Bazel для Android

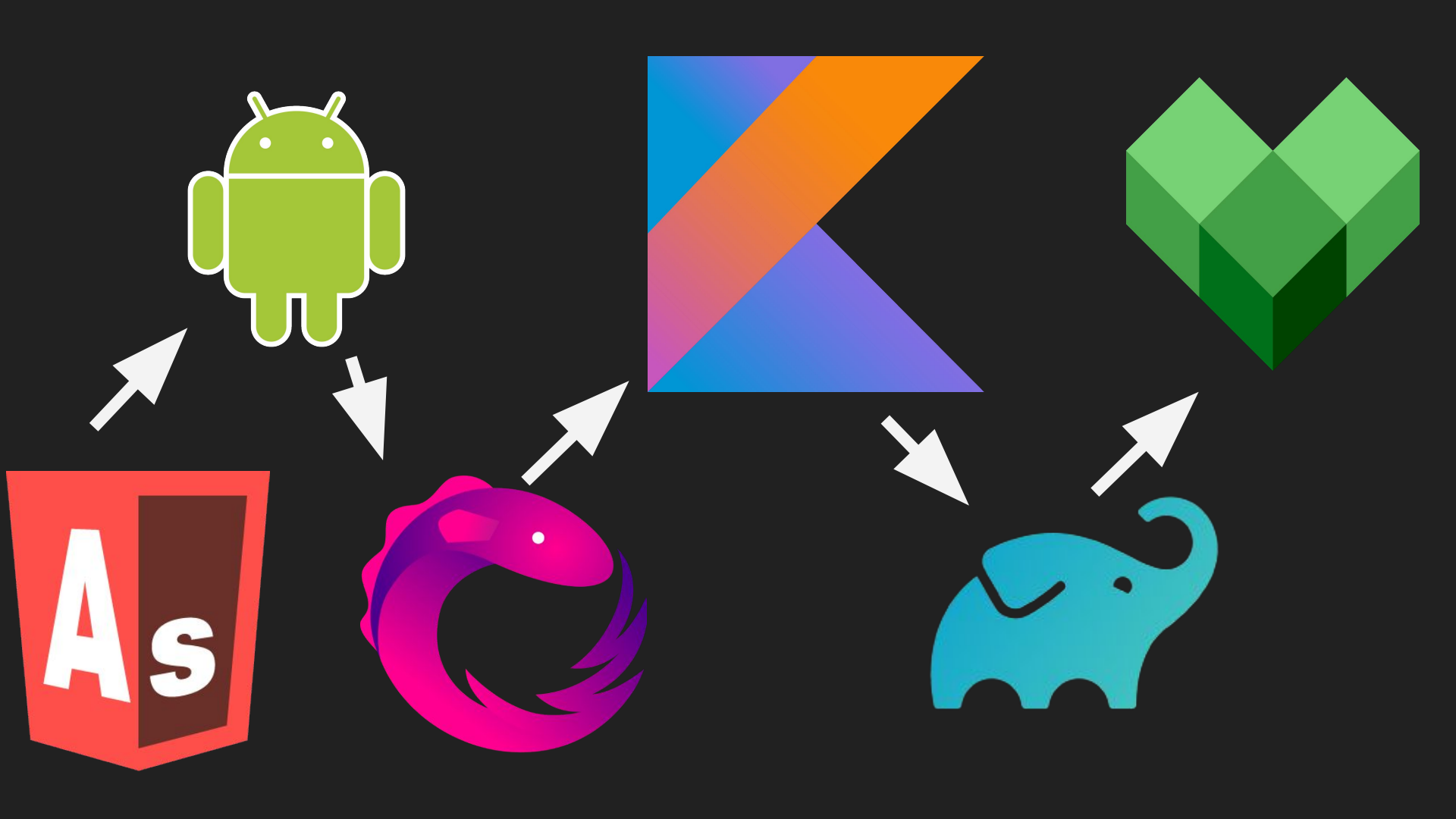
это не страшно

Обо мне

Staff Software Engineer @ Lyft, Level 5
Ушел из мобильной разработки
Bazel фанбой

Ex Kotlin фанбой

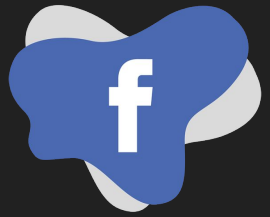
~10 лет в мобильной разработке
~3 года в билд инфраструктуре



Пара слов о Gradle

- Плохо работает для больших Android проектов
- Сложно поддерживать билд конфигурацию
- Сложно писать плагины(e.g. AGP, Kotlin)
- Шаткие основы

- Почему?
- Легаси



Square

{Fast, Correct} - Choose two

Как нам собирать приложения любого размера быстро и корректно?

Bazel/Blaze

Bazel был создан чтоб собирать приложения в

Google монорепозитории*

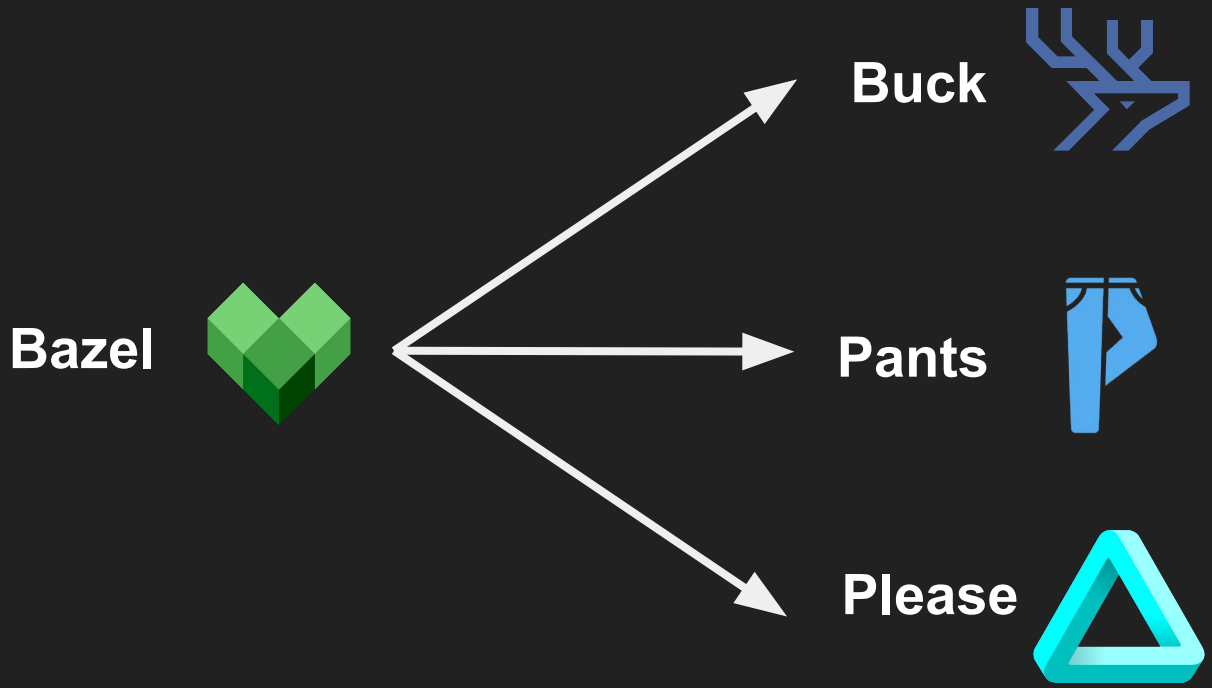
~16 лет назад

Google монорепозиторий (данные за 2016 год)

- >2млрд строк кода, размером 86TB+
- >1млрд файлов
- >35млн коммитов

https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext

Альтернативы



Почему доклад про **Bazel**?

Что такое Bazel?

Как это работает?

Изоляция

- Сандбоксинг
- Конфигурация окружения

Rules!*

Правила

Что это за правила такие?

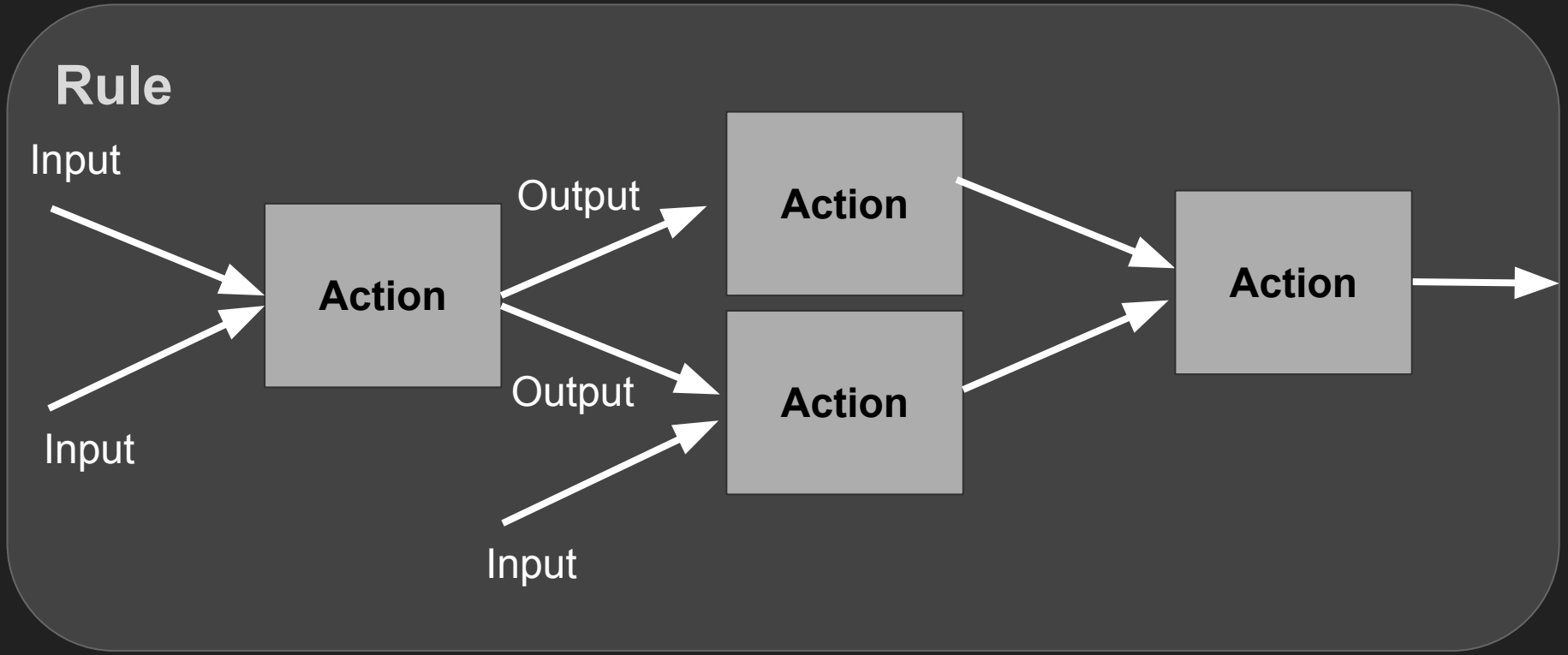
Что это за правила такие?

Правила определяют отношения между входными и выходными данными, а также шаги(экшены), необходимые для их преобразования.

Ввод | вывод

- Все входные и выходные данные должны быть явно заданы
- Входными и выходными данными могут быть файлы, CLI аргументы, версии OS, флаги, переменные окружения и т.д.

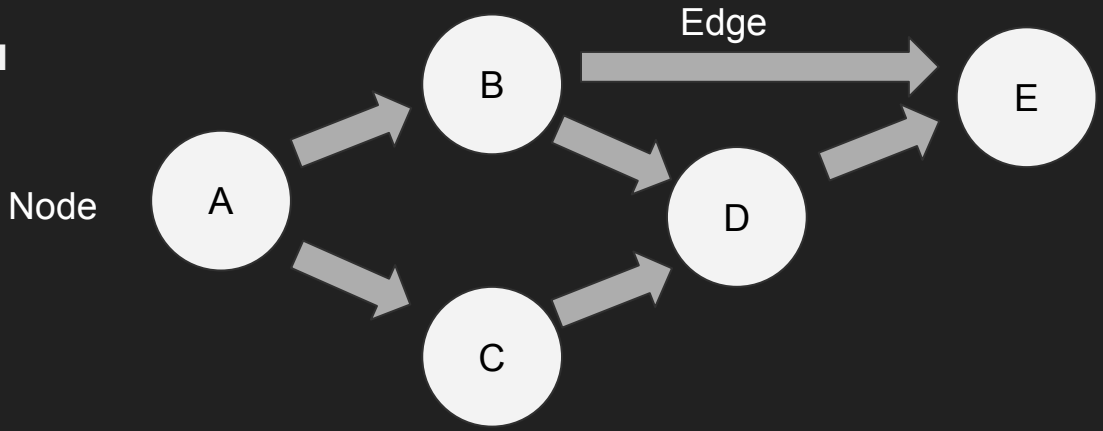
Большая картина



Graph

Правила и экшены используются чтобы построить **Направленный Ацикличный Граф**

Вершины - экшены
Ребра - зависимости



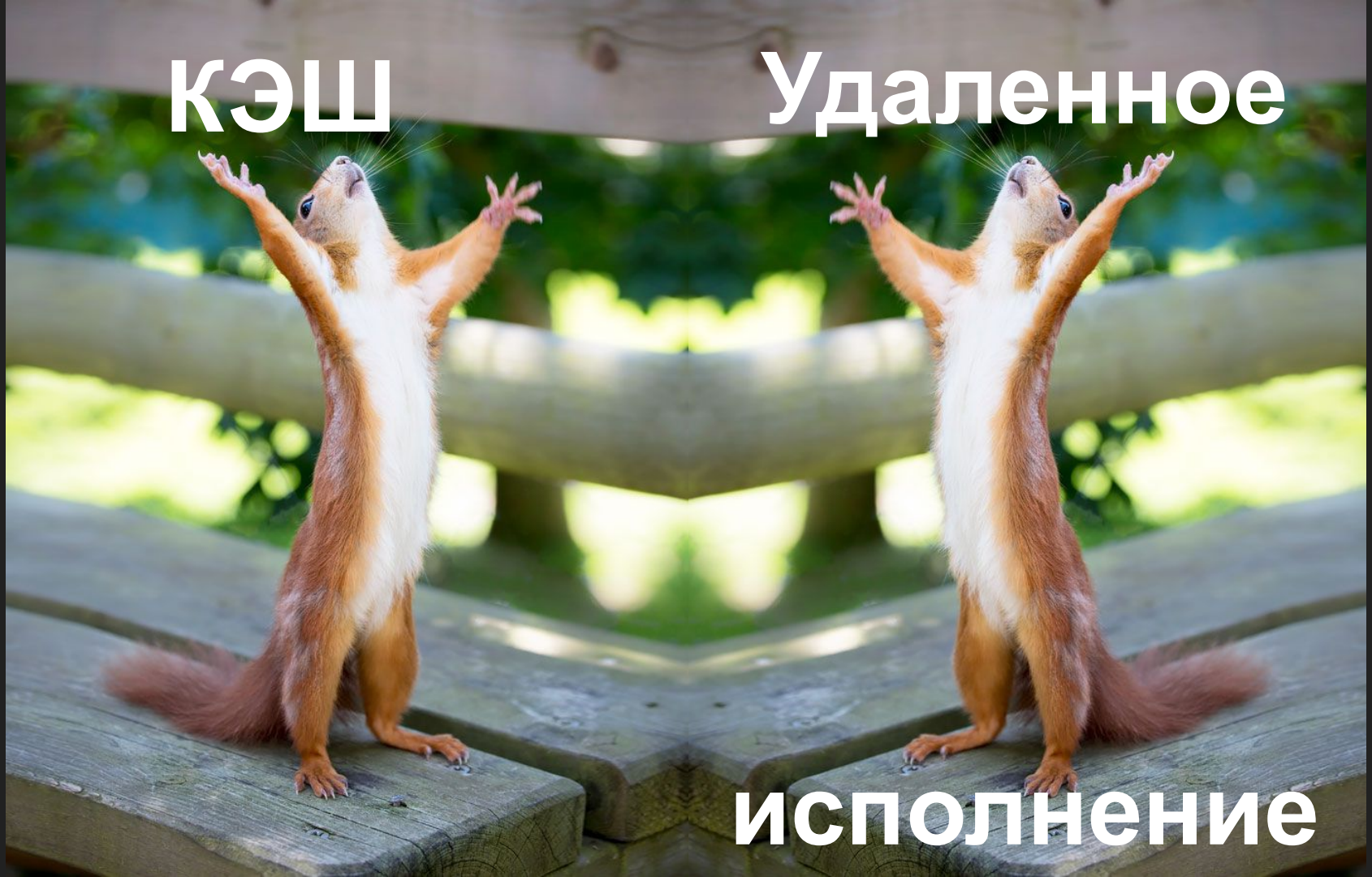
Динамическая модификация графа запрещена 

Все операции с графом - инкрементальные 

Ну и что?

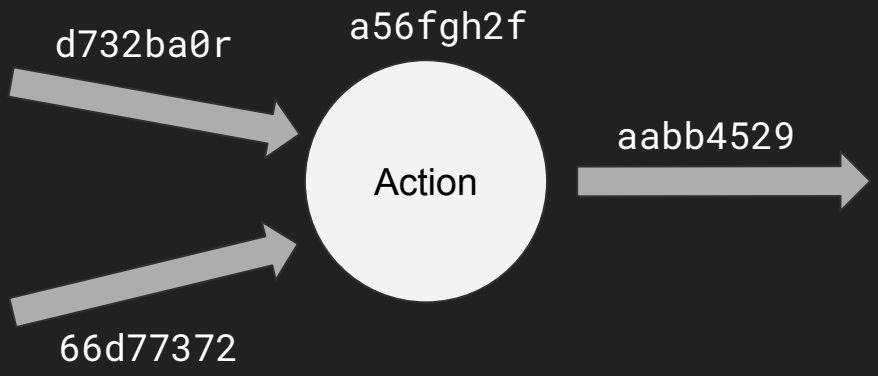
Ну и что?

Когда мы явно знаем все входные и выходные данные - мы можем использовать **КЭШ**!


КЭШ
Удаленное
исполнение

Кэш

- Входные данные + окружение + хэш правила - ключ кеша
- Если мы знаем ключ и все входные данные - мы можем получить выходные откуда угодно



Локальный кэш - папка

Удаленный кэш - сервер

Чем еще уникален Bazel?


SHOW ME
THE CODE!!!

Starlark (aka Skylark)

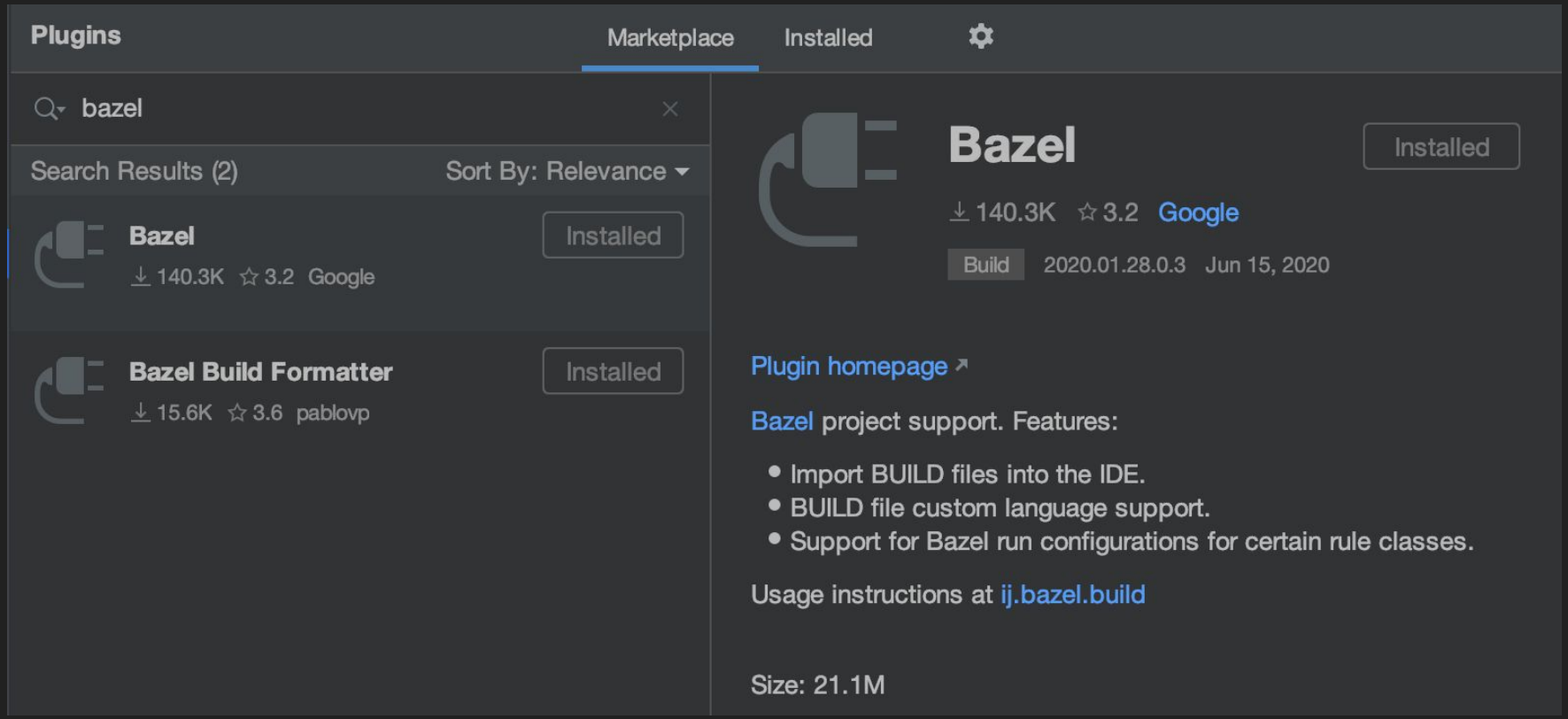
- Язык для написания правил
- Очень похож на Python 3
- Ограниченная стандартная библиотека
- Совместим с Buck

```
def fizz_buzz(n):
  """Print Fizz Buzz numbers from 1 to n."""
  for i in range(1, n + 1):
    s = ""
    if i % 3 == 0:
      s += "Fizz"
    if i % 5 == 0:
      s += "Buzz"
    print(s if s else i)

fizz_buzz(20)
```

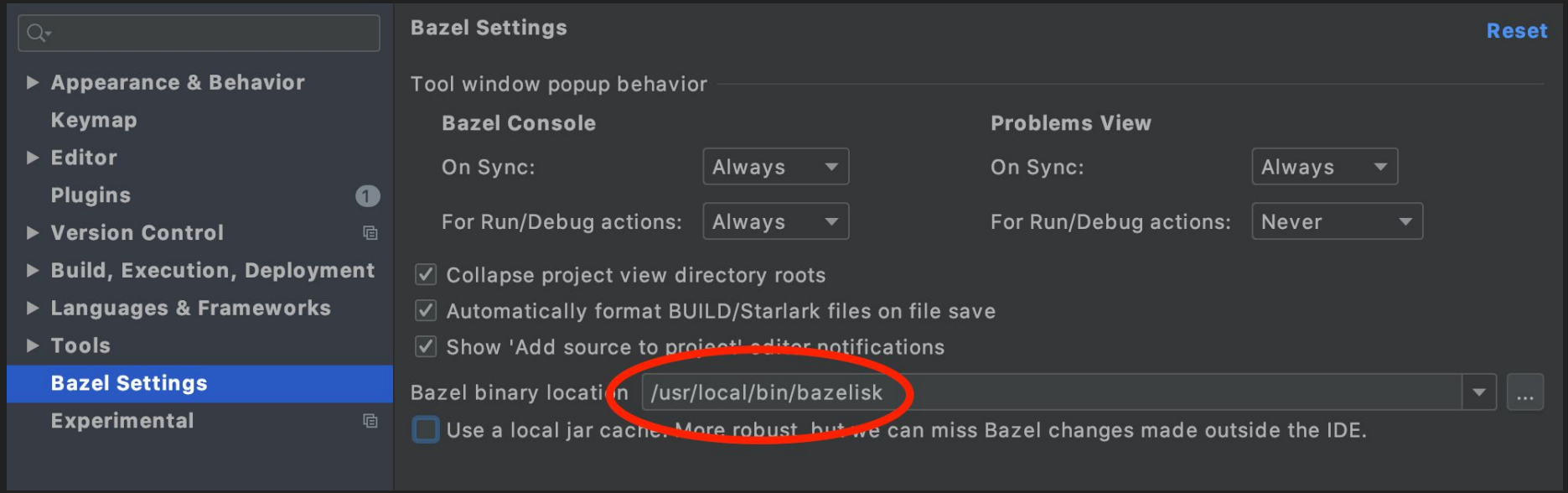
Установка Bazelisk aka Bazel Wrapper

```
~ brew install bazelisk
# or
~ npm install -g @bazel/bazelisk
```

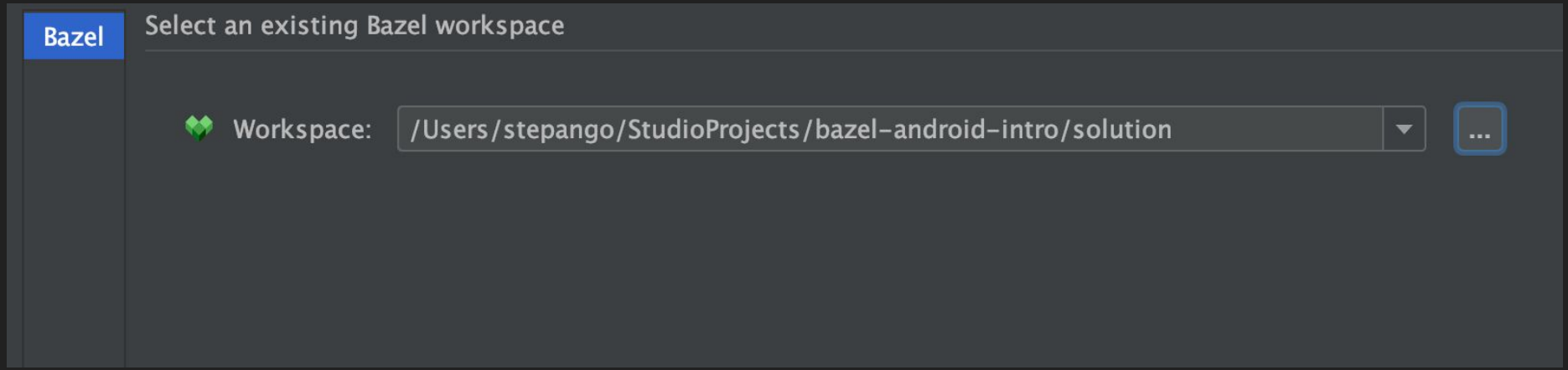
Android Studio (3.6) - Bazel plugin



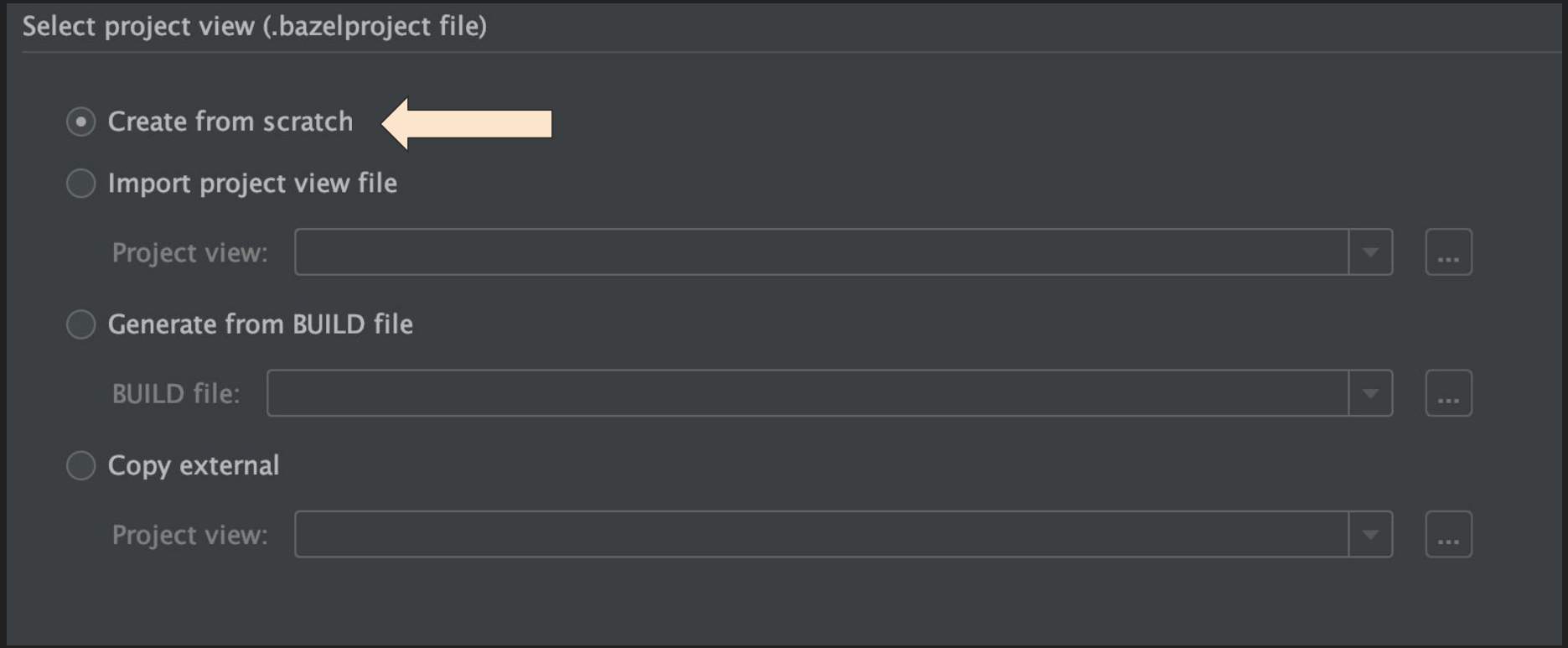
Поправим путь к бинарю



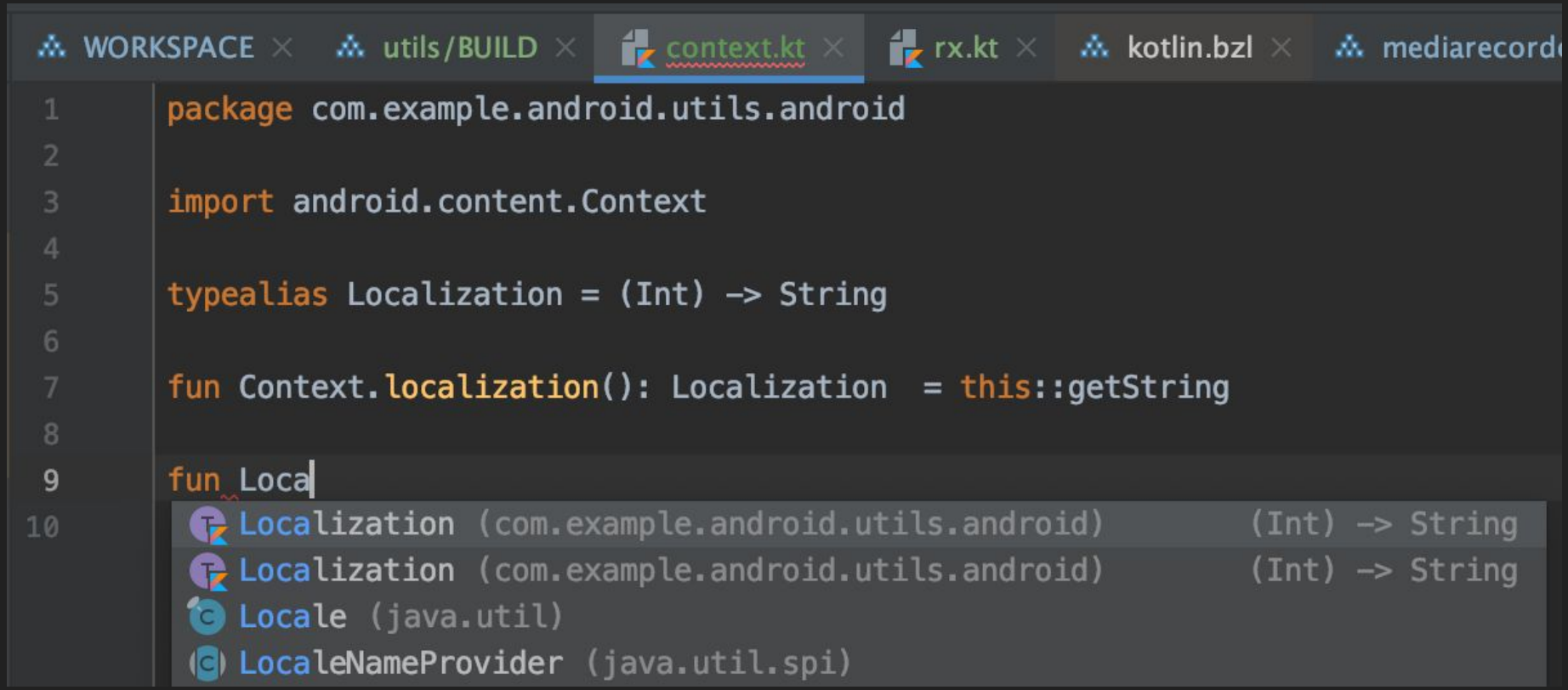
Импорт проекта

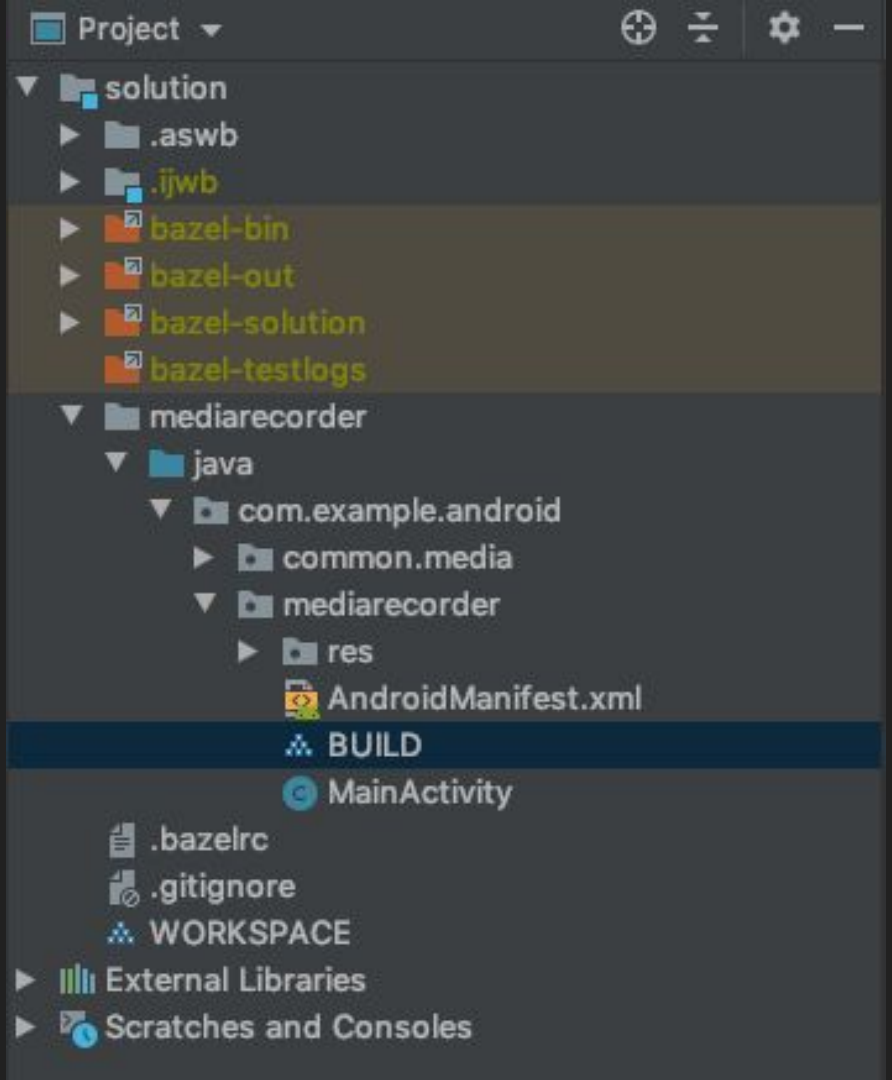


Импорт проекта



Отличная поддержка в IDE

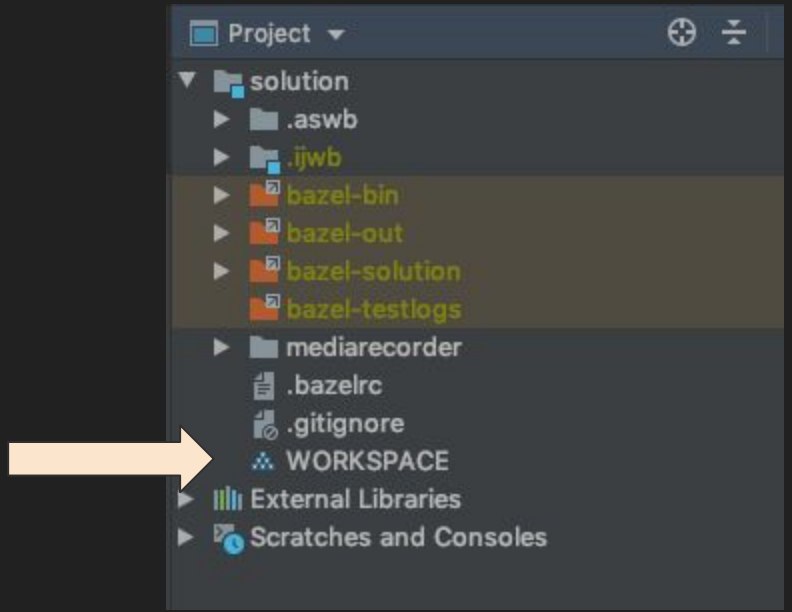



Project
solution
.aswb
.ijwb
bazel-bin
bazel-out
bazel-solution
bazel-testlogs
mediarecorder
java
com.example.android
common.media
mediarecorder
res
AndroidManifest.xml
BUILD
MainActivity
.bazelrc
.gitignore
WORKSPACE
External Libraries
Scratches and Consoles

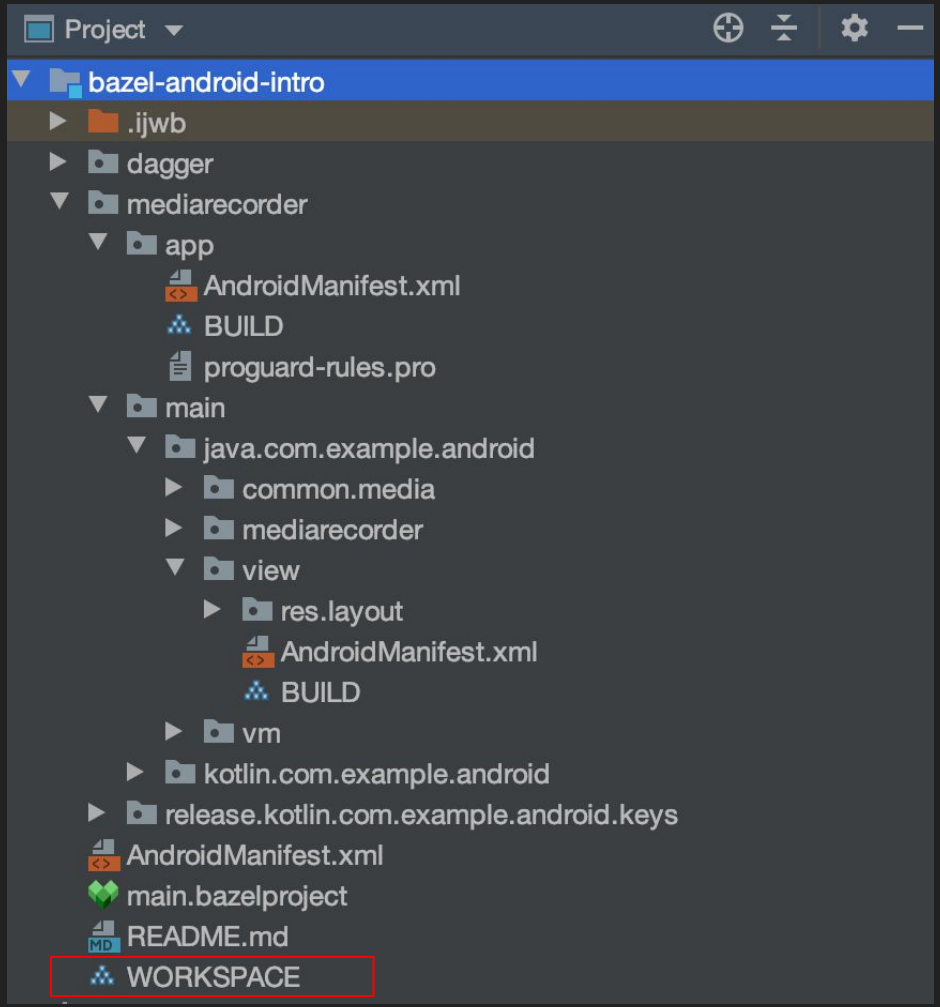
Plain Old Android

WORKSPACE == Root Project

WORKSPACE


Project
solution
.aswb
.ijwb
bazel-bin
bazel-out
bazel-solution
bazel-testlogs
mediarecorder
.bazelrc
.gitignore
WORKSPACE
External Libraries
Scratches and Consoles

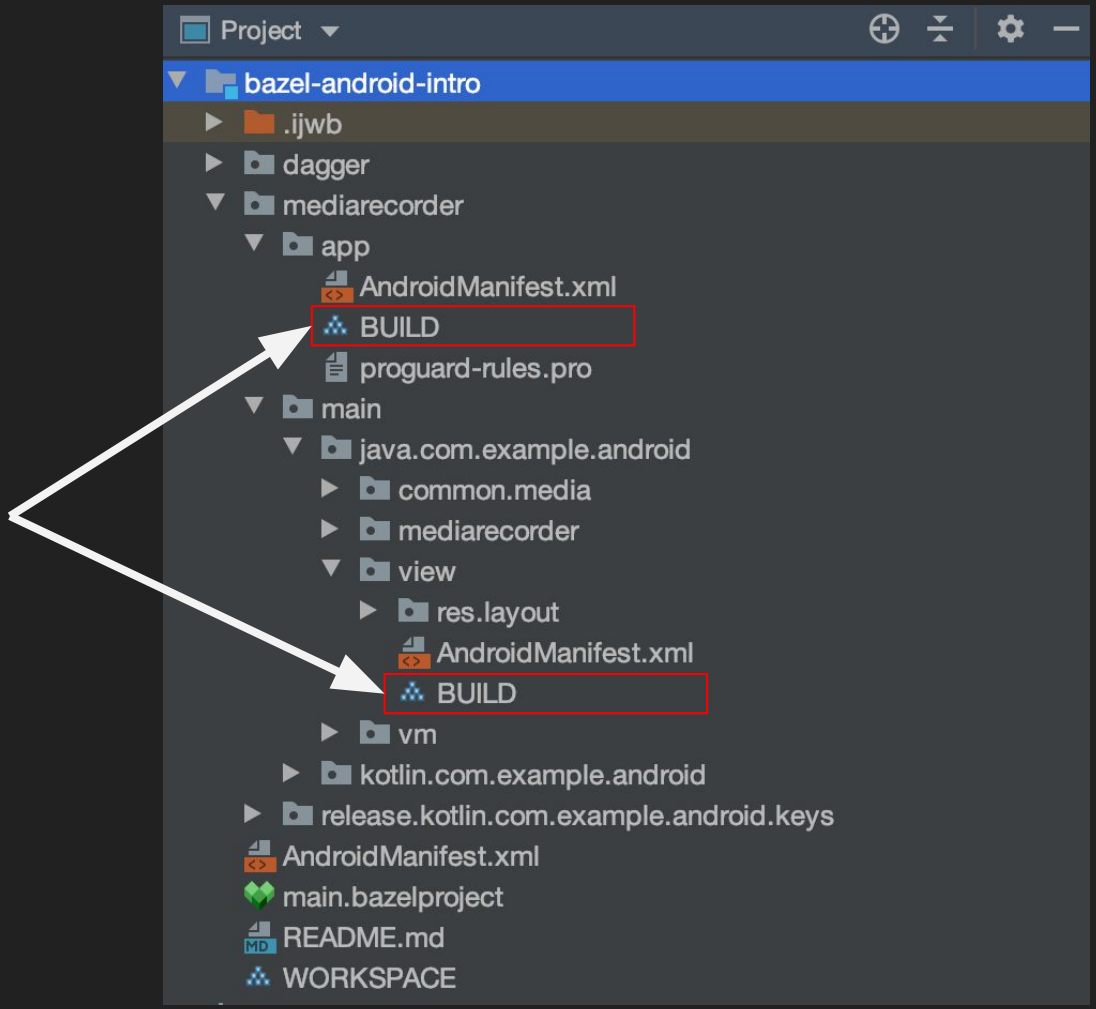
Структура проекта



Структура проекта

Каждый BUILD файл может содержать несколько таргетов(target)

Так в терминологии Bazel называются “модули”



WORKSPACE

```
# Первым делом!
# Загрузим Http Archive rule
# load - встроенная функция, аналог import
# @ - Root (WORKSPACE)
# bazel_tools - репозиторий, часть Bazel

load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")
 ^        ^                    ^                  ^          ^
fun   |Repository |      Package          |Extension|    Symbol     |

# Теперь мы можем использовать http_archive чтоб загружать правила
```

WORKSPACE

```
# Конфигурация Android SDK

load("@build_bazel_rules_android//android:rules.bzl", "android_sdk_repository")
android_sdk_repository(
    name = "androidsdk",
    api_level = 28,
)
```

build.gradle

```
subprojects {
    android {
        compileSdkVersion 28
    }
}
```

Android Java App - BUILD.bazel

```
# Создаем awesomeapp target (модуль)
android_binary(
    name = "awesomeapp",
    manifest = "AndroidManifest.xml",
    srcs = ["MainActivity.java"]
)
```

MainActivity is too big. How do I split it into multiple files

Asked 5 years, 7 months ago Active 5 years, 7 months ago Viewed 1k times

```
4079        public void rebuildAllFragments(boolean last) {
4080            if (layersActionBarLayout != null) {
4081                layersActionBarLayout.rebuildAllFragmentViews(last, last);
4082            } else {
4083                actionBarLayout.rebuildAllFragmentViews(last, last);
4084            }
4085        }
4086
4087        @Override
4088        public void onRebuildAllFragments(ActionBarLayout layout, boolean last) {
4089            if (AndroidUtilities.isTablet()) {
4090                if (layout == layersActionBarLayout) {
4091                    rightActionBarLayout.rebuildAllFragmentViews(last, last);
4092                    actionBarLayout.rebuildAllFragmentViews(last, last);
4093                }
4094            }
4095            drawerLayoutAdapter.notifyDataSetChanged();
4096        }
4097    }
```

Android Java App - BUILD.bazel

```
# Just manifest files

android_binary(
    name = "awesomeapp",
    manifest = "AndroidManifest.xml",
    srcs = glob(["*.java"])
)
```

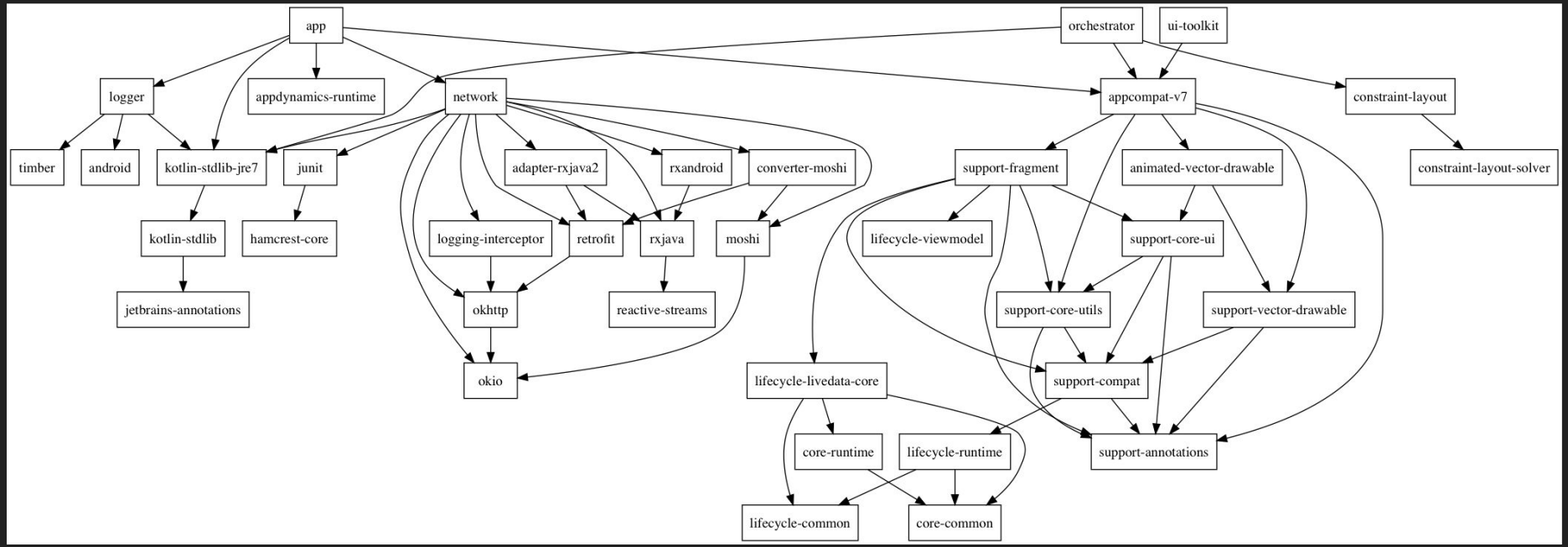
WORKSPACE

```
# Source of the Android build rules
# Bazel creates target with name supplied, now we can access it via @
# http_archive - just imported symbol

http_archive(
    name = "build_bazel_rules_android",
    urls = ["https://github.com/bazelbuild/rules_android/archive/v0.1.1.zip"],
    sha256 = "cd06d15dd8bb59926e4d65f9003bfc20f9da4b2519985c27e190cddc8b7a7806",
    strip_prefix = "rules_android-0.1.1",
)
```

Gradle - build.gradle (root)

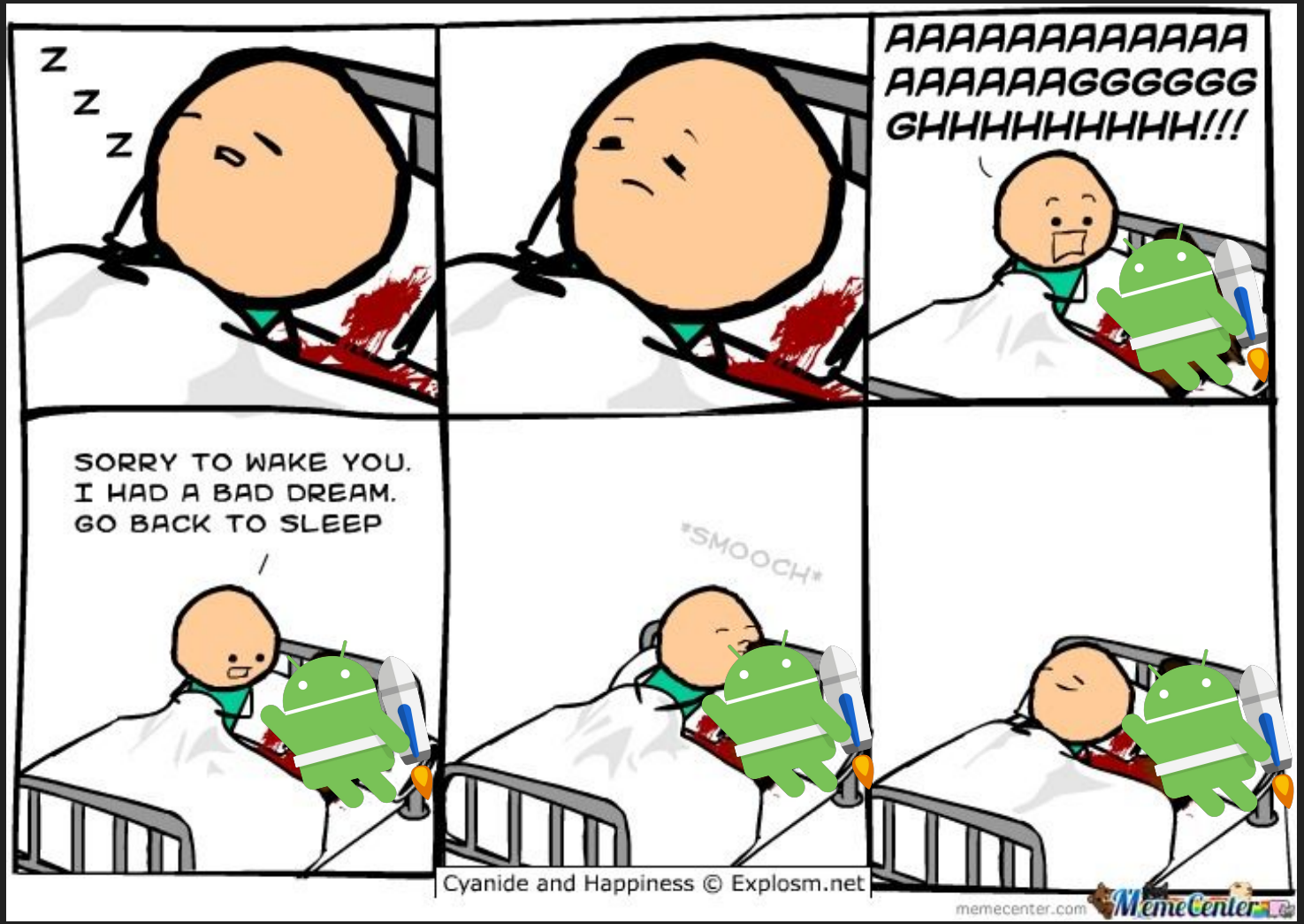
```
buildscript {
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.1'
    }
}
```

Я знаю что делаю!!!



Android library - BUILD.bazel

```
android_library(
    name = "main",
    srcs = ["MainActivity.java"],
    manifest = "AndroidManifest.xml",
    resource_files = glob(["res/**"]),
    deps = [
        # Зависимости
        "//awesomeapp/utils:string",
        ...
    ],
)
```


Z
Z
Z
AAAAAAAAAAAA
AAAAAAGGGGGG
GHHHHHHHHH!!!
SORRY TO WAKE YOU.
I HAD A BAD DREAM.
GO BACK TO SLEEP
SMOOCH
Cyanide and Happiness © Explosm.net
memecenter.com
MemeCenter

WORKSPACE

```
# Нам нужно научить Bazel работать с Maven

http_archive(
    name = "rules_jvm_external",
    strip_prefix = "rules_jvm_external-3.2",
    url = "https://github.com/bazelbuild/rules_jvm_external/archive/3.2.zip",
)

load("@rules_jvm_external//:defs.bzl", "maven_install")

# Ну вот, теперь можно использовать внешние зависимости
```

Rules JVM External - WORKSPACE

```
# Define repositories and list of artifacts
maven_install(
    artifacts = [
        "androidx.paging:paging-runtime:" + paging_version
        ...
    ],
    repositories = [
        "https://jcenter.bintray.com/",
    ],
)
```

build.gradle

```
ext {
    supportAnnotations = "androidx.paging:paging-runtime:" + paging_version
}
allprojects{
    repositories {
        jcenter()
    }
}
```

Сгенерированные имена зависимостей

```
"androidx.paging:paging-runtime:" + paging_version
```

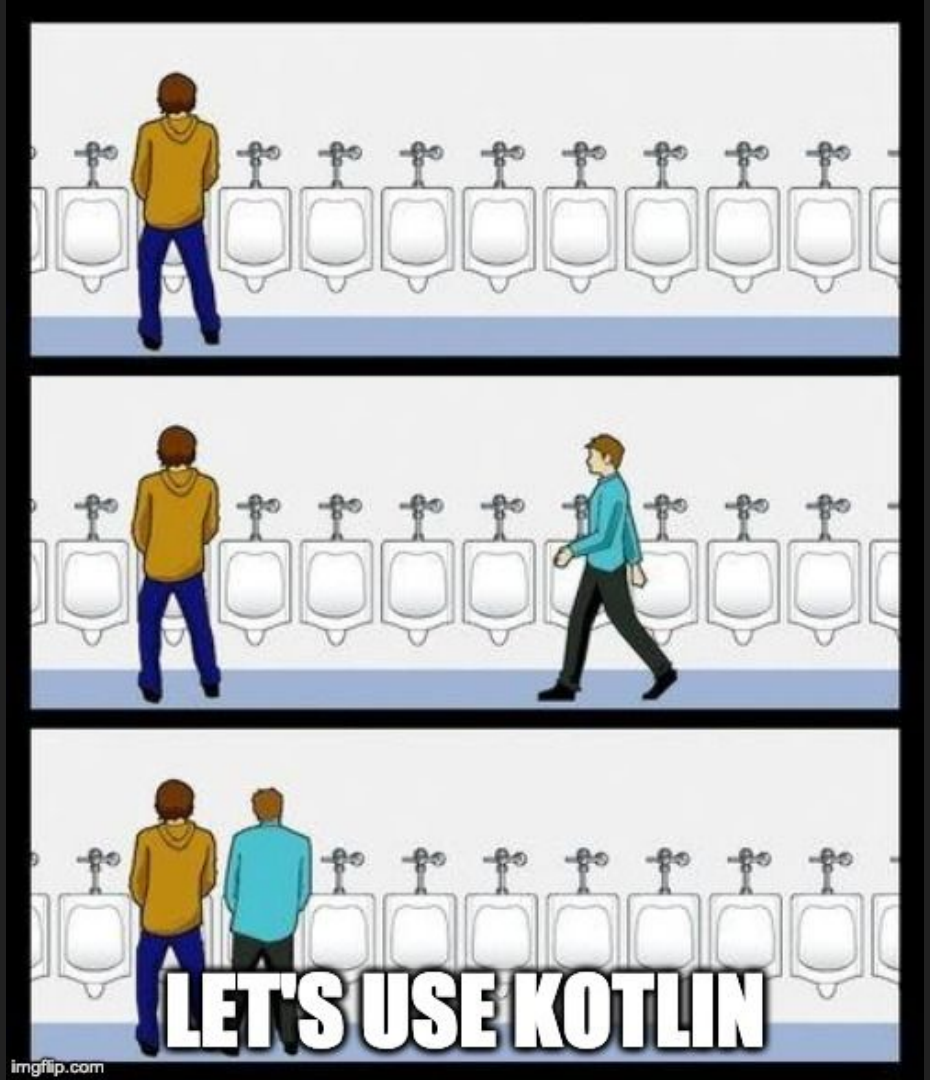
```
"@maven://androidx_paging_paging_runtime"
```

Android App - BUILD.bazel

```
load("@build_bazel_rules_android//android:rules.bzl", "android_binary")

android_binary(
    name = "mediarecorder_dev",
    manifest = "AndroidManifest.xml",
    deps = [":main"],
    crunch_png = False,
)
```

Android App - BUILD.bazel

```
load("@build_bazel_rules_android//android:rules.bzl", "android_library")

# Only java files
android_library(
    name = "main",
    srcs = ["MainActivity.java"],
    manifest = "AndroidManifest.xml",
    resource_files = glob(["res/**"]),
    deps = [
        # link to your local target
        "//mediarecorder/java/com/example/android/common/media",
        "@maven//:com_android_support_support_annotations"
    ],
)
```


LET'S USE KOTLIN
imgflip.com

Kotlin rules #1 - WORKSPACE

```
rules_kotlin_version = "legacy-1.3.0"

http_archive(
    name = "io_bazel_rules_kotlin",
    url = "https://github.com/bazelbuild/rules_kotlin/archive/legacy-1.3.0.zip",
    strip_prefix = "rules_kotlin-legacy-1.3.0",
)
```

Kotlin rules #2 - WORKSPACE

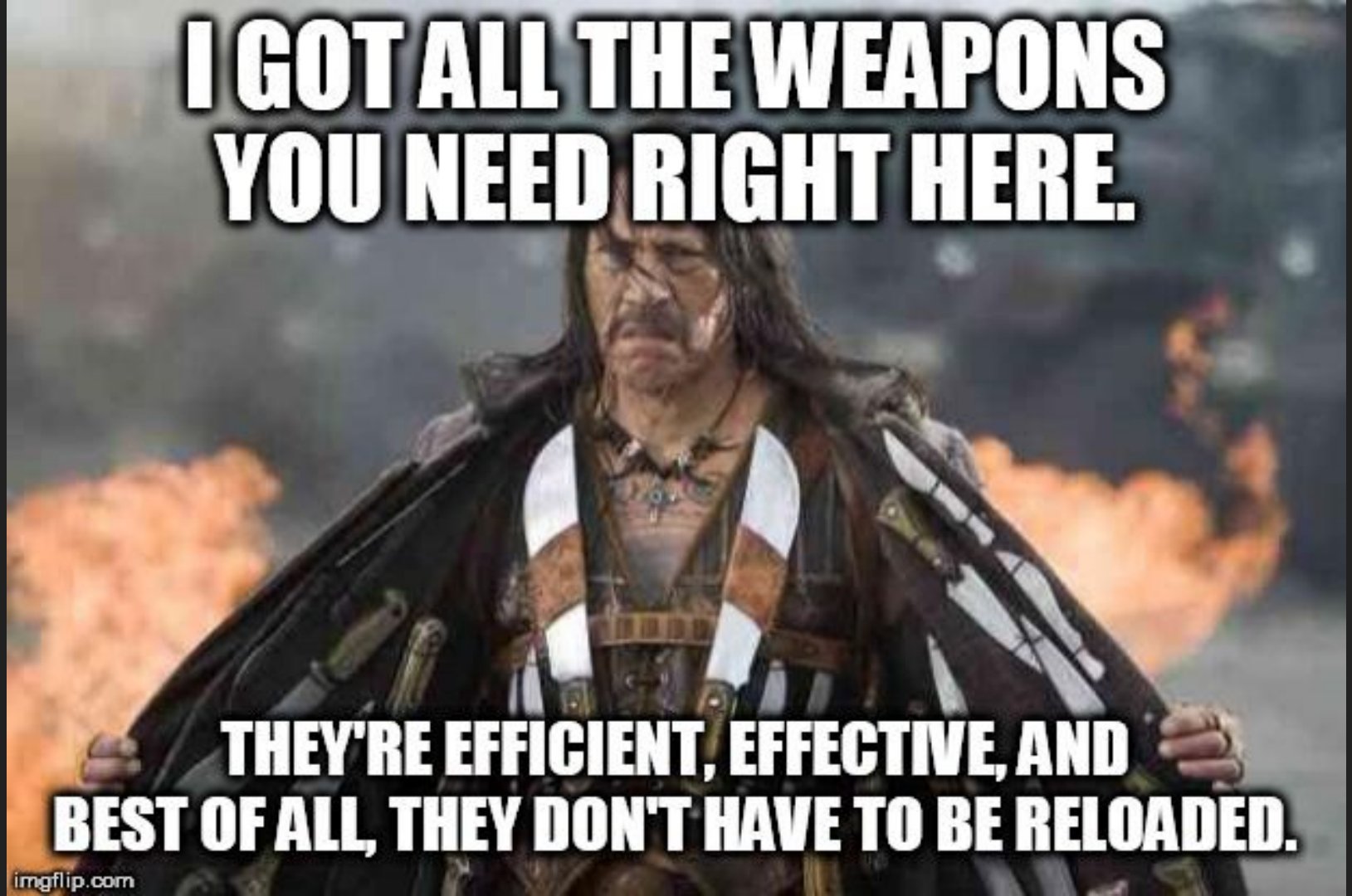
```
load("@io_bazel_rules_kotlin//kotlin:kotlin.bzl",
     "kotlin_repositories",
     "kt_register_toolchains"
)

kotlin_repositories()
kt_register_toolchains()
```

Kotlin libraries - BUILD.bazel

```
# Finally Kotlin
load("@io_bazel_rules_kotlin//kotlin:kotlin.bzl", "kt_android_library", "kt_jvm_library")


kt_jvm_library(
    name = "jvm_utils",
    srcs = glob([
        "jvm/*.kt",  # It's a good practice to create Bazel target per package
        "jvm/*.java" # Kotlin could also compile java files
    ]),
    deps = [
        "@maven//:io_reactivex_rxjava2_rxjava",
    ],
)


# We can declare multiple targets in single BAZEL file
kt_android_library(
    name = "android_utils",
    srcs = glob(["android/*.kt"]),
)
```

Kotlin libraries - BUILD.bazel

```
load("@io_bazel_rules_kotlin//kotlin:kotlin.bzl", "kt_jvm_test")

kt_jvm_test(
    name = "jvm_utils_test",
    srcs = glob([
          # It's a good practice to create Bazel target per package
          "jvm/*.kt",
    ]),
    deps = [
          "@maven//:io_reactivex_rxjava2_rxjava",
          "@maven//:junit_junit",
    ],
)

# No kotlin_android_test support for now :(
```


I GOT ALL THE WEAPONS
YOU NEED RIGHT HERE.
THEY'RE EFFICIENT, EFFECTIVE, AND
BEST OF ALL, THEY DON'T HAVE TO BE RELOADED.
imgflip.com

Dagger #1 - BUILD.bazel

```
# Настройка dagger compiler
java_library(
    name = "compiler",
    exports = [
        "@maven//:com_google_dagger_dagger_compiler",
    ],
    # Флаг позволит нам исключить библиотеку из APK
    neverlink = True,
    # Ограничиваем область видимости
    visibility = ["//visibility:private"]
)
```

Dagger #2 - BUILD.bazel

```
# Annotation Processor plugin
java_plugin(
    name = "plugin",
    processor_class = "dagger.internal.codegen.ComponentProcessor",
    deps = [
        ":compiler",
        "@maven//:javax_inject_javax_inject",
        "@maven//:com_google_dagger_dagger",
    ],
    visibility = ["//visibility:private"]
)
```

Dagger #3 - BUILD.bazel

```
# Define java plugin for annotation processing
java_library(
    name = "lib",
    exported_plugins = ["plugin"],
    exports = [
        ":compiler",
        "@maven//:com_google_dagger_dagger",
        "@maven//:javax_inject_javax_inject",
    ],
    visibility = ["//visibility:public"]
)
```

Dagger usage - BUILD.bazel

```
# Now we can use dagger anywhere
kt_jvm_library(
    name = "info",
    srcs = glob(["*.kt"]),
    deps = [
        # Like this
        "//dagger:lib",
    ],
)
```

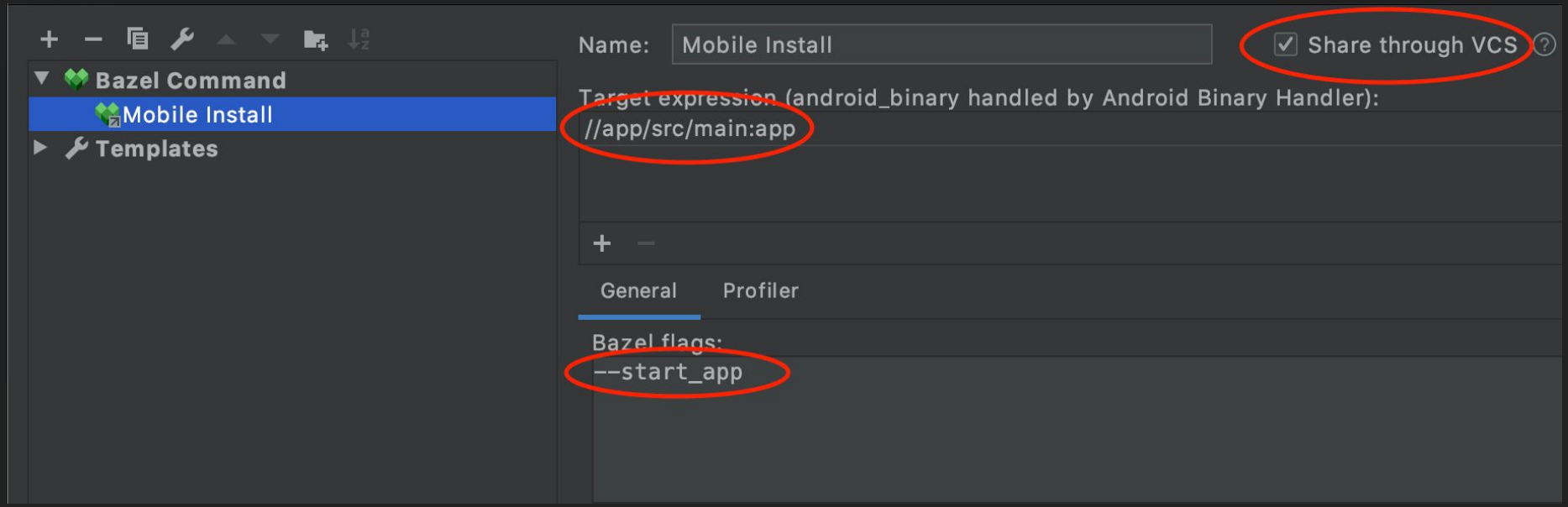
Теперь у нас есть все о чем мечтают Android разработчики

Как теперь запустить приложение?

Mobile-install

- Альтернатива Apply Changes
- В большинстве случаев быстрее
- Надежнее
- И старше

IDE Setup


Bazel Command
Mobile Install
Templates
Name:
Mobile Install
Share through VCS
Target expression (android_binary handled by Android Binary Handler):
//app/src/main:app
General
Profiler
Bazel flags:
--start_app

А запускать тесты?

```
$ bazelisk test //...
```

А как же `%gradle_feature%`?

Manifest Values & Multidex - BUILD.bazel

```
android_binary(
    name = "app",
    manifest = "AndroidManifest.xml",
    manifest_values = {
        "versionCode": "1",
        "versionName": "1.0",
        "minSdkVersion": "21",
        "targetSdkVersion": "29",
    },
    ...
    multidex = "legacy" # off|manual_main_dex|native
)
```

Android App Flavors - BUILD.bazel

```
android_binary(
    name = "app_release",
    manifest = "AndroidManifest.xml",
    deps = [":main"],
    crunch_png = True,
    shrink_resources = 1,
    proguard_specs = [
        "proguard-rules.pro"
    ]
    ...
)
```

```
android_binary(
    name = "app_debug",
    manifest = "AndroidManifest.xml",
    deps = [":main"],
    custom_package = "com.yourapp.debug"
)
```

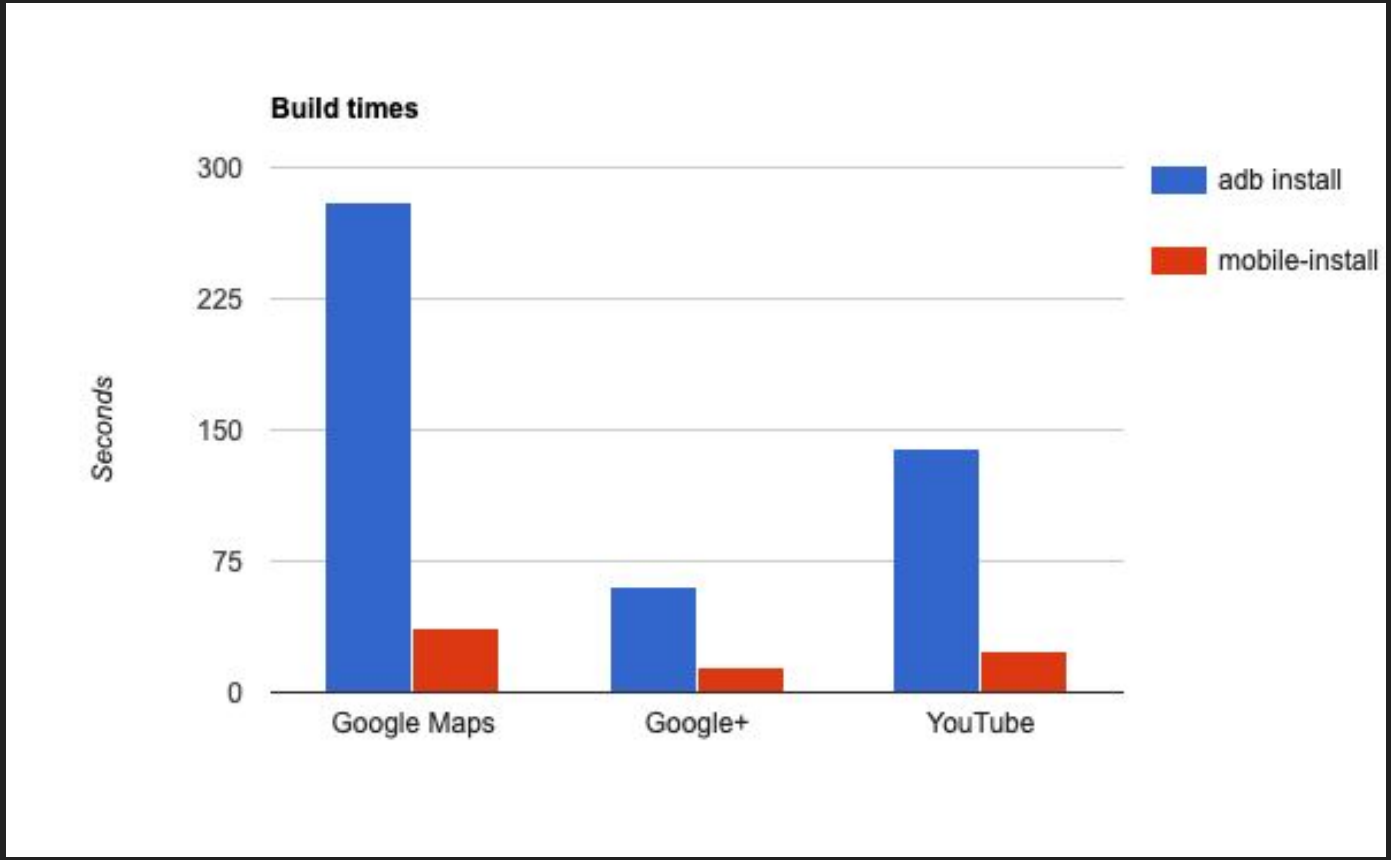
Подпись релизных билдов

- Android Bazel Rules не поддерживает подписи релизных билдов
- Ключи не должны храниться в репозитории
- Всегда можно реализовать собственные Rules, либо заменить на внешний скрипт

Bazel CLI

```
# bazel mobile-install //src:app_debug
# bazel build //app_release
# bazel test //...
```

А как же скорость компиляции?



https://docs.bazel.build/versions/master/mobile-install.html

Но почему все используют Gradle?

Причины

- По инерции
- Инкрементальная компиляция
- Больше доступной информации
- Готовые плагины
- Новые фишечки

Bazel плюсы

- Масштаб - эффективно работает с огромными приложениями и монорепами
- Надежный кеш(локальный и удаленный)
- Десятки поддерживаемых языков
- Быстрый синк с IDE
- Удаленная|смешанная компиляция
- Кеш конфигурации
- Огромное комьюнити
- Bazel query - язык запросов
- И т.д.

Bazel минусы

- Starlark имеет много ограничений
- Маленькие модули вместо инкрементальной компиляции
- Явная декларация зависимостей
- Поддержка IDE могла быть и лучше
- Придется изучать еще одну билд систему

Чего еще не хватает в Bazel?

- DataBindings v2
- Kotlin Native
- Instant Apps
- Compose
- Продвинутого управления зависимостями

Попробуйте!

Примеры кода можно найти тут

👇

https://github.com/stepango/**bazel-android-intro**
https://github.com/stepango/**bazel-android-workshop**

Всем спасибо!

Вопросы?