Оптимизация сборок Android приложений: ProGuard, D8, R8. Тайны обфускации

О себе

- Тимлид Android-команды суперсервиса «Мои субсидии»
- Разрабатывал «Самокатус»
- До Android занимался backend-разработкой на Java, Python.
- Автор Text-To-Speech-системы для антропоморфного робота (DOI: 10.1109/DeSE.2019.00055).

О компании

- 5 лет занимаемся цифровой трансформацией крупного бизнеса.
- Разрабатываем цифровые сервисы
- Занимаемся технологическим консалтингом
- Клиенты: Альфа Банк, КАМАЗ, СИБУР, Gett, МТС, QIWI
- https://technokratos.com/

ТЕХНОКРАТИЯ

1. Введение

- В среднем на смартфоне установлено 40 приложений
- Каждые 6 МБ размера сборки сокращают количество установок на 1%

1. Введение

- Около 3 часов в день житель США проводит в мобильных приложениях
- Около 20% пользователей мгновенно переключаются на приложение-аналог при медленной работе

1. Введение

- В 2019 году 43% организаций пожертвовали мобильной безопасностью
- Менее 50% из 3000 ведущих мировых приложений для финансовых услуг на рынке имеют надлежащий уровень защиты

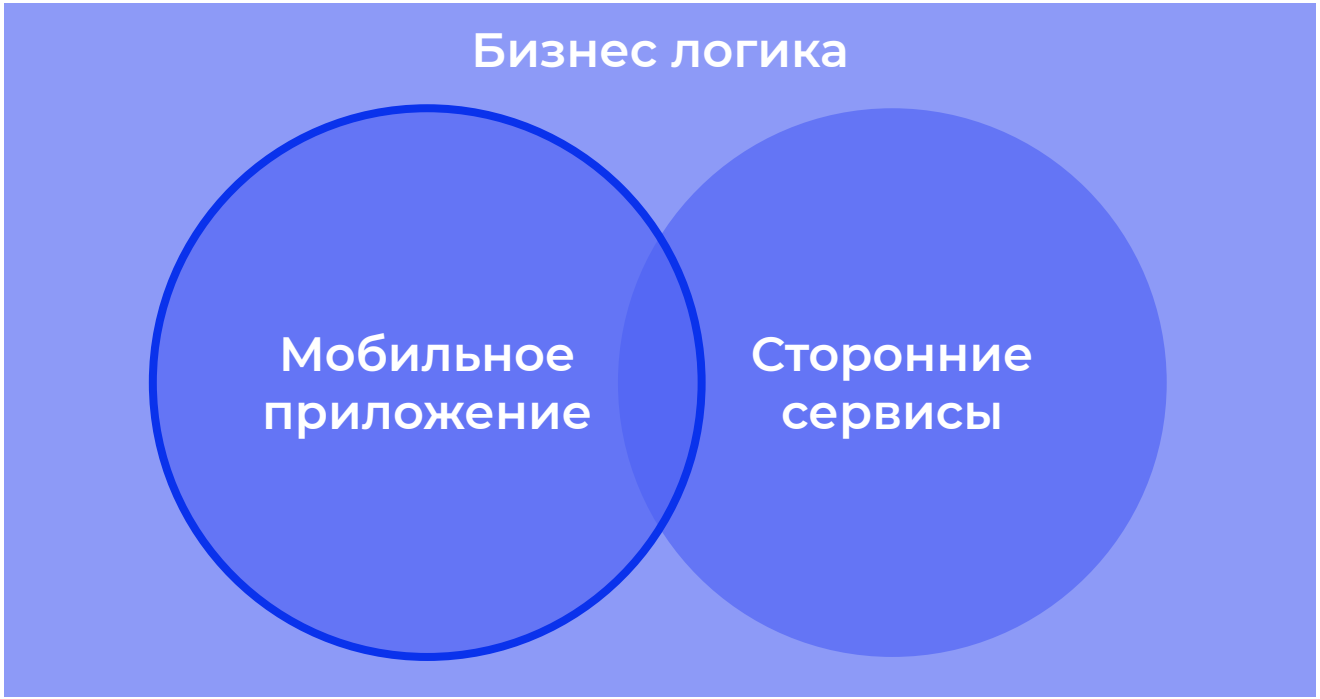
1. Введение

Как пользователь / разработчик я хочу приложение:

- Размером 0 бит
- Быстрее света
- Безопаснее Зоны 51

2. Оптимизация сборки

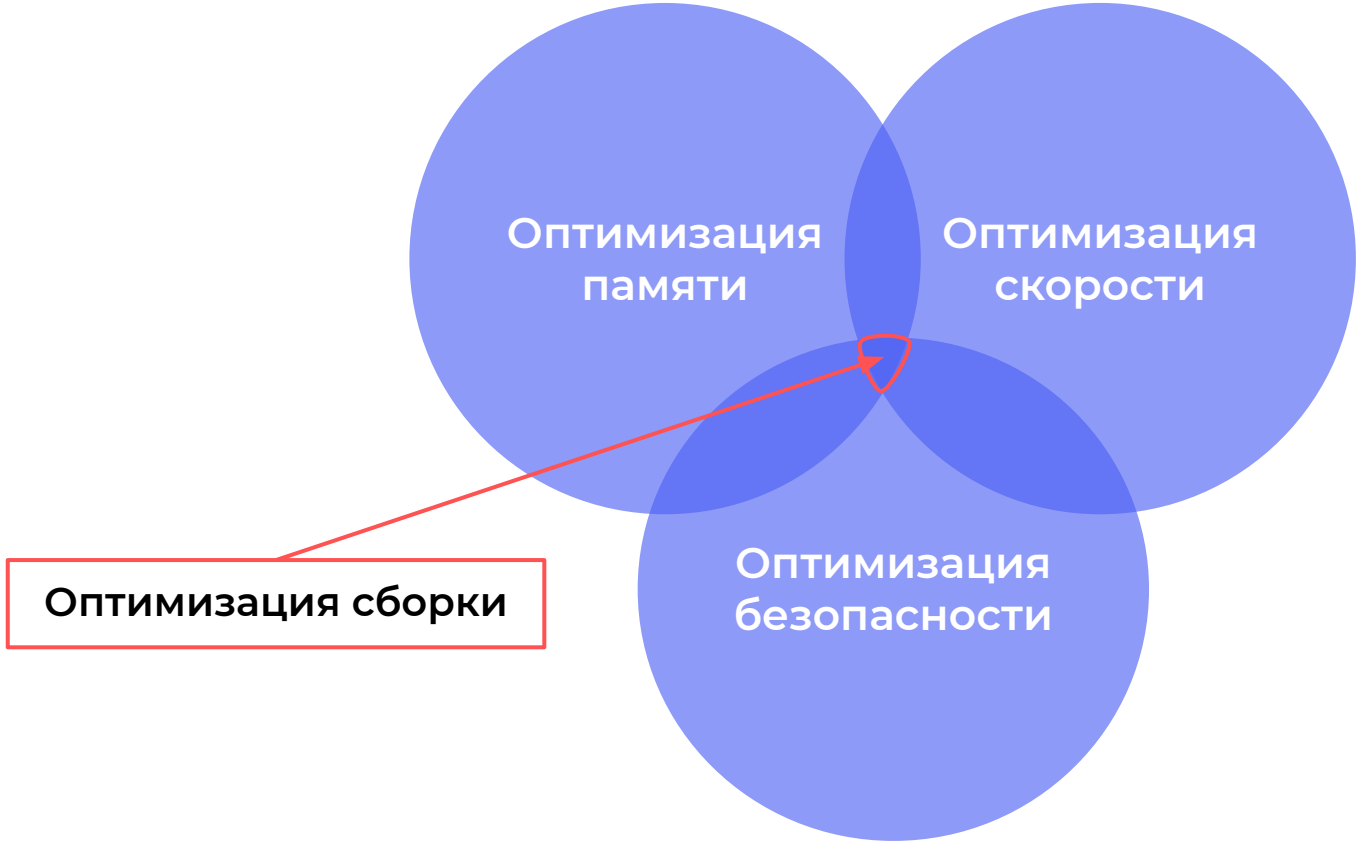
2. Оптимизация сборки



2. Оптимизация сборки

- Уменьшение размера приложения
  - Хранение данных вне
  - App bundle
- Увеличение скорости работы
  - Многопоточность
  - АиСД
- Повышение безопасности
  - Защищенные каналы связи
  - Пароли, ПИН коды

2. Оптимизация сборки



2. Оптимизация сборки

Оптимизация мобильного приложения

- Шифрование
- Dynamic Delivery
- Рендеринг
- ...
- Логирование
- **Оптимизация сборки**

2. Оптимизация сборки

Оптимизировать сборку:

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

Reference

3. ProGuard

«ProGuard is a Java class file shrinker, optimizer, obfuscator, and preverifier».

3. ProGuard

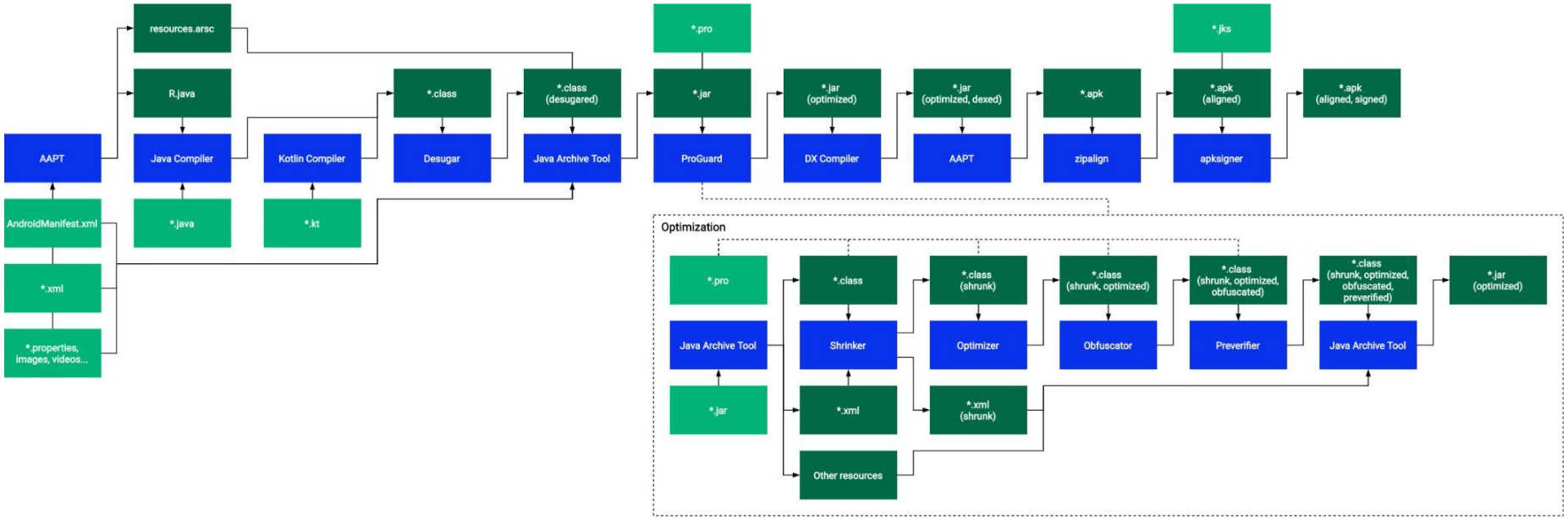
Reference

Guardsquare занимается разработкой программ для защиты мобильных приложений.

- iXGuard - защиты iOS приложений и SDK
- ProGuard - оптимизация Java и Android приложений
- DexGuard - улучшенная версия ProGuard
- ThreatCast - мониторинг безопасности iOS и Android приложений

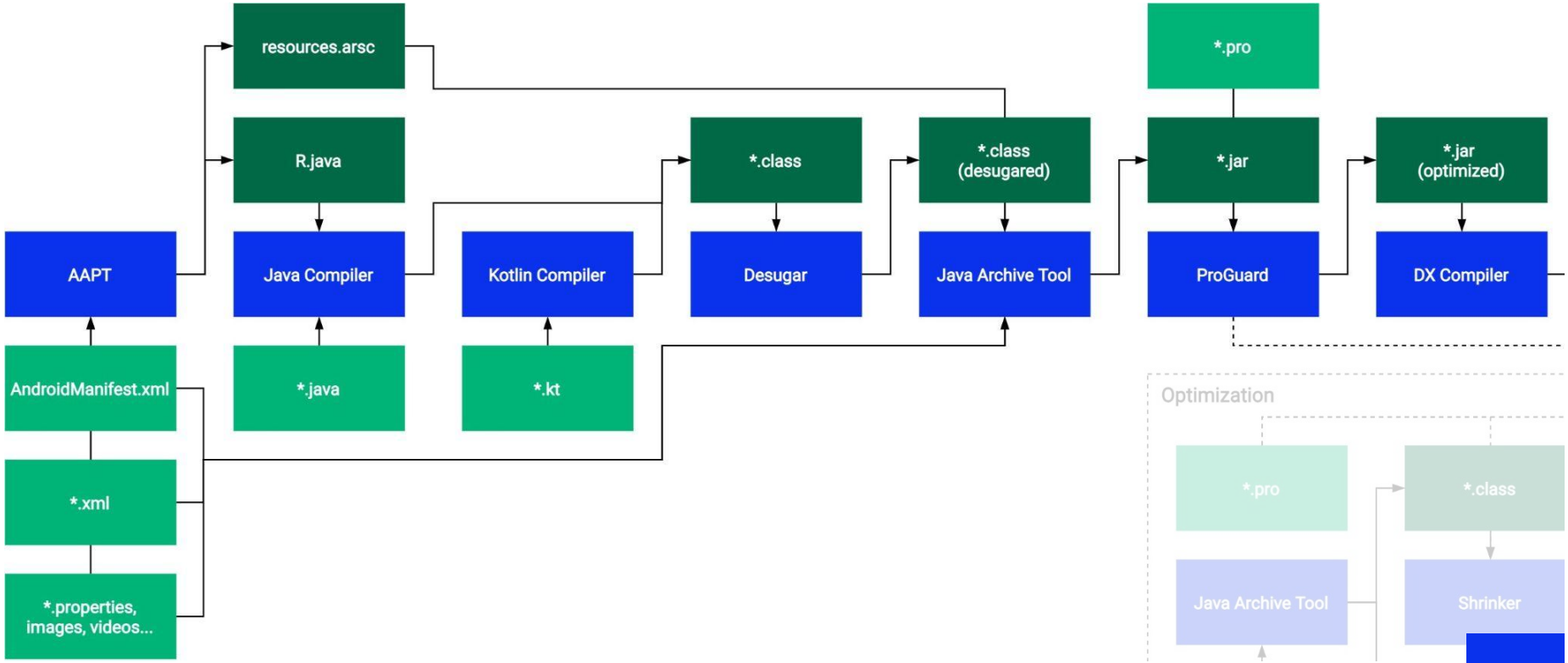
3. ProGuard

Reference

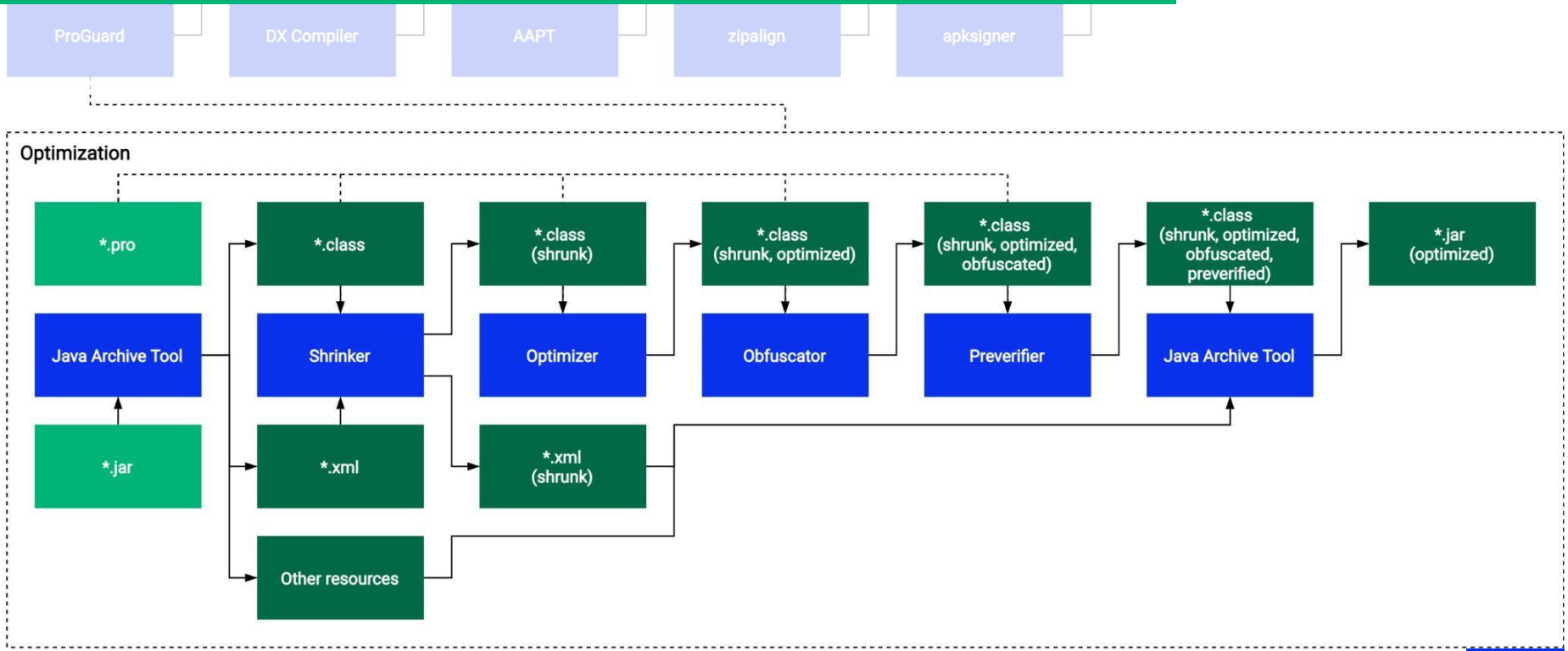


3. ProGuard

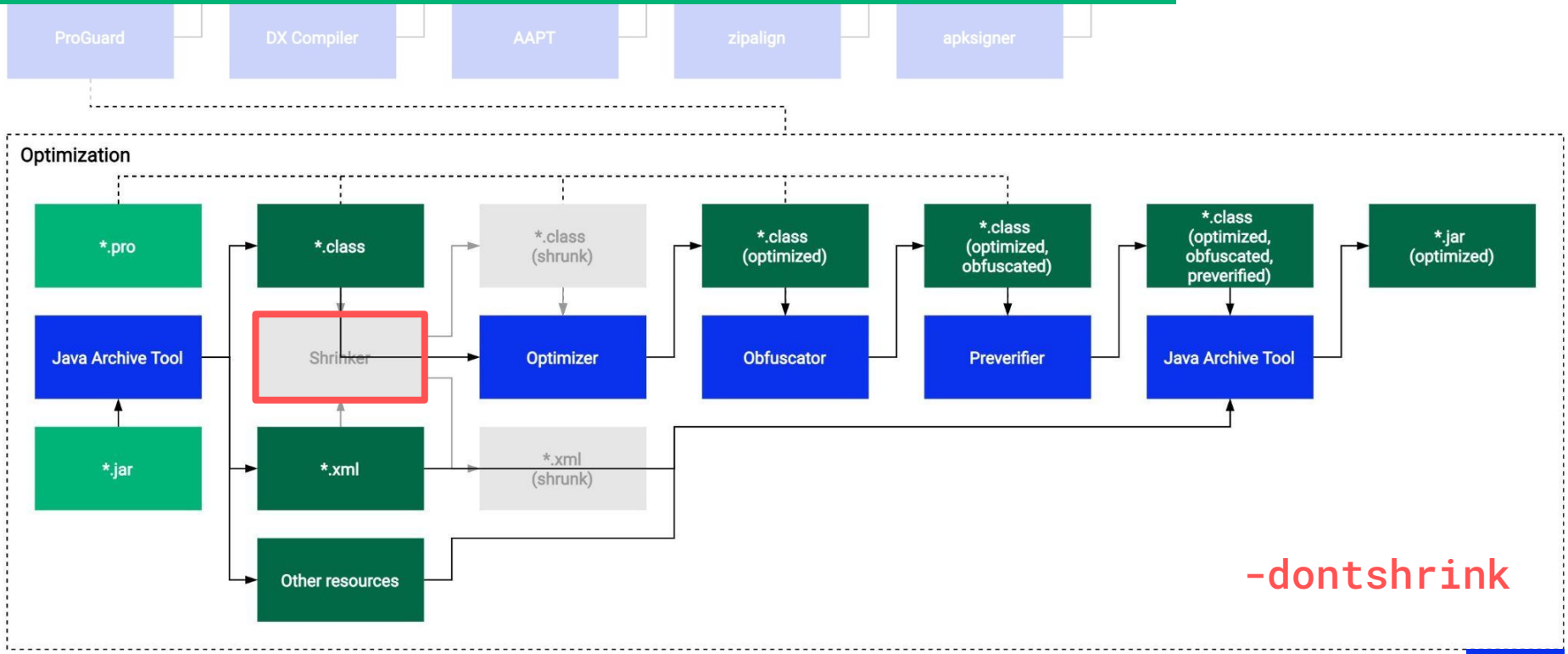
Reference



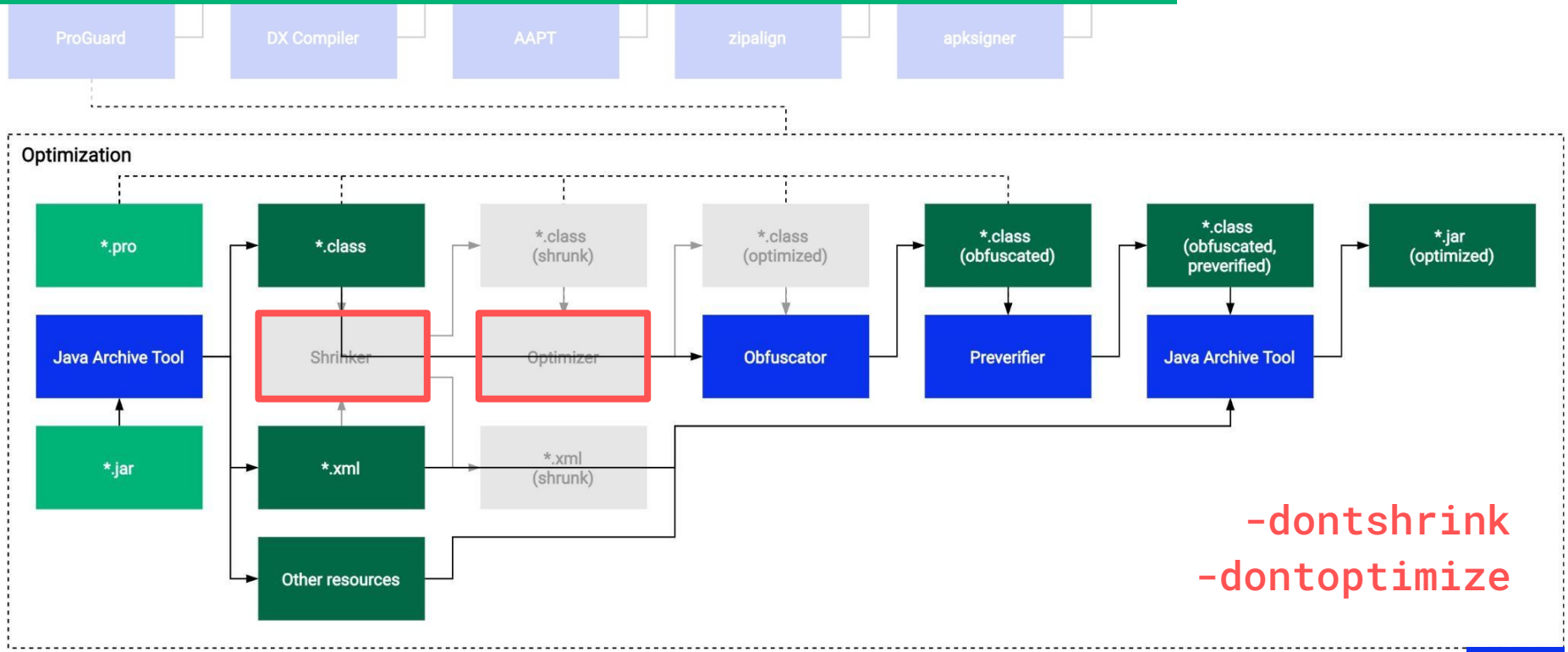
3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (shrunk, optimized)
*.class (shrunk, optimized, obfuscated)
*.class (shrunk, optimized, obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources

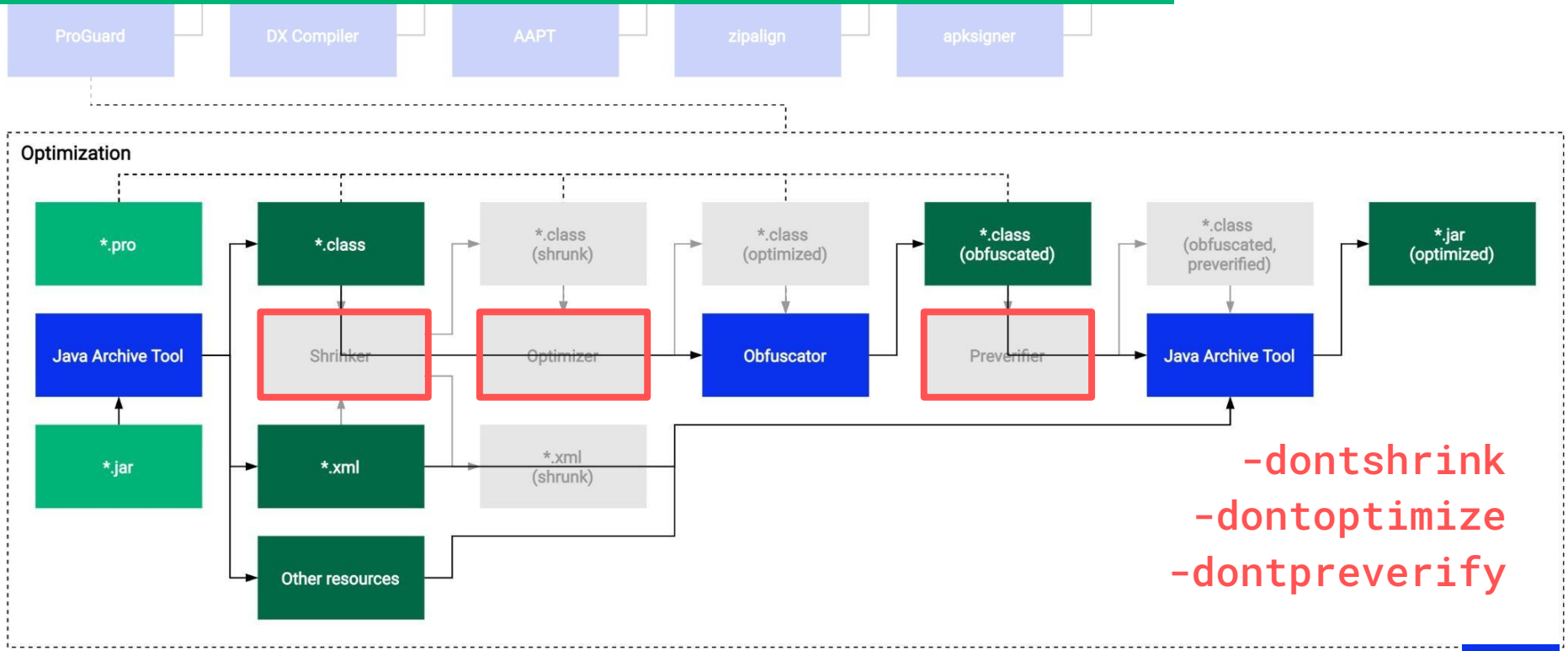
3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (optimized)
*.class (optimized, obfuscated)
*.class (optimized, obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources
-dontshrink

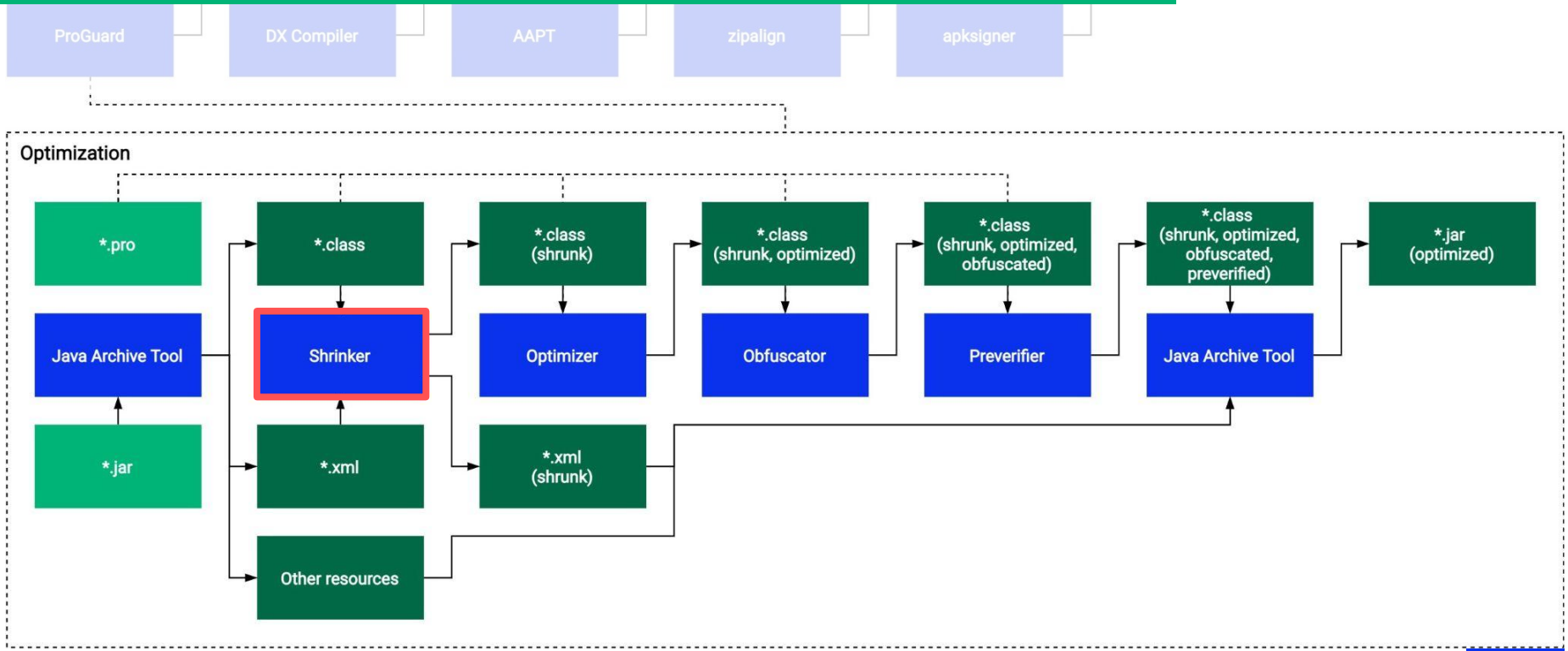
3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (optimized)
*.class (obfuscated)
*.class (obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources
-dontshrink
-dontoptimize

3. ProGuard

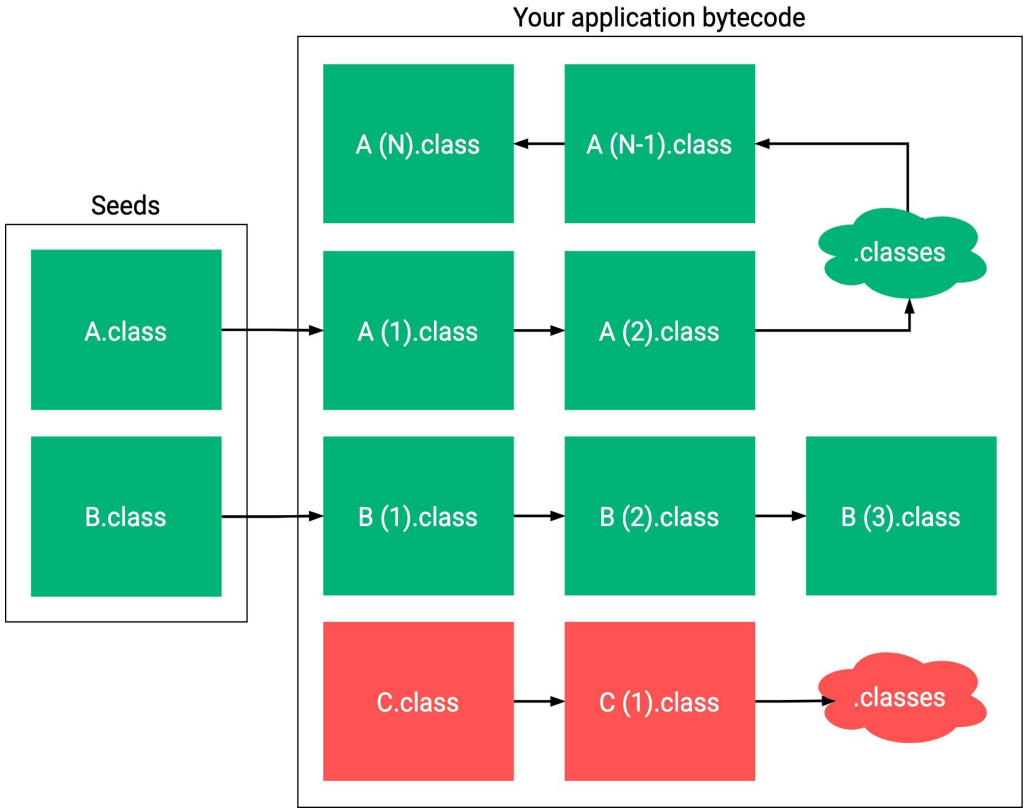

ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (optimized)
*.class (obfuscated)
*.class (obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources
-dontshrink
-dontoptimize
-dontpreverify

3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (shrunk, optimized)
*.class (shrunk, optimized, obfuscated)
*.class (shrunk, optimized, obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources

3. ProGuard

Reference



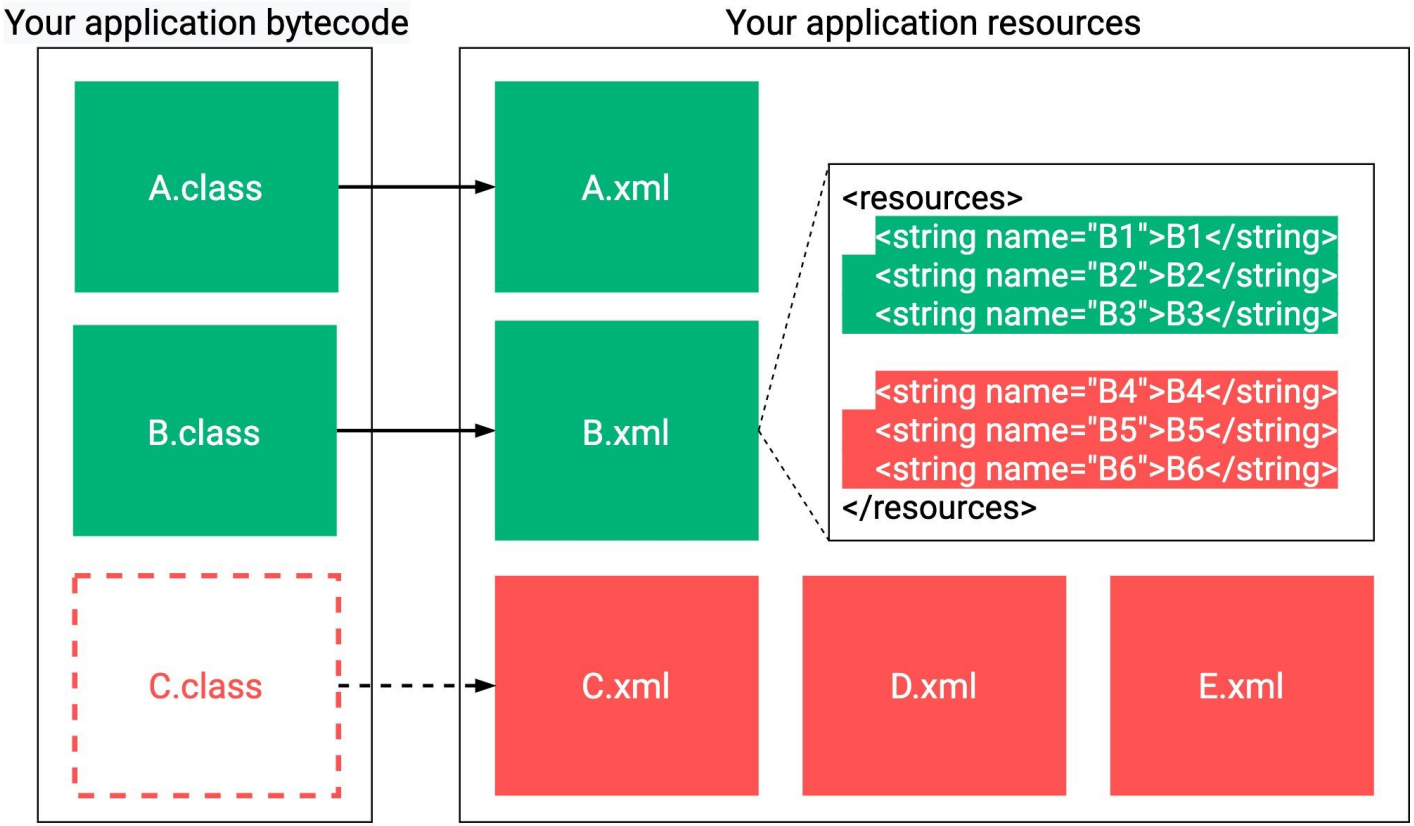
1. Определяются входные точки в программу (seeds)
2. Для каждой входной точки вычисляется граф достижимого кода
3. Достижимый код сохраняется, всё остальное удаляется

3. ProGuard

Reference

- Правил из SDK
  - proguard-android.txt
  - proguard-android-optimize.txt
- Правил из подключенных библиотек
  - <library-dir>/proguard.txt (.aar)
  - <library-dir>/META-INF/proguard/ (.jar)
- Пользовательских правил
  - <module-dir>/proguard-rules.pro

3. ProGuard

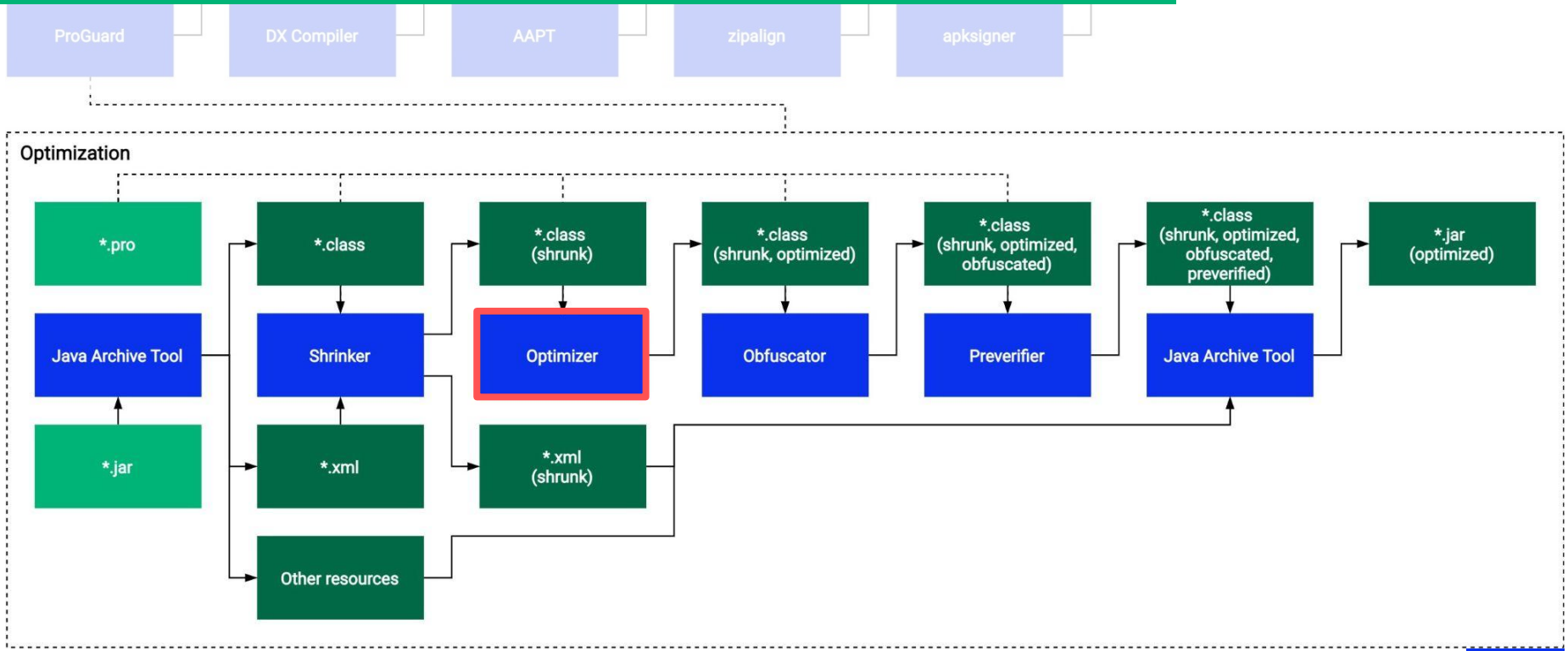

Your application bytecode
Your application resources
A.class
B.class
C.class
A.xml
B.xml
C.xml
D.xml
E.xml
<resources>
<string name="B1">B1</string>
<string name="B2">B2</string>
<string name="B3">B3</string>
<string name="B4">B4</string>
<string name="B5">B5</string>
<string name="B6">B6</string>
</resources>

3. ProGuard

Reference

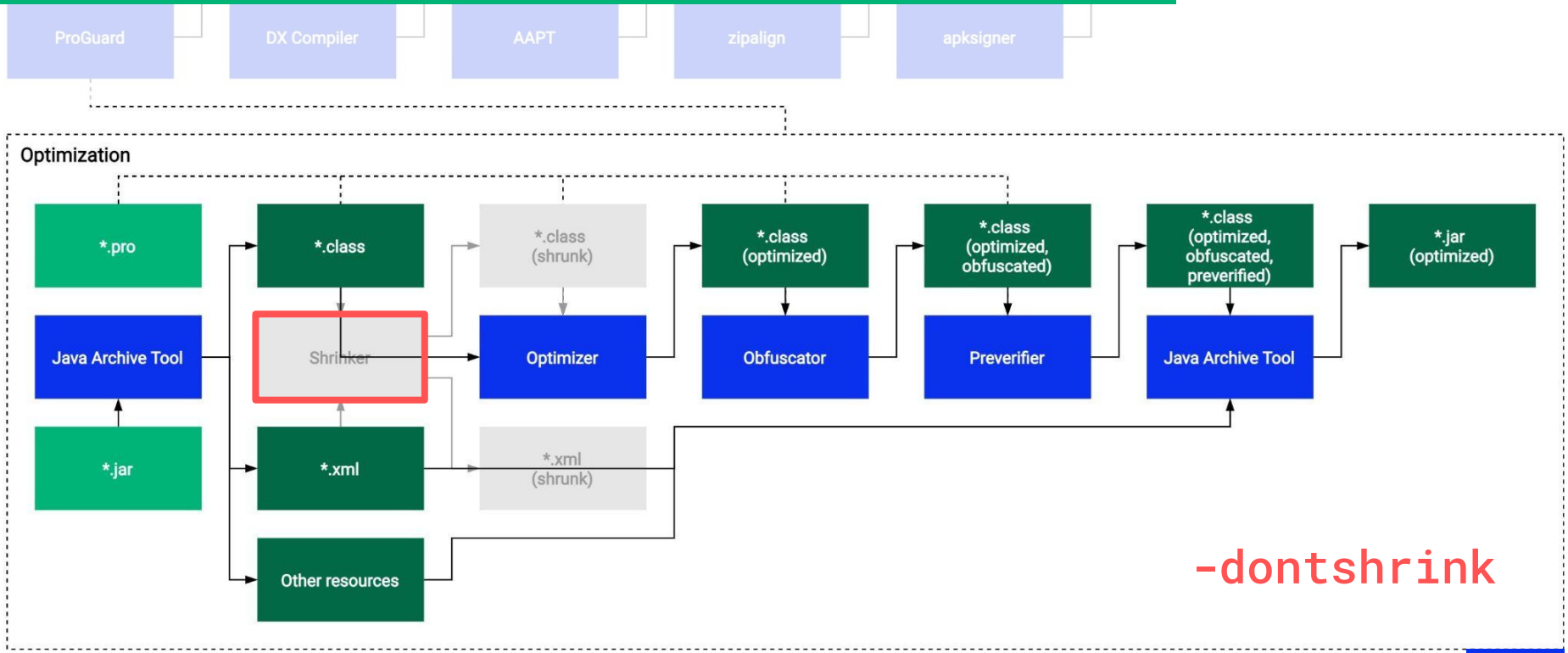
- Сохраняет ресурсы
  - Используемые напрямую
  - Используемые косвенно
- Режимы работы
  - Defense: сохранит ресурсы, используемые напрямую или косвенно
  - Strict: сохранит ресурсы, используемые напрямую
- Поддерживает пользовательские правила
  - keep.xml

3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class
(shrunk)
*.class
(shrunk, optimized)
*.class
(shrunk, optimized,
obfuscated)
*.class
(shrunk, optimized,
obfuscated,
preverified)
*.jar
(optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml
(shrunk)
Other resources

3. ProGuard

Reference



3. ProGuard

Reference

- Control flow analysis
  - Изменение графа потока управления
- Data-flow analysis
  - Оптимизация аллокаций и распространение данных
- Partial evaluation
  - Проактивное вычисление значений
- ...

3. 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);
    }
}
```

3. 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 самом деле изменяется БАЙТ-КОД

3. 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);
    }
}
```

3. ProGuard

Reference

```
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 host = "*.example.com".startsWith("*.")
+           ? "*.example.com".substring("*.".length())
+           : "*.example.com";
        String canonical = HttpUrl.get("http://" + host).host();
        System.out.println(canonical);
    }
}
```

Constant Propagation

3. ProGuard

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

3. ProGuard

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

3. ProGuard

Reference

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

3. ProGuard

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

3. ProGuard

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

3. ProGuard

Reference

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

Branch Elimination

3. ProGuard

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

3. ProGuard

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

3. ProGuard

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

3. ProGuard

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

3. ProGuard

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

3. 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);
    }
}
```

3. ProGuard

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

3. ProGuard

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

3. ProGuard

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

3. ProGuard

```
class Test {
    public static void main(String... args) {
        String canonical = "example.com";
        System.out.println(canonical);
    }
}
```

3. ProGuard

```
class Test {
    public static void main(String... args) {
        String canonical = "example.com";
        System.out.println(canonical);
    }
}
```

3. ProGuard

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

Constant Propagation

3. ProGuard

```
class Test {
    public static void main(String... args) {
        System.out.println("example.com");
    }
}
```

3. 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);
    }
}
```

3. 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");
    }
}
```

3. ProGuard

```
class Test {
    public static void main(String... args) {
        System.out.println("example.com");
    }
}
```

3. ProGuard

```
class Test {
    public static void main(String... args) {
        System.out.println("example.com");
    }
}
```

Методы оптимизации варьируются в зависимости от используемой технологии

3. ProGuard

Reference

-optimizations

- class/marking/final
  - помечает классы как final, где это возможно
- class/unboxing/enum
  - заменяет enum на целочисленные константы, где это возможно
- library/gson
- ...

3. ProGuard

Reference

```
-optimizations
    code/allocation/variable,
    code/simplification/arithmetic,
    method/inlining/*,
    !method/inlining/unique
```

! — исключение

* — все оптимизации

3. ProGuard

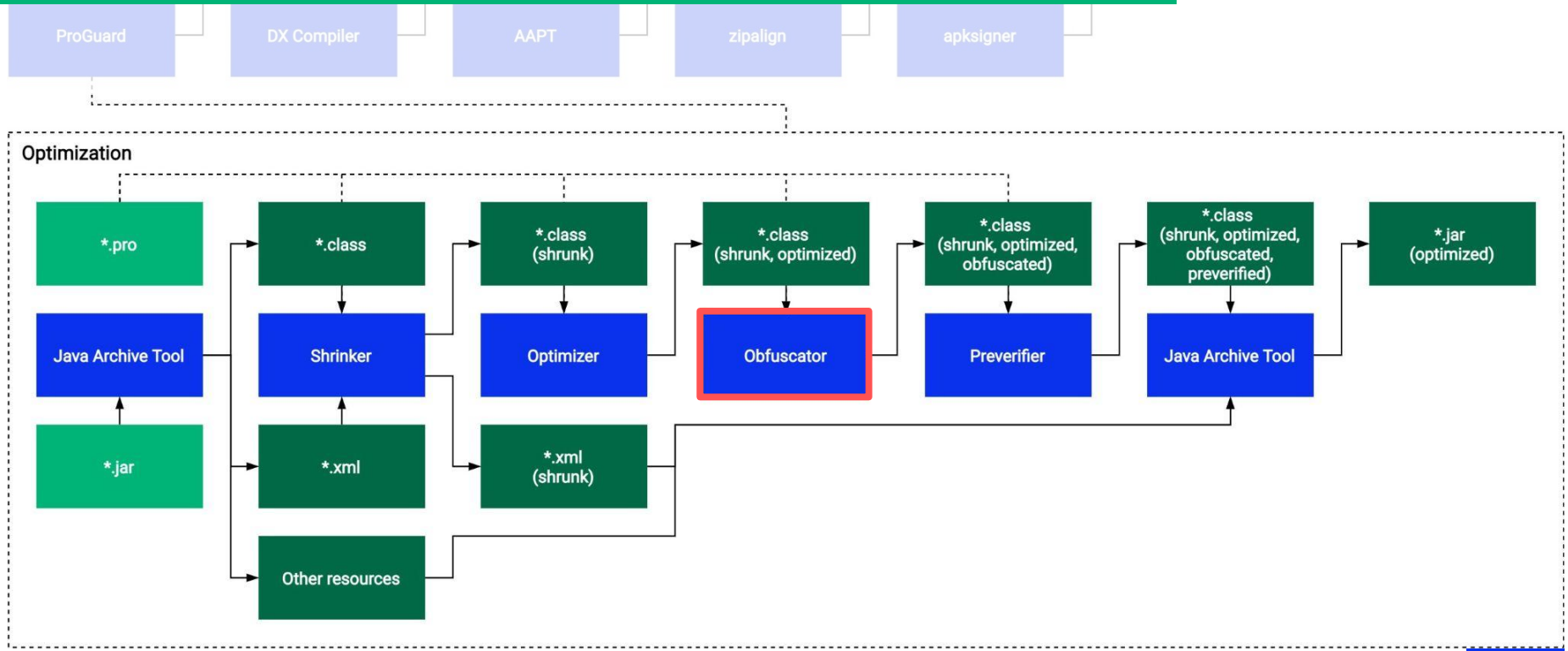
Reference

```
-optimizationpasses N
```

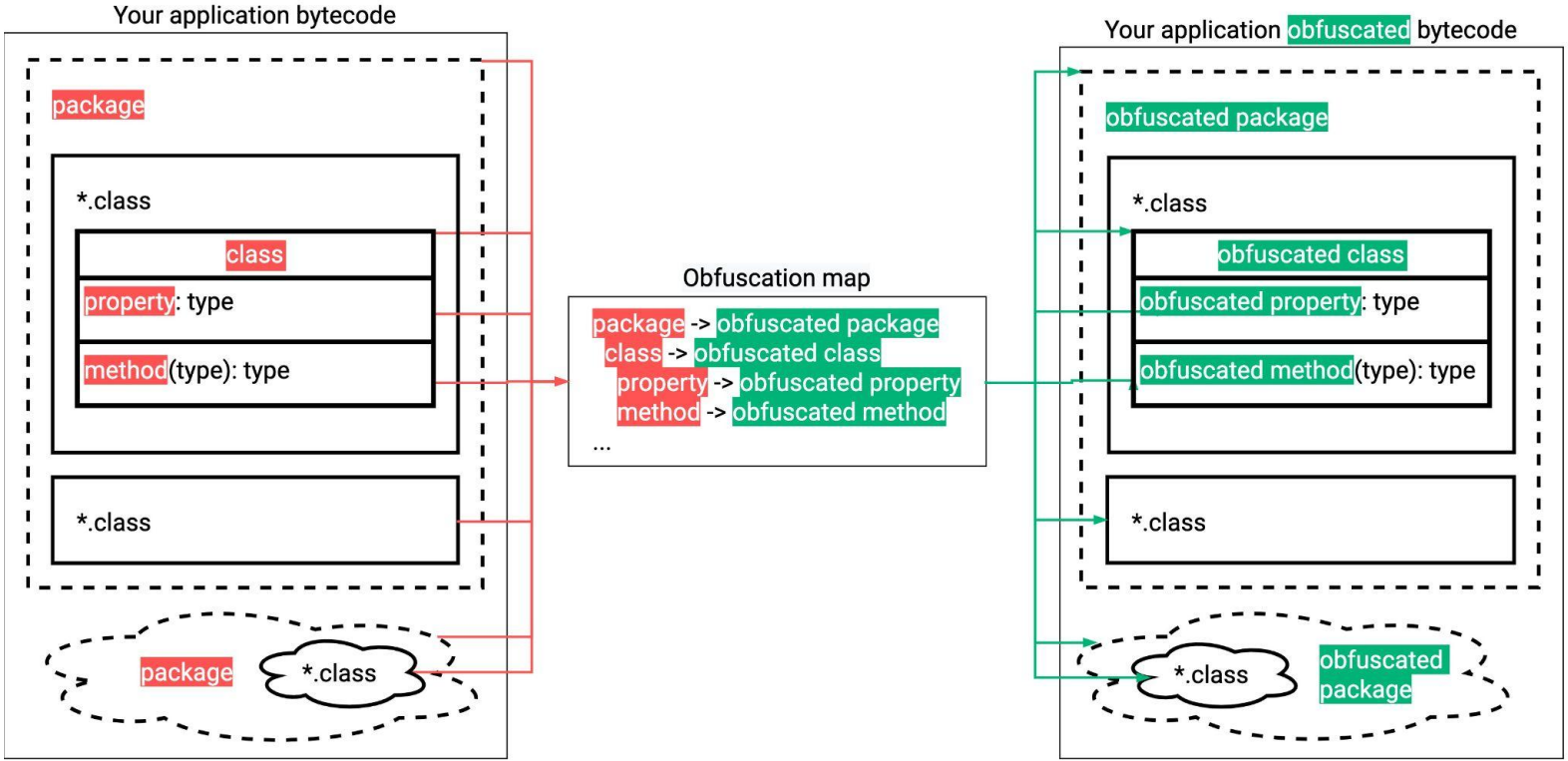
«Specifies the number of optimization passes to be performed. By default, a single pass is performed. Multiple passes may result in further improvements. If no improvements are found after an optimization pass, the optimization is ended. Only applicable when optimizing.»

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

3. ProGuard


ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class (shrunk)
*.class (shrunk, optimized)
*.class (shrunk, optimized, obfuscated)
*.class (shrunk, optimized, obfuscated, preverified)
*.jar (optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml (shrunk)
Other resources

3. ProGuard


Your application bytecode
package
*.class
class
property: type
method(type): type
*.class
package
*.class
Obfuscation map
package -> obfuscated package
class -> obfuscated class
property -> obfuscated property
method -> obfuscated method
...
Your application obfuscated bytecode
obfuscated package
*.class
obfuscated class
obfuscated property: type
obfuscated method(type): type
*.class
*.class
obfuscated package

3. 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);
        }
    }
}
```

3. 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);
        }
    }
}
```

На самом деле изменяется БАЙТ-КОД

3. 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);
        }
    }
}
```

3. ProGuard

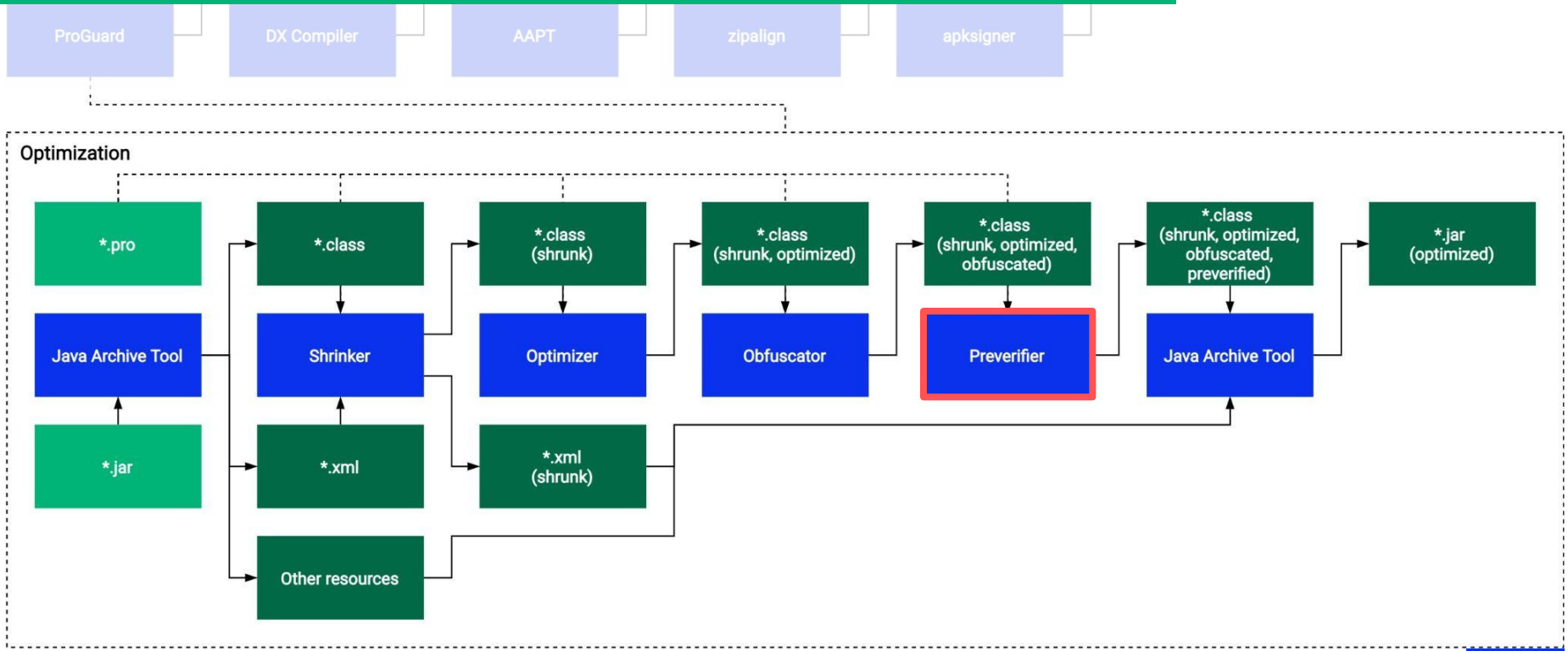
Source name	Obfuscated name
com.example.proguard.obfuscation	a.a.a.a
PinCodeBuilder	a
pinCodeLength	b
pinCodeStringBuilder	c
increment	d

3. ProGuard

```
package a.a.a.a;
public class a {
    private int b;
    private StringBuilder c = new StringBuilder(pinCodeLength);

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

3. ProGuard

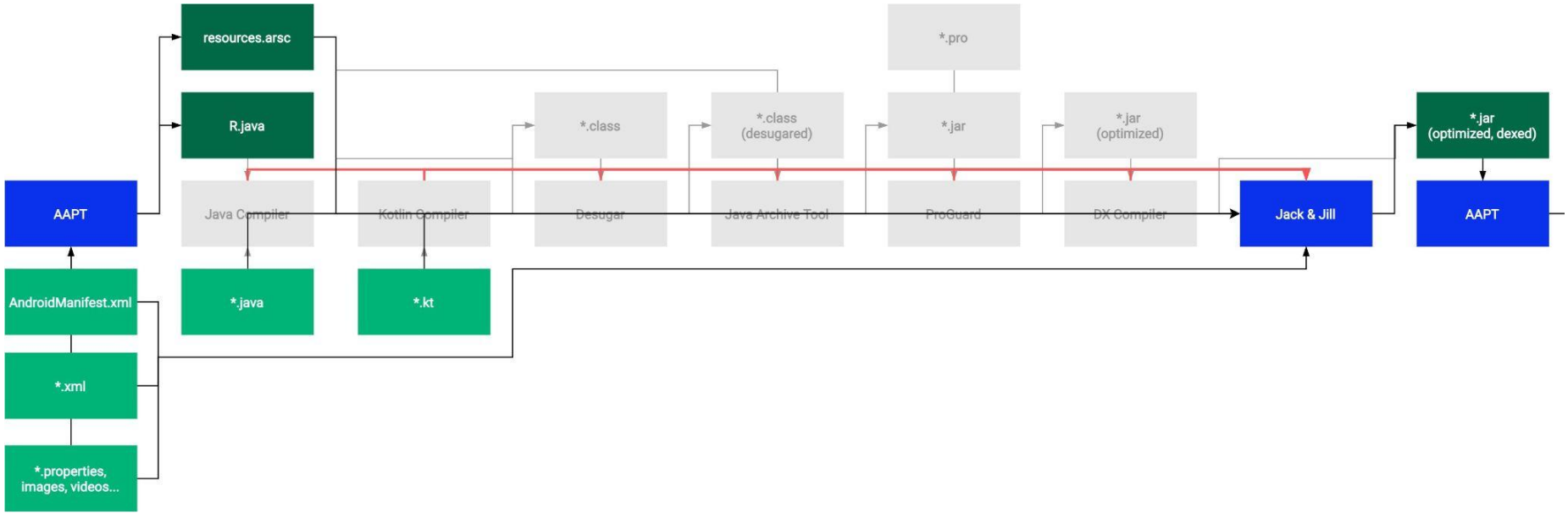

ProGuard
DX Compiler
AAPT
zipalign
apksigner
Optimization
*.pro
*.class
*.class
(shrunk)
*.class
(shrunk, optimized)
*.class
(shrunk, optimized,
obfuscated)
*.class
(shrunk, optimized,
obfuscated,
preverified)
*.jar
(optimized)
Java Archive Tool
Shrinker
Optimizer
Obfuscator
Preverifier
Java Archive Tool
*.jar
*.xml
*.xml
(shrunk)
Other resources

3. ProGuard

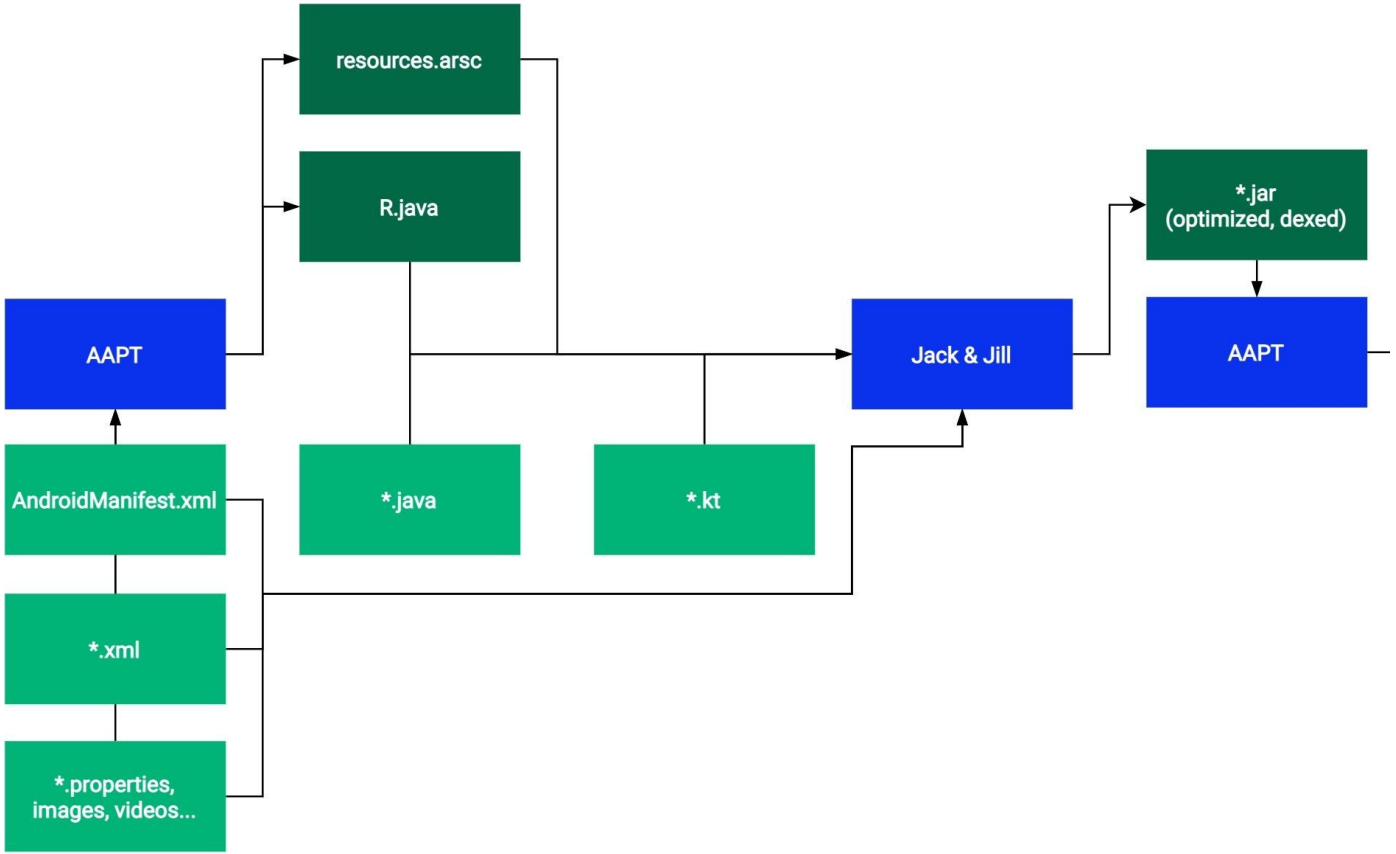
- -android укажет на необходимость произвести проверку на соответствие target API и совместимости
- Для .dex компилятор и Dalvik VM не поддерживает верификация, поэтому можно использовать -donotpreverify для сокращения времени оптимизации

4. D8, R8

Reference



4. D8, R8


resources.arsc
R.java
*.jar
(optimized, dexed)
AAPT
Jack & Jill
AAPT
AndroidManifest.xml
*.java
*.kt
*.xml
*.properties,
images, videos...

Reference

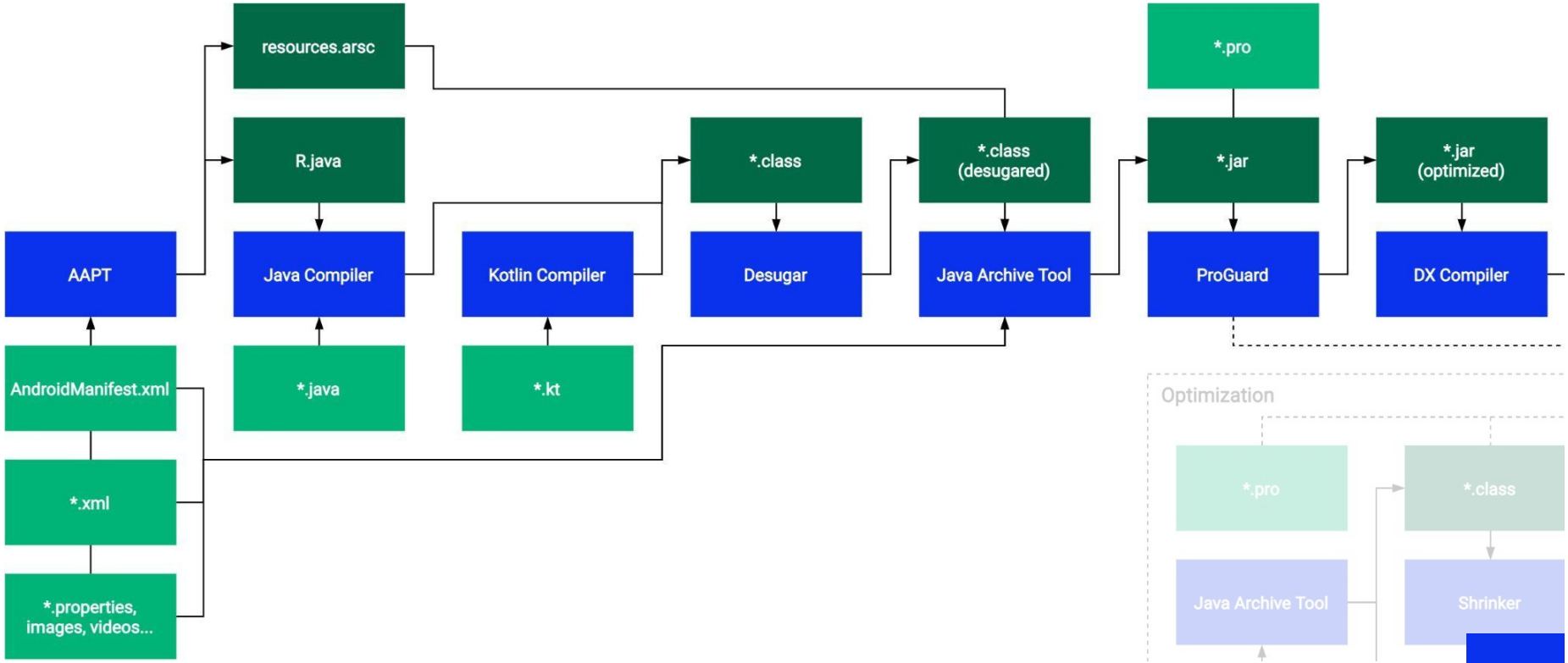
4. D8, R8

D8 is the next-generation dex compiler

- Used by default since AGP 3.2

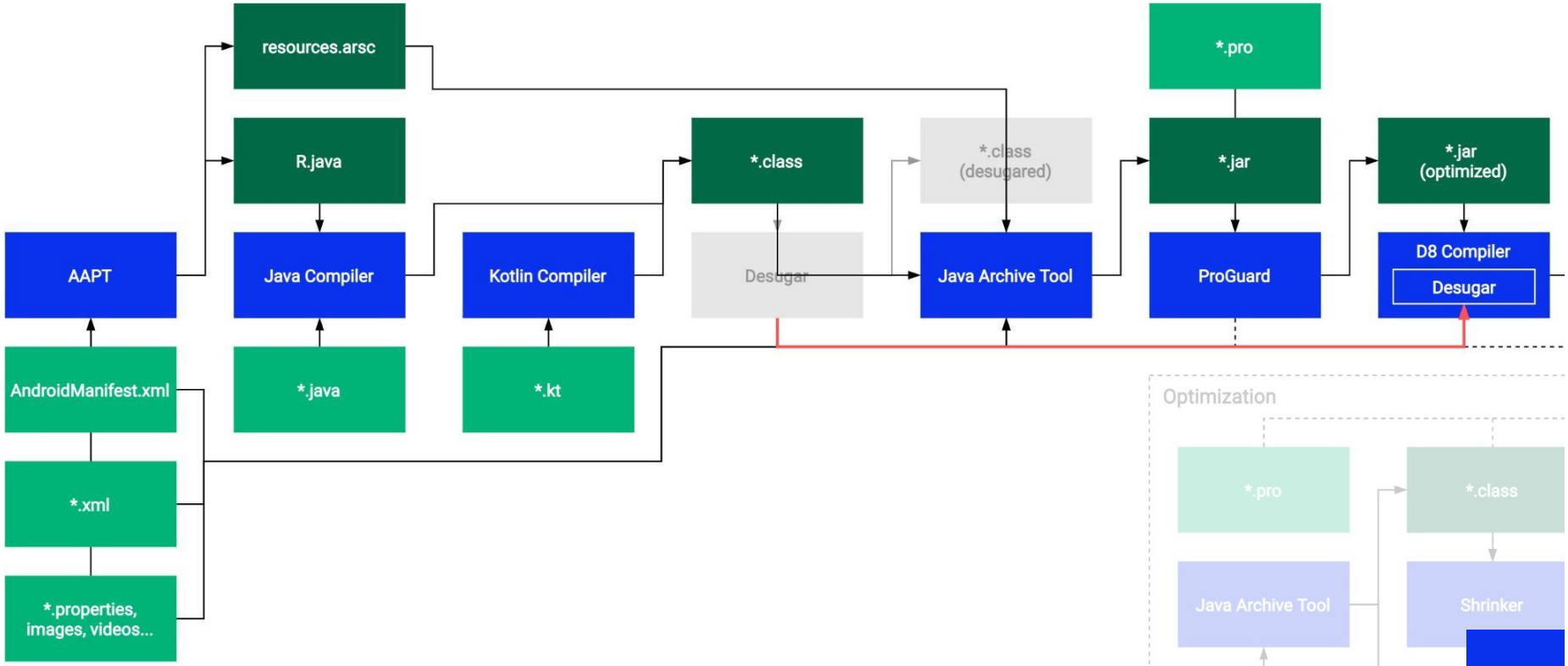
4. D8, R8.

Reference



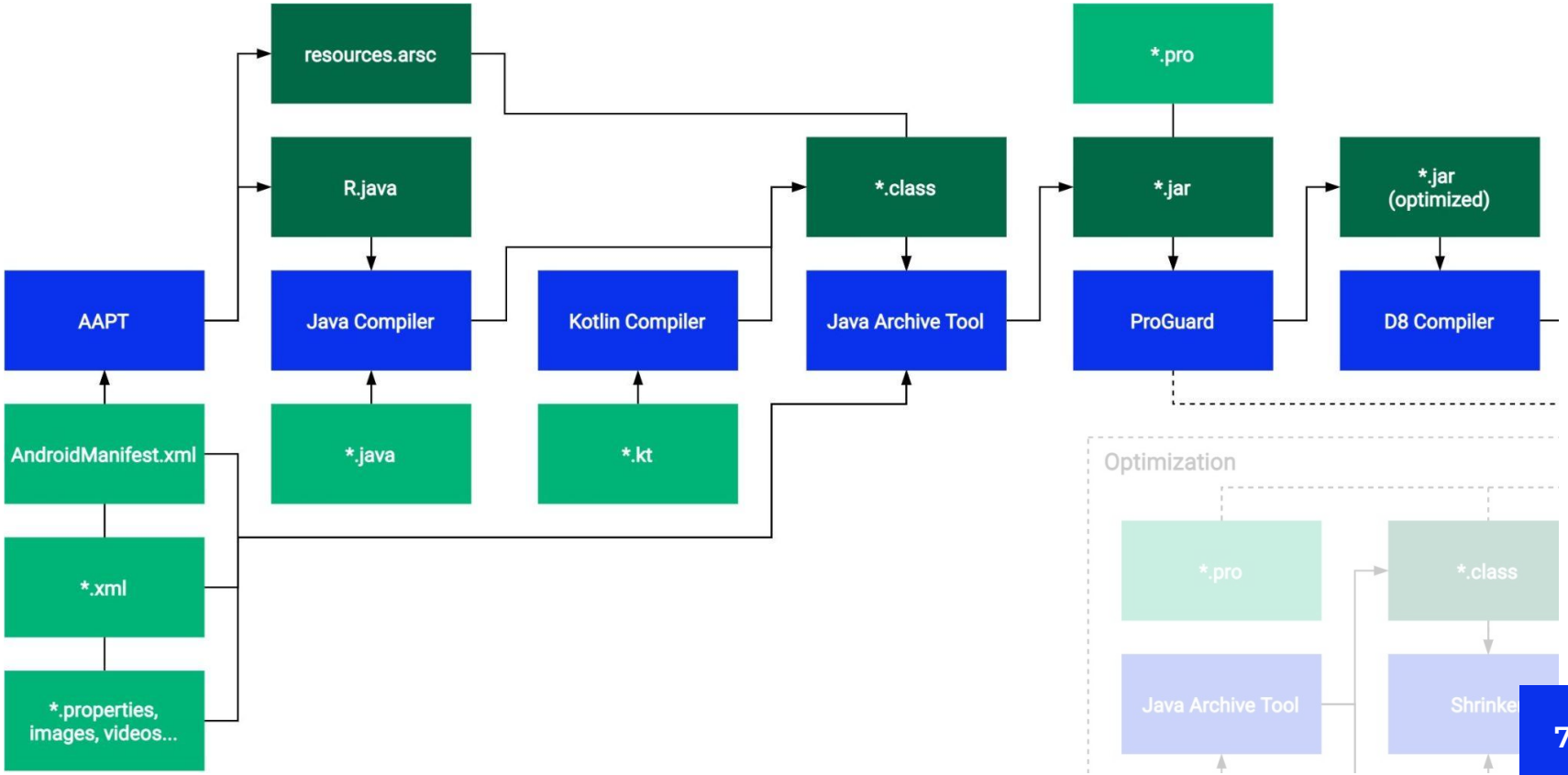
4. D8, R8.

Reference



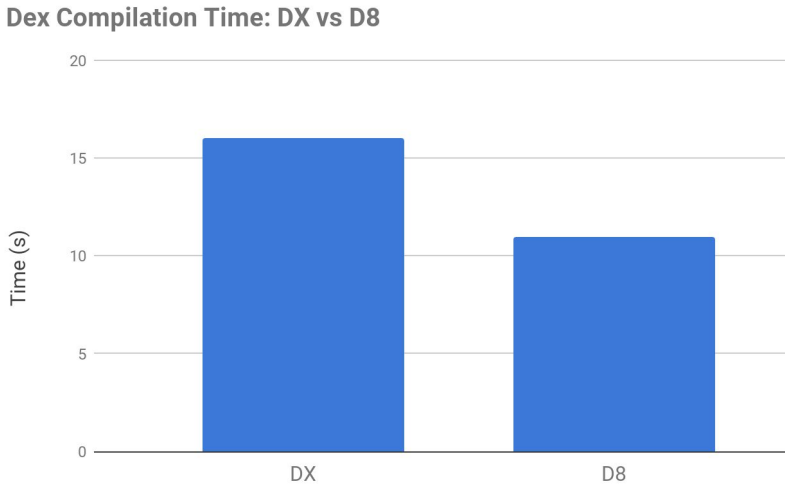
4. D8, R8.

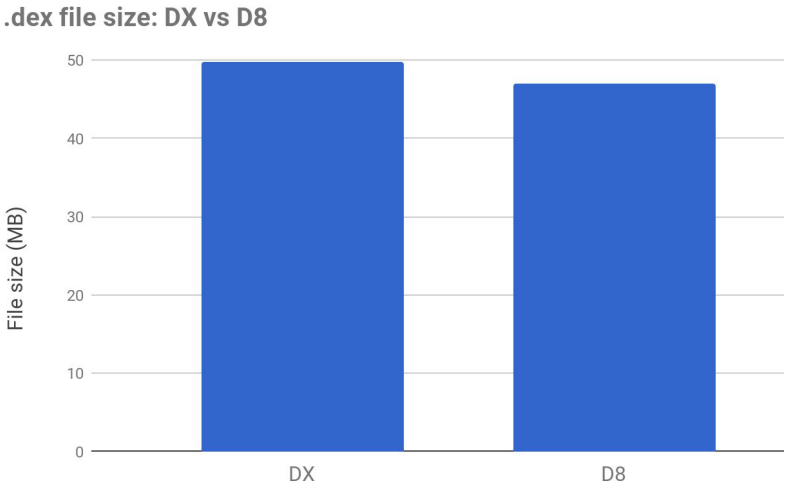
Reference



4. D8, R8.

Reference





- D8 быстрее компилирует
- D8 сильнее уменьшает размер .dex

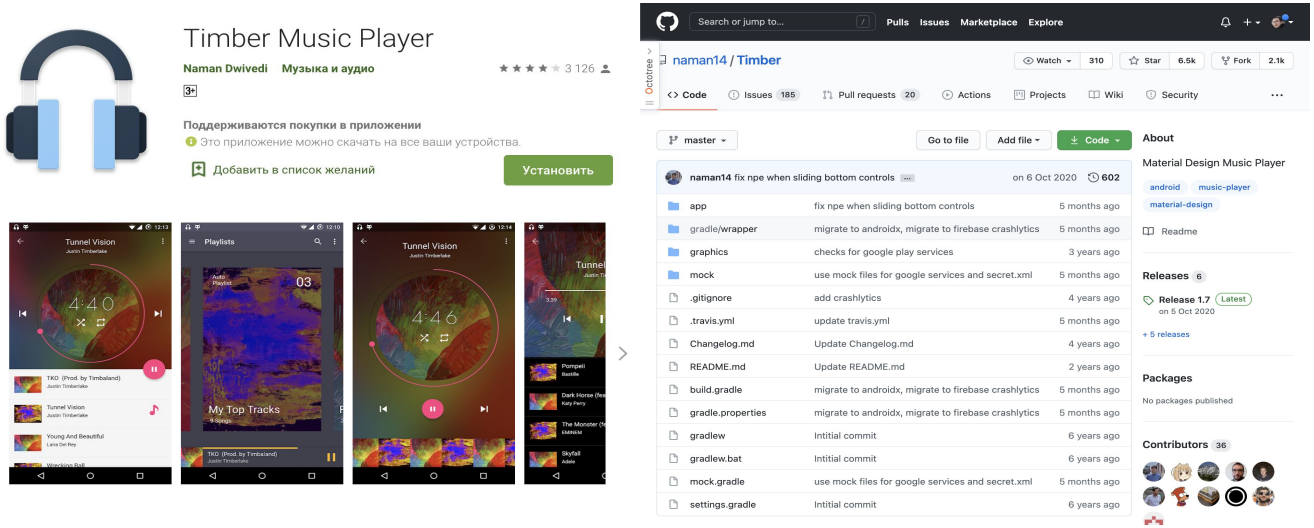
4. D8, R8

Далее мой эксперимент

- Проводится в исследовательских целях
- Результат не претендует на абсолютную точность

4. D8, R8

- Timber - Material Design Music Player (Google Play Store, GitHub)



4. D8, R8

Reference

Условия эксперимента:

Code	
.java files number	164
all files number	317
.java SLOC	20562
all SLOC	29130
.java size (KB)	~1005
all size (KB)	~1614

Project	
AGP version	3.3.1
GP version	4.10.1
minSdkVersion	16
compileSdkVersion	28
Commit	3da50f11
Build variant	release

4. D8, R8

1. Настроить gradle на использование **DX**

   `<project-name>/gradle.properties:`

   ```
   android.enableD8=false
   android.enableD8.desugaring=false
   ```

2. Компиляция .dex (повторить 5 раз)

   ```
   <project-name> % ./gradlew clean; ./gradlew cleanBuildCache
   <project-name> % time ./gradlew build(Your build variant)
   ```

3. Фиксируем результаты
   - 3.1. Вычисляем среднее арифметическое времени компиляции
   - 3.2. Забираем <project-name>/app/build/intermediates/dex/.../classes*.dex

4. D8, R8

1. Настроить gradle на использование **D8**

   `<project-name>/gradle.properties:`

   ```
   android.enableD8=true
   android.enableD8.desugaring=true
   ```

2. Компиляция .dex (повторить 5 раз)

   ```
   <project-name> % ./gradlew clean; ./gradlew cleanBuildCache
   <project-name> % time ./gradlew build(Your build variant)
   ```

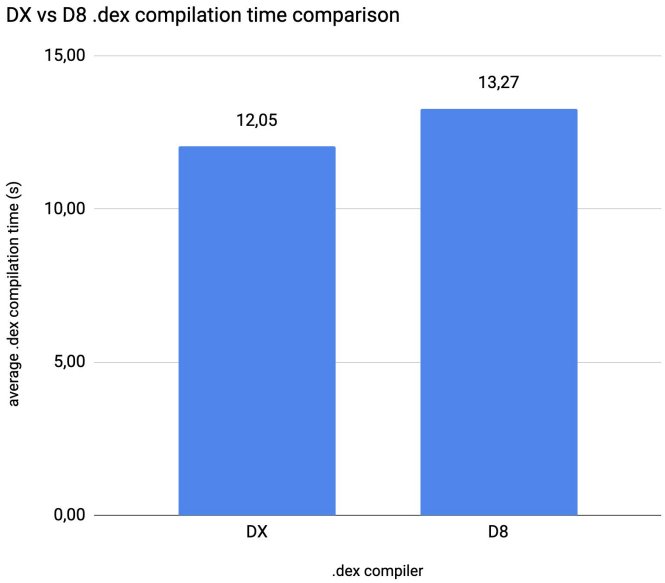
3. Фиксируем результаты

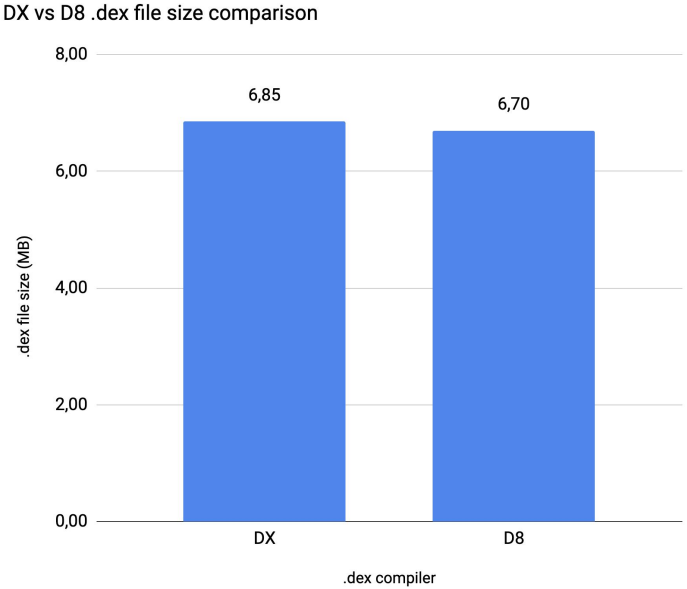
   3.1. Вычисляем среднее арифметическое времени компиляции

   3.2. Забираем <project-name>/app/build/intermediates/dex/.../classes*.dex

4. D8, R8

Reference





- DX быстрее компилирует
- D8 сильнее уменьшает размер .dex

4. D8, R8

Вопрос: Почему скорость компиляции D8 больше DX?

Предположение: В Timber не используются фичи Java, требующие desugaring.

4. D8, R8

Timber/app/build.gradle:

```
compileOptions {
    sourceCompatibility = JavaVersion.VERSION_1_8
    targetCompatibility = JavaVersion.VERSION_1_8
}
```

4. D8, R8

Вопрос: Откуда появляется разница в размере .dex?

Ответ: DX и D8 имеют небольшие различия в генерации Dalvik байткода.

4. D8, R8

Reference

```
...
public String getUsername() {
    if (mUserSession != null) return mUserSession.mUsername;
    return null;
}
...
```

Reference

4. D8, R8

Дизассемблируем Dalvik байткод в smali код:

```
dex2jar-2.0 % sh ./d2j-dex2smali.sh
<input>.dex -o <output-dir>
```

4. D8, R8

```
...
.method public getUsername()Ljava/lang/String;
    .registers 2
    .prologue
    .line 278
    iget-object v0, p0,
Lcom/naman14/timber/lastfmapi/LastFmClient;->mUserSession:Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;
    if-eqz v0, :L1
    iget-object v0, p0,
Lcom/naman14/timber/lastfmapi/LastFmClient;->mUserSession:Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;
    iget-object v0, v0, Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;->mUsername:Ljava/lang/String;
    :L0
    .line 279
    return-object v0
    :L1
    const/4 v0, 0
    goto :L0
.end method
...
```

4. D8, R8

```
...
.method public getUsername()Ljava/lang/String;
    .registers 2
    .line 278
    iget-object v0, p0,
Lcom/naman14/timber/lastfmapi/LastFmClient;->mUserSession:Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;
    if-eqz v0, :L0
    iget-object v0, v0, Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;->mUsername:Ljava/lang/String;
    return-object v0
    :L0
    const/4 v0, 0
    return-object v0
.end method
...
```

D8 smali

4. D8, R8

```
...
.method public getUsername()Ljava/lang/String;
    .registers 2
-   .prologue
    .line 278
    iget-object v0, p0,
Lcom/naman14/timber/lastfmapi/LastFmClient;->mUserSession:Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;
-   if-eqz v0, :L1
+   if-eqz v0, :L0
-   iget-object v0, p0,
Lcom/naman14/timber/lastfmapi/LastFmClient;->mUserSession:Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;
    iget-object v0, v0, Lcom/naman14/timber/lastfmapi/models/LastfmUserSession;->mUsername:Ljava/lang/String;
+   return-object v0
    :L0
-   .line 279
-   return-object v0
-   :L1
    const/4 v0, 0
-   goto :L0
+   return-object v0
.end method
...
```

DX smali → D8 smali

4. D8, R8

LastFmClient.smali (getUsername)			
Param	DX	D8	Diff (D8-DX)
Words	71	52	-19 (-26%)
Characters	590	424	-196 (-33%)

4. D8, R8

DX

```
6465 780a 3033 3500 c591 8dc7 f678 60ca
c299 7060 d2f8 aaca 9131 6959 afcf e579
d09b 6d00 7000 0000 7856 3412 0000 0000
0000 0000 f055 1500 7cdc 0000 7000 0000
331e 0000 6072 0300 632e 0000 2ceb 0300
5cbb 0000 d017 0600 dcc9 0000 b0f2 0b00
a318 0000 9041 1200 e045 5800 f055 1500
c0b1 4a00 c2b1 4a00 17b8 4a00 1ab8 4a00
1db8 4a00 23b8 4a00 56b8 4a00 95b8 4a00
98b8 4a00 a6b8 4a00 adb8 4a00 b8b8 4a00
ccb8 4a00 d8b8 4a00 f0b8 4a00 05b9 4a00
21b9 4a00 3eb9 4a00 54b9 4a00 57b9 4a00
...
```

D8

```
6465 780a 3033 3500 13e9 76a9 72e0 0d29
95c2 5f4f 56d7 7b33 e484 51cb b4c5 01d4
6429 6b00 7000 0000 7856 3412 0000 0000
0000 0000 8828 6b00 5cd9 0000 7000 0000
331e 0000 e065 0300 632e 0000 acde 0300
5cbb 0000 500b 0600 dcc9 0000 30e6 0b00
a318 0000 1035 1200 f4df 5500 7049 1500
aeb3 4600 b0b3 4600 05ba 4600 08ba 4600
0bba 4600 11ba 4600 44ba 4600 83ba 4600
86ba 4600 94ba 4600 9bba 4600 a6ba 4600
baba 4600 c6ba 4600 deba 4600 f3ba 4600
0fbb 4600 2cbb 4600 42bb 4600 45bb 4600
...
```

Reference

4. D8, R8

R8 is the next-generation app optimizator

- Used by default since AGP 3.4

4. D8, R8

- proguard-rules.pro
- consumer-rules.pro
- ./build.gradle:

```
proguardFiles
getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
```

4. D8, R8

ProGuard и R8 — разные технологии

4. D8, R8

Хотим использовать AGP 3.4+, но не хотим использовать R8.

```
<project-name>/gradle.properties:
    android.enableR8=false
    android.enableR8.libraries=false
```

4. D8, R8

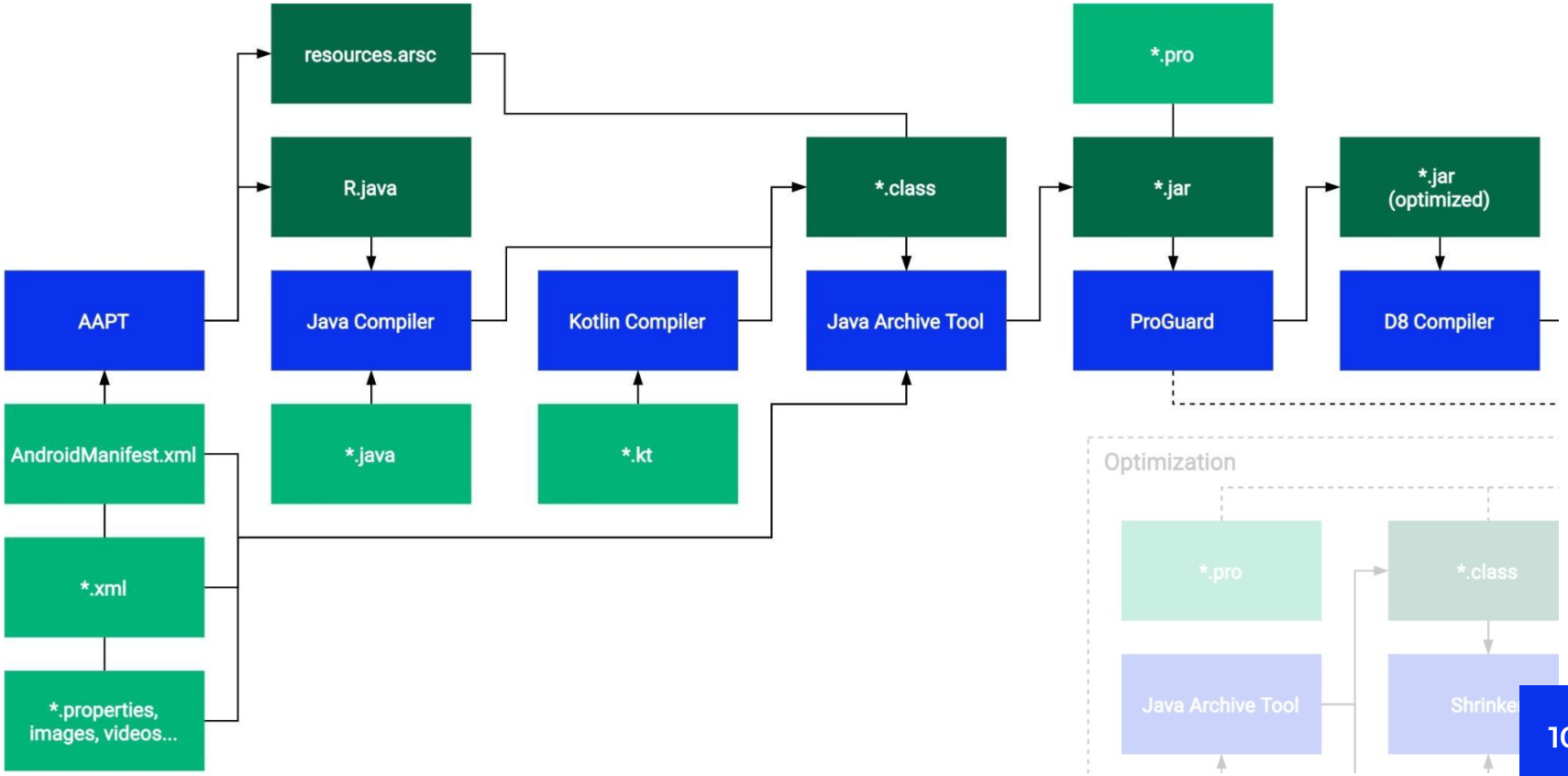
Хотим использовать AGP 3.4+ и R8, но раньше использовали ProGuard.

Появились конфликты:

1. Смотрим в `<module-name>/build/outputs/mapping/<build-type>/`:
   - `configuration.txt` (`-printconfiguration`)
   - `mapping.txt` (`-printmapping`)
   - `seeds.txt` (`-printseeds`)
   - `usage.txt` (`-printusage`)
   - `resources.txt`
2. Анализируем, исправляем, наслаждаемся

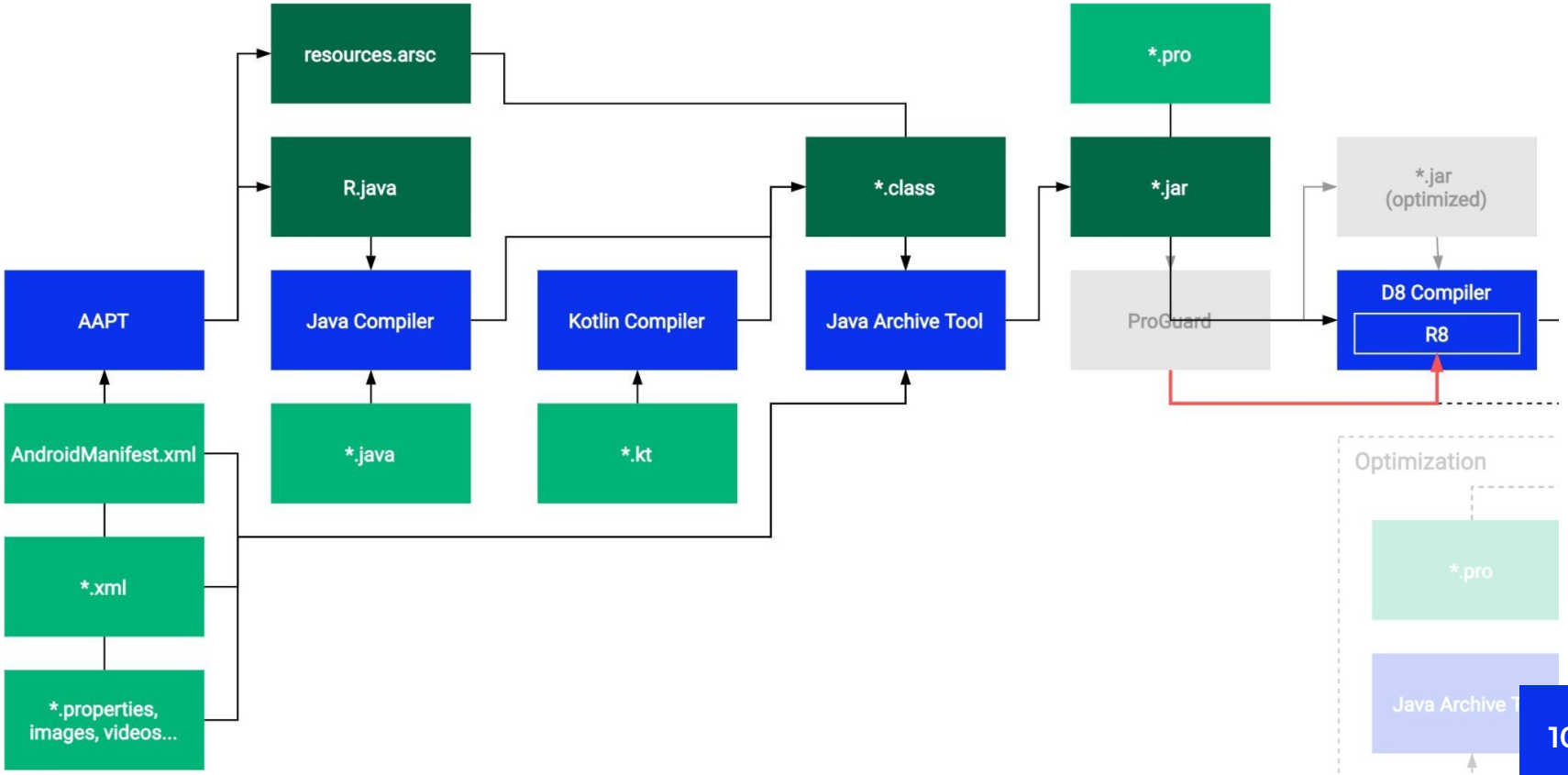
4. D8, R8

Reference

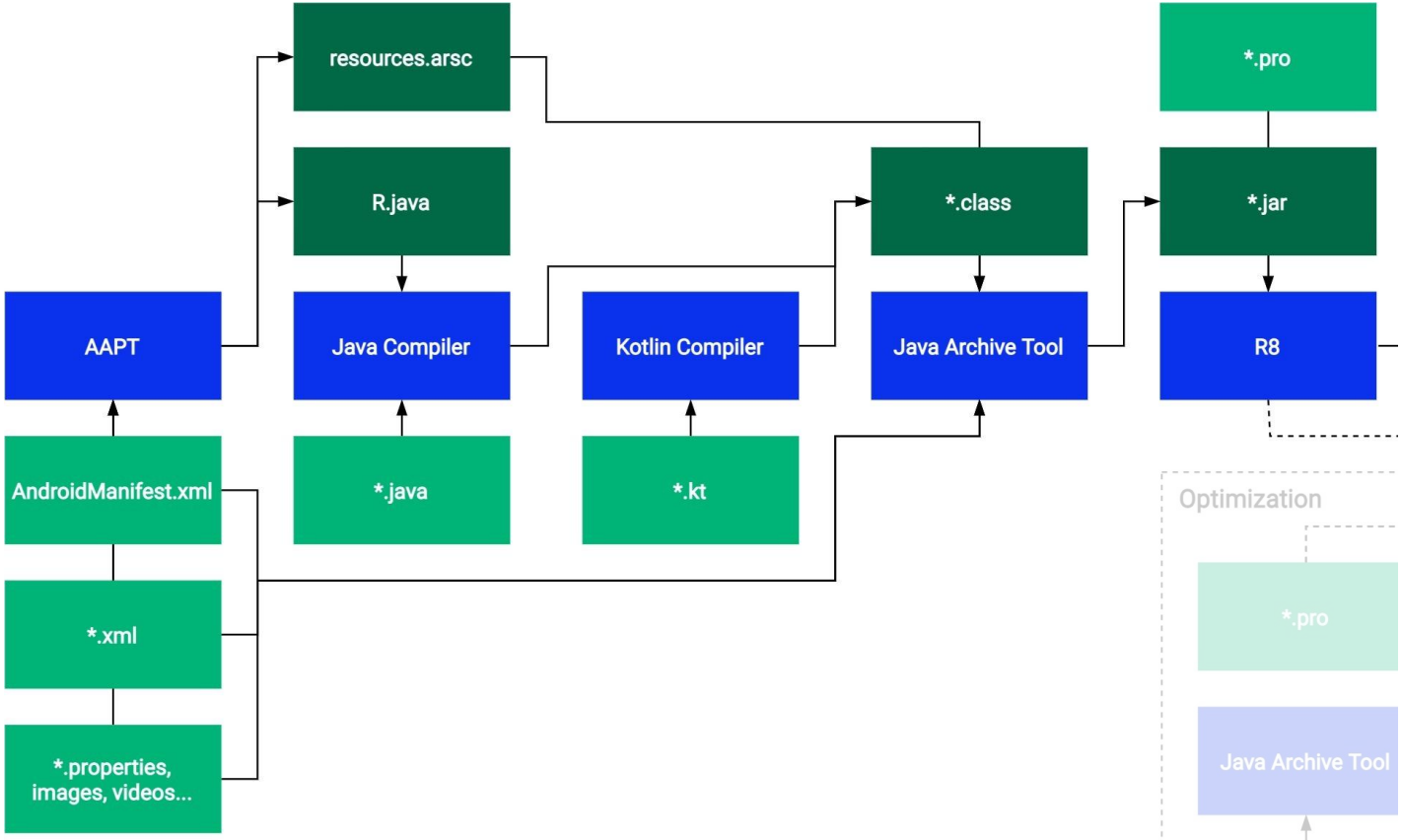


4. D8, R8

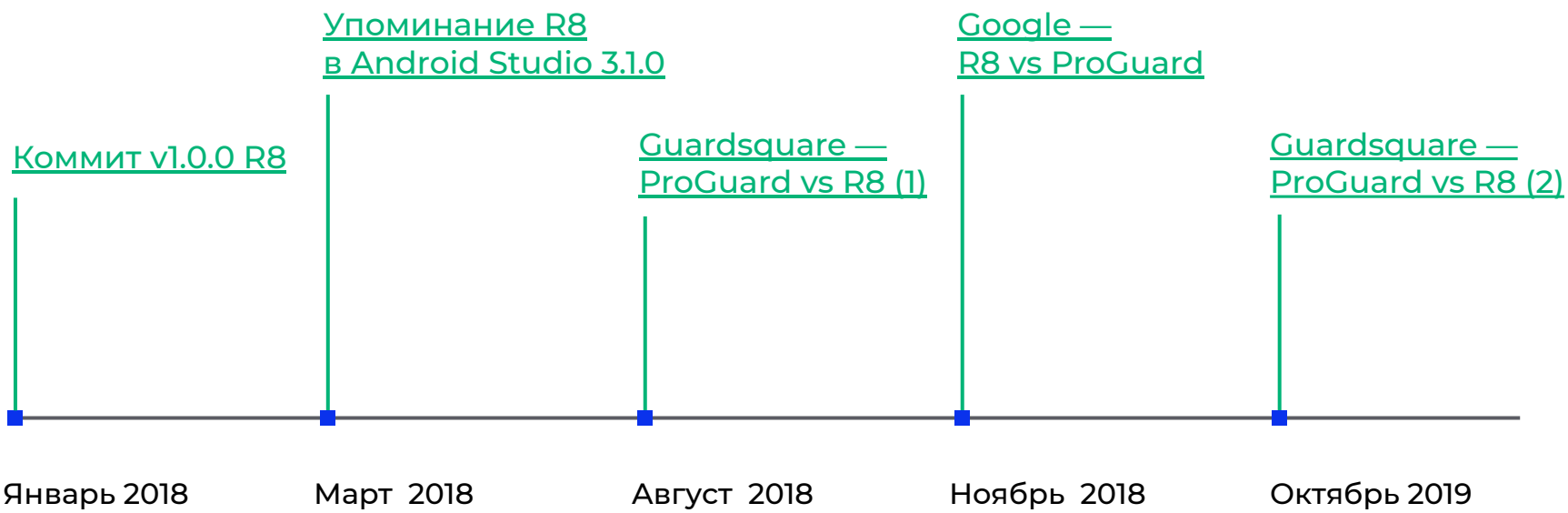
Reference



4. D8, R8


resources.arsc
R.java
AAPT
Java Compiler
Kotlin Compiler
*.class
Java Archive Tool
*.pro
*.jar
R8
AndroidManifest.xml
*.java
*.kt
*.xml
*.properties,
images, videos...
Optimization
*.pro
Java Archive Tool

4. D8, R8

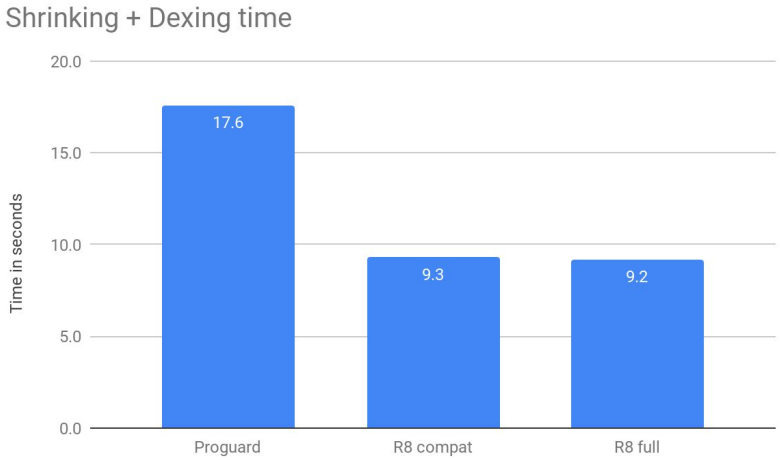

Коммит v1.0.0 R8
Январь 2018
Упоминание R8
в Android Studio 3.1.0
Март 2018
Guardsquare —
ProGuard vs R8 (1)
Август 2018
Google —
R8 vs ProGuard
Ноябрь 2018
Guardsquare —
ProGuard vs R8 (2)
Октябрь 2019

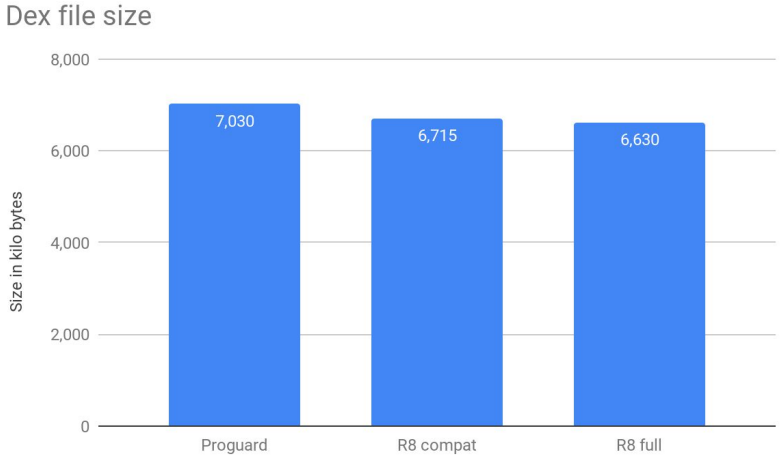
4. D8, R8

Характеристика	2018		2019	
	PG	R8	PG	R8
Время оптимизации	40 сек (1) 60 сек (5)	60 сек	11 сек (1) (-29 сек) 37 сек (5) (-23 сек)	32 сек (-28 сек)
Оптимизации кода	22	14	543 (+521)	25 (+11)
Оптимизации кода (уникальные)	12	3	7 (-5)	3
Сокращение размера приложения	-	-	8,5%	10%
Лет в разработке	15	0	16 (+1)	1 (+1)

4. D8, R8

Reference





- R8 быстрее компилирует .dex
- R8 генерирует .dex меньшего размера

5. Тайны обфускации

```
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);
        }
    }
}
```

5. Тайны обфускации

```
package a.a.a.a;
public class a {
    private int b;
    private StringBuilder c = new StringBuilder(pinCodeLength);

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

Reference

5. Тайны обфускации

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

5. Тайны обфускации

```
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);
        }
    }
}
```

5. Тайны обфускации

```
package a.a.a.a;
public class a {
    private int b;
    private StringBuilder c = new StringBuilder(pinCodeLength);

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

5. Тайны обфускации

```
package a.a.a.a;
public class a {
    private int b;
    private StringBuilder c = new StringBuilder(pinCodeLength);
    public a(int pinCodeLength) {
    }
    public void d(int pinCodeDigit) {
        if (c.length() < b) {
            c.append(pinCodeDigit);
        }
    }
}
```

$$\frac{\text{до обфускации}}{\text{после обфускации}} = \frac{\text{402 символа}}{\text{250 символов}}$$

5. Тайны обфускации

./build.gradle:

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

Reference

5. Тайны обфускации

«...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...»

5. Тайны обфускации

По умолчанию обфускация ProGuard, R8 — минификация

5. Тайны обфускации

```
Caused by: java.lang.Exception: Hello Mobius 2021!
    at c.i.l.i.a.a(:18)
    at c.i.c.e.a.a(:14)
    at c.i.c.f.b.<init>(:27)
    at c.i.c.d.d.b(:45)
    at com.technokratos.articles.presentation.ArticlesFragment$c.c(:72)
    at com.technokratos.articles.presentation.ArticlesFragment$c.c(:24)
    at c.i.i.i.a.a(:19)
    at b.q.z.a(:164)
    at b.q.z.a(:130)
    at com.technokratos.articles.presentation.ArticlesFragment.b1(:116)
    at c.i.i.h.d.a(:39)
    at b.n.a.j.a(:892)
    at b.n.a.j.a(:2100)
```

5. Тайны обфускации

```
package a.a.a.a;
public class a {
    private int a = "YOUR_TOP_SECRET";
    ...
    public void b(int value) {
        if (value % 1234 == 0) {
            // Send 1 billion dollars
            ...
        } else if (value.toString() == a) {
            // Send 2 billion dollars
            ...
        }
    }
}
```

5. Тайны обфускации

Почему не стоит беспокоиться?

- Приложение не содержит ценную бизнес логику
- Приложение не хранит чувствительные пользовательские данные
- ...

Почему стоит беспокоиться?

- Ценная бизнес логика прослеживается
- Видны важные литералы
- ...

5. Тайны обфускации

Reference

Техники обфускации кода:

- Запутывание названий
- Шифрование литералов
- Обфускация control flow
- Преобразования языковых конструкций
- Вставка фиктивного кода
- Удаление метаданных
- Слияние кода
- Anti-tampering
- ...

5. Тайны обфускации

Обфускация может увеличить размер байт-кода и понизить скорость работы!

5. Тайны обфускации

Более тонкая настройка ProGuard, R8

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

5. Тайны обфускации

Обфускаторы Android приложений

- Obfuscapk
- AAMO
- Paranoid

Обфускаторы Java байт-кода

- yGuard
- JODE
- JavaGuard
- jarg

Обфускаторы JavaScript кода

- javascript-obfuscator
- js-obfuscator

5. Тайны обфускации

DexGuard

- Коммерческий продукт
- Улучшенная версия ProGuard
- Полная совместимость с ProGuard конфигурацией
- Обфускация имён, арифметических выражений, control flow, нативного кода, имён библиотек, ресурсов, вызовов SDK...
- Шифрование классов, ресурсов, нативных библиотек...

6. Резюме

Перед началом оптимизации расставьте приоритеты и ограничения.

6. Резюме

Итоговый результат оптимизации зависит от множества факторов: бизнес логика, сторонние сервисы, мобильное приложение.

6. Резюме

Оптимизация сборки — удаление кода, ресурсов приложения, увеличение скорость работы, защита от реверс-инжиниринга.

6. Резюме

ProGuard

- 18 лет в разработке
- 543 техник оптимизации кода
- 8.5% сокращение размера сборки
- 37 секунд на оптимизацию сборки

6. Резюме

R8 (+D8)

- 3 года в разработке
- 25 техник оптимизации кода
- 10% сокращение размера сборки
- 32 секунд на оптимизацию сборки

6. Резюме

DexGuard, Obfuscapk, AAMO, Paranoid — альтернативные оптимизаторы Android сборок.

6. Резюме

За оптимизацию нужно платить временем и силами. Постарайтесь сократить расходы и равномерно распределить эти ресурсы :)

7. Ссылки

1. Android Developers. Shrink, obfuscate, and optimize your app.
2. Android Developers, Medium. Practical ProGuard rules examples.
3. Guardsquare. ProGuard manual.
4. Android Developers, Medium. Troubleshooting ProGuard issues on Android
5. IMStudio, Medium. Android Journey: Proguard, D8, R8 what are they?
6. Otus, YouTube. ProGuard / R8 ...
7. Android Developers, YouTube. Shrinking Your App with R8.
8. Yonatan V. Levin. Android CPU, Compilers, D8 & R8.
9. Jake Wharton, R8 Optimizations.
10. inwady, Habr. 6 способов спрятать данные в Android-приложении.
11. forceLain, Habr. Как перестать бояться Proguard и начать жить.
12. PreEmptive Solutions. The Unofficial R8 Documentation.

Оптимизация сборок Android приложений: ProGuard, D8, R8. Тайны обфускации

Петров Валерий. Технократия.