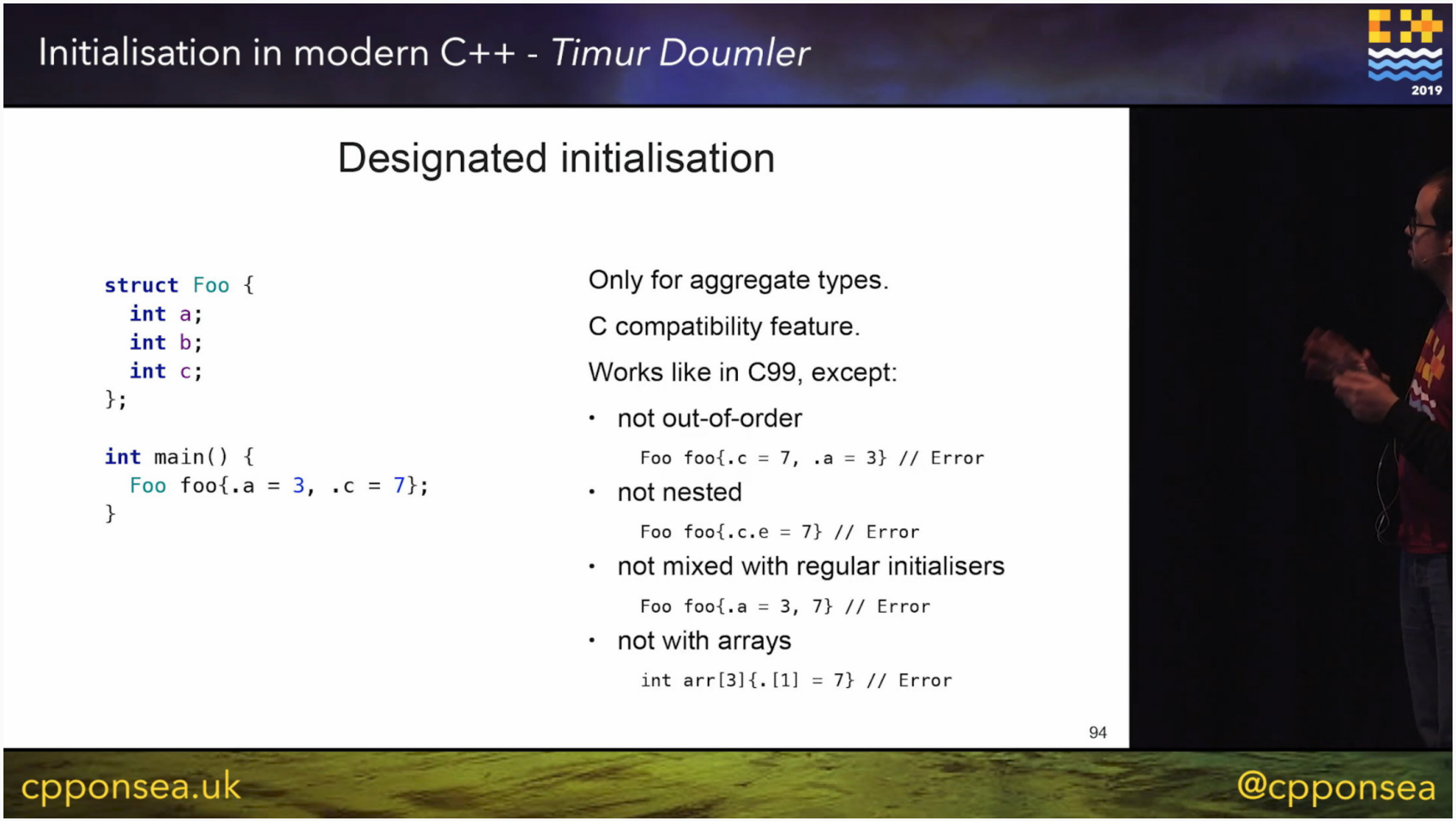
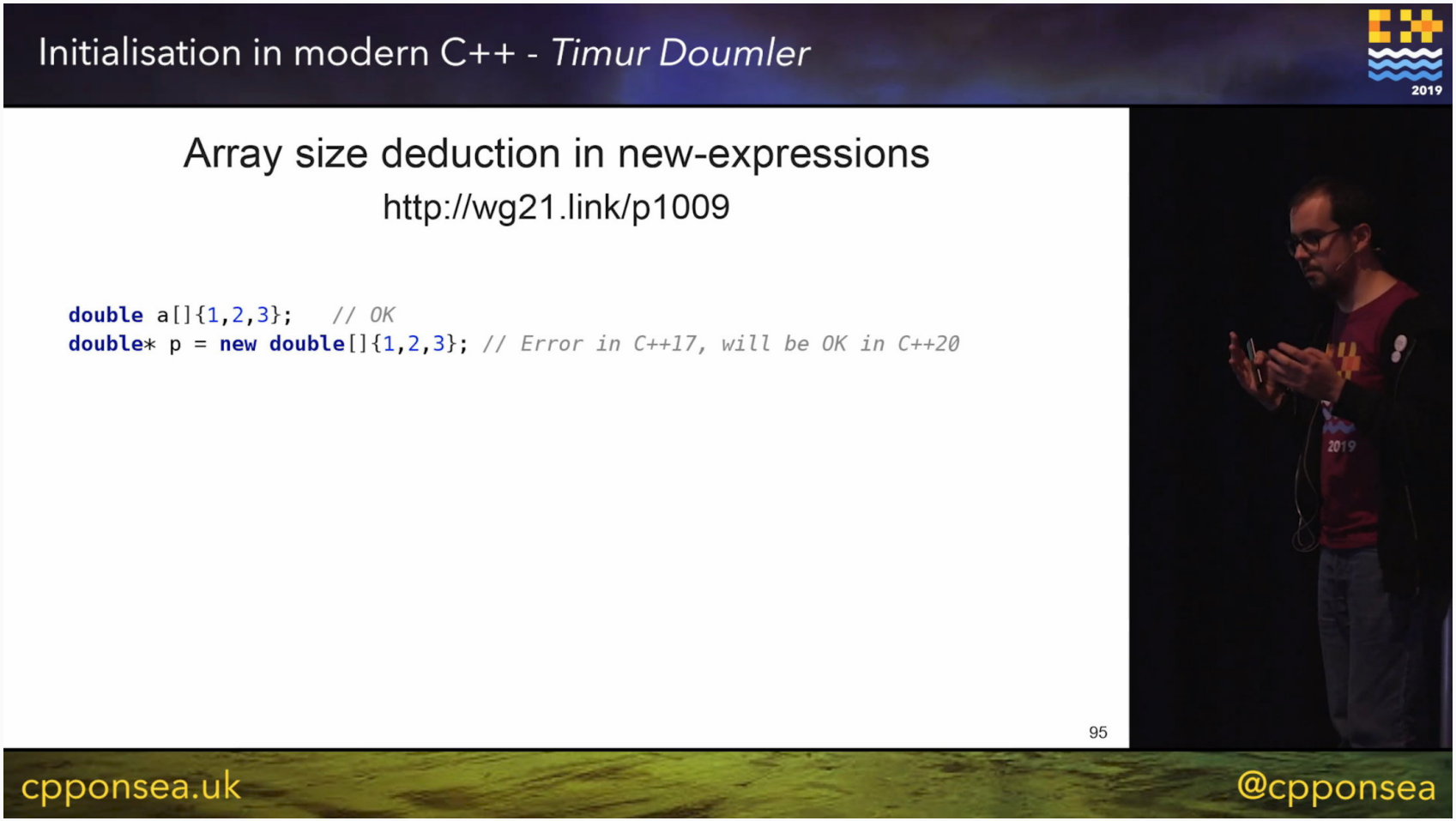
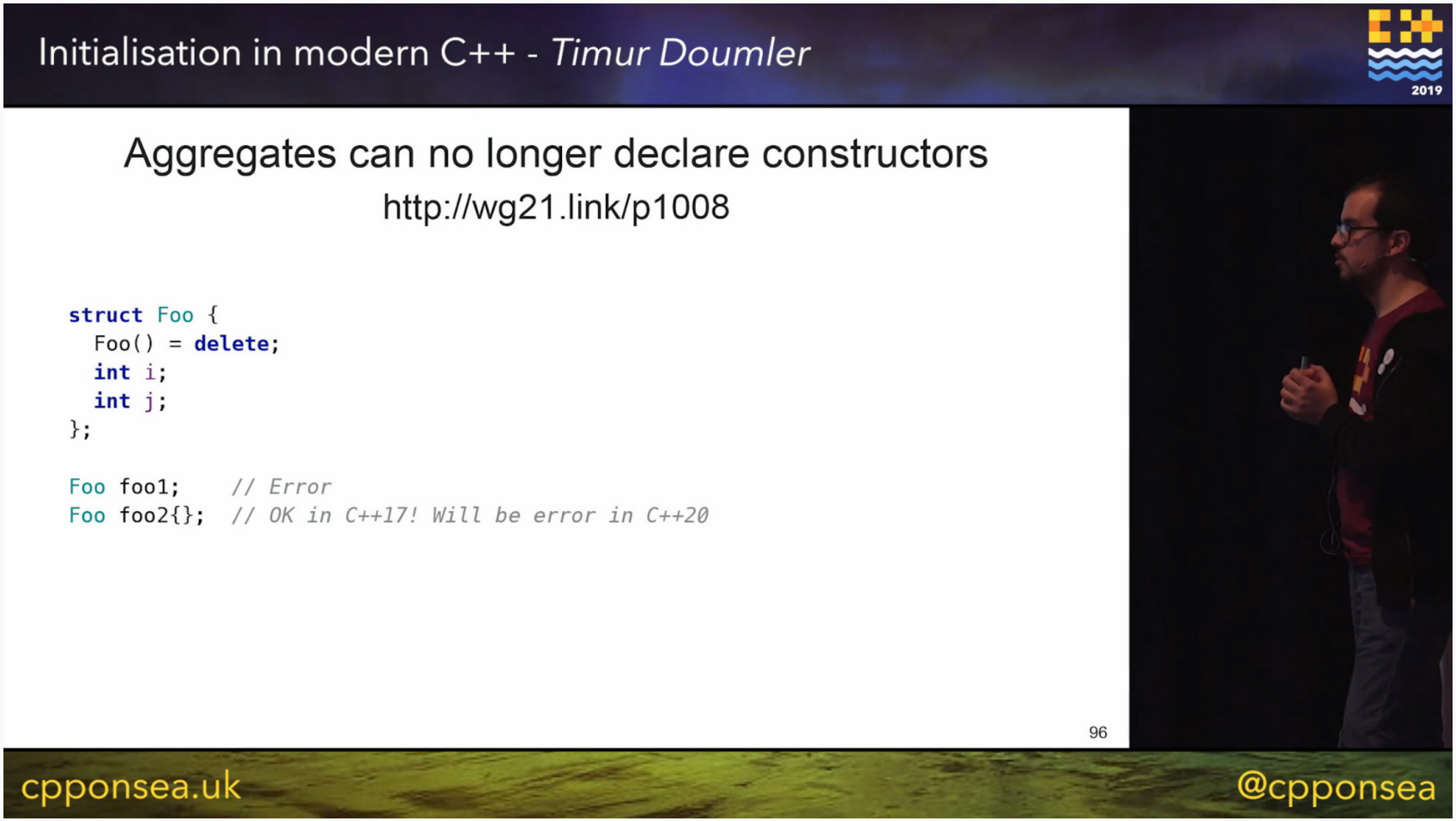
C++ Club Meeting Notes

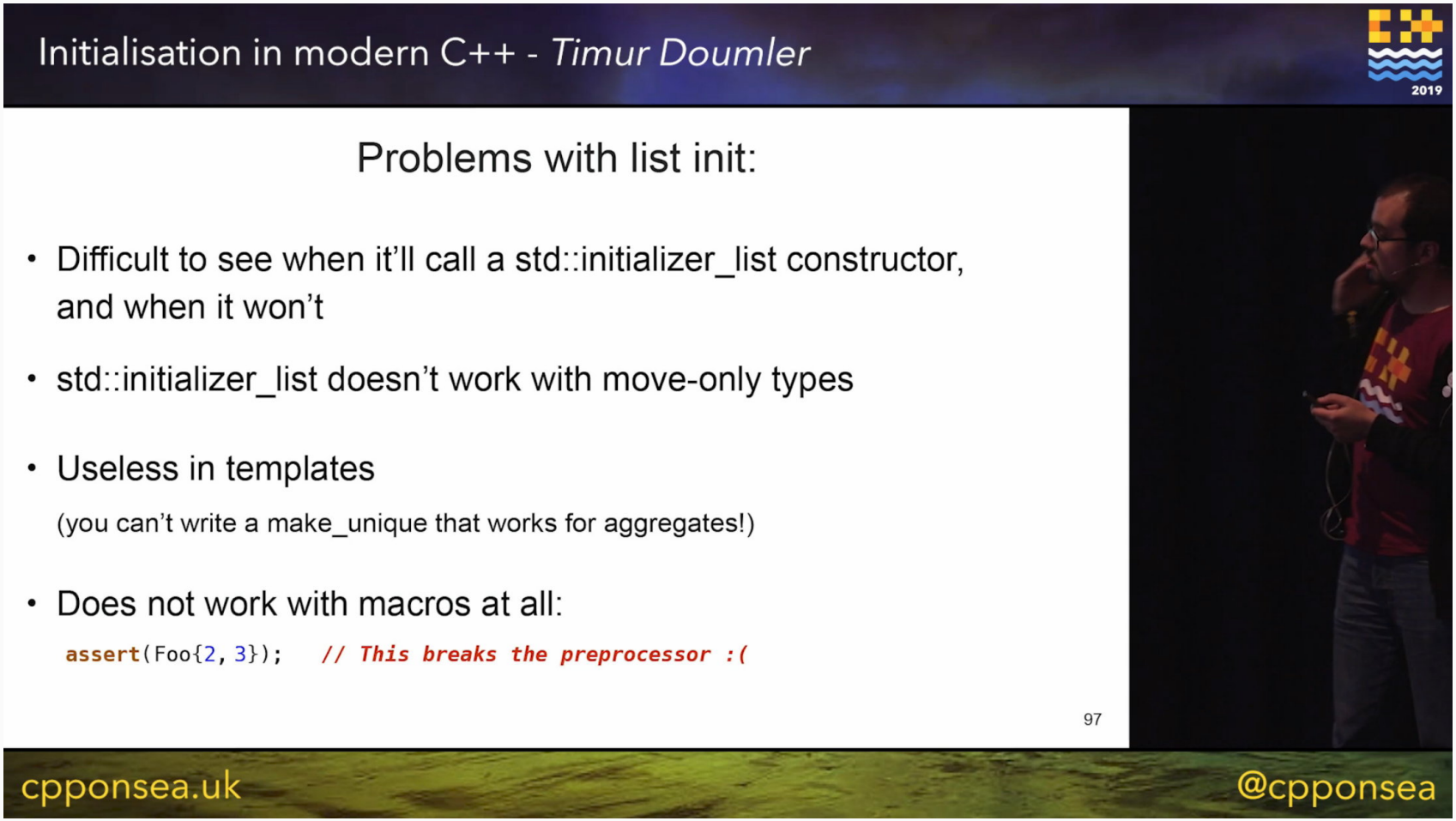
Gleb Dolgich

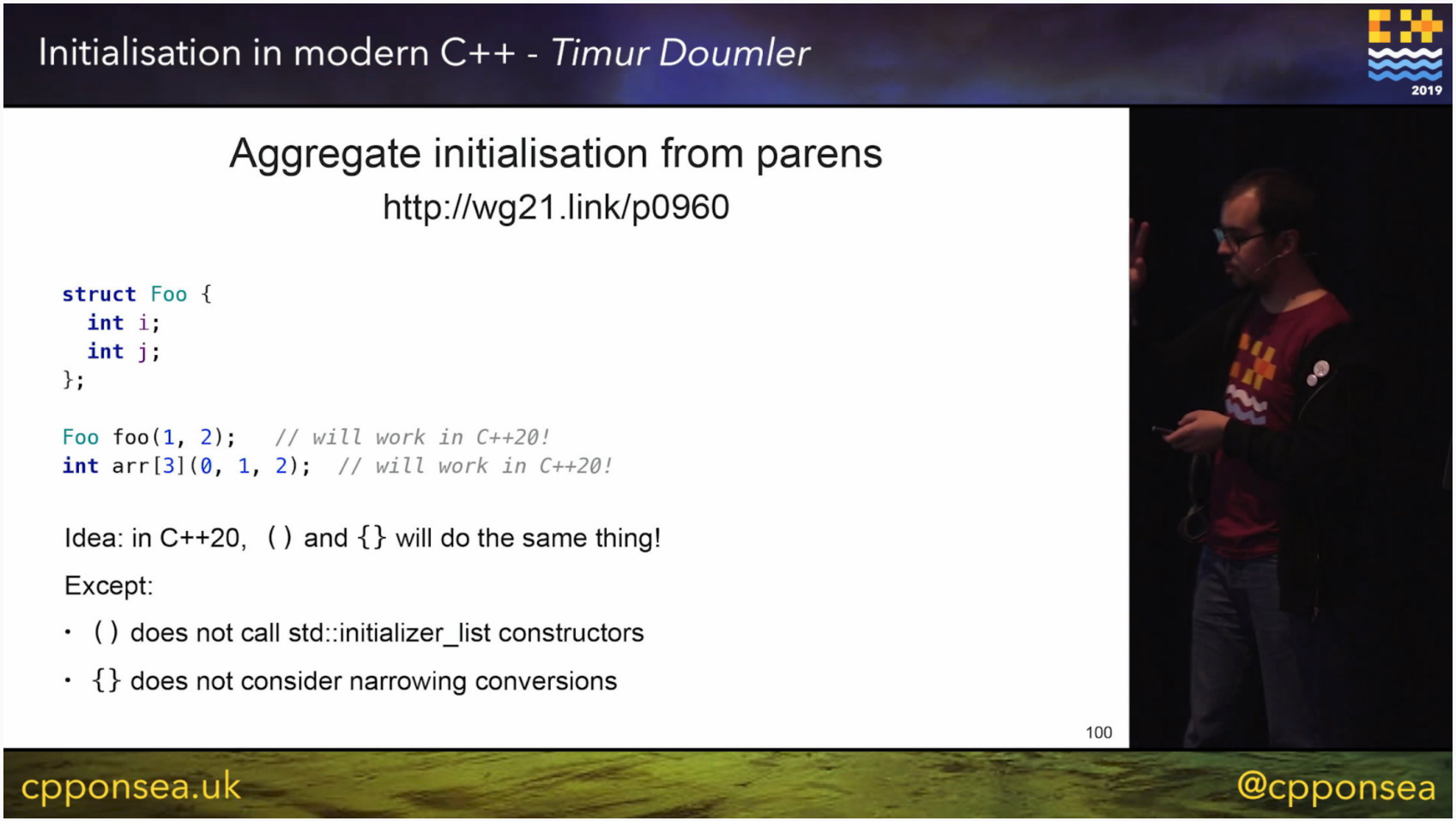
2020-01-30

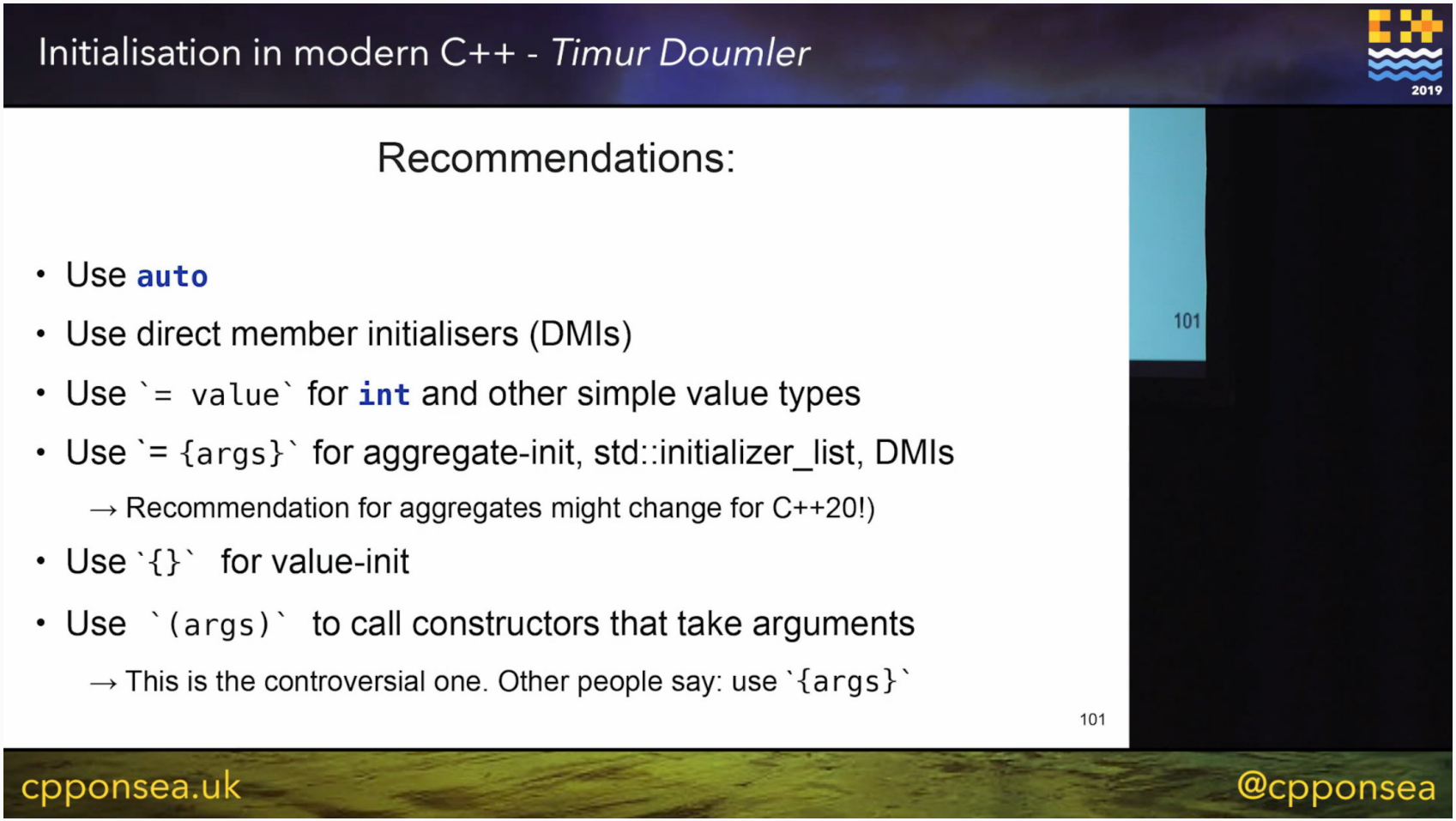

Initialisation in modern C++ - Timur Doumler
2019
Designated initialisation
struct Foo {
int a;
int b;
int c;
};
int main() {
Foo foo{.a = 3, .c = 7};
}
Only for aggregate types.
C compatibility feature.
Works like in C99, except:
• not out-of-order
Foo foo{.c = 7, .a = 3} // Error
• not nested
Foo foo{.c.e = 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

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

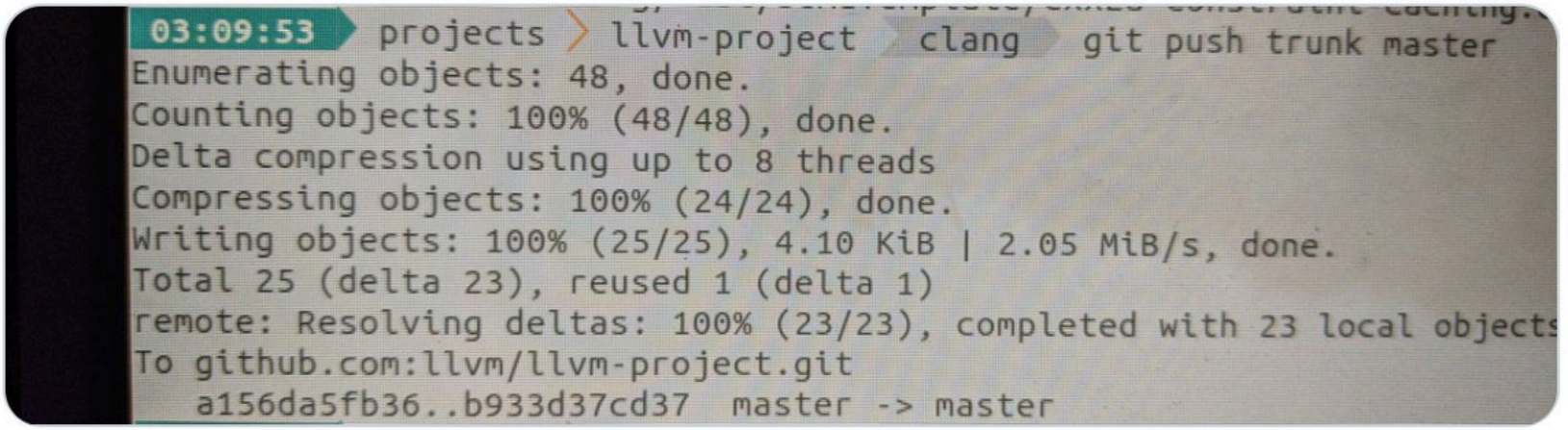
- Message
- Reddit

> *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 };
```