#Mobius #hh_ru

The Lord of The Modules

Обо мне

Александр Блинов

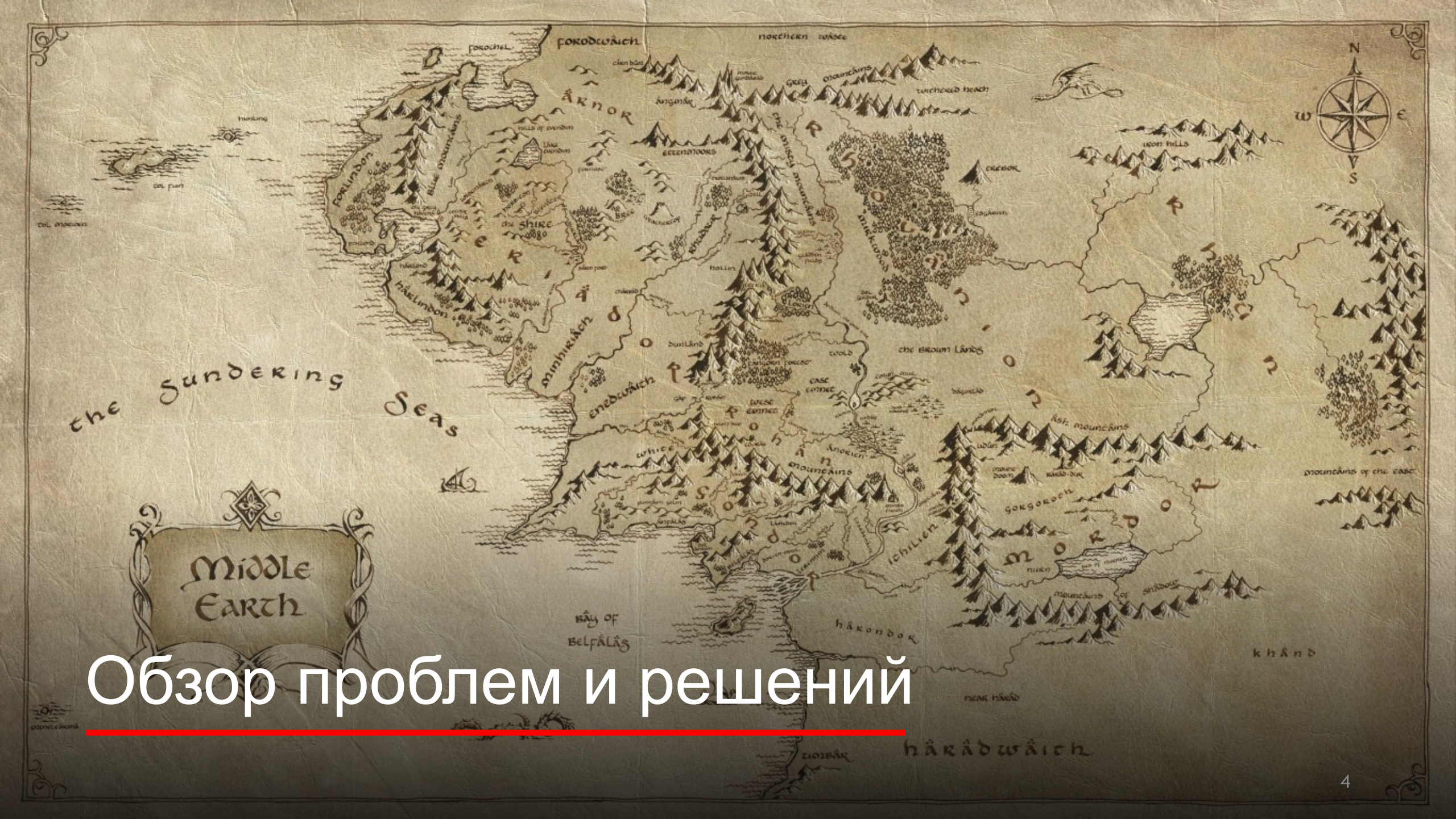
Руководитель Android направления

Содержание

1. Обзор проблем
2. Картина “TO BE”
3. Переход от “AS IS” к “TO BE”
4. Динамические фиче-модули
5. Подведение итогов

Слайды

goo.gl/sQMfda



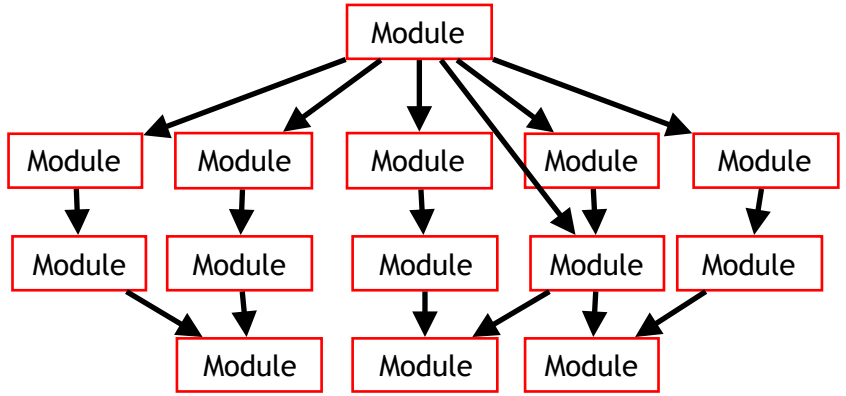
Обзор проблем и решений

Модуляризация и зачем она нужна

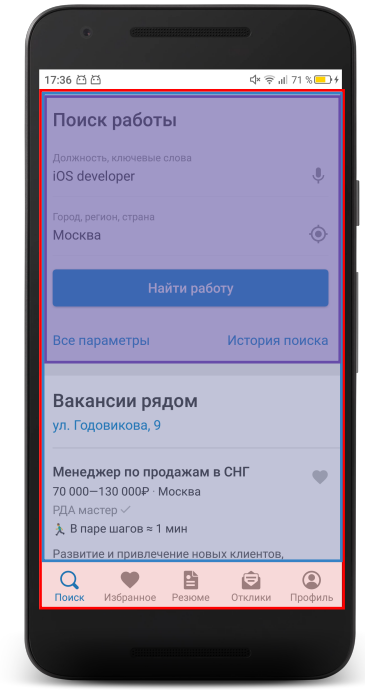
1. Изоляция фичей
2. Скорость сборки

Связи в приложении

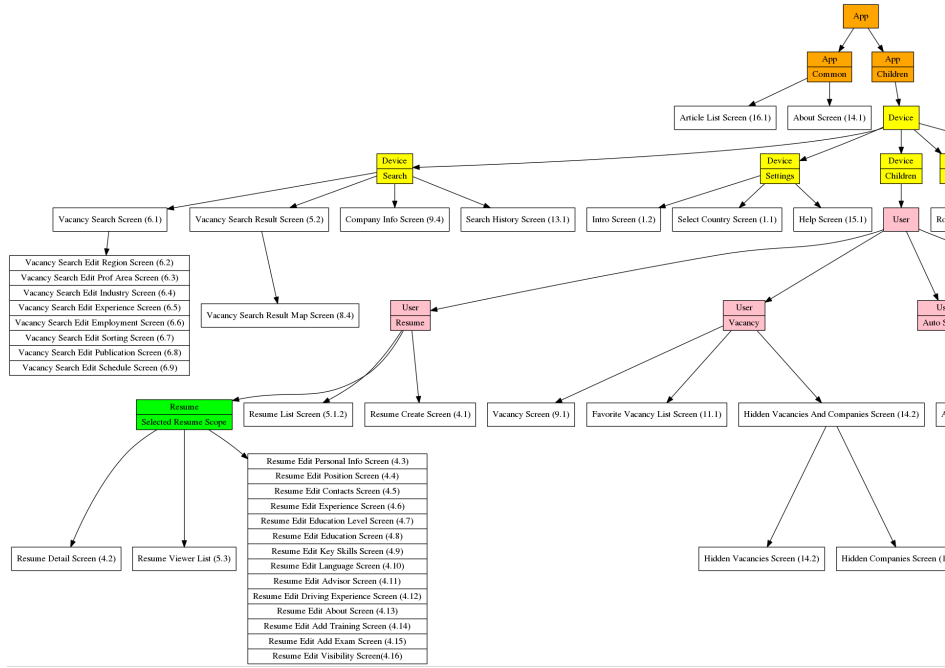
Зависимости сборки



Иерархия view



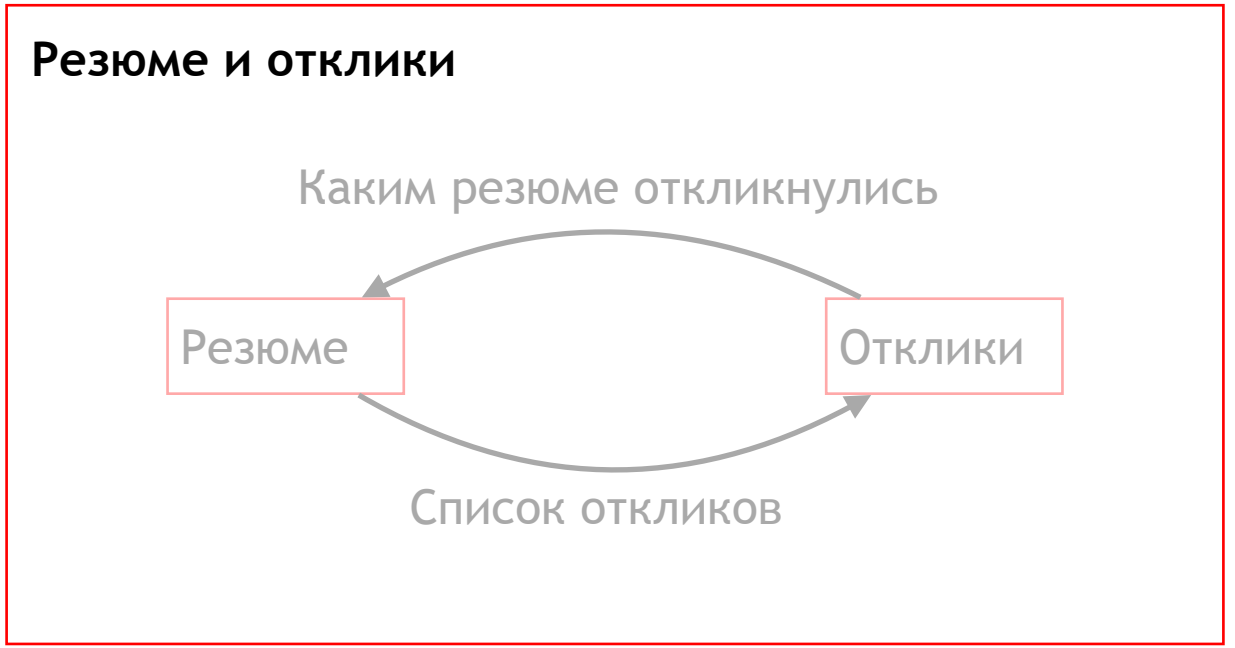
Время жизни скоупов



Циклические зависимости

От циклических зависимостей не уйти

Толстофичи



Толстофичи как черные дыры

Картина “TO BE”

Разбиение на фиче-модули

Ожидания от подхода

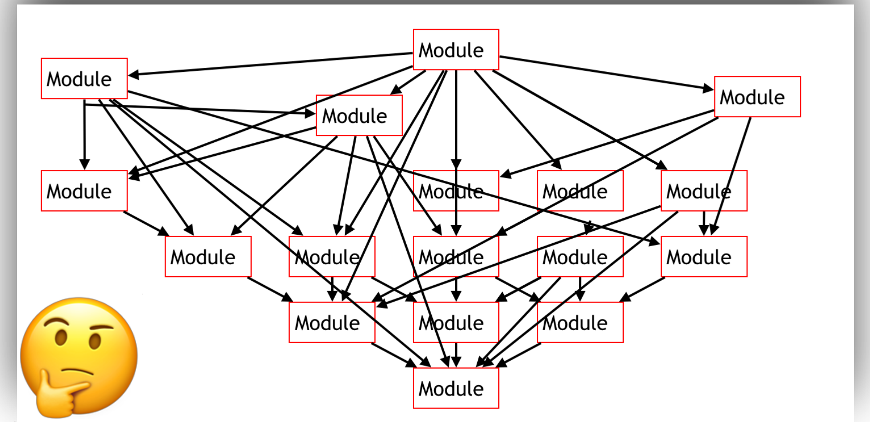
1. В фиче доступны все необходимые зависимости
2. Код фичи легко отделить и тиражировать для экспериментов
3. Низкая связность фичей
4. Наличие механизм навигации (диплинки / апплинки)

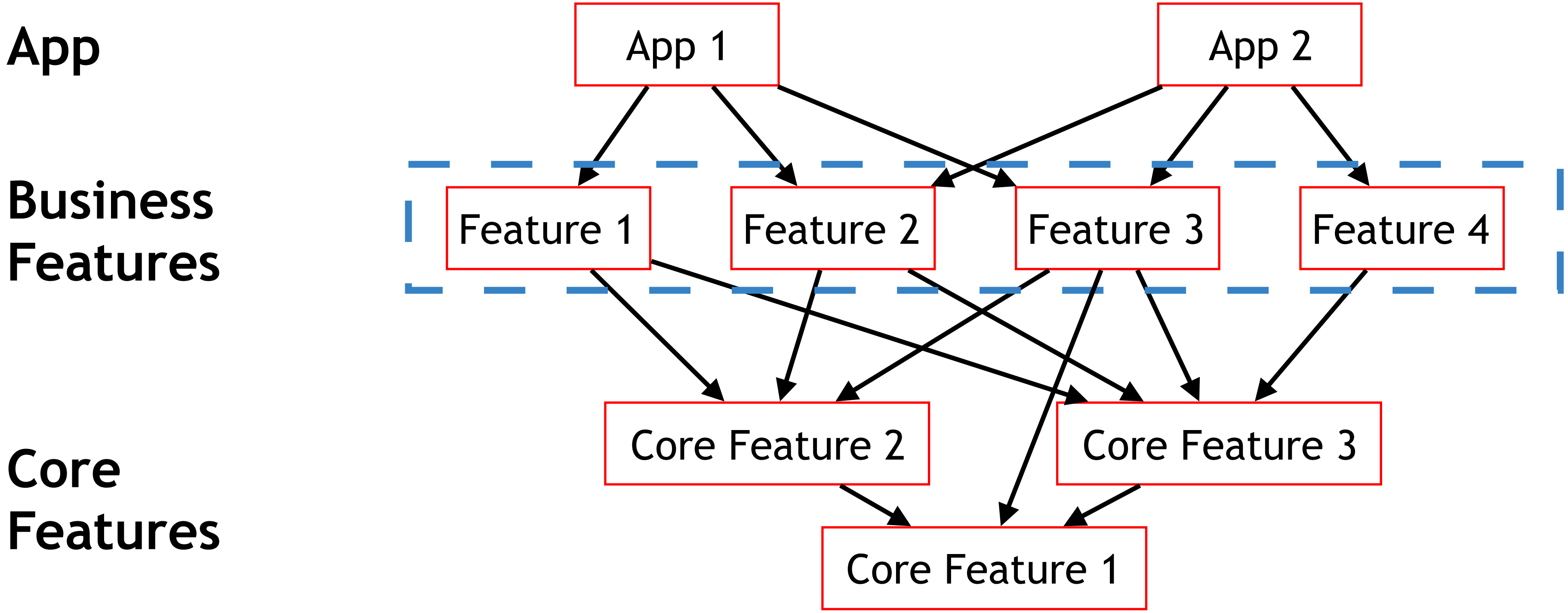
Что такое Feature?

Программный модуль со следующими характеристиками:

	Business Feature	Core Feature
Характеристика Программного модуля	• Логически законченный • Максимально независимый • Имеет четко обозначенные внешние зависимости • **Решает конкретную бизнес задачу**	• Логически законченный • Максимально независимый • Имеет четко обозначенные внешние зависимости • **Практически не относится к бизнес логике приложения**
Пример фичи	• Авторизация • Профиль • Резюме	• Base UI • List & Pagination • Metrics & Analytics • Network
Логические слои	Включает все слои (по clean architecture) логики	Произвольные части

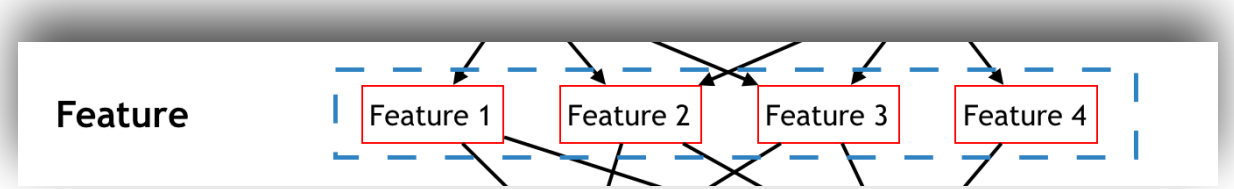
Слои модулей





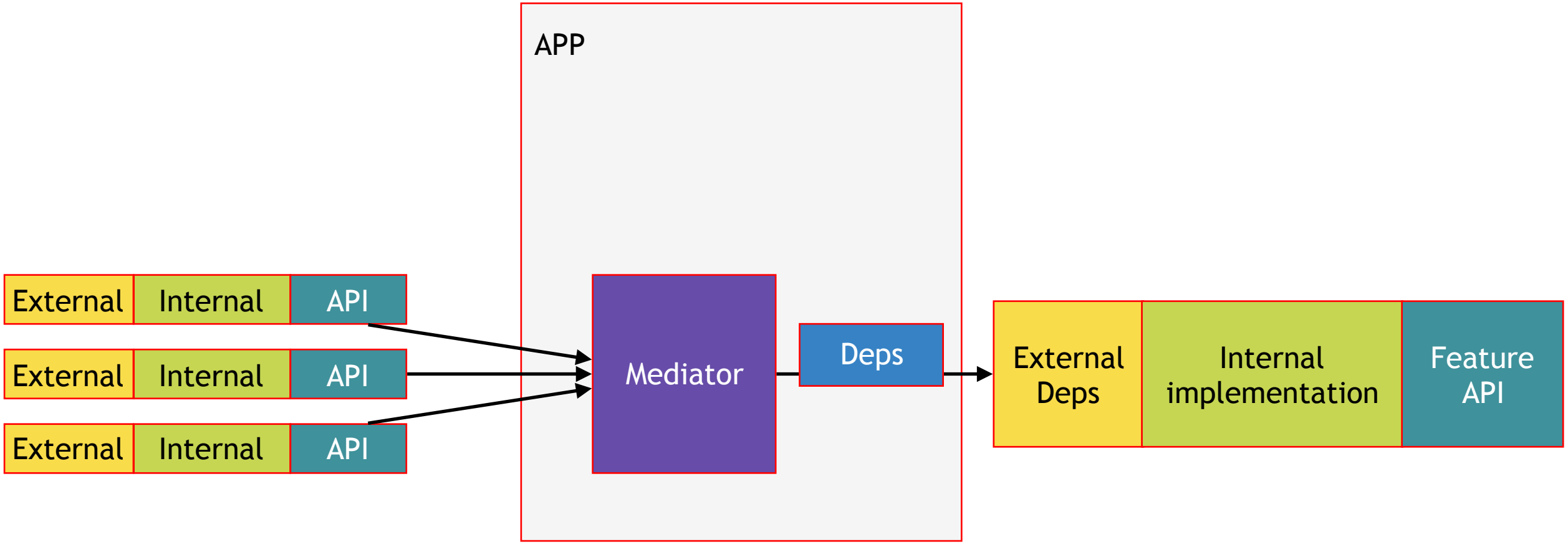
Business Feature

Устройство

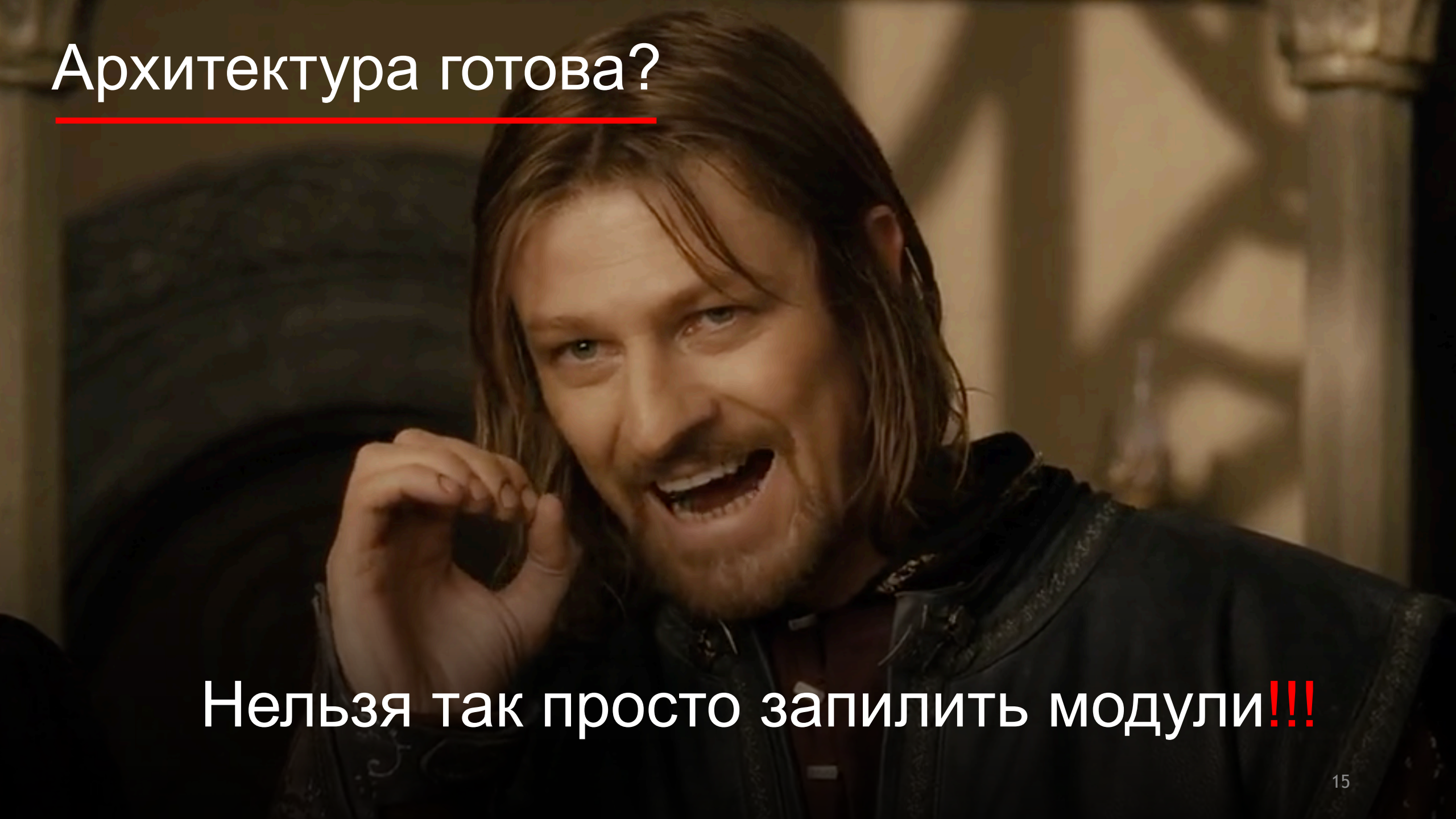


Business Feature

Взаимодействие через Mediator



Архитектура готова?

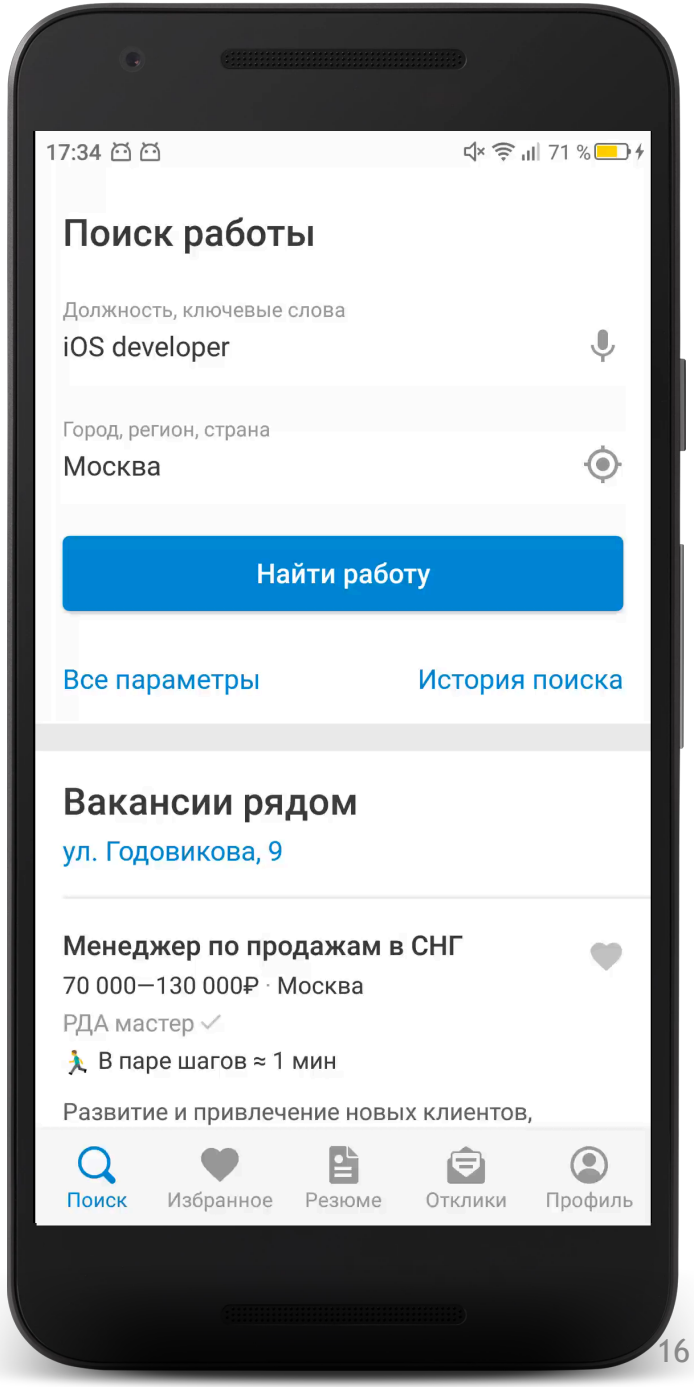


Нельзя так просто запилить модули!!!

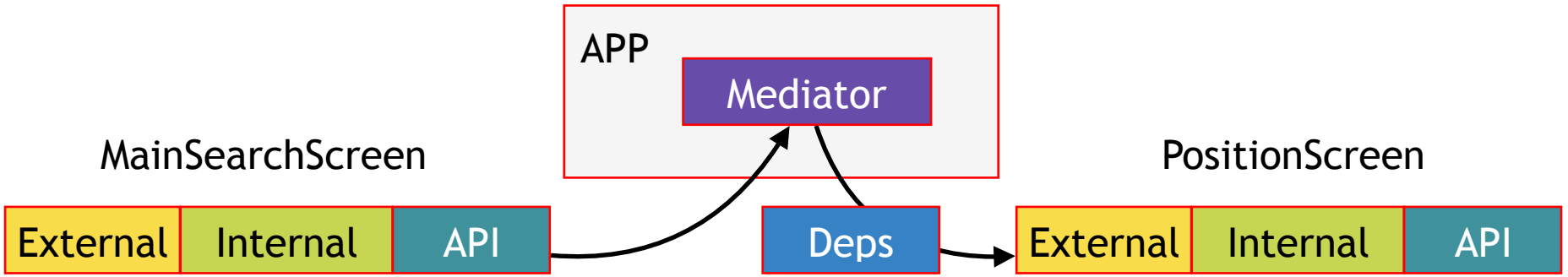
Устройство фичи

Пример работы

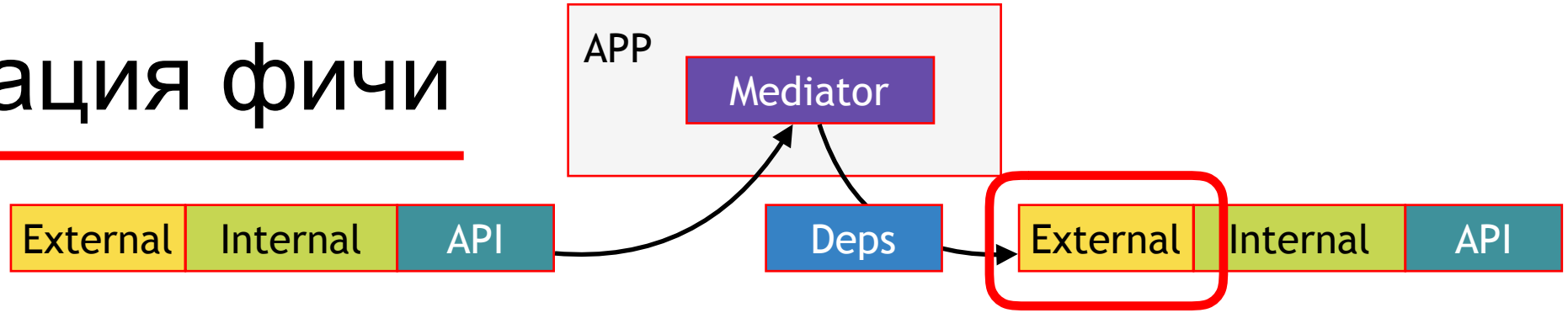
Feature Position



Инициализация фичи

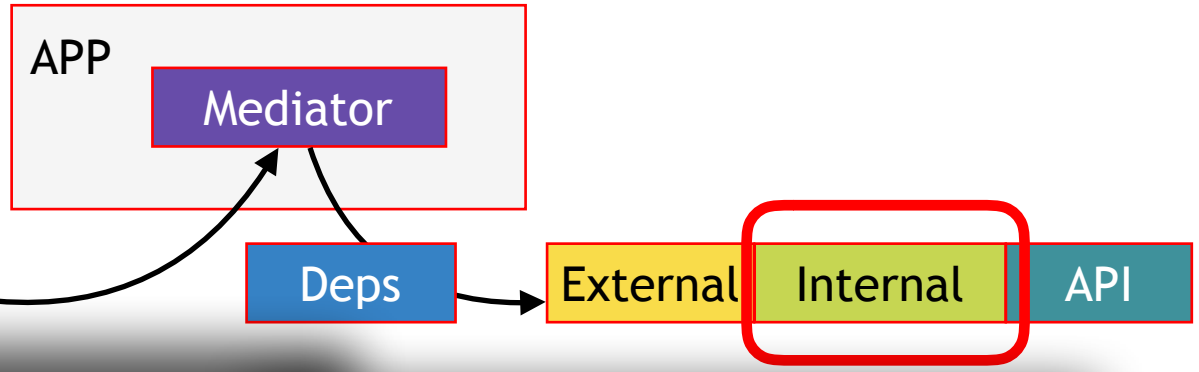


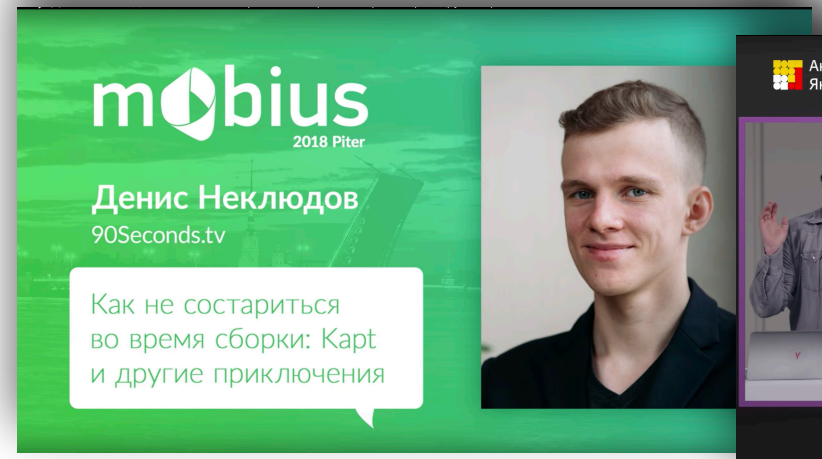
Инициализация фичи



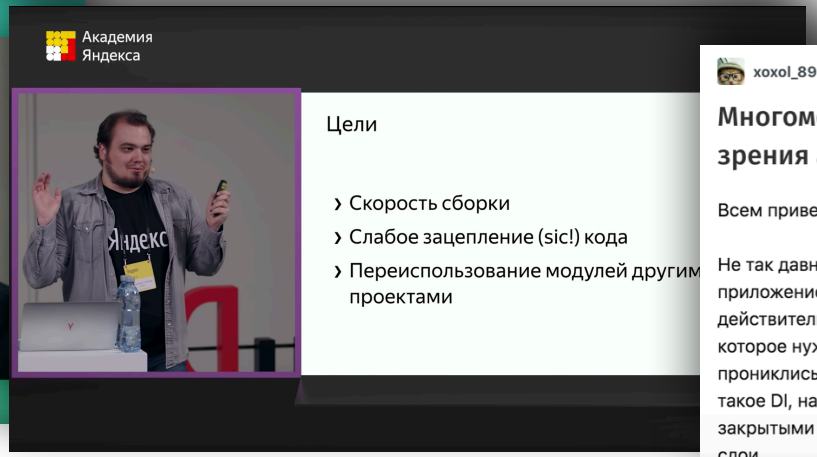

```
interface PositionDependencies {

    fun getCurrentPosition(): String

    fun setCurrentPosition(position: String)

}
```

Инициализация фичи





goo.gl/BQHXEM

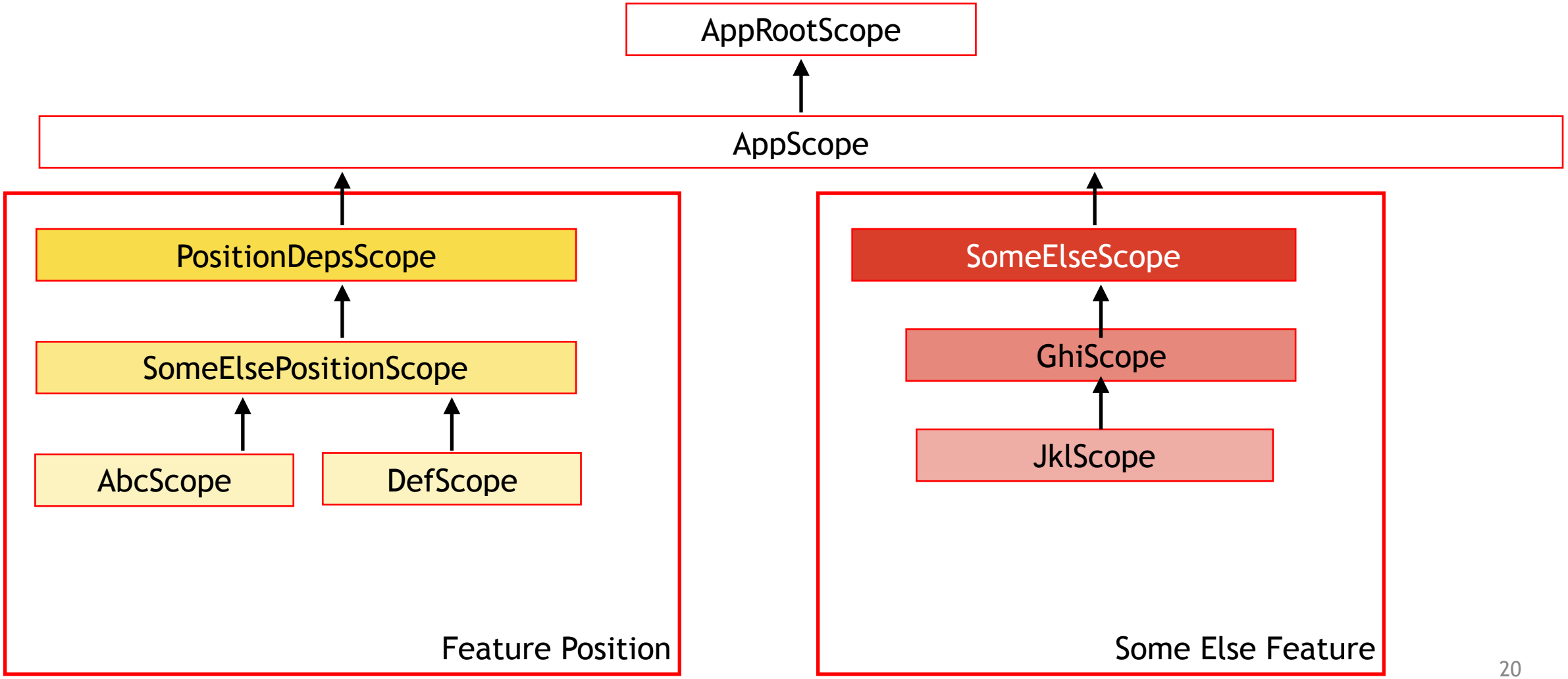


goo.gl/9c61dz

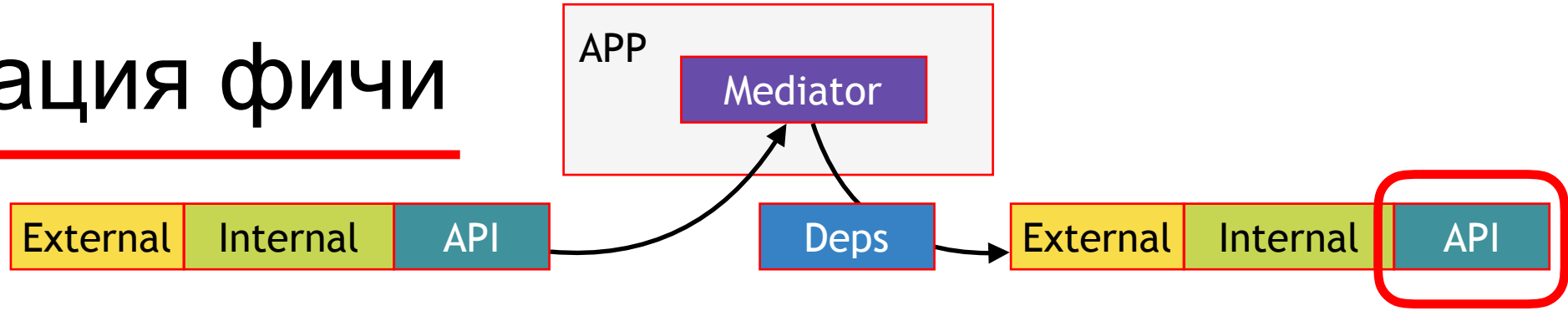
goo.gl/rej6kN

1. Dagger 2
2. Toothpick
3. . . .

Глубина скоупов зависимостей

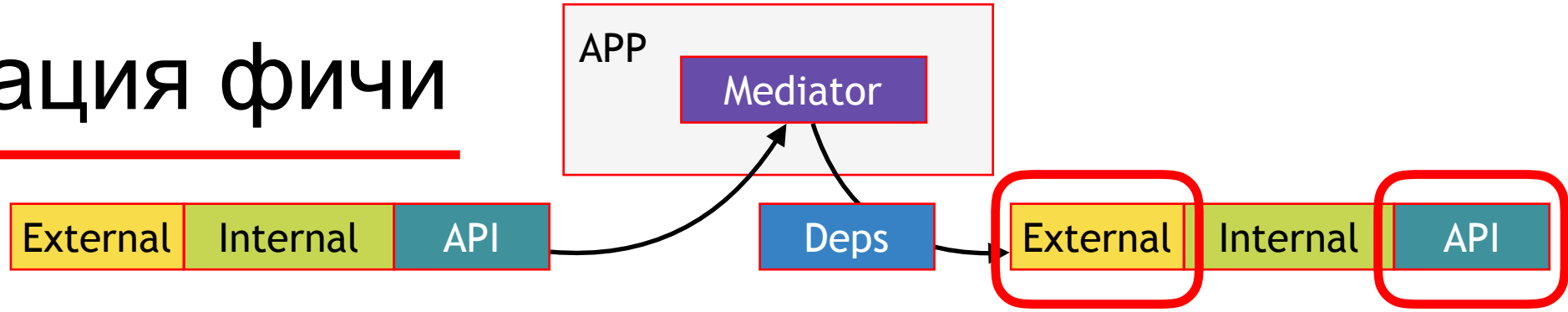


Инициализация фичи



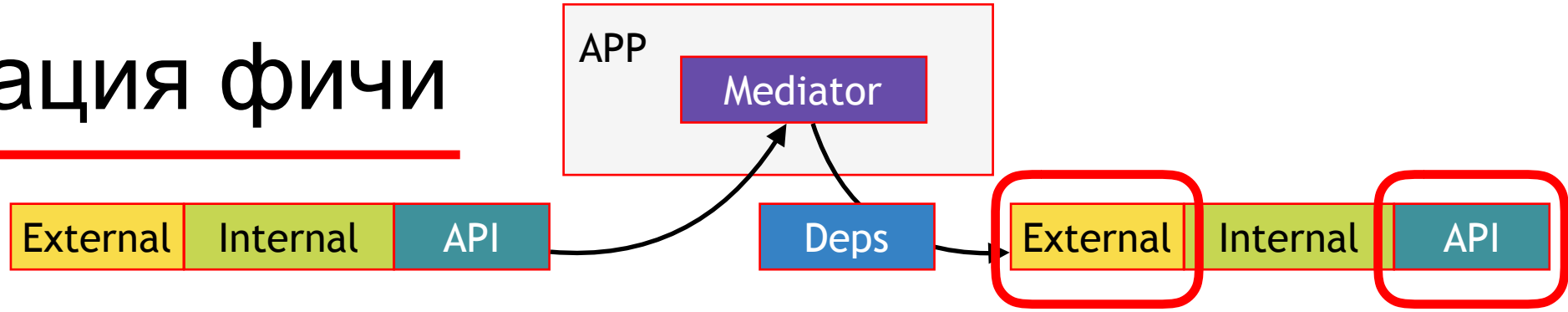

```
interface FeatureApi {

    fun getPositionFragment(): Fragment

    //Допустимые внешние зависимости

    fun getSomeIntent(): SomeIntent

    fun getSomeInteractor(): SomeInteractor

    fun getSomeRepository(): SomeRepository

    // …
}
```

Инициализация фичи



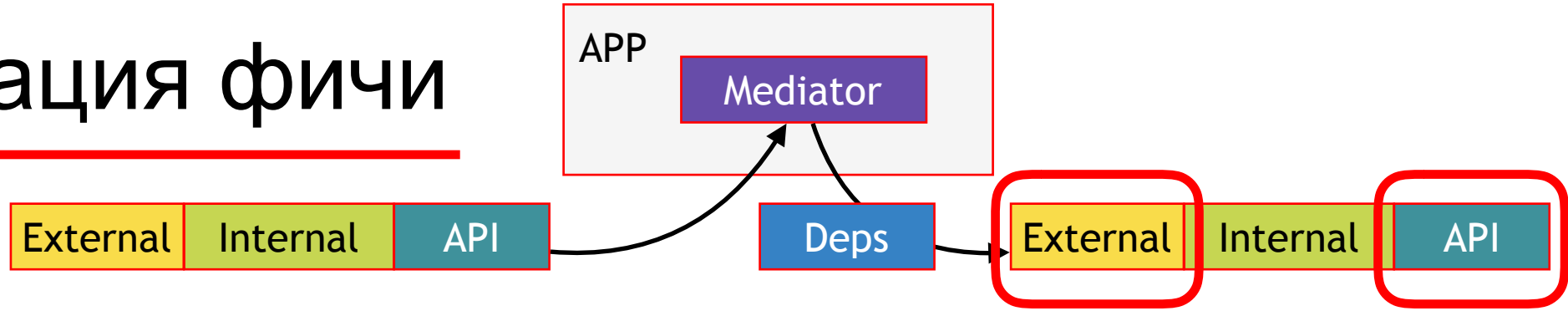

```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
    }

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, ROOT_SCOPE)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(ROOT_SCOPE)
    }

    val api: FeatureApi = FeatureApiIml(ROOT_SCOPE)
}
```

Инициализация фичи



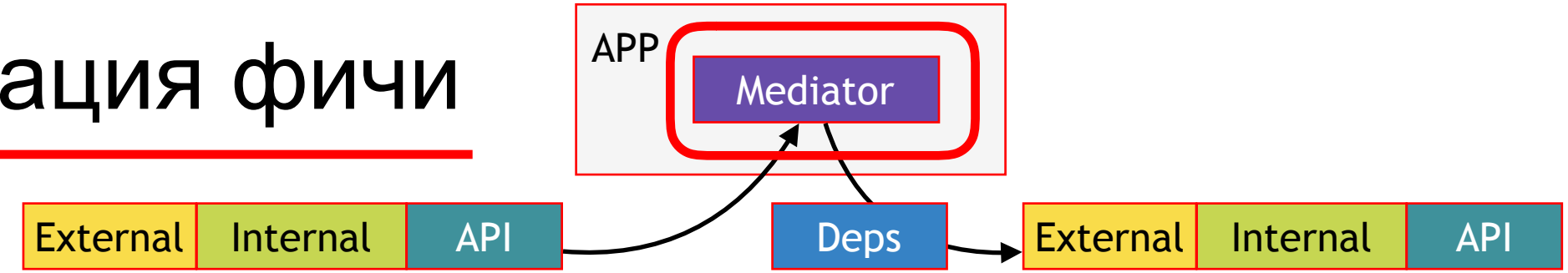

```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
    }

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, ROOT_SCOPE)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(ROOT_SCOPE)
    }

    val api: FeatureApi = FeatureApiIml(ROOT_SCOPE)
}
```

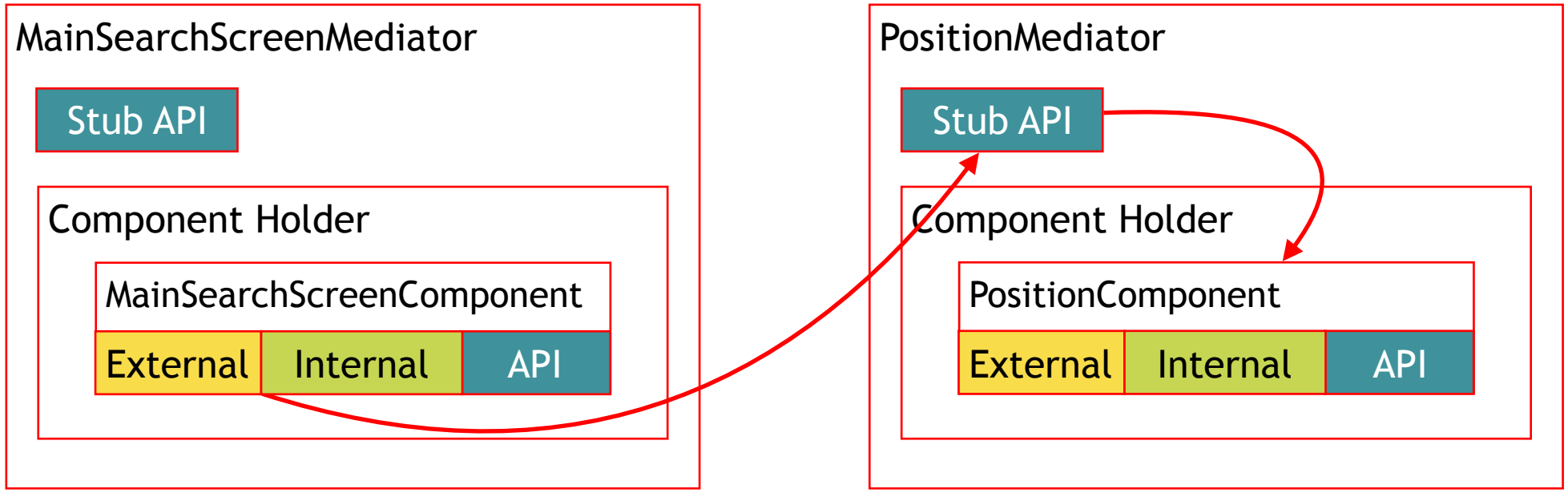
Инициализация фичи



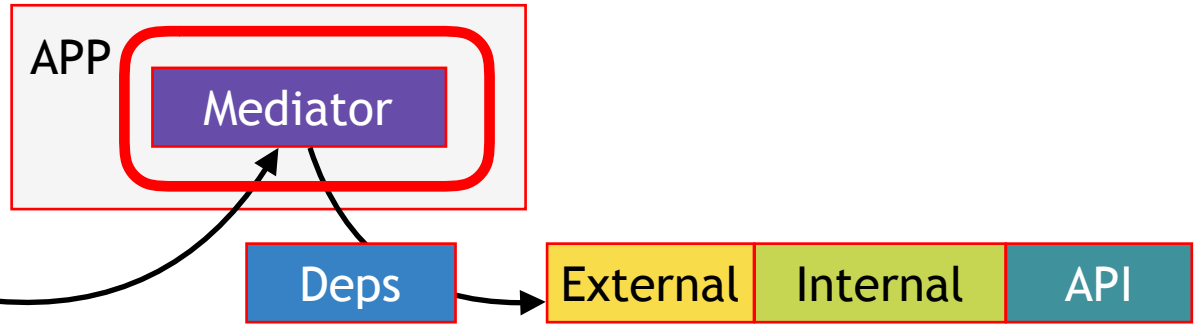

```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
    }

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, ROOT_SCOPE)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(scopeName)
    }

    val api: FeatureApi = FeatureApiIml(ROOT_SCOPE)
}
```

Инициализация фичи



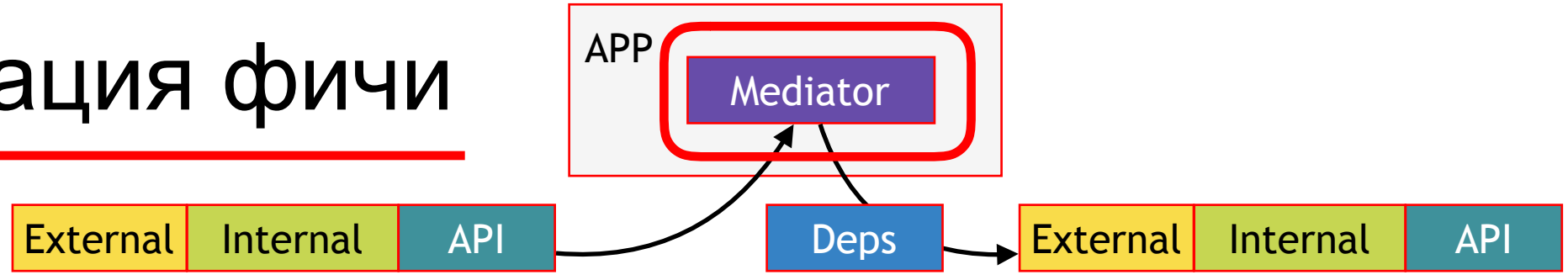


Инициализация фичи



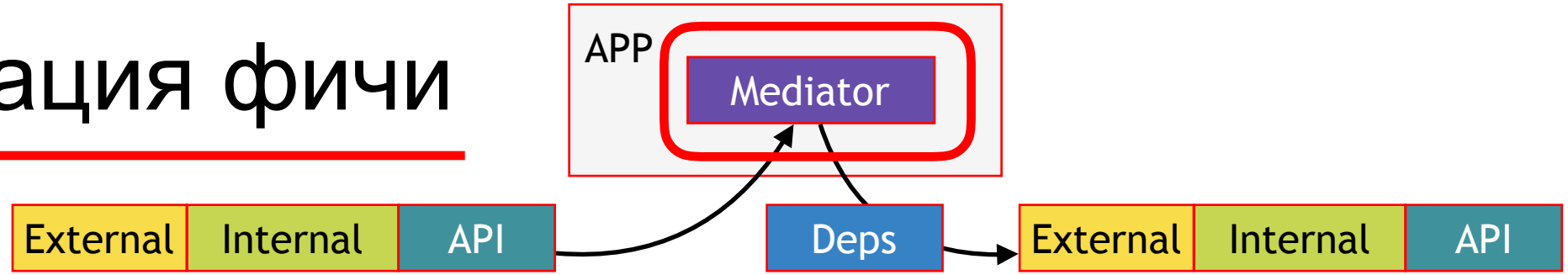

```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
       }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи



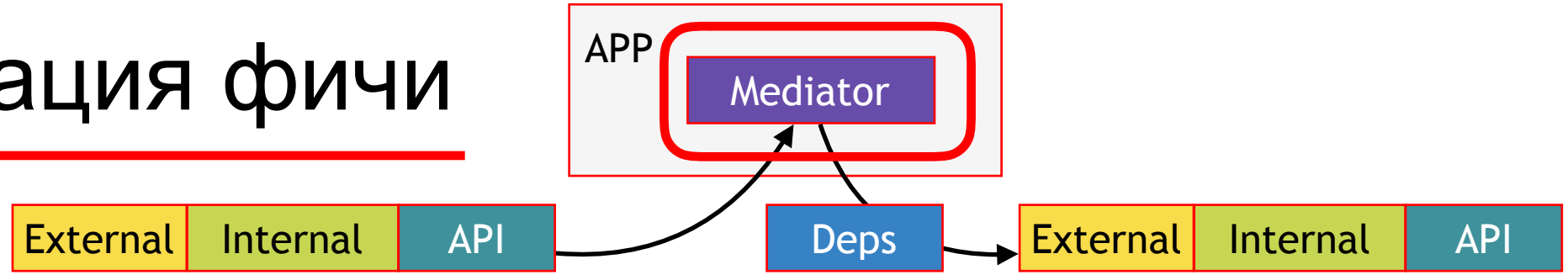

```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
       }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи



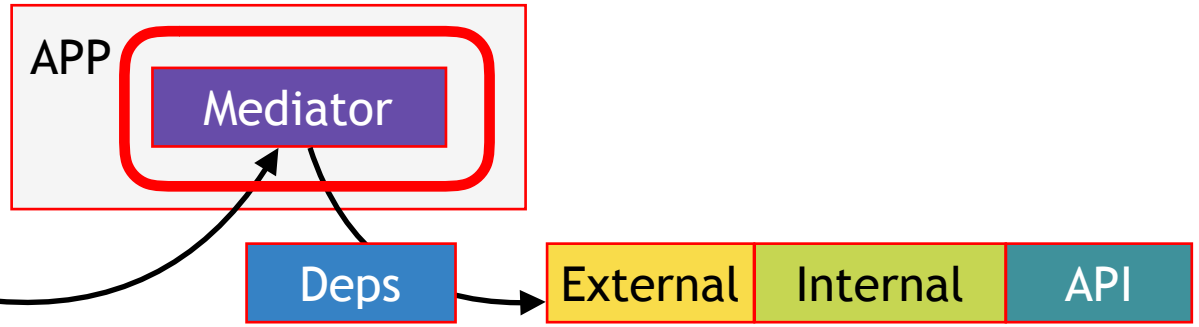

```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
       }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи



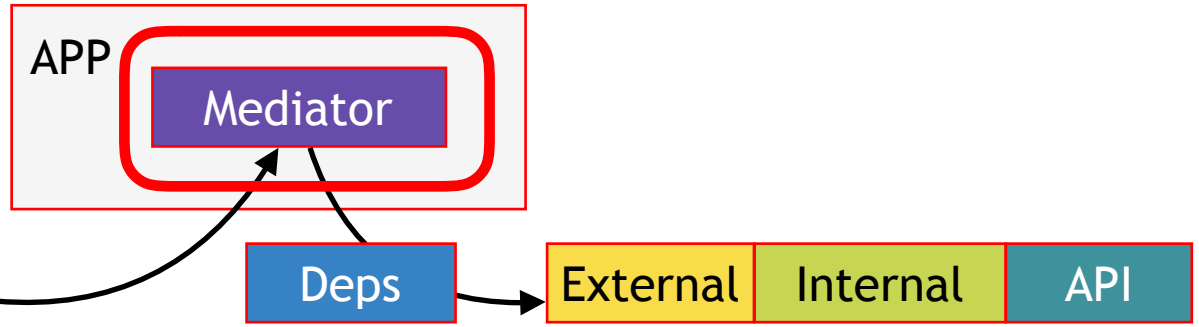

```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
        }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи



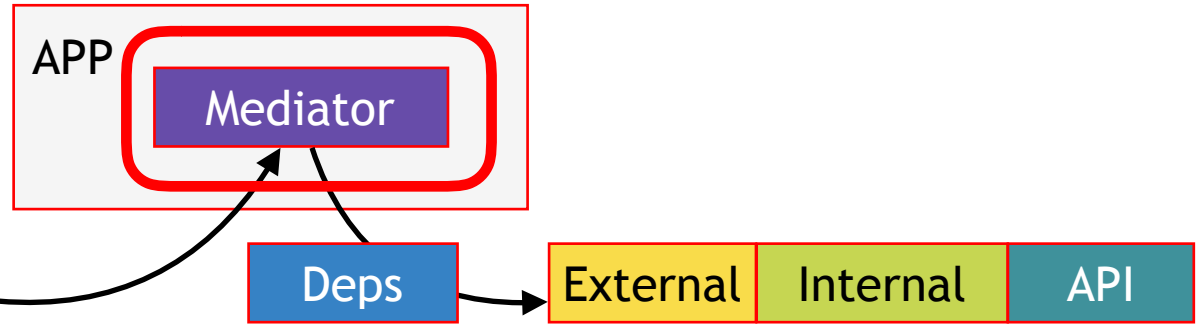

```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
        }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи

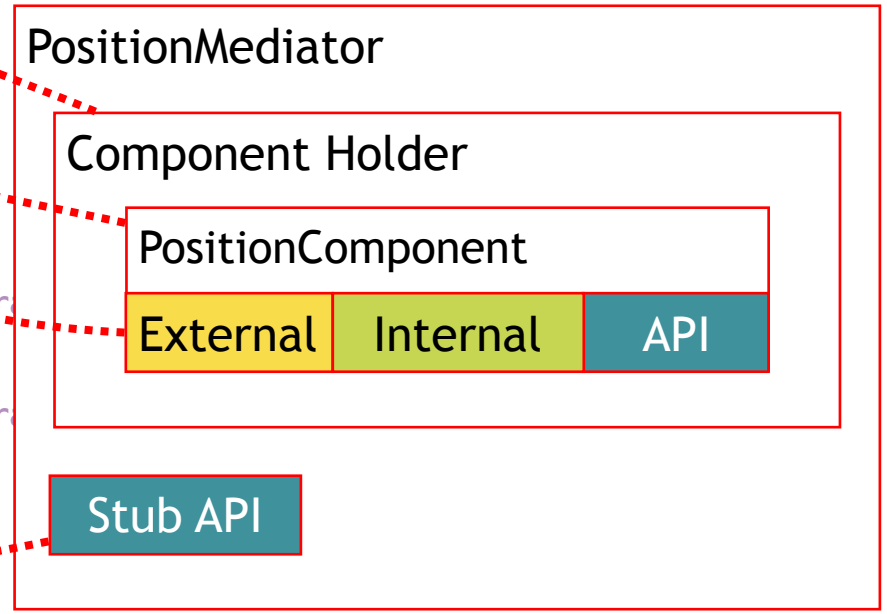



```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.getPositionFragment()
                }
                override fun setCurrentPosition(p: String) {
                   mediatorManager.mainSearchMediator.apiStub.positionInteractor.setPositionFragment(p)
                }
          })
        }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```

Инициализация фичи




```
class PositionMediator(private val mediatorManager : MediatorManager) {

    private val componentHolder: SingleComponentHolder<PositionComponent, PositionDependencies> =
          SingleComponentHolder { deps -> PositionComponent(deps) }

    private fun provideComponent(): PositionComponent {

        if(!componentHolder.hasComponent()){

          componentHolder.initComponent(object : PositionDependencies {
                override fun getCurrentPosition(): String() {
                   mediatorManager.mainSearchMediator.apiStub.positionInter
                }
                override fun setCurrentPosition(p: String){
                   mediatorManager.mainSearchMediator.apiStub.positionInter
                }
          })
       }
       return componentHolder.provideComponent(deps)
    }

    val apiStub = object : FeatureApi {
        override fun getPositionFragment(): Fragment {
            return provideComponent().api.getPositionFragment()
        }
    }
     // ...
}
```



Навигация

Основные принципы

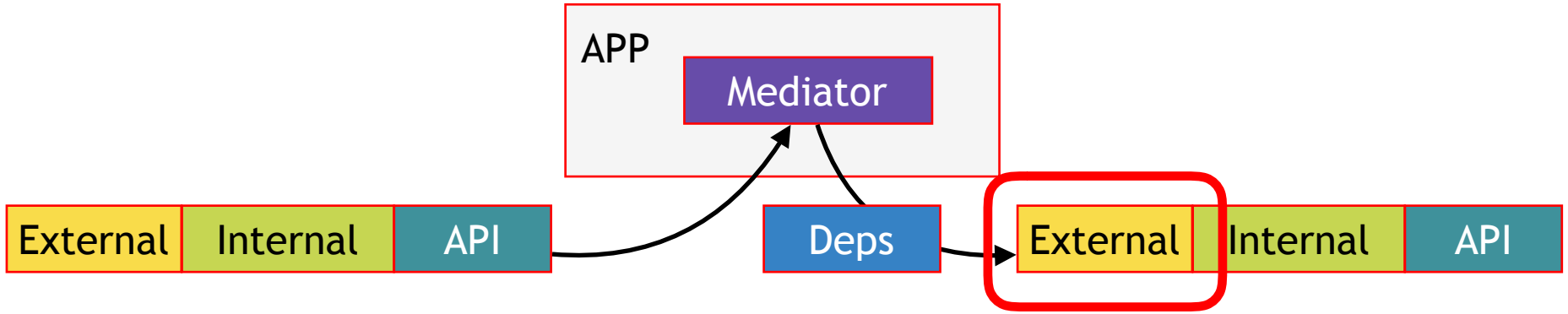
1. Навигация в фичи и App модулях
   - A. Старина Cicerone
   - B. Smart Router
   - C. State Machine
2. Навигация за предел фичи через Deps

goo.gl/Tnqw2a

goo.gl/tDmzEF

Навигация

Закрытие фичи

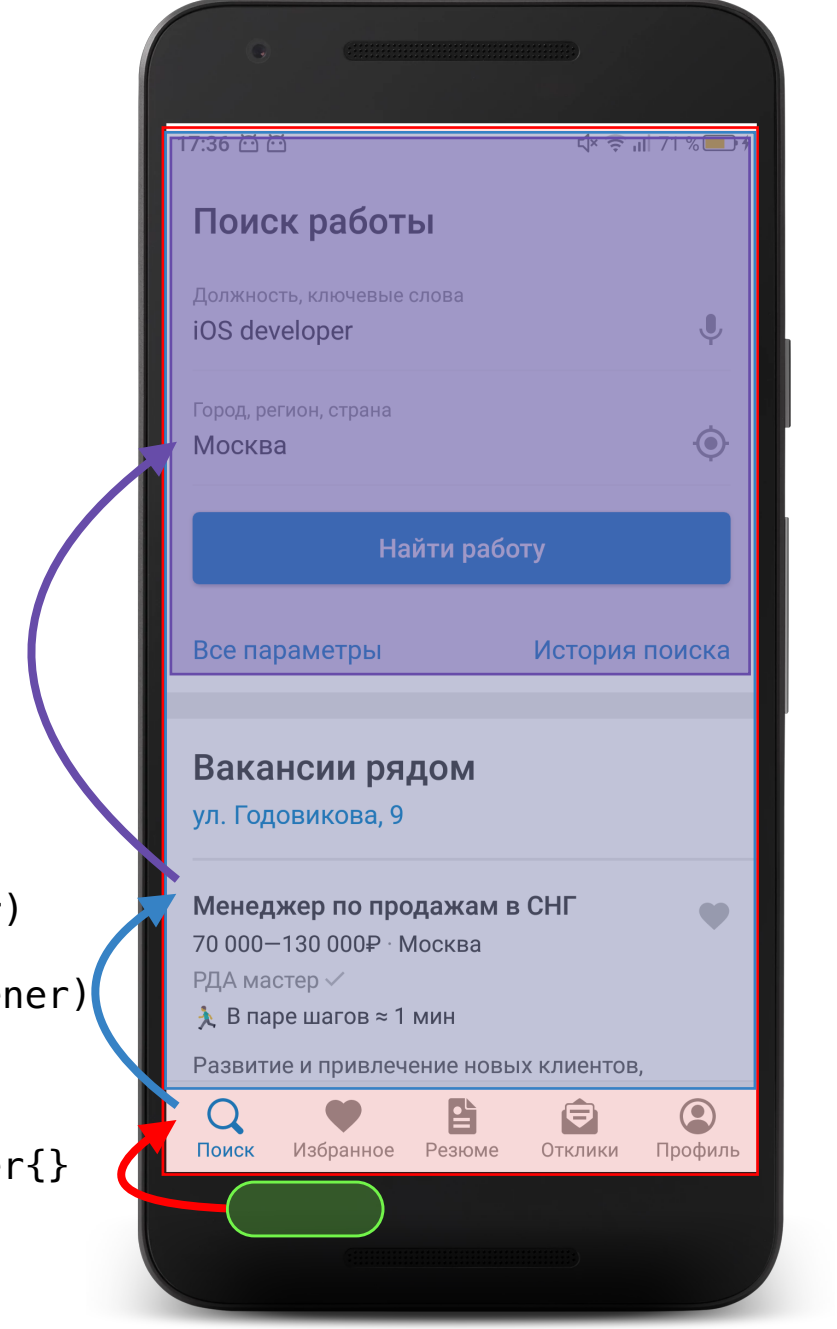



```
interface PositionDependencies {

    fun getCurrentPosition(): String

    fun setCurrentPosition(position: String)

    fun onClosePositionScreen()

}
```

Навигация

Туда и обратно

```
interface OnBackPressedListener {
    /**
     * @return true, если команда была обработана
     */
    fun onBackPressed(): Boolean
}

interface OnBackPressable {

    fun addOnBackPressedListener(listener: OnBackPressedListener)

    fun removeOnBackPressedListener(listener: OnBackPressedListener)
}

BaseFragment : Fragment(), OnBackPressable, OnBackPressedListener{}
```



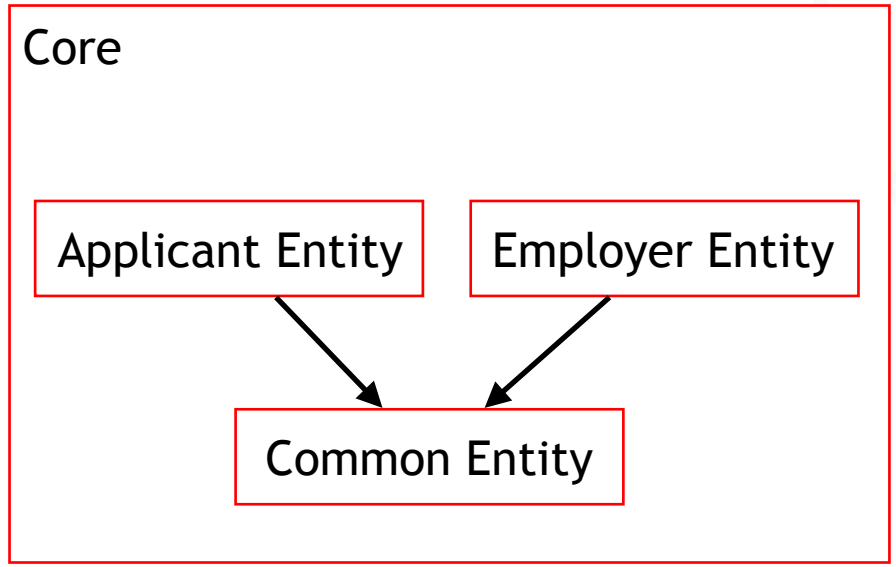
Навигация

Большие объекты

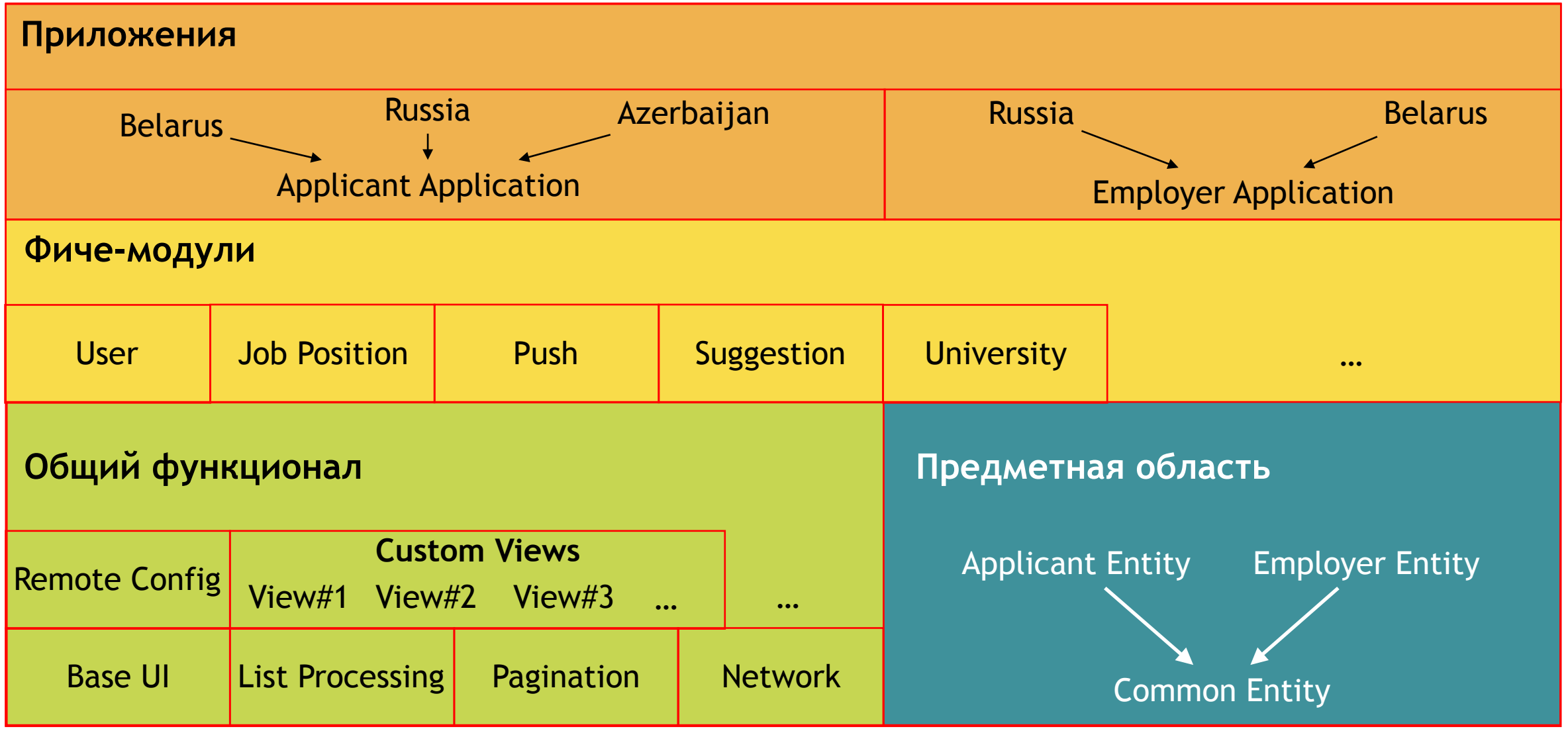
```
public class Resume {
    private Collection<SpecializationDto> specializations;
    private ProfessionalExperienceDto professionalExperience;
    private Set<ResumeCertificateDto> certificates;
    private AdditionalEducationDto additionalEducation;
    private BusinessTripReadiness businessTripsReadiness;
    private ContactInformationDto contactInformation;
    private RecommendationInfoDto recommendationInfo;
    private List<UserImageDto> userPortfolio;
    private EducationHistoryDto educationHistory;
    private RelocationInfoDto relocationInfo;
    private ResumeLanguageDto languages;
    private WorkExperience workExperience;
    private Set<Employment> employments;
    private EducationLevel education;
    private LocalDateTime creationTime;
    private LocalDateTime lastChangeTime;
    private Set<Schedule> schedules;
    private RoadTimeType roadTime;
    private ResumeStatus resumeStatus;
    private UserImageDto userImage;
    private AreaDto area;
    private AccessDto access;
    private LocalDate birthdayDate;
    private AreaDto[] citizenship;
    private AreaDto[] workTickets;
    private MoneyDto desirableCompensation;
    private MetroDto nearestMetro;
    private Gender gender;
    private String[] keySkills;
    private String hash;
    private String title;
    private String aboutMe;
    private int id;
    private int siteId;
    private boolean hasVehicle;
    private String[] driverLicense;
    //…
}
```

+

```
interface PositionDependencies {

    fun getCurrentPosition(): String

    fun setCurrentPosition(position: String)

    fun onClosePositionScreen()

}
```



Итоговая картина



Переход от “AS IS” к “TO BE”

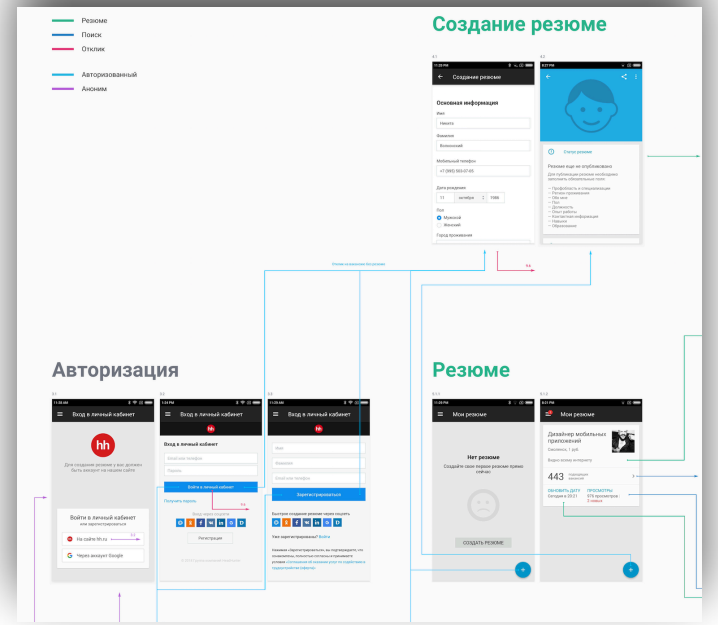
Контекст

1. Команда 3 человека, которая увеличится до 10
2. 2 приложения на одной кодовой базе в (!) одном модуле
3. Multi-Activity подход и монолитные Activity и Fragment
4. Dagger 2 и пухлый ApplicationScope
5. Отсутствие Event Bus, наличие синглтонов
6. End2end тесты основных сценариев на голом Espresso

Анализ слабых мест

Сложность приложения

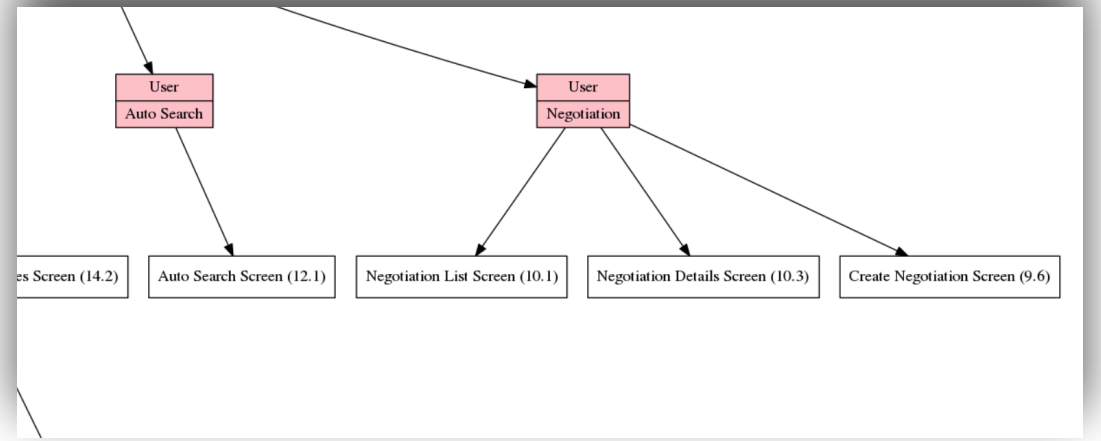
Карта экранов сценариев



overflow.io

Анализ зависимостей

```
structUserNegotiation [fillcolor="pink", style="filled", label="{ User| Negotiation }"];

NegotiationListScreen [label="Negotiation List Screen (10.1)"];
NegotiationDetailsScreen [label="Negotiation Details Screen (10.3)"];
CreateNegotiationScreen [label="Create Negotiation Screen (9.6)"];
```



graphviz.org

Шаг 0. Реанимация тестов

Тесты — это магия

Выбивание костылей должно регулярно тестироваться

Разделяем монолит

Шаг 1. Отделение Applicant и Employer

Flavors

Кодовая база

- Applicant code
- Employer code
- Common code

Иерархия

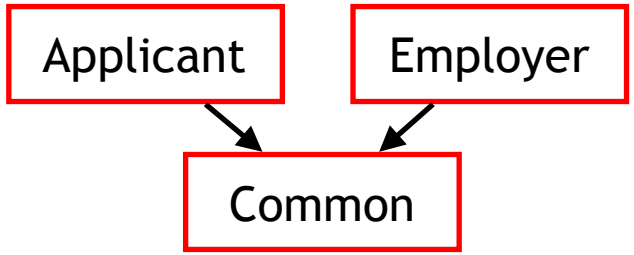
- Application

Modules

Кодовая база

- Applicant code
- Employer code
- Common code

Иерархия



Шаг 2. Подготовка DI

Временная(?) Замена Dagger2 на Toothpick

Dagger и гибкость

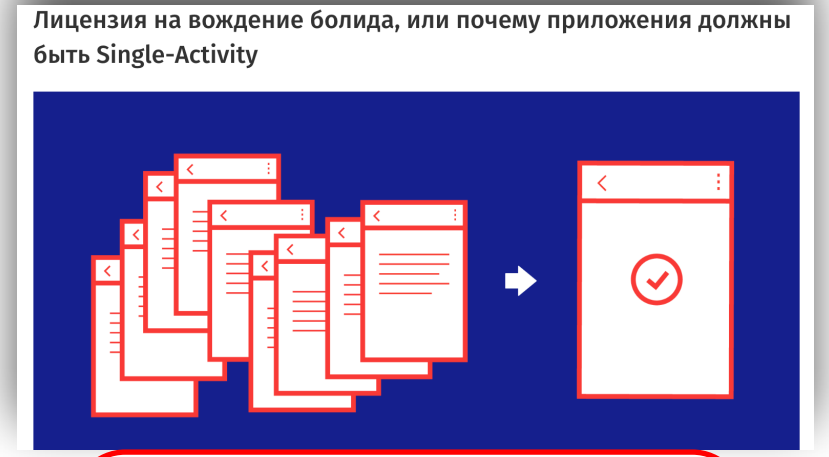
DI фреймворки можно комбинировать 🤔

Шаг 3. Вынос Core модулей

1. Core представляет иерархию модулей
2. Выносить Core фичи легко и весело

Шаг 4. Слезаем с Multi Activity

Зачем?



goo.gl/8QwRp4

Разные Activity должны связывать только сущности, сохраняемые системой

Шаг 4. Слезаем с Multi Activity

Действия

1. Замена Activity на Activity + Fragment
2. Замена транзакции фрагментов на Cicerone
3. Делаем механизм для роутинга (Smart Router)
4. Заменяем Activity + Fragment на Fragment

Не занимайтесь внедрением MVP / MVVM / MVI на этом шаге

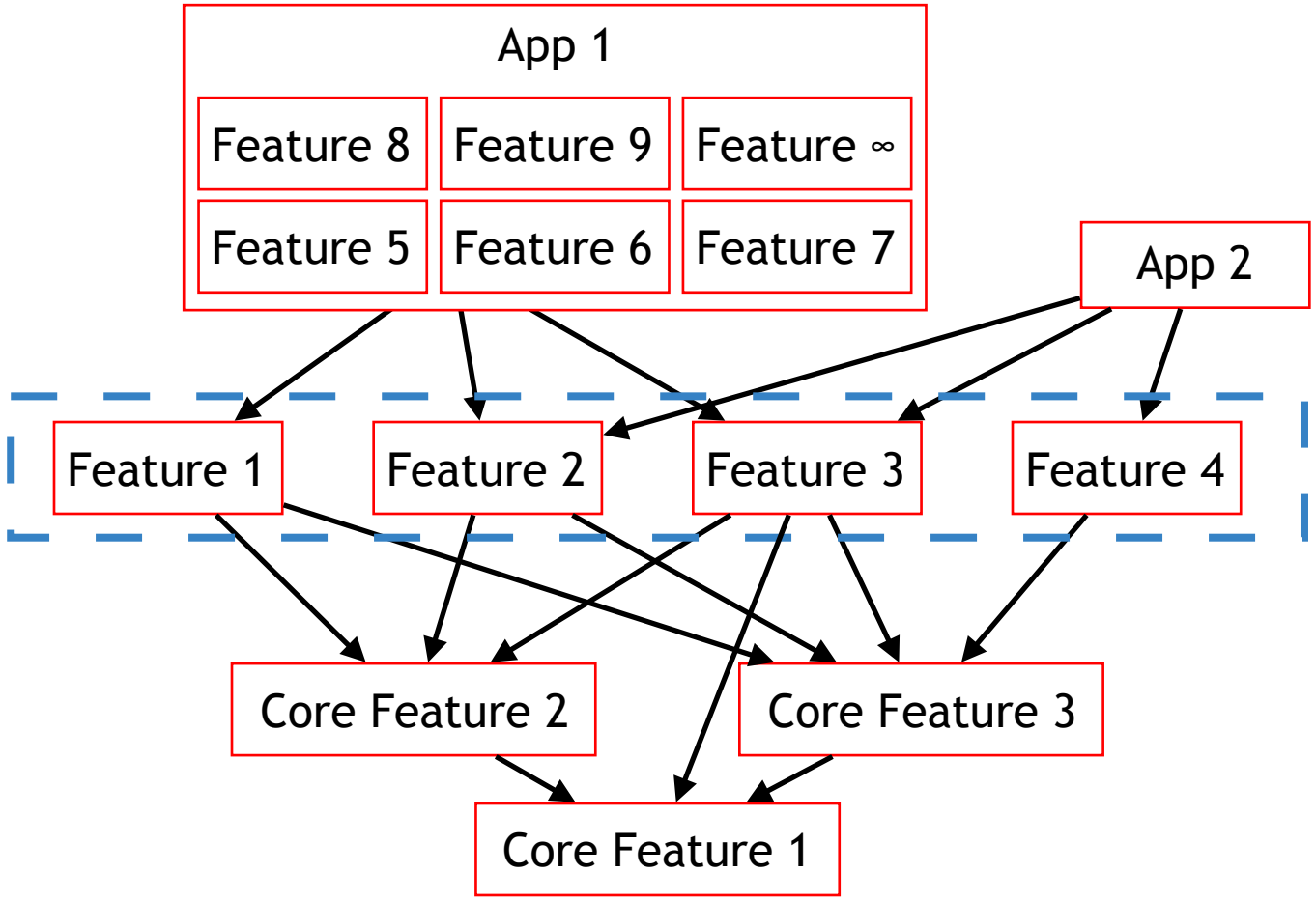
Шаг 5. Отпочкование фичей

Толстяк App модуль

App

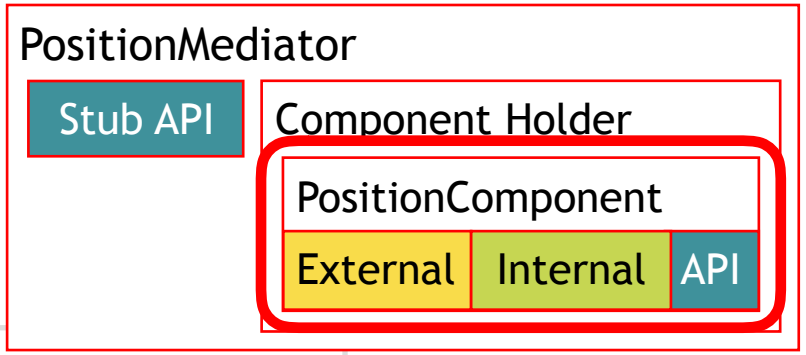
Business Features

Core Features



Шаг 5. Отпочкование фичей

Переиспользование

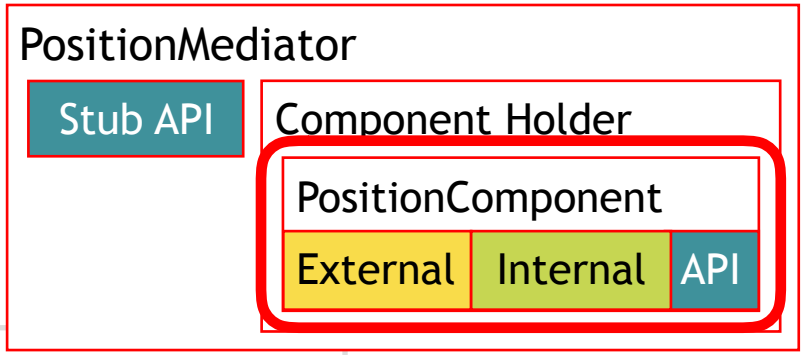



```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
        private val counter = AtomicInteger()
    }

    private val scopeName = ROOT_SCOPE + counter.incrementAndGet()

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, scopeName)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(scopeName)
    }

    val api: FeatureApi = FeatureApiIml(scopeName)
}
```

Переинициализация зависимостей

Шаг 5. Отпочкование фичей

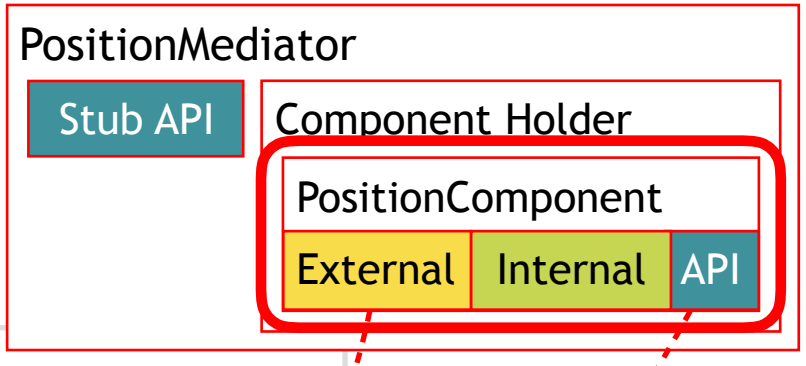
Переиспользование




```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
        private val counter = AtomicInteger()
    }

    private val scopeName = ROOT_SCOPE + counter.incrementAndGet()

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, scopeName)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(scopeName)
    }

    val api: FeatureApi = FeatureApiIml(scopeName)
}
```

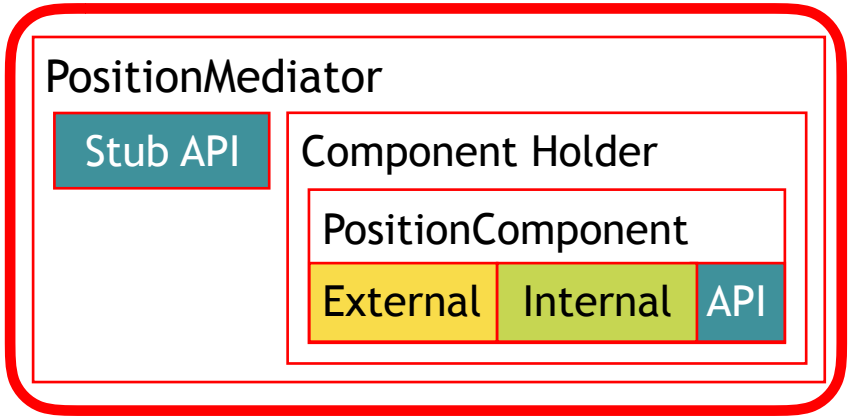
Шаг 5. Отпочкование фичей

Переиспользование




```
class PositionComponent(positionDependencies: PositionDependencies) {

    companion object {
        private const val ROOT_SCOPE = "${BuildConfig.APPLICATION_ID}_POSITION"
        private val counter = AtomicInteger()
    }

    private val scopeName = ROOT_SCOPE + counter.incrementAndGet()

    init {
        Toothpick.openScopes(APP_ROOT_SCOPE, APP_SCOPE, scopeName)
                .installModules(DependenciesModule(positionDependencies))
    }

    fun destroyComponent() {
        Toothpick.closeScope(scopeName)
    }

    val api: FeatureApi = FeatureApiIml(scopeName)
}
```

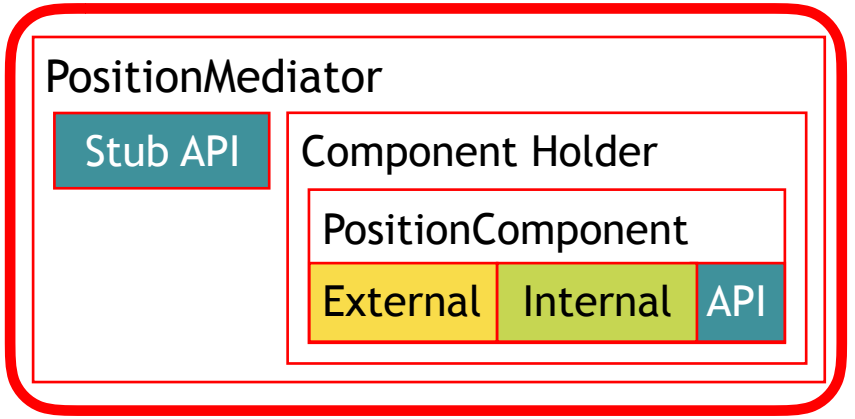
Шаг 5. Отпочкование фичей



Переиспользование

```
class PositionScreenMediator() {

    private val componentHolder: MultiComponentHolder<PositionComponent, PositionDependencies>

      = MultiComponentHolder { deps -> PositionComponent(deps) }

    fun initComponent(key: String, deps: PositionDependencies) {
        componentHolder.initComponent(key, deps)
    }

    fun provideApiStub(key: String): FeatureApi {
        return componentHolder.provideComponent(key).api
    }

    // ...
}
```

Шаг 5. Отпочкование фичей

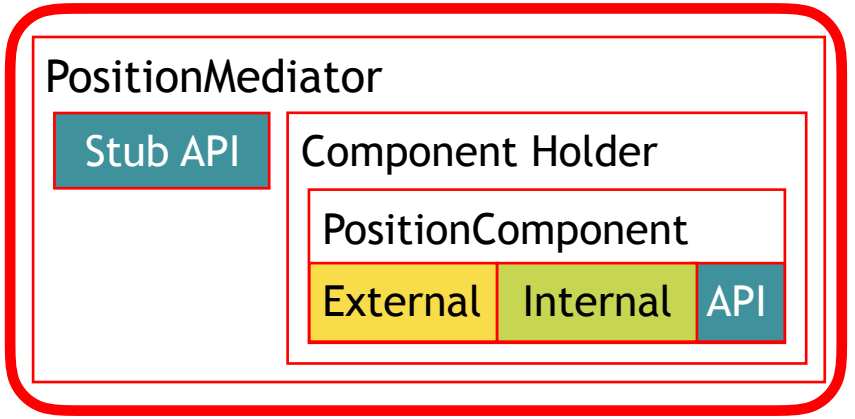


Переиспользование

```
class PositionScreenMediator() {

    private val componentHolder: MultiComponentHolder<PositionComponent, PositionDependencies>

      = MultiComponentHolder { deps -> PositionComponent(deps) }

    fun initComponent(key: String, deps: PositionDependencies) {
        componentHolder.initComponent(key, deps)
    }

    fun provideApiStub(key: String): FeatureApi {
        return componentHolder.provideComponent(key).api
    }

    // ...
}
```

Шаг 5. Отпочкование фичей

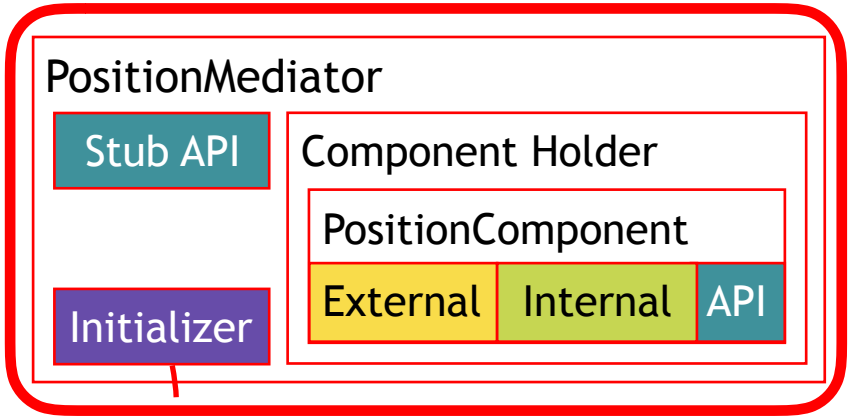
Переиспользование




```
class PositionScreenMediator() {

    private val componentHolder: MultiComponentHolder<PositionComponent, PositionDependencies>

      = MultiComponentHolder { deps -> PositionComponent(deps) }

    fun initComponent(key: String, deps: PositionDependencies) {
        componentHolder.initComponent(key, deps)
    }

    fun provideApiStub(key: String): FeatureApi {
        return componentHolder.provideComponent(key).api
    }

    // ...
}
```

Шаг 5. Отпочкование фичей

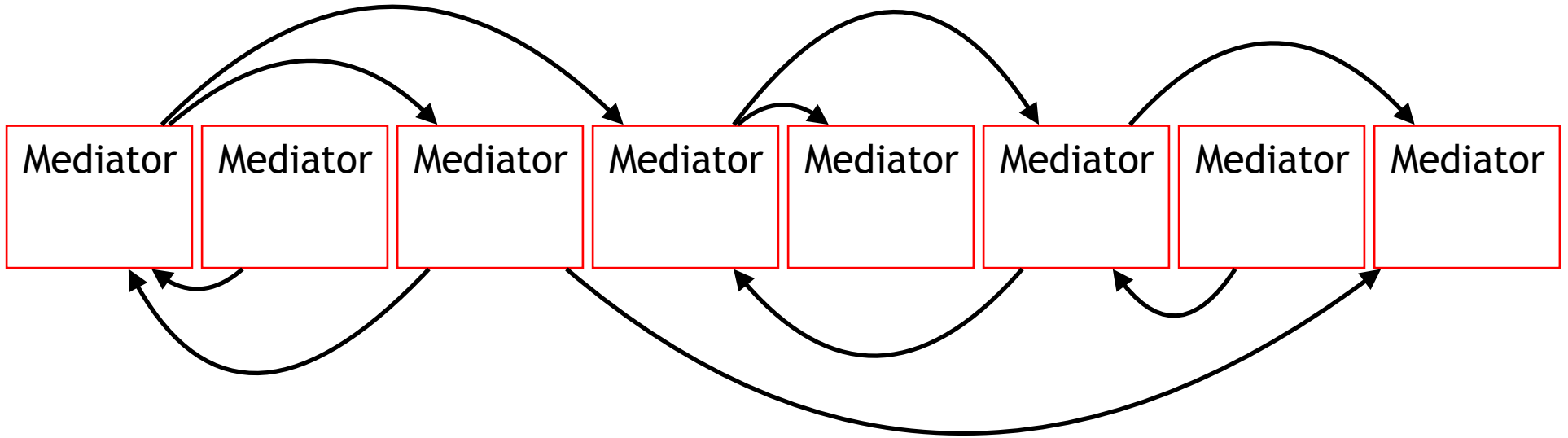
Переиспользование




```
class PositionScreenMediator() {

    private val componentHolder: MultiComponentHolder<PositionComponent, PositionDependencies>

      = MultiComponentHolder { deps -> PositionComponent(deps) }

    fun initComponent(key: String, deps: PositionDependencies) {
        componentHolder.initComponent(key, deps)
    }

    fun provideApiStub(key: String): FeatureApi {
        return componentHolder.provideComponent(key).api
    }

    // ...
}
```

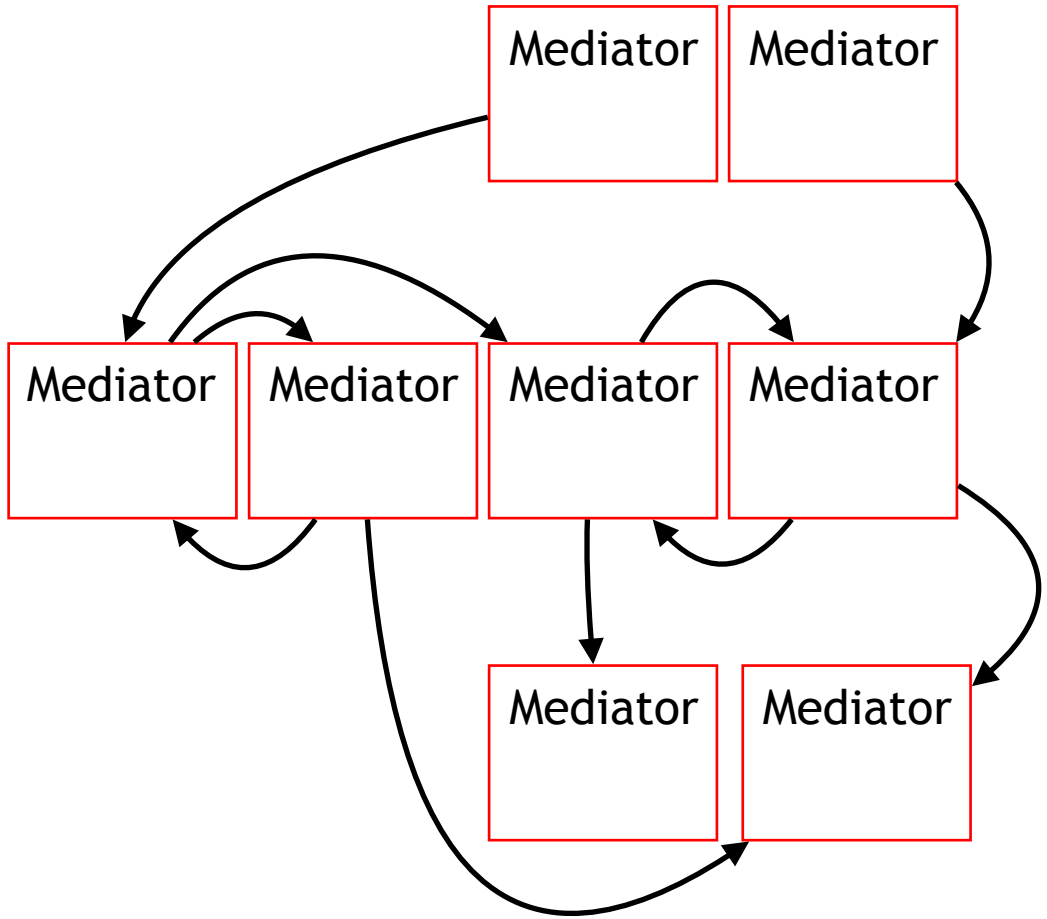
Шаг 6. Формирование медиаторов

Зависимости



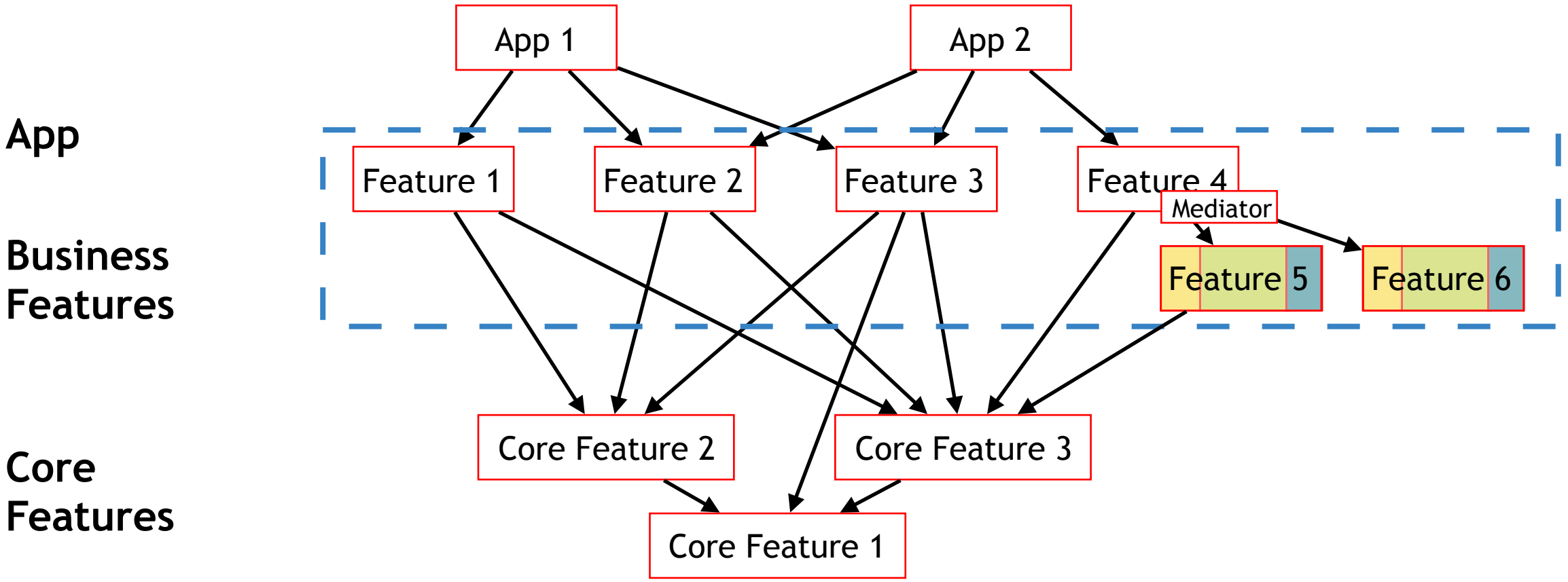
Шаг 6. Формирование медиаторов

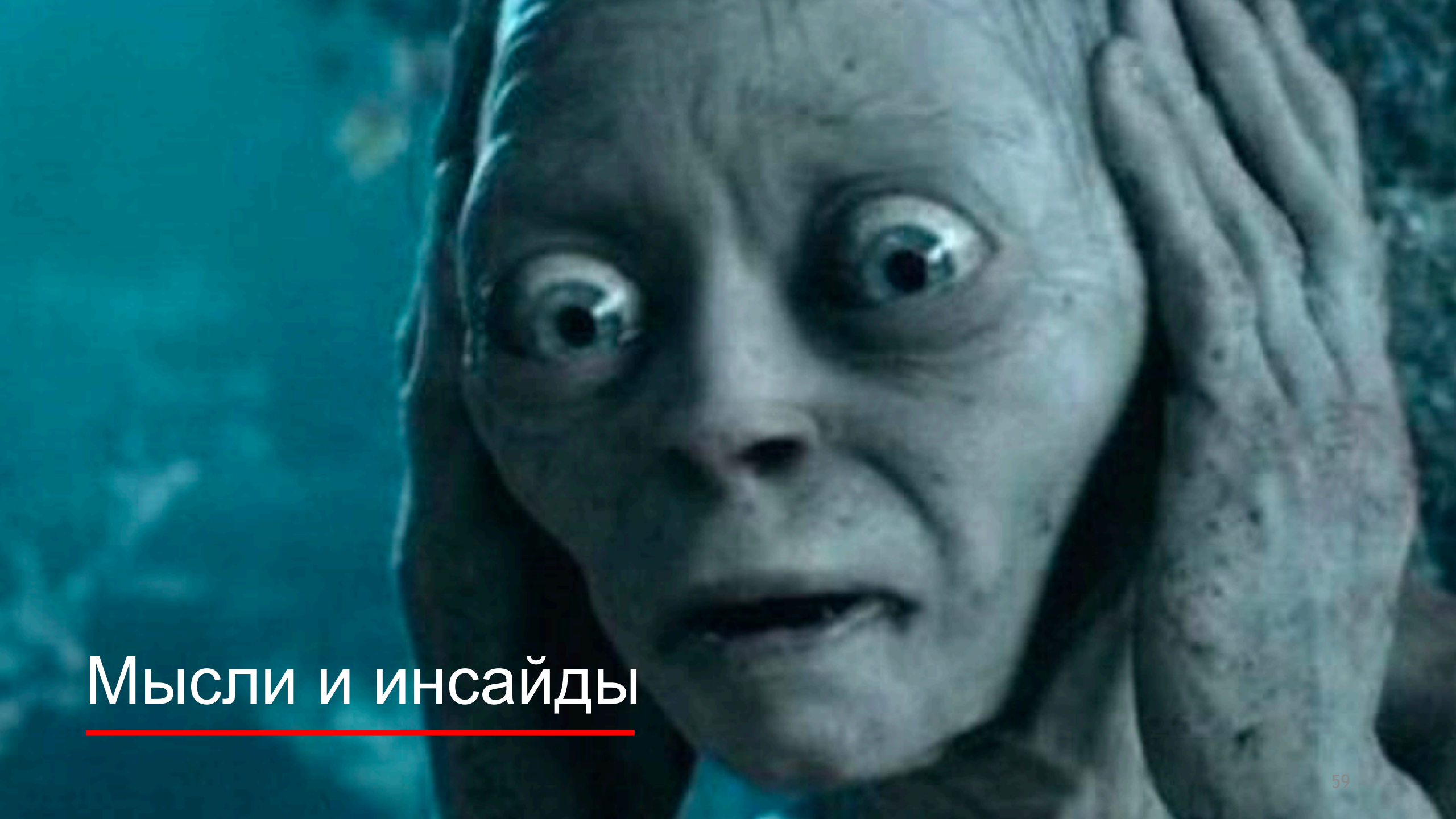
Зависимости (Возможная картина)



Шаг 6. Формирование медиаторов

Возможная (!) иерархия фичей





Мысли и инсайды

Мысли и инсайды

Проведение экспериментов на Console проекте

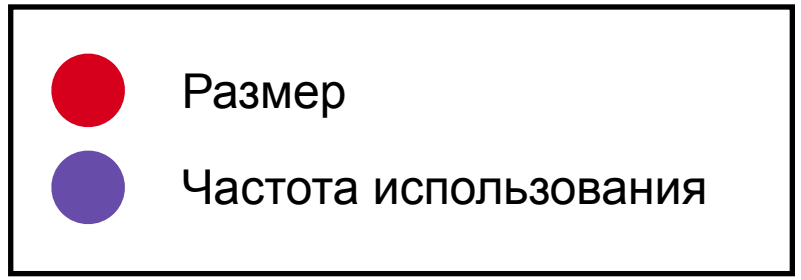
Описание архитектуры в вики / статье для понимания

Создание тестовых App модулей для фичи

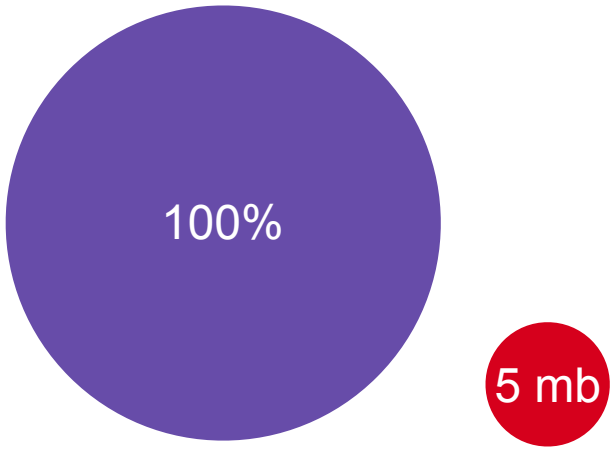
Время сборки и борьба с IDE

Динамические фиче-модули

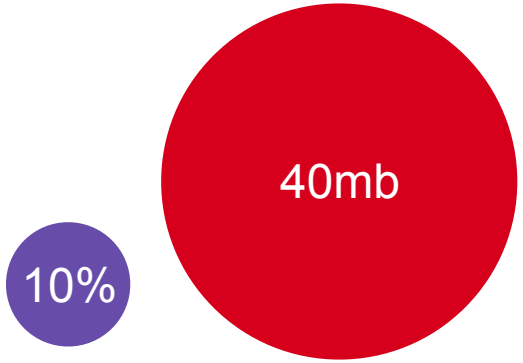
Причины использования



Основной функционал



VR модуль

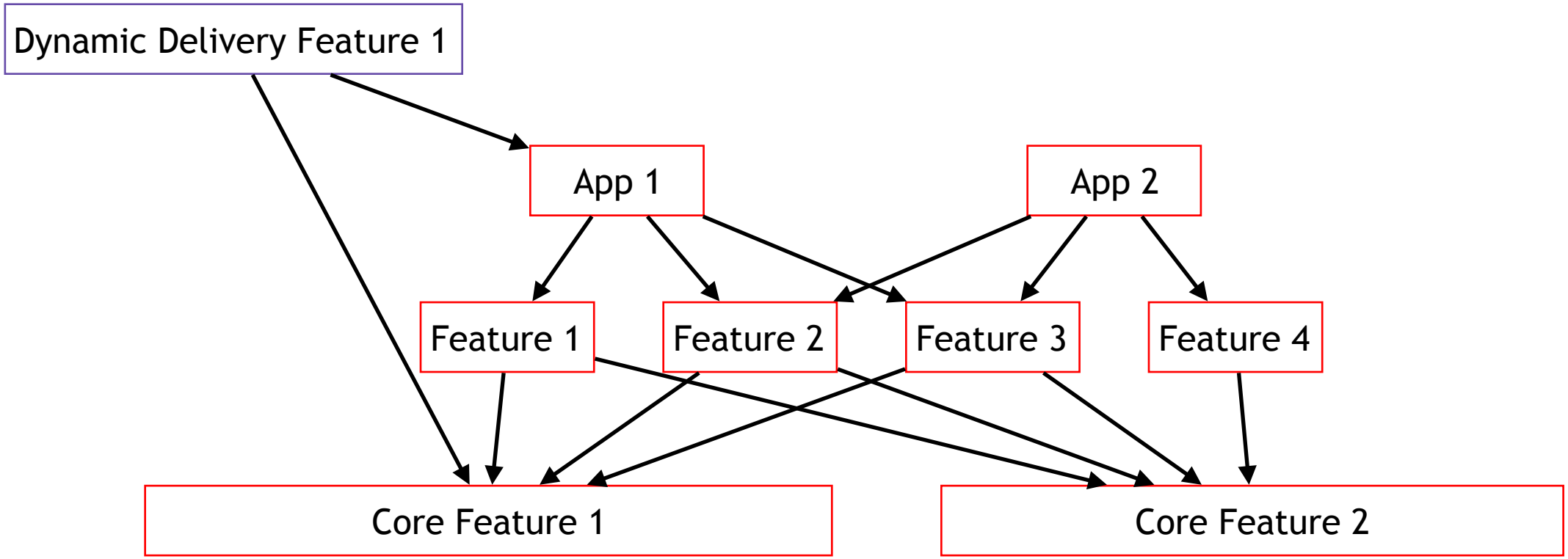


https://developer.android.com/guide/app-bundle/

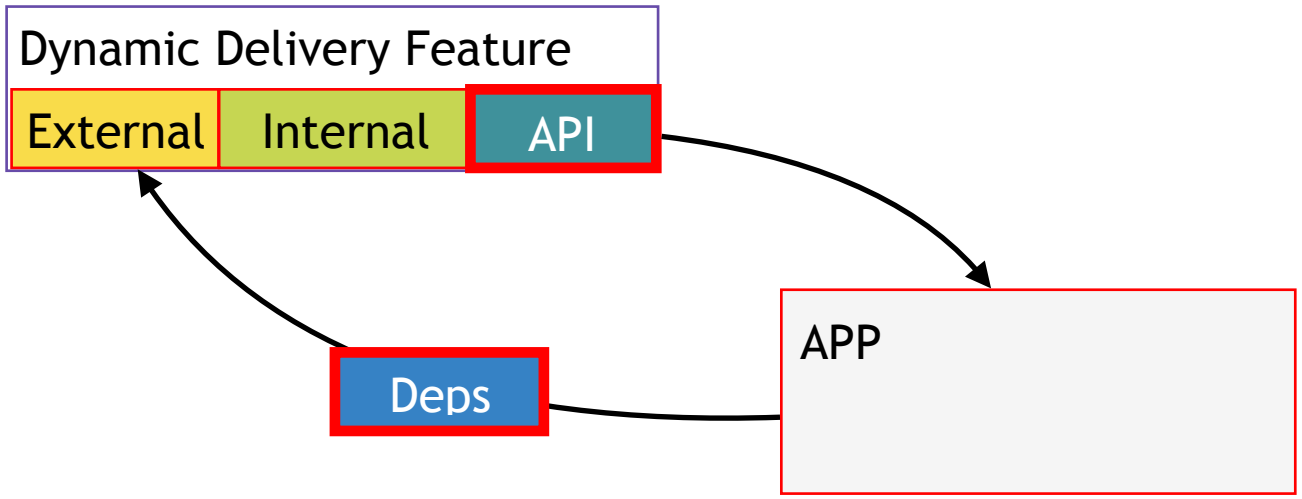
https://developer.android.com/studio/projects/dynamic-delivery

Иерархия

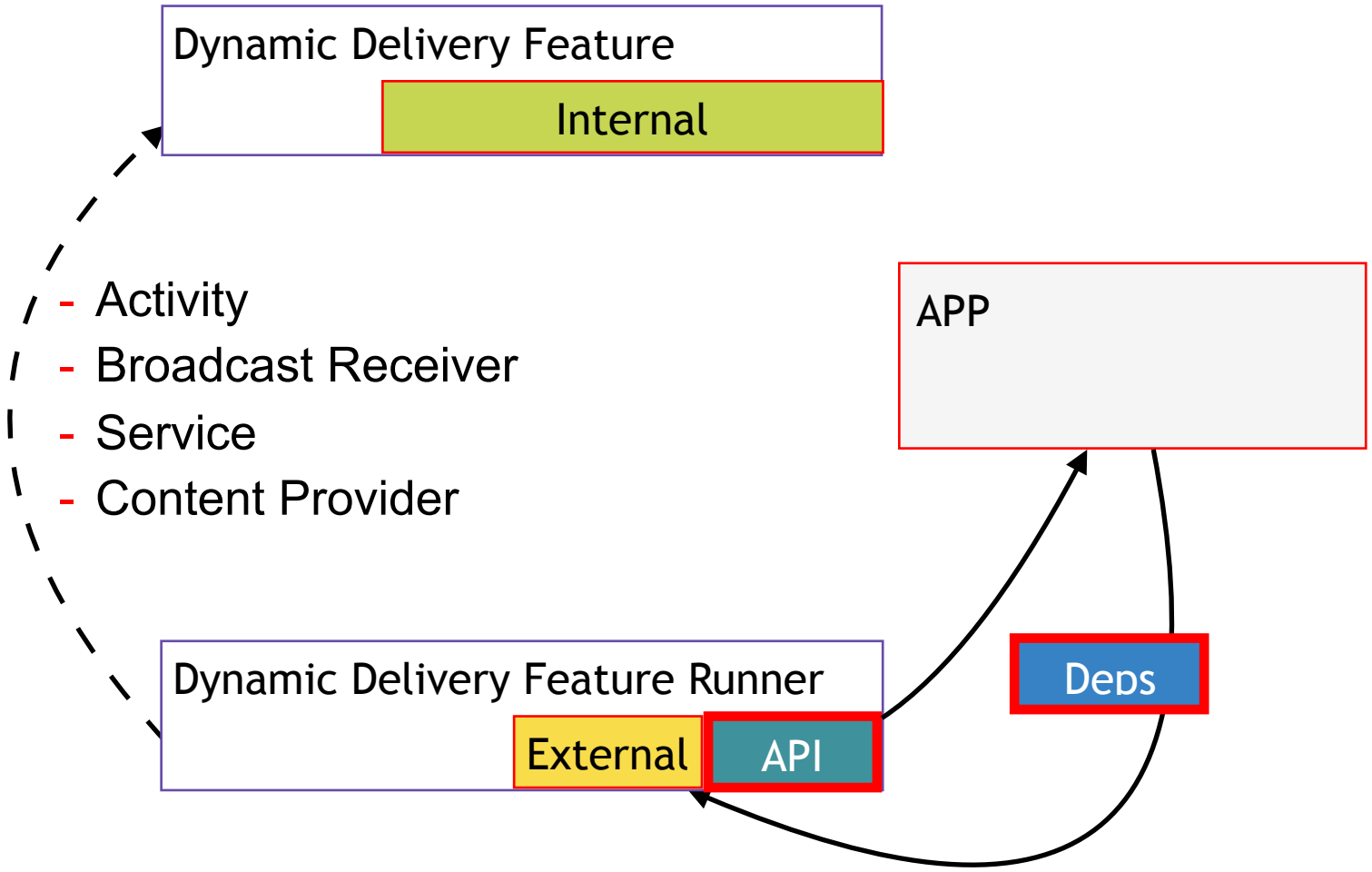
Всё пропало, шеф



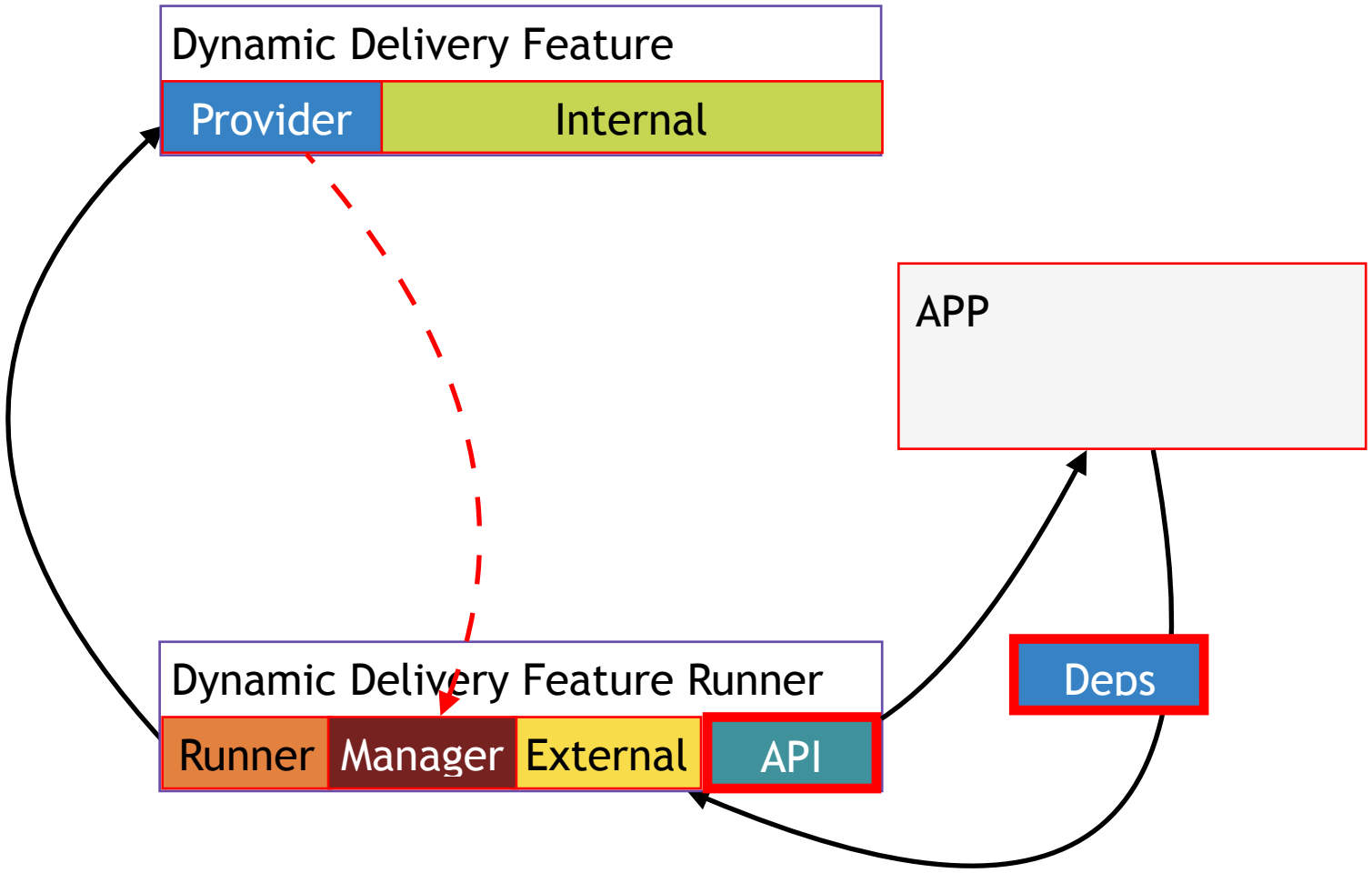
Взаимодействие dynamic feature



Взаимодействие dynamic feature

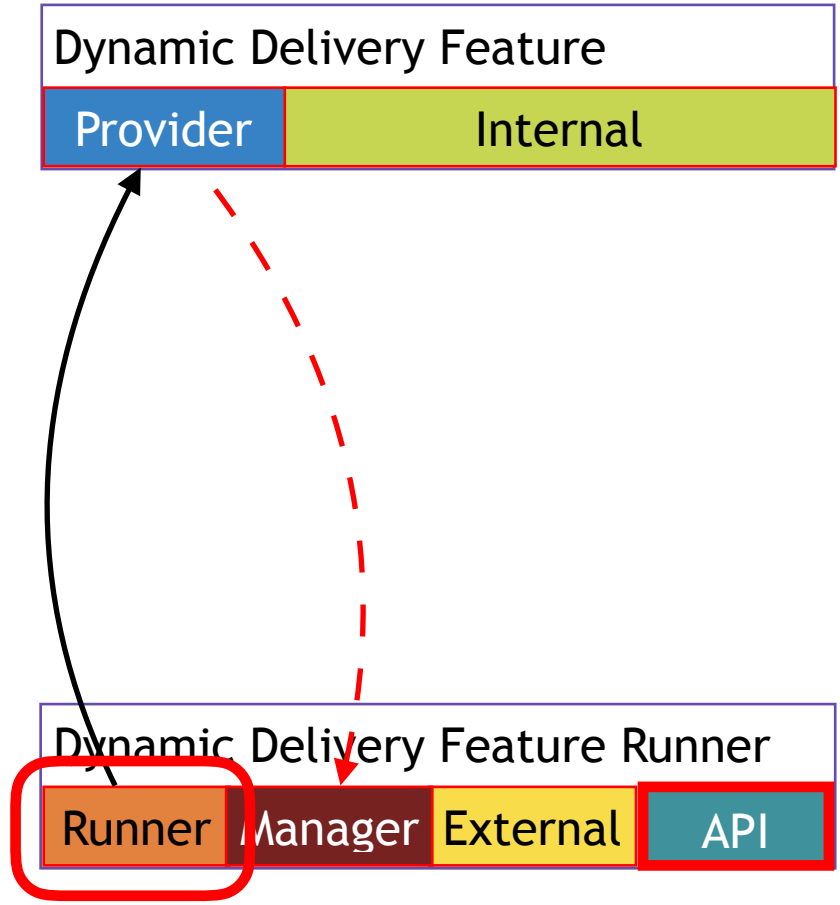


Взаимодействие dynamic feature



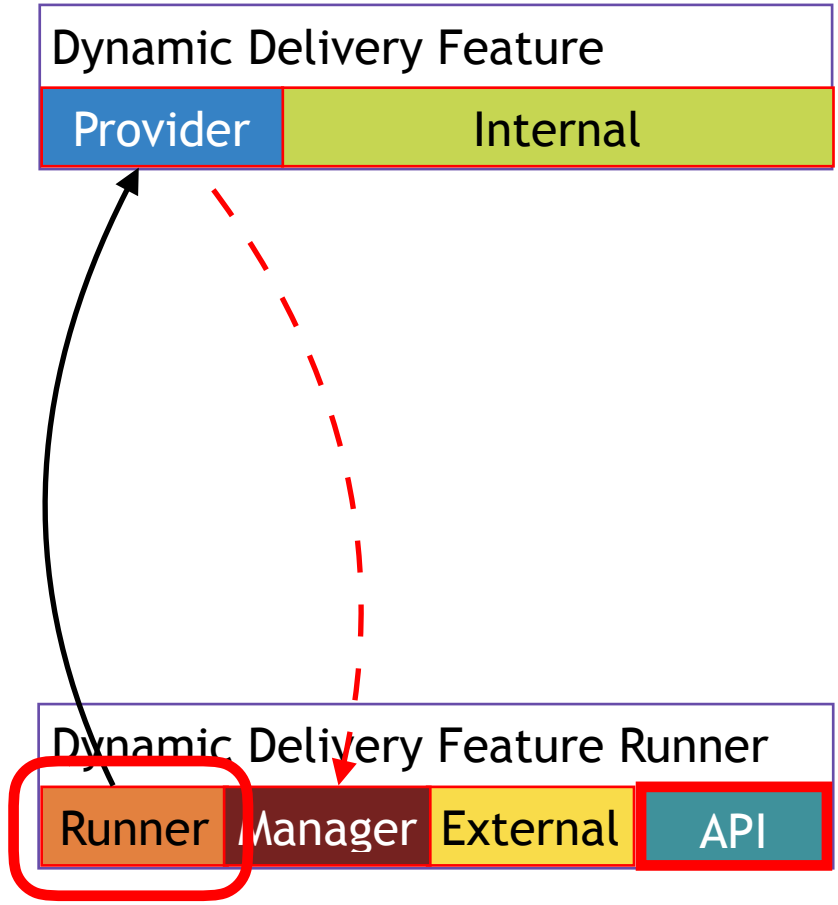
Загрузка модулей

```
class SplitFeatureRunner(private val context: Context) {

    private val splitInstallManager: SplitInstallManager =
            SplitInstallManagerFactory.create(context).also {

                it.registerListener { status -> doSomething(status) }
            }


    fun installModule() {

        val request =
            SplitInstallRequest
                .newBuilder()
                .addModule("dynamic_feature_vr")
                .build();

        splitInstallManager
            .startInstall(request)
            .addOnSuccessListener { sessionId ->
              Intent().setClassName(packageName, serviceClassName)
                .also { startService(it) }
            }
            .addOnFailureListener { exception ->
               // something on fail
            }

    }
}
```



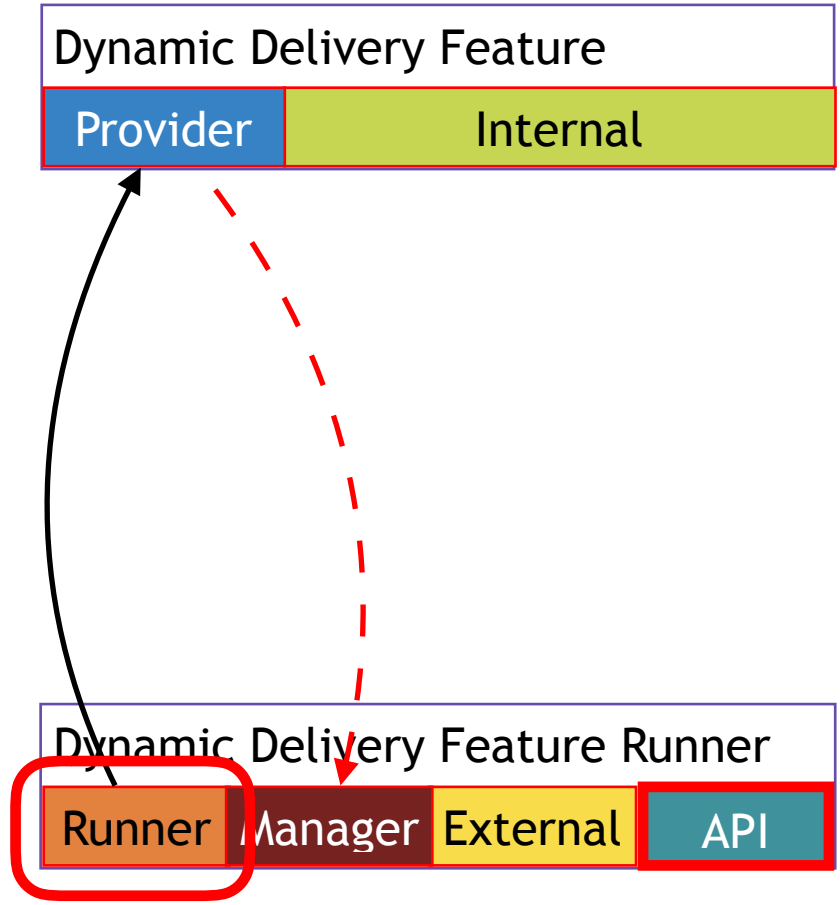
Загрузка модулей

```
class SplitFeatureRunner(private val context: Context) {

    private val splitInstallManager: SplitInstallManager =
            SplitInstallManagerFactory.create(context).also {

                it.registerListener { status -> doSomething(status) }
            }


    fun installModule() {

        val request =
            SplitInstallRequest
                .newBuilder()
                .addModule("dynamic_feature_vr")
                .build();

        splitInstallManager
            .startInstall(request)
            .addOnSuccessListener { sessionId ->
              Intent().setClassName(packageName, serviceClassName)
                .also { startService(it) }
            }
            .addOnFailureListener { exception ->
               // something on fail
            }

    }
}
```



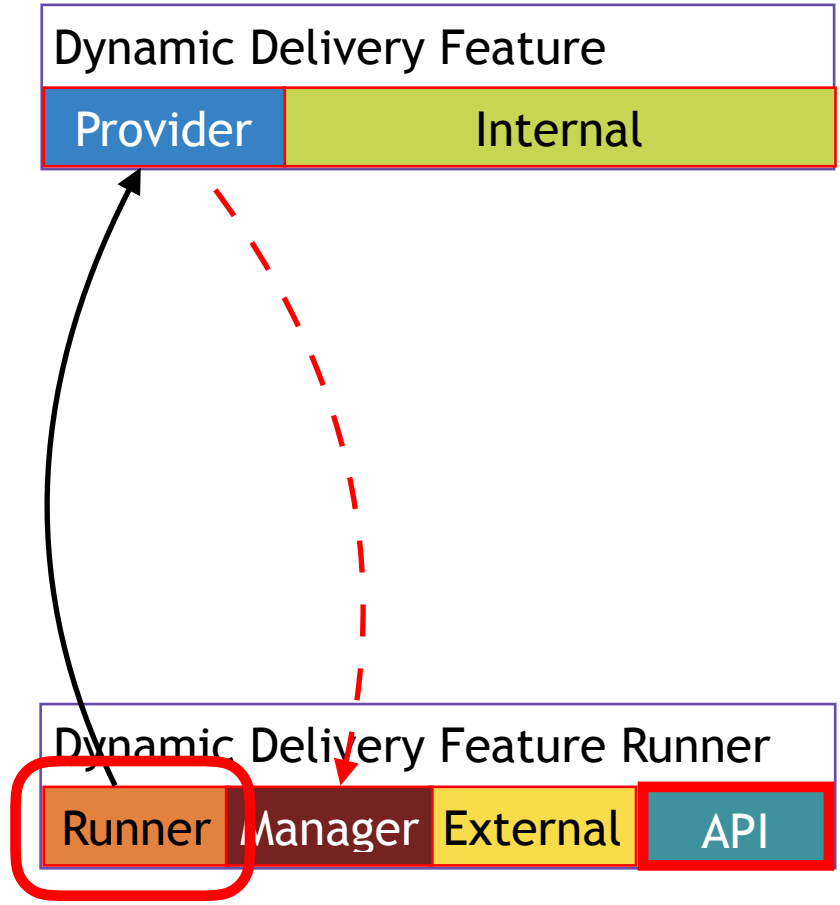
Загрузка модулей

```
class SplitFeatureRunner(private val context: Context) {

    private val splitInstallManager: SplitInstallManager =
            SplitInstallManagerFactory.create(context).also {

                it.registerListener { status -> doSomething(status) }
            }


    fun installModule() {

        val request =
            SplitInstallRequest
                .newBuilder()
                .addModule("dynamic_feature_vr")
                .build();

        splitInstallManager
            .startInstall(request)
            .addOnSuccessListener { sessionId ->
               Intent().setClassName(packageName, serviceClassName)
                .also { sartService(it) }
            }
            .addOnFailureListener { exception ->
               // something on fail
            }

    }
}
```



Загрузка модулей

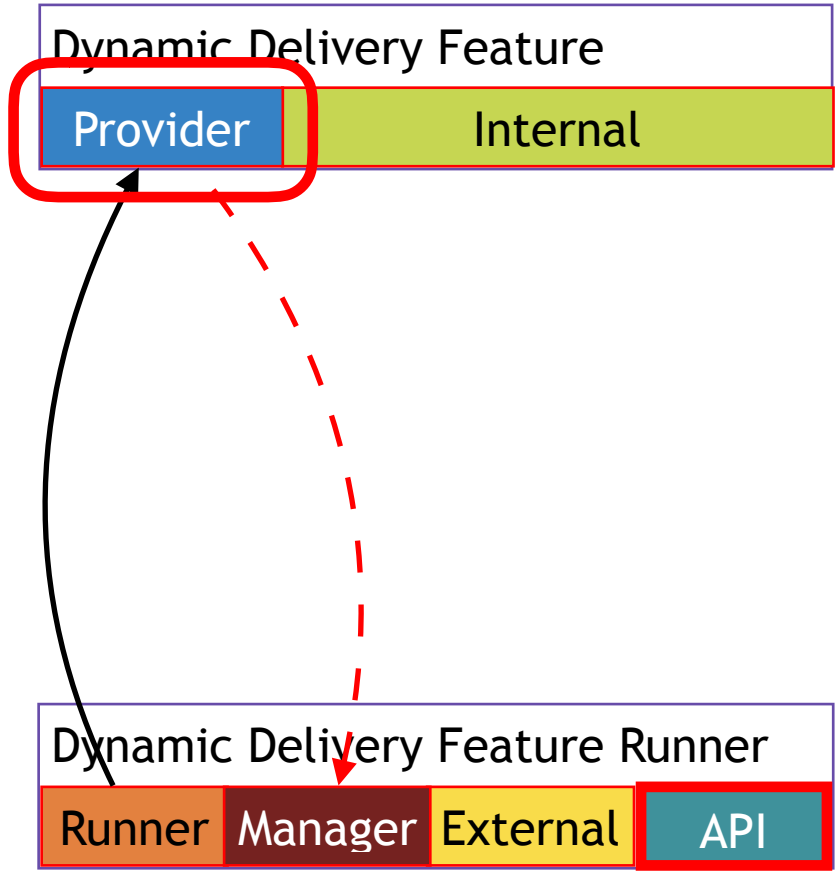
```
class SplitFeatureRunner(private val context: Context) {

    private val splitInstallManager: SplitInstallManager =
            SplitInstallManagerFactory.create(context).also {

                it.registerListener { status -> doSomething(status) }
            }


    fun installModule() {

        val request =
            SplitInstallRequest
                .newBuilder()
                .addModule("dynamic_feature_vr")
                .build();

        splitInstallManager
            .startInstall(request)
            .addOnSuccessListener { sessionId ->
               Intent().setClassName(packageName, serviceClassName)
                 .also { startService(it) }
            }
            .addOnFailureListener { exception ->
               // something on fail
            }

    }
}
```



Взаимодействие dynamic feature

```
open class LaunchModuleService : IntentService("LaunchModuleService") {

    override fun onHandleIntent(intent: Intent?) {

        DynamicFeatureModuleManager.setLauncher(DynamicFeatureLauncherImpl())
    }

}
```

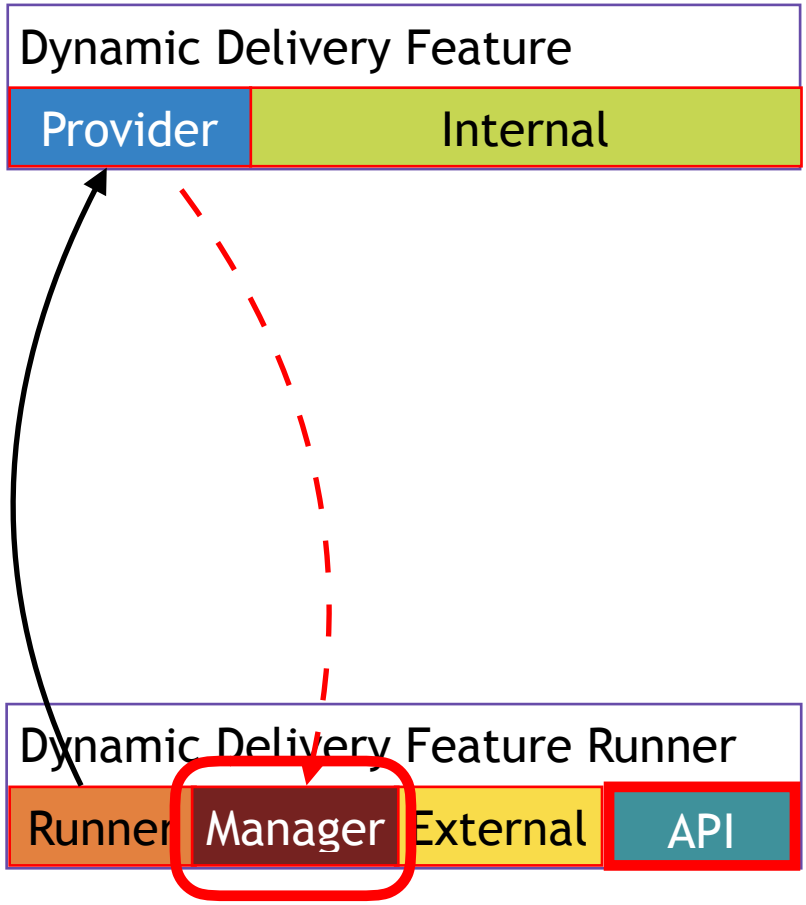
```
object DynamicFeatureModuleManager {

    private var listenerSet: MutableSet<CheckInstallListener> = mutableSetOf()

    fun setLauncher(launcher: DynamicFeatureLauncher) {
        for (listener in listenerSet) {
            listener.onCheckInstalled(launcher)                    }
        }
    }

    fun addListener(checkInstallListener: CheckInstallListener) {
        listenerSet.add(checkInstallListener)
    }
     // . . .
}
```



Взаимодействие dynamic feature

```
open class LaunchModuleService : IntentService("LaunchModuleService") {

    override fun onHandleIntent(intent: Intent?) {

        DynamicFeatureModuleManager.setLauncher(DynamicFeatureLauncherImpl())
    }

}
```

```
object DynamicFeatureModuleManager {

    private var listenerSet: MutableSet<CheckInstallListener> = mutableSetOf()

    fun setLauncher(launcher: DynamicFeatureLauncher) {
        for (listener in listenerSet) {
            listener.onCheckInstalled(launcher)                    }
        }
    }

    fun addListener(checkInstallListener: CheckInstallListener) {
        listenerSet.add(checkInstallListener)
    }
     // . . .
}
```



Ограничения

На сайте Android Developers

1. Android App Bundles
2. Не поддерживает “APK expansion files” и APK > 100 mb
3. Конфликтует с инструментами меняющими таблицу ресурсов
4. Для работы необходимы Android 5.0 (API level 21) и последняя версия приложения Play Store
5. Прочие баги, ограничения и недоработки

https://developer.android.com/guide/app-bundle/#known_issues

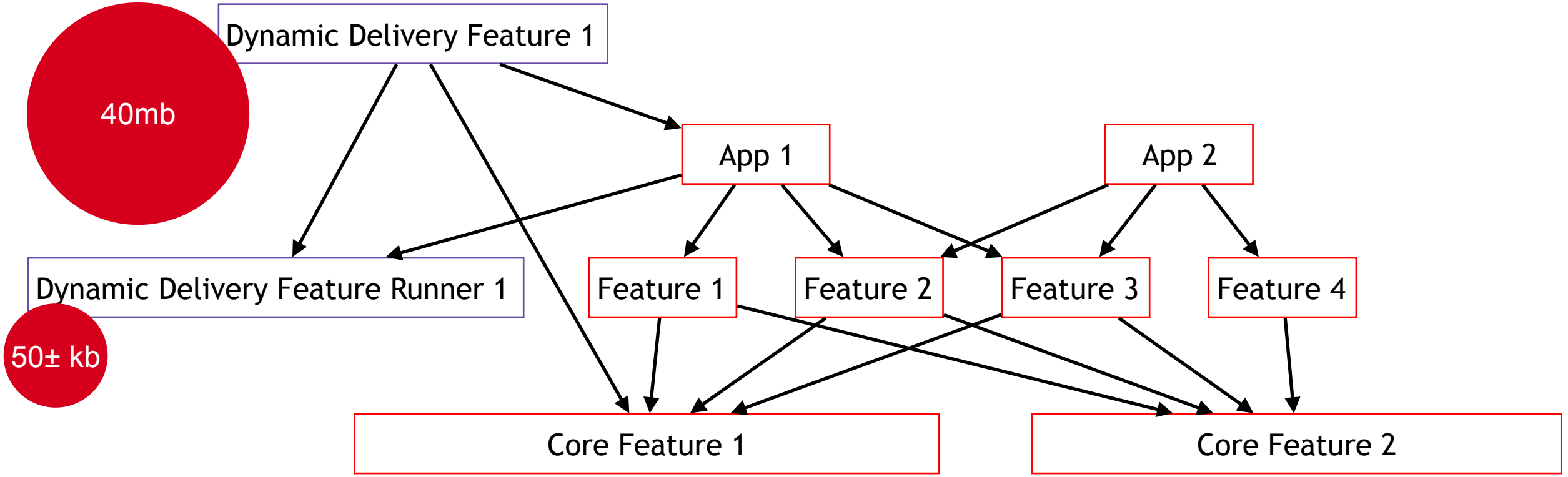
Ограничения

Что еще

1. Moxy
2. Фреймворки, которые работают аналогичным образом

Динамические фиче-модули

Выводы



Подведение итогов

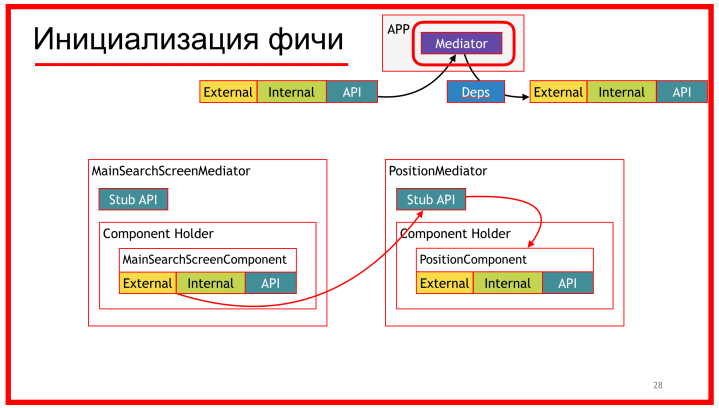
Выводы

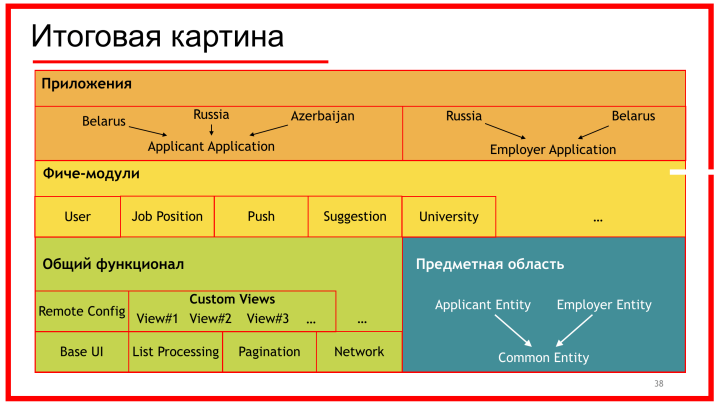
Самое сложное - выстроить связи модулей. Выстроили = Победили

Концепция сложная и подойдет для зрелых проектов

Экспериментируйте. В экспериментах рождается истина

О чем мы говорили





Обзор проблем

Картина “TO BE”

Переход от “AS IS” к “TO BE”

Динамические фиче-модули

Подведение итогов

Выводы

Самое сложное - выстроить связи модулей. Выстроили = Победили

Концепция сложная и подойдет для зрелых проектов

Экспериментируйте. В экспериментах рождается истина

Слайды

Контакты: t.me/xanderblinov