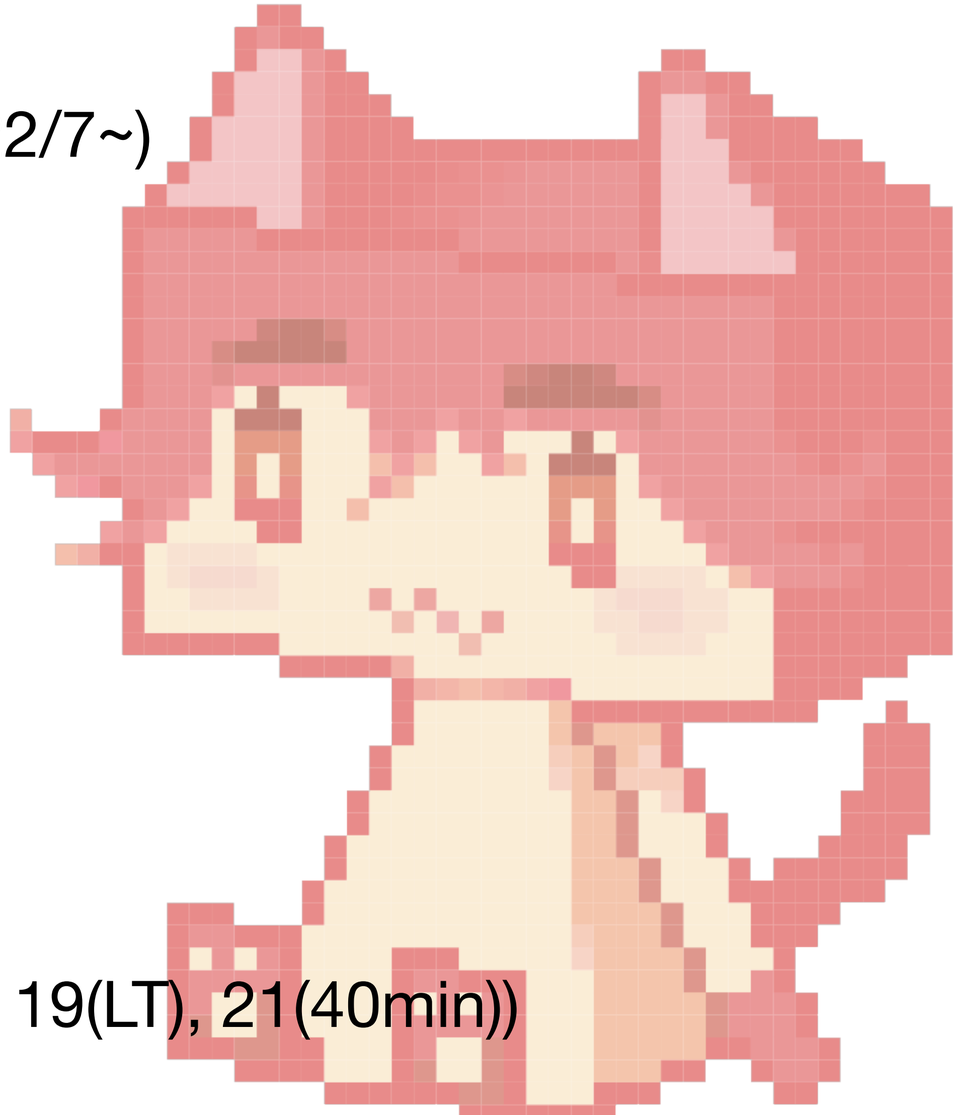
こんにちは

- @giginet
- LINE株式会社 Developer Experience Team(2022/7~)
- 技術顧問
  - 株式会社マネーフォワード
  - 株式会社ユビレジ
- Core Committer
  - XcodeGen/fastlane/Carthage …
- iOSDC登壇2年ぶり6回目 (2017(LT), 18(30min), 19(LT), 21(40min))



Swift Packageを使った
巨大な依存グラフのキャッシュ戦略

@giginet

2023/09/01 iOSDC 2023 Day 0

今日話すこと

1. **LINEアプリのビルド環境の紹介**
   - 入社時(2022/7)に直面した数々の問題点 🔥
   - なぜパッケージのキャッシュが必要になったのか
2. **Swift Packageをキャッシュ可能にする手法の紹介**
   - 新しいビルドツールScipioの紹介
   - Swift Packageを再配布可能に扱うために
3. **LINEアプリのビルド環境をどう改善したか**
   - 依存関係のビルドが要らない世界に

学べること

- 大規模プロジェクトのビルド環境のケーススタディ
- ビルドツールの内部動作
  - Swift Package Managerの各種機能やビルド設定
  - XCFrameworkの仕組みや構造
  - リモートキャッシュツールの構築アイディア

第1章

LINE iOSアプリビルド環境の課題


9:41
LINE

iOSDC LINEオープンチャットでわいわいしよう 🎉

匿名で参加できます（名前を設定できます）


プロジェクトがデカい

LINEプロジェクトの開発規模(2023/8)

コード行数	200万行以上
Xcodeプロジェクト数	300以上
ビルド済みフレームワーク数	150以上

LINE iOSのビルドプロセス(~2022)

Bazel

Prebuiltタスク

STEP 1

Xcode

アプリケーションビルド

STEP 2

LINE iOSのビルドプロセス(~2022)

Bazel

Prebuiltタスク

STEP 1

LINE

Xcode

アプリケーションビルド

STEP 2

LINE iOSのビルドプロセス(~2022)

Bazel

Prebuiltタスク

STEP 1

LINE

Xcode

アプリケーションビルド

STEP 2

Prebuiltタスク

- OSSライブラリのFramework化
- 社内実装のFramework化
- バイナリフレームワークの取得
- APIスキームからのコード生成

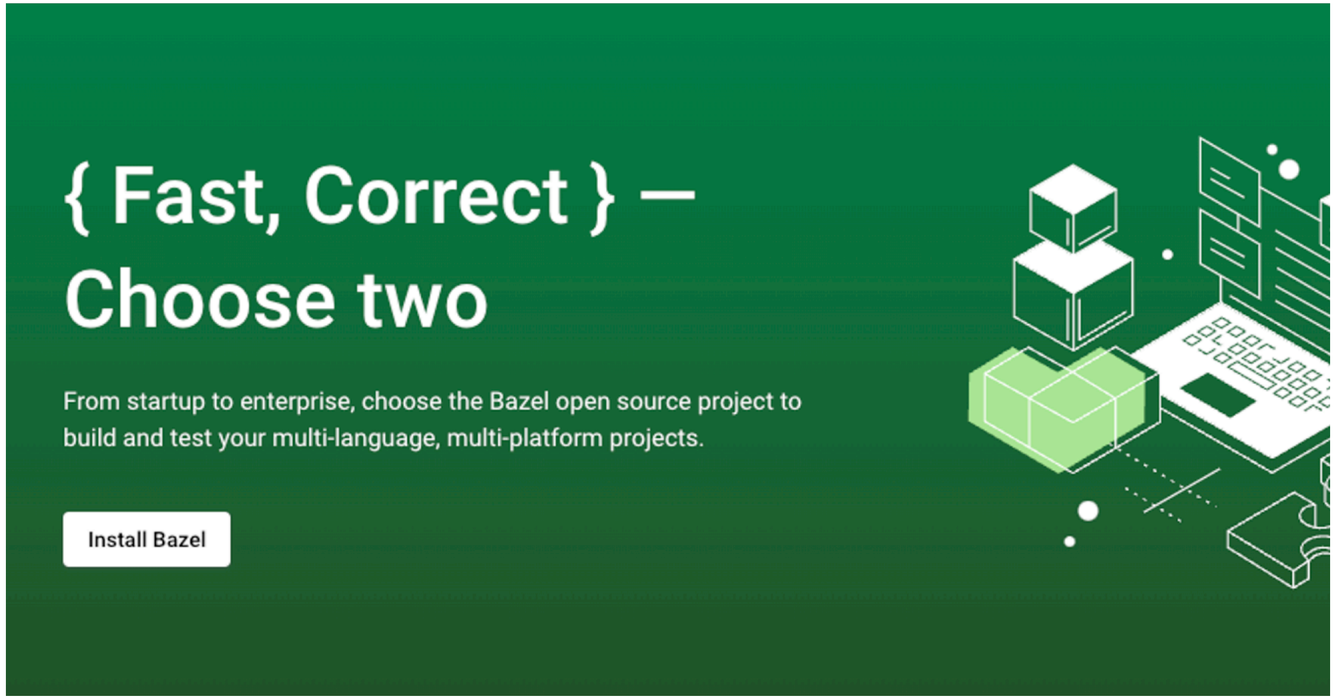
Bazel

Prebuiltタスク

STEP 1

Bazelとは

- Googleの開発したビルドツール
- ビルドの再現性を維持したまま、高速なビルドパイプラインを構築できる
  - https://bazel.build/?hl=ja



Prebuiltタスク

- OSSライブラリのFramework化
- 社内実装のFramework化
- バイナリフレームワークの取得
- APIスキームからのコード生成

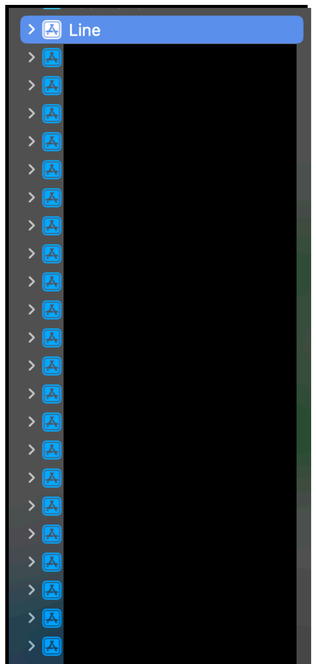
Bazel

Prebuiltタスク

STEP 1

アプリケーションビルド

- XCWorkspace + 複数(100~)のxcodeprojで構成
  - XcodeGenで各プロジェクト生成
- Prebuiltタスクで作成したフレームワークを統合



Xcode

アプリケーションビルド

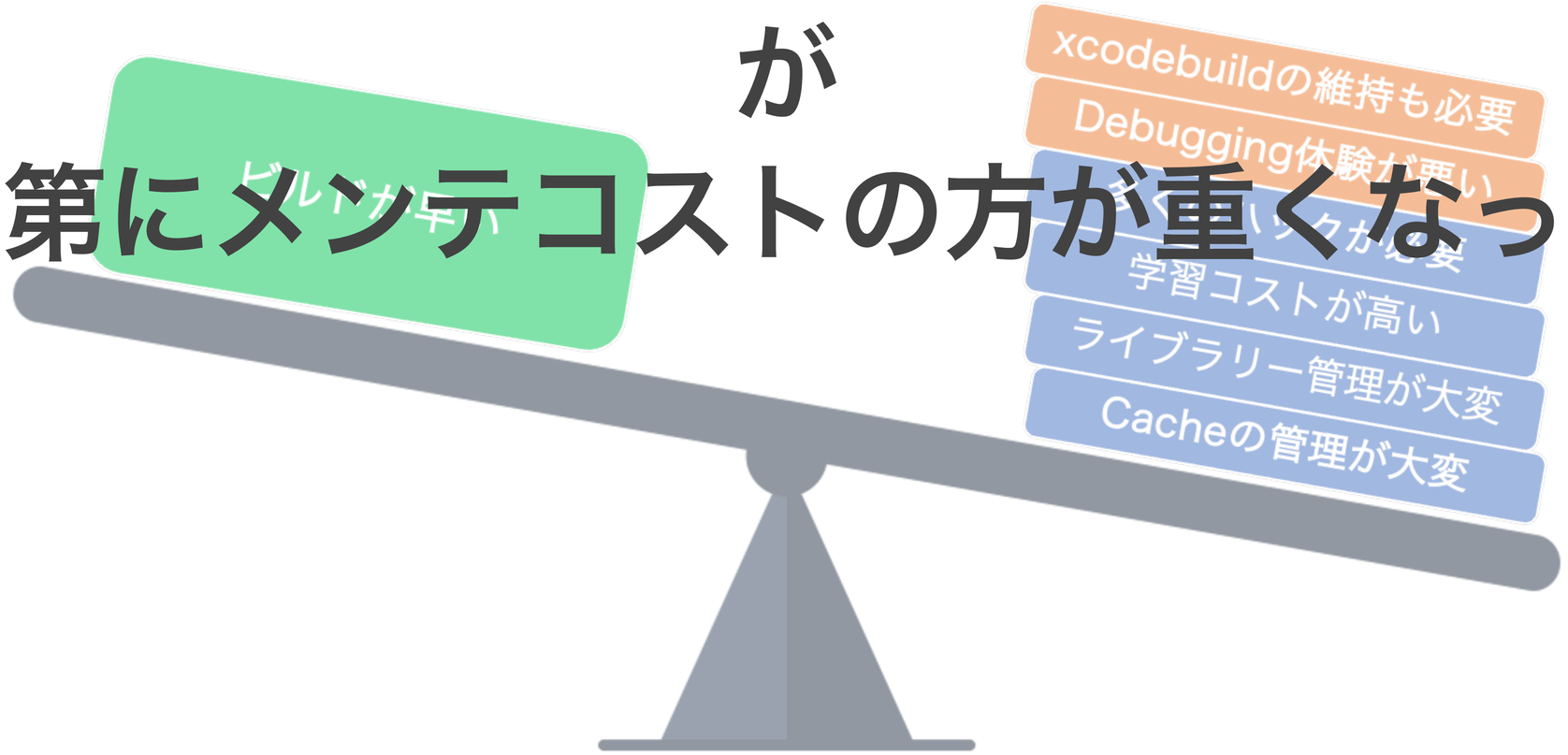
STEP 2

Bazelの導入とビルド速度

- 膨大なコードベースによりビルド速度が問題に
- 2019年頃からのBazelの導入により、一時はビルド速度が速くなった 🎉

Bazelの導入とビルド速度

- 膨大なコードベースによりビルド速度が問題に
- 2019年頃からのBazelの導入により、一時はビルド速度が速くなった 🎉



LINE iOS
ビルド環境の変遷

Giuk Jung

LINE Platform Dev Center 2

Developer Experience Team

iOSDC 2022 スポンサーセッション

LINE iOS ビルド環境の変遷

https://speakerdeck.com/line_developers/changes-in-line-ios-build-environment

LINEはなぜBazelを使わないことにしたのか？

https://engineering.linecorp.com/ja/blog/line-bazel

課題 -Bazelのメンテナビリティ-

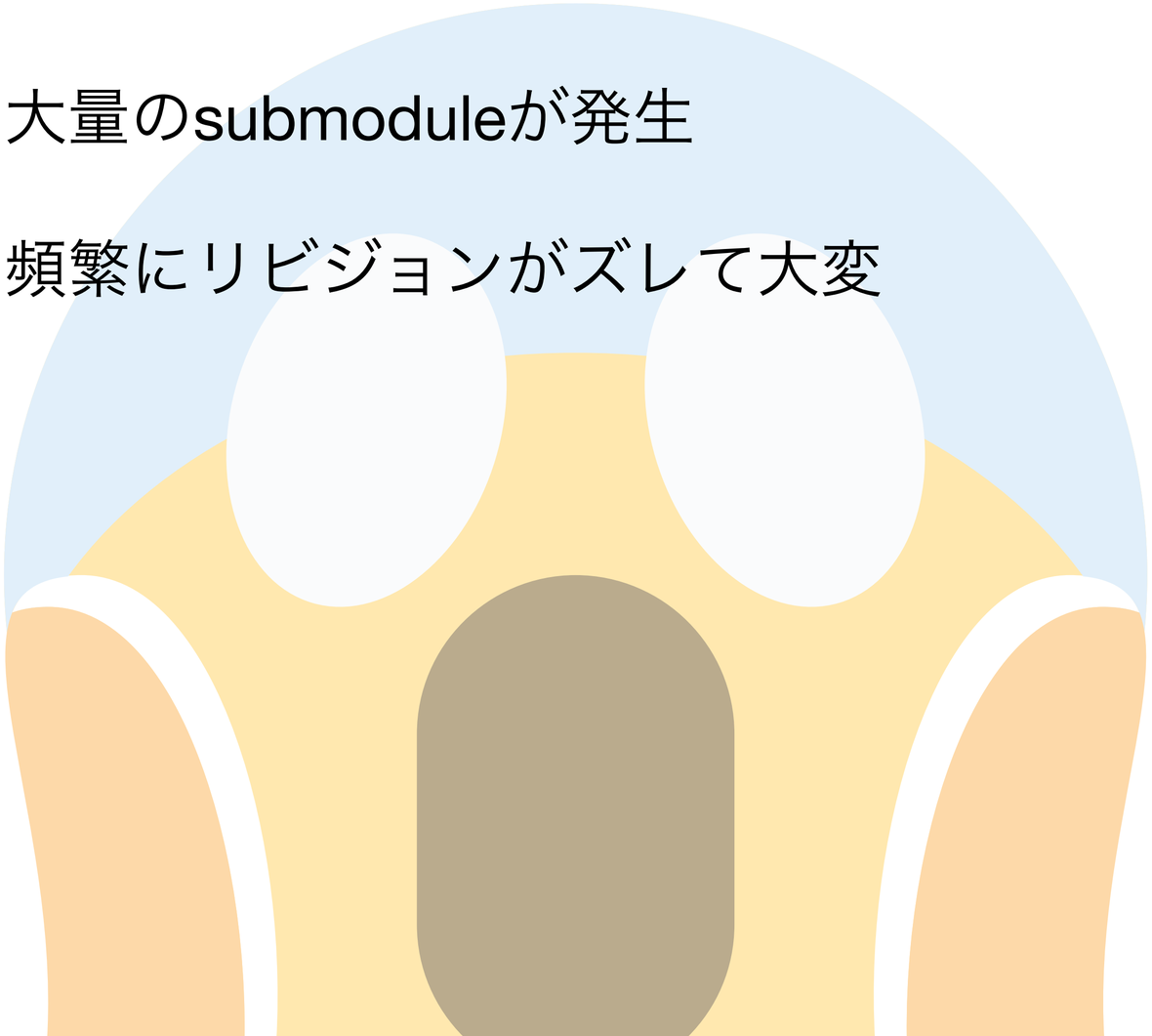
- Bazelでのビルドパイプライン構築の学習コストが高い
  - 深刻なBazel職人の後継者不足(Starlark)
  - テストがない独自Rule
- Bazelのバージョンが古いまま数年放置
  - 最近ようやっと上げた
  - forkまでしてたので最近消した

apple_static_library...

```
38  def _apple_static_library_impl(ctx):
39      # Validation of the platform type and minimum version OS currently ha
40      # `transition_support.apple_platform_transition`, either implicitly t
41      # `dotted_version` or explicitly through `fail` on an unrecognized pl
42
43      link_result = linking_support.register_static_library_linking_action(
44
45      files_to_build = [link_result.library]
46      runfiles = ctx.runfiles(
47          files = files_to_build,
48          collect_default = True,
49          collect_data = True,
50      )
51
52      return [
53          DefaultInfo(files = depset(files_to_build), runfiles = runfiles),
54          AppleBinaryInfo(
55              binary = link_result.library,
56              infoplist = None,
57          ),
58          link_result.objc,
59          link_result.output_groups,
60      ]
61
62  apple_static_library = rule(
63      implementation = _apple_static_library_impl,
64      attrs = dicts.add(
65          rule_factory.common_tool_attributes,
```

課題 -コードベースの統合方法-

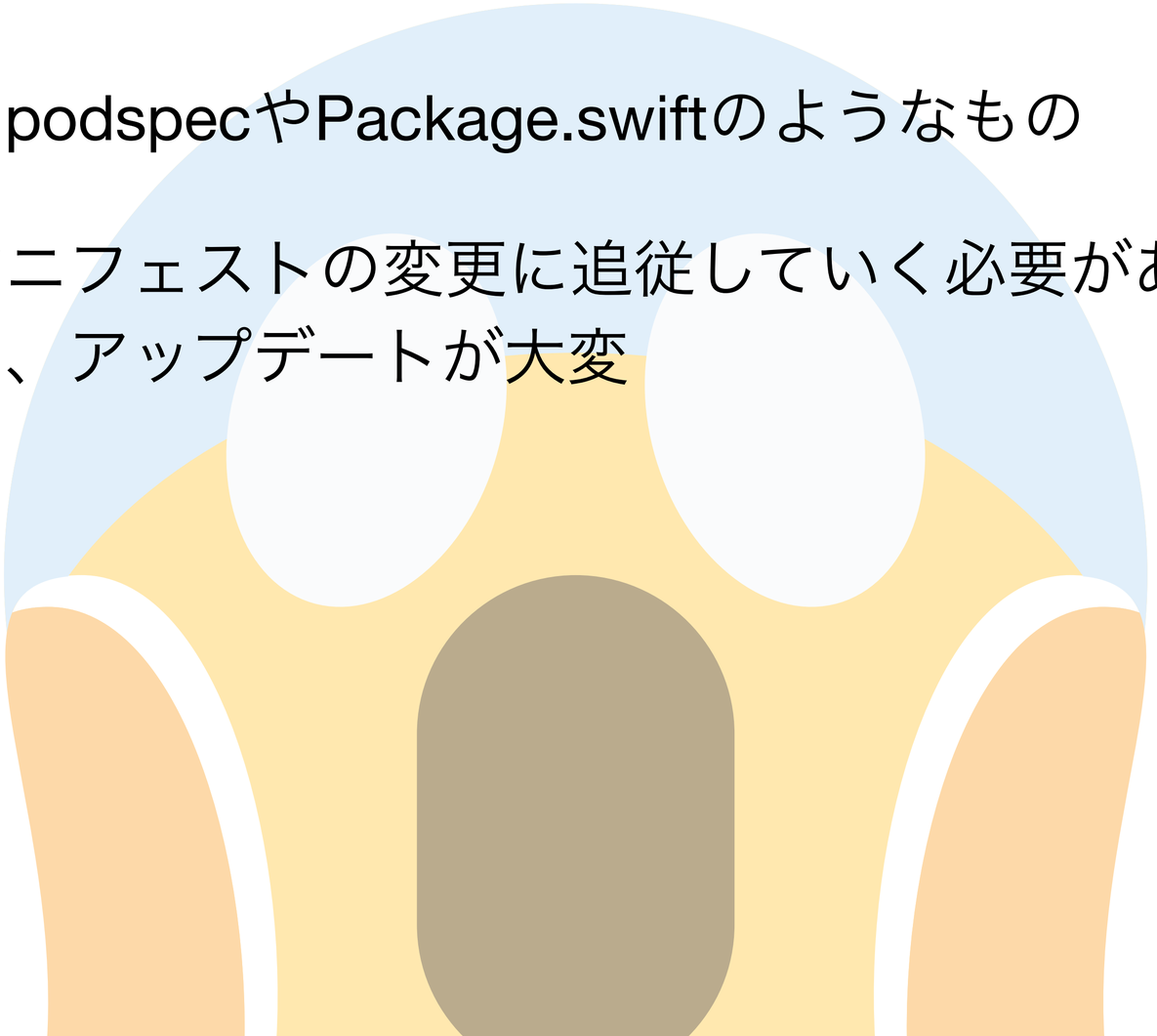
- コードの取得はBazelを使うかsubmodule
  - 大量のsubmoduleが発生
  - 頻繁にリビジョンがズレて大変



```
line-ios (master) ● git submodule
70db0ef48e25908488c311a82c9329f0c330fb99 Externals/
64b0e1b13cc80e6322f47a8effd59dd6d26310d4 Externals/
56278ea800d7fdc20f42cd77bdbf0fecc8c1c0c8 Externals/
a0bf26c8f84f6095bcbc7cdb0d01af19f70e8a2d Externals/
5badb18edf474f6fb5b7ded2d57dd32abc665b11 Externals/
ccebdf4da57026e520238163f328d66bbfe5f446 Externals/
49b031309c60ece08dd2ca52829a6a4d3062639e Externals/
8b746fc57ff05e1874e50b888b911589775000c9 Externals/
d872e5813231f97b1f6d5d8dd95bf399a5ce6572 Externals/
3d7d9ddcfc536f2f74f8cee5daaa7a272d801ef4 Externals/
1daf2cc1322d0eeb1b8118c3c0b2199fb276dd78 Externals/
4a22531ee67e992894edd449d81143de28e978b0 Externals/
b89b4967692331b2d890db77fa0fd6a6d37909ff Externals/
4536525ab0a87e67e8f8813b839902eeb8e32f11 Externals/
461fbefac209657a6bebd4cf10eca1b4a784a4cb Externals/
065adfc39a56e9fbb7acca0c94a0deaffdcd6cef Externals/
aabc6392c65d4498414b2a247c7448432a666a4f Externals/
7be58363296790de6f9cb31c8c5fc06fcfc8d5d6 Externals/
96977de4bc58190a5467bca096343d44b5873518 Externals/
c126ab0a0a1d8dfec252a7ef3c423fb0b3805d1b Externals/
71b99fb3de17aef7a77ecb80f904bb16951b016d Externals/
f85d3f6d4ae75c780f5440dc87c7318c092d87bb Externals/
ae51c52d4fac290e23f73e6629d03d3b831e4f6f Externals/
0a99f6b9d2fbd5eddf4bef3ad6a1909545f28d24 Externals/
deab69c560b028fdc756036131ec86363fc6f1c8 Externals/
c2bc48bdc16aaa26a49f3b3c546acf4b118c0ffd Externals/
6fe203dc33195667ce1759bf0182975e4653ba1c Externals/
374dc7f7b9e76c6aeb393f6a84590c6d387e1ecb Externals/
```

課題 -BUILDファイルの定義-

- 各依存に対してBUILDファイルの定義が必要
  - podspecやPackage.swiftのようなもの
- マニフェストの変更に追従していく必要があり、アップデートが大変

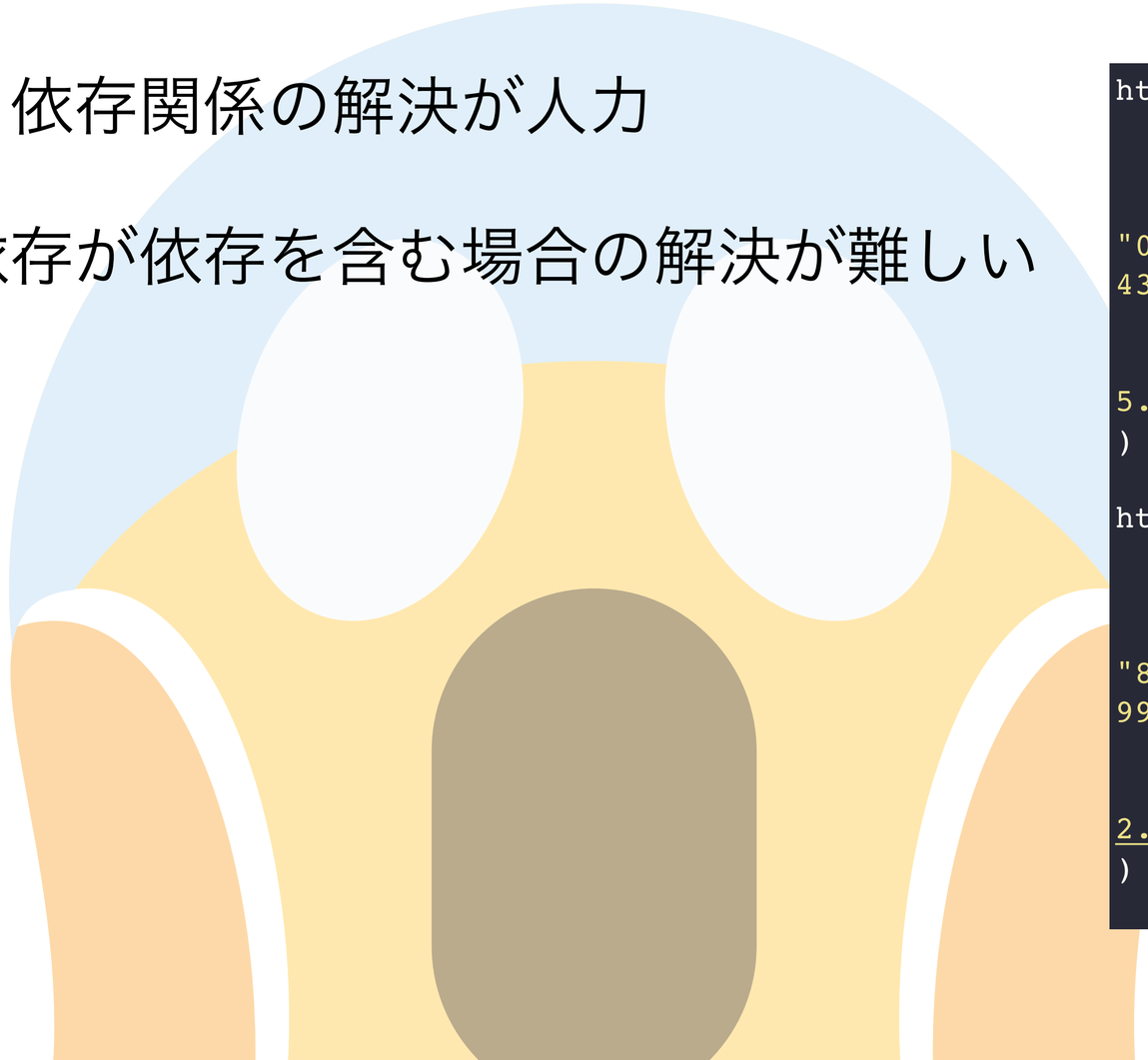


```
load(
    "//apple:ios.bzl",
    "ios_static_framework",
)

ios_static_framework(
    name = "Logging",
    srcs = [
        "Sources/Logging/**/*.swift"
    ],
)
```

課題 -バージョンリゾルバがない-

- バージョンリゾルバがない
  - 依存関係の解決が人力
- 依存が依存を含む場合の解決が難しい

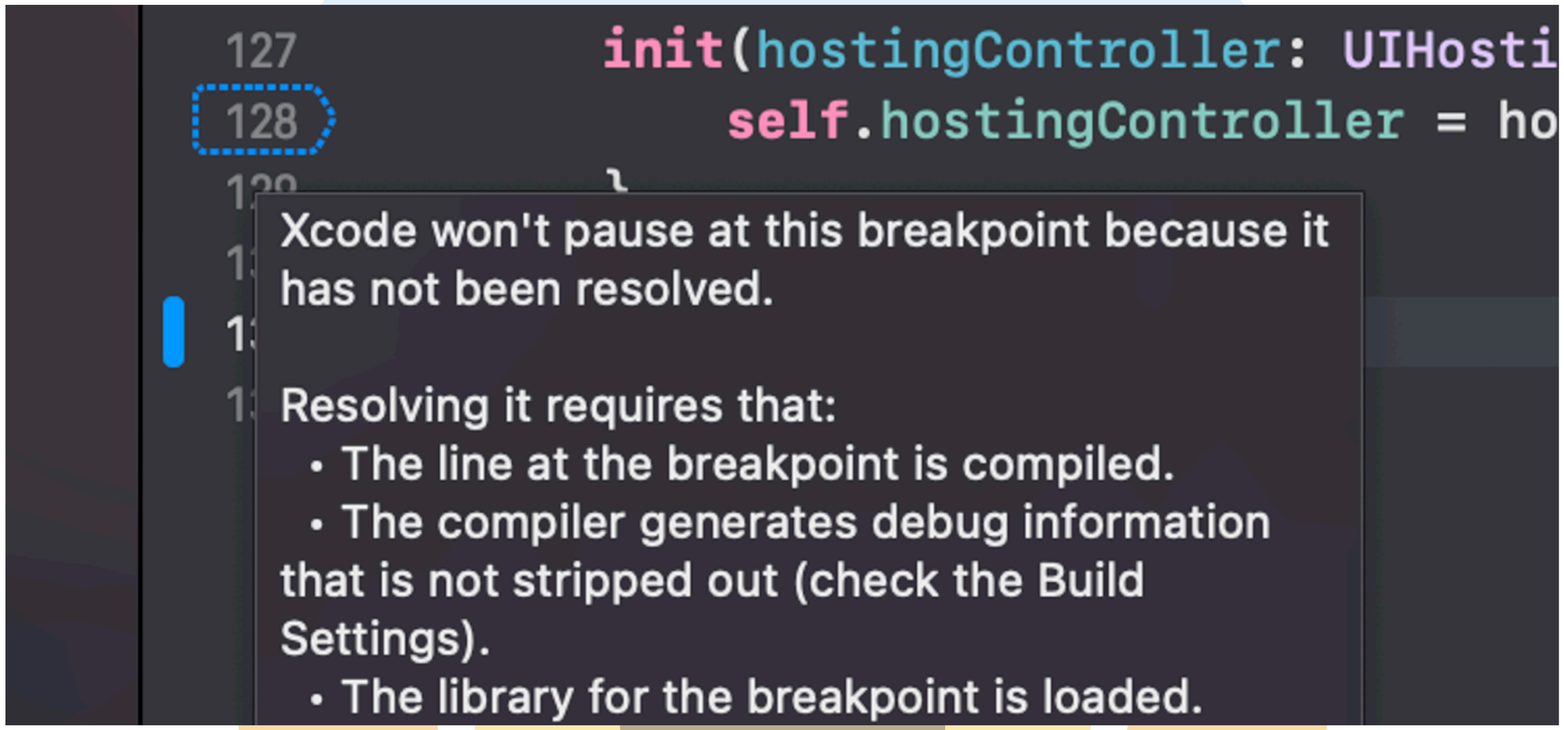


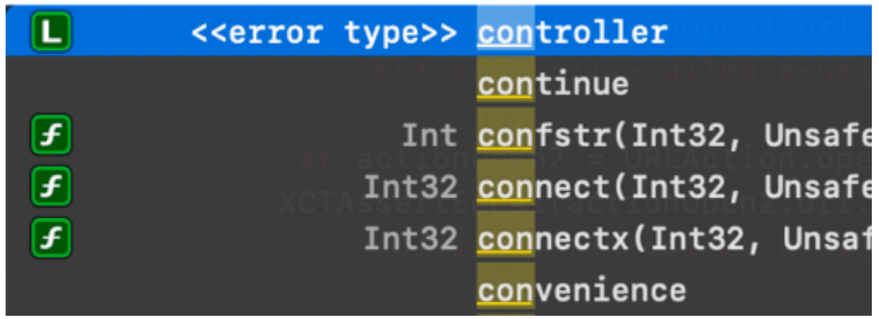
```
http_archive(
    name = "APIKit",
    build_file = "//Configuration/repositories:APIKit.BUILD",
    sha256 =
"0d19ebb5769ca464c7b10671ca1df21b8e9edad198f59e2d854c48cc580383
43",
    strip_prefix = "APIKit-5.0.0",
    url = "https://github.com/ishkawa/APIKit/archive/
5.0.0.tar.gz",
)

http_archive(
    name = "APNGKit",
    build_file = "//Configuration/repositories:APNGKit.BUILD",
    sha256 =
"89245edaabcda2872141ecfcfad6d88fa5f0293af52a4cbcd2dd2fd6e3f931
99",
    strip_prefix = “APNGKit-2.2.1”,
    url = “https://github.com/onevcat/APNGKit/archive/
2.2.1.tar.gz”,
)
```

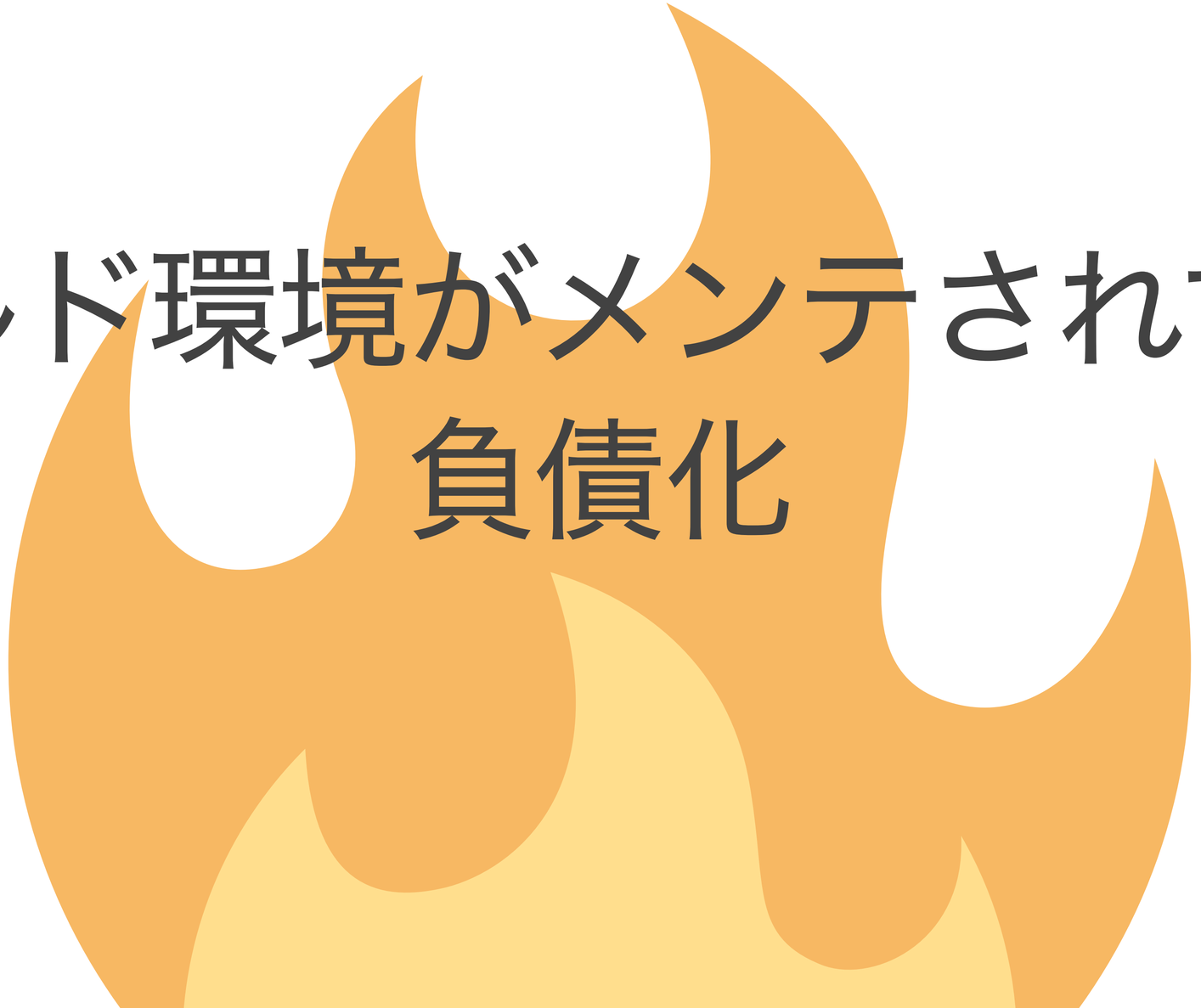
課題 -Xcodeとの連携不備-

- 標準機能の利用が難しい
  - デバッガ、補完、コードジャンプ





ビルド環境がメンテされずに
負債化



LINE iOSにおけるBazelの功罪

- 確かにビルドは早くなったが・・・・・・
- メンテナンスコストの増大
  - 学習コストの高さ、ビルド設定の複雑さ
- 標準環境と離れることのデメリット
  - 新機能追従に時間がかかる(Xcode Beta, Apple Silicon etc…)
- Apple Siliconの登場でメリットが限定的に
- Bazelは劇薬だった

おことわり

- Bazelの利用によりビルド時間に効果があったのは事実
- Bazelコミュニティによる iOSサポートは積極的に続いている
  - rules_apple, rules_ios, rules_swift_package_manager, rules_xcodeprojなど
- 大規模プロジェクトによる使用実績も多い
- Bazelを継続的にメンテできるのであれば有用
- LINEでは適切に運用できなかった

新たな要求

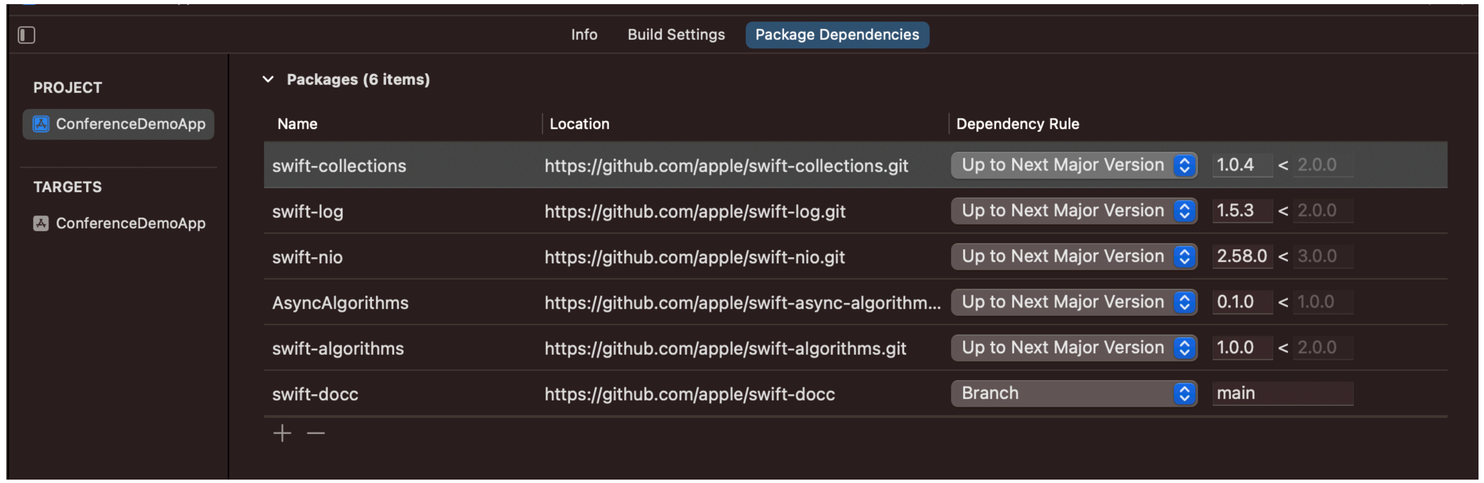
- とにかく標準のビルドシステムを使いたい
- ビルド成果物が再利用できてキャッシュしやすい
- 開発しやすくテストしやすい
- 依存関係のアップデートがしやすい

we created Swift Packages which

WWDC19 Platform State of the Union https://developer.apple.com/videos/play/wwdc2019/103/?time=1302

Swift Package + Xcode

- Xcode標準のSwift Package Managerは**最高の解決策**
- **これが使えるならばこれを使うべき**

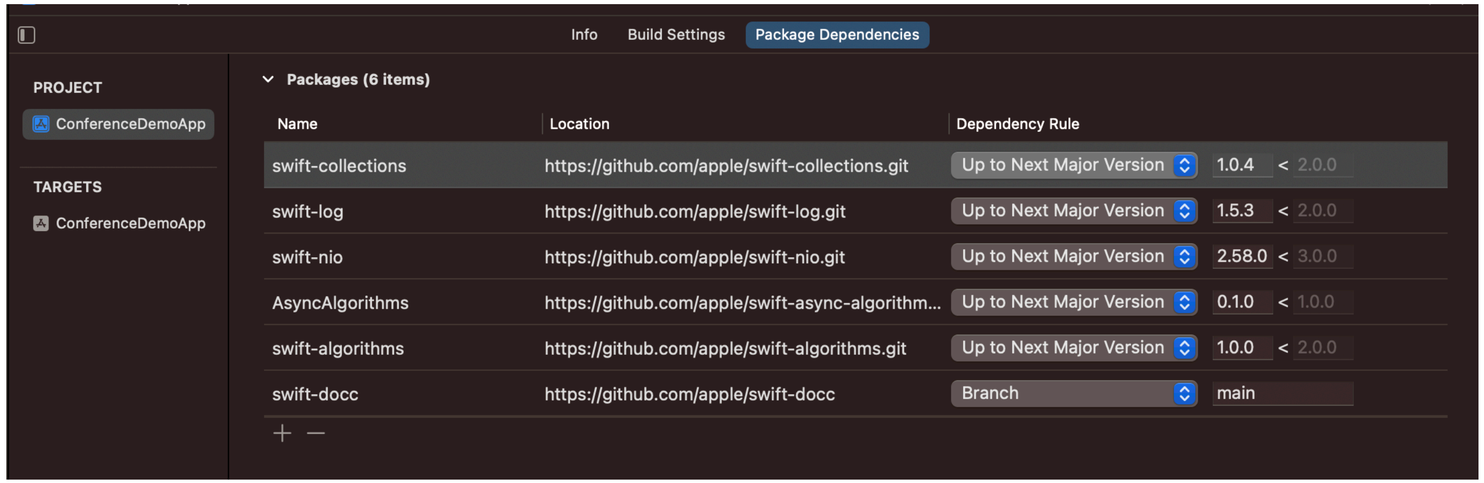


完

ご清聴ありがとうございました

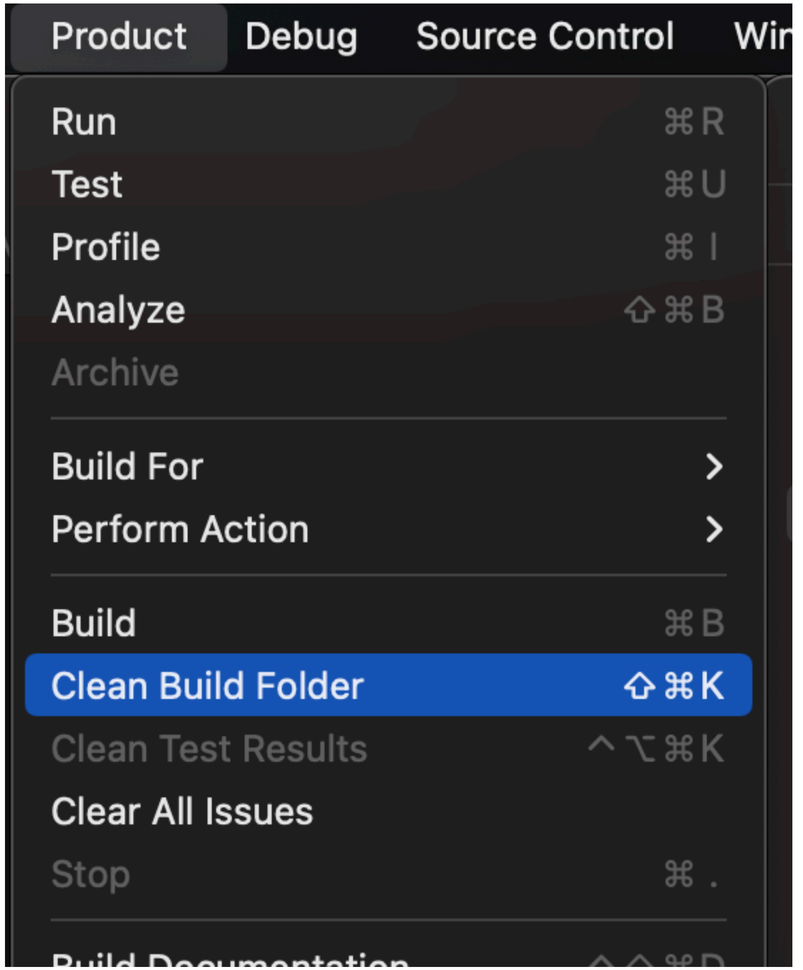
Swift Package + Xcode

- Xcode標準のSwift Package Managerは**最高の解決策**
- これが使えるならばこれを使うべき
- **しかし、LINEアプリの規模 + プロジェクト構成では標準機能だけでは難しい**



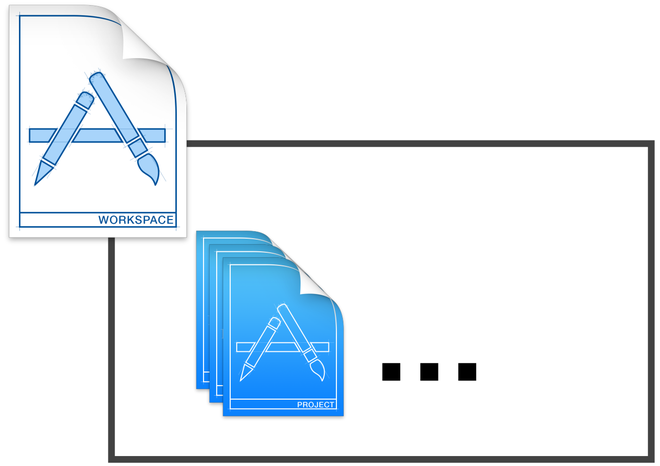
キャッシュ機構の不足

- Swift Packageの成果物が揮発しやすい
  - DerivedDataに保存
- **→100を越える依存関係のキャッシュには不適**



複数のxcodeprojからの利用の難しさ

- LINE iOSプロジェクトはxcworkspaceに数百のxcodeprojを統合している
- このプロジェクト構成でXcode標準のSwift Package統合を使うのは難しい



理想

1. SwiftPMによるSwift Packageの依存解決と取得

2. ビルド成果物をキャッシュ、共有、再利用

3. Xcodeプロジェクトからの利用

Scipio

- An awesome build tool -

https://github.com/giginet/Scipio

Scipio Public

Pin | Unwatch 11 | Fork 11 | Star 299

main | 11 branches | 18 tags | Go to file | Add file | Code

giginet Merge pull request #96 from giginet/documentation ... 05e8238 4 days ago 620 commits

.github/workflows	Remove swift-docc for distribution	5 days ago
Plugins/GenerateScipioVersion	Rename plugin name	3 months ago
Sources	Update Sources/scipio/scipio.docc/prepare-cache-for-applications....	4 days ago
Tests	Merge pull request #94 from giginet/...	2 weeks ago
.gitignore	Add links to DocC	last week
.swiftlint.yml	disable lint rule	3 weeks ago
LICENSE.md	Create LICENSE	last year
Package.resolved	Use 14.3.1	last month
Package.swift	Remove swift-docc for distribution	5 days ago
README.md	Tweak badges	last week

README.md

Scipio

build passing | Swift 5.8 | SwiftPM compatible | Documentation available | Platform iOS|macOS|watchOS|tvOS|visionOS | Xcode 14.3 | Xcode 15.0 | License MIT

Carthage depends on

About

A new build tool to generate XCFramework

giginet.github.io/Scipio/documentatio...

Readme
MIT license
Activity
299 stars
11 watching
11 forks

Releases 14

0.16.0 Latest
2 weeks ago

+ 13 releases

Packages

No packages published
Publish your first package

Contributors 7

第2章

新しいビルドツール Scipioの紹介

Scipio

- SwiftPMを利用してXCFrameworkを生成するビルドツール
- Swift Packageを再配布可能な形に変換して扱える
- 主な機能
  - リソースバンドルのサポート
  - Static/Dynamic Frameworkの生成
  - リモートキャッシュシステム

XCFrameworkとは

- Appleが開発したビルド済みライブラリを配布するための形式
- WWDC19で発表
- 従来の🧰Frameworkに比べて複数のプラットフォーム向けのバイナリを同時に配布できる
- 実態はただのディレクトリ（Bundle）
  - 複数のプラットフォームのFrameworkを格納している

MyFramework.xcframework

ディレクトリ

Info.plist

メタデータ

ios-arm64

プラットフォーム、アーキテクチャ

MyFramework.framework

Framework

ios-arm64_x86_64-simulator

MyFramework.framework

Scipioの動作の流れ

- SwiftPMでパッケージマニフェストから依存関係を解決
- パッケージからXCFrameworkを生成

例えばCarthage

- Carthage/Carthage
  - 老舗のパッケージマネージャー
  - XCFrameworkサポートはあるが
    - ライブラリ作者がxcodeprojを設置する必要がある
    - 最近は対応しないライブラリがほとんど
  - メンテが止まっている

0.39.0 Latest

Compare

giginet released this Jan 28 · 0.39.0 · 187a78c

Fixed

- In Xcode 14, Carthage does not create tvOS & watchOS builds when Bitcode is disabled · Issue #3292 · Carthage/Carthage

What's Changed

- Add: --use-xcframeworks check by **@ryu1sazae** in #3204
- Fix typo in fileReadCorruptFile error description by **@TTOzzi** in #3231
- README.md: HTTP => HTTPS by **@Schweinepriester** in #3244
- Update Xcode12Workaround.md by **@Huang-Libo** in #3248
- Use modern Alamofire version in README by **@jshier** in #3200
- Fix issue where Carthage doesn't build for watchOS or tvOS if bitcode is disabled in Xcode 14 by **@daltonclaybrook** in #3293
- Bump up version to 0.39.0 by **@giginet** in #3318

New Contributors

- **@ryu1sazae** made their first contribution in #3204
- **@TTOzzi** made their first contribution in #3231
- **@Schweinepriester** made their first contribution in #3244
- **@Huang-Libo** made their first contribution in #3248
- **@jshier** made their first contribution in #3200

Scipio

- An awesome build tool -

https://github.com/giginet/Scipio

Scipio Public

Pin | Unwatch 11 | Fork 11 | Star 299

main | 11 branches | 18 tags | Go to file | Add file | Code

giginet Merge pull request #96 from giginet/documentation ... 05e8238 4 days ago 620 commits

.github/workflows	Remove swift-docc for distribution	5 days ago
Plugins/GenerateScipioVersion	Rename plugin name	3 months ago
Sources	Update Sources/scipio/scipio.docc/prepare-cache-for-applications....	4 days ago
Tests	Merge pull request #94 from giginet/...	2 weeks ago
.gitignore	Add links to DocC	last week
.swiftlint.yml	disable lint rule	3 weeks ago
LICENSE.md	Create LICENSE	last year
Package.resolved	Use 14.3.1	last month
Package.swift	Remove swift-docc for distribution	5 days ago
README.md	Tweak badges	last week

README.md

Scipio

build passing | Swift 5.8 | SwiftPM compatible | Documentation available | Platform iOS|macOS|watchOS|tvOS|visionOS | Xcode 14.3 | Xcode 15.0 | License MIT

Carthage depends on

About

A new build tool to generate XCFramework

giginet.github.io/Scipio/documentatio...

Readme
MIT license
Activity
299 stars
11 watching
11 forks

Releases 14

0.16.0 Latest
2 weeks ago

+ 13 releases

Packages

No packages published
Publish your first package

Contributors 7

第2章で話すこと

- Scipioの紹介と仕組み
  - 利用方法
  - キャッシュシステム
- フレームワーク生成の方法
  - 生成手法の紹介
  - フレームワークの構造
  - 生成時の問題点と対策

Scipioの動作モード

- prepareモード
  - アプリケーションに使う依存関係を記述してまとめてビルド
- createモード
  - 単体のSwift PackageからXCFrameworkを生成

1. パッケージマニフェストの定義

```
// swift-tools-version: 5.7
import PackageDescription

let package = Package(
    name: "Dependencies",
    platforms: [.iOS(.v13),],
    dependencies: [
        .package(url: "https://github.com/apple/swift-log.git",
                 from: "1.4.4"),
        .package(url: "https://github.com/apple/swift-collections.git",
                 .upToNextMinor(from: "1.0.0")),
    ],
    targets: [
        .target(
            name: "Dependencies",
            dependencies: [
                .product(name: "Logging", package: "swift-log"),
                .product(name: "OrderedCollections", package: "swift-collections"),
                .product(name: "DequeModule", package: "swift-collections"),
                .product(name: "Collections", package: "swift-collections"),
            ]),
    ]
)
```

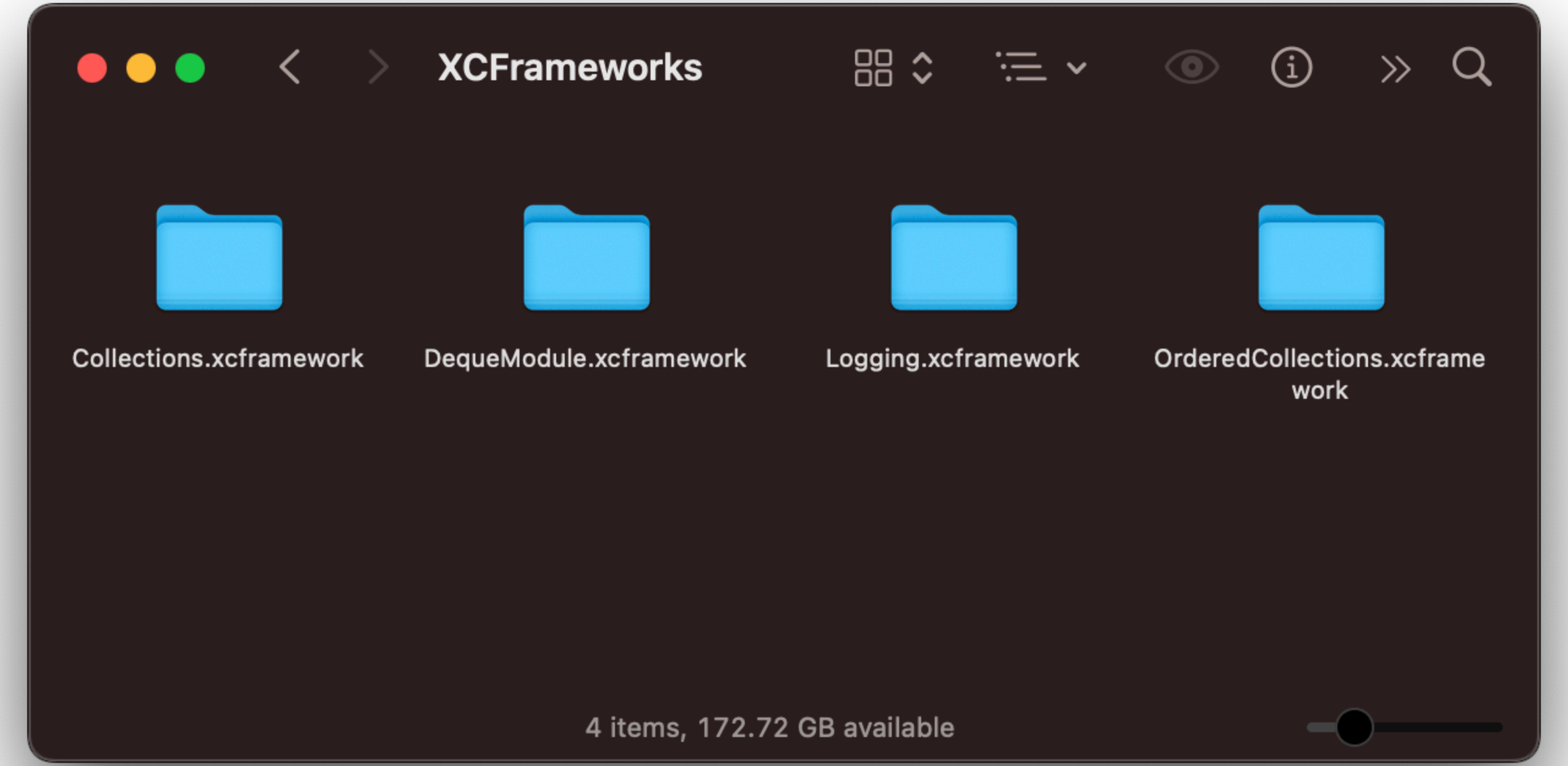
ビルド対象のプラットフォーム

パッケージ

ビルドするターゲット

2. *prepare*コマンドの実行

```
$ scipio prepare /path/to/Dependencies
🔁 Resolving Dependencies...
🗑️ Cleaning Dependencies...
📦 Building OrderedCollections for iOS
🚀 Combining into XCFramework...
📦 Building DequeModule for iOS
🚀 Combining into XCFramework...
📦 Building Collections for iOS
🚀 Combining into XCFramework...
📦 Building Logging for iOS
🚀 Combining into XCFramework...
✅ Succeeded.
```


XCFrameworks
Collections.xcframework
DequeModule.xcframework
Logging.xcframework
OrderedCollections.xcframework
4 items, 172.72 GB available

*create*モード

- SwiftPMでパッケージマニフェストから依存関係を解決

- パッケージからXCFrameworkを生成

```
$ scipio create path/to/swift-log —platforms iOS —support-simulators
🔁 Resolving Dependencies...
📦 Building Logging for iOS, iPhone Simulator
🚀 Combining into XCFramework...
✅ Succeeded.
```

Article

Prepare All Dependencies for Your Application

Use `prepare` mode to generate required frameworks for a project

Concept

The concept of Scipio, all dependencies wanted to be used in your application should be defined in one Package manifest.

`prepare` command in to build dependencies as XCFrameworks in the manifest.

詳しくはDocC

Practical Usage

Let's see how to use scipio in `prepare` mode.

1. Create a New Swift Package to Describe Dependencies

First, create a new Swift Package to describe required dependencies.

Generally, it's recommended to make this in the same directory as your application's Xcode project.

```
$ mkdir MyAppDependencies
$ cd MyAppDependencies
$ swift package init
```

https://giginet.github.io/Scipio/documentation/scipio

キャッシュシステム

Scipioのキャッシュシステム

- Scipioはキャッシュシステムを備えている
- prepareモードは生成済みの成果物があれば再利用ができる

```
$ scipio prepare /path/to/Dependencies
🔁 Resolving Dependencies...
🗑️ Cleaning Dependencies...
✅ Valid OrderedCollections.xcframework is exists. Skip building.
✅ Valid DequeModule.xcframework is exists. Skip building.
✅ Valid Collections.xcframework is exists. Skip building.
✅ Valid Logging.xcframework is exists. Skip building.
✨ Succeeded.
```

生成済みのフレームワークは
再利用される

VersionFile

- **VersionFile**というビルドコンテキストを含んだJSONファイルを生成
- ビルド前にこれからビルドするパッケージと、ビルド済み成果物の同一性をチェック
- 同一のものがあればキャッシュを利用してビルド

.MyFramework.version

MyFramework.xcframework

```
{
  "buildOptions" : {
    "buildConfiguration" : "debug",
    "enableLibraryEvolution" : false,
    "extraBuildParameters" : {
      "COPY_PHASE_STRIP" : "NO",
      "SWIFT_OPTIMIZATION_LEVEL" : "-Onone"
    },
    "frameworkType" : "static",
    "isDebugSymbolsEmbedded" : true,
    "sdks" : [
      "iOS",
      "iOSSimulator",
      "watchOS",
      "watchOSSimulator"
    ]
  },
  "clangVersion" : "clang-1403.0.22.14.1",
  "pin" : {
    "revision" : "32e8d724467f8fe623624570367e3d50c5638e46",
    "version" : "1.5.2"
  },
  "scipioVersion" : "9456405b33f2a25c479ea31cad0ba3c0222d9e20",
  "targetName" : "Logging"
}
```

Projectキャッシュ

- 出力先ディレクトリのみをキャッシュとして扱うポリシー
- 出力先のXCFrameworkがこれからビルドするものと同一ならばスキップ

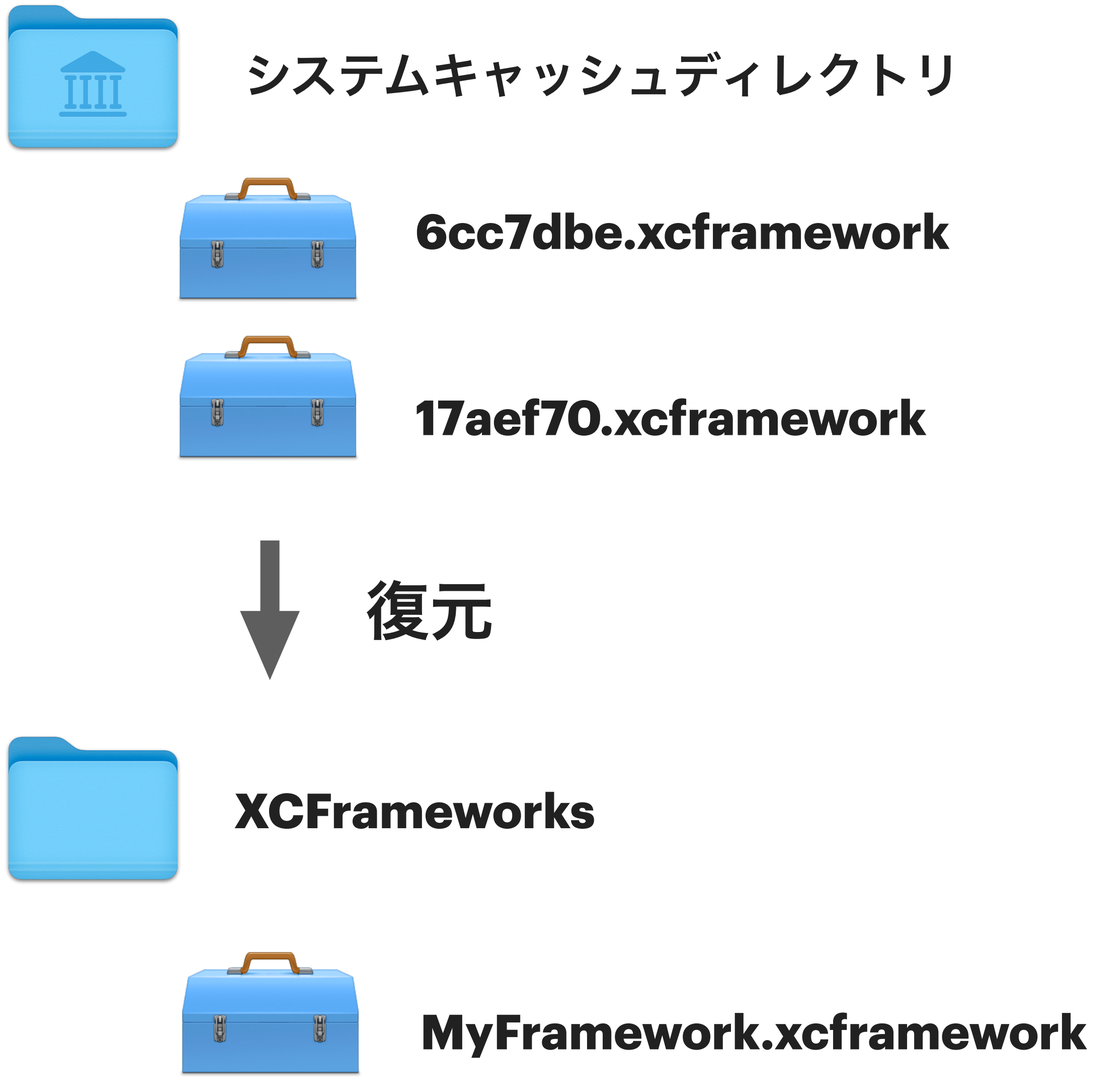
XCFrameworks

.MyFramework.version

MyFramework.xcframework

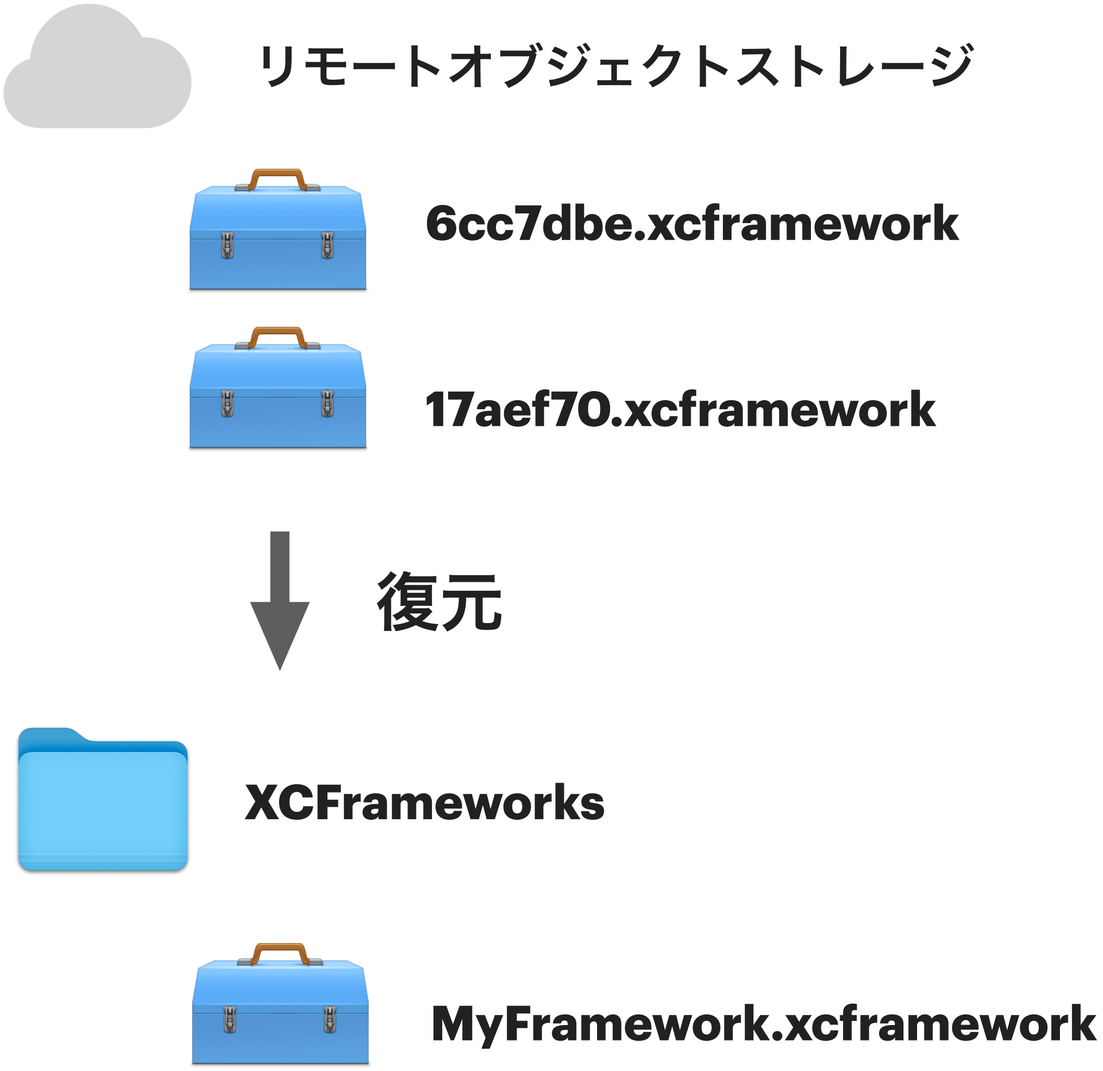
Local Diskキャッシュ

- システムキャッシュディレクトリ(~/Library/Cache)に成果物を待避
- ビルド時にVersionFileのハッシュからキャッシュがあれば復元
  - 同じマシン内で過去にビルドしたことがある成果物を使い回せる
  - 複数のXcodeバージョンの切り替え時やブランチの切り替え時に有用



Remote Diskキャッシュ

- Local Disk Cacheをリモートディスクに拡張
- 開発者間でキャッシュの再利用が可能に
- Amazon S3用のバックエンドを提供



XCFrameworkの生成手法

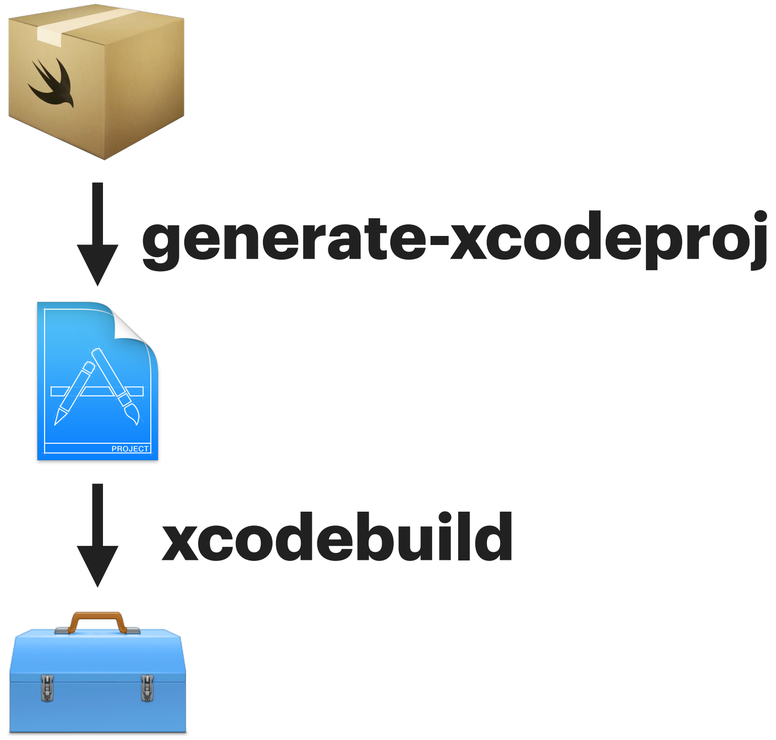
swift-create-xcframework

- unsignedapps/swift-create-xcframework
- SwiftPMからXCFrameworkを作成するツール
  - Scipioのcreateモードに近い
- xcodeprojを生成し、xcodebuildでビルドしている
  - xcodeprojの生成にはgenerate-xcodeprojというSwiftPMのサブコマンドを使っている

旧手法：xcodeprojを使ったFramework生成

- SwiftPMからxcodeprojを生成→xcodebuildでビルド
- SwiftPMにはxcodeprojを生成するコマンドが付属している

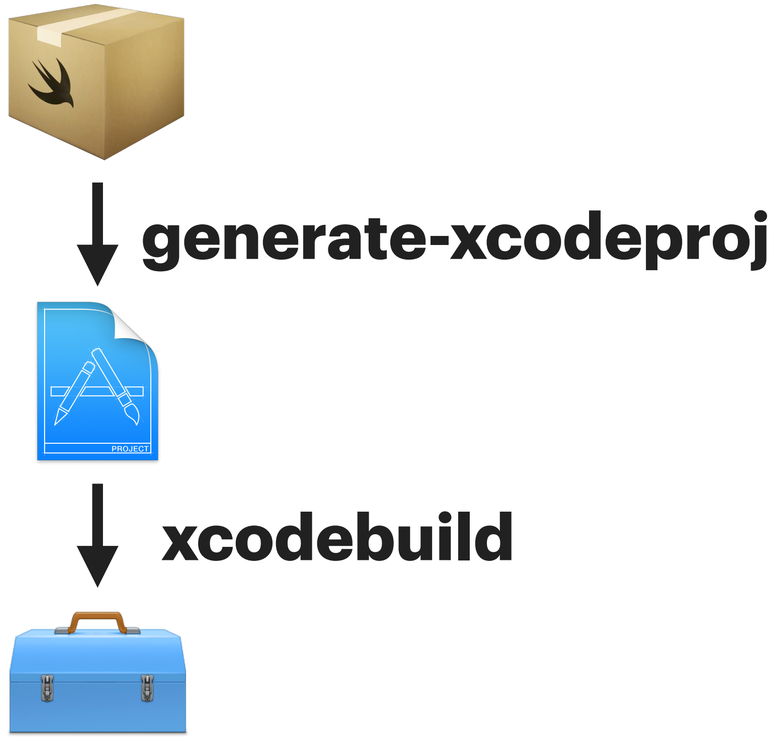
```
$ swift package generate-xcodeproj
```



旧手法：xcodeprojを使ったFramework生成

- SwiftPMからxcodeprojを生成→xcodebuildでビルド
- SwiftPMにはxcodeprojを生成するコマンドが付属している
  - →が、Swift 5.8(Xcode 14.0)から削除

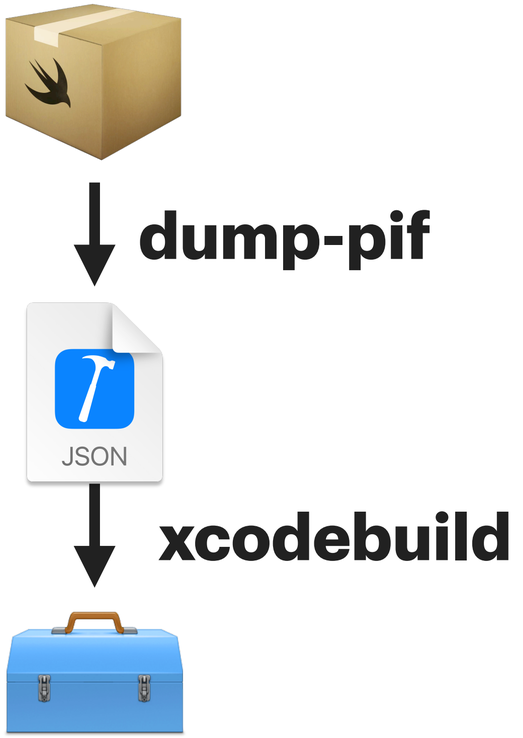
```
$ swift package generate-xcodeproj
```



提案手法：PIFを使ったFramework生成

- **PIF**と呼ばれる内部表現を利用してフレームワーク
  ビルドする手法

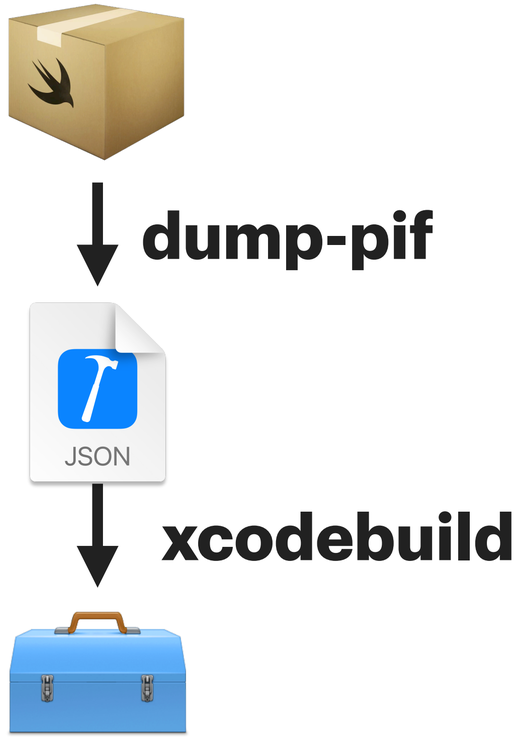
  - ScipioはSwiftPMが生成するPIFを適宜書き換え
    てフレームワークを生成している



PIF(The Project Interchange Format)

- SwiftPMがビルドシステムと通信するためのフォーマット
  - 実態はJSON
- xcodeprojの構造に酷似
- SwiftPMのサブコマンドで生成出来る

```
$ swift package dump-pif
```



```
[
  {
    "contents" : {
      "guid" : “Workspace:/Users/giginet/.ghq/github.com/apple/swift-log@11",
      "name" : "swift-log",
      "path" : "/Users/giginet/.ghq/github.com/apple/swift-log",
      "projects" : [
        "c44c70d170d7f592dc973df88b2326051b86b7958e1560c32c2b31d19fb714aa",
        "939870cd480866e3c56cbb743773b1add4e3a3d79277ede53c31eb6377cbea92",
        "024e214b7e54ed6ea3153ebf0698b3ab88ca40336b27bc1c9dbdb39d2d261ccf"
      ]
    },
    "signature" : "35c4ad5e4b5e02d91fb54f4805e7aed41ba7f21d782e1622519ee2cd2b8980e4",
    "type" : "workspace"
  },
  {
    "contents" : {
      "buildConfigurations" : [
        {
          "buildSettings" : {
            "CLANG_ENABLE_OBJC_ARC" : "YES",
            "CODE_SIGN_IDENTITY" : "",
            "CODE_SIGNING_REQUIRED" : "NO",
            "COPY_PHASE_STRIP" : "NO",
            "DEBUG_INFORMATION_FORMAT" : "dwarf",
            "DRIVERKIT_DEPLOYMENT_TARGET" : "19.0",
            "DYLIB_INSTALL_NAME_BASE" : "@rpath",
            "ENABLE_NS_ASSERTIONS" : "YES",
            "ENABLE_TESTABILITY" : "YES",
            "ENABLE_TESTING_SEARCH_PATHS" : "YES",
            "ENTITLEMENTS_REQUIRED" : "NO",
            "FRAMEWORK_SEARCH_PATHS[__platform_filter=ios;ios-simulator]" : [
              "$(inherited)",
              "$(PLATFORM_DIR)/Developer/Library/Frameworks"
            ],
            "FRAMEWORK_SEARCH_PATHS[__platform_filter=macos]" : [
              "$(inherited)",
```


XCConfigみたいなのが埋まってる

PIFを使ったパッケージのビルド

- パッケージのビルドにXcodeを使うモードを選べる

```
$ swift build --build-system xcode
```

- SwiftPMが内部的にPIFを生成し、Xcodeに渡してパッケージをビルド

XCBuild

- XcodeやxcodebuildがInternalで利用しているビルドツールらしい
  - XcodeのToolchainに付属している
- 引数にPIFを渡すとそこからビルドしてくれる

```
 $ export xcbuild=/Applications/Xcode.app/Contents/
SharedFrameworks/XCBuild.framework/Versions/A/Support/
xcbuild
 $ xcbuild build my-package.pif \
  --target MyPackage \
  --derivedDataPath ./build/scipio
```

XCFrameworkの生成

- xcodebuildの*-create-xcframework*コマンドでできる
- 複数のFrameworkを渡して統合するだけ

```
$ xcodebuild -create-xcframework \
 -framework Release-iphoneos/MyPackage.framework \
 -framework Release-iphonesimulator/MyPackage.framework \
 -output MyFramework.xcframework
```

- **https://help.apple.com/xcode/mac/11.0/#/dev544efab96**

フレームワークの構造と課題

フレームワークの構造と課題

- Frameworkの種類と課題
  - Swiftターゲット
    - Library Evolutionの制約
  - Clangターゲット
    - ヘッダー構造の問題
- リソースサポートの仕組み

Swiftターゲット

- Swiftのみで構成されたターゲット
- ほとんどの場合、簡単にフレームワーク化できる
- 一部それだけではビルドできない例がある（後述）

MyFramework.framework

Info.plist

メタデータ

MyFramework

ライブラリ本体

Modules

インターフェイス公開するバイナリを持つ
（swiftmodule）

MyFramework.swiftmodule

Library Evolution

- ビルド済みのフレームワークをSwiftバージョンによらずに使えるようにする仕組み（ABI Stability）
  - https://www.swift.org/blog/library-evolution/
  - *BUILD_LIBRARY_FOR_DISTRIBUTION*を有効化すると生成
- Frameworkに**swiftinterface**という定義が生成される
  - 実態はテキストファイル

MyFramework.framework

Info.plist

MyFramework

Modules

MyFramework.swiftmodule

MyFramework.swiftinterface

swiftinterface
コンパイラバージョンに寄らず互換が保たれるインターフェイス

```
// swift-interface-format-version: 1.0
// swift-compiler-version: Apple Swift version 5.8.1 (swiftlang-5.8.0.124.5 clang-1403.0.22.11.100)
// swift-module-flags: -target arm64-apple-ios11.0-simulator -enable-objc-interop -enable-library-evolution
-swift-version 5 -enforce-exclusivity=checked -O -module-name Logging
// swift-module-flags-ignorable: -enable-bare-slash-regex
import Darwin
import Swift
import _Concurrency
import _StringProcessing
public protocol LogHandler : Logging._SwiftLogSendableLogHandler {
  var metadataProvider: Logging.Logger.MetadataProvider? { get set }
  func log(level: Logging.Logger.Level, message: Logging.Logger.Message, metadata: Logging.Logger.Metadata?,
source: Swift.String, file: Swift.String, function: Swift.String, line: Swift.UInt)
  @available(*, deprecated, renamed: "log(level:message:metadata:source:file:function:line:)")
  func log(level: Logging.Logger.Level, message: Logging.Logger.Message, metadata: Logging.Logger.Metadata?, file:
Swift.String, function: Swift.String, line: Swift.UInt)
  subscript(metadataKey _: Swift.String) -> Logging.Logger.Metadata.Value? { get set }
  var metadata: Logging.Logger.Metadata { get set }
  var logLevel: Logging.Logger.Level { get set }
}
extension Logging.LogHandler {
  public var metadataProvider: Logging.Logger.MetadataProvider? {
    get
    set
  }
}
extension Logging.LogHandler {
  @available(*, deprecated, message: "You should implement this method instead of using the default
implementation")
  public func log(level: Logging.Logger.Level, message: Logging.Logger.Message, metadata:
Logging.Logger.Metadata?, source: Swift.String, file: Swift.String, function: Swift.String, line: Swift.UInt)
  @available(*, deprecated, renamed: "log(level:message:metadata:source:file:function:line:)")
```

Library Evolutionの制限

- 一部のパッケージはLibrary Evolutionに対応できないためビルドできない
  - swift-nio, swift-collectionsなども
    - 実装のインライン化など、互換性維持ができない言語仕様を使用しているケースがあるため
  - これらをフレームワーク化するときは無効化する必要がある

Library Evolutionを無効化したXCFramework

- 普通にXCFrameworkを作ると、Library Evolutionが無効化されたFrameworkからは生成できない（エラーが出てしまう）

- Library Evolutionを無効化したフレームワークをXCFramework化する場合、`-allow-internal-distribution` が必要

```
$ xcodebuild -create-xcframework \
 -framework Release-iphoneos/MyPackage.framework \
 -framework Release-iphonesimulator/MyPackage.framework \
 -output MyFramework.xcframework \
 -allow-internal-distribution
```

ScipioでのLibrary Evolution

- 普通はXCFrameworkの生成はLibrary Evolutionの有効化が推奨される
- Scipioのユースケースのように同じプロジェクト内での利用が保証される場合、無効にすることもできる
  - Scipioではデフォルトでオフになっている
    - —enable-library-evolutionで有効化

Clangターゲット

- C, ObjCなどの実装を持つターゲット
- Swiftターゲットとは異なりHeader, modulemapの出力が必須
- この辺の構成によりシンプルにFramework化できない例がある

MyFramework.framework

Info.plist

メタデータ

MyFramework

ライブラリ本体

Modules

module.modulemap

modulemap

Headers

MyFramework.h

ヘッダー

modulemapとは

- 複数のヘッダーをモジュールとして扱うためのマッピングを記述するファイル

```
framework module MyFramework {
  header “MyFramework.h"
  header “Model.h"
  header “Runner.h"
  export *
}
```

モジュール名

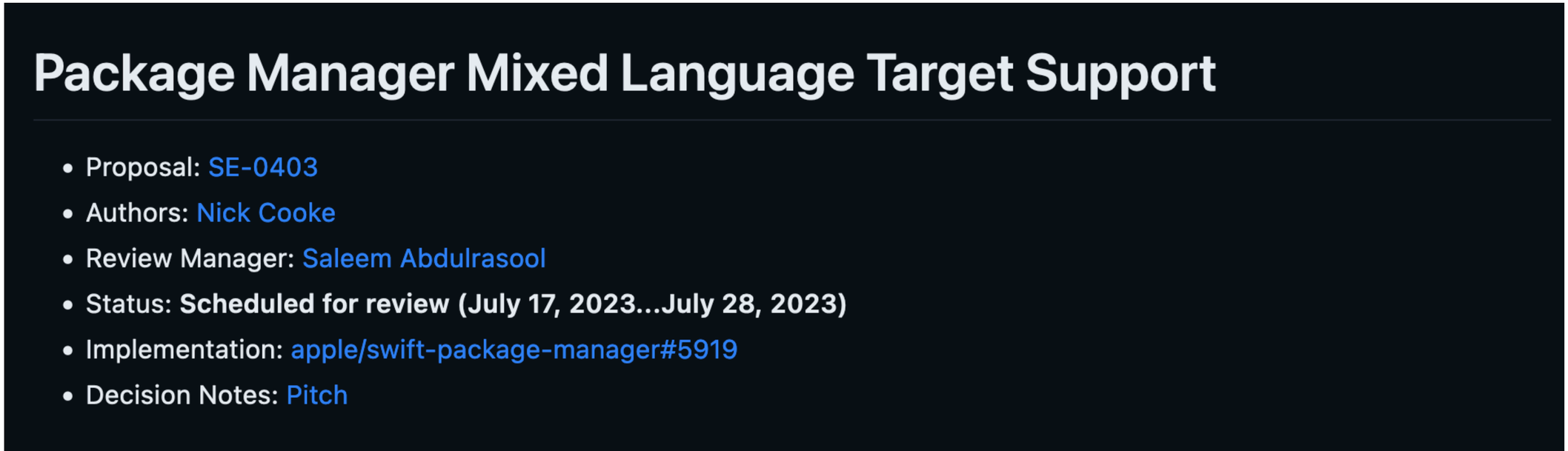
- これによって *import MyFramework* でヘッダーに定義されたインターフェイスを参照できる

Clangターゲットのフレームワーク化

- Swiftターゲットのフレームワーク化と異なり複雑
- ヘッダーのinclude指定によりフレームワーク化したときに壊れることがある
  - Scipioでビルドできないケースの多くはこれに起因する

Mixed Languageターゲット

- SwiftとC Familyなど、複数の言語を持つターゲット
- 現状のSwiftPMでは未対応のため、Scipioでもサポートしていない
- SE-403 Package Manager Mixed Language Target Supportとして実装が決まっているがまだ使えない



Resourceサポート

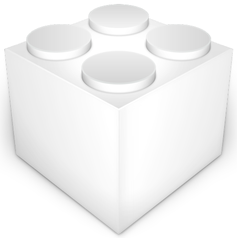
- Swift Packageはリソースが扱える
- Framework化した場合もリソースバンドルを含むことができる
  - リソースバンドルはPIFの時点で定義されているため、ビルドシステムが勝手に作ってくれる

MyFramework.framework

Info.plist

MyFramework

Modules



MyFramework.bundle

リソースバンドル（ディレクトリ）

Bundle.moduleサポート

- Swift PackageはBundle.moduleからリソースにアクセスできる

```
let resourceURL = Bundle.module.url(
    forResource: "image",
    withExtension: “png”
)
```

- 実態はSwiftPMがビルド時にextensionを挿入している
- Scipioでフレームワーク化した場合も同じように使える

```
340             mainPathSubstitution =
341                 #"Bundle.main.bundleURL.appendingPathComponent("\#(bundlePath.basename.asSwiftStringLiteralConstant)").path"#
342         }
343
344         let content =
345             """
346             \(self.toolsVersion < .vNext ? "import" : "@_implementationOnly import") Foundation
347
348             extension Foundation.Bundle {
349                 static let module: Bundle = {
350                     let mainPath = \(mainPathSubstitution)
351                     let buildPath = "\(bundlePath.pathString.asSwiftStringLiteralConstant)"
352
353                     let preferredBundle = Bundle(path: mainPath)
354
355                     guard let bundle = preferredBundle ?? Bundle(path: buildPath) else {
356                         fatalError("could not load resource bundle: from \\(mainPath) or \\(buildPath)")
357                     }
358
359                     return bundle
360                 }()
361             }
362             """
363
364         let s
365
366         // Add the file to the derived sources.
367         self.derivedSources.relativePaths.append(subpath)
```

SwiftPMがビルド時にextension用のコードを生成して追加している

https://github.com/apple/swift-package-manager/blob/667a00670becac532b85dcd46caeacd1c3a5bdda/Sources/Build/BuildDescription/SwiftTargetBuildDescription.swift#L324

Scipioの利点

- SwiftPM + Xcodeをベースとしているため**壊れづらい**
  - 依存解決や取得を通常のSwiftPMと同様に行う
  - SwiftPMがメンテされる限りは大きく変更されづらい

Scipioの利点

- **辞めやすい**
  - 依存がSwift Packageであることは変わらないため、通常利用で問題ない場合は、通常の統合方法に戻せる
  - 部分的にXcode統合を行い、キャッシュしたい部分だけXCFrameworkとして扱うこともできる

第2章まとめ

- Scipioというビルドツールを作ってXCFrameworkを生成している
  - SwiftPMの基盤に乗ることで、なるべく簡単に生成できるようにしている
- キャッシュシステムもあり、成果物の再利用や共有が容易
- どうぞご利用ください
  - https://github.com/giginet/Scipio

第3章

LINE iOSの
ビルドシステムでの活用

LINE iOSビルド基盤への適応

- 開発したビルドツールでLINE iOSのビルド基盤をどのように改善したか
- Prebuilt TaskからどのようにBazelの利用を外していくか

Bazel

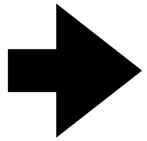
Prebuiltタスク

Prebuiltタスクの標準化

- OSSライブラリのFramework化
- 社内実装のFramework化
- バイナリフレームワークの取得
- APIスキームからのコード生成

Prebuiltタスクの標準化

- OSSライブラリのFramework化
- 社内実装のFramework化
- バイナリフレームワークの取得
- APIスキームからのコード生成



XCFramework化

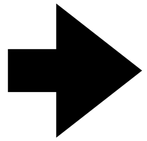
XCFramework or Xcodeプロジェクト化

Swift Package化

Build Tools Plugin化

Prebuiltタスクの標準化

- **OSSライブラリのFramework化**
- 社内実装のFramework化
- バイナリフレームワークの取得
- APIスキームからのコード生成



XCFramework化

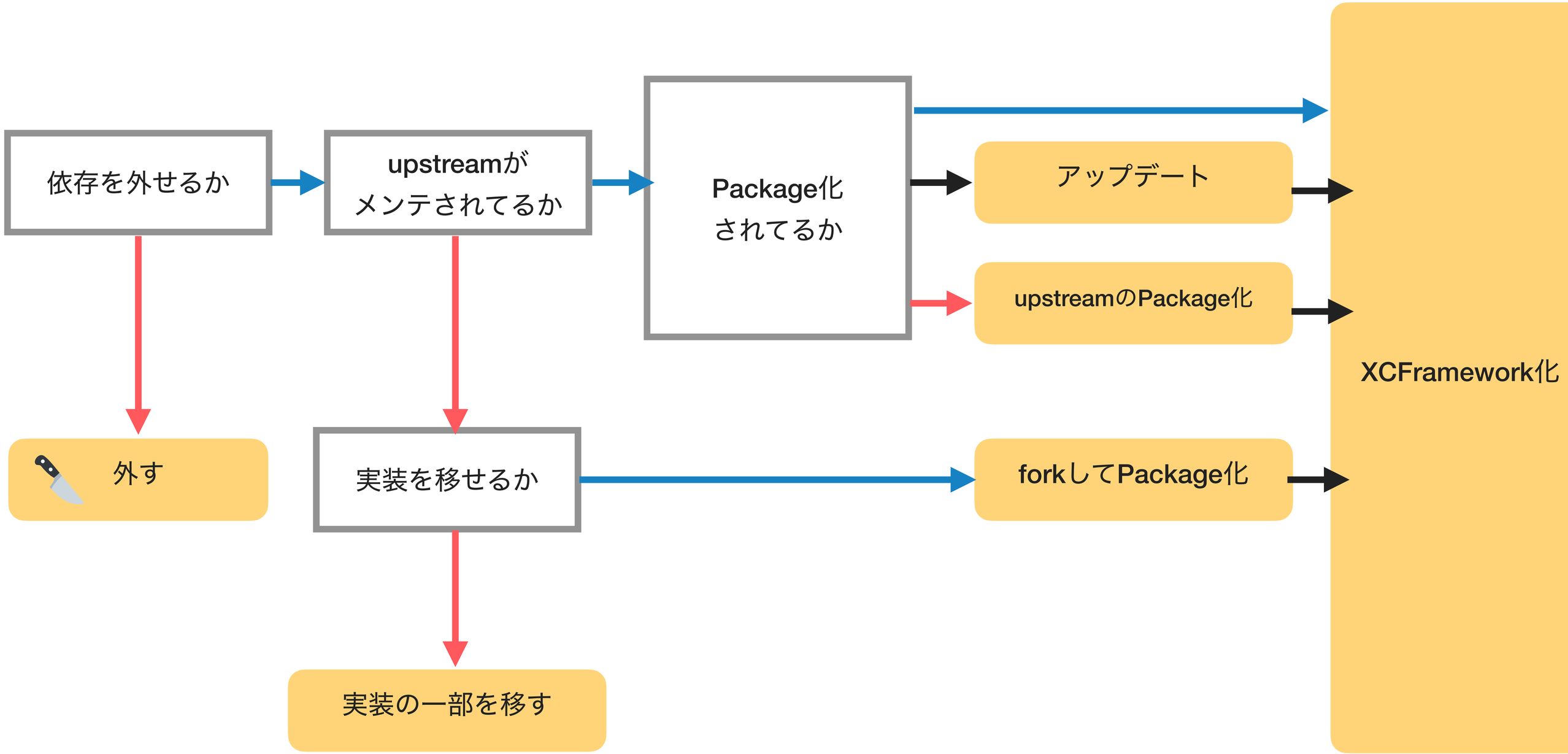
XCFramework or Xcodeプロジェクト化

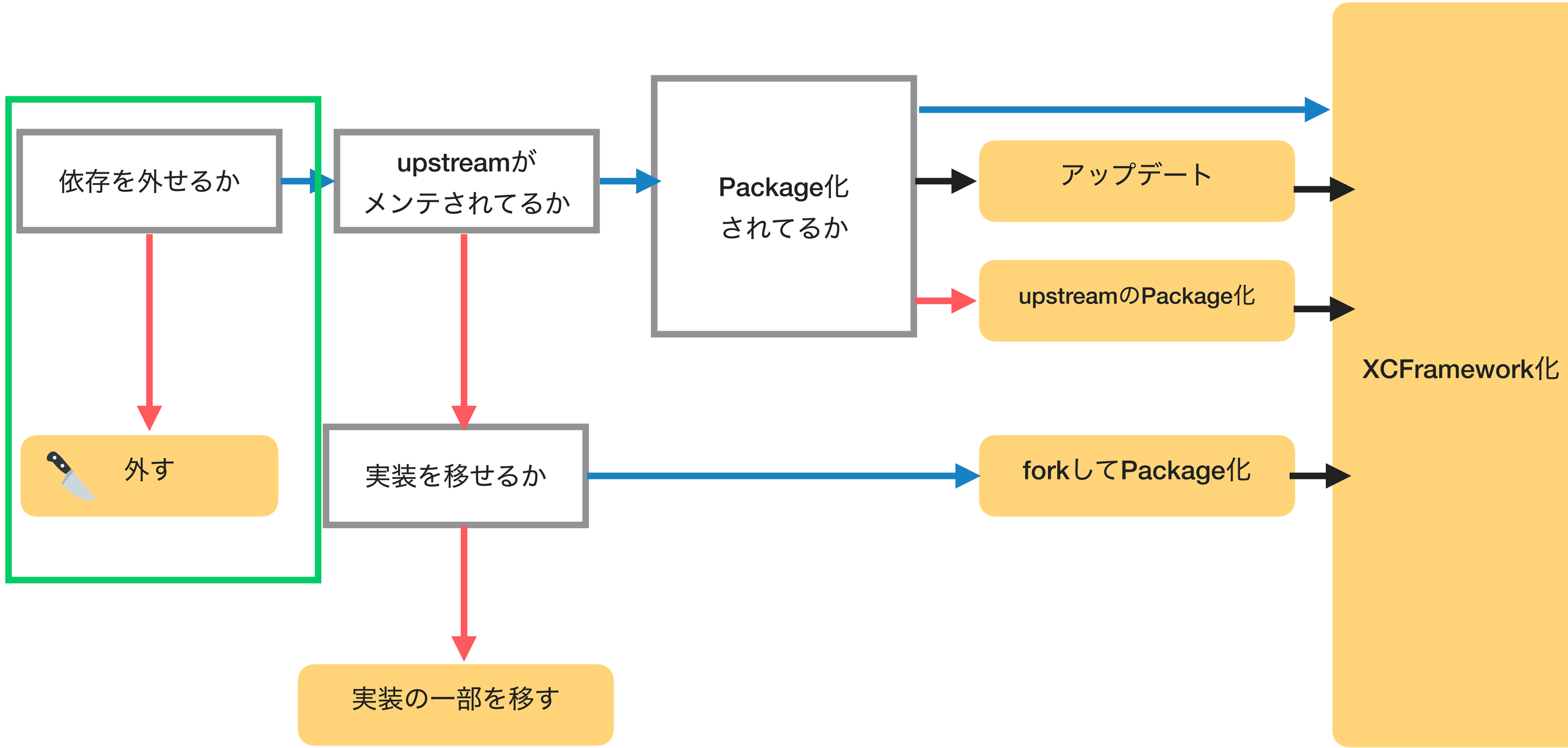
Swift Package化

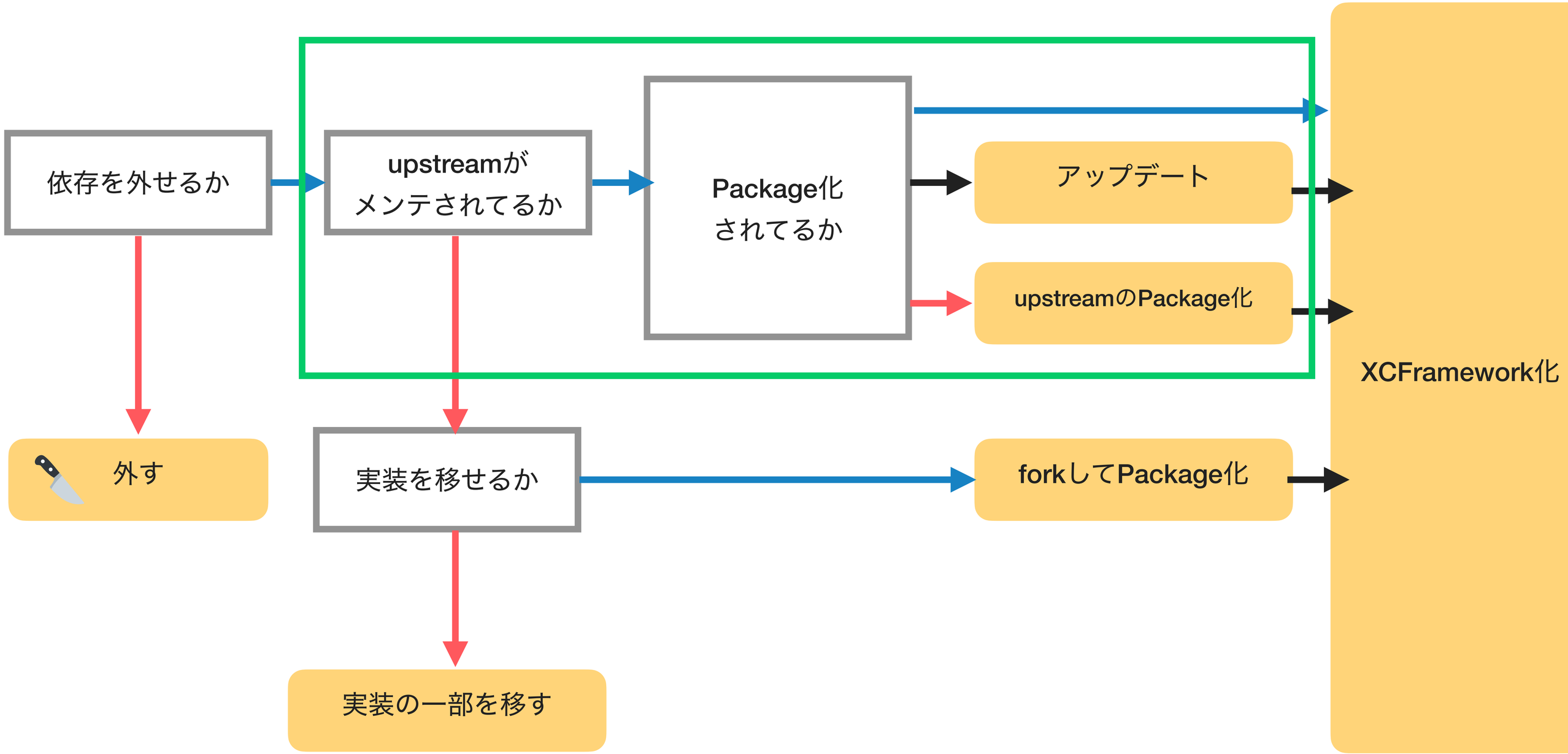
Build Tools Plugin化

OSSライブラリの状況

- 60~70近いOSSライブラリ
  - 利用状況が不明なもの
  - 数年前からバージョンが置いていかれてるもの
  - upstreamがobsoleteになっているもの
  - forkしてpatchを当てているもの

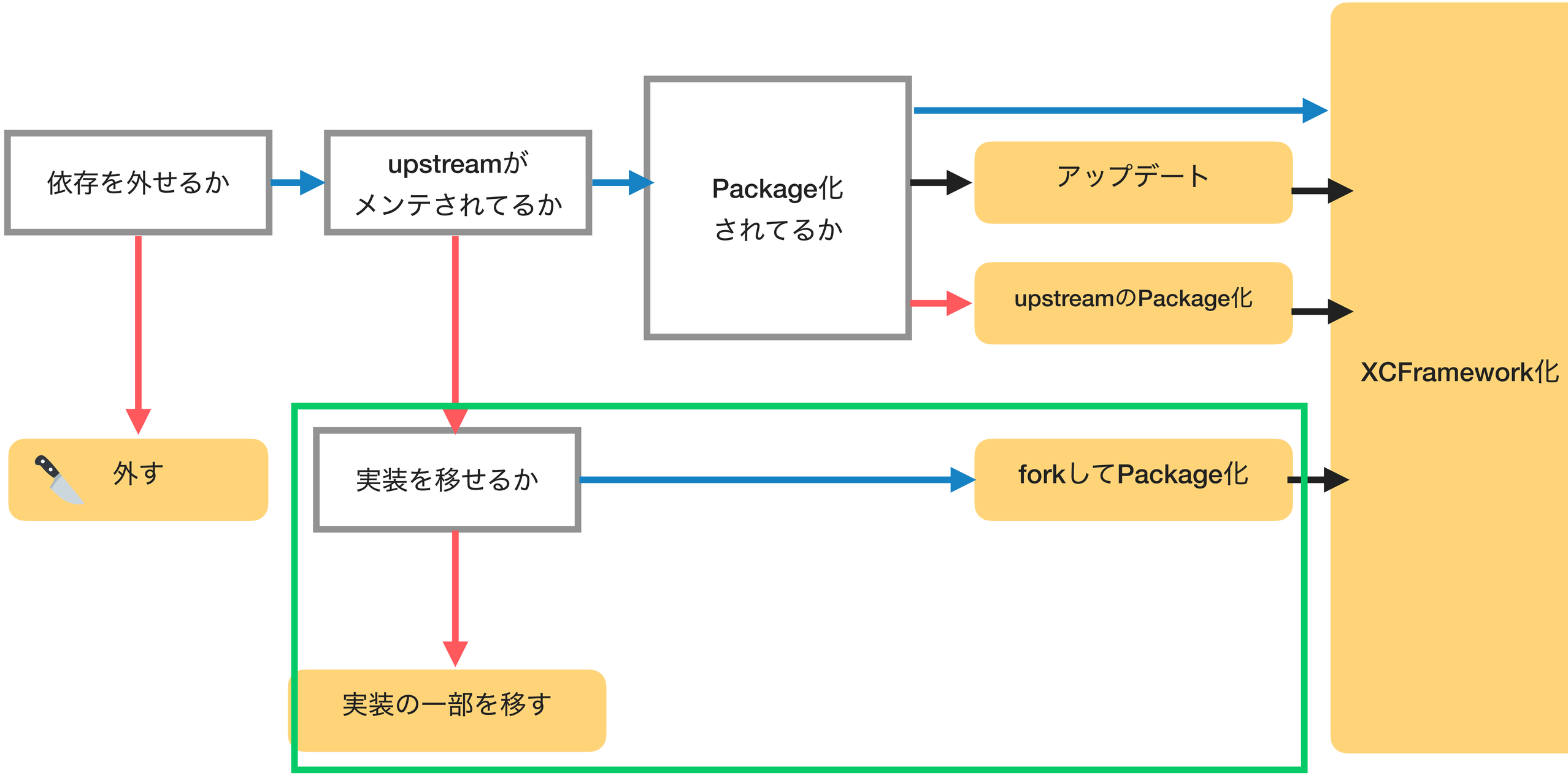

依存を外せるか
upstreamが
メンテされてるか
Package化
されてるか
アップデート
upstreamのPackage化
XCFramework化
外す
実装を移せるか
forkしてPackage化
実装の一部を移す


依存を外せるか
upstreamが
メンテされてるか
Package化
されてるか
アップデート
upstreamのPackage化
外す
実装を移せるか
forkしてPackage化
実装の一部を移す
XCFramework化


依存を外せるか
upstreamが
メンテされてるか
Package化
されてるか
アップデート
upstreamのPackage化
XCFramework化
外す
実装を移せるか
forkしてPackage化
実装の一部を移す

メンテされてるライブラリ

- すでにPackage対応されているものから徐々に対応していく
- バージョンが置いて行かれてるものはアップデート
- upstreamが対応していないものは対応


依存を外せるか
upstreamが
メンテされてるか
Package化
されてるか
アップデート
upstreamのPackage化
XCFramework化
外す
実装を移せるか
forkしてPackage化
実装の一部を移す

メンテが止まったライブラリ

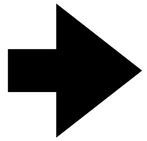
- ObjC時代のライブラリなど、数年前に統合されて、誰もメンテしないまま放置された依存関係が多数
- 一部実装だけ移せる場合は、ライセンスを維持したままコードのみを移し、依存は削除

forkは最終手段

- どうにもならない場合は最終手段でfork
  - forkをビルドしやすいようにパッケージ化
  - 将来的に消せるように実装を減らすタスク化。長期的に🔪していく

Prebuiltタスクの標準化

- OSSライブラリのFramework化
- **社内実装のFramework化**
- バイナリフレームワークの取得
- APIスキームからのコード生成



XCFramework化

XCFramework or Xcodeプロジェクト化

Swift Package化

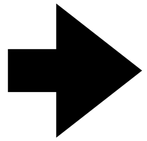
Build Tools Plugin化

社内ライブラリのSwiftPM化

- 社内ライブラリでPrebuiltフレームワーク化しやすいものはパッケージ対応
  - デモアプリの開発で標準的な利用ができるようになった
- 一部難しいパッケージ化が難しいケースはXcodeプロジェクト化
  - Mixed Language Target(Swift/ObjCの混成ターゲット)
  - 依存が多く再帰的にパッケージ化が必要なケース

Prebuiltタスクの標準化

- OSSライブラリのFramework化
- 社内実装のFramework化
- **バイナリフレームワークの取得**
- APIスキームからのコード生成



XCFramework化

XCFramework or Xcodeプロジェクト化

Swift Package化

Build Tools Plugin化

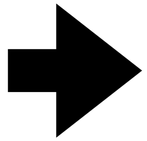
バイナリフレームワークのライブラリ化

- 深遠な事情により、ビルド済みのフレームワークを統合する必要がある
- Swift Packageの***BinaryTarget***を利用
  - ScipioはbinaryTargetも扱うことができる

```
let package = Package(
    name: "CompanySDK",
    platforms: [
        .iOS(.v13)
    ],
    products: [
        .library(
            name: "CompanySDK",
            targets: ["CompanySDK"]
        ),
    ],
    targets: [
        .binaryTarget(
            name: "CompanySDK",
            path: "path/to/CompanySDK.xcframework"
        ),
    ]
)
```

Prebuiltタスクの標準化

- OSSライブラリのFramework化
- 社内実装のFramework化
- バイナリフレームワークの取得
- **APIスキームからのコード生成**



XCFramework化

XCFramework or Xcodeプロジェクト化

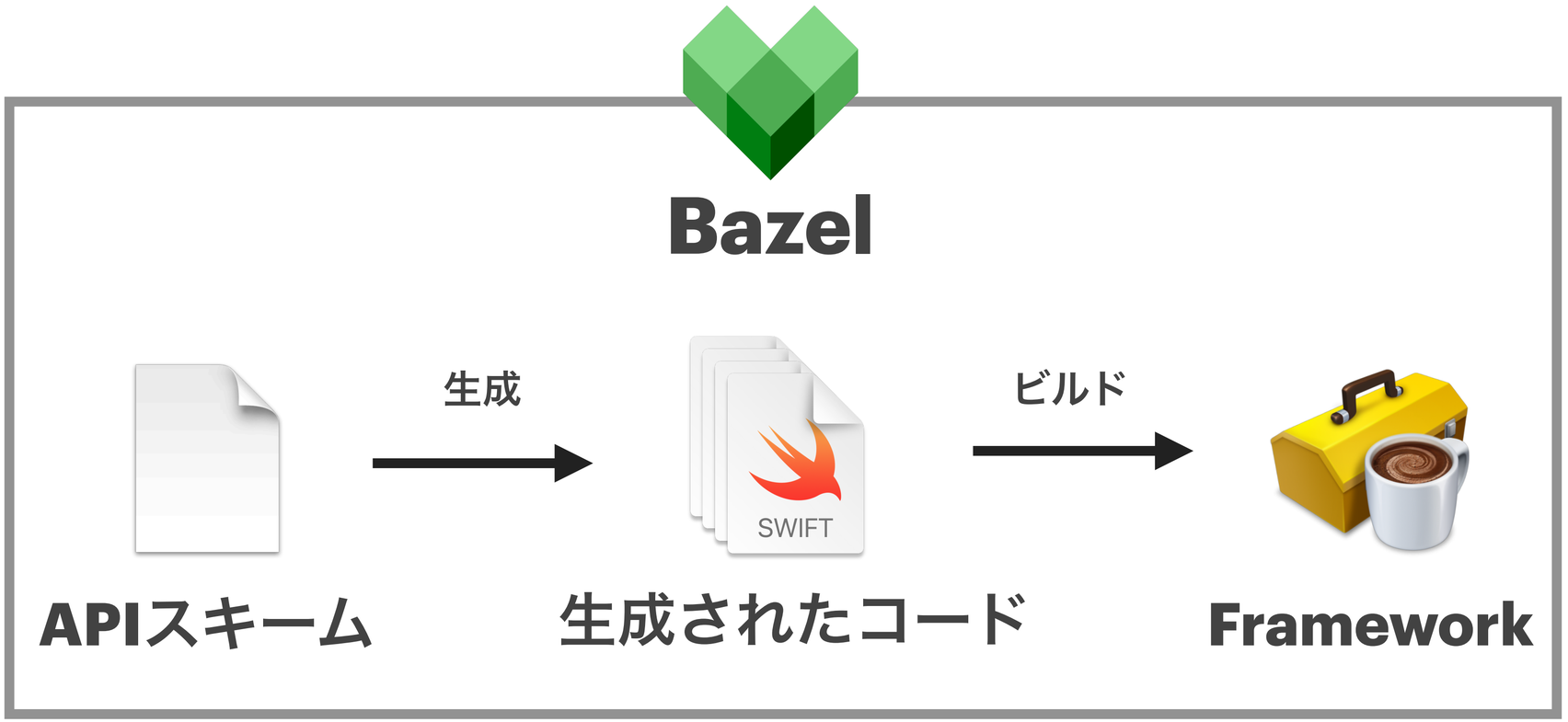
Swift Package化

Build Tools Plugin化

コード生成タスクの移行

- SwiftGenやAPIスキーマ(Apache Thrift)のコードを生成し、prebuiltフレームワークを作成していた

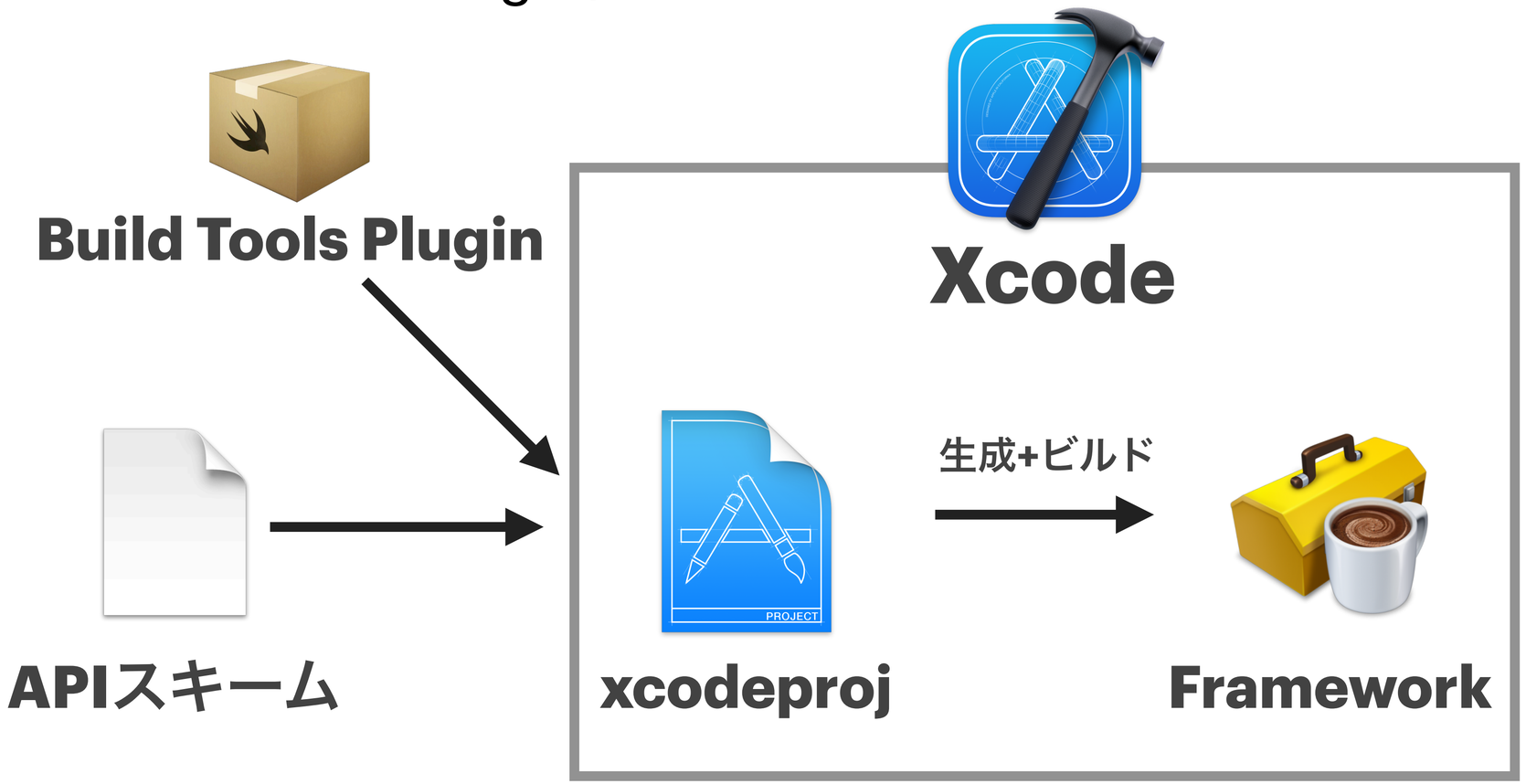
Before



コード生成タスクの移行

- prebuilt化は難しかったため、Xcodeプロジェクト化してXCWorkspaceに統合
- コード生成はBuild Tools Plugin化

After



まとめ：Bazelから標準のビルドシステムへの移行

- 多くのライブラリは**Swift Package**化してScipioで**XCFramework**にビルド
- ビルド済みフレームワークはSwift Packageの**Binary Target**化
- 複雑な構成のビルドは**Xcodeプロジェクト化 + XCWorkspace統合**
- コード生成など前処理は**Build Tools Plugin**化

Framework

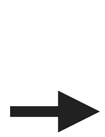
移行した！

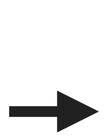
XCFramework

Apple Siliconシミュレータ対応

- 今まではBazelからFrameworkを生成していたので、複数回ビルドして強引にシミュレーター対応をしていた

Before





Library.framework(ios-arm64)

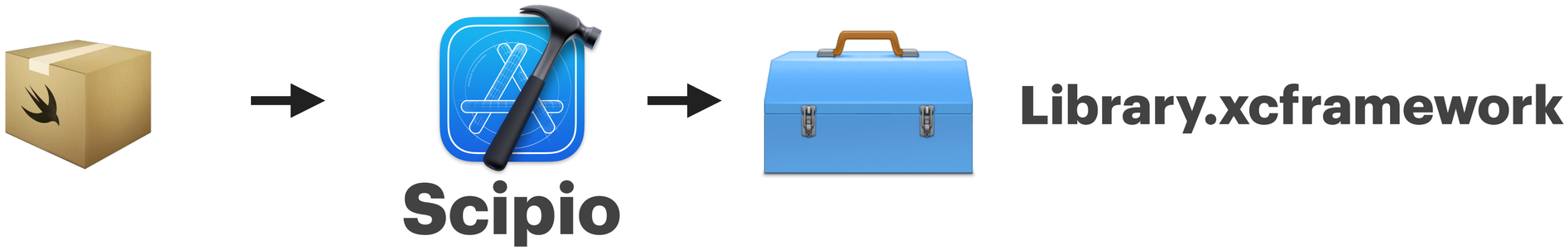
Library.framework(simulator-x86_64)

Library.framework(simulator-arm64)

Apple Siliconシミュレータ対応

- XCFrameworkに統合されることでシンプルになった

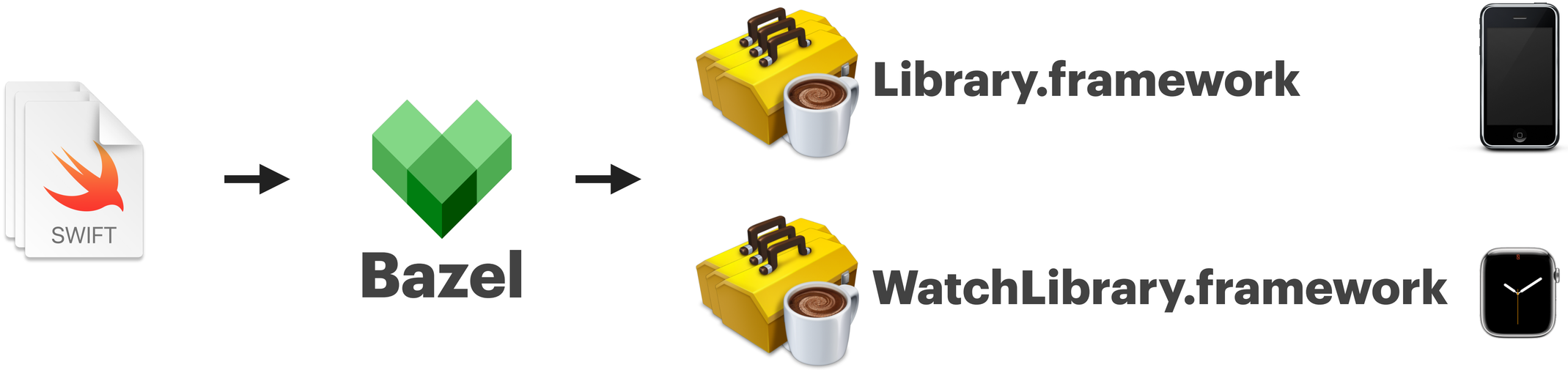
After



watchOS用フレームワーク対応

- iOS, watchOS両方から同一フレームワークを利用するとき、リネームして別名のwatchOS用を作成していた（シミュレーター用も）

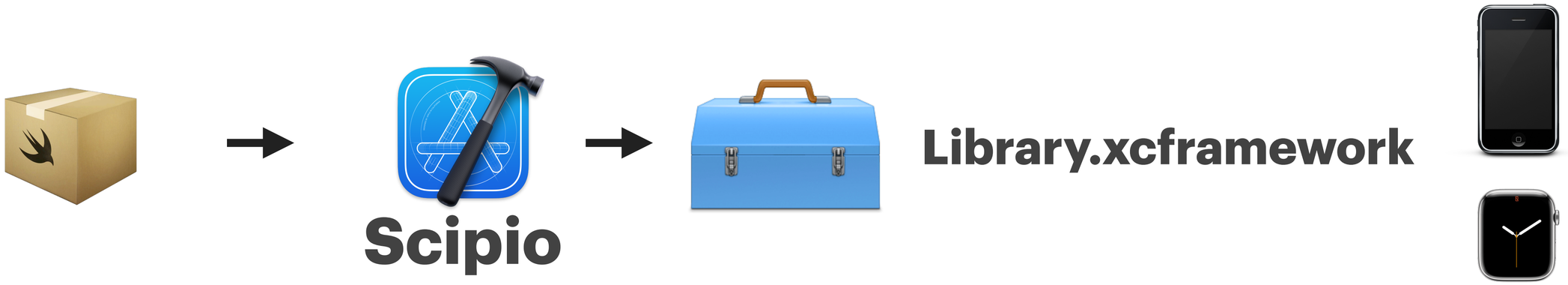
Before



watchOS用フレームワーク対応

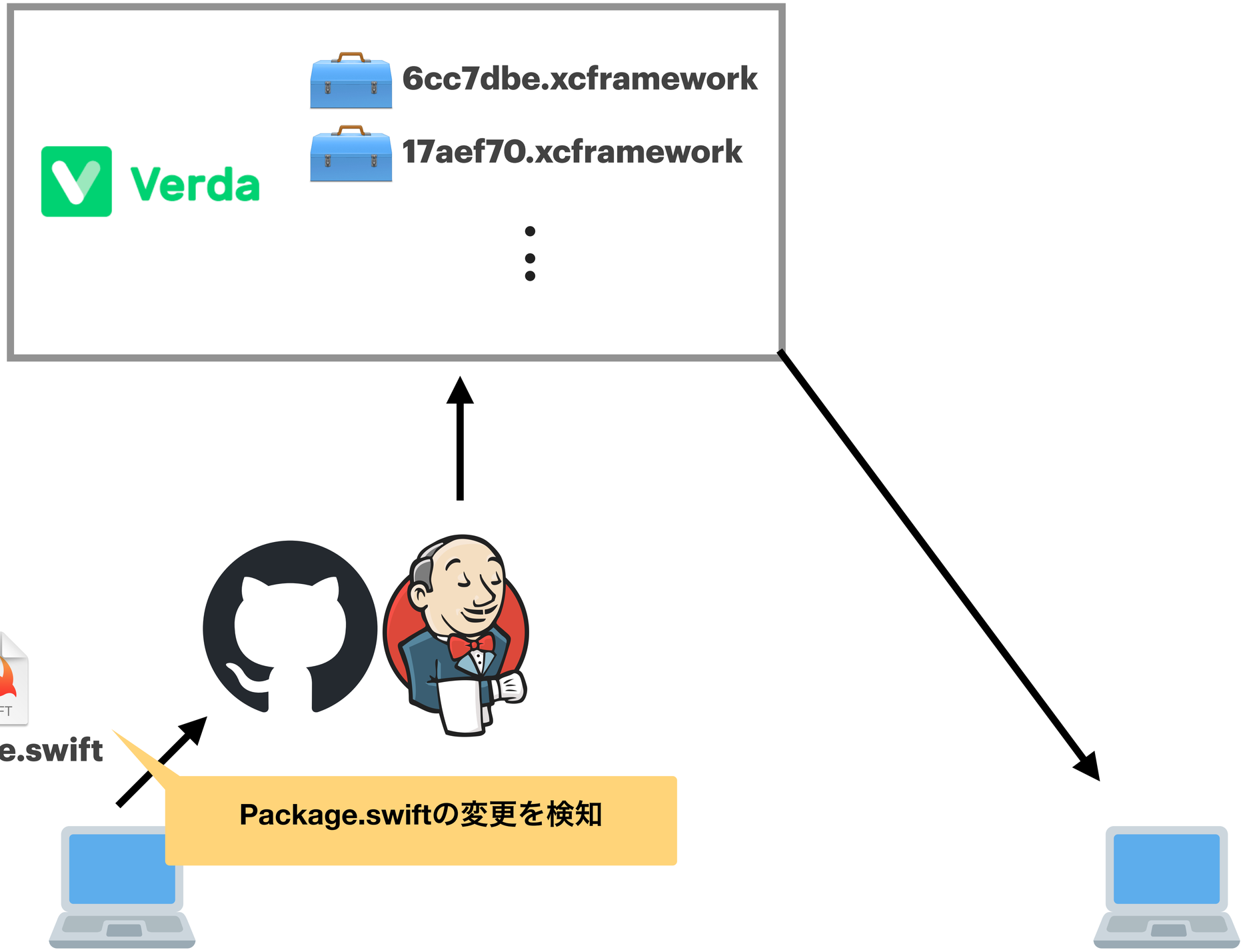
- 1つのXCFrameworkに複数のプラットフォームを含めるようになったので、1つに統合できた。リネームも不要

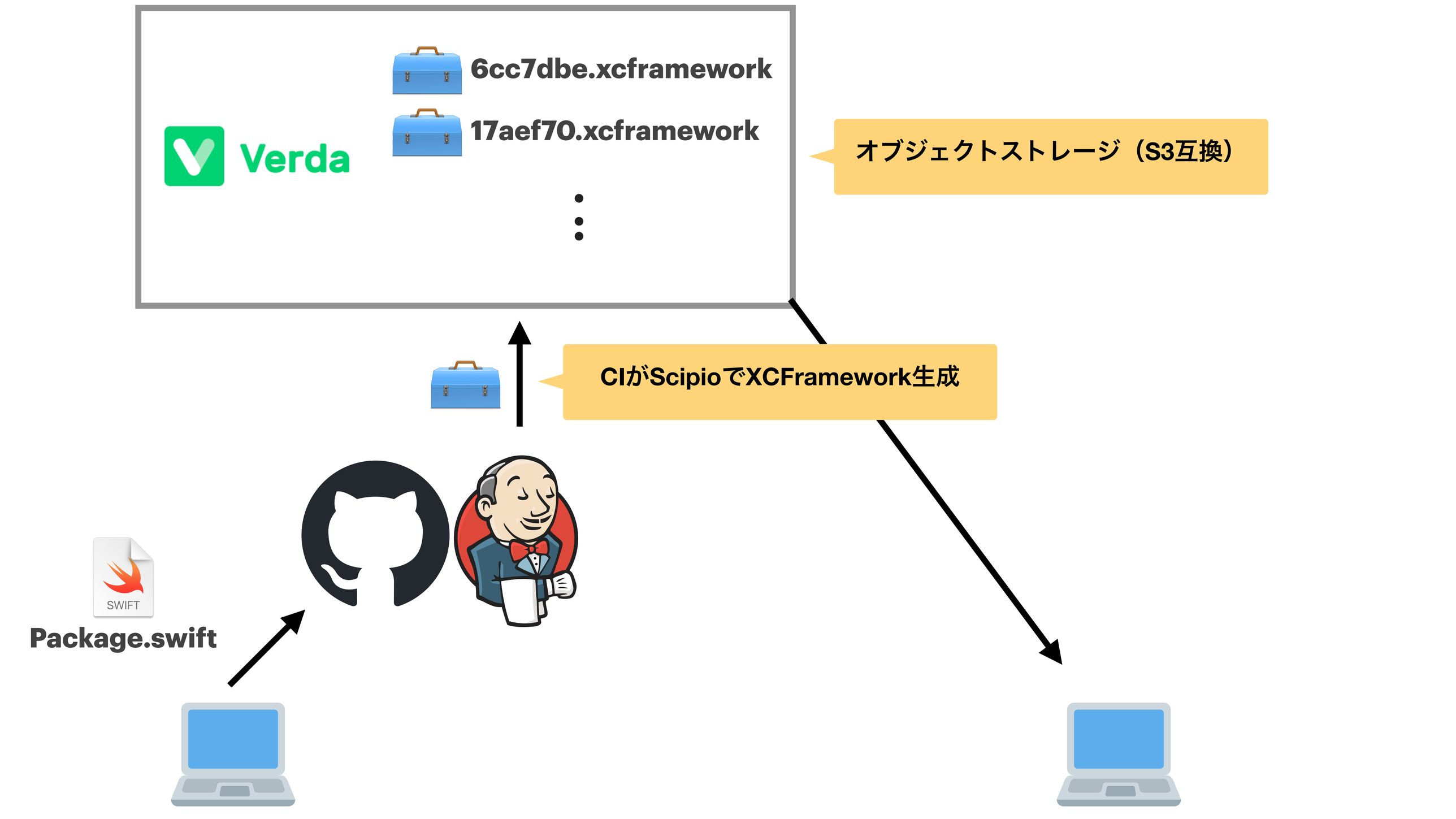
After

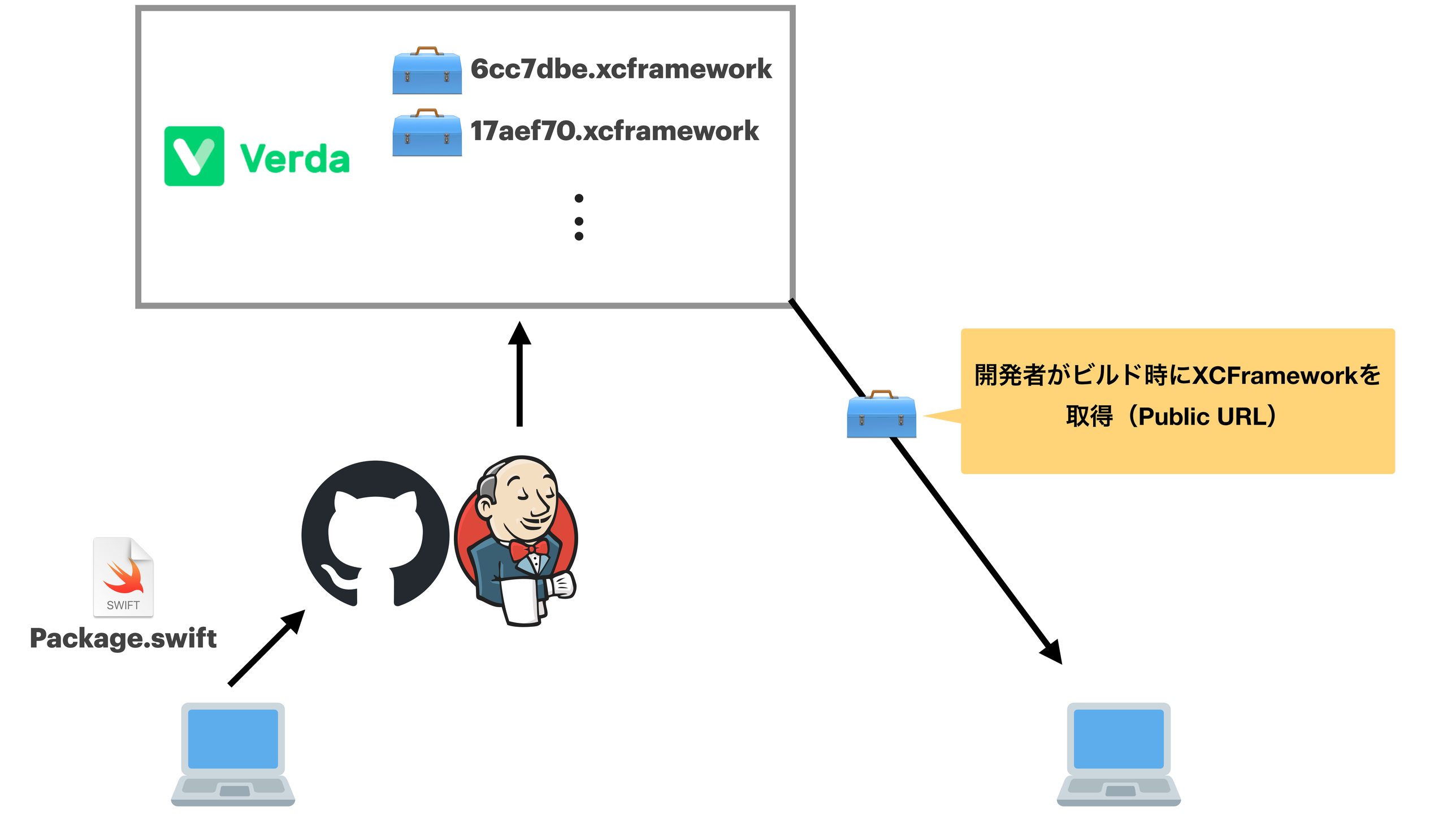


リモートキャッシュの運用

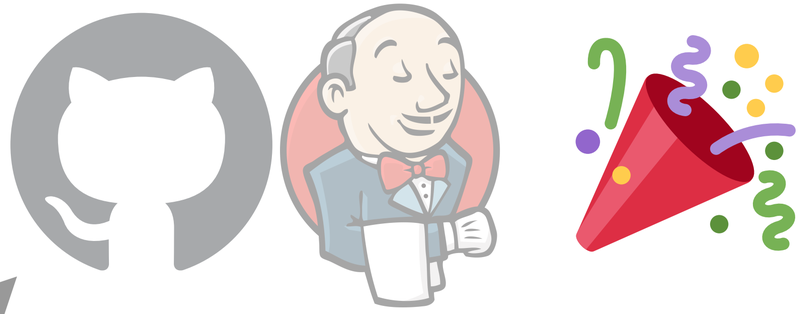
- Scipioのリモートキャッシュ機能を使って、オブジェクトストレージにXCFrameworkを保存
- 開発者はフレームワークを取得するだけでアプリケーションビルドができるように


Verda
6cc7dbe.xcframework
17aef70.xcframework
SWIFT
Package.swift
Package.swiftの変更を検知


6cc7dbe.xcframework
17aef70.xcframework
Verda
オブジェクトストレージ（S3互換）
CIがScipioでXCFramework生成
SWIFT
Package.swift


6cc7dbe.xcframework
17aef70.xcframework
Verda
開発者がビルド時にXCFrameworkを
取得（Public URL）
SWIFT
Package.swift

フレームワークのビルドが不要に



Package.swift

✅ Restore NIOConcurrencyHelpers.xcframework (278968b0490f5215b32bfcada76e7ab7cef0aaf5f2931542def588721857bb6c) from cache storage.
✅ Restore RealModule.xcframework (a2087caafd6a3f15b818f1c33e45af53cc23bad11fb232b0de2f6ec90a7a3768) from cache storage.
✅ Restore Thrift.xcframework (4e344f7ce18d029df0db7f6e6e0bf8e511b5d93d8a7db37d3bd425ef9434568d) from cache storage.
✅ Restore NIOHPACK.xcframework (c715b8d69d62e402737448796aa7ddbae08e2465de93f548dd78df33f147e095) from cache storage.
✅ Restore SDWebImage.xcframework (0289f9af440bbdb18c16ab20cce620b3f436a1b77bfa37a564c2f943947ba436) from cache storage.
✅ Restore NIOPosix.xcframework (0d51308a3722a4bcf7e07235de9a251e2c4165b6a05cd1a43abe211e736d208e) from cache storage.
✅ Restore FLEX.xcframework (ac837b2cd122fe20600902c05181fa8d7dcacb3d6b883daa4f6078cfbe036158) from cache storage.
✅ Restore GoogleMapsCore.xcframework (2edb47b46c431edf6753974a7f99f4b43f755c0d25673fe3847217aaaee202ca) from cache storage.
✅ Restore GoogleUtilities_Logger.xcframework (8804a8a359b3028be6f12839e407053af3e9ae194a8b84bf1cc453c16d9eaec1) from cache storage.
✅ Restore GoogleAppMeasurementIdentitySupport.xcframework (3befebb77e514e9514f7199e54fc72472383b840d6b5738d6ec6525756f369c3) from cache sto

移行の結果 - ビルド時間

	Prebuiltタスク	Xcodeビルド	合計
2022/11	920s	500s	1,420s
現在（キャッシュなし）	776s	590s	1,366s
現在（キャッシュあり）	220s		810s

移行の結果 - ビルド時間

	Prebuiltタスク	Xcodeビルド	合計
2022/11	920s	500s	1,420s
現在（キャッシュなし）	776s	590s	
現在（キャッシュあり）	220s		

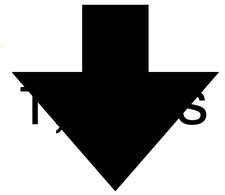
9分16秒改善🚀

x3.5 🚀

移行の結果 - ビルド時間

	Prebuiltタスク	Xcodeビルド	合計
2022/11	920s	500s	1,420s
現在（キャッシュなし）	776s	590s	[illegible]s
現在（キャッシュあり）	220s		810s

x1.75 🚀



今後の展望

- パッケージマニフェストの自動更新
  - バージョンが置いて行かれないようにする

今後の展望

- WWDC23の新機能のサポート
  - XCFrameworkのMergeable Library対応
  - XCFrameworkの署名やプライバシーマニフェスト対応

当初の目標

- とにかく標準のビルドシステムを使いたい
- ビルド成果物が再利用できてキャッシュしやすい
- 開発しやすくテストしやすい
- 依存関係のアップデートがしやすい

達成？

- ✅ とにかく標準のビルドシステムを使いたい
- ✅ ビルド成果物が再利用できてキャッシュしやすい
- 🤔 開発しやすくテストしやすい
- ✅ 依存関係のアップデートがしやすい

標準？

- Bazelを使うよりは標準的な構成に近づいたが
- 現れた新たなビルドシステム
  - 標準の仕組みを使う工夫はしたが……
  - Scipio自体の複雑さ
  - 第2、第3の負債を産んでいるだけ説もある

Swift Package移行は正解

- とはいえ依存のSwift Package化は正しいはず
- 今後標準のSwift Package統合が強化されて、そこに乗るのが一番良い
  - →将来今の構成を捨てやすくなった

標準の
ビルドシステムを
使おう！！！

まとめ

- ビルド環境移行によりBazelが全部外れる目処が立ってきた
- XCFramework化で、成果物をキャッシュしつつ開発できるようになった
- Scipioは実用段階なのでライブラリのビルドにお困りの方はご利用ください
- XcodeのSwift Package統合を使って問題がなければ普通に開発しよう
  - 普通がいちばん

Scipio

- An awesome build tool -

https://github.com/giginet/Scipio

Scipio Public

Pin | Unwatch 11 | Fork 11 | Star 299

main | 11 branches | 18 tags | Go to file | Add file | Code

giginet Merge pull request #96 from giginet/documentation ... 05e8238 4 days ago 620 commits

.github/workflows	Remove swift-docc for distribution	5 days ago
Plugins/GenerateScipioVersion	Rename plugin name	3 months ago
Sources	Update Sources/scipio/scipio.docc/prepare-cache-for-applications....	4 days ago
Tests	Merge pull request #94 from giginet/...	2 weeks ago
.gitignore	Add links to DocC	last week
.swiftlint.yml	disable lint rule	3 weeks ago
LICENSE.md	Create LICENSE	last year
Package.resolved	Use 14.3.1	last month
Package.swift	Remove swift-docc for distribution	5 days ago
README.md	Tweak badges	last week

README.md

Scipio

build passing | Swift 5.8 | SwiftPM compatible | Documentation available | Platform iOS|macOS|watchOS|tvOS|visionOS | Xcode 14.3 | Xcode 15.0 | License MIT

Carthage depends on

About

A new build tool to generate XCFramework

giginet.github.io/Scipio/documentatio...

Readme
MIT license
Activity
299 stars
11 watching
11 forks

Releases 14

0.16.0 Latest
2 weeks ago

+ 13 releases

Packages

No packages published
Publish your first package

Contributors 7

ご清聴
ありがとう
ございました