Dissecting iOS Modularity

Arek Macudziński

arek.macudzinski@gmail.com

iOS developer

FIBARO®

HOME INTELLIGENCE


FIBARO


FIBARO


Home Center 2


2500W


RGBW Controller
RGBW Controller
FIBARO SYSTEM


9:41
Welcome Home
102 W
Consumption
21 °C
Temperature
50 %
Humidity
23 km/h
Wind
Lighting
Climate
Scenes
Blinds
Rooms
Alarm
Cameras
Plugins
D/W Sensors
Motion Sensors
Notifactions
Virtual Devices
Equipment
Videogates
Intercom


Home
Monday
18°C
13 08
22°C
OFF
44%


Fibaro
11:59
Dinner
Movie
Goodnight

iOS Modularity

WHY?

Every app uses the same code

App1
iOS

OAuth

API calls

Domain models

App1
Watch

OAuth

API calls

Domain models

App2

OAuth

API calls

Domain models

App3

OAuth

API calls

Domain models

App1
iOS

OAuth

API calls

Domain models

Networking

Storage

App1
Watch

OAuth

API calls

Domain models

Networking

Storage

App2

OAuth

API calls

Domain models

Networking

Storage

App3

OAuth

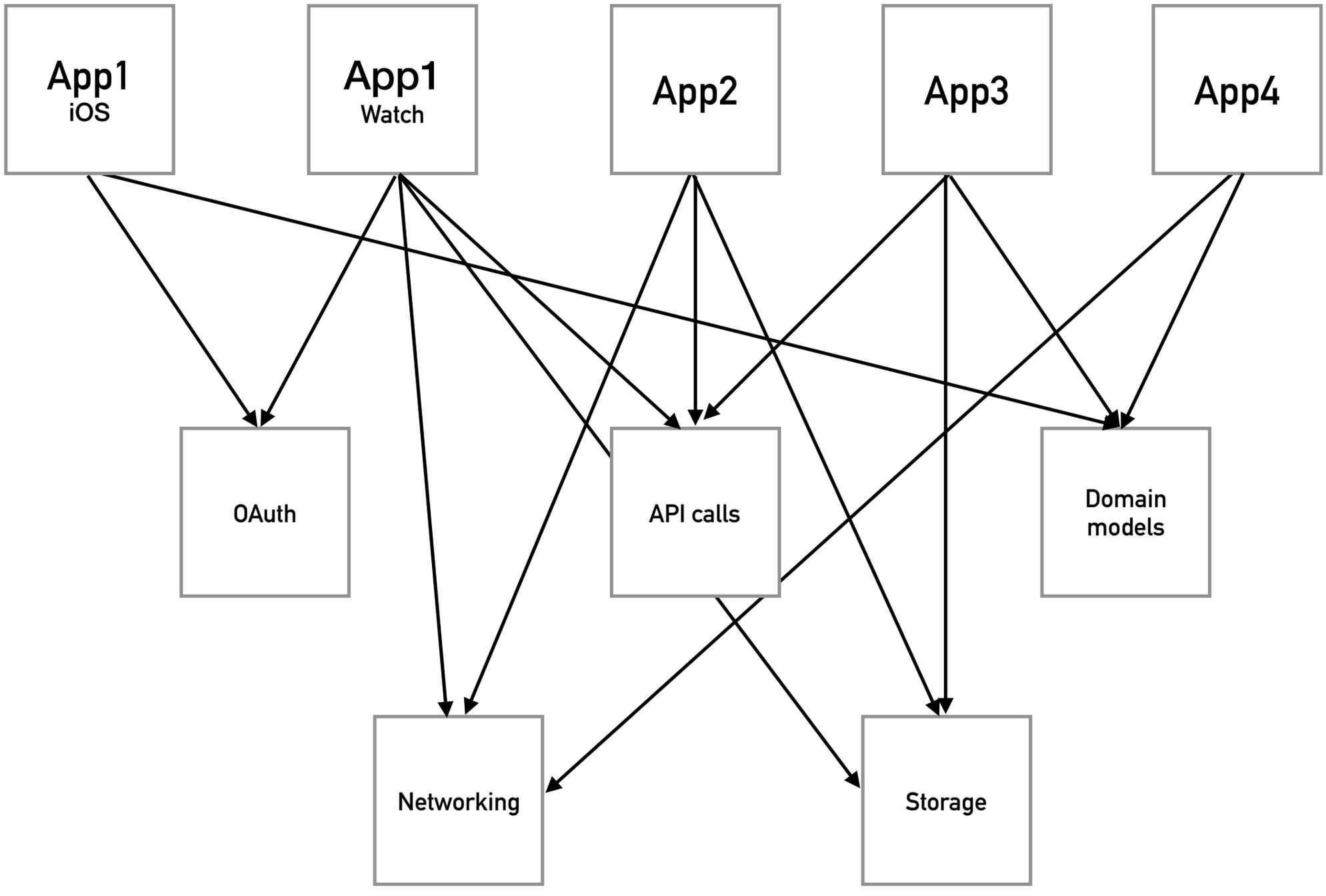
API calls

Domain models

Networking

Storage

DRY


App1
iOS
App1
Watch
App2
App3
App4
OAuth
API calls
Domain
models
Networking
Storage

Modules

- Reusable code (DRY)

Modules

- Reusable code (DRY)
- Good code separation - better architecture

Modules

- Reusable code (DRY)
- Good code separation - better architecture
- Gain development speed
- Simplify development by many teams

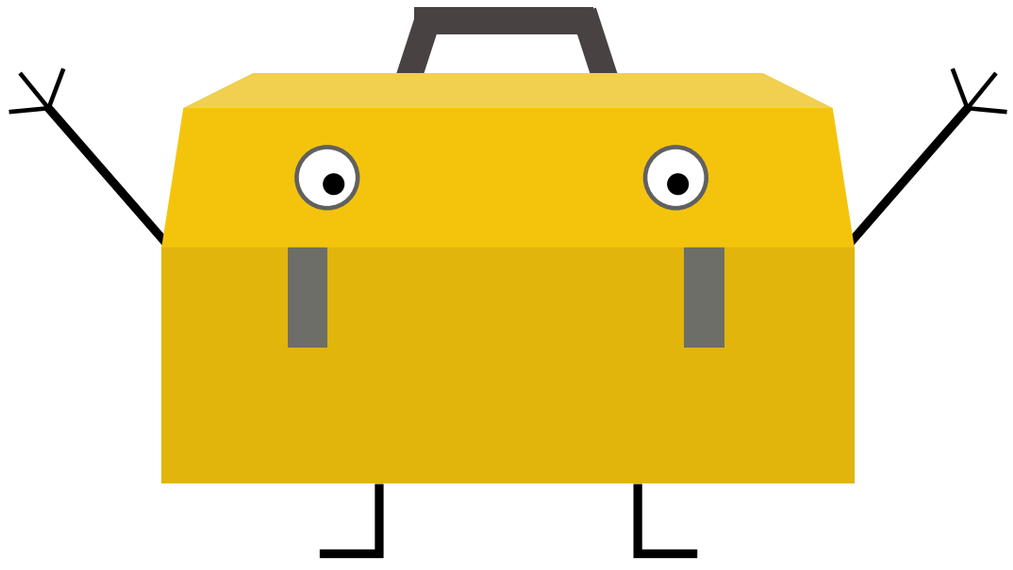
Definition of

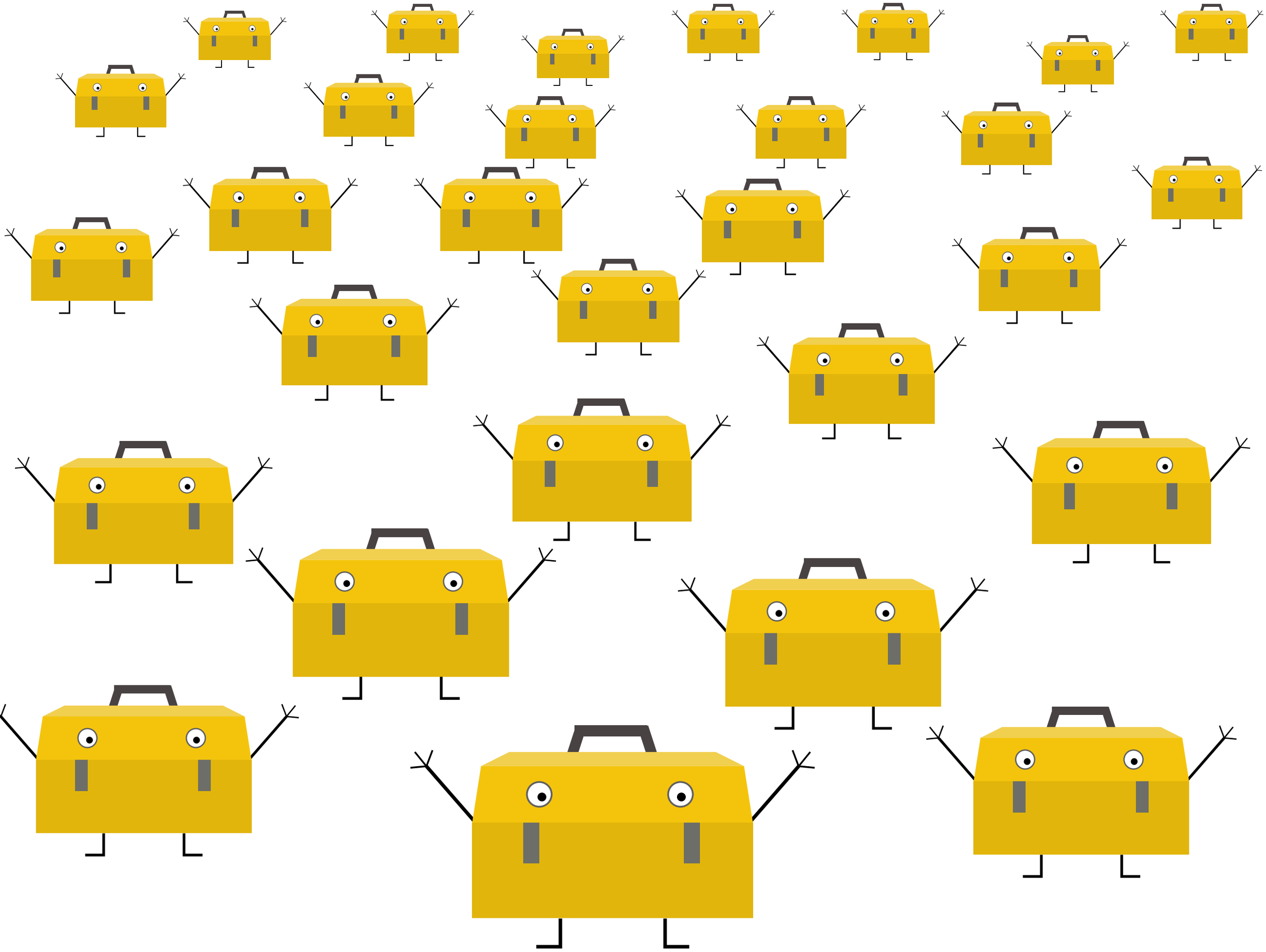
Module

A separated part of code implementing some closed functionality (features, utils, ...)

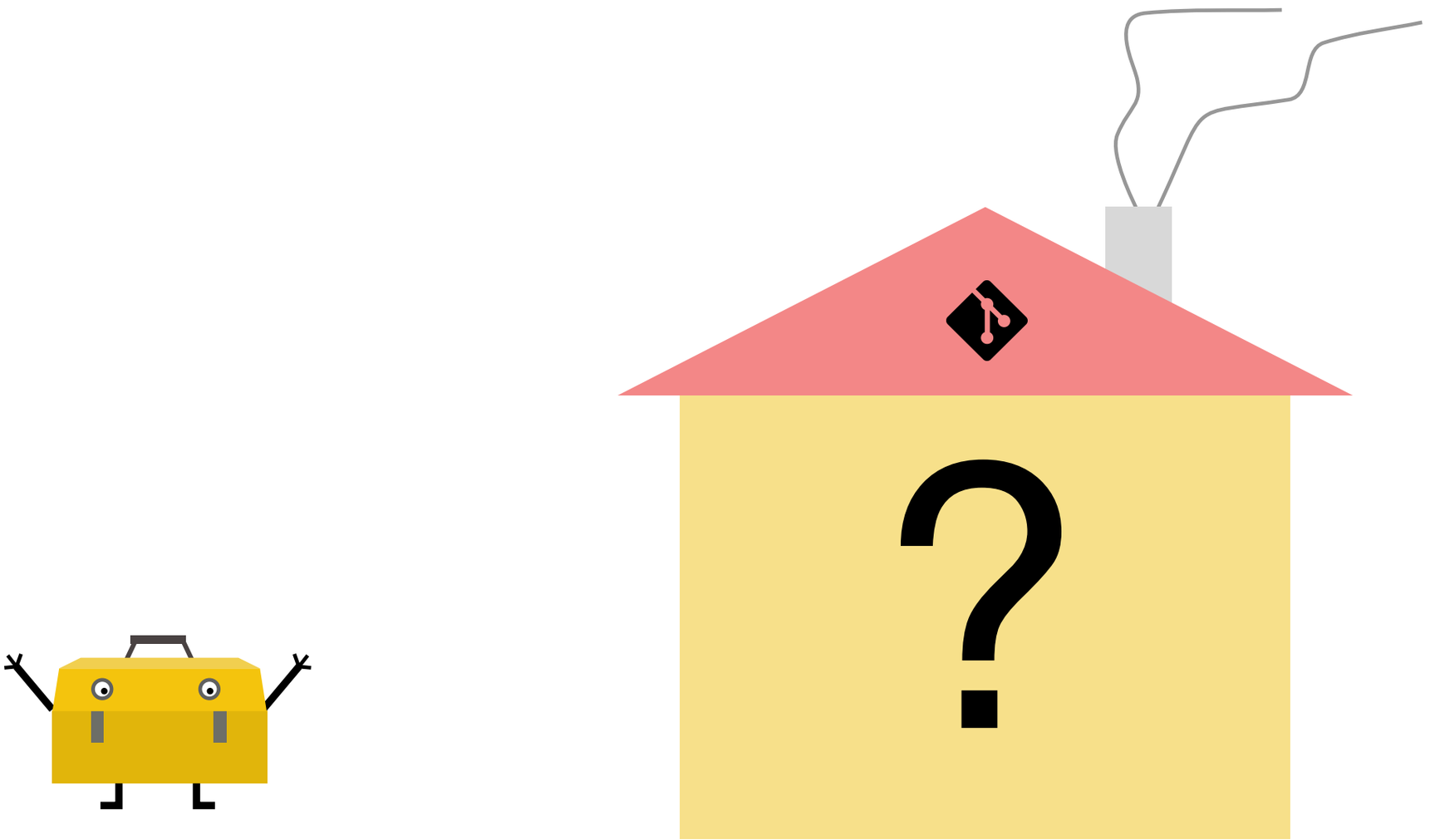
iOS Modularity

Mr Module

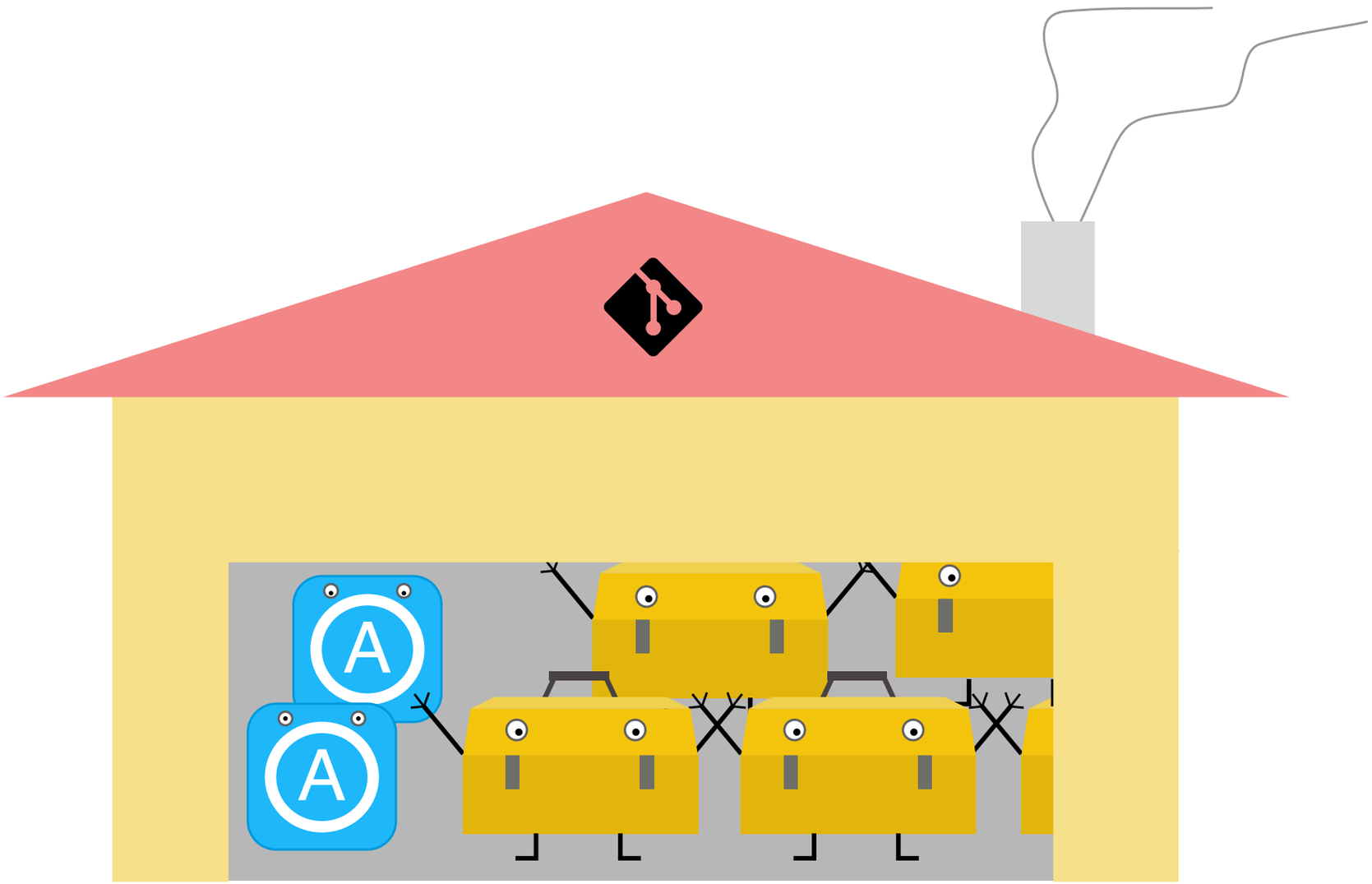




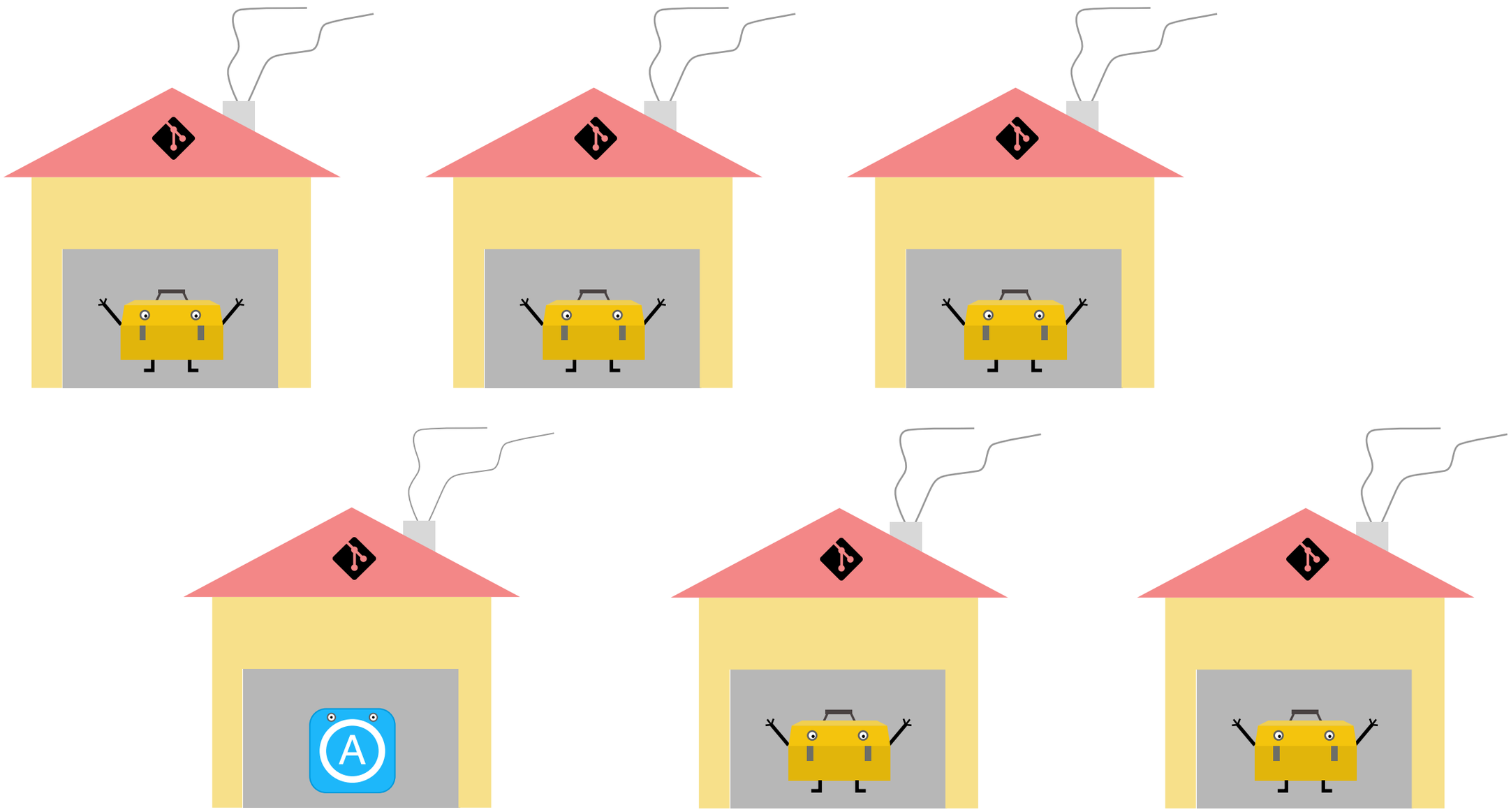
Modules go home



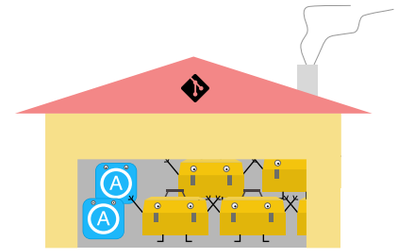
Monorepo

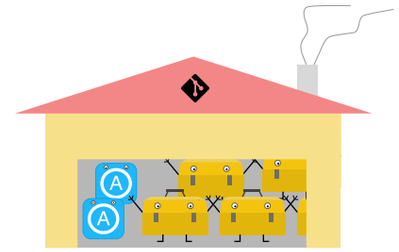


Manyrepo



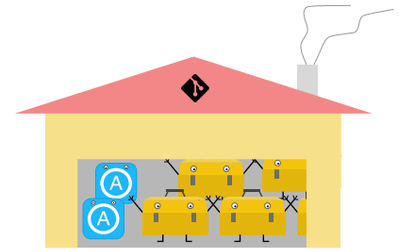
Monorepo pros





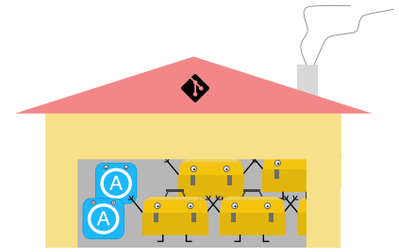
Monorepo pros

- faster development



Monorepo pros

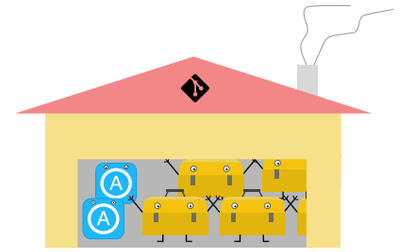
- faster development
- easier refactor

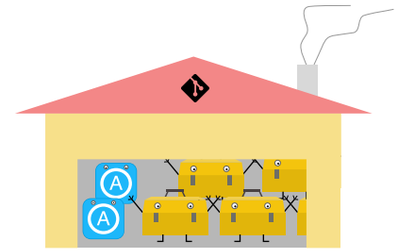


Monorepo pros

- faster development
- easier refactor
- no modules versioning

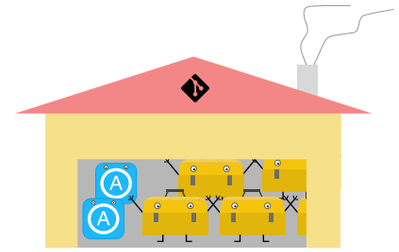
Monorepo cons





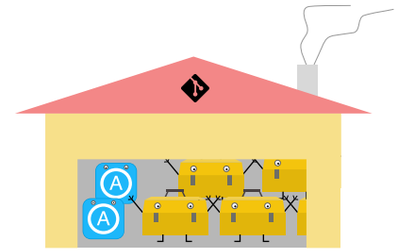
Monorepo cons

- huge repo size



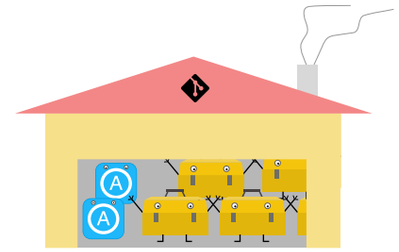
Monorepo cons

- huge repo size
- git performance slows



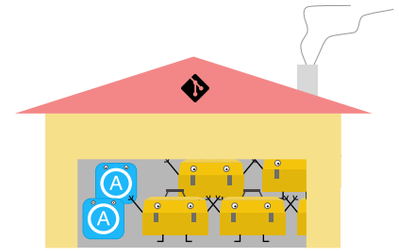
Monorepo cons

- huge repo size
- git performance slows
- long build time



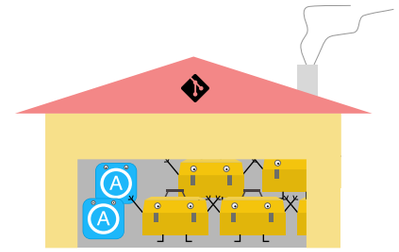
Monorepo cons

- huge repo size
- git performance slows
- long build time
- no access control



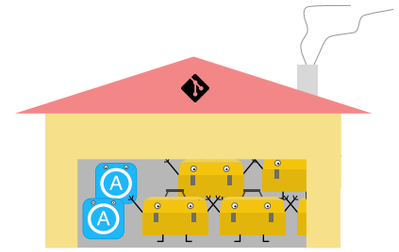
Monorepo cons

- huge repo size
- git performance slows
- long build time
- no access control
- hard app versioning



Monorepo cons

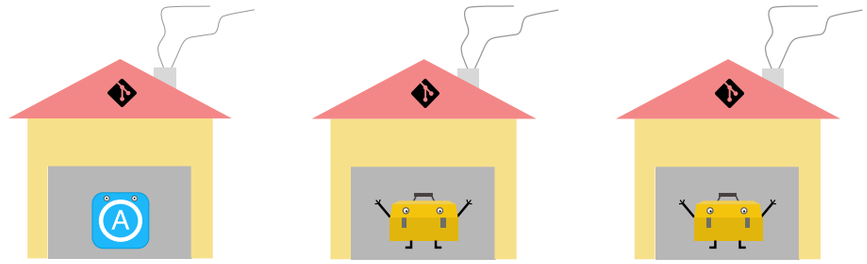
- huge repo size
- git performance slows
- long build time
- no access control
- hard app versioning
- hard dependency manger export



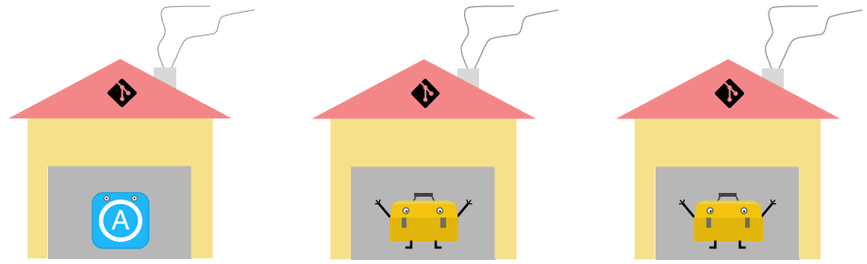
Monorepo cons

- huge repo size
- git performance slows
- long build time
- no access control
- hard app versioning
- hard dependency manger export
- regression

Manyrepo pros

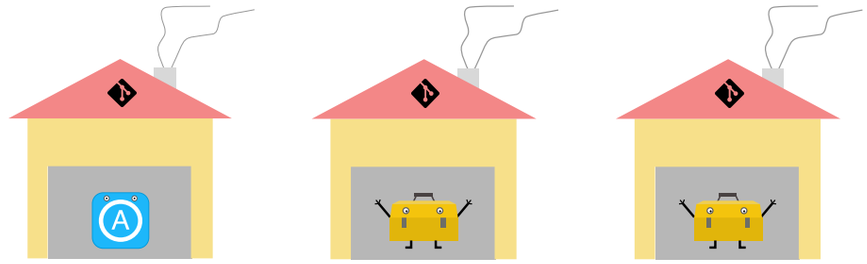


Manyrepo pros



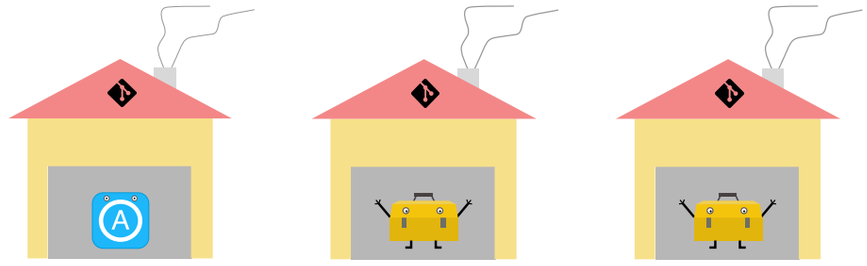
- good physical separation

Manyrepo pros



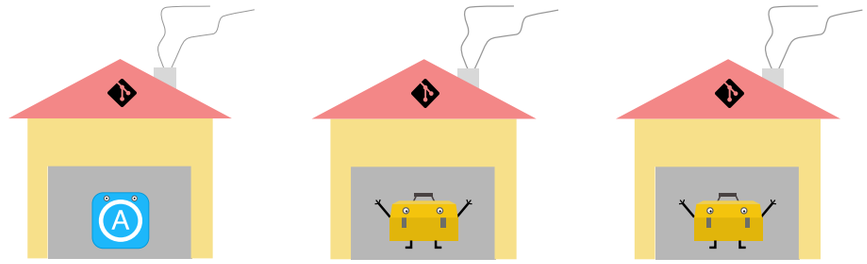
- good physical separation
- individual versioning

Manyrepo pros



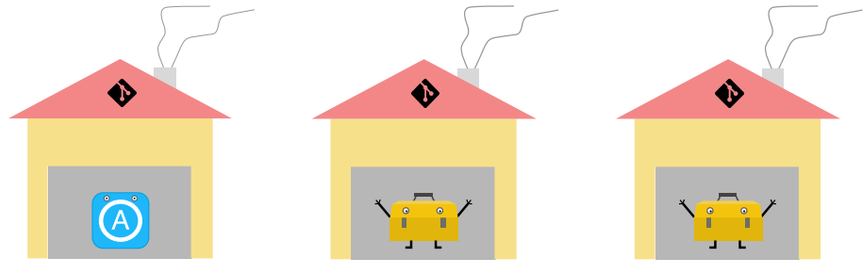
- good physical separation
- individual versioning
- dependency manager export

Manyrepo pros



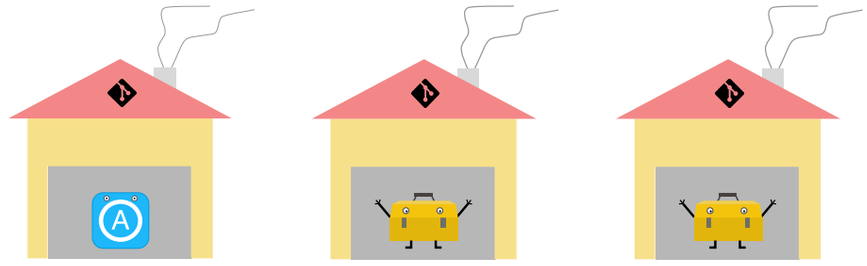
- good physical separation
- individual versioning
- dependency manager export
- access control

Manyrepo pros

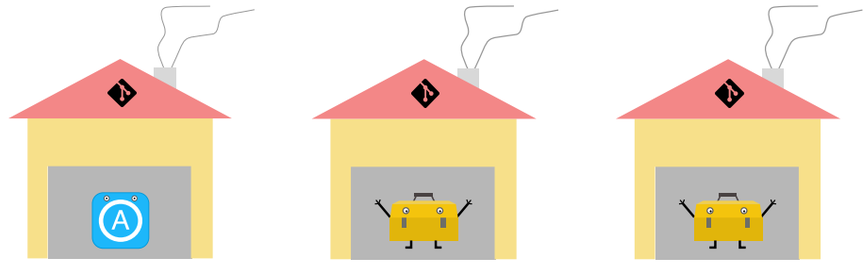


- good physical separation
- individual versioning
- dependency manager export
- access control
- good git performance

Manyrepo pros



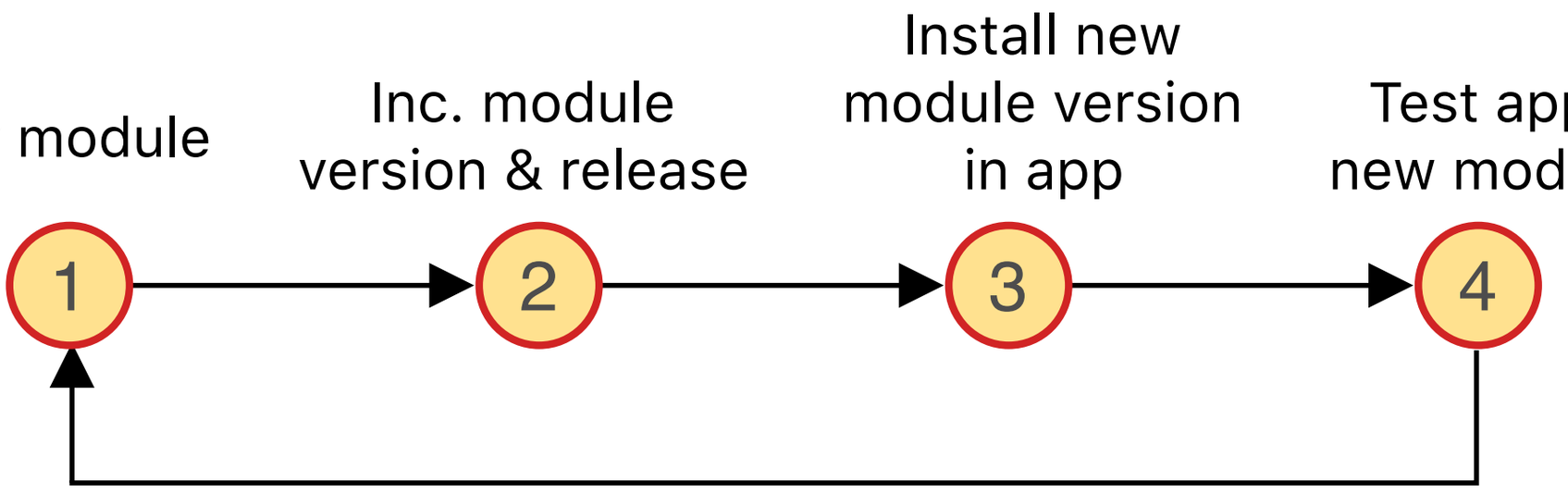
- good physical separation
- individual versioning
- dependency manager export
- access control
- good git performance
- no regressions



Manyrepo problem

- complicated workflow

Manyrepo development workflow

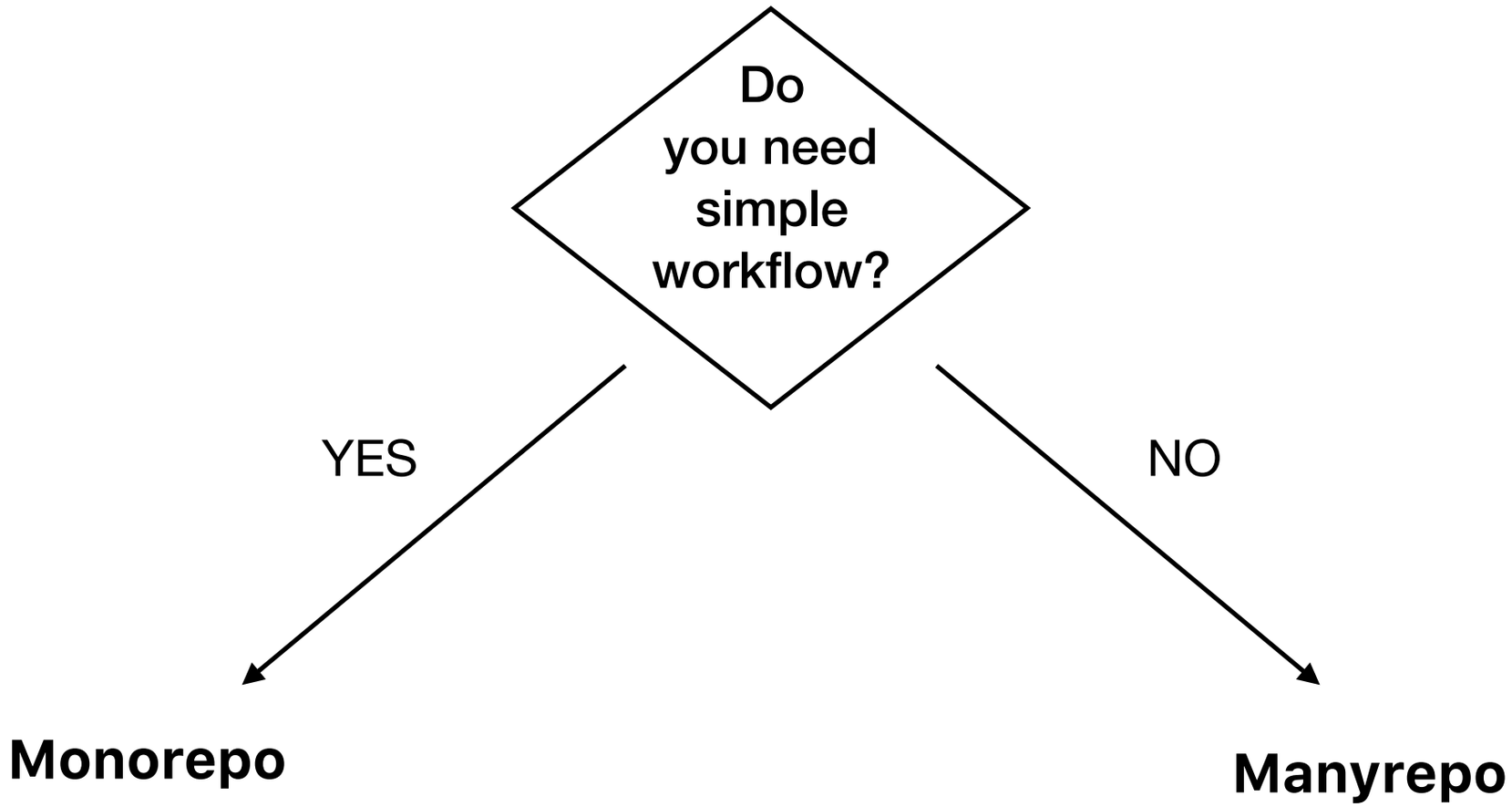


Test fail – repeat

Monorepo vs Manyrepo

?

What to choose?



What to choose?

Monorepo

- do you need simple workflow?

Manyrepo

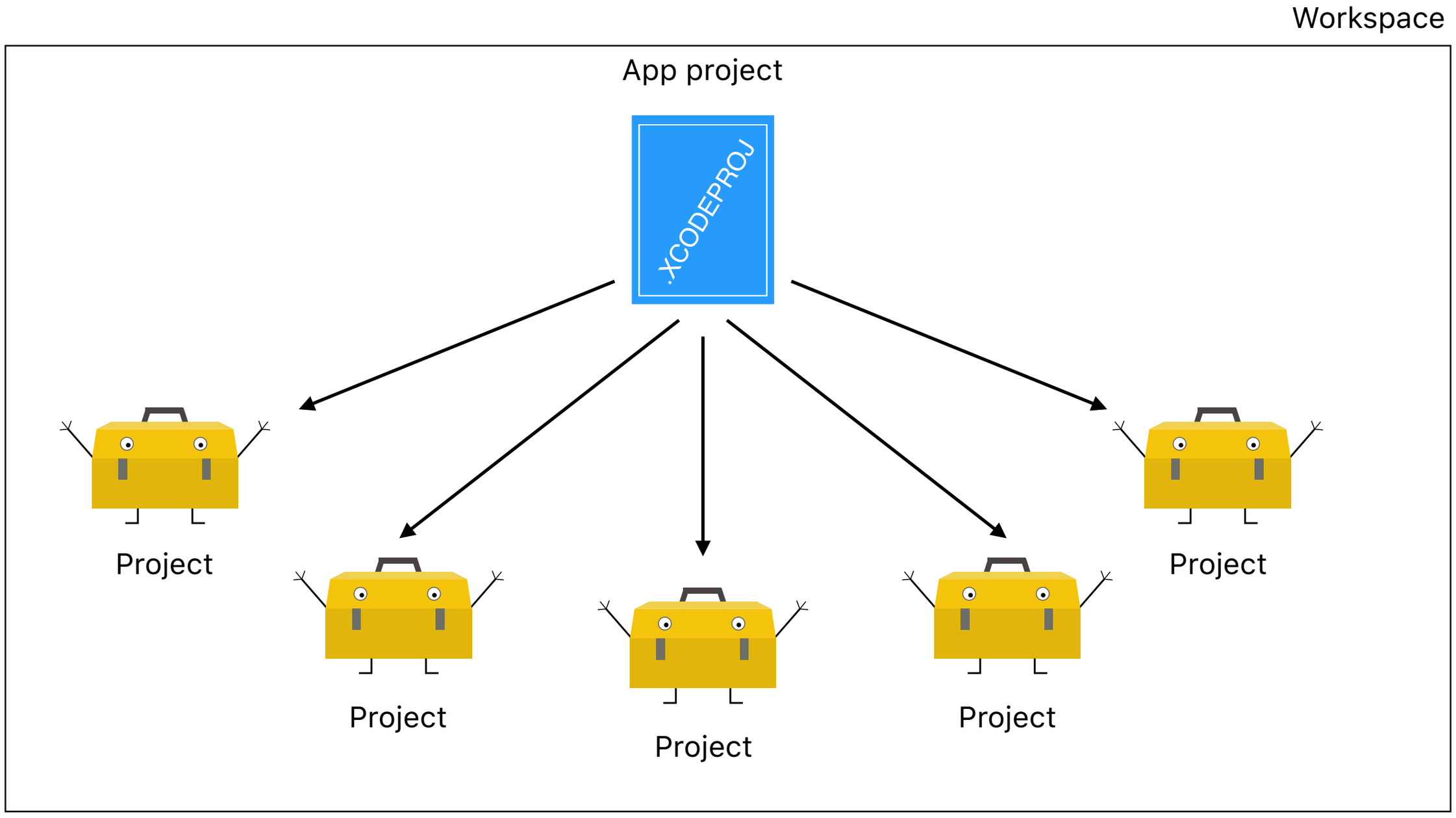
- do you have may apps?
- do you have large team?
- do you want to have strict access control?
- do you want open source?

Decide according to your needs

in any moment you can change your mind

Connection with the app

Direct

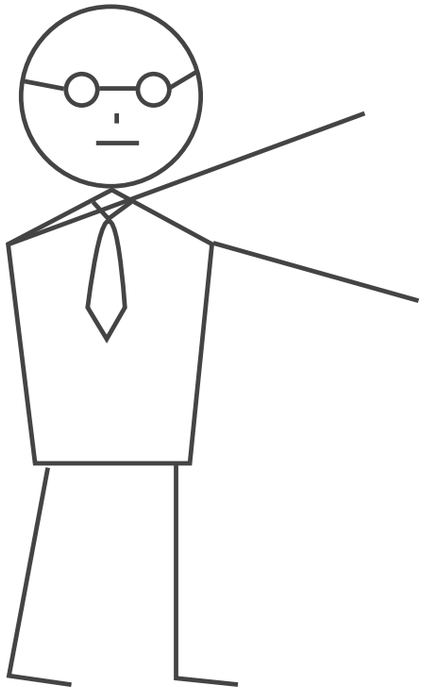

Workspace
App project
.XCODEPROJ
Project
Project
Project
Project
Project

Dependency Manager

App project

.XCODEPROJ

Dependency Manager



Module Xcode Project

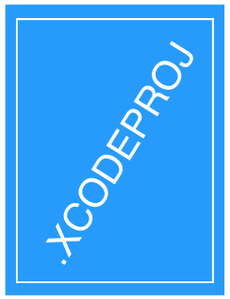
Module Xcode Project

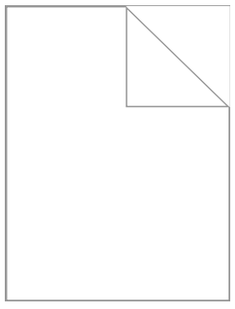
Module Xcode Project

iOS Dependency Managers

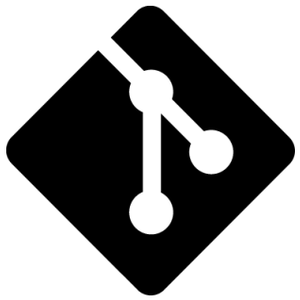
- Cocoapods
- Carthage
- Swift Package Manager

Cocoapods

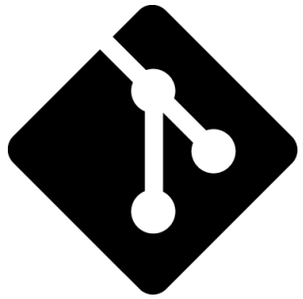




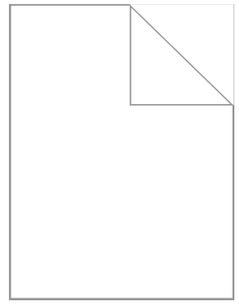
Podfile



Pod spec repo

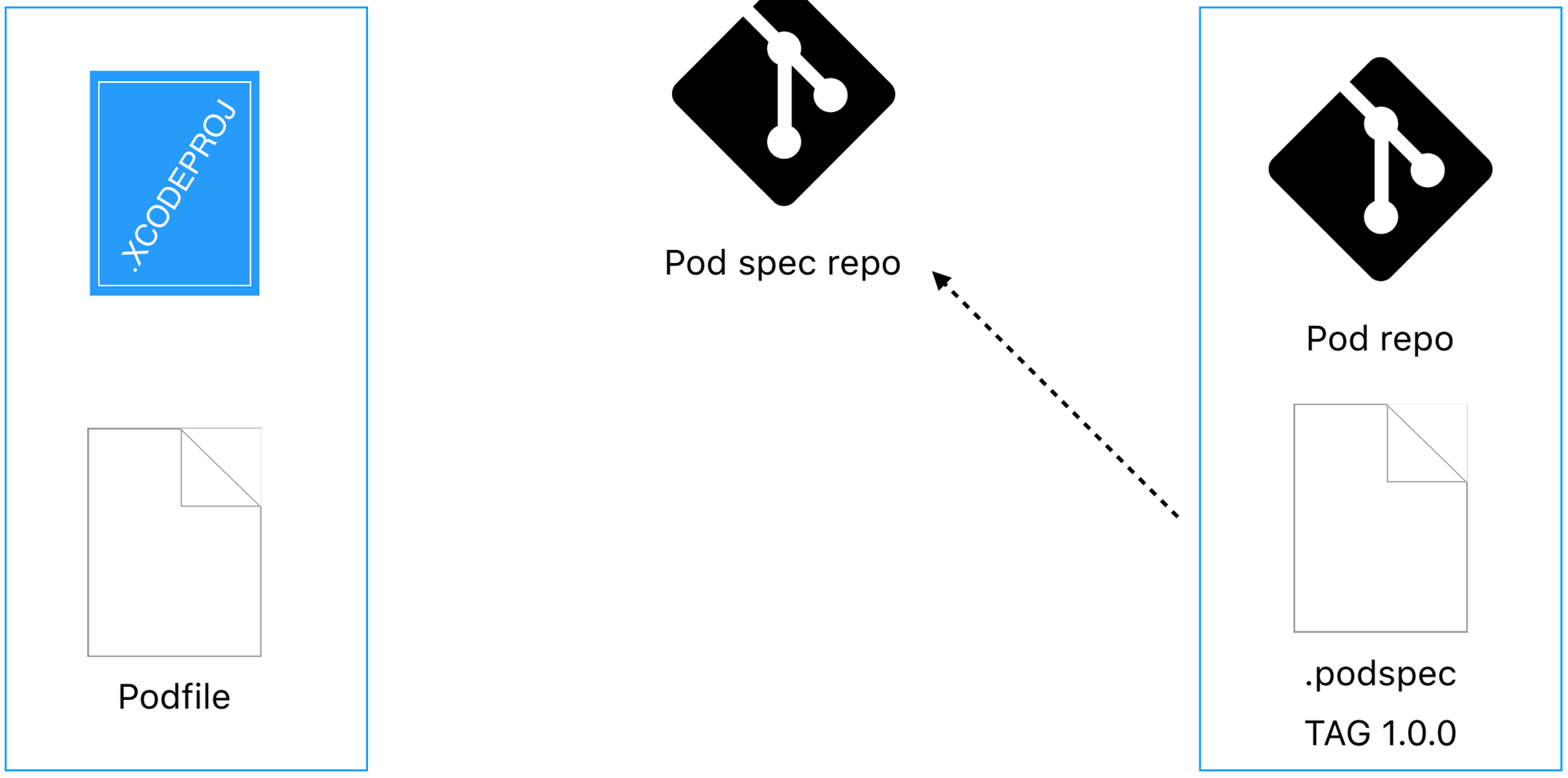


Pod repo

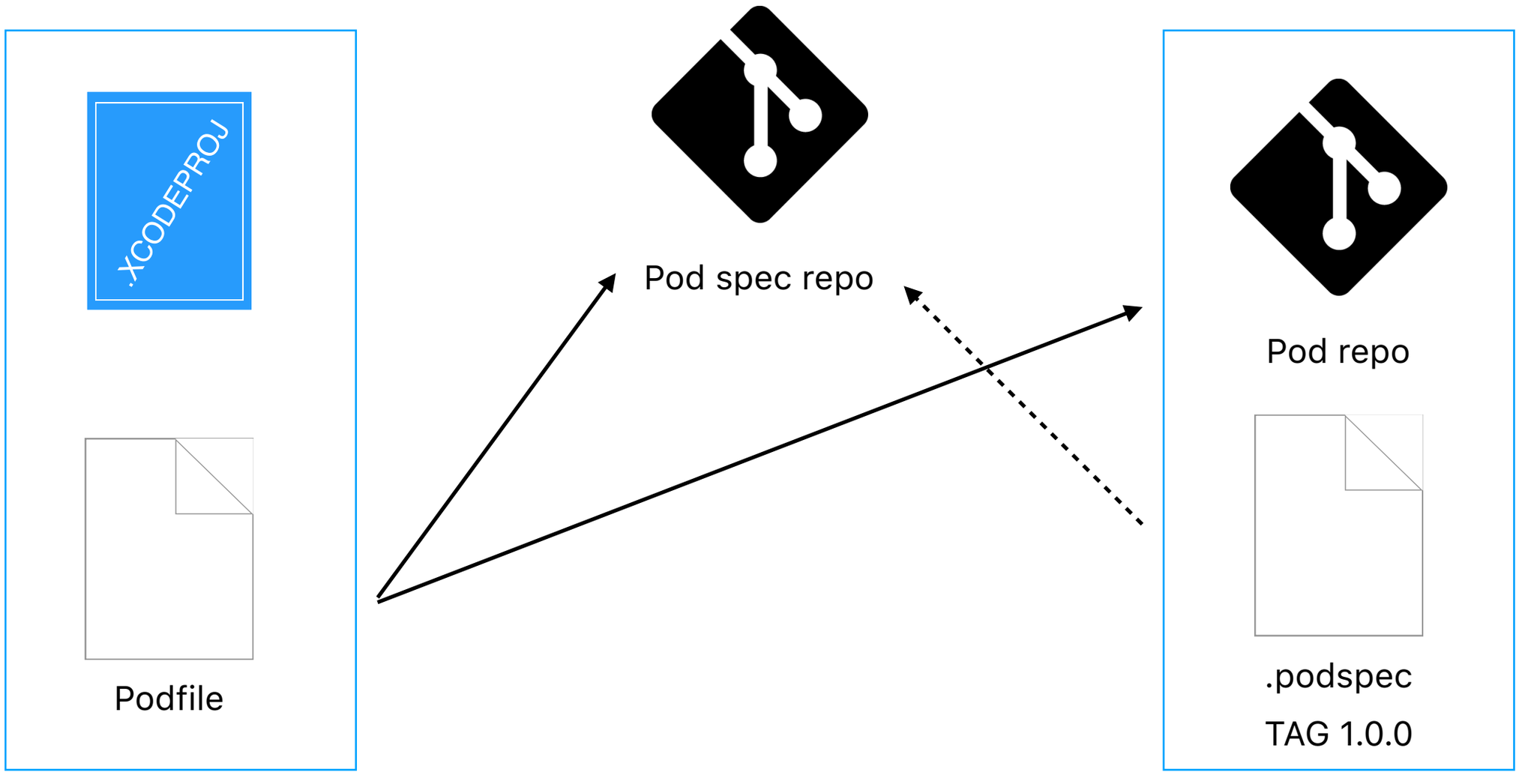


.podspec

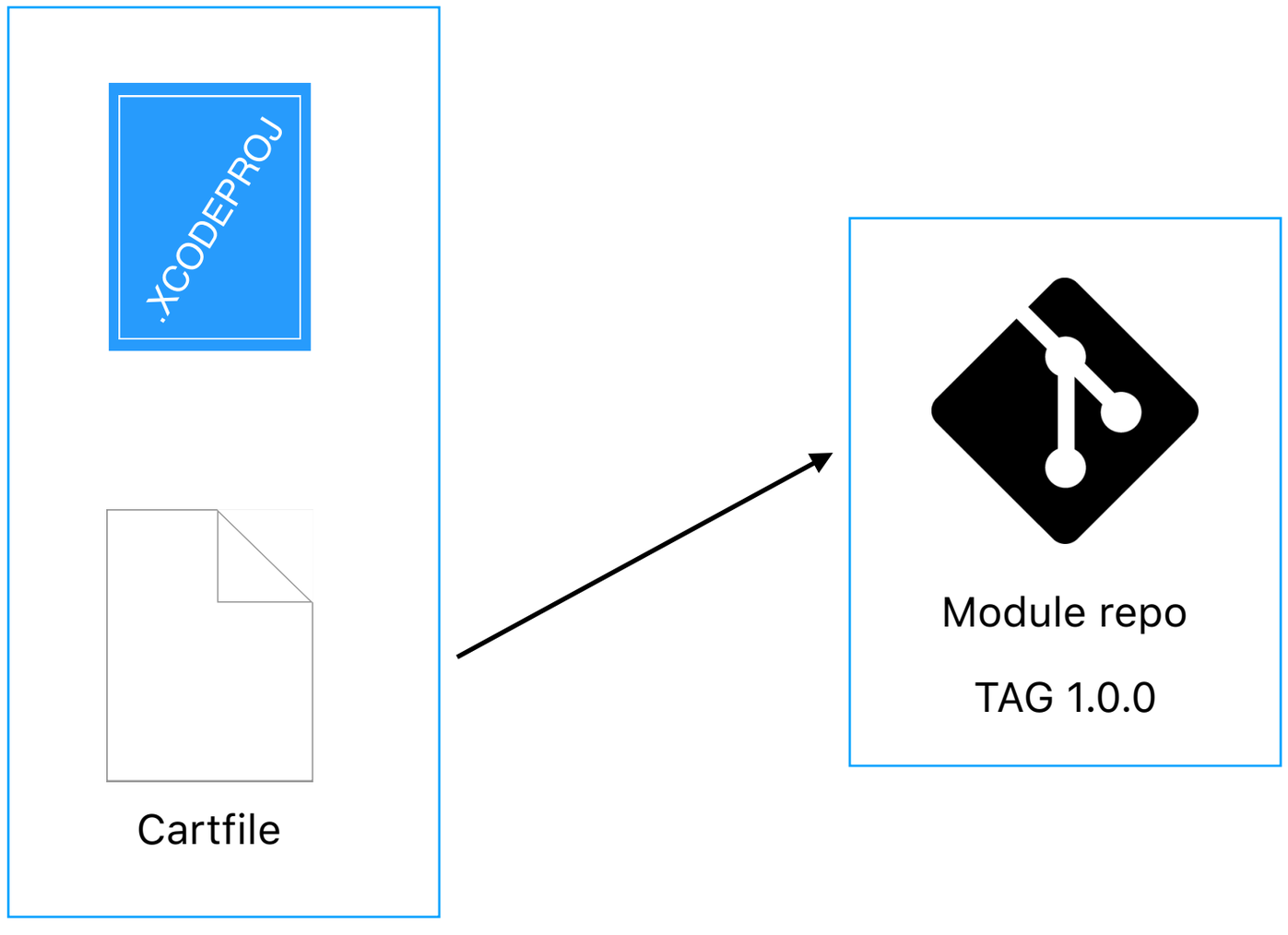
Cocoapods



Cocoapods

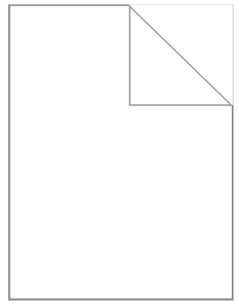


Carthage



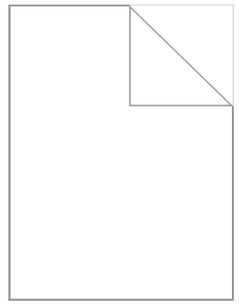
Swift Package Manager

App project



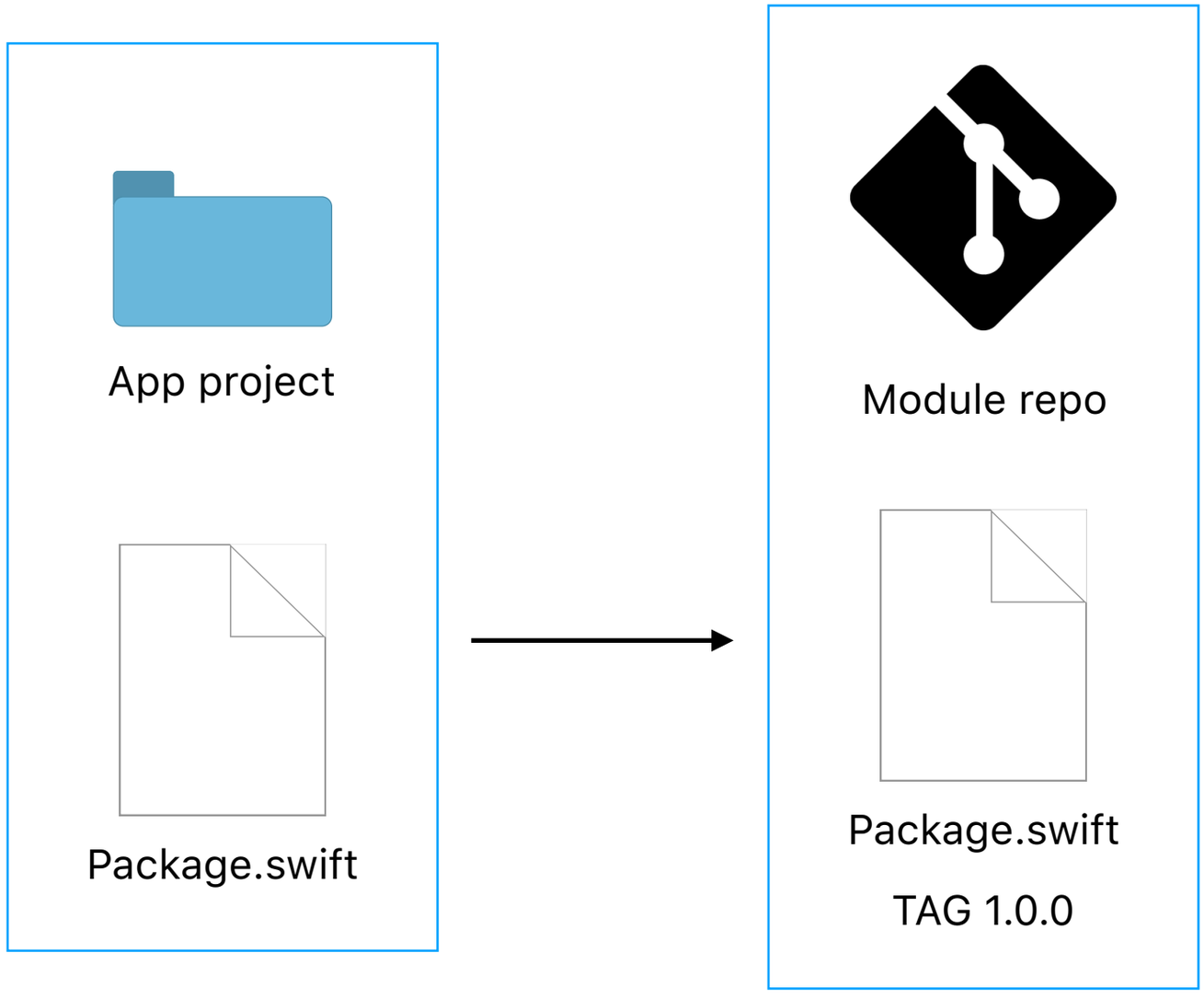
Package.swift

Module repo



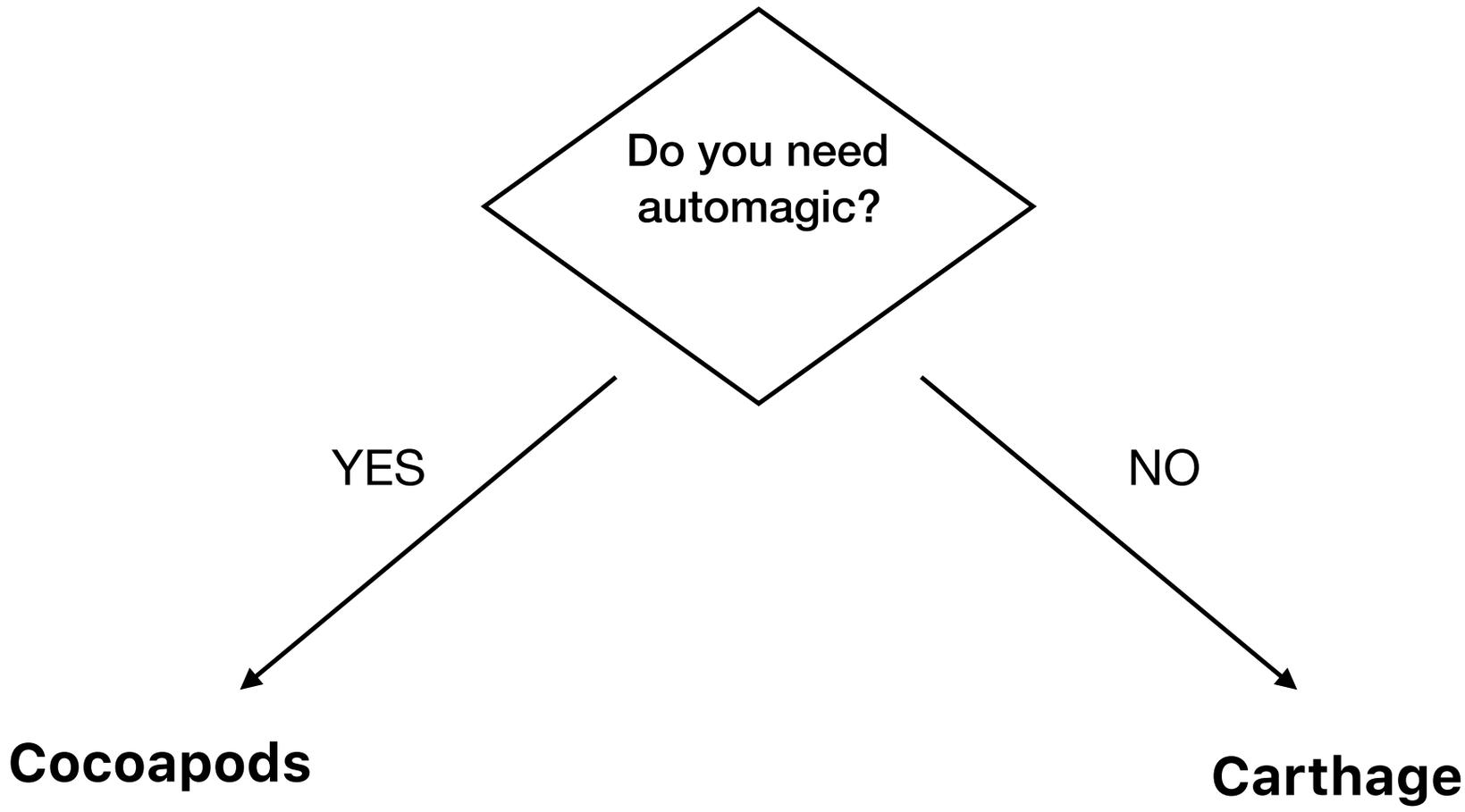
Package.swift

Swift Package Manager

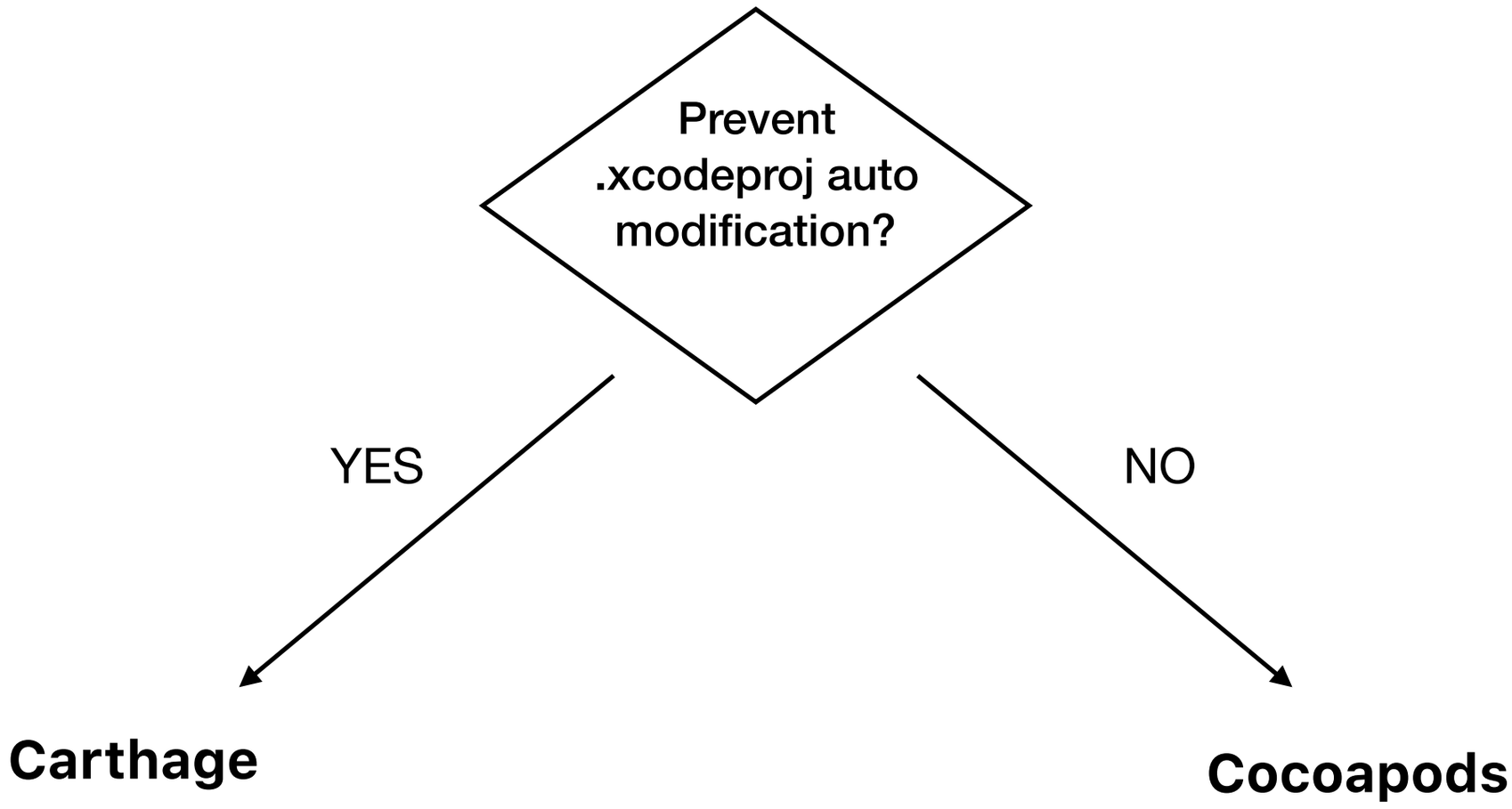


What to choose?

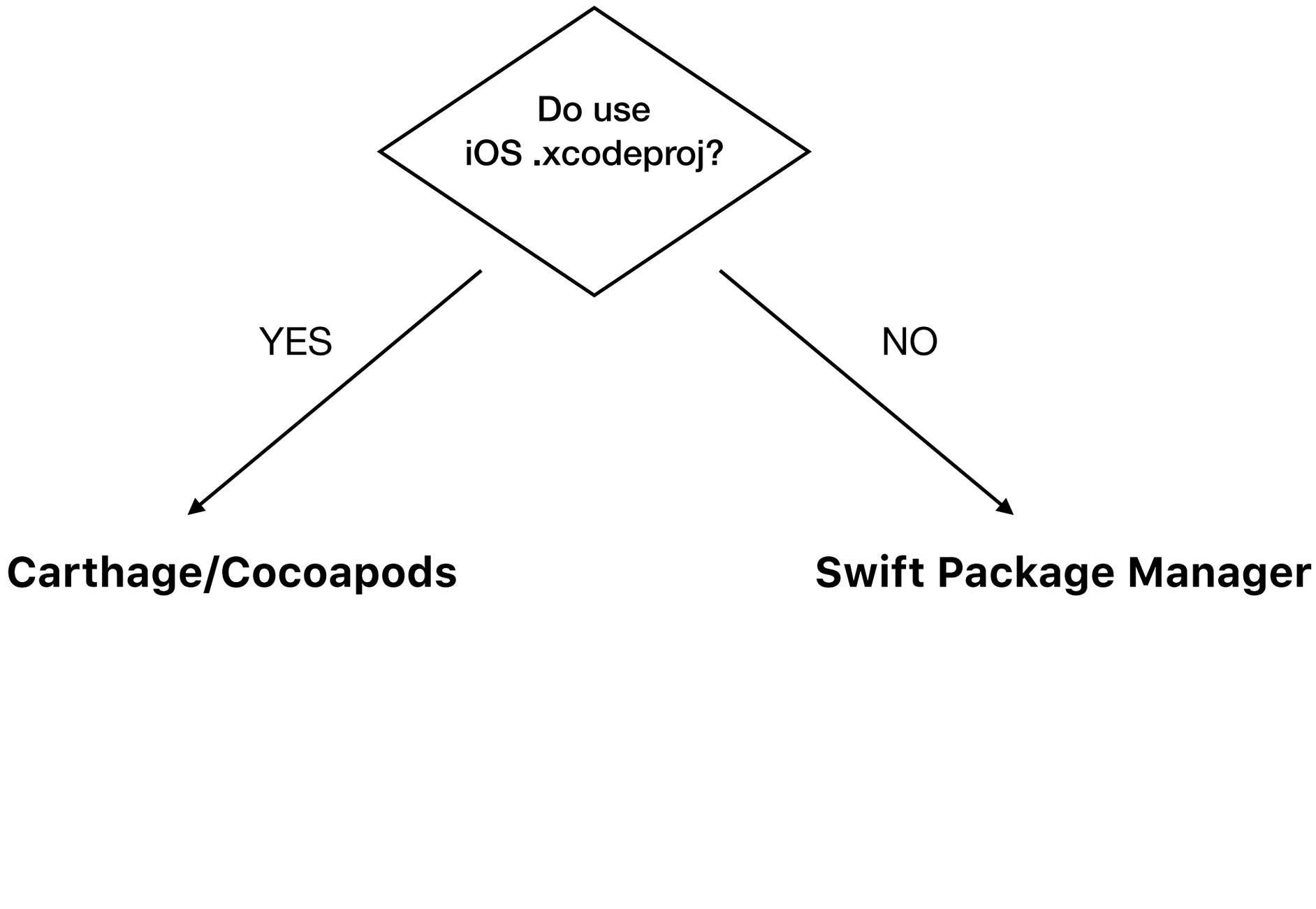
What to choose?



What to choose?

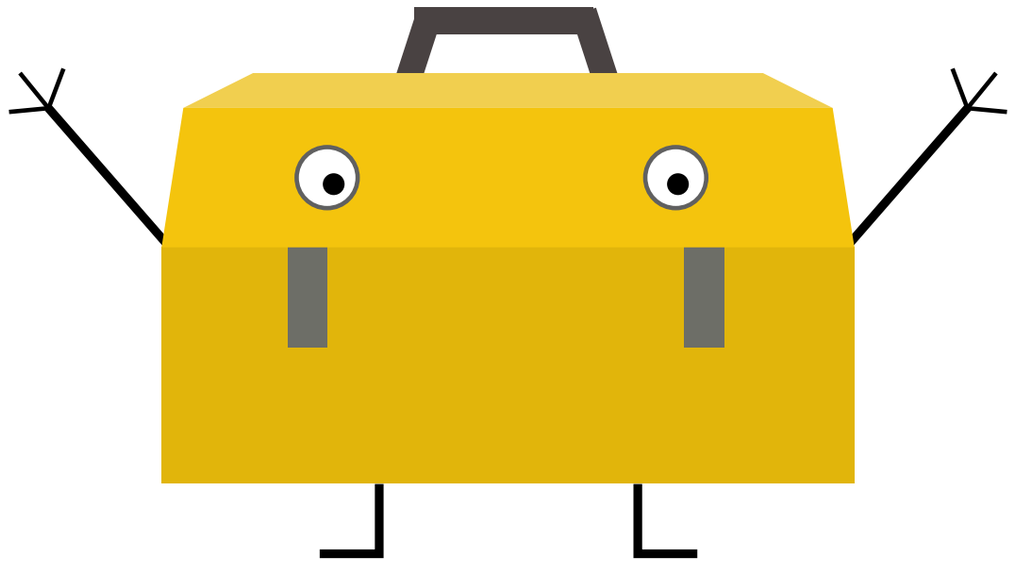


What to choose?



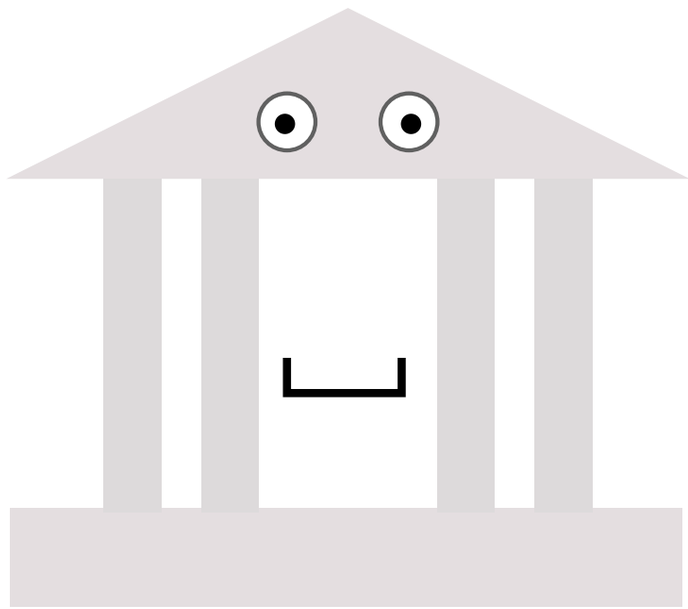
Startup time

Dynamic Framework



Slow startup time

Static Library

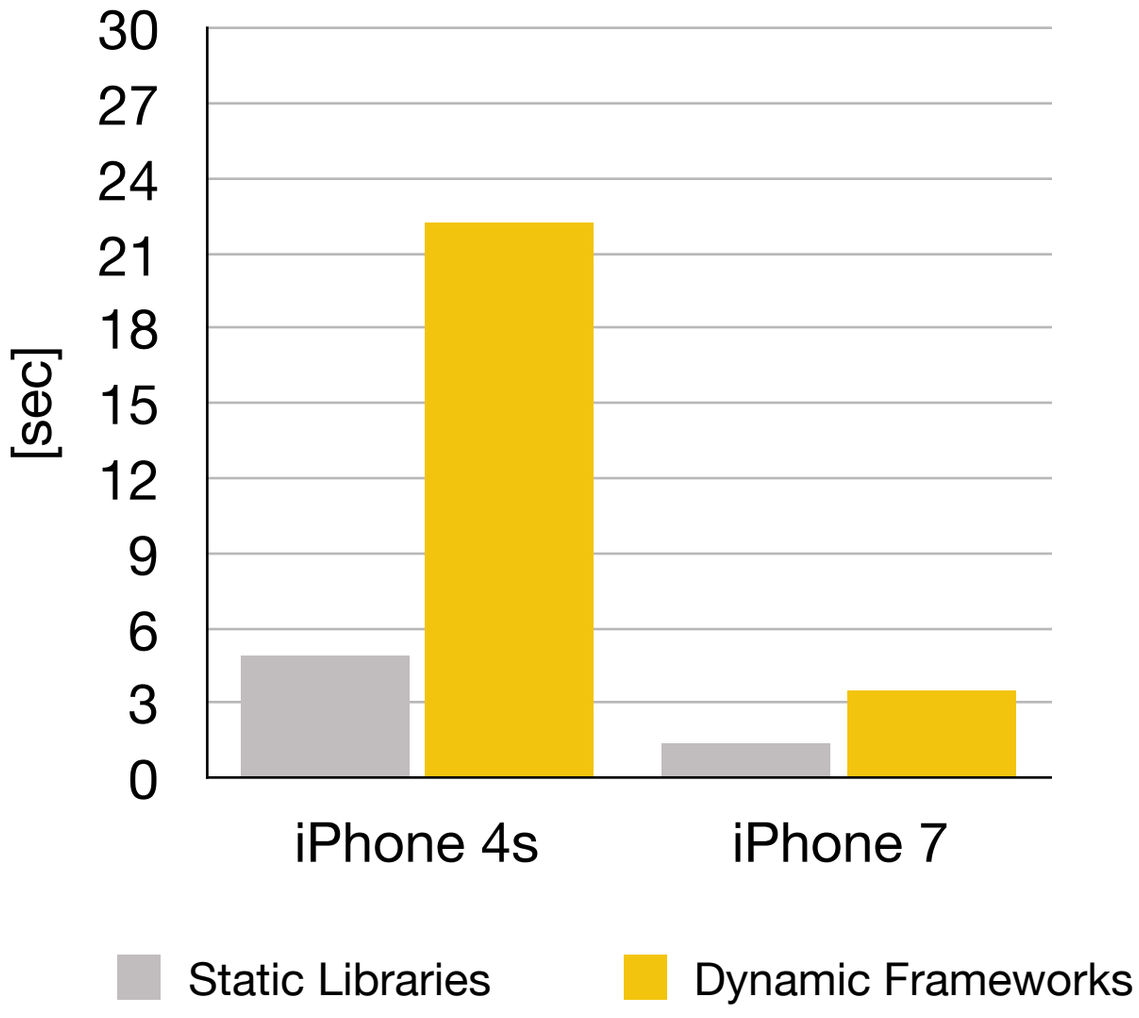


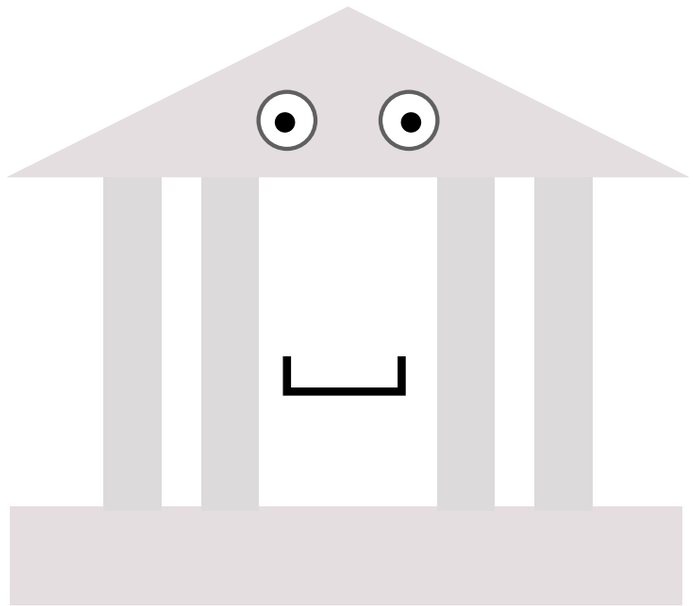
Faster startup time

Startup time test

- 100 modules
- 10 classes
- dynamic frameworks vs static libraries
- iPhone 4s, iPhone 7

Startup time





Static Libraries in Dependency Managers

Cocoapods

~~use_frameworks!~~

Carthage

- Linking → Match-O Type: Static Library
- remove Carthage Run Script
- add *./Carthage/Build/$(PLATFORM_NAME)/Static* files to Build Phases

- 3-rd party modules ???

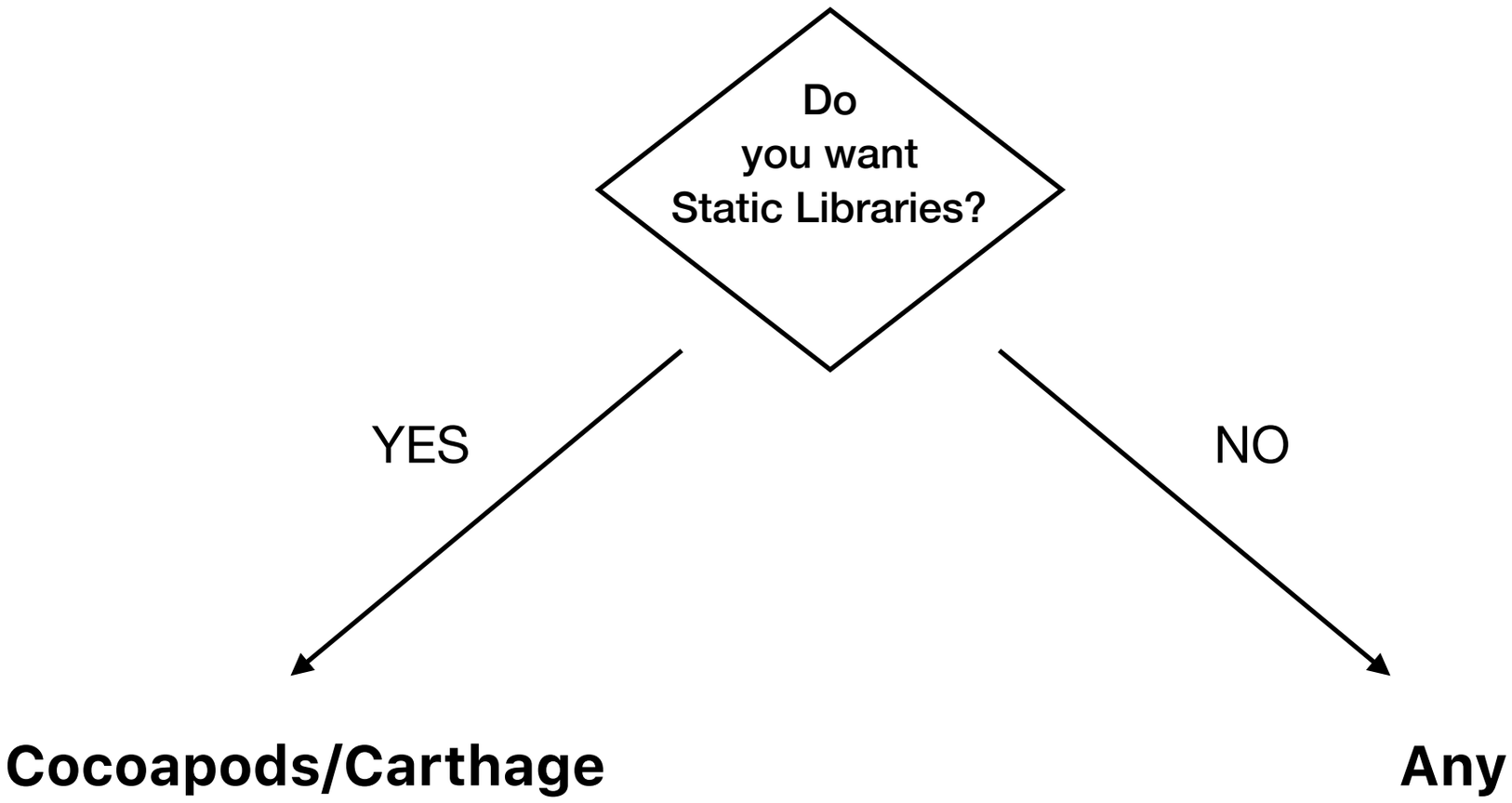
Static Libraries in Dependency Managers

Swift Package Manager

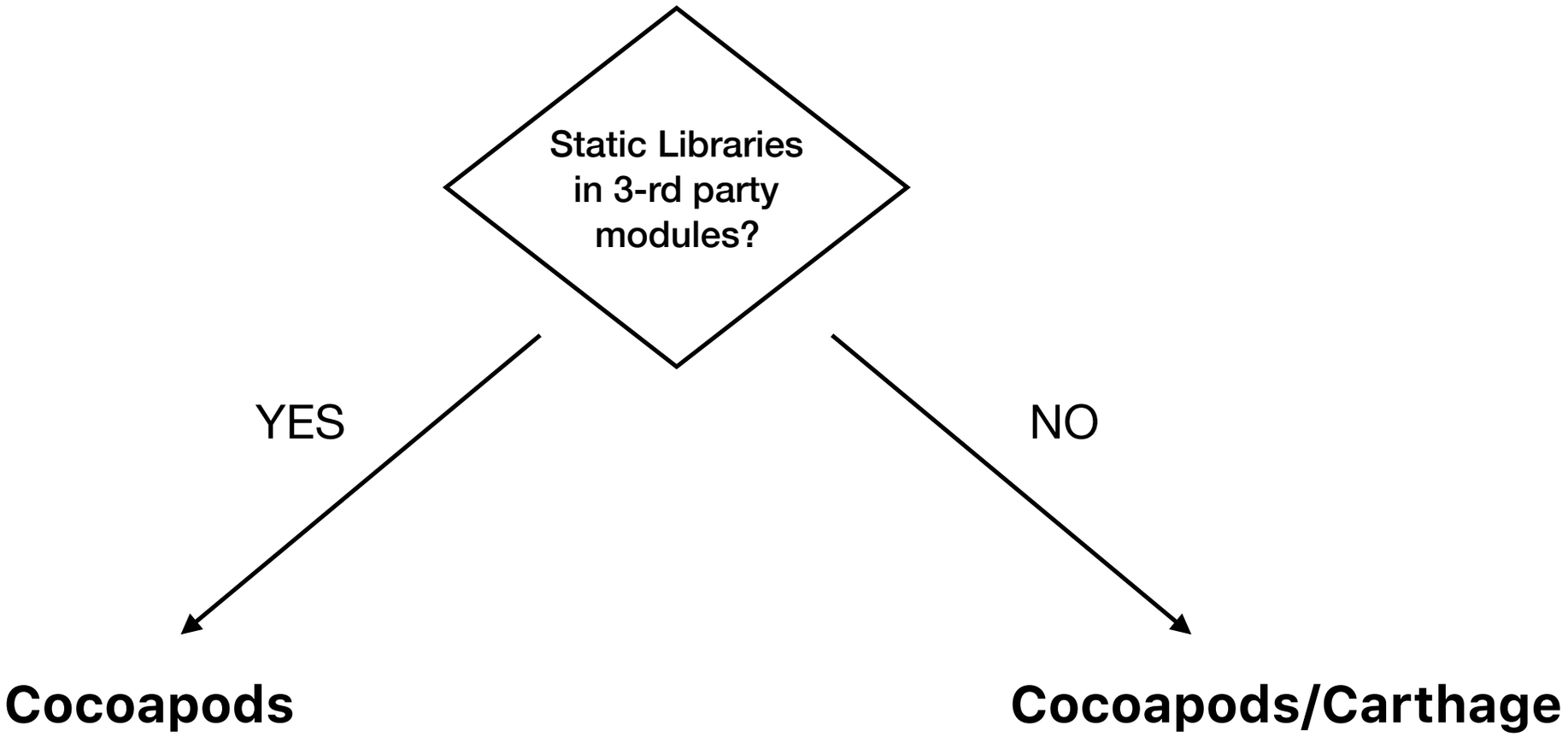
Does not support Static Libraries!!!

What to choose?

What to choose?



What to choose?



Editors pick

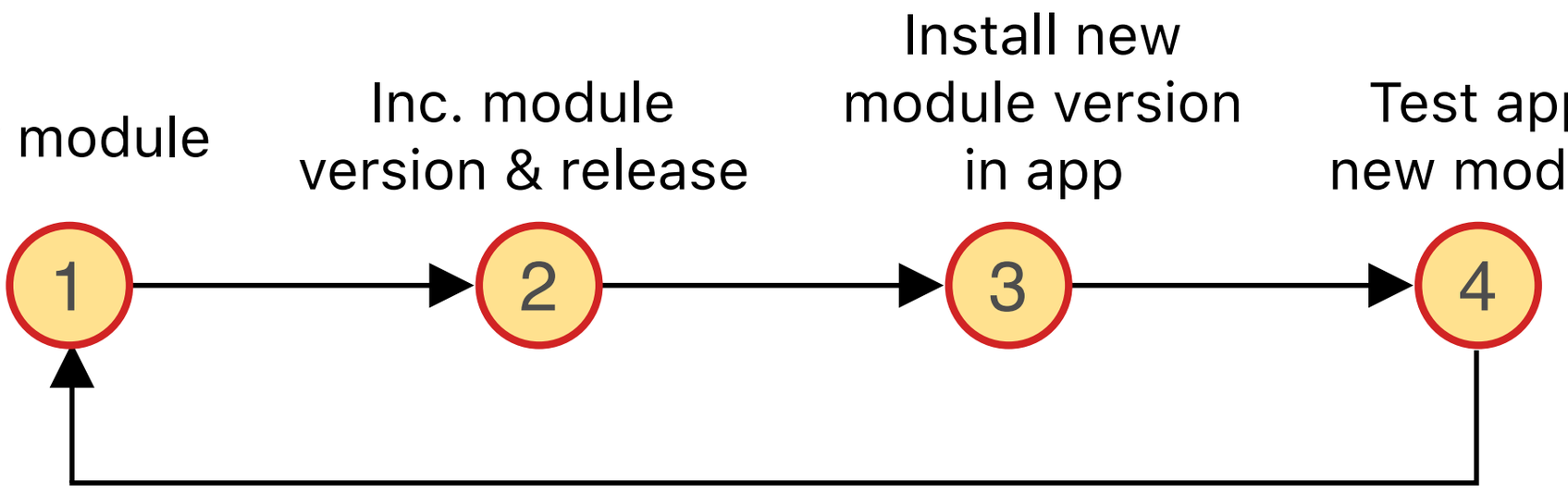
Our needs

- many apps
- many developers
- short startup time
- same dependency manager for internal and 3-rd party modules

Our choice

- ✅ Manyrepo
- ✅ Static Libraries
- ✅ CocoaPods

Manyrepo development workflow



Test fail – repeat

No tools

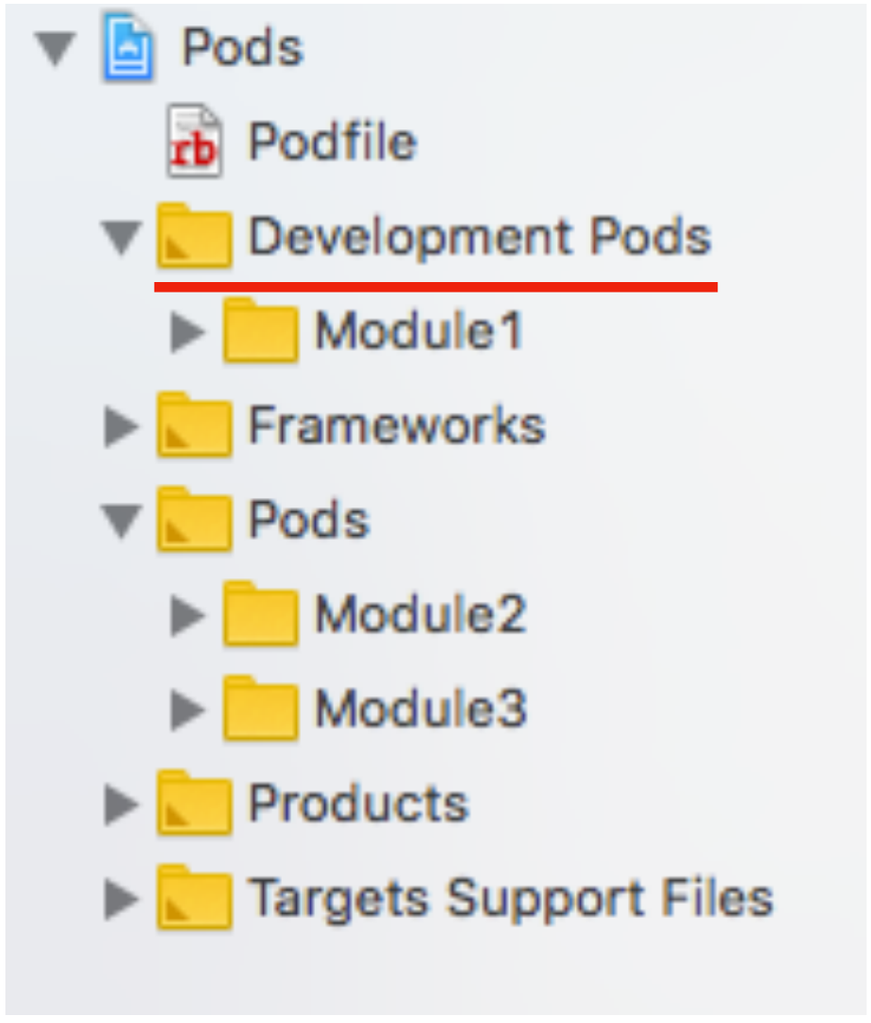
Podfile

```
pod 'Module1', '1.0.1'
```

Module development

```
pod 'Module1', :path => '../Module1'
```

Development Pods

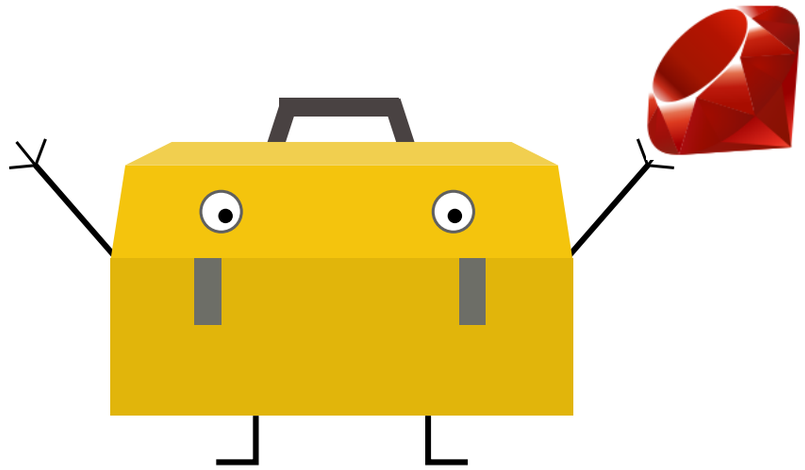


WIP commit

```
pod 'Module1',
    :git => 'https://.../Module1.git',
    :commit => '0f506b1c45'
```

Manual work

Ruby

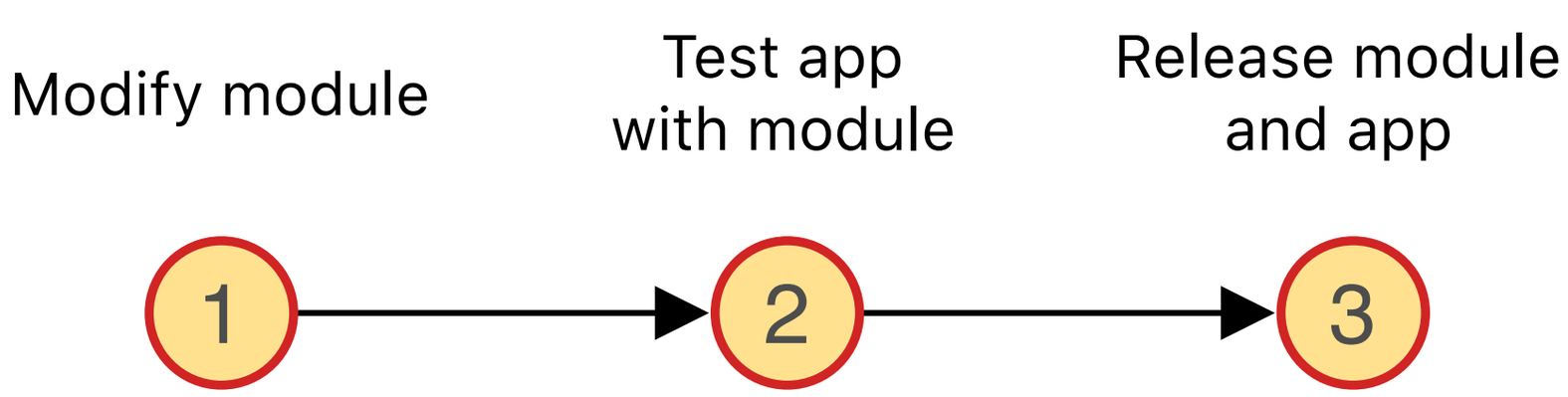


```
Podfile is Ruby source file
```

Ruby scripts automation

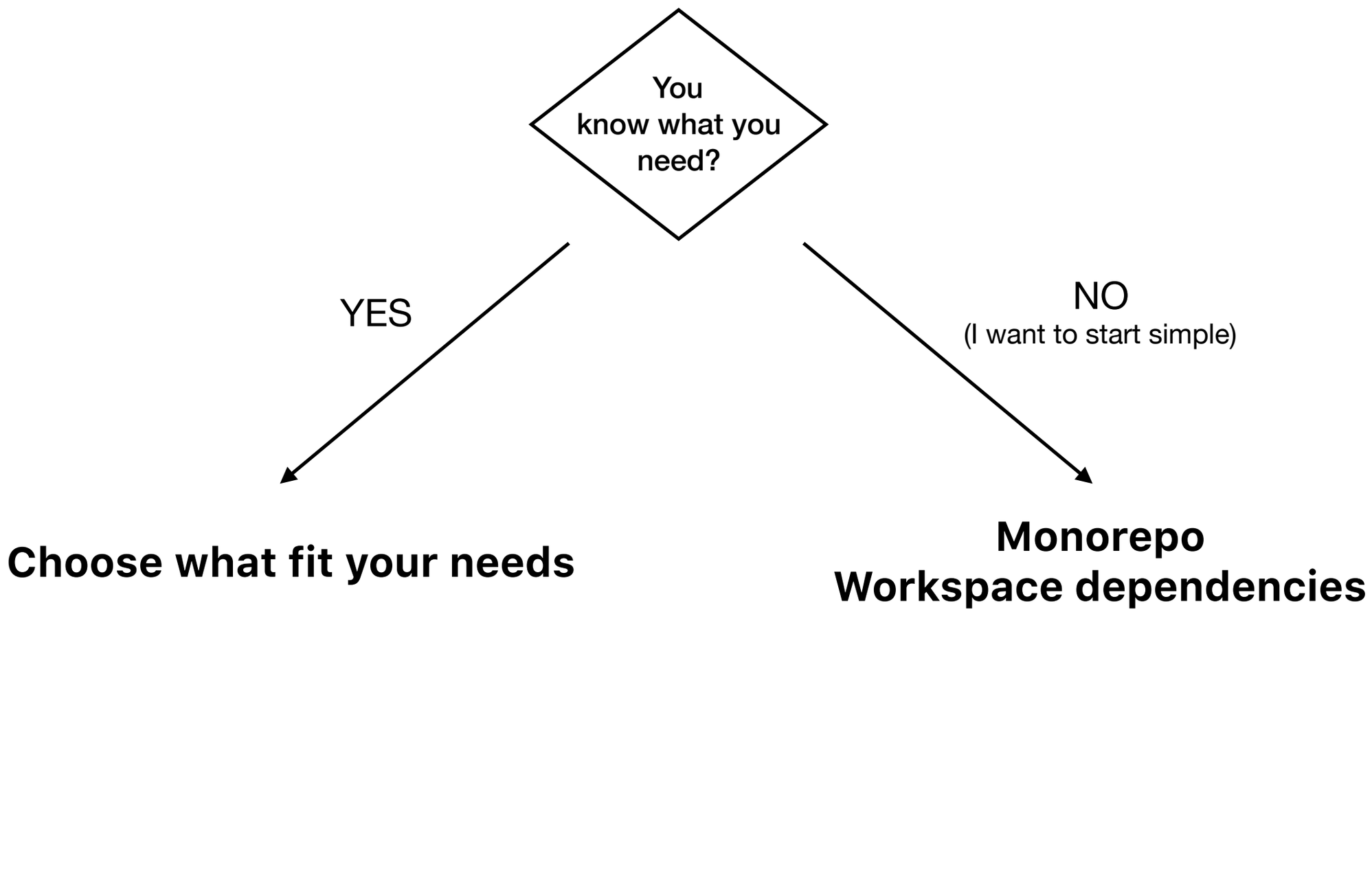
- manipulate **`Podfile`**
  - switch to **`:path`** mode
  - switch to **`:commit`** mode
- automate **`git`** commands on modules
  - auto checkout specific module

New development workflow



Sum up

Want to start Modularizing?



Q&A

Arek Macudziński

arek.macudzinski@gmail.com

iOS developer

Lessons learned

- choose tools which are suitable directly to you needs
- don't be afraid to play with this tools and extend its functionality
- it’s good to know Ruby