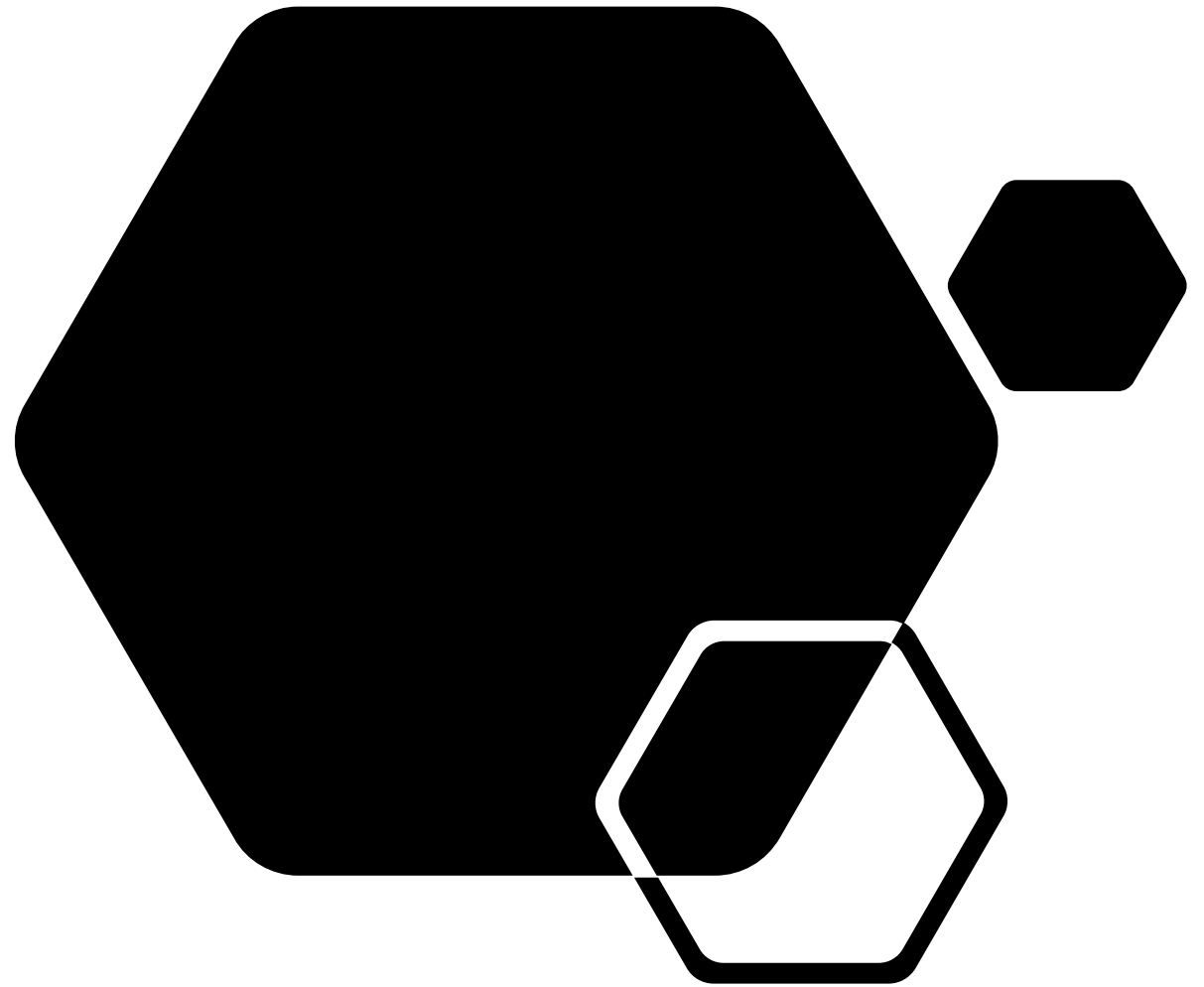


CSCI 690 - Week 3

Continuous Integration / Continuous
Delivery



Acknowledgment

- Jez Humble
- GitLab
- Armando Fox and David Patterson
- Martin Fowler
- Plutora
- IBM

Continuous Integration

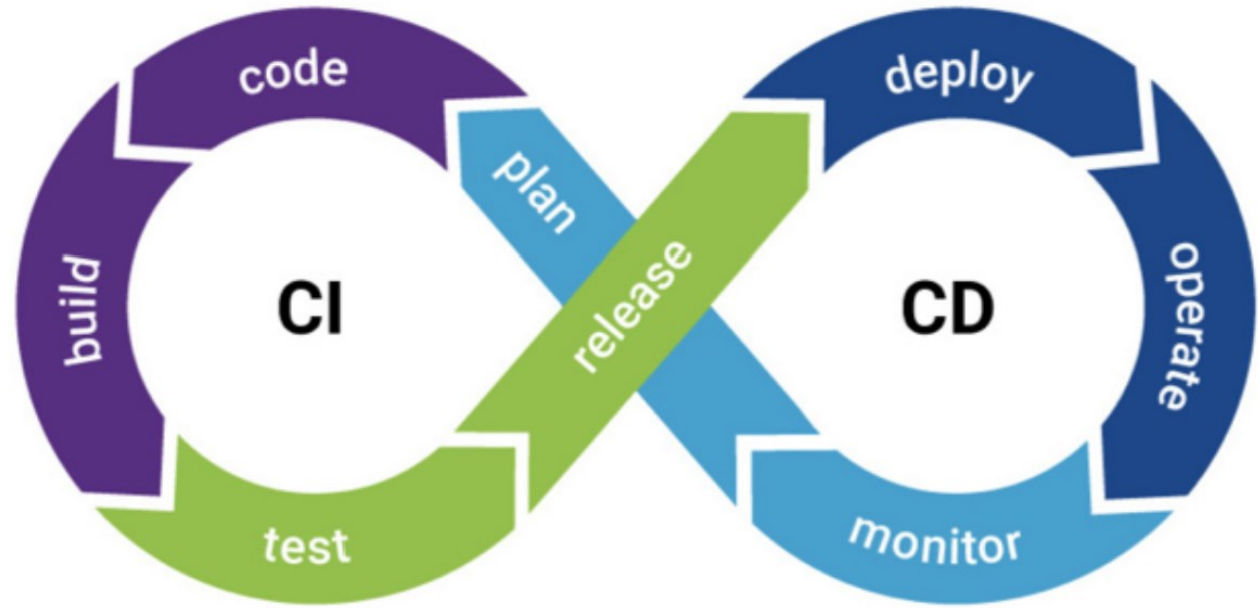
Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove. – Martin Fowler

Why CI?

- Differences between dev & production envs
- Cross-browser or cross-version testing
- Testing SOA integration when remote services act wonky
- Hardening: protection against attacks
- Stress testing/longevity testing of new features/code paths
- Example: Salesforce CI runs 150K+ tests and automatically opens bug report when test fails

Continuous Delivery (definition)



The ability to get changes—features, configuration changes, bug fixes, experiments—into production or into the hands of users safely and quickly in a sustainable way.

Why Continuous Delivery?



- Make releases painless, low risk event
- Reduce time to market
- Increase quality and stability
- Reduce cost of ongoing development
- Increase customer and employee satisfaction




Releases Then and Now

- Facebook: master branch pushed once a week, aiming for once a day (Bobby Johnson, Dir. of Eng., in late 2011)
- Amazon: several deploys per week
- StackOverflow: multiple deploys per day (Jeff Atwood, co-founder)
- GitHub: tens of deploys per day (Zach Holman)
- *Rationale: risk == # of engineer-hours invested in product since last deploy!*

*Like development and feature check-in, deployment should be a **non-event** that happens all the time*



Before CI/CD	With CI/CD
Annual releases	Frequent releases
Waterfall approach	Collaborative approach
Coding in isolation	New code regularly checked into shared repository
Merge conflicts	Seamless merges
Manual testing	Automated testing
Pre-production bottlenecks	Code is production-ready at all times
Manual deployments	Automated deployments
Feedback gathered too late for next release	Rapid feedback loop



Why Continuous Delivery?

“The flaw in the Apache Struts framework was fixed on March 6. Three days later, the bug was already [under mass attack](#) by hackers who were exploiting the flaw to install rogue applications on Web servers. Five days after that, the exploits [showed few signs of letting up](#). Equifax has said the breach on its site occurred in mid-May, more than two months after the flaw came to light and a patch was available.”

<https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug/>

Deployment vs. Delivery (Plutora)

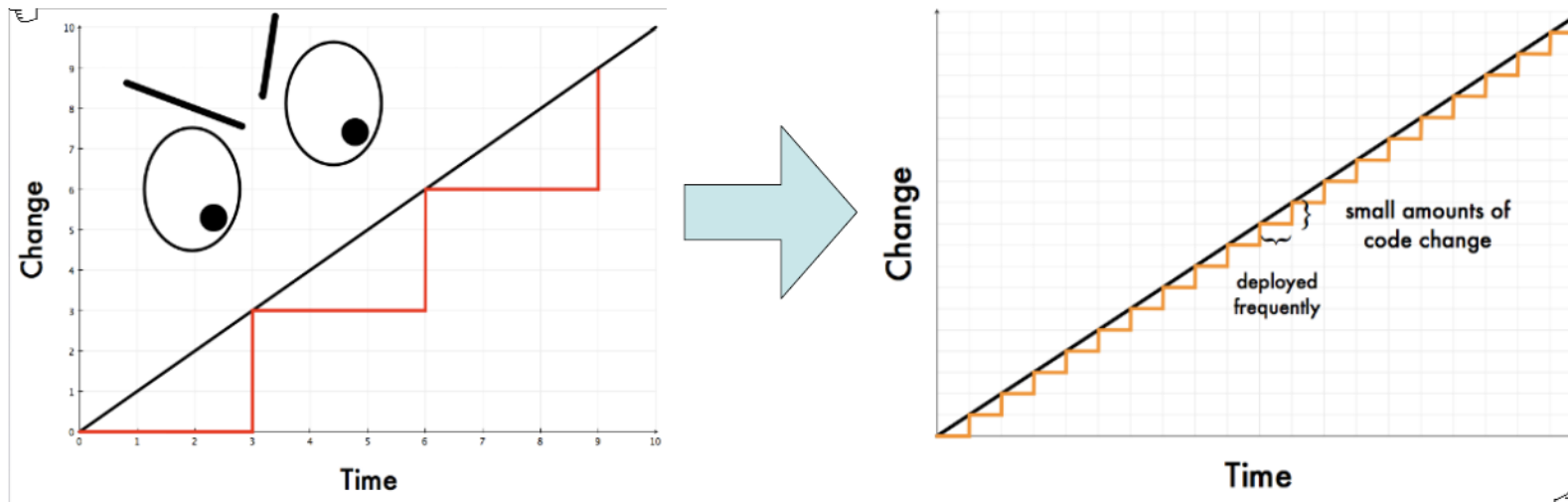
Continuous Deployment



Continuous Delivery



Releasing frequently



John Allspaw: "Ops Metametrics" <http://slidesha.re/dsSZlr>

Optimize for MTRS



2016 IT Performance by Cluster

	High IT Performers	Medium IT Performers	Low IT Performers
Deployment frequency <i>For the primary application or service you work on, how often does your organization deploy code?</i>	On demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months
Lead time for changes <i>For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?</i>	Less than one hour	Between one week and one month	Between one month and 6 months
Mean time to recover (MTTR) <i>For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?</i>	Less than one hour	Less than one day	Less than one day*
Change failure rate <i>For the primary application or service you work on, what percentage of the changes either result in degraded service or subsequently require remediation (e.g., lead to service impairment, service outage, require a hotfix, rollback, fix forward, patch)?</i>	0-15%	31-45%	16-30%

* Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers.

Continuous Delivery Ingredients

- Configuration management
- Continuous integration
- Continuous testing

Configuration management via VCS



Develop, local workstation

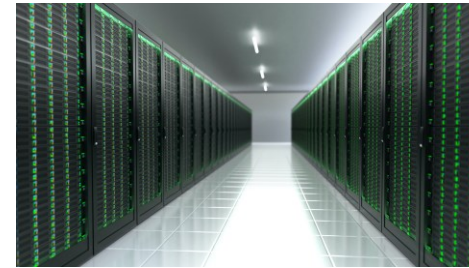


Build

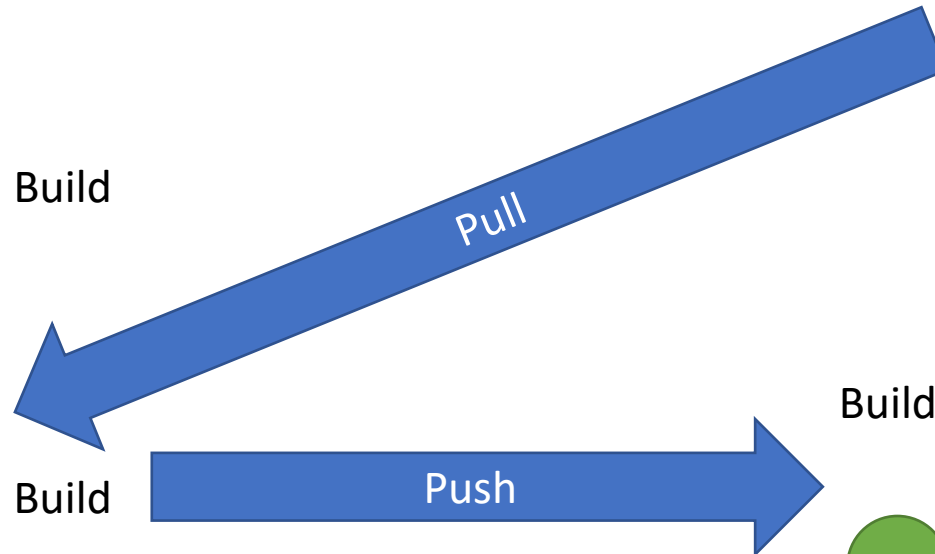


Build

Mainline Server



Build



Different kinds of testing

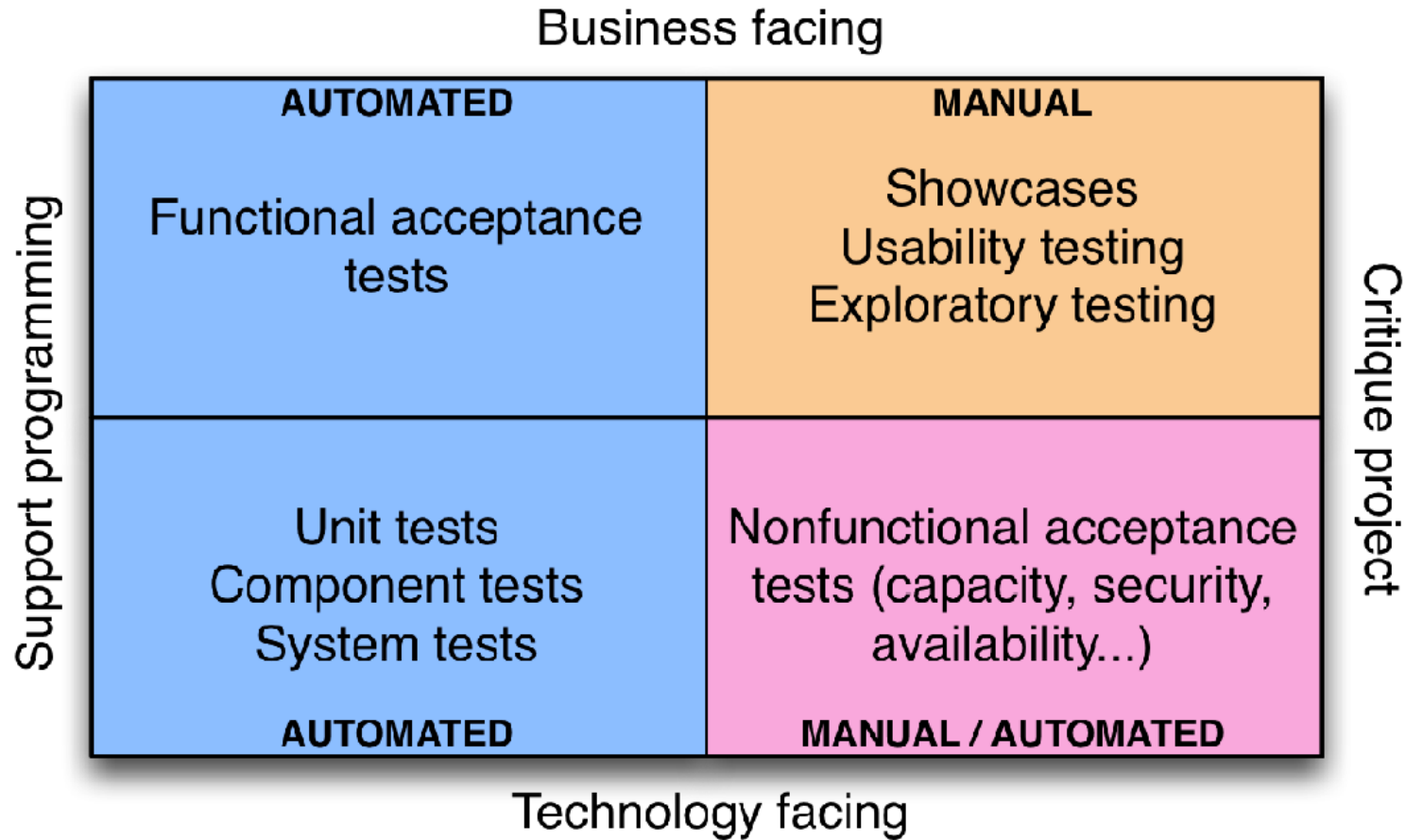
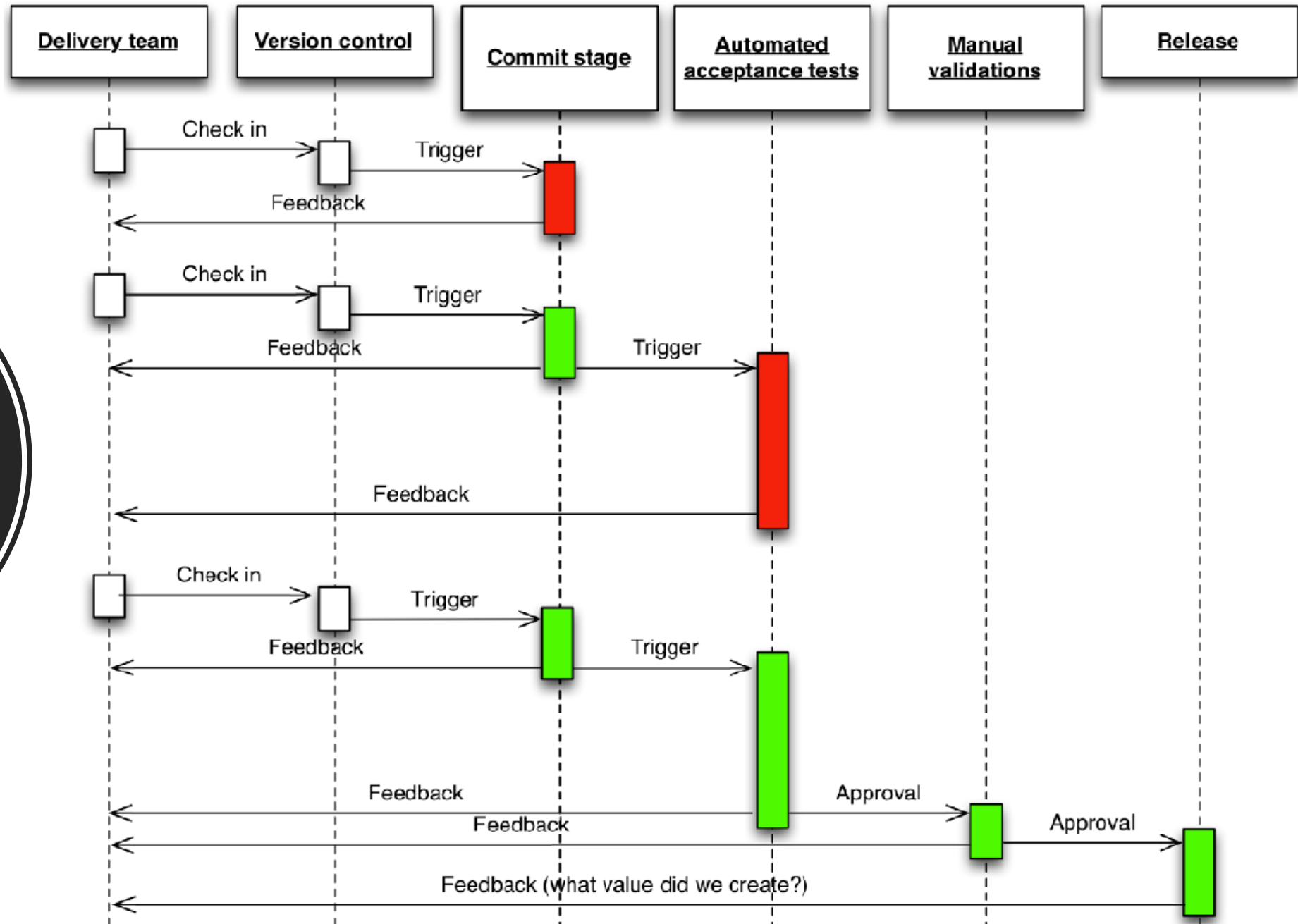
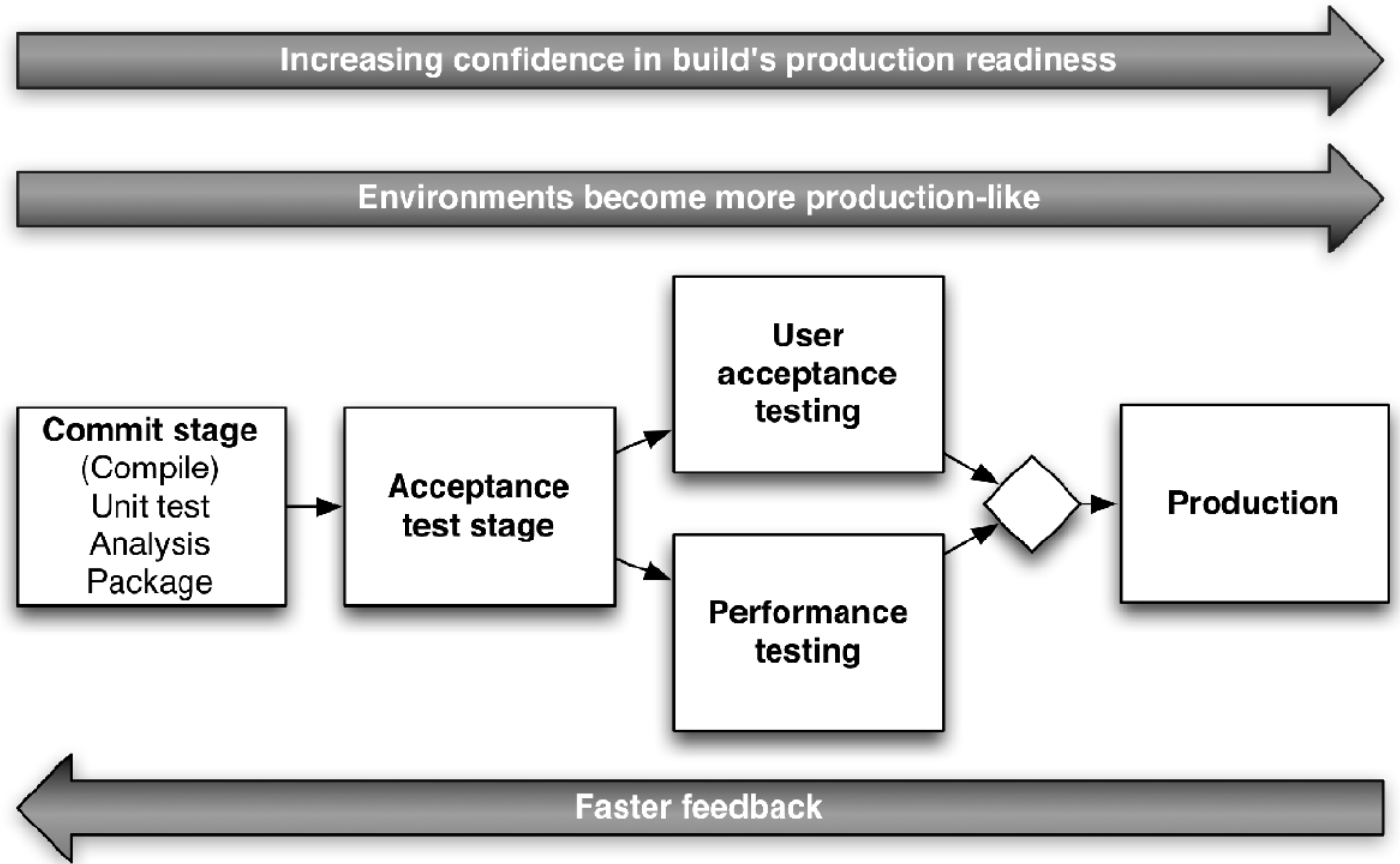


Diagram invented by Brian Marick

Deployment pipeline (Jez Humble)



Trade offs

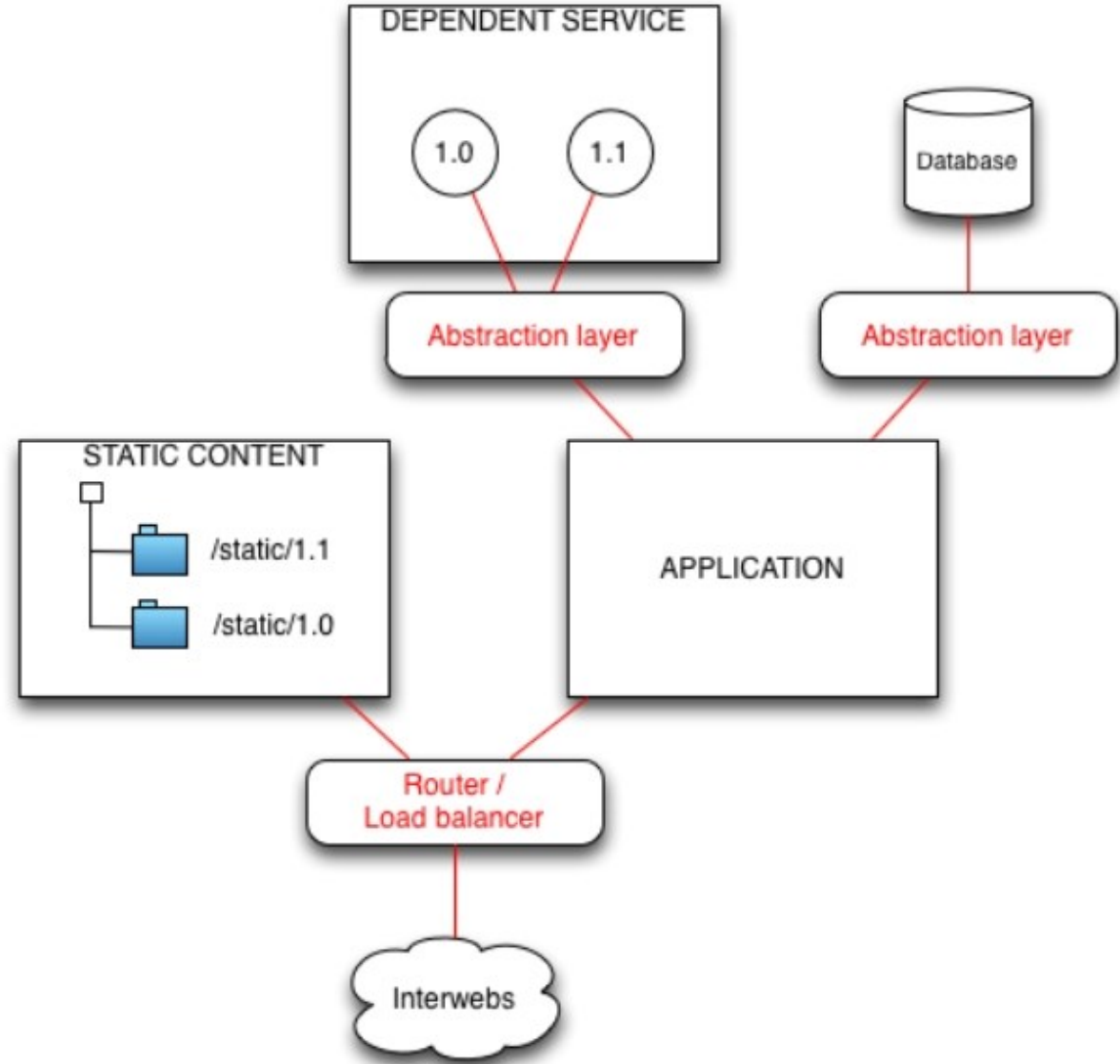


Release \neq
deploy

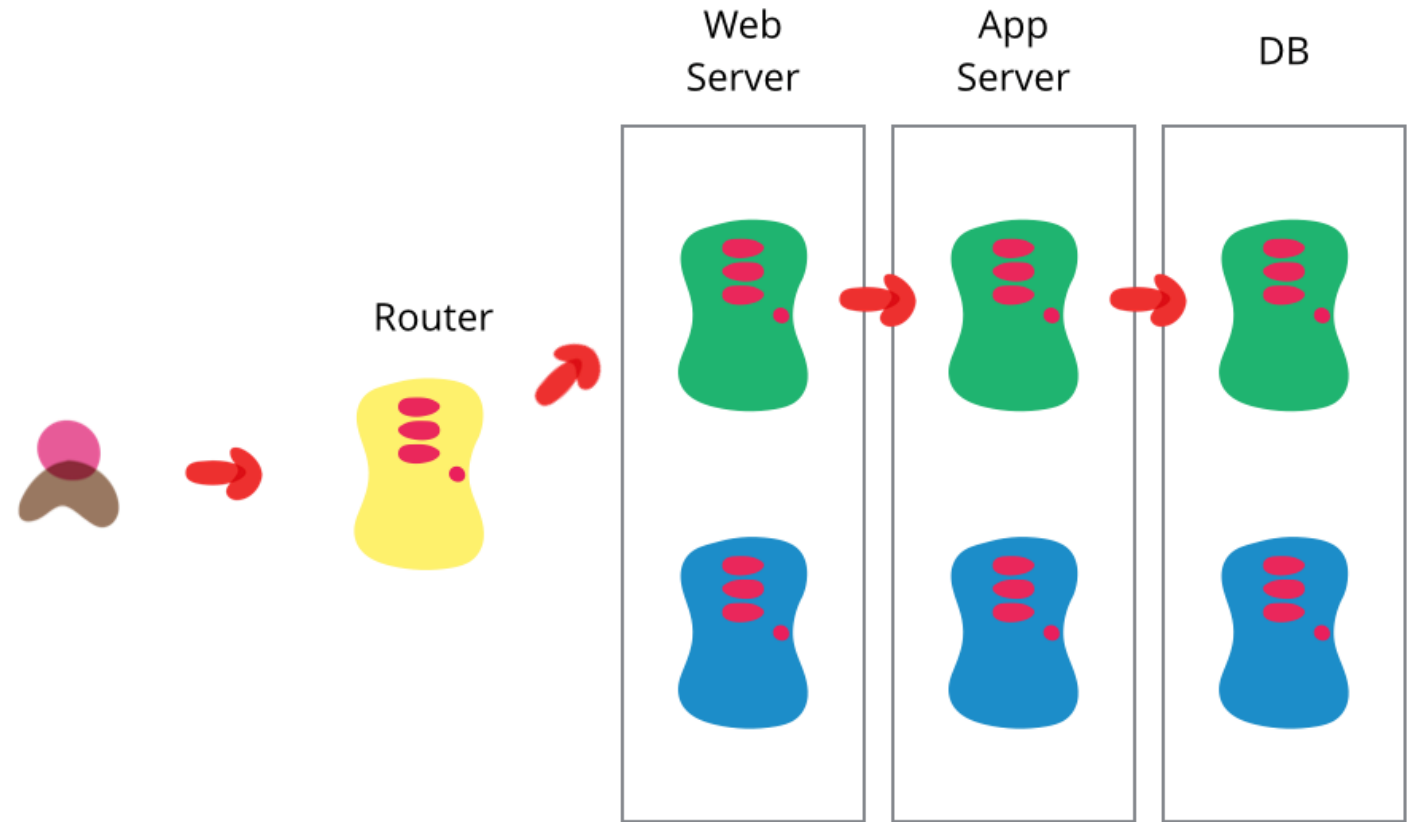
Low risk releases are incremental

- Blue-green deployments
- Feature flags/toggles
- Canary releases
- Dark launching
- Branch by abstraction

Static content?



Blue-green deployment



<https://martinfowler.com/bliki/BlueGreenDeployment.html>

The trouble with upgrades

- What if upgraded code is rolled out to many servers?
 - During rollout, some will have version n and others version $n+1$...will that work?
- What if upgraded code goes with schema migration?
 - Schema version $n+1$ breaks current code
 - New code won't work with current schema

Naïve update

1. Take service offline
 2. Apply [destructive migration](#), including data copying
 3. Deploy new code
 4. Bring service back online
- May result in unacceptable downtime

naive_migration.rb

```
1  class ChangeNameToFirstAndLast < ActiveRecord::Migration
2    def change
3      add_column 'moviegoers', 'first_name', :string
4      add_column 'moviegoers', 'last_name', :string
5      Moviegoer.all.each do |m|
6        m.update_attributes(:first => $1, :last => $2) if
7          m.name =~ /^(.*)\s+(.*)$/
8      end
9      remove_column 'moviegoers', 'name'
10   end
11 end
```


Incremental Upgrades with Feature Flags

1. Do [nondestructive migration](#)
2. [Deploy method protected by feature flag](#)
3. Flip feature flag on; if disaster, flip it back
4. Once all records moved, deploy new code without feature flag
5. Apply migration to remove old columns

schema_migration_a.rb

```
1  class SplitName1 < ActiveRecord::Migration
2    def change
3      add_column 'moviegoers', 'first_name', :string
4      add_column 'moviegoers', 'last_name', :string
5      add_column 'moviegoers', 'migrated', :boolean
6      add_index 'moviegoers', 'migrated'
7    end
8  end
```

[method with feature flags.rb](#)

```
1  class Moviegoer < ActiveRecord::Base
2    # here's version n+1, using Setler gem for feature flag:
3    scope :old_schema, where :migrated => false
4    scope :new_schema, where :migrated => true
5    def self.find_matching_names(string)
6      if Featureflags.new_name_schema
7        Moviegoer.new_schema.where('last_name LIKE :s OR first_name LIKE :s',
8          :s => "%#{string}%") +
9        Moviegoer.old_schema.where('name like ?', "%#{string}%")
10     else # use only old schema
11       Moviegoer.where('name like ?', "%#{string}%")
12     end
13   end
14   # automatically update records to new schema when they are saved
15   before_save :update_schema, :unless => lambda { |m| m.migrated? }
16   def update_schema
17     if name =~ /^(.*)\s+(.*)$/
18       self.first_name = $1
19       self.last_name = $2
20     end
21     self.migrated = true
22   end
23 end
```

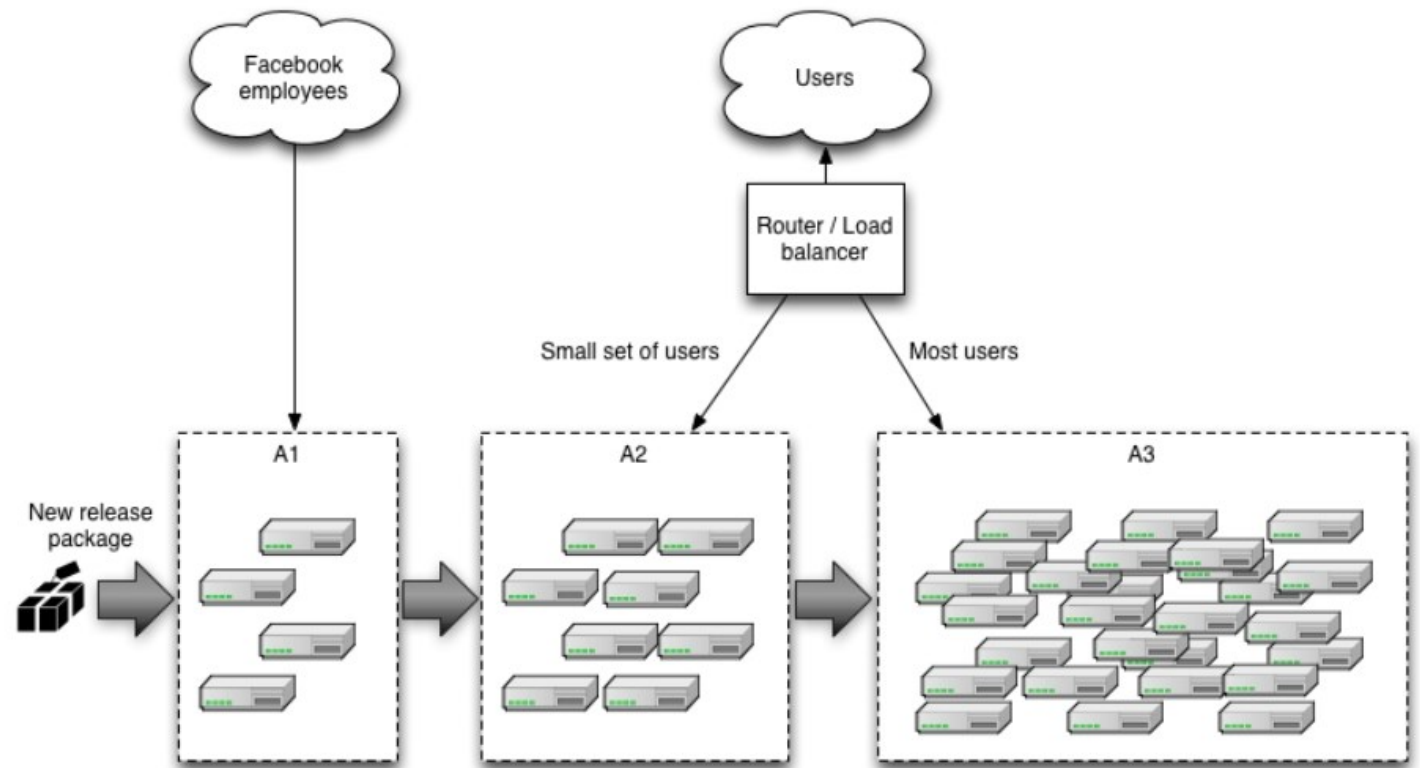
“Undoing” an upgrade

- Disaster strikes...use down-migration?
 - is it thoroughly tested?
 - is migration reversible?
 - are you sure someone else didn't apply an irreversible migration?
- Use feature flags instead
 - downmigrations are primarily for *development*

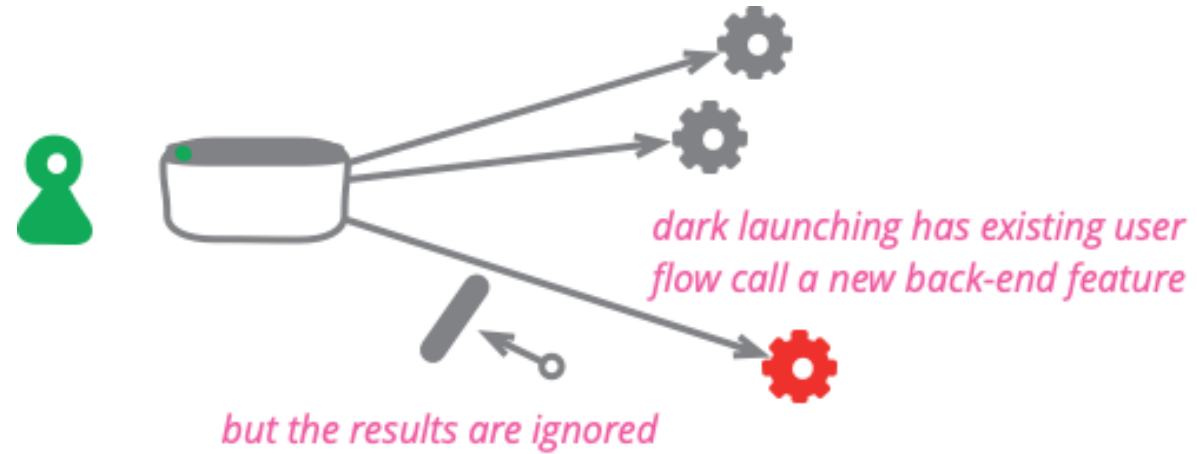
Other uses for feature flags

- Preflight checking: gradual rollout of feature to increasing numbers of users
 - to scope for performance problems, e.g.
- A/B testing
- Complex feature whose code spans multiple deploys

Canary Releasing



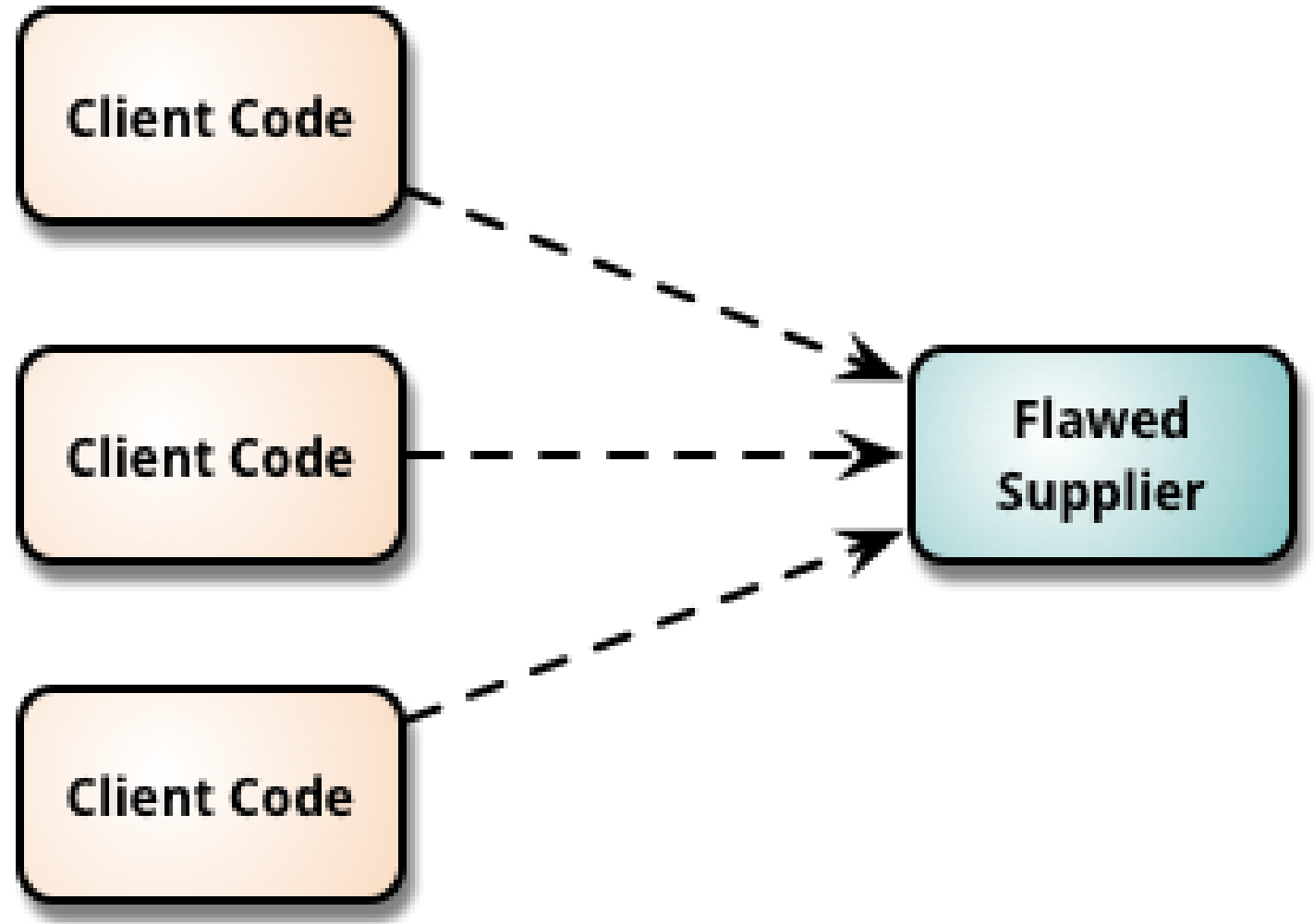
Dark Launching



this allows us to monitor the load on existing systems before the feature is officially released

<https://martinfowler.com/bliki/DarkLaunching.html>

Branch By Abstraction



Microservices architecture (IBM)

A cloud native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services that...

- have their own technology stack, inclusive of the database and data management model
- communicate with one another over a combination of REST APIs, event streaming, and message brokers
- are organized by business capability, with the line separating services often referred to as a bounded context

Value of microservices (IBM)

- Code can be updated more easily - new features or functionality can be added without touching the entire application
- Teams can use different stacks and different programming languages for different components.
- Components can be scaled independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.

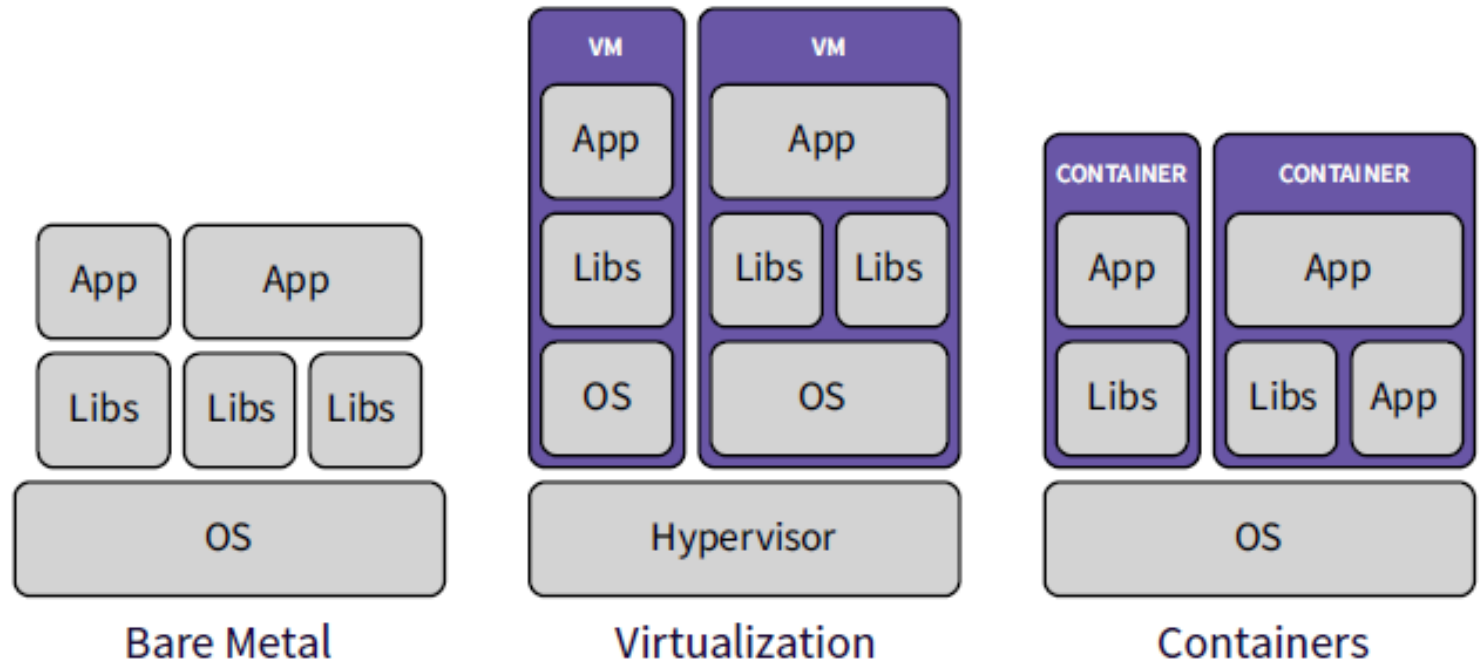
“Microservices both enable, and require, DevOps”



What is a container? (GitLab)

- A container is a method of operating system-based virtualization that allows you to securely run an application and its dependencies independently without impacting other containers or the operating system.
- Containers work much like a virtual machine except that, instead of packaging your code with an operating system, containers are run as a Linux process inside of the kernel.
- This means that each container only holds the code and dependencies needed to run that specific application, making them smaller and faster to run.

Virtualization



Containers retain the same repeatability factor as virtual machines, but are much faster and use less resources to run.



CI/CD with containers (GitLab)

- For DevOps teams, this means that developers can find and fix errors faster, reducing dependencies on operations, and effectively speeding up the development lifecycle.
- CI/CD coupled with containers has the potential to fully automate the build, test, and deploy stages of the software development lifecycle, making it easier and faster for developers to review their code and ensure that it is production-ready.

Infrastructure as Code (IaC) (IBM)

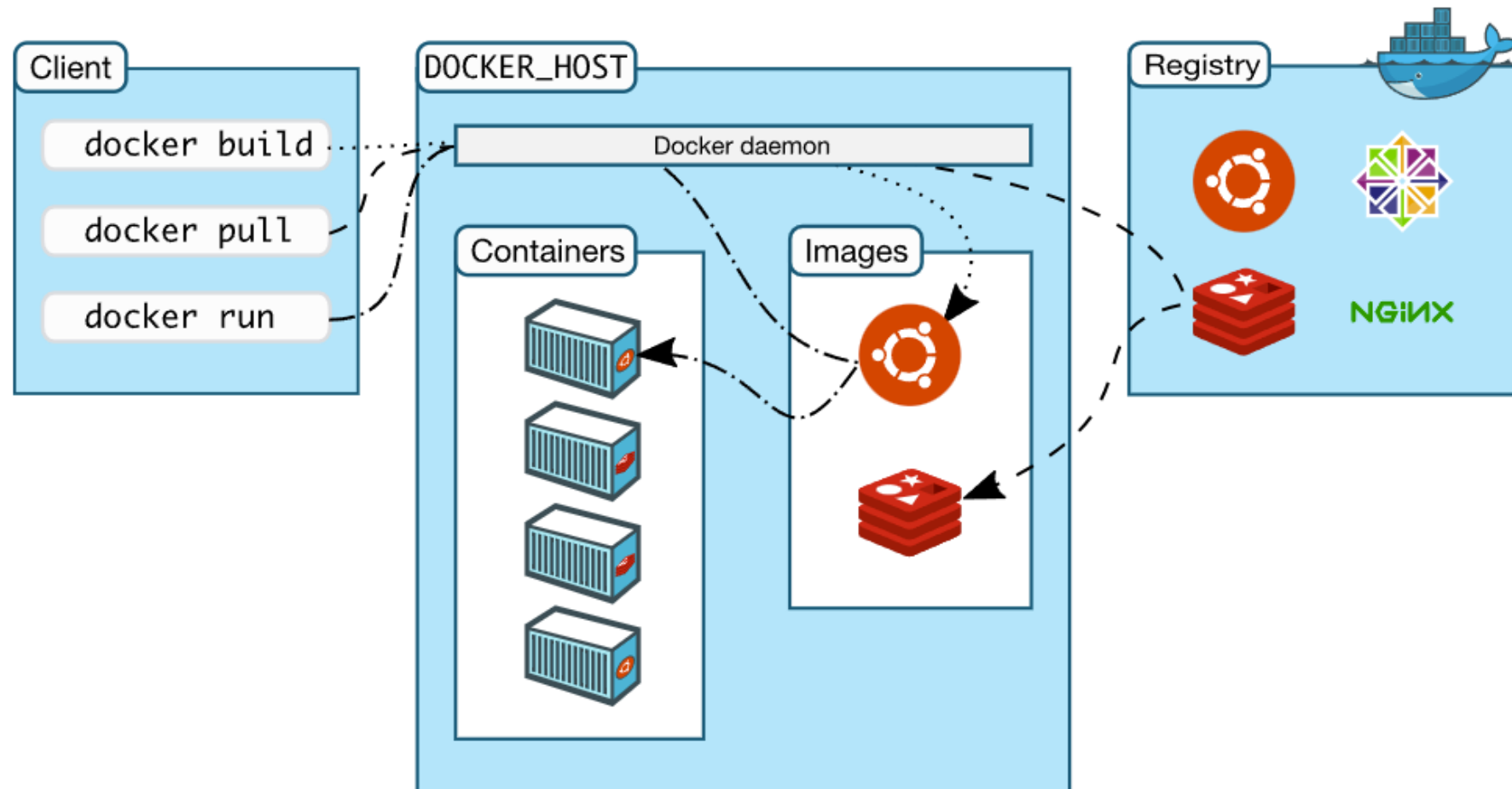
- A high-level descriptive coding language automates the provisioning of IT infrastructure.
- Infrastructure automation helps control costs, reduce risks, and respond with speed to new business opportunities and competitive threats.
- IaC is also an essential DevOps practice, indispensable to a competitively paced software delivery lifecycle.
- It enables DevOps teams rapidly create and version infrastructure in the same way they version source code and to track these versions.

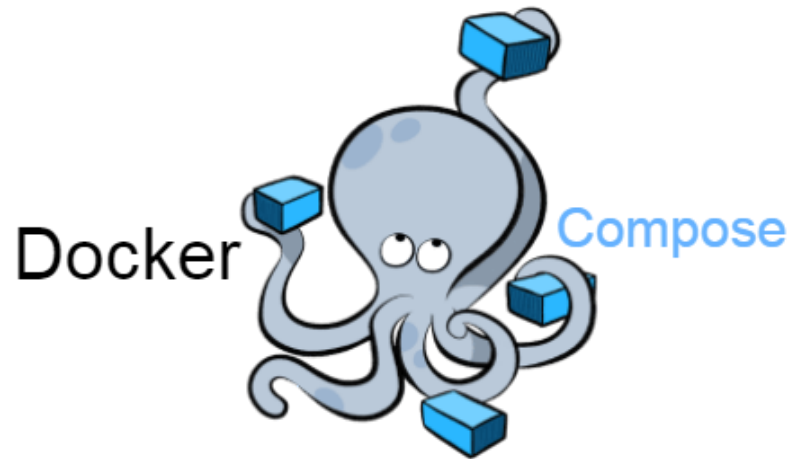


Enabling technologies

- Containers
- Docker
- Kubernetes
- Terraform
- ... and so many more

Docker architecture





Docker Compose

- Docker Compose is software used for defining and running multi-container Docker applications.
- It can handle multiple containers simultaneously in the production, staging, development, testing, and CI environment.

Kubernetes



Monolithic Application



Transition to Microservices



Docker

Create containers for your application



Kubernetes

Launch your containerised application in K8s



What's next

- Lecture quiz
- Reading and then discussion on Docker Compose/Kubernetes
- HW1: Getting started with Docker