Google

Trends in Very Large Scale Continuous Integration

By Dr. Tim A. D. Henderson
Build, Test, Release
Google LLC
tadh@google.com
hackthology.com
linkedin.com/in/tadh/

Agenda

- What is Continuous Integration
- The Goals of Continuous Integration
- What is "Very Large Scale"
- Challenges Posed by Large Scale CI
- Solutions Space
- Trends

What is Continuous Integration?

Google

A **system** and *practice* of automatically merging changes into a source of truth for your organization's source code and related artifacts.

Merging Changes

Change A

```
@@ -133,18 +133,18 @@ func (g *gen) fnStmt
```

```
  for idx, param := range fn.params {
-   r := g.newRegister()
+   r := g.newRegister(param.ptype)
    g.symbols.Put(param.name, r)
    entry.add(&Instruction{
-       Op: ops.PRM, A: &Constant{value: idx}, Result: r})
+       Op:     ops.PRM,
+       A:      &Constant{value: idx, ctype: param.ptype},
+       Result: r})
  }
```

Merging Changes

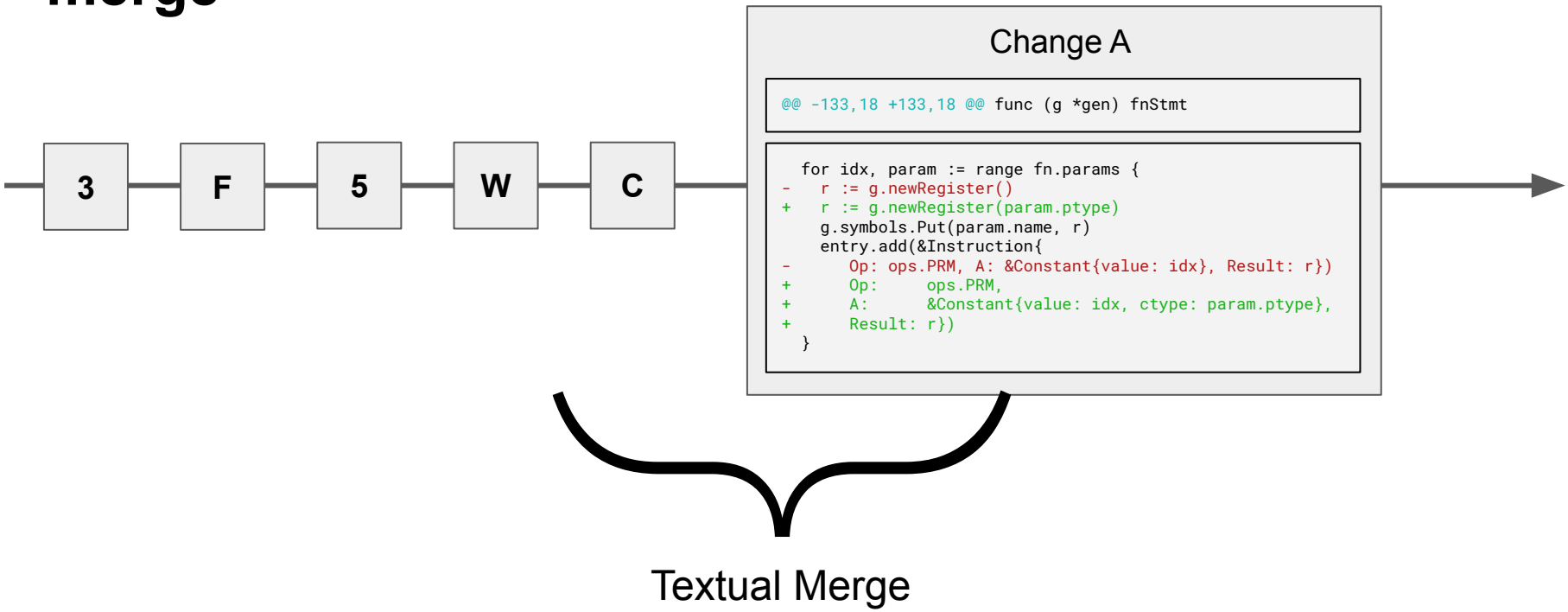
Change B

```
@@ -160,18 +160,22 @@ func (g *gen) createClosure
```

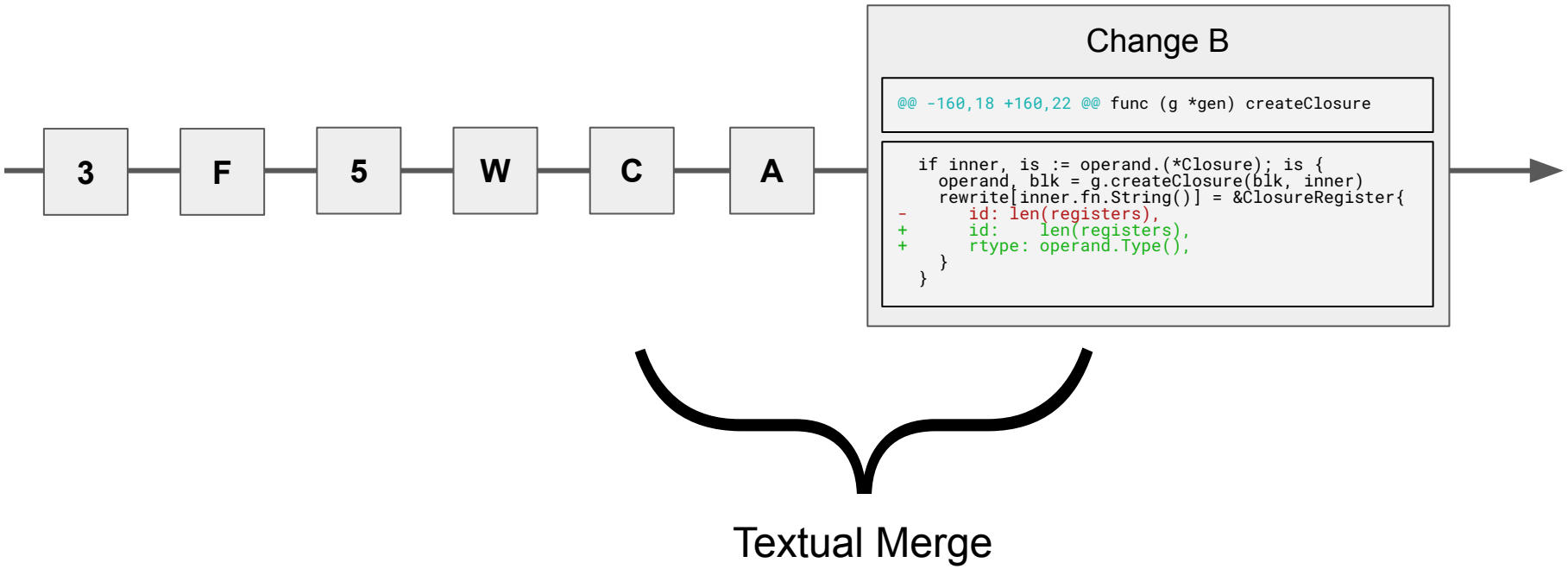
```
  if inner, is := operand.(*Closure); is {
    operand, blk = g.createClosure(blk, inner)
    rewrite[inner.fn.String()] = &ClosureRegister{
-       id: len(registers),
+       id:    len(registers),
+       rtype: operand.Type(),
    }
  }
```

Existing versions in your source of truth (main branch)

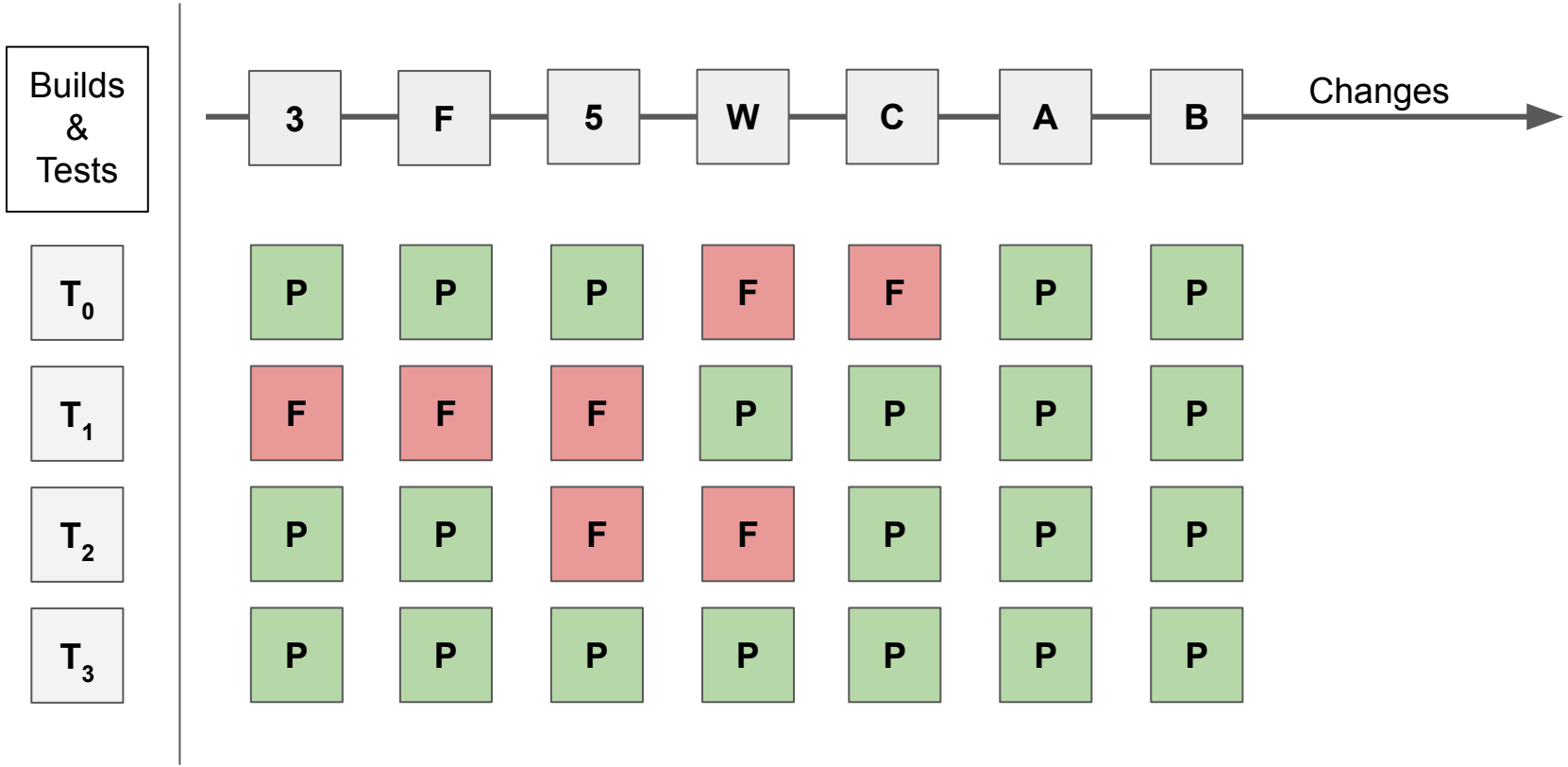
The new changes are merged using a text based merge



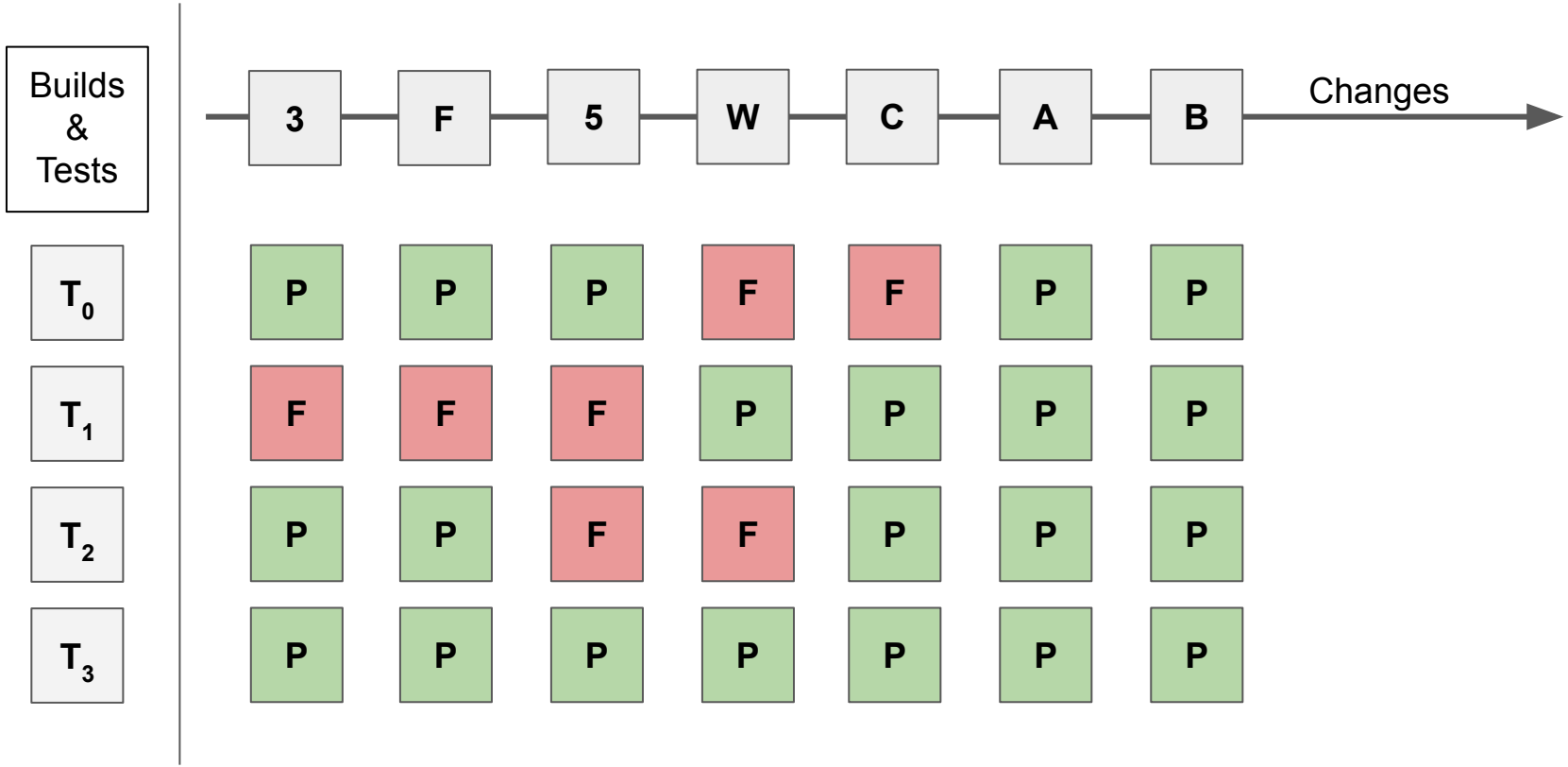
These changes are linearized into a specific order



Just merging isn't enough, we need to run tests.



Tests should be run both before the merge and after.



Google

What is Continuous Integration?

A system and practice of automatically merging changes into a source of truth for your organization's source code and related artifacts.

1. Automatically integrates new changes into the main branch in your source code management system.

2. Runs builds and tests to ensure the code still compiles and the tests pass.

3. May include additional functionality.

Google

Goals of Continuous Integration

Goals of Continuous Integration

1. Happy Developers!
    a. Main branch is not constantly broken
    b. Provides quick "dev loop" feedback
    c. Provides tools for managing debugging and fixing breakages
    d. Handle flaky tests

2. Trustworthy Releases
    a. Runs all tests which could affect the result for a release
    b. Provides results on-time at team defined frequency (ex. 1 per hour, 4 per day, 2 per week)
    c. Ensures releases can be made reliably
    d. Handle flaky tests

What is "Very Large Scale"?

Google

Prerequisite for Very Large Scale:

All code being integrated is integrated **into the same branch in the same repository.** Code which integrates into **different branches** or **different repositories** can be *efficiently sharded into separate CI instances.*

Very Large Scale

1. Supports commit submission rates **exceeding** the minimum time it takes to run a single build or test on the code base.

2. Multiple **resource management** techniques have been applied to manage resource demand.

3. Scale continues to grow and it remains an **ongoing** organizational priority to manage.

Scaling Factors

- Size of code base (# of lines)
- # of tests
- # of test configurations
- # of test environments (server, web, iOS, android, etc…)
- Frequency of commits
- Frequency of releases
- # of developers (users)
- # of distinct "projects" or "release artifacts"
- # of flaky (non-deterministic) tests
- # of flaky machines
- Complexity of test environment: hermetic unit test ⇒ multi-machine and platform end-to-end system tests.

Google

Solutions Space for Scaling Scenarios

Google

Solutions Space for Scaling Scenarios

1. Limit the number of commits which get tested.

2. Limit the number of tests which get run.

Research and Industrial Trends

Google

Understand which tests need to be run.

Use static analysis coarse (build dependencies) ⇒ fine (program dependence graph) grained to determine which tests are affected by a change.

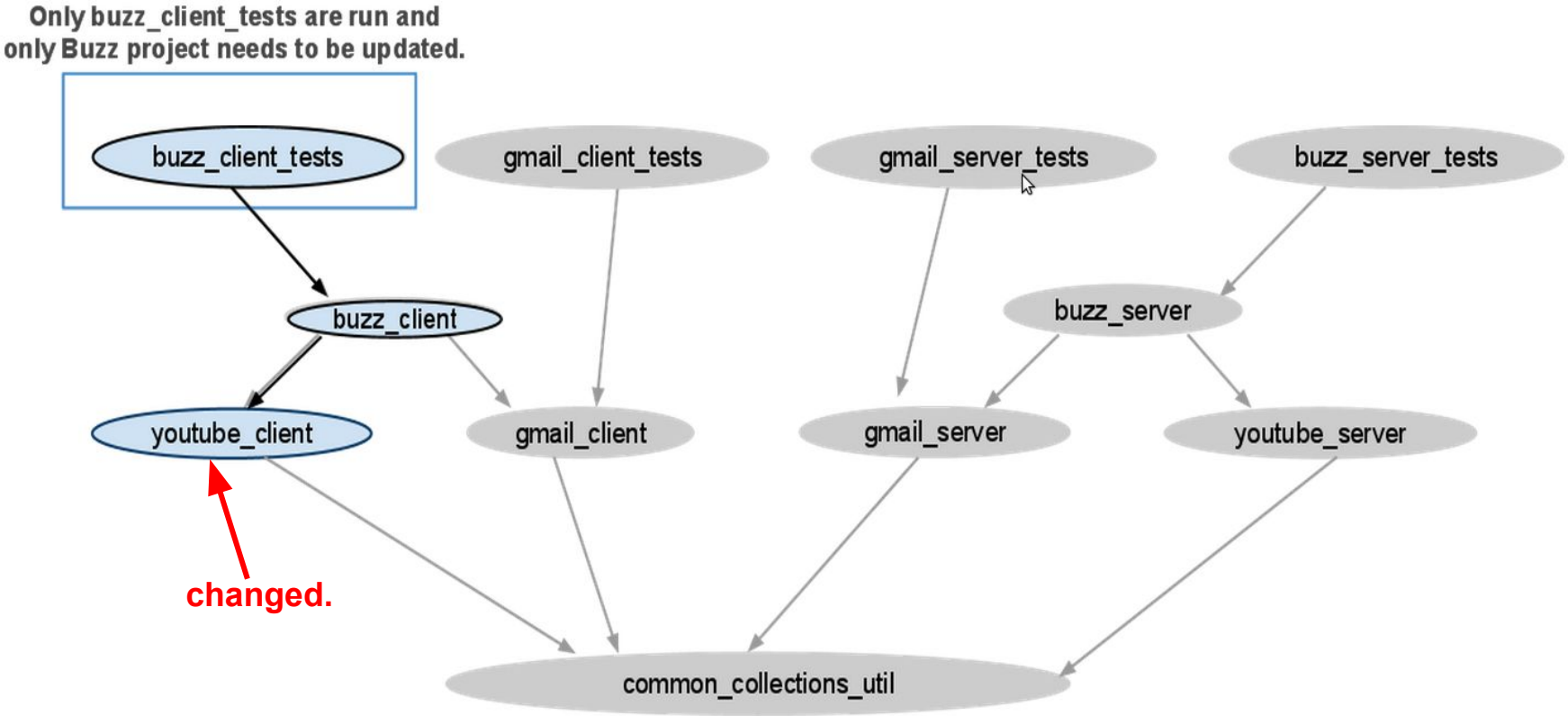
Pooja Gupta, Mark Ivey and John Penix. **Testing at the speed and scale of Google**. 2011. http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html

John Micco. **Tools for Continuous Integration at Google Scale.** Google Tech Talk. Google NYC. June 19 2012. https://youtu.be/KH2_sB1A6lA [**Google**]

S. Ananthanarayanan et al., "**Keeping master green at scale**," Proc. 14th EuroSys Conf. 2019, 2019, https://dx.doi.org/10.1145/3302424.3303970. [**Uber**]

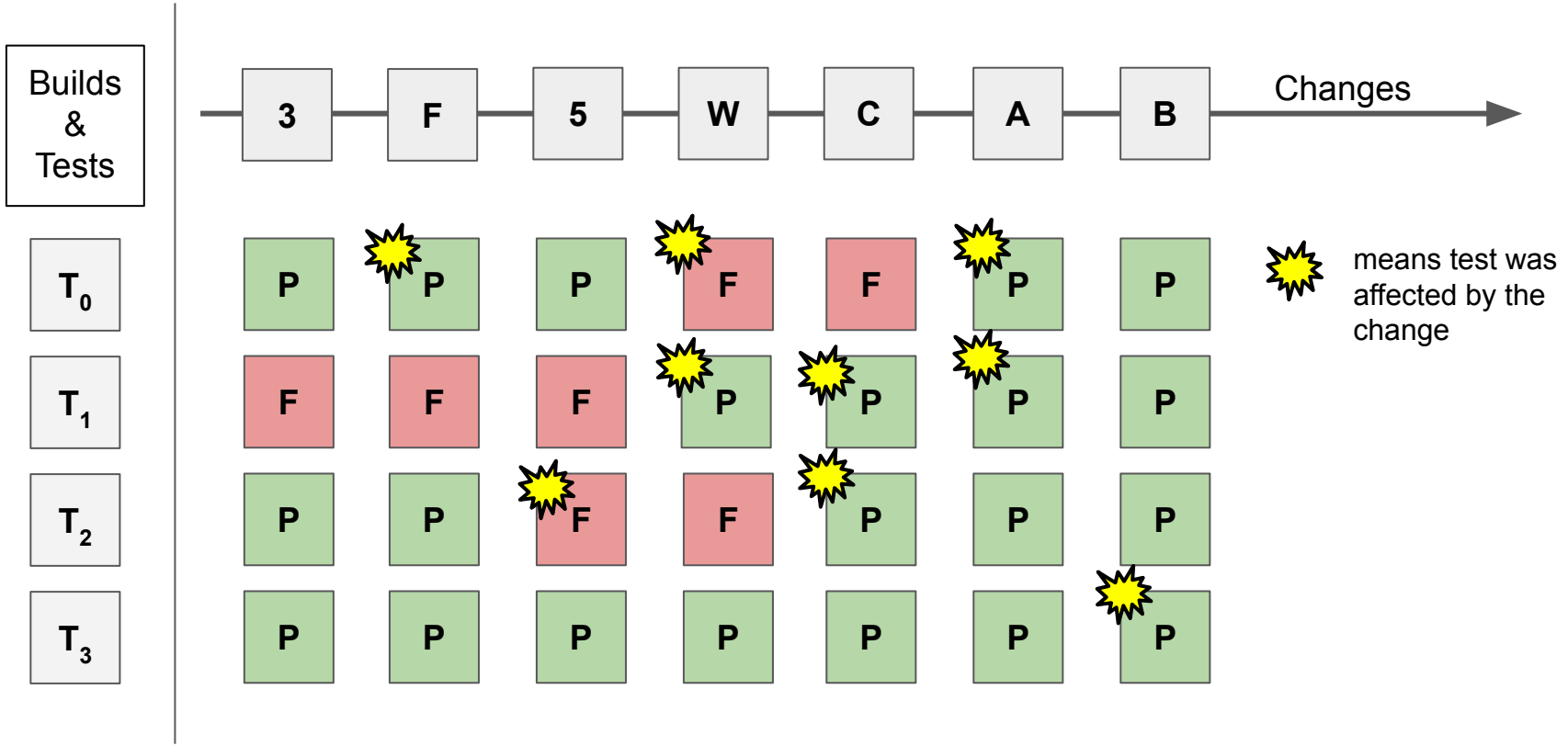
Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. 2015. **Practical Regression Test Selection with Dynamic File Dependencies**. In International Symposium on Software Testing and Analysis (ISSTA). 211–222.

Understand which tests need to be run.

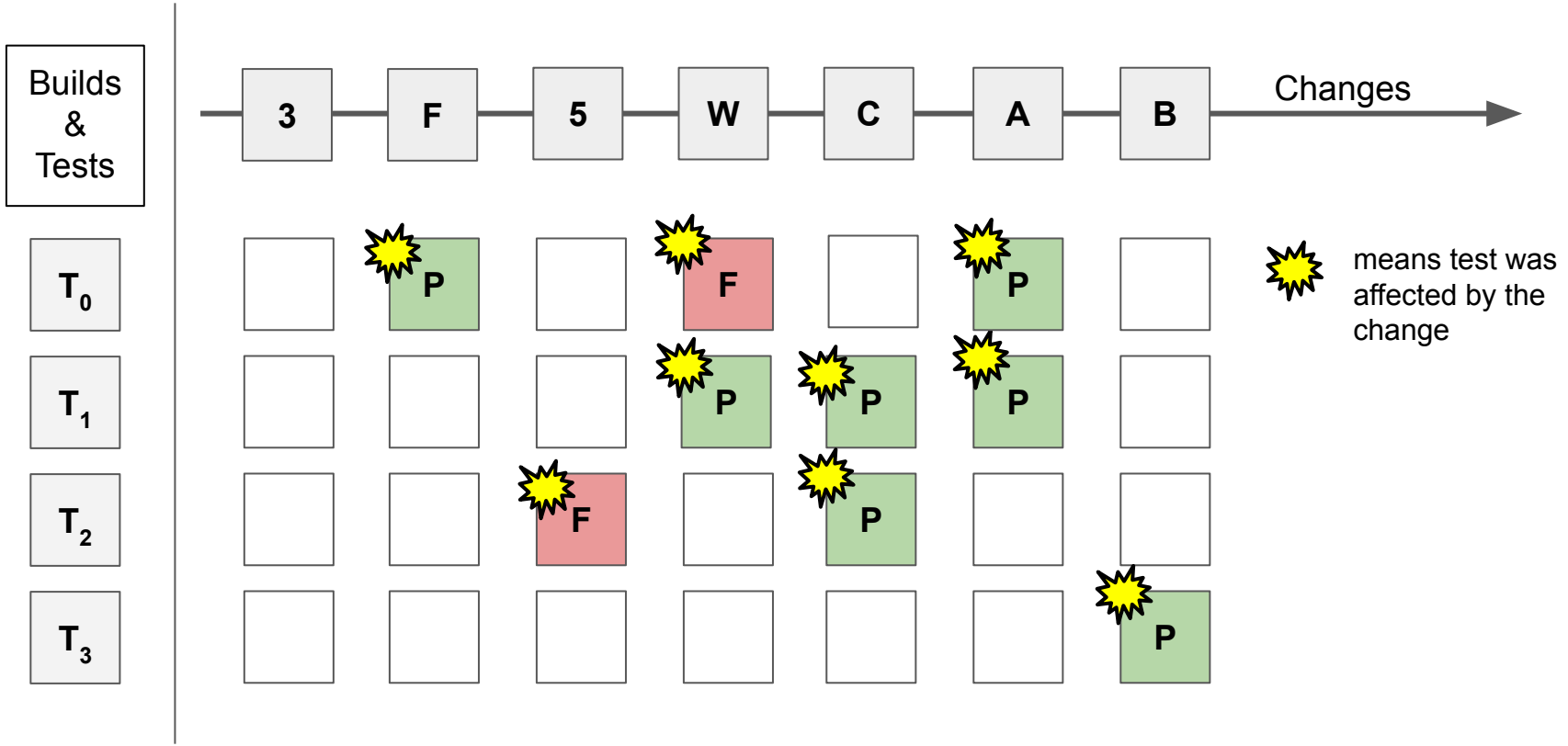


from: http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html

Not every test needs to be run every change



Huge savings to be had by skipping unaffected tests



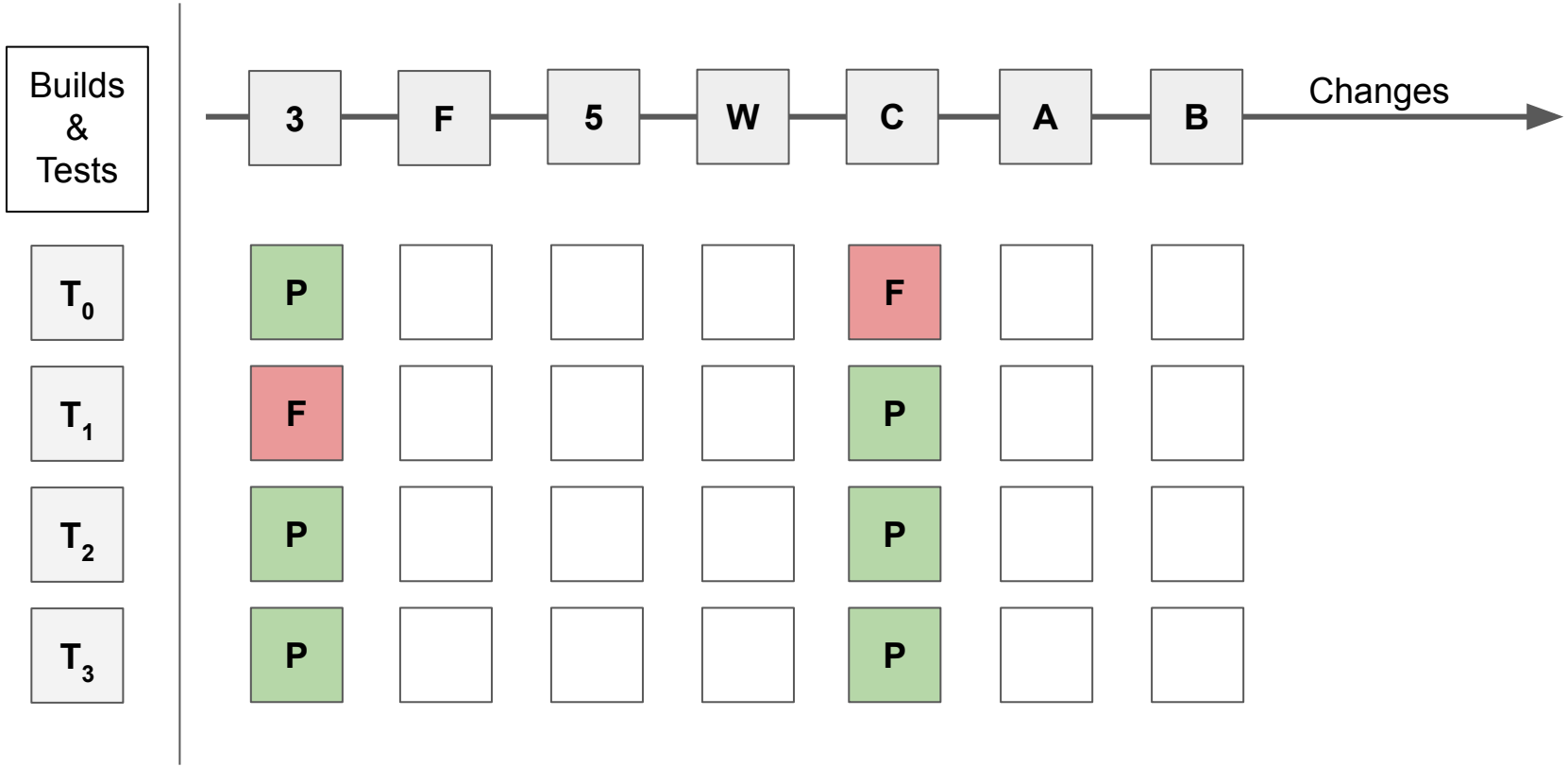
Throttling test executions to prevent delays

In order to manage the finite resource of build and test execution machines, do not run tests at every commit. Wait until the execution system will have resources and then schedule (enqueue) all tests which need to be run. Execution system should **prioritize** latency sensitive builds and tests.

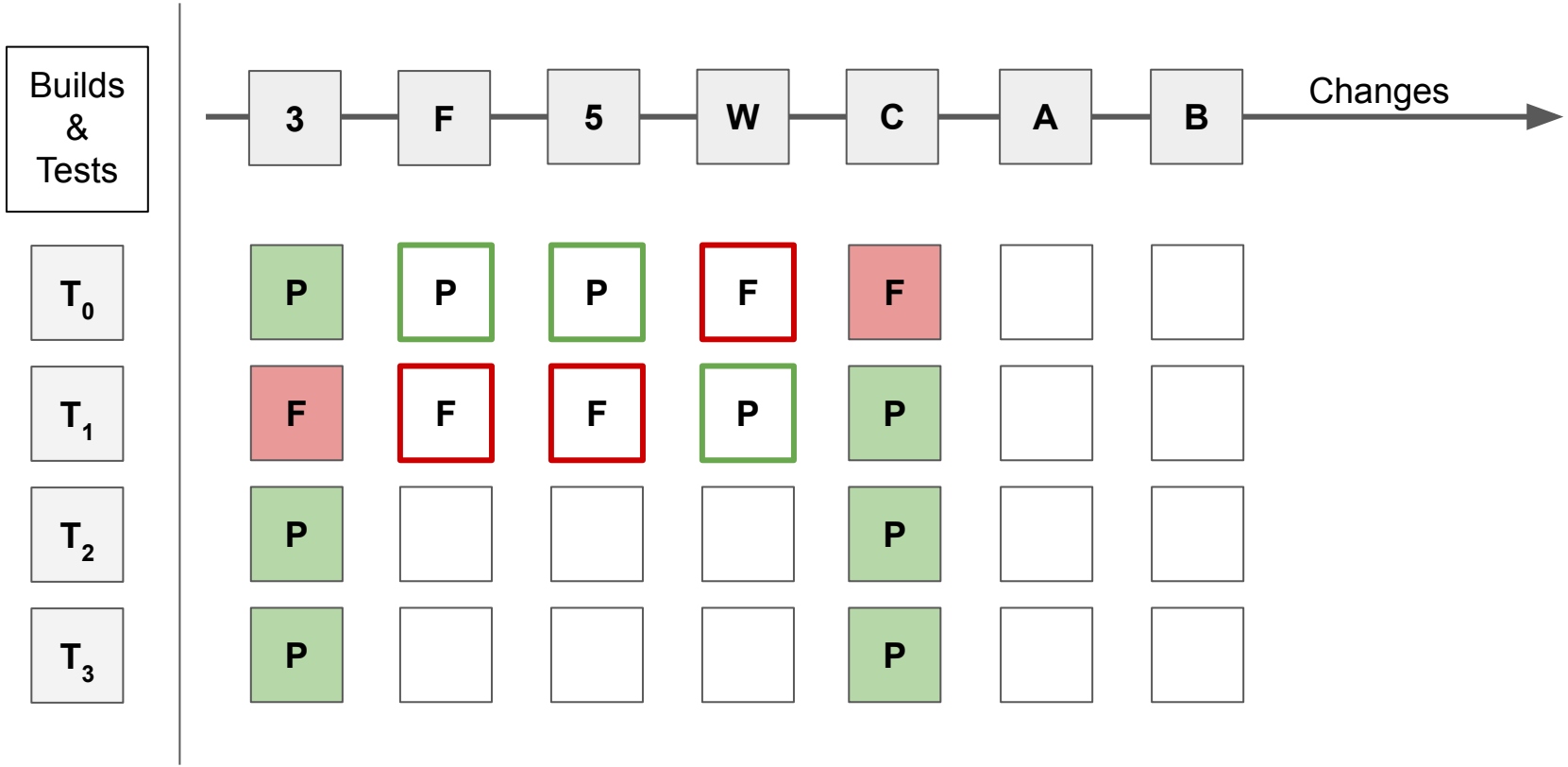
John Micco and Developer Infrastructure. "**Continuous integration at google scale.**" Eclipse Con 2016. [slides] **[Google]**

A. Memon et al., "**Taming Google-scale continuous testing**," International Conference on Software Engineering: Software Engineering in Practice Track ICSE-SEIP, 2017. https://dx.doi.org/10.1109/ICSE-SEIP.2017.16. **[Google]**

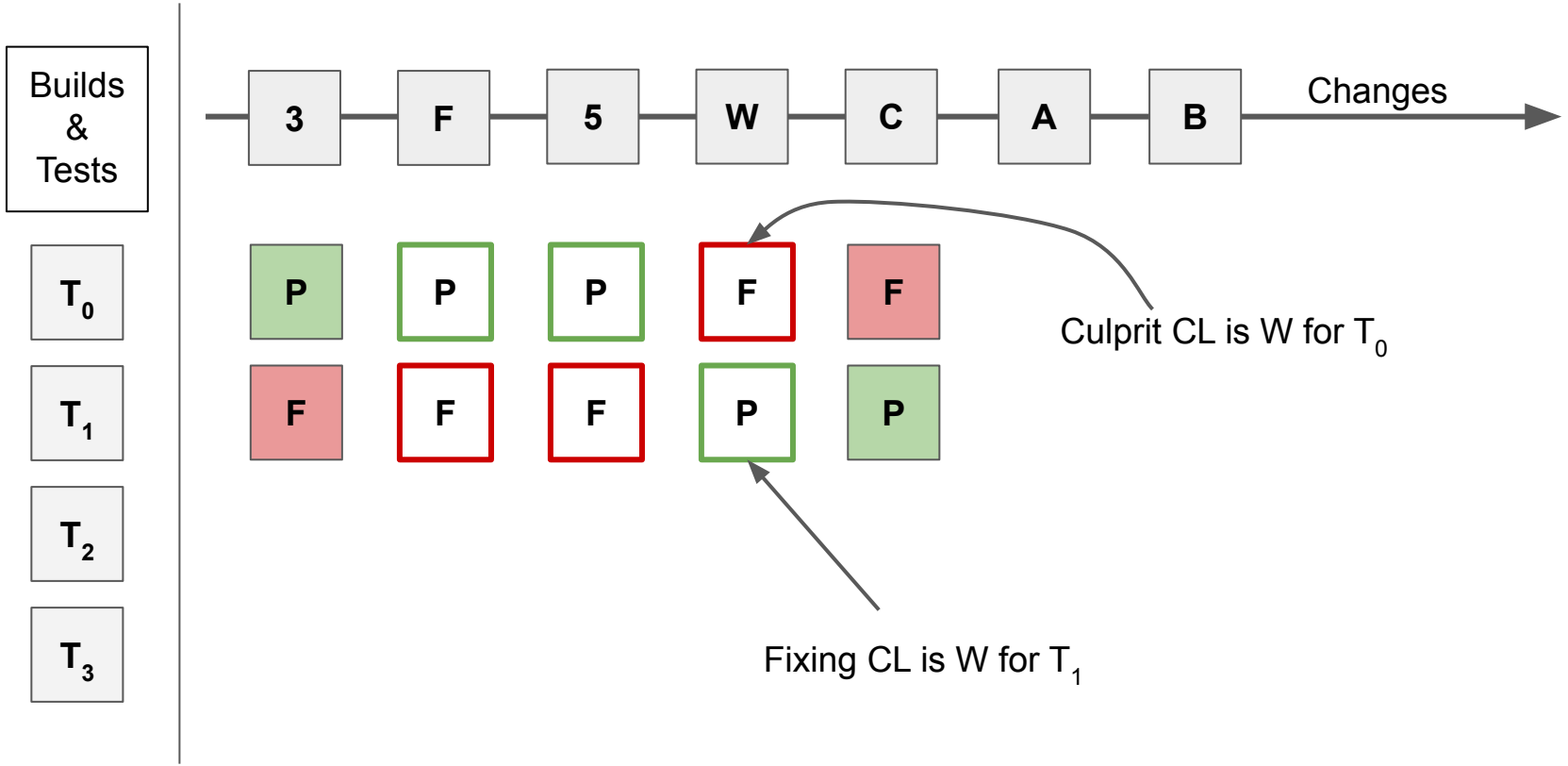
Throttling test executions to prevent delays



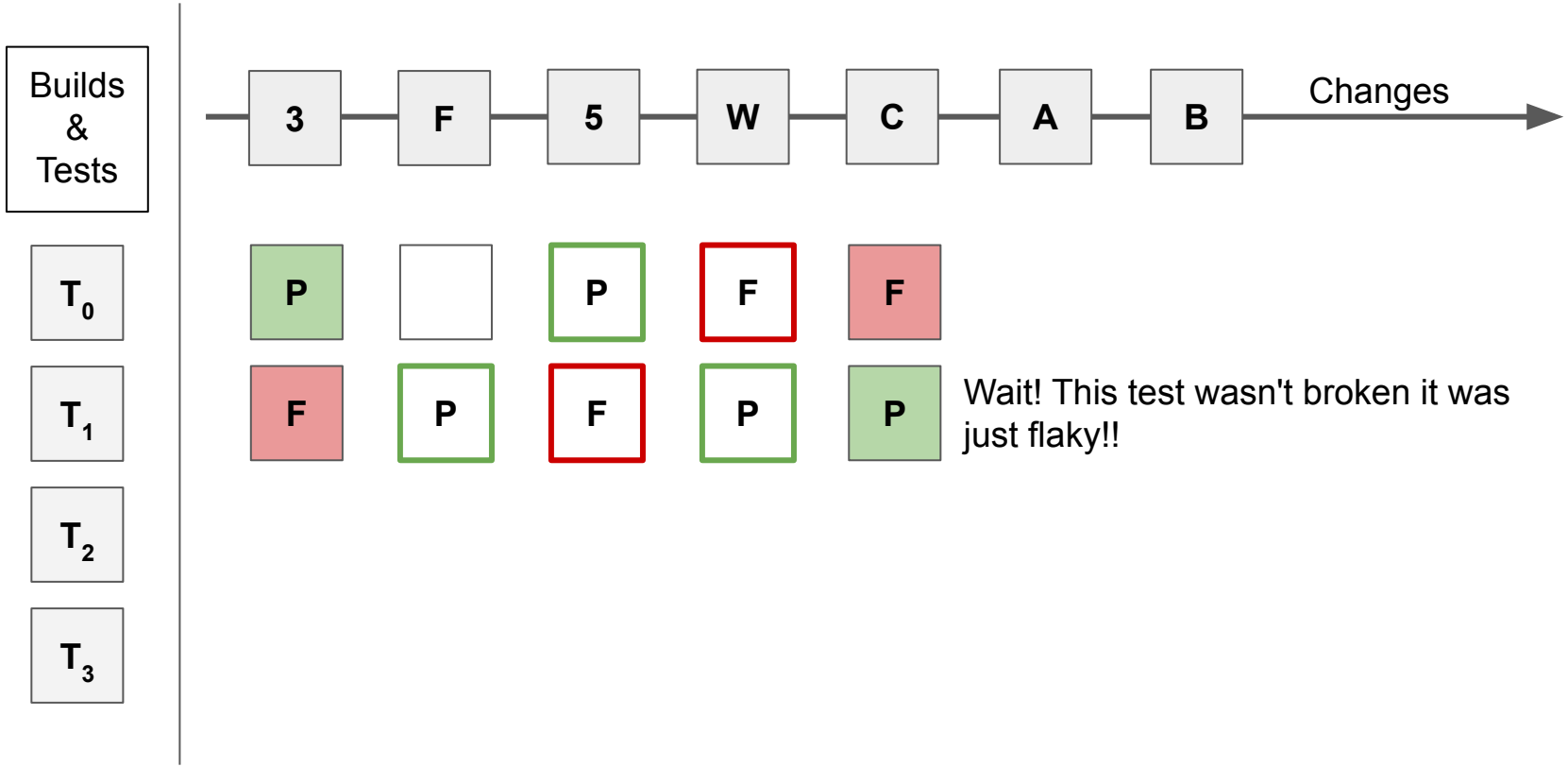
Throttling test execution: requires culprit finding



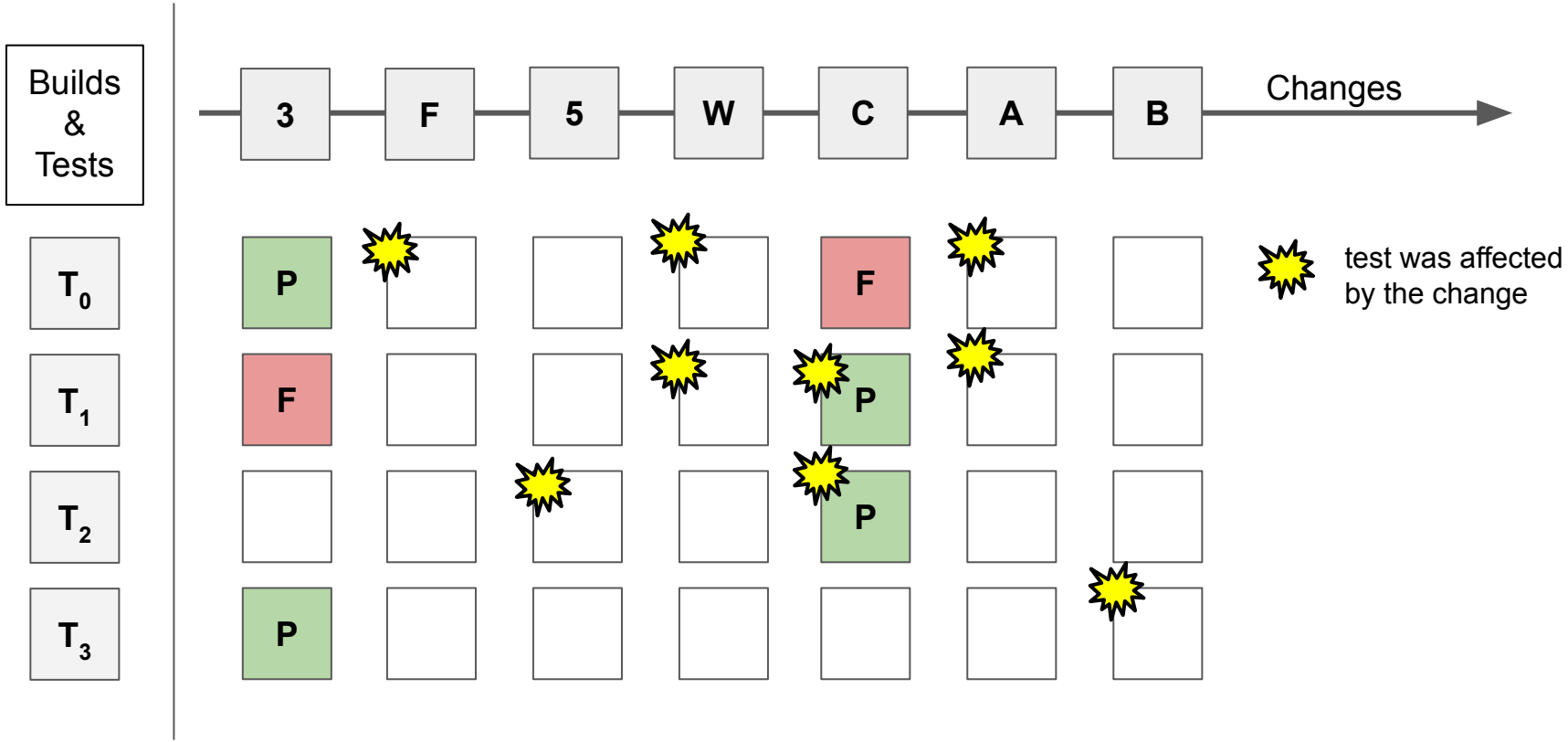
In practice: parallel binary search is often used



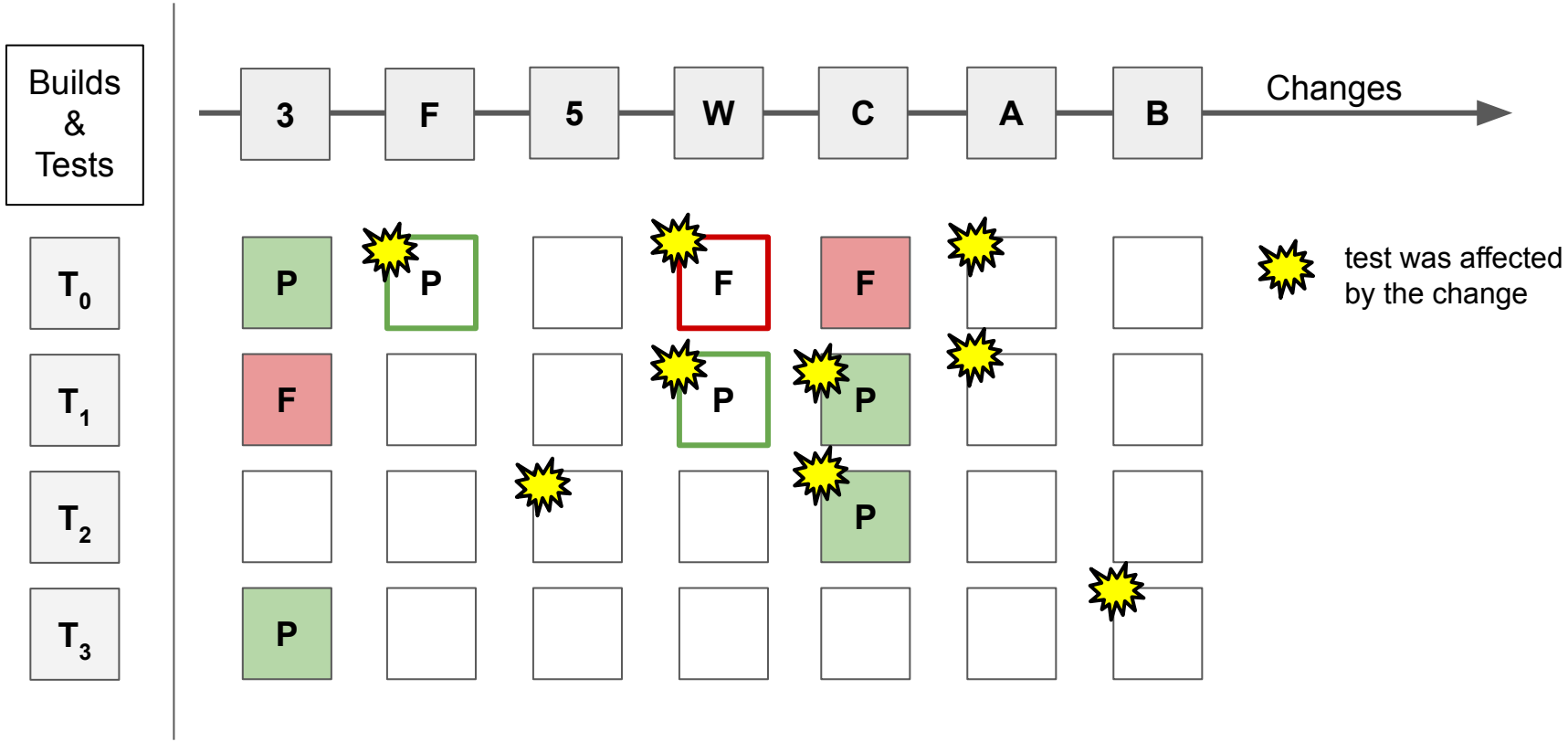
But, watch out for flaky tests!



Combining dependency based selection and throttling



Dependency information speeds up culprit finding



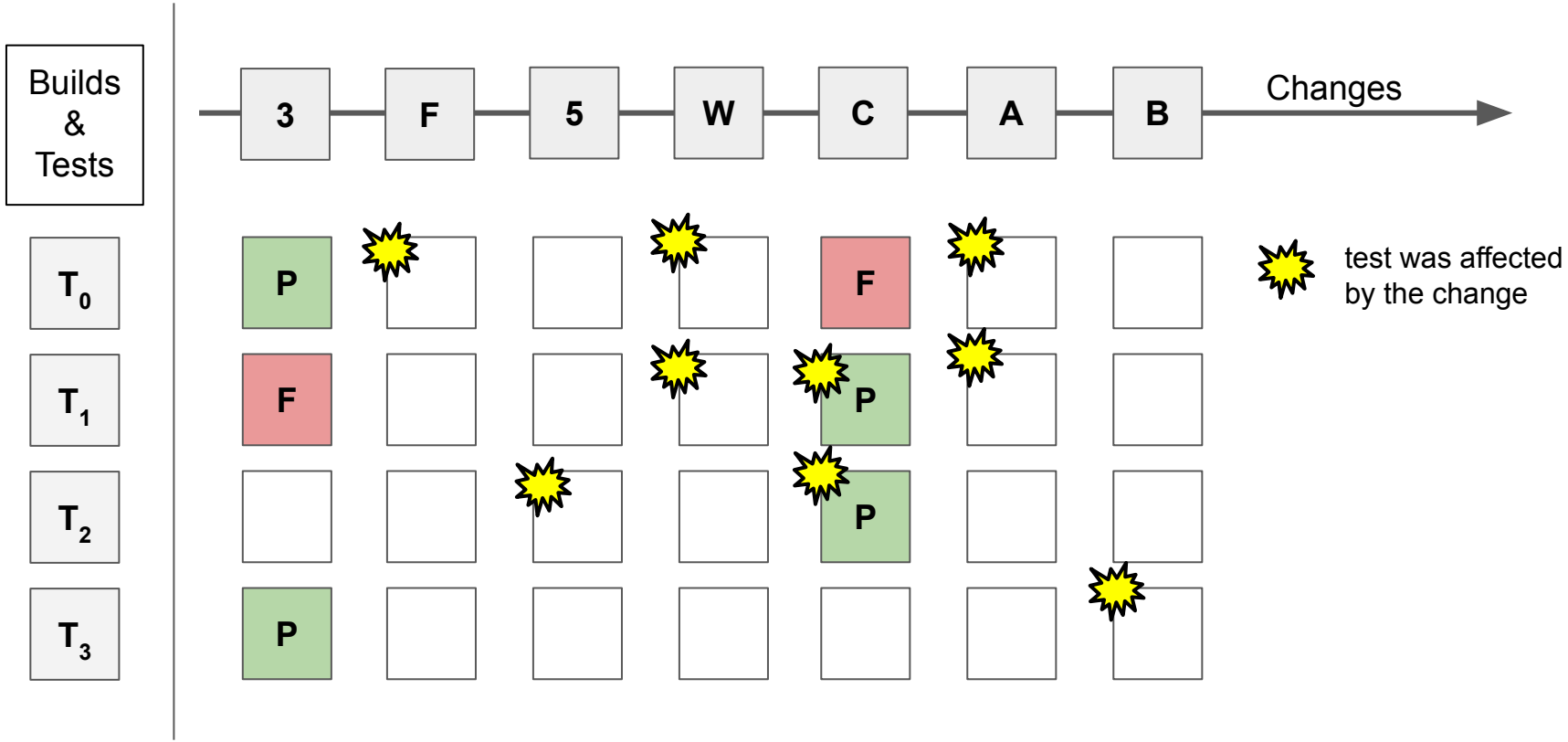
Predicting which tests are most likely to fail.

Academic work includes a **broad** categories of techniques from precise static analysis to coarse grained heuristics + machine learning. Industrial implementations tend to use heuristics as analysis based approaches is challenging. **Must account for flakiness.**

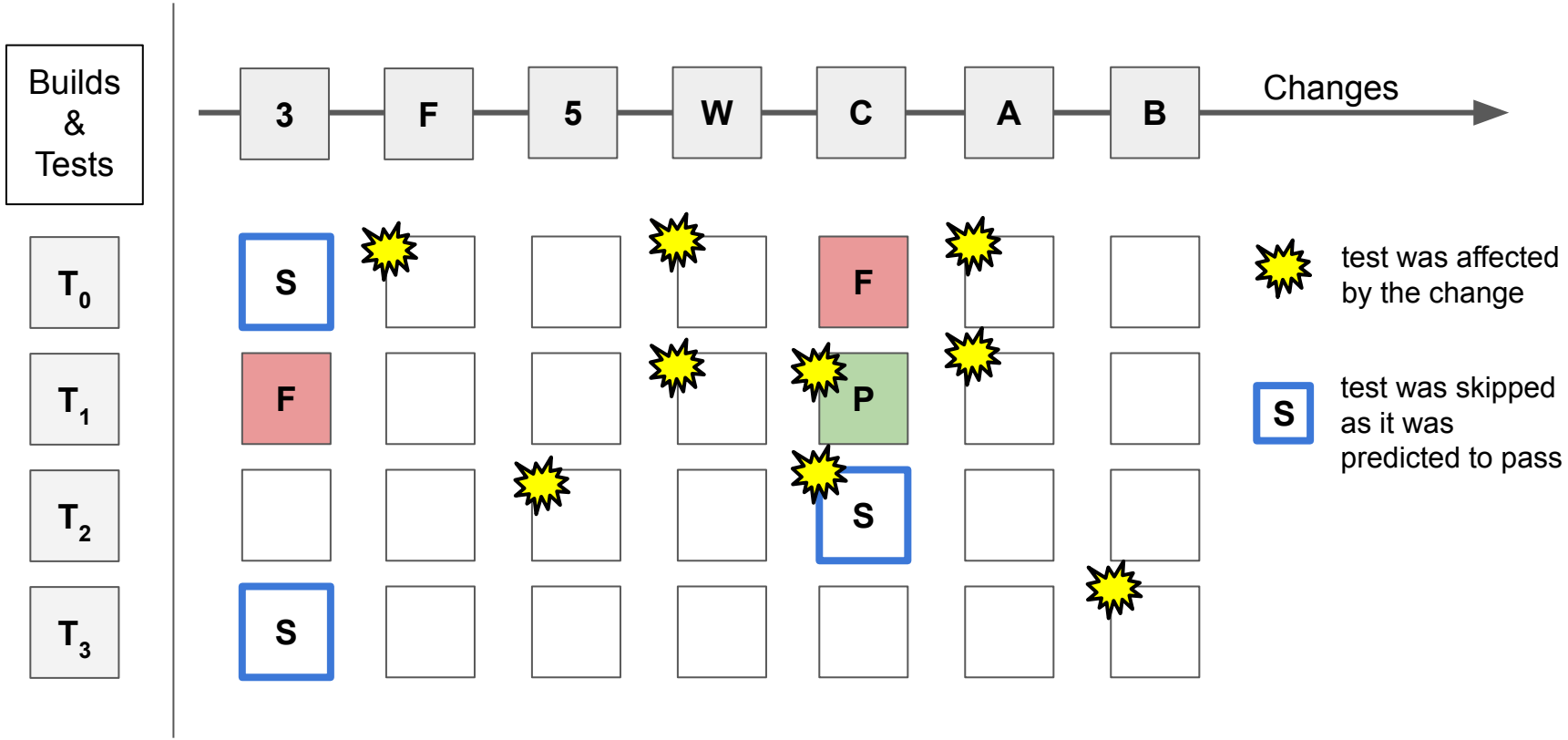
M. Machalica, A. Samylkin, M. Porth, and S. Chandra, "**Predictive Test Selection**," International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP, 2019, doi: https://dx.doi.org/10.1109/ICSE-SEIP.2019.00018. **[Facebook]**

C. Leong, A. Singh, M. Papadakis, Y. Le Traon, and J. Micco, "**Assessing Transition-Based Test Selection Algorithms at Google**," International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2019. https://dx.doi.org/10.1109/ICSE-SEIP.2019.00019. **[Google]**

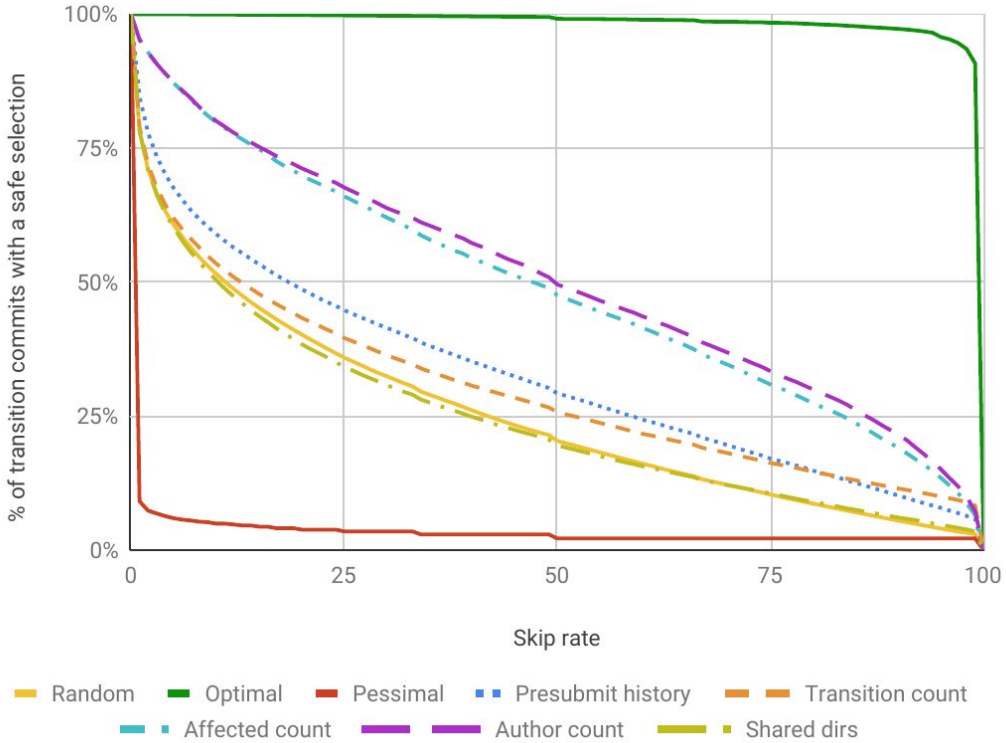
Combining dependency based selection and throttling



Adding in Predictive Selection



Predicting which tests are most likely to fail.



C. Leong, A. Singh, M. Papadakis, Y. Le Traon, and J. Micco, "**Assessing Transition-Based Test Selection Algorithms at Google**," International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2019. https://dx.doi.org/10.1109/ICSE-SEIP.2019.00019. **[Google]**

Manage the flaky tests

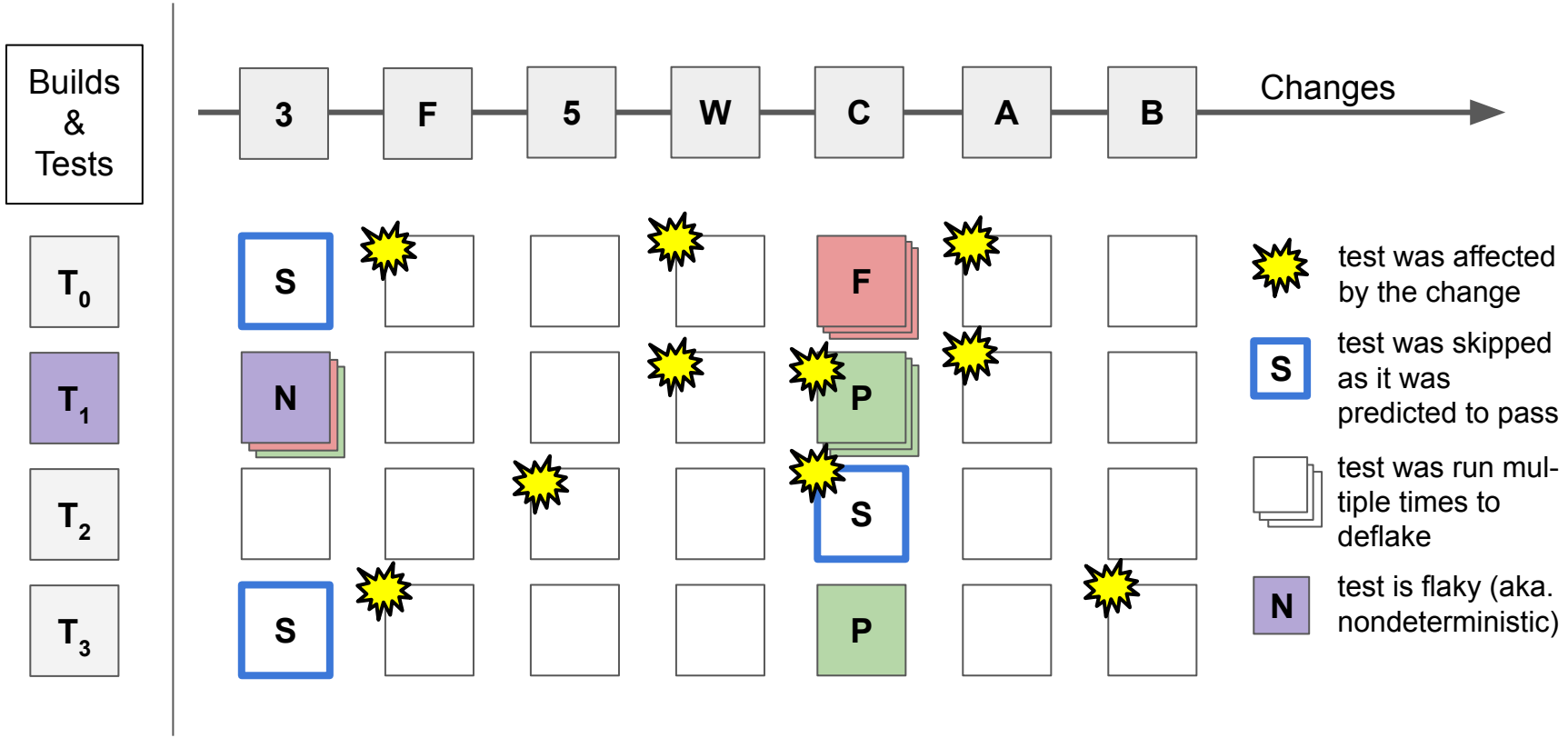
Flaky tests are tests with non-deterministic outcomes. These **must** be managed at every stage in a CI system. They should be automatically identified and triaged, potentially excluded from release gating, and surfaced to developers with tooling support to identify the root causes of the nondeterminicity.

C. Leong, A. Singh, M. Papadakis, Y. Le Traon, and J. Micco, "**Assessing Transition-Based Test Selection Algorithms at Google**," International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2019. https://dx.doi.org/10.1109/ICSE-SEIP.2019.00019.

W. Lam, K. Muslu, H. Sajnani, and S. Thummalapenta, "**A Study on the Lifecycle of Flaky Tests**," International Conference on Software Engineering ICSE, 2020. Available: https://www.microsoft.com/en-us/research/publication/a-study-on-the-lifecycle-of-flaky-tests/.

J. Bell et al., **DeFlaker: Automatically Detecting Flaky Tests**. International Conference on Software Engineering ICSE, 2018. https://dx.doi.org/10.1145/3180155.3180164.

Detecting flakes with re-runs of failures and known flaky tests



Manage the flaky tests

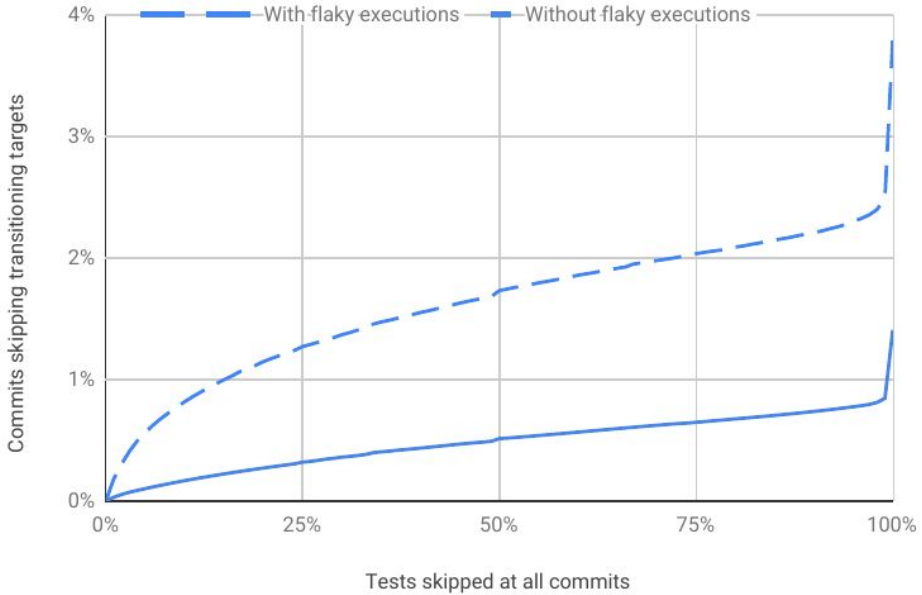


Fig. 1. Performance of the transition count algorithm (Section VI-B), on data with and without flaky executions.

> Flaky tests are a significant source of transitions at Google: in our data over 80% of observed transitions were caused by confirmed flaky results.

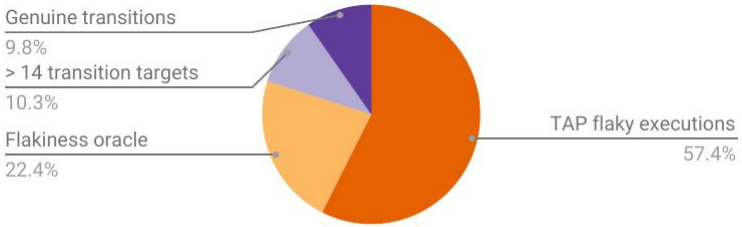


Fig. 2. The sources of transitions in raw test data.

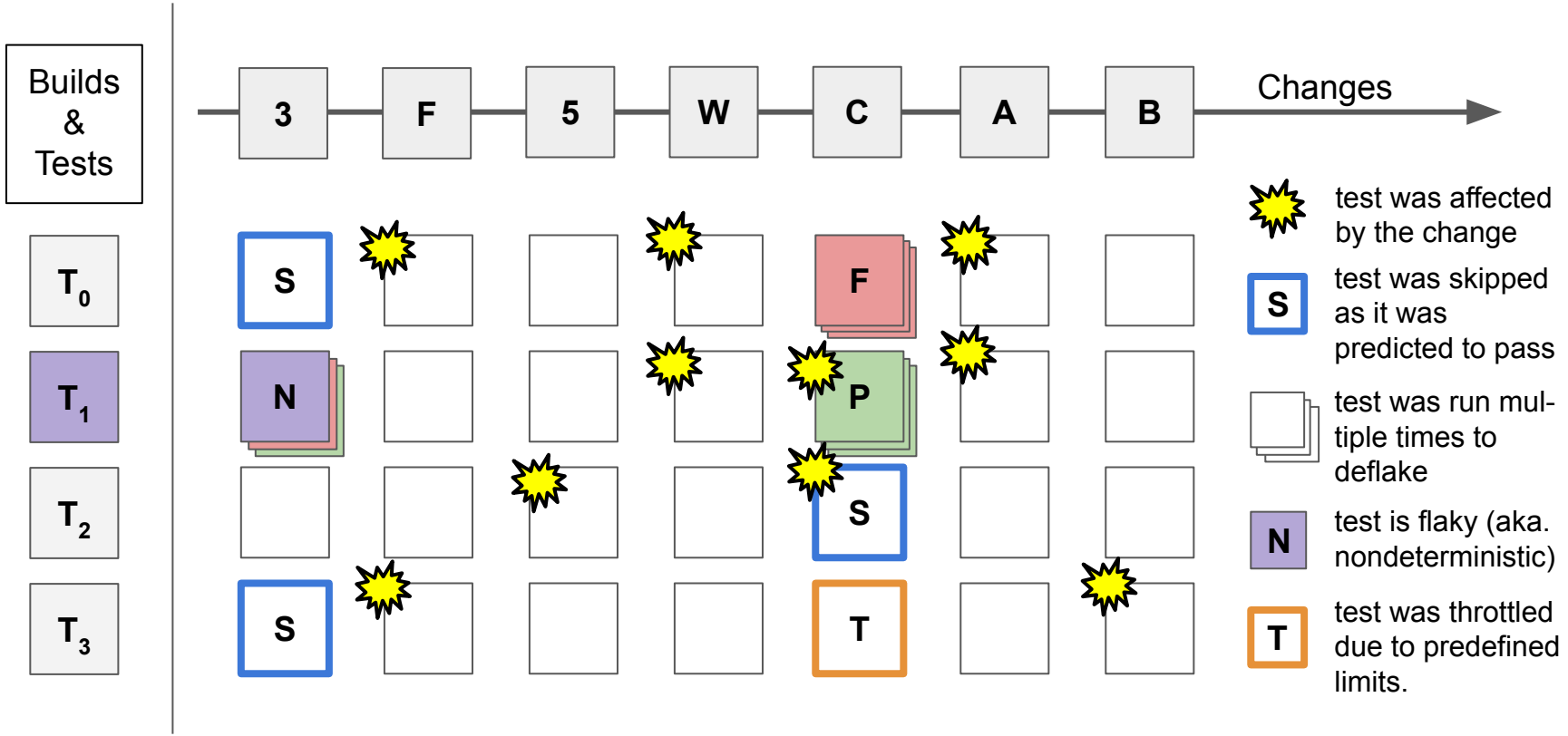
C. Leong, A. Singh, M. Papadakis, Y. Le Traon, and J. Micco, "**Assessing Transition-Based Test Selection Algorithms at Google**," International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2019. https://dx.doi.org/10.1109/ICSE-SEIP.2019.00019. **[Google]**

Managing demand through economics

It is important to ensure the end user is aware of the costs of their usage of CI. In large integrated organizations using a monorepo this can be challenging. One approach is to force each product to internally pay for their expected usage and then throttle them based on how many resources they actually bought. This will encourage developers to optimize their tests.

T. Bach, R. Pannemans, and S. Schwedes, “**Effects of an economic approach for test case selection and reduction for a large industrial project**,” International Conference on Software Testing, Verification and Validation Workshops ICSTW 2018. 2018. https://dx.doi.org/10.1109/ICSTW.2018.00076.

Predefined limits throttle the execution of certain tests



Google

Understanding what CI model works best

There are many different CI models. Some organization enforce correctness by serializing the merge and test operations to bad commits are not allowed into be integrated. Others allow a small percentage of "collisions" and then apply culprit finding and automatic rollback. There are many open questions, here are three:

1. At what stages (pre-merge, post-merge, release) do the various options for test selection work most effectively?

2. What merge-gating techniques are most cost effective?

3. Are there ways to estimate the economic cost of test failure and prioritize tests with higher costs — not just tests which are predicted to be failure prone?

Google

Complexity

As CI systems grow in features and smart capabilities the complexity of the system needs to be actively managed. Some techniques the system used a few years ago may be moderately effective but be too costly in either resources or CI developer time to scale with the growing demands of your organization.

Trade-offs between resources spent on testing and CI infrastructure have to be made against implementation, scaling, and maintenance costs of better and smarter algorithms.

Google

Questions?

mail me: tadh@google.com
visit me: hackthology.com

Citations

Pooja Gupta, Mark Ivey and John Penix. **Testing at the speed and scale of Google**. 2011. http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html

John Micco. **Tools for Continuous Integration at Google Scale.** Google Tech Talk. Google NYC. June 19 2012. https://youtu.be/KH2_sB1A6lA [**Google**]

John Micco and Developer Infrastructure. "**Continuous integration at google scale.**" Eclipse Con 2016. [slides] [Google]

A. Memon et al., "**Taming Google-scale continuous testing**," International Conference on Software Engineering: Software Engineering in Practice Track ICSE-SEIP, 2017. https://dx.doi.org/10.1109/ICSE-SEIP.2017.16.

J. Bell et al., **DeFlaker: Automatically Detecting Flaky Tests**. International Conference on Software Engineering ICSE, 2018. https://dx.doi.org/10.1145/3180155.3180164.