C++ Club Meeting Notes

Gleb Dolgich

2020-01-16

C++ `std::string_view` for better performance: An example use case

Article

Reddit

Arthur O'Dwyer: `std::string_view` is a borrow type

> *Borrow types are essentially "borrowed" references to existing objects. They lack ownership; they are short-lived; they generally can do without an assignment operator. They generally appear only in function parameter lists; because they lack ownership semantics, they generally cannot be stored in data structures or returned safely from functions.*

cppreference: `std::basic_string_view` (C++17)

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

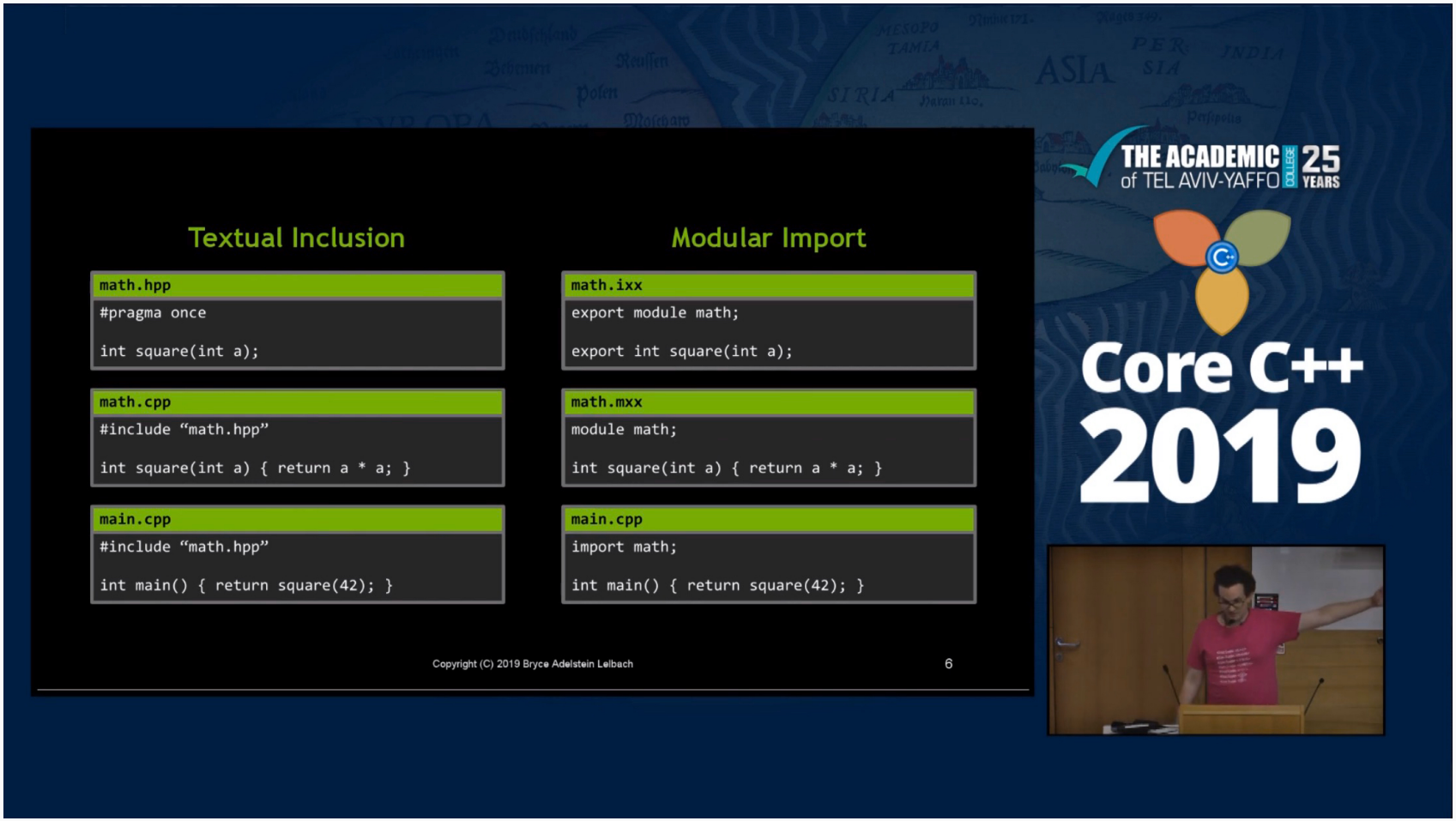
> *Argumentum is a C++17 library for writing command-line program interfaces, inspired by Python argparse*

- GitHub: C++17, MPL
- Reddit

- YouTube
- Reddit

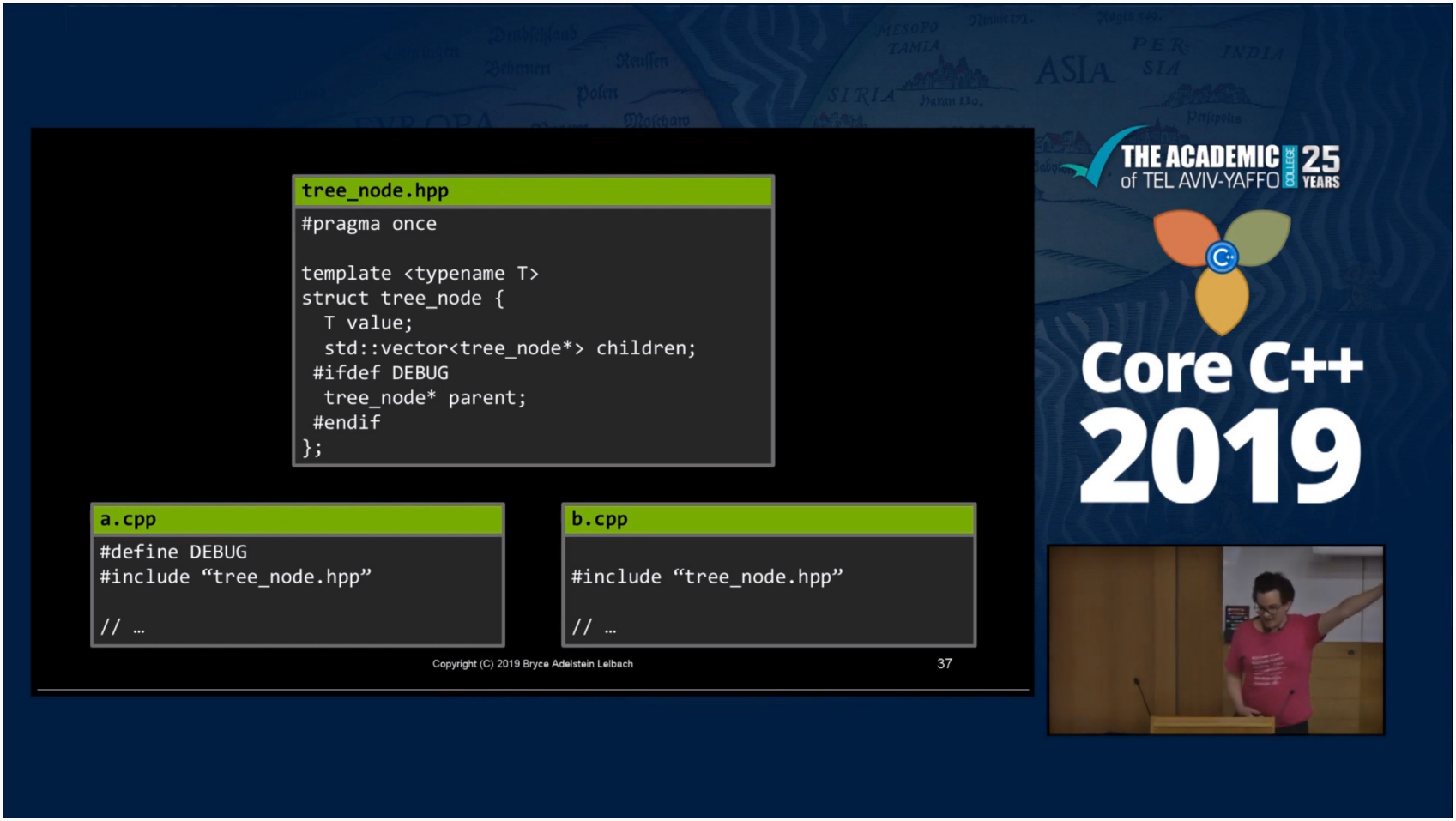
> *Modules will have a greater impact than any other feature added post-c++98.*

Modules are Coming - Bryce Adelstein Lelbach



> *"Ill-formed, no diagnostics required" (IFNDR)*

Modules are Coming – Bryce Adelstein Lelbach



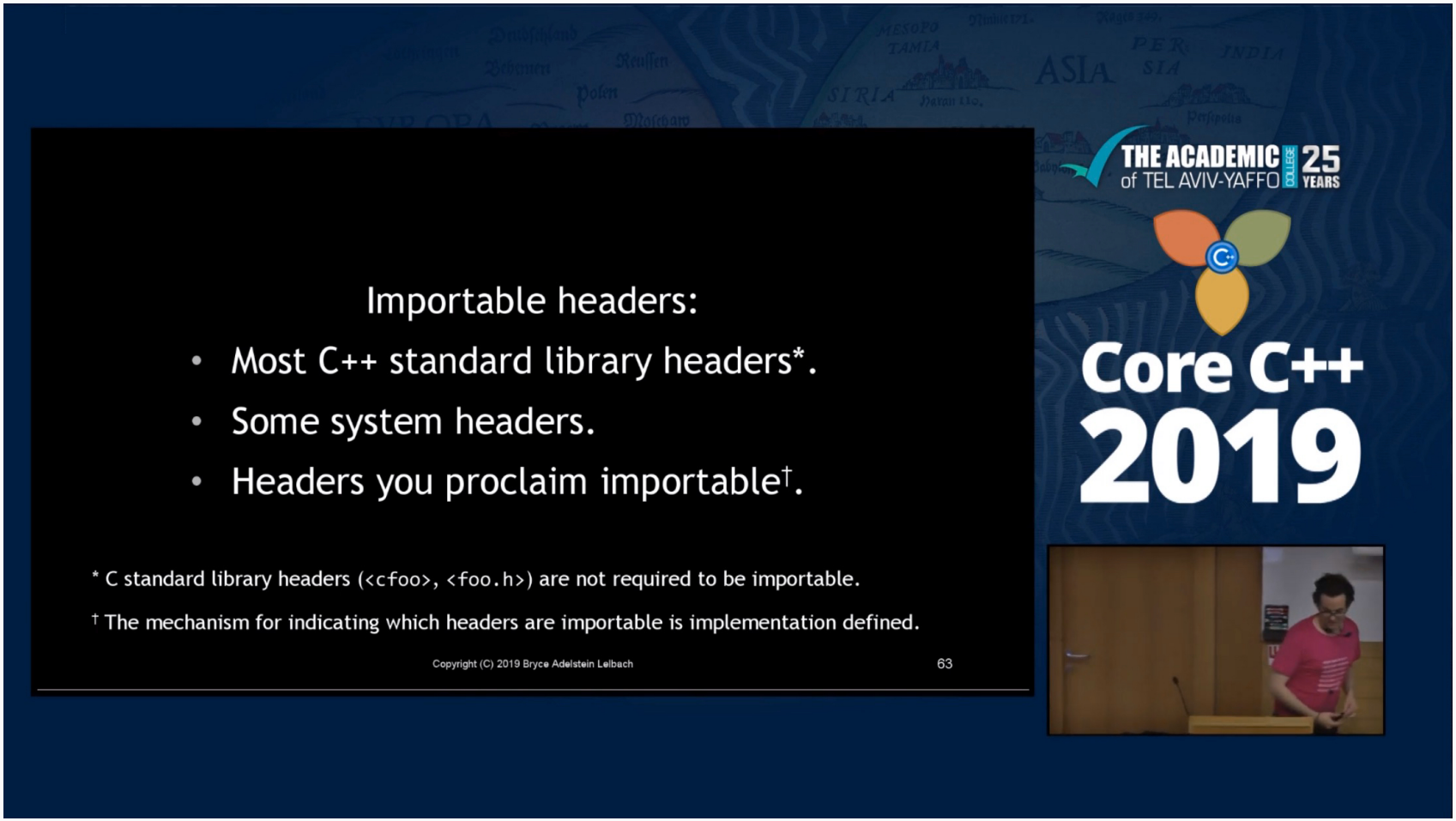
Textual inclusion

```
1 #include <foo.hpp>
2 #include "foo.hpp"
```

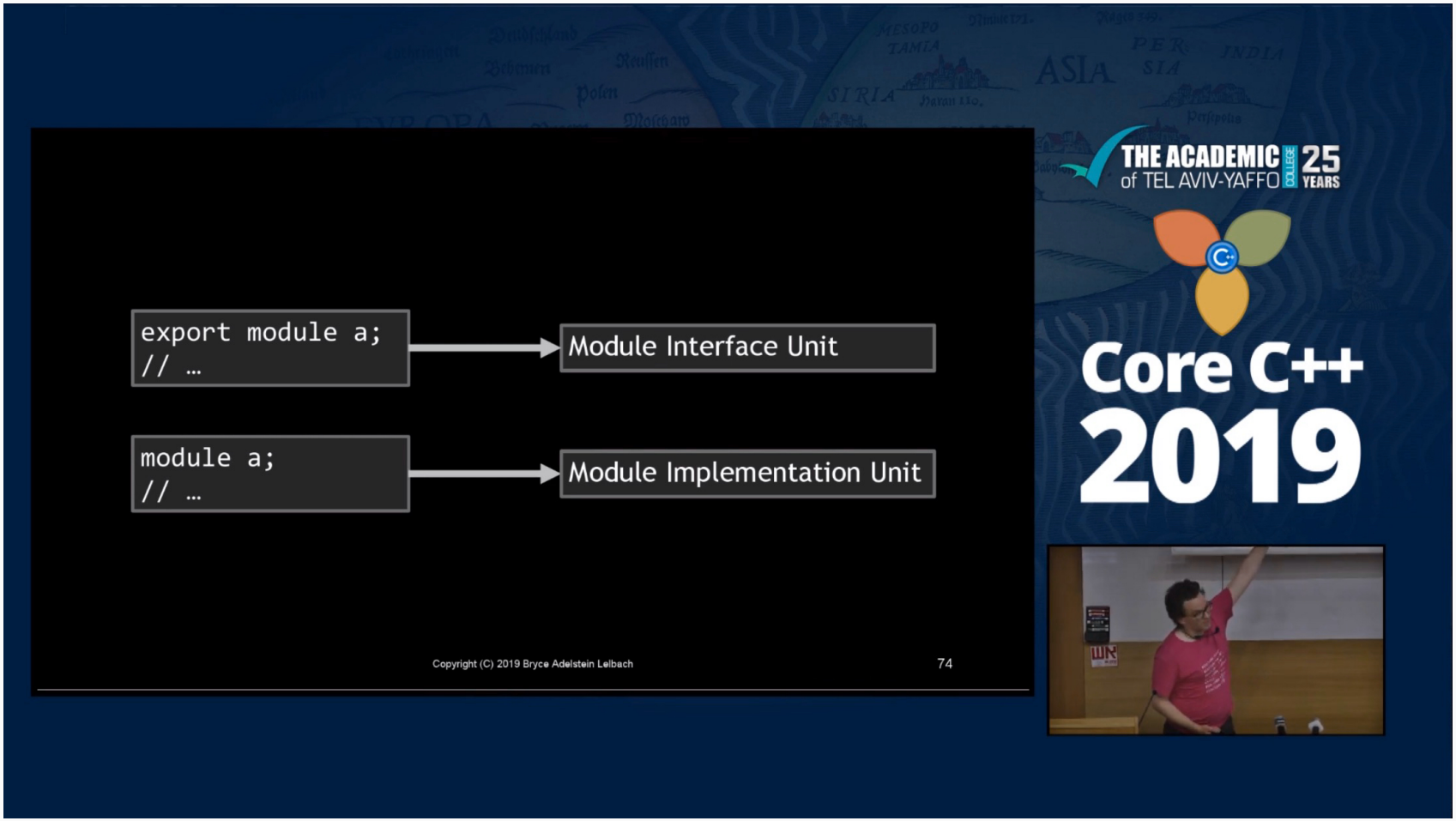
Modular import

```
1 import foo;
2 import <foo.hpp>;
3 import "foo.hpp";
```

Modules are Coming – Bryce Adelstein Lelbach



Modules are Coming – Bryce Adelstein Lelbach



Only one module declaration per translation unit:

Interface

```
1 export module a;
2 //...
3 export module b; // Error
```

Implementation

```
1 module a;
2 //...
3 module b; // Error
```

Module unit structure

```
1 export module ...;
2 import ...;
3 ...
```

```
export declaration ...
export {
    declaration ...
}
```

```
export {
    void f();
    struct A;
    int i{0};
}
```

```
export void f();
export struct A;
export int i{0};
```

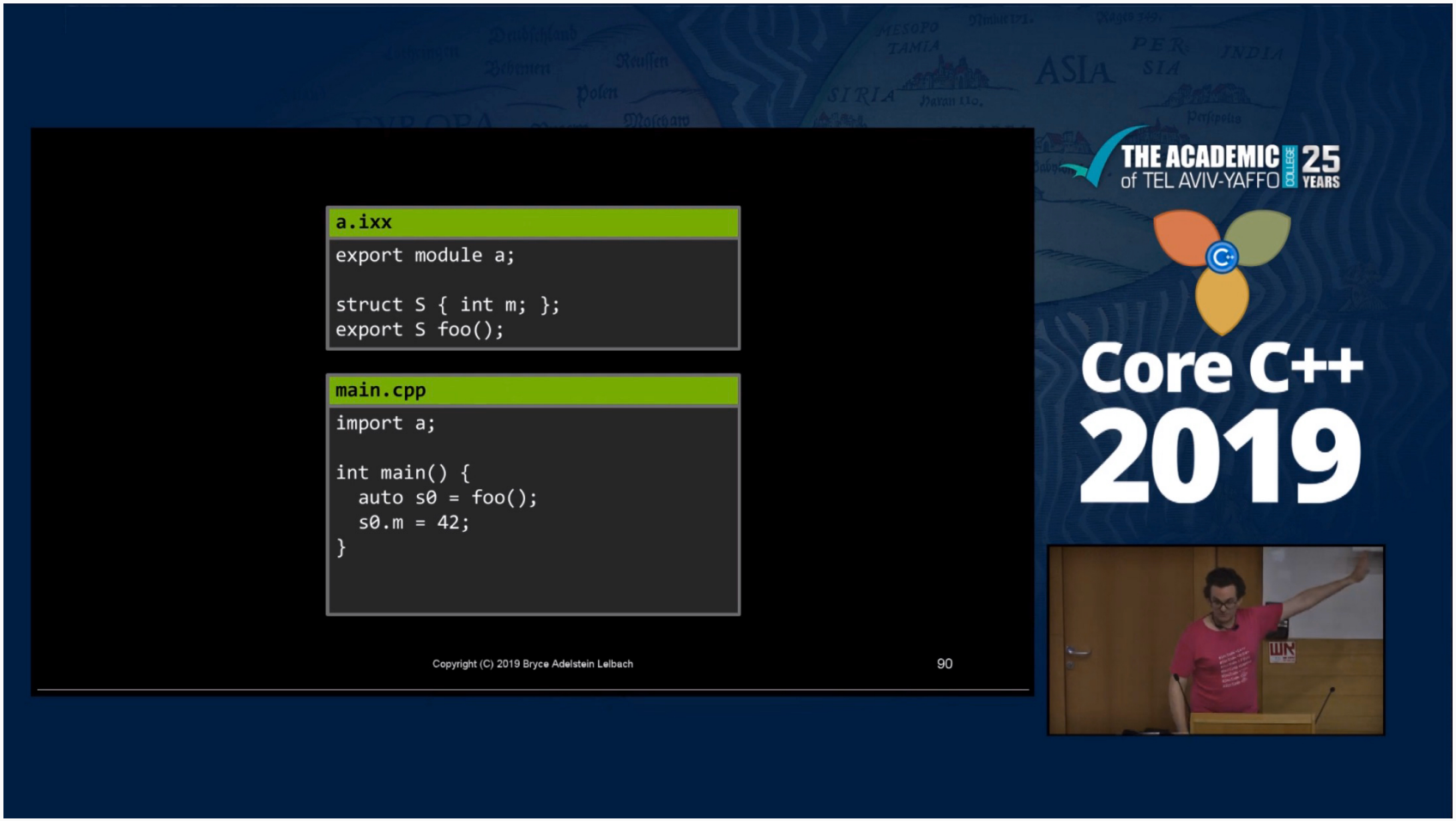
```
1 export template <typename T>
2 T square(T t) { return t*t; }
3
4 export template <typename T>
5 struct is_const : false_type {};
6
7 export template <typename T>
8 struct is_const<T const> : true_type {};
```

```
1 export namespace foo { struct A; } // foo::A exported
2
3 namespace foo { struct B; } // foo::B not exported
```

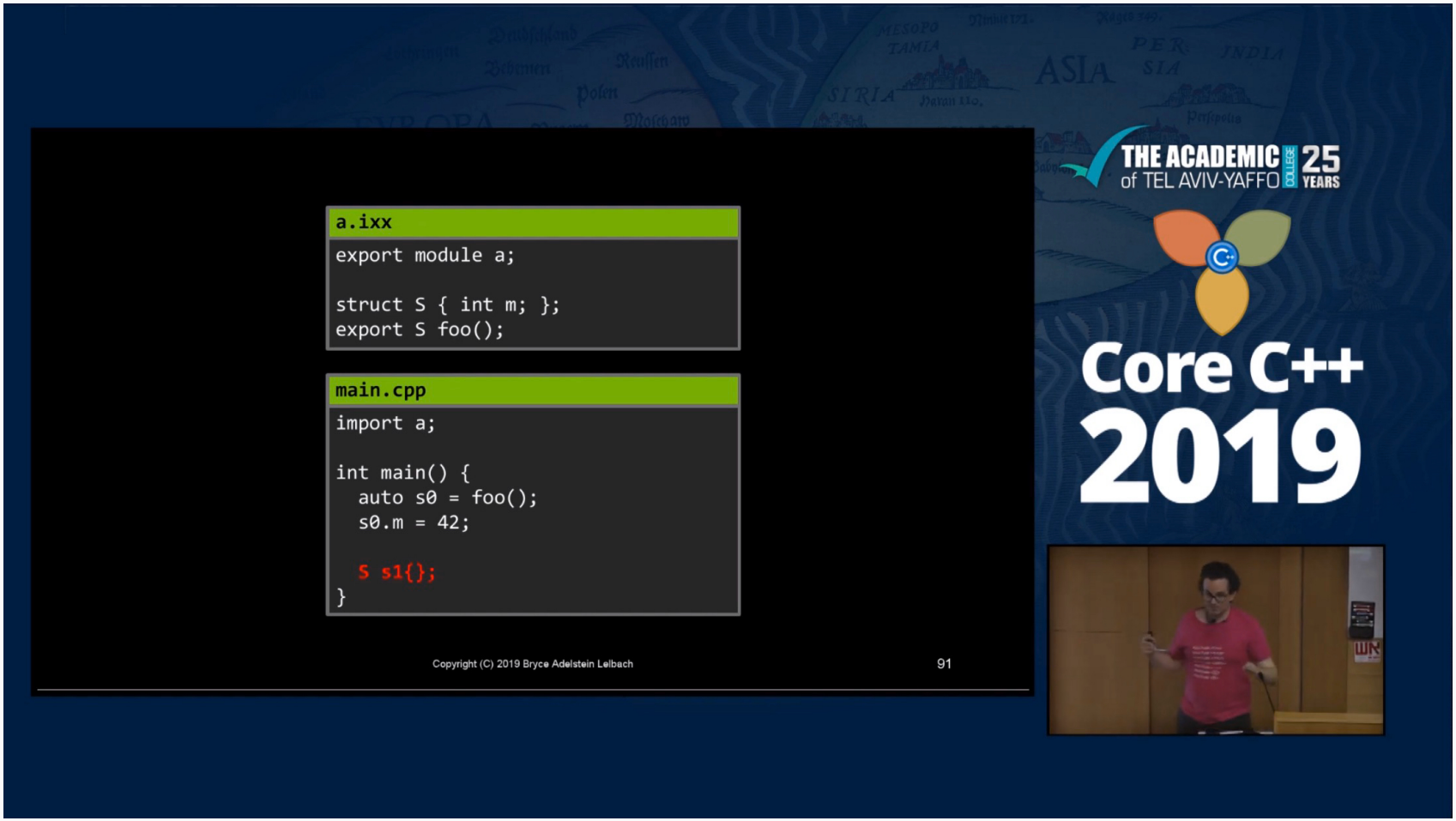
```
1 export typedef int int32_t;
2
3 export using unsigned uint32_t;
```

```
1 export import a;
```

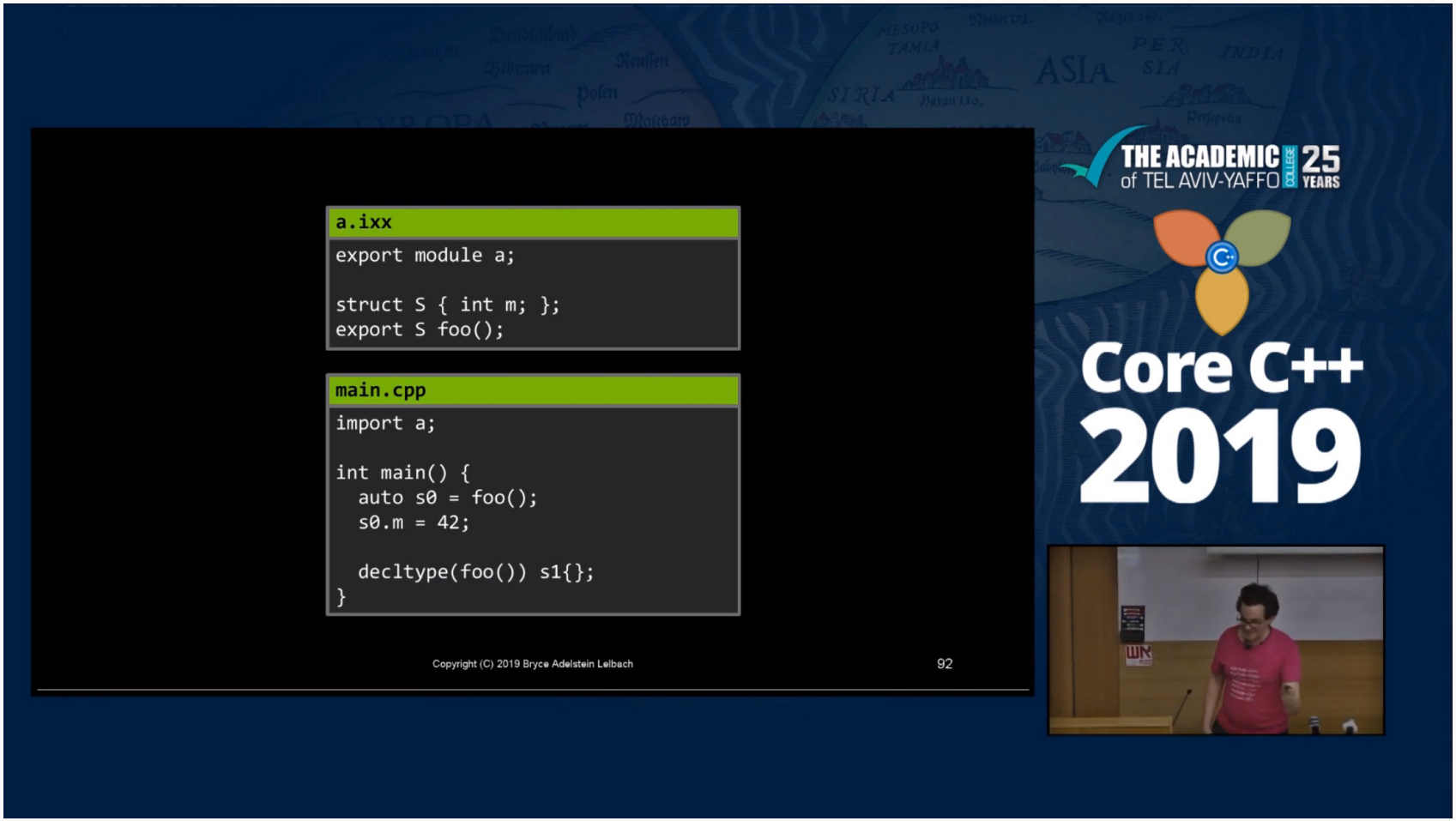
Modules are Coming - Bryce Adelstein Lelbach



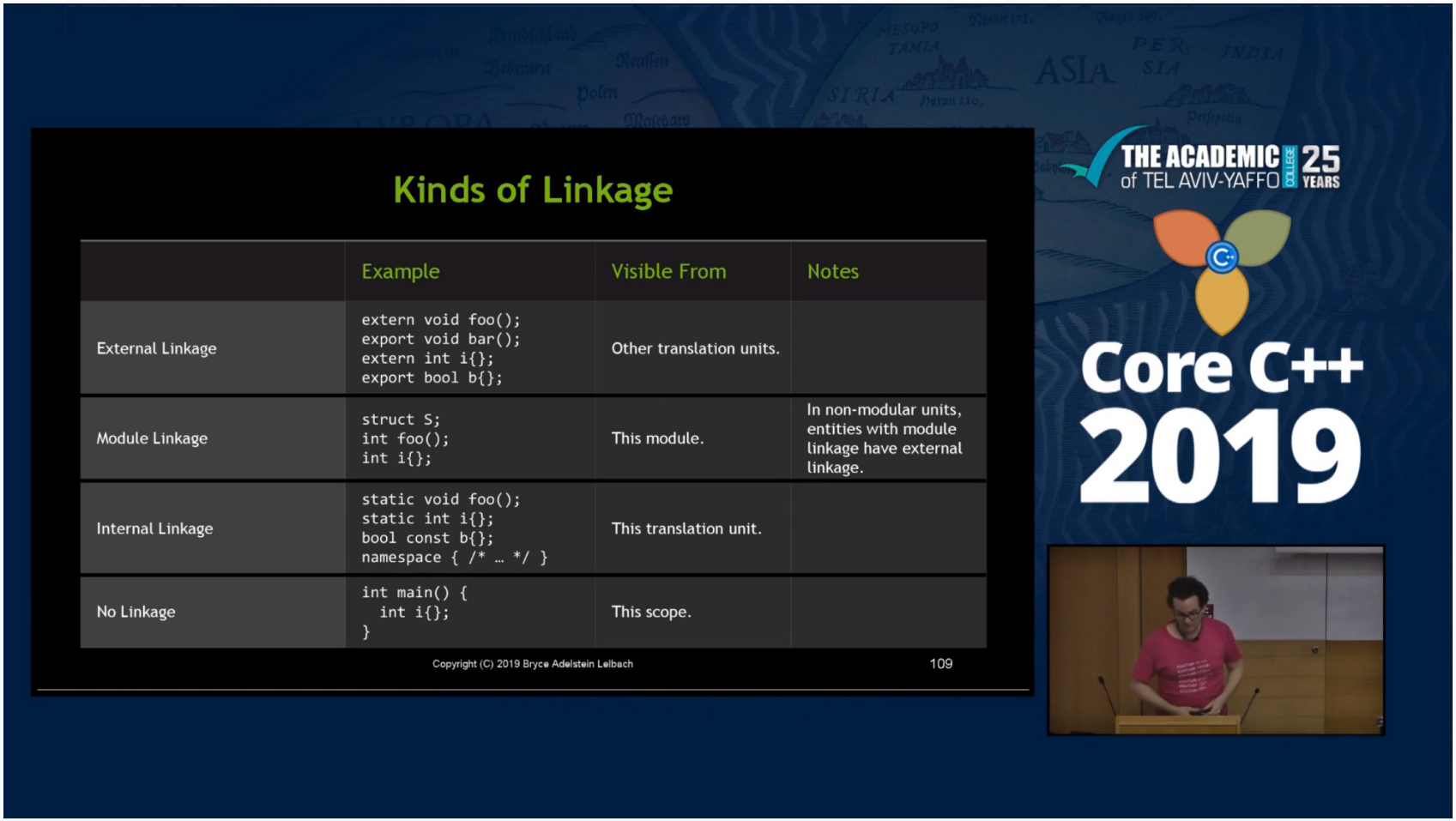
Modules are Coming – Bryce Adelstein Lelbach



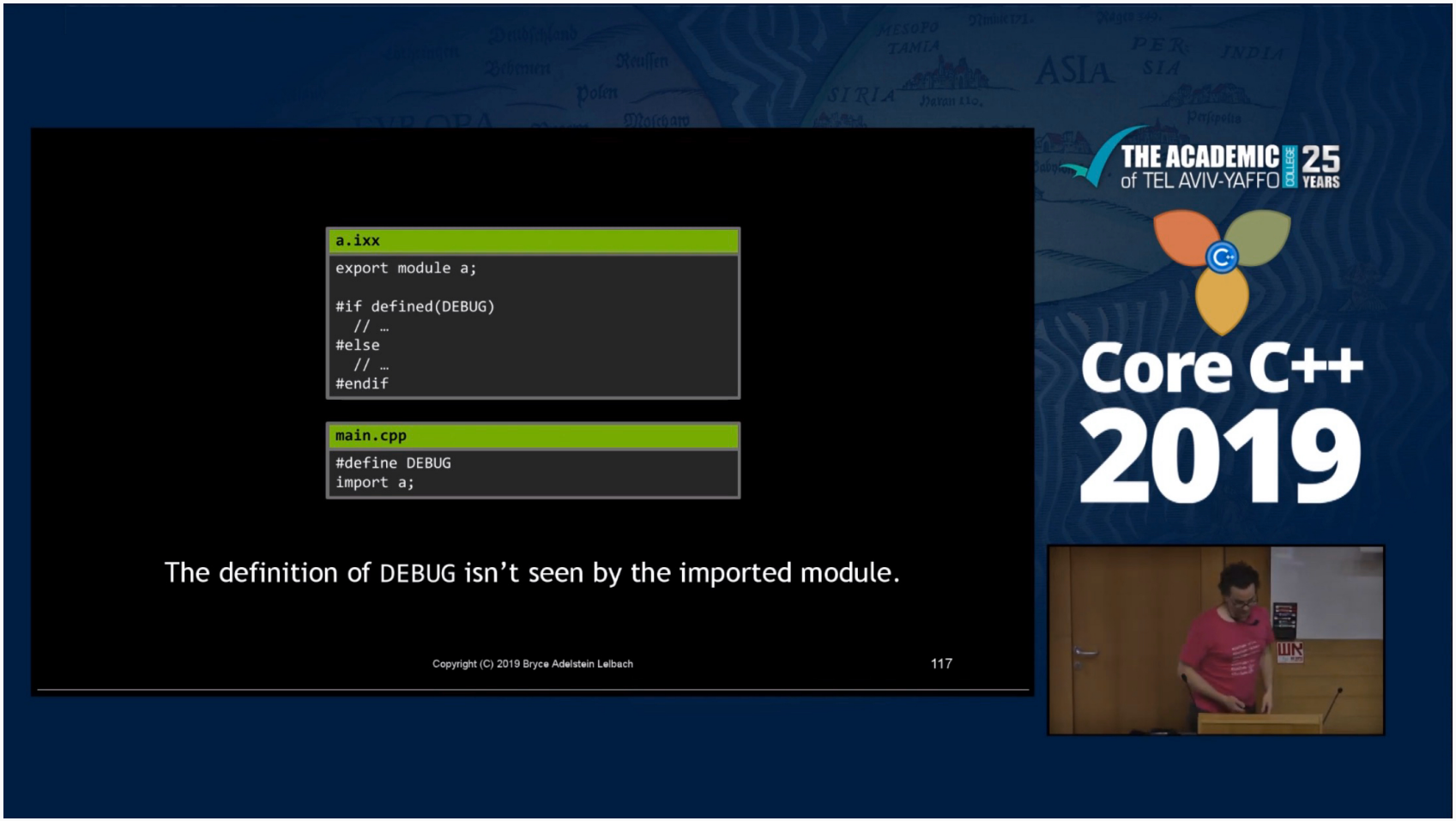
Modules are Coming – Bryce Adelstein Lelbach



- *Visible*: in scope, can be named
- *Reachable*: in scope, not necessarily namable


Kinds of Linkage
Example
Visible From
Notes
External Linkage
extern void foo();
export void bar();
extern int i{};
export bool b{};
Other translation units.
Module Linkage
struct S;
int foo();
int i{};
This module.
In non-modular units, entities with module linkage have external linkage.
Internal Linkage
static void foo();
static int i{};
bool const b{};
namespace { /* … */ }
This translation unit.
No Linkage
int main() {
int i{};
}
This scope.
Copyright (C) 2019 Bryce Adelstein Lelbach
109
THE ACADEMIC of TEL AVIV-YAFFO COLLEGE 25 YEARS
Core C++ 2019

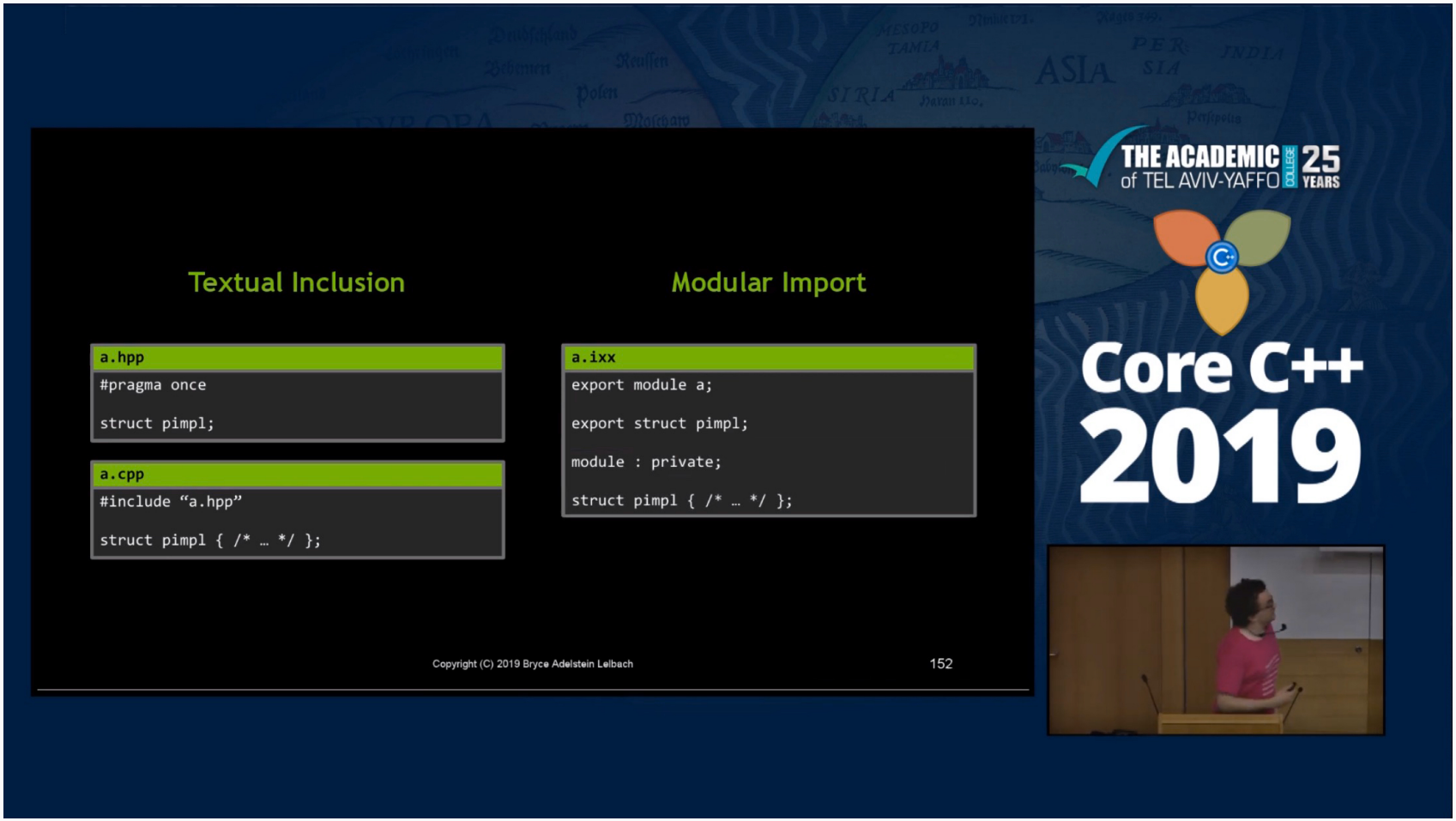
Modules are Coming – Bryce Adelstein Lelbach



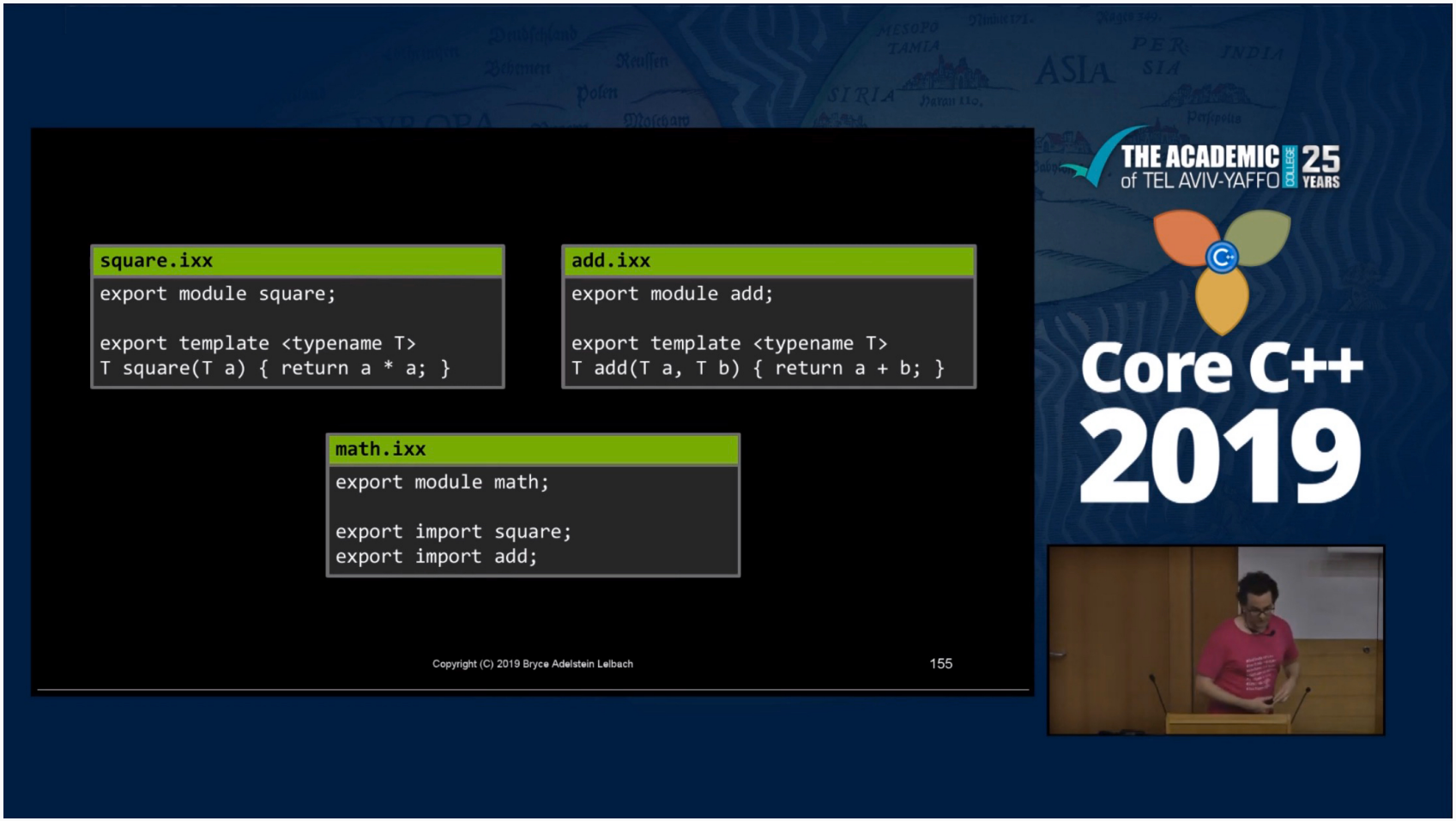
Module unit structire

```
1 module;
2 #pp-directive ...;
3 export module ...;
4 import ...;
5 ...
6 module : private;
7 ...
```

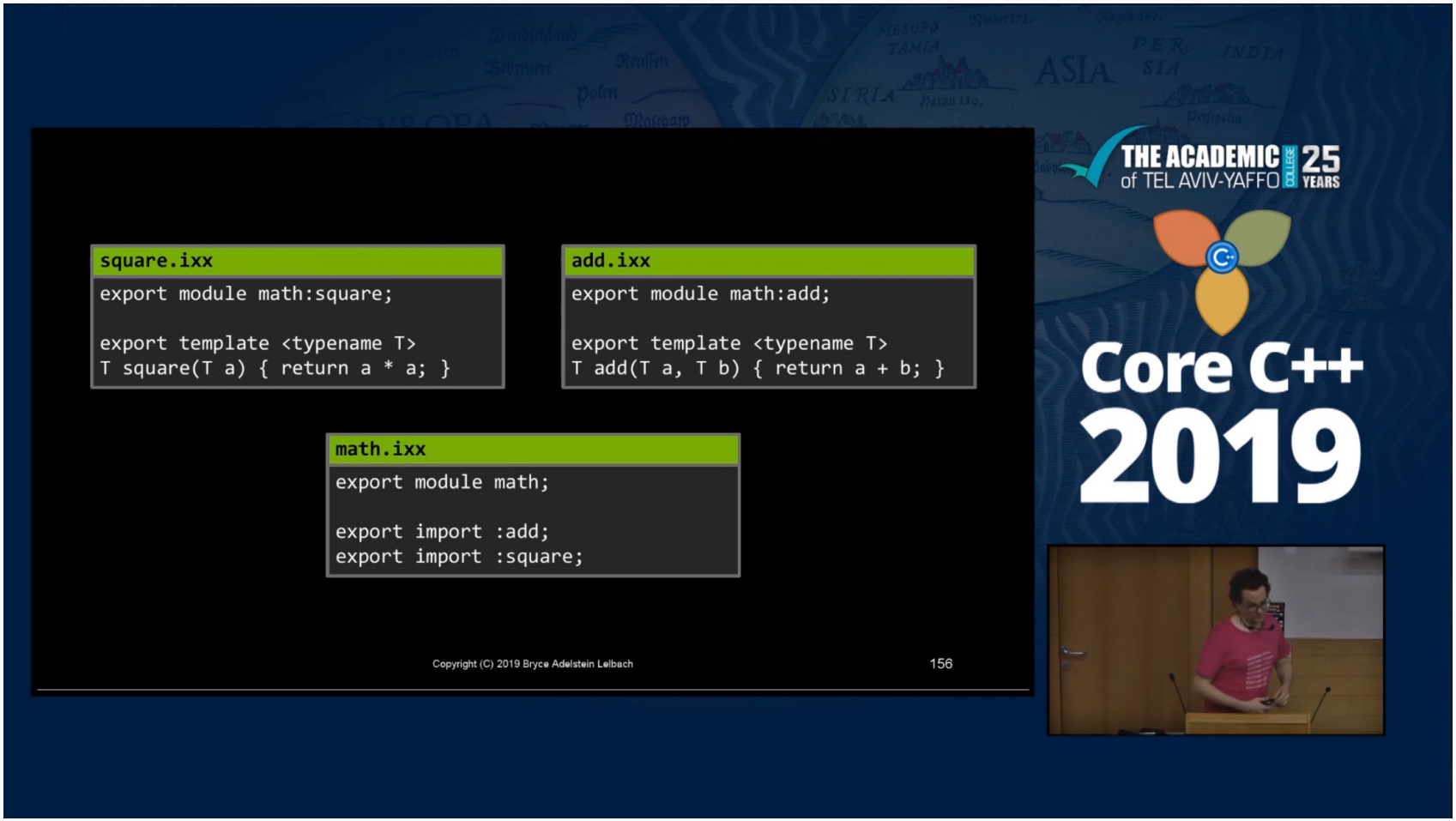
Modules are Coming – Bryce Adelstein Lelbach



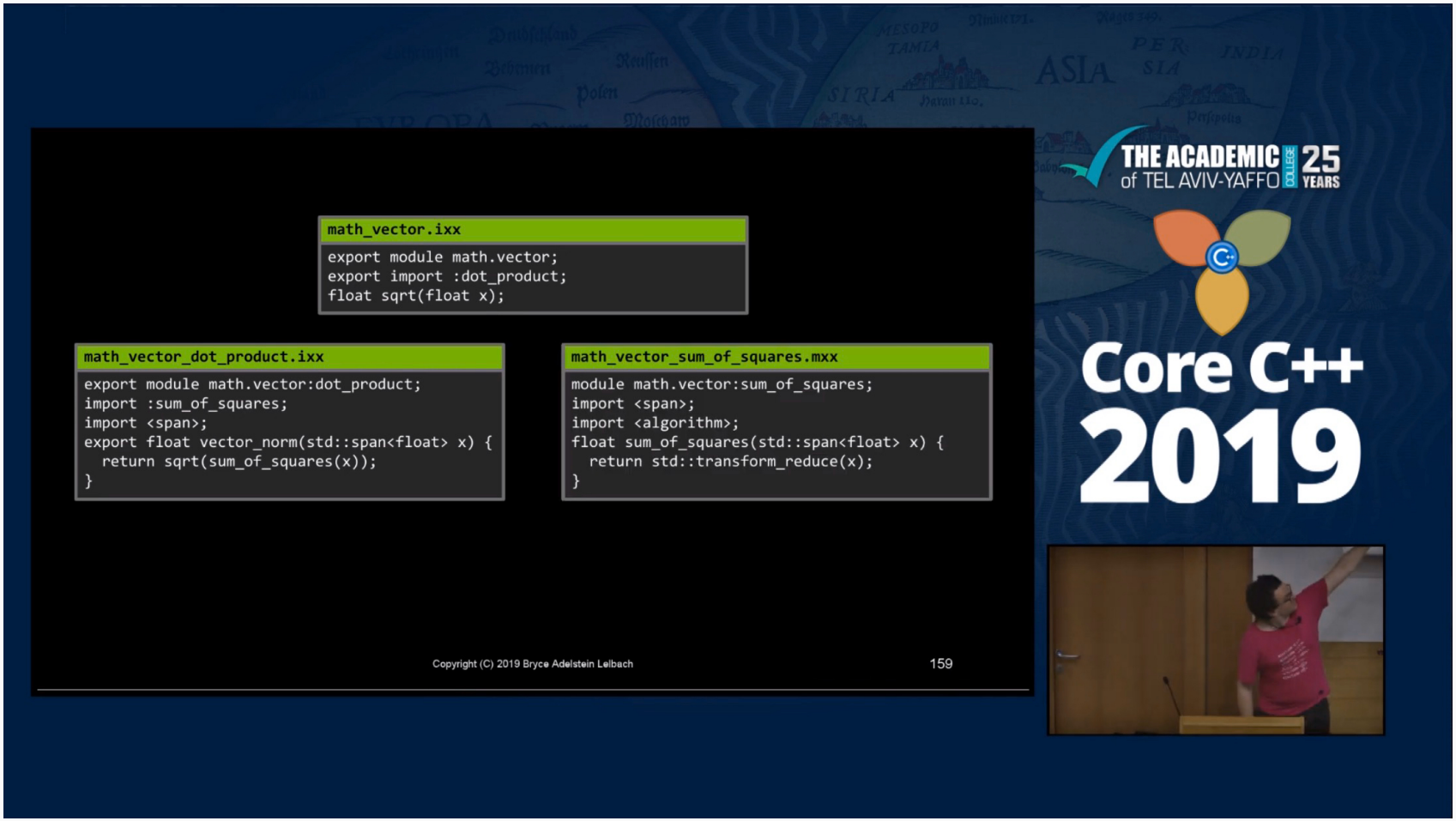
Modules are Coming – Bryce Adelstein Lelbach



Modules are Coming – Bryce Adelstein Lelbach



Modules are Coming - Bryce Adelstein Lelbach



Modules are Coming – Bryce Adelstein Lelbach

Kinds of Translation Units

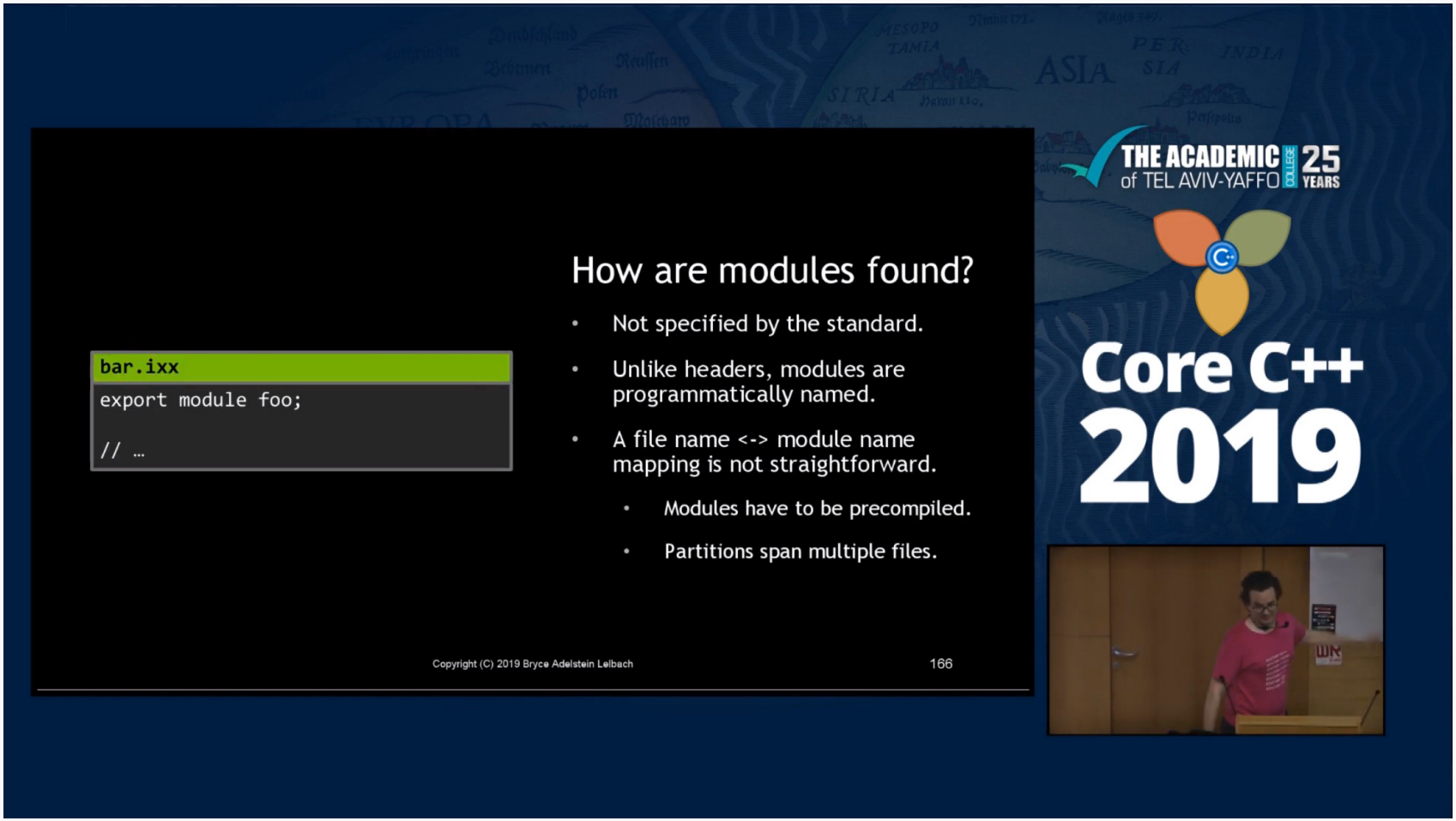
	Example	Extension	Artifact	Notes
Non-Modular Unit	`#include "…"` …	`.cpp`	`.o`	
Header Unit	`// Created by:` `import <…>;`	`.hpp`	`.cmi` `.o` (optional)	
Module Interface Unit	`export module …;` …	`.ixx`	`.cmi` `.o` (optional)	Exactly one per module.
Module Implementation Unit	`module …;` …	`.mxx`	`.o`	At most one per module.
Module Partition Interface Unit	`export module …:…;` …	`.ixx`	`.cmi` `.o` (optional)	
Module Partition Implementation Unit	`module …:…;` …	`.mxx`	`.o`	

161

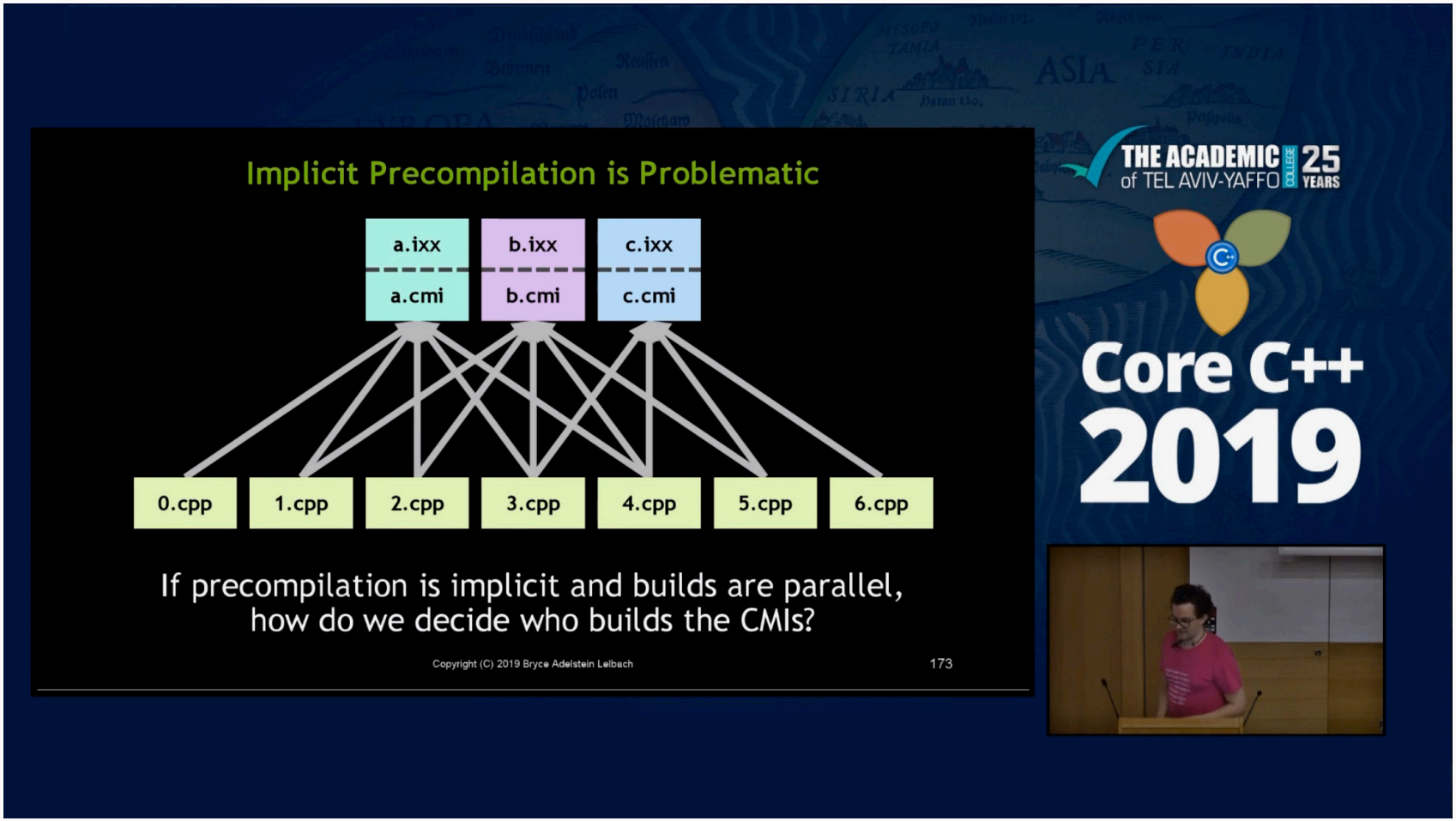
THE ACADEMIC of TEL AVIV-YAFFO COLLEGE 25 YEARS

Core C++ 2019

Modules are Coming – Bryce Adelstein Lelbach



Modules are Coming – Bryce Adelstein Lelbach



- Tools can no longer rely on simple lookup mechanism (include directories and header file names) to understand C++ projects.
- Dependency scanning now requires a C++ parser, not just a C preprocessor.