Improving the Efficiency and Correctness of Implicit Modules

Connor Sughrue

Clang modules compiler extension

```
commit e703f2dc30720cc298e2583cabc99815fe9297f3
Author: Douglas Gregor <dgregor@apple.com>
Date:   Wed Mar 20 06:25:14 2013 +0000

    Work-in-progress documentation on the experimental modules feature.

    llvm-svn: 177491
```

Standard C++ Modules

10 Modules [module]

10.1 Module units and purviews [module.unit]

module-declaration:
    export-keyword$_{opt}$ *module-keyword* *module-name* *module-partition*$_{opt}$ *attribute-specifier-seq*$_{opt}$;

module-name:
    module-name-qualifier$_{opt}$ *identifier*

module-partition:
    : *module-name-qualifier*$_{opt}$ *identifier*

module-name-qualifier:
    identifier .
    module-name-qualifier identifier .

Clang Modules*

```
//--- circle.h
#pragma once
double getCircleArea(double r) {
  return 3.14 * r * r;
}
```



```
//--- tu_1.cpp
#include "circle.h"
int main() {
  double area = getCircleArea(5.0);
  return 0;
}
```

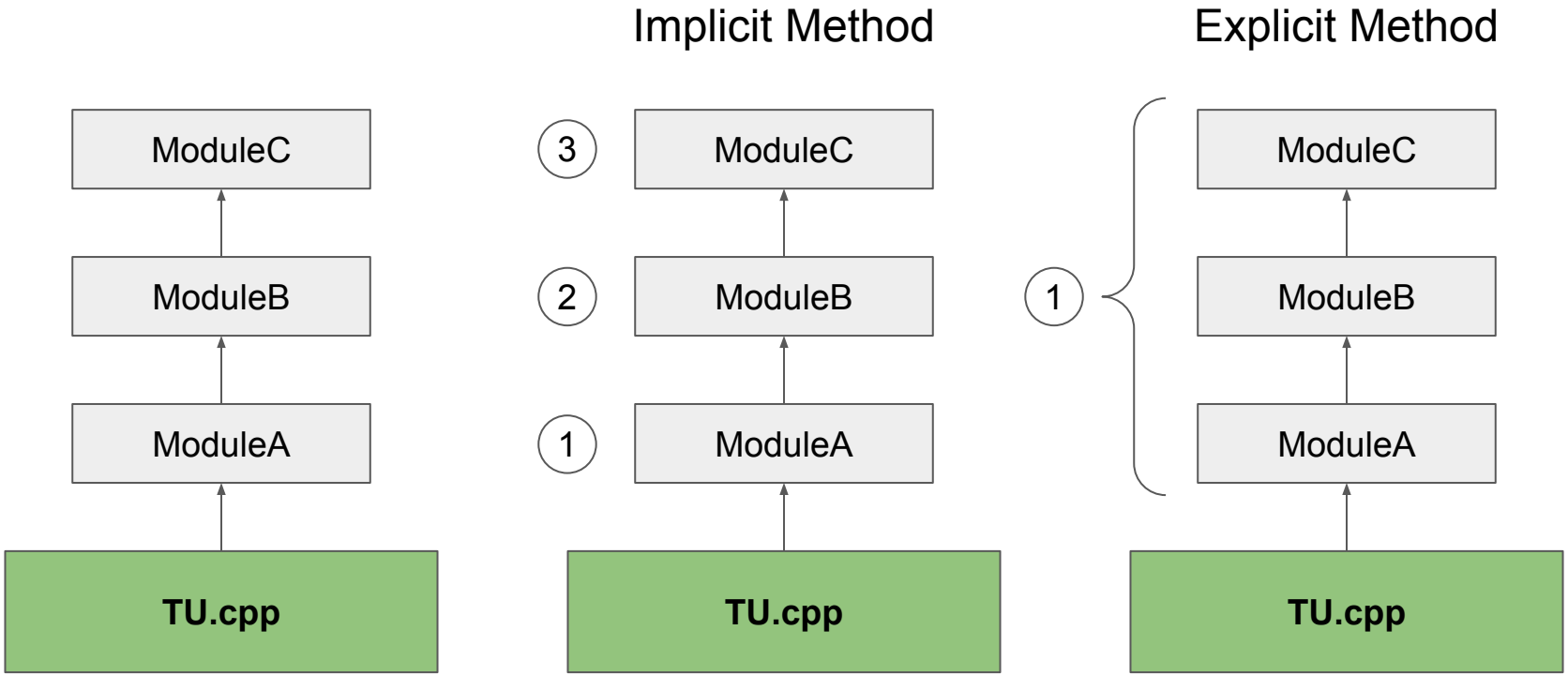
Standard C++ Modules

```
//--- circle.cppm
export module circle;
export double getCircleArea(double r) {
  return 3.14 * r * r;
}
```



```
//--- tu_2.cpp
import circle
int main() {
  double area = getCircleArea(5.0);
  return 0;
}
```

*In the clang module example the module map is left out


Implicit Method
Explicit Method
ModuleC
ModuleB
ModuleA
TU.cpp
3
2
1
ModuleC
ModuleB
ModuleA
TU.cpp
1
ModuleC
ModuleB
ModuleA
TU.cpp

Module Variants

Hash used to determine when two translation units can use the same module variant

- Ex) `cache/2OMZ3GMQI3909/ModuleA.pcm`
- Module adopts command line of predecessor

Implicit Context Hash

- Hash of a subset of a compiler invocation

Explicit Hash

- Hash of the canonical arguments to build the module

Issues

Implicit Method Issues

Incomplete implicit context hash

```
//--- tu.cpp
#include "VectorUtils.h"
```

```
//--- VectorUtils.h
#include "Utils.h"

//--- module.modulemap
module VectorUtils {
  header "VectorUtils.h"
}
```

ARM

```
//--- Utils.h
int A() {...}
```

x86-64

```
//--- Utils.h
int A() {...}
```

TU ARM: clang++ -c tu.cpp -o tu.o ~~-I ARM~~ ← Ignored when calculating context hash

Saves `VectorUtils` built with `ARM/Utils` to `cache/HASH1/VectorUtils.pcm`

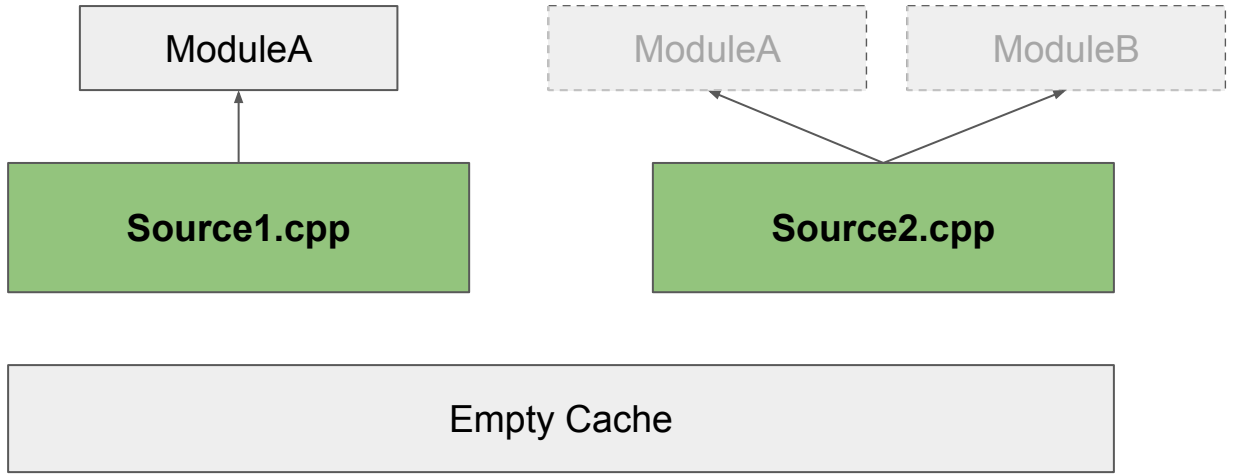
TU x86-64: clang++ -c tu.cpp -o tu.o ~~-I x86-64~~ ← Ignored when calculating context hash

Will reuse `cache/HASH1/VectorUtils.pcm`

Implicit Method Issues

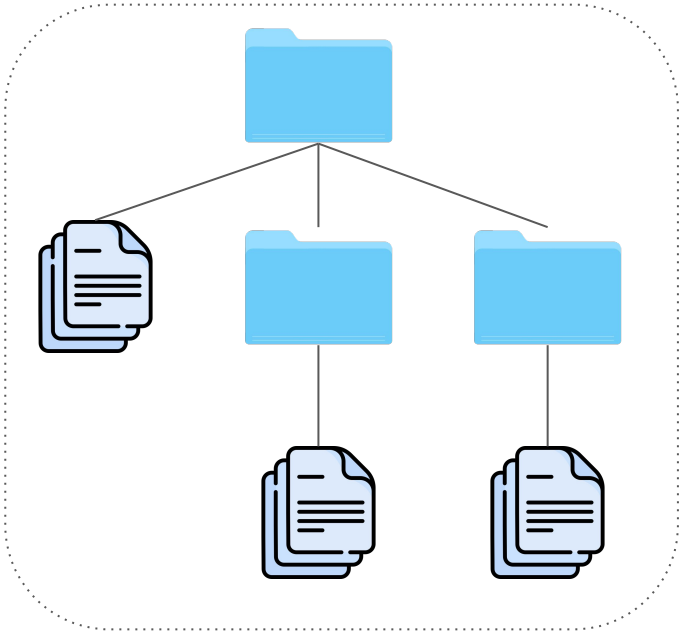
Inefficient builds

TUs are compiled in parallel



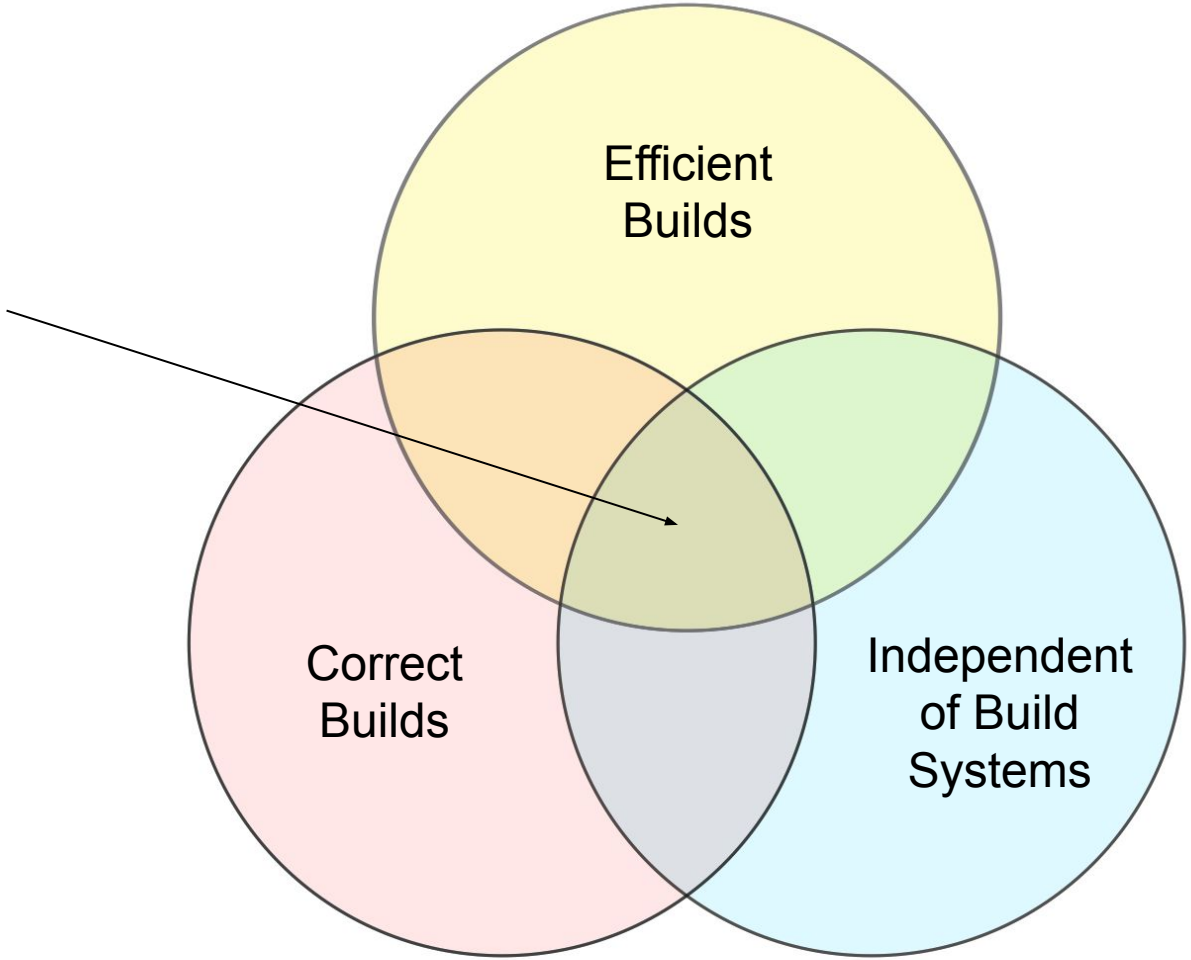
Explicit Method Issues

Dependent on build systems

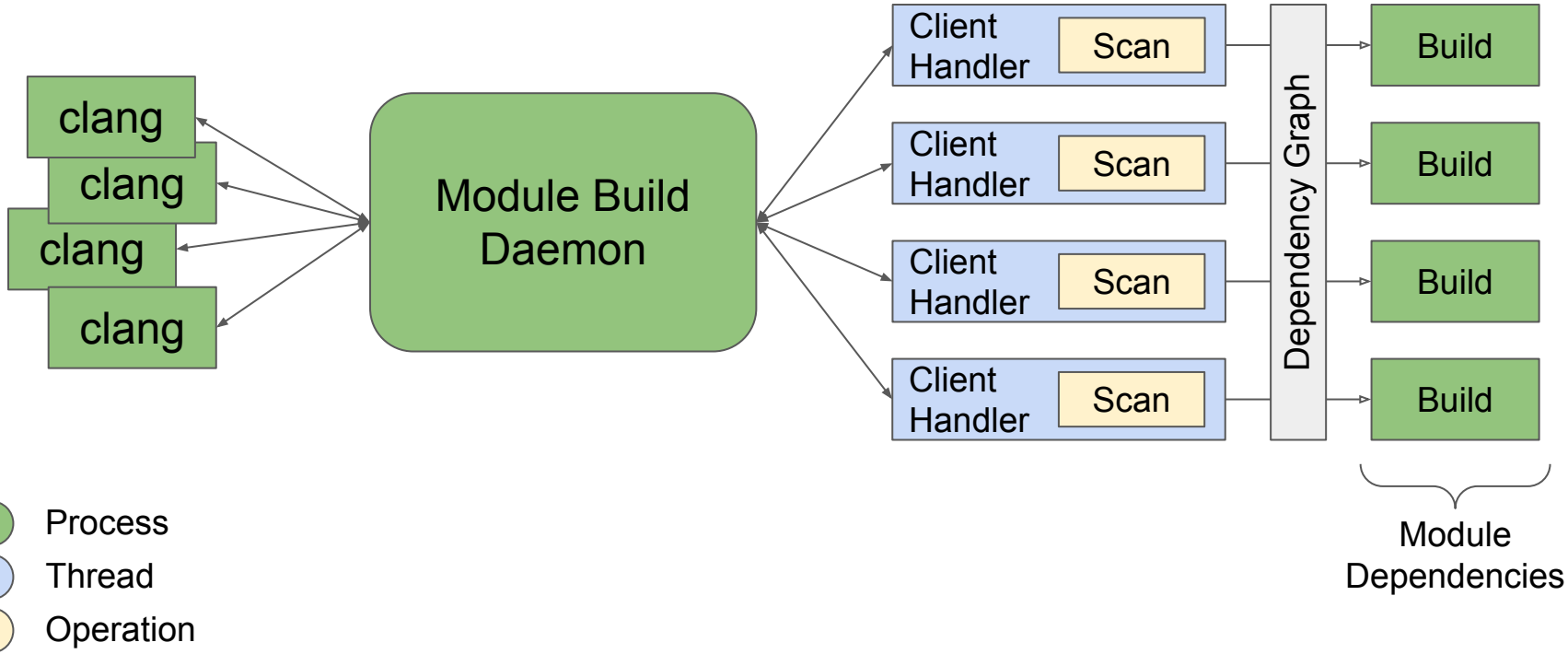


*CMake support for Standard C++ modules is non-experimental beginning in version 3.28

Goals


Efficient
Builds
Correct
Builds
Independent
of Build
Systems

Module Build Daemon



Enable Module Build Daemon

- Minimal additional set up
- Single driver command line argument

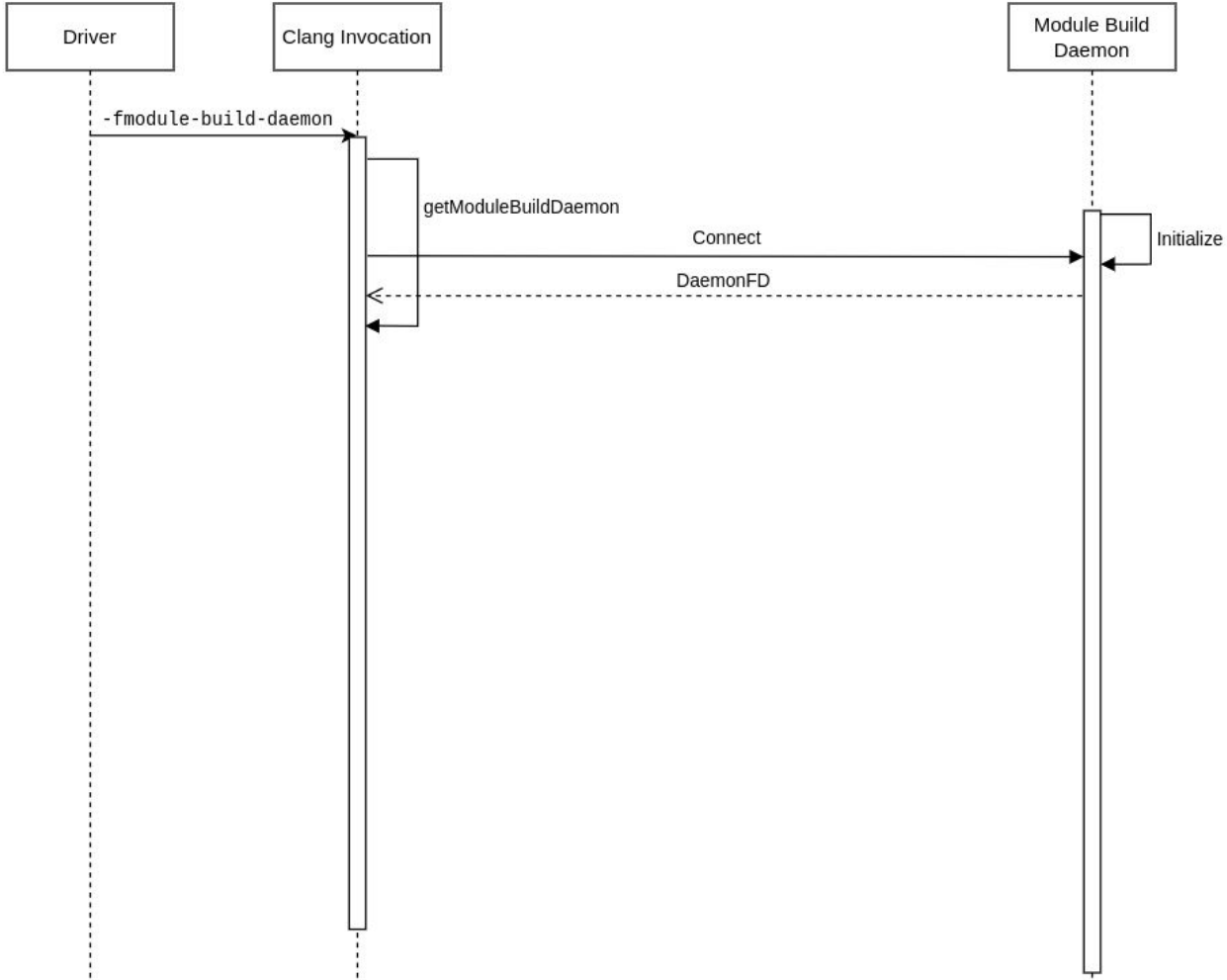
```
clang++ -fmodule-build-daemon -fmodules ... main.cpp
```

Control Flow of Clang Invocation

- Driver forwards `-fmodule-build-daemon` to frontend

```
clang++ -cc1 -fmodule-build-daemon -fmodules ... main.cpp
```

- Right before `ExecuteCompilerInvocation` the frontend connects with the module build daemon


Driver
Clang Invocation
Module Build Daemon
-fmodule-build-daemon
getModuleBuildDaemon
Connect
Initialize
DaemonFD

Getting a Module Build Daemon

- Clang invocation tries to connect to the daemon
- If it does not exist launch the daemon with `llvm::sys::ExecuteNoWait`
- Periodically check if daemon has initialized until timeout or connection

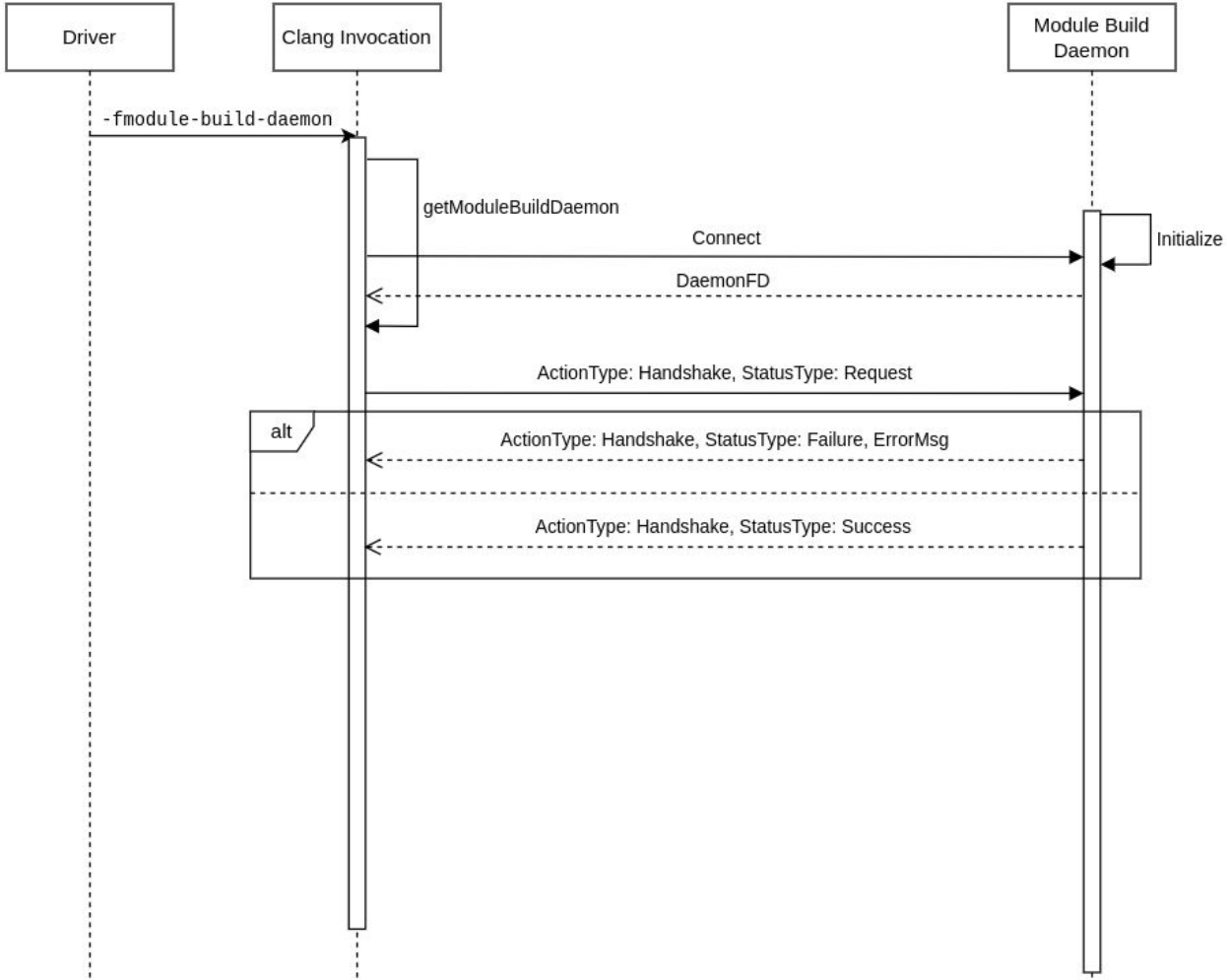
Control Flow of Module Build Daemon

`clang -cc1modbuildd <BasePath>`

1. `ModuleBuildDaemonServer Daemon(BasePath);`
2. `Daemon.forkDaemon()`
3. `Daemon.createDaemonSocket()`
4. `Daemon.listenForClients()`

Daemon BasePath

- Daemon is specific to clang version and is based on the `BasePath`
  - `BasePath = /tmp/BLAKE3HashOfClangFullVersion`
  - `Socket = BasePath/mbd.sock`
  - `STDOUT = BasePath/mbd.out`
  - `STDERR = BasePath/mbd.err`


Driver
Clang Invocation
Module Build Daemon
-fmodule-build-daemon
getModuleBuildDaemon
Connect
Initialize
DaemonFD
ActionType: Handshake, StatusType: Request
alt
ActionType: Handshake, StatusType: Failure, ErrorMsg
ActionType: Handshake, StatusType: Success

Messaging Protocol

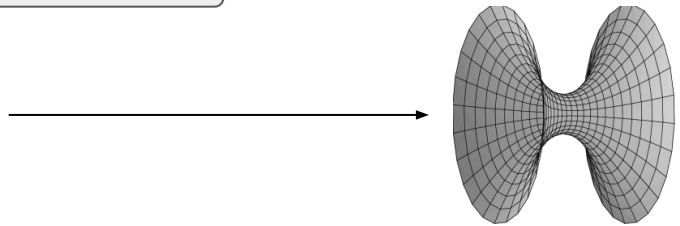
- Use YAML I/O
  - LLVM library
  - Translate structs to strings and strings to structs
- Defined by structs

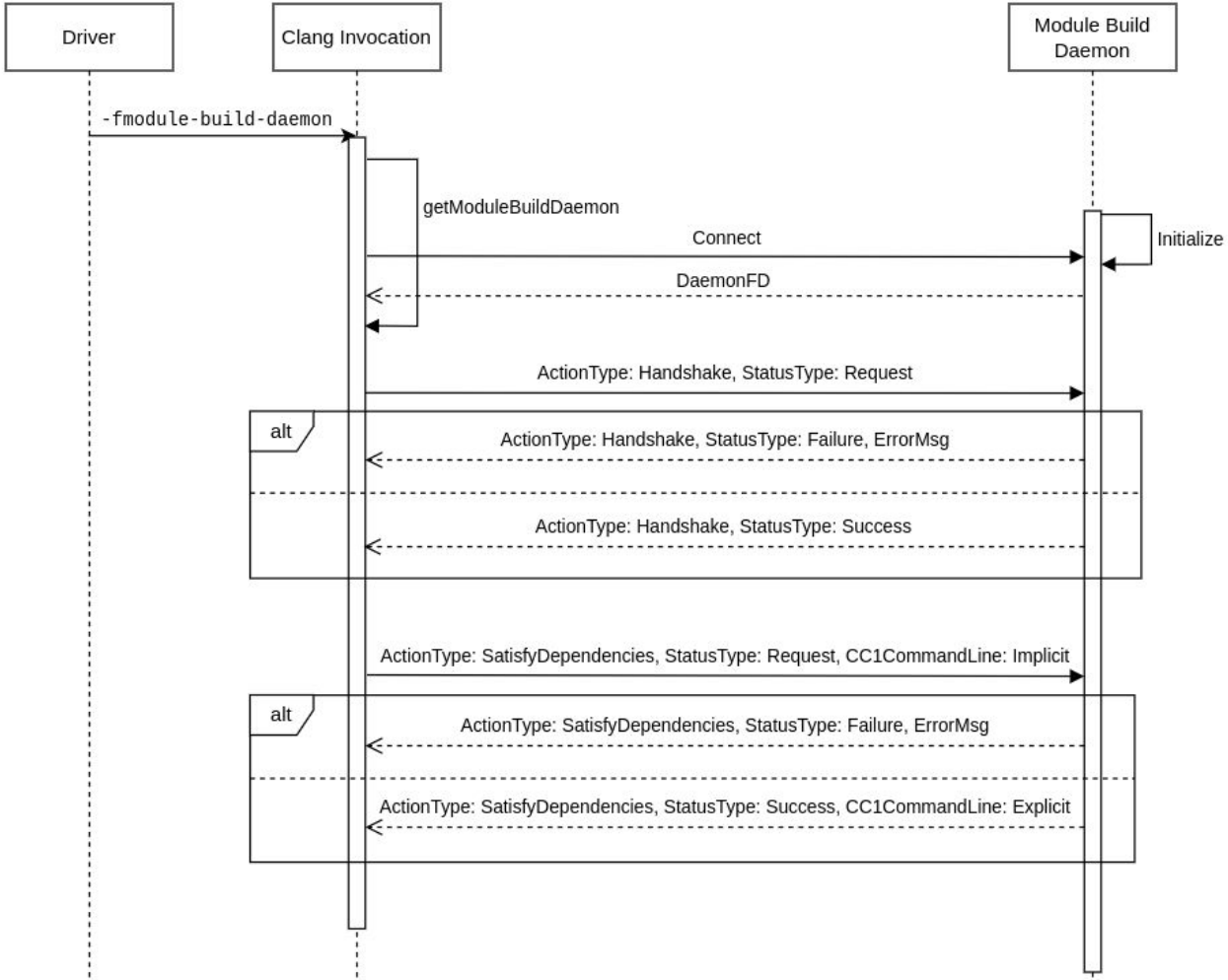
Messaging Protocol

```
enum class ActionType { Handshake, SatisfyDependencies };
enum class StatusType { Request, Success, Failure };

struct HandshakeMsg {
  ActionType MsgAction;
  StatusType MsgStatus;

  HandshakeMsg(ActionType Action, StatusType Status)
      : MsgAction(Action), MsgStatus(Status)  {}
};
```

```
HandshakeMsg Request{ActionType::Handshake, StatusType::Request};
```

```
"---\nAction:  HANDSHAKE\nStatus:  REQUEST\n...\n"
```




Driver
Clang Invocation
Module Build Daemon
-fmodule-build-daemon
getModuleBuildDaemon
Connect
Initialize
DaemonFD
ActionType: Handshake, StatusType: Request
alt
ActionType: Handshake, StatusType: Failure, ErrorMsg
ActionType: Handshake, StatusType: Success
ActionType: SatisfyDependencies, StatusType: Request, CC1CommandLine: Implicit
alt
ActionType: SatisfyDependencies, StatusType: Failure, ErrorMsg
ActionType: SatisfyDependencies, StatusType: Success, CC1CommandLine: Explicit

Module Build Daemon Client Handler

Process command line

- Scans translation unit with `getTranslationUnitDependencies`
- Stores dependency info in daemon's dependency graph
  - For each new `ModuleID`
  - Add to global dependency graph

Module Build Daemon Client Handler

- Builds modular dependencies
  - Topological order
  - Recursively iterate through dependency graph

```
processTU():
  for ModuleID in TranslationID.ClangModuleDeps:
    buildModuleID(ModuleID)

buildModuleID(ModuleID):
  for ModuleID in ModuleID.ClangModuleDeps:
    buildModuleID(ModuleID)
  precompileModuleID(ModuleID)

precompileModuleID(ModuleID):
  llvm::sys::ExecuteAndWait()
```

* `ClangModuleDeps` is the list of modules a translation unit directly depends on, not including transitive dependencies.

Module Build Daemon Client Handler

- Get explicit command line from scanner
- Explicit command line gets sent to the clang invocations
- `ExecuteCompilerInvocation` uses new command line

Note:

- Resource management
  - Abide by `-j` limit

Remaining Work

- Finish upstreaming [68498 & 67562]
- Support incremental builds
- Improve portability
- Improve concurrency
- C++20
  - What is the command line to build a module
  - Information will come from outside the module build daemon
  - Whatever the tooling study group (SG15) decides can be supported

References

- RFC: https://discourse.llvm.org/t/rfc-modules-build-daemon-build-system-agnostic-support-for-explicitly-built-modules
- Current pull request under review: https://github.com/llvm/llvm-project/pull/67562
- Complete patch draft: https://github.com/llvm/llvm-project/pull/68498
- CMake support:
  - https://gitlab.kitware.com/cmake/cmake/-/issues/18355
  - https://mathstuf.fedorapeople.org/fortran-modules/fortran-modules.html
- Relevant talks
  - Implicitly discovered, explicitly built Clang modules
    - https://youtu.be/W5kjEeSmCBU?si=uXJmQLIGDe2eG6v5
  - clang-scan-deps: Fast dependency scanning for explicit modules
    - https://youtu.be/Ptr6e4CVTd4?si=Z9xOOt2oDLgRS_Od
- Build Systems
  - https://ndmitchell.com/downloads/paper-build_systems_a_la_carte_theory_and_practice-21_apr_2020.pdf
- Implicit vs Explicit modules
  - https://forums.swift.org/t/explicit-module-builds-the-new-swift-driver-and-swiftpm/36990