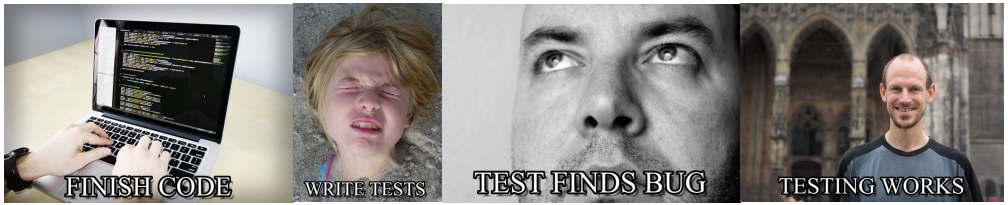
The State of Continuous Integration Testing @Google

By: John Micco - jmicco@google.com
投稿者：ジョン・ミッコ

Testing Scale at Google

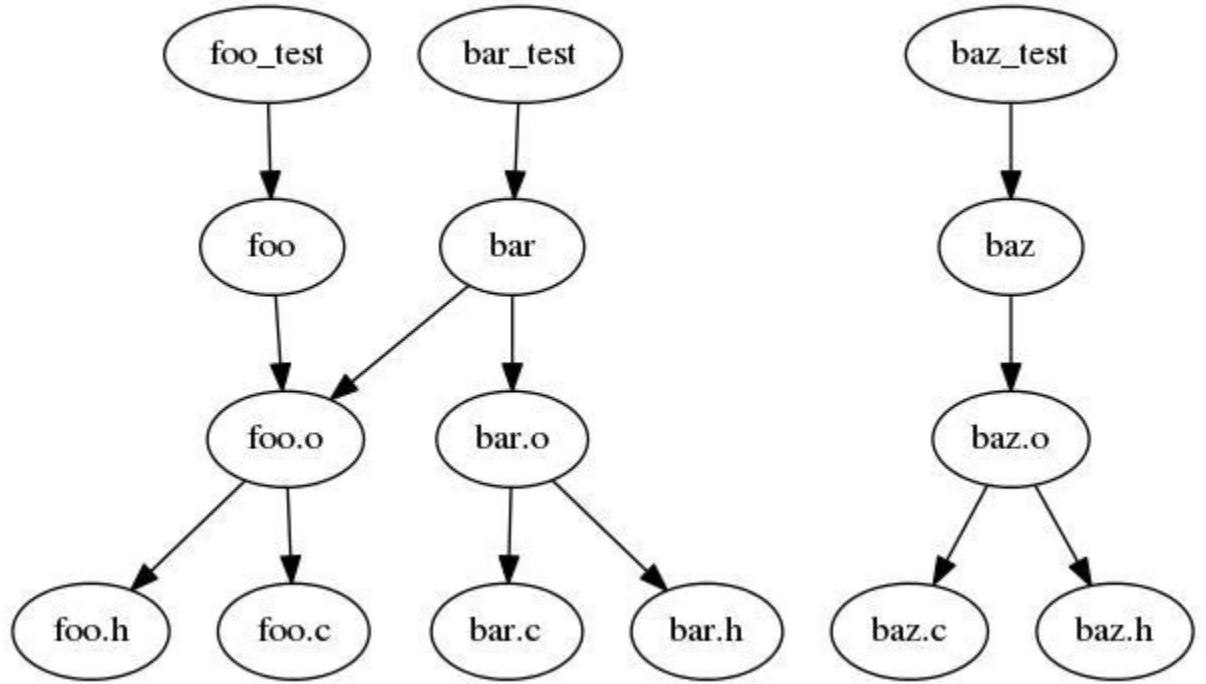
- 4.2 million individual tests running continuously
  - Testing runs before and after code submission
- 150 million test executions / day (averaging 35 runs / test / day)
- Distributed using internal version of bazel.io to a large compute farm
- Almost all testing is automated - no time for Quality Assurance
- 13,000+ individual project teams - all submitting to one branch
- Drives continuous delivery for Google
- 99% of all test executions pass



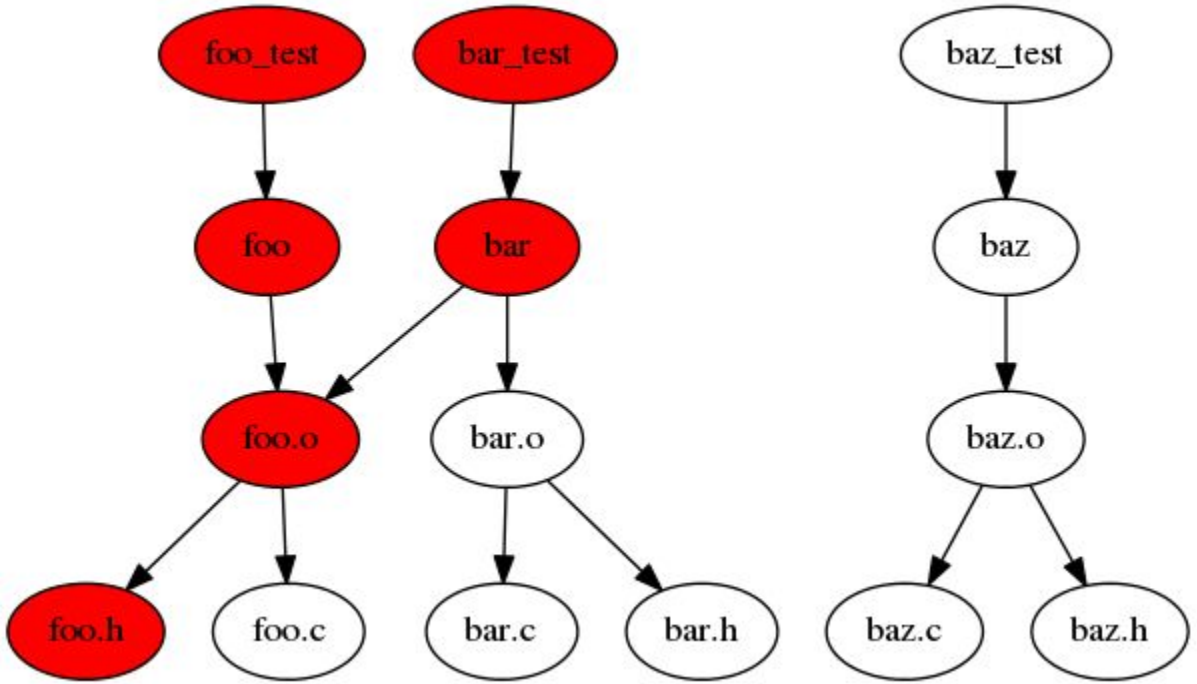
Testing Culture @ Google

- ~10 Years of testing culture promoting hand-curated automated testing
  - Testing on the toilet and Google testing blog started in 2007
  - GTAC conference since 2006 to share best practices across the industry
  - Part of our new hire orientation program
- SETI role
  - Usually 1-2 SETI engineers / 8-10 person team
  - Develop test infrastructure to enable testing
- Engineers are expected to write automated tests for their submissions
- Limited experimentation with model-based / automated testing
  - Fuzzing, UI waltkthroughs, Mutation testing, etc.
  - Not a large fraction of overall testing

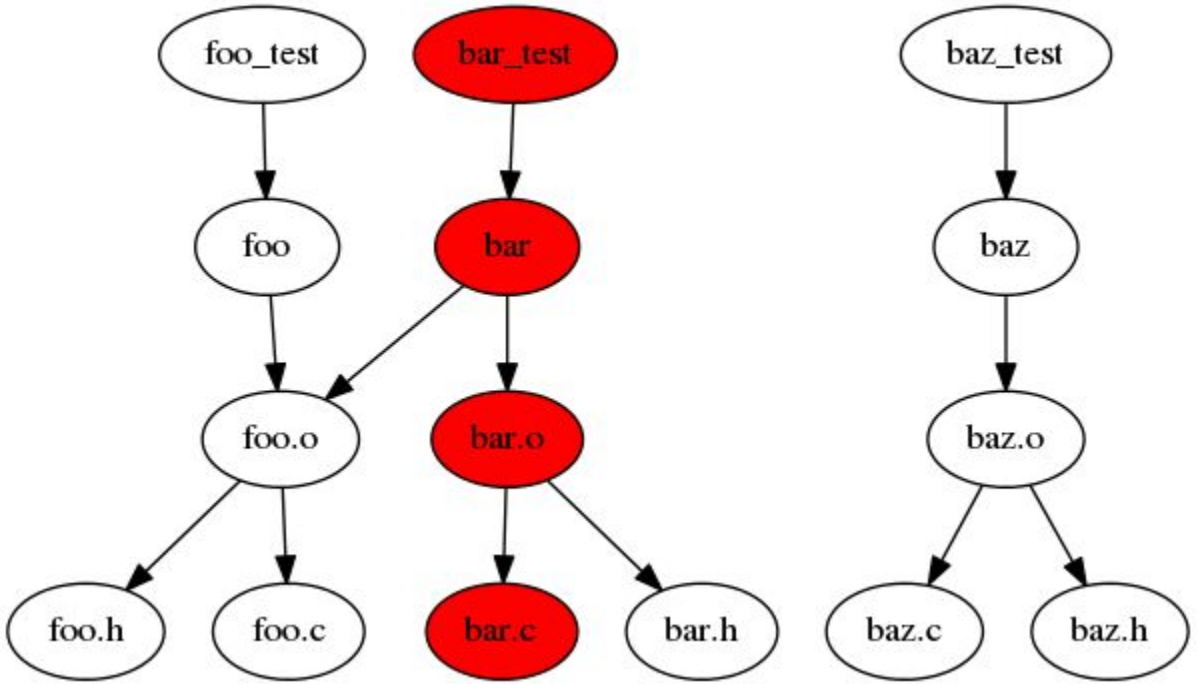
Regression Test Selection (RTS)



Regression Test Selection (RTS)



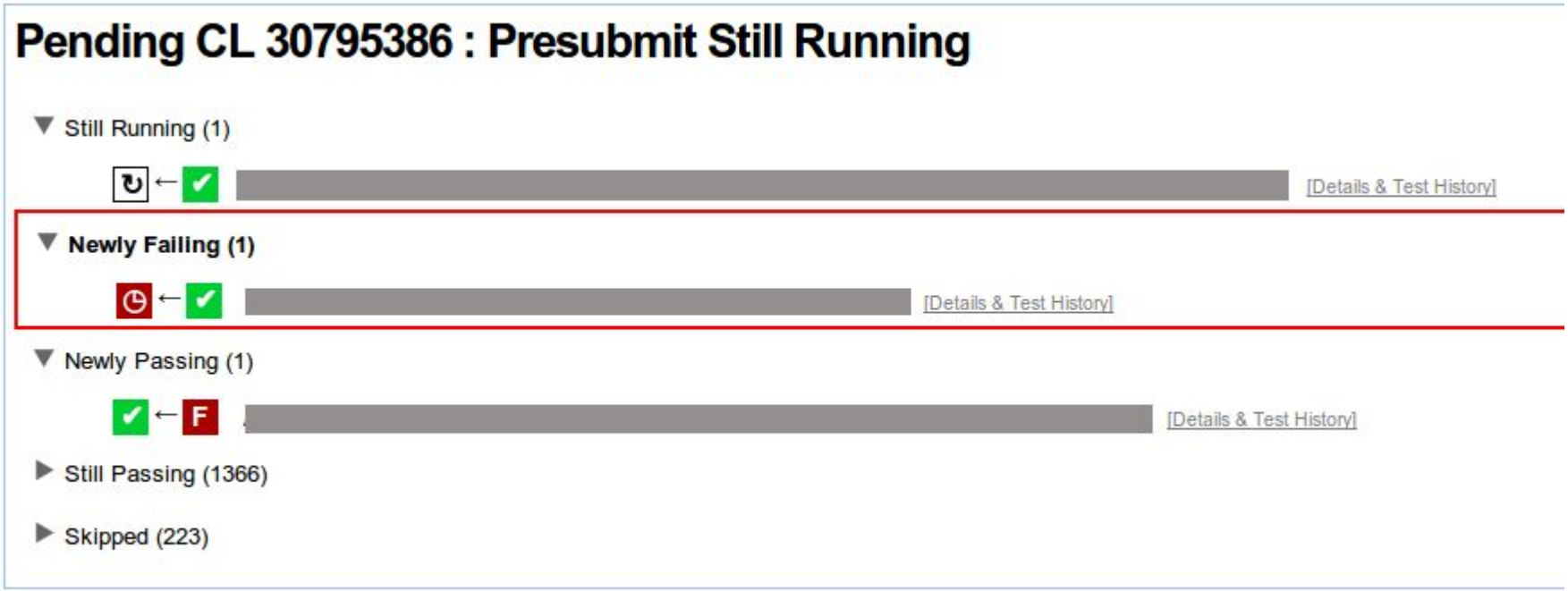
Regression Test Selection (RTS)



Presubmit Testing

- Uses fine-grained dependencies
- Uses same pool of compute resources
- Avoids breaking the build
- Captures contents of a change and tests in isolation
  - Tests against HEAD
- Integrates with
  - submission tool - submit iff testing is green
  - Code Review Tool - results are posted to the review

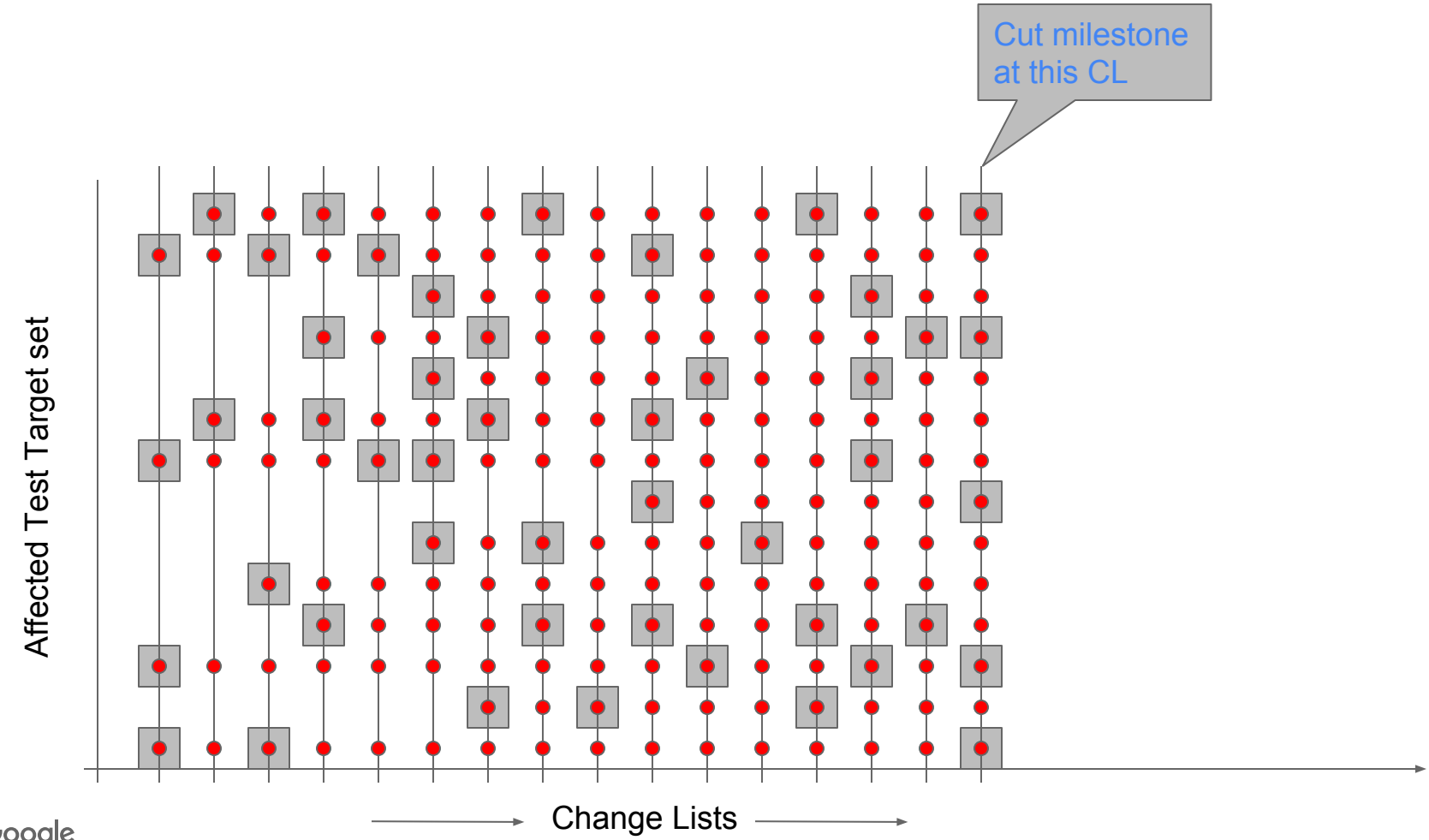
Example Presubmit Display

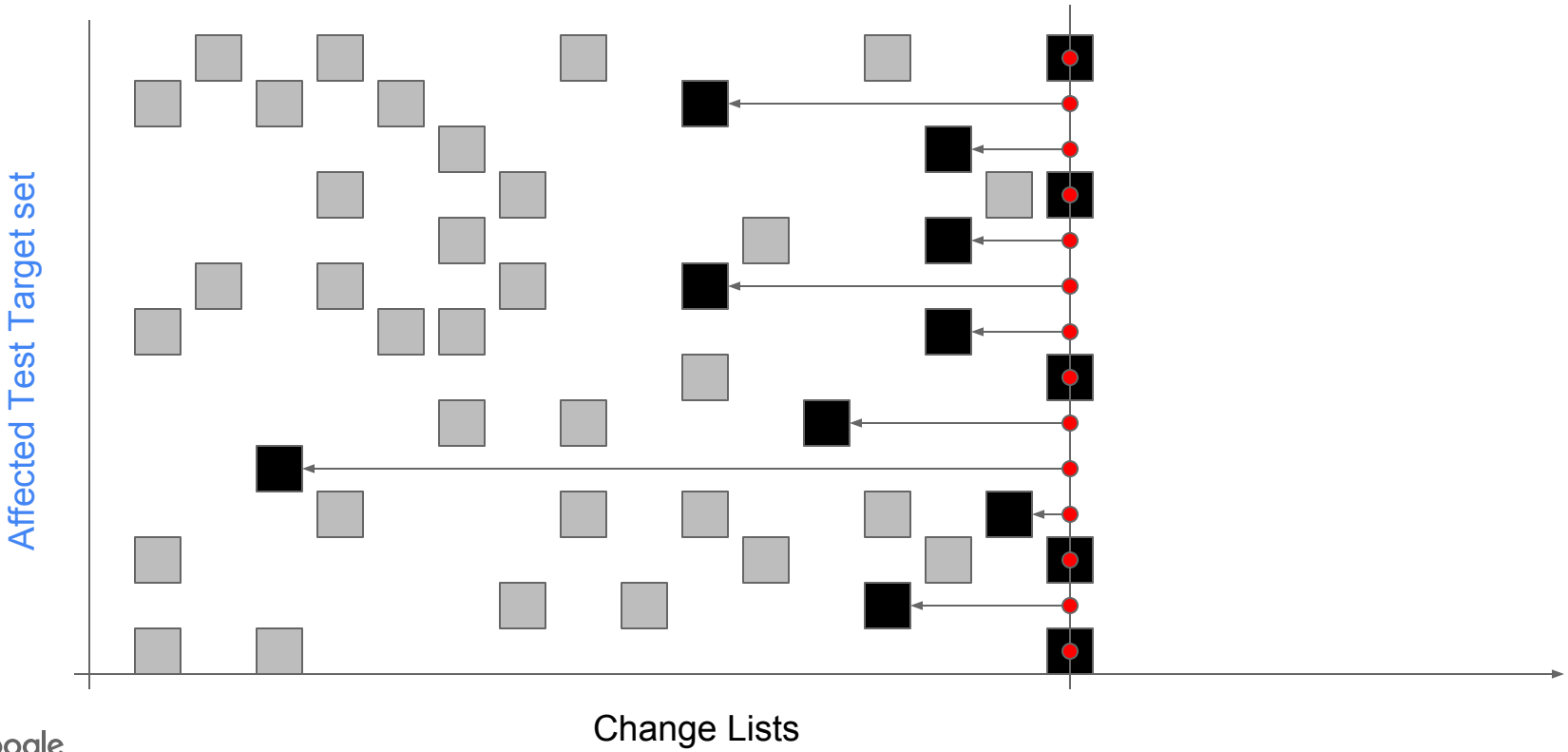


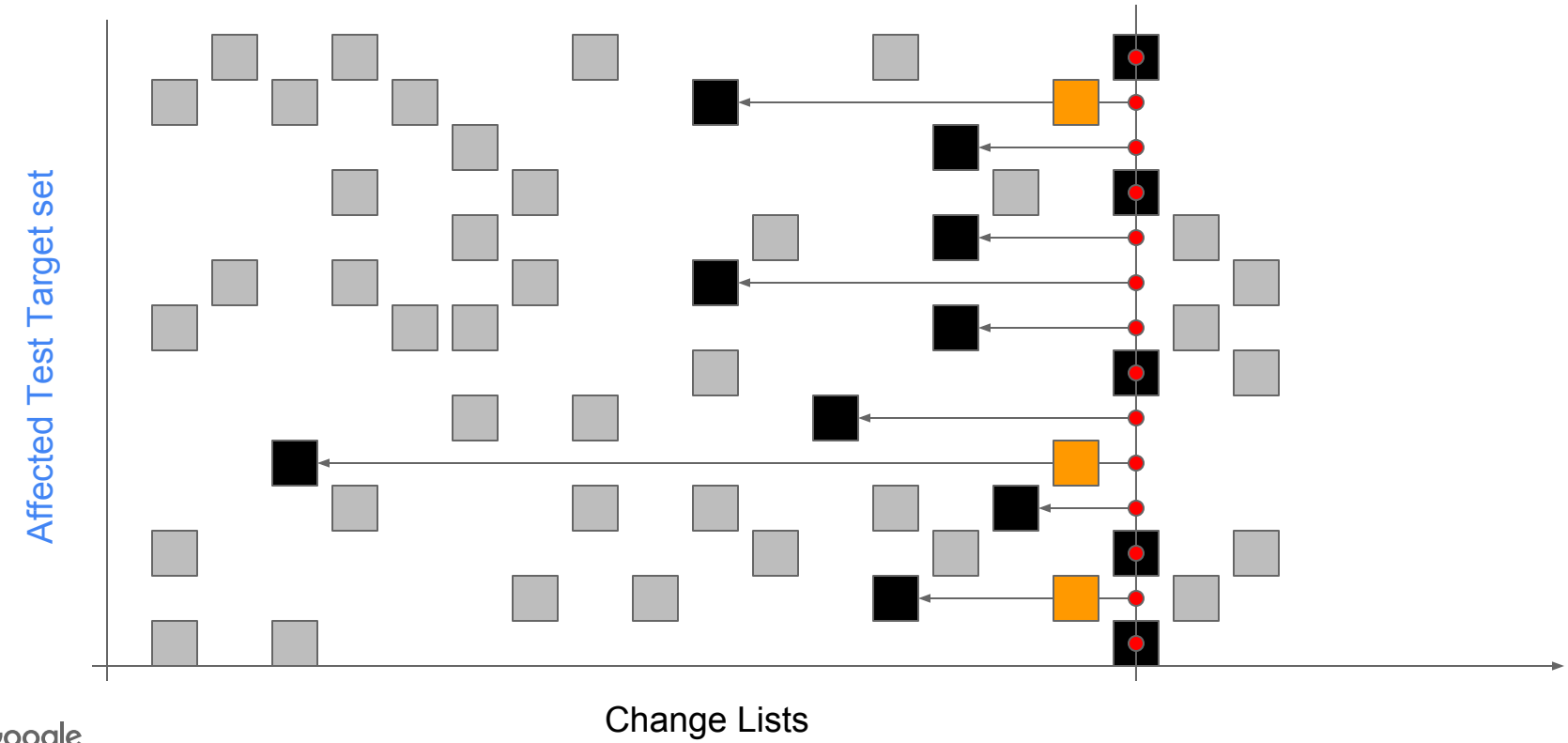
Postsubmit testing

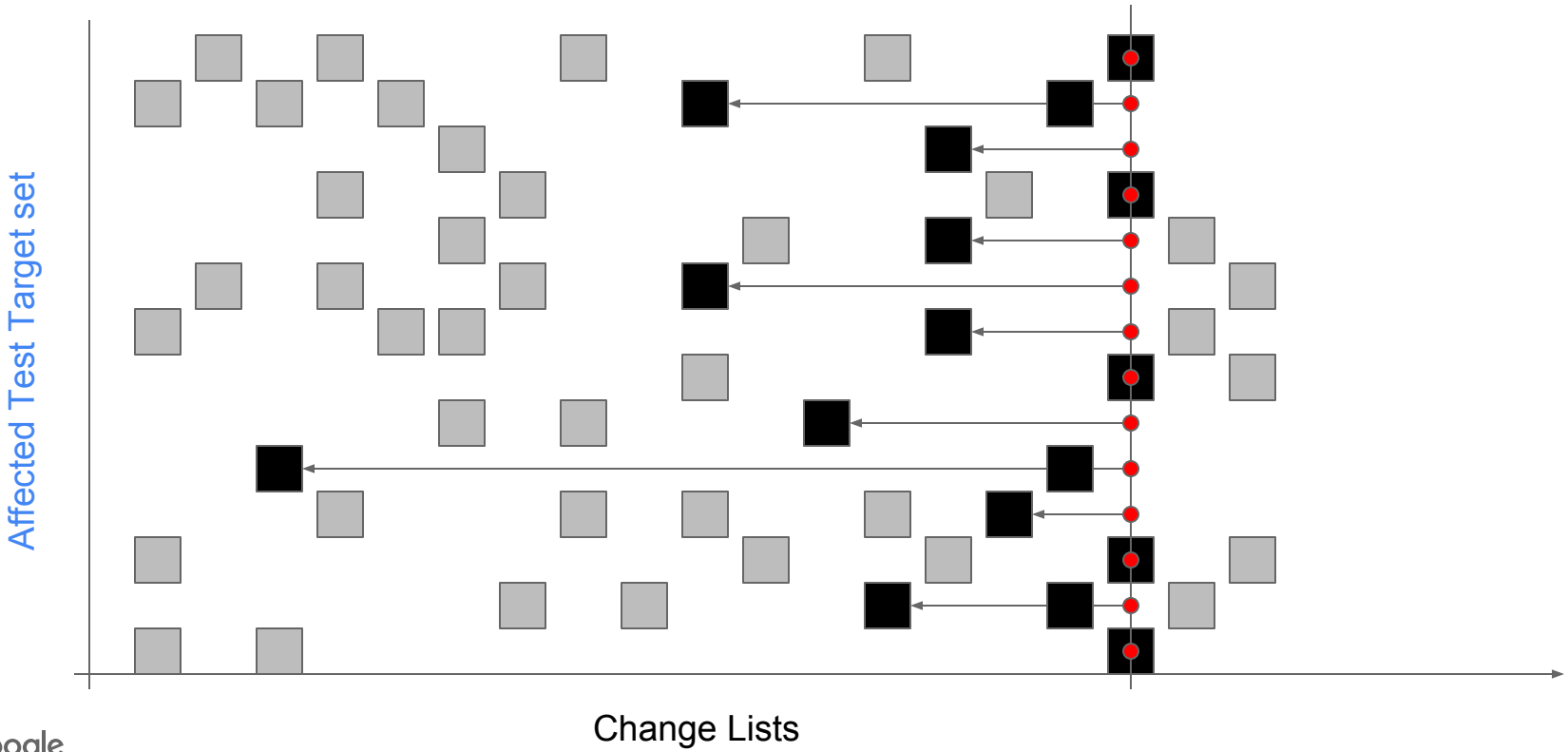
- Continuously runs 4.2M tests as changes are submitted
  - A test is affected iff a file being changed is present in the transitive closure of the test dependencies. (Regression Test Selection)
  - Each test runs in 2 distinct flag combinations (on average)
  - Build and run tests concurrently on distributed backend.
  - Runs as often as capacity allows
- Records the pass / fail result for each test in a database
  - Each run is uniquely identified by the test + flags + change
  - We have 2 years of results for all tests
  - And accurate information about what was changed

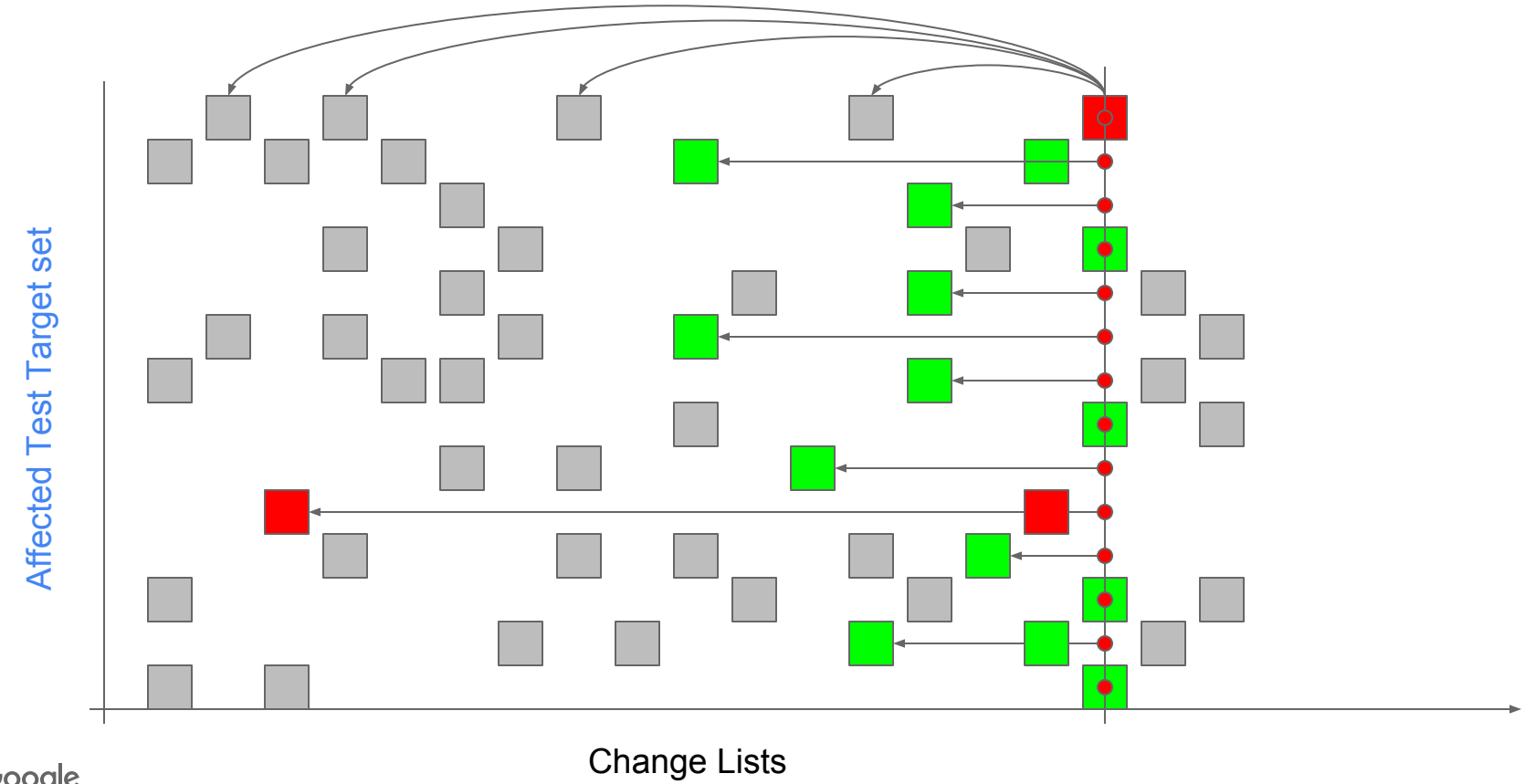
See: prior deck about Google CI System, See this paper about piper and CLs


Cut milestone
at this CL
Affected Test Target set
Change Lists

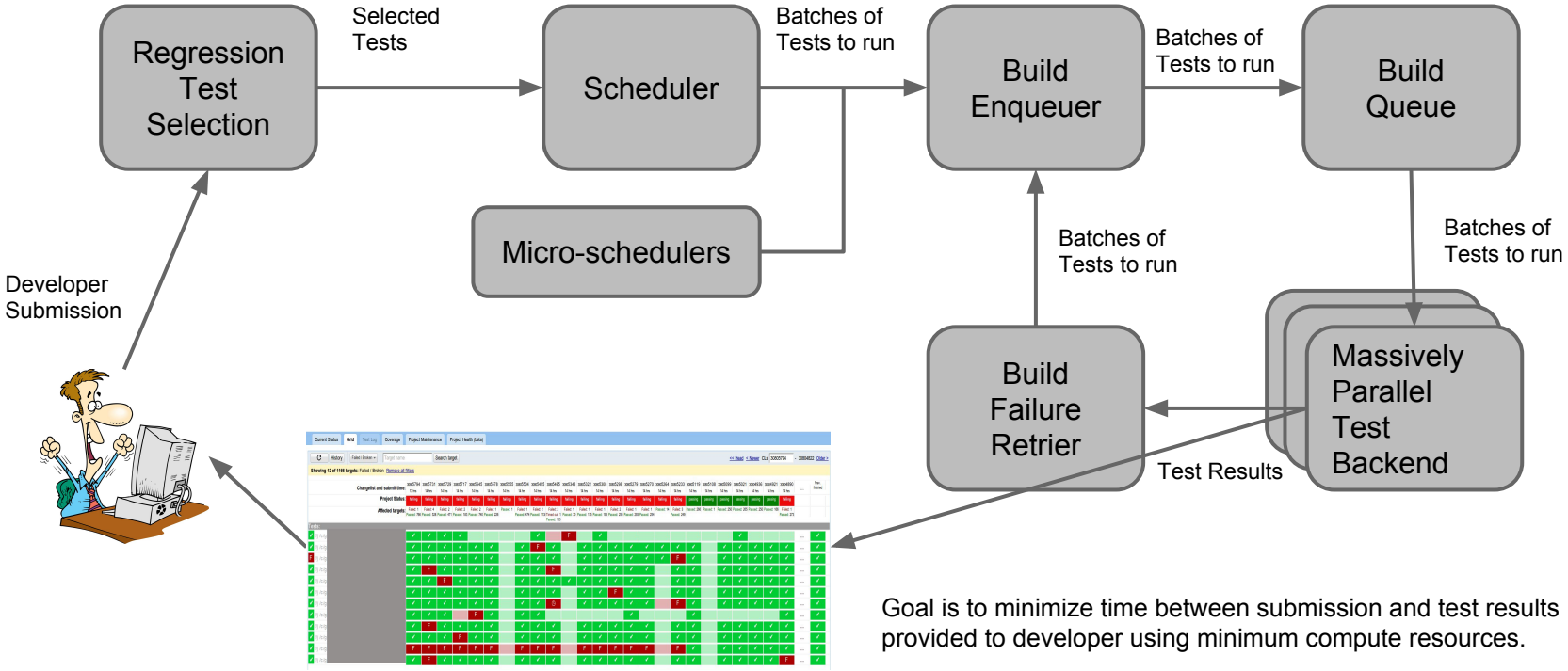

Affected Test Target set
Change Lists


Affected Test Target set
Change Lists


Affected Test Target set
Change Lists


Affected Test Target set
Change Lists

Life of a Test Execution

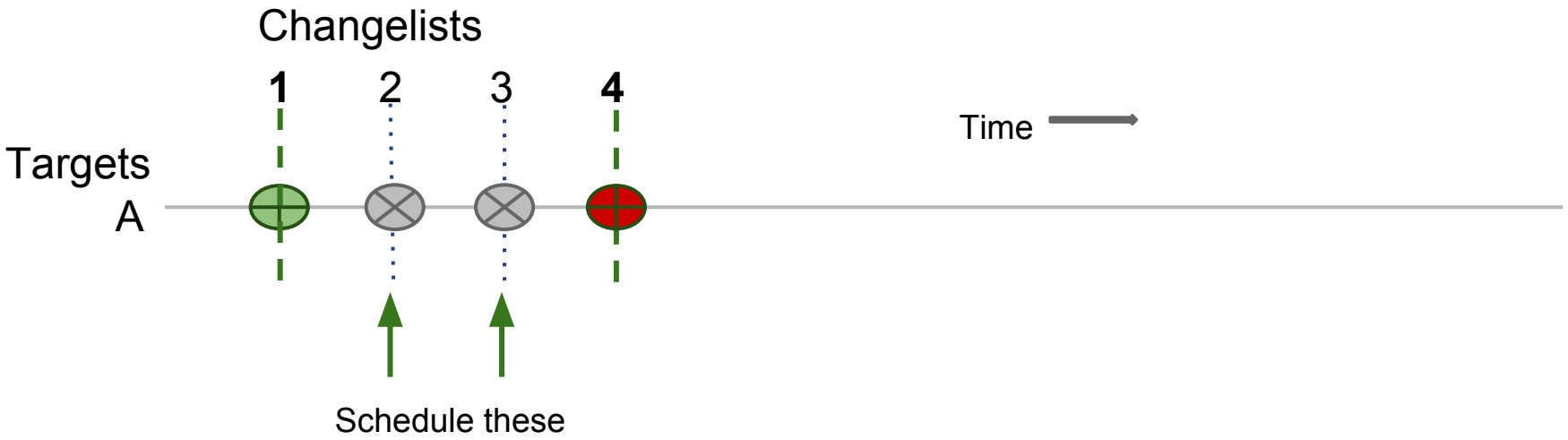


Goal is to minimize time between submission and test results provided to developer using minimum compute resources.

Micro-schedulers

- Selectively run any target at any CL
- Fill the gaps in the main scheduler
  - Missed targets
  - Not-yet-run targets
- Research hypotheses can be quickly tested

Cuprit Finding - Transition to Fail



Cuprit Finding - Transition to Fail

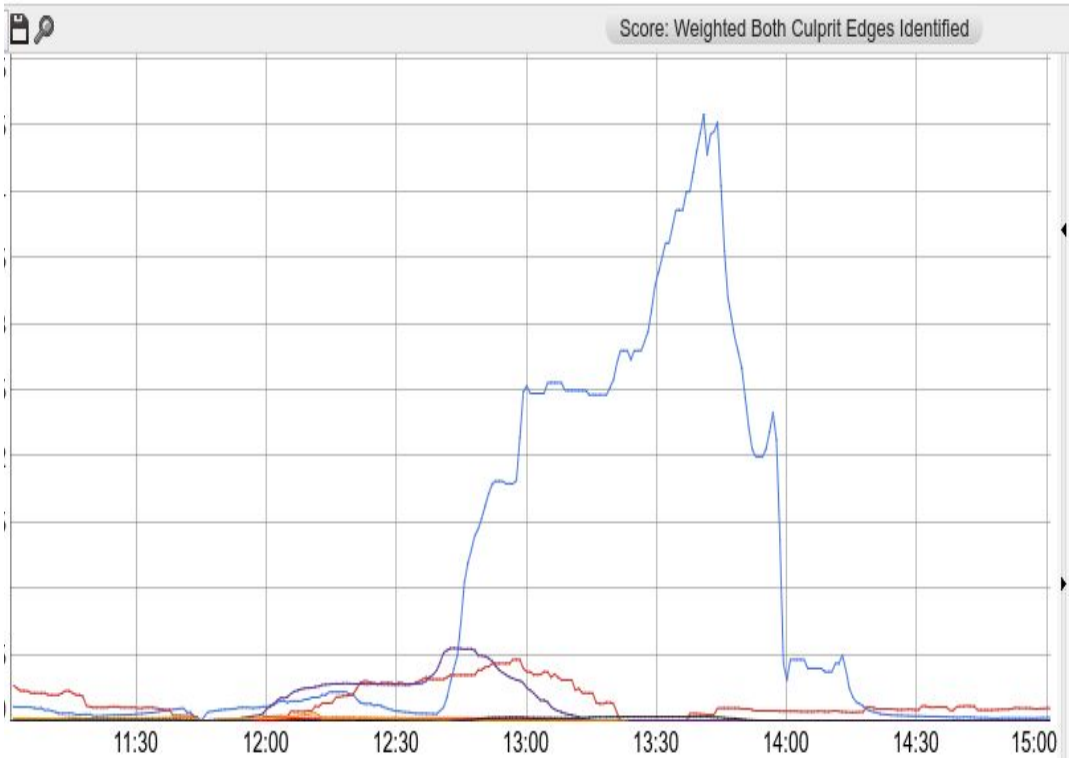
A: Change 3 broke test A.

Passed
Affected, but not run (yet)
Failed

Milestone
Non-milestone

Other micro-schedulers

- Culprit finder
  - Ranked culprit finder
  - Flakiness culprit finder
- Breakage predictor
  - Hot spots seeker
  - Brain-based predictor
  - Crowd sourcer
- Fix detector
- Auto-rollback



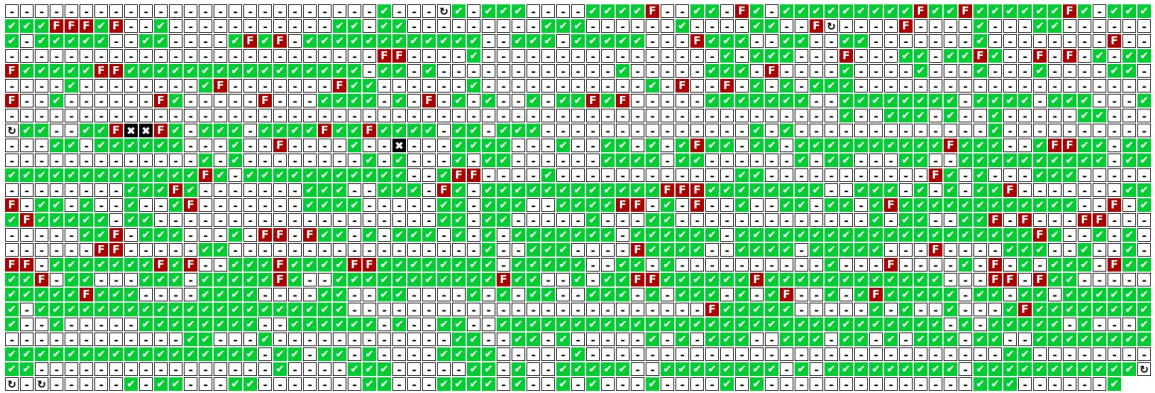
Analysis of Test Results at Google

- Analysis of a large sample of tests (1 month) showed:
  - 84% of transitions from Pass -> Fail are from "flaky" tests
  - Only 1.23% of tests ever found a breakage
  - Frequently changed files more likely to cause a breakage
  - 3 or more developers changing a file is more likely to cause a breakage
  - Changes "closer" in the dependency graph more likely to cause a breakage
  - Certain people / automation more likely to cause breakages (oops!)
  - Certain languages more likely to cause breakages (sorry)

See: prior deck about Google CI System, See this paper about piper and CLs

Flaky Tests

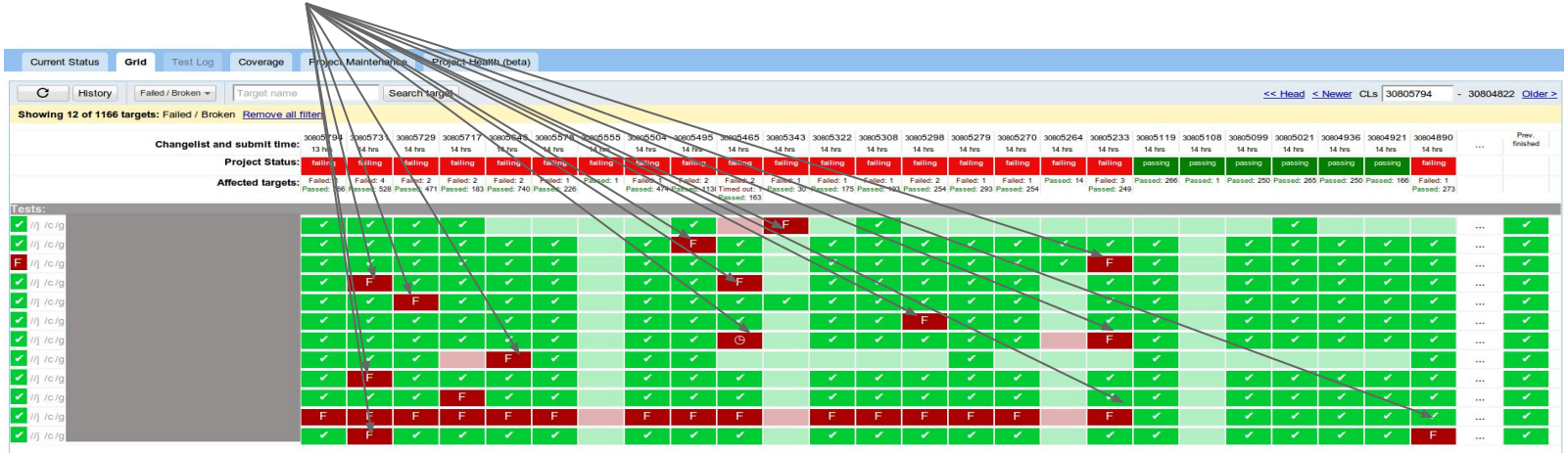
- Test Flakiness is a huge problem
- Flakiness is a test that is observed to both Pass and Fail with the same code
- Almost 16% of our 4.2M tests have some level of flakiness
- Flaky failures frequently block and delay releases
- Developers ignore flaky tests when submitting - sometimes incorrectly
- We spend between 2 and 16% of our compute resources re-running flaky tests



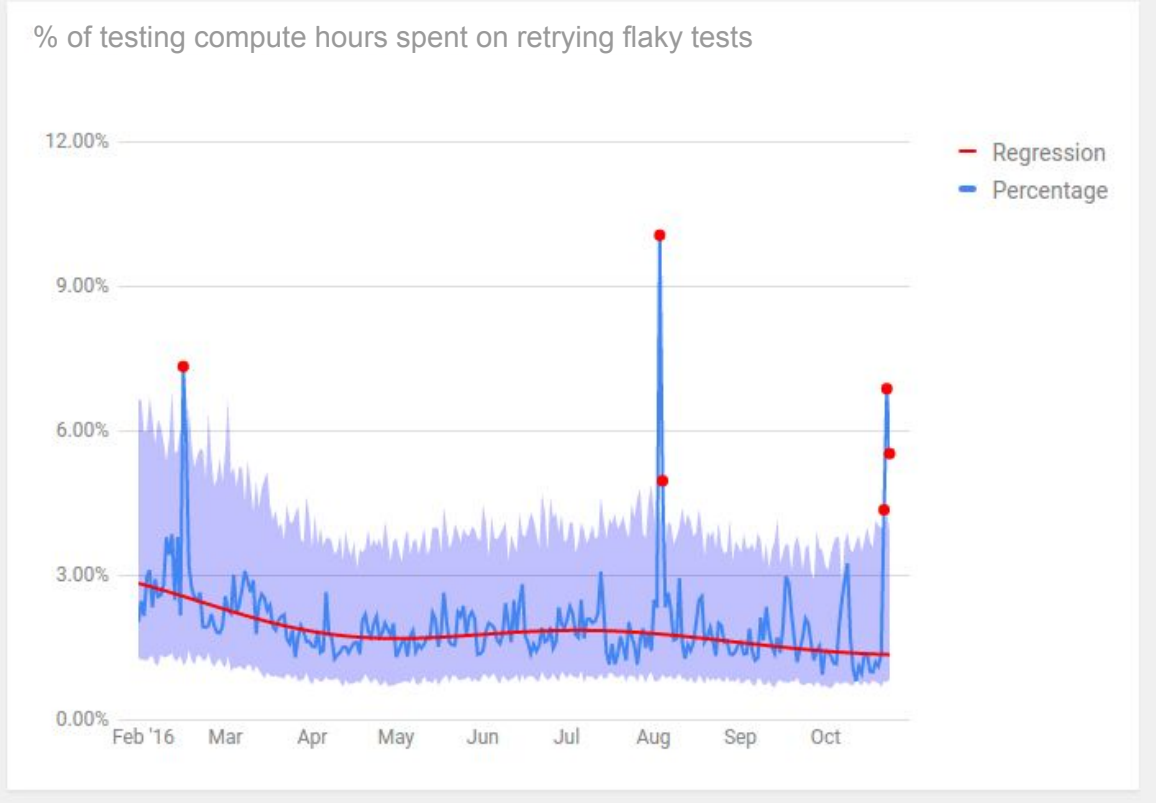
Flaky test impact on project health

- Many tests need to be aggregated to qualify a project
- Probability of flake aggregates as well
- Flakes
  - Consume developer time investigating
  - Delay project releases
  - Waste compute resources re-running to confirm

Flakes

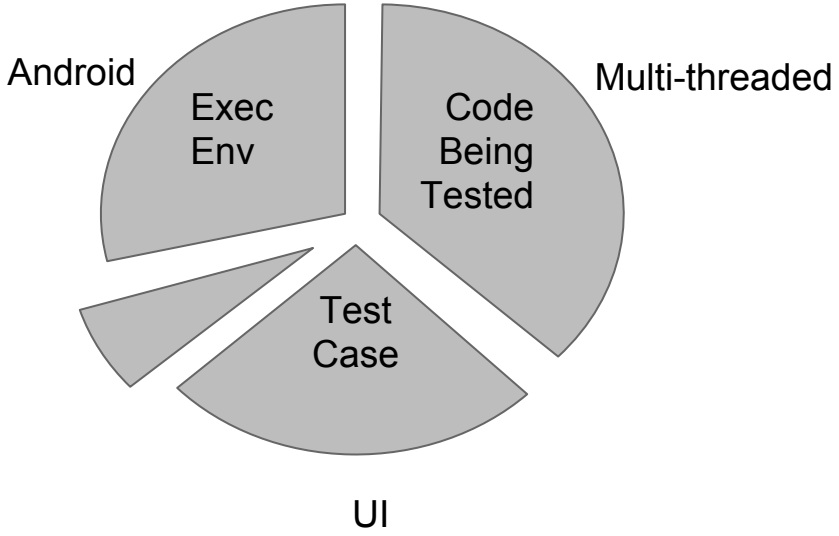


Percentage of resources spent re-running flakes



Sources of Flakiness

- Factors that cause flakes
  - Test case factors
    - Waits for resource
    - sleep()
    - Webdriver test
    - UI test
  - Code being tested
    - Multi-threaded
  - Execution environment/flags
    - Chrome
    - Android
  - ...



 See: https://pdfs.semanticscholar.org/02da/46889ee3c6bc44bfa0fc45071195781b99ce.pdf

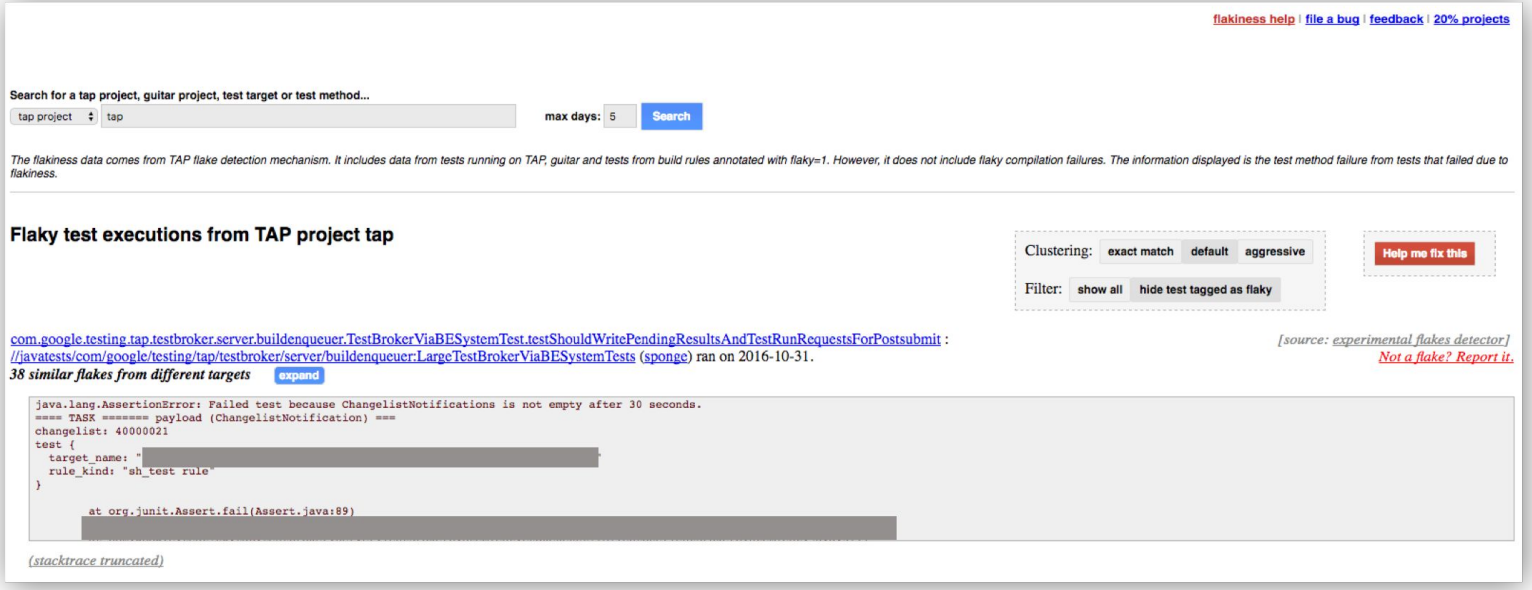
Flakes are Inevitable

- Continual rate of 1.5% of test executions reporting a "flaky" result
- Despite large effort to identify and remove flakiness
  - Targeted "fixits"
  - Continual pressure on flakes
- Observed insertion rate is about the same as fix rate

Conclusion: Testing systems must be able to deal with a certain level of flakiness. Preferably minimizing the cost to developers

Flaky Test Infrastructure

- We re-run test failure transitions (10x) to verify flakiness
  - If we observe a pass the test was flaky
  - Keep a database and web UI for "known" flaky tests



Google's Internal Development Systems

- Much of what Google uses internally is proprietary
- We have started open sourcing our tools starting with Bazel (bazel.io)
- Bazel is the same build tool that we use internally (with the Google proprietary parts removed)

An example bazel BUILD file

java/BUILD:

```
java_library(
    name = "mylib",
    srcs = ["my/webapp/TestServlet.java"],
    deps = [":javax.servlet.api"],
)

appengine_war(
    name = "myapp",
    jars = ["mylib"],
    resources = ["//dart:dart"],
    ...
)
```

dart/BUILD:

```
dart_library(
    name = "mylib",
    srcs = glob(["mylib/**/*.dart"]),
)
dart_library(
    name = "dart",
    deps = ["mylib"],
)
dart_test(
    name = "mydart_test",
    deps = ["dart", "mylib"]
    srcs = global(["mytests/**/*.dart"]),
)
```

rule's name

highly accurate dependencies

Tests appear with accurate dependencies

Enabling Google-Scale Research in Academia

- Most academic work tests hypotheses in open source projects
  - Limited codebase
  - No historical Pass / Fail results
  - Old projects with low churn rate / relevance

- What we are doing about it
  - Sponsor researchers to come in - student interns and visiting faculty
  - Test hypotheses against Google code base at scale
  - Full access to historic Pass / Fail data helps to test hypotheses
  - Publish results and relevant data sets
  - Creating API frameworks and extensibility (like micro-schedulers) to ease experimentation

Academic Research in Software Testing @ Google

- Join us for an internship or the Visiting Faculty Program!
  - Test hypotheses against real data at scale
  - Publish relevant papers
    - With sanitized data sets!
  - Test ideas more quickly
  - Make data from Google scale application development more widely available
- Participate in our journal club
  - Review relevant papers monthly
  - Paper authors often join the discussion
- Apply for a Google Faculty Research Award

Q&A

For more information:

- Google Testing Blog on CI system
- Youtube Video of Previous Talk on CI at Google

- Flaky Tests and How We Mitigate Them

- Why Google Stores Billions of Lines of Code in a Single Repo
- GTAC 2016 Flaky Tests Presentation
- (ICSE 2017) "Who Broke the Build? Automatically Identifying Changes That Induce Test Failures In Continuous Integration at Google Scale" by Celal Ziftci and Jim Reardon
- (ICSE 2017) “Taming Google-Scale Continuous Testing,” by Atif Memon, Zebao Gao, Bao Nguyen, Sanjeev Dhanda, Eric Nickell, Rob Siemborski and John Micco