



QCon 全球软件开发大会
INTERNATIONAL SOFTWARE
DEVELOPMENT CONFERENCE

BEIJING 2017

用AI高效测试移动应用

杨峻峰



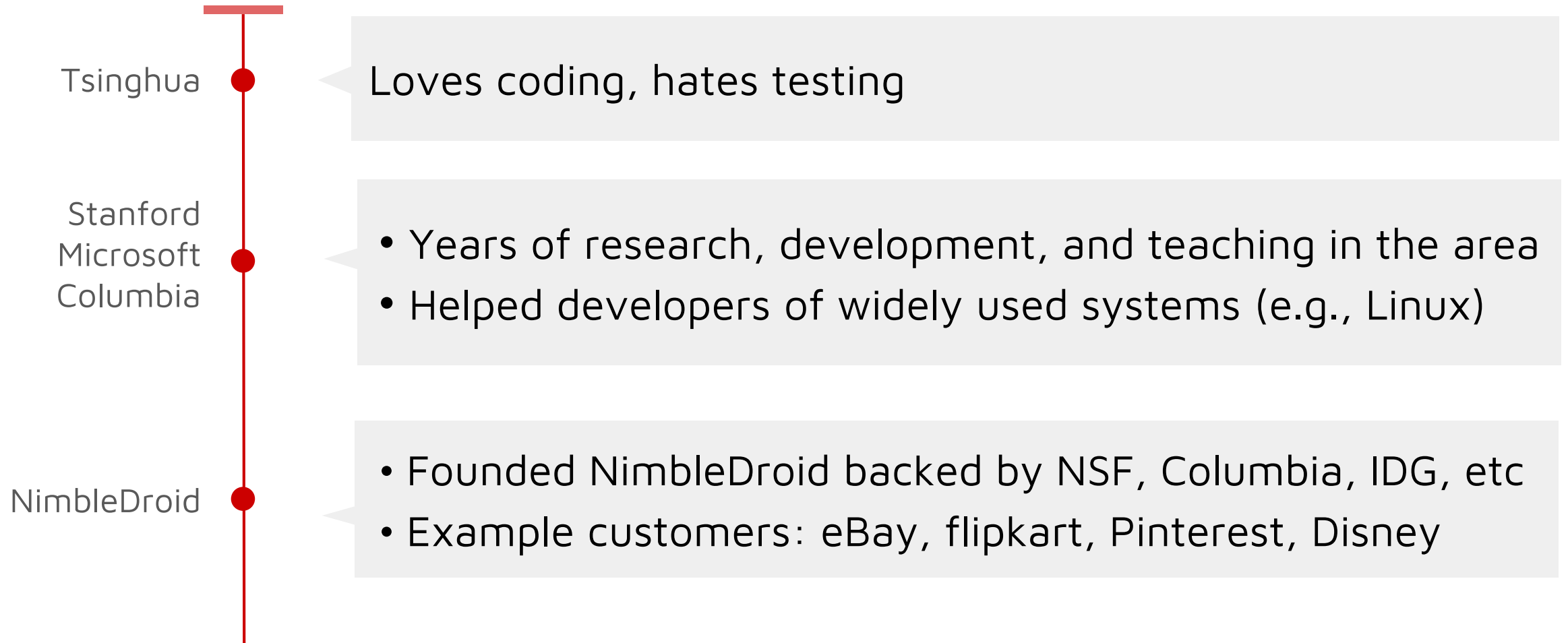
nimbledroid

Use AI to Effectively Test Mobile Apps

Junfeng Yang

<https://nimbledroid.com>

My Passion: Build Great Dev Tools



Disclaimer

This talk is based on experiences analyzing apps in overseas markets

May not be completely applicable to apps in Chinese markets

In fact, our service nimbledroid.com requires VPN to access from China

Can be helpful for app developers who target overseas markets

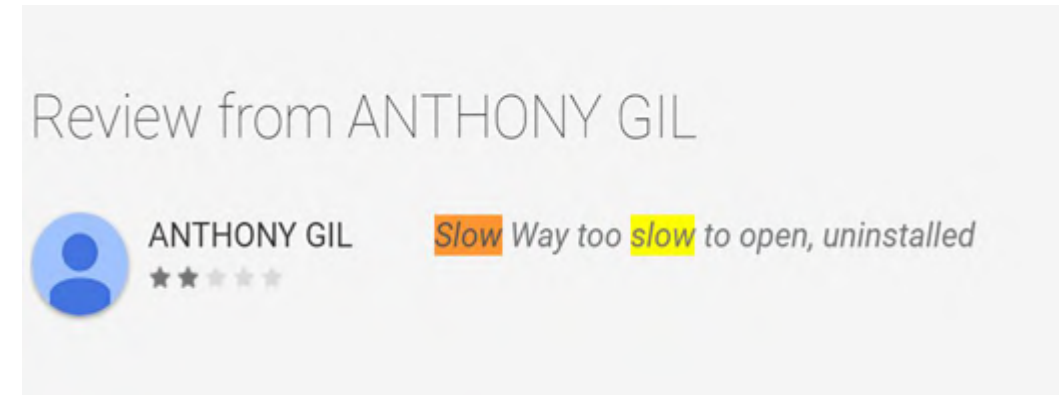
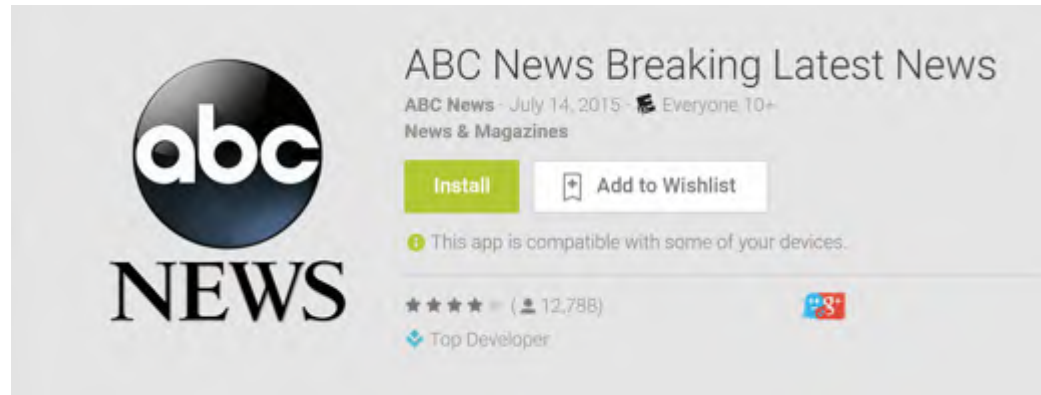
Two Competing Goals of App Development

App Quality

Development Speed

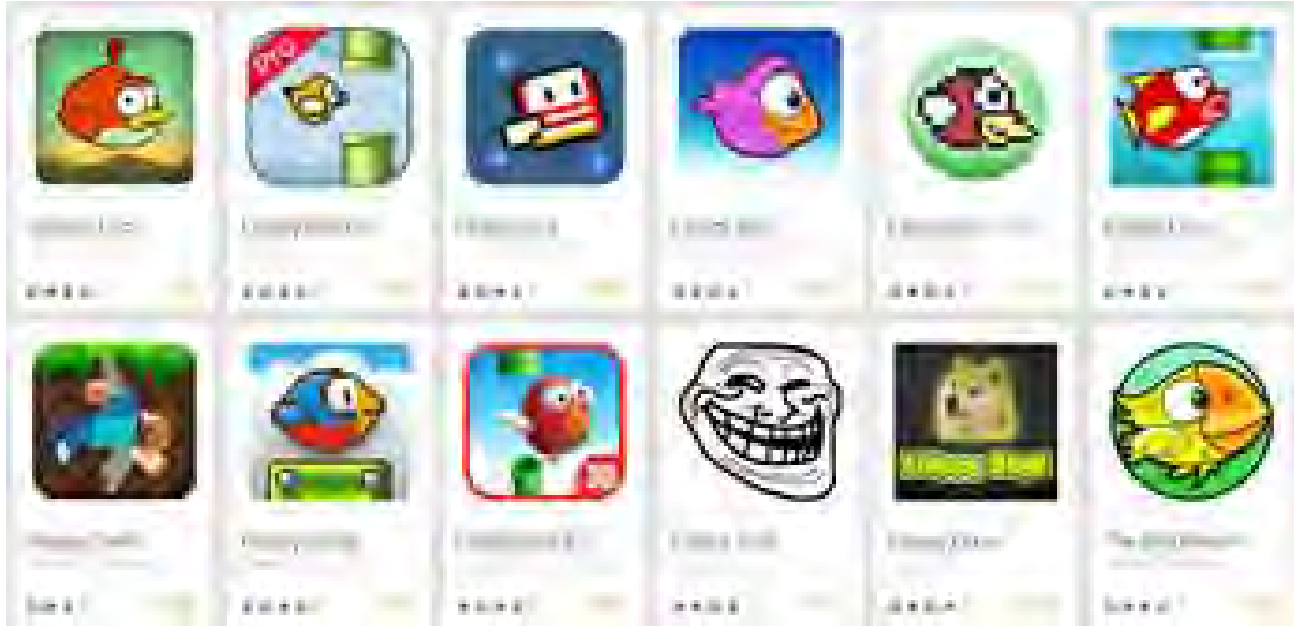


App Quality Matters



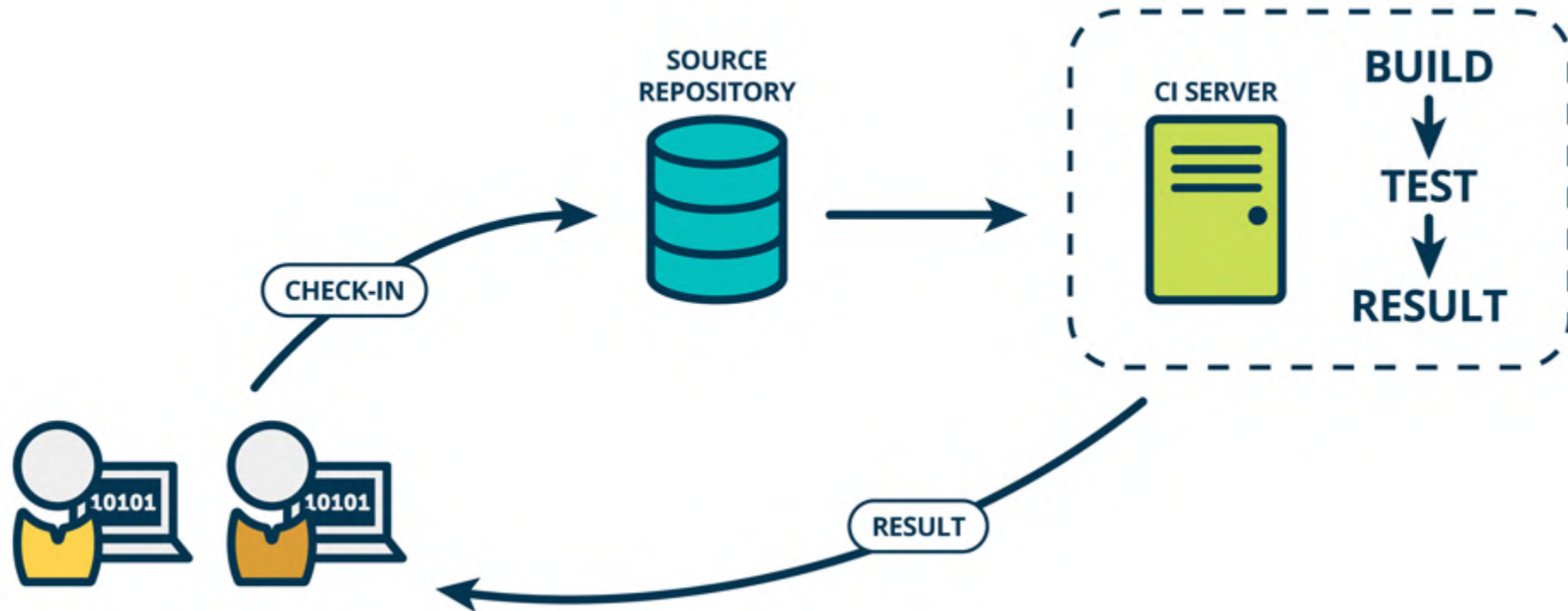
Issues like crashes, memory leaks, and slowdowns **decrease user engagement and revenue**

Development Speed Matters



User demands and market competitions change all too fast

Continuous Integration (CI) to the Rescue



CI Advantages

Better app quality

- A comprehensive CI testsuite continuously tests for errors
- Avoid building on the wrong foundation

Faster development speed

- Developers get instant feedback on code checkins, instead of forensically analyzing where errors creep in
- More errors detected during development ⇒ shorter code freeze or manual testing durations

But, CI Poses Big Challenges to Testing

Manual testing

- Slow and expensive, at odds with CI
- Imagine poor manual testers trying to keep up with 10+ checkins per day on 20+ devices with manual device rotation testing after each click

Automated testing

- Supposedly test automation is good for CI
- But, in practice, automated app UI test automation still requires much **babysitting** from developers

Today's Automation Needs Devs to **Babysit**

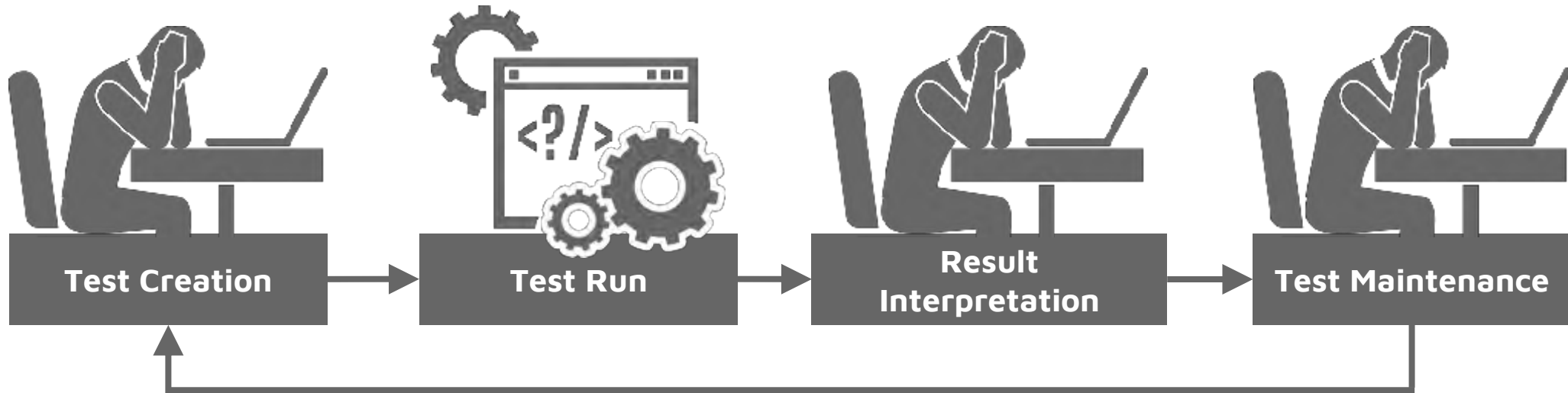
“ I have worked in several companies that have had goals of automated UI regression test suites, but I've never worked at a company that pulled it off successfully

”

atiffany @HackerNews

Huge pain point echoed by 100+ devs we've interviewed

Why Are Developers Forced to Spend So Much Time Here?



Have to create **low-level**, click-by-click tests

Test failures are often caused by tests themselves or test infrastructure, **not** bugs in code

Have to update tests whenever they become **outdated** to app UI

Today's Automation is **Not Smart** (1)

Lacks human vision, language understanding, context, judgement

Example test: "adding to shopping cart works"

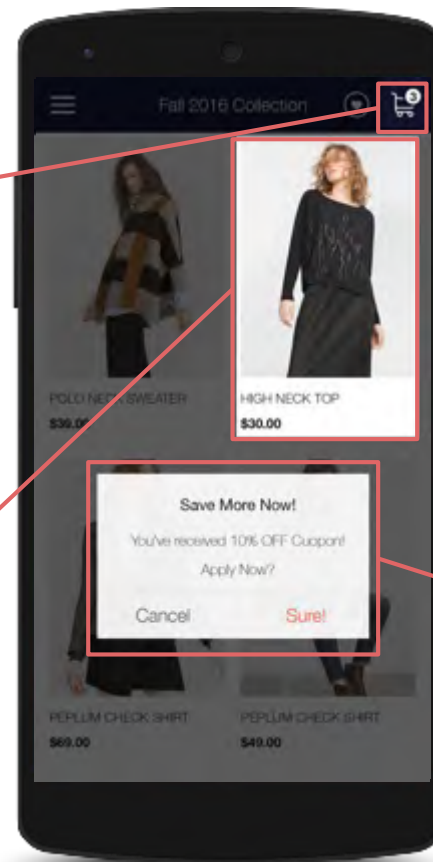


Human vision

recognize shopping cart image



Language understanding recognize items for sale and prices



Context

shopping app, need to test

- items added to cart are there
- items not added are not there
- ...



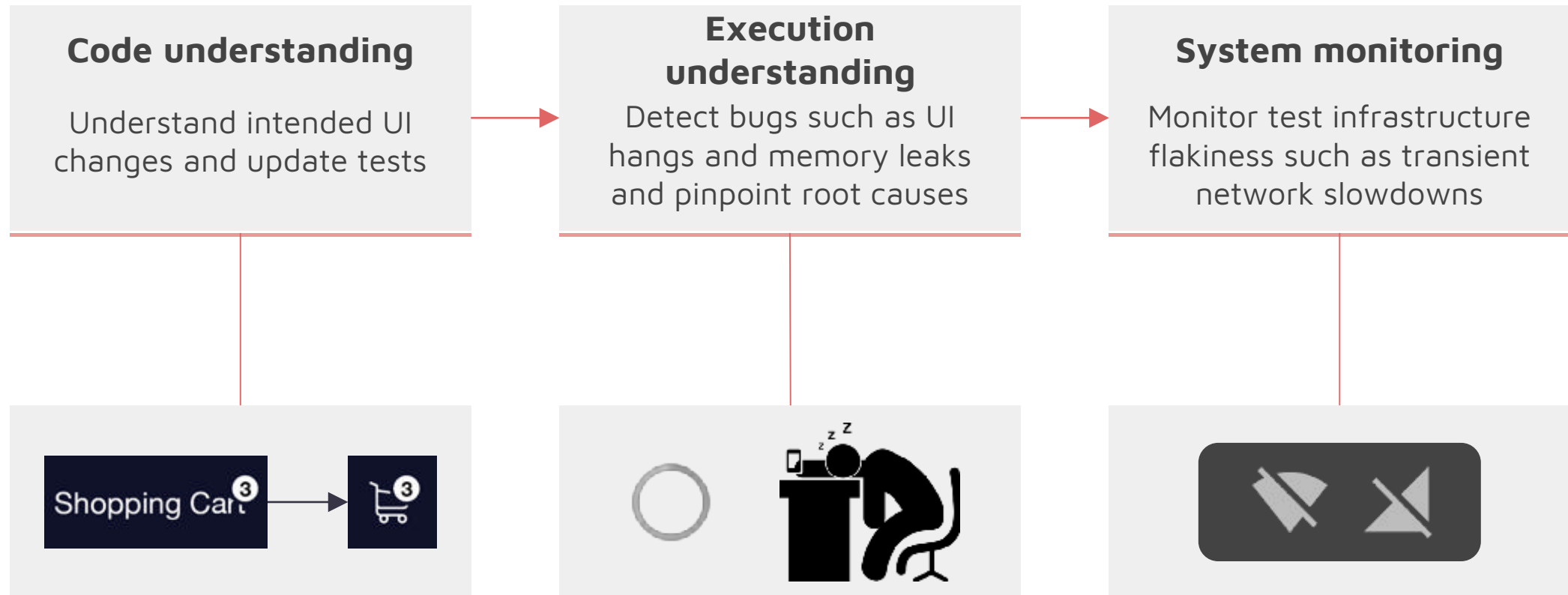
Judgement

this popup ad should be bypassed without failing the test



Today's Automation is **Not Smart** (2)

Lacks app code and execution understanding, system monitoring



Our Vision: AI for App QA

If we take:

Advanced program analysis including

- App instrumentation and monitoring
- Automatic error detection

Advanced AI and machine learning including

- Natural language processing
- Computer vision / image recognition

Q: How **intelligent** can we make an automated QA platform?

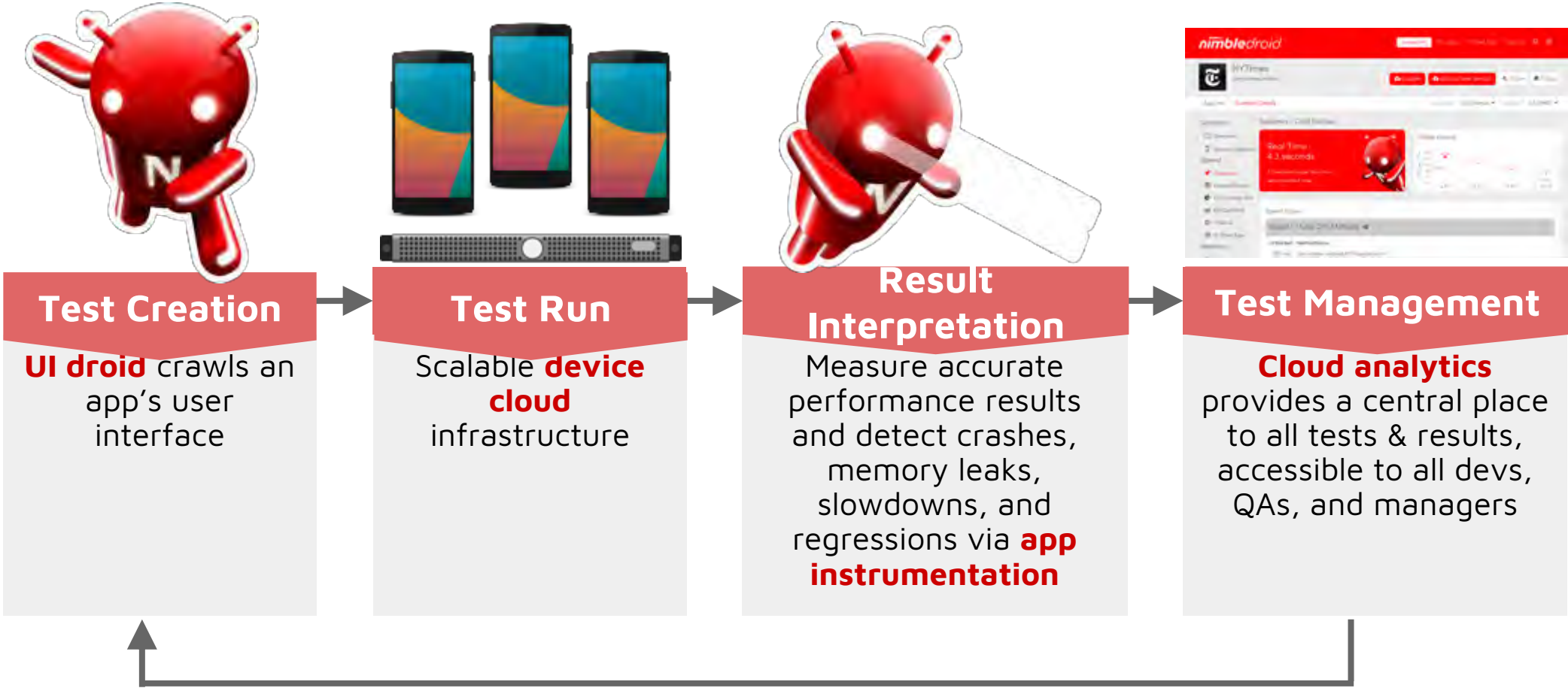
A: **Very.**

Outline

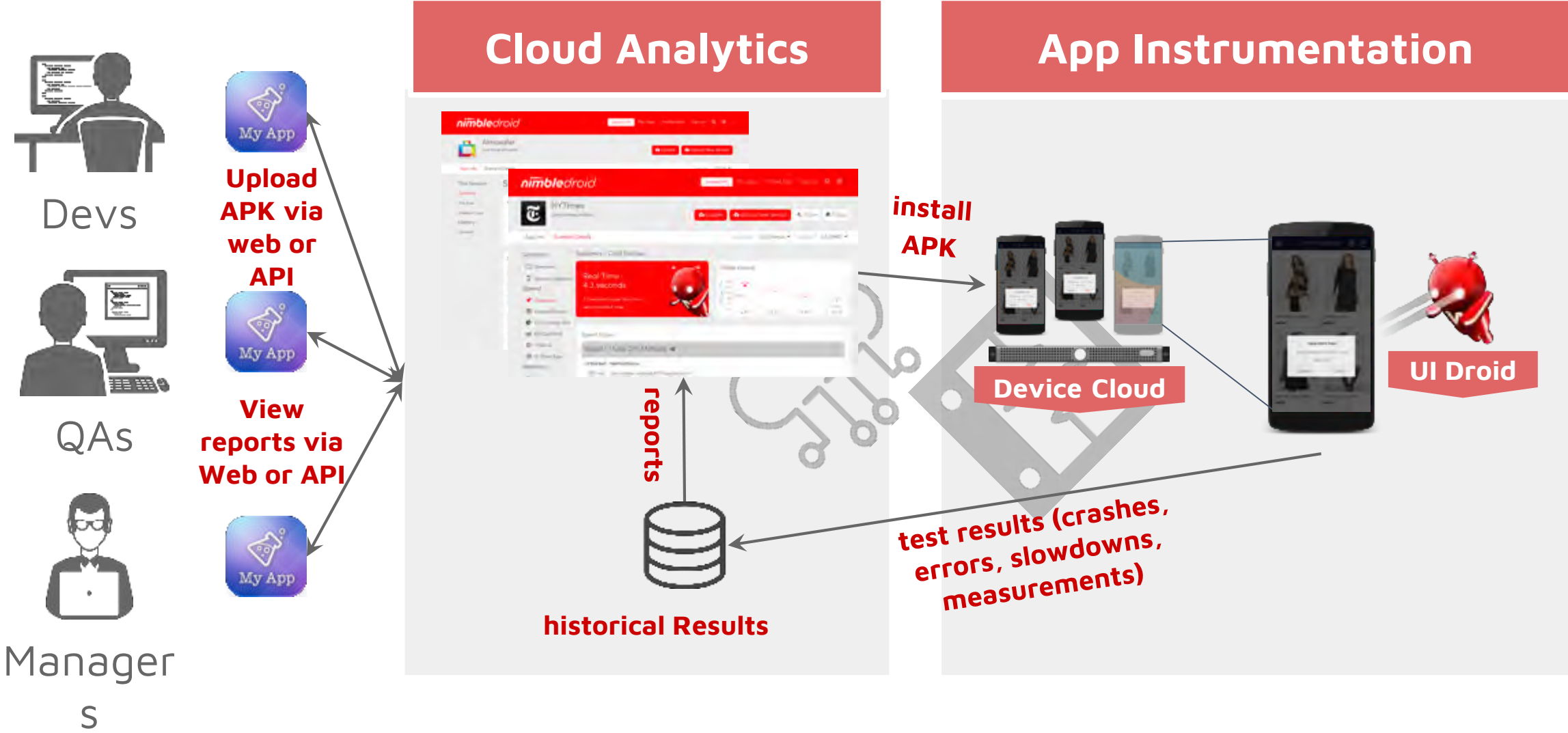
- Vision: AI for app QA
- ➔ • Current system we've built
- Zoom in: top issues slowing down your app

Intelligent Mobile App QA

“ *Auto-test every build of your Android app for critical issues* ”



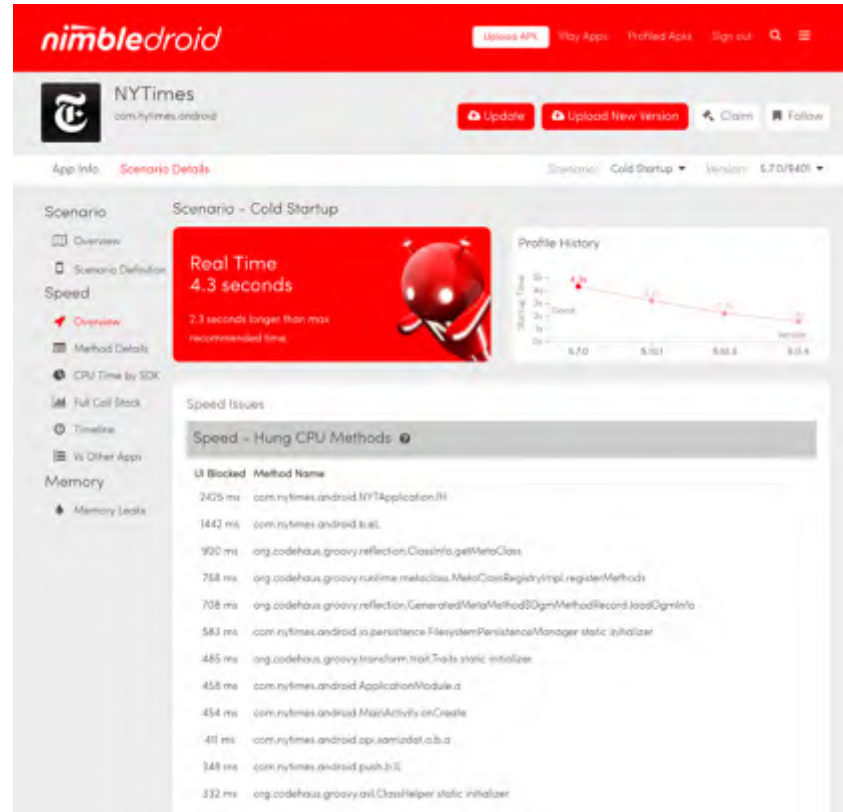
System Architecture



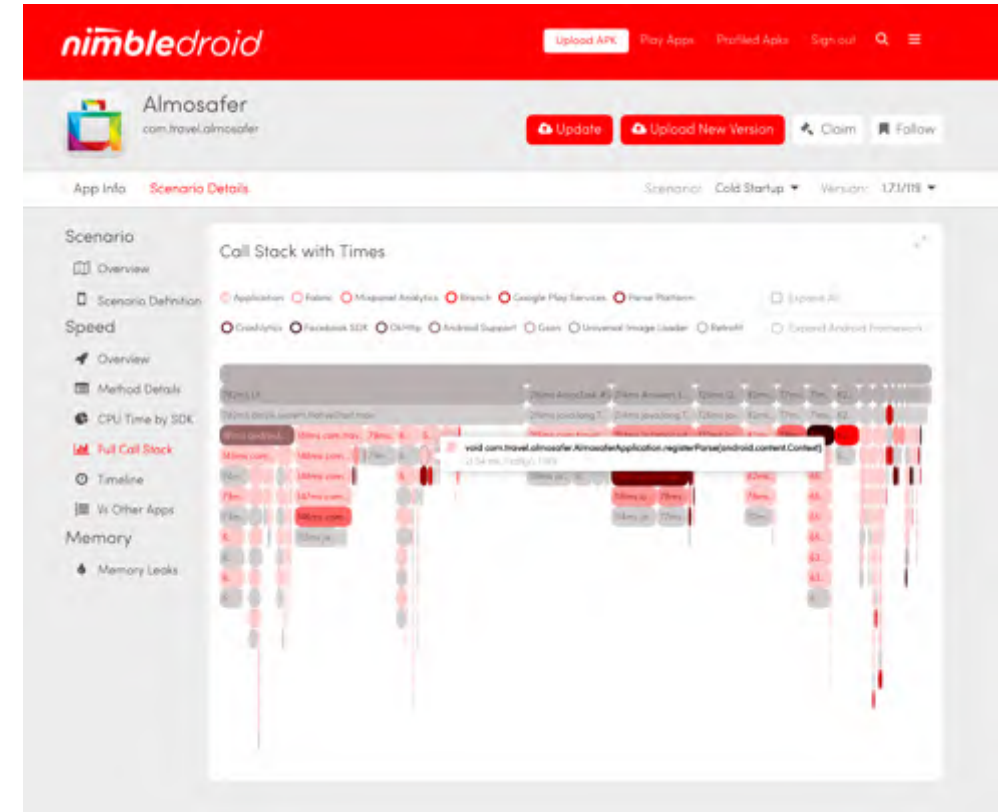
Demo



1. Multiple scenarios crawled by UI droid



2. Scenario overview with trending analytics



3. Call stack with times for detailed diagnosis

User Quotes



Sr Mobile Engineering Manager, Yahoo

"Performance tuning is difficult, tedious and time consuming. NimbleDroid makes the task suck infinitely less."



Mobile Practice Lead, Abercrombie & Fitch

"I would highly recommend nimbledroid for anyone looking to improve startup performance of their Android application."



Software Engineer, Azimo

"... NimbleDroid makes the task easier and more pleasant."



Android Software Architect, New York Times

"By far my favorite androiddev tool in a long time"



VP of Product and Engineering, RunKeeper

"Automatic discovery of user flows was so good we thought it was humans doing it via Mechanical Turk!"

Community Recognition

[NYTimes dev blog post](#) on how they leveraged NimbleDroid to make their Android app start 3x faster

Top result of Google search "[sdk slowing app down](#)"

National Science Foundation [SBIR Award](#)

Invited to run **App Garage Performance Clinic** at Droidcon London

Invited **talks on Android performance** at Droidcon and AnDevCon

90% of our tech blog posts were **featured by Android Weekly**, the largest Android development newsletter

Outline

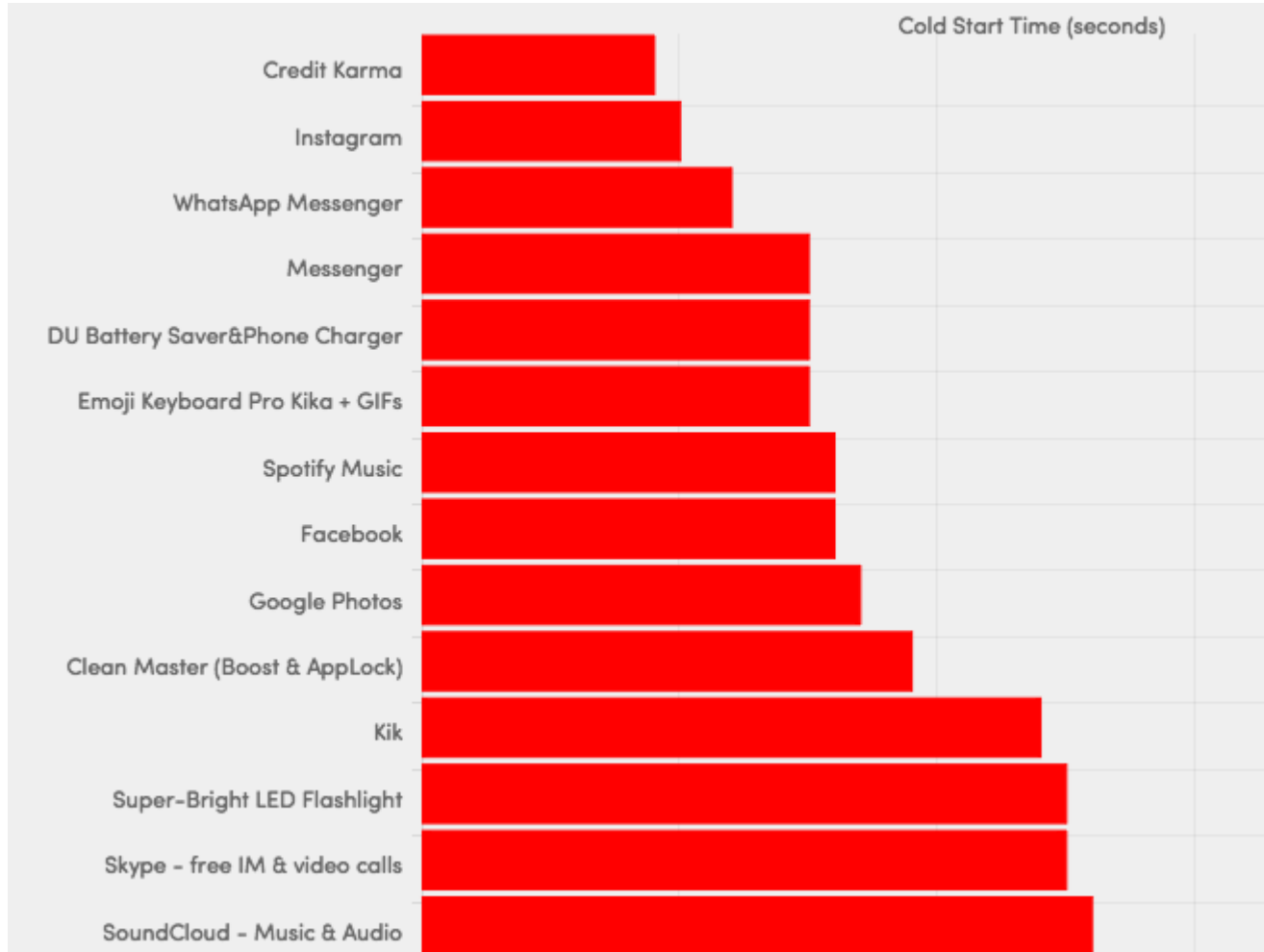
- Vision: AI for app QA
- Current system we've built
- ➔ • Zoom in: top issues slowing down your app

Three Types of App Starts



- **Cold start:** after user hasn't run app for a while
 - App killed by Android
 - Need to load app's code and assets, create activities, etc
- **First start:** fresh after installation
 - App and Android do more than cold start, such as db init, cache init, DEX compilation (if multidex), letting users input credentials
- **Warm start:** shortly after user switches away from app
 - App still cached in memory

Limit Cold Start to 2 Seconds



30-40% top apps start in 2s
70% start in 3s

Top 5 Issues that Slow Down App Start

- ➔ • Reflection
- Dependency injection
- Too much work in main thread
- `ClassLoader.getResourceAsStream()` and the like
- Slow 3rd party SDKs

Reflection Micro-benchmark

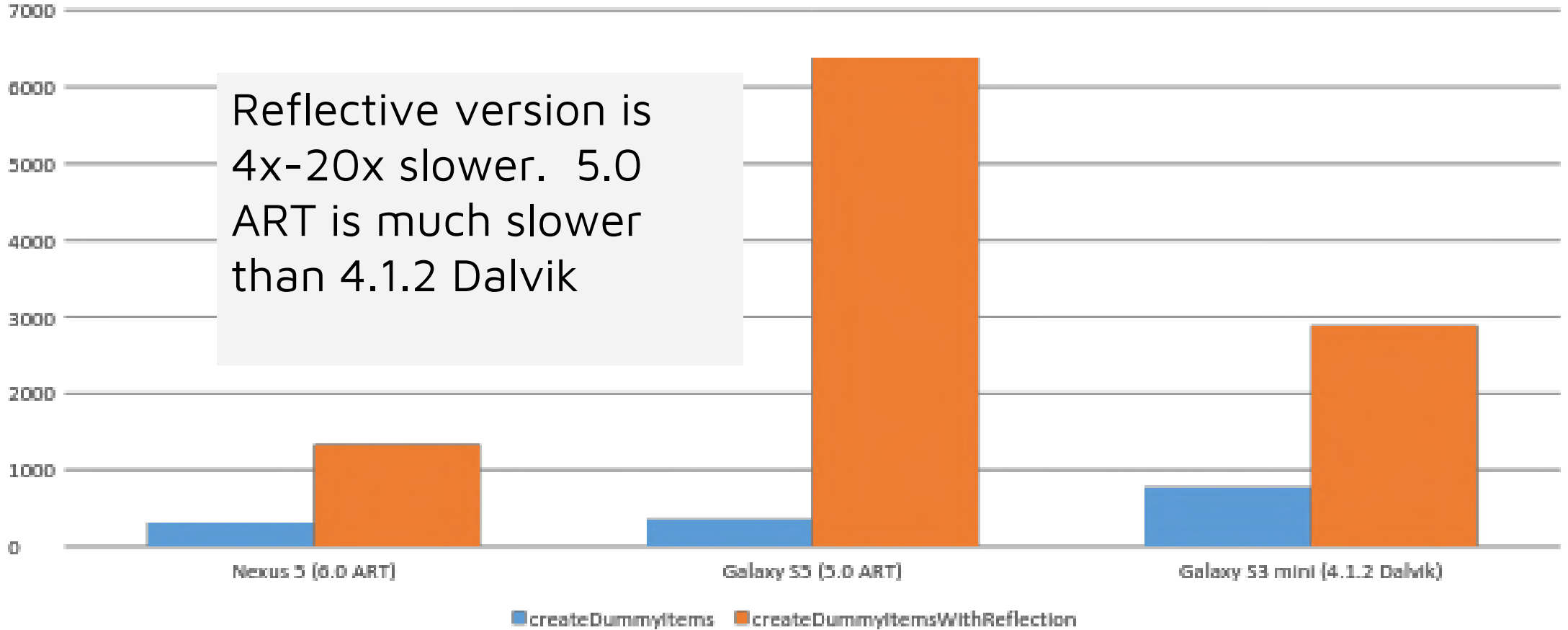
```
public static class DummyItem {}

private void createDummyItem() {
    for(int i = 0; i < 1_000_000; i++)
        new DummyItem();
}

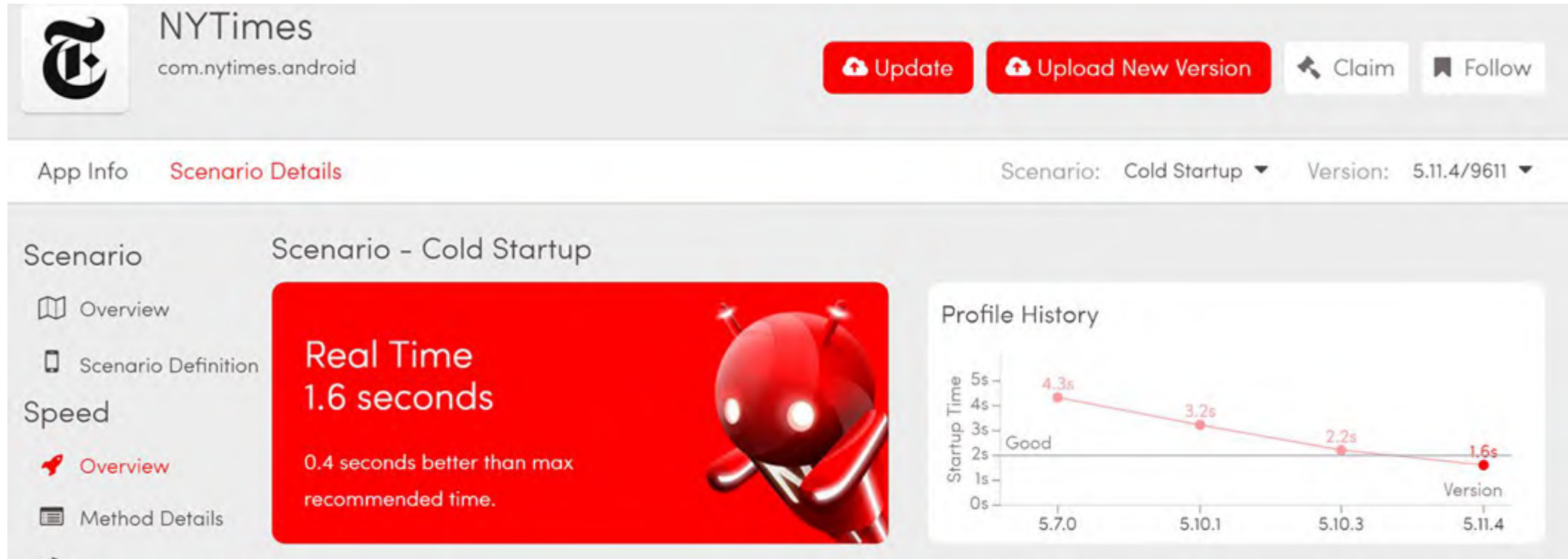
private void createDummyItemWithReflection() {
    for(int i = 0; i < 1_000_000; i++)
        DummyItem.class.newInstance();
}
```

<https://github.com/NimbleDroid/ReflectionTests>

Reflection Micro-benchmark Results (ms)

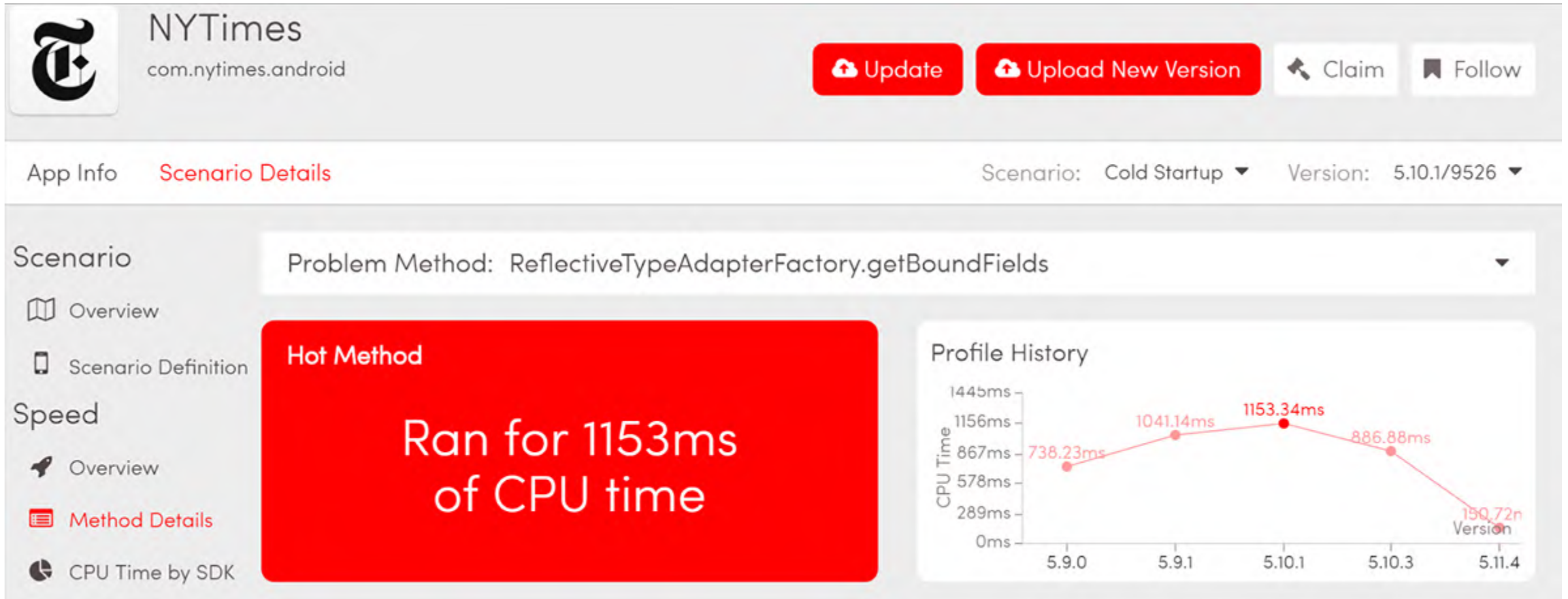


Example: NYTimes



- <http://open.blogs.nytimes.com/2016/02/11/improving-startup-time-in-the-nytimes-android-app/>

One Performance Issue: Reflection



- NYTimes switched to custom type adapters
- Use Immutables to keep developer overhead to a minimum

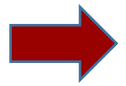
Reducing reflection overhead

- Be extra careful with reflection in your Android app
- Avoid reflective type adapters when dealing with many objects

<http://blog.nimbleandroid.com/2016/02/23/slow-Android-reflection.html>

Top 5 issues that slow down app start

- Reflection




- Dependency injection

- Too much work in main thread

- `ClassLoader.getResourceAsStream()` and the like

- Slow 3rd party SDKs

Dependency injection

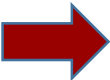
- Two possible approaches: *dynamic* (RoboGuice) vs *static* (Dagger1&2)
- Dynamic approach slows down app startup significantly!
 - RoboGuice is 5x or more slower on our micro-benchmark
 -  ↓ 0.8s, new version switched to dagger
 - American Express 1.7s (switched to dagger), Groupon 0.8s, Fandango 3.4s, ...
- One more thing: RoboGuice has 10,000 more methods than Dagger

Reducing dependency injection overhead

- Use Dagger libraries for dependency injection
- Use Dagger 2 if you can because it is slightly better than Dagger 1

<http://blog.nimbleandroid.com/2016/03/07/performance-of-dependency-injection-libraries.html>

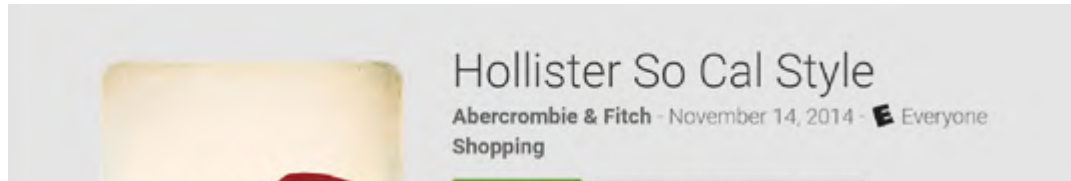
Top 5 issues that slow down app start

- Reflection
- Dependency injection
-  • Too much work in main thread
- `ClassLoader.getResourceAsStream()` and the like
- Slow 3rd party SDKs

Too much work in main thread

- Main thread responds to user actions
- Too much work → slow, janky
- Android's Strict Mode catches network, storage, and db accesses
- We've also seen data parsing, crypto, class initialization, ...

Example: Hollister



500,000 - 1,000,000 installs

Version 3.1.2

Startup: **5.22 seconds**

Review from Jennifer Brundage



Jennifer Brundage



SLOW It never loads. Two articles of clothing will show up. I leave my phone for an hour still not loaded.

- Hollister spends **~2600ms** to parse nationwide store info:

stores_af.json - 188 KB

stores_hco.json - 419 KB

stores_kids.json - 108 KB

Hollister 3.1.2 startup: 5.22s

The screenshot shows the NimbleDroid web interface for the app 'Hollister So Cal Style'. The browser address bar shows the URL: <https://www.nimbleandroid.com/play/com.abercrombie.hollister?p=24MpWQo0PFfe7cX#Summary>. The app's icon is a red bird, and the developer is 'Abercrombie & Fitch'. A red button labeled 'Upload New Version' is visible. The analysis scenario is 'Cold Startup' and the version is '3.1.2/90'. A red callout box highlights the startup time of 5.22 seconds, noting it is 3.22 seconds longer than the recommended time. A 'Profile History' chart shows a decrease in startup time from 5.22s at version 3.1.2 to 2.21s at version 4.0.0. The 'Analysis Info' section includes links for Summary, Dependencies, and Vs Other Apps. The 'Problem Methods' section is also visible.

Scenario: Cold Startup ▾ Version: 3.1.2/90 ▾ Claim this app Follow this app

Analysis Info

- Summary
- Dependencies
- Vs Other Apps

Problem Methods

Startup Time
5.22 seconds

3.22 seconds longer than max recommended time.

Profile History

Version	Startup Time
3.1.2	5.22s
4.0.0	2.21s

Why so slow?

Analysis Info

- Summary
- Dependencies
- Vs Other Apps

Problem Methods

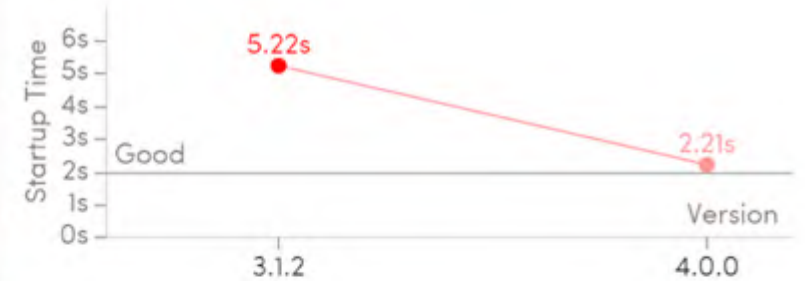
- Thread.parkFor()
- AFSDK.prefetchData()
- CollectionDeserializer.deserialize()
- ObjectMapper._findRootDeserializer()
- MobileConfig.loadConfig()
- BeanDeserializerBase.resolve()
- AsyncHttpClient.getDefaultScheme()
- Options.loadFromProperties()
- BeanDeserializerFactory.addBean

Startup Time
5.22 seconds

3.22 seconds longer than max recommended time.



Profile History



Hung methods make the UI lag and become unresponsive to the user for longer than 32ms. Methods reported here are doing so because they are using the CPU during this time.

Hung Methods, CPU: 11 issues found

com.abercrombie.android.sdk.AFSDK.prefetchData()	5709 ms
com.fasterxml.jackson.databind.deser.std.CollectionDeserializer.deserialize()	4987 ms
com.fasterxml.jackson.databind.ObjectMapper._findRootDeserializer()	682 ms
com.adobe.mobile.MobileConfig.loadConfig()	555 ms
com.fasterxml.jackson.databind.deser.BeanDeserializerBase.resolve()	477 ms

Hollister 4.0.0 startup: 2.21s

The screenshot displays the NimbleDroid website interface for the app 'Hollister So Cal Style'. The page shows the app's details, including the developer 'Abercrombie & Fitch' and the version '4.0.0/40000'. The startup time is highlighted as 2.21 seconds, which is 0.21 seconds longer than the maximum recommended time. A line graph titled 'Profile History' shows the startup time decreasing from 5.22s at version 3.1.2 to 2.21s at version 4.0.0. The graph also indicates a 'Good' performance level for the current version.

Startup Time
2.21 seconds

0.21 seconds longer than max recommended time.

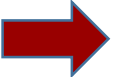
Profile History

Version	Startup Time
3.1.2	5.22s
4.0.0	2.21s

Reducing work in main thread




- Move work to background
- Write more efficient code

Top 5 issues that slow down app start

- Reflection
- Dependency injection
- Too much work in main thread
-  `ClassLoader.getResourceAsStream()` and the like
- Slow 3rd party SDKs

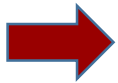
ClassLoader.getResourceAsStream()

```
public class DemoApplication extends Application {
    @Override
    public void onCreate () {
        super.onCreate();
        Properties properties = new Properties();
        try {
            properties.load(getClass().getResourceAsStream("/assets/test1.properties"));
        } catch (IOException e) { ...
        }
    }
}
```

- ~7ms on PC but **~1700 ms** on Android (with 3K resource files!)
 - Extra work done in first call in Android: index all resources in APK, verify certificate, parse manifest file. Delay is proportional to APK size
-  ↓0.4s  ↓1.3s  ↓1.7s
- Avoid `ClassLoader.getResource*()`; use Android's Resources

Top 5 issues that slow down app start

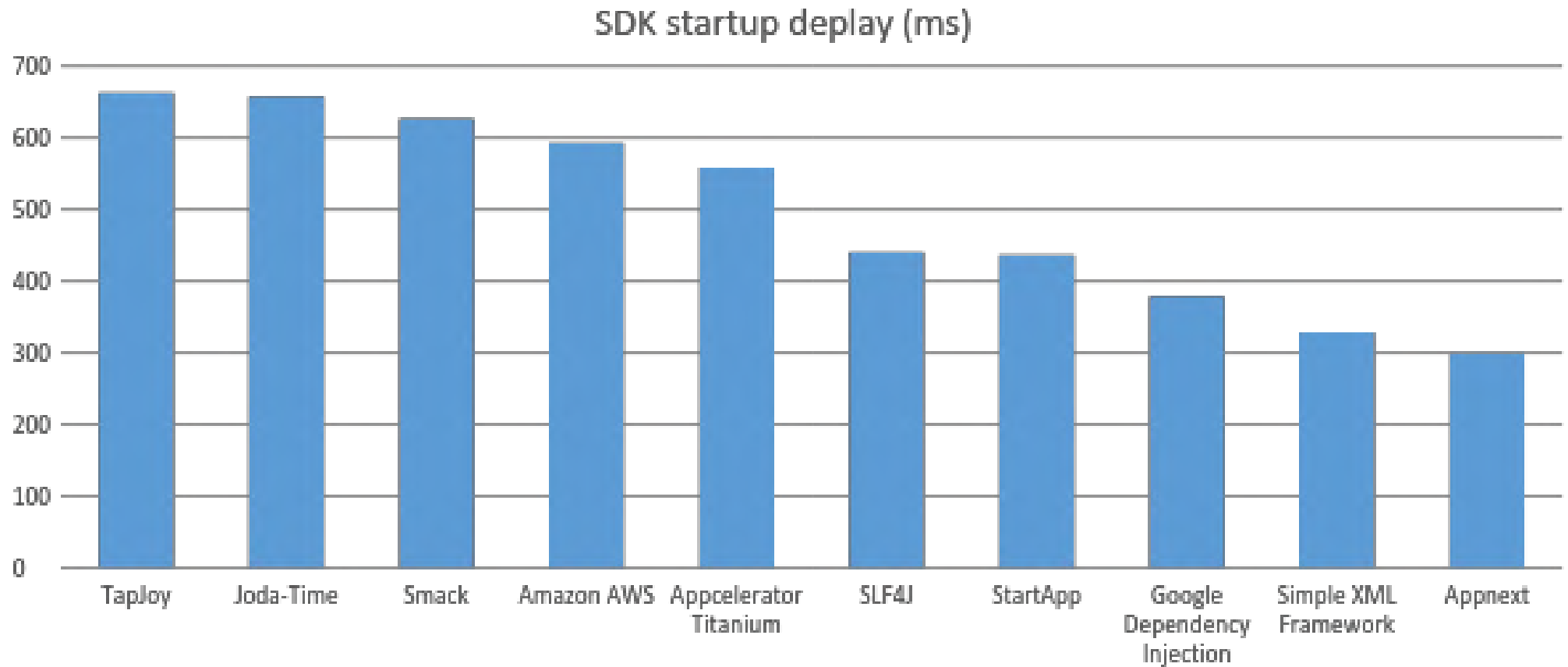
- Reflection
- Dependency injection
- Too much work in main thread
- `ClassLoader.getResourceAsStream()` and the like
- Slow 3rd party SDKs



Slow 3rd party SDKs

- They can slow down your app
- Hard to track down because code isn't written by you
- Not your fault, but you have to work around it 😞


10 slowest SDKs based on app startup delay



org.joda.time.DateTime()

```
import org.joda.time.DateTime;
import android.app.Application;

public class DemoApplication extends Application {
    static {
        new DateTime();
    }
}
```

- Even for this simple app, ↓0.4s
-  ↓2s
- Culprit: DateTime() calls getResourceAsStream() to load time zone data from APK
- Create your own fast DateTimeZoneProvider! See [stackoverflow](#)
- Or use date4j or joda-time-android

SDKs that call `ClassLoader.getResource*()`

SDK	# apps affected
mobileCore	251
SLF4J	70
StartApp	67
Joda-Time	55
TapJoy	52
Google Dependency Injection	46
BugSense	41
RoboGuice	35
OrmLite	26
...	...

**20% SDKs call
`getResource*()`**

Apps affected by SDK getResources*()

Affected apps	Downloads
com.amazon.kindle	> 100,000,000
tunein.player	> 100,000,000
com.sgiggle.production	> 100,000,000
com.mobilemotion.dubsmash	> 50,000,000
com.sirma.mobile.bible.android	> 50,000,000
com.melodis.midomiMusicIdentifier.freemium	> 50,000,000
com.badoo.mobile	> 50,000,000
com.outfit7.talkingangela.free	> 50,000,000
com.loudtalks	> 50,000,000
com.quvideo.xiaoying	> 50,000,000
...	...

**11% apps
affected**

Conclusions

- Vision: AI for Software QA
 - Competing app development goals: speed vs quality
 - CI helps, but raises challenges for test automation
 - AI can be the rescure
- Our current system
 - Intelligent crawler + app instrumentation
- Zoom in: top issues slowing down your app
 - Reflection, dependency injection, too much work in main thread, `ClassLoader.getResourceAsStream()` and the like, and slow 3rd-party SDKs



Test your app now at
<https://nimbledroid.com>



关注QCon微信公众号，
获得更多干货！

Thanks!



主办方 **Geekbang** > **InfoQ**
极客邦科技