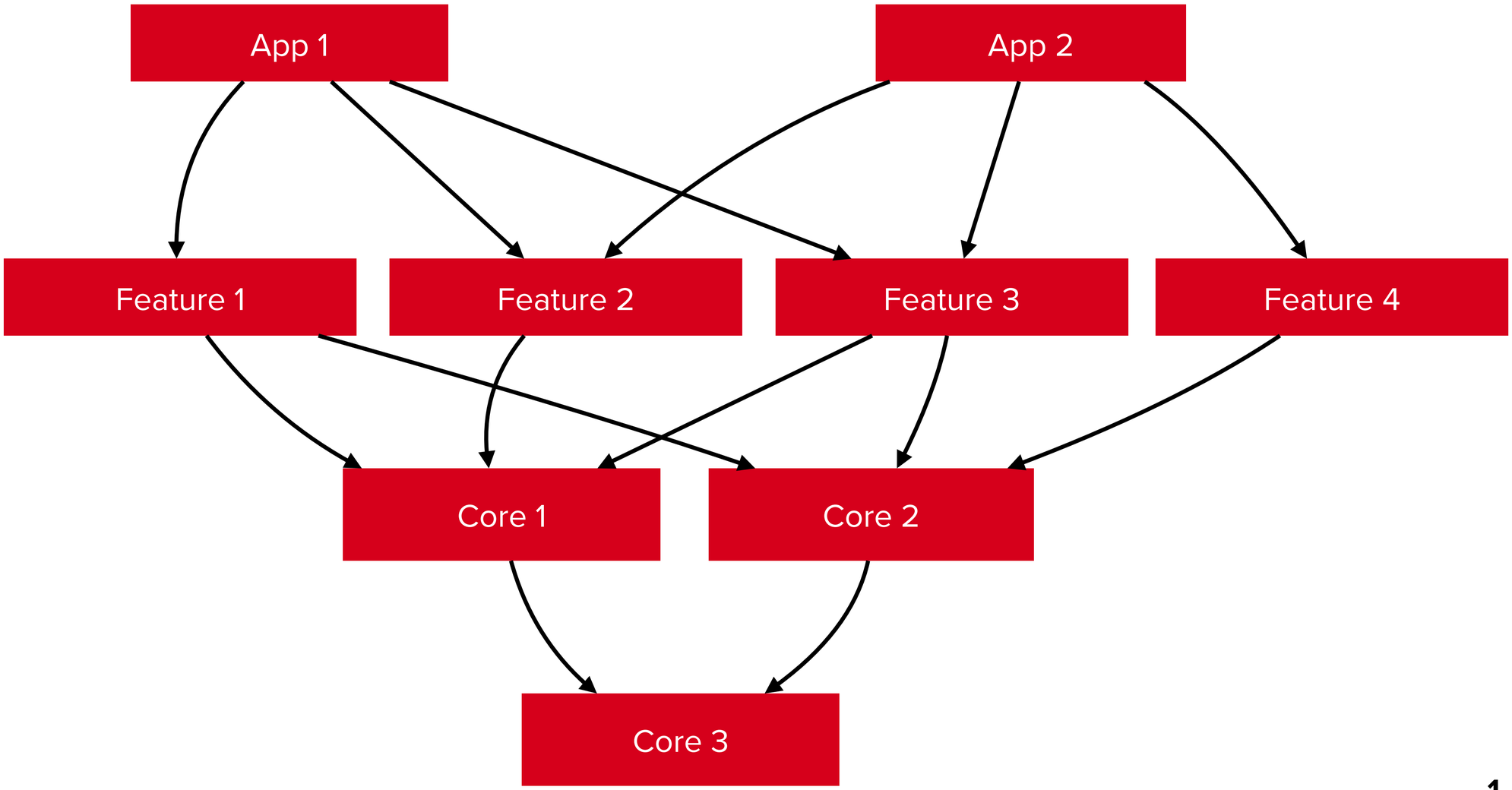
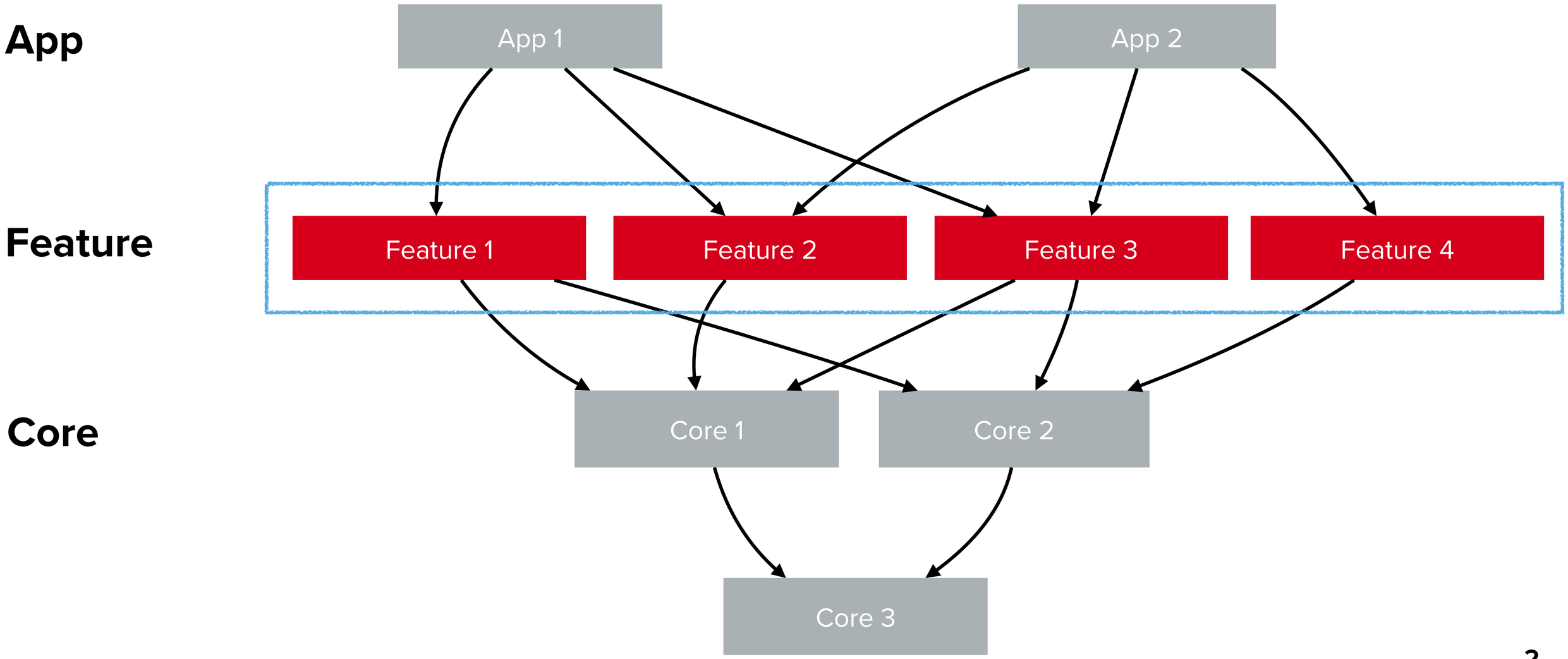
Слои модулей

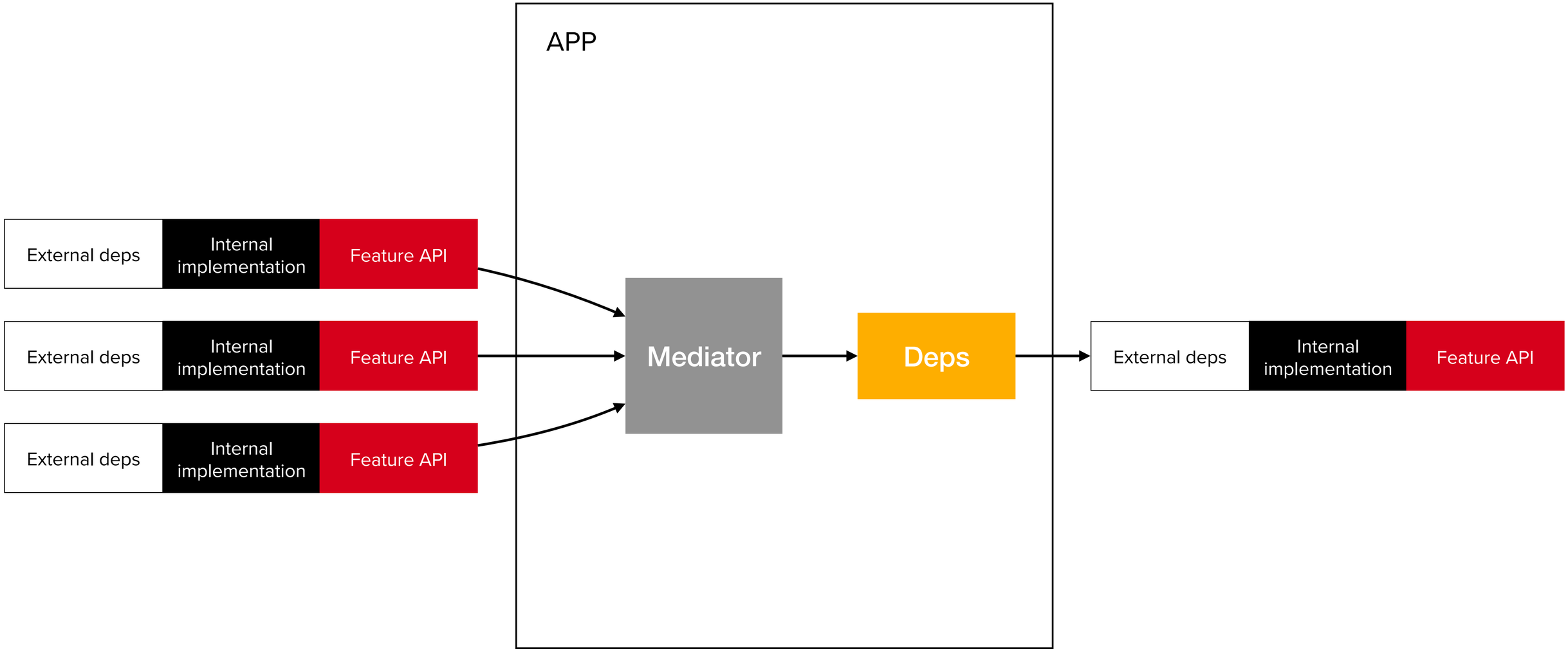


Feature-модули независимы друг от друга

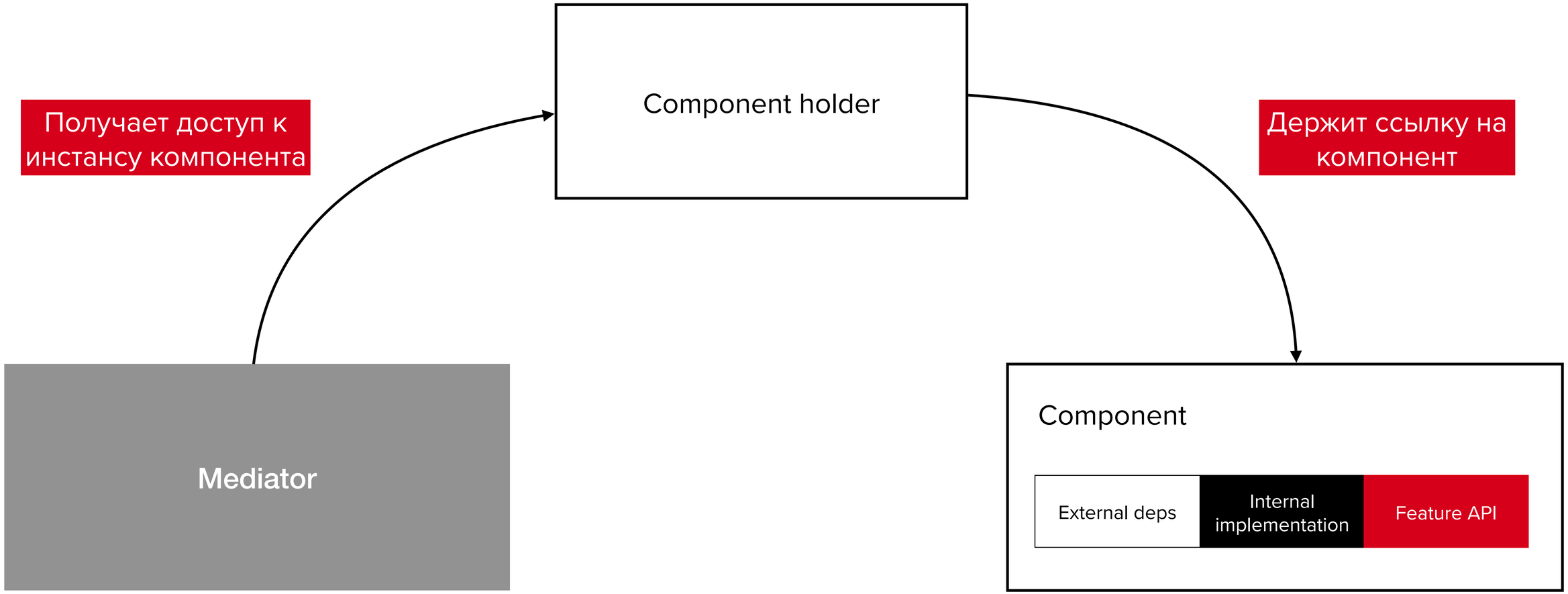


Анатомия Feature

Взаимодействие через Mediator



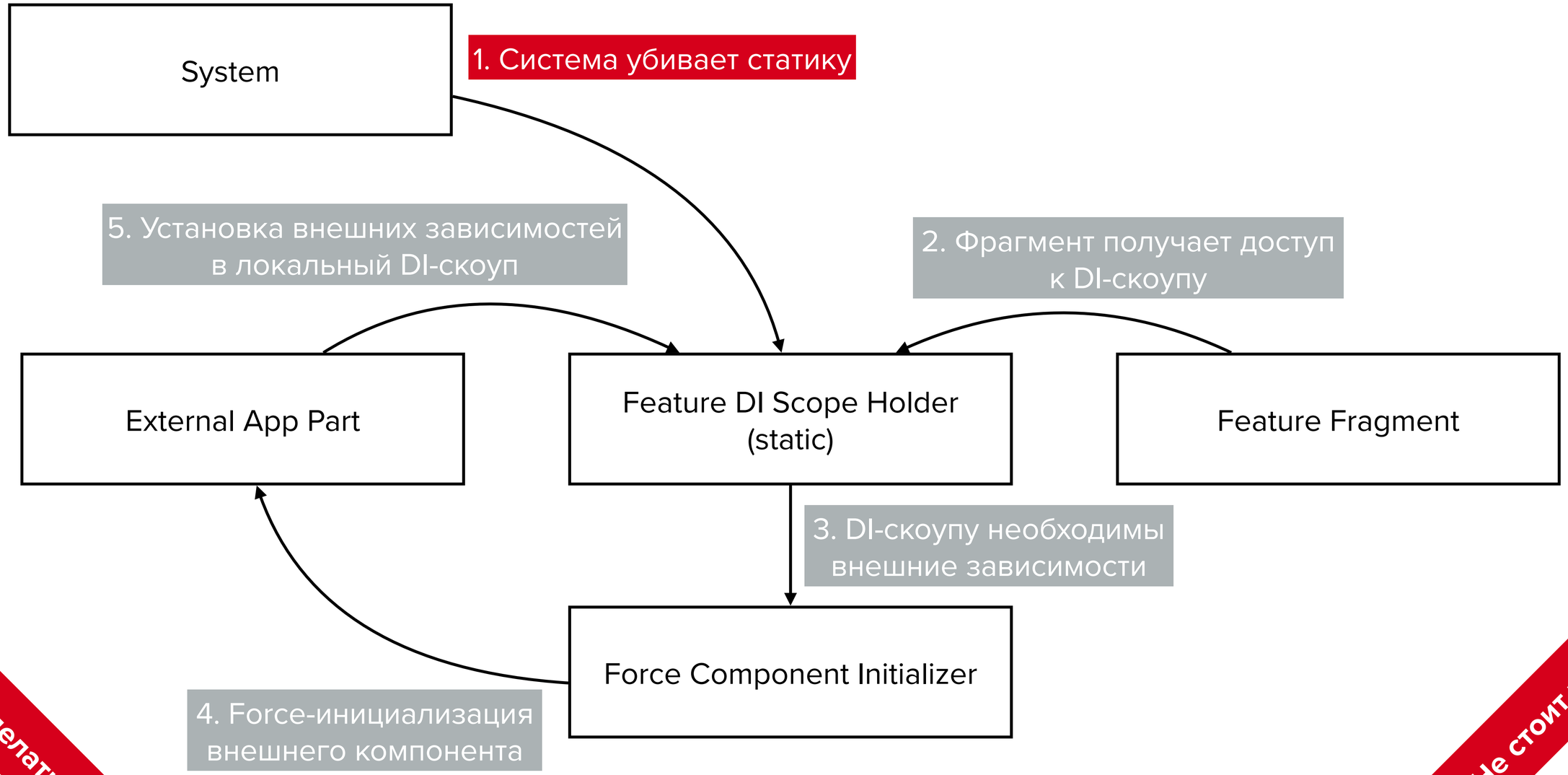
Детали реализации



Не стоит так делать

Не стоит так делать

Инициализация компонентов



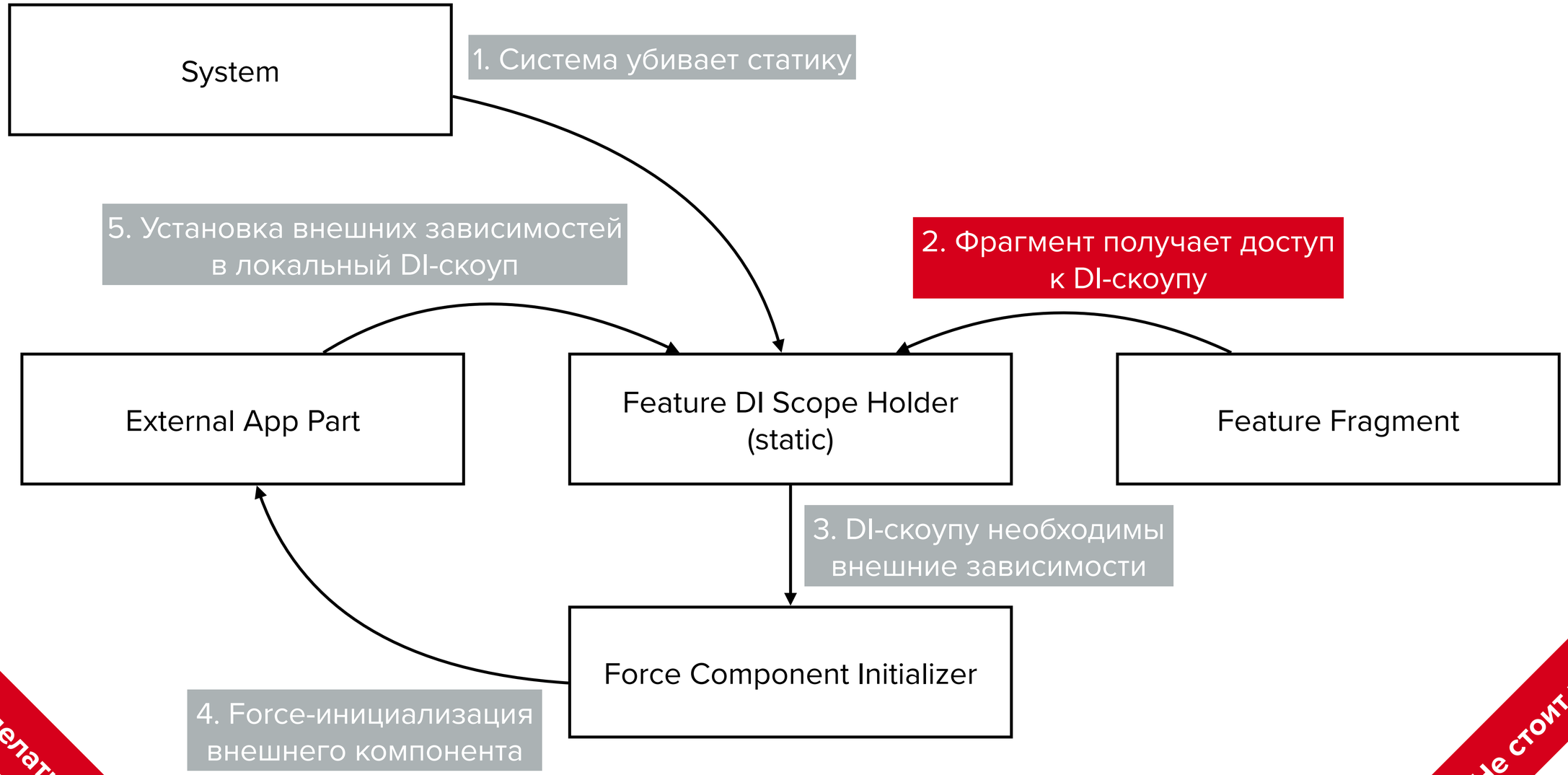
Не стоит так делать

Не стоит так делать

Не стоит так делать

Не стоит так делать

Инициализация компонентов



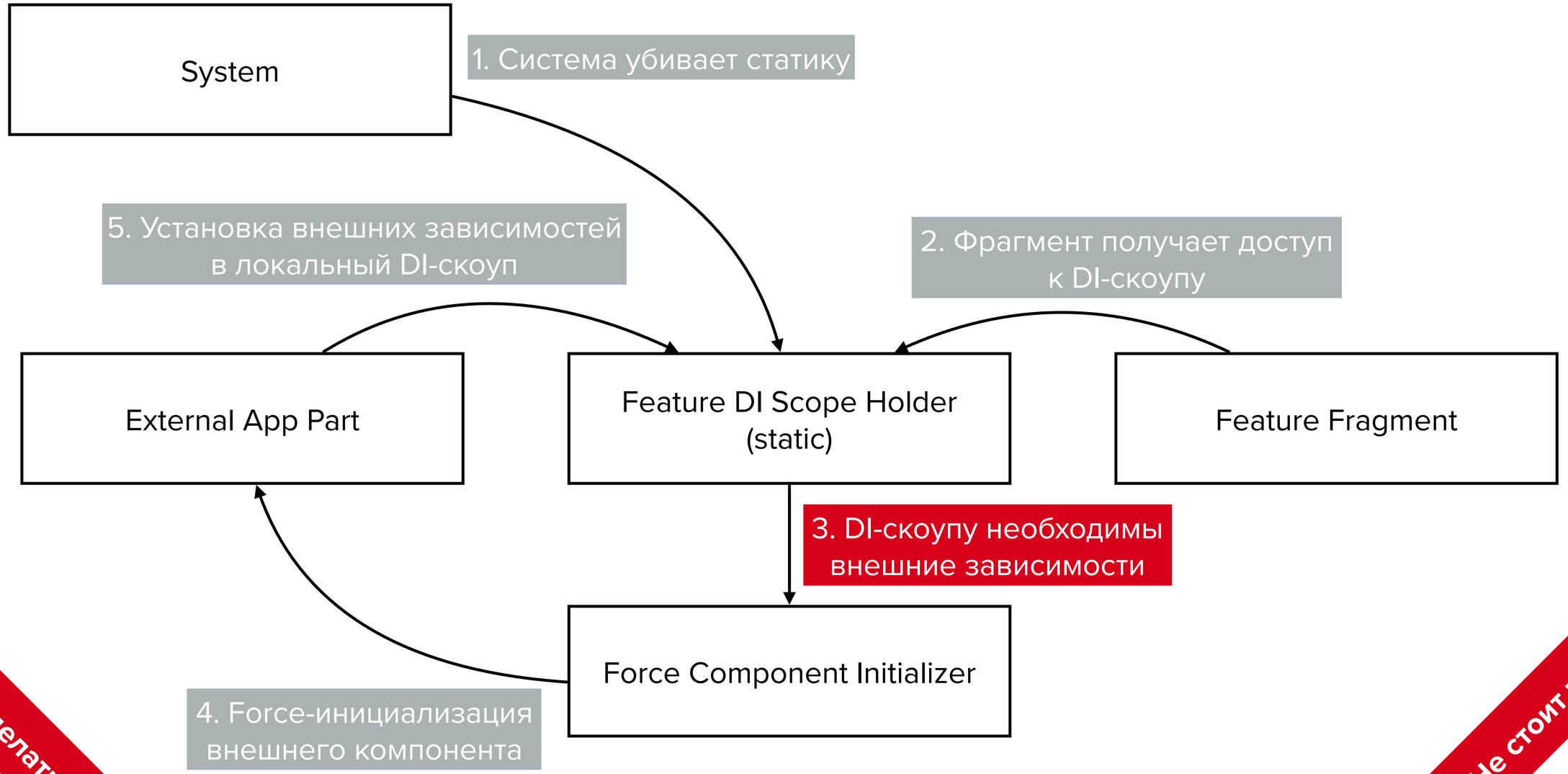
Не стоит так делать

Не стоит так делать

Не стоит так делать

Не стоит так делать

Инициализация компонентов



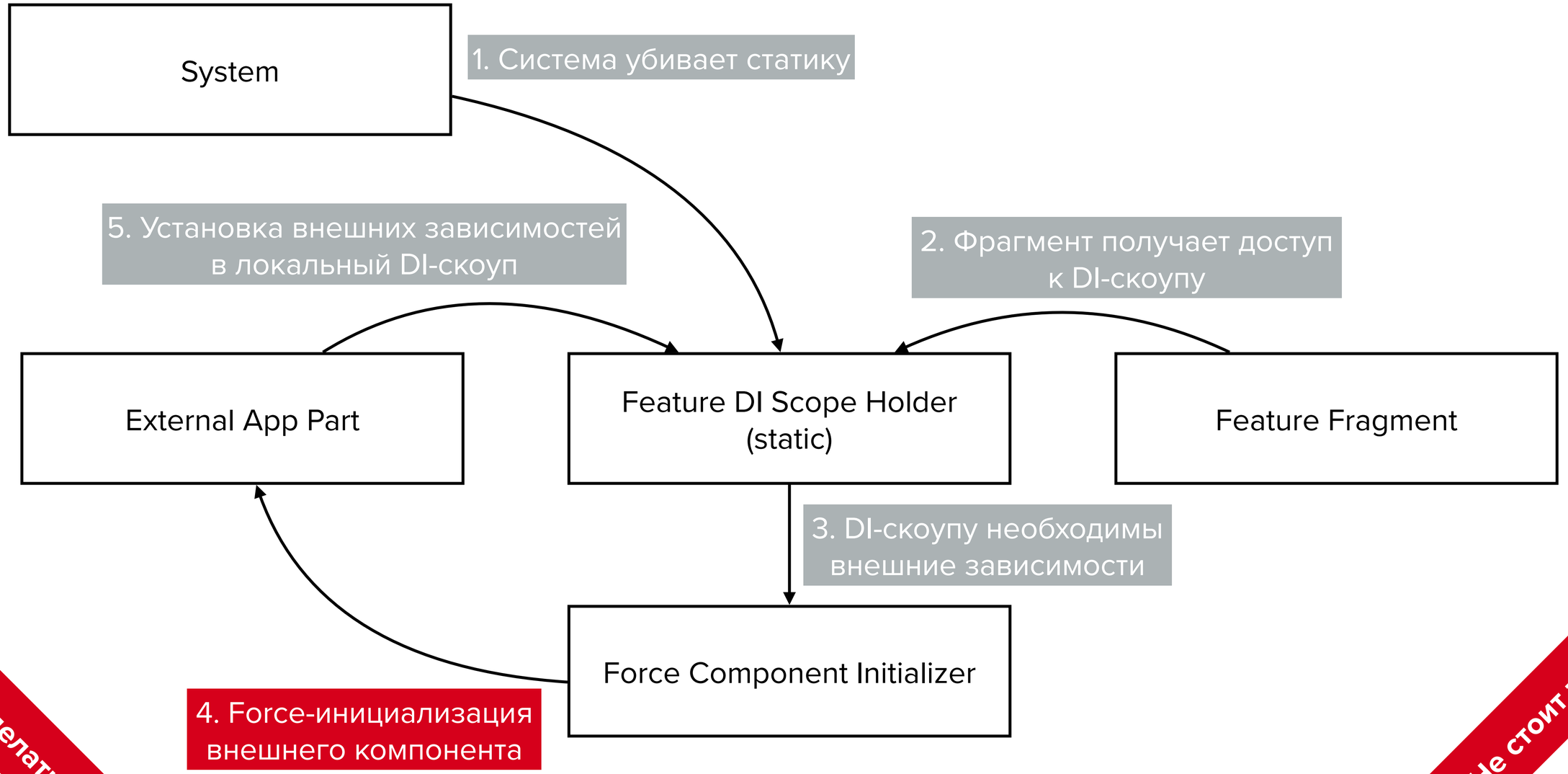
Не стоит так делать

Не стоит так делать

Не стоит так делать

Не стоит так делать

Инициализация компонентов



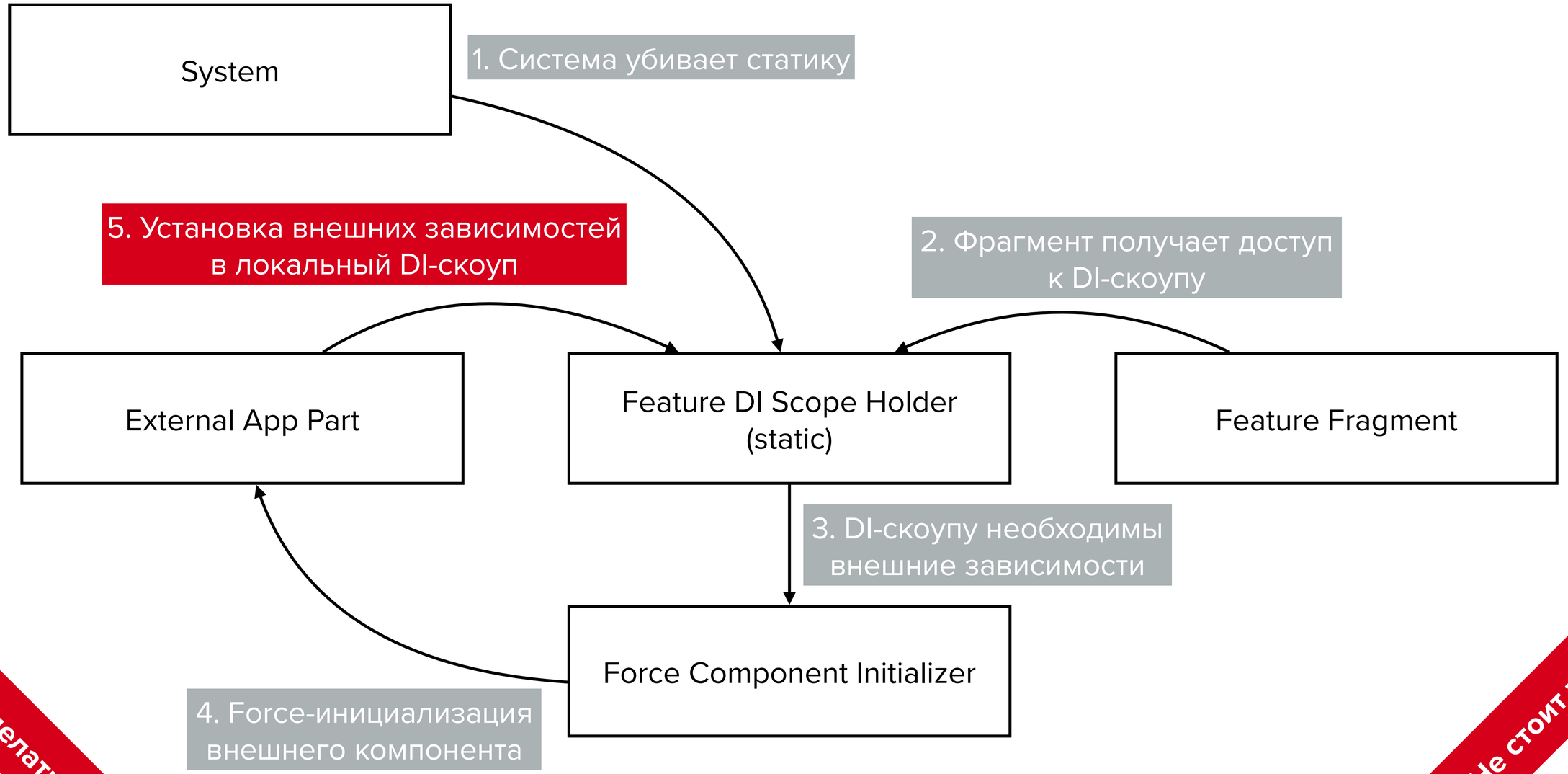
Не стоит так делать

Не стоит так делать

Не стоит так делать

Не стоит так делать

Инициализация компонентов



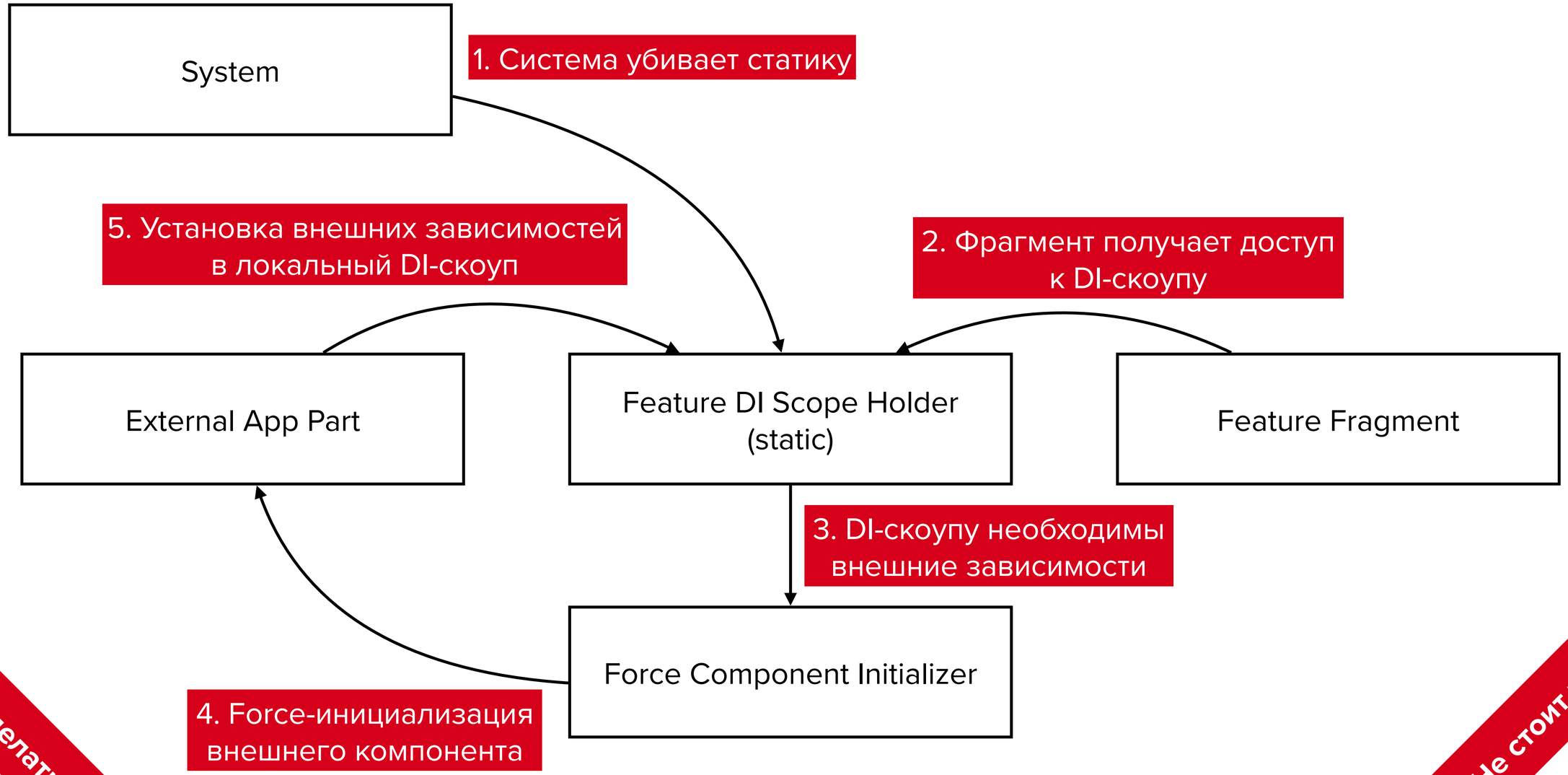
Не стоит так делать

Не стоит так делать

Не стоит так делать

Не стоит так делать

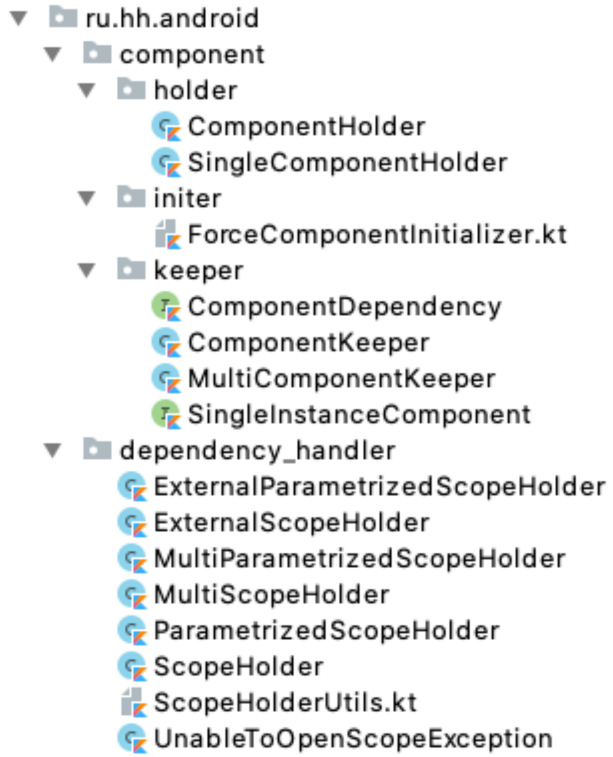
Инициализация компонентов



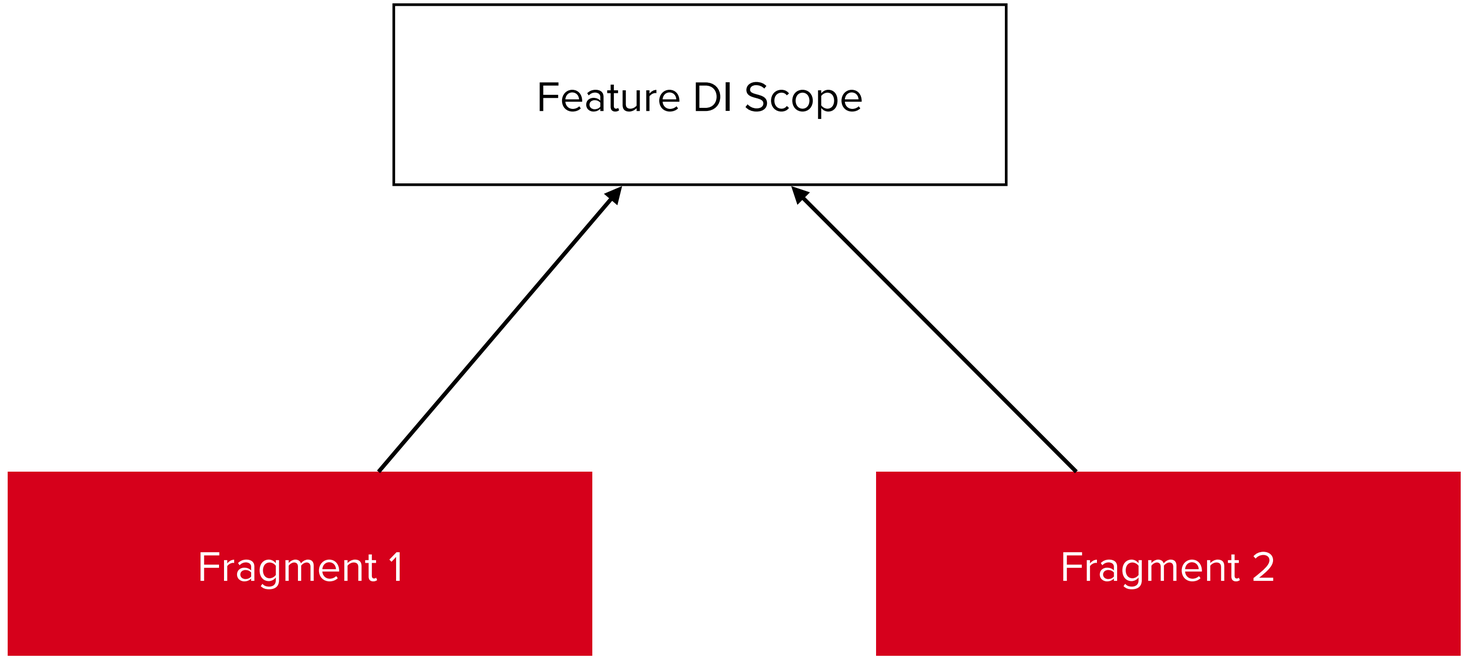
Не стоит так делать

Не стоит так делать

Сложность реализации

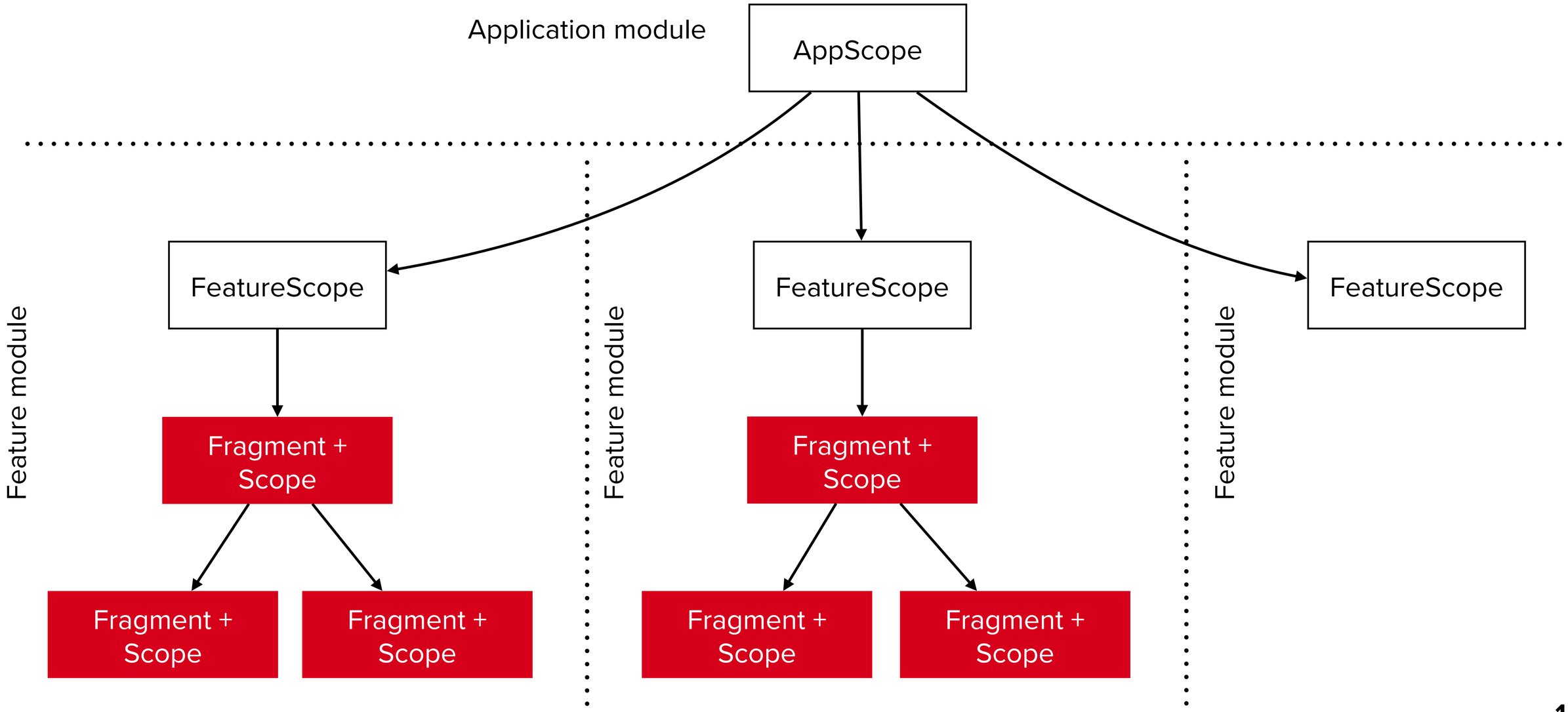


Не всегда есть прямое соответствие между фрагментами и DI-скоупами

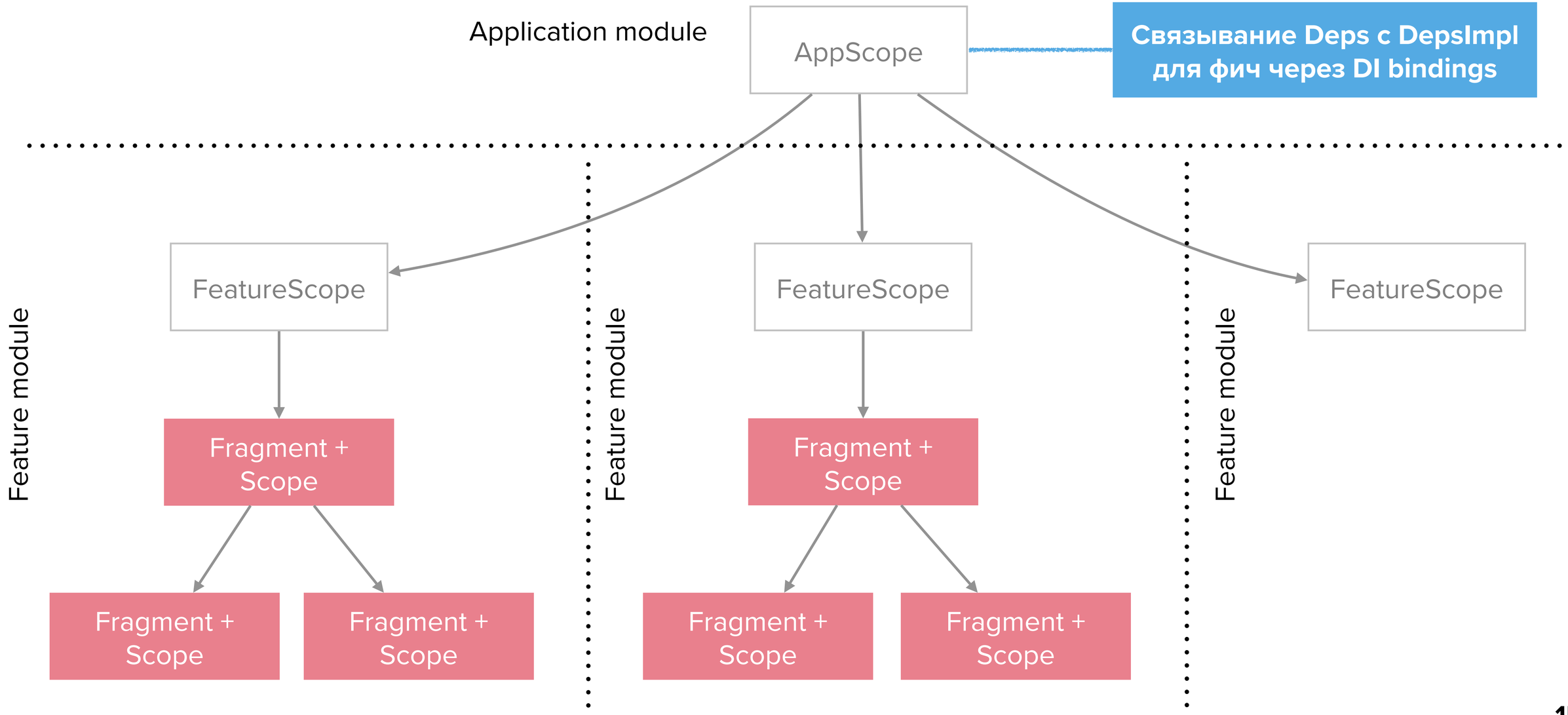


Какой из этих фрагментов должен открывать и закрывать DI Scope? В какой момент? 🤔

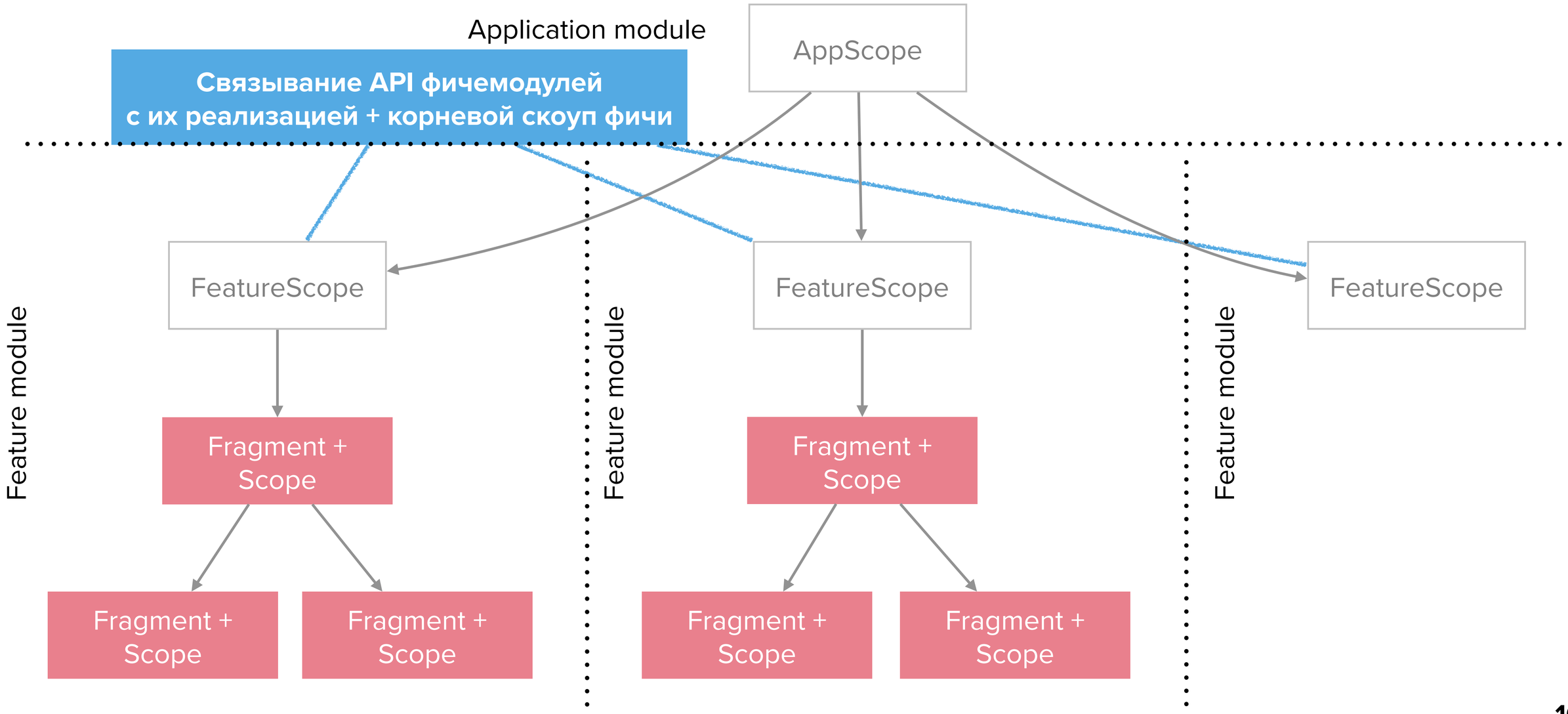
Иерархия скоупов приложения



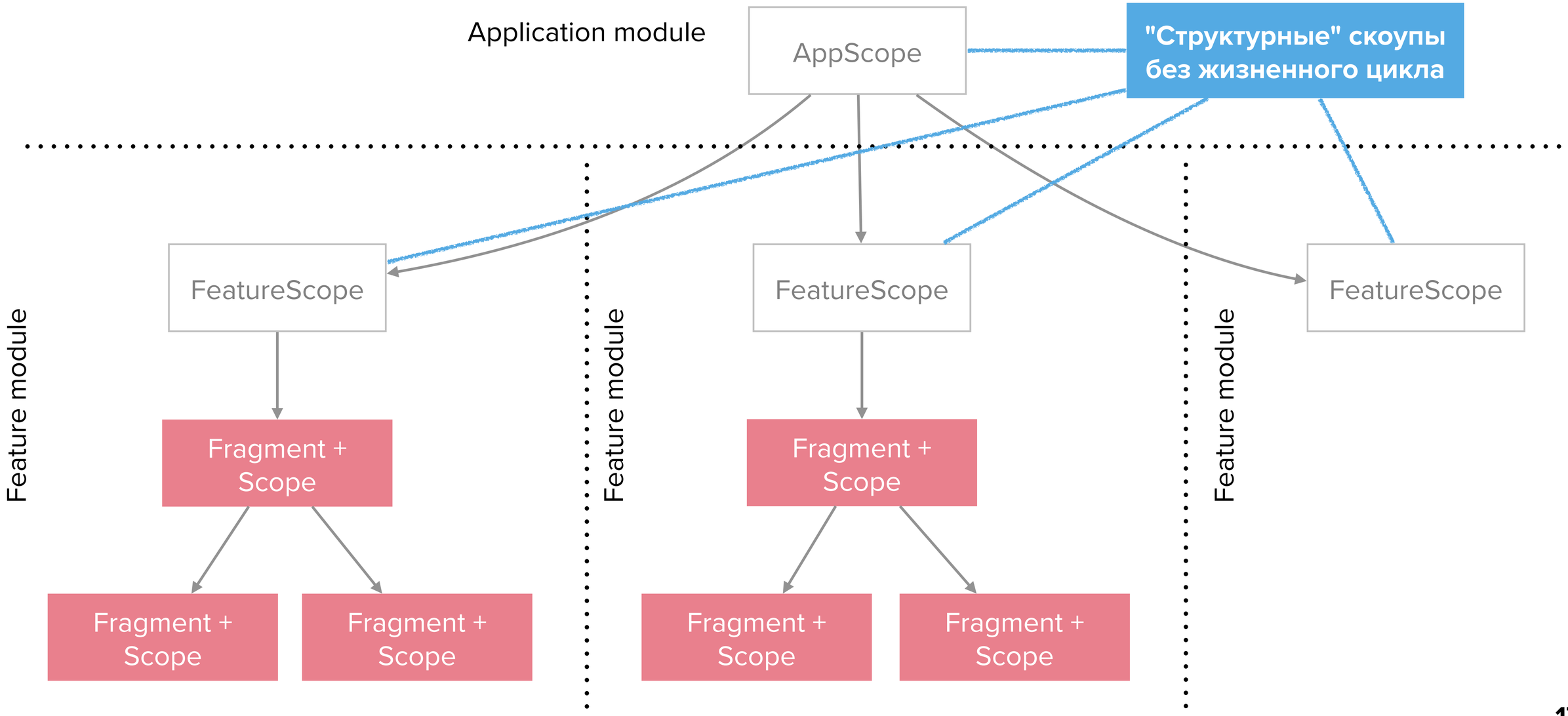
Иерархия скоупов приложения



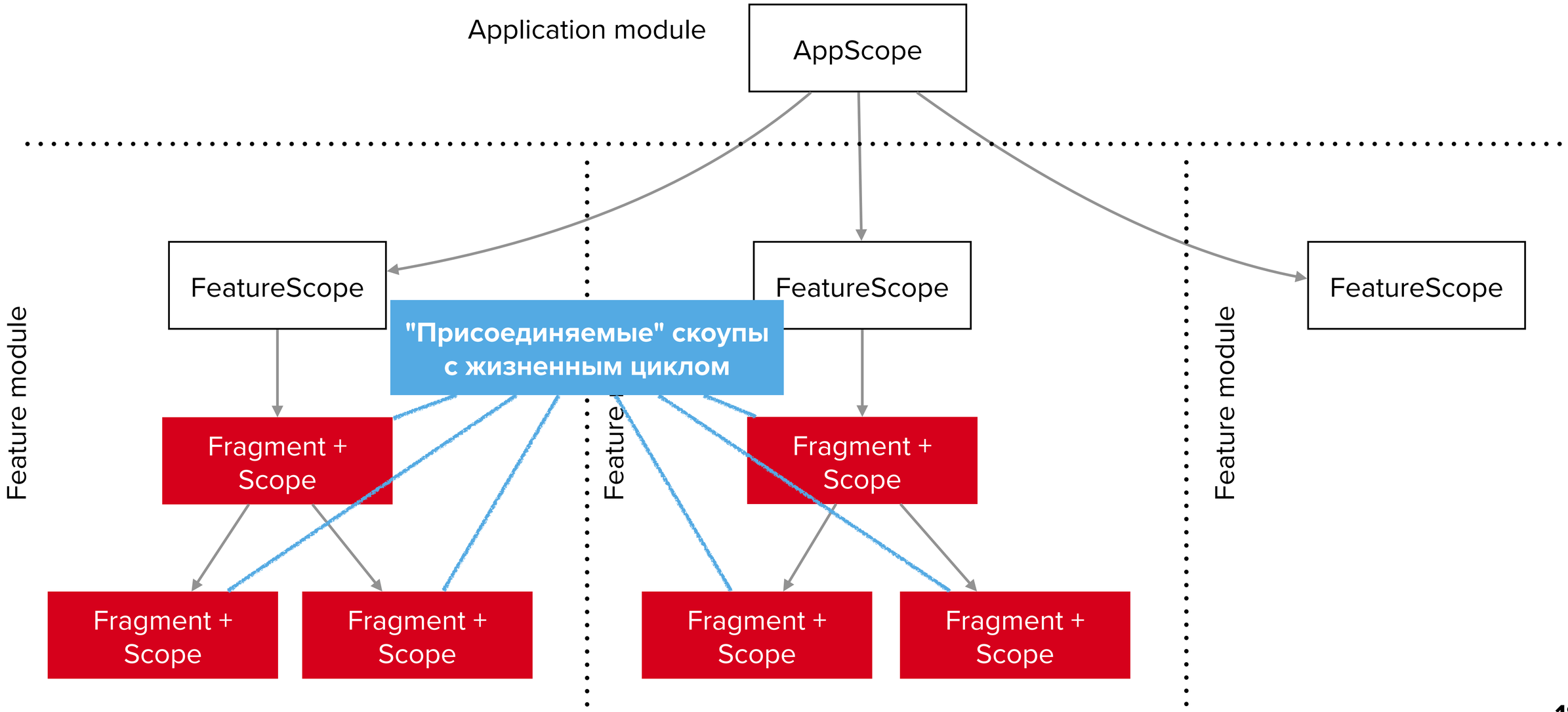
Иерархия скоупов приложения



Иерархия скоупов приложения



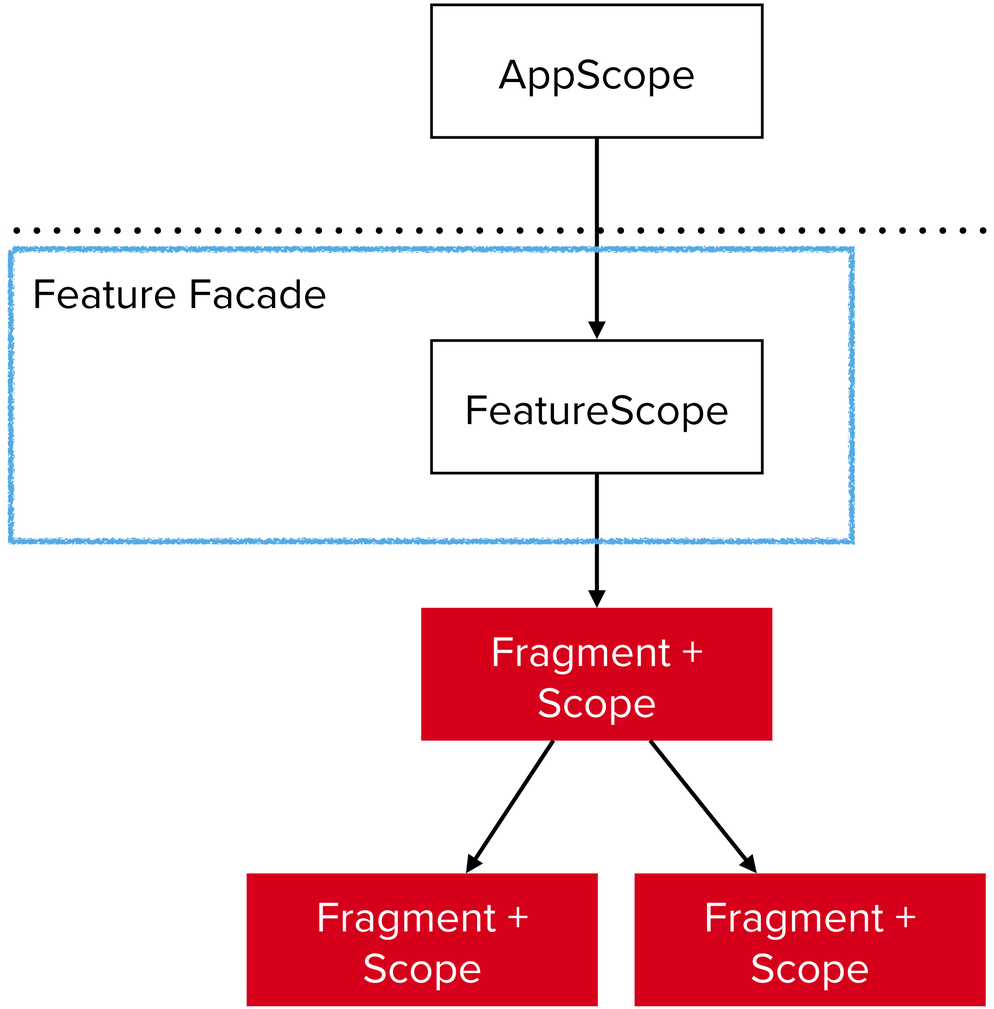
Иерархия скоупов приложения



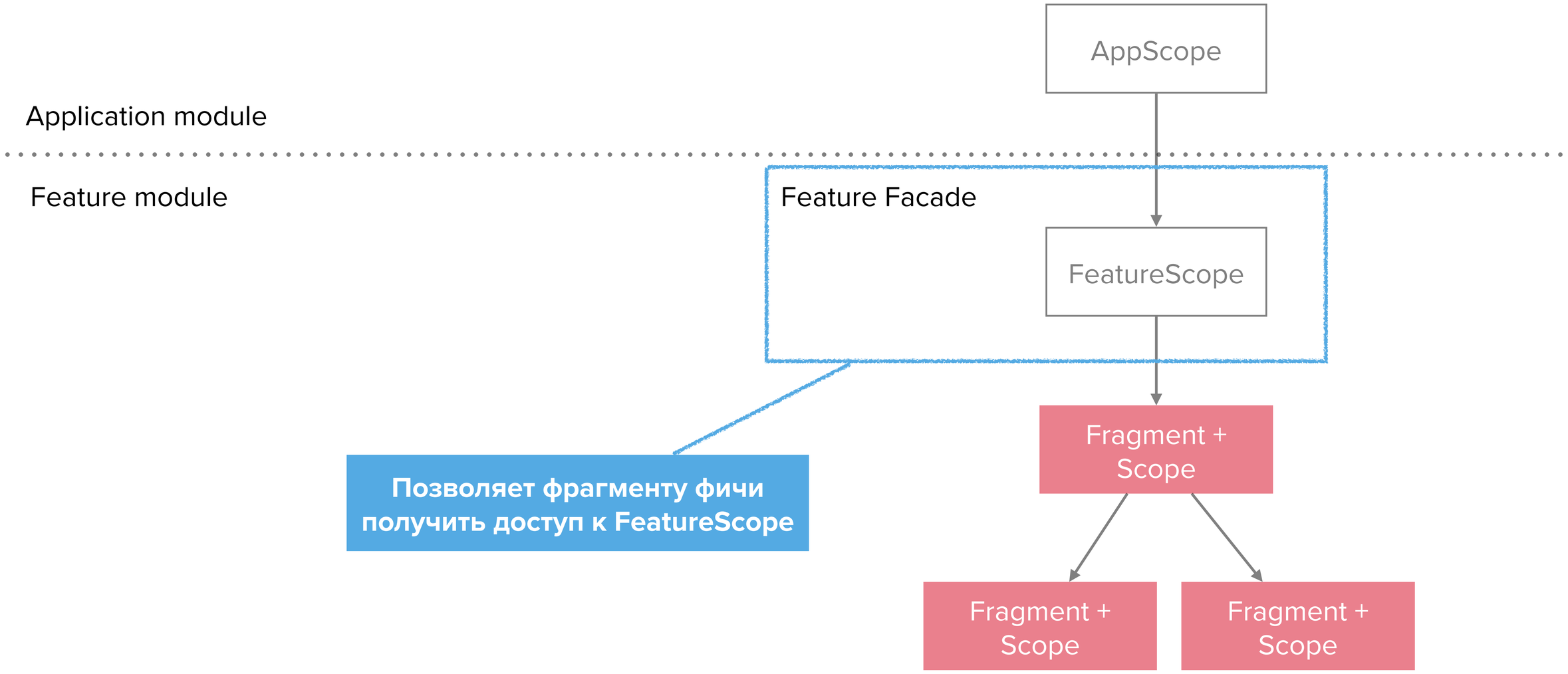
Feature Facade

Application module

Feature module

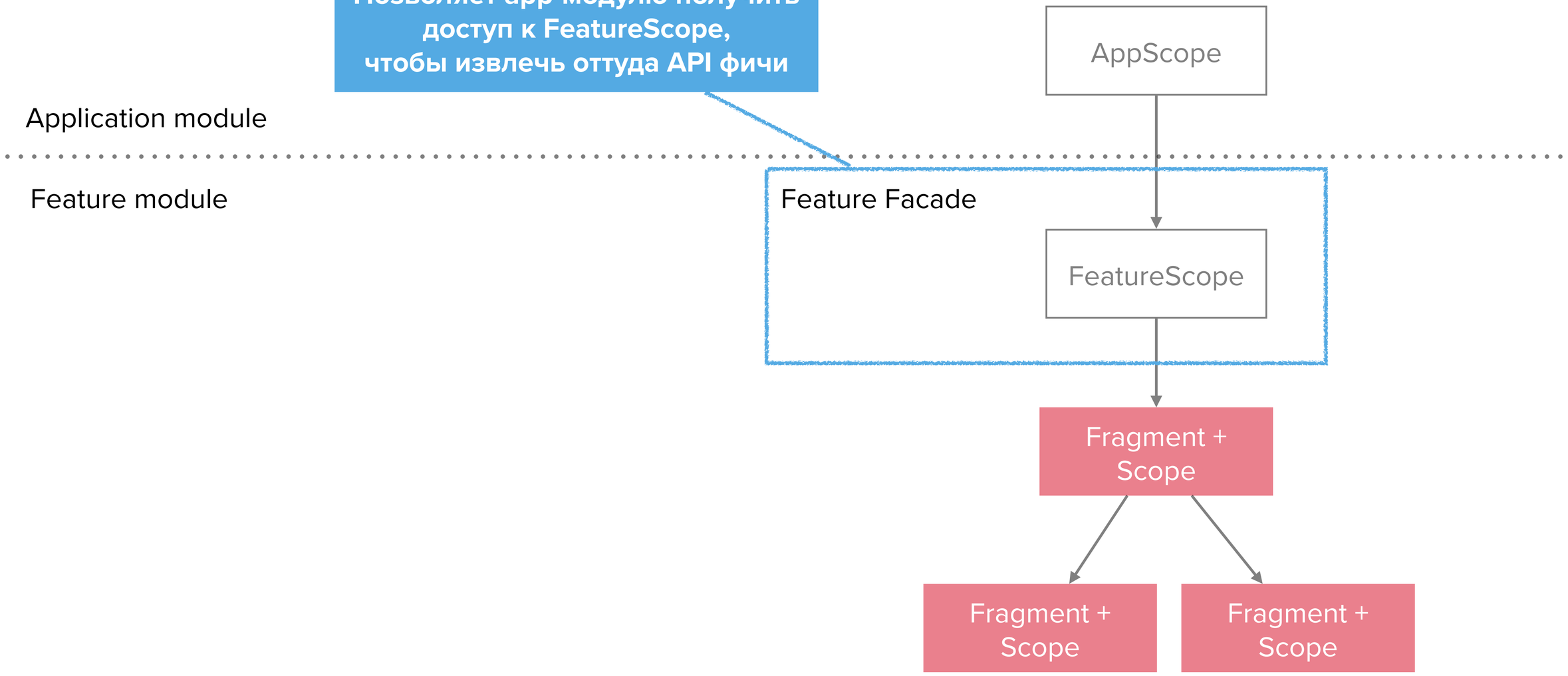


Feature Facade



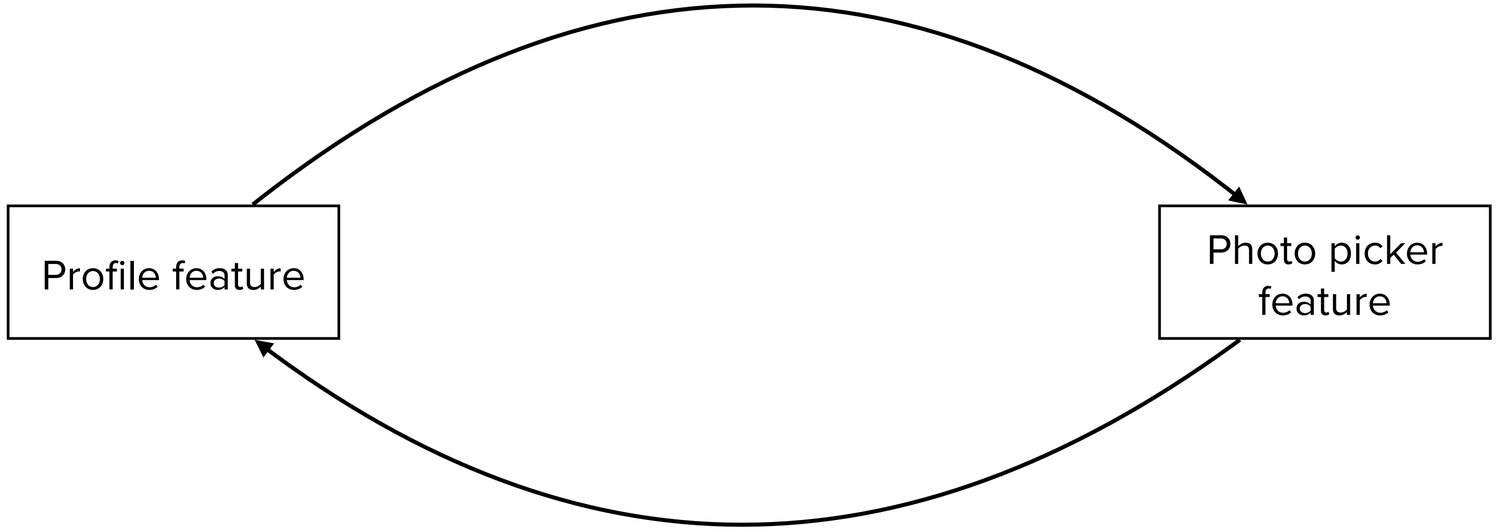
Feature Facade

Позволяет app-модулю получить доступ к FeatureScope, чтобы извлечь оттуда API фичи



Пример: выбор фото для профиля

Пользователь может выбрать новое фото
для профиля через пикер из другого модуля



Фича выбора аватара
возвращает результат в фичу профиля

Внутри feature-модуля profile

```
interface ProfileDeps {

    fun photoPickerFragment(profileId: String): Fragment

    fun photoSelections(profileId: String): Observable<String>

}
```

Внутри feature-модуля profile

```
@InjectConstructor
class ProfileApi {

    fun profileFragment(userProfile: UserProfile): Fragment =
        ProfileFragment.newInstance(userProfile)

}
```

Внутри feature-модуля profile

```
class ProfileFacade : FeatureFacade<ProfileDeps, ProfileApi>(
    depsClass = ProfileDeps::class.java,
    apiClass = ProfileApi::class.java,
    featureScopeName = "ProfileFeature",
    featureScopeModule = {
        Module().apply {
            bind<ProfileApi>().singleton().releasable()
        }
    }
)
```

Внутри feature-модуля profile

```
internal class ProfileFragment : Fragment(R.layout.fragment_profile) {

    private val di = DiFragmentPlugin(
        fragment = this,
        parentScope = { ProfileFacade().featureScope },
        scopeNameSuffix = { userProfile.id },
        scopeModules = { arrayOf(ProfileScreenModule(userProfile)) }
    )

    private val viewModel by lazy { di.get<ProfileViewModel>() }

    /*...*/
}
```

Скоуп фрагмента фичи открывается от структурного скоупа фичи

В реализации экрана фичемодуля можем инжектить внешние Deps

```
@InjectConstructor
internal class ProfileViewModel(
    private val initialUserProfile: UserProfile,
    private val deps: ProfileDeps,
    disposable: CompositeDisposable
)
```

Внутри feature-модуля photo picker

```
data class PhotoSelection(
    val selectionId: String,
    val photo: Photo
)
```

```
@InjectConstructor
class PhotoPickerApi {

    private val photoSelectionRelay =
        PublishRelay.create<PhotoSelection>()

    fun photoPickerFragment(args: PhotoPickerArgs): Fragment =
        PhotoPickerFragment.newInstance(args)

    fun photoSelections(): Observable<PhotoSelection> =
        photoSelectionRelay.hide()

    internal fun postPhotoSelection(photoSelection: PhotoSelection) =
        photoSelectionRelay.accept(photoSelection)

}
```

Внутри feature-модуля photo picker

```
class PhotoPickerFacade : FeatureFacade<PhotoPickerDeps, PhotoPickerApi>(
    depsClass = PhotoPickerDeps::class.java,
    apiClass = PhotoPickerApi::class.java,
    featureScopeName = "PhotoPickerFeature",
    featureScopeModule = {
        Module().apply {
            bind<PhotoPickerApi>().singleton()
        }
    }
)
```

Внутри app-модуля

```
@InjectConstructor
internal class ProfileDepsImpl(
    // для реализации зависимостей фичемодуля, может понадобиться Api другого фичемодуля
    private val photoPickerApi: PhotoPickerApi
) : ProfileDeps {

    override fun photoPickerFragment(profileId: String): Fragment =
        photoPickerApi.photoPickerFragment(PhotoPickerArgs((profileId)))

    override fun photoSelections(profileId: String): Observable<String> =
        photoPickerApi.photoSelections()
            .filter { it.selectionId == profileId }
            .map { it.photo.url }

}
```

Внутри app-модуля

```
private fun initTp() {
    // Используем rootScope Toothpick-а в качестве AppScope
    // и устанавливаем туда зависимости для фичемодулей
    Toothpick.openRootScope()
        .installModules(FeatureDepsModule())
}


/**
 * Здесь происходит описание связей для склейки фичемодулей
 */
internal class FeatureDepsModule : Module() {
    init {
        bind<ProfileDeps>().toClass<ProfileDepsImpl>()
        bind<PhotoPickerApi>().toProviderInstance { PhotoPickerFacade().api }
    }
}
```

Иерархия скоупов приложения