

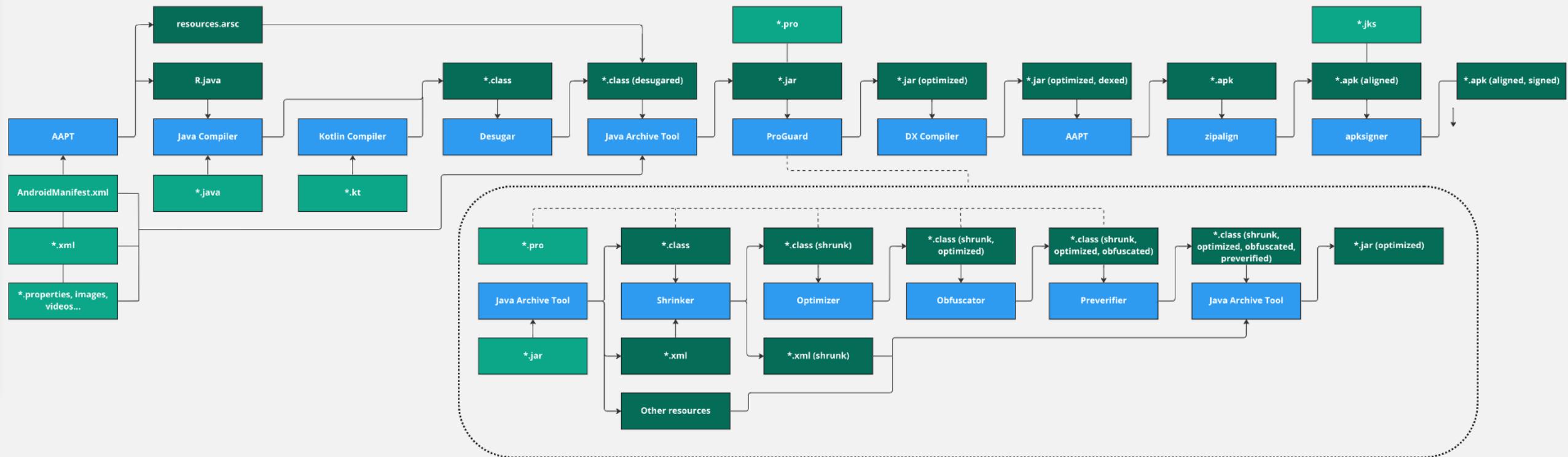
# Сборка Android приложения

Машенко Богдан

# Сборка Android приложения

Что значит «сборка Android приложения»?

# Сборка Android приложения



# Сборка Android приложения

1. Компилярование Java/Kotlin-исходников.

.java source -> javac compiler -> .class files  
.kt source -> kotlinc compiler -> .class files

2. Конвертация Java-байткода в Dalvik-байткод.

.class, .jar, .aar -> dx -> classes.dex

3. Сборка ресурсов и .dex-файлов в APK.

classes.dex, resource files -> aapt -> .apk file

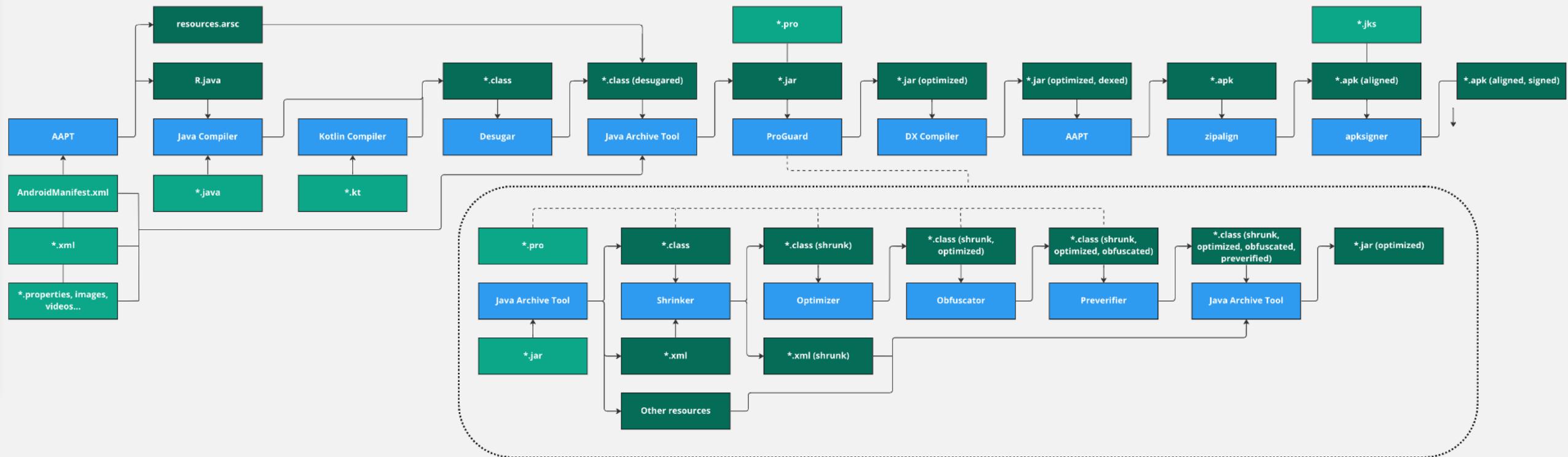
4. Оптимизация и подпись .apk-файла.

.apk file -> zipalign -> apksigner -> signed .apk file

# Виртуальные машины

- Dalvik
  - Модель компиляции JIT (Just-in-time)
- ART (с версии 4.4, KitKat)
  - Модель компиляции AOT (Ahead-of-time)
  - С версии 7.0 Nougat JIT-компилятор с профилированием кода для ART

# Сборка Android приложения



# Оптимизация сборок

1. Оптимизация сборки
2. ProGuard
3. D8, R8



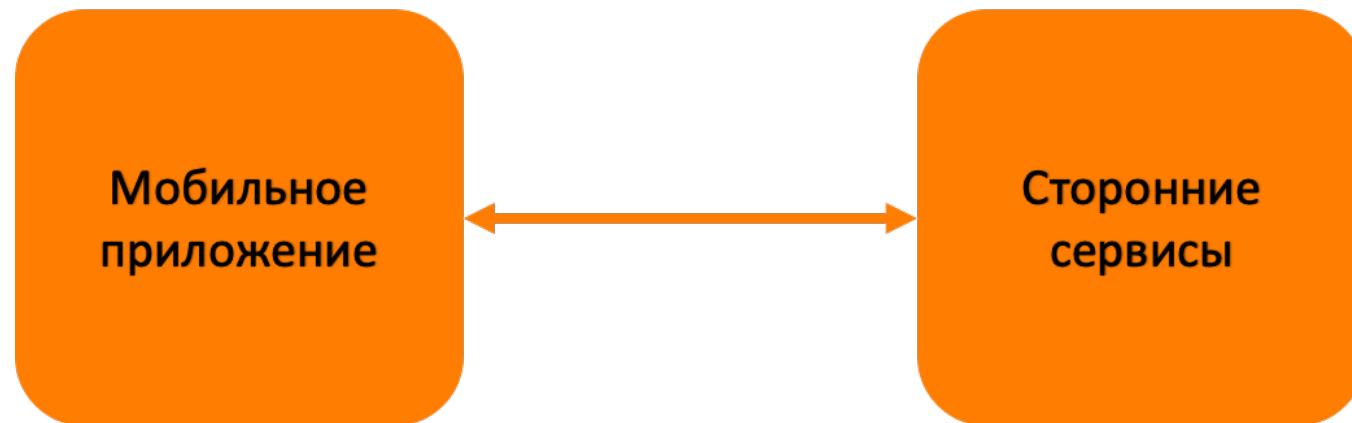
# Введение

Как пользователь/разработчик я хочу от приложения

1. Размер → 0 бит
2. Самое быстрое
3. Самое безопасное

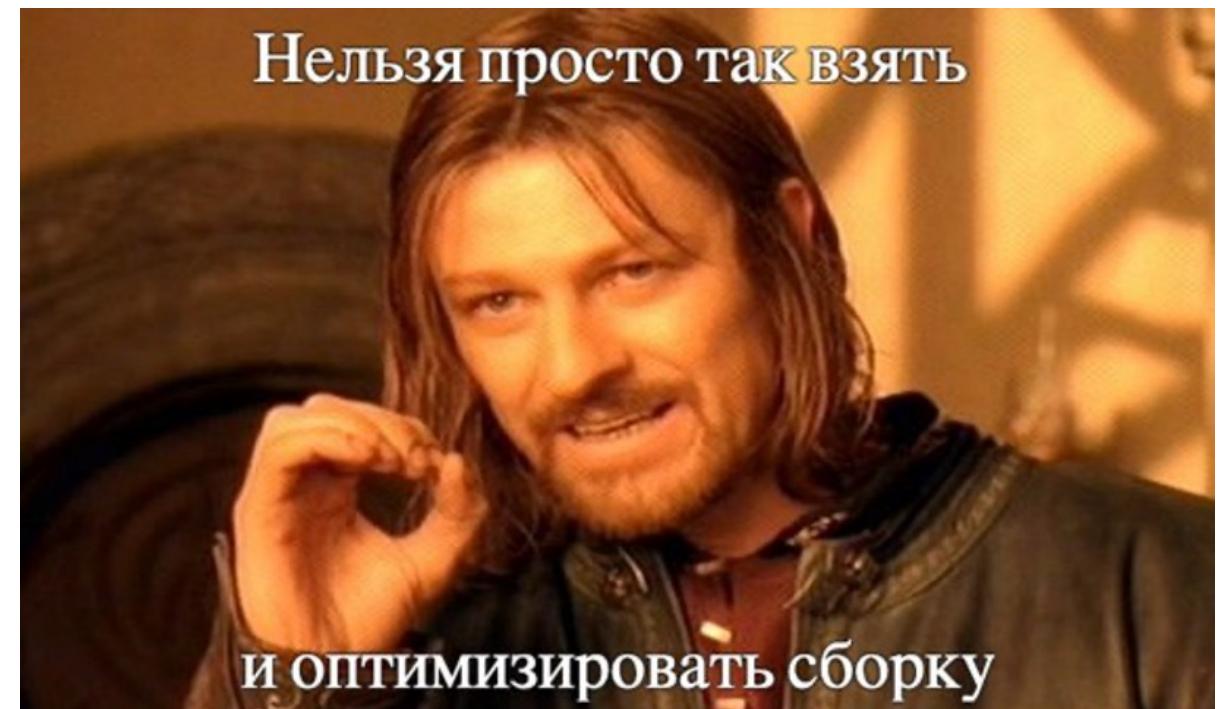
# Оптимизация сборки

## Бизнес логика

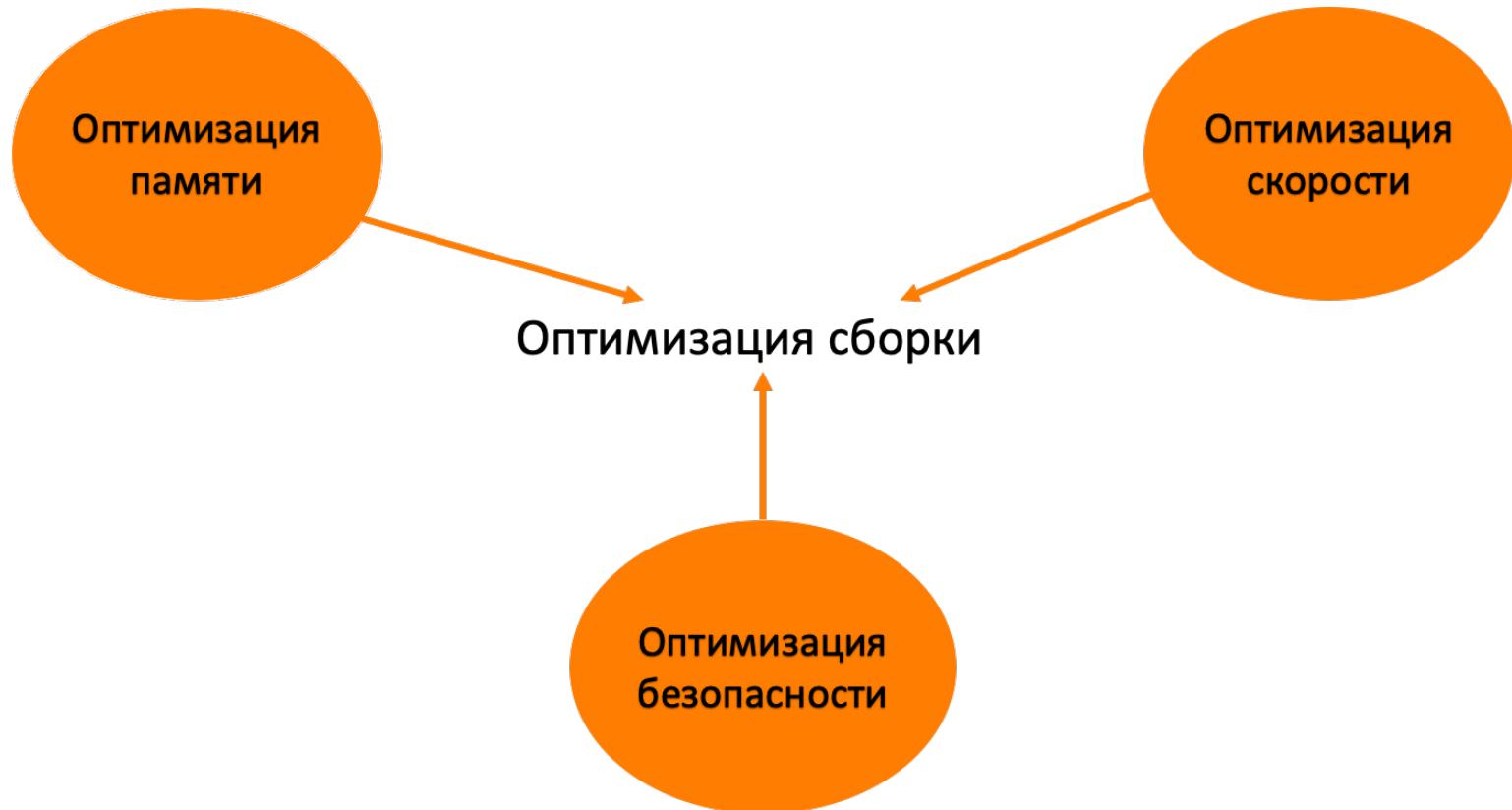


# Оптимизация сборки

1. Минимизировать потребляемую память (App bundle, хранение данных во вне)
2. Повысить скорость работы (правильно выбирать АиСД, оптимизировать код и тд)
3. Повысить безопасность (шифровать данные приложения, пароли, ПИН коды)



# Оптимизация сборки



# Оптимизация сборки

Как оптимизировать сборку:

- Удалять неиспользуемый байт-код, ресурсы
- Оптимизировать байт-код
- Запутывать байт-код

# ProGuard

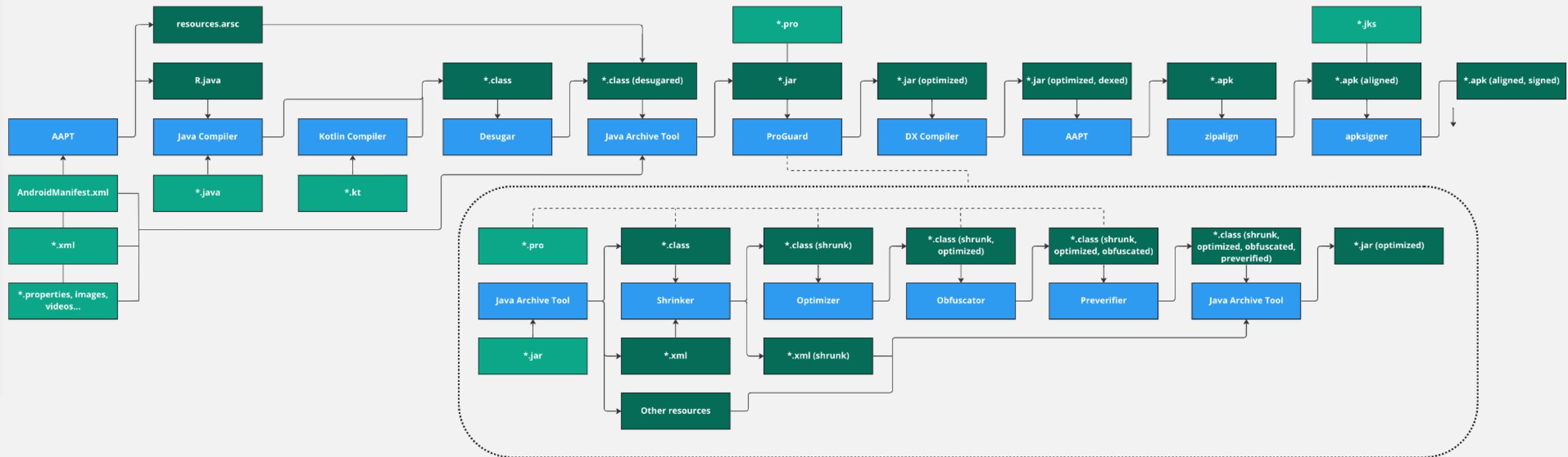
- «ProGuard is a Java class file shrinker, optimizer, obfuscator, and preverifier».
- Guardsquare - компания, занимающаяся разработкой программ для защиты мобильных приложений.

# ProGuard

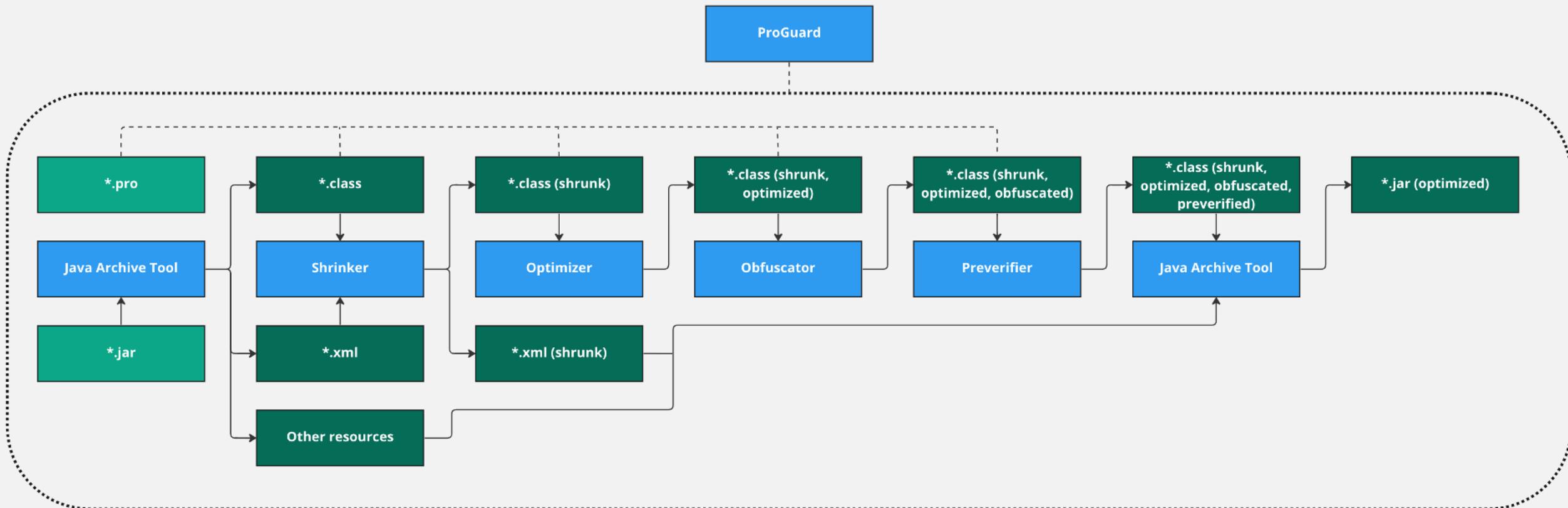
```
buildTypes {  
    release {  
        minifyEnabled true  
        proguardFile 'proguard-rules.txt'  
    }  
}
```

- proguard-rules.txt — файл, в который предлагается писать свои правила для ProGuard.
- За включение ProGuard отвечает директива minifyEnabled

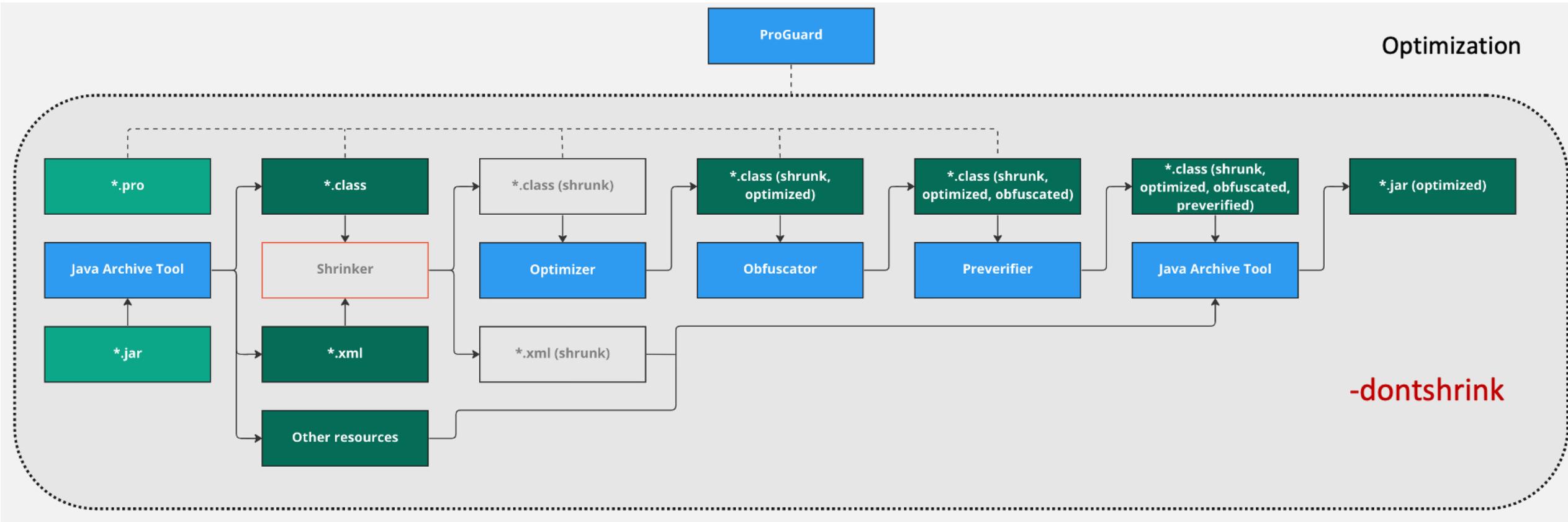
# ProGuard



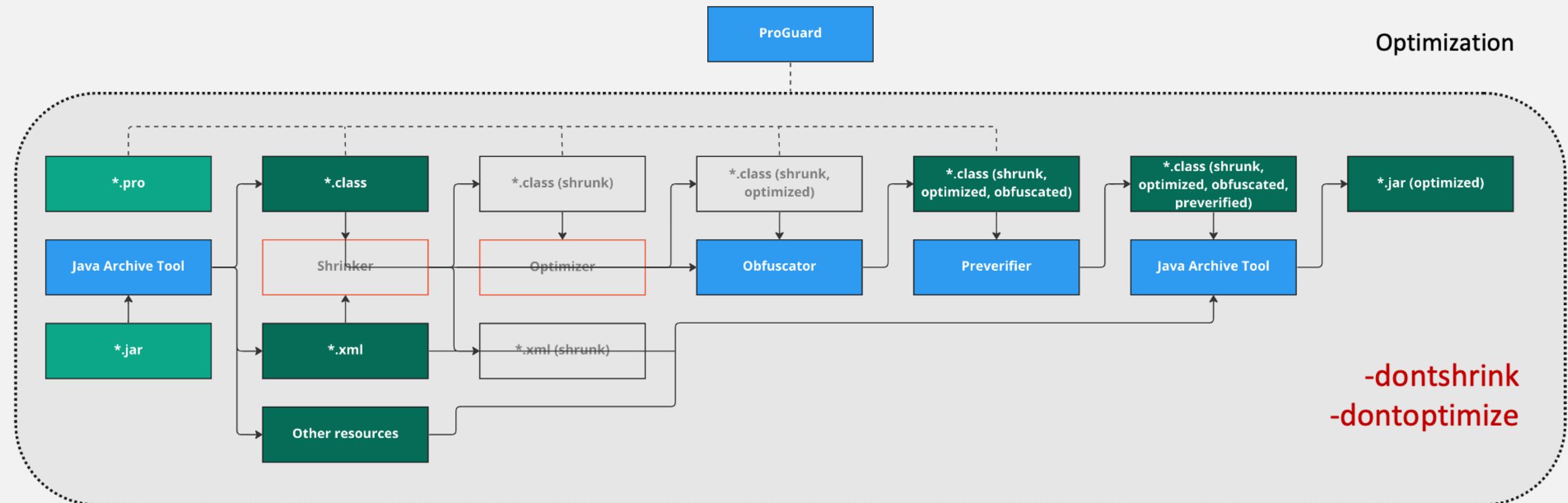
# ProGuard



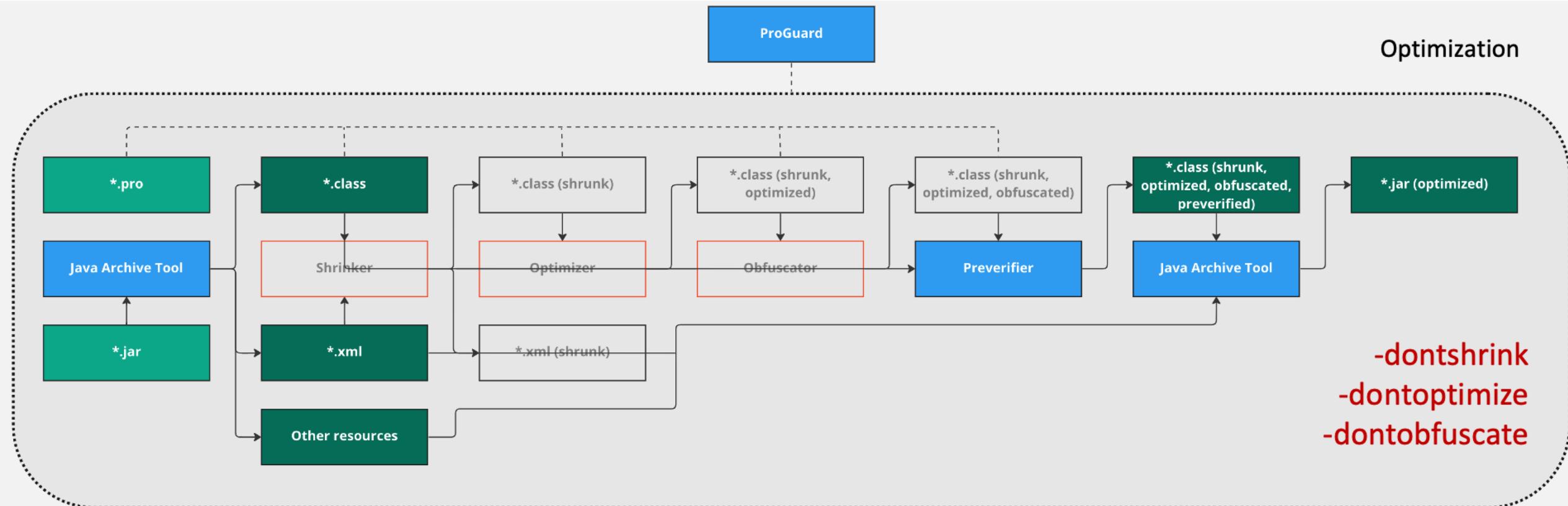
# ProGuard



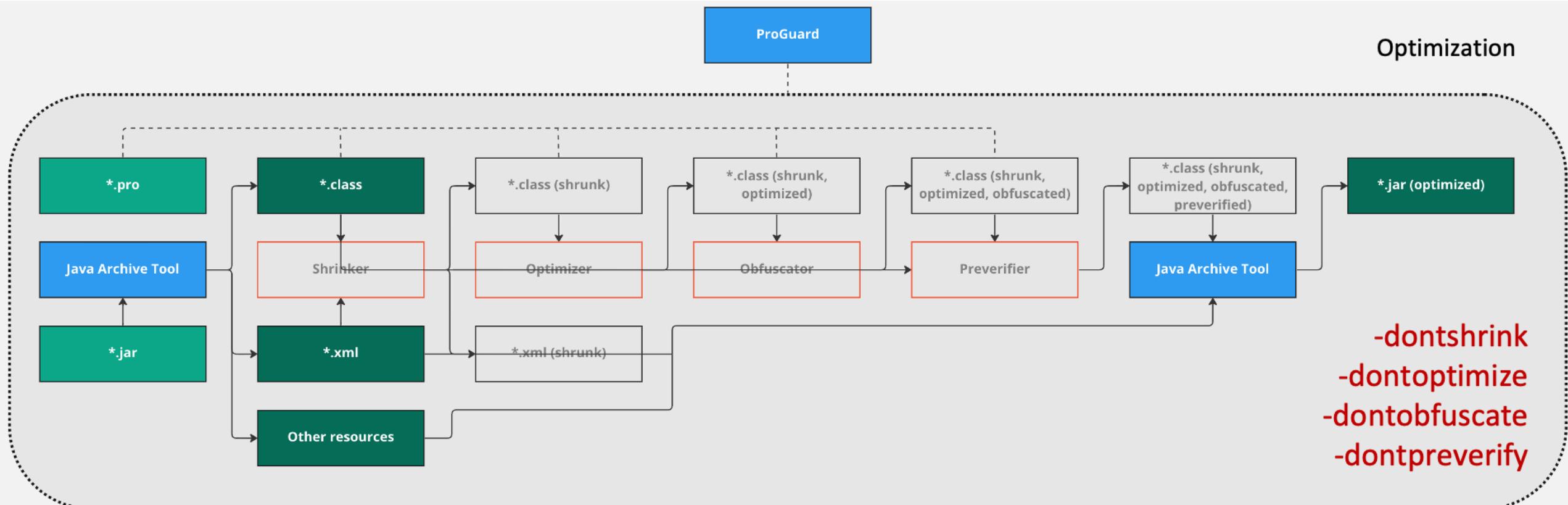
# ProGuard



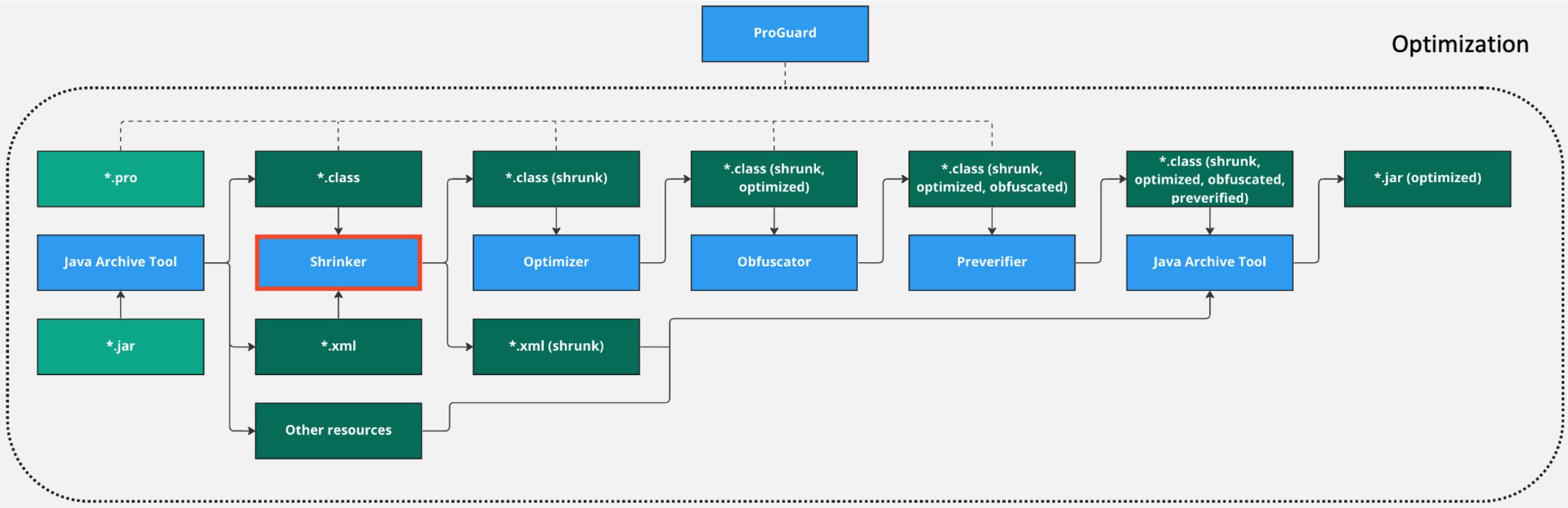
# ProGuard



# ProGuard

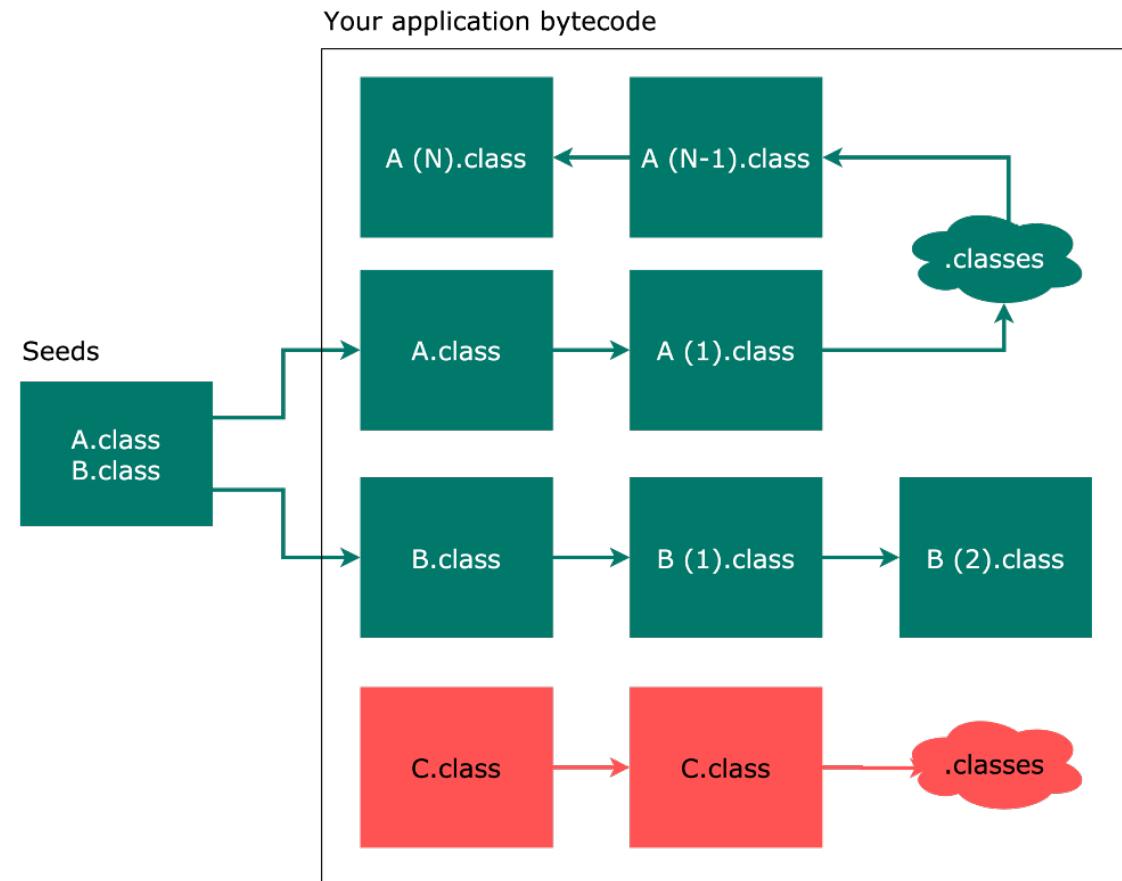


# ProGuard



# ProGuard

1. Определяются входные точки в программу (seeds)
2. Для каждой входной точки вычисляется граф достижимого кода
3. Достижимый код сохраняется, остальное удаляется

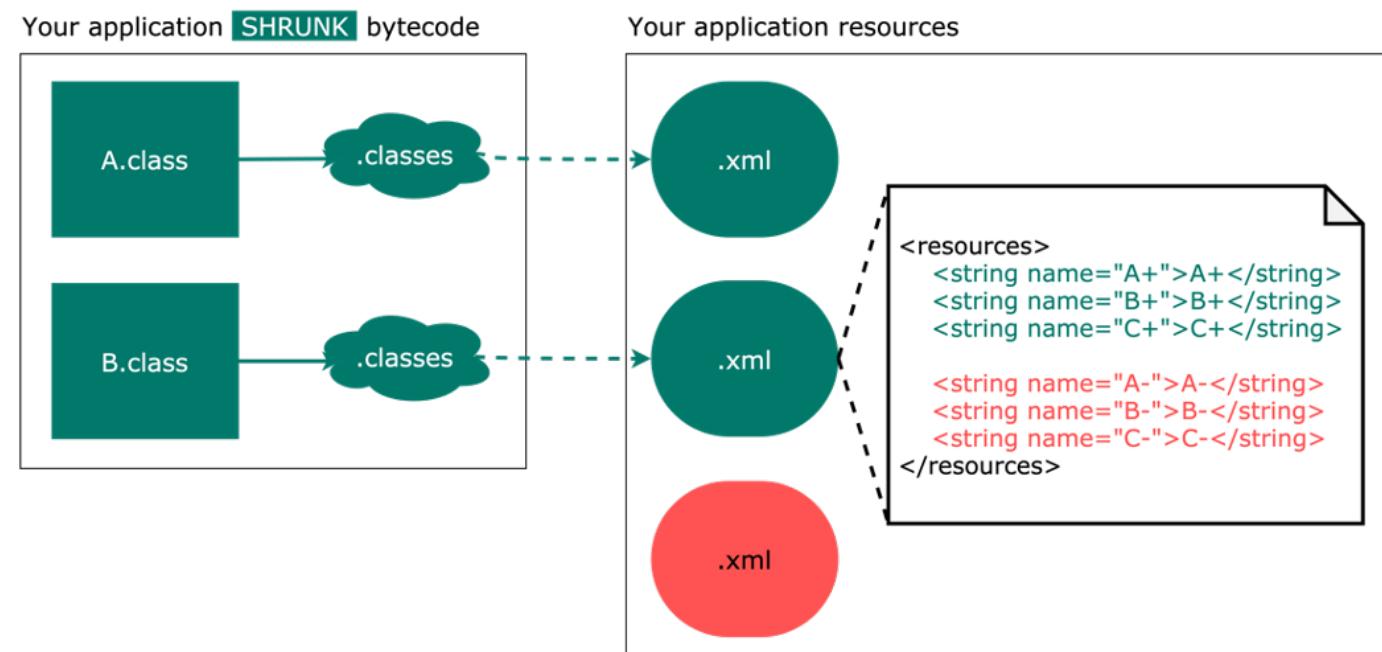


# ProGuard

1. Сканируется весь достичимый код на факт использования ресурсов

2. Используемые ресурсы помечаются как достичимые. Внутри одного файла могут быть как достичимые, так и недостичимые ресурсы.

3. Достичимый код сохраняется, остальное удаляется



# ProGuard

- Сохраняет ресурсы используемые напрямую или косвенно

```
String positiveText = context.getString(R.string.unsubscribe_2);
String negativeText = context.getString(R.string.close);
```

Прямое использование

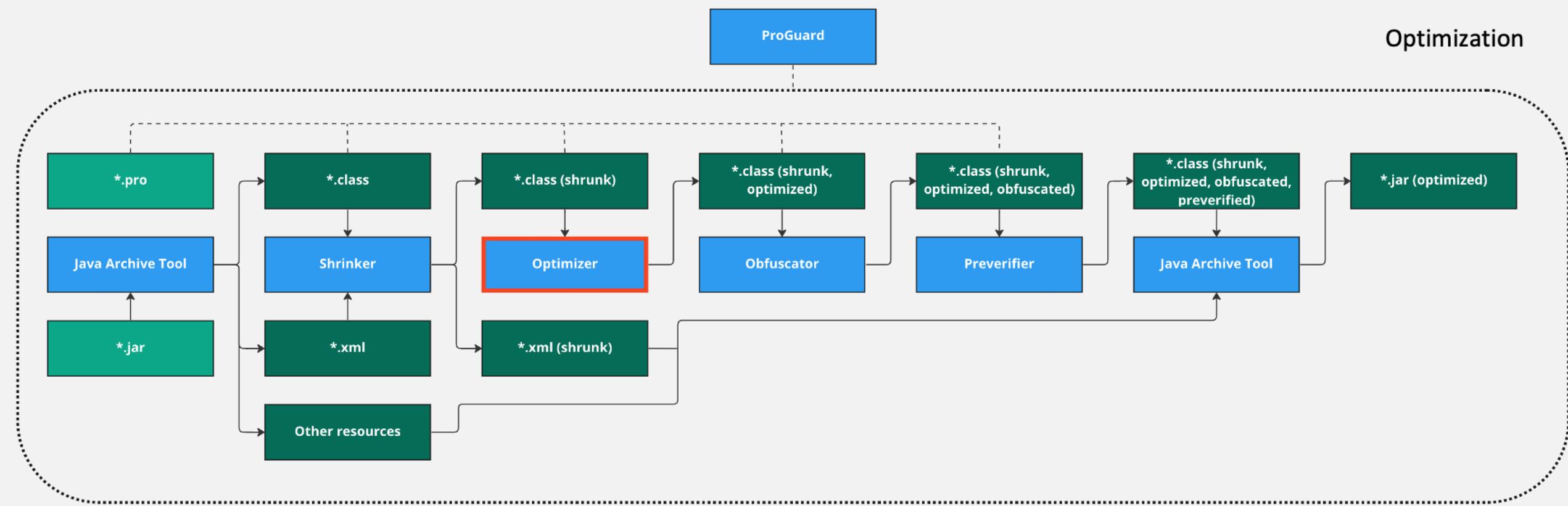
- Режимы работы Defense и Strict

```
.activity.getResources().getIdentifier(key, defType: "string", activity.getPackageName());
```

Косвенное использование

- Поддерживает пользовательские правила в keer.xml

# ProGuard



# ProGuard

```
class Test {  
    private static final String WILDCARD = "*.";  
    public static void main(String... args) {  
        String pattern = "*.example.com";  
        String host = pattern.startsWith(WILDCARD)  
            ? pattern.substring(WILDCARD.length())  
            : pattern;  
        String canonical = HttpUrl.get("http://" + host).host();  
        System.out.println(canonical);  
    }  
}
```

На деле идёт работа с байт-кодом!!

# ProGuard

1. Constant Propagation
2. Control flow analysis
3. Data-flow analysis
4. Partial evaluation
5. Branch Elimination
6. ... (*и другие*)

# ProGuard

```
class Test {  
    - private static final String WILDCARD = "*.";  
    public static void main(String... args) {  
        - String pattern = "*.example.com";  
        - ? pattern.substring(WILDCARD.length())  
        - : pattern;  
        + String host = "*.example.com".startsWith("*")  
        + ? "*.example.com".substring(2)  
        + : "*.example.com";  
        String canonical = HttpUrl.get("http://" + host).host();  
        System.out.println(canonical);  
    }  
}
```

[Constant Propagation](#)

# ProGuard

```
class Test {  
    public static void main(String... args) {  
        - String host = ".example.com".startsWith("*.")  
        + String host = true  
            ? ".example.com".substring(2)  
            : ".example.com";  
        String canonical = HttpUrl.get("http://" + host).host();  
        System.out.println(canonical);  
    }  
}
```

# ProGuard

```
class Test {  
    public static void main(String... args) {  
        - String host = true  
        -     ? ".example.com".substring(2)  
        -     : ".example.com";  
        + String host = "example.com"  
        String canonical = HttpUrl.get("http://" + host).host();  
        System.out.println(canonical);  
    }  
}
```

# ProGuard

```
class Test {  
    public static void main(String... args) {  
        - String host = "example.com"  
        - String canonical = HttpUrl.get("http://" + host).host();  
        + String canonical = HttpUrl.get("http://example.com").host();  
        System.out.println(canonical);  
    }  
}
```

# ProGuard

```
class Test {  
    public static void main(String... args) {  
        - String canonical = HttpUrl.get("http://example.com").host();  
        - System.out.println(canonical);  
        + System.out.println("example.com");  
    }  
}
```

# ProGuard

```
class Test {  
    - private static final String WILDCARD = "*.";  
    public static void main(String... args) {  
        - String pattern = "*.example.com";  
        - String host = pattern.startsWith(WILDCARD)  
        - ? pattern.substring(WILDCARD.length())  
        - : pattern;  
        - String canonical = HttpUrl.get("http://" + host).host();  
        - System.out.println(canonical);  
        + System.out.println("example.com");  
    }  
}
```

# ProGuard

## -optimizations

1. class/marking/final
2. class/unboxing/enum
3. library/gson
4. code/simplification/arithmetic
5. ... (и [другие](#))

```
-optimizations
    !code/simplification/arithmetic,
    !code/simplification/cast,
    !field/*,
    !class/merging/*
```

! – исключение

\* - все оптимизации

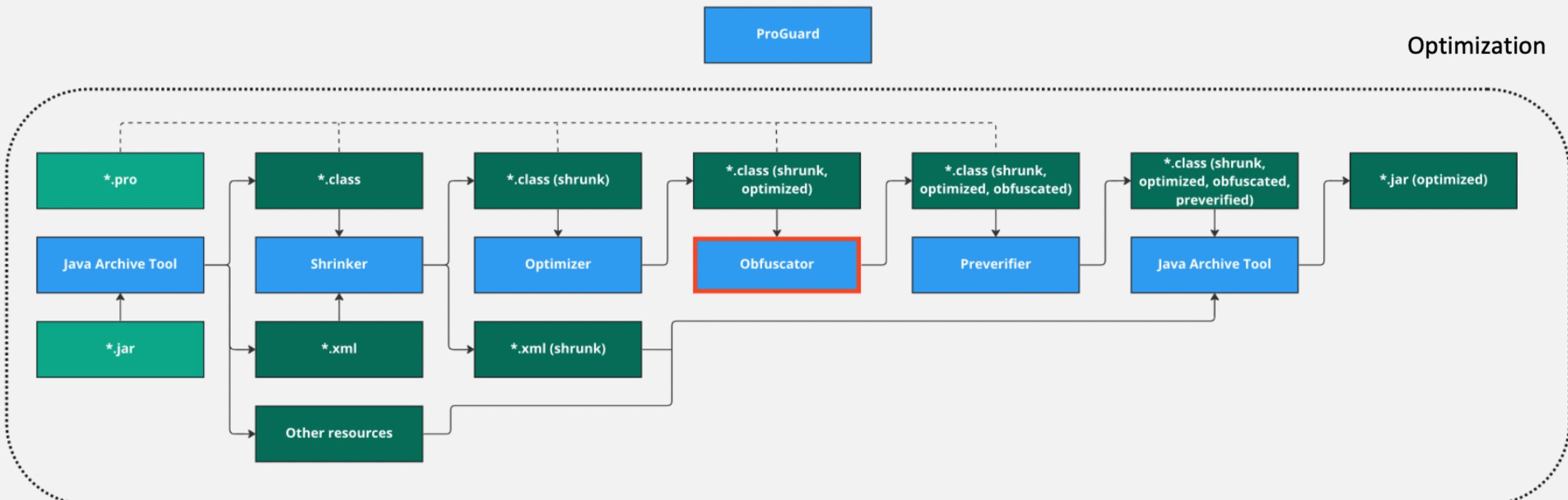
# ProGuard

## -optimizationpasses N

«Specifies the number of optimization passes to be performed. By default, a single pass is performed. Multiple passes may result in further improvements. If no improvements are found after an optimization pass, the optimization is ended. Only applicable when optimizing.»

- По умолчанию 1 проход
- Чем больше проходов, тем больше оптимизаций

# ProGuard



# ProGuard

```
public class ProGuardOptimizerTest {  
  
    private static final String WILDCARD = "*.";  
  
    private static void test() {  
        String pattern = "*.ok.ru";  
        String host = pattern.startsWith(WILDCARD)  
            ? pattern.substring(WILDCARD.length())  
            : pattern;  
        String canonical = HttpUrl.get("https://" + host).host();  
        Logger.d(canonical);  
    }  
}
```

Всё понятно и легко разобраться

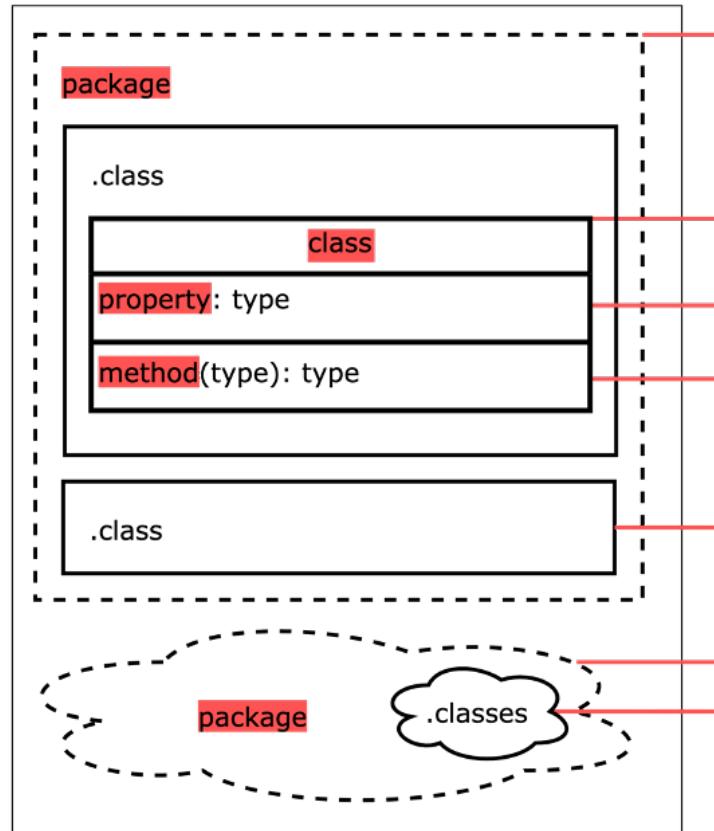


```
public class ProGuardOptimizerTest {  
    private static final String WILDCARD = "*.";  
    private static void test() {  
        String pattern = "*ok.ru";  
        String host = pattern.startsWith(WILDCARD)  
            ? pattern.substring(WILDCARD.length())  
            : pattern;  
        String canonical = HttpUrl.get("http://X" + host).host();  
        Logger.d(canonical);  
    }  
}
```

Ничего не понятно, но работает точно так же

# ProGuard

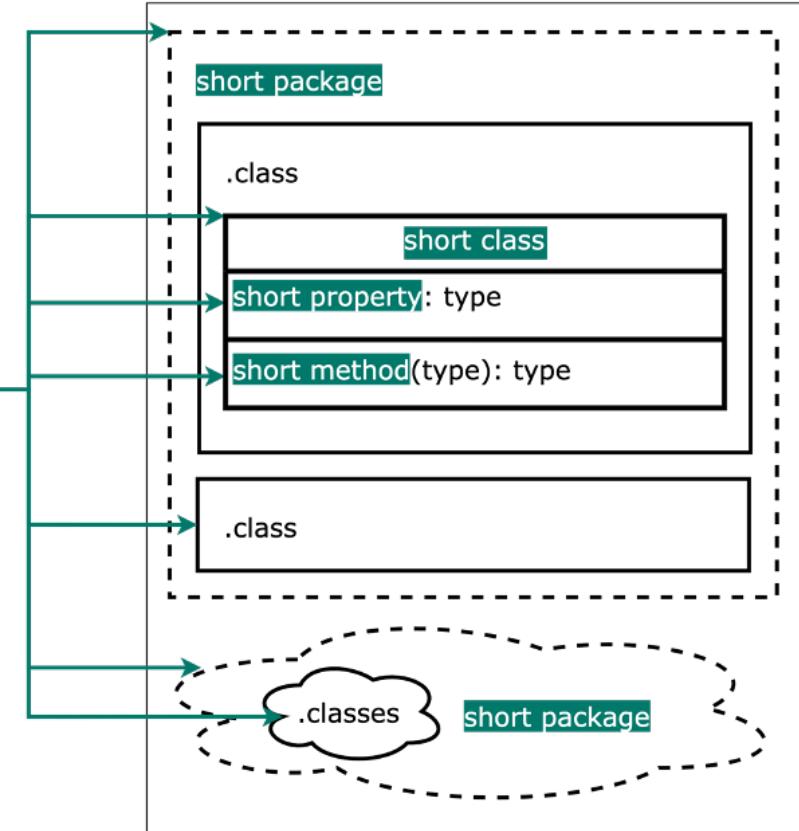
Your application **SHRUNK and OPTIMIZED** bytecode



Obfuscation map

```
package -> short package
class -> short class
property -> short property
method -> short method
...
```

Your application **OBFUSCATED** bytecode



# ProGuard

```
2022-08-11 16:49:32.009 10223-10316/? E/AndroidRuntime: FATAL EXCEPTION: Bus-2
Process: ru.ok.android, PID: 10223
java.lang.IllegalAccessError: Illegal class access ('v10.g' attempting to access 'bc0.a') in attempt to invoke virtual method void bc0.a.write(java.lang.String) (declaration of 'v10.g' appears in base.apk)
    at v10.g.q(SourceFile:13)
    at v10.g.d2(SourceFile:9)
    at ru.ok.android.api.away.AwayApiParam.c(SourceFile:2)
    at v10.g.m(SourceFile:8)
    at ru.ok.android.api.http.HttpApiUriEngine.f(SourceFile:29)
    at ru.ok.android.api.http.HttpApiUriEngine.c(SourceFile:6)
    at v10.b.a(SourceFile:14)
    at wk1.a.a(SourceFile:2)
    at xk1.a.a(SourceFile:1)
    at ru.ok.android.services.transport.OkApiSessionCallback.c(SourceFile:1)
    at ru.ok.android.services.transport.OkApiSessionCallback.a(SourceFile:5)
    at t10.f.a(SourceFile:12)
    at t10.c.b(Unknown Source:13)
    at xk1.h.d(SourceFile:2)
```

Project update recommended  
Android Gradle Plugin can be [upgraded](#).

Problems App inspection TODO Profiler Build Terminal Git Event Log Run Logcat Layout Inspector  
1:1 LF UTF-8 4 spaces ANDROID-28722-kotlin-1.7  
doklassniki-android-release.apk installed (8 minutes ago)

# ProGuard

“The purpose of obfuscation is to reduce your app size by shortening the names of your app’s classes, methods, and fields”

# ProGuard

```
package com.example.proguard.obfuscation;

public class PinCodeBuilder {
    private int pinCodeLength;
    private StringBuilder pinCodeStringBuilder = new StringBuilder(pinCodeLength);

    public PinCodeBuilder(int pinCodeLength) {
        this.pinCodeLength = pinCodeLength;
    }

    public void increment(int pinCodeDigit) {
        if (pinCodeStringBuilder.length() < pinCodeLength) {
            pinCodeStringBuilder.append(pinCodeDigit);
        }
    }
}
```

# ProGuard

```
package a.a.a.a;

public class a {
    private int a;
    private StringBuilder b = new StringBuilder(pinCodeLength);

    public a(int pinCodeLength) {
        this.a = pinCodeLength;
    }

    public void c(int pinCodeDigit) {
        if (b.length() < a) {
            b.append(pinCodeDigit);
        }
    }
}
```

# ProGuard

“...obfuscate code to conceal its purpose (security through obscurity) or its logic or implicit values embedded in it, primarily, in order to prevent tampering, deter reverse engineering...”

# ProGuard

- GuardSquare, Google: обfuscation - сокращение имён.
- Wikipedia: обfuscation - запутывание смысла кода

# ProGuard

```
buildTypes {  
    release {  
        minifyEnabled true  
    }  
}
```

# ProGuard

```
2022-08-11 16:49:32.009 10223-10316/? E/AndroidRuntime: FATAL EXCEPTION: Bus-2
Process: ru.ok.android, PID: 10223
java.lang.IllegalAccessError: Illegal class access ('v10.g' attempting to access 'bc0.a') in attempt to invoke virtual method void bc0.a.write(java.lang.String) (declaration of 'v10.g' appears in base.apk!class
at v10.g.q(SourceFile:13)
at v10.g.d2(SourceFile:9)
at ru.ok.android.api.away.AwayApiParam.c(SourceFile:2)
at v10.g.m(SourceFile:8)
at ru.ok.android.api.http.HttpApiUriEngine.f(SourceFile:29)
at ru.ok.android.api.http.HttpApiUriEngine.c(SourceFile:6)
at v10.b.a(SourceFile:14)
at wk1.a.a(SourceFile:2)
at xk1.a.a(SourceFile:1)
at ru.ok.android.services.transport.OkApiSessionCallback.c(SourceFile:1)
at ru.ok.android.services.transport.OkApiSessionCallback.a(SourceFile:5)
at t10.f.a(SourceFile:12)
at t10.c.b(Unknown Source:13)
at xk1 h d(SourceFile:2)
```

Project update recommended  
Android Gradle Plugin can be upgraded.

Emulator Device File Explorer

Problems App Inspection TODO Profiler Build Terminal Git Event Log Run Logcat Layout Inspector

dnoklassniki-android-release.apk installed (8 minutes ago)

1:1 LF UTF-8 4 spaces ANDROID-28722-kotlin-1.7

# ProGuard

```
package a.a.a.a;

public class a {
    private int a;
    private StringBuilder b = new StringBuilder(pinCodeLength);

    public a(int pinCodeLength) {
        this.a = pinCodeLength;
    }

    public void c(int pinCodeDigit) {
        if (b.length() < a) {
            b.append(pinCodeDigit);
        }
    }
}
```

# ProGuard

Техники обfuscации кода:

- Запутывание названий
- Шифрование литералов
- Обfuscация control flow
- Преобразования языковых конструкций
- Вставка фиктивного кода
- Удаление метаданных
- Слияние кода
- Anti-tampering
- ... (и [другие](#))

# ProGuard

Некоторые техники обfuscации могут увеличить размер байт-кода и/или понизить скорость работы

# ProGuard

Возможные настройки [ProGuard](#), [R8](#)

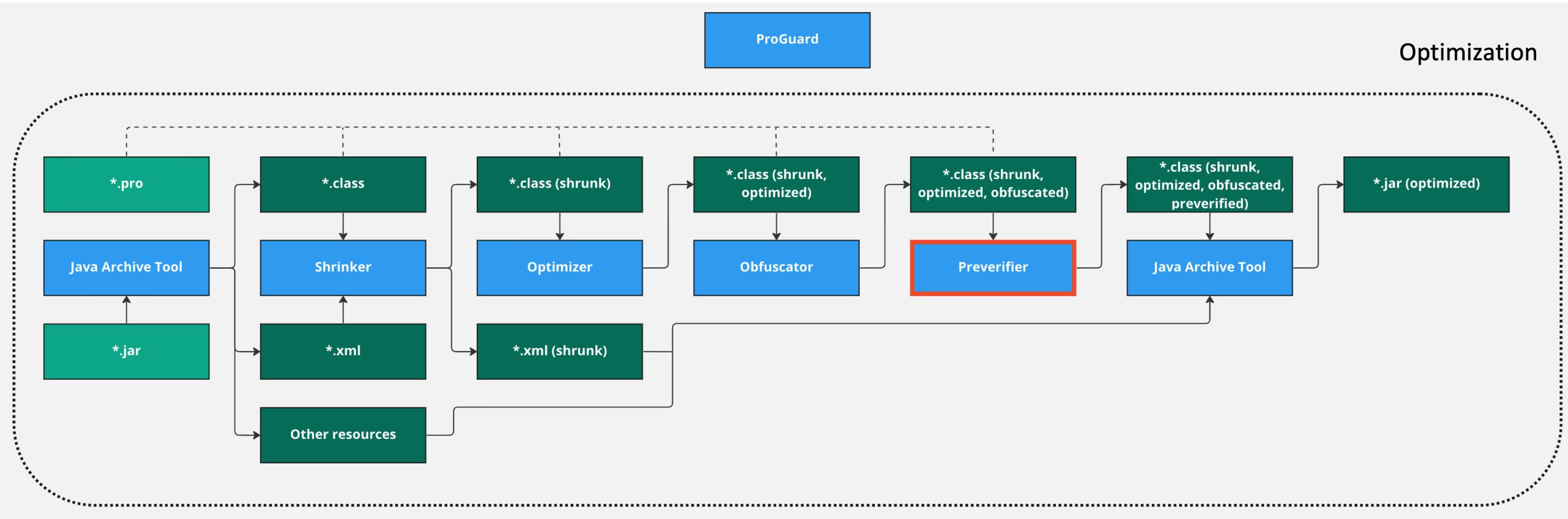
- -flattenpackagehierarchy [...] (всё в один пакет)
- -overloadaggressively (максимально переиспользовать имена)
- -adaptclassstrings [...] (обfuscation имён строковых констант)
- ...

# ProGuard

Обфускаторы Android приложений

- [Obfuscate](#)
- [AAMO](#)
- [Paranoid](#)
- [DexGuard\\*](#)

# ProGuard



# ProGuard

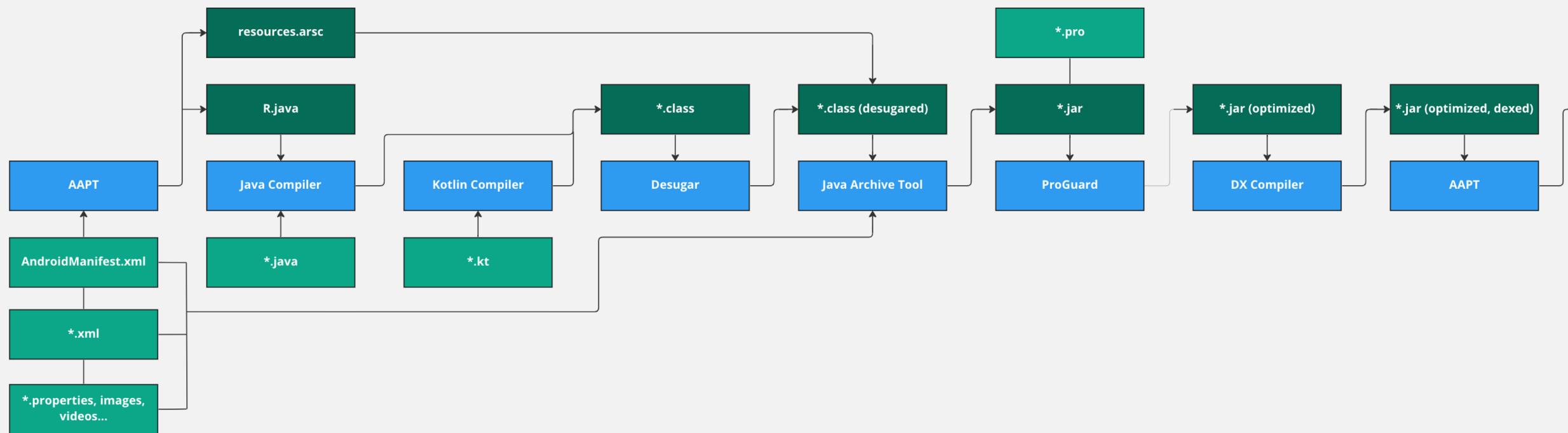
- **-android** укажет на необходимость произвести проверку на соответствие target API и совместимости
- Для .dex компилятора и Dalvik VM не поддерживается верификация, поэтому можно использовать **-dontpreverify** для сокращения времени оптимизации

# R8, D8

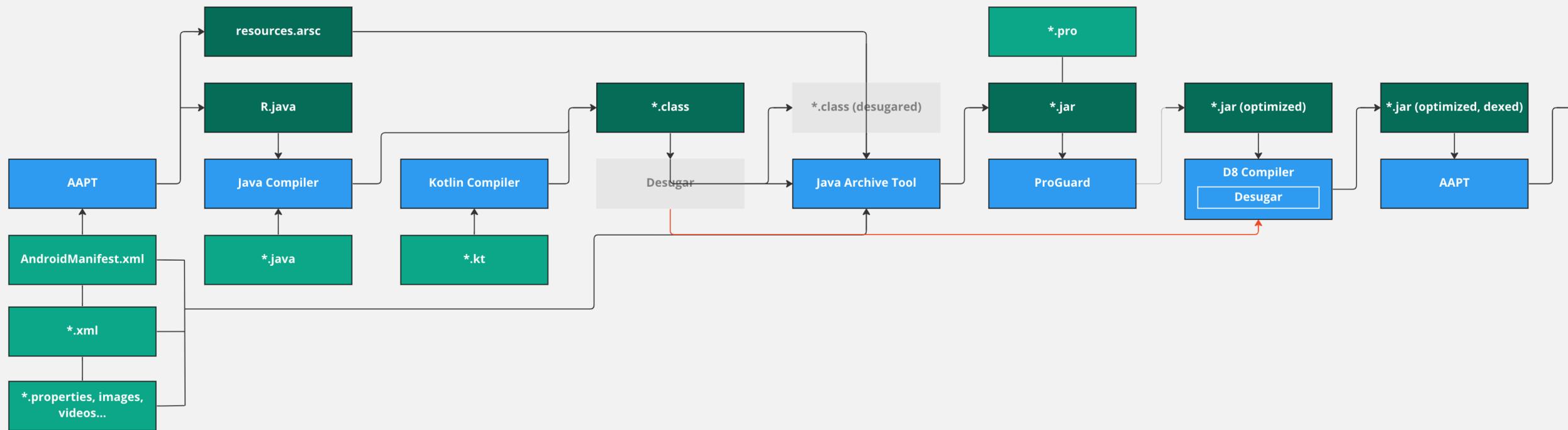
D8 is the next-generation dex compiler

- Used by default since AGP 3.2

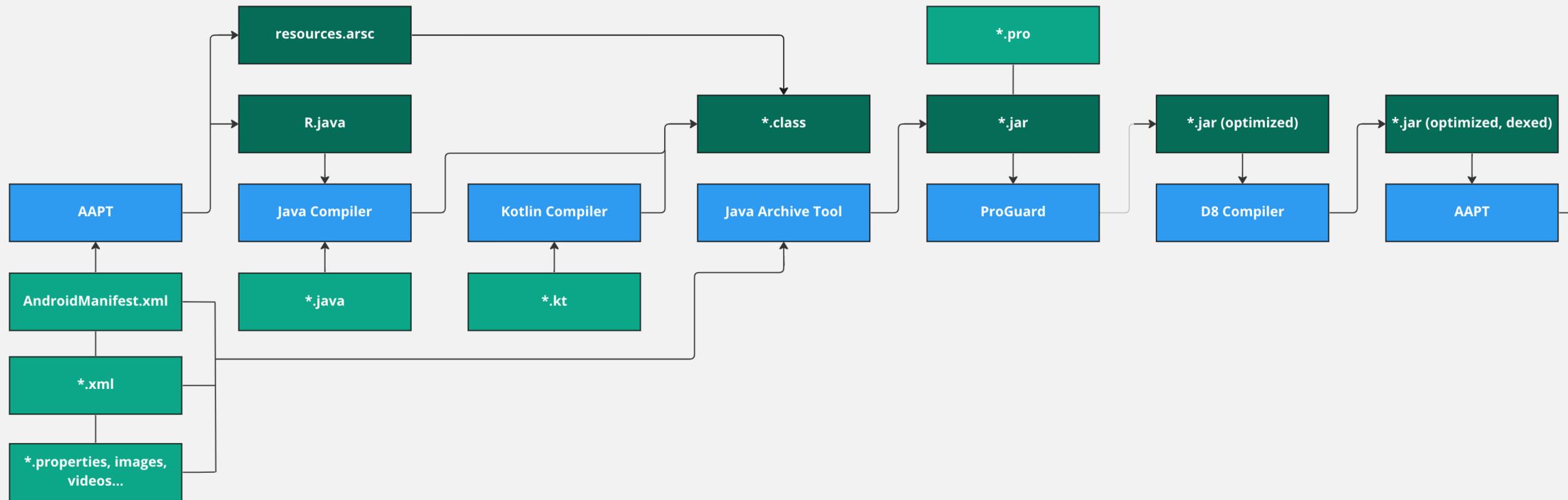
## R8, D8



# R8, D8

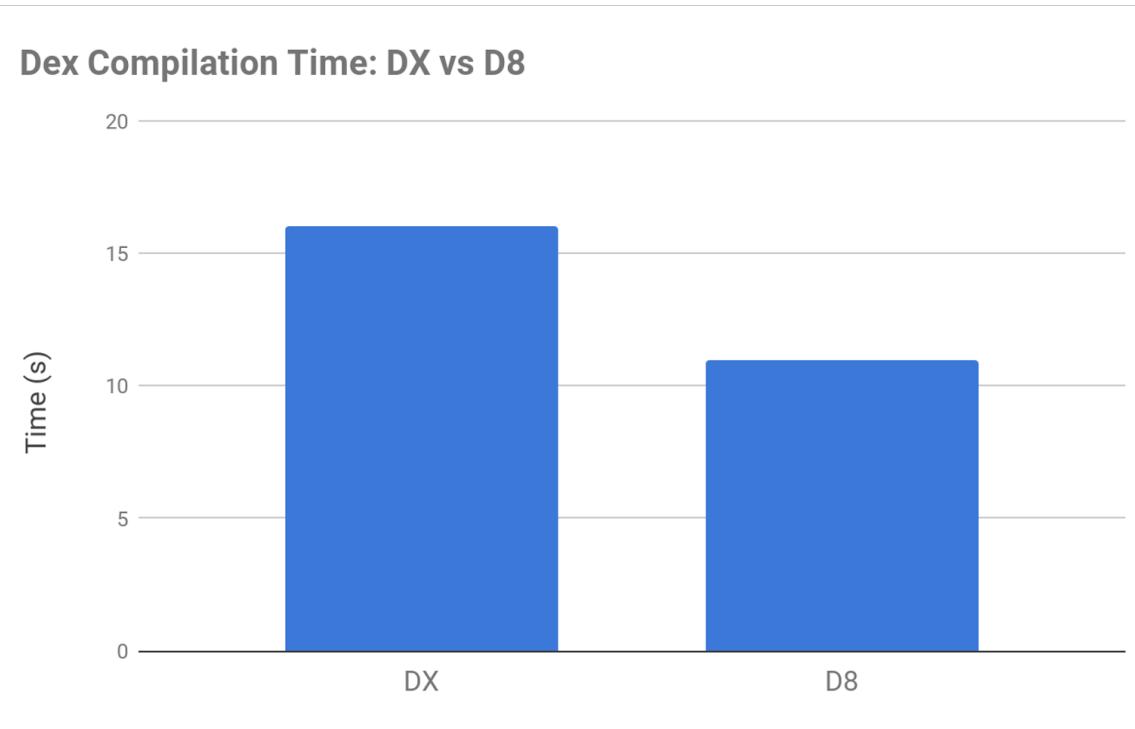


# R8, D8



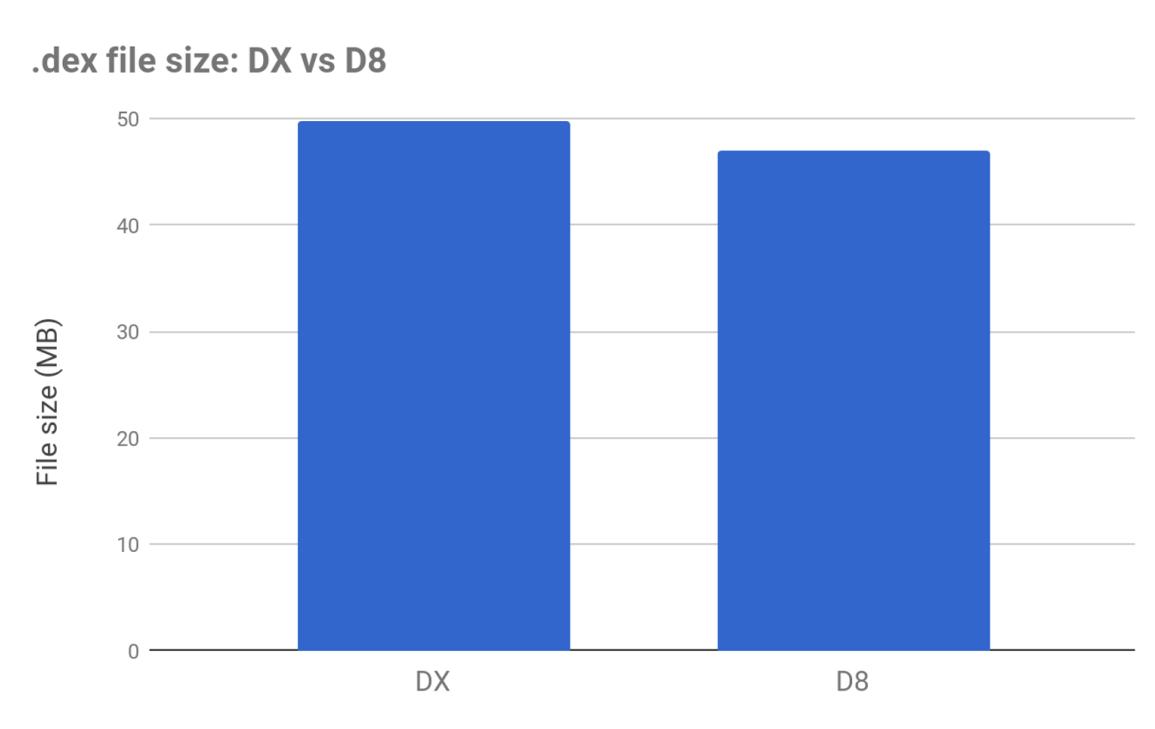
# R8, D8

Dex Compilation Time: DX vs D8



- D8 быстрее компилирует

.dex file size: DX vs D8



- D8 сильнее уменьшает размер .dex

# R8, D8

- DX и D8 имеют небольшие отличия в генерации Dalvik байт-кода

# R8, D8

R8 is the next-generation app optimizer

- Used by default since AGP 3.4

# R8, D8

```
minifyEnabled true
shrinkResources true
proguardFile 'proguard-rules.txt'
if (instrument) {
    proguardFile 'proguard-instrument-rules.txt'
}
```

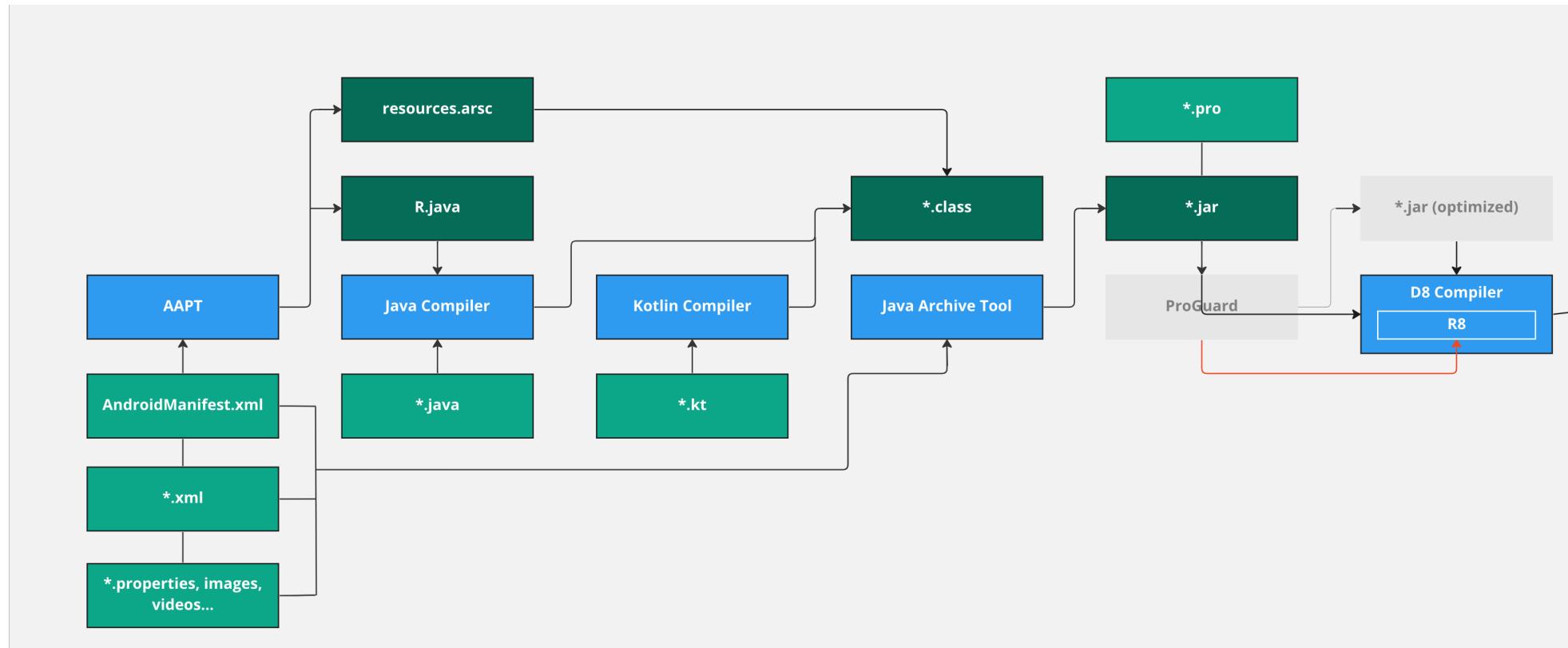
ProGuard и R8 – разные технологии

# R8, D8

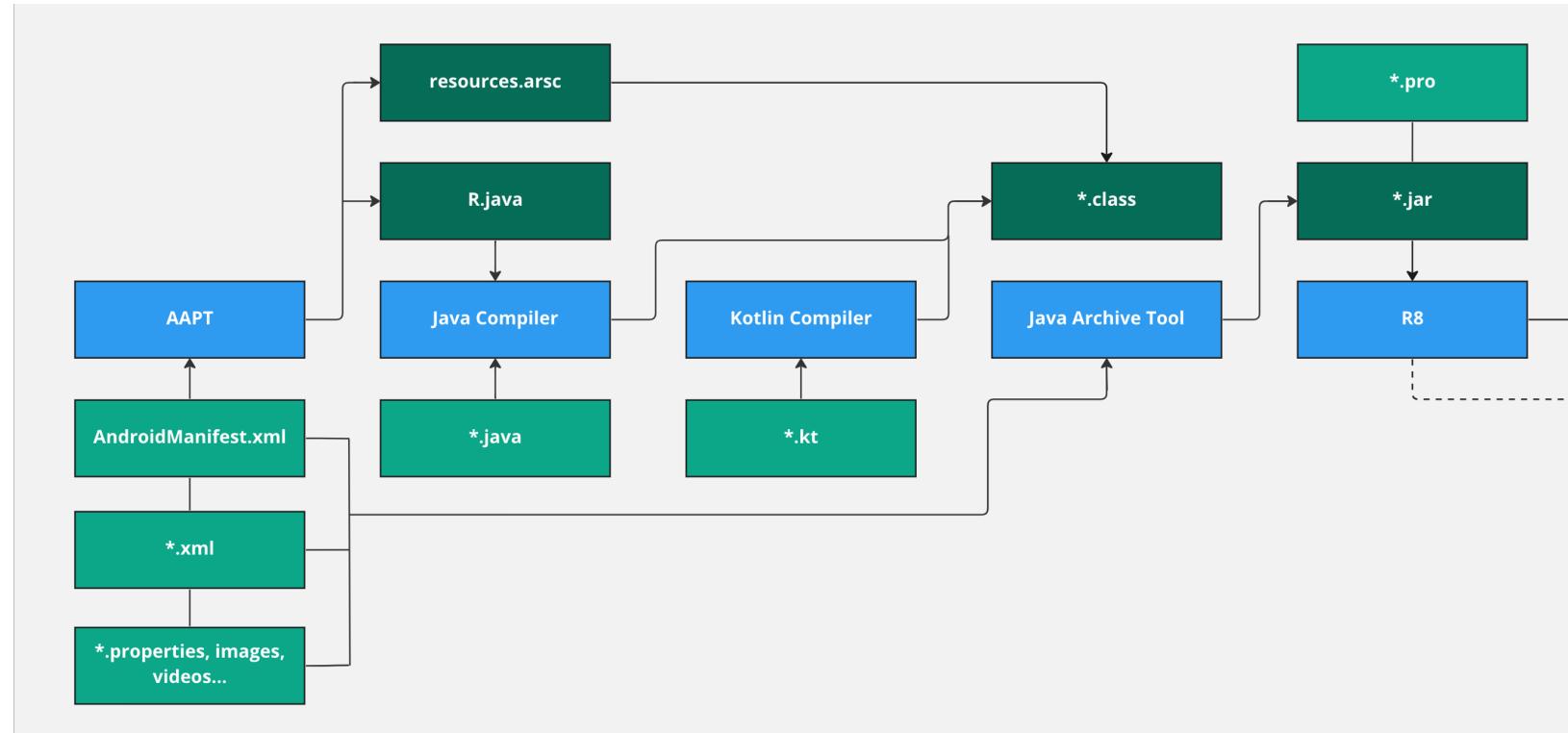
odnoklassniki-android/build/outputs/mapping/release/

- configuration.txt
- mapping.txt
- seeds.txt
- usage.txt
- resources.txt

# R8, D8

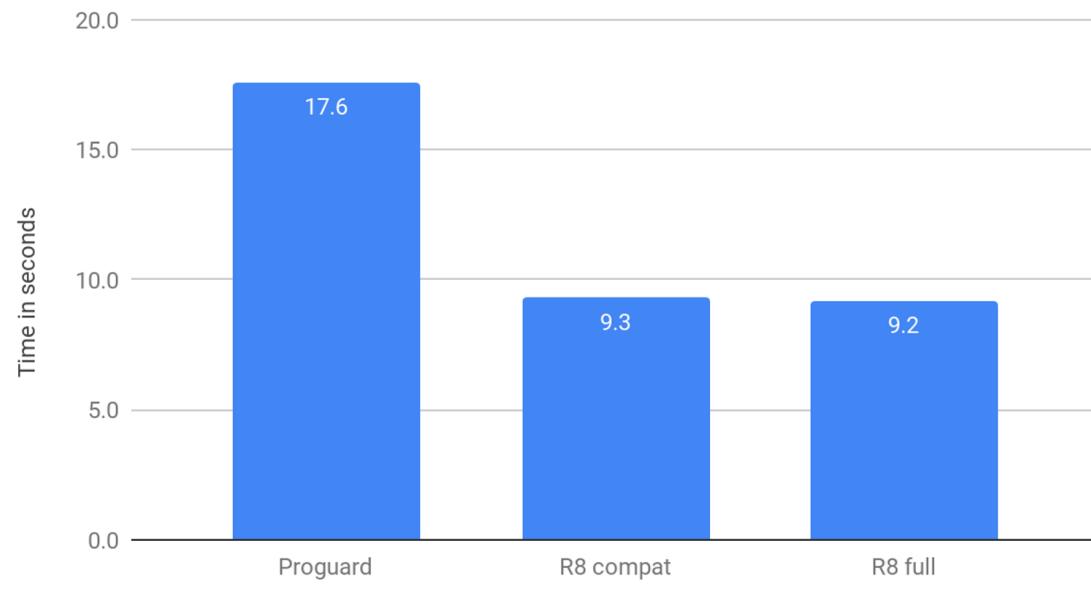


# R8, D8



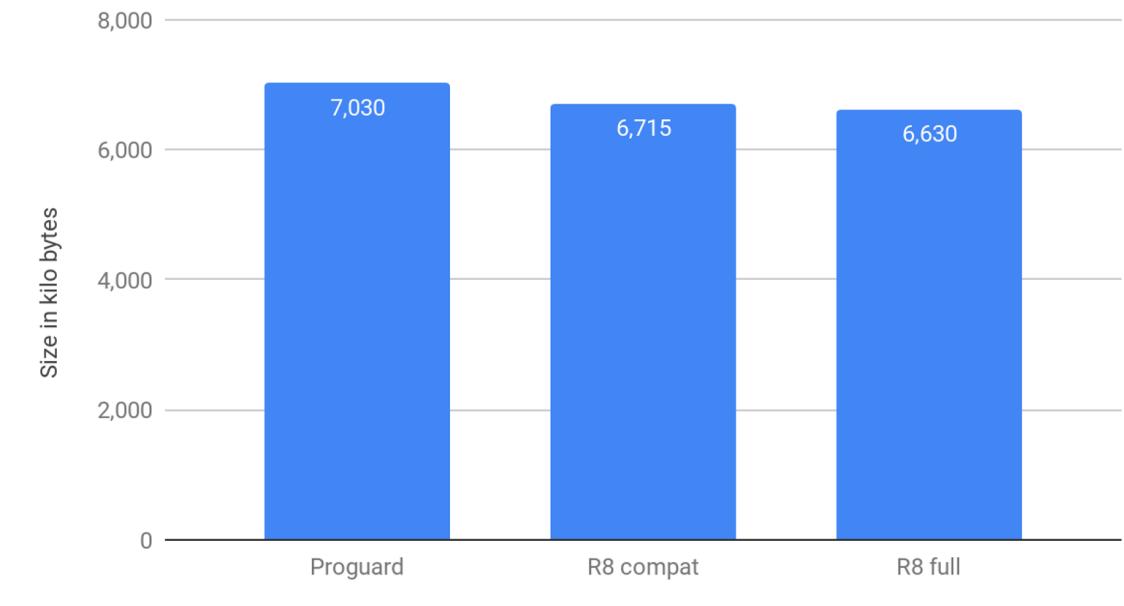
# R8, D8

Shrinking + Dexing time



- R8 быстрее компилирует .dex

Dex file size



- R8 генерирует .dex меньшего размера

# R8, D8

- In full mode, R8 performs more aggressive optimizations

`android.enableR8.fullMode=true`

# R8, D8

- Задача, сторонние сервисы, мобильное приложение — вот компоненты, от которых зависит итоговый размер приложения, скорость его работы и безопасность его использования.
- Процесс сборки Android приложений состоит из основных этапов: java / kotlin компиляция, dex компиляция. Между основными этапами определены строгие контракты, поддерживая которые можно встроить дополнительные этапы.
- Оптимизация сборки в общем случае состоит из этапов: сокращение кода, ресурсов (память); оптимизация кода (скорость); обfuscация (безопасность).

# R8, D8

- Обfuscation — приведение кода к виду, сохраняющему его функциональность, но затрудняющему анализ. Запутывание имен — одна из многих техник обfuscации, часто используемая в популярных оптимизаторах.
- ProGuard — java байт-код оптимизатор, способный оптимизировать сборки Android приложений. Разработан GuardSquare.
- D8, R8 — технологии, разработанные Google, призванные улучшить процесс сборки (D8) и ее оптимизации (R8).

# Спасибо!

