



DGTL

Семантический Gradle

Публикуем артефакты со спокойствием и уверенностью



Газаров Александр

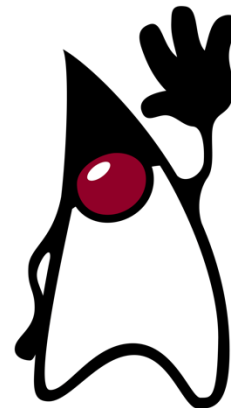
Инженер-эксперт по разработке и сопровождению



Всем привет!
Я — Газаров Александр



Android – с 2010 года



Java ME – с 2006 года



Инженер-эксперт по разработке и сопровождению, вице-президент



Ответственный за приложение Android для малого и среднего бизнеса



Автор плагина семантического версионирования Raiffeisen

О ЧЁМ МЫ ПОГОВОРИМ



Зачем семантическое версионирование, и при чём тут Gradle?

Ищем автоматизированное решение

Пишем задачи для плагина

Соединяем и конфигурируем

Подведение итогов

Зачем семантическое версионирование

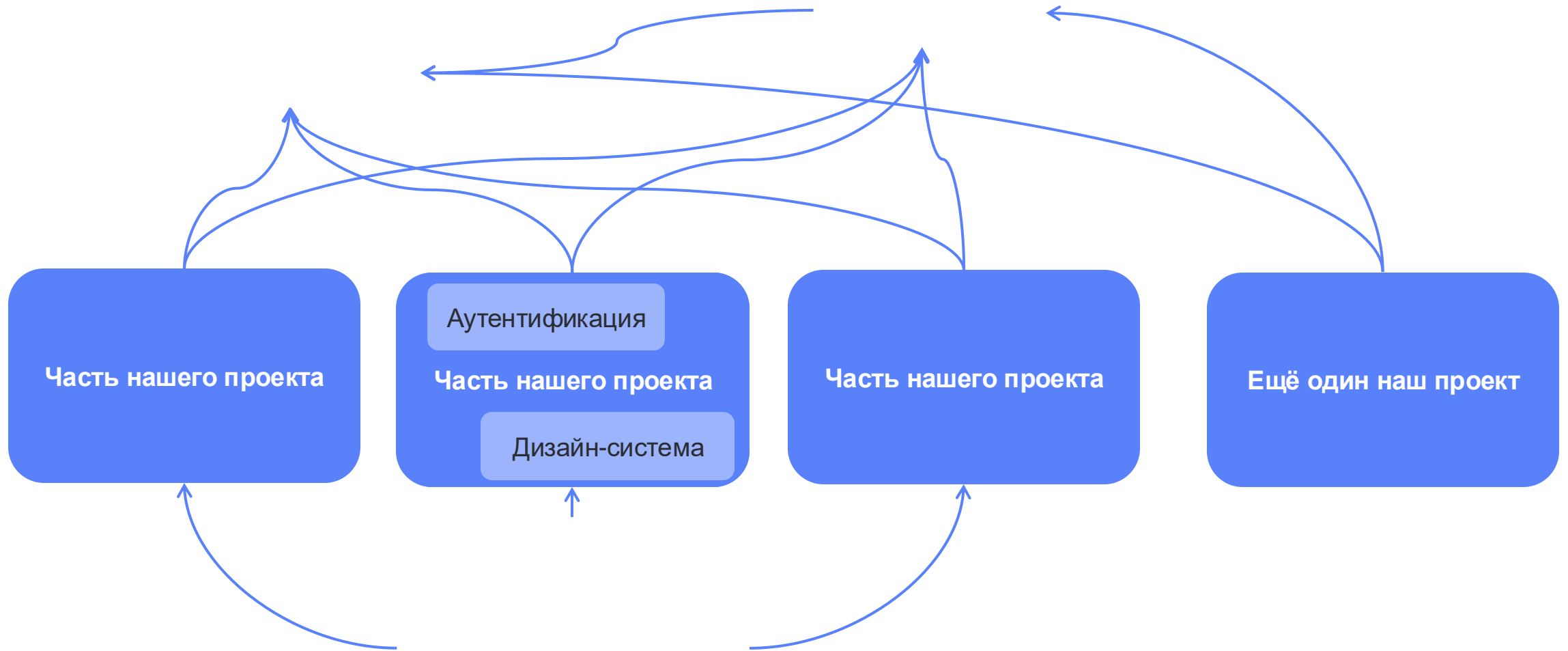
И при чём тут Gradle?

DGTL



01

Что творится, когда проект растёт



Код ломает код



Неприятность в API

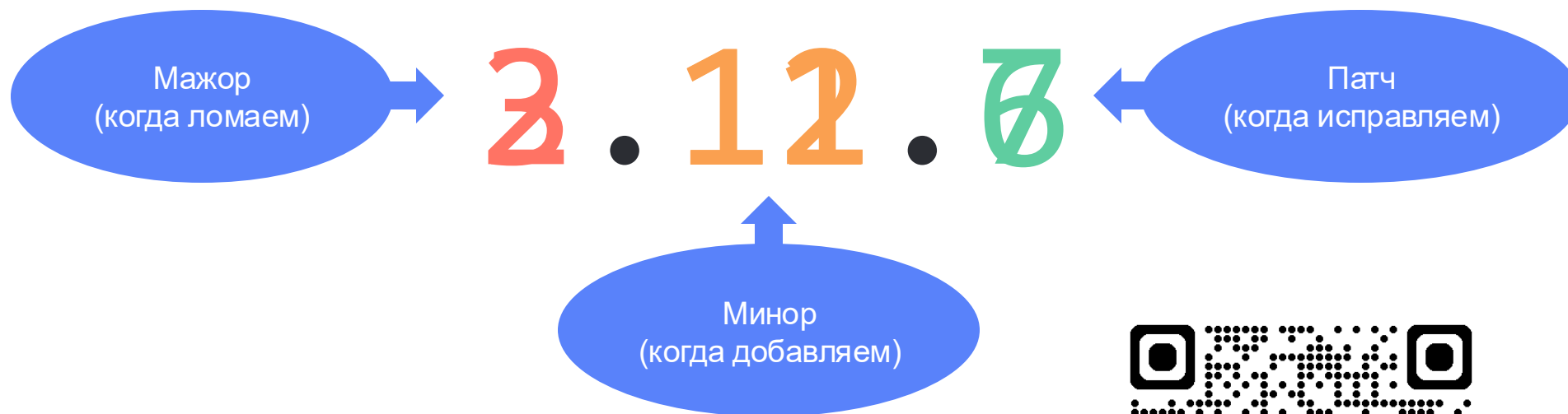
```
fun ourAwesomeFunction(parameter: Int)
fun ourAwesomeFunction(parameter: Int, anotherParameter: String)
```

- Частые изменения – проблемы разработчиков
- Большие проекты – большие проблемы
- Отсутствие поломок – отсутствие развития

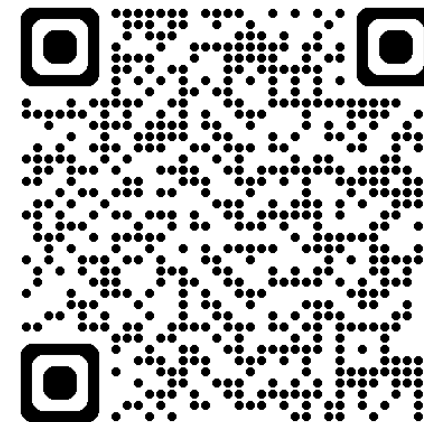
Оптимум

Контролируемые изменения, очевидные для потребителей

Семантическое версионирование



<https://semver.org>



Видно, что когда ломается (но не всегда)



```
fun ourAwesomeFunction(parameter: Int)
fun ourAwesomeFunction(parameter: Int, anotherParameter: String? = null)
```

Код компилируется без изменений! 😊

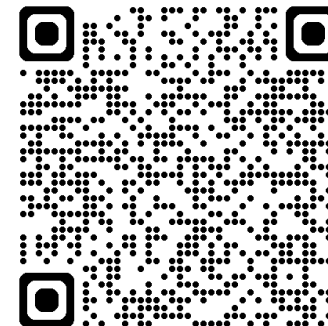
Другая сигнатура. Оно сломалось. ☹️

```
data class OurAwesomeClass(val field: Int)
data class OurAwesomeClass(val field: Int, val anotherField: String? = null) {
    constructor(field: Int) : this(field, null)
}
```

Код работает без изменений! 😊

Другая сигнатура у сору(). Оно сломалось. ☹️

<https://kotlinlang.org/docs/api-guidelines-backward-compatibility.html>



Проект в нескольких репозиториях



- Мультимодульность означает мультиартефактность
- Нужен BoM (Bill of Materials), чтобы выровнять версии
- Каждый модуль требует свою версию, и BoM в том числе

Библиотечный репозиторий без приложения

BoM

Библиотека

Библиотека

Библиотека

Библиотека

Библиотека

Библиотека

Ищем автоматизированное решение

DGTL



И как написать своё, когда не нашли чужое

Что уже есть и почему не годится



Что?	Что может?	С чем не справляется?
Binary compatibility validator (плагин от JetBrains)	<ul style="list-style-type: none">• Сбрасывает публичный API в текстовый файл, который добавляется в git• Говорит, есть ли изменения по сравнению с тем, что было	<ul style="list-style-type: none">• Ничего не знает про версии• Не различает характер изменений• Не понимает специфику Android

<https://github.com/Kotlin/binary-compatibility-validator/>



Принимаемся за свой плагин



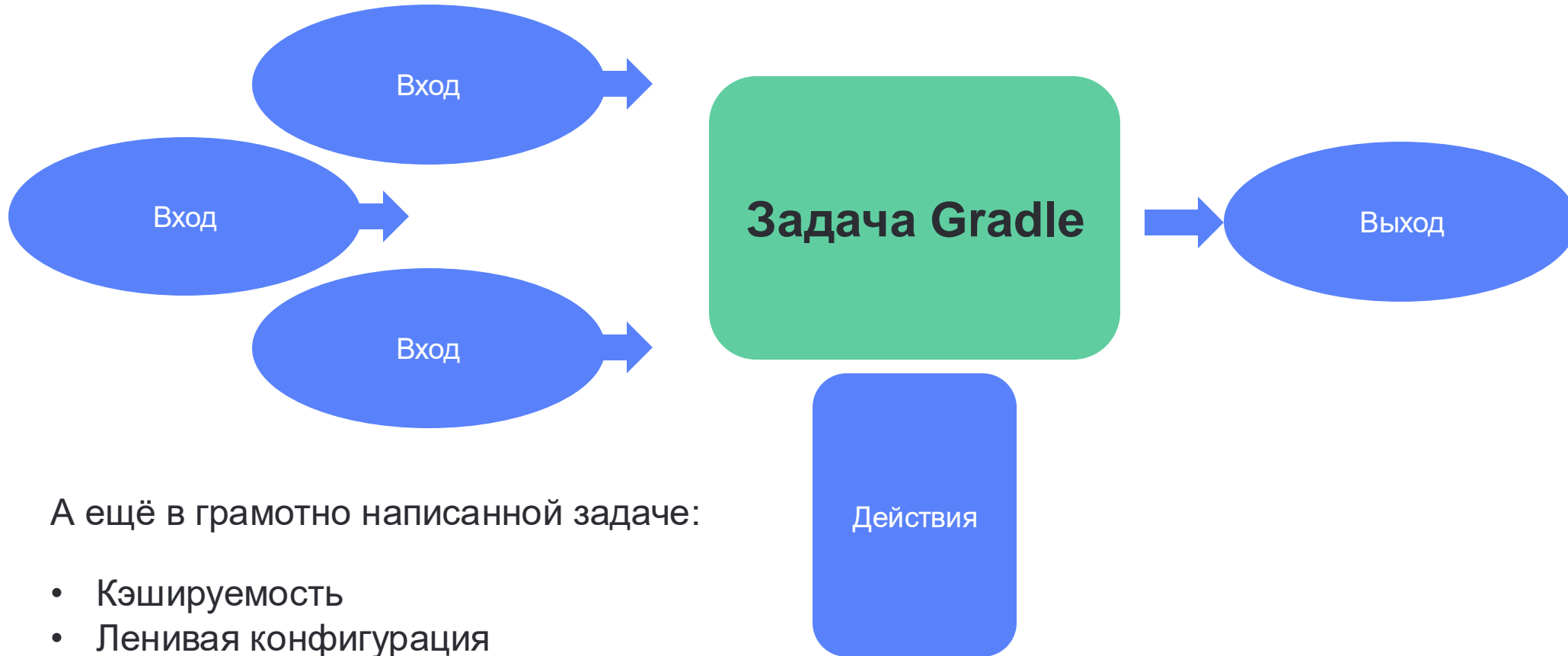
- Gradle: граф задач, сначала составляется, потом исполняется
- Каждый плагин получает в своё распоряжение проект и что-то может добавить своё

```
class RbmPlugin : Plugin<Project> {  
    override fun apply(target: Project) {  
        // Подцеплять тут  
    }  
}
```

Каждый наш
модуль –
проект Gradle

- Цепляемся к проекту с настроенными плагинами и дописываем свои задачи
- Начинаем с существующего плагина конвенций

Пишем задачи грамотно



А ещё в грамотно написанной задаче:

- Кэшируемость
- Ленивая конфигурация

Пишем задачи для плагина

DGTL



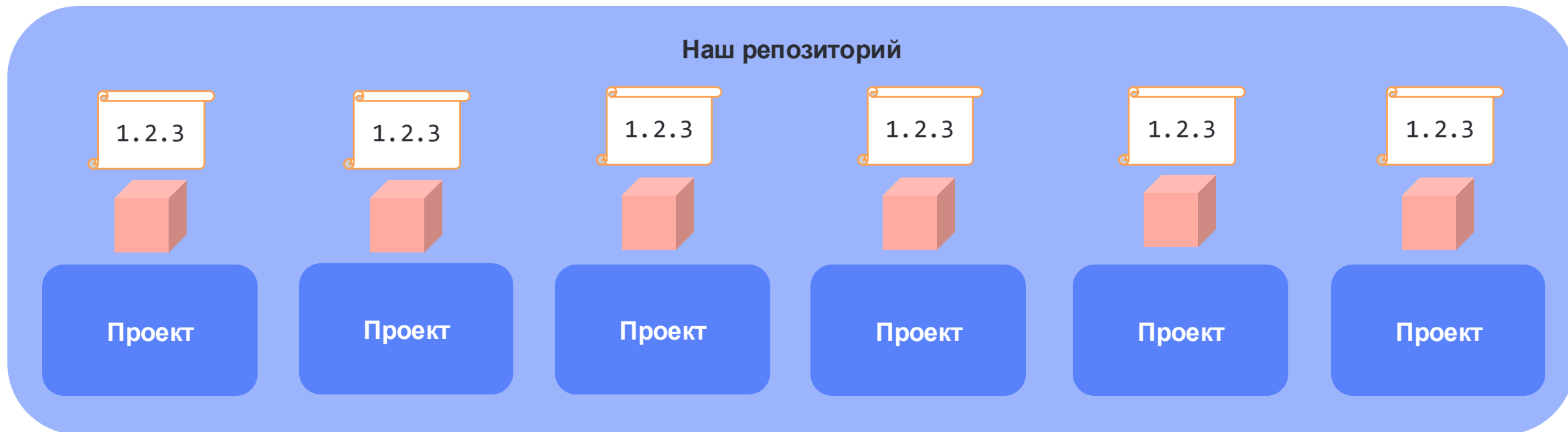
Разработаем тактику и будем её
придерживаться

03

Разрабатываем план



- Собираем каждый модуль и ~~записываем изменения~~ ~~в файл~~ ~~вообще~~
- ~~Дерево сборки для VoM~~ ~~Данные~~ ~~версиями~~ (кэш поможет)
- ~~Действие~~ ~~публикуем~~ ~~для~~ ~~VoM~~
- Всё публикуем



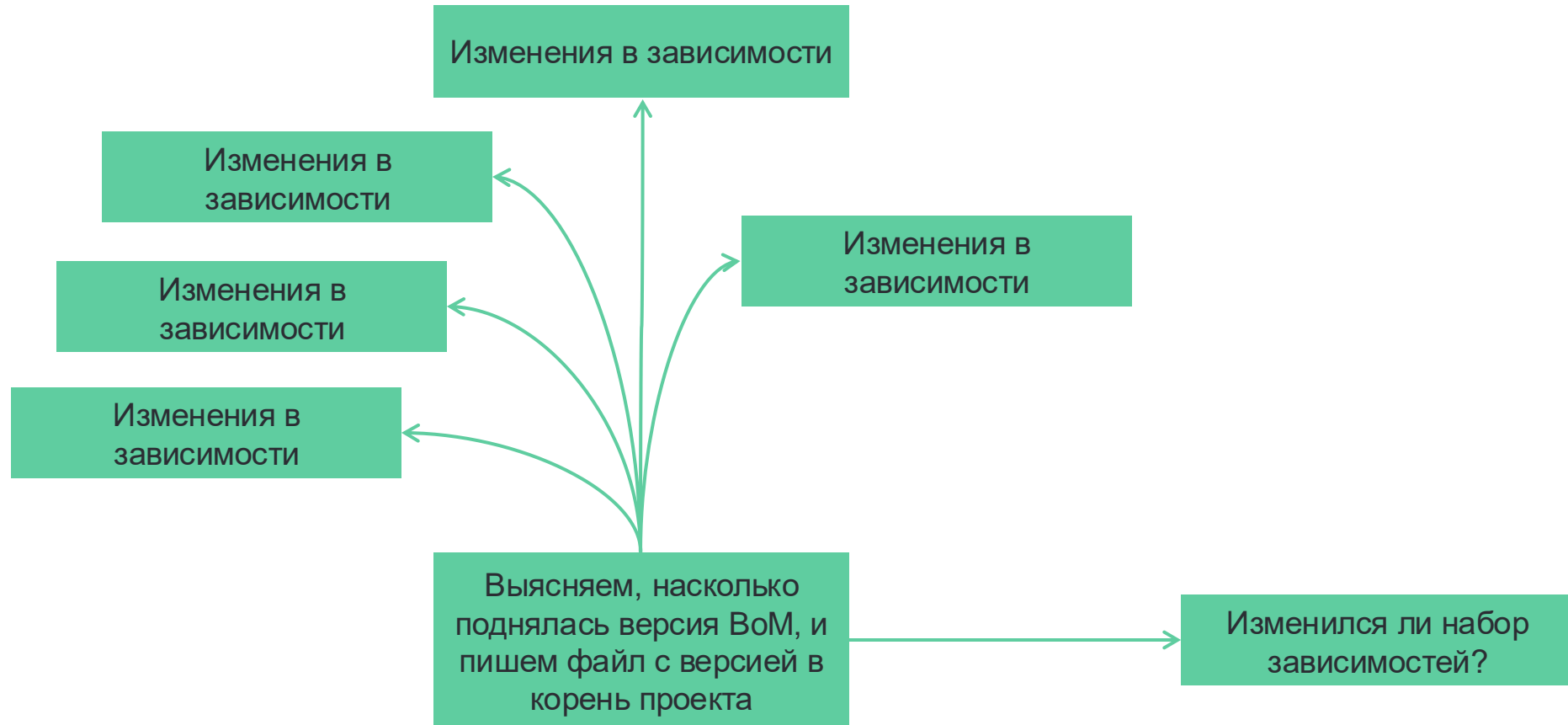
И ещё добавляем настройку, чтобы не публиковаться с мажорными изменениями

Рисуем будущий граф для проекта



Изменения в зависимостях не считаем отдельно, достаточно изменений нашего артефакта

План для VoM



Как выглядят наши задачи



```
internal abstract class BreakingChangesReportTask : DefaultTask() {
    @get:InputFiles abstract val configFiles: ListProperty<RegularFile> // Получаем вход лениво
    @get:OutputFile abstract val projectsWithBreakingChangesReportFile: RegularFileProperty // И выход тоже

    init {
        group = HelpTasksPlugin.HELP_GROUP
        description = "Reports if semantic versioning has found any breaking changes."
    }

    @TaskAction
    fun report() {
        // Делаем своё дело
    }
}
```

Выясняем, какая версия последняя



На вход

```
@get:Input abstract val groupId: Property<String>  
@get:Input abstract val artifactId: Property<String>
```

На выход

```
@get:OutputFile abstract val releasedVersionFile: RegularFileProperty
```

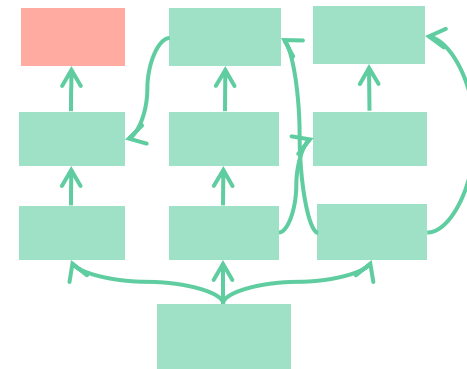
Для похода в сеть

```
@get:ServiceReference("artifactory")  
abstract val artifactoryClientService: Property<HttpClientService>
```

При конфигурации: регистрация сервиса

```
gradle.sharedServices.registerIfAbsent("artifactory", HttpClientService::class.java) {  
    // Тут конфигурируем  
}
```

Нужно класть в файл, обычные значения в качестве выходов Gradle не воспринимает



Пишем поход в Artifactory



Действия

@TaskAction

```
fun getVersion() = runBlocking {  
    // Идём на Artifactory и спрашиваем про наш артефакт  
    releasedVersionFile.asFile.get().writeText(latestVersion)  
}
```

Внутреннее свойство для удобства

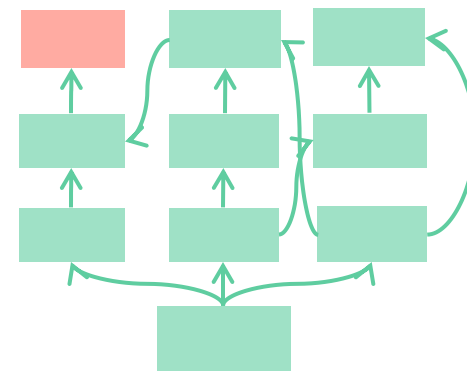
```
@Suppress("UnstableApiUsage")  
@get:Internal  
val releasedVersion: Provider<Version>  
    get() = project  
        .providers  
        .fileContents(releasedVersionFile)  
        .asText  
        .filter { it.isNotEmpty() }  
        .map(Version::parse)
```

Обходим
невозможность
работать с null
в map

<https://github.com/gradle/gradle/issues/12388>

Пример:

`task.releasedVersion`



Идём за удалённым артефактом

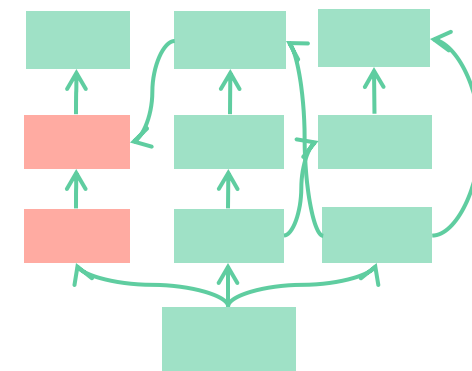
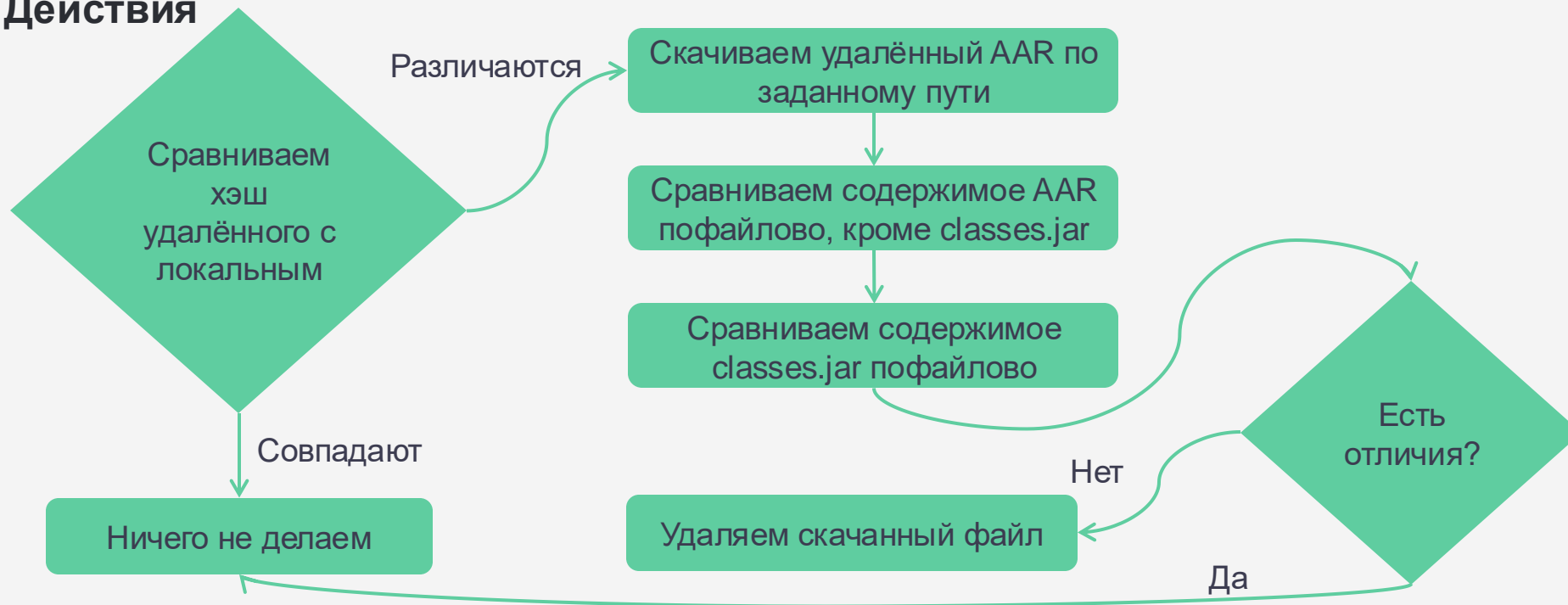


На вход: координаты удалённого артефакта и ещё локальный

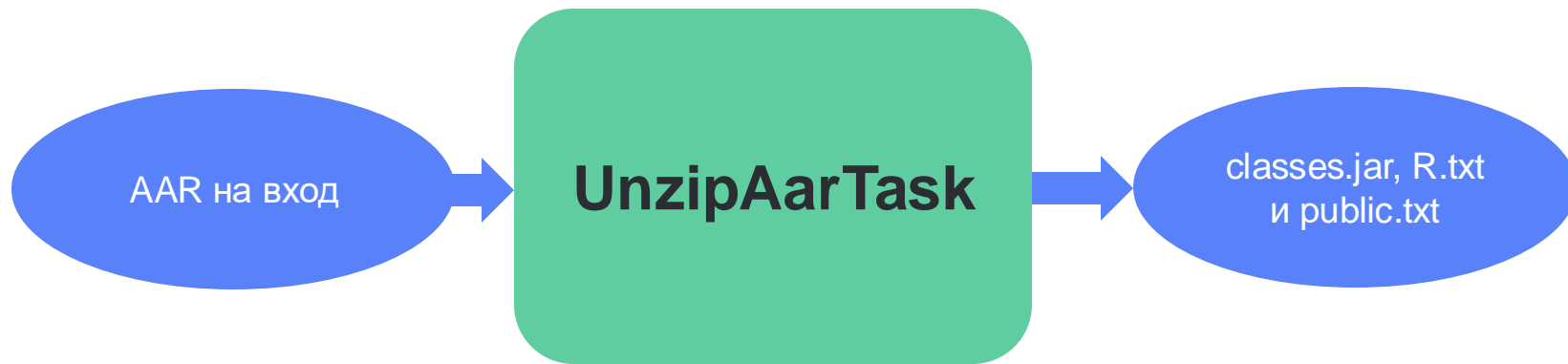
```
@get:InputFile  
abstract val localArtifactFile: RegularFileProperty
```

На выход: удалённый артефакт (если есть отличия от локального)

Действия

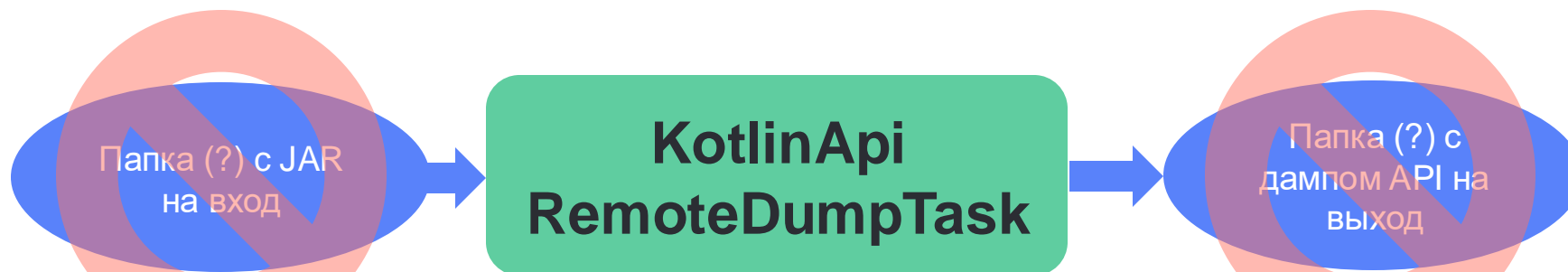


Распаковываем и сравниваем API



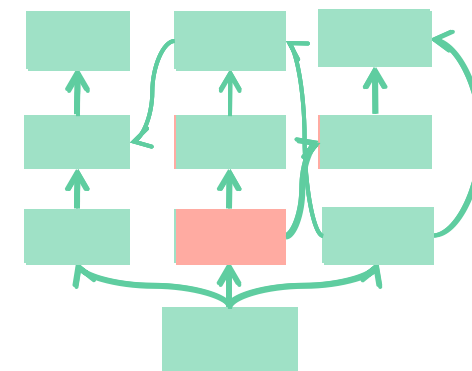
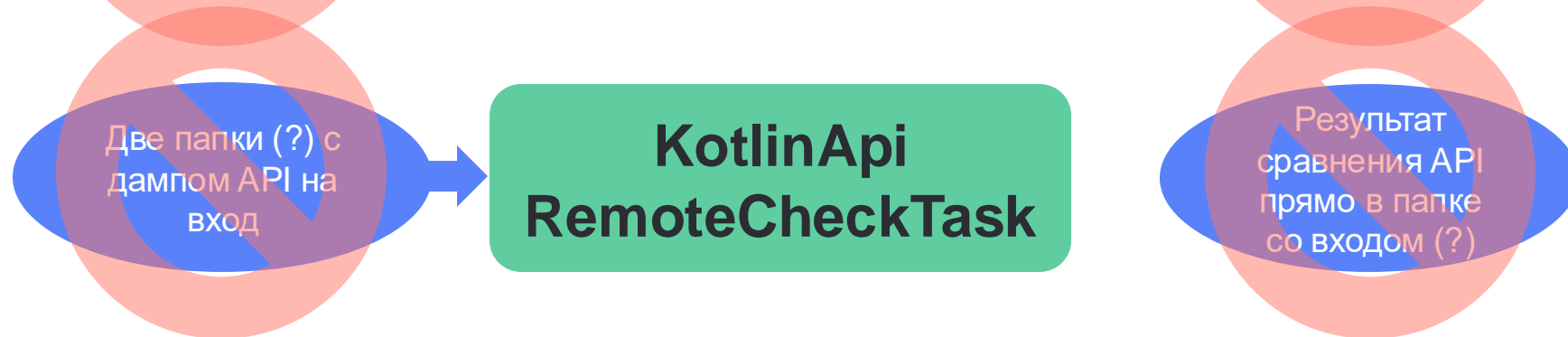
x2:

для удалённого и для локального



x2:

для удалённого и для локального



Рассчитываем новую версию



На вход: результаты из предыдущих задач (сравнения API, изменения в артефакте и т.д.)

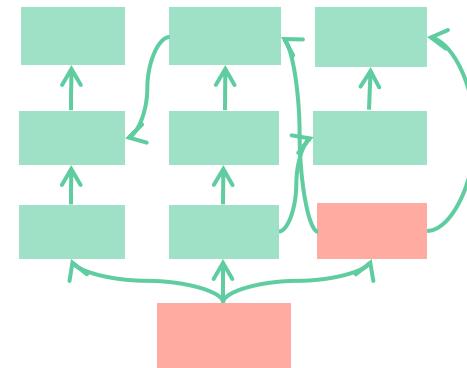
Для потенциально отсутствующих

```
@get:Input  
@get:Optional // Not set if the artifact hasn't been published  
abstract val releasedVersion: Property<Version>
```

Специальный вариант для файла, @InputFiles

```
@get:InputFiles // The file can be missing if the artifact hasn't been published  
abstract val remoteArtifactRFile: RegularFileProperty
```

На выход: JSON с информацией о новой и старой версии проекта



Рассчитываем новую версию



Действия

Рассчитываем разницу в ресурсах по R.txt и public.txt

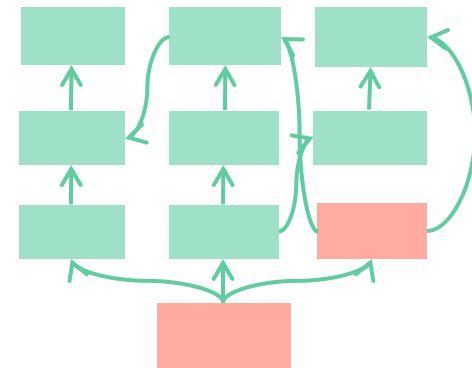
Берём максимальное изменение из всех источников (код, ресурсы, артефакт)

Записываем результат в JSON

Действия для ВоМ

Скачиваем удалённый ВоМ и находим добавления и удаления

Берём максимальное изменение из всех остальных проектов плюс изменения из предыдущего шага



Соединяем и конфигурируем

DGTL



Пазл вот-вот сложится

04

Соединяем вместе



```
private fun Project.registerSemanticVersionTasksForLibrary(publication: MavenPublication) {  
    val releaseArtifact = publication.artifacts.find { it.classifier == "release" } ?: return  
    // Разные задачи...  
    val downloadTask = tasks.register<DownloadReleasedArtifactIfDifferentTask>("downloadReleasedArtifact") {  
        groupId.set(getReleasedVersionTask.flatMap { it.groupId })  
        artifactId.set(getReleasedVersionTask.flatMap { it.artifactId })  
        version.set(getReleasedVersionTask.flatMap { it.releasedVersion })  
        localArtifactFile.set(releaseArtifact.file)  
        outputFile.set(outputDirectoryProvider.map { it.file("remote.aar") })  
  
        onlyIf { getReleasedVersionTask.get().releasedVersion.isPresent }  
        dependsOn(releaseArtifact.buildDependencies, getReleasedVersionTask)  
    }  
    // Разные задачи...  
}
```

Ленивая
конфигурация
в действии

onlyIf выполняется
непосредственно
перед самой
задачей, а не
заранее

Отчитываемся о ломающих изменениях



```
if (target.rootProject == target) {
    target.tasks.register<BreakingChangesReportTask>("breakingChanges") {
        projectsWithBreakingChangesReportFile.set(target.layout.buildDirectory.file("reports/nice-report-file.txt"))
    }

    for (subproject in target.subprojects) {
        subproject.tasks.withType<WriteSemanticVersionTask>().all {
            this@register.configFiles.add(outputVersionFile)
        }
    }
}
```

all как аналог слушателя для текущих и будущих элементов

Отключаем публикацию ломающих изменений



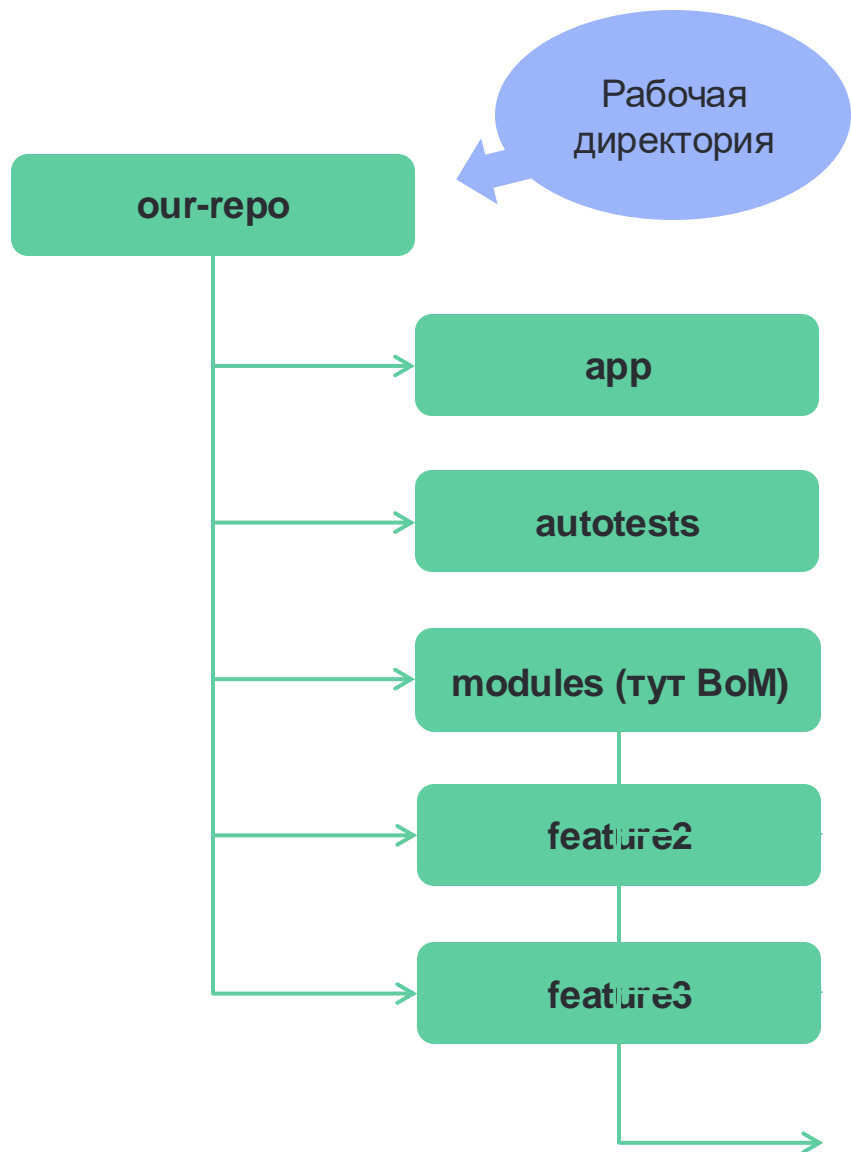
Отложенная
конфигурация
с afterEvaluate

Настройка
сборки через
gradleProperty

```
afterEvaluate {  
    val skipPublishing = providers.gradleProperty(PROPERTY_SKIP_PUBLISHING).orNull?.toBoolean() == true  
    if (skipPublishing) {  
        val reportTask = rootProject.tasks.withType<BreakingChangesReportTask>().singleOrNull()  
            ?: error("The root project doesn't contain the breaking changes report task, is the plugin applied?")  
  
        tasks.withType<AbstractPublishToMaven> {  
            onlyIf("breaking changes are present") {  
                reportTask.projectsWithBreakingChanges.get().isEmpty()  
            }  
            dependsOn(reportTask)  
        }  
    }  
    // Конфигурируем остальное  
}
```

В консоли: `./gradlew publish -PskipPublishingOnBreakingChanges=true`

Исключаем проекты из сравнения селектором



```
cd modules  
../gradlew writeIncrementedSemanticVersion
```

Включаем на CI



Задача для публикации без ломающих изменений

script:

- cd modules
- gradle writeIncrementedSemanticVersion
- cd ..
- gradle publish -PskipPublishingOnBreakingChanges=true
- echo "BOM_INCREMENT=\$(jq -r .increment modules/modules.ver)" > increment.env

Задача для публикации с ломающими изменениями

script:

- cd modules
- gradle writeIncrementedSemanticVersion
- cd ..
- gradle publish

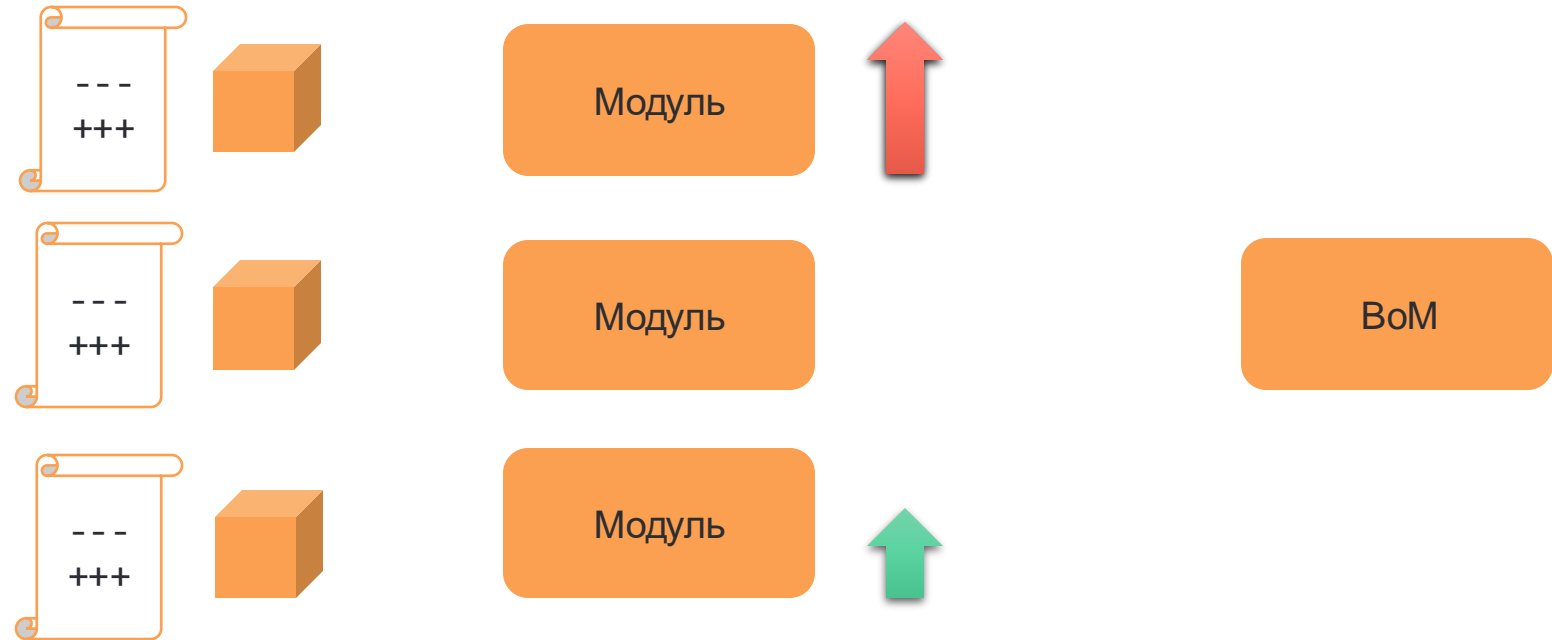
Подведение итогов

DGTL



Что там у нас в результате вышло?

Как в итоге работает плагин



Видение на будущее



- Отказ от средства JetBrains и создание не основанного на анализе текста инструмента
- Проверка совместимости версий библиотек





DGTL

ГОТОВ ОТВЕТИТЬ НА ВАШИ ВОПРОСЫ

Газаров Александр

Инженер-эксперт по разработке и сопровождению,
Райффайзен Банк

Telegram: @DrMetallius

