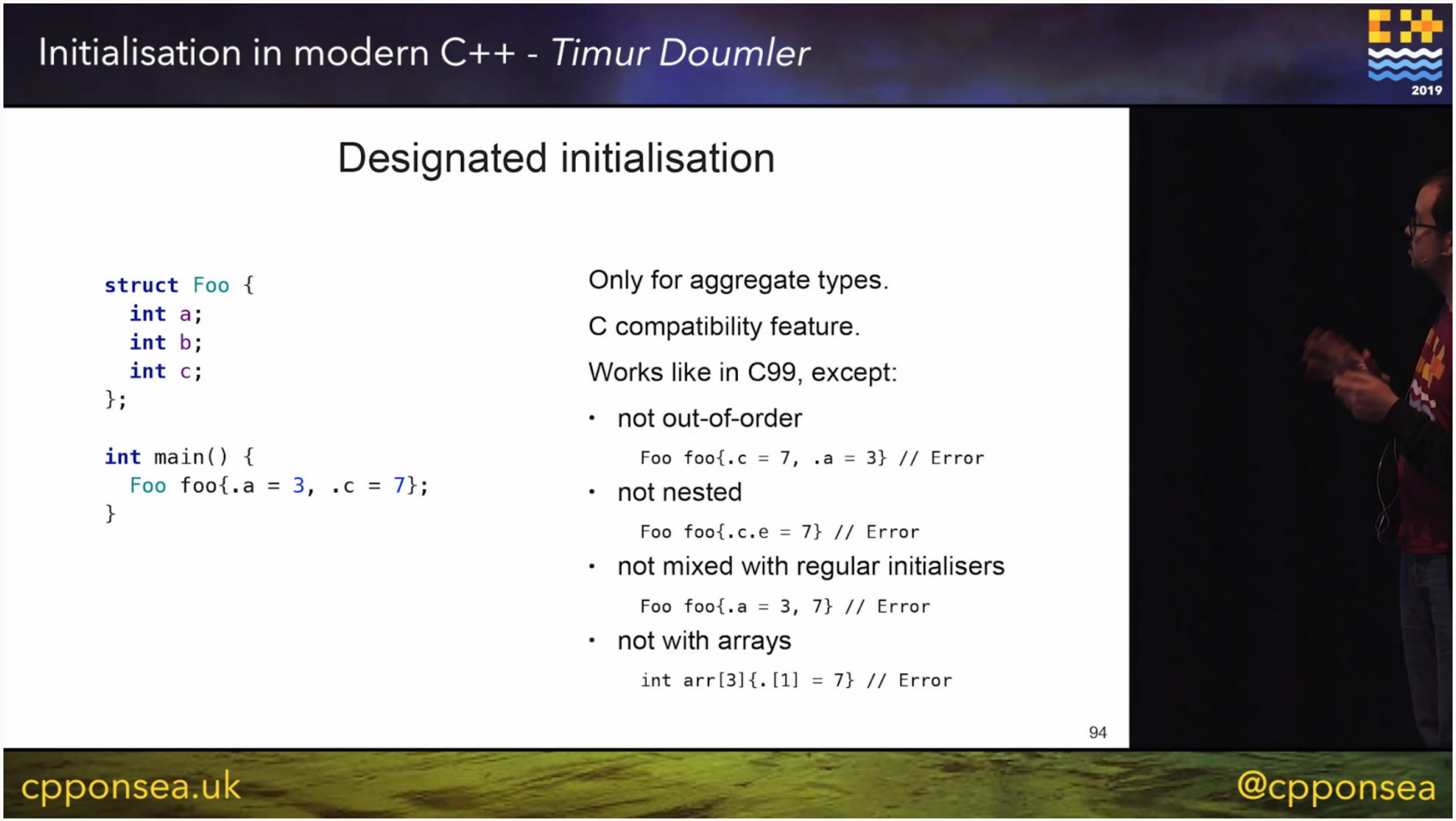
C++ Club Meeting Notes

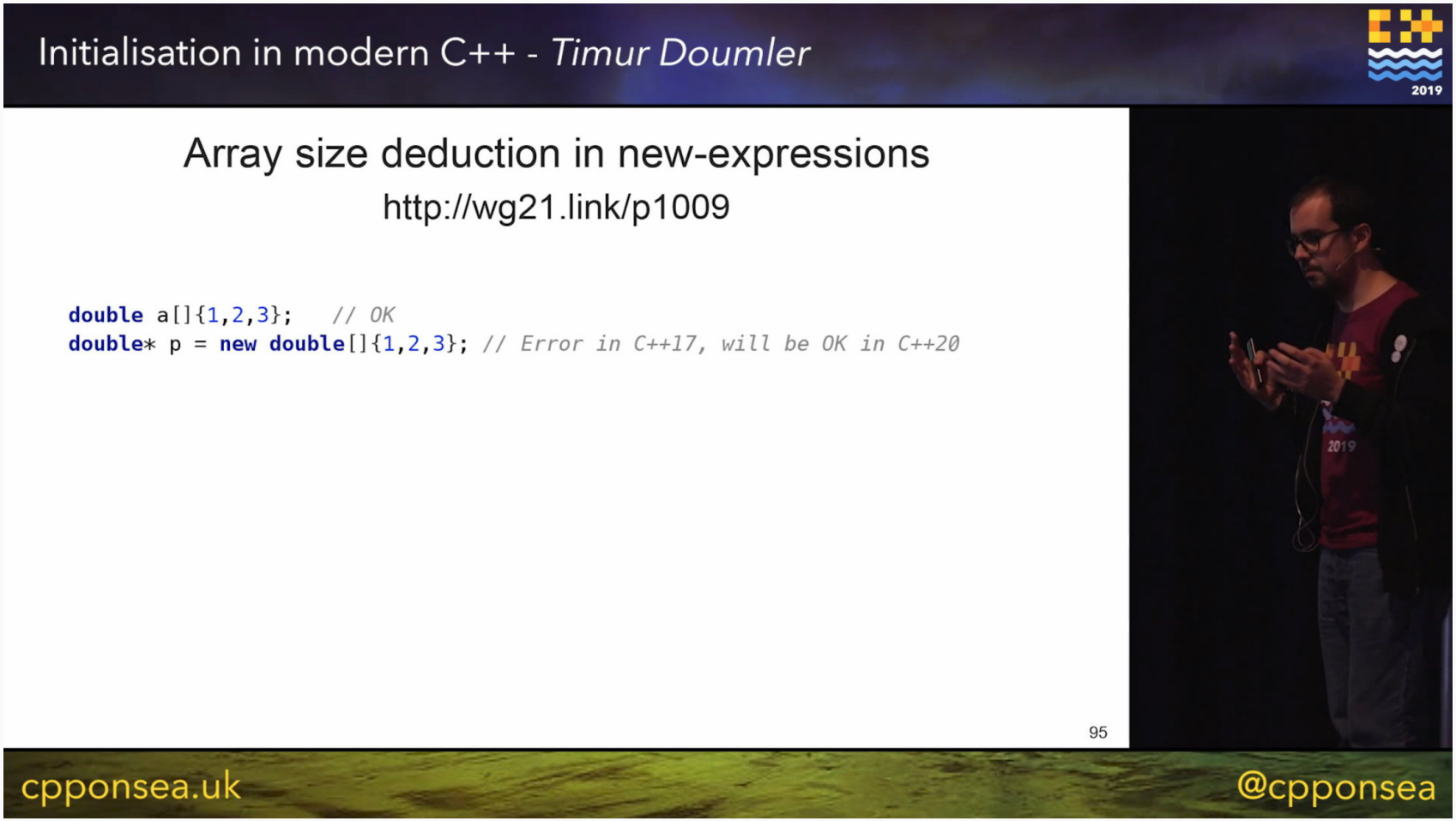
Gleb Dolgich

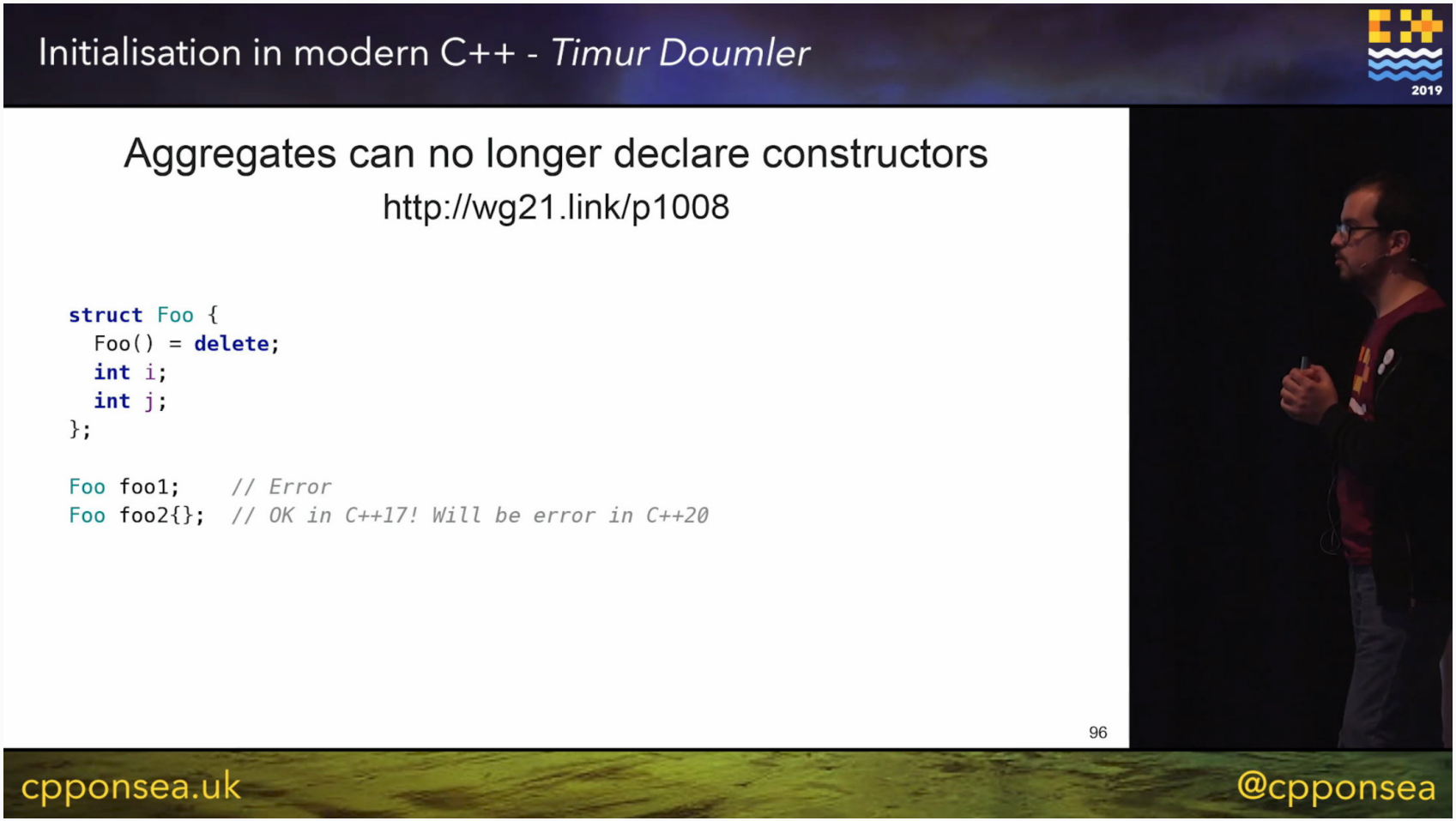
2020-01-30

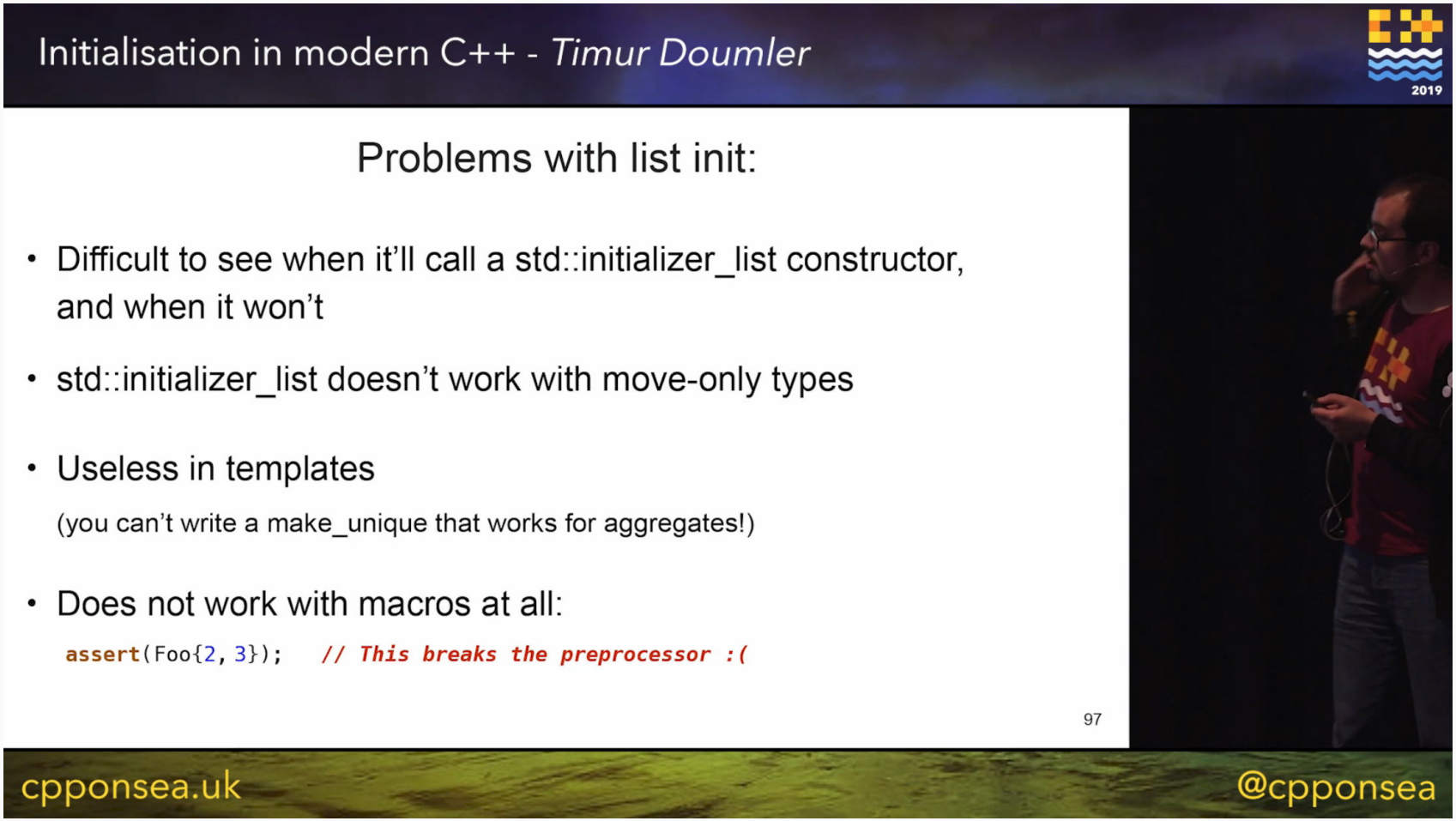
CppOnSea 2019: Timur Doumler - Initialisation in modern C++

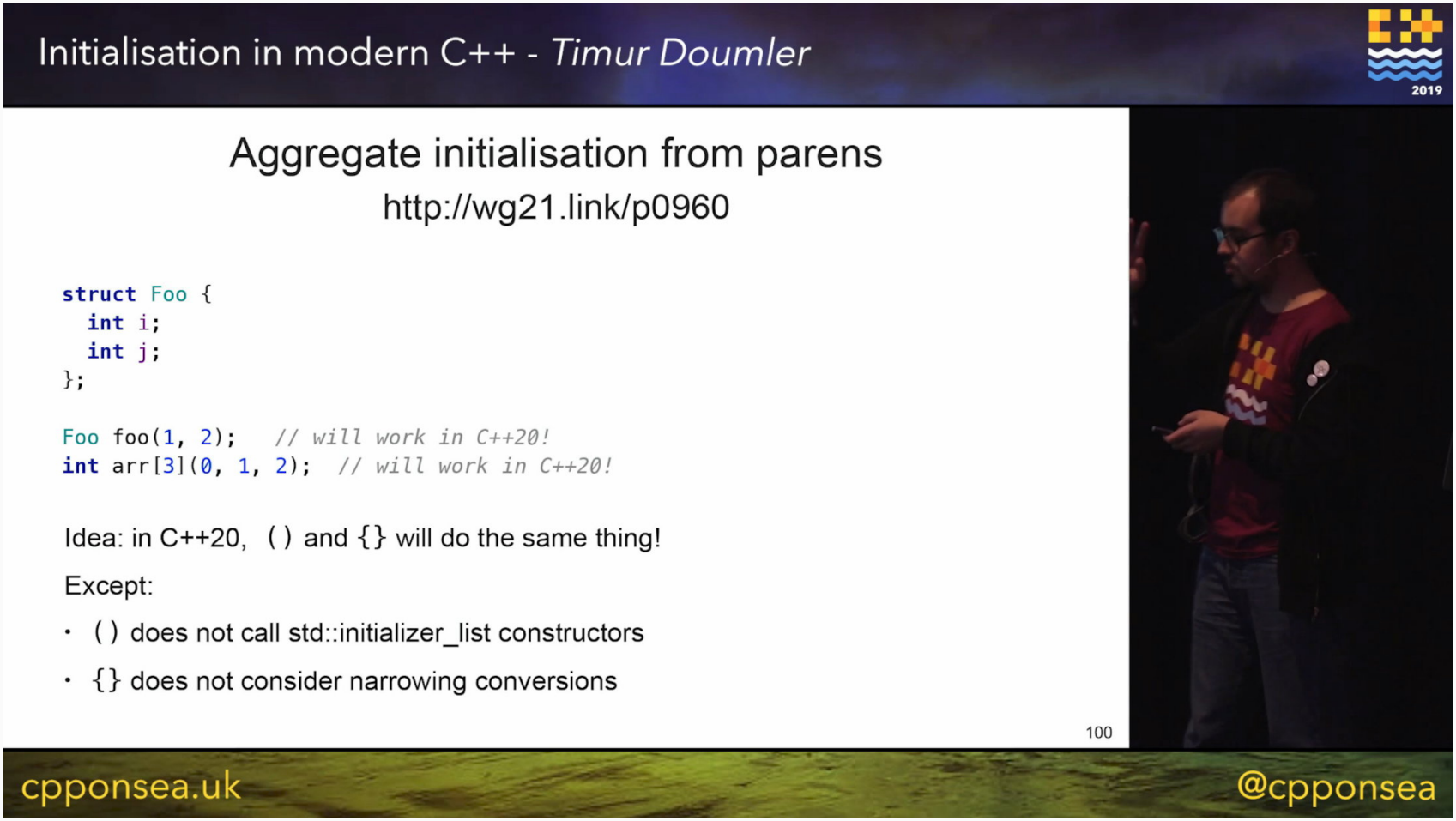
Video

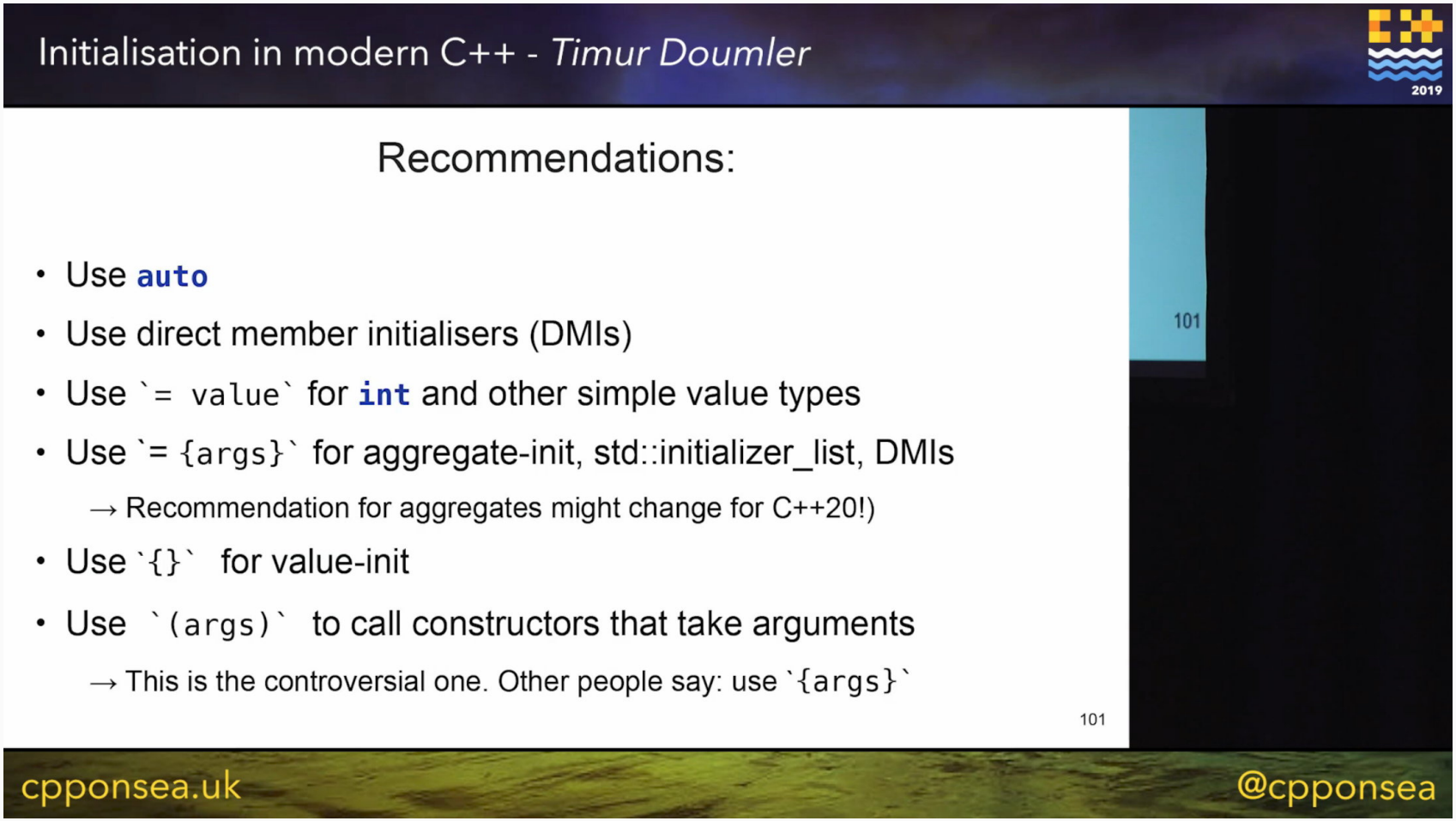

Initialisation in modern C++ - Timur Doumler
2019
Designated initialisation
struct Foo {
int a;
int b;
int c;
};
int main() {
Foo foo{.a = 3, .c = 7};
}
Only for aggregate types.
C compatibility feature.
Works like in C99, except:
not out-of-order
Foo foo{.c = 7, .a = 3} // Error
not nested
Foo foo{.c.e = 7} // Error
not mixed with regular initialisers
Foo foo{.a = 3, 7} // Error
not with arrays
int arr[3]{.[1] = 7} // Error
94
cpponsea.uk
@cpponsea


Initialisation in modern C++ - Timur Doumler
2019
Array size deduction in new-expressions
http://wg21.link/p1009
double a[]{1,2,3}; // OK
double* p = new double[]{1,2,3}; // Error in C++17, will be OK in C++20
95
cpponsea.uk
@cpponsea

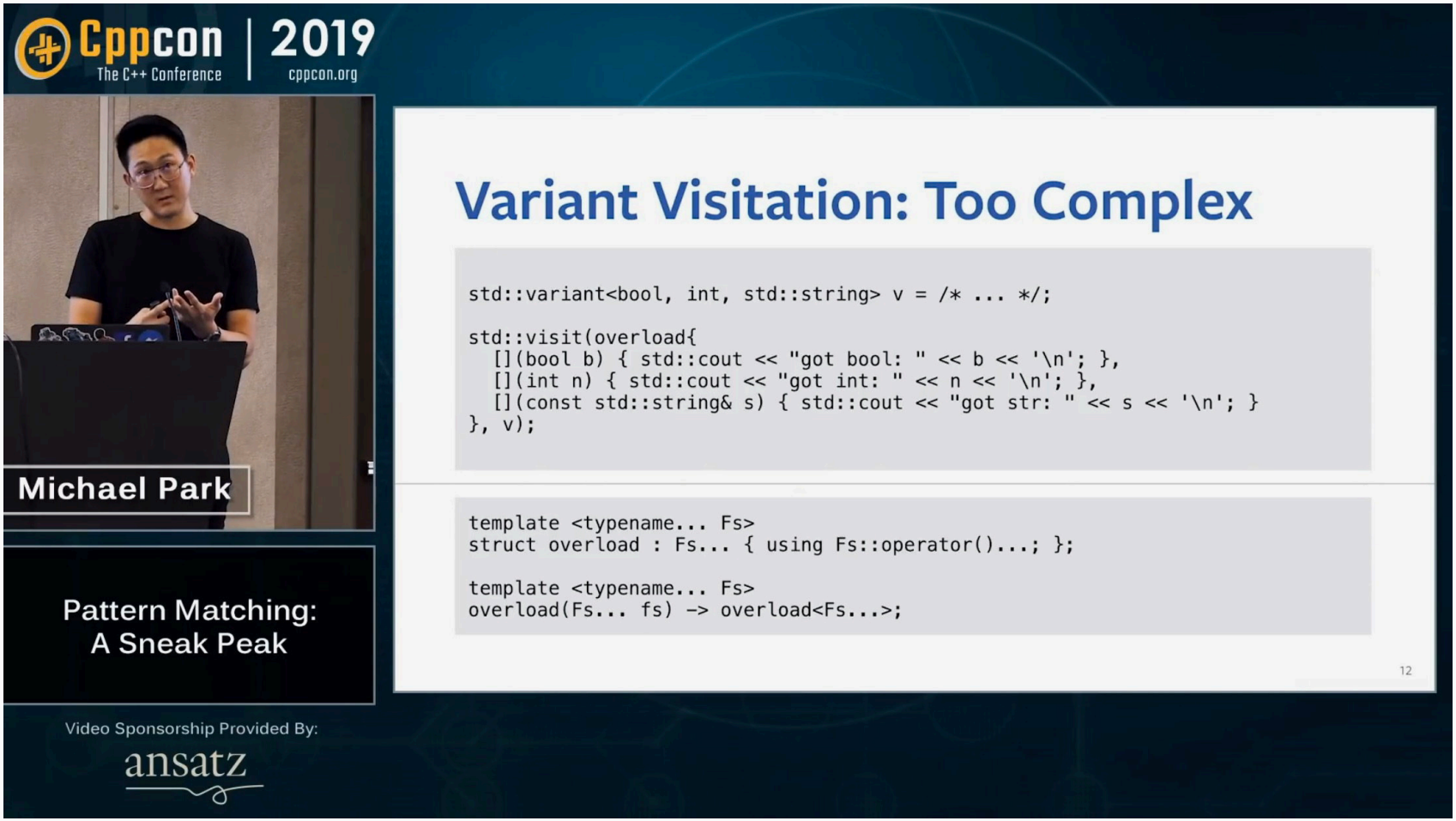

Initialisation in modern C++ - Timur Doumler
2019
Aggregates can no longer declare constructors
http://wg21.link/p1008
struct Foo {
Foo() = delete;
int i;
int j;
};
Foo foo1; // Error
Foo foo2{}; // OK in C++17! Will be error in C++20
96
cpponsea.uk
@cpponsea

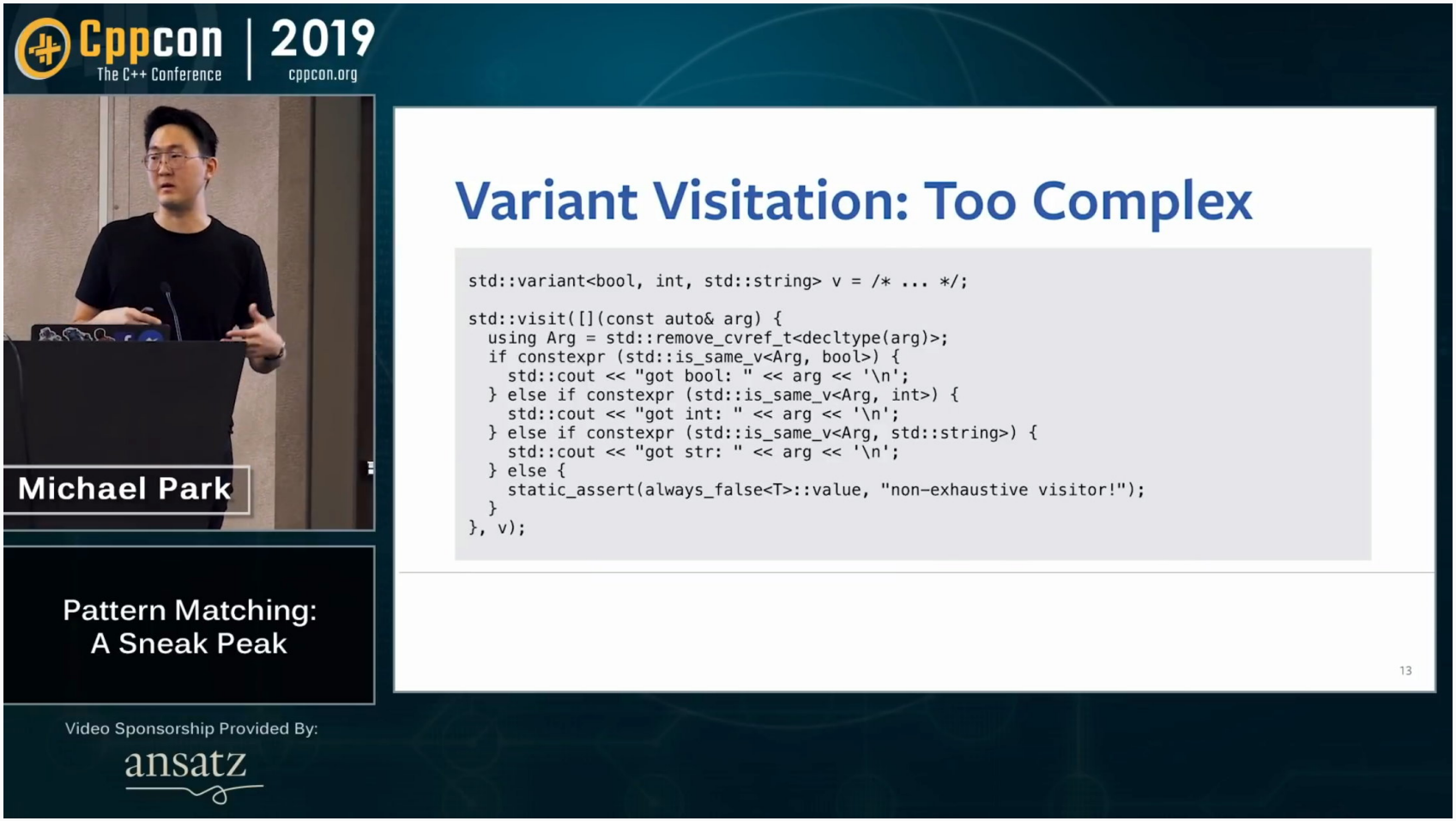

Initialisation in modern C++ - Timur Doumler
2019
Problems with list init:
• Difficult to see when it'll call a std::initializer_list constructor, and when it won't
• std::initializer_list doesn't work with move-only types
• Useless in templates
(you can't write a make_unique that works for aggregates!)
• Does not work with macros at all:
assert(Foo{2, 3}); // This breaks the preprocessor :(
97
cpponsea.uk
@cpponsea

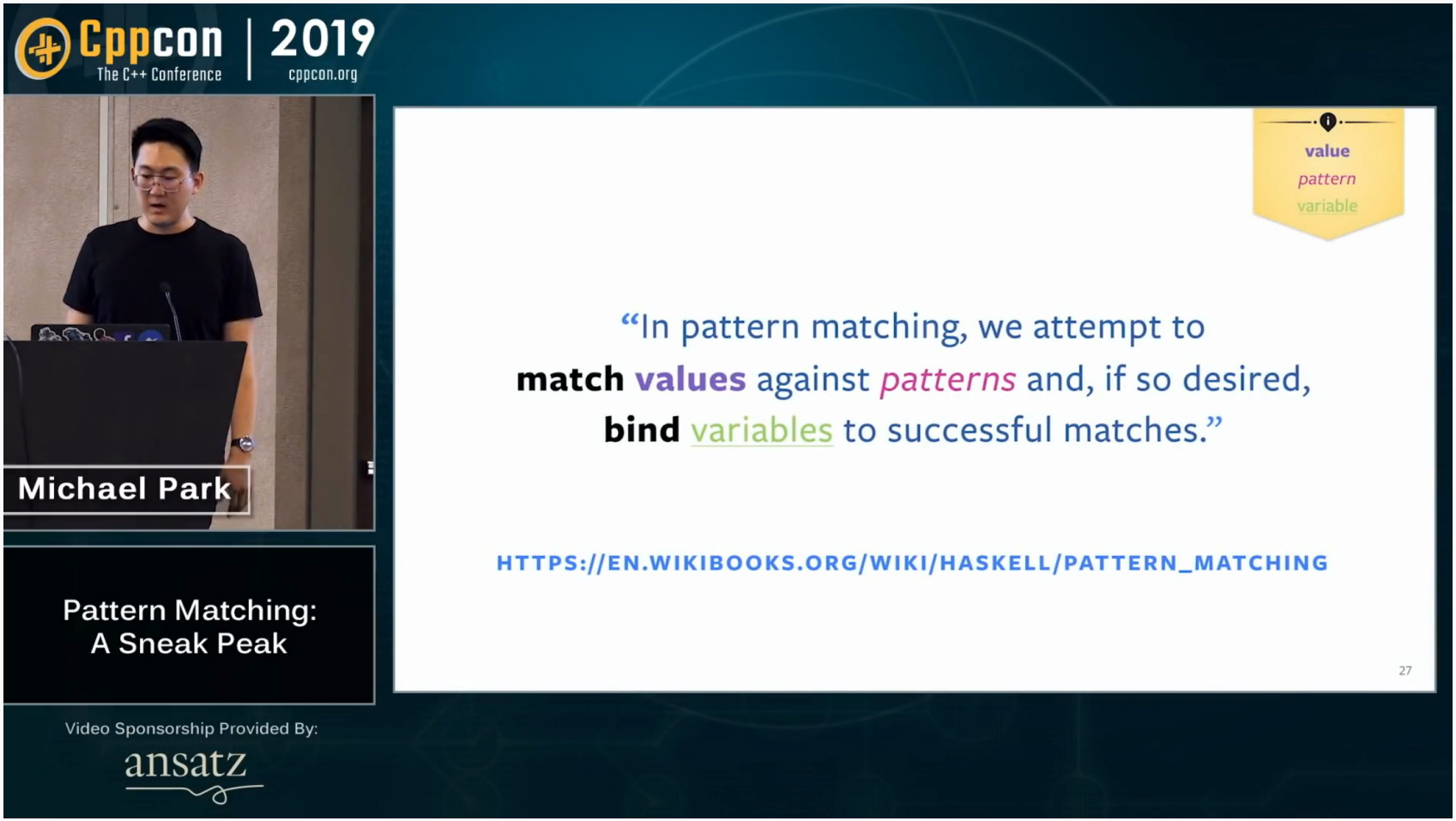

Initialisation in modern C++ - Timur Doumler
2019
Aggregate initialisation from parens
http://wg21.link/p0960
struct Foo {
int i;
int j;
};
Foo foo(1, 2); // will work in C++20!
int arr[3](0, 1, 2); // will work in C++20!
Idea: in C++20, ( ) and {} will do the same thing!
Except:
• ( ) does not call std::initializer_list constructors
• {} does not consider narrowing conversions
100
cpponsea.uk
@cpponsea


Initialisation in modern C++ - Timur Doumler
2019
Recommendations:
• Use auto
• Use direct member initialisers (DMIs)
• Use `= value` for int and other simple value types
• Use `= {args}` for aggregate-init, std::initializer_list, DMIs
→ Recommendation for aggregates might change for C++20!)
• Use `{}` for value-init
• Use `(args)` to call constructors that take arguments
→ This is the controversial one. Other people say: use `{args}`
101
101
cpponsea.uk
@cpponsea

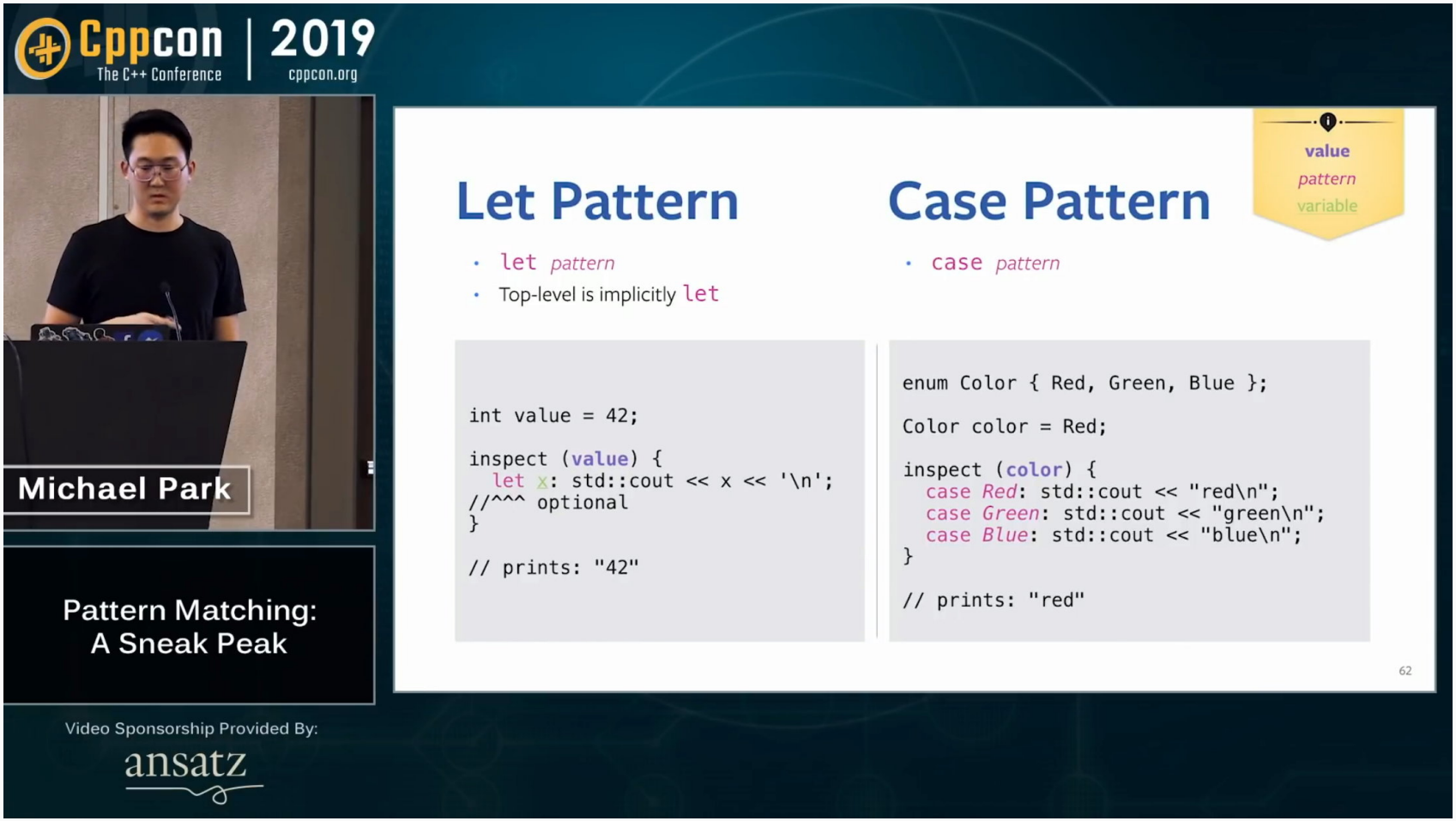
Video

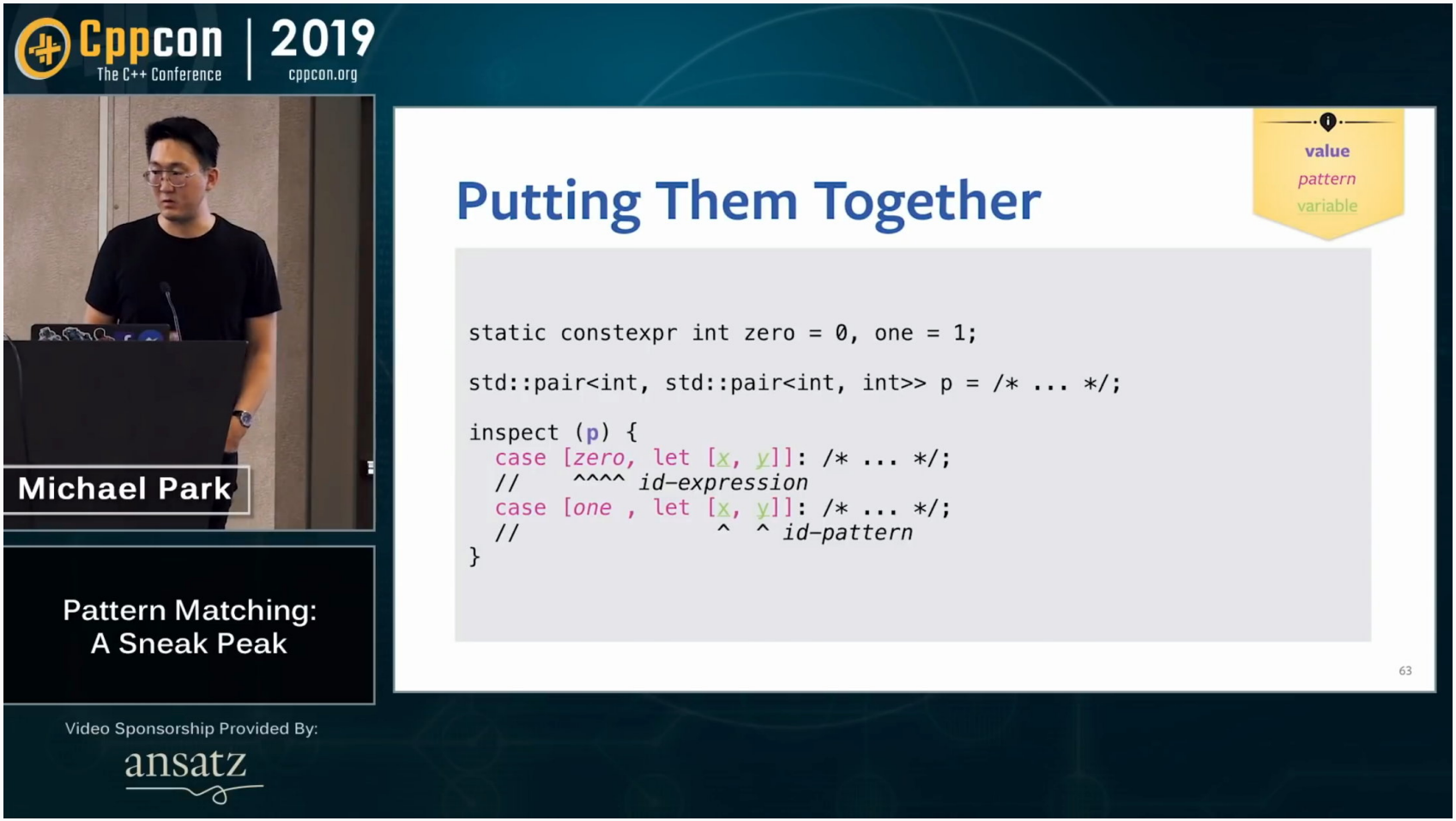

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Variant Visitation: Too Complex
std::variant<bool, int, std::string> v = /* ... */;
std::visit(overload{
[](bool b) { std::cout << "got bool: " << b << '\n'; },
[](int n) { std::cout << "got int: " << n << '\n'; },
[](const std::string& s) { std::cout << "got str: " << s << '\n'; }
}, v);
template <typename... Fs>
struct overload : Fs... { using Fs::operator()...; };
template <typename... Fs>
overload(Fs... fs) -> overload<Fs...>;
12

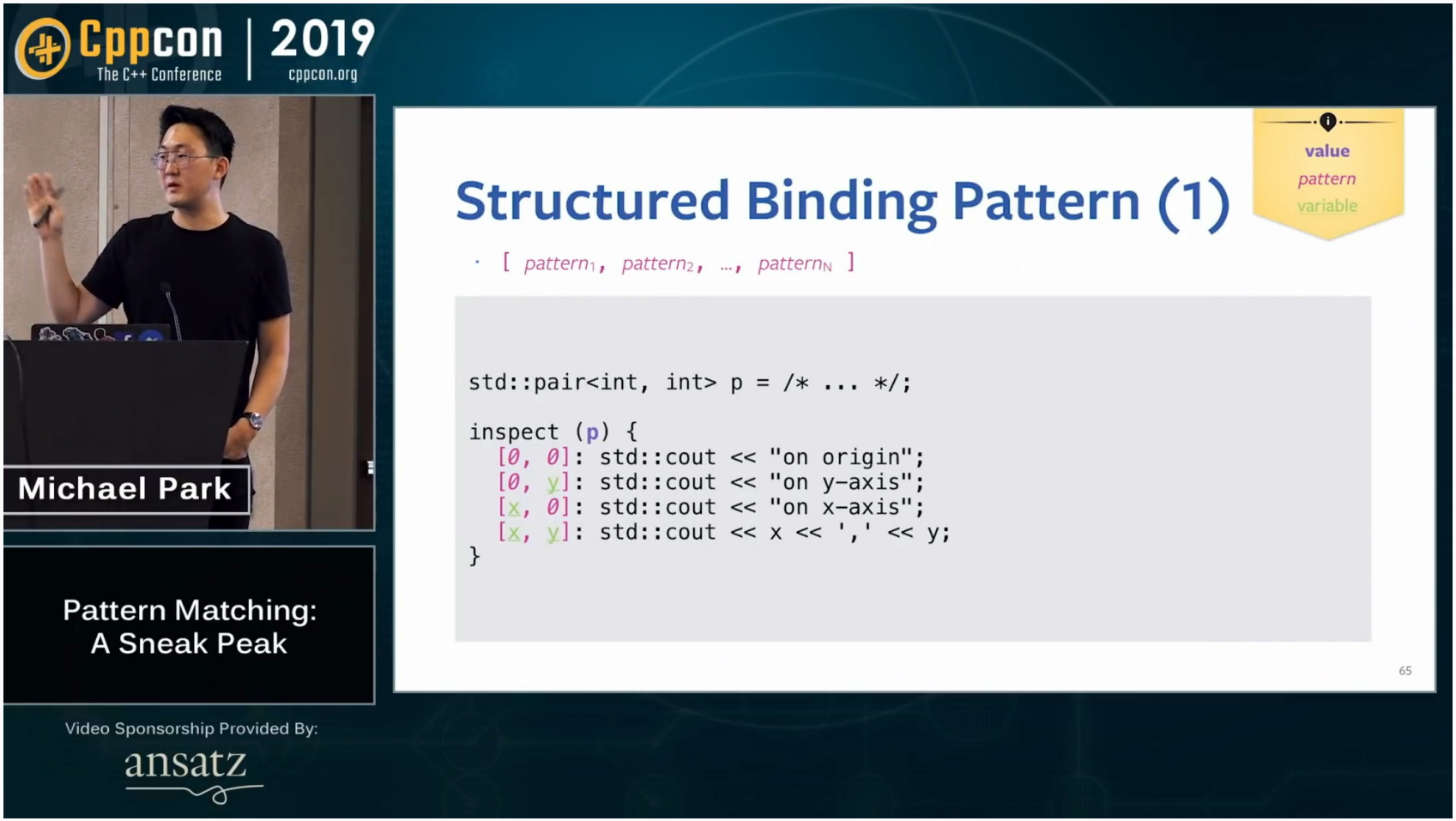

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Variant Visitation: Too Complex
std::variant<bool, int, std::string> v = /* ... */;
std::visit([](const auto& arg) {
using Arg = std::remove_cvref_t<decltype(arg)>;
if constexpr (std::is_same_v<Arg, bool>) {
std::cout << "got bool: " << arg << '\n';
} else if constexpr (std::is_same_v<Arg, int>) {
std::cout << "got int: " << arg << '\n';
} else if constexpr (std::is_same_v<Arg, std::string>) {
std::cout << "got str: " << arg << '\n';
} else {
static_assert(always_false<T>::value, "non-exhaustive visitor!");
}
}, v);
13

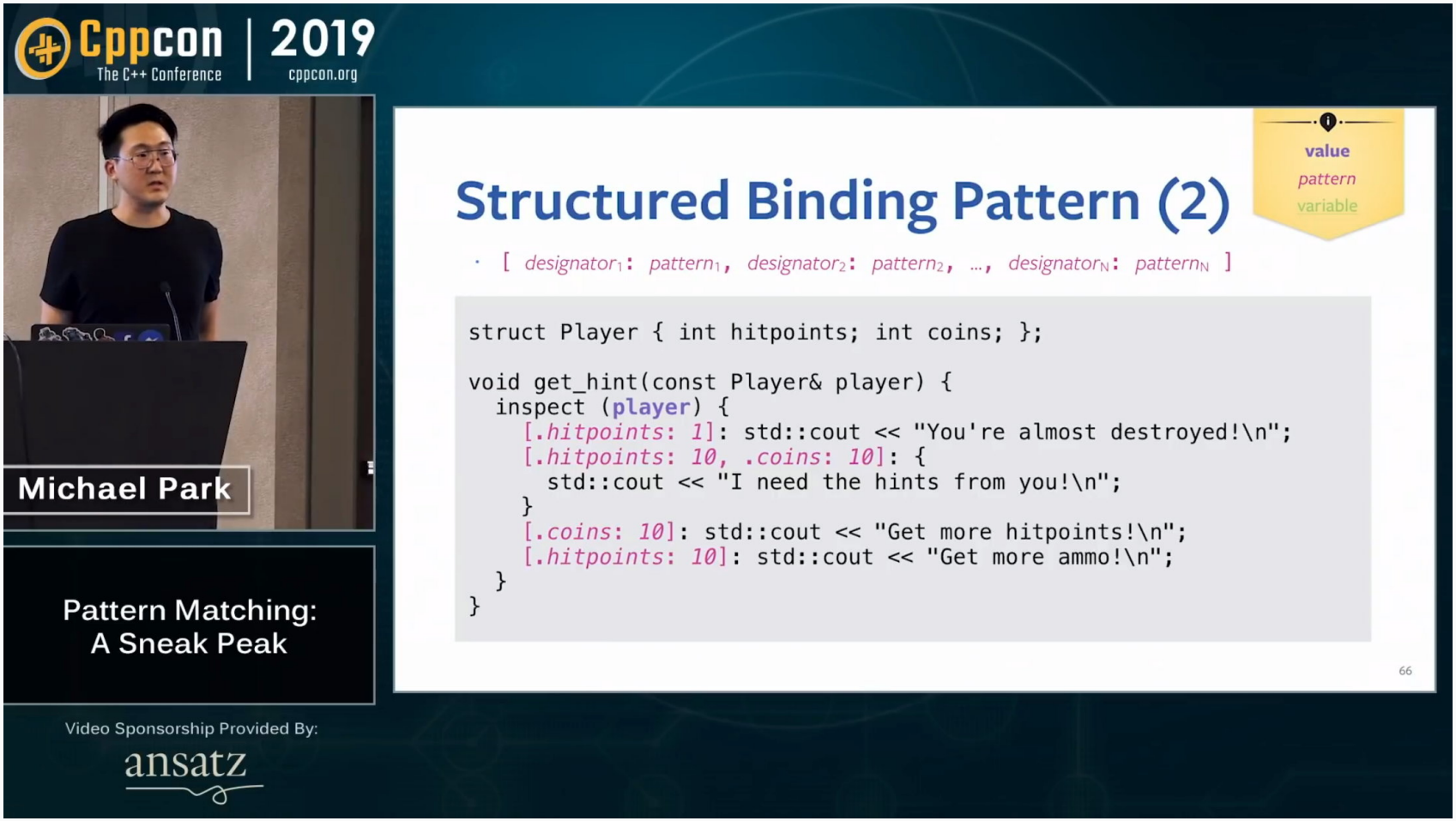

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
value
pattern
variable
"In pattern matching, we attempt to
match values against patterns and, if so desired,
bind variables to successful matches."
HTTPS://EN.WIKIBOOKS.ORG/WIKI/HASKELL/PATTERN_MATCHING
27

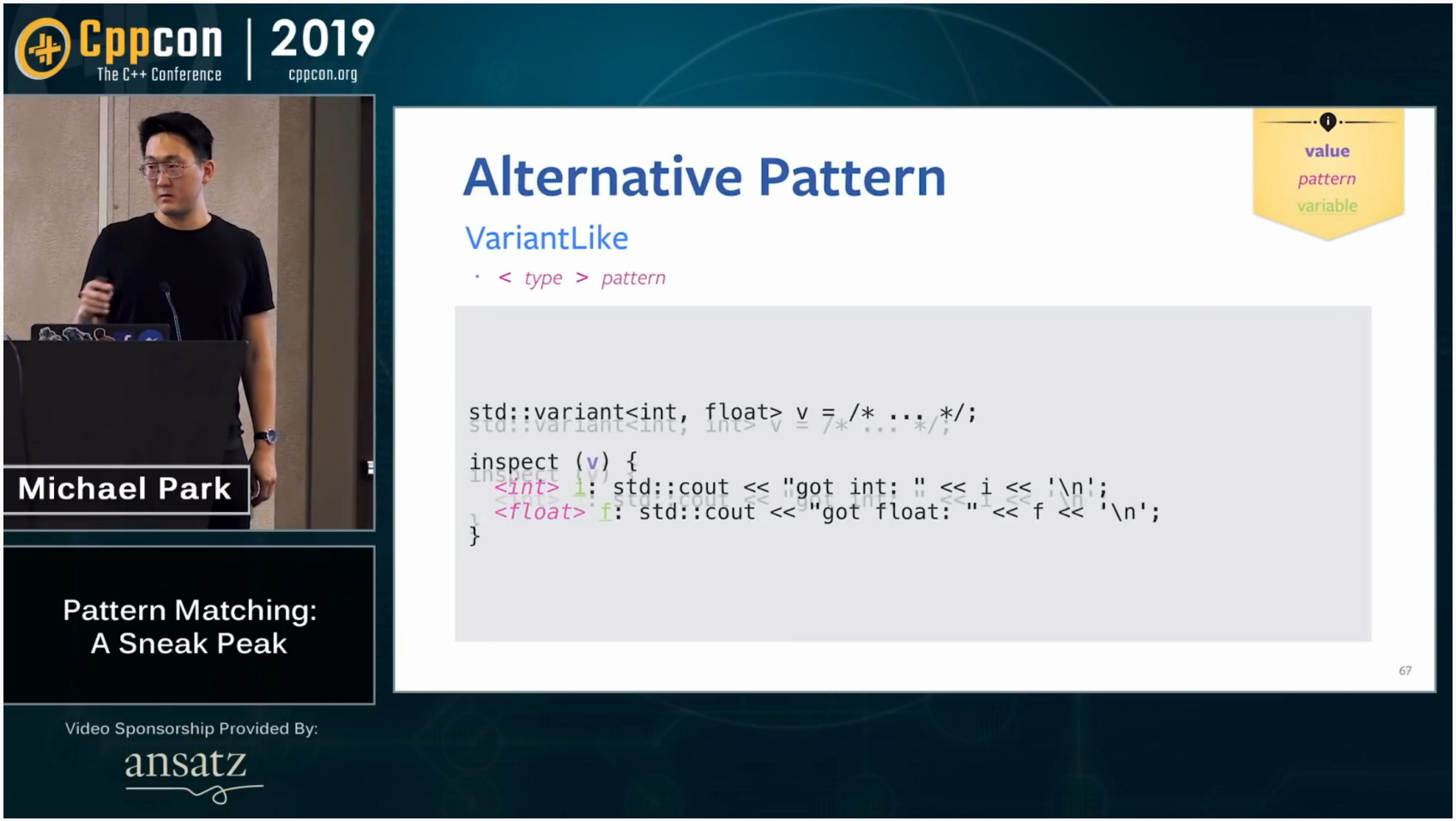
CppCon 2019: Michael Park – Pattern Matching: A Sneak Peek

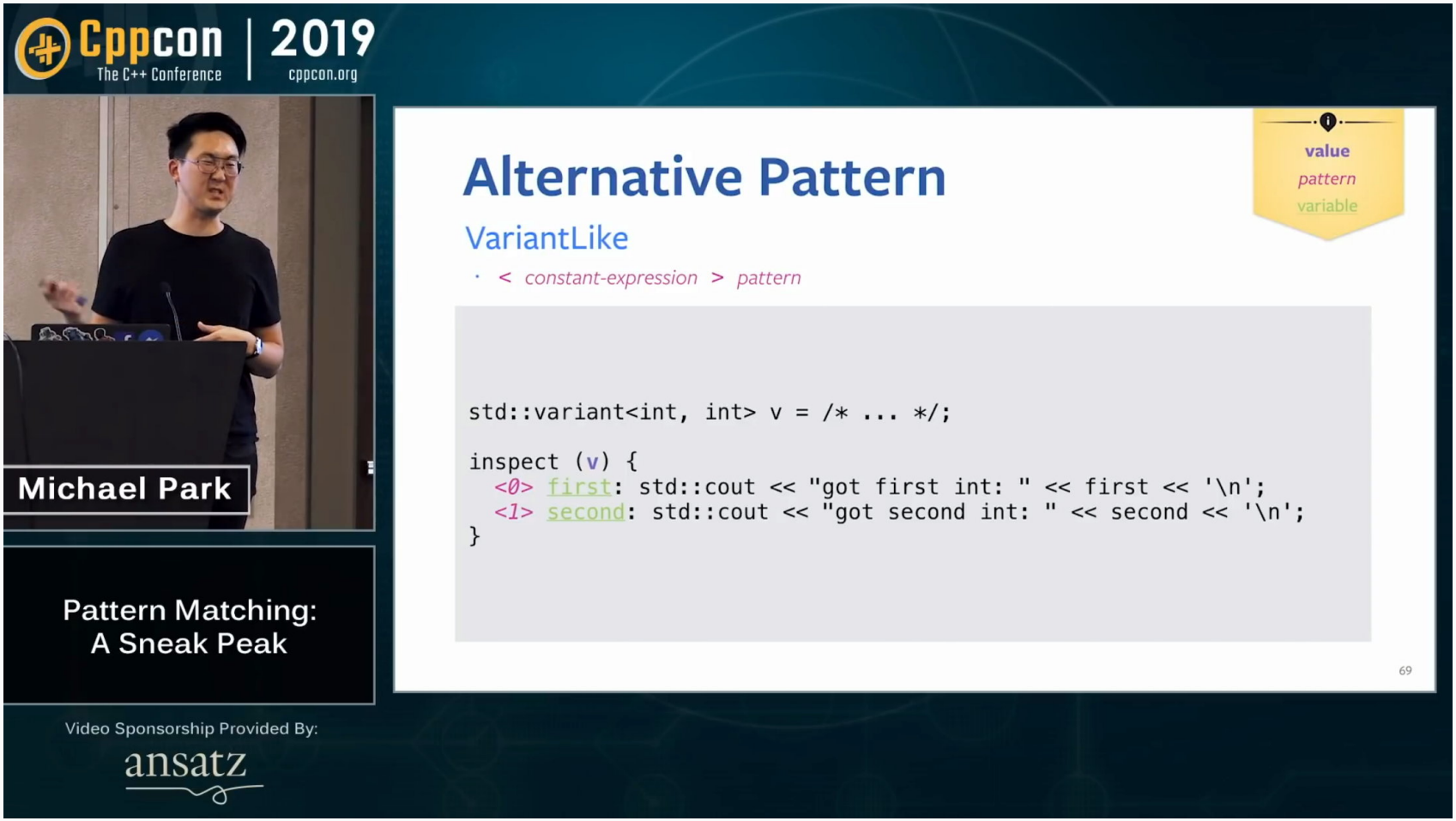


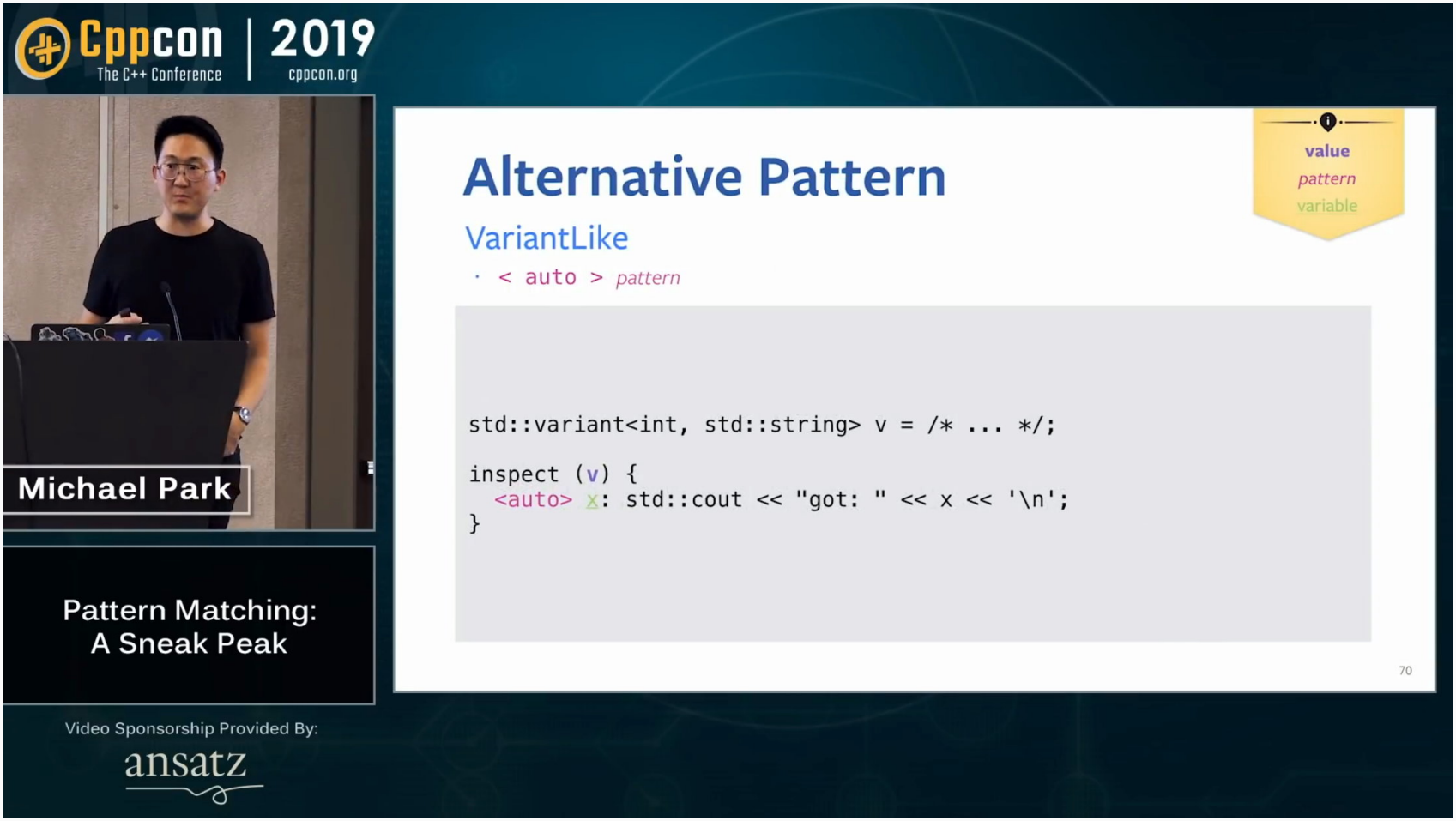

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Putting Them Together
value
pattern
variable
static constexpr int zero = 0, one = 1;
std::pair<int, std::pair<int, int>> p = /* ... */;
inspect (p) {
case [zero, let [x, y]]: /* ... */;
// ^^^^ id-expression
case [one , let [x, y]]: /* ... */;
// ^ ^ id-pattern
}
63

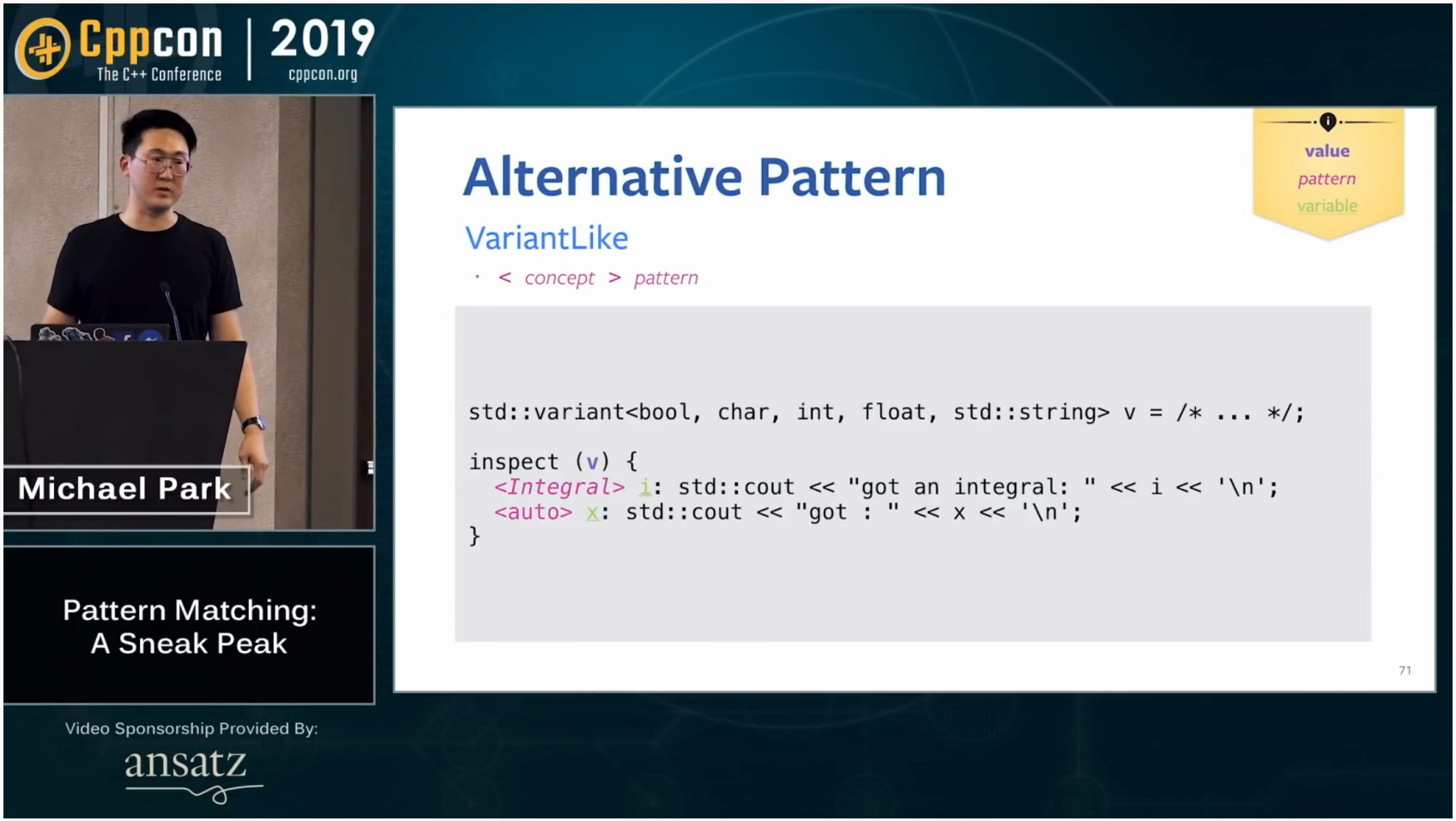

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Structured Binding Pattern (1)
value
pattern
variable
[pattern1, pattern2, ..., patternN]
std::pair<int, int> p = /* ... */;
inspect (p) {
[0, 0]: std::cout << "on origin";
[0, y]: std::cout << "on y-axis";
[x, 0]: std::cout << "on x-axis";
[x, y]: std::cout << x << ',' << y;
}
65

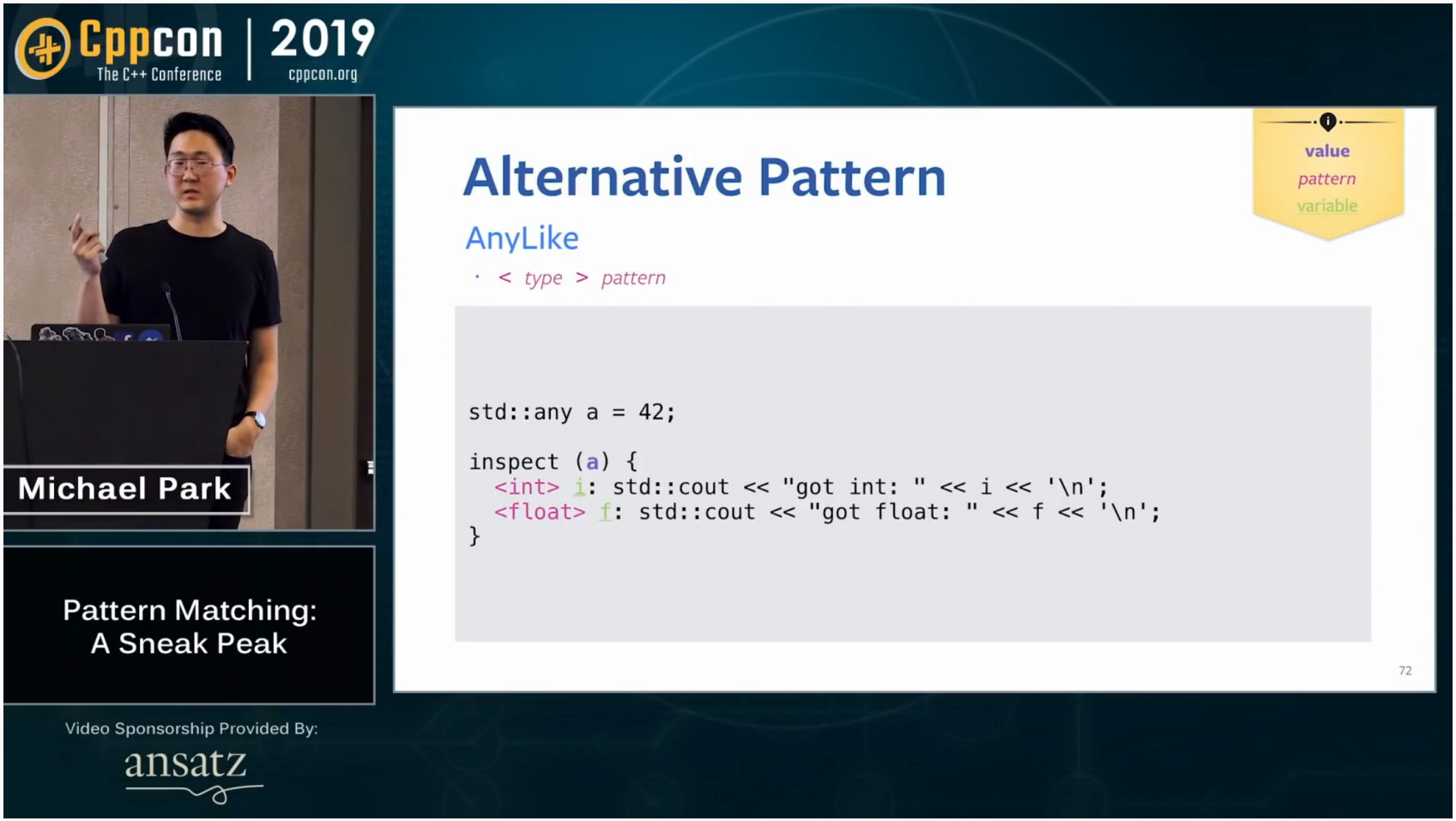

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Structured Binding Pattern (2)
value
pattern
variable
[designator1: pattern1, designator2: pattern2, ..., designatorN: patternN]
struct Player { int hitpoints; int coins; };
void get_hint(const Player& player) {
inspect (player) {
[.hitpoints: 1]: std::cout << "You're almost destroyed!\n";
[.hitpoints: 10, .coins: 10]: {
std::cout << "I need the hints from you!\n";
}
[.coins: 10]: std::cout << "Get more hitpoints!\n";
[.hitpoints: 10]: std::cout << "Get more ammo!\n";
}
}
66

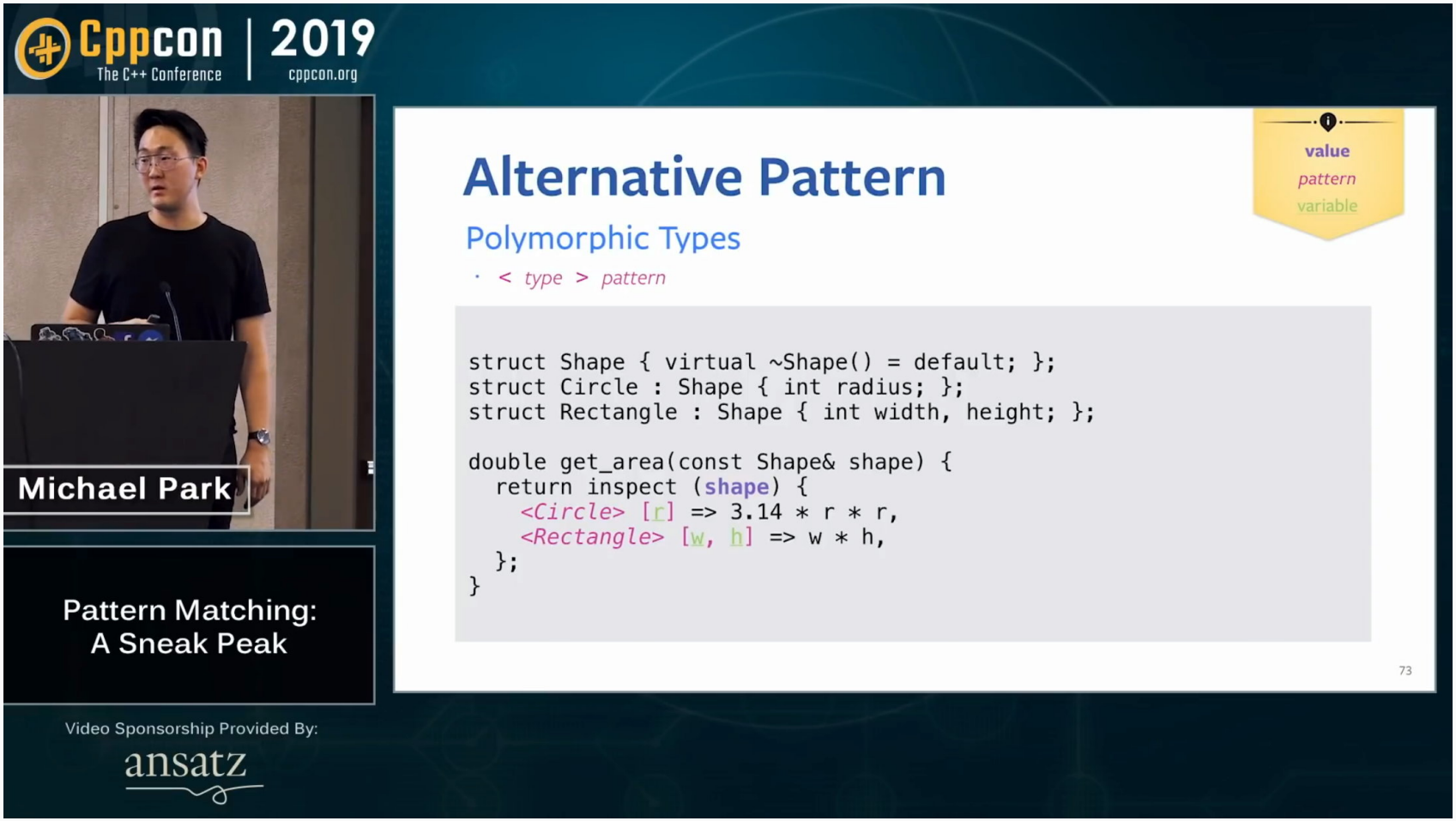

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
VariantLike
· < type > pattern
value
pattern
variable
std::variant<int, float> v = /* ... */;
inspect (v) {
<int> i: std::cout << "got int: " << i << '\n';
<float> f: std::cout << "got float: " << f << '\n';
}
67

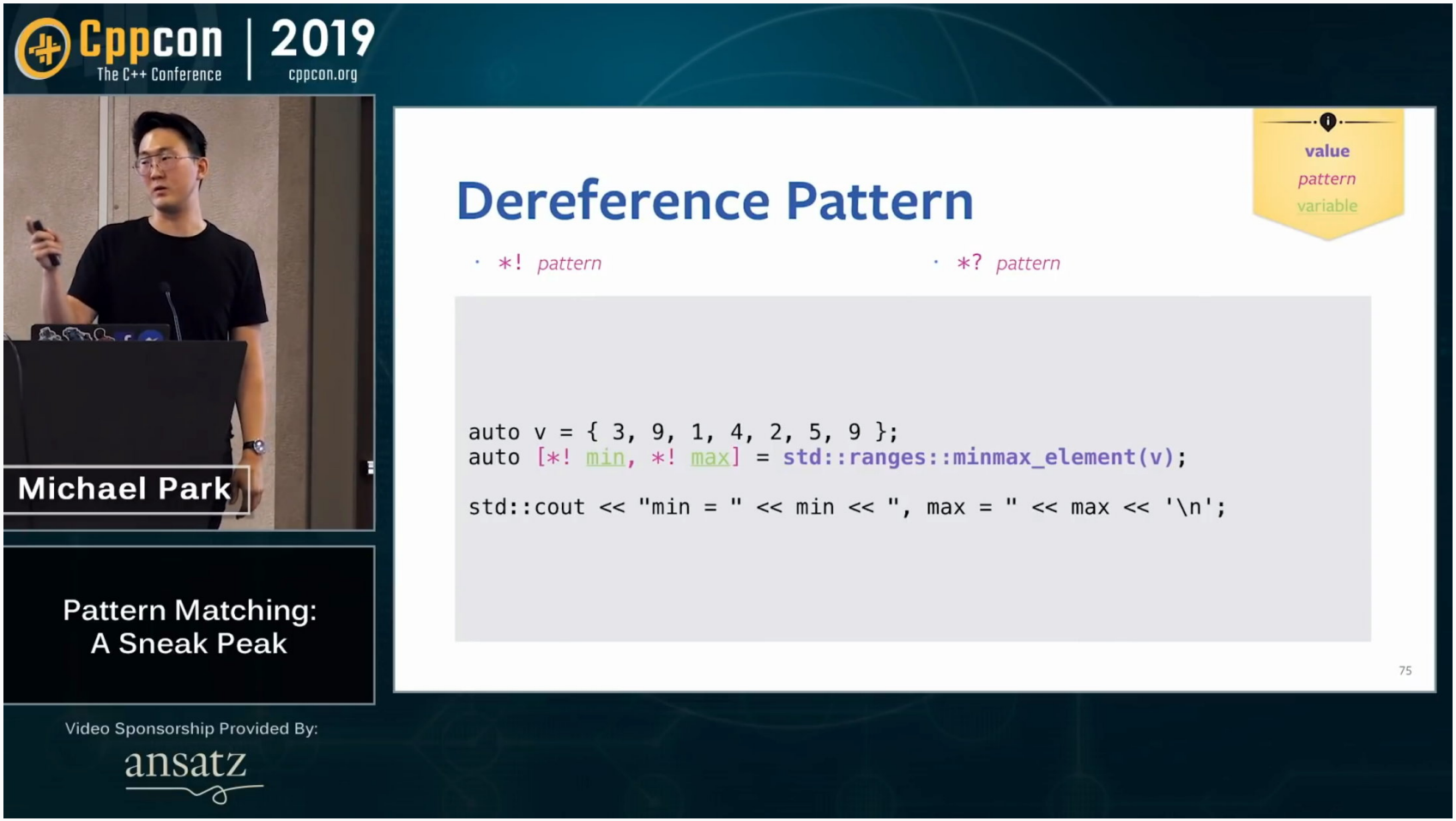

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
VariantLike
· < constant-expression > pattern
value
pattern
variable
std::variant<int, int> v = /* ... */;
inspect (v) {
<0> first: std::cout << "got first int: " << first << '\n';
<1> second: std::cout << "got second int: " << second << '\n';
}
69

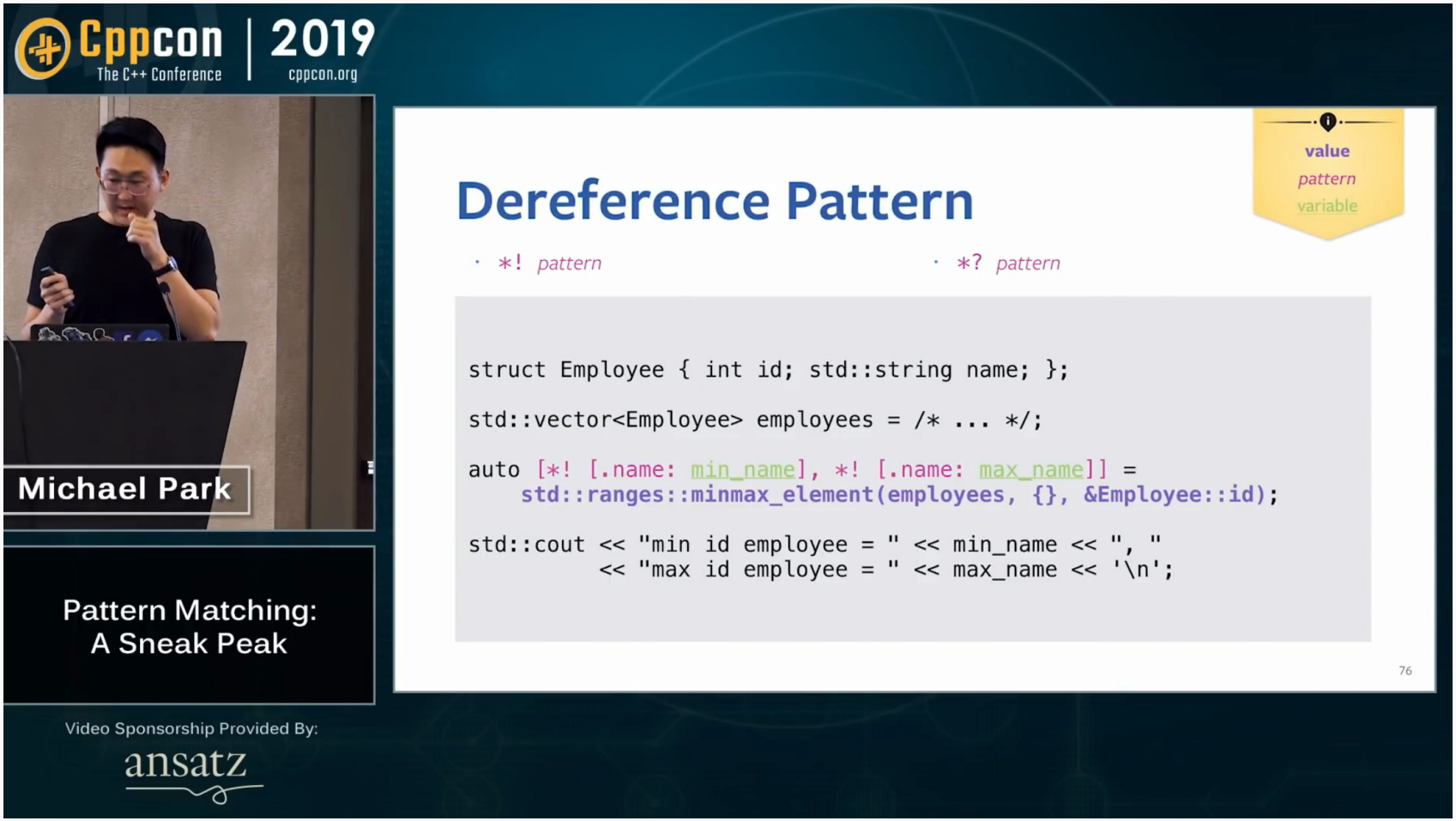

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
VariantLike
· < auto > pattern
value
pattern
variable
std::variant<int, std::string> v = /* ... */;
inspect (v) {
<auto> x: std::cout << "got: " << x << '\n';
}
70

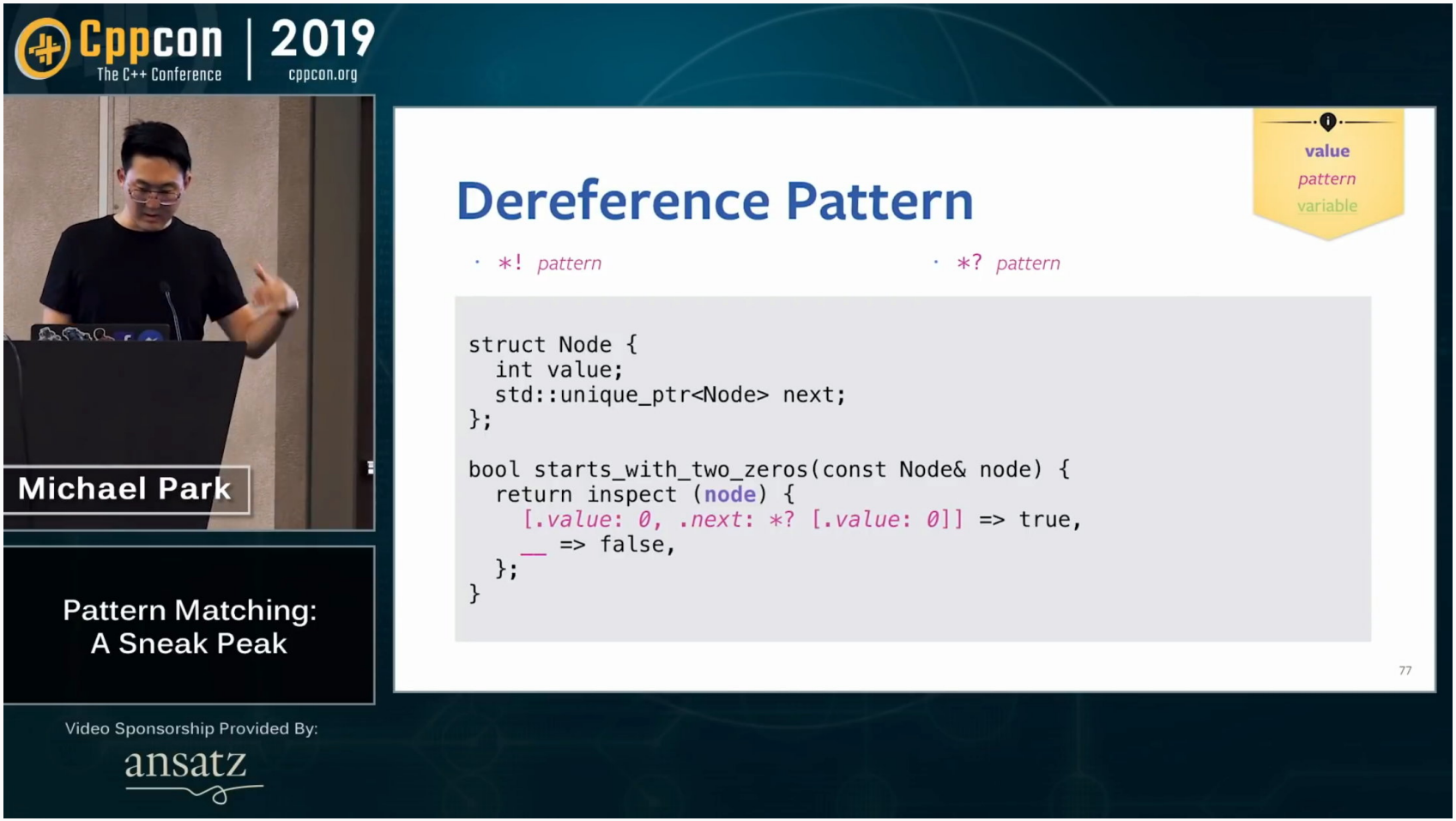

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
VariantLike
· < concept > pattern
value
pattern
variable
std::variant<bool, char, int, float, std::string> v = /* ... */;
inspect (v) {
<Integral> i: std::cout << "got an integral: " << i << '\n';
<auto> x: std::cout << "got : " << x << '\n';
}
71

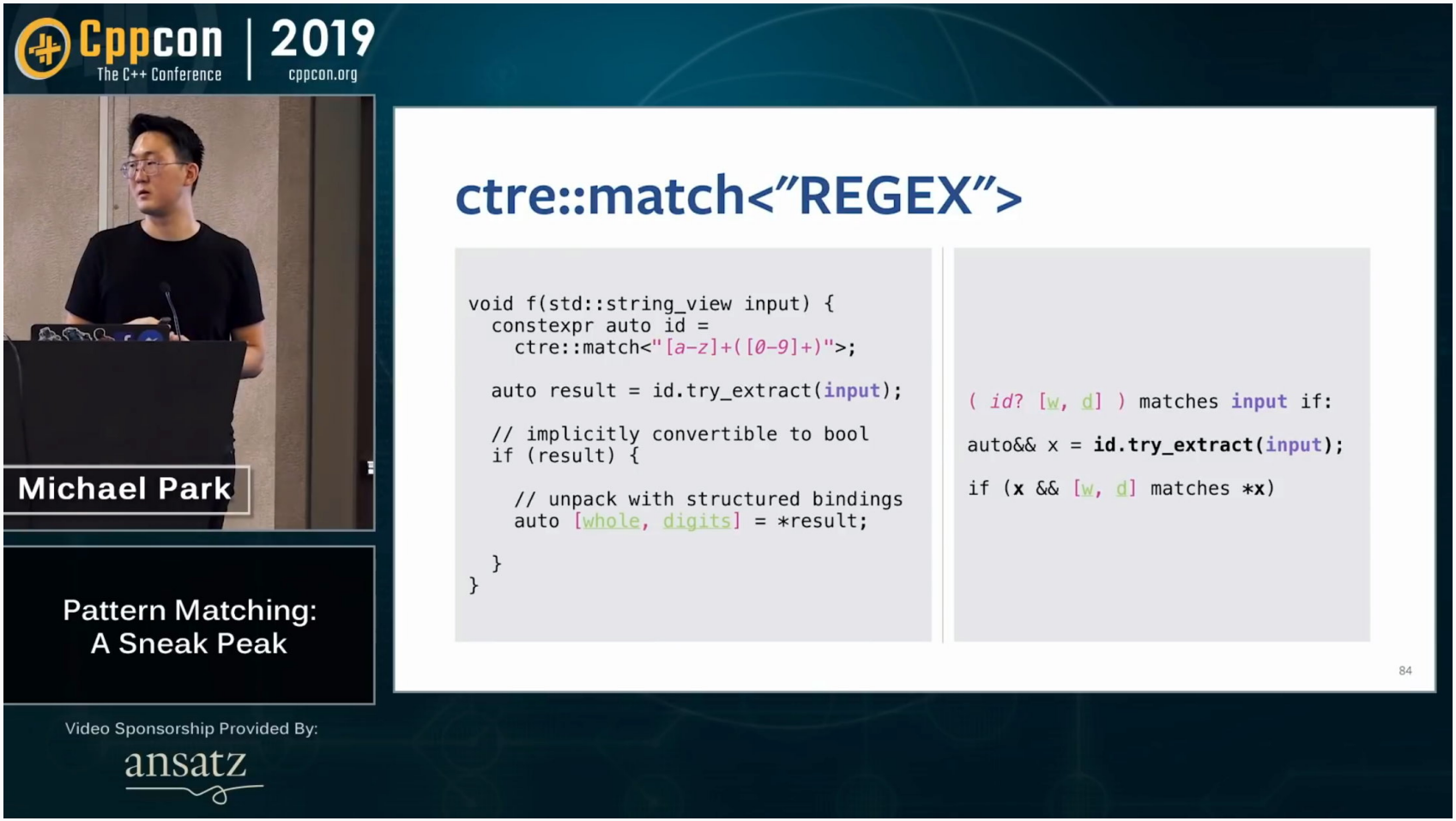

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
AnyLike
· < type > pattern
value
pattern
variable
std::any a = 42;
inspect (a) {
<int> i: std::cout << "got int: " << i << '\n';
<float> f: std::cout << "got float: " << f << '\n';
}
72

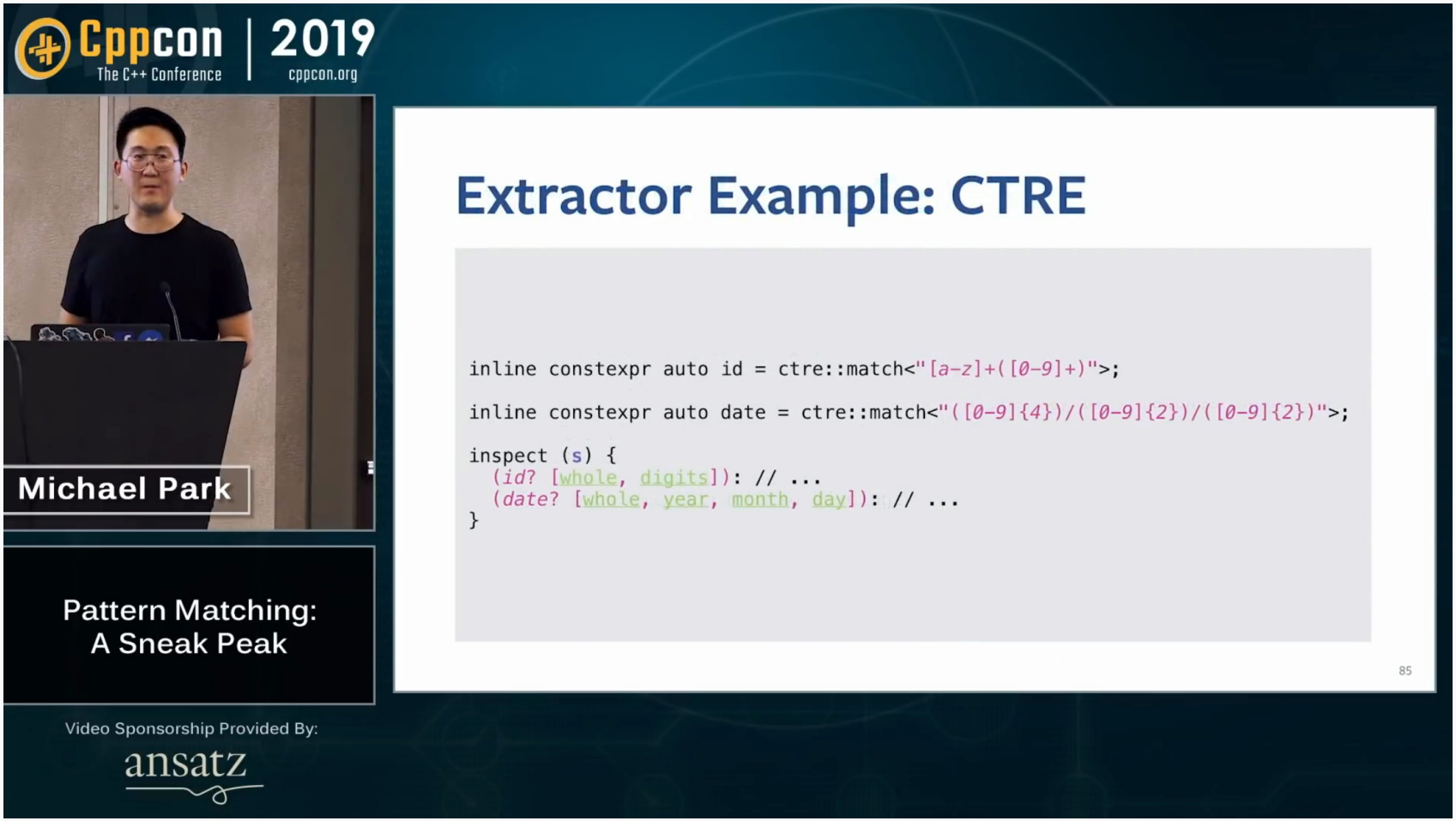

Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Alternative Pattern
Polymorphic Types
· < type > pattern
value
pattern
variable
struct Shape { virtual ~Shape() = default; };
struct Circle : Shape { int radius; };
struct Rectangle : Shape { int width, height; };
double get_area(const Shape& shape) {
return inspect (shape) {
<Circle> [r] => 3.14 * r * r,
<Rectangle> [w, h] => w * h,
};
}
73


Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Dereference Pattern
value
pattern
variable
· *! pattern
· *? pattern
auto v = { 3, 9, 1, 4, 2, 5, 9 };
auto [*! min, *! max] = std::ranges::minmax_element(v);
std::cout << "min = " << min << ", max = " << max << '\n';
75


Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Dereference Pattern
value
pattern
variable
· *! pattern
· *? pattern
struct Employee { int id; std::string name; };
std::vector<Employee> employees = /* ... */;
auto [*! [.name: min_name], *! [.name: max_name]] =
std::ranges::minmax_element(employees, {}, &Employee::id);
std::cout << "min id employee = " << min_name << ", "
<< "max id employee = " << max_name << '\n';
76


Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Dereference Pattern
value
pattern
variable
· *! pattern
· *? pattern
struct Node {
int value;
std::unique_ptr<Node> next;
};
bool starts_with_two_zeros(const Node& node) {
return inspect (node) {
[.value: 0, .next: *? [.value: 0]] => true,
__ => false,
};
}
77


Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
ctre::match<"REGEX">
void f(std::string_view input) {
constexpr auto id =
ctre::match<"[a-z]+([0-9]+)">;
auto result = id.try_extract(input);
// implicitly convertible to bool
if (result) {
// unpack with structured bindings
auto [whole, digits] = *result;
}
}
(id? [w, d]) matches input if:
auto&& x = id.try_extract(input);
if (x && [w, d] matches *x)
84


Cppcon
The C++ Conference
2019
cppcon.org
Michael Park
Pattern Matching:
A Sneak Peak
Video Sponsorship Provided By:
ansatz
Extractor Example: CTRE
inline constexpr auto id = ctre::match<"[a-z]+([0-9]+)">;
inline constexpr auto date = ctre::match<"([0-9]{4})/([0-9]{2})/([0-9]{2})">;
inspect (s) {
(id? [whole, digits]): // ...
(date? [whole, year, month, day]): // ...
}
85

A new decade, a new tool: libman

- Colby Pike (vector-of-bool)
- Reddit
- GitHub
- Specification

libman is a new level of indirection between package management and build systems.

dds is Drop-Dead Simple build and package manager.

- CppCon 2019: Robert Schumacher “How to Herd 1,000 Libraries”

Lucid C++ index

- Lucid C++ index
- Reddit

A hidden gem: `inner_product`

- Article

A hidden gem: `inner_product`

4

Conor Hoekstra @code_report
@cjdb_ns & @TartanLlama

This makes me so incredibly happy! I literally just yesterday googled, C++17 / C++20 zip to see if they had anything, because I wrote some code in both C++ and #Python and Python was so much more beautiful.

```
int solve(int h, vector<int> w, vector<int> l) {
   int p = 0;
   for (int i = 0; i < w.size(); ++i)
      p = max(p, w[i] - l[i] / 4);
   return max(0, p - h);
}
```

```
def solve(h, w, l):
    p = max(a - b//4 for a, b in zip(w, l))
    return max(0, p - h)
```

29w • 03/12/2018 • 17:47

Conor Hoekstra @code_report
@cjdb_ns & @TartanLlama

Also, I just discovered std::inner_product - a beautiful temporary solution to a lack of zip. #cpp #inner_product

```
int solve(int h, vector<int> w, vector<int> l) {
   return max(0, inner_product(begin(w), end(w), begin(l), 0,
      [](auto a, auto b) { return max(a, b); },
      [](auto a, auto b) { return a - b / 4; }) - h);
}
```

27w • 16/12/2018 • 09:30

Lingo 0.1.0: Text encoding for modern C++

- GitHub: C++11, MIT
- Reddit

Argumentum

> *Argumentum is a C++17 library for writing command-line program interfaces, inspired by Python argparse*

- GitHub: C++17, MPL
- Reddit

Structured Exceptions (Win32) and C++

- Raymond Chen: How can I handle both structured exceptions and C++ exceptions potentially coming from the same source?
  - Reddit
- Raymond Chen: Can I throw a C++ exception from a structured exception?

Book: C++ Move Semantics - The Complete Guide, by Nicolai Josuttis

- Website

> *Move semantics, introduced with C++11, has become a hallmark of modern C++ programming. However, it also complicates the language in many ways. Even after several years of support of move semantics experienced programmers struggle with all the details of move semantics. And style guides still don't recommend the right consequences for programming even of trivial classes.*

- Reddit

> *Happy there is a book. Not happy this requires a book.*

[C++ coroutines] Initial implementation pushed to GCC master

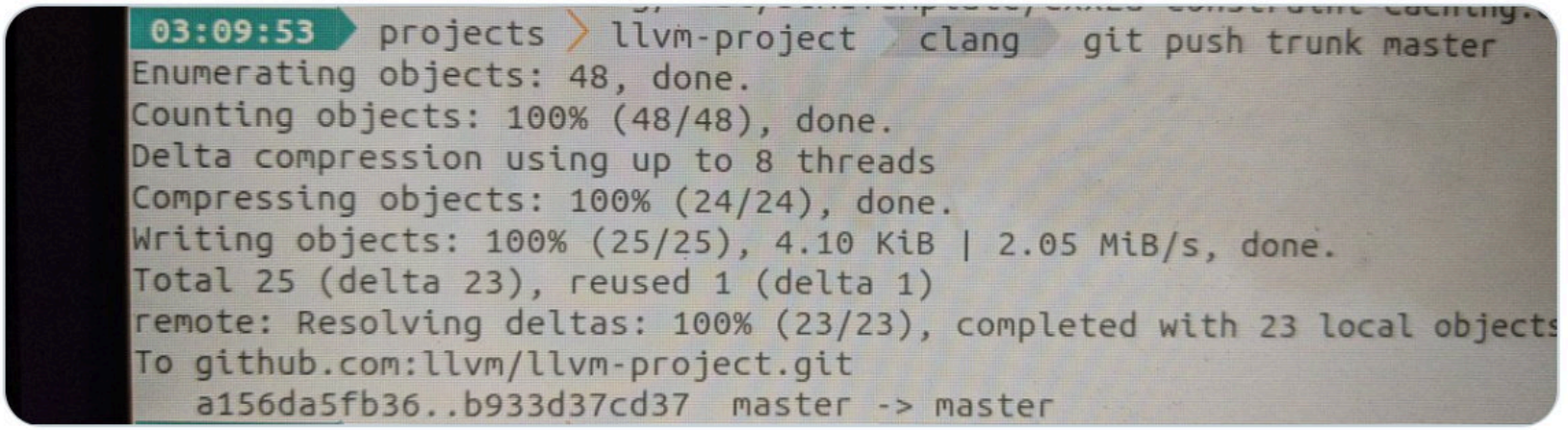
- Message
- Reddit

> *This is not enabled by default (even for -std=c++2a), it needs* ***-fcoroutines***

Concepts pushed to Clang master

Saar Raz
@saarraz1

#clang #concepts #trunk.



2:31 AM · Jan 22, 2020 · Twitter for Android

- Reddit

“Making new friends” idiom by Dan Saks

Wikibooks

> *The goal is to simplify creation of friend functions for a class template.*

```
1  template<typename T>
2  class Foo {
3     T value;
4  public:
5     Foo(const T& t) { value = t; }
6     friend ostream& operator <<(ostream& os, const Foo<T>& b)
7     {
8        return os << b.value;
9     }
10 };
```

- Article

Twitter: Pure virtual function syntax (1/2)

Shafik Yaghmour @shafikyaghmour

Rereading “The Design and Evolution of C++”

= 0

syntax was used for pure virtual function in order to avoid having to add a new keyword such as pure or abstract because the feature was added close to the next release.

Twitter: Pure virtual function syntax (2/2)

13.2.3 Syntax

The curious `=0` syntax was chosen over the obvious alternative of introducing a keyword `pure` or `abstract` because at the time I saw no chance of getting a new keyword accepted. Had I suggested `pure`, Release 2.0 would have shipped without abstract classes. Given a choice between a nicer syntax and abstract classes, I chose abstract classes. Rather than risking delay and incurring the certain fights over `pure`, I used the traditional C and C++ convention of using `0` to represent "not there." The `=0` syntax fits with my view that a function body is the initializer for a function and also with the (simplistic, but usually adequate) view of the set of virtual functions being implemented as a vector of function pointers (§3.5.1). In fact, `=0` is not best implemented by putting a `0` in the `vtbl`. My implementation places a pointer to a function called `__pure_virtual_called` in the `vtbl`; this function can then be defined to give a reasonable run-time error.

I chose a mechanism for specifying individual functions pure rather than a way of declaring a complete class abstract because the pure virtual function notion is more flexible. I value the ability to define a class in stages; that is, I find it useful to define some virtual functions and leave the definition of the rest to further derived classes.

Twitter: Testing Boeing 777

My first exposure to this was a story told to me by Gail Murphy, then my professor in a third-year software engineering course, about the production of the Boeing 787's predecessor, the 777. The 777 was Boeing's first "fly-by-wire" plane. In other words, the software had to work, as it was purely software that was controlling the flaps and rudder and preventing the plane from falling out of the sky. Gail recounted that, due to the criticality of the software, Boeing decided to put all the heads of software engineering on the test flight. During the test flight, the plane started shaking, and the software engineers were able to implement a midflight fix via the turbulence control software.[13] I have yet to find a better example of an organization putting software leaders' skin in the game of high-stakes product development.

The depth of Boeing's understanding of the business implications of