

CS 4530 Software Engineering

Lesson 8.1: Continuous Integration

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand

Khoury College of Computer Sciences

© 2022 released under [CC BY-SA](#)

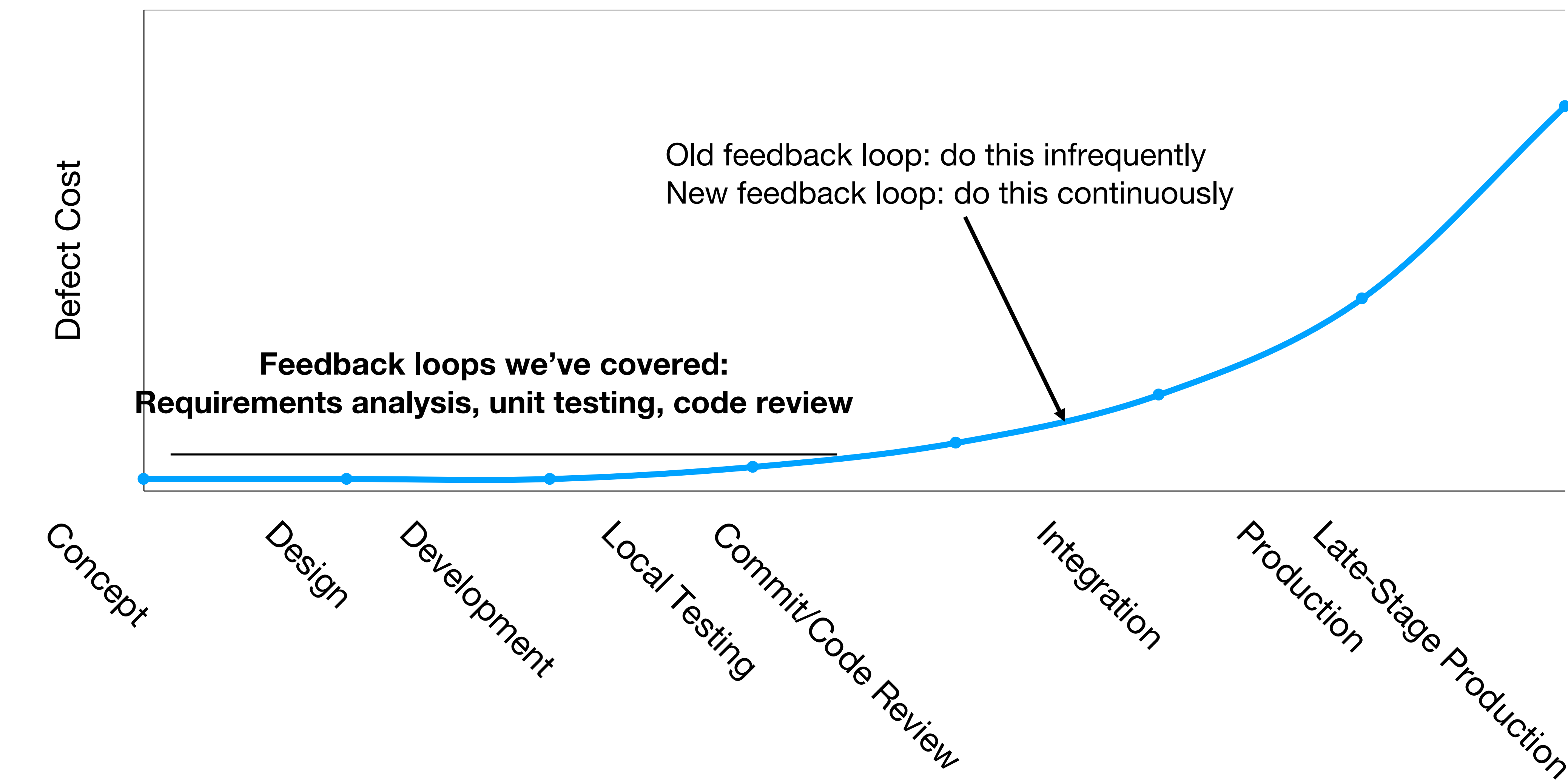
Learning Objectives for this Lesson

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

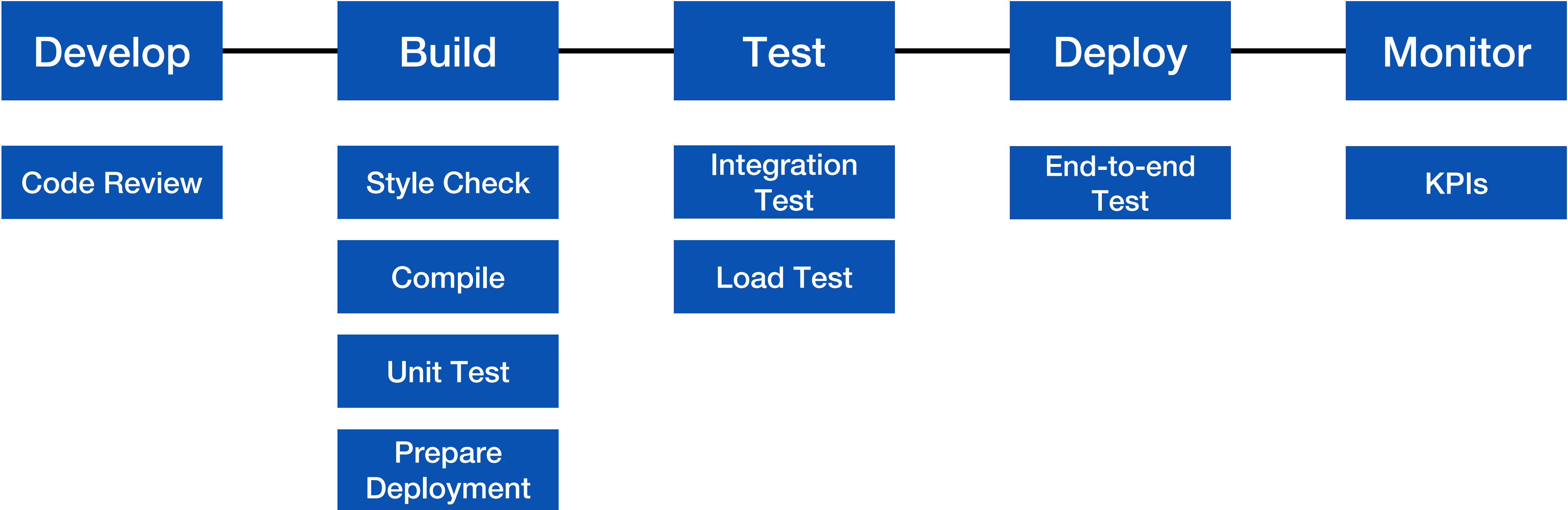
Cost to Fix a Defect Over Time

Rough estimate



Continuous Development

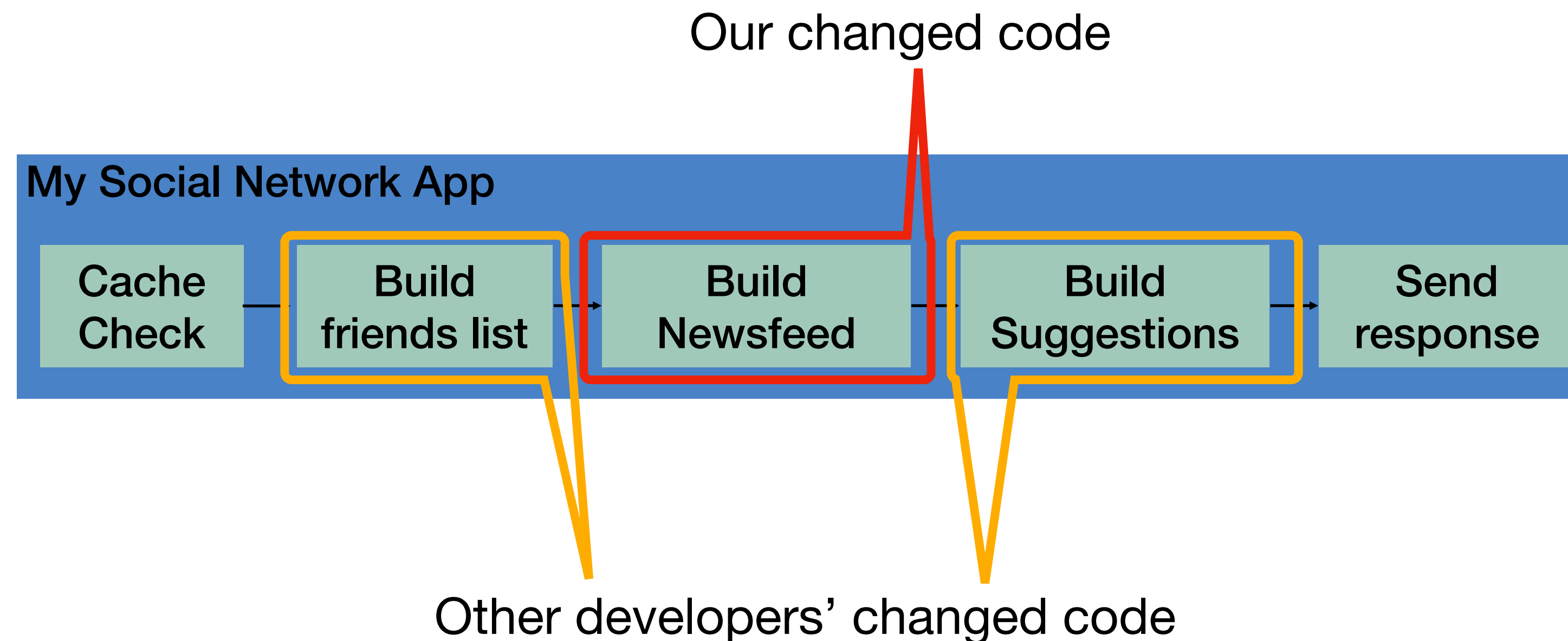
Improving quality & velocity with frequent, fast feedback loops



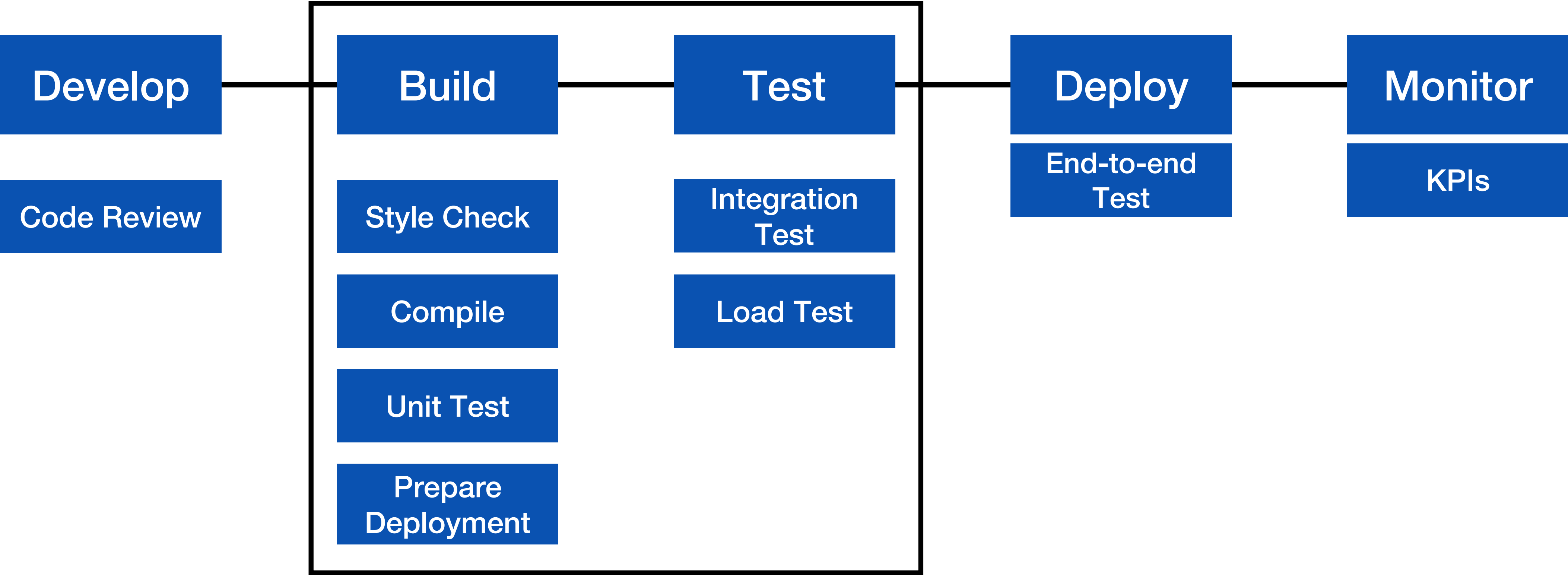
Continuous Integration

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

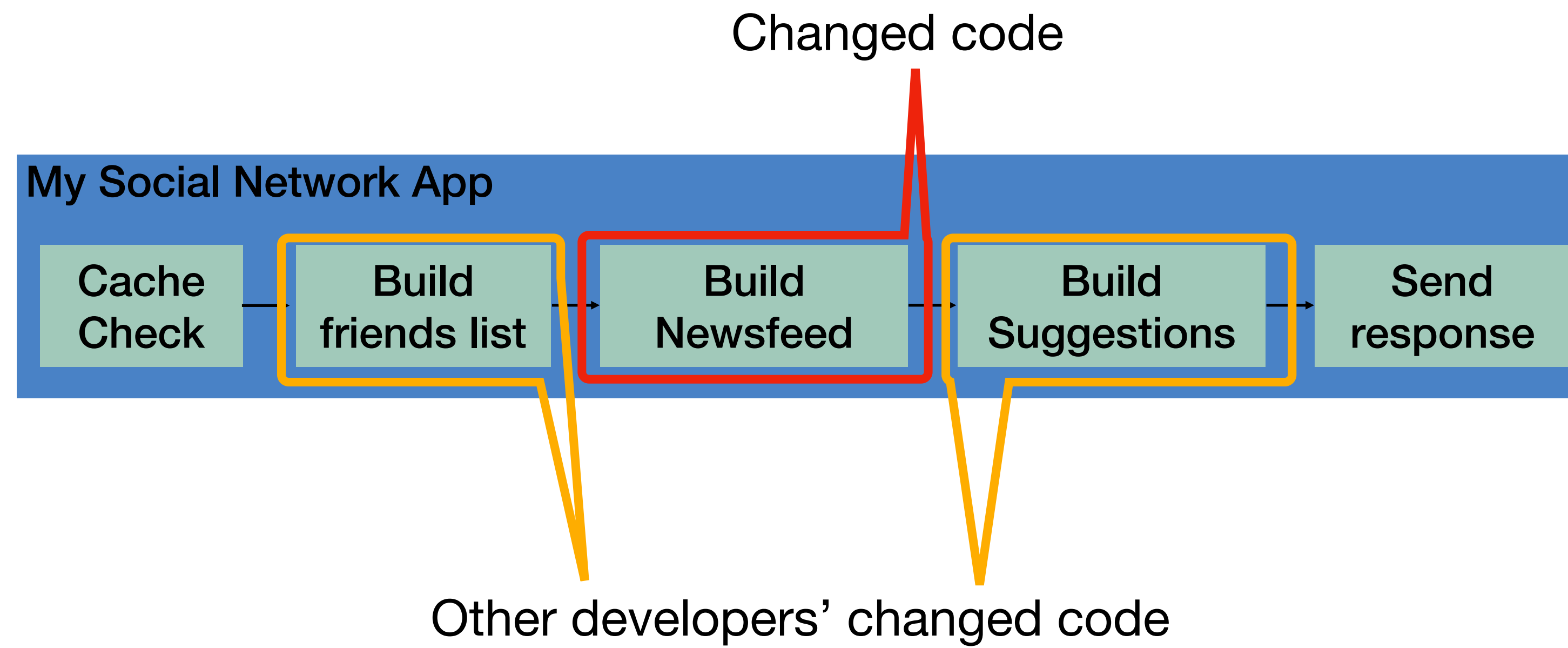
Not just compilation

- 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?

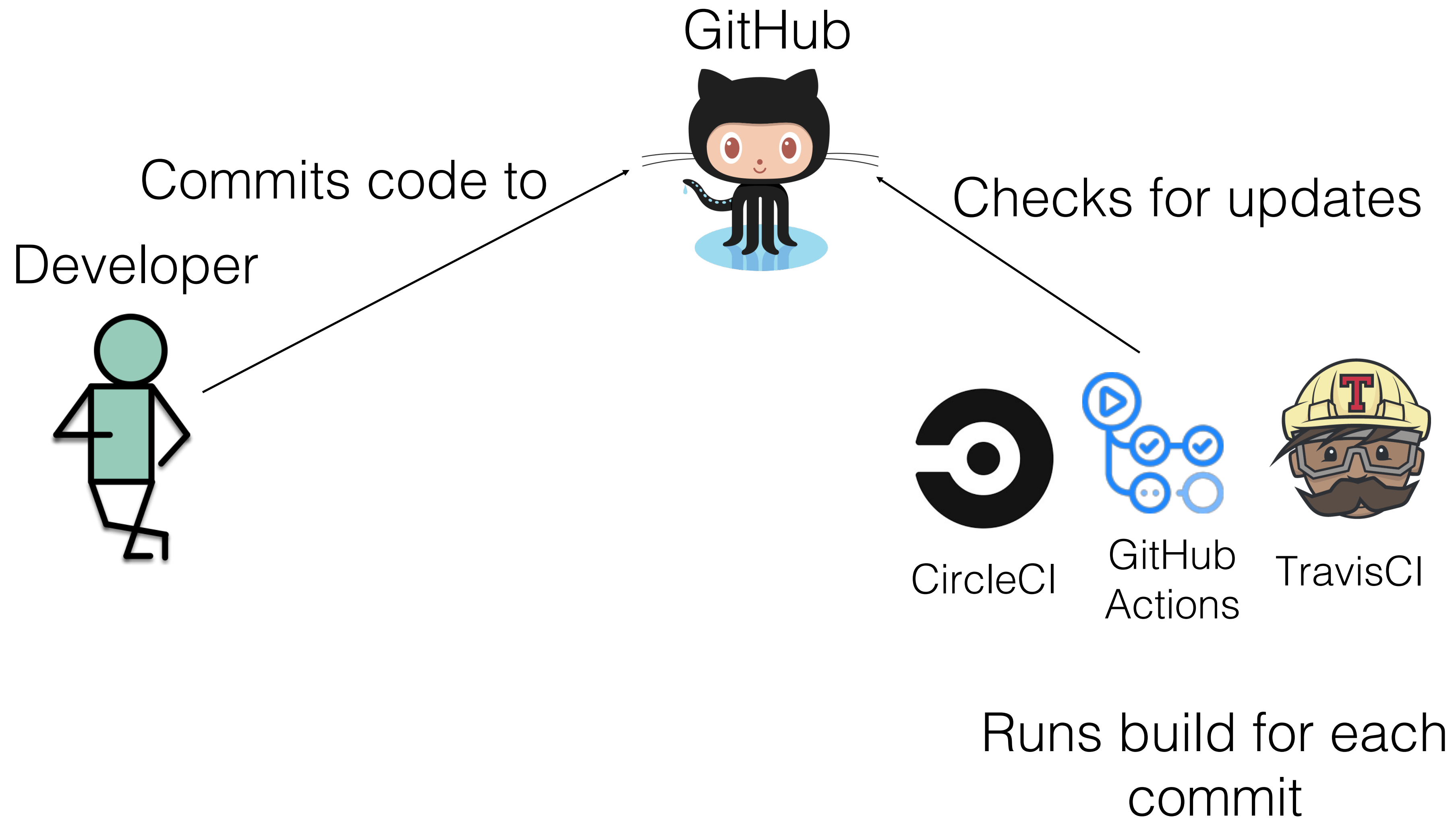
Testing the right things at the right time

- 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



Example CI Pipeline

Open source project: PrestoDB

prestodb / presto

build

passing

Current

Branches

Build History

Pull Requests

More options

✖

Pull Request #15372

Fix extracting logic in dynamic filtering when

When integrating with filter pushdown, we extract dynamic filter

Commit cde9e65

#15372: Fix extracting logic in dynamic filtering when integrated with

Branch master

Ke

#52304 failed

Ran for 17 min 40 sec

Total time 10 hrs 26 min 10 sec

10 hours ago

Build jobs

View config

✖	# 52304.1	AMD64	Trusty	Java	MAVEN_CHECKS=true	10 min 51 sec
✓	# 52304.2	AMD64	Trusty	Java	WEBUI_CHECKS=true	58 sec
✓	# 52304.3	AMD64	Trusty	Java	TEST_SPECIFIC_MODULES=presto-tests	6 min 7 sec
✓	# 52304.4	AMD64	Trusty	Java	TEST_SPECIFIC_MODULES=presto-tests	24 min 50 sec
✓	# 52304.5	AMD64	Trusty	Java	TEST_SPECIFIC_MODULES=presto-tests	7 min 45 sec
✓	# 52304.6	AMD64	Trusty	Java	TEST_SPECIFIC_MODULES=presto-tests	8 min 4 sec

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

Example CI Pipeline - TravisCI

At a glance, see history of build

prestodb / presto

build passing

Current

Branches

Build History

Pull Requests

More options

<div>✓ master</div> <div>James Sun</div>	<div>This patch bumps Alluxio dependency to 2.3.0-2</div>	<div>✓ #52300 passed</div> <div>36392a2</div>	<div>10 hrs 49 min 31 sec</div> <div>2 days ago</div>
<div>! master</div> <div>Andrii Rosa</div>	<div>Handle query level timeouts in Presto on Spark</div>	<div>✗ #52287 errored</div> <div>aa55ea7</div>	<div>11 hrs 6 min 44 sec</div> <div>2 days ago</div>
<div>! master</div> <div>Wenlei Xie</div>	<div>Fix flaky test for TestTempStorageSingleStreamSp</div>	<div>✗ #52284 errored</div> <div>193a4cd</div>	<div>11 hrs 50 min 37 sec</div> <div>2 days ago</div>
<div>✓ master</div> <div>Andrii Rosa</div>	<div>Check requirements under try-catch</div>	<div>✓ #52283 passed</div> <div>fff331f</div>	<div>11 hrs 3 min 20 sec</div> <div>2 days ago</div>
<div>✓ master</div> <div>Maria Basmanova</div>	<div>Update TestHiveExternalWorkersQueries to create</div>	<div>✓ #52282 passed</div> <div>746d7b5</div>	<div>10 hrs 55 min 37 sec</div> <div>2 days ago</div>
<div>✓ master</div> <div>Maria Basmanova</div>	<div>Introduce large dictionary mode in SliceDictionar</div>	<div>✓ #52277 passed</div> <div>a29d93a</div>	<div>10 hrs 43 min 30 sec</div> <div>2 days ago</div>

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

CI In Practice: HW3 Autograder

test.yml (CI workflow file)

```
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

test.yml

on: push

✓ build 30s

Matrix: test

✓ test (a) 3m 6s

✓ test (b) 3m 3s

✓ test (c) 2m 58s

✓ test (ts-ignore) 5s

✓ test (linting-error) 31s

✓ test (non-green-tests) 35s

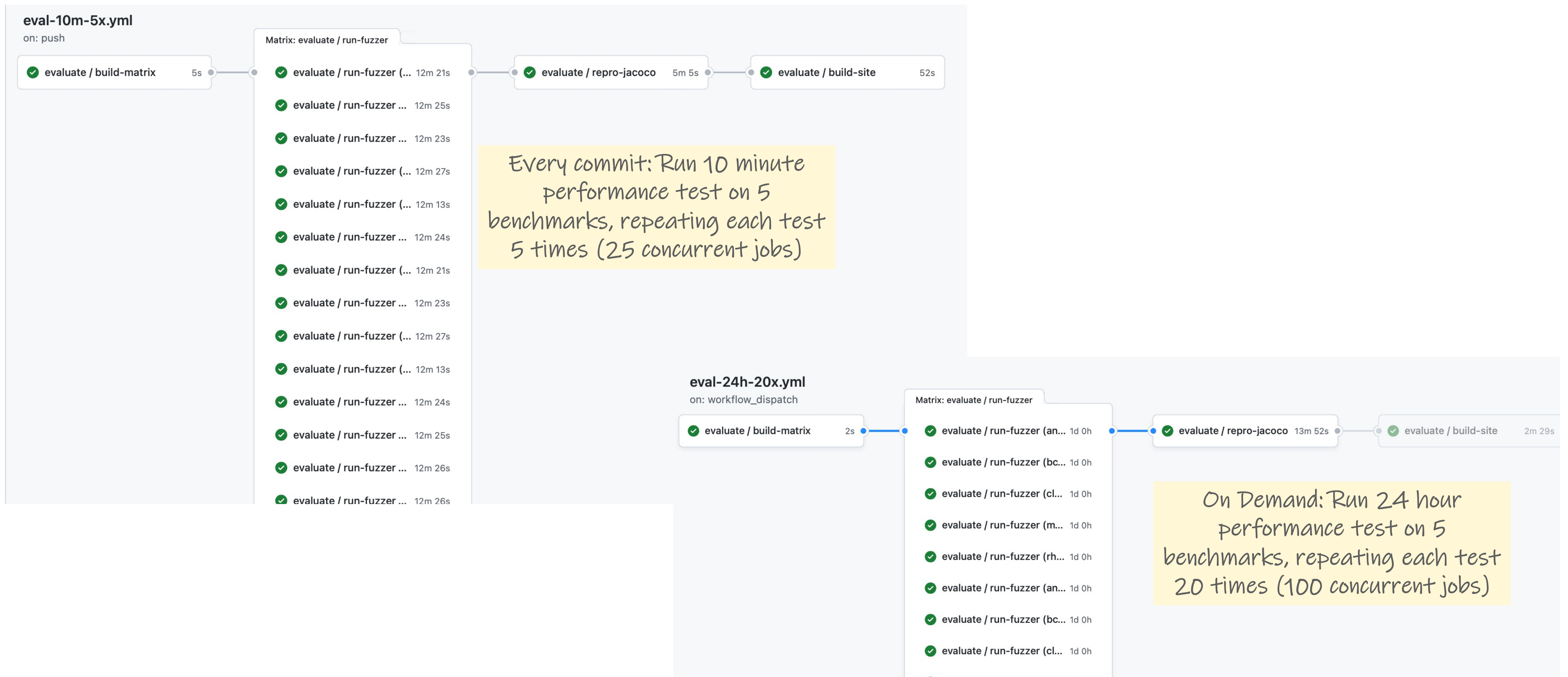
✓ test (empty) 4s

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

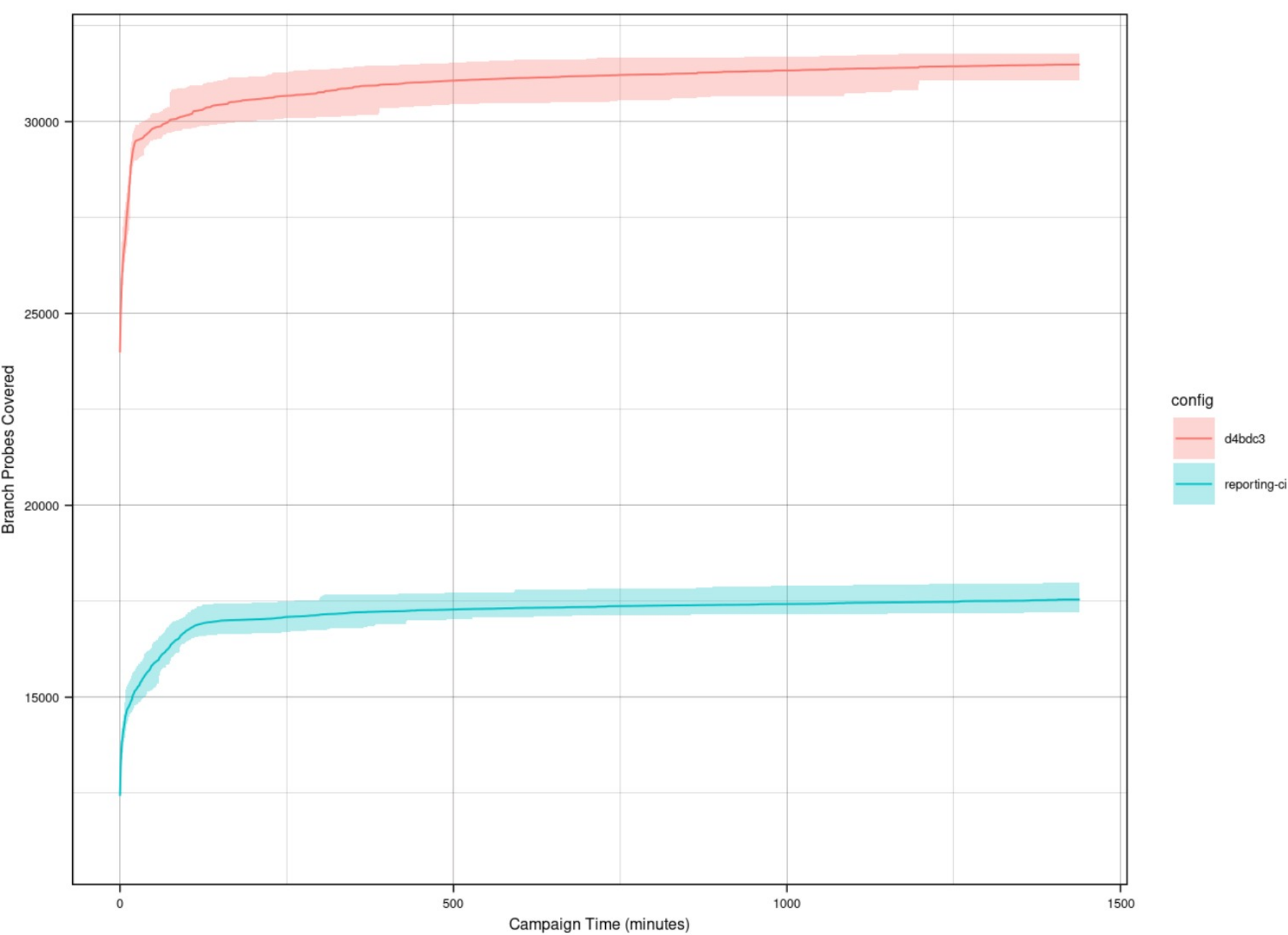
CI Pipelines Automate Performance Testing



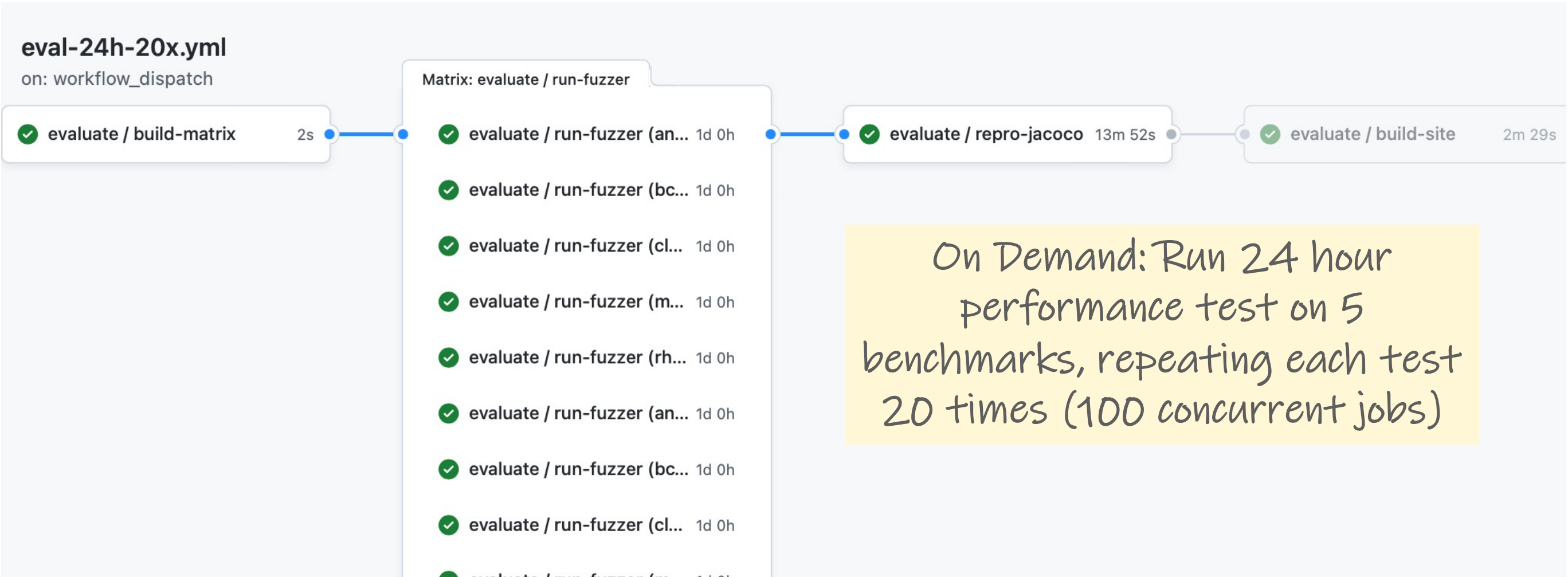
CI Pipelines Automate Performance Testing

closure

Branch Probes Over Time



[Download this graph as PDF](#)



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

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]