



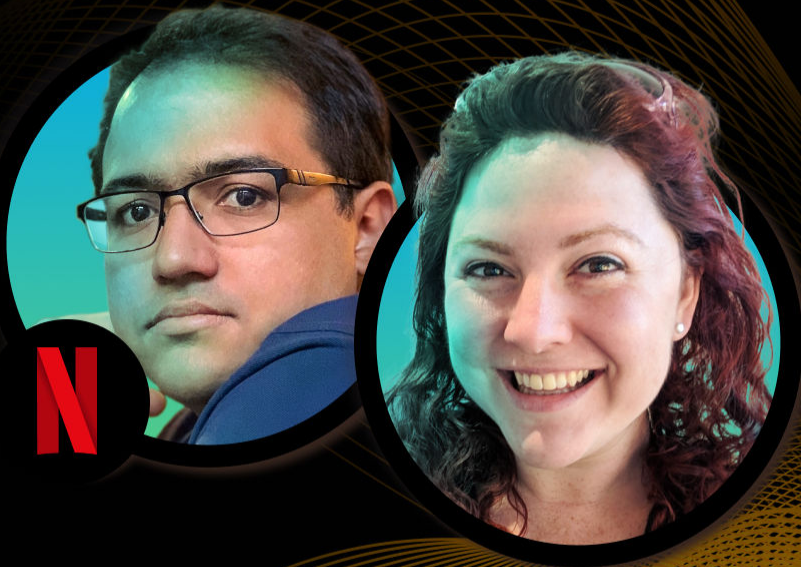
DPE
SUMMIT

📍 San Francisco, CA
📅 September 20-21

How Improving the Testing Experience Goes Beyond Quality: A Dev Productivity Point of View

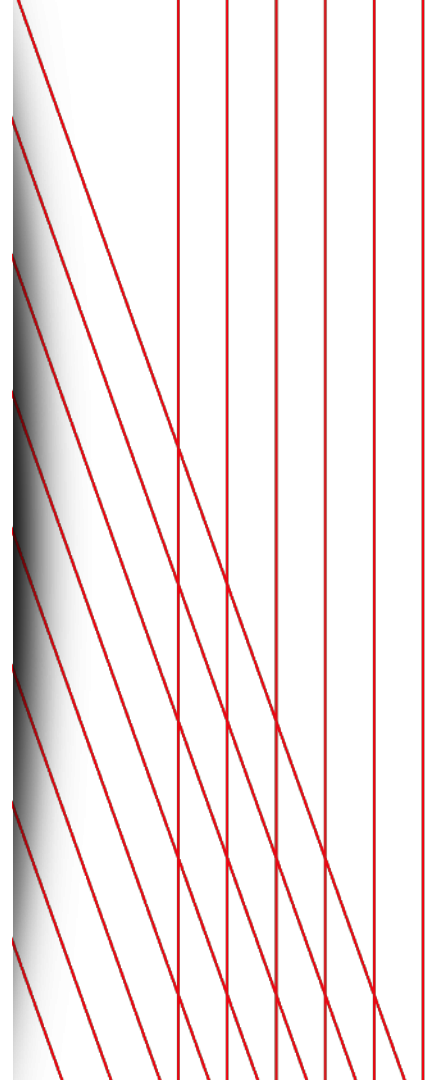
Roberto Perez Alcolea & Aubrey Chipman

Developer Productivity Team
Netflix



Agenda

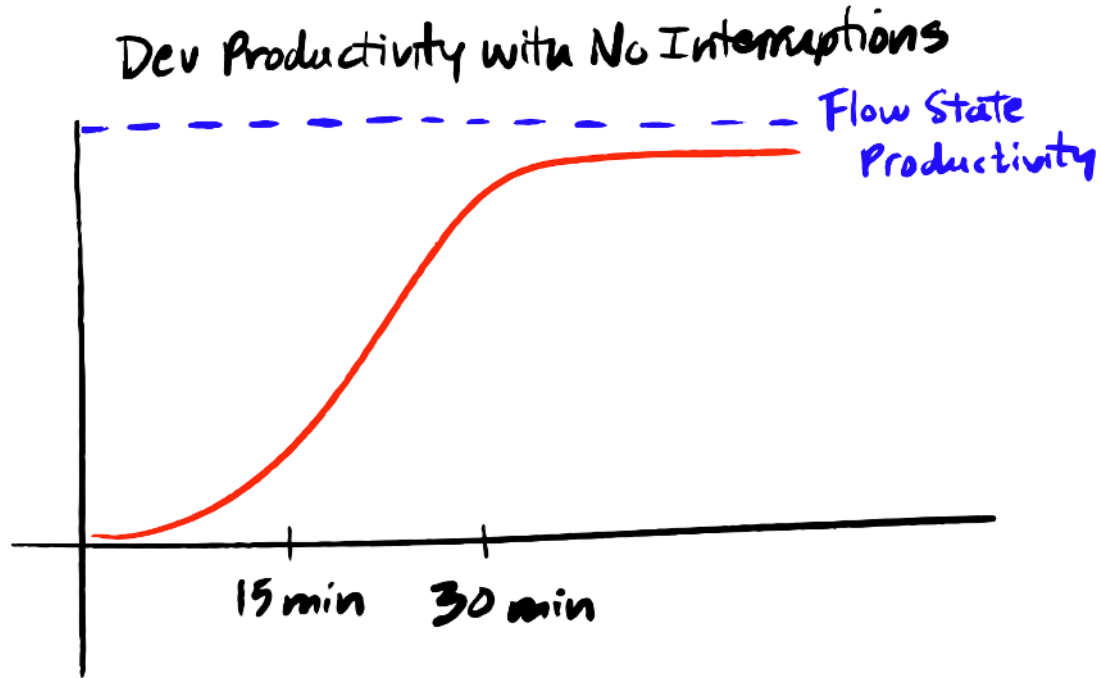
- Why improving the Testing Experience?
- Day in life of a developer
- Netflix JVM Build Landscape
- Common problems in the software testing experience
- Improving the testing experience for developers
- Learnings along the way
- Q&A



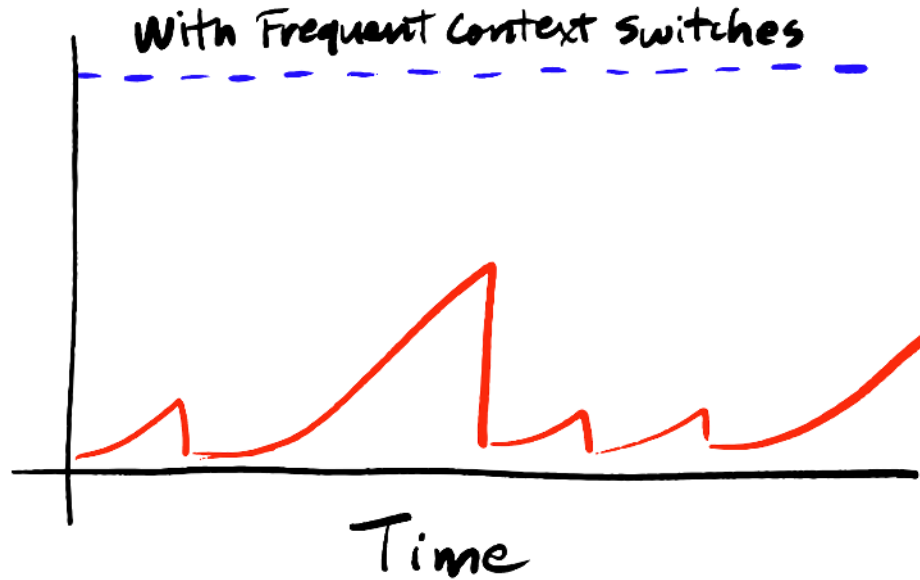


Why focus on improving the Testing Experience?

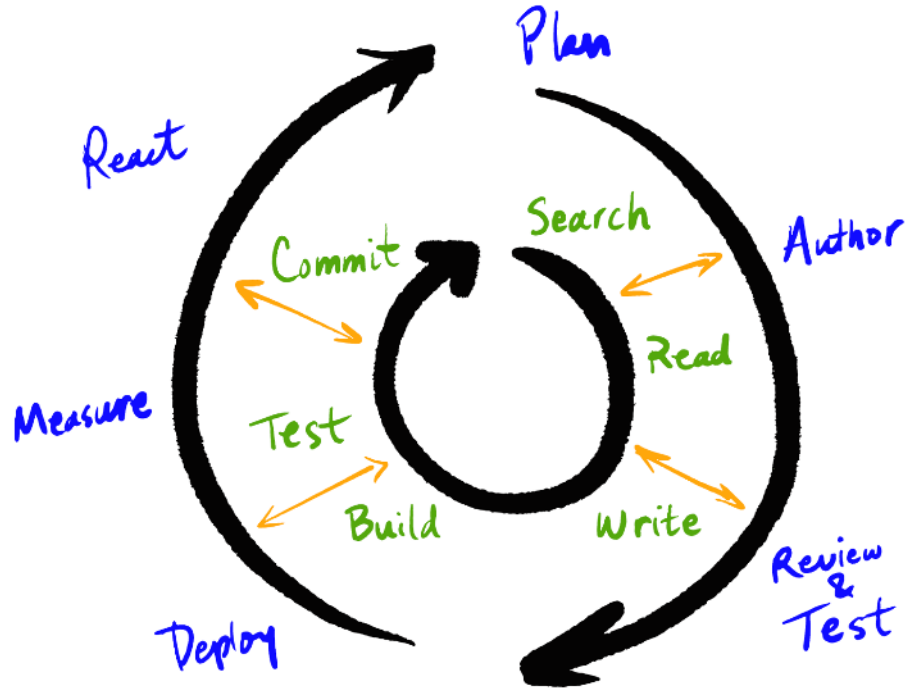
Flow state



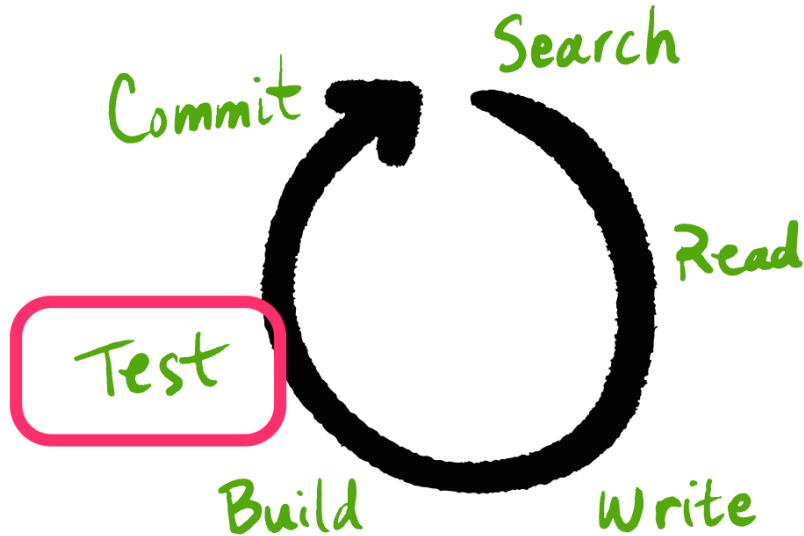
Mental context switches = out of Flow state



Developer Inner Loop, Outer Loop



Developer Inner Loop





**Testing is crucial in the inner
loop and suboptimal
experience leads to context
switching and losing flow state**





Day in the life of a developer

(Low productive environment...)

Tuesday Afternoon



```

package test.library;

import com.google.common.collect.Lists;
import java.util.List;
2 usages

public class Library {
1 usage
    private final List<String> fruits = Lists.newArrayList(
        ...elements: "orange", "banana", "kiwi", "mandarin");
1 usage
    public boolean hasFruit(String fruit) {
        return fruits.contains(fruit);
    }
}

```

Build	Status	Author
#265	passed	by ctobor
#264	passed	by mulson
#263	passed	by rlfeser
#262	passed	by gordon
#261	passed	by rlfeser
#260	passed	by go/dori
#259	passed	

Build #280

Build details

Build #280: Passed about 29 mins ago

Commit: 037d1cc by go/dori

Message: Merge pull request #83 in SPKR/ggr nfx from fixOutput to master
Squashed commit of the following commits:
#280: 7f9190d: **fixOutput**
#280: 4e911e: **fixOutput**
#280: 4e911e: **fixOutput**
#280: 4e911e: **fixOutput**

Author: Go/Dori

Change return type from strings to map for build's output

Build output

```

system) merged: PM 033 taken: master (037d1cc) by go/dori
...

```

Gradle Enterprise

test-library build Sep 21, 2022 5:42:52 PM PDT

- Summary
- Console log**
- Failure
- Deprecations
- Timeline
- Performance
- Tests
- Projects
- Dependencies
- Build dependencies
- Plugins
- Custom values

14 lines View raw Download raw

```

:lib:compileJava
:lib:processResources NO-SOURCE
:lib:classes
:lib:jar
:lib:assemble
:lib:compileTestJava
:lib:processTestResources NO-SOURCE
:lib:testClasses
:lib:test
:lib:check
:lib:build

```

BUILD SUCCESSFUL in 786ms

4 actionable tasks: 4 executed

TEST keeldemo-prestaging 1 ▲ / 1 = : 50%

US-WEST-2

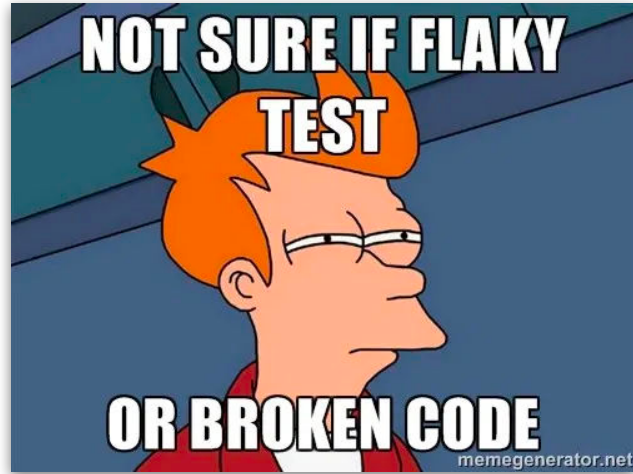
Build	Status	Test Results
V075: Build: #44	Success	1 ▲ / 1 = : 100%
V074: Build: #44	Failure	1 = / 0 = : 0%

A red arrow points to the test result icon for build V075.

Wednesday Morning



The developer:





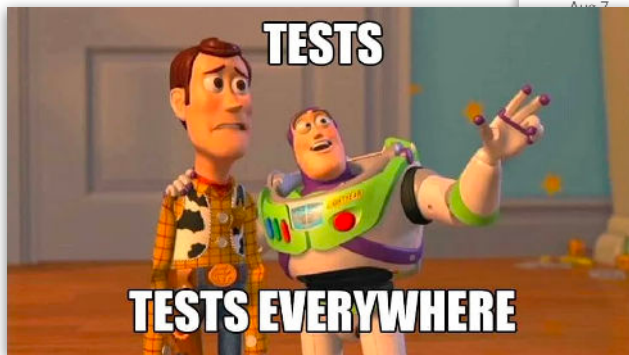
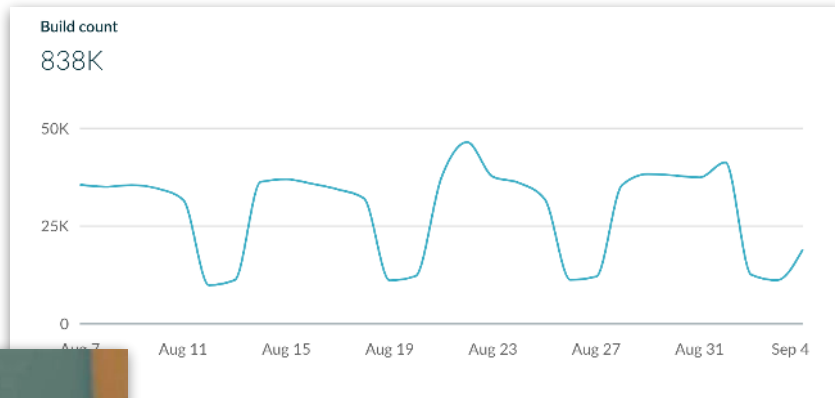
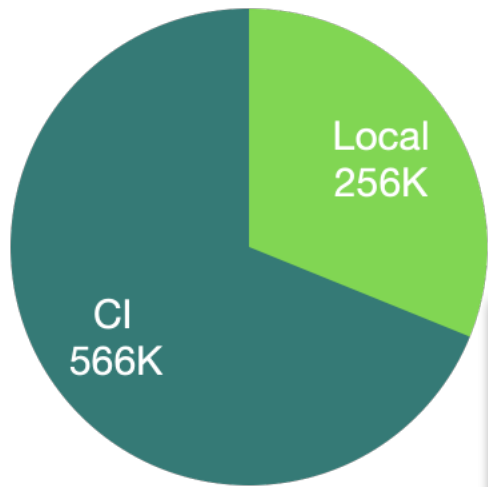
The developer doesn't achieve much, is frustrated and unmotivated



Netflix JVM Build Landscape

Builds executing tests

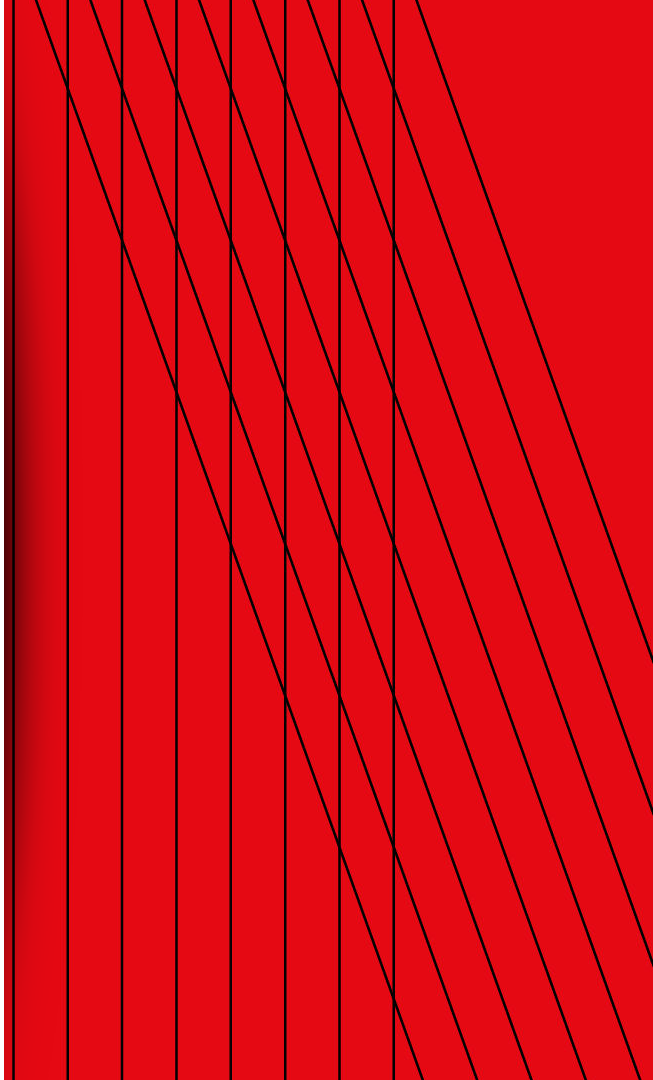
Per 28 days



14 Million test cases executed in 28 days

**Let's focus
on testing...**

**at the
project level**

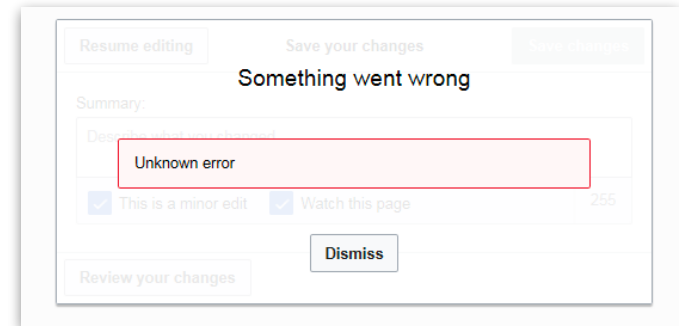




Common problems in the software testing experience

Testing becomes difficult

- Hard to write tests with provided tools
- Lack of actionable feedback from outputs
- Test suites evolution
- Lack of documentation or examples

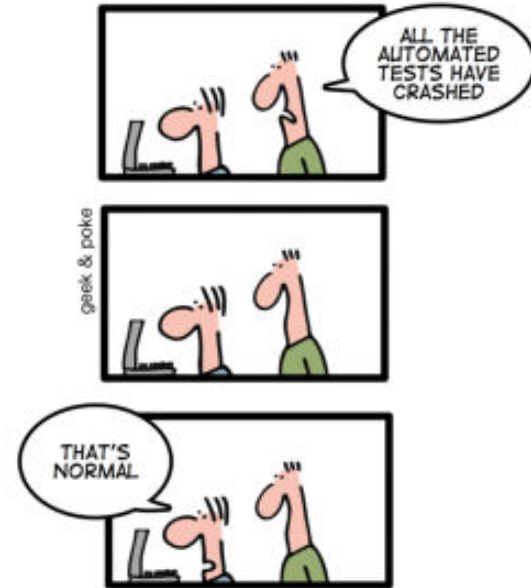


There are tests, but they are flaky

Flaky tests fail to produce the same outcome with each individual test run.

Potential reasons:

- Asynchronous waits, concurrency
- Test Order Dependency
- Poorly modeled tests



Results of having flaky tests

This can and will frustrate developers.
In order to ship a change, folks might:

- Delete the test
 - Ignore the test and might never revisit it
- ... versus
- Identify the flaky test and fix it in the moment

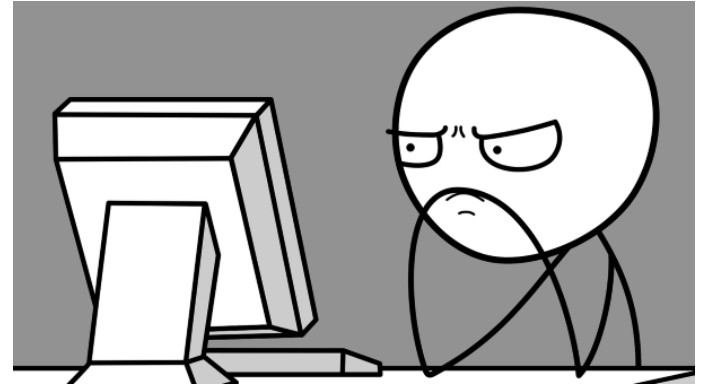
In any case, they might start losing confidence in the test suite



There are tests, but they are slow

Variety in

- Test setup time
- Shared resources
- Parallelization



There are tests, but they are inconsistent

Variety in

- Local Mac machines
- CI Linux machines
- Network and security access



**Are these issues
causing problems
in my
organization?**

**Most likely,
yes!**



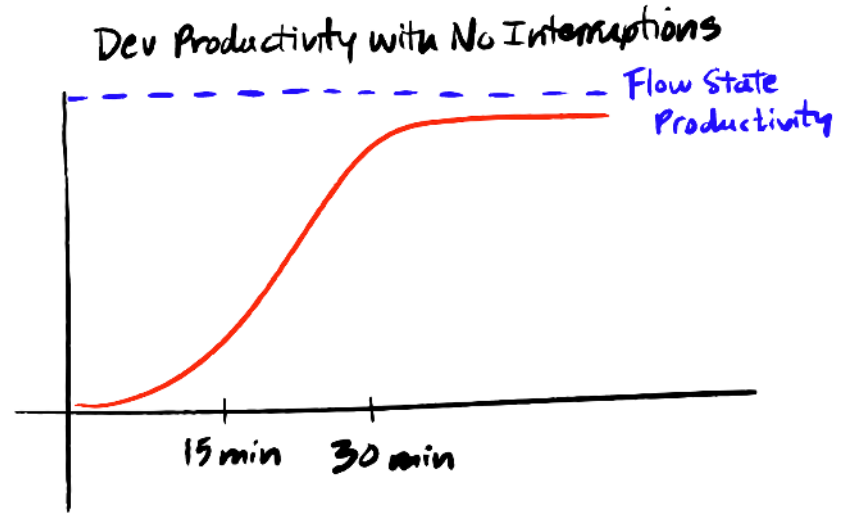
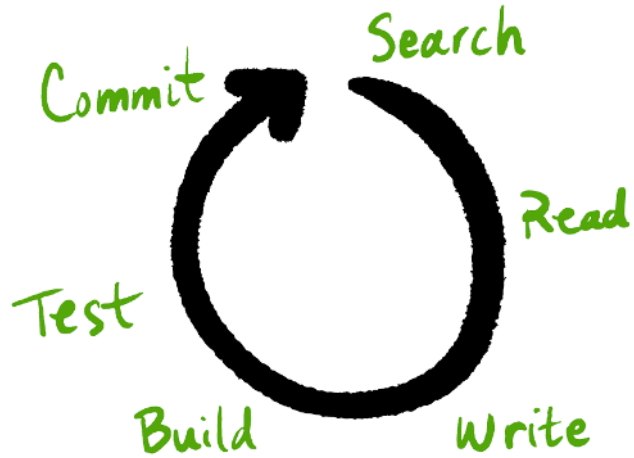
Unwanted situations

Developers will be frustrated and it is possible to fall into the following traps:

- Avoid writing tests
- Ignore or remove tests in order to verify their changes
- Avoid running tests locally and wait for CI job executions to provide feedback for every single small change



Developer Inner Loop



And...

“It is staggering how tolerant engineers are of toil and frustration and friction.”

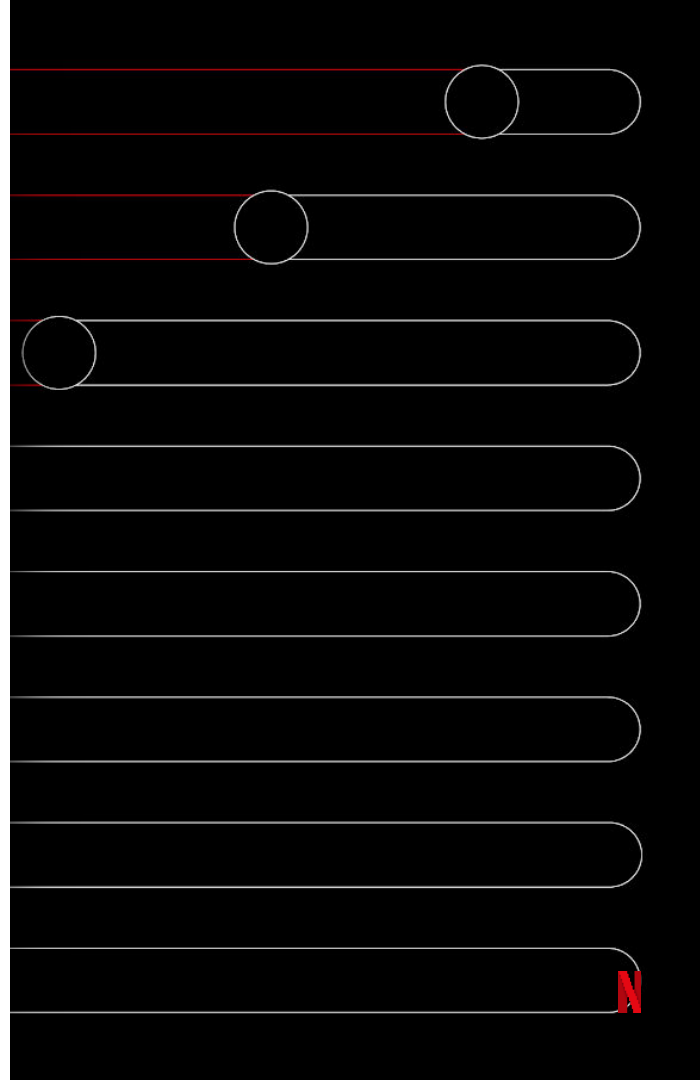
**At Netflix, we are
not immune to that**

**And we have
invested on it...**



Improving the testing experience for developers

Faster test startup

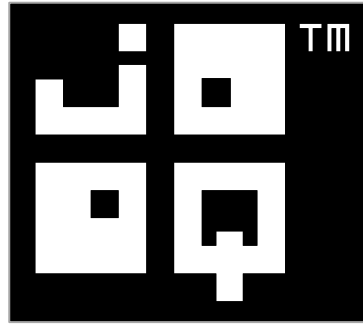




**Let's look at
a real world
example**



Project setup



Great tools!!
but...

Running a single test class was slow

com.netflix.spotlightapi.db.SmokeTest PASSED

:spotlight-api-db-server:smokeTest

28 results in last 7 days, 28 passed [View test history](#)

Execution 1 of 1 PASSED Total / own / serial time 2m 5.062s / 2m 3.334s / 2m 5.062s Started May 18 2023 at 13:49:13 PDT

No test class setup or cleanup failures occurred during this test class execution

Output

Test output is not captured for successful test executions, except for the first successful retry after a prior failure.

Test executions

Test	Outcome	Total time
contextLoads	PASSED	1.728s

The test method is not the problem



The actual bottlenecks

```
: Successfully completed migration of schema "public" to version "34 - addBdpWorkflowEnumType" [non-transactional]
: Schema History table "public"."flyway_schema_history" successfully updated to reflect changes
: Updating lock in Flyway schema history table
: Successfully applied 34 migrations to schema "public", now at version v34 (execution time 00:49.006s)
```

49 seconds applying flyway migrations!

```
: starting up nf-testcontainers for CockroachDB
: Waiting for database connection to become available at jdbc:postgresql://localhost:53791/defaultdb using query

: Container is started (JDBC URL: jdbc:postgresql://localhost:53791/defaultdb)
: Container docker-hub.netflix.net/cockroachdb/cockroach:v22.1.16 started in PT13.001699S
```

13 seconds waiting for crdb to be ready in Docker on M1!

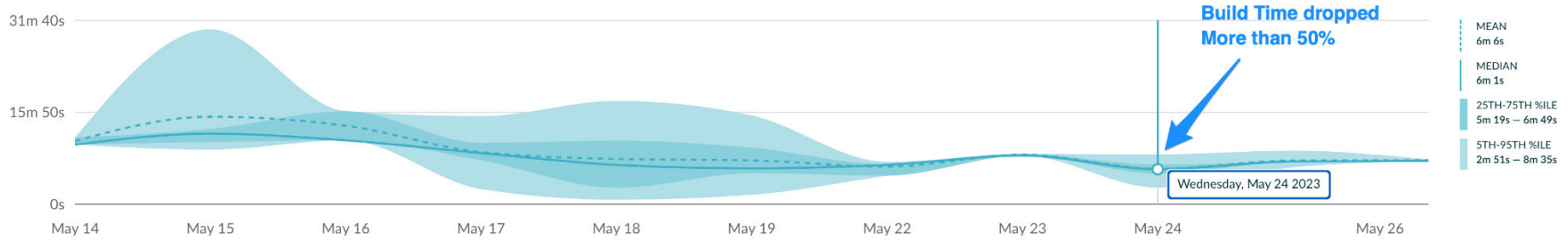
**The tools were not
the problem, but
how we used them!**

Why so slow?

- Many migration files with multiple SQL statements
- Docker in ARM based Mac
- Not reusing containers



Faster tests -> faster builds



:spotlight-api-db-server:smokeTest

28 results in last 7 days, 28 passed [View test history](#)

Execution 1 of 1 PASSED Total / own / serial time: 48.600s / 48.000s / 48.600s Started May 19 2023 at 15:22:12 PDT

No test class setup or cleanup failures occurred during this test class execution

Output

Test output is not captured for successful test executions, except for the first successful retry after a prior failure.

Test executions

Test	Outcome	Total time
contextLoads	PASSED	0.600s

The test method is even faster



What did we change?

- Database migrations baselines
- Testcontainers [singleton pattern](#)
- [Testcontainers Cloud](#)



Other suggestions

- Test slicing (reduced application context and data)
- Context initialization
- Invest on application startup time
 - Better modularization
 - Trim dependencies

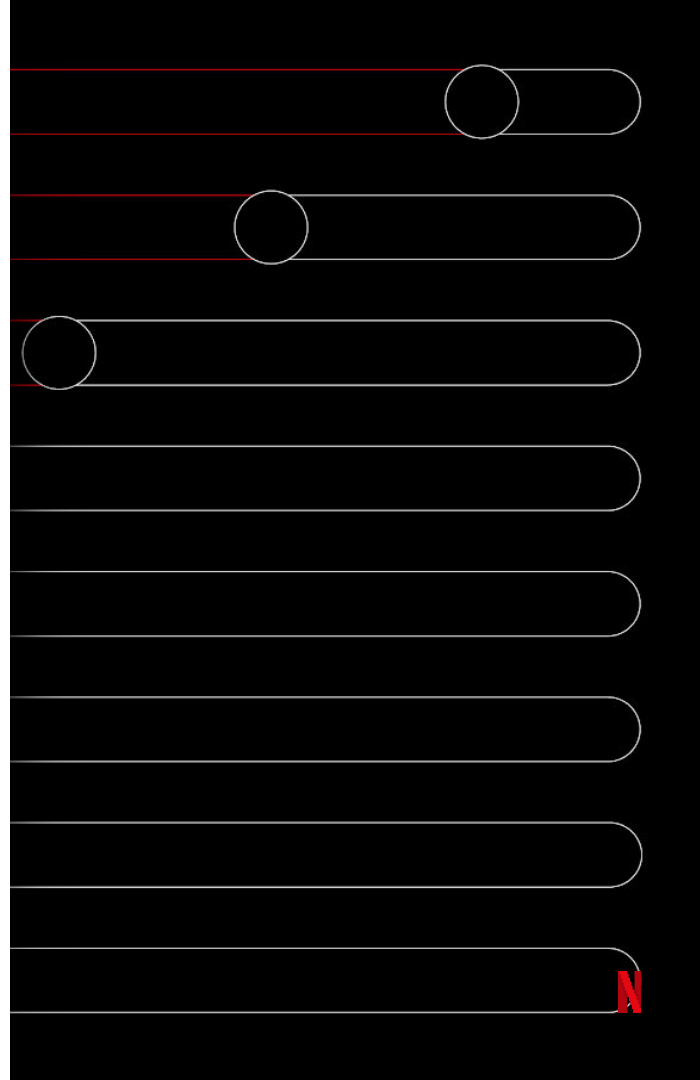




You might find opportunities to standardize tools across teams!

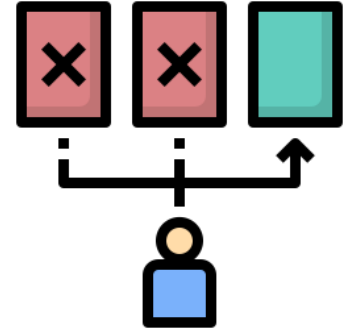


Flaky Test Detection & remediation

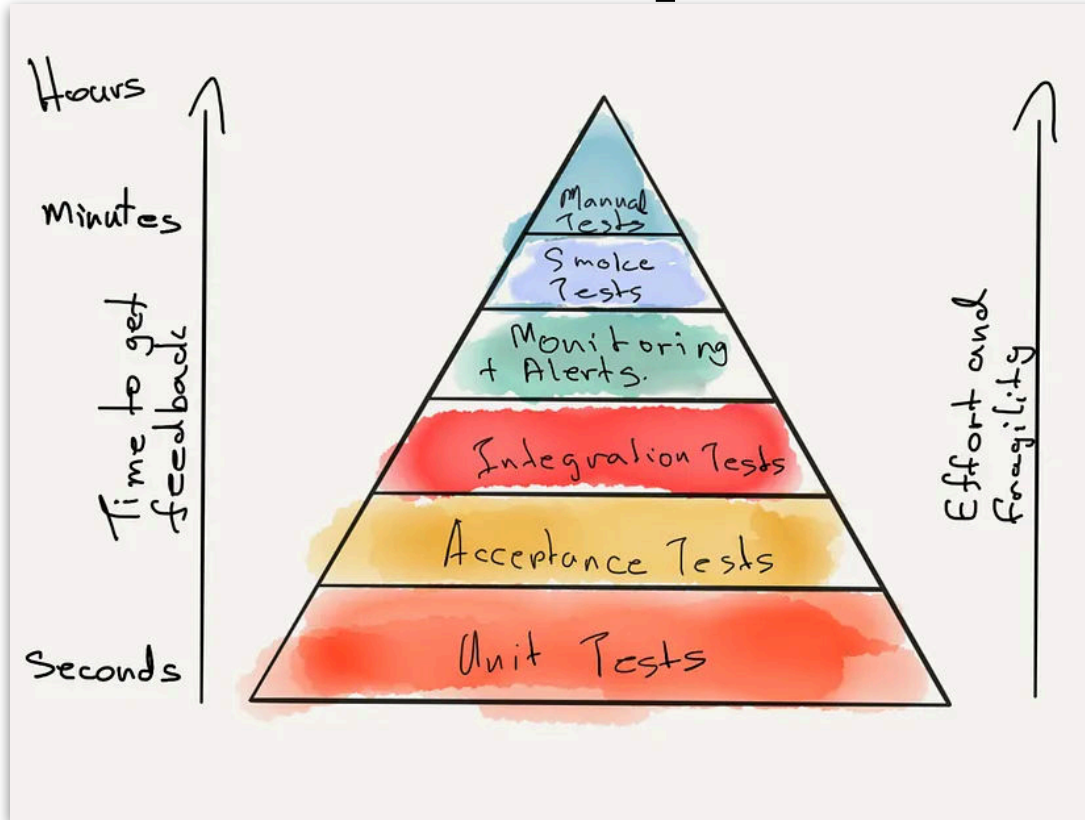


Flaky Test Detection

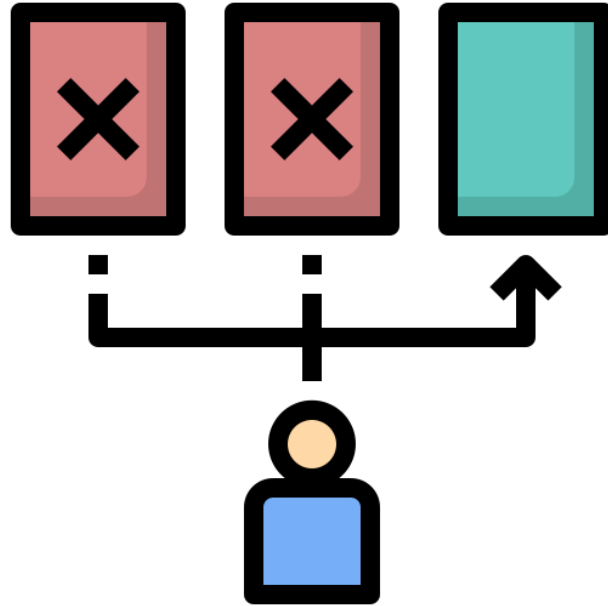
When a test fails, how do we determine if it's flaky or not?



Why is this important?



Flaky Test Visualization

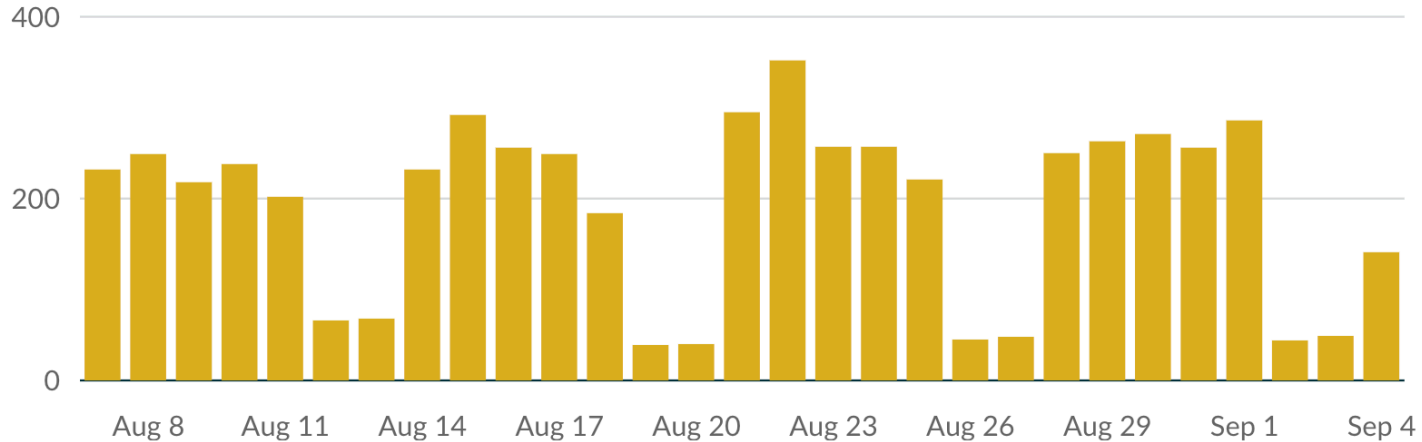


We use Gradle's [Test Retry Gradle plugin](#) with Gradle Enterprise to catch and visualize flakiness.

Flaky Test Detection

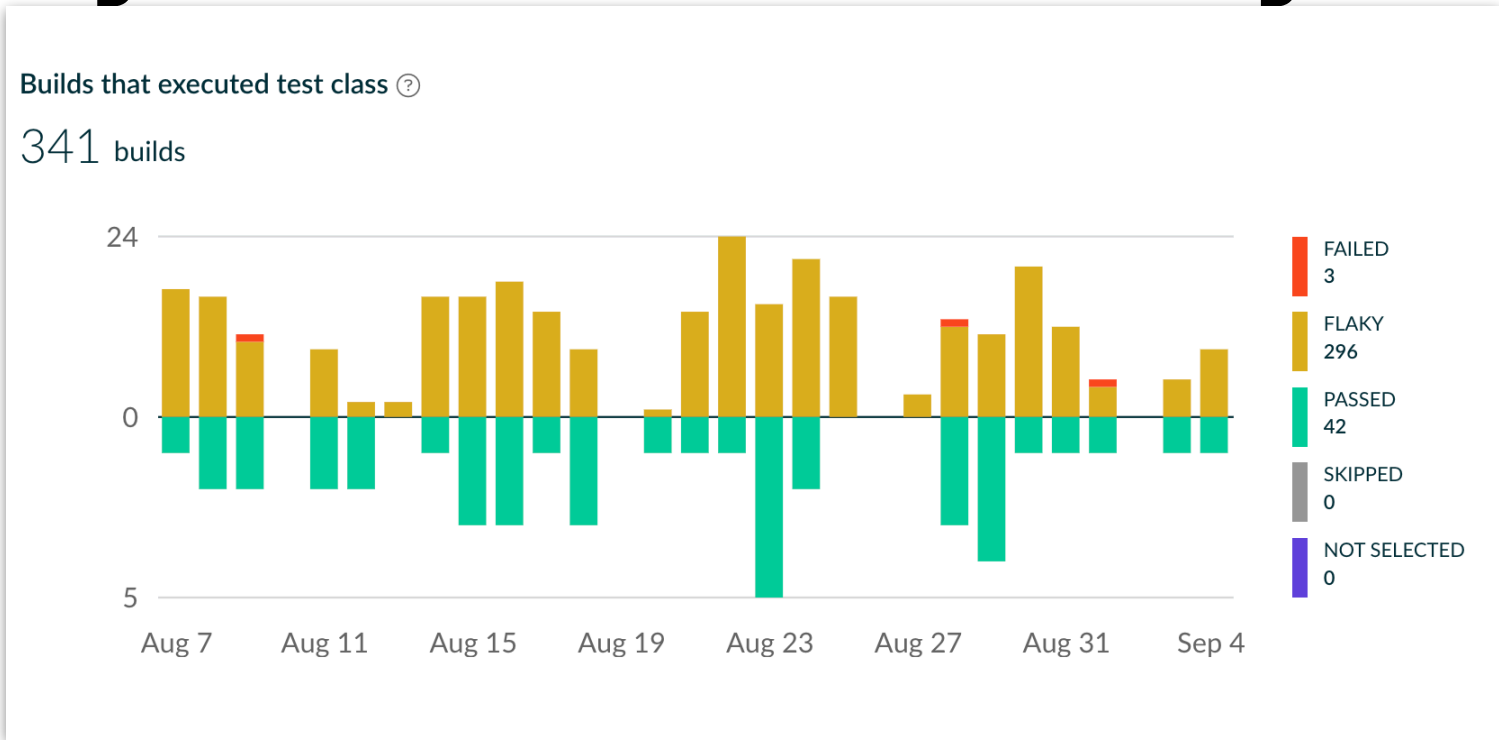
Builds with flaky tests [?](#)

5.60K builds (1% of 469K builds that executed tests)



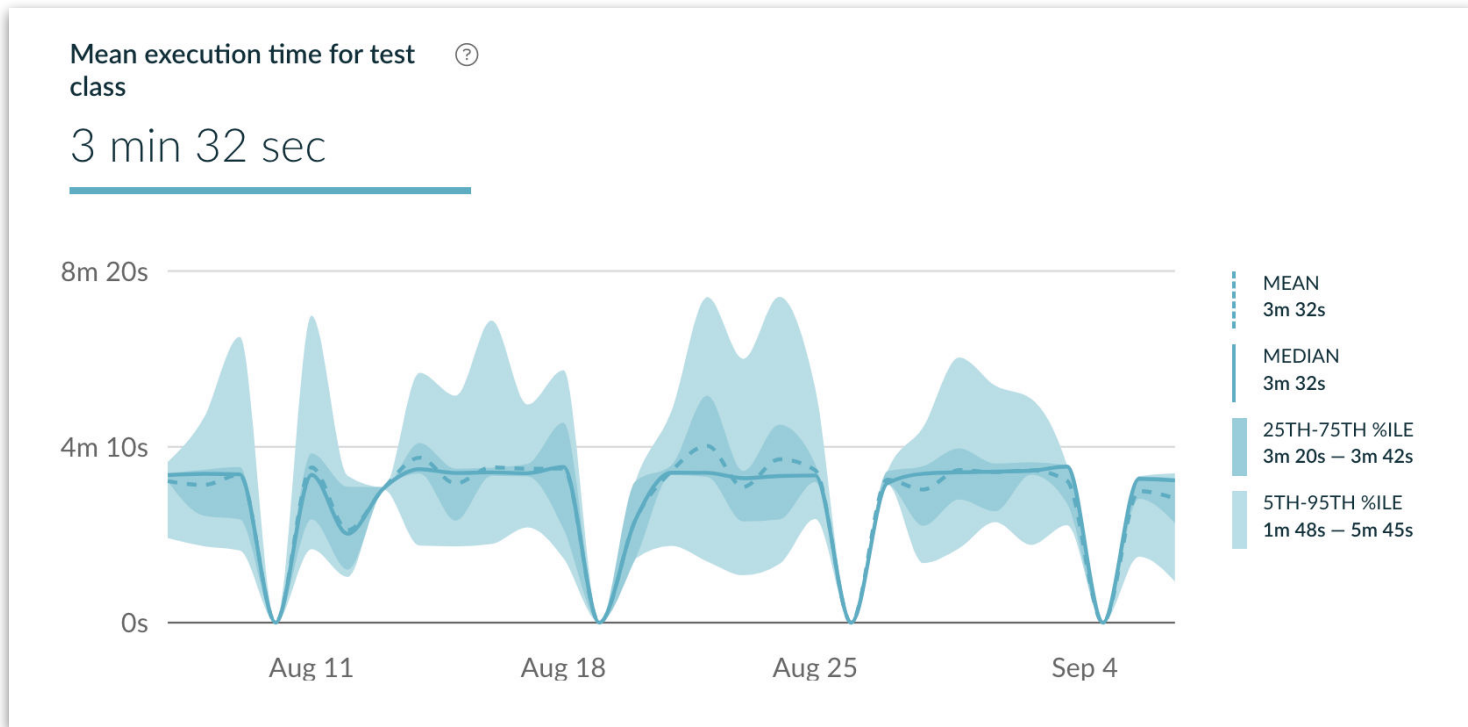
In a month, 5.6k Builds (1%) have flaky tests, not good!

Flaky Test Detection by class



We know which tests are constantly reported as flaky

Flaky Test Detection by class



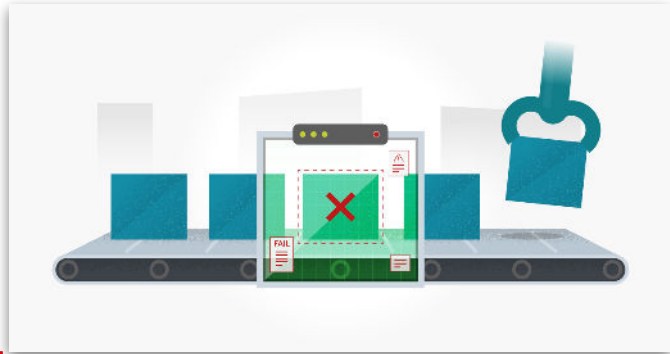
And with how long they usually take to run, this test used over 17 hours for the 296 retries



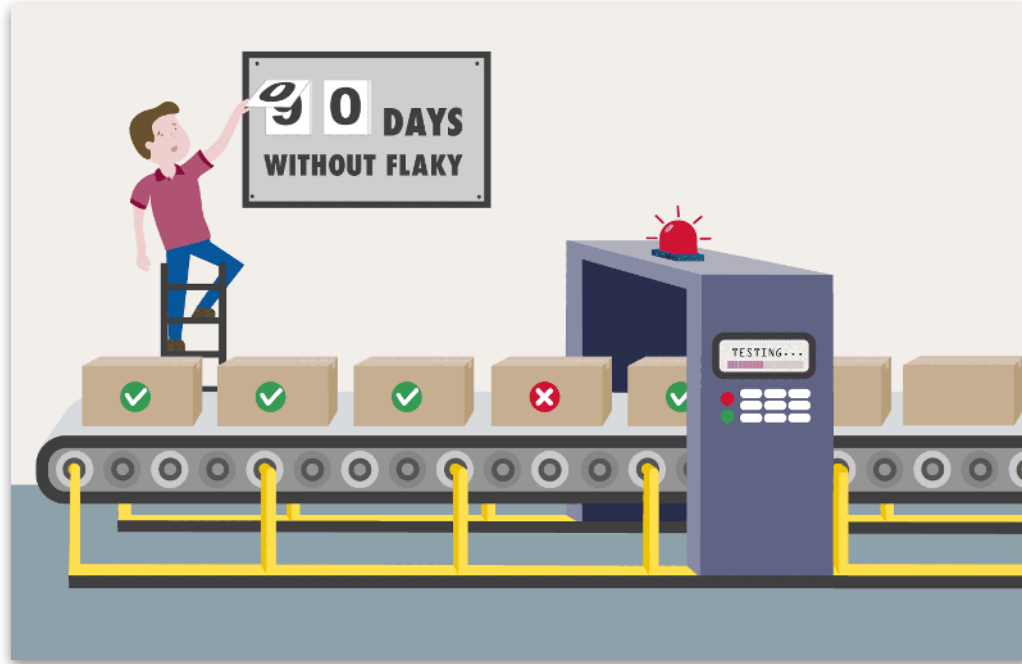
**Detection \neq Solving
the problem 🙄**

What can we do about it?

- Surface this information to project owners on a friendly way
- Culture change
- Quarantine the tests

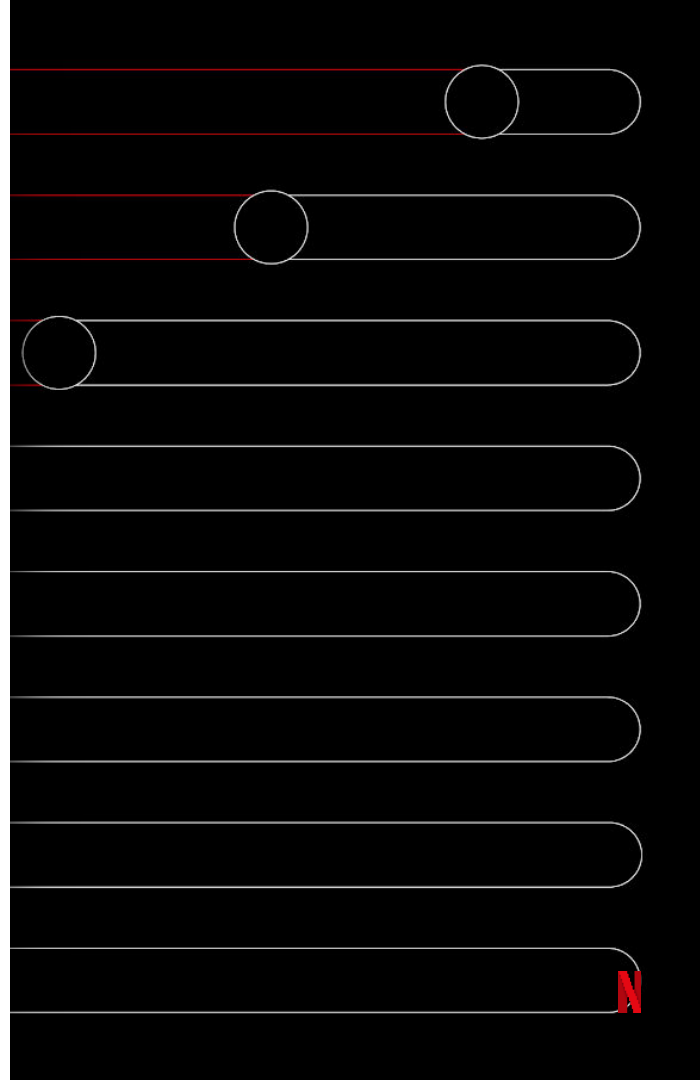


The goal...



Hopefully one day!

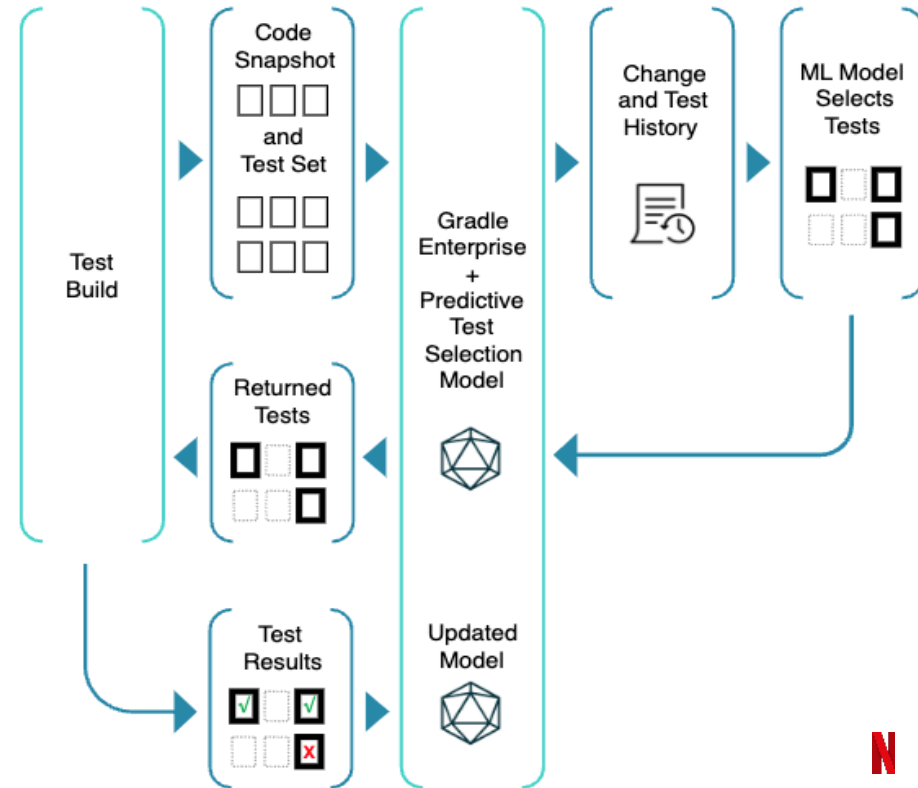
Predictive Test Selection



Predictive Test Selection

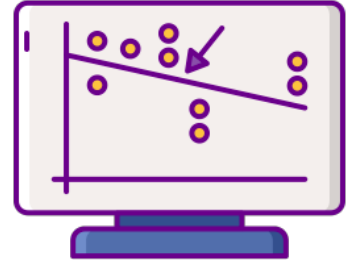
Increases developer productivity by automatically and intelligently selecting and executing the subset of tests that are most relevant to a code change, providing faster feedback.

Popularized by Meta. Read the [paper!](#)



How does PTS work?

- Predictive model by observing code changes and test outcomes from your Build Scan data
- Predictive Test Selection will not attempt to make predictions for test tasks or goals for which fewer than 14 days of code
- Tests will always be chosen if they are recently new, recently changed, recently failed, or recently flaky.



NOTE: We are trading testing comprehensiveness for faster feedback, making it worthwhile for many test executions where reducing feedback time is critical, such as local and pre-merge/pull-request builds

Monthly PTS Simulations for local and CI builds

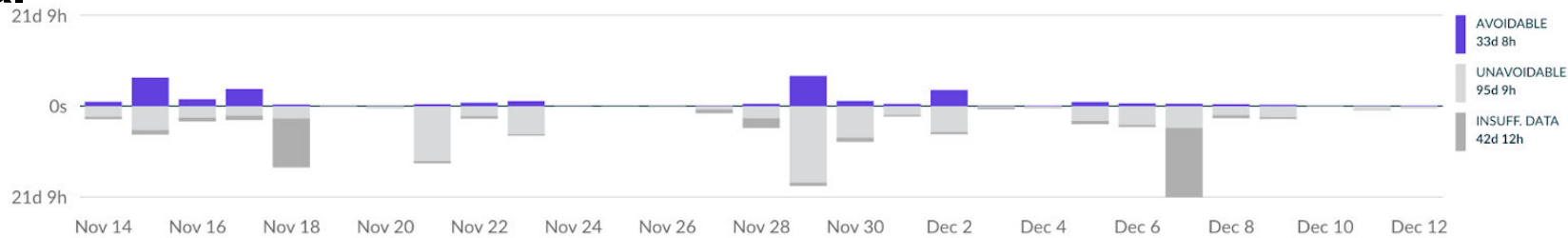
Savings potential ⓘ

33 d 8 hr (19%)

Avoidable tests ⓘ

187K (27%)

Local



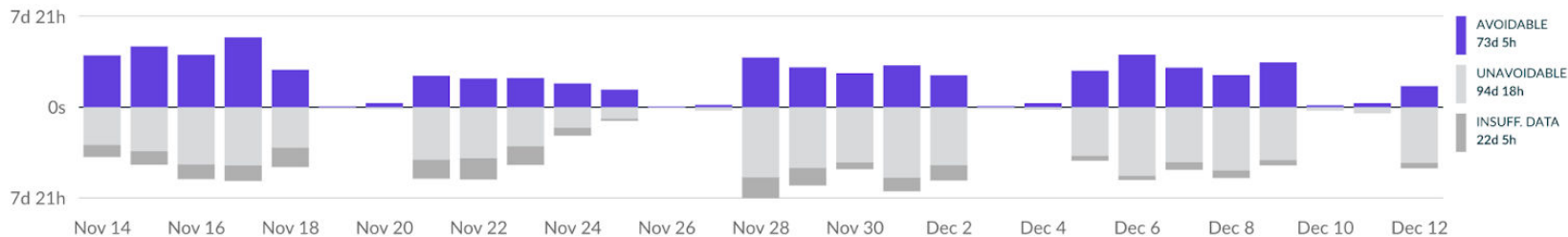
Savings potential ⓘ

73 d 5 hr (38%)

Avoidable tests ⓘ

1.11M (42%)

CI

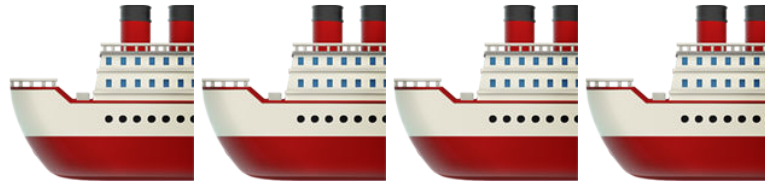




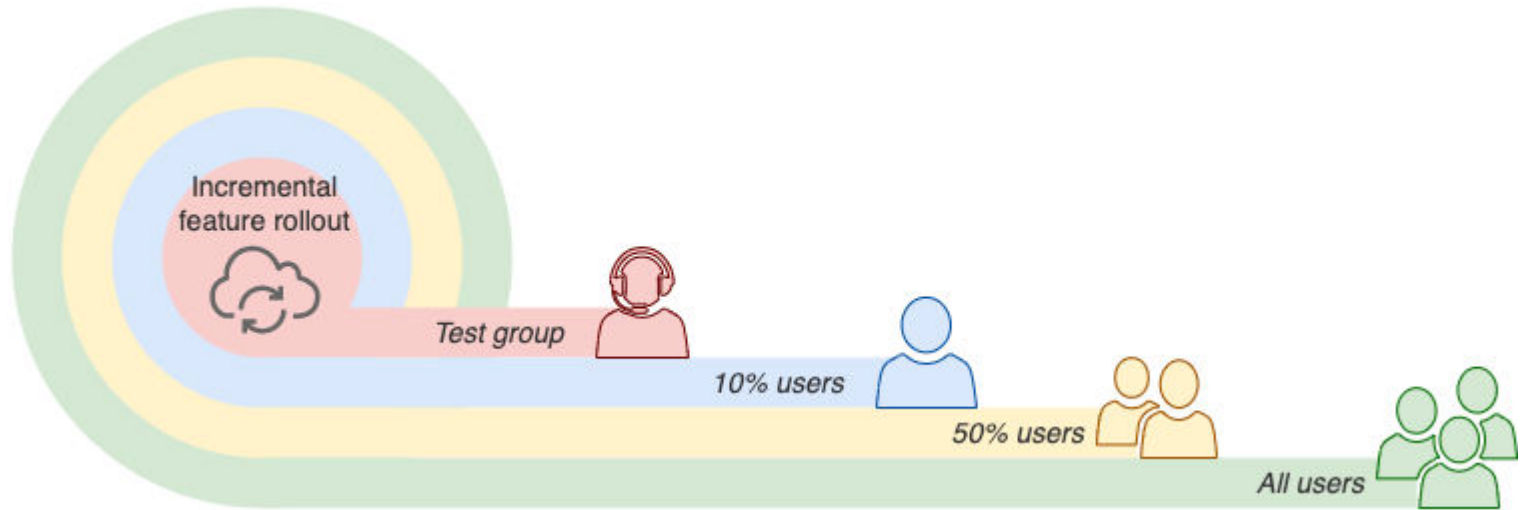
**30,684 potential
hour savings
in a year**



**We rolled it out
to all compatible
projects**



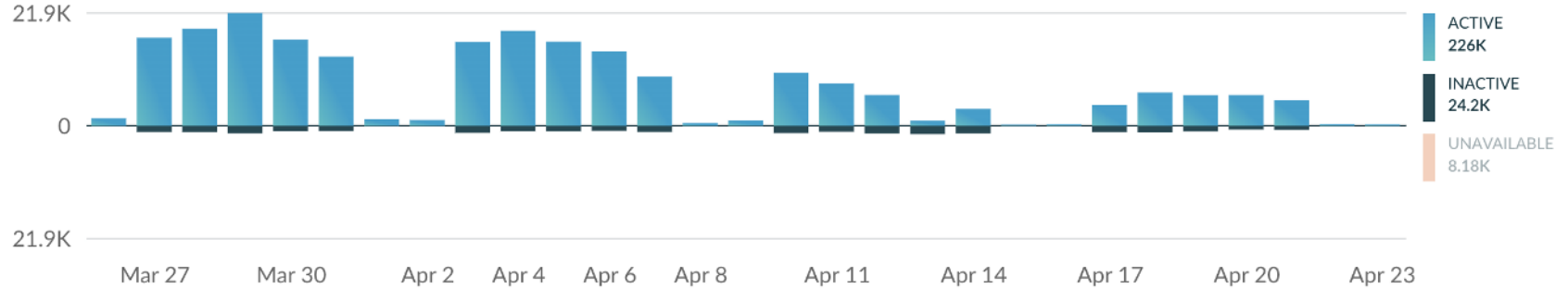
How did we roll out PTS?



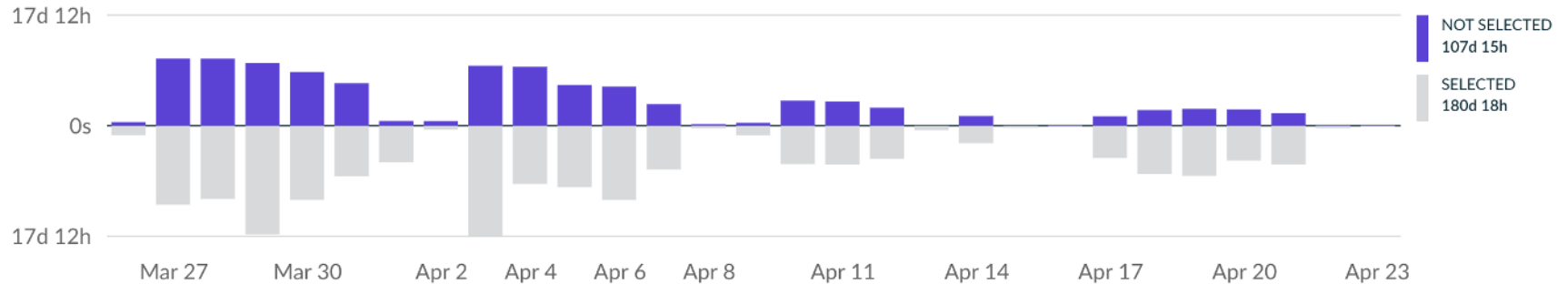
PTS results the first month

Test tasks which enabled Predictive Test Selection [?](#)

226K (87% of total)



107 d 15 hr (88% of 122 d 5 hr total savings potential)



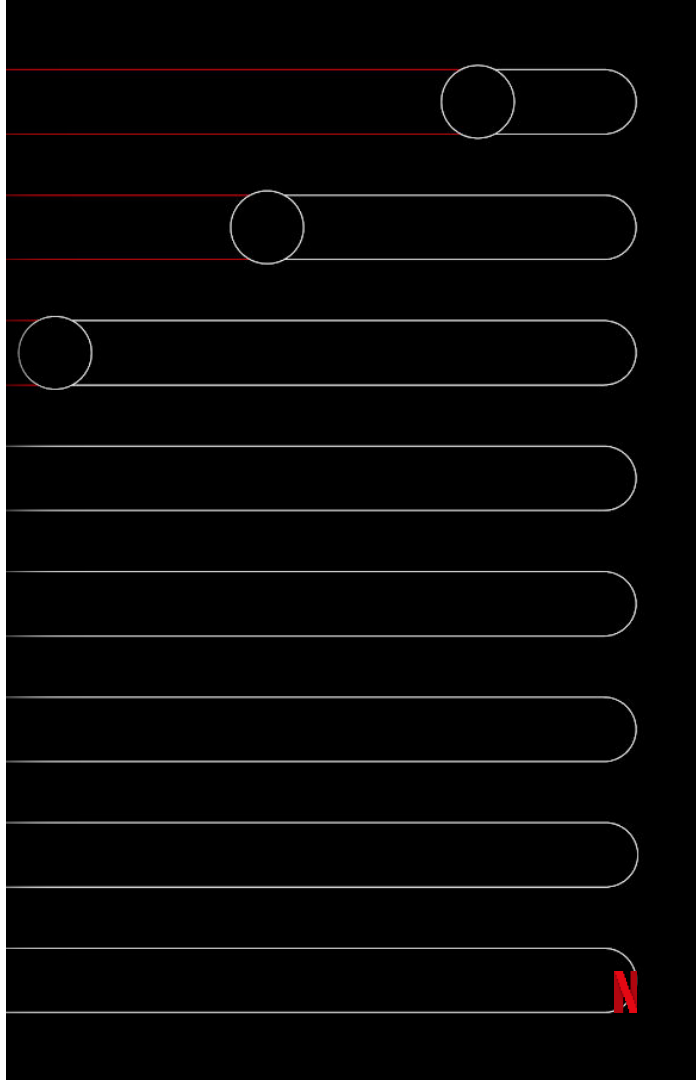
**But not
everything goes
as expected,
unfortunately**

Learnings

- Developers might not like trading testing comprehensiveness for speed
- Missing inputs/outputs in test configurations
- Impact on code coverage tooling



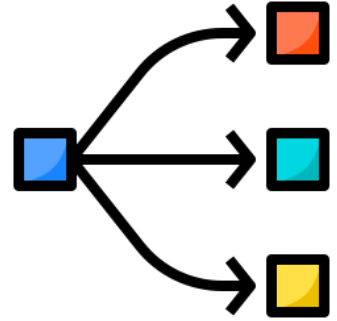
Remote Test Execution



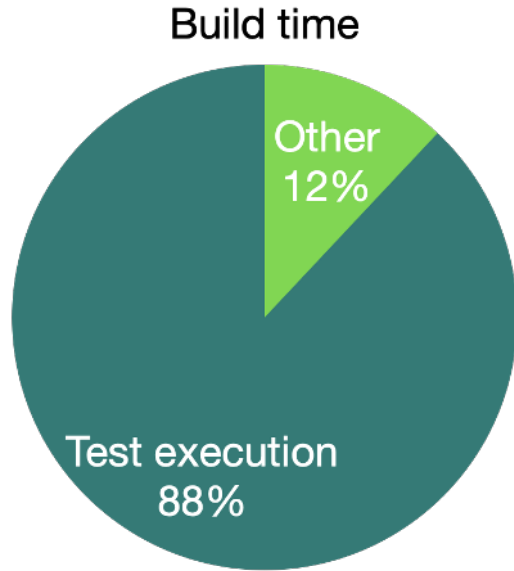
Remote Test Execution

Take existing test suites and distribute them across remote agents to execute them faster

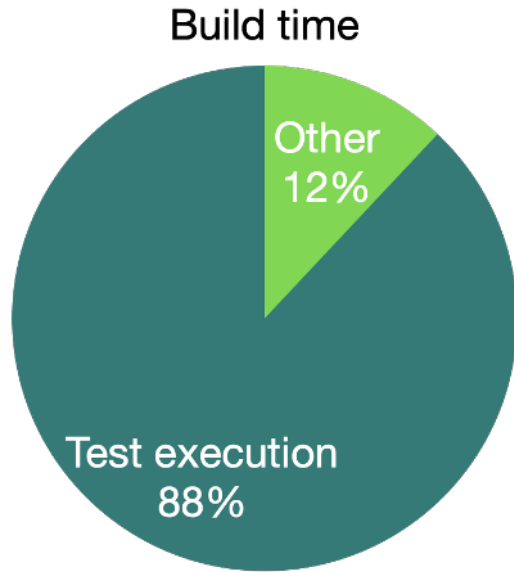
The tests and their supporting files are transferred to each agent and executed, with their logging and results streamed back to the build in real time.



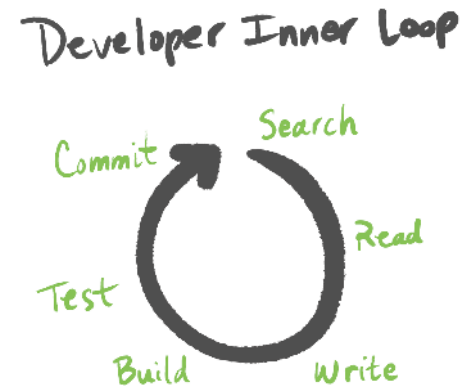
Why Remote Test Execution?



Why Remote Test Execution?



- Consistent experience between local and CI
- Better compute resource usage
- Faster feedback
- Run more tests locally



Life before remote test execution

Gradle Enterprise netflix-gradle-lint integrationTest Nov 30, 2021 4:13:18 PM PST Build Scans

33 tasks executed in 2 projects, **1 failed task** in 1h 2m 15.568s, with 2 avoided tasks saving 12.554s

Execution

:integrationTest

Order: Execution

:processIntegTestResources	54.274s	0.898s	org.gradle.language.jvm.tasks.ProcessResources
:integTestClasses	55.172s	0.000s	org.gradle.api.DefaultTask
:nebulaVersionWriterTask	55.173s	0.024s	com.netflix.nebula.version.NebulaDistributionUrlWriter
:pluginUnderTestMetadata	55.197s	0.050s	org.gradle.plugin.devel.tasks.PluginUnderTestMetadata

:integrationTest FAILED

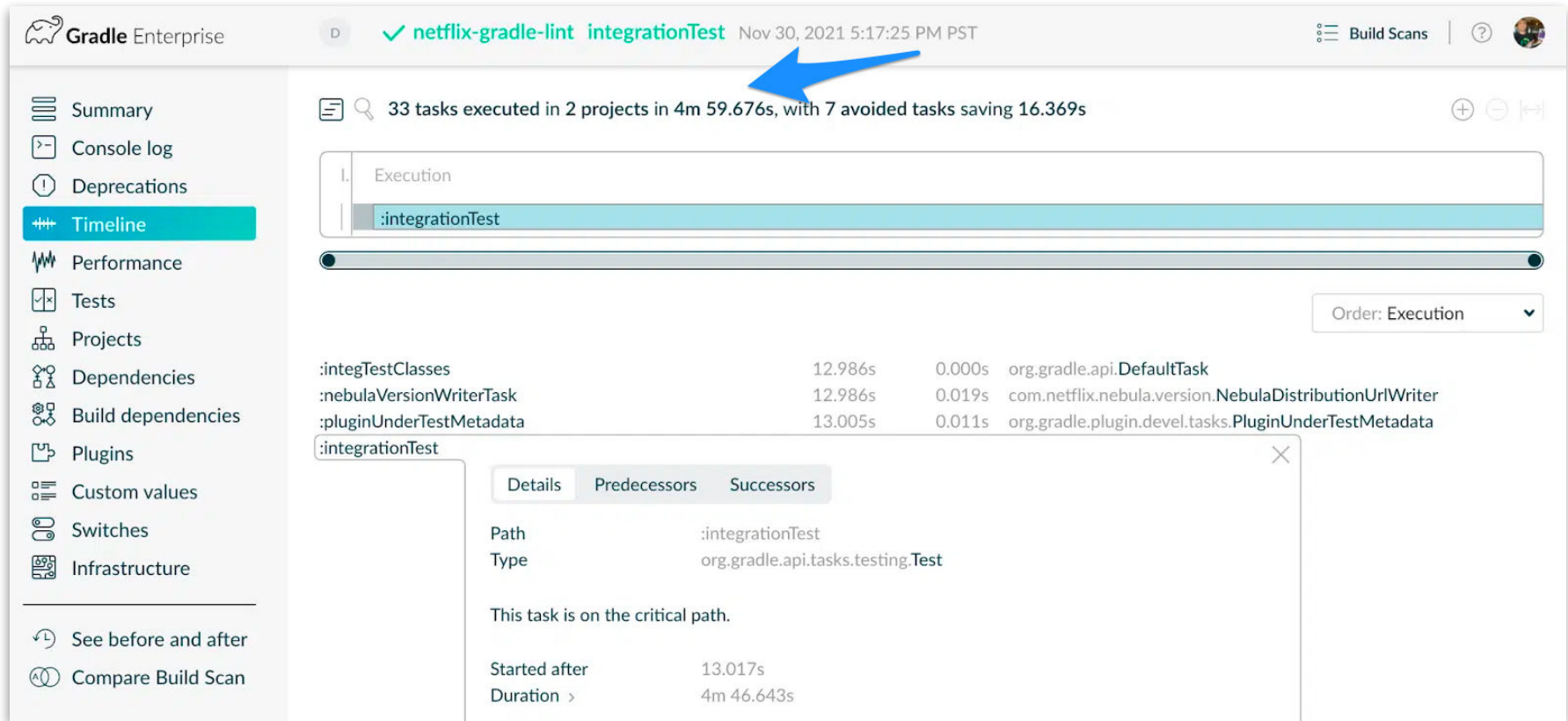
Details Predecessors Successors

Path :integrationTest
Type org.gradle.api.tasks.testing.Test

This task is on the critical path.

Started after 55.248s
Duration > 1h 1m 20.291s

Life after remote test execution



Gradle Enterprise

netflix-gradle-lint integrationTest Nov 30, 2021 5:17:25 PM PST

Build Scans

Summary
Console log
Deprecations
Timeline
Performance
Tests
Projects
Dependencies
Build dependencies
Plugins
Custom values
Switches
Infrastructure

See before and after
Compare Build Scan

33 tasks executed in 2 projects in 4m 59.676s, with 7 avoided tasks saving 16.369s

Execution

:integrationTest

Order: Execution

:integTestClasses	12.986s	0.000s	org.gradle.api.DefaultTask
:nebulaVersionWriterTask	12.986s	0.019s	com.netflix.nebula.version.NebulaDistributionUrlWriter
:pluginUnderTestMetadata	13.005s	0.011s	org.gradle.plugin.devel.tasks.PluginUnderTestMetadata

:integrationTest

Details Predecessors Successors

Path :integrationTest

Type org.gradle.api.tasks.testing.Test

This task is on the critical path.

Started after 13.017s

Duration > 4m 46.643s

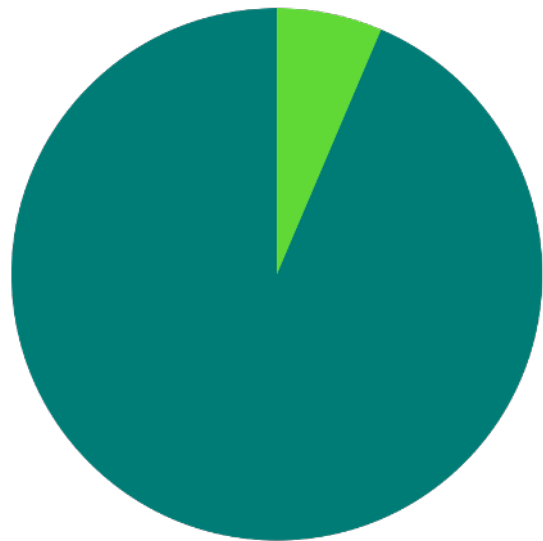
Be aware of potential limitations

- Network or security access
- Different environment debugging
- Network traffic and slow connections

Current use as a Beta offering

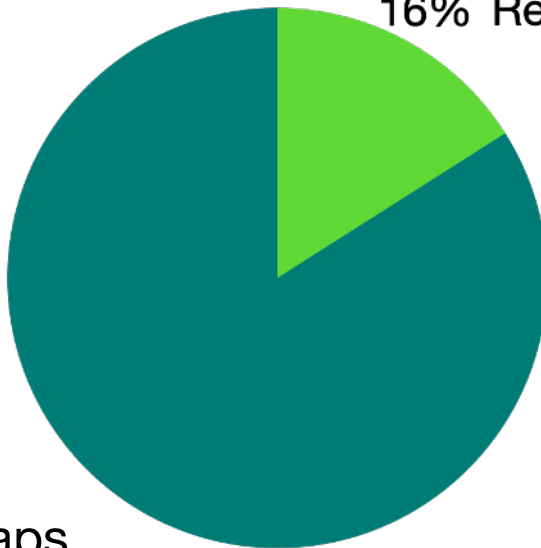
6.4% Remote execution (builds)

**14.8k of 233k
builds**



16% Remote execution (repositories)

**544 of 3.4k
repositories**



| We are working to close our gaps



Learnings along the way

Sure, there are 😊😊!

Technological solutions

We improved other experiences, too

- Enabled CI jobs parallel executions
- Increase on testing against real datastores and AWS cloud resources
- Reduction on CI agents failures due to misconfigured Test Suites
- Reduce CI compute resource usage



Social solutions

Abstracting tooling where appropriate

- All the solutions discussed can be applied on a small scope of project-by-project to a large scope of all projects at once
- Use good judgement to determine if the solution should be targeted to a select set of repositories versus applied for all



Make it simple

- Be clear on what is changing, when and what to expect.
- Over-communicate the ways folks can find useful information.
- Prevent migration fatigue. Enable low-effort opt-in or opt-out.
- Judiciously decide when to require user changes.



Understandability, documentation, and tracking

- Provide actionable error messages that point to further documentation where needed
- Provide actionable Pull Requests with informative messages that are tracked in a campaign
- Spread awareness of the new features that save them time and energy! Newsletters and townhall feature reviews are great for this



Beta test this features with your team and/or close partners

- Find customers who would benefit the most from this to work with and help lead this effort
- Sharing with close partners helps uncover scenarios that you probably didn't think of
- Communicate expected outcomes and risks
- Measure before and after a feature has been introduced



Request feedback, feedback is key





ARE YOU WITH ME?

NETFLIX

I

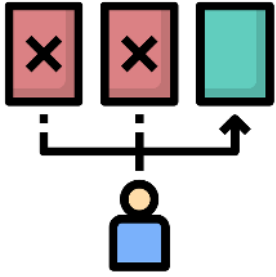
N

Last thoughts...

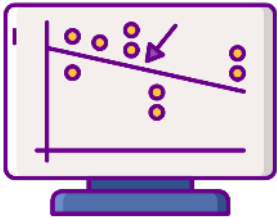
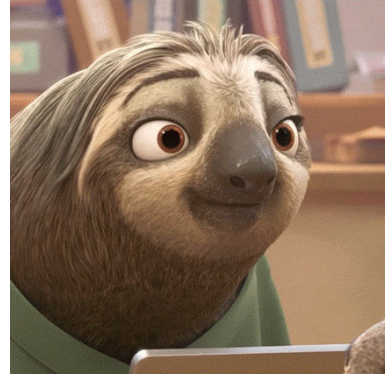
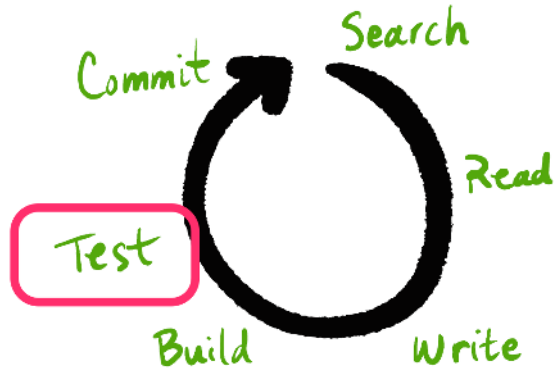
- Scale doesn't matter
- Treat the testing process well!
- Invest on testing experience
- Enable fast testing cycles
- Treat test suites like production code



How?



Developer Inner Loop



Questions?



Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com



Thank you!



Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com

