

Build Fluid Apps with Android Profiling Tools

Junfeng Yang



*nimble*droid

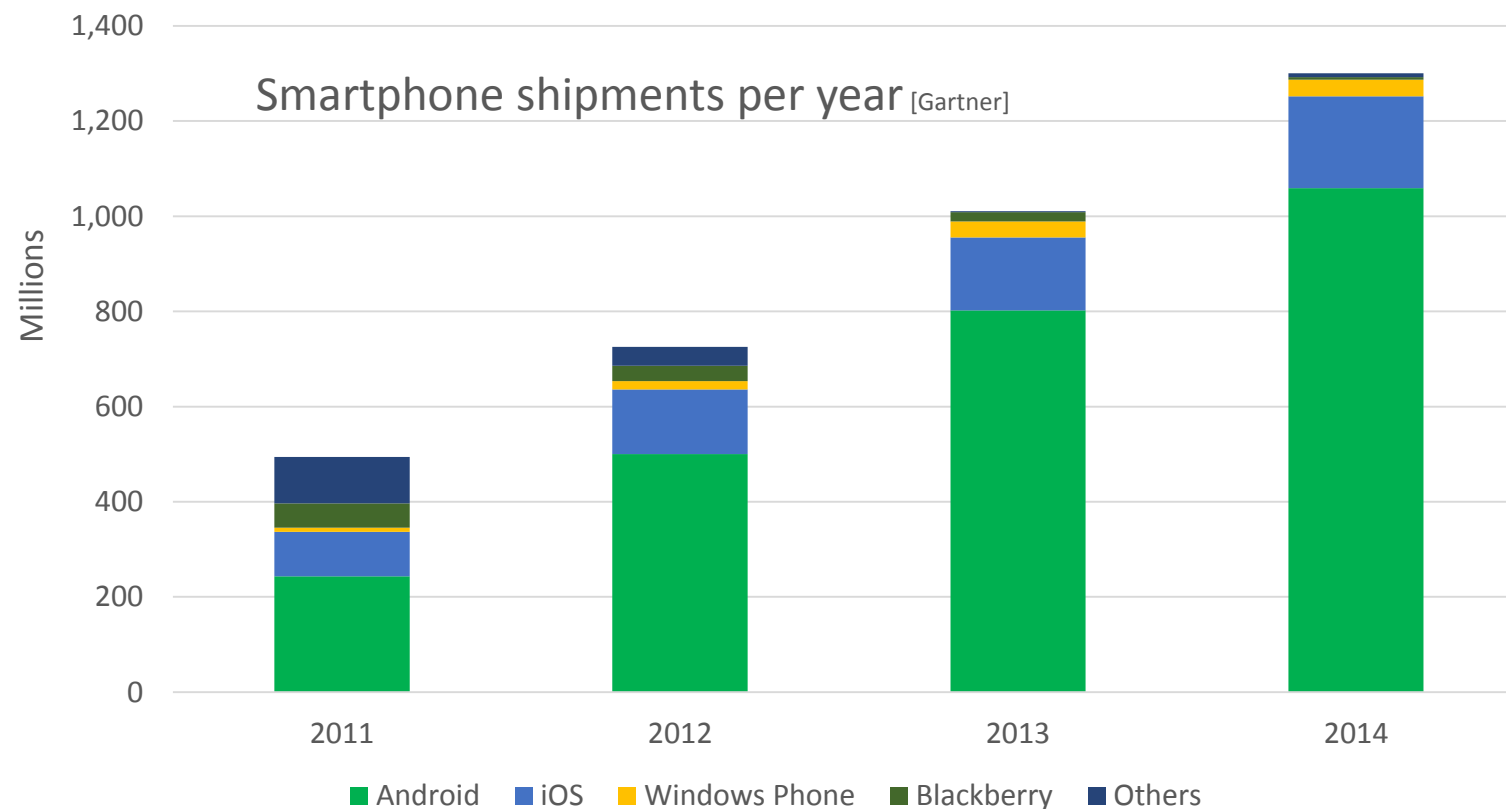


About your speaker

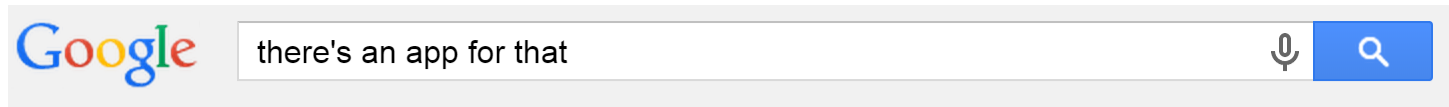
- Two jobs: Columbia Professor; NimbleDroid co-Founder, CEO
- **One passion:** building great developer tools
 - 15 years of research, development, and teaching in the area
 - Helped developers of widely used systems (e.g., Linux, VMWare, Microsoft)
- Current focus: help developers craft awesome mobile apps
 - In particular, improve **app performance**

Why you should care about
your app's performance

Mobile/Android is pervasive



Apps are eating the world



Web Images Shopping Videos News More ▾ Search tools

Page 2 of about 326,000,000 results (0.49 seconds)

Diagnosing cancer? There's an app for that
<https://www.bostonglobe.com/...there-app-for-that/.../s>
Jul 2, 2015 - Technology could soon bring us medicine-on-

Gotta go pee during a movie? There's an app for that
www.seattletimes.com/.../gotta-go-pee-during-a-movie-
2 days ago - The RunPee **app** tells moviegoers when it's safe

Kids and Summer Chores: There's an App For That - ABC News
abcnews.go.com › Technology

Jun 25, 2015 - Gregg Murset, whose kids range in age from 8 to 17, said his idea for the My Job Chart **app** came four years ago from the "craziness" of making ...

There's an app for that? - TechRadar

www.techradar.com/.../there-s-an-app-for-that-10-oddest-apps... ▾ TechRadar ▾
Sep 29, 2013 - **There** are applications that turn your device into a beautiful musical instrument, a living book or a productivity powerhouse, and **there** are **apps** ...

Forbes

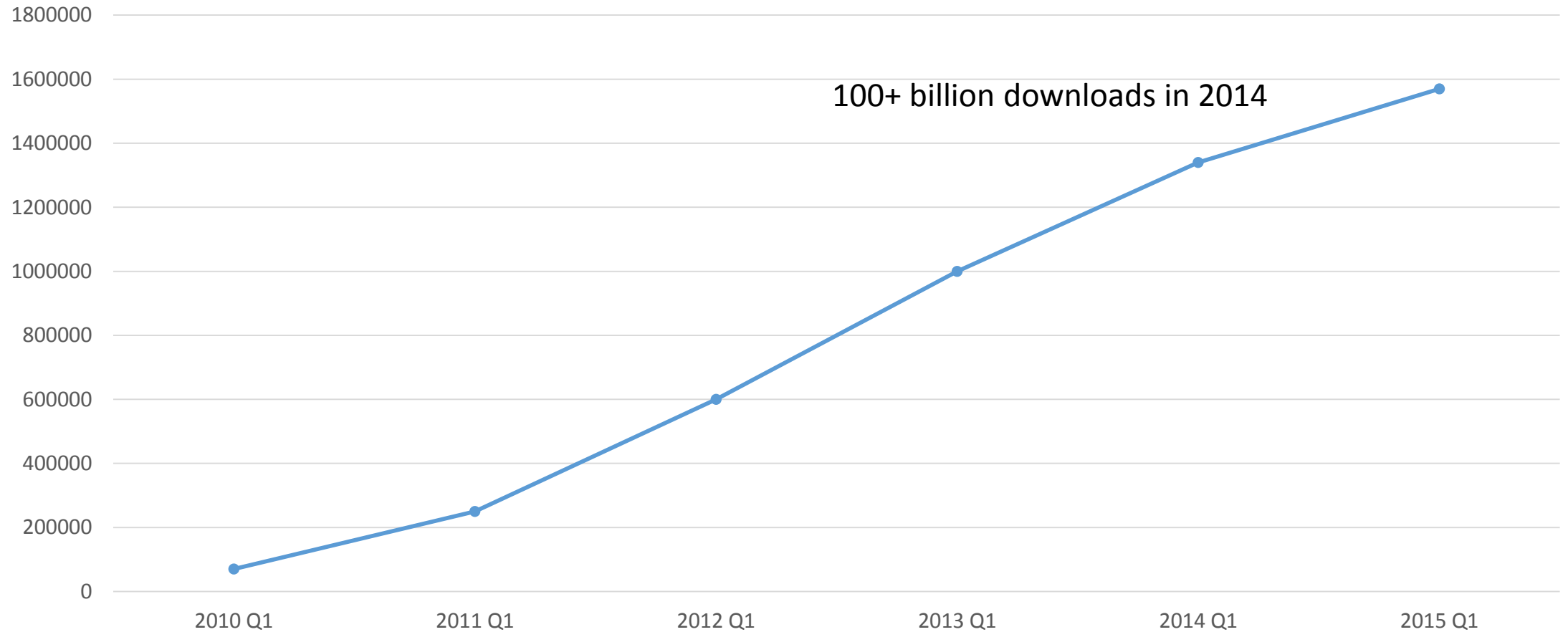
Here's Why Your Business Needs Its Own Mobile App

Melanie Haselmayr , Contributor

Number of Apps [Gartner]



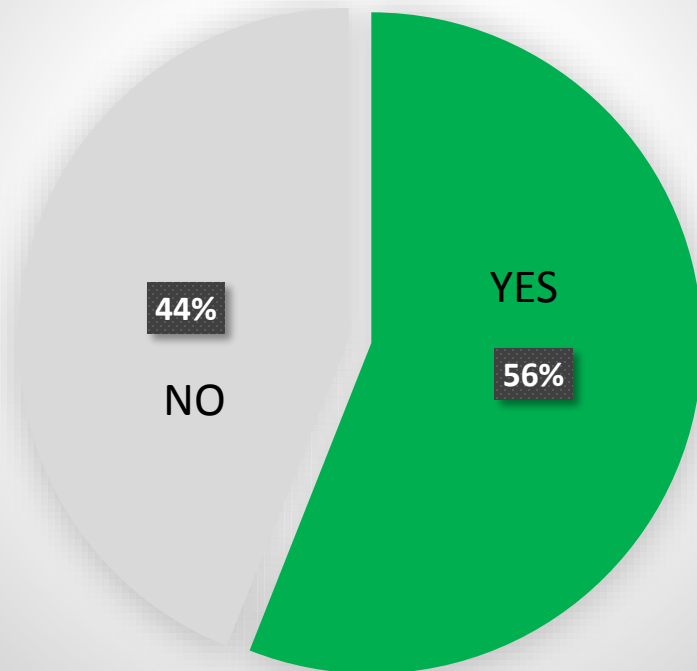
Google play



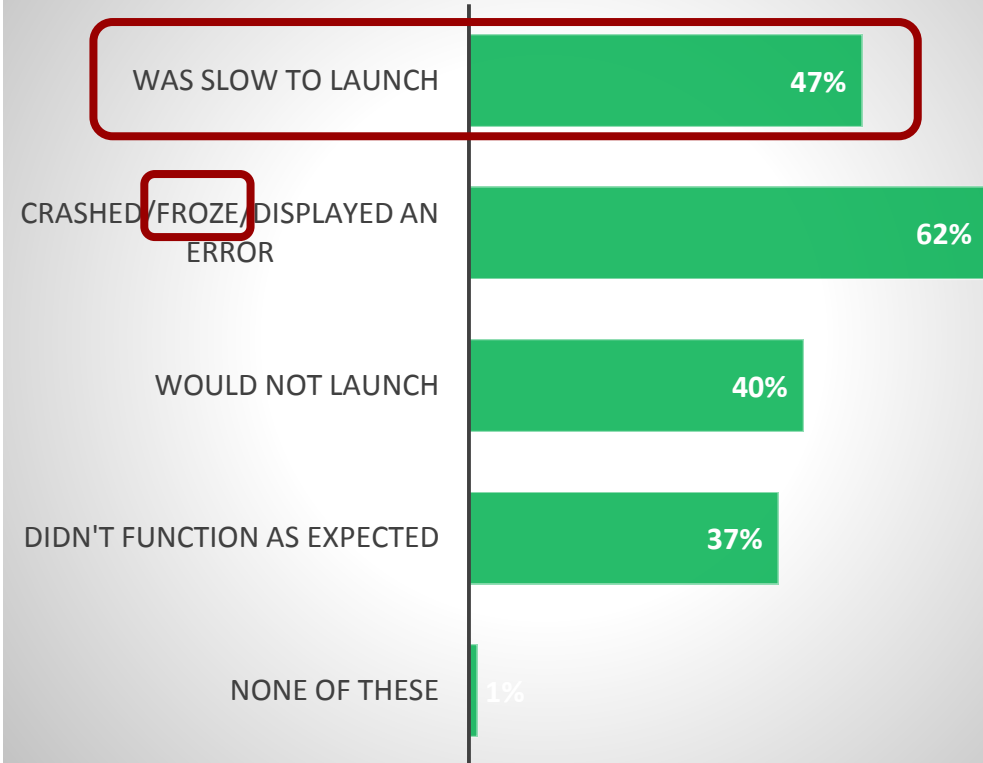
But app performance problems are common

[TechCrunch article covering a survey of 3000+ users commissioned by Compuware]

Have you experienced a mobile app problem in last 6 months?



Type of problem encountered



Performance problem: more elusive than crash

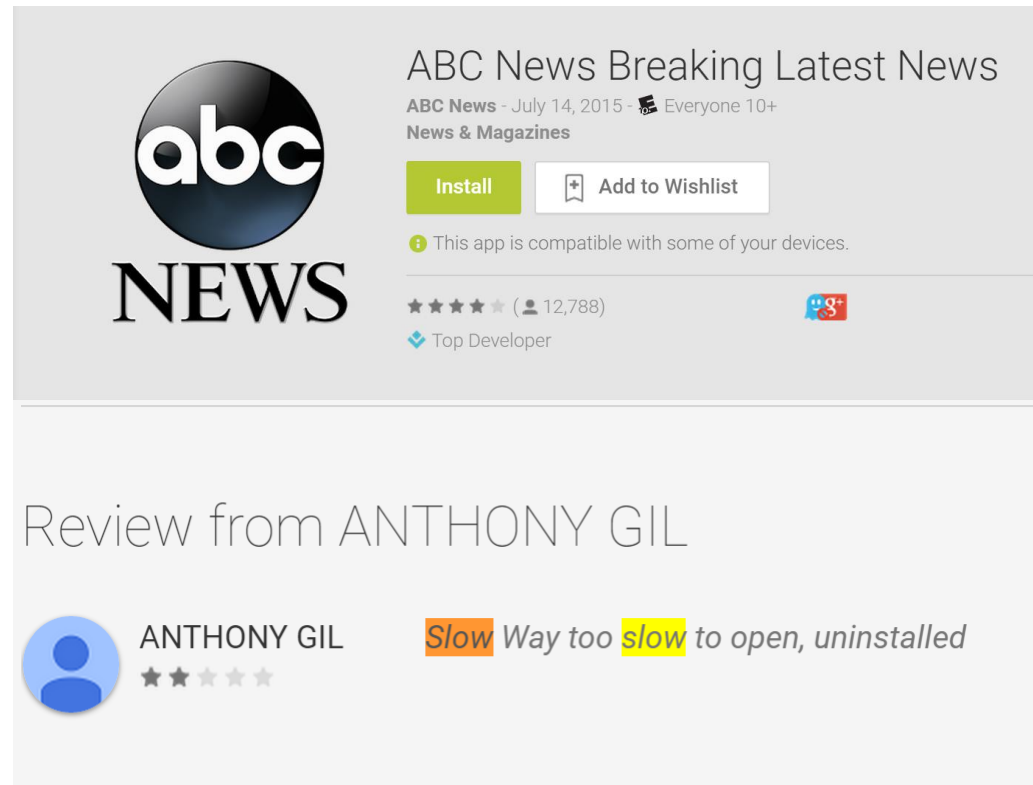
- Crashes: easy to notice
 - Get caught in testing
 - Low overhead to monitor on user devices
 - Tools such as Crashlytics are available to help
- Performance problems: elusive
 - Not easy to test
 - High overhead to monitor
 - Lack of tools

Performance in mobile: **harder** than PC/web

- Mobile has wimpy hardware, constrained by battery life
 - Unlike PC
 - Particularly true for Android
- Mobile has complex programming model
 - Unlike web
- Mobile is relatively young → Lack of effective tools

Users have **little tolerance** for sluggish apps

- 86% of users deleted or uninstalled at least one mobile app because of problems with its performance [AppDynamics]



Build fluid apps, boost user engagement

- Delight your users
- Win more users
- Strengthen your brand
- Stay ahead of the competition
- Earn more money 😊



Developers of popular apps are already doing it



“We managed to cut the Instagram app start time in half over the past year”

“To achieve these gains, we spent a lot of time profiling the app”



“[Building a better Instagram app for Android](#)” Tyler Kieft



We found performance monitoring code in other popular apps, too

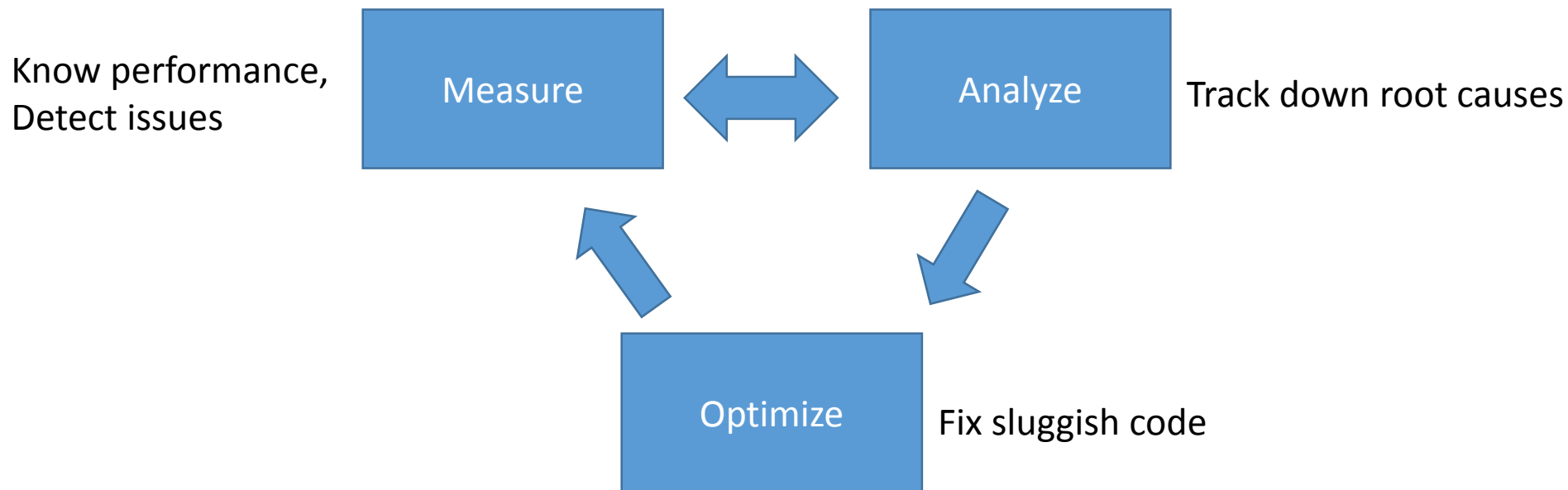


We’ve worked closely with their developers.
Will use as examples in this talk

How to make my app fluid?



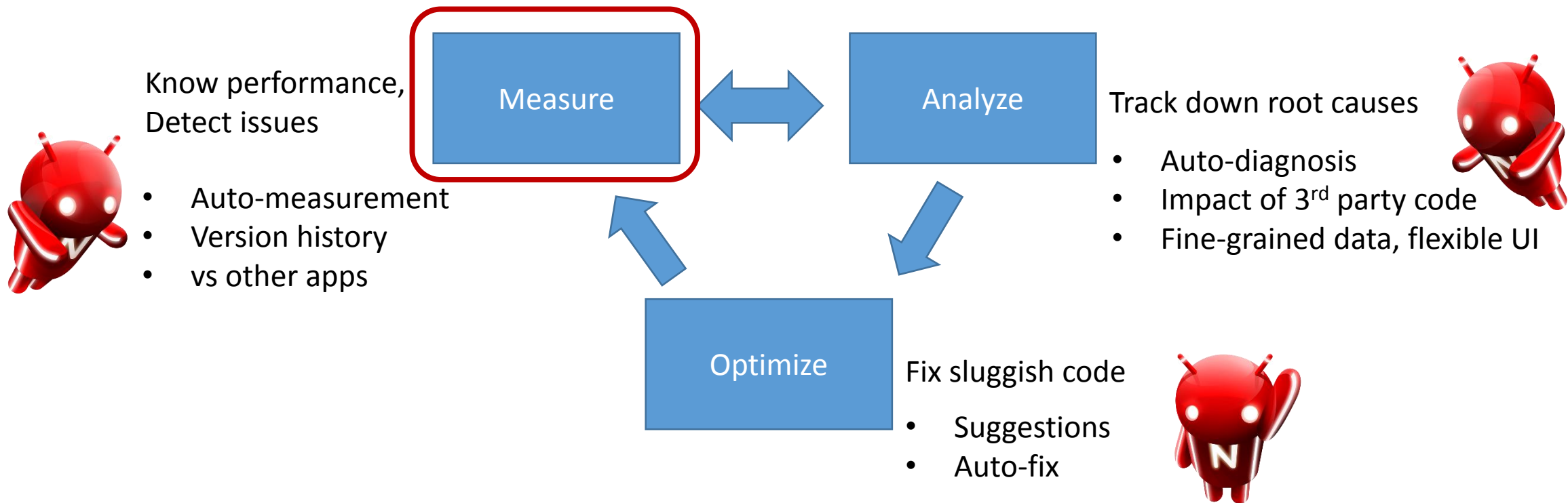
Establish a performance optimization **process**



Can be tedious, time consuming, and difficult 😞



Let NimbleDroid help you



Measure

- What app performance **metrics** to measure?
 - Metrics 1: response time
 - Important special case: app start time
 - Metrics 2: smoothness
- What **thresholds** define good vs bad?
- **How** to measure?
- **When** to measure?

Metric 1: response time

- How long does it take your app to respond to a user action?
- Example scenarios
 - Start your app
 - View a news article
 - Load contact list
 - Check a friend's profile

Important special case: start time

- First impression matters!
 - 79% users would retry a problematic app only once or twice [compuware]

“Users don’t want to wait forever for their apps to start up.”



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 user input credentials
- *Warm start*: shortly after user switches away from app
 - App still cached in memory



Optimize cold start first

- Cold start has high UX impact

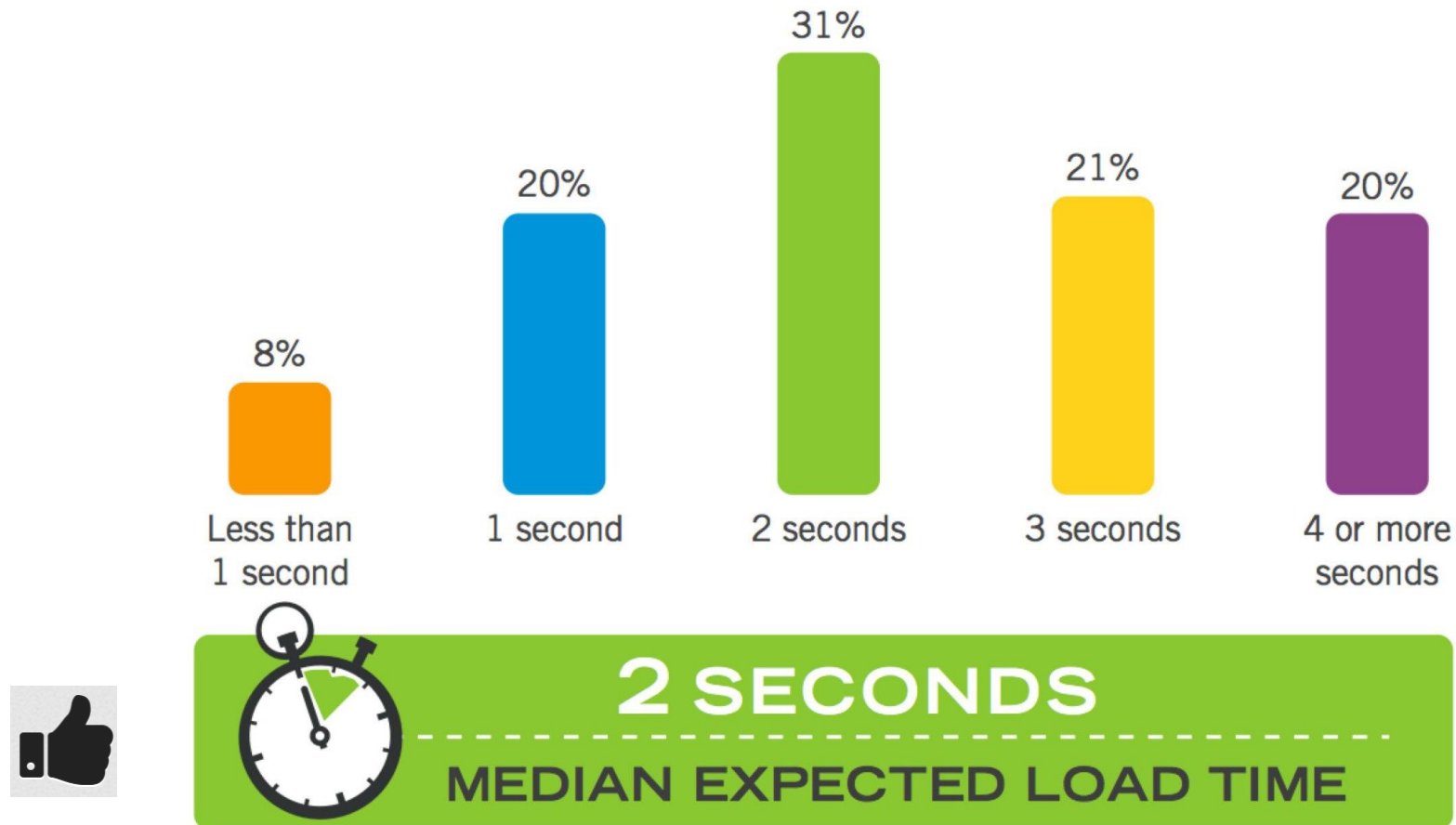
Type	Time	Occurrence	UX Impact
Cold start	Long	Often	High
First start	Longest	Rare	Low
Warm start	Short	Most often	Low



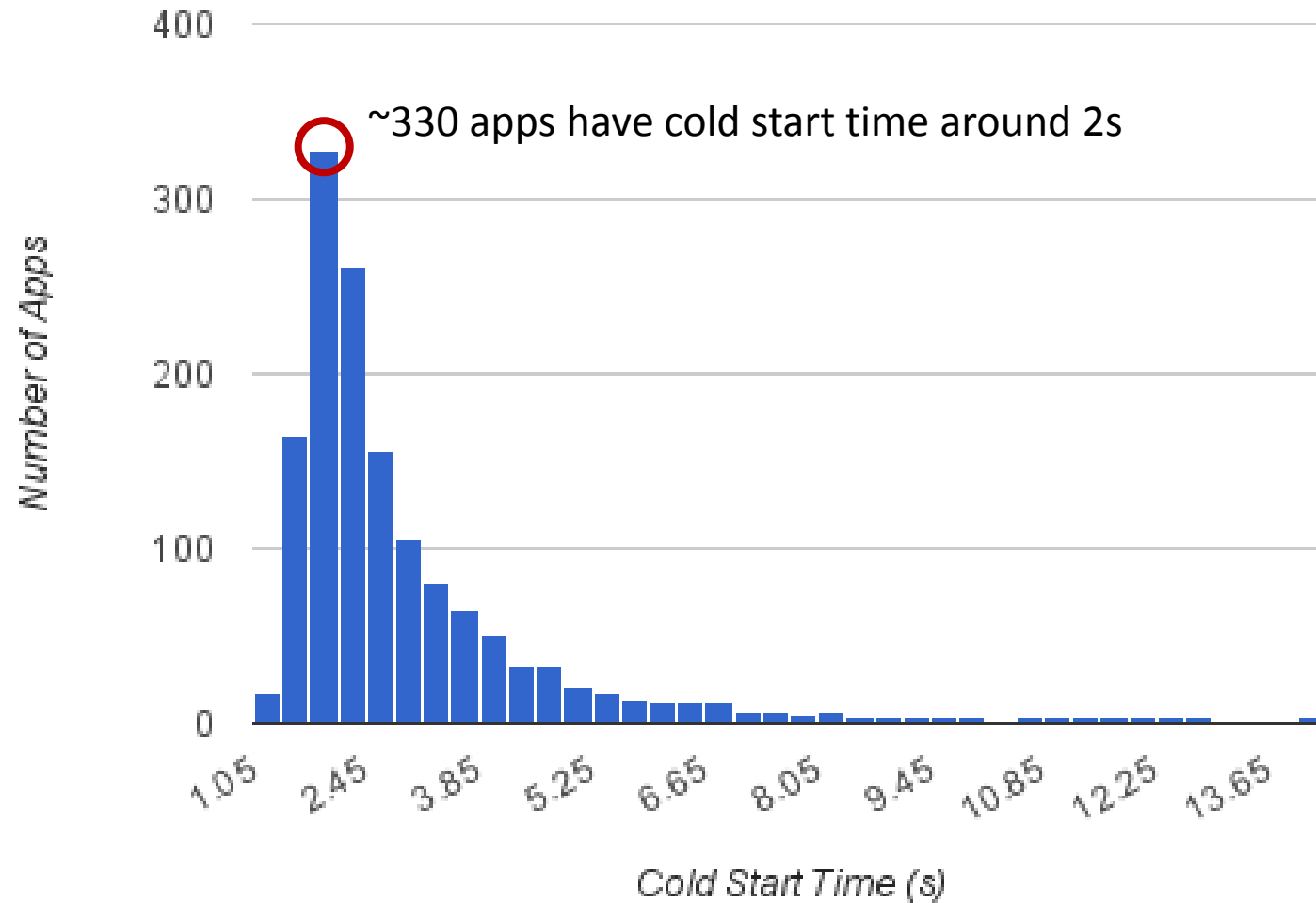
- Faster cold start → faster first start

Threshold for cold start

[TechCrunch article covering a survey of 3000+ users commissioned by Compuware]



Cold Start Times (~1400 non-game apps with 5M+ downloads)



Min: 0.80s
Median: 2.35s
Mean: 2.86s
Max: 14.09s

Methods to measure cold start time

- Manual: eyeball + wall clock
 - Time consuming, imprecise
- Semi-manual: record with high speed camera
 - Time consuming
- Programmatic: write a performance testcase
 - One time cost of writing testcase, but still have to monitor and analyze
- **Fully-automatic**: let NimbleDroid do it for you



Demo: measure start time with NimbleDroid

- Request your app to be profiled; or upload your app *Coming soon*

- Q: how do we know when your app is done starting up?

- Heuristic: when main activity is displayed
- Or you tell us using a logging statement *Coming soon*

```
Log.i("nimbledroid", "I'm done starting up");
```

- Q: what if your app requires login?

- You give us test username/password when uploading app *Coming soon*
- We analyze app's UI and automatically log it in for you

Measure response time in general



- Write a testcase for each scenario you want to measure
 - Multiple UI testing frameworks exist (robotium, uiautomator, espresso...)
- Python wrapper of uiautomator (<https://github.com/xiaocong/uiautomator>)
 - Very lightweight



Install `$ pip install uiautomator`

Run

```
from uiautomator import device as d
d.screen.on()
d(text="Next").click()
```

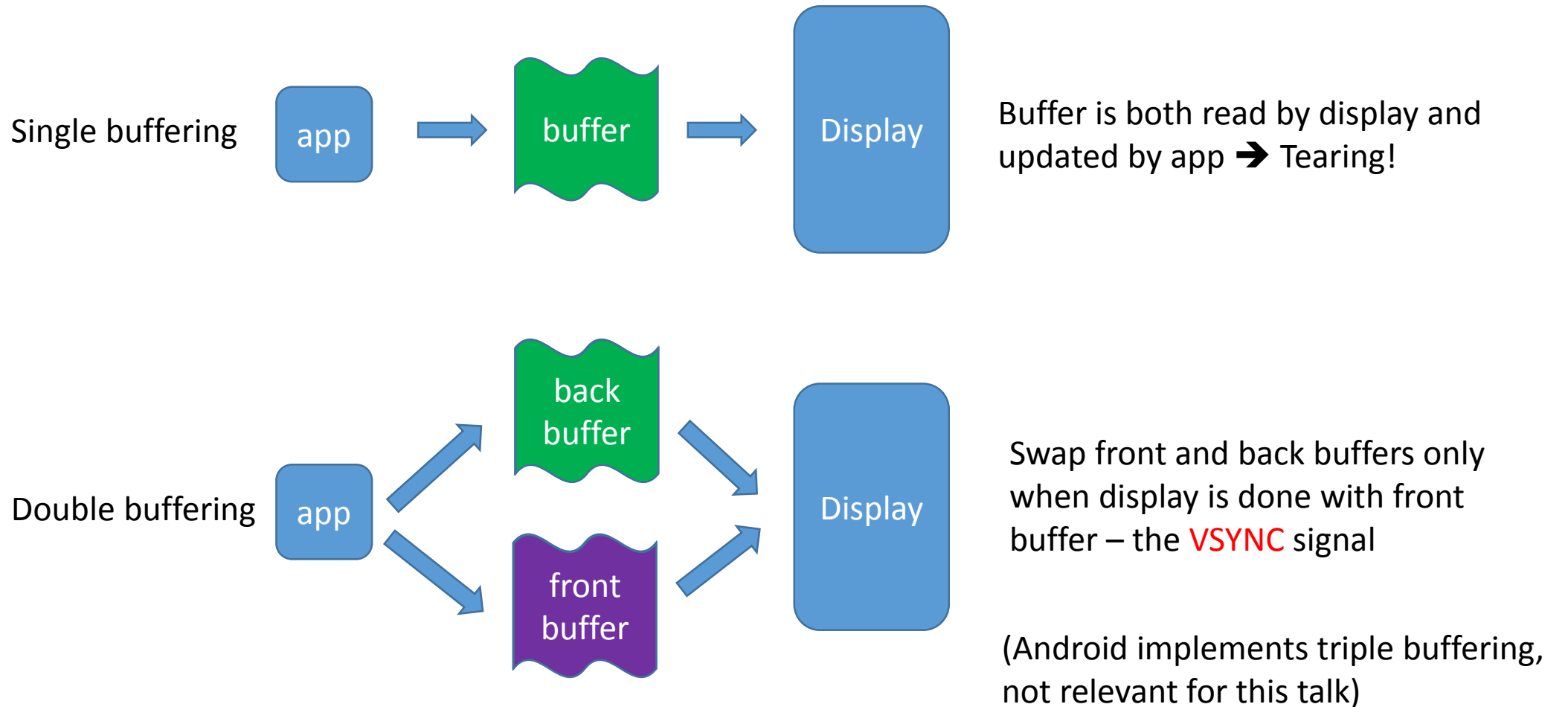
- Measure response times by uploading testcases *Coming soon*

Motivation for maintaining smoothness

“The doing is often more important than the outcome.” - Arthur Ashe

- It is important to have fast overall response time, but it is also important that the process of responding is smooth without “lag” / “gap” / “stutter” / “choppiness” / “jitter” -- not just for games
- Human eyes are quite sensitive to temporal lags [PMC]
 - Average population could perceive 22ms or bigger lag
 - 25% of population could perceive 2ms -- 16.7ms lag
- But how to measure?

Background: VSYNC to avoid display tearing



Implications of VSYNC

- Typical displays refresh at 60 FPS → VSYNC fires every $1/60\text{s} = 16.7\text{ms}$
- System needs time to draw, too
- App doesn't finish drawing within 16ms → Lag
- In Android, only UI thread does the drawing
- Thresholds
 - No methods in UI thread should run longer than 16ms
 - Or, cut yourself some slack and make it 32ms

Understanding smoothness

- App's FPS (frames per second), number of dropped frames (frames that miss the 16.7ms deadline, also called “janky” frames)

```
$ adb shell dumpsys gfxinfo <package name>
```



- Same command with “Profile GPU rendering” on device outputs simple frame timing data for the last 120 frames
- Detailed frame timing for the last 120 frames

```
$ adb shell dumpsys gfxinfo <package name> framestats
```



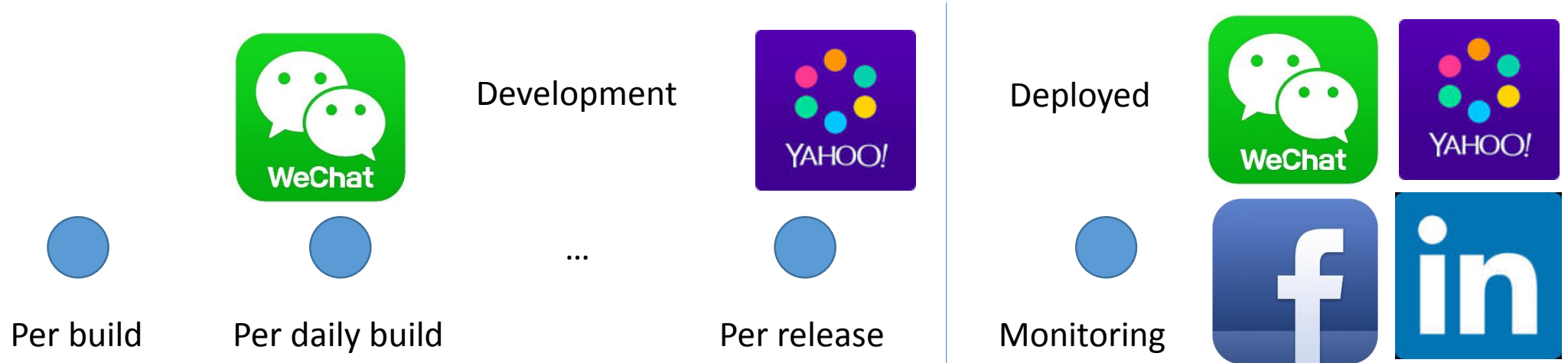
- Long-running methods in UI thread

- Manual: systrace, traceview



- **Fully automatic:** let NimbleDroid do it for you

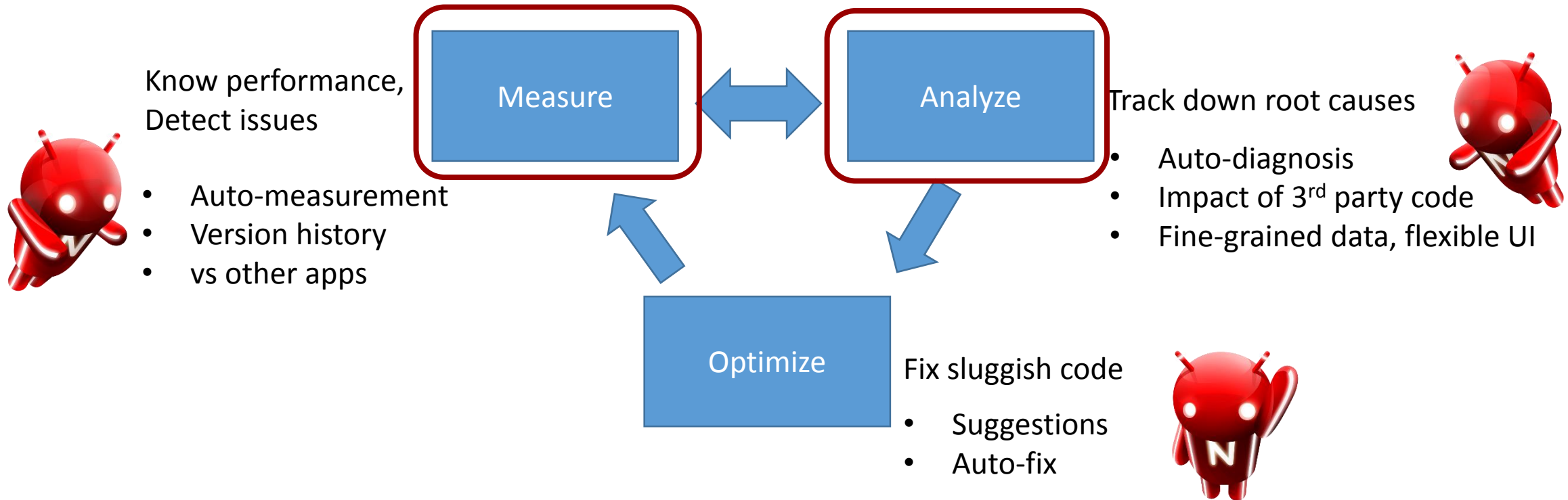
When to measure?



- No reason not to do them all if you can
- The **more often** you measure, the **earlier** you catch issues

Demo: version history in NimbleDroid

Recall: optimization process





Tips for analyzing performance issues

- Delegate analysis to NimbleDroid to save your precious cycles
- Understand common mistakes to avoid them in your code
- Avoid surprises in 3rd party code

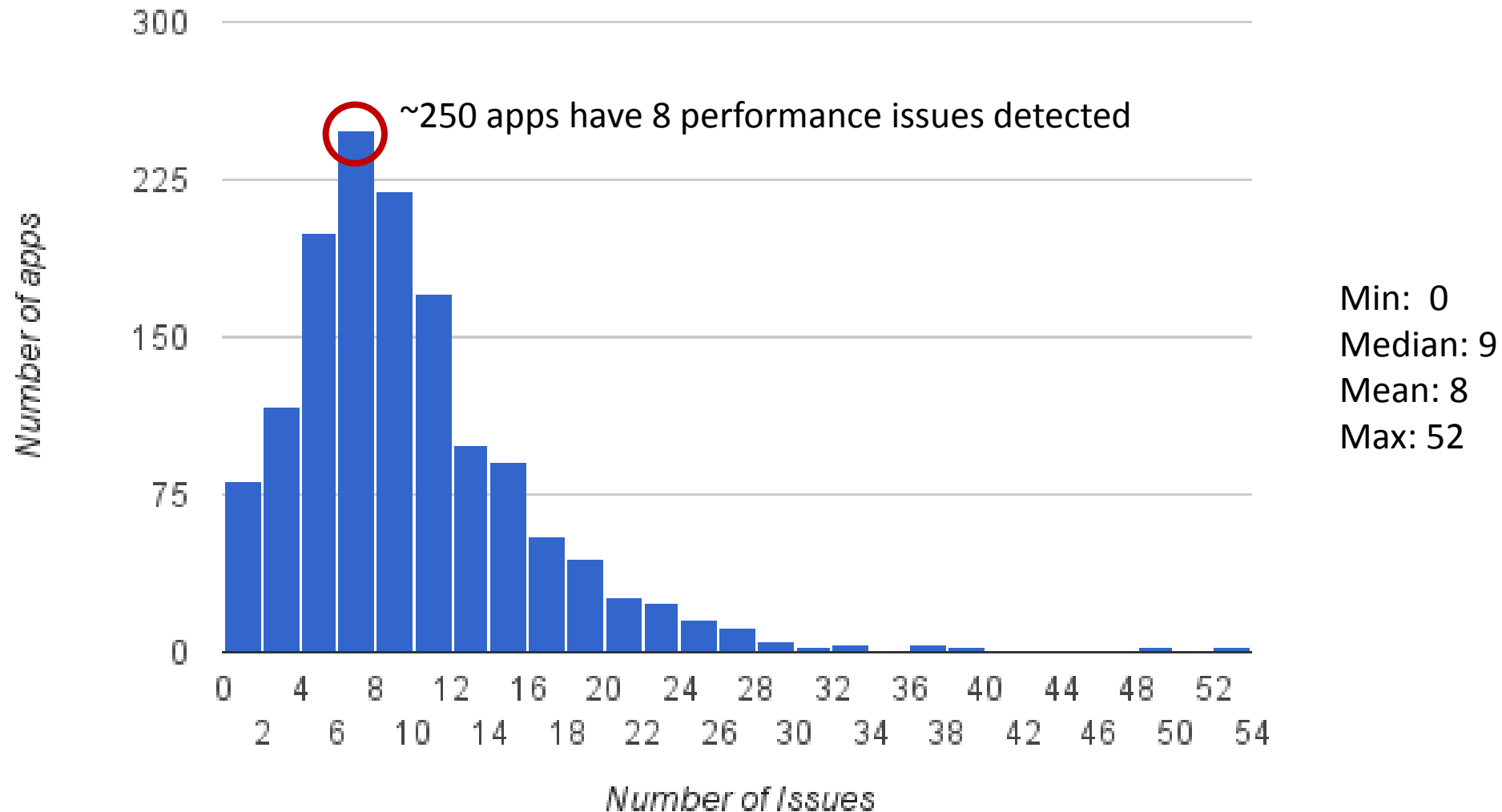
Taxonomy

- *Hung method*: a method in the UI thread that runs/blocks for longer than 16.7ms
 - Many possible reasons: network I/O, expensive computation, lock contention
- *Hot method*: a method that consumes a large percentage of CPU time
 - Steals CPU time from UI thread or background threads doing work for UI

“News was originally implemented as a webview, and after profiling we were surprised to find that *it was creating a lot of threads on startup, stealing time from the main processor.*”



Histogram of Number of Performance Issues Detected (~1400 apps with 5M+ downloads)



Example issue: load large resources



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

stores_af.json - 188 KB
stores_hco.json - 419 KB
stores_kids.json - 108 KB



- Don't load large resources during startup



Example issue: `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 by Android: index all resources in APK, verify certificate, parse manifest file
-  ↓0.4s  ↓1.3s  ↓1.7s
- Don't use `ClassLoader.getResourceAsStream()` during startup



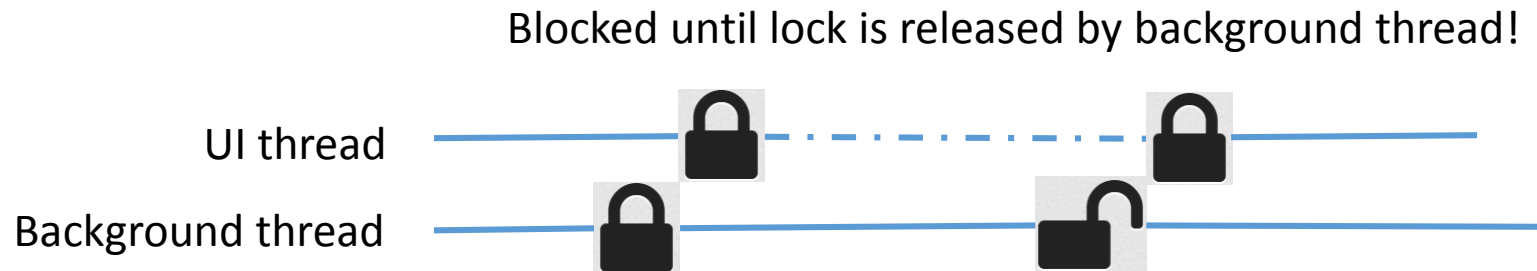
Example issue: dynamic dependency injection


- *Dependency injection*: a software design pattern that passes dependencies needed by an object via constructor or property setter
- Two possible approaches: *dynamic* (RoboGuice) vs *static* (dagger1&2)
- Dynamic approach slows down app startup significantly!
-  ↓3s  ↓0.8s
- Both apps now use dagger
- Use static dependency injection!



Example issue: synchronization

- Problem: reading data source is fast, but updating is slow
- Naive solution: move updating to background but keep read in UI



-  ↓0.9s (check friend's profile)
- Be extra careful with synchronization in UI thread!




Surprises in 3rd party code

- 3rd party code (SDKs, Android) can slow down your app unexpectedly
- Hard to track down because code isn't written by you
- Not your fault, but you have to live with or work around it ☹️

Example surprise: org.joda.time.DateTime()

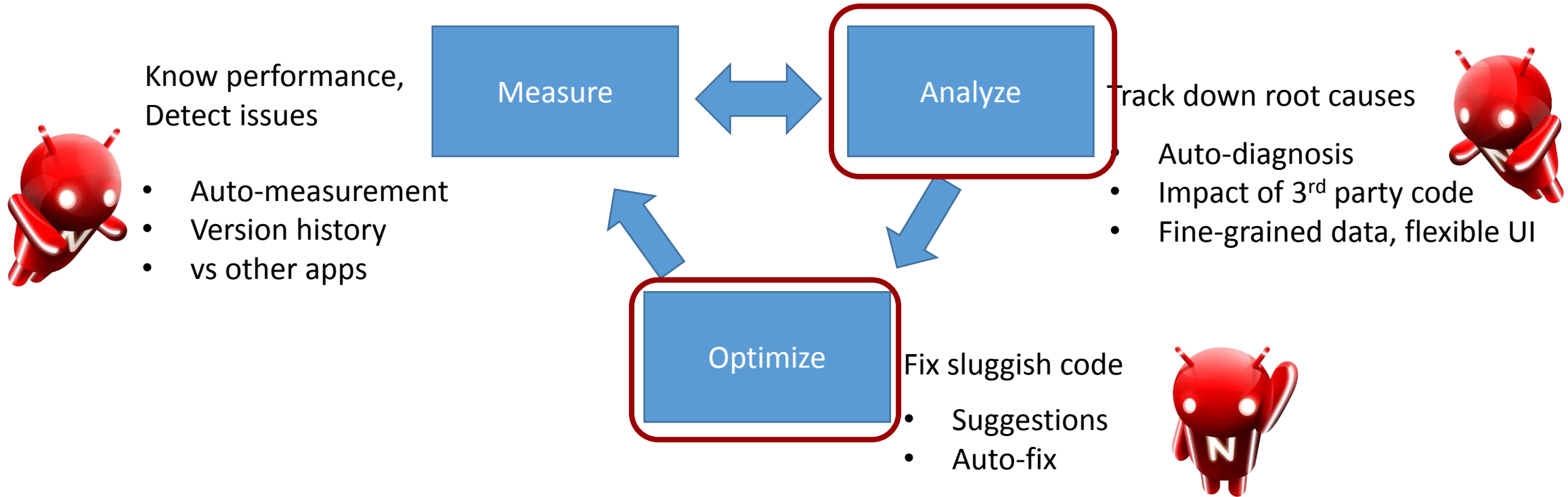
```
import org.joda.time.DateTime;
import android.app.Application;

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](https://stackoverflow.com/questions/24832322/creating-a-fast-datetimezoneprovider)



Recall: optimization process



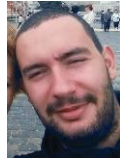
General fix strategies

1. Make method run faster
 - More efficient data representation, algorithm, and implementation
2. Defer to background

	Your code	Other code
Hung method	1, 2	2
Hot method	1	Switch? Complain?

Let NimbleDroid fix some of the issues for you *Coming soon*

A true story



Anton Krasov, android developer 2
popular libs, 1K+ stars on Github

I'd like to work for NimbleDroid.



May 11, 2015

One day later,
still no luck

Try diagnose Yahoo Fantasy Sports startup

Two minutes later,
root cause found

Try use NimbleDroid to diagnose



App creates ...DateTimeZone...

Two hours
later, fix found



what we need to do, is implement own
`org.joda.time.DateTimeZone.provider`

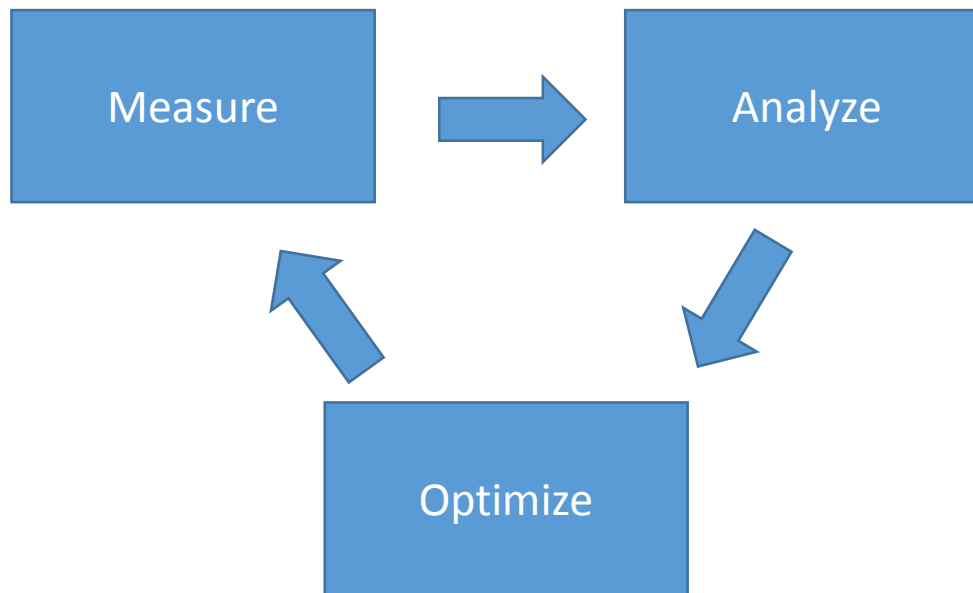
Issue with
Traceview



Joda Time was already initialized at the moment I
profiled app startup

Conclusions

- Build fluid apps, boost user engagement
- Establish a performance optimization process
- Let NimbleDroid help you



- Office hours: 1:30 – 2pm at BZMedia booth
- Thoughts? junfeng@nimbledroid.com,
twitter.com/junfengy
- We're hiring: <https://angel.co/nimbledroid/jobs>



See your app's performance now at
<http://www.nimbledroid.com>