
Яндекс

Яндекс

Как модульность изменила не только наш код, но и весь процесс

Николай Лихогруд

Руководитель группы разработки Яндекс.Карт для iOS

План

- Мотивация
- Какие модули мы выделили
- Особенности использования Cocoapods
- Архитектура модульного приложения
- Что мы в итоге получили

Мотивация

Причины разбиения на модули

Вылет сборки из-за слишком большого количества файлов

- ~1900 файлов на swift в главном таргете **год назад**
- ~3200 на данный момент по всем модулям

Причины разбиения на модули

Долгая компиляция

- ~220к строк чистого кода на swift в главном таргете **год назад**
- ~3 мин после изменения одной строчки (используем SWIFT_WHOLE_MODULE_OPTIMIZATION)

~320к на данный момент по всем модулям

Цели разбиения на модули

- Преодоление проблем большого монолита
- Удобство работы с тестовыми проектами для фичей

Ценность тестовых проектов

Быстрое прототипирование

- Нет необходимости выполнять интеграцию в основное приложение, чтобы выложить первую сборку

Ценность тестовых проектов

На порядок меньше кода, чем в основном проекта

- Быстрый билд
- Не тормозит Xcode

Ценность тестовых проектов

Упрощается независимая разработка фичей

- Разработчики пересекаются только при интеграции фичей в приложение

Тестовые проекты - наше все!

Какие модули выделили

Цели

- Независимые тестируемые модули
- Потенциал переиспользования
- Общий код в основном приложении и в тестовых
- Минимальный объем тестовых проектов

С чего начинать?

Выделение низкоуровневых модулей

- Utils/Helpers
- UI
- Фреймворк для таблиц
- Обертки библиотек (MapKit, RxSwift, Crashlytics)
- …

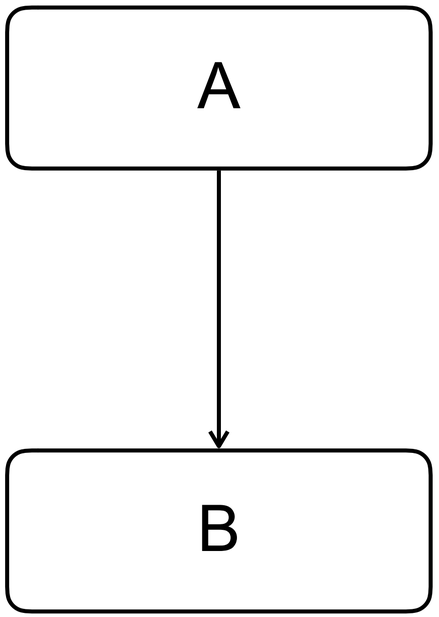
Смысл низкоуровневого модуля

Не содержат бизнес-логики и UI, завязанных на приложение

- Могут быть без изменений использованы в других приложениях

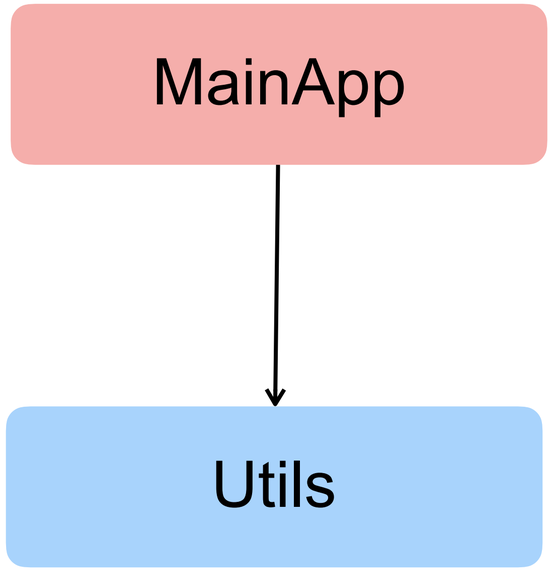
Низкоуровневые модули разрешаем напрямую использовать из модулей верхнего уровня

Условные обозначения



- Модуль **A** использует код модуля **B**
- Модуль **A** не может быть использован без модуля **B**

Схема



Как выделять не утилитные модули

По фичам

- Самодостаточность
- Любая внутренняя архитектура модуля, главное - интерфейс

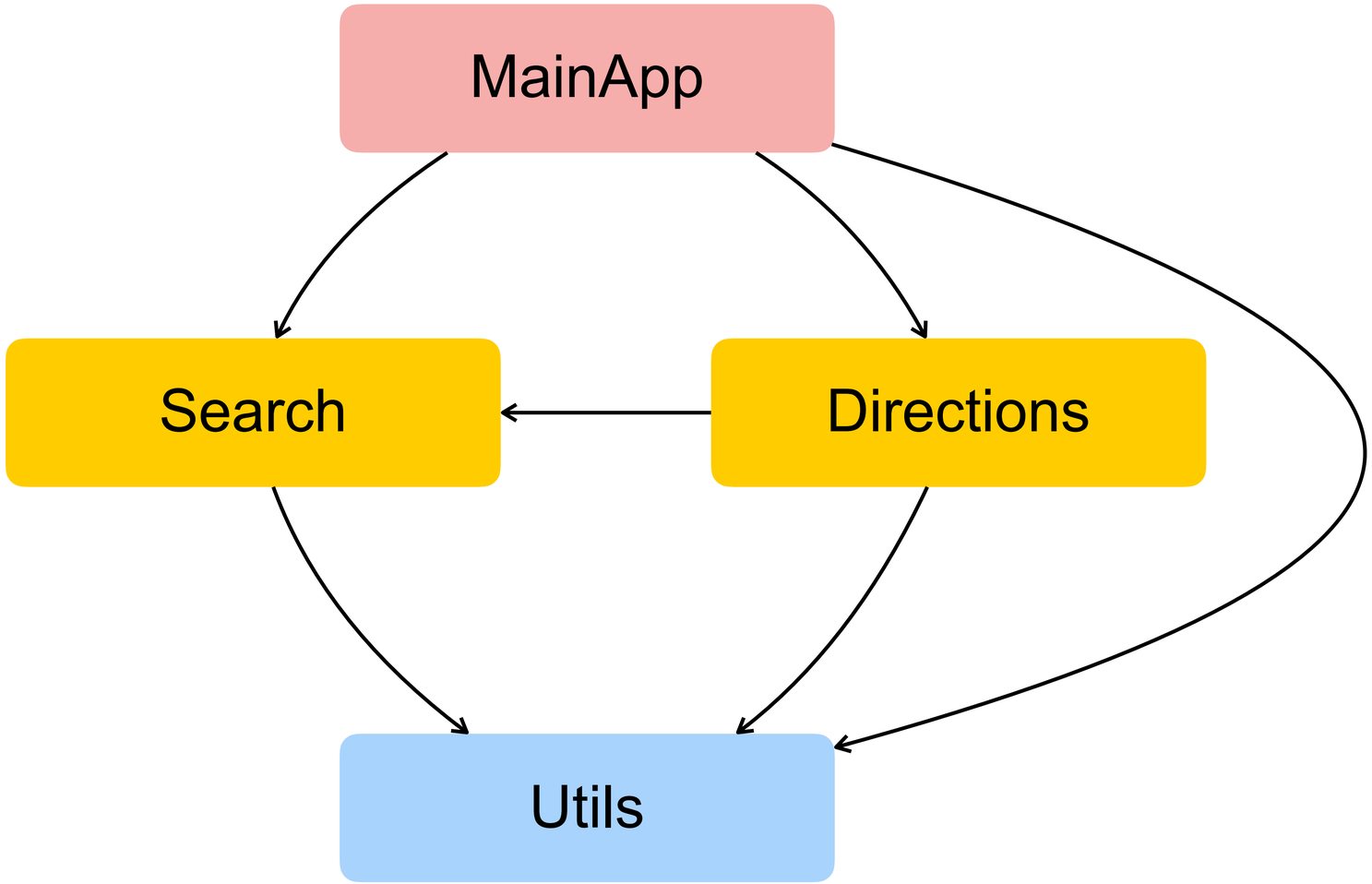
Search Directions Suggest ...

Как выделять не утилитные модули

feature-модули не зависят от главного модуля приложения

- Декларируют абстракции своих зависимостей
- Реализации лежат в приложении и инъектируются

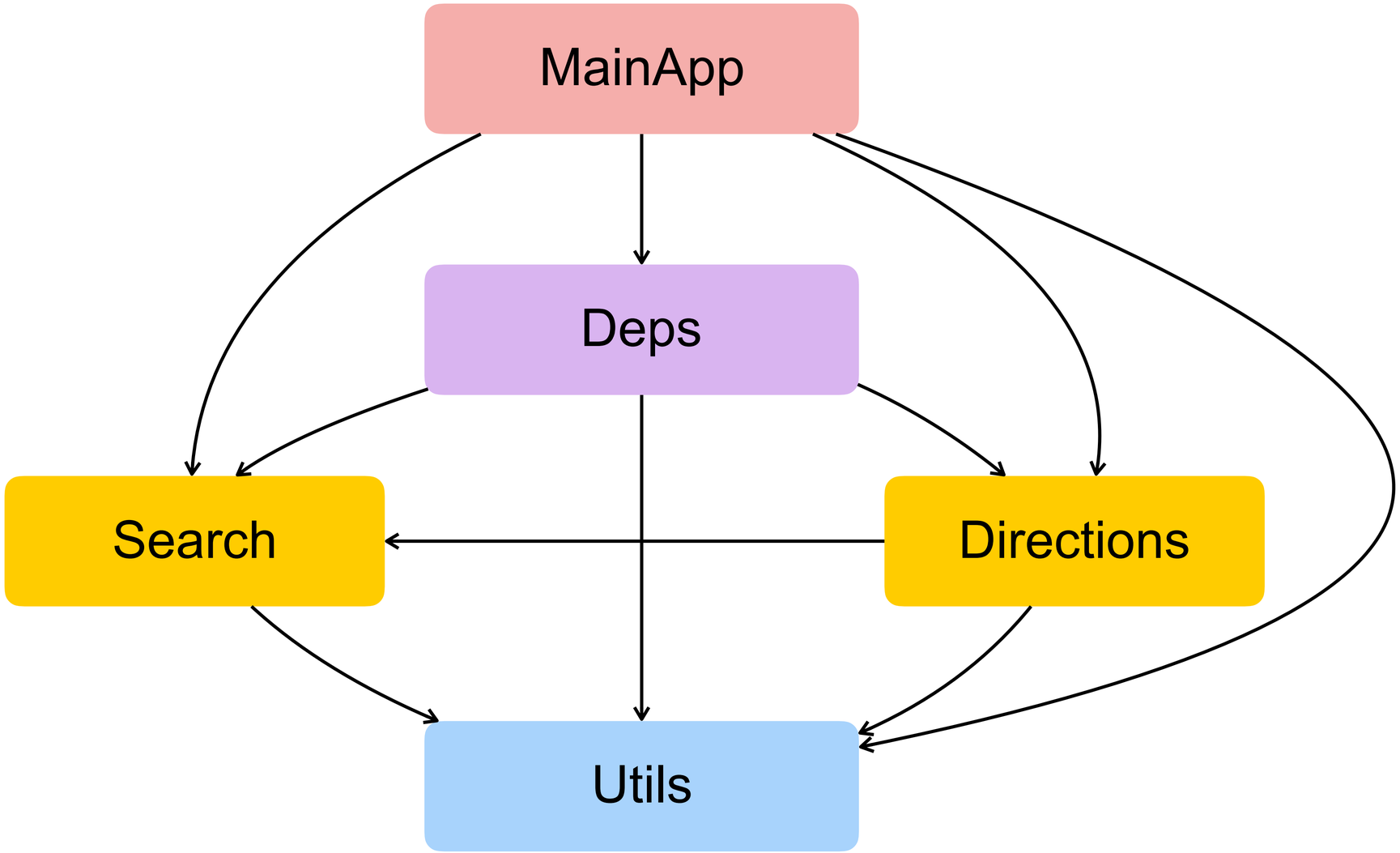
Схема



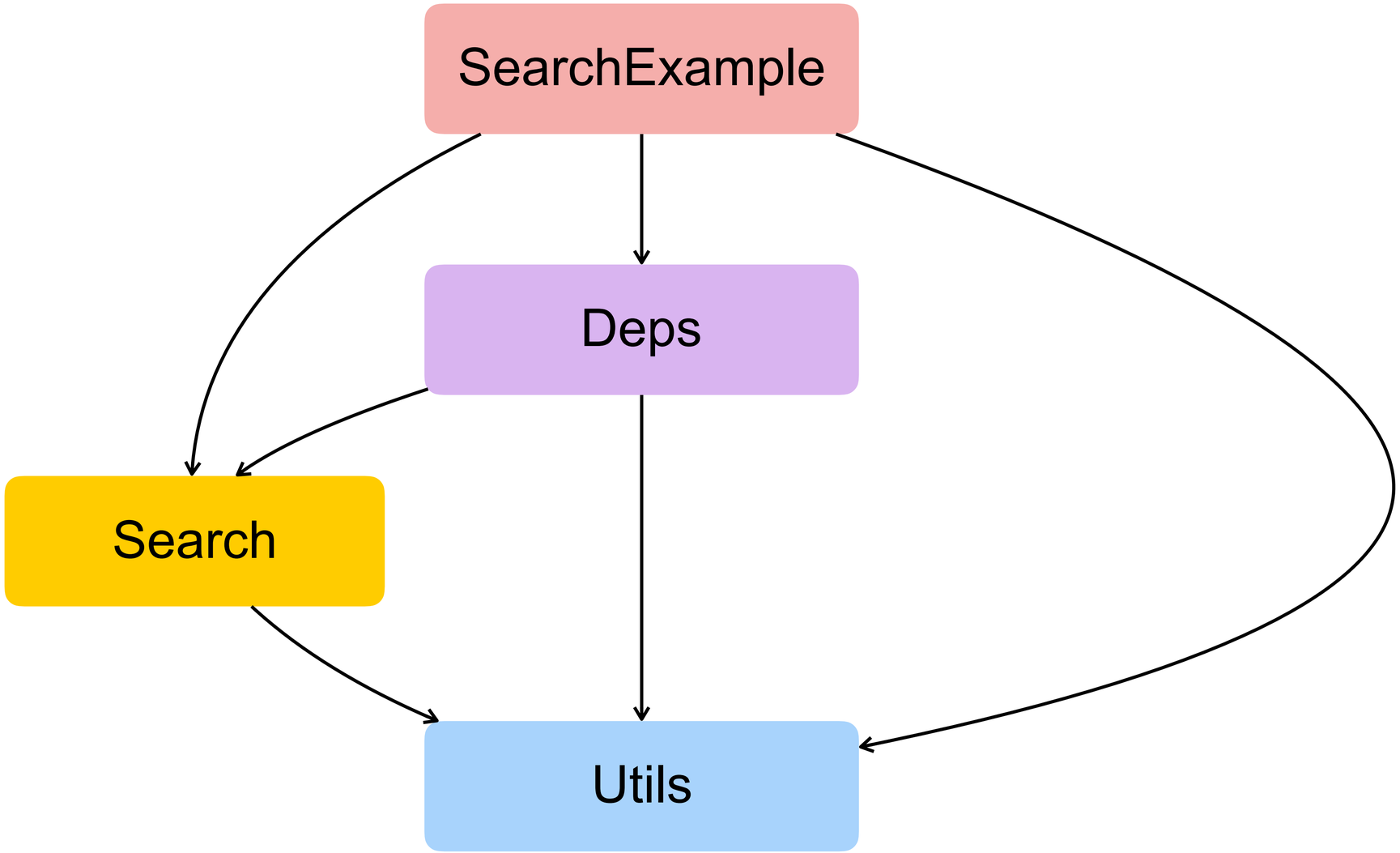
Тестовые проекты

- Для разработки фичи в отдельном проекте нужны имплементации её зависимостей
- Соберем все имплементации зависимостей feature-модулей в отдельном модуле **Deps**
- Будем использовать его в тестовых проектах

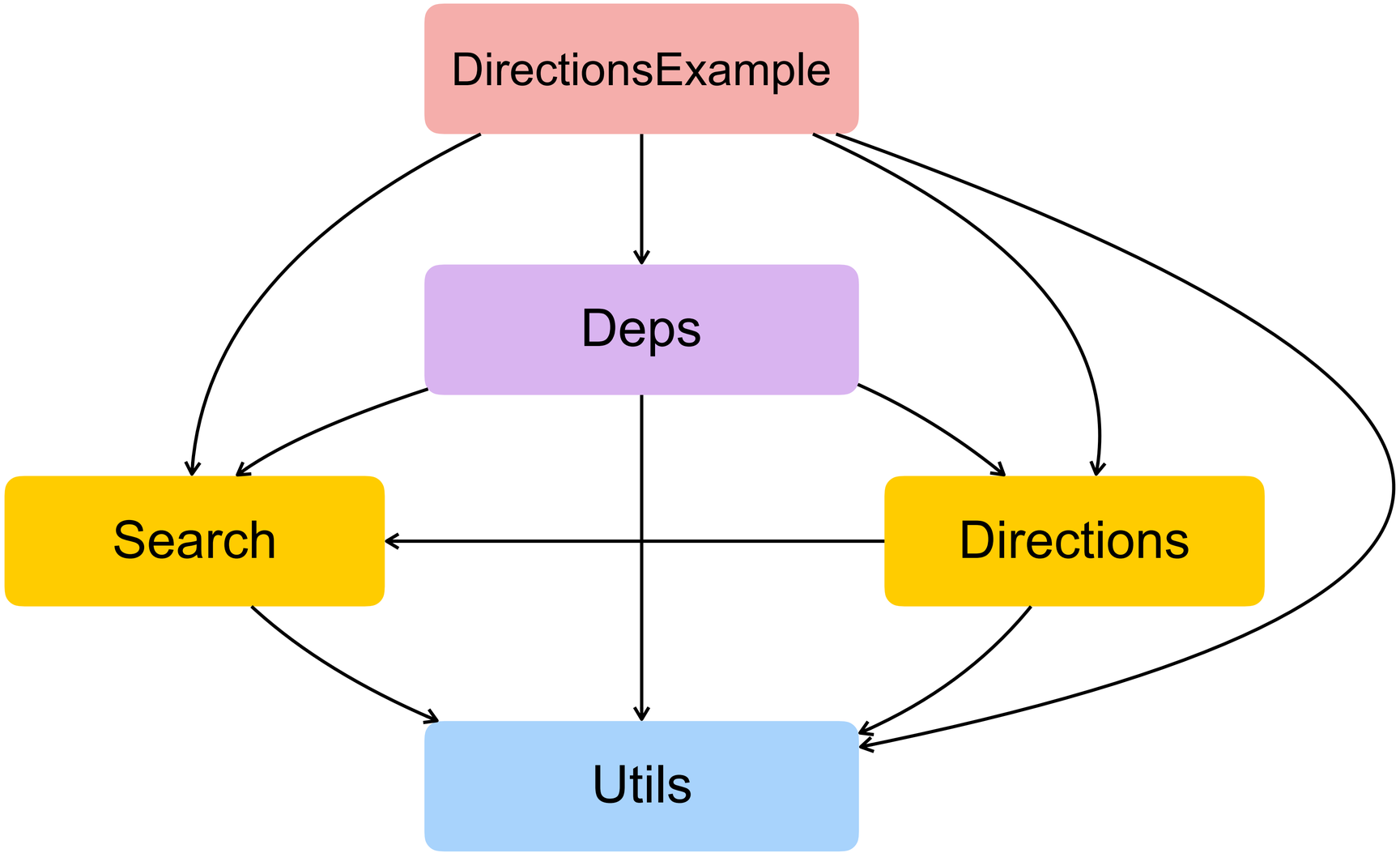
Главный проект



Проект поиска



Проект маршрутизации



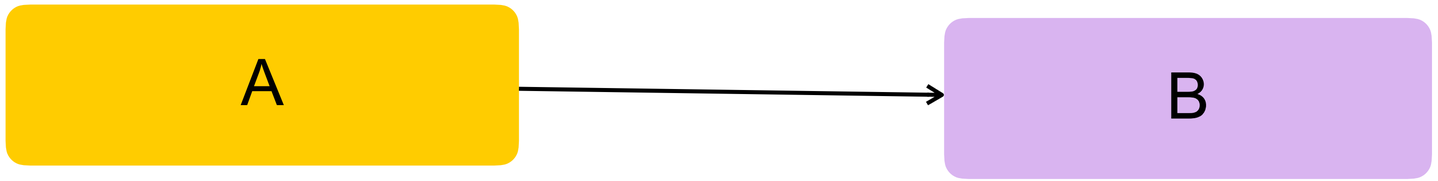
Проблемы

- Зависимости между feature-модулями
- Зависимость Deps от feature-модулей
- Дублирование абстракций общих зависимостей

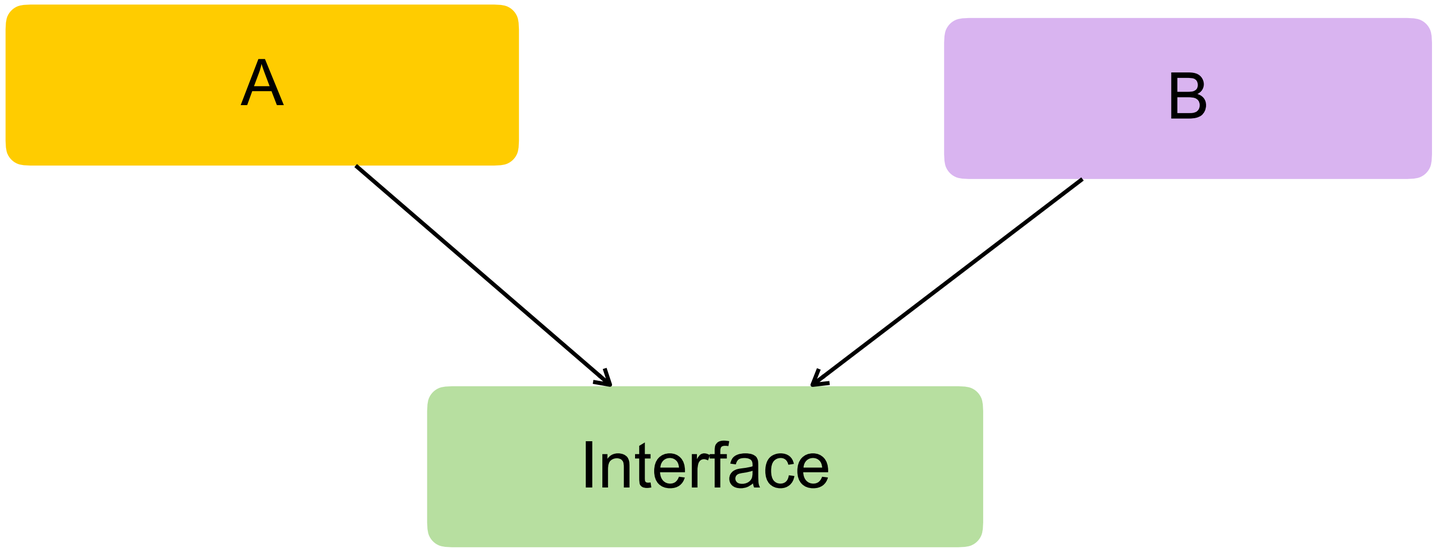
Проблемы

- Размер тестовых приложений
- Переиспользуемость модулей

Инверсия зависимостей

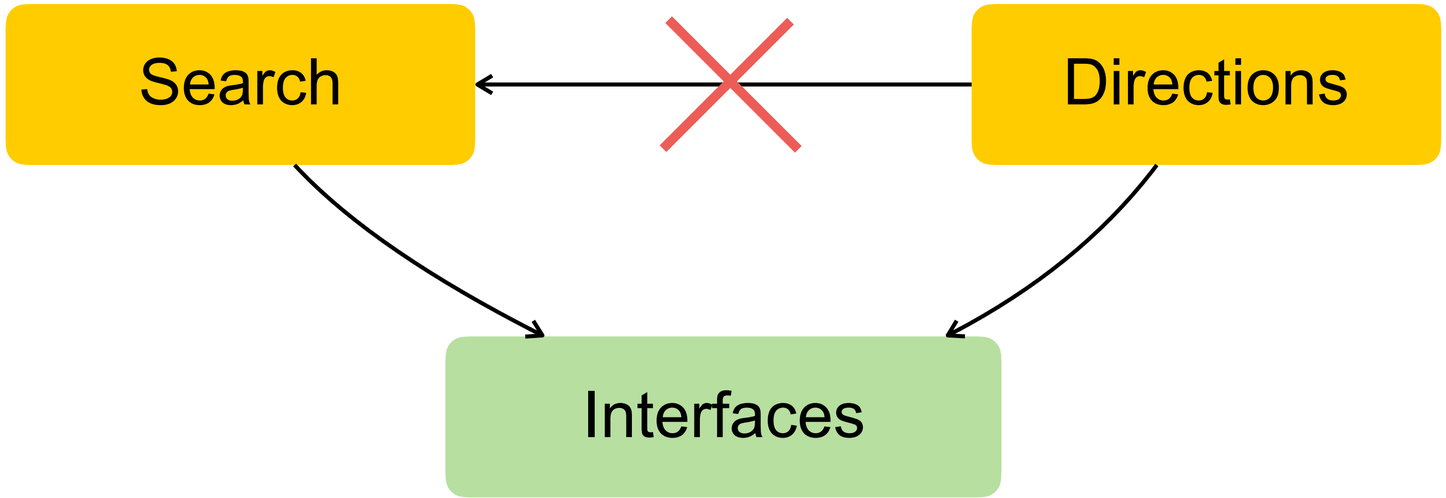


Инверсия зависимостей



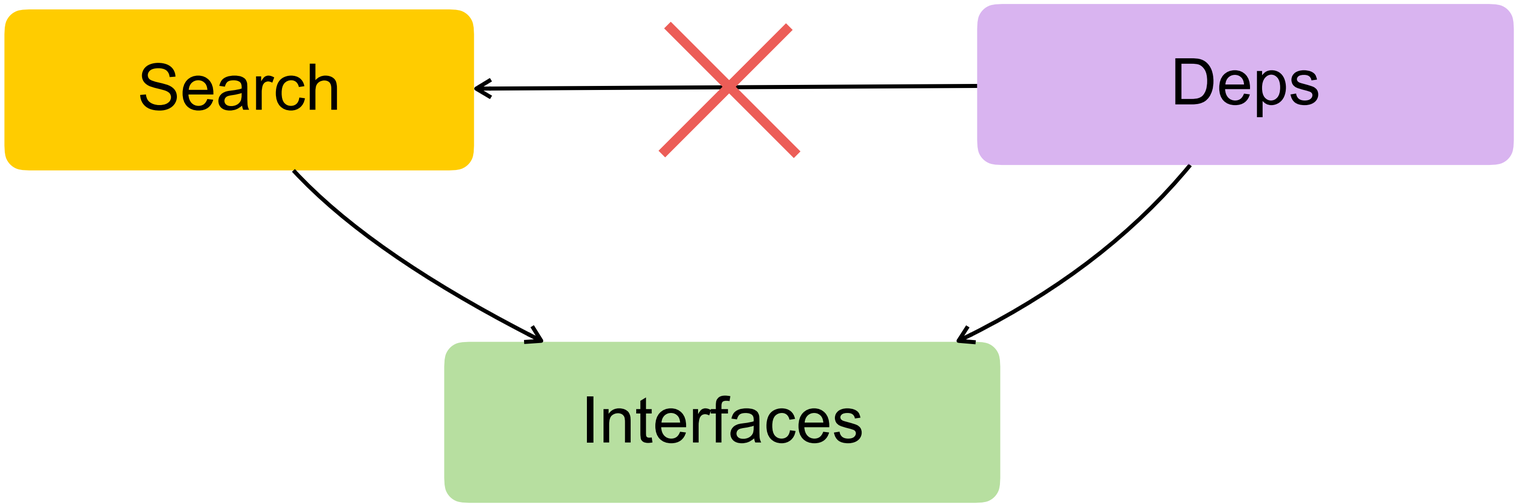
Инверсия зависимостей

Кладем в модуль **Interfaces** внешние интерфейсы feature-модулей

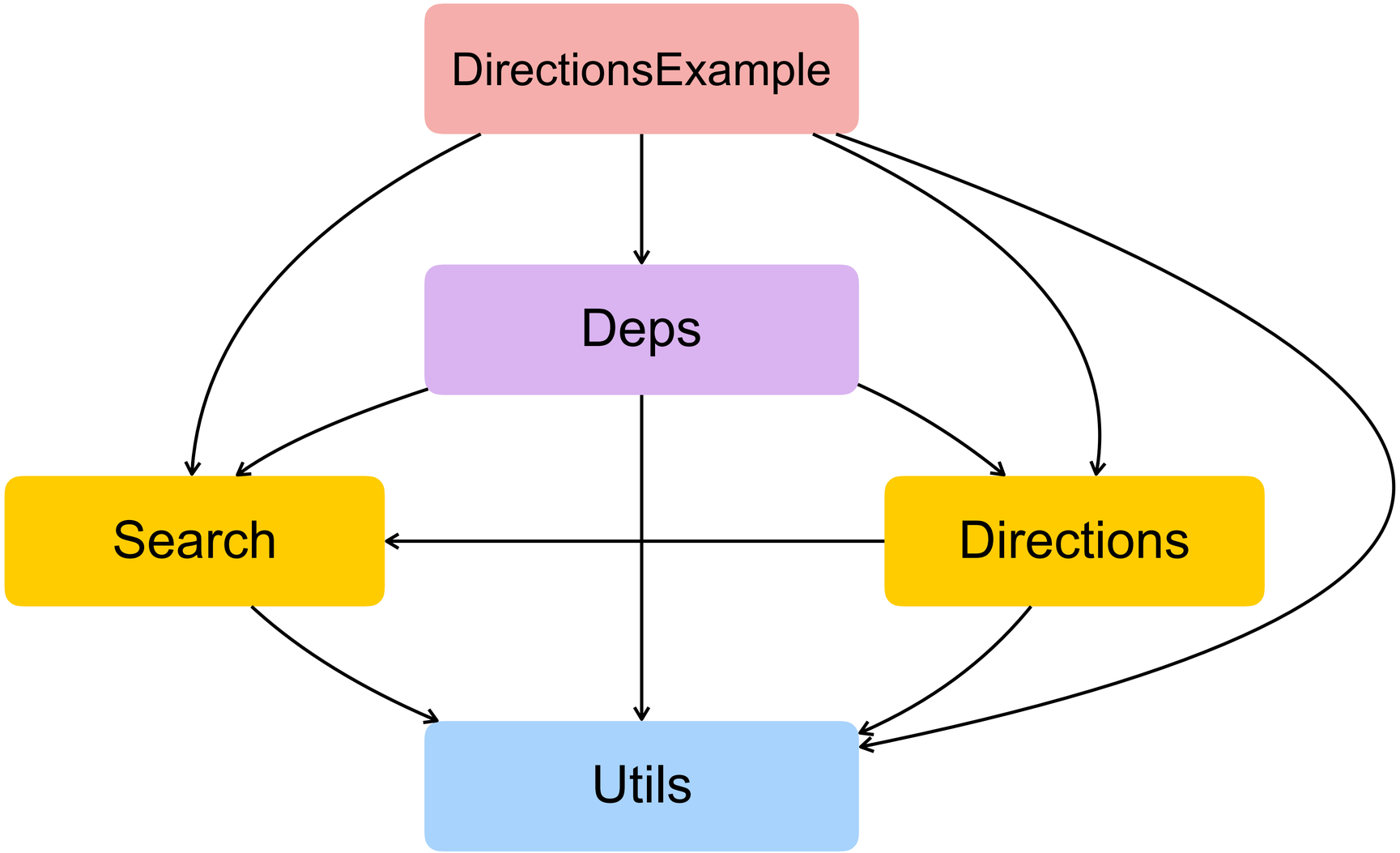


Инверсия зависимостей

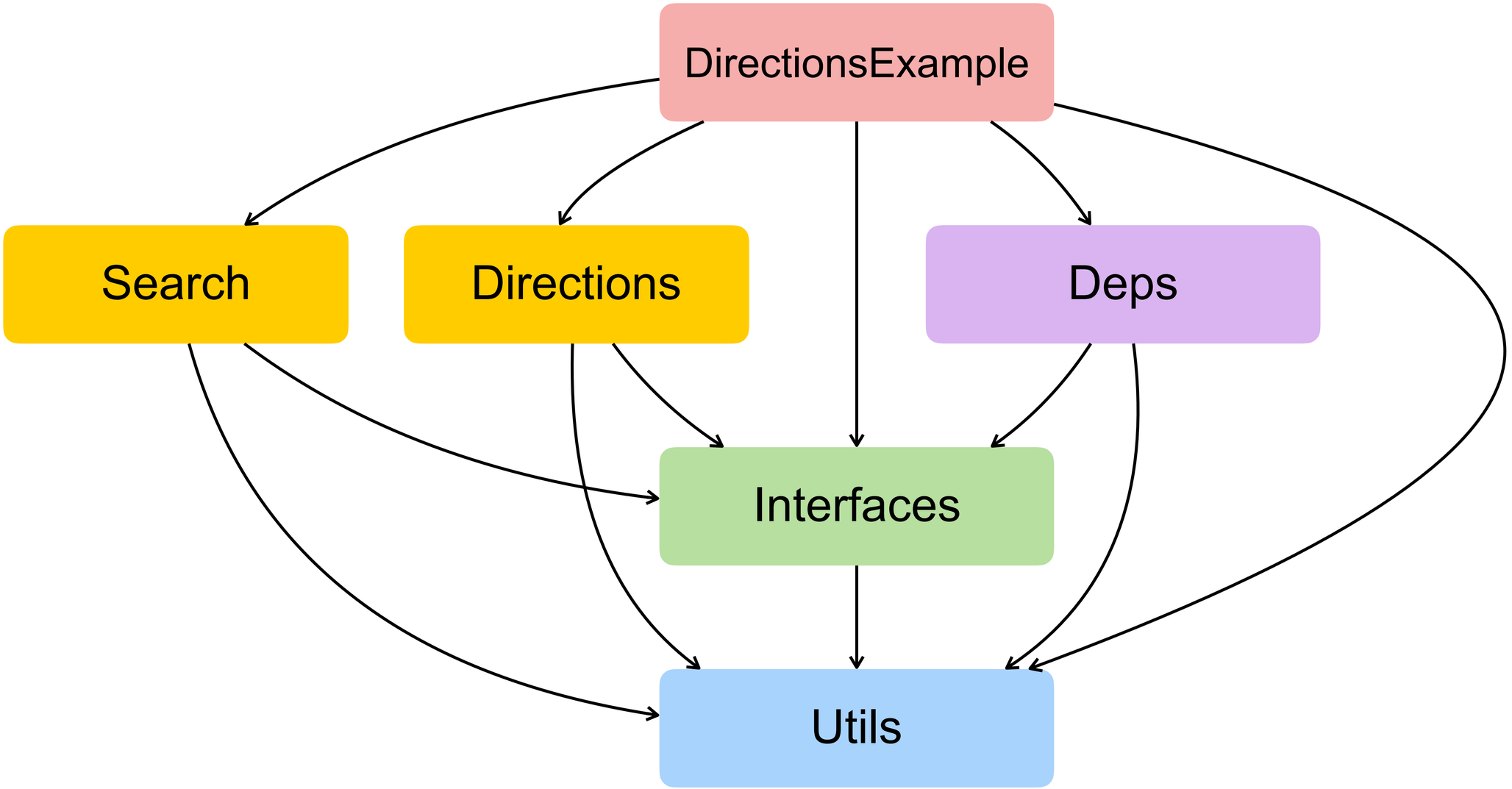
Кладем в модуль **Interfaces** абстракции зависимостей feature-модулей



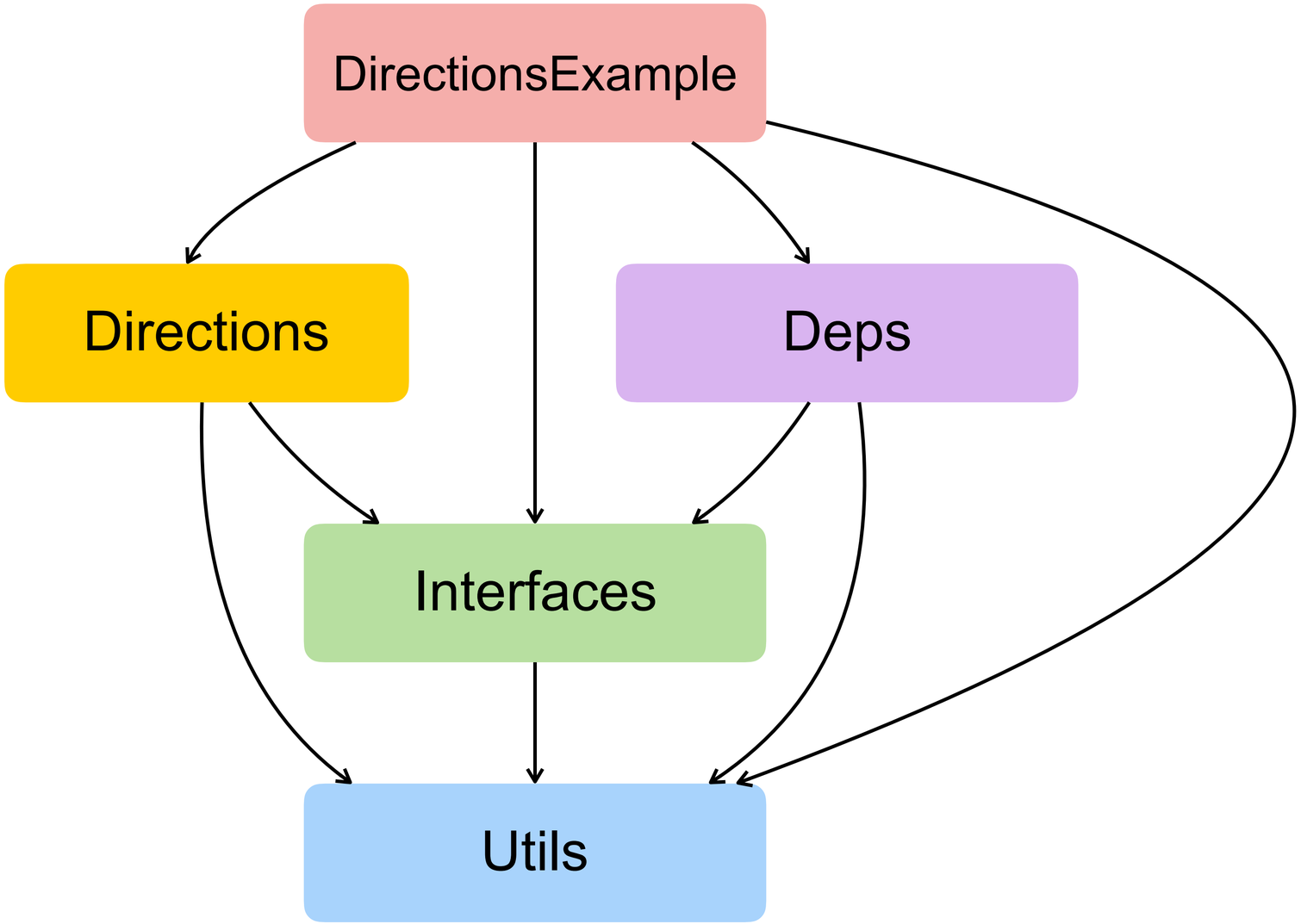
Было



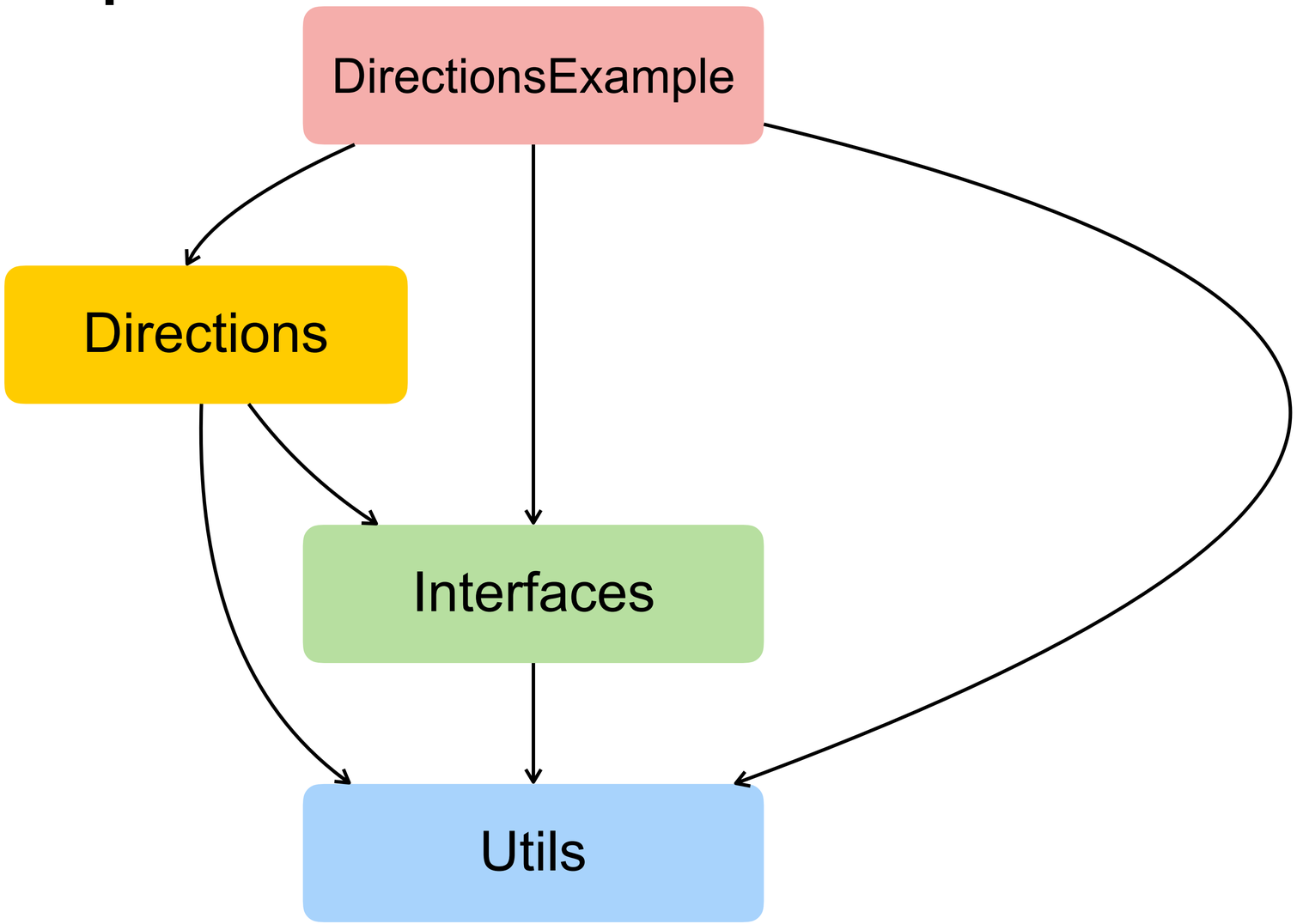
Стало


DirectionsExample
Search
Directions
Deps
Interfaces
Utils

Замокали Search



Замокали Deps



Чеклист по типам модулей

App	Создание сущностей, потоки данных, роутинг
Feature	Бизнес-логика и UI фичей
Deps	Имплементации зависимостей фичей
Interfaces	Интерфейсы фичей и их абстракции зависимостей
Utils	Код, не завязанный на приложение

Гранулирование модуля зависимостей

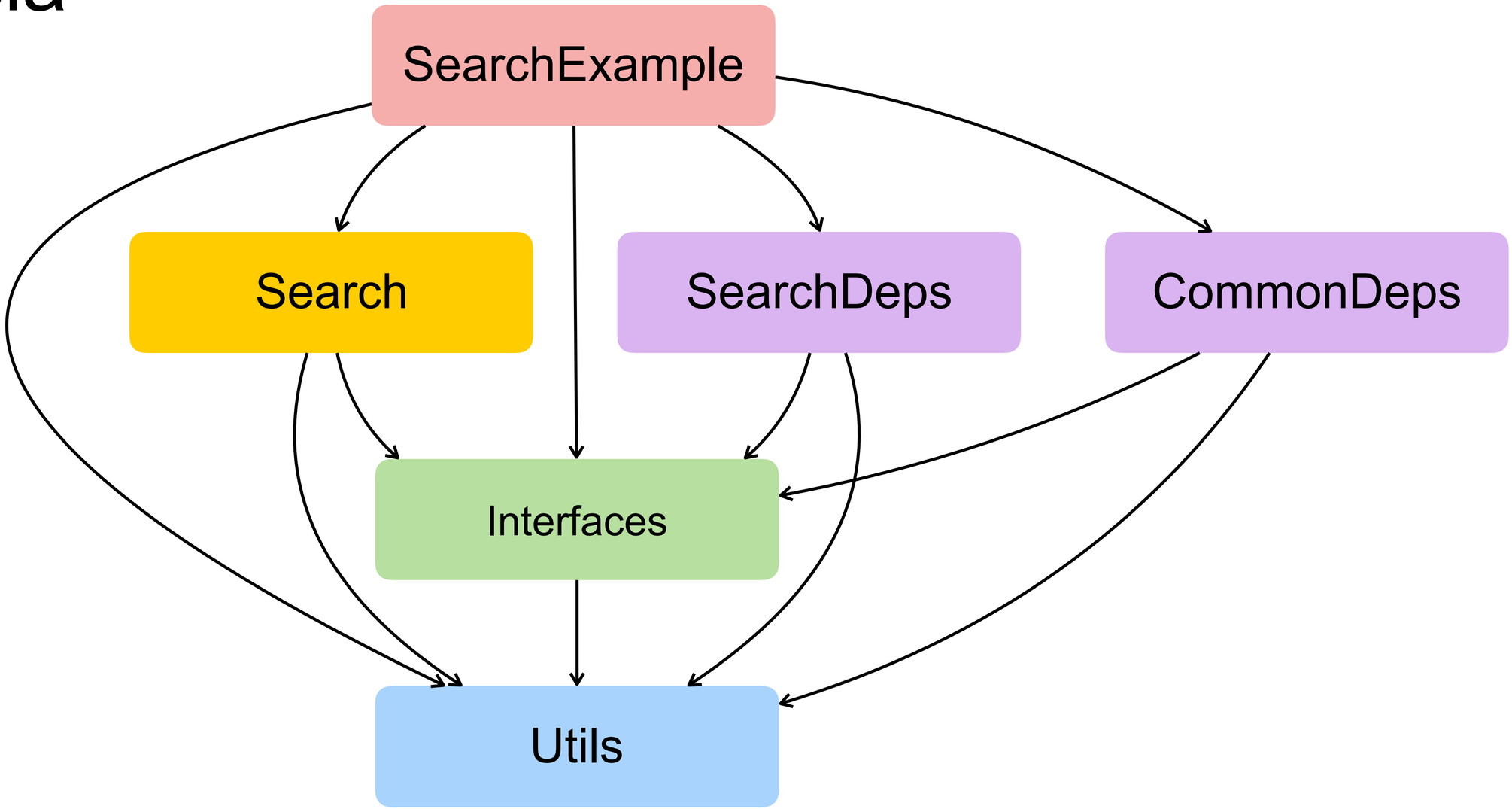
Deps в итоге может разрастись

Выделим модули с приватными зависимостями фичей

- SearchDeps
- DirectionsDeps

Все общее кладем в **CommonDeps**

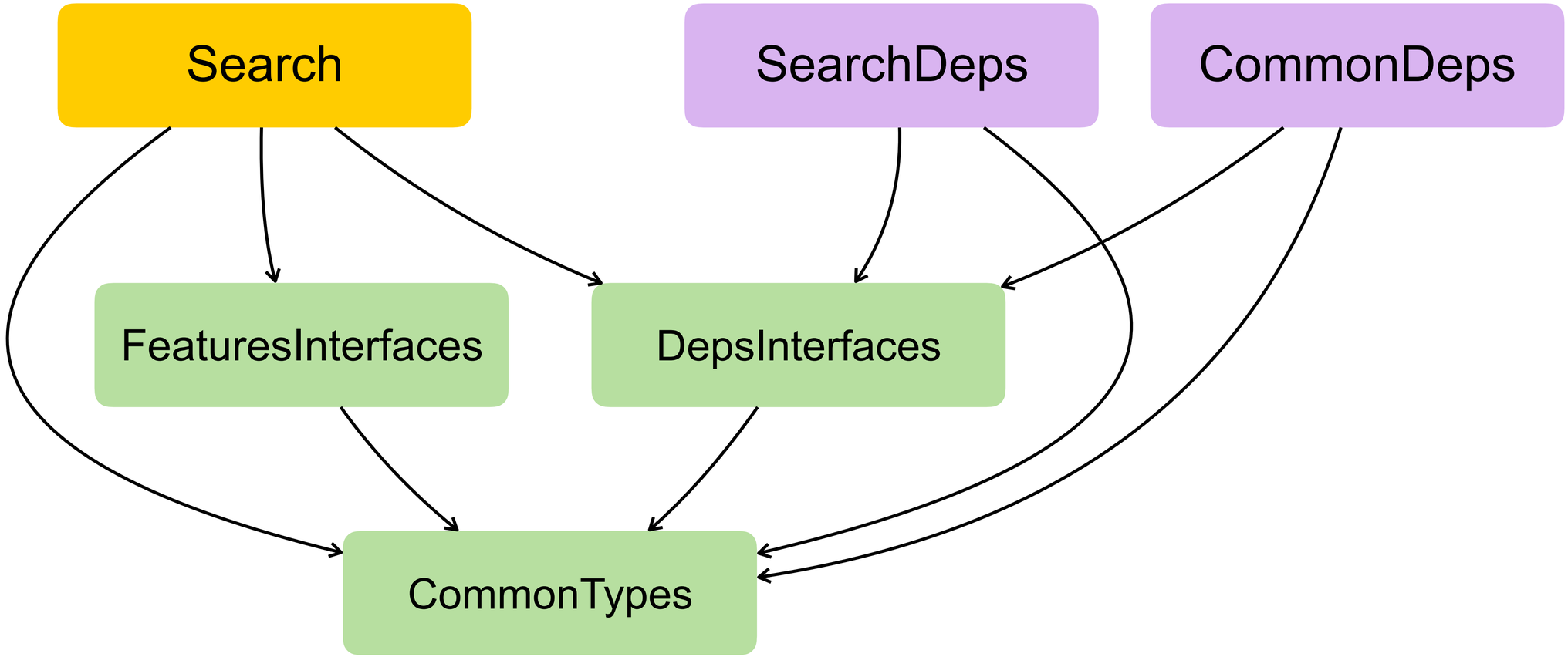
Схема


SearchExample
Search
SearchDeps
CommonDeps
Interfaces
Utils

Гранулирование модуля интерфейсов

- Внешние интерфейсы фичей - **FeaturesInterfaces**
- Зависимости фичей - **DepsInterfaces**
- Общие структуры и enum-ы - **CommonTypes**

Схема


Search
SearchDeps
CommonDeps
FeaturesInterfaces
DepsInterfaces
CommonTypes

В результате

За счет выноса зависимостей в **Deps** легко создавать тестовые проекты для фичей

- Не нужно мокать или перетаскивать файлики имплементаций из главного проекта

В результате

Инверсия зависимостей

- Нет прямых зависимостей между модулями
- Есть возможность замокать зависимости модуля
- Есть возможность замокать целый модуль

В результате

Инверсия зависимостей

- Гарантия тестируемости
- Потенциал для переиспользования

В результате

За счет грануляции **Deps** & **Interfaces** в тестовые проекты подключается минимум кода

Особенности использования Cocoapods

Разработка собственных подов

- Development Pods
- Ресурсы и тесты подов
- Сабспеки для упрощения графа модулей

Foo.podspec

```
Pod::Spec.new do |s|
    s.name     = ‘Foo'
    s.version  = '0.0.1'
    s.source   = { :git => '', :tag => s.version }
    s.author   = { 'yandex' => 'yandex.ru' }
    s.homepage = s.name
    s.summary  = s.name

    s.source_files = 'Sources/**/*.swift'

end
```

Development pods

- Кладем локально код и спеку
- Добавляем в Podfile проекта путь

```
pod 'Foo', :path => 'lib/Foo'
```

- Делаем `$bundle install` как обычно

Development pods

Обычные таргеты с точки зрения Xcode

- Меняем код
- Добавляем / удаляем файлы прямо в таргетах

Development pods

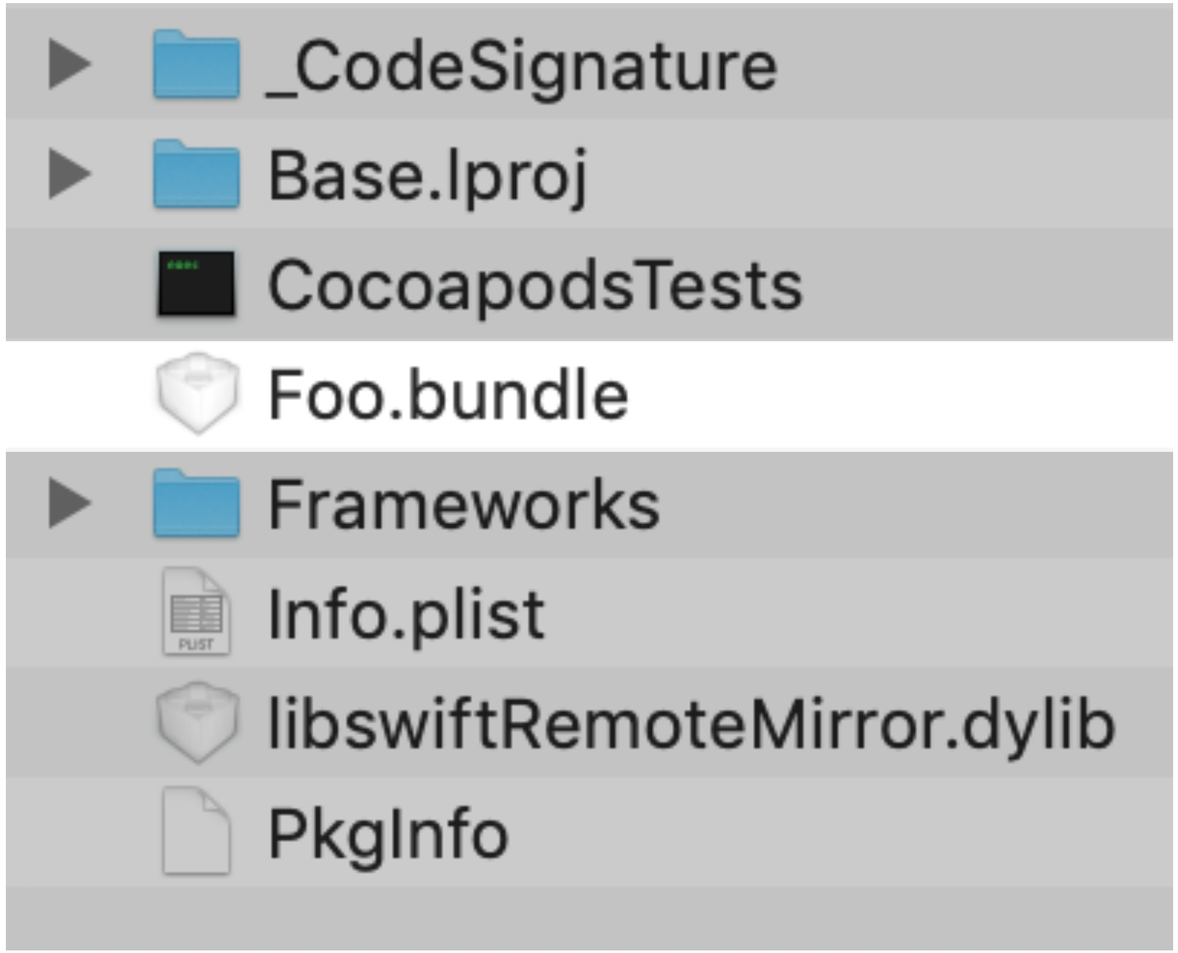
Остаемся в монорепе

- Не нужно держать отдельные репозитории
- Принудительное решение проблемы совместимости

Работа с ресурсами

```
Pod::Spec.new do |s|
    s.name = 'Foo'
    ...
    s.resource_bundles = {
        s.name => 'Resources/*.{xcassets,png}'
    }
end
```

Бандл копируется в бандл приложения



Тесты

Начиная с v1.3 поды можно подключать вместе с тестами

- Разрабатываем тесты в тестовых проектах
- Собираем все тесты в главном проекте и прогоняем на CI

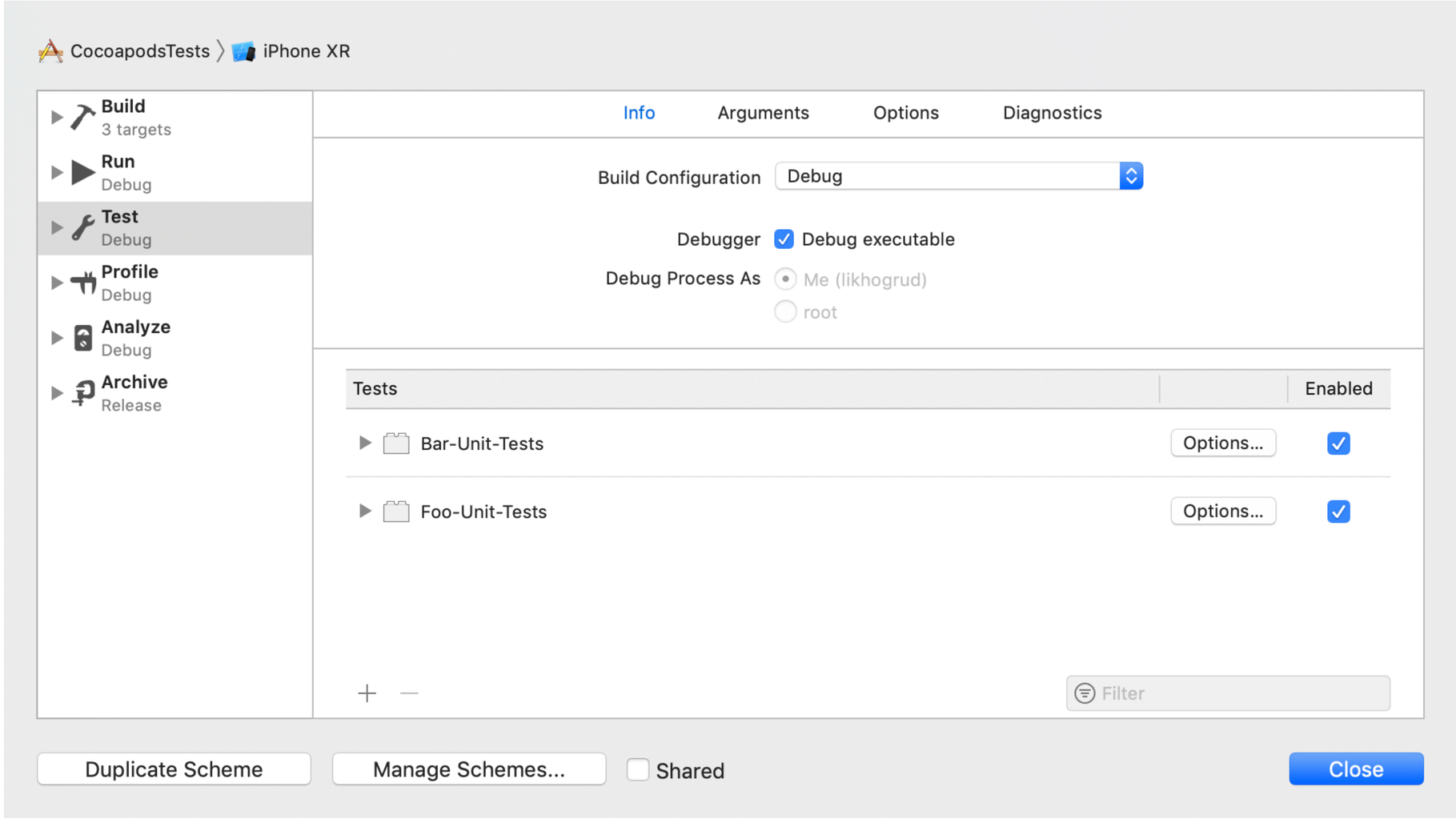
Спецификация тестов

```
Pod::Spec.new do |s|
    s.name = 'Foo'
    ...
    s.test_spec 'Tests' do |spec|
        spec.source_files = 'Tests/**/*.swift'
    end
end
```

Подключение тестов

```
target :CocoapodsTests do
    project 'CocoapodsTests'
    pod 'Alamofire'
    pod 'Foo', :path => 'lib/Foo', :testspecs => ['Tests']
    pod 'Bar', :path => 'lib/Bar', :testspecs => ['Tests']
end
```

Бандлы тестов



Snapshot-тесты

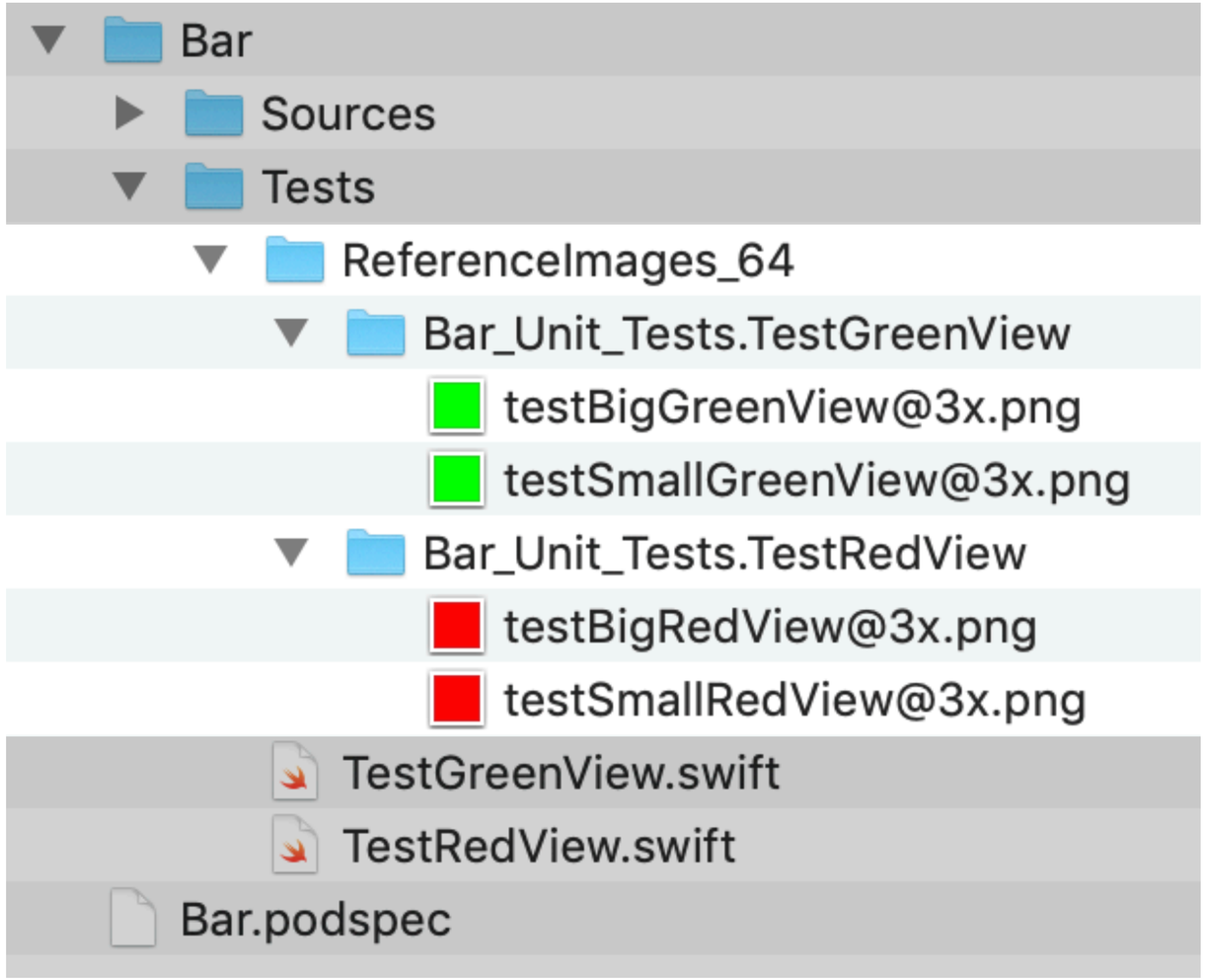
```
import Bar
import FBSnapshotTestCase

class TestRedView: FBSnapshotTestCase {

    func testSmallRedView() {
        let view = View()
        view.frame = CGRect(x: 0, y: 0, width: 100, height: 100)
        view.backgroundColor = .red

        FBSnapshotVerifyView(view)
    }
}
```

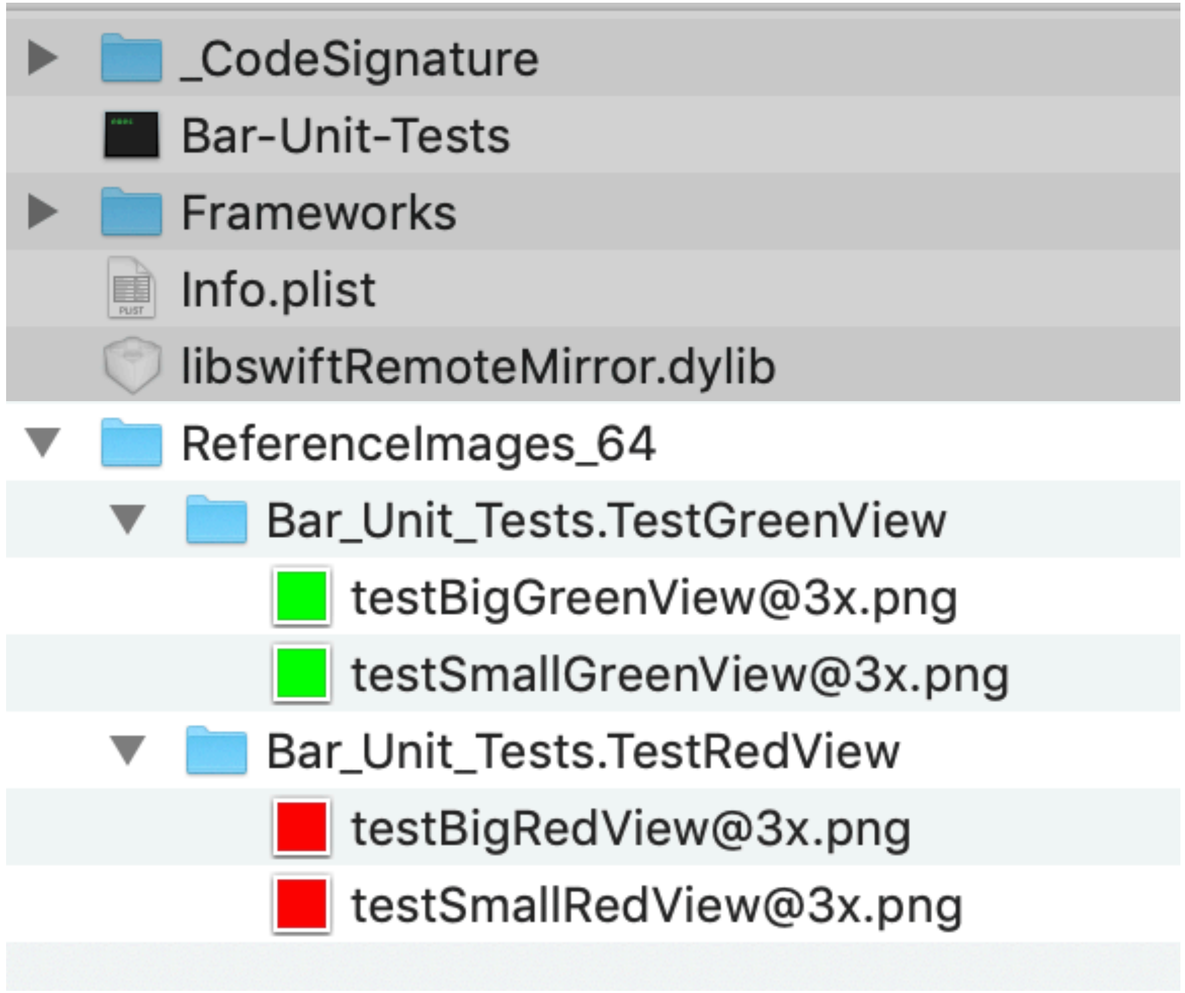
Положить Reference в папку пода



Задать пути до Reference

```
Pod::Spec.new do |s|
    s.name = 'Bar'
    ...
    s.test_spec 'Tests' do |test_spec|
        test_spec.source_files = 'Tests/**/*.swift'
        test_spec.dependency 'FBSnapshotTestCase'
        test_spec.resources = ['Tests/ReferenceImages_64']
    end
end
```

Reference копируются в бандл тестов



Сабспеки

Спецификации пода под различные ситуации использования

- Могут иметь разный код, vendored_frameworks, зависимости, собственные тесты
- Наследуют зависимости рутовой спеки

Сабспеки

- Используем для объединения всех *Deps в один под
- Используем для объединения всех *Interfaces в один под
- Сохраняем возможность раздельного подключения

Interfaces.podspec

```
Pod::Spec.new do |s|
  s.name = 'Interfaces'
  s.dependency 'CommonTypes'
  ...
  s.subspec 'Features' do |sp|
    sp.source_files = 'Sources/Features/**/*.swift'
  end

  s.subspec 'Deps' do |sp|
    sp.source_files = 'Sources/Deps/**/*.swift'
  end
end
```

Deps.podspec

```
Pod::Spec.new do |s|
  s.name = 'Deps'
  s.dependency 'CommonTypes'
  s.dependency 'Interfaces/Deps'
  ...
  s.subspec 'Common' do |sp|
    sp.source_files = 'Sources/Common/**/*.swift'
  end

  s.subspec 'Search' do |sp|
    sp.source_files = 'Sources/Search/**/*.swift'
  end
end
```

SearchExample

Podfile

```
target :SearchExample do
  project 'SearchExample'
  pod 'Search', :path => 'lib/Search'
  pod 'Deps/Common', :path => 'lib/Deps'
  pod 'Deps/Search', :path => 'lib/Deps'
  pod 'Interfaces', :path => 'lib/Interfaces'
  pod 'CommonTypes', :path => 'lib/CommonTypes'
  pod 'Utils', :path => 'lib/Utils'
end
```

Статические библиотеки

v1.4 - используем cocoapods-amimono

- https://habr.com/company/yandex/blog/335768/

v1.5 - работает из коробки для модулей, не зависящих от статических библиотек

v1.6beta - без проблем

Не вошло в доклад

Сравнение систем управления зависимостями

- Cocoapods
- Carthage
- SwiftPackageManager

Будет на https://medium.com/yandex-maps-ios

Архитектура модульного приложения

Задачи

- Межмодульные побочные эффекты
  - Обмен данными между модулями
  - Обновление состояния другого модуля
- Переходы между экранами (роутинг)

Пример: Ведение по маршруту

Ведение по маршруту постоянно взаимодействует с картой

- Отрисовка полилинии маршрута
- Автозум
- Автовращение
- События на маршруте

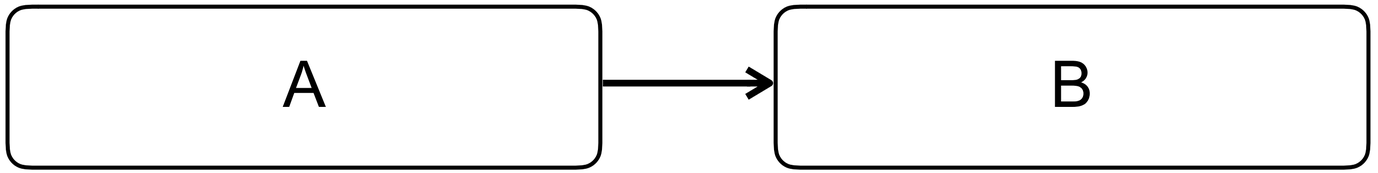
Пример: Ведение по маршруту

Ведение без логики работы с картой бессмысленно

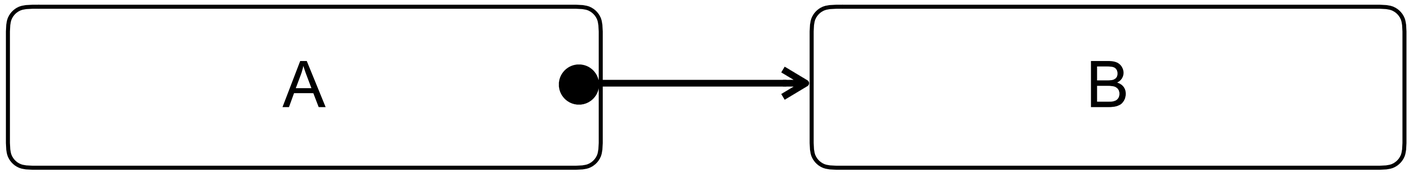
- Решение - прямая инъекция абстракции карты в модуль ведения по маршруту

Условные обозначения

A использует B



A **создает** и использует B



Условные обозначения

A использует B через интерфейс

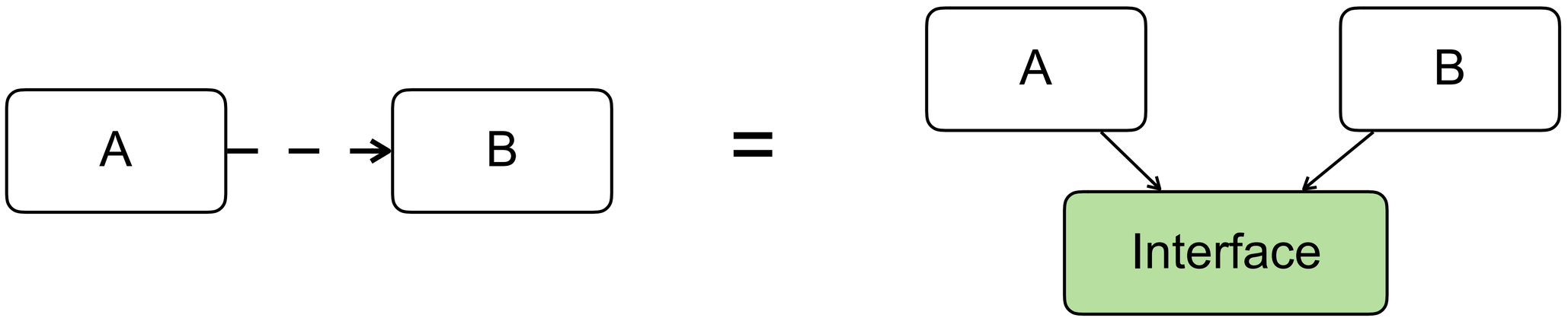
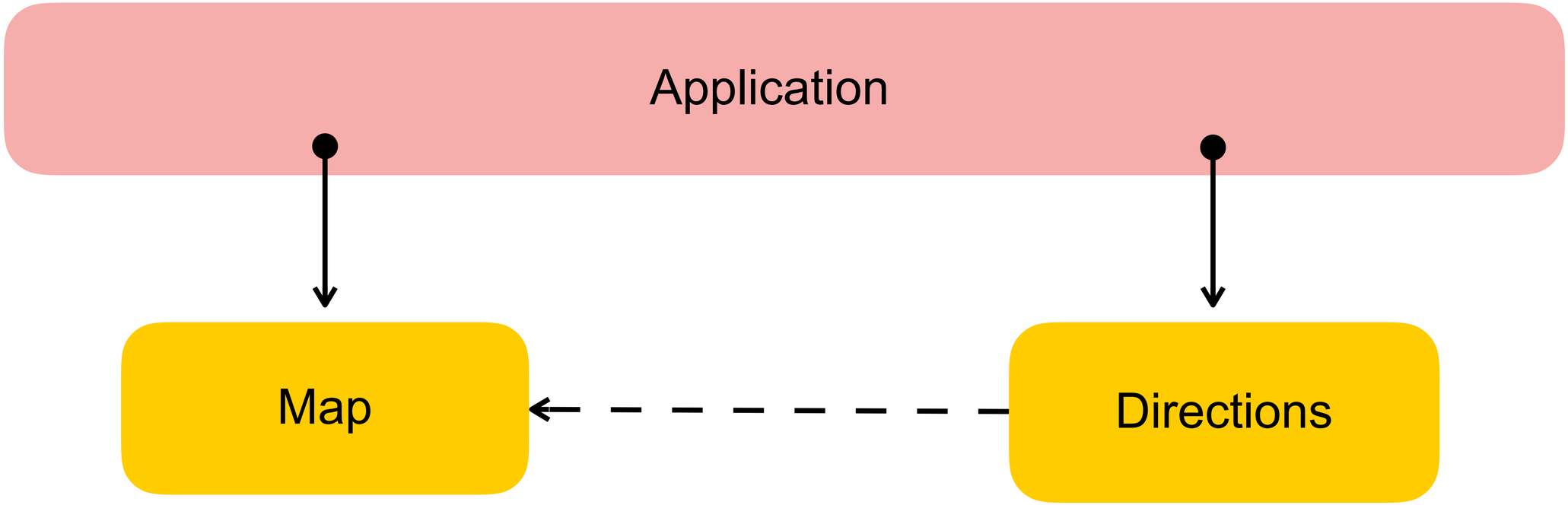
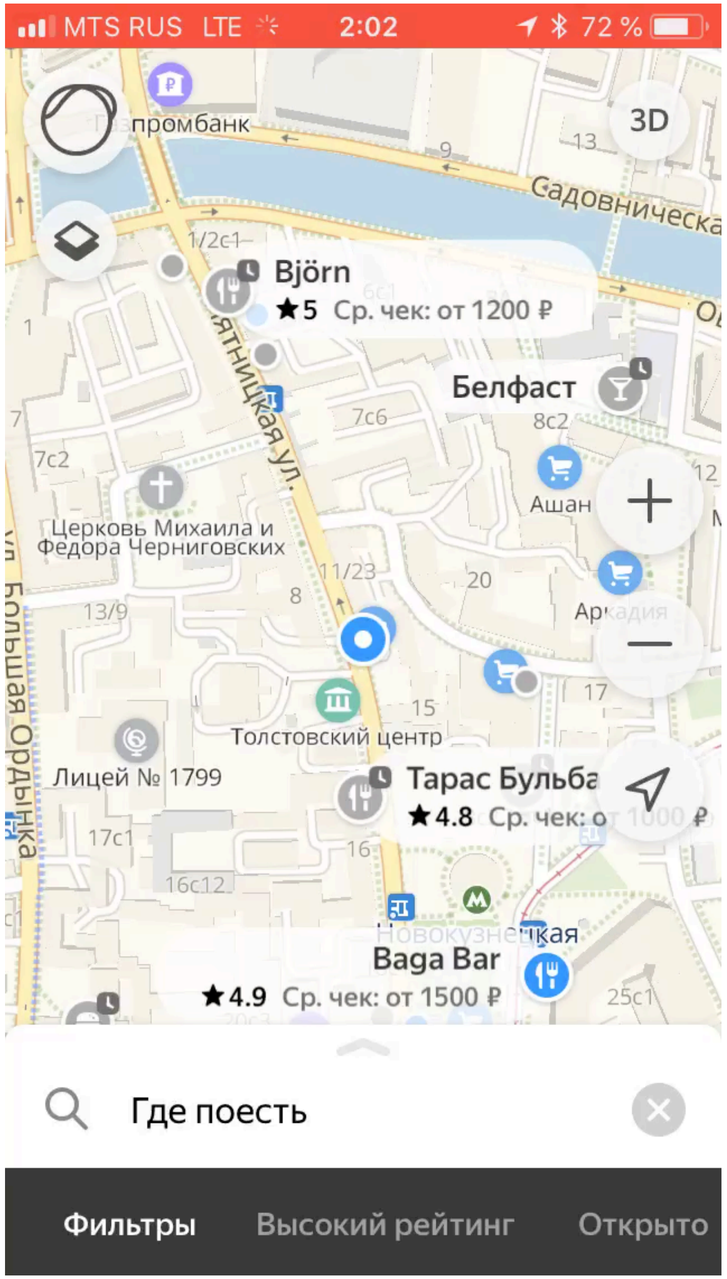


Схема с инъектированием



Пример: Экран организации

- При открытии экрана организации
  - выделить метку организации
  - подскроллить карту под точку организации
- При закрытии
  - Развыделить карточку



Решение через инъекцию

Карточка узнает про карту, метки на карте, позицию камеры и прочее

- У карточки становится больше зависимостей

На карточку возлагаются дополнительные обязанности

- У карточки становится больше логики

Решение через делегирование

Модуль карточки организации имеет смысл без карты

- Основная ответственность - показ информацией об организации

Ценность модуля не потеряется, если делегировать работу с картой наружу

Решение через делегирование

Отдадим наружу информацию, необходимую для выполнения побочных эффектов

- Координату организации
- Категорию организации

Кто должен выполнить побочные эффекты?

Binder

«Связывает» модуль с остальным приложением

Содержит весь грязный код про побочные эффекты

- «Грязный» из-за большого количества связей

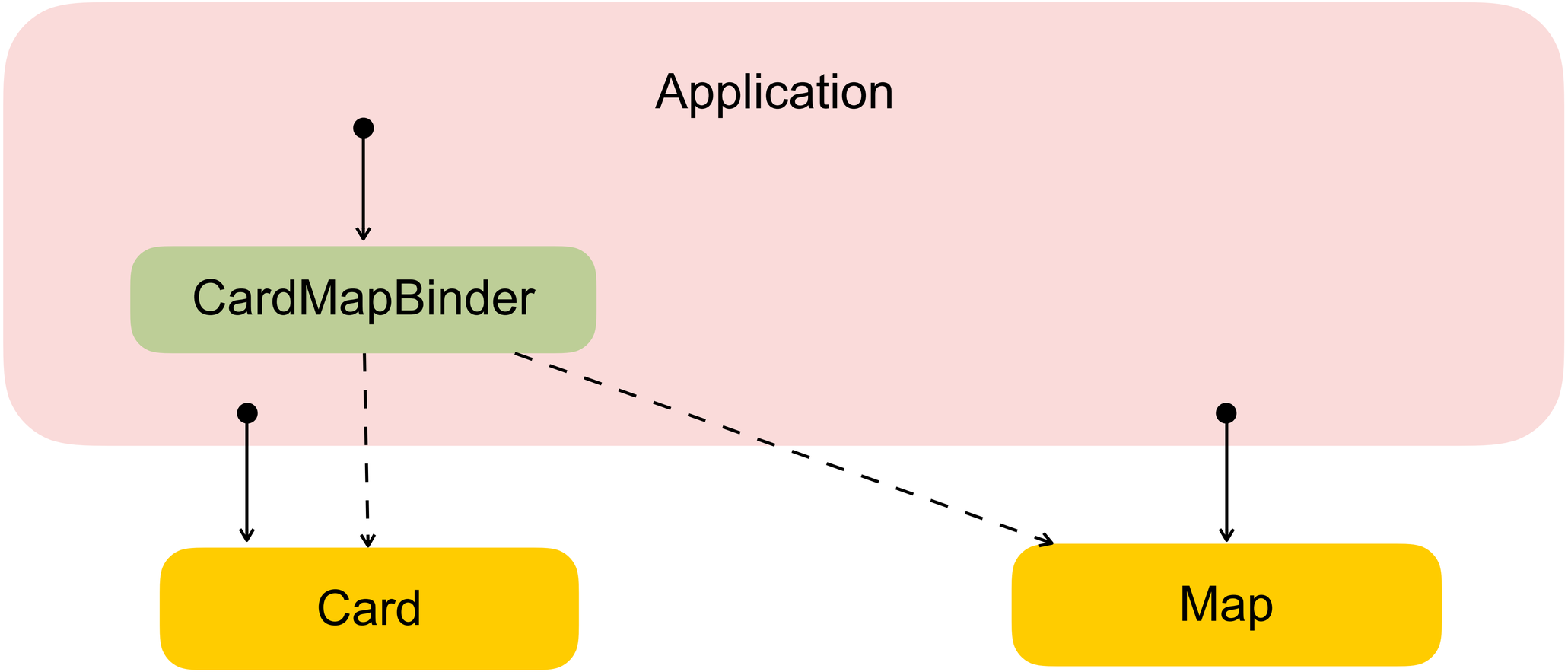
CardMapBinder

```
import Utils
import Interfaces
import Deps

class CardMapBinder {
    init(card: Card, map: Map) { }

    func activate() { }
    func deactivate() { }
}
```

Схема с Binder



CompositeBinder

Дробим Binder-ы для повышения переиспользуемости и SR

- CameraPositionBinder
- PlacemarkBinder
- MapControlsBinder
- AnalyticsEventsBinder
-

Binder

Получили новый слой приложения, инкапсулирующий второстепенные побочные эффекты жизни модуля

Кладем в **App** или в **Deps**

1. Очистили от этого кода сами модули
2. Договорились куда такой грязный код помещать и как называть

Кроссмодульные побочные эффекты

Через инъектирование

- Подходит, когда работа с сервисом/репозиторием - важная часть ответственности модуля

Через input/output модулей и Binder-ы

- Подходит для побочных эффектов, от которых хотелось бы модуль очистить

Роутинг

Процесс показа модуля

- Создать зависимости
- Создать сущности модуля
- Создать и настроить Binder-ы
- Отобразить с нужной анимацией в нужном месте иерархии вьюх/вьюконтроллеров
- …

Роутинг

Выполнять все эти действия прямо в модуле - не вариант

- Жесткая зависимость между модулями
- Дублирование кода при показе одного модуля из разных мест

Router

- Выполняет все действия, связанные с показом модуля
- Содержит весь «грязный» код по показу модуля из любого состояния приложения

Router

Выполняет все действия, связанные с показом модуля

```
protocol OrganizationCardsRouter {
    func routeToOrganizationCard(id: String)
}

protocol DirectionsRouter {
    func routeToDirections(wayPoint: WayPoint)
}
```

Router

Используется где нужно открыть модуль

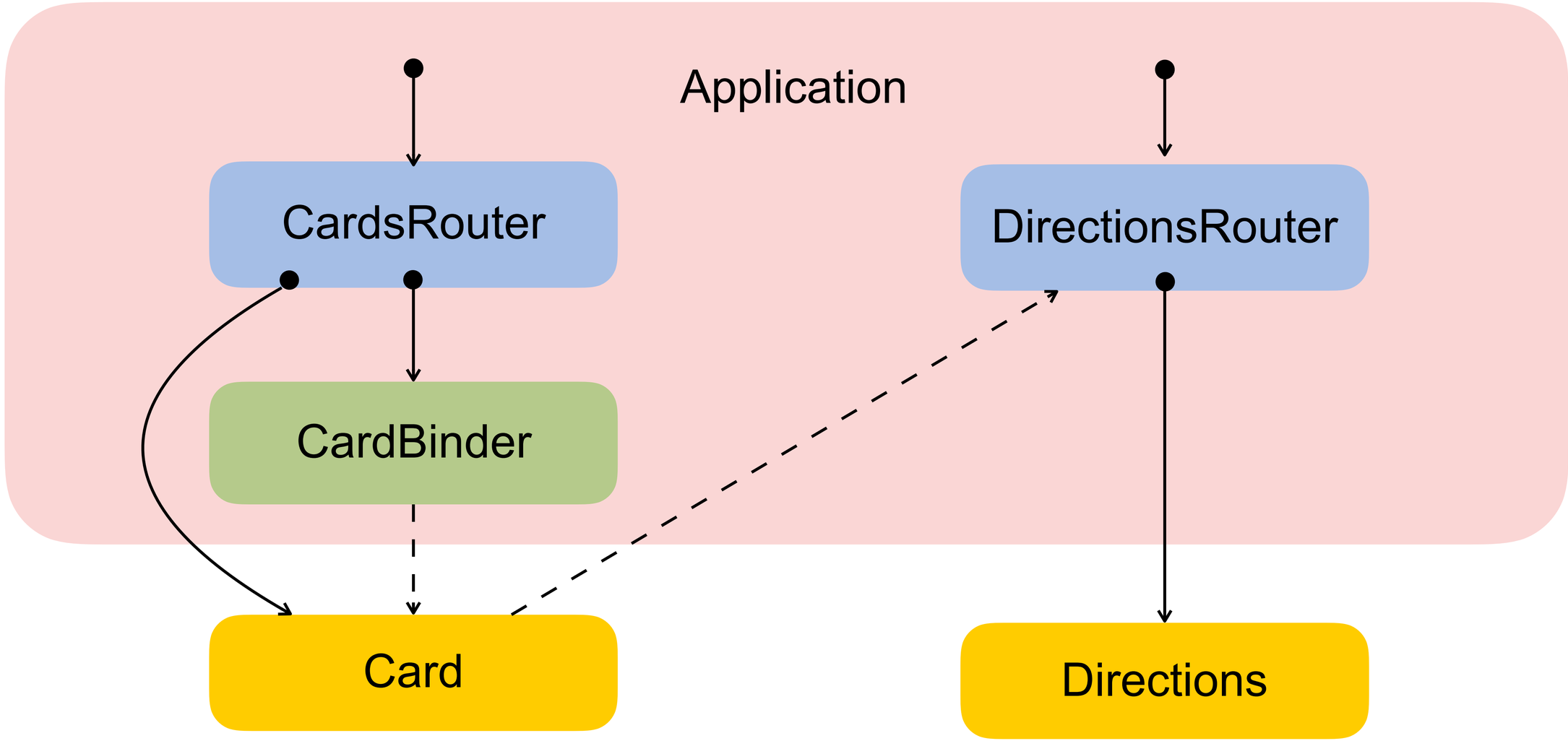
```
class OrganizationCard {

    init(directionsRouter: DirectionsRouter) {
        //...
    }

    func onDirectionButtonTapped() {
        directionsRouter.routeToDirections(wayPoint: wayPoint)
    }

}
```

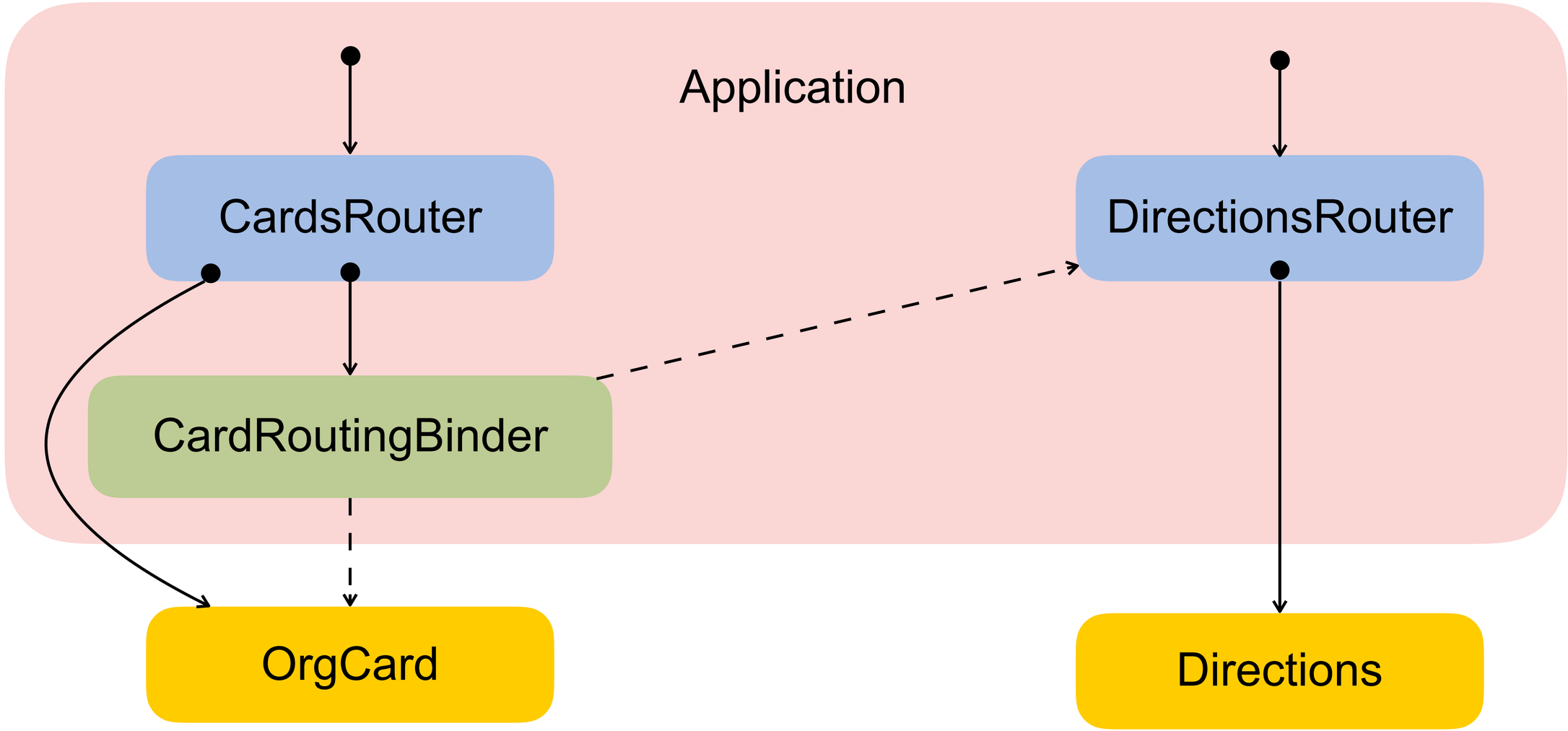
Схема с Router



RoutingBinder

- Отдаем из модуля наружу события возможных переходов
- Снаружи создаем **RoutingBinder**, обрабатывающий эти события

Схема с RoutingBinder



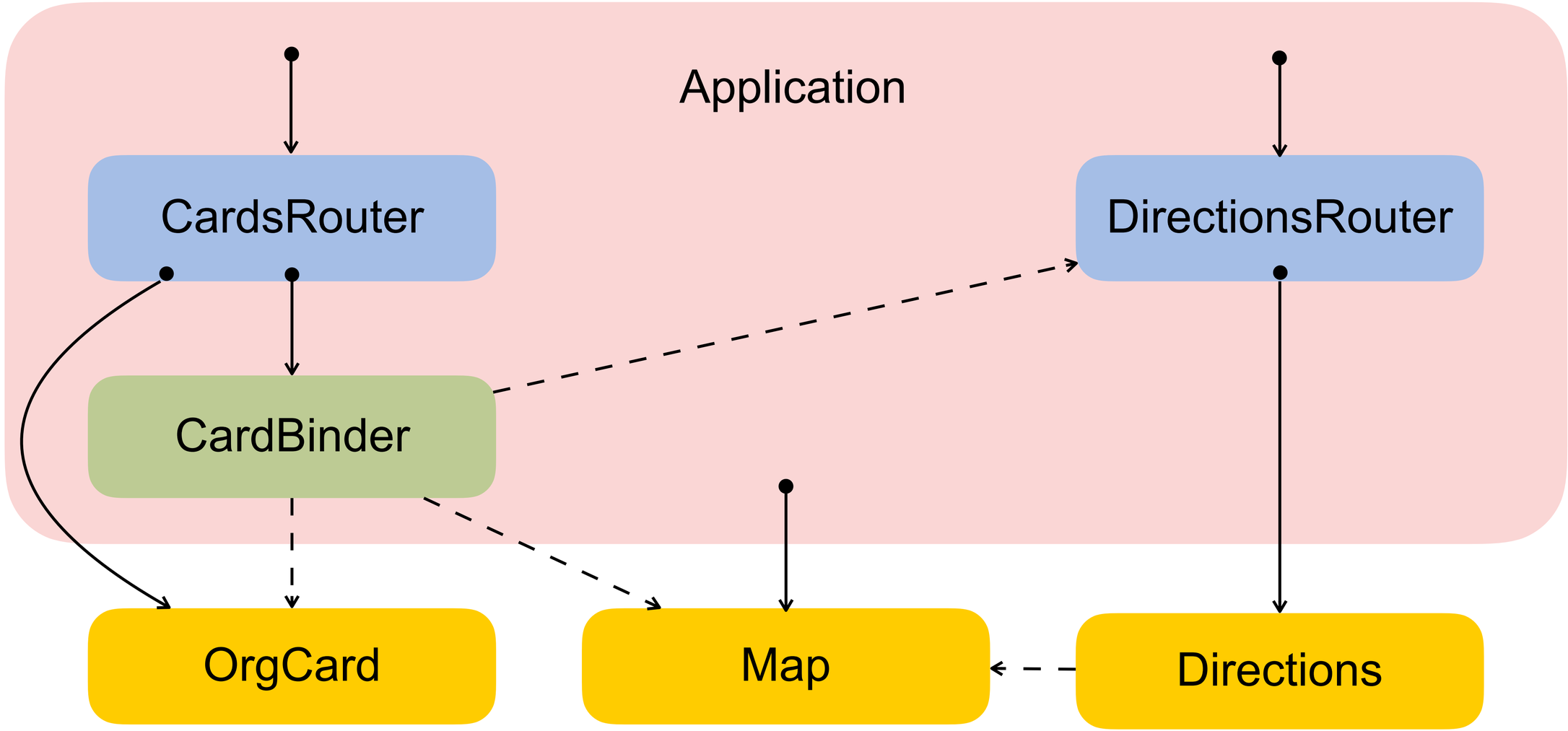
Router & RoutingBinder

Получили новый слой приложения, инкапсулирующий логику обработки переходов

Кладем в **App** или в **Deps**, отдельно от модулей

- Очистили от этого кода сами модули
- Потенциально модули не зависят от того, как их показывают

Собираем вместе



Что получили

- Два новых слоя приложения - Routers & Binders
- Минимум зависимостей у модулей
- Меньше зона ответственности
- Модули легче тестировать
- Понятно куда класть «грязный код»

Остались за кадром

- Внутримодульная архитектура
- Особенности реализации роутинга
  - Сброс состояния приложения
  - Восстановление состояния приложения
  - Обработка URL-схем

Заключение

Цели разбиения на модули

- Преодоление проблем большого монолита 
- Удобство работы с тестовыми проектами для фичей 

Deps

Вынесли имплементации зависимостей модулей из основного приложения в Deps

- Легче создавать тестовые проекты

Interfaces

Вынесли интерфейсы в отдельные модули

- Инверсия зависимостей между модулями
- Переиспользуемость модулей другими командами
- Возможность писать продвинутые тесты

Cocoapods

- Монорепа
- Модули с тестами и ресурсами
- Сабспеки для грануляции Deps & Interfaces

Routers & Binders

Вынесли из модулей лишние зависимости и логику

- Четкая ответственность модулей
- Упрощение тестирования
- Выше потенциал переиспользования

Профессиональное развитие

Разбиение приложения на модули - вызов для команды

Возникает масса архитектурных и организационных проблем, весь опыт проходит проверку на прочность

- Было интересно!

Заключение

Инфраструктурные и архитектурные изменения нужно рассматривать как инвестиции

- Подумайте, что хотите получить в итоге, прежде чем садиться за модульность
- Всегда должен быть план

Спасибо за внимание

Николай Лихогруд

Руководитель группы разработки Яндекс.Карт для iOS

 likhogrud@yandex-team.ru

https://medium.com/yandex-maps-ios