3/4)

Post by KDAB

“Cache member-variables and reference-parameters”

Proposed “economy” version:

```
template<typename T>
class complex {
    complex& operator*=(const complex<T>& a) {
        const auto real_this{real}; // modified by our code
        const auto real_a{a.real}; // may be modified if aliased
        real = real * a.real - imag * a.imag;
        imag = real_this * a.imag + imag * real_a;
        return *this;
    }
};
```

Follow-up: Four Habit-Forming Tips to Faster C++ (4/4)

Post by KDAB

“Cache member-variables and reference-parameters”

Proposed “easy” version:

```
template<typename T>
class complex {
    complex& operator*=(complex<T> a) {
        const auto self{*this};
        real = self.real * a.real - self.imag * a.imag;
        imag = self.real * a.imag + self.imag * a.real;
        return *this;
    }
};
```

C++ Quiz by Shafik Yaghmour via Twitter (1/2)

Tweet

C:

```c
malloc(0)
```

C++:

```cpp
new int[0]
```

What's the behavior of these expressions?

1. Both return NULL/nullptr
2. Both return an address that's invalid to dereference
3. Behavior can vary
4. Undefined behavior

C++ Quiz by Shafik Yaghmour via Twitter (2/2)

Tweet

In C99 and C11 malloc(0) is allowed to return either NULL or pointer that is not valid to dereference:

- C99 N1256 7.20.3
- C11 N1570 7.22.3

In C++ we get back an array with zero elements:

- expr.new
- dynamic.allocation
- dynamic.allocation

Allocaling 0 bytes

How much memory does malloc(0) allocate?

On most systems, this little C program will soak up all available memory:

```
while (1) {
   malloc(0);
}
```

so the answer is not the obvious "zero." But before getting into `malloc(0)`, let's look at the simpler case of `malloc(1)`.

There's an interesting new C programmer question about `malloc`: "Given a pointer to dynamically allocated memory, how can I determine how many bytes it points to?" The answer, rather frustratingly, is "you can't." But when you call `free` on that same pointer, the memory allocator knows how big the block is, so it's stored *somewhere*. That somewhere is commonly adjacent to the allocated memory, along with any other implementation-specific data needed for the allocator.

Now we can approach the case of allocating zero bytes. It turns out there's a silly debate about the right thing to do, and it hasn't been resolved, so technically allocating zero bytes is implementation-specific behavior. One side thinks that `malloc(0)` should return a null pointer and be done with it. It works, if you don't mind a null return value serving double duty. It can either mean "out of memory" or "you didn't request any memory."

The more common scheme is that `malloc(0)` returns a unique pointer. You shouldn't dereference that pointer because it's conceptually pointing to zero bytes, but we know from our adventures above that at least `dlmalloc` is always going to allocate a 32 byte block on a 64-bit system, so that's the final answer: it takes 32 bytes to fulfill your request for no memory.

Metashell GUI

- Code (MIT)
  - Reddit
- Metashell “The goal of this project is to provide an interactive template metaprogramming shell.” (GPL-3.0)

Ericsson CodeCompass

CodeCompass is a software comprehension tool for large scale software written in C/C++ and Java

- Code (GPL-3.0)

Expect the Expected, by Andrei Alexandrescu

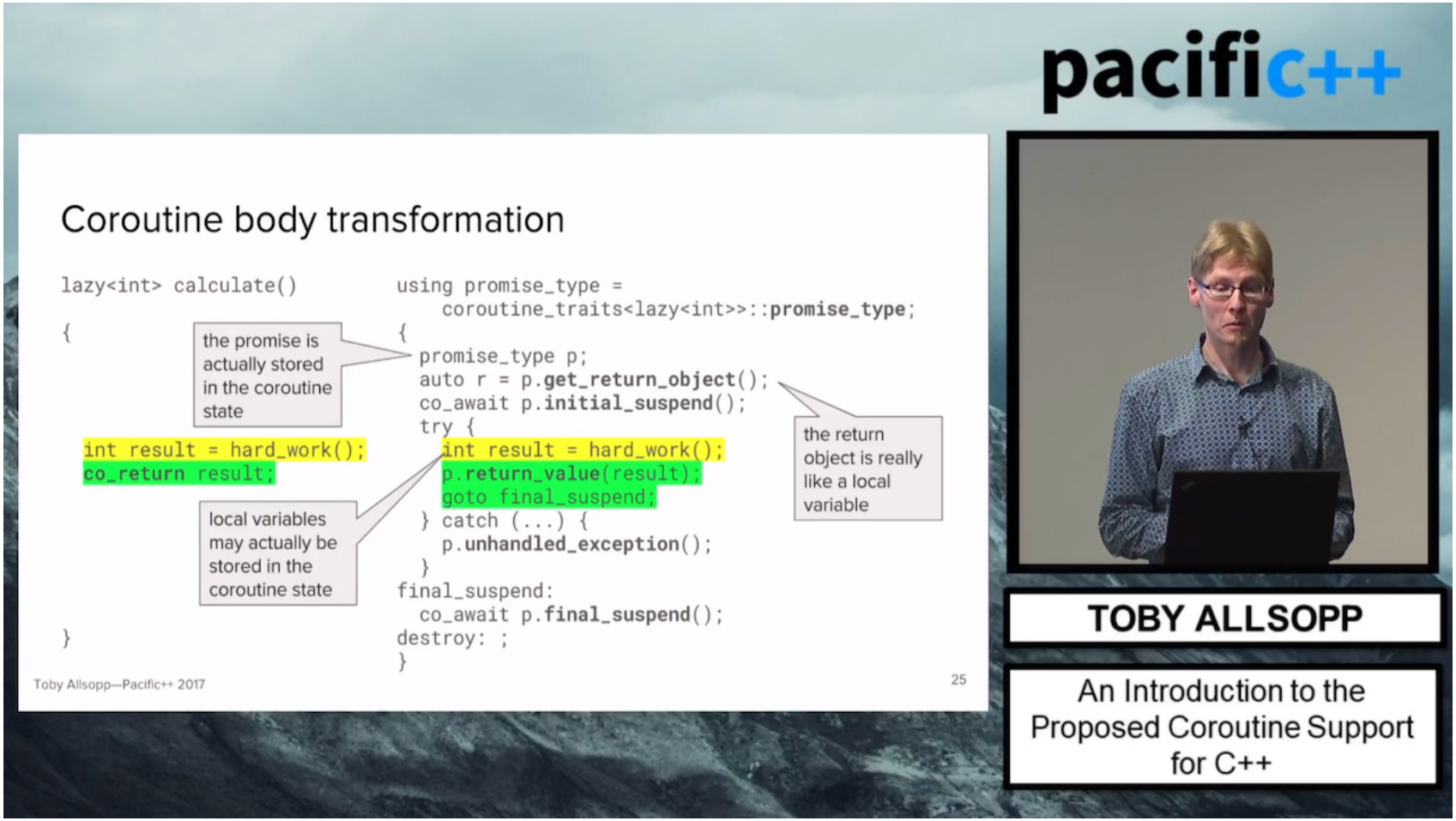
- Video
- Reddit
- Howard Hinnant on how to initialize a class using delegating constructors

CppCast: Parallel Ranges with Chris DiBella

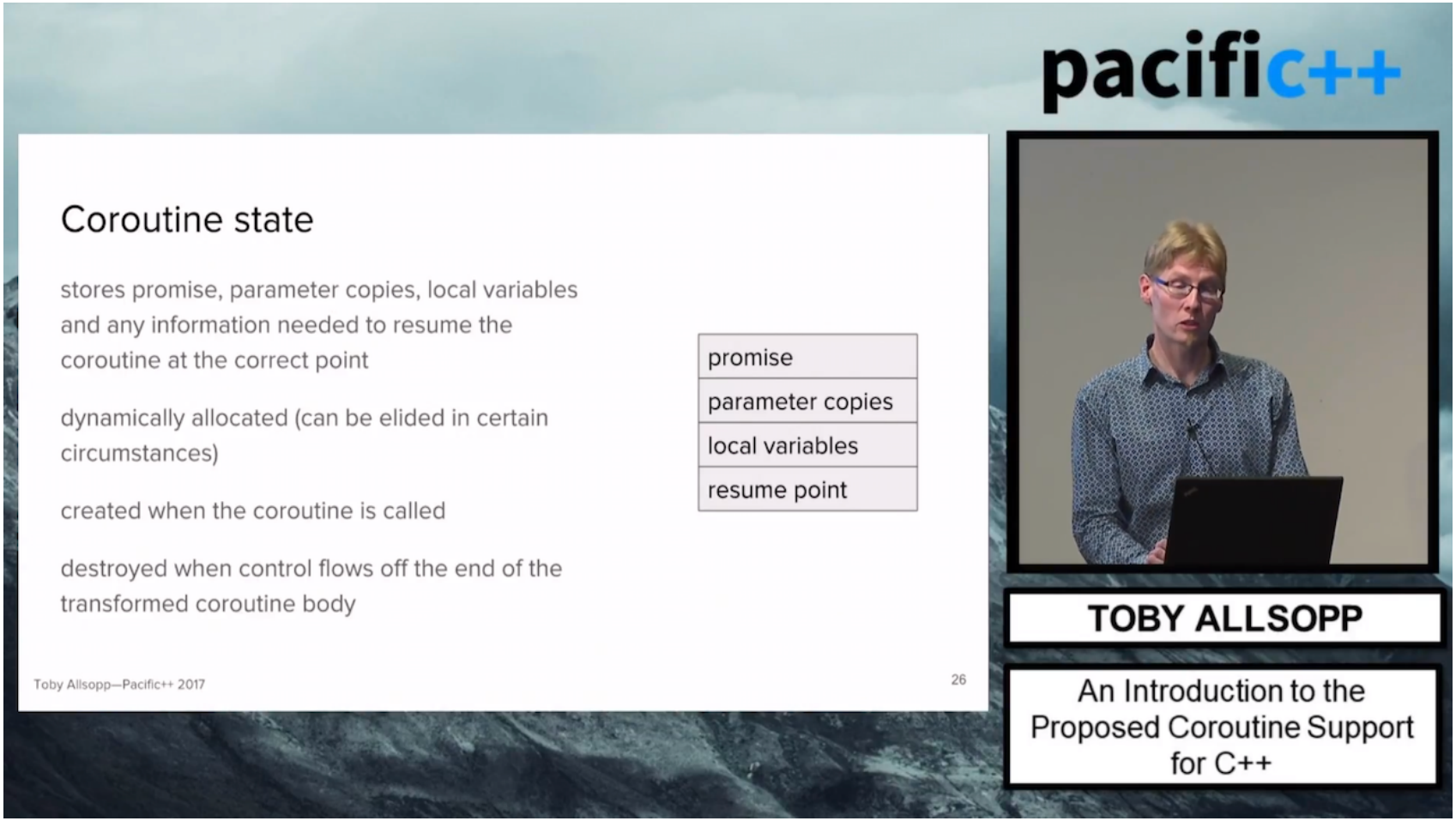
- Podcast
- P0836R1 Introduce Parallelism to the Ranges TS

Pacific++ 2017: Toby Allsopp “An Introduction to the Proposed Coroutine Support for C++” (1/2)

- Video



Pacific++ 2017: Toby Allsopp “An Introduction to the Proposed Coroutine Support for C++” (2/2)



Qt and the Coroutines TS

- Post by Jeff Trull
- Post by Jesús Fernández
- Boost.Asio coroutine support
- Coroutine Theory Series, by Lewis Baker
- cppcoro: Coroutine Utilities, by Lewis Baker
- How Coroutines Work, by Kirit Sælensminde

Exploiting Coroutines to Attack the “Killer Nanoseconds”

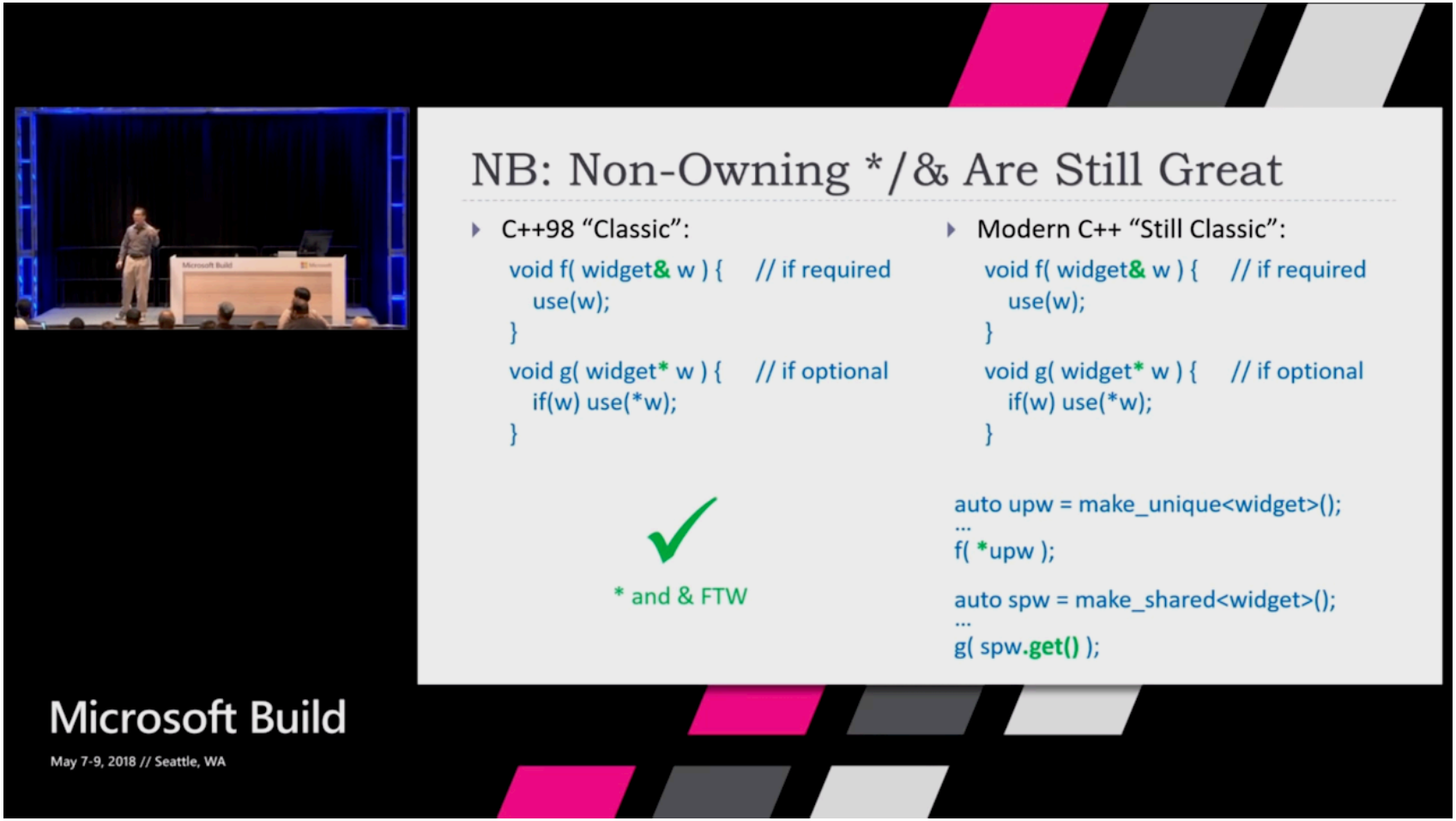
- PDF

In this work, we compare and contrast the state-of-the-art approaches to reduce CPU stalls due to cache misses for pointer-intensive data structures. We present an in-depth experimental evaluation and a detailed analysis using four popular data structures: hash table, binary search, Masstree, and Bw-tree. Our focus is on understanding the practicality of using coroutines to improve throughput of such data structures.

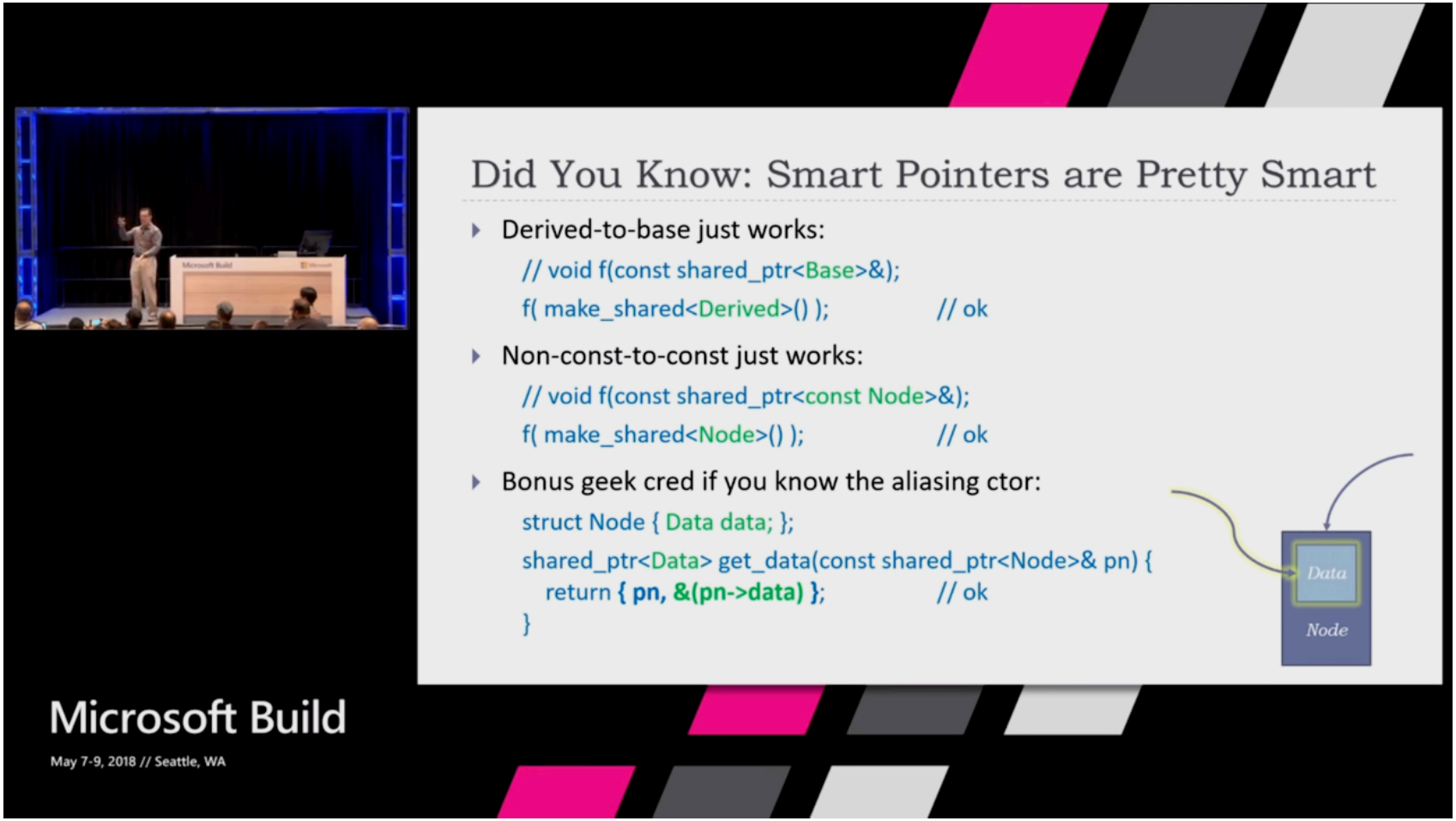
Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (1/7)

- Video
  - Reddit

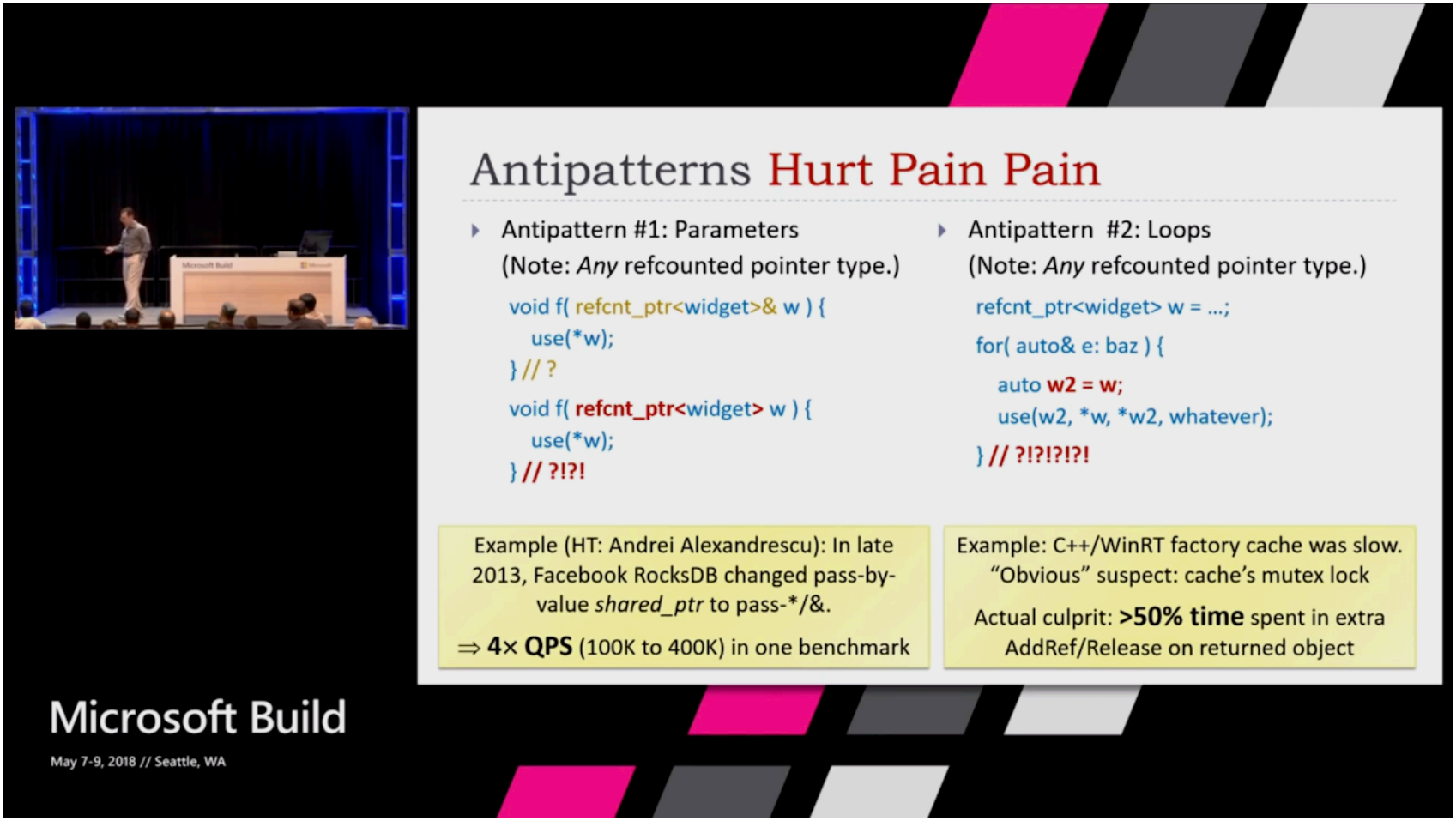
Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (2/7)



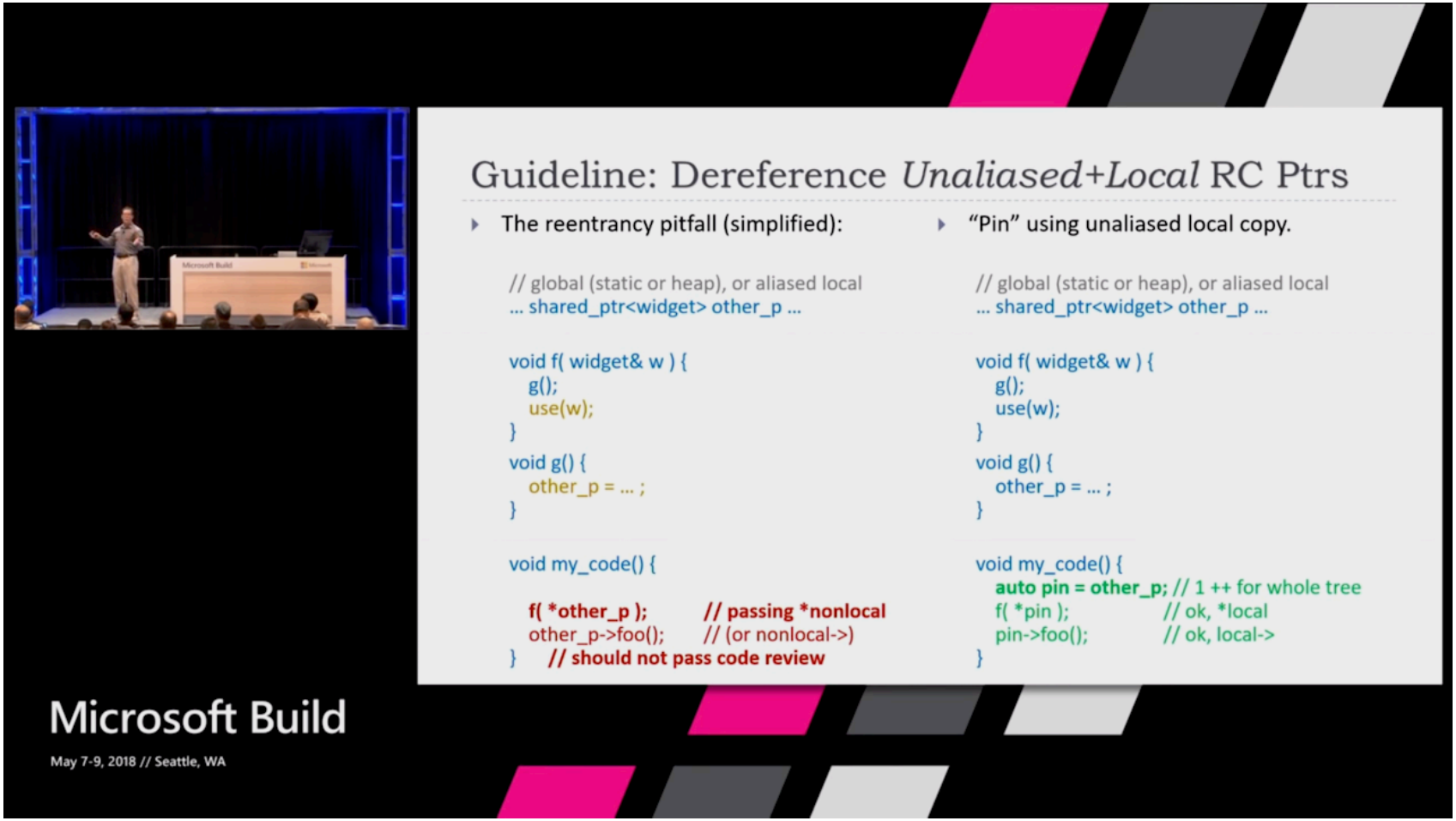
Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (3/7)



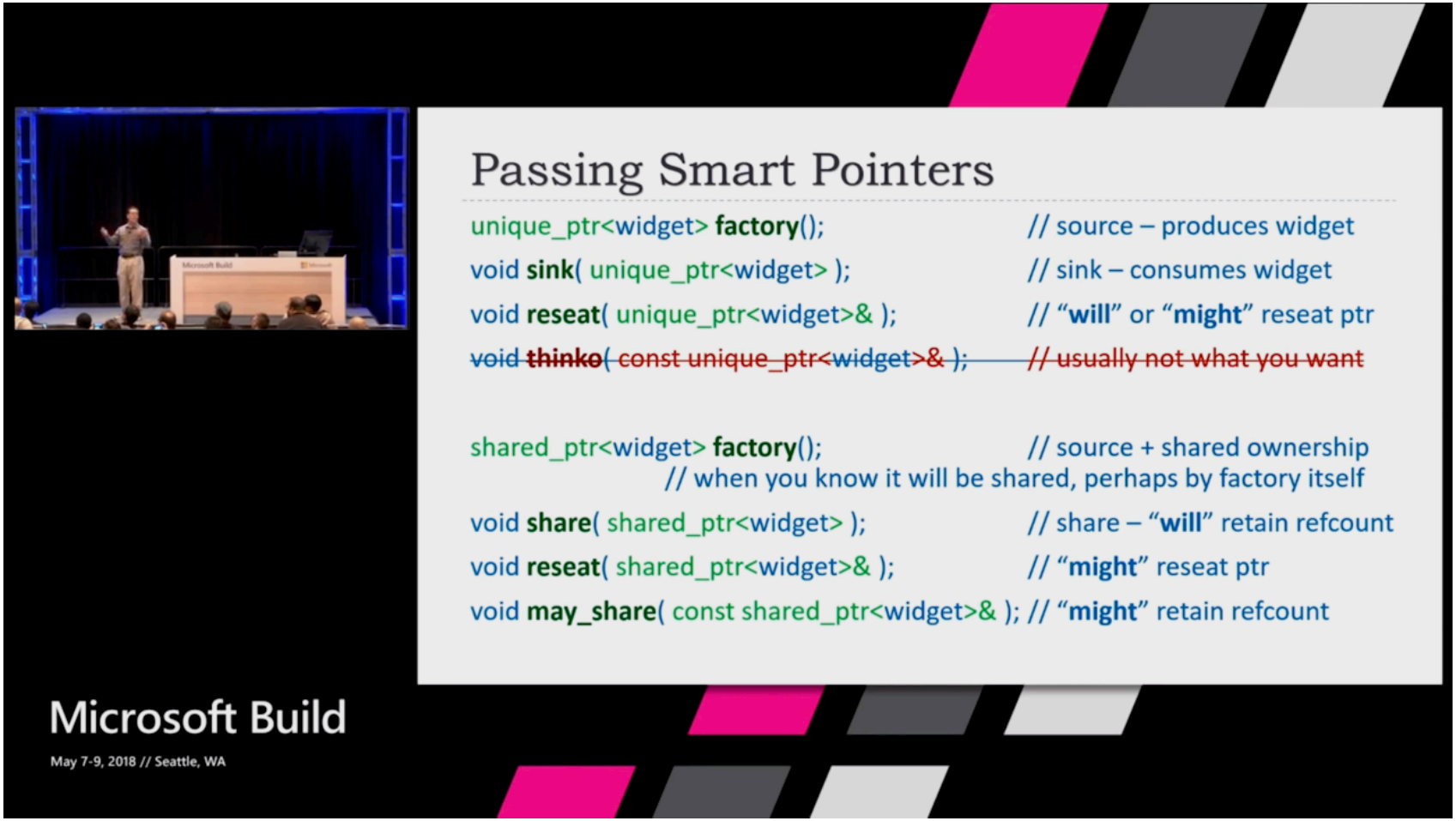
Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (4/7)



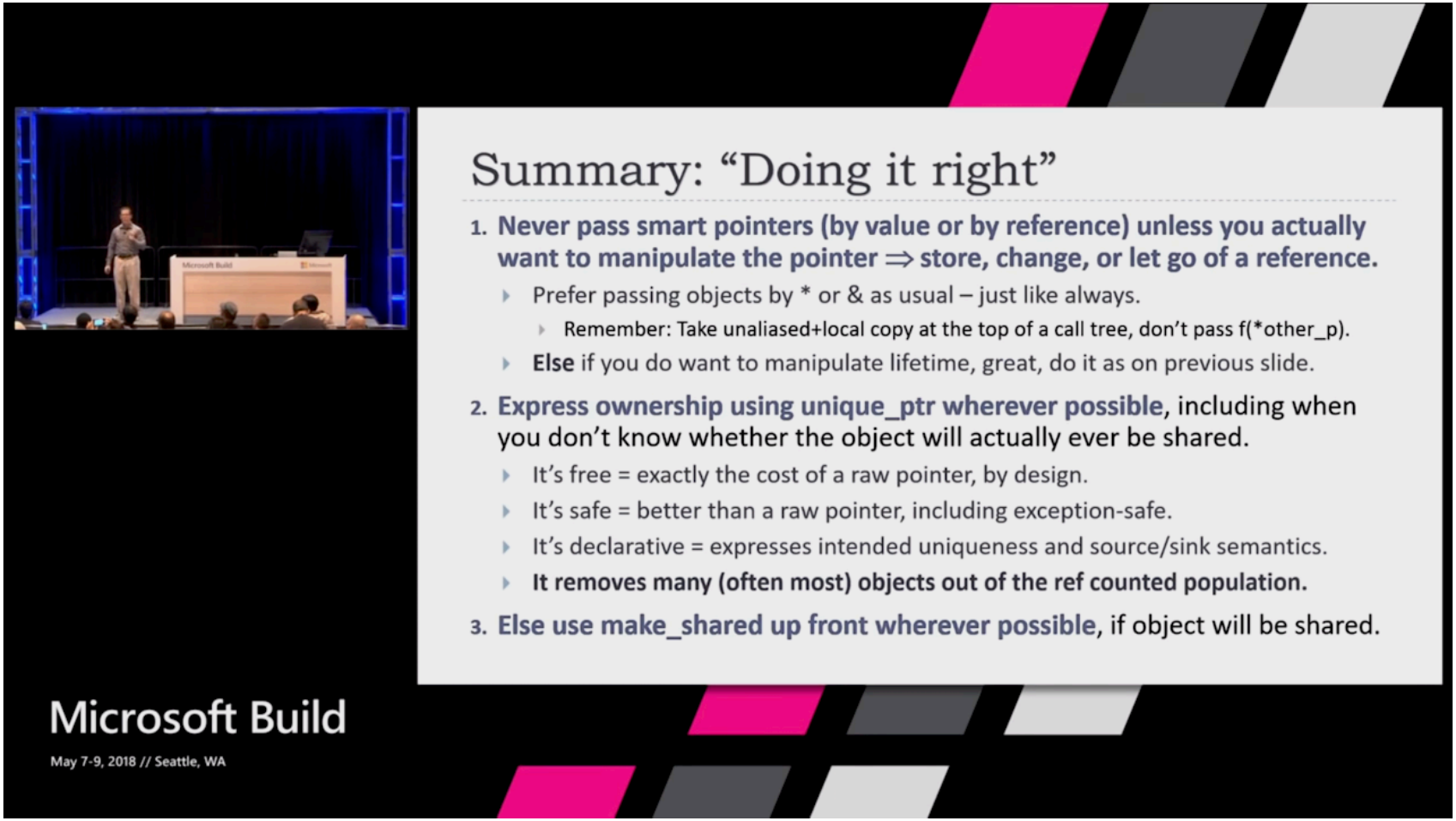
Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (5/7)



Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (6/7)



Herb Sutter: How to Adopt Modern C++17 into Your C++ Code (7/7)



Unity (“jumbo”) builds

AKA: amalgamated, or Single Compilation Unit (SCU) builds.

- A Guide to Unity Builds by Viktor Kirilov
- Support for Unity (Jumbo) Files in Visual Studio 2017 15.8 (Experimental)
- Chromium instructions and GoogleDoc
- Reddit
- Other tools:
  - CMake cotire and Unity Build macro
  - FASTBuild
  - Meson
  - waf
  - RudeBuild plugin for Visual Studio

`const auto*` versus `const auto` for Pointer Types

Article

Twitter


std::data::pup 🐶
@6b766e
3 Replies, 5 Quotes
whys is called ‘the rust community’ and not ‘the cargo cult’
27/07/2018, 22:58 (Friday)
Twitter for iPhone
Retweeted by @odinthenerd
29/07/2018, 11:52
1225 Likes
196 Retweets
Thread