DPE Summit 2024

Reducing Build Times by 50%

A Story of Tools, Data, and Persistence

PELOTON

Ward Bonnefond
Senior Staff Engineer

Douglas Crossley
Director of Engineering, Mobile

The Problem

Peloton Android App Ecosystem

2012

Peloton Android App Ecosystem
2024

1 Android Repository

Allows code sharing across projects

100+ Android Devs

1000+ Merged PRs (monthly)

27000+ Unit Tests

900+ Snapshot Tests

15 Gradle Projects

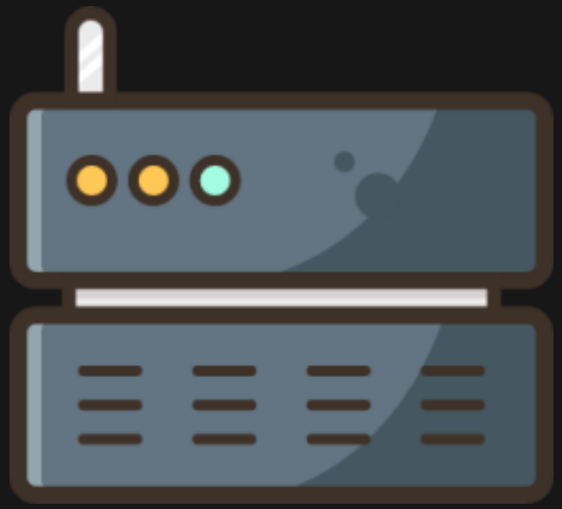
910 Gradle Modules

800+ Weekly PR Builds

200+ Weekly Master Builds

Android PR Build Job

Required for any code change to the repository



TRANSLATIONS

TOOLS

BUILD PROJECTS

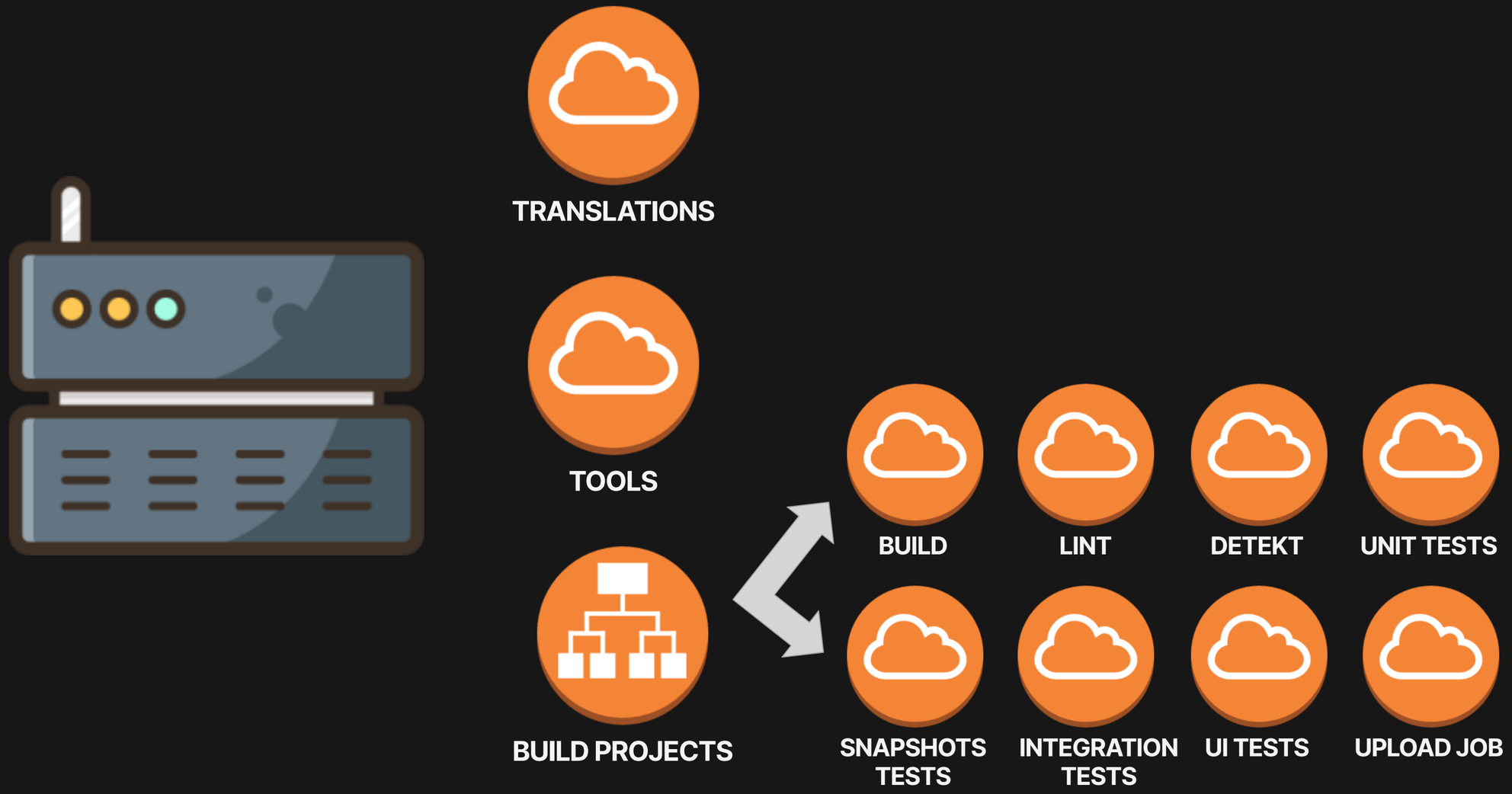
KICKS OFF SINGULAR WORKFLOWS

NUMEROUS TOOLS JOBS (EX: GITSTREAM, DANGER)

WORKFLOW PER PROJECT

Android PR Build Job

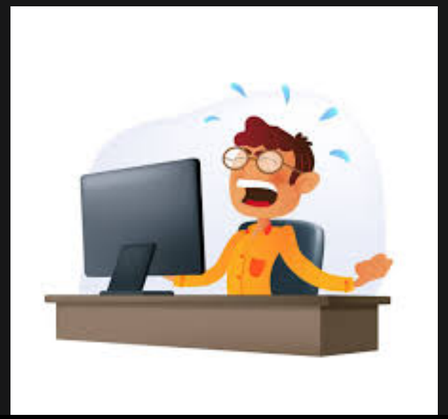
Each project has a number of jobs kick off




BUILD PROJECTS
BUILD
LINT
DETEKT
UNIT TESTS
SNAPSHOTS
INTEGRATION TESTS
UI TESTS
UPLOAD JOB

Build Times Kept Rising

Additional jobs and projects continuously added



Understanding the Problem

Lack of Observability made the problem feel subjective

“The build feels like it’s gotten slower”

“The builds take forever now”

“Why are the builds so much longer now?”

PELOTON

TOOLS & DATA

Develocity

Develocity

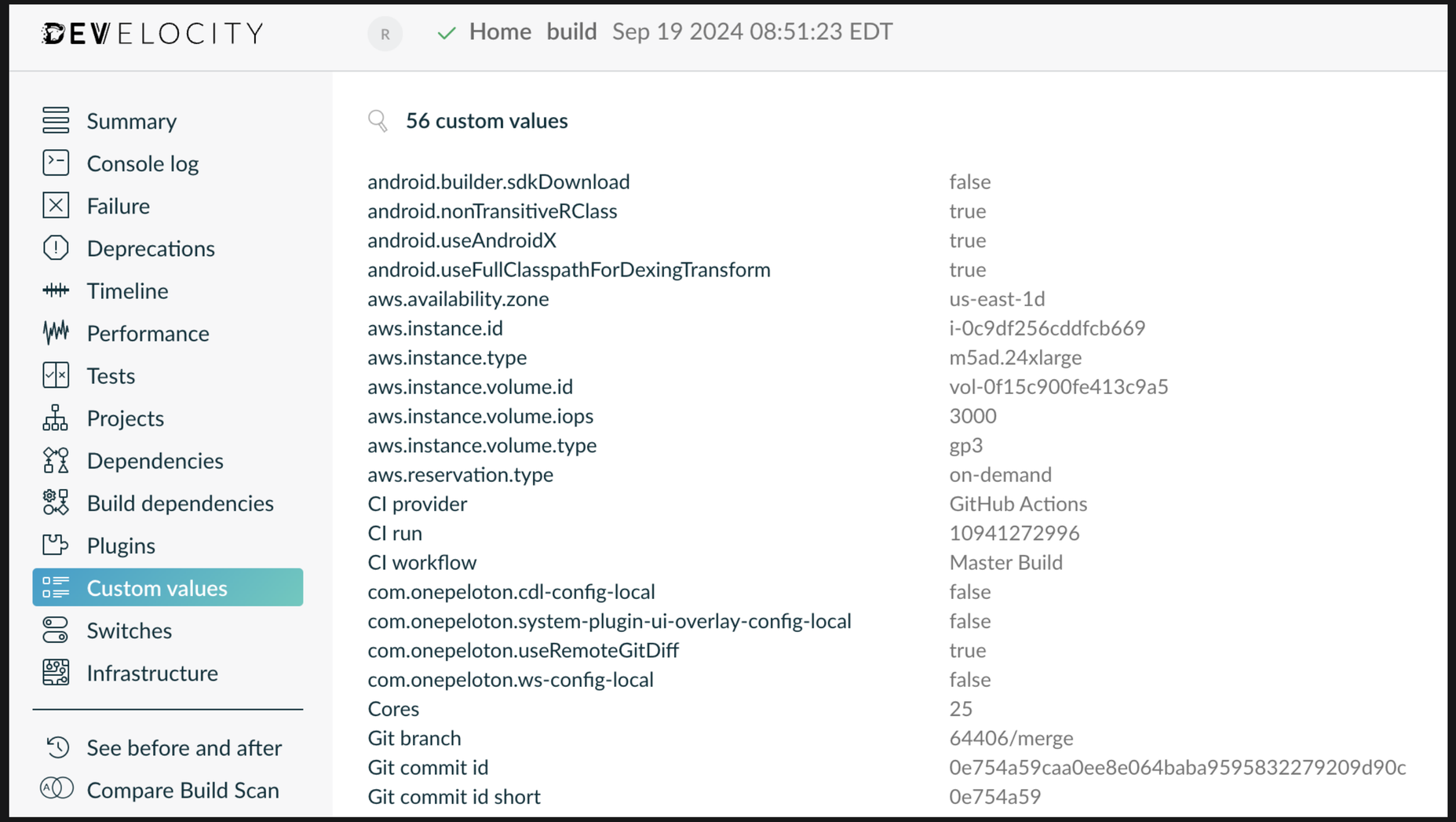
Visibility for builds across CI and Dev machines

- **Integrate Common Custom User Data Gradle Plugin**
  Enhances published build scans by adding a set of tags, links and custom values

- **Added custom values to query on different data**
  Allowed us to debug builds across different machine types

Develocity

Custom build scan values to support debugging



PELOTON

TOOLS & DATA

Datadog

Datadog
Insights into the entire end to end CI pipeline

- **Establish core build KPIs that we wanted to track**
  Build times p50/p95, build failure rate, uptime
- **Build Error Transparency**
  When the build fails, classify and track those error types
- **Build Resource Usage**
  Understand how we can optimize the hardware we’re running on

Datadog
Core build KPIs

Master Build Times

[P95] Master Build Time

5.8 hr

1.17 hr

0

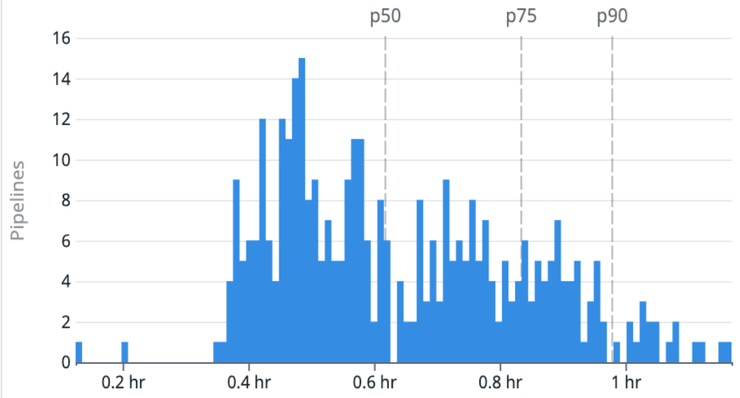
[P50] Master Build Time

5.8 hr

37.09 min

0

Master Build Times



PR Build Times

[P95] PR Build Time

5.8 hr

54.23 min

0

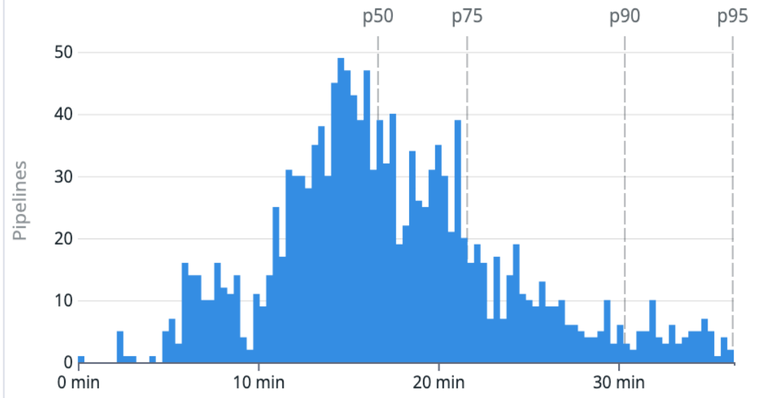
[P50] PR Build Time

5.2 hr

19.17 min

0

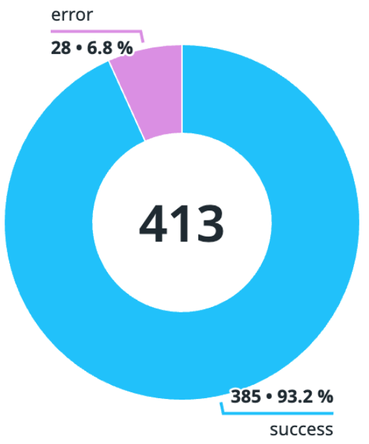
PR Build Times



Master Branch Build Uptime [ALWAYS CURRENT TIME]

Previous week	Week to date	Previous month
98.45%	100.0%	95.47%
Target 85%	Target 85%	Target 85%
Sep – Sep 8	Sep 8	Aug – Sep

Pipeline Status



PELOTON

TOOLS & DATA

Gradle Profiler

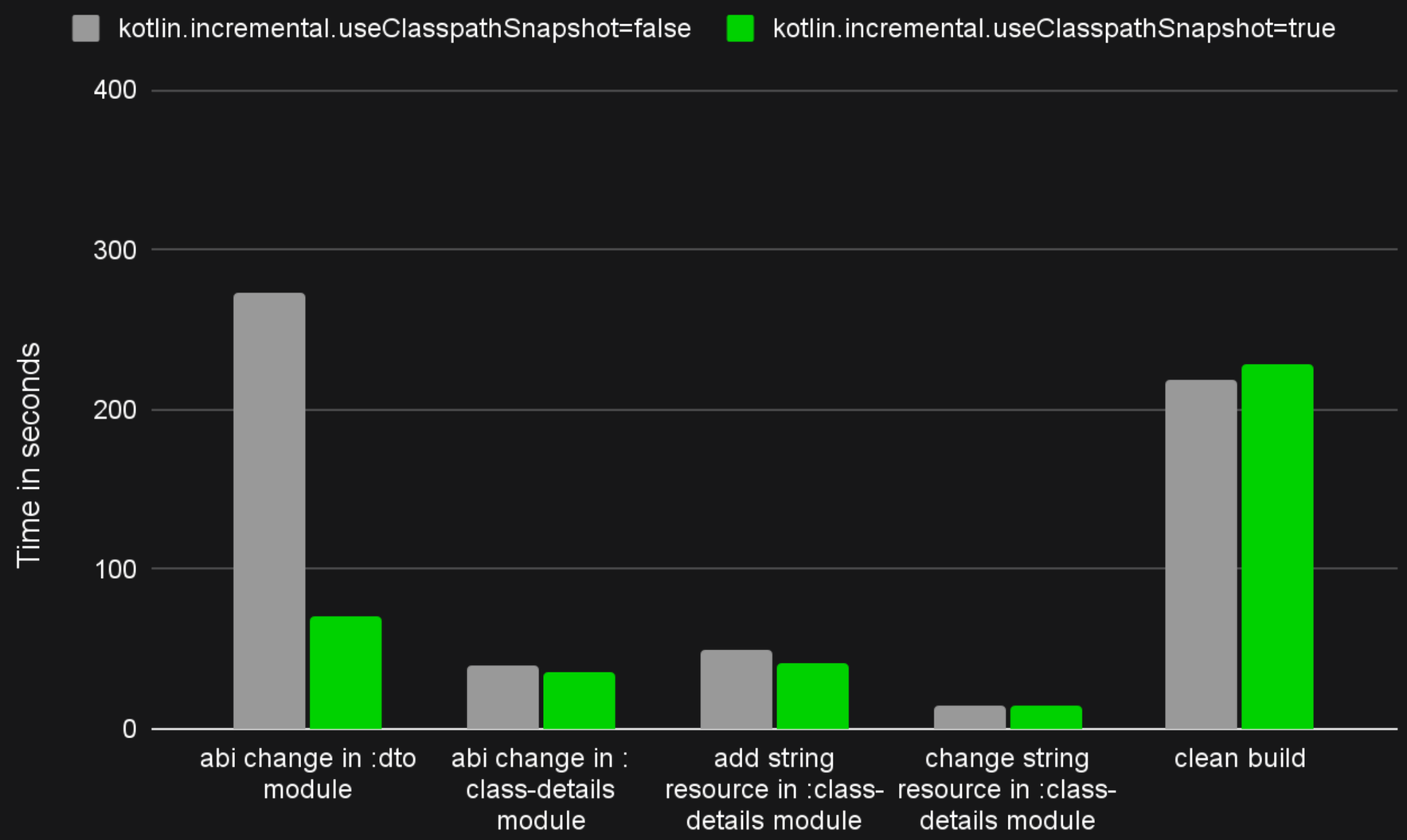
Gradle Profiler

Profiling and benchmarking for Gradle builds

- **Build changes are hard to measure and high risk**
  Gradle Profiler helped us develop confidence in all build changes we made
- **Setup CI workflow for the Gradle Profiler**
  Can create a performance scenario and add branches to compare
- **Local Usage**
  Quickly validate and generate build scan diffs

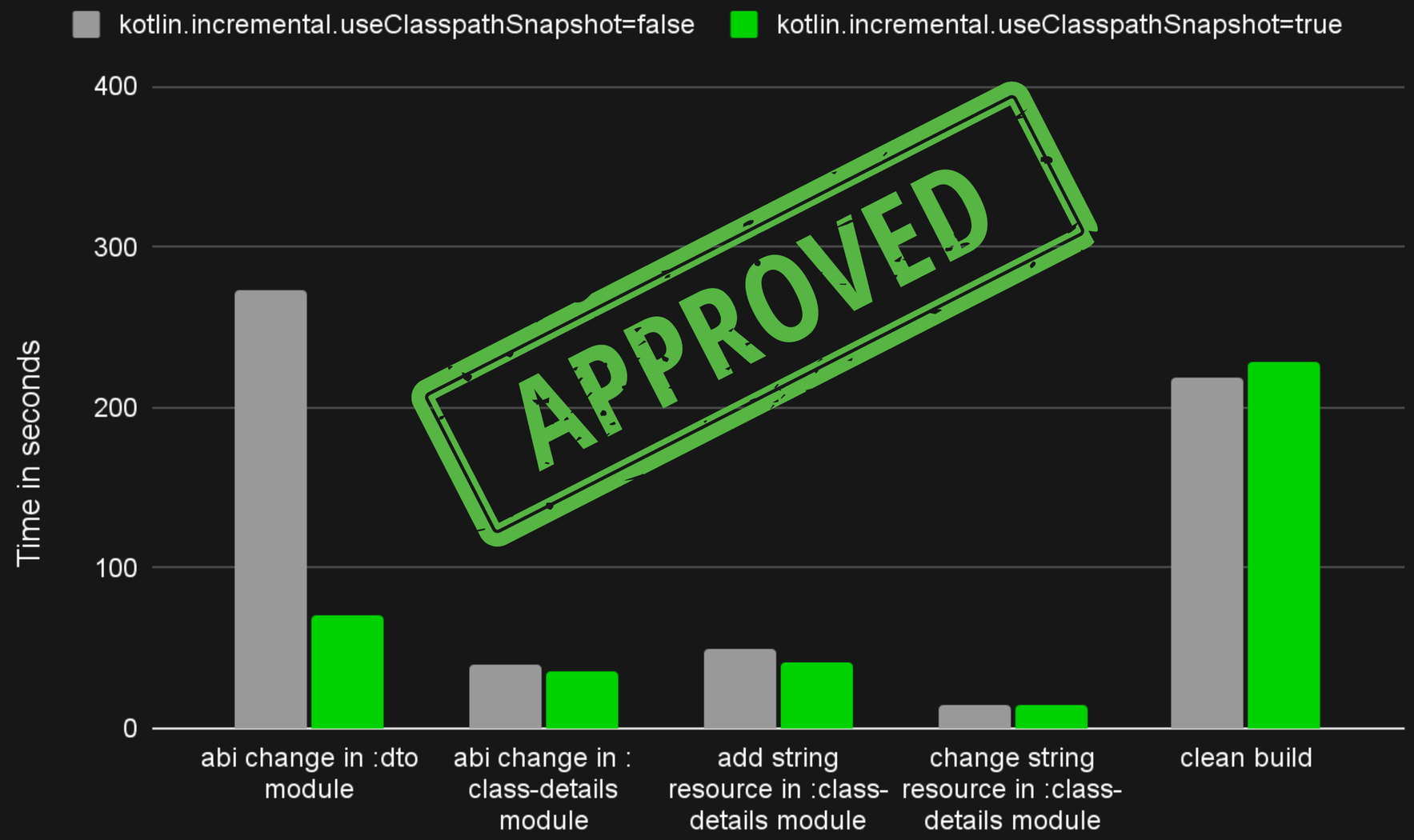
Gradle Profiler

Test early adoption of new Gradle properties



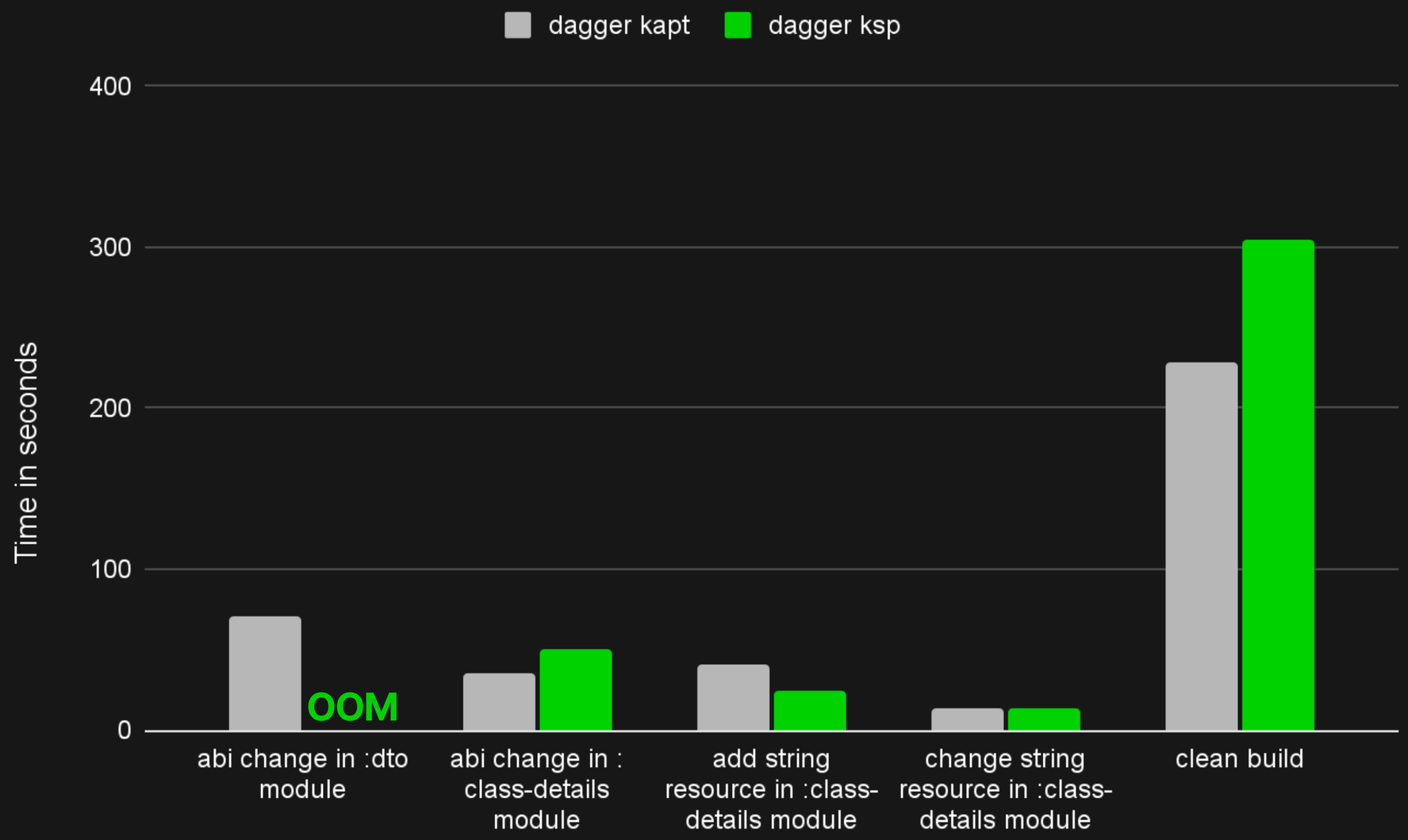
Gradle Profiler

Test early adoption of new Gradle properties



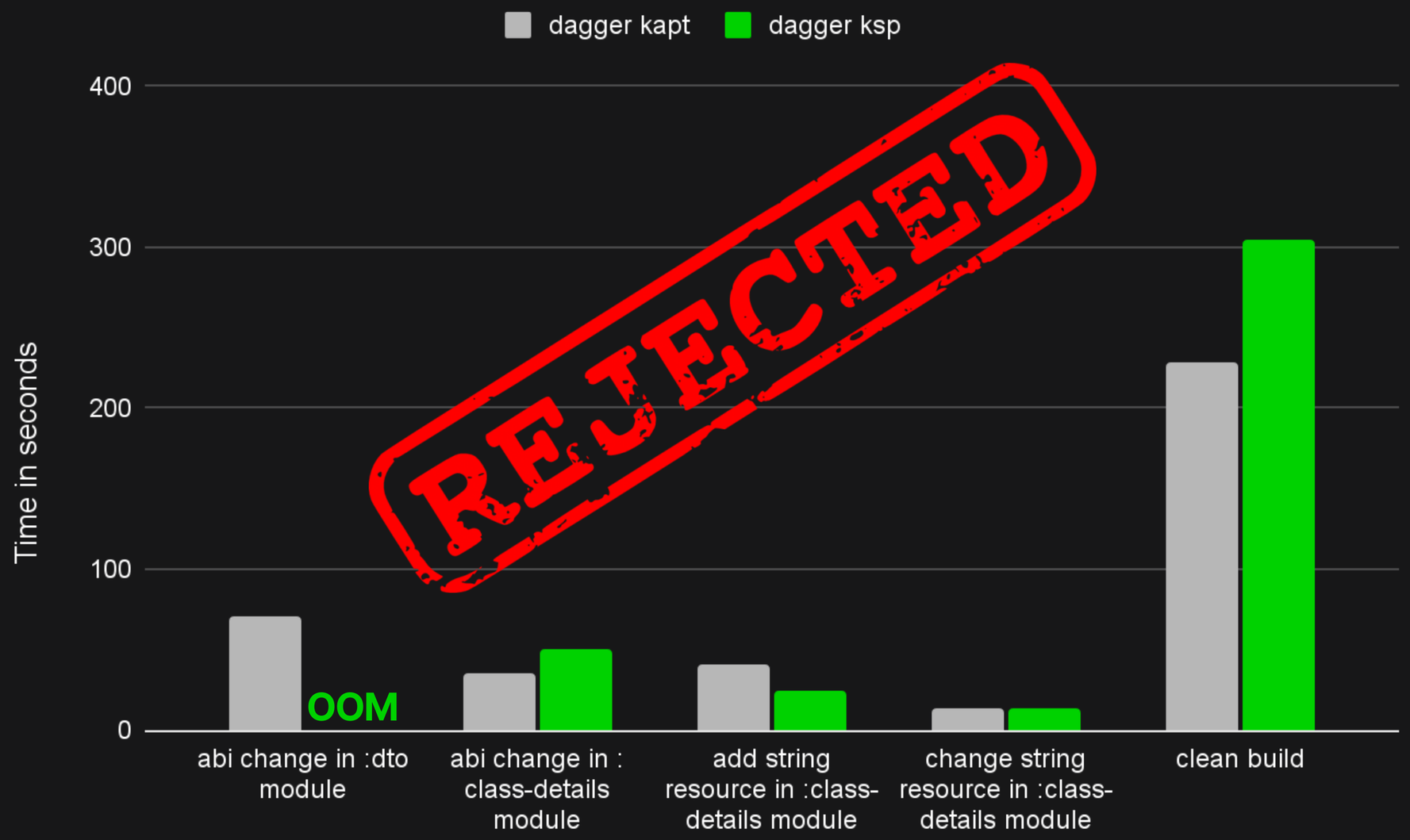
Gradle Profiler

Evaluate migration from Dagger KAPT to KSP



Gradle Profiler

Evaluate migration from Dagger KAPT to KSP



PELOTON

PERSISTENCE

Putting tools & data to work

Persistence
Putting tools & data to work

- **Optimizing unit test performance**
  Tracking down unit test issues on CI
- **Identifying high cost, low value jobs**
  What PR jobs provide the least ROI
- **AWS Infrastructure changes**
  Using the right instance types for our builds

Optimizing unit test performance

Unit Test Performance

Investigation into tasks with high “own times”

Test	Outcome	Total time	Own time	Serial time
Showing 1-200 out of 229 total items «First page ‹Previous Next› Last page»				
:companion-devices:ui:testDebugUnitTest ›	PASSED	5m 15.578s	58.880s	8m 48.901s
:tags-core:testDebugUnitTest ›	PASSED	4m 22.254s	3m 11.921s	5m 24.925s
:class-thumbnail:testDebugUnitTest ›	PASSED	4m 12.579s	2m 54.763s	4m 22.787s
:browse-components:scenic:testDebugUnitTest ›	PASSED	4m 10.490s	3m 6.396s	7m 45.047s
:auth:choose-profile:testDebugUnitTest ›	PASSED	4m 3.598s	2m 37.878s	4m 54.300s
:dynamic-scenic:core:testDebugUnitTest ›	PASSED	3m 52.535s	1m 36.529s	8m 26.360s
:training-programs:testDebugUnitTest ›	PASSED	3m 49.286s	59.976s	9m 56.921s
:metrics:ui:testDebugUnitTest ›	PASSED	3m 48.092s	1m 52.485s	9m 15.600s
:freestyle:core:testDebugUnitTest ›	PASSED	3m 35.199s	1m 42.395s	7m 59.776s
:inclass:feed:testDebugUnitTest ›	PASSED	3m 29.247s	2m 6.038s	5m 11.654s
:odyssey:app:testNomadDebugUnitTest ›	PASSED	3m 27.784s	1m 54.480s	6m 16.284s

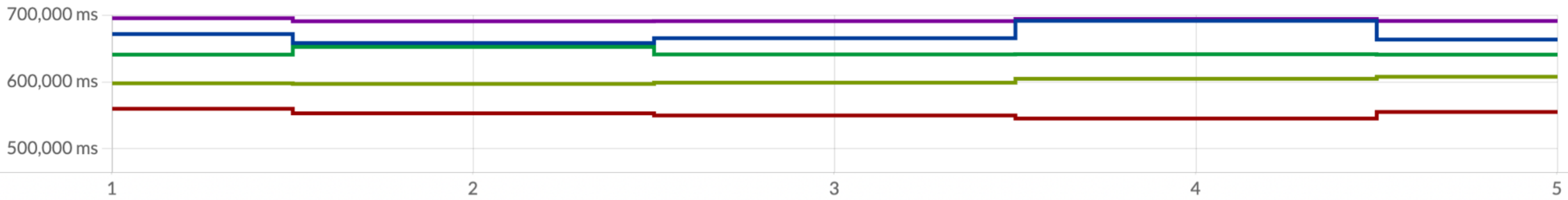
Unit Test Performance Optimizations
Understanding impact of maxParallelForks

```
// Gradle rec https://docs.gradle.org/current/userguide/performance.html
tasks.withType<Test>().configureEach {
    maxParallelForks =
        (Runtime.getRuntime().availableProcessors() / 2).coerceAtLeast(1)
}
```

Unit Test Performance Optimizations

Profile with different number of maxParallelForks

Benchmark results — Gradle Profiler



Historical | Sorted | Select all | Clear all | Show all | Show selected only

Scenario	Baseline	Sample	Mean	Min	P25	Median	P75	Max	Std.dev
+ testAll 05 maxForks	☐	total execution time	552,524.02 ms	545,260.00 ms	549,680.43 ms	552,798.53 ms	555,118.52 ms	559,762.58 ms	4,900.11 ms
+ testAll 10 maxForks	☐	total execution time	601,205.85 ms	596,888.58 ms	598,016.48 ms	598,875.93 ms	604,558.21 ms	607,690.06 ms	4,183.85 ms
+ testAll 15 maxForks	☐	total execution time	643,271.15 ms	640,838.27 ms	640,934.84 ms	641,030.80 ms	641,245.88 ms	652,305.97 ms	4,519.43 ms
+ testAll 20 maxForks	☐	total execution time	669,982.77 ms	658,169.70 ms	663,252.53 ms	665,395.56 ms	671,730.70 ms	691,365.39 ms	11,542.61 ms
+ testAll 23 maxForks	☐	total execution time	692,498.26 ms	690,834.21 ms	690,981.19 ms	691,290.08 ms	694,115.28 ms	695,270.53 ms	1,834.69 ms

Unit Test Performance

Current build infrastructure benefited from single fork

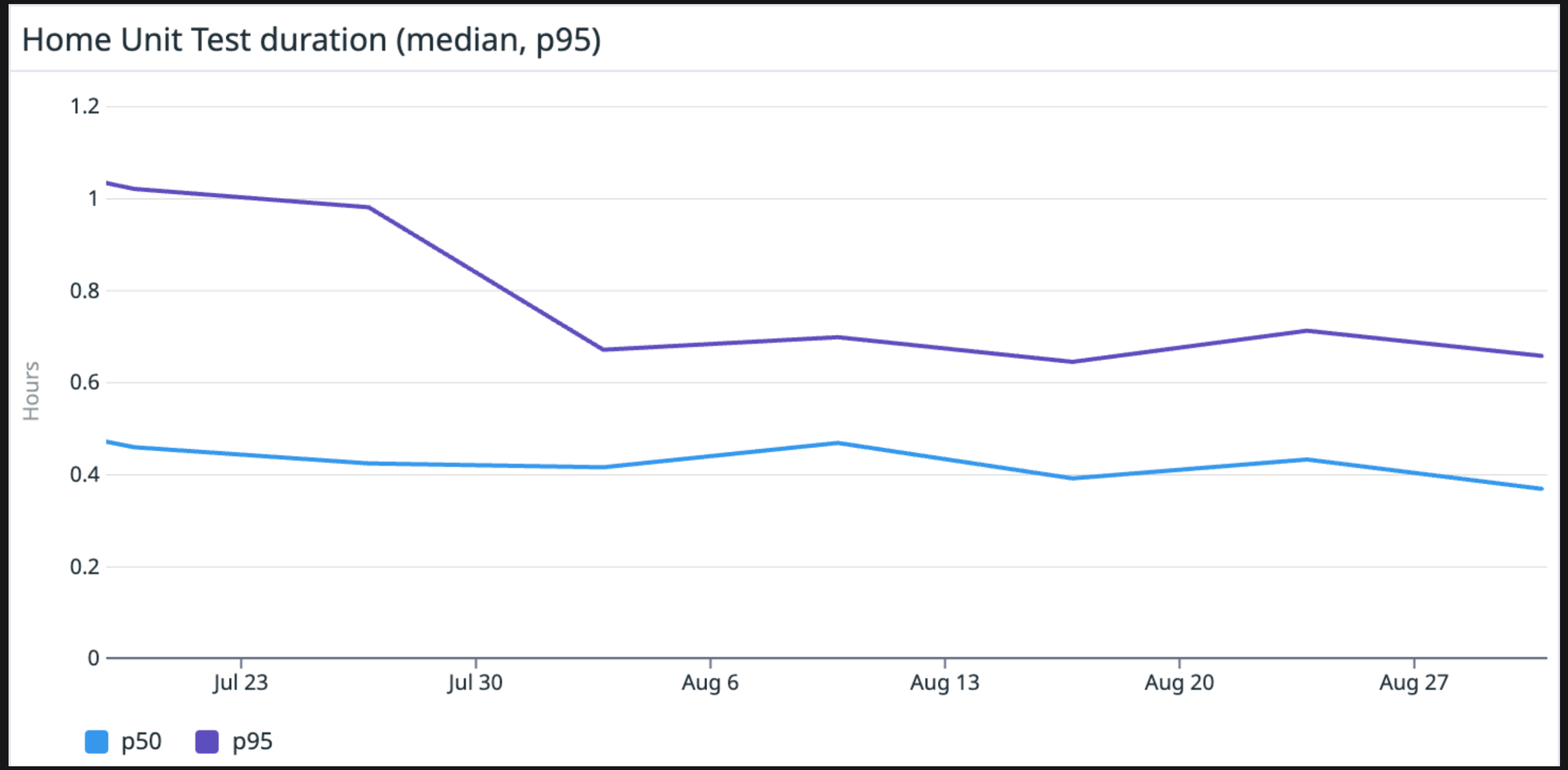
```
tasks.withType<Test>().configureEach {
    // more than 1 fork causes memory pressure on CI and longer test times
    maxParallelForks = 1
}
```

Unit Test Performance

Significant decrease in unit test duration on PR jobs

p95
58m → 40m

p50
25m → 24m



Eliminating high cost, low value jobs

Eliminating high cost, low value jobs

Legacy codebase accumulates jobs over time

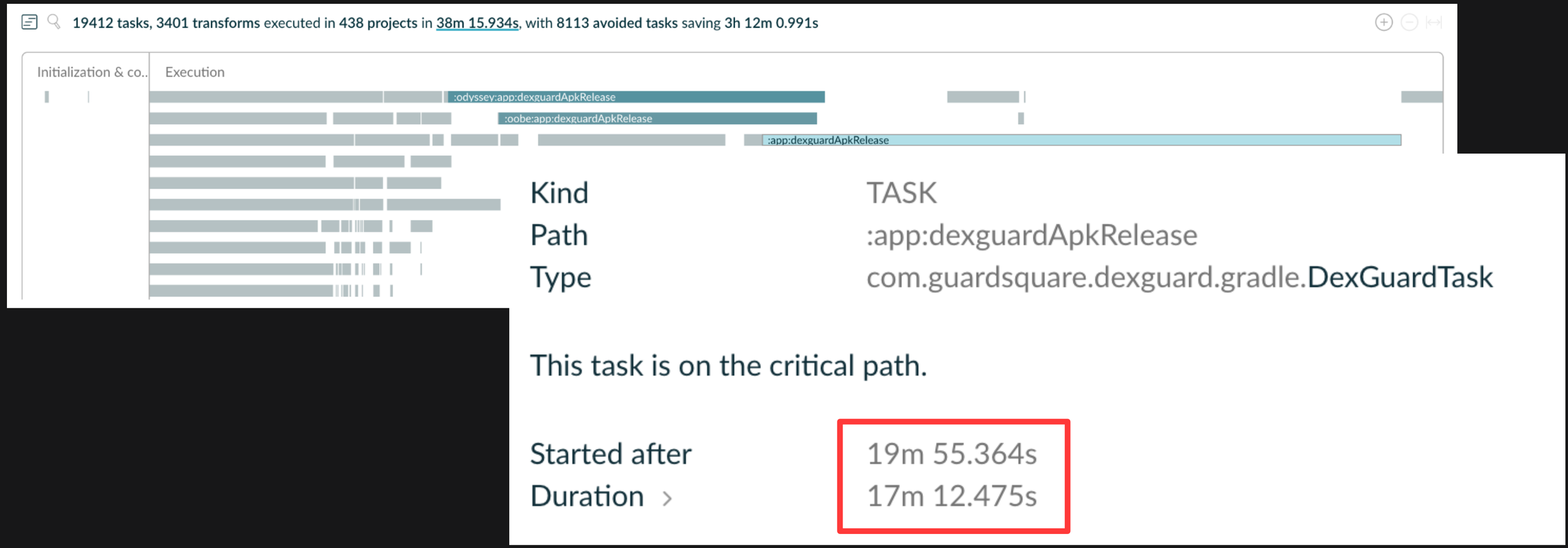
- **Take inventory of all jobs and tasks run on PRs**
  Many jobs added may no longer have the same value

- **Identify high cost, low value jobs**
  Long running jobs that have a low chance of breaking on any single commit

Building obfuscated release builds on PRs

Scans showed DexGuard task was very time consuming

- **Obfuscating release builds was 30-50% of total PR build time**
  Very rarely would a release build fail compilation on CI

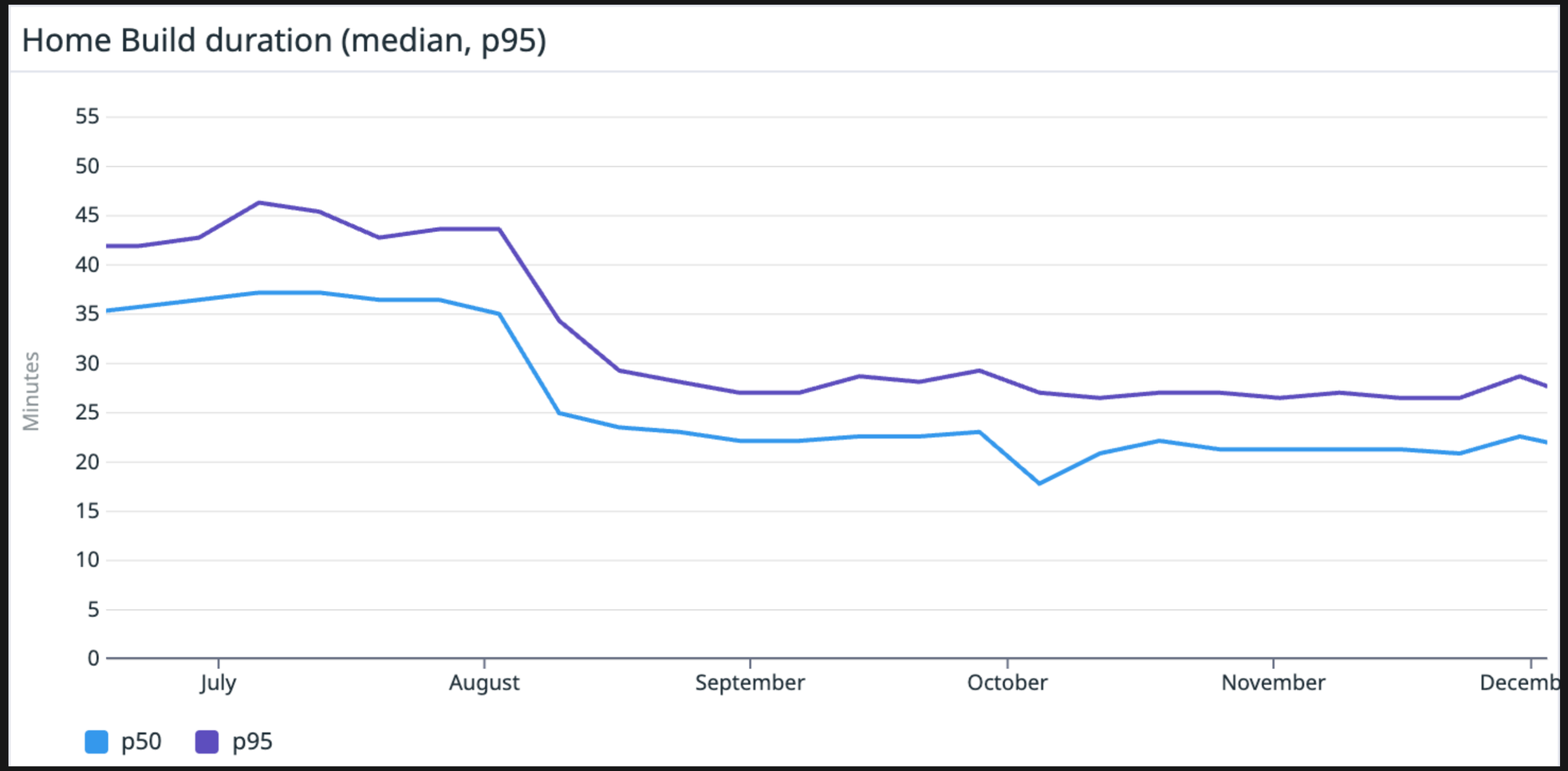


Building obfuscated release builds on PRs

Results without DexGuard task running

p95
44m → 34m

p50
35m → 25m



Identifying slow tests

Identifying slow tests

Investigation into unit test tasks with longest times

Test	Outcome	Total time	Own time	Serial time
Showing 1-200 out of 14512 total items «First page ‹Previous Next› Last page»				
:home-library:testDebugUnitTest ›	PASSED	8m 49.139s	4.418s	8m 49.139s
:performance:testDebugUnitTest ›	PASSED	6m 31.204s	2.456s	6m 31.204s
:activation:testDebugUnitTest ›	PASSED	5m 58.001s	3.134s	5m 58.001s
:leaderboard:ui:testDebugUnitTest ›	PASSED	3m 20.393s	2.004s	3m 20.393s
:class-details-v2:ui:testDebugUnitTest ›	PASSED	3m 16.446s	2.419s	3m 16.446s
:companion-devices:hestia:testDebugUnitTest ›	PASSED	2m 48.179s	2.110s	2m 48.179s
:companion-devices:ui:testDebugUnitTest ›	PASSED	2m 17.464s	2.306s	2m 17.464s
:metrics:ui:testDebugUnitTest ›	PASSED	2m 16.671s	2.066s	2m 16.671s
:hardware-control:testDebugUnitTest ›	PASSED	2m 15.989s	2.519s	2m 15.989s
:training-programs:testDebugUnitTest ›	PASSED	2m 0.815s	2.088s	2m 0.815s
:profile-settings:main:testDebugUnitTest ›	PASSED	1m 51.349s	2.915s	1m 51.349s
:dynamic-scenic:core:testDebugUnitTest ⌄	PASSED	1m 45.137s	1.977s	1m 45.137s

Identifying slow tests

Understanding the value and cost of Robolectric test

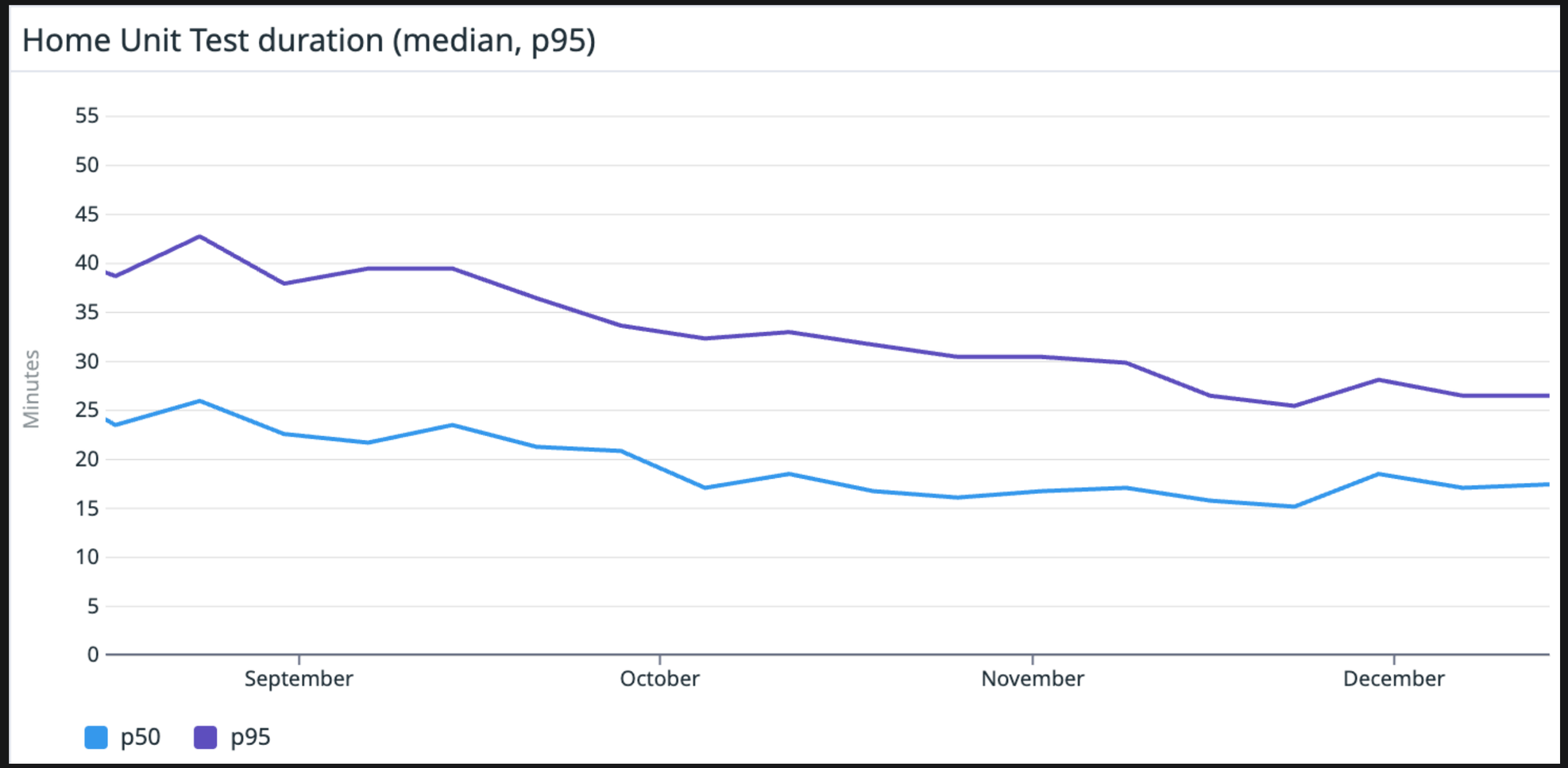
- **Robolectric tests were mostly redundant and no longer high value**
  Added to codebase before we had a consistent and testable architecture
- **Small number of Robolectric tests took significant runtime**
  2% of tests were Robolectric yet they accounted for 40% of test time

Identifying slow tests

Results from fully removing all Robolectric tests

p95
41m → 26m

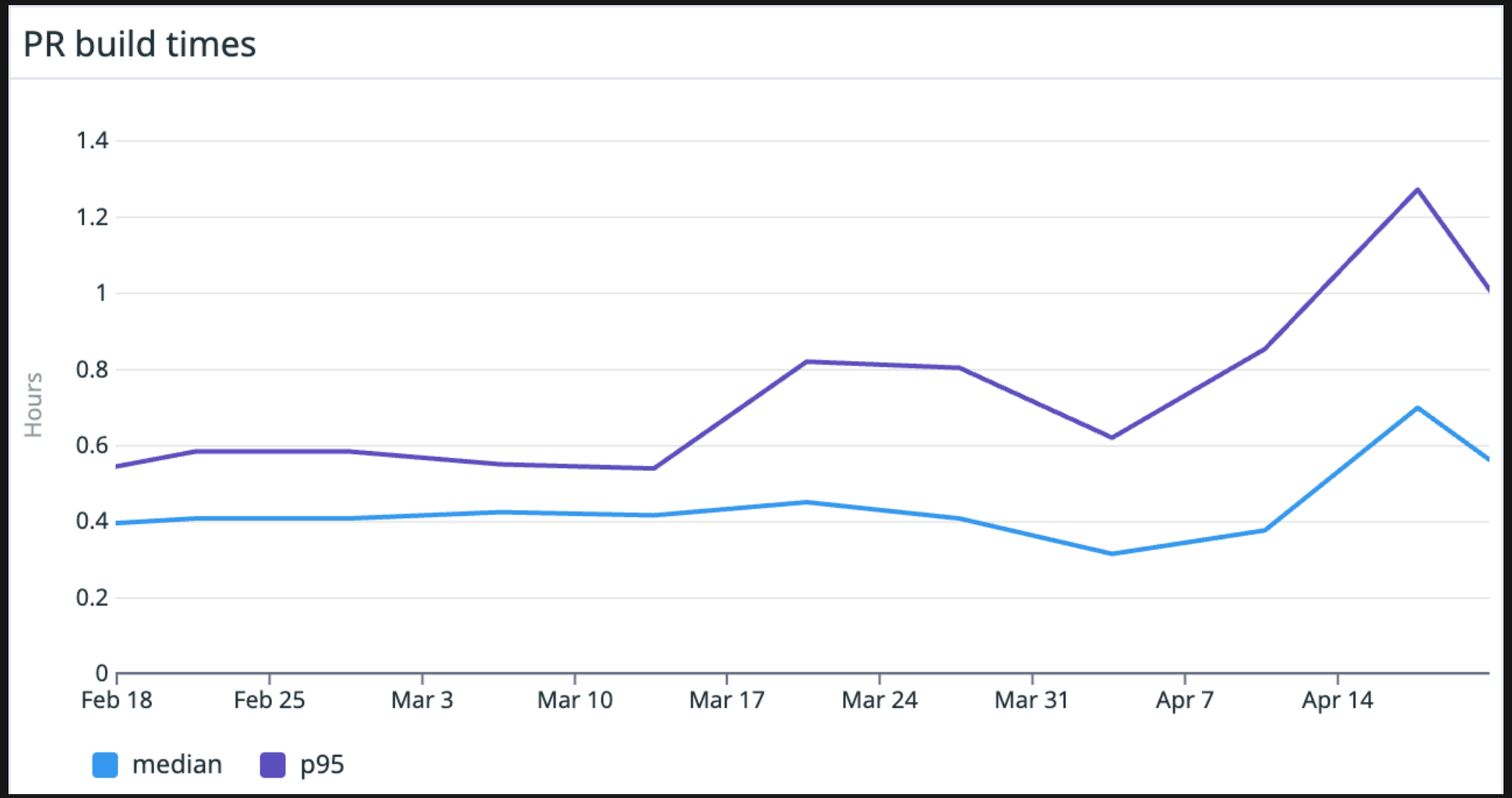
p50
28m → 17m



AWS infrastructure changes

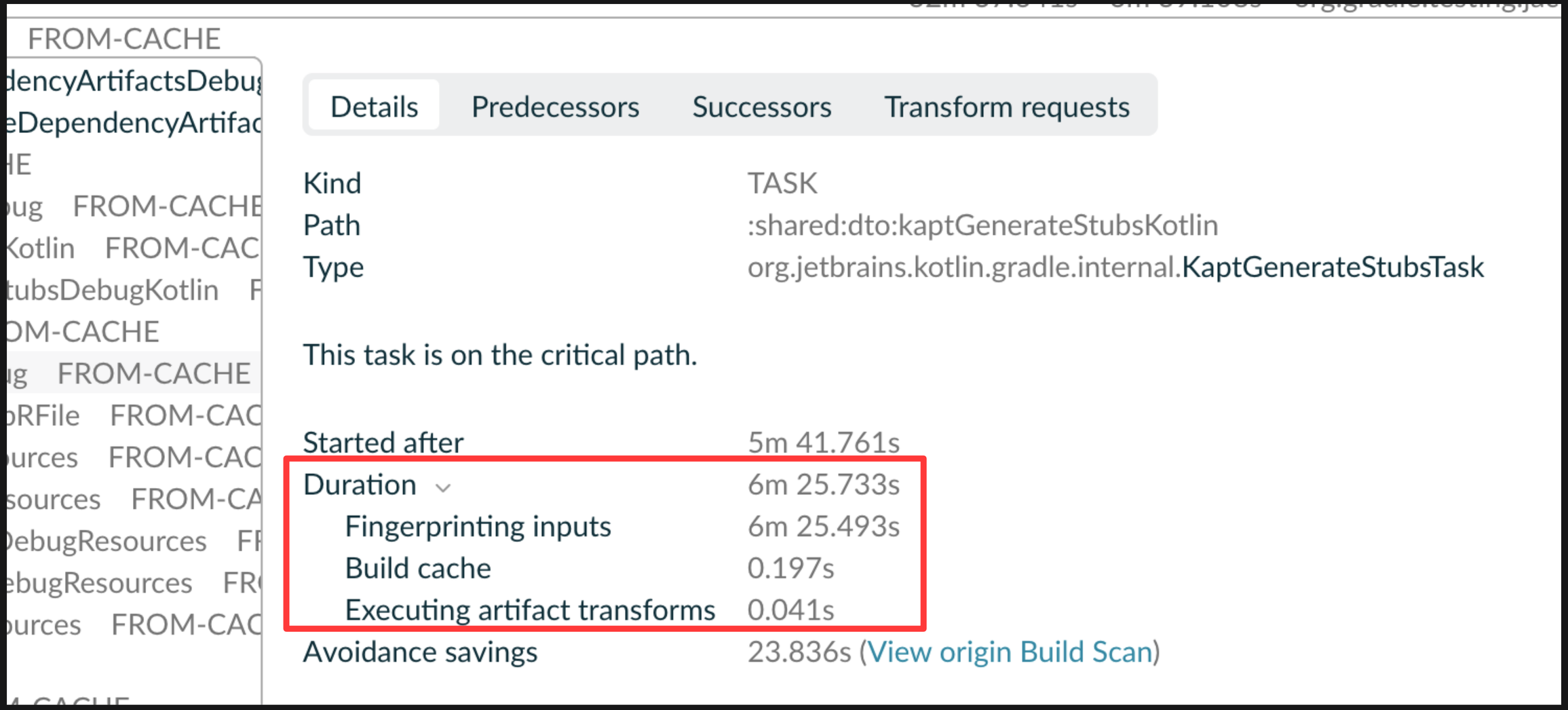
AWS infrastructure

Infrastructure change caused spike in PR times



AWS infrastructure

500x increase in fingerprinting inputs indicates I/O bottleneck



AWS infrastructure

Move builds to NVMe disks for higher IOPS

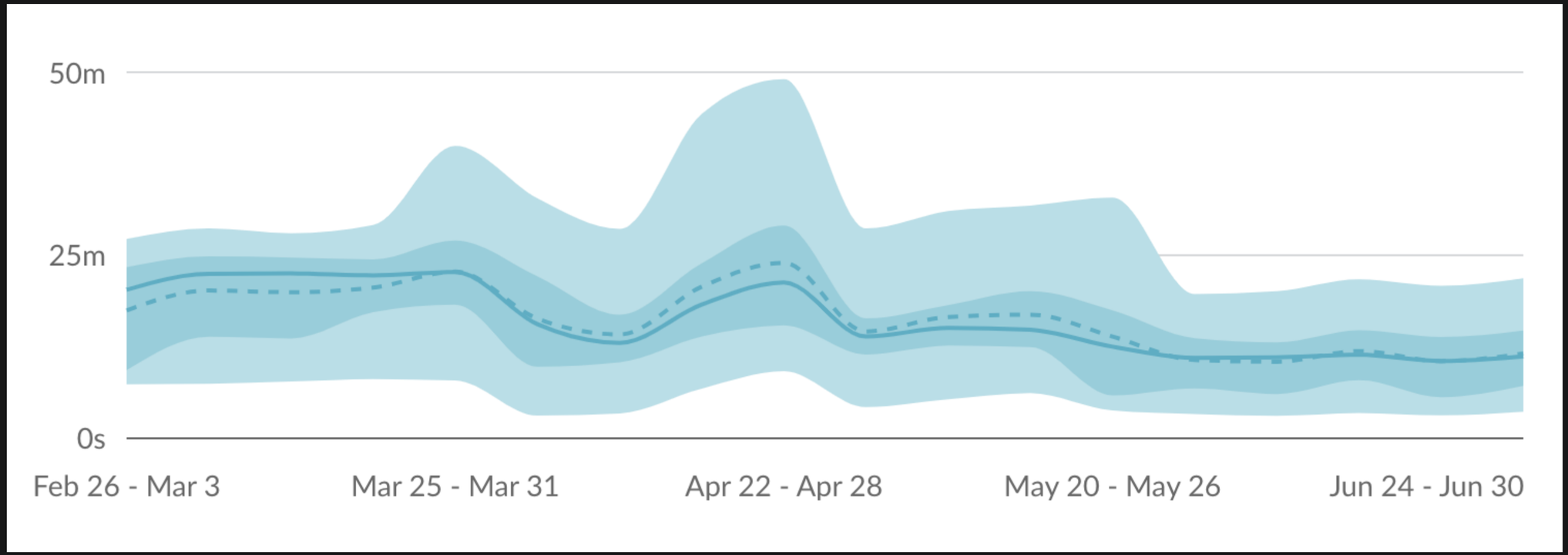
Instance Size	vCPU	Memory (GiB)	Instance Storage (GB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Mbps)
m5a.8xlarge	32	128	EBS Only	Up to 10	4,750
m5a.12xlarge	48	192	EBS-Only	10	6,780
m5a.16xlarge	64	256	EBS Only	12	9,500
m5a.24xlarge	96	384	EBS-Only	20	13,570
m5ad.8xlarge	32	128	2 x 600 NVMe SSD	Up to 10	4,750
m5ad.12xlarge	48	192	2 x 900 NVMe SSD	10	6,870
m5ad.16xlarge	64	256	4 x 600 NVMe SSD	12	9,500
m5ad.24xlarge	96	384	4 x 900 NVMe SSD	20	13,570

AWS infrastructure

Significant improvement for largest project build task

p95
27m → 21m

p50
20m → 11m



The Results

More improvements

Additional investments drove build times down even more

- **Utilizing Develocity's Predictive Test Selection**
- **Added Develocity's Test Distribution**
- **Prefetch dependencies daily for ephemeral CI runners**
- **Removed Dexguard in favor of R8**

Persistence

PR build times in September 2024 vs July 2023

p95

~~64m~~ → 30m

p50

~~44m~~ → 16m

PELOTON

Thank You