UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

GREEN Pauware

For a power-thrifty mobile app marketplace

Olivier Le Goaër
olivier.legoaer@univ-pau.fr

http://green.pauware.com

Rationales

- A growing landscape of devices
  - Billions of smartphone and tablets owners but also wearables, TVs, and much more to come with IoT
  - Powered by mobile platforms like Android (leader in market share)

- An overwhelming number of apps (the case of Android)
  - 2,6 million of apps available on store (2018)
  - 19 billions of downloads each year (2017)

Does one of the biggest software industries on this planet is eco-responsible?

A negative impact on the environment

- When microwatts are precious
  - Taming the power consumption of mobile apps is part of the responses to the global ecological challenge
  - Avoiding that battery-limited devices are in charge too often ultimately fights against resource exhaustion
- The gains add up
  - Making an app more energy-friendly, even modestly, means saving energy on each device where it is installed
  - The impact is huge in the case of behemoth apps (Instagram, Facebook,…), but apps with a lower audience matter too

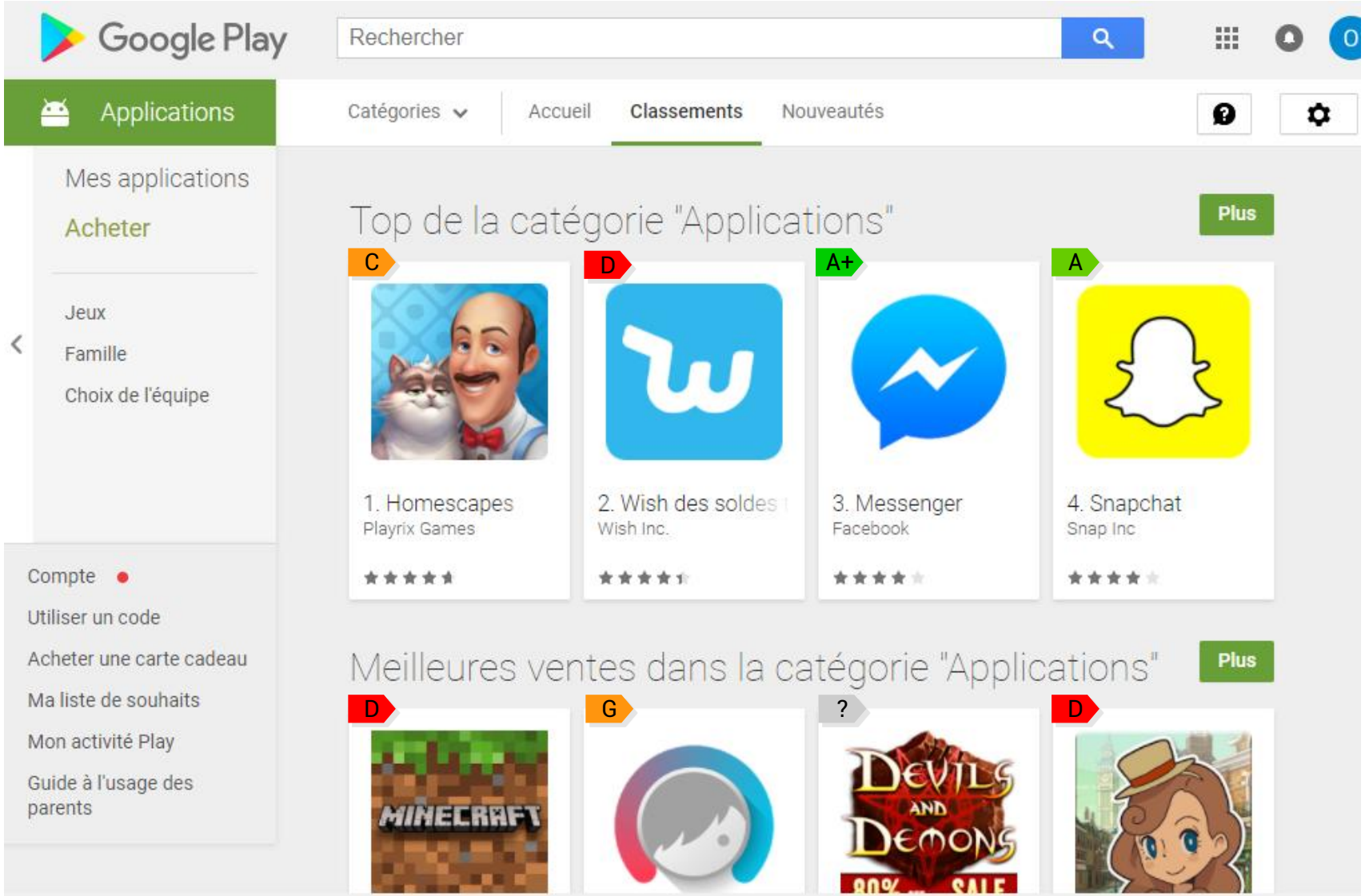
A negative impact on the User eXperience

- Blaming the app
  - « when I use this app, my phone's battery life goes down the tubes »
  - Leads to poor reviews and rating. Massive uninstalling.
- Blaming the device/OS
  - « I'm always having to charge this @#%@$*!!! Thing »
  - Leads to a depressed market of the device… and apps that support this device

Sources of power drain

- Screen
- CPU
- Radios (Wifi, mobile data, Bluetooth, etc)
- “Disk” I/O
- Sensors (accelerometer, GPS, camera, etc)

A label for Google Play (fictional)



One label to rule them all

- Websites became mobile-friendly as soon as Google announced that it would take them into account in its SEO. Mobile apps will become energy-friendly as soon as an energy label is displayed on Google Play.

- End-users already choose the apps they perceive as the most energy-efficient. The developers know this and want to satisfy this expectation. An energy label is just an acknowledgement of that.

State of commitment

- Device manufacturers continuously improve the efficiency of batteries
  - The futur of Lithium-ion: Lithium-air, Graphene Battery, …
- Mobile OS provide intelligent features for power management
  - Doze mode and App Standby since Android 6.0
  - Adaptive Battery since Android 9.0
- It still raises the question of how energy-intensive the apps themselves are…
  - Energy efficiency is a non-functional requirement (as is security for example)
  - Eco-responsible design of mobile apps implies writing green code

Sustainable Software Development

How to assess the green-ness of an app

Energy Diagnosis of an app

- Diagnosis at run-time (dynamic analysis)
  - Instrumented measure of the power consumption of a running app
  - A realistic (not monkey one) testing scenario must be tailored for each app, and played several times
  - What about scalability?

- **Diagnosis at design-time (static analysis)**
  - Evaluate if an app is « Green-by-design »
  - Flags potential energy bugs, regardless the nature of the app
  - Provided the source code is available, the diagnosis may apply automatically

Green-by-design: a bonus-malus system

ECOLOGICAL BONUS

- Demonstrates the developer's willingness to use the most energy-efficient APIs and his intent to adhere to some proven coding guidelines.

ECOLOGICAL MALUS

- Power-related code smells (flaws) reside on the source code. They may be the result of carelessness or lack of knowledge on the part of the developer.

Android Bonus #1

- NAME
  - Battery-efficient Location
- DESCRIPTION
  - Monitoring location changes is a very battery-intensive task when done in the regular way, while there exist optimized solutions
    - FusedLocationProvider API (Google)
    - HyperTrack SDK (Third-party)
- DIAGNOSIS
  - Check if these APIs have been imported (and used) into the project, instead of the classic one

Android Bonus #2

- NAME
  - Defering (Lazy First principle)
- DESCRIPTION
  - “Does an app need to perform an action right away? For example, can it wait until the device is charging before it backs data up to the cloud?” From here
- DIAGNOSIS
  - Check if the app has registered on the `ACTION_POWER_CONNECTED` broadcasted platform event in the purpose to do some stuff

Android Bonus #3

- NAME
  - Dark UI
- DESCRIPTION
  - Provide a UI with dark background colors. This is particularly beneficial for mobile devices with AMOLED screens, which are more energy efficient when displaying dark colors.
- DIAGNOSIS
  - Check if Activities are associated with `Theme.Holo.Dark` style (and its variants) or if layouts aren't using bright background colors

Android Bonus #4

- NAME
  - Sensors Coalesce
- DESCRIPTION
  - "An alternative function allows events to stay temporarily in the hardware FIFO (queue) before being delivered. The events can be stored in the hardware FIFO up to `maxReportLatencyUs` microseconds. [...] Setting `maxReportLatencyUs` to a positive value allows to reduce the number of interrupts the AP (Application Processor) receives, hence reducing power consumption, as the AP can switch to a lower power state while the sensor is capturing the data." From here
- DIAGNOSIS
  - Check the calls to the old method `registerListener (SensorEventListener listener, Sensor sensor, int samplingPeriodUs)`

Android Bonus #5

- NAME
  - Bluetooth Low Energy (BLE)
- DESCRIPTION
  - "In contrast to Classic Bluetooth, Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption". From here
- DIAGNOSIS
  - Check if the package `android.bluetooth.le` is imported instead of `android.bluetooth`

Android Malus #1

- NAME
  - Sensors Leak*
- PROBLEM
  - " Always make sure to disable sensors you don't need, especially when your activity is paused. Failing to do so can drain the battery in just a few hours. Note that the system will not disable sensors automatically when the screen turns off." from here
- DIAGNOSIS
  - Check if the calls to `registerListener()` and `unregisterListener()` on a sensor manager are pairwised and well-positioned

*This Malus encompasses the GPS leak (`requestLocationUpdates/removeUpdates`) and the Camera leak (`open/release`)

Android Malus #2

- NAME
  - Everlasting Service
- PROBLEM
  - The Service component is used for long-running operations. Any started service should be stopped properly
- DIAGNOSIS
  - Check if for the call to `startService()`, it exist either a call to `stopService()` or `stopSelf()` or `stopSelfResult()`

Android Malus #3

- NAME
  - Internet In The Loop
- PROBLEM
  - Performing a call to internet repeatedly (a.k.a pull method) requires a superfluous connectivity (WiFi or mobile data). Study here
- DIAGNOSIS
  - Check if instances of classes `org.apache.http.client.HttpClient` or `java.net.HttpURLConnection` are used inside a loop statement

Android Malus #4

- NAME
  - Wake Lock Plague
- DESCRIPTION
  - "To avoid draining the battery, an Android device that is left idle quickly falls asleep. However, there are times when an application needs to wake up the screen or the CPU and keep it awake to complete some work." from here.
- DIAGNOSIS
  - Check if the `android.permission.WAKE_LOCK` permission was declared in the manifest (easier than checking acquisition of locks via the `android.os.PowerManager` class)

Android Malus #5

- NAME
  - Excessive Logging
- DESCRIPTION
  - Developers resort to logging in their mobile apps to ensure their correct behavior and simplify bug reporting. However, logging operations are creating overhead on energy consumption without creating value to the end user. Study here
- DIAGNOSIS
  - Check if the number of calls to `android.util.Log` is greater than a threshold (depending of size of the program)

Green Linter

Enforcing green coding rules

Green-aware IDE

- Android Studio is the official Android IDE and developer tools for building apps on every type of Android device
  - Custom packaging of the JetBrains' IntelliJ IDEA
  - Used by at least 5,9 millions of developers (report, 2016)
- This world-class IDE should push ahead green software
  - Right place for resolving the green technical debt
  - General "these things are BAD" but also "these things are GOOD" recommendations
  - Quick fixes when possible
  - Diagnosis and reporting in several formats

Android Lint

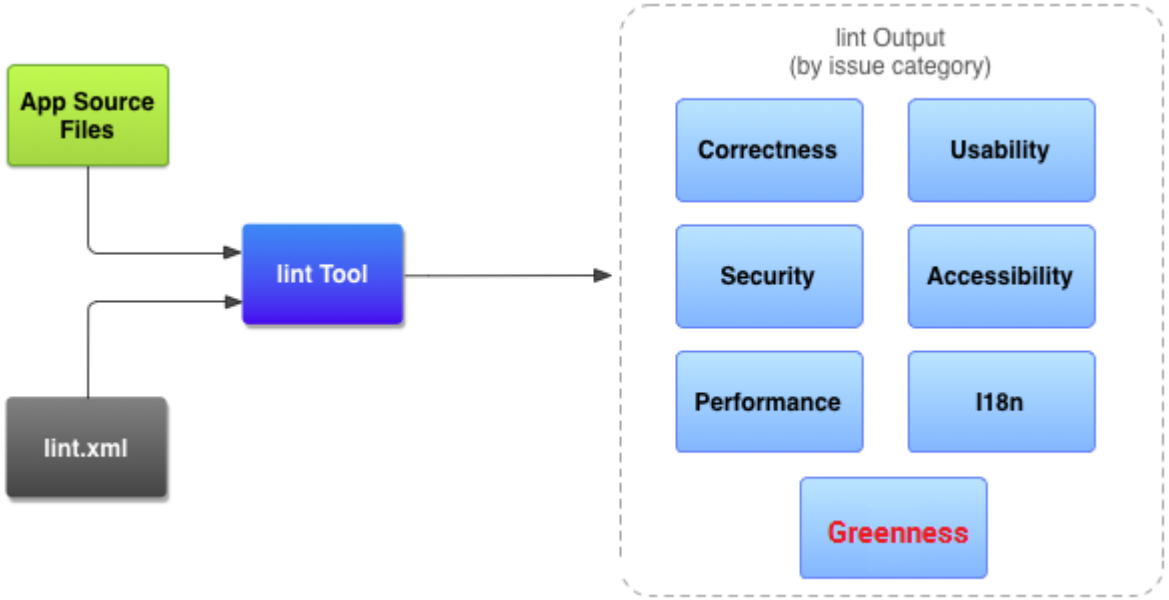
- The tool Android Lint is integrated into Android Studio
  - Android Lint has ~350 checks (list here)

- 3 checks built-in Android Lint identified as energy-related

Name	Description	Category	Source
ShortAlarm	Frequent alarms are bad for battery life. As of API 22, theAlarmManager will override near-future and high-frequency alarm requests, delaying the alarm at least 5 seconds into the future and ensuring that the repeat interval is at least 60 seconds.	Correctness	here
BatteryLife (Background optimizations)	This issue flags code that either * negatively affects battery life, or * uses APIs that have recently changed behavior to prevent background tasks from consuming memory and battery excessively. Generally, you should be using JobScheduler or GcmNetworkManager instead.	Correctness	here
Wakelock (incorrect usage)	Failing to release a wakelock properly can keep the Android device in a high power mode, which reduces battery life. There are several causes of this, such as releasing the wake lock in onDestroy() instead of in onPause(), failing to call release() in all possible code paths after an acquire(), and so on.	Performance	here

Extending Android Lint

- Android Tools Lint API is an extensible framework
- First create a new category **Greenness**
  - Bonus and malus become « issues »
  - Implement detectors for these issues (either in Java or Kotlin)



Under the hood of Android lint

- Android lint supports various scopes of analysis
  - Manifest, Android resources, Source code (.java, .kt files), Bytecode (.class files), Gradle files, ProGuard files, Property Files, Other files
- Since 2017, source code is modeled through the Universal AST API
  - UAST covers both Java and Kotlin
  - Augments the PSI API (Program Structure Interface), tied to the IntelliJ platform
  - The Lombok AST API was abandoned
- Quick fix works as a text replacement

Proof Of Concept

- Demo-time…
  - Sensors Leak
  - Sensors Coalesce (+Quick Fix)
  - Dark UI (+ Quick fix)
  - Bluetooth Low-Energy
  - Internet-In-The-Loop
  - Excessive Logging

Energy labels

How consumers will make informed decisions

Eco-score formula

- Every green checks (bonus/malus) is associated **empirically** with a weight (positive/negative)
  - Defering: +1
  - Everlasting Service: -3
  - …
- The raw score of a mobile app is then computed as follow:

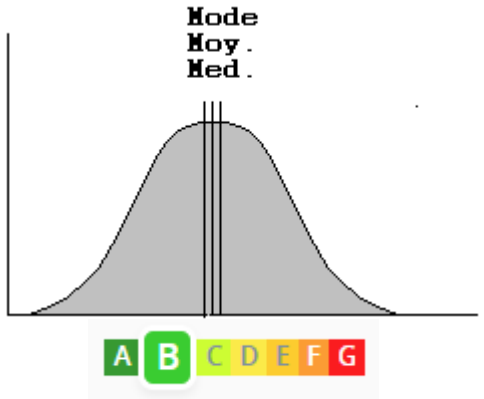
$$\sum_{i=1}^{C} N_i \, W_i$$

  - With
    - C, the total number of checks in the catalog
    - N_i, the occurrence of the checks
    - W_i, the weight of the checks

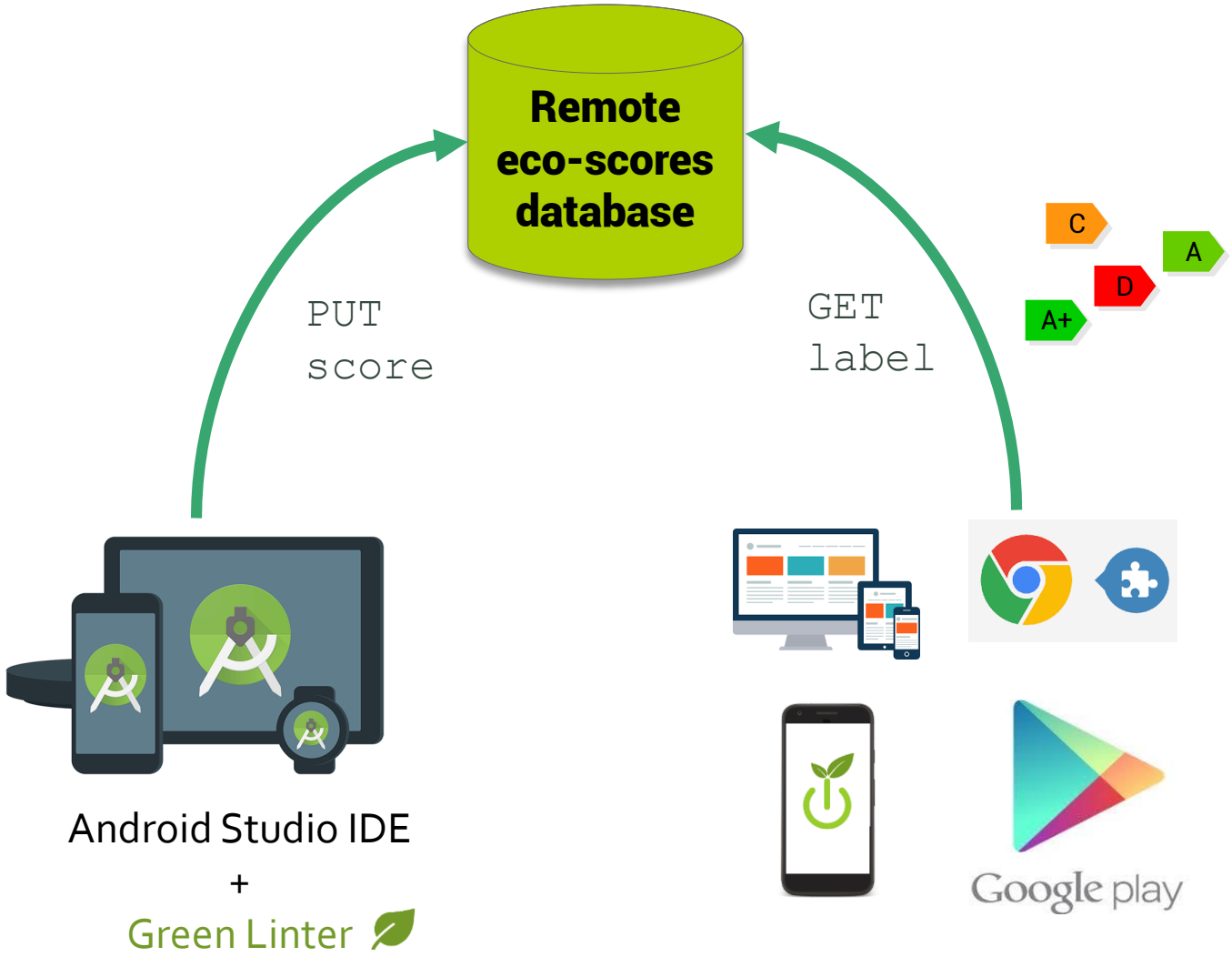
Eco-label

- The score must be mapped with a user-friendly label

- Observation of a statistical distribution
  - Get data from open source android projects first (list here)
  - Isolation of 6 classes (A, B, C, D, E ,F ,G)



End-to-end solution



Technical guidelines

- NoSQL Database => Key-value storage
  - `{"AppId" : { EcoScore, ApkChecksum }}`

- Displaying labels as close as possible to the end-users
  - Option 1: Create a dedicated website
    - https://www.green.app
  - Option 2: Create plugins for major Web browsers
    - Superimpose labels on top of https://play.google.com/store/apps website
  - Option 3 : Create the specific Android app « Green Pauware »
    - Once installed, it is triggered by any new app installation referenced in the database, and display the label as a notification
  - Option 4: Google Play queries our database
    - Maybe I'm dreaming there 😀

Challenges

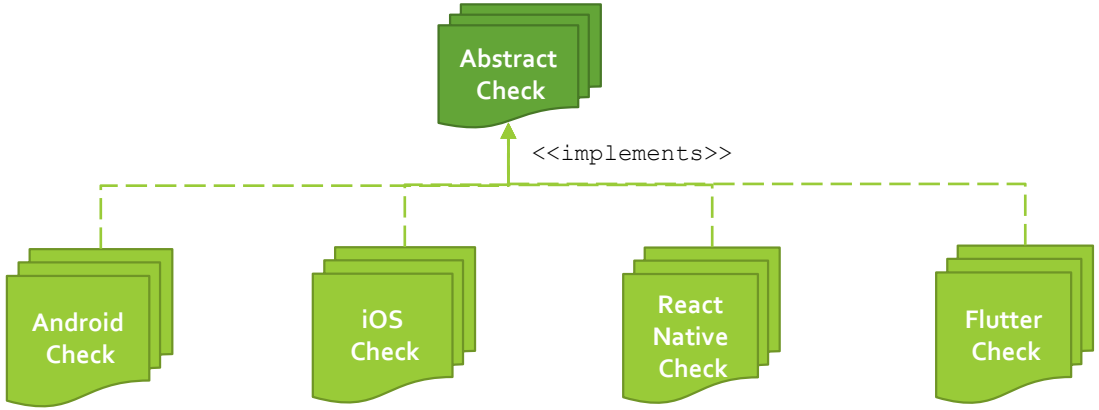
Research questions to be tackled

The art of writing an Android lint check

- Writing a custom lint rule, or wading through undocumented waters + instable API
- A limited power of expression
  - Many bonus/malus cannot be expressed formally (ex: caching data)
  - False-positives and false-negatives
- Visitor-style programming is tedious
  - Continuation-passing style
  - Lack of a query support
    - The Querying Helpers of Spoon API, Complex GRAL predicates of The Graph Repository Query Language (GReQL), etc.

Cross-platform linter?

- Fast-paced mobile technologies => fragmentation
  - Java, Kotlin, Objective C, Swift, Apache week, React Native, Native Script, Flutter…
- Yet, some checks are shared accross platforms & techs
  - Dark UI, Sensors Leak, …
- Abstracting away these linter's checks (with a pivot language)



Questions?

That's all folks