COCOAHEADS AC APRIL 2016

ALEX HOPPEN (@alex_hoppen)

TAKING A LOOK INSIDE SWIFT

PART 1

WITHOUT CODE

BUGS.SWIFT.ORG

- JIRA bug tracker
- Should replace Radars for Swift bugs
  - Visible for external committers
- Named SR-XXXX
- Very open: Anyone can register and edit any issue

SWIFT EVOLUTION

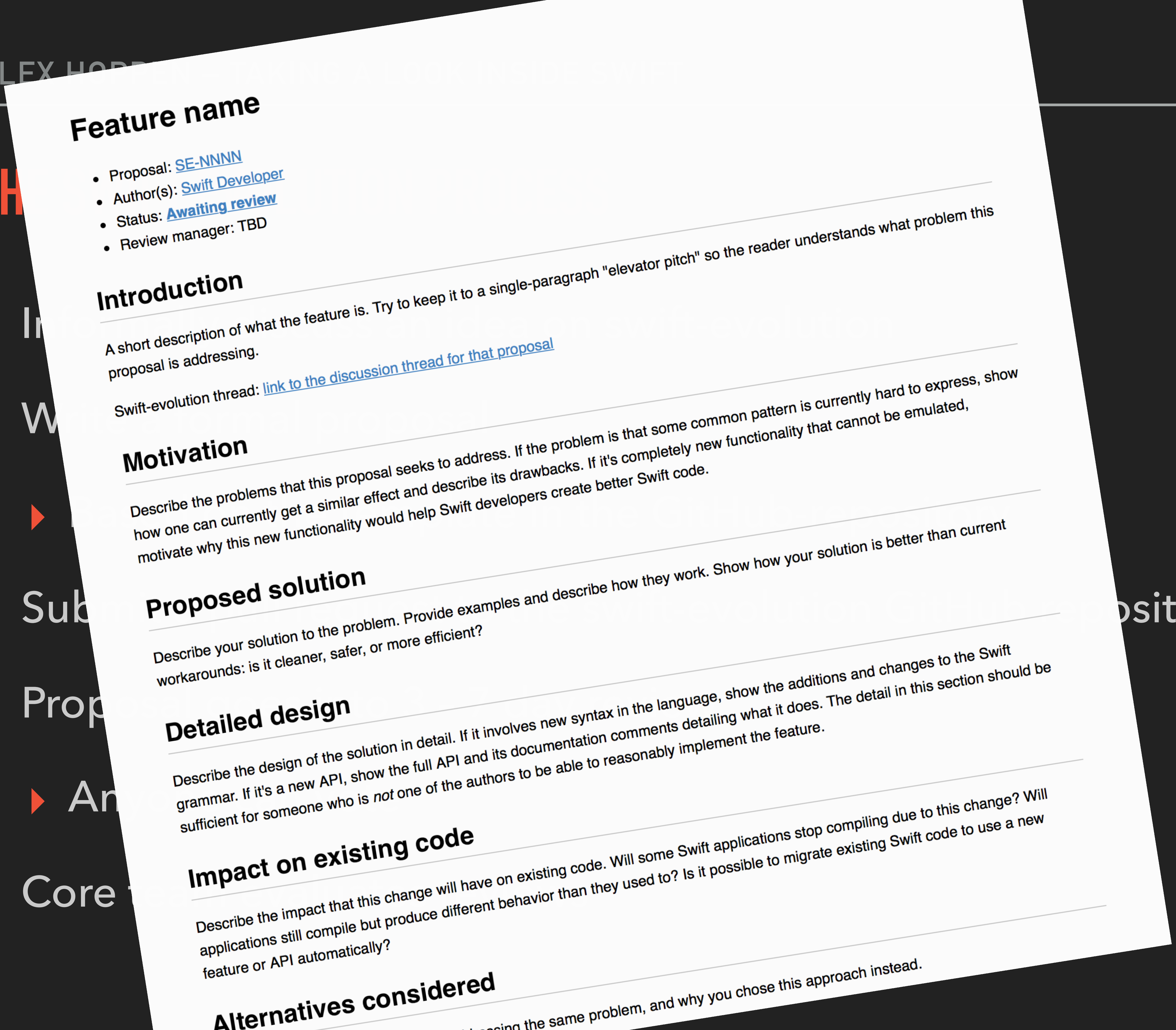
- Determines which features should be included in the next Swift releases
- Organised by the Swift evolution mailing list
  - High traffic (50 – 100 mails per day)
  - Anyone can subscribe to it
- Dominated by community but strong Core Team influence

THE SWIFT EVOLUTION PROCESS

- Informally discuss an idea on swift-evolution
- Write a formal proposal
  - Based on the template in the GitHub-repository
- Submit a pull request to the swift-evolution GitHub repository
- Proposal goes into 3 – 7 day review
  - Anyone can vote
- Core team evaluates reviews

TH

- In
- W
  -
- Sub ository
- Prop
  - An
- Core



Feature name

- Proposal: SE-NNNN
- Author(s): Swift Developer
- Status: **Awaiting review**
- Review manager: TBD

Introduction

A short description of what the feature is. Try to keep it to a single-paragraph "elevator pitch" so the reader understands what problem this proposal is addressing.

Swift-evolution thread: link to the discussion thread for that proposal

Motivation

Describe the problems that this proposal seeks to address. If the problem is that some common pattern is currently hard to express, show how one can currently get a similar effect and describe its drawbacks. If it's completely new functionality that cannot be emulated, motivate why this new functionality would help Swift developers create better Swift code.

Proposed solution

Describe your solution to the problem. Provide examples and describe how they work. Show how your solution is better than current workarounds: is it cleaner, safer, or more efficient?

Detailed design

Describe the design of the solution in detail. If it involves new syntax in the language, show the additions and changes to the Swift grammar. If it's a new API, show the full API and its documentation comments detailing what it does. The detail in this section should be sufficient for someone who is *not* one of the authors to be able to reasonably implement the feature.

Impact on existing code

Describe the impact that this change will have on existing code. Will some Swift applications stop compiling due to this change? Will applications still compile but produce different behavior than they used to? Is it possible to migrate existing Swift code to use a new feature or API automatically?

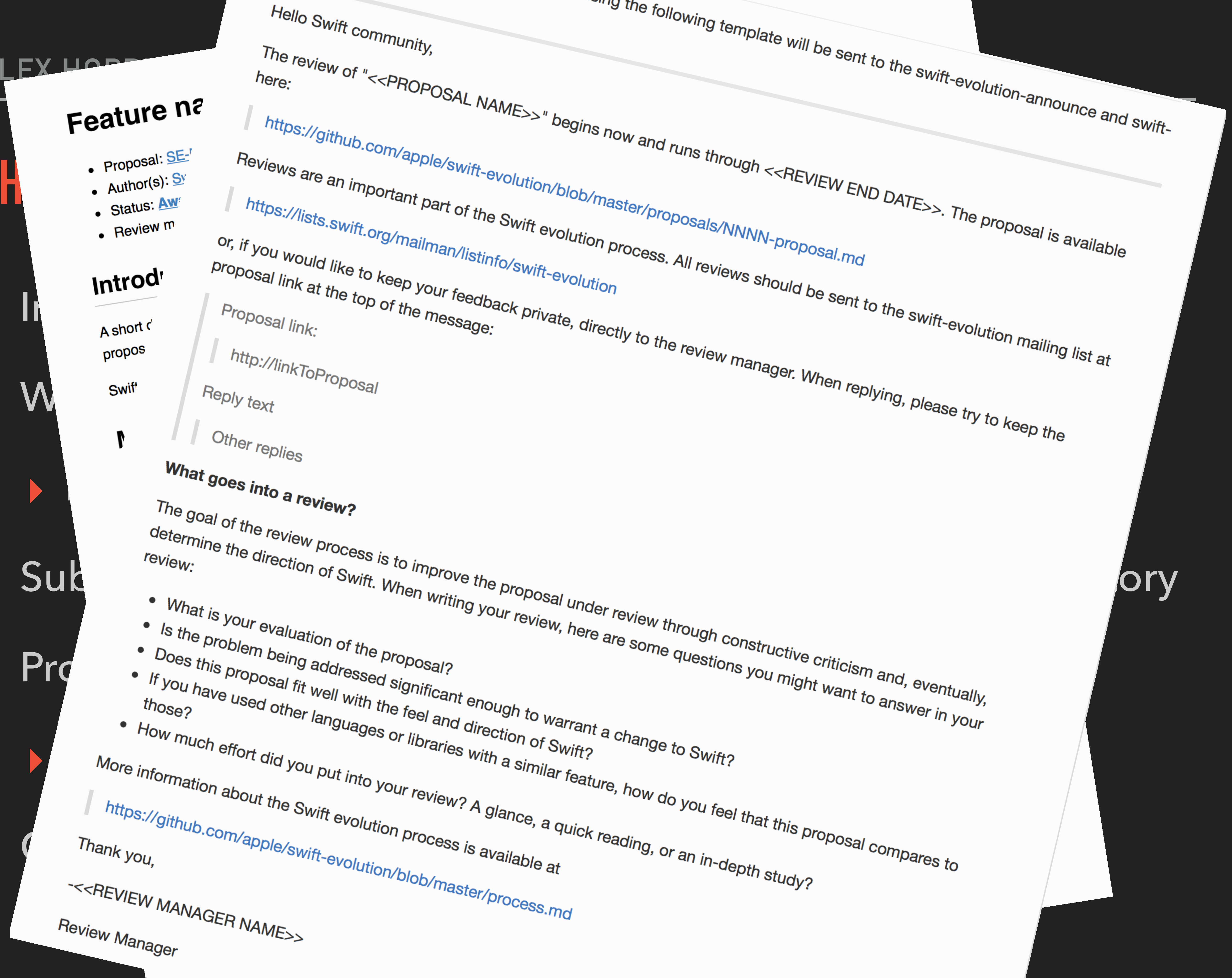
Alternatives considered

ssing the same problem, and why you chose this approach instead.

ALEX HOPPEN

TH

- In
- W
  -
- Sub ory
- Pro
  -
- C



Feature na

- Proposal: SE-
- Author(s): Sw
- Status: Aw
- Review m

Introd

A short d
propos

Swift

M

ing the following template will be sent to the swift-evolution-announce and swift-

Hello Swift community,

The review of "<<PROPOSAL NAME>>" begins now and runs through <<REVIEW END DATE>>. The proposal is available here:

> https://github.com/apple/swift-evolution/blob/master/proposals/NNNN-proposal.md

Reviews are an important part of the Swift evolution process. All reviews should be sent to the swift-evolution mailing list at

> https://lists.swift.org/mailman/listinfo/swift-evolution

or, if you would like to keep your feedback private, directly to the review manager. When replying, please try to keep the proposal link at the top of the message:

> Proposal link:
>
> > http://linkToProposal
>
> Reply text
>
> > Other replies

What goes into a review?

The goal of the review process is to improve the proposal under review through constructive criticism and, eventually, determine the direction of Swift. When writing your review, here are some questions you might want to answer in your review:

- What is your evaluation of the proposal?
- Is the problem being addressed significant enough to warrant a change to Swift?
- Does this proposal fit well with the feel and direction of Swift?
- If you have used other languages or libraries with a similar feature, how do you feel that this proposal compares to those?
- How much effort did you put into your review? A glance, a quick reading, or an in-depth study?

More information about the Swift evolution process is available at

> https://github.com/apple/swift-evolution/blob/master/process.md

Thank you,

-<<REVIEW MANAGER NAME>>

Review Manager

MAILING LISTS

- swift-users: User questions about Swift
- swift-evolution: Discussion of evolution proposals
- swift-evolution-announce: Review announcements
- swift-dev: Swift compiler, runtime, standard library, and Source Kit
- swift-corelibs-dev: Swift core libraries (NSStuff)
- swift-lldb-dev: Swift REPL and Swift-specific aspects of LLDB
- swift-build-dev: Swift package manager, low lever build system
- + Twitter

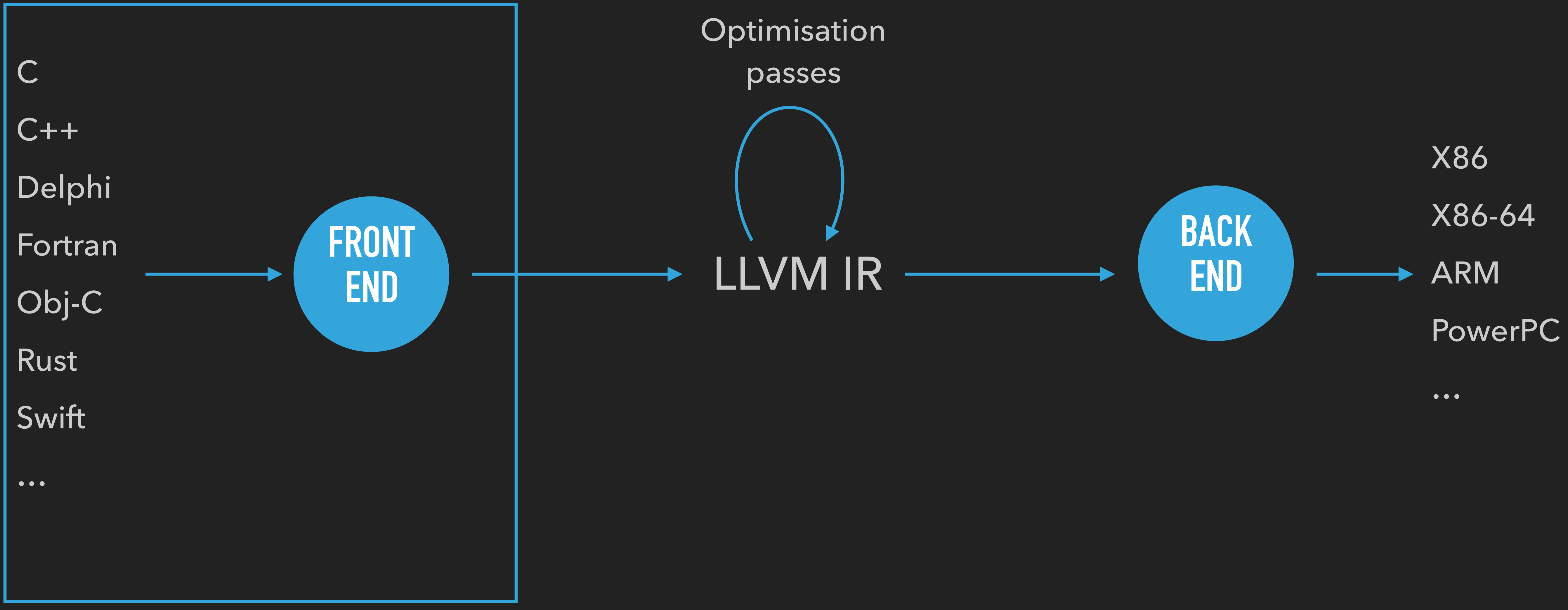
PART 2

WITH CODE

LLVM – A HISTORY

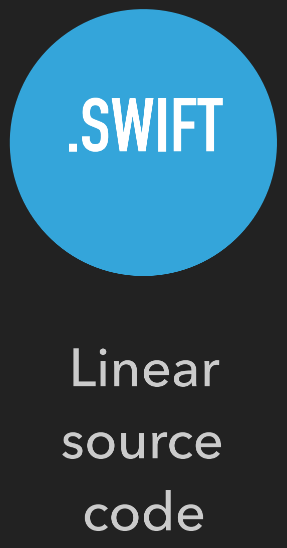
- Master thesis project of Chris Lattner @ University of Illinois: “An Infrastructure for Multi-Stage Optimization”
- Base of the C-family compiler clang
- Lattner hired by Apple in 2005
- Used by Apple since then

LLVM – A MODULAR COMPILER



Only part that needed rewriting

THE SWIFT FRONTEND



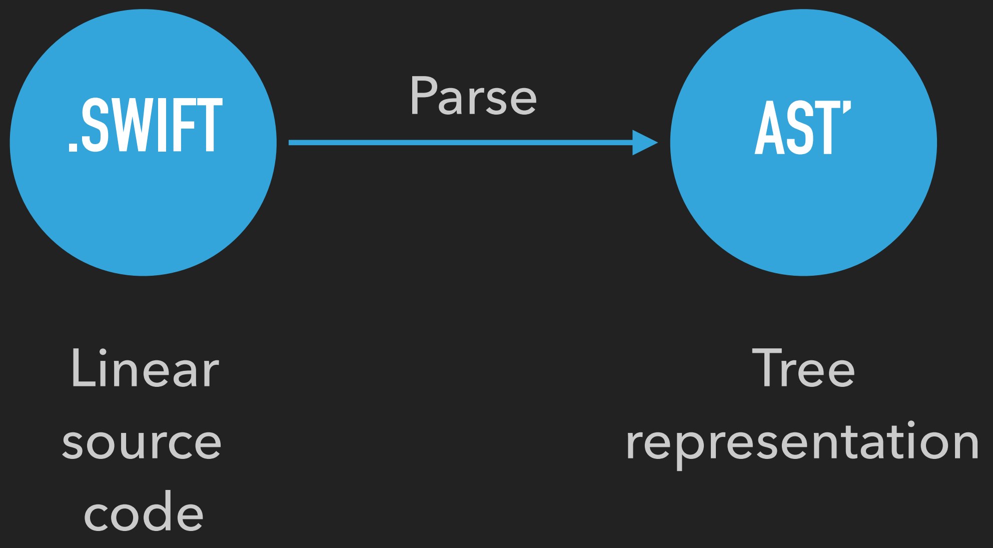
.SWIFT

Linear
source
code

CAT HELLO.SWIFT

```
// public func print(items: Any...,
//                   separator: String = " ",
//                   terminator: String = "\n")

print("Hello world")
```

THE SWIFT FRONTEND



.SWIFT

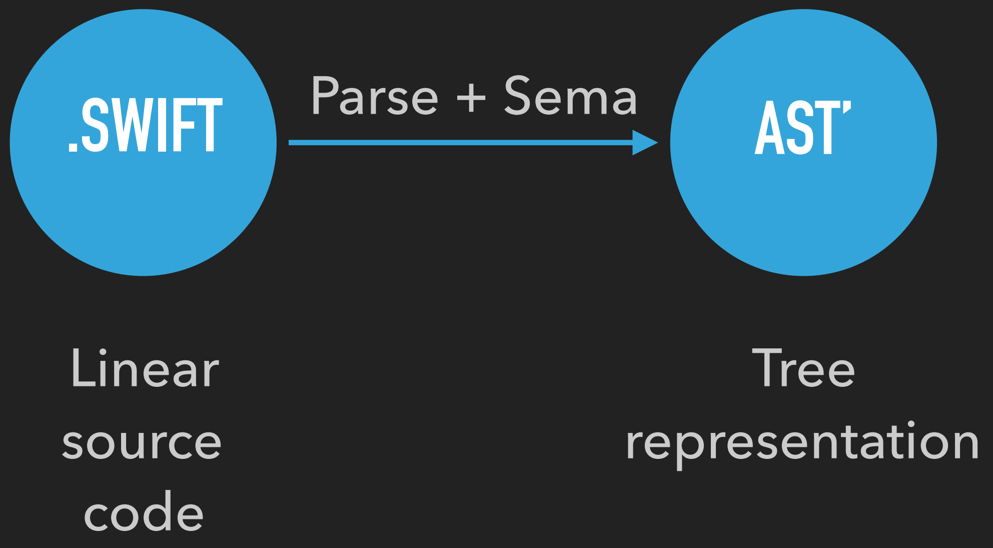
Parse

AST

Linear source code

Tree representation

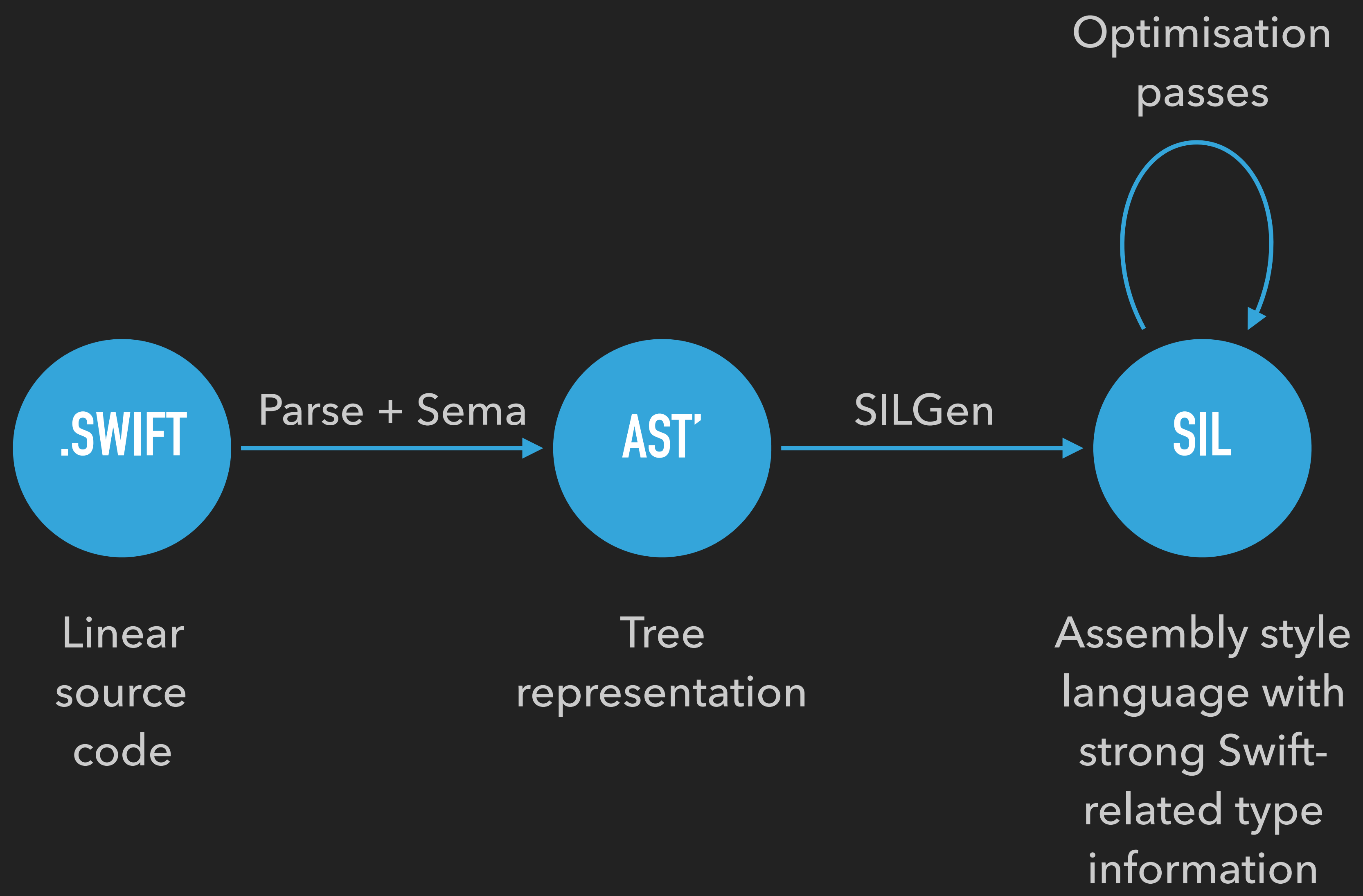
THE SWIFT FRONTEND



SWIFTC -DUMP-AST HELLO.SWIFT

```
(source_file
  (top_level_code_decl
    (brace_stmt
      (call_expr type='()' location=test.swift:1:1 range=[test.swift:1:1 - line:1:20] nothrow
        (declref_expr type='(Any..., separator: String, terminator: String) -> ()' location=test.swift:
1:1 range=[test.swift:1:1 - line:1:1] decl=Swift.(file).print(_:separator:terminator:) specialized=no)
        (tuple_shuffle_expr implicit type='(Any..., separator: String, terminator: String)'
location=test.swift:1:7 range=[test.swift:1:6 - line:1:20] sourceIsScalar elements=[-2, -1, -1]
variadic_sources=[0]
          (paren_expr type='Any' location=test.swift:1:7 range=[test.swift:1:6 - line:1:20]
            (erasure_expr implicit type='Any' location=test.swift:1:7 range=[test.swift:1:7 - line:1:7]
              (call_expr implicit type='String' location=test.swift:1:7 range=[test.swift:1:7 - line:1:7]
nothrow
                (constructor_ref_call_expr implicit type='(_builtinStringLiteral: RawPointer, byteSize:
Word, isASCII: Int1) -> String' location=test.swift:1:7 range=[test.swift:1:7 - line:1:7] nothrow
                  (declref_expr implicit type='String.Type -> (_builtinStringLiteral: RawPointer,
byteSize: Word, isASCII: Int1) -> String' location=test.swift:1:7 range=[test.swift:1:7 - line:1:7]
decl=Swift.(file).String.init(_builtinStringLiteral:byteSize:isASCII:) specialized=no)
                  (type_expr implicit type='String.Type' location=test.swift:1:7 range=[test.swift:1:7 -
line:1:7] typerepr='String'))
                (string_literal_expr type='(_builtinStringLiteral: Builtin.RawPointer, byteSize:
Builtin.Word, isASCII: Builtin.Int1)' location=test.swift:1:7 range=[test.swift:1:7 - line:1:7]
encoding=utf8 value="Hello world")))))))))
```

THE SWIFT FRONTEND



Optimisation passes

.SWIFT

Parse + Sema

AST'

SILGen

SIL

Linear source code

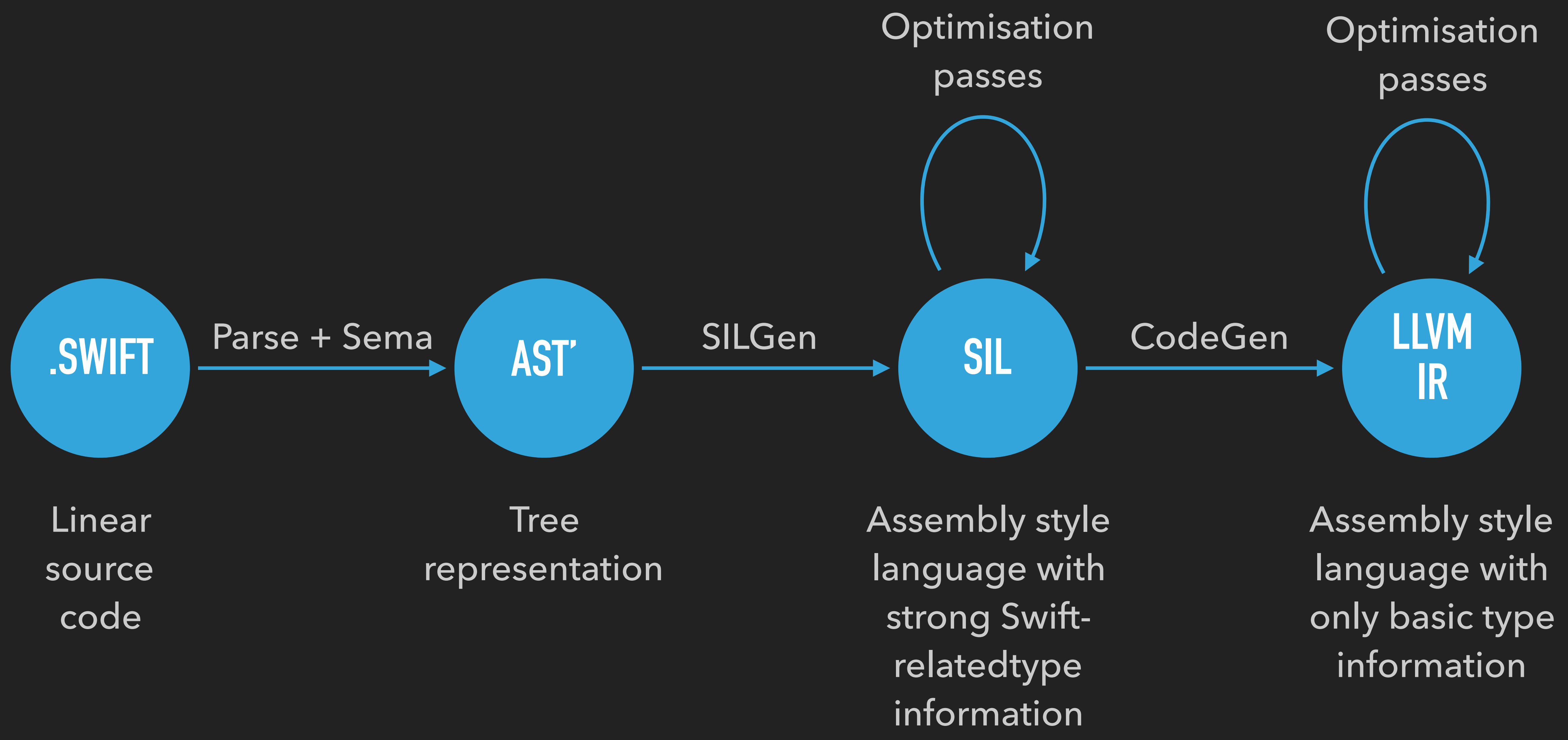
Tree representation

Assembly style language with strong Swift-related type information

SWIFTC -EMIT-SILGEN HELLO.SWIFT | XCRUN SWIFT-DEMANGLE

```
sil_stage raw

import Builtin
import Swift
import SwiftShims

sil @main : $@convention(c) (Int32, UnsafeMutablePointer<UnsafeMutablePointer<Int8>>) -> Int32 {
bb0(%0 : $Int32, %1 : $UnsafeMutablePointer<UnsafeMutablePointer<Int8>>):
  // ...
  %4 = function_ref @Swift.print ([protocol<>], separator : Swift.String, terminator : Swift.String) -> ()
: $@convention(thin) (@owned Array<protocol<>>, @owned String, @owned String) -> () // user: %23
  // ...
  %12 = function_ref @Swift.String.init (_builtinStringLiteral : Builtin.RawPointer, byteSize :
Builtin.Word, isASCII : Builtin.Int1) -> Swift.String : $@convention(thin) (Builtin.RawPointer,
Builtin.Word, Builtin.Int1, @thin String.Type) -> @owned String // user: %17
  %13 = metatype $@thin String.Type               // user: %17
  %14 = string_literal utf8 "Hello world"         // user: %17
  // ...
  %17 = apply %12(%14, %15, %16, %13) : $@convention(thin) (Builtin.RawPointer, Builtin.Word,
Builtin.Int1, @thin String.Type) -> @owned String // user: %18
  store %17 to %11 : $*String                     // id: %18
  %19 = function_ref @Swift.(print ([protocol<>], separator : Swift.String, terminator : Swift.String) ->
()).(default argument 1) : $@convention(thin) () -> @owned String // user: %20
  %20 = apply %19() : $@convention(thin) () -> @owned String // user: %23
  %21 = function_ref @Swift.(print ([protocol<>], separator : Swift.String, terminator : Swift.String) ->
()).(default argument 2) : $@convention(thin) () -> @owned String // user: %22
  %22 = apply %21() : $@convention(thin) () -> @owned String // user: %23
  %23 = apply %4(%8, %20, %22) : $@convention(thin) (@owned Array<protocol<>>, @owned String, @owned
String) -> ()
  // ...
  return %25 : $Int32                             // id: %26
}
// ...
```

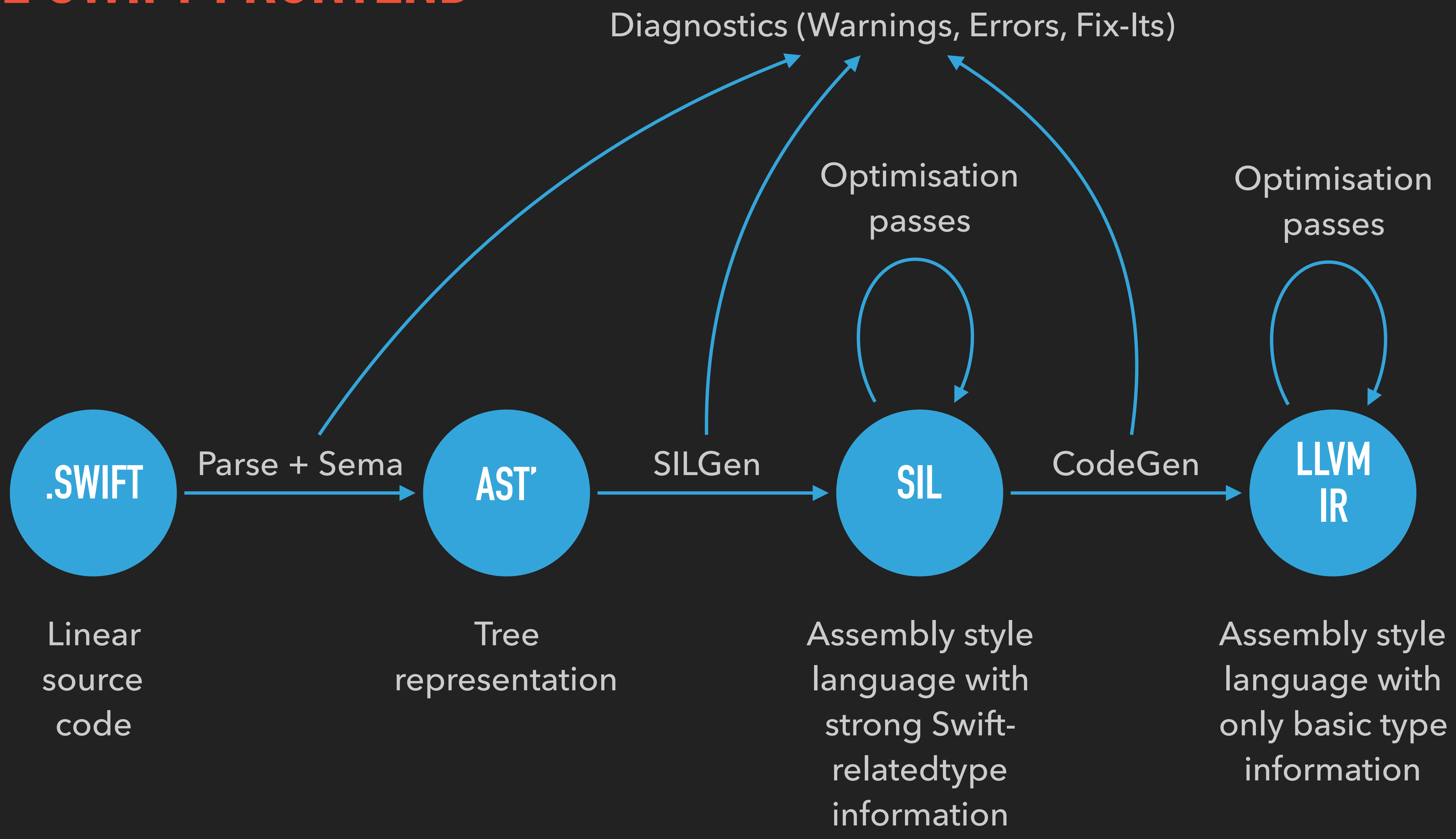
THE SWIFT FRONTEND


Optimisation passes
Optimisation passes
.SWIFT
Parse + Sema
AST'
SILGen
SIL
CodeGen
LLVM IR
Linear source code
Tree representation
Assembly style language with strong Swift-relatedtype information
Assembly style language with only basic type information

SWIFTC -EMIT-IR -O HELLO.SWIFT | XCRUN SWIFT-DEMANGLE

```
; ...

@0 = private unnamed_addr constant [12 x i8] c"Hello world\00"

; ...

define i32 @main(i32, i8**) #0 {
entry:
; ...

type metadata accessor for Swift._ContiguousArrayStorage<protocol<>>.exit:         ; preds = %once_done, %type metadata
accessor for protocol<>.exit.i
  ; ...
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @0, i64 0, i64 0), i8** %object._core._baseAddress._rawValue, align 8
  %object._core._countAndFlags = getelementptr inbounds i8, i8* %16, i64 40
  %23 = bitcast i8* %object._core._countAndFlags to <2 x i64>*
  store <2 x i64> <i64 11, i64 0>, <2 x i64>* %23, align 8
  %24 = call { i8*, i64, i64 } @Swift.(print ([protocol<>], separator : Swift.String, terminator : Swift.String) -> ()).
(default argument 1)()
  %25 = extractvalue { i8*, i64, i64 } %24, 0
  %26 = extractvalue { i8*, i64, i64 } %24, 1
  %27 = extractvalue { i8*, i64, i64 } %24, 2
  %28 = call { i8*, i64, i64 } @Swift.(print ([protocol<>], separator : Swift.String, terminator : Swift.String) -> ()).
(default argument 2)()
  %29 = extractvalue { i8*, i64, i64 } %28, 0
  %30 = extractvalue { i8*, i64, i64 } %28, 1
  %31 = extractvalue { i8*, i64, i64 } %28, 2
  call void @Swift.print ([protocol<>], separator : Swift.String, terminator : Swift.String) -> ()(%swift.bridge* %19, i8* %25,
i64 %26, i64 %27, i8* %29, i64 %30, i64 %31)
  ret i32 0
}

; ...
```

THE SWIFT FRONTEND


Diagnostics (Warnings, Errors, Fix-Its)
Optimisation passes
Optimisation passes
.SWIFT
Parse + Sema
AST
SILGen
SIL
CodeGen
LLVM IR
Linear source code
Tree representation
Assembly style language with strong Swift-relatedtype information
Assembly style language with only basic type information

TODO-LIST OF THE SEMA PHASE

- Resolve method names
- Resolve generics
- Perform type checking

THE TYPE-CHECKER – CONSTRAINT GENERATION

- Resolve function calls and operators to all overloads
- Add constraints for method parameters
- Add constraints for the return type
- Add constraints for generic type requirements

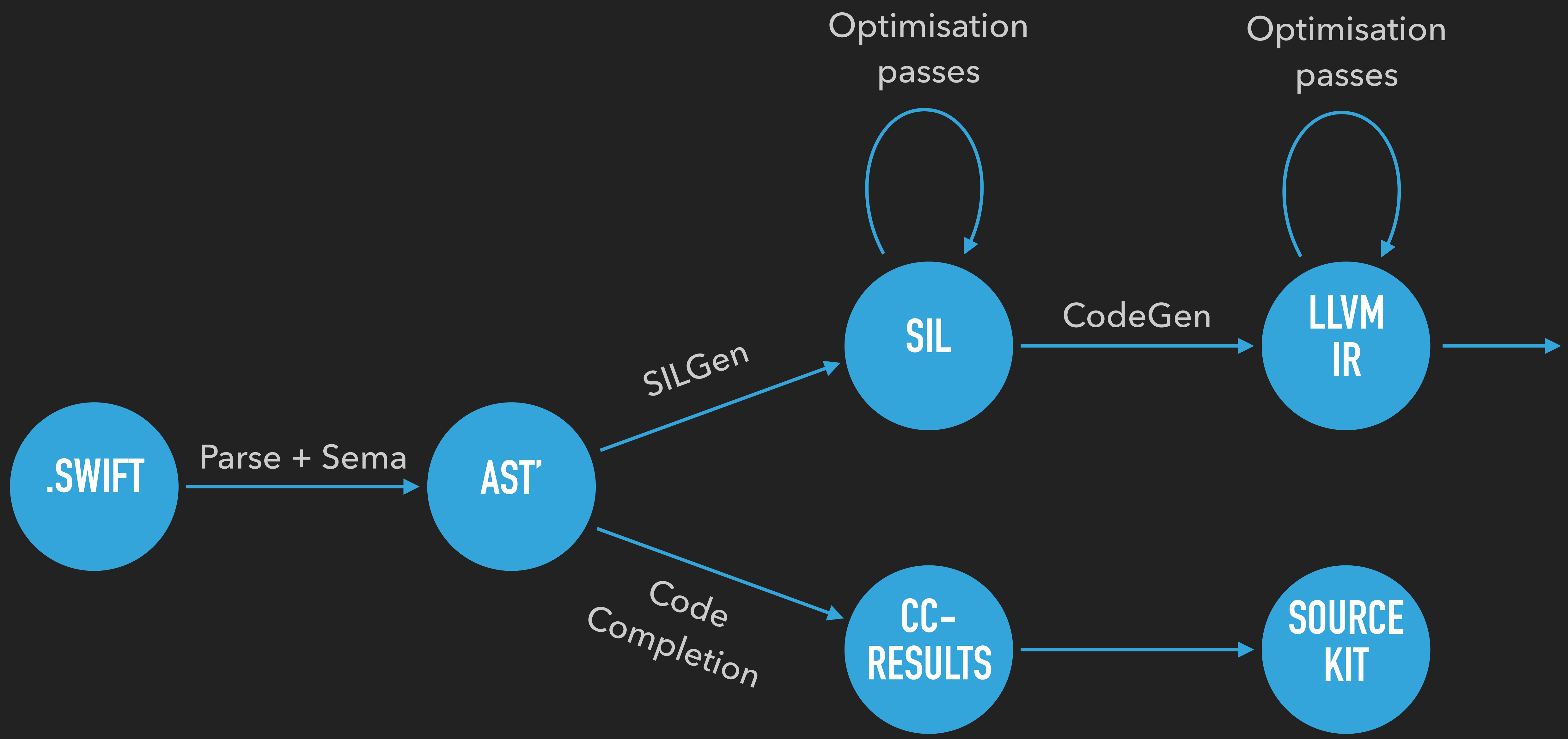
THE TYPE-CHECKER – CONSTRAINT SOLVING

- Problem has inherently exponential complexity
- Simplify the constraint system
- Perform guesses (e.g. for conjunction constraints)
- Repeat until a specific solution has been found
- Give the solution a score
- Solution with best score is chosen

THE TYPE-CHECKER – CONSTRAINT APPLICATION

- Replace the types in the AST with the result of the constraint system
- Generates type checked AST with resolved function calls

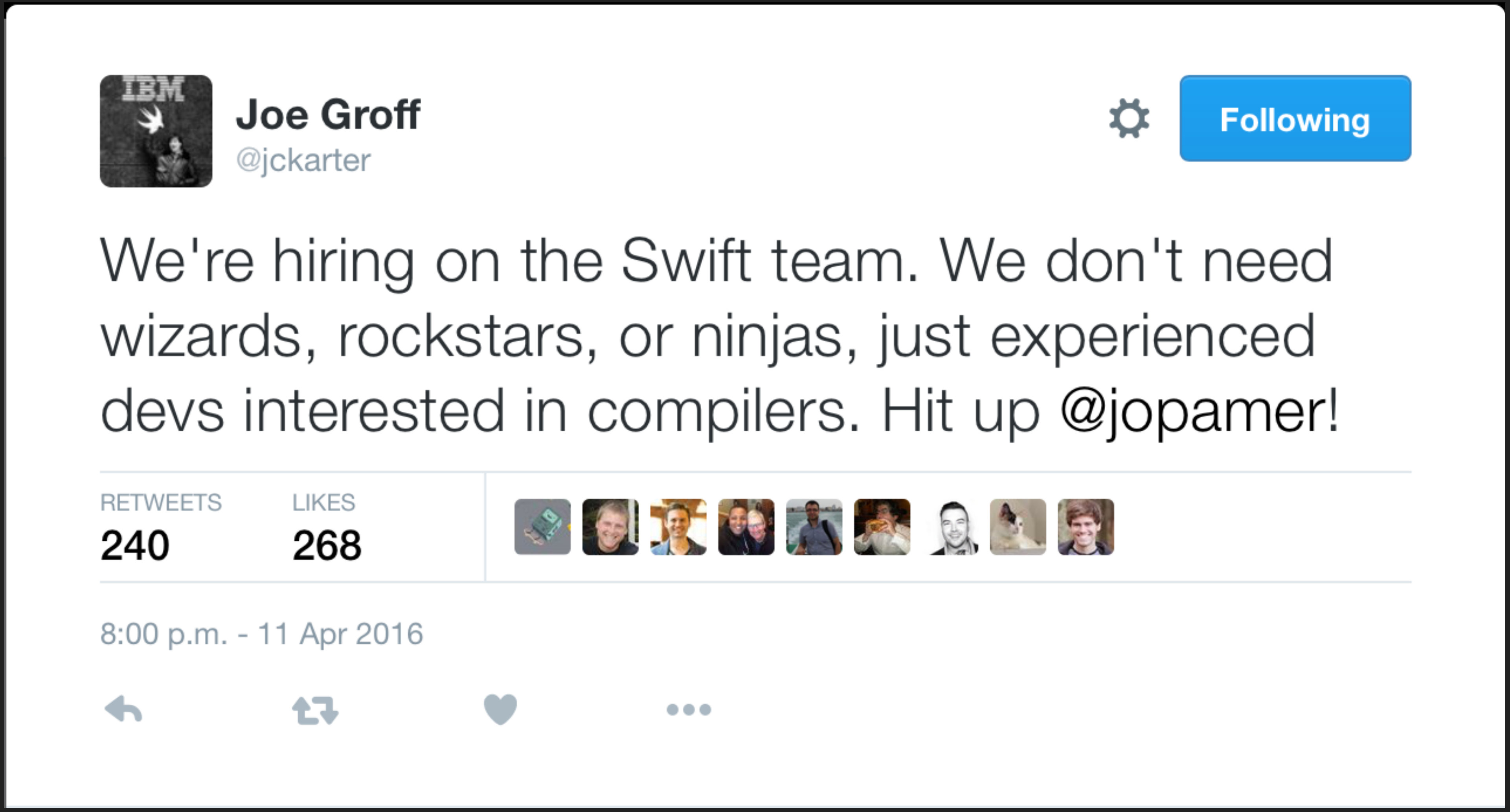
WHAT ABOUT CODE COMPLETION?



SWIFT STANDARD LIBRARY

- Contains basic Swift types like Int, Bool, Array, Optional, …
- Written entirely in Swift and Python-Swift mix .gyb
- Has access to the Builtin module that maps directly to LLVM IR instructions / types

LIKED THIS STUFF?



Joe Groff
@jckarter

We're hiring on the Swift team. We don't need wizards, rockstars, or ninjas, just experienced devs interested in compilers. Hit up @jopamer!

RETWEETS
240

LIKES
268

8:00 p.m. - 11 Apr 2016