Modular Codegen

Further Benefits of Explicit Modularization

Module Flavours

Motivating Example

```
#ifndef FOO_H
#define FOO_H
inline void foo() { ... }
#endif
```

```
#include "foo.h"
void bar() {
   foo();
}
```

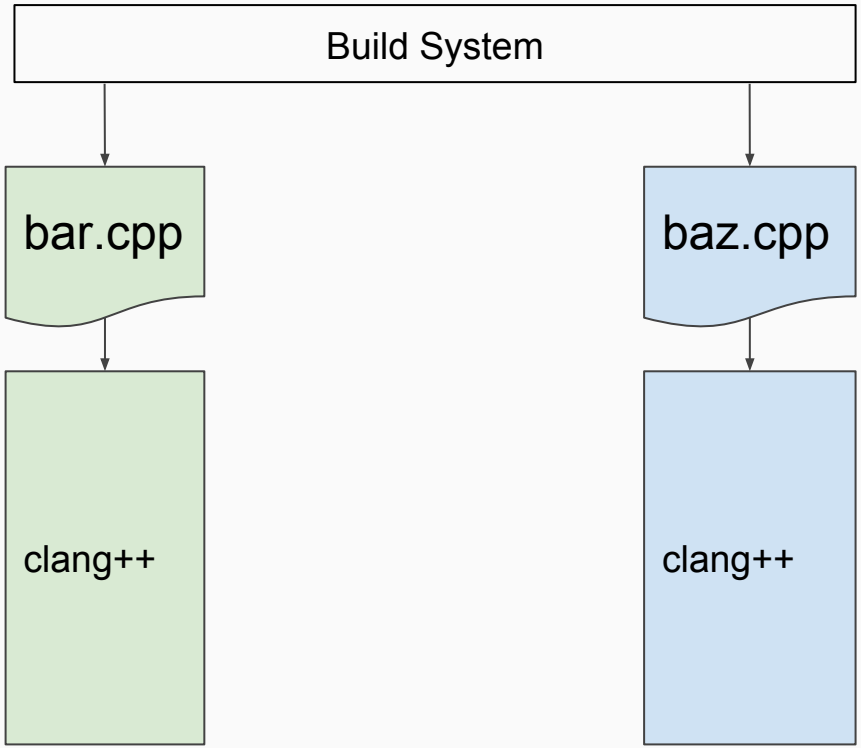
```
#include "foo.h"
void baz() {
   foo();
}
```

Implicit Modules

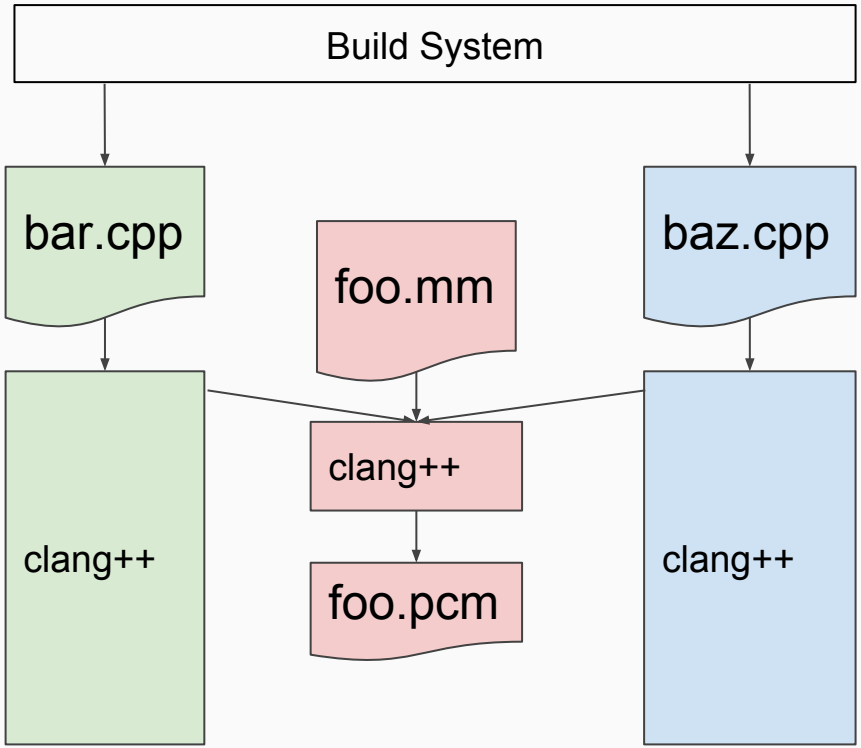
- User writes .modulemap files

```
module foo {
  header "foo.h"
  export *
}
```

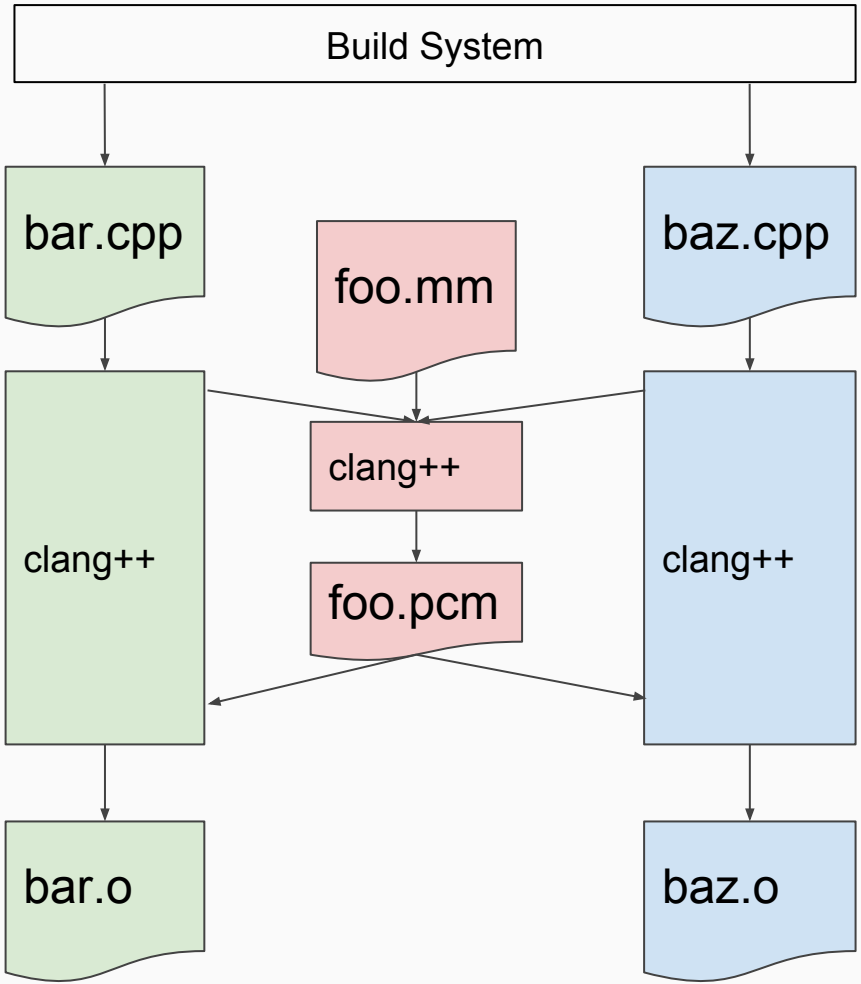
Implicit Modules Build Process



Implicit Modules Build Process



Implicit Modules Build Process



Implicit Modules

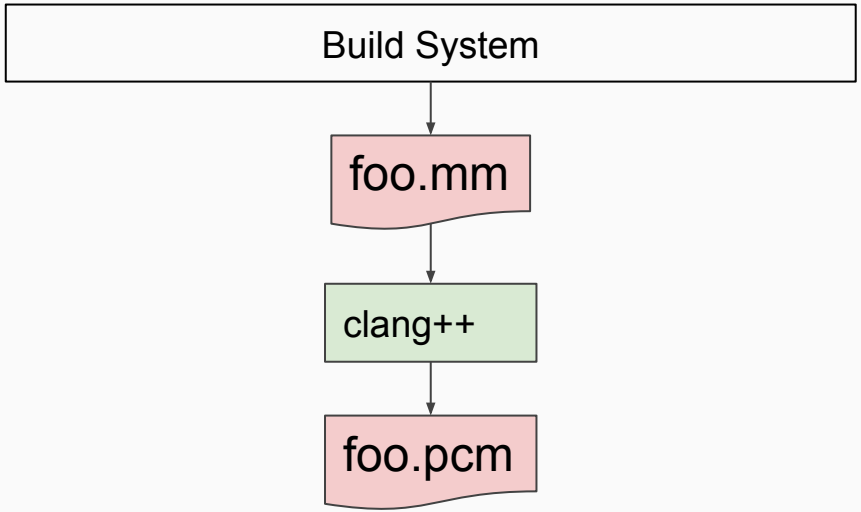
- User writes .modulemap files
- Compiler finds them and implicitly builds module descriptions in a filesystem cache
- Build system agnostic

- Difficult to parallelize - build system isn’t aware of the dependencies
- Doesn’t distribute (clang doesn’t know about distribution scheme)

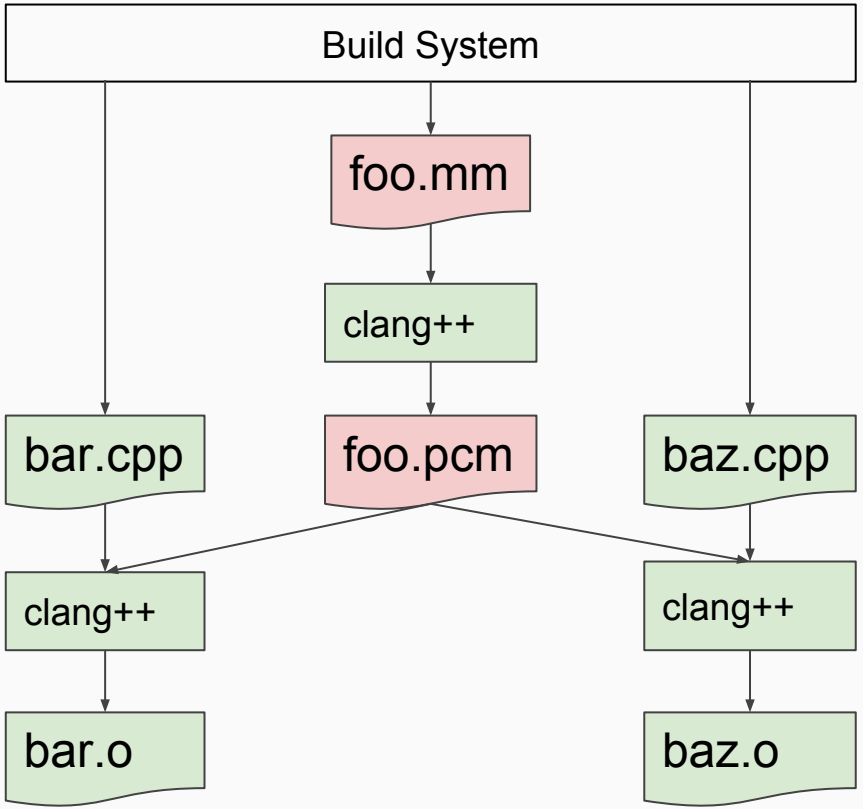
Explicit Modules

- Build system explicitly invokes the compiler on .modulemap files
- Passes resulting .pcm files when compiling .cpp files for use

Explicit Modules Build Process



Explicit Modules Build Process



Modules TS (Technical Specification)

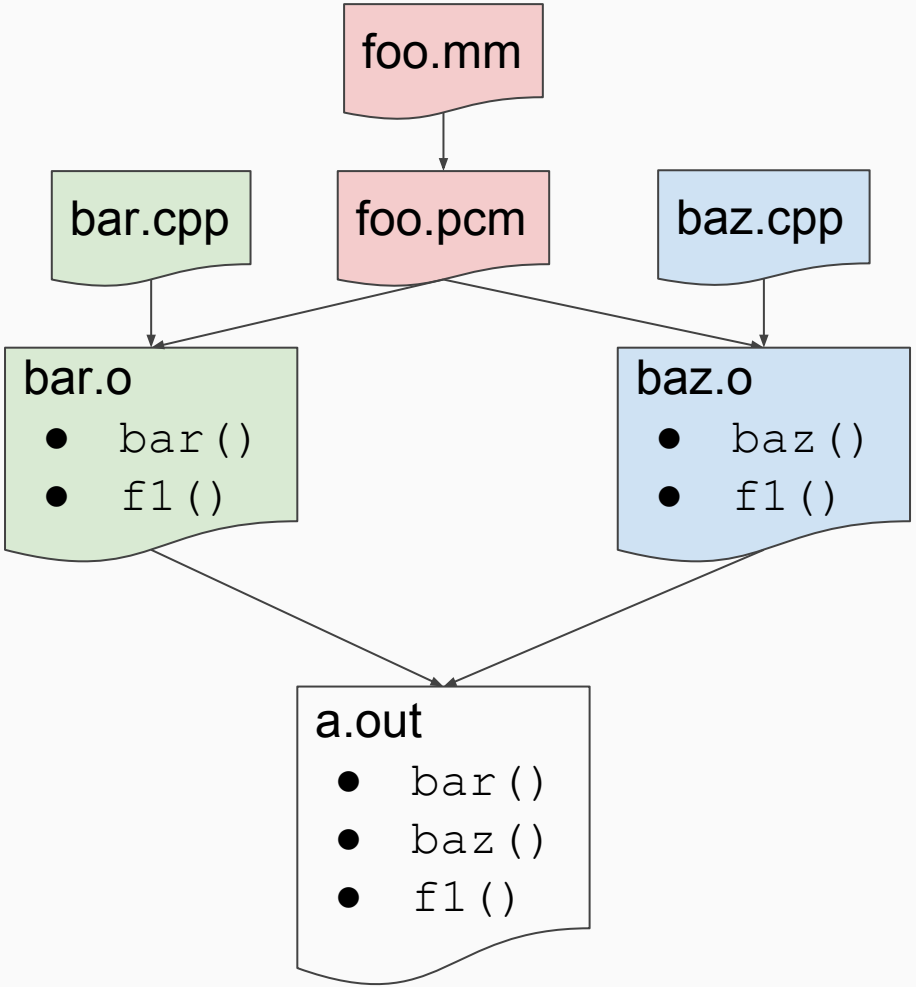
- New file type (C++ with some new syntax - .cppm?)
- New import syntax
- Also needs build system support

Modular Codegen

Duplication in Object Files

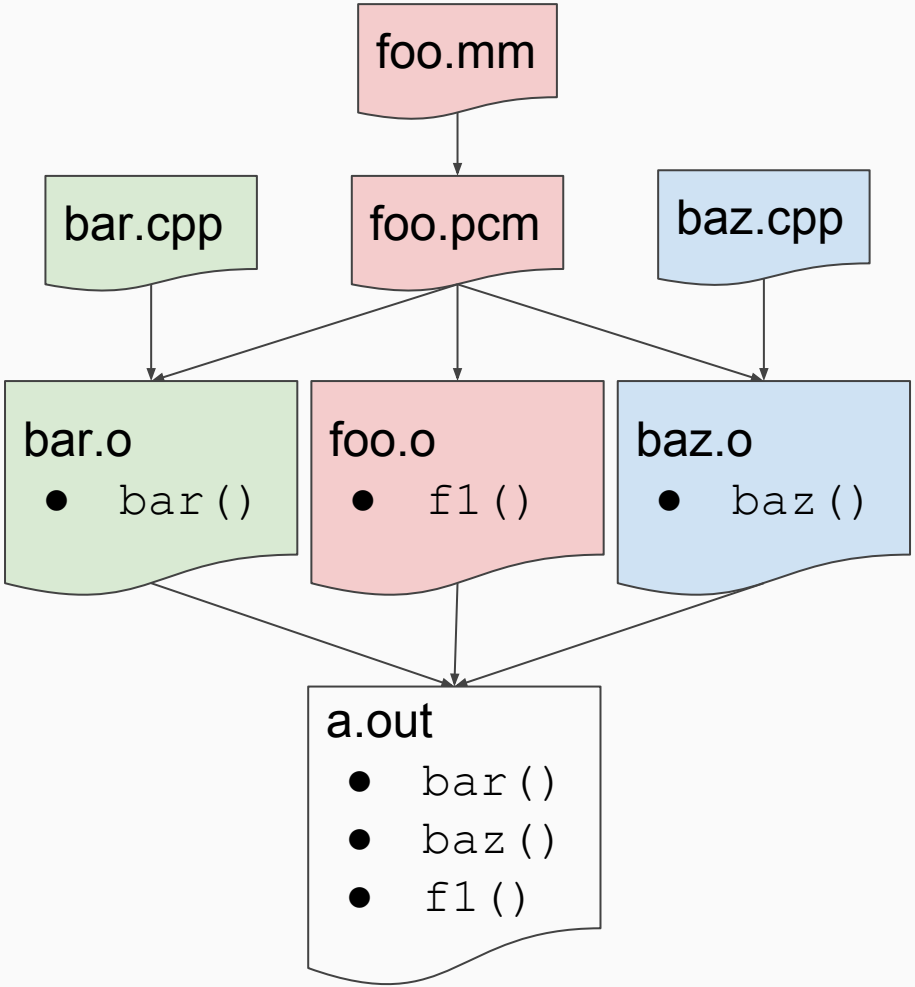
Each object file contains independent definitions of:

- Uninlined ‘inline’ functions (& some other bits)
- Debug information descriptions of classes



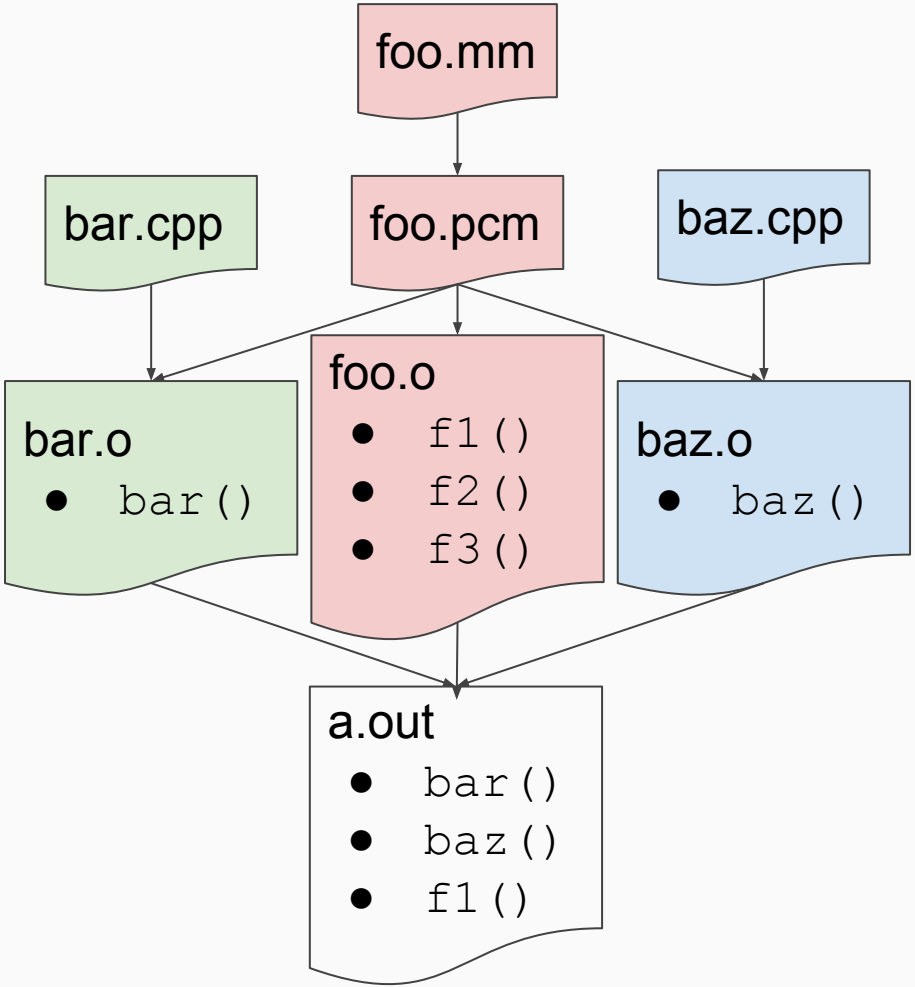
Modular Objects

The module can be used as a ‘home’ for these entities so they don’t need to be carried by every user.



Risks

Unused entities may increase linker inputs.



Constraints

- Headers are compiled separately (& only once) from uses
- Dependencies must be well formed
  - Headers cannot be implemented by a different library - they form circular dependencies no longer broken by duplicated definitions at every use.

Diversion: ‘How Unix Linkers Work (lite)’

```
void a1() { b(); }
void a2() { … }
```

```
void b() { a2(); }
```

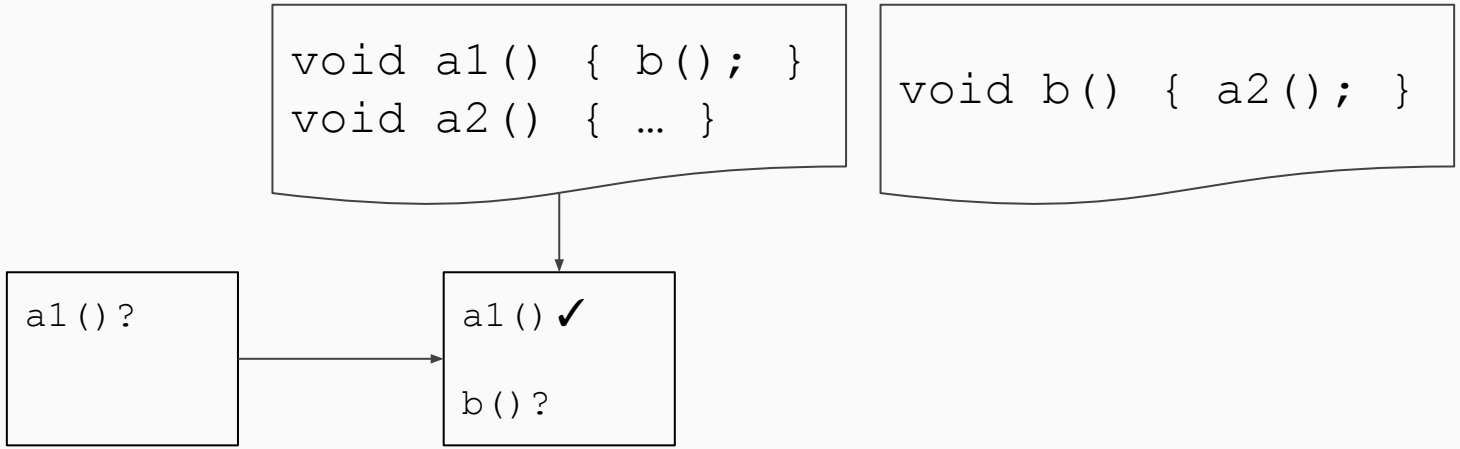
Diversion: ‘How Unix Linkers Work (lite)’

```
void a1() { b(); }
void a2() { … }
```

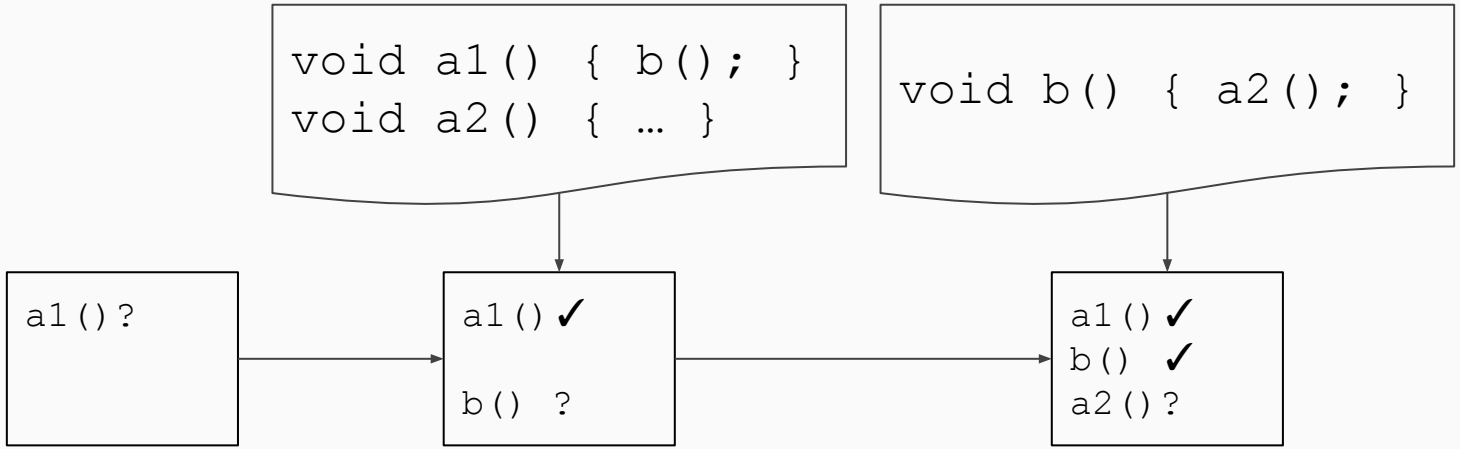
```
void b() { a2(); }
```

```
a1()?
```

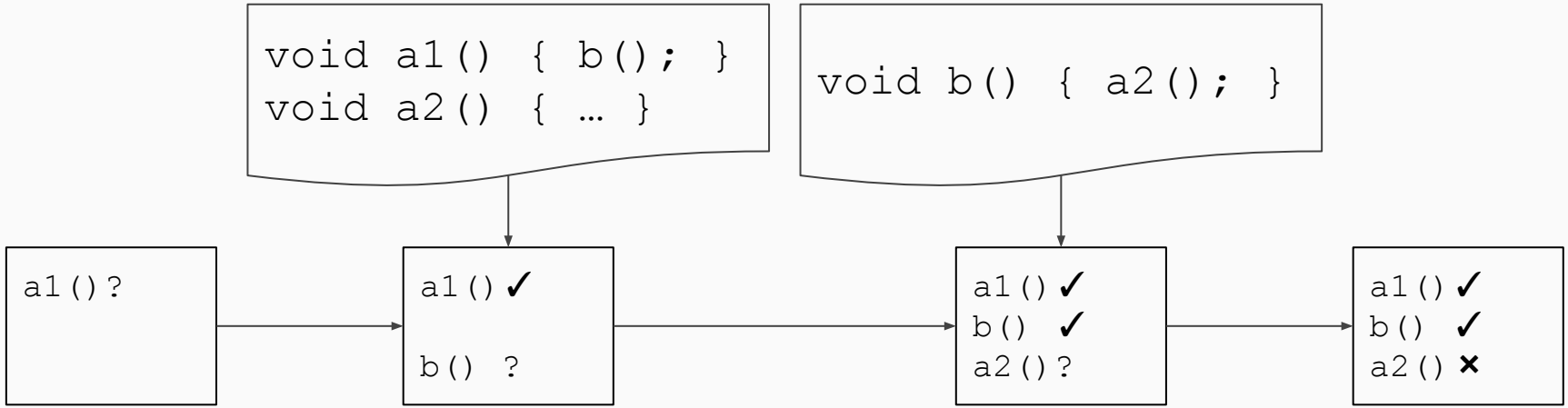
Diversion: ‘How Unix Linkers Work (lite)’



Diversion: ‘How Unix Linkers Work (lite)’



Diversion: ‘How Unix Linkers Work (lite)’

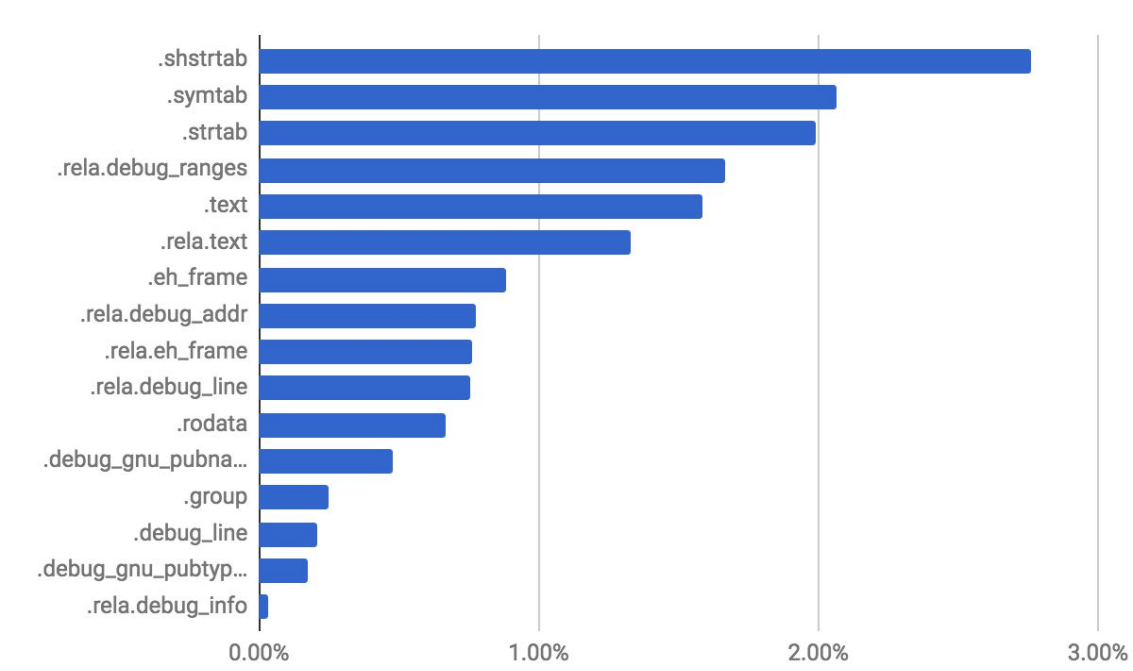


Clang/LLVM Codebase

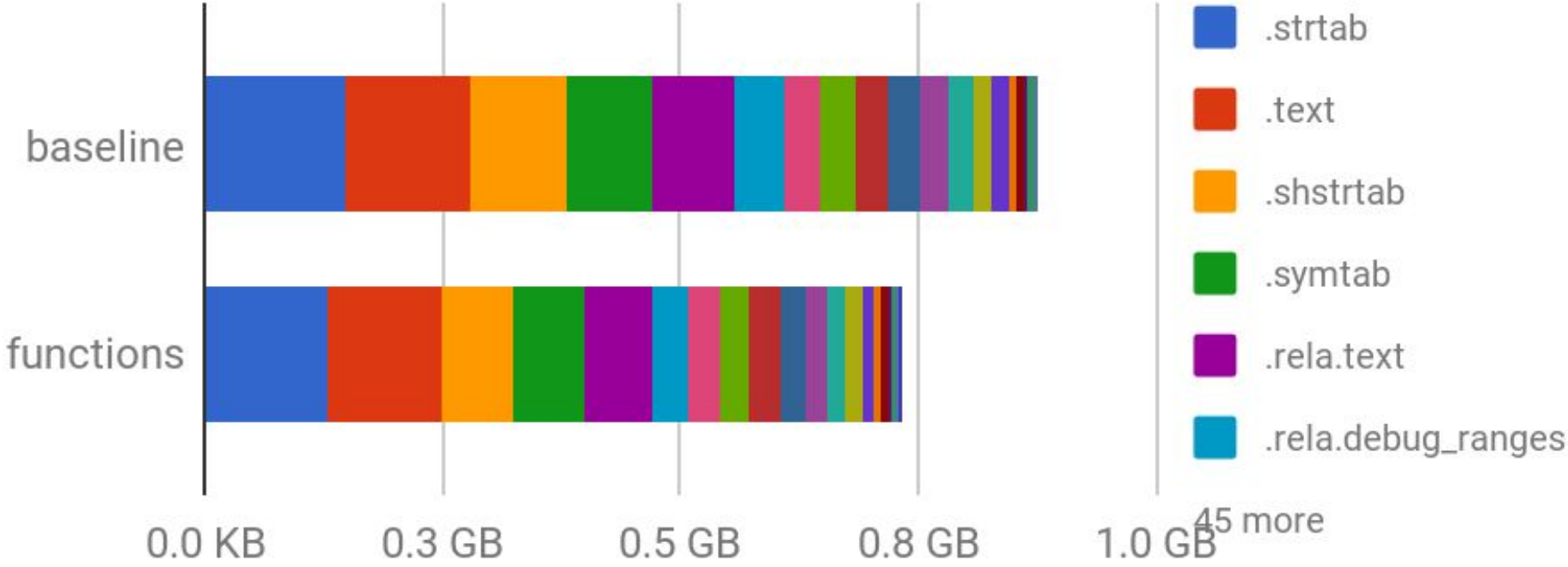
- *.def files are textual/non-modular
- lib/Support/regc* are non-modular
- MCTargetOptionsCommandFlags.h non-modular
- CommandFlags.h non-modular
- Target ASM Parsers depend on MC Target Description
- **static namespace-scope functions in headers -> inline, non-static**
- Missing #includes
- No idea what to do with abi-breaking.h
- Weird things in Hexagon (non-modular headers that are included exactly once…)
- ASTMatchers defining global variables in headers… no idea how this isn't causing link errors, maybe they've got implicit internal linkage.

Results

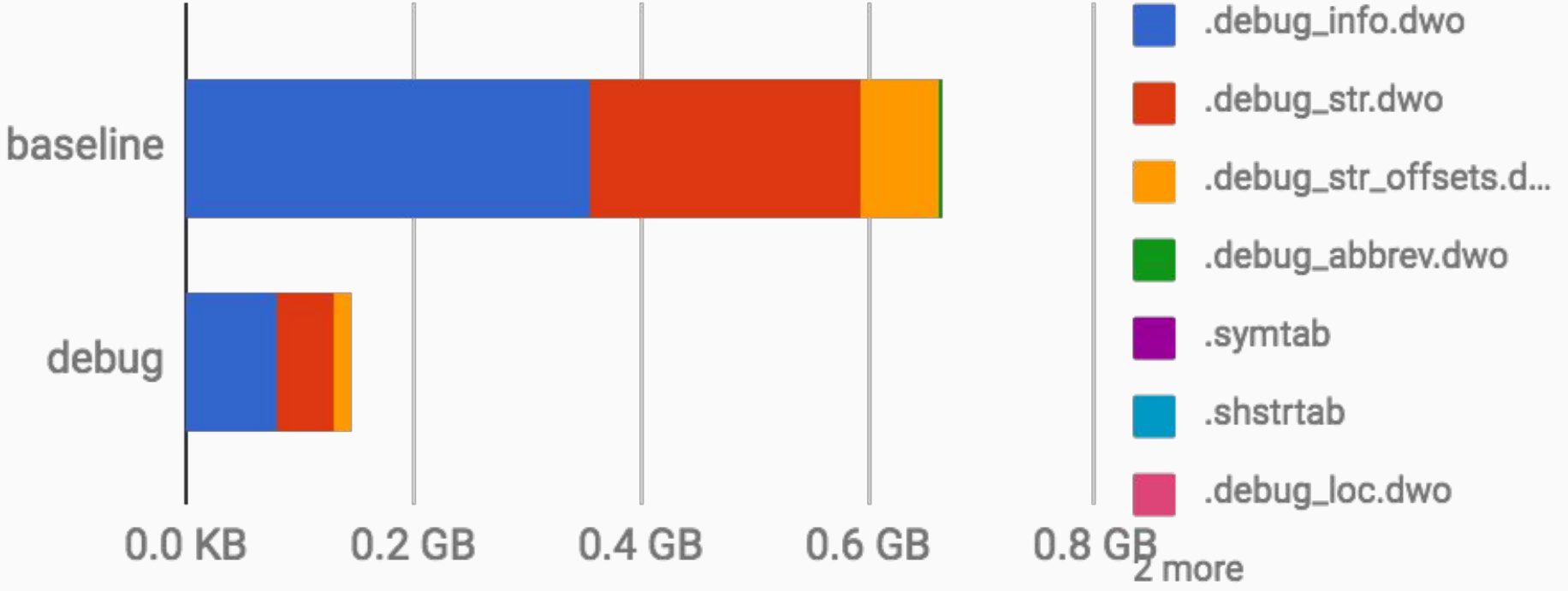
Object Section Sizes
-O0 -fmodules-codegen -gsplit-dwarf



Object Section Sizes
-O0 -fmodules-codegen -gsplit-dwarf

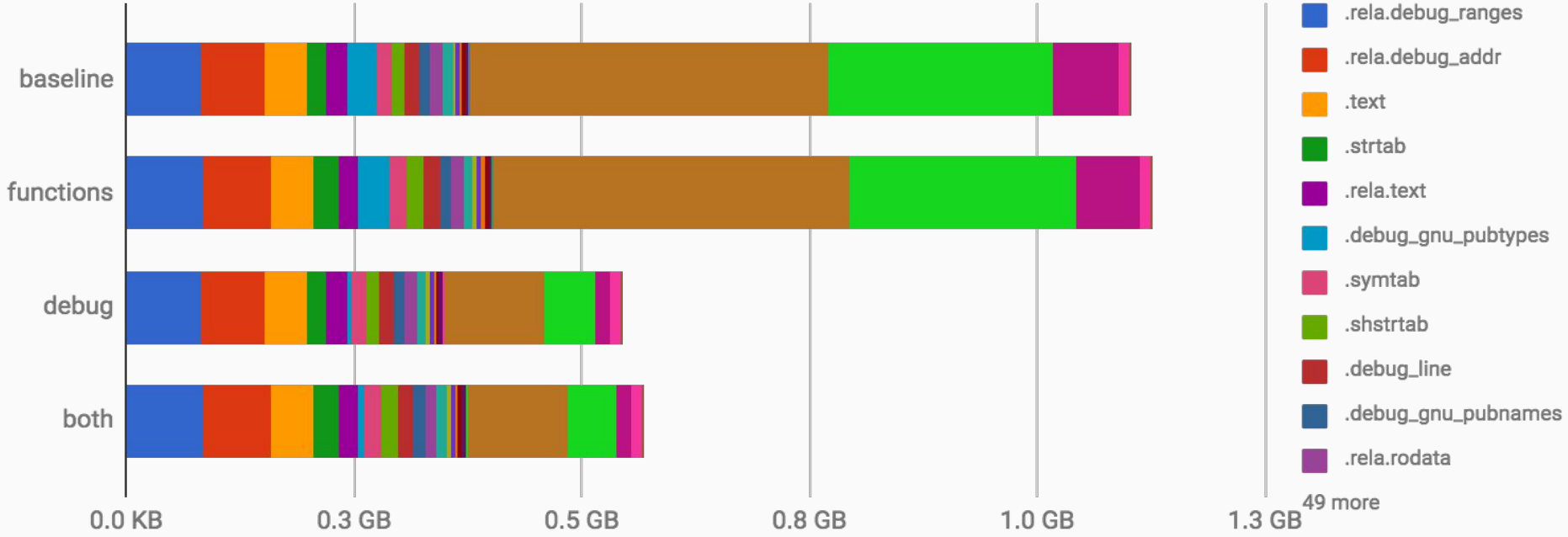


Object Section Sizes
-O0 -fmodules-codegen -gsplit-dwarf



Object Section Sizes
-O0 -fmodules-codegen -gsplit-dwarf

-O3


baseline
functions
debug
both
0.0 KB
0.3 GB
0.5 GB
0.8 GB
1.0 GB
1.3 GB
.rela.debug_ranges
.rela.debug_addr
.text
.strtab
.rela.text
.debug_gnu_pubtypes
.symtab
.shstrtab
.debug_line
.debug_gnu_pubnames
.rela.rodata
49 more

Further Work

- Other aspects needed for Modules TS
  - Variables (implemented - could be backported to non-TS style, may not be needed)
  - ???
- Avoid homing `alwaysinline` functions (maybe other reasonable inlining heuristics to avoid homing functions unlikely to remain uninlined)
- Avoid type units when a home is likely to be unique (not an implicit template instantiation, or has a strong vtable, etc)

Thanks!

David Blaikie

Email/etc: dblaikie@gmail.com
Twitter: @dwblaikie

20%

Use this slide to show a major stat. It can help enforce the presentation’s main message or argument.

This is the most important takeaway that everyone has to remember.

Final point

A one-line description of it

“This is a super-important quote”

- From an expert