# More Effective Testing on Android Devices

by Aurimas Liutikas / Google

# More Effective Testing on Android Devices

# Don’t - if you can avoid it

# Best - JUnit tests on JVM

Pros

- Cacheable in most build systems
- Multiple orders of magnitude faster
- Nudges tests to unit test scope

Cons

- Requires refactoring to pure JVM projects / isolation from android.* APIs

# Best - JUnit tests on JVM

Tips

- Run multiple tests at once maxParallelForks
- Gradle Enterprise test distribution

# Robolectric

A framework for running Android tests on JVM.

Built from Android source code with additional fakes.

Google-employee maintainers, but not an Google-owned product

# Good - Robolectric tests

Pros

- Cacheable
- Multiple orders of magnitude faster
- Able to test components that have Android tie-ins
- Easily fake system state (e.g. WiFi off)

Cons

- Not an accurate representation of a real Android device
- Google support is shaky

# Good - Robolectric tests

Tips

- Cache system image downloads in CI
- Try to minimize Android API usage
- 4.10 support @GraphicsMode(NATIVE)

# Okay - Activity-less on device

Pros

- Can be <100ms per test method
- Testing real Android behavior

Cons

- No caching* unless using Gradle Managed Devices (GMD) or custom runner
- Sharding on through multiple connected devices
- Flaky due to device instability

# If you must - with Activity on device

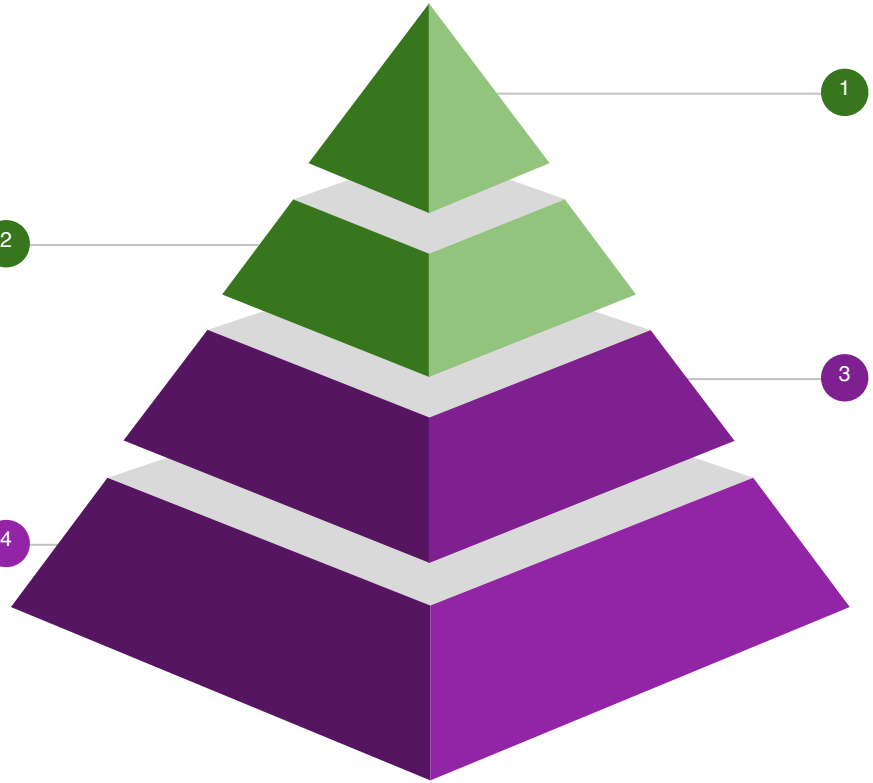
Pros

- Testing real Android behavior

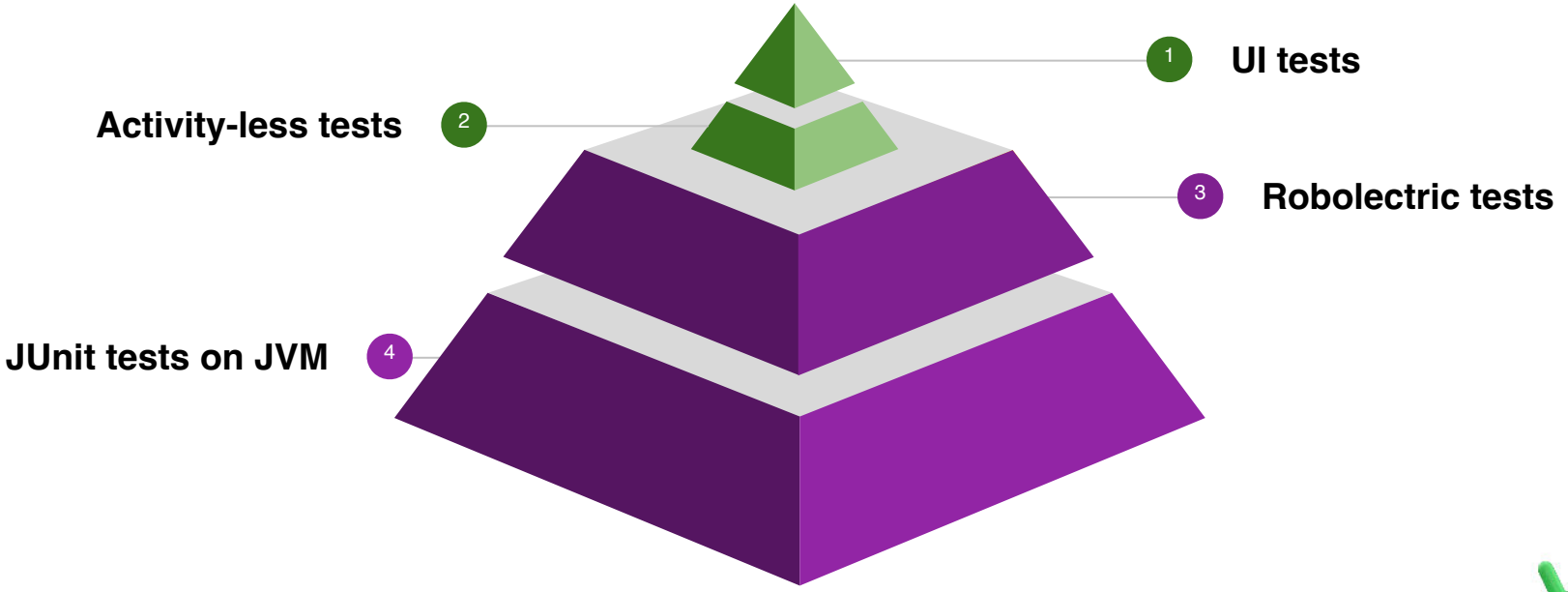
Cons

- Really slow
- No caching* unless using GMD or custom runner
- Sharding on through multiple connected devices
- Flaky due to device instability

Fidelity




Performance


1
UI tests
Activity-less tests
2
3
Robolectric tests
JUnit tests on JVM
4

# Test Stability Highly Important

- Flaky JVM tests are bad, flaky Android tests are worse
- Disable/delete flaky tests as running them has high costs
- State clean-up (e.g. @After)
- Factory reset or Android User Profiles in custom lab

# On Device Tips

# Only run what you need

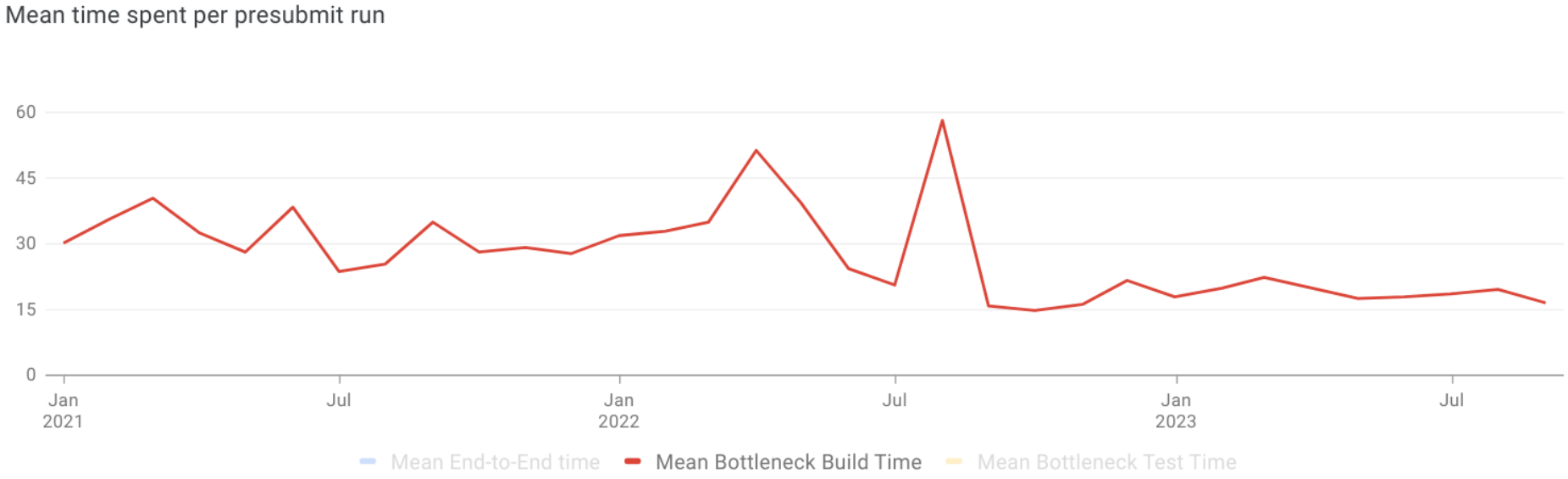
- AOSP system images
  - Disable noisy applications (adb shell pm disable-user)
- Automated Test Devices (ATD) images

# Modularize Tests Along With Features

- Splitting tests allows to shard
- Less interference between tests

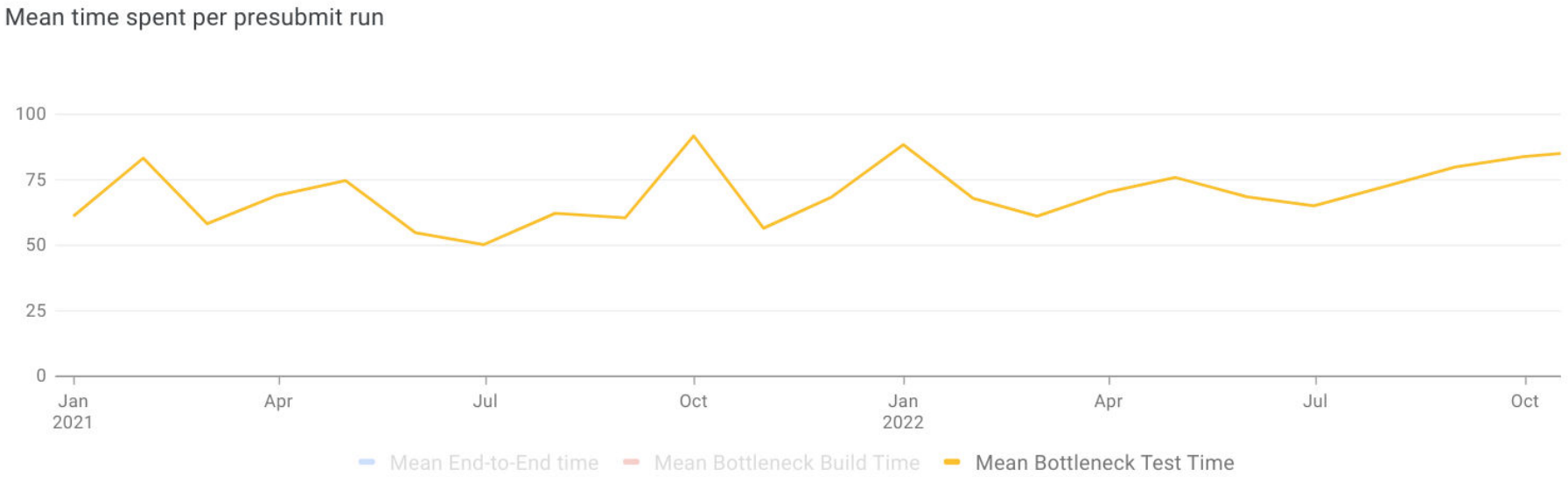
# AndroidX case study

# Build Time at Bay




https://dpesummit.com/chasing-the-speed-of-gradle-builds/

# Test Time Continuing to Grow




# Key Insight on APK checksums

Test results don’t change if both application and test APKs are the same

Combined with modularization → higher hit rate

# Unstable APK generation

- baseline.profm (issuetracker.google.com/issues/231837768)

# Unstable APK generation

- baseline.profm (issuetracker.google.com/issues/231837768)
- shadow jar including incremental kotlin data (r.android.com/2089482)

# Unstable APK generation

- baseline.profm (issuetracker.google.com/issues/231837768)
- shadow jar including incremental kotlin data (r.android.com/2089482)
- AndroidManifest.xml android:compileSdkVersionCodename (issuetracker.google.com/issues/277836549)

# Unstable APK generation

- baseline.profm (issuetracker.google.com/issues/231837768)
- shadow jar including incremental kotlin data (r.android.com/2089482)
- AndroidManifest.xml android:compileSdkVersionCodename (issuetracker.google.com/issues/277836549)
- r8 + API 34 record types regression

# Migration From Custom Lab to Firebase Test Lab

**Caching**

APK checksum result caching

# Migration From Custom Lab to Firebase Test Lab

**Caching**

APK checksum result caching

**Sharding**

from n devices to run m APK sets → 1:1

# Migration From Custom Lab to Firebase Test Lab

**Caching**

APK checksum result caching

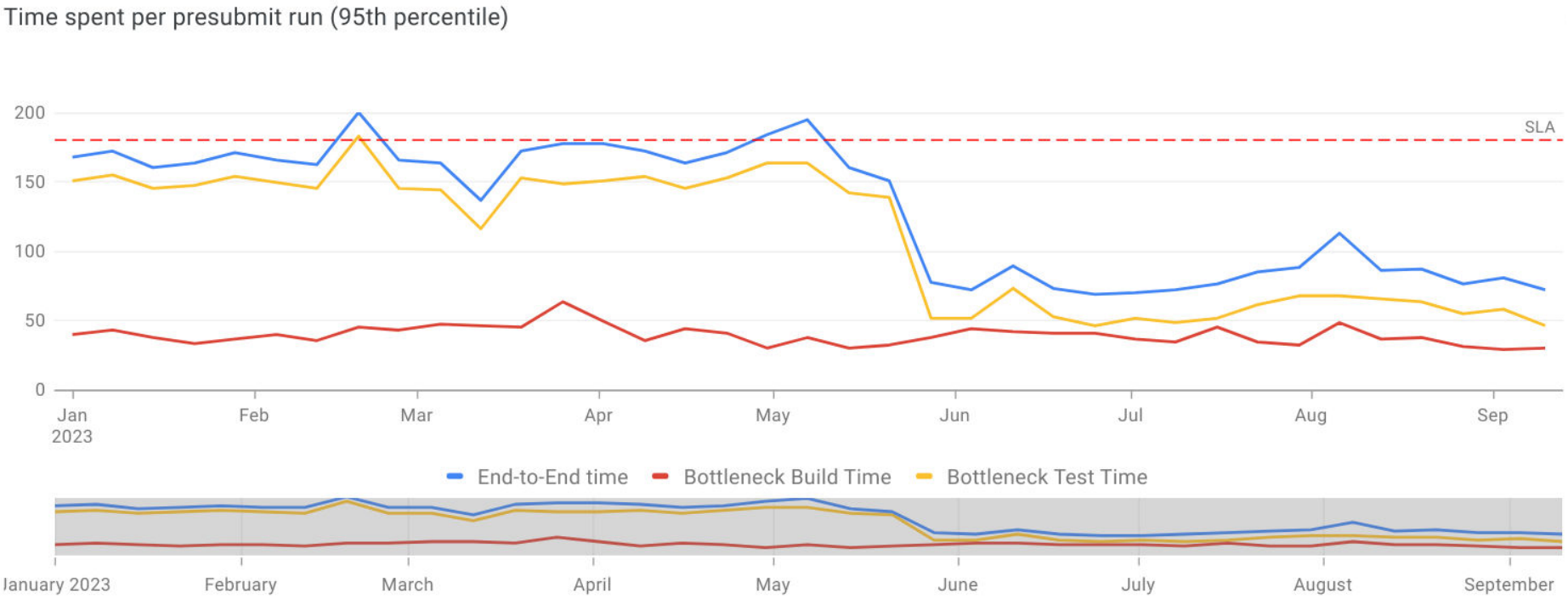
**Sharding**

from n devices to run m APK sets → 1:1

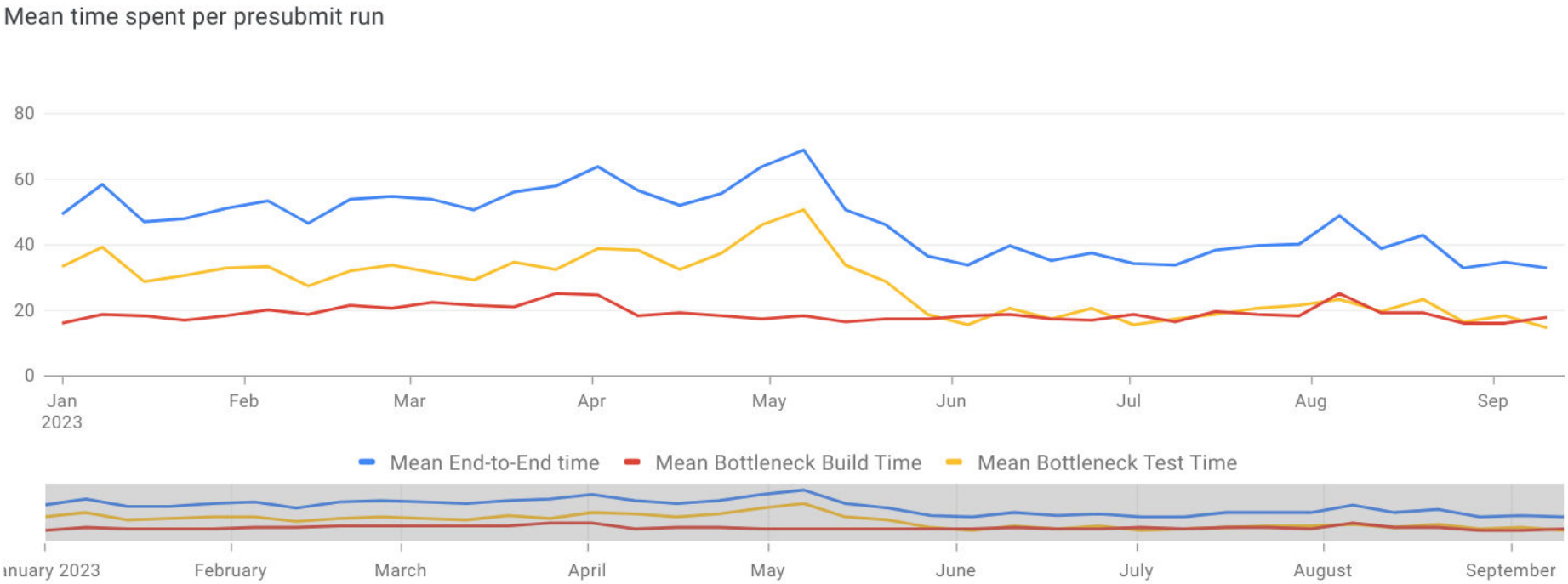
**Isolation**

multiple APKs sets per device →dedicated device per APK set

# Effects on 95th Percentile



# Effects on Mean Time



# What’s next?

- Replace FTL shard retries to per method retries
- Emulator stability work

Thanks!