# CS 4530 Software Engineering

## Lesson 8.1: Continuous Integration

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand
Khoury College of Computer Sciences

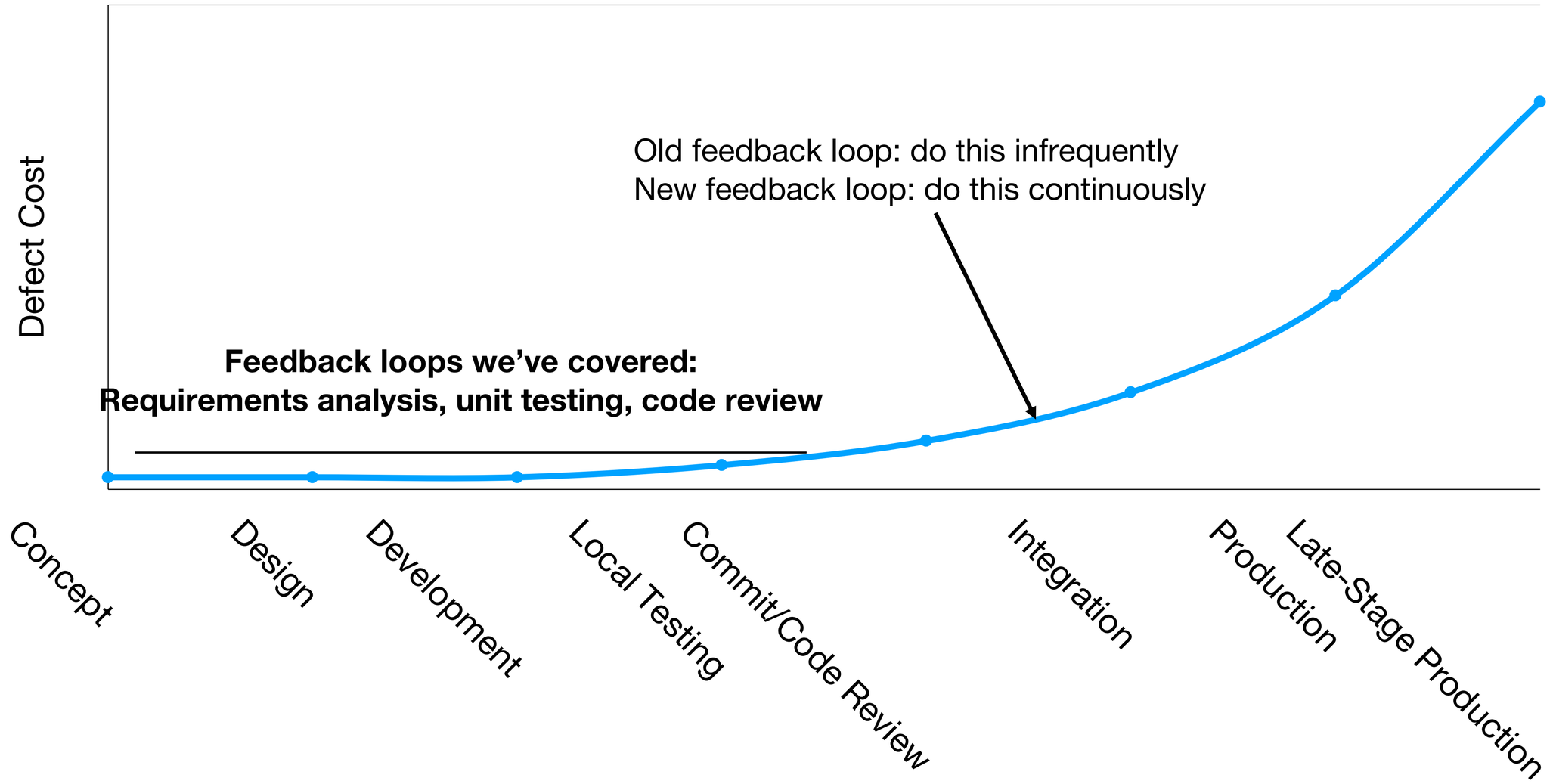
# Learning Objectives for this Lesson

**By the end of this lesson, you should be able to…**

- Describe how continuous integration helps to catch errors sooner in the software lifecycle
- Use continuous integration systems to automate testing in real software projects

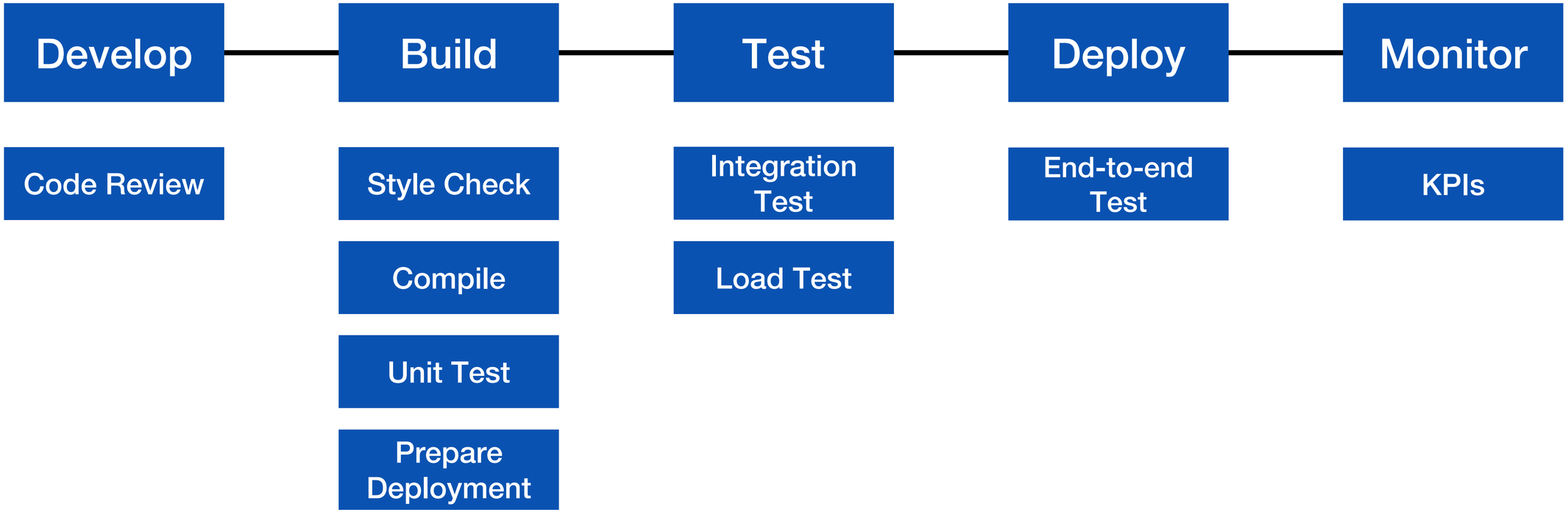
# Cost to Fix a Defect Over Time

## Rough estimate



# Continuous Development

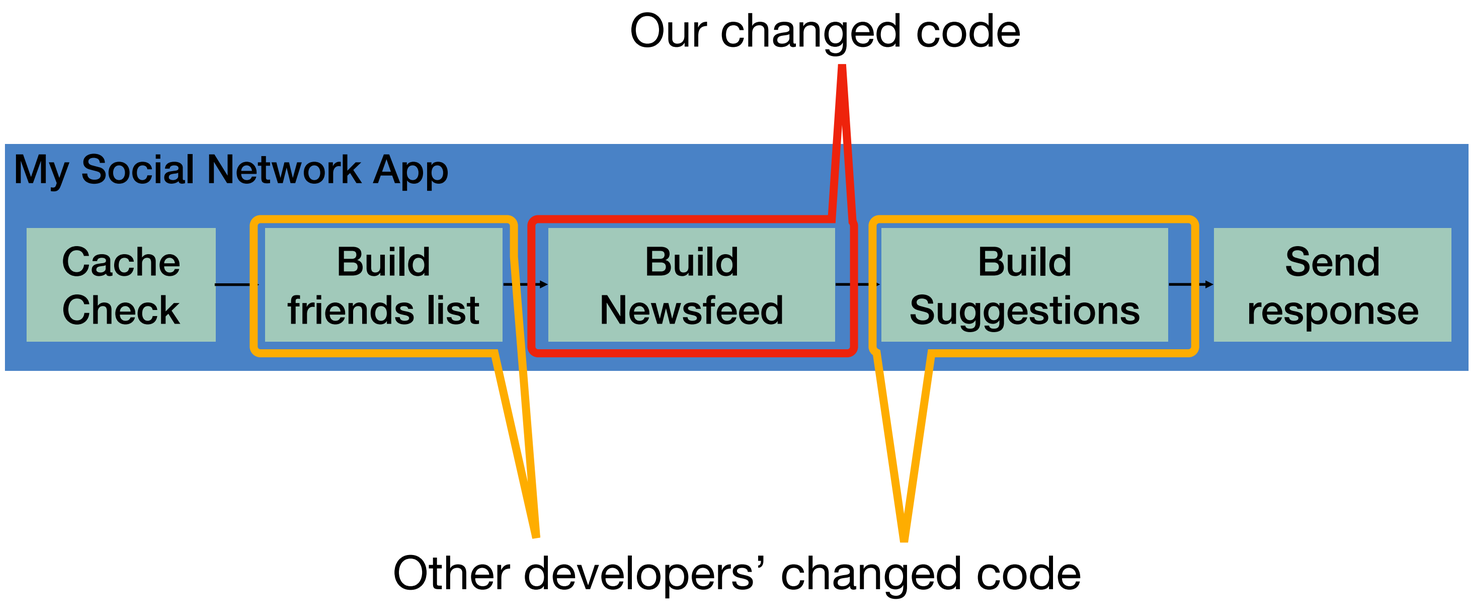
## Improving quality & velocity with frequent, fast feedback loops



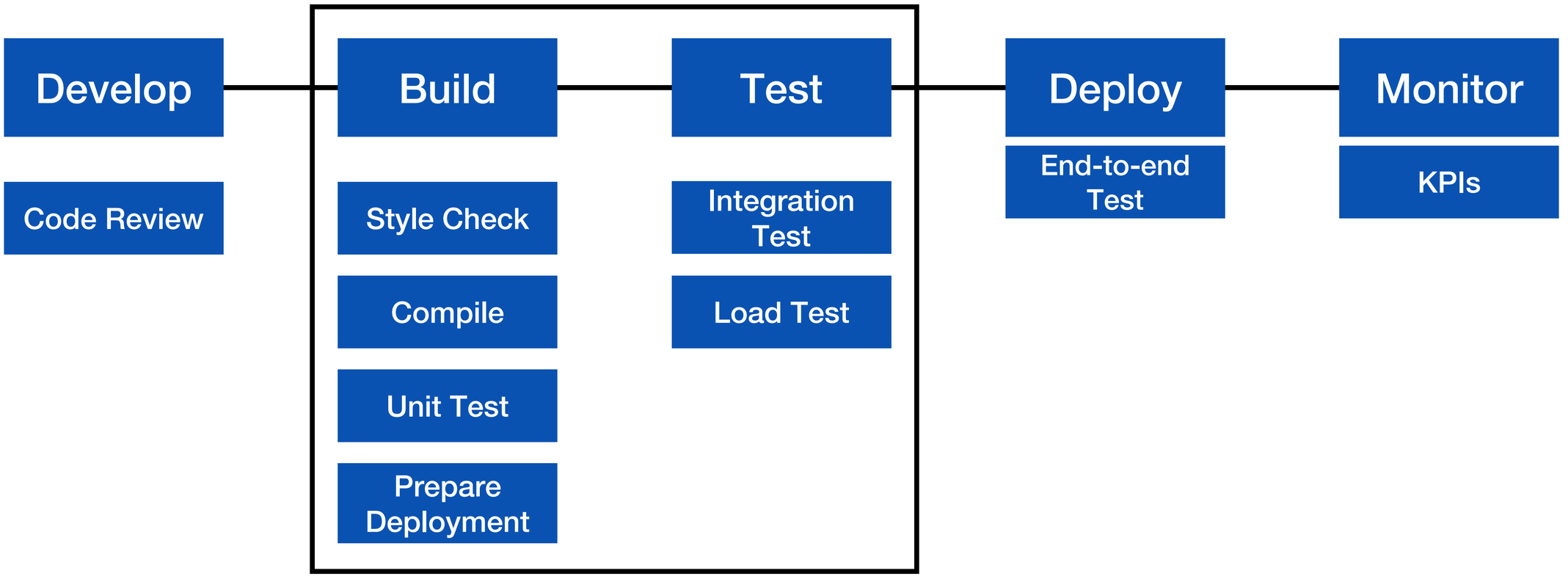
# Continuous Integration

## Motivation

- Our systems involve many components, some of which might even be in different version control repositories
- How does a developer get feedback on their (local) change?



# Continuous Integration is a Software Pipeline




**Automate this centrally, provide a central record of results**

# Build Systems

## Automatically compiling code and generating executables

- You've probably used multiple of these:
  - Make, maven, ant, gradle, grunt, sbt
- Why use a build system?
  - Builds should be repeatable
  - Builds should be reproducible
  - Builds should be standard

# Build Systems

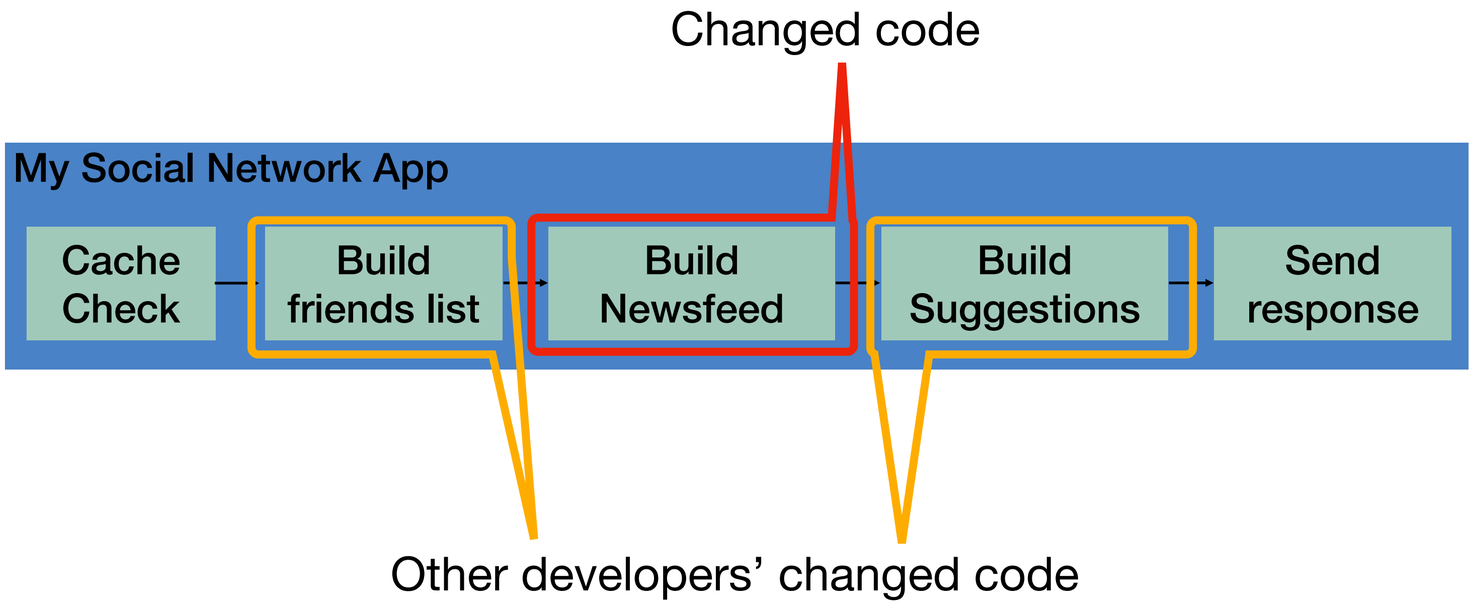
## Not just compilation

- Fetch dependencies and link them (using a package manager like maven, pip or npm)
- Provision & teardown resources for integration testing
- Run tests
- Generate a release archive
- Ideally, do this all in parallel as much as possible

# How do we apply continuous integration?

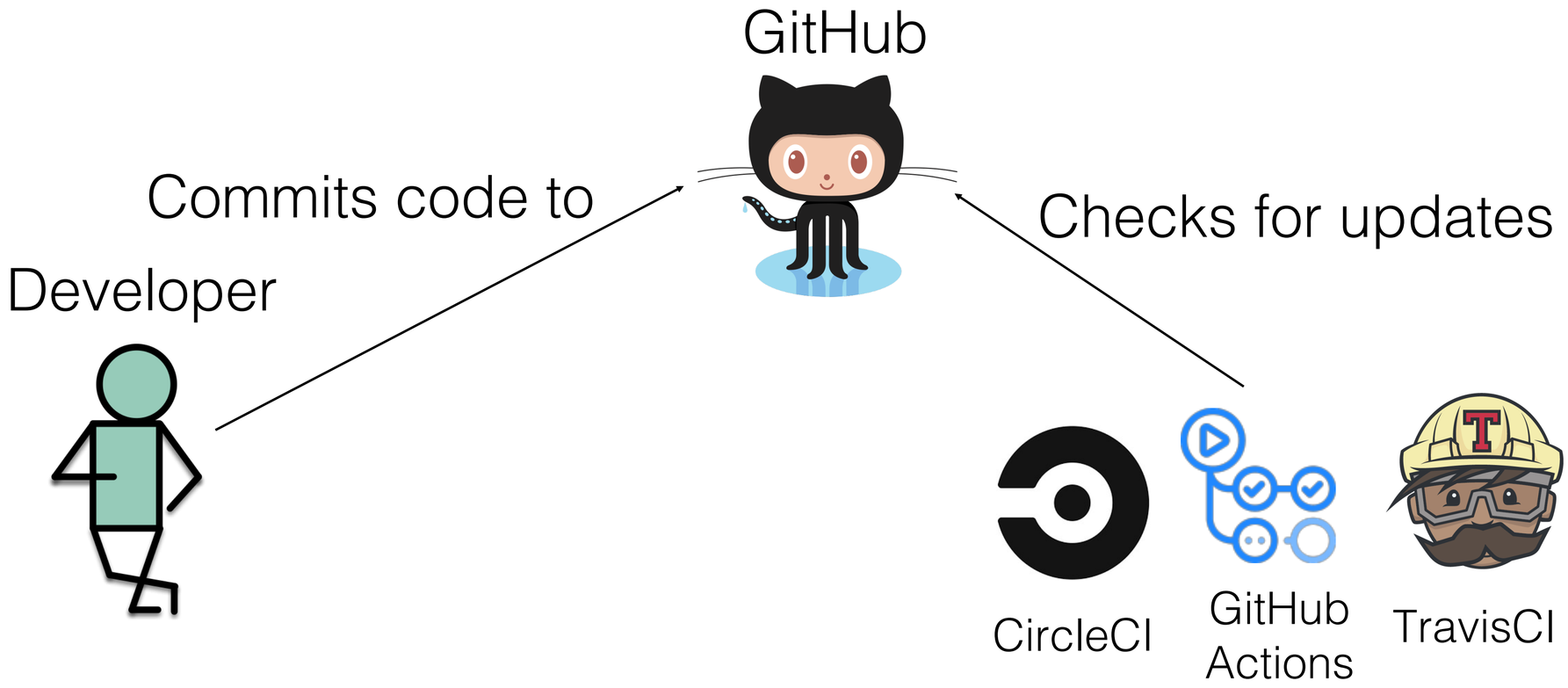
## Testing the right things at the right time

- Do we integrate changes immediately, or do a pre-commit test?
- Which tests do we run when we integrate?
- How do we compose the system under test at each point?



# Continuous Integration in Practice

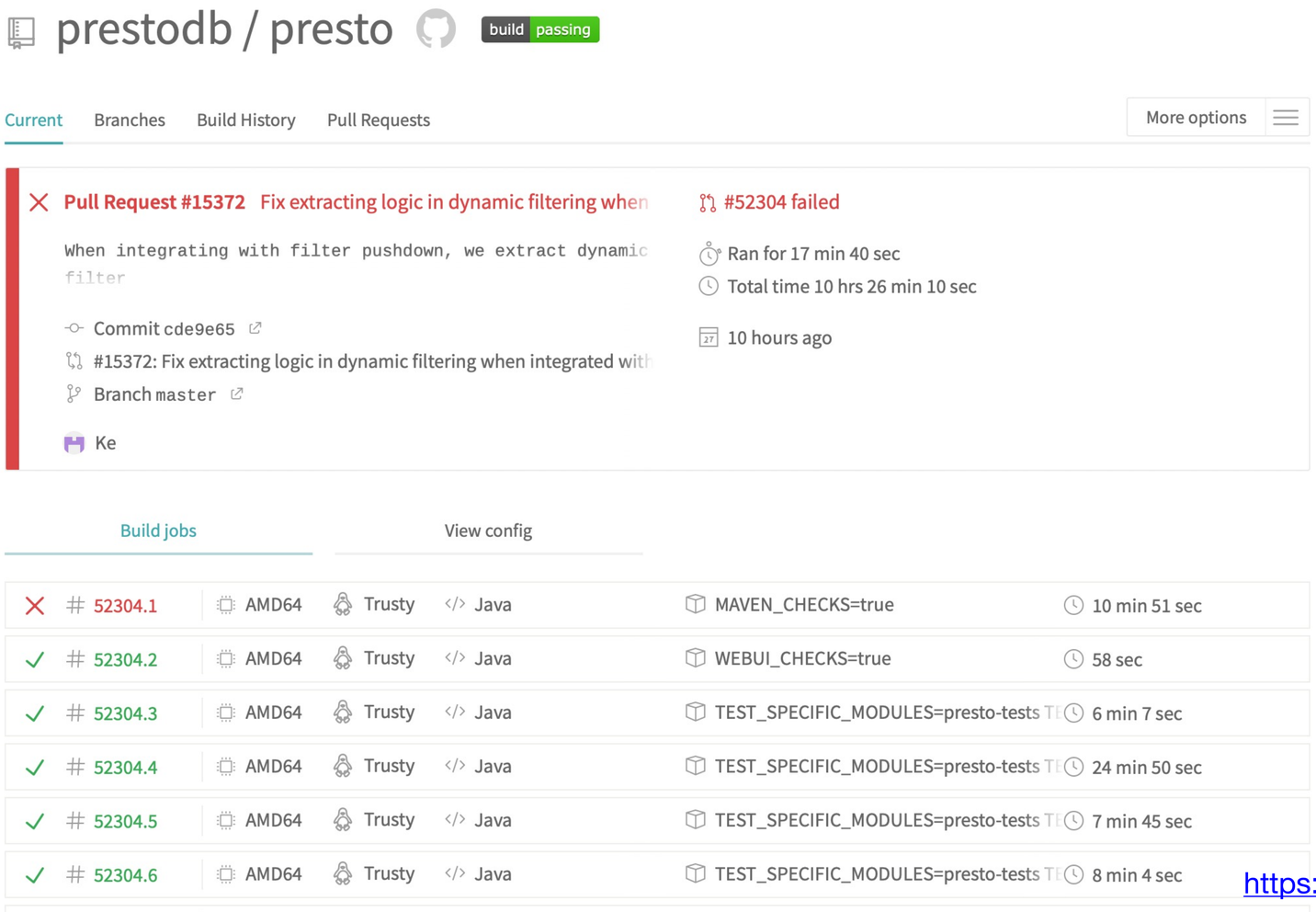
**Small scale, with a service like CircleCI, GitHub Actions or TravisCI**




Runs build for each commit

# Example CI Pipeline

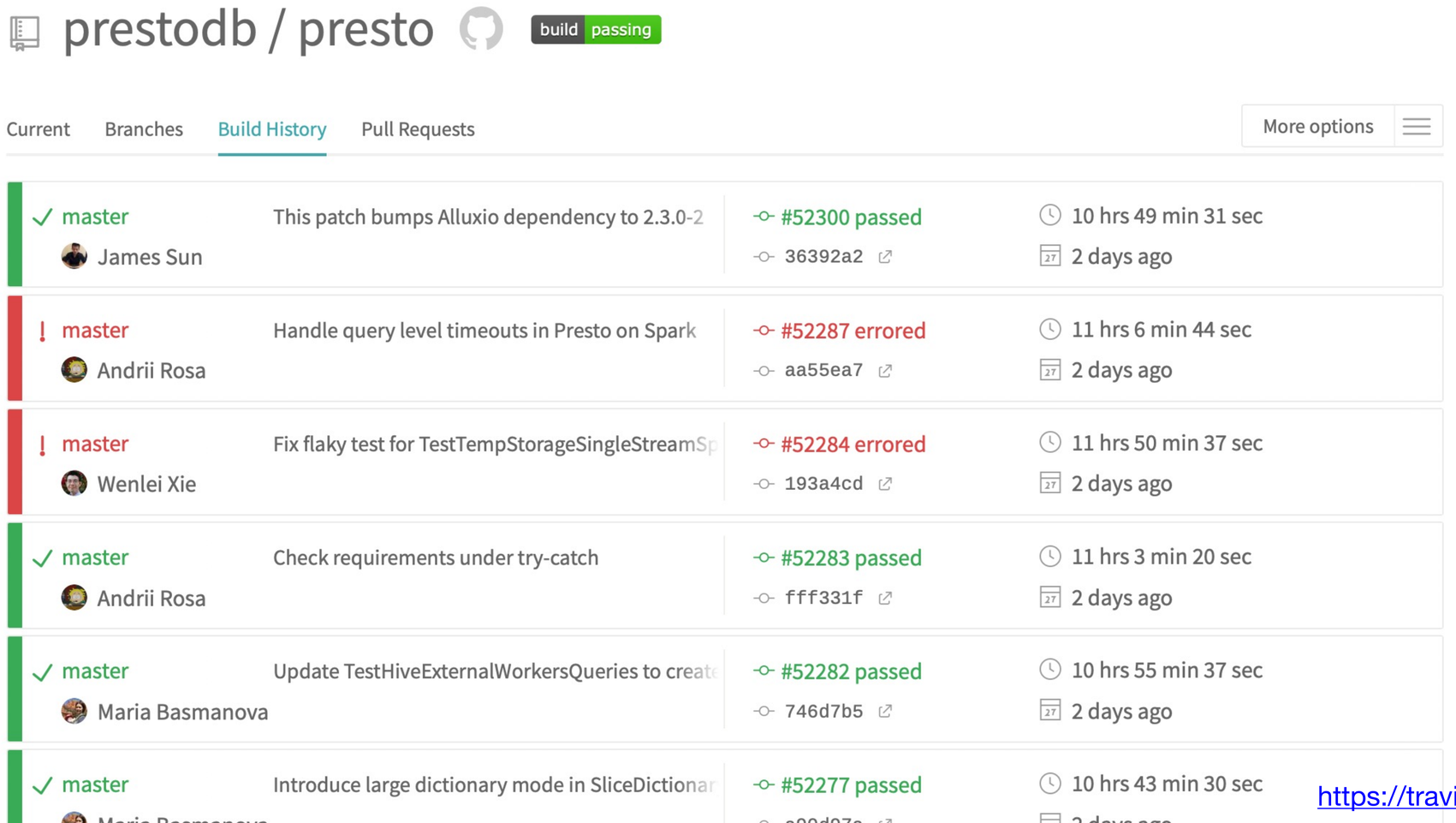
## Open source project: PrestoDB




https://travis-ci.com/github/prestodb/presto

# Example CI Pipeline - TravisCI

## At a glance, see history of build



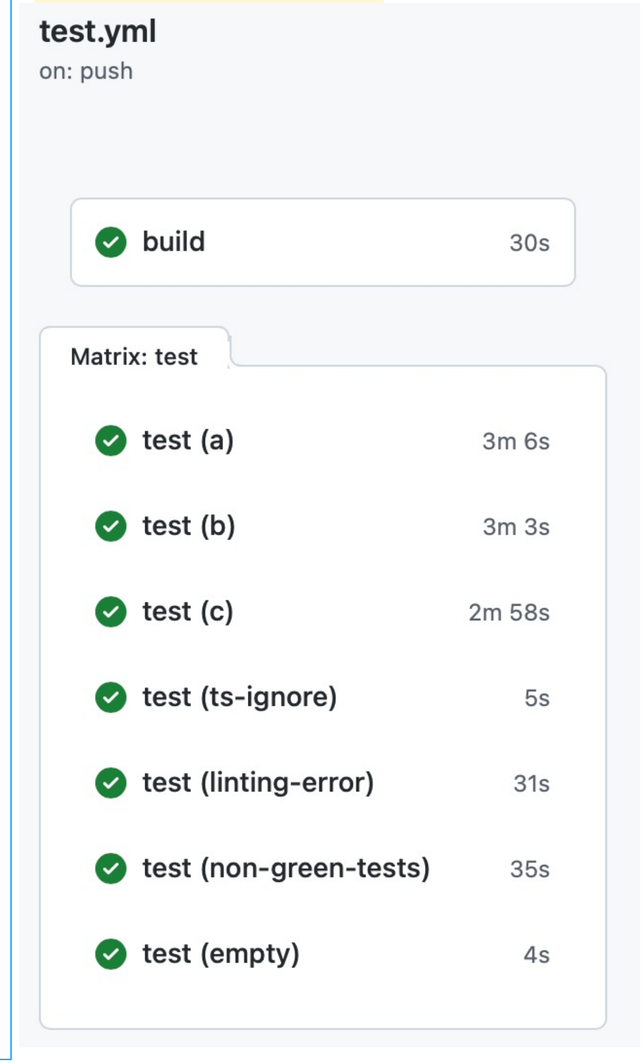

https://travis-ci.com/github/prestodb/presto

# CI In Practice: HW3 Autograder

test.yml (CI workflow file)

```yaml
name: 'Build and Test the Grader'
on: # rebuild any PRs and main branch changes
  pull_request:
  push:
    branches:
      - main
      - 'releases/*'
jobs:
  build:
    runs-on: self-hosted
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '16'
      - run: |
          npm install
  test:
    runs-on: self-hosted
    strategy:
      matrix:
        submission: [a, b, c, ts-ignore, linting-error, non-green-tests, empty]
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '16'
      - uses: ./
        with:
          submission-directory: solutions/${{ matrix.submission }}
```

GitHub Actions Results

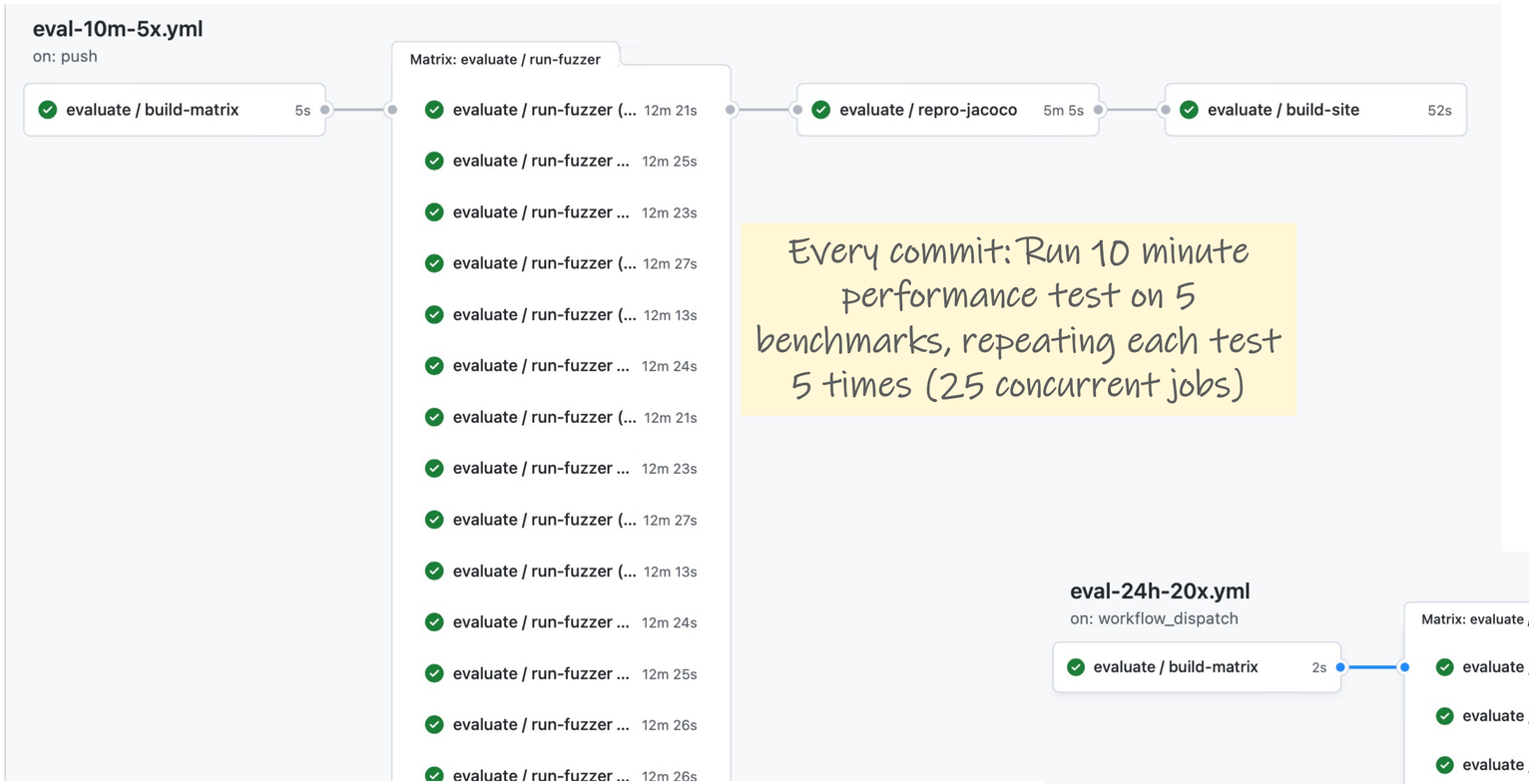


# CI Pipelines Automate Performance Testing

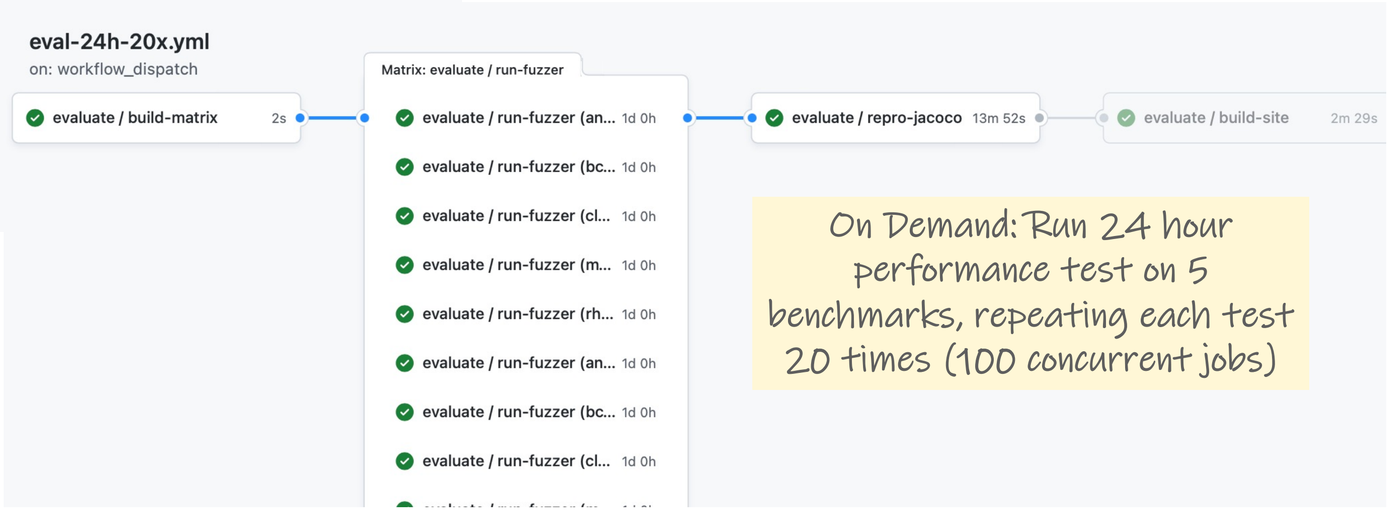
## Example: Developing a Fuzzer

- “Fuzzers” are automated testing systems that aim to automatically generate inputs to programs that cover code and reveal bugs
- Fuzzers are non-deterministic: to evaluate with confidence, need repeated, long-running trials
- Evaluating fuzzers is time consuming, determining which changes impact performance is confusing

# CI Pipelines Automate Performance Testing




Every commit: Run 10 minute performance test on 5 benchmarks, repeating each test 5 times (25 concurrent jobs)



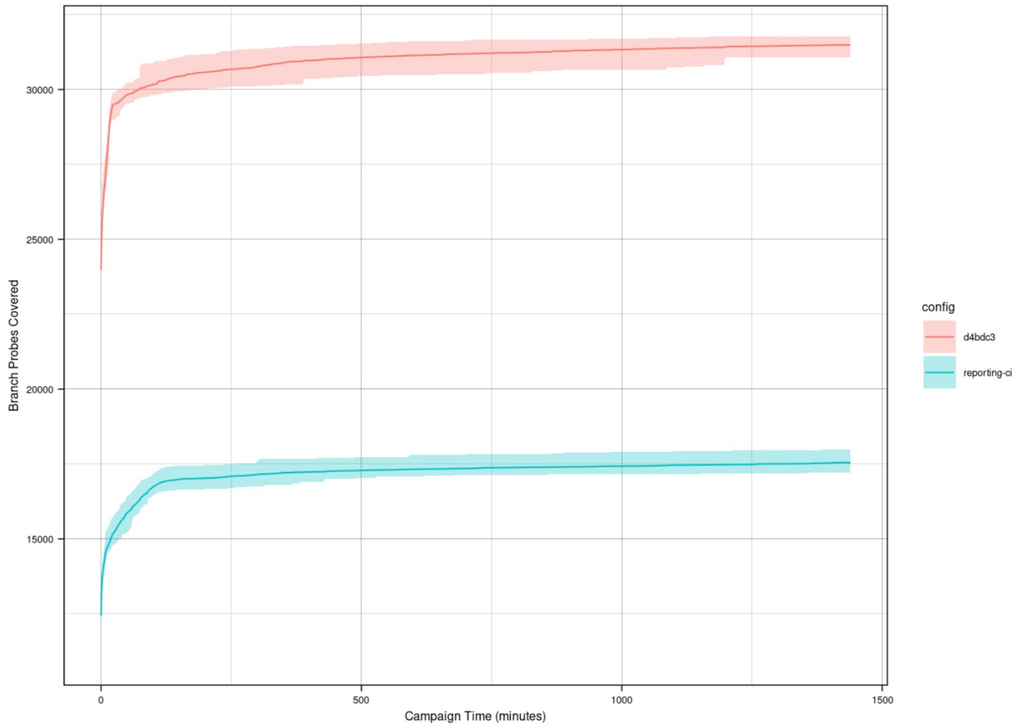

On Demand: Run 24 hour performance test on 5 benchmarks, repeating each test 20 times (100 concurrent jobs)

https://github.com/neu-se/CONFETTI/actions

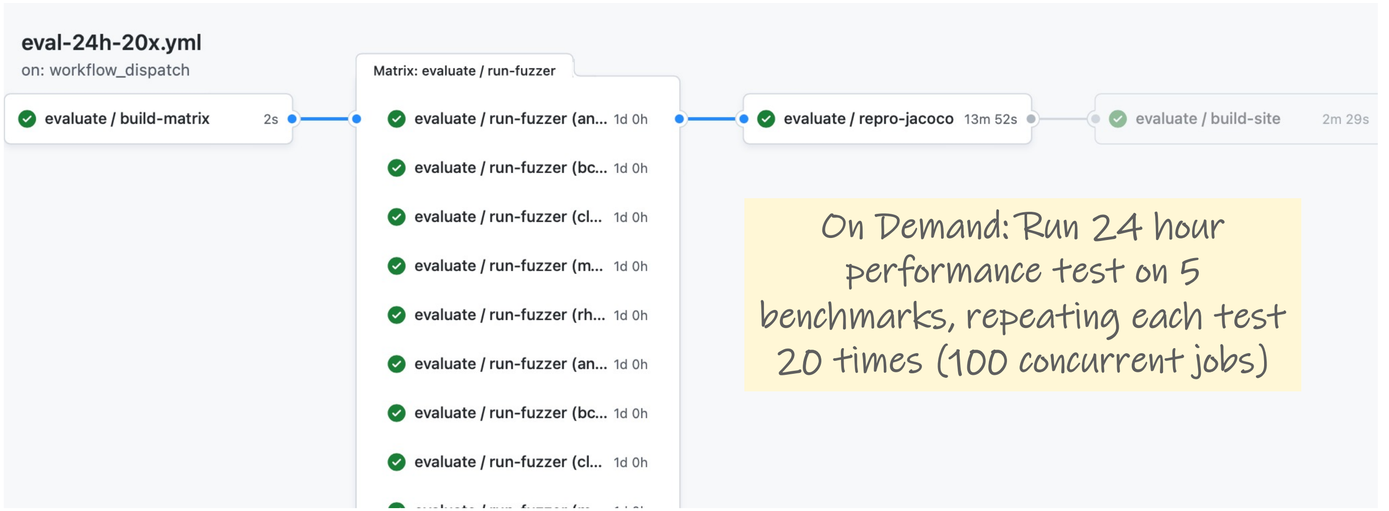
# CI Pipelines Automate Performance Testing

closure

Branch Probes Over Time




Download this graph as PDF




https://github.com/neu-se/CONFETTI/actions

# Continuous Integration in Practice

## Large scale example: Google TAP

- >50,000 unique changes per-day, > 4 billion test cases per-day
- Pre-submit optimization: run fast tests for each individual change (before code review). If fast tests pass, allow the merge to continue
- Then: run all affected tests; “build cop” monitors and acts immediately to roll-back or fix
- Build cop monitors integration test runs
- Average wait time to submit a change: 11 minutes

“Software Engineering at Google: Lessons Learned from Programming Over Time,” Wright, Winters and Manshreck, 2020 (O’Reilly)

# Continuous Integration

## Summary and next steps

- CI helps catch errors sooner in the software lifecycle by performing integration and end-to-end tests sooner
- CI can be applied in small-scale projects by running complete test suites for each commit, or in larger projects by running pre-commit tests per-commit and complete integrations regularly
- CI assumes the ability to automatically provision infrastructure on which to run those integration tests [next lesson]