C++ Modules

Nathan Sidwell
Facebook
nathans@fb.com

Modules 101

```
// User of modules
import Foo; // module Foo gives me stuff
import Bar; // as does Bar
…
int a = some_foo_fn (…);
some_bar_type x (…);

some_internal_foo_fn (1,2,3); // ERROR

void name_in_foo (int); // Not the same
```

Interfaces and Implementations

- Interface Unit

```
export module Foo;
int acc = 0; // Only Foo sees this
export int frob (int);
static int unused; // Only here
```

- Implementation units

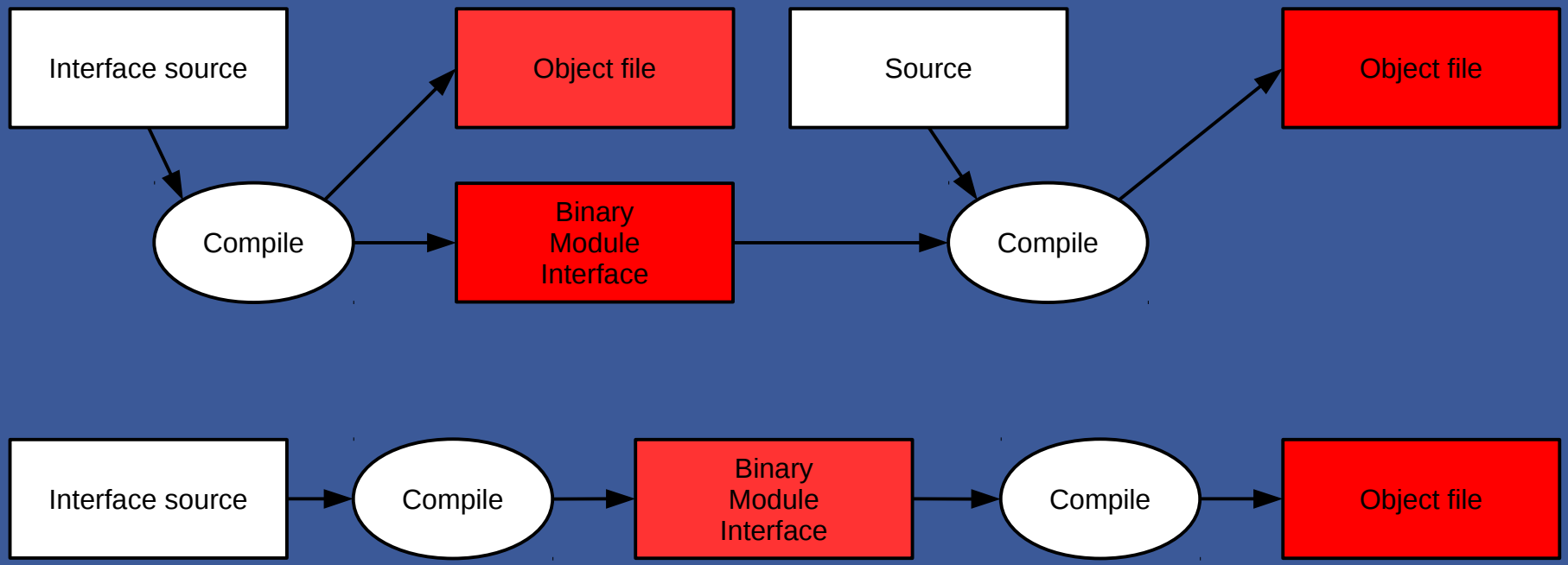
```
module Foo; // See the interface
int frob (int a) {
  acc += a;
  return acc;
}
```

Look Ma! It’s Combined

```
export module Foo;
int acc = 0;
export int frob (int a) {
  acc += a;
  return acc;
}
```

Why Modules?

- It’s all about the build speed
  - Oh, better encapsulation is good too



How Do We Get There From Here?

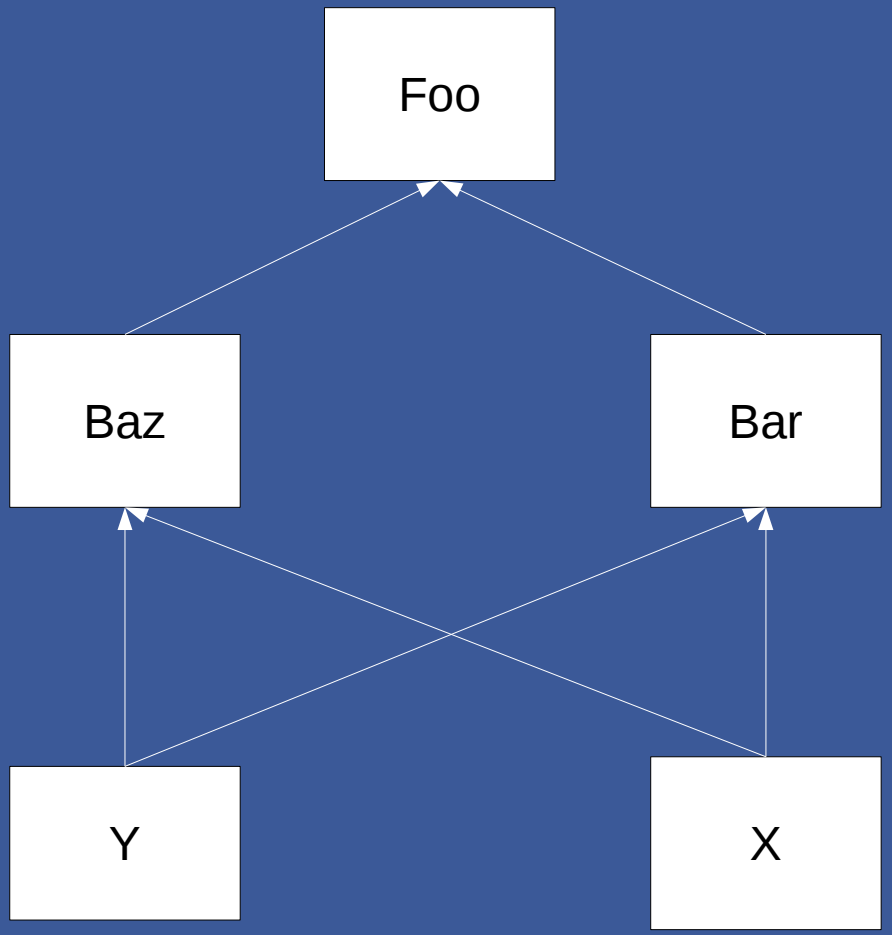
Photo by Ken Treloar on Unsplash

#includes

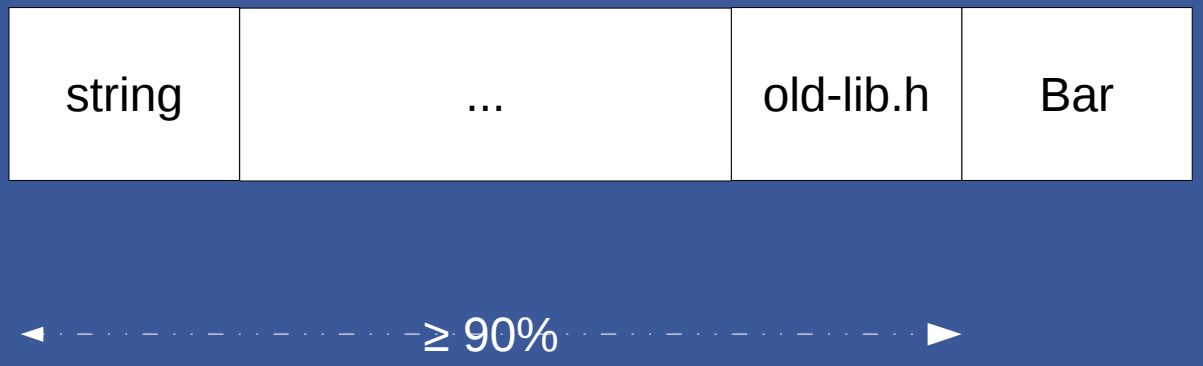
- I cannot `#include <string>` in module purview
- ```
  module;
  // global module fragment here ...
  #include <string>
  #include “old-lib.h”

  export module Foo;
  // module Foo’s purview here ...
  export std::string cons ();
  ```
- Each fragment is a different view into a single ‘module’
  ```

Building Modules



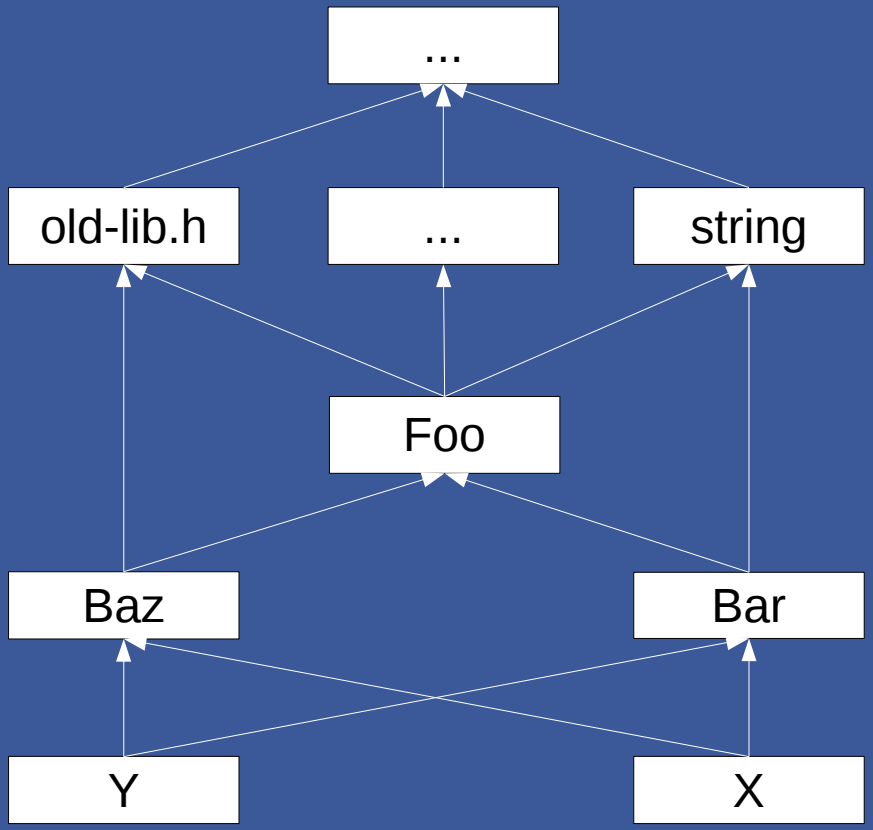
#include Parsing



- Need to merge declarations across global module fragments
  - Ameliorated by only requiring referenced decls

Header Files

- Many headers are modularish




Another Take On Modules
Photo by Sharon McCutcheon on Unsplash

Legacy Header Units

```
export module Foo;
import "old-lib.h";
import <string>;
export std::string cons (…);
```

- Special compilation mode to create legacy module
  - Everything is exported
  - Each is global module fragment
- Blind to earlier #defines
- Its #defines are exported too

Partitions:part1

- ```
  export module Baz;
  export import Baz.part1;
  export import Baz.part2;
  ```
- What if I need to move entities between part1 and part2?
  - I'll break binary compatibility
- ```
  export module Baz;
  export import :part1;
  export import :part2;
  ```
- ```
  module Baz:part1;
  ```
  ```

Partitions:part2

- ```
  module Baz:impl;
  struct my_1 { ... };
  ```
- ```
  export module Baz;
  export struct my_type;
  import :impl;
  ```
- Allows simpler exported semantics rule
- Replaces proclaiming decls
  - extern module Foo : *declaration*

Preamble

```
export module Baz;
import "old-lib.h";
import <string>;
import Foo;
// ends here
int frob;
```

- Block of imports at start
  - No earlier declarations
  - No later imports
- Legacy import #defines immediately available

Merging Proposals

Photo by Kelly Neil on Unsplash

Working Modules Document

- Some ATOM went in
  - Legacy import units
  - Partitions
  - Preamble (for modules)
- Some TS removed
  - Proclaiming Ownership
  - Reachable Semantic Properties

To Do

- ADL Path of Instantiation
- Global Module Pruning
- End of Preamble Simplification
- Inline Partitions

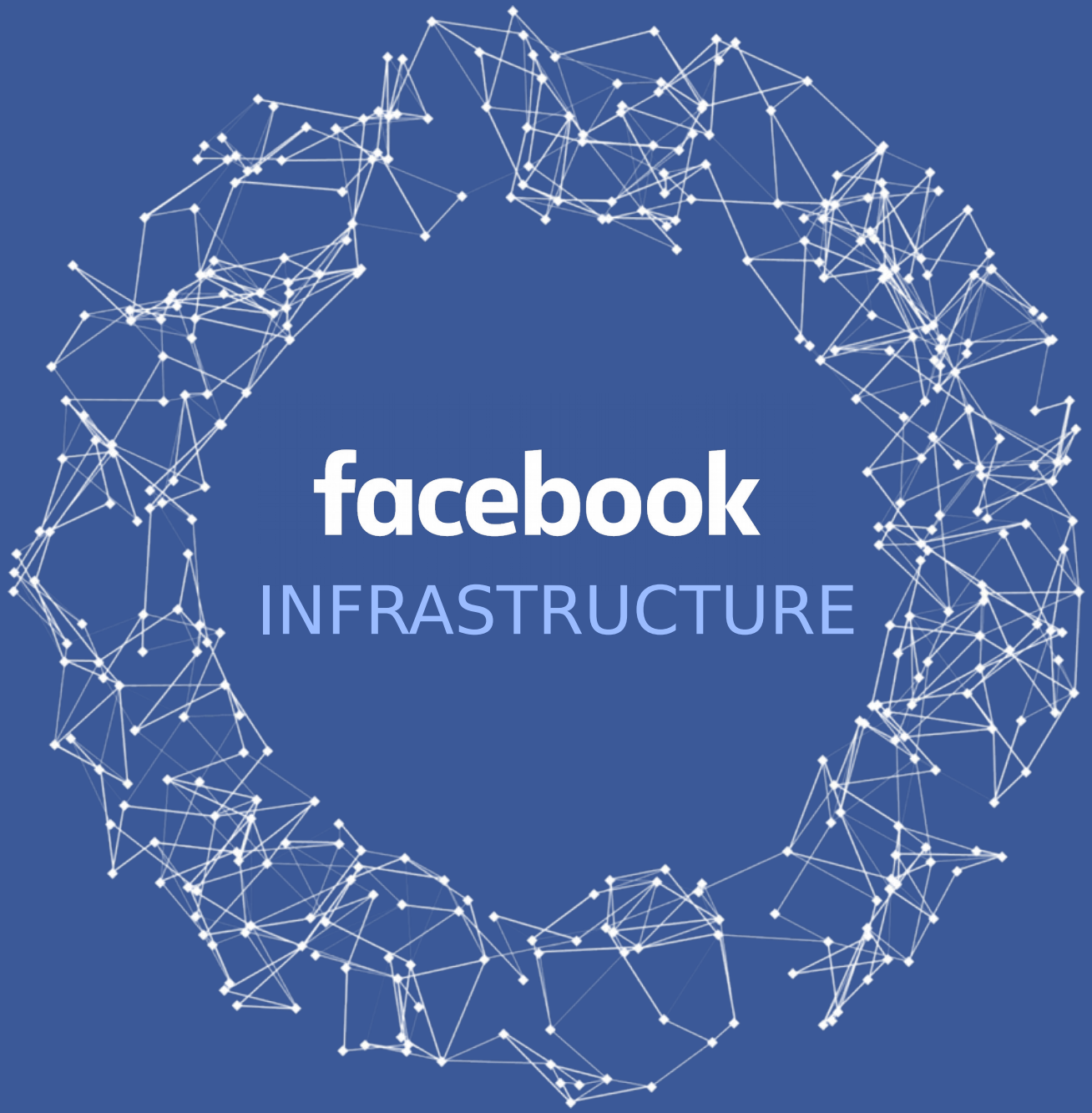

Driving The Compiler
WAR-GOD
Photo by TUAN ANH TRAN on Unsplash

Driving the Compiler

- Some assembly required:
  - `https://gcc.gnu.org/wiki/cxx-modules`
- `g++ -fmodules-ts -c hello.cc`
  - `hello.o` Regular object file
  - `hello.nms` Binary Module Interface
- `g++ -fmodules-ts -c main.cc`
  - Reads `hello.nms`
- `g++ main.o hello.o -o main`

ATOM

- Replace `-fmodules-ts` with `-fmodules-atom`
  - `g++ -fmodules-atom -c hello.cc`
  - `g++ -fmodules-atom -c main.cc`
- Legacy header mode
  - `g++ -fmodule-legacy -c header.h`


facebook
INFRASTRUCTURE

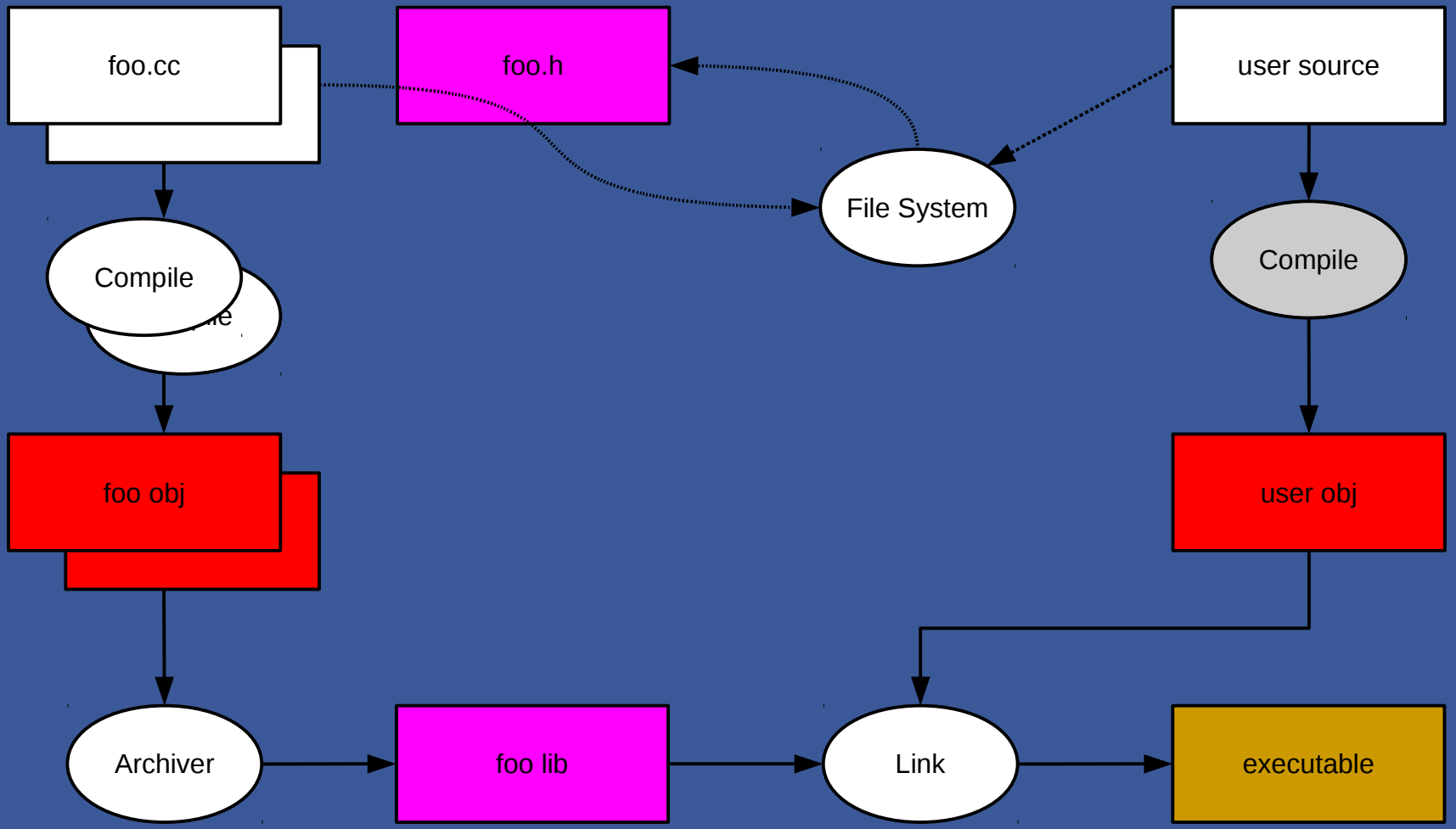
Time Line

- Standardization:
  - July 2018: Merged TS – Some TS out, some ATOM in
  - Sept 2018: Modules workshop
  - Nov 2018: San Diego C++ meeting
  - Feb 2019: Kona C++ meeting
  - … C++ 20

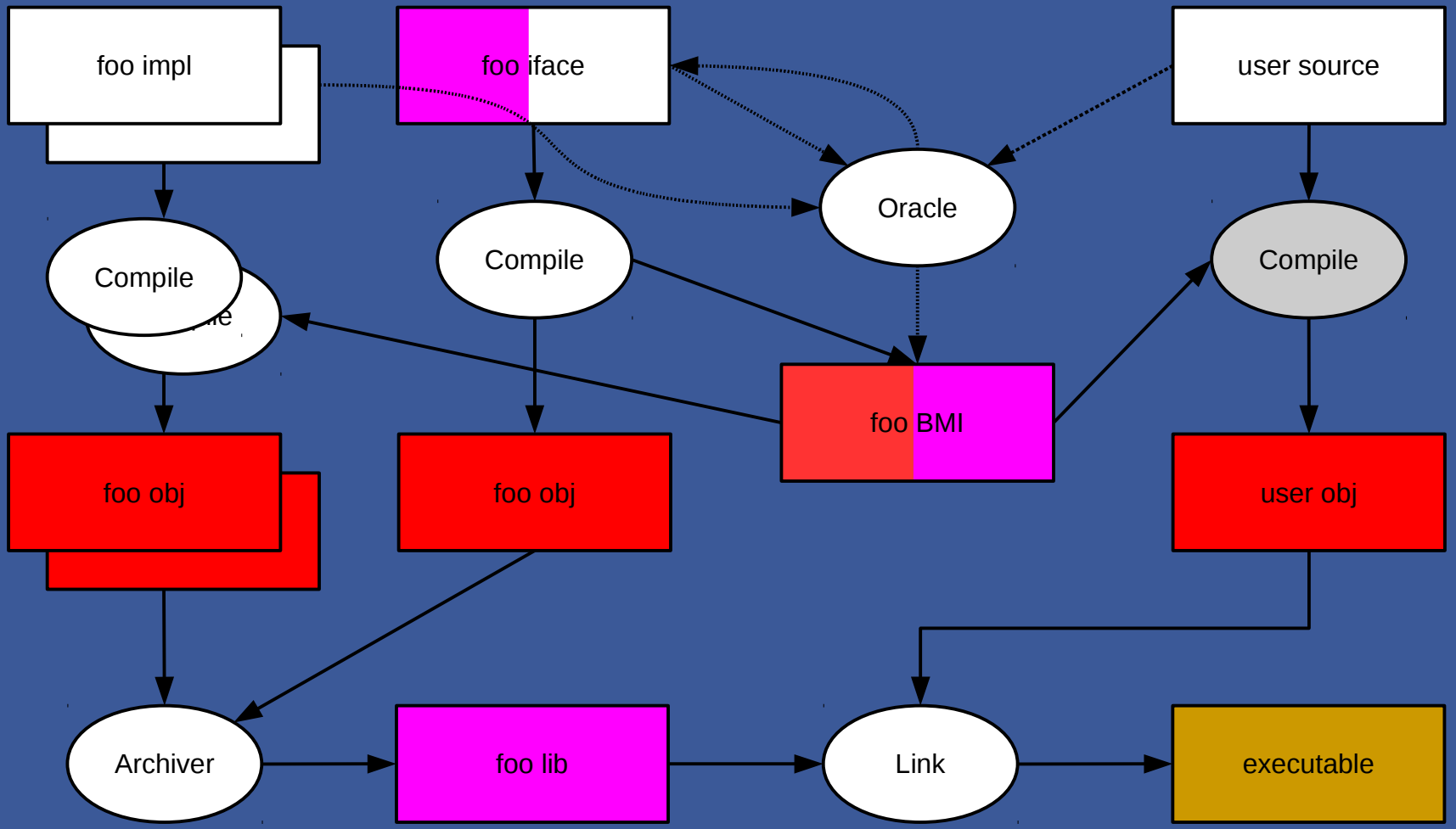
- To Do Implementaton
  - Legacy modules, partitions, ADL & global module

Photo by Eirik Skarstein on Unsplash

Headers


foo.cc
foo.h
user source
File System
Compile
Compile
foo obj
user obj
Archiver
foo lib
Link
executable

Modules


foo impl
foo iface
user source
Oracle
Compile
Compile
Compile
foo BMI
foo obj
foo obj
user obj
Archiver
foo lib
Link
executable

Thoughts #1, #2 & #3

- BMIs are like generated headers
  - Maybe from known sources
  - Probably very compiler-specific form
- Try a search-path?
  - -fmodule-include=….
- Lock filename to module name?
- What if it needs compiling?
  - -fmodule-wrapper=...

Lazy Graph Discovery

- Discover import graph during build
  - Compiler sees ‘`import Foo;`’
  - Asks an oracle for Foo
  - Waits for response before continuing
- That’s unfortunately serial
- Preamble to the rescue!
  - Module decl and all imports are before anything else
  - Batch imports at end of preamble
  - Asks an oracle for Bob, Stuart & Kevin
  - Waits for all responses before continuing

A Client/Server Protocol

- Interface between a build instance and its compilations
  - Not a general purpose compile server
  - Mechanism not policy
  - Defaults so the simple case Just Work
- Human readable text-based query/response
- Batchable to reduce round trips

ATOM Legacy Modules

- `import “legacy-header.h”;`
  - Requires human to determine moduleness
- In legacy header mode:
  - `#include “other-header.h”`
  - Turn it into ‘`import “other-header.h”;`’?
    - → INCLUDE “other-header.h” “current-file”
    - ← INCLUDE [“path/to/other-header.h”]
    - ← IMPORT [“legacy-module-name”]
    - ← SEARCH → give me an absolute path
  - Perhaps build can cache include locations?

Internals

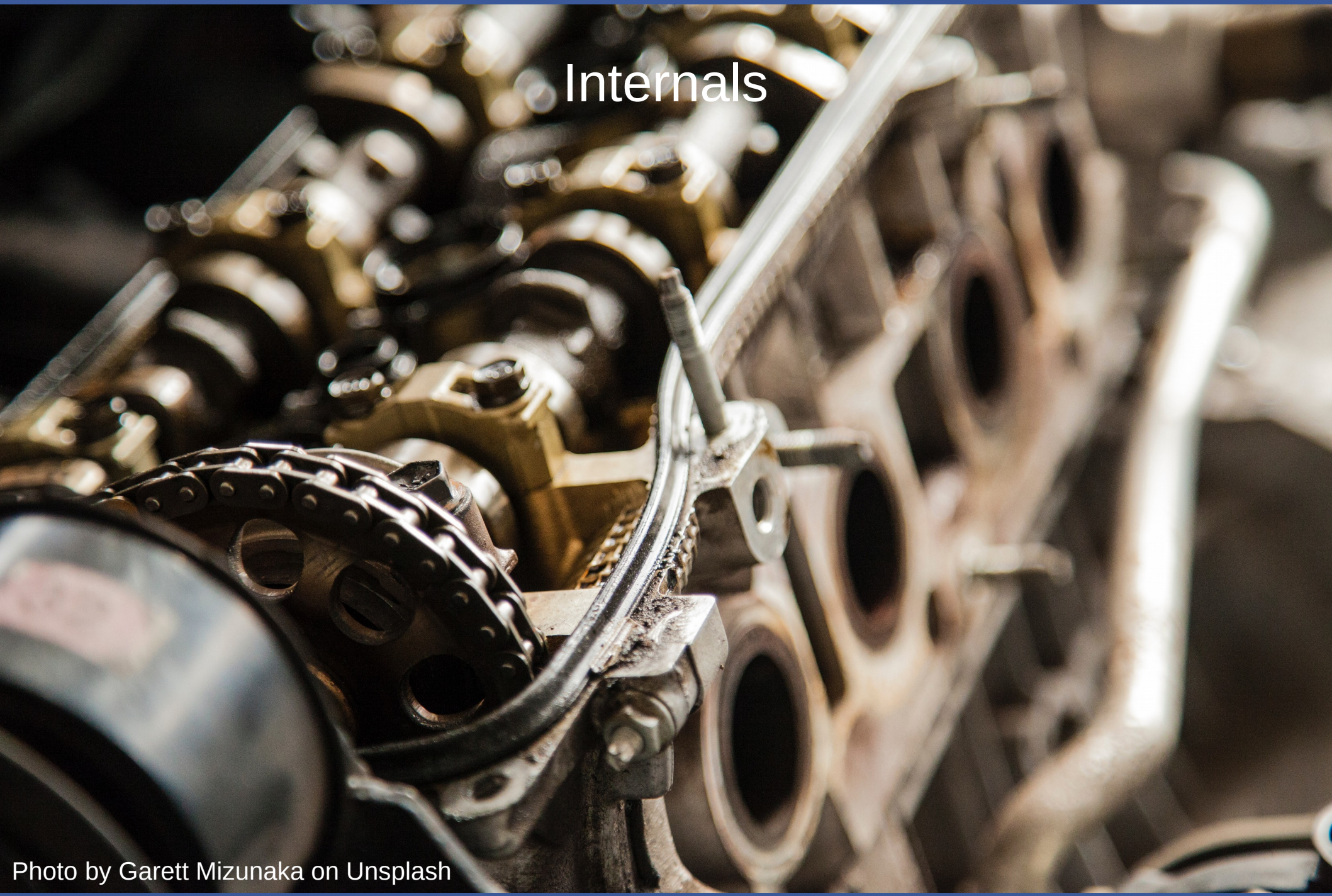


Photo by Garett Mizunaka on Unsplash

BMI Format

- Tool Reuse

```
[ 0]                        NULL
[ 1] gnu.c++.README         STRTAB
[ 2] ::counter              PROGBITS
[ 3] ::hwm{}                PROGBITS
[ 4] ::get{}                PROGBITS
[ 5] gnu.c++.nms            PROGBITS
[ 6] gnu.c++.bnd            PROGBITS
[ 7] gnu.c++.vld            PROGBITS
[ 8] gnu.c++.elo            PROGBITS
[ 9] gnu.c++.llo            PROGBITS
[10] gnu.c++.cfg            PROGBITS
[11] .strtab                STRTAB
```

- Mmapped lazy reading and writing

Introspection

- `readelf -p gnu.c++.README PiL.nms`

```
String dump of section 'gnu.c++.README':
  [     4]  C++ Module (ATOM)
  [    16]  compiler:9.0.0 20180808
(experimental) c++-modules:20180808-1155
  [    57]  version:2018/08/08-11:55
  [    70]  module:PiL
  [    7b]  source:unnamed-1_a.C
  [    90]  options:
```

- `objdump -s -j .gnu.g++.cfg bob.nms`

```
0000 4d308605 0b833794 f9db46c7 00424646
```

Lazy Loading

- Cross-module references by qualified name
  - Intra-module references by qualified name too
- Granularity is namespace binding
  - Except ...

    ```
    int foo (int);
    int bar (int = foo (1));
    int foo (int = bar (1));
    ```
- Partition graph into Strongly Connected Clusters
  - Tarjan’s Algorithm
- Name lookup loads on demand
  - Place cookie in binding slot

Entities That Cannot Be Named

- export auto foo () { struct X {}; return X (); }
- export auto lambda () { return [] () {}; }
- import foo;
  export auto indirect () { return foo (); }
- Find during partitioning walk
  - Build array, use index