



Intelligent Development at Google

By: John Micco

Google Inc. (*jmicco@google.com*)

Google developer scale

30,000+

developers

1 billion

files¹

800,000

builds per day

45,000

commits per workday²

9 million

source files

150 million

test cases run per day

20,000

code reviews per workday

2 billion

lines of code

2+ PB

of build outputs per day

¹ Including release branches

² Automated and Interactive


A day in the life of a Google developer

- 1 Write a patch against a component with many dependencies.
- 2 Test against *the entire Google codebase*. Pass!
- 3 Send for review. LGTM!



Googlers want an amazing dev stack

- A comprehensive set of well integrated tools
- Access to high-quality libraries
- Zero DevOps overhead



Developer's Journey

Understanding code

The screenshot shows a code editor with a file named `file.h`. The code defines a function `Open` in the `::util::` namespace. The function signature is `Status Open(StringPiece filename, StringPiece mode, ::File** f, const file::Options& options);`. The comments indicate it opens a file in a specified mode, typically `open(3)`, and returns a `File` object on success. The sidebar on the right shows the following information:

- XRefs** History Warnings Docs Project
- 24692 references to/from `Open`** provided by the [Kythe](#) project. [See a problem?](#)
- Definitions** (2 occurrences, 1 displayed, 1 not loaded)
 - `323 util::Status Open(StringPiece filename, StringPiece mode,`
- Declarations** (2 occurrences, 1 displayed, 1 not loaded)
 - `73 ::util::Status Open(StringPiece filename, StringPiece mode, ::File** f,`
- Call Hierarchy** (517 occurrences, 11956 not loaded)
 - `156 TestBody() {... EXPECT_OK(file::Open(file_name, ...)}`

Changing code (see Tricorder [paper](#))

The screenshot shows an IDE interface with a file explorer on the left, a code editor in the center, and a bottom panel for errors and changes.

File Explorer (Left):

- 'clean' synced @52508564
- Bookmarked Directories
- Recent Directories
 - build.sh
 - di deps.dot
 - genfiles
 - internal
 - Absent.java
 - Absent_CustomFieldSerial
 - AbstractIterator.java
 - Ascii.java
 - BUILD
 - Base.gwt.xml.part
 - BinaryPredicate.java
 - BinaryPredicates.java
 - CaseFormat.java
 - CharMatcher.java
 - CharsetCache.java
 - Charsets.java
 - Converter.java
- Entire Google3

Code Editor (Center):

File: Pair.java

```
213 * <p>This implementation returns a string in the form
214 * {code (first, second)}, where {code first} and {code second} are the
215 * String representations of the first and second elements of this pair, as
216 * given by {@link String#valueOf(Object)}. Subclasses are free to override
217 * this behavior.
218 * @param bogus bla
219 */
220 @Override public String toString() {
221     String bla = String.format("a %d", "abd");
222
223     private static final long serialVersionUID = 747826592375603043L;
224 }
```

Error Message (Overlaid):

[Formatter, InvalidFormatConversion]: Formatter has invalid conversion 'd' for type 'java.lang.String'. See <http://go/invalid-format-conversion>.

Bottom Panel:

- Pending Changes
- Errors
 - Build Errors
 - j/c/g/common/base/Pair.java:
 - 12. Line 218 Col 13 [DocComments, DocCommentsParams]: The parameter name 'bogus' in the comment does not match with any of the parameters. See <http://go/sap-doccomments-analyzer>.

Collaborate

Code Search

TAP

Sponge

Buganizer

Data Search

Rapid

Sigma

YACS

Search CLs

?

Q

Changelist

Client

★ CL 147867437 by ghasemloo

Submitted

Reply...

Reviewers

CC

Bugs

Modify CC

Roll back...

OCL

Submitted

Workspace

Rapid

Internal BuildFox protos for storage of jobs and leases and the leasing service.

Based on the following doc:
<http://go/buildfox-internal-protos>

runfox.proto, srcfox.proto, and bazel.proto are initial attempts for minimal functionality needed and are expected to be updated later.

Score

LGTM

Approval

Analysis

Builder

Presubmit

Presubmit:CheckTests

Submit

TapPresubmit

From earlier snapshot(s): DeletedArtifactAnalyzer

Files

Analysis

Progression

Expand diffs

Run analyses

?

Order by:

SmartSort

		Comments	Inline	Modified	Delta
<input type="checkbox"/>	BUILD	Added	ericburnett: 1	Diff	Feb 17 69
<input type="checkbox"/>	ascii.proto	Added	pkm: 1 sethkoehler: 1	Diff	Feb 17 9
<input type="checkbox"/>	bazel.proto	Added		Diff	Feb 17 10
<input type="checkbox"/>	job.proto	Added	ericburnett: 3 ghasemloo: 2 pkm: 3 sethkoehler: 6	Diff	Feb 17 75
<input type="checkbox"/>	lease.proto	Added	ericburnett: 3 pkm: 3 sethkoehler: 1	Diff	Feb 17 44
<input type="checkbox"/>	lessor.proto	Added	sethkoehler: 1	Diff	Feb 17 65
<input type="checkbox"/>	runfox.proto	Added		Diff	Feb 17 20
<input type="checkbox"/>	srcfox.proto	Added		Diff	Feb 17 19
					311

“Please fix”

```
<!-- Pull in styles for the Linechart library. -->
<style jsuse="//java/com/google/gws/common/linechart/shared_style.html#CommonS
type"></style>

<!-- This is some scary CSS! -->
<style>
div:last-child {
  float: left;
}

div:not(.g) {

  float: right;
  background-image:url('paper.gif');
}

.g {
  box-shadow: 10px 10px 5px #888888;
}
</style>

<!--
  TODO(limban): Pass ClientType in the proto, since we don't wanna figure it o
```

▼ CssAnalyzer The :not selector doesn't work in IE8

[Please fix](#)

[Not useful](#)

Show me the fix

```
package com.google.devtools.staticanalysis;
```

```
public class Test {
```

▼ **Lint** Missing a Javadoc comment.

Java
1:02 AM, Aug 21

[Please fix](#)

[Not useful](#)

```
    public boolean foo() {  
        return getString() == "foo".toString();  
    }
```

▼ **ErrorProne** String comparison using reference equality instead of value equality
(see <http://code.google.com/p/error-prone/wiki/StringEquality>)

StringEquality
1:03 AM, Aug 21

[Please fix](#)

Suggested fix attached: [show](#)

[Not useful](#)

```
    }  
  
    public String getString() {  
        return new String("foo");  
    }  
}
```

“Apply Fix”

//depot/google3/java/com/google/devtools/staticanalysis/Test.java

```
package com.google.devtools.staticanalysis;
```

```
public class Test {  
    public boolean foo() {  
        return getString() == "foo".toString();  
    }  
}
```

```
    public String getString() {  
        return new String("foo");  
    }  
}
```

```
package com.google.devtools.staticanalysis;
```

```
import java.util.Objects;
```

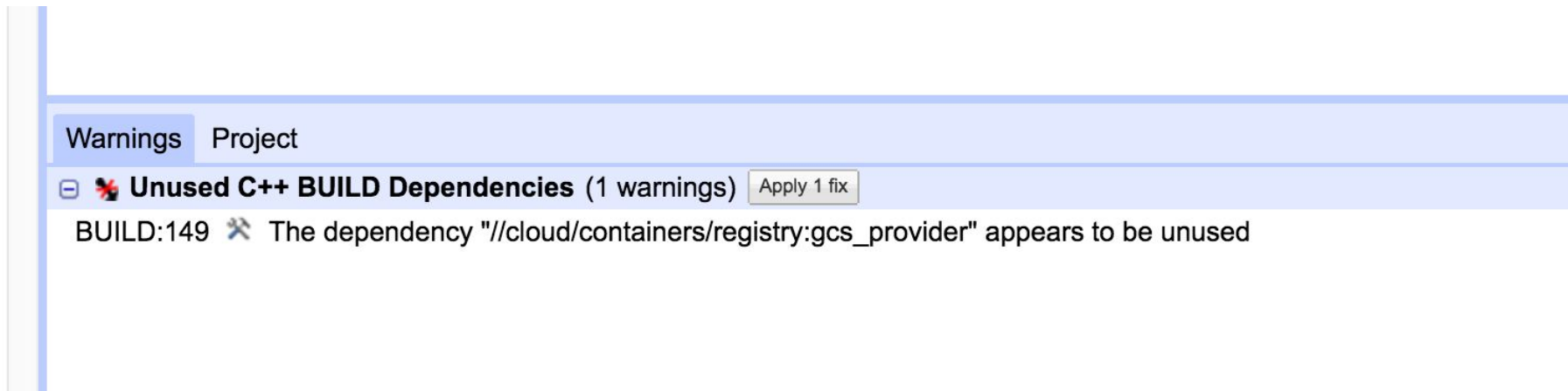
```
public class Test {  
    public boolean foo() {  
        return Objects.equals(getString(), "foo".toString());  
    }  
}
```

```
    public String getString() {  
        return new String("foo");  
    }  
}
```

Apply


Cancel


Fix it for me



The screenshot shows a warning panel in a code editor. At the top, there are two tabs: "Warnings" and "Project", with "Warnings" being the active tab. Below the tabs, the panel title is "Unused C++ BUILD Dependencies (1 warnings)" with a red bug icon on the left and an "Apply 1 fix" button on the right. The main content of the panel shows a single warning: "BUILD:149" followed by a blue wrench icon and the text "The dependency '//cloud/containers/registry:gcs_provider' appears to be unused".

Warnings Project

 **Unused C++ BUILD Dependencies** (1 warnings) [Apply 1 fix](#)

BUILD:149  The dependency "//cloud/containers/registry:gcs_provider" appears to be unused

Code submitted... test continuously

Provide real-time information to build monitors

- Identify failures.
- Identify culprit changes.

Develop Safely

- Sync to last green changelist.
- Identify whether changes break the build before submitting.

Provide frequent green builds for cutting releases

- Show results of all testing together.
- Allow release tooling to choose a green build.

Code submitted... test continuously

Continuously runs 4.5M tests as changes are submitted

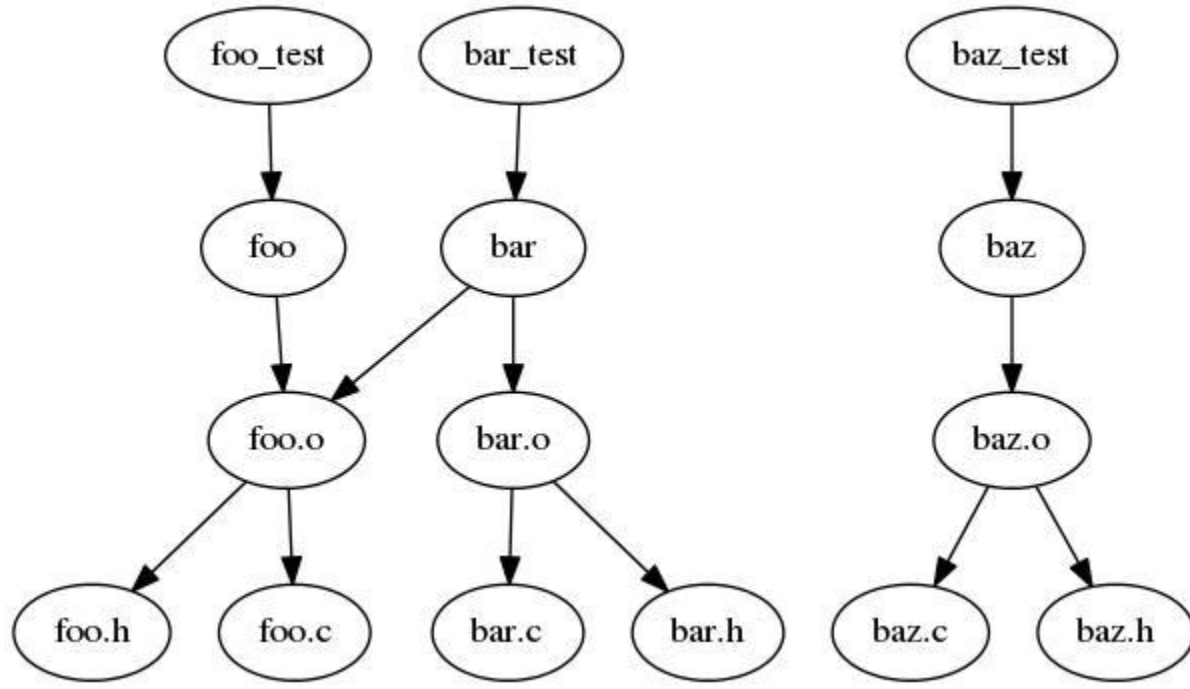
- Only “triggers” a test if the test depends (transitively) on the change
- Each test runs in 2 distinct flag combinations

Records the pass / fail result for each test in a database

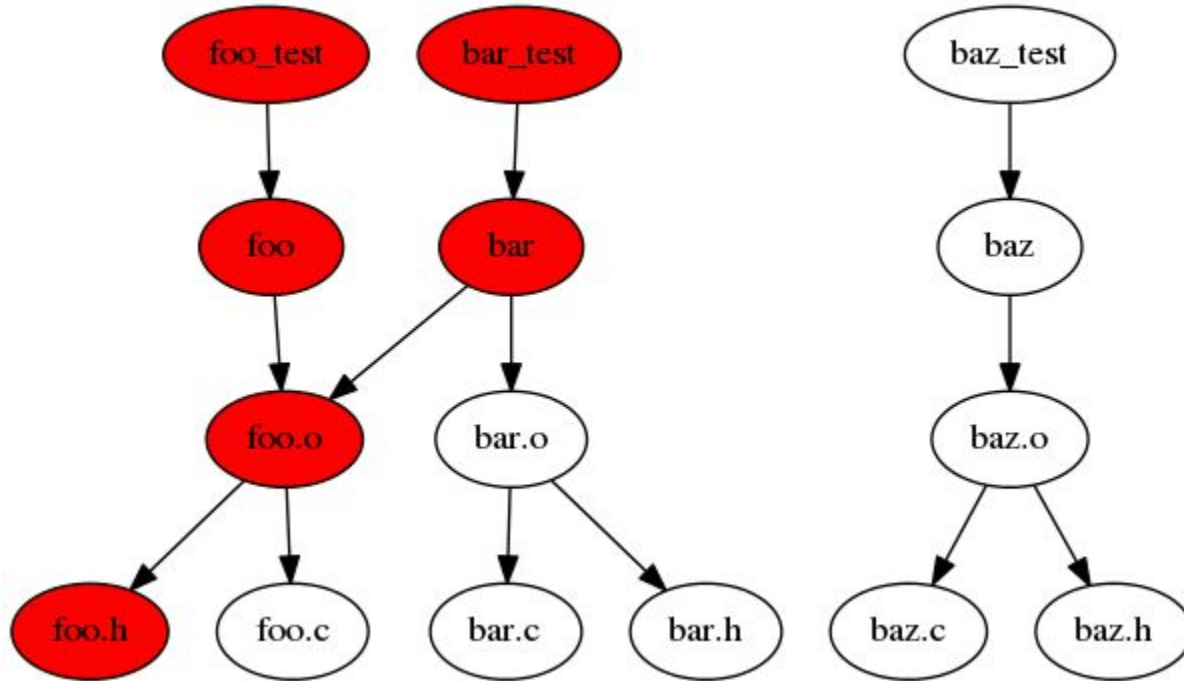
- Each run is uniquely identified by the test + flags + change
- We have 2 years of results for all tests

See: prior [deck](#) about Google CI System, See this [paper](#) about piper and CLs

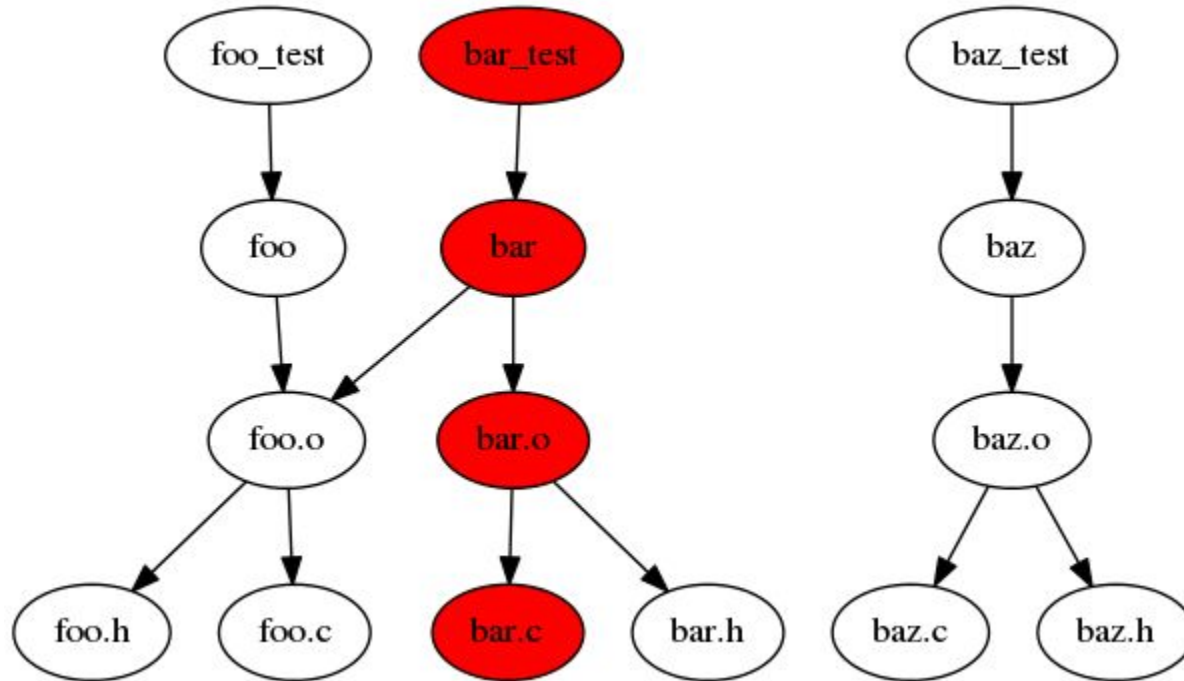
Regression Test Selection (RTS)

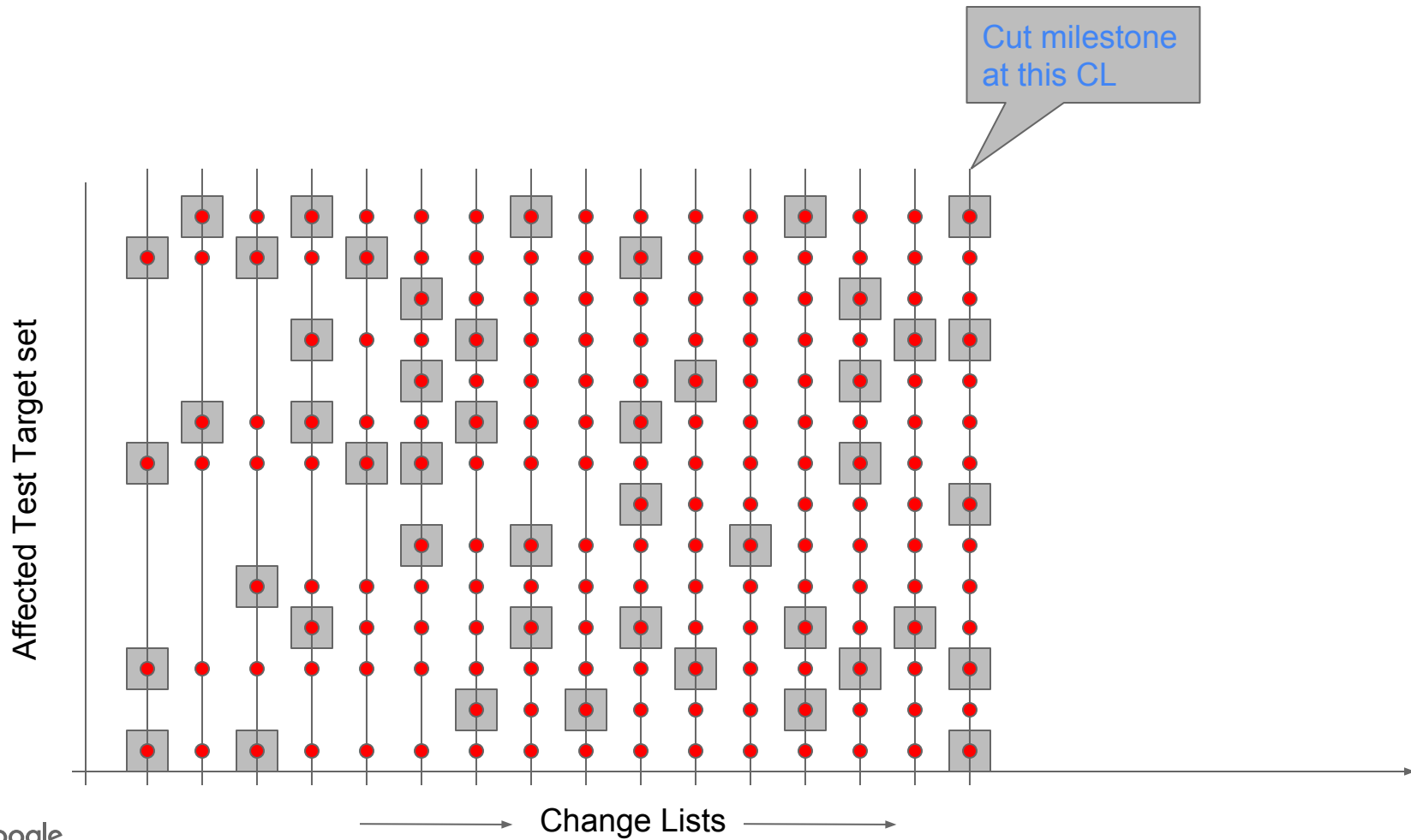


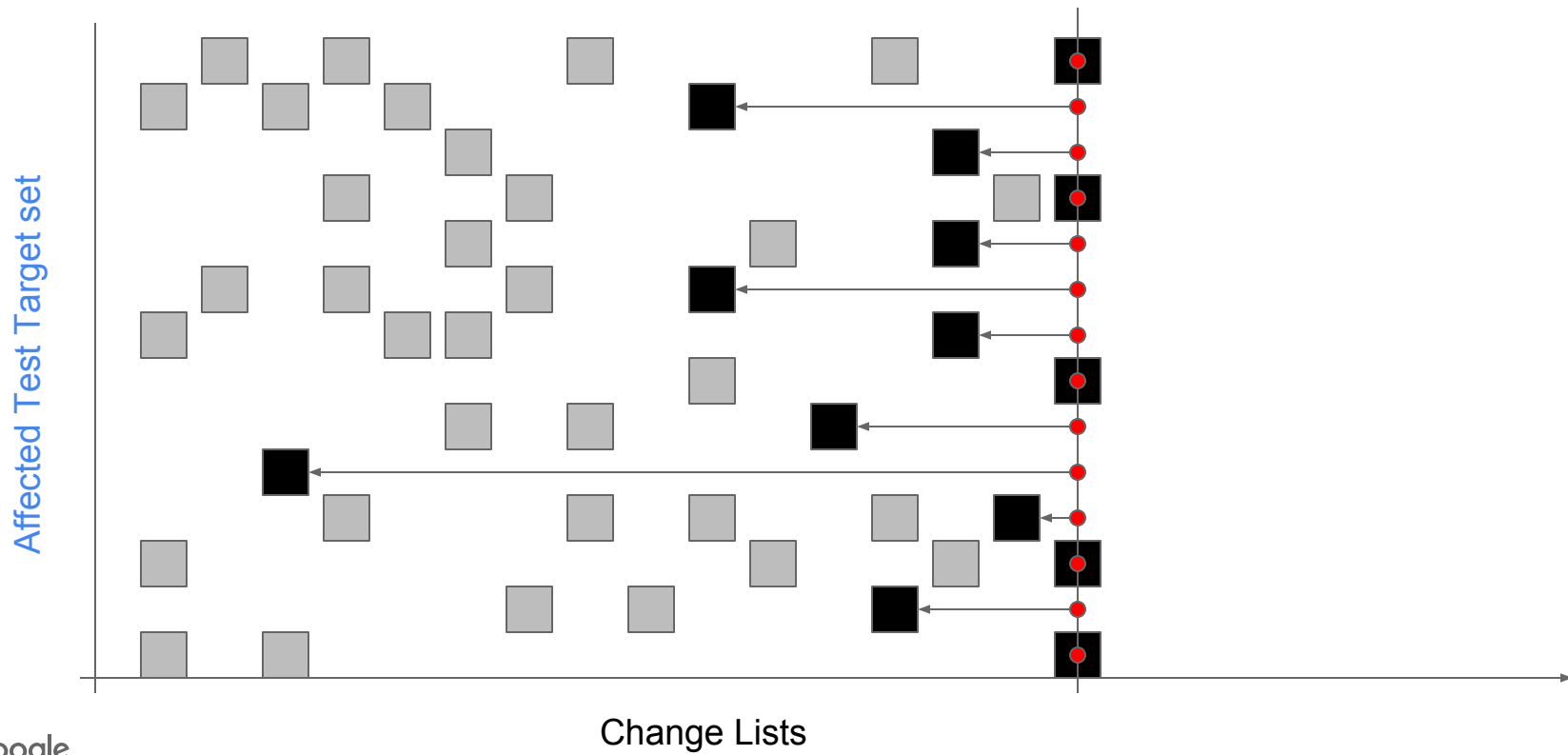
Regression Test Selection (RTS)



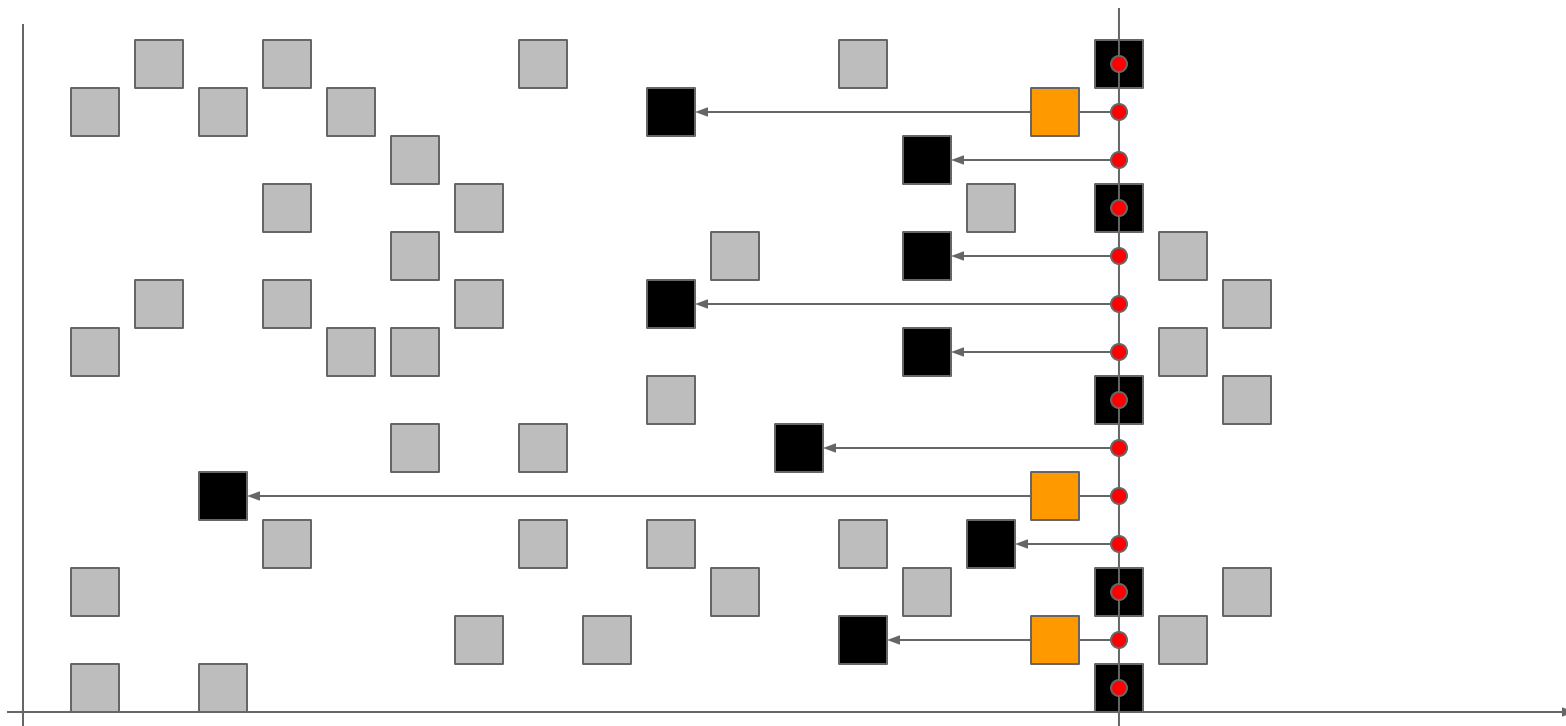
Regression Test Selection (RTS)



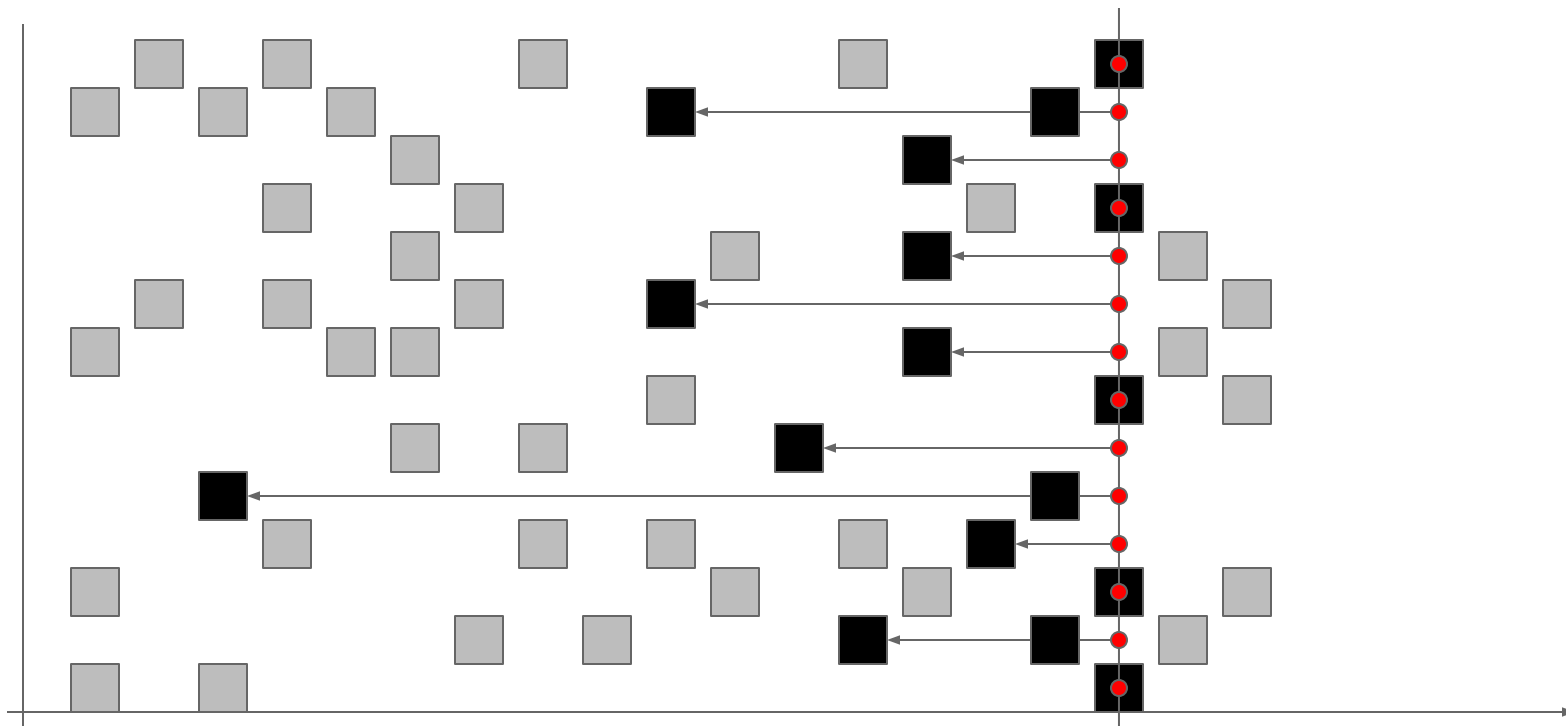




Affected Test Target set

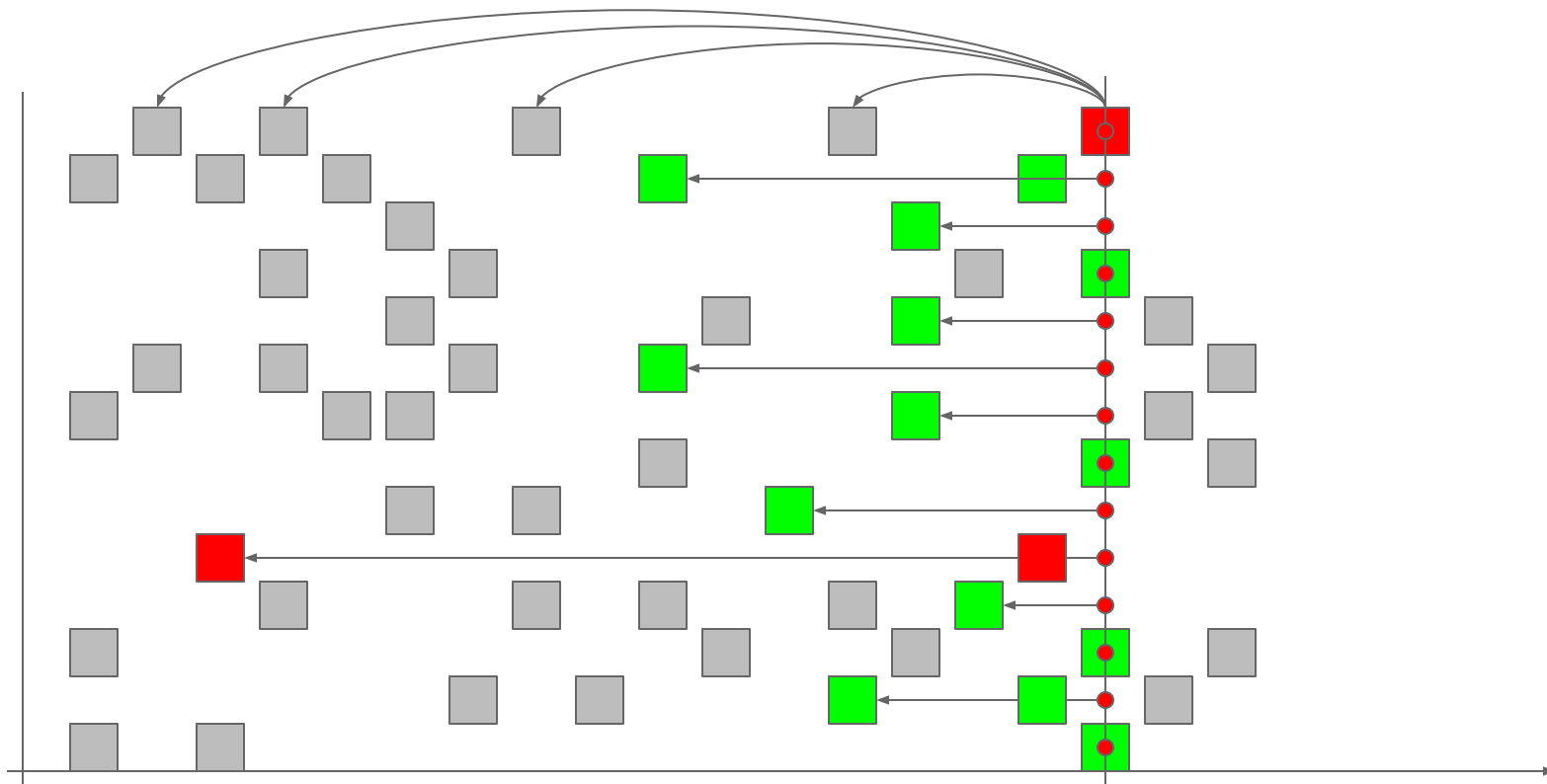


Affected Test Target set



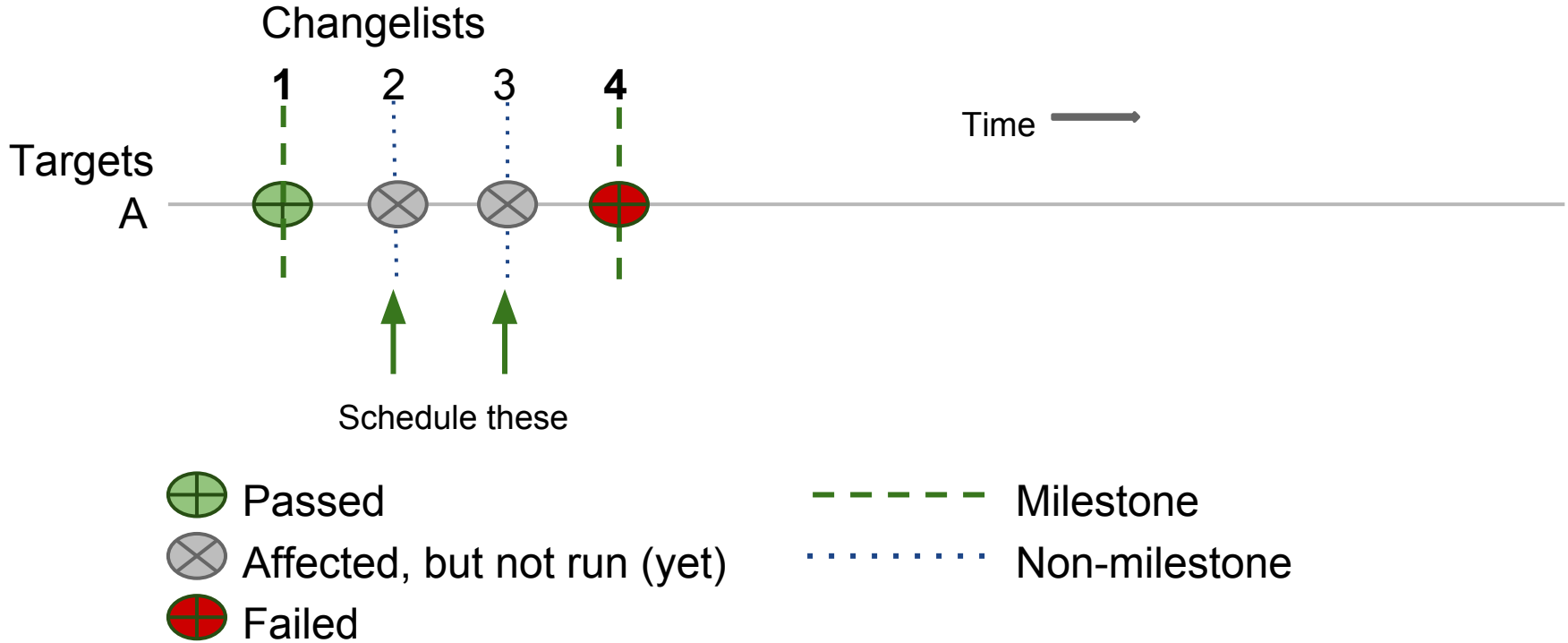
Change Lists

Affected Test Target set



Change Lists






Cuprit Finding - Transition to Fail



Cuprit Finding - Transition to Fail



A: Change 3 broke test A.

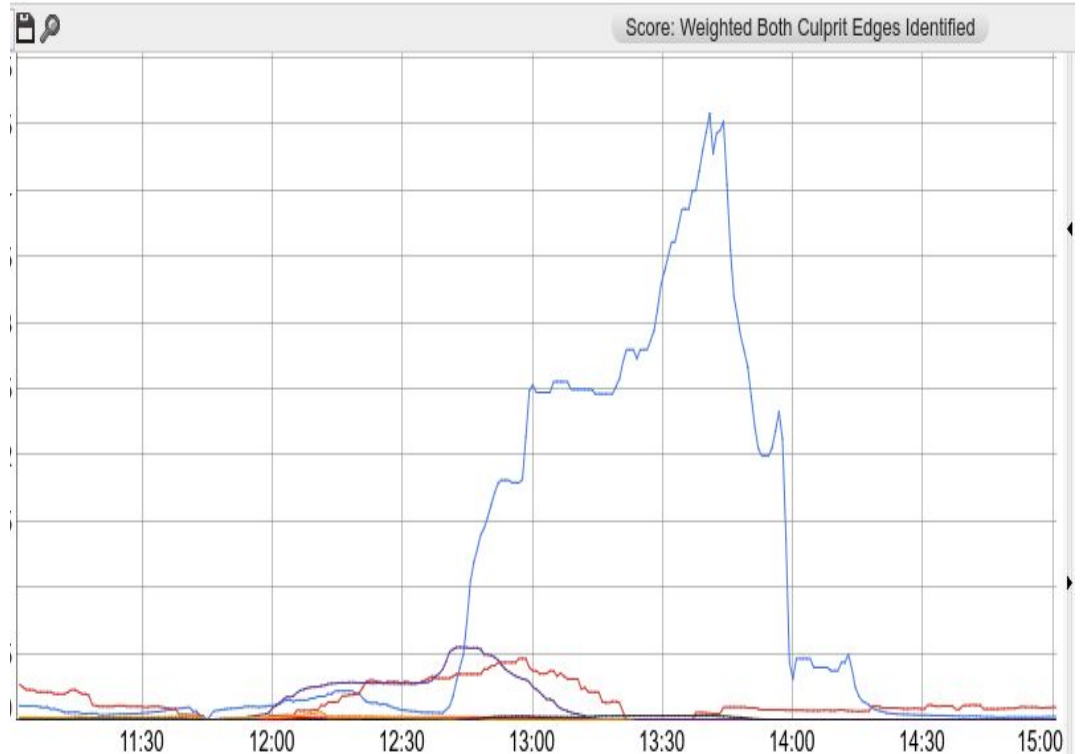
-  Passed
-  Affected, but not run (yet)
-  Failed
-  Milestone
-  Non-milestone

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

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





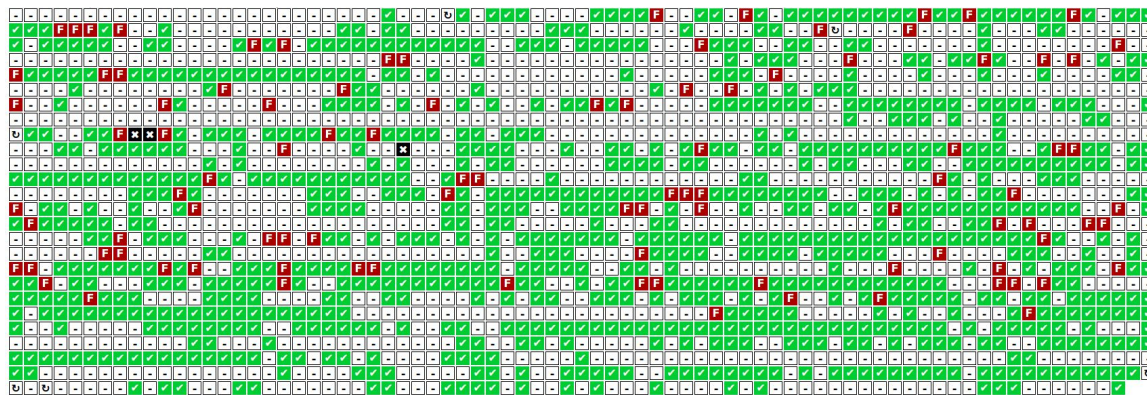
WHY DID IT HAVE
TO BE FLAKES!

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 accepted [Paper](#) (by Atif Memon) at ICSE 2017

Flaky Tests

- Test [Flakiness](#) is a huge problem
- Flakiness is a test that is observed to both Pass and Fail with the same code
- We observe that 84% of transitions from Pass -> Fail are flakes!
- Almost 16% of our 4.5M tests have some level of flakiness
- Flaky failures frequently block and delay releases
- We spend between 2 and 16% of our CI compute resources re-running flaky tests



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

The screenshot shows a web interface for managing flaky tests. At the top right, there are links: [flakiness help](#), [file a bug](#), [feedback](#), and [20% projects](#). Below this is a search bar with the placeholder text "Search for a tap project, guitar project, test target or test method...". The search bar contains "tap project" and "tap". To the right of the search bar is a "max days: 5" dropdown and a "Search" button. Below the search bar is a disclaimer: "The flakiness data comes from TAP flake detection mechanism. It includes data from tests running on TAP, guitar and tests from build rules annotated with flaky=1. However, it does not include flaky compilation failures. The information displayed is the test method failure from tests that failed due to flakiness."

The main section is titled "Flaky test executions from TAP project tap". To the right of this title are two filter boxes. The first box is labeled "Clustering:" and has three buttons: "exact match", "default", and "aggressive". The second box is labeled "Filter:" and has two buttons: "show all" and "hide test tagged as flaky". To the right of these boxes is a red button labeled "Help me fix this".

Below the filters, there is a link to a specific test failure: [com.google.testing.tap.testbroker.server.buildenqueuer.TestBrokerViaBESystemTest.testShouldWritePendingResultsAndTestRunRequestsForPostsubmit :](#) [//javatests/com/google/testing/tap/testbroker/server/buildenqueuer:LargeTestBrokerViaBESystemTests \(sponge\)](#) ran on 2016-10-31. To the right of this link is a note: "[source: [experimental flakes detector](#)]". Below the link is a red button labeled "Not a flake? Report it."

Below the link, there is a section titled "38 similar flakes from different targets" with an "expand" button. Below this is a code block showing a Java stack trace:

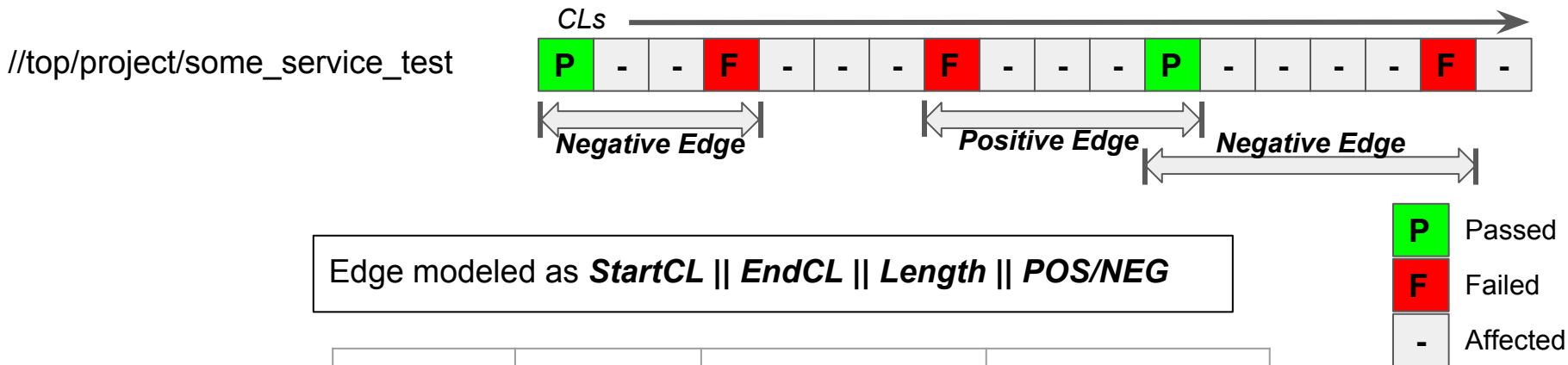
```
java.lang.AssertionError: Failed test because ChangelistNotifications is not empty after 30 seconds.
==== TASK ===== payload (ChangelistNotification) ===
changelist: 40000021
test {
  target_name: "[REDACTED]"
  rule_kind: "sh_test_rule"
}
    at org.junit.Assert.fail(Assert.java:89)
    [REDACTED]
```

At the bottom left of the code block, there is a note: "(stacktrace truncated)".

Flaky Test Infrastructure (continued)

- Identifying Flaky tests without re-running them
 - A. Follow intuition
 - Simple signal of $P \rightarrow F \rightarrow P$ patterns to indicate flakiness
 - B. Develop statistical models of features highly correlated with flakes
 - First models show promise - classifying 90% of the flakes correctly
 - C. Develop statistical models of features highly correlated with real failures
 - Deviations highly likely to be flakes
- Formally model flakes and their behavior

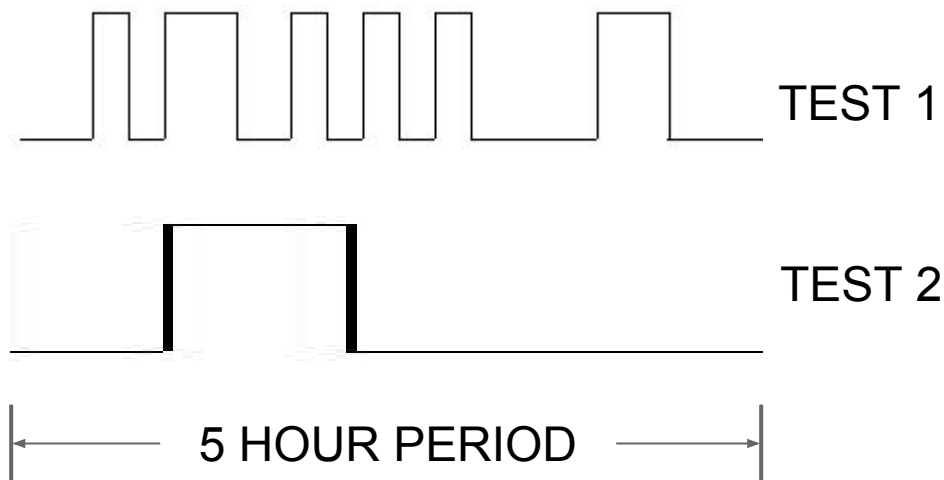
Modeling Test Target Behavior (via Edges)



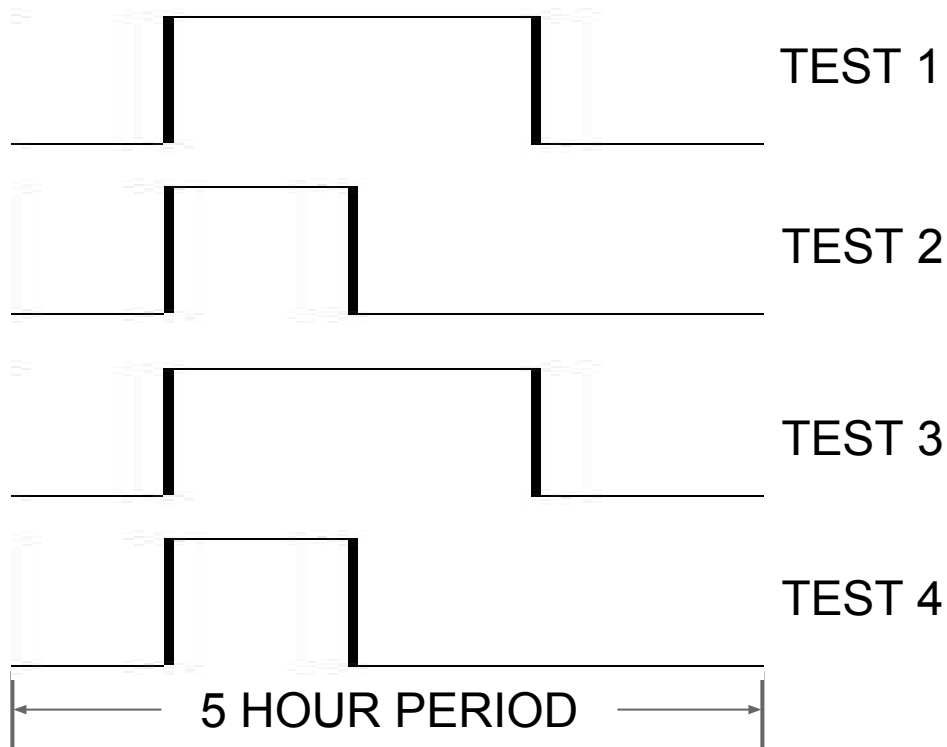
	All Edges	Confidently due to Flakes	Most likely not including Flakes
Positive	574,282	485,435 (84.5%)	88,847 (15.5%)
Negative	563,993	474,654 (84.2%)	89,339 (15.8%)

Take away message: Small % (1.5-2%) tests flakes (TAP spanner database/total targets in Feb11-Mar11 period); BUT, they lead to majority of edges (*edges are better indicators of overall impact of flakes*)

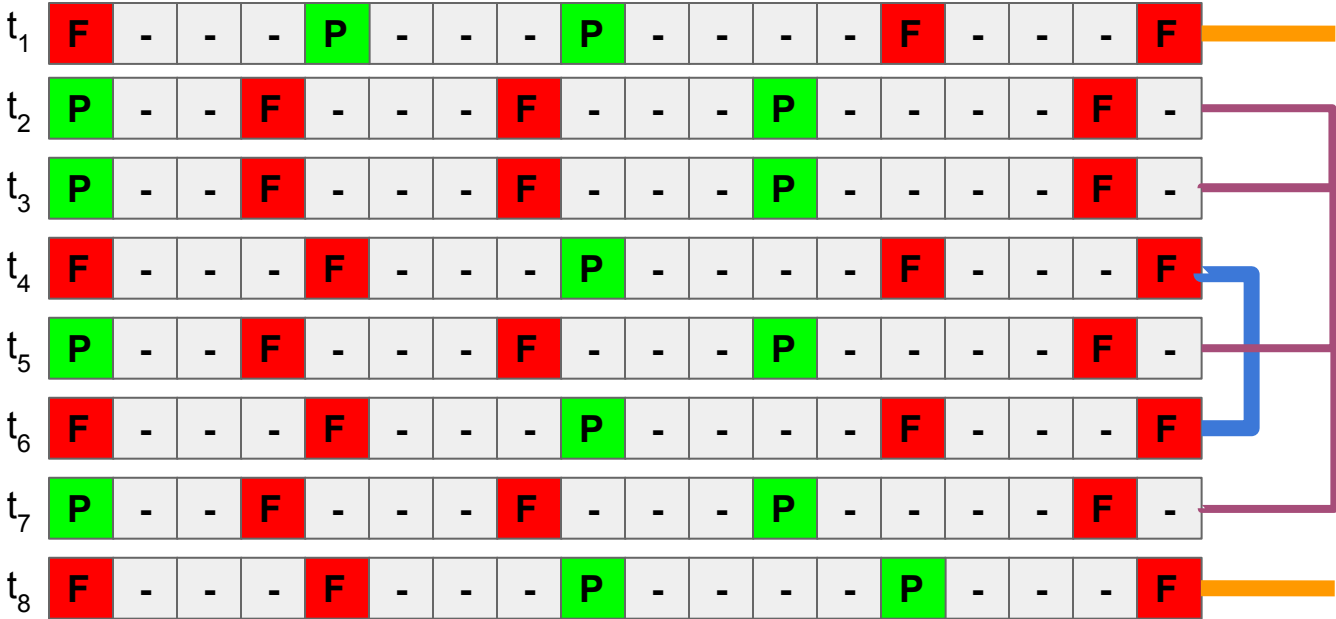
FLAKES HAVE LARGER NUMBER OF EDGES PER TIME PERIOD.



FLAKES ARE UNLIKELY TO SHARE THEIR HISTORIES WITH OTHERS.



Modeling Histories of Tests



“Length of Edge History” vs. Shared Outcomes

“Target History” = Concat All Edges over time period.
Multiple targets share history.

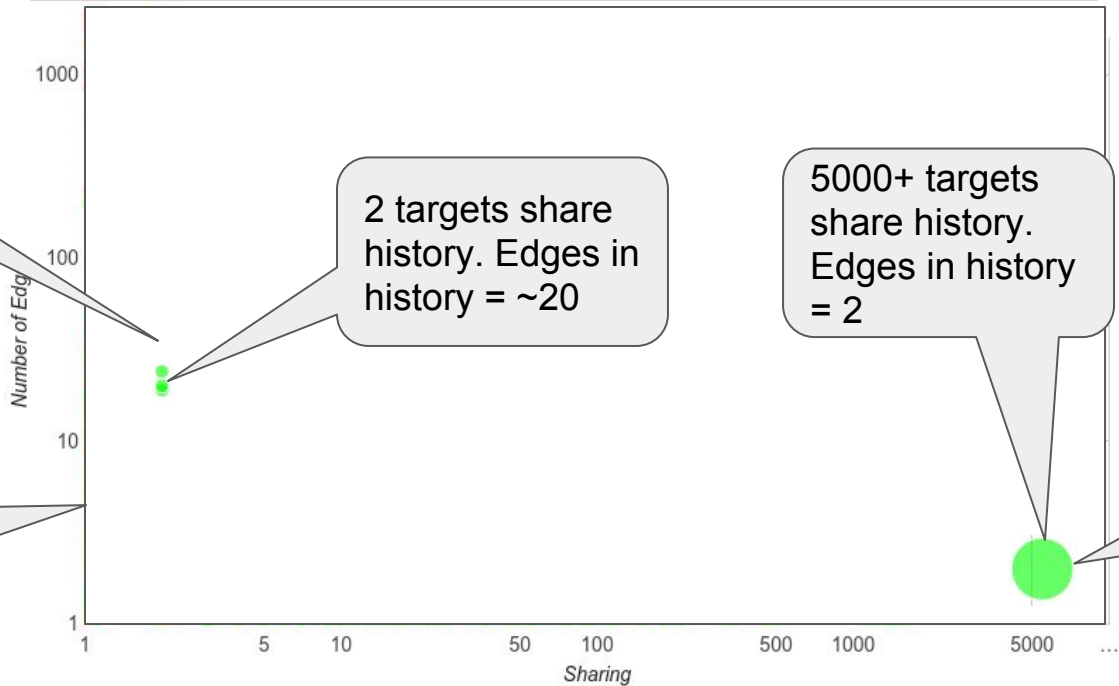
Very little
sharing (2) in
Sharing=2
column

2 targets share
history. Edges in
history = ~20

5000+ targets
share history.
Edges in history
= 2

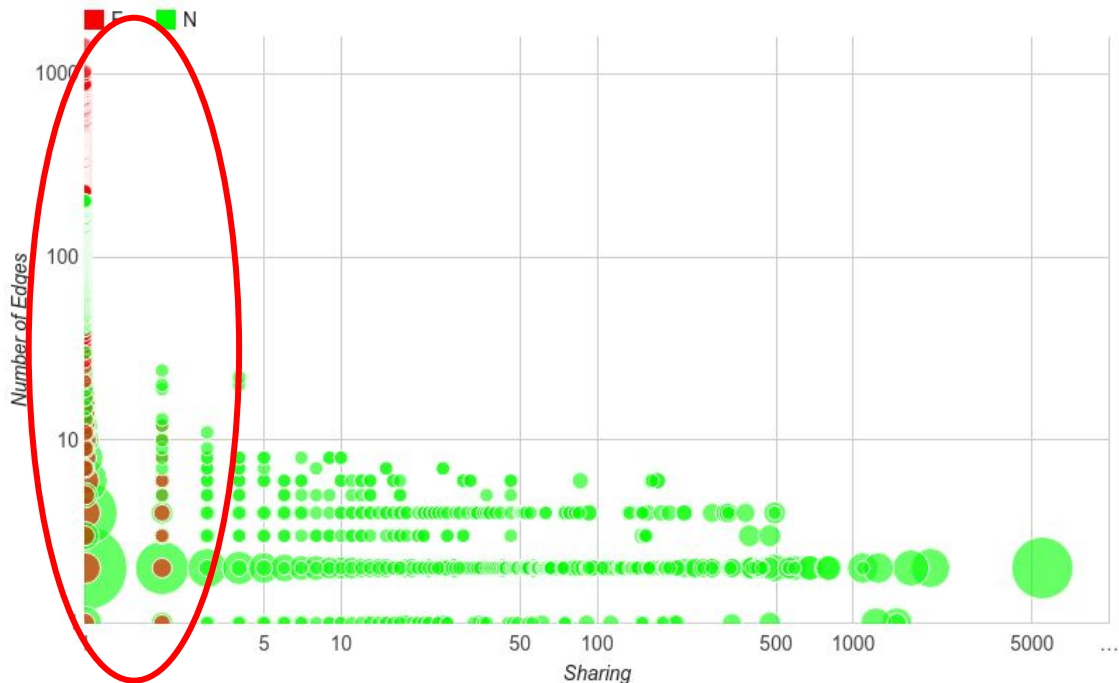
Lots of sharing

No sharing
along y-axis



“Length of Edge History” vs. Shared Outcomes

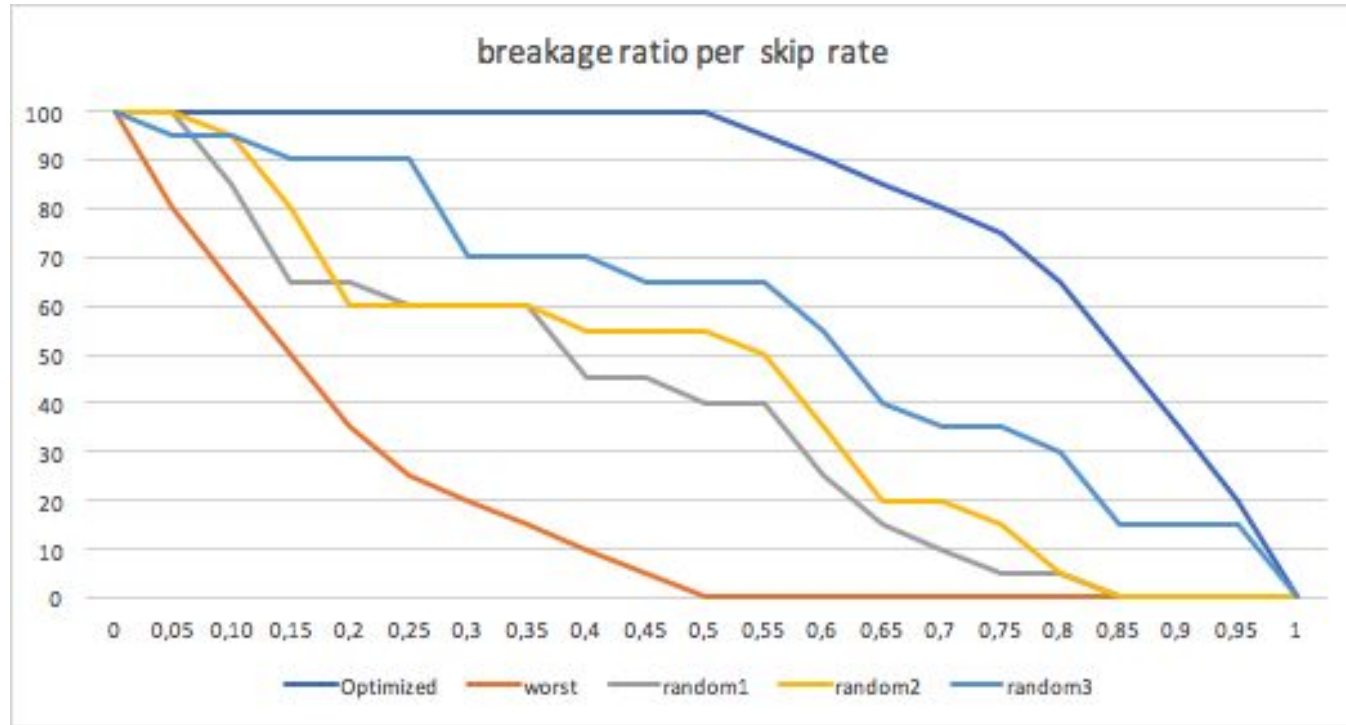
“Target History” = Concat All Edges over time period.
Multiple targets share history.



All Flakes lie in
“No Sharing” or
“Very little sharing”
area here

Take away message: Test targets that share history with other targets very unlikely to be flakes.
 (“degree of sharing” = signal for flake detection)

Future Directions



Scheduler testing framework - tests for safety and savings against historical record

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
- (ICSE 2015) "[Tricorder: Building a Program Analysis Ecosystem](#)" by Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Söderberg, Collin Winter