

CS 4530 Software Engineering

Module 14: Continuous Development Processes

Jon Bell, Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

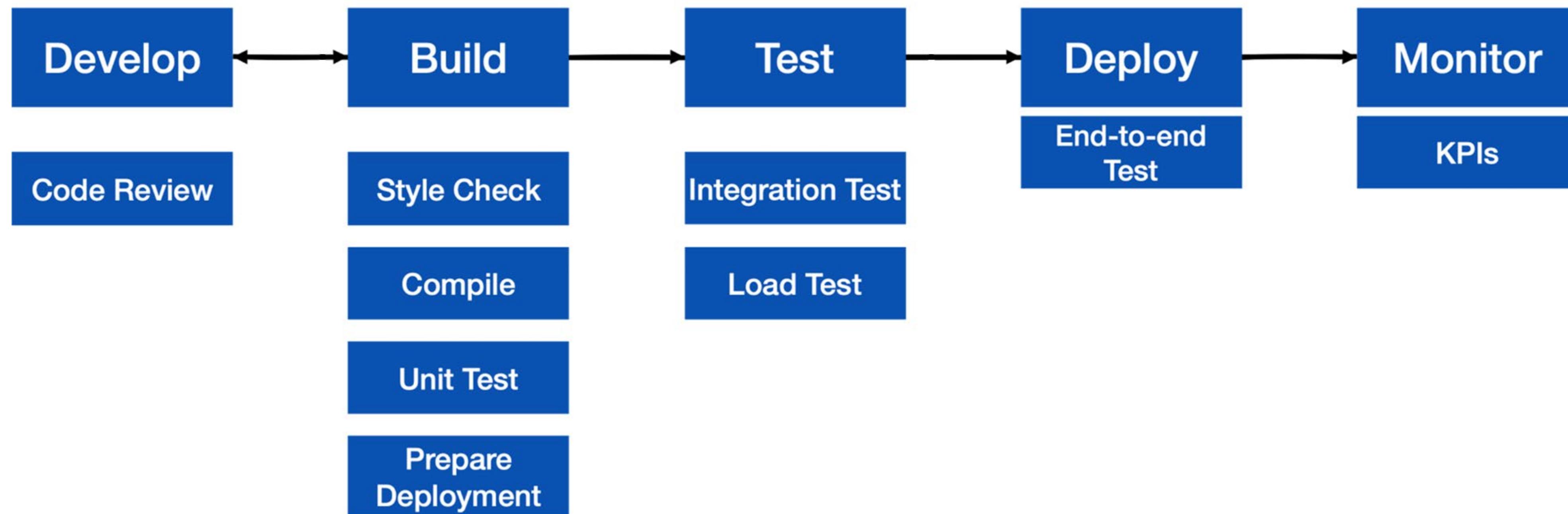
Learning objectives for this lesson

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

CD/CI improves code quality and dev velocity

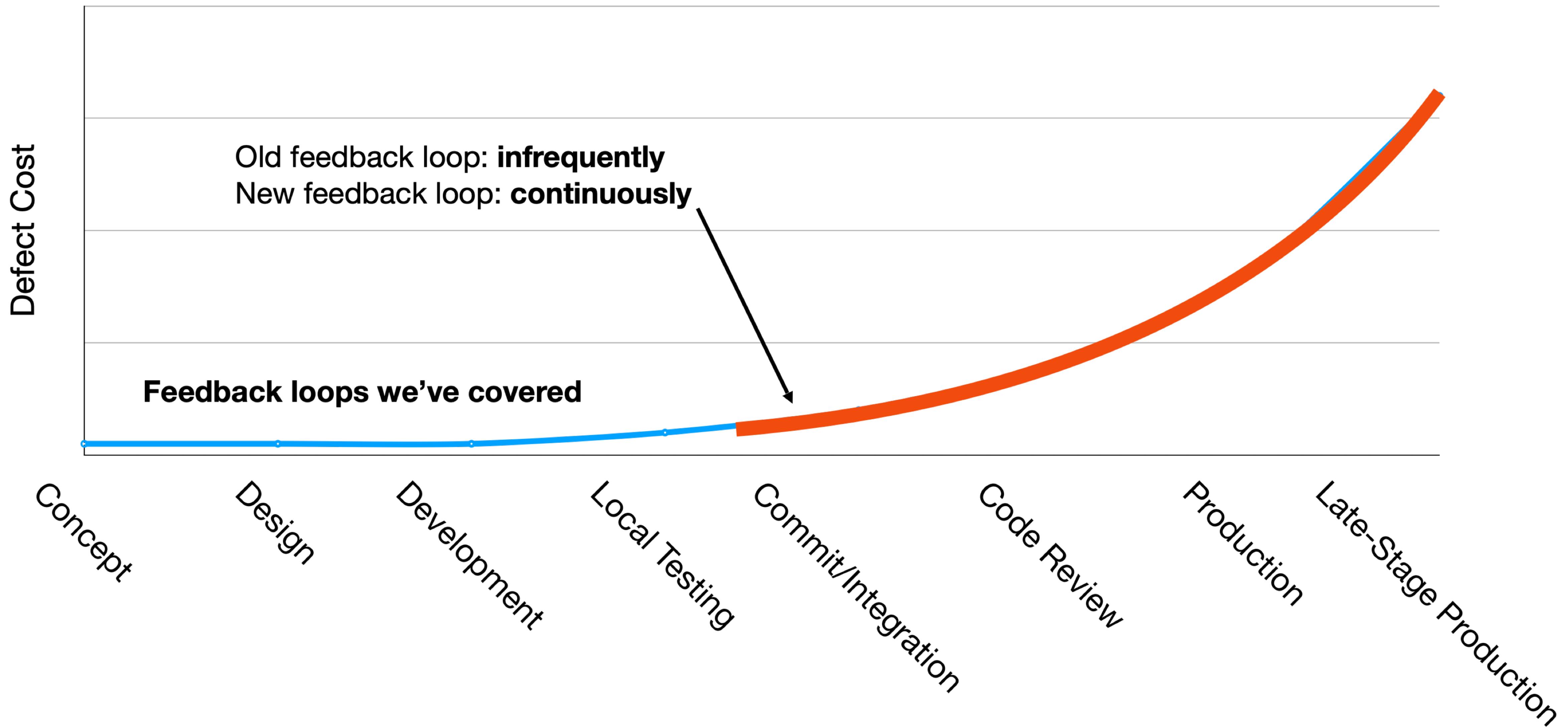
- Perform frequent integrations with entire codebase
- Run integration scale tests frequently
- Deploy frequently and monitor



Module 14.1: Continuous Integration

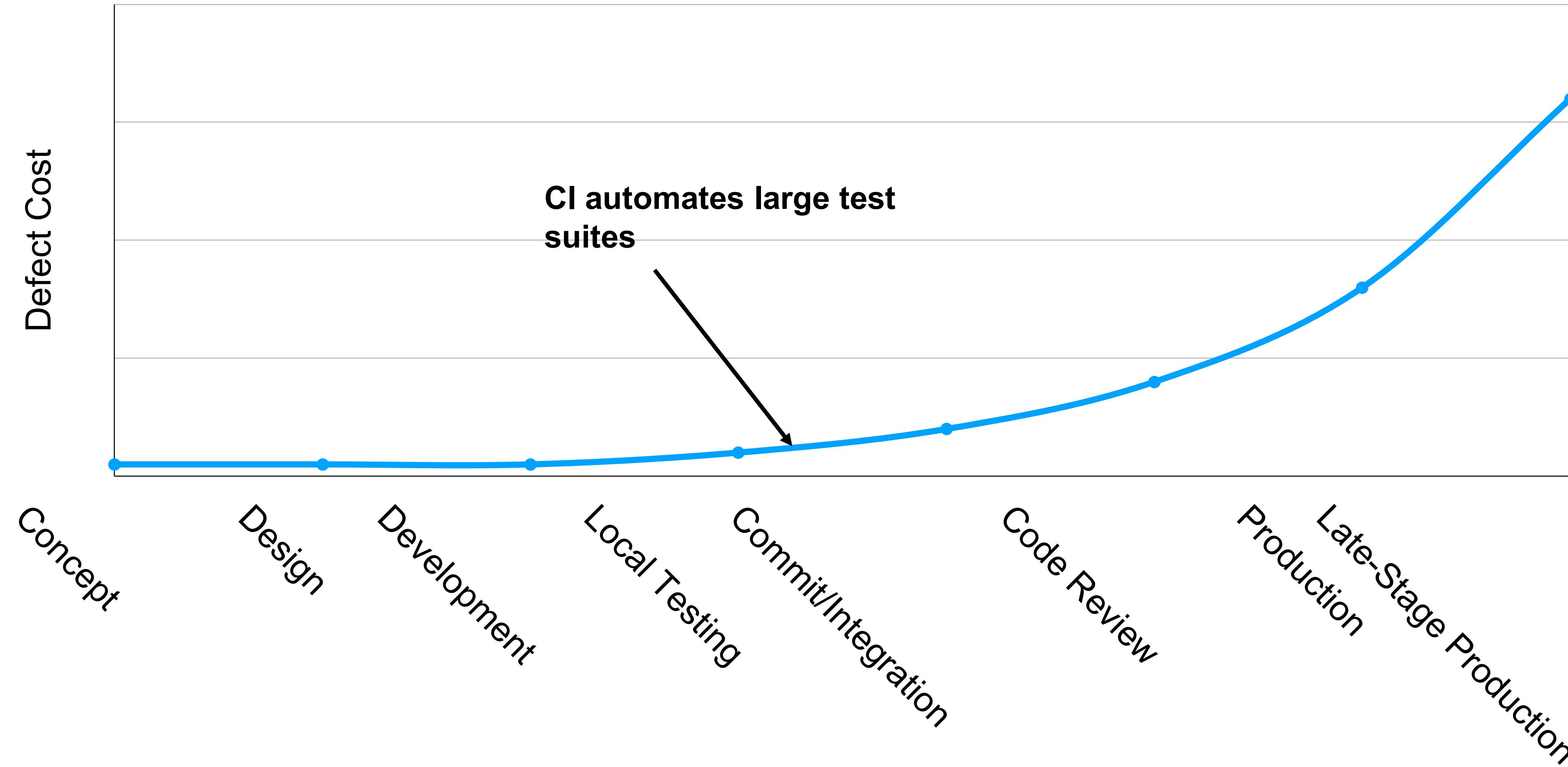
Agile values fast quality feedback loops

Faster feedback = lower cost to fix bugs



Continuous Integration automates testing

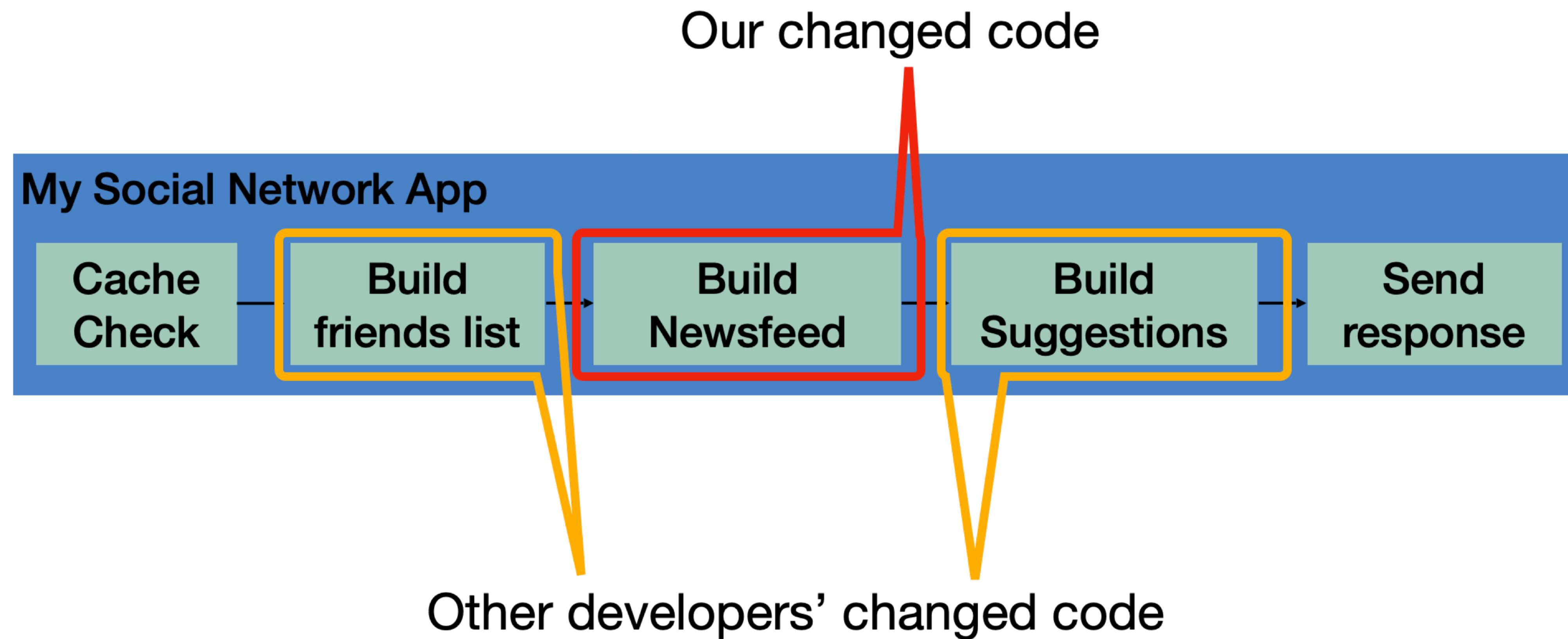
Fast feedback on integration errors



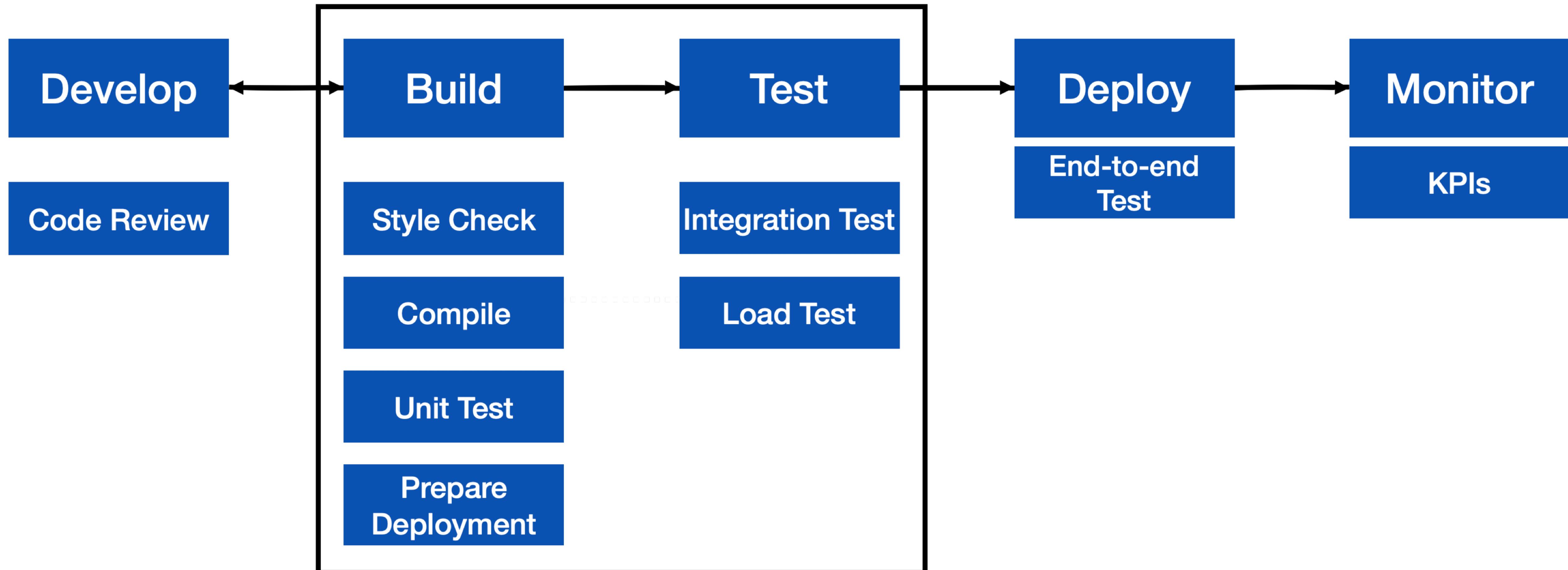
Continuous Integration

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?



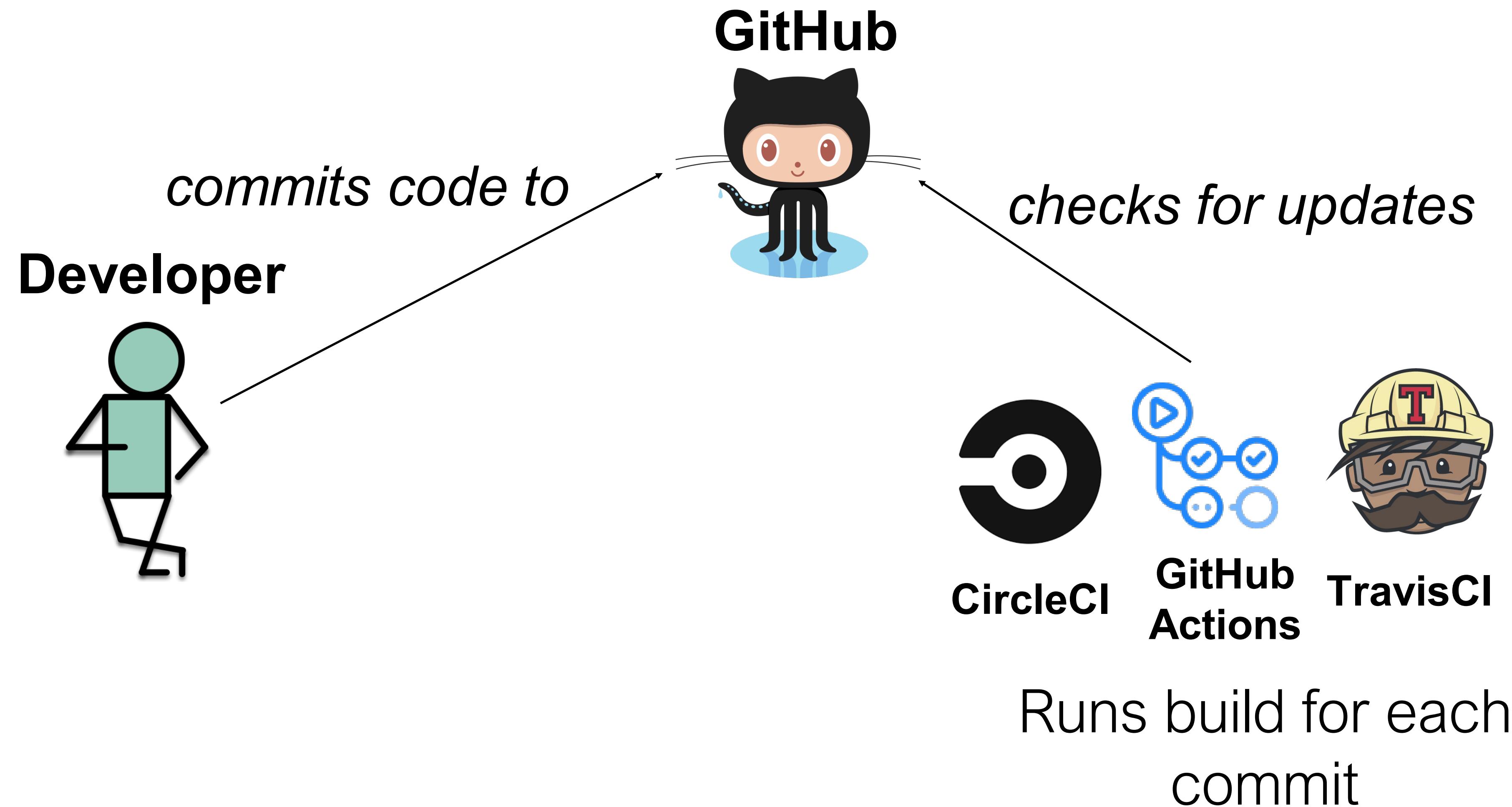
CI is a software pipeline



Automate this centrally, provide a central record of results

CI in practice

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



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

The screenshot shows a CI build status page with the following details:

- Header: ✓ Output the full test name
- Section: All checks have passed
- Count: 9 successful checks
- Items:
 - ✓ Build and Test the Grader / build (push) Successful... Details
 - ✓ Check dist/ / check-dist (push) Successful in 30s Details
 - ✓ Build and Test the Grader / test (reference) (push) ... Details
 - ✓ Build and Test the Grader / test (b) (push) Succes... Details
 - ✓ Build and Test the Grader / test (ts-ignore) (push) ... Details

The screenshot shows a GitHub pull request with the following details:

- Commit 1: Tools: extract_features.py: correct define name for AP_RPM_ENABLED by peterbarker committed 5 days ago
- Commit 2: AP_Mission: prevent use of uninitialised stack data by peterbarker committed 5 days ago (2 comments)
- Commit 3: AP_HAL_ChibiOS: disable DMA on I2C on bdshot boards to free up DMA ch... by andyp1per authored and tridge committed 6 days ago
- Commit 4: SITL: Fixed rounding lat/Ing issue when running JSBSim SITL by ShivKhanna authored and tridge committed 6 days ago
- Commit 5: AP_HAL_ChibiOS: define skyviper short board names by yuri-rage authored and tridge committed 6 days ago

CI In Practice: 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

Example CI Pipeline - TravisCI

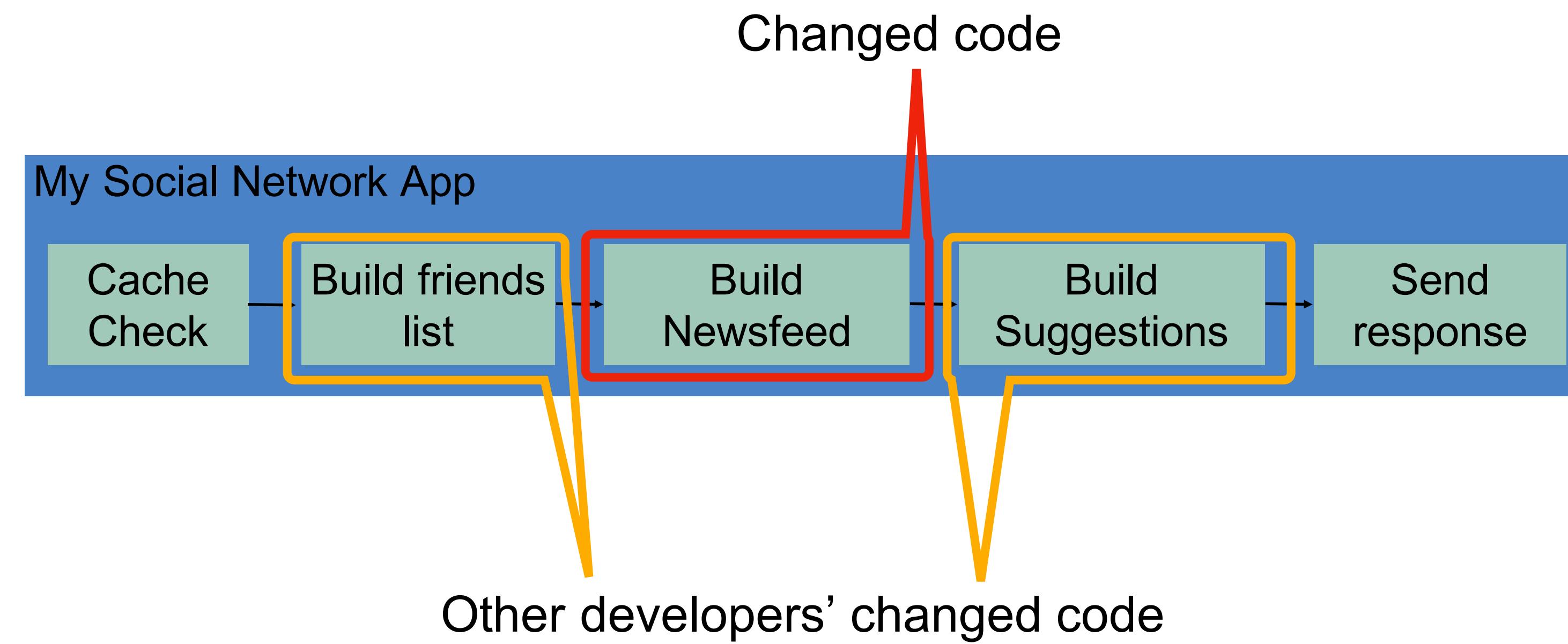
At a glance. see history of build



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

eval-10m-5x.yml

on: push

✓ evaluate / build-matrix

5s

Matrix: evaluate / run-fuzzer

✓ evaluate / run-fuzzer (... 12m 21s)

✓ evaluate / run-fuzzer ... 12m 25s

✓ evaluate / run-fuzzer ... 12m 23s

✓ evaluate / run-fuzzer (... 12m 27s)

✓ evaluate / run-fuzzer (... 12m 13s)

✓ evaluate / run-fuzzer ... 12m 24s

✓ evaluate / run-fuzzer (... 12m 21s)

✓ evaluate / run-fuzzer ... 12m 23s

✓ evaluate / run-fuzzer (... 12m 27s)

✓ evaluate / run-fuzzer (... 12m 13s)

✓ evaluate / run-fuzzer ... 12m 24s

✓ evaluate / run-fuzzer ... 12m 25s

✓ evaluate / run-fuzzer ... 12m 26s

✓ evaluate / run-fuzzer ... 12m 26s

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

eval-24h-20x.yml

on: workflow_dispatch

✓ evaluate / build-matrix

2s

Matrix: evaluate / run-fuzzer

✓ evaluate / run-fuzzer (an... 1d 0h)

✓ evaluate / run-fuzzer (bc... 1d 0h)

✓ evaluate / run-fuzzer (cl... 1d 0h)

✓ evaluate / run-fuzzer (m... 1d 0h)

✓ evaluate / run-fuzzer (rh... 1d 0h)

✓ evaluate / run-fuzzer (an... 1d 0h)

✓ evaluate / run-fuzzer (bc... 1d 0h)

✓ evaluate / run-fuzzer (cl... 1d 0h)

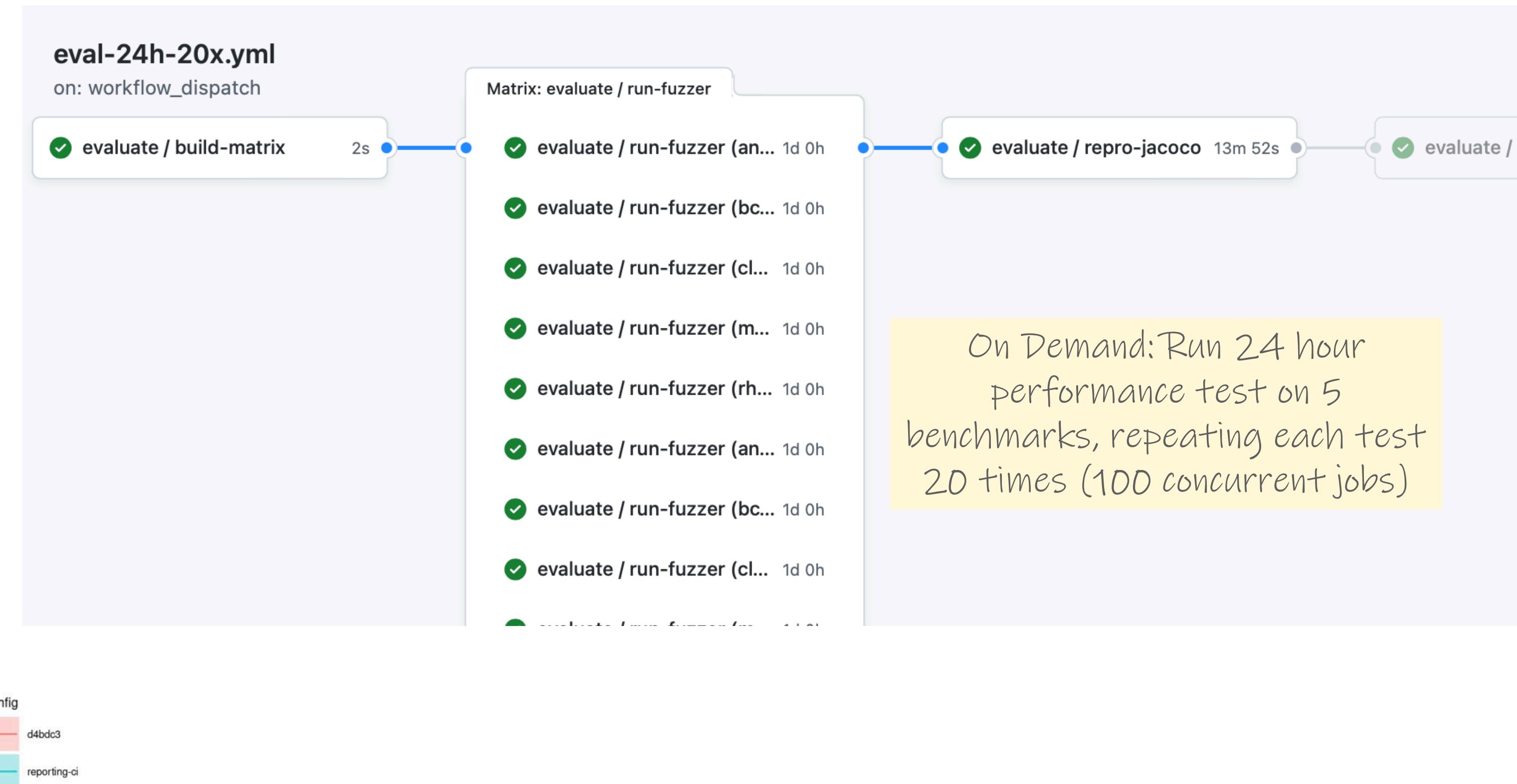
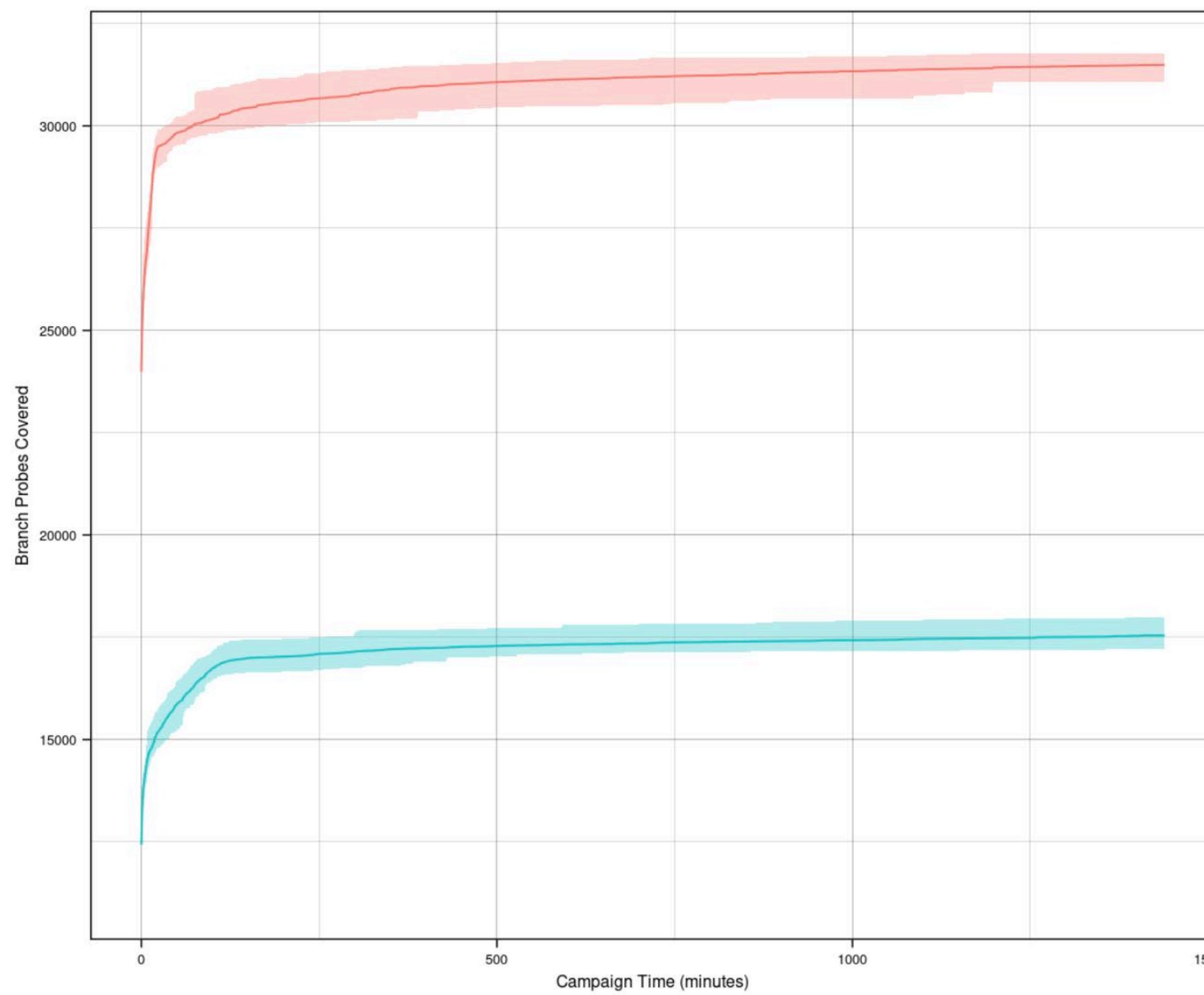
On Demand: Run 24 hour performance test on 5 benchmarks, repeating each test 20 times (100 concurrent jobs)

<https://github.com/neuse/CONFETTI/actions>

CI Pipelines automate benchmarking

closure

Branch Probes Over Time



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

Download this graph as PDF

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

CI in practice

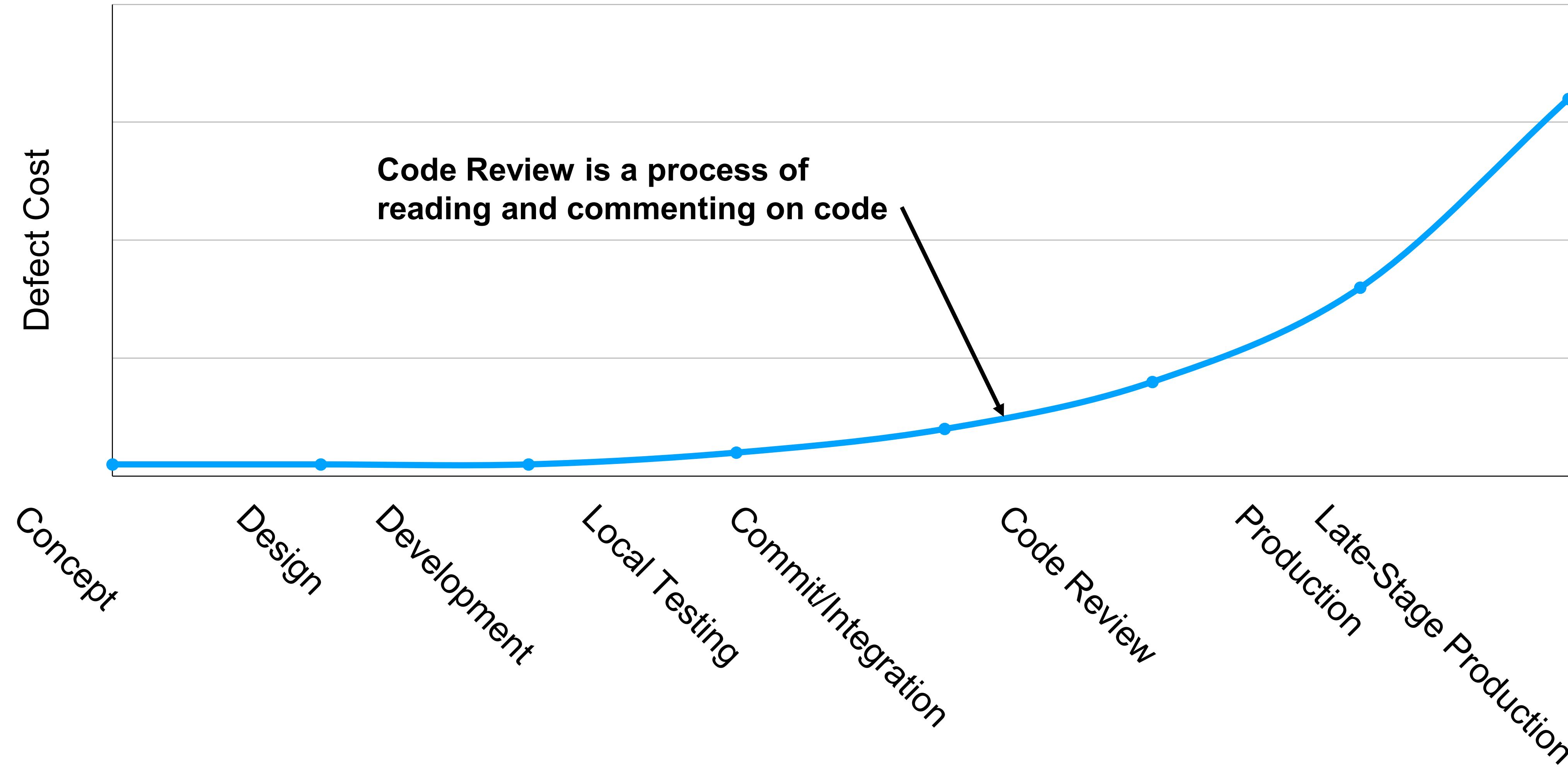
Large scale example: Google TAP

- 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

Module 14.2: Code Review

Code review is an important step of CD

Rough estimate



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

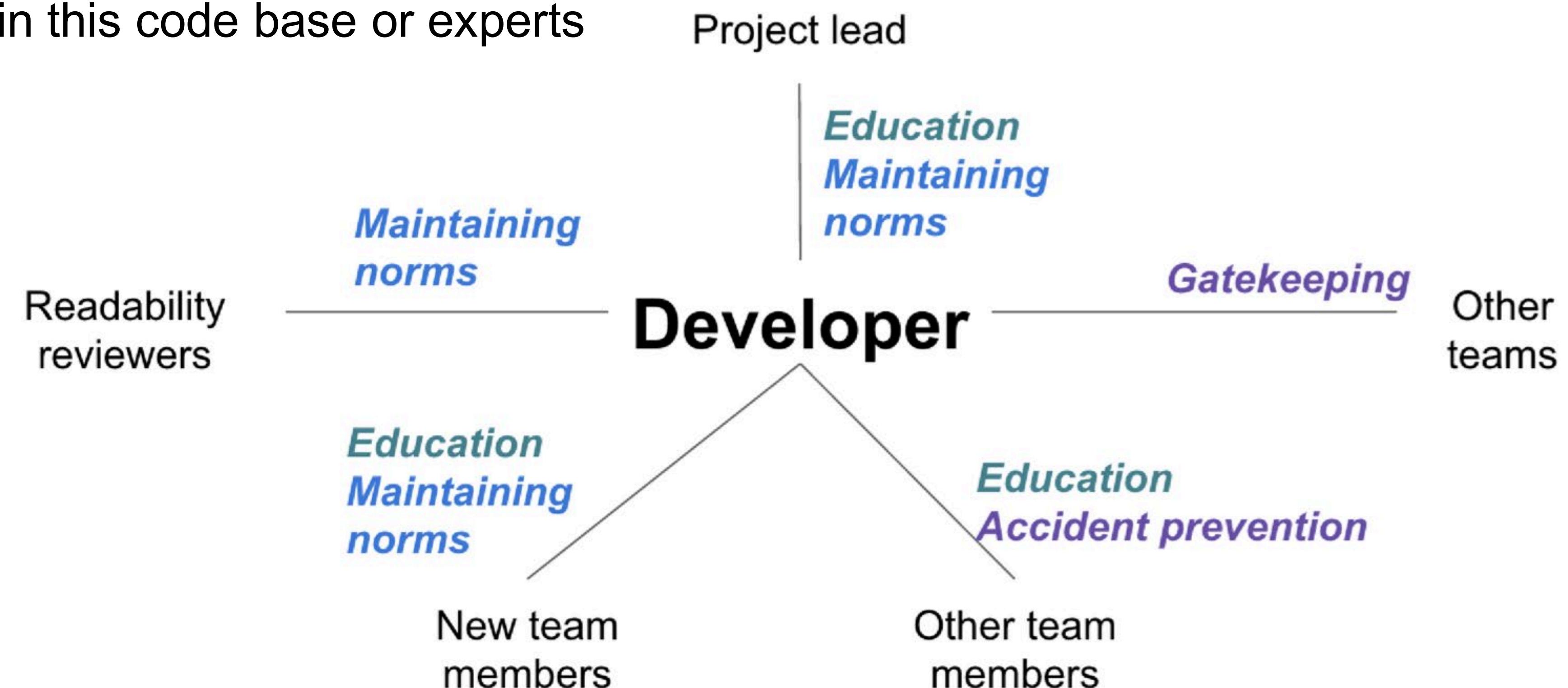
Code reviews improve quality of code base:

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

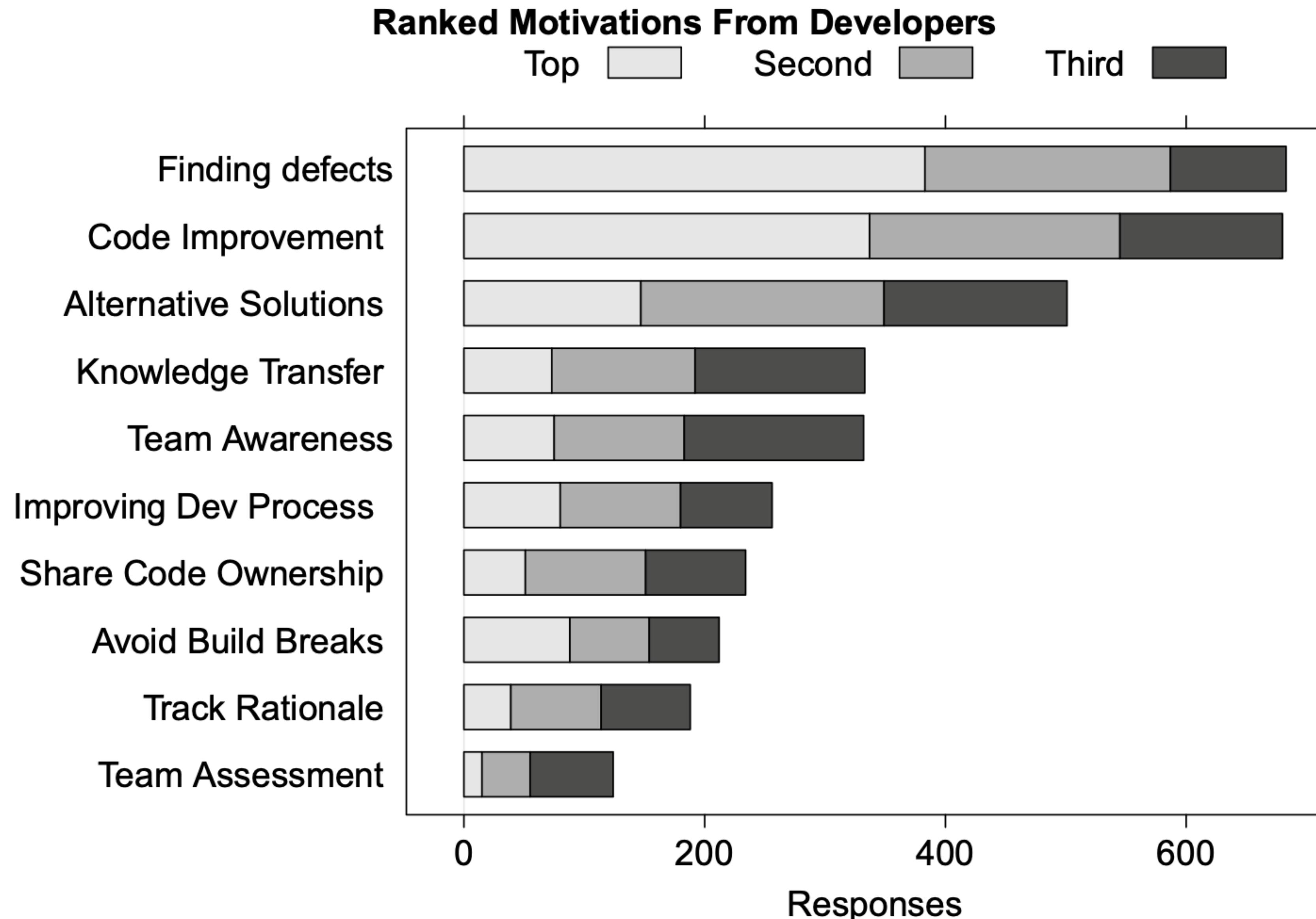
Many stakeholders can benefit from code review

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



Code reviews have many benefits



Code reviews Descend from 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).

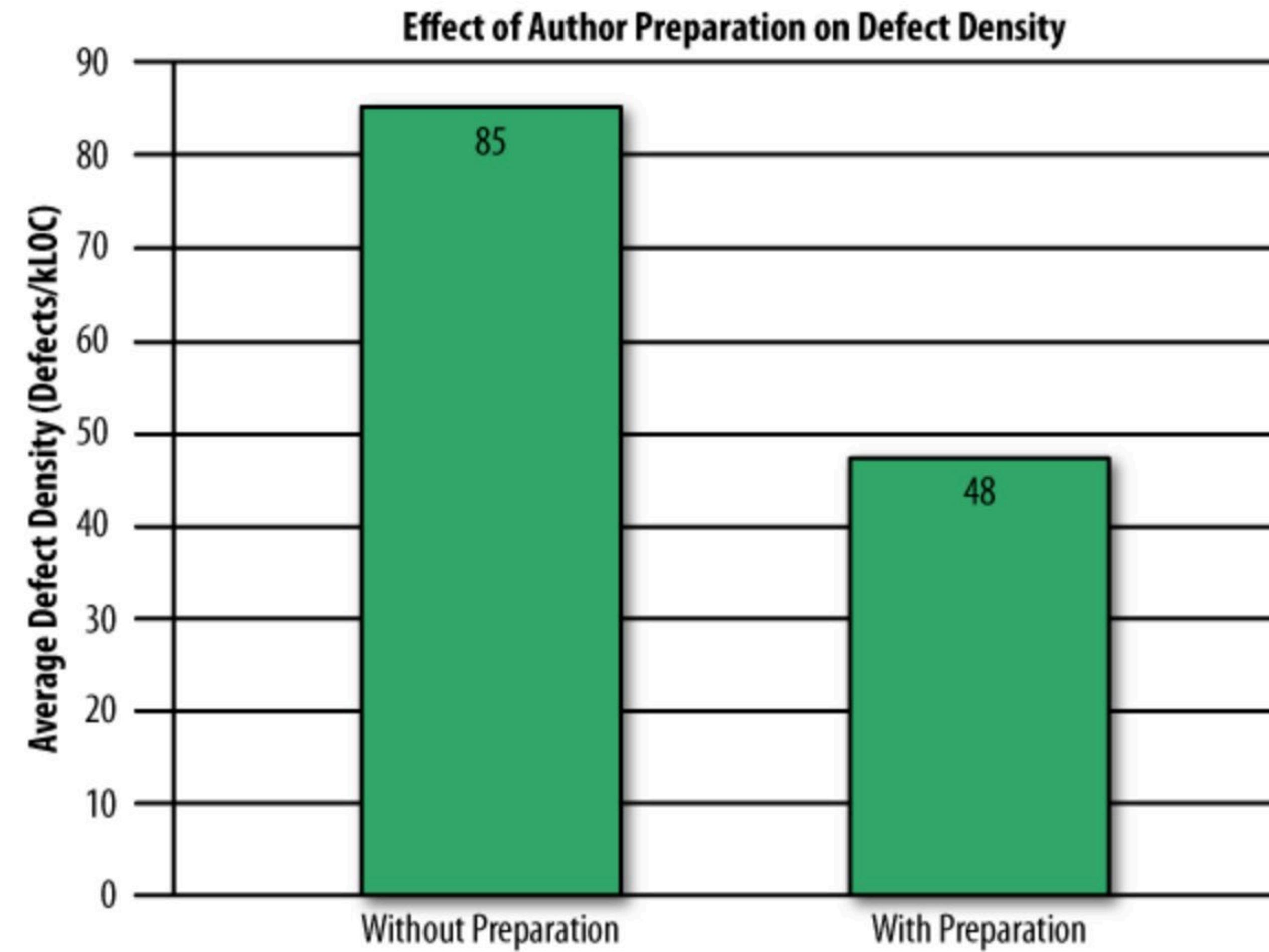
Don’t rehash design arguments (defer to author).

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

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:

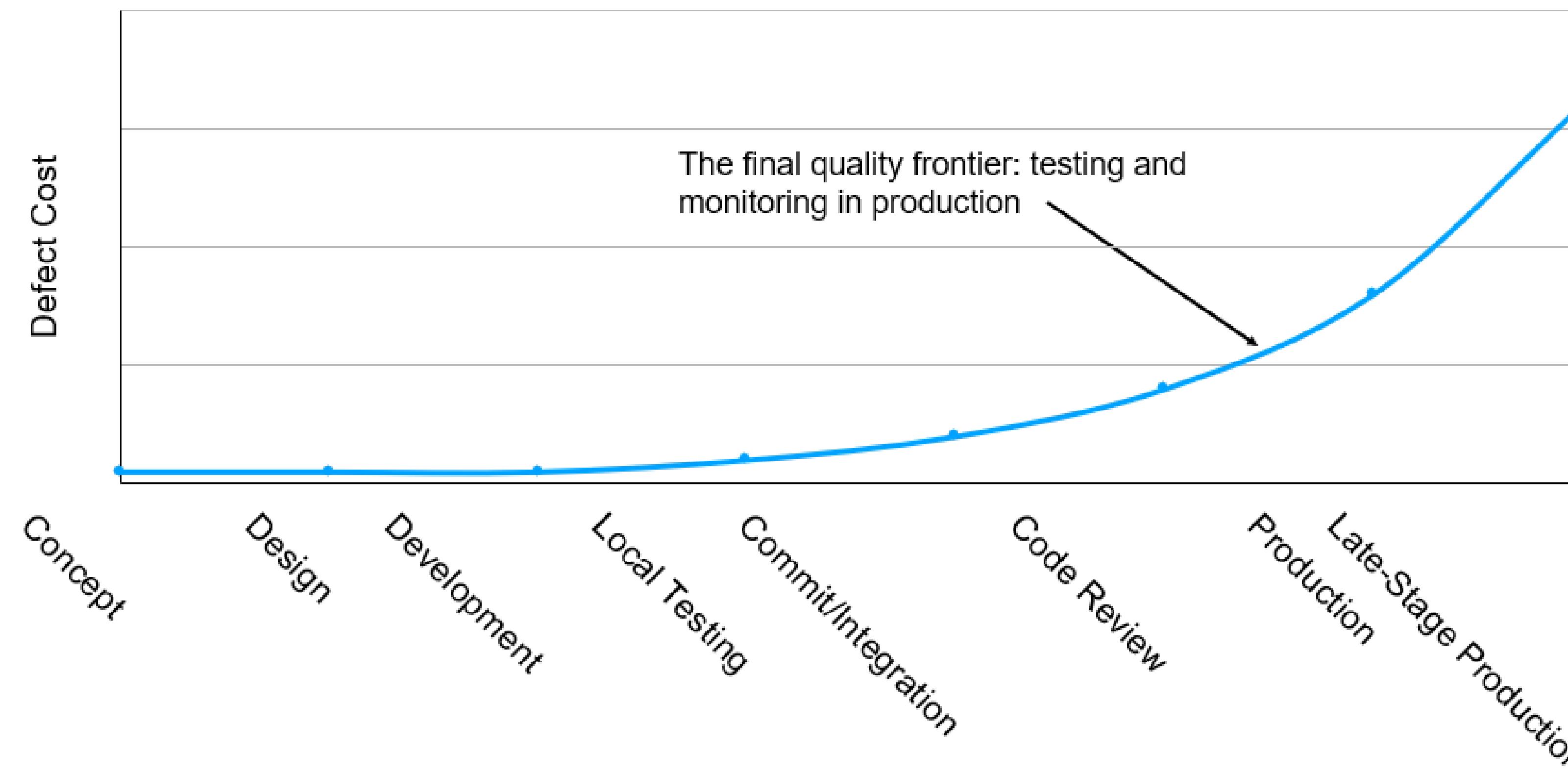
- Code review means someone's looking over your work
- You might have some attachment to it
- Criticisms: sometimes hard not to take personally
- Acknowledge a criticism and move on
- Acknowledgment doesn't imply that the author agrees with the content of the criticism
- The review is not about you,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

Module 14.3: Continuous Deployment

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

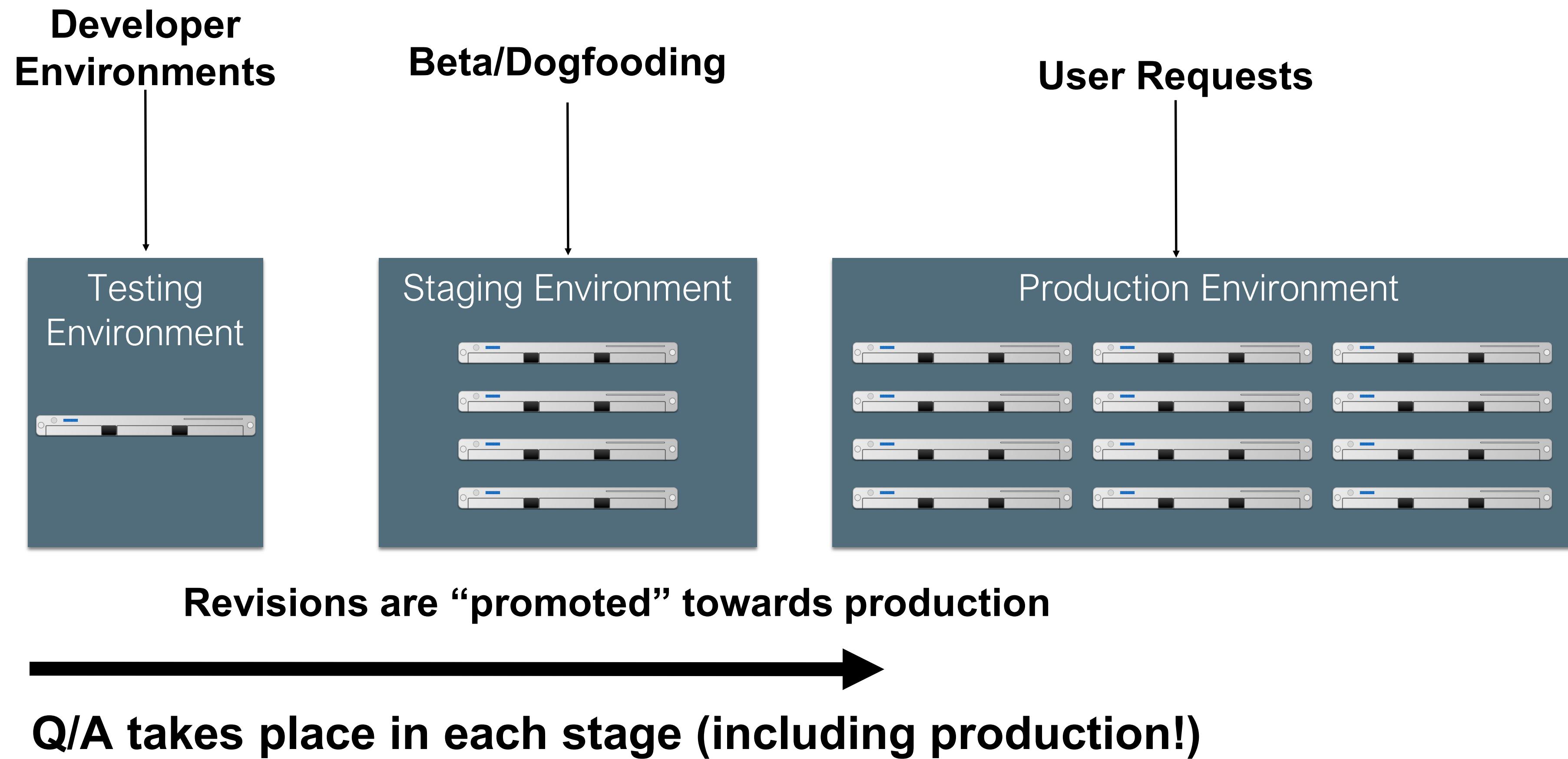
Examples:

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

Lower risk if a problem occurs in staging than in production

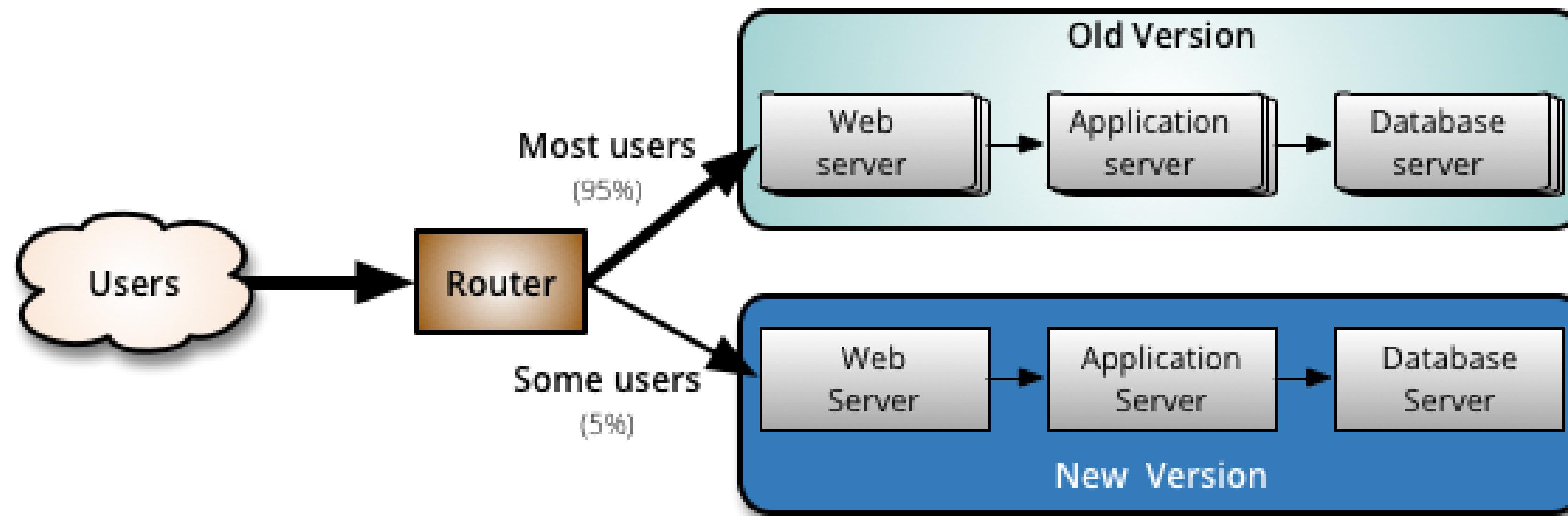
Test-Stage-Production

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

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	

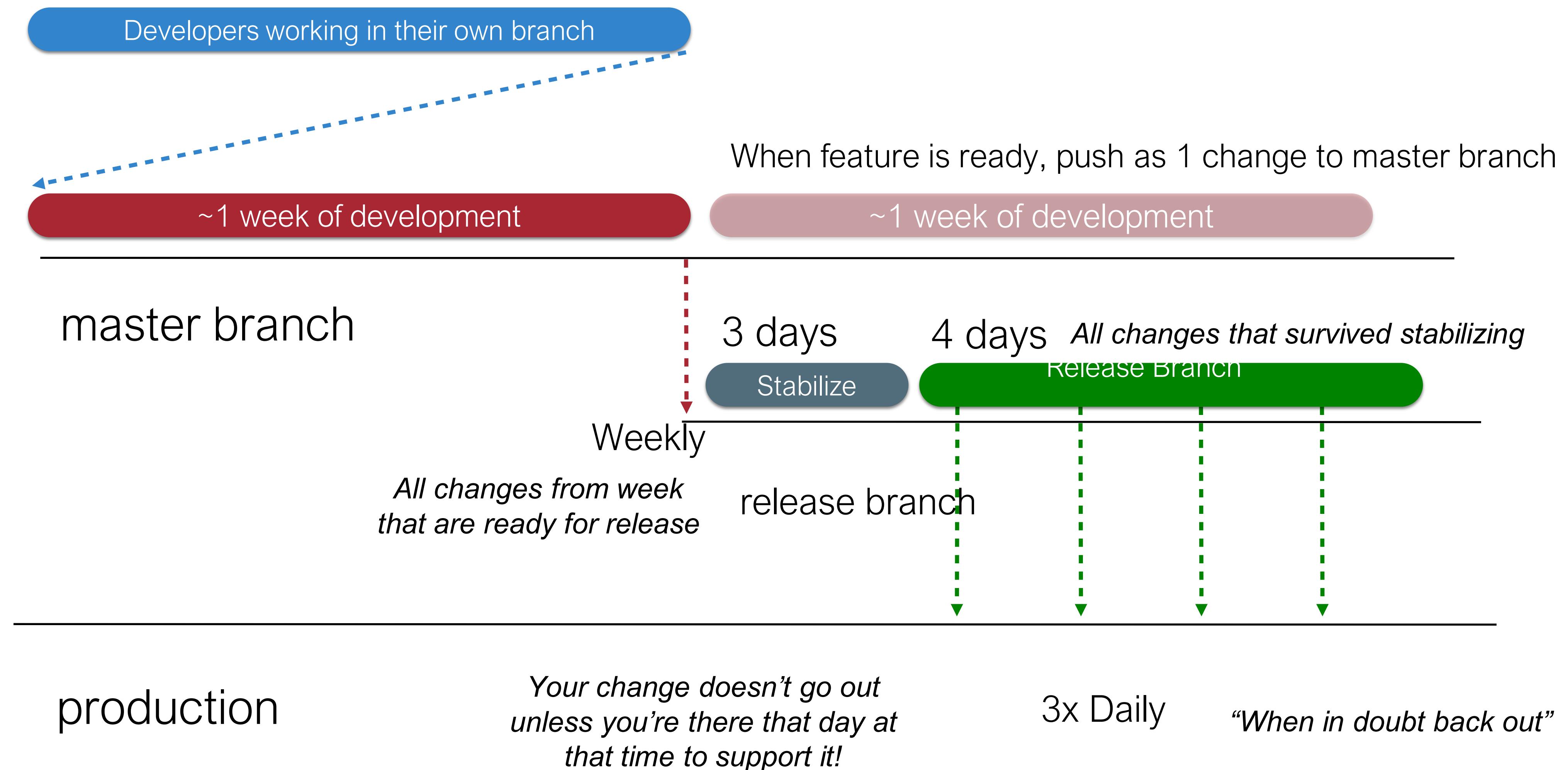
Release Pipelines

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

Deployment Example: Facebook.com

Pre-2016



Deployment Example

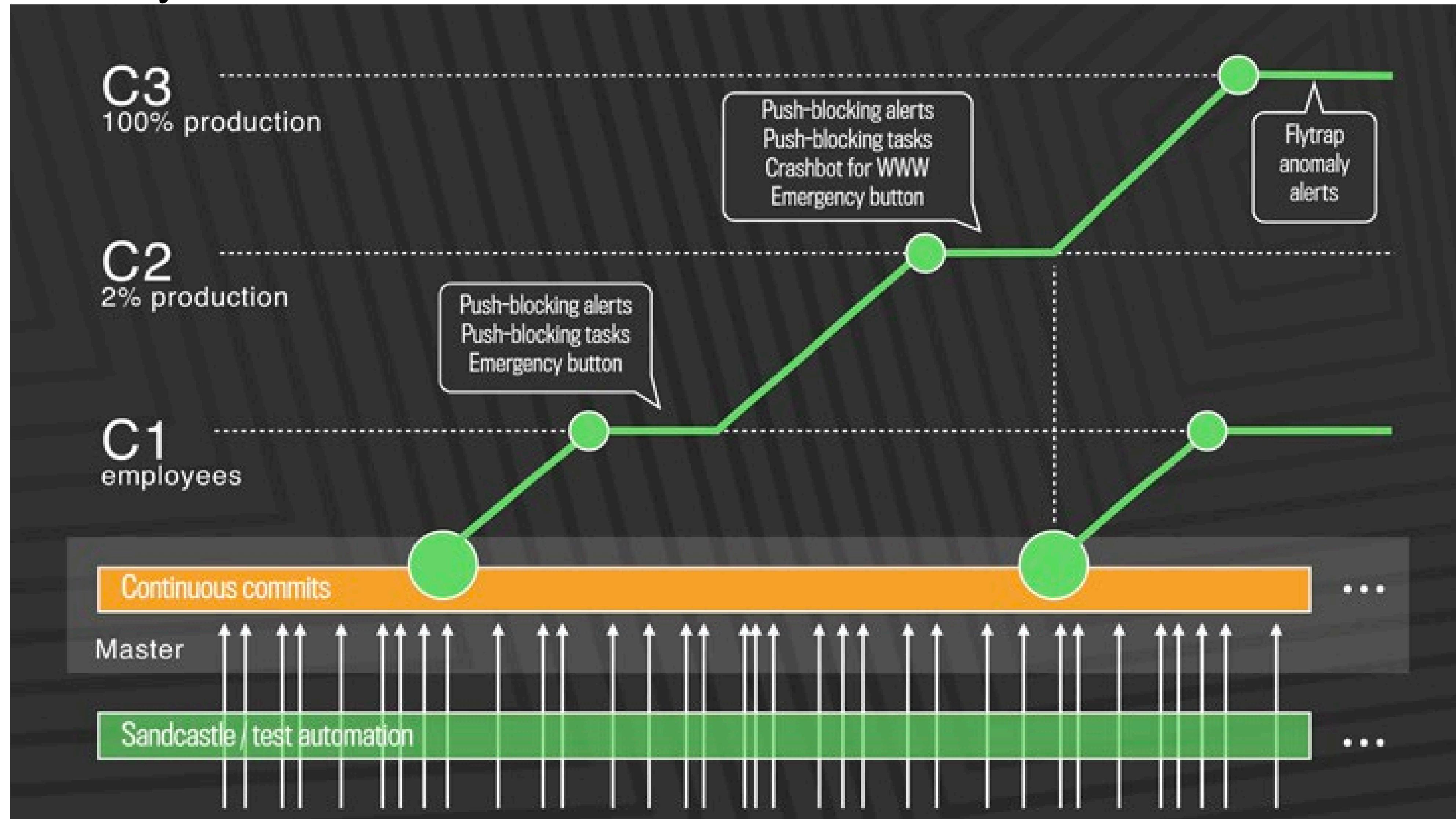
Chuck Rossi, Director Software Infrastructure & Release Engineering @ Facebook



“Our main goal was to make sure that the new system made people’s experience better — or at least, didn’t make it worse. After a year of planning and development, over the course of three days **we enabled 100% of our production web servers to run code deployed directly from master”**

Deployment Example

Post-2016: Truly continuous releases from master branch



Monitoring helps gather insights

The last step in continuous deployment: track metrics

Hardware

- Voltages, temperatures, fan speeds, component health

OS

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

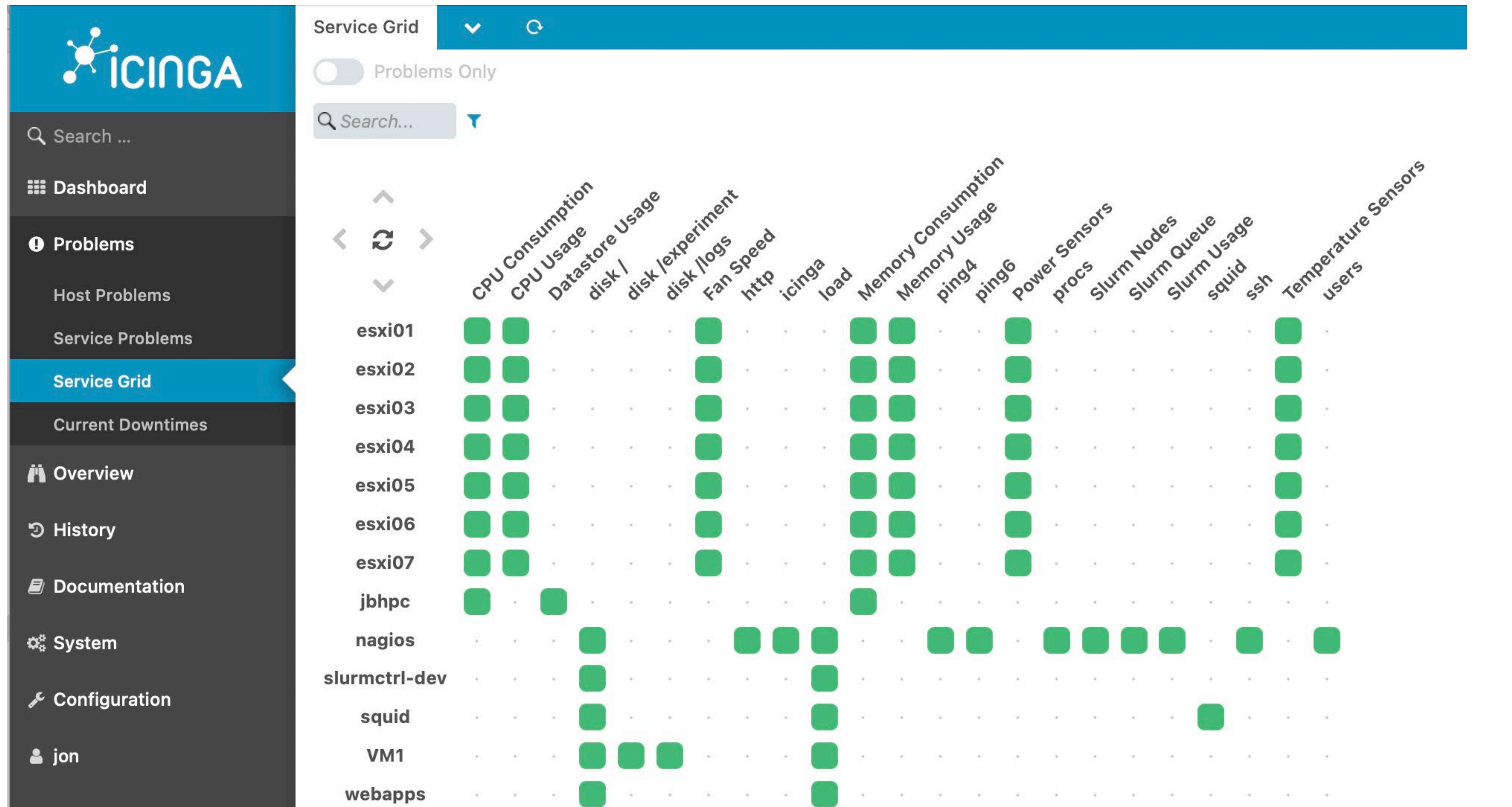
Middleware

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

Applications

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

Monitoring services aggregate system status



Monitoring services take automated actions

ICINGA

Search ...

Dashboard

Problems

Overview

History

Event Grid

Event Overview

Notifications

Timeline

Documentation

System

Configuration

jon

Notifications

Notification Start

OK
2022-02-18
08:49:05

Slurm Nodes on nagios

OK - 0 nodes unreachable, 332 reachable

OK
2022-02-18
08:49:05

Slurm Nodes on nagios

OK - 0 nodes unreachable, 332 reachable

WARNING
2022-02-18
08:45:05

Slurm Nodes on nagios

WARNING - 7 nodes unreachable, 326 reachable

WARNING
2022-02-18
08:45:05

Slurm Nodes on nagios

WARNING - 7 nodes unreachable, 326 reachable

CRITICAL
2022-02-18
08:42:05

Slurm Nodes on nagios

CRITICAL - 65 nodes unreachable, 161 reachable

CRITICAL
2022-02-18
08:42:05

Slurm Nodes on nagios

CRITICAL - 65 nodes unreachable, 161 reachable

WARNING
2022-02-18
08:40:05

Slurm Nodes on nagios

WARNING - 12 nodes unreachable, 205 reachable

WARNING
2022-02-18
08:40:05

Slurm Nodes on nagios

WARNING - 12 nodes unreachable, 205 reachable

CRITICAL
2022-02-18
08:34:07

Slurm Nodes on nagios

CRITICAL - 204 nodes unreachable, 145 reachable

Sent to jon

Sent to icingaadmin

Sent to jon

Sent to icingaadmin

Sent to icingaadmin

Sent to icingaadmin

Sent to jon

Sent to icingaadmin

Sent to icingaadmin

Sent to jon

Sent to icingaadmin

Sent to icingaadmin

Sent to icingaadmin

Notification

Current Service State

UP since 2021-11-12 127.0.0.1

OK for 1m 52s Service: Slurm Nodes

Event Details

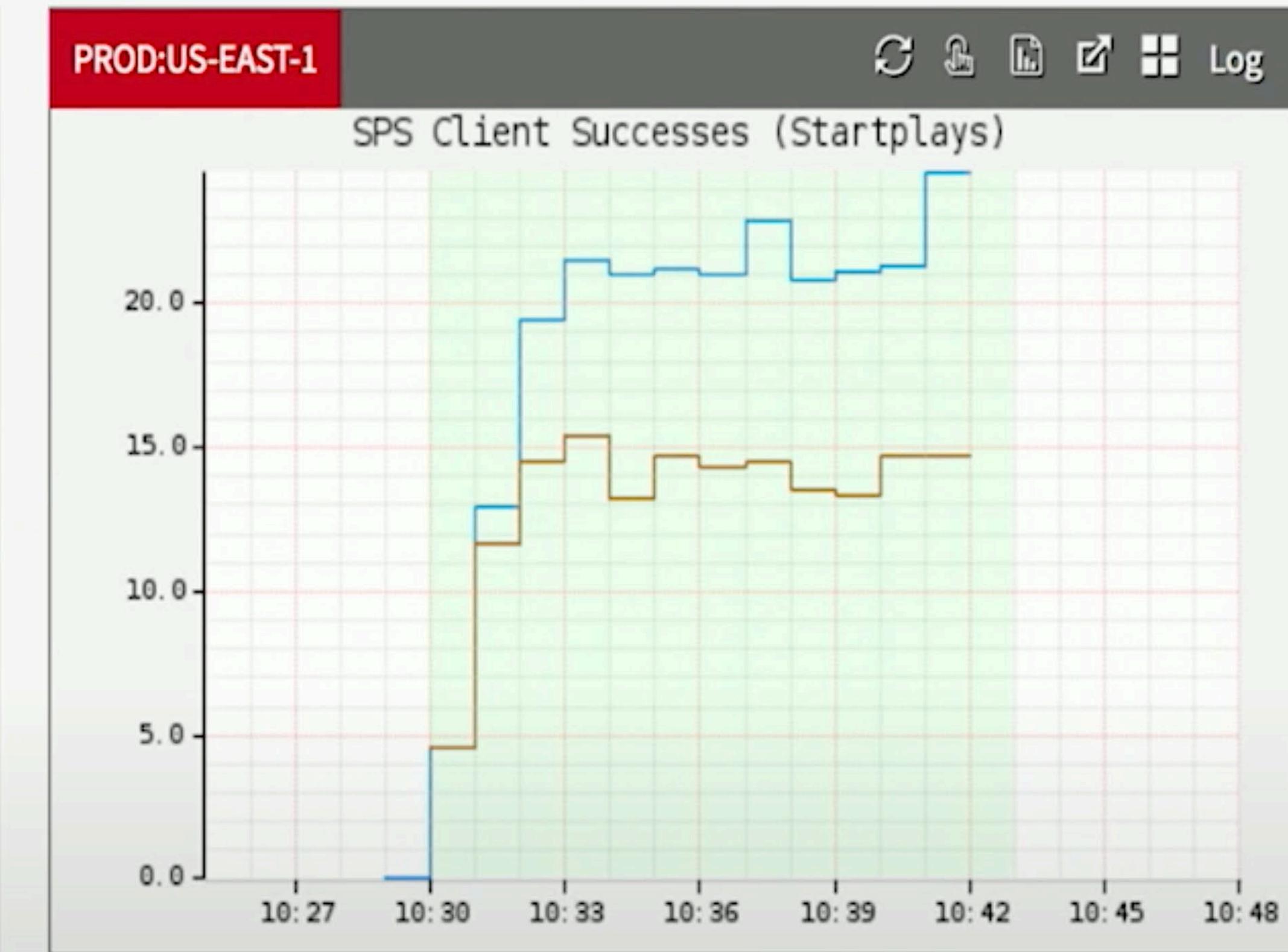
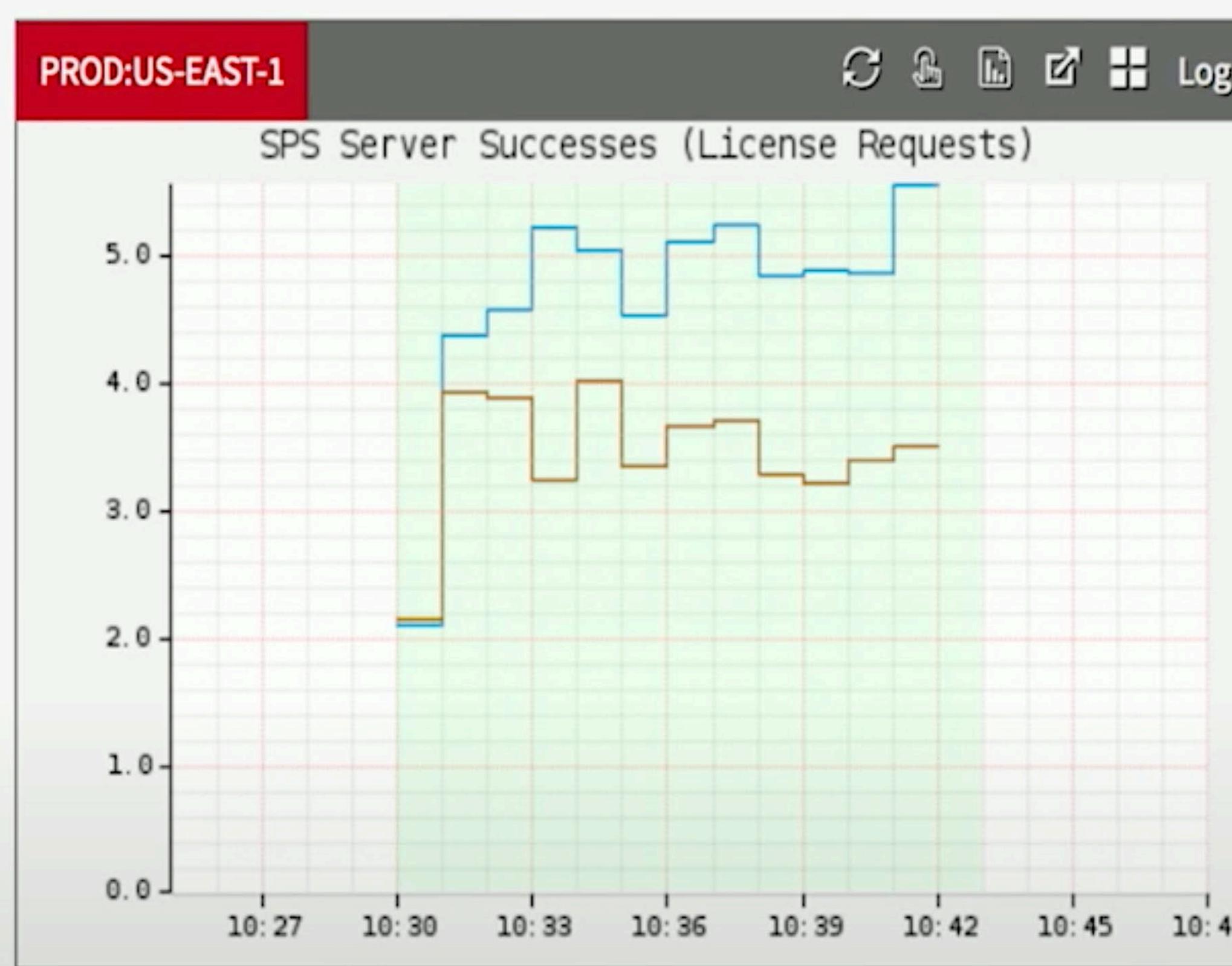
Type	Notification
Start time	2022-02-18 08:42:05
End time	2022-02-18 08:42:05
Reason	Normal notification
State	CRITICAL
Escalated	No
Contacts notified	2
Output	CRITICAL - 65 nodes unreachable, 161 reachable

Monitoring services take automated actions

Automatically detecting irregular behavior at Netflix

SPS

Legend: Experiment (Red) Control (Blue)



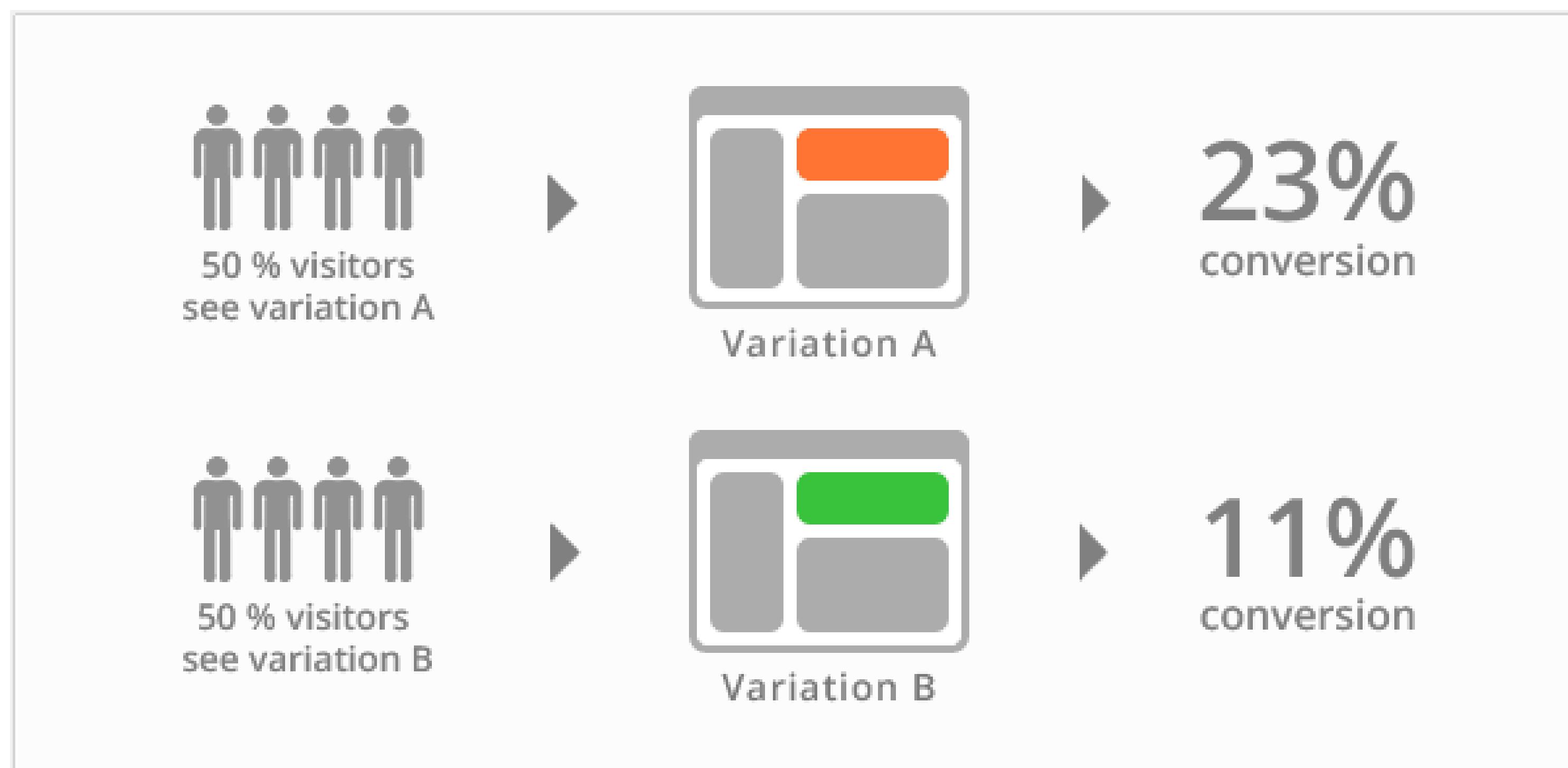
MONITORING!

<https://www.youtube.com/watch?v=qzymLij9ag>

Usability testing in continuous development

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



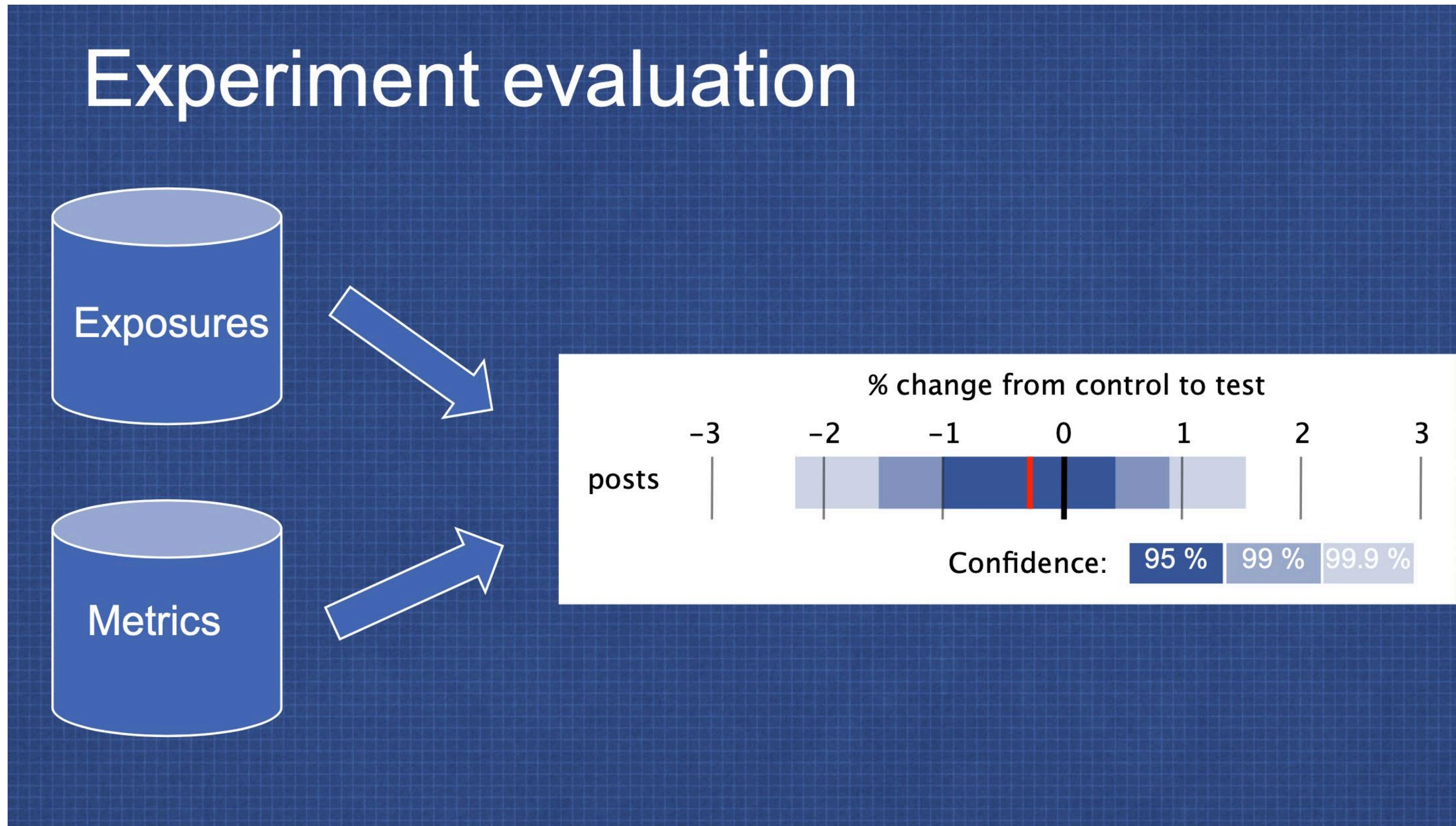
Usability testing in continuous development

A/B Testing: PlanOut from Facebook (“N=10⁹ user study”)



Usability testing in continuous development

A/B Testing: PlanOut from Facebook ("N=10⁹ user study")



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



Case study of a failed deployment

Knightmare: A DevOps Cautionary Tale

 D7  DevOps  April 17, 2014  6 Minutes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly \$400 million in assets went bankrupt in 45-minutes because of a failed deployment.



“In the week before go-live, a Knight engineer manually deployed the new RLP code in SMARS to its 8 servers. However, he made a mistake and did not copy the new code to one of the servers. Knight did not have a second engineer review the deployment, and neither was there an automated system to alert anyone to the discrepancy. “

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