Advanced Techniques to Reduce the Build Time in Xcode

Kumar Reddy, Lead iOS Engineer, Swiggy

https://twitter.com/kumarreddy_b

https://medium.com/@kumarreddy_b

Agenda

What is Build system and Compilers ?

What is LLVM ?

Swift Frontend for LLVM

What is Build time ?

Tips and techniques to speedup the Build time

Demos and Analyze the sample application

What is Build System

Build systems are a functional based languages mapping a set of source resources (in most cases, files) to a target (executable)

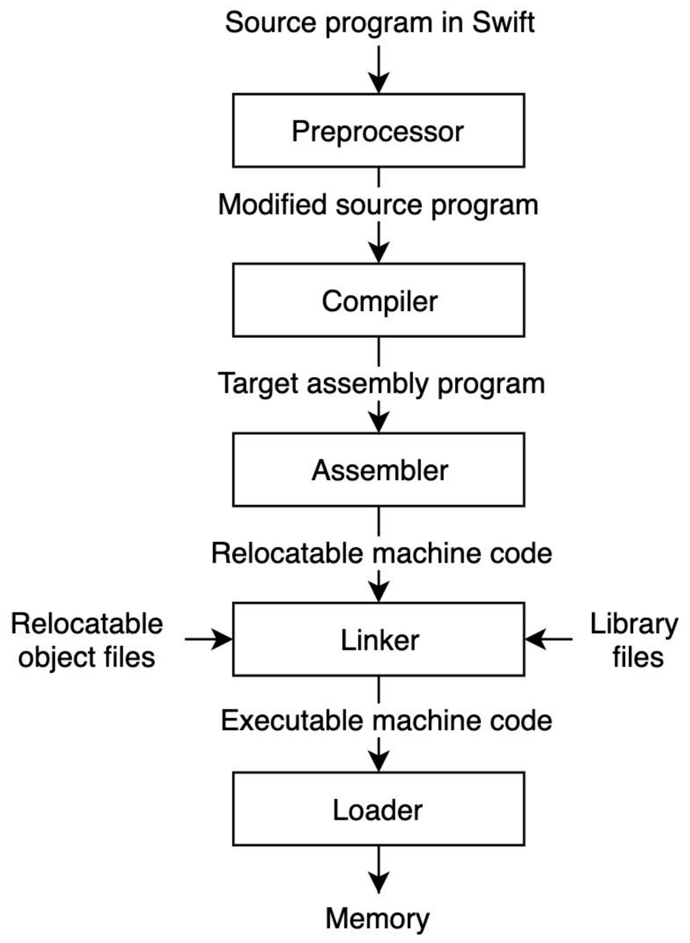
Compilation + Linking + Creating an Executable

It must fully understand the dependencies of the build

Parallelization

Build actions are idempotent.

XCode Build System



Preprocessor

The purpose of preprocessing step is to transform your program in a way that it can be fed to a compiler.

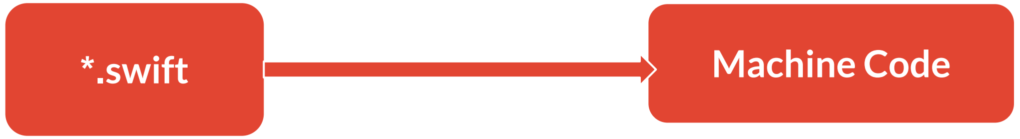
It replaces macros with their definitions, discovers dependencies and resolves preprocessor directives.

Discover dependencies with low level build system llbuild

Compiler

Converts your source code to Machine code. (that's what machines understand right ?)

Swift compiler converts all the swift source code to machine code.



clang is Apple's official compiler for the C languages family. It is open-sourced here: swift-clang.

swiftc is a Swift compiler executable which is used by Xcode to compile and run Swift source code.

Assembler

Assembler translates human-readable assembly code into relocatable machine code.

It produces Mach-O files which are basically a collection of code and data.

Mach-O is a stream of bytes grouped in some meaningful chunks that will run on the ARM processor of an iOS device or the Intel processor on a Mac.

Linker

Linker is a computer program that merges various object files and libraries together in order to make a single Mach-O executable file that can be run on iOS or macOS system

Object Files + dylib, .a , .tbd => Single Executable file

Loader

Lastly, loader which is a part of operating system, brings a program into memory and executes it.

Loader allocates memory space required to run the program and initializes registers to initial state.

Loads dylibs and other dynamic libraries required to run the program.

Loader times directly proportional to App Launch Times.

Ideal build system

Easy to share redundant work

Compiler can optimize the entire build

Build systems can optimize via rich compiler API

Consistent incremental builds & debuggable architecture

1. Library-based compiler
2. Extensible build system
3. Compiler plugin

llbuild

llbuild is a set of libraries for building build systems.

llbuild is designed around a reusable, flexible, and scalable general purpose *build engine* capable of solving many "build system"-like problems.

1. Incremental
2. Consistent
3. Persistent
4. Parallel & Efficient

Improving Compile time

Distributed compile time

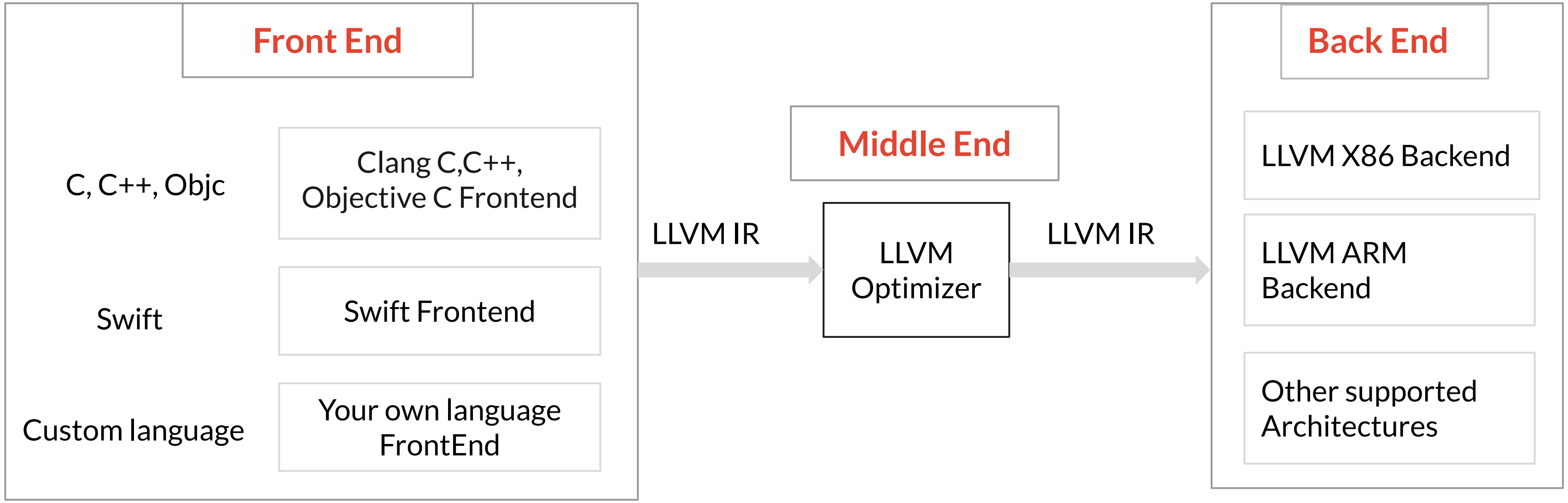
Fancy Caching (distributed and shared)

Ideally less work If possible do less work

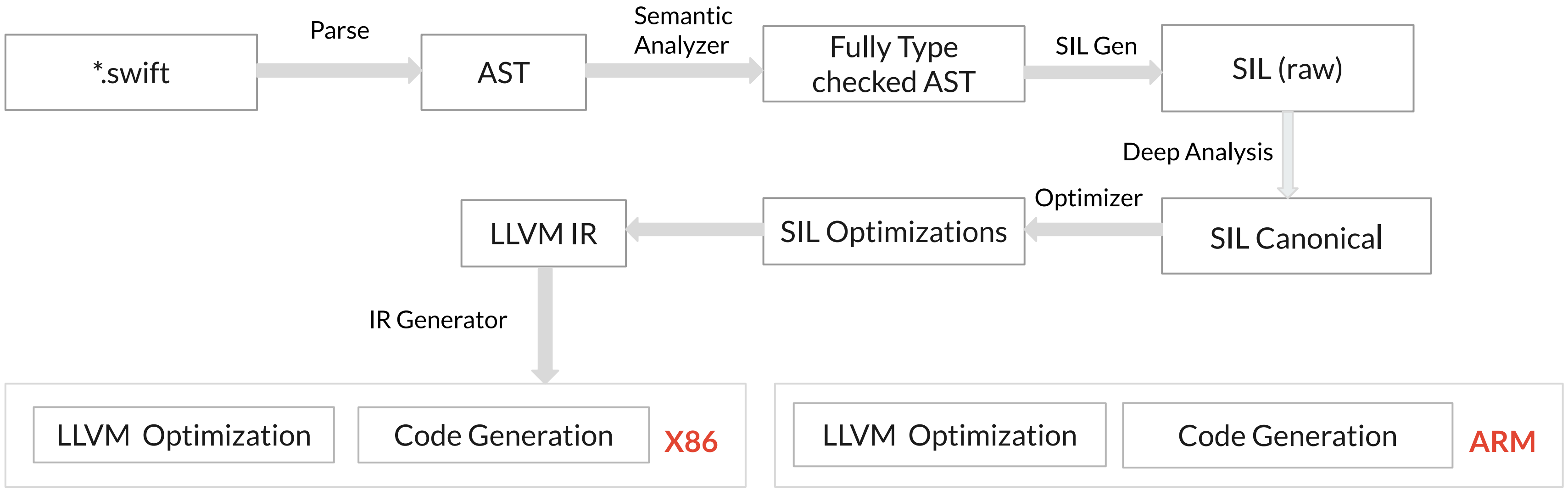
Optimization level, Parser , Lexer

What is LLVM

A collection of compiler and toolchain technologies.

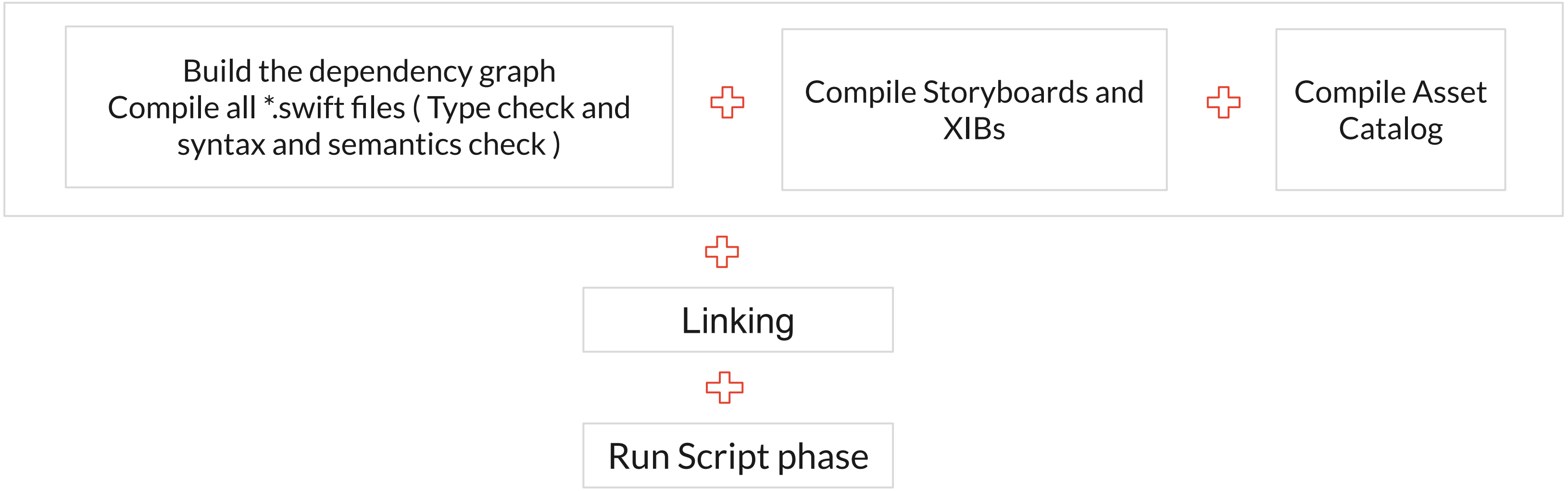


Swift FrontEnd



What is Build Time

The amount of time used by the compiler to transform source code to binary form.


Build the dependency graph
Compile all *.swift files (Type check and syntax and semantics check)
Compile Storyboards and XIBs
Compile Asset Catalog
Linking
Run Script phase

Tips and Techniques to speedup the build time

Parallelize build

Understand Dependency Graph

Understand and Optimize Build Settings and Build Phases

Type Inference impact on build times

What is Dependency graph

Understand dependency graph will give better estimation of build times.

Change the function body of swift file	Will not compile the any of the dependencies as swift is type safe
Add new function or add new struct in swift file	Swift conservatively build all the files depends on the swift file
Change in dependent frameworks	swift builds the entire container app.

Understand Build Settings & Build Phases

What is Target ?

Compilation Mode (Whole Module and Incremental)

Explicit and Implicit dependencies

Type Inference & Impact on build time

We should provide warning if any of the expression takes more than certain threshold time.

Type inference is good but we need to be careful for a good reasons.

Easy to understand for us and for compiler too. :)

-Xfrontend -warn-long-expression-type-checking=100

Optimize Run Script phase

Run script phase will be execute every time when you build the project. We should optimize here to get lesser build times.

Use Input/Output files to cache the script phases to not to execute for incremental builds.

Swift & Objective C projects

Limit Objc inference

Shrink the Bridging header and Swift generated header copy

Steps to get better build times

Changes the compilation mode from whole module to incremental

Optimize the build script execution

Reduce complex expression in incremental fashion (1000ms, 500ms, 250ms)

Visualize dependency graph of your project and change

Careful if you are using framework based architecture, take a close look at the where files should reside

Pre build modules and share the cache for larger projects (buck by FB)

Swiggy App Build Times

Changes the compilation mode from whole module to incremental	**Faster Incremental builds**
Optimize the build script execution	**~ 20 Seconds**
Reduce complex expression in incremental fashion (1000ms, 500ms, 250ms)	**~ 40 Seconds**
Pre build modules and share the cache for larger projects (buck by FB) Working on it	
Clean Build	**~ 250 Seconds (4 Minutes)**
Incremental Build	**10 ~ 40 Seconds**

Demo