

Release Engineering

Advanced Software Engineering FS17

Gerald Schermann, schermann@ifi.uzh.ch



**University of
Zurich** ^{UZH}

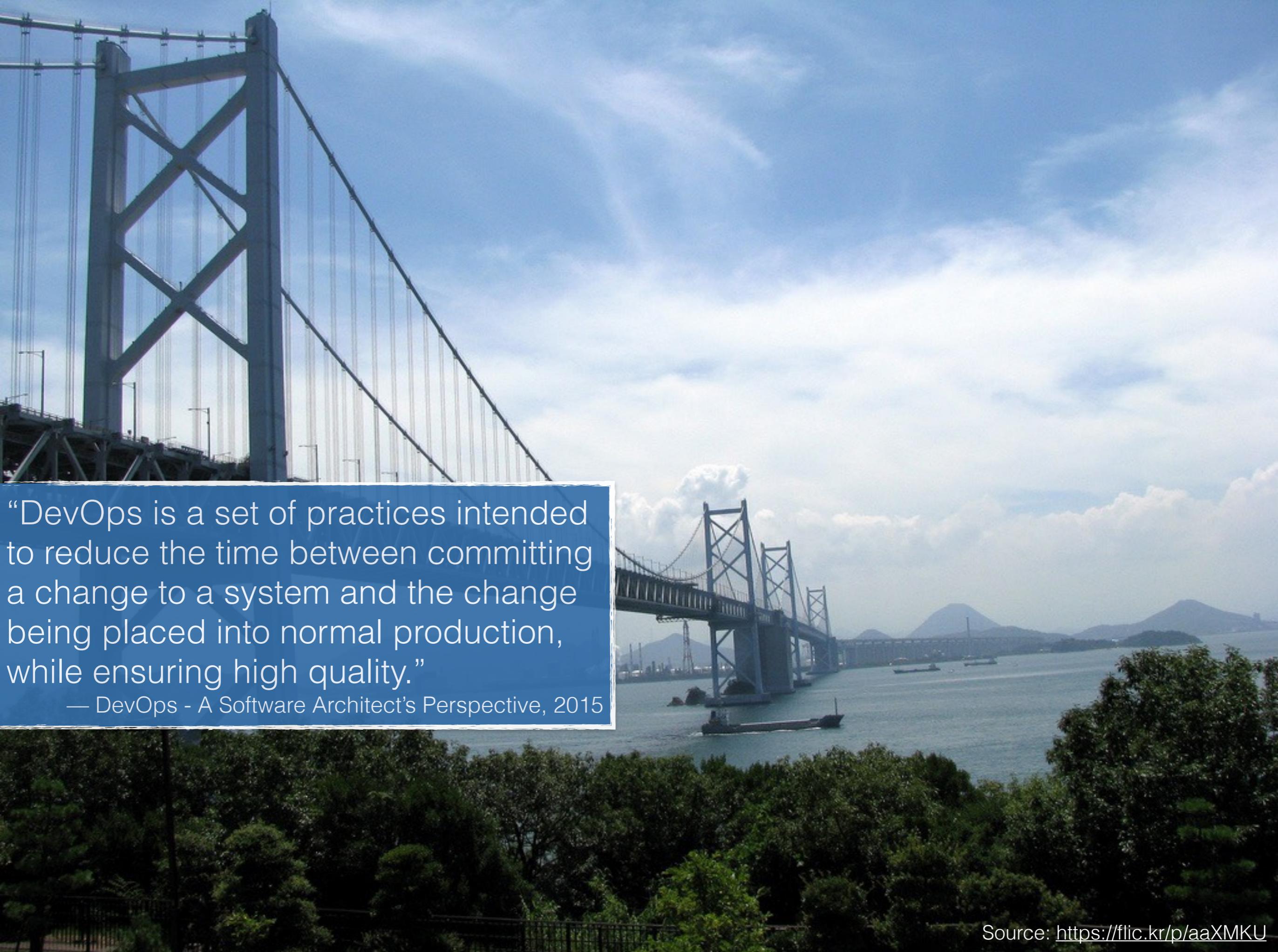


A long, green steel truss bridge spans across a river at dusk. The bridge's structure is composed of numerous vertical and diagonal beams forming a series of triangles. The sky is a deep blue, and the city lights in the distance are visible. The bridge is the central focus of the image, extending from the left towards the right.

“Release engineering is a software engineering discipline concerned with the development, implementation, and improvement of processes to deploy high-quality software reliably and predictably.”

— Andrej Dyck, Towards Definitions for Release Engineering and DevOps, 2015

Release Engineering



“DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.”

— DevOps - A Software Architect's Perspective, 2015

DevOps - Benefits

Technical benefits:

- > Continuous software delivery
- > Less complex problems to fix
- > Faster resolution of problems

Business benefits:

- > Faster delivery of features (time to market)
- > More stable operating environments
- > More time available to add value (rather than fix/maintain)

DevOps - Why

Common conflicts in software development or enterprise IT:

Adopt latest technology vs. long term planning

Increasing rate of technology change, the gap between leading edge and every-one-else is growing

Moving quickly vs. stability and security

Facebook's move fast and break things

DevOps - Implications

Ensure quality of deployed change

by test automation, tests in production on a limited set of users (-> CD practices)

Ensure high velocity, thus highly automated,
repeatable delivery mechanisms

avoid downtimes, keep system available

Two important time periods

- developer commits code: change enters deployment pipeline
- code is pushed to production: change is released to users

DevOps - Practices

Treat operations as first-class citizens from the point of view of requirements

i.e., ensure that they get the monitoring data and logs they need

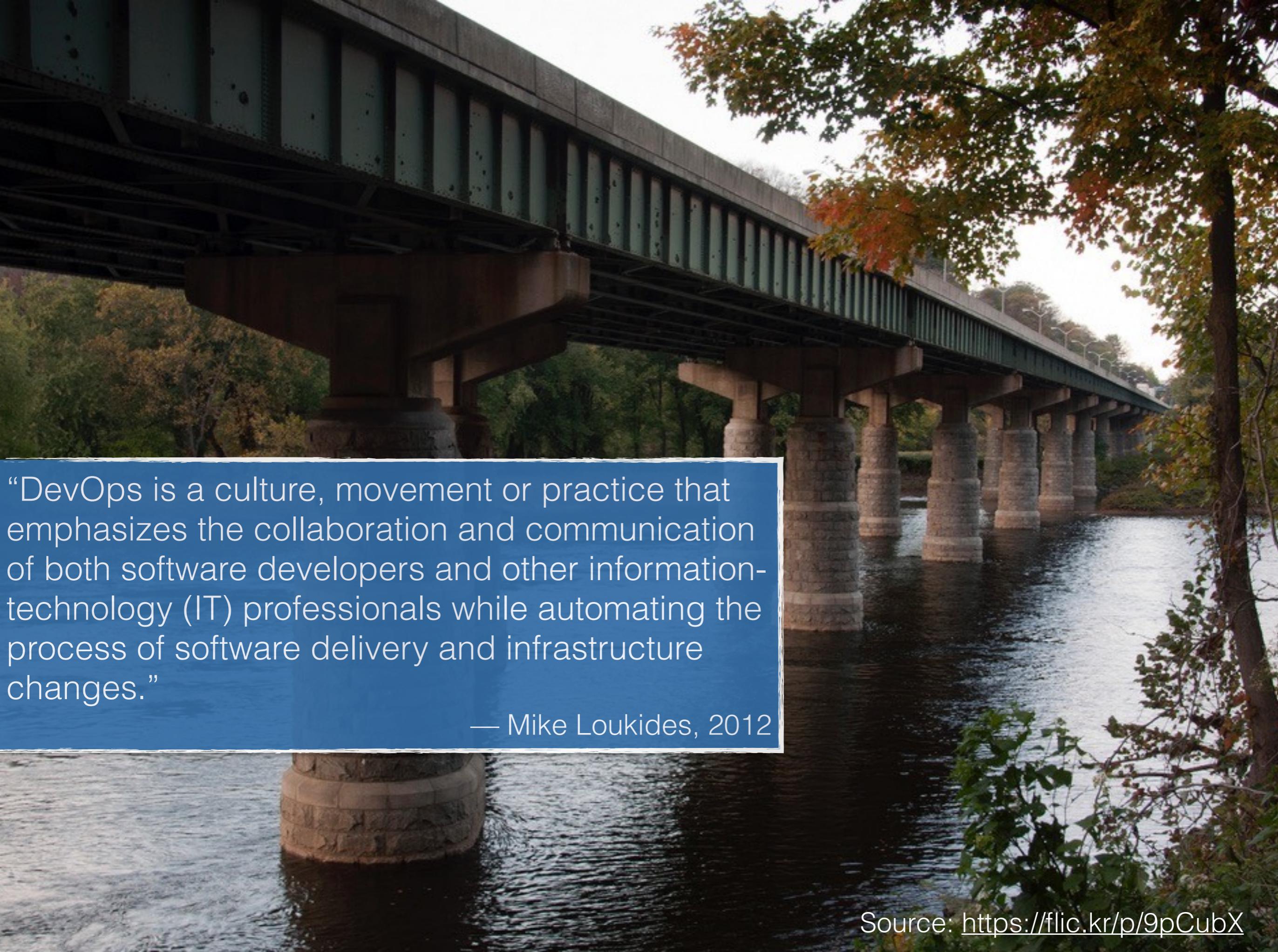
Make developers more responsible for relevant incident handling

i.e., Dev on Call, allows identifying and fixing issues faster

Use continuous delivery/deployment

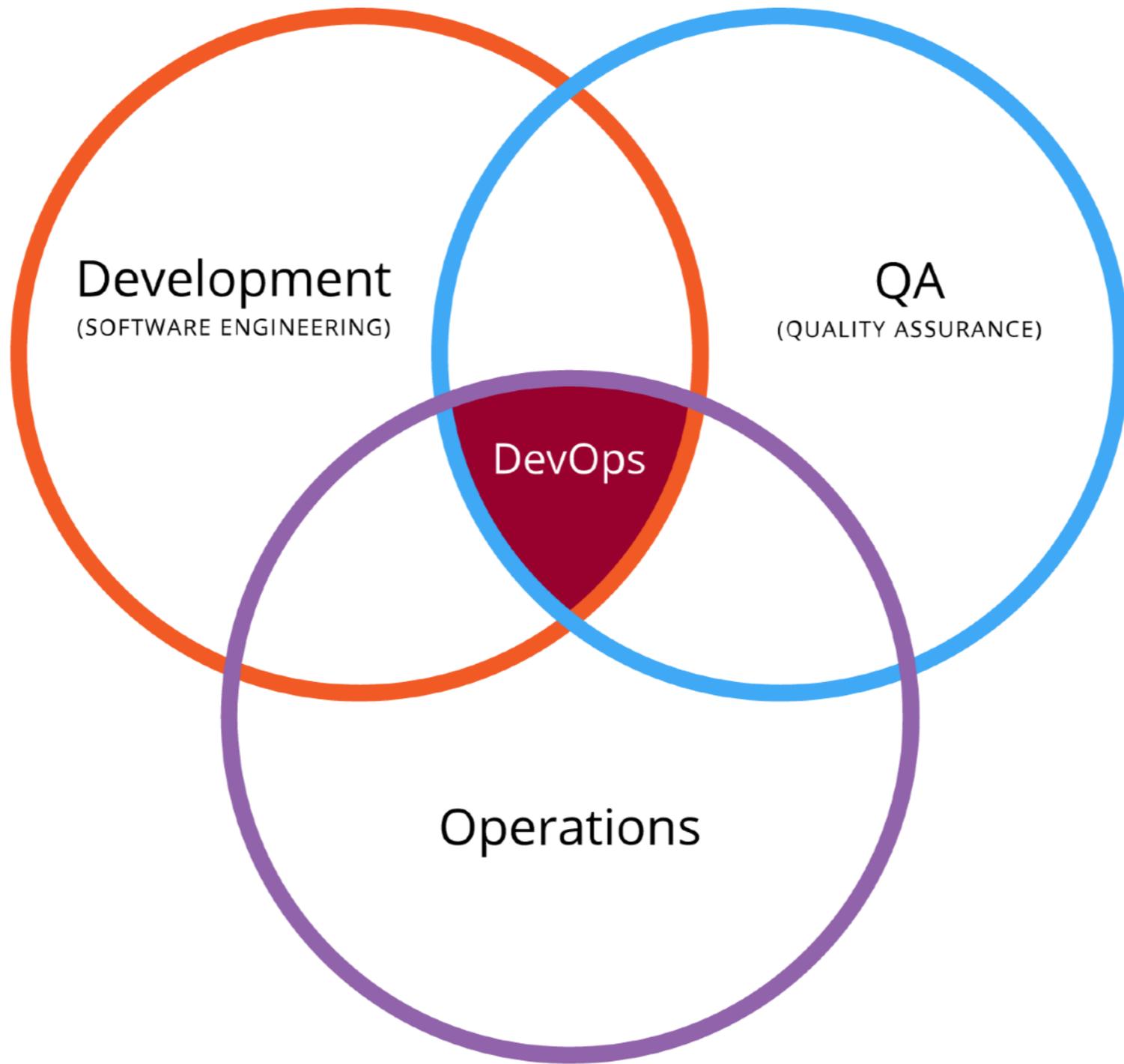
Develop infrastructure code, such as deployment scripts, with the same set of practices as application code

i.e., Infrastructure-as-Code, allows version-control and mitigates misconfiguration

A long bridge with green steel beams and stone pillars spans across a river. The bridge is supported by several stone pillars. The water is dark and reflects the bridge. There are trees with autumn-colored leaves on the right side of the river. The sky is overcast.

“DevOps is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.”

— Mike Loukides, 2012



Development
(SOFTWARE ENGINEERING)

QA
(QUALITY ASSURANCE)

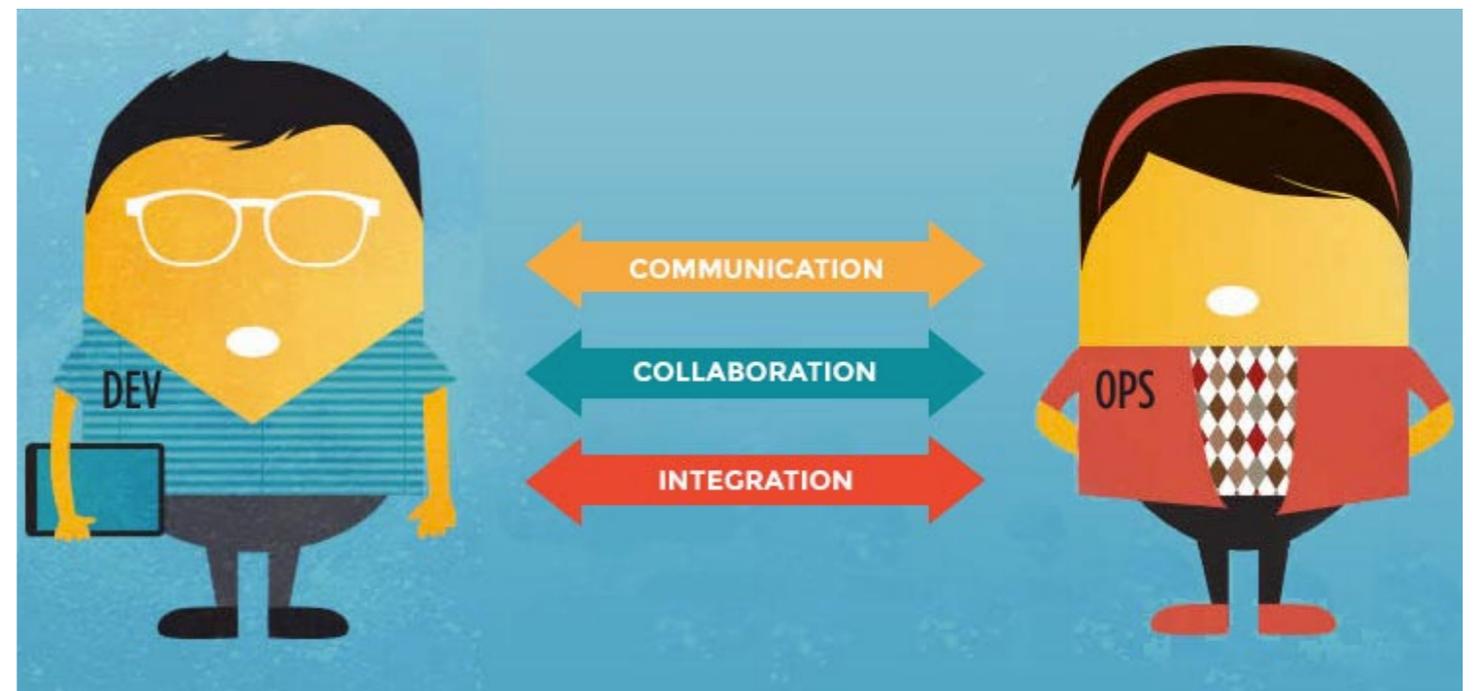
DevOps

Operations

DevOps

DevOps is not purely technical, includes several aspects summarized under **CALMS**:

- > Culture
- > Automation
- > Lean
- > Measurement
- > Sharing



DevOps - CALMS

Culture

teams over individuals
cross-functional teams instead of “silos”
embrace change & experimentation



DevOps - CALMS

Culture

teams over individuals
cross-functional teams instead of “silos”
embrace change & experimentation

Automation

Continuous Delivery / Deployment
Infrastructure as Code

Lean

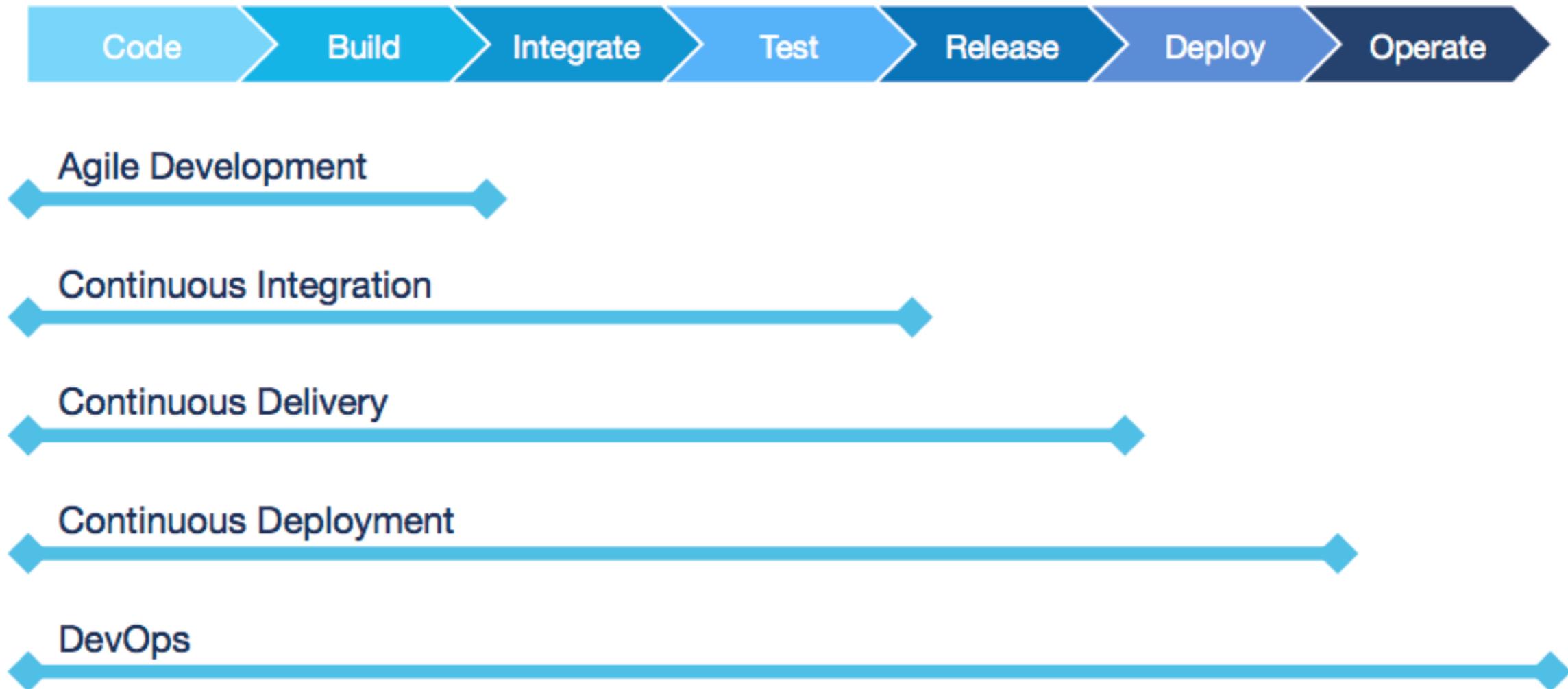
be minimalistic (meeting numbers and times, team sizes, ...)
focus on producing value for the end user

Measurement

collect data on everything
ensure to provide visibility into all systems and events (e.g.,
dashboards)

Sharing

collaboration & communication instead of “throwing things
over the fence”
not just reporting facts, regular exchange of ideas







“Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day.”

— Martin Fowler

Source: <https://flic.kr/p/aoqZry>

CI - Why

In many software projects, for long periods of time the application is **not in a working state**. Usually no one is interested in running the whole application until it is finished.

Drawbacks:

- > Integration is a long and unpredictable process
e.g., may take weeks or even months
- > Testing happens late in the process, thus bugs being often detected late
- > Delayed and infrequent releases
- > Poor project visibility, not clear what is actually working
- > High maintenance costs

Continuous Integration

CI requires that **every time** somebody **commits any change**, the entire **application is built** and a comprehensive set of **automated tests is run** against it. If the **build or test process fails**, the dev team **fixes** the problem **immediately**.

CI is a paradigm shift:

- > Without CI, software is broken until somebody proves it works
- > With CI, software is proven to work with every new change. The moment it breaks, you fix it.

Teams with CI are able to

- > deliver software faster
- > with fewer bugs and
- > bugs are caught earlier in the process when they are cheaper to fix
- > providing cost and time savings

CI - Prerequisites

> Version control



> Automated build -
able to build the application
from the command line

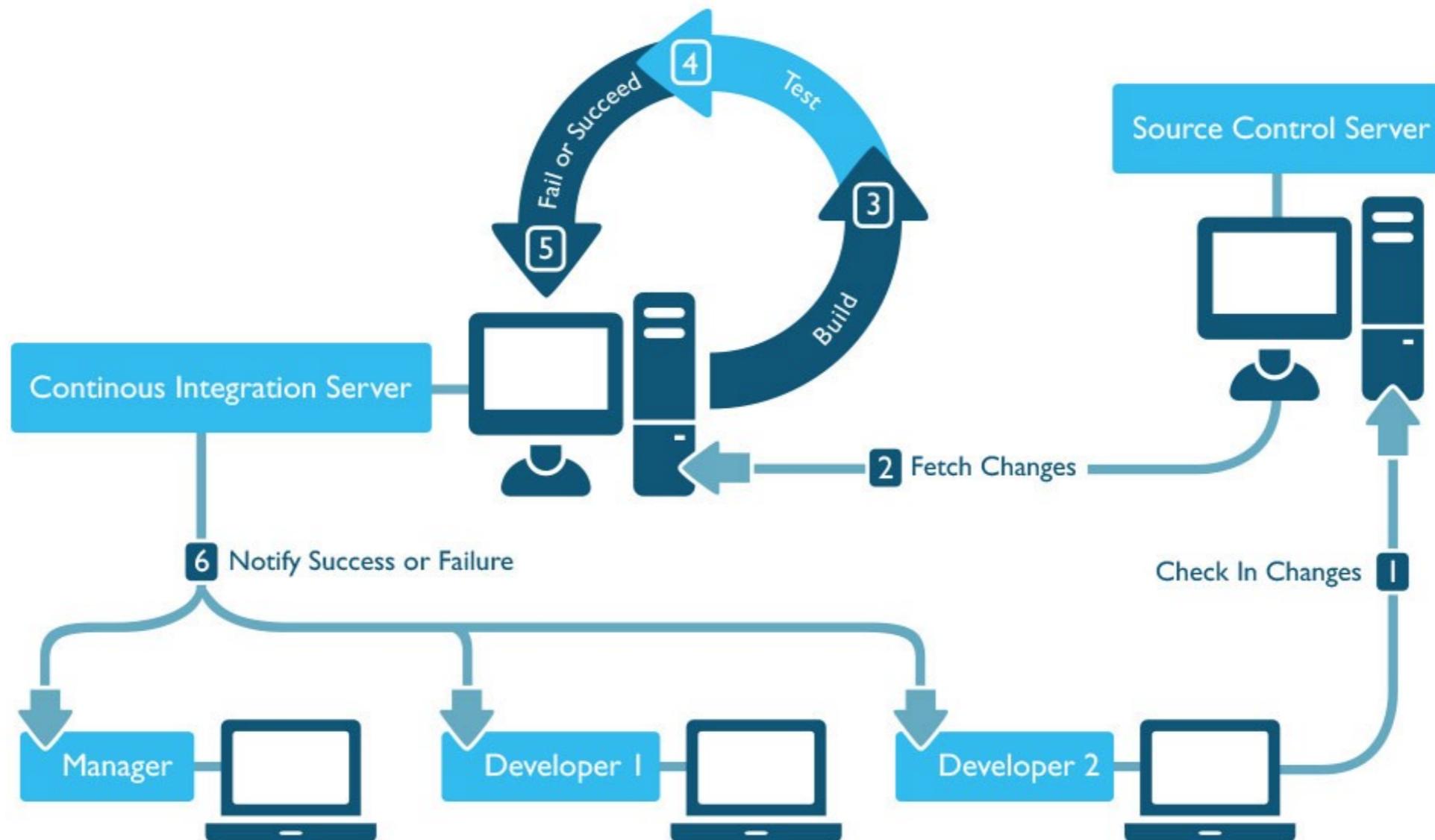
maven



> Automated test suite
unit tests, integration tests, ...

> Agreement and discipline across the teams, frequently
check in small changes, fix failed builds before
committing further changes

CI - Overview



CI - Tools



Jenkins



circle**ci**



GitLab

 **Bamboo**



CODESHIP



Travis CI

-  Zurück zum Projekt
-  Status
-  Änderungen
-  **Console Output**
-  View as plain text
-  Build-Informationen editieren
-  Build löschen
-  Git Build Data
-  No Tags
-  Vorheriger Build

Konsolenausgabe

```
Started by user group07
Building in workspace /var/lib/jenkins/jobs/sopra-fs16-group7/workspace
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from 2 remote Git repositories
> git config remote.heroku.url git@heroku.com:sopra-fs16-group7.git # timeout=10
Fetching upstream changes from git@heroku.com:sopra-fs16-group7.git
> git --version # timeout=10
using GIT_SSH to set credentials sopra-fs16-group7
> git -c core.askpass=true fetch --tags --progress git@heroku.com:sopra-fs16-group7.git +refs/heads/*:refs/remotes/heroku/*
> git config remote.origin.url git@github.com:sealuzh/sopra-fs16-group7-server.git # timeout=10
Fetching upstream changes from git@github.com:sealuzh/sopra-fs16-group7-server.git
using GIT_SSH to set credentials sopra-fs16-group7
> git -c core.askpass=true fetch --tags --progress git@github.com:sealuzh/sopra-fs16-group7-server.git +refs/heads/*:refs/remotes/origin/*
Seen branch in repository heroku/master
Seen branch in repository origin/GameStateViews
Seen branch in repository origin/InitializeGameService
Seen branch in repository origin/OptimizedCarriageInitialization
Seen branch in repository origin/banditChooser
Seen branch in repository origin/cardOrder
Seen branch in repository origin/cardRelation
Seen branch in repository origin/datamodel
Seen branch in repository origin/dev
Seen branch in repository origin/facade
Seen branch in repository origin/feature/sonar-integration
Seen branch in repository origin/gameRelation
Seen branch in repository origin/initGameChanges
Seen branch in repository origin/jsonViewTest
Seen branch in repository origin/lobby
Seen branch in repository origin/lootService
Seen branch in repository origin/marshal
Seen branch in repository origin/master
Seen branch in repository origin/playerAPI
Seen branch in repository origin/playerService
Seen branch in repository origin/playerViews
Seen branch in repository origin/testUtil
Seen branch in repository origin/toString
Seen 23 remote branches
Checking out Revision cbe5a542053c17e2086cdae5bdf83a50e11c14aa (origin/master, heroku/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f cbe5a542053c17e2086cdae5bdf83a50e11c14aa
> git rev-list cbe5a542053c17e2086cdae5bdf83a50e11c14aa # timeout=10
> git rev-list cbe5a542053c17e2086cdae5bdf83a50e11c14aa # timeout=10
[Gradle] - Launching build.
[workspace] $ /var/lib/jenkins/jobs/sopra-fs16-group7/workspace/gradlew test
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE

BUILD SUCCESSFUL

Total time: 7.516 secs
```

- Executed Gradle Tasks
- [compileJava](#)
 - [processResources](#)
 - [classes](#)
 - [compileTestJava](#)
 - [processTestResources](#)
 - [testClasses](#)
 - [test](#)
 - [compileJava](#)
 - [processResources](#)
 - [classes](#)
 - [compileTestJava](#)
 - [processTestResources](#)
 - [testClasses](#)
 - [test](#)
 - [sonarqube](#)





“Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.”

— Martin Fowler

Continuous Delivery

- > based on CI
- > CI focuses on development teams, the output of CI is the input to manual testing process and to the rest of the release process
- > much of time is wasted on this way through testing and operations (e.g., testers wait for “good” builds of the software, operations teams wait for documentation or fixes)
- > software sometimes undeployable because it takes too long to get it into production-like environments and because of a slow feedback loop between dev, QA, and operations

CD prevents this by:

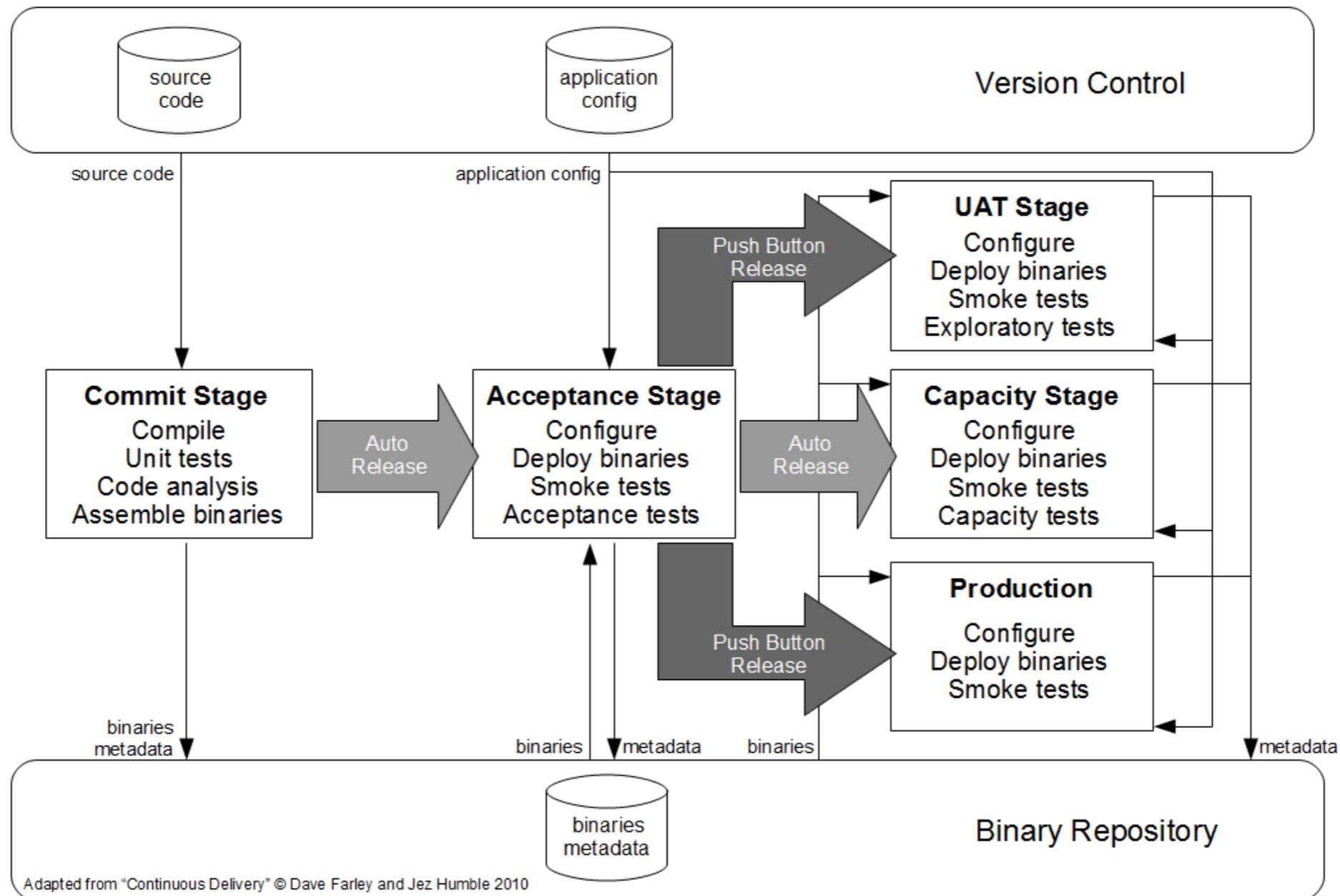
close, collaborative working relationship between the stakeholders involved in delivery

==> DevOps culture

extensive automation of all parts of the delivery process

==> CD pipeline

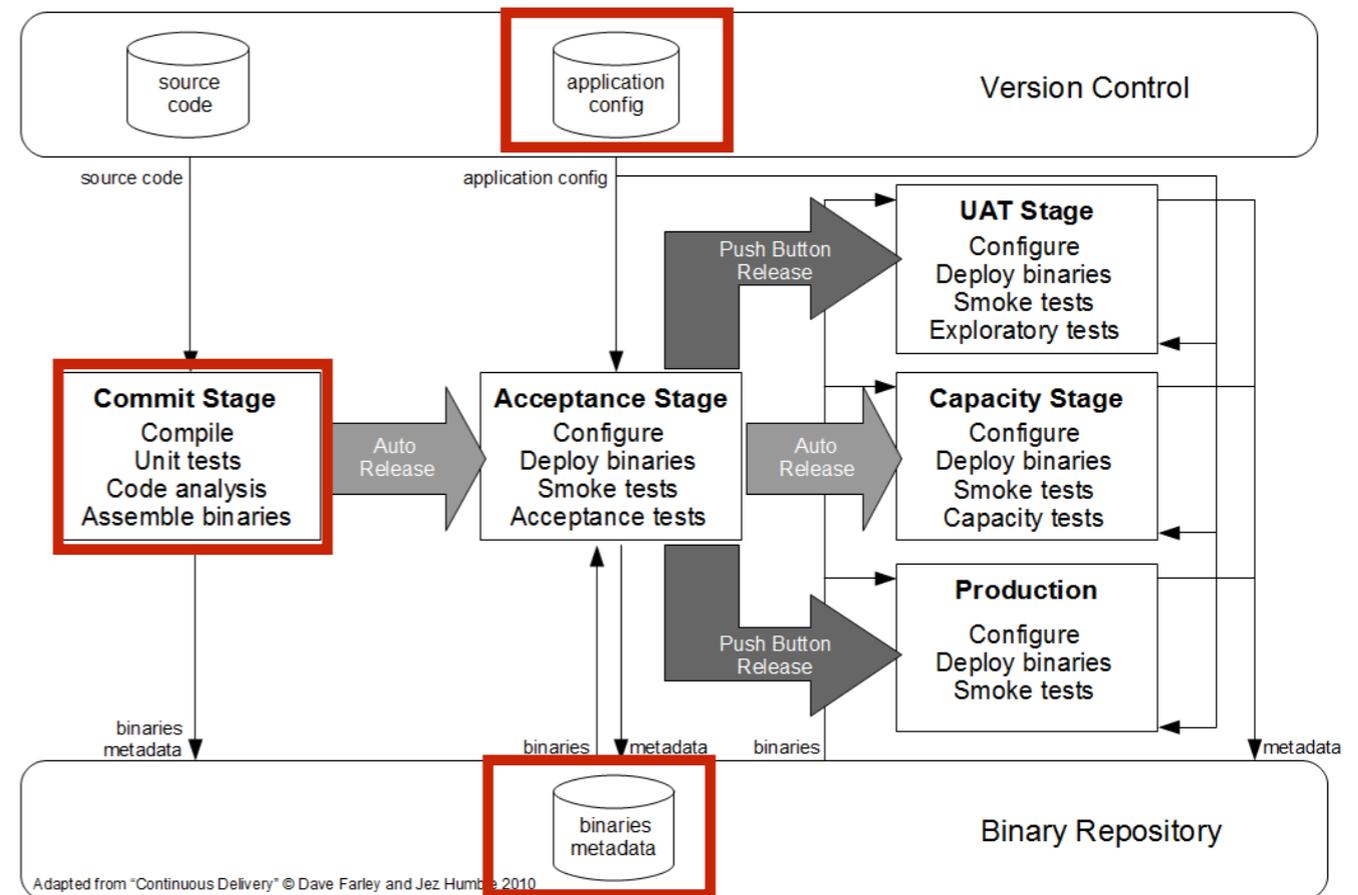
Continuous Delivery Pipeline



Continuous Delivery Pipeline

CI Phase:

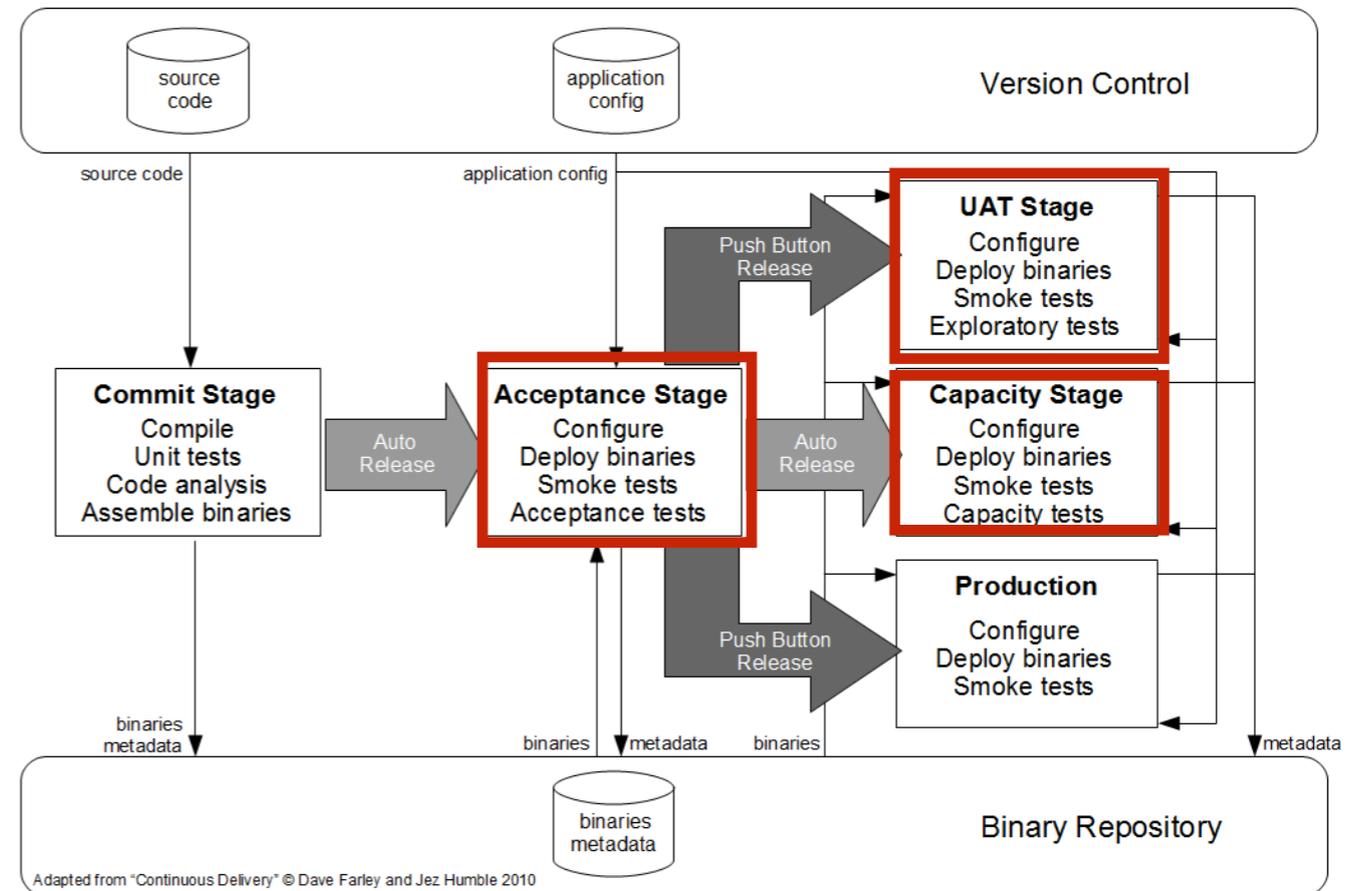
- > build application only once throughout release process
- > store resulting binary in repository
- > reuse binary for all following environments (internal test stages as well as production)
- > platform/environment specific configurations for binaries are version controlled
- > binary repos including config files allow rolling back to specific previous versions in case of issues without the need of recompilation



Continuous Delivery Pipeline

Post CI Phases:

- > consist of multiple environments/stages for testing purposes
- > include manual and automated testing
- > some stages might run in parallel (e.g., user acceptance testing (UAT) on UI and performance tests on backend systems)
- > stages are either automatically executed when previous stage is finished, or after manual approval (e.g., push of a button)



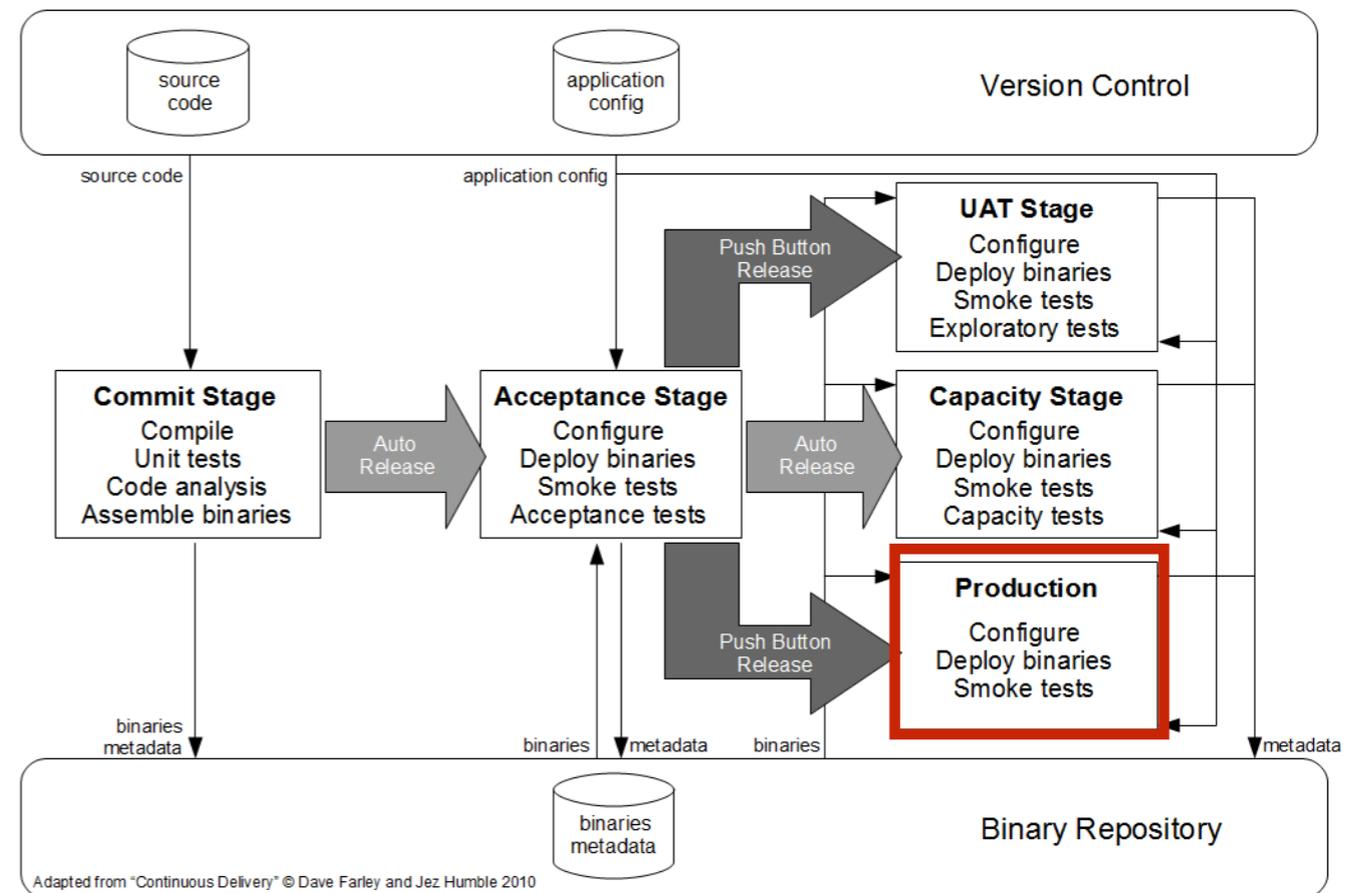
Continuous Delivery Pipeline

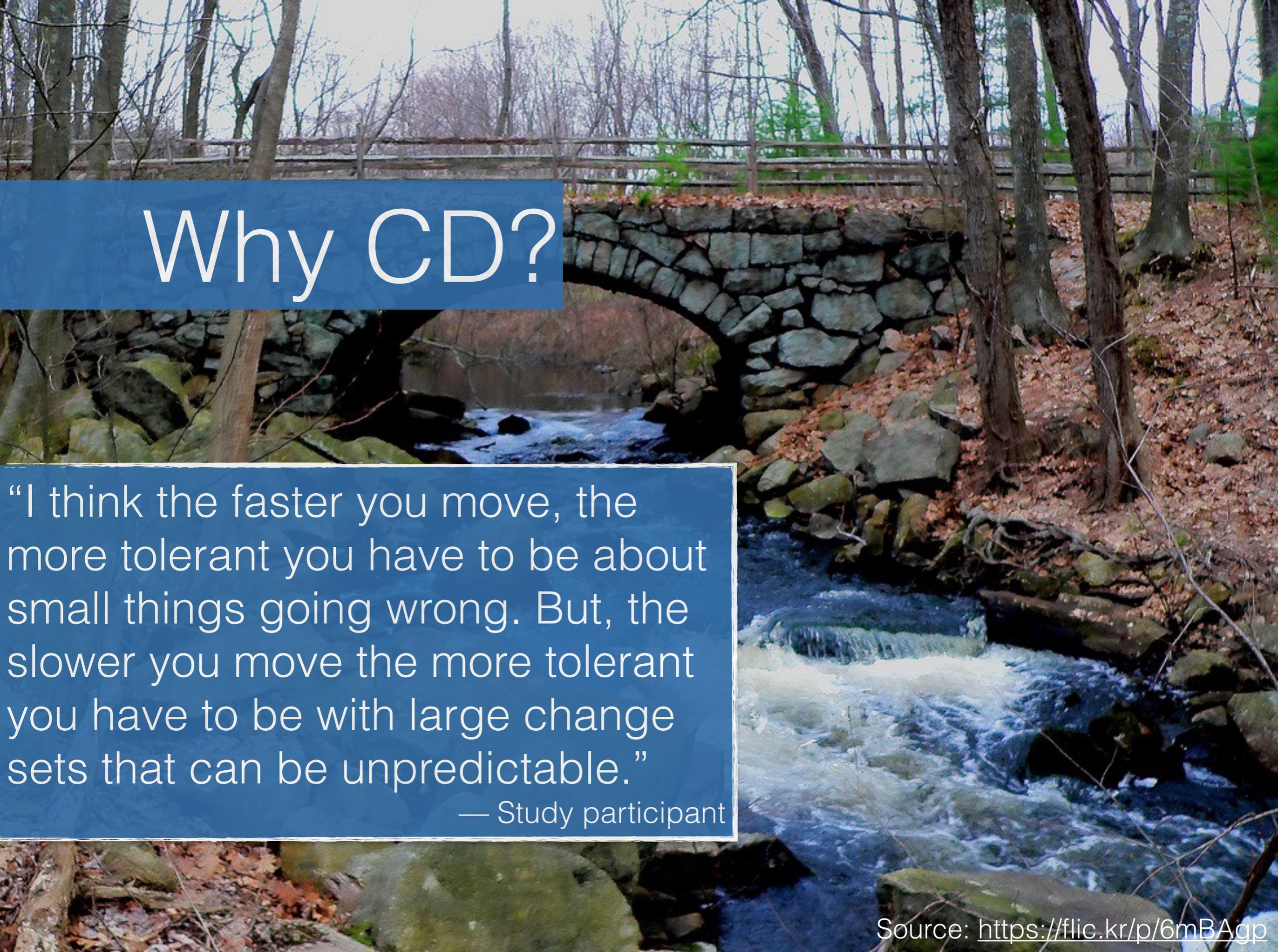
Deployment Phase:

> new version is available to be released to customers

> various strategies and practices exist for rolling out:

- canary release
- A/B testing
- dark launches
- gradual rollouts
- blue/green deployments



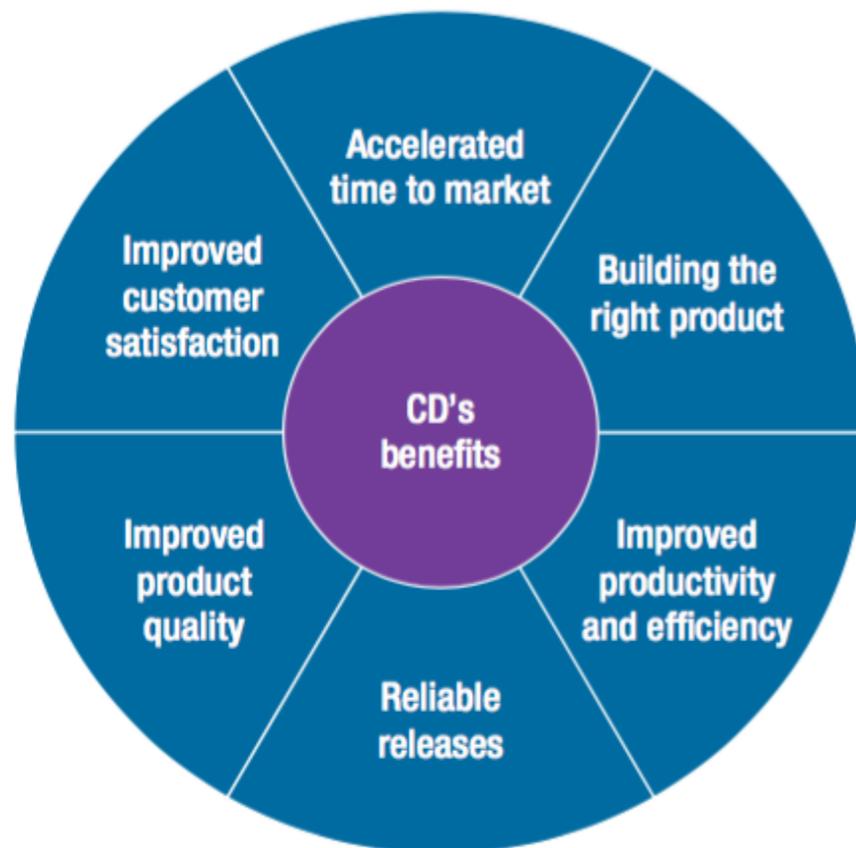
A photograph of a stone arch bridge spanning a stream in a forest. The ground is covered with fallen brown leaves, and the trees are mostly bare, suggesting an autumn or winter setting. The bridge is constructed from large, grey stones. The stream flows through the center of the arch, with some white water rapids visible downstream.

Why CD?

“I think the faster you move, the more tolerant you have to be about small things going wrong. But, the slower you move the more tolerant you have to be with large change sets that can be unpredictable.”

— Study participant

CD Benefits



- > instead of releasing 3-4 times a year, releasing on a weekly or daily basis allows companies **staying ahead of competition**
- > more frequent releases means obtaining user feedback more quickly, thus **building the right product**
- > setting up a CD pipeline and automation leads to **increased productivity and efficiency**
- > releases become routine, thus **more reliable** instead of having stressful releases 3-4 times a year
- > highly automated quality checks **preserve human-caused errors** and increase product quality
- > having reliable, frequent releases incorporating customer feedback leads to **customer satisfaction**

CD - How to

There is no “holy grail” for CD, usually this is a slow process and involves multiple challenges companies are faced with:

Organizational

- > domain constraints and traditional processes (e.g., manual approvals)
- > top-management must be convinced (added value of CD), rather high costs for automation
- > suppliers that rely on the company's software

Technical

- > legacy applications
- > application architecture
- > ensure product quality, automation (test and deployment)
- > dealing with higher risks

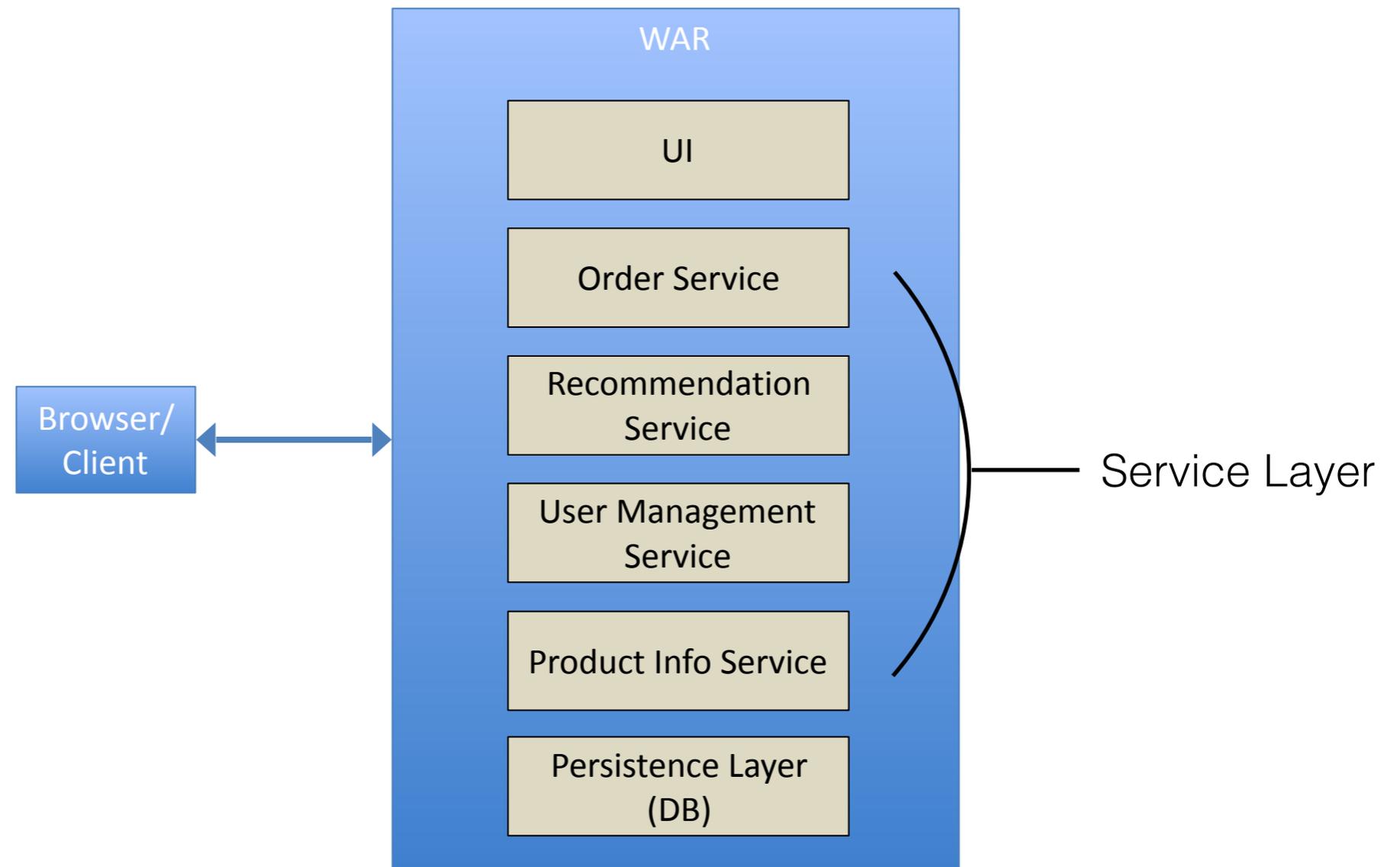
Social

- > resistance to change
- > requires strong collaboration between teams, break down barriers and promote collaborative culture
- > changing responsibilities (e.g., dev on call, code ownership)

DETOUR



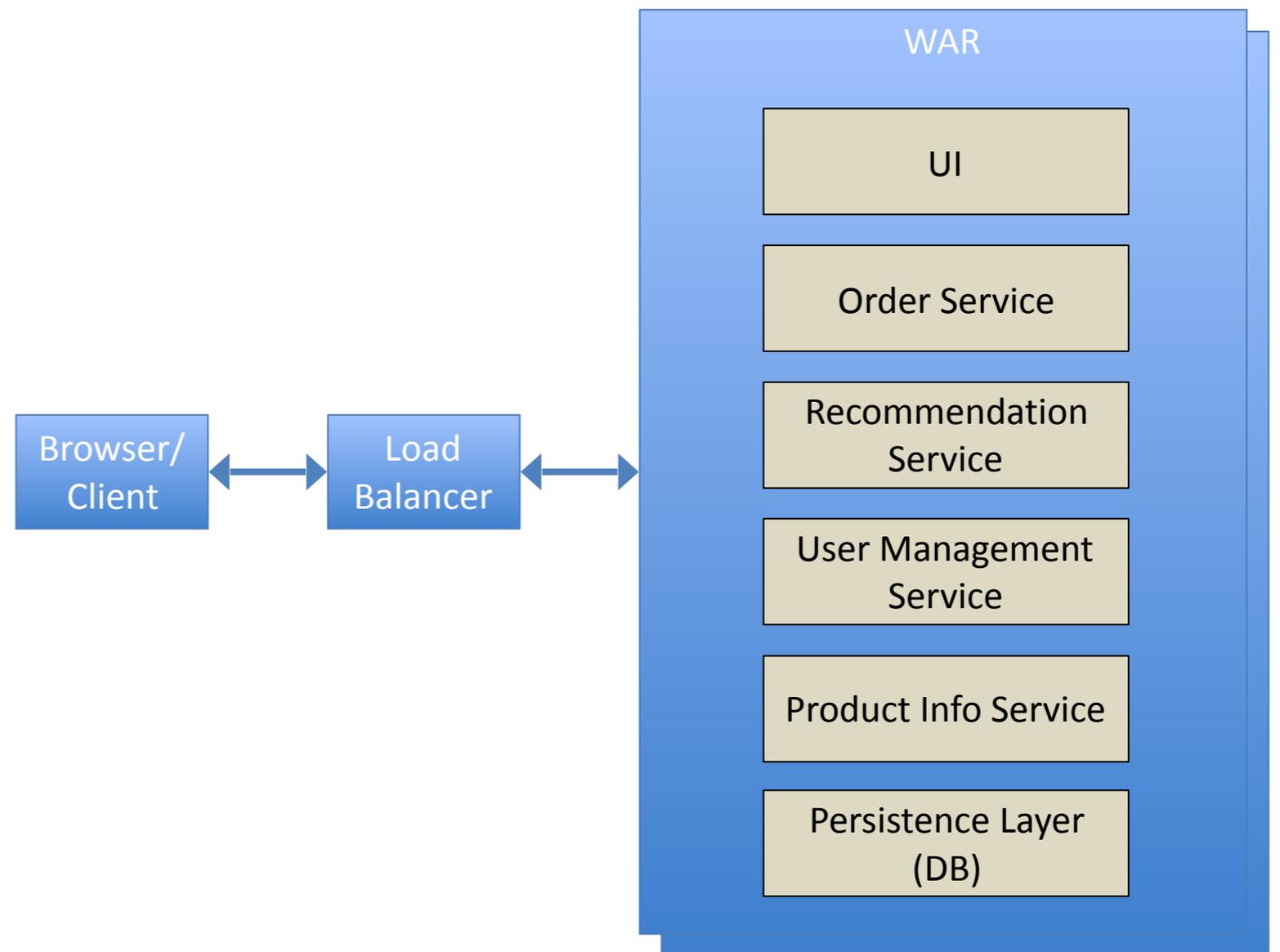
Traditional 3-tier Web Application Architecture - Example



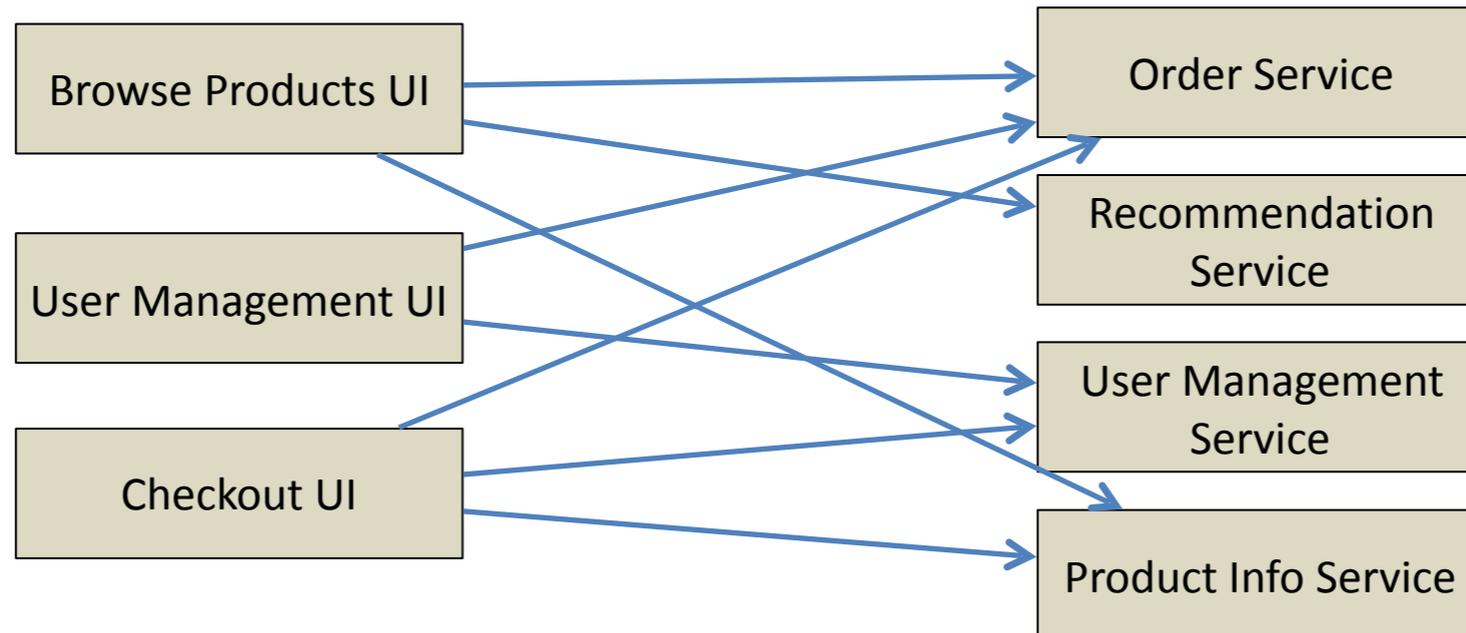
Traditional 3-tier Web Application Architecture - Example

Downsides:

- > Scaling
- > Every change requires redeployment of whole application



(Micro-)services Architecture

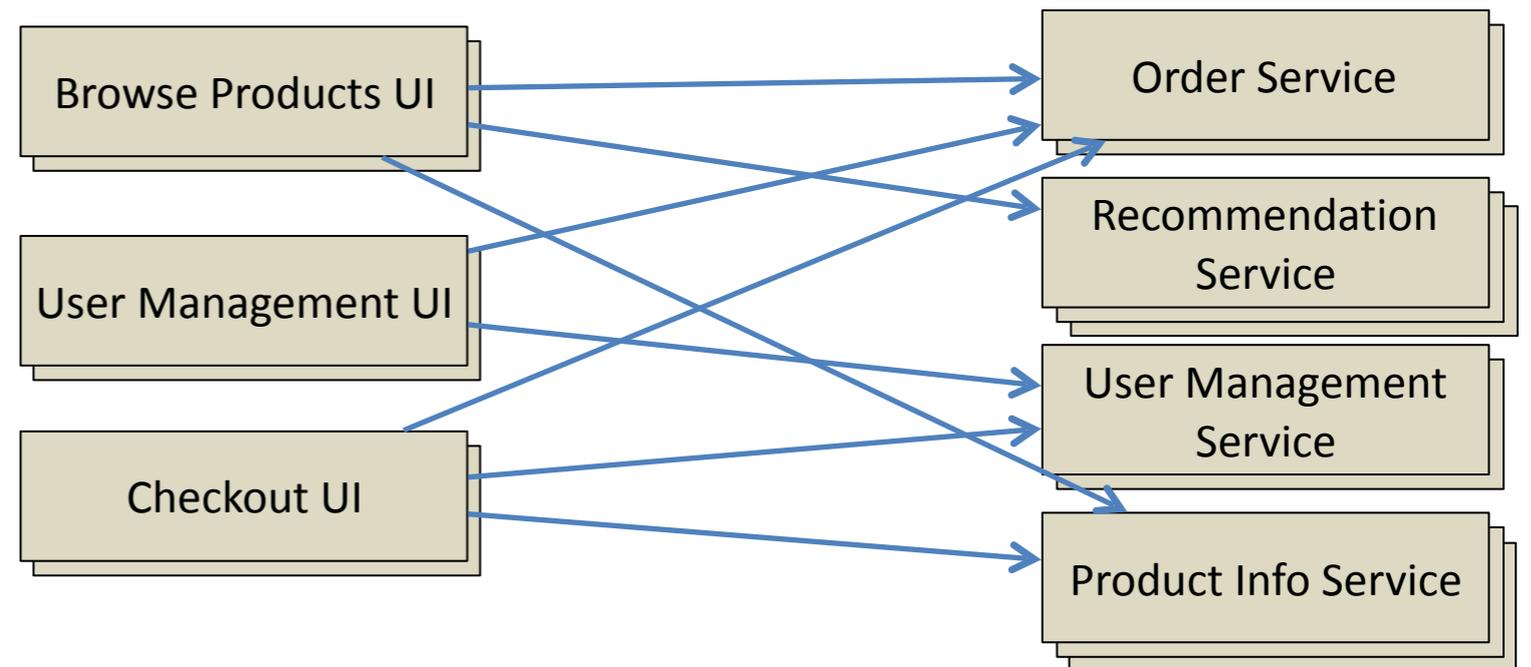


(Micro-)services Architecture

Advantages:

- > scale on service level
- > deploy new versions of single services
- > run multiple versions at the same time (-> CD rollout practices)
- > services communicate using standardized communication protocols (e.g., HTTP), thus single services can be based on the technology which fits best

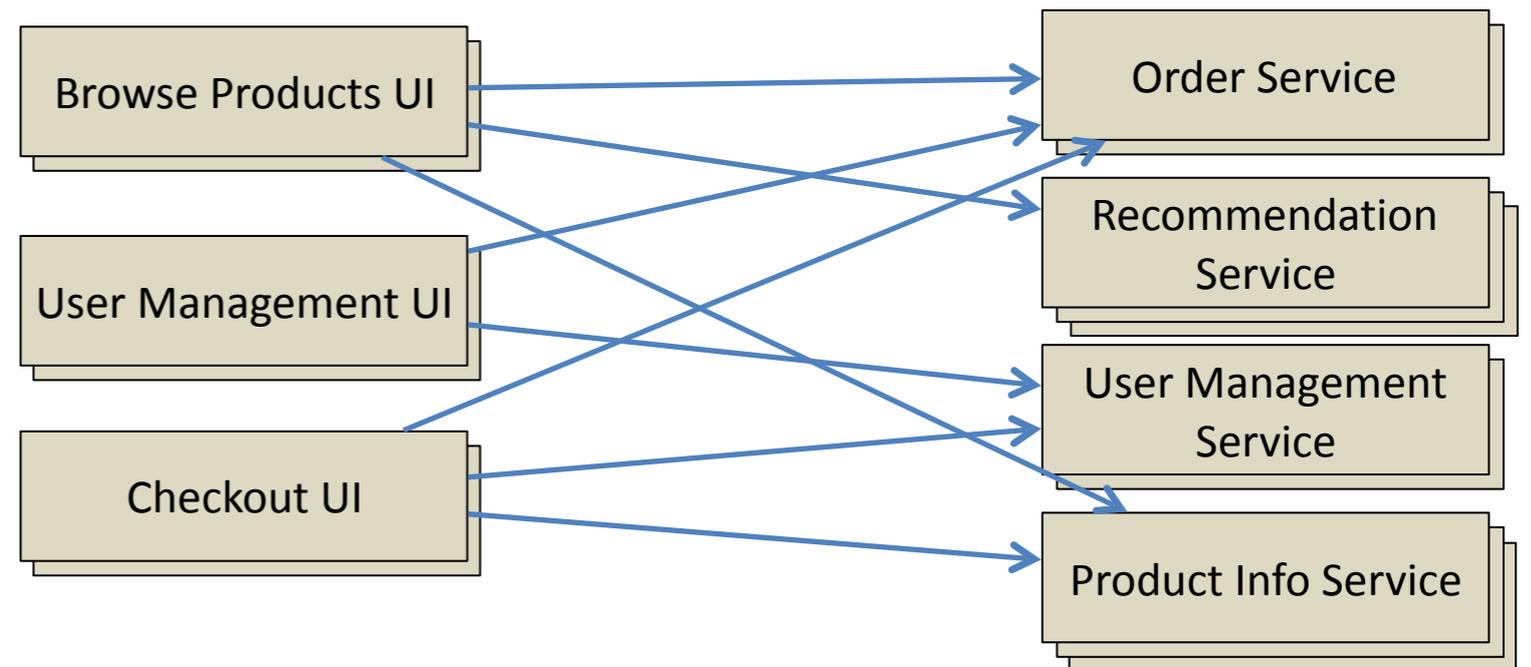
In DevOps, teams are often responsible for single services (both development and operating them)



(Micro-)services Architecture

Downsides:

- > higher communication costs
- > hard to start from scratch with microservices, usually functionality is broken down once it gets too complex





Continuous Delivery or Continuous Deployment?

Continuous deployment is the “next step” of continuous delivery: every change that passes the automated tests is deployed to production **automatically**.

Continuous delivery doesn't mean every change is deployed to production. It means every change is proven to be **deployable at any time**. However, the decision to release a new version is made by a **human**.

Live Experimentation



Live Experimentation

Releasing on a weekly or even daily basis implies a higher **risk that something might go wrong**.

However, the change is “smaller”, thus problems can be identified and fixed faster.

Live experimentation and rollout practices can be seen as additional **risk mitigation strategies**, e.g., test a new feature on a small user group first, then decide about further rollouts.

Live Experimentation Practices

- > Eat your own dog food
- > Canary releases
- > Gradual rollouts
- > A/B testing
- > Dark launches
- > Blue/green deployments

Eat your own dog food

facebook.

Facebook employees use the newest version internally, if everything is as expected they roll it out to further users

“Eating your own dog food, also called dogfooding, is a slang term used to reference a scenario in which a company uses its own product to test and promote the product.”

— wikipedia



Canary Releases

Idea: release a new version to a subset of users first, while all other users still see the old, stable version

In case of issues with new version, only a small amount of users is affected, thus the impact of the issue is kept small

Canary is compared to the existing version in terms of a set of criteria such as stability, performance, or correctness



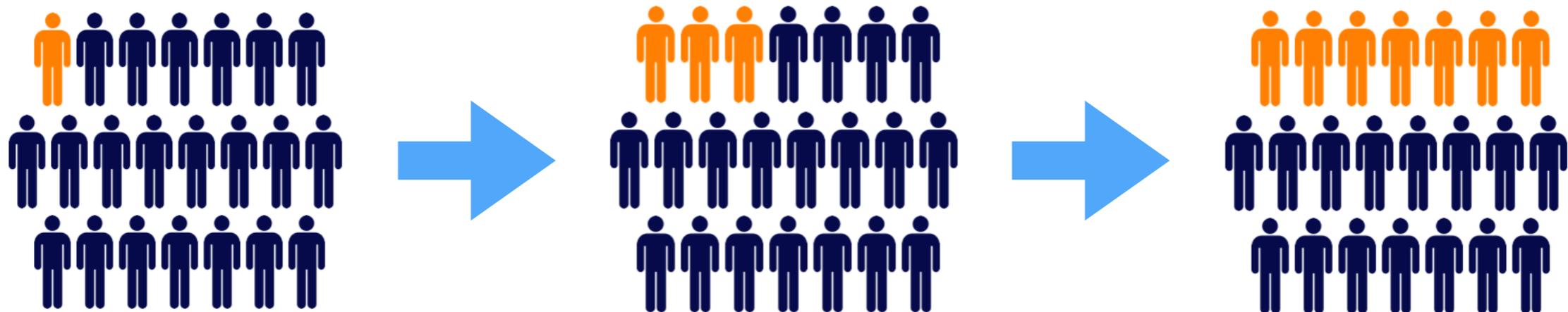
- User selection based on:
- (geographic) location
 - role (admin, early, stable, etc.)
 - random basis (e.g., 1% of traffic)

Gradual Rollouts

Idea: gradually/stepwisely increase the amount of users assigned to the newest version until it completely replaces the former version

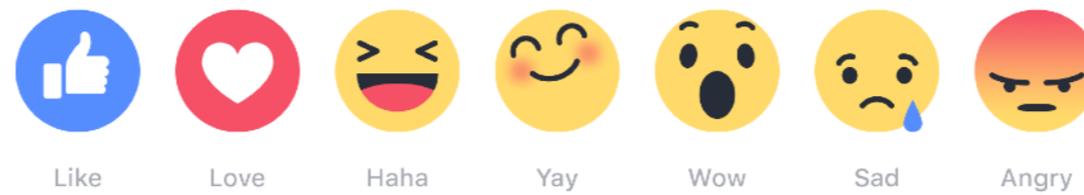
shows whether the new version can cope with increasing load and scales correctly

often used in combination with canary releases

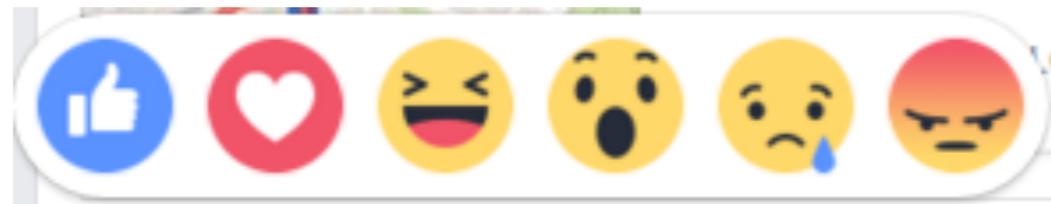


Canary / Gradual Example

How they started in Spain and Ireland:



How they rolled it out globally (after months):



Dark Launches

Idea: Mitigate performance and reliability issues of new features when facing production-like load levels by deploying those features on production without being visible for users

Through monitoring, dev teams are able to identify and fix remaining bugs and scalability concerns before enabling the feature for users

Example: Facebook Chat

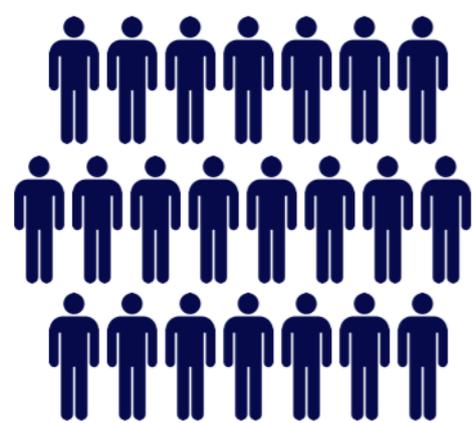
<https://www.facebook.com/notes/facebook-engineering/facebook-chat/14218138919/>

A/B Testing

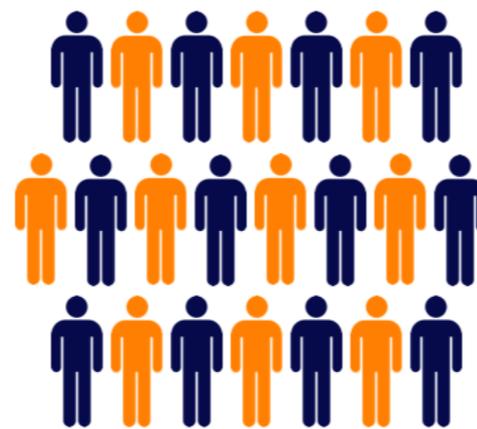
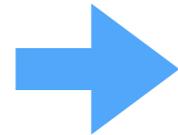
Idea: Compare two versions of software with each other, often only differentiated in one tested aspect, to determine the effect of a certain change

form of statistical hypothesis testing, thus requires big enough sample size to have “statistical power”

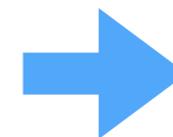
mainly used on UI for testing various layouts or design aspects



all users same version



50% users new version
50% users old version



decide about winner using collected metrics and statistics

A/B Testing



50% of users



50% of users

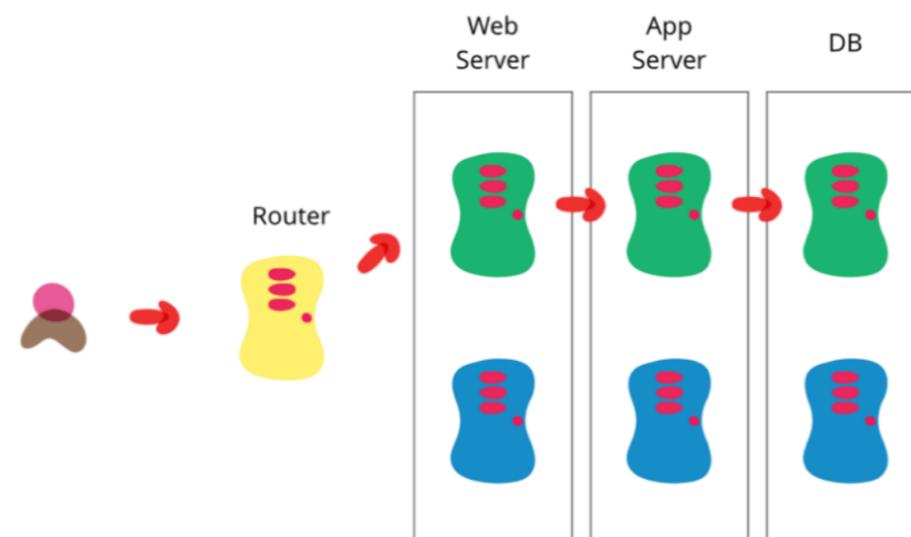
Blue/Green Deployments

Idea: Have two identical production environments, one hosting (green) the current, stable version, the other (blue) represents the final stage of testing for the new version.

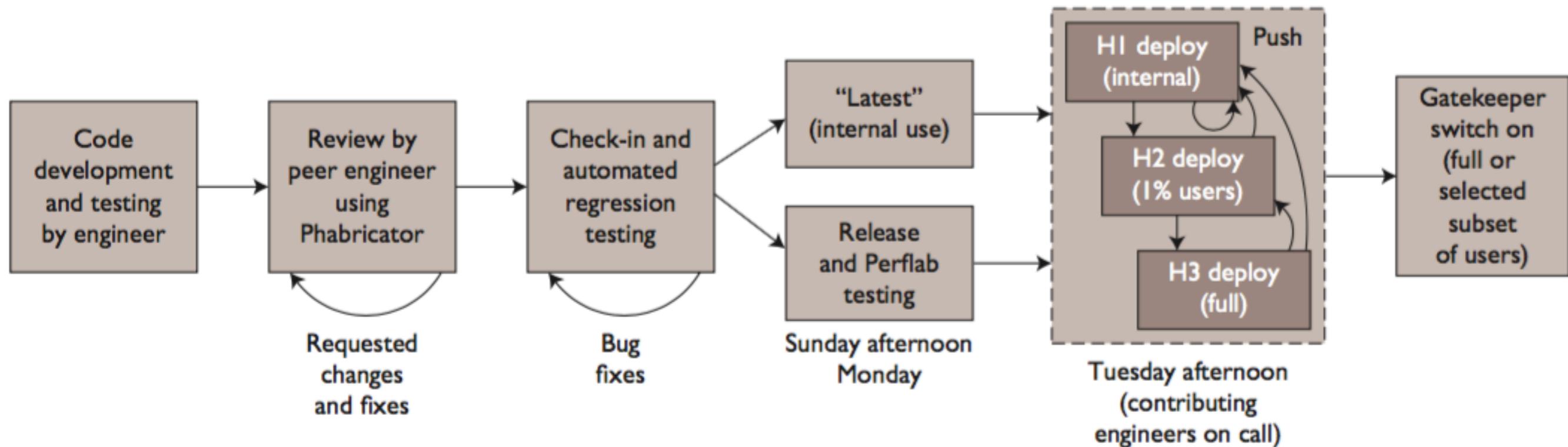
Release: Simply switch the router so that all incoming traffic goes to the blue (or green) environment and use the green (or blue) environment for testing the next version.

Advantage: In case of problems directly after the release, a quick rollback (switch) to the previous version is possible

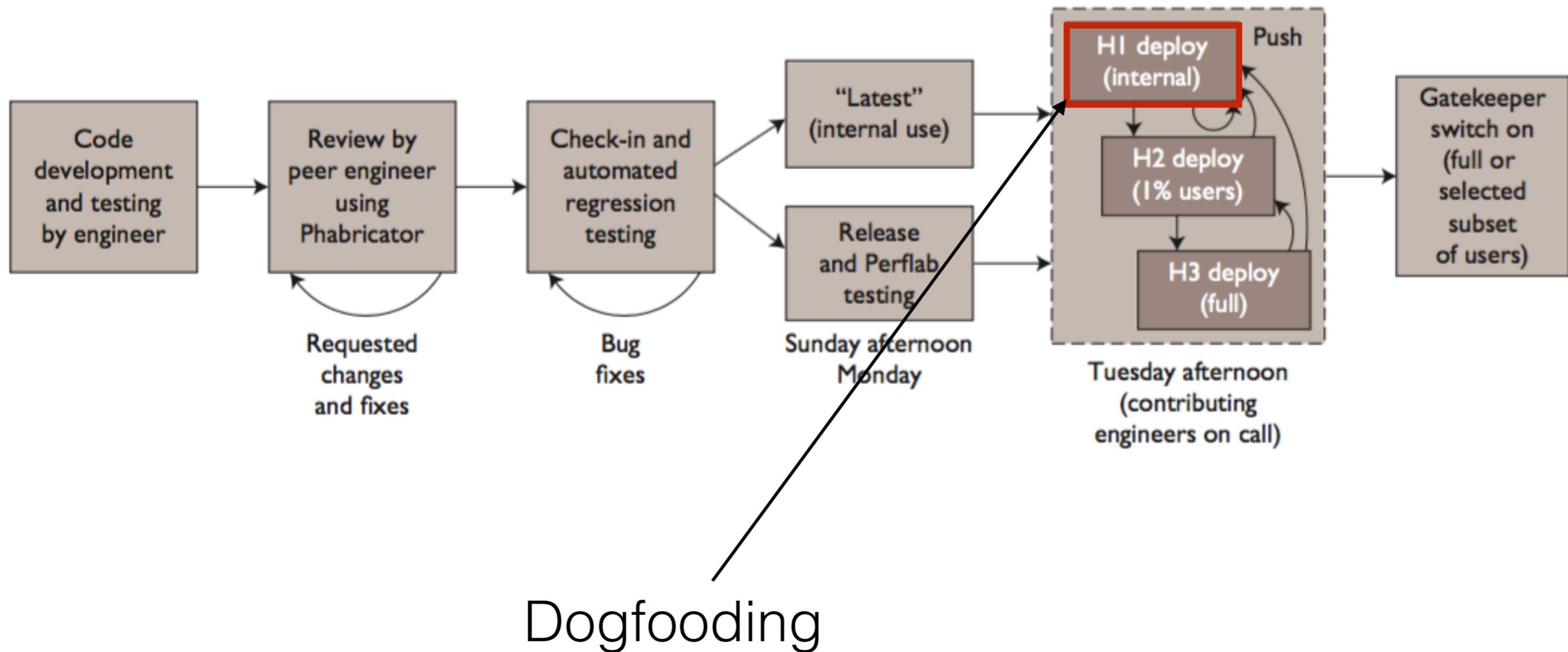
Obstacle: DBs are part of both environments, thus a switch requires DB migration/synchronization



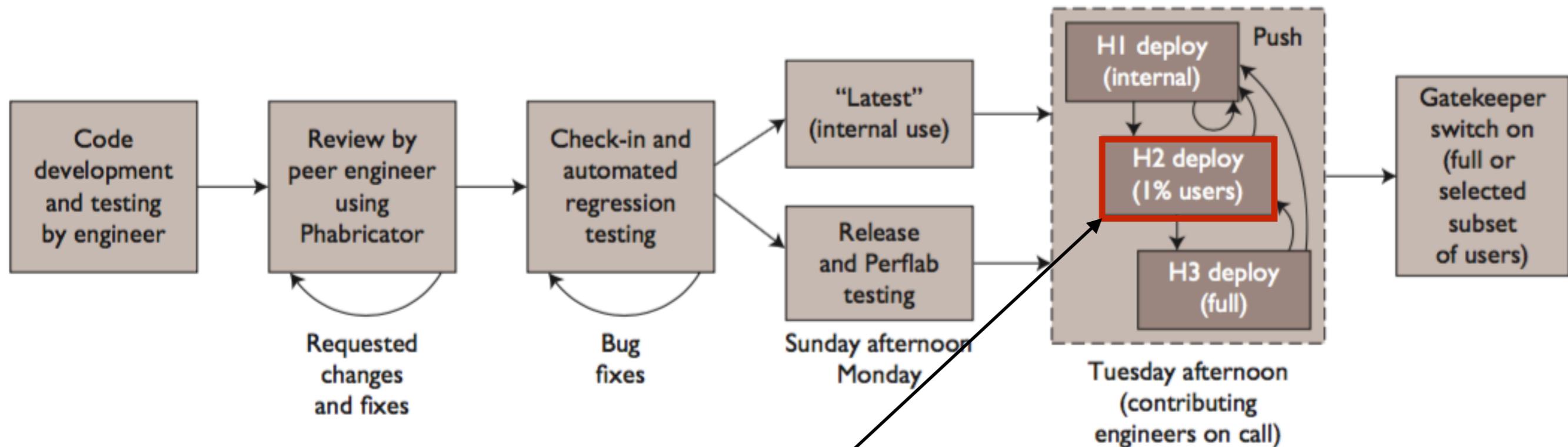
Facebook Deployment Pipeline



Facebook Deployment Pipeline

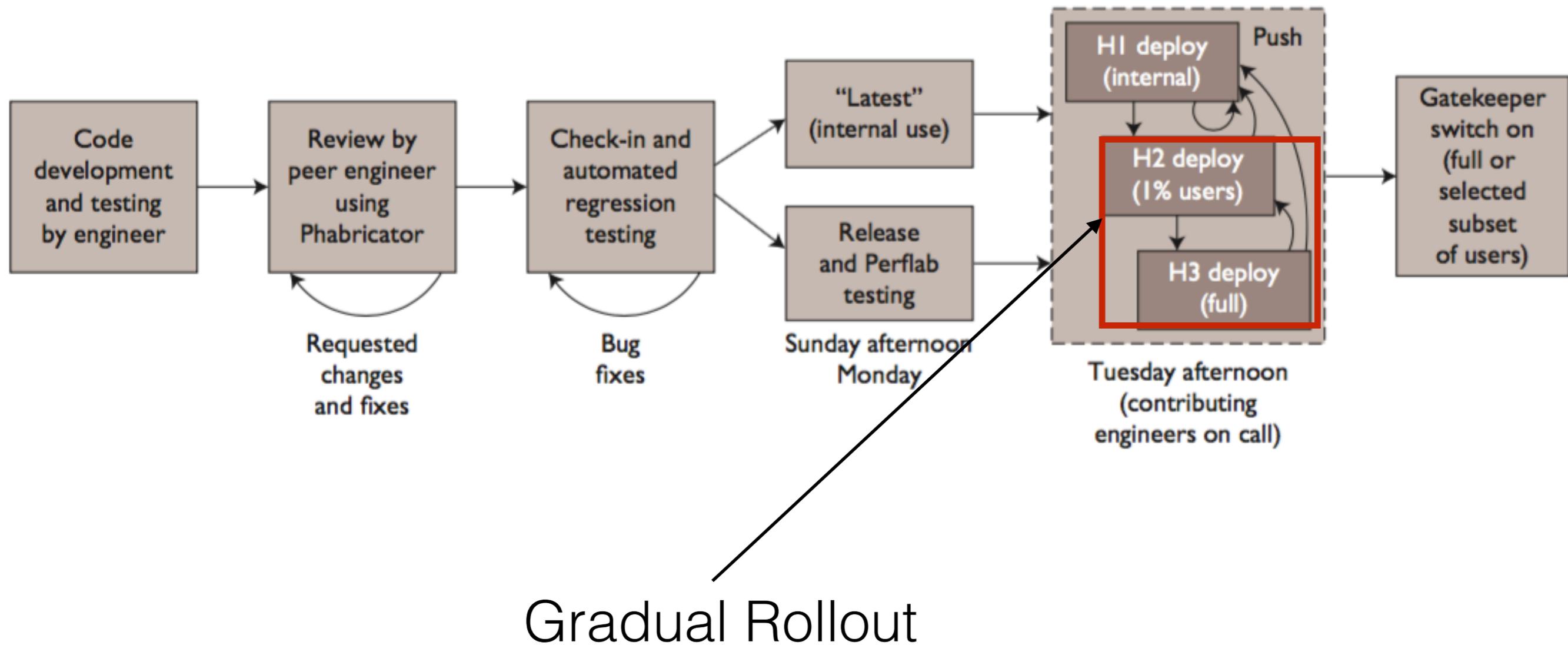


Facebook Deployment Pipeline



Canary Release

Facebook Deployment Pipeline



Monitoring

Monitoring

Monitoring is the process of observing and recording system state changes (e.g., logs) and data flows (e.g., communication between components).

Monitoring is important to

- > verify that everything runs as expected (identify failures, performance problems)
- > characterize workload (capacity planning, billing purposes)
- > support rollout decisions (e.g., increase traffic assigned to canary based on measured user reactions)
- > detecting intruders

Monitoring

Performance degradation observed by comparing current performance to historical data or by complaints from clients/users

Typical performance measures include

- > latency
- > throughput
- > utilization

measured on application and infrastructure level

Business Metrics

A set of an organization's particular set of metrics to determine the effectiveness of their offerings and their support services.

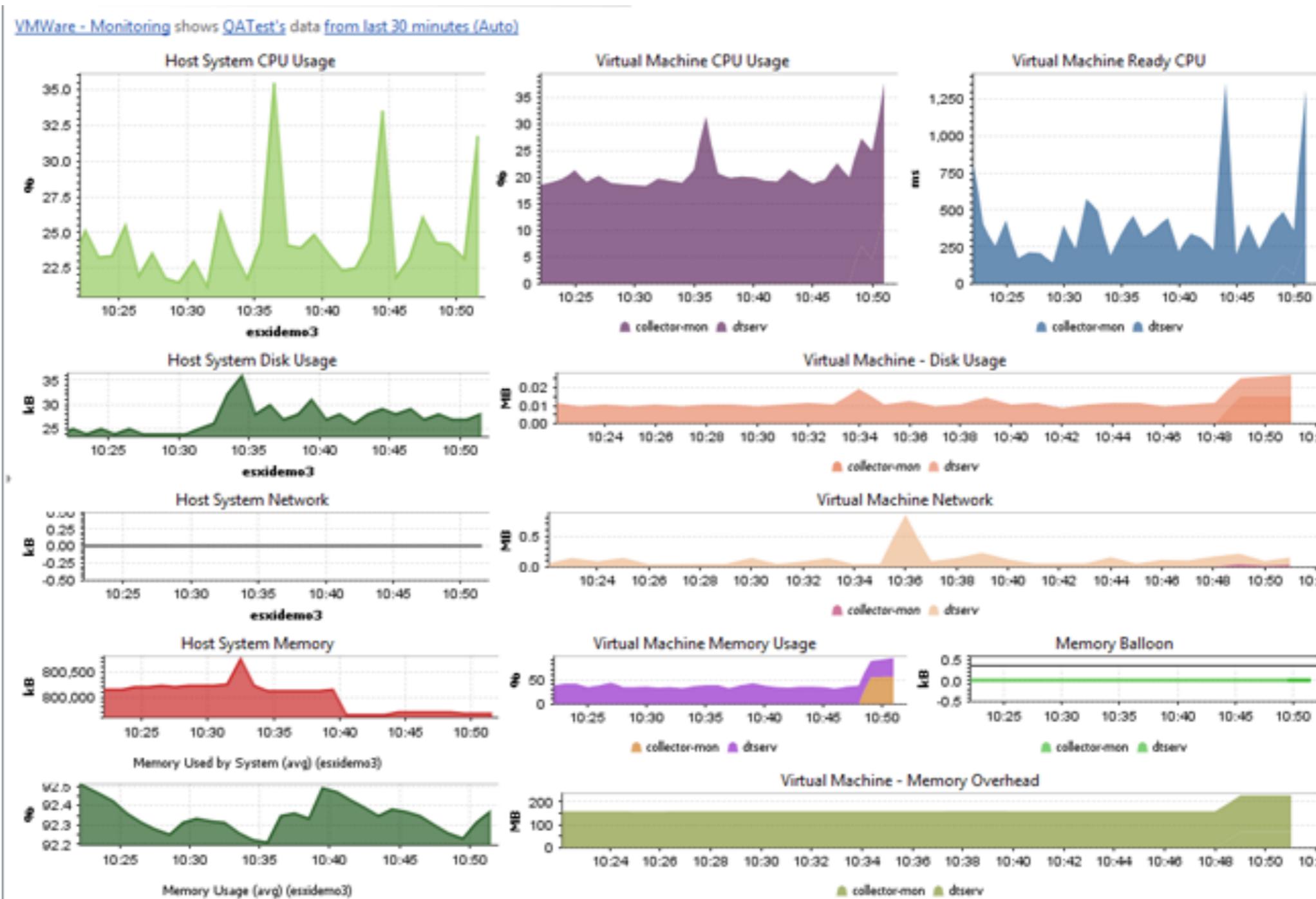
Examples:

photo gallery website: photo upload rate, photo processing times, ...

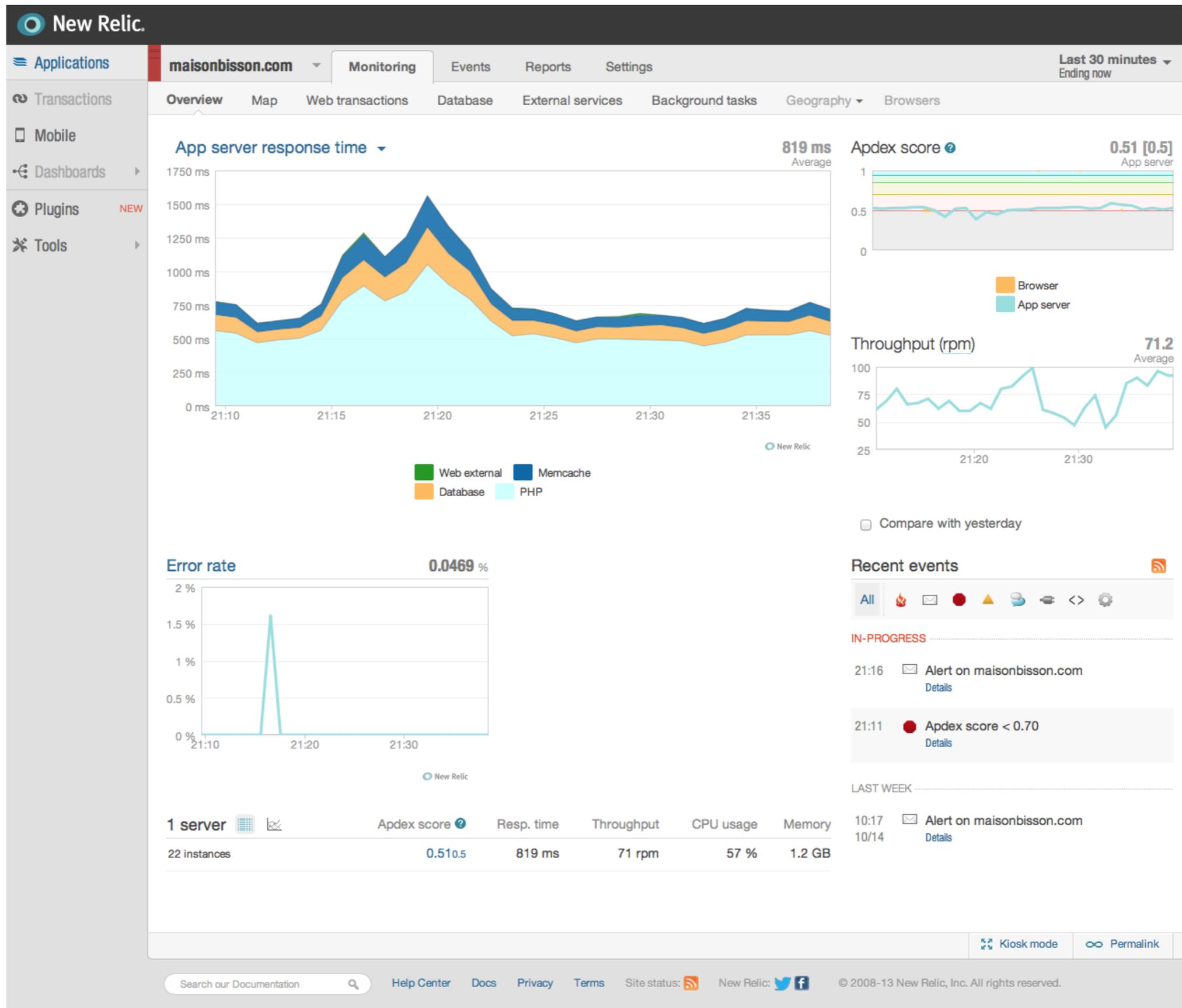
online shop: number of sold products within 24 hours, number of added user reviews, levels of user activity, ...

video platform: average of daily views, number of clicked ads, ...

Dashboards - Example VM Utilization



Dashboards - Example APM



Dashboards - Example Amazon EC2

EC2 Dashboard

Events

Tags

INSTANCES

- Instances
- Spot Requests
- Reserved Instances

IMAGES

- AMIs
- Bundle Tasks

ELASTIC BLOCK STORE

- Volumes
- Snapshots

NETWORK & SECURITY

- Security Groups
- Elastic IPs
- Placement Groups
- Load Balancers
- Key Pairs
- Network Interfaces

AUTO SCALING

- Launch Configurations
- Auto Scaling Groups

Launch Instance Connect Actions

Filter: All instances All instance types Search Instances

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP	Key Name	Monitoring
ec2-0042	i-00000000	m1.medium	eu-west-1a	running	2/2 checks ...	None	e-0-54-247-116-225-...	...	ec2-0042_key	disable

Instance: ec2-0042 (i-00000000) Elastic IP: e-0-54-247-116-225-...

Description Status Checks **Monitoring** Tags

CloudWatch alarms: No alarms configured [Create Alarm](#)

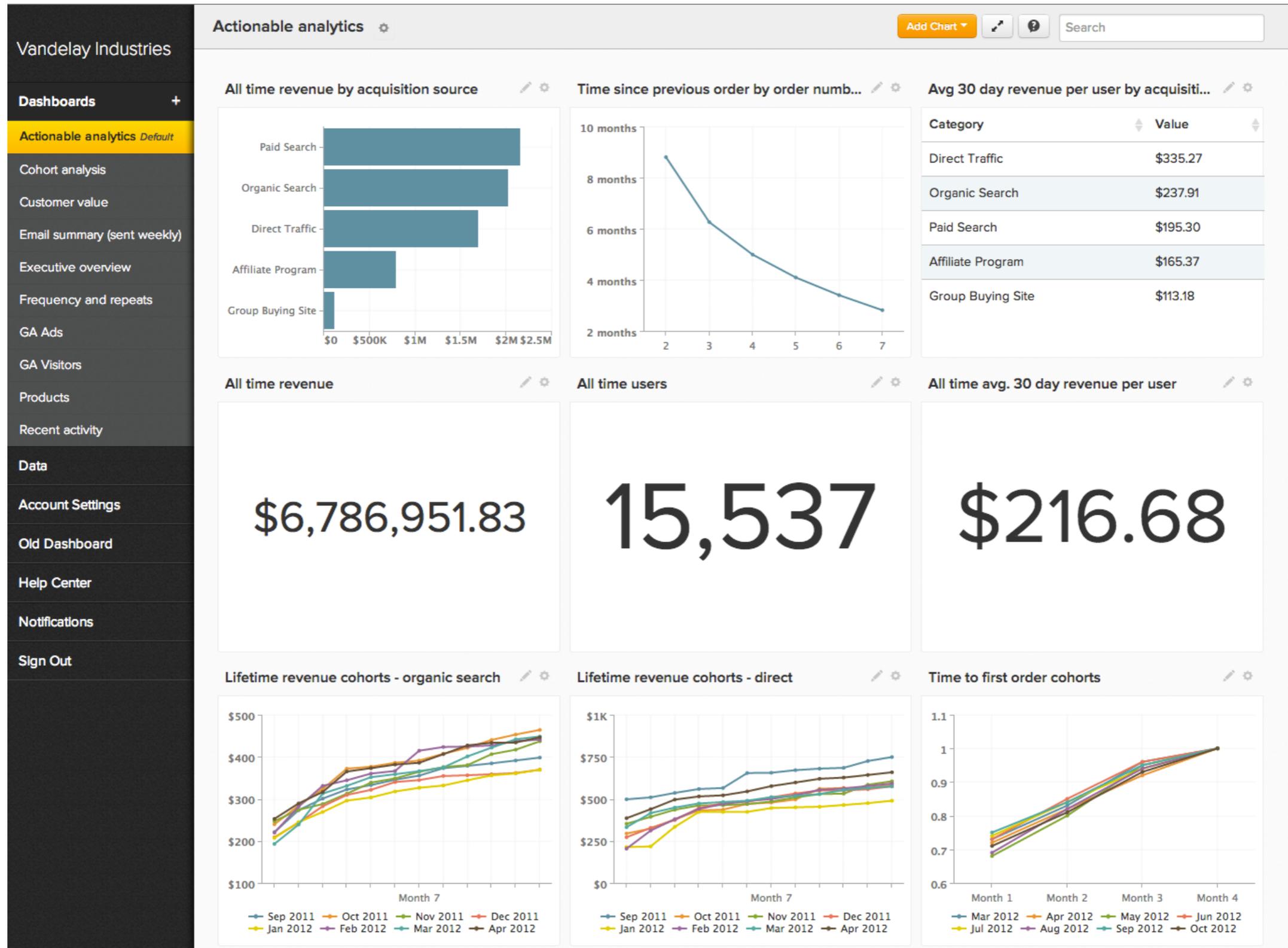
No alarms created. You can create an alarm using the Create Alarm button above.

CloudWatch metrics: Basic monitoring. [Enable Detailed Monitoring](#) Showing data for: Last 24 Hours

Below are your CloudWatch metrics for the selected resources (a maximum of 10). Click on a graph to see an expanded view. All times shown are in UTC. [View all CloudWatch metrics](#)

- CPU Utilization (Percent)**: Line graph showing CPU usage fluctuating between 25% and 100% over the last 24 hours.
- Disk Reads (Bytes)**: Line graph showing disk read activity, mostly near zero.
- Disk Read Operations (Operations)**: Line graph showing disk read operations, mostly near zero.
- Disk Writes (Bytes)**: Line graph showing disk write activity, mostly near zero.
- Disk Write Operations (Operations)**: Line graph showing disk write operations, mostly near zero.
- Network In (Bytes)**: Line graph showing network input, with a significant spike around 08:00 on 3/23.
- Network Out (Bytes)**: Line graph showing network output, with a significant spike around 08:00 on 3/23.
- Status Check Failed (Any) (Count)**: Line graph showing the number of failed status checks, which is zero.

Dashboards - Business Metrics



Live Experimentation - Implementation Techniques

Feature Toggles

- > dynamically enable and disable code segments (i.e., multiple “versions” in the code)
- > basically as simple as an if-statement
- > useful and cheap, might introduce complexity over time

Dynamic Routing

- > use a router to determine what service (and thus which version) should receive and process a request
- > multiple versions (thus instances) are running in parallel

Feature Toggles

```
function doSomething() {  
    if( featureIsEnabled("use-new-algorithm") ) {  
        ...  
    } else {  
        ...  
    }  
}
```

Usage scenarios:

- > prevent incomplete features from being executed in production, i.e. enabled only on internal testing environments
- > control rollout experimentation, e.g., which users see the newest version/feature (A/B testing, canary releases, gradual rollouts)

Feature Toggles

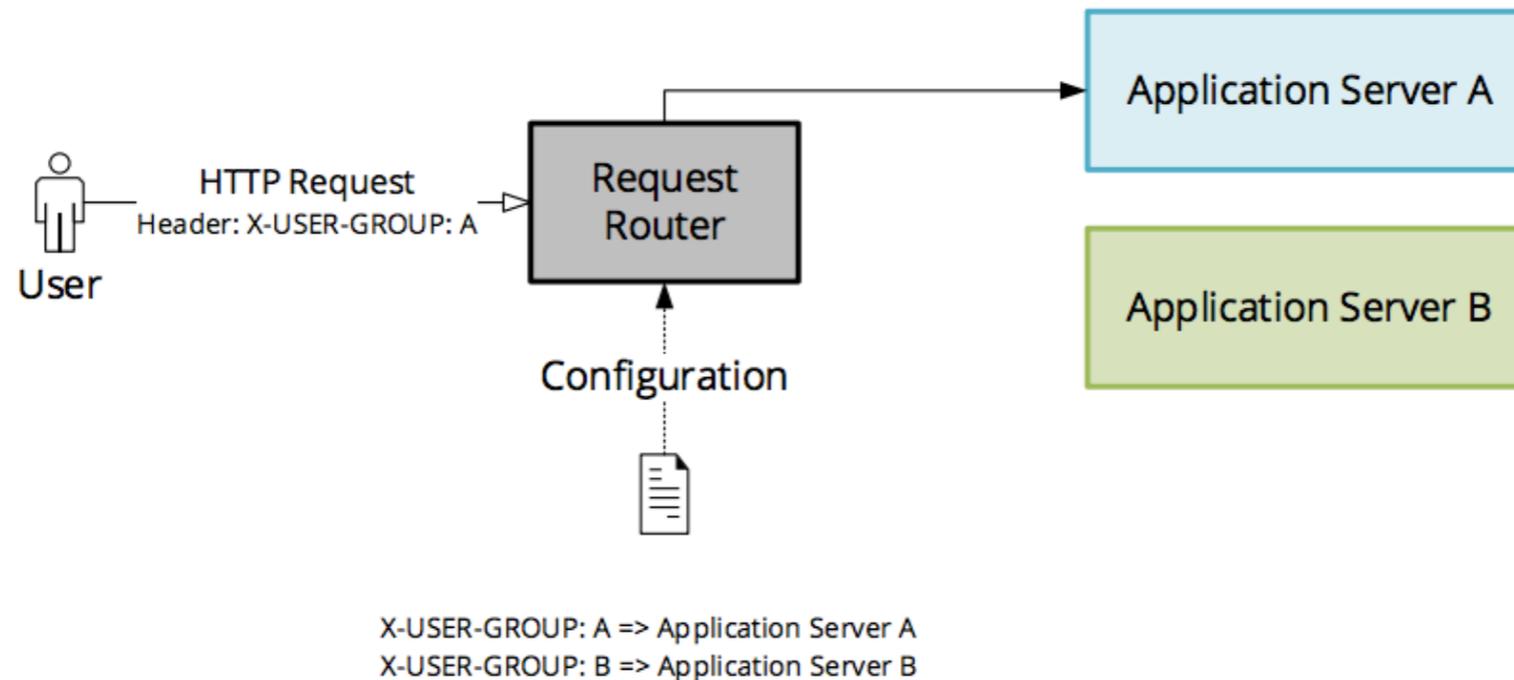
Feature toggles have to be **treated with care** as controlling and maintaining them could become problematic, especially the more feature toggles you have

toggles could interfere with each other (i.e., hard to test)

should be removed at some point from the code base, which might introduce new issues

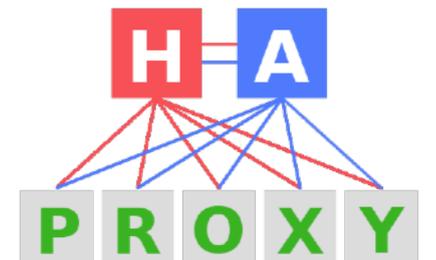


Dynamic Routing

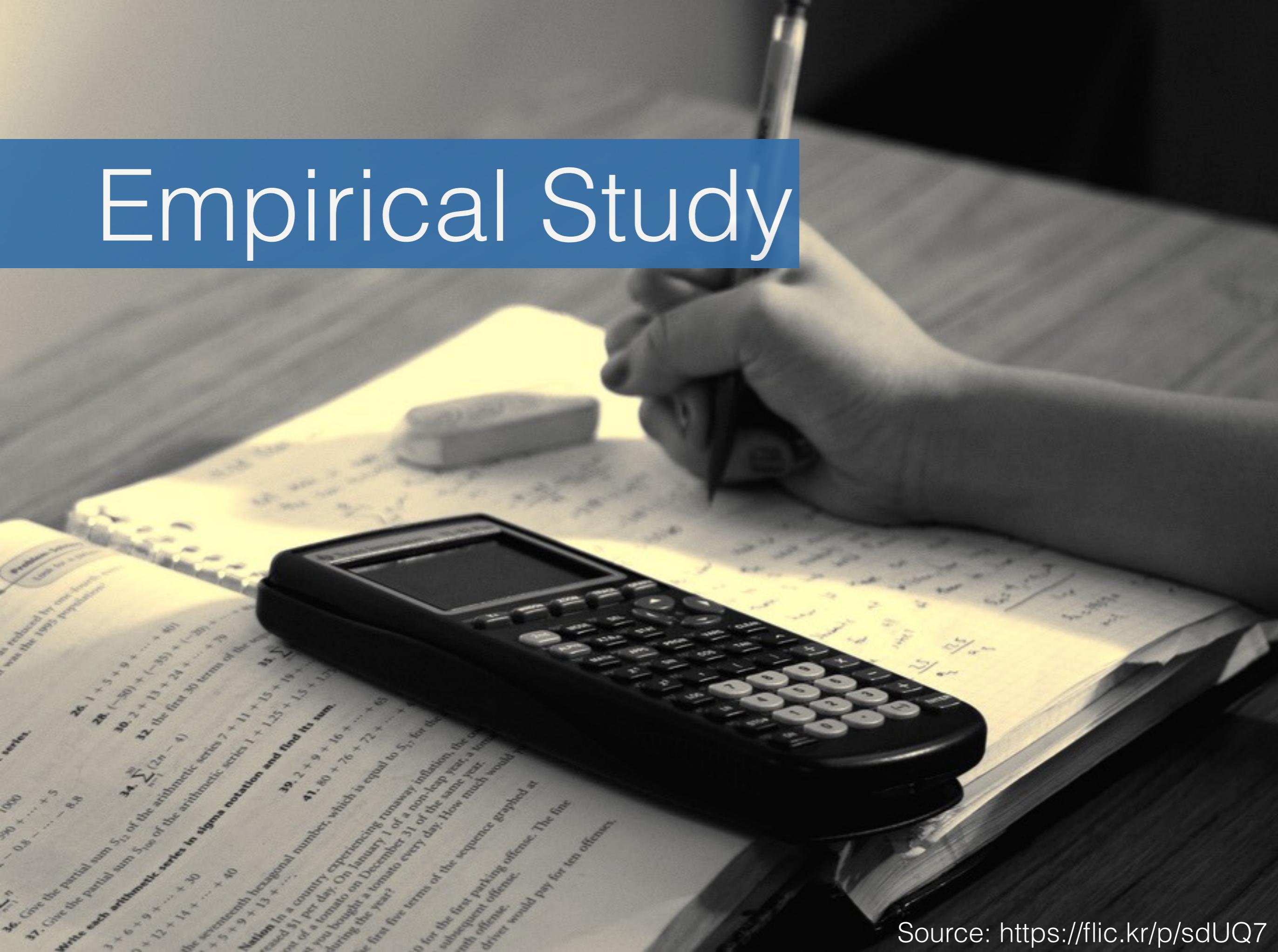


Removes feature toggle complexity from the code base, requires a properly managed router/proxy mechanism and multiple running instances

NGINX



Empirical Study

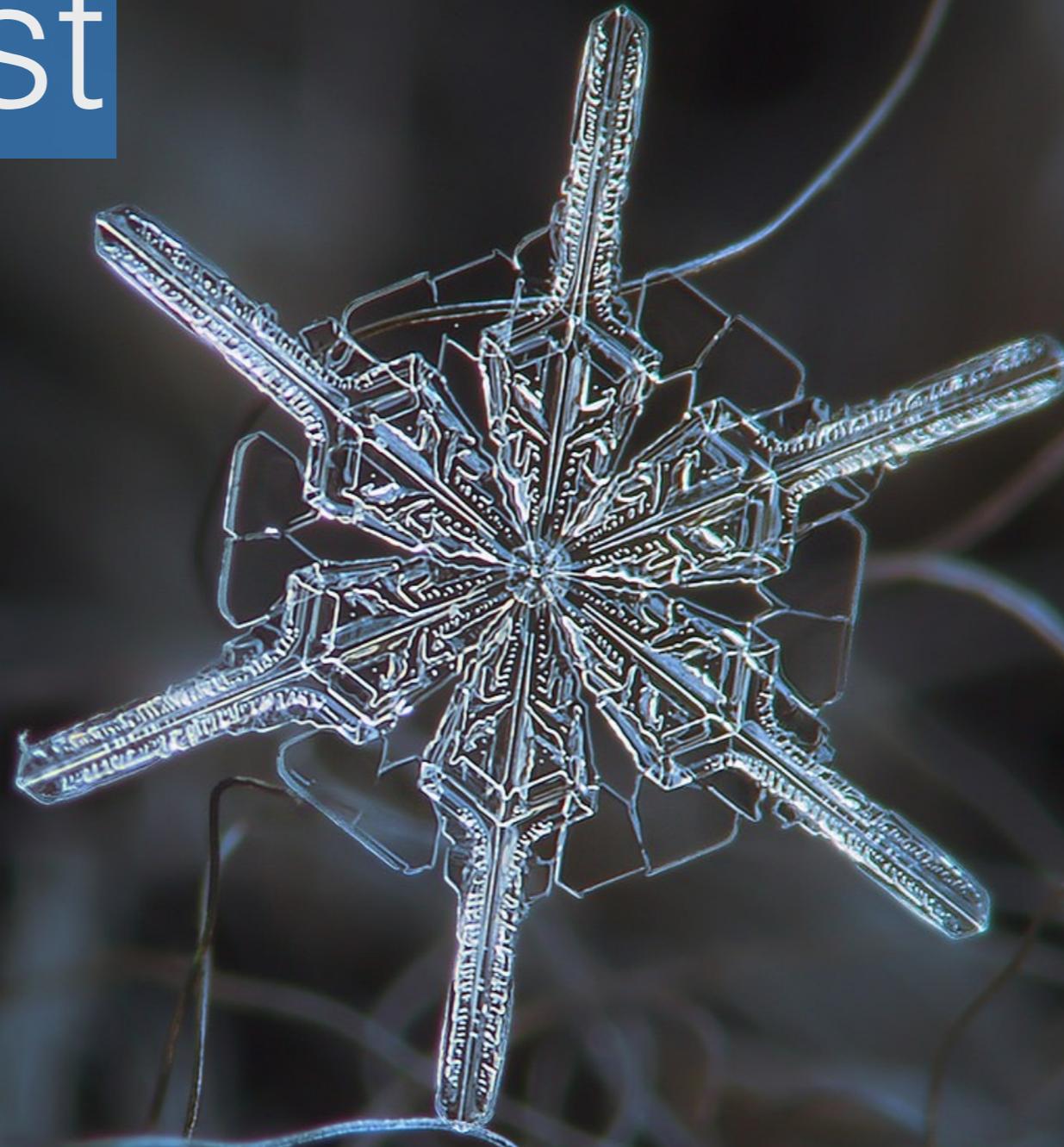


26. Give the partial sum S_{12} of the arithmetic series $7 + 11 + 15 + 19 + \dots$
27. Give the partial sum S_{100} of the arithmetic series $1 + 1.25 + 1.5 + 1.75 + \dots$
28. $1 + 3 + 5 + \dots + 40$
29. $1 + 3 + 5 + \dots + 40$
30. $2 + 13 + 24 + \dots + 79$
31. $1 + 3 + 5 + \dots + 40$
32. the first 30 terms of the series $1 + 3 + 5 + \dots$
33. $\sum_{n=1}^{10} (2n - 4)$
34. Write each arithmetic series in sigma notation and find its sum.
35. $3 + 6 + 9 + \dots + 30$
36. $2 + 12 + 14 + \dots + 40$
37. the seventeenth hexagonal number, which is equal to S_{17} for the series $1 + 3 + 6 + 10 + 15 + \dots$
38. Nation in a country experiencing runaway inflation, the cost of a tomato on January 1 of a non-leap year, a tomato cost you \$1 per day. On December 31 of the same year, a tomato cost you \$100 per day. How much more did you pay for a tomato during the year?
39. $2 + 9 + 16 + 25 + \dots + 100$
40. the first five terms of the sequence graphed at $(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)$
41. \$10 for the first parking offense. The fine for the second parking offense is \$15. The fine for the third parking offense is \$20. The fine for the fourth parking offense is \$25. The fine for the fifth parking offense is \$30. How much more would a driver pay for ten offenses?

State of Live Experimentation

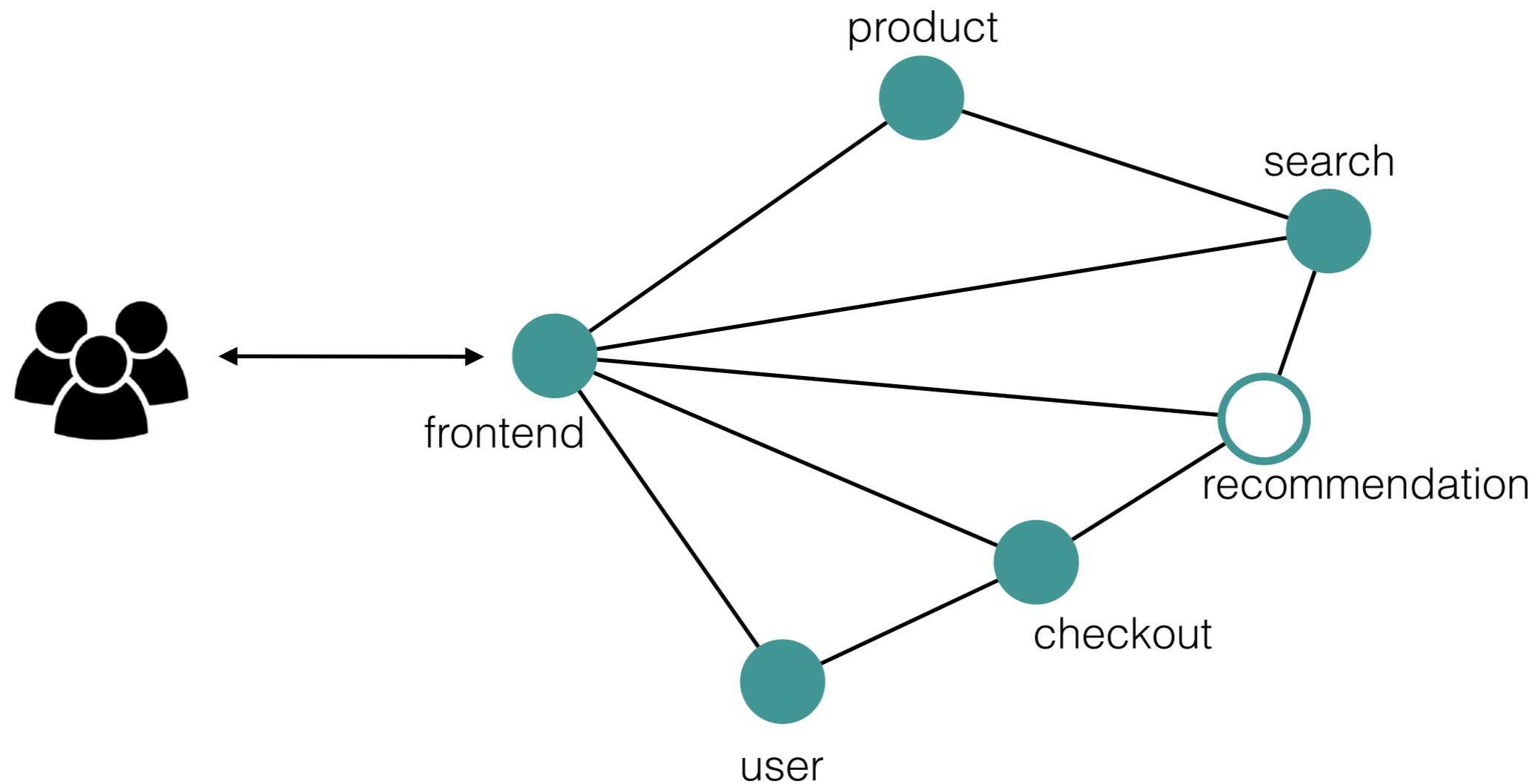
	Regression-Driven Experiments	Business-Driven Experiments
Main Goal	Mitigation of technical problems	Evaluation from a business perspective (e.g., monetary incentives)
Practices	Canary Releases, Dark Launches, Gradual Rollouts, Blue/Green Deployments	A/B Testing
Data Interpretation	Often intuitive, less process driven	Hypothesis- and data-driven
Duration	Minutes to multiple days	Order of weeks
User Selection	Small scoped, sometimes gradually increased	Two or more groups, constant size
Responsibility	Siloization	Multiple teams and services
Implementation	Feature toggles, traffic routing, distribution of binaries	
Obstacles	Architecture, limited number of users, missing business value or not worth investments, lack of expertise	

Bifrost



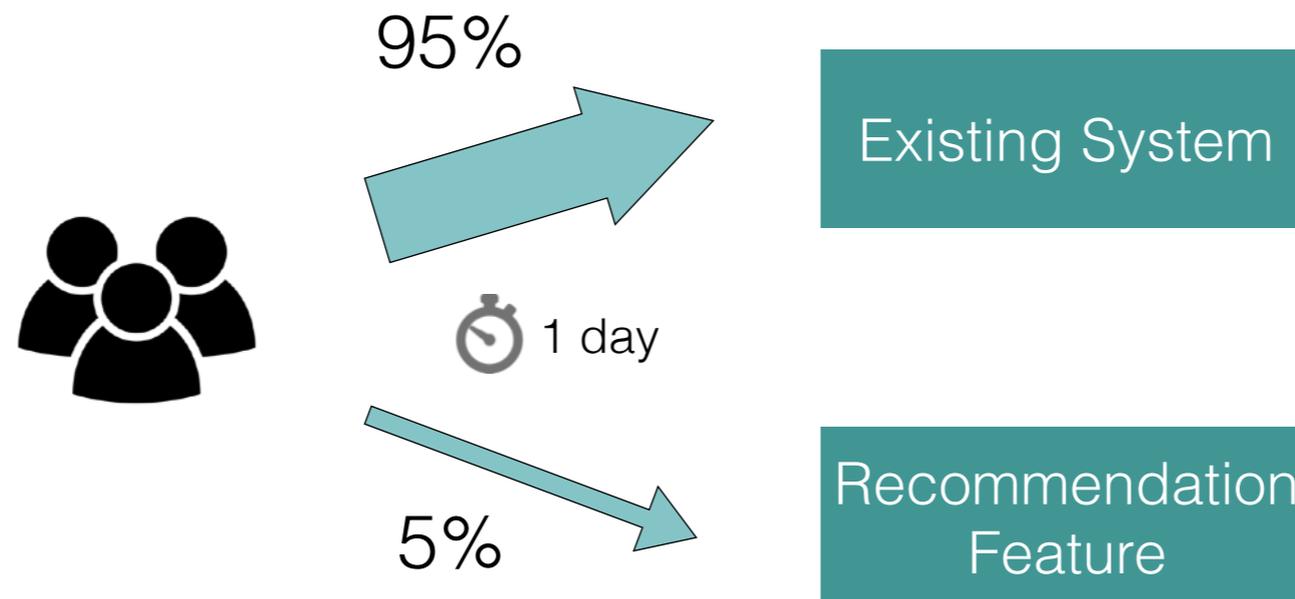
Multi-Phased Live Experimentation

Live Experimentation - Example



Live Experimentation - Example

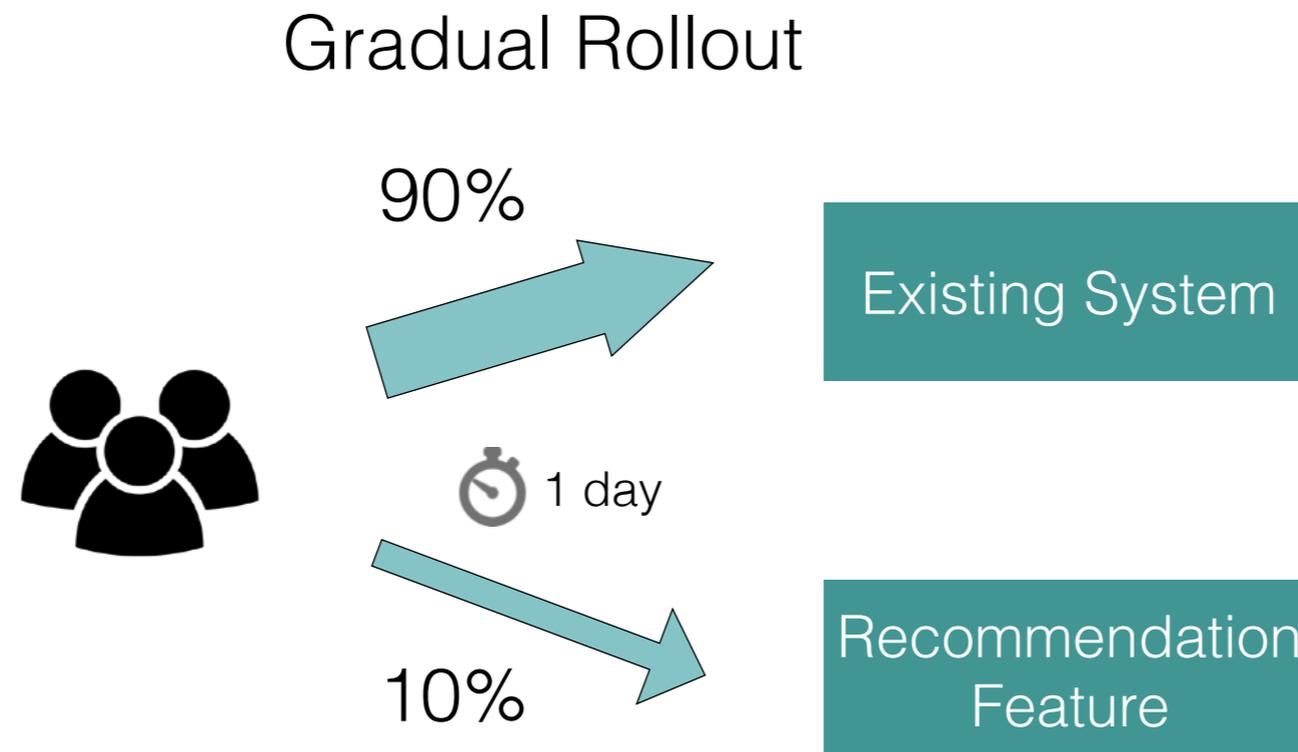
Gradual Rollout



Response Time < 50 ms

CPU Utilization < 80 %

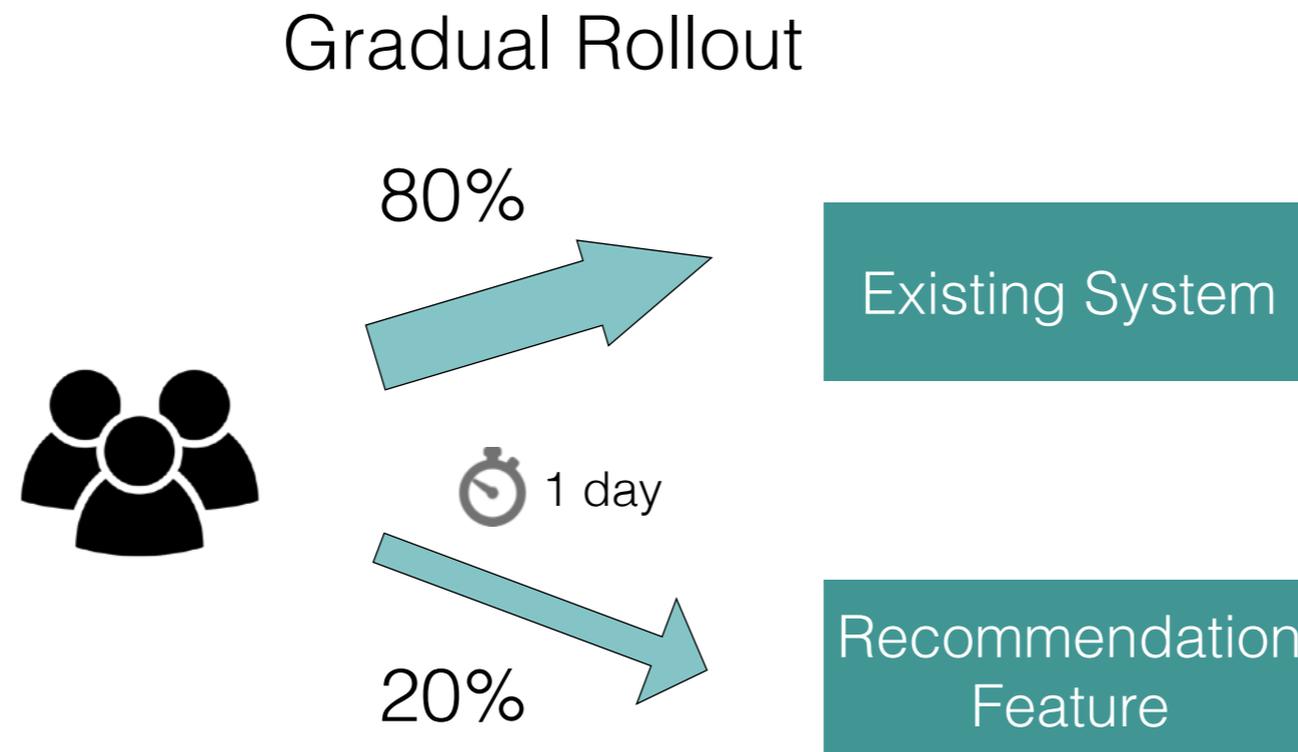
Live Experimentation - Example



Response Time < 50 ms

CPU Utilization < 80 %

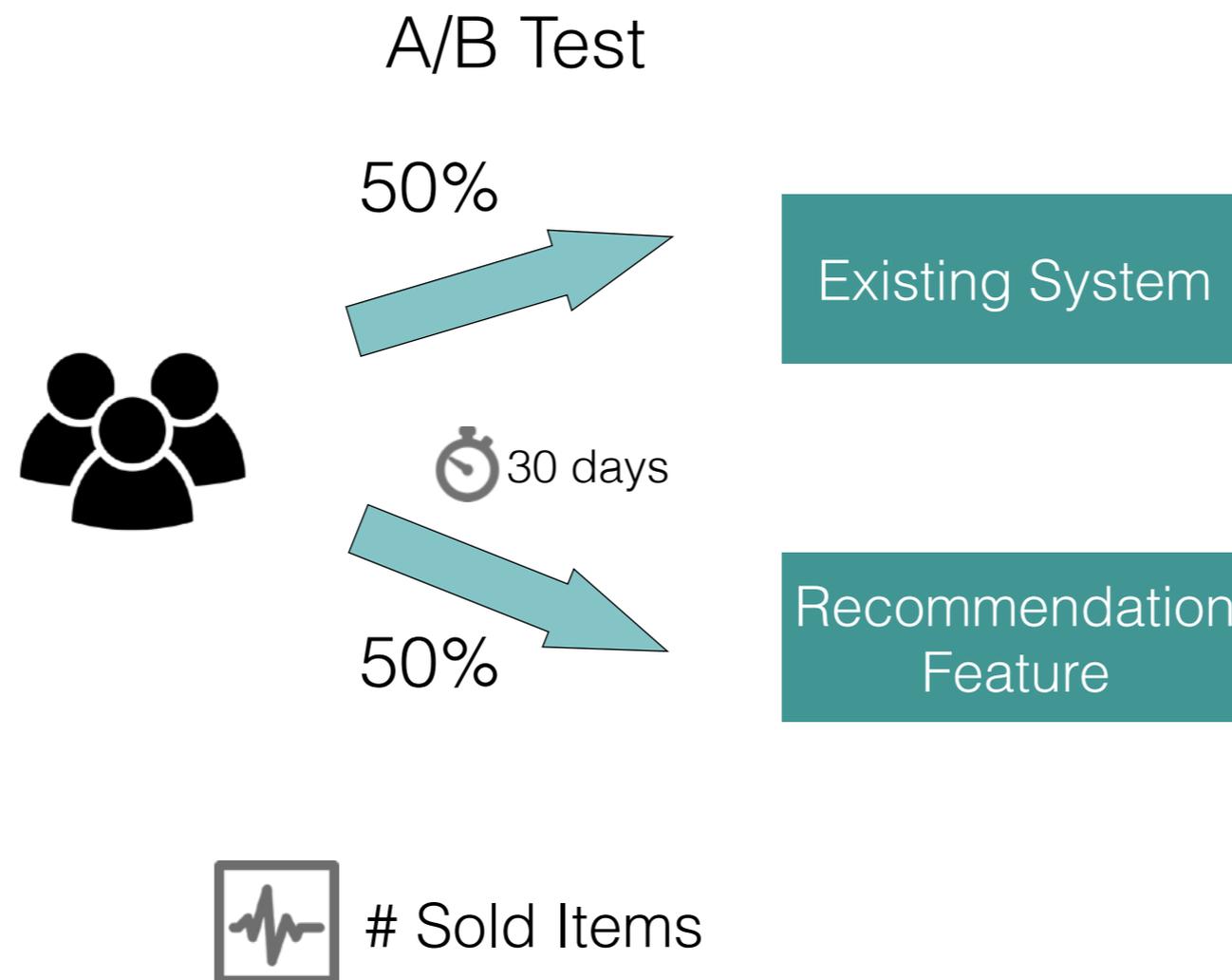
Live Experimentation - Example



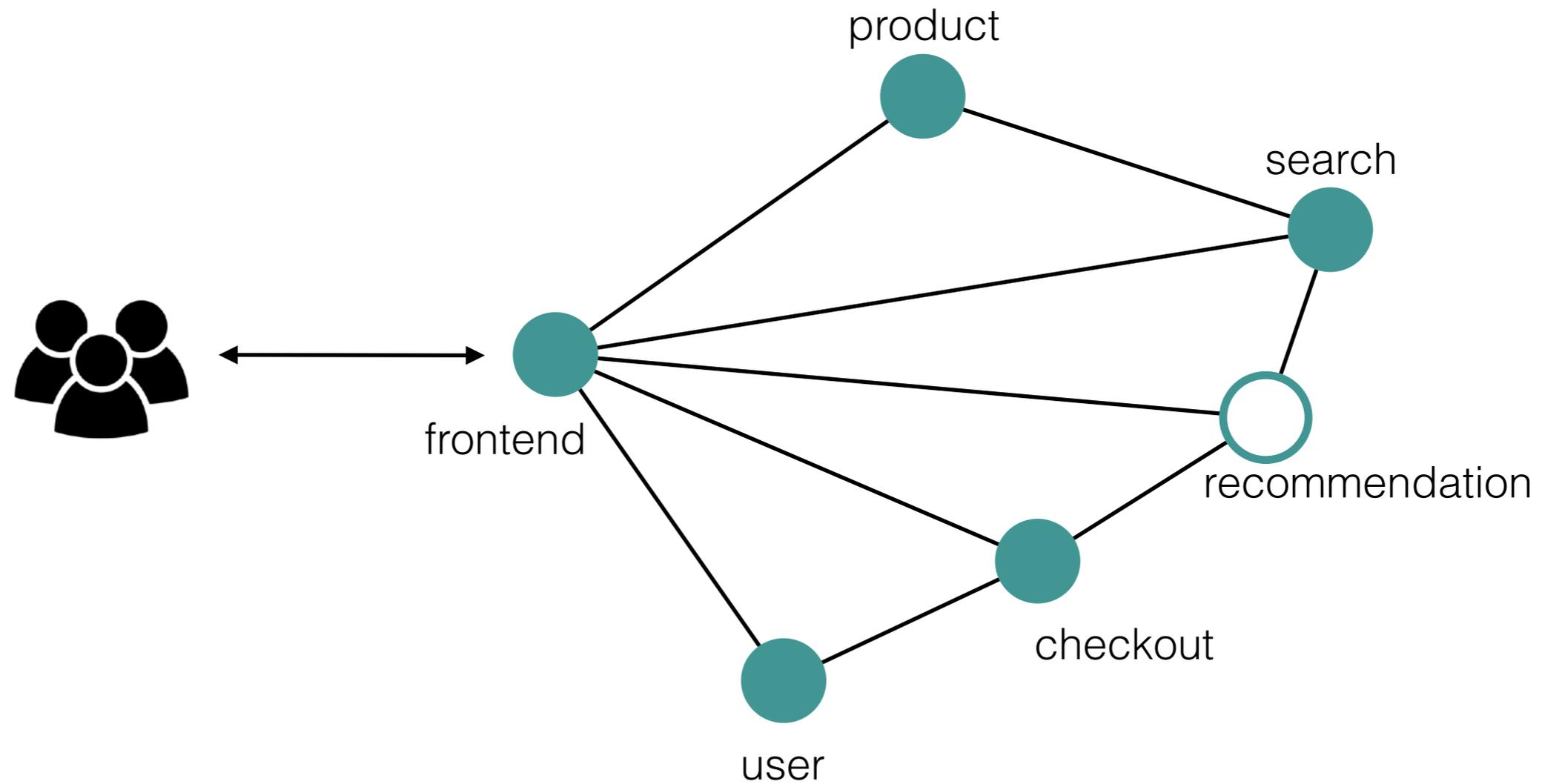
Response Time < 50 ms

CPU Utilization < 80 %

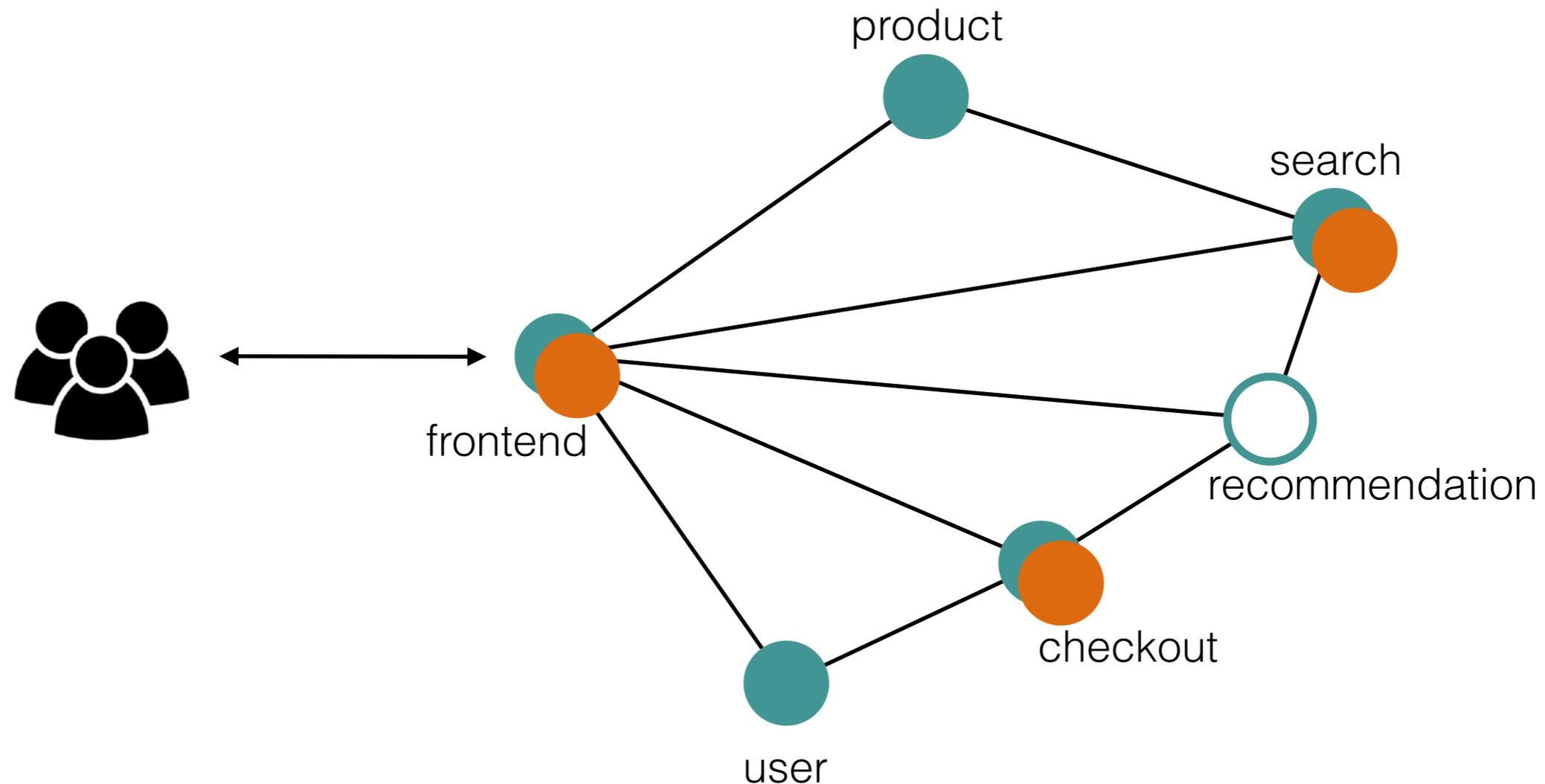
Live Experimentation - Example



Live Experimentation - Example



Live Experimentation - Example



⚙️ operate multiple versions at the same time

🧪 metrics to monitor, duration of experiment, user groups

Bifrost: Conducting Automated, Data-Driven, Multi-Phased Live Experiments



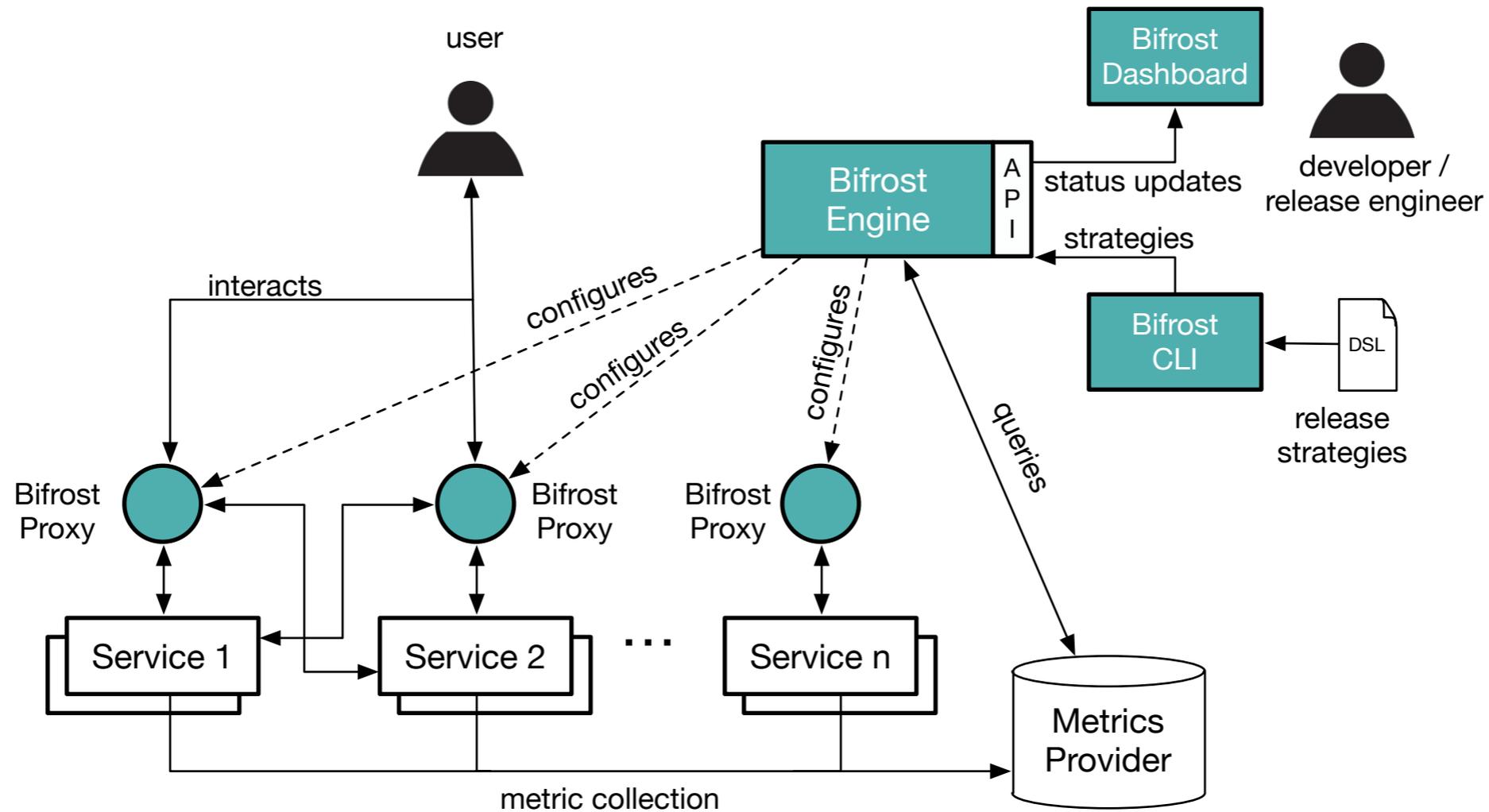
Bifrost:

Middleware for executing multi-phased live experiments
Designed for microservices-based applications



Existing tooling limited to specific types of live experimentation
e.g., Optimizely, CanaryAdvisor [Tarvo et al.'15], vamp.io

Bifrost: Conducting Automated, Data-Driven, Multi-Phased Live Experiments



Bifrost Architecture

Bifrost: Conducting Automated, Data-Driven, Multi-Phased Live Experiments

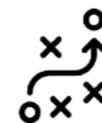
Explicit live experimentation using a DSL

```
12 12 strategies:
13 13   - name: Canary_Launch
14 14     actions:
15 15       - AND:
16 16         actions:
17 17           - route:
18 18             from: search
19 19             to: search_rec
20 20             intervalTime: 1h
21 21             filters:
22 22               - traffic:
23 23                 - percentage: 5
24 24                 + header:
25 25                   field: X-User-Group
26 26                   value: Premium
27 27             - metric:
28 28               providers:
29 29                 query: request_errors{instance="search_rec:80"}
30 30                 intervalTime: 5s
31 31                 intervalLimit: 720
32 32                 - threshold: 600
33 33                 - validator: "<5"
34 34                 + threshold: 650
35 35                 + validator: "<4"
36 36             onTrue: Load_Test
37 37             onFalse: rollback
```

Experimentation as Code



Fosters version-controlling, sharing, and reusing of strategies



Traceability, which experiments are executed, what is monitored, which users are part of what experiments

PERIODIC TABLE OF DEVOPS TOOLS (V1) XebiaLabs Deliver Faster

Os Open Source

Fr Free

Fm Freemium

Pd Paid

En Enterprise

Database	SCM	Build
CI	Repo Mgmt	Testing
Deployment	Config / Provisioning	Containerization
Cloud / IaaS / PaaS	Release Mgmt	Collaboration
BI / Monitoring	Logging	Security

2 Fm
Aws
Amazon Web Services

1 En O 12c																2 Fm Aws Amazon Web Services	
3 Os My MySQL	4 Os Gt Git															10 Pd Az Azure	
11 En Mq MSSQL	12 Os Sv Subversion															18 Fm Hk Heroku	
19 Os Pq PostgreSQL	20 Fm Gh Github	21 Os Mv Maven	22 Os Gr Gradle	23 En Mr Meister	24 Os Jn Jenkins	25 Pd Ba Bamboo	26 Os Tr Travis CI	27 Fr Ar Archive	28 Os Fn FitNesse	29 Fr Se Selenium	30 Os Gn Gatling	31 Pd Gd Deployment Manager	32 Os Sf SmartFrog	33 Fr Cb Cobbler	34 Os Bc Bcfg2	35 Os Kb Kubernetes	36 En Rs Rackspace
37 Os Mg MongoDB	38 Fm Bb Bitbucket	39 Os Br Buildr	40 Os At ANT	41 Fm Bm BuildMaster	42 Fm Cs Codeship	43 Fm Sn Snap CI	44 Fm Cr CircleCI	45 Os Nx Nexus	46 Fr Cu Cucumber	47 Os Cj Cucumberjs	48 Fr Qu Qunit	49 Fr Cp Capistrano	50 Fr Ju JuJu	51 Os Rd Rundeck	52 Os Cf CFEngine	53 Fr Pk Packer	54 Fm Bx Bluemix
55 En Db DB2	56 Os Hg Mercurial	57 Fm Qb QuickBuild	58 En Ub UrbanCode Build	59 Pd Ta Visual Build	60 Fm Tc TeamCity	61 Fm Sh Shippable	62 Os Cc CruiseControl	63 Os Ay Artifactory	64 Fr Jt JUnit	65 Fr Jm JMeter	66 Fr Tn TestNG	67 En Ry RapidDeploy	68 Fm Cy CodeDeploy	69 En Oc Octopus Deploy	70 Os No CA Nfolio	71 En Eb ElasticBox	72 En Ad Apprenda
73 Fr Cs Cassandra	74 En Hx Helix	75 Os Msb MSBuild	76 Os Rk Rake	77 Os Lb LuntBuild	78 Os Co Continuum	79 Fm Ca Continua CI	80 Os Gu Gump	81 Os Ng NuGet	82 Os Ap Appium	83 En Xltv XL TestView	84 En Tc TestComplete	85 Os Go Go	86 En Ef ElectricFlow	87 En Xld XL Deploy	88 En Ud UrbanCode Deploy	89 Os Mo Mesos	90 En Os OpenShift
91 En Xlr XL Release	92 En Ur UrbanCode Release	93 En Ls CA Service Virtualization	94 En Bm BMC Release Process	95 En Hp HP Codar	96 Pd Ex Excel	97 En Pl Plutora Release	98 En Sr Serena Release	99 Fm Tr Trello	100 Pd Jr Jira	101 Fm Rf HipChat	102 Fm Sl Slack	103 Fm Fd Flowdock	104 Pd Pv Pivotal Tracker	105 En Sn ServiceNow			
106 Os Ki Kibana	107 Fm Nr New Relic	108 Os Ni Nagios	109 Os Gg Ganglia	110 Os Ct Cacti	111 Os Gr Graphite	112 Fm Le Logentries	113 En Sp Splunk	114 Fm Sl Sumo Logic	115 Os Ls Logstash	116 Fm Lg Loggly	117 Os Gr Graylog	118 Os Sn Snort	119 Os Tr Tripwire	120 En Cy CyberArk			

Share

Embed

Become Excellent!

[Subscribe here!](#)

