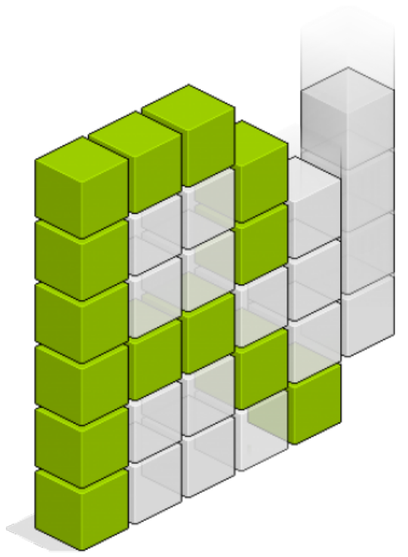
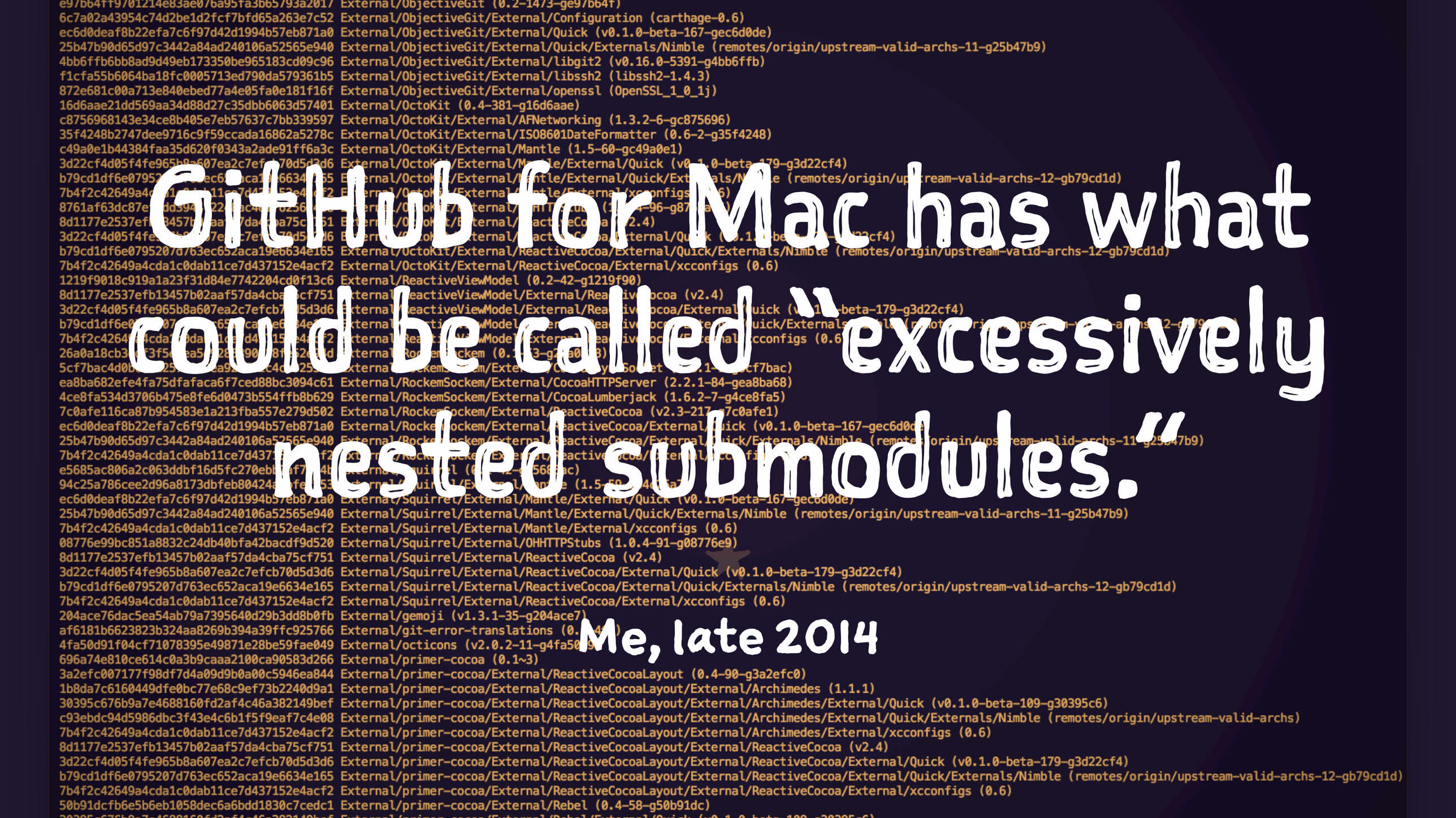
Ruthlessly Simple Dependency Management with Carthage

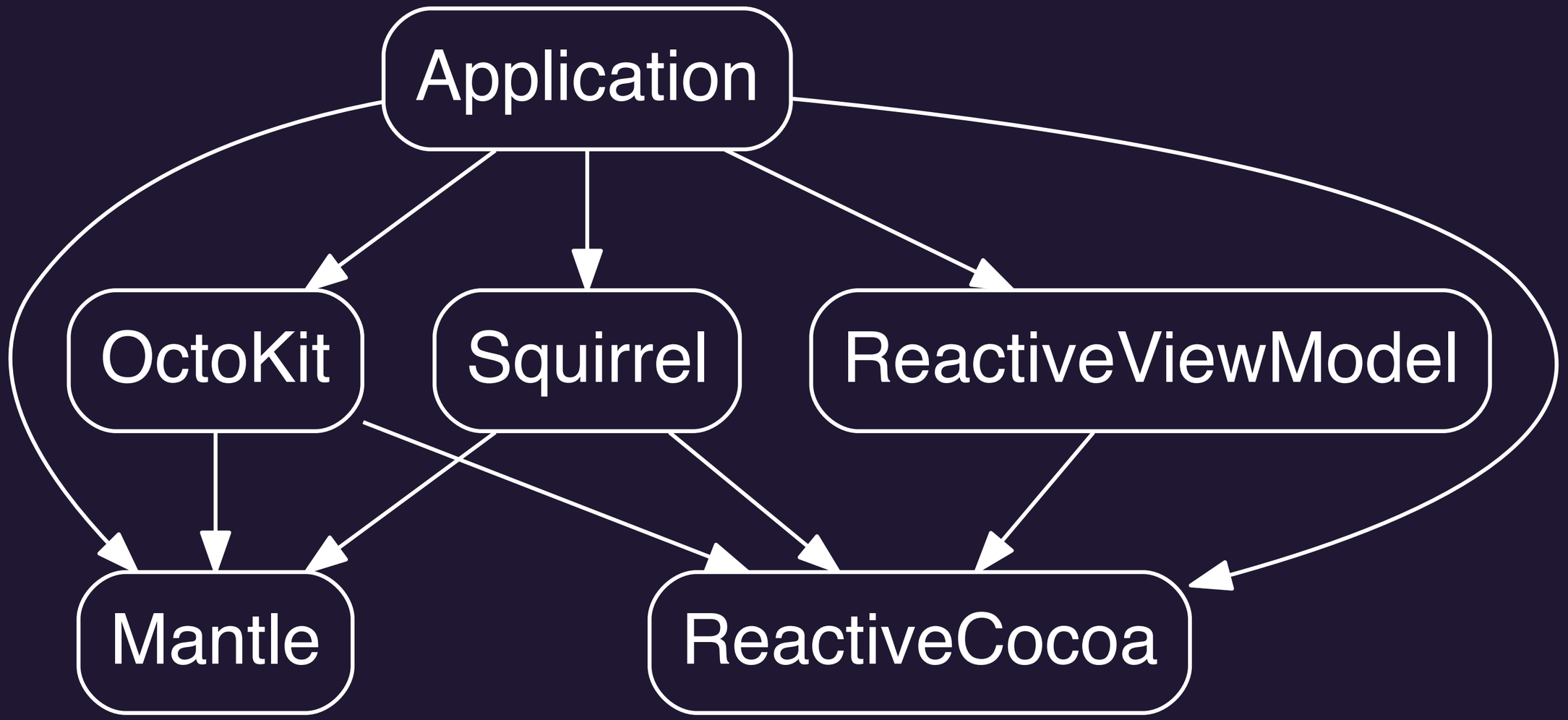
Justin Spahr-Summers

@jspahrsummers



The Problem


GitHub for Mac has what could be called "excessively nested submodules."
Me, late 2014


Application
OctoKit
Squirrel
ReactiveViewModel
Mantle
ReactiveCocoa

Why not use CocoaPods?

Why not use CocoaPods?

Podspecs

Why not use CocoaPods?

Less control over integration

Why not use CocoaPods?

Requires a central authority

Why not use CocoaPods?

Written in Ruby

@robrix

@mdiep

@keithduncan

@alanjrogers


XCODE

What does Carthage do?

1. Picks a compatible version of each dependency
2. Checks out each dependency with Git
3. Builds each dependency with Xcode
4. Gives you ready-to-use framework binaries

How do you use it?

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"
*** xcodebuild output can be found in /var/folders/t6/tjsdgjqd6j7_vjgb66qvwlb80000gn/T/carthage-xcodebuild.lisVLC.log
```

Step 3: Linked Frameworks and Libraries

Step 4: Strip architectures (iOS only)

That's it!

"Ruthlessly simple"

Simple vs. Easy

Easy: *familiar* or *approachable*

Simple: fewer concepts and concerns

See Rich Hickey's talk, "Simple Made Easy"

Simple vs. Easy

CocoaPods is *easy*

Carthage is *simple*

Simpler tools are...

Easier to maintain

Simpler tools are...

Easier to understand

Simpler tools are...

Easier to contribute to

Simpler tools are...

More flexible and composable

Simpler tools are...

Automatically made better as their integration points get better

How does it work behind the scenes?

`carthage update` proceeds through the following steps:

1. Parse the Cartfile
2. Resolve the dependency graph
3. Download and check out all dependencies
4. Build all dependencies

Parsing the Cartfile

1. Parse OGDL
2. Break down into a list of dependencies
3. Determine the type of dependency (`github` or `git`)
4. Parse any version constraint

Resolving the dependency graph

1. Propose a graph using the latest allowed version for all dependencies
2. Incorporate any Cartfiles for those dependencies (at those versions!) into the graph
3. If the graph is no longer valid, throw it out and try with the next possible version
4. Rinse and repeat until a valid graph is found

Downloading a dependency

1. Fetch the repository into a global Carthage cache
2. Run a modified `git checkout` to copy the right version of the repository into Carthage/Checkouts

Building a dependency

1. Symlink the Carthage/Build folder from the application project into the dependency's folder
2. Run `xcodebuild -list` on the rootmost `xcodeproj` to find shared schemes
3. Filter out any schemes which do not build a dynamic framework

Bonus: prebuilt binaries!

```
$ 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"
```

CarthageKit

Why is it written in Swift (and not Objective-C)?

- Type safety
- Value types (especially enums)
- Much faster to write
- Better modularization
- Desire to write a complete app using The Next Big Thing

Why does it use ReactiveCocoa?

- Simplifies the invocation of shell tasks (see ReactiveTask)
- Simplifies networking (e.g., the GitHub API)
- Simplifies the dependency resolution algorithm
- Needed a real world application of RAC's Swift API

1.0

1. Per-project settings
2. CarthageKit API review
3. CLI parameter review
4. Profit!!! 💸

Questions?

Thanks to: TODO