Agile Alliance®

ORLANDO FLORIDA AUGUST 7-11

AGILE2017

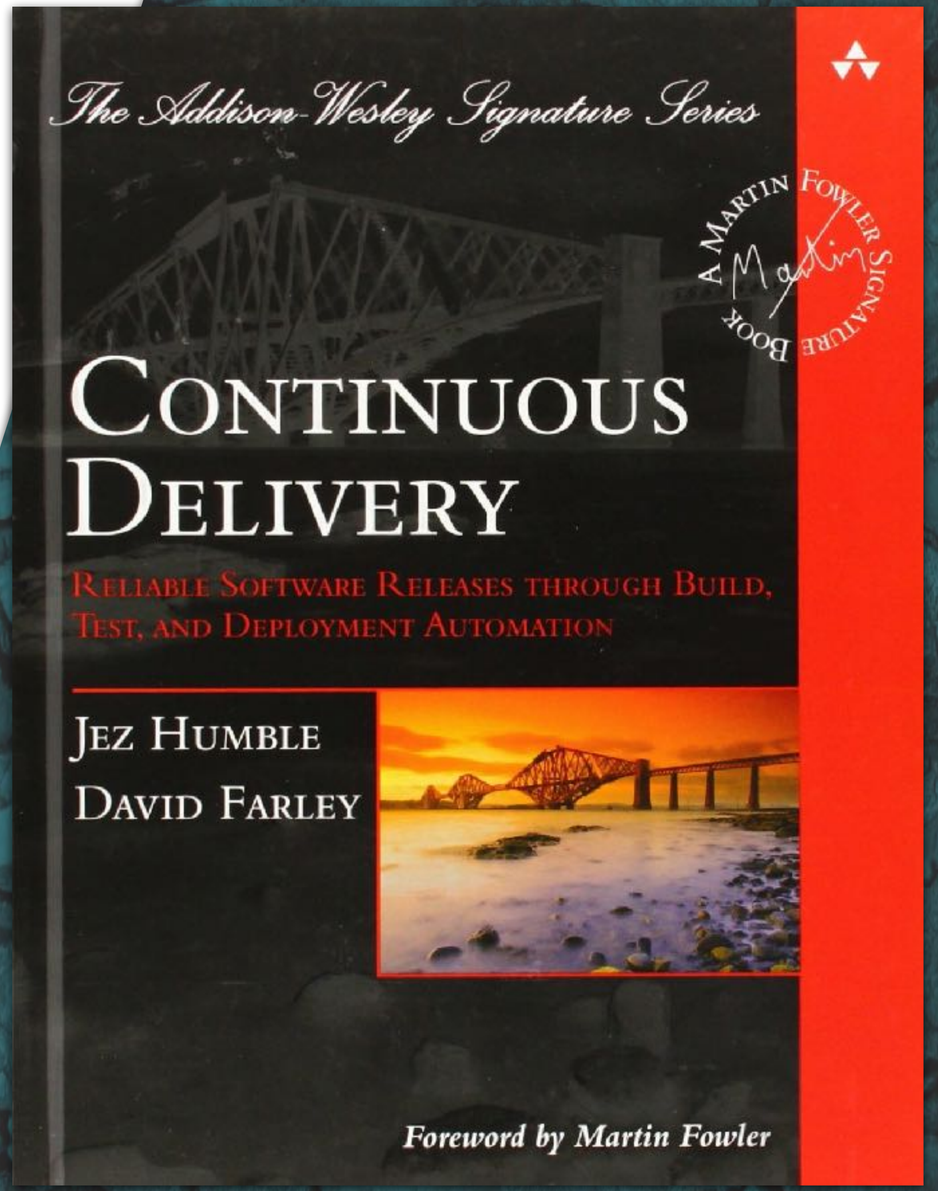
Rachel Laycock @rachellaycock

Continuous Delivery Explained

ThoughtWorks®

Workshop materials created by Jez Humble, Martin Fowler, Tom Sulston, Neal Ford & Rachel Laycock

Delivering value early and often



Release Cadence

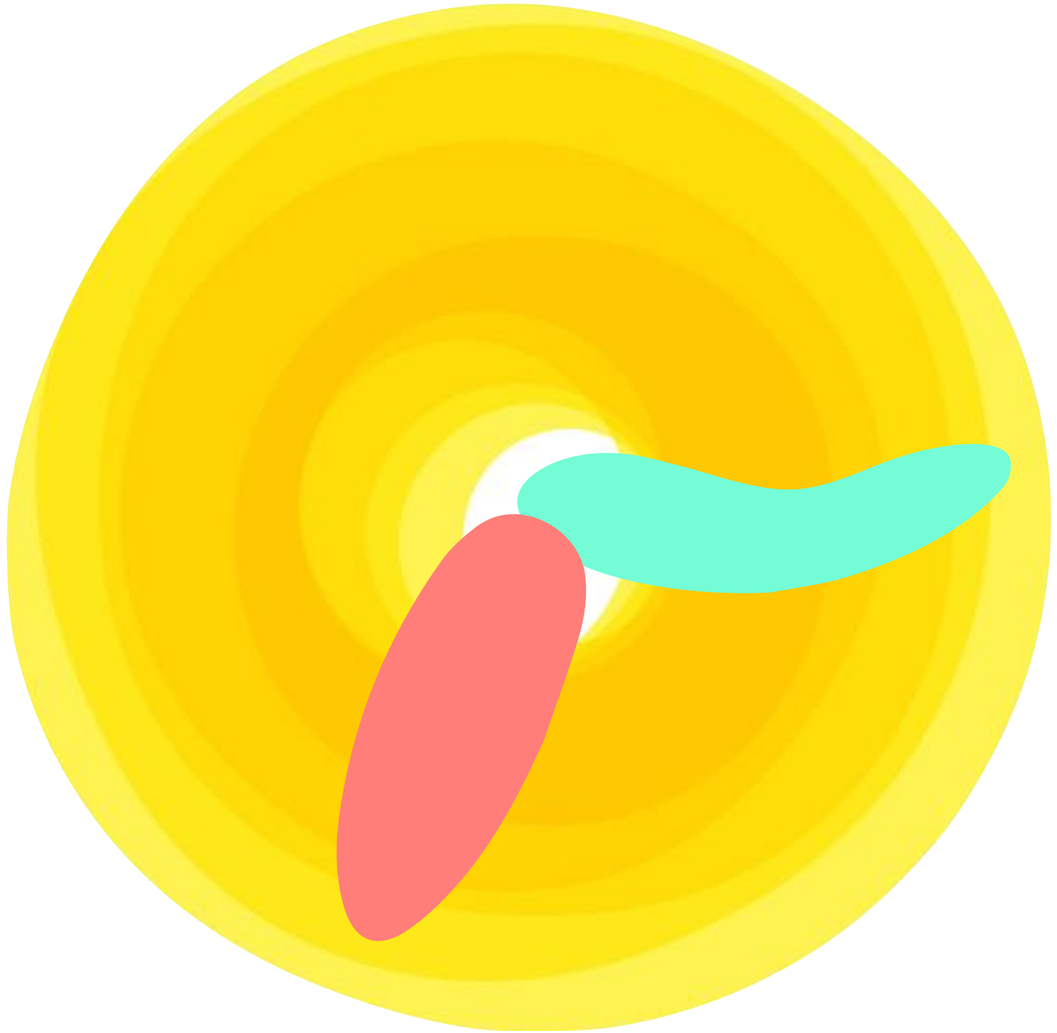
Mary Poppendieck

Tom Poppendieck

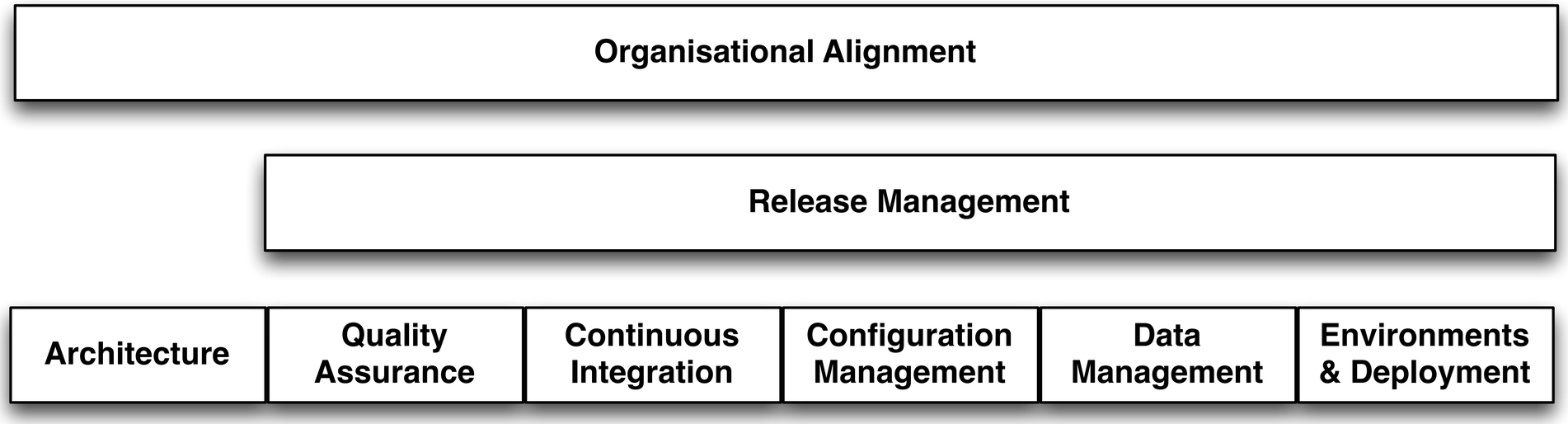
“How long would it take your organization to deploy a change that involves just one single line of code?”

“Can you do this on a repeatable, reliable basis?”

Release Cadence

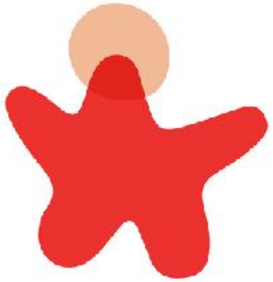


Continuous Delivery is **BIG**

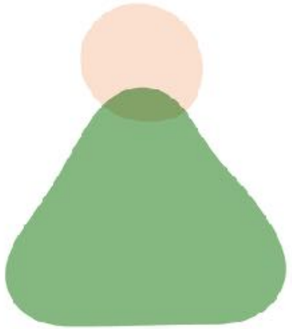


Who are You?

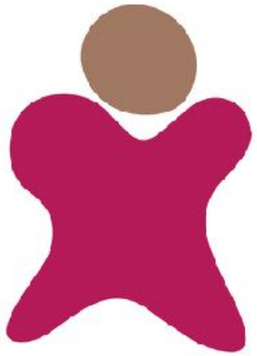
Manager



DBA



BA



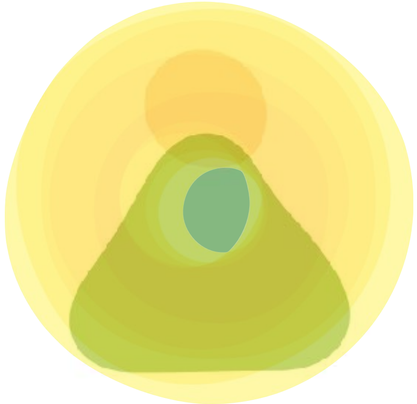
Operations

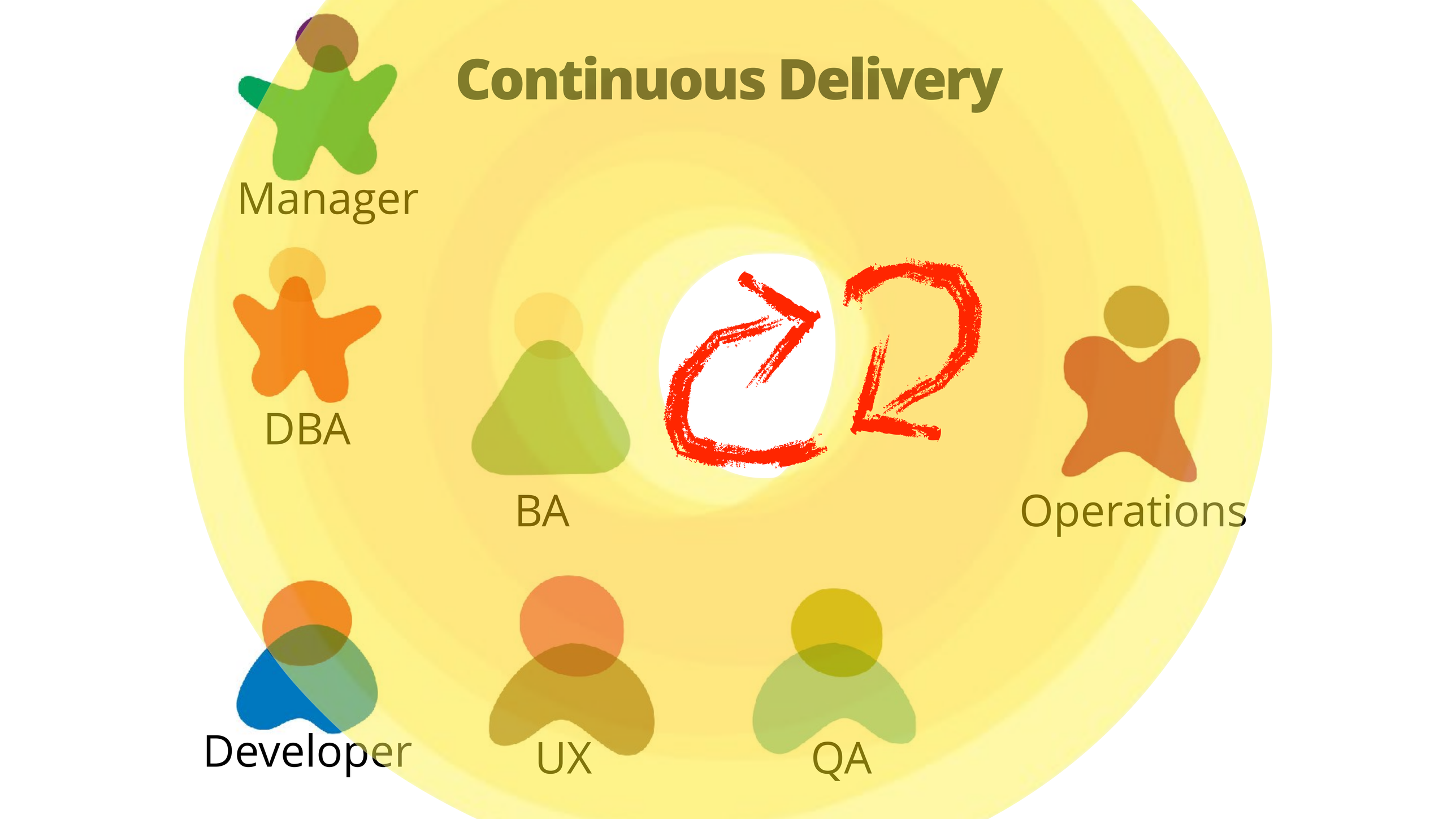
Developer

UX

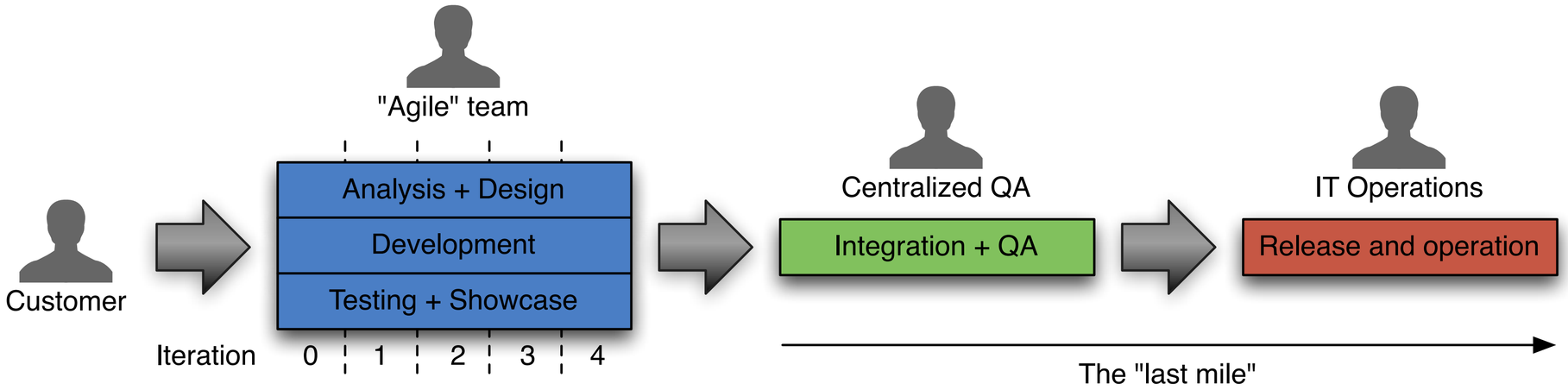
QA

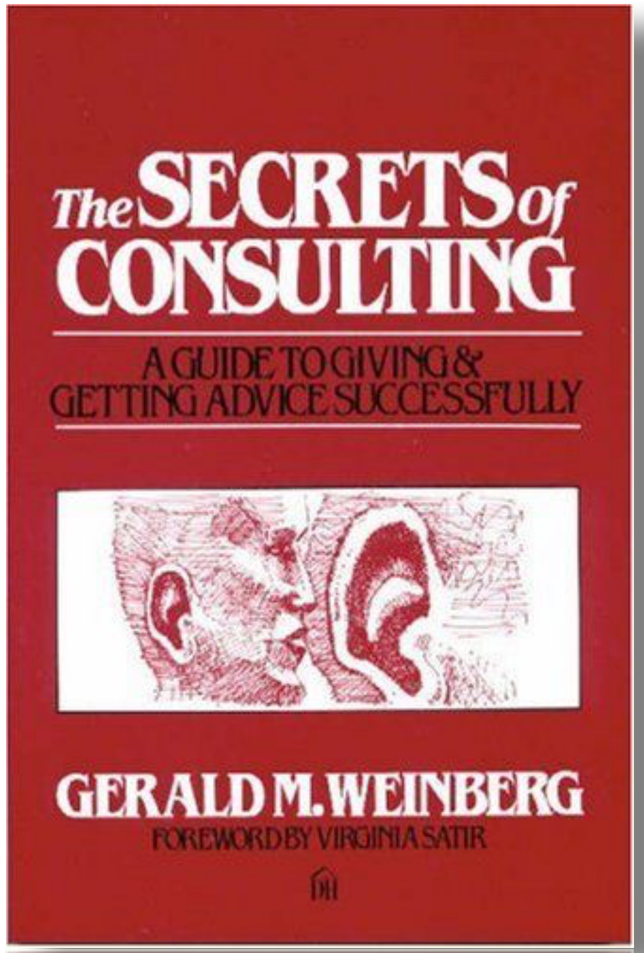
Agile Transformation




Continuous Delivery
Manager
DBA
BA
Operations
Developer
UX
QA

Agile 101


"Agile" team
Customer
Analysis + Design
Development
Testing + Showcase
Iteration
0
1
2
3
4
Centralized QA
Integration + QA
IT Operations
Release and operation
The "last mile"



“No matter how it looks at first, it's always a people problem.”

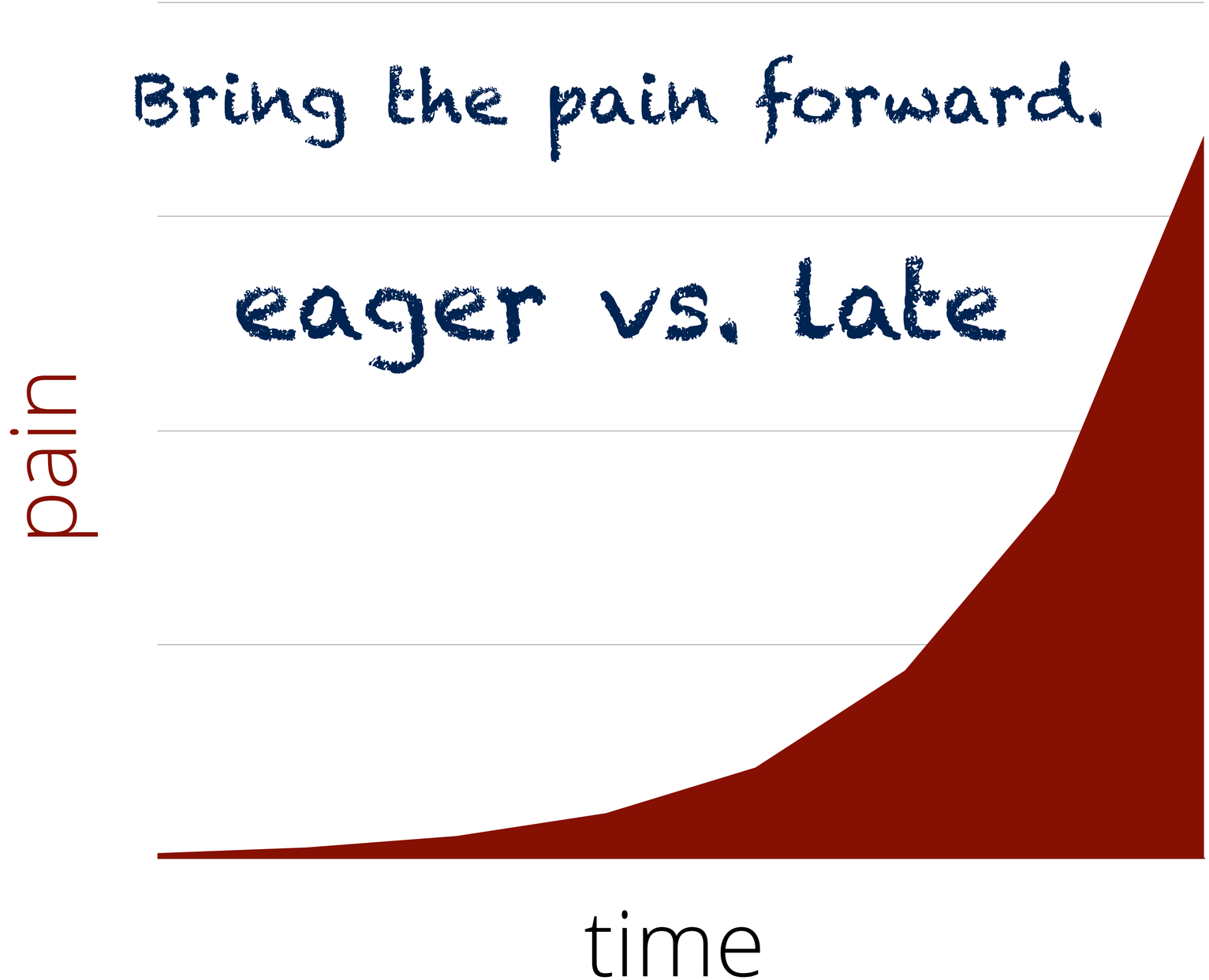
Continuous Integration

Integration early and often.

Everyone checks into **`trunk`** at least once a day.

Bring the pain forward.

eager vs. late



Continuous Integration

Fast, automated feedback on the correctness of your application every time there is a change to code

Integration

Integration early and often.

Everyone checks into **`trunk`** at least once a day.

Continuous Deployment

Deploy as the final stage of continuous integration.

Integration

Integration early and often.

Everyone checks into **`trunk`** at least once a day.

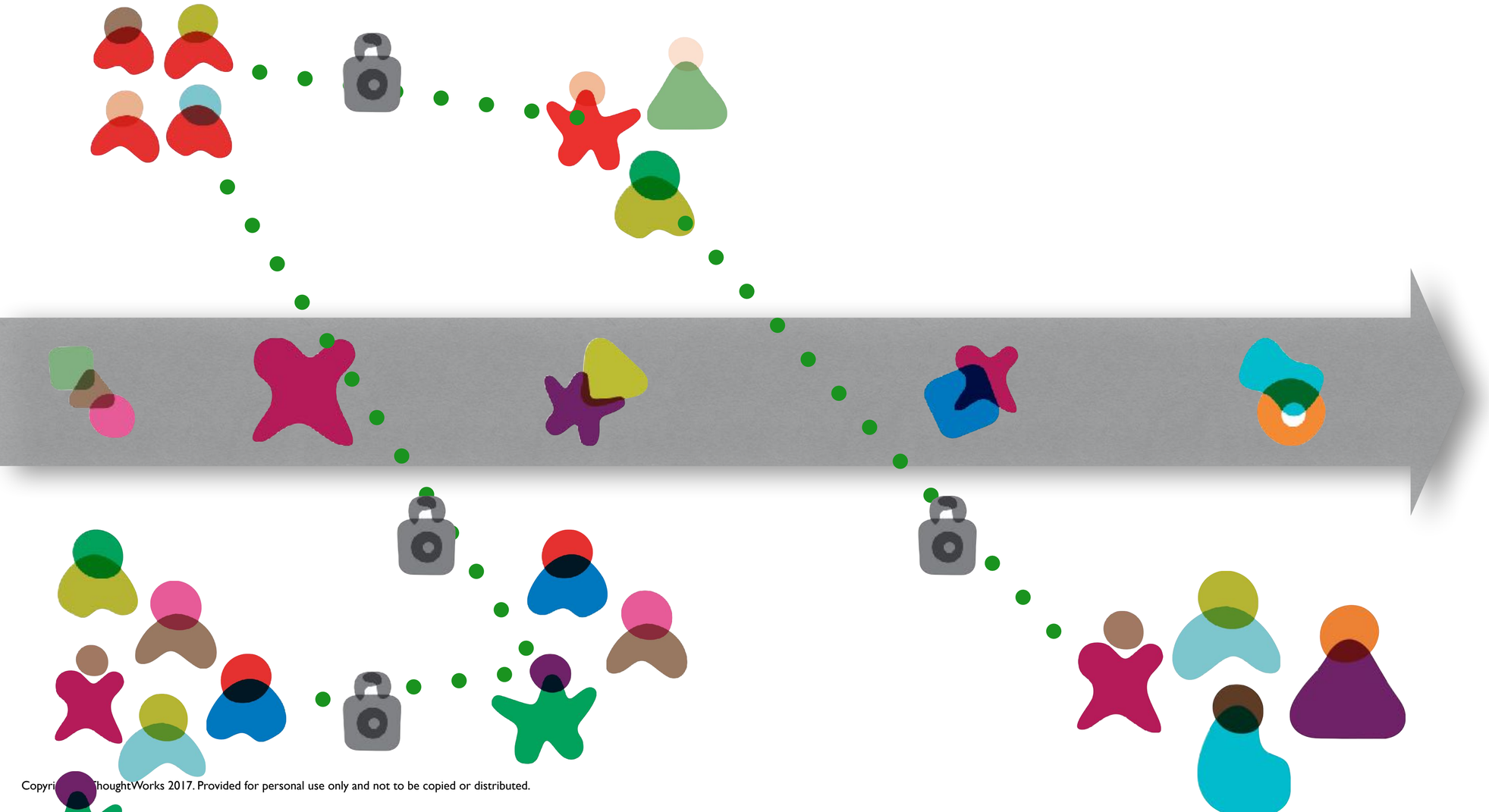
Deployment

Deploy as the final stage of continuous integration.

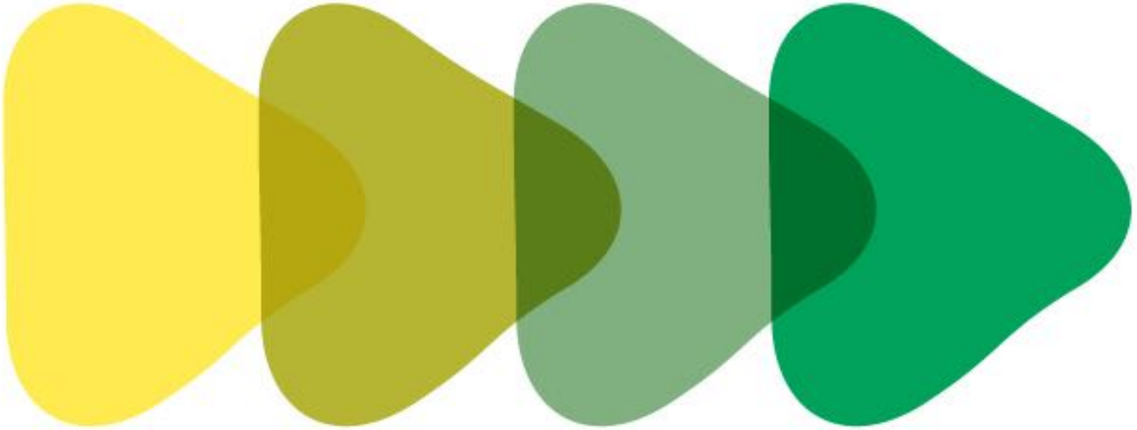
Continuous Delivery

Software is always in a deployable state.

So what's your plan?



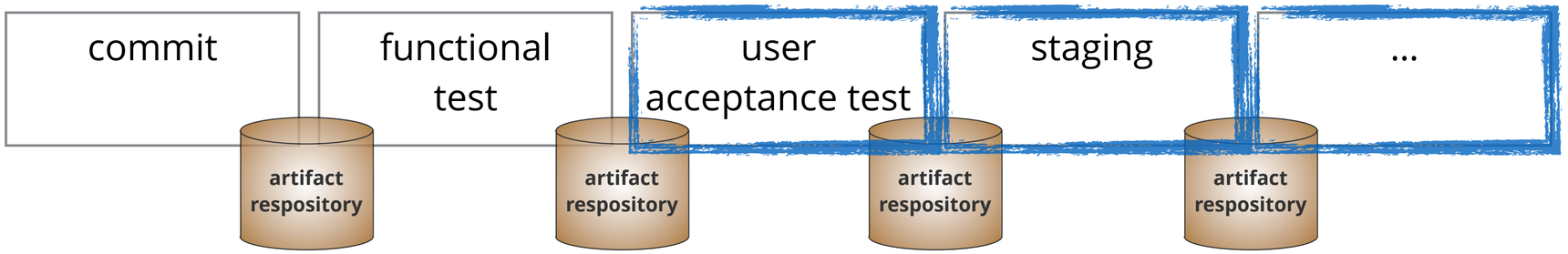
Deployment Pipelines



Deployment Pipeline

Fast, automated feedback on the **production readiness** of your application every time there is a change — to **code**, **infrastructure** or **configuration**

Pipeline Construction



Pipeline stages = feedback opportunities

commit Stage

source code
commit tests
build scripts

Run against each check-in

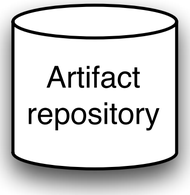
Starts building a release candidate

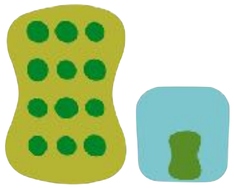
If it fails, fix it immediately

UAT Stage

acceptance tests
deployment scripts
configuration data

test reports
metadata



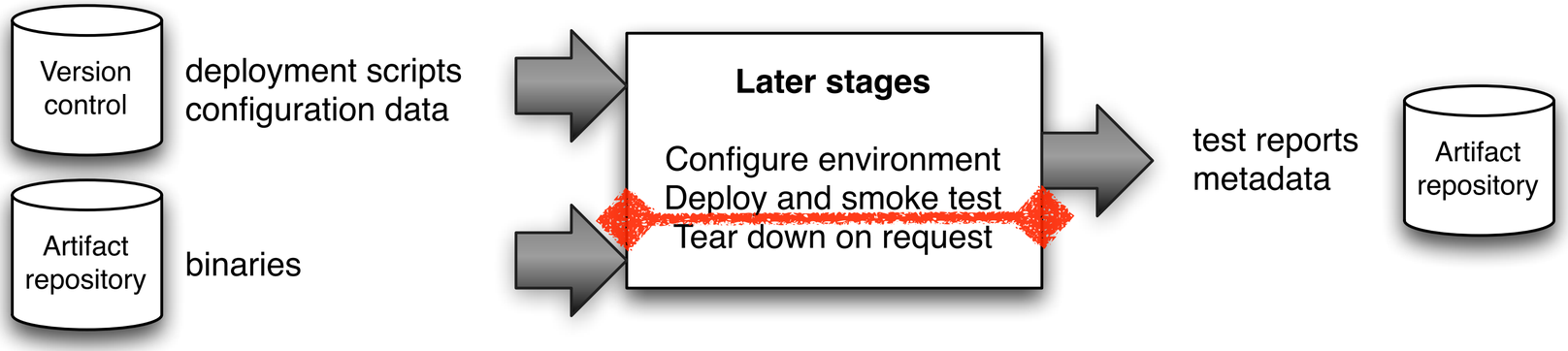


End-to-end tests in production-like environment

Triggered when upstream stage passes

First DevOps-centric build

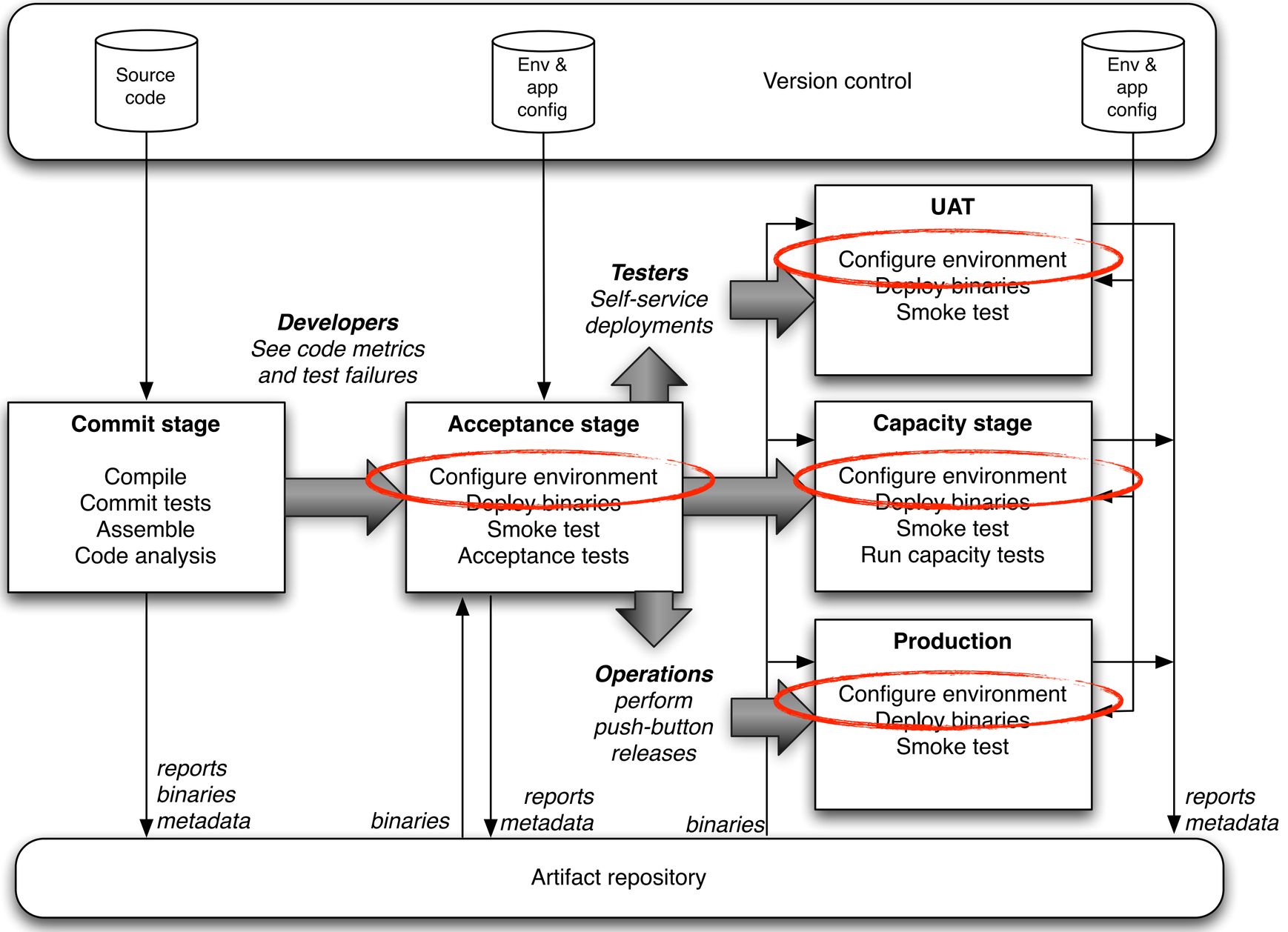
Manual **Stage**



UAT, staging, integration, production, ...

Push versus Pull model

Deployments self-serviced through push-button process


Source code
Env & app config
Version control
Env & app config
UAT
Configure environment
Deploy binaries
Smoke test
Testers
Self-service deployments
Developers
See code metrics and test failures
Commit stage
Compile
Commit tests
Assemble
Code analysis
Acceptance stage
Configure environment
Deploy binaries
Smoke test
Acceptance tests
Capacity stage
Configure environment
Deploy binaries
Smoke test
Run capacity tests
Production
Configure environment
Deploy binaries
Smoke test
Operations
perform push-button releases
reports
binaries
metadata
binaries
reports
metadata
binaries
reports
metadata
Artifact repository

Machinery

continuous integration ++

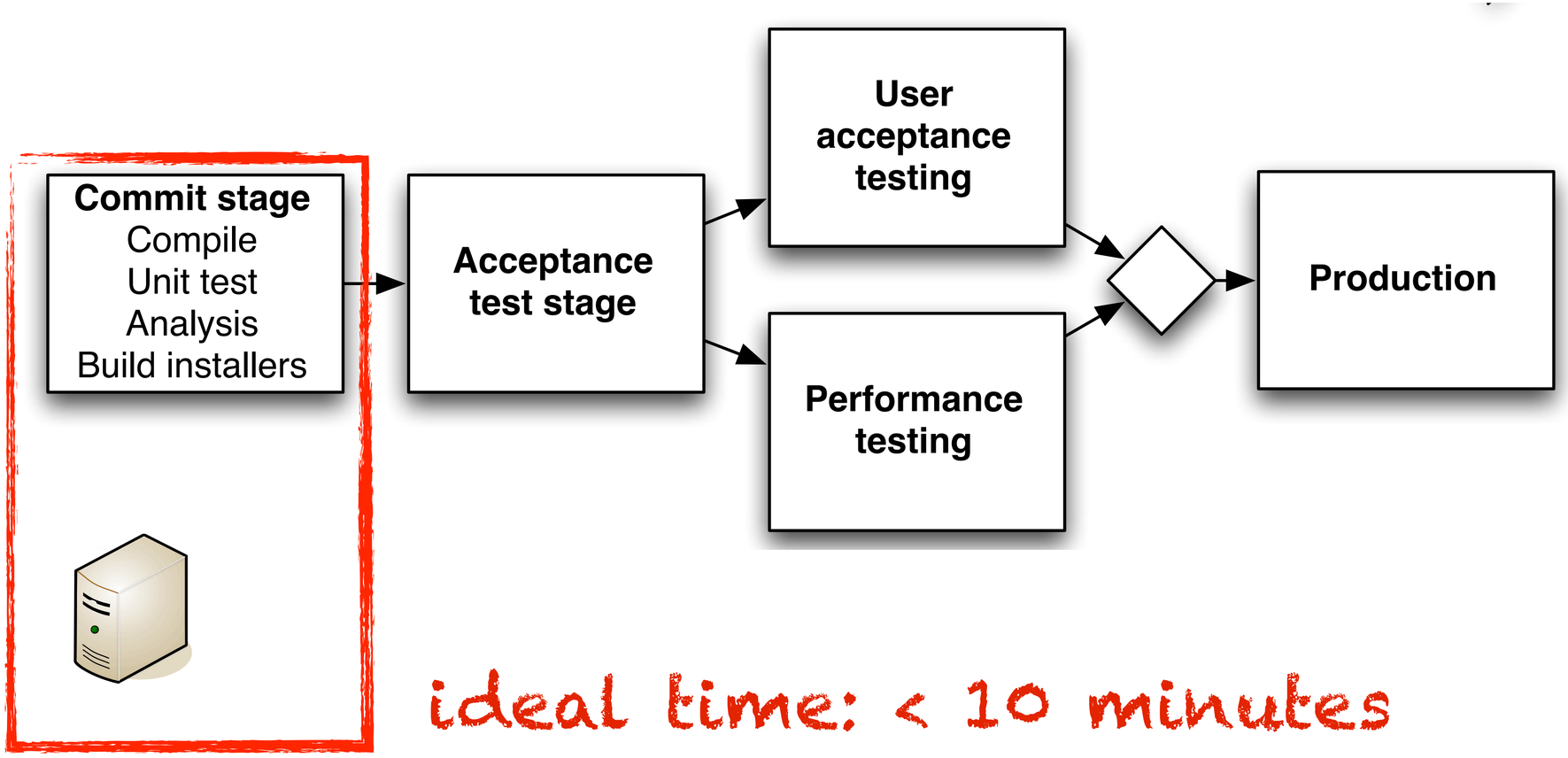
Jenkins

go
Continuous Delivery

www.thoughtworks.com/products/go-continuous-delivery

Pipeline Anti-patterns

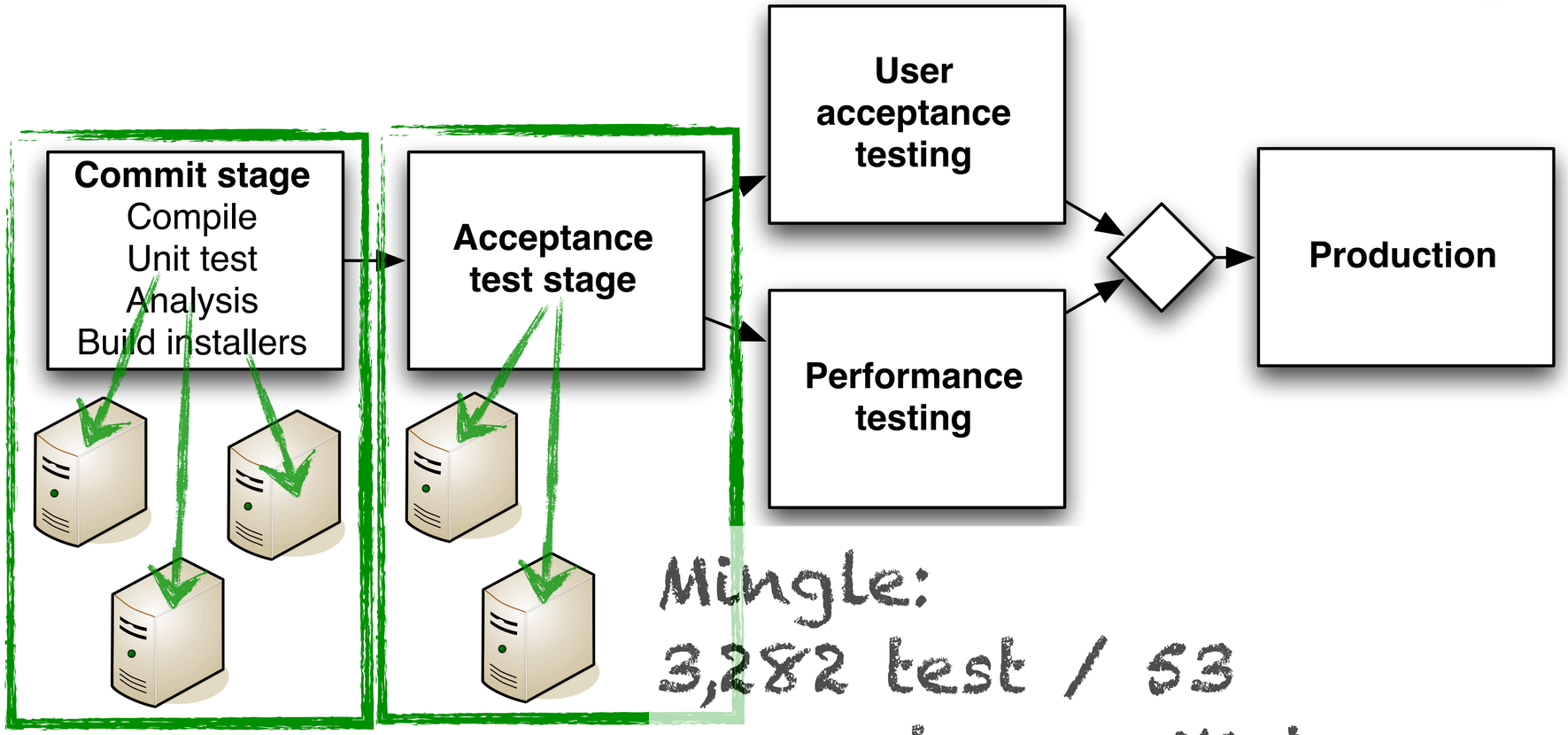
insufficient parallelization



ideal time: < 10 minutes

Pipeline Anti-patterns

insufficient parallelization



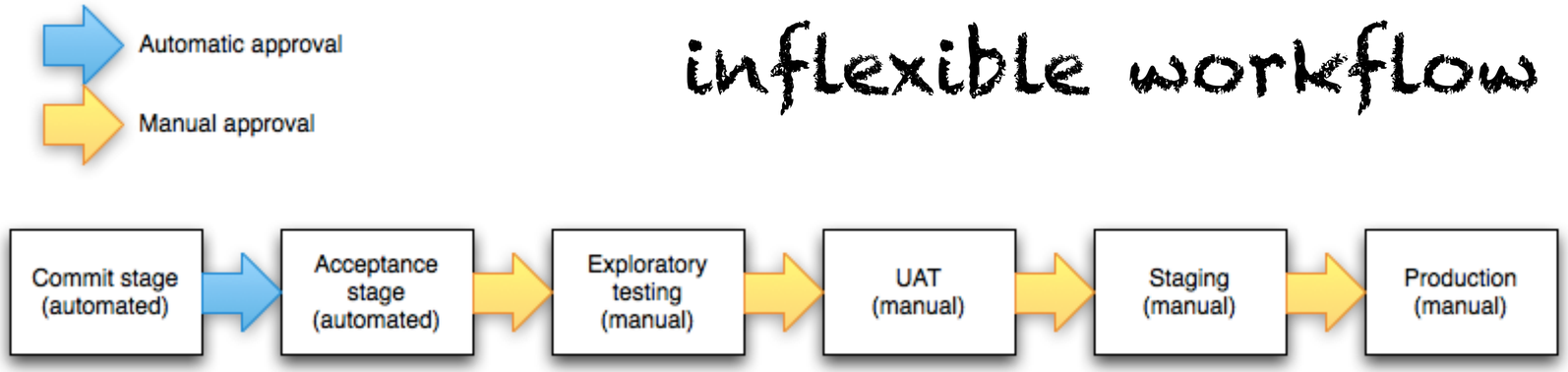
Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*

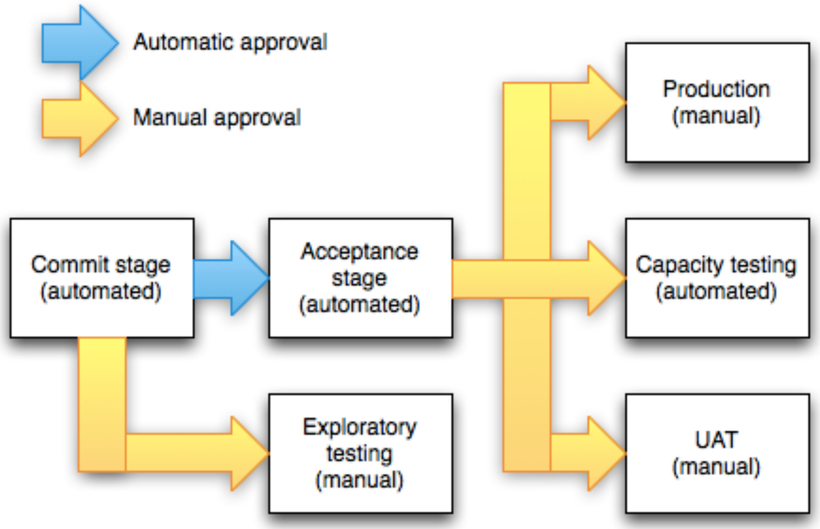
parallelize each stage as much as you can

create more stages if necessary to optimize feedback

Pipeline Anti-patterns



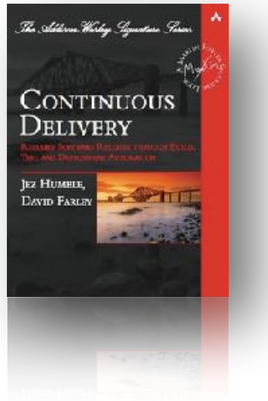
inflexible workflow



pipeline fans out as soon as it makes sense to do so

Principles

automate almost everything



keep everything you need to build, deploy, test, & release in version control

- requirements documents
- test scripts
- automated test cases
- network configuration scripts
- technical documentation
- database creation, upgrade, downgrade, and initialization scripts
- application stack configuration scripts
- libraries
- deployment scripts
- tool chains

Infrastructure Consistency

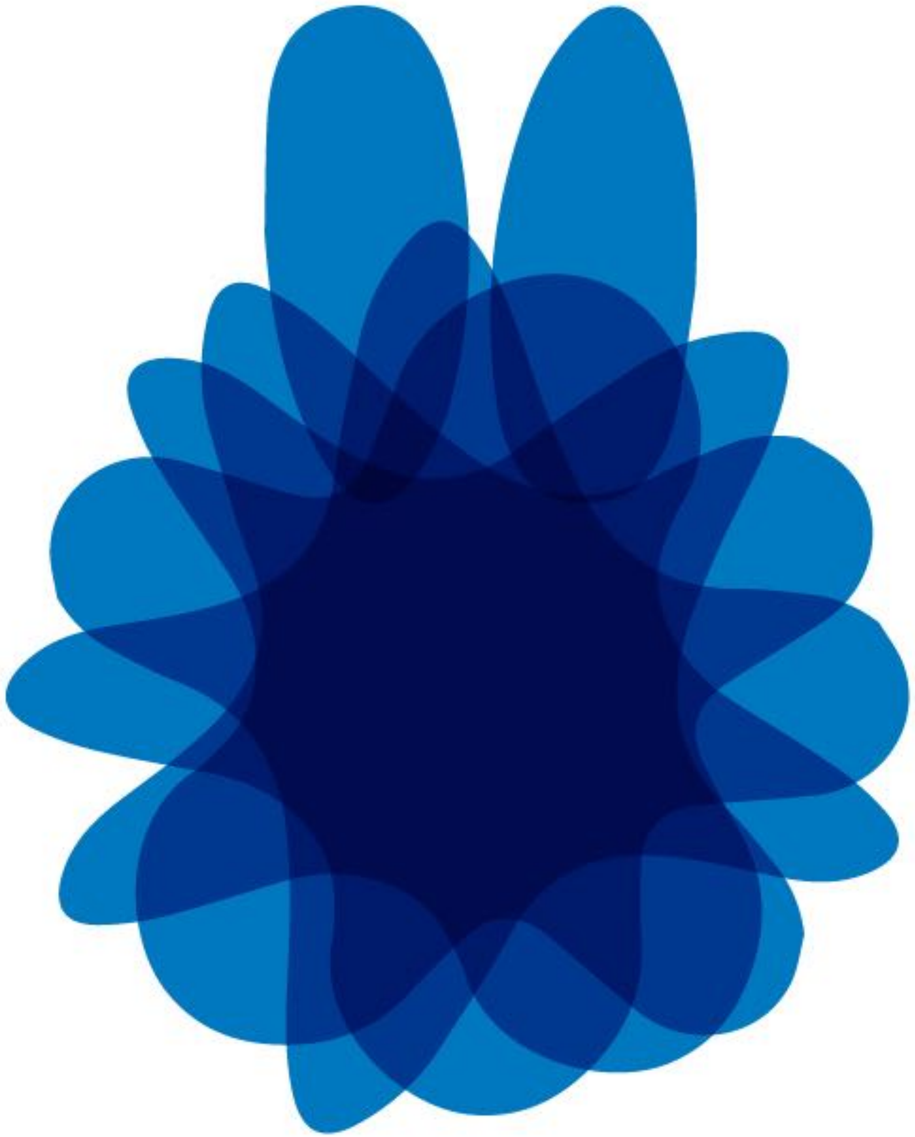
Chef

puppet labs

ANSIBLE

boxen.github.com

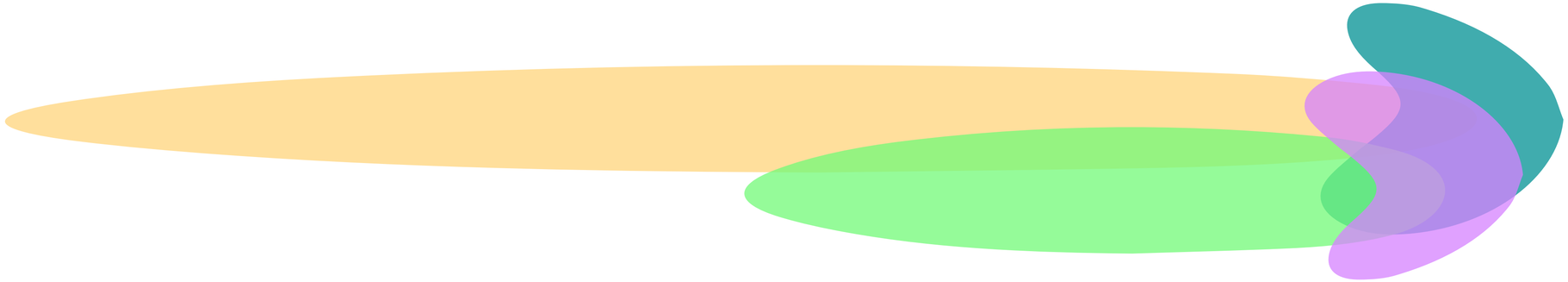
Identify & remove friction



Continuous Delivery Metrics

lead time

the time between the initiation and completion of a production process.



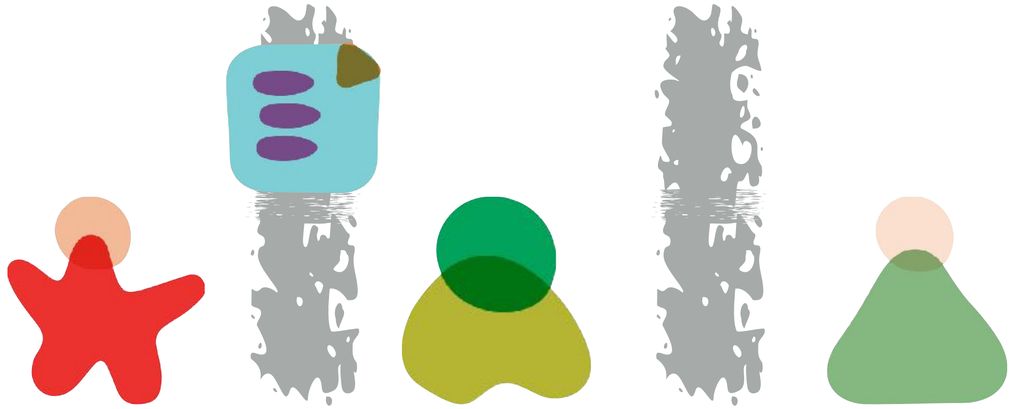
cycle time

the total elapsed time to move a unit of work from the beginning to the end of a physical process

Potential Hindrances

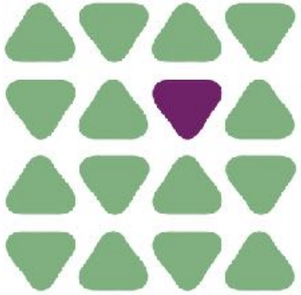
Lead time is too long

Last mile is too painful



Poor collaboration

Prerequisites

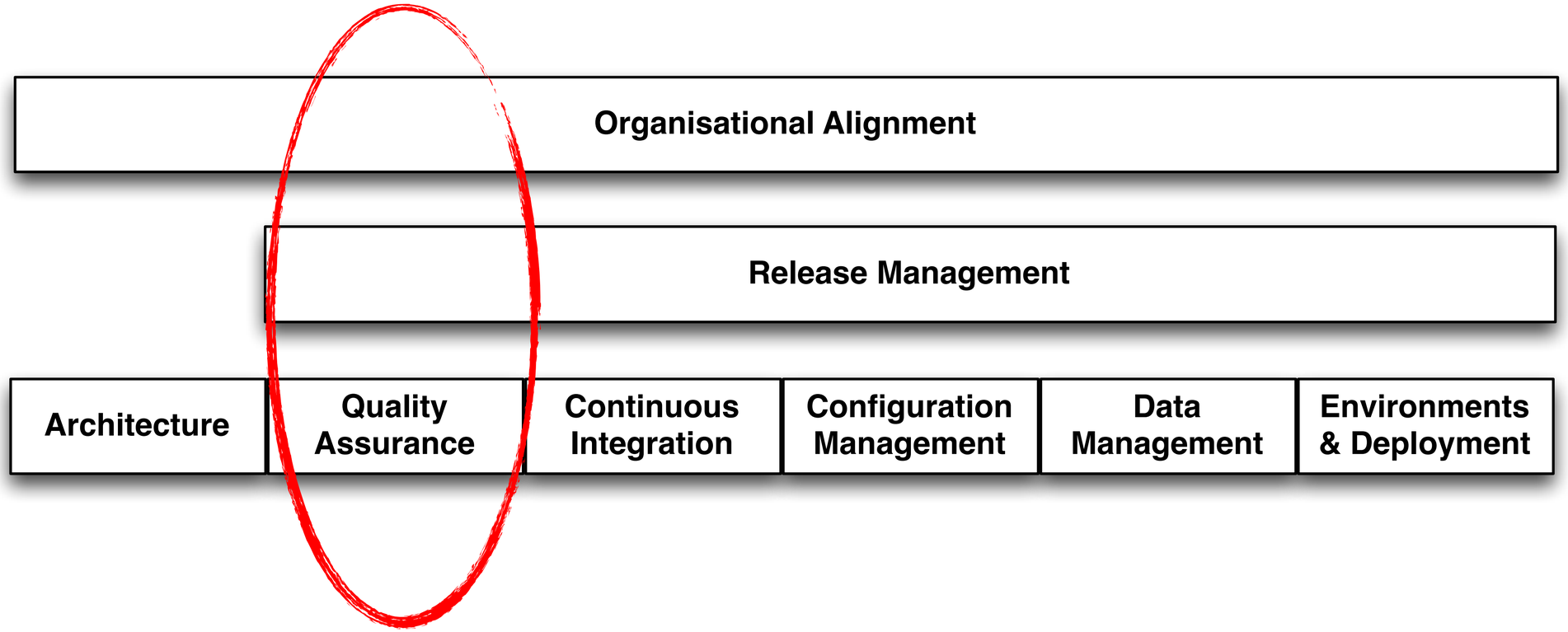


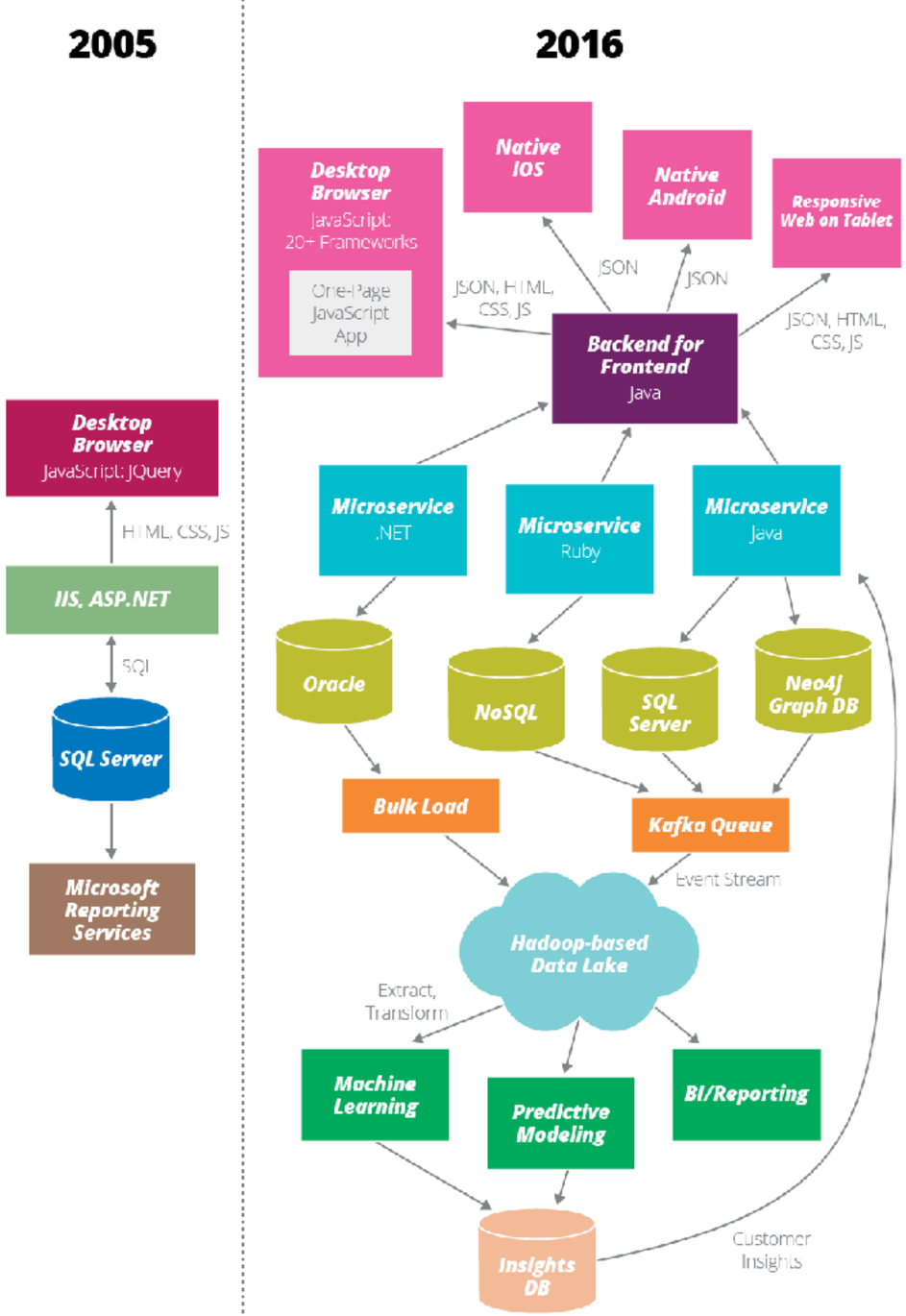
comprehensive
configuration management

continuous integration

excellent automated testing at all levels

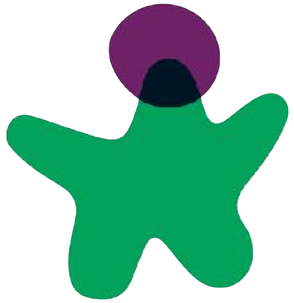
Continuous Delivery



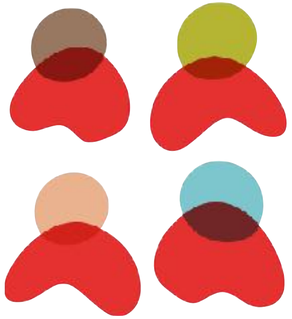


Modern software is *complex!*

https://www.thoughtworks.com/insights/blog/implications-tech-stack-complexity-executives



effort

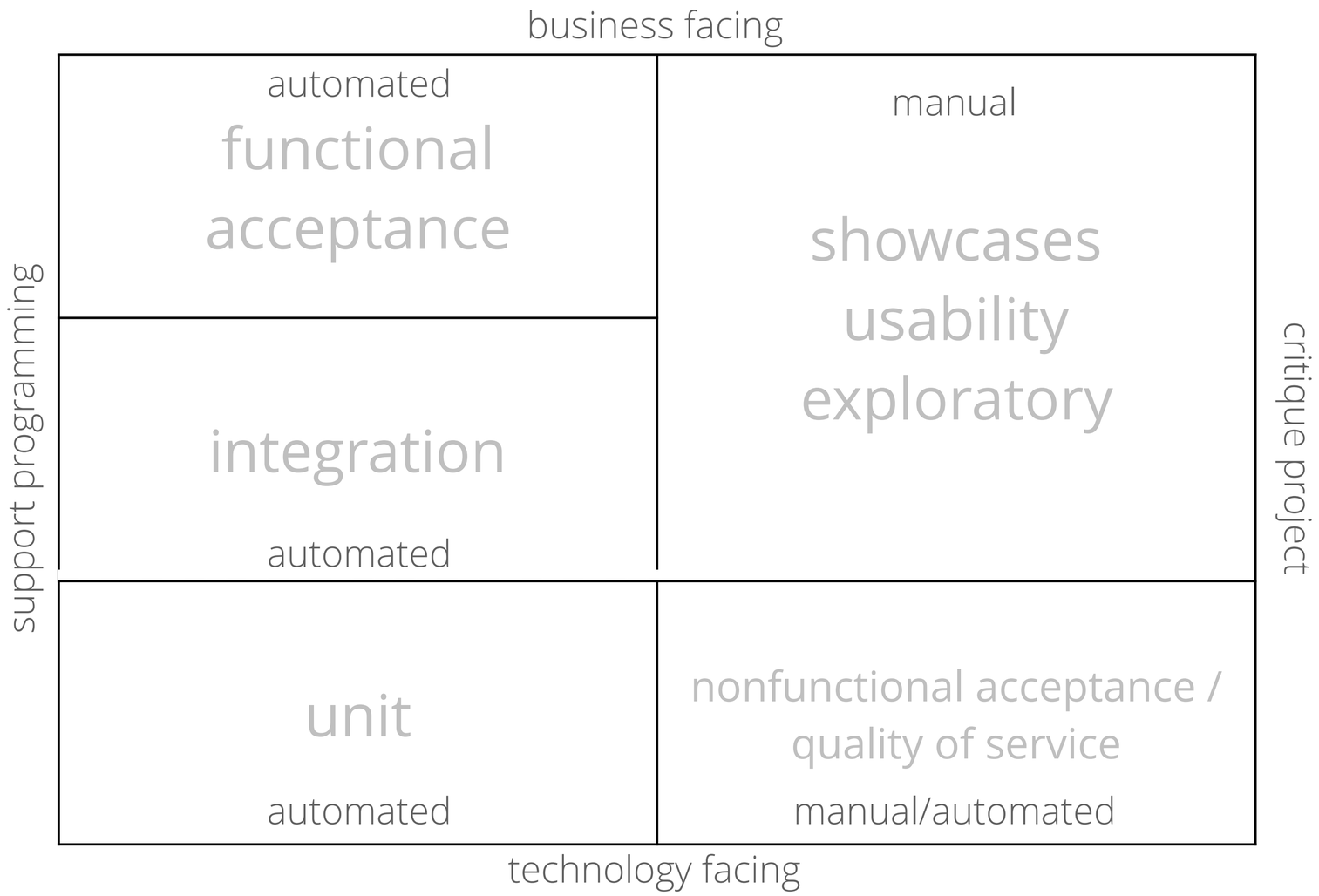


audience

feedback

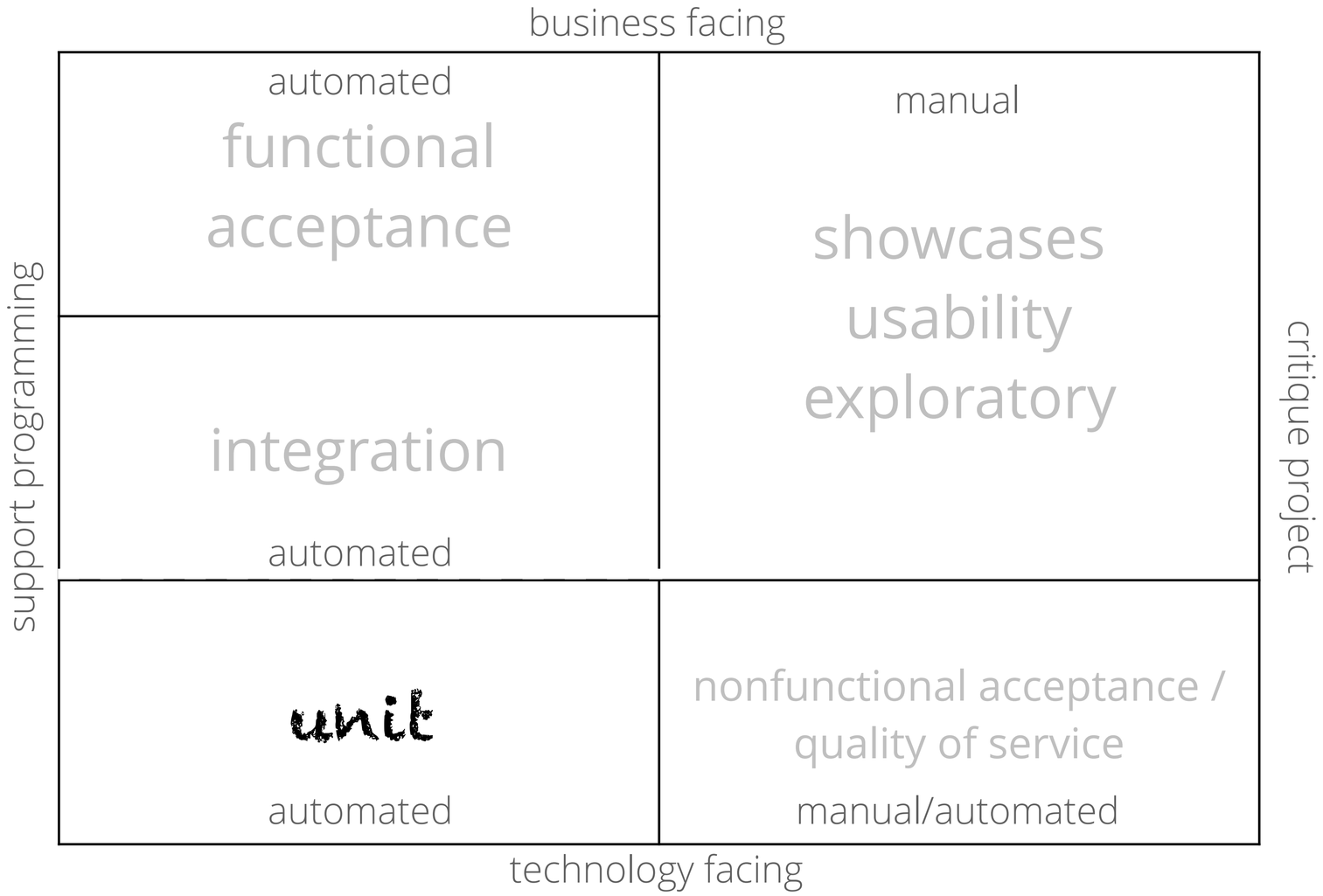
Testing

Testing Quadrants



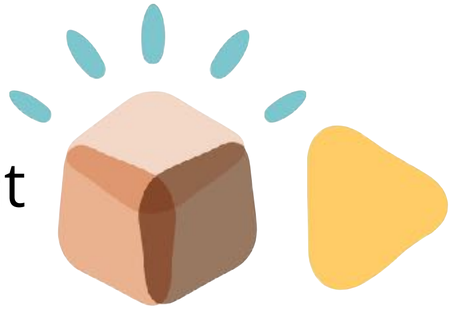
source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications

Testing Quadrants



source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications

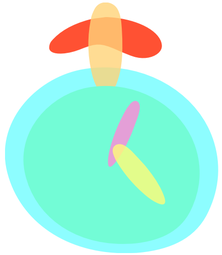
unit Testing



prefer test-driven to test-after development

prefer pragmatism over dogmatic metrics

optimize for the target audience

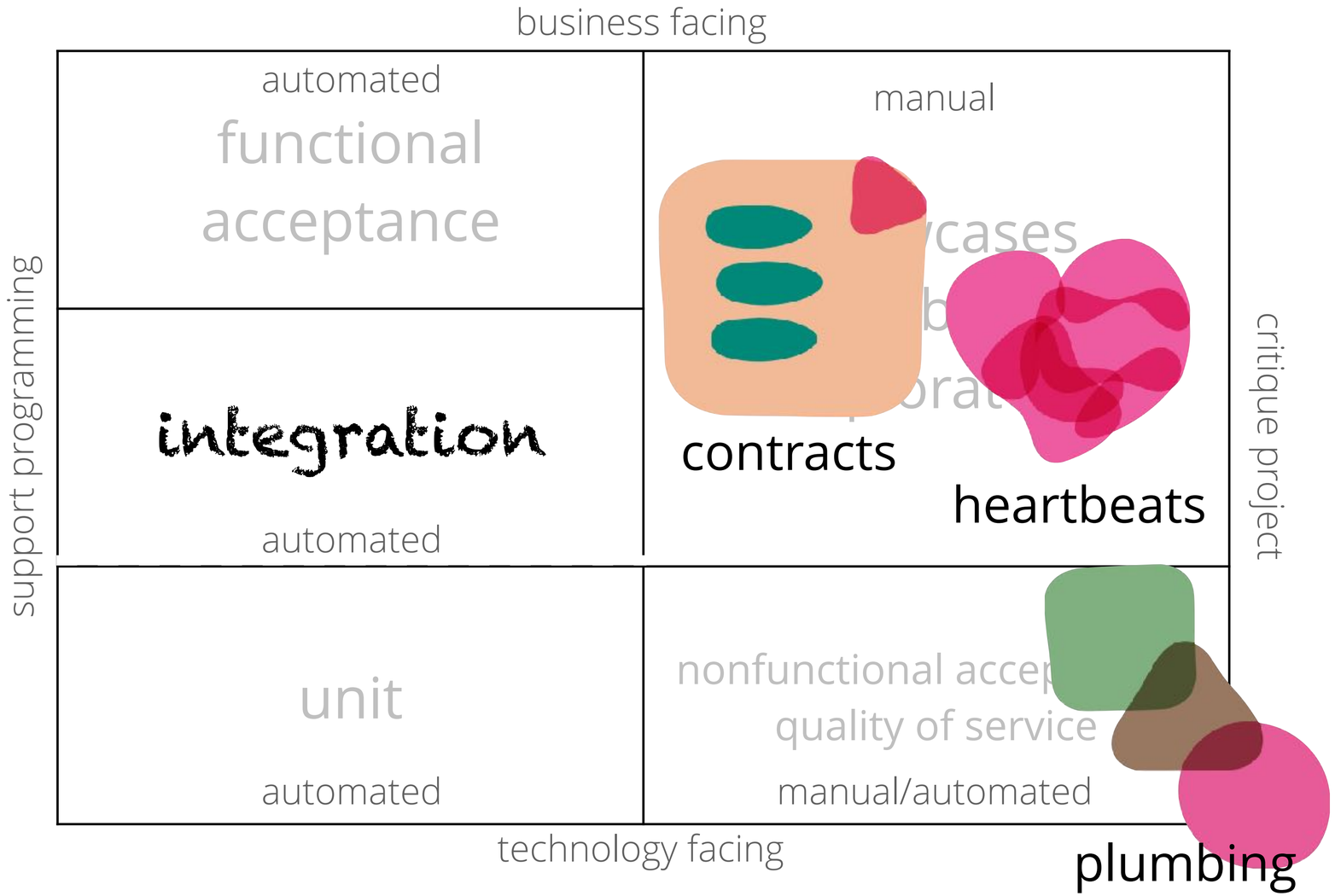


how much time?

Common Anti-pattern

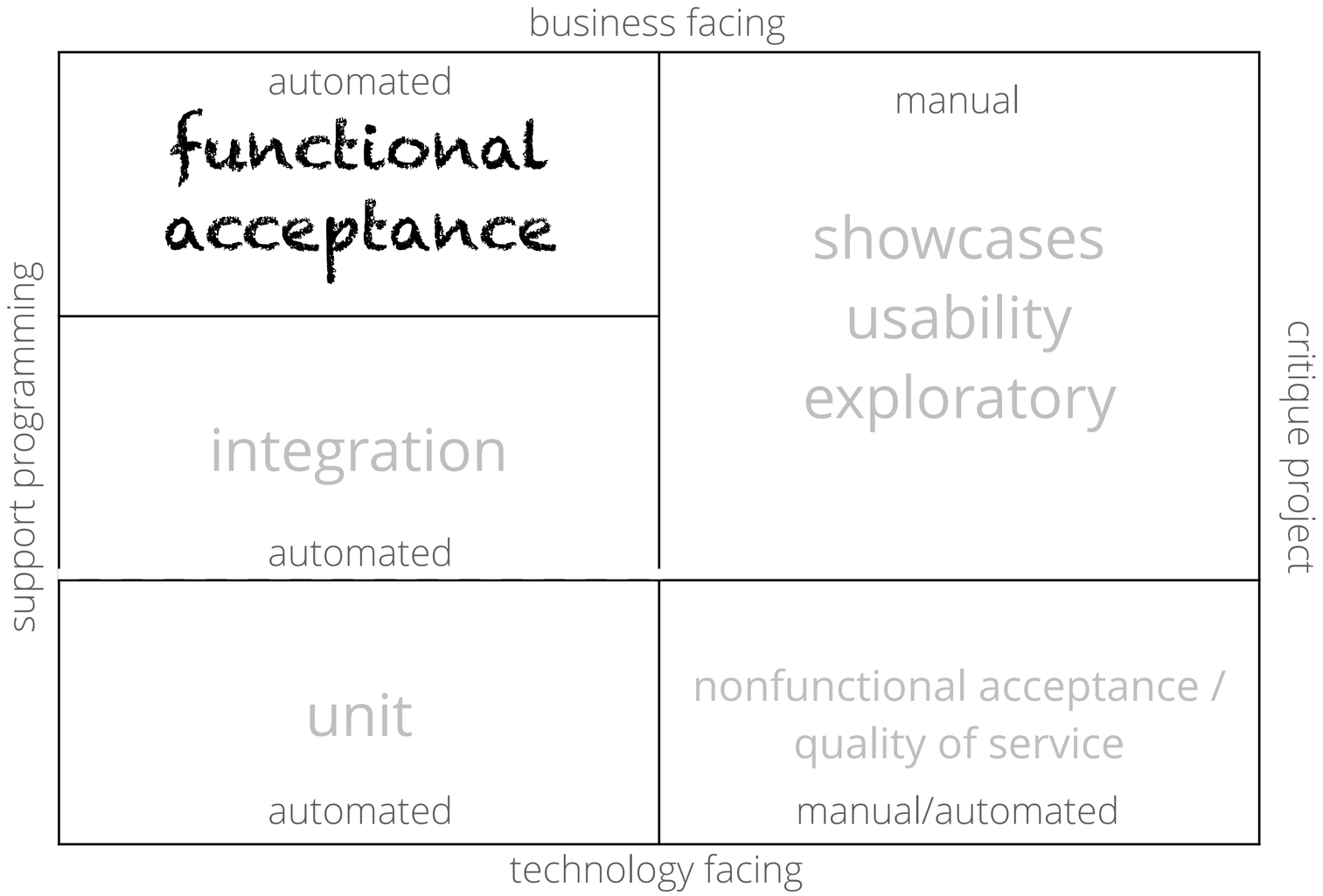
mixed unit/
functional tests

Testing Quadrants



source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications

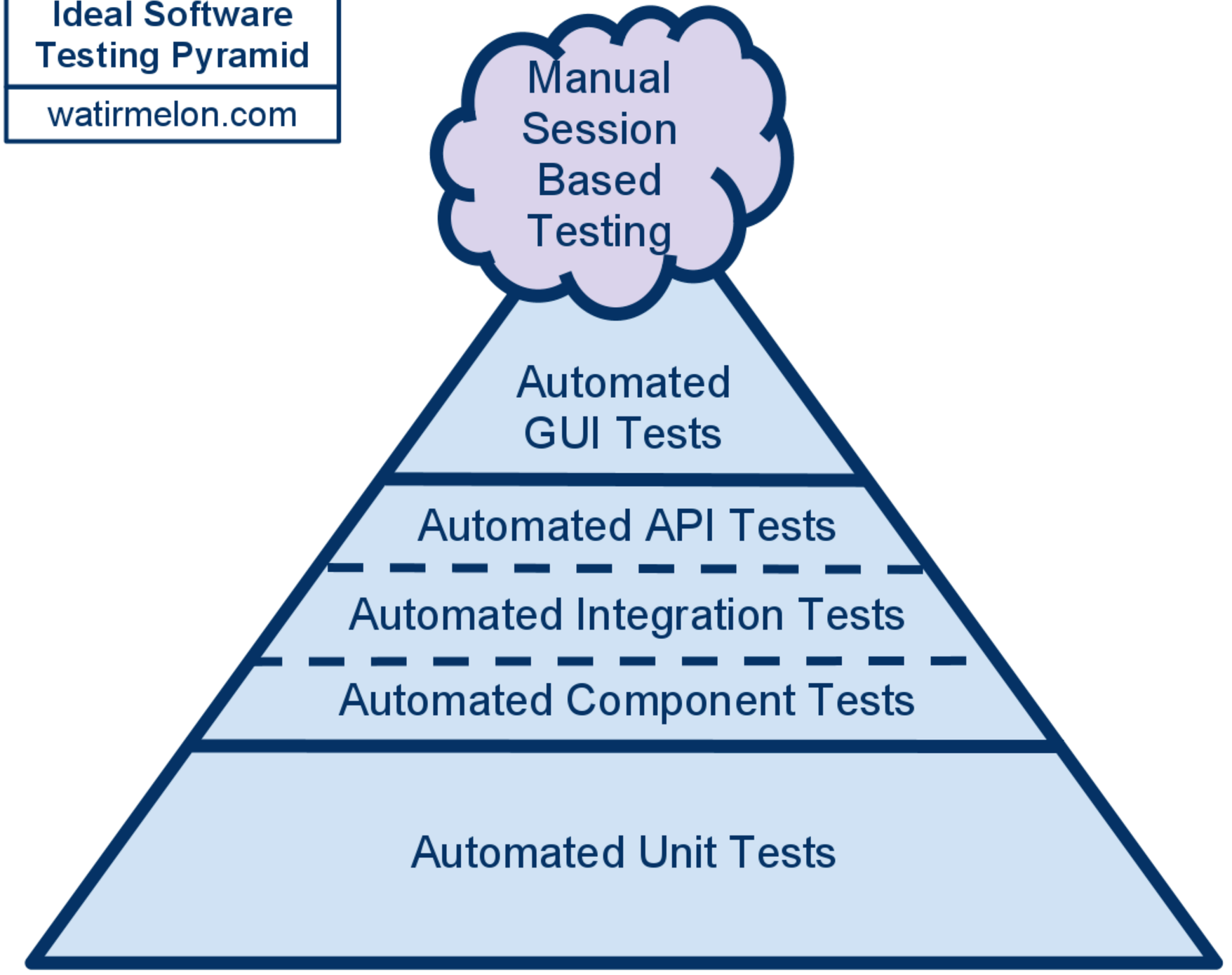
Testing Quadrants



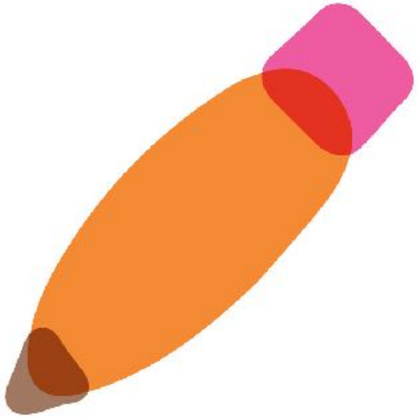
source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications

Ideal Software Testing Pyramid

watirmelon.com

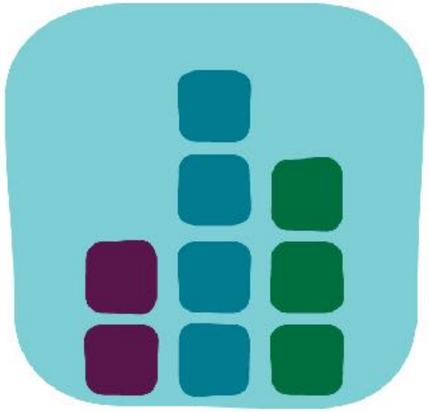


To Cuke or not to Cuke...



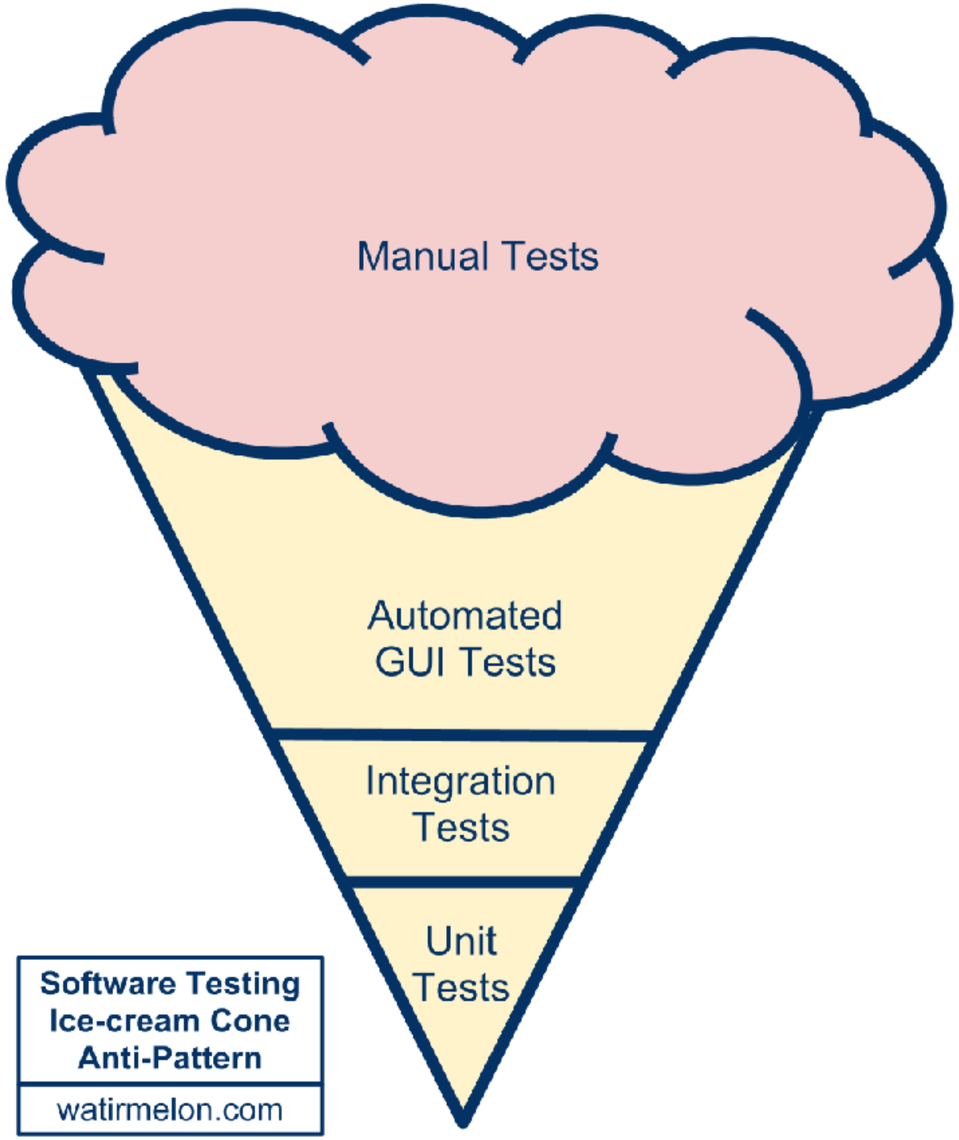
BDD ≠ 

use when it provides useful feedback to the target audience

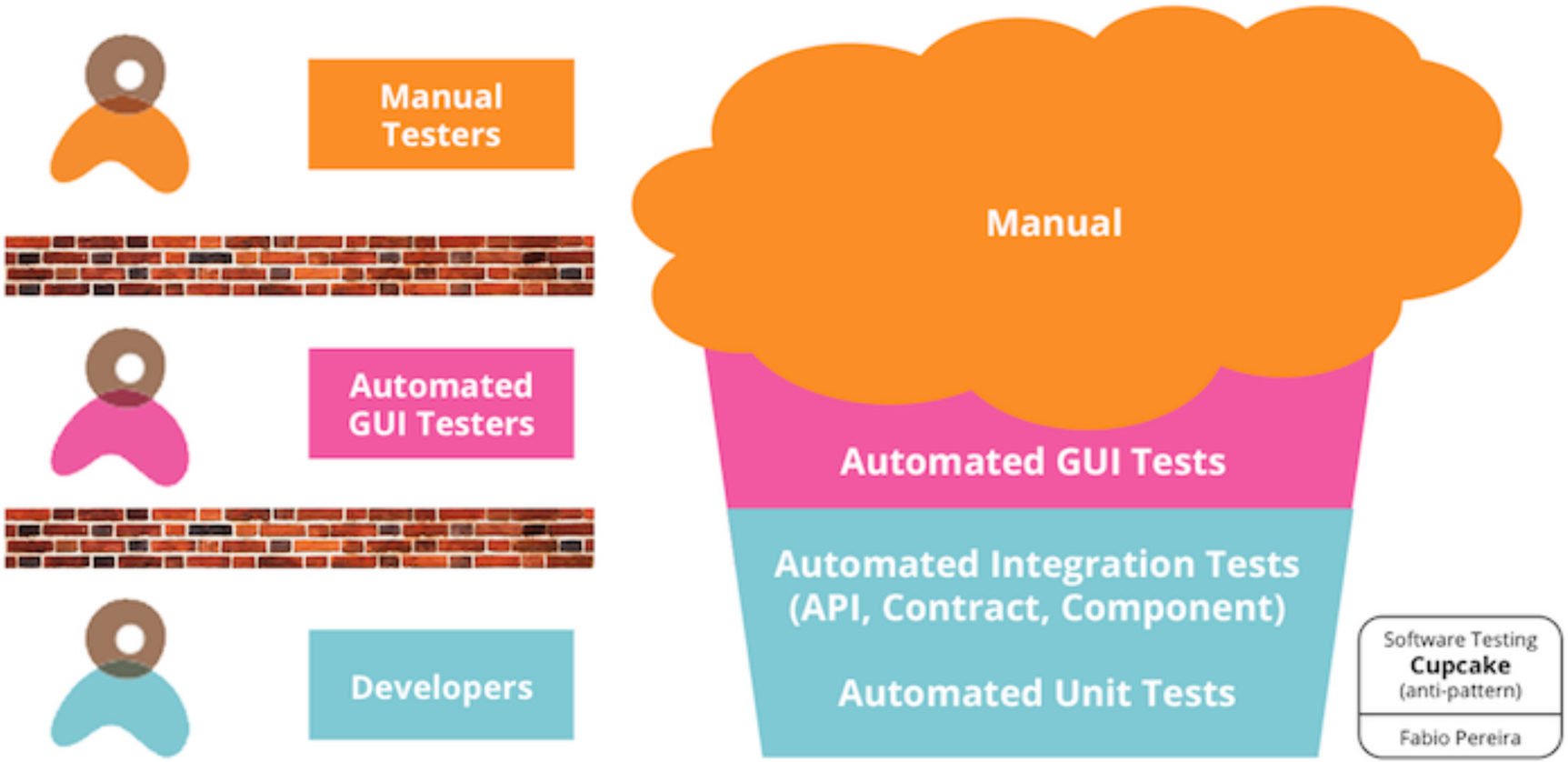


http://www.thoughtworks.com/insights/blog/3-misconceptions-about-bdd

Anti-pattern: Ice-cream Cone



Cupcake Anti-pattern



http://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern

Avoiding Cupcakes

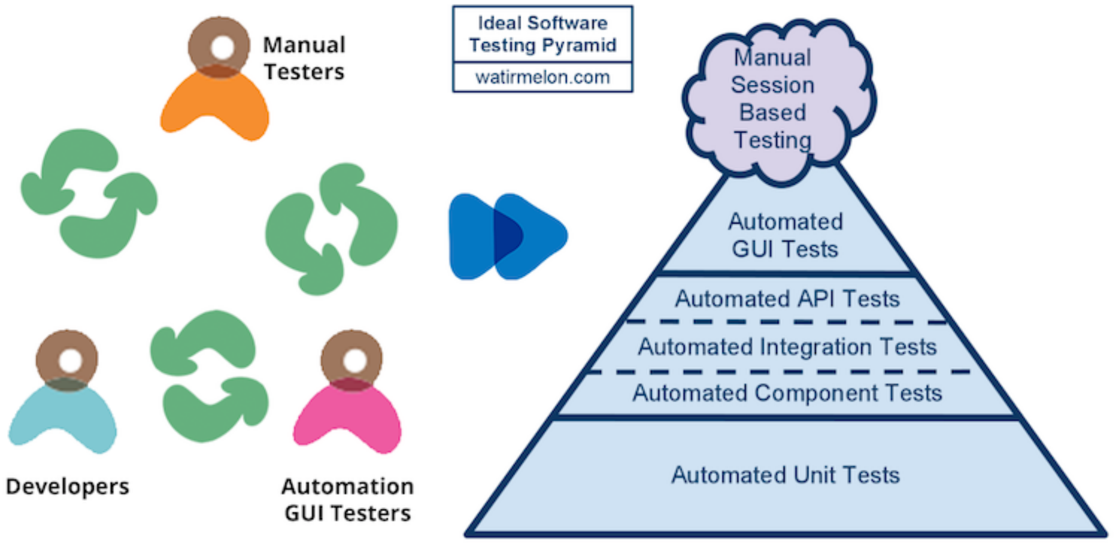
collaborate

- work in sync
- cross-role pair programming
- story kickoff

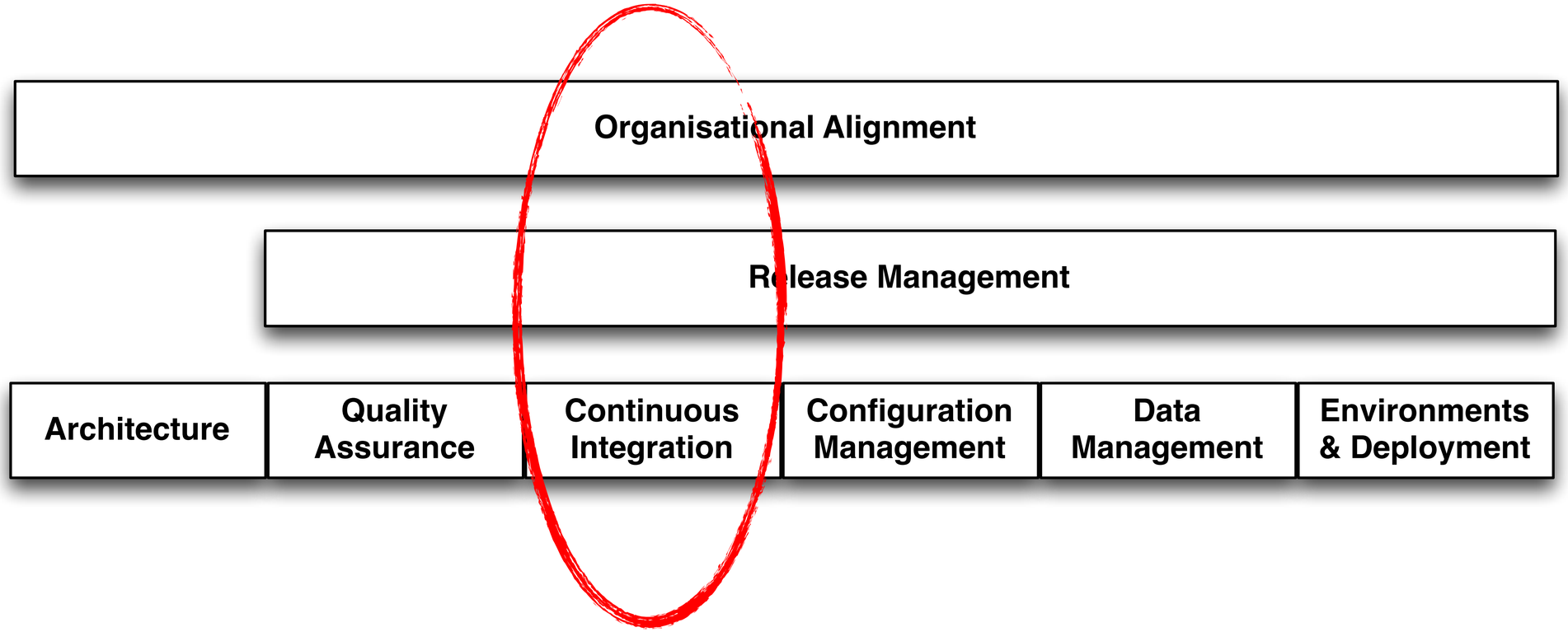
test at the lowest level

merge teams
when possible

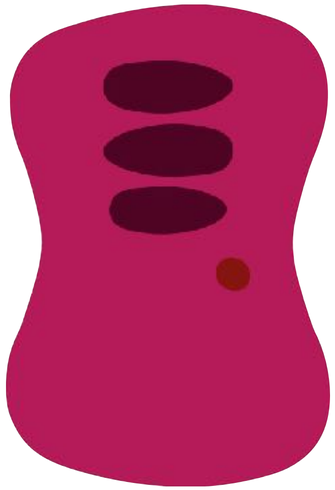
agree on goals
and metrics



Continuous Delivery

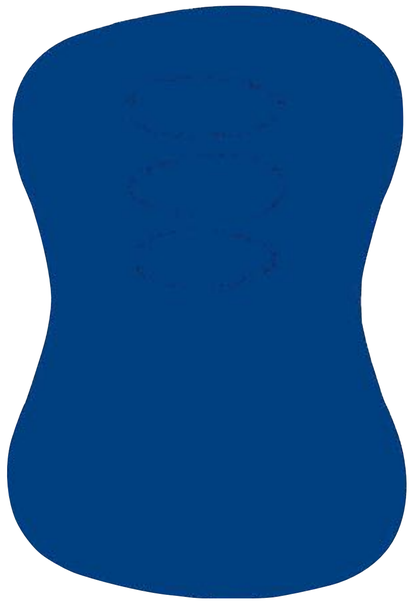


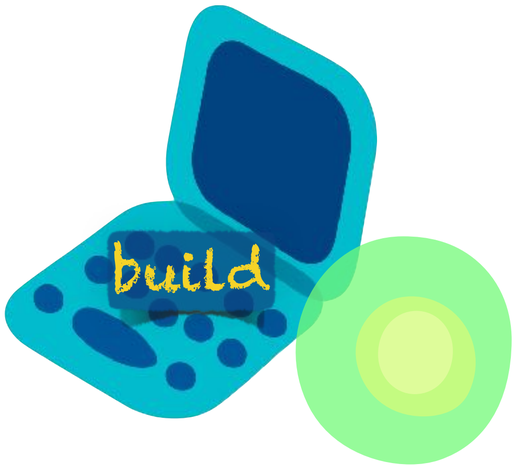
local
workstation

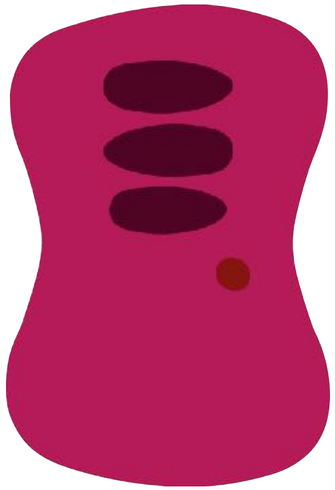


version
control

continuous integration
server

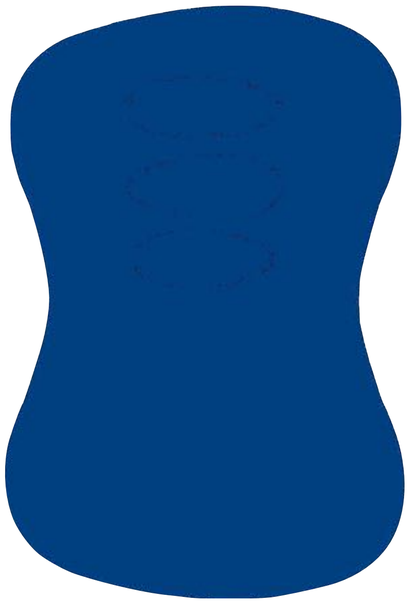


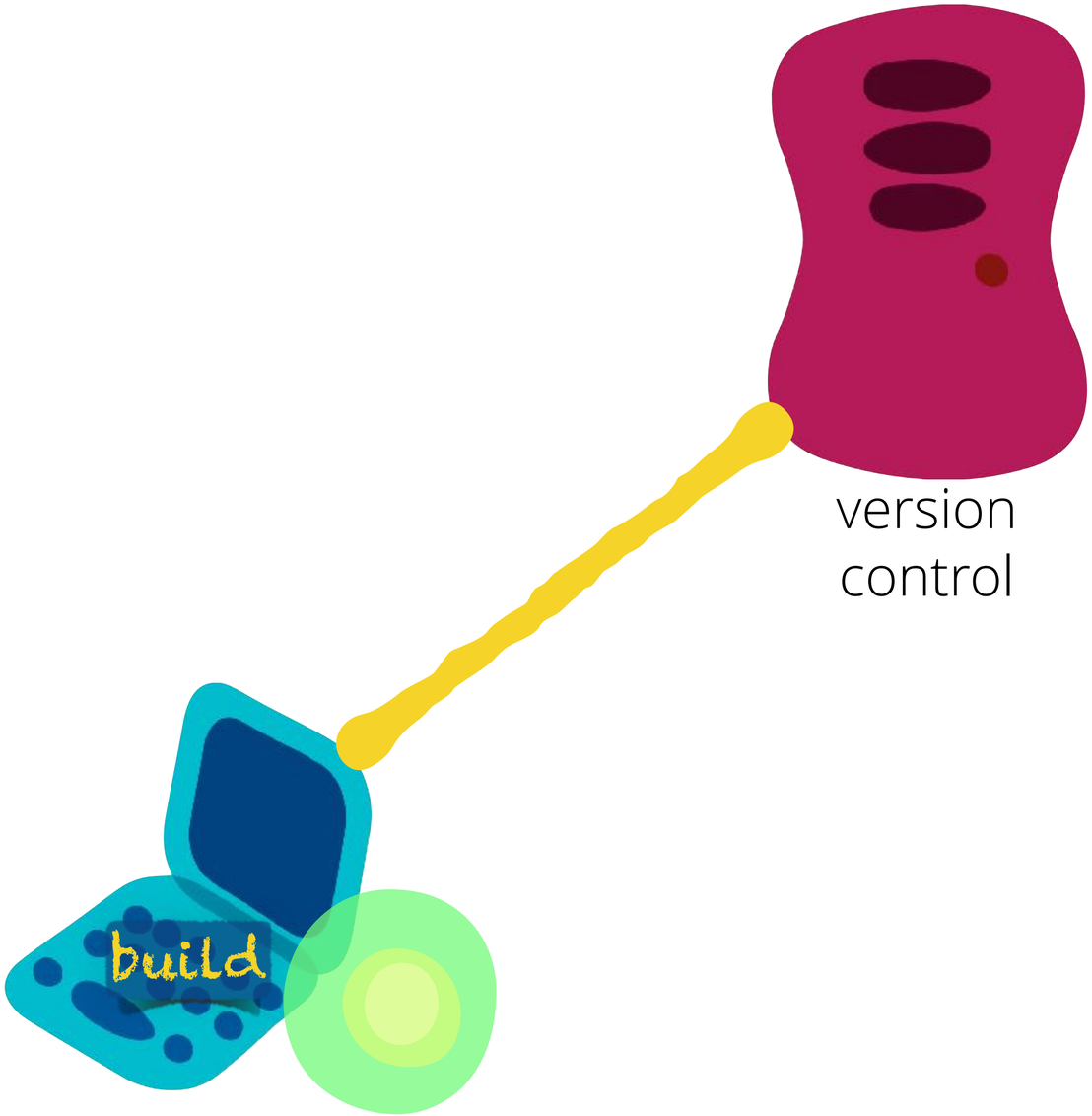




version control

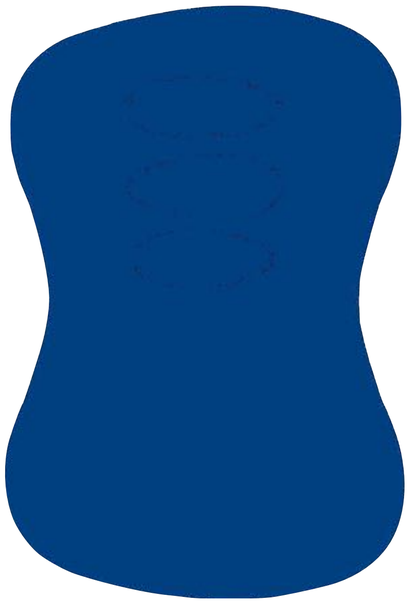
continuous integration server

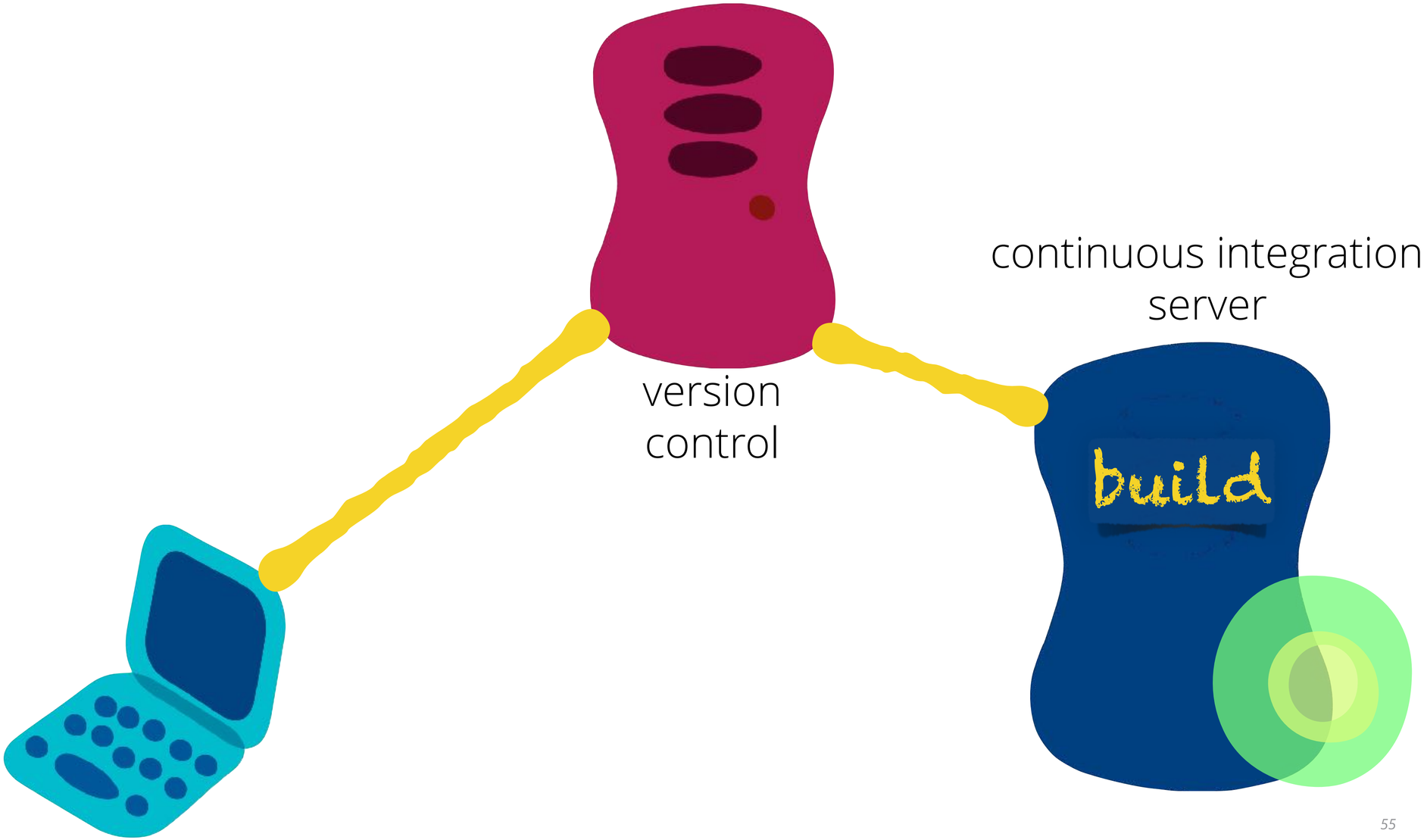




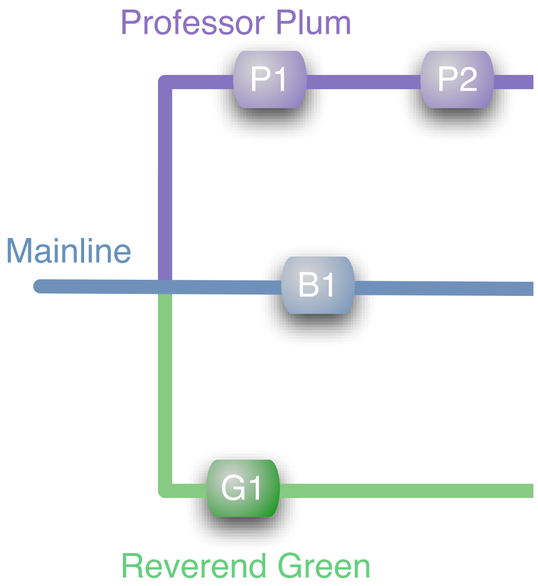
version control

continuous integration server

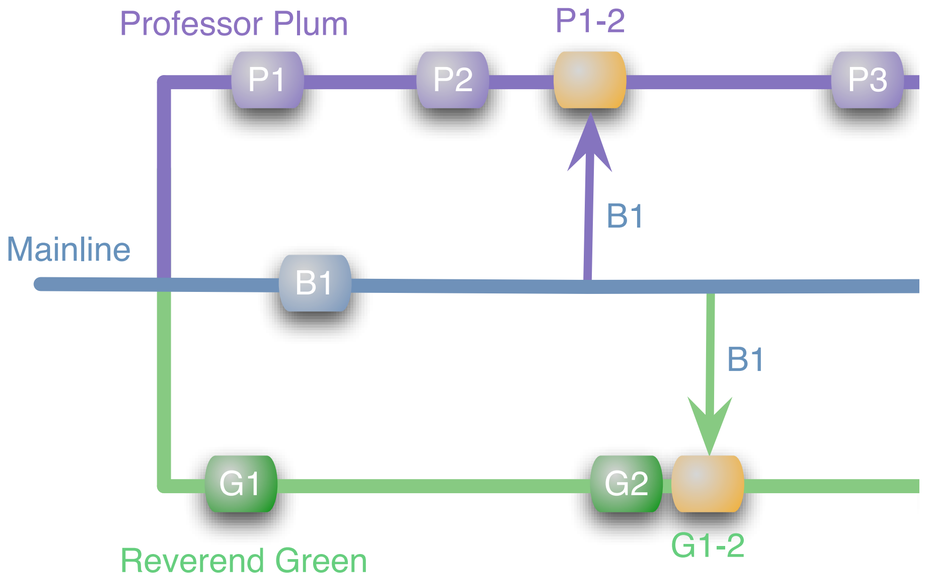



version
control
continuous integration
server
build

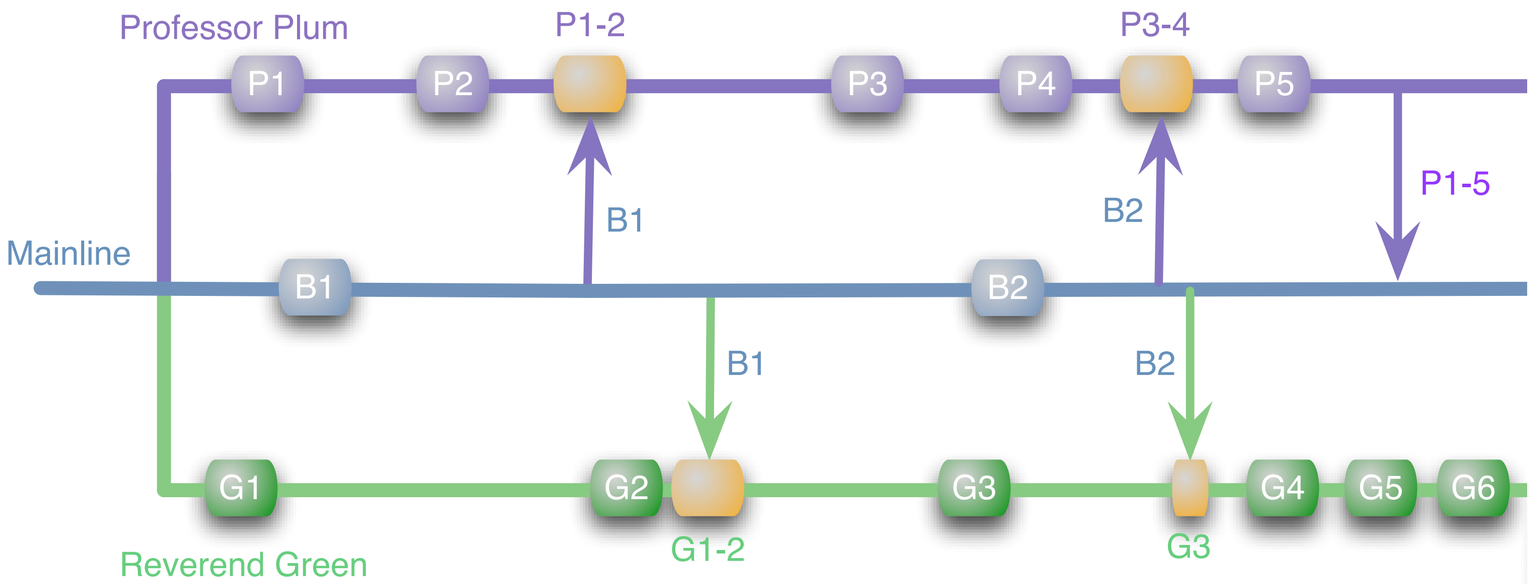
everyone commits to trunk at least once a day



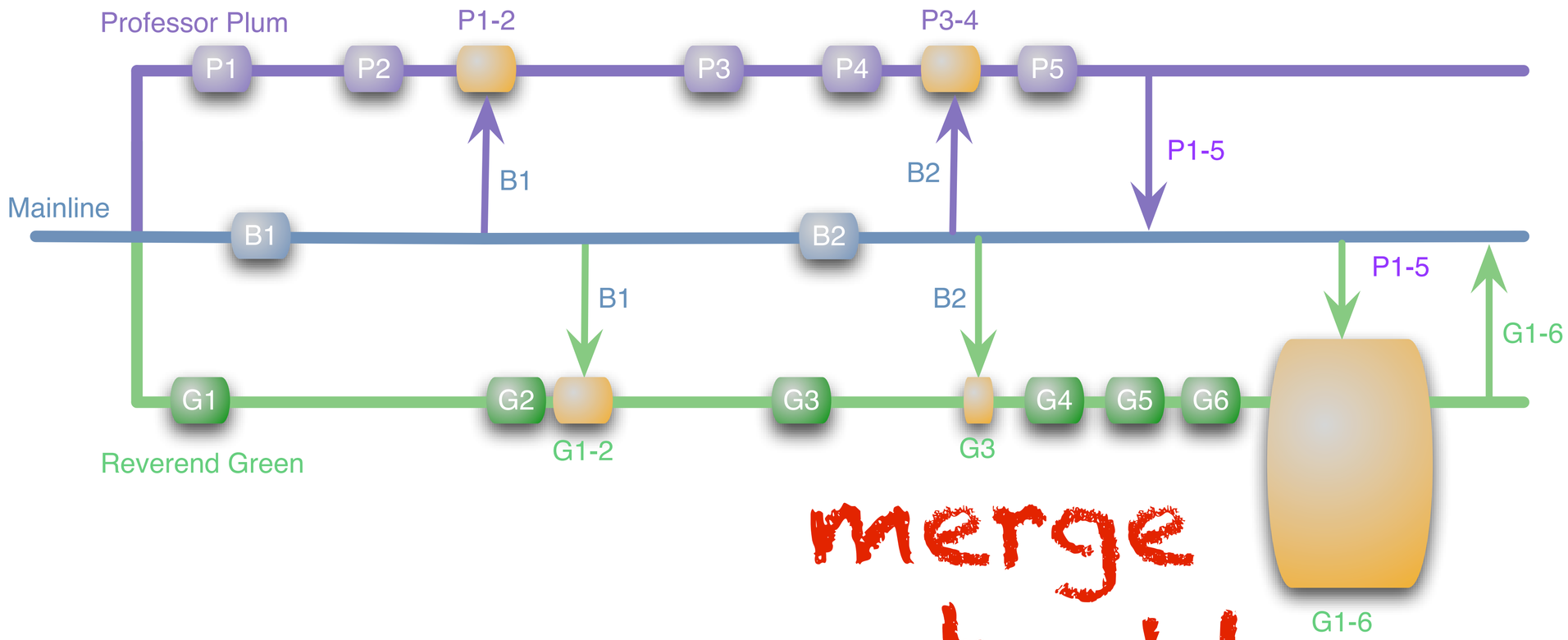
Feature Branching



Feature Branching



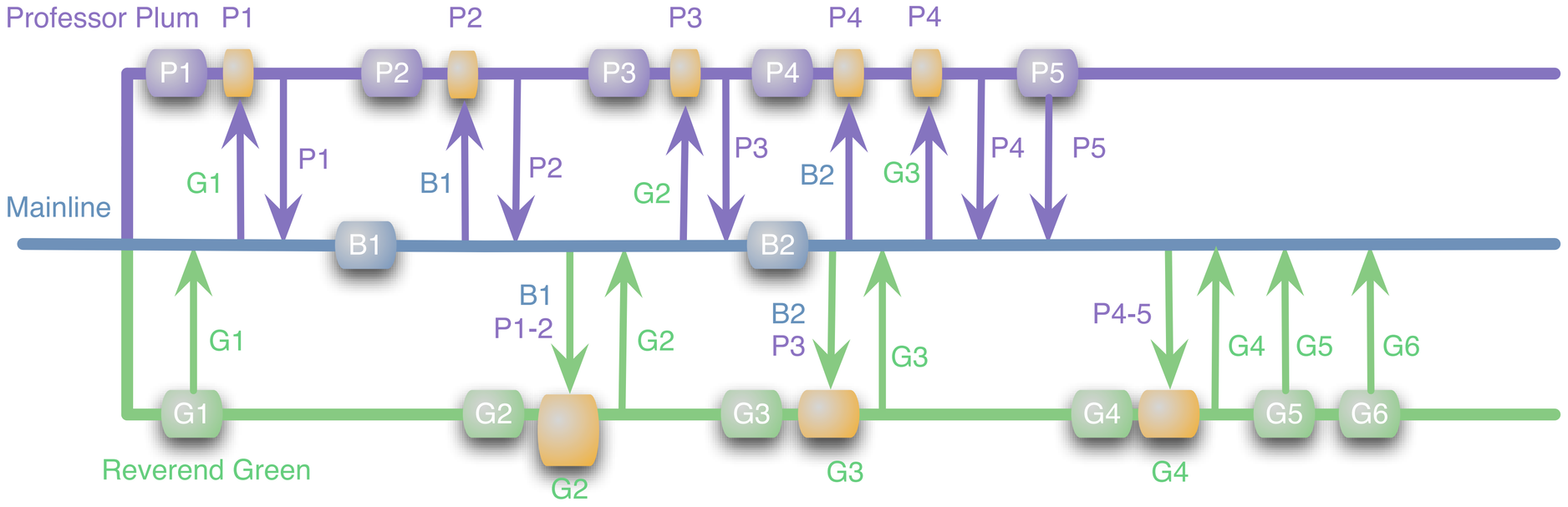
Feature Branching



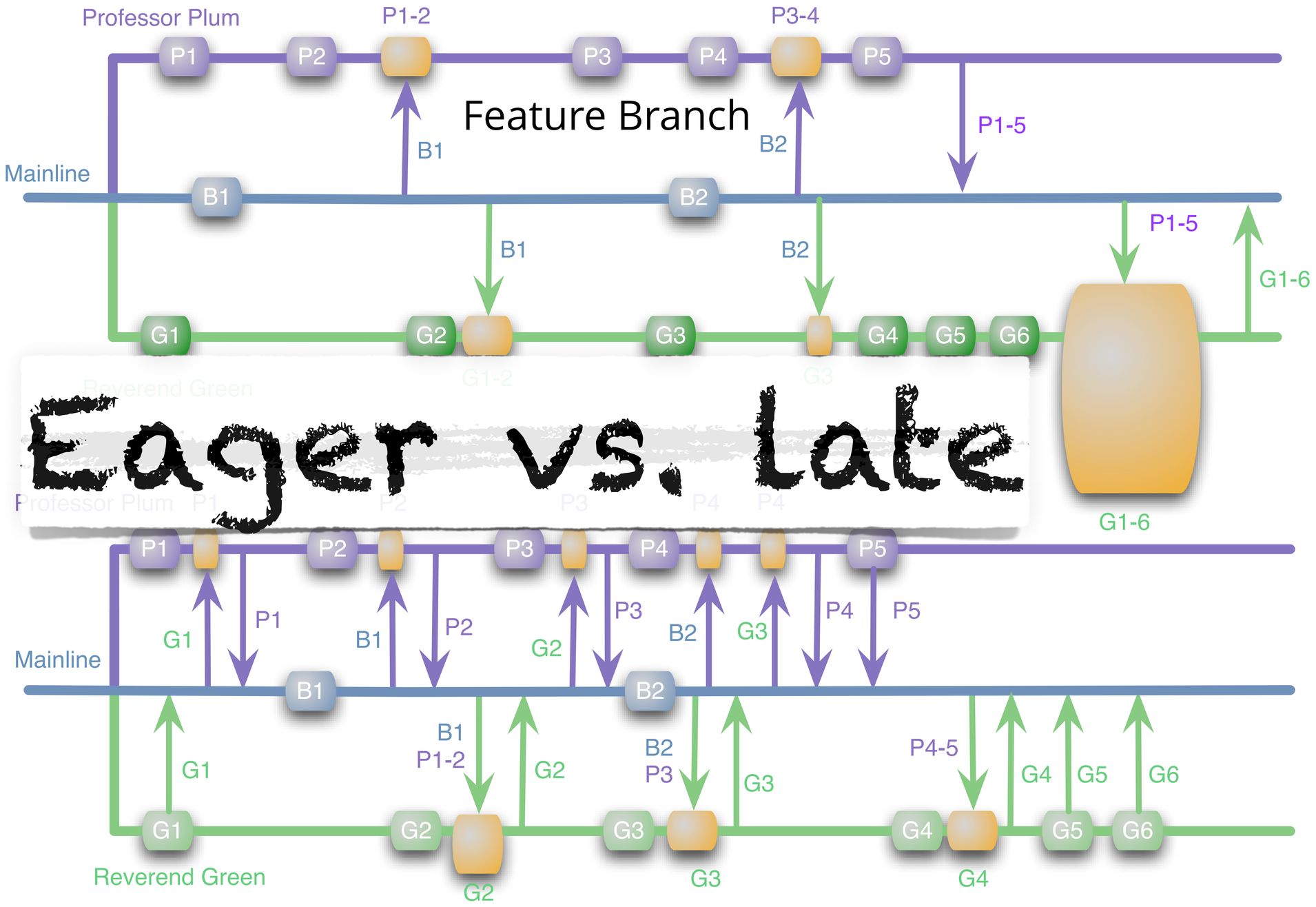
merge ambush!

Feature Branching

trunk-based development



Continuous Integration removes the pain...



Continuous Integration

feature toggles

Config File

```
[featureToggles]
wobblyFoobars: true
flightyForkHandles: false
```

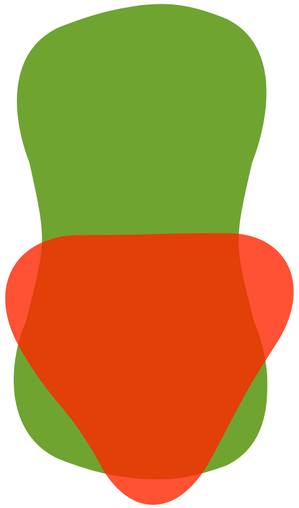
some.jsp

```
<toggle name=wobblyFoobars>
  ... various UI elements
</toggle>
```

other.java

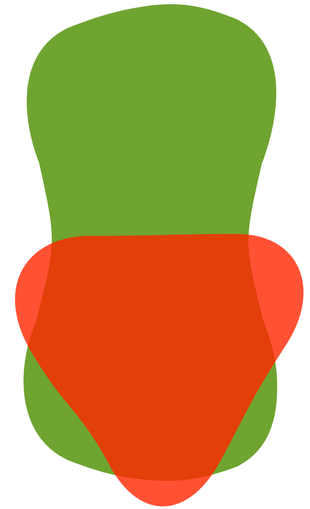
```
forkHandle = (featureConfig.isOn('flightlyForkHandles)) ?
                new FlightyForkHander(aCandle) :
                new ForkHandler(aCandle)
```

www.togglz.org

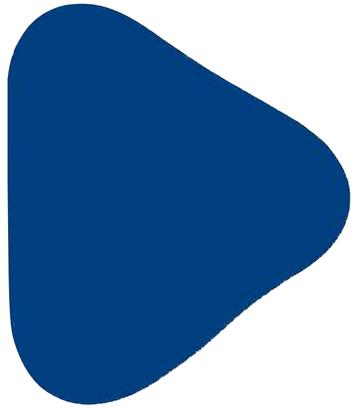


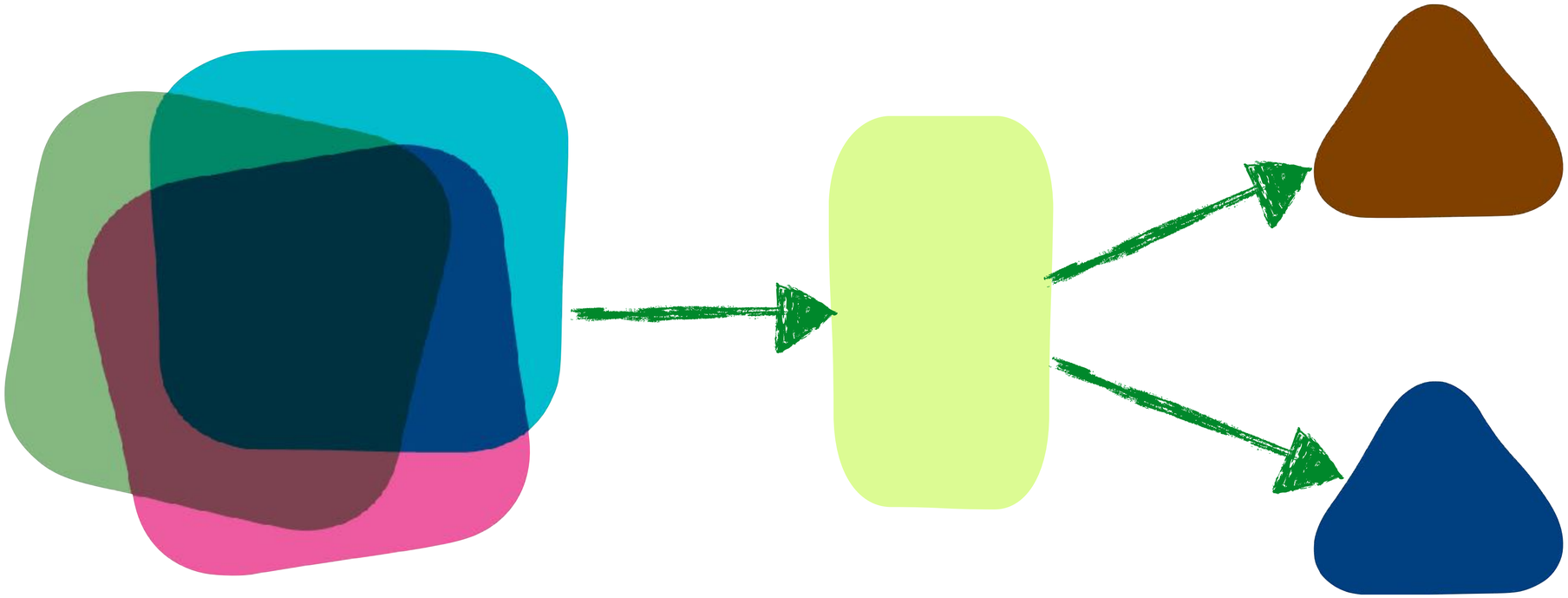
removed as soon as feature decision is resolved

Feature toggles are purposeful technical debt added to support engineering practices like Continuous Delivery.

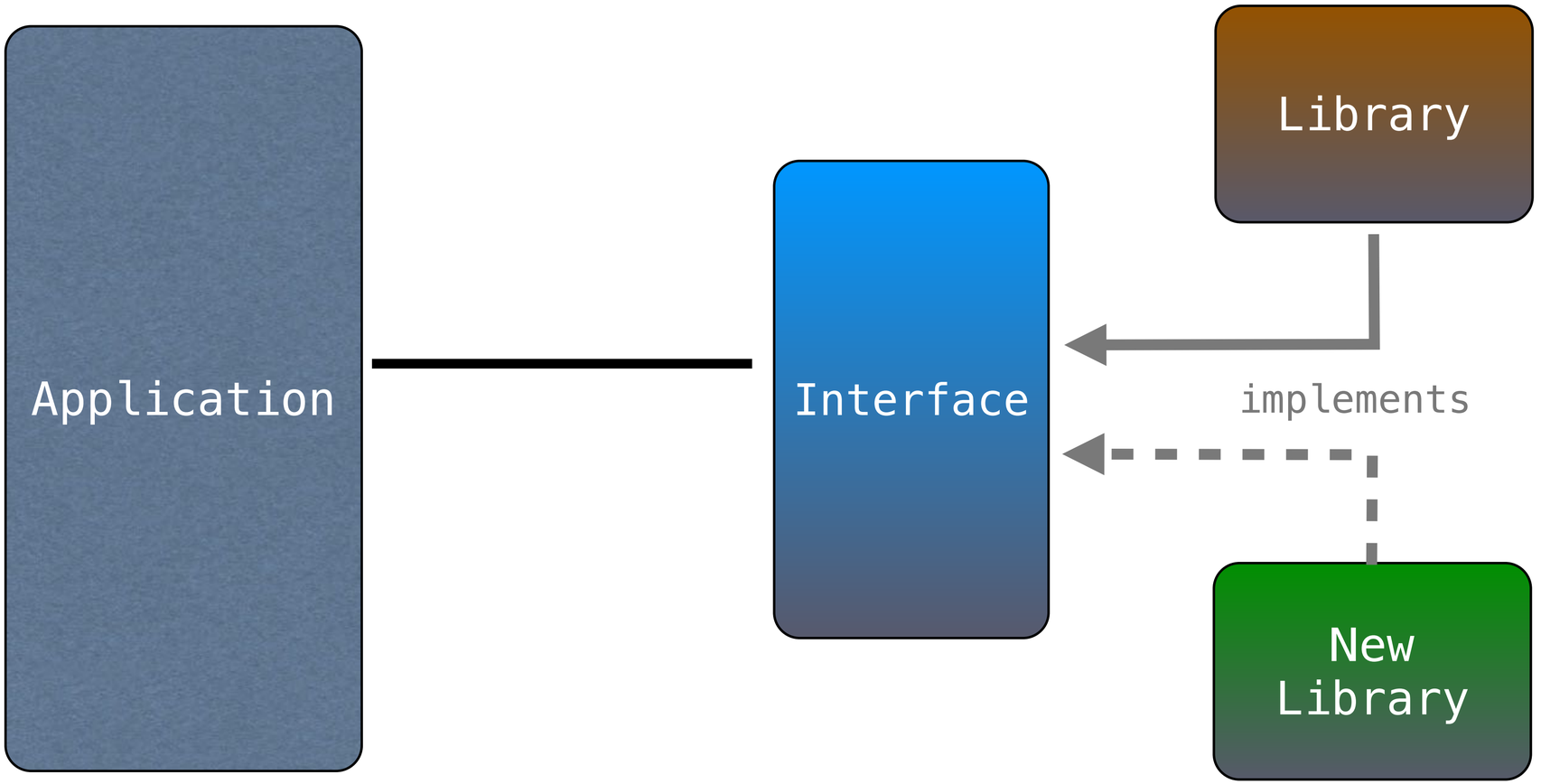


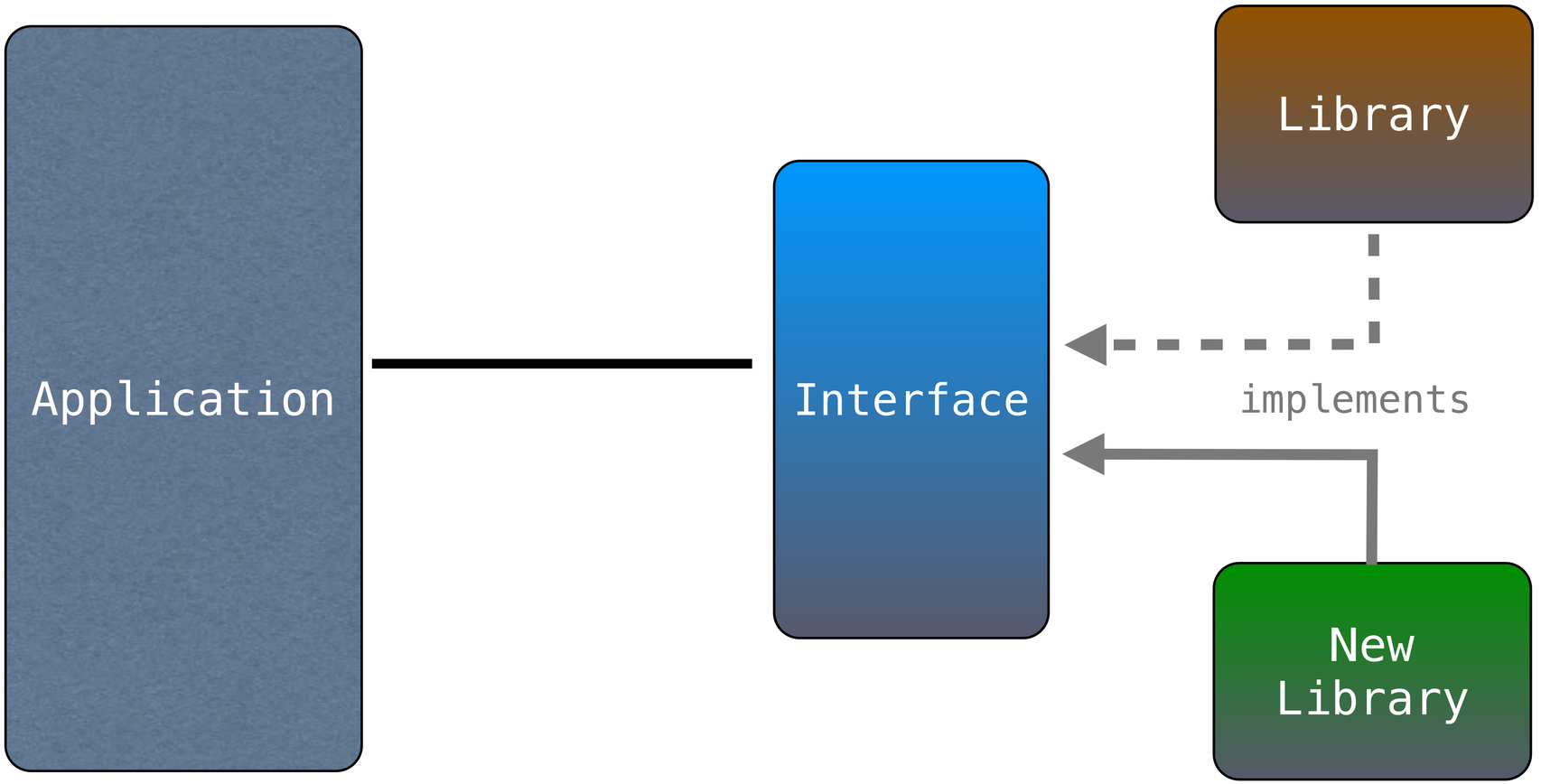
build-time vs. run-time

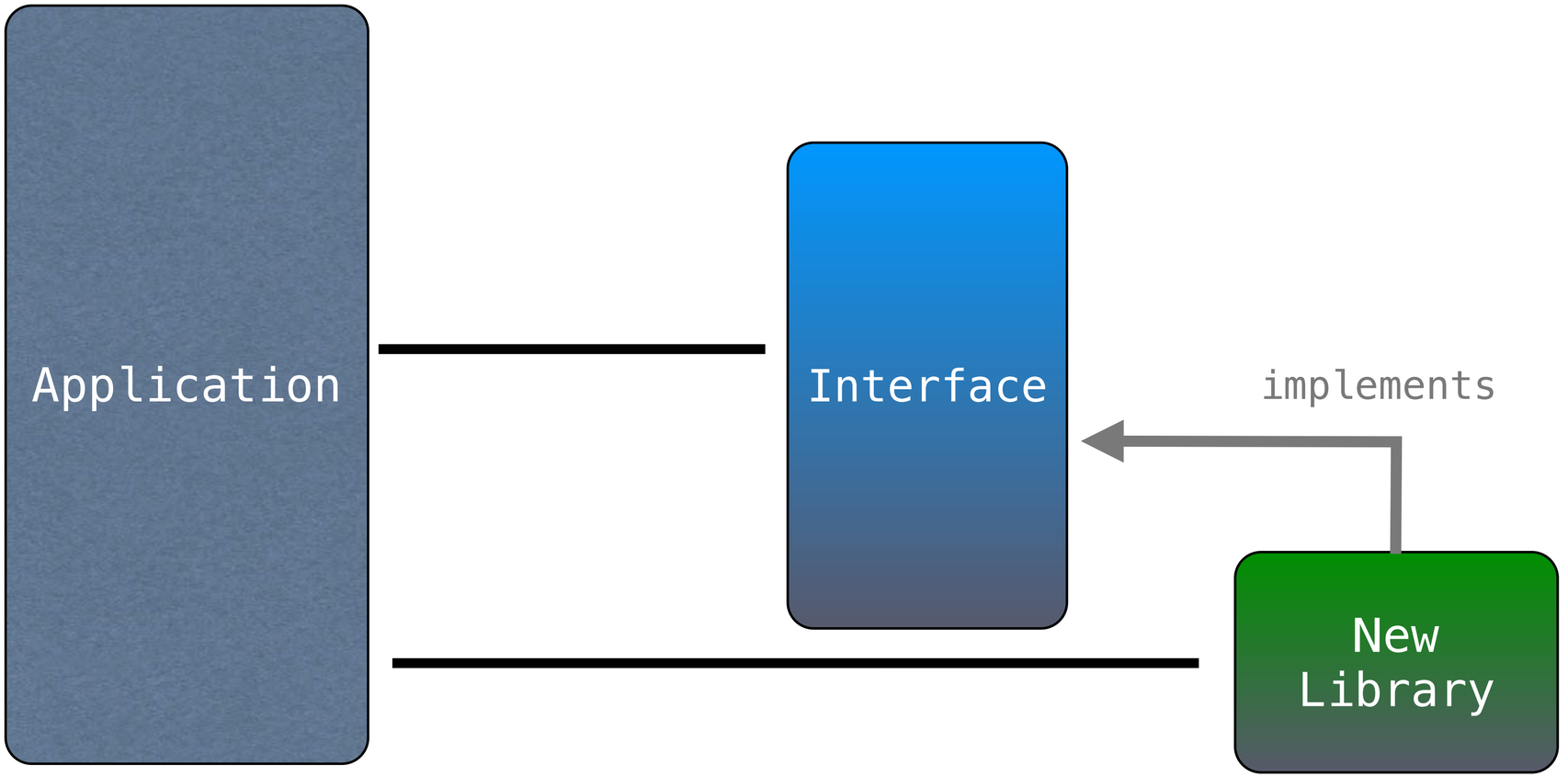




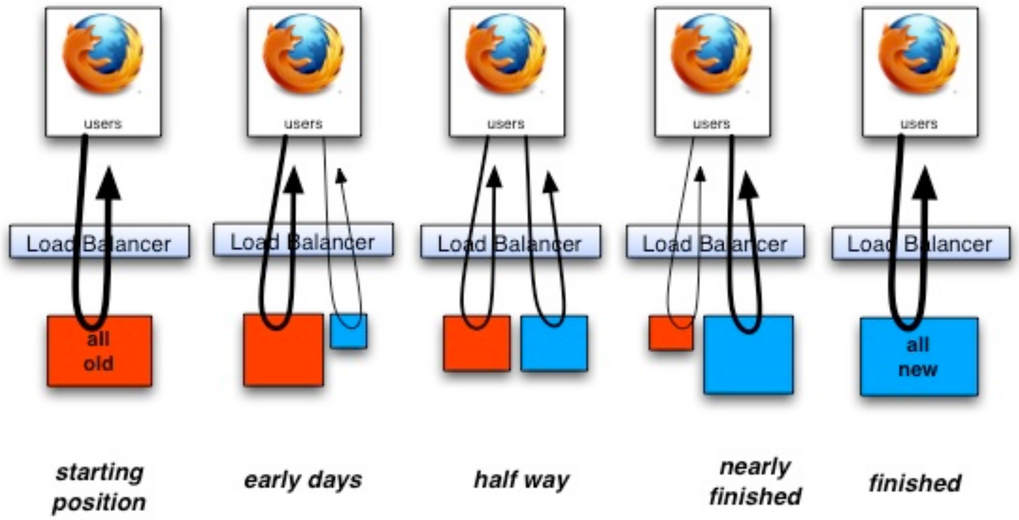
Branch by Abstraction


Application
Interface
Library
implements
New
Library


Application
Interface
Library
implements
New
Library


Application
Interface
implements
New
Library

“Strangler” Pattern

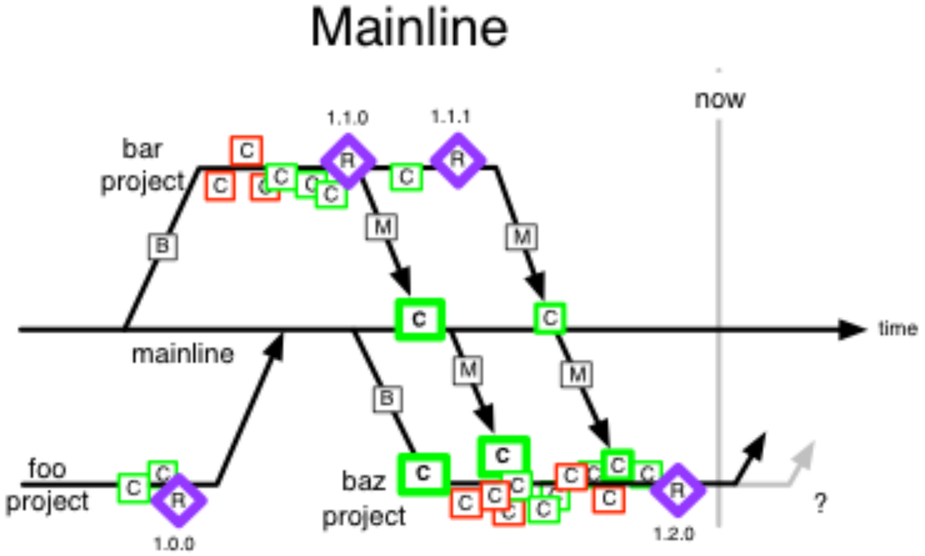


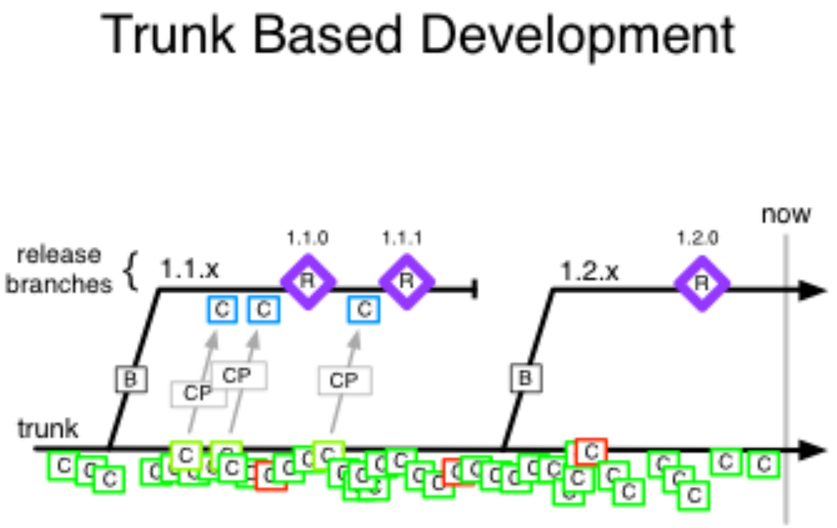
make something new that obsoletes a small percentage of something old

put them live together

rinse, repeat

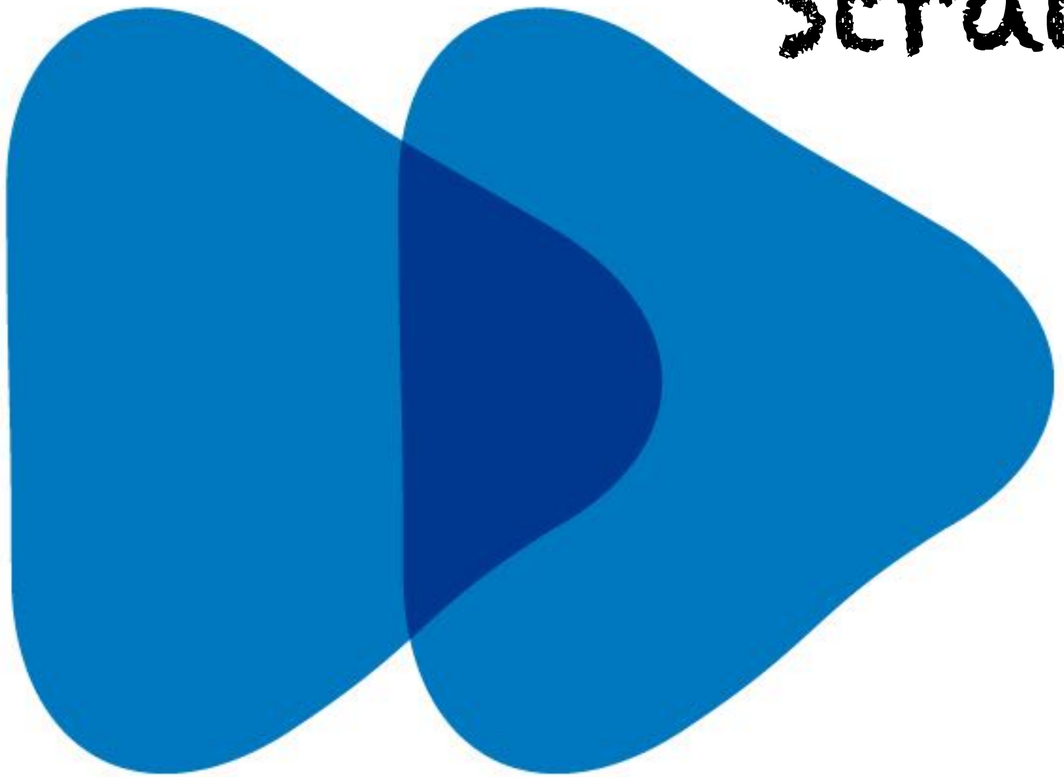
Release branches are OK...





Long-lived branches damage continuous integration.

Release Strategies



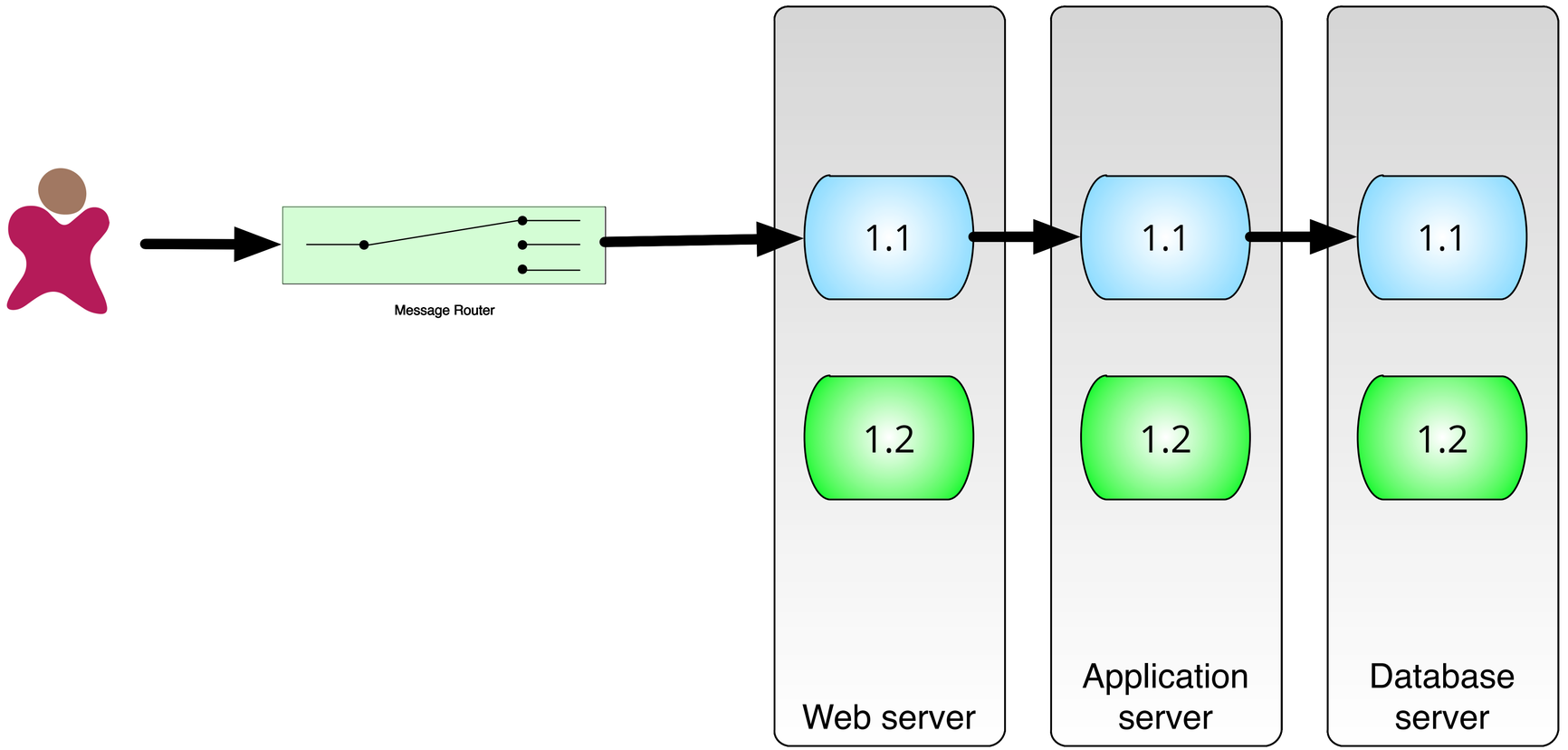
blue-green deployments

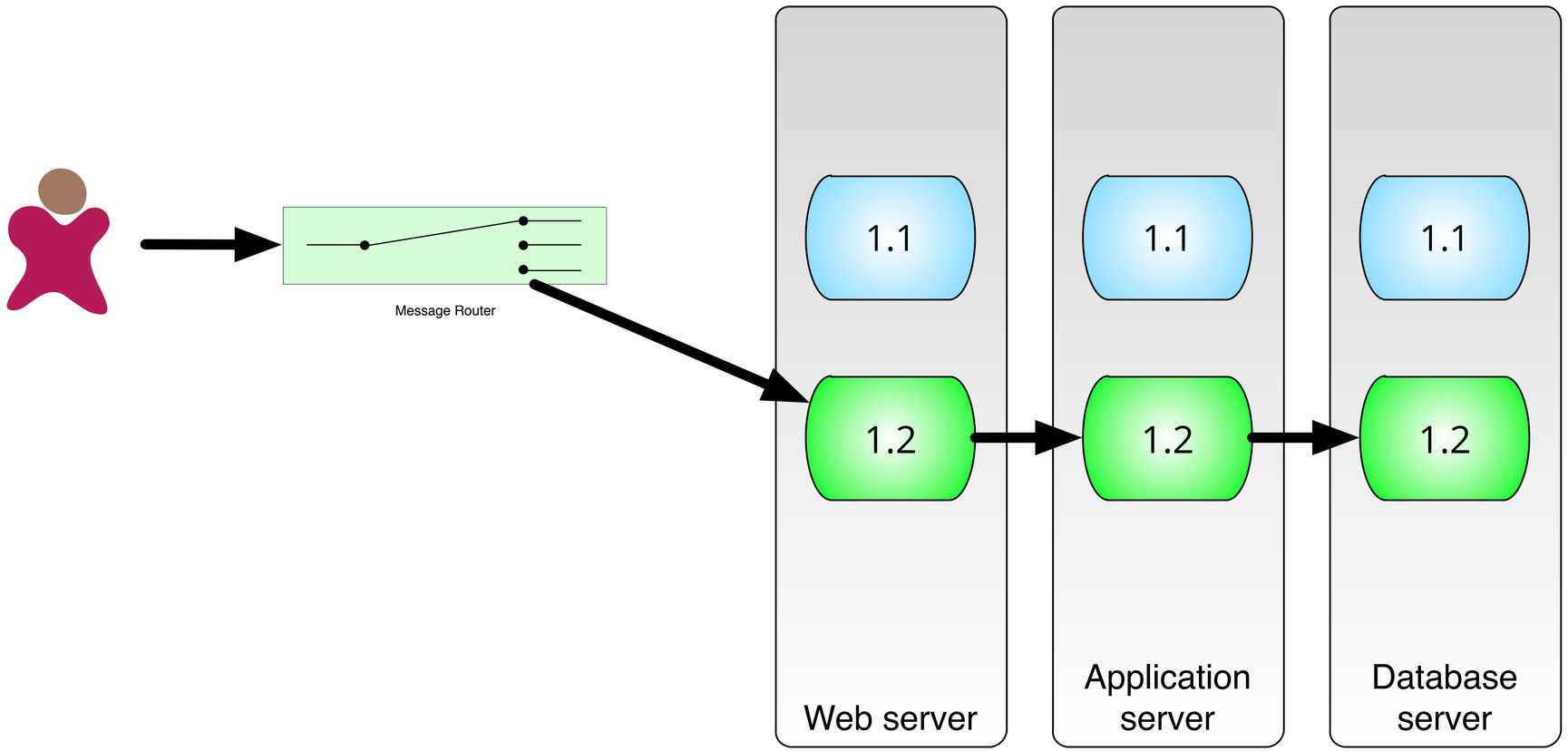
canary releases

Incremental Release Strategies

dark launching

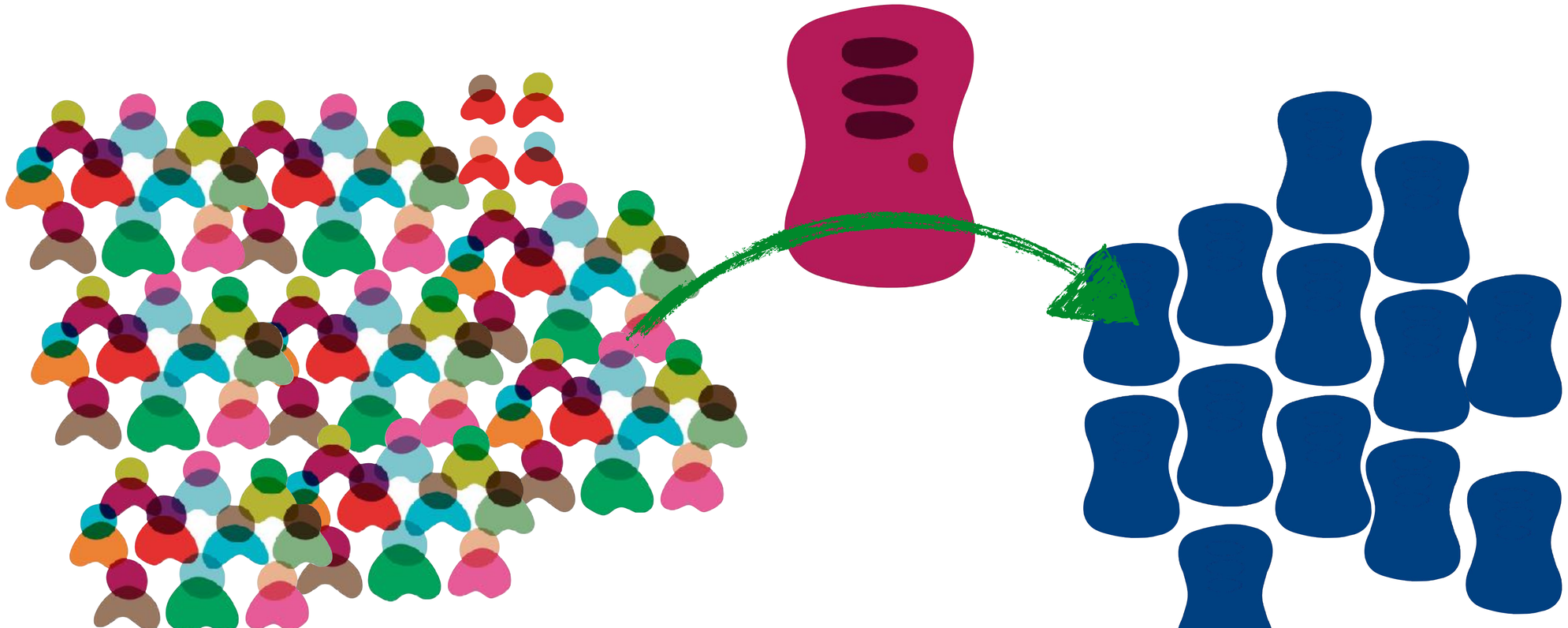
blue-green deployments


Message Router
1.1
1.2
1.1
1.2
1.1
1.2
Web server
Application server
Database server

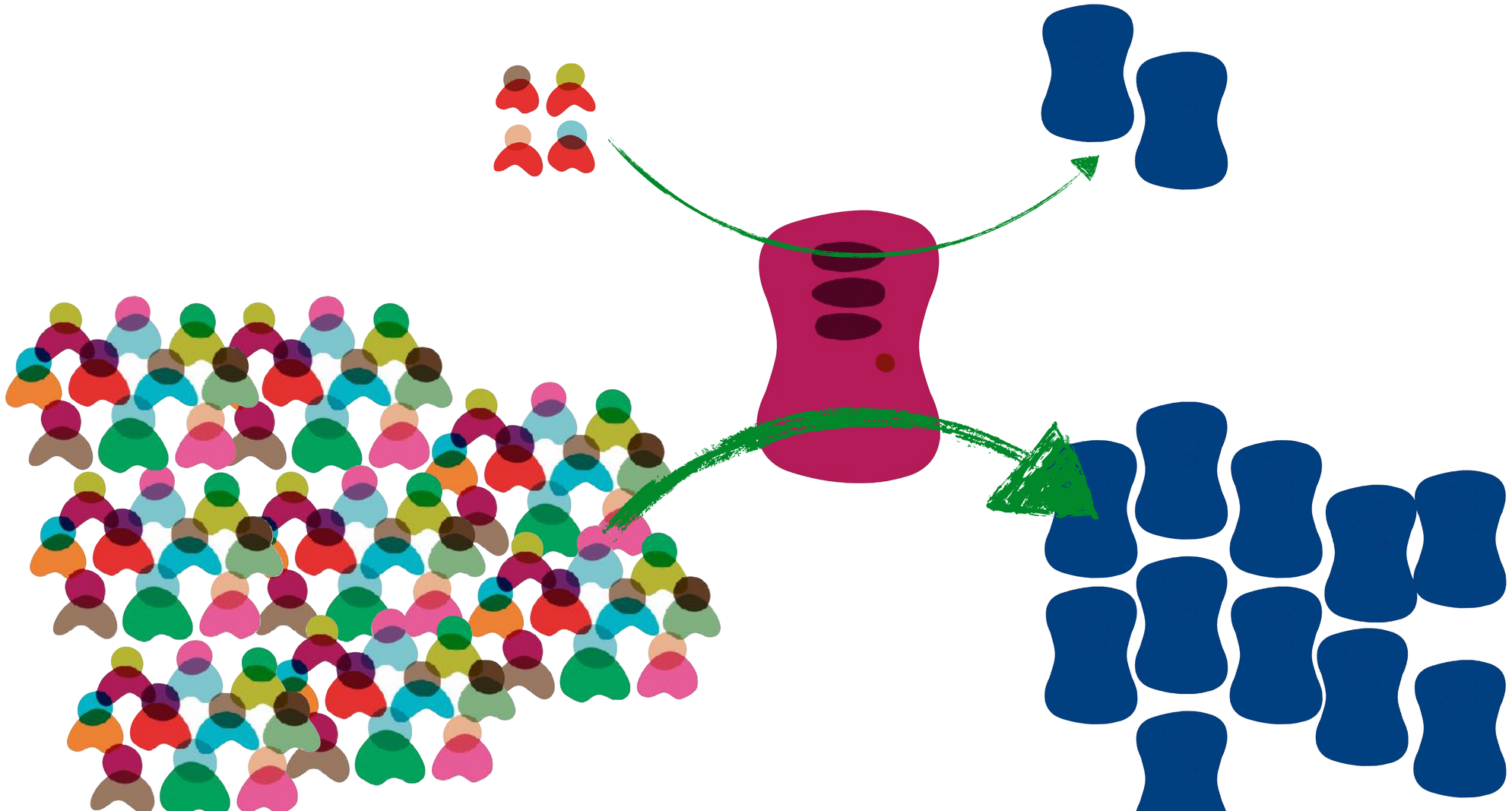

Message Router
1.1
1.2
1.1
1.2
1.1
1.2
Web server
Application server
Database server

Canary Releasing

Canary Releasing

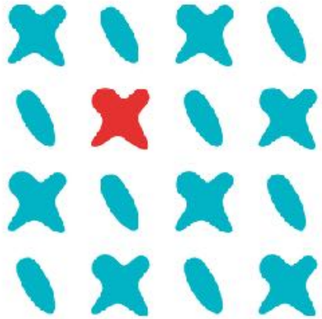


Canary Releasing

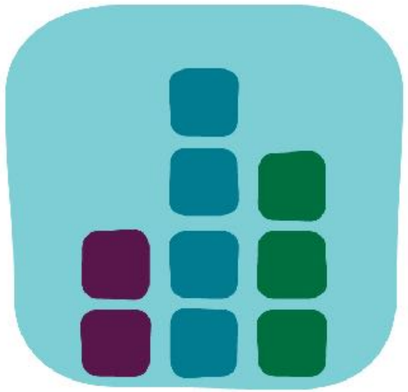


Canary Releasing

reduce risk of release

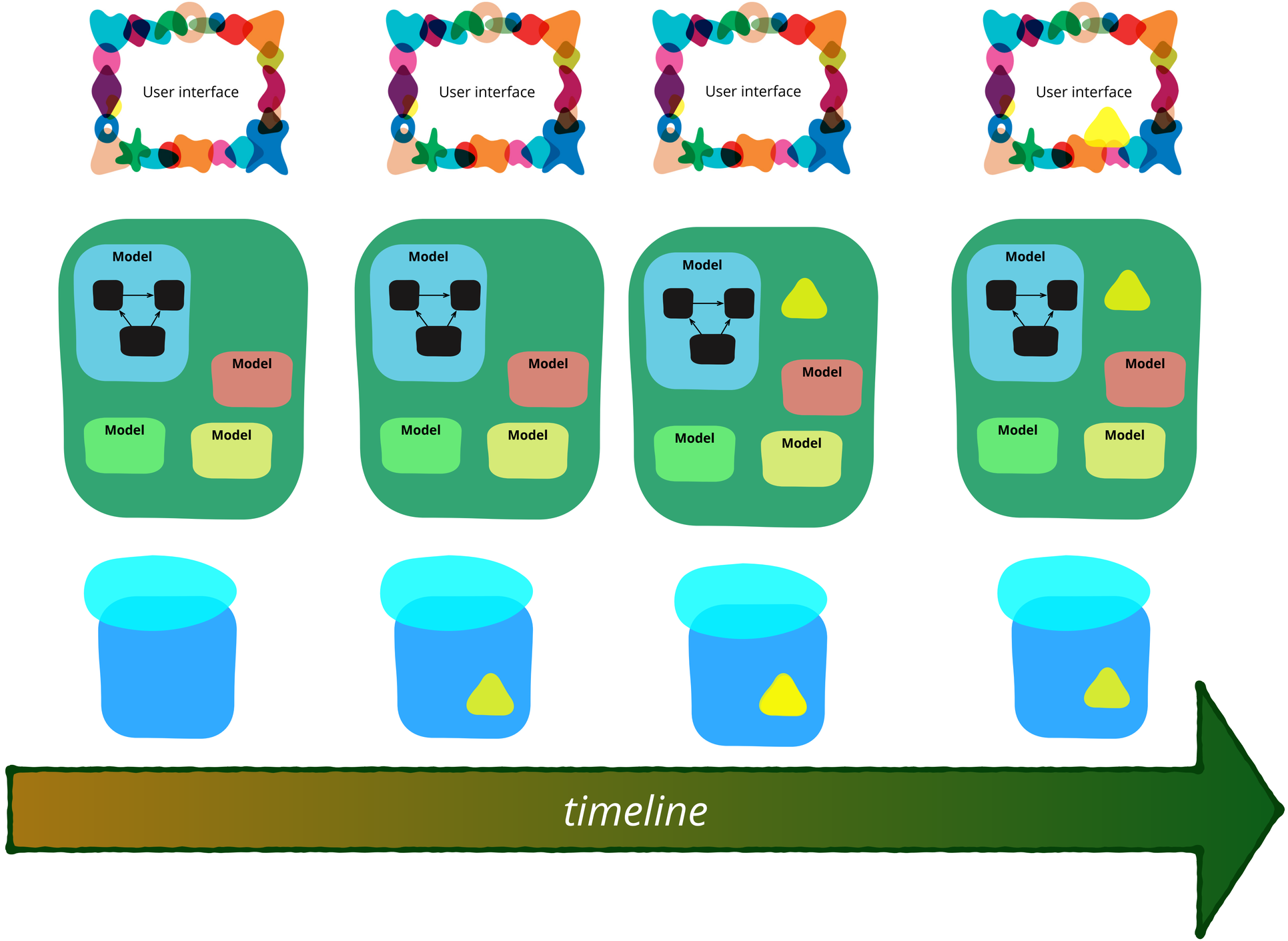


multi-variant testing

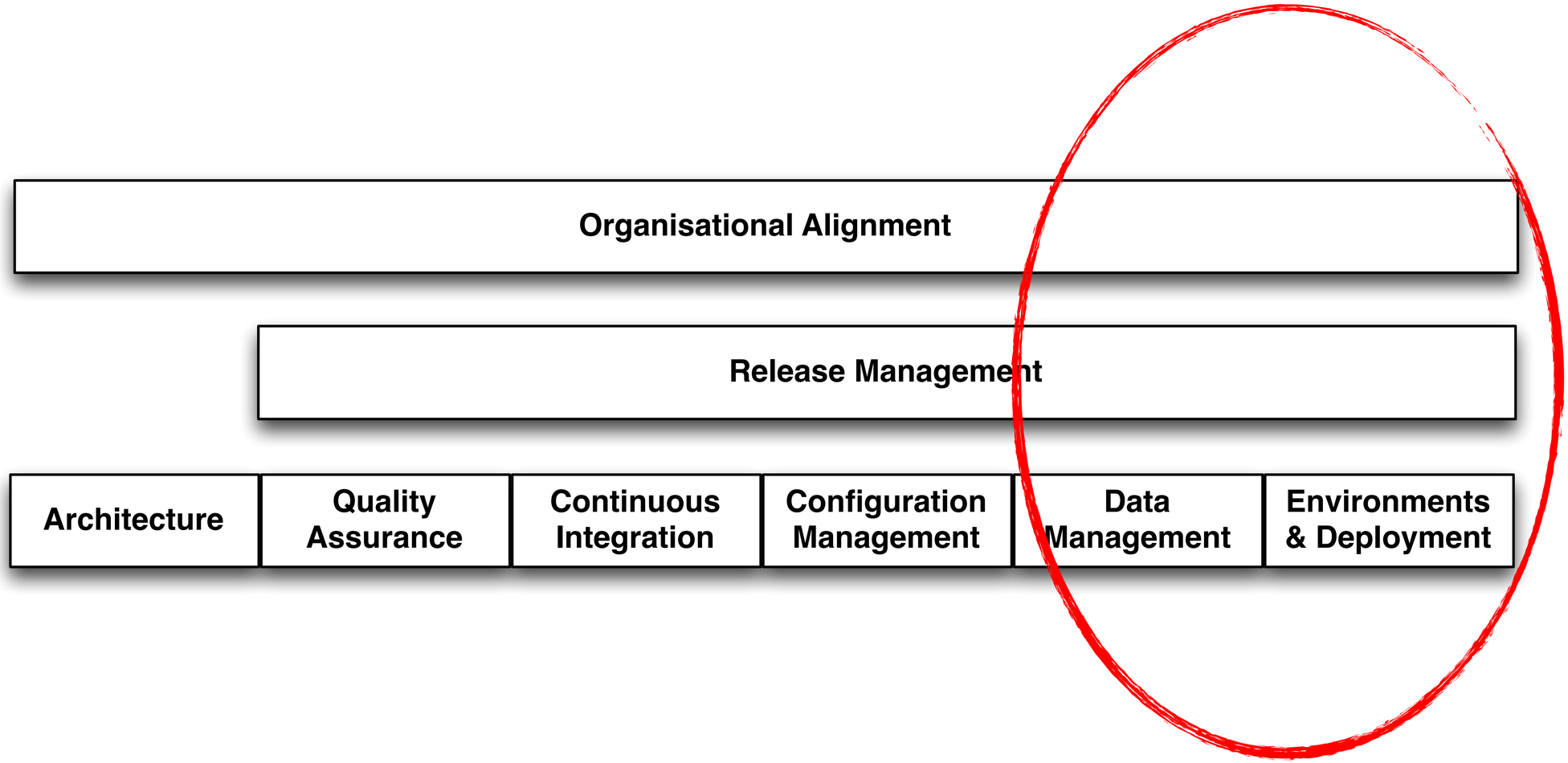


performance testing

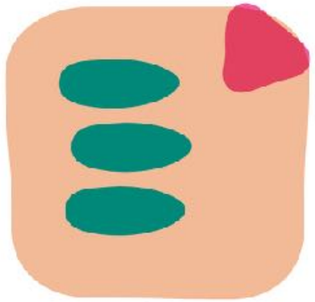
Dark Launching


User interface
User interface
User interface
User interface
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
Model
timeline

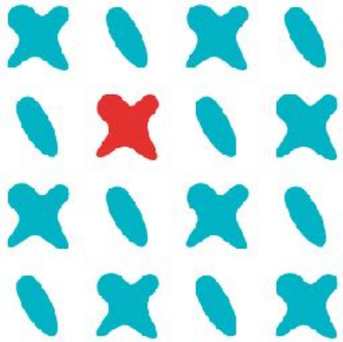
Continuous Delivery



essential complexity in data



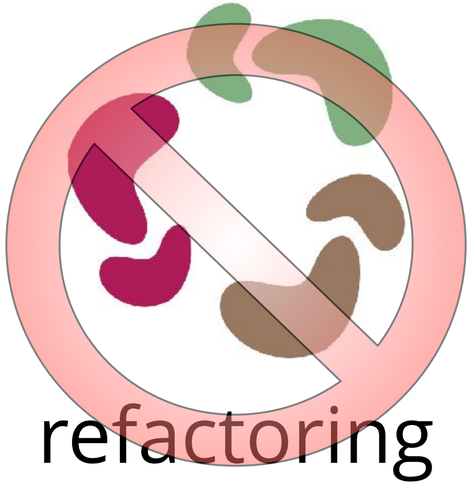
persistent



impedance mismatch

bulky

accidental complexity in data



refactoring

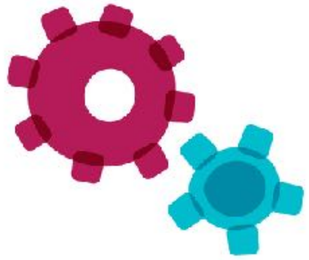
poor collaboration

late integration

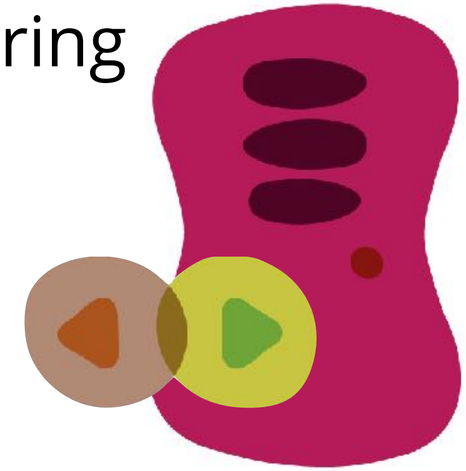
manual work

DB Evolution & Deployment

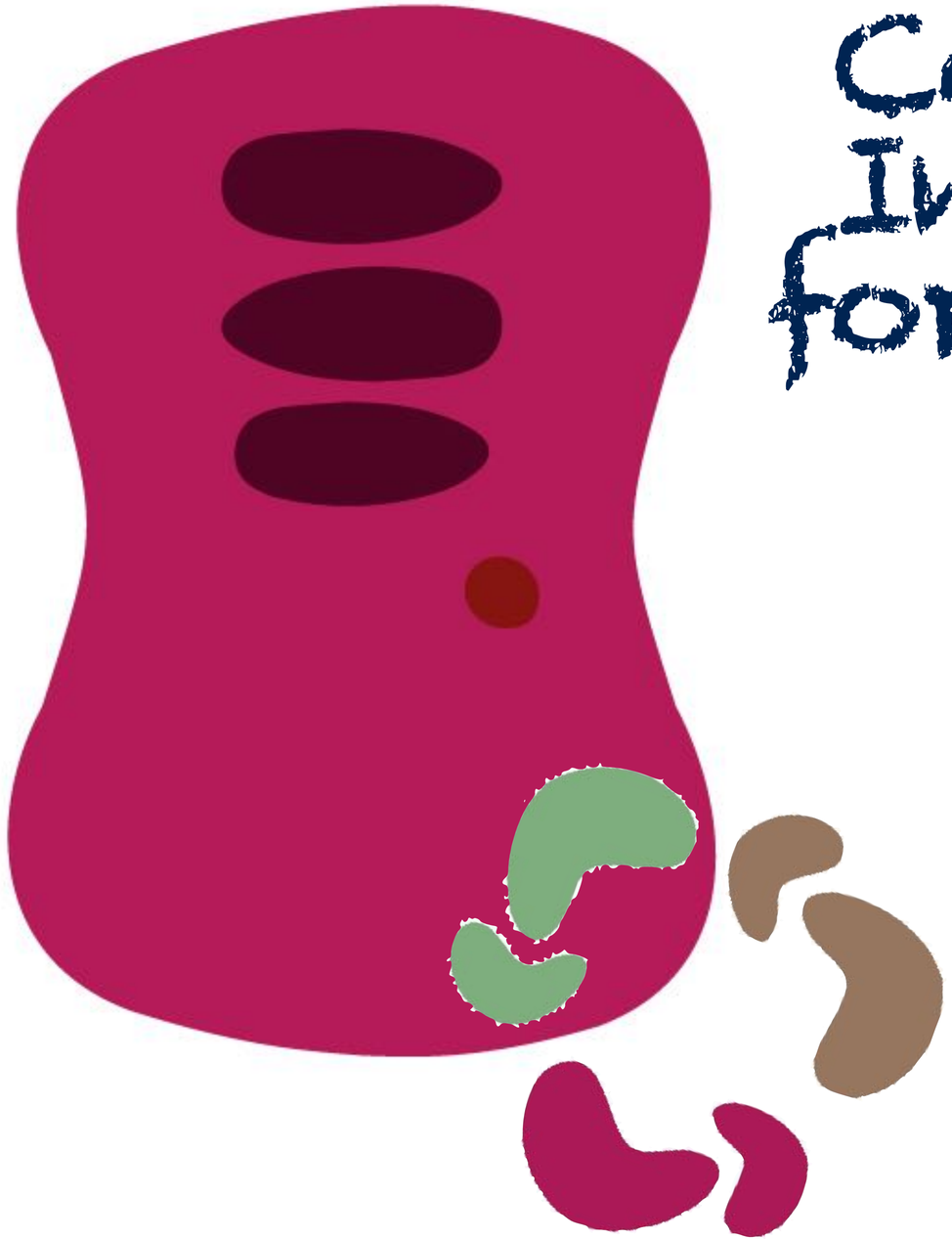
scripting all db changes incrementally



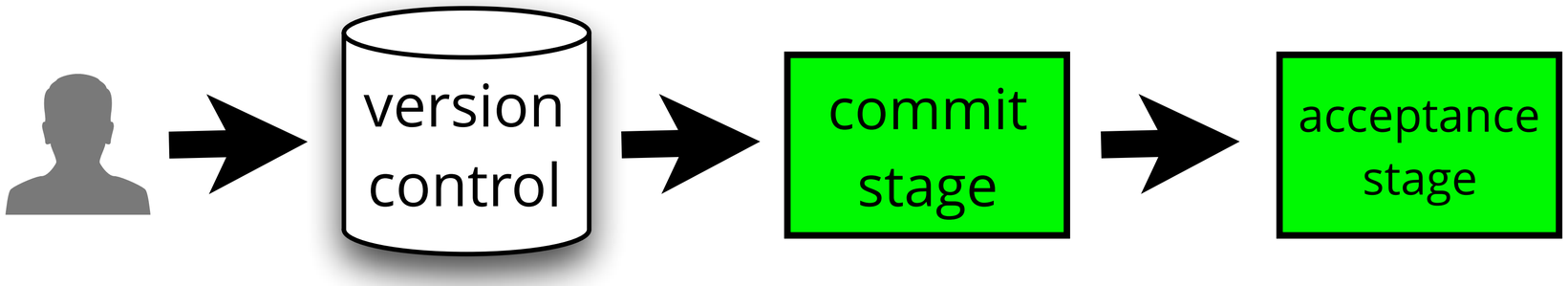
db refactoring

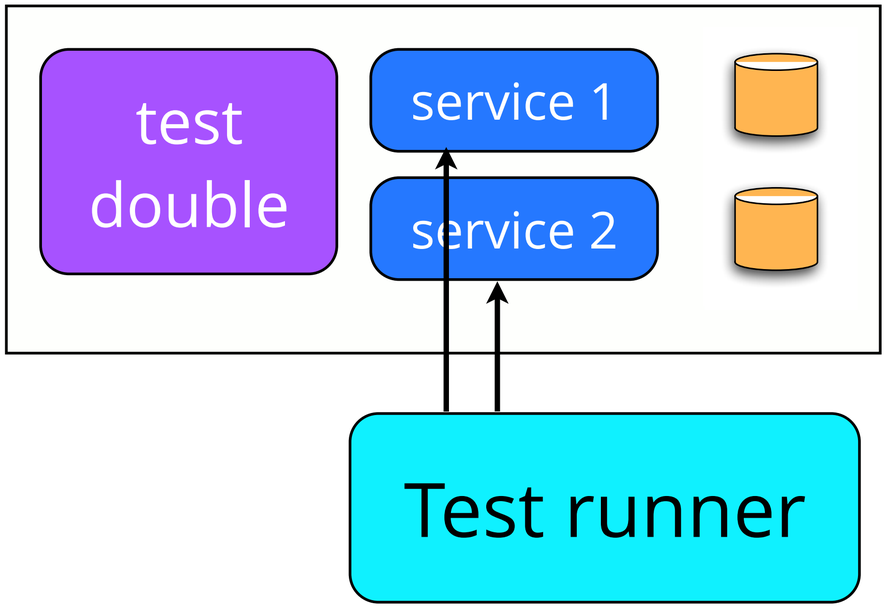


decouple db migration
from app migration



Continuous Integration for Databases





Prepare environment

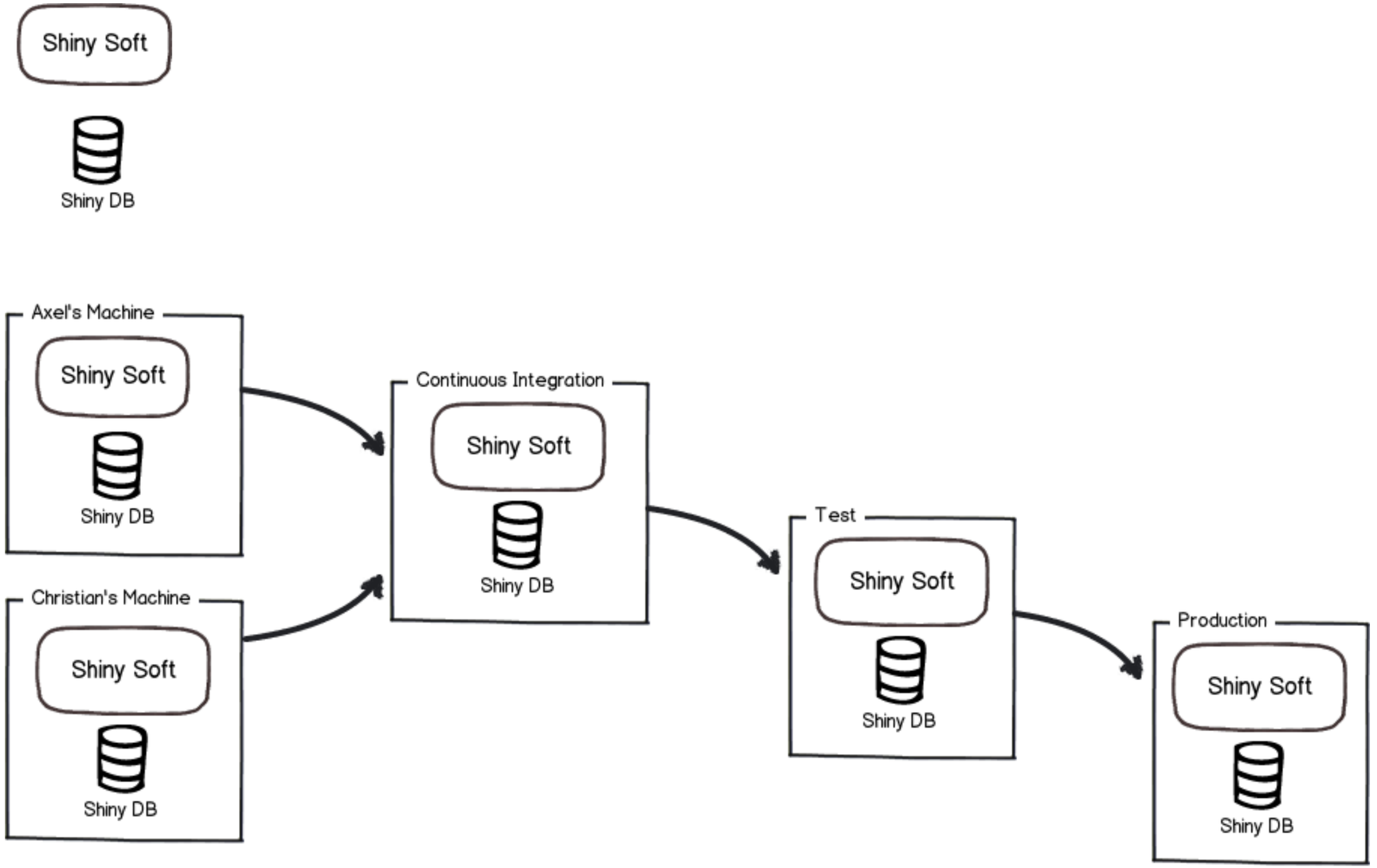
Deploy app

Create dbs, apply schema

Add app reference data

Run acceptance tests

DbDeploy Pattern



DbBeploy Tool

db updates are code

small incremental deltas

metadata in the database

fail fast

http://dbdeploy.com *http://www.liquibase.org/* *https://flywaydb.org*

001_create_initial_tables.sql:

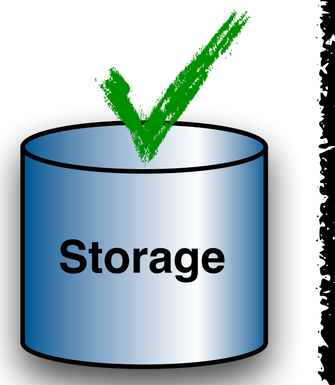
```
CREATE TABLE customer (
  id   BIGINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1)
  PRIMARY KEY,
  firstname VARCHAR(255),
  lastname VARCHAR(255)
);
```

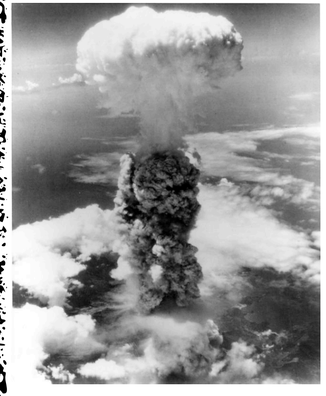
002_add_customer_date_of_birth.sql

```
ALTER TABLE customer ADD COLUMN dateofbirth DATETIME;

--//@UNDO

ALTER TABLE customer DROP COLUMN dateofbirth;
```

For DB CI We Need To:

start with a clean database





apply changes incrementally

use the same process everywhere

be comprehensive in change management

Apply Deltas

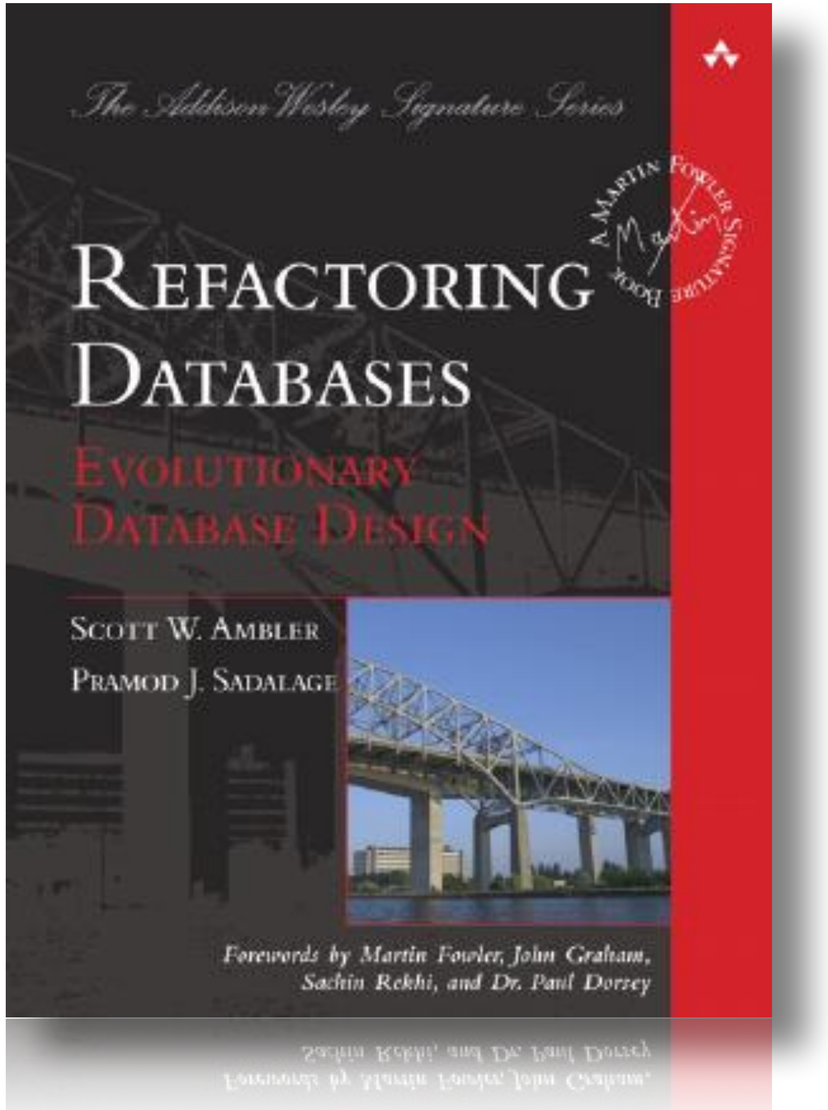
run each delta in order

stop the line if one delta fails

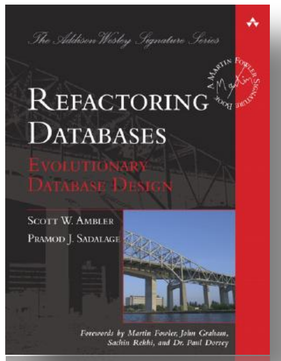
auto-rollback if possible

record success in db metadata table

Refactoring Databases



Decouple DB Updates: the Expand/contract Pattern



Every change you make must be backward compatible

app v205 compatible with db v13 and v14

app v230 compatible with db v14

app v234 compatible with db v14

app v241 compatible with db v14 and 15

app v248 compatible with db v15

DB version 13

DB version 14

DB version 15

app v205 deployed

migrate db to v14

app v230 deployed

app v234 deployed

app v234 deployed

migrate db to v15

app v248 deployed

Time

decouple db updates

dark launch db updates

Never tie DB migrations to application deploys

long-running upgrades

abstraction layer in code or stored procs / views

DB Deployments Still Hard

practice, practice, practice

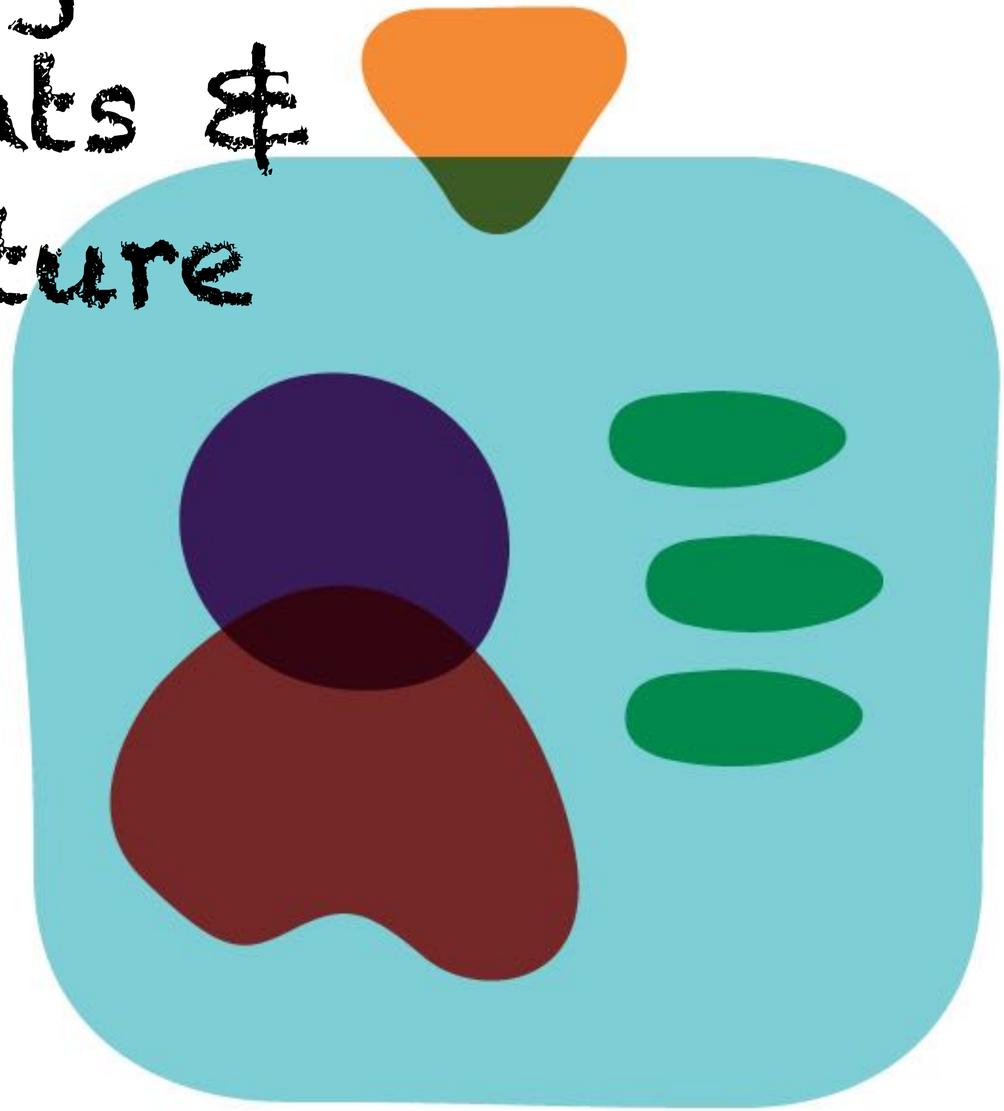
fail fast

bring the pain forward

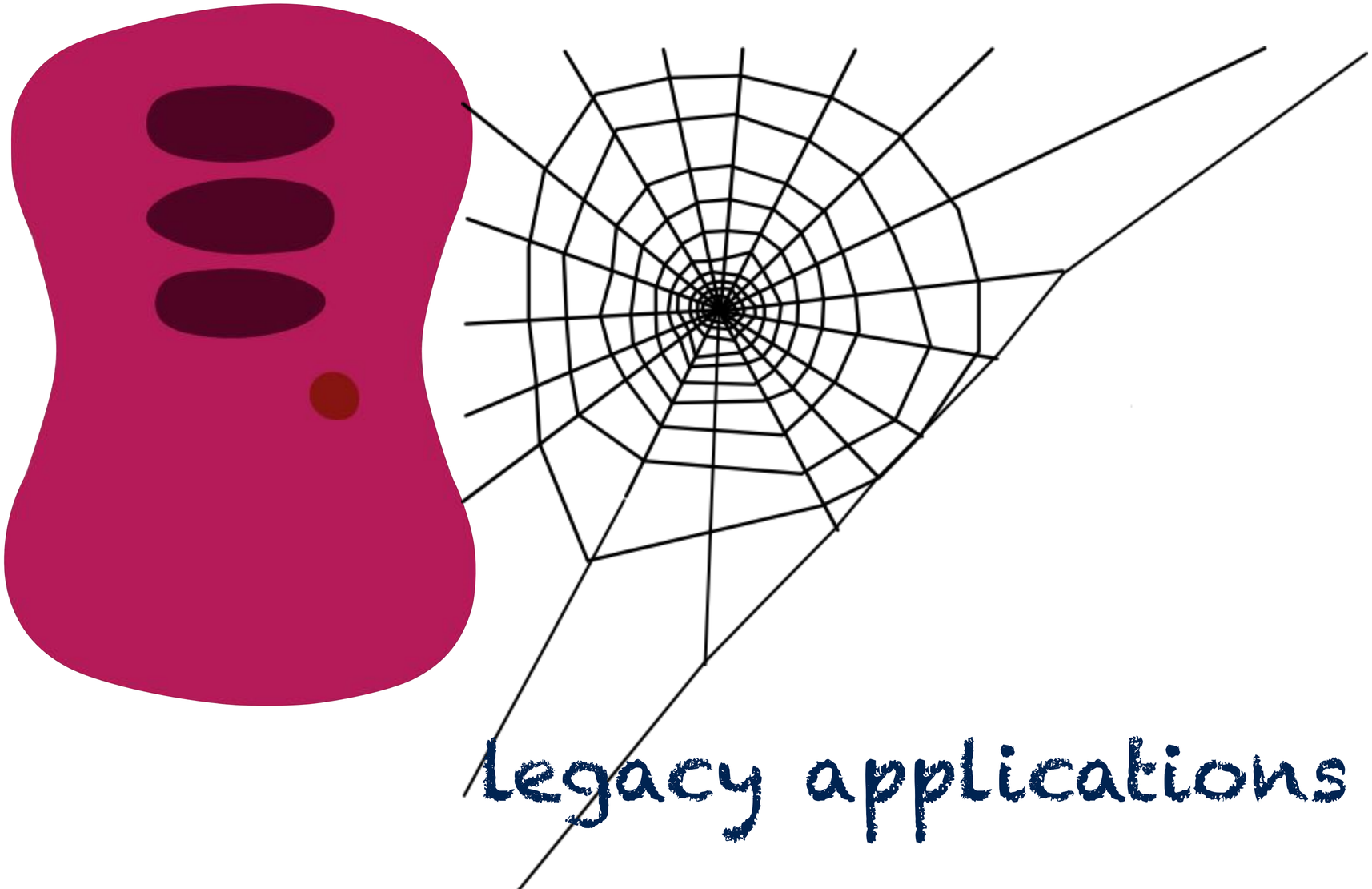
refactor the db

update engineering practices

Managing Environments & Infrastructure

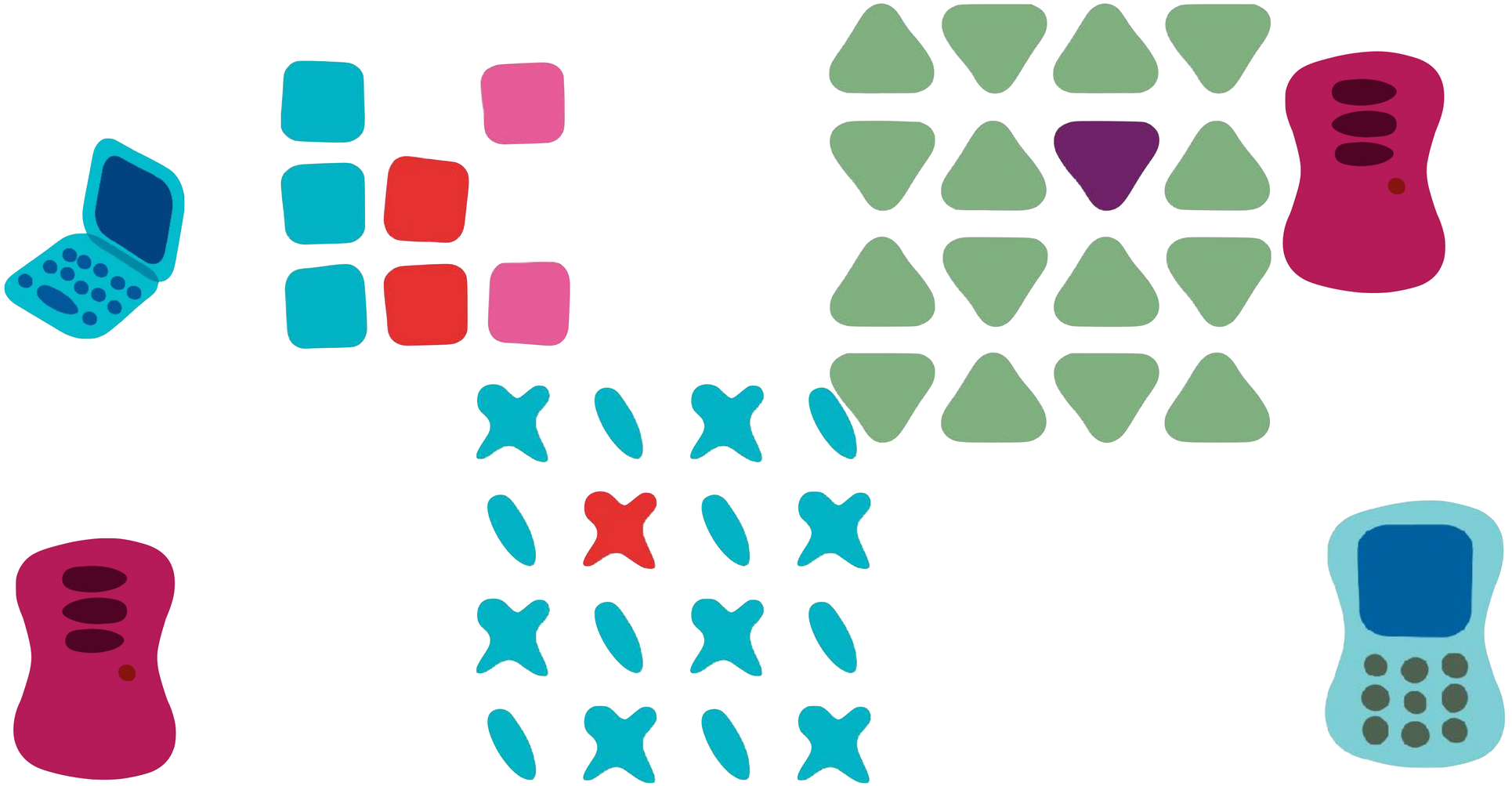


The Pain of Operations

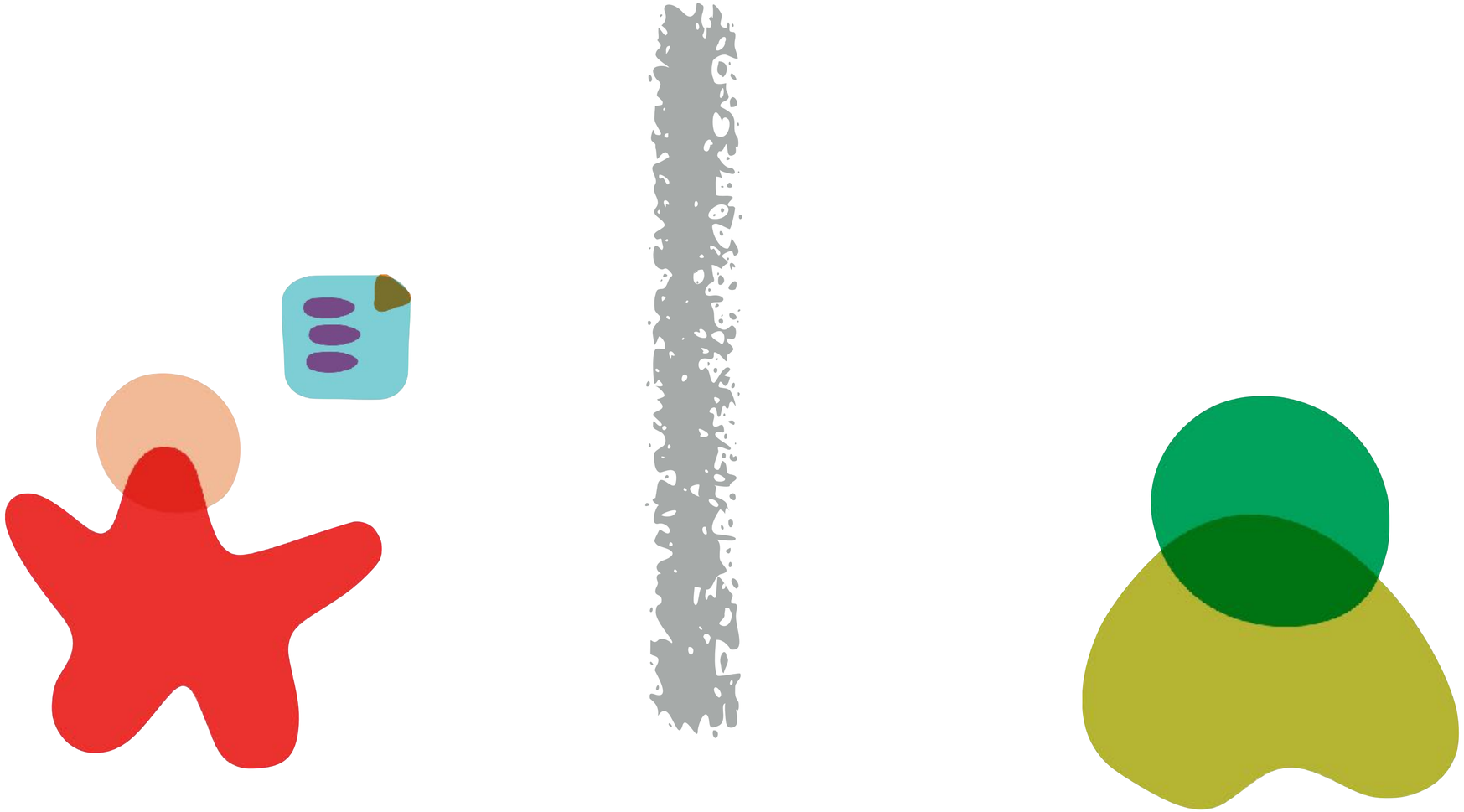


The Pain of Operations

heterogeneous platforms

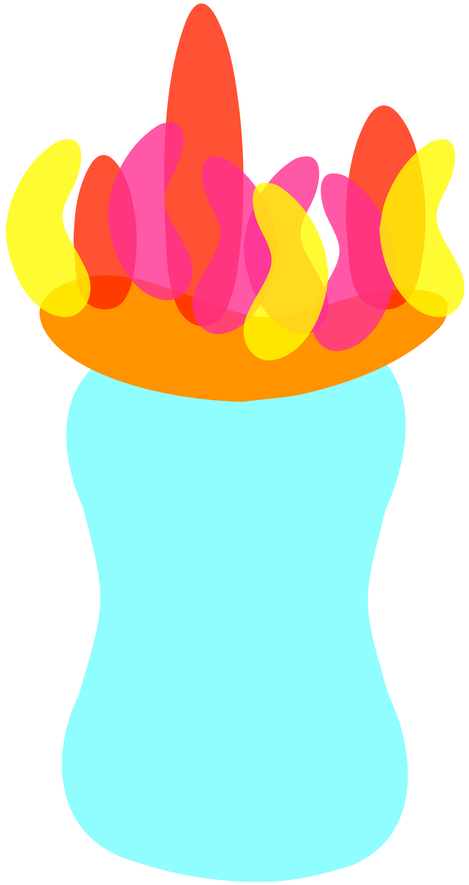


The Pain of Operations



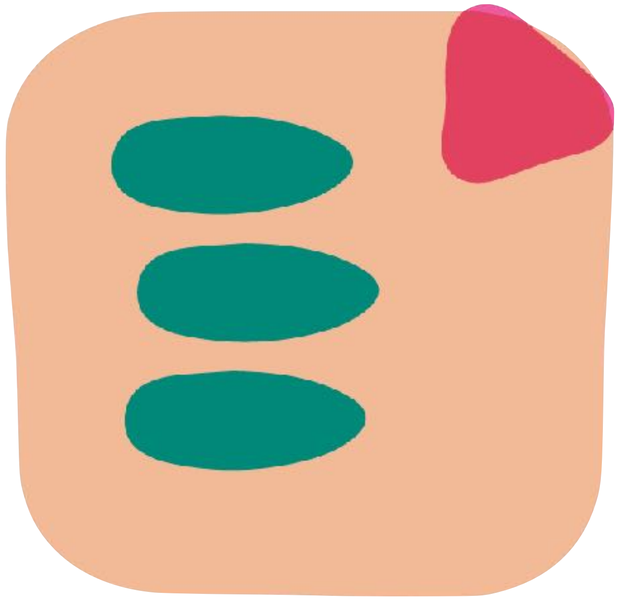
poor quality software thrown over a wall

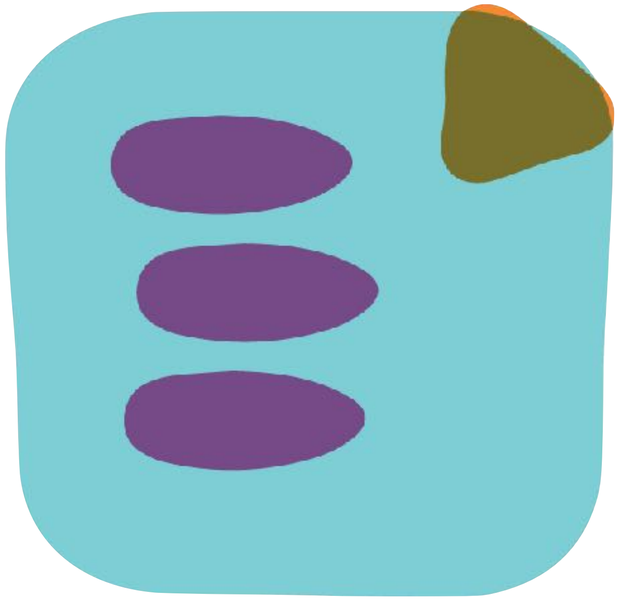
The Pain of Operations



inordinate amount of firefighting

The Pain of Operations





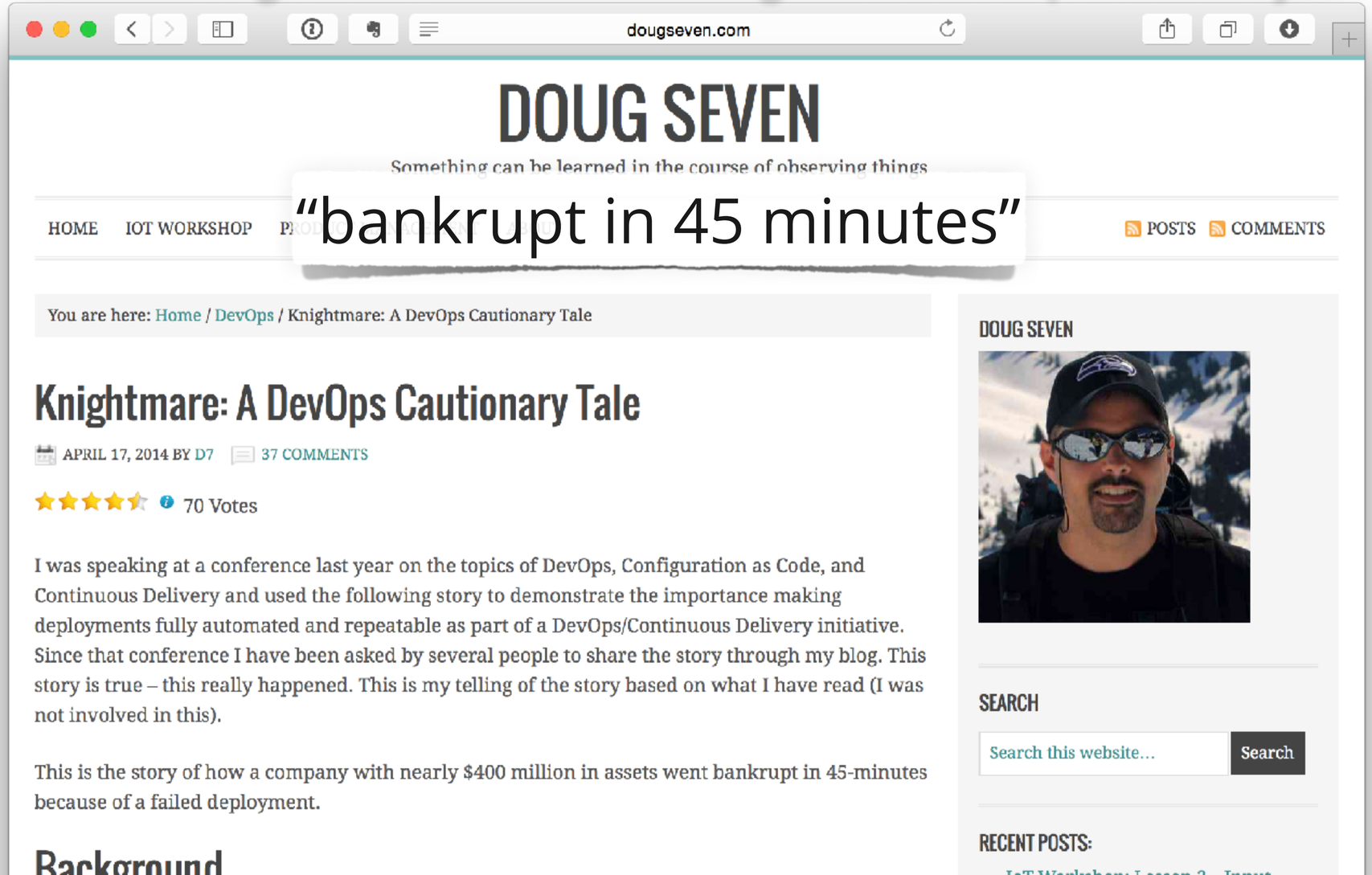
conservative, process heavy

The Pain of Operations

huge budget for operations

Horror Stories

dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/



"bankrupt in 45 minutes"

DevOps

not it’s own silo, but a liaison between operations and developers

at inceptions, showcases, retros

devs work in ops and carry pagers

devs create more deployable software

Managing Infrastructure

infrastructure = environments and supporting services (networking, vcs, storage, mail, dns...)

desired state specified in version control

autonomic (self-corrects to desired state)

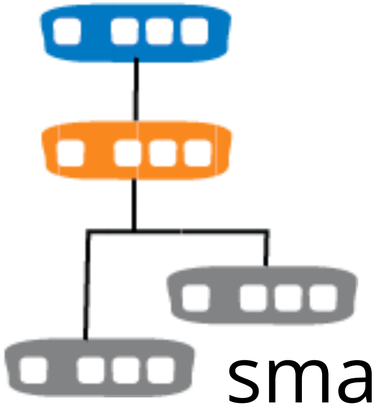
state should be known through monitoring

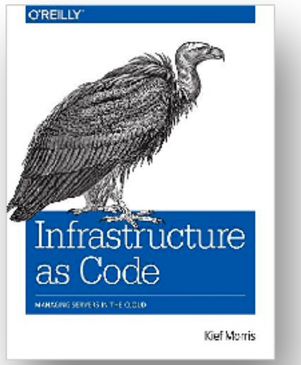
Infrastructure as Code

version all the things

continuously test
systems & processes

definition files



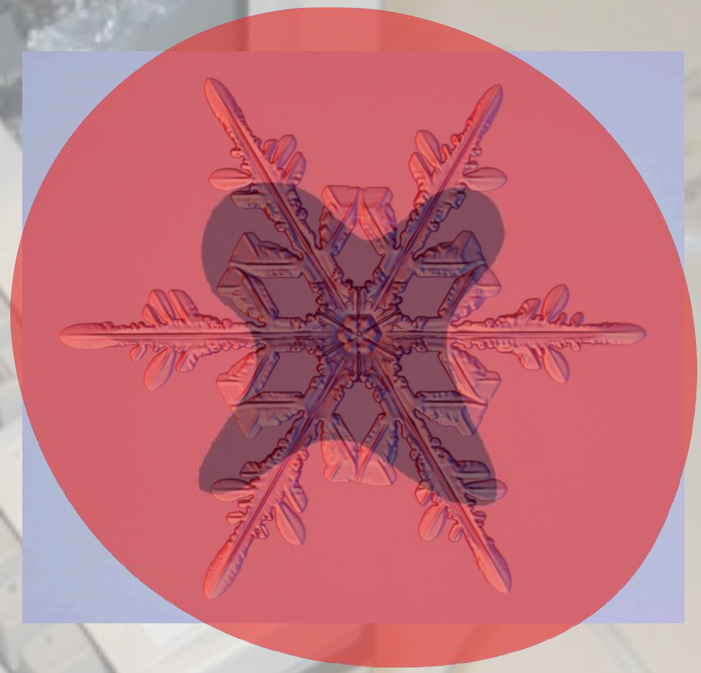


small changes
over large batches

self-documented
systems & processes

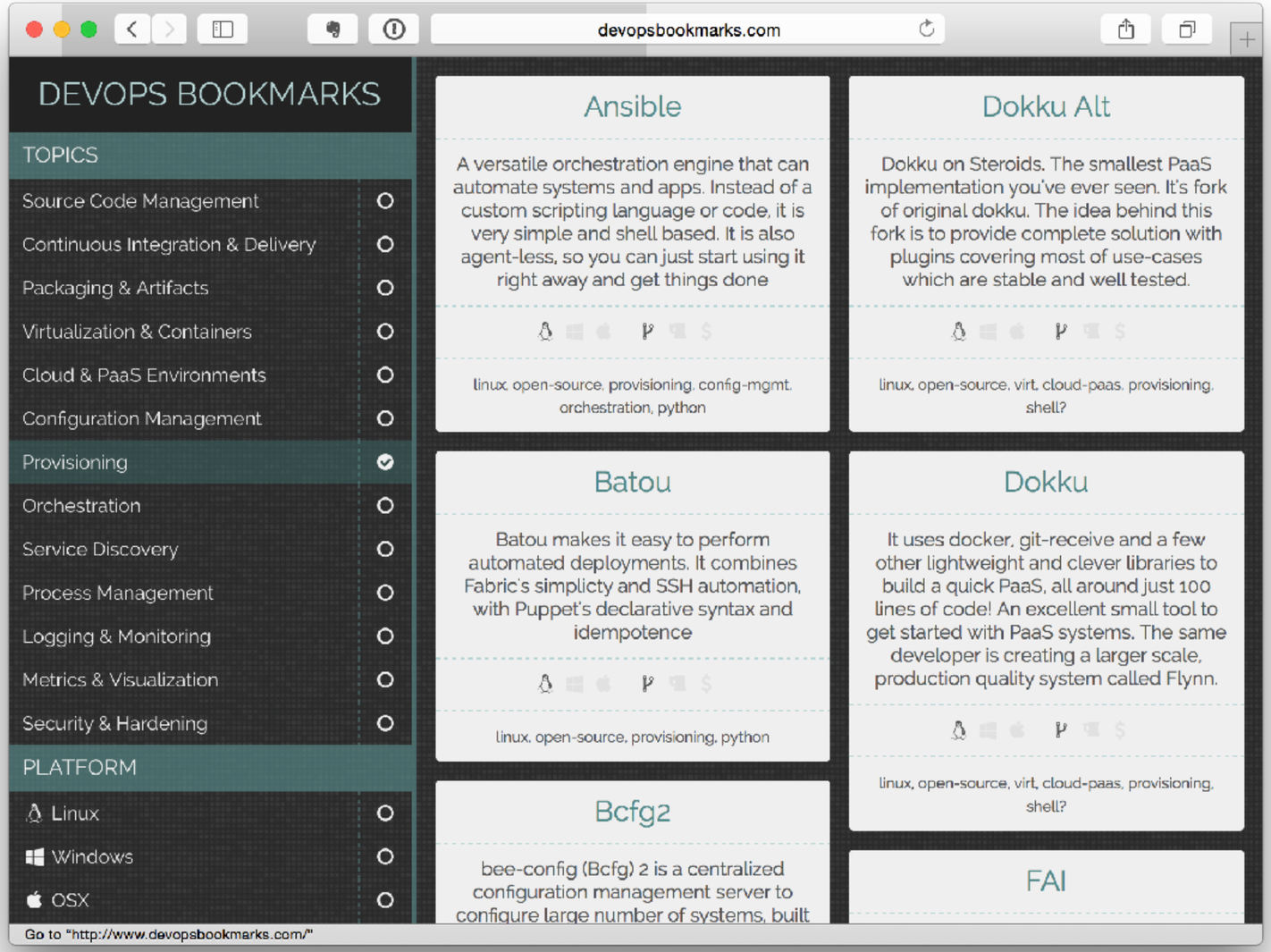
keep services available continuously

Destroy Works of Art



If someone threw a server out of the window, how long would it take to recreate it?

Tools



www.devopsbookmarks.com/

Tools

ANSIBLE

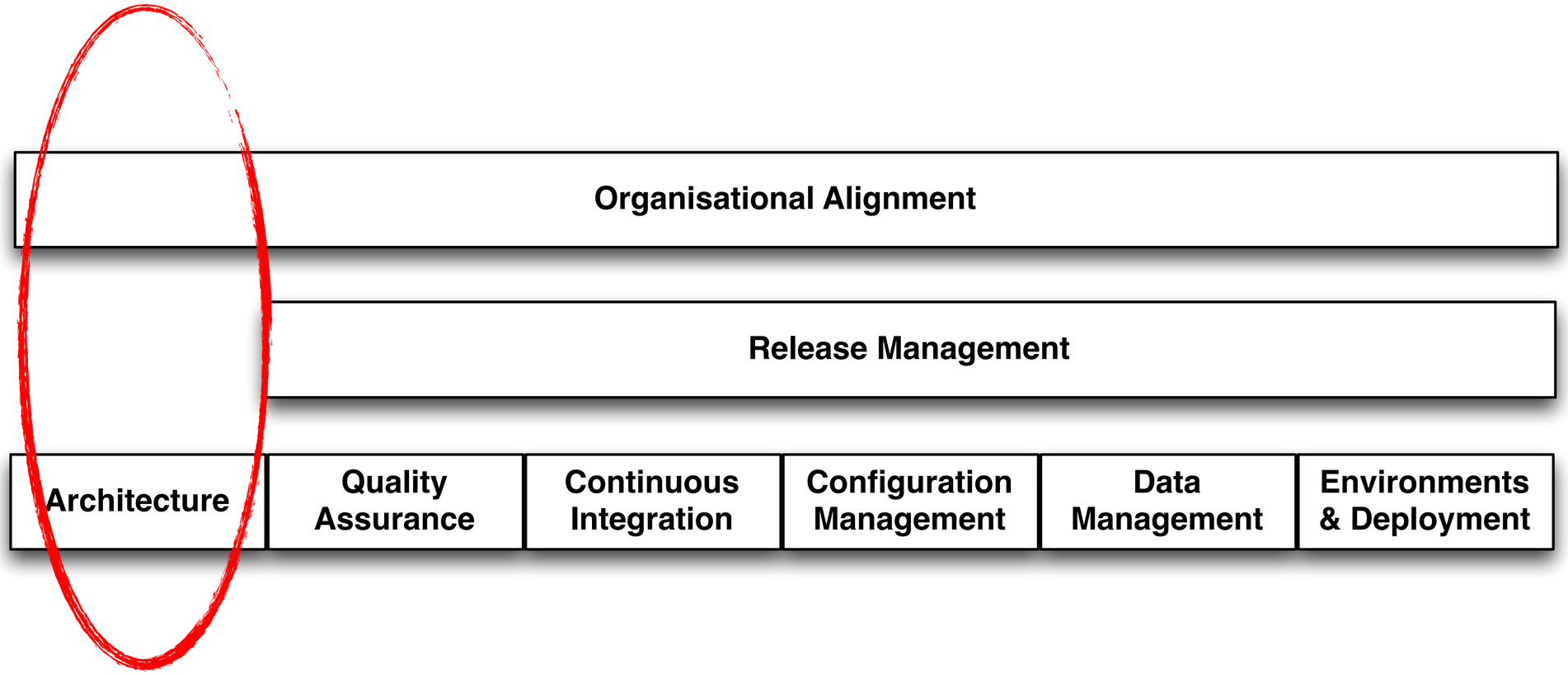
manage many systems

manage configuration

enforce consistency

treat infrastructure as code

Continuous Delivery

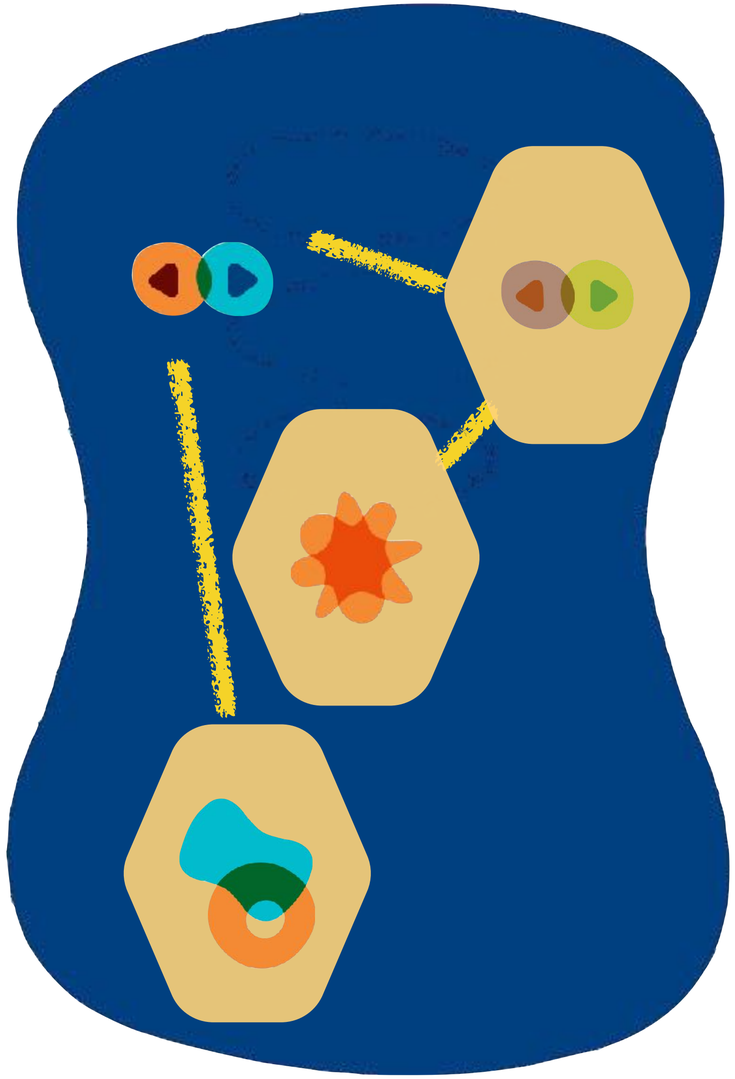


Microservices

Components are *deployed*.

Features are *released*.

Applications consist of *routing*.

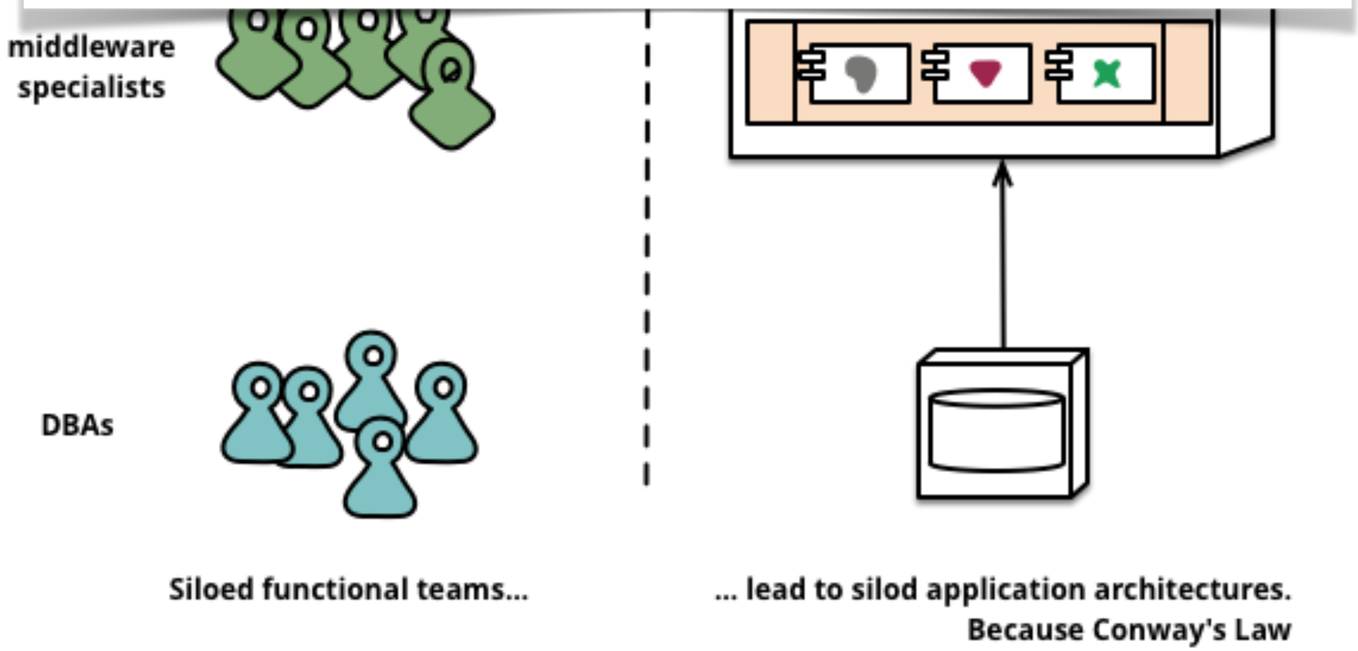


production

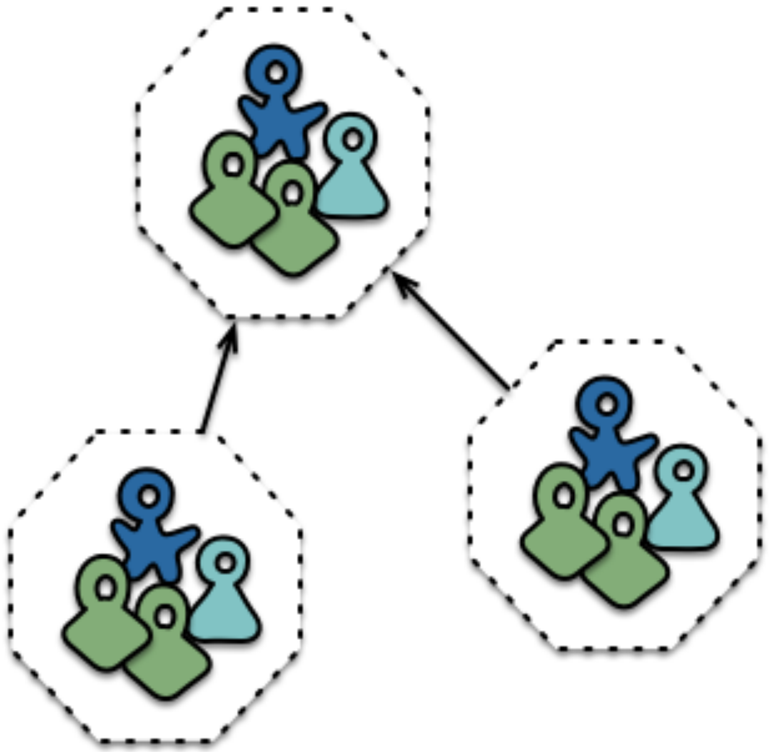
Conway's Law

"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"

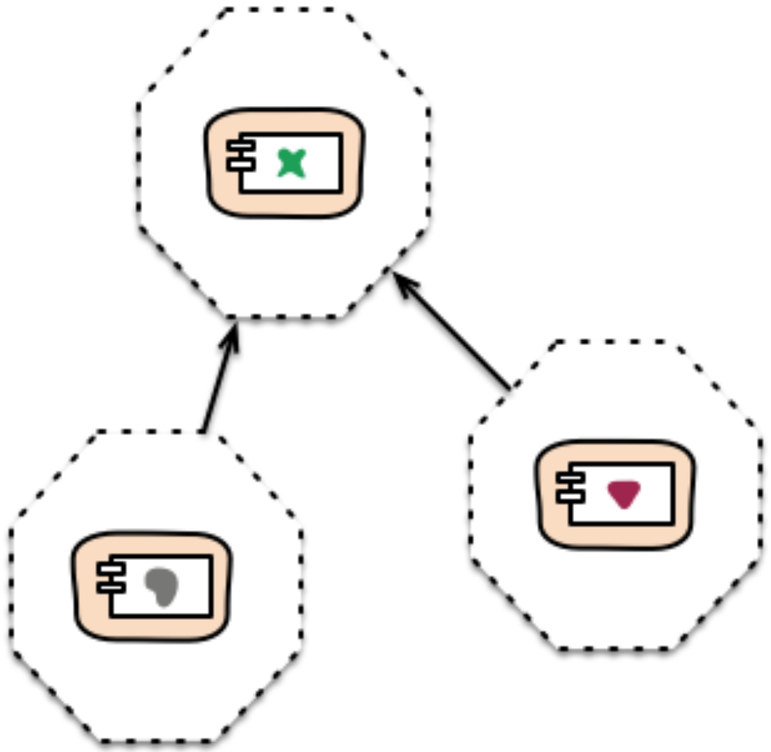
—Melvin Conway



Inverse Conway Maneuver



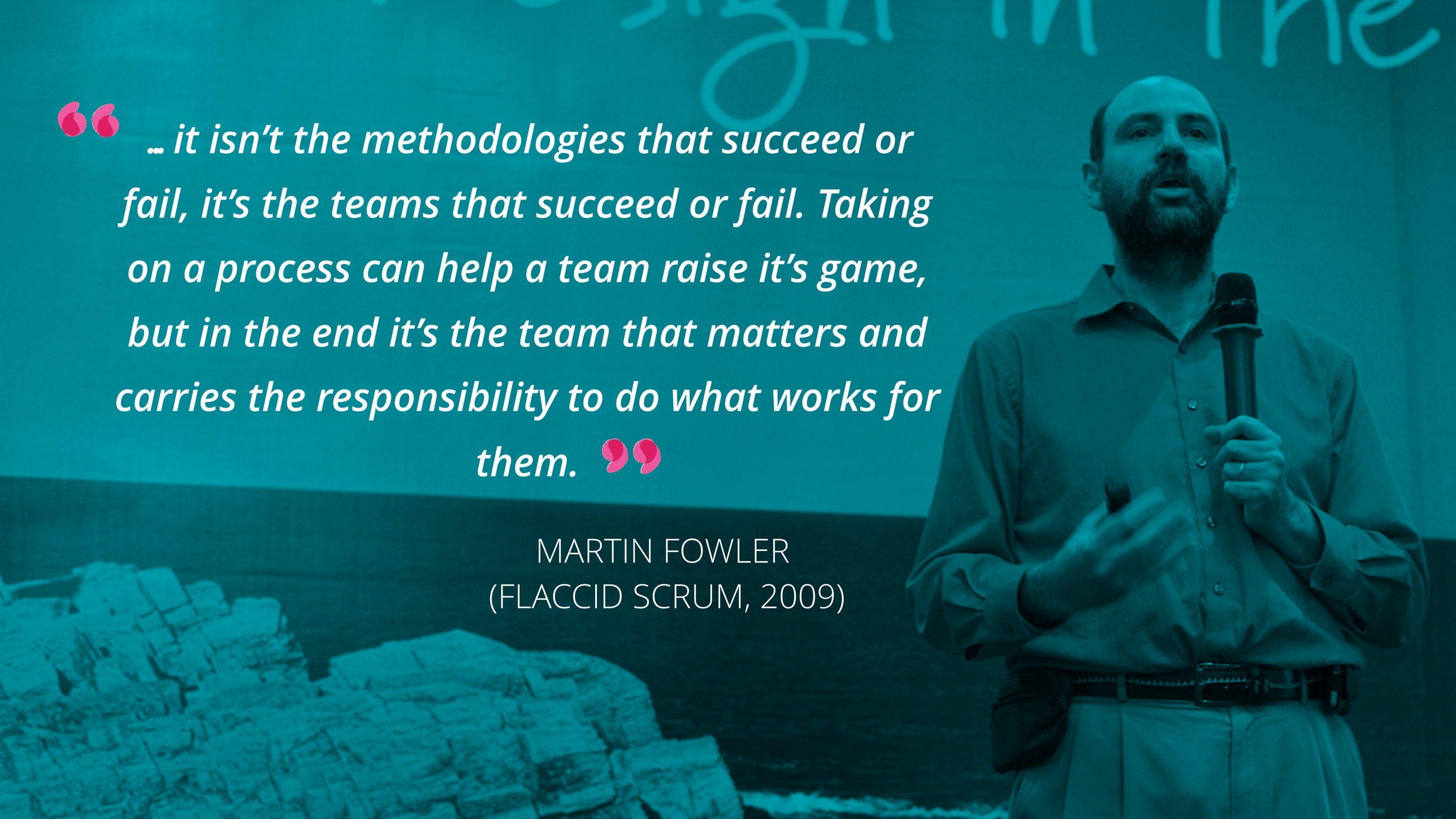
Cross-functional teams...



... organised around capabilities
Because Conway's Law

“team designs are the first draft of your architecture”

- Michael Nygard



“… it isn’t the methodologies that succeed or fail, it’s the teams that succeed or fail. Taking on a process can help a team raise it’s game, but in the end it’s the team that matters and carries the responsibility to do what works for them.”

MARTIN FOWLER
(FLACCID SCRUM, 2009)

Continuous Delivery

reduce friction

automate everything you can

incorporate everyone into Continuous Delivery practices

measure success via cycle time

continue to improve