CS 4530 Software Engineering

Module 14: Continuous Development Processes

Adeel Bhutta and 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
- Describe the benefits of a culture of code review
- Describe strategies for performing quality-assurance on software as and after it is delivered

CS 4530 Software Engineering

Module 14.1: Code Review

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Why should we perform code review?

Code review increases breadth of knowledge of code:

- Other people ”know” the code
- Easier to handle someone cycling off project

Verbalizing decisions improves their quality:

- The process of writing an explanation encourages critical thinking

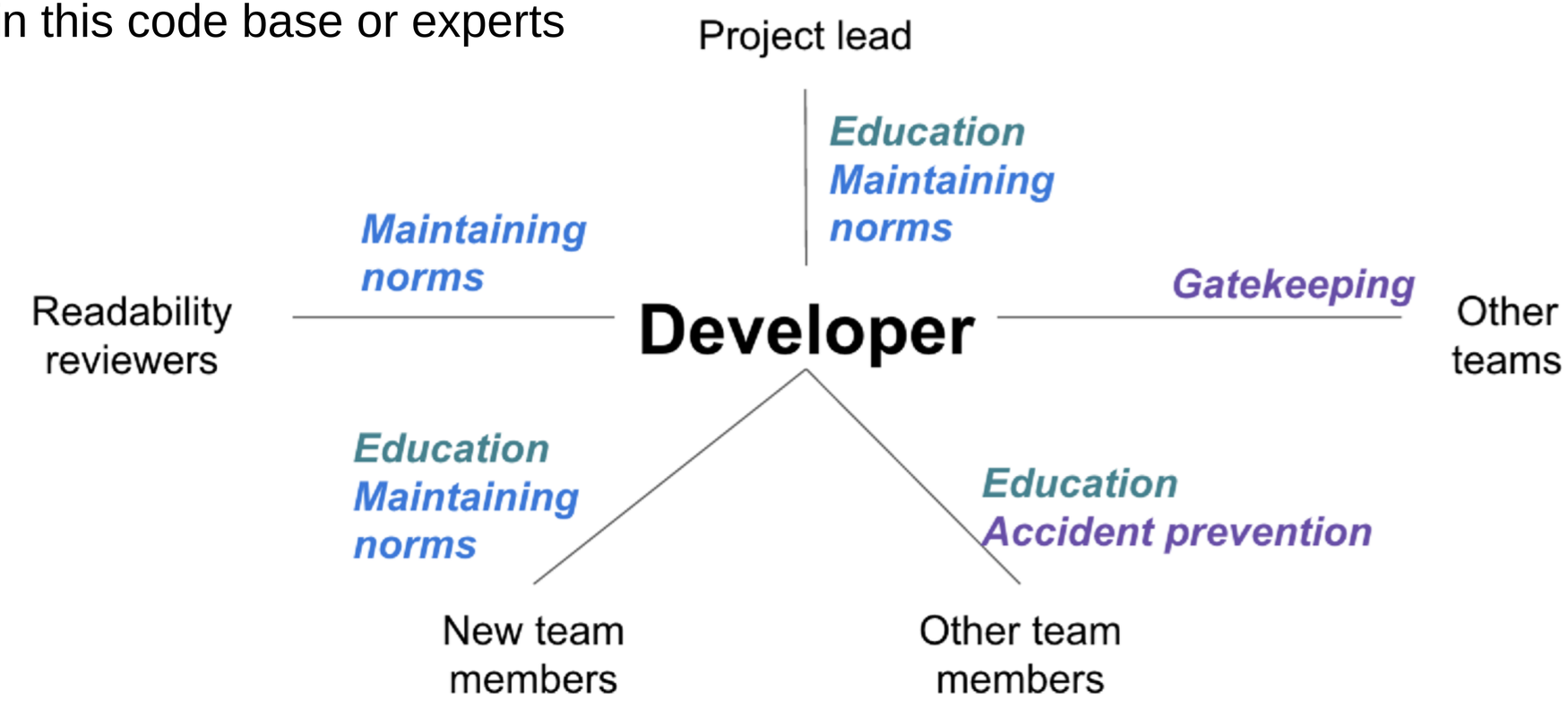
Code reviews improve quality of code base:

- Knowing code is reviewed pushes devs to make it more presentable and understandable

Many stakeholders can benefit from code review

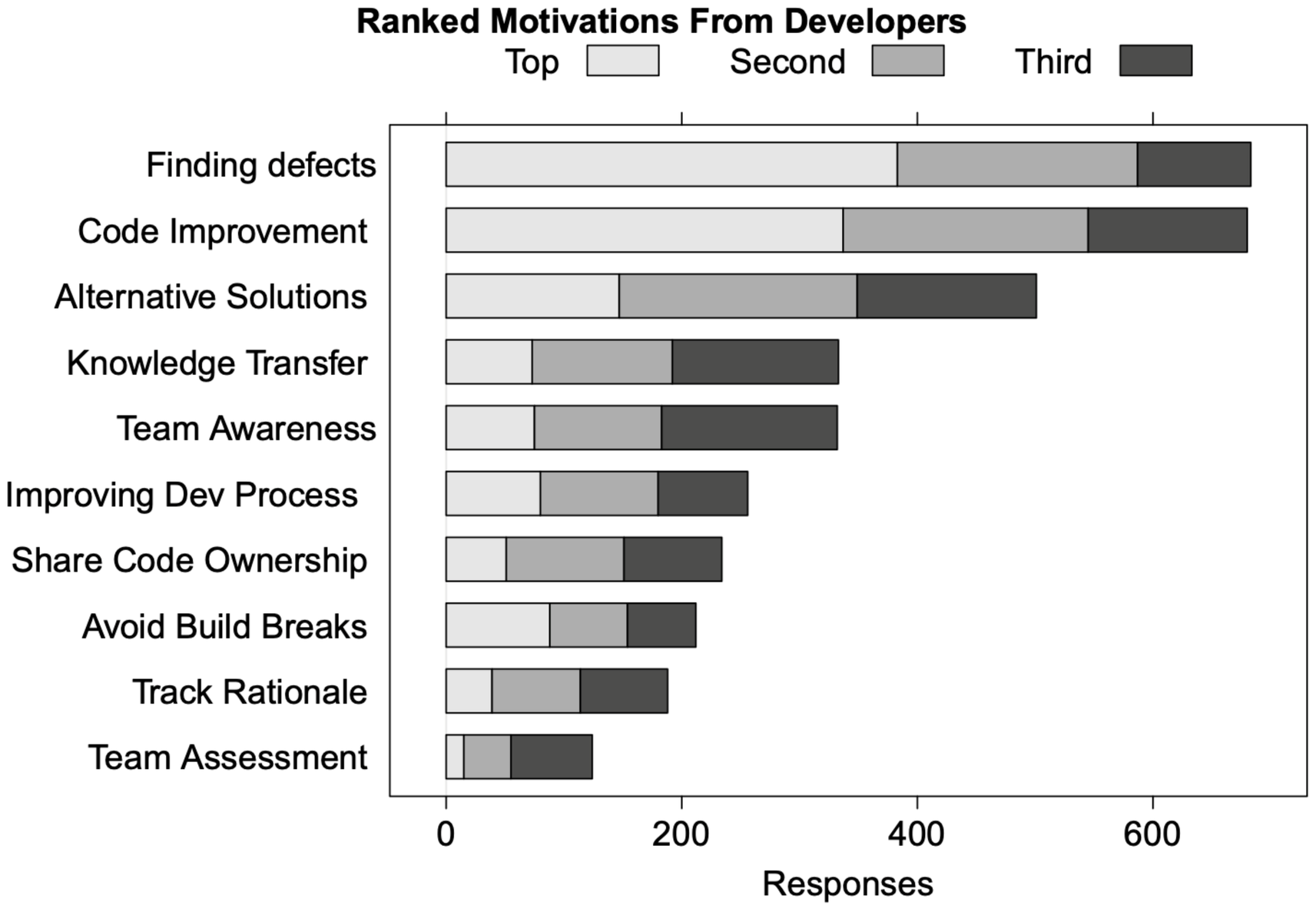
Reviewers might be…

- An owner of the code being changed or added to
- Someone to verify that the code meets standards.
- Someone to ensure documentation is consistent.
- Other people interested in this code base or experts



“Modern Code Review: A Case Study at Google”, Sadowski et al, ICSE 2018

Code reviews have many benefits



"Expectations, Outcomes, and Challenges of Modern Code Review", Bacchelli & Bird, ICSE 2013

Code Inspection

Formal process of reading through code as a group;

- Applied to all project documents;
- A 3-5 person team reads the code aloud and explains what is being done;
- Each person has a specific role (moderator, reviewer, reader, scribe, observer, author)
- Usually a 60 minute meeting;
- Less efficient (defects/cost) than modern review processes.
- Very waterfall.
- Traceable, measurable

Code review should be a formal process

A code review is the process in which the code's author explains it to peers:

- What should it do?
- How does it do it?
- How confident are we in it?
- What are results of running tests?

A code review often concerns a code change ("diff")

Code review checklist

Consider:

- Am I able to understand the code easily?
- Does the code follow our style guidelines?
- Is the same code duplicated more than once?
- Is this file (or change) too big?
- Does this code meet our non-functional requirements?
- Is this code maintainable?
- Does this code have unintended side-effects?

Code review: How they do it at Google

At Google, reviewers get changes, explanation and all test results: review is asynchronous.

Elsewhere reviews can be in person:

- More heavyweight, cannot be as common.

Review must be professional and impersonal:

- No one is being “attacked” (or, no one should be).

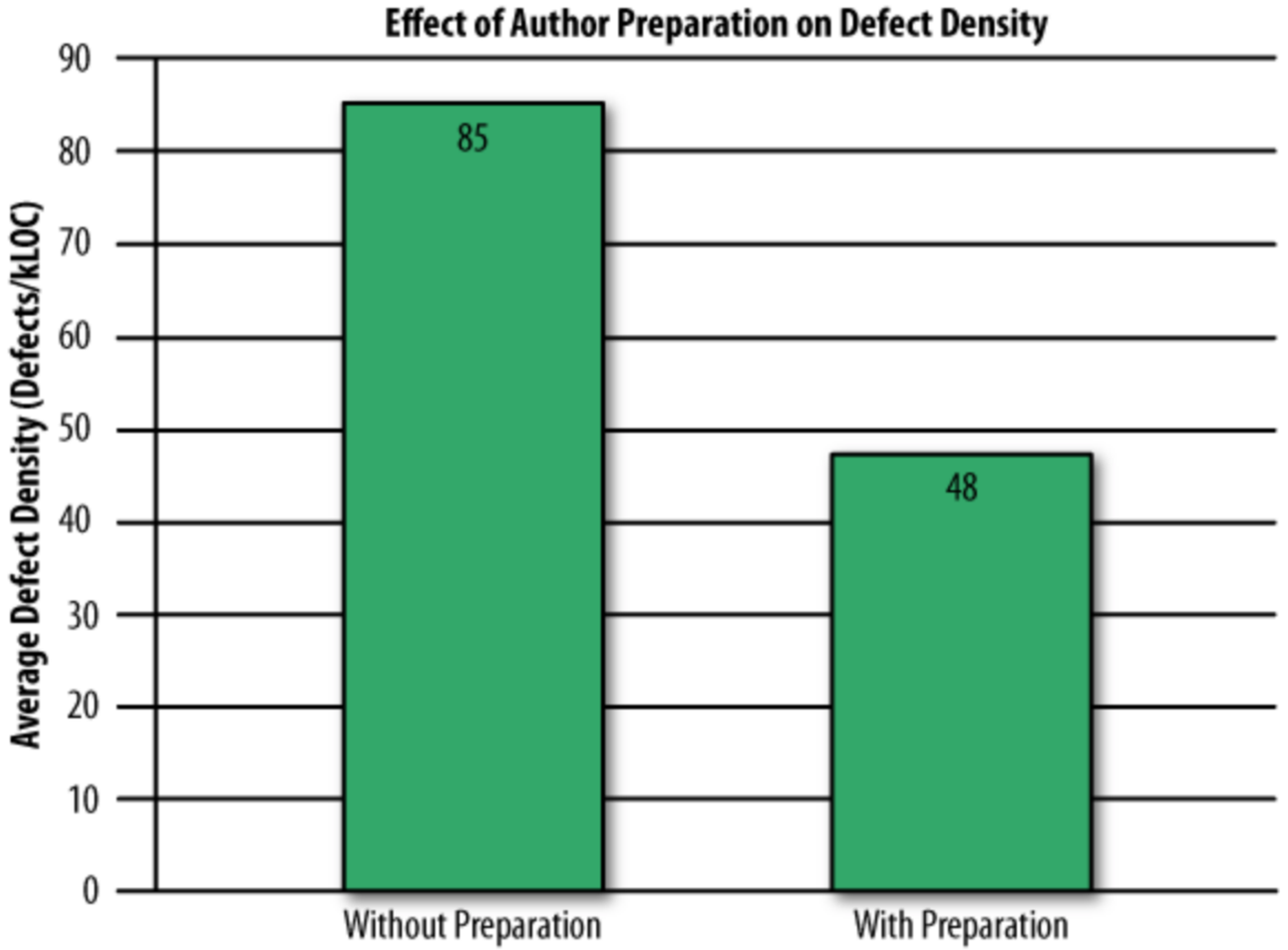
Don’t rehash design arguments (defer to author).

All suggestions and criticisms must be addressed:

- At least in the negative.

Self-review is no substitute for peer review

Study of 300 reviews at Cisco in 2006



Even if developers pre-review their code, many defects still found in peer review

"Best Kept Secrets of Peer Code Review", Jason Cohen, SmartBear Software, 2006

Code review: example on pull request

...re-api/src/main/java/org/apache/maven/surefire/booter/CommandReader.java — Hide resolved

```
            case BYE_ACK:
                //After SHUTDOWN no more commands can come. Hence, do NOT go back to blocking in IO
                callListeners( command );
                return;
            default:
                callListeners( command );
```

Tibor17 on Nov 12, 2019 Contributor

The listeners are called here. But we can put IF condition:
`IF BYE_ACK -> return` at the end of the default case.

Tibor17 on Nov 12, 2019 Contributor

Instead of calling the `return` we can make softer exit with `CommandReader.this.state.set( TERMINATED )`.

eolivelli on Dec 17, 2019 Contributor

Yes, I came to this same conclusion, change the state to TERMINATED.

jon-bell on Dec 19, 2019 Author Contributor

Changed.

Reply...

Unresolve conversation

jon-bell marked this conversation as resolved.

Code Reviews and Programmer’s Ego

Remember:

- Code review means someone’s looking over your work
- You might have some attachment to it
- Criticisms: sometimes hard not to take personally
- Acknowledge a criticism and move on
- Acknowledgment doesn’t imply that the author agrees with the content of the criticism
- The review is not about you, the goal is to improve code

CS 4530 Software Engineering

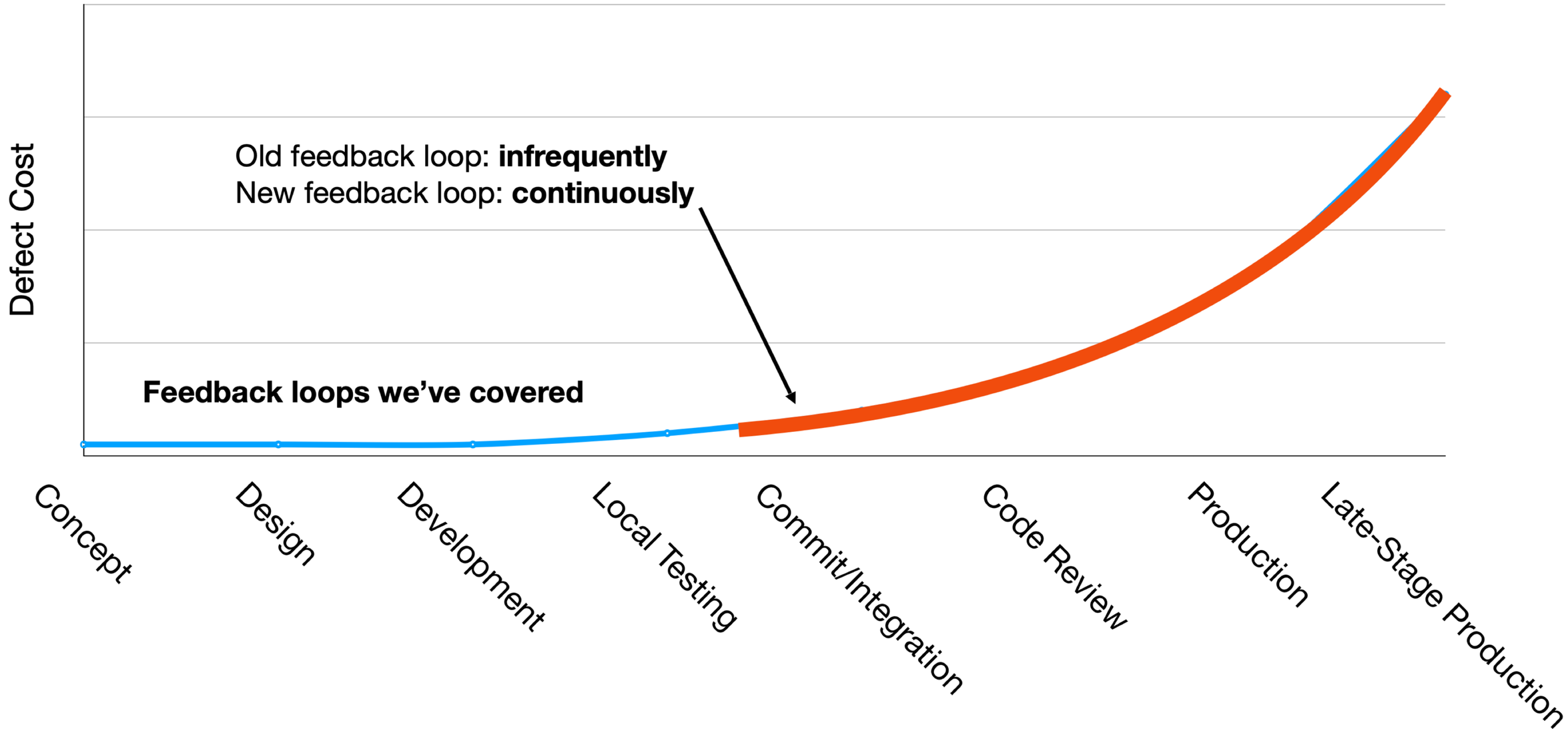
Module 14.2: Continuous Integration

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

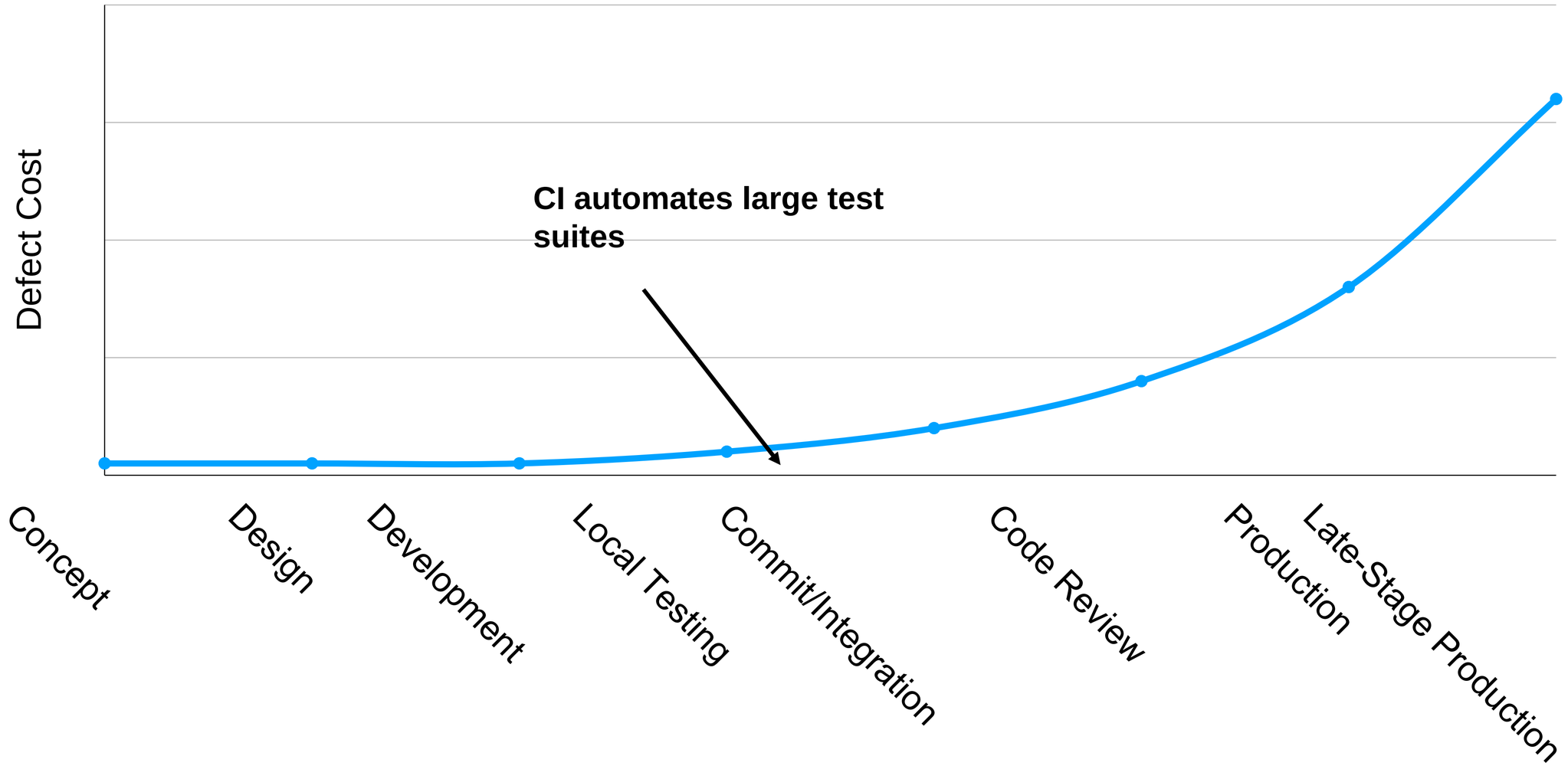
Agile values fast quality feedback loops

Faster feedback = lower cost to fix bugs



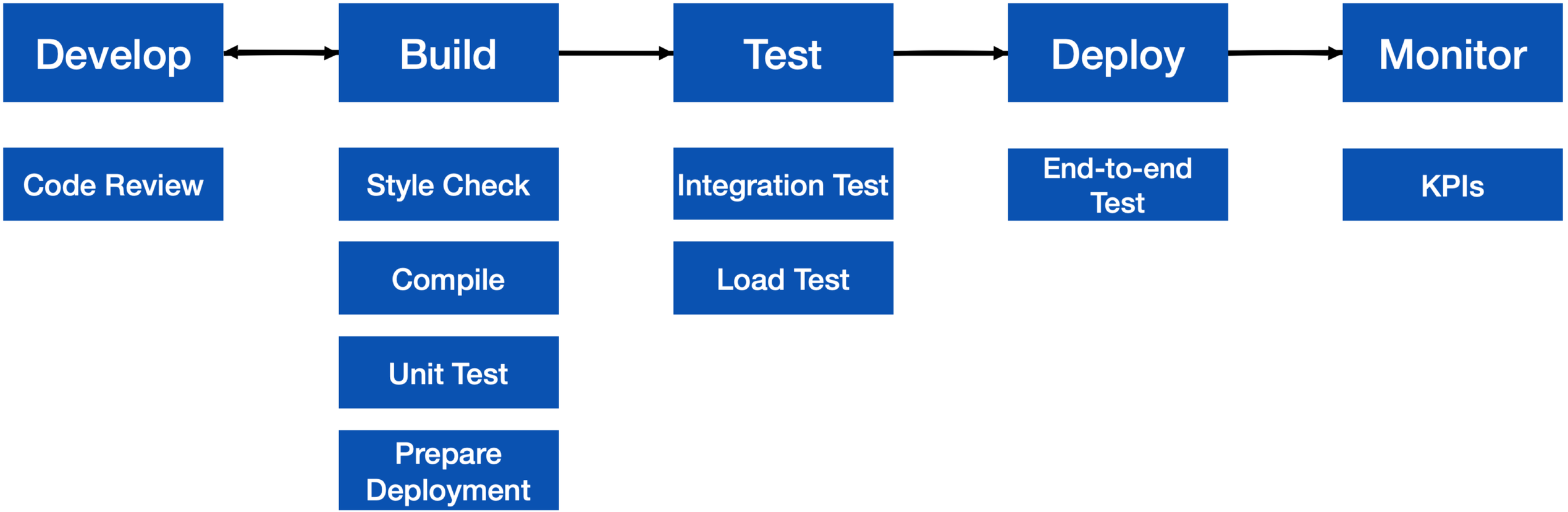
Continuous Integration

Fast feedback on integration errors



Continuous Development/Continuous Integration

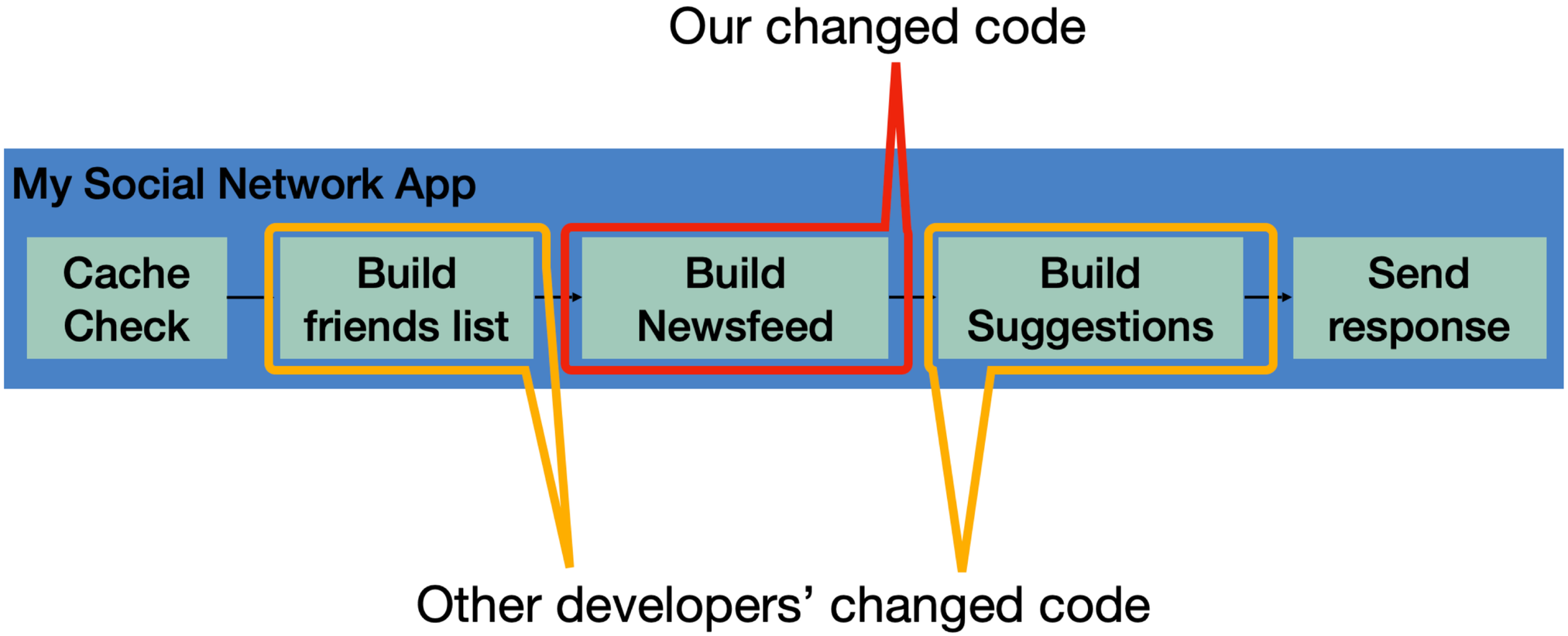
Improving quality & velocity with frequent, fast feedback loops



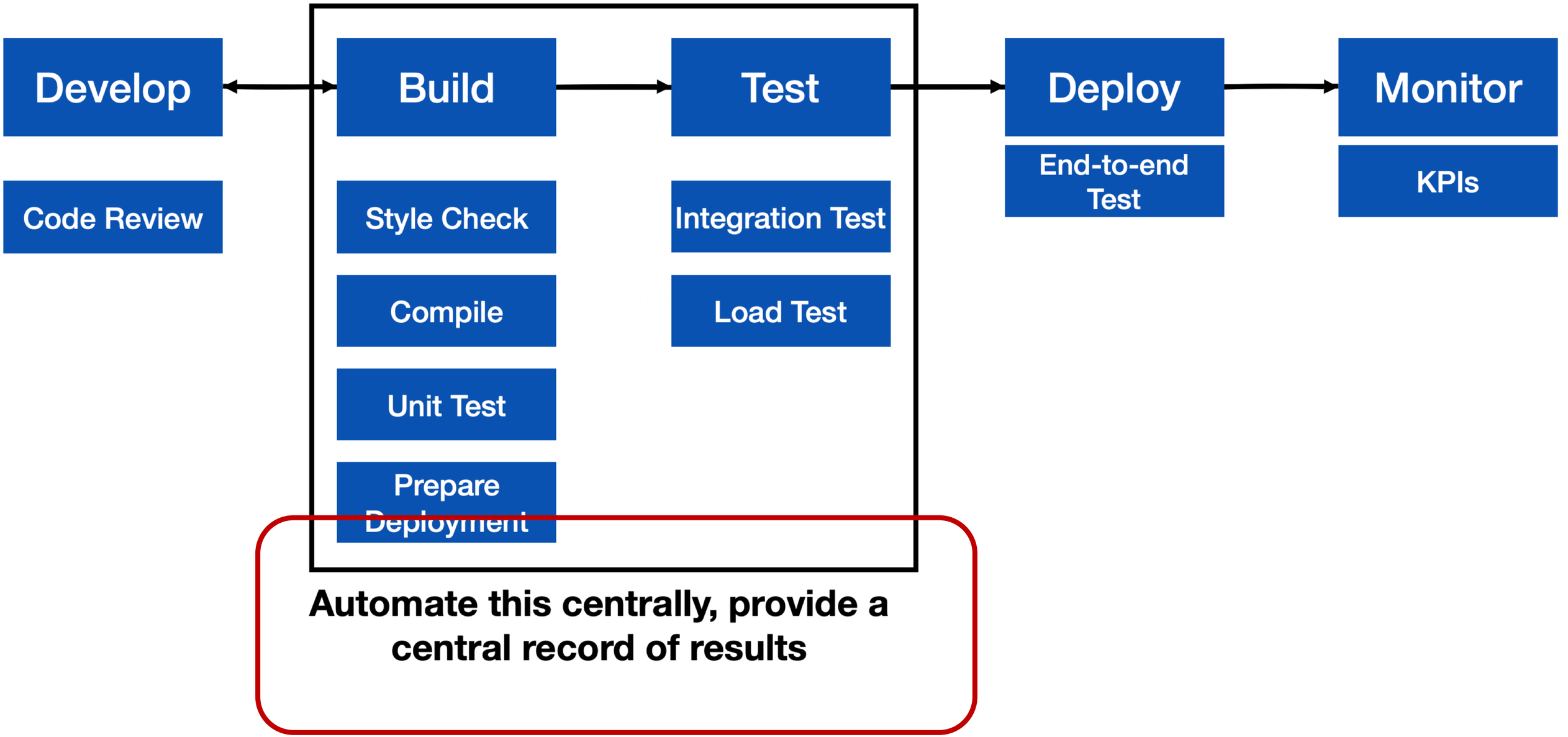
Continuous Integration

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?

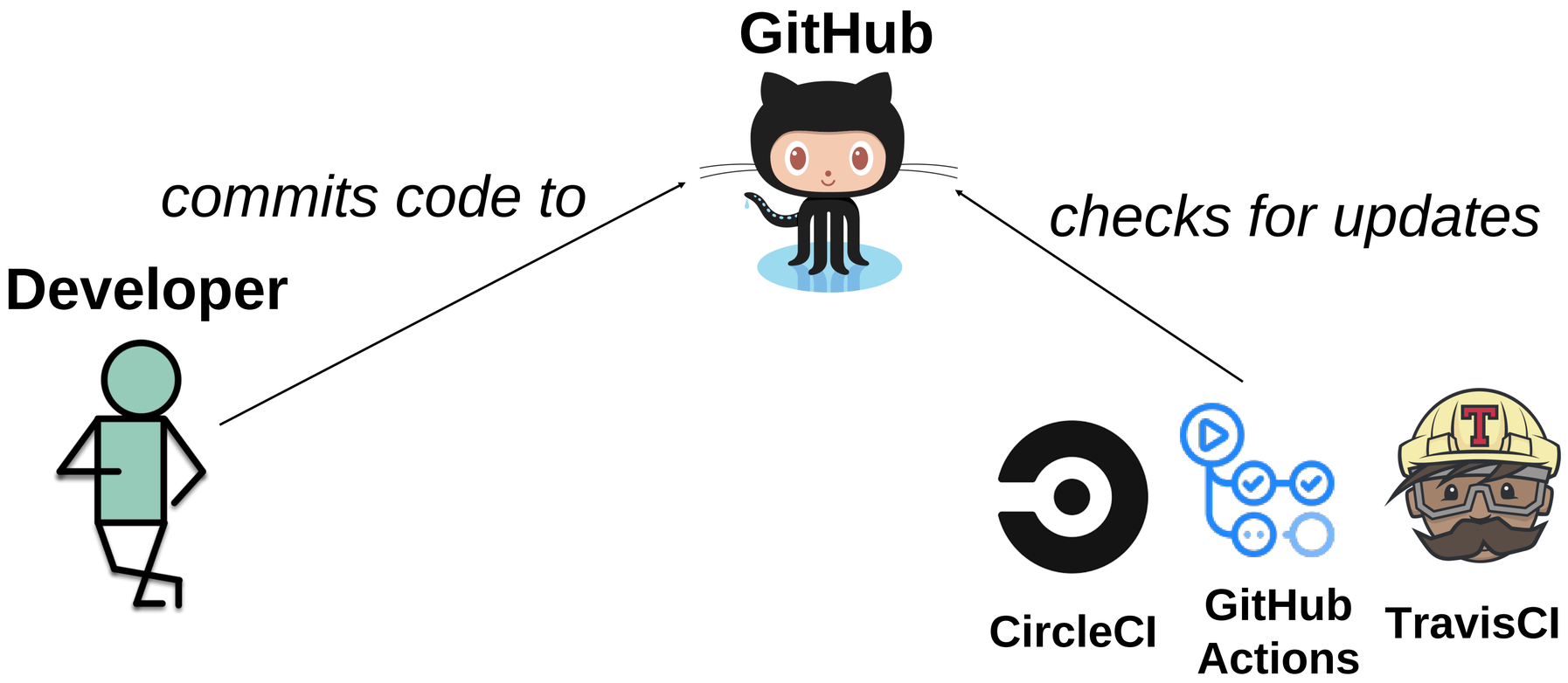


CI is a software pipeline



CI in practice

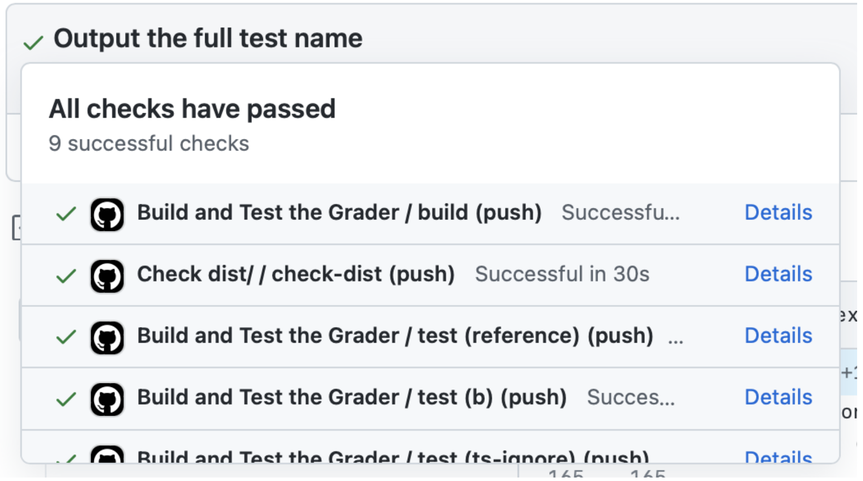
Small scale, with a service like CircleCI, GitHub Actions or TravisCI

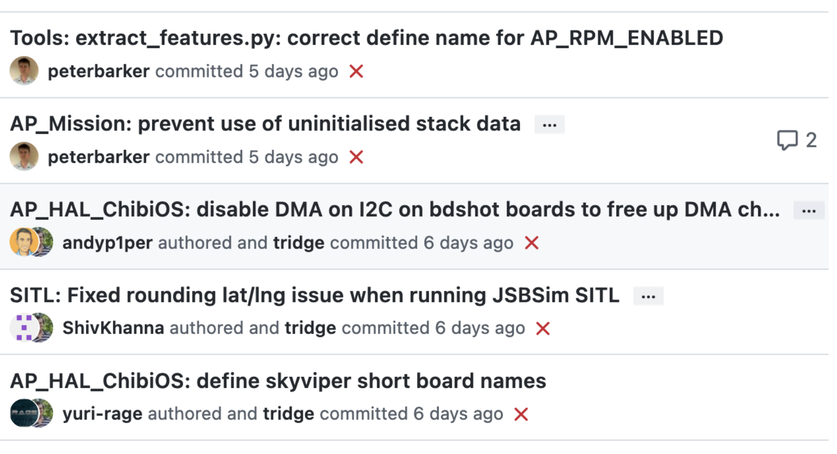


Runs build for each commit

Attributes of effective CI processes

- Do not allow builds to remain broken for a long time
- CI should run for every change
- CI should be fast, providing feedback within minutes or hours
- CI should not completely replace pre-commit testing



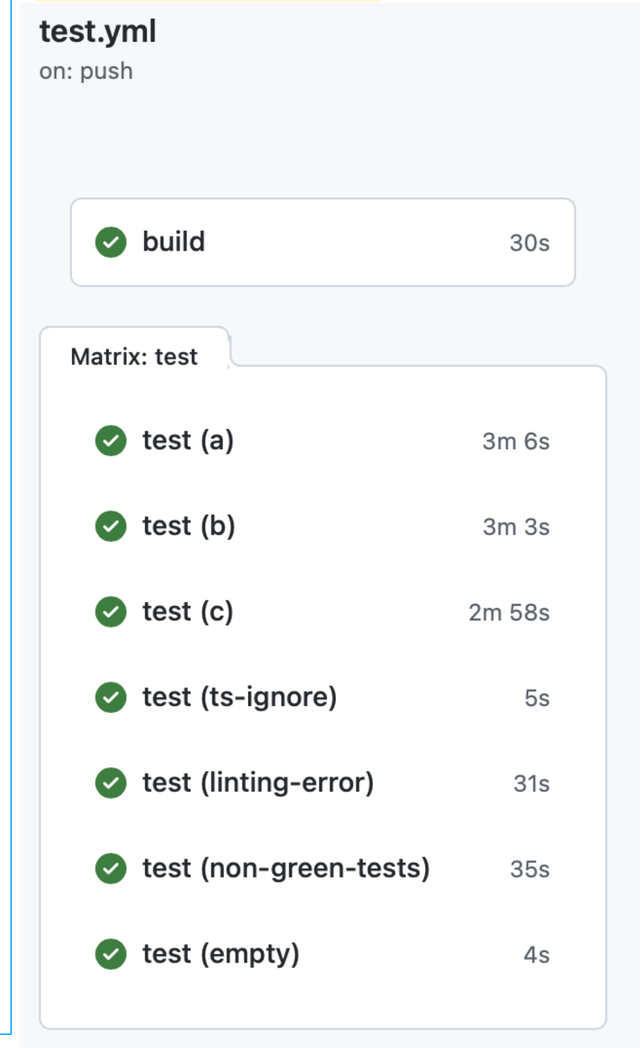


CI In Practice: autograder

test.yml (CI workflow file)

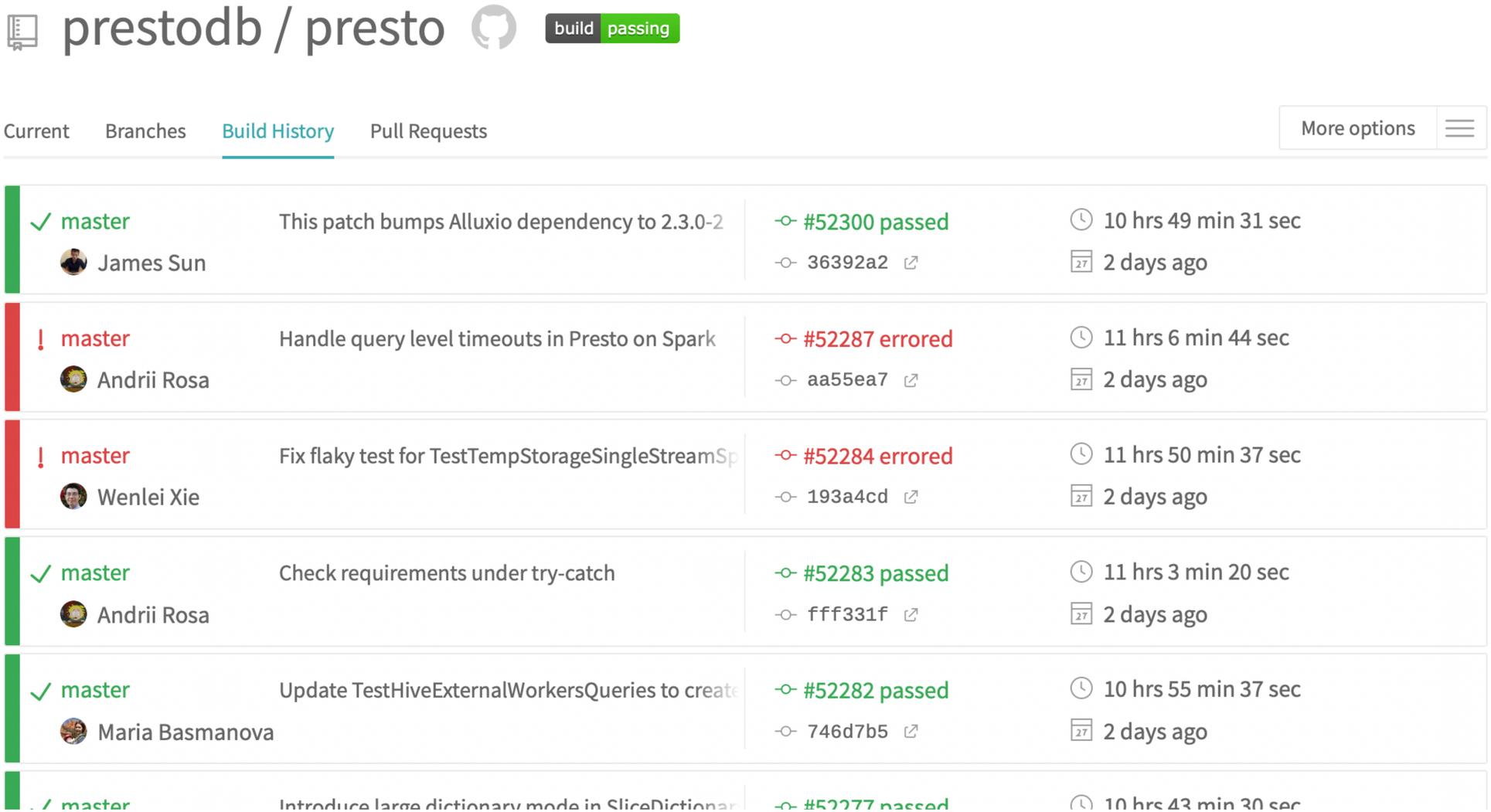
```yaml
name: 'Build and Test the Grader'
on: # rebuild any PRs and main branch changes
  pull_request:
  push:
    branches:
      - main
      - 'releases/*'
jobs:
  build:
    runs-on: self-hosted
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          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



Example CI Pipeline - TravisCI

At a glance. see history of build

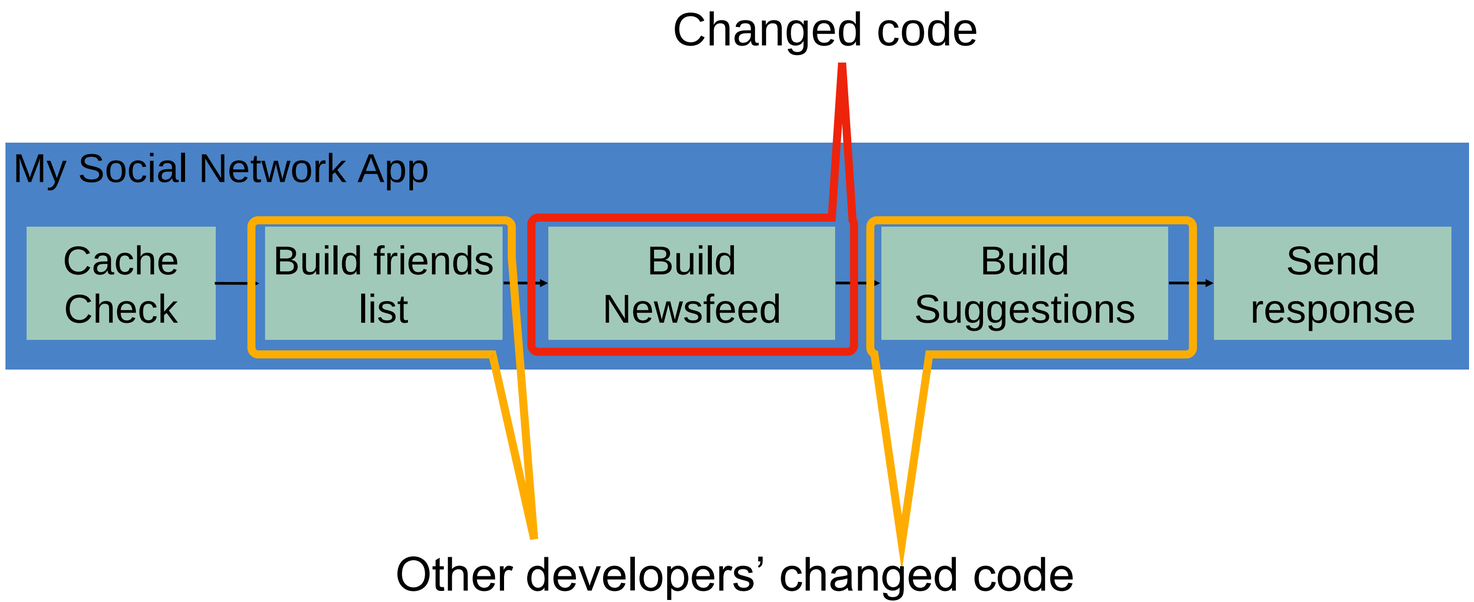


https://travis-ci.com/github/prestodb/presto

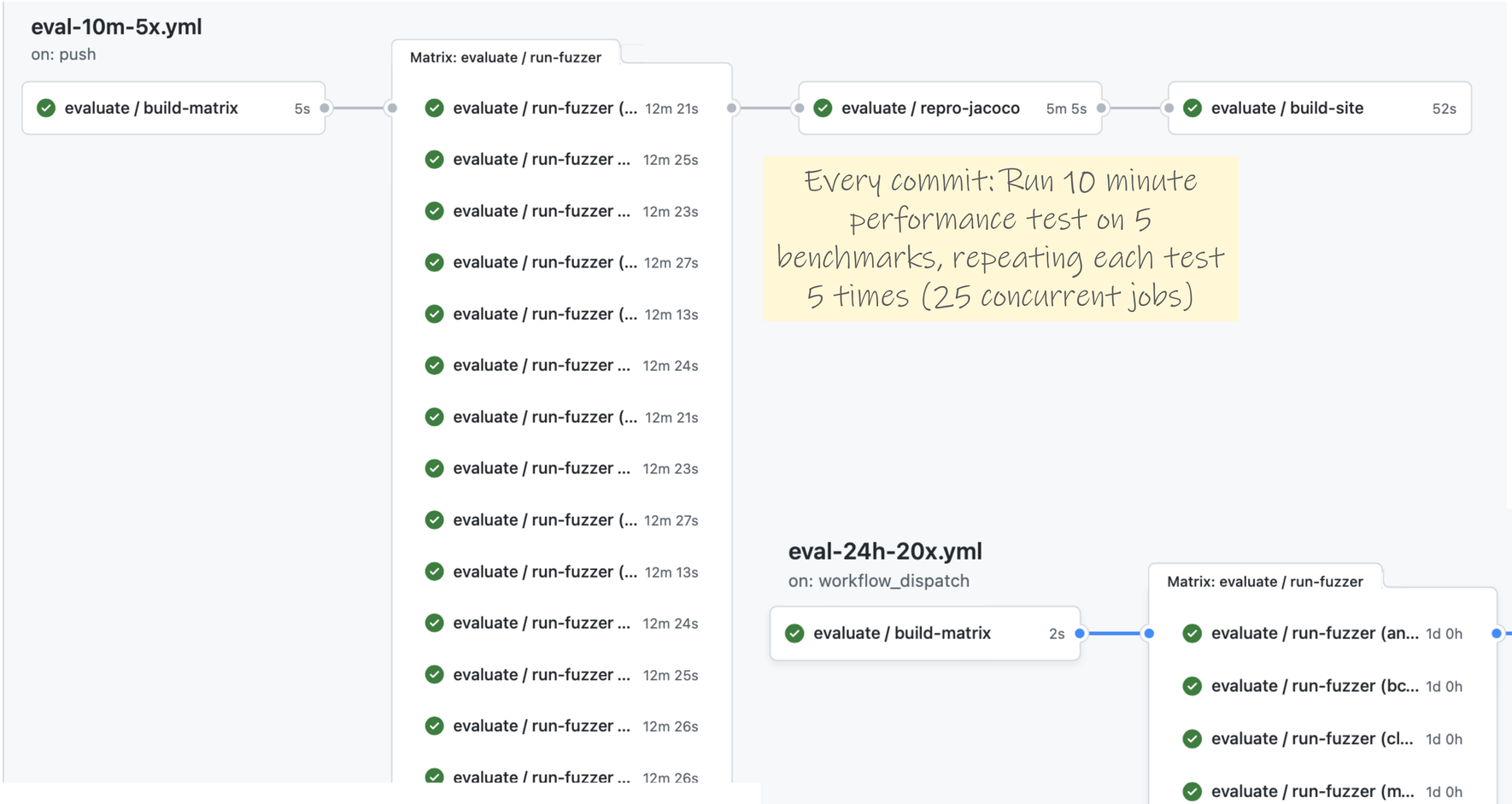
How do we apply continuous integration?

Testing the right things at the right time

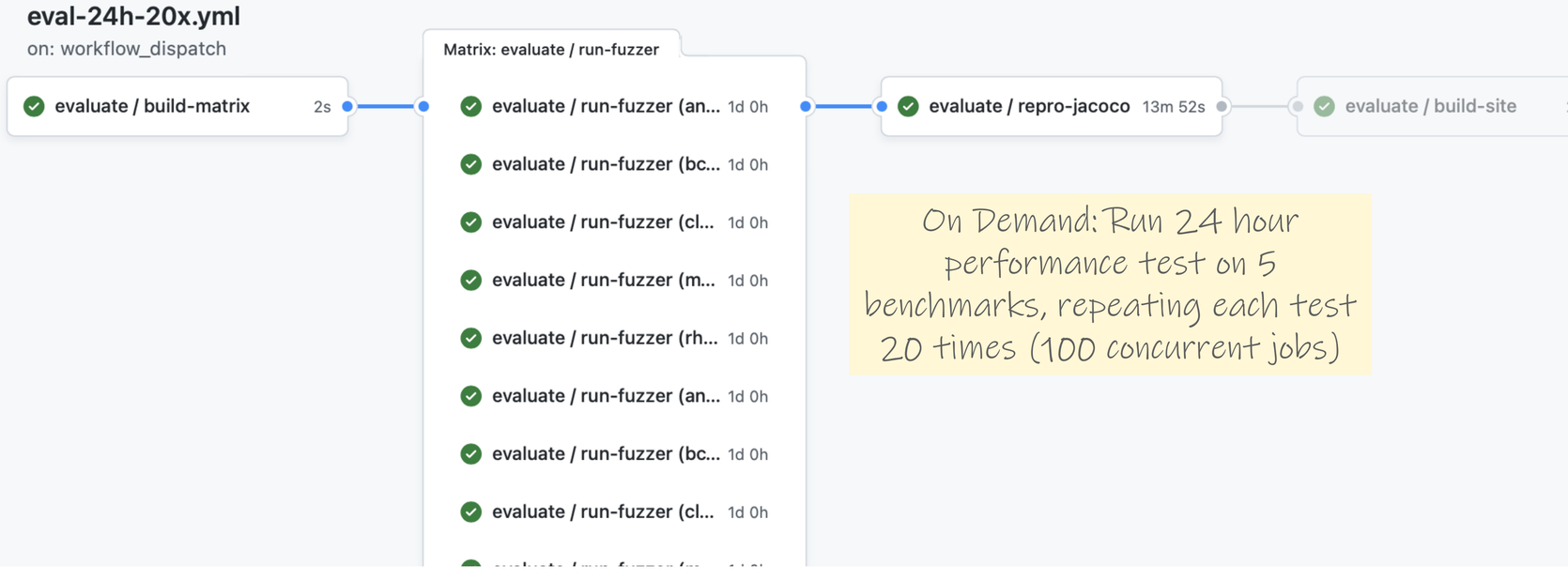
- Do we integrate changes immediately, or do a pre-commit test?
- Which tests do we run when we integrate?
- How do we compose the system under test at each point?



CI Pipelines automate performance testing



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



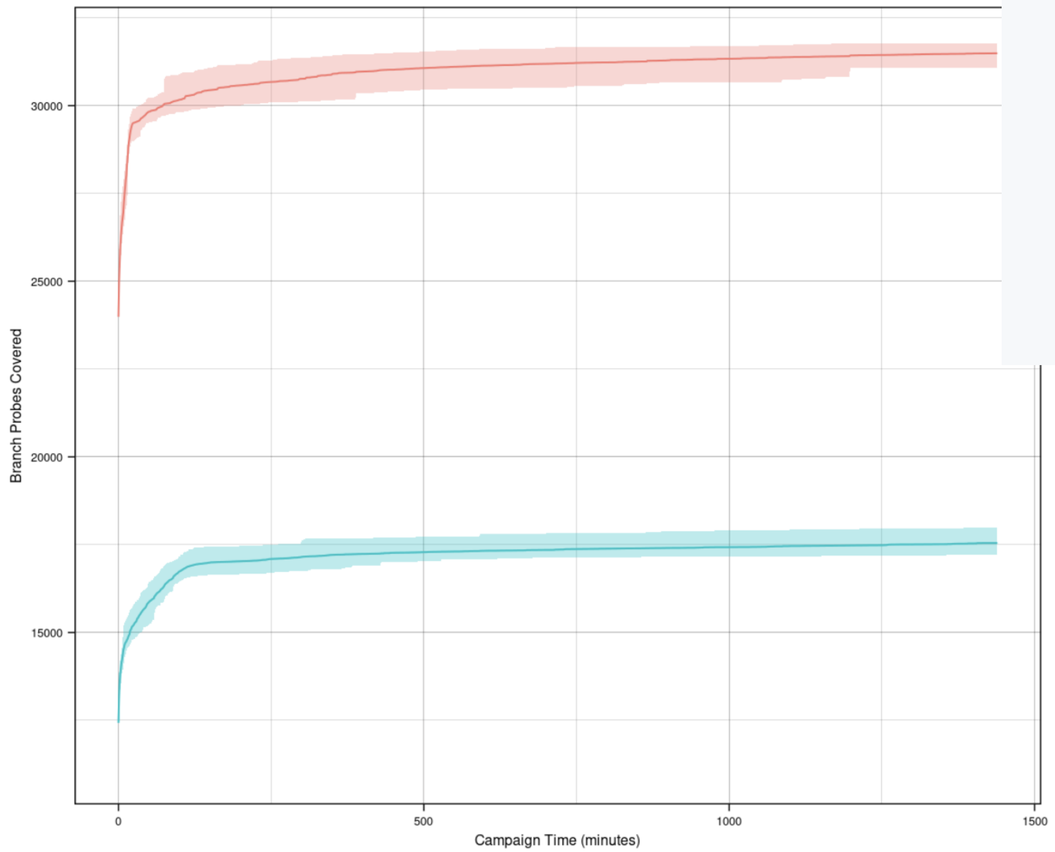
On Demand: Run 24 hour performance test on 5 benchmarks, repeating each test 20 times (100 concurrent jobs)

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

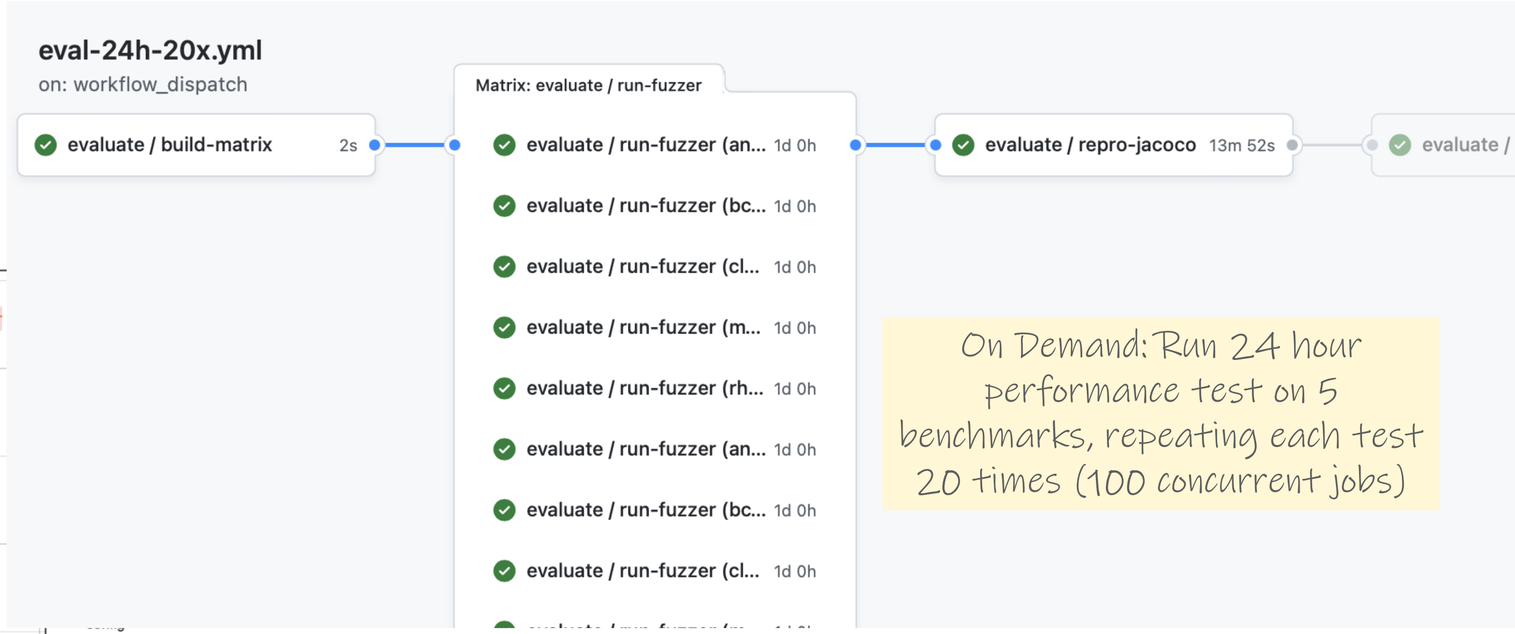
CI Pipelines automate benchmarking

closure

Branch Probes Over Time



Download this graph as PDF



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 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). Block merge if they fail.
- 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)

CS 4530 Software Engineering

Module 14.3: Continuous Deployment

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Continuous Delivery

“Faster is safer”: Key values of continuous delivery

- Release frequently, in small batches
- Maintain key performance indicators to evaluate the impact of updates
- Phase roll-outs
- Evaluate business impact of new features

Staging environments

Enabling Continuous Delivery

As software gets more complex with more dependencies, it's impossible to simulate the whole when testing

Idea: Deploy to a complete production-like environment, but don't have all use it

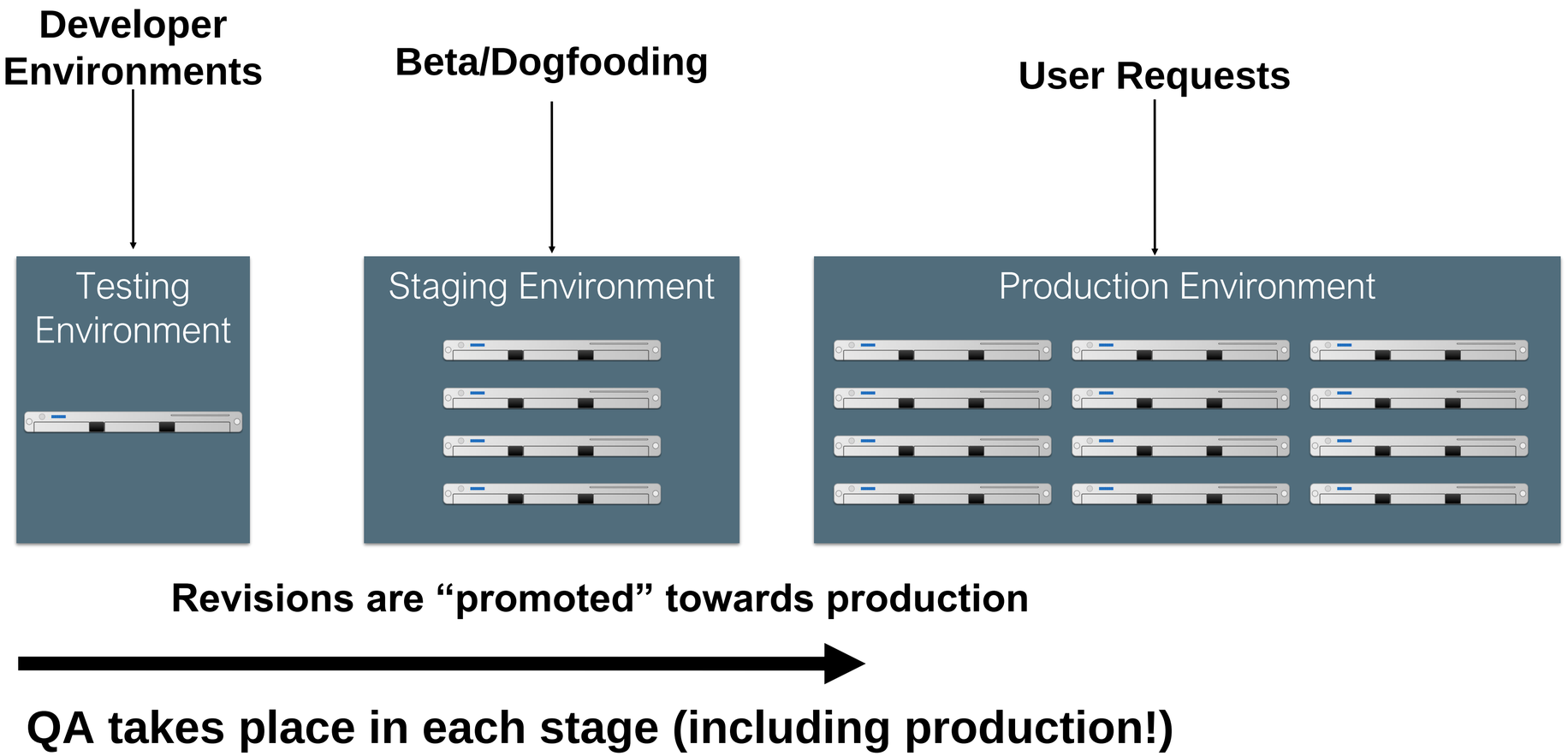
Examples:

- “Eat your own dogfood”
- Beta/Alpha testers

Lower risk if a problem occurs in staging than in production

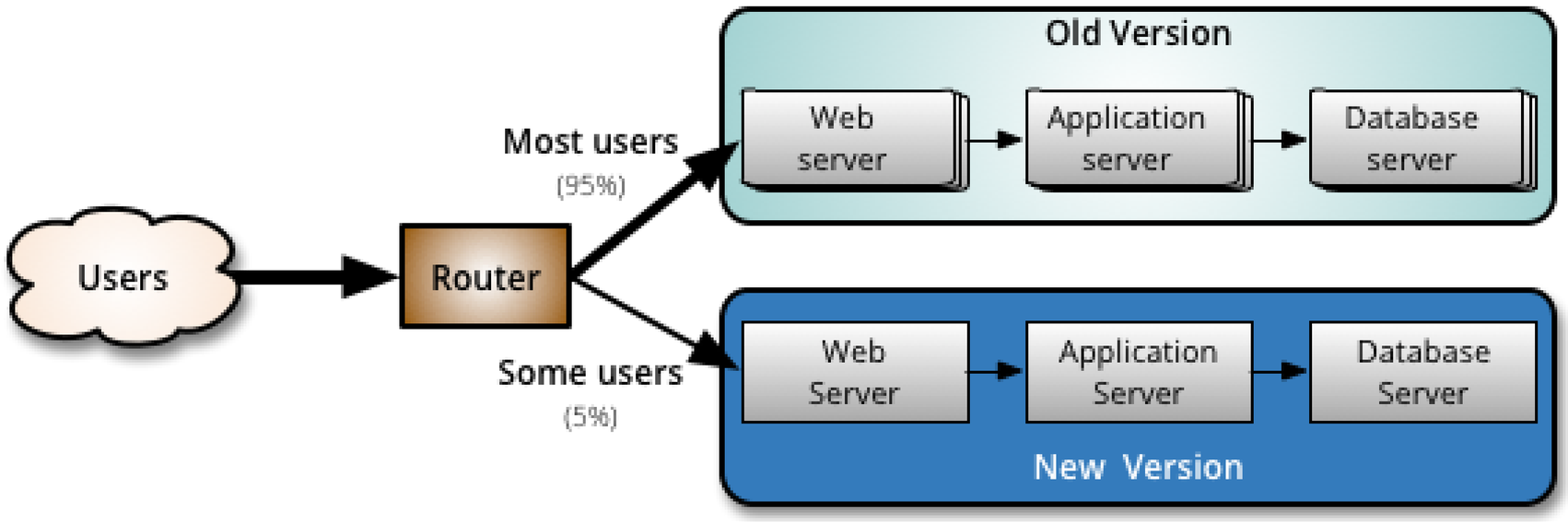
Test-Stage-Production

Continuous Delivery in Action



A/B Deployments with Canaries

Mitigating risk in continuous delivery



Monitor both:
But minimize impact of problems in new version

Deployment Philosophy: Instagram

“Faster is safer”

“If stuff blows up it affects a very small percentage of people”

Instagram cofounder and CTO Mike Krieger

https://www.fastcompany.com/3047642/do-the-simple-thing-first-the-engineering-behind-instagram

Operations Responsibility

DevOps in a slide

Once we deploy, someone has to monitor, make sure it's running OK, no bugs, etc

Assume 3 environments: Test, Staging, Production

Whose job is it?

	Developers			Operators		
Waterfall				Test	Staging	Production
Agile	Test				Staging	Production
DevOps	Test	Staging	Production			Production

Release Pipelines

How quickly is my change deployed?

- Even if you are deploying every day, you still have some latency
- A new feature I develop today won't be released today
- But, a new feature I develop today can begin the release pipeline today (minimizes risk)
- Release Engineer: gatekeeper who decides when code ready to go out, oversees deployment process

Monitoring

The last step in continuous deployment: track metrics

Hardware

- Voltages, temperatures, fan speeds, component health

OS

- Memory usage, swap usage, disk space, CPU load

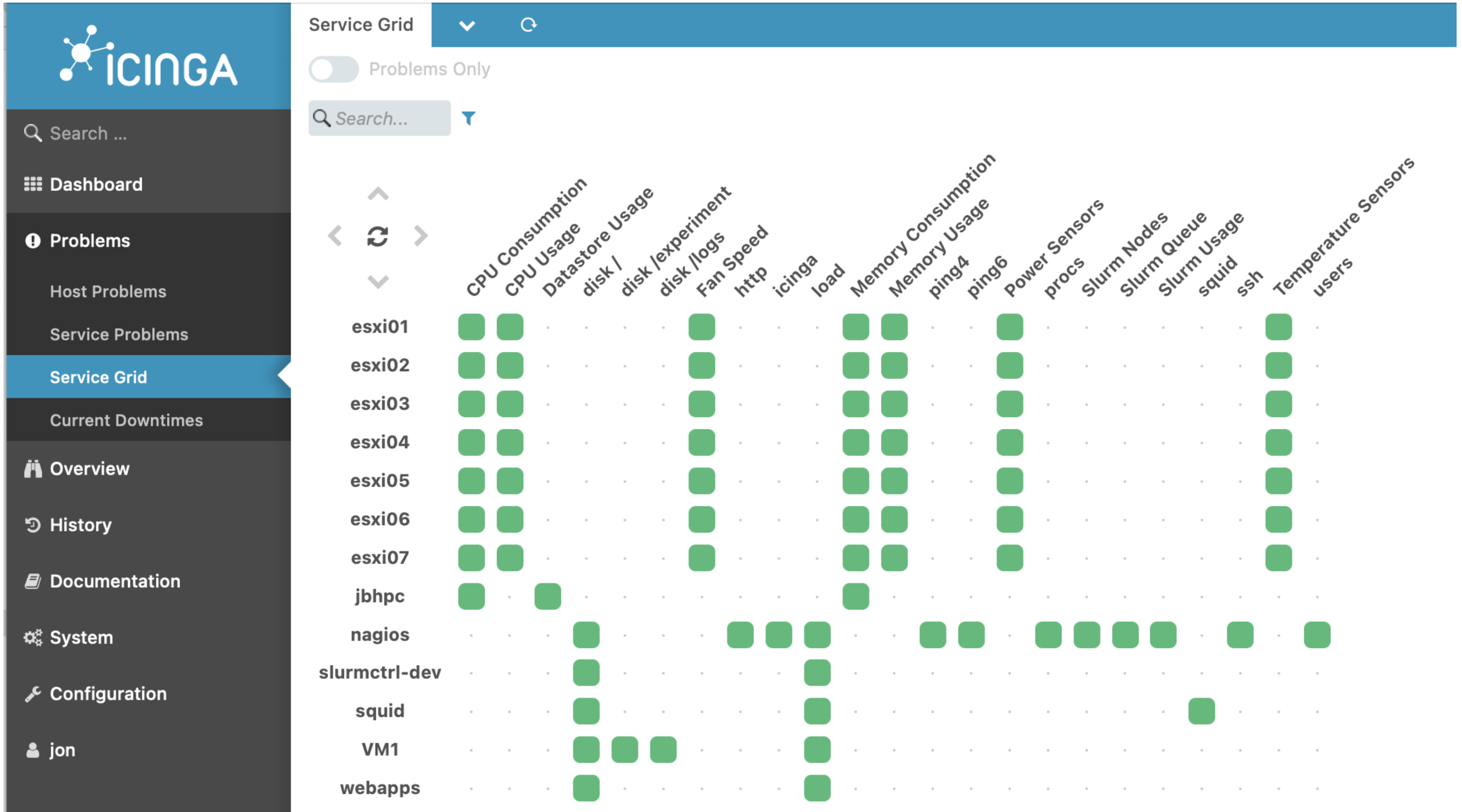
Middleware

- Memory, thread/db connection pools, connections, response time

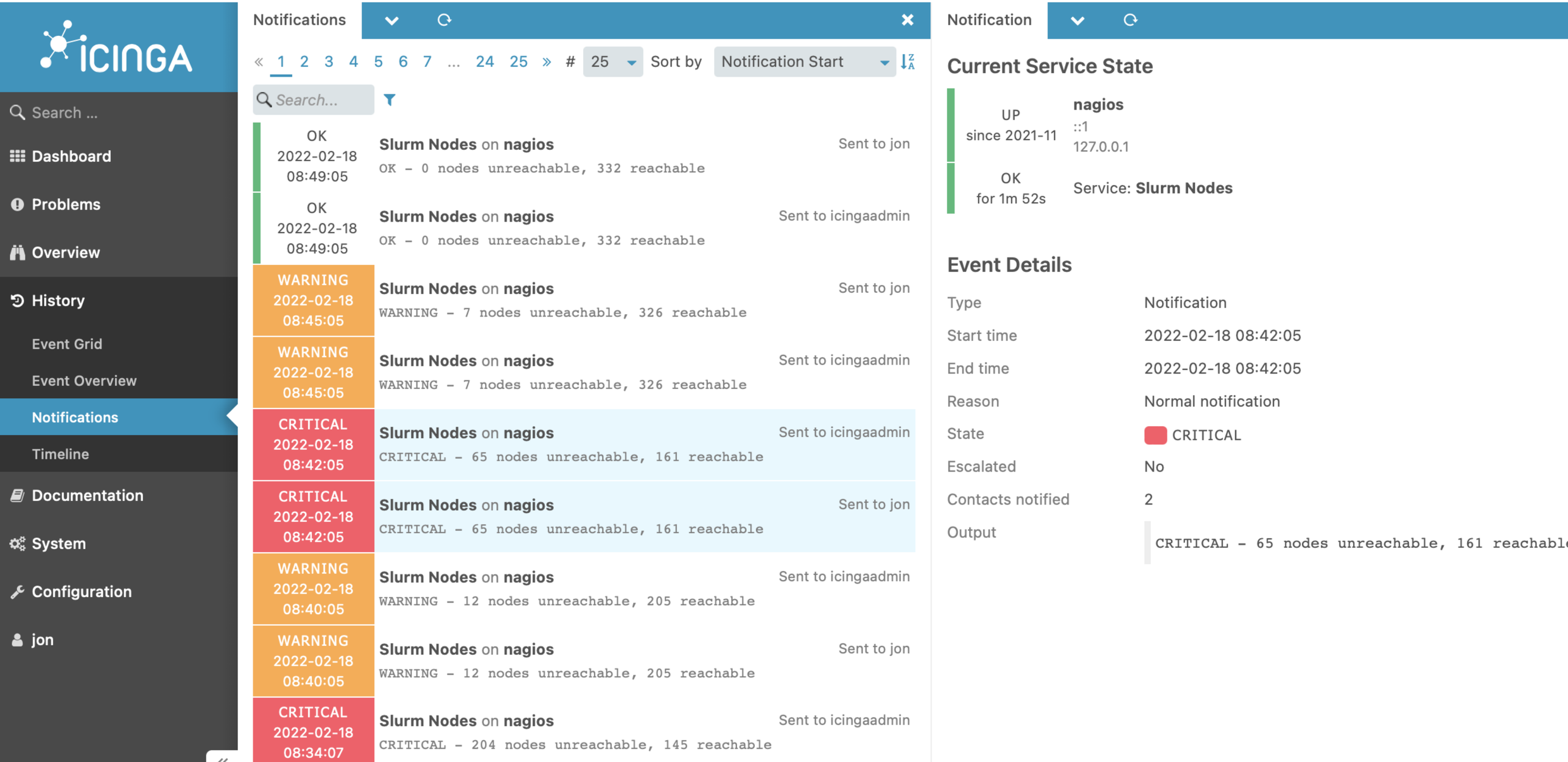
Applications

- Business transactions, conversion rate, status of 3rd party components

Monitoring services can display system status



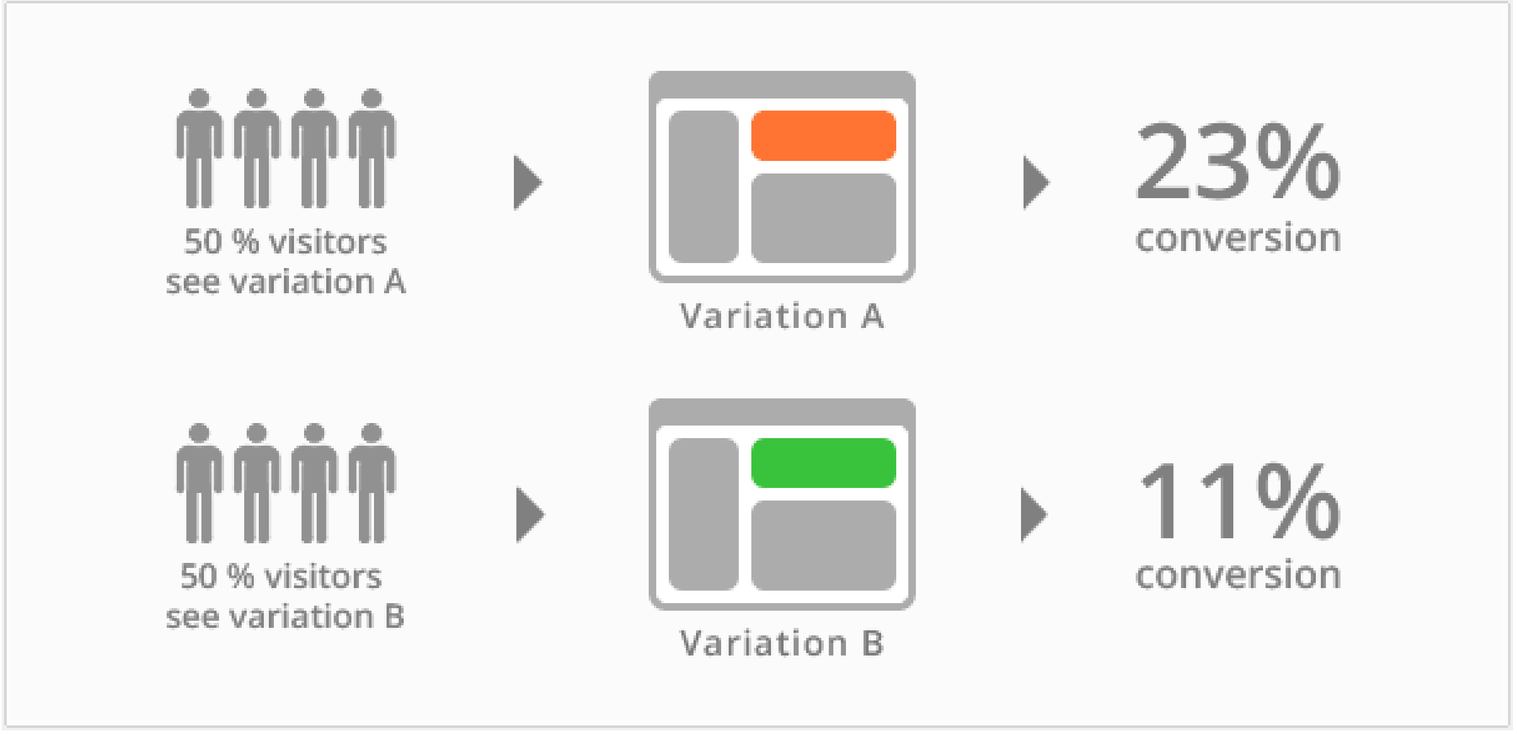
Monitoring services can take automated actions



Continuous Deployment facilitates usability testing

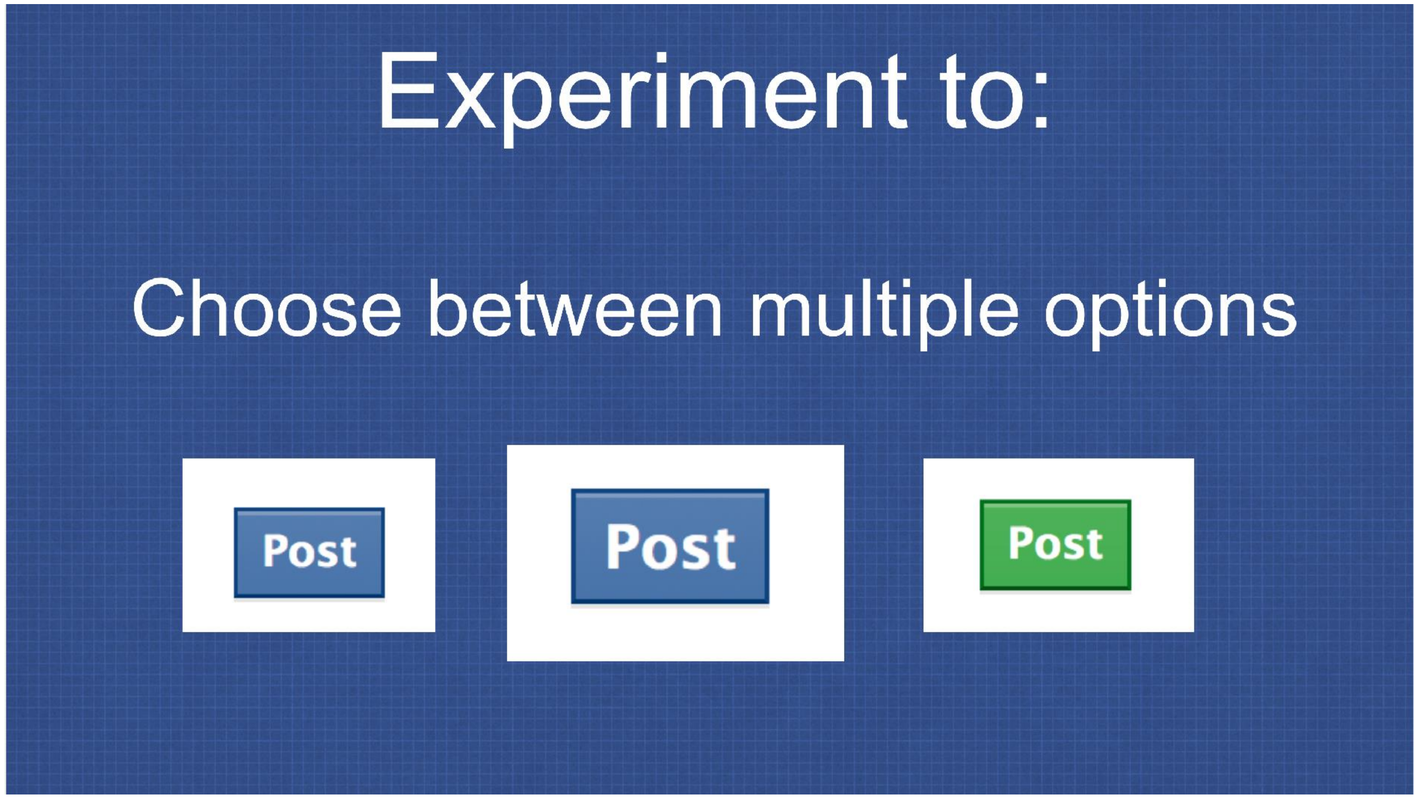
A/B Testing

- Ways to test new features for usability, popularity, performance without a focus group
- Show 50% of your site visitors version A, 50% version B, collect metrics on each, decide which is better



Usability testing in continuous development

A/B Testing: PlanOut from Facebook (“N=10^9 user study”)

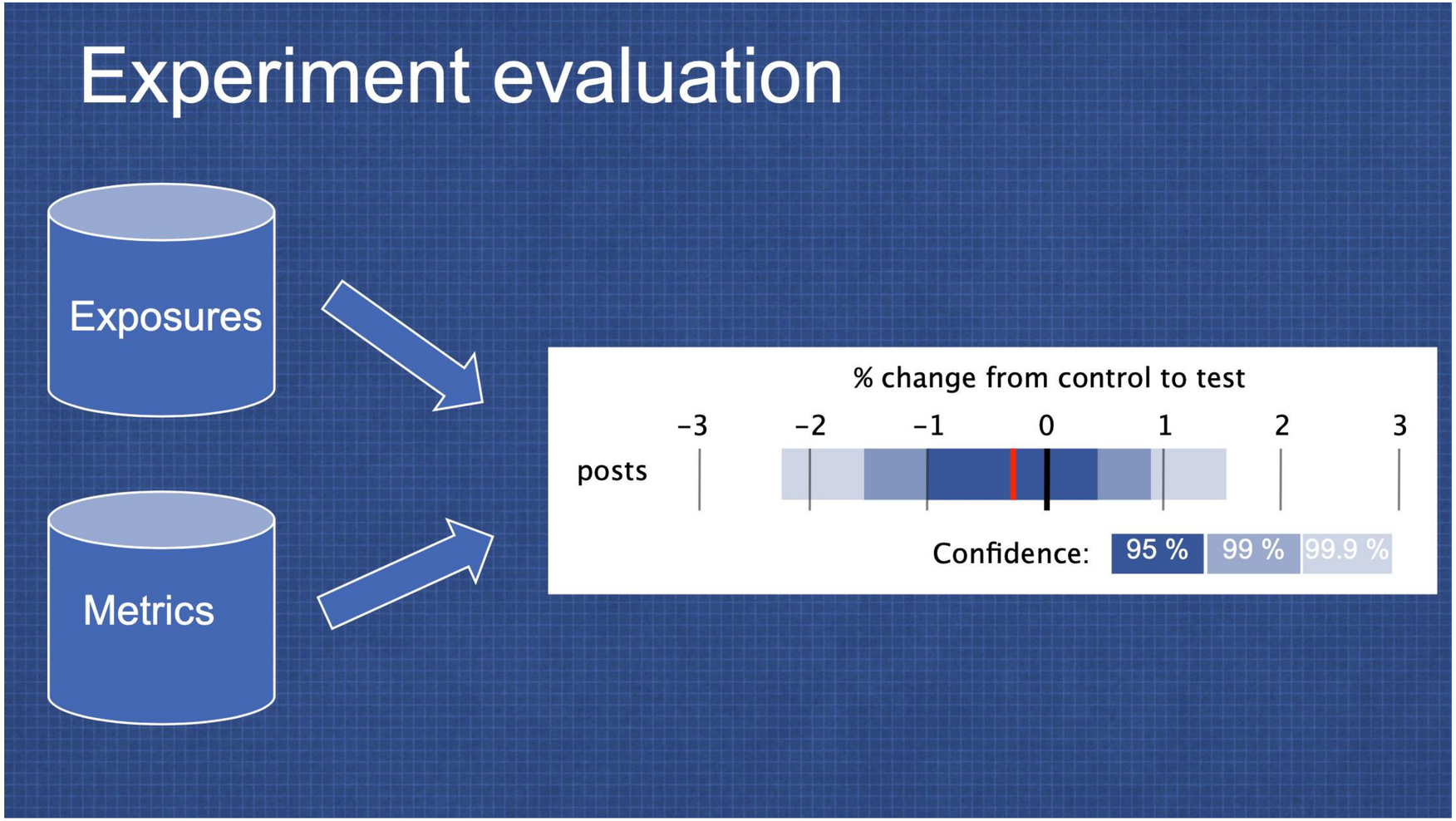


https://github.com/facebook/planout

https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Usability testing in continuous development

A/B Testing: PlanOut from Facebook (“N=10^9 user study”)



https://github.com/facebook/planout

https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Beware of Metrics

McNamara Fallacy

- Measure whatever can be easily measured
- Disregard that which cannot be measured easily
- Presume that which cannot be measured easily is not important
- Presume that which cannot be measured easily does not exist

What could Knight capital have done better?

Use capture/replay testing instead of driving market conditions in a test

Avoid including “test” code in production deployments

Automate deployments

Define and monitor risk-based KPIs

Create checklists for responding to incidents

Review

By now, you should be able to…

- Describe how continuous integration helps to catch errors sooner in the software lifecycle
- Describe the benefits of a culture of code review
- Describe strategies for performing quality-assurance on software as and after it is delivered