

# Ruthlessly Simple Dependency Management with Carthage

# What



# Why



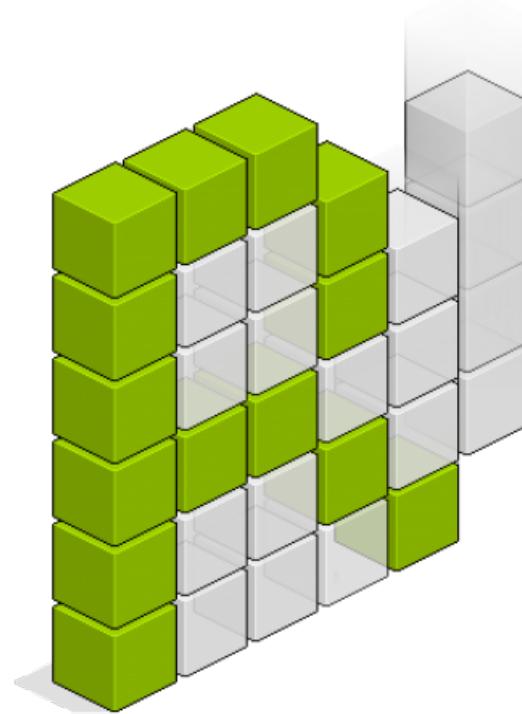
# How to use it



# How it works

# Justin Spahr-Summers

## @jspahrsummers



The Problem

# Dependencies in Cocoa (historically)

1. Copied source files
2. Zipped binaries
3. SVN externals, Git submodules
4. Git subtrees

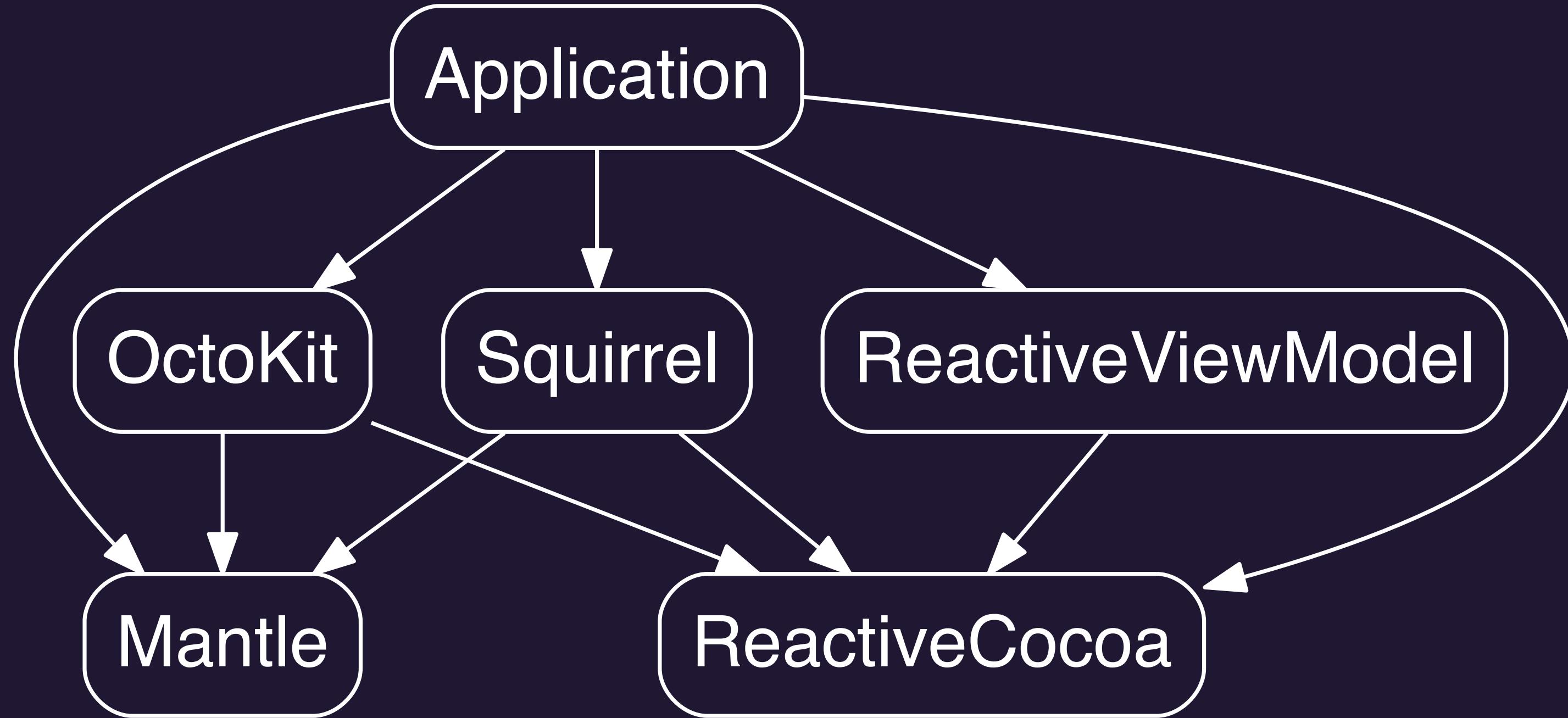
# Dependencies on other platforms

## I. Dependency managers

Our Problem

GitHub for Mac has what  
could be called "excessively  
nested submodules."

Me, late 2014



Why not use  
cocoapods?

Podspecs

Less control

# Centralized



@robrix

---

@mdiep

---

@keithduncan

---

@alanjrogers

# Our goals



1. Pick compatible versions for all dependencies
2. Check out dependencies with Git
3. Build frameworks with Xcode

# Dynamic frameworks vs. static libraries

- iOS 8+ only
- Can include resources
- Self-contained and ready-to-use
- Avoid duplicate symbol errors
- Required for Swift

# Using Carthage

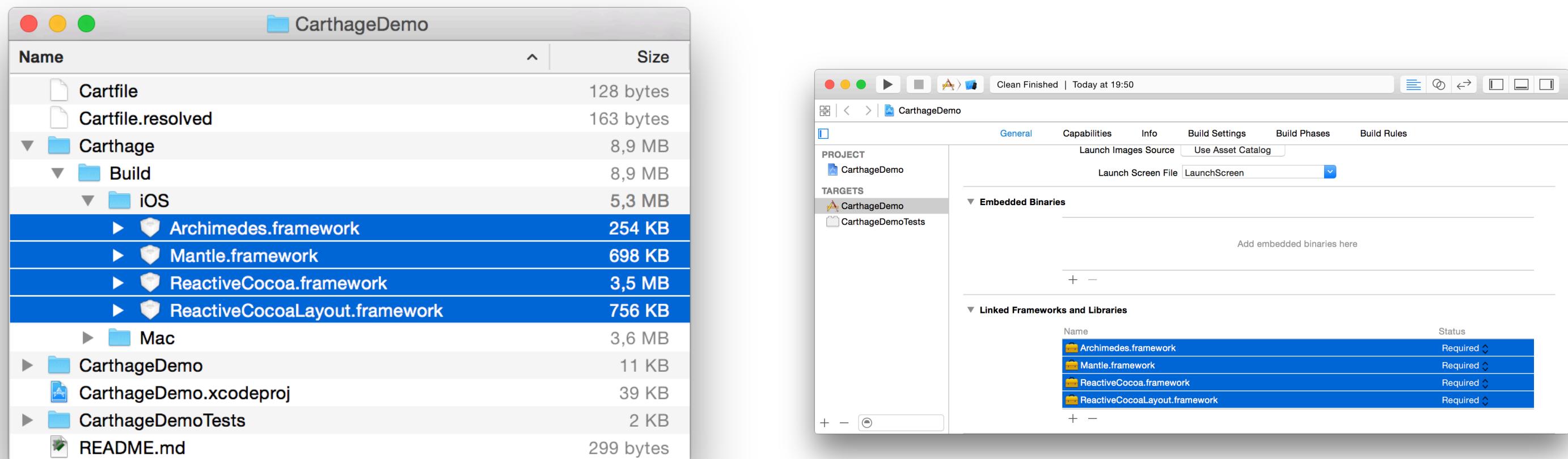
# Step 1: Create a Cartfile

```
github "Mantle/Mantle" ~> 1.5
github "ReactiveCocoa/ReactiveCocoa" ~> 2.4.7
github "ReactiveCocoa/ReactiveCocoaLayout" == 0.5.2
```

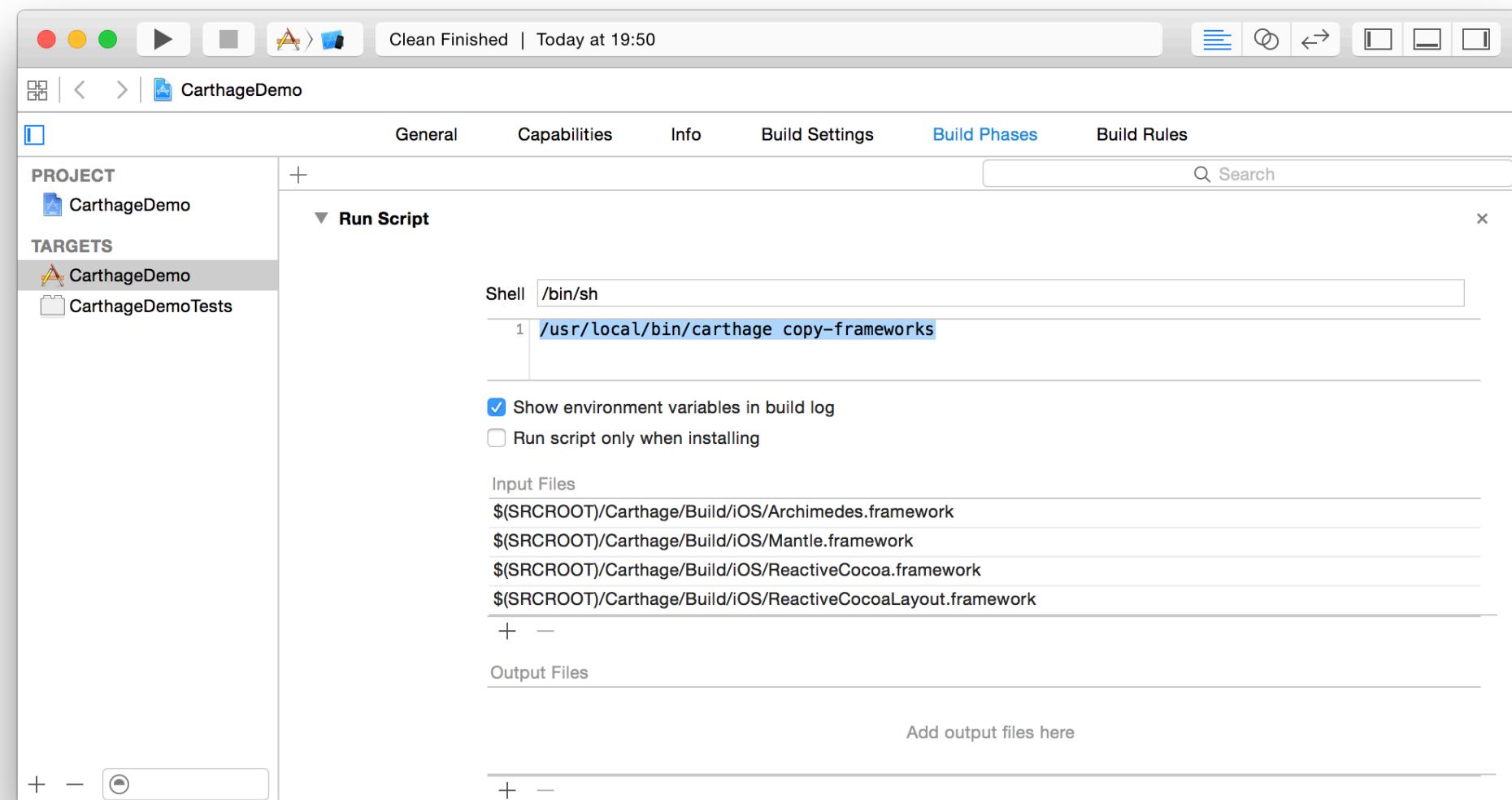
# Step 2: Run Carthage

```
$ carthage update
*** Fetching Mantle
*** Fetching ReactiveCocoa
*** Fetching ReactiveCocoaLayout
*** Fetching Archimedes
*** Downloading Archimedes at "1.1.4"
*** Downloading Mantle at "1.5.4"
*** Downloading ReactiveCocoa at "v2.4.7"
*** Downloading ReactiveCocoaLayout at "0.5.2"
```

# Step 3: Link Frameworks



# Step 4: Strip architectures (iOS only)



That's it!

“Ruthlessly Simple”

Easy: familiar or approachable

---

Simple: fewer concepts and concerns<sup>1</sup>

<sup>1</sup> See Rich Hickey's talk, [Simple Made Easy](#)

# CocoaPods is easy



# Carthage is simple

Simpler tools are...

Easier to maintain

Easier to understand

Easier to contribute to

More flexible

---

More composable

Enhanced when other tools improve

Behind the Scenes

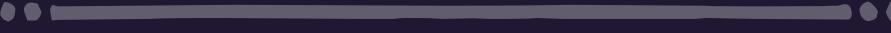
# Parse the Cartfile



## Resolve the dependency graph



## Download all dependencies



## Build each framework



# Parsing

Parse OGDL into a list of dependencies

```
github "ReactiveCocoa/ReactiveCocoa" ~> 2.4.7
```



# Parsing

**Parse OGDL into a list of dependencies**

```
github "ReactiveCocoa/ReactiveCocoa" ~> 2.4.7
```

**Determine the type of each dependency**

```
github "ReactiveCocoa/ReactiveCocoa"
```



# Parsing

**Parse OGDL into a list of dependencies**

```
github "ReactiveCocoa/ReactiveCocoa" ~> 2.4.7
```

**Determine the type of each dependency**

```
github "ReactiveCocoa/ReactiveCocoa"
```

**Parse any version constraint**

```
~> 2.4.7
```

 Resolving

I. Create a graph of the latest dependency versions

 Resolving

1. Create a graph of the latest dependency versions
2. Insert dependency Cartfiles into the graph

 Resolving

1. Create a graph of the latest dependency versions
2. Insert dependency Cartfiles into the graph
3. If requirements conflict, throw out the graph



# Resolving

1. Create a graph of the latest dependency versions
2. Insert dependency Cartfiles into the graph
3. If requirements conflict, throw out the graph
  - Try a new graph with the next possible version



# Resolving

1. Create a graph of the latest dependency versions
2. Insert dependency Cartfiles into the graph
3. If requirements conflict, throw out the graph
  - Try a new graph with the next possible version
4. Repeat until a valid graph is found



# Downloading

I. Fetch the repository into Carthage's cache



# Downloading

1. Fetch the repository into Carthage's cache
2. Copy the right version into Carthage/Checkouts



# Building

I. Symlink Carthage/Build into the dependency folder



# Building

1. Symlink Carthage/Build into the dependency folder
2. List framework schemes from the .xcodeproj



# Building

1. Symlink Carthage/Build into the dependency folder
2. List framework schemes from the .xcodeproj
3. Build each scheme for all supported architectures



# Building

1. Symlink Carthage/Build into the dependency folder
2. List framework schemes from the .xcodeproj
3. Build each scheme for all supported architectures
4. Combine multiple architectures with lipo



# Building

1. Symlink Carthage/Build into the dependency folder
2. List framework schemes from the .xcodeproj
3. Build each scheme for all supported architectures
4. Combine multiple architectures with lipo
5. Copy the built frameworks into Carthage/Build



# BONUS: Prebuilt binaries!



ReactiveCocoa / ReactiveCocoa

Unwatch 506 Unstar 7,339 Fork 1,038

Releases Tags Edit release Delete

Latest release v2.4.7

v2.4.7 jspahrsummers released this on Feb 11 · 16 commits to master since this release  
Includes minor fixes #1707, #1734, and #1741.

Downloads

ReactiveCocoa.framework.zip 1.86 MB

Source code (zip)

Source code (tar.gz)

\*\*\* Downloading ReactiveCocoa at "v2.4.7"

CarthageKit

# Technical choices

# Swift vs. Objective-C

- Type safety
- Value types (especially enums)
- Much faster to write
- Better modularization
- The Next Big Thing™

# ReactiveCocoa

- Simplifies networking (with the GitHub API)
- Simplifies the dependency resolver
- Simplifies shelling out, via ReactiveTask
- Carthage helps test RAC 3.0 in the real world



# Per-project settings

# Review CarthageKit API

---

Review command line flags

Profit!!!



# What



# Why



# How to use it



# How it works

# Questions? Comments?

Slides and notes, plus my demo project, are available at:

<https://github.com/jspahrsummers/carthage-talk>

Thanks to everyone who reviewed this presentation,  
and to Realm for inviting me to speak!<sup>2</sup>

---

<sup>2</sup> ❤ Matt Diephouse, Rob Rix, Alan Rogers, Keith Duncan, Nacho Soto, Tom Brow, James Lawton, Arwa Jumkawala, JP Simard, Tim Anglade, Brendan Forster, Benjamin Encz, Robert Böhnke, and you!