LLVM Out-of-tree Passes in Python

Stefan Gränitz // Berlin Systems Group // 05 February 2025

Compiler pass for a niche audience

For example: hack the ABI surface of our applications

```
➜ nm test
0000000100000f80 T _bar
0000000100000f70 T _foo
0000000100000f90 T _main
```

Compiler pass for a niche audience

For example: hack the ABI surface of our applications

```
➜ nm test
0000000100000f80 T _bar
0000000100000f70 T _foo
0000000100000f90 T _main
```

↓ encode function names as SHA256 ↓

```
➜ nm test
0000000100000f70 T _2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae
0000000100000f80 T _fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9
0000000100000f90 T _main
```

Preface

LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

https://llvm.org

Preface

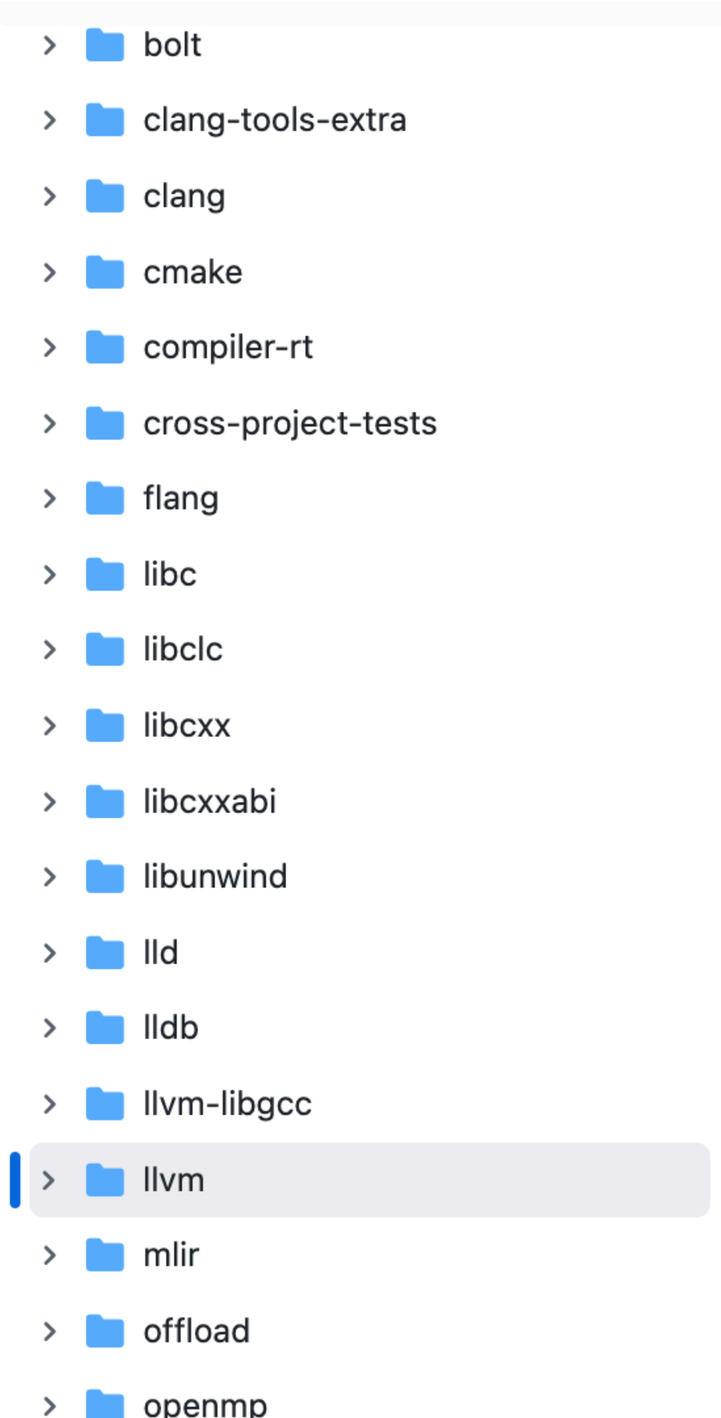
LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

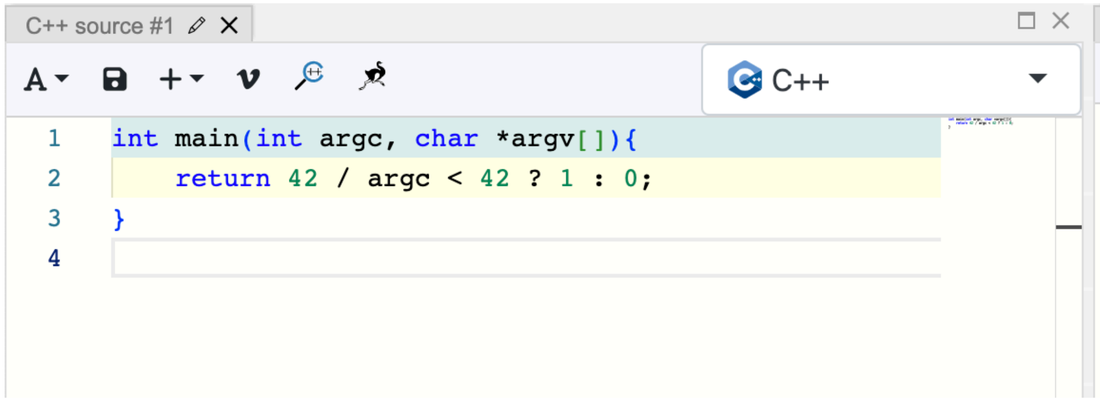
https://llvm.org

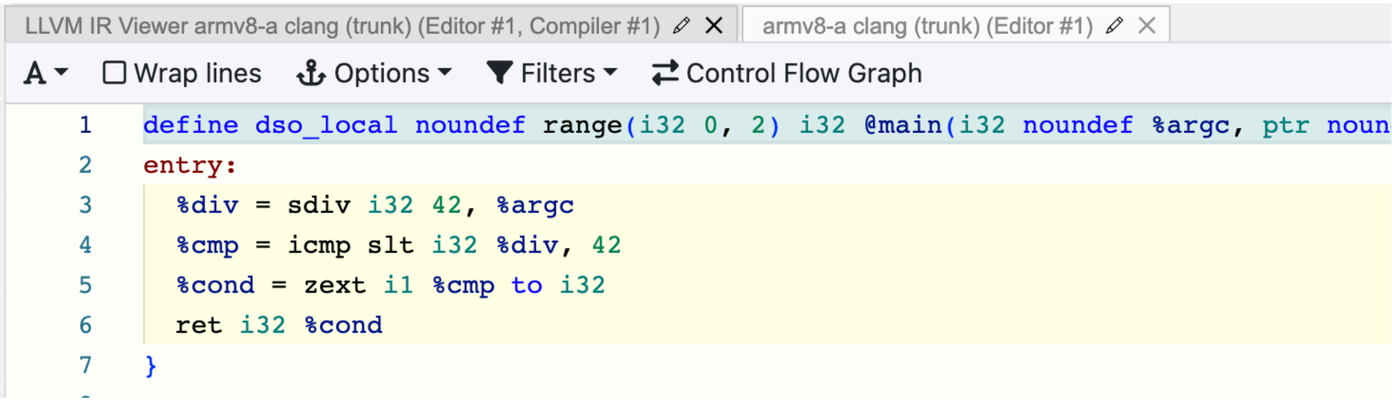
"in-tree"

https://github.com/llvm/llvm-project



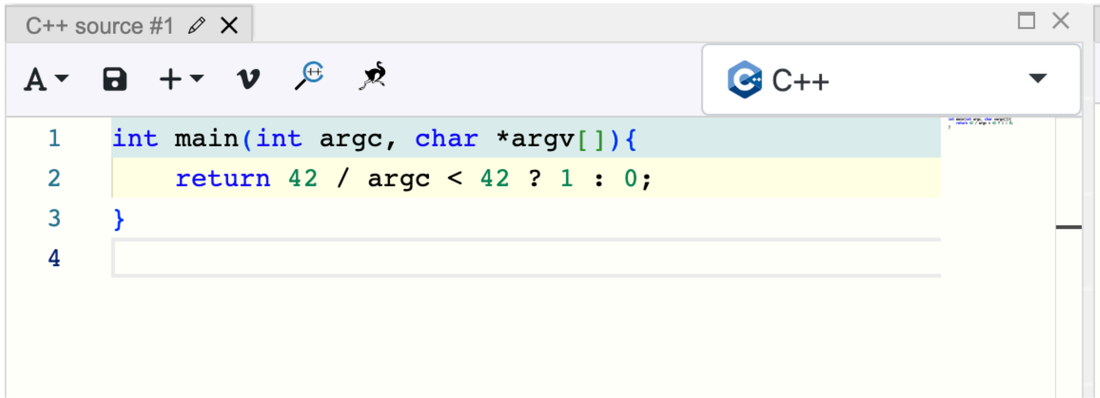
Preface

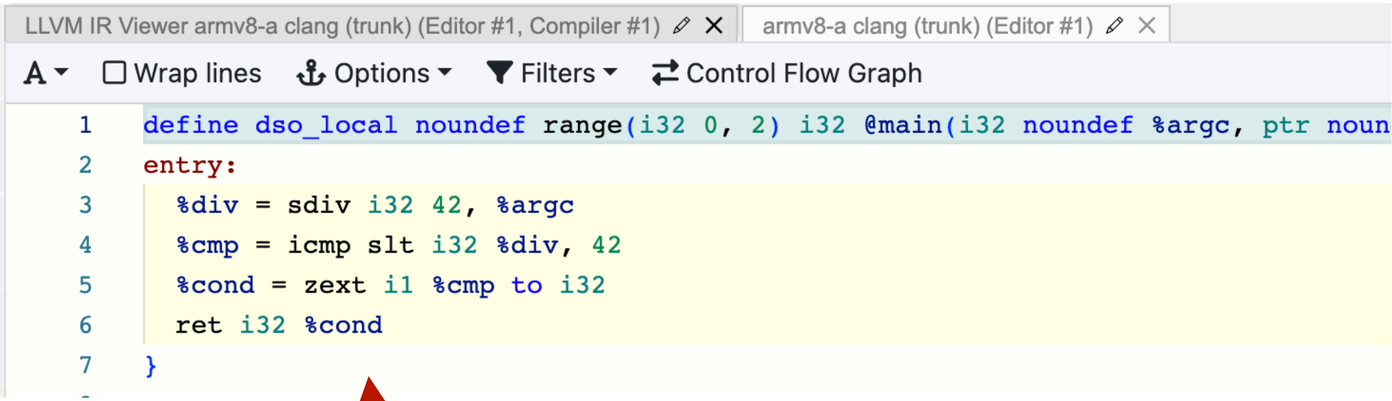




https://godbolt.org/z/Tnh8qvEqb

Preface





https://godbolt.org/z/Tnh8qvEqb

"passes" do transformations

https://llvm.org/docs/Passes.html

Compiler pass for a niche audience

For example: hack the ABI surface of our applications

```
--- original
+++ processed
@@ -4,12 +4,12 @@
 target triple = "x86_64-apple-macosx14.0.0"

 ; Function Attrs: nounwind ssp uwtable
-define i32 @foo() #0 {
+define i32 @"2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae"() #0 {
   ret i32 42
 }

 ; Function Attrs: nounwind ssp uwtable
-define i32 @bar() #0 {
+define i32 @fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9() #0 {
   ret i32 0
 }

@@ -20,11 +20,11 @@
```

Compiler pass for a niche audience

For example: hack the ABI surface of our applications

```
@@ -20,11 +20,11 @@
   br i1 %4, label %5, label %7

 5:                                                ; preds = %2
-  %6 = call i32 @foo()
+  %6 = call i32 @"2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae"()
   br label %9

 7:                                                ; preds = %2
-  %8 = call i32 @bar()
+  %8 = call i32 @fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9()
   br label %9

 9:                                                ; preds = %7, %5
```

Compiler pass for a niche audience

For example: hack the ABI surface of our applications

```
from hashlib import sha256
from llvmlite.binding.common import _encode_string as utf8bytes
from llvmlite.binding.module import ModuleRef

def run(mod: ModuleRef):
    for func in mod.functions:
        if func.name != "main":
            func.name = sha256(utf8bytes(func.name)).hexdigest()
```

"The Golden Age of Compilers in an Era of HW/SW Co-design"

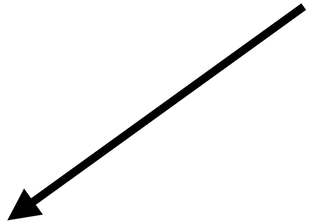
— Chris Lattner, ASPLOS 2021

"The Golden Age of Compilers in an Era of ~~HW/SW Co-design~~"

Fragmentation?

Compiler pass for a niche audience

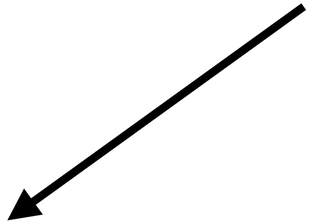
What are the options for distribution?



Get it into the mainline compiler?

Compiler pass for a niche audience

What are the options for distribution?



Get it into the mainline compiler?

This is a niche topic

No mainline maintainer cares about the PR

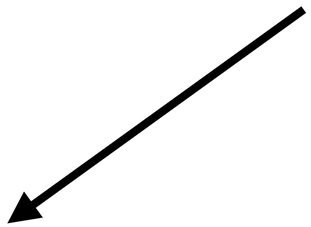
No-one understands the point

Few people will use it, so

why must everyone have it?

Compiler pass for a niche audience

What are the options for distribution?



~~Get it into the machine compiler?~~

Ship your own fork of the compiler?

Compiler pass for a niche audience

What are the options for distribution?

Good luck surviving downstream

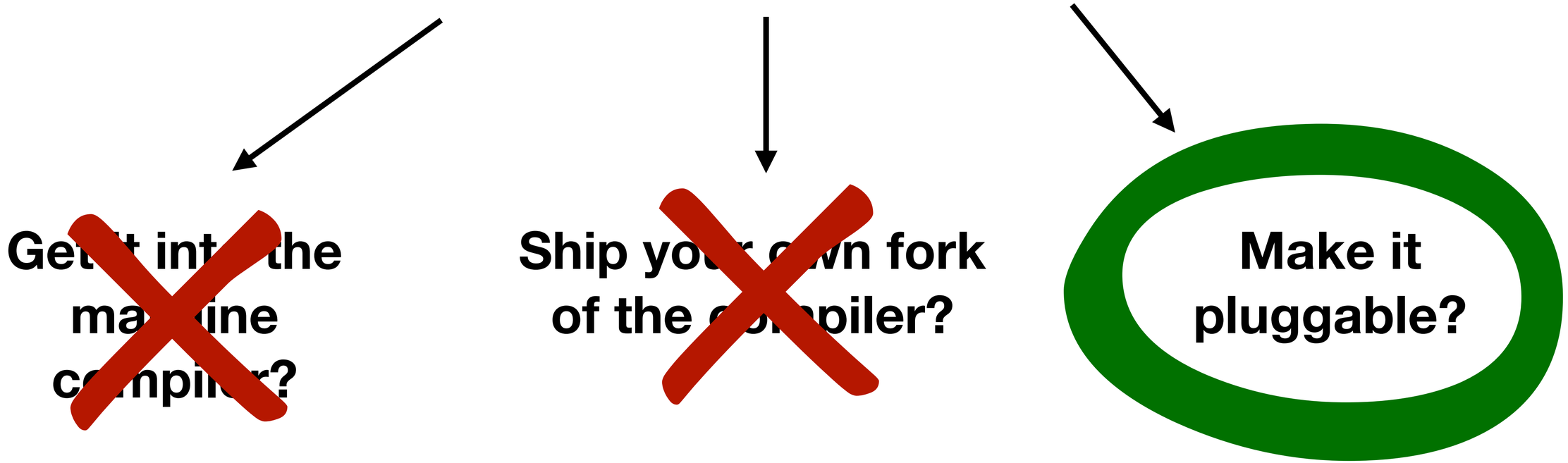
Non-signed compilers are a security issue

Ship your own fork of the compiler?

Compilers are monolithic and distributed as variants: lang × arch × OS × SDK version

Compiler pass for a niche audience

What are the options for distribution?



Quick tour of what might happen next

Find the docs

https://llvm.org/docs/WritingAnLLVMNewPMPass.html

Registering passes as plugins

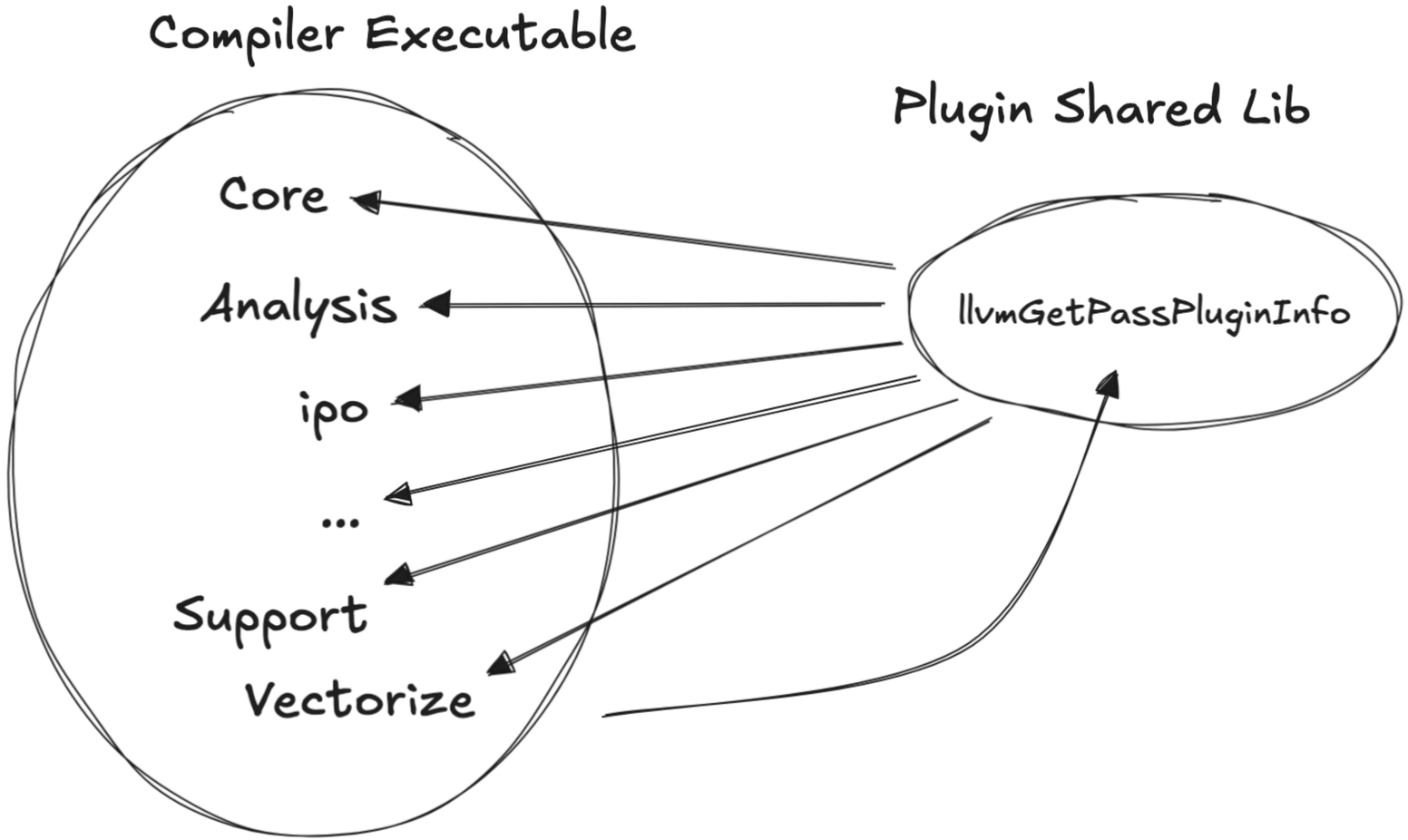
LLVM provides a mechanism to register pass plugins within various tools like `clang` or `opt`. A pass plugin can add passes to default optimization pipelines or to be manually run via tools like `opt`. For more information, see Using the New Pass Manager.

Create a CMake project at the root of the repo alongside other projects. This project must contain the following minimal `CMakeLists.txt`:

```
add_llvm_pass_plugin(MyPassName source.cpp)
```

See the definition of `add_llvm_pass_plugin` for more CMake details.

Quick tour of what might happen next

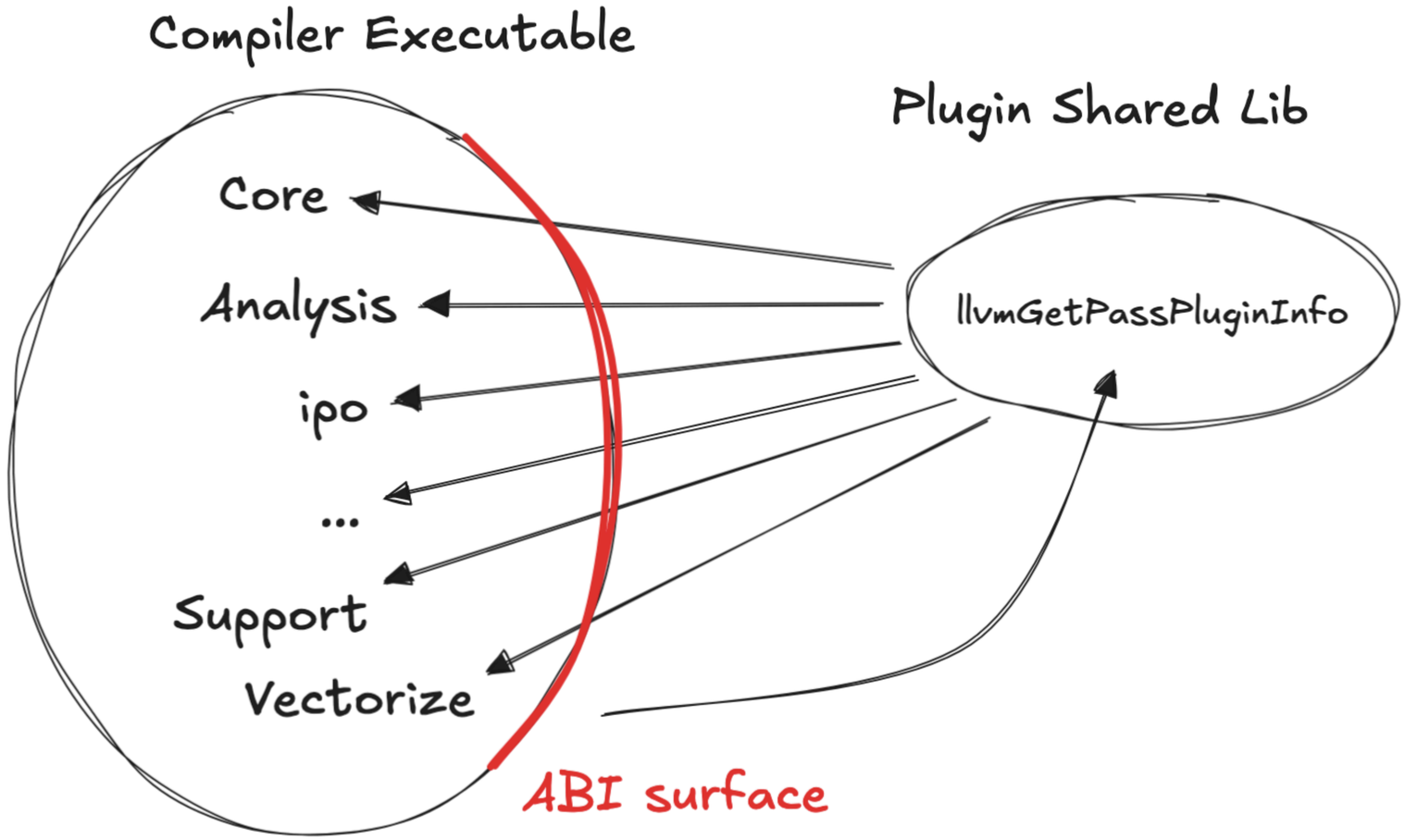


Quick tour of what might happen next

Build and run

Works!

Quick tour of what might happen next



Quick tour of what might happen next

ABI surface

```
                 U __ZNK4llvm9StringRef4findES0_m
0000000000001cf0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
0000000000001cd0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
00000000000015c0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
00000000000015a0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
0000000000001d10 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001d00 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001cc0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001cb0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001d20 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015e0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015d0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001590 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001580 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015f0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001670 t __ZNSt3__16vectorINS_10unique_ptrIN4llvm6detail11PassConceptINS2_6ModuleENS2_15AnalysisMa
0000000000001660 t __ZSt28__throw_bad_array_new_lengthB8nn180100v
0000000000002078 s __ZTVN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_
00000000000020b8 s __ZTVNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEE
0000000000002030 s __ZTVNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEE
00000000000012e0 t __ZZ21llvmGetPassPluginInfoEN3$_08__invokeERN4llvm11PassBuilderE
                 U __ZdlPv
                 U __Znwm
                 U ___stack_chk_fail
                 U ___stack_chk_guard
                 U ___stderrp
0000000000003060 d __dyld_private
                 U _abort
                 U _fprintf
                 U _free
                 U _fwrite
00000000000012b0 T _llvmGetPassPluginInfo
                 U _memcpy
                 U _memmove
                 U dyld_stub_binder
```

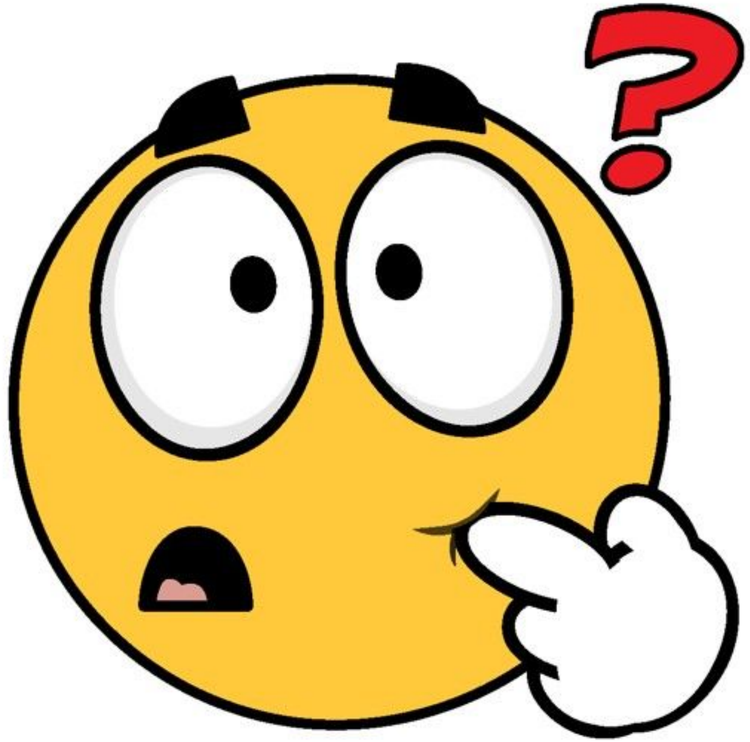
Quick tour of what might happen next

ABI surface

```
                 U __ZN4llvm11raw_ostream5writeEPKcm
                 U __ZN4llvm15SmallVectorBaseIjE13mallocForGrowEmmRm
                 U __ZN4llvm17PreservedAnalyses14AllAnalysesKeyE
0000000000001460 t __ZN4llvm23SmallVectorTemplateBaseINSt3__18functionIFbNS_9StringRefERNS_11PassManagerINS_
00000000000013b0 t __ZN4llvm23SmallVectorTemplateBaseINSt3__18functionIFbNS_9StringRefERNS_11PassManagerINS_
0000000000001460 t __ZN4llvm23SmallVectorTemplateBaseINSt3__18functionIFvRNS_11PassManagerINS_6ModuleENS_15A
00000000000013b0 t __ZN4llvm23SmallVectorTemplateBaseINSt3__18functionIFvRNS_11PassManagerINS_6ModuleENS_15A
                 U __ZN4llvm24DisableABIBreakingChecksE
0000000000003068 d __ZN4llvm30VerifyDisableABIBreakingChecksE
                 U __ZN4llvm5Value7setNameERKNS_5TwineE
0000000000001b10 t __ZN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_15
00000000000018a0 t __ZN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_15
0000000000001890 t __ZN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_15
0000000000001880 t __ZN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_15
                 U __ZNK4llvm5Value7getNameEv
0000000000001ca0 t __ZNK4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_1
0000000000001c00 t __ZNK4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_1
                 U __ZNK4llvm9StringRef4findES0_m
0000000000001cf0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
0000000000001cd0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
00000000000015c0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
00000000000015a0 t __ZNKSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEU
0000000000001d10 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001d00 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001cc0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001cb0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001d20 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015e0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015d0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001590 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001580 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
00000000000015f0 t __ZNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEEUl
0000000000001670 t __ZNSt3__16vectorINS_10unique_ptrIN4llvm6detail11PassConceptINS2_6ModuleENS2_15AnalysisMa
0000000000001660 t __ZSt28__throw_bad_array_new_lengthB8nn180100v
0000000000002078 s __ZTVN4llvm6detail9PassModelINS_6ModuleEN12_GLOBAL__N_16PyPassENS_17PreservedAnalysesENS_
00000000000020b8 s __ZTVNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEE
0000000000002030 s __ZTVNSt3__110__function6__funcIZZ21llvmGetPassPluginInfoENK3$_0clERN4llvm11PassBuilderEE
00000000000012e0 t __ZZ21llvmGetPassPluginInfoEN3$_08__invokeERN4llvm11PassBuilderE
                 U __ZdlPv
                 U __Zpwm
```

Quick tour of what might happen next

Why all these weak symbols and undefined externals to LLVM?

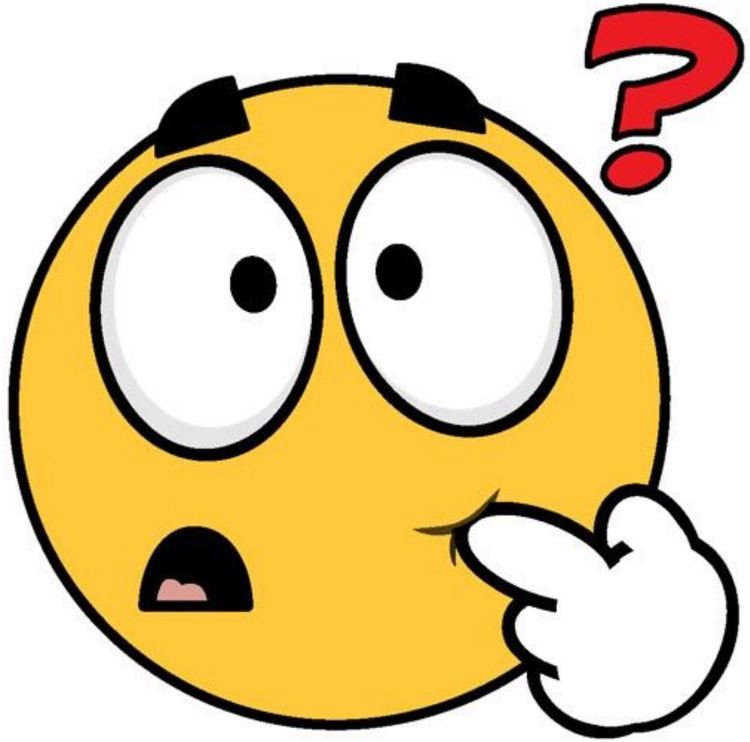


Quick tour of what might happen next

<long story here>

TL;DR:

- mainstream releases don't link libLLVM dynamically
- plugins still work BUT...



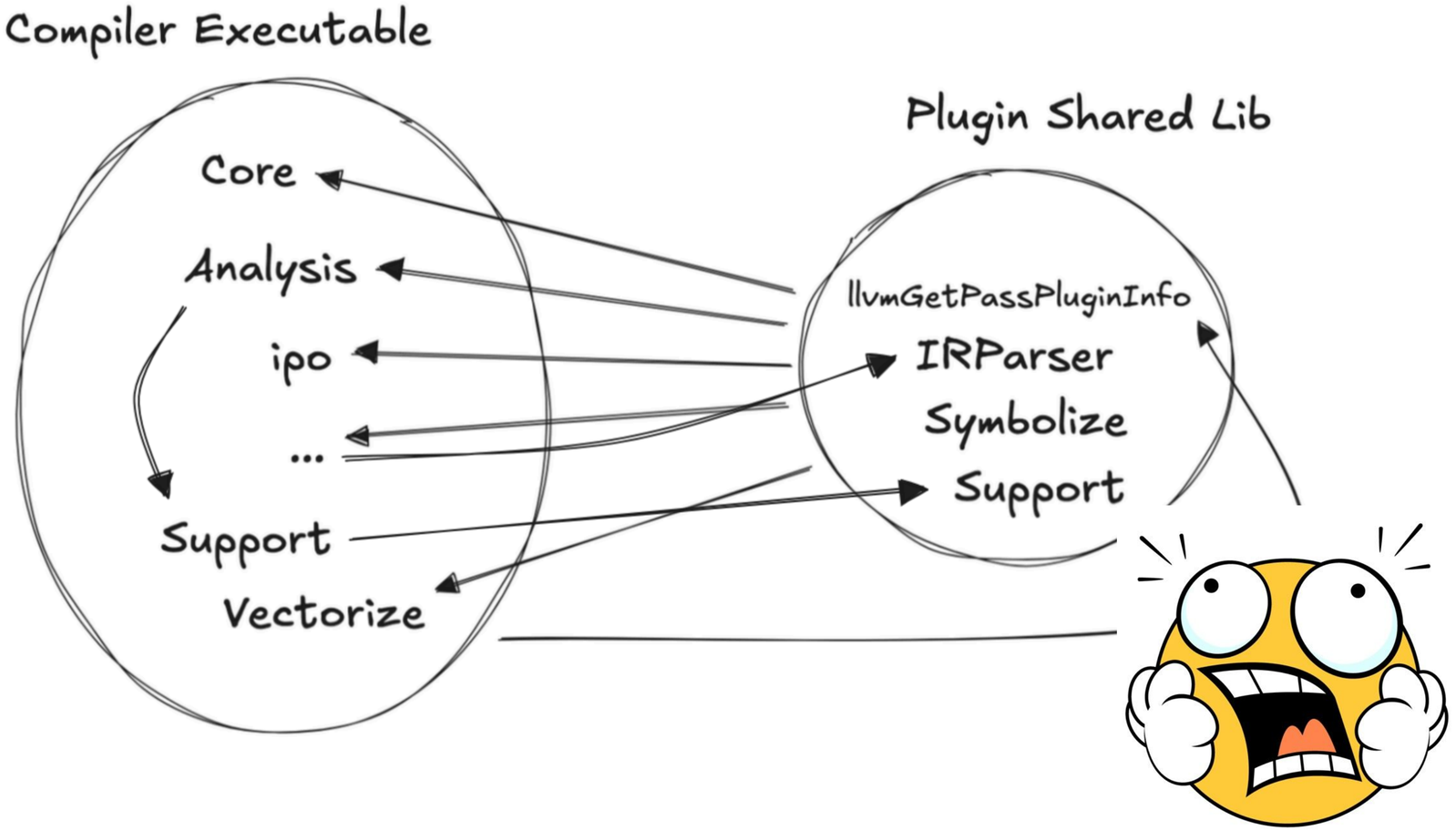
Quick tour of what might happen next

Global variables

Static init

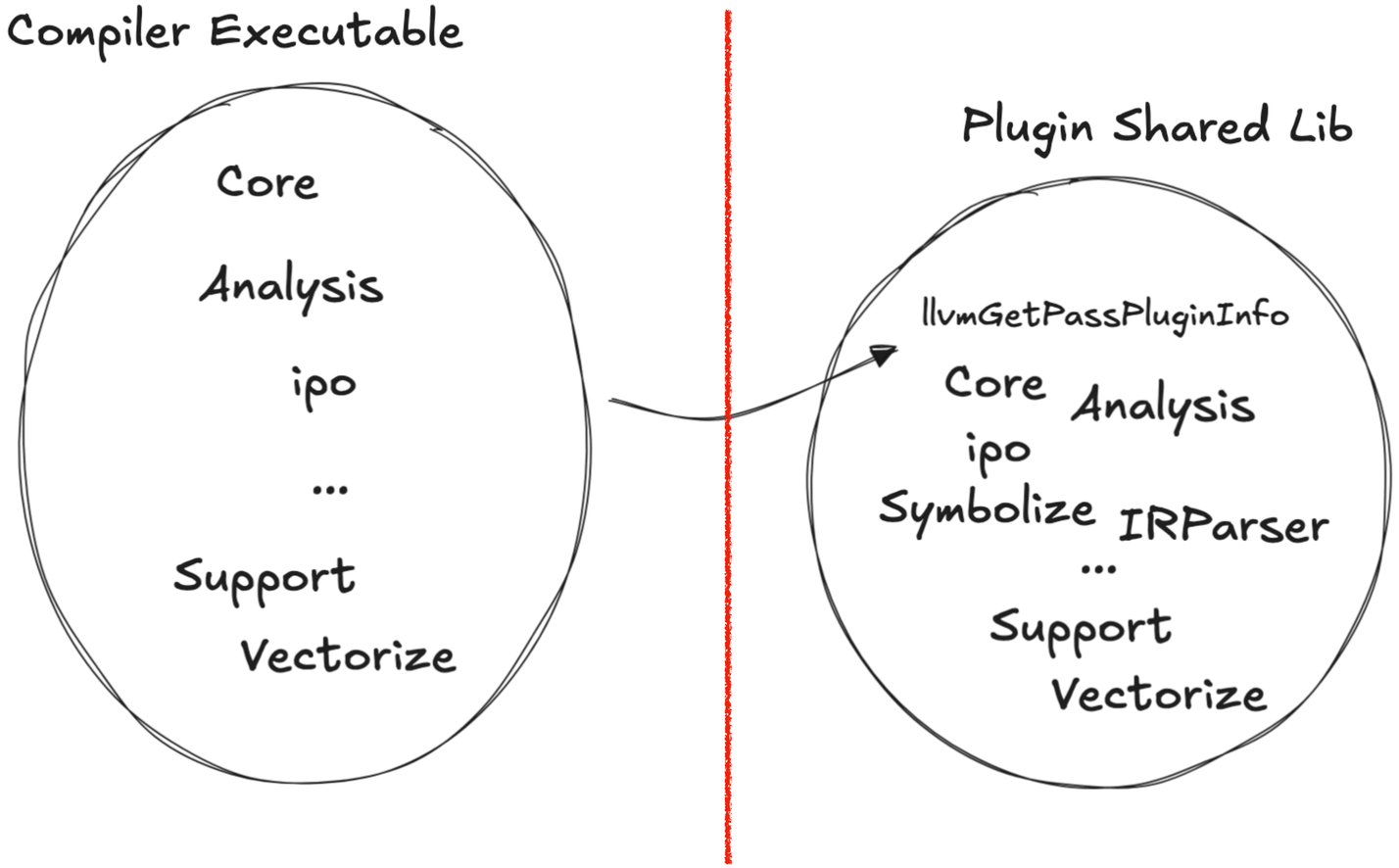
Weak symbols

Duplicate options



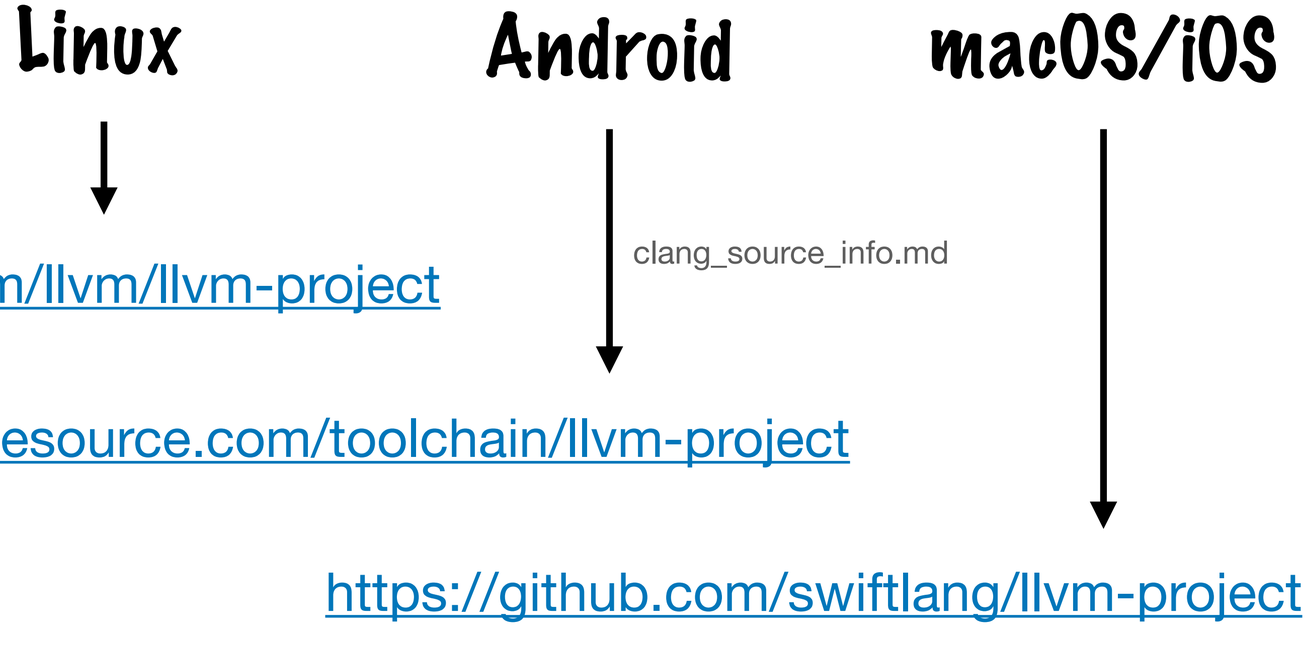
Quick tour of what might happen next

Link the world in the plugin



Quick tour of what might happen next

What sources to build the Plugin against?



Quick tour of what might happen next

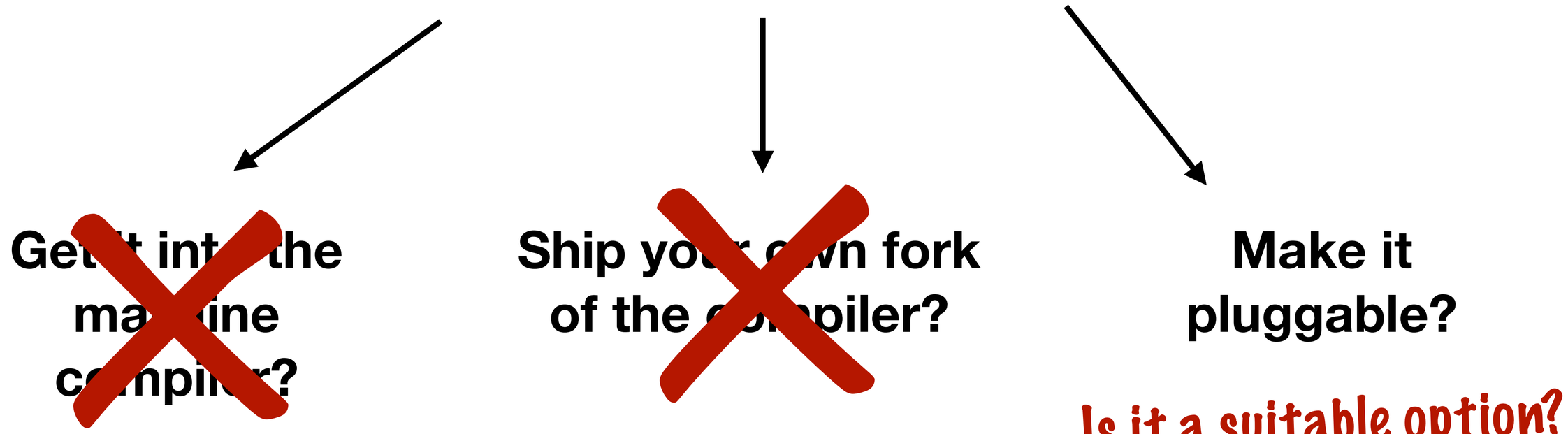
Which variants of the Plugin do we need?

one for each:

- LLVM version — 2 releases per year
- host-arch — linux, macOS (Windows?)
- host-OS — arm64, x86_64 (x86?, RISC-V?)
- target-SDK version

Compiler pass for a niche audience

What are the options for distribution?

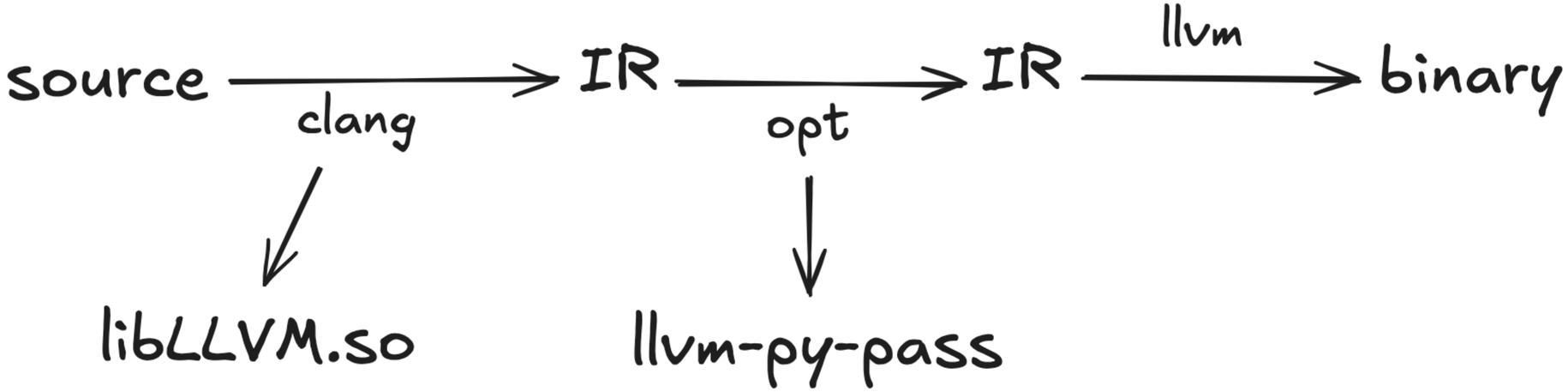


Whenever we don't know what to do we...

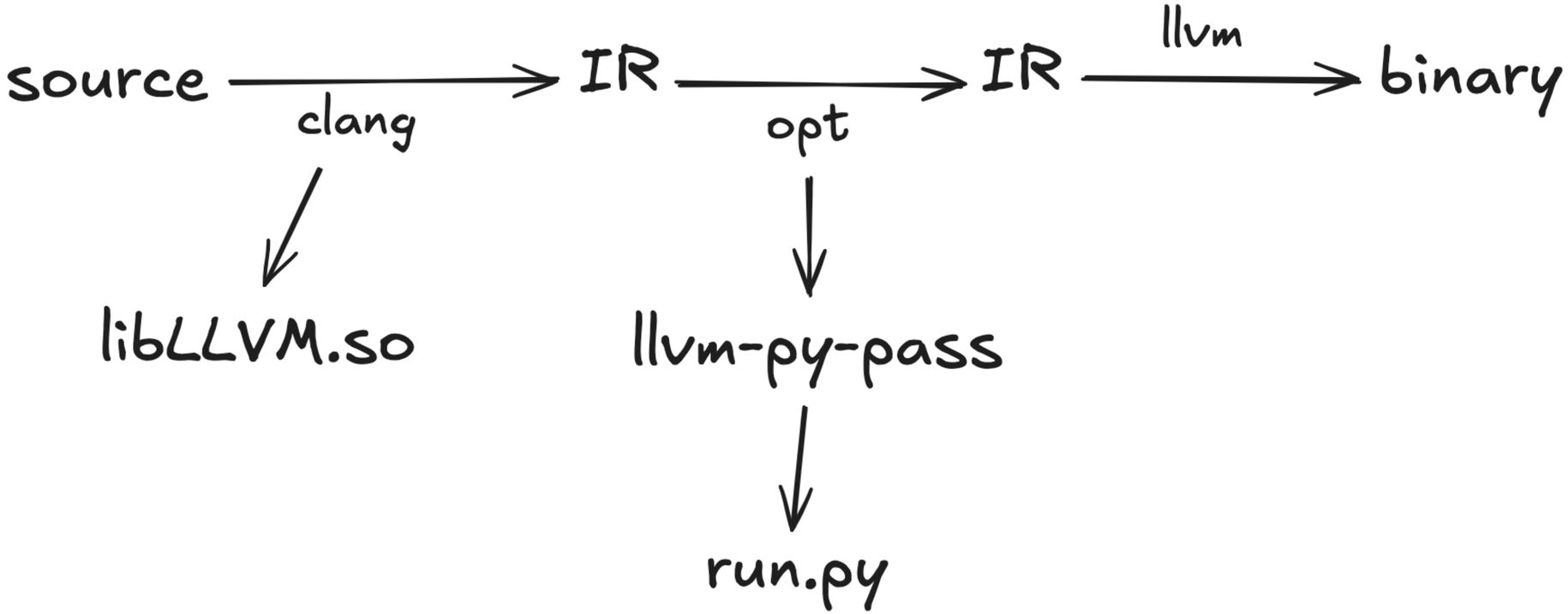
introduce a new layer of abstraction

A generic plugin container with a Python programming layer

A generic plugin container with a Python programming layer

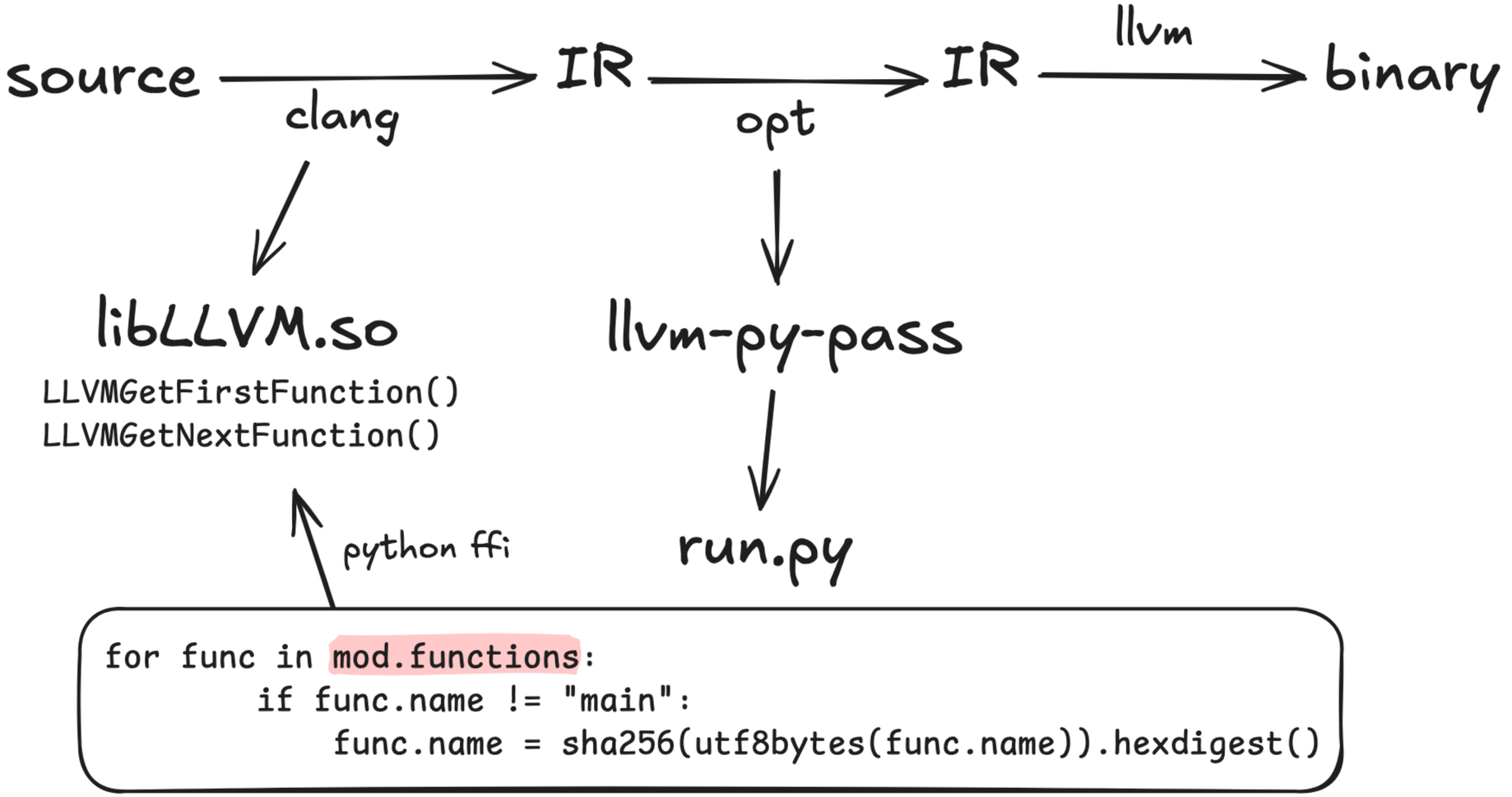


A generic plugin container with a Python programming layer




```
for func in mod.functions:
        if func.name != "main":
            func.name = sha256(utf8bytes(func.name)).hexdigest()
```

A generic plugin container with a Python programming layer



A generic plugin container with a Python programming layer

- Separation of concerns: integration vs. behavior
- Behavior described in Python:
  - abstraction from implementation
  - evaluation at runtime = isolation from plugin

A generic plugin container with a Python programming layer

- Separation of concerns: integration vs. behavior
- Behavior described in Python:
  - abstraction from implementation
  - evaluation at runtime = isolation from plugin
- Only build once for each: LLVM version × arch × OS × SDK version
- Compatible with Clang, Swift oss, Rust unofficial, clang-repl, opt

DEMO: llvm-py-pass

Questions?

Is that compatible with MLIR?

Why do we use LLVM 14?

Is llvmlite the only option as Python FFI?

Why isn't the ABI surface sufficient to determine ABI compatibility?

Date:

April 15-16

Deadline early bird tickets:

March 12

Updates:

discourse.llvm.org/t/2025-eurollvm-updates-hotel-travel-grants-etc/84370

Berlin Meetup:

meetup.com/llvm-social-berlin/