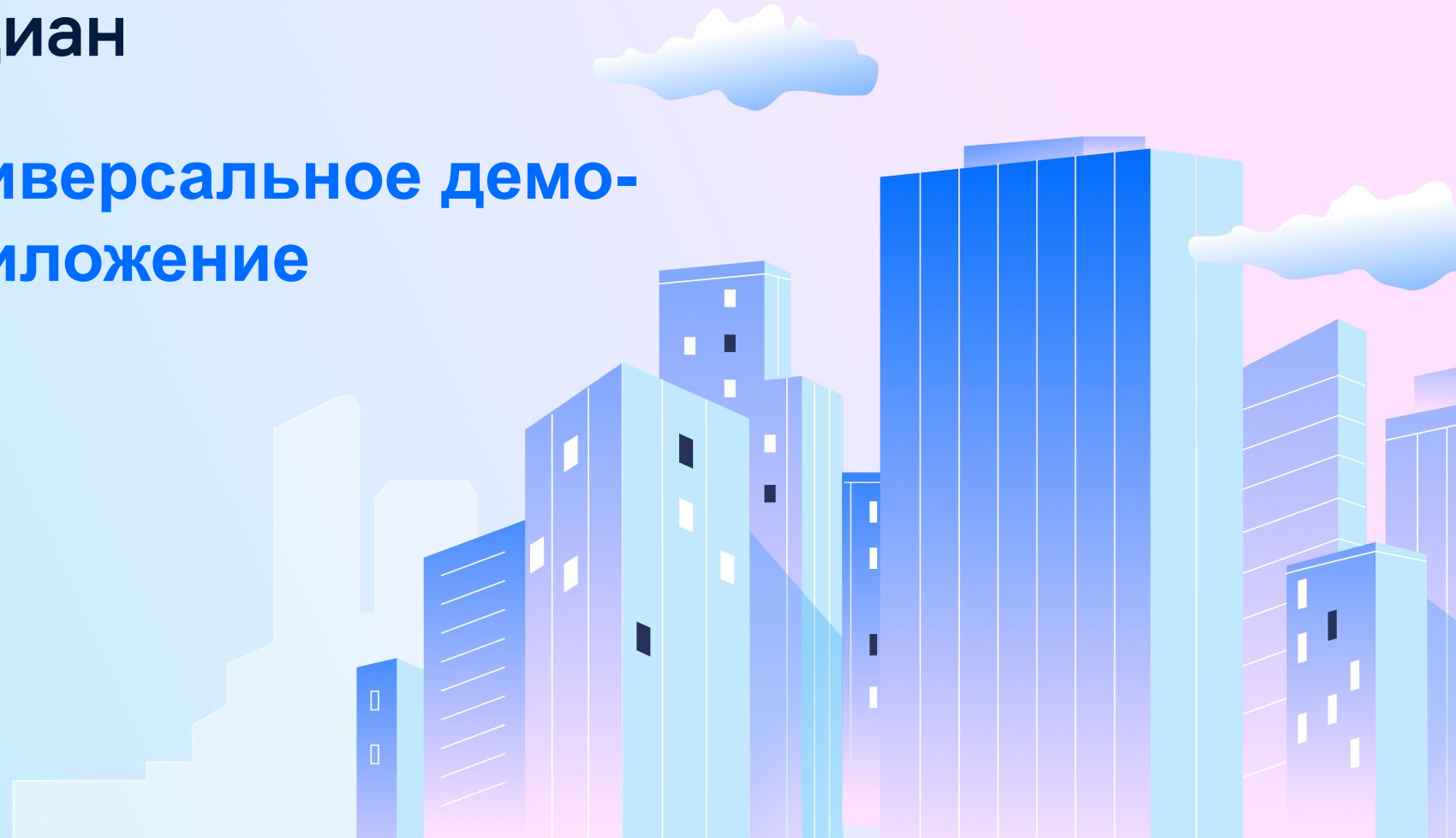
циан

Универсальное демо-приложение

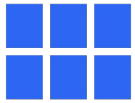


Давайте знакомиться

Данил Перевалов
Android Developer

циан

Проект



Gradle

650 модулей

Команда

21 разработчик

Kotlin

1.5 млн. строк

циан

Проблемы большого проекта

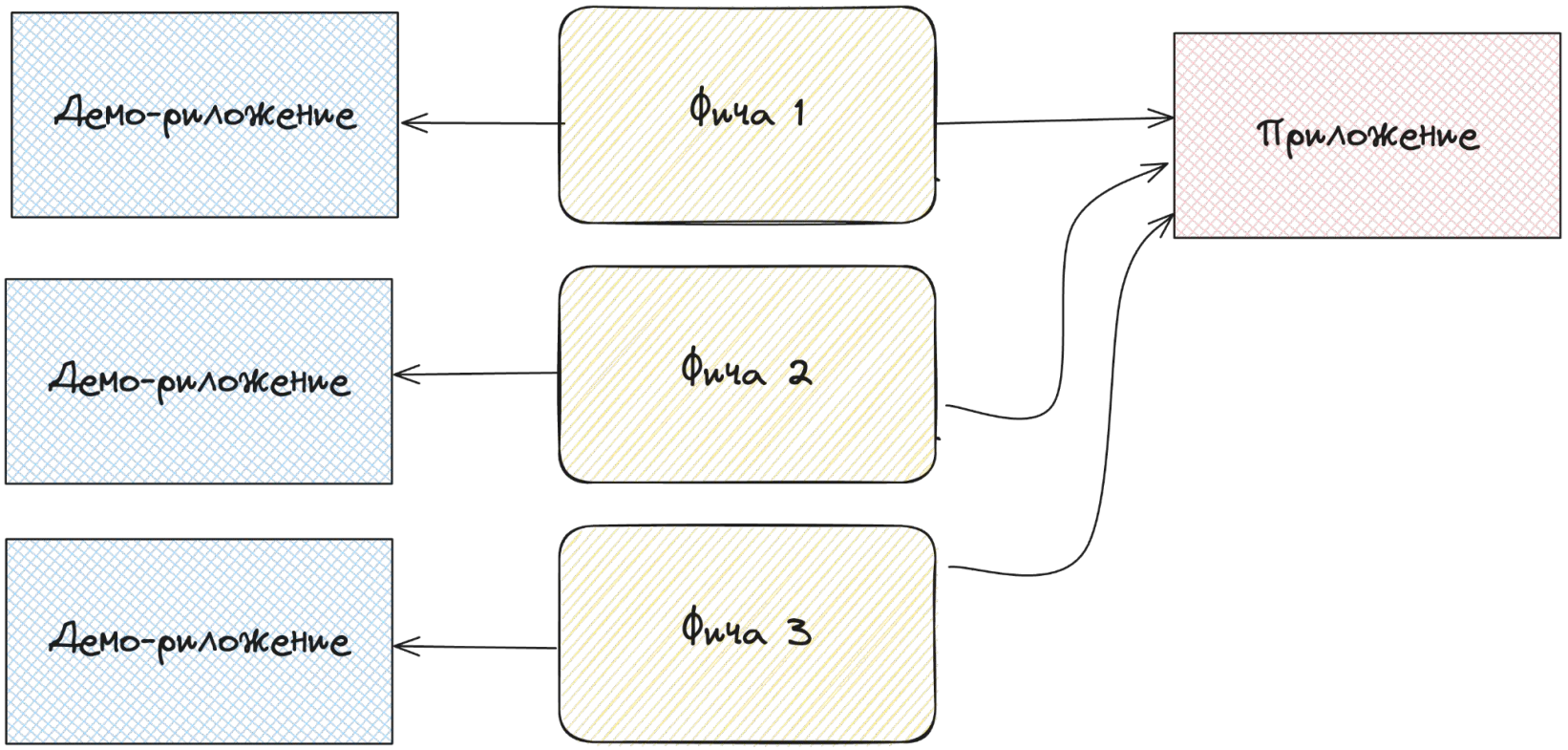
Много связей

Много лишних связей при компиляции и в то же время много лишнего маппинга

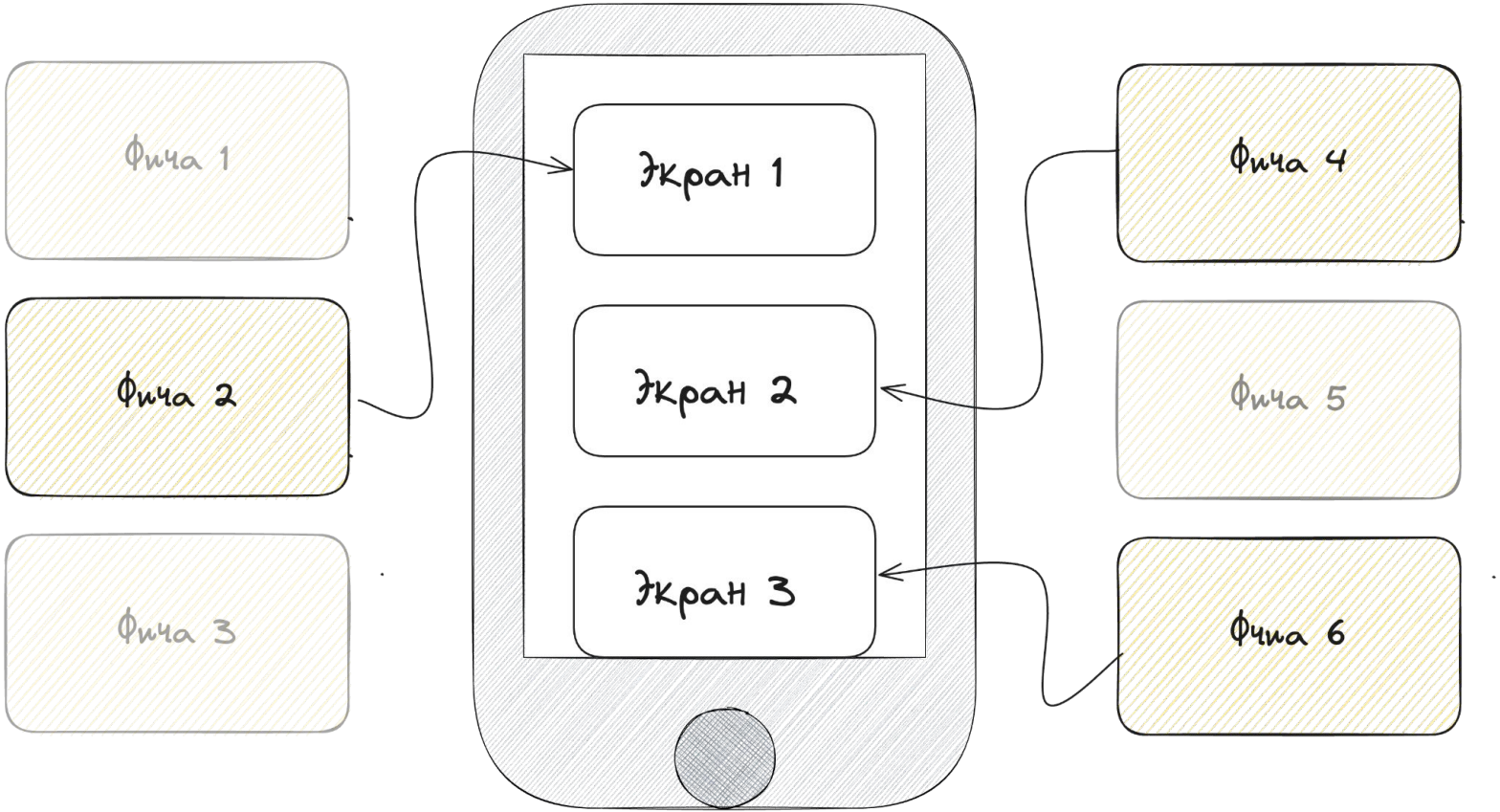
Скорость

Скорость работы Android Studio, конфигурации и компиляции

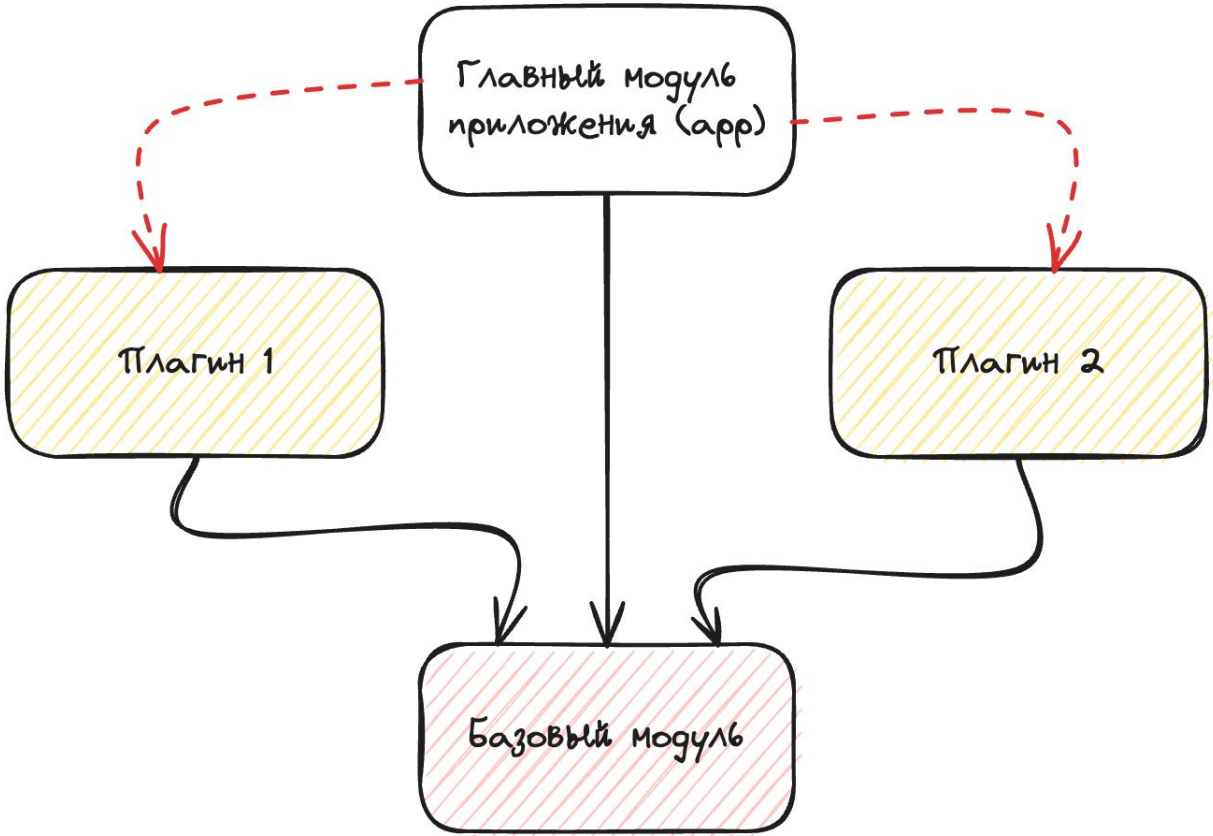
Демо-приложение



Универсальное демо-приложение



Плагиновидная структура модулей



циан

Результаты

1. Потребление памяти AS стало не превышает 3 Гб
2. Время синхронизации уменьшилось в 8 раз
3. Время горячей сборки уменьшилось с 1 минуты до 6 секунд
4. Время холодной сборки уменьшилось с 13 минут до 90 секунд

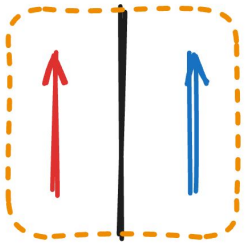
Про что сегодня поговорим

- Что такое плагиновидная структура модулей
- Как реализовать плагиновидую структуру
- Какие могут возникнуть проблемы
- Как из этого сделать универсальное демо-приложение

циан

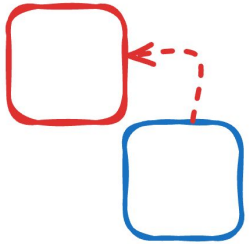
Что такое плагиновидная структура модулей?

Что такое плагин?



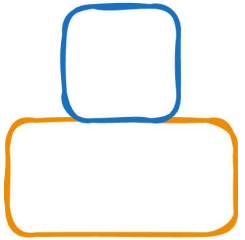
Независимо компилируется

Независимо, не значит отдельно



Динамически подключается

То есть после того, как код базы уже собран

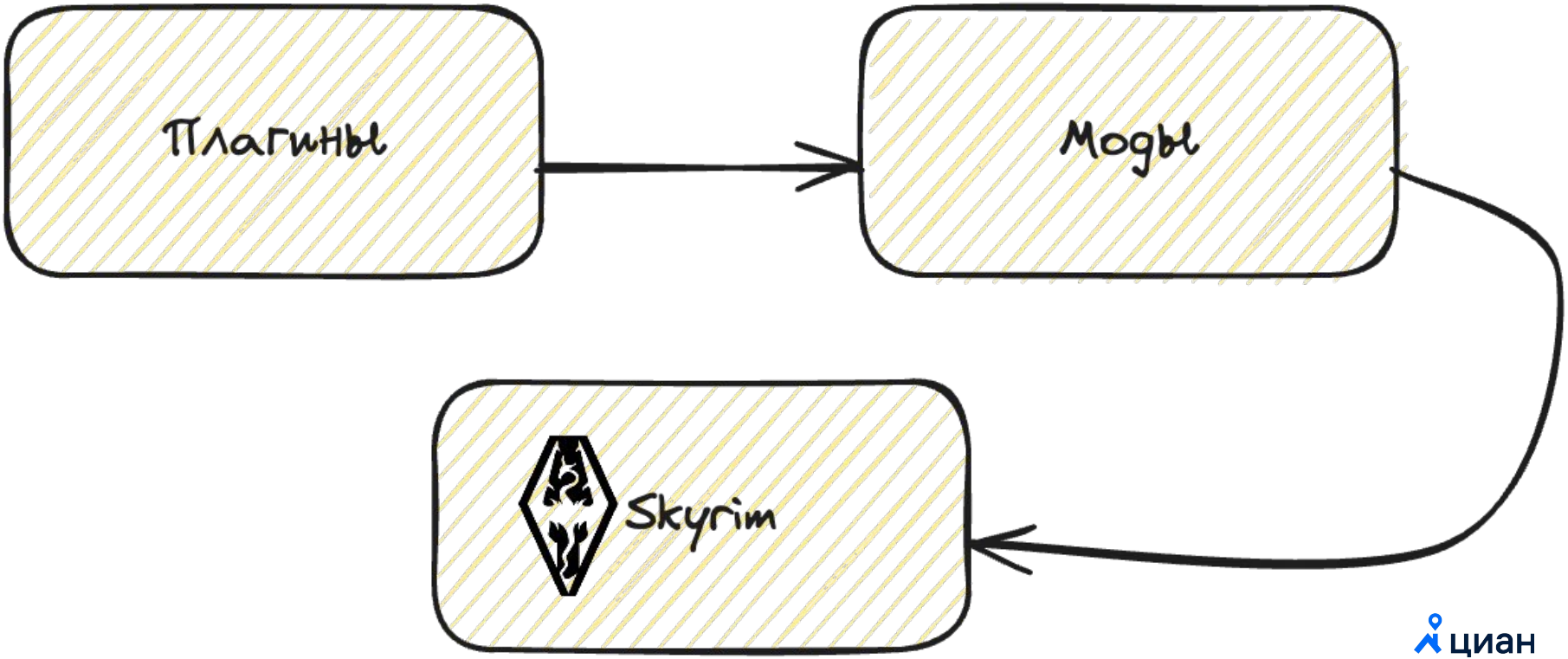


Расширяет возможности

Не самостоятелен, а базируется на базе

Модули это - сложно

Поэтому упрощаем

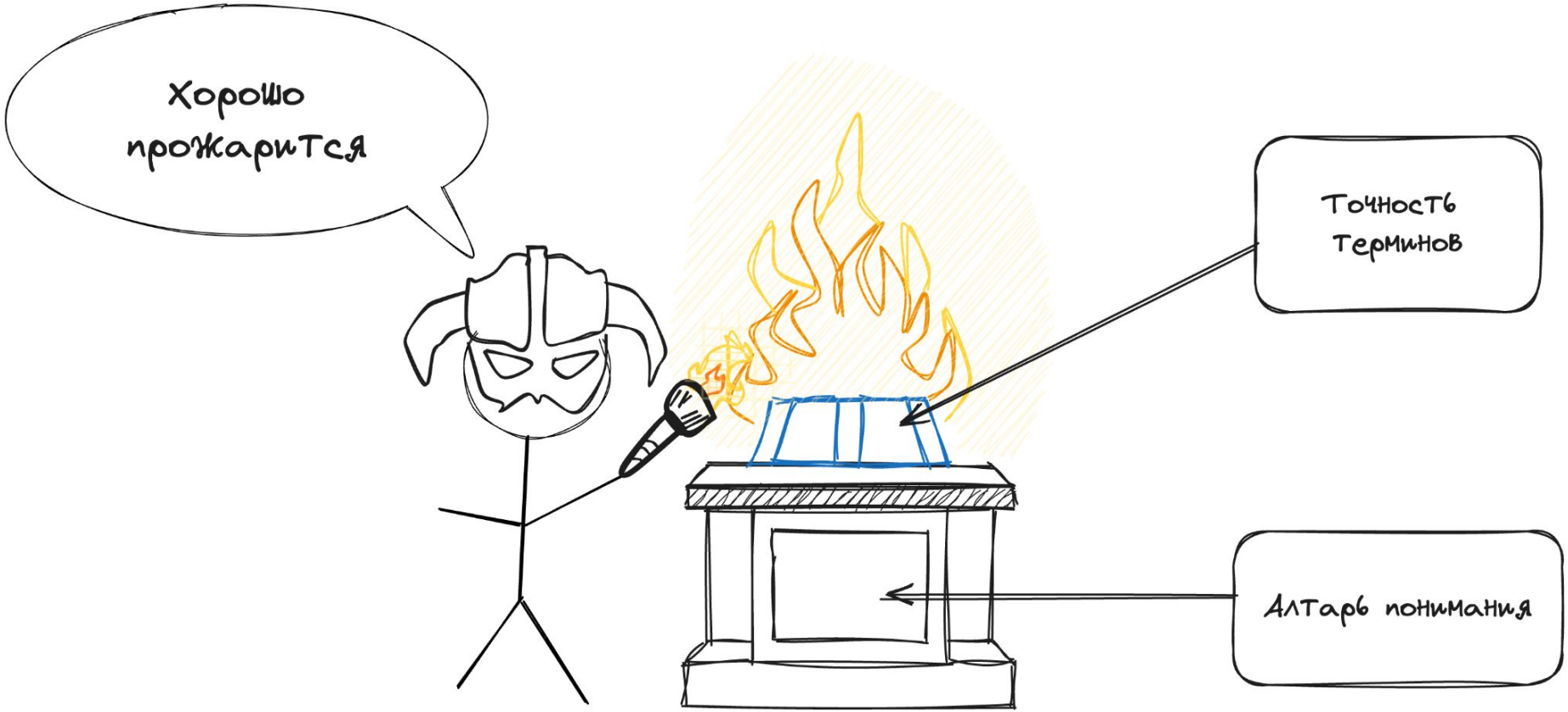


циан

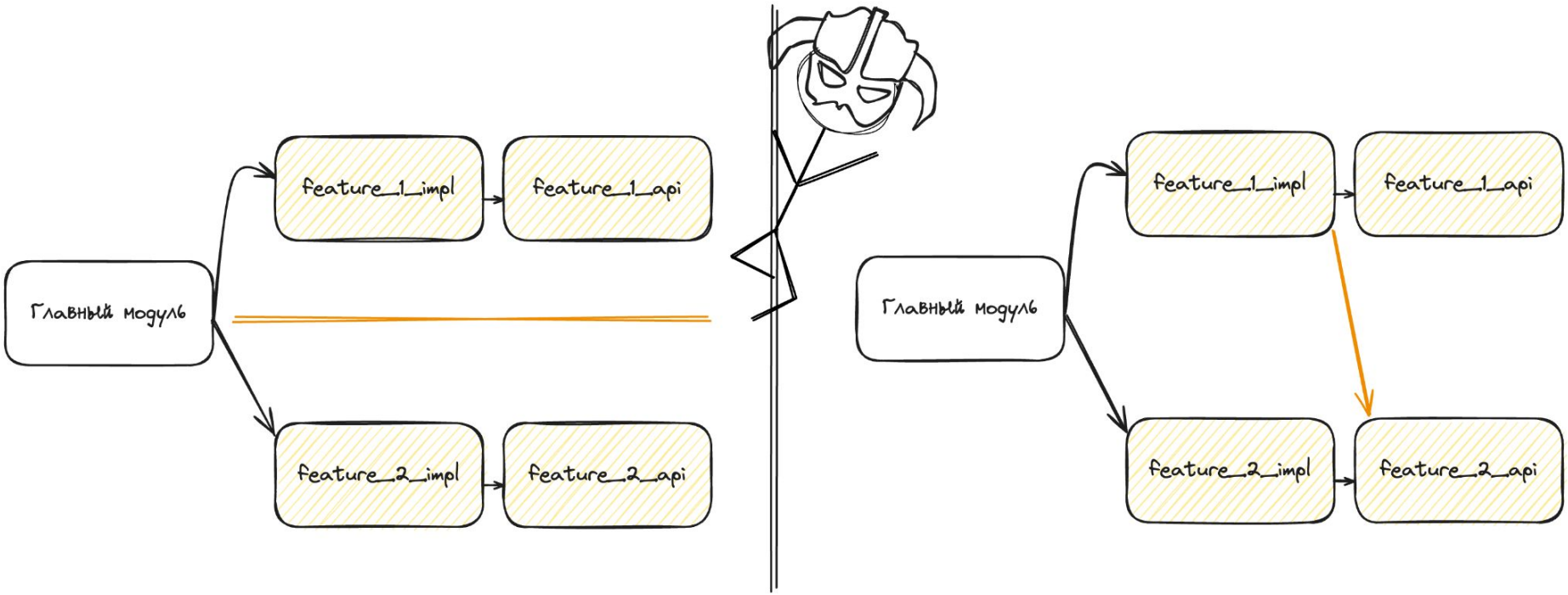
Помогаем довакину

циан

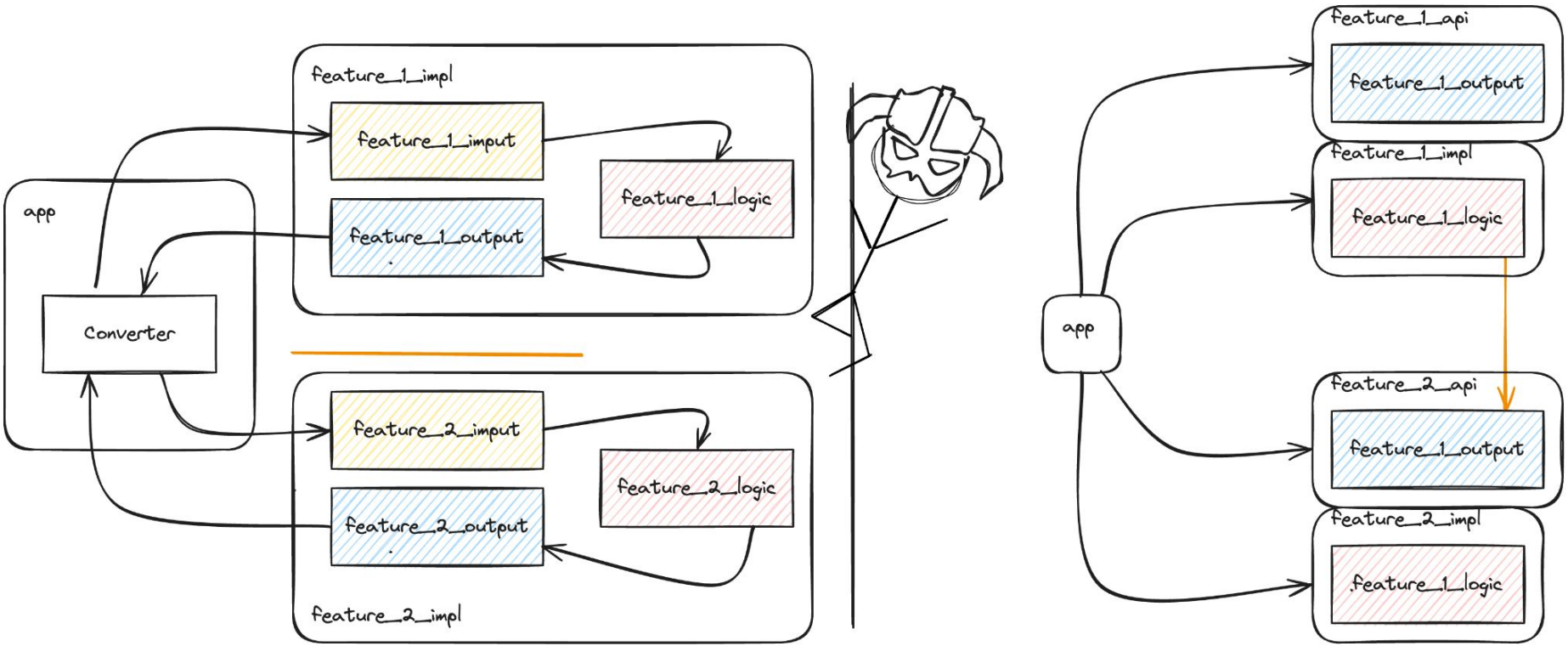
Здесь не будет точных определений



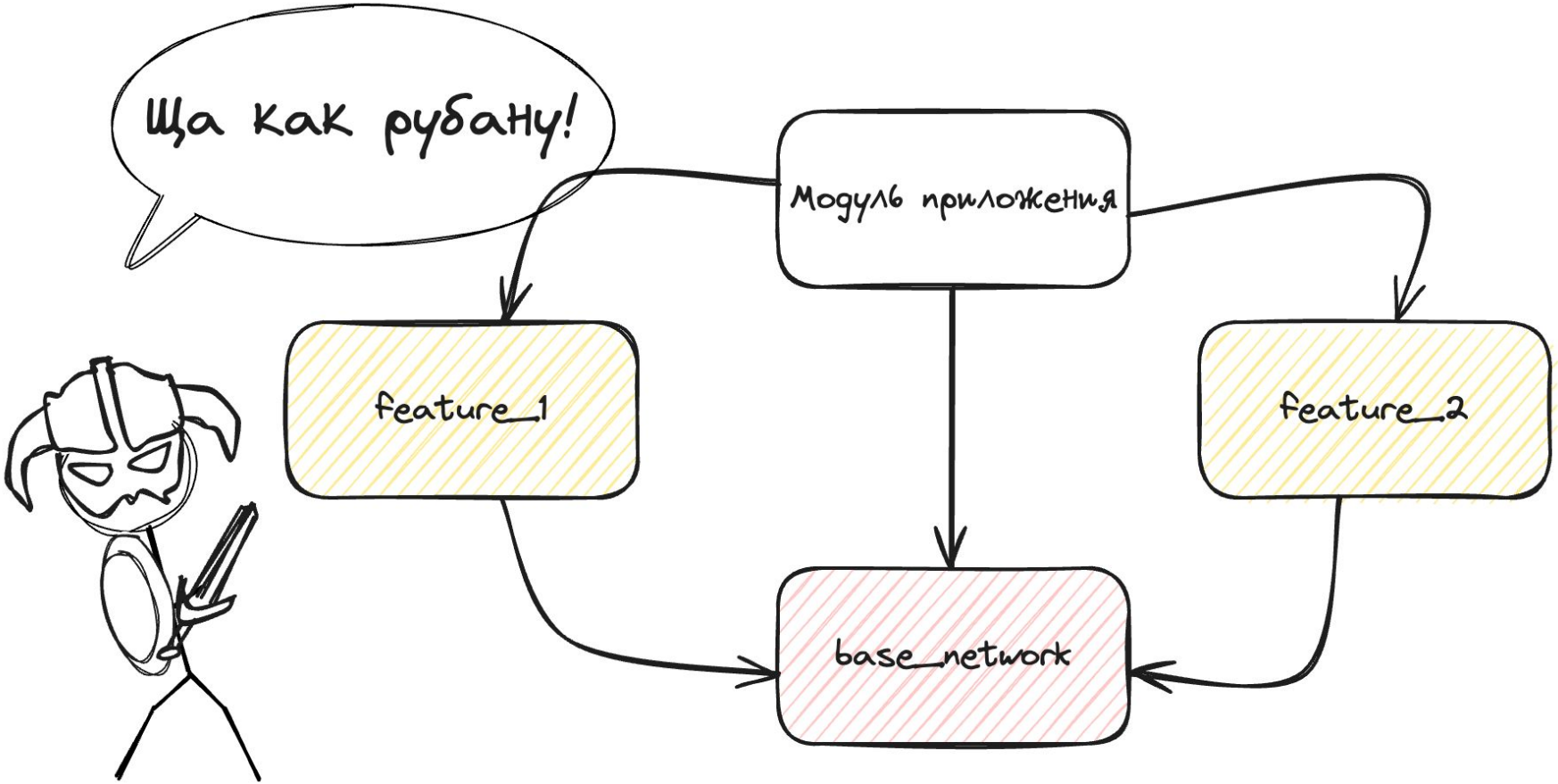
Два подхода к модулям. Изолированные и нет



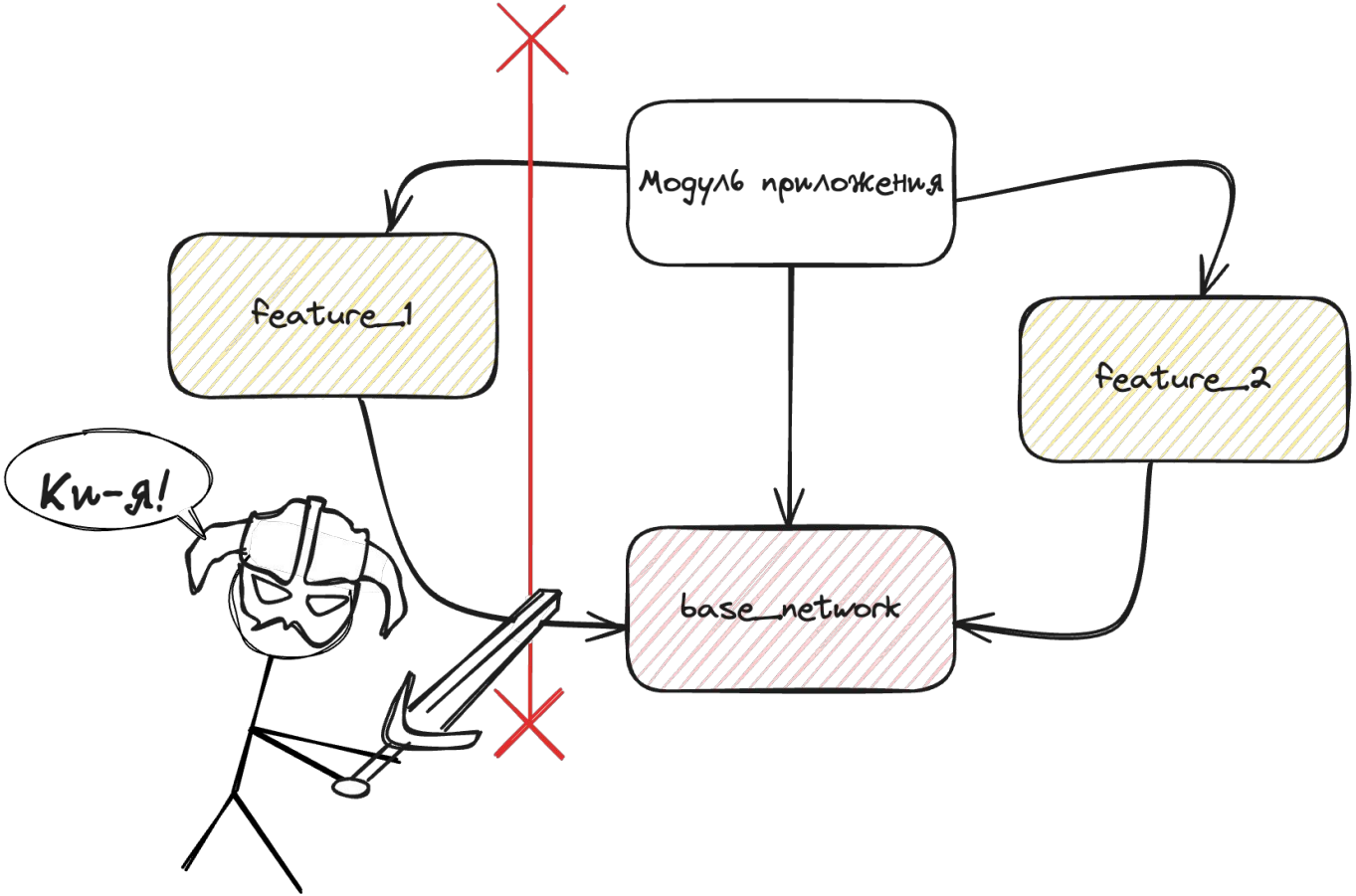
Два подхода к модулям. Уровень сущностей



Модульное приложение

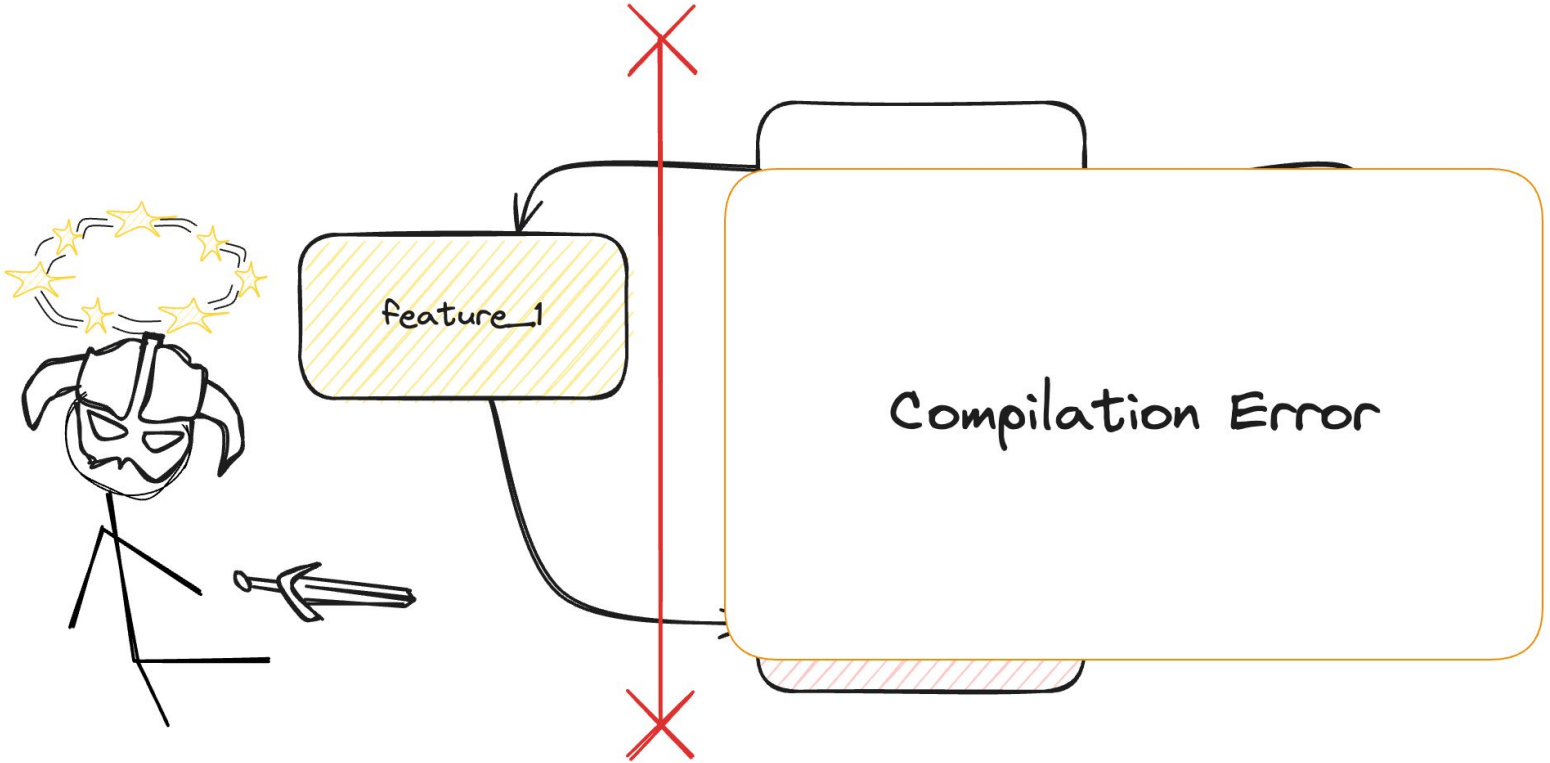


Пытаемся отсечь кусок

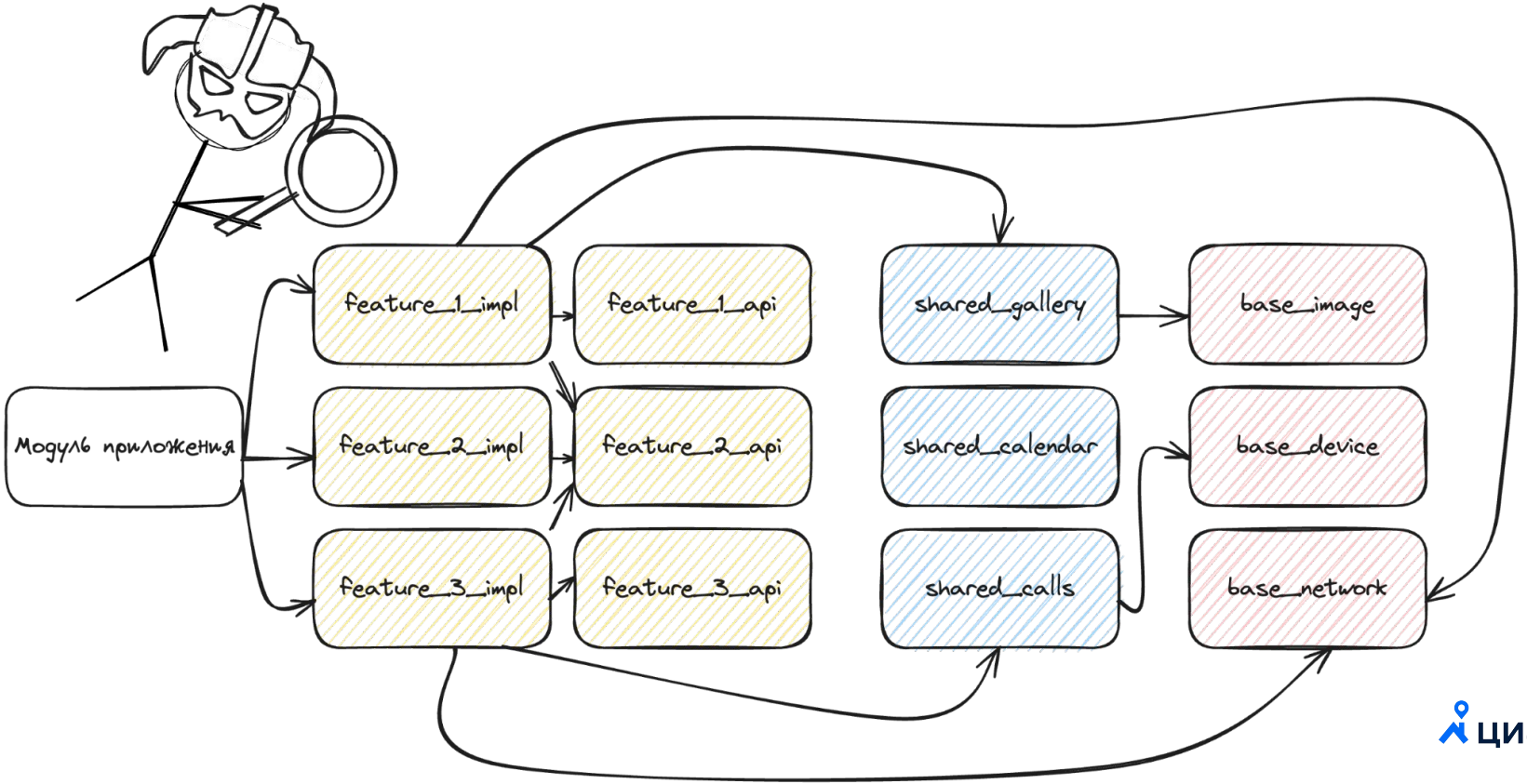


циан

Произойдёт ошибка компиляции

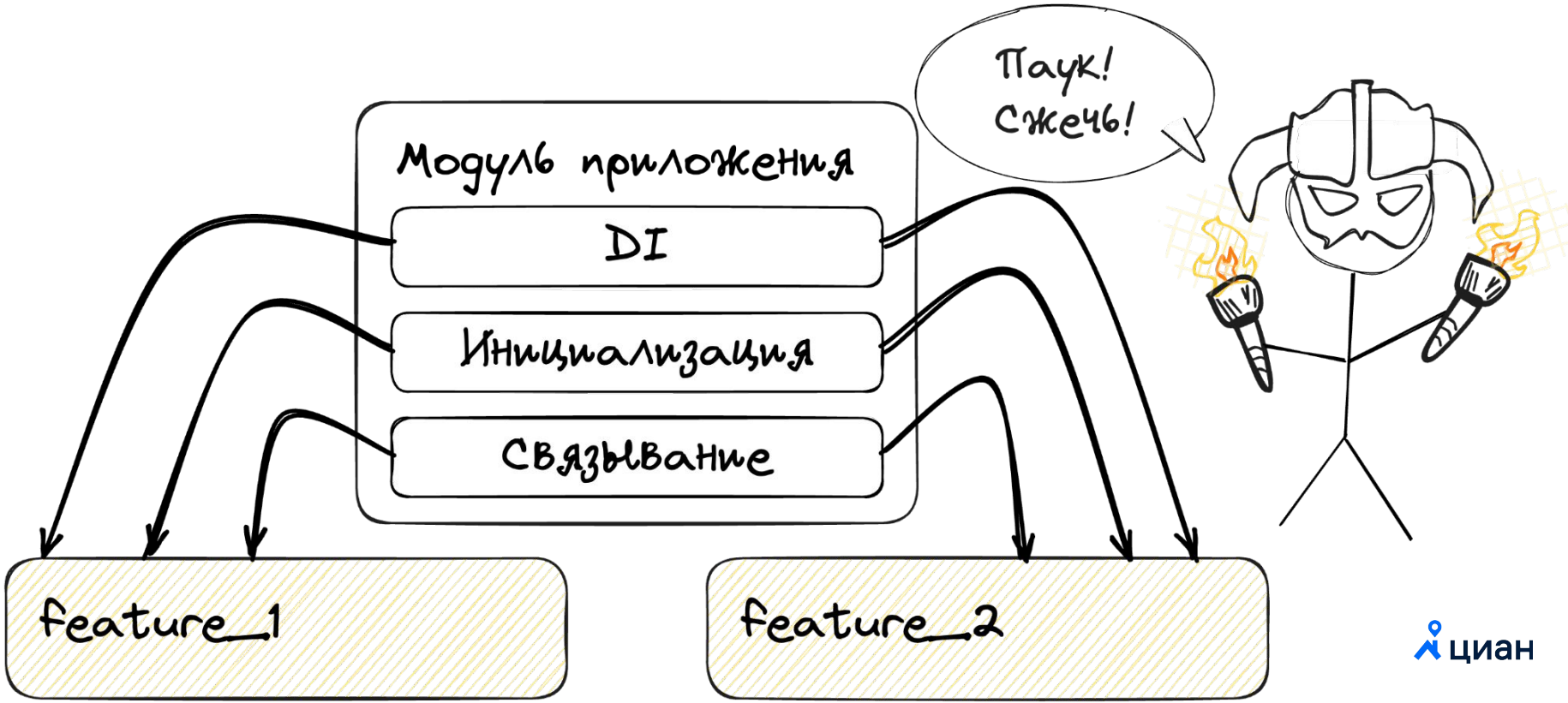


Смотрим поближе



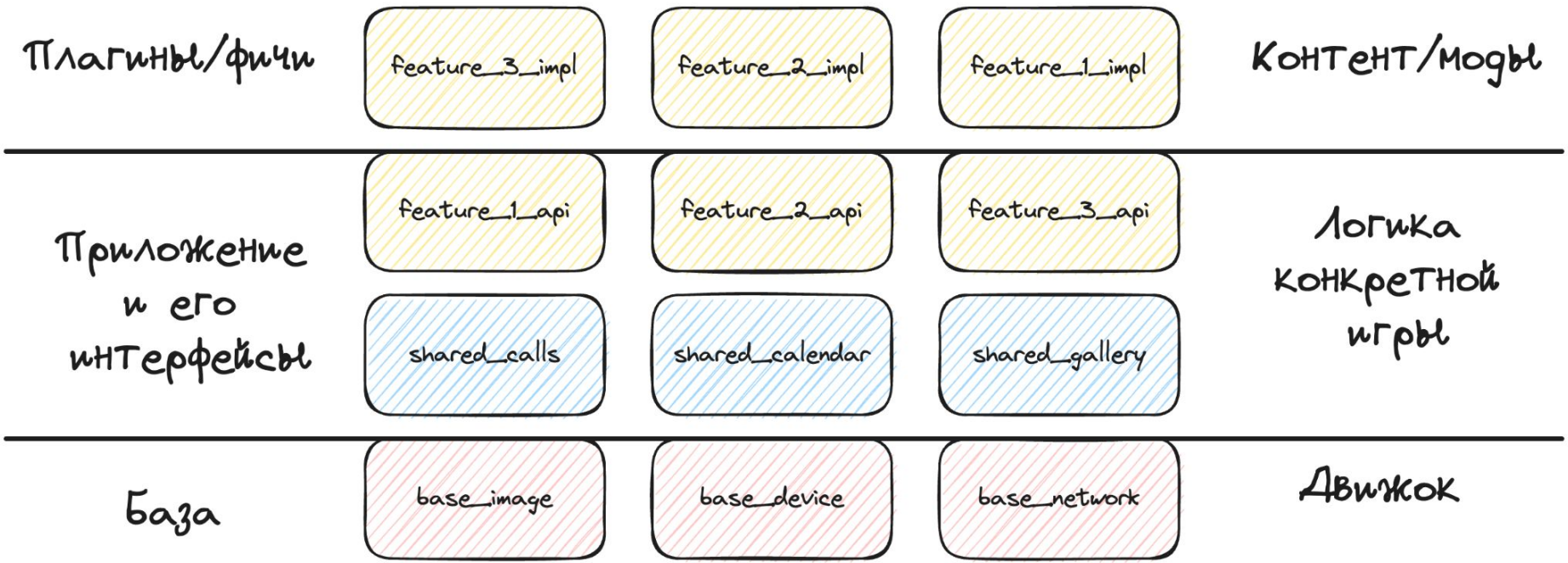
циан

А зачем?


Паук!
Сжечь!
Модуль приложения
DI
Инициализация
Связывание
feature_1
feature_2

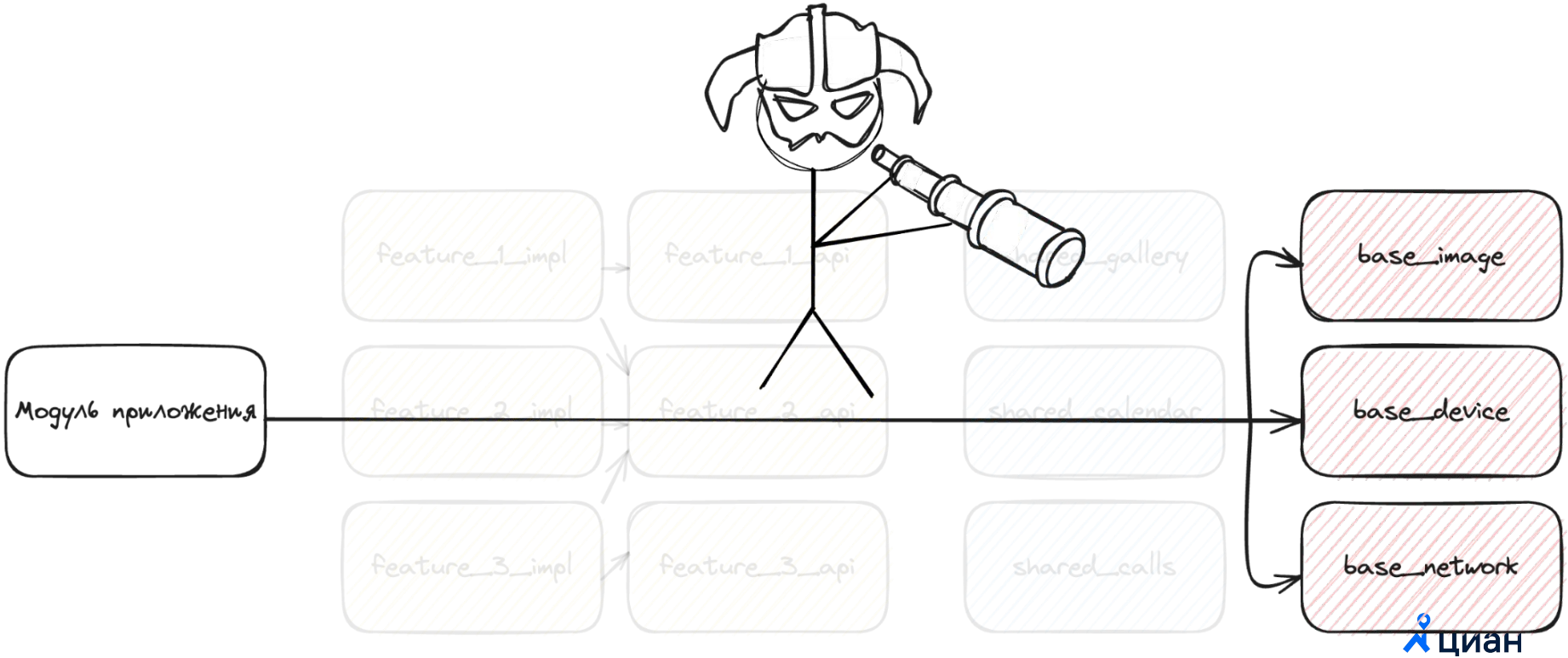
циан

Систематизируем



циан

Минимальная структура это база/движок



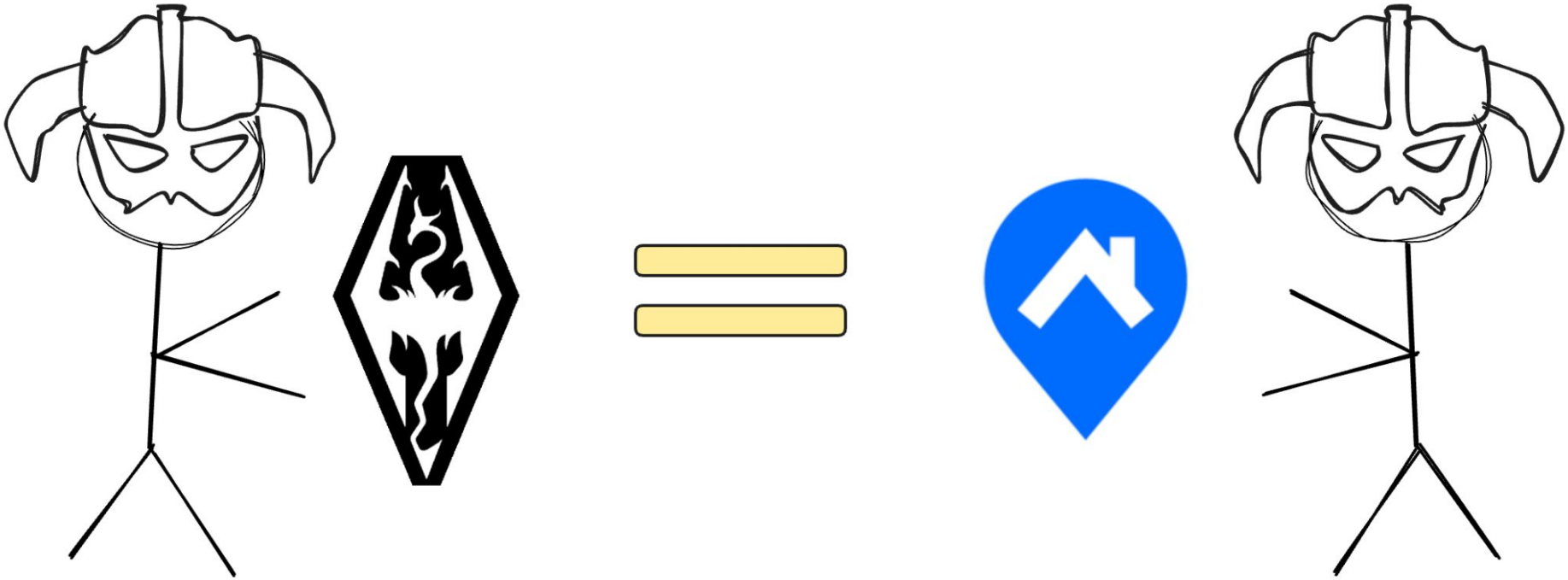
Поверх базы можно накатить любое приложение

Поверх приложения можно накатить любые фичи

Что-то модифицируется лучше, а что-то нет

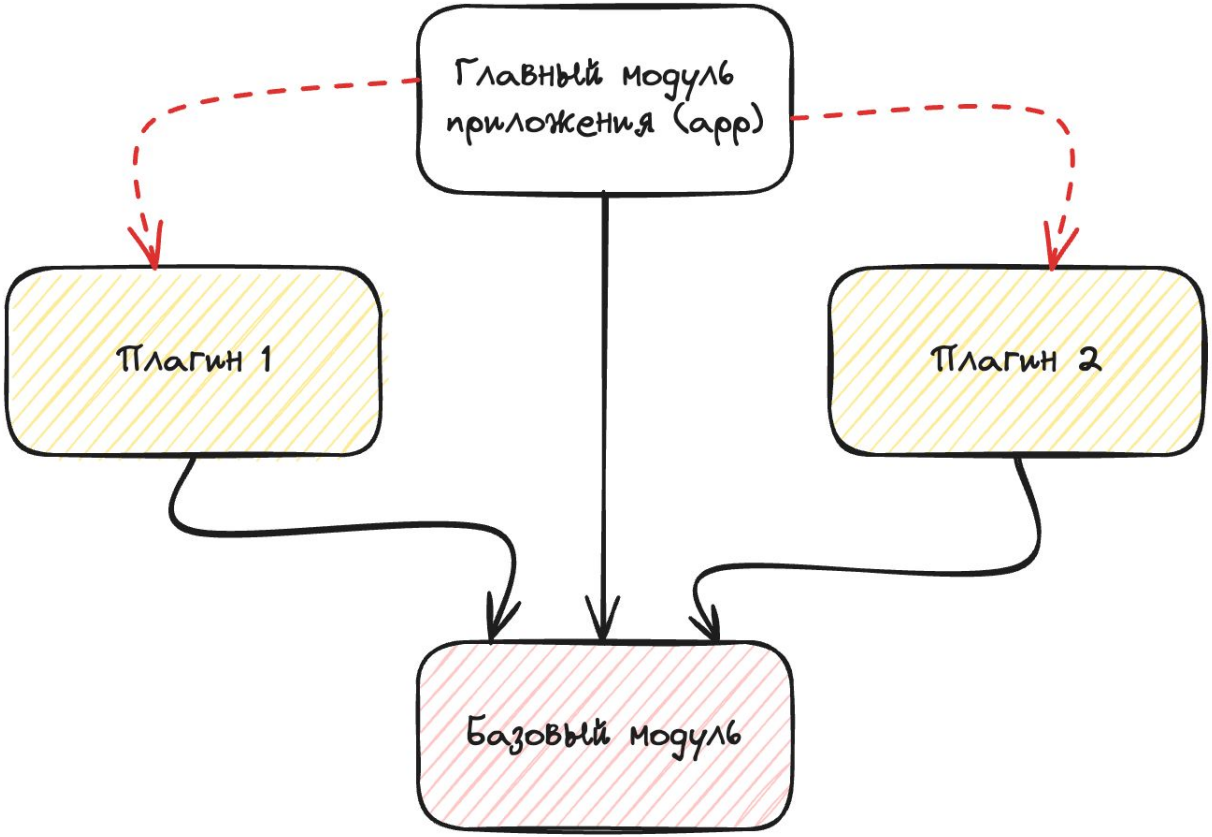
циан

В играх это работает!



циан

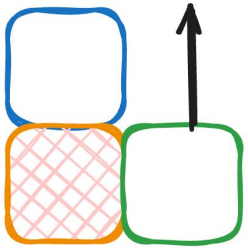
Что такое плагиновидная архитектура?



А зачем?

Ничего лишнего

Нет лишних связей при компиляции и в то же время нет лишнего маппинга



Демо-приложения

Можно с лёгкостью собрать базу только с теми фичами, что нужны

Скорость

Скорость работы Android Studio, конфигурации и компиляции

Про что сегодня поговорим

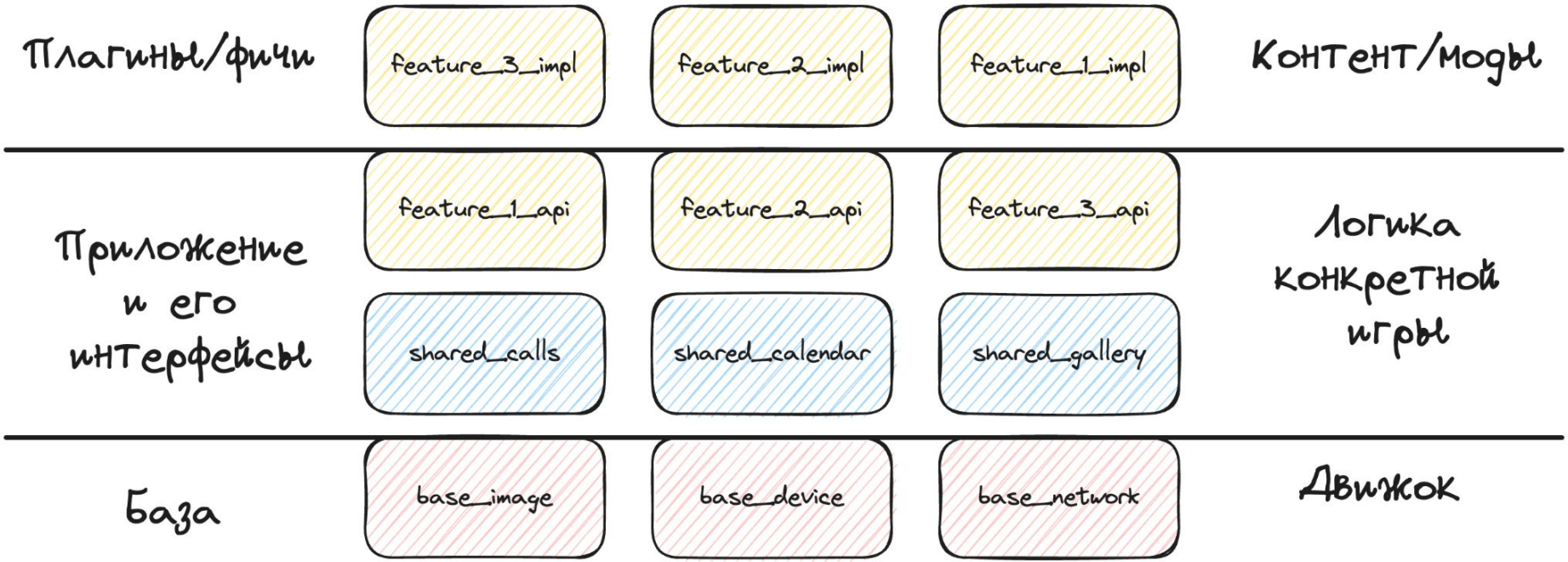
- ~~Что такое плагиновидная структура модулей~~
- Как реализовать плагиновидую структуру
- Какие могут возникнуть проблемы
- Как из этого сделать универсальное демо-приложение

циан

Как реализовать плагиновидую структуру?

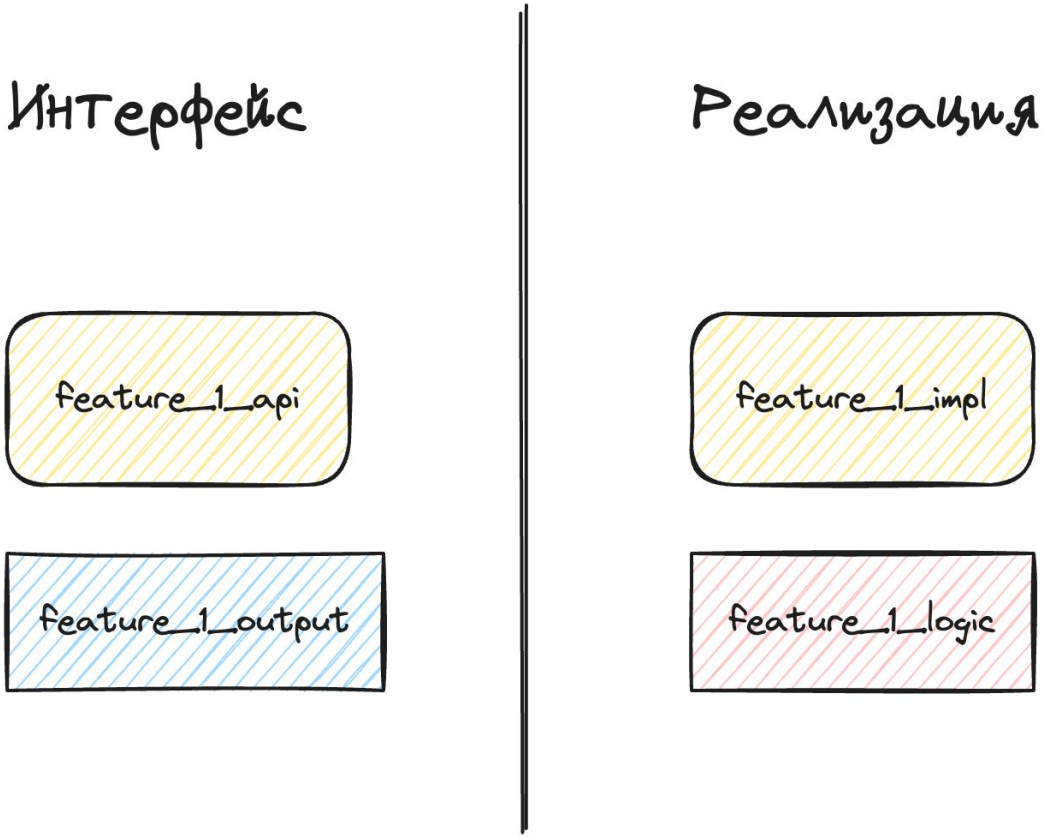
циан

Напоминаю структуру



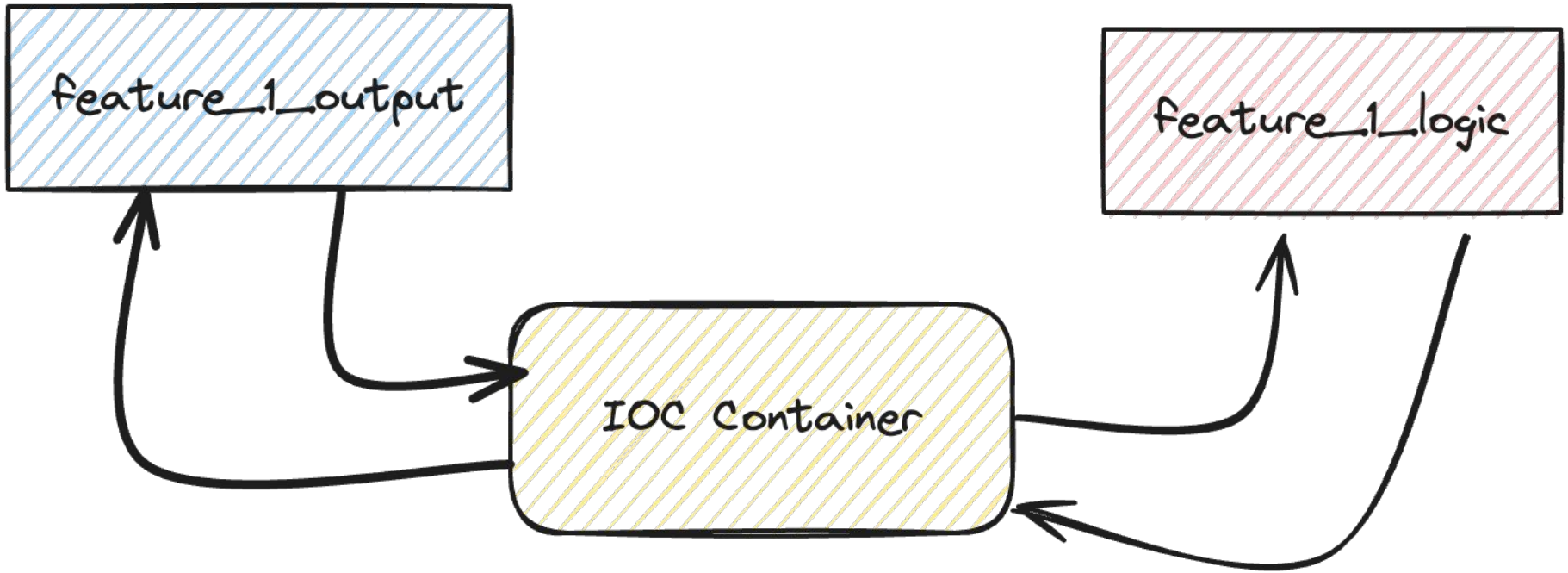
циан

Api/Impl

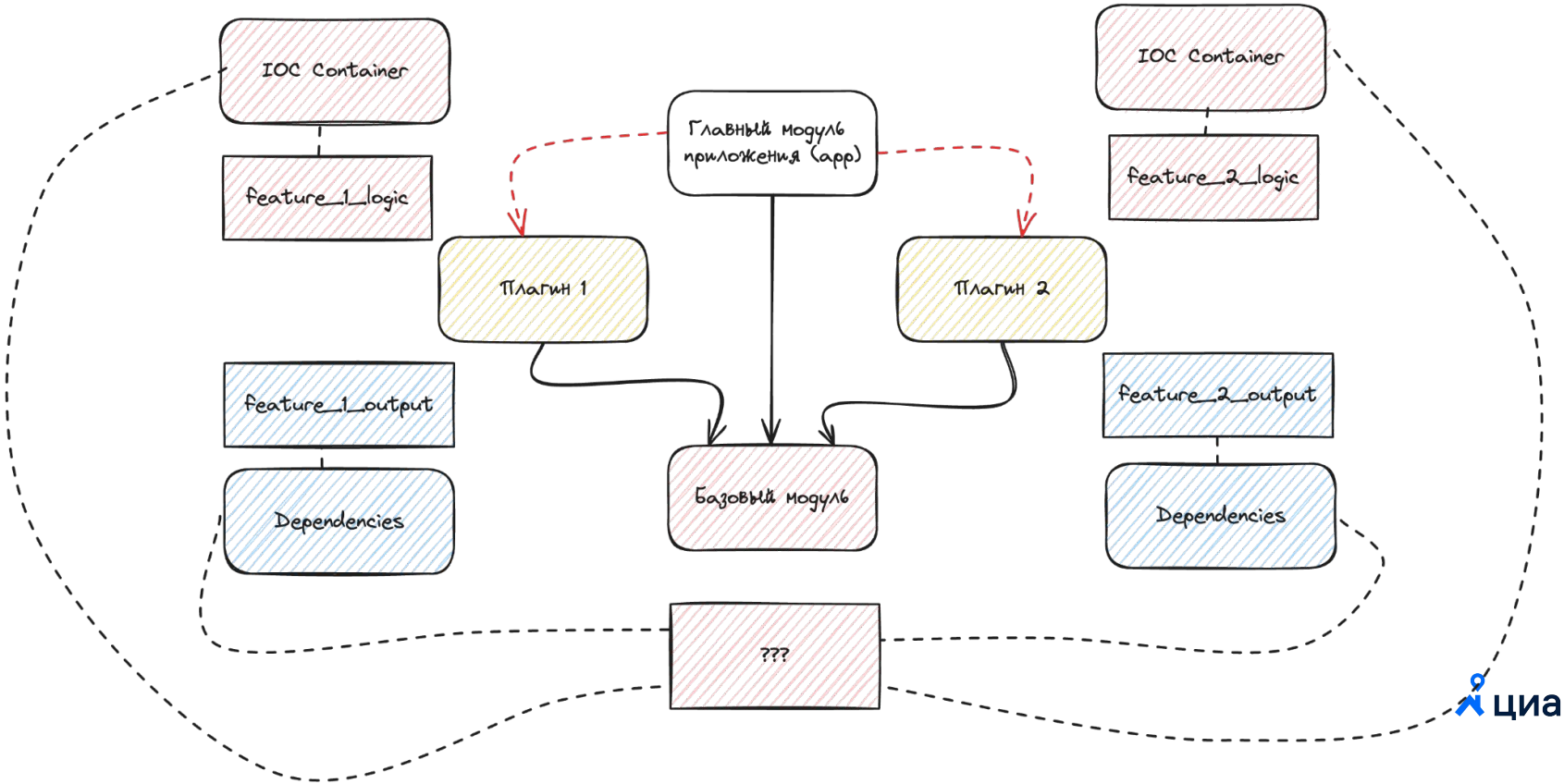


циан

Нам нужно получить реализацию по интерфейсу

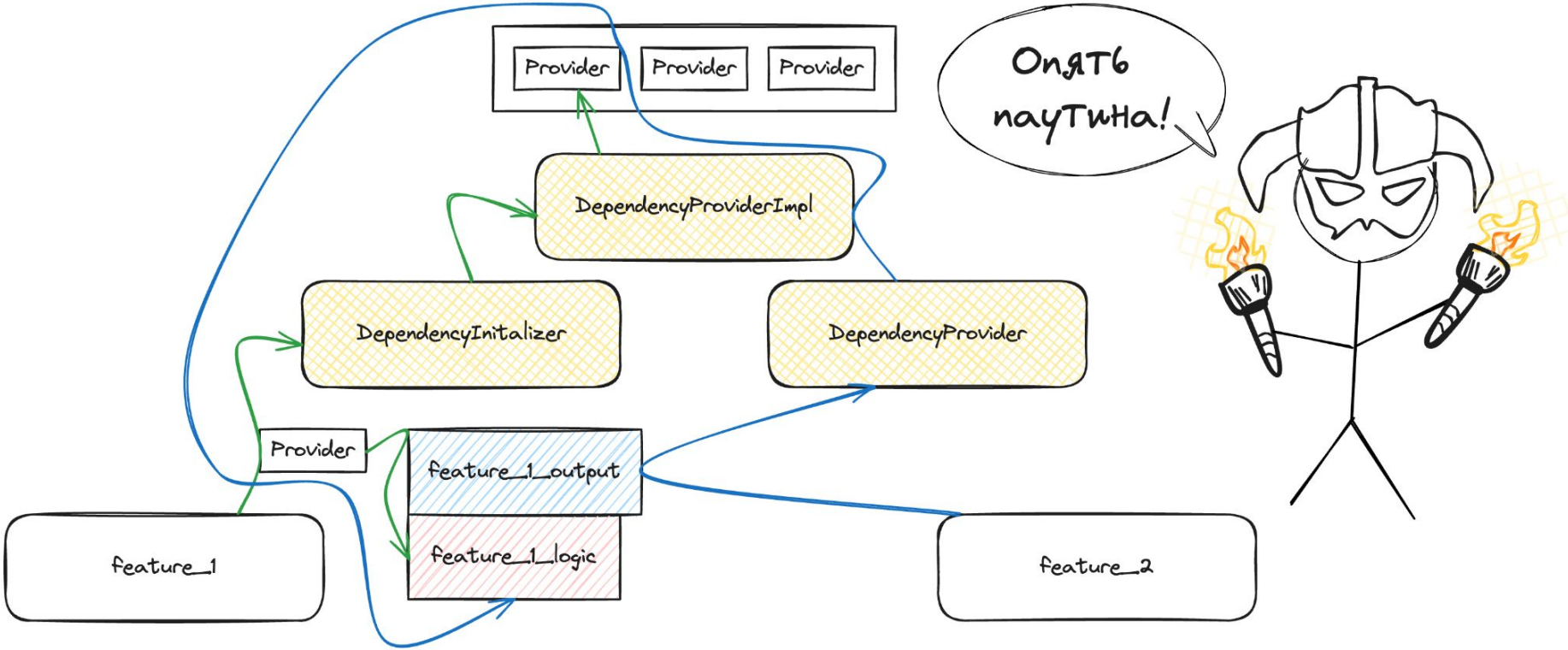


А как это сделать для плагинов?

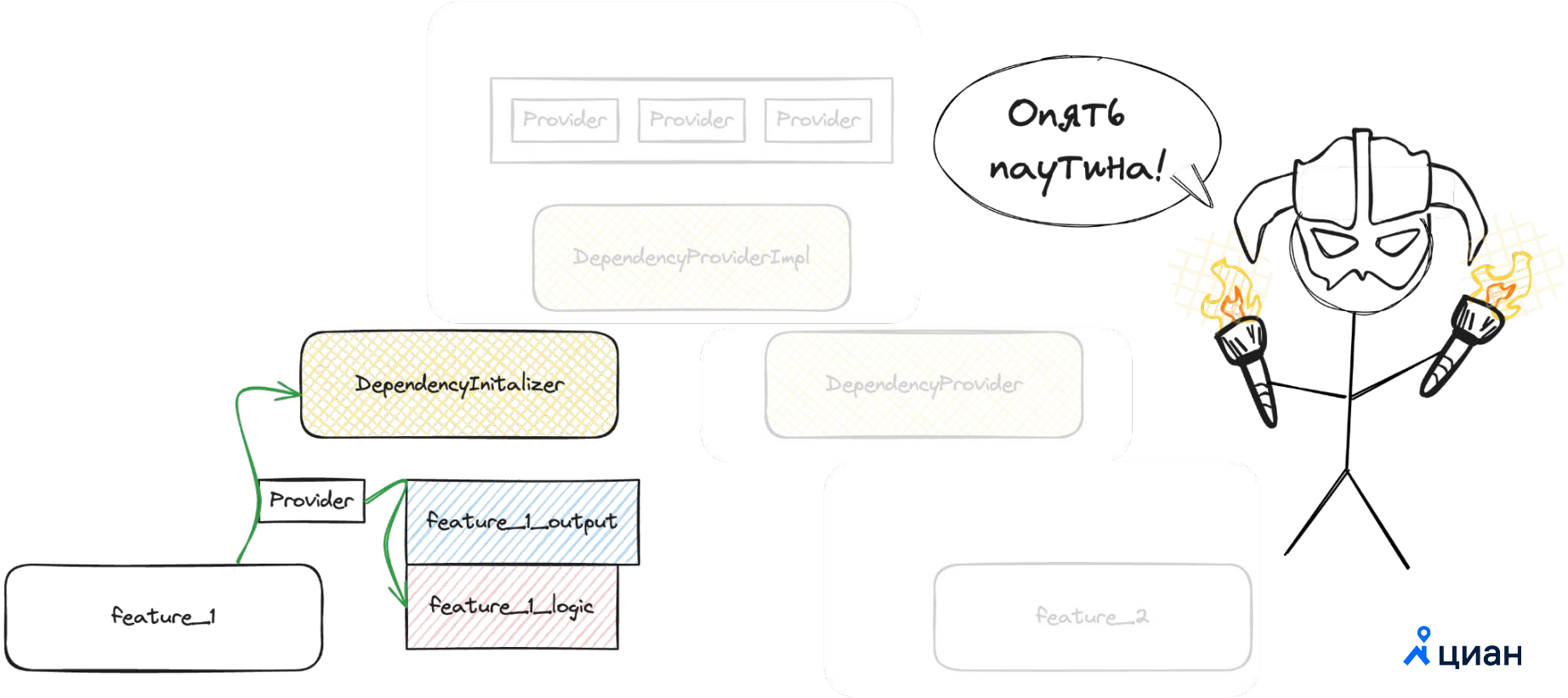


циан

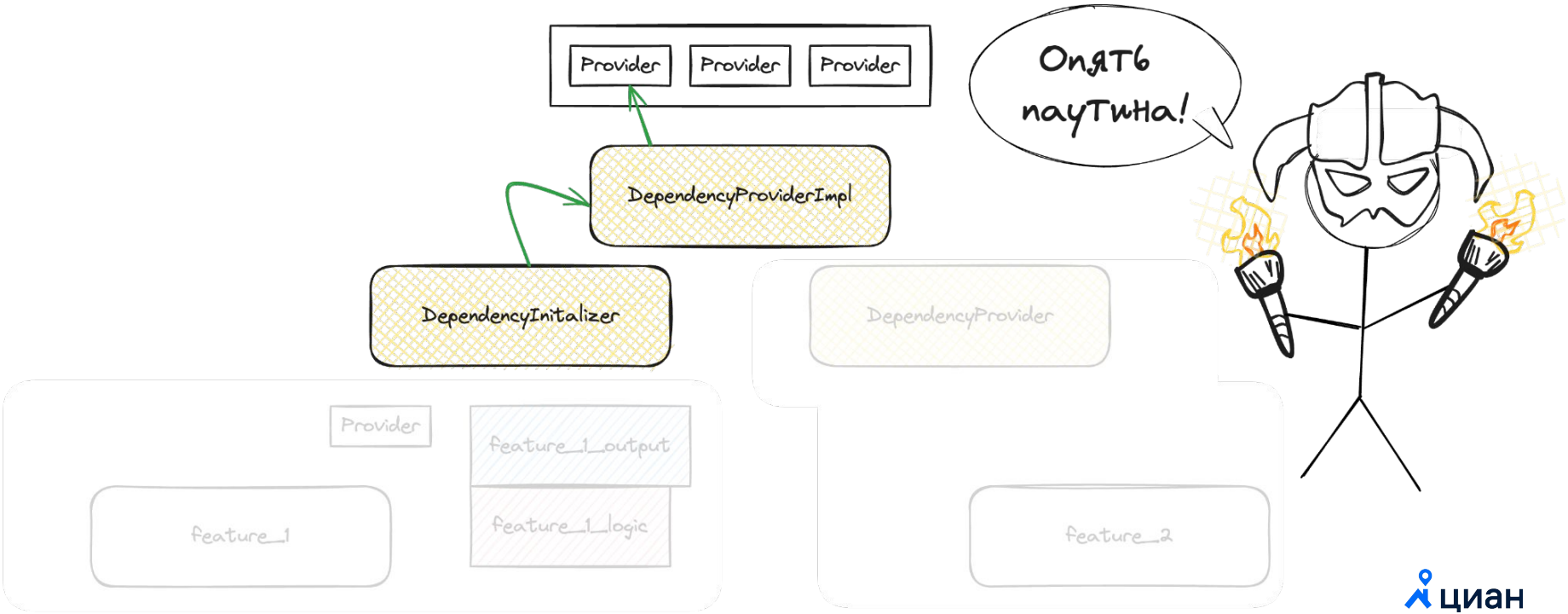
DependencyProvider и DependenciesInitializer



DependencyProvider и DependenciesInitializer

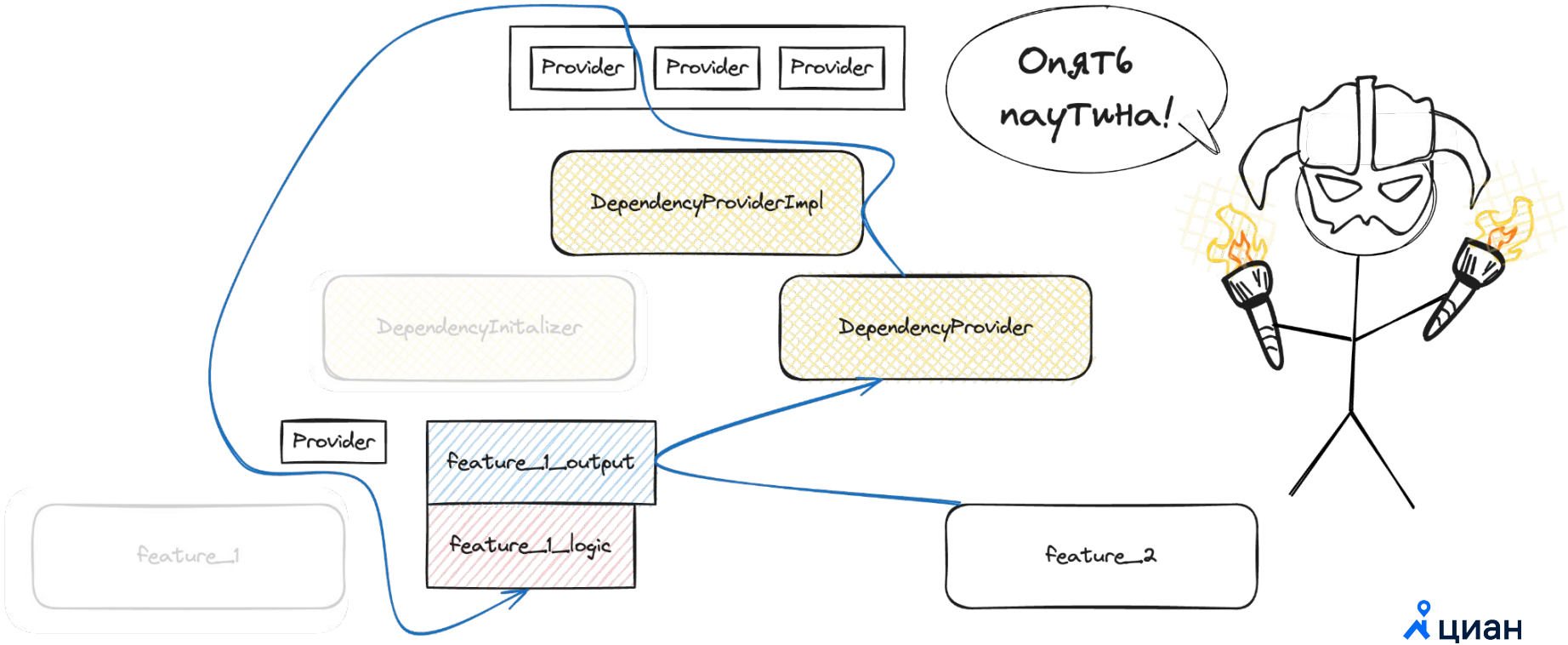


DependencyProvider и DependenciesInitializer



циан

DependencyProvider и DependenciesInitializer



циан

DependenciesInitializer


```
interface DependenciesInitializer {

    fun <T : ProvidableDependency> register(
        key: KClass<T>,
        strategy: DependencyCacheStrategy, // SINGLETON, WEAK_REFERENCE, NONE
        initializer: () -> T
    )
}
```

DependenciesInitializer. Использование


```
dependenciesInitializer.register(
    key = Feature1Dependencies::class,
    strategy = DependencyCacheStrategy.WEAK_REFERENCE,
    initializer = { Feature1ComponentFactory.createComponent() },
)
```

циан

DependencyProvider

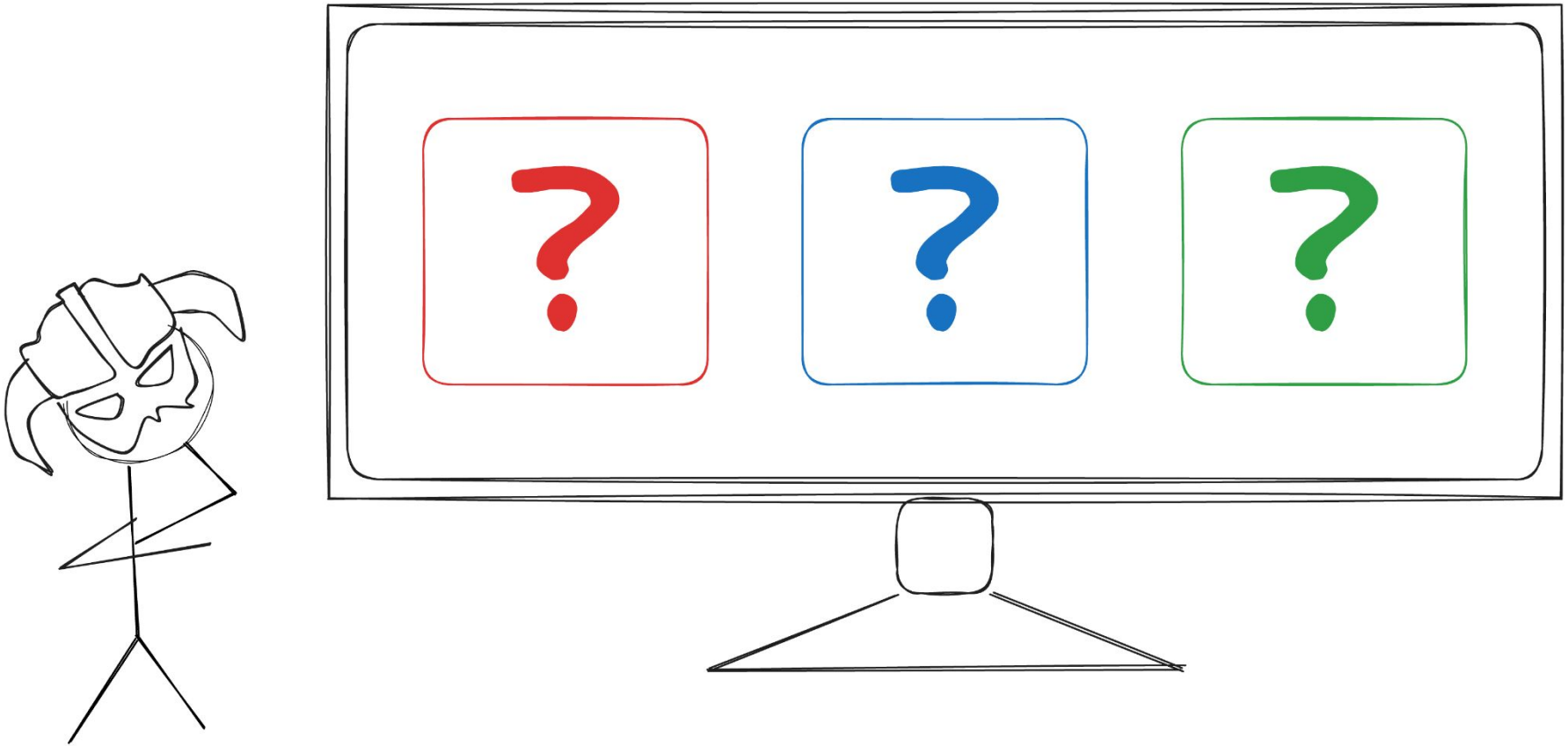

```
interface DependenciesProvider {

   fun <T : ProvidableDependency> provide(clazz: Class<T>): T

   fun <T : ProvidableDependency> provide(clazz: KClass<T>): T
}

inline fun <reified T : ProvidableDependency> DependenciesProvider.provide(): T {
   return provide(T::class)
}
```

циан

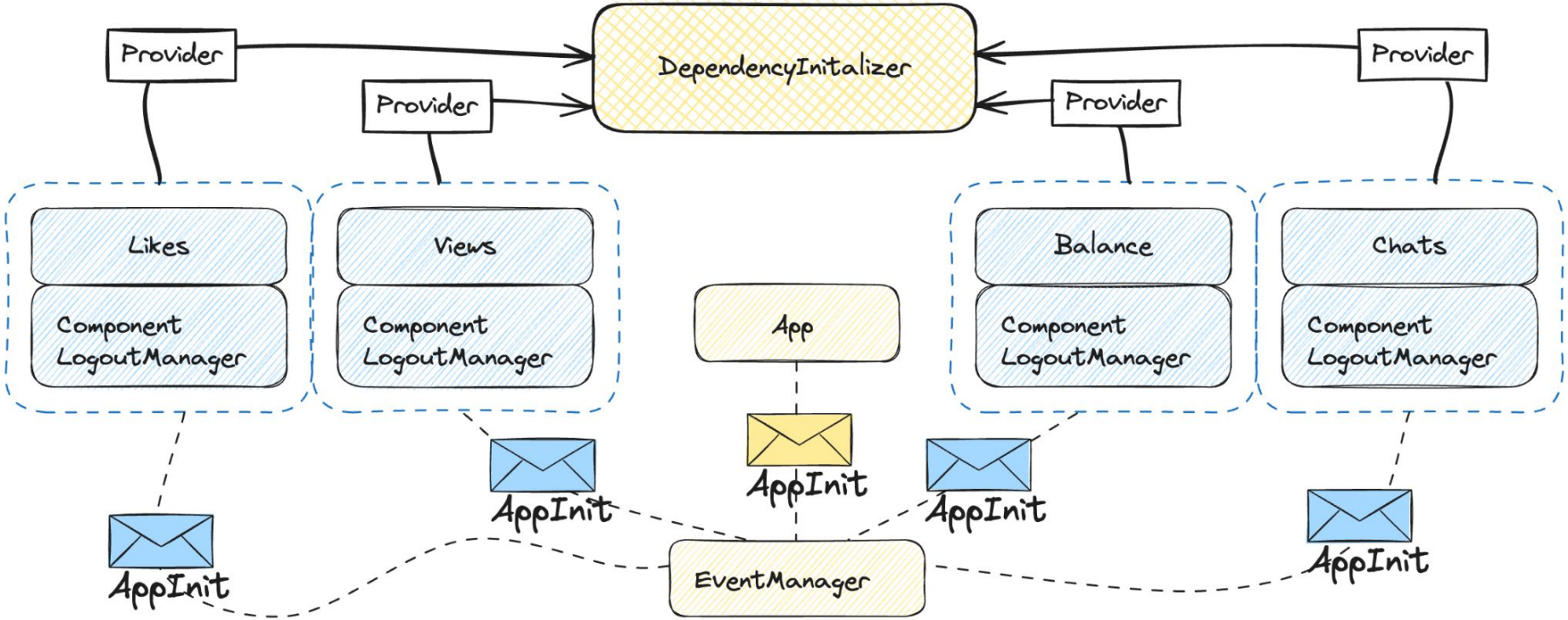
DependencyProvider. Использование


```
dependenciesProvider.provide<Feature1Dependencies>()
```

А как общаться между модулями?

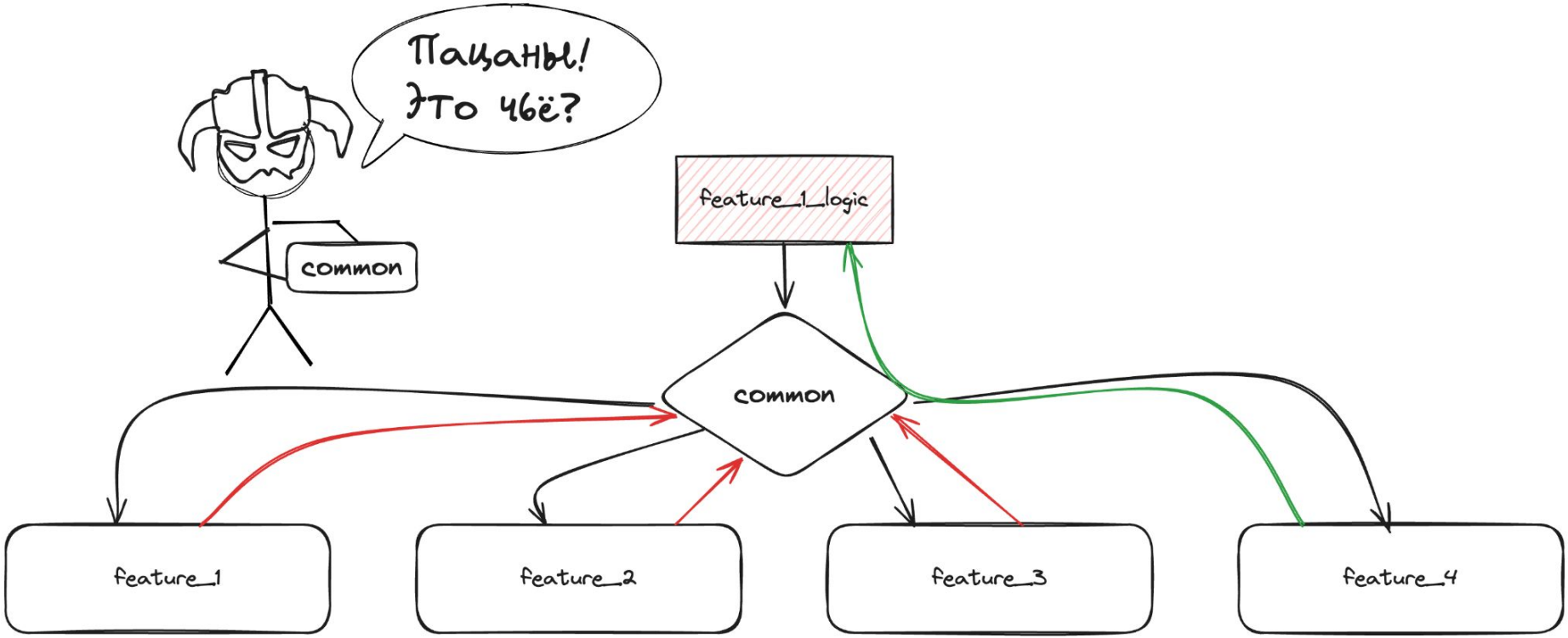


Глобальные события

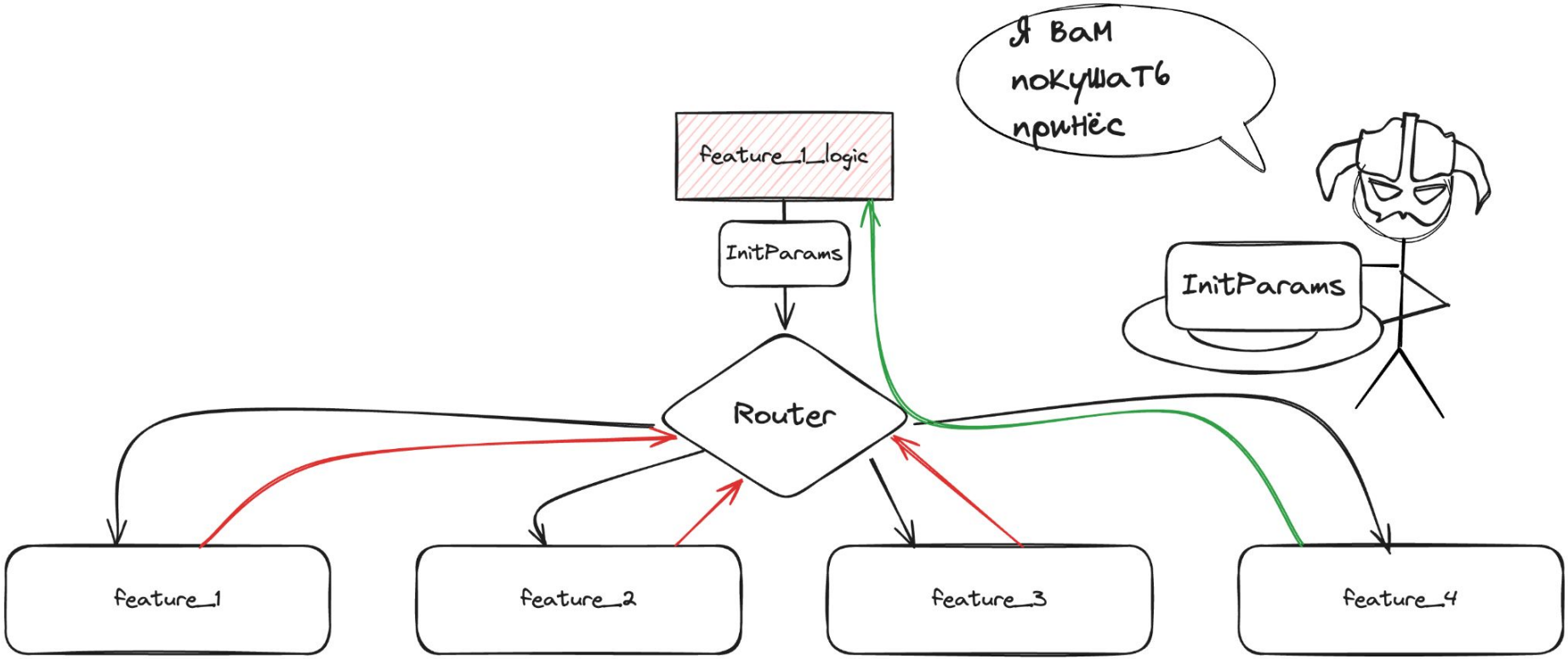


циан

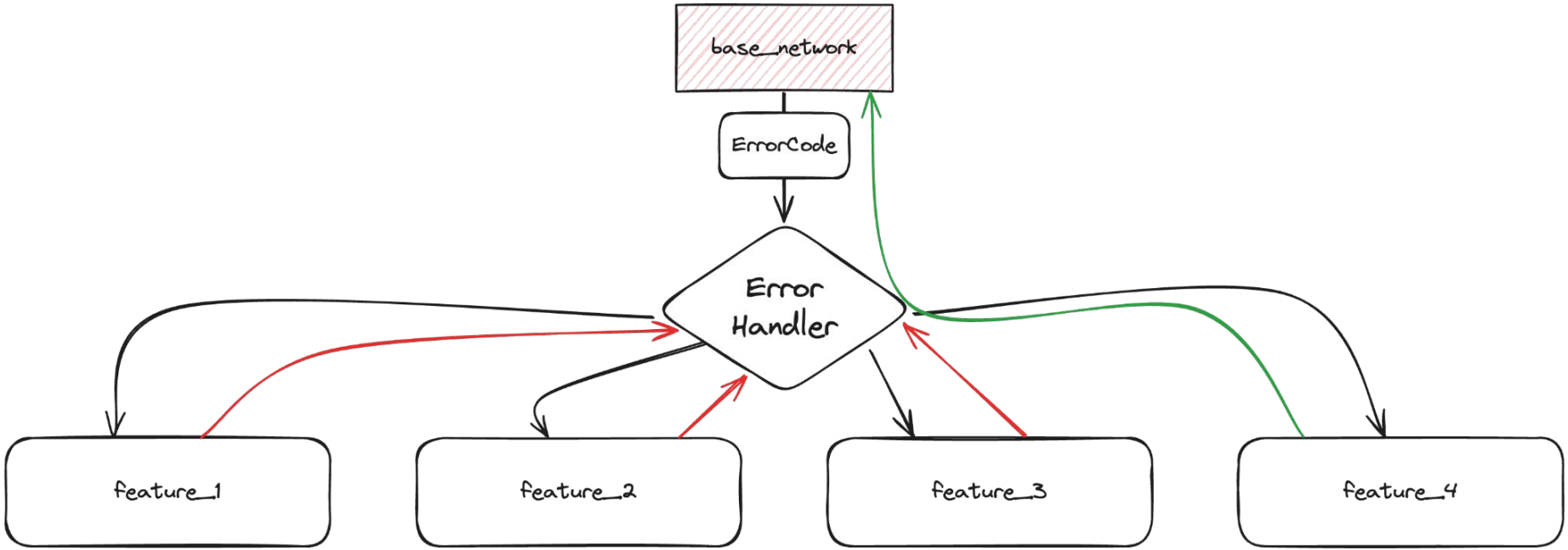
Как что-то получить?



Пример. Роутинг

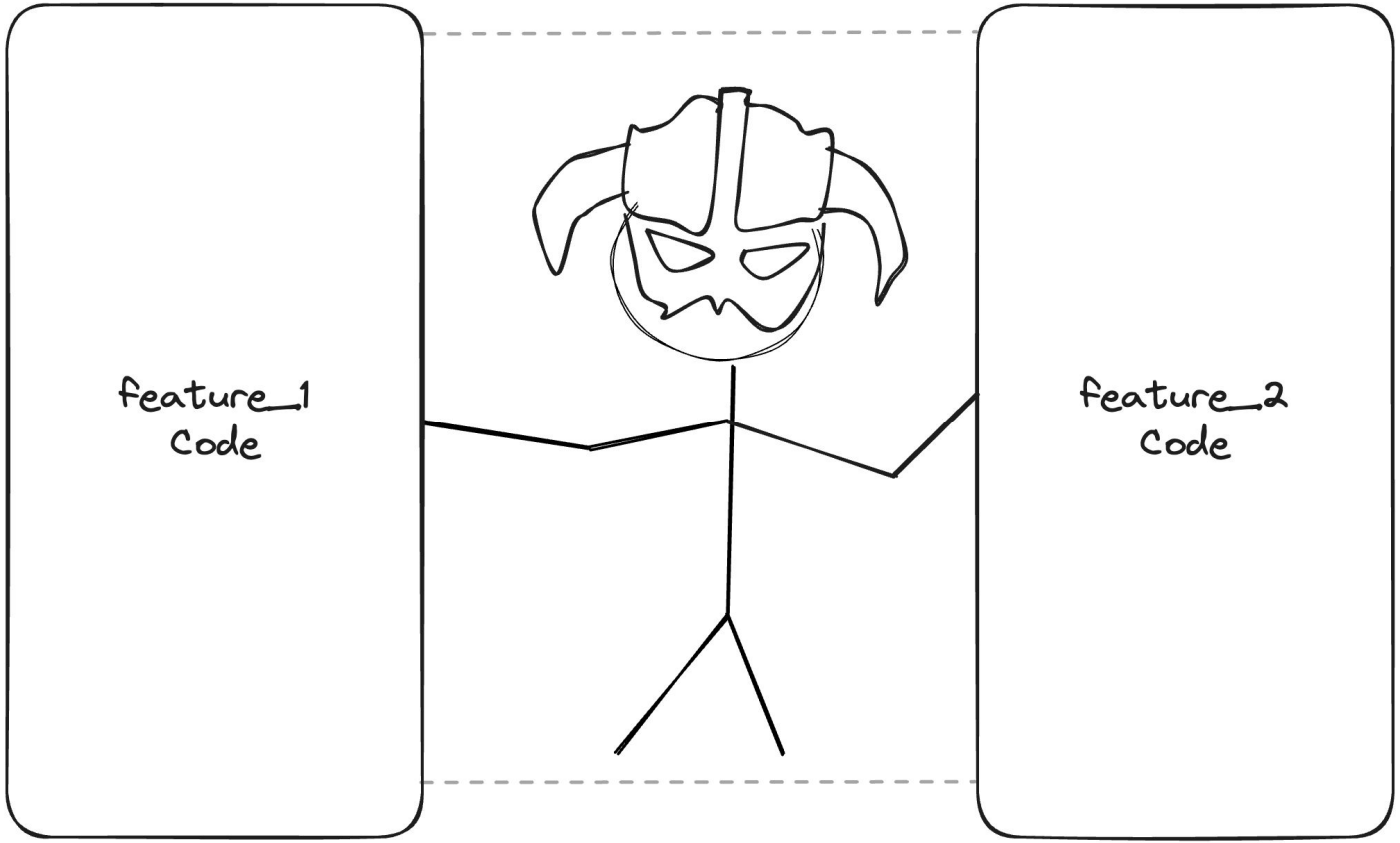


Пример. Обработка ошибок 400



циан

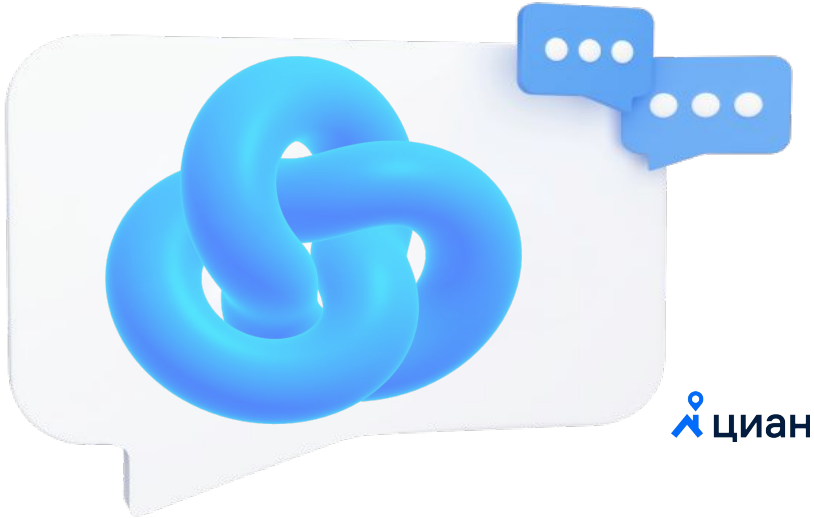
С точки зрения кода всё ок.



циан

Про что сегодня поговорим

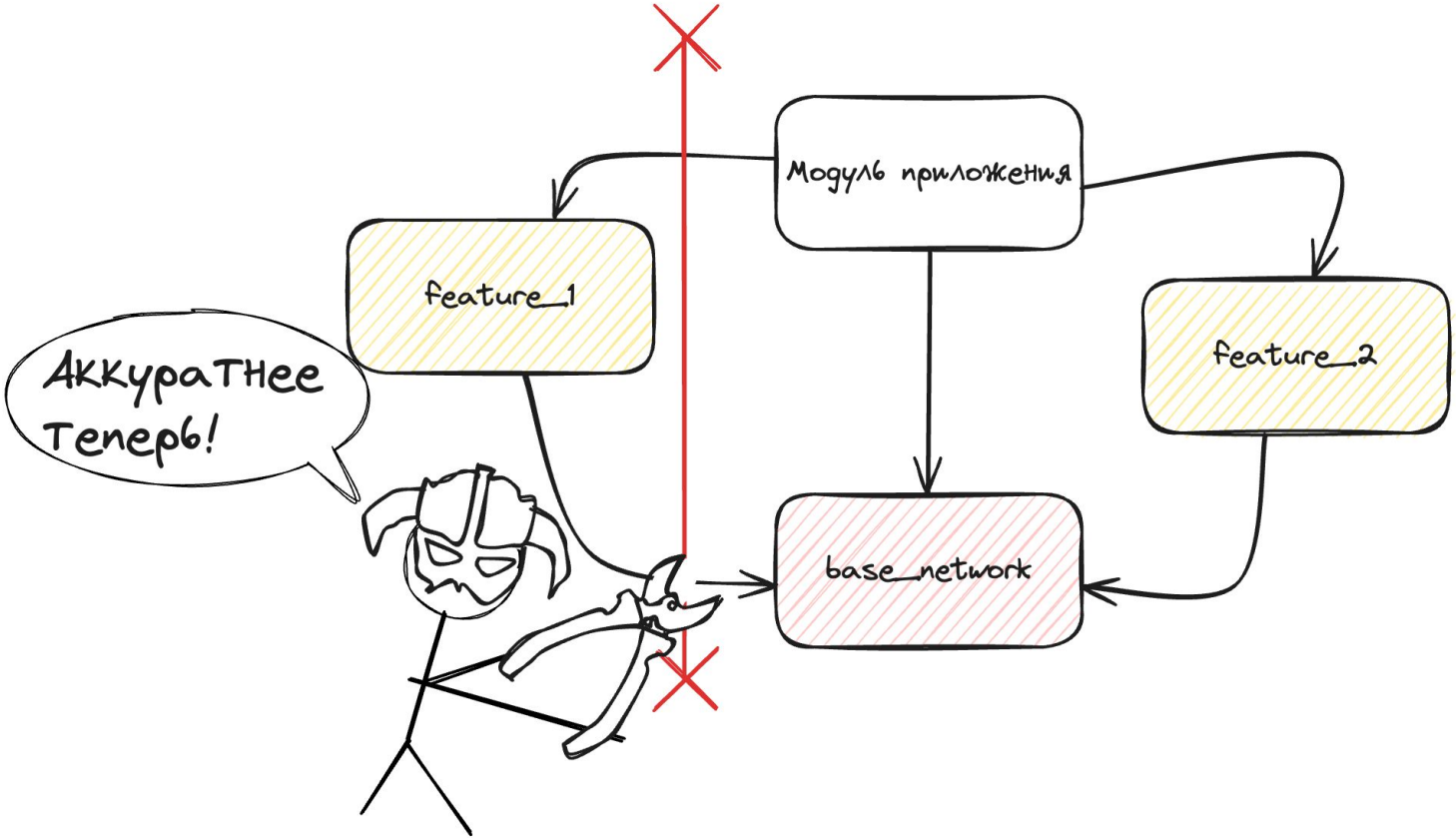
- ~~Что такое плагиновидная структура модулей~~
- ~~Как реализовать плагиновидную структуру~~
- Какие могут возникнуть проблемы
- Как из этого сделать универсальное демо-приложение



циан

Выключаем модули

Отключаем модуль

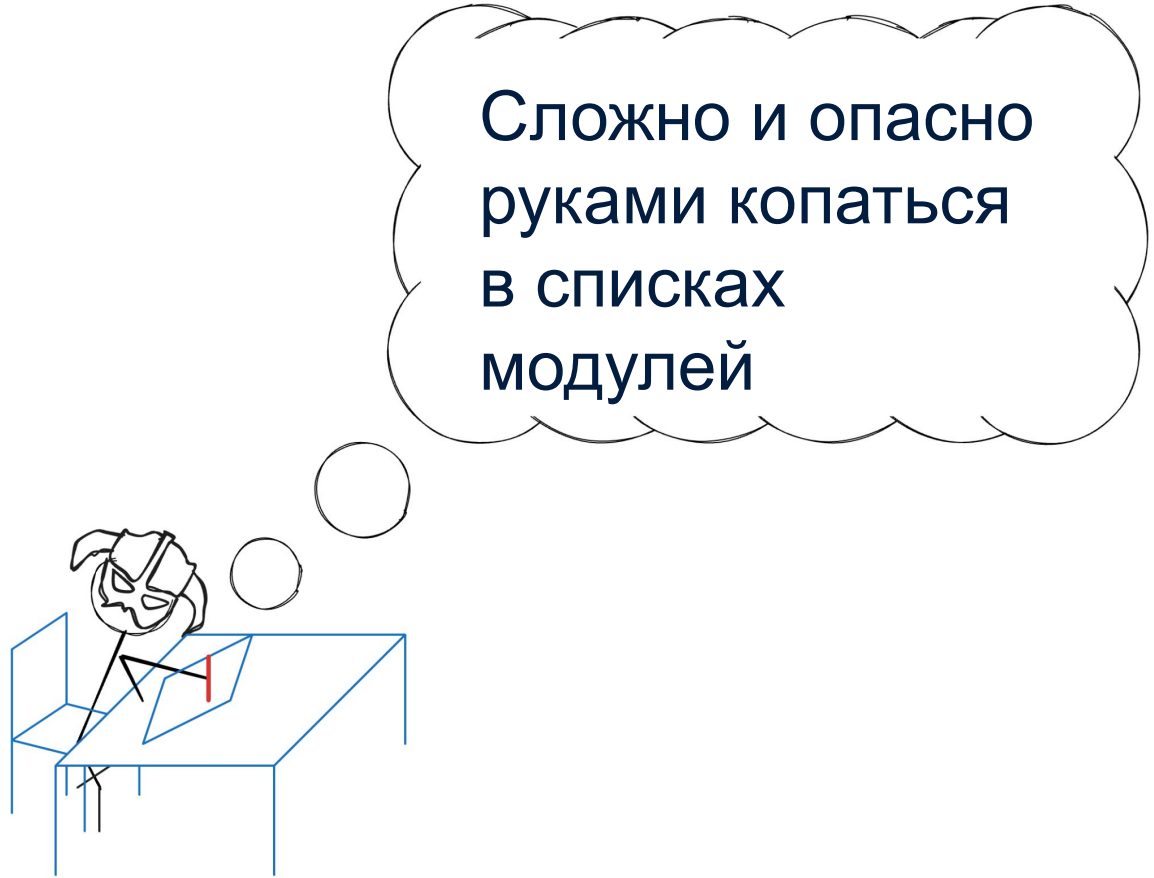


settings.gradle и build.gradle


```
// settings.gradle.kts
// include(":feature_1")/
// project(":feature_1").projectDir = File(rootDir, "modules/feature/feature_1")

// app/build.gradle.kts
// implementation(project(":feature_1"))
```

циан

Запоминаем проблему 1.

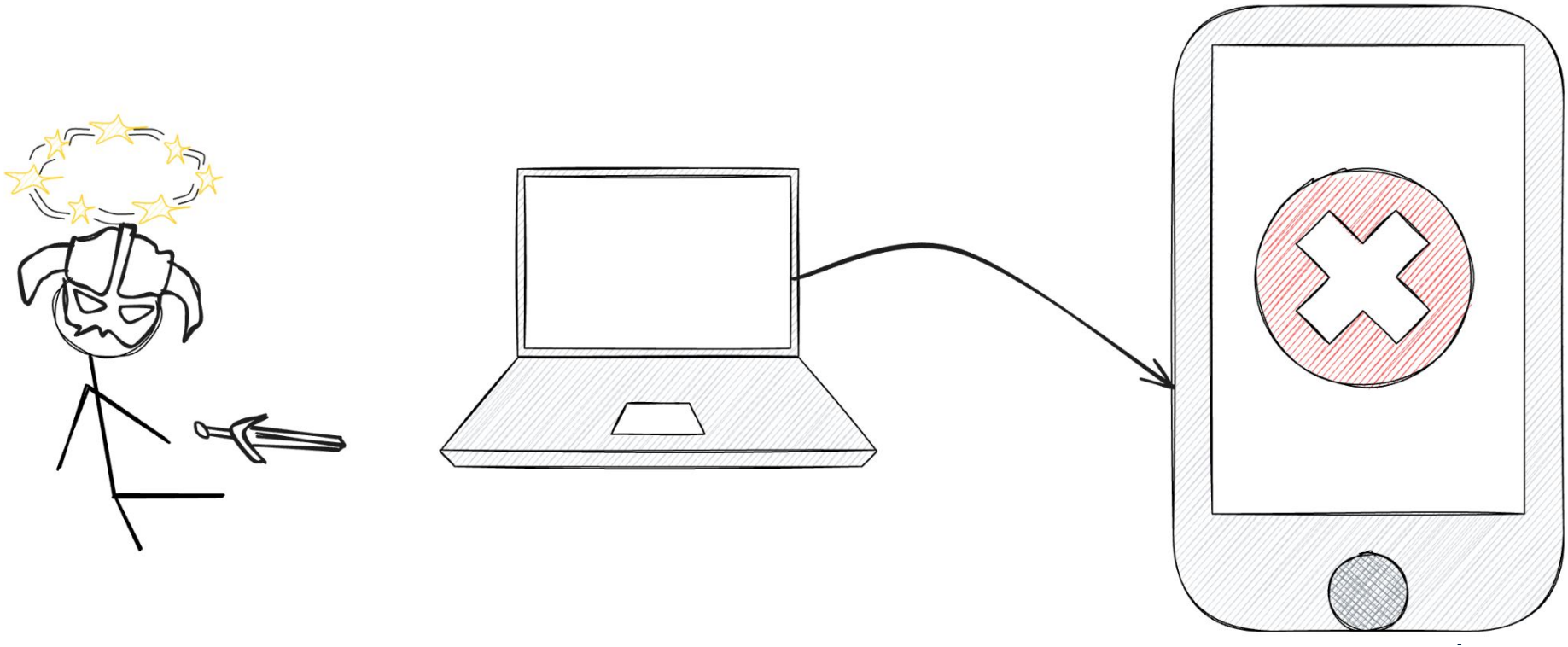


Отключаем ещё один


```
// settings.gradle.kts
// include(":feature_2")
// project(":feature_2").projectDir = File(rootDir, "modules/feature/feature_2")

// app/build.gradle.kts
// implementation(project(":feature_2"))
```

циан

Запускаем и получаем краш

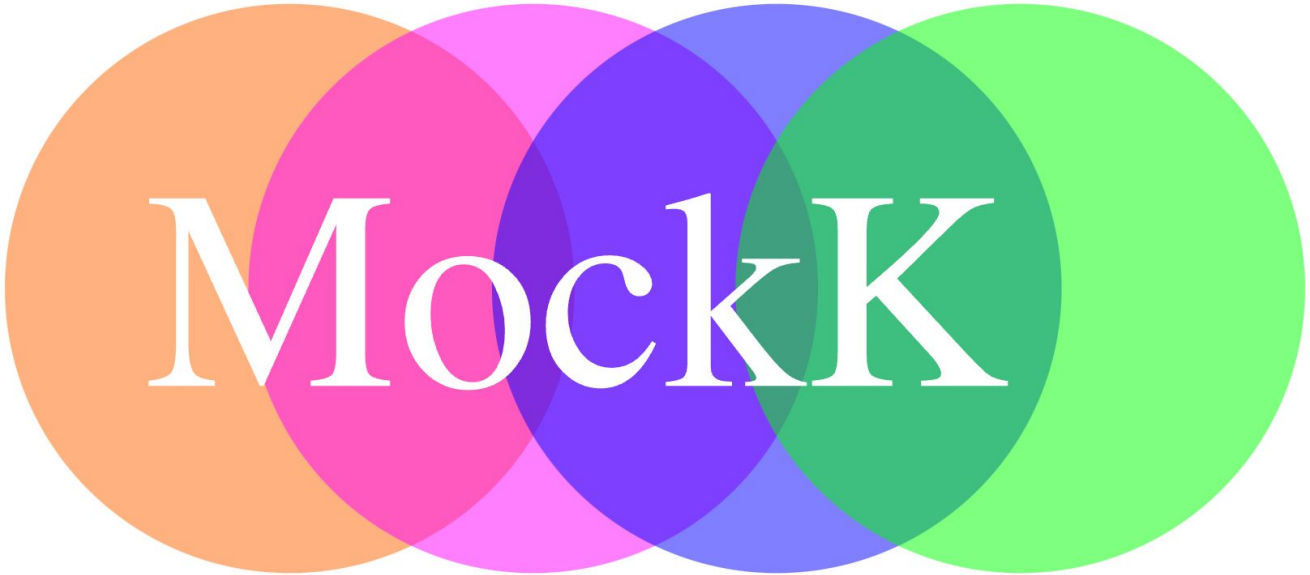


Что делать?


```
// ru.cian.base.di.DependenciesProviderImpl
if (providerContainer == null) {
   throw IllegalArgumentException("Provider for key '$key' not found")
}
```

циан

Используем моки



циан

Заставляем DependencyProvider возвращать моки

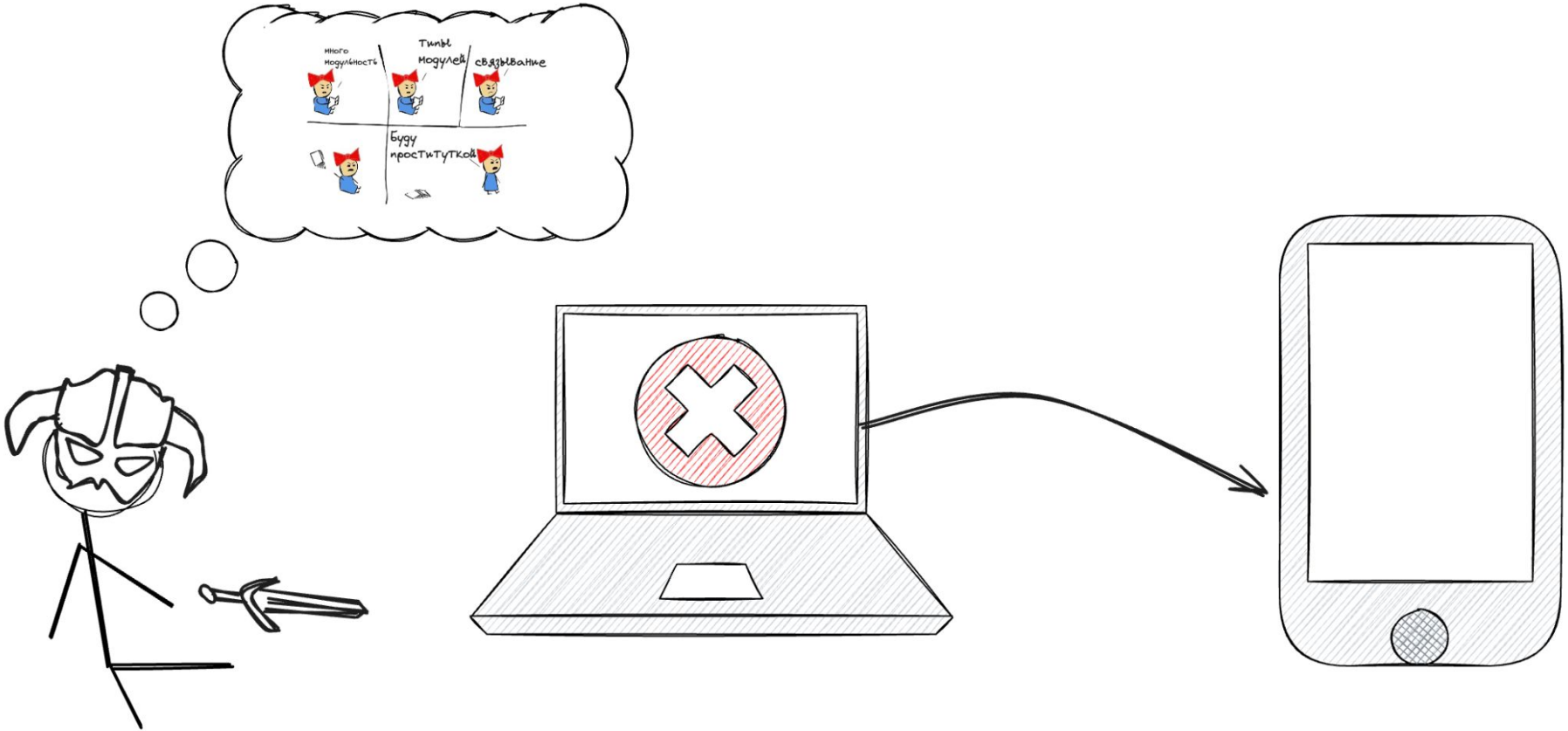

```
private class MockDependencyProvider(
    private val origin: DependencyProvider
) : DependenciesProvider {

    override fun <T : ProvidableDependency> provide(clazz: KClass<T>): T {
        return try {
            origin.provide(clazz)
        } catch (throwable: IllegalArgumentException) {
            mockByClass(clazz)
        }
    }
}
```

Отключаем ещё несколько модулей

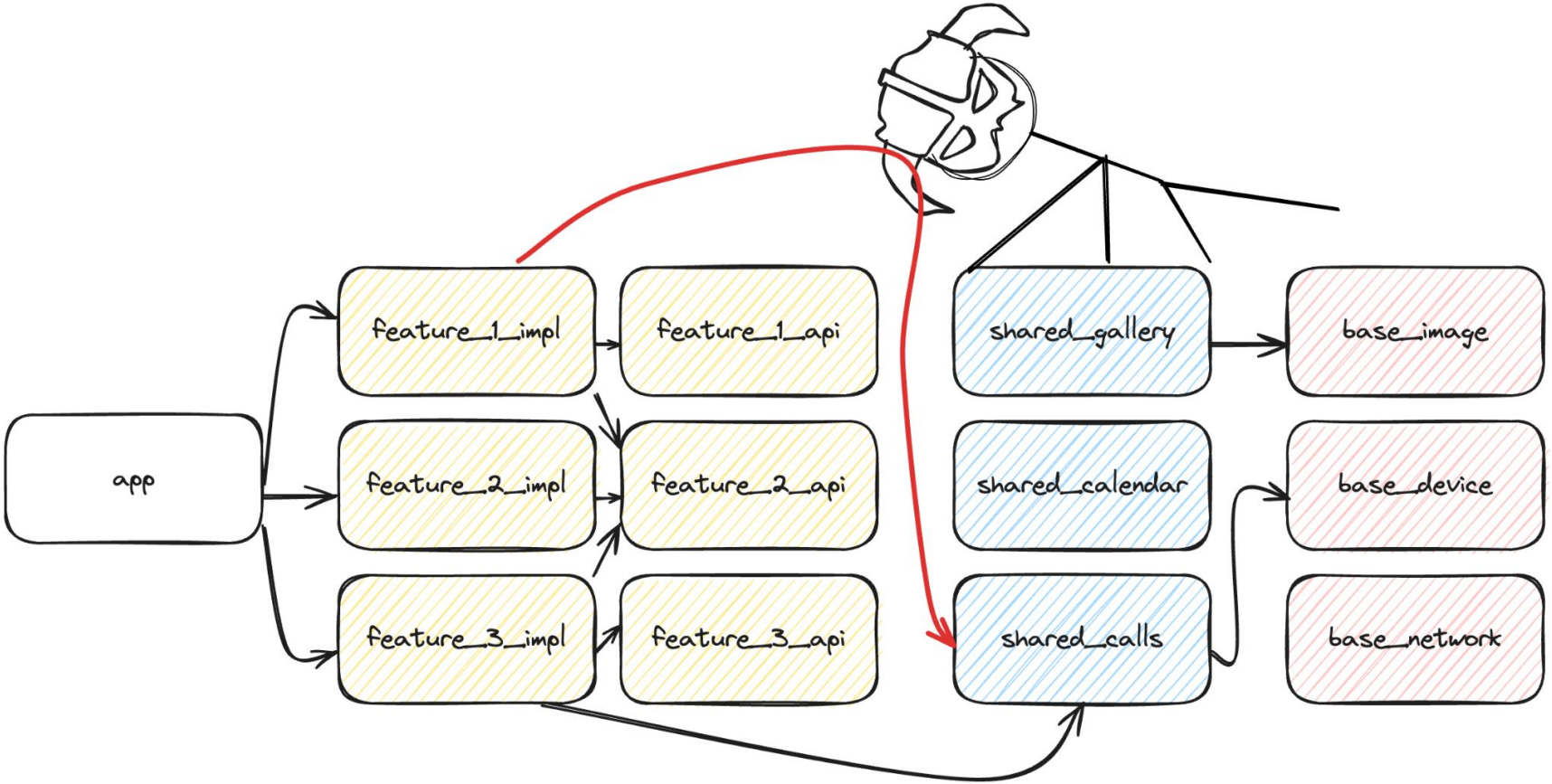

```
// settings.gradle.kts
include(":shared_calls")
project(":shared_calls").projectDir = File(rootDir, "modules/shared/shared_calls")
include(":feature_3")
project(":feature_3").projectDir = File(rootDir, "modules/feature/feature_3")

// app/build.gradle.kts
implementation(project(":shared_calls"))
implementation(project(":feature_3"))
```

циан

Оно не собирается

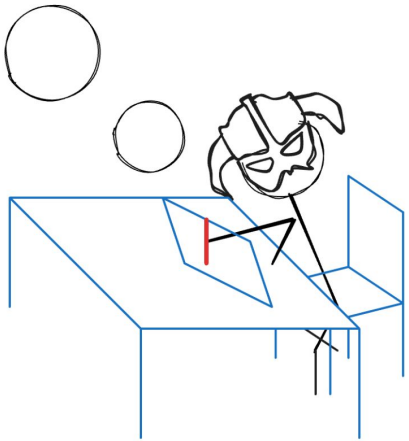


Иерархия модулей.



Запоминаем проблему 2.

Модули нужно отключать с учётом иерархии

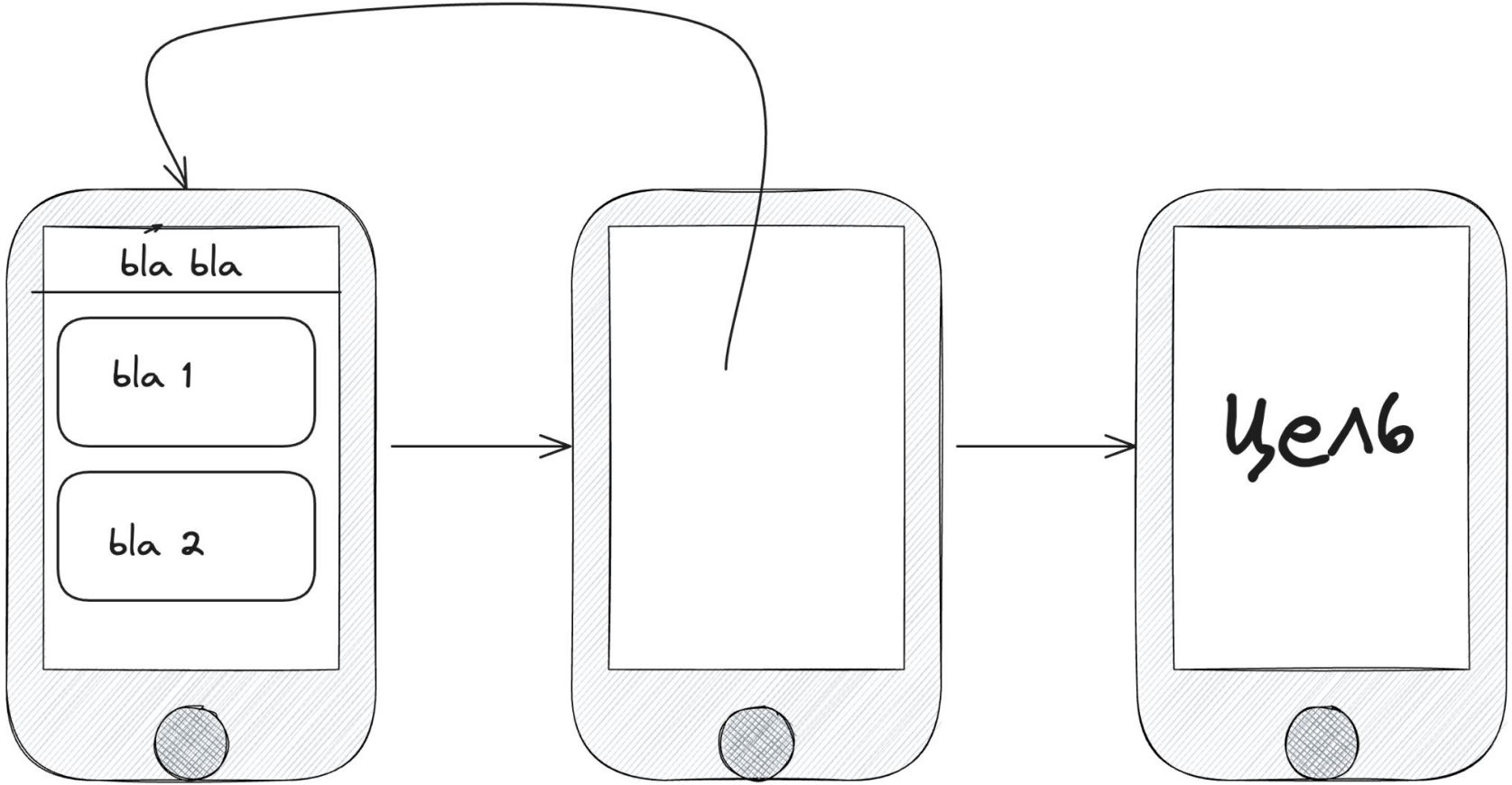


циан

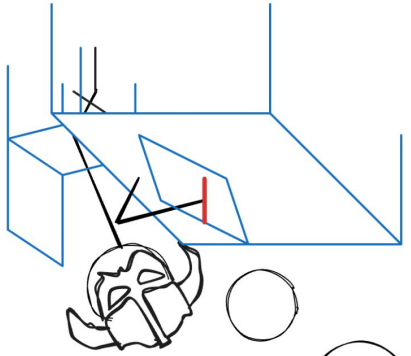
Выключаем модули с учётом иерархии


```
// settings.gradle.kts
include(":shared_calls")
project(":shared_calls").projectDir = File(rootDir, "modules/shared/shared_calls")
include(":feature_3")
project(":feature_3").projectDir = File(rootDir, "modules/feature/feature_3")
include(":feature_4")
project(":feature_4").projectDir = File(rootDir, "modules/feature/feature_4")

// app/build.gradle.kts
implementation(project(":shared_calls"))
implementation(project(":feature_3"))
implementation(project(":feature_4"))
```

Увы. Мы не можем попасть на экран



Запоминаем проблему 3

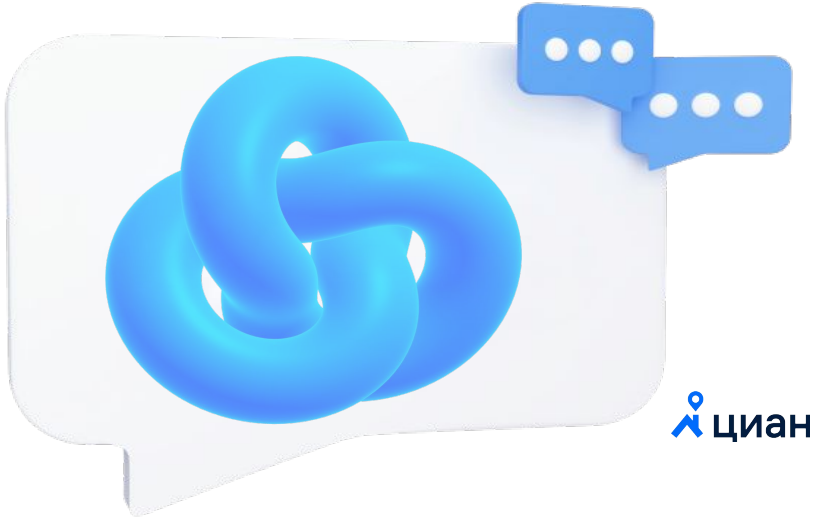


Нужно попадать на экран независимо от выбранных модулей

циан

Про что сегодня поговорим

- ~~Что такое плагиновидная структура модулей~~
- ~~Как реализовать плагиновидную структуру~~
- ~~Какие могут возникнуть проблемы~~
- Как из этого сделать универсальное демо-приложение



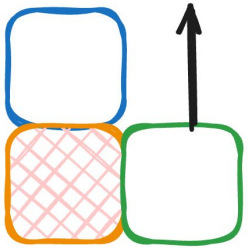
циан

Решение проблем

Напоминаю зачем нам это

Ничего лишнего

Нет лишних связей при компиляции и в то же время нет лишнего маппинга



Демо-приложения

Можно с лёгкостью собрать базу только с теми фичами, что нужны

Скорость

Скорость работы Android Studio, конфигурации и компиляции

Решение проблем мы назвали SuperDemoApp

циан

Напоминаю наши проблемы

1. Сложно и опасно руками копаться в списках модулей
2. Модули нужно отключать с учётом иерархии
3. Нужно попадать на экран независимо от выбранных модулей

Проблема 1. Сложно и опасно руками копаться в списках модулей

Стандартный settings.gradle


```
// settings.gradle.kts
include(":shared_calls")
project(":shared_calls").projectDir = File(rootDir, "modules/shared/shared_calls")
include(":feature_1")
project(":feature_1").projectDir = File(rootDir, "modules/feature/feature_1")
include(":feature_2")
project(":feature_2").projectDir = File(rootDir, "modules/feature/feature_2")
include(":feature_3")
project(":feature_3").projectDir = File(rootDir, "modules/feature/feature_3")
include(":feature_4")
project(":feature_4").projectDir = File(rootDir, "modules/feature/feature_4")
```

Наш settings.gradle


```
// settings.gradle.kts
includeSharedModule(":shared_calls")
includeFeatureModule(":feature_1")
includeFeatureModule(":feature_2")
includeFeatureModule(":feature_3")
includeFeatureModule(":feature_4")

fun includeSharedModule(moduleName: String) { ... }
fun includeFeatureModule(moduleName: String) { ... }
```

Кто-то решал такую проблему?

циан

Focus от Dropbox

README Apache-2.0 license

🧘 Focus

A Gradle plugin that generates module-specific `settings.gradle` files, allowing you to focus on a specific feature or module without needing to sync the rest of your monorepo.

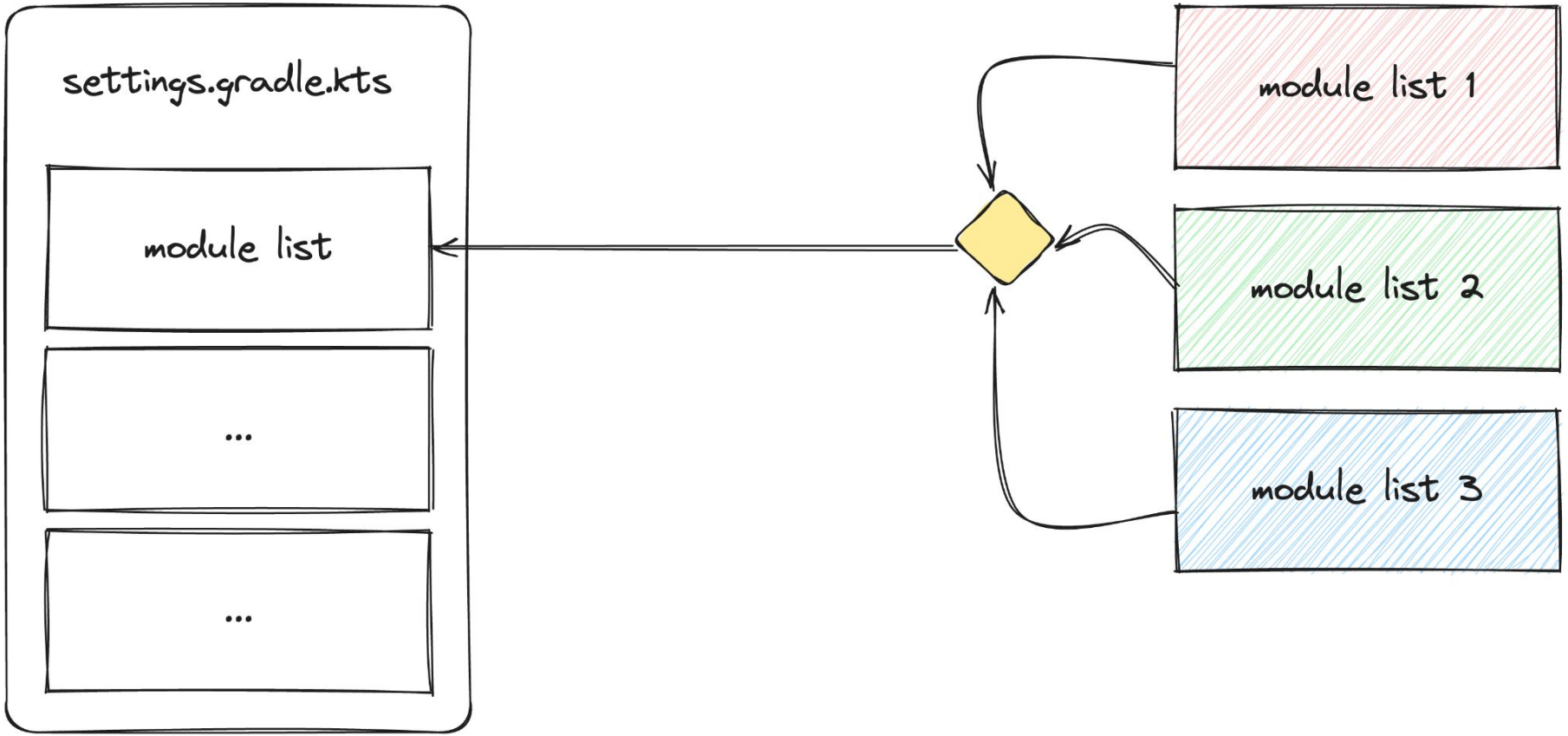
The Focus plugin evaluates your project setup and creates a unique `settings.gradle` file for the module you want to focus on, which only includes the dependencies required by that module. It then creates a `.focus` file that references the currently focused module.

With these files in place only the modules that you need will be configured by Gradle when you sync your project. Deleting the `.focus` file, which can be done using the `clearFocus` task, will revert to using the includes file to configure your entire project.

Но она слишком мала

циан

Список модулей отдельно от settings.gradle



Список модулей через gradle extras


```
// settings/app.gradle.kts
val feature: MutableList<String> by extra
val sharedUi: MutableList<String> by extra

with(feature) {
    add("feature_1")
    add("feature_2")
    add("feature_3")
    add("feature_4")
}

with(sharedUi) {
    add("shared_calls")
}
```

Используем правила

```
// module_rules.gradle.kts
val rules by extra(mutableMapOf<String, (String) -> List<Pair<String, String>>>())

rules["sharedUi"] = { name -> listOf(":$name" to "modules/shared-ui/$name") }
rules["feature"] = { name ->
    if (name.contains("/")) {
        listOf(":${name.replace("/", "_")}" to "modules/feature/$name")
    } else {
        listOf(
            ":$name_api" to "modules/feature/$name/api",
            ":$name_impl" to "modules/feature/$name/impl",
        )
    }
}
```

Добавляем модули


```
// include_modules.gradle.kts
val rules: MutableMap<String, (String) -> List<Pair<String, String>>> by extra

val allModules = rules.map { (ruleName, rule) ->
   val modulesInRule = extra[ruleName] as MutableList<String>
   modulesInRule.map { path -> rule(path) }.flatten()
}.flatten()

allModules.forEach { (moduleName, moduleDir) ->
   val moduleDirFile = File(rootDir, moduleDir)
   if (moduleDirFile.exists()) {
       include(moduleName)
       project(moduleName).projectDir = moduleDirFile
   }
}
```

циан

Реализация в settings.gradle


```
// settings.gradle.kts
apply(from = "$rootDir/build_scripts/module_rules.gradle.kts")
apply(from = "$rootDir/build_scripts/settings/app.gradle.kts")
apply(from = "$rootDir/build_scripts/include_modules.gradle.kts")
```

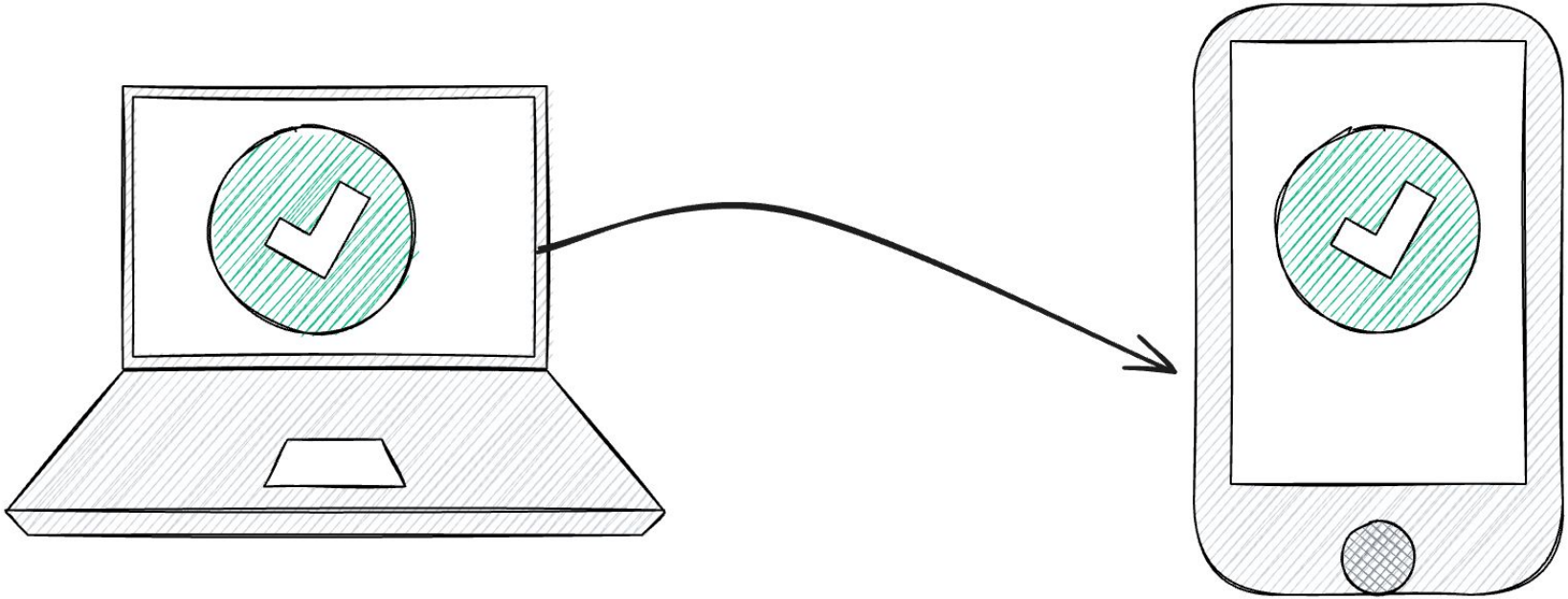
Добавляем возможность использовать другой файл


```
// settings.gradle.kts
val settingsFile = if (providers.gradleProperty("settings.file").isPresent) {
   "$rootDir/${providers.gradleProperty("settings.file").get()}"
} else {
   "$rootDir/build_scripts/settings/app.gradle.kts"
}

apply(from = "$rootDir/build_scripts/module_rules.gradle.kts")
apply(from = settingsFile)
apply(from = "$rootDir/build_scripts/include_modules.gradle.kts")
```

Используем это

```
// На один запуск через Terminal
./gradlew assembleDebug \
-Psettings.file=/build_scripts/settings/my_module_list.gradle.kts

// На всё время работы проекта (для Android Studio)
// gradle.properties или local.properties
settings.file=build_scripts/settings/my_module_list.gradle.kts
```

Запускаем - работает



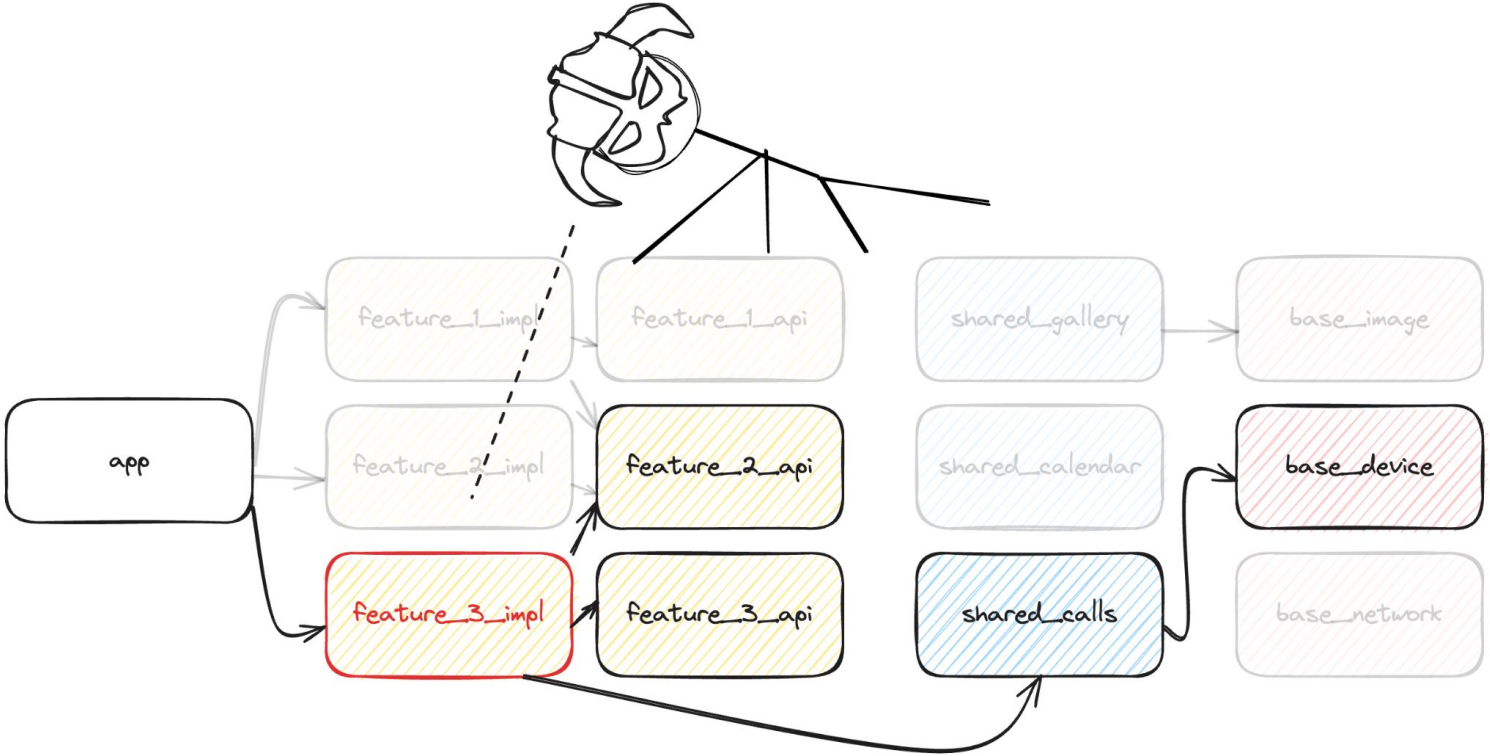
циан

Напоминаю наши проблемы

1. ~~Сложно и опасно руками копаться в списках модулей~~
2. Модули нужно отключать с учётом иерархии
3. Нужно попадать на экран независимо от выбранных модулей

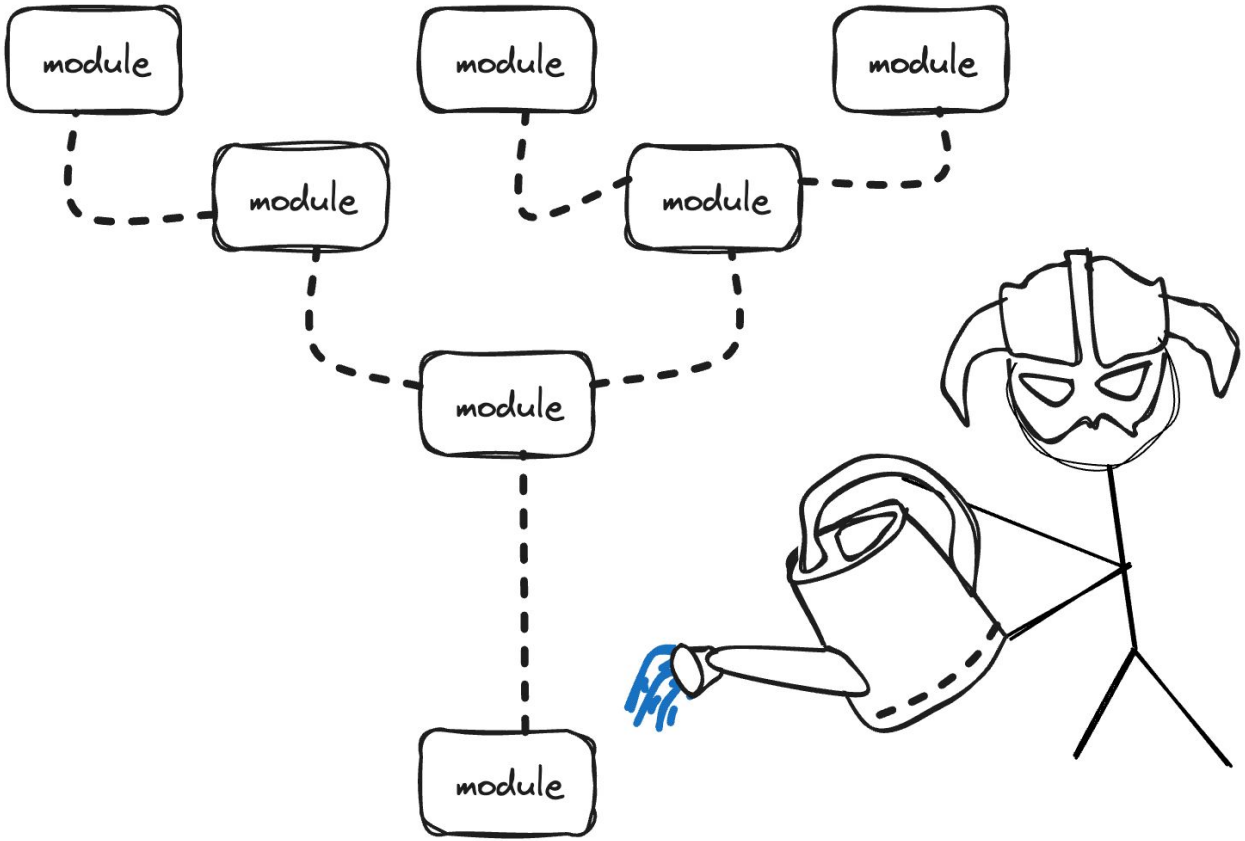
Проблема 2. Иерархия

При выборе модуля, ненужные отключаются

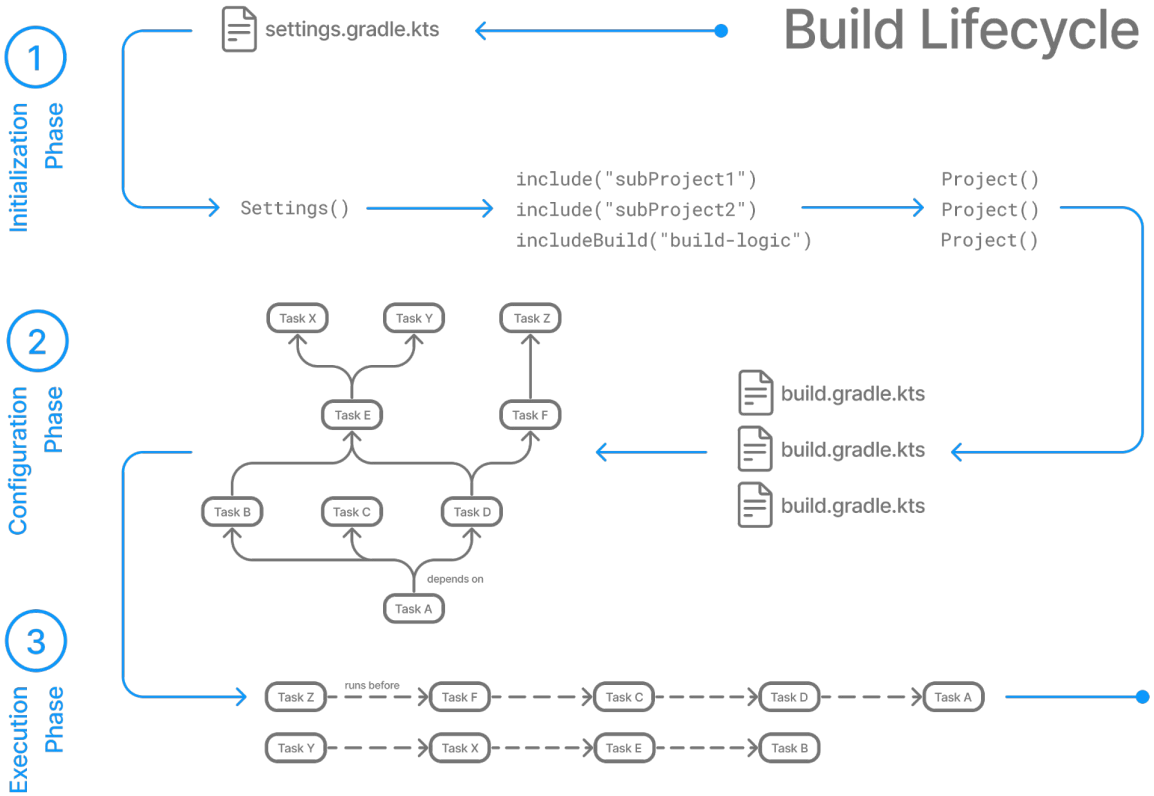


циан

Для начала нам надо понять иерархию



Конфигурация в Gradle



циан

Но она медленная

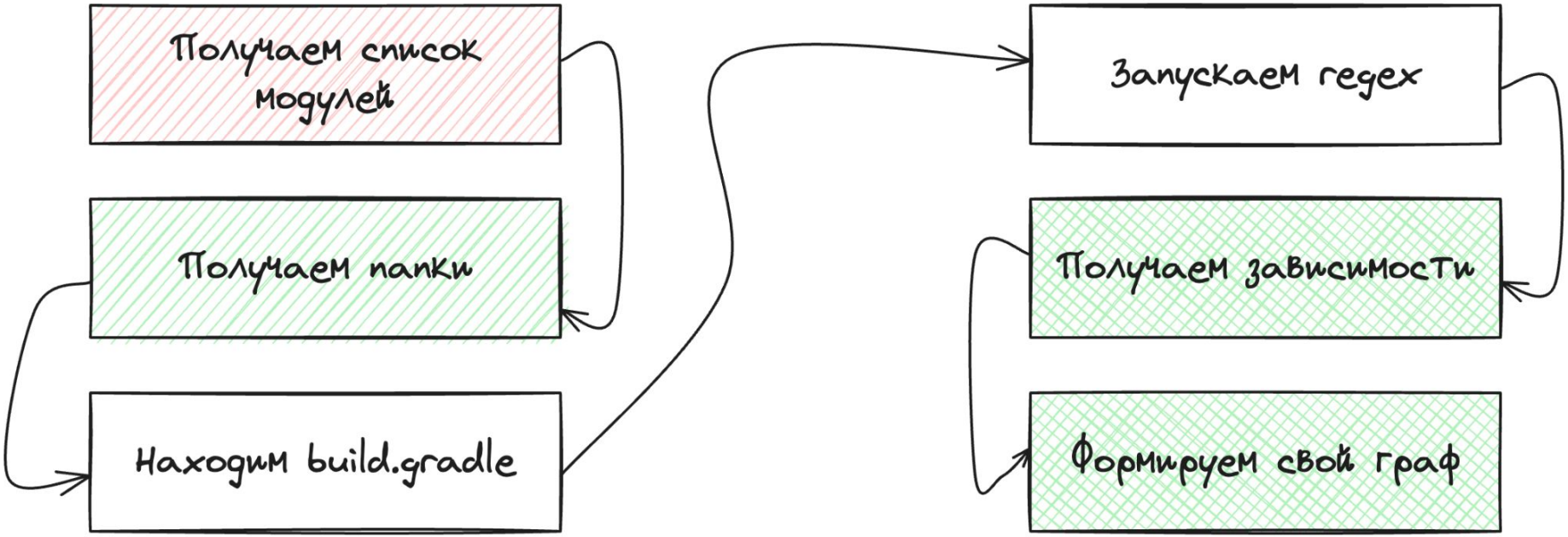
1. Компилирует kts и выполняет build.gradle скрипты
2. Выкачивает зависимости
3. Строит граф Tasks
4. Создаёт Project
5. …

Делаем костыль, кривой, но быстрый

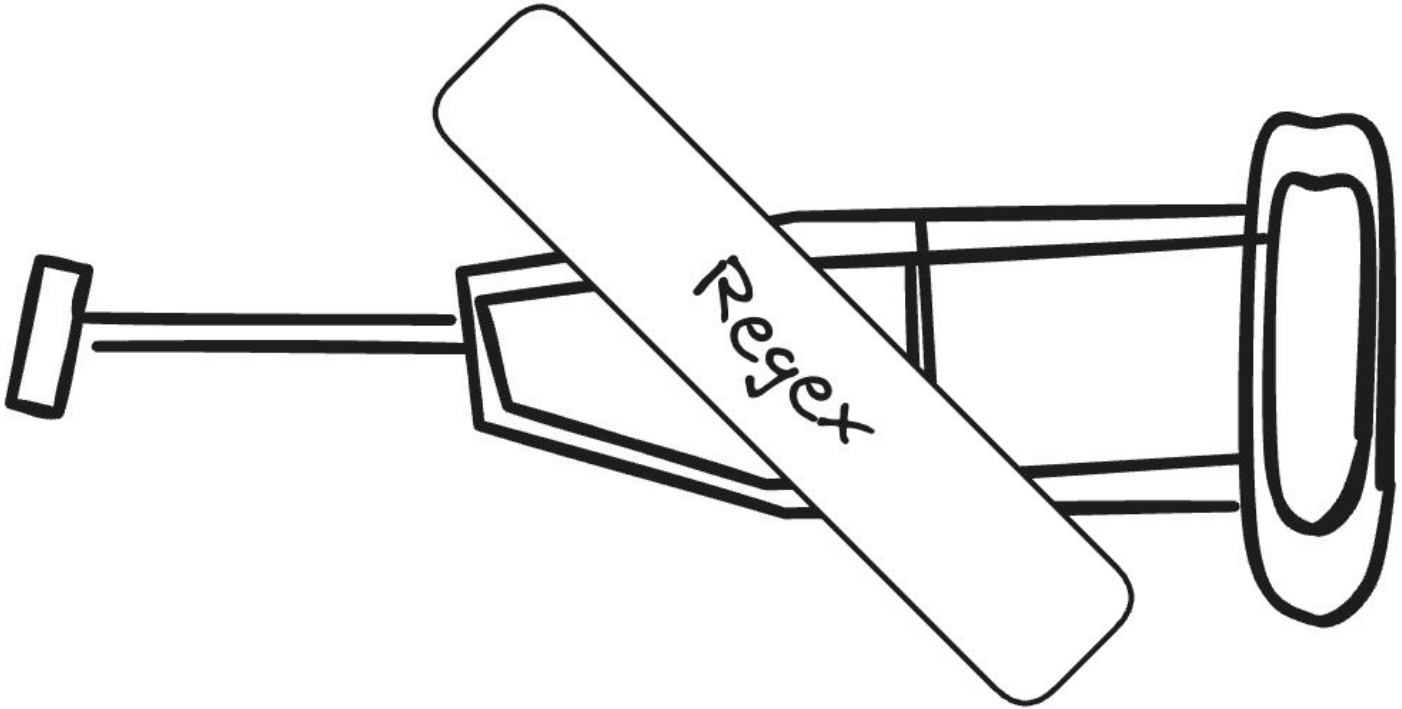
циан

Используем список и правила

Алгоритм костыля

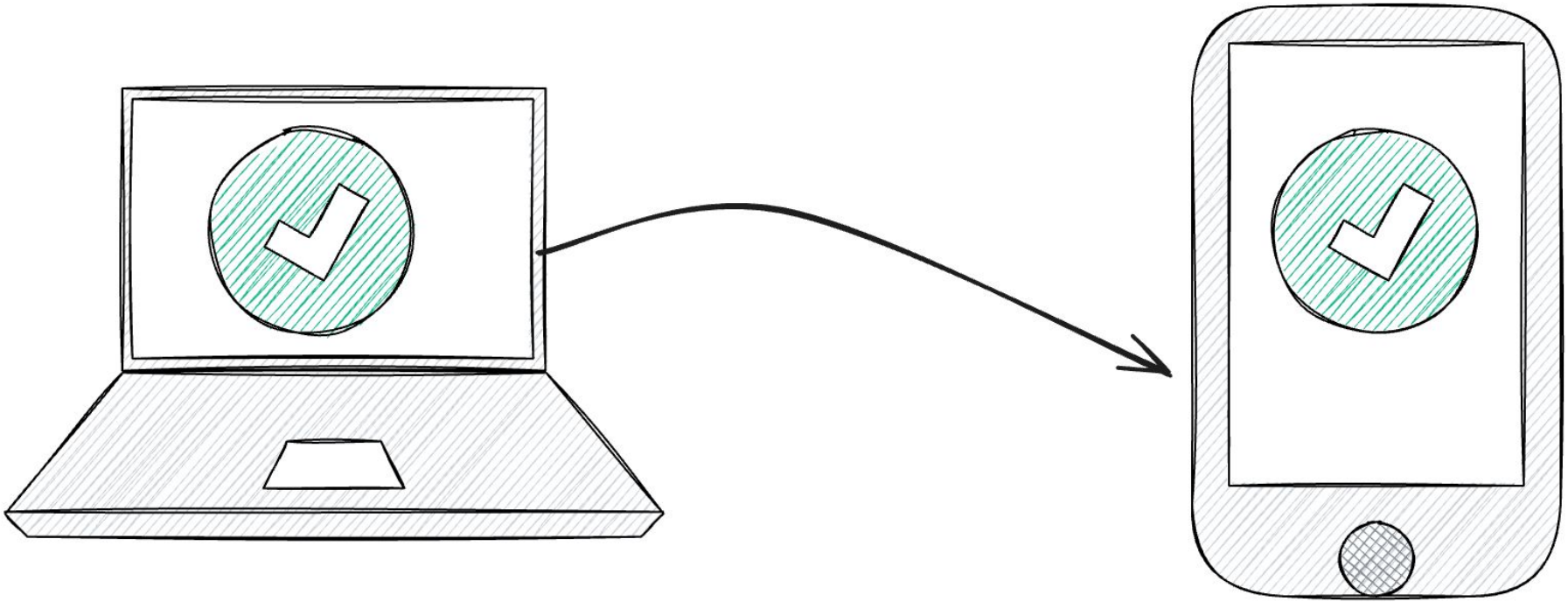


Не костыль, а костылище



циан

Запускаем - работает



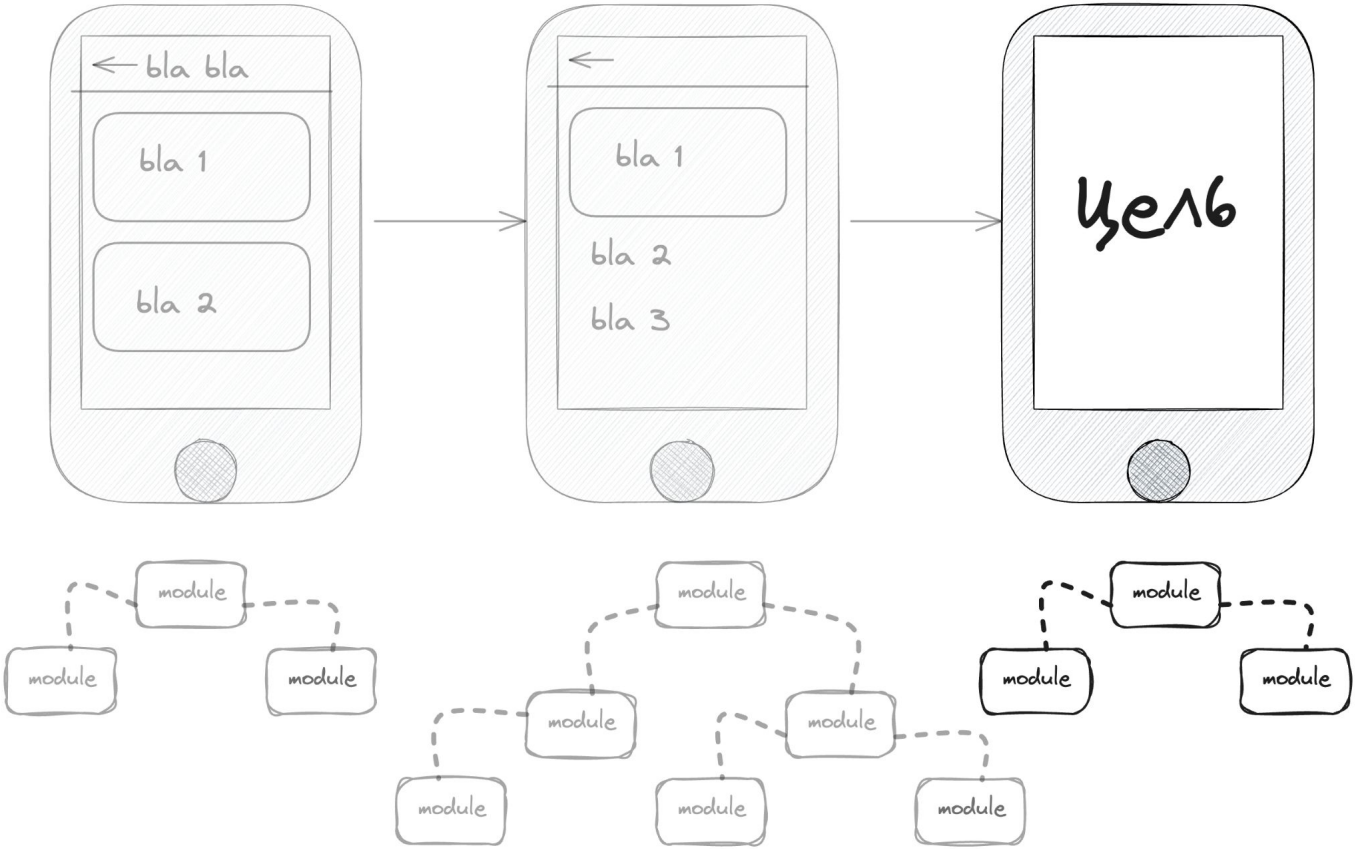
циан

Напоминаю наши проблемы

1. ~~Сложно и опасно руками копаться в списках модулей~~
2. ~~Модули нужно отключать с учётом иерархии~~
3. Нужно попадать на экран независимо от выбранных модулей

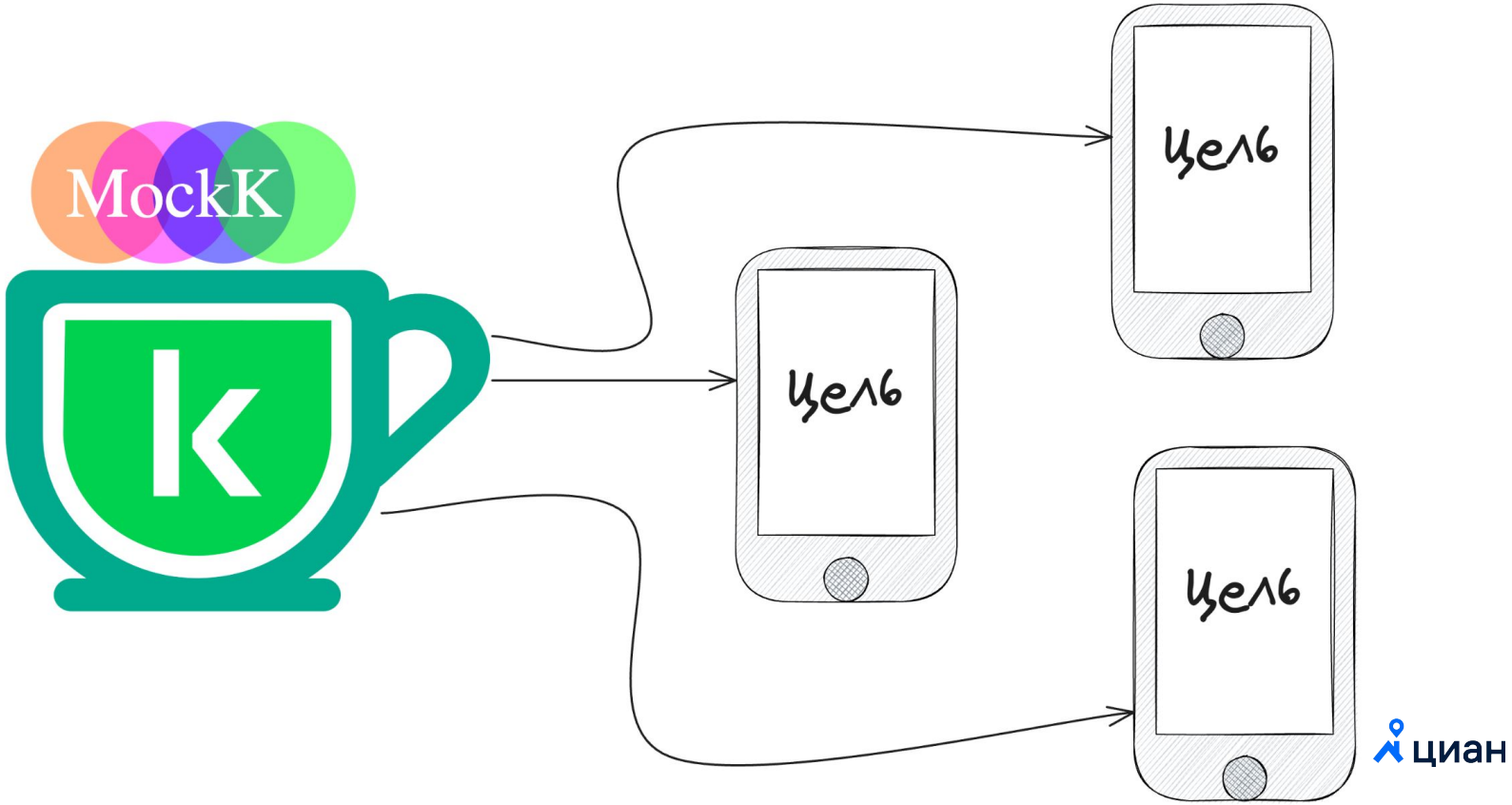
Проблема 3. UI

Предыдущие экраны не нужны



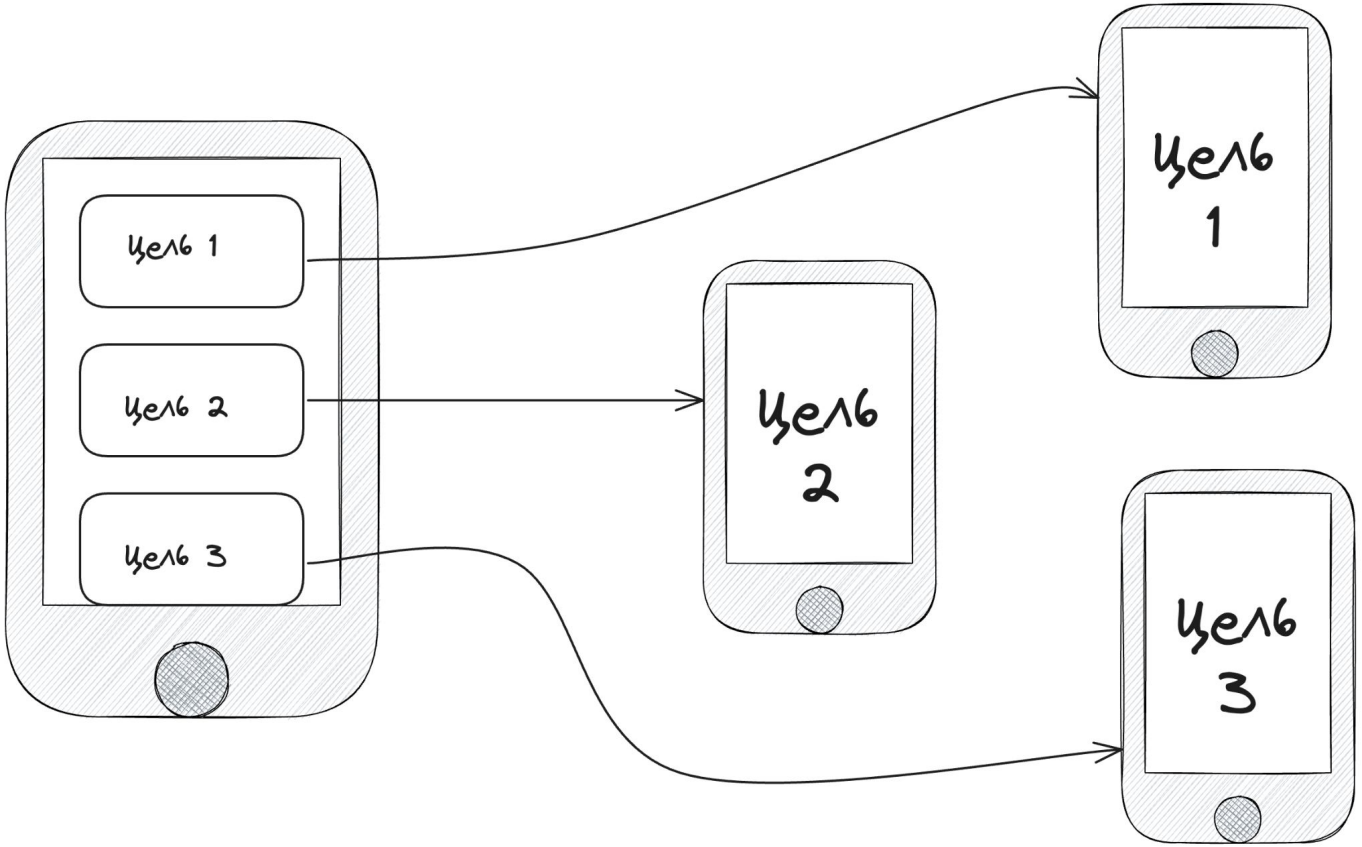
циан

Какие есть решения?



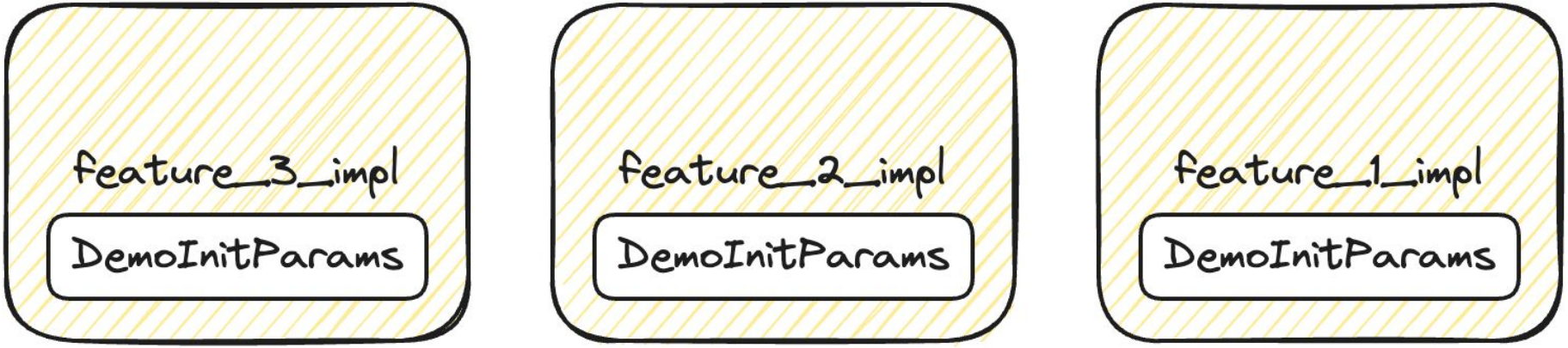
циан

Отдельный разводящий экран



циан

Отдельная сущность



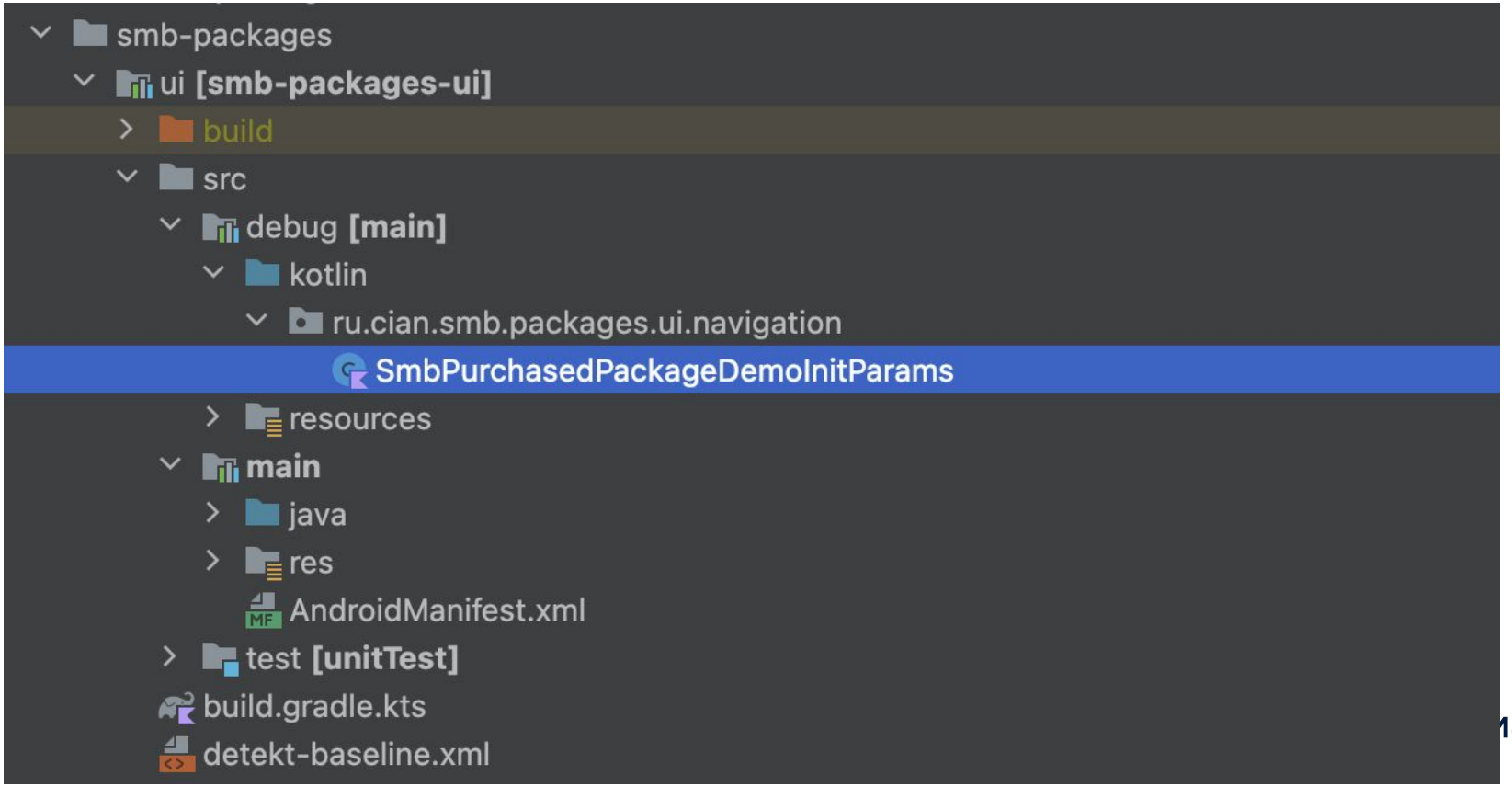
Почему в модуле?

1. Есть проверка при компиляции модуля, что упрощает поддержку
2. Находятся в зоне ответственности команды модуля
3. Есть доступ к internal коду модуля, что упрощает моки
4. Сохраняет плагиновидность

Отдельная сущность. DemoInitParams


```
interface DemoInitParams {
    val title: String
    val description: String
    val initParams: InitParams
    val team: DemoTeam
    /**
     * Выполнится перед открытием экрана, подходит для мокирования
     */
    fun onBeforeOpen() {}
}
```

Расположение



Пример

```
data class SmbPurchasedPackageDemoInitParams(
    override val team: DemoTeam = DemoTeam.MONEY,
    override val title: String = "Мои пакеты",
    override val description: String = "Пакет для риелтора. Долгосрочная аренда коммерческой",
    override val initParams: InitParams = SmbPurchasedPackageInitParams(<id>)
) : DemoInitParams {

    override fun onBeforeOpen() {
        val forMock = dependencyProvider.provide<MoneyDependencies>().provideBalanceInteractor()
        mockk { forMock.getBalance() } returns 132
    }
}
```

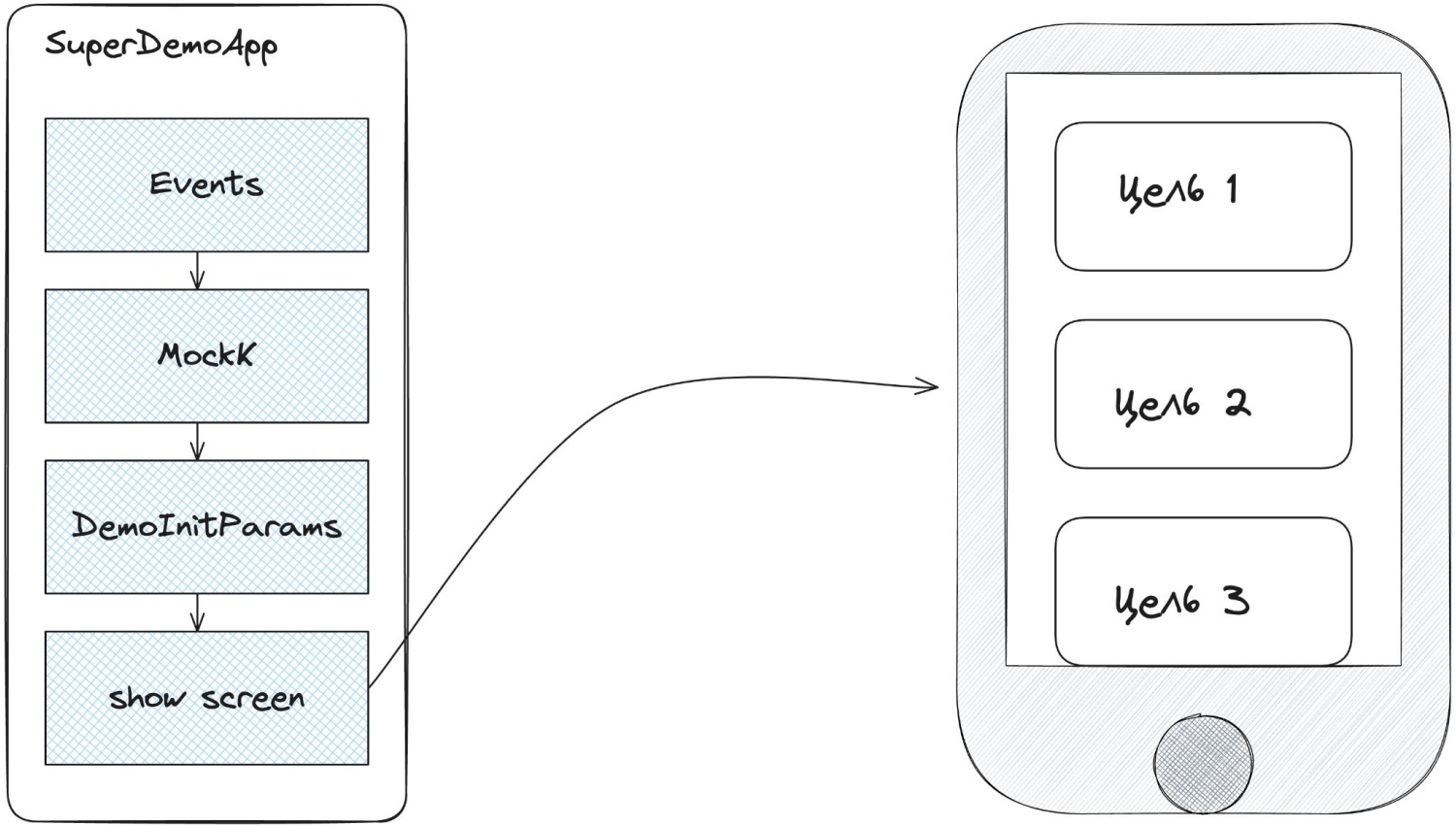
Собираем их через ServiceLoader


```
val initParams: List<DemoInitParams> = DemoInitParams::class.java.let {
    ServiceLoader.load(it, it.classLoader).toList()
}
```

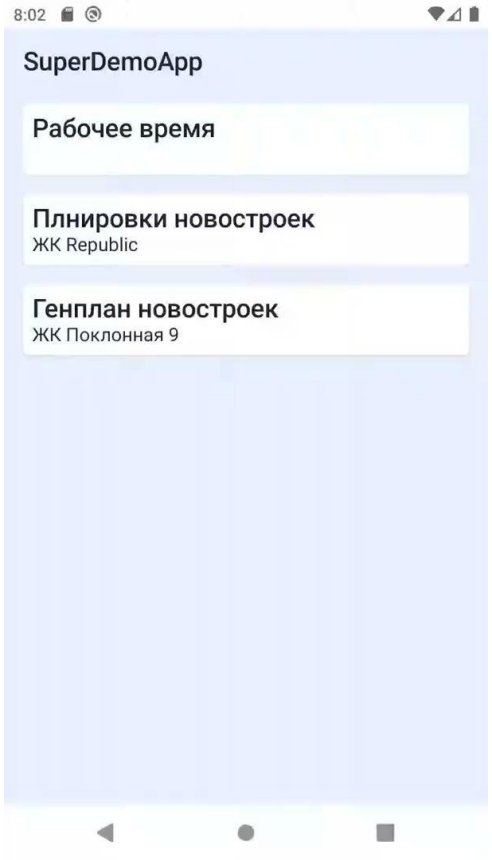
А как он понимает, где искать?


```
// modules / feature / feature1 / impl / src / debug / resources / META-INF.services
// ru.cian.router.init.params.DemoInitParams
ru.cian.smb.packages.ui.navigation.SmbPurchasedPackageDemoInitParams
```

Делаем приложение



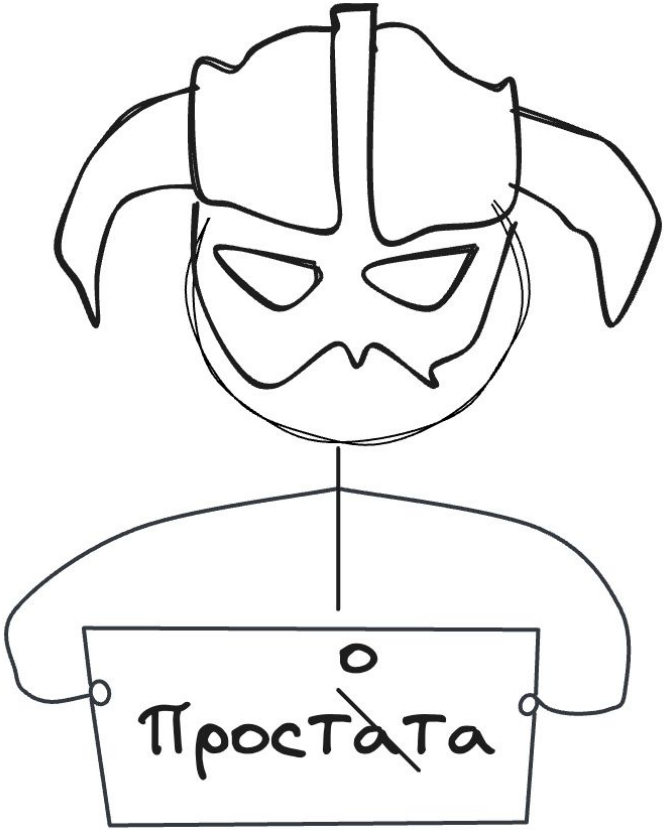
Запускаем - работает



циан

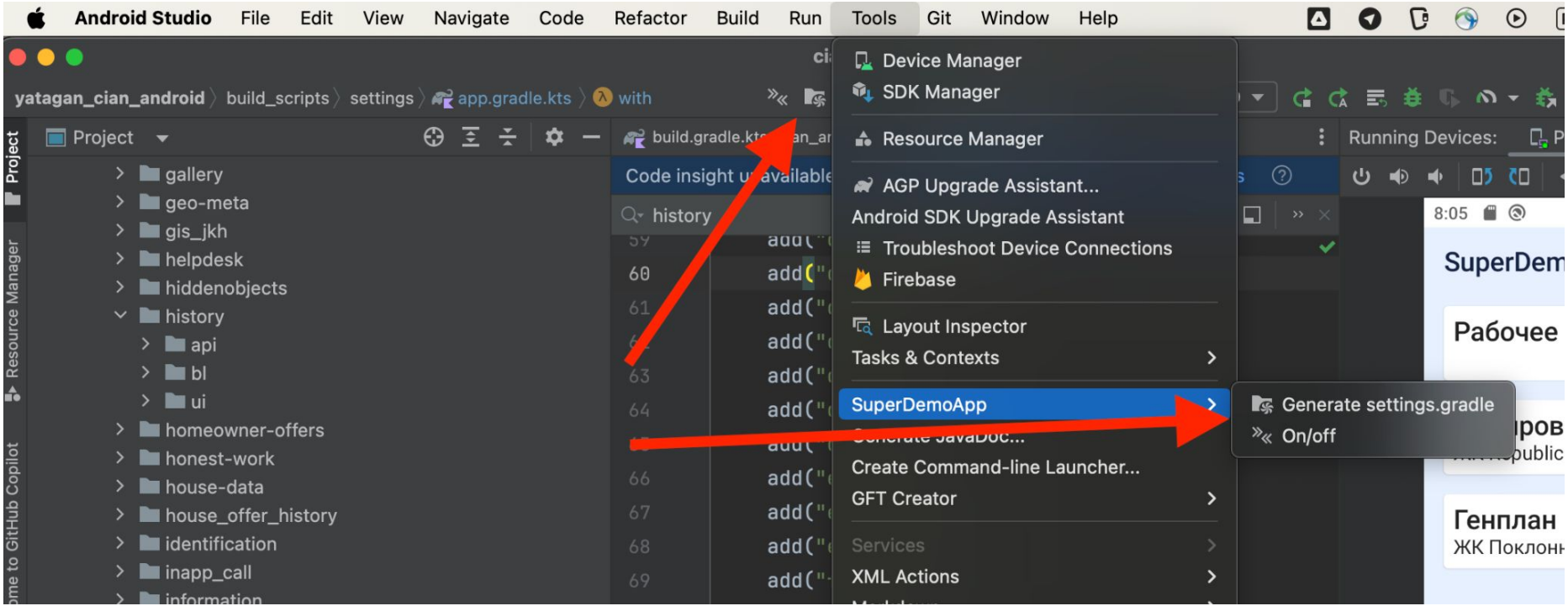
Плагин

Для чего?



циан

Две кнопки

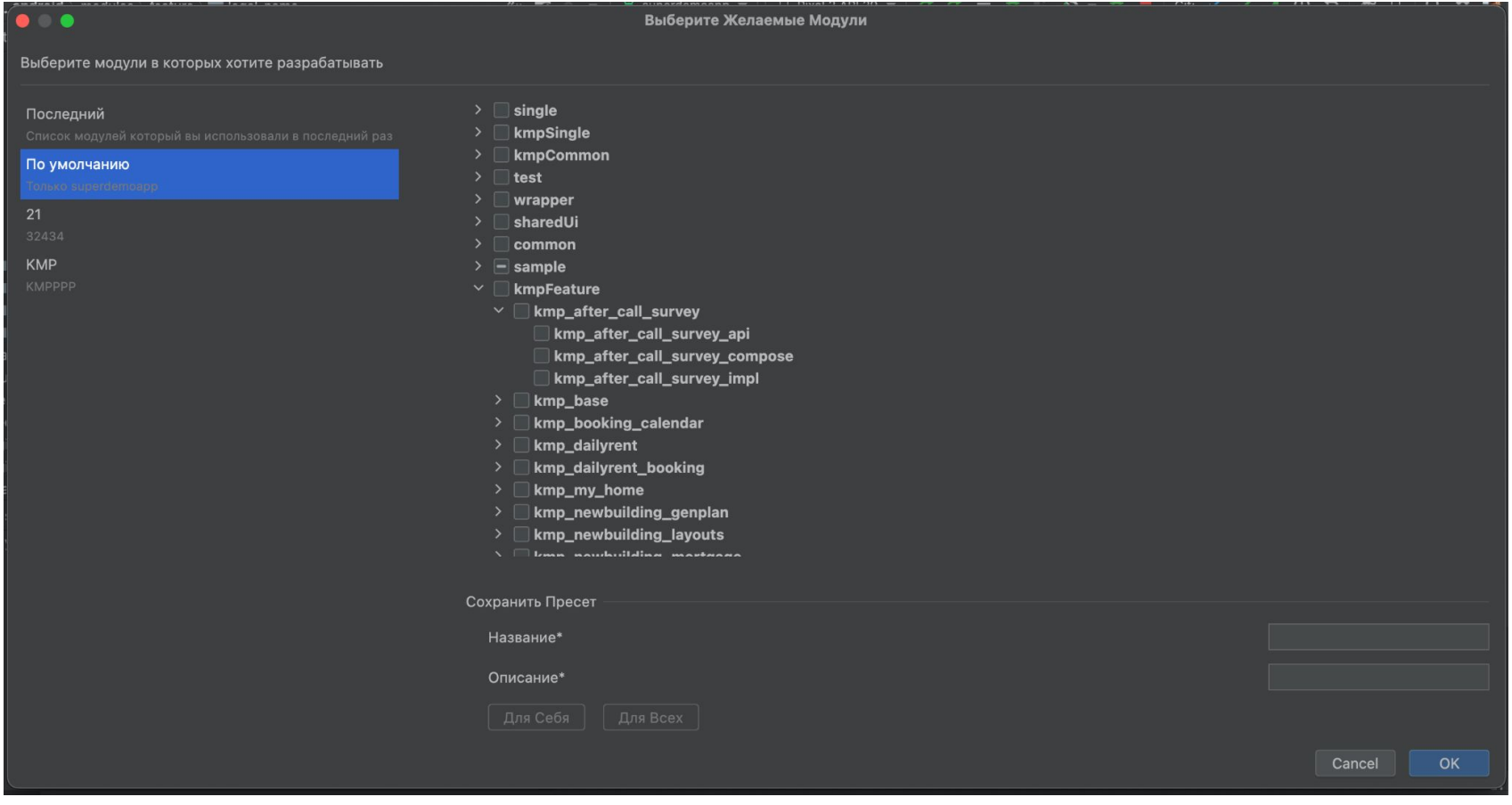


циан

Включение - выключение

1. Включаем - выключаем настройку в gradle.properties

Выбор модулей и пресетов

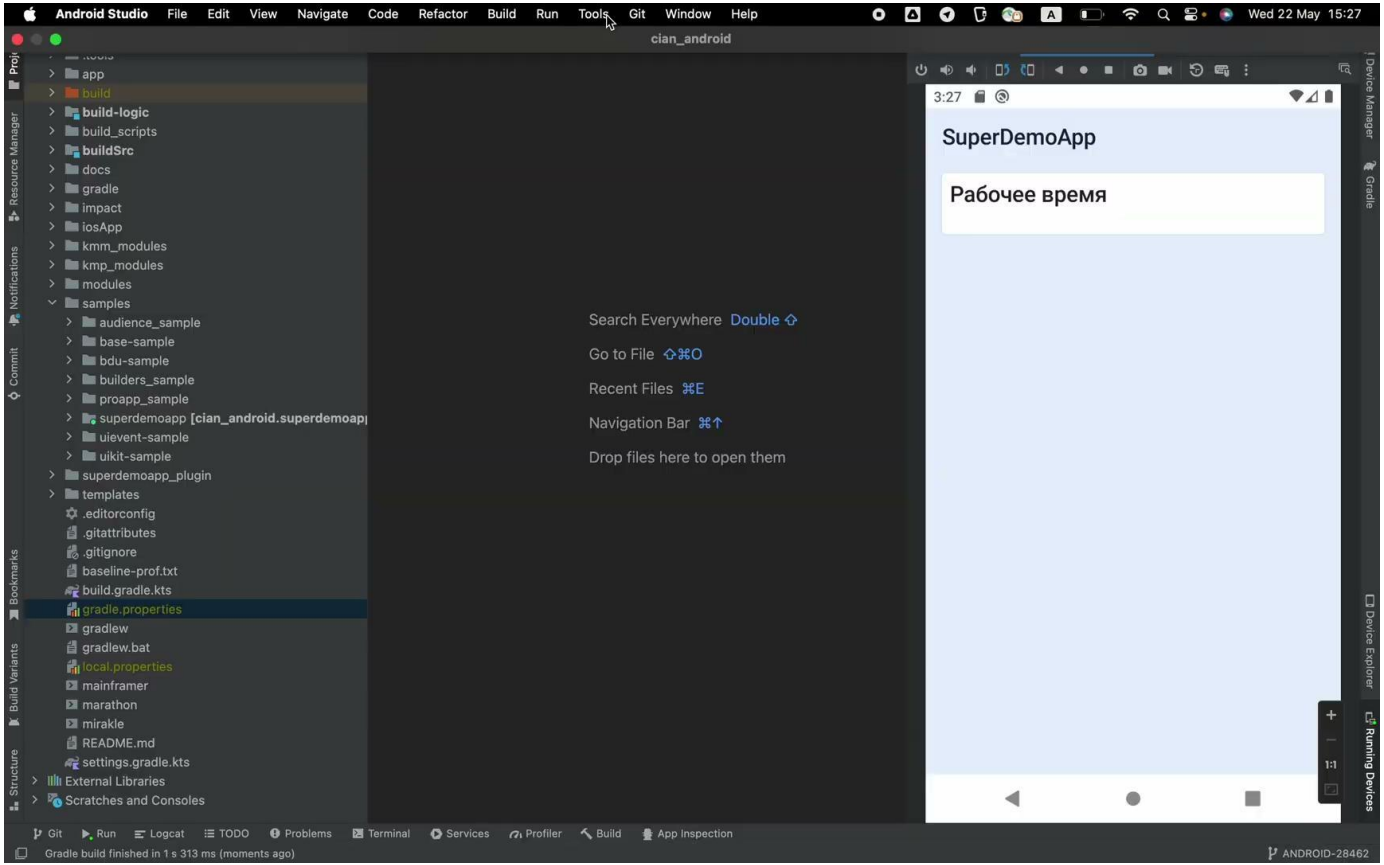


Под капотом


```
./gradlew
-Psettings.file=/build_scripts/settings/empty.gradle.kts
-Pkts.plugins=super_demo_app_graph
-Pp_target_modules=$selectedModules
```

Демонстрация

Видос


Android Studio File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Wed 22 May 15:27
cian_android
app
build
build-logic
build_scripts
buildSrc
docs
gradle
impact
iosApp
kmm_modules
kmp_modules
modules
samples
audience_sample
base-sample
bdu-sample
builders_sample
proapp_sample
superdemoapp [cian_android.superdemoapp
uievent-sample
uikit-sample
superdemoapp_plugin
templates
.editorconfig
.gitattributes
.gitignore
baseline-prof.txt
build.gradle.kts
gradle.properties
gradlew
gradlew.bat
local.properties
mainframer
marathon
mirakle
README.md
settings.gradle.kts
External Libraries
Scratches and Consoles
Search Everywhere Double ⇧
Go to File ⇧⌘O
Recent Files ⌘E
Navigation Bar ⌘↑
Drop files here to open them
3:27
SuperDemoApp
Рабочее время
Git Run Logcat TODO Problems Terminal Services Profiler Build App Inspection
Gradle build finished in 1 s 313 ms (moments ago)
ANDROID-28462

циан

Что потенциально дальше?

1. APK весит много
2. Долгий запуск
3. Авторизация

Вопросы и любовные письма

@princeparadoxes

циан