

# CS 4530 Software Engineering

## Module 14: Continuous Development Processes

Jon Bell, Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

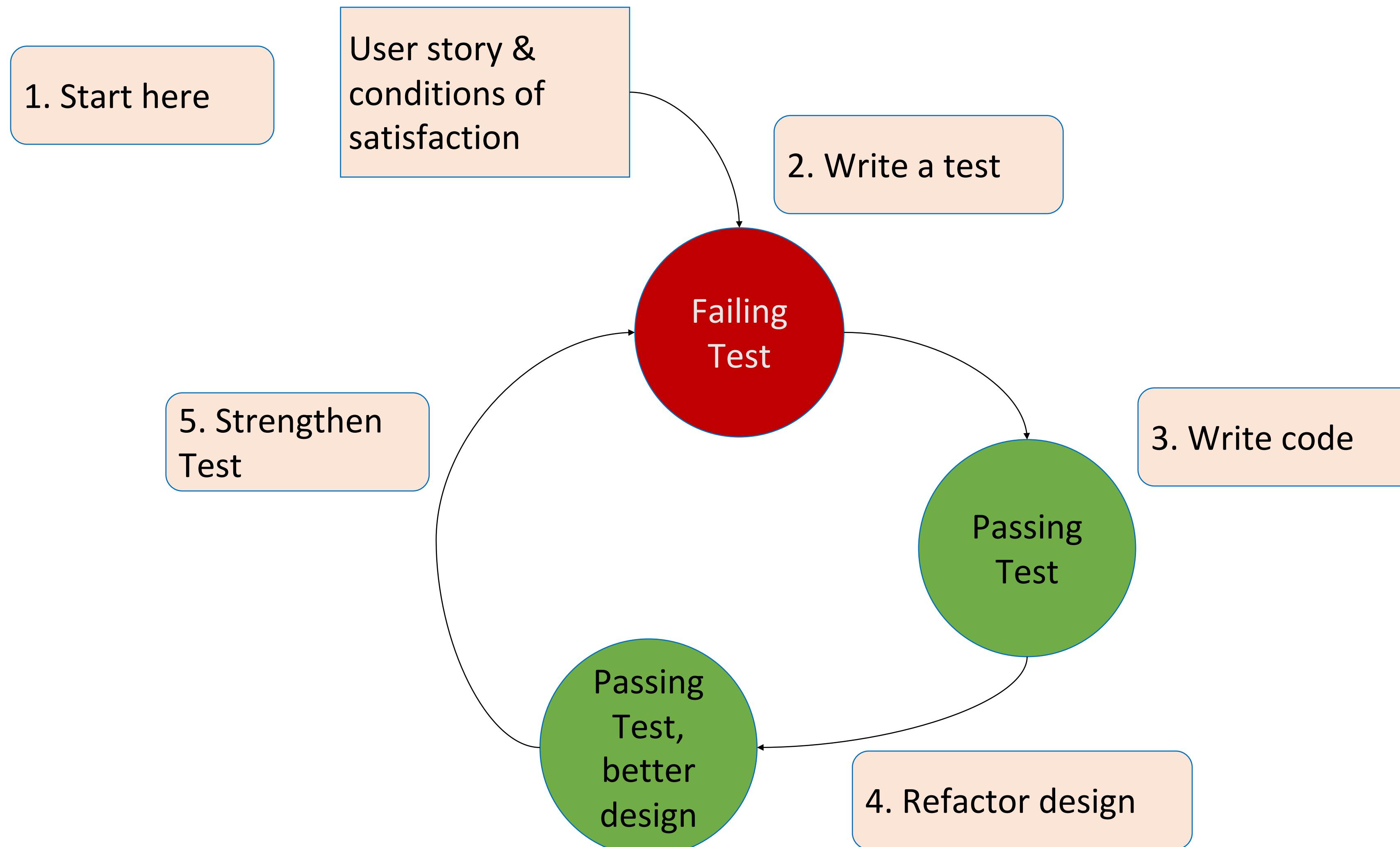
# Learning objectives for this lesson

- By the end of this lesson, you should be able to...
  - Describe how continuous development helps to catch errors sooner in the software lifecycle
  - Describe strategies for performing quality-assurance on software as and after it is delivered
  - Compare and contrast continuous delivery with test driven development as a quality assurance strategy

# Review: The Agile Model Reduces Risk by Embracing Change (~2000)

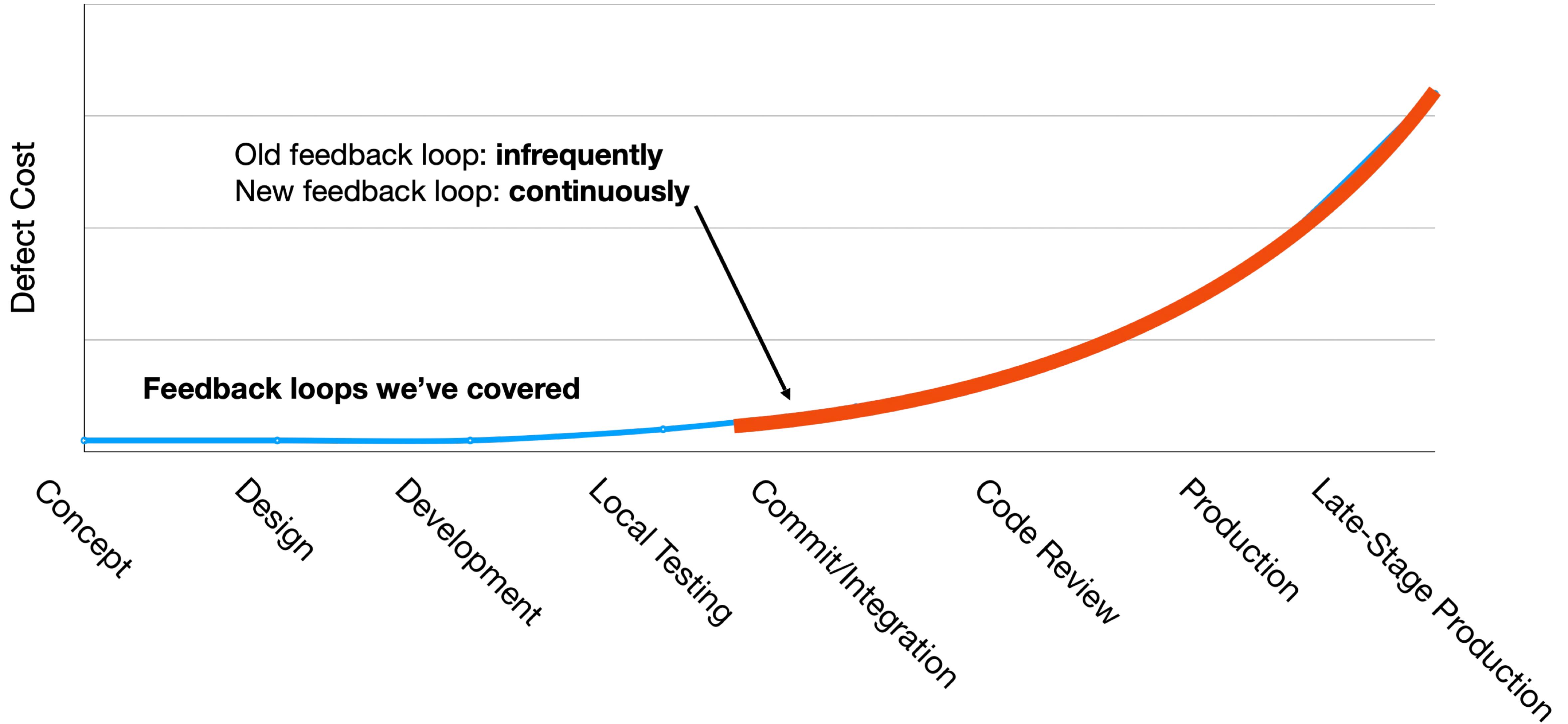
- The Waterfall philosophy:
  - "The project is too large and complex, and it will take months (or years!) to plan, so once we come up with the plan, that plan can not change"
  - Reduce risk by proceeding in stages
- The Agile philosophy:
  - The project is too large and complex, it is unlikely that we will know exactly what we need right now, and to some extent, we are inventing something new. We think that as we make it, we will figure it out as we go"
  - Reduce risk by limiting time on any one stage; then reassess. ("time-boxing")

# Review: Test Driven Development (TDD) creates fast feedback loops



# Agile values fast quality feedback loops

Faster feedback = lower cost to fix bugs



# Example: Some bugs slip through testing, even in highly-regulated industries

Aviation

## After Alaska Airlines planes bump runway while taking off from Seattle, a scramble to ‘pull the plug’

By Dominic Gates, The Seattle Times

Updated: February 20, 2023

Published: February 20, 2023

“That morning, a software bug in an update to the DynamicSource tool caused it to provide seriously undervalued weights for the airplanes.

The Alaska 737 captain said the data was on the order of 20,000 to 30,000 pounds light. With the total weight of those jets at 150,000 to 170,000 pounds, the error was enough to skew the engine thrust and speed settings.

Both planes headed down the runway with less power and at lower speed than they should have. And with the jets judged lighter than they actually were, the pilots rotated too early

Both the Max 9 and 737-900ER have long passenger cabins, which makes them more vulnerable to a tail strike when the nose comes up too soon.” ...



Photo: saiters\_photography (IG, different plane/airport)

... “A quick interim fix proved easy: When operations staff turned off the automatic uplink of the data to the aircraft and switched to manual requests “we didn’t have the bug anymore.”

Peyton said his team also checked the integrity of the calculation itself before lifting the stoppage. All that was accomplished in 20 minutes.

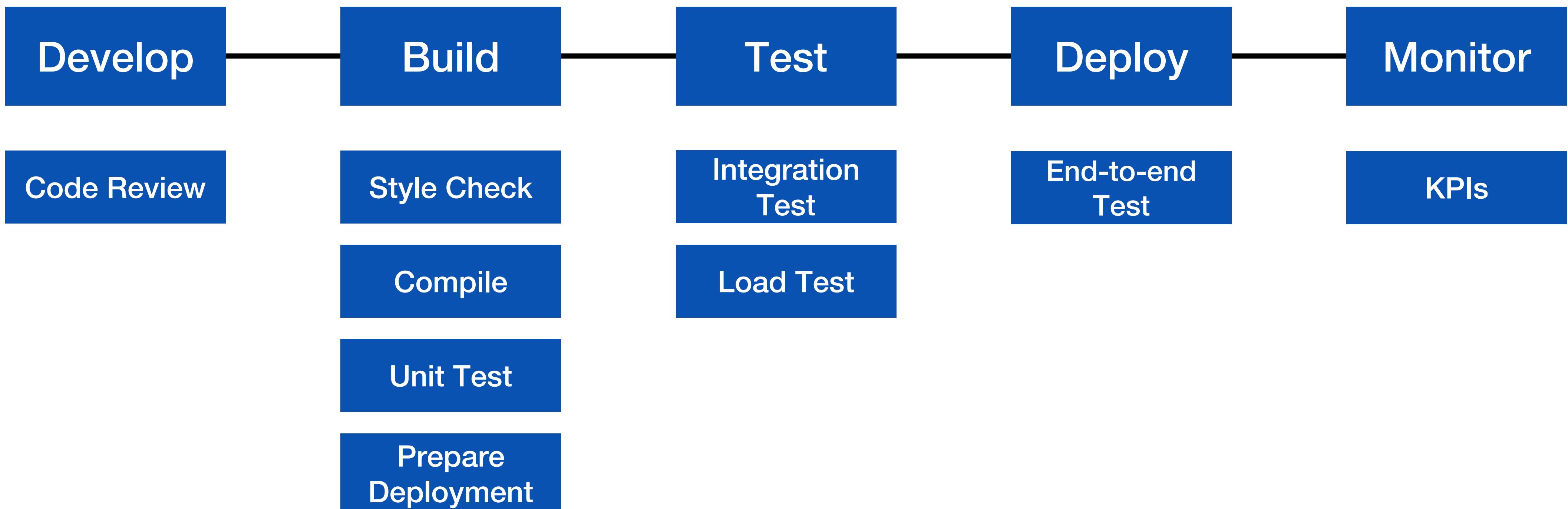
The software code was permanently repaired about five hours later.

Peyton added that even though the update to the DynamicSource software had been tested over an extended period, the bug was missed because it only presented when many aircraft at the same time were using the system.

Subsequently, a test of the software under high demand was developed.”

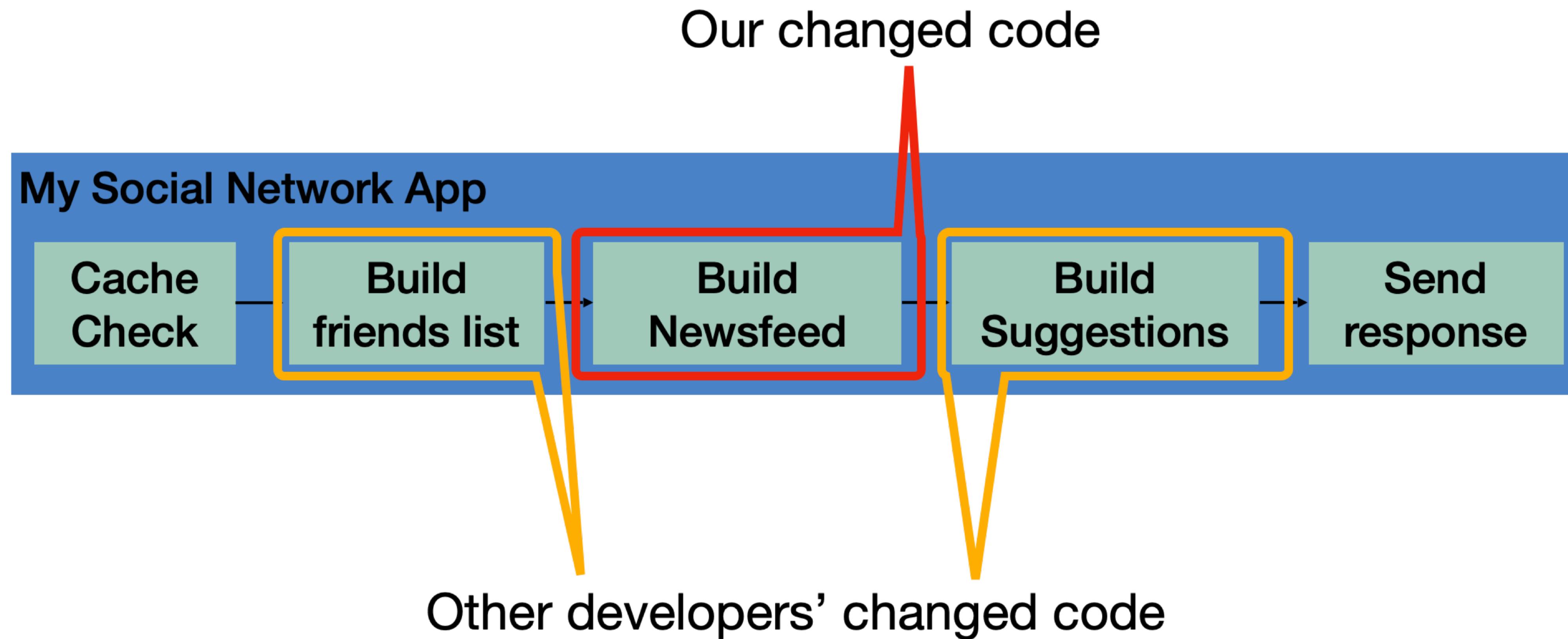
# Continuous development practices improves code quality and dev velocity

- Continuous integration: Perform frequent integrations with entire codebase, running integration-scale tests
- Continuous delivery: Deploy frequently and monitor



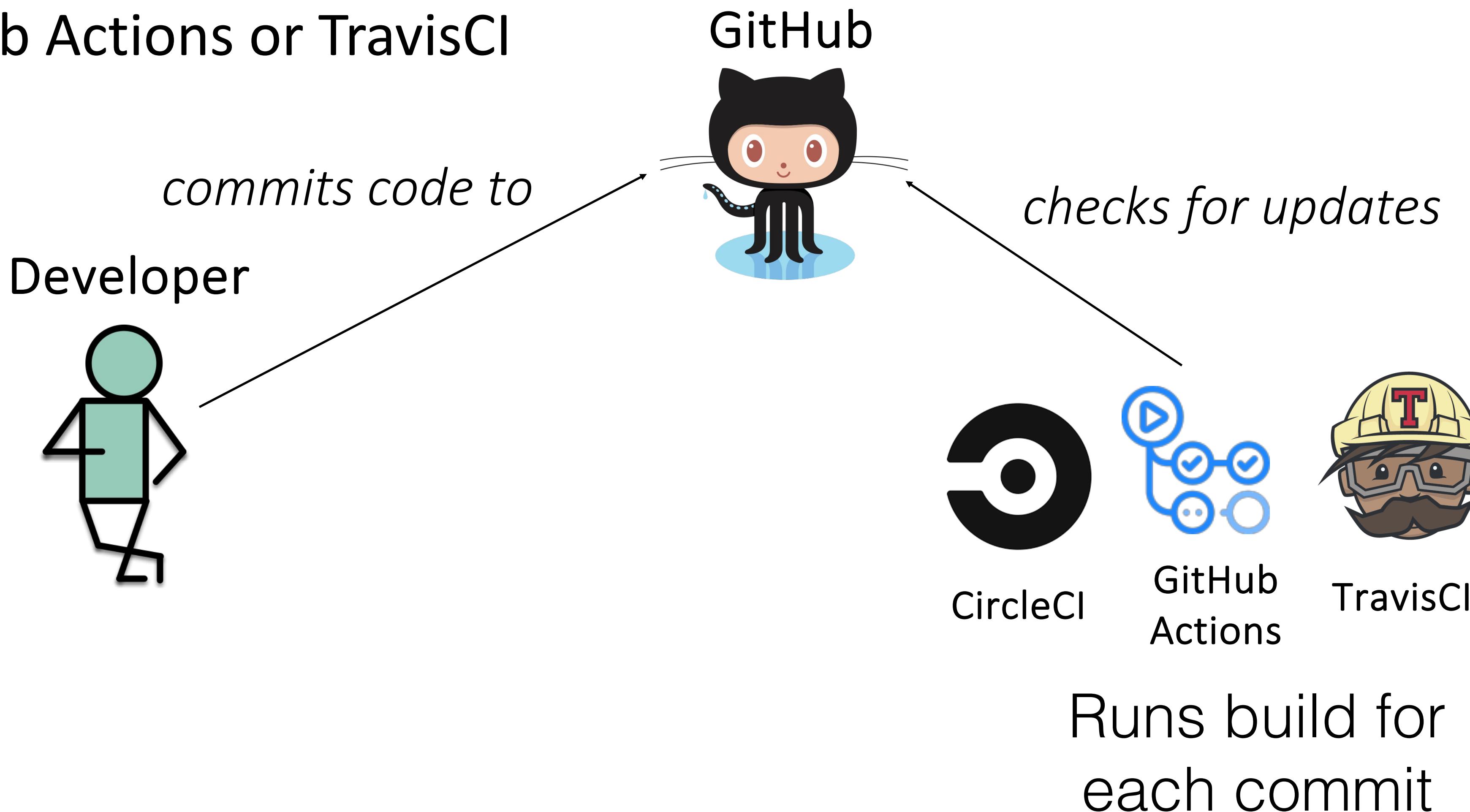
# Continuous Integration (CI) provides global feedback on local changes

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

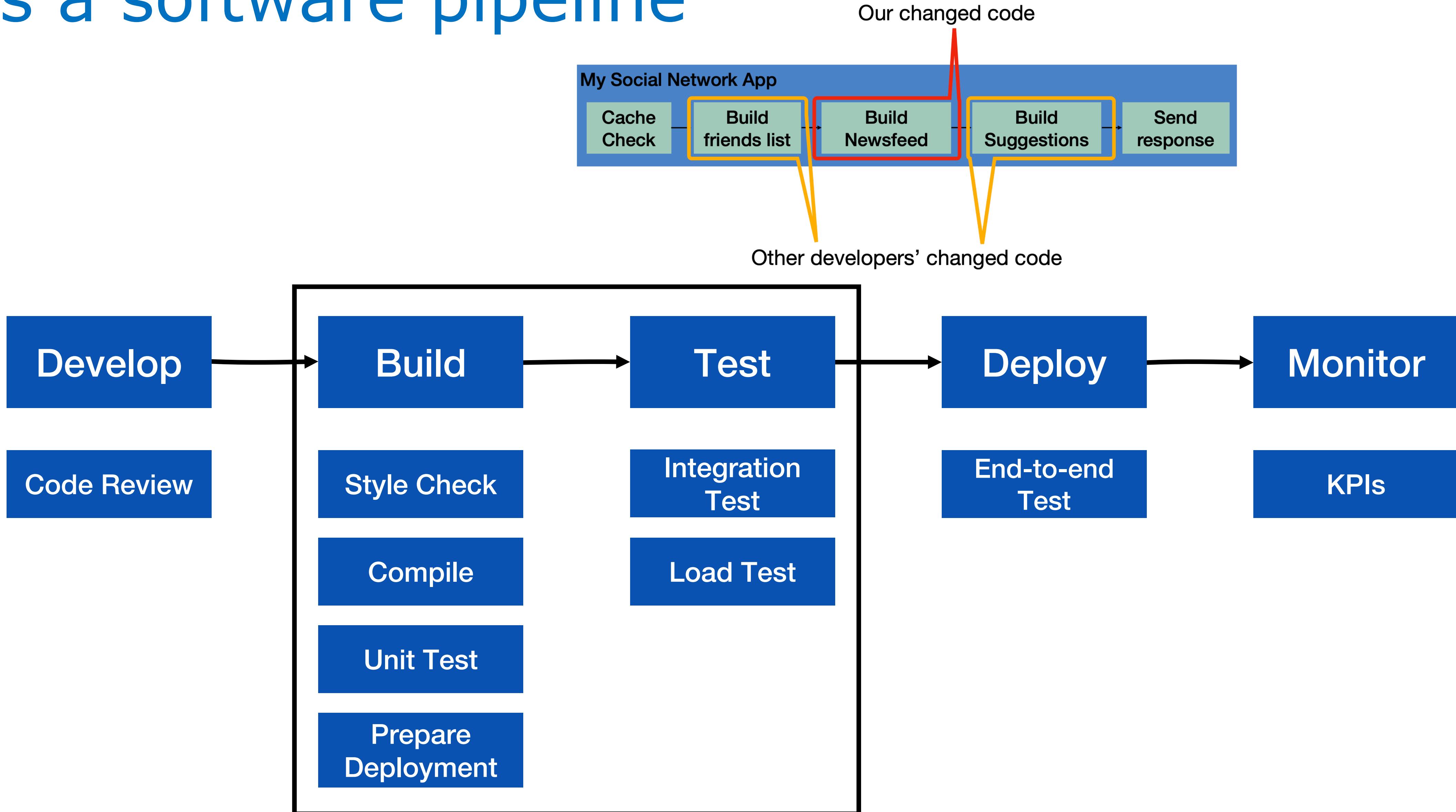


# CI is triggered by commits, pull requests, and other actions

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



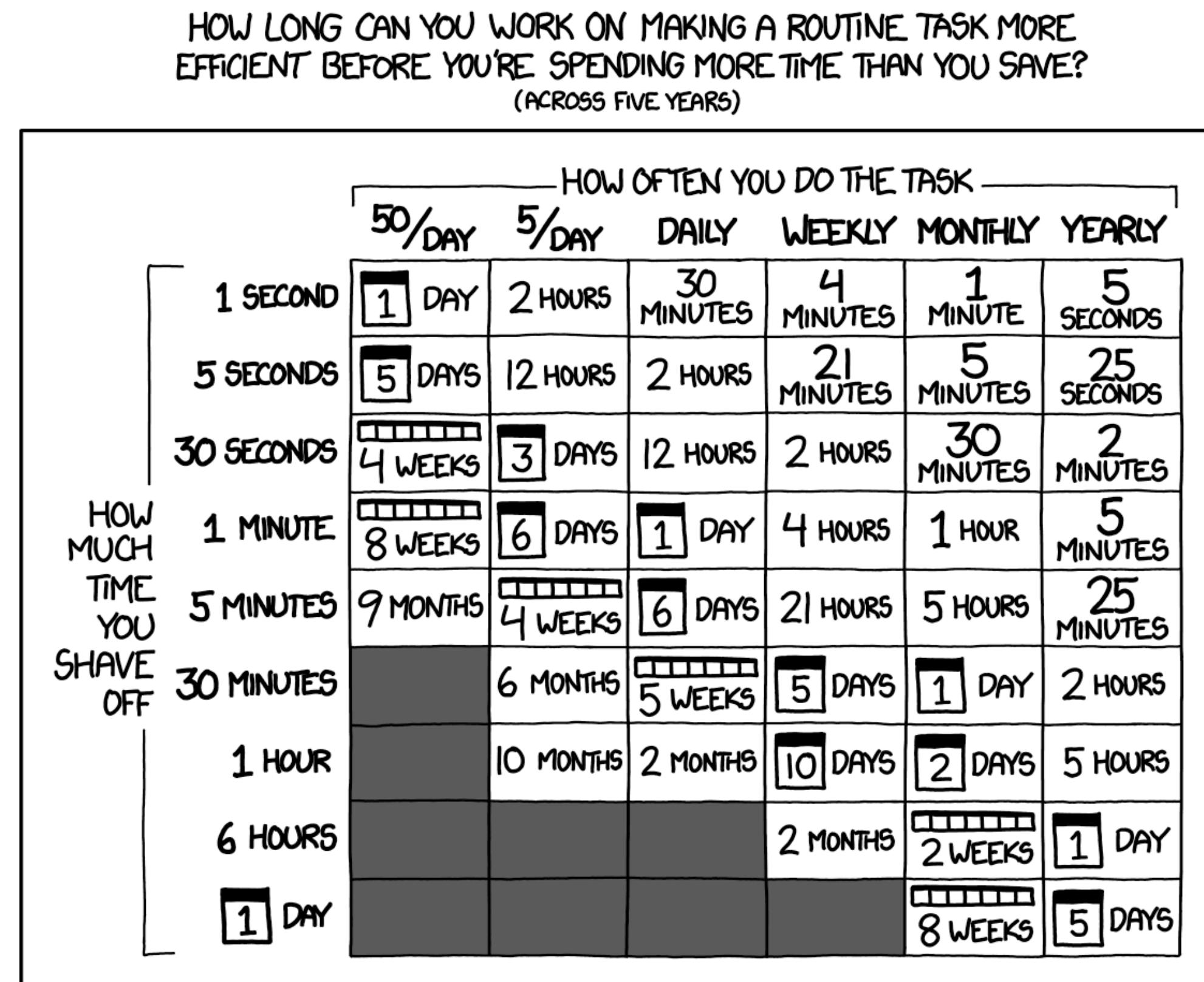
# CI is a software pipeline



Automate this centrally, provide a central record of results

# Automating Feedback Loops is Powerful

Consider tasks that are done by *dozens* of developers (e.g. testing/deployment)



# 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 - Autograder

At a glance, see history of build

<p>✓ <b>linting</b> Build and Test the Grader #11: Commit <a href="#">f3da101</a> pushed by <a href="#">jon-bell</a></p>	main	5 months ago ... 4m 20s
<p>✓ <b>Update handout and reference solution ba...</b> Check dist/ #10: Commit <a href="#">3073a5b</a> pushed by <a href="#">jon-bell</a></p>	main	5 months ago ... 41s
<p>✗ <b>Update handout and reference solution ba...</b> Build and Test the Grader #10: Commit <a href="#">3073a5b</a> pushed by <a href="#">jon-bell</a></p>	main	5 months ago ... 4m 29s
<p>✗ <b>Max 2 hints per mutant, provide the tests t...</b> Build and Test the Grader #9: Commit <a href="#">4cfe4ee</a> pushed by <a href="#">jon-bell</a></p>	main	6 months ago ... 4m 45s
<p>✓ <b>Max 2 hints per mutant, provide the tests t...</b> Check dist/ #9: Commit <a href="#">4cfe4ee</a> pushed by <a href="#">jon-bell</a></p>	main	6 months ago ... 39s
<p>✓ <b>New hint generator</b> Check dist/ #8: Commit <a href="#">012e440</a> pushed by <a href="#">jon-bell</a></p>	main	6 months ago ... 39s
<p>✗ <b>New hint generator</b> Build and Test the Grader #8: Commit <a href="#">012e440</a> pushed by <a href="#">jon-bell</a></p>	main	6 months ago ... 5m 9s

# 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)

✓ evaluate / run-fuzzer (m... 1d 0h)

✓ evaluate / run-fuzzer (rh... 1d 0h)

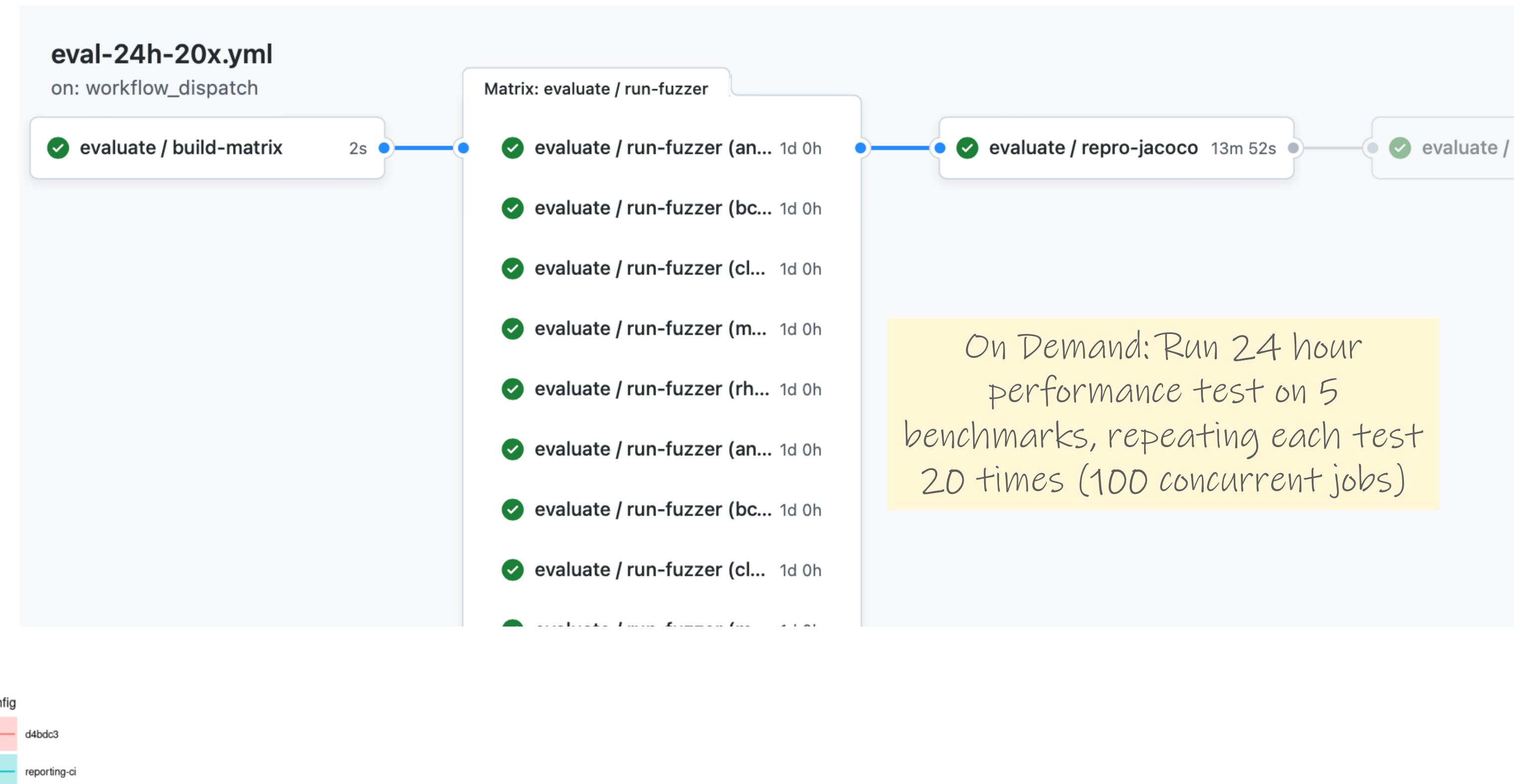
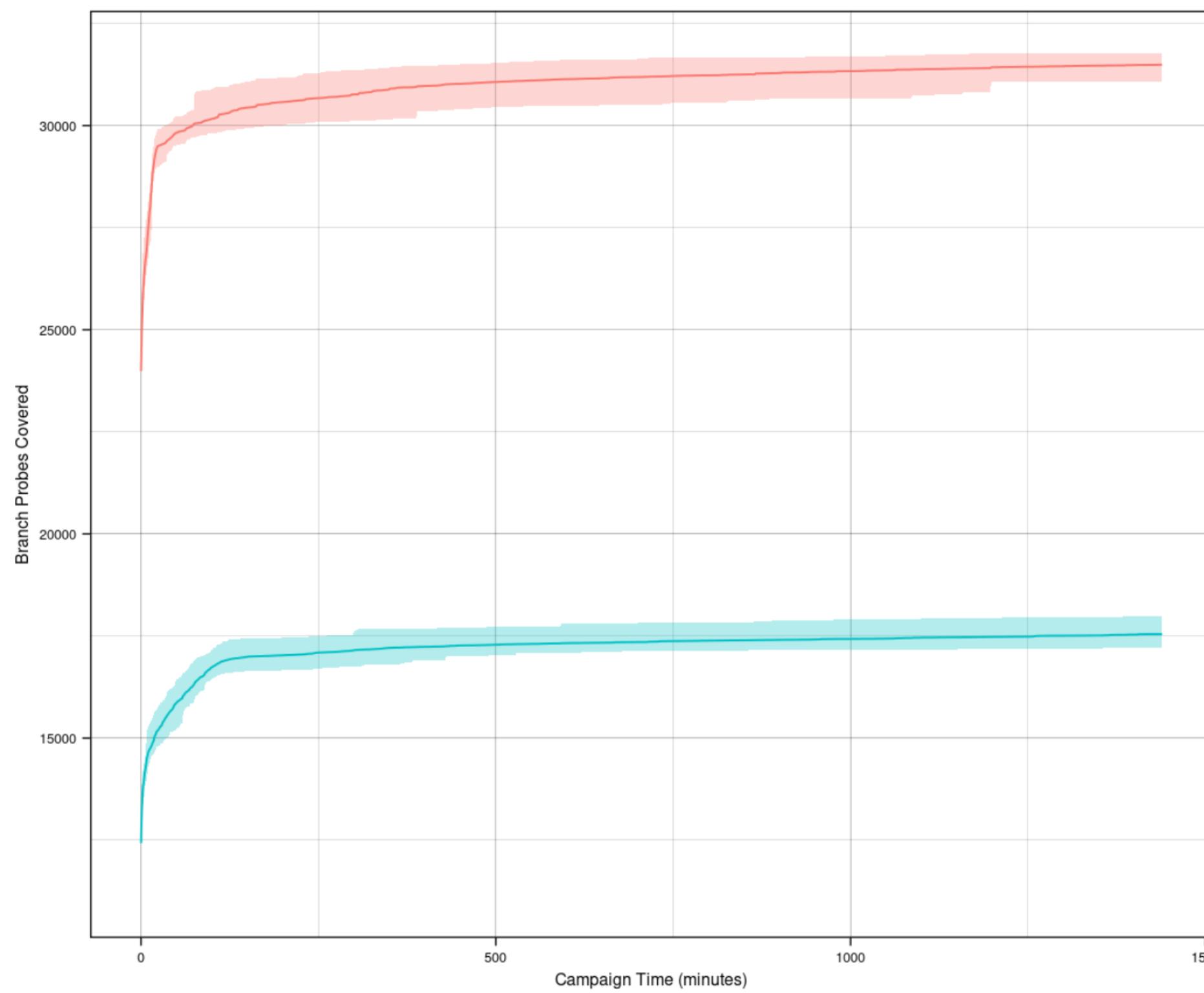
On Demand: Run 24 hour performance test on 5 benchmarks, repeating each test 20 times (100 concurrent jobs)

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

# CI Pipelines automate benchmarking

closure

Branch Probes Over Time



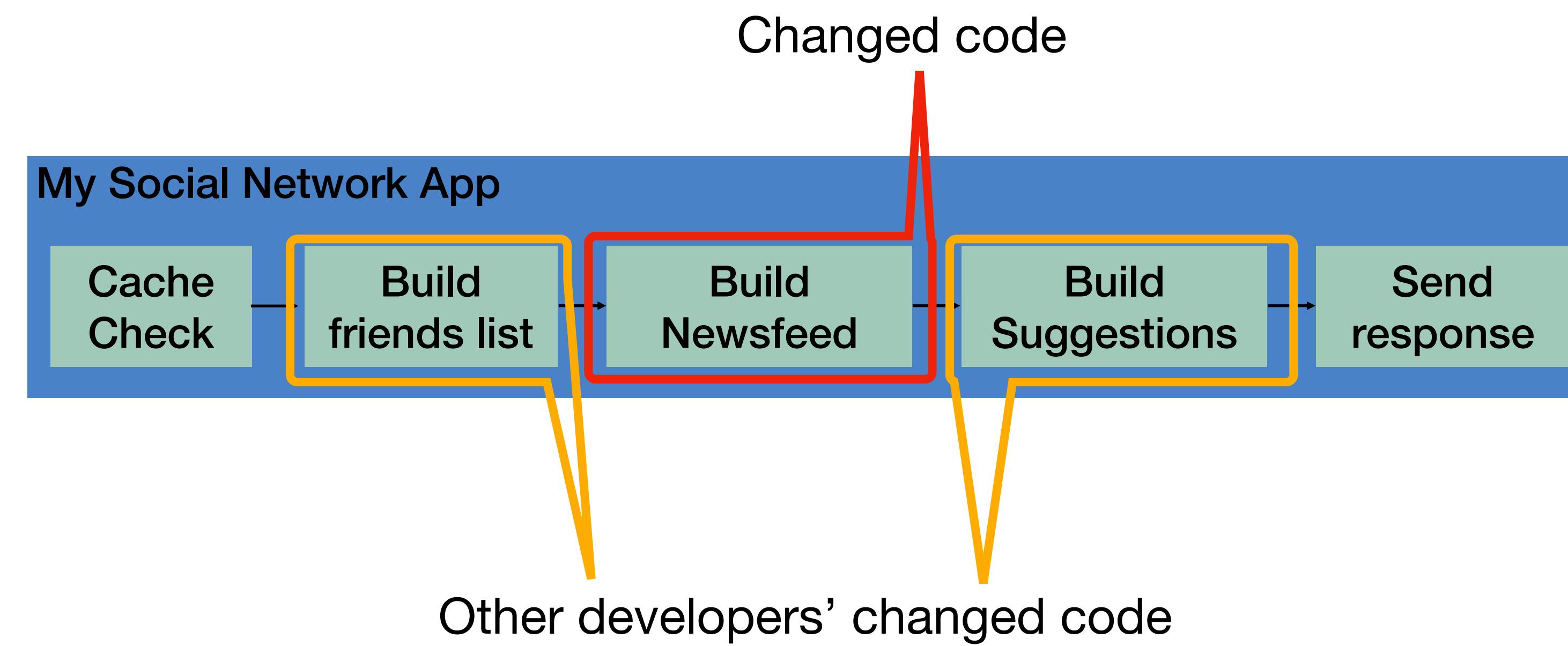
Download this graph as PDF

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

# Continuous Integration is Highly Configurable

Determining *how* to apply CI can be non-trivial for a larger project, all with a cost vs quality tradeoff: what is the cost of automation vs the value of developer time?

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



# Attributes of effective CI processes

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

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

- Output the full test name**: All checks have passed (9 successful checks).
- Check details:
  - Build and Test the Grader / build (push) - Successful
  - Check dist/ / check-dist (push) - Successful in 30s
  - Build and Test the Grader / test (reference) (push) - ...
  - Build and Test the Grader / test (b) (push) - Success...
  - Build and Test the Grader / test (ts-ignore) (push) - ...

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

- Tools: extract\_features.py: correct define name for AP\_RPM\_ENABLED - peterbarker committed 5 days ago (X)
- AP\_Mission: prevent use of uninitialised stack data - peterbarker committed 5 days ago (X) (2 comments)
- AP\_HAL\_ChibiOS: disable DMA on I2C on bdshot boards to free up DMA ch... - andyp1per authored and tridge committed 6 days ago (X)
- SITL: Fixed rounding lat/lng issue when running JSBSim SITL - ShivKhanna authored and tridge committed 6 days ago (X)
- AP\_HAL\_ChibiOS: define skyviper short board names - yuri-rage authored and tridge committed 6 days ago (X)

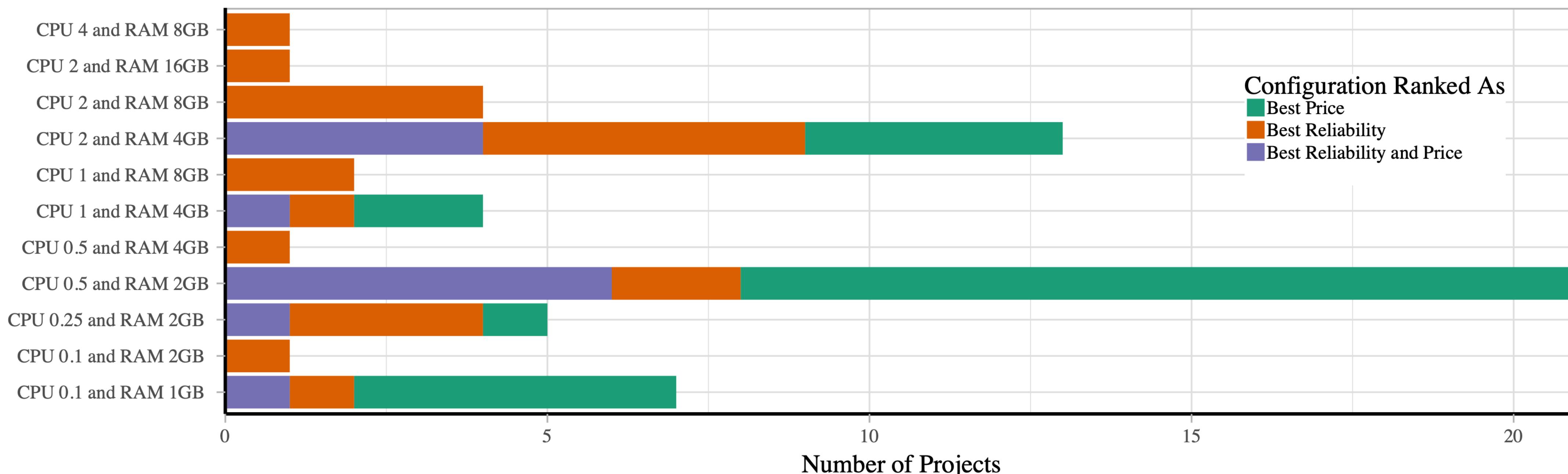
# Effective CI processes are run often enough to reduce debugging effort

- Failed CI runs indicate a bug was introduced, and caught in that run
- More changes per-CI run require more manual debugging effort to assign blame
- A single change per-CI run pinpoints the culprit

prestodb / presto			
Current	Branches	Build History	Pull Requests
✓ master	James Sun	This patch bumps Alluxio dependency to 2.3.0-2 #52300 passed	10 hrs 49 min 31 sec 2 days ago
! master	Andrii Rosa	Handle query level timeouts in Presto on Spark #52287 errored	11 hrs 6 min 44 sec 2 days ago
! master	Wenlei Xie	Fix flaky test for TestTempStorageSingleStreamSp #52284 errored	11 hrs 50 min 37 sec 2 days ago
✓ master	Andrii Rosa	Check requirements under try-catch #52283 passed	11 hrs 3 min 20 sec 2 days ago
✓ master	Maria Basmanova	Update TestHiveExternalWorkersQueries to create #52282 passed	10 hrs 55 min 37 sec 2 days ago
✓ master	Maria Basmanova	Introduce large dictionary mode in SliceDictionary #52277 passed	10 hrs 43 min 30 sec 2 days ago
! master	Maria Basmanova	Add Top N queries to TestHiveExternalWorkersQu #52271 errored	10 hrs 46 min 36 sec 3 days ago
✗ master	Leiqing Cai	Fix client-info test-name output #52266 failed	10 hrs 35 min 49 sec 3 days ago
✓ master	Andrii Rosa	Add Thrift transport support for TaskStatus #52263 passed	11 hrs 13 min 42 sec 3 days ago

# Effective CI processes allocate enough resources to mitigate flaky tests

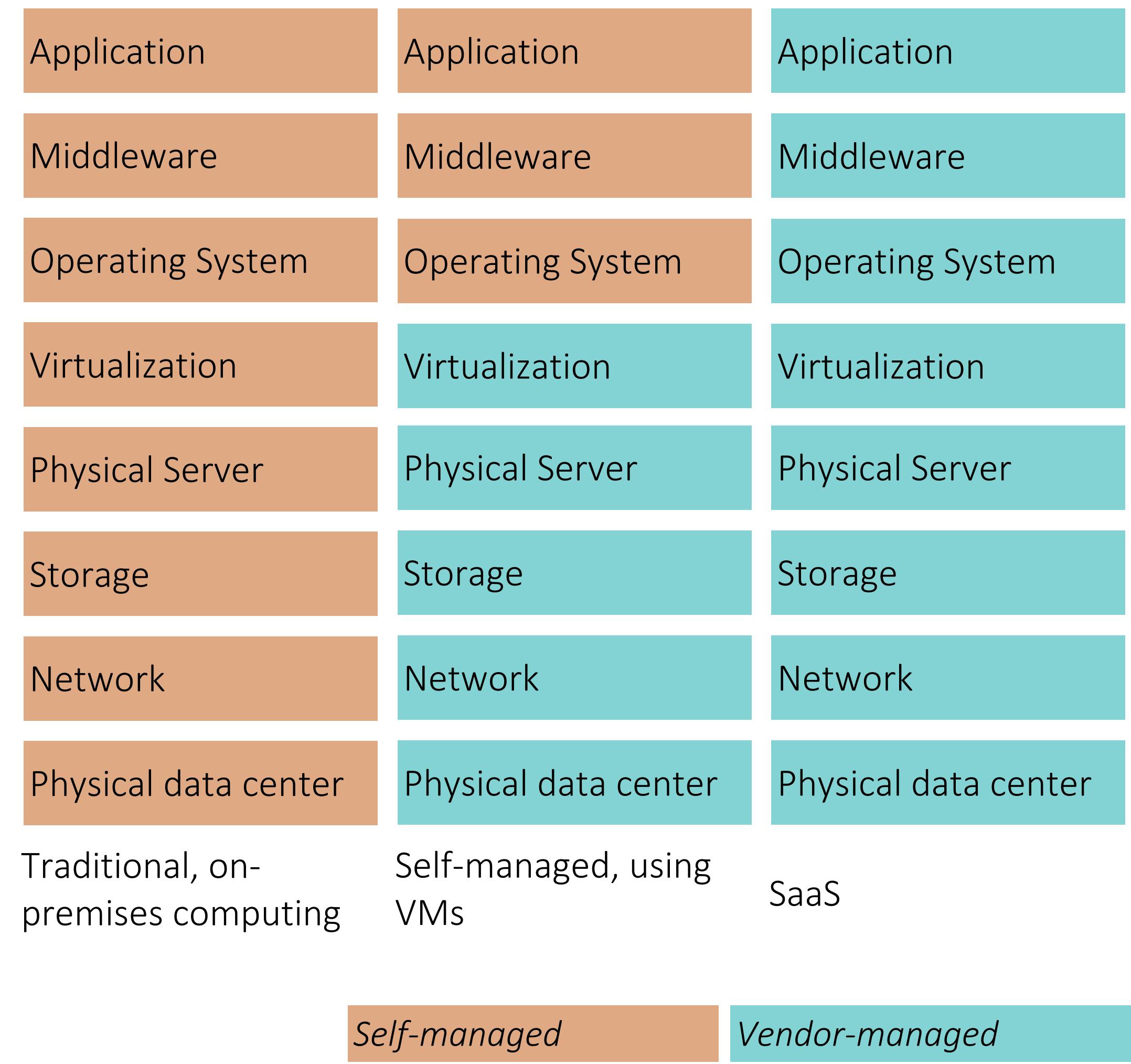
- *Flaky* tests might be dependent on timing (failing due to timeouts)
- Running tests without enough CPU/RAM can result in increased flaky failure rates and unreliable builds



[“The Effects of Computational Resources on Flaky Tests”, Silva et al](#)

# Continuous Integration Service Models

- Self-hosted/managed on-premises or in cloud
  - Jenkins
- Fully cloud managed
  - GitHub Actions, CircleCI, Travis, many more...
  - Billing model: pay per-build-minute running on SaaS infrastructure
  - “Self-hosted runners” run builds on your own infrastructure, usually “free”



# CI in practice at Google

## 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

# Challenges and Solutions for Repeatable Builds

---

- Which commands to run to produce an executable? (build systems)
- How to link third-party libraries? (dependency managers)
- How to specify system-level software requirements? (containers)
- How to specify infrastructure requirements? (Infrastructure as code)

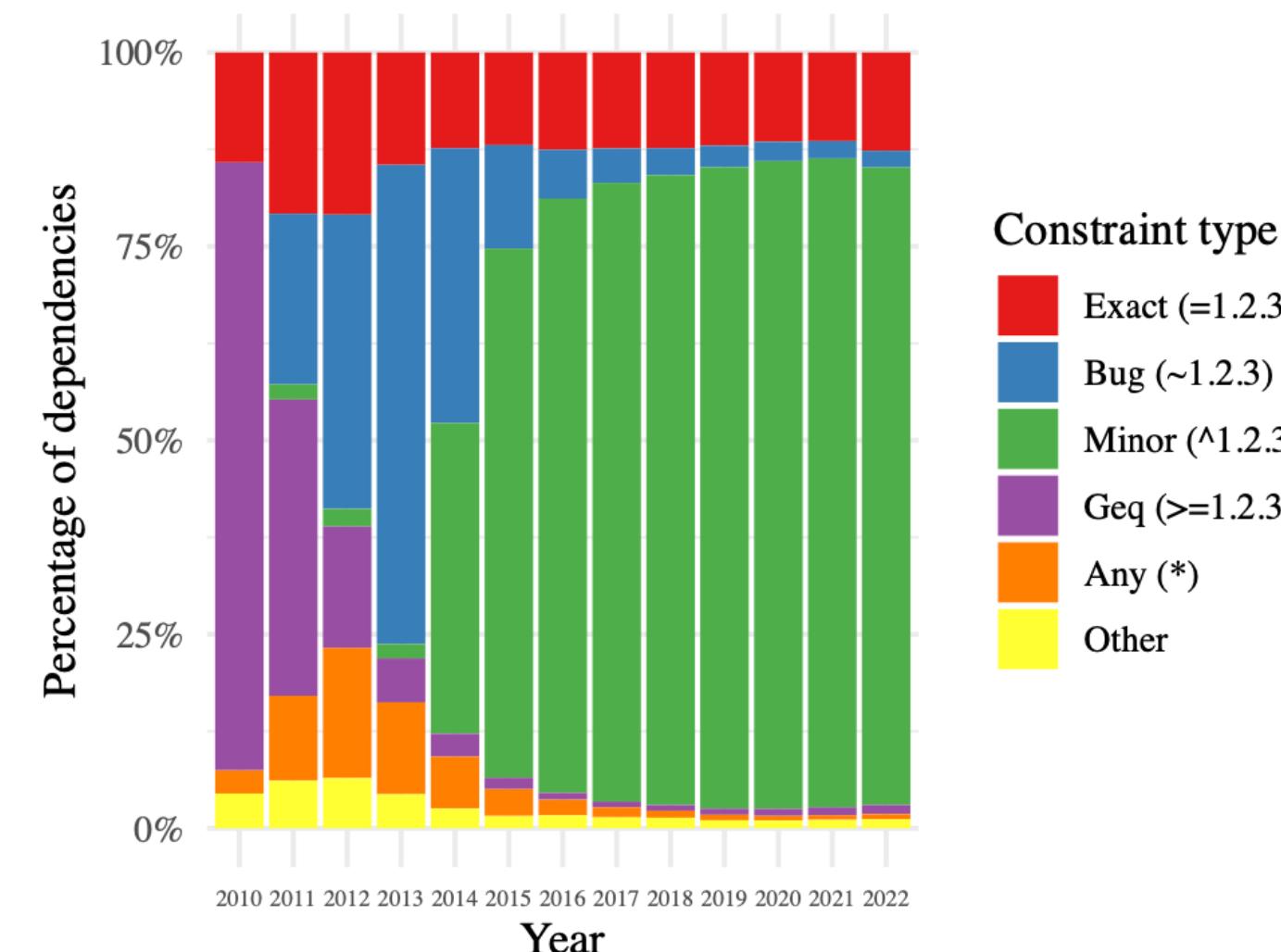
# Dependency Managers Organize External Dependencies

---

- Addresses this problem: “Before you compile this code, install commons-lang from the Apache website”
- Declare a dependency using coordinates (unique ID of a package plus version)
- Packages are archived in common repositories; fetched/linked by dependency manager
- Dependency managers handle transitive dependencies 
- Examples: Maven, NPM, pip, cargo, apt

# Specify and Depend on Package Versions with Care

- Semantic Versioning is often expected:
  - Library maintainers expected to indicate breaking changes with version numbers
  - Dependency consumers can specify constraints on versions (e.g. accept 2.0.x)



Distribution of dependencies of all packages in NPM over time (2023, Pinckney et al)

## Semantic Versioning 2.0.0

**Summary**

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes
2. MINOR version when you add functionality in a backwards compatible manner
3. PATCH version when you make backwards compatible bug fixes

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Constraint type	2.0.0	2.0.0-rc.2	2.0.0-rc.1	1.0.0	1.0.0-beta
Exact (=1.2.3)	Red	Red	Red	Red	Red
Bug (~1.2.3)	Blue	Blue	Blue	Blue	Blue
Minor (^1.2.3)	Green	Green	Green	Green	Green
Geq (>=1.2.3)	Purple	Purple	Purple	Purple	Purple
Any (*)	Orange	Orange	Orange	Orange	Orange
Other	Yellow	Yellow	Yellow	Yellow	Yellow

# Build Systems Orchestrate Software Engineering Tasks

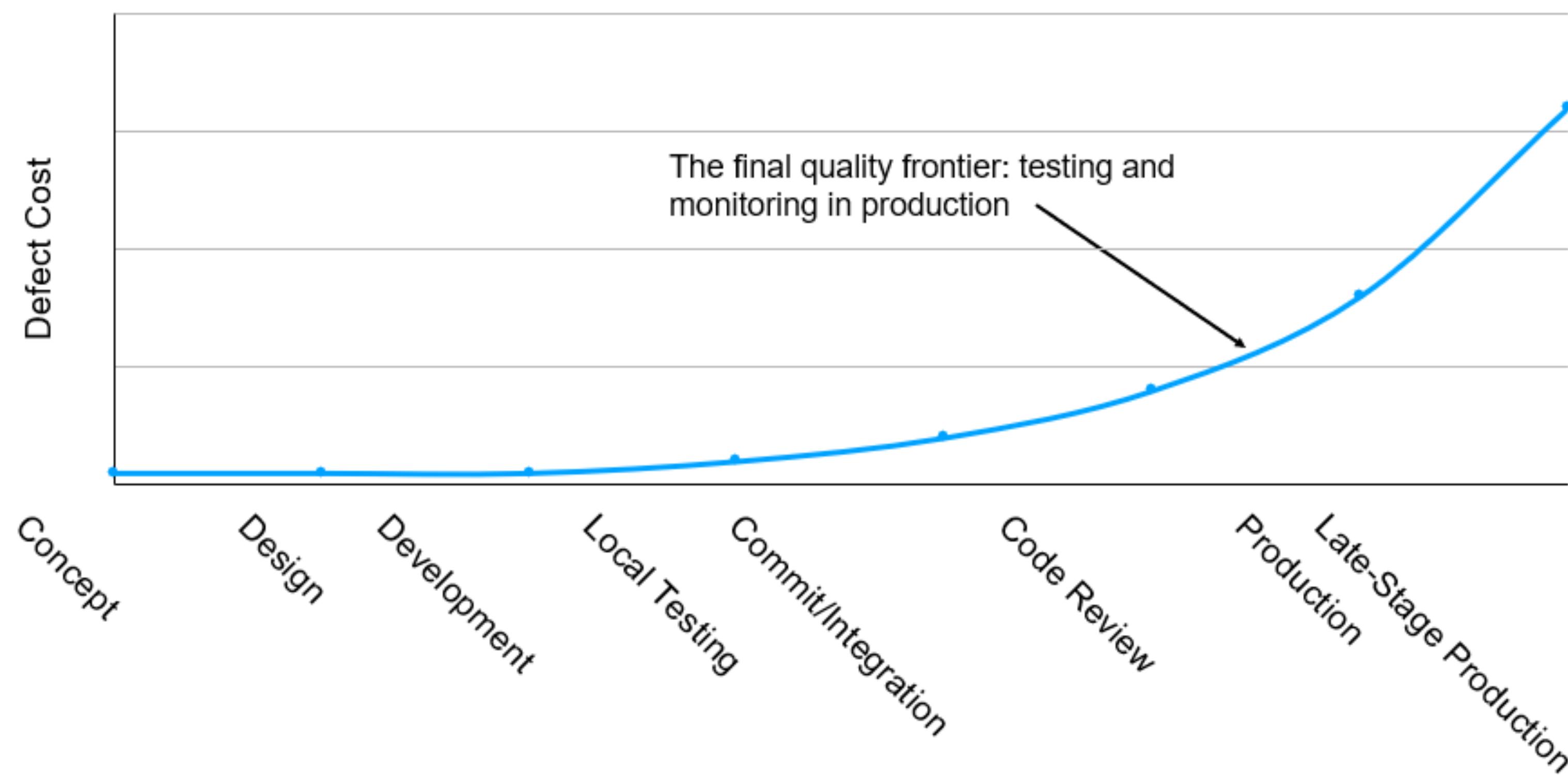
---

- “Orchestrate” -> Execute in the right order, ideally with concurrency, example tasks:
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Deploying Code
- Example build systems: xMake, ant, maven, gradle, npm...

# 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



# Motivating scenario: Failed Deployment at Knight Capital

## 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 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

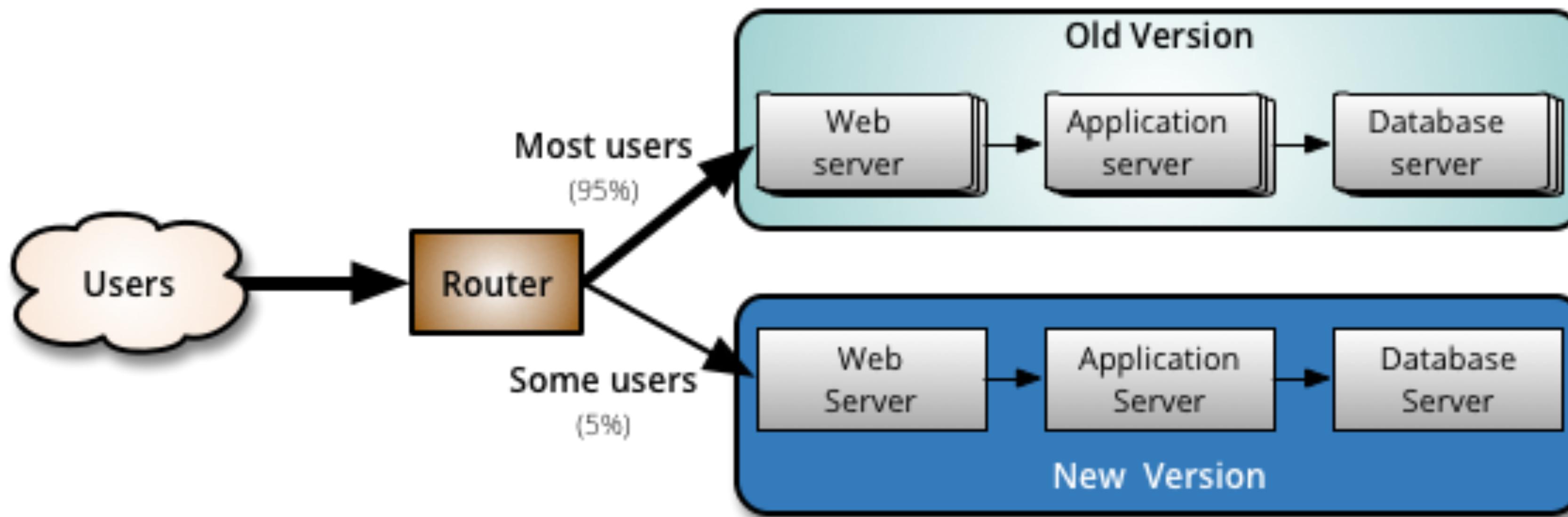
# Continuous Delivery != Immediate Delivery

---

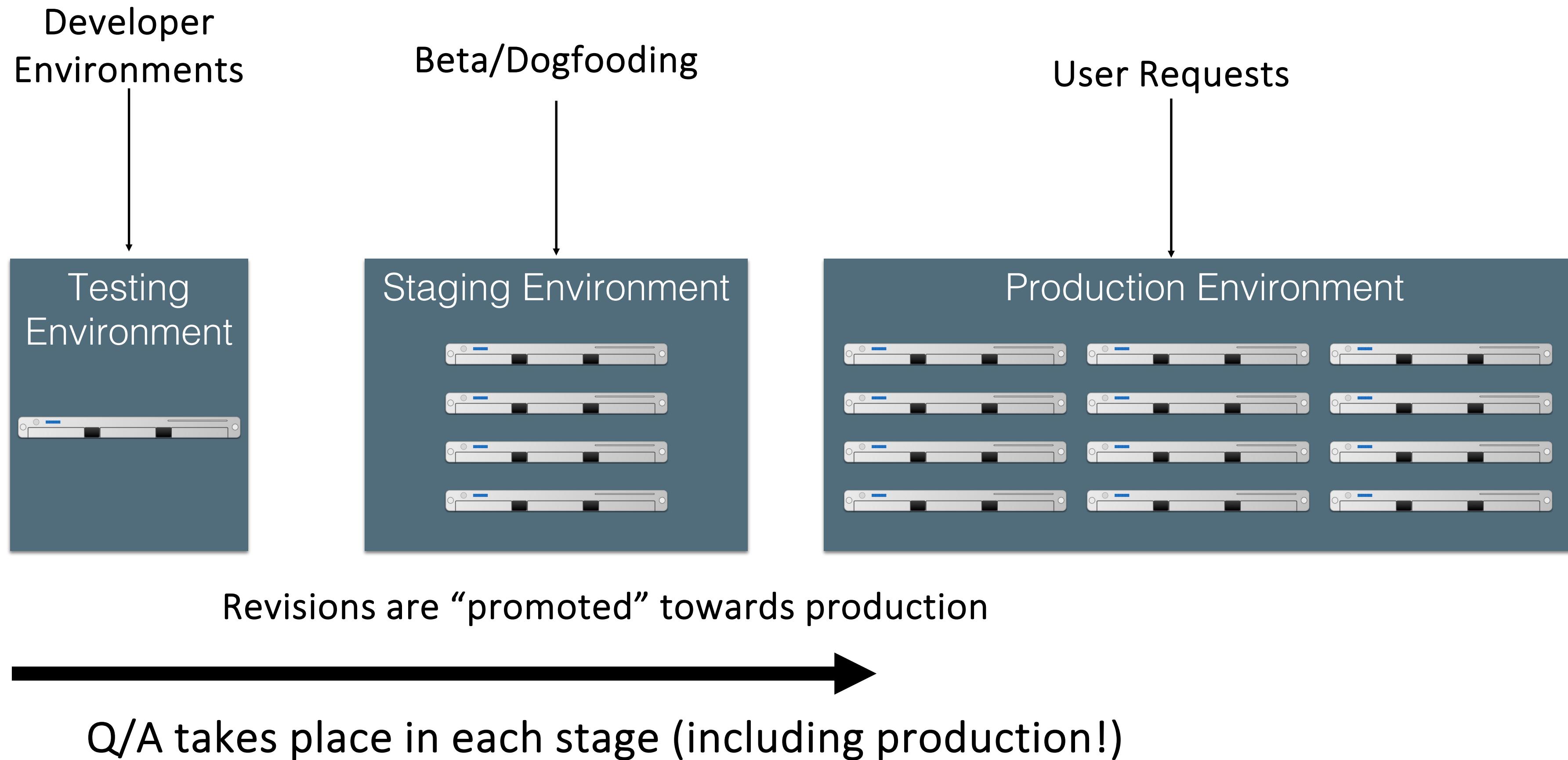
- Even if you are deploying every day (“continuously”), you still have some latency
- A new feature I develop today won't be released today
- But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)
- **Release Engineer:** gatekeeper who decides when something is ready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

# Split Deployments Mitigate Risk

- Idea: Deploy to a complete production-like environment, but don't have users use it, collect preliminary feedback
- Lower risk if a problem occurs in staging than in production
- Examples:
  - “Eat your own dogfood”
  - Beta/Alpha testers

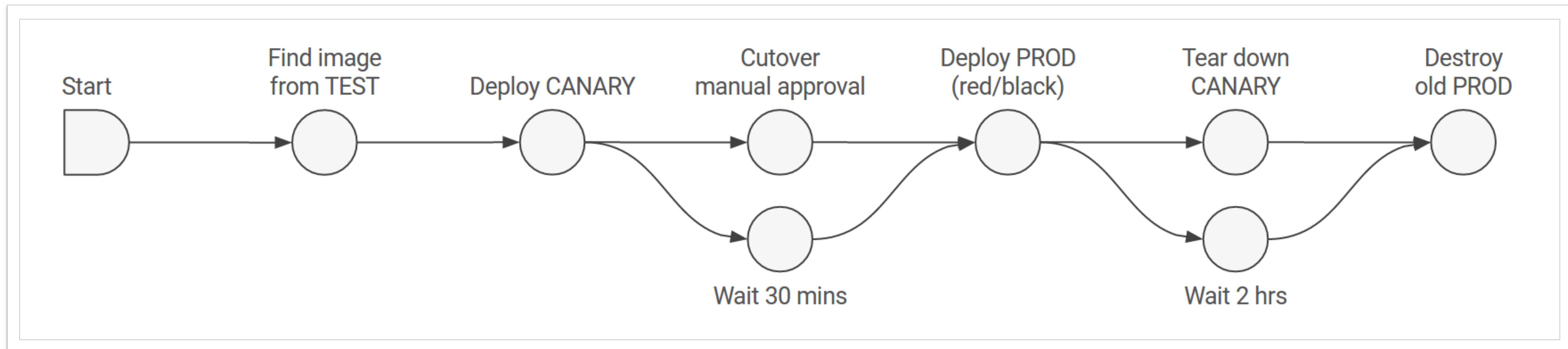


# Continuous Delivery Leverages Relies on Staging Environments



# Continuous Delivery Tools

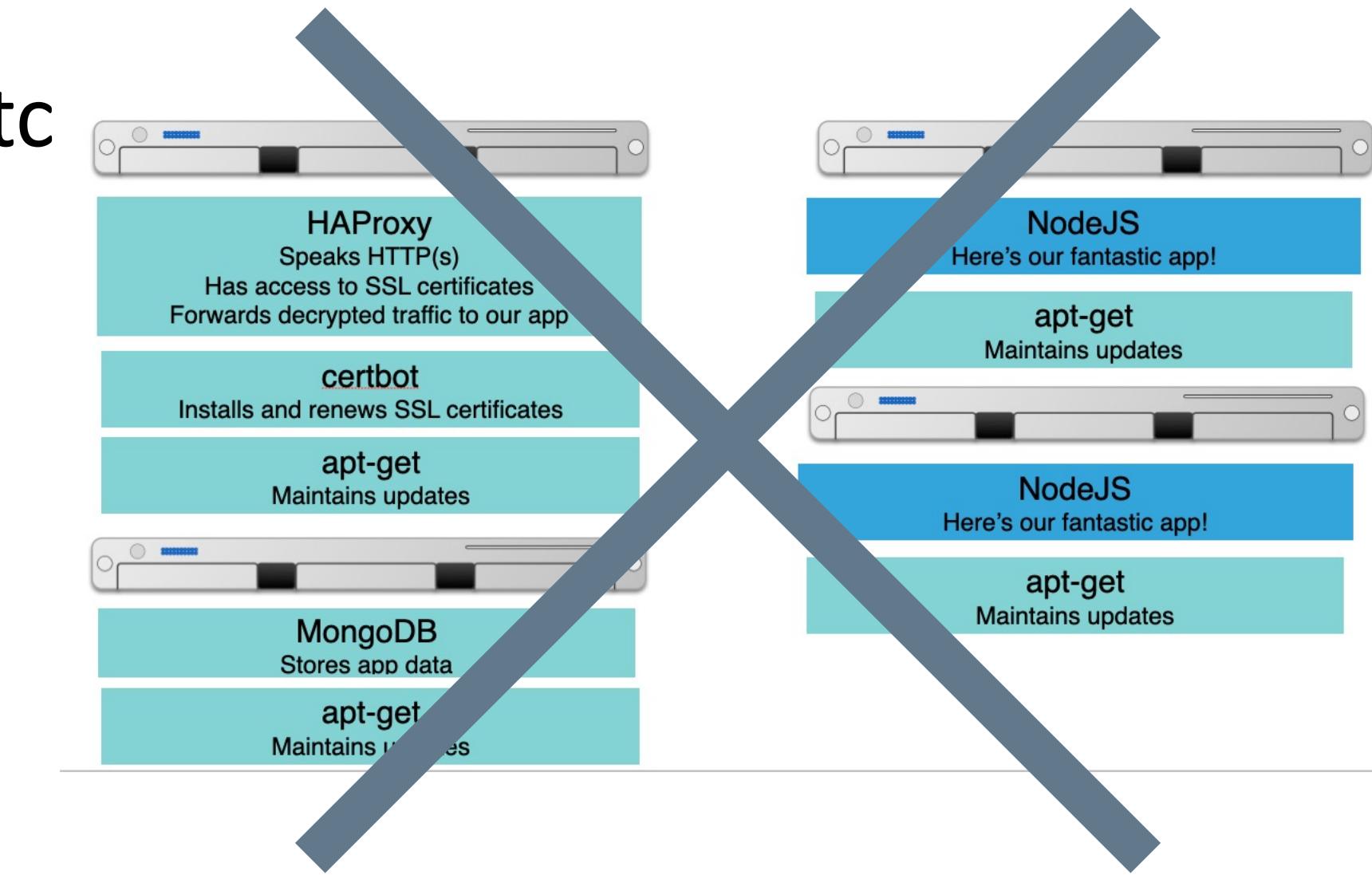
- Simplest tools deploy from a branch to a service (e.g. Render.com, Heroku)
- More complex tools:
  - Auto-deploys from version control to a staging environment + promotes through release pipeline
  - Monitors key performance indicators to automatically take corrective actions
  - Example: “[Spinnaker](#)” (Open-Sourced by Netflix, c 2015)



Example CD pipeline from Spinnaker's documentation: <https://spinnaker.io/docs/concepts/#application-deployment>

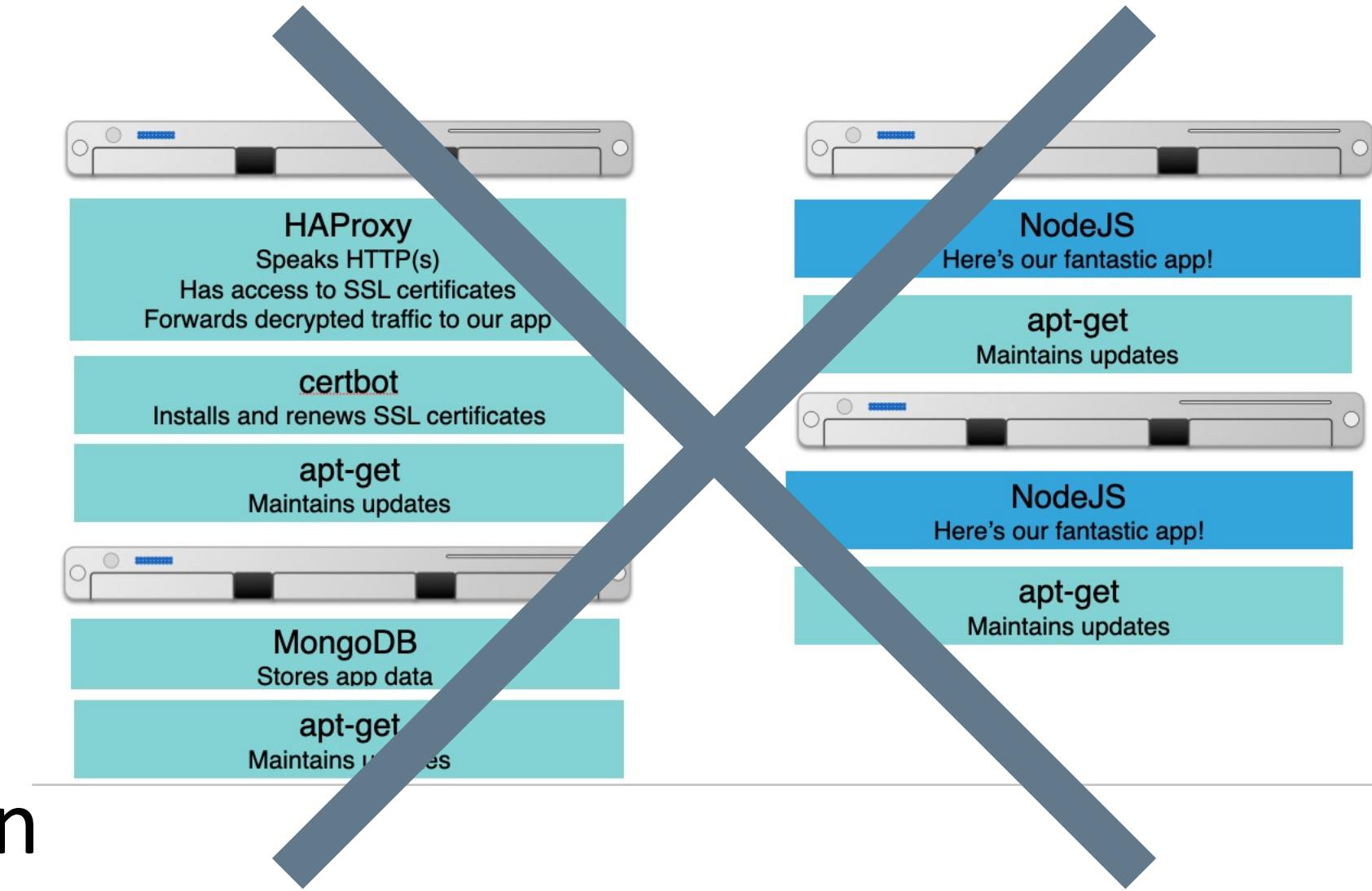
# Continuous Delivery Relies on Infrastructure As Code

- Provisioning servers is tedious and error prone
  - Deploy a VM, then ssh to it, install some packages, etc
- Keeping servers up-to-date is also a struggle
- Ideal:
  - “Give me HAProxy with some configuration file, and keep that configuration in a git repo, and when I change it, roll out an update”
  - “Give me some containers running my NodeJS app, and when I update my app, roll it out to those containers”
  - “Give me a bunch of servers with MongoDB set up in a cluster”



# Infrastructure as Code represents complex infrastructure in “recipes”

- Goal: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
  - These configurations can then go into version control, code review, etc
- Metaphor: “Recipes” for configuring servers, organized into “cookbooks”
  - Engineers define “healthy” states for infrastructure, then system automatically provisions, validates, and (if needed) repairs deployed resources
    - “Oh, this is how they do things at Amazon” - Inspiration for [Chef, c 2009](#)
    - Other tools with similar aims: Puppet (c 2005), Ansible (c 2012)



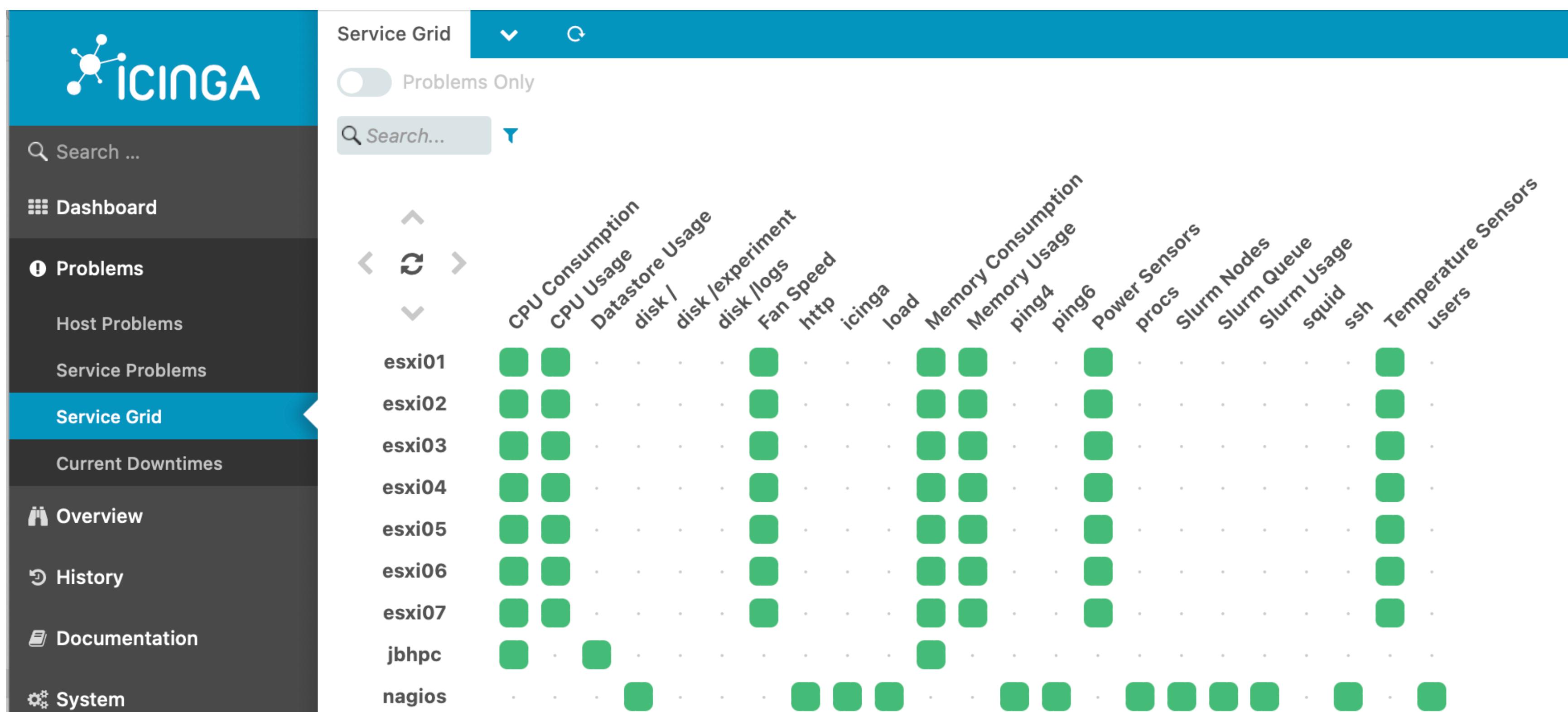
# Continuous Delivery Relies on Monitoring

---

- Consider both direct (e.g. business) metrics, and indirect (e.g. system) 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

# Tools for Monitoring Deployments

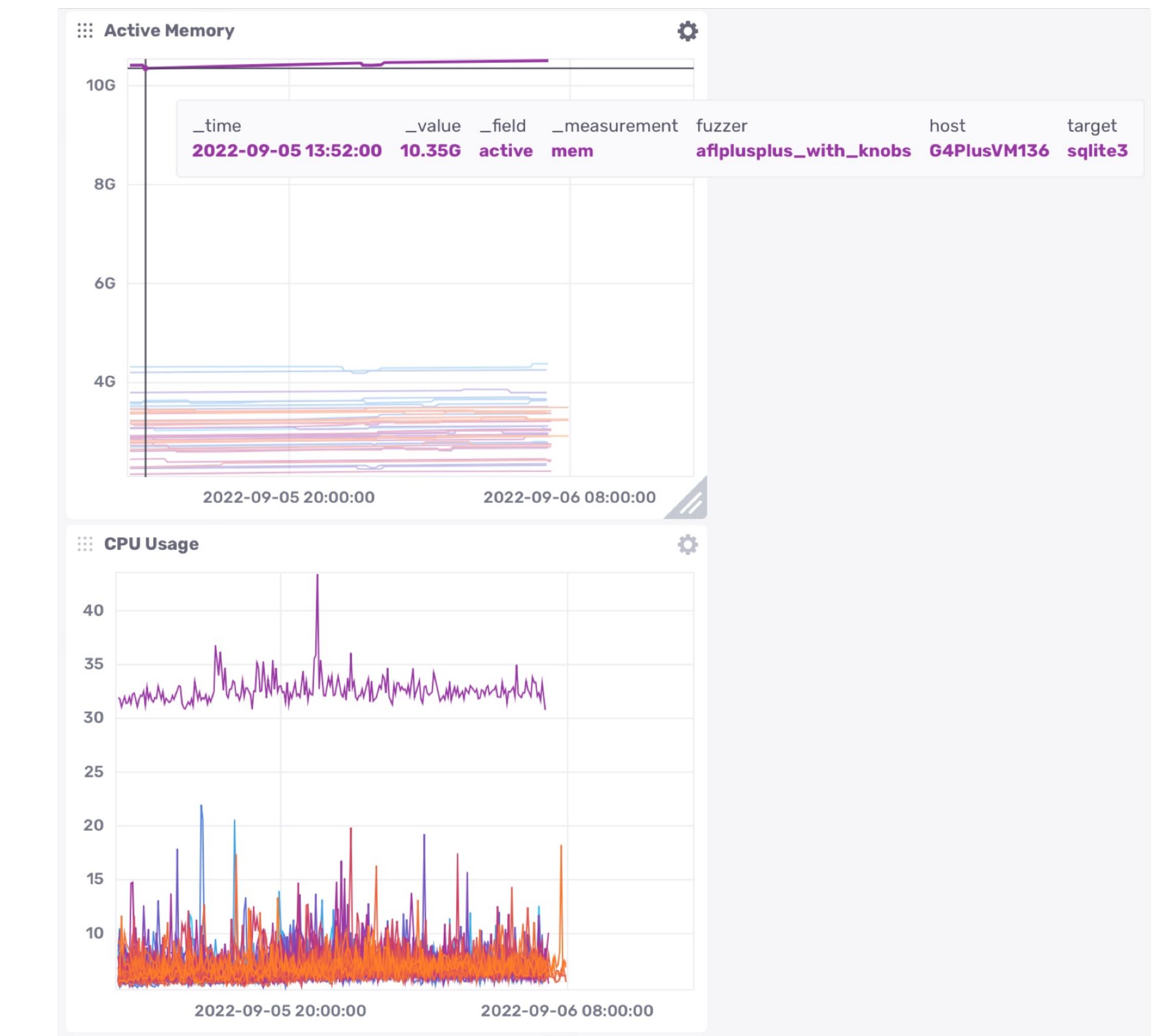
- Nagios (c 2002): Agent-based architecture (install agent on each monitored host), extensible plugins for executing “checks” on hosts
  - Track system-level metrics, app-level metrics, user-level KPIs



# Monitoring can help identify operational issues



Grafana (AGPL, c 2014)



InfluxDB (MIT license, c 2013)

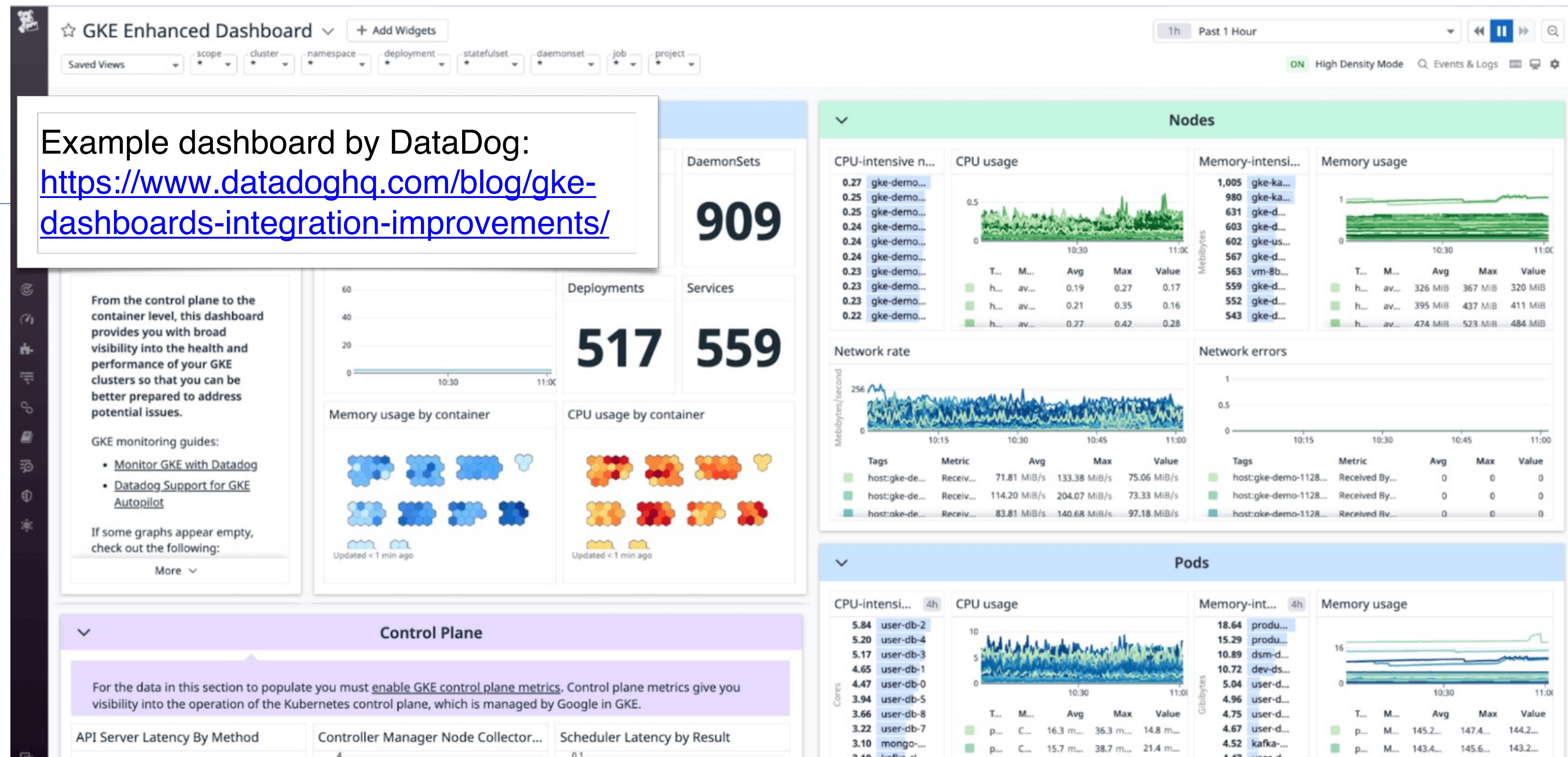
# Continuous Delivery Tools Take Automated Actions

- Example: Automated roll-back of updates at Netflix based on SPS

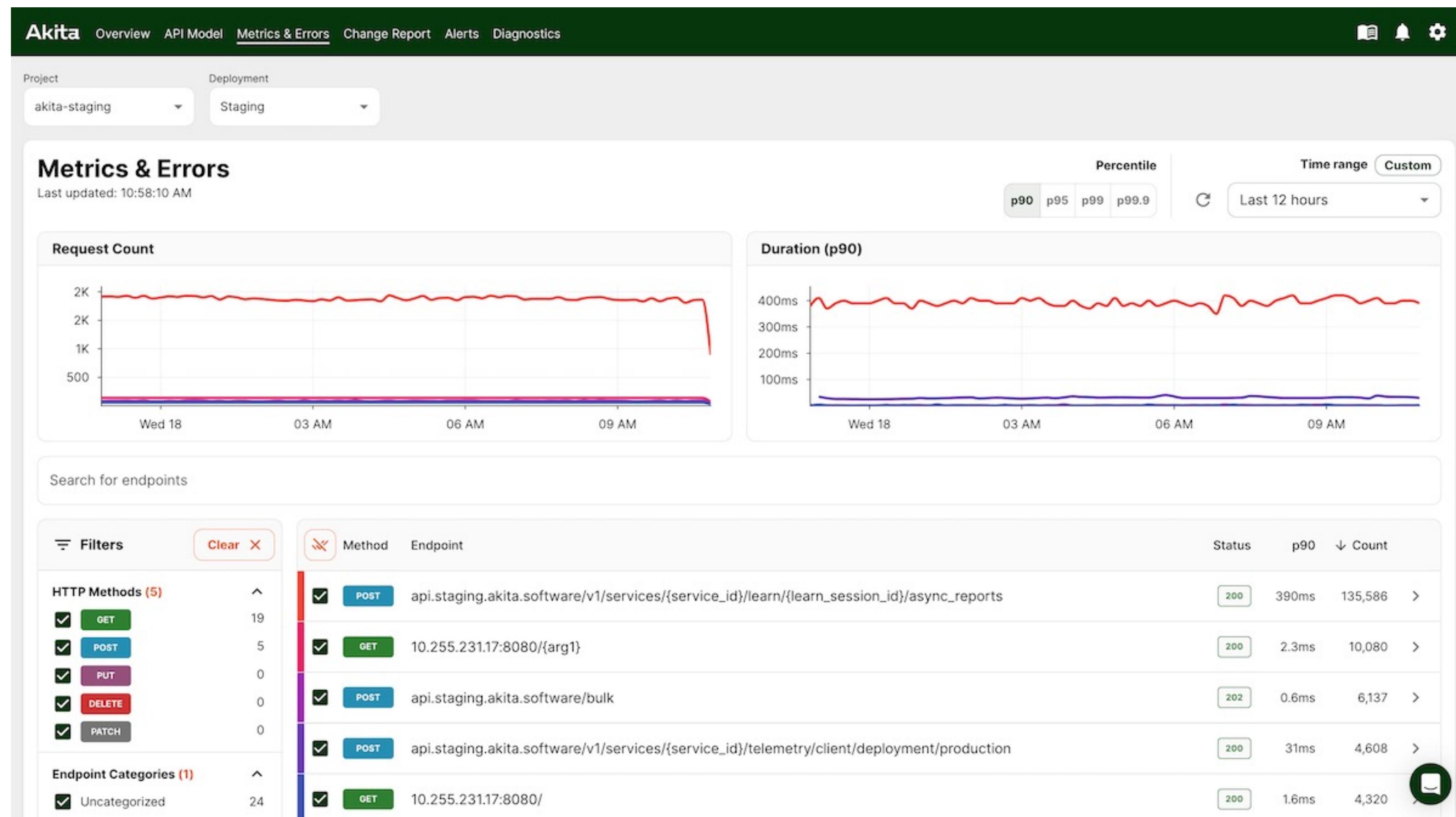


# From Monitoring to Observability

Understanding what is going on inside of our deployed systems

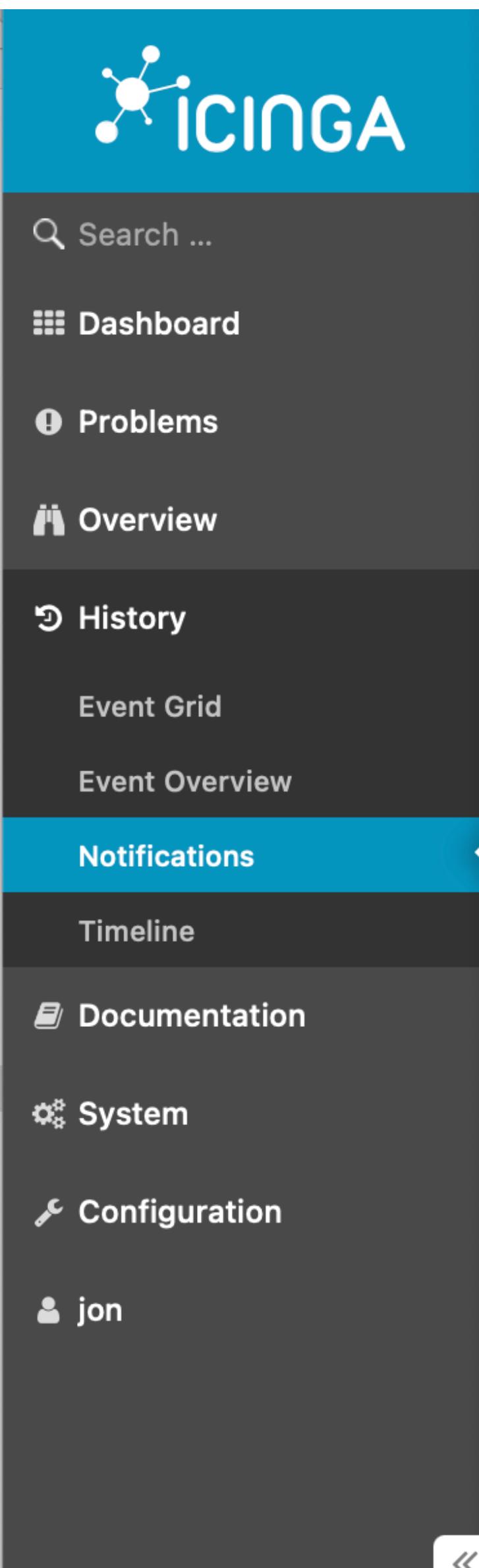


# New Tools allow Observability inside of Apps, Too



Screenshot: <https://www.akitasoftware.com/blog-posts/plug-and-play-endpoint-views-for-metrics-errors>

# Monitoring Services Take Automated Actions



Notifications		
<a href="#">OK</a> 2022-02-18 08:49:05 Slurm Nodes on nagios OK - 0 nodes unreachable, 332 reachable Sent to jon		
<a href="#">OK</a> 2022-02-18 08:49:05 Slurm Nodes on nagios OK - 0 nodes unreachable, 332 reachable Sent to icingaadmin		
<a href="#">WARNING</a> 2022-02-18 08:45:05 Slurm Nodes on nagios WARNING - 7 nodes unreachable, 326 reachable Sent to jon		
<a href="#">WARNING</a> 2022-02-18 08:45:05 Slurm Nodes on nagios WARNING - 7 nodes unreachable, 326 reachable Sent to icingaadmin		
<a href="#">CRITICAL</a> 2022-02-18 08:42:05 Slurm Nodes on nagios CRITICAL - 65 nodes unreachable, 161 reachable Sent to icingaadmin		
<a href="#">CRITICAL</a> 2022-02-18 08:42:05 Slurm Nodes on nagios CRITICAL - 65 nodes unreachable, 161 reachable Sent to jon		
<a href="#">WARNING</a> 2022-02-18 08:40:05 Slurm Nodes on nagios WARNING - 12 nodes unreachable, 205 reachable Sent to icingaadmin		
<a href="#">WARNING</a> 2022-02-18 08:40:05 Slurm Nodes on nagios WARNING - 12 nodes unreachable, 205 reachable Sent to jon		
<a href="#">CRITICAL</a> 2022-02-18 08:34:07 Slurm Nodes on nagios CRITICAL - 204 nodes unreachable, 145 reachable Sent to icingaadmin		

Notification	
Current Service State	
UP since 2021-11	<b>nagios</b> ::1 127.0.0.1
OK for 1m 52s	Service: <b>Slurm Nodes</b>
Event Details	
Type	Notification
Start time	2022-02-18 08:42:05
End time	2022-02-18 08:42:05
Reason	Normal notification
State	<span style="background-color: red; border-radius: 50%; width: 15px; height: 15px; display: inline-block;"></span> CRITICAL
Escalated	No
Contacts notified	2
Output	CRITICAL - 65 nodes unreachable, 161 reachable

# Beware of Metrics

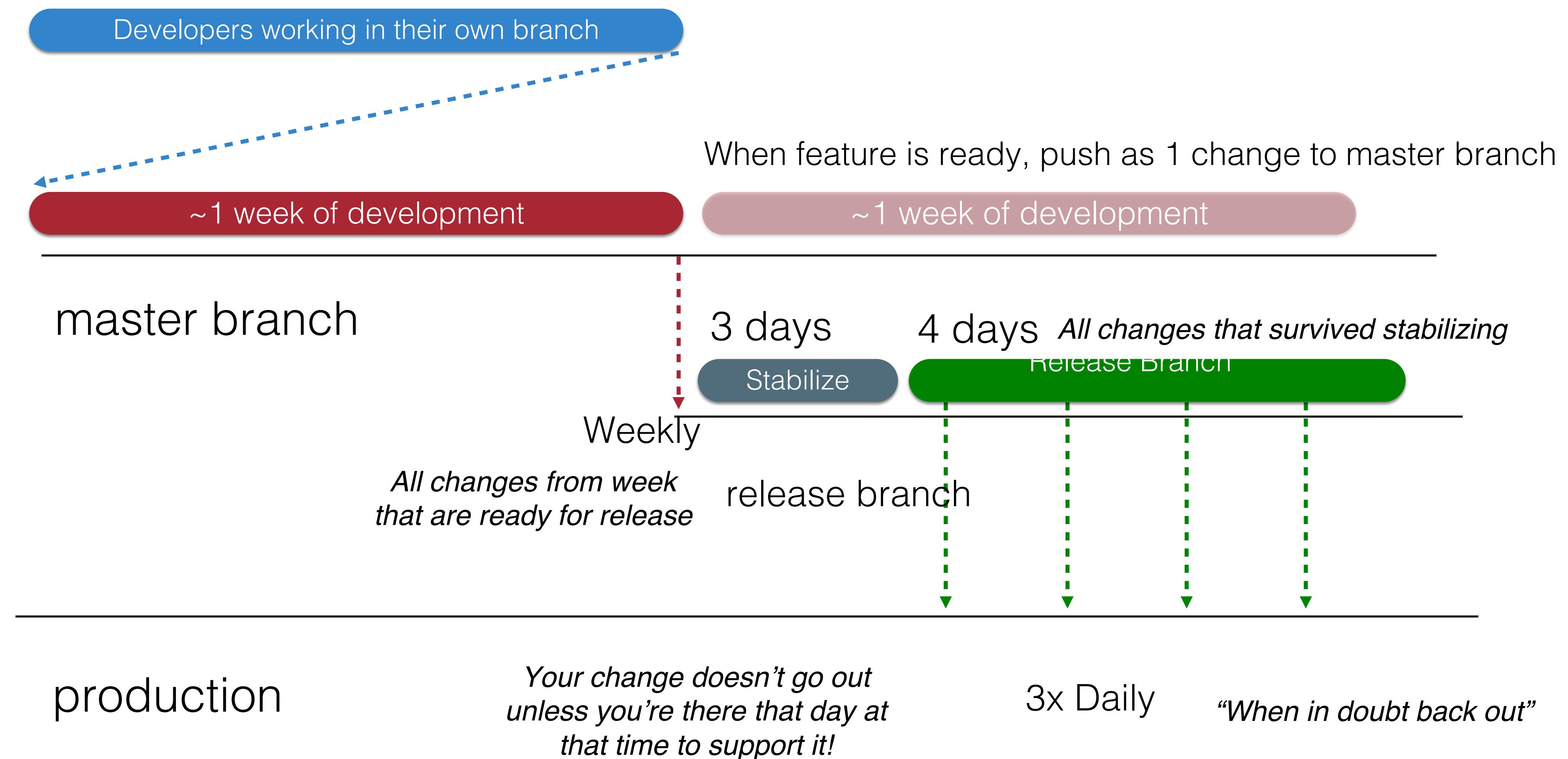
## McNamara Fallacy

- 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



# Deployment Example: Facebook.com

Pre-2016



# Deployment Example

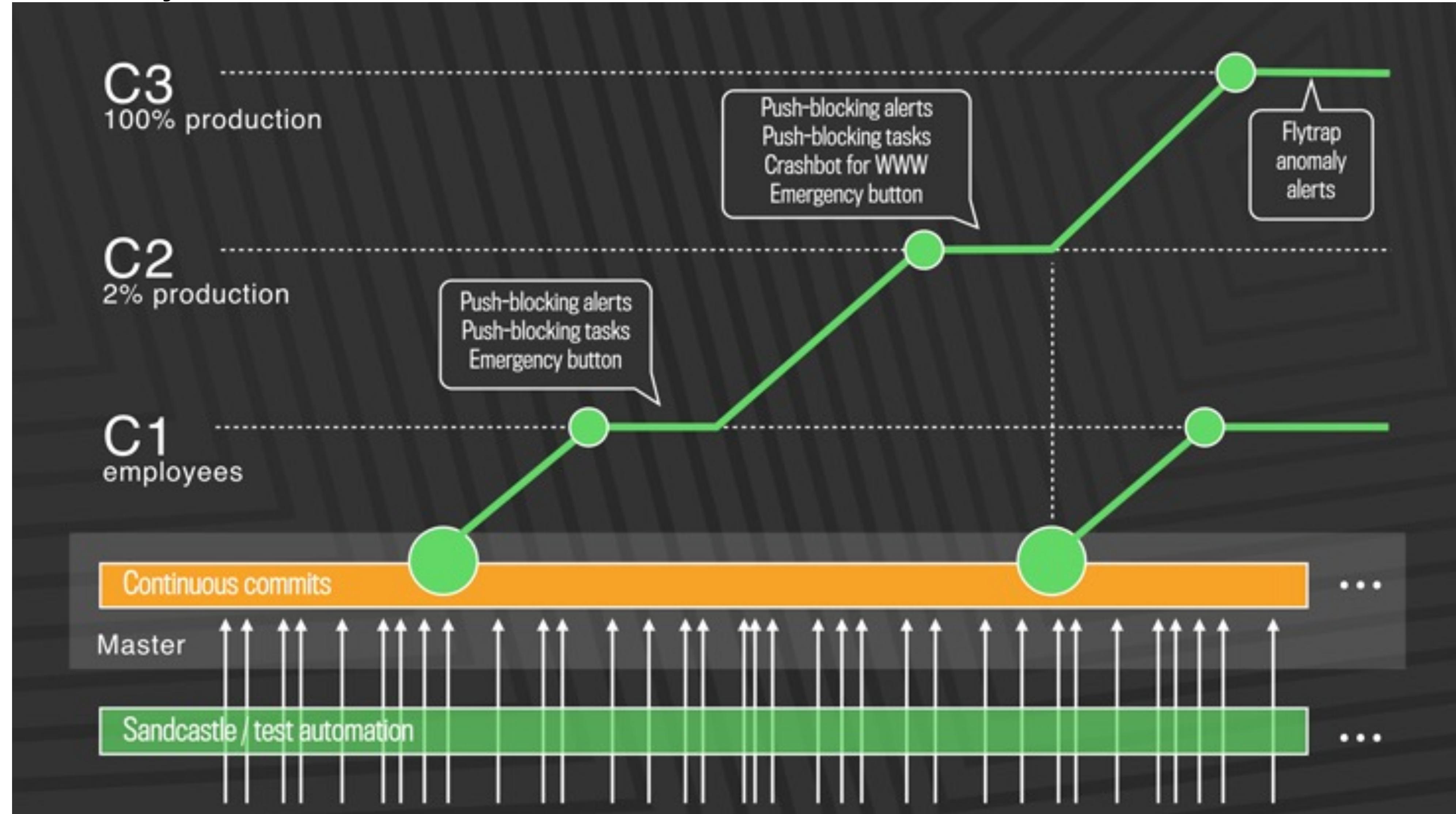
Chuck Rossi, Director Software Infrastructure & Release Engineering @ Facebook



“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



# Compare Continuous Delivery and TDD

---

- Test driven development
  - Write and maintain tests per-feature
  - Unit tests help locate bugs (at unit level)
  - Integration/system tests also needed to locate interaction-related faults
- Continuous delivery
  - Write and maintain high-level observability metrics
  - Deploy features one-at-a-time, look for canaries in metrics
  - Write fewer integration/system tests

# Review

- By now, you should be able to...
  - Describe how continuous development helps to catch errors sooner in the software lifecycle
  - Describe strategies for performing quality-assurance on software as and after it is delivered
  - Compare and contrast continuous delivery with test driven development as a quality assurance strategy