NVIDIA

MODULES ARE COMING

Bryce Adelstein Lelbach

@blelbach

CUDA C++ Core Libraries Lead

ISO C++ Library Evolution Incubator Chair, ISO C++ Tooling Study Group Chair

#include <C++>

Copyright (C) 2019 Bryce Adelstein Lelbach

NVIDIA

MODULES ARE COMING

Bryce Adelstein Lelbach

@blelbach

CUDA C++ Core Libraries Lead

ISO C++ Library Evolution Incubator Chair, ISO C++ Tooling Study Group Chair

Modules are:

- A new compilation model for C++.
- A new way to organize C++ projects.

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

math.hpp

```
#pragma once

int square(int a);
```

math.cpp

```
#include “math.hpp”

int square(int a) { return a * a; }
```

main.cpp

```
#include “math.hpp”

int main() { return square(42); }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

math.hpp

```
#pragma once

int square(int a);
```

math.cpp

```
#include “math.hpp”

int square(int a) { return a * a; }
```

main.cpp

```
#include “math.hpp”

int main() { return square(42); }
```

Modular Import

math.cppm

```
export module math;

export int square(int a);
```

math.cpp

```
module math;

int square(int a) { return a * a; }
```

main.cpp

```
import math;

int main() { return square(42); }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

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

Copyright (C) 2019 Bryce Adelstein Lelbach

C++11's <thread> changes how…

… C++ code is written

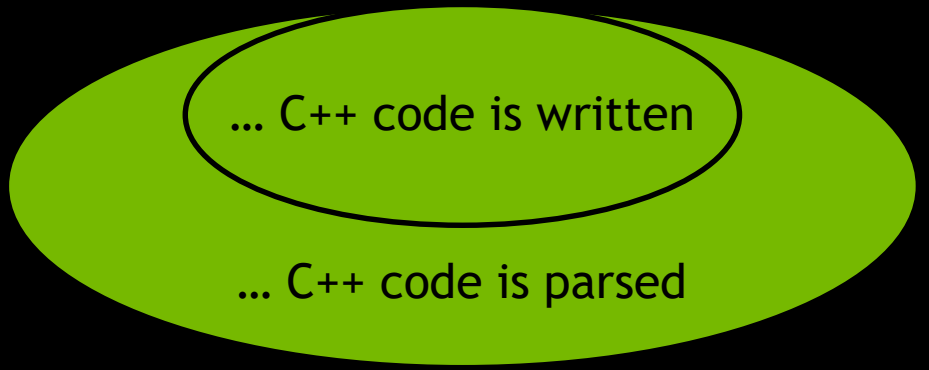
Copyright (C) 2019 Bryce Adelstein Lelbach

C++11's lambdas change how…

… C++ code is written

Copyright (C) 2019 Bryce Adelstein Lelbach

C++11's lambdas change how…



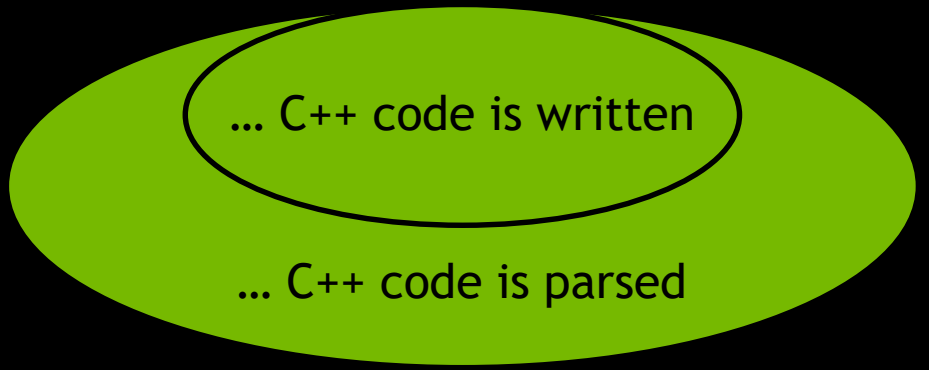
Copyright (C) 2019 Bryce Adelstein Lelbach

C++20's modules change how…

… C++ code is written

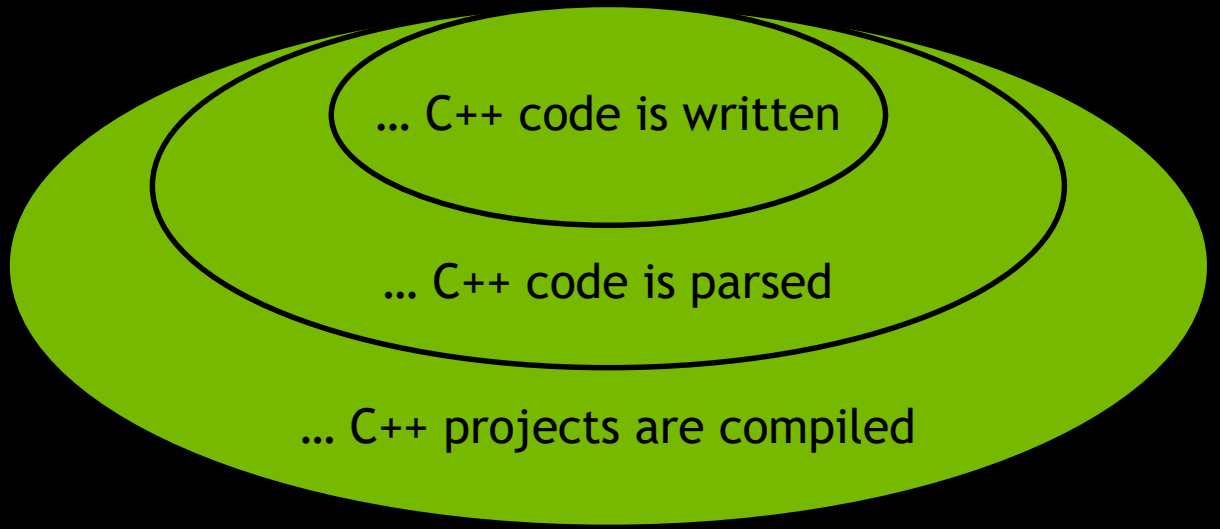
Copyright (C) 2019 Bryce Adelstein Lelbach

C++20's modules change how…



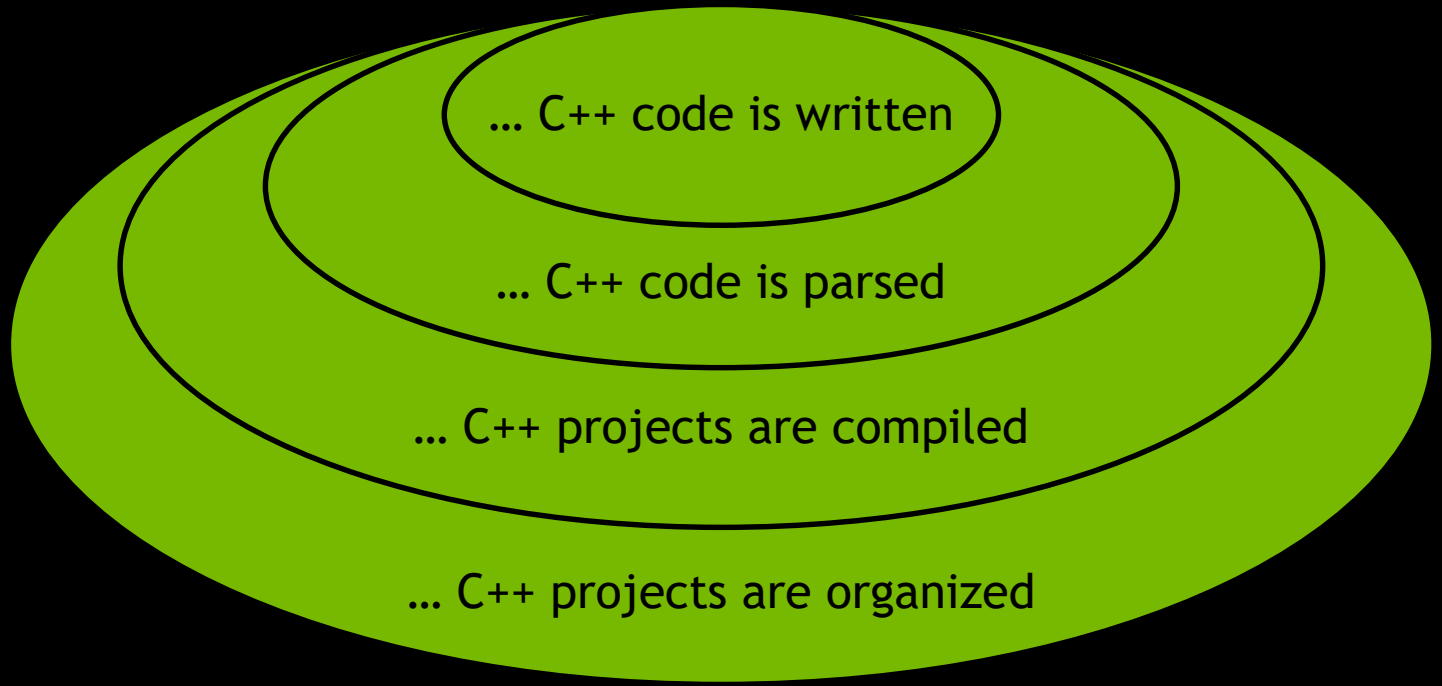
Copyright (C) 2019 Bryce Adelstein Lelbach

C++20's modules change how…



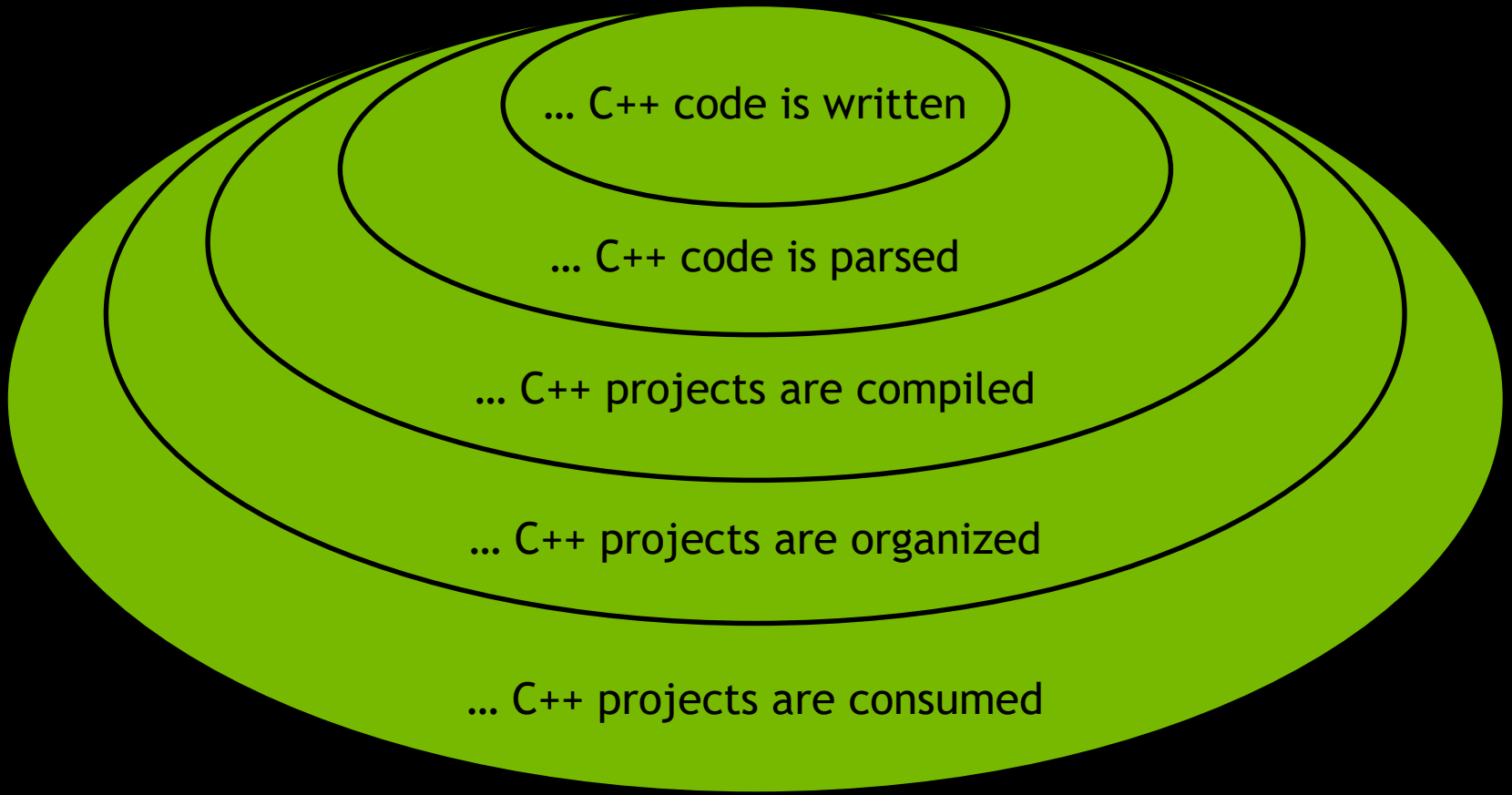
Copyright (C) 2019 Bryce Adelstein Lelbach

C++20's modules change how…



Copyright (C) 2019 Bryce Adelstein Lelbach

C++20's modules change how…



Copyright (C) 2019 Bryce Adelstein Lelbach

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

Copyright (C) 2019 Bryce Adelstein Lelbach

Today's Compilation Model

Copyright (C) 2019 Bryce Adelstein Lelbach

What is C++’s compilation model today?

How do we organize C++ projects today?

Copyright (C) 2019 Bryce Adelstein Lelbach

“The text of the program is kept in units called ***source files*** in this International Standard.”

[lex.separate] p1 s1

Copyright (C) 2019 Bryce Adelstein Lelbach

"A source file together with all the headers and source files included via the preprocessing directive `#include`, less any source lines skipped by any of the conditional inclusion preprocessing directives, is called a ***translation unit***."

[lex.separate] p1 s2

Copyright (C) 2019 Bryce Adelstein Lelbach

Kinds of Translation Units

	Example	Extension	Artifact	Notes
Non-Modular Unit	`#include "…"` …	`.cpp`	`.o`	

Copyright (C) 2019 Bryce Adelstein Lelbach

"A ***program*** consists of one or more *translation units* linked together."

[basic.link] p1 s1

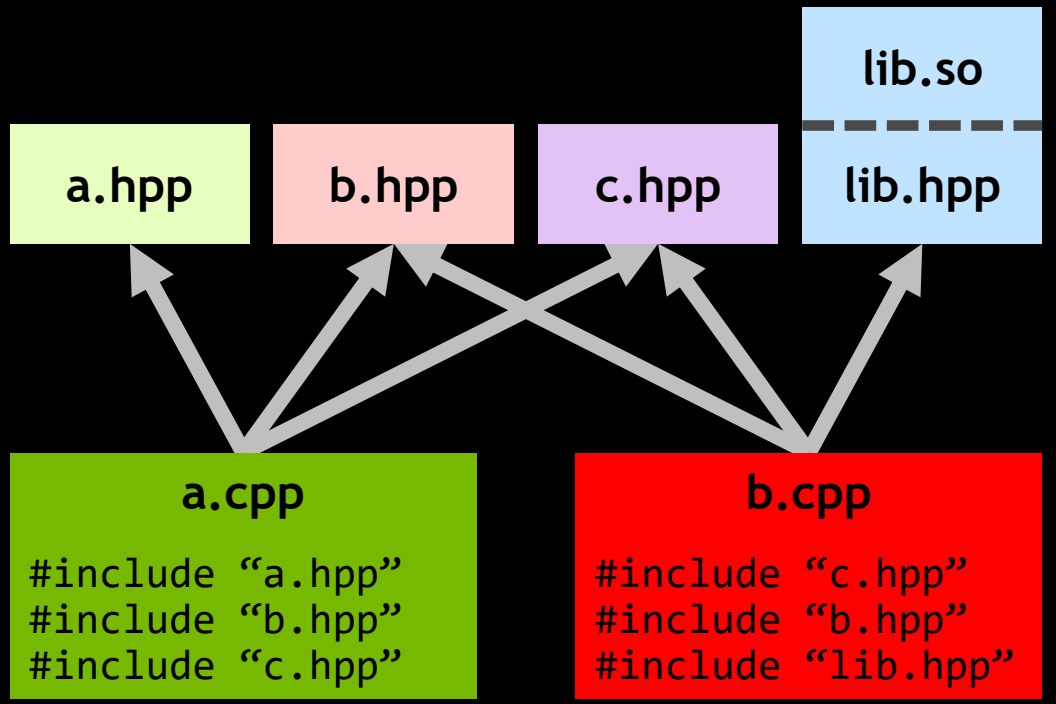
Copyright (C) 2019 Bryce Adelstein Lelbach

“Previously translated *translation units* and instantiation units can be preserved individually or in libraries.”

[lex.separate] p2 s1

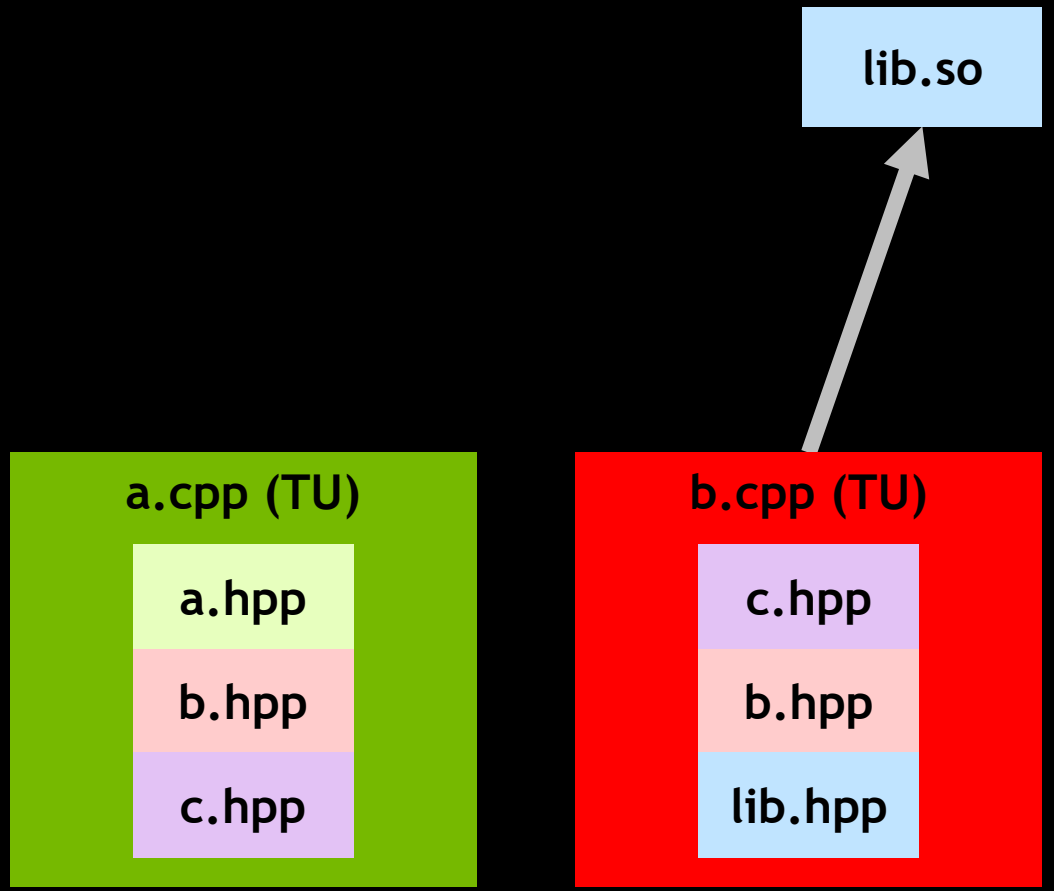
Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion: Preprocess



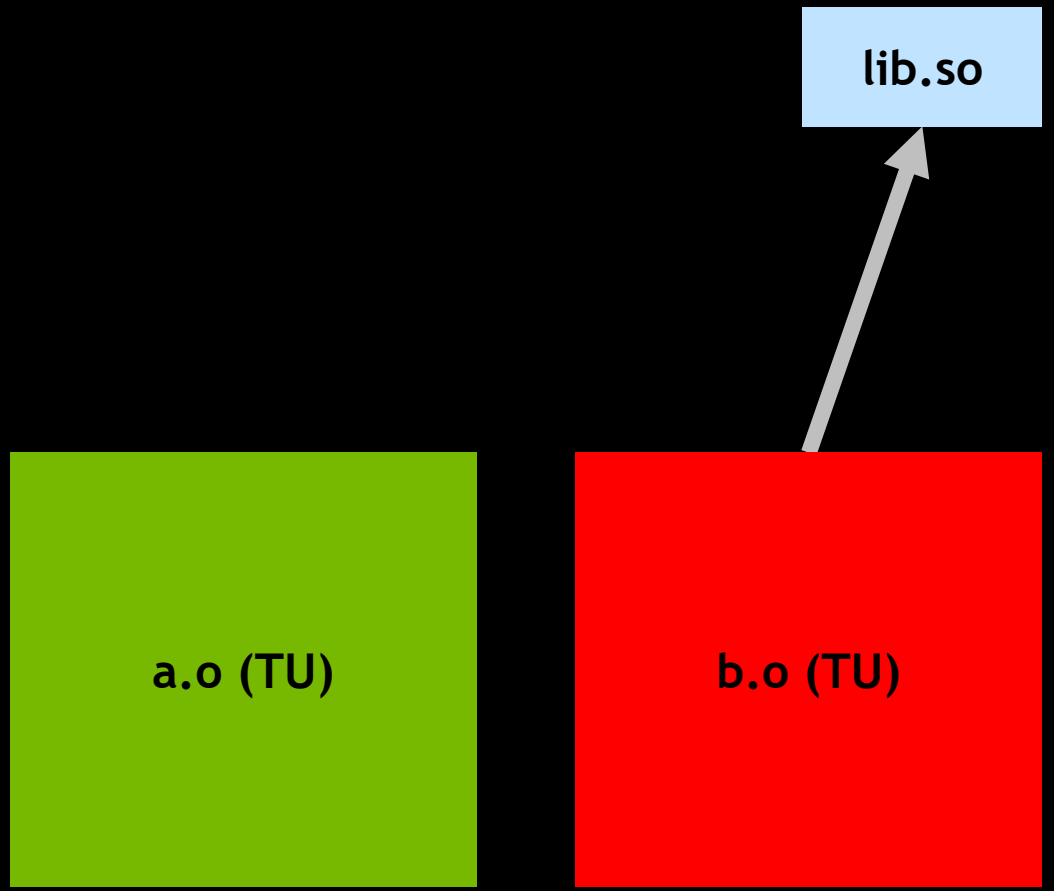
Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion: Compile



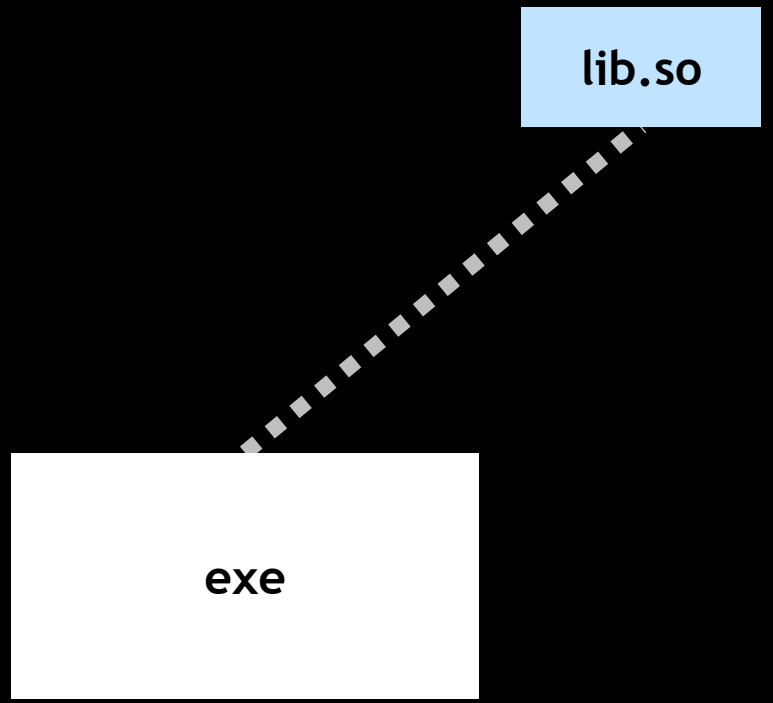
Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion: Link



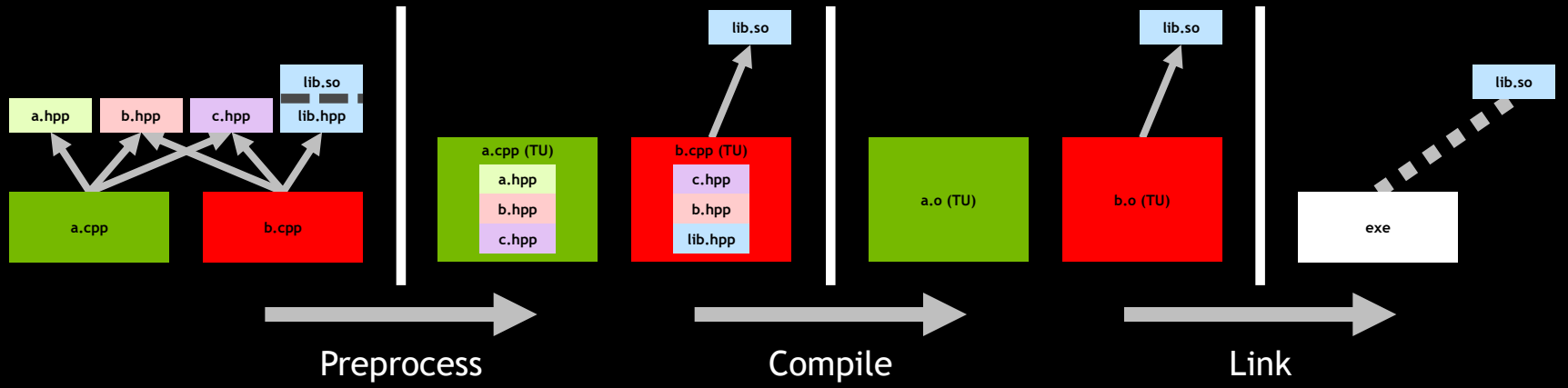
Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion



Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion



Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

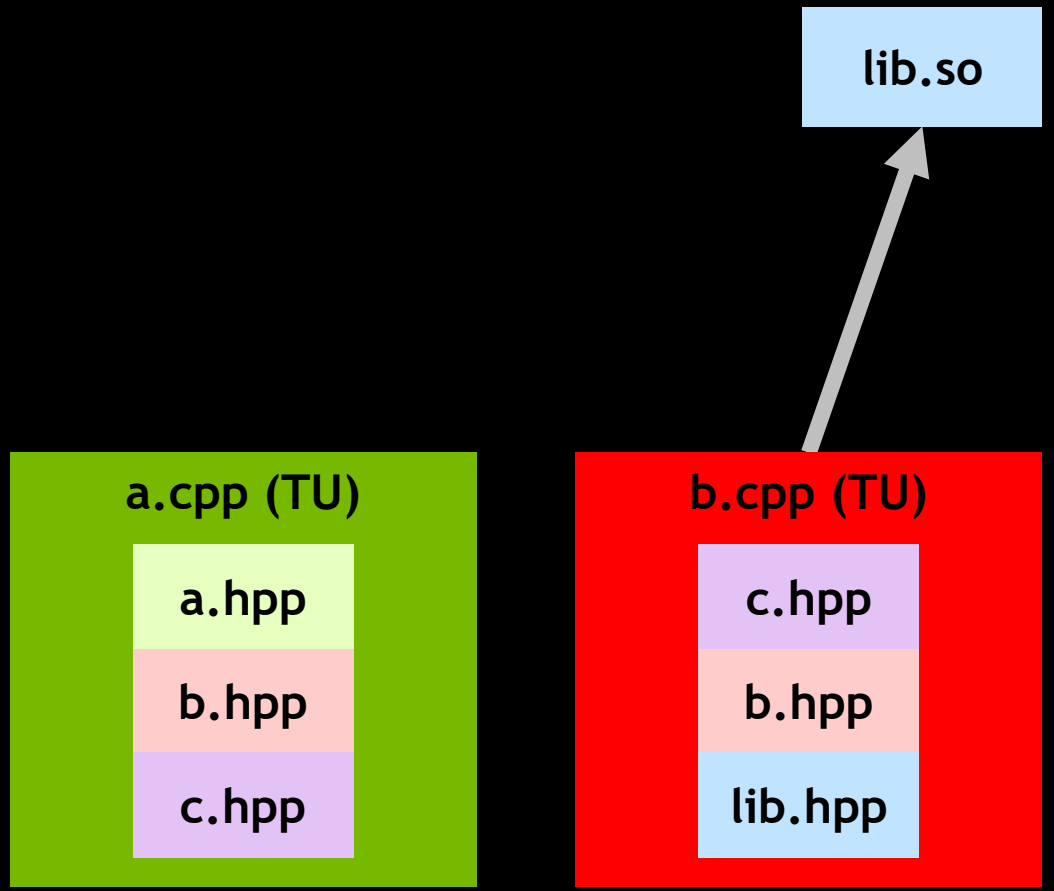
Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

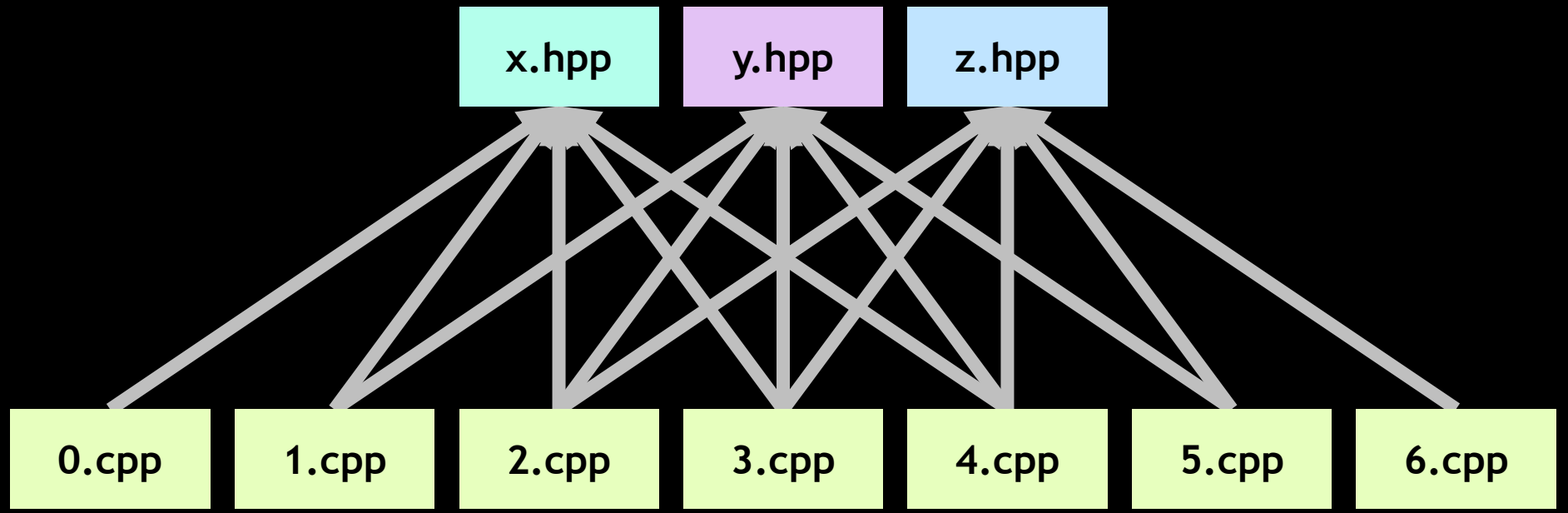
Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion: Compile



Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion



Pro: Embarrassingly parallel.

Con: x.hpp, y.hpp, and z.hpp are compiled 7 times.

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

“There can be more than one definition of [an entity] in a program provided that no prior definition is necessarily reachable at the point where a definition appears, and provided the definitions satisfy the following requirements …”

[basic.def.odr] p12

Copyright (C) 2019 Bryce Adelstein Lelbach

Ill formed, no diagnostic required

(IFNDR)

Copyright (C) 2019 Bryce Adelstein Lelbach

tree_node.hpp

```
#pragma once

template <typename T>
struct tree_node {
  T value;
  std::vector<tree_node*> children;
 #ifdef DEBUG
  tree_node* parent;
 #endif
};
```

a.cpp

```
#define DEBUG
#include “tree_node.hpp”

// …
```

b.cpp

```
#include “tree_node.hpp”

// …
```

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

namespace c {
 struct A {
  private:
   template <typename T>
   void impl();
 };

 namespace detail::unsupported {
  template <typename T>
  void __please_dont_use();
 }
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once
#include “b.hpp”

struct Y;
struct X { Y* y; };
```

b.hpp

```
#pragma once
#include “a.hpp”

struct X;
struct Y { X* x; };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

struct S { /* … */ };
```

b.hpp

```
#pragma once

void foo(S s);
```

c.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

struct S { /* … */ };
```

b.hpp

```
#pragma once

void foo(S s);
```

c.cpp

```
#include “a.hpp”
#include “b.hpp”
```

d.cpp

```
#include “b.hpp”
#include “a.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

Using Modules

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

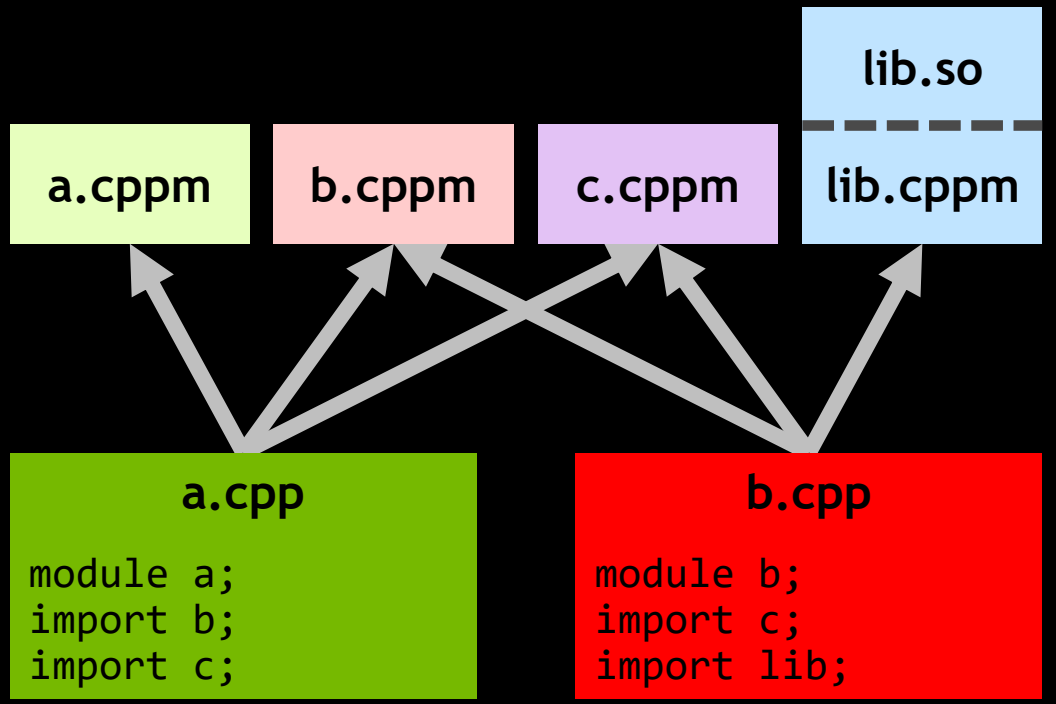
```
#include “foo.hpp”
```

Modular Import

```
import foo;
```

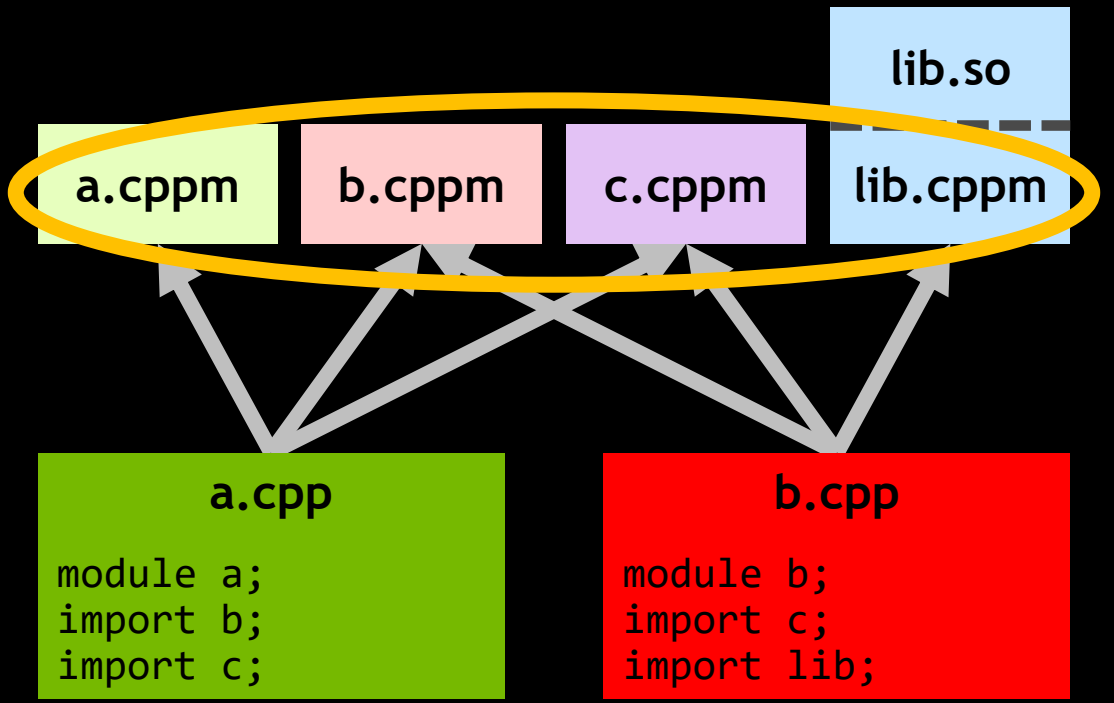
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import



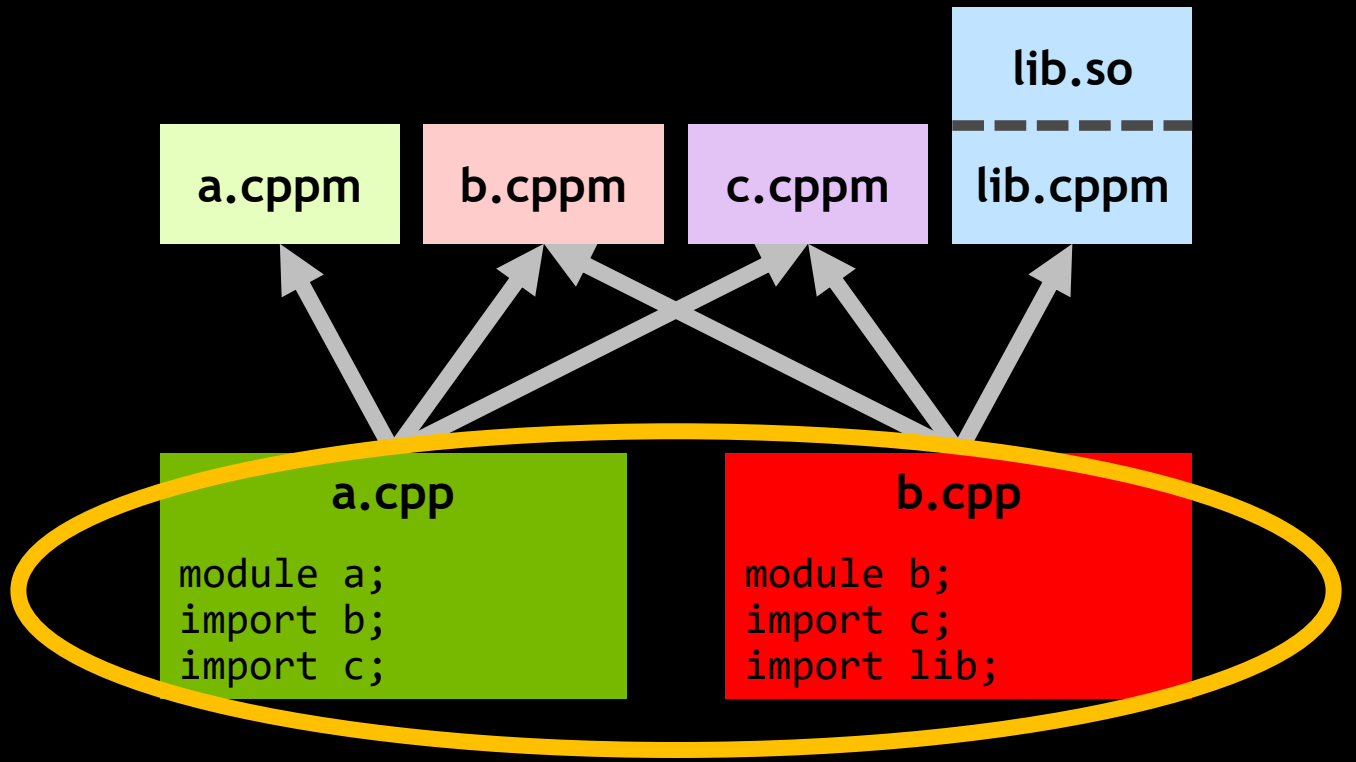
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import



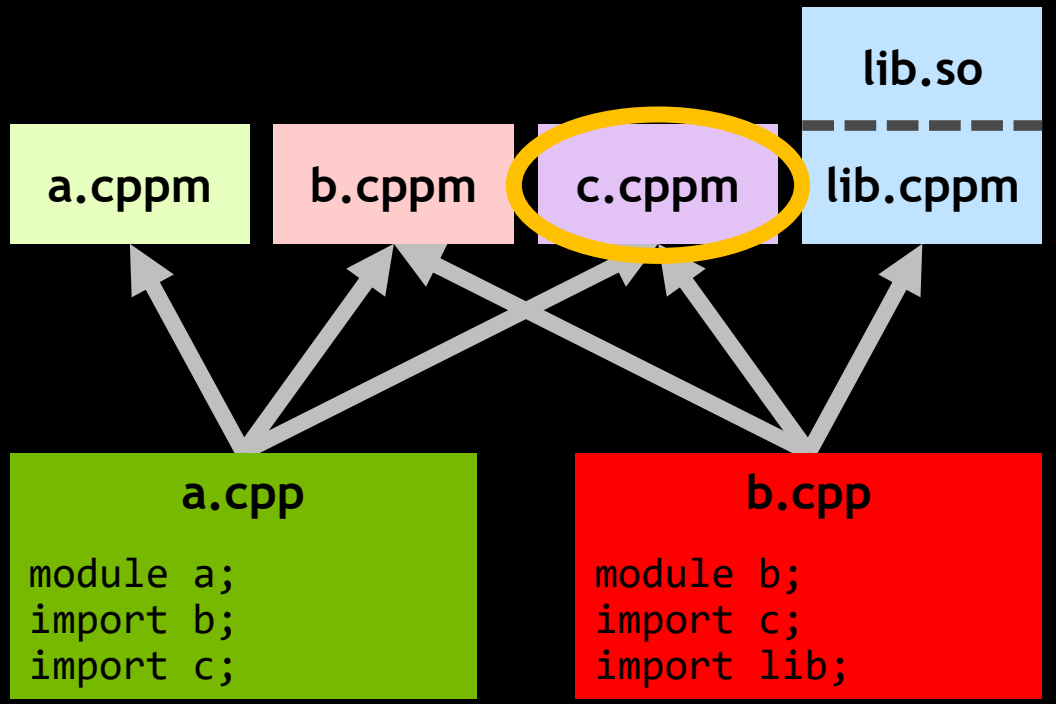
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import



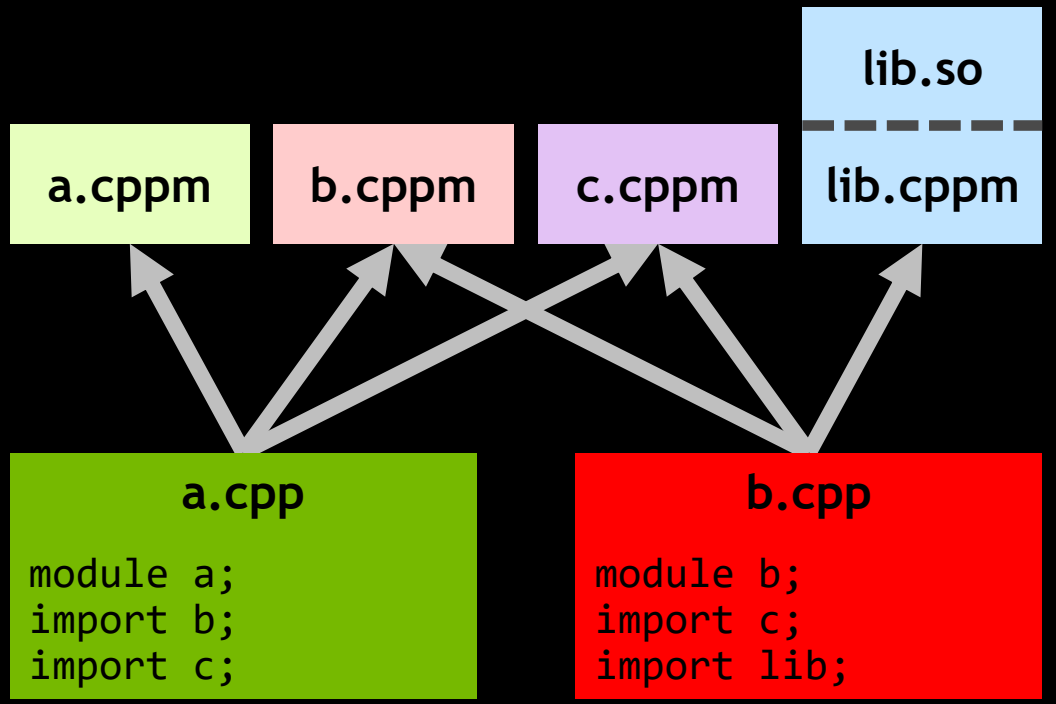
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import



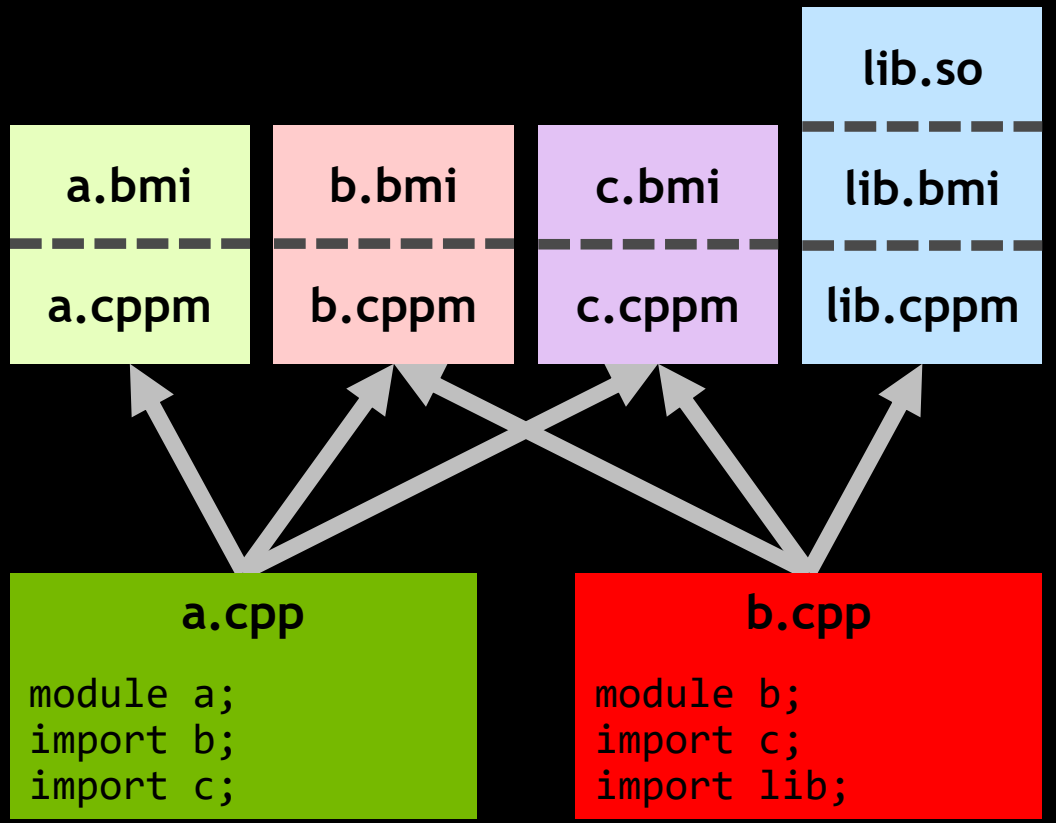
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import: Precompile



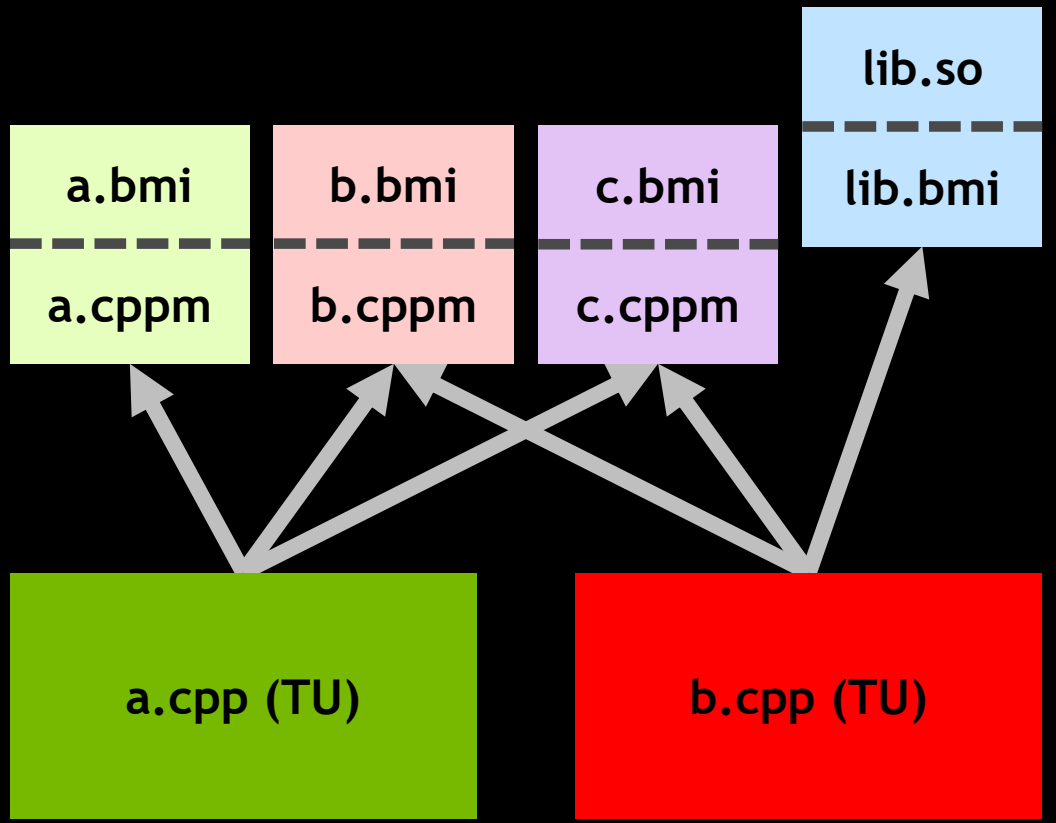
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import: Preprocess



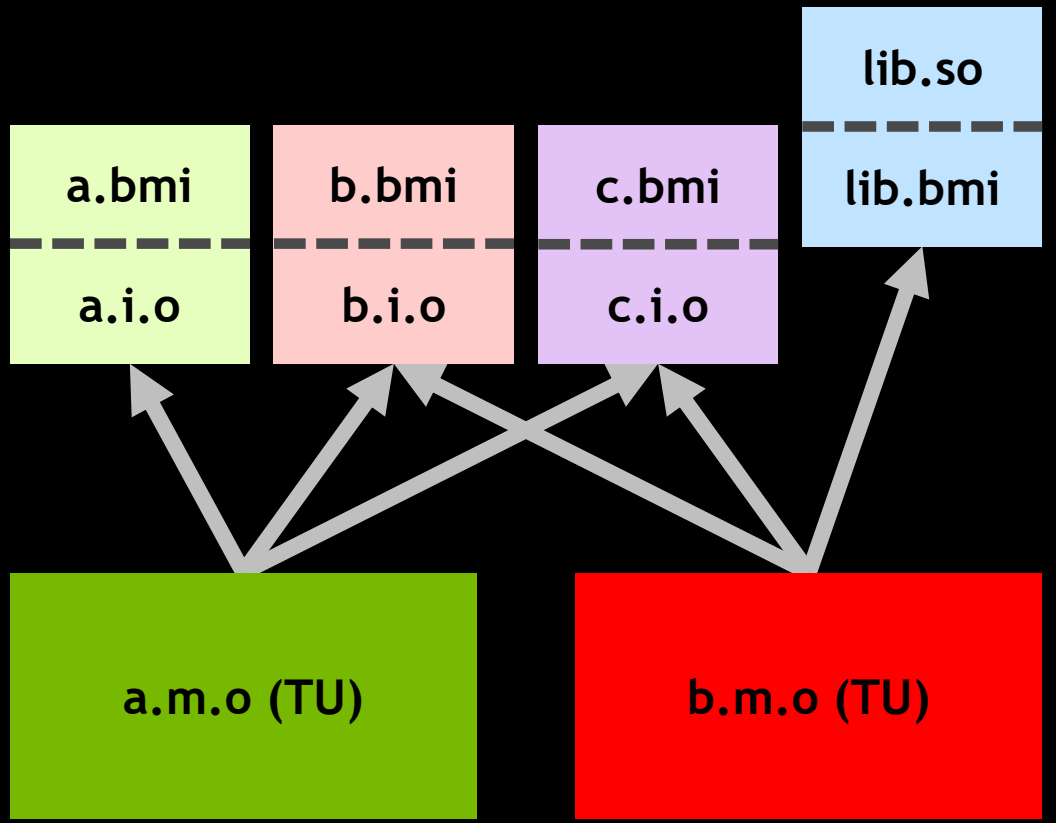
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import: Compile



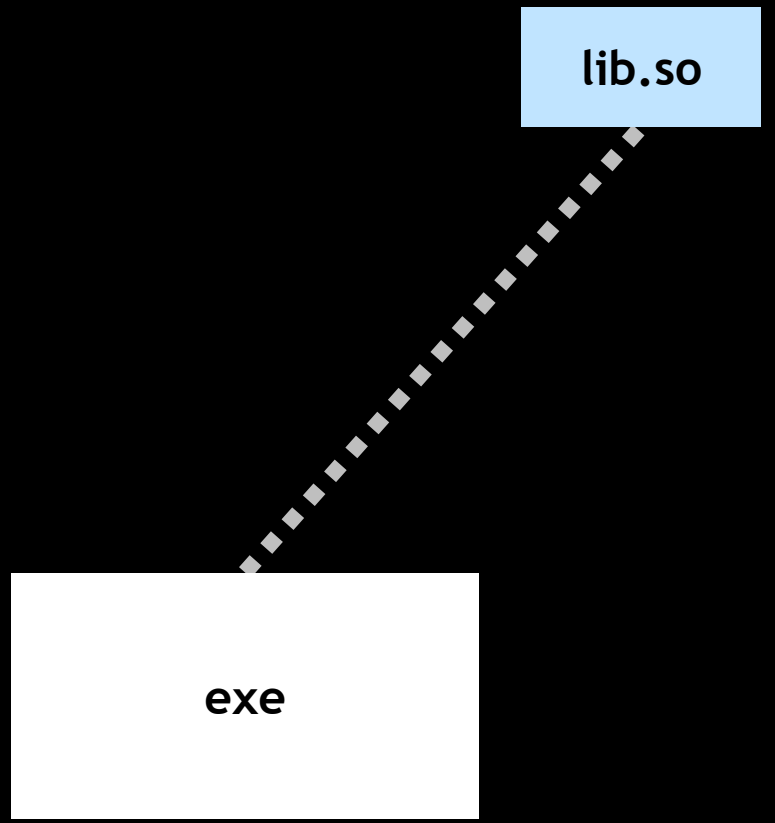
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import: Link



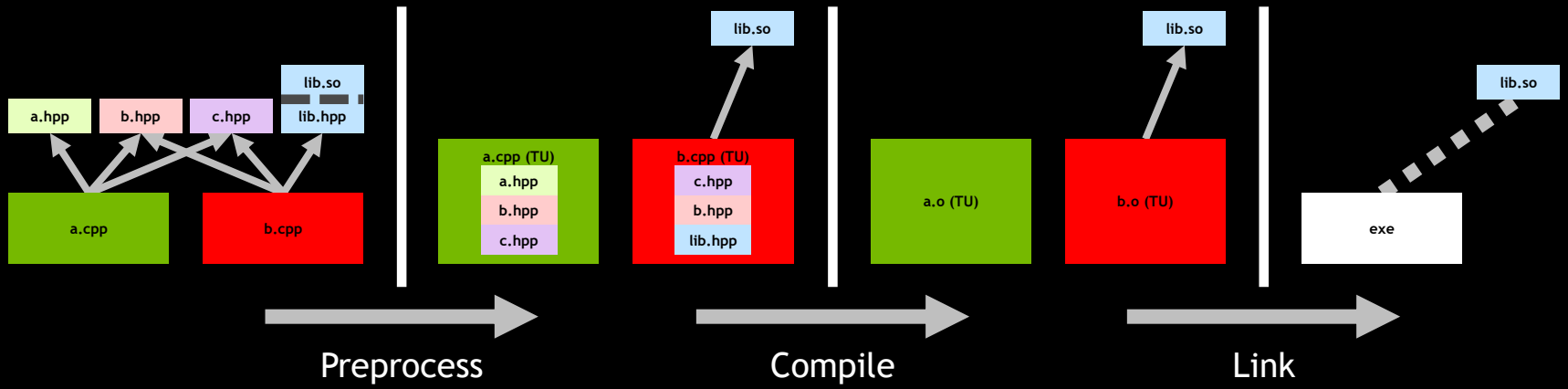
Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import

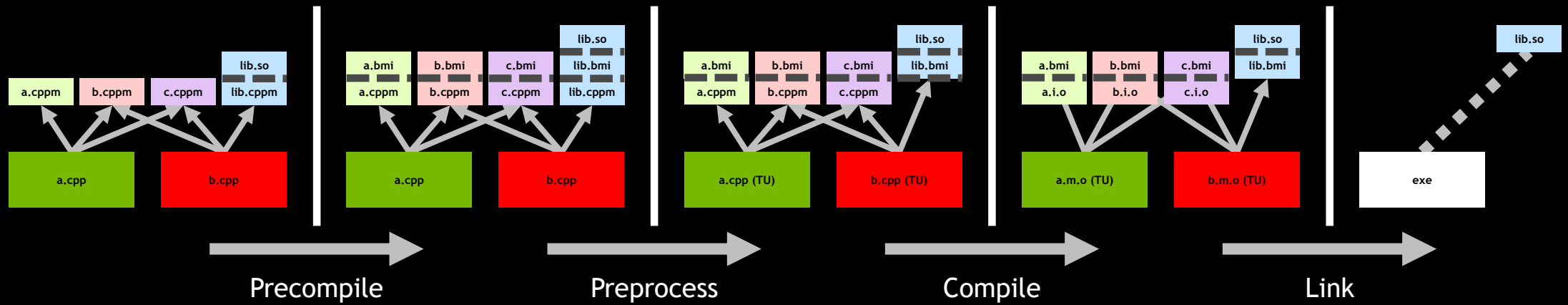


Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

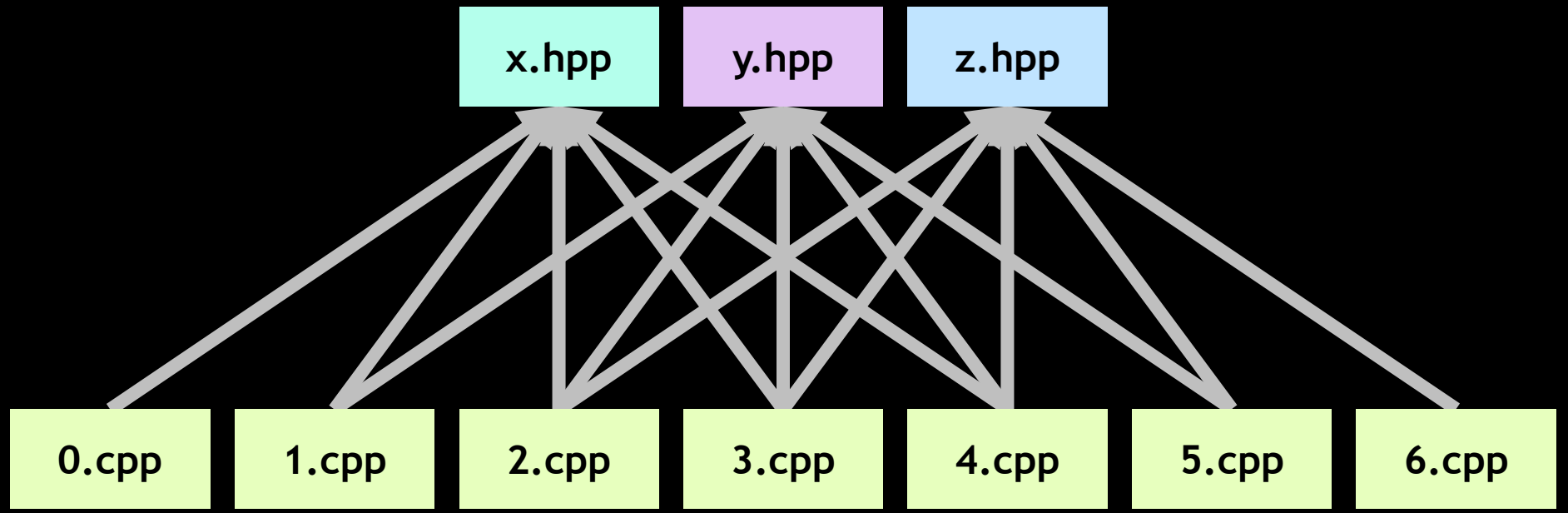


Modular Import



Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

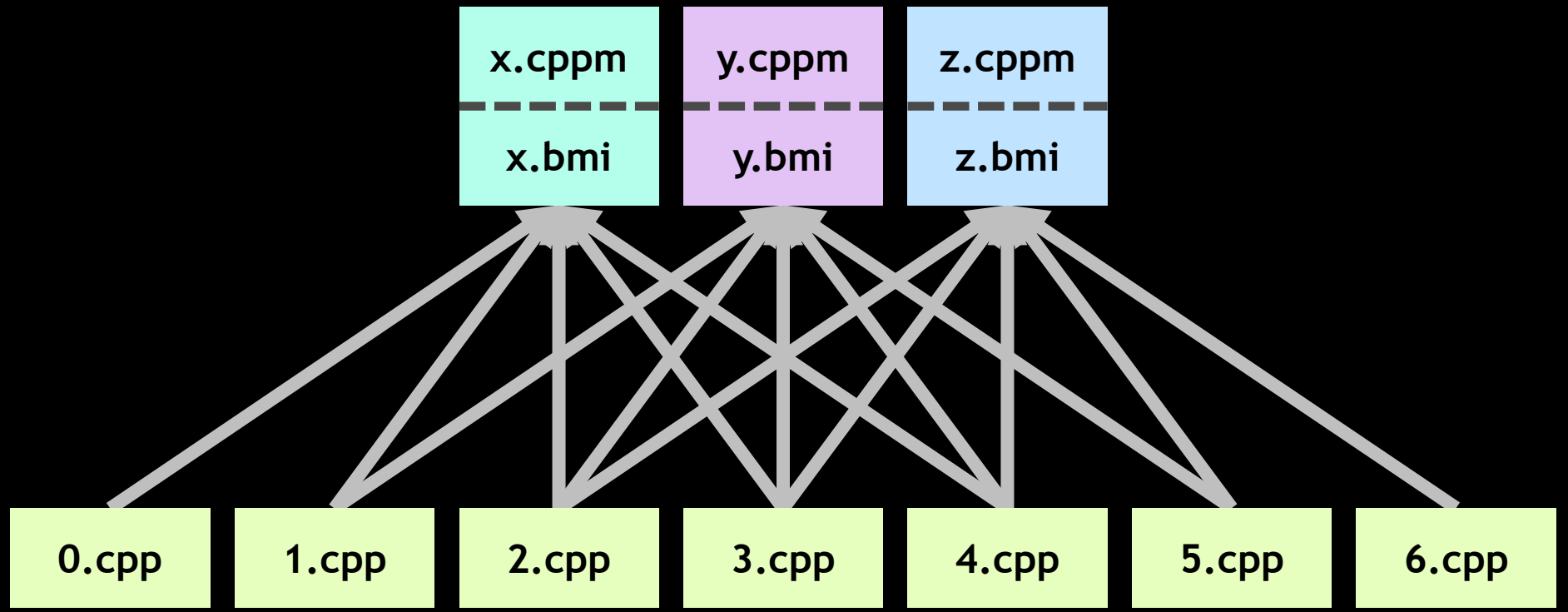


Pro: Embarrassingly parallel.

Con: x.hpp, y.hpp, and z.hpp are compiled 7 times.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Inclusion



Pro: x.cppm, y.cppm, and z.cppm are precompiled once.

Con: Not embarrassingly parallel.

Copyright (C) 2019 Bryce Adelstein Lelbach

Kinds of Translation Units

	Example	Extension	Artifact	Notes
Non-Modular Unit	`#include "…"` …	`.cpp`	`.o`	
Module Interface Unit	`export module …;` …	`.cppm`	`.bmi` `.o`	Exactly one per module.
Module Implementation Unit	`module …;` …	`.cpp`	`.o`	

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

```
#include <foo.hpp>
#include “foo.hpp”
```

Modular Import

```
import foo;
import <foo.hpp>;
import “foo.hpp”;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Importable headers:

- Most C++ standard library headers*.
- Some system headers.
- Headers you proclaim importable†.

* C standard library headers (`<cfoo>`, `<foo.h>`) are not required to be importable.

† Today, the mechanism for indicating importability is implementation defined (see P1905).

Copyright (C) 2019 Bryce Adelstein Lelbach

assert.h

```
#ifdef NDEBUG
#define assert(condition) ((void)0)
#else
#define assert(condition) /* … */
#endif
```

Copyright (C) 2019 Bryce Adelstein Lelbach

cstddef

```
namespace std { /* ... */ }
#define NULL /*see definition*/
#define offsetof(P, D) /*see definition*/
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
import <foo>;
```

Ill-formed if foo is not importable.

Your implementation defines the set of importable headers.

Copyright (C) 2019 Bryce Adelstein Lelbach

Your Code

```
#include <vector>
#include <iostream>

// …
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Your Code

```
#include <vector>
#include <iostream>

// …
```

Compiler Interpretation

Include Translation: Implementations are free to interpret #includes of importable headers as imports.

Copyright (C) 2019 Bryce Adelstein Lelbach

Kinds of Translation Units

	Example	Extension	Artifact	Notes
Non-Modular Unit	`#include "…"` `…`	`.cpp`	`.o`	
Header Unit	`// Created by:` `import <…>;`	`.hpp`	`.bmi` `.o`	
Module Interface Unit	`export module …;` `…`	`.cppm`	`.bmi` `.o`	Exactly one per module.
Module Implementation Unit	`module …;` `…`	`.cpp`	`.o`	

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

Interfaces (headers) are not translation units; they are expanded into translation units.

Modular Import

Interfaces are translation units.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are not translation units.

Copyright (C) 2019 Bryce Adelstein Lelbach

Writing Modules

Copyright (C) 2019 Bryce Adelstein Lelbach

```
import boost.spirit;
import ctre;
import blas.level1;
```

Module names are dot-separated identifiers.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
import boost.spirit;
import ctre;
import blas.level1;
```

Module names are dot-separated identifiers.
Dots in module names have no semantic meaning.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
module a;
// …
module b;
// …
```

Only one module declaration per translation unit.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export module a;
// …
export module b;
// …
```

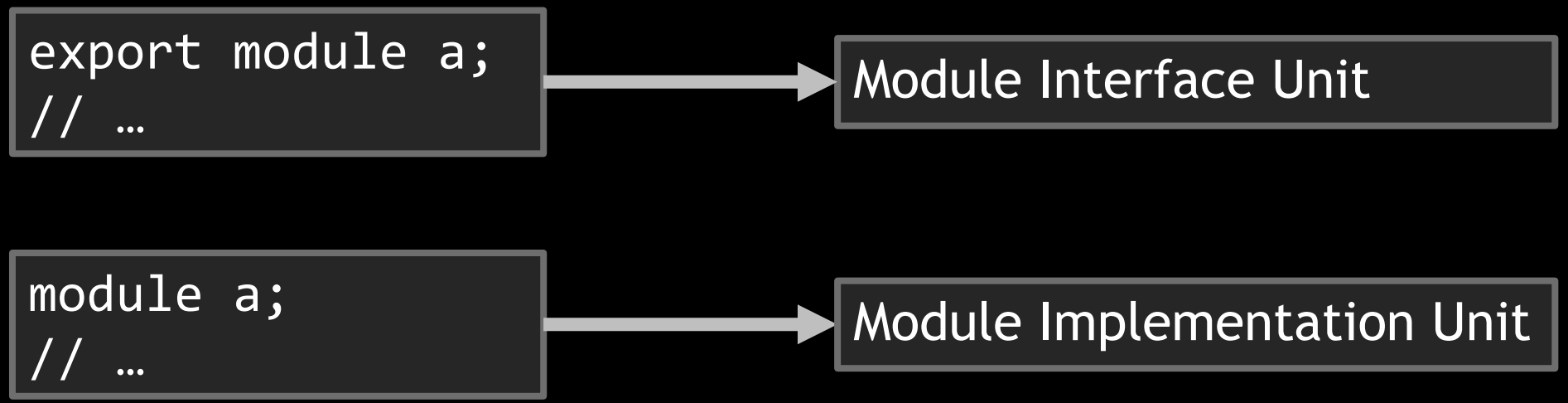
Only one module declaration per translation unit.

Copyright (C) 2019 Bryce Adelstein Lelbach

Module Unit Structure

```
export module …;
import …;
…
```

Copyright (C) 2019 Bryce Adelstein Lelbach



Copyright (C) 2019 Bryce Adelstein Lelbach

```
export declaration

export {
  declaration …
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export void f();

export int i{0};

export struct A;

export using B = A;

export using std::any;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

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

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export template <typename T>
T square(T t) { return t * t; }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export template <typename T>
struct is_const : false_type {};

export template <typename T>
struct is_const<T const> : true_type {};
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export namespace foo { struct A; }

namespace foo { struct B; }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export namespace foo { struct A; }

namespace foo { struct B; }
```

Only `foo::A` is exported.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
export import a;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

square.cppm

```
export module math.square;

export template <typename T>
T square(T a) { return a * a; }
```

add.cppm

```
export module math.add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

square.cppm

```
export module math.square;

export template <typename T>
T square(T a) { return a * a; }
```

add.cppm

```
export module math.add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

math.cppm

```
export module math;

export import math.square;
export import math.add;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

struct S { int m; };
export S foo();
```

main.cpp

```
import a;

int main() {
  auto s0 = foo();
  s0.m = 42;
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

struct S { int m; };
export S foo();
```

main.cpp

```
import a;

int main() {
  auto s0 = foo();
  s0.m = 42;

  S s1{};
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

struct S { int m; };
export S foo();
```

main.cpp

```
import a;

int main() {
  auto s0 = foo();
  s0.m = 42;

  decltype(foo()) s1{};
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Visible: In scope, can be named.

Reachable: In scope, not necessarily namable.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

struct S { int m; };
export S foo();
```

main.cpp

```
import a;
```

In `main.cpp`:

- `S` is reachable.
- `foo` is reachable and visible.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules enable true encapsulation.

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

struct foo { /* … */ };
```

b.hpp

```
#pragma once

void bar(foo f);
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

struct foo { /* … */ };
```

b.hpp

```
#pragma once

void bar(foo f);
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Modular Import

a.cppm

```
export module a;

export struct foo { /* … */ };
```

b.cppm

```
export module b;

export void bar(foo f);
```

main.cpp

```
import a;
import b;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

struct foo { /* … */ };
```

b.hpp

```
#pragma once

void bar(foo f);
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Modular Import

a.cppm

```
export module a;

export struct foo { /* … */ };
```

b.cppm

```
export module b;

export void bar(foo f);
```

main.cpp

```
import a;
import b;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

// Implementation detail/not part of the API.
template <typename T>
T id(T t) { return t; }
```

b.hpp

```
#pragma once

// Implementation detail/not part of the API.
template <typename T>
struct id { using type = T; };
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

// Implementation detail/not part of the API.
template <typename T>
T id(T t) { return t; }
```

b.hpp

```
#pragma once

// Implementation detail/not part of the API.
template <typename T>
struct id { using type = T; };
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

namespace a {
 template <typename T>
 T id(T t) { return t; }
}
```

b.hpp

```
#pragma once

namespace b {
 template <typename T>
 struct id { using type = T; };
}
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

namespace a::detail {
 template <typename T>
 T id(T t) { return t; }
}
```

b.hpp

```
#pragma once

namespace b::detail {
 template <typename T>
 struct id { using type = T; };
}
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

namespace a::detail {
 template <typename T>
 T __dont_use_id(T t) { return t; }
}
```

b.hpp

```
#pragma once

namespace b::detail {
 template <typename T>
 struct __dont_use_id { using type = T; };
}
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

namespace a::detail {
 template <typename T>
 T __dont_use_id(T t) { return t; }
}
```

b.hpp

```
#pragma once

namespace b::detail {
 template <typename T>
 struct __dont_use_id { using type = T; };
}
```

main.cpp

```
#include “a.hpp”
#include “b.hpp”
```

Modular Import

a.cppm

```
export module a;

template <typename T>
T id(T t) { return t; }
```

b.cppm

```
export module b;

template <typename T>
struct id { using type = T; };
```

main.cpp

```
import a;
import b;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

No more `detail/impl` namespaces.

No more uglifying identifiers.

Copyright (C) 2019 Bryce Adelstein Lelbach

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

Non-header-unit never expose macros.

```
import foo;         // No macros brought in.
import <foo.hpp>; // Macros brought in.
import “foo.hpp”; // Macros brought in.
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

#define foo int
```

main.cpp

```
import a;

foo f{};
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

#define foo int
```

main.cpp

```
import a;

foo f{};
```

The definition of `foo` isn't brought in by the `import`.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

#define foo int
```

main.cpp

```
import <a>;

foo f{};
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

#define foo int
```

main.cpp

```
import <a>;

foo f{};
```

The definition of `foo` is brought in by the `import`.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules never see the definition of programmatic macros defined in other modules.

Programmatic macros == `#define FOO`

Non-programmatic macros == `-DFOO`

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

export struct foo{};
```

main.cpp

```
#define foo bar
import a;
#undef foo

foo f{};
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

export struct foo{};
```

main.cpp

```
#define foo bar
import a;
#undef foo

foo f{};
```

The definition of `foo` isn't seen by the imported module.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

#if defined(DEBUG)
  // …
#else
  // …
#endif
```

main.cpp

```
#define DEBUG
import a;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

#if defined(DEBUG)
  // …
#else
  // …
#endif
```

main.cpp

```
#define DEBUG
import a;
```

The definition of DEBUG isn’t seen by the imported module.

Copyright (C) 2019 Bryce Adelstein Lelbach

a.hpp

```
#pragma once

#if defined(DEBUG)
  // …
#else
  // …
#endif
```

main.cpp

```
#define DEBUG
import <a>;
```

The definition of DEBUG isn’t seen by the imported header-unit.

Copyright (C) 2019 Bryce Adelstein Lelbach

Programmatic macros are:

- Never seen by modules.
- Never exported from non-header-unit modules.

Copyright (C) 2019 Bryce Adelstein Lelbach

With modules we no longer have to defend names:

- No more `detail/impl` namespaces.
- No more uglifying identifiers.

Copyright (C) 2019 Bryce Adelstein Lelbach

Bug ID:	2547402
Date:	3/6/2019 2:56:55 PM
Synopsis:	CUB: NAMD defines WARP_ANY as a macro, which conflicts with CUB
Module:	CUDA - C++ Core Libraries - Thrust

```
281  /**
282   * Warp any
283   */
284  __device__  __forceinline__ int WARP_ANY(int predicate, unsigned int member_mask)
285  {
286  #ifdef CUB_USE_COOPERATIVE_GROUPS
287      return __any_sync(member_mask, predicate);
288  #else
289      return ::__any(predicate);
290  #endif
291  }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
#define _LIBCPP_NO_EXCEPTIONS
import <vector>;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
#define _LIBCPP_NO_EXCEPTIONS
import <vector>;
```

The definition of _LIBCPP_NO_EXCEPTIONS isn't seen by <vector>.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
#define _LIBCPP_NO_EXCEPTIONS
#include <vector>
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
#define _LIBCPP_NO_EXCEPTIONS
#include <vector>
```

If include translation occurs here,
the definition of `_LIBCPP_NO_EXCEPTIONS` isn't seen by `<vector>`.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
#define NDEBUG
#include <cassert>

// For `readlink`.
#define _XOPEN_SOURCE
#include <unistd.h>
```

How do we deal with non-modular headers?

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#define NDEBUG
#include <cassert>

// …
```

Modular Import

a.cppm

```
export module a;
#define NDEBUG
import <cassert>;

// …
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#define NDEBUG
#include <cassert>

// …
```

Modular Import

a.cppm

```
export module a;
#define NDEBUG
#include <cassert>

// …
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Module Unit Structure

```
module;
#pp-directive …;
export module …;
import …;
…
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#define NDEBUG
#include <cassert>

// …
```

Modular Import

a.cppm

```
module;
#define NDEBUG
#include <cassert>
export module a;

// …
```

Copyright (C) 2019 Bryce Adelstein Lelbach

```
module;
#include <boost/circular_buffer>
export module boost.circular_buffer;
namespace boost {
  export using ::boost::circular_buffer;
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are order independent.

Copyright (C) 2019 Bryce Adelstein Lelbach

```
import a;
import b;
```

==

```
import b;
import a;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules cannot have cycles.

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#include “b.hpp”

struct Y;
struct X { Y* y; };
```

b.hpp

```
#pragma once
#include “a.hpp”

struct X;
struct Y { X* x; };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#include “b.hpp”

struct Y;
struct X { Y* y; };
```

b.hpp

```
#pragma once
#include “a.hpp”

struct X;
struct Y { X* x; };
```

Modular Import

a.cppm

```
export module a;
import b;

struct Y;
export struct X { Y* y; };
```

b.cppm

```
export module b;
import a;

struct X;
export struct Y { X* x; };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once
#include “b.hpp”

struct Y;
struct X { Y* y; };
```

b.hpp

```
#pragma once
#include “a.hpp”

struct X;
struct Y { X* x; };
```

Modular Import

a.cppm

```
export module a;
import b;

struct Y;
export struct X { Y* y; };
```

b.cppm

```
export module b;
import a;

struct X;
export struct Y { X* x; };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import

a.cppm

```
export module a;

struct Y;
export struct X { Y* y; };
```

b.cppm

```
export module b;

struct X;
export struct Y { X* x; };
```

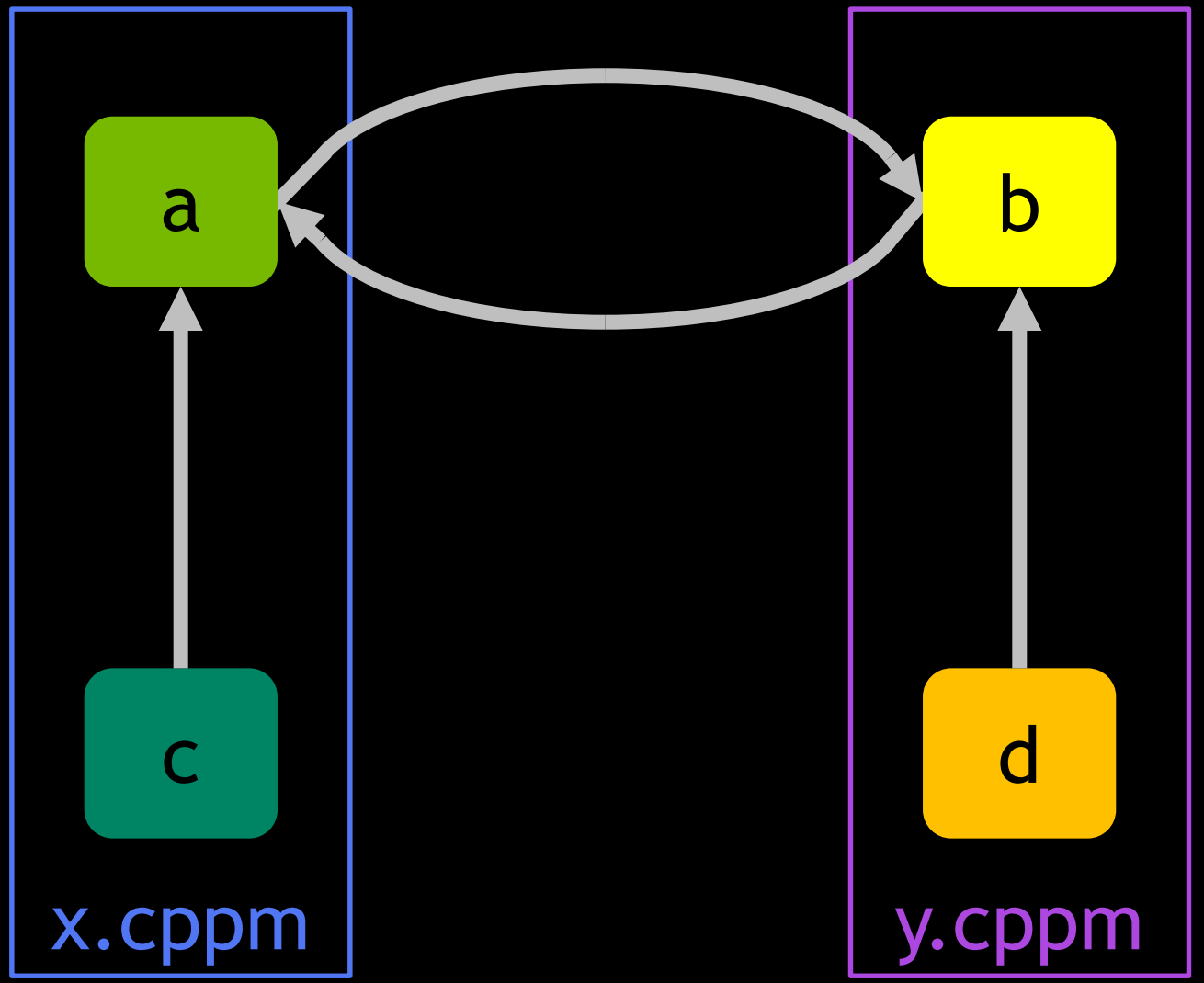
main.cpp

```
import a;
import b;
```

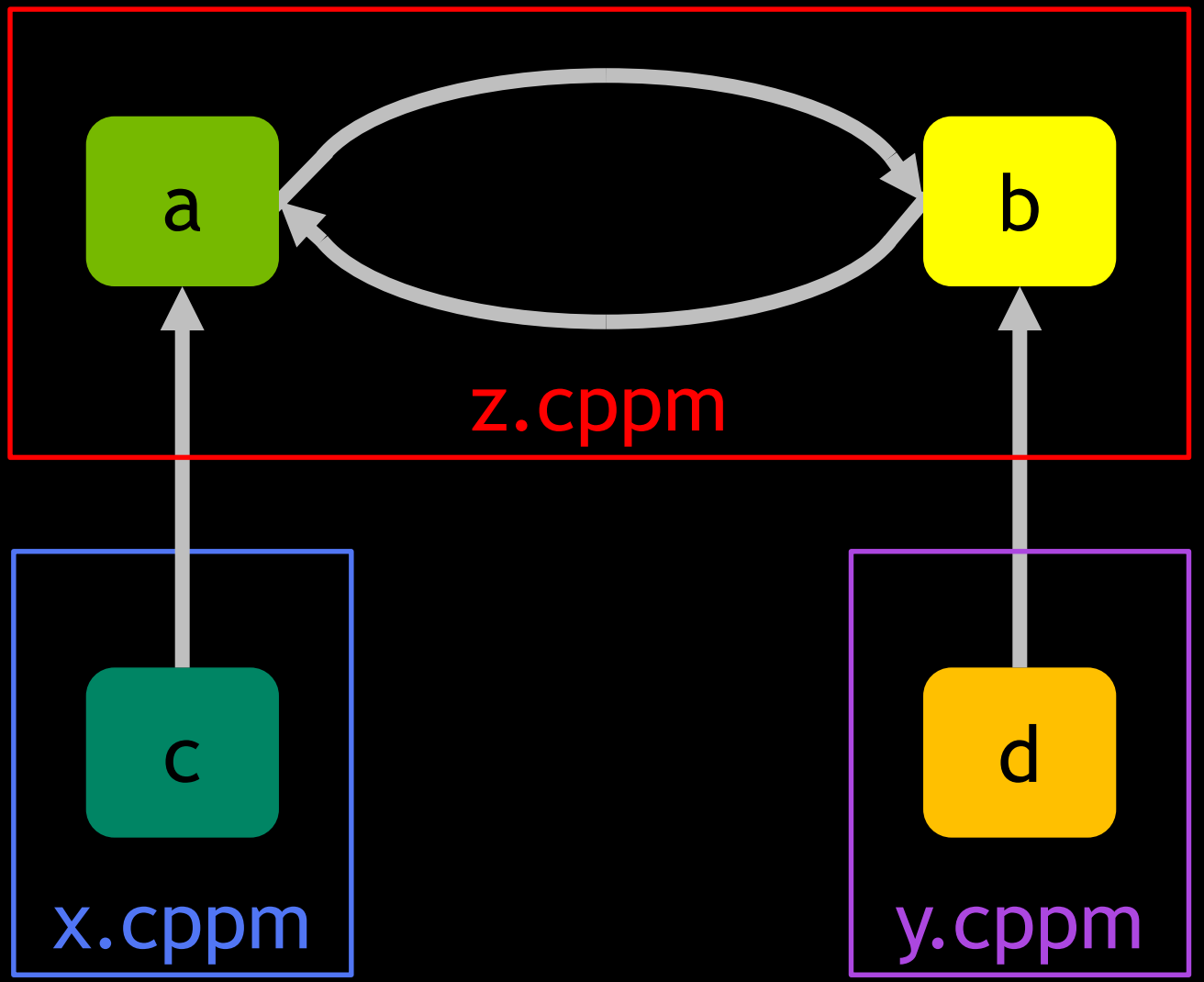
Copyright (C) 2019 Bryce Adelstein Lelbach

How do we break cycles?

Copyright (C) 2019 Bryce Adelstein Lelbach



Copyright (C) 2019 Bryce Adelstein Lelbach



Copyright (C) 2019 Bryce Adelstein Lelbach

Breaking cyclic dependencies:

- Is difficult to automate.
- May require major structural changes.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules own their declarations.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modular Import

a.cppm

```
export module a;

struct Y;
export struct X { Y* y; };
```

b.cppm

```
export module b;

struct X;
export struct Y { X* x; };
```

main.cpp

```
import a;
import b;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

You can not forward declare an entity
that lives in another module.

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

void foo();
```

a.cpp

```
#include “a.hpp”

void foo() { /* … */ }
```

b.cpp

```
#include “a.hpp”

static void foo() { /* … */ }
```

Modular Import

a.cppm

```
export module a;

export void foo();
```

a.cpp

```
module a;

void foo() { /* … */ }
```

b.cpp

```
import a;

static void foo() { /* … */ }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

void foo();
```

a.cpp

```
#include “a.hpp”

void foo() { /* … */ }
```

b.cpp

```
#include “a.hpp”

static void foo() { /* … */ }
```

Modular Import

a.cppm

```
export module a;

export void foo();
```

a.cpp

```
module a;

void foo() { /* … */ }
```

b.cpp

```
import a;

static void foo() { /* … */ }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

export void foo();
```

b.cppm

```
export module b;

export void foo();
```

Copyright (C) 2019 Bryce Adelstein Lelbach

a.cppm

```
export module a;

export void foo();
```

b.cppm

```
export module b;

export void foo();
```

Ill formed, no diagnostic required (IFNDR).

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules can be contained in one file.

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

math.hpp

```
#pragma once

template <typename T>
T square(T a) { return a * a; }
```

main.cpp

```
import square;

int main() { return square(42); }
```

Modular Import

math.cppm

```
export module math;

export template <typename T>
T square(T a) { return a * a; }
```

main.cpp

```
import math;

int main() { return square(42); }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

struct pimpl;
```

a.cpp

```
#include “a.hpp”

struct pimpl { /* … */ };
```

Modular Import

a.cppm

```
export module a;

export struct pimpl;
```

a.cpp

```
module a;

struct pimpl { /* … */ };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Textual Inclusion

a.hpp

```
#pragma once

struct pimpl;
```

a.cpp

```
#include “a.hpp”

struct pimpl { /* … */ };
```

Modular Import

a.cppm

```
export module a;

export struct pimpl;

module : private;

struct pimpl { /* … */ };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Module Unit Structure

```
module;
#pp-directive …;
export module …;
import …;
…
module : private;
…
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules can be split across multiple files.

Copyright (C) 2019 Bryce Adelstein Lelbach

square.cppm

```
export module math.square;

export template <typename T>
T square(T a) { return a * a; }
```

add.cppm

```
export module math.add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

math.cppm

```
export module math;

export import math.square;
export import math.add;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

square.cppm

```
export module math:square;

export template <typename T>
T square(T a) { return a * a; }
```

add.cppm

```
export module math:add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

math.cppm

```
export module math;

export import :add;
export import :square;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

math_vector.cppm

```
export module math.vector;
export import :dot_product;
float sqrt(float x);
```

Copyright (C) 2019 Bryce Adelstein Lelbach

math_vector.cppm

```
export module math.vector;
export import :dot_product;
float sqrt(float x);
```

math_vector_dot_product.cppm

```
export module math.vector:dot_product;
import :sum_squares;
import <span>;
export float vector_norm(std::span<float> x) {
  return sqrt(sum_squares(x));
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

math_vector.cppm

```
export module math.vector;
export import :dot_product;
float sqrt(float x);
```

math_vector_dot_product.cppm

```
export module math.vector:dot_product;
import :sum_squares;
import <span>;
export float vector_norm(std::span<float> x) {
  return sqrt(sum_squares(x));
}
```

math_vector_sum_squares.cpp

```
module math.vector:sum_squares;
import <span>;
import <algorithm>;
export float sum_squares(std::span<float> x) {
  return std::transform_reduce(x);
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

math_vector.cppm

```
export module math.vector;
export import :dot_product;
float sqrt(float x);
```

math_vector_dot_product.cppm

```
export module math.vector:dot_product;
import :sum_squares;
import <span>;
export float vector_norm(std::span<float> x) {
  return sqrt(sum_squares(x));
}
```

math_vector_sum_squares.cpp

```
module math.vector:sum_squares;
import <span>;
import <algorithm>;
float sum_squares(std::span<float> x) {
  return std::transform_reduce(x);
}
```

math_vector.cpp

```
module math.vector;
float sqrt(float x) { /* ... */ }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

Kinds of Translation Units

	Example	Extension	Artifact	Notes
Non-Modular Unit	`#include "…"` `…`	`.cpp`	`.o`	
Header Unit	`// Created by:` `import <…>;`	`.hpp`	`.bmi` `.o`	
Module Interface Unit	`export module …;` `…`	`.cppm`	`.bmi` `.o`	Exactly one per module.
Module Implementation Unit	`module …;` `…`	`.cpp`	`.o`	
Module Partition Interface Unit	`export module …:…;` `…`	`.cppm`	`.bmi` `.o`	
Module Partition Implementation Unit	`module …:…;` `…`	`.cpp`	`.o`	

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules do not force a file layout on you.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are not translation units.

Copyright (C) 2019 Bryce Adelstein Lelbach

`#include` is problematic:

- Slow to compile.
- ODR violations.
- Lack of encapsulation.
- Cyclic dependencies.
- Order dependent.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are coming:

- Modules will bring substantial algorithmic build throughput improvements.
- Modules offer true encapsulation which will eliminate many structural challenges with writing C++ at scale.
- Transitioning to modules will not be free.

Copyright (C) 2019 Bryce Adelstein Lelbach

Ecosystem Impact

Copyright (C) 2019 Bryce Adelstein Lelbach

math.hpp

```
#pragma once

int square(int a);
```

math.cpp

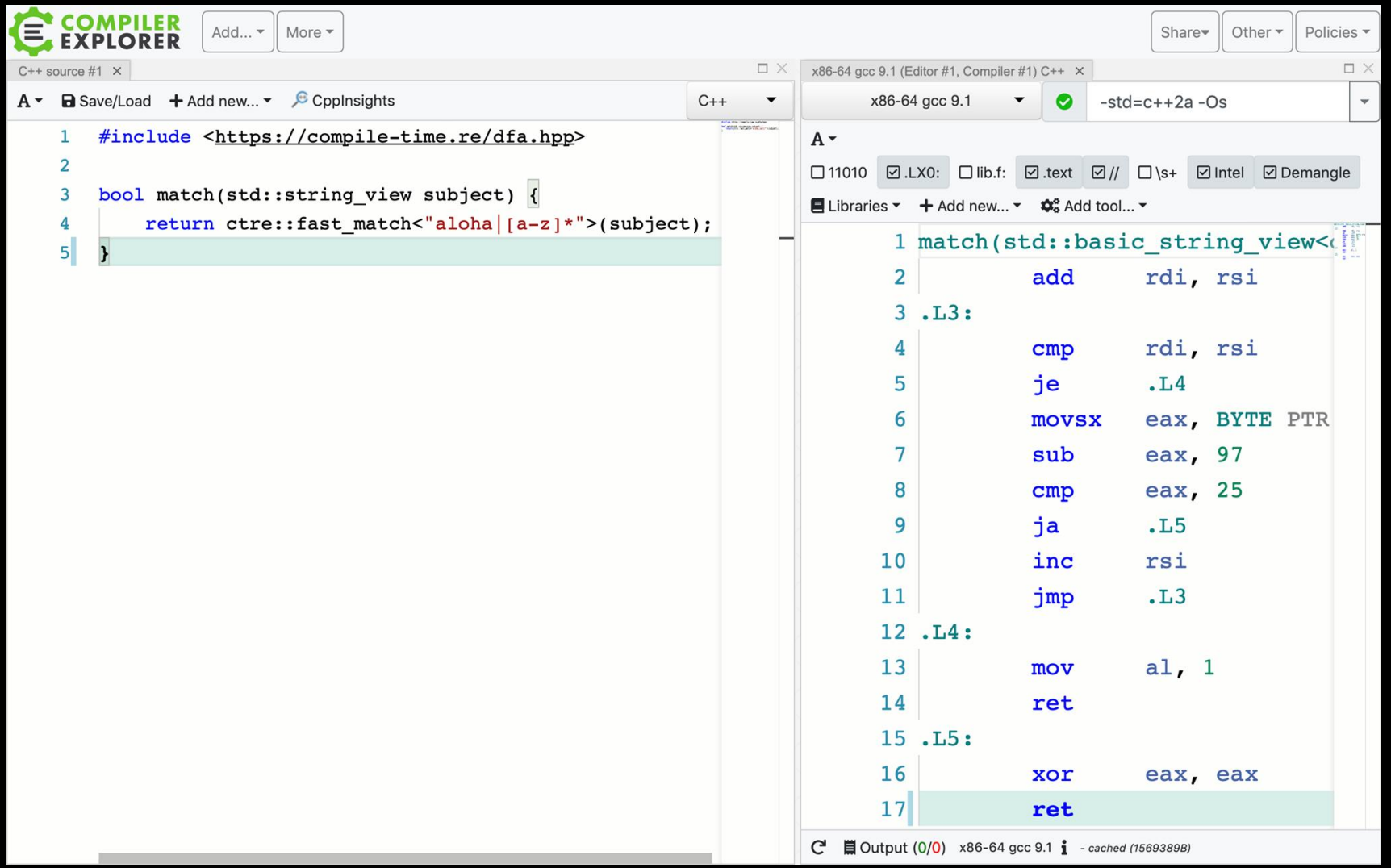
```
#include “math.hpp”

int square(int a) { return a * a; }
```

How are headers found?

- Not specified by the standard.
- In practice, all implementations assume a mapping between file names and #includes.
- The file is searched for in a set of include paths.

Copyright (C) 2019 Bryce Adelstein Lelbach



Copyright (C) 2019 Bryce Adelstein Lelbach

bar.cppm

```
export module foo;

// …
```

How are modules found?

- Not specified by the standard.
- Unlike headers, modules are programmatically named.
- A file name <-> module name mapping is not straightforward.
  - Modules have to be precompiled.
  - Partitions span multiple files.

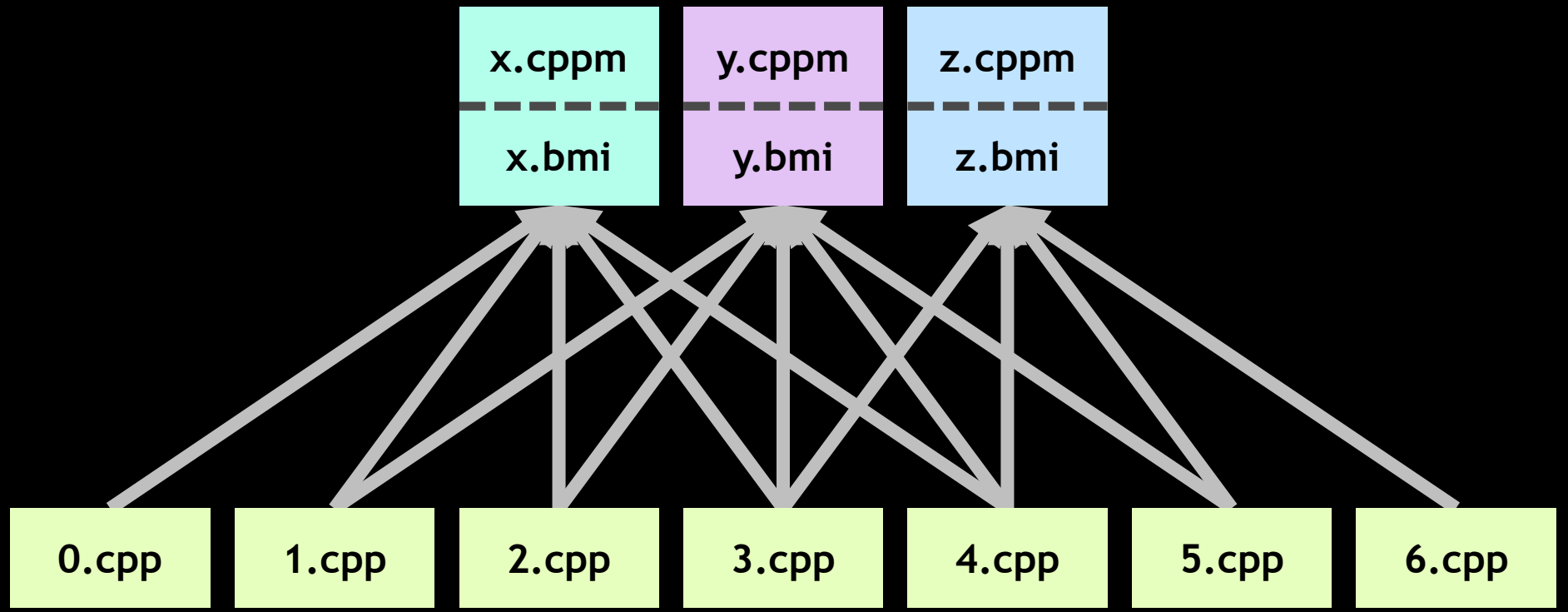
Copyright (C) 2019 Bryce Adelstein Lelbach

Strategies for Module Lookup

- (Fast) Dependency Scanner.
- Assume File Name == Module Name and Search.
- Explicit Mapping.
- Client/Server Mapper.

Copyright (C) 2019 Bryce Adelstein Lelbach

Implicit Precompilation is Problematic



If precompilation is implicit and builds are parallel, how do we decide who builds the BMIs?

Copyright (C) 2019 Bryce Adelstein Lelbach

Implicit Precompilation is Problematic

Assuming the existence of fast dependency scanners, you can implicitly discover what BMIs need to be built.

This comes with a cost, however; you'll need to do this scanning as an additional pass prior to compilation.

Copyright (C) 2019 Bryce Adelstein Lelbach

Module Recipes

- BMIs are built with a certain set of compiler options and global macro definitions (the module recipe).
  - Ex: GCC 7.3, -O3, -DDEBUG
- The BMI may only be used when compiling with the same set of compiler options and global macro definitions.
- Tools need to either interface with the compiler and read BMIs, or be able to build BMIs of their own using the module recipe.
- Module lookup isn't just a matter of mapping a module name to a module interface unit (MIU) + a BMI. It's mapping a module name + module recipe to a MIU + a BMI.

Copyright (C) 2019 Bryce Adelstein Lelbach

Guidelines for BMI Redistribution

- BMIs are built with a certain set of compiler options and global macro definitions (the module recipe).
- The source for module interface units (MIUs) must always be distributed.
- BMIs may be distributed as an optimization alongside MIU sources.
- BMIs must be distributed with module recipes to allow tools to rebuild BMIs if needed.

Copyright (C) 2019 Bryce Adelstein Lelbach

Tools can no longer rely on simple lookup mechanism (include directories and header file names) to understand C++ projects.

Tools must be able to either consume BMIs or rebuild BMIs using module recipes.

Dependency scanning now requires a C++ parser, not just a C preprocessor.

Copyright (C) 2019 Bryce Adelstein Lelbach

C++ Ecosystem Technical Report

(P1688, P1838)

Copyright (C) 2019 Bryce Adelstein Lelbach

C++ Ecosystem Technical Report:

- Packaging Modules (P1767).
- Module Mapping (P1302, P1484).
- Module Naming (P1634).
- Module Transition Path.
- Module Recipes (P1788).
- Built Module Interface (BMI) Distribution (P1788).
- Dependency Scanning (P1689, P1857).

Copyright (C) 2019 Bryce Adelstein Lelbach

C++ Ecosystem Technical Report Metadata Formats:

- Package Manifest Format (P1767).
- Dependency Format (P1689).
- Module Recipe Format (P1788).

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are coming:

- Modules will bring substantial algorithmic build throughput improvements.
- Modules offer true encapsulation which will eliminate many structural challenges with writing C++ at scale.
- Transitioning to modules will not be free.

Copyright (C) 2019 Bryce Adelstein Lelbach

Modules are a team effort.

Thanks to everyone involved!

Copyright (C) 2019 Bryce Adelstein Lelbach

Copyright (C) 2019 Bryce Adelstein Lelbach