CS 4530: Fundamentals of Software Engineering
Lesson 5.4
Continuous Delivery

Rob Simmons

Khoury College of Computer Sciences

Continuous Delivery

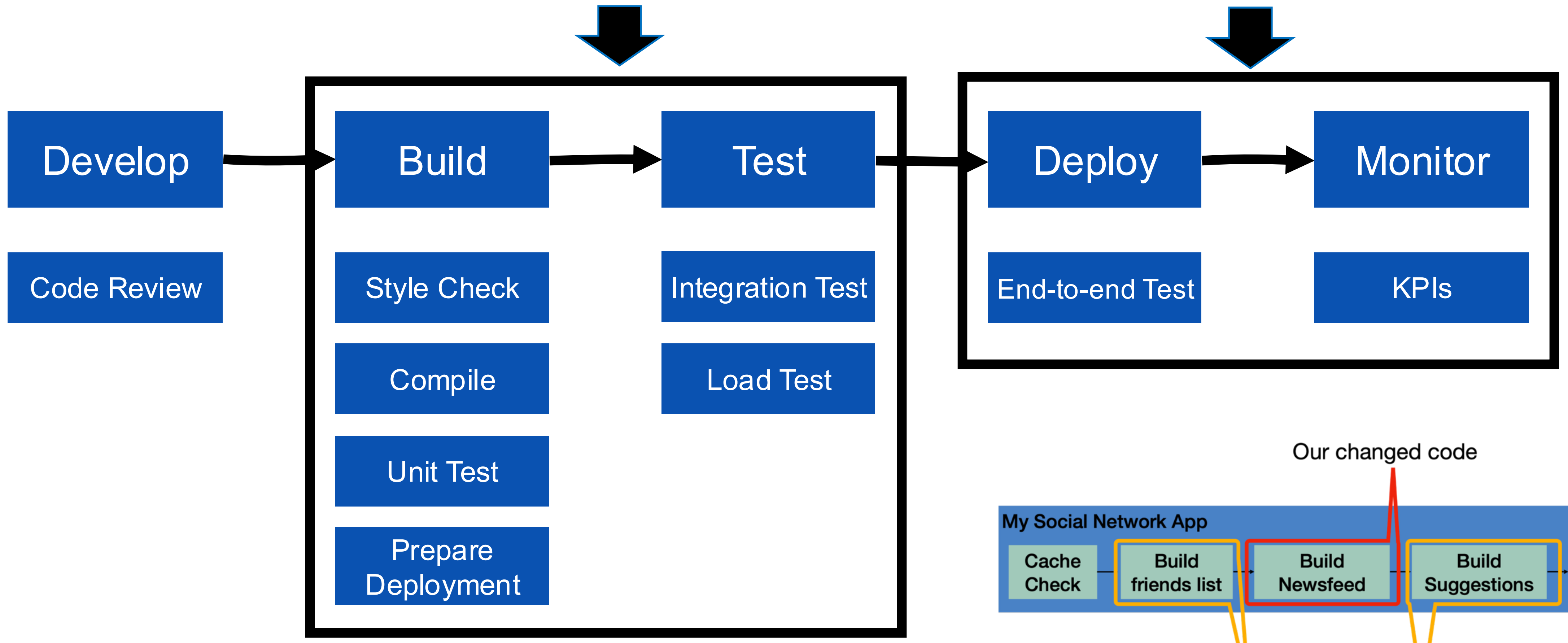
- “Faster is safer”: Key values of continuous delivery
  - Release frequently, in small batches
  - Maintain key performance indicators to evaluate the impact of updates
  - Phase roll-outs
  - Evaluate business impact of new features

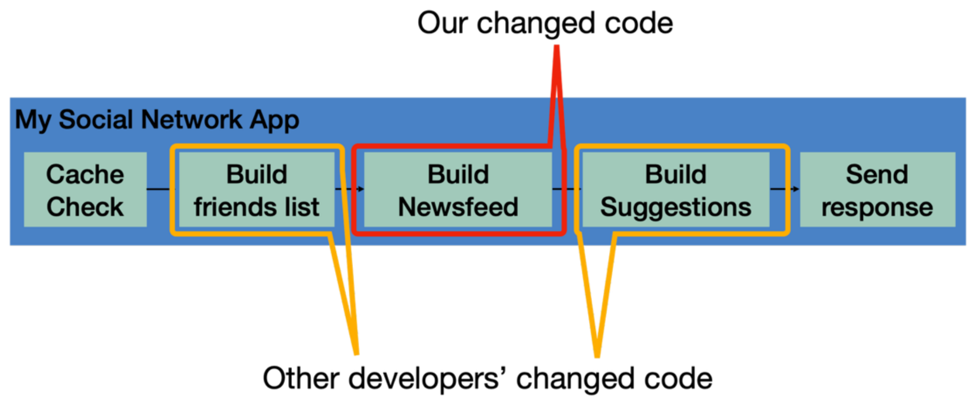
Continuous Delivery is about deciding which new features to deliver, and when

- You have a large system with many engineers working on new features (and bug fixes ☺)
- When a new feature or fix is ready, how do you roll it out to your users?

A continuous-delivery process is also a software pipeline

Automate this centrally, provide a central record of results





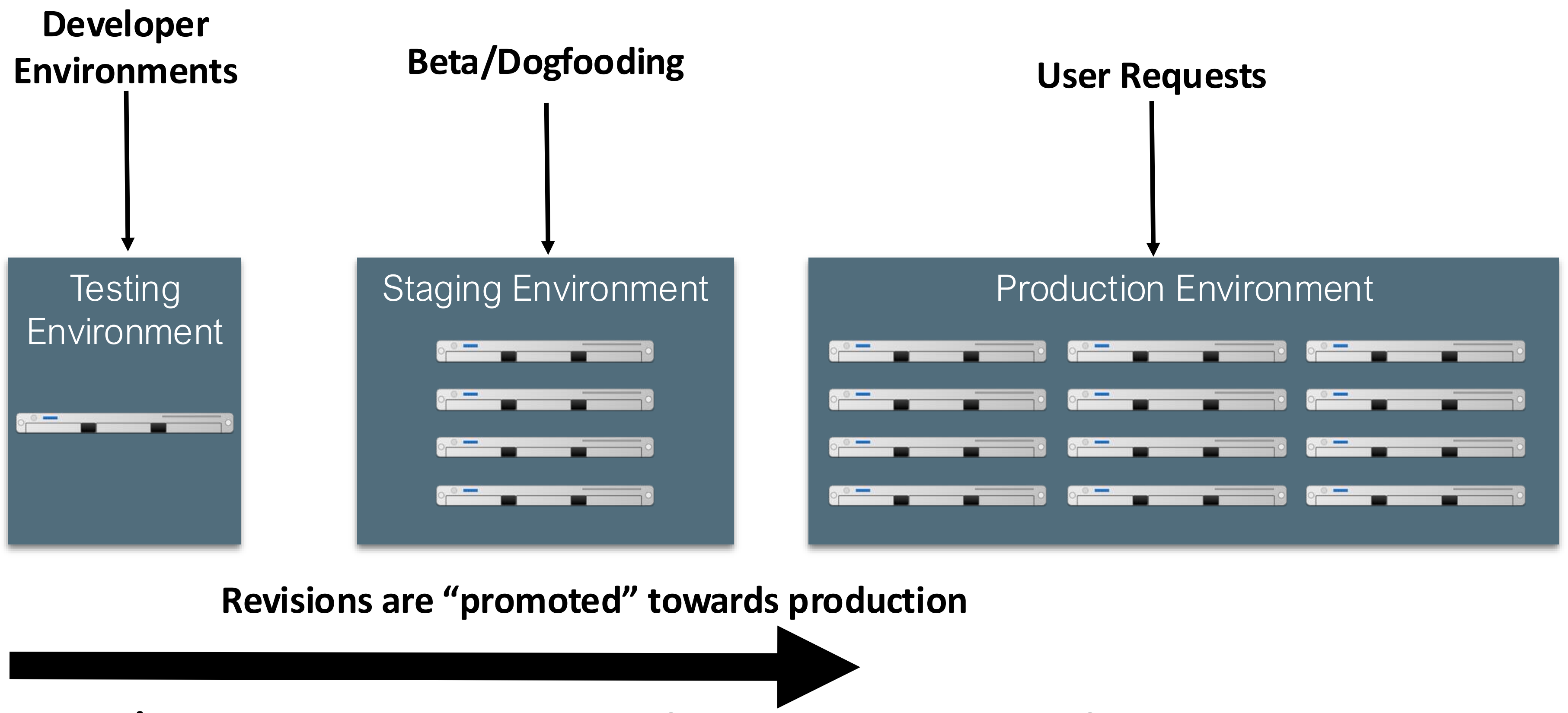
Continuous Delivery does not mean Immediate Delivery

- 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

Ways to mitigate deployment risks

- Use a realistic staging environment
- Use post-deployment monitoring
- Use split deployments
- Use tools to automate deployment tasks

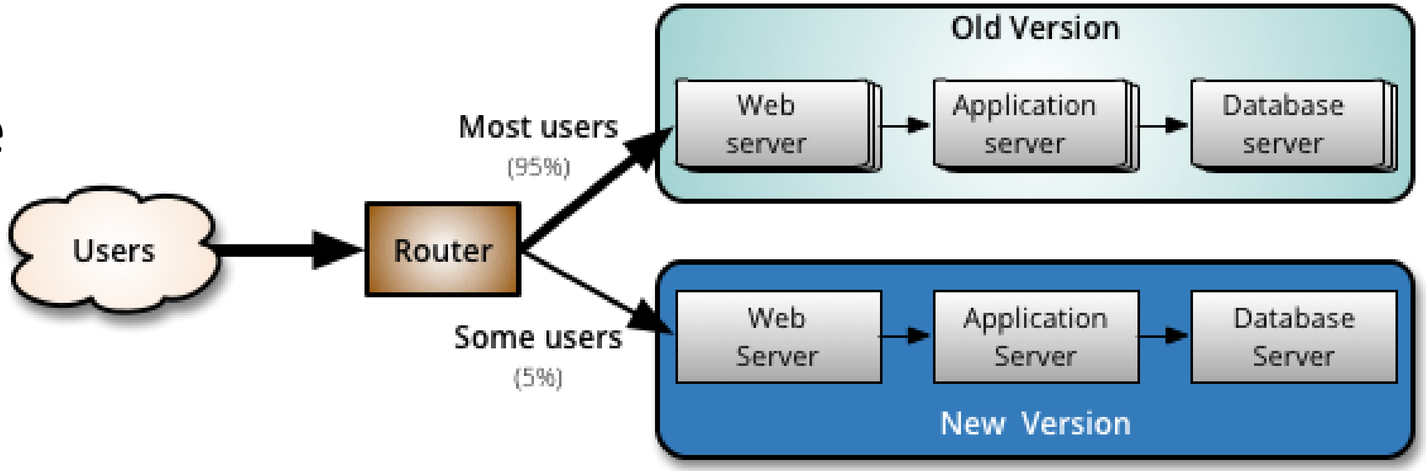
Build a staging environment to qualify features for delivery



Q/A takes place in each stage (including production!)

Split Deployments Mitigate Risk

- Lower risk if a problem occurs in staging than in production
- Or deploy to a small set of users before deploying more widely
- Names:
  - “Eat your own dogfood”
  - Beta/Alpha testers
  - A/B testing
  - "canaries"

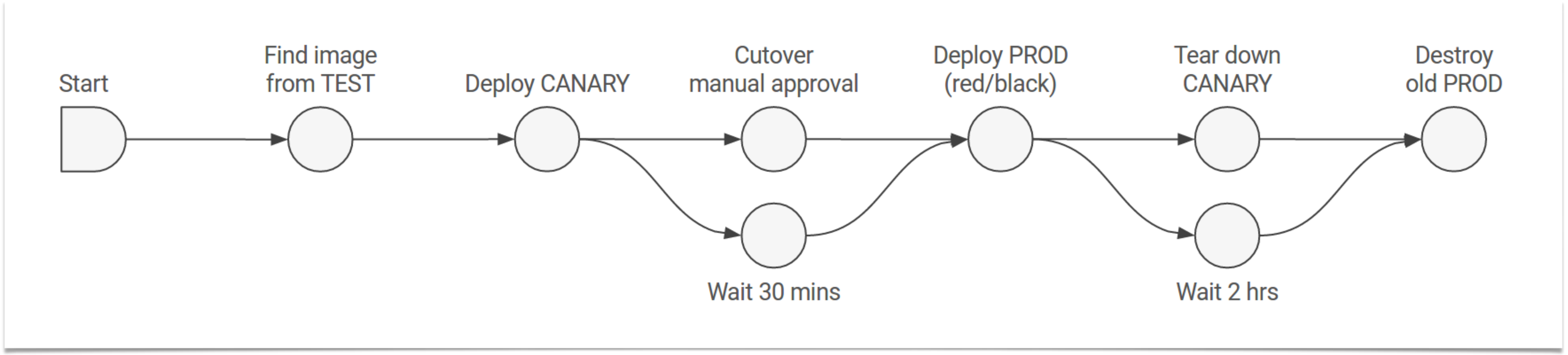


Post-delivery monitoring mitigates risk

- 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

Continuous Delivery Tools

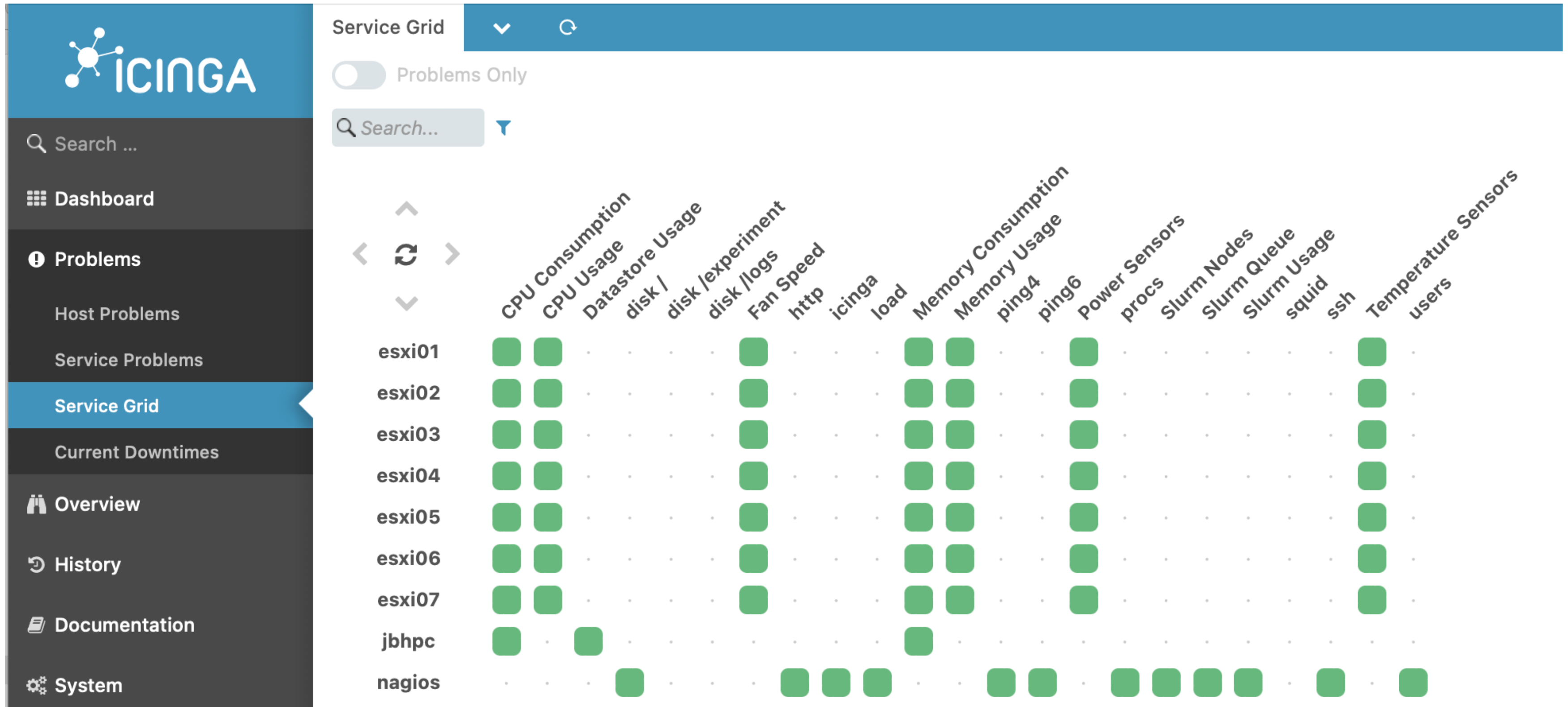
- Simplest tools deploy from a branch to a service (e.g. Render.com, Heroku)
- More complex tools:
  - Auto-deploys from version control to a staging environment + promotes through release pipeline
  - Monitors key performance indicators to automatically take corrective actions
  - Example: "Spinnaker"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



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

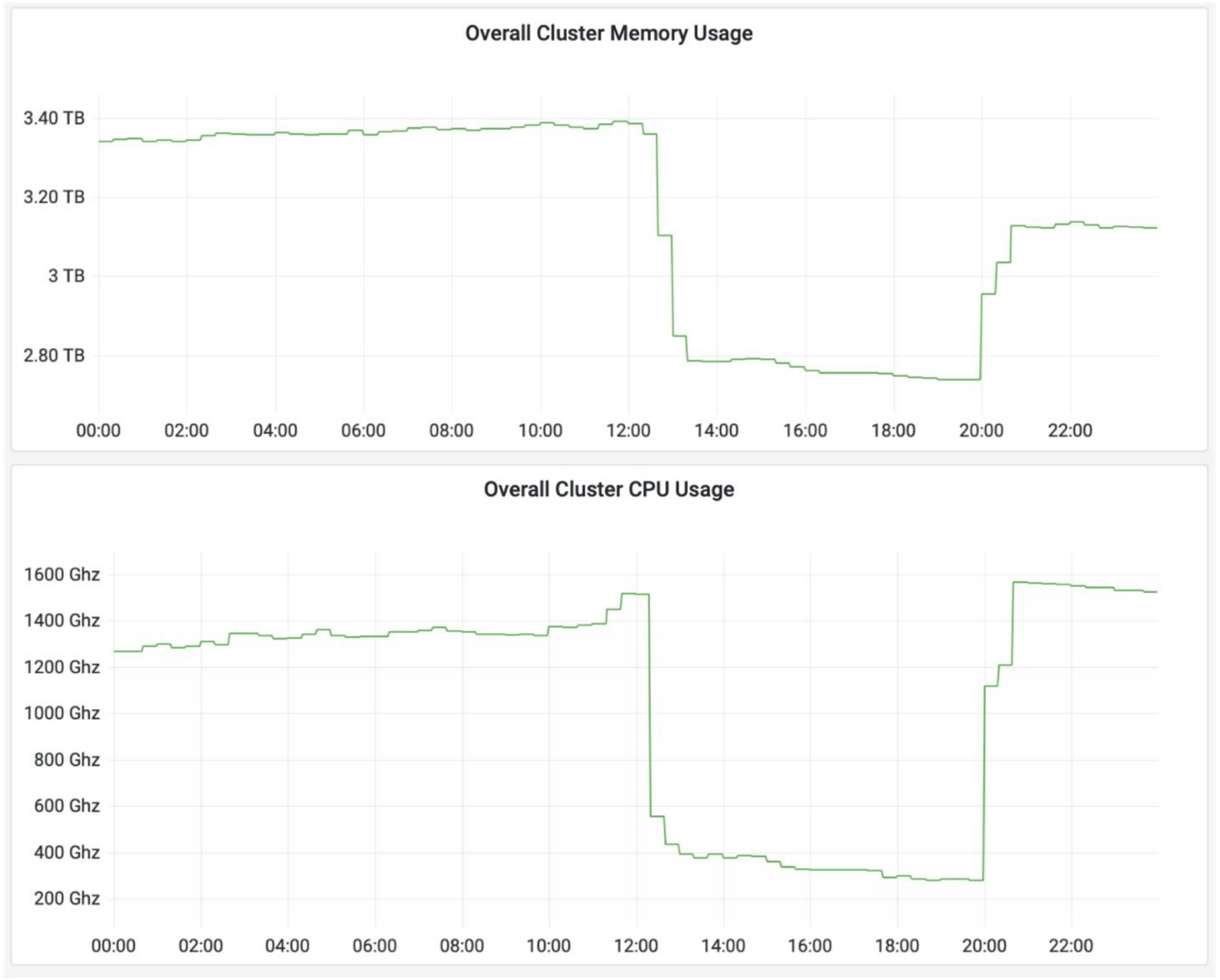
Tools for Monitoring Deployments

- 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

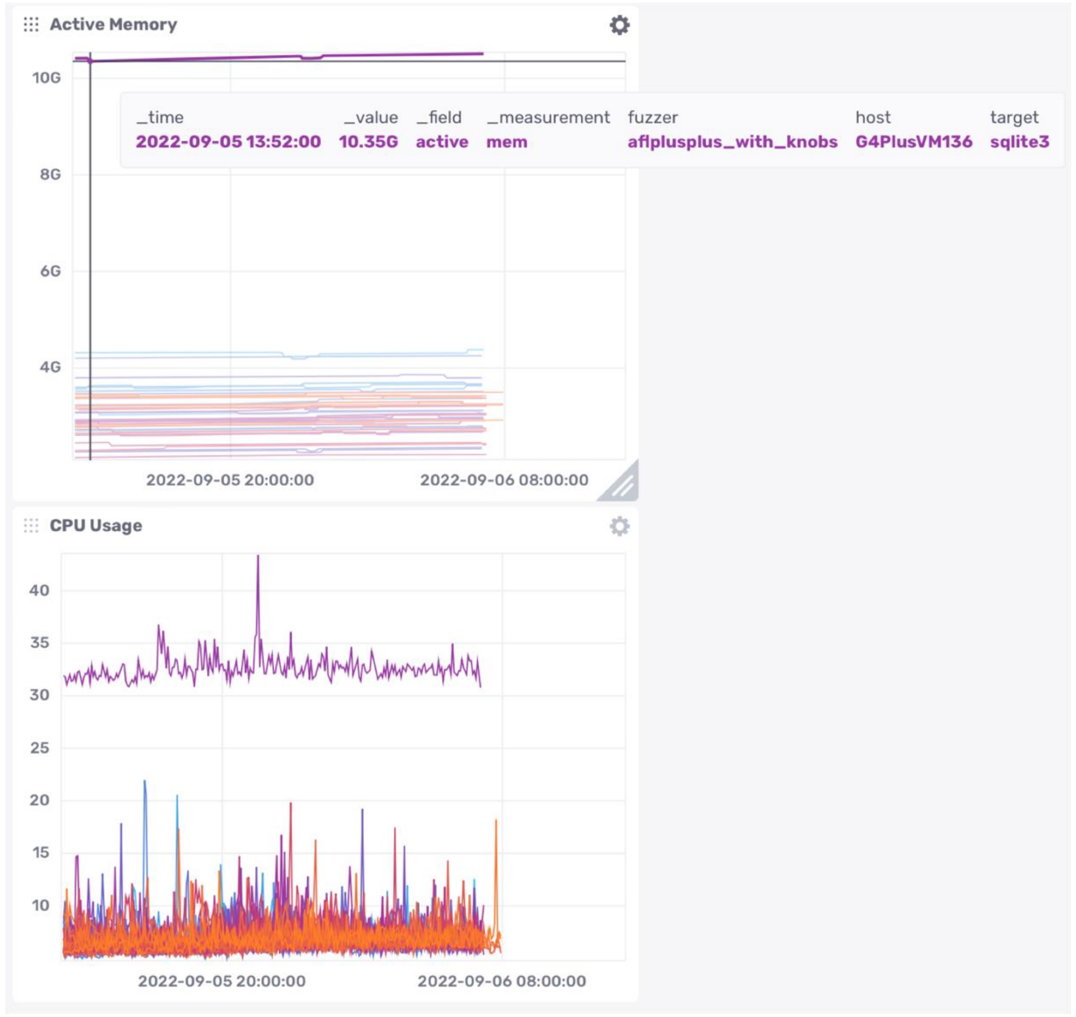


	CPU Consumption	CPU Usage	Datastore Usage	disk /	disk /experiment	disk /logs	Fan Speed	http	icinga	load	Memory Consumption	Memory Usage	ping4	ping6	Power Sensors	procs	Slurm Nodes	Slurm Queue	Slurm Usage	squid	ssh	Temperature Sensors	users
esxi01	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi02	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi03	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi04	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi05	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi06	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
esxi07	🟩	🟩					🟩				🟩	🟩			🟩							🟩	
jbhpc	🟩		🟩								🟩												
nagios				🟩				🟩	🟩	🟩			🟩	🟩		🟩	🟩	🟩	🟩		🟩		🟩

Monitoring can help identify operational issues



Grafana (AGPL, c 2014)



_time	_value	_field	_measurement	fuzzer	host	target
2022-09-05 13:52:00	10.35G	active	mem	aflplusplus_with_knobs	G4PlusVM136	sqlite3

InfluxDB (MIT license, c 2013)

How should we allocate our testing resources?

- How much unit testing should be required?
- When should we do code reviews?
- How often should we do integration tests?
- Different organizations may make different choices

Two extremes(?)
Continuous Delivery vs. TDD

- Test driven development
    - Write and maintain tests per-feature (manual! hard!)
    - 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

CI at scale: Google Test Automation Platform (TAP (2020))

- Massive continuous build of entire Google codebase
  - in a dedicated data center
  - 50,000 unique changes per-day, 4 billion test cases per-day
- Engineers submit unit tests along with their changes
  - Block merge if they fail
- If they pass, change is put in the codebase.
  - visible to entire company!
  - average wait time to this point: 11 minutes
- Then (asynchronously) run all affected integration tests
  - If any fail, change is sent back to a human on the submitter's team (the "build cop") who must act immediately to roll-back or fix.

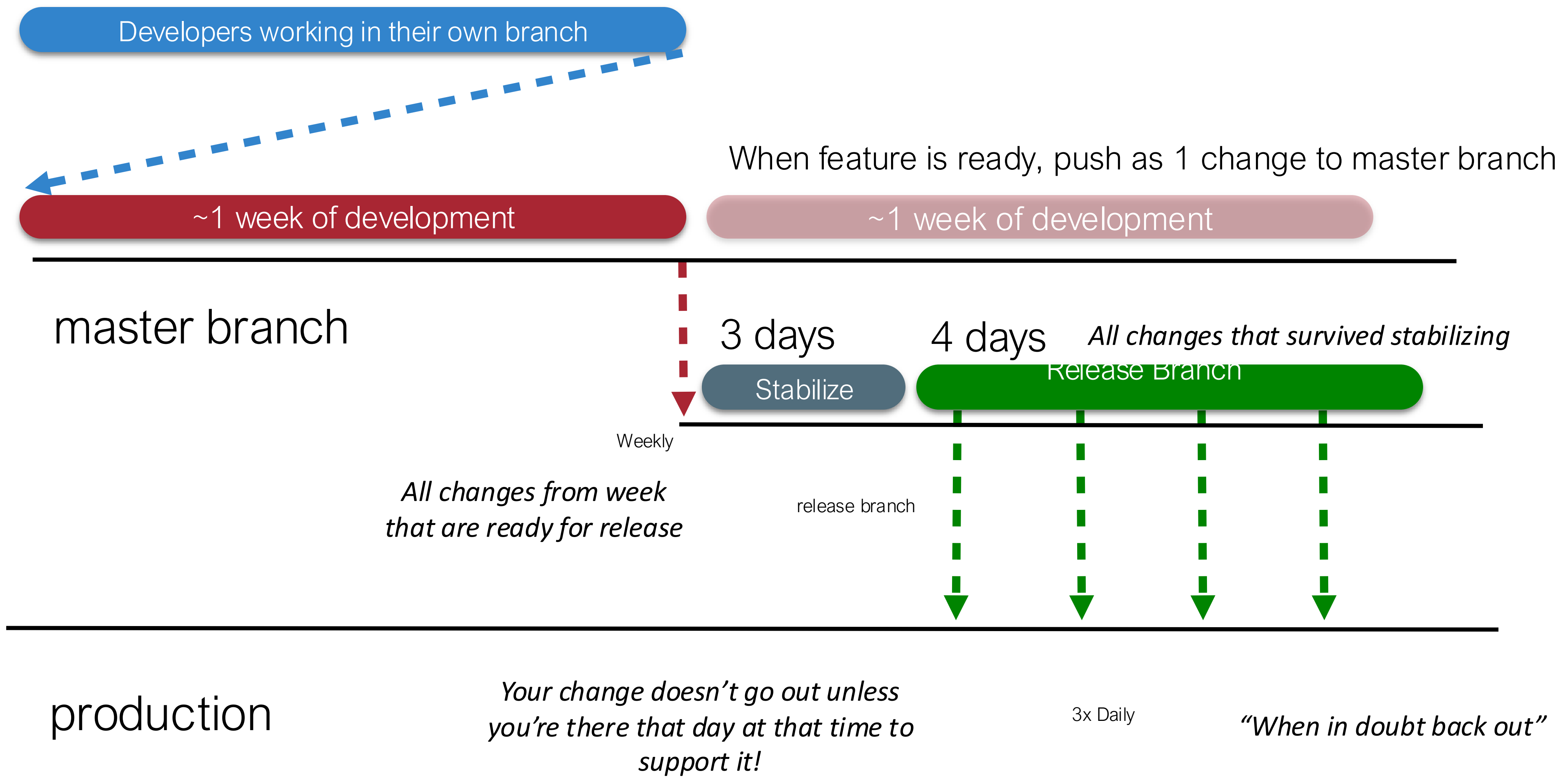
"Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly), pp. 494-497

Facebook: "Move fast and break things"

- de-prioritize unit tests
- Emphasis on getting features to users quickly
- Strategy: push many small changes to fractions of the user base. ("split deployments")

Deployment Example: Facebook.com

- Pre-2016


Developers working in their own branch
When feature is ready, push as 1 change to master branch
~1 week of development
~1 week of development
master branch
3 days
4 days
All changes that survived stabilizing
Stabilize
Release Branch
Weekly
All changes from week that are ready for release
release branch
production
Your change doesn't go out unless you're there that day at that time to support it!
3x Daily
"When in doubt back out"

Facebook used to have an elaborate system of branches

- dev branches got merged into master,
- then once a week all changes from the past week were pulled into a release branch (often 10,000 changes per week)
- For 3 days they “stabilized” the release branch – find changes that are causing very bad behavior and back them out. (manual process!!)
- Then for the last 4 days of the week, every change that survived that stabilization got *individually pushed* to production batched so that this happens 3x/day.
- Important to do small deploys so that you could isolate bad changes.

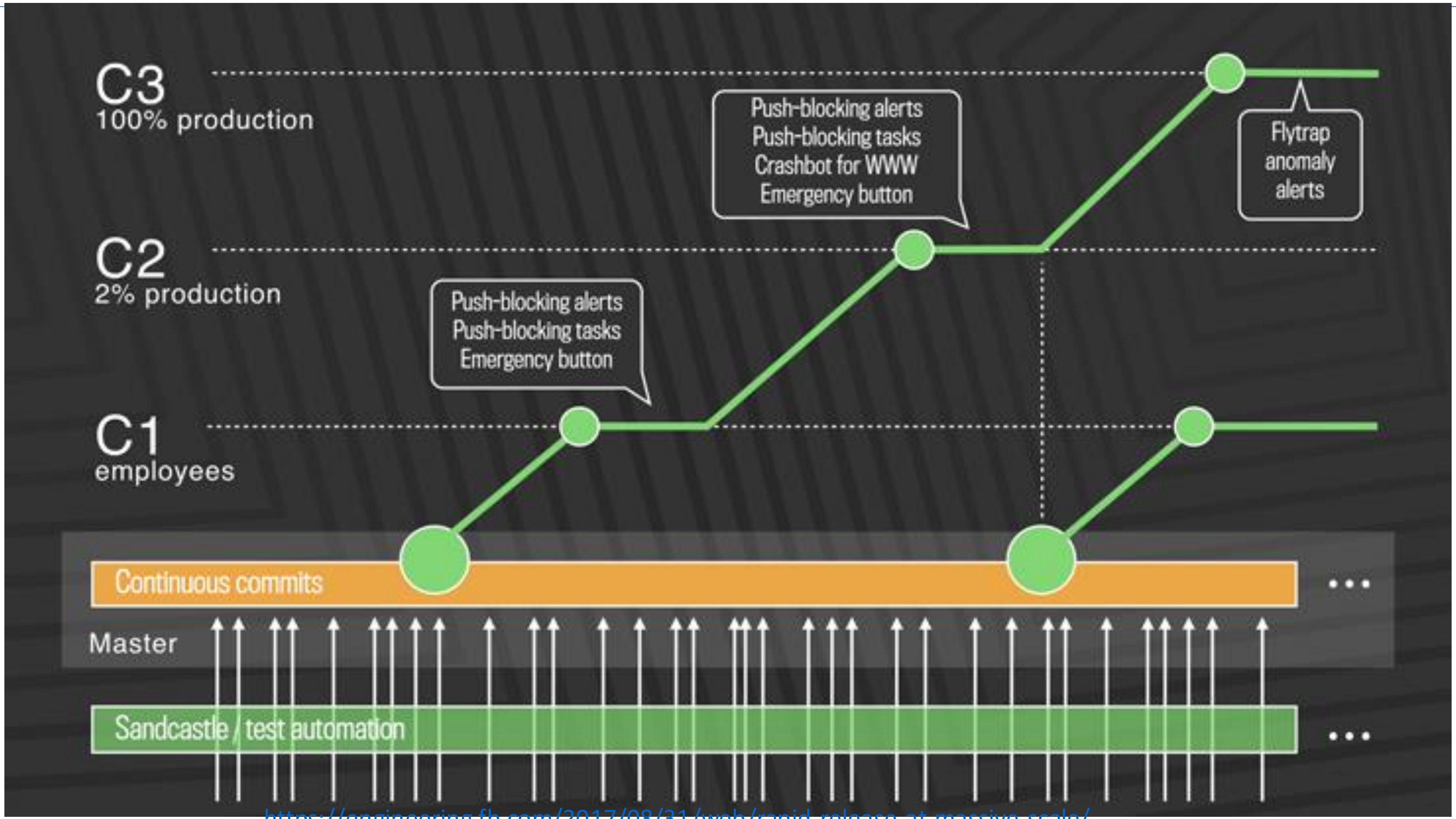
Deployment Example

"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**"

- Chuck Rossi, Director Software Infrastructure & Release Engineering @ Facebook

"Rapid release at massive scale" https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

Post-2016: truly continuous releases from master branch



https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

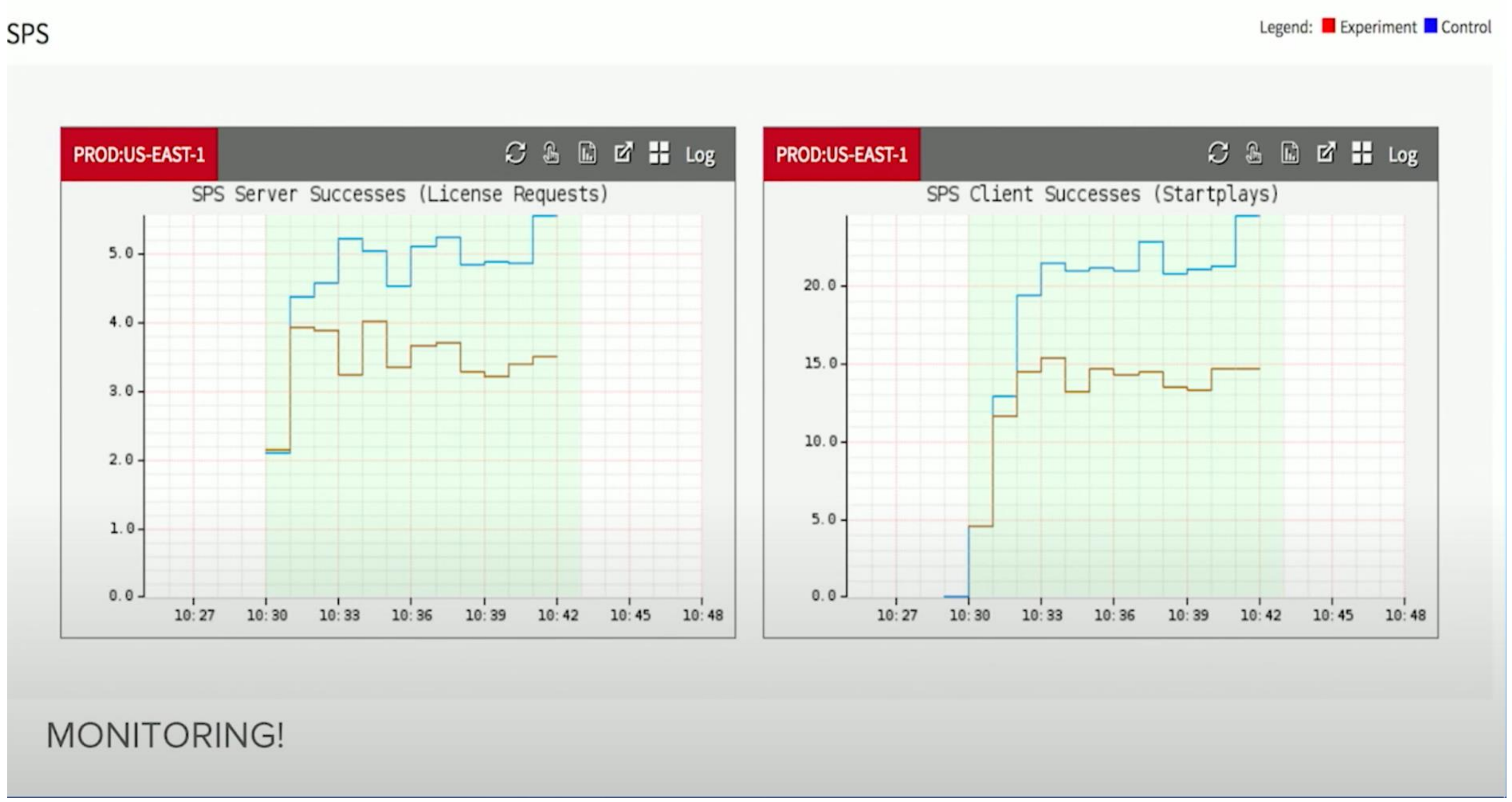
Post-2016: Truly Continuous Releases from Master Branch (excerpts from blog post)

1. First, diffs that have passed a series of automated internal tests and land in master are pushed out to Facebook employees.
2. In this stage, get push-blocking alerts if we've introduced a regression, and an emergency stop button lets us keep the release from going any further.
3. If everything is OK, push the changes to 2 percent of production, where again we collect signal and monitor alerts, especially for edge cases that our testing or employee dogfooding may not have picked up.
4. Finally, roll out to 100 percent of production, where our Flytrap tool aggregates user reports and alerts us to any anomalies.
5. Many of the changes are initially kept behind feature flags, which allows to roll out mobile and web code releases independently from new features, helping to lower the risk of any particular update causing a problem.
6. If we do find a problem, simply switch the feature off rather than revert back to a previous version or fix forward.

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

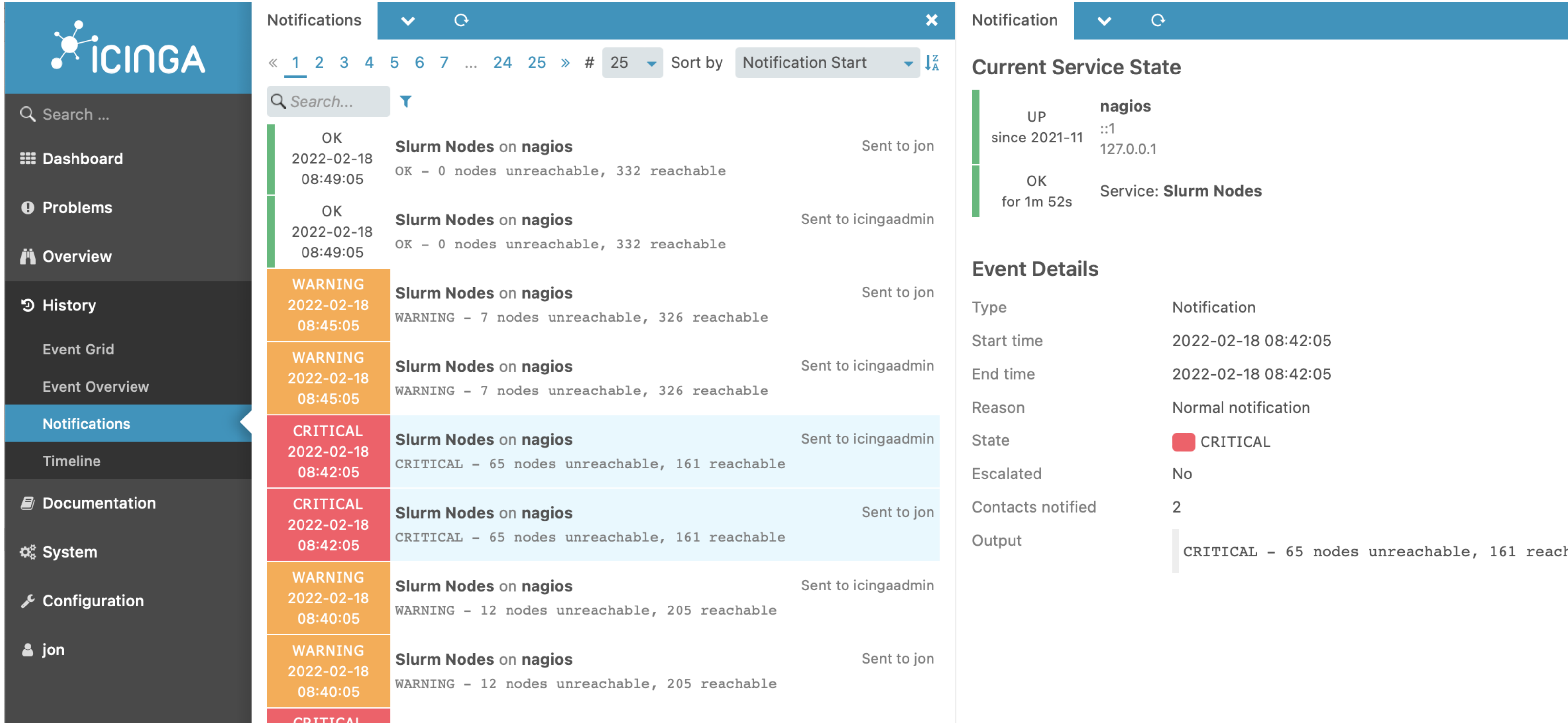
Continuous Delivery Tools Can Take Automated Actions

- Example: Automated roll-back of updates at Netflix based on "streams-per-second" (SPS)



https://www.youtube.com/watch?v=qyzymLlj9ag

Monitoring Services Can Take Automated Actions



State	Notification	Sent to
OK 2022-02-18 08:49:05	**Slurm Nodes** on **nagios** `OK - 0 nodes unreachable, 332 reachable`	Sent to jon
OK 2022-02-18 08:49:05	**Slurm Nodes** on **nagios** `OK - 0 nodes unreachable, 332 reachable`	Sent to icingaadmin
WARNING 2022-02-18 08:45:05	**Slurm Nodes** on **nagios** `WARNING - 7 nodes unreachable, 326 reachable`	Sent to jon
WARNING 2022-02-18 08:45:05	**Slurm Nodes** on **nagios** `WARNING - 7 nodes unreachable, 326 reachable`	Sent to icingaadmin
CRITICAL 2022-02-18 08:42:05	**Slurm Nodes** on **nagios** `CRITICAL - 65 nodes unreachable, 161 reachable`	Sent to icingaadmin
CRITICAL 2022-02-18 08:42:05	**Slurm Nodes** on **nagios** `CRITICAL - 65 nodes unreachable, 161 reachable`	Sent to jon
WARNING 2022-02-18 08:40:05	**Slurm Nodes** on **nagios** `WARNING - 12 nodes unreachable, 205 reachable`	Sent to icingaadmin
WARNING 2022-02-18 08:40:05	**Slurm Nodes** on **nagios** `WARNING - 12 nodes unreachable, 205 reachable`	Sent to jon
CRITICAL		

Current Service State

UP since 2021-11 — **nagios** ::1 127.0.0.1

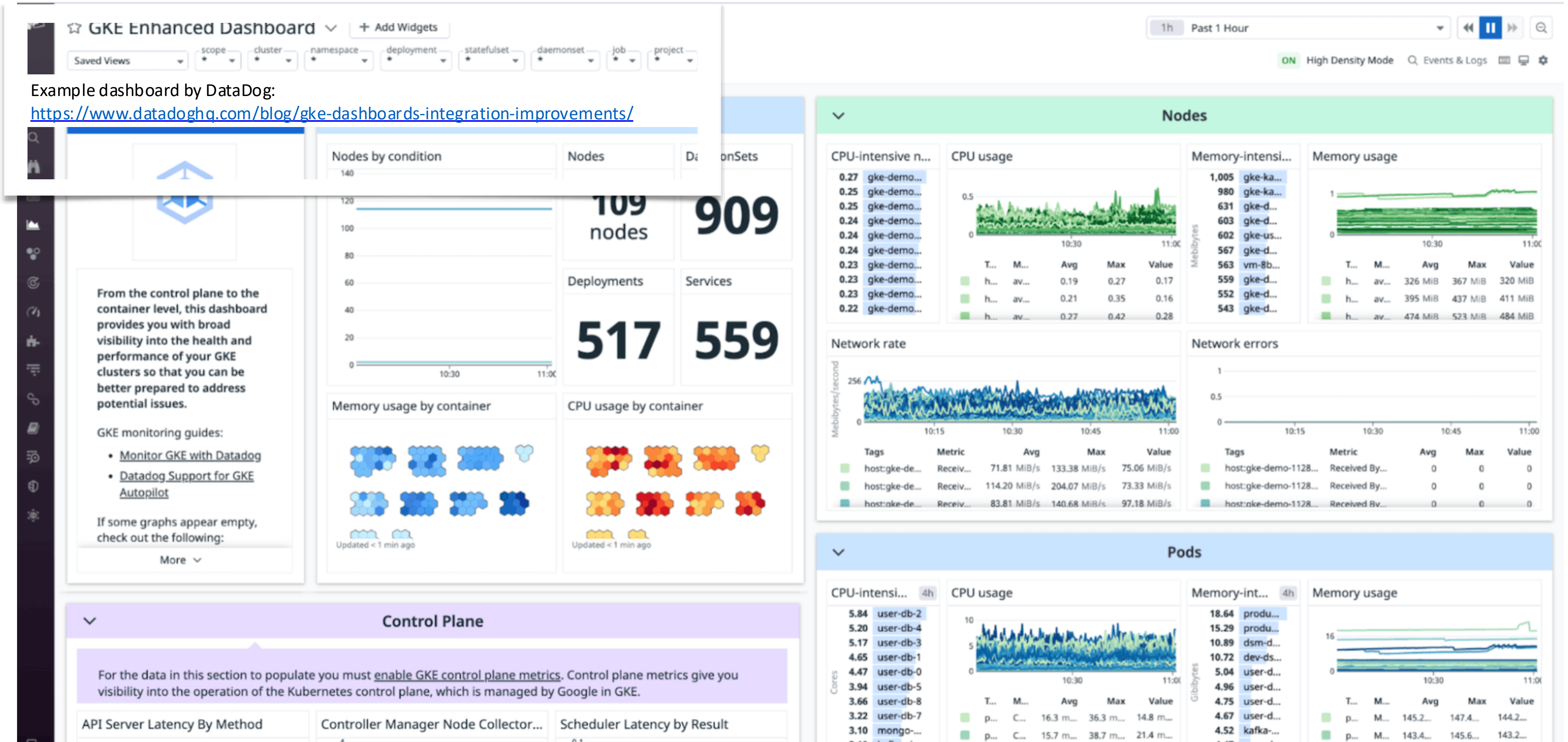
OK for 1m 52s — Service: **Slurm Nodes**

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 reach`

From Monitoring to Observability

- Understanding what is going on inside of our deployed systems by visualizing internal metrics



Example dashboard by DataDog:
https://www.datadoghq.com/blog/gke-dashboards-integration-improvements/

109 nodes

909

517

559

From the control plane to the container level, this dashboard provides you with broad visibility into the health and performance of your GKE clusters so that you can be better prepared to address potential issues.

GKE monitoring guides:

- Monitor GKE with Datadog
- Datadog Support for GKE Autopilot

If some graphs appear empty, check out the following:

For the data in this section to populate you must enable GKE control plane metrics. Control plane metrics give you visibility into the operation of the Kubernetes control plane, which is managed by Google in GKE.

Tags	Metric	Avg	Max	Value
host:gke-de...	Receiv...	71.81 MiB/s	133.38 MiB/s	75.06 MiB/s
host:gke-de...	Receiv...	114.20 MiB/s	204.07 MiB/s	73.33 MiB/s
host:gke-de...	Receiv...	83.81 MiB/s	140.68 MiB/s	97.18 MiB/s

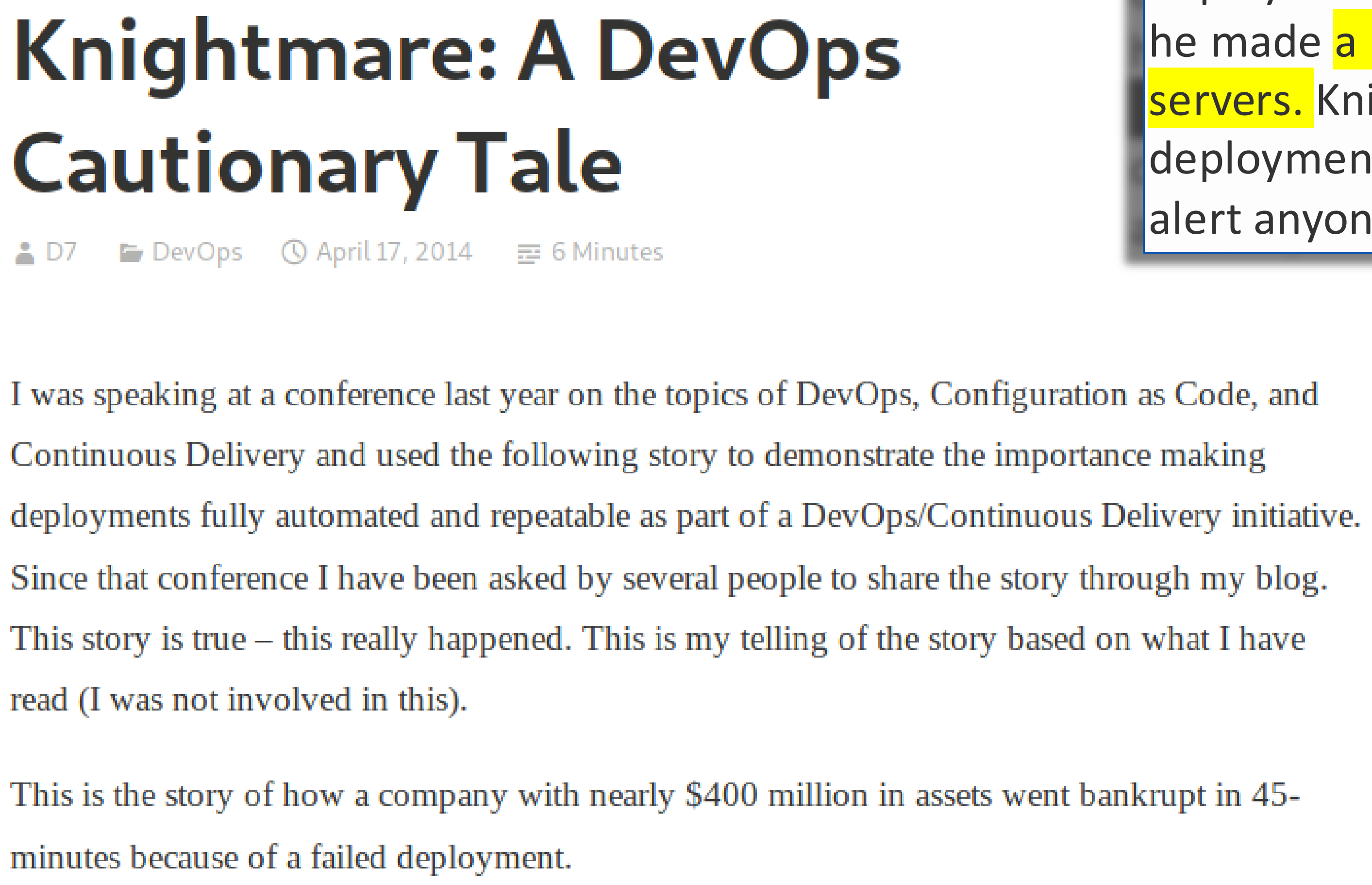
Tags	Metric	Avg	Max	Value
host:gke-demo-1128...	Received By...	0	0	0
host:gke-demo-1128...	Received By...	0	0	0
host:gke-demo-1128...	Received By...	0	0	0

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 not to do: Failed Deployment at Knight Capital

"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. "



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.

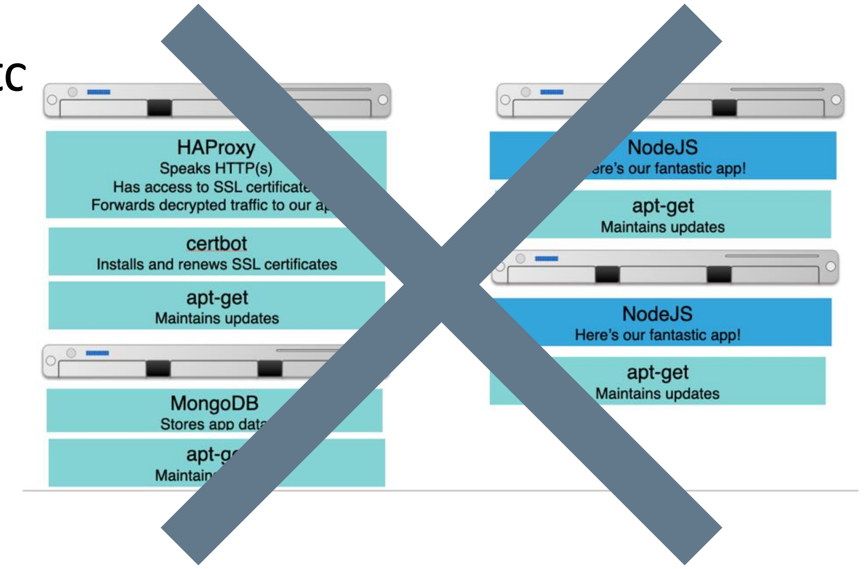
https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

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

Aside: Infrastructure As Code

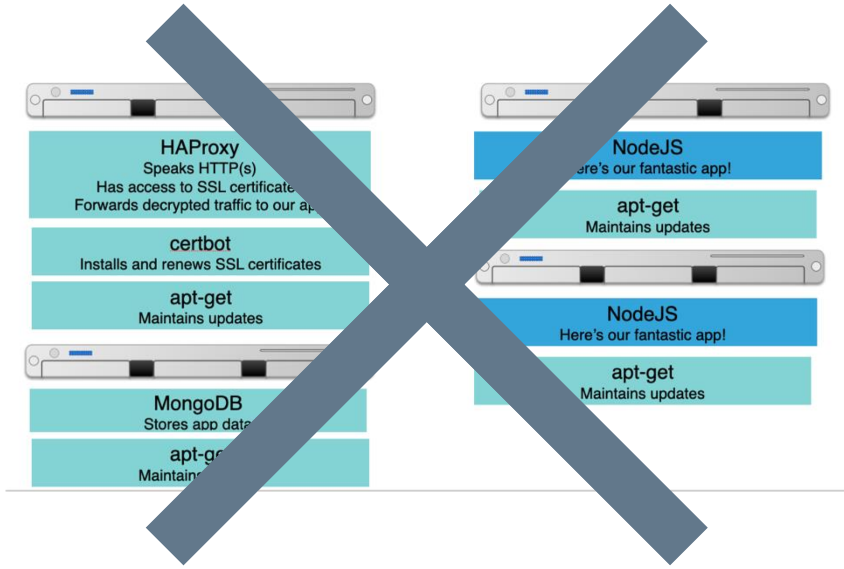
- Provisioning servers is tedious and error prone
  - Deploy a VM, then ssh to it, install some packages, etc
- Keeping servers up-to-date is also a struggle
- Ideal:
  - "Give me HAProxy with some configuration file, and keep that configuration in a git repo, and when I change it, roll out an update"
  - "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"



HAProxy
Speaks HTTP(s)
Has access to SSL certificate
Forwards decrypted traffic to our a
certbot
Installs and renews SSL certificates
apt-get
Maintains updates
MongoDB
Stores app data
NodeJS
Here's our fantastic app!
apt-get
Maintains updates
NodeJS
Here's our fantastic app!
apt-get
Maintains updates

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)



Learning objectives for this lecture

- It’s the end of today’s lecture, so you should be able to…
  - Describe how continuous integration 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