



aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler



@neal4d
nealford.com

aRChiTeCtuRe

fOr

cONTiNuoUs DeLiVeRy

Continuous Delivery

integration as an engineering practice over time

integration as an engineering practice over time

1980

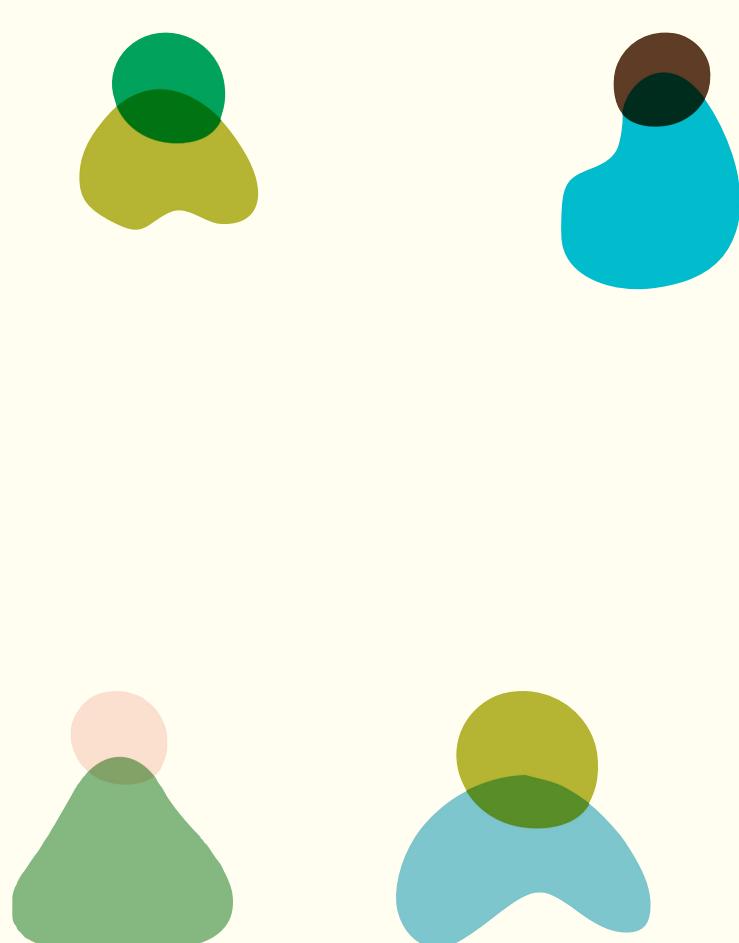
1990

2000

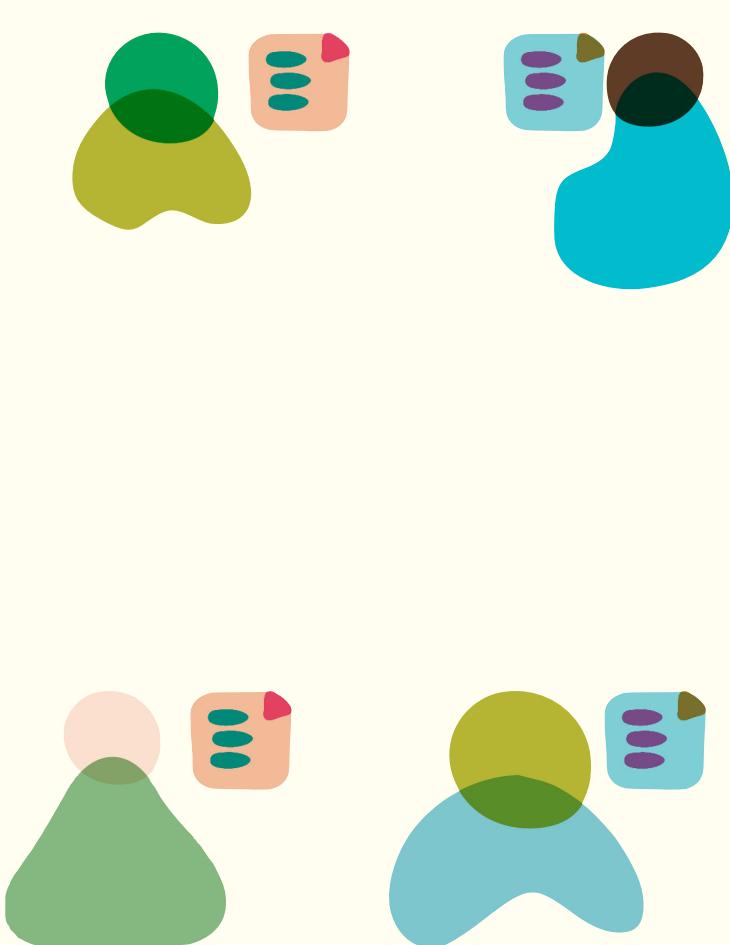
2010

current
day

1980

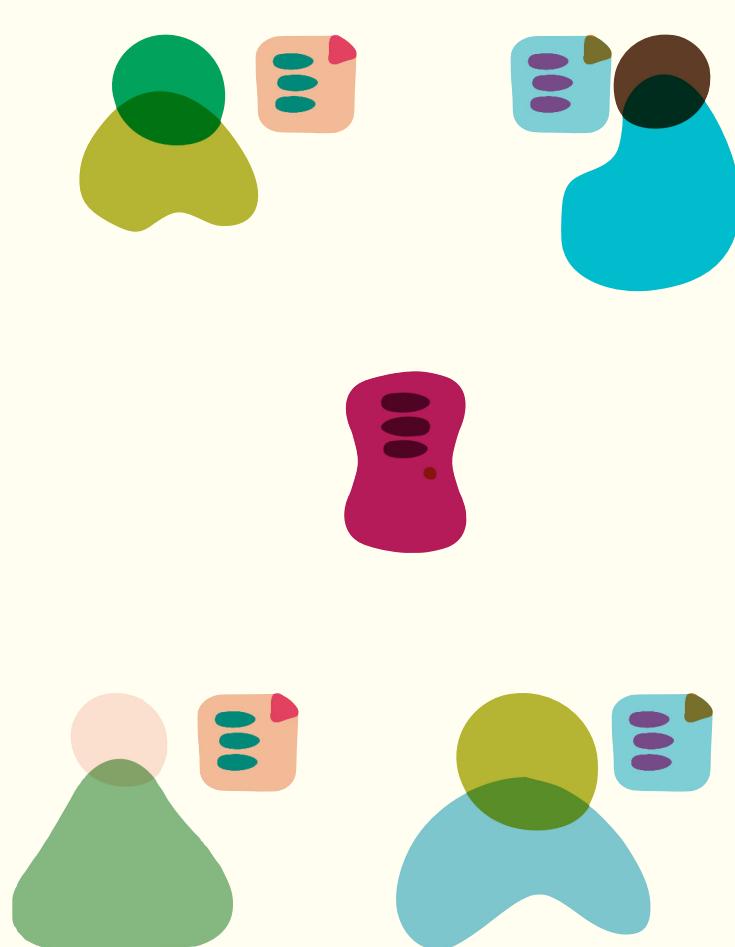


1980

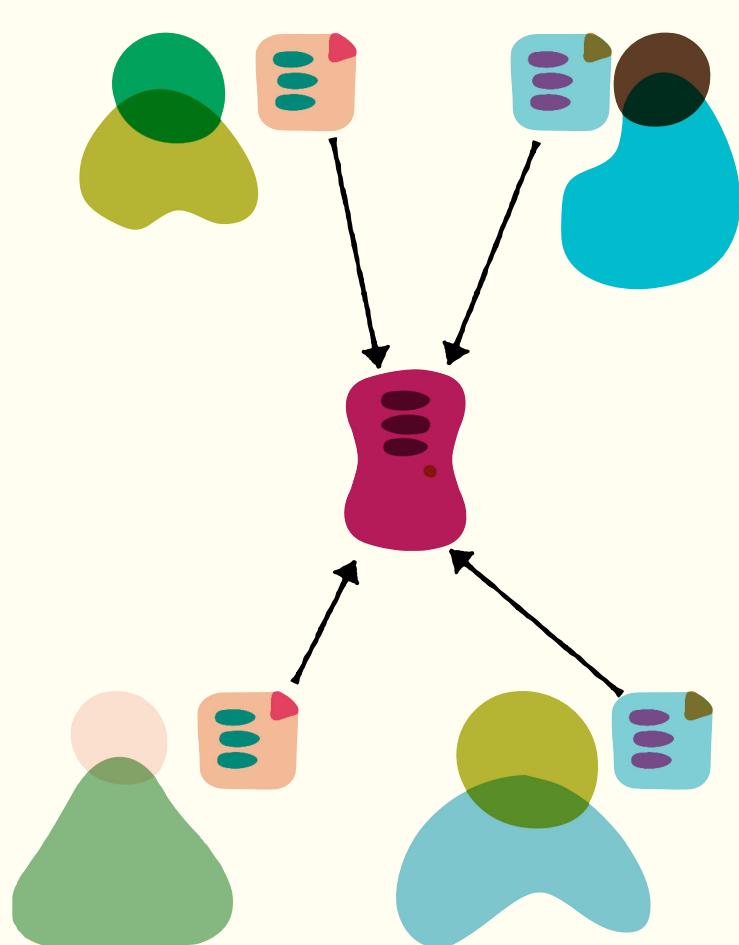


1980

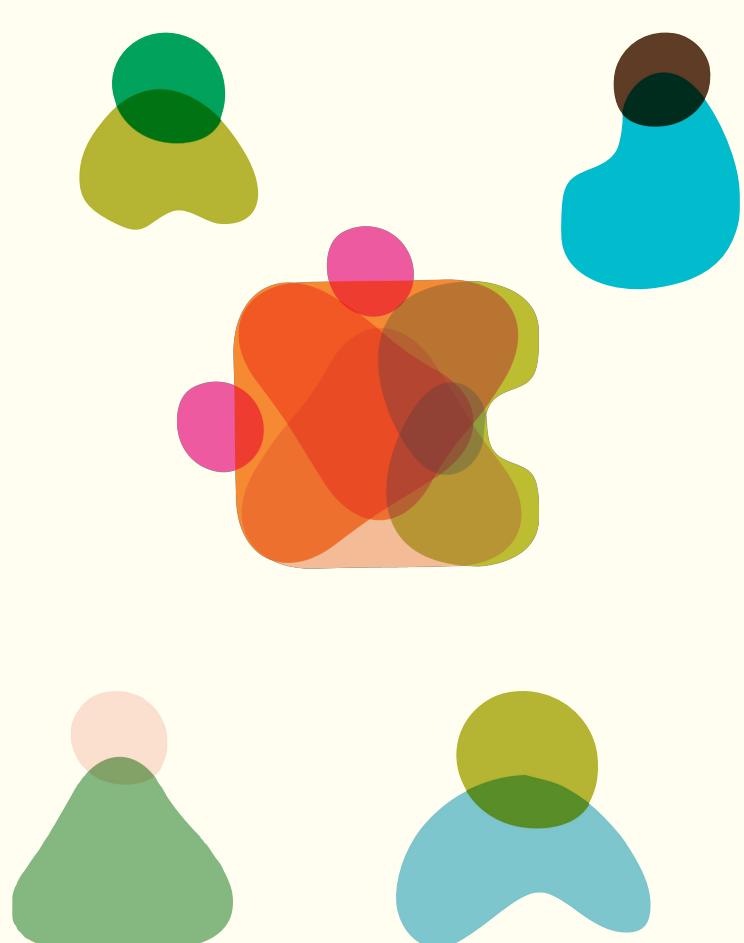
integration
phase



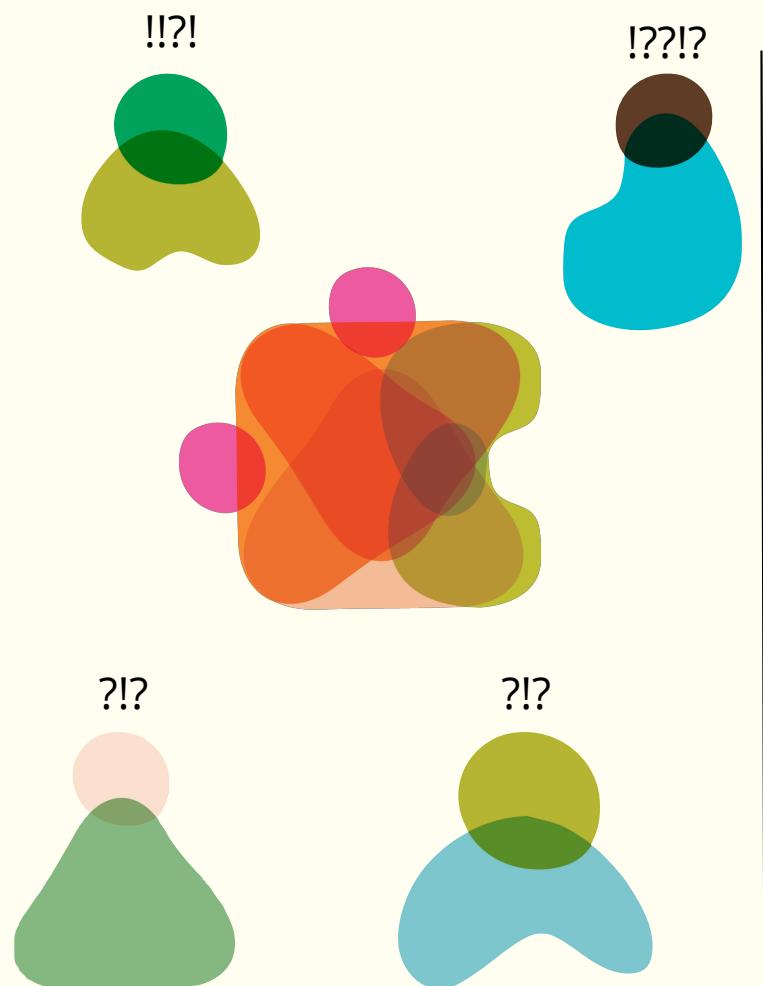
1980



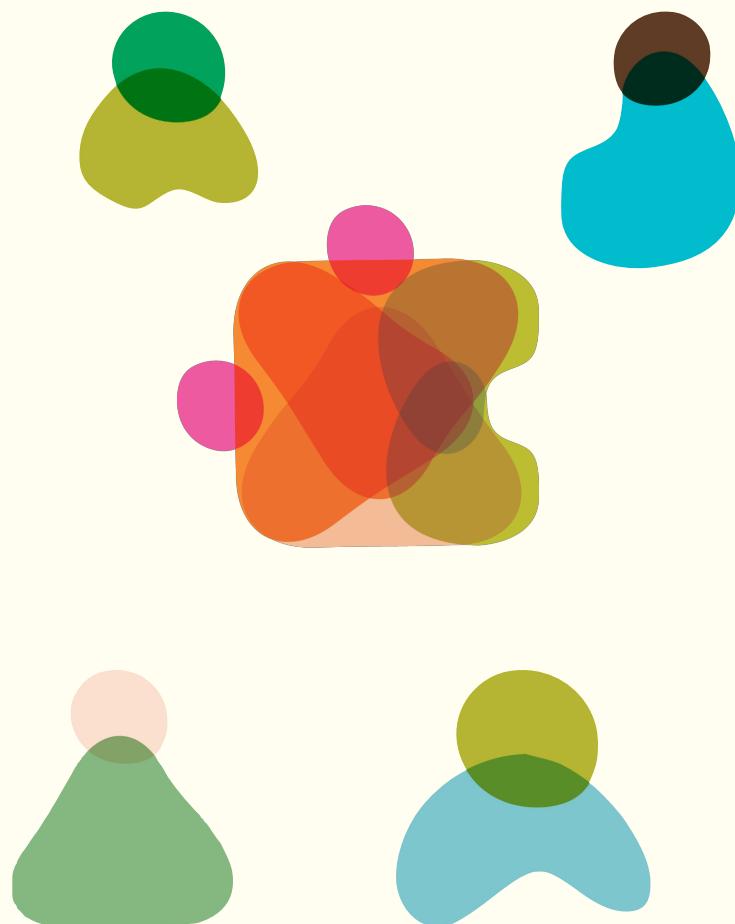
1980



1980

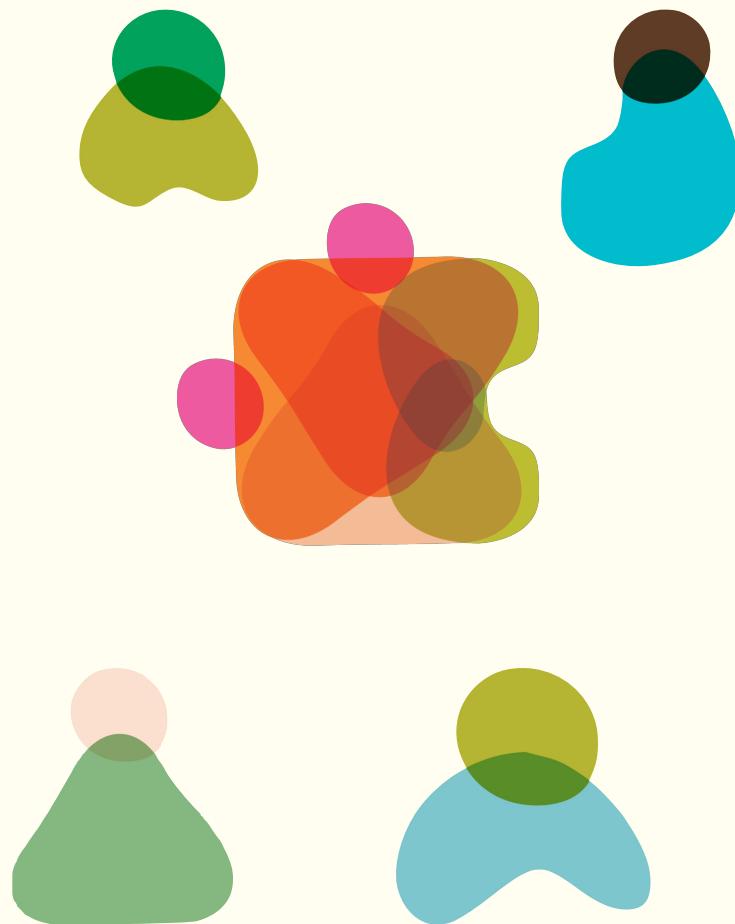


1980

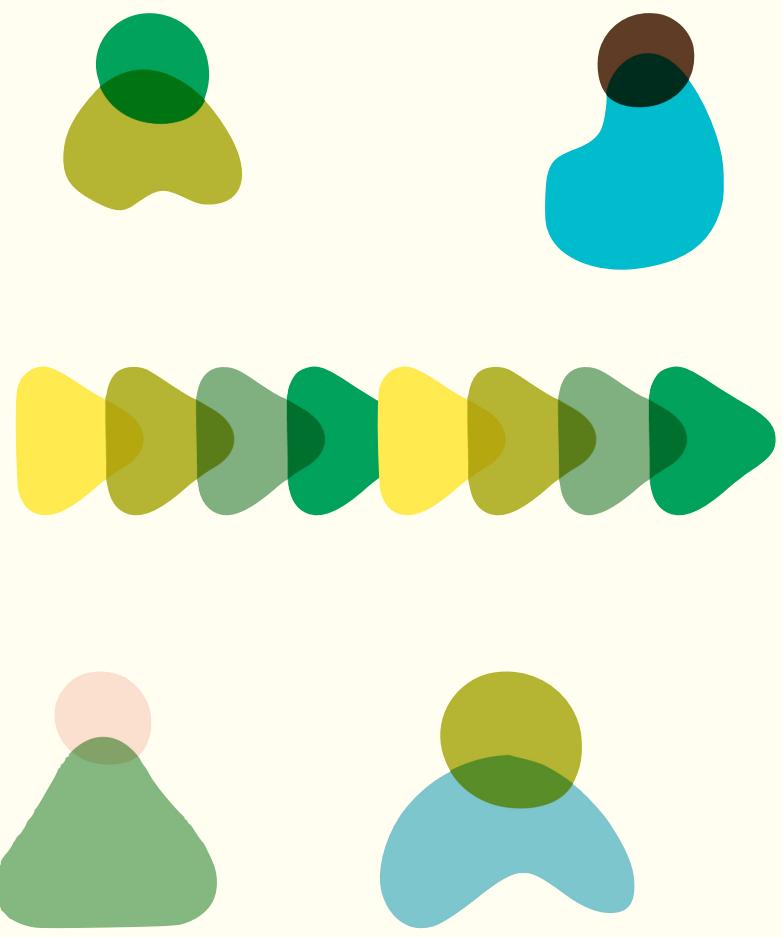


1990

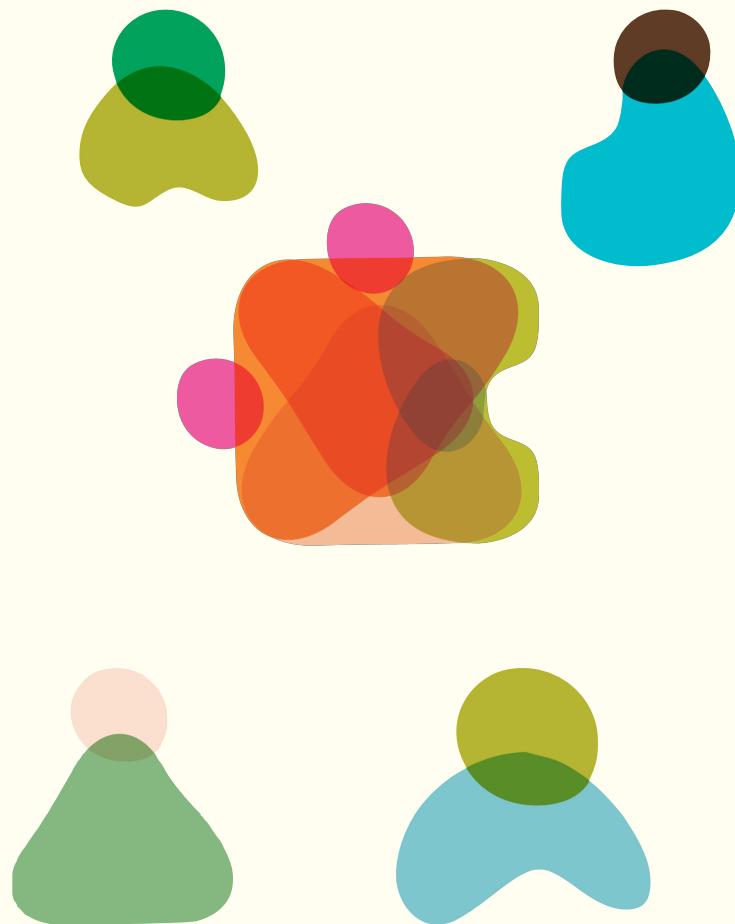
1980



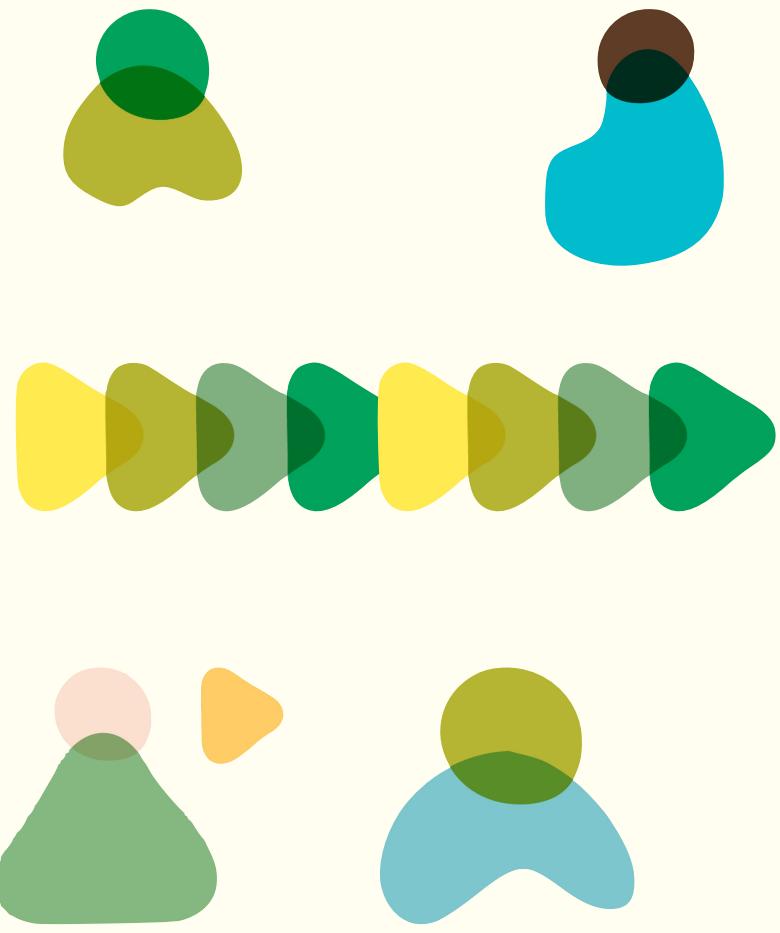
1990



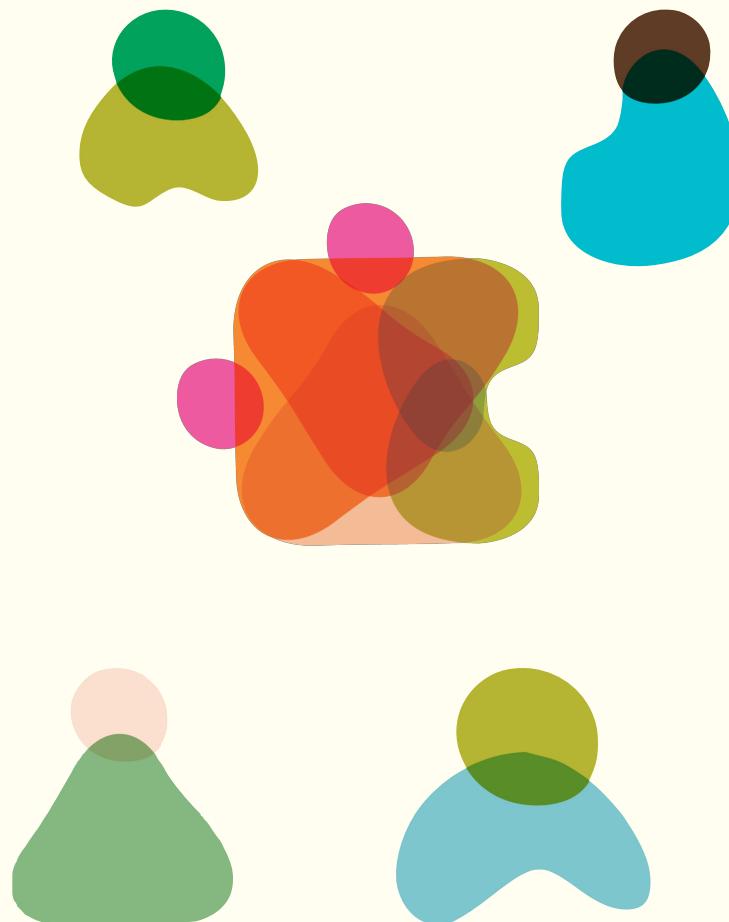
1980



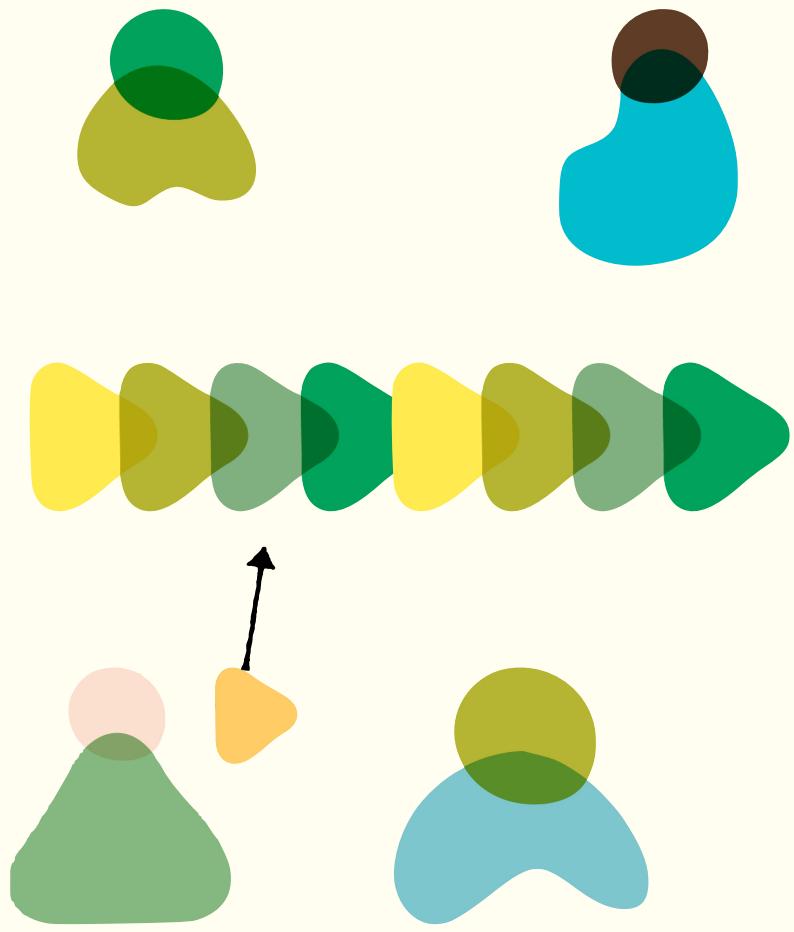
1990



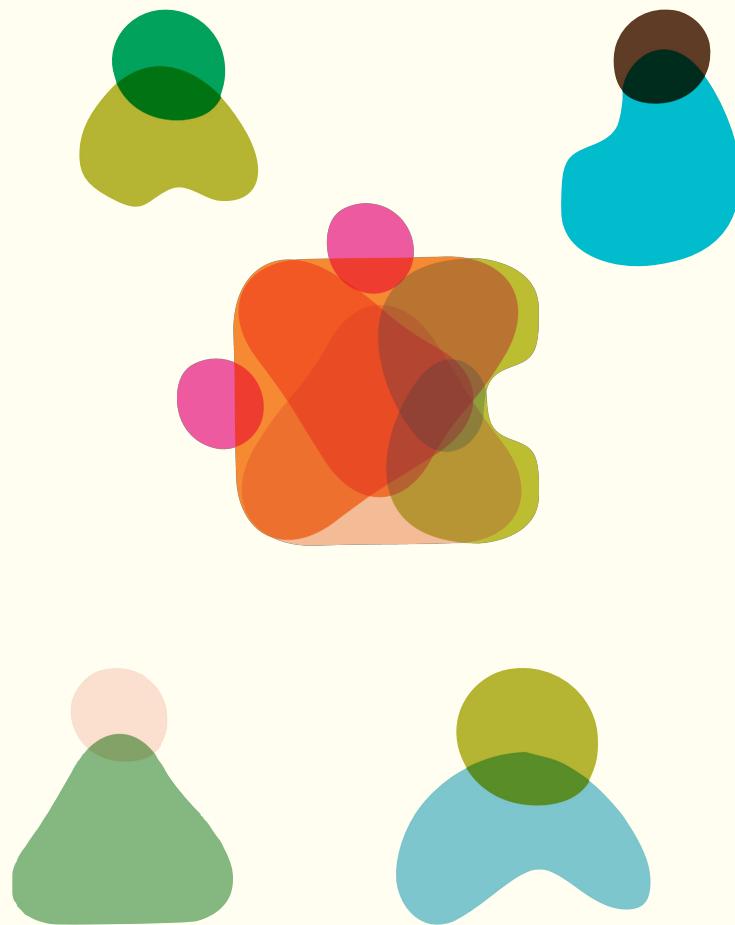
1980



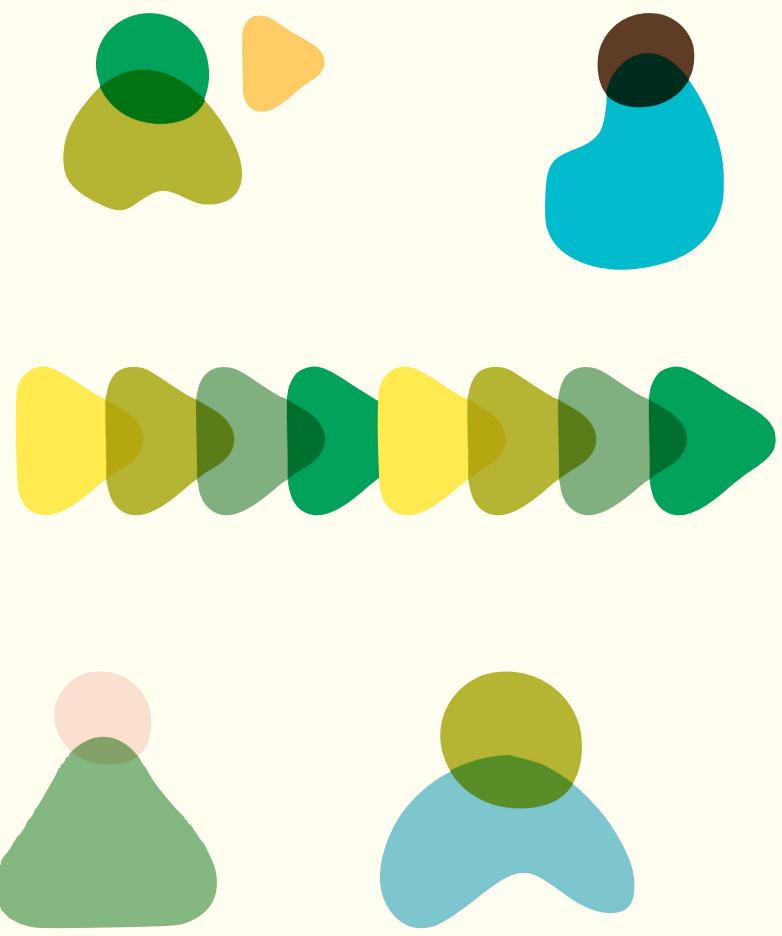
1990



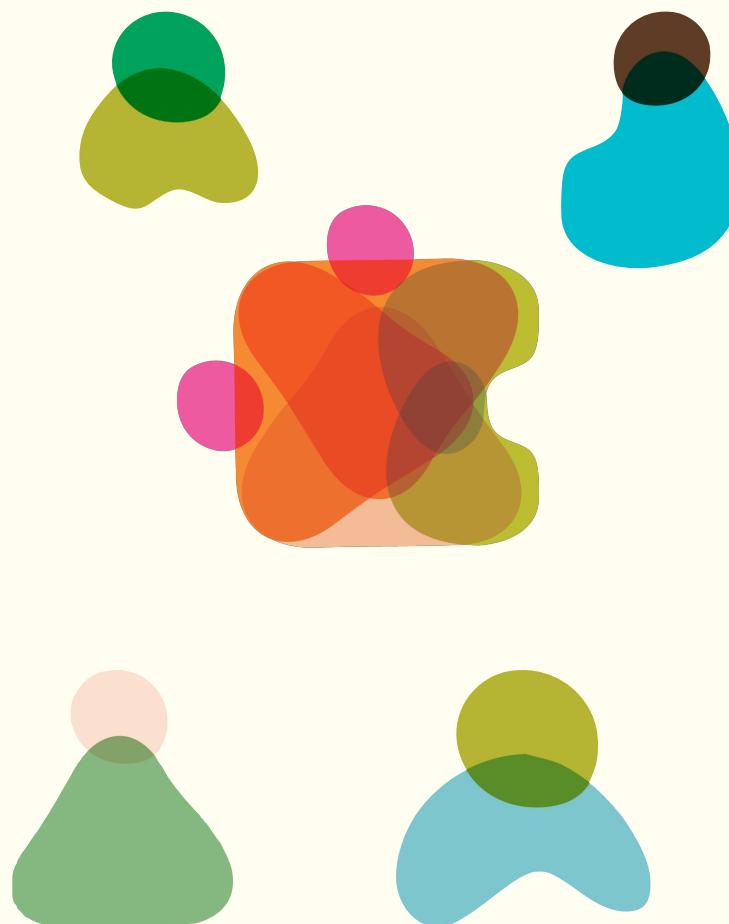
1980



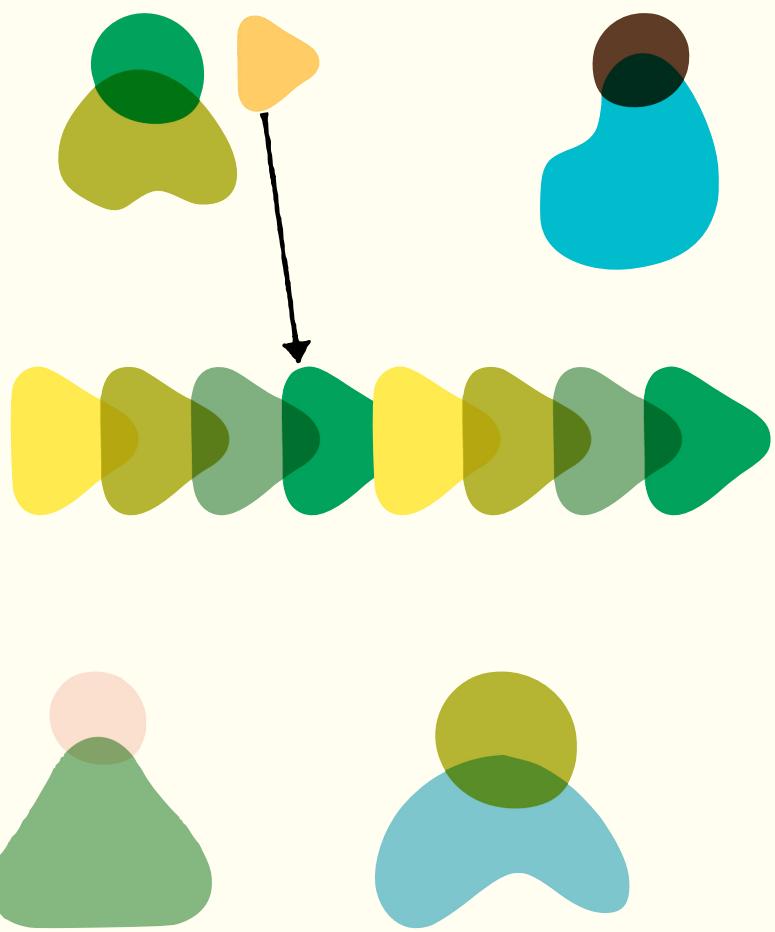
1990



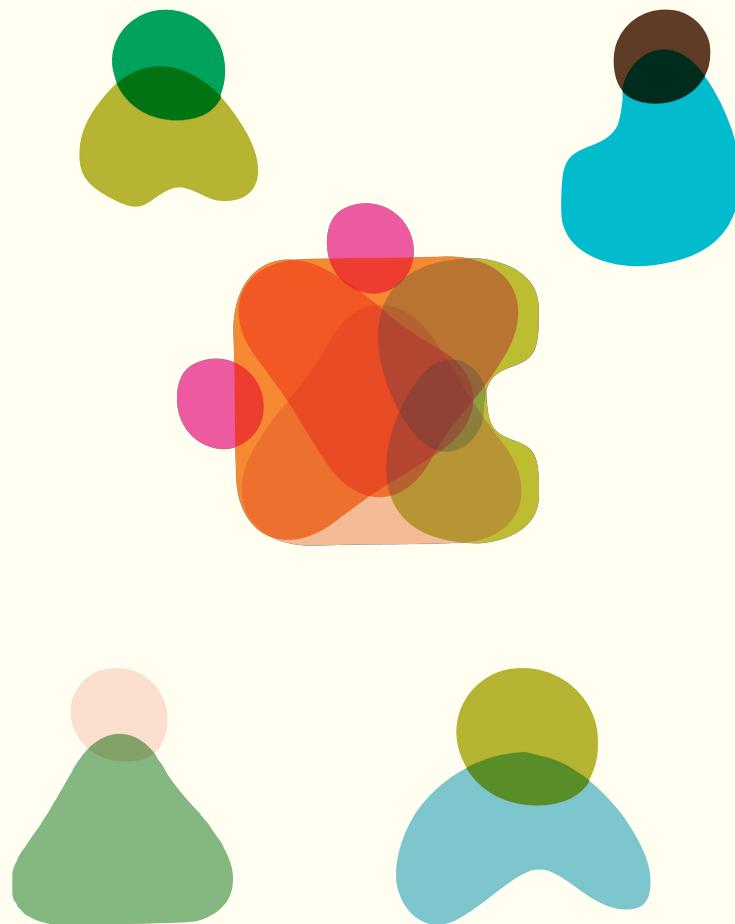
1980



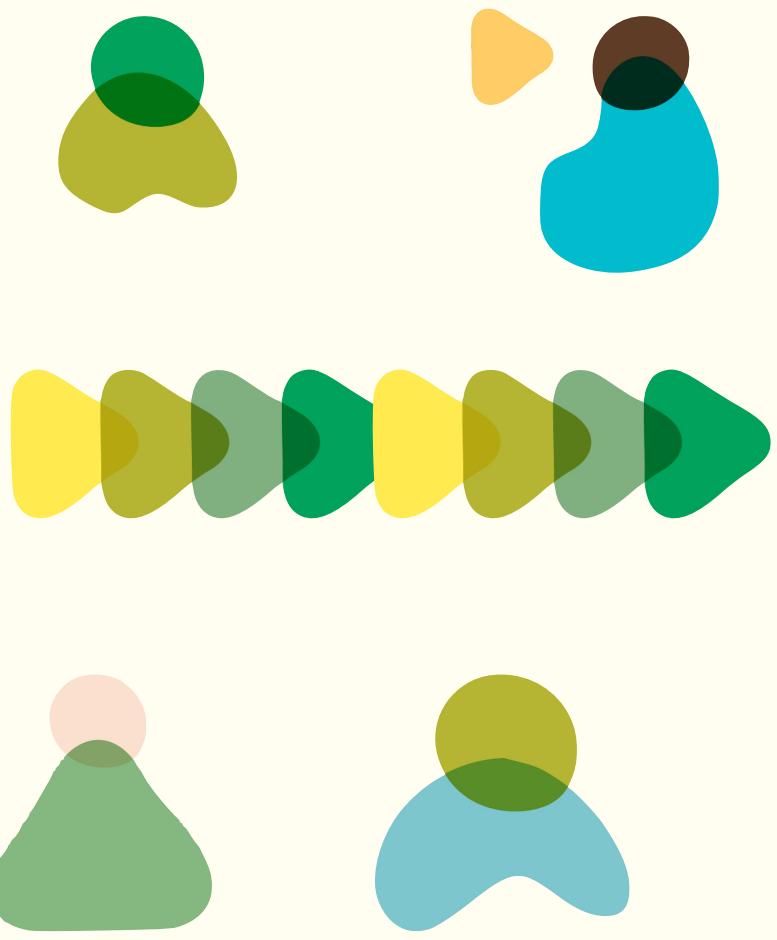
1990



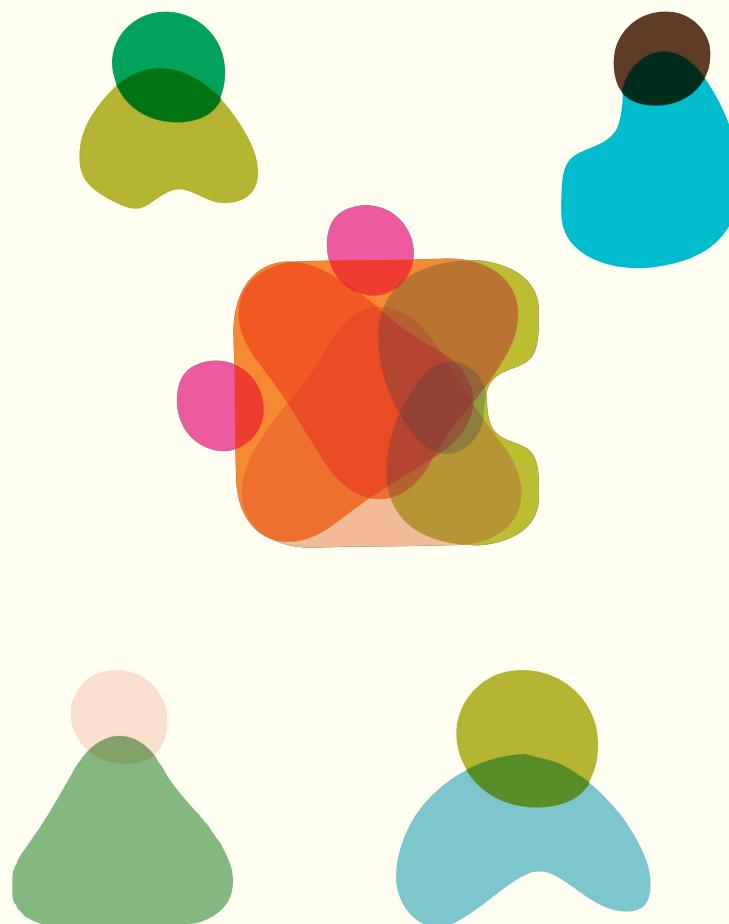
1980



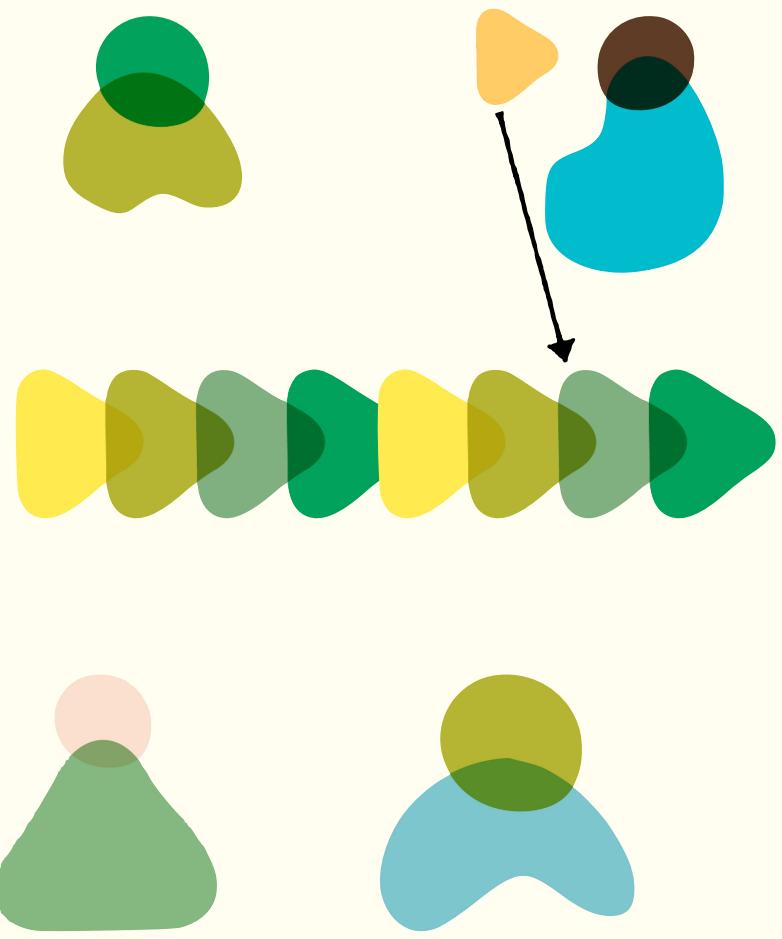
1990



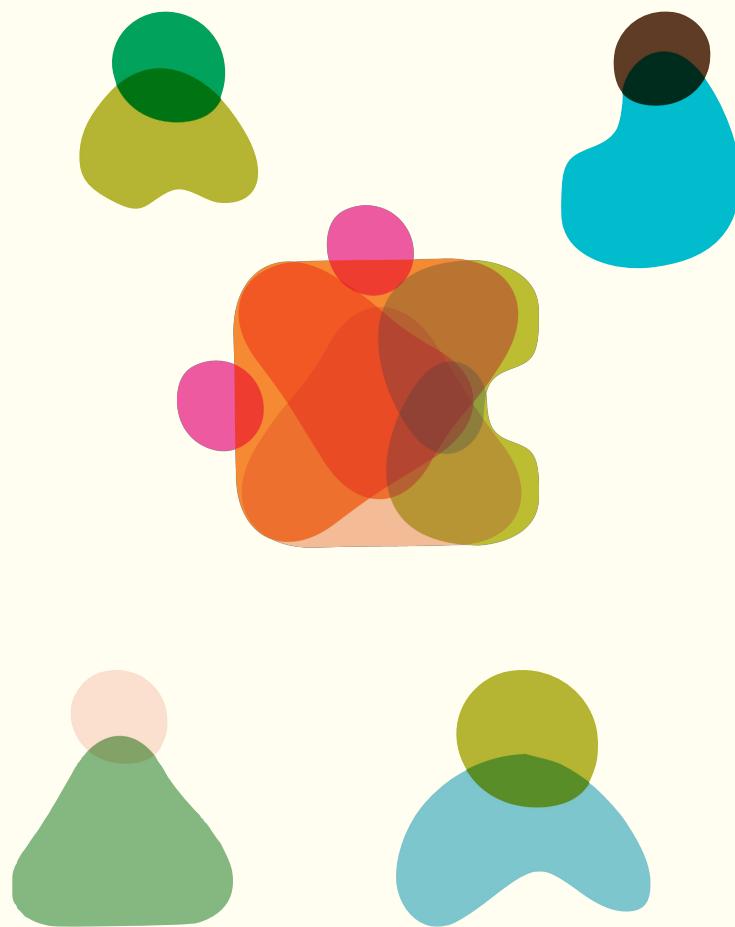
1980



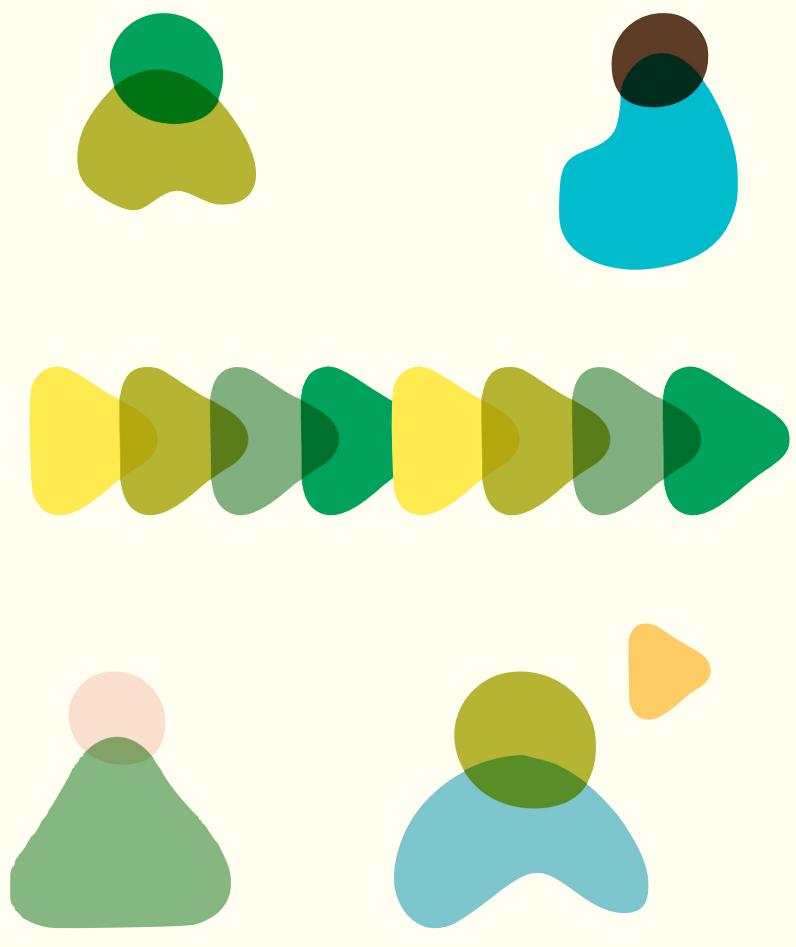
1990



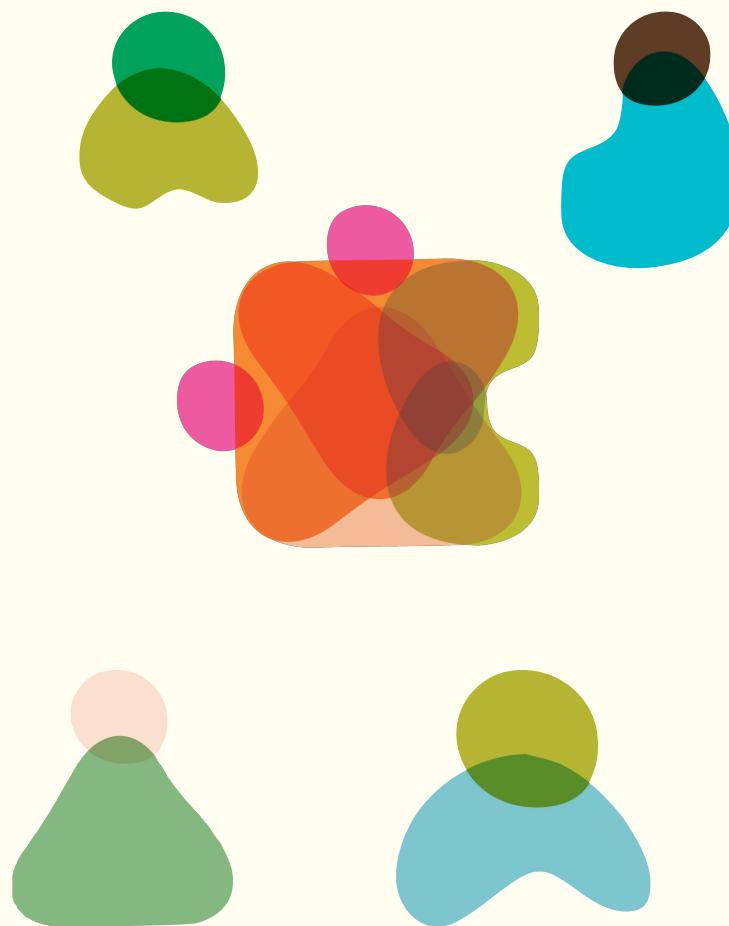
1980



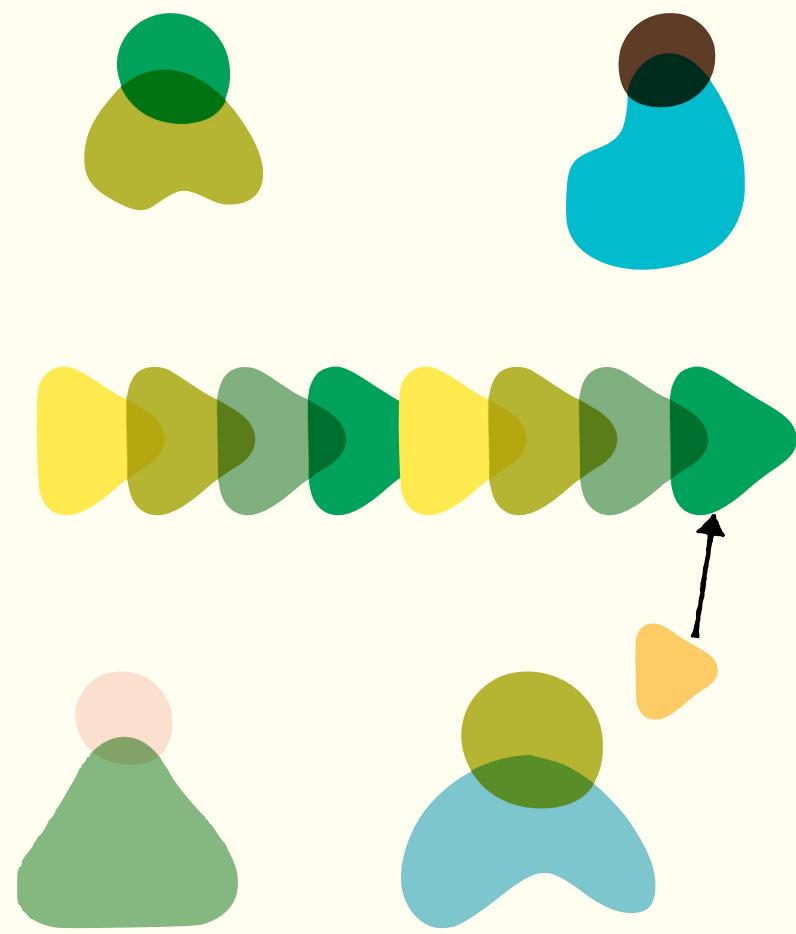
1990

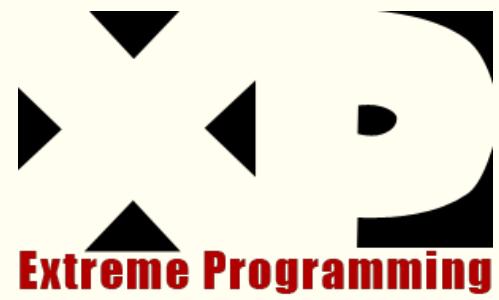


1980

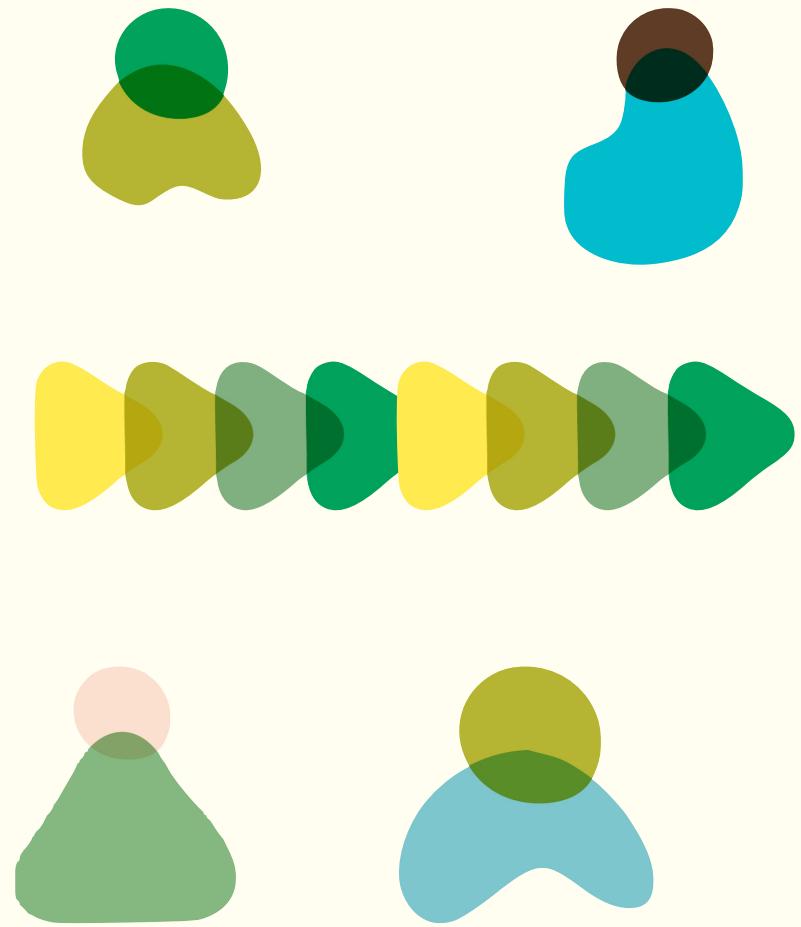


1990



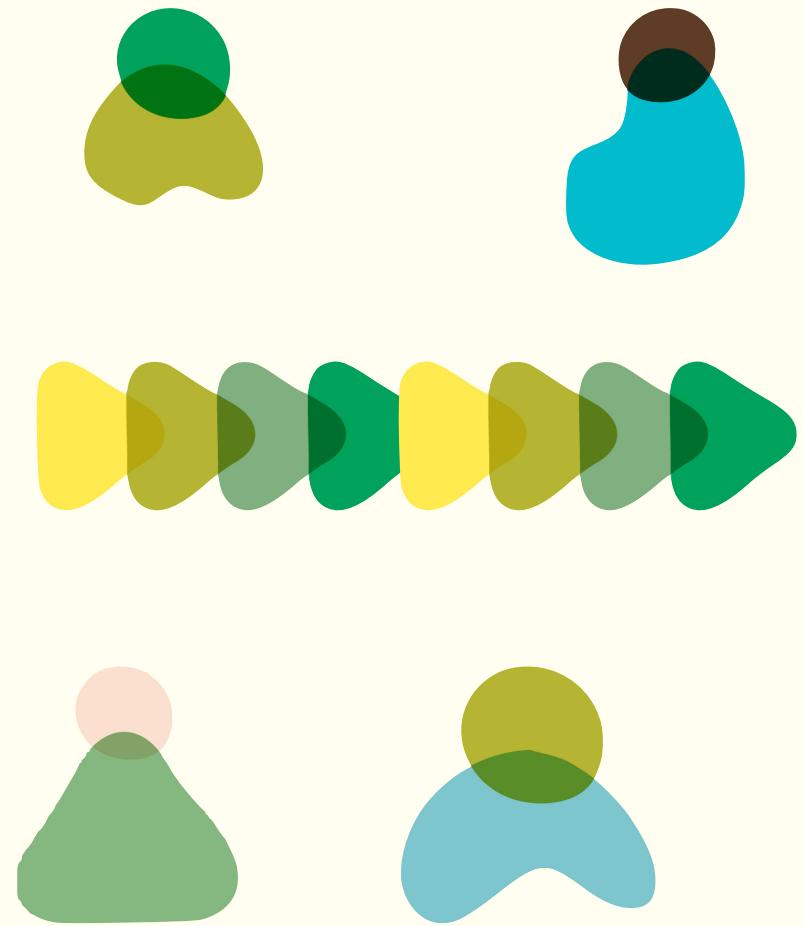


1990



continuous integration:

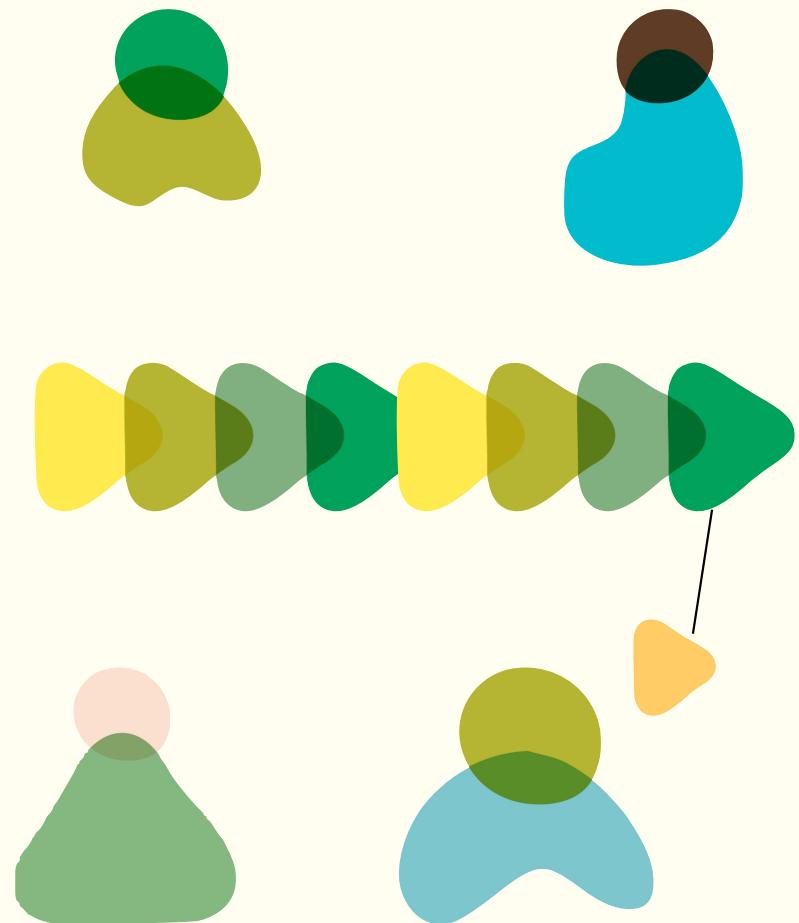
1990



continuous integration:

everyone commits to
trunk at least once per day

1990

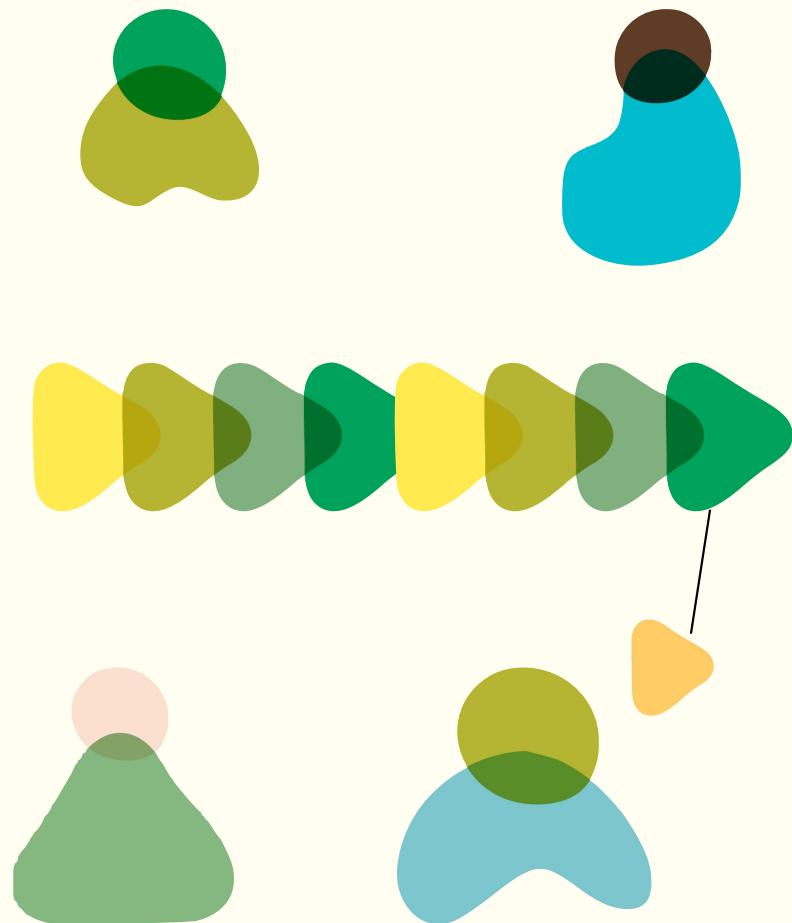


continuous integration:

everyone commits to
trunk at least once per day*

*if you do feature branches, you
aren't doing continuous integration

1990



integration as an engineering practice over time

1980

1990

2000

2010

current
day

integration as an engineering practice over time

*continuous
integration*

1980

1990

2000

2010

current
day

integration as an engineering practice over time

*continuous
integration*

1980

1990

2000

2010

current
day



*continuous
integration*



*continuous
integration*



canonical integration point

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

integration testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

integration testing

automated machine provisioning

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

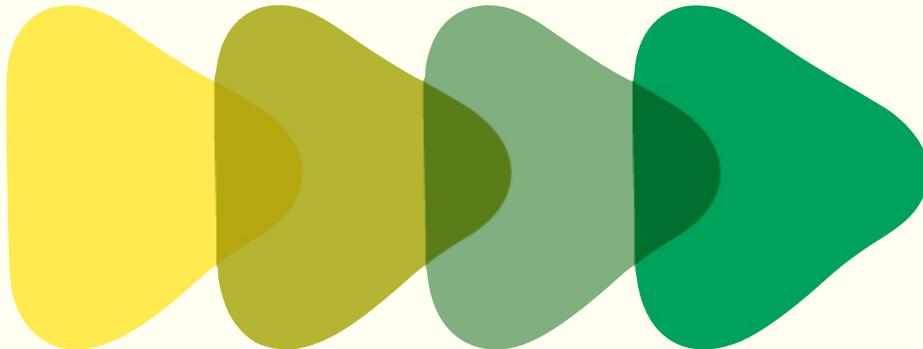
functional testing

integration testing

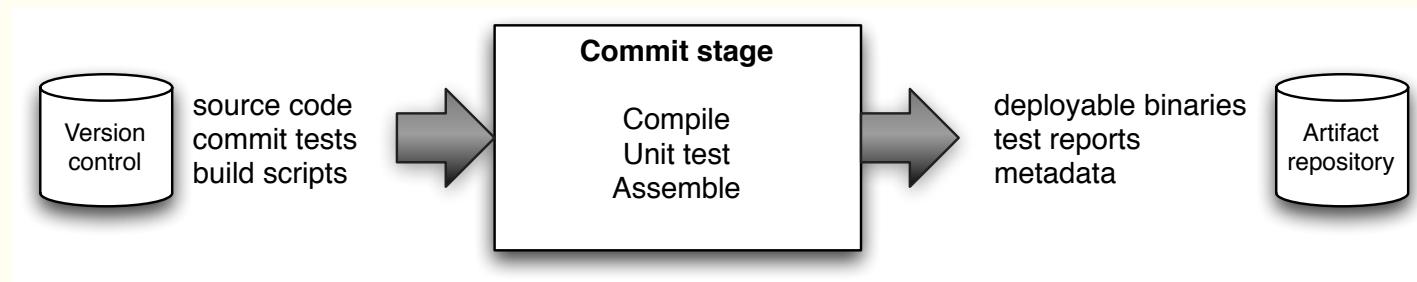
automated machine provisioning

deployment to production?

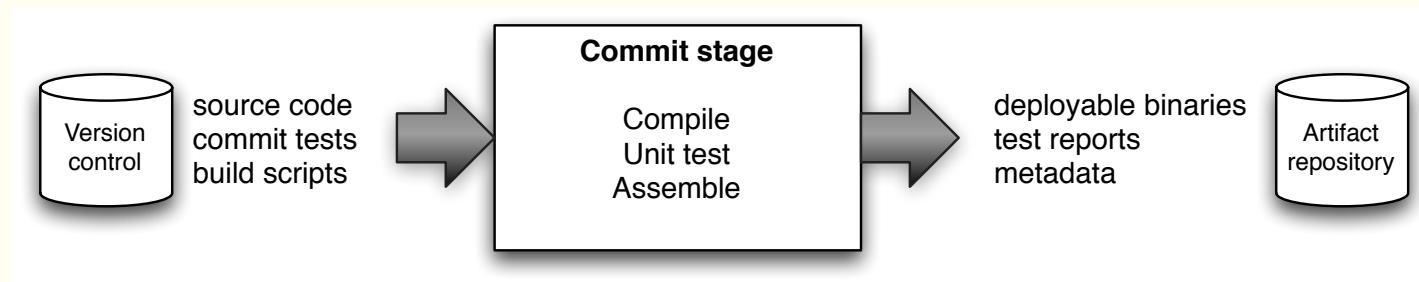
Deployment Pipelines



commit Stage

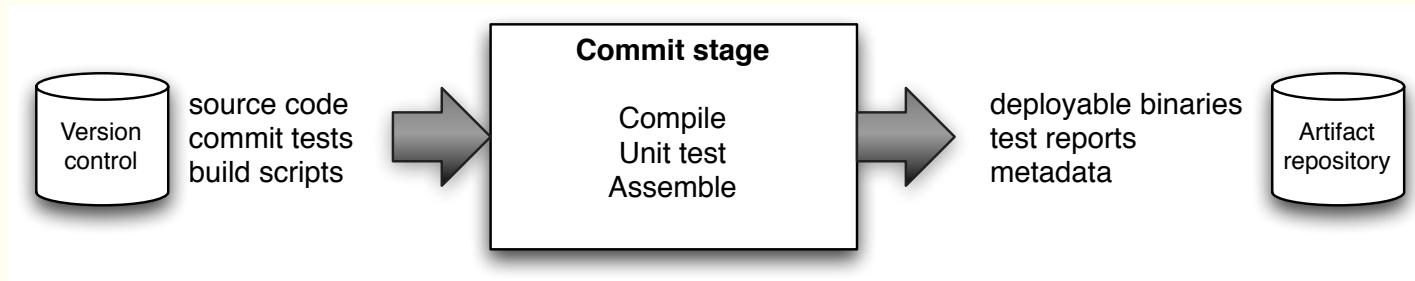


commit Stage



Run against each check-in

commit Stage

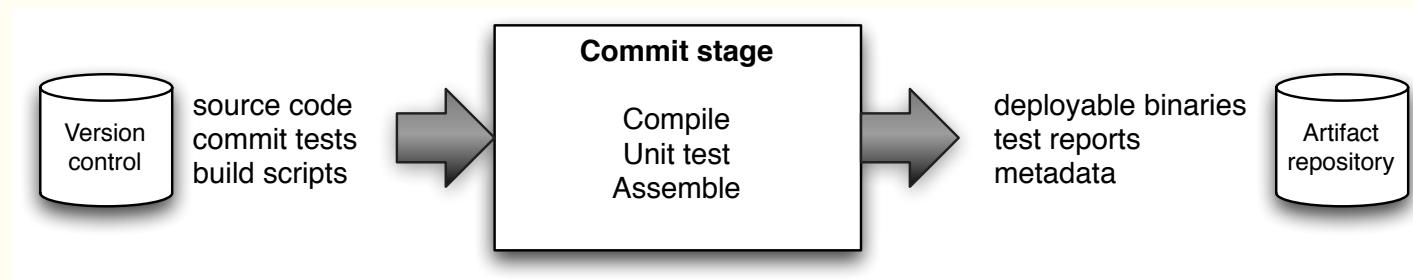


Run against each check-in



Starts building a release candidate

commit Stage



Run against each check-in

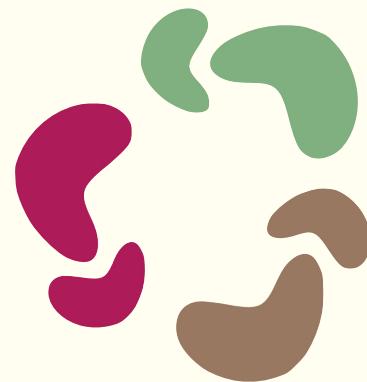
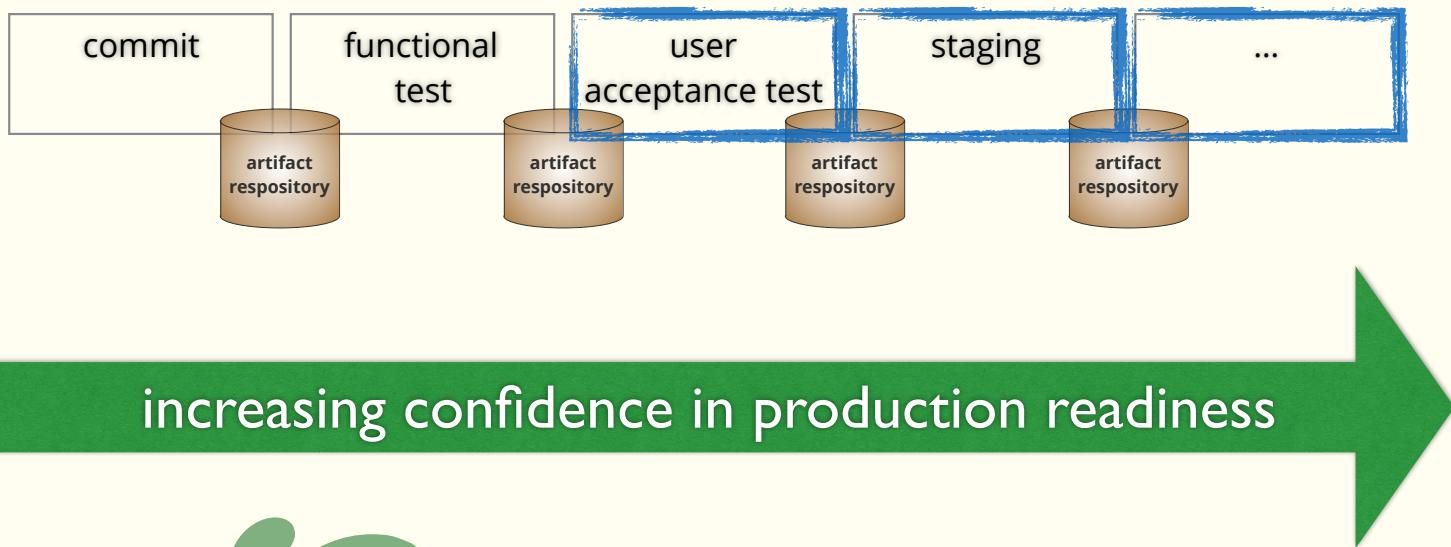


Starts building a release candidate

If it fails, fix it immediately

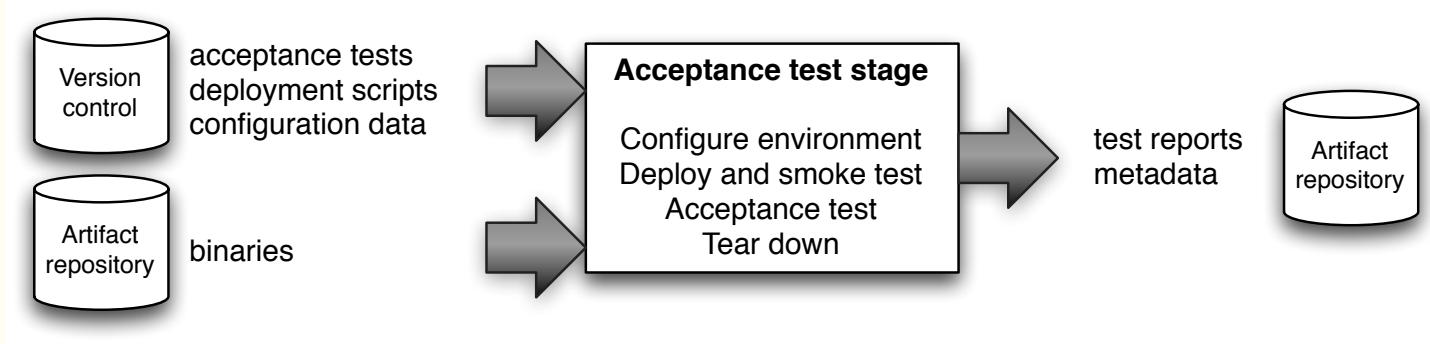


Pipeline Construction

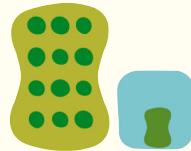
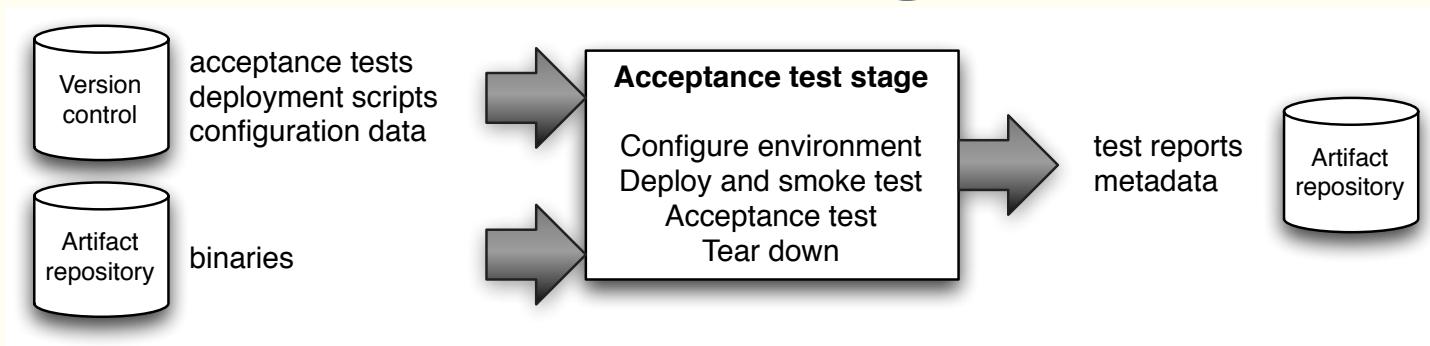


Pipeline stages = feedback opportunities

UAT Stage

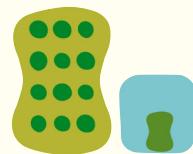
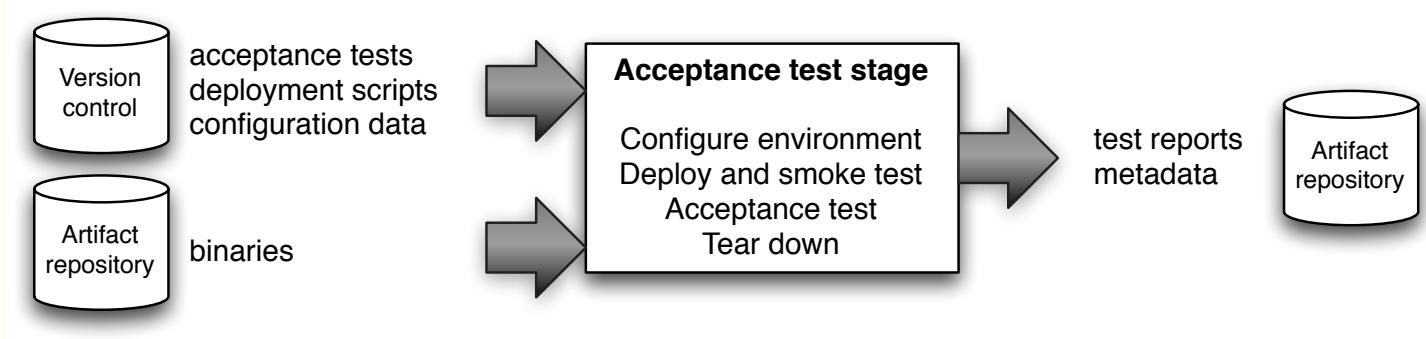


UAT Stage



End-to-end tests in production-like environment

UAT Stage

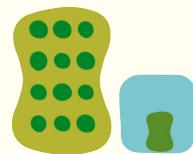
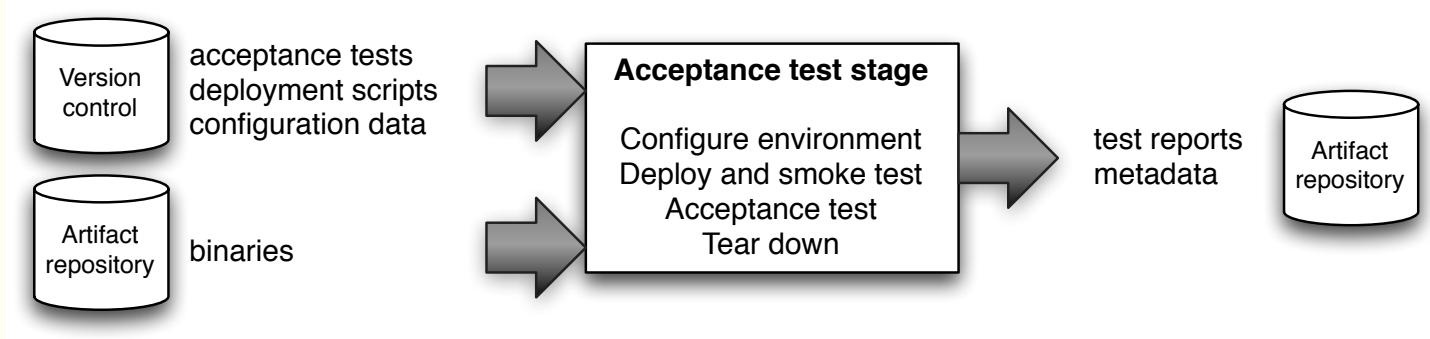


End-to-end tests in production-like environment



Triggered when upstream stage passes

UAT Stage



End-to-end tests in production-like environment

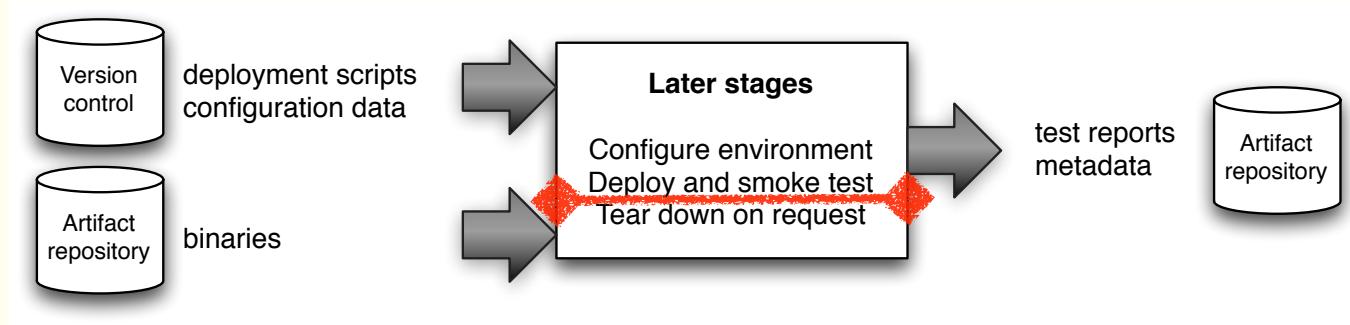


Triggered when upstream stage passes

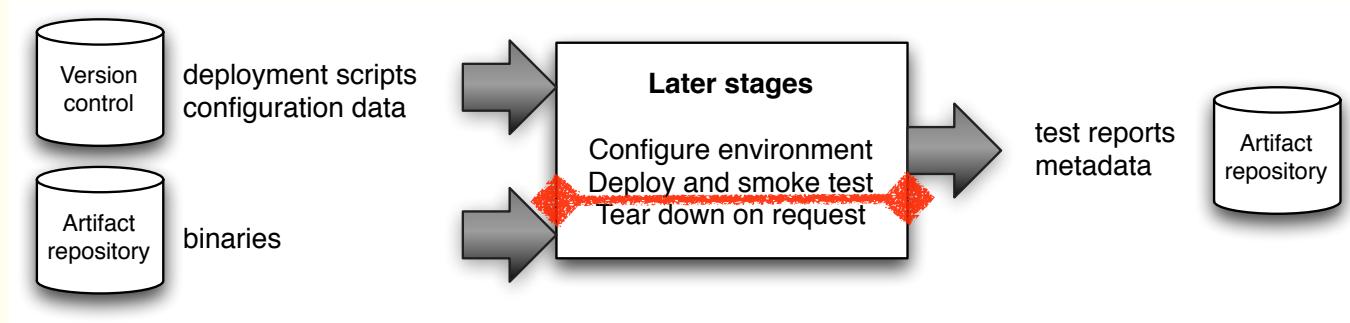


- First DevOps-centric build

Manual Stage

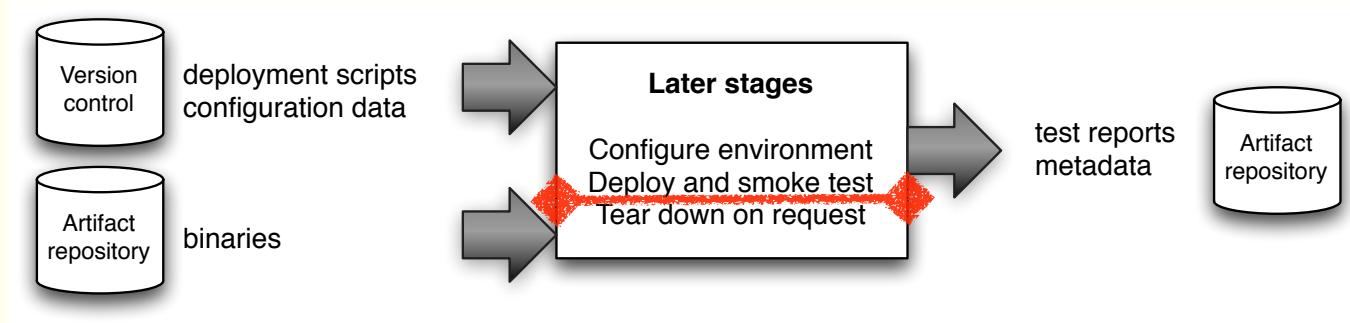


Manual Stage



UAT, staging, integration, production, ...

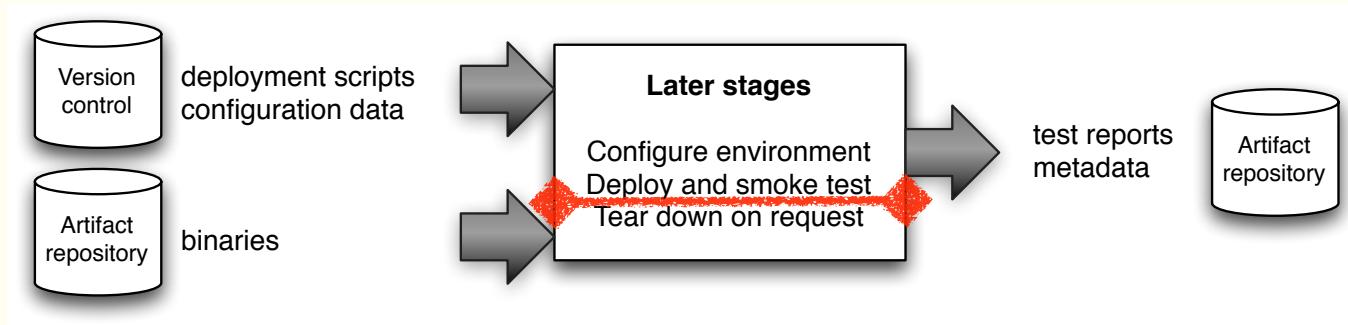
Manual Stage



 UAT, staging, integration, production, ...

 Push versus Pull model 

Manual Stage

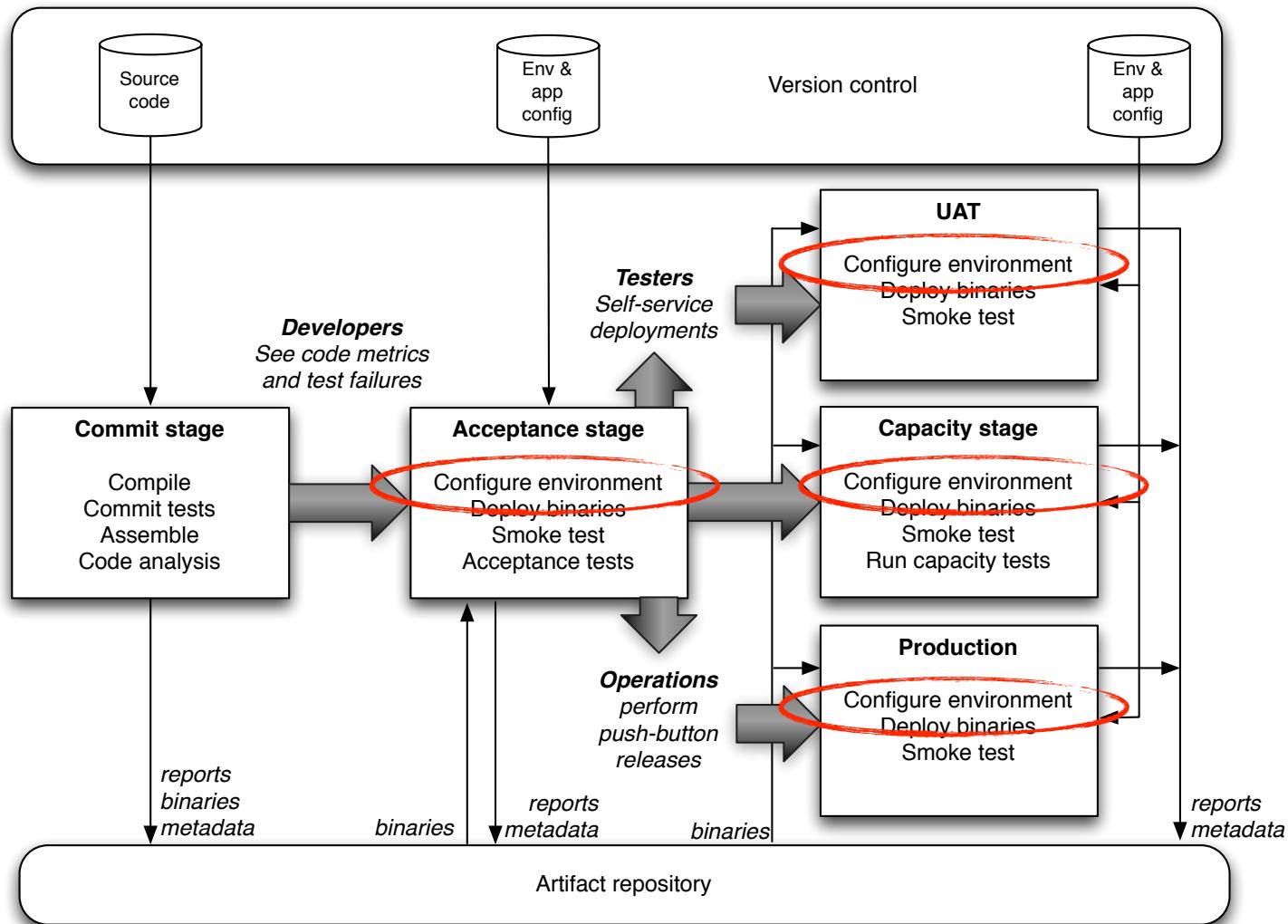


 UAT, staging, integration, production, ...

 Push versus Pull model 

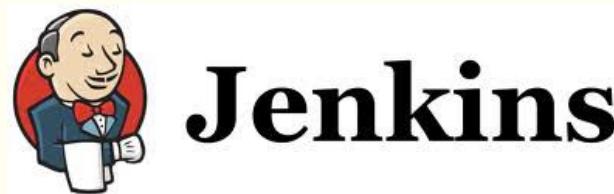
Deployments self-serviced through
push-button process



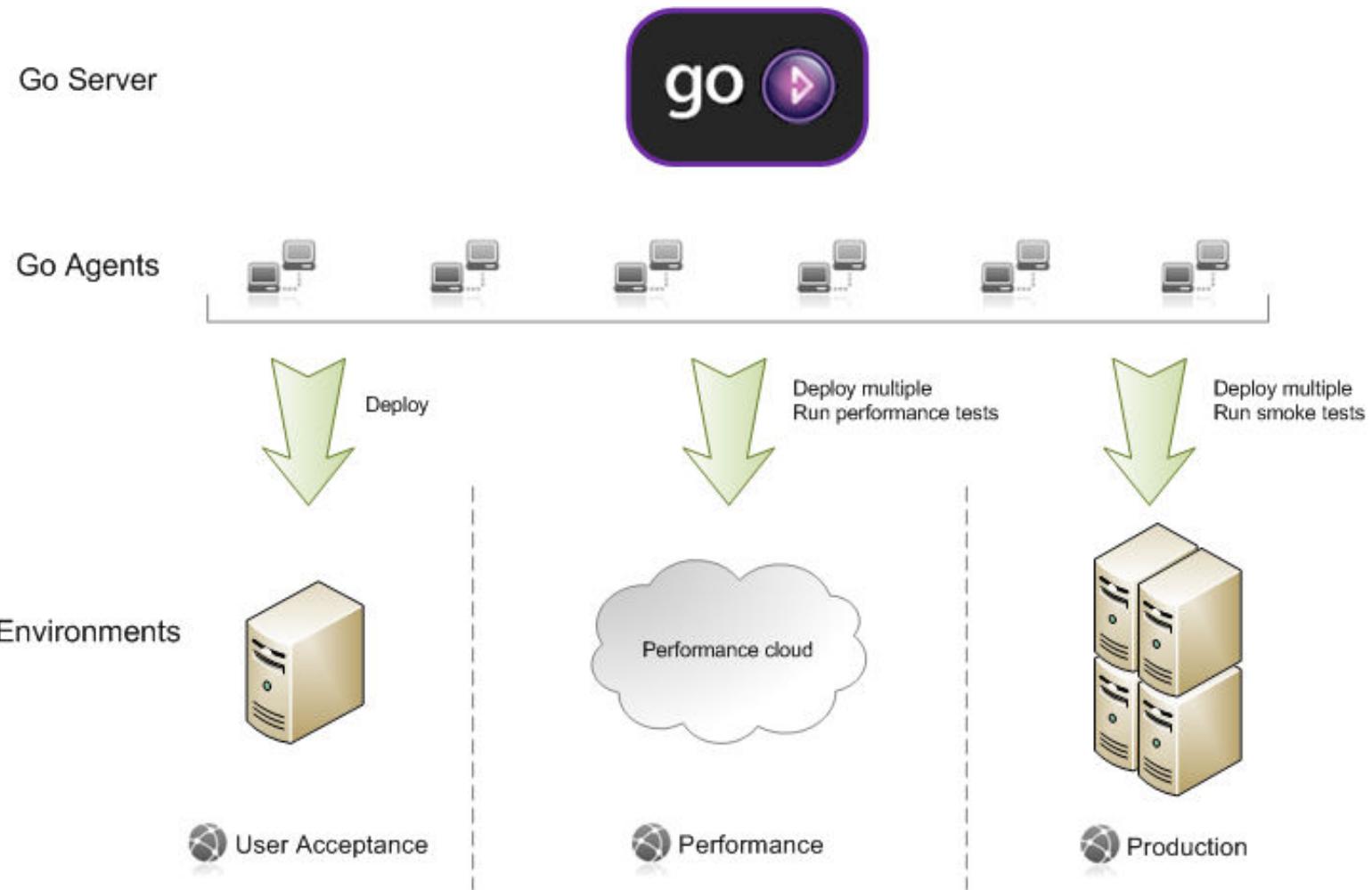


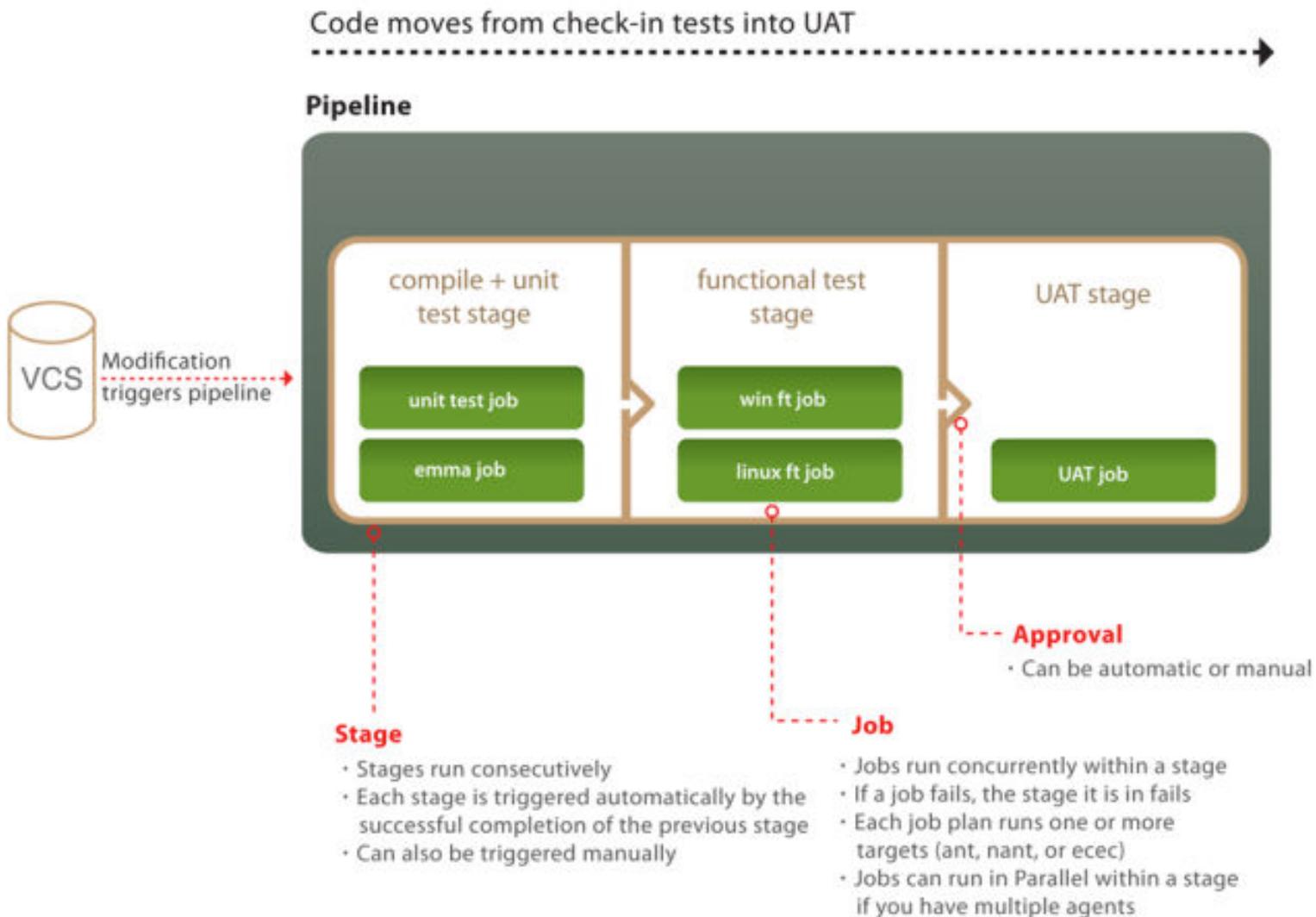
Machinery

continuous integration ++



<https://www.gocd.org>





Pipelines PERSONALIZE 13

MyApplication

my-app-web 2

Label: 39 3

Compare | Changes ▾

(Triggered by antony 14 days ago) 5

Passed: analyze

4

6 7 8

my-app-middleware

Label: 55

Compare | Changes ▾ 9

(Triggered by anushr 3 minutes ago)

Building: build

11

10 Previously: Passed

my-functional-tests

Label: 39-54

Compare | Changes ▾

(Triggered by changes 8 days ago)

Passed: functional-tests

12

▶ ⏪+ ⏸

▶ ⏪+ ⏸

▶ ⏪+ ⏸

The screenshot displays a CI pipeline interface with three main sections:

- acceptance** (Left Panel):
 - Label: 2.1.0.5447
 - CHANGES ▾ (Triggered by changes 44 minutes ago)
 - Failed: twist
- JOBS** (Top Right Panel):
 - Overview Pipeline Depen
 - JOBS Failed: 6
 - firefox-1 (red square)
 - firefox-2
 - firefox-3
 - firefox-4
 - firefox-5
 - firefox-7
- narayan-firefox/13/twist/1/firefox-1 job | failed** (Bottom Center Panel):
 - SCHEDULED ON: 2010-05-31T18:26:04+05:30
 - DURATION: 00:36:32
 - BUILD CAUSE: modified by narayan
 - COMPLETED ON: 2010-05-31T19:03:36+05:30 [more...](#)
 - AGENT: blrstdcrsuat02.thoughtworks.com (ip:10.4.8.2)

Failure Details:

 - Failure ArtifactUploadFetch.scn
 - Failure AgentsUIScreen.scn
 - Unit Test Failure and Error Details (2)
 - Test: ArtifactUploadFetch.scn
 - Type: Failure

Message: wait timed out after THREE_MINUTES for: Wait for pipeline: [pipeline-artifa]

JOBS HISTORY (Right Panel):

 - narayan-firefox/13/twist/1/firefox-1 29 days ago
 - narayan-firefox/12/twist/1/firefox-1 about 1 month ago
 - narayan-firefox/11/twist/1/firefox-1 about 1 month ago
 - narayan-firefox/10/twist/1/firefox-1 about 1 month ago
 - narayan-firefox/9/twist/3/firefox-1 about 1 month ago

Pipeline Activity

PAUSE

dev

dist

smoke-firefox

dist-all

dist-sol

smoke-ie

analysis

2.0.0.5125

revision: bdcf135f9bc0...
about 6 hours ago
modified by Jake & RRR &
JJ & PS & Yogi & Anush



2.0.0.5124

revision: 25fcfb492d54...
1 day ago
modified by ShilpaG &
Jake



Mercurial - trunk - https://ccepair:*****@fmbstdscm01.thoughtworks.com/go



ShilpaG & Jake

#4257 - reverting the confirmation popup
added for pipeline trigger in pipeline

25fcfb492d54b60b1cb383901d84ee4d470e5f6f

Git - twist - go @10.4.3.137:/repo/go_qa



unknown

<vgarg@corporate.thoughtworks.com>

Added one more fail check for UAT
upgrades.

14bb9f3fd5d5f404929d9f4dc73ef589f0ef1911



smoke-ie

analysis



smoke-le

analysis

2.0.0.5122

revision: 2b006920224b...
1 day ago
modified by ShilpaG &
Jake



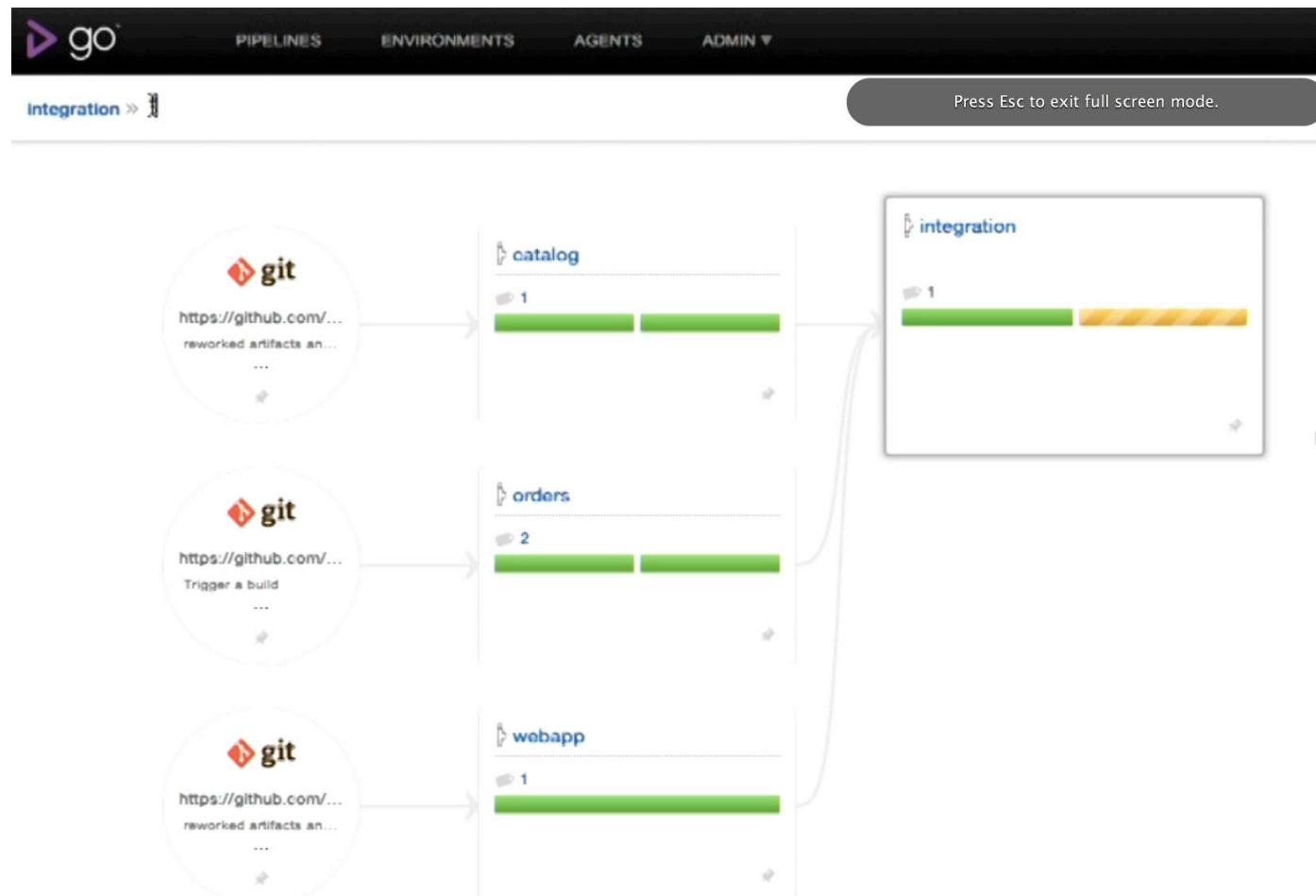
2.0.0.5121

revision: 15a21097f8d6...
4 days ago
modified by Yogi, PS

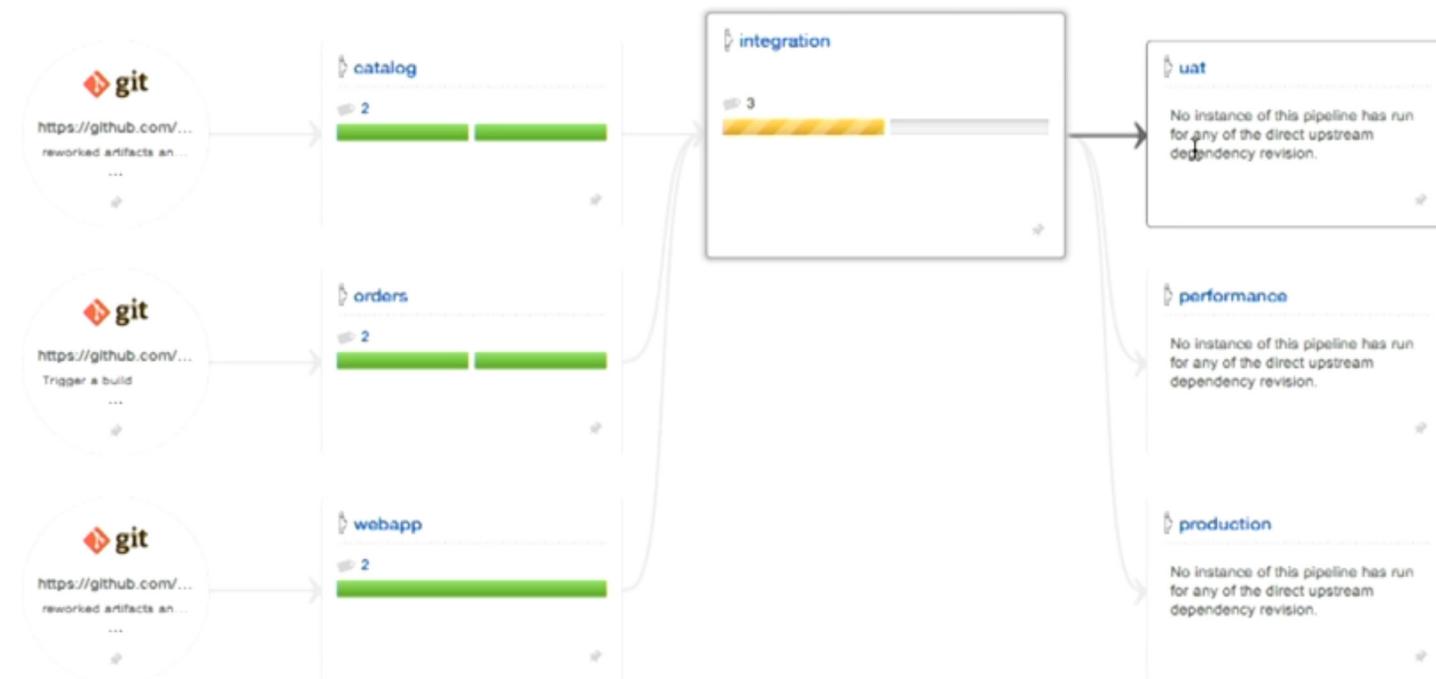


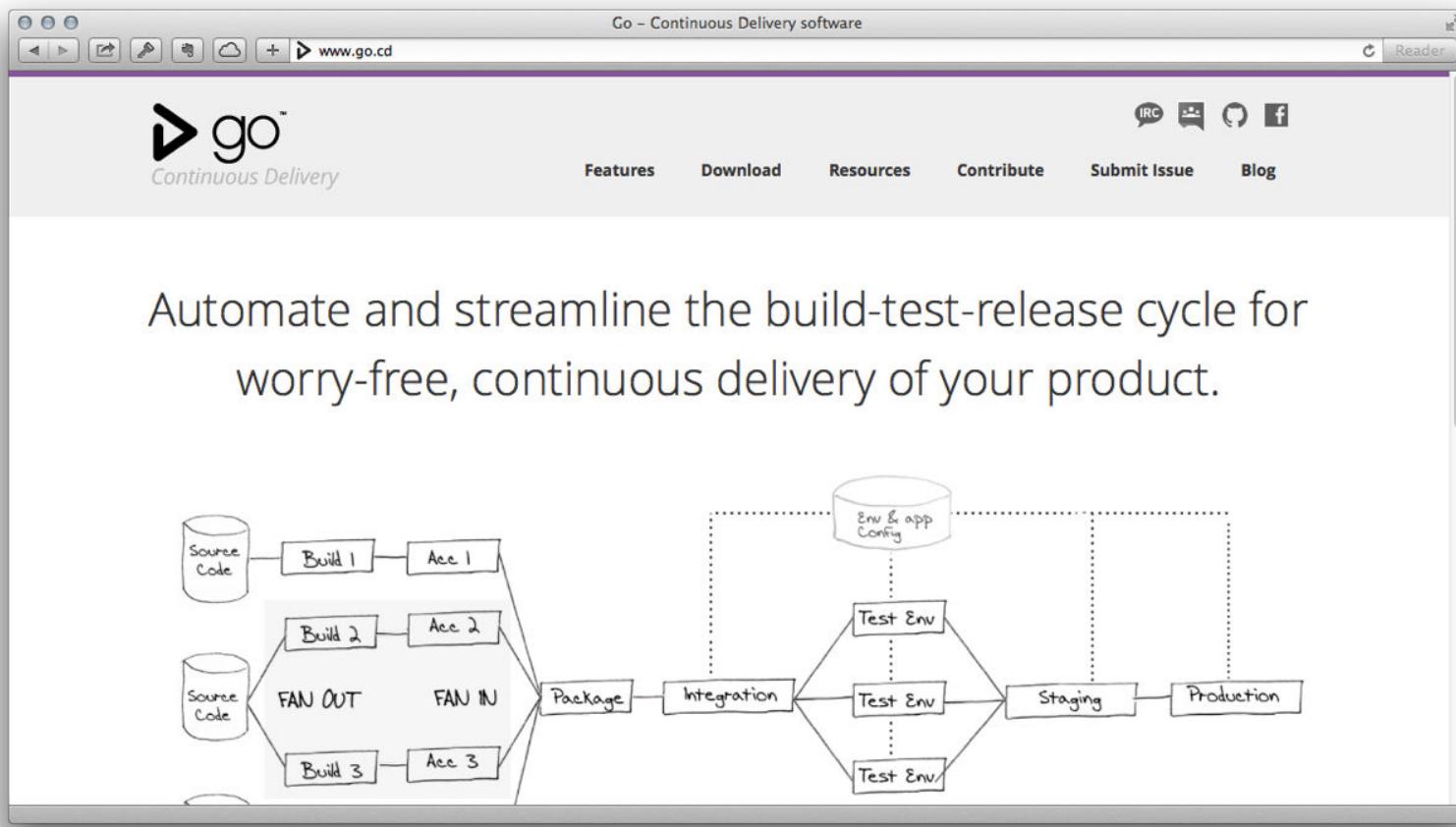
See JIRA for details
See JIRA for details
See JIRA for details
See JIRA for details

Integration Pipeline in Go CD

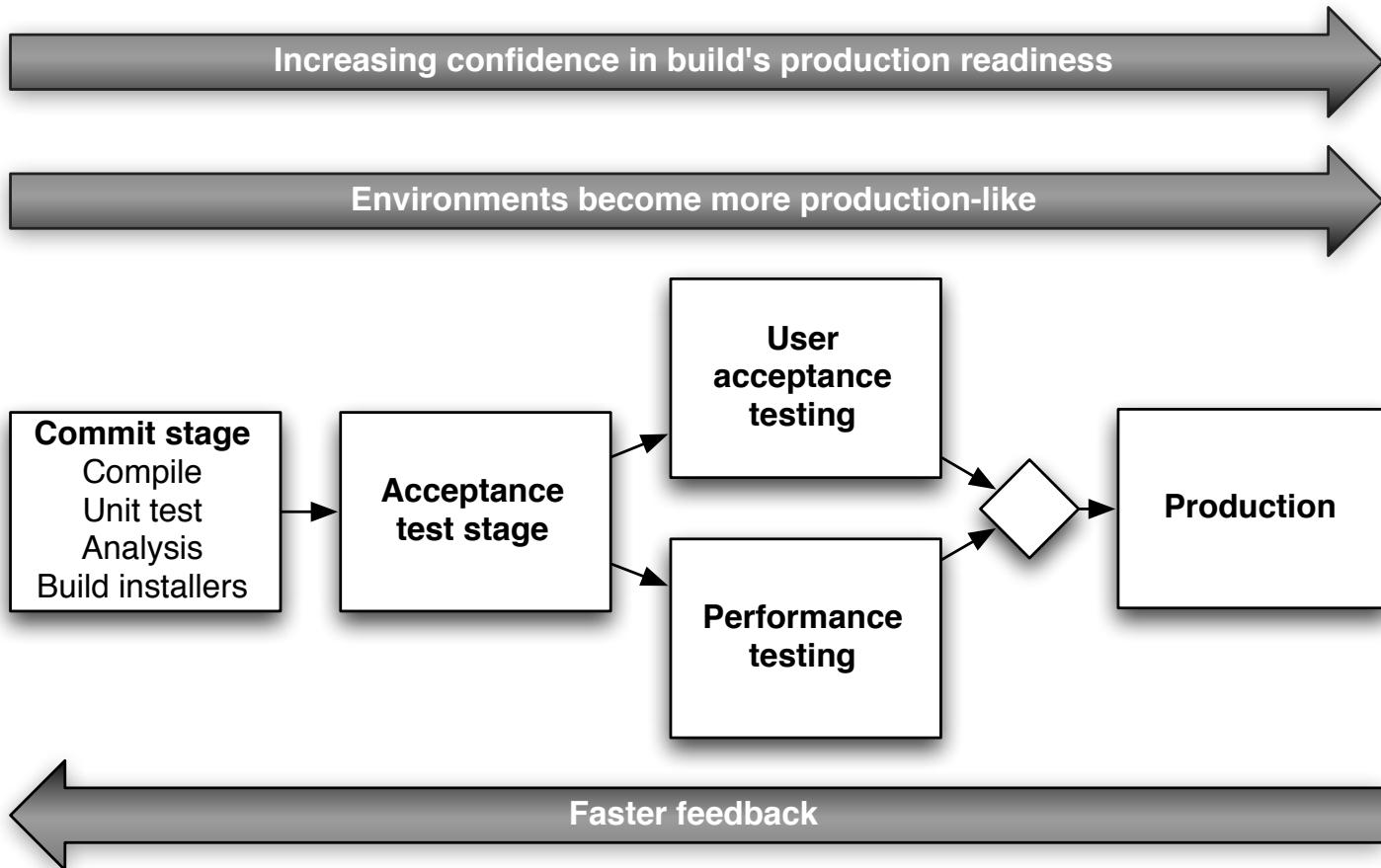


Integration Pipeline in Go CD



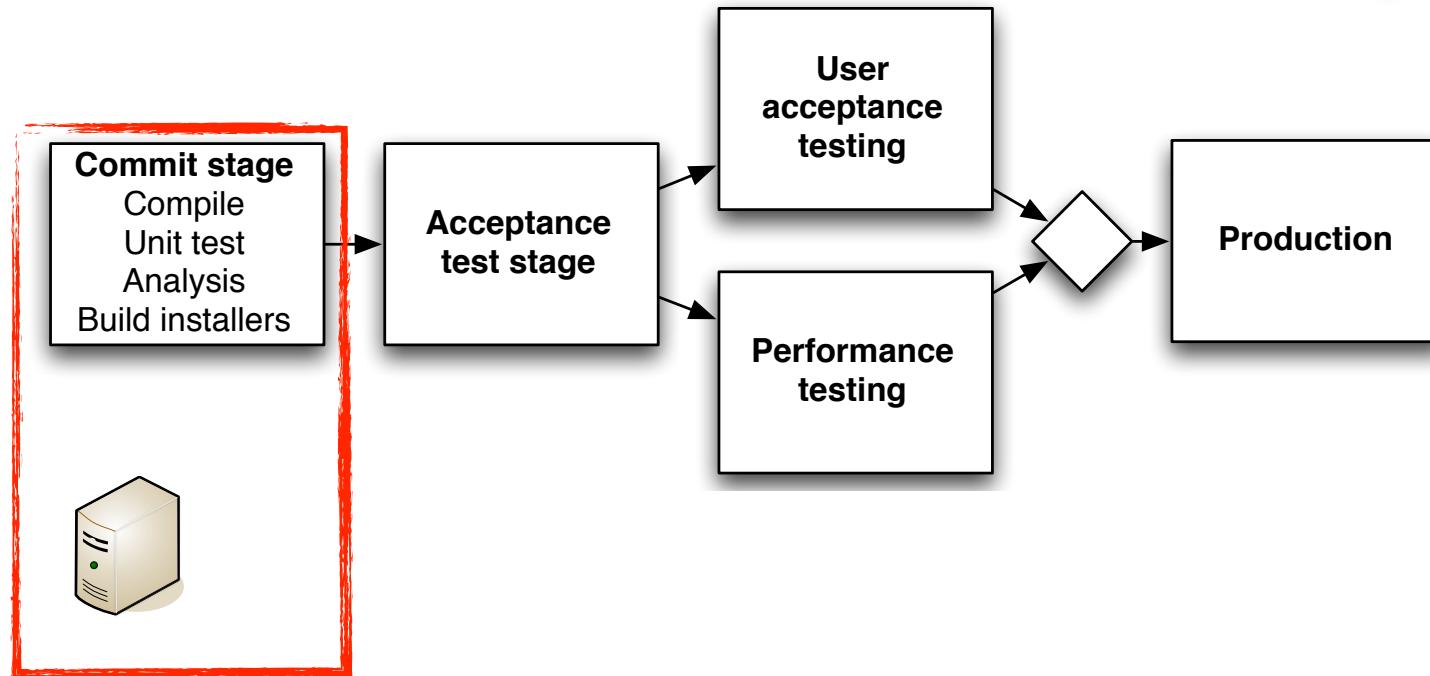


<https://www.gocd.org>



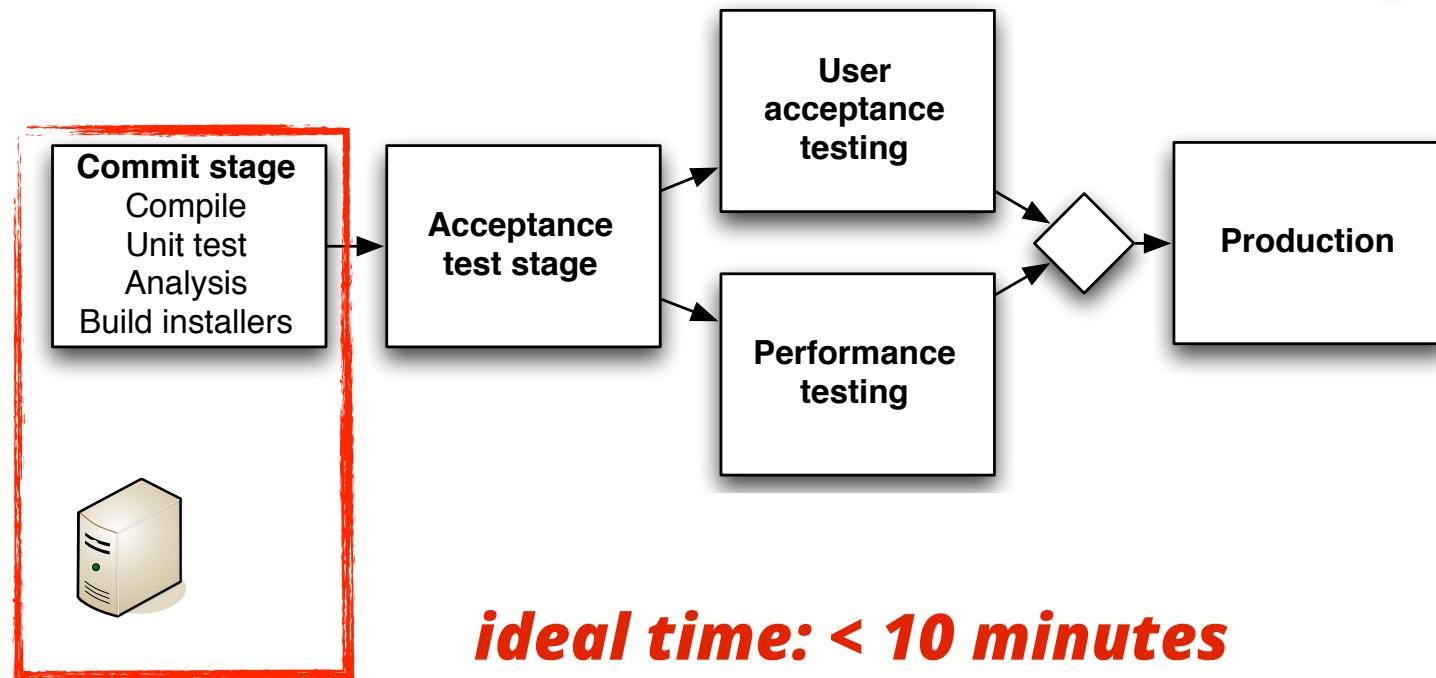
Pipeline Anti-patterns

insufficient parallelization



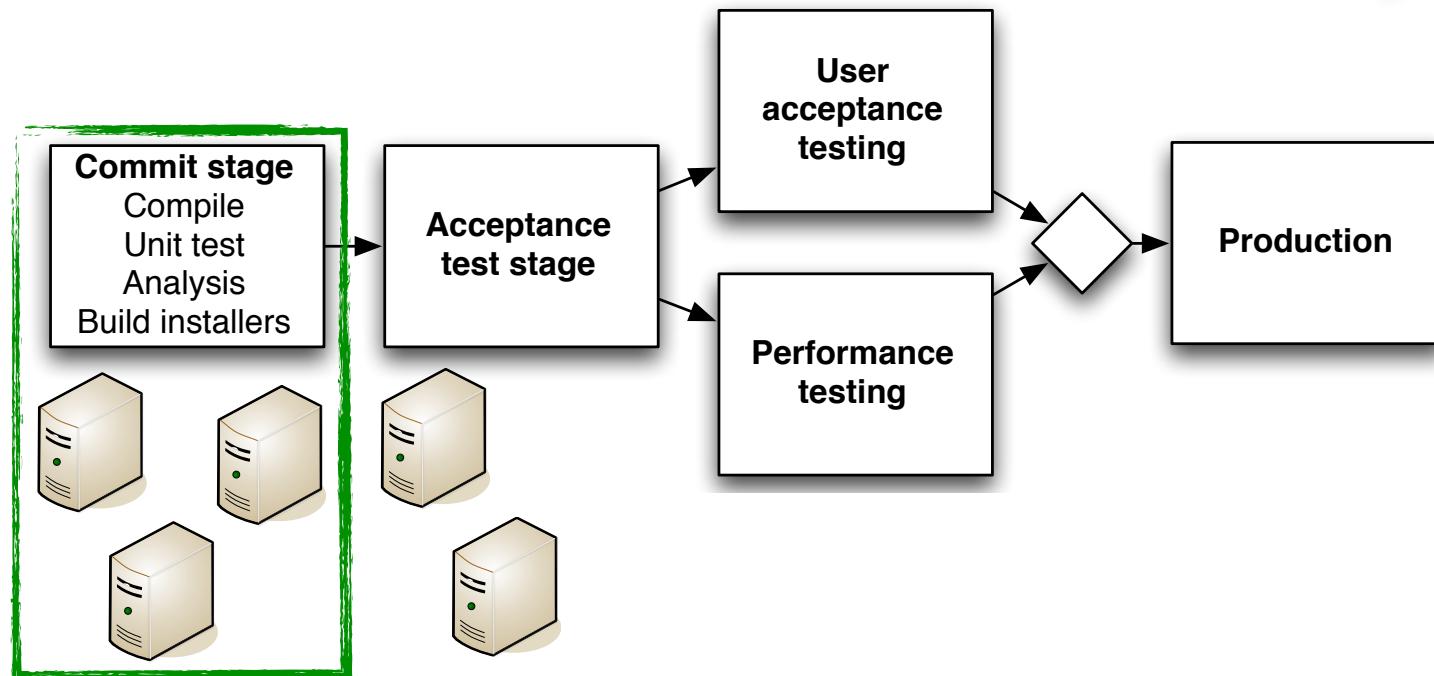
Pipeline Anti-patterns

insufficient parallelization



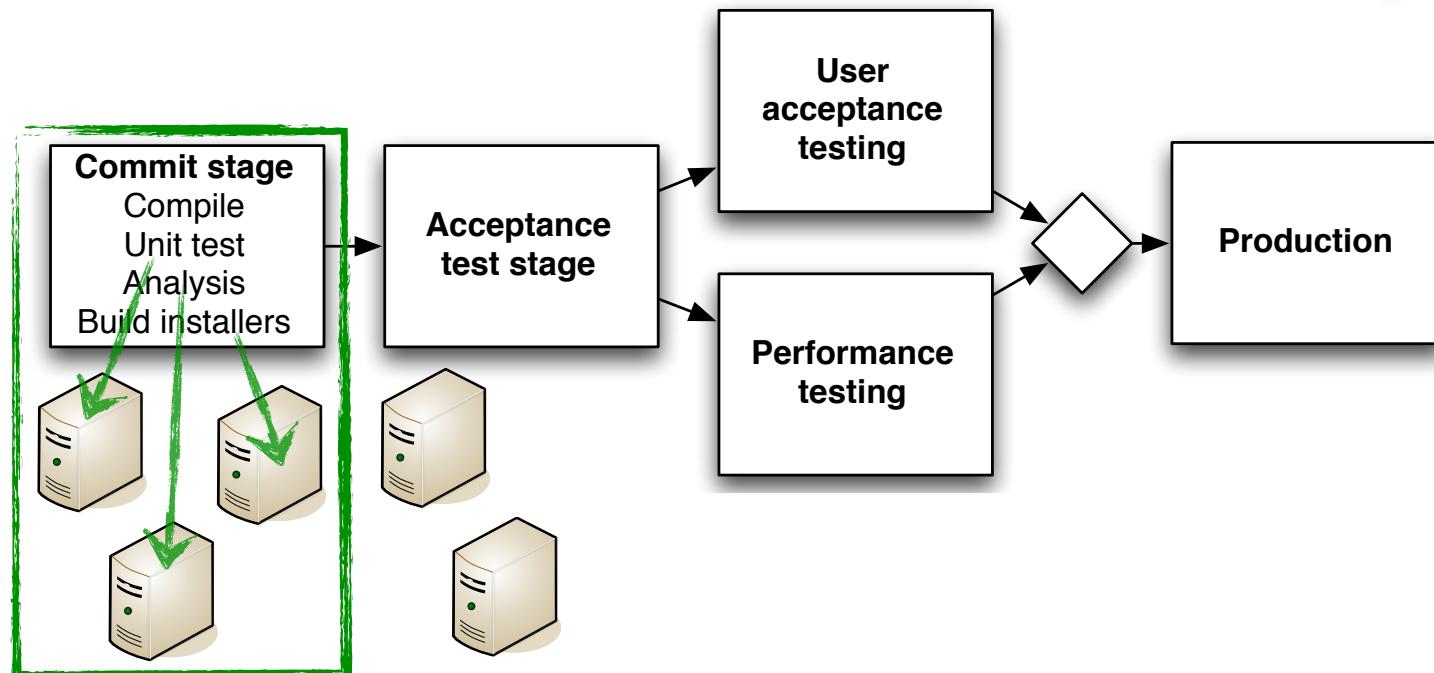
Pipeline Anti-patterns

insufficient parallelization



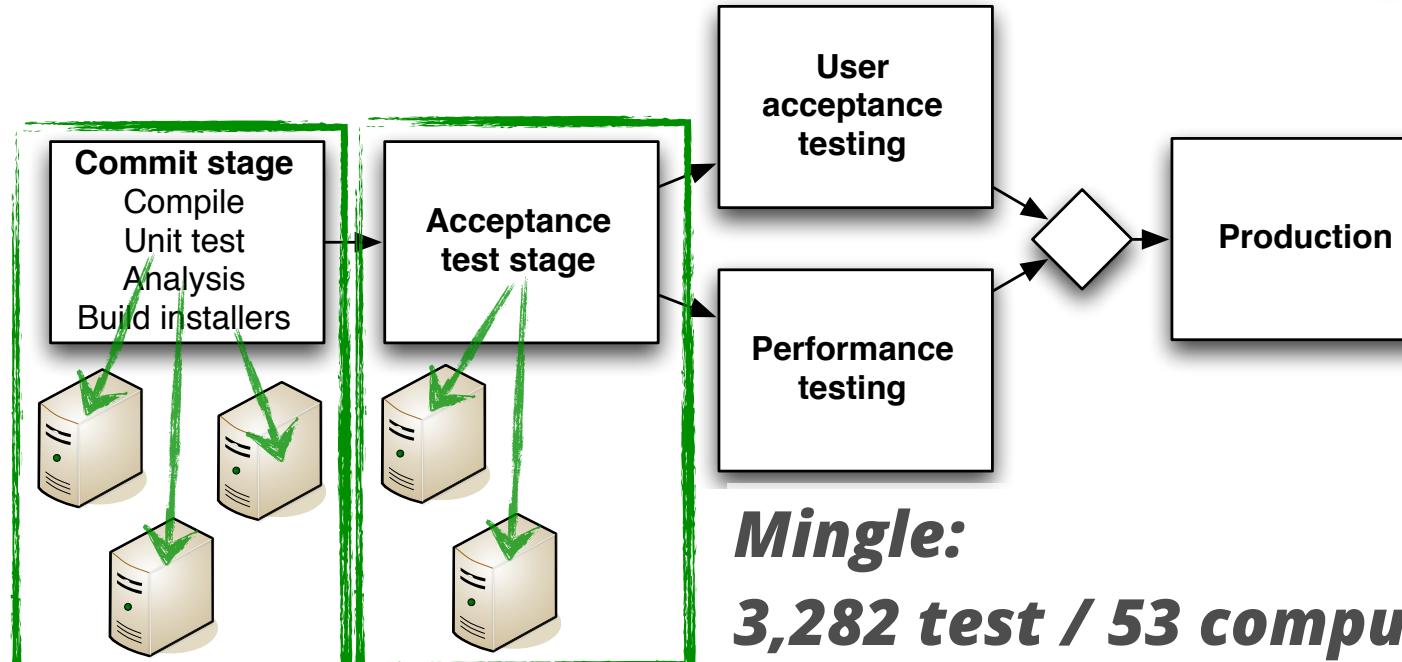
Pipeline Anti-patterns

insufficient parallelization



Pipeline Anti-patterns

insufficient parallelization



Mingle:
3,282 test / 53 computers
= ~1 hour

Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*



Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*

reduce the number of stages as much
as possible

Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*

reduce the number of stages as much
as possible

parallelize each stage as much as you
can

Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*

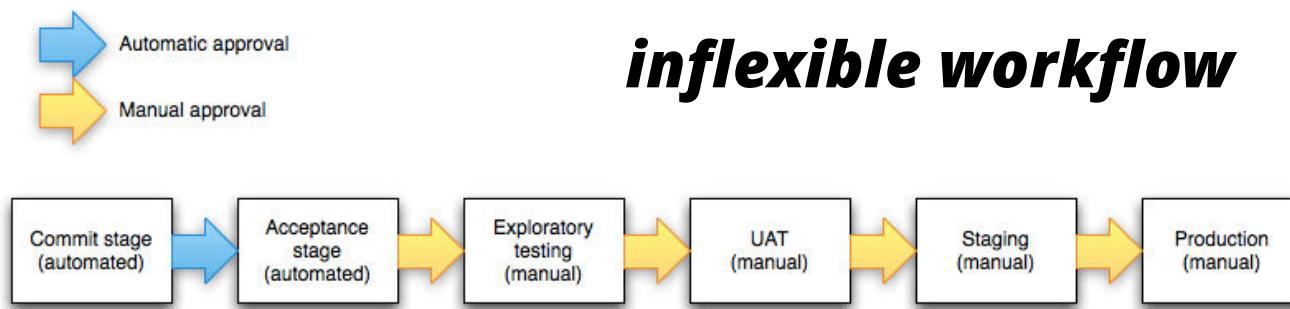
reduce the number of stages as much
as possible

parallelize each stage as much as you
can

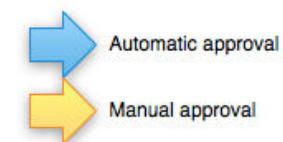
create more stages if necessary to
optimize feedback



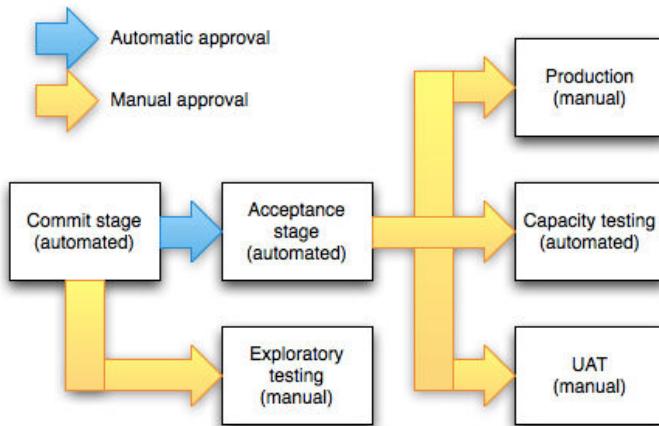
Pipeline Anti-patterns



Pipeline Anti-patterns

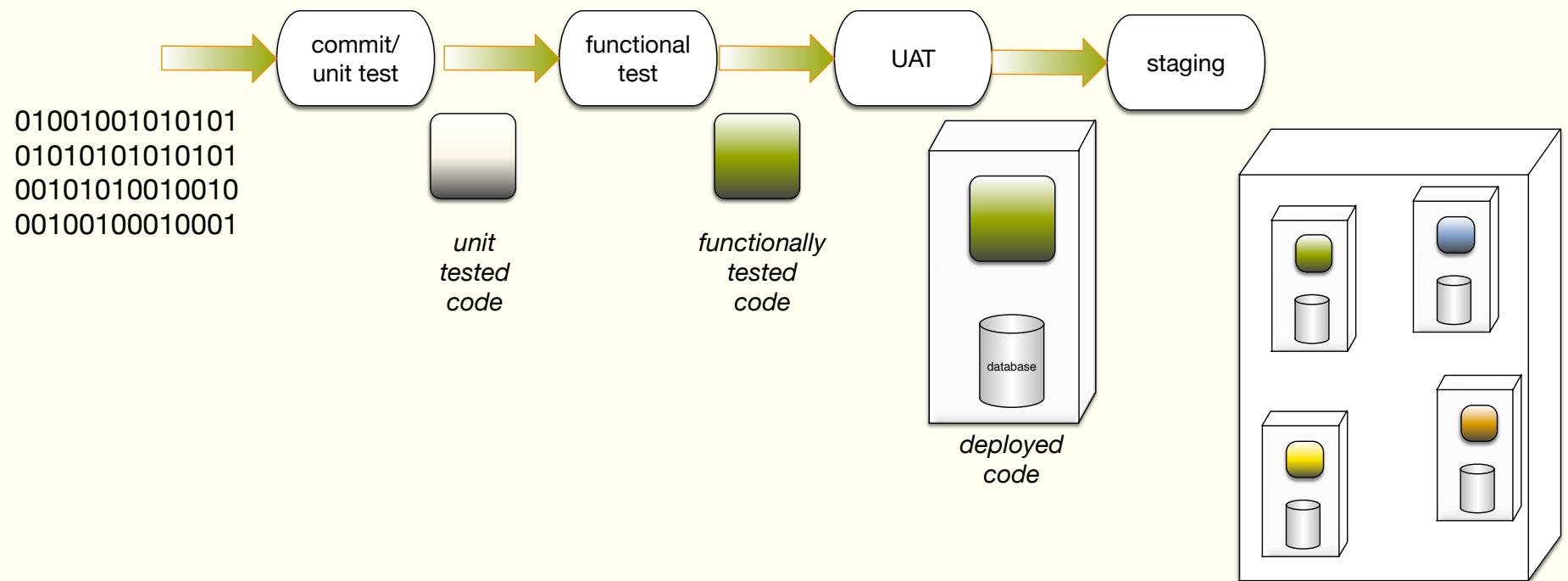


inflexible workflow

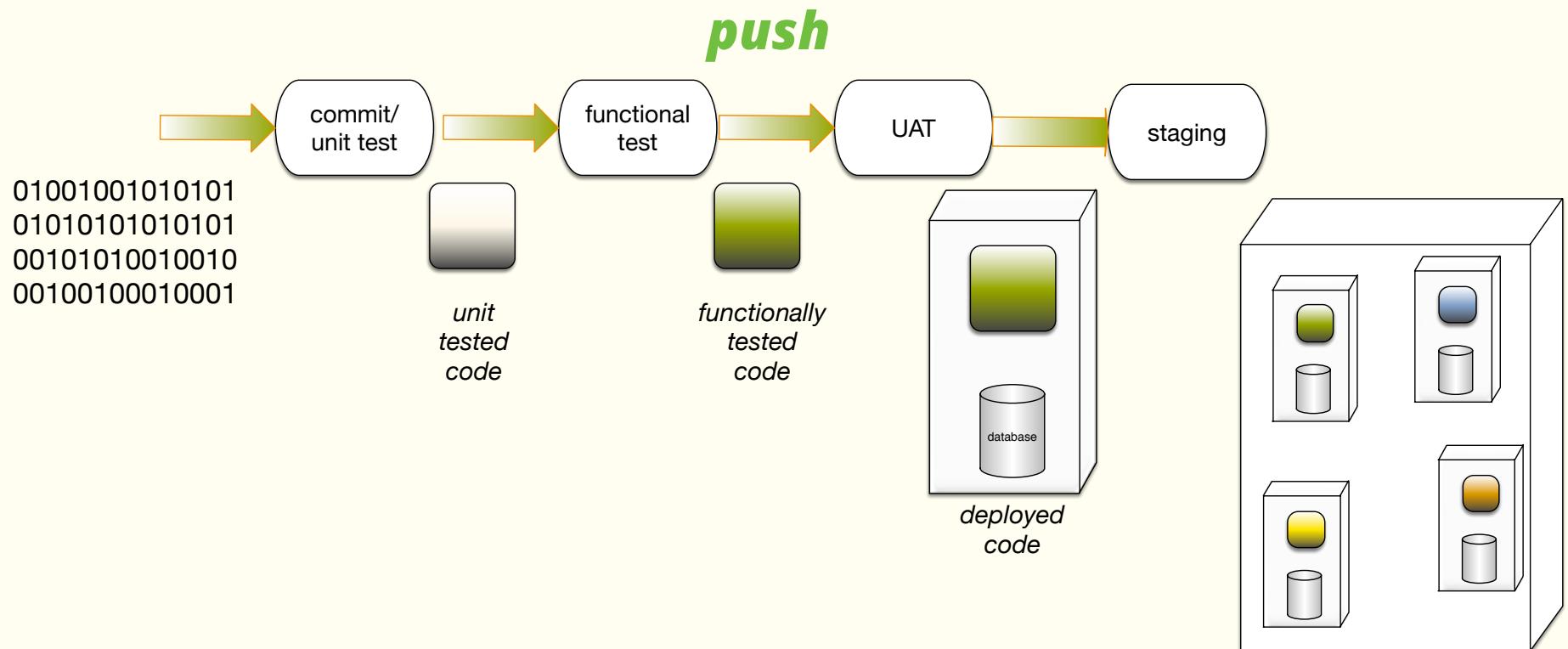


pipeline fans out as soon as it makes sense to do so

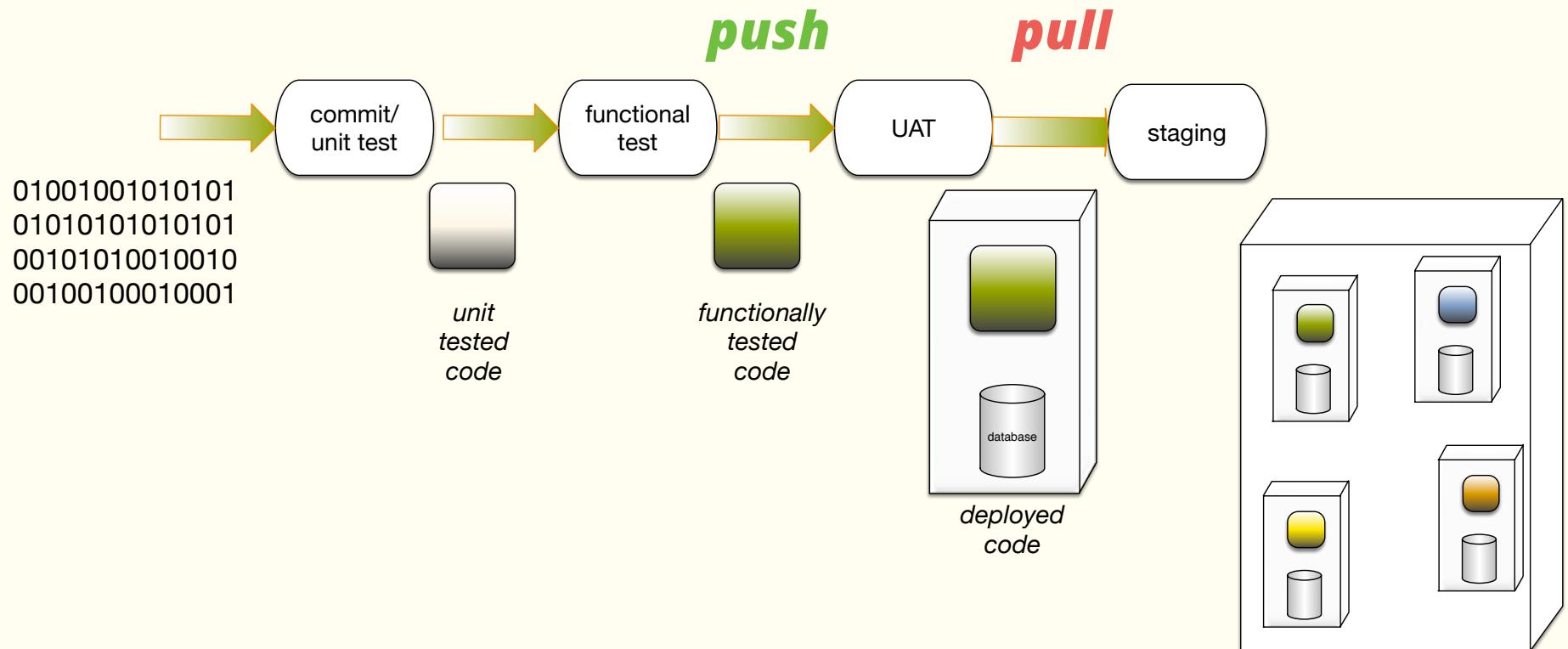
Deployment Pipeline

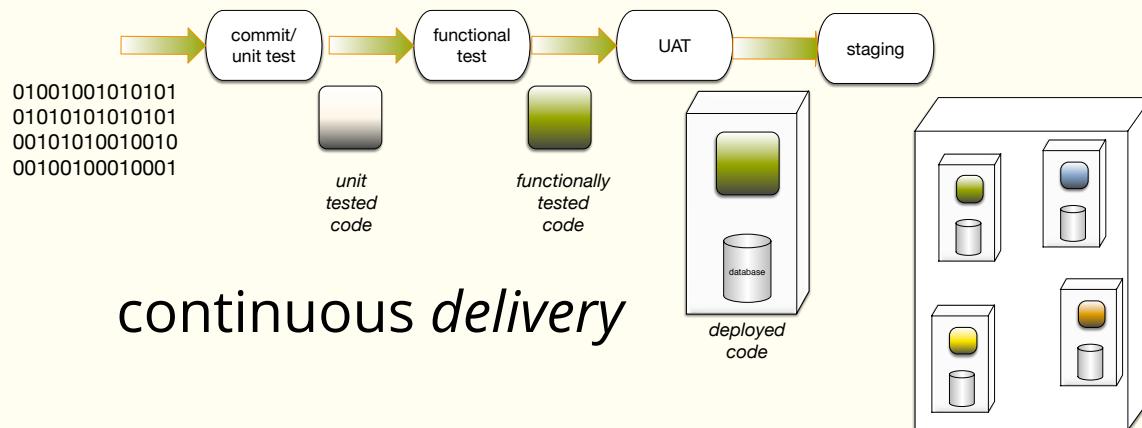


Deployment vs Delivery?

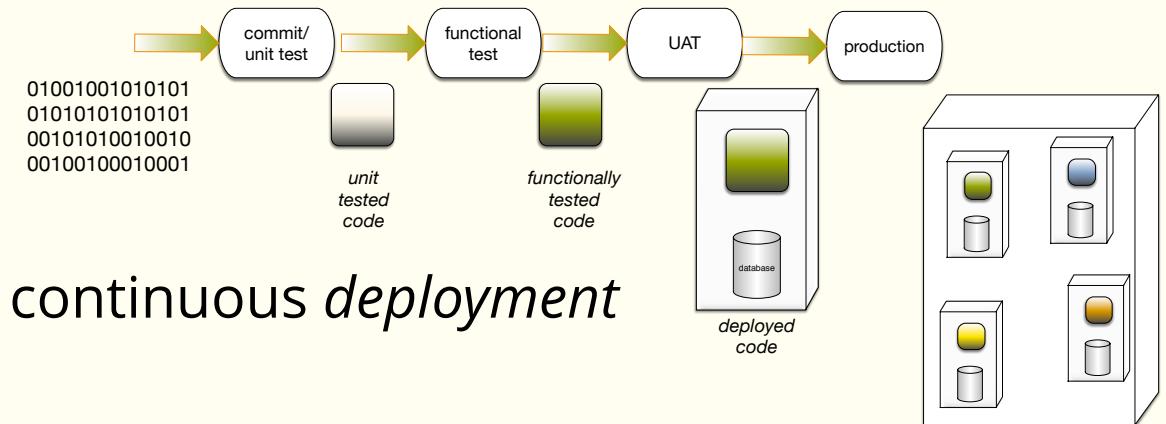


Deployment vs Delivery?

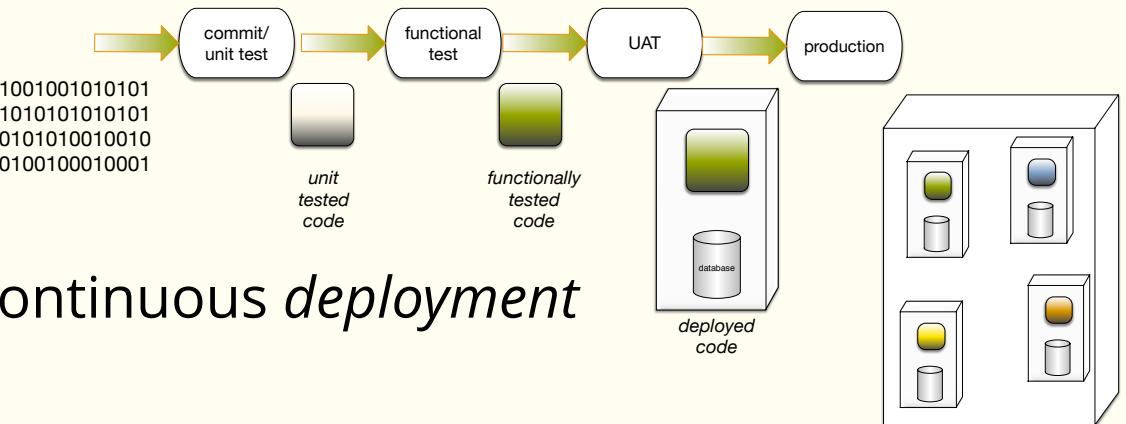
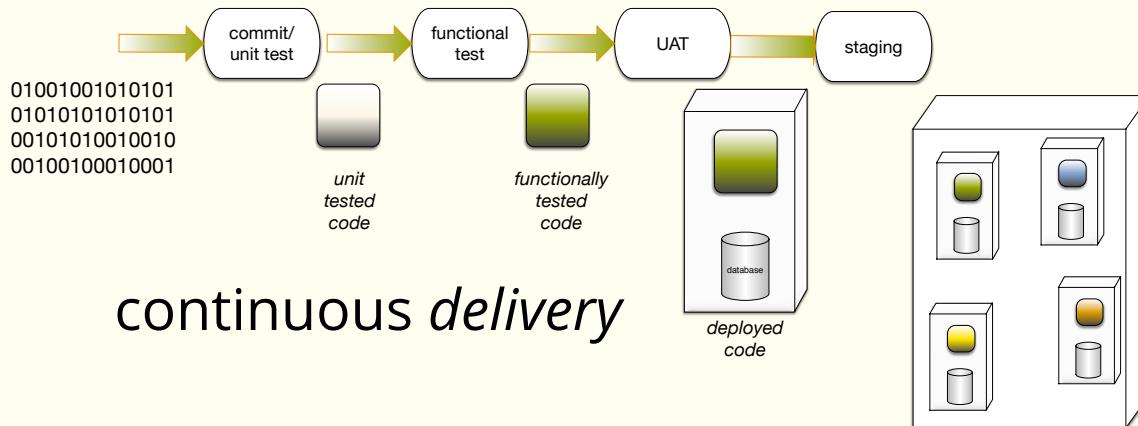




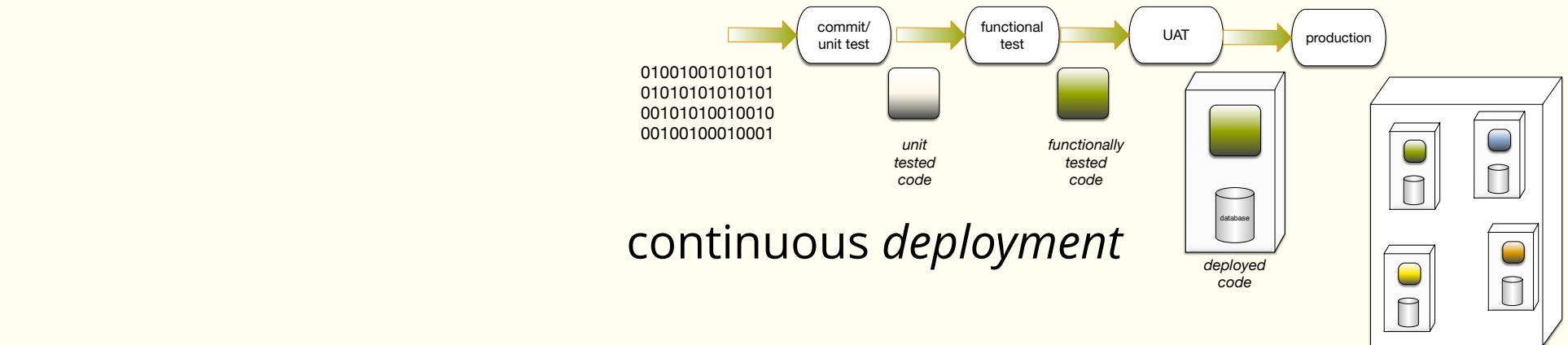
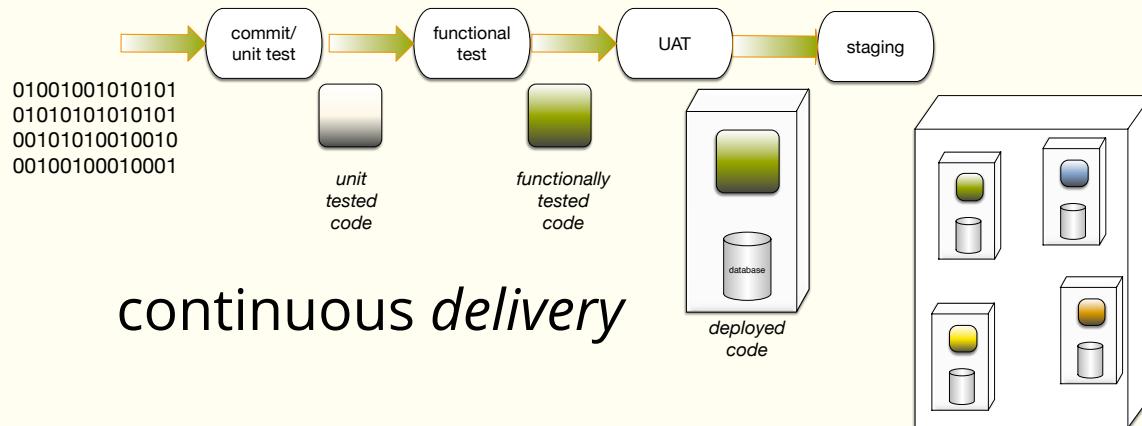
continuous delivery



continuous deployment

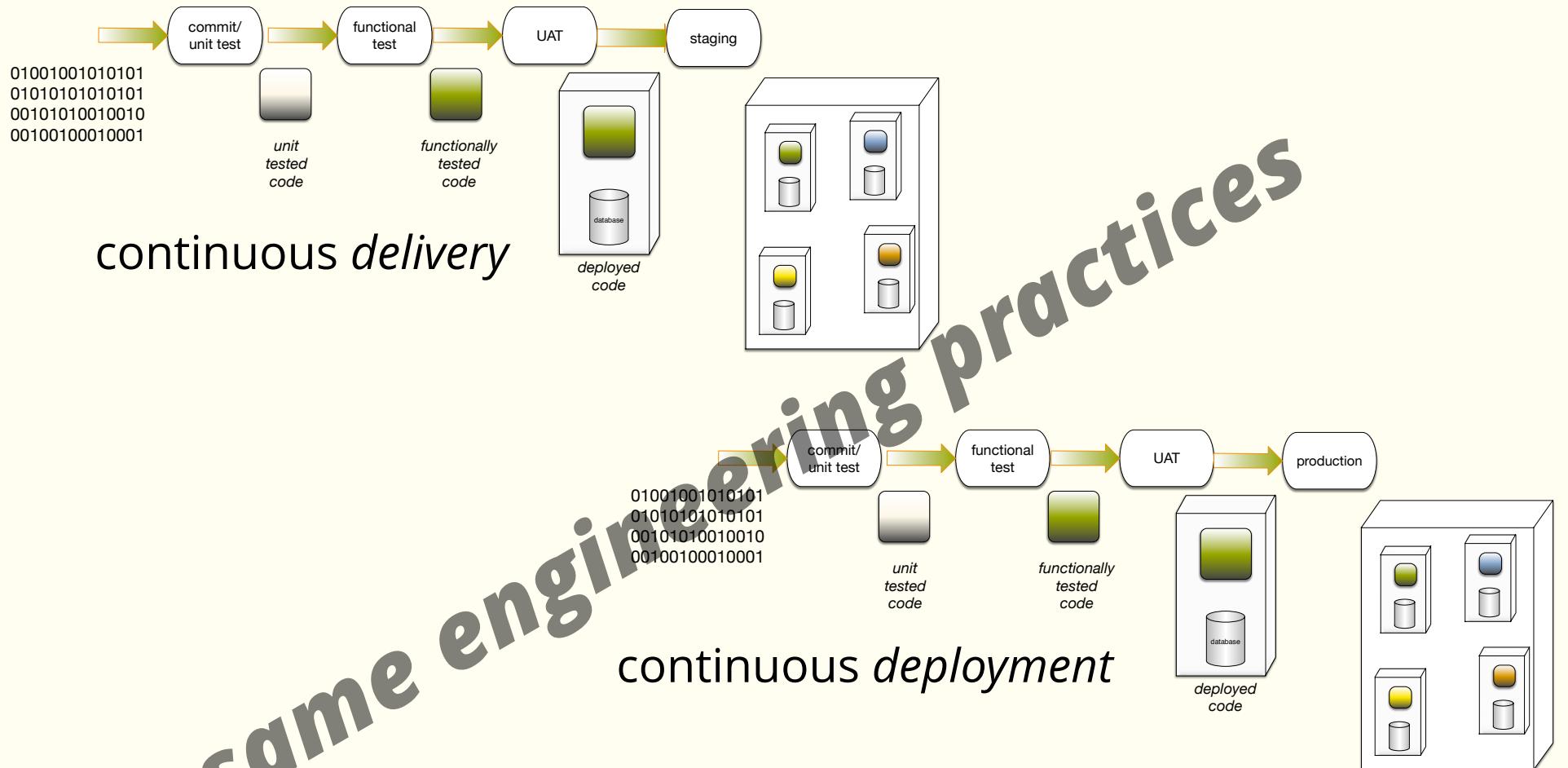


Continuous deployment: all **push** stages



Continuous *deployment*: all **push** stages

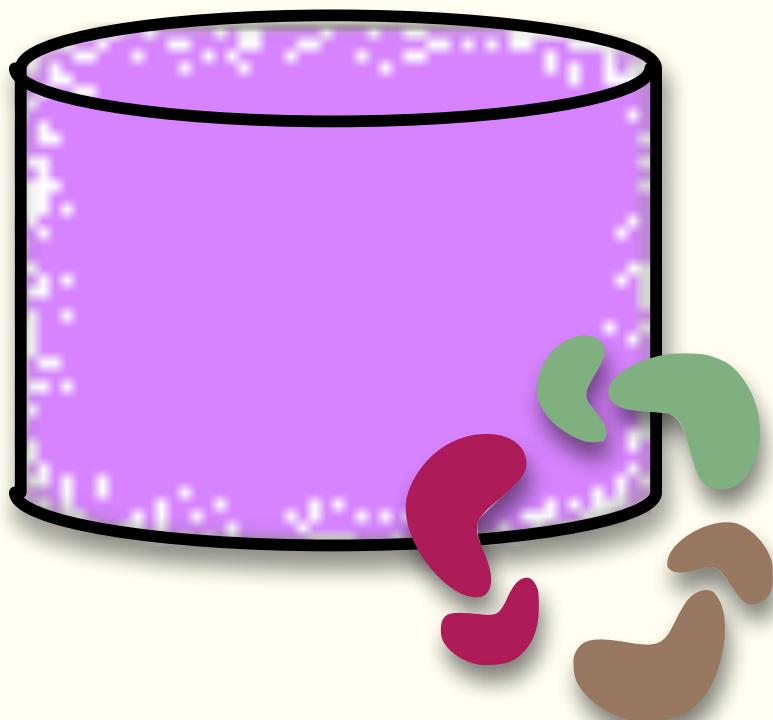
Continuous *delivery*: at least one **pull** stage

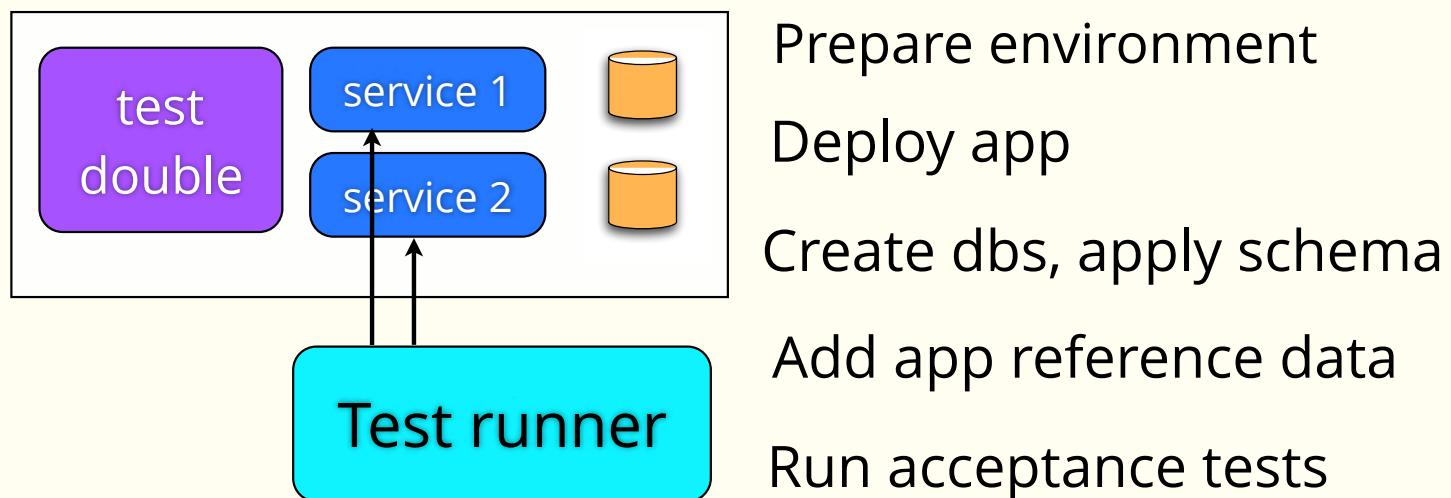
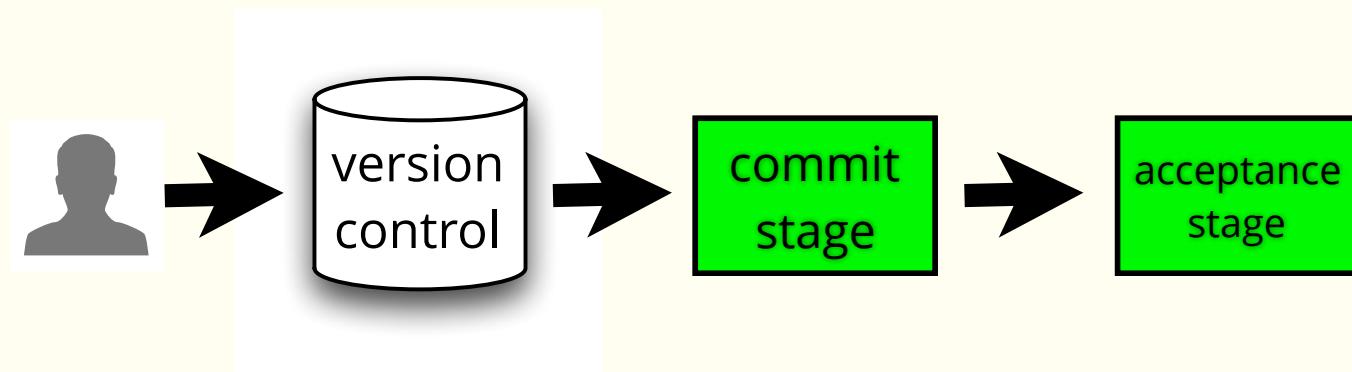


Continuous deployment: all **push** stages

Continuous delivery: at least one **pull** stage

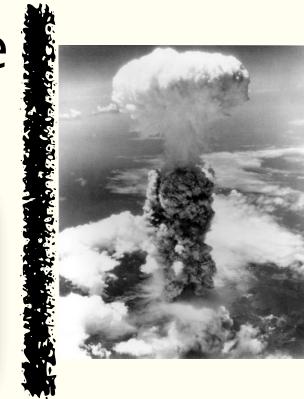
Continuous Integration for Databases





For DB CI We Need To:

start with a clean database



apply changes incrementally

use the same process everywhere

be comprehensive in change management

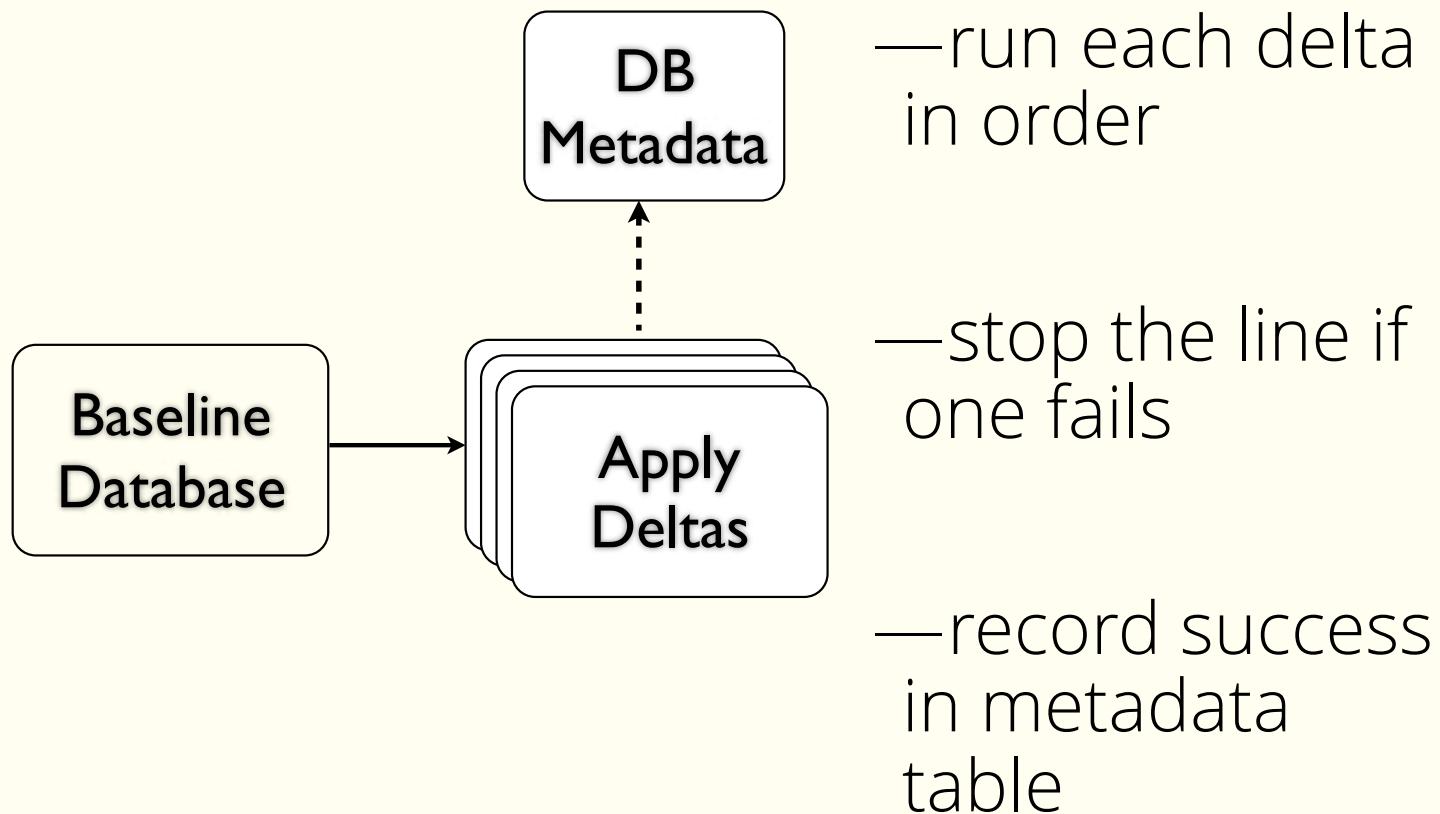
#1: Baseline

- Create database
- Add metadata table

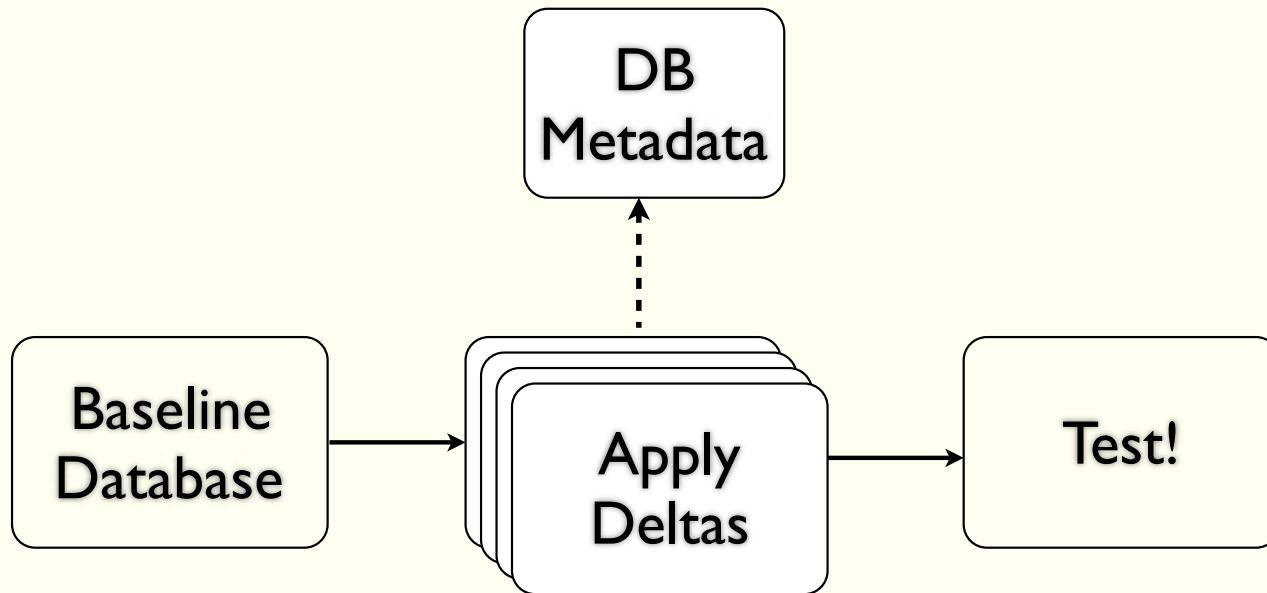
**Baseline
Database**

- Restore schema & reference data to current production state

#2: Apply Deltas

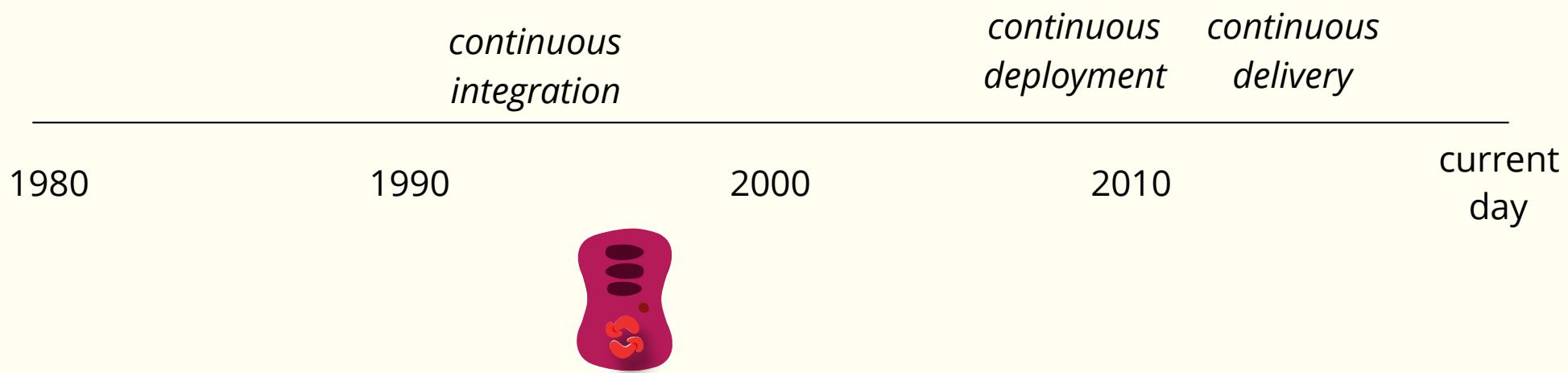


#3: Run Tests

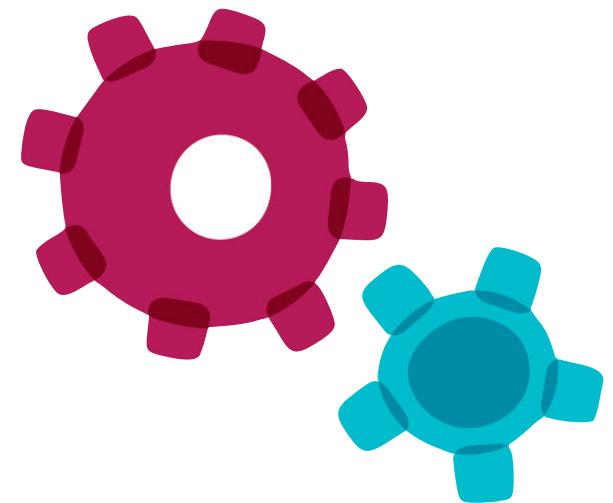


acceptance tests verify database scripts worked

integration as an engineering practice over time



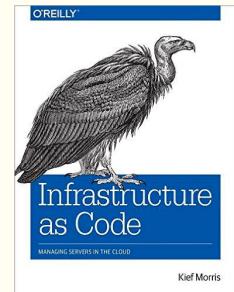
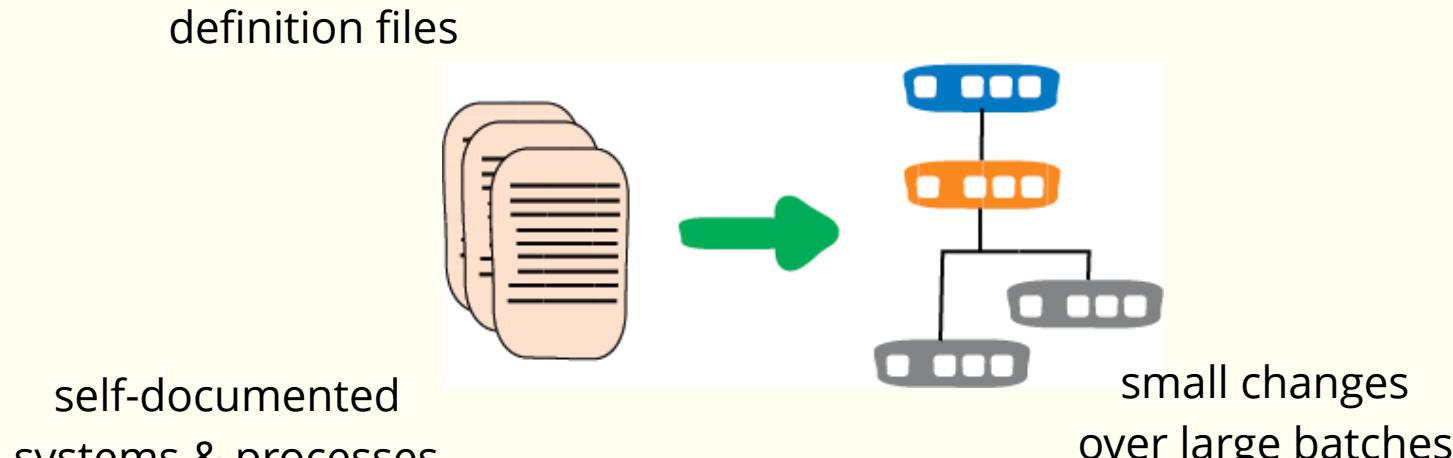
engineering practices to support continuous delivery



infrastructure as code

version all the things

continuously test
systems & processes



keep services available continuously

kNighT cApiTaL

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

The screenshot shows a blog post titled "Knightmare: A DevOps Cautionary Tale" by Doug Seven. The post is dated April 17, 2014, and has 37 comments. It received 70 votes and has a 5-star rating. The post discusses a company that went bankrupt in 45 minutes due to a failed deployment. The author's bio includes a photo of him wearing a Seattle Seahawks cap and sunglasses. The sidebar features a search bar and a "RECENT POSTS" section.

DOUG SEVEN
Something can be learned in the course of observing things

HOME IOT WORKSHOP PRODUCT MANAGEMENT ABOUT

POSTS COMMENTS

You are here: Home / DevOps / Knightmare: A DevOps Cautionary Tale

Knightmare: A DevOps Cautionary Tale

APRIL 17, 2014 BY D7 37 COMMENTS 70 Votes

★★★★★ 70 Votes

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.

Background

Knight Capital Group is an American global financial services firm engaging in [market making](#), electronic execution, and institutional sales and trading. In 2012 Knight was the largest trader in US equities with market share of around 17% on each the NYSE and NASDAQ. Knight's Electronic

DOUG SEVEN

SEARCH

Search this website... Search

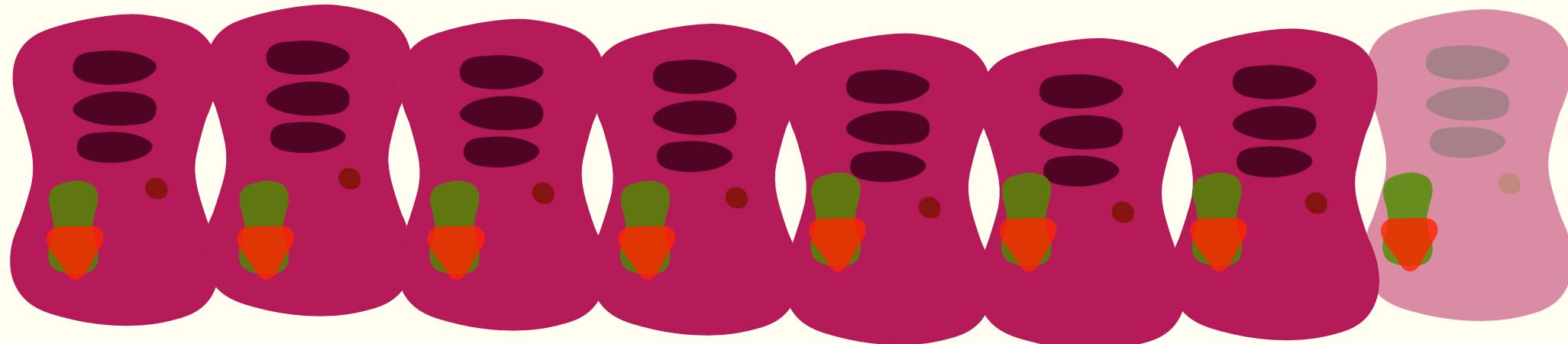
RECENT POSTS:

- IoT Workshop: Lesson 3 – Input Controls Output
- Arduino: Reading Analog Voltage
- Arduino 'Hello, World!' – Sending Digital Output

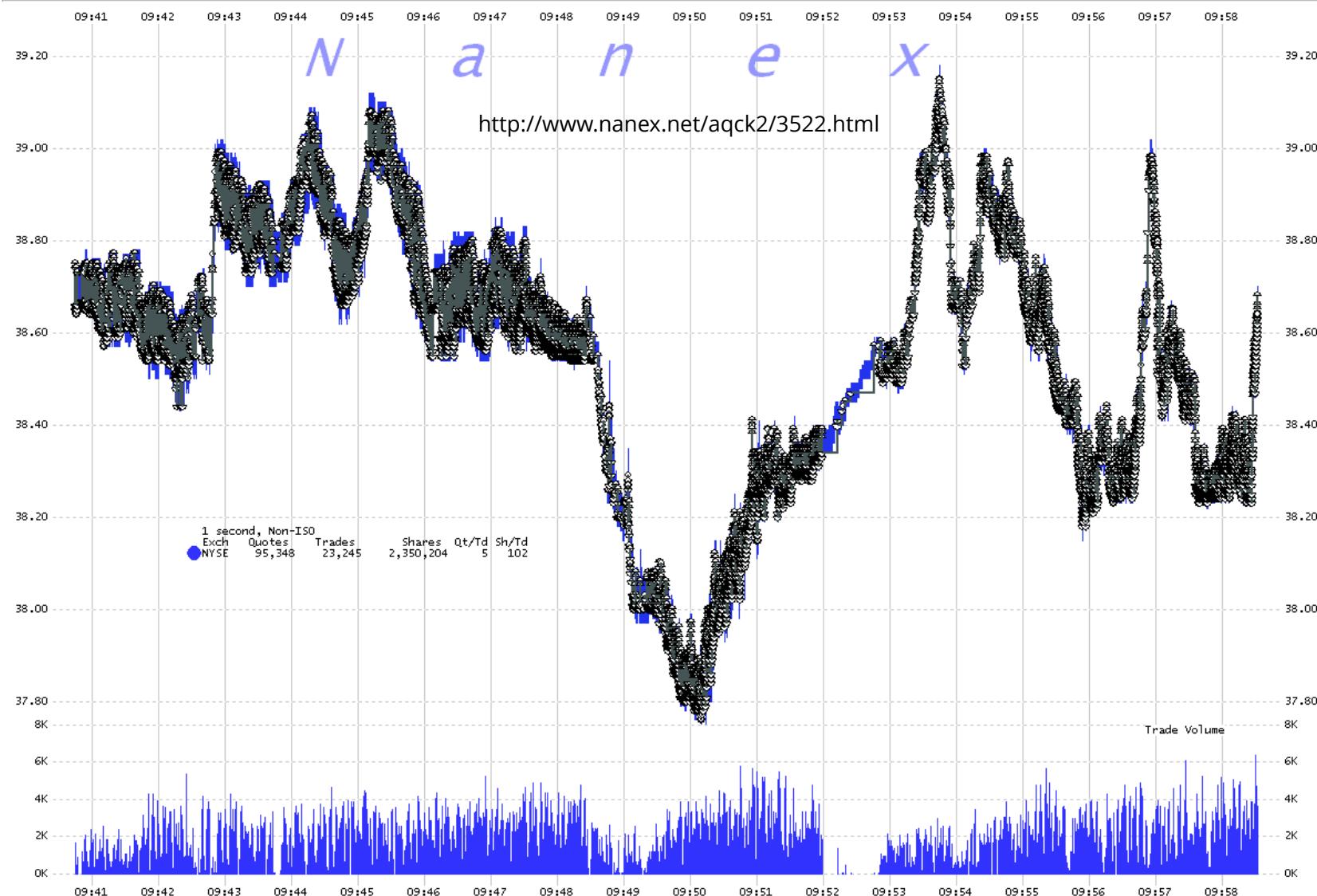
"bankrupt in 45 minutes"

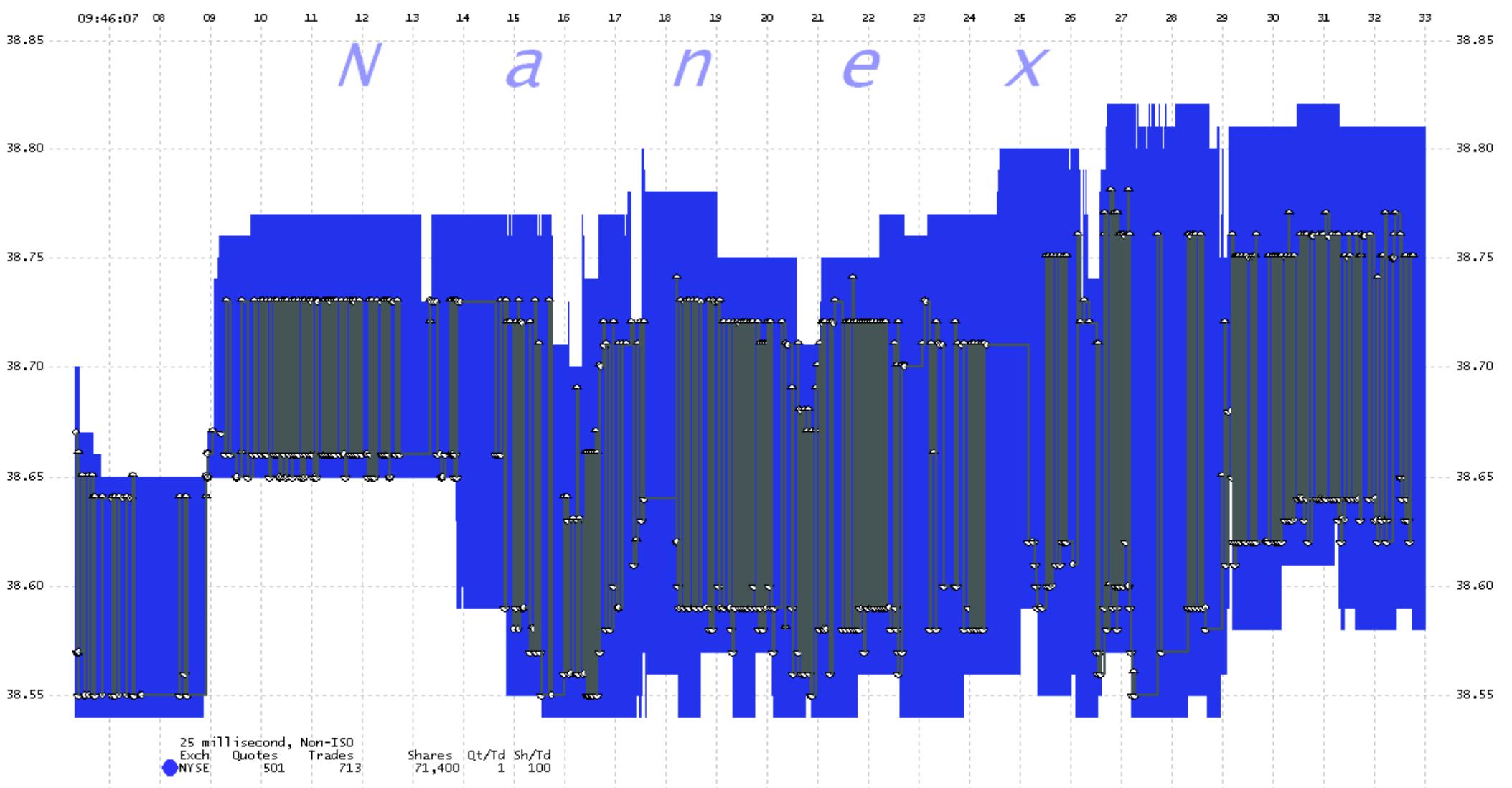
SMARS

PowerPeg



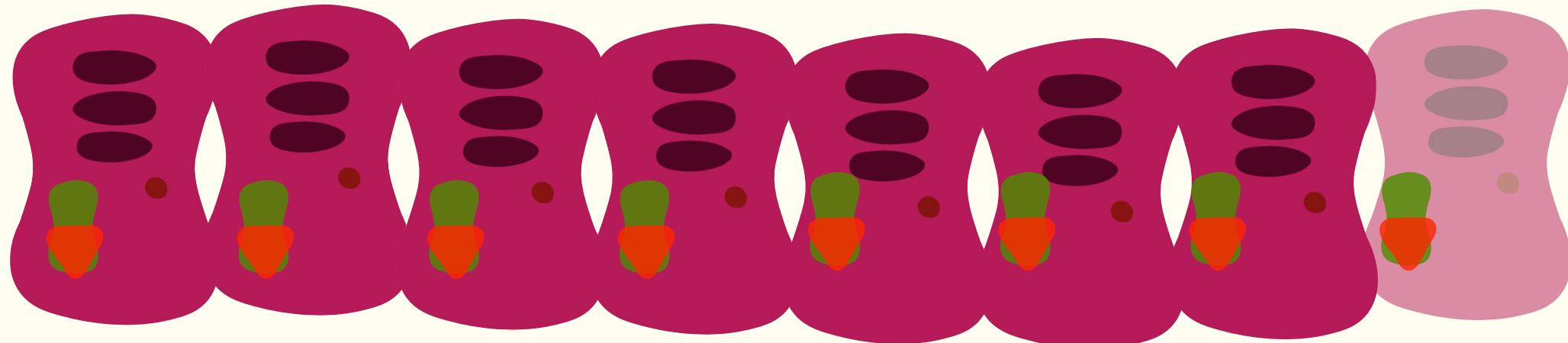
1. EXC One second interval chart. Circles are trades, the blue coloring is the NYSE bid and ask which is mostly covered by gray lines that connect the trades.





SMARS

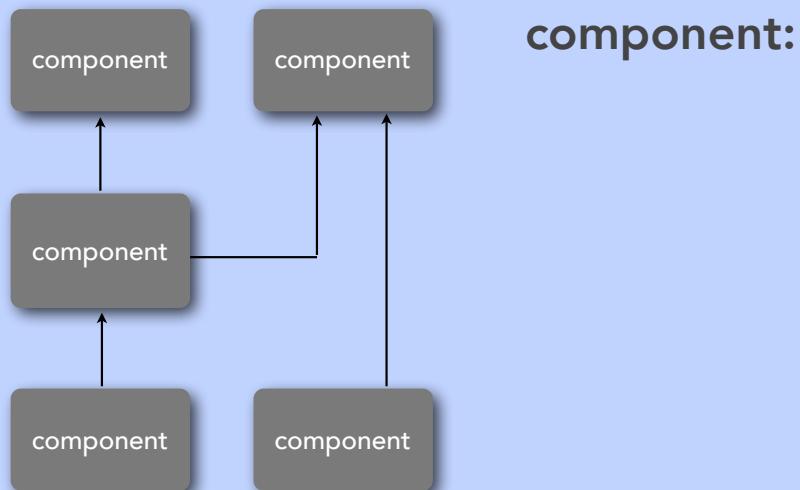
PowerPeg



aRChiTeCtuRe
fOr
cONTiNuoUs DeLiVeRy

component identification

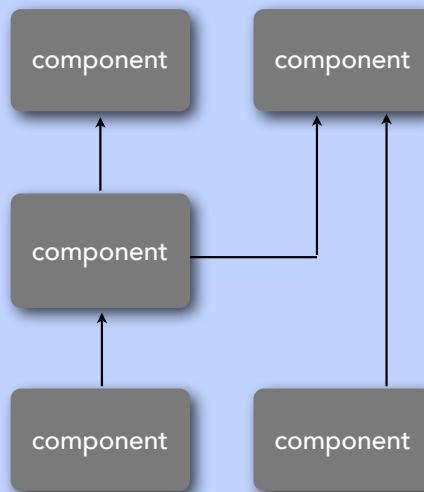
as an architect, you should think about the artifacts
within the architecture in terms of *components*



component:

component identification

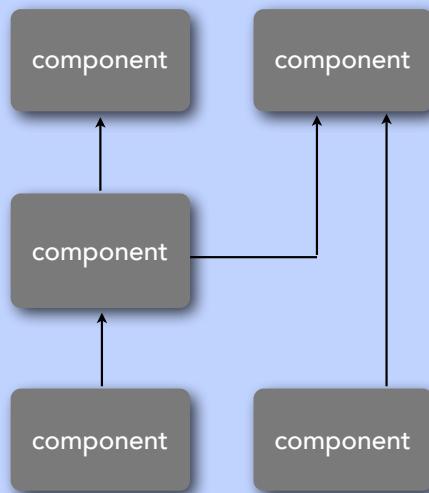
as an architect, you should think about the artifacts
within the architecture in terms of *components*



component:
building block of the application

component identification

as an architect, you should think about the artifacts
within the architecture in terms of *components*

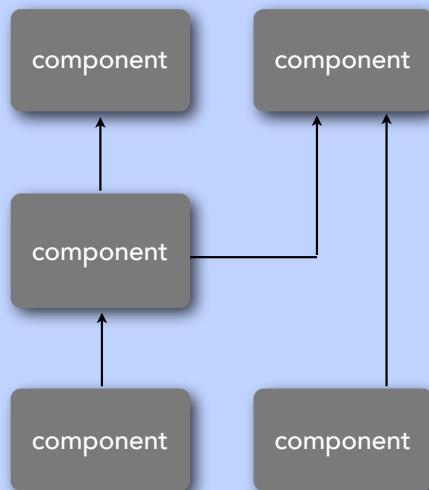


component:

building block of the application
well defined set of operations

component identification

as an architect, you should think about the artifacts
within the architecture in terms of *components*

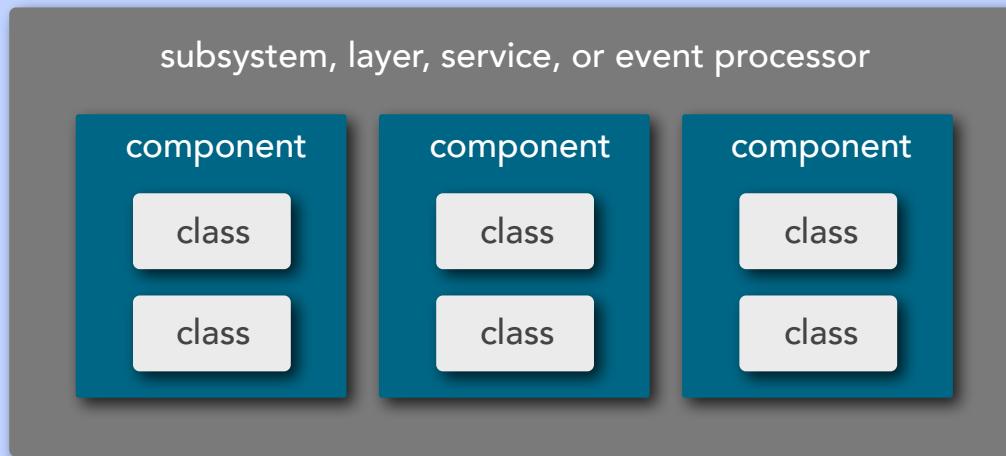


component:

building block of the application
well defined set of operations
well defined role and responsibility

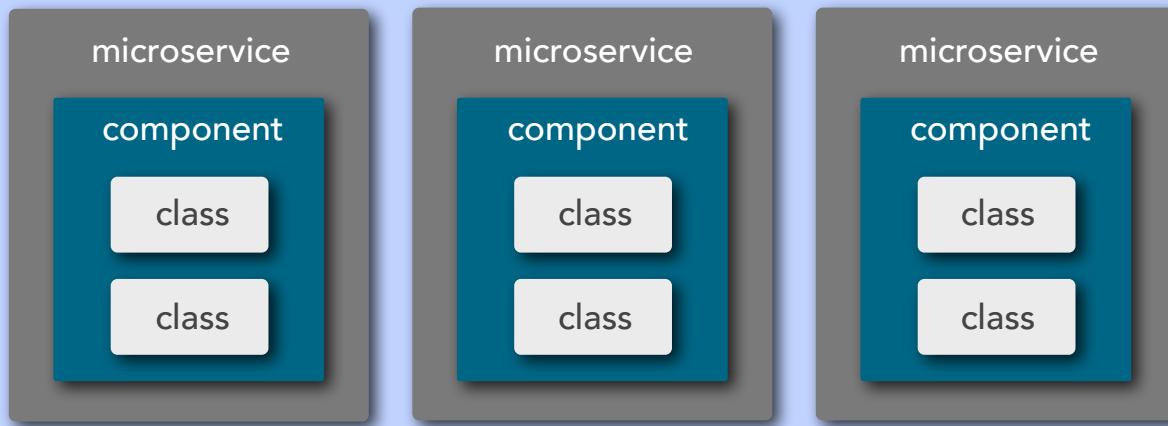
component identification

component scope



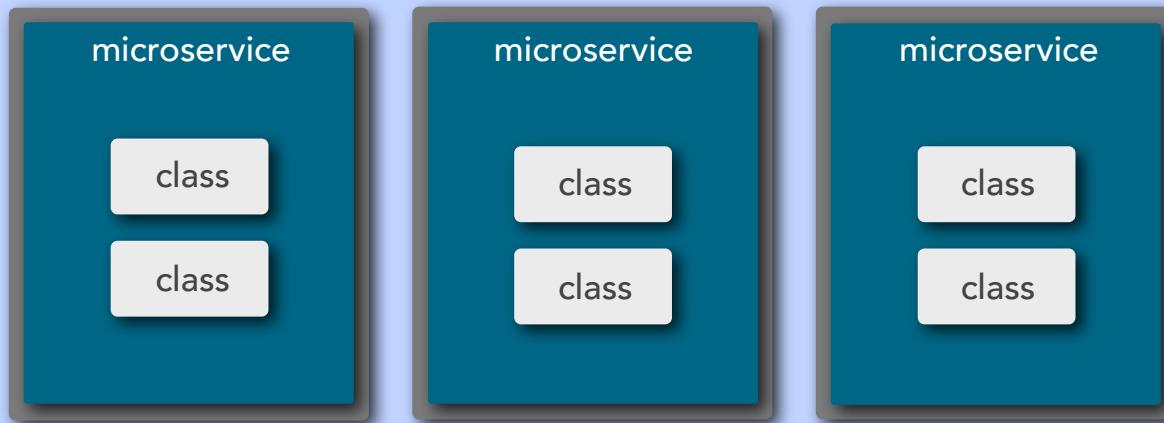
component identification

component scope

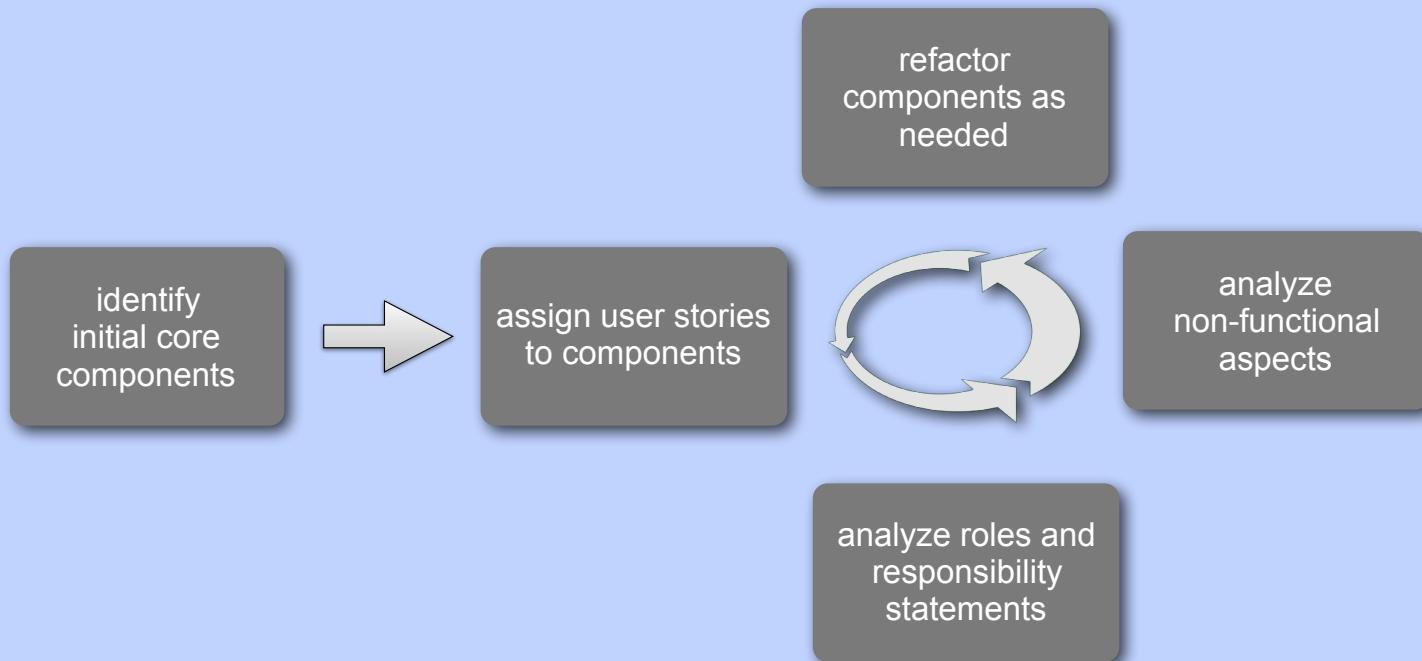


component identification

component scope

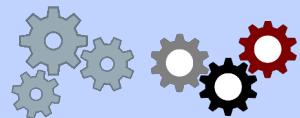


component identification



service identification

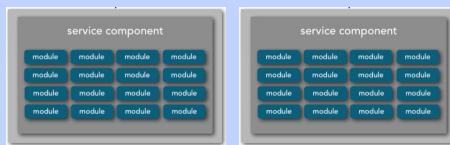
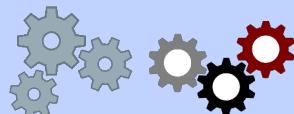
identify coarse-grained functional areas



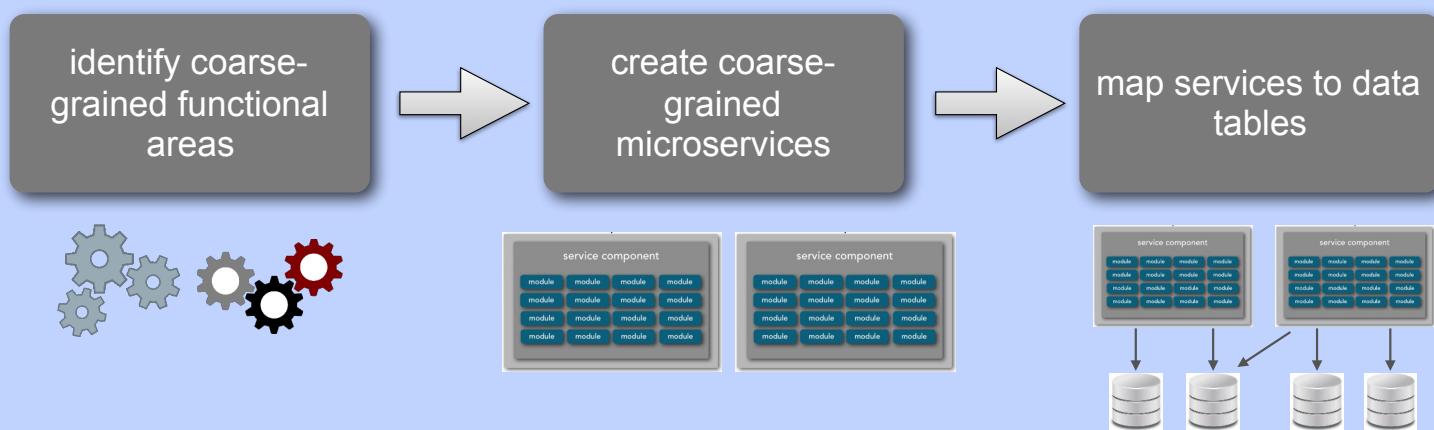
service identification

identify coarse-grained functional areas

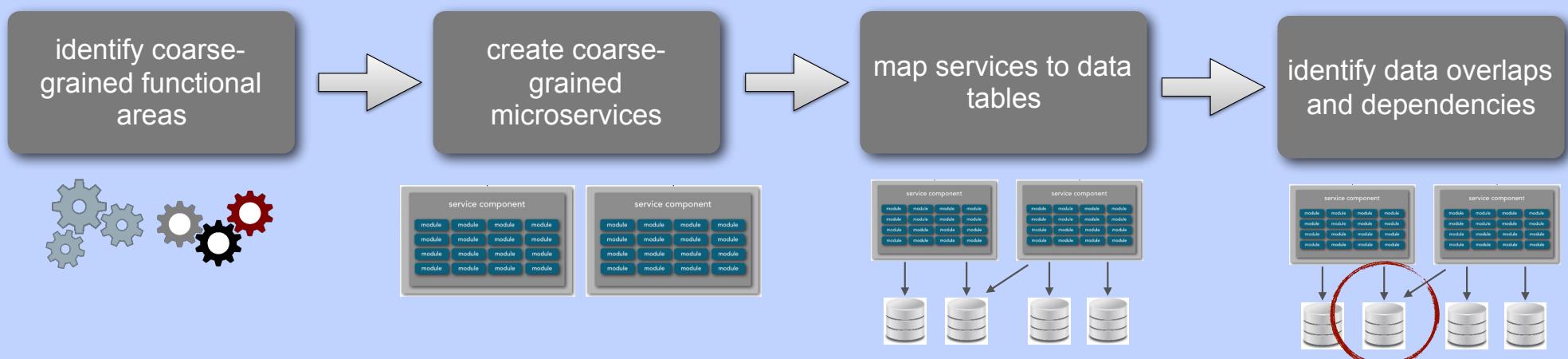
create coarse-grained microservices



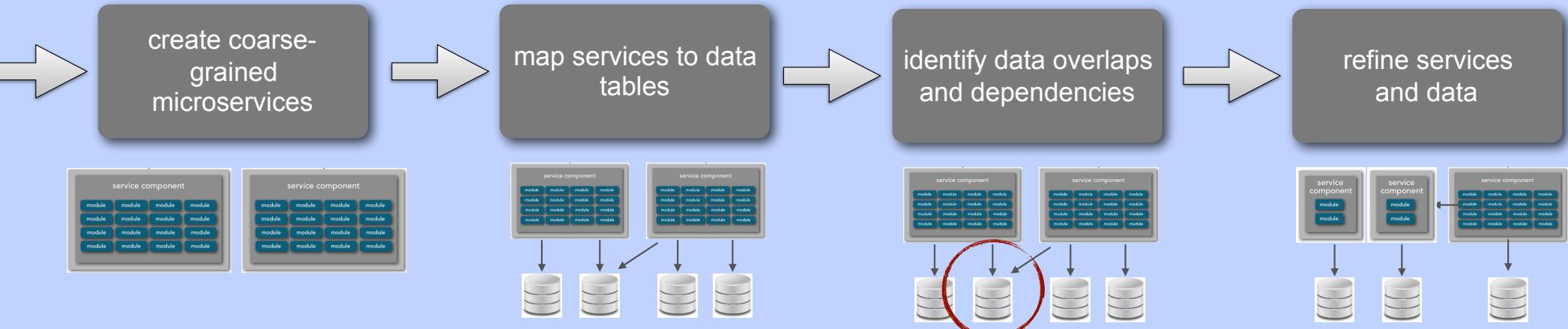
service identification



service identification



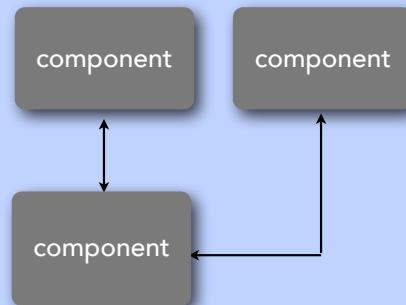
service identification



component coupling

component coupling

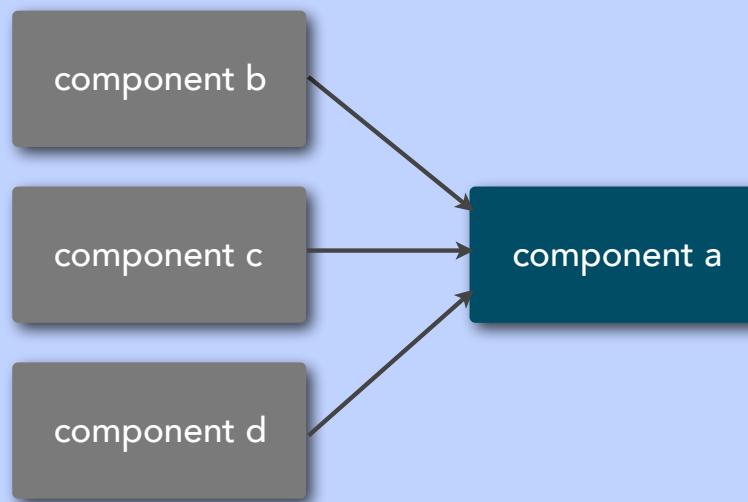
the extent to which components know
about each other



component coupling

afferent coupling

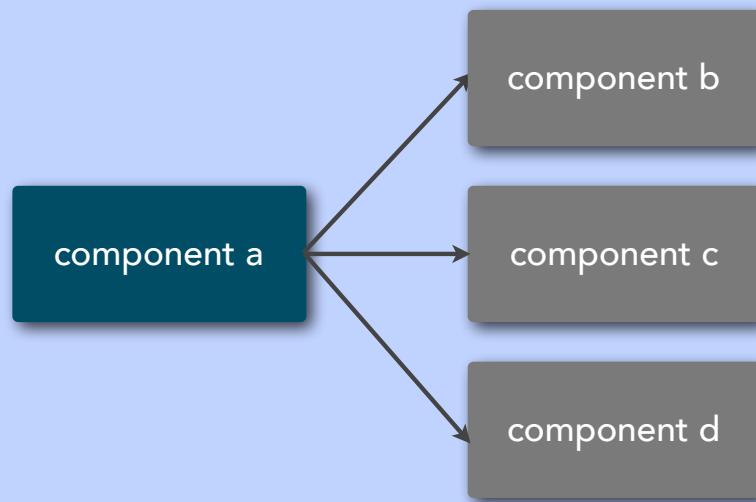
the degree to which other components are dependent on the target component



component coupling

efferent coupling

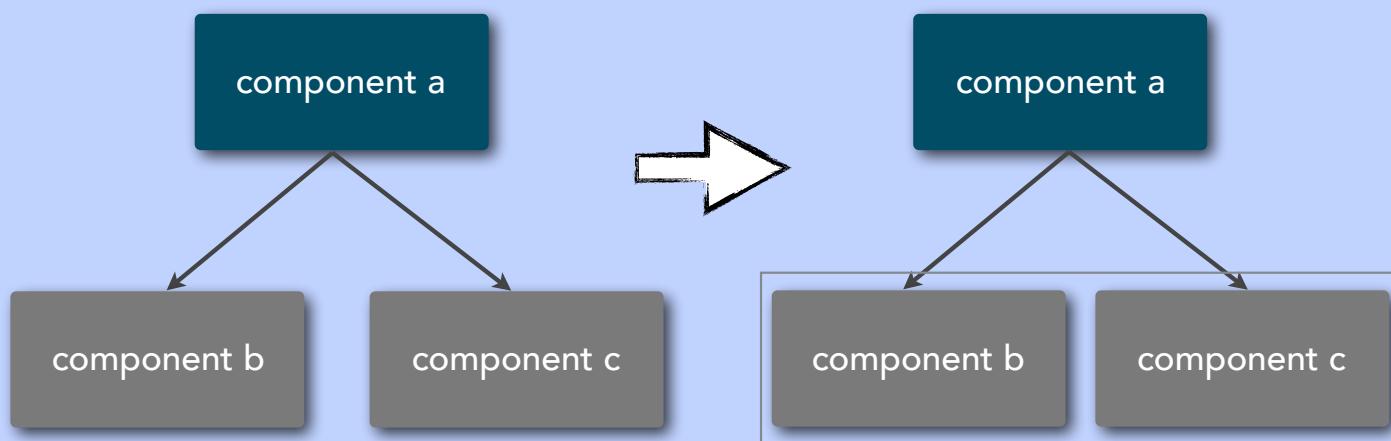
the degree to which the target component
is dependent on other components



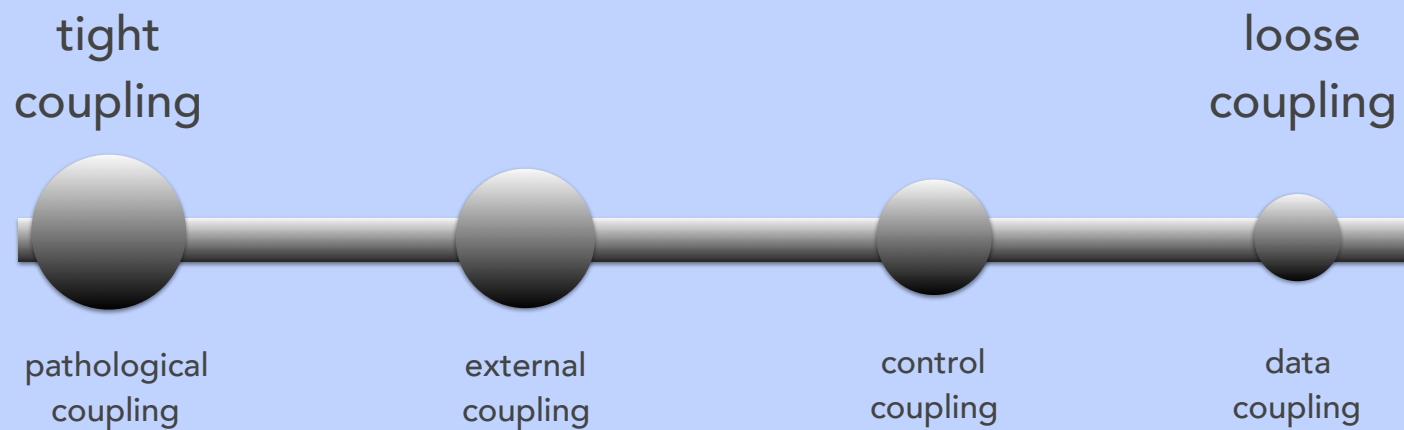
component coupling

temporal coupling

components are coupled due to non-static or timing dependencies



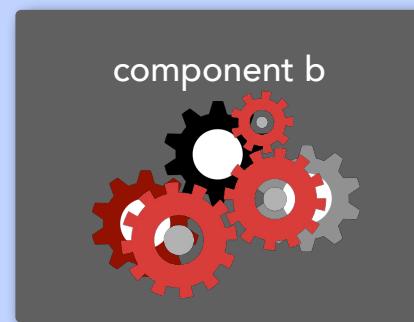
component coupling



component coupling

pathological coupling

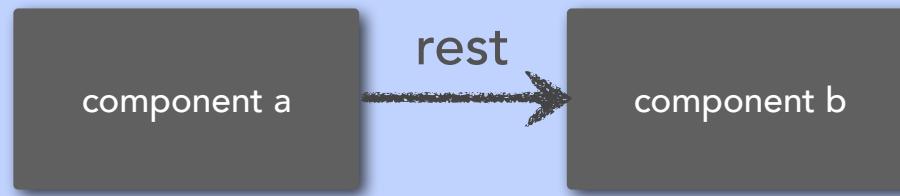
one component relies on the inner workings of another component



component coupling

external coupling

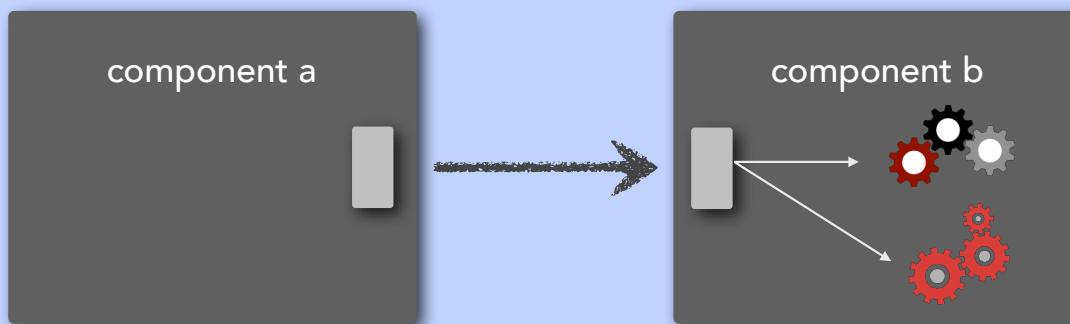
multiple components share an externally imposed protocol or data format



component coupling

control coupling

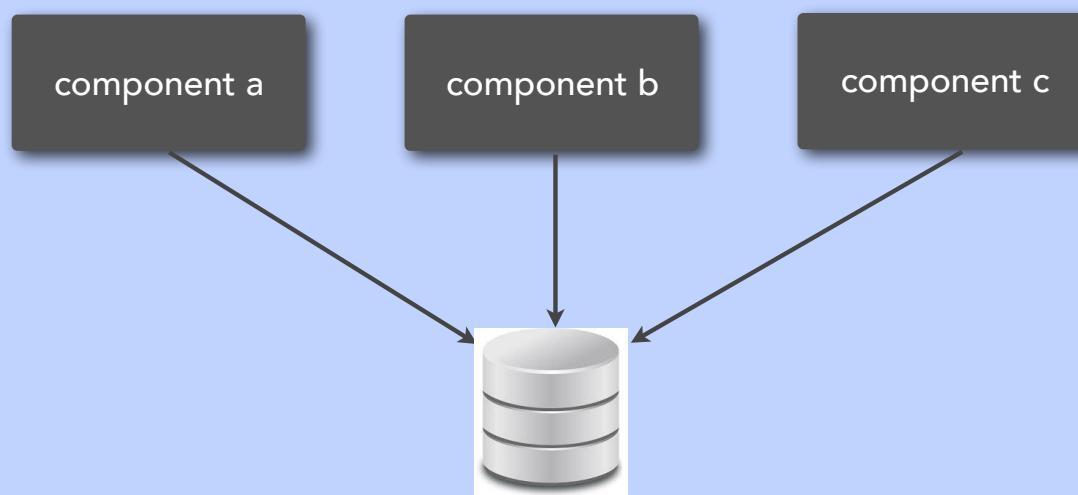
one component passes information to another component on what to do



component coupling

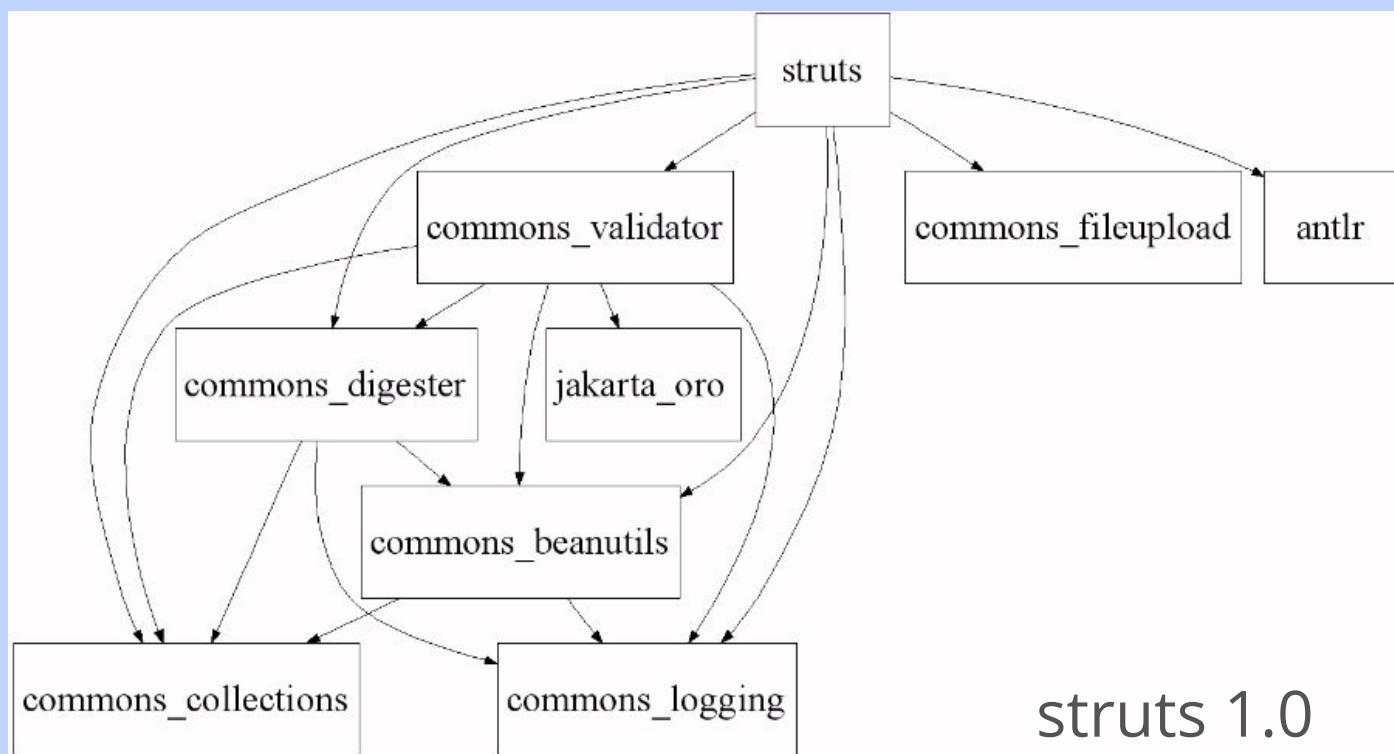
data coupling

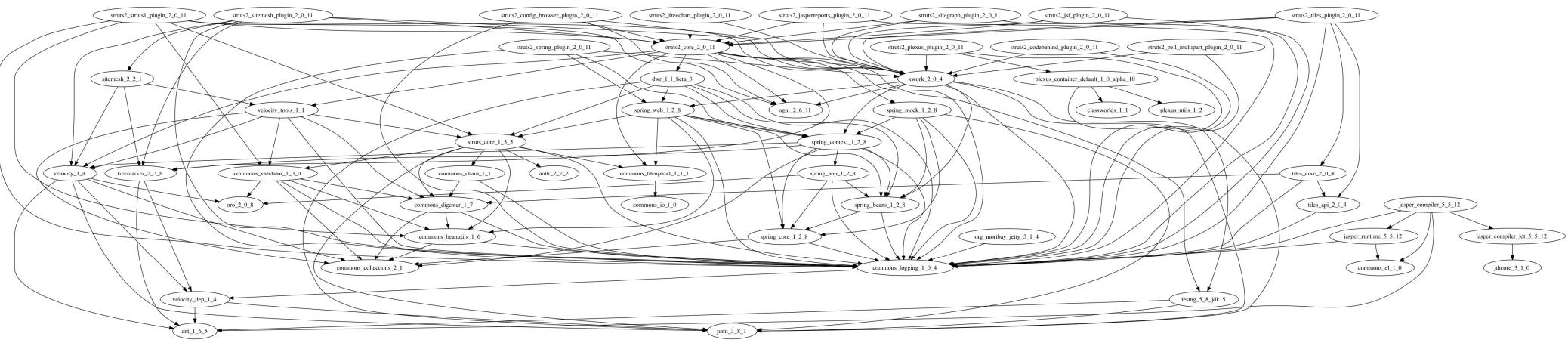
the degree to which components are
bound to a shared data context



component coupling

consequences of ignoring...





component cohesion

component cohesion

the degree and manner to which the operations of a component are related to one another

component cohesion

the degree and manner to which the operations of a component are related to one another



component cohesion

the degree and manner to which the operations of a component are related to one another

customer
maintenance

- add customer
- update customer
- get customer
- notify customer
- get customer orders
- cancel customer orders

component cohesion

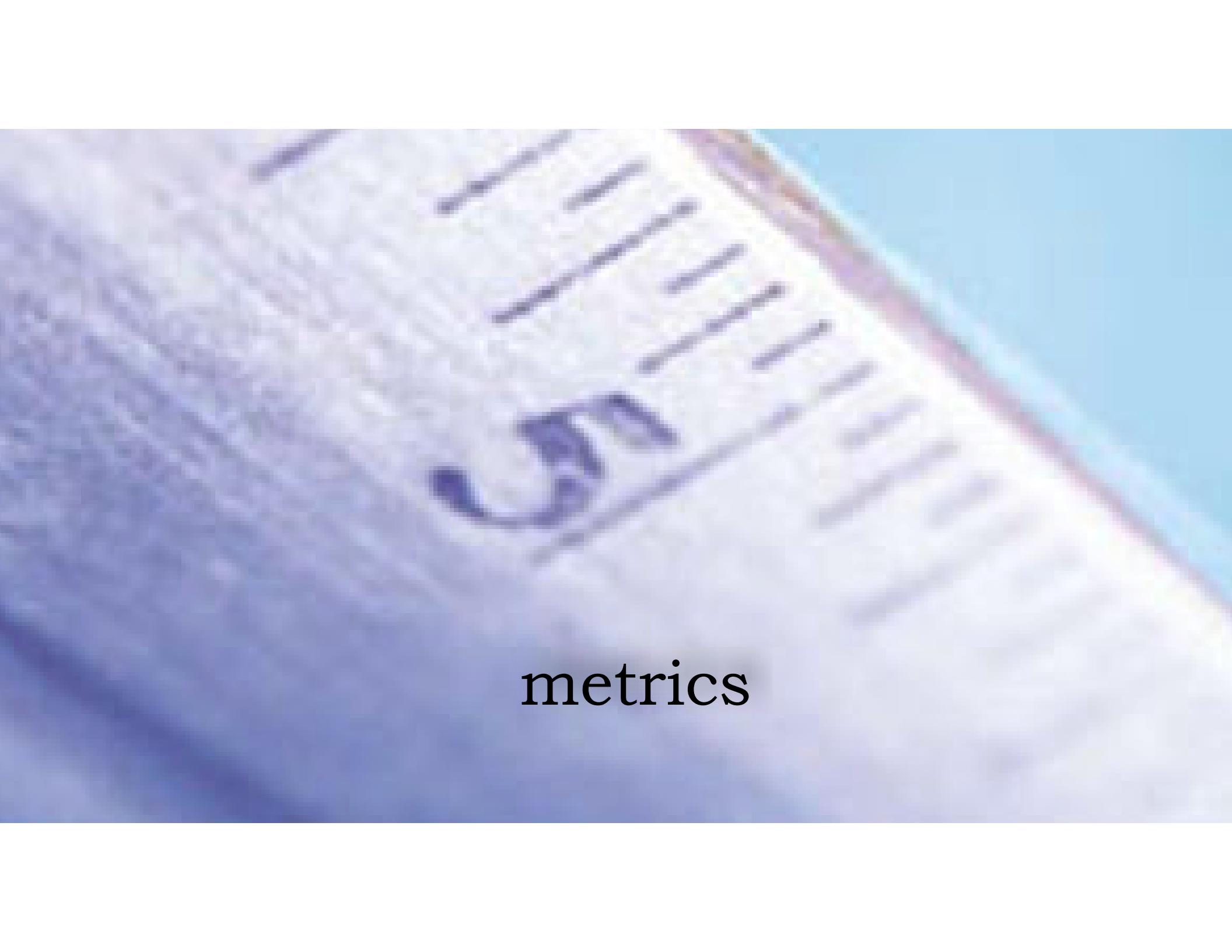
the degree and manner to which the operations of a component are related to one another

customer
maintenance

- add customer
- update customer
- get customer
- notify customer

order
maintenance

- get customer orders
- cancel customer orders

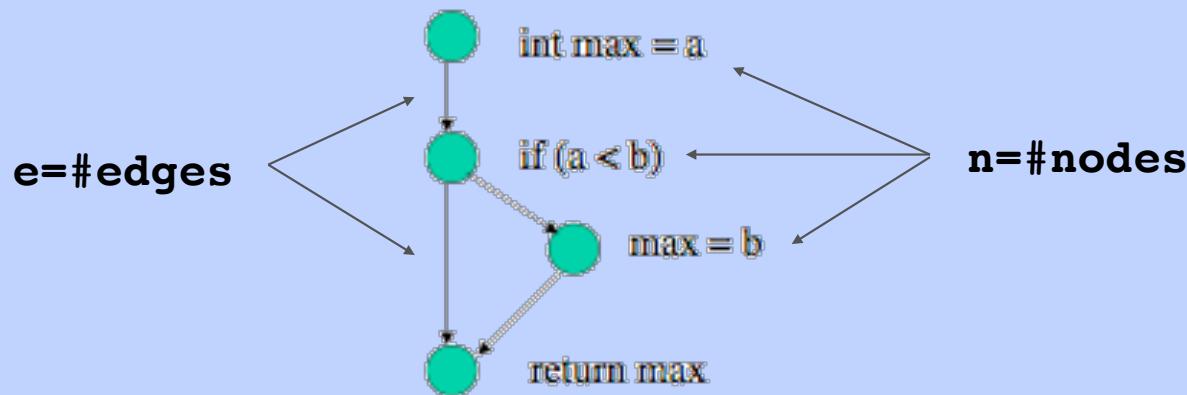
A blurry, out-of-focus photograph of a document or ledger page. The page features horizontal rows of lines for writing entries. A prominent blue header at the top left contains the word "BALANCE". The rest of the page is mostly illegible due to the blur.

metrics

metrics and structural decay

cyclomatic complexity

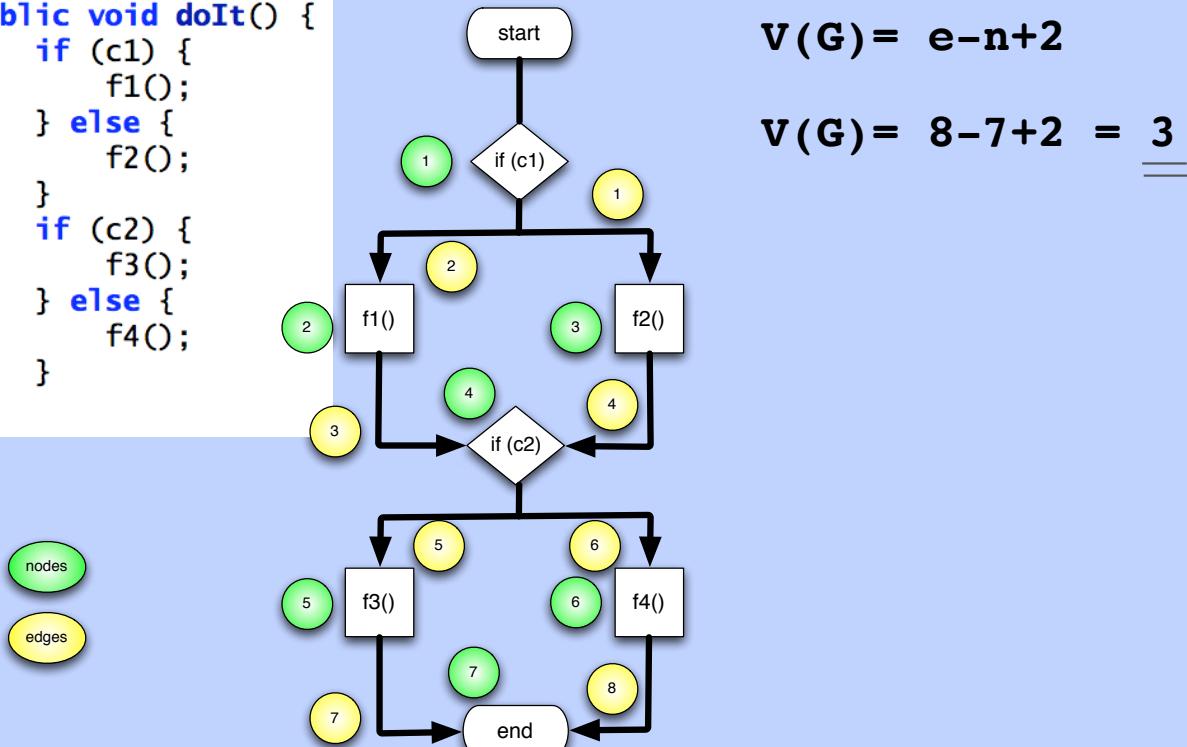
$$V(G) = e - n + 2$$



metrics and structural decay

cyclomatic complexity

```
public void doIt() {  
    if (c1) {  
        f1();  
    } else {  
        f2();  
    }  
    if (c2) {  
        f3();  
    } else {  
        f4();  
    }  
}
```



$$V(G) = e - n + 2$$

$$V(G) = 8 - 7 + 2 = 3$$

metrics and structural decay

core metrics

- ✓ number of classes per package
- ✓ number of lines of source code per package
- ✓ percent comments (range: 8-20)
- ✓ max complexity (1+num_paths thru method; range: 2-8)
- ✓ average complexity (range: 2.0 - 4.0)

metrics and structural decay

Chidamber & Kemerer Metrics

- ✓ DIT (depth of inheritance tree)
- ✓ WMC (weighted methods/class; sum of CC)
- ✓ CE (efferent coupling count)
- ✓ CA (afferent coupling count)

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

$$LCOM96b = \frac{1}{a} \sum_{j=1}^a m - \mu(Aj)$$

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero.

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

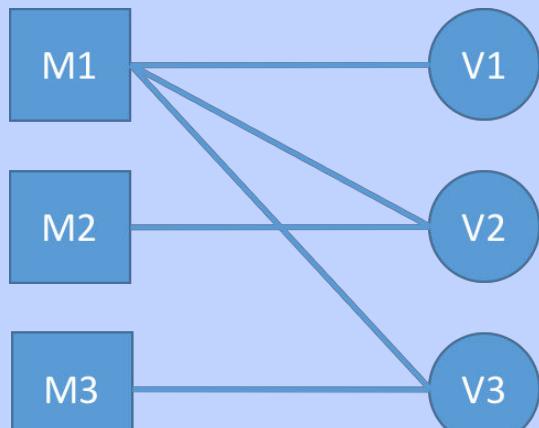
The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero.

The sum of sets of methods not shared via sharing fields.

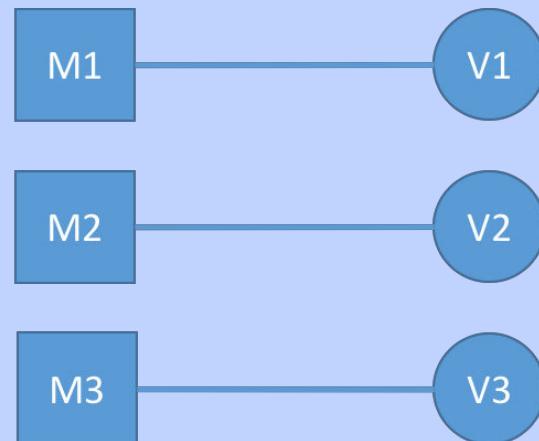
metrics and structural decay

Chidamber & Kemerer Metrics

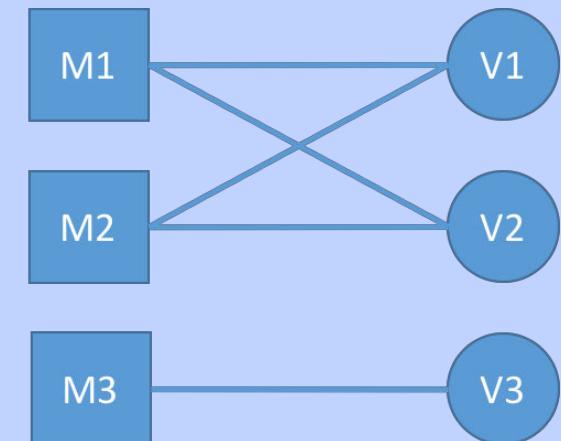
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

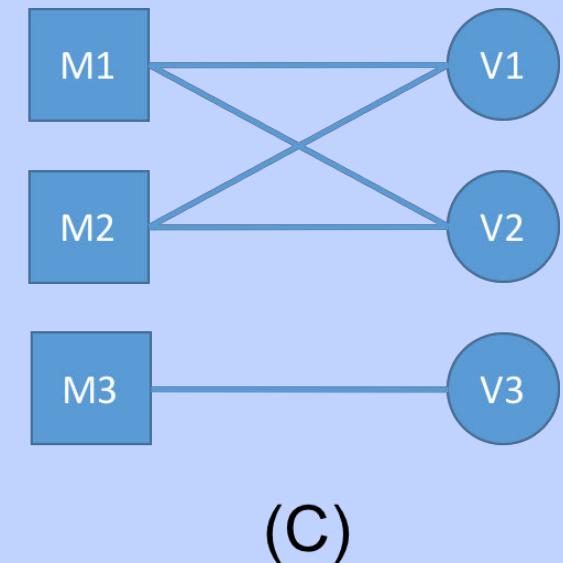
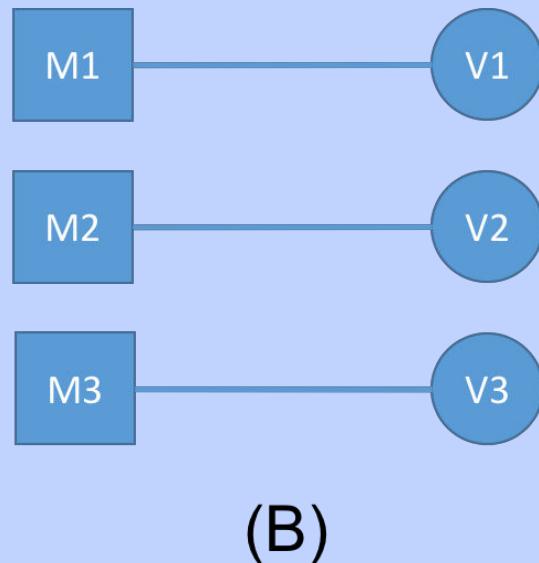
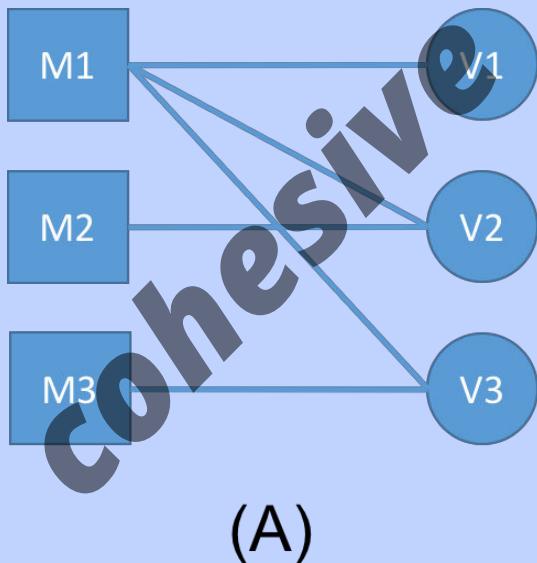


(C)

metrics and structural decay

Chidamber & Kemerer Metrics

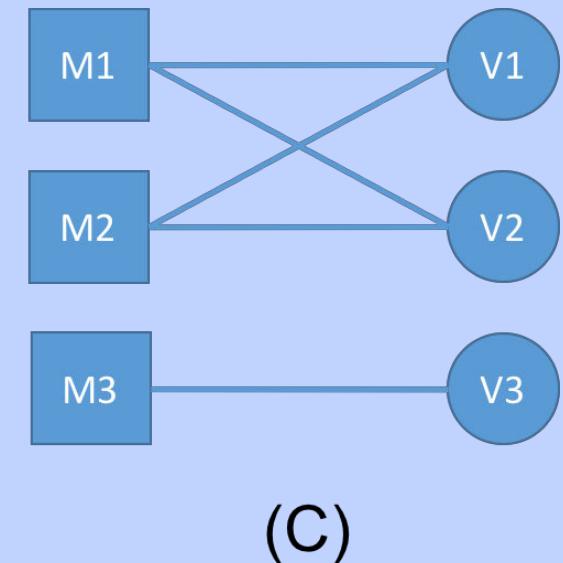
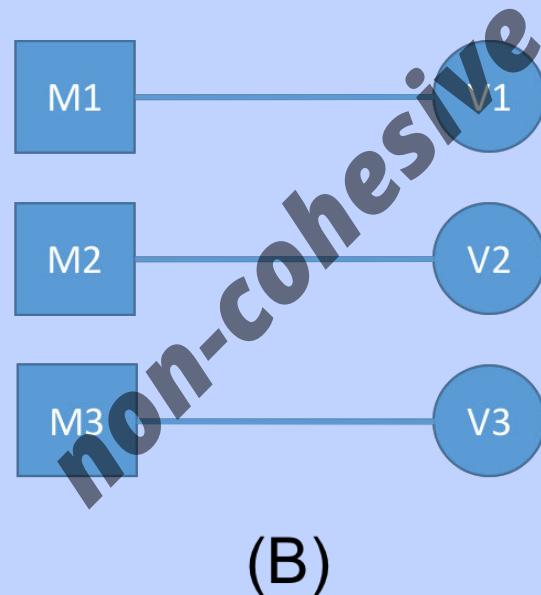
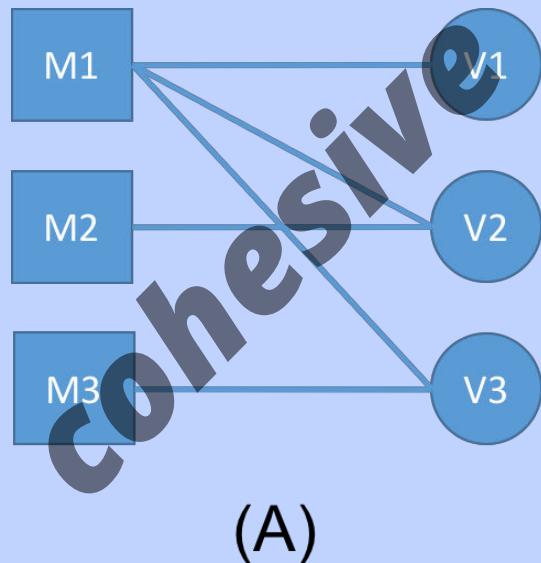
✓ LCOM (Lack of Cohesion in Methods)



metrics and structural decay

Chidamber & Kemerer Metrics

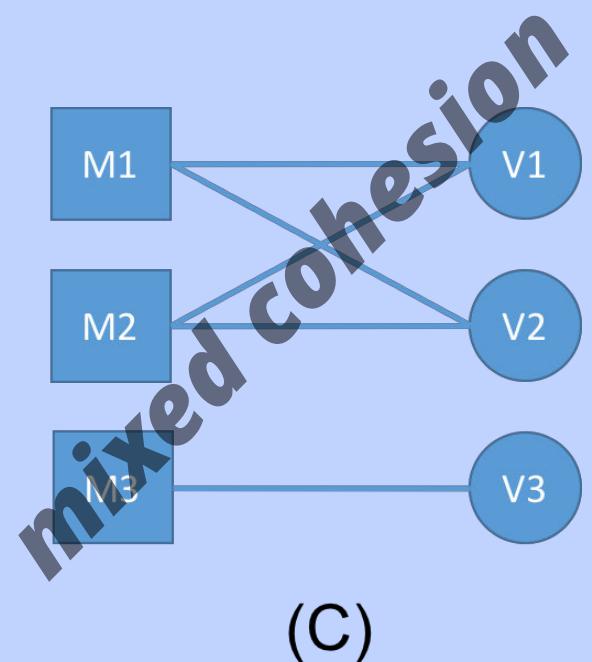
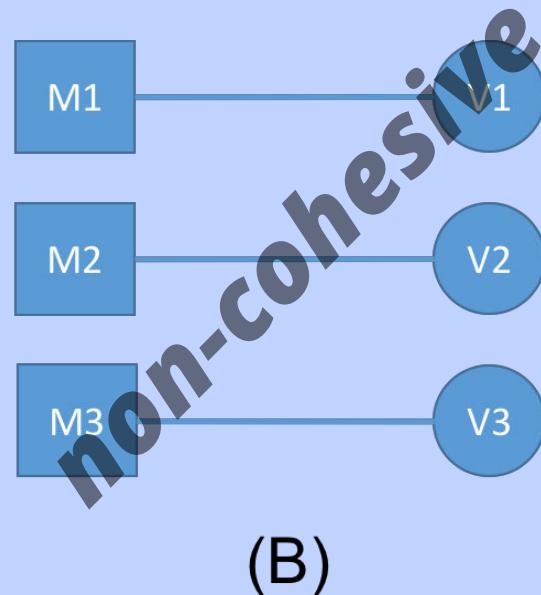
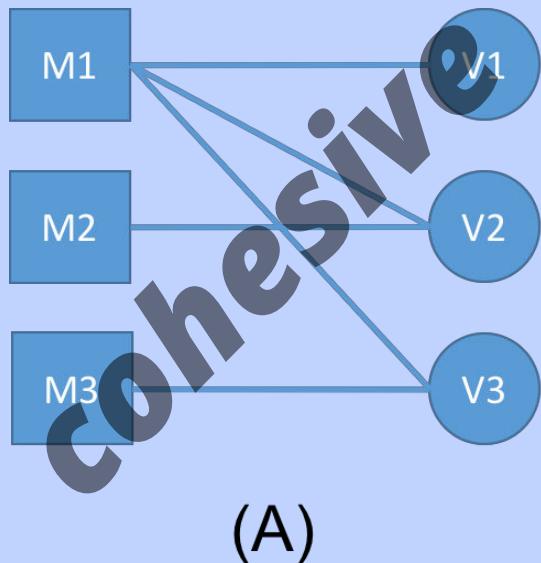
✓ LCOM (Lack of Cohesion in Methods)



metrics and structural decay

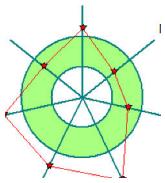
Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

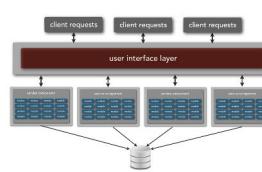


metrics and structural decay

architecture characteristics mapping



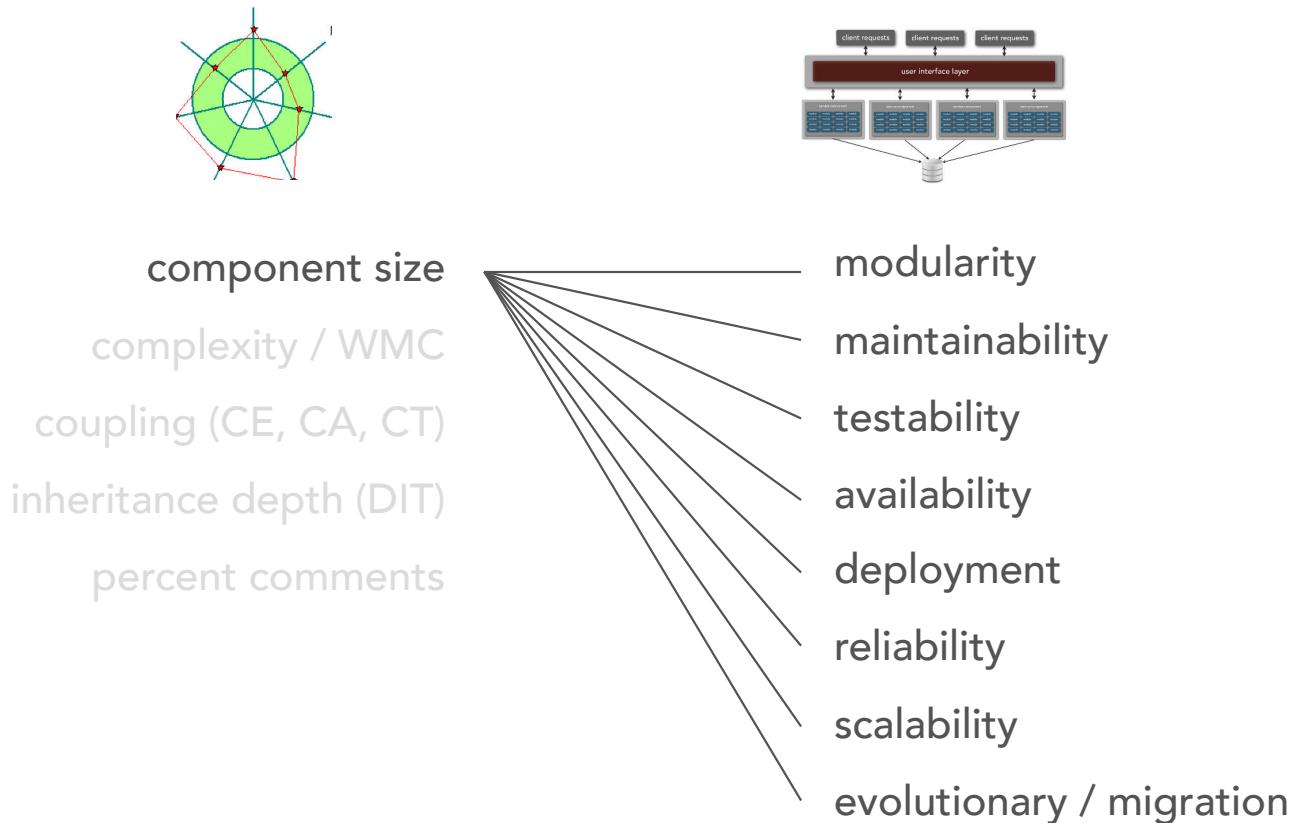
component size
complexity / WMC
coupling (CE, CA, CT)
inheritance depth (DIT)
percent comments



modularity
maintainability
testability
availability
deployment
reliability
scalability
evolutionary / migration

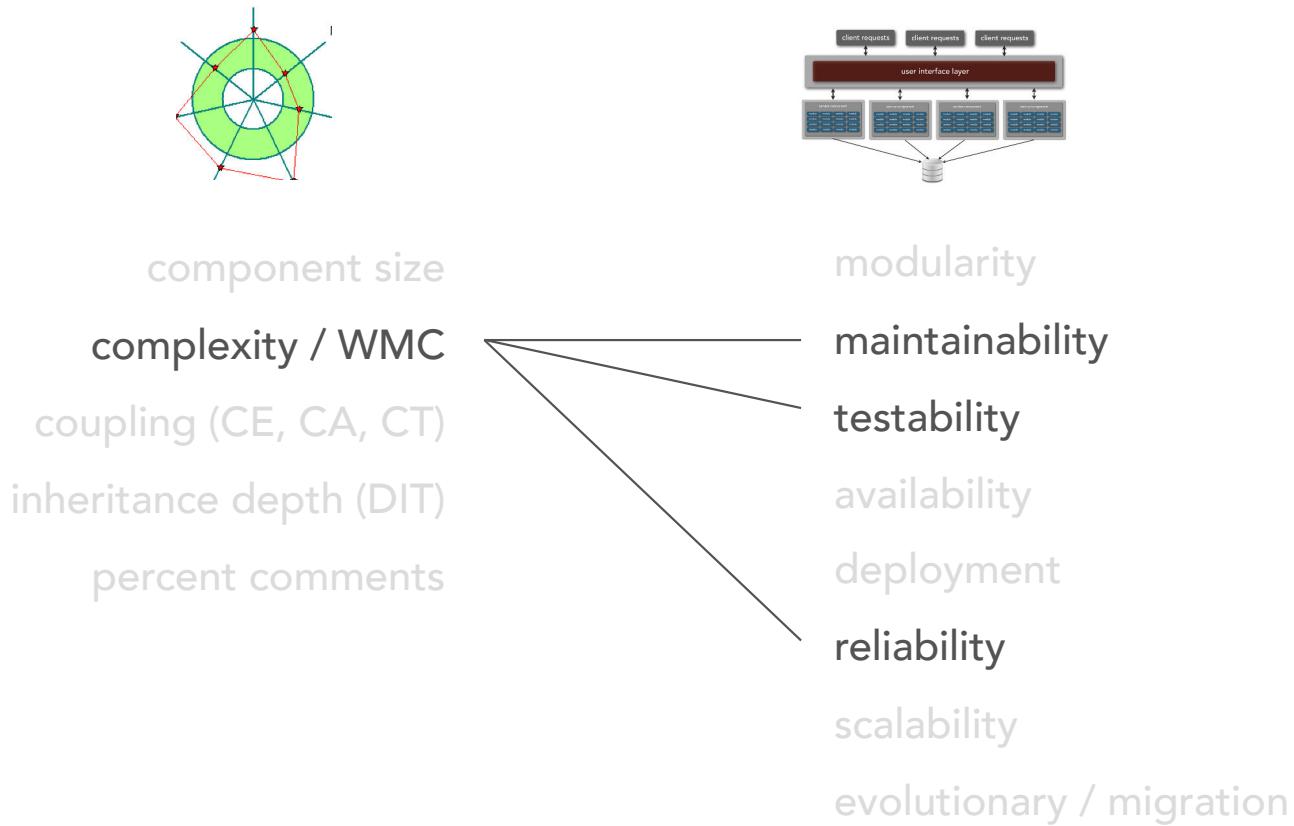
metrics and structural decay

architecture characteristics mapping



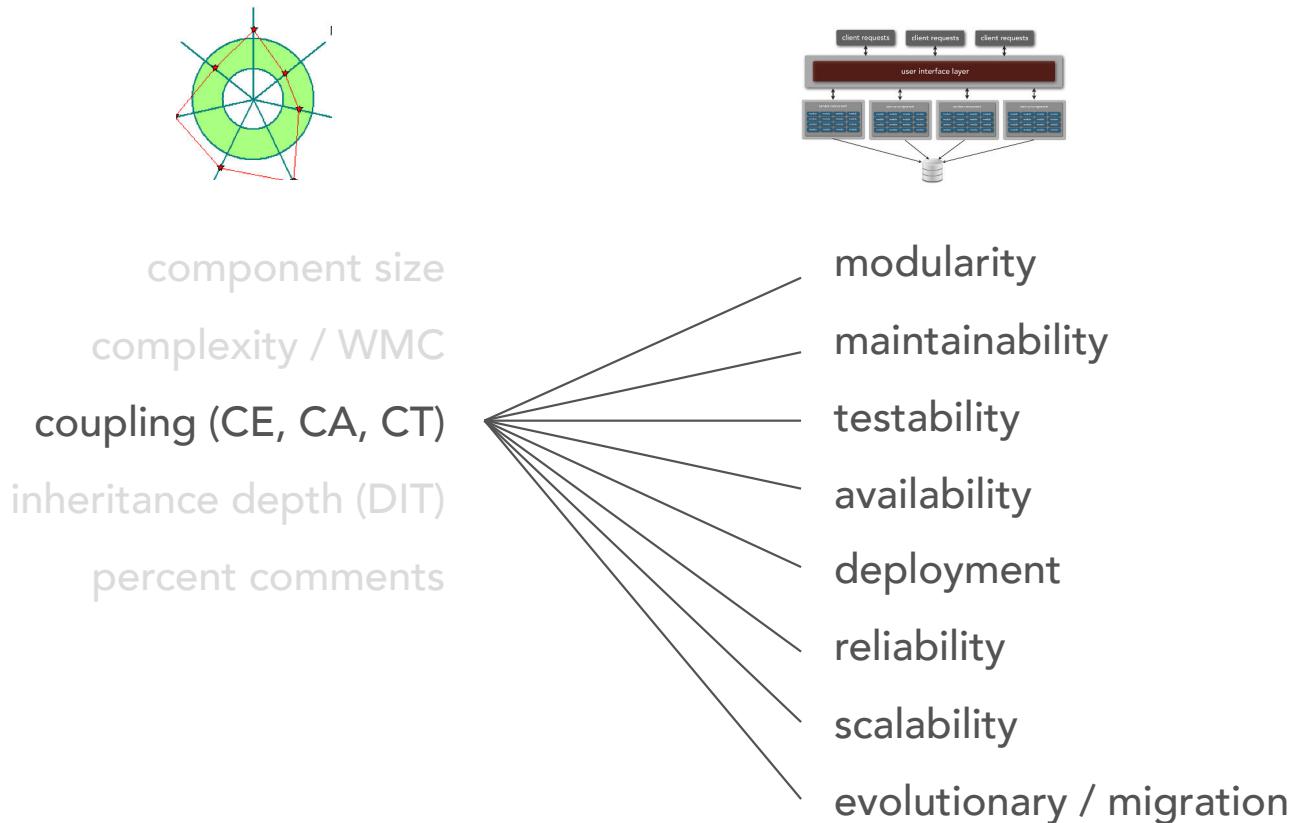
metrics and structural decay

architecture characteristics mapping



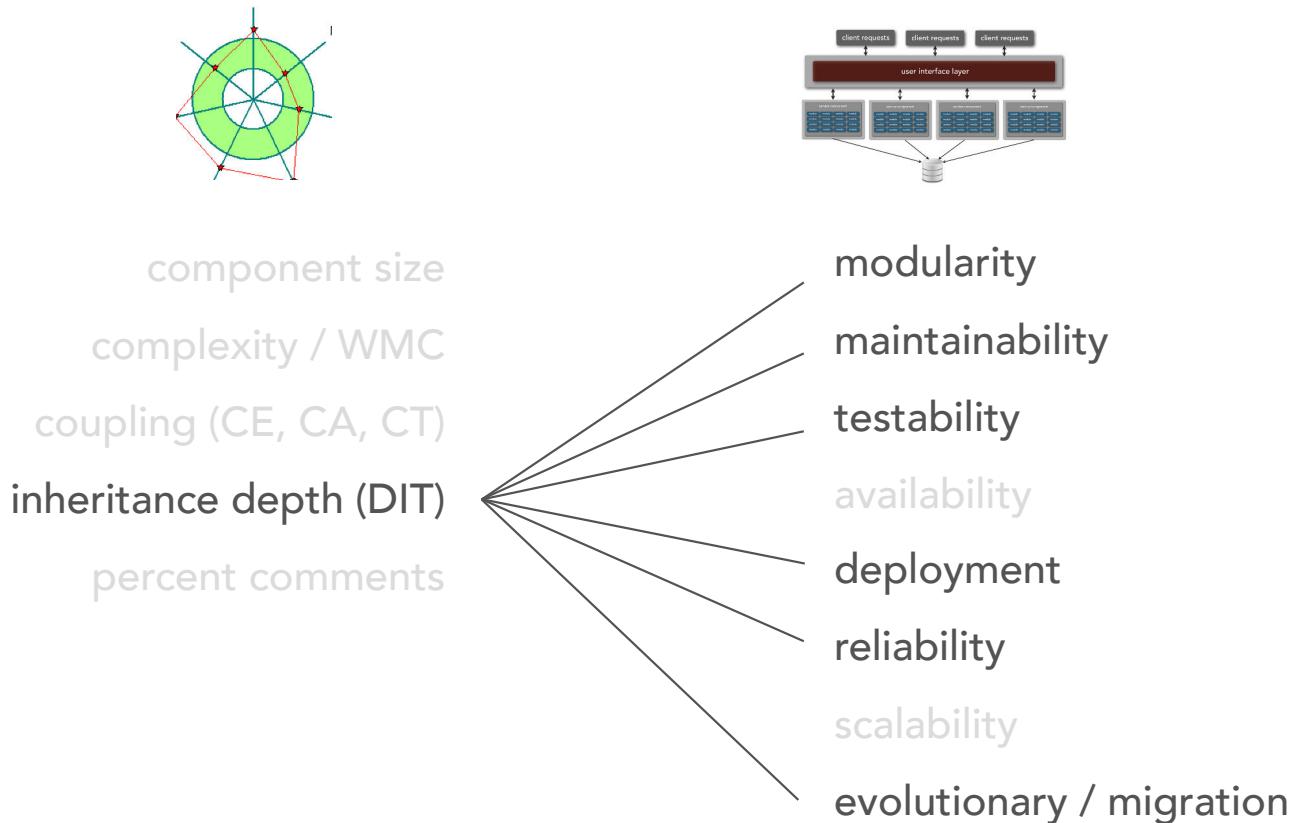
metrics and structural decay

architecture characteristics mapping



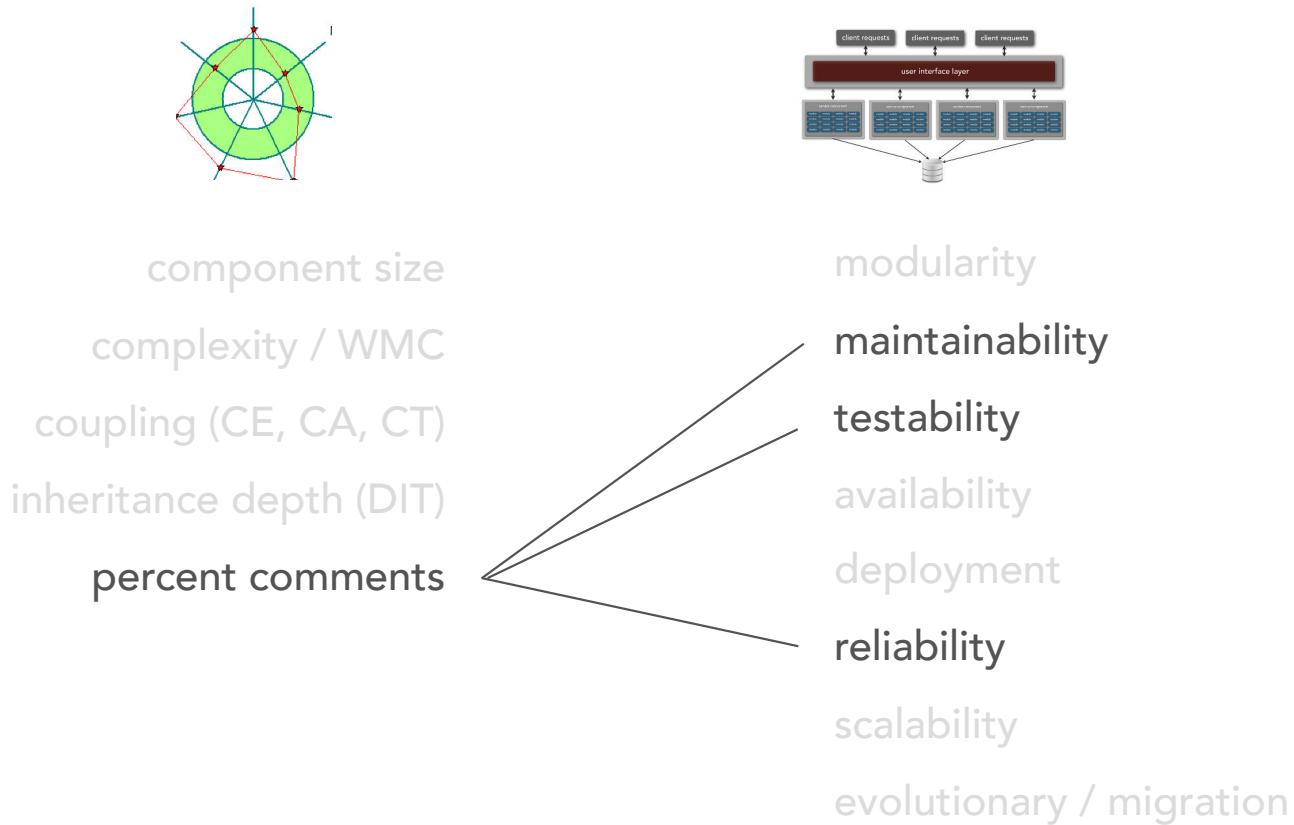
metrics and structural decay

architecture characteristics mapping

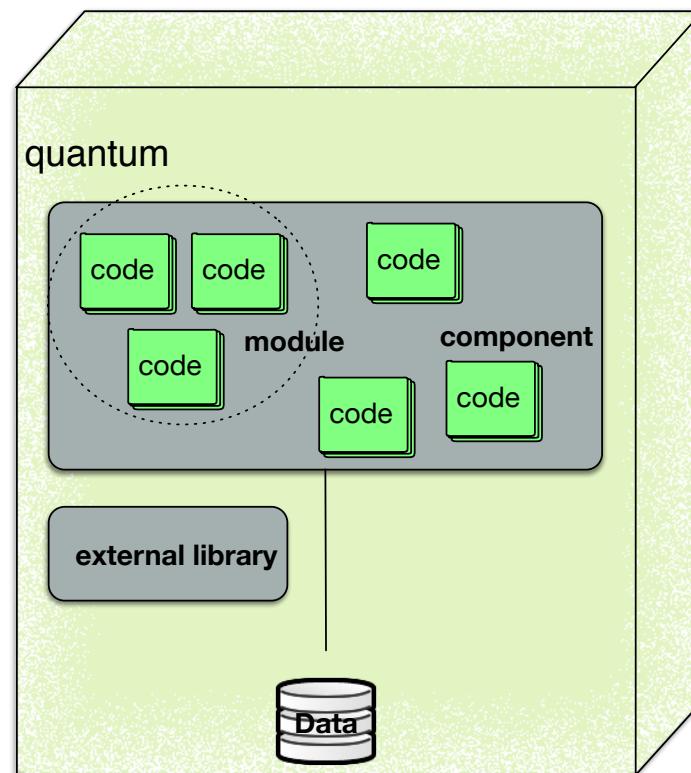


metrics and structural decay

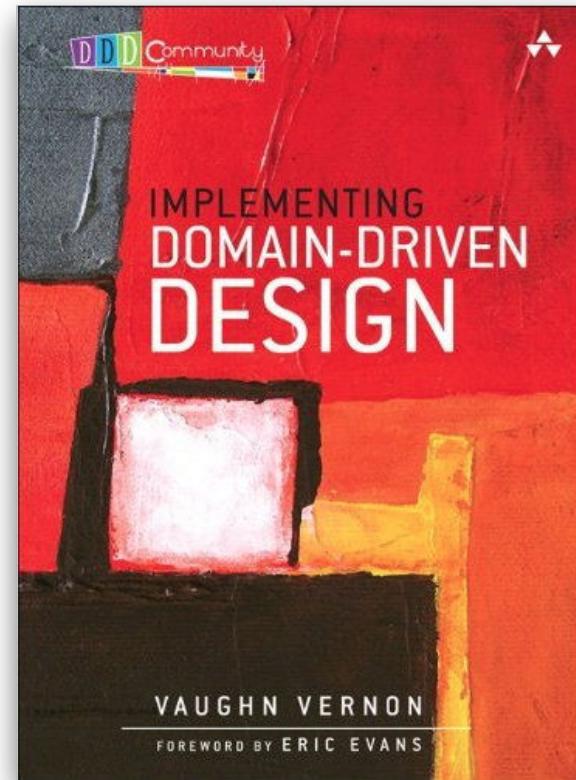
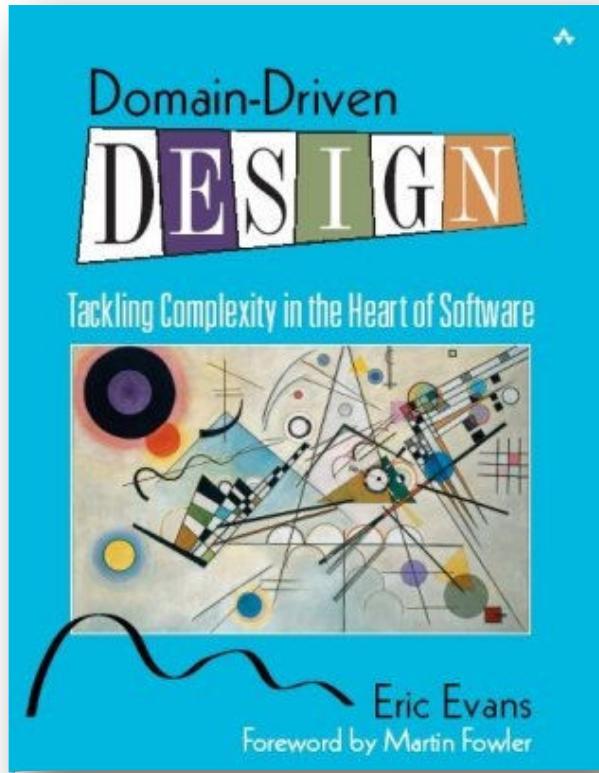
architecture characteristics mapping

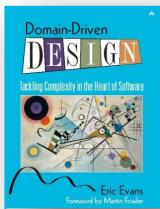


Architectural Quantum



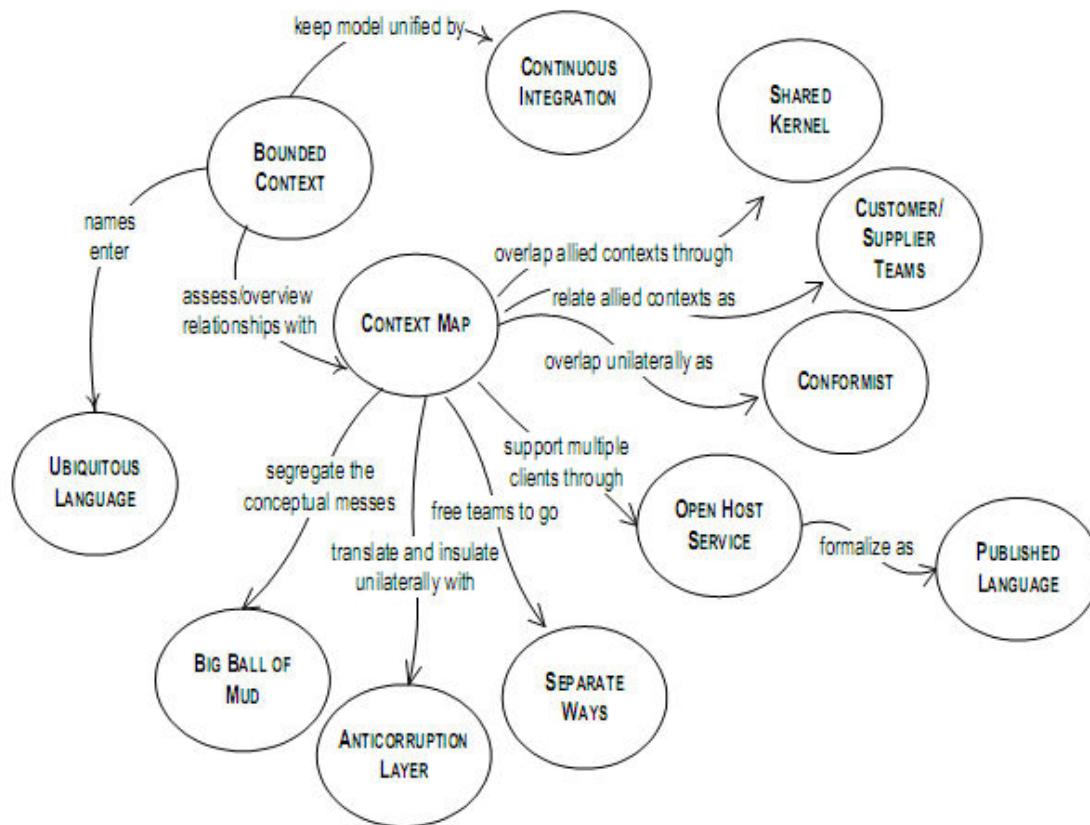
Domain Driven Design





Bounded Context

Maintaining Model Integrity

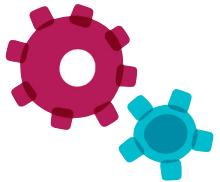


Architectural Quantum

An architectural quantum is an independently deployable component with high functional cohesion, which includes all the structural elements required for the system to function properly.

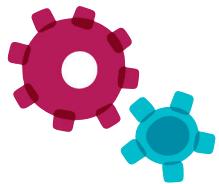
Why Quantum?

Why Quantum?

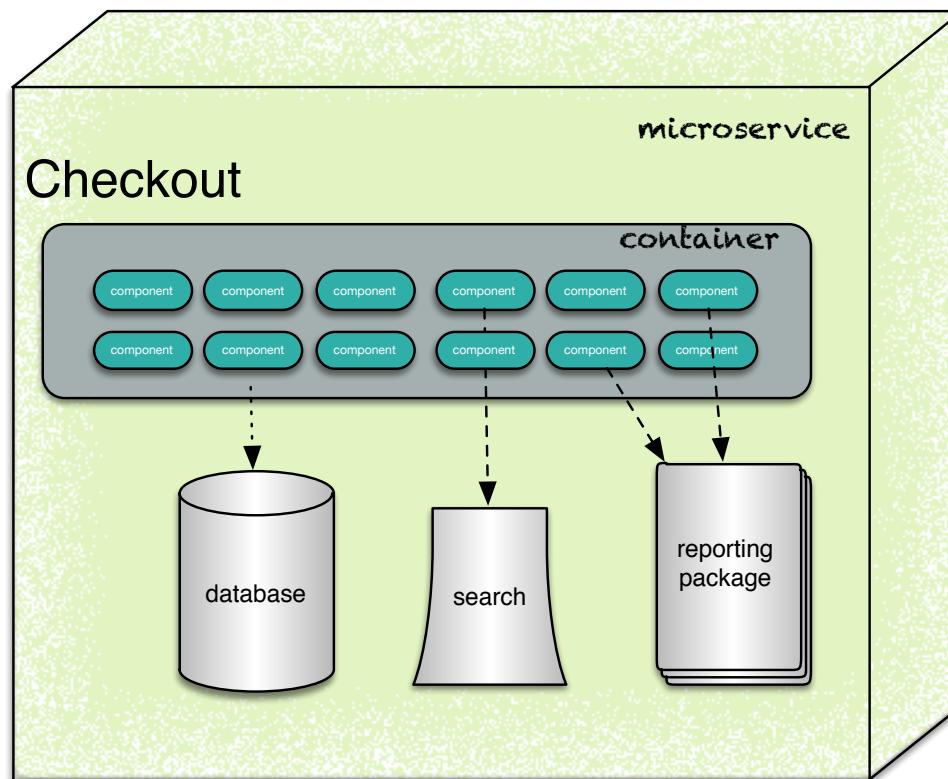


operational view
of architecture

Why Quantum?



operational view
of architecture



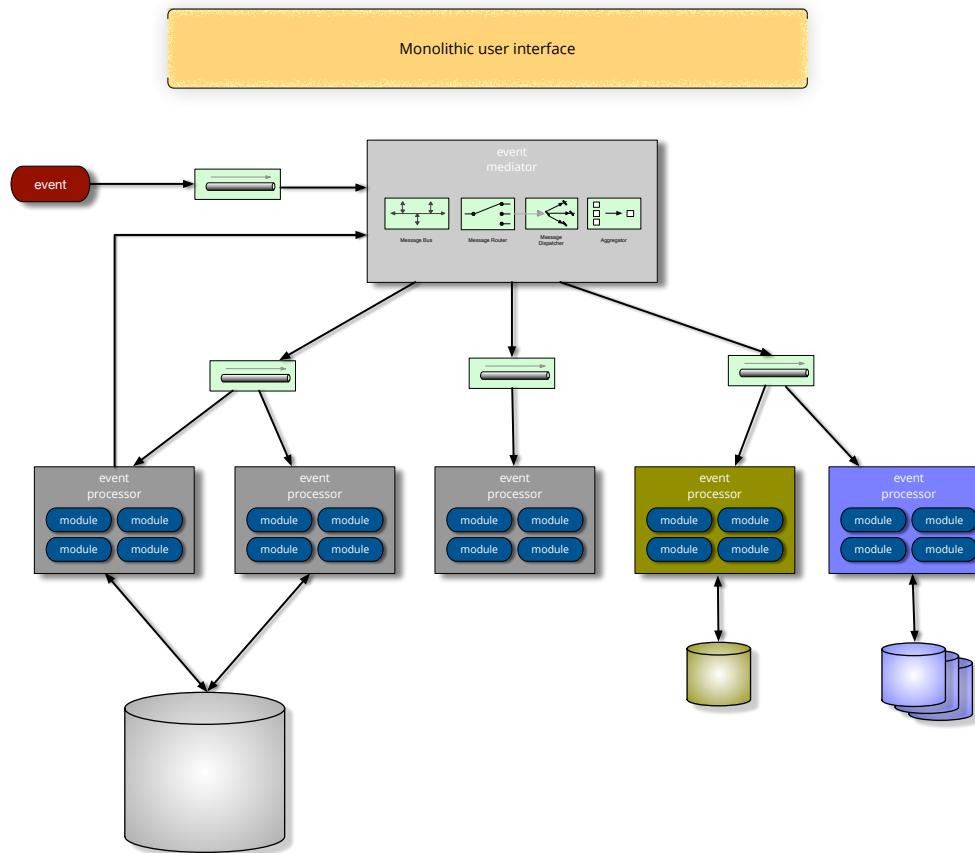
Why Quantum?

holistic



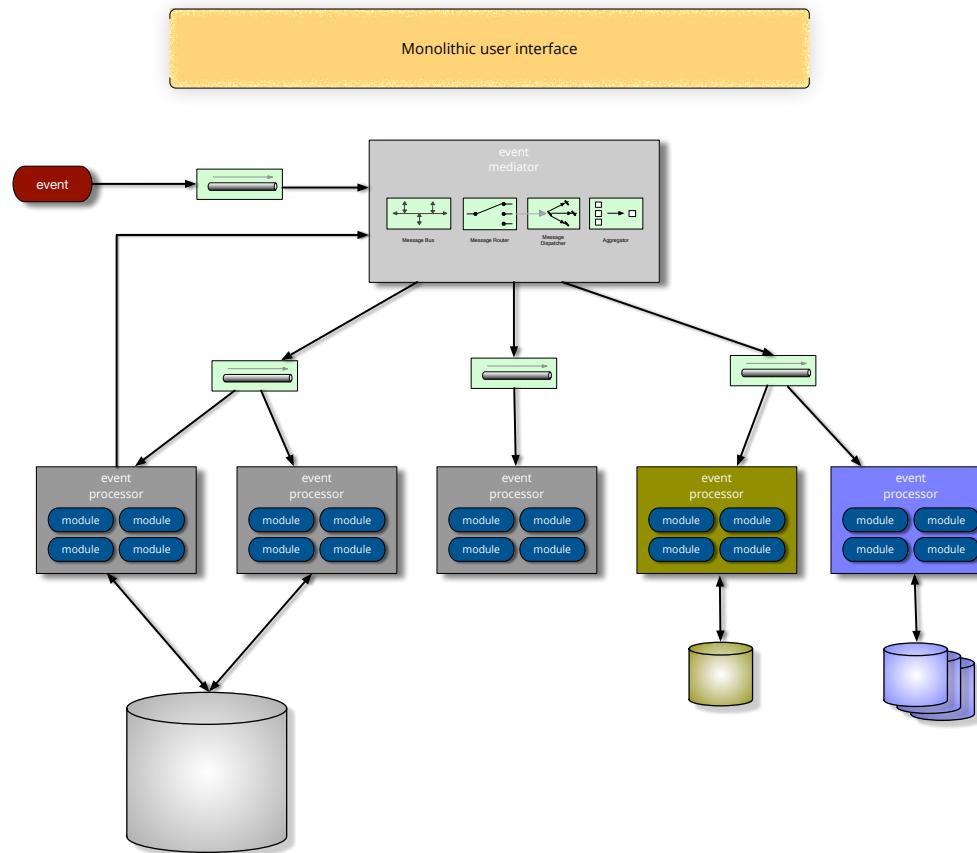
holistic

Why Quantum?



holistic

Why Quantum?



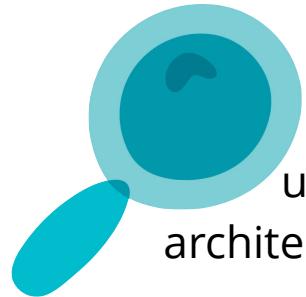
"multiple dimensions"



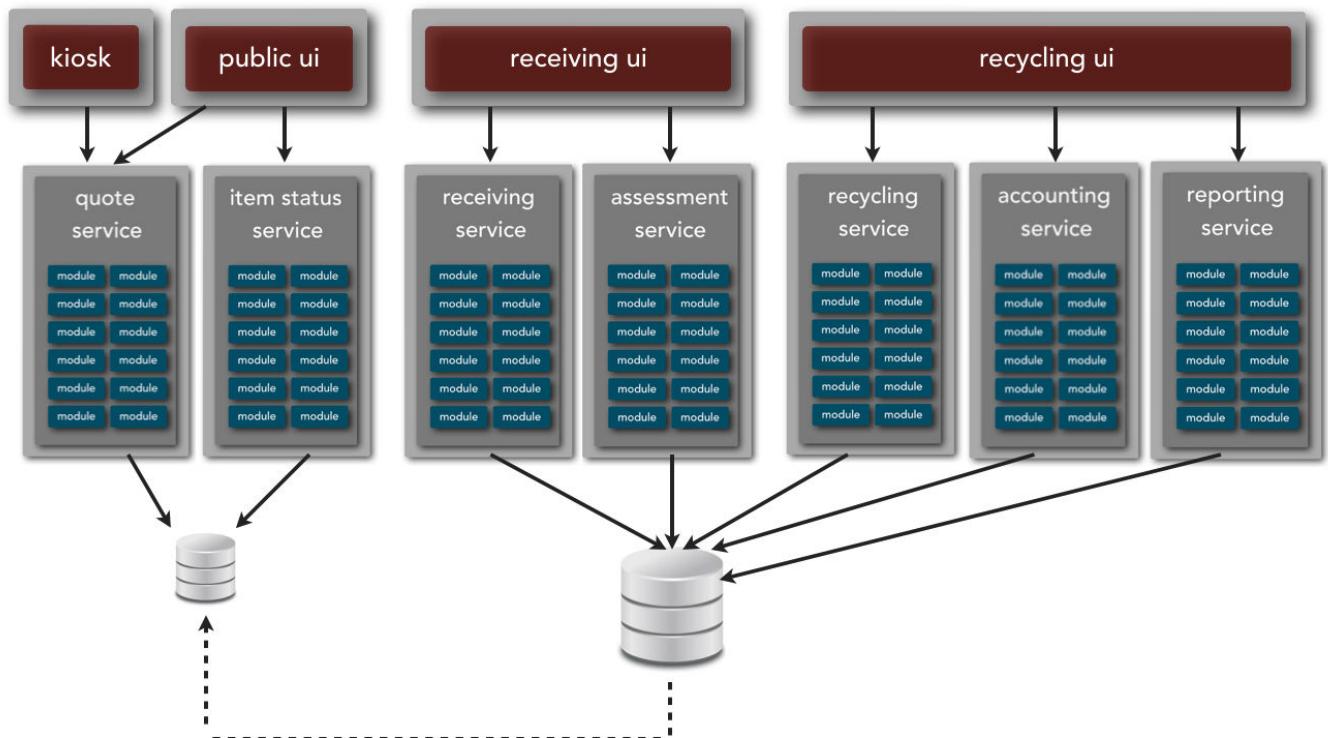
Why Quantum?

useful for
architectural analysis

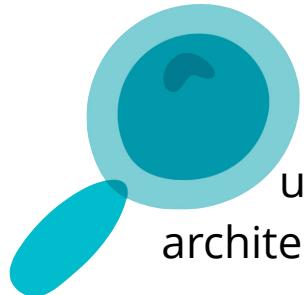
Why Quantum?



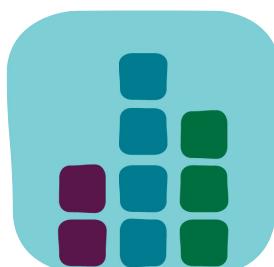
useful for
architectural analysis



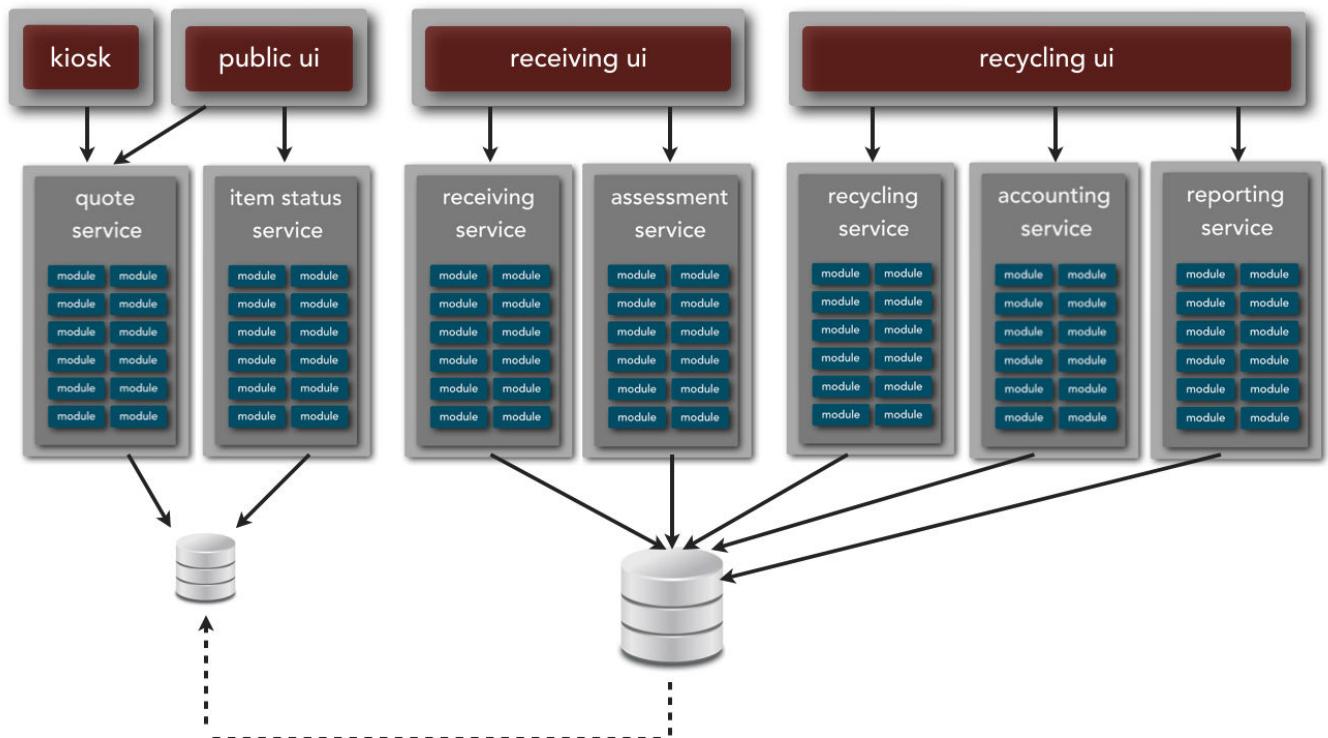
Why Quantum?



useful for
architectural analysis

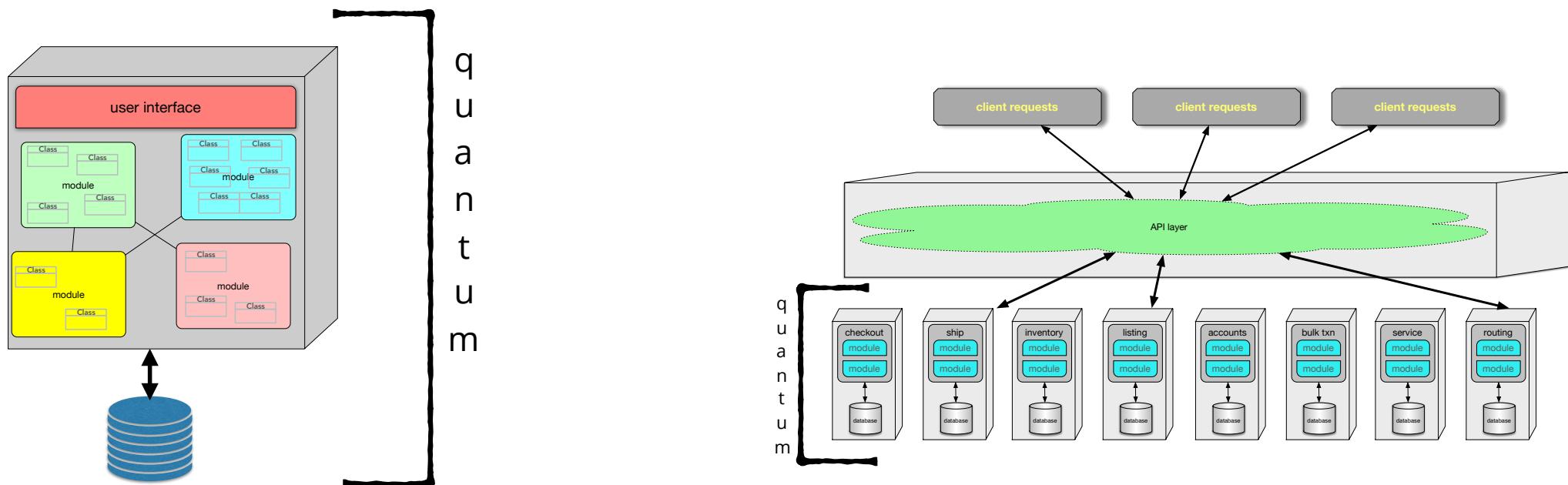


*helps analyze
coupling*

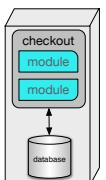
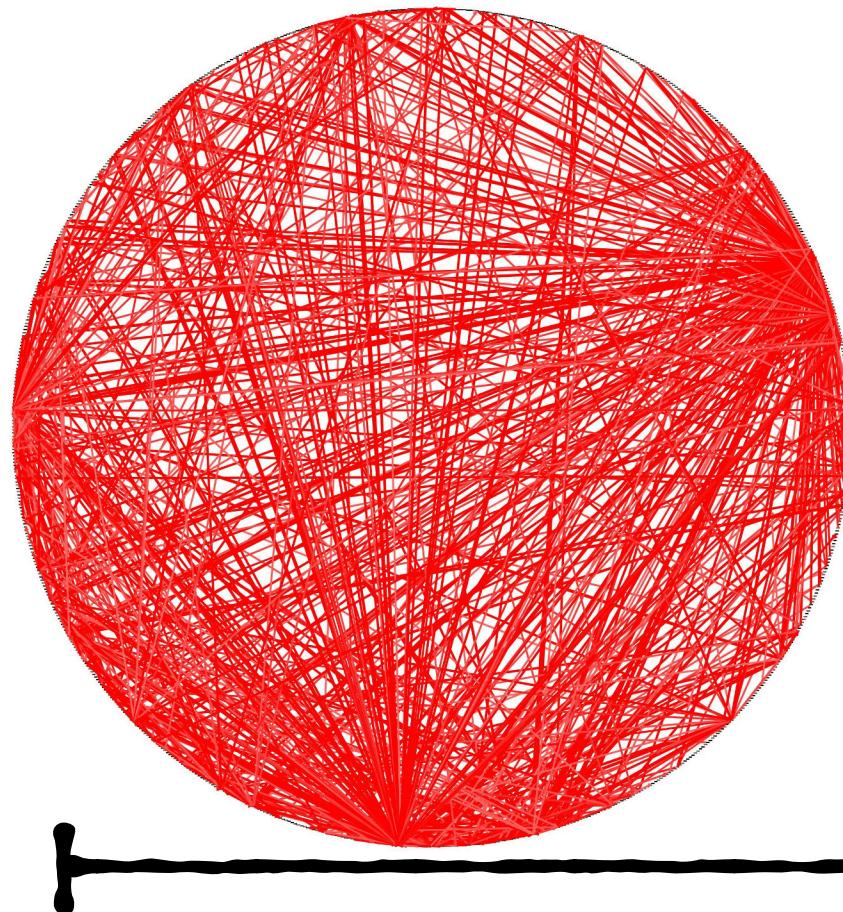


Why Quantum?

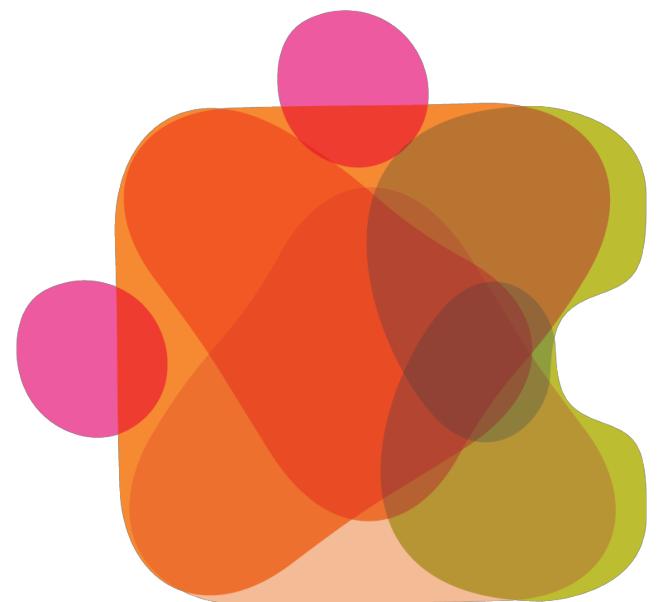
The quantum is where architectural characteristics live.



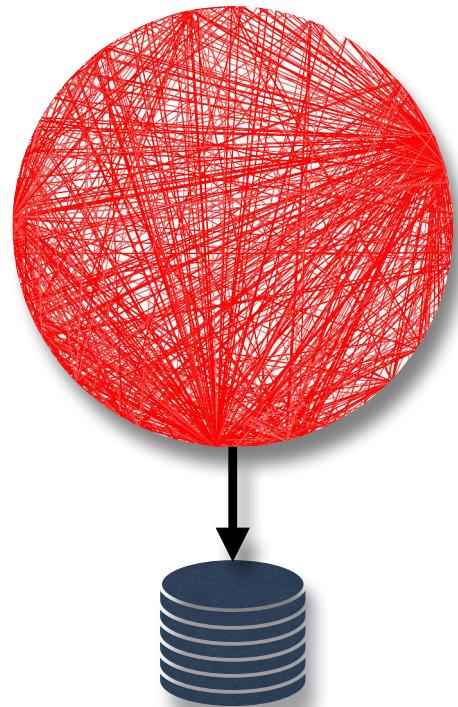
Quantum: Large to Small



Monoliths

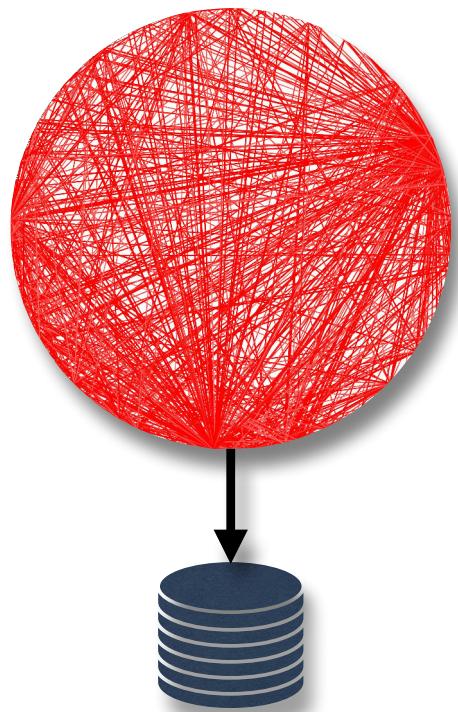


Monoliths

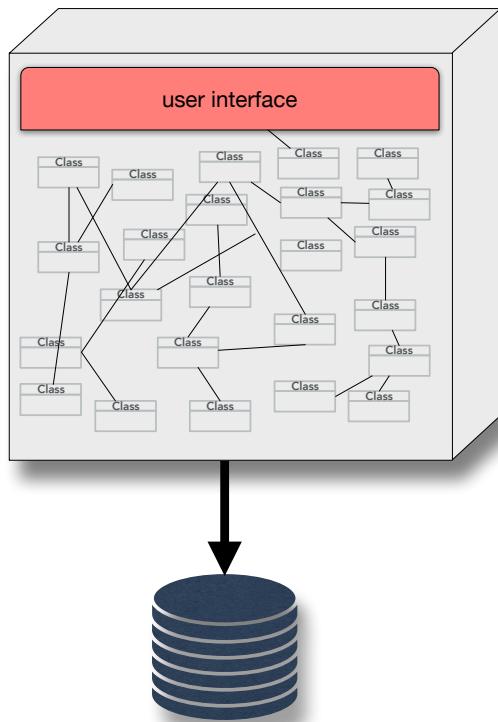


“Big Ball of Mud”

https://en.wikipedia.org/wiki/Big_ball_of_mud



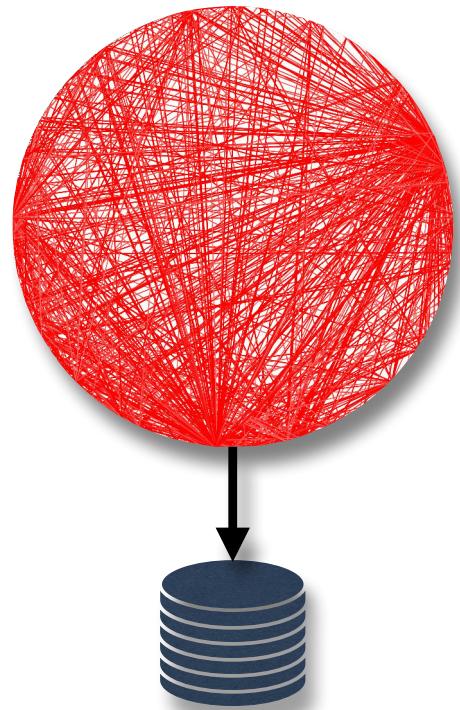
Monoliths



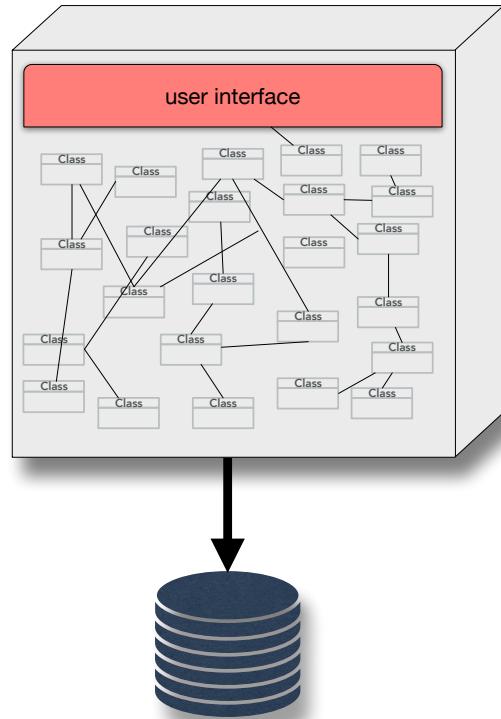
unstructured
monolith

"Big Ball of Mud"

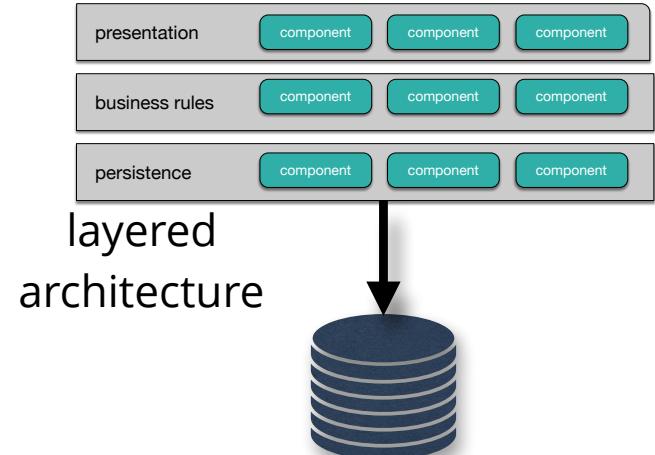
https://en.wikipedia.org/wiki/Big_ball_of_mud



Monoliths

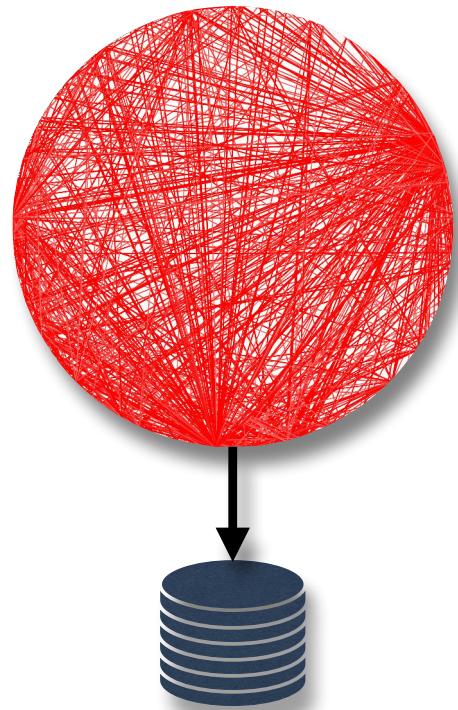


unstructured
monolith



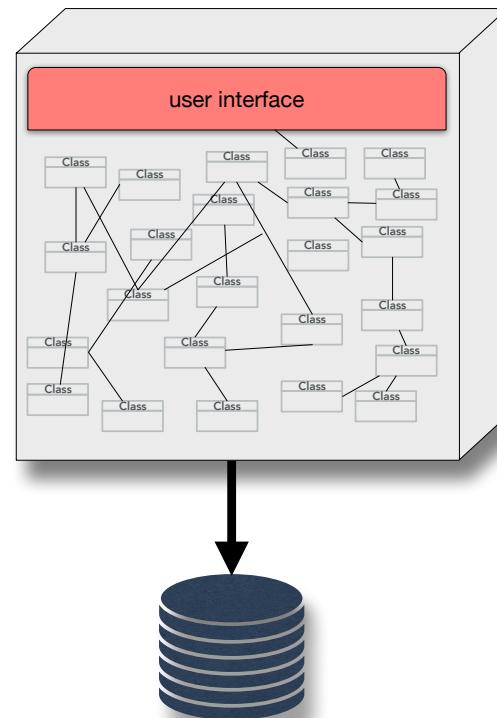
layered
architecture

"Big Ball of Mud"
https://en.wikipedia.org/wiki/Big_ball_of_mud



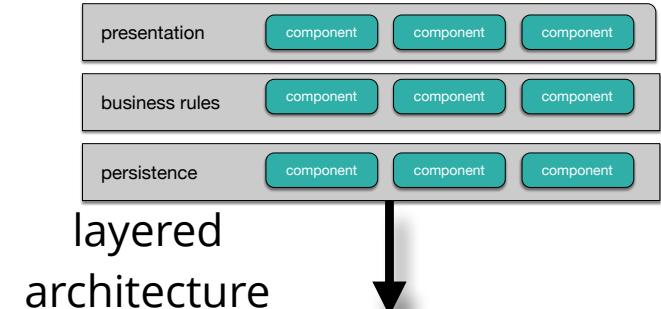
"Big Ball of Mud"

https://en.wikipedia.org/wiki/Big_ball_of_mud

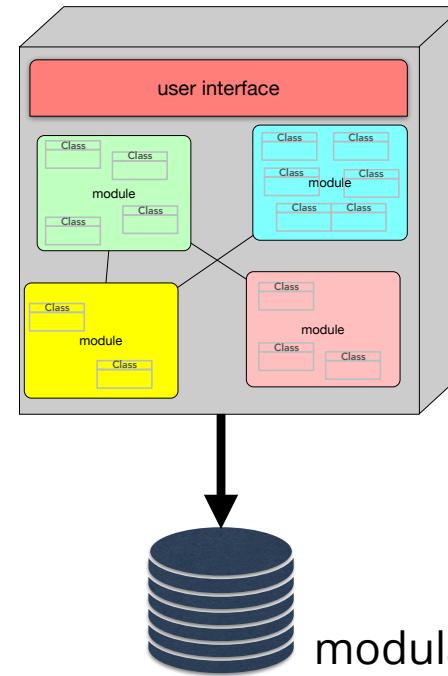


unstructured
monolith

Monoliths

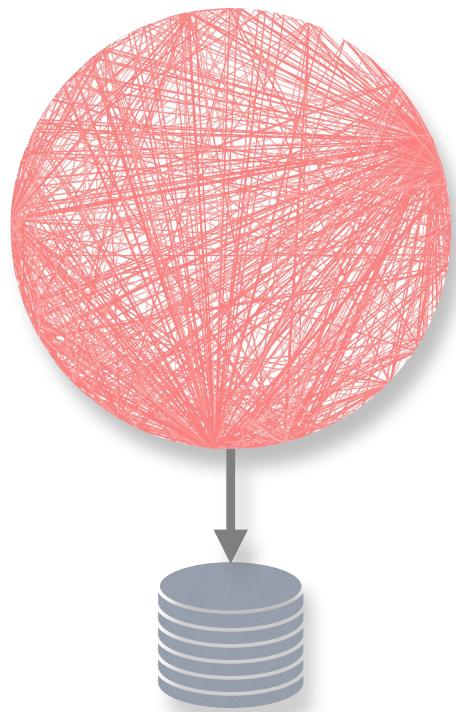


layered
architecture



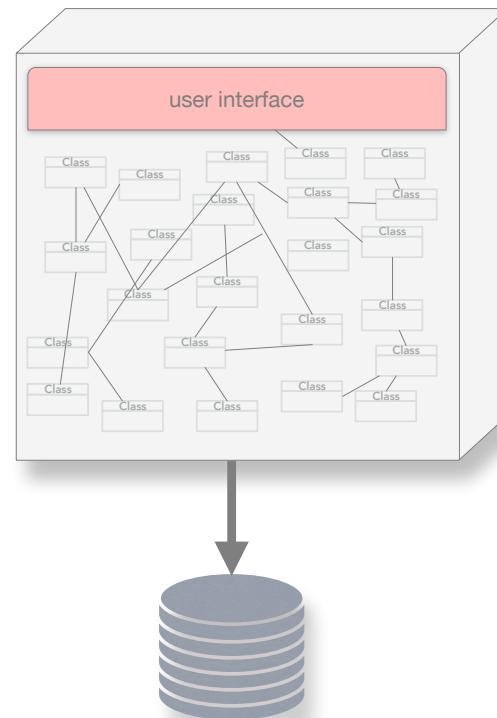
modular monolith

<http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths>



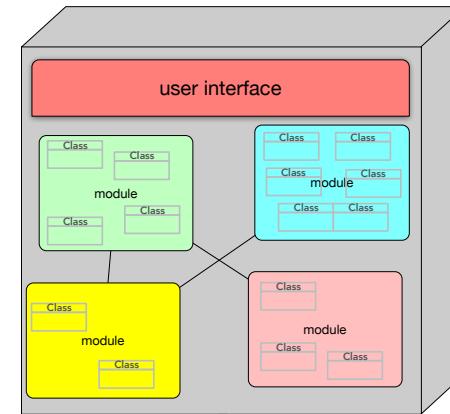
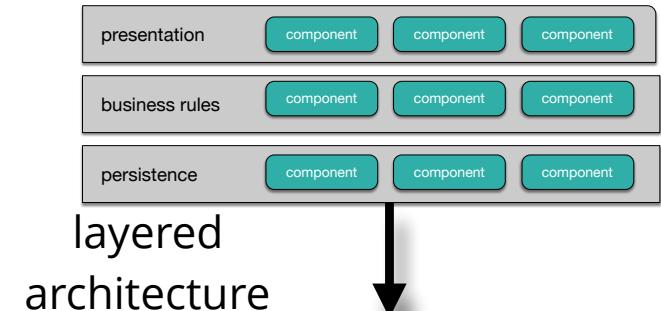
"Big Ball of Mud"

https://en.wikipedia.org/wiki/Big_ball_of_mud



unstructured
monolith

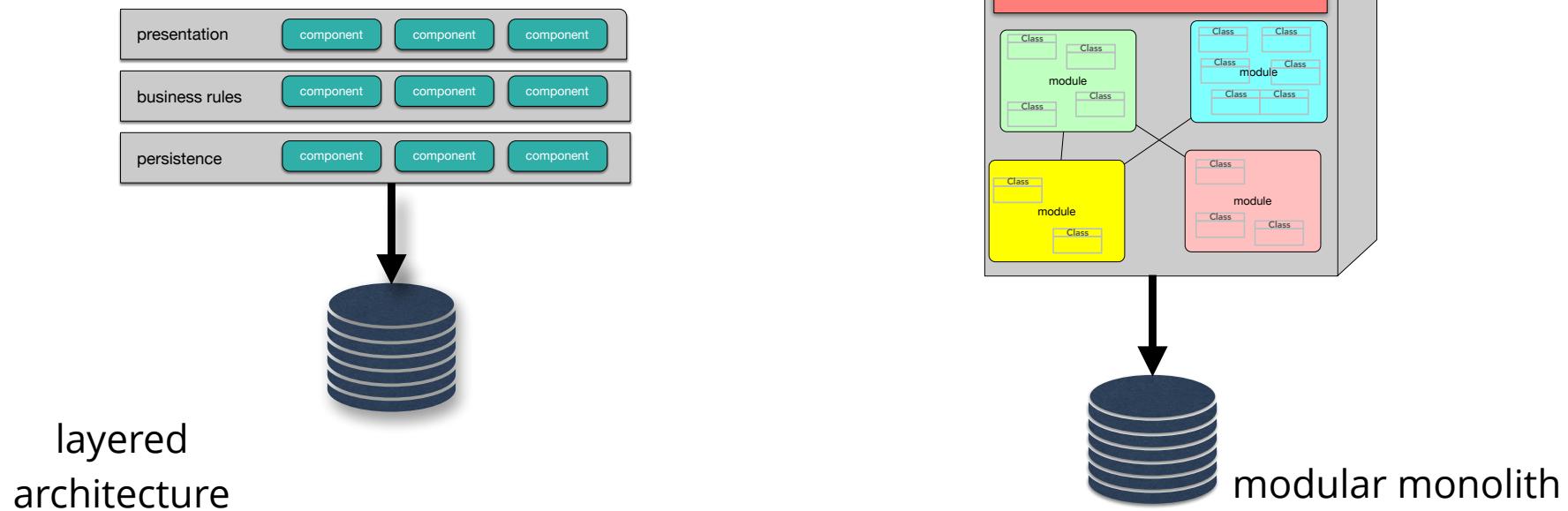
Monoliths



modular monolith

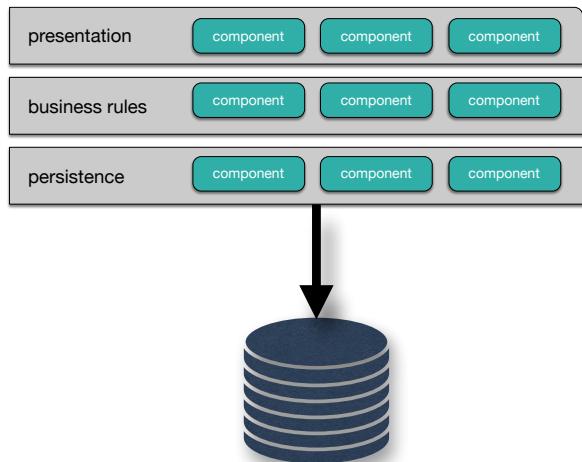
<http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths>

Monoliths

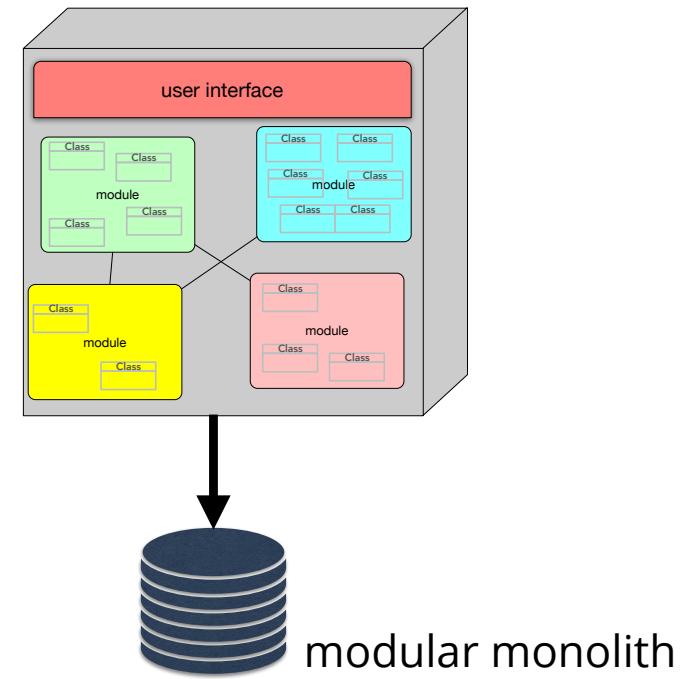


Monoliths

technical partitioning



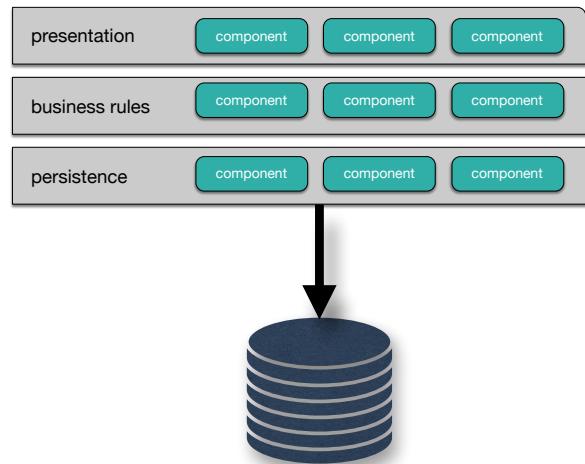
layered
architecture



modular monolith

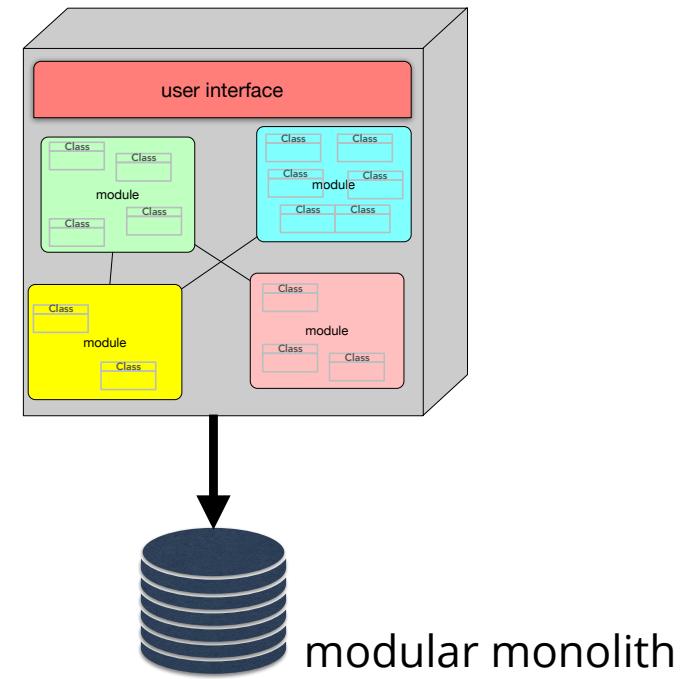
Monoliths

technical partitioning

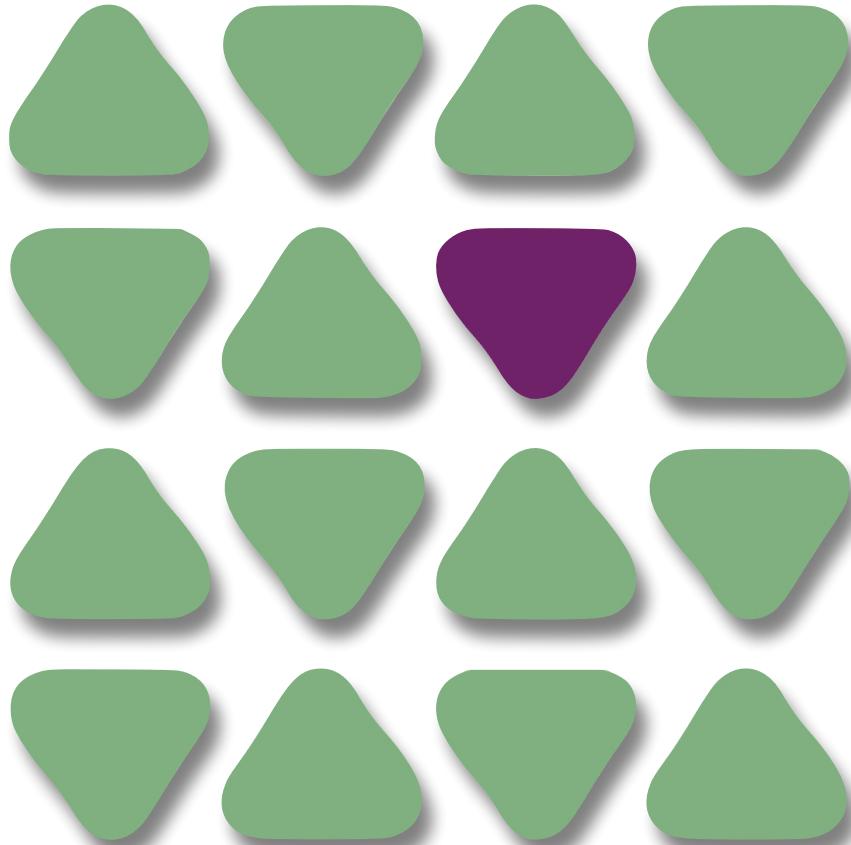


layered
architecture

domain partitioning

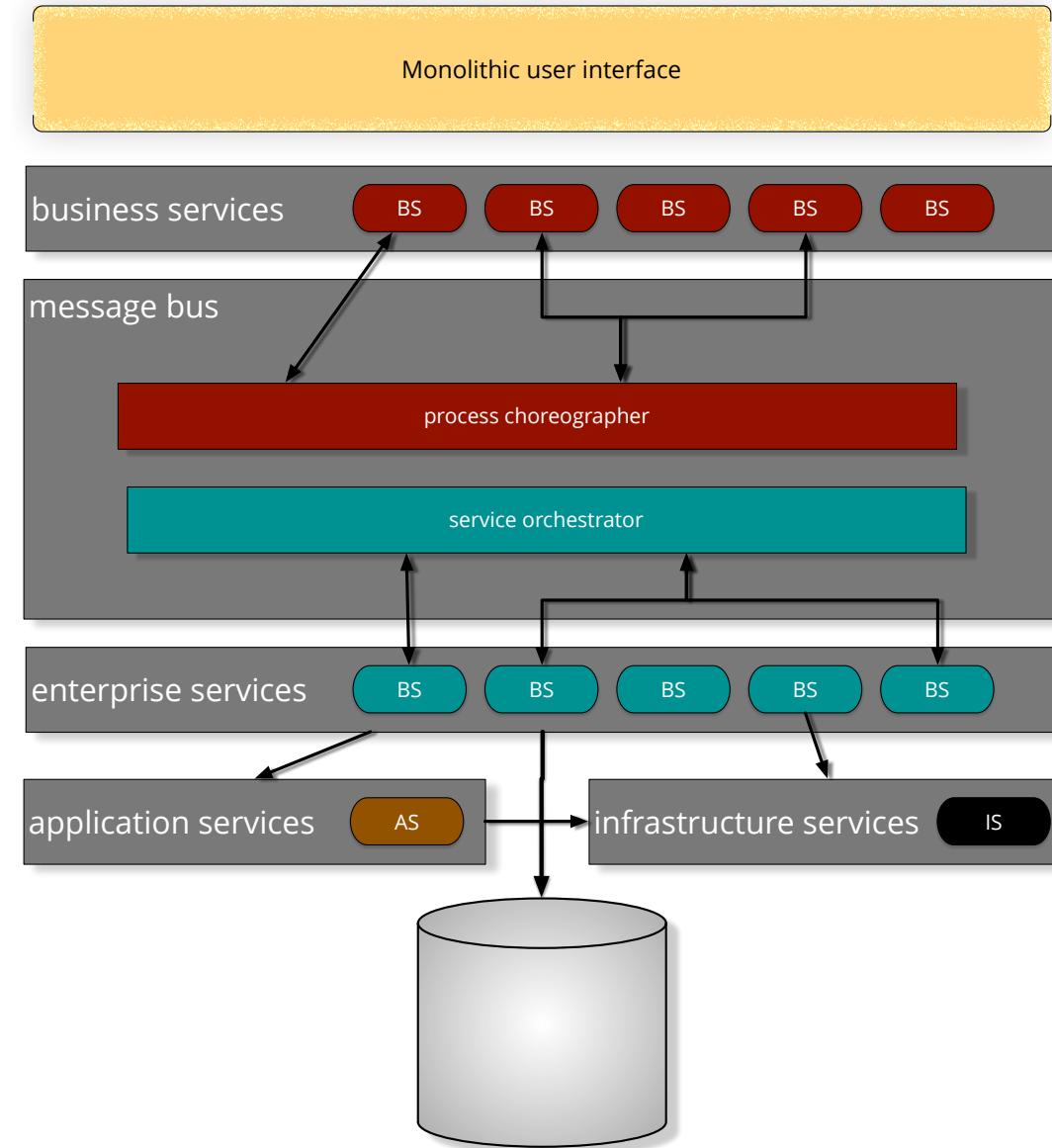


modular monolith

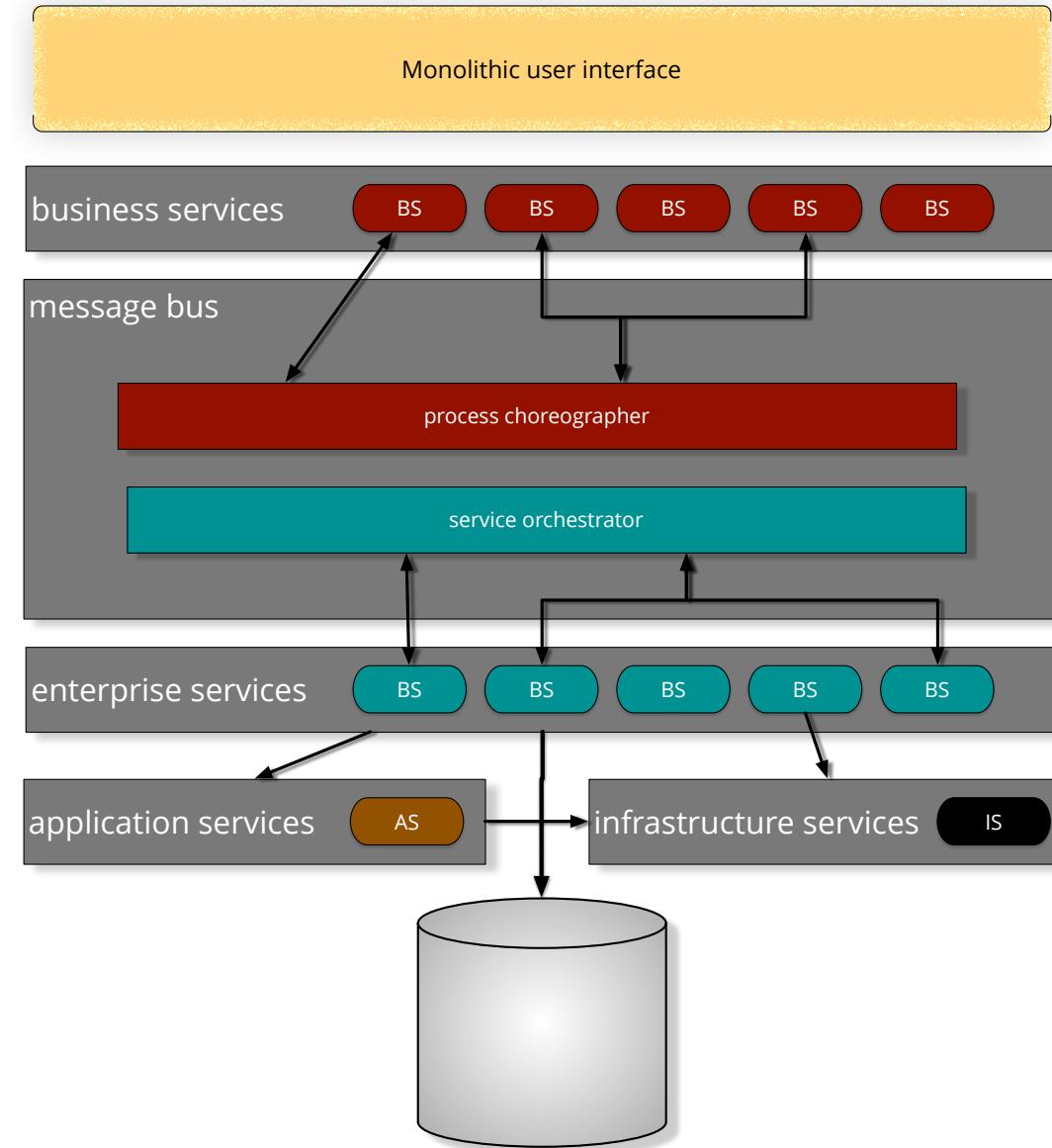


distributed systems

ESB-driven SOA

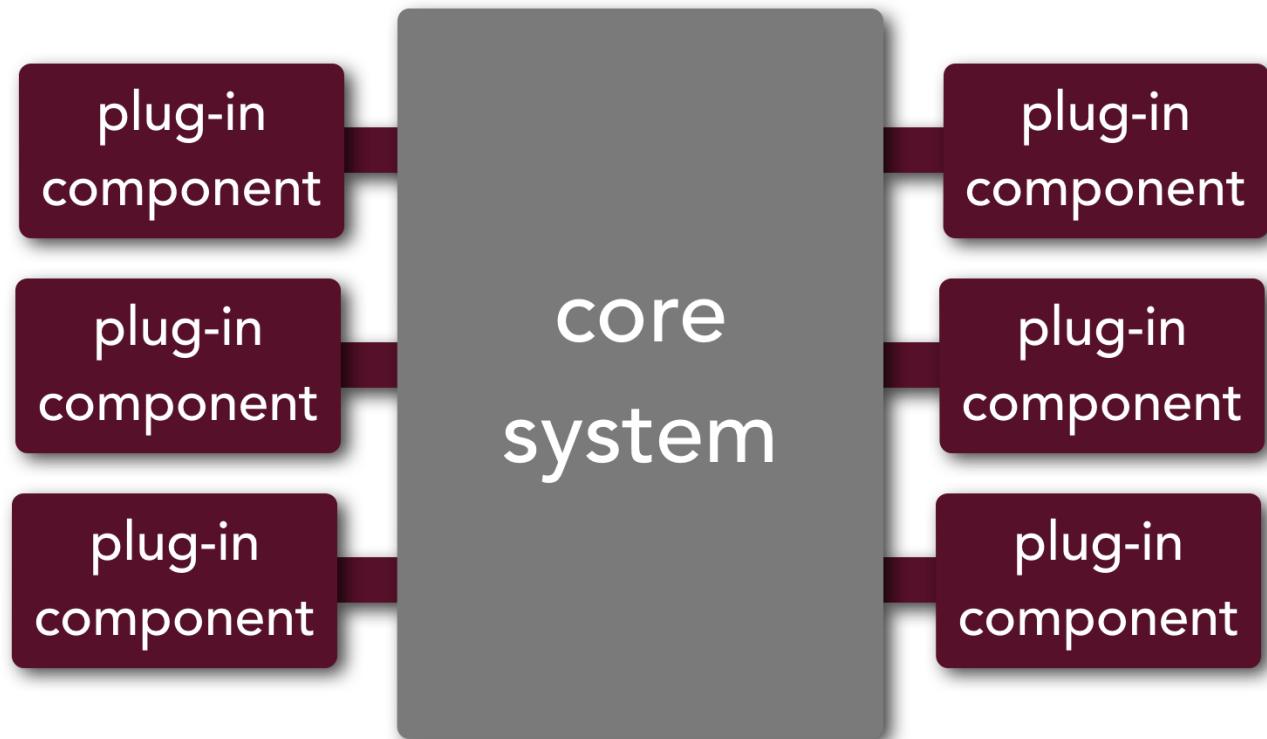


ESB-driven SOA



technical partitioning

Microkernel

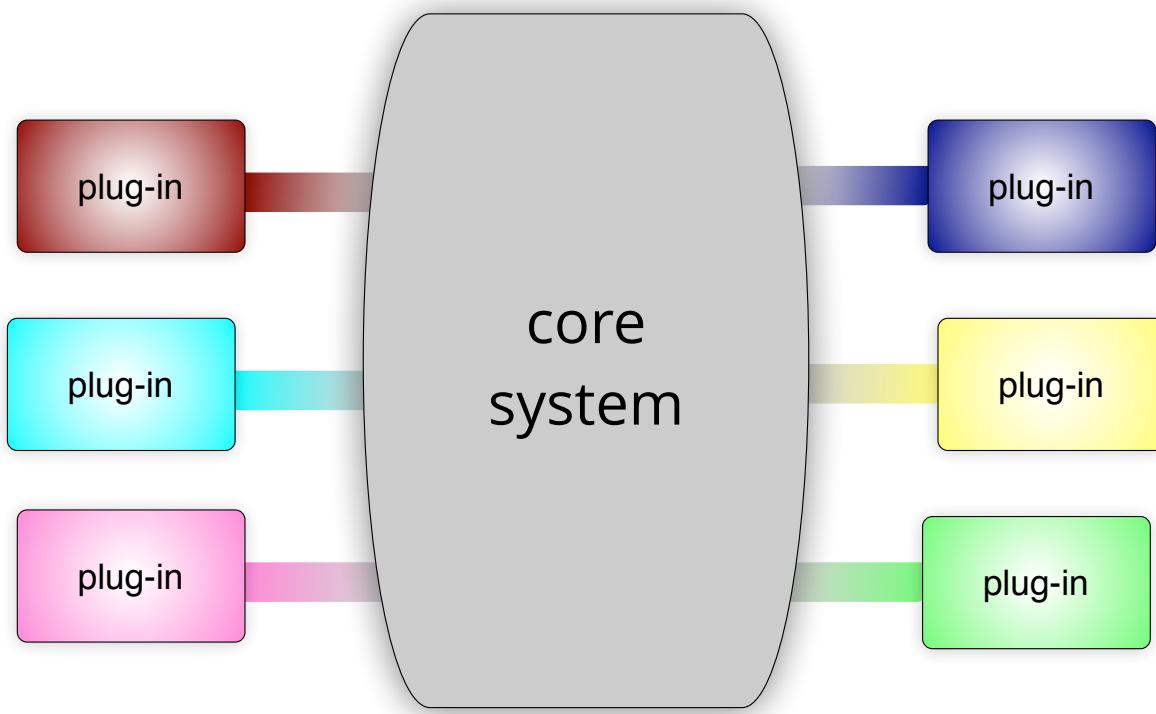


Microkernel

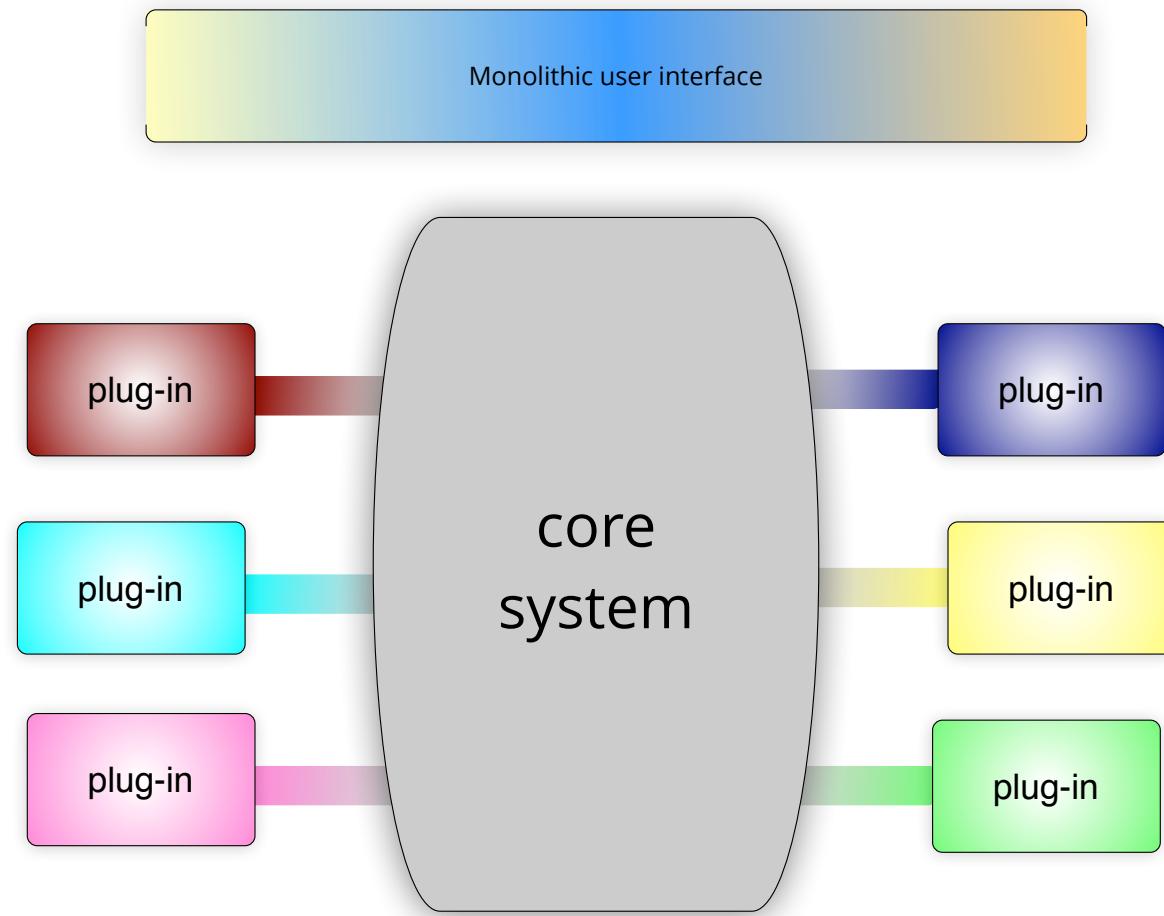
claims processing



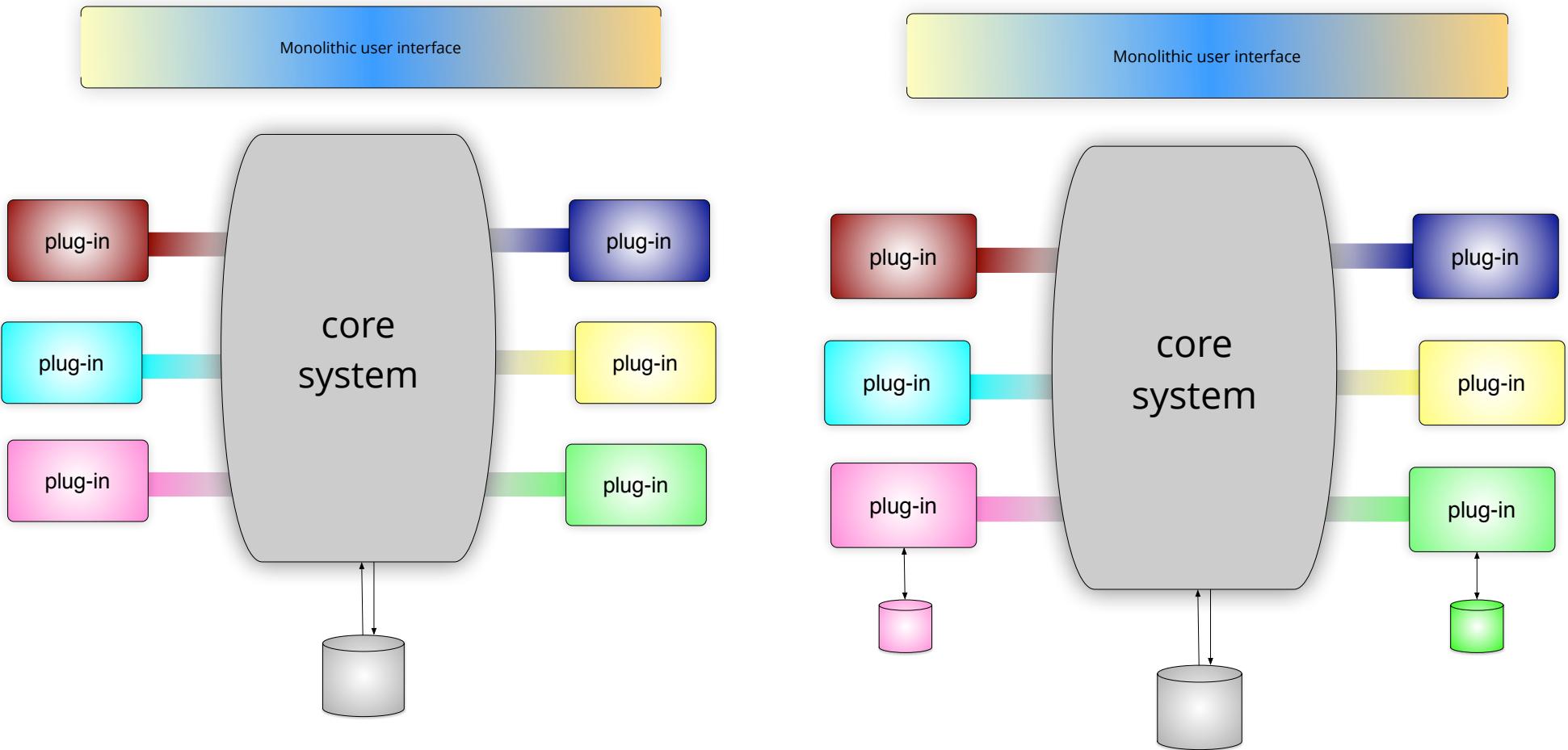
Microkernel Quanta



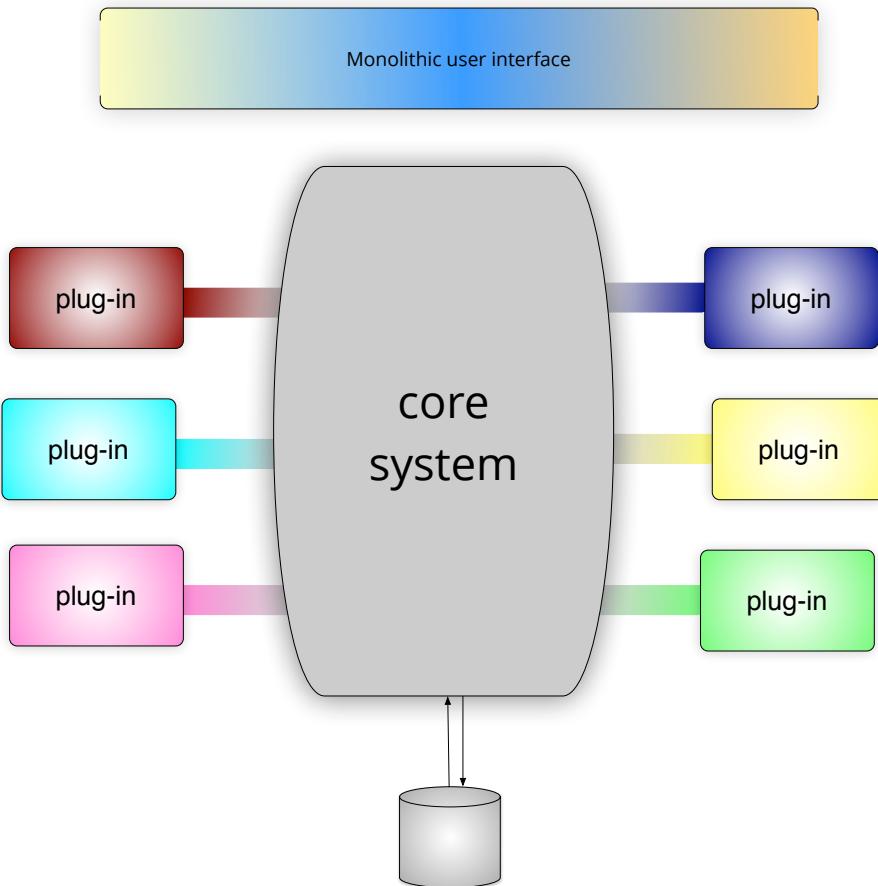
Microkernel: UI



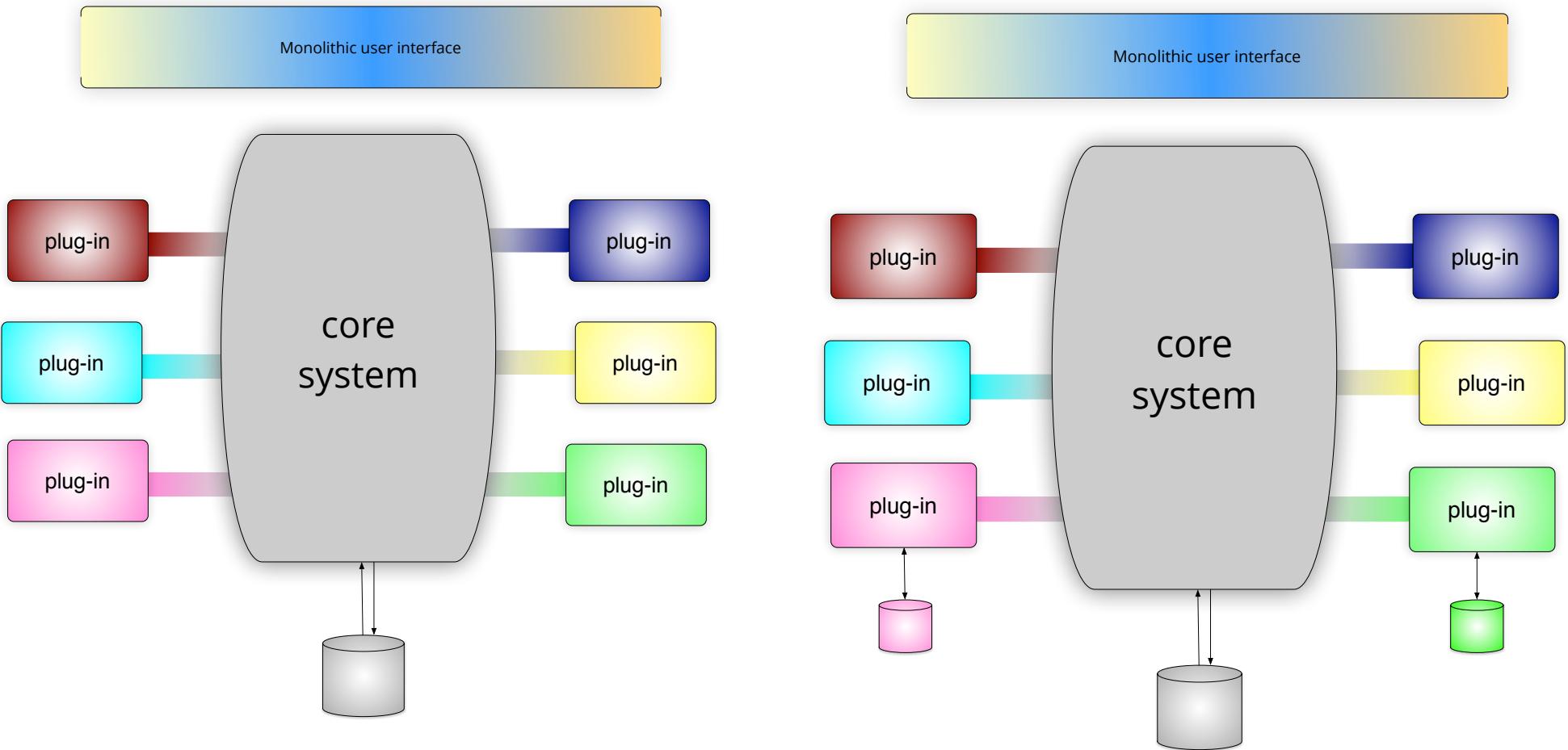
Microkernel: Data



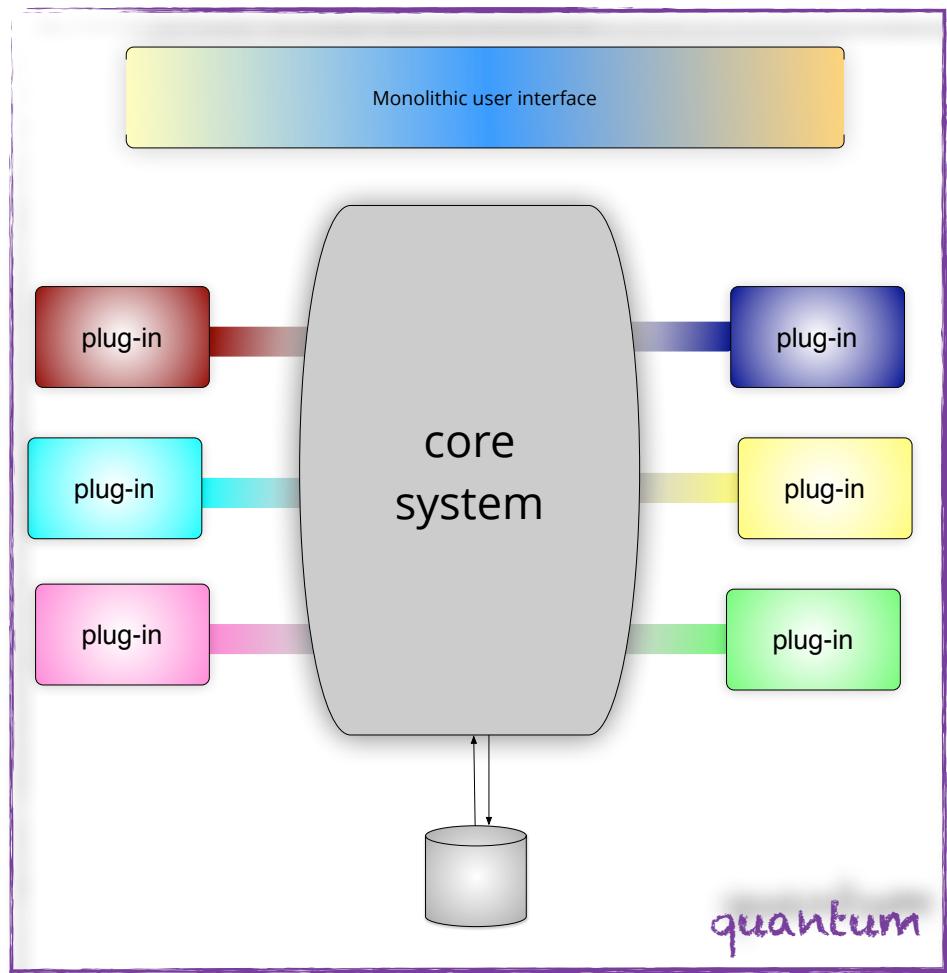
Microkernel: Data



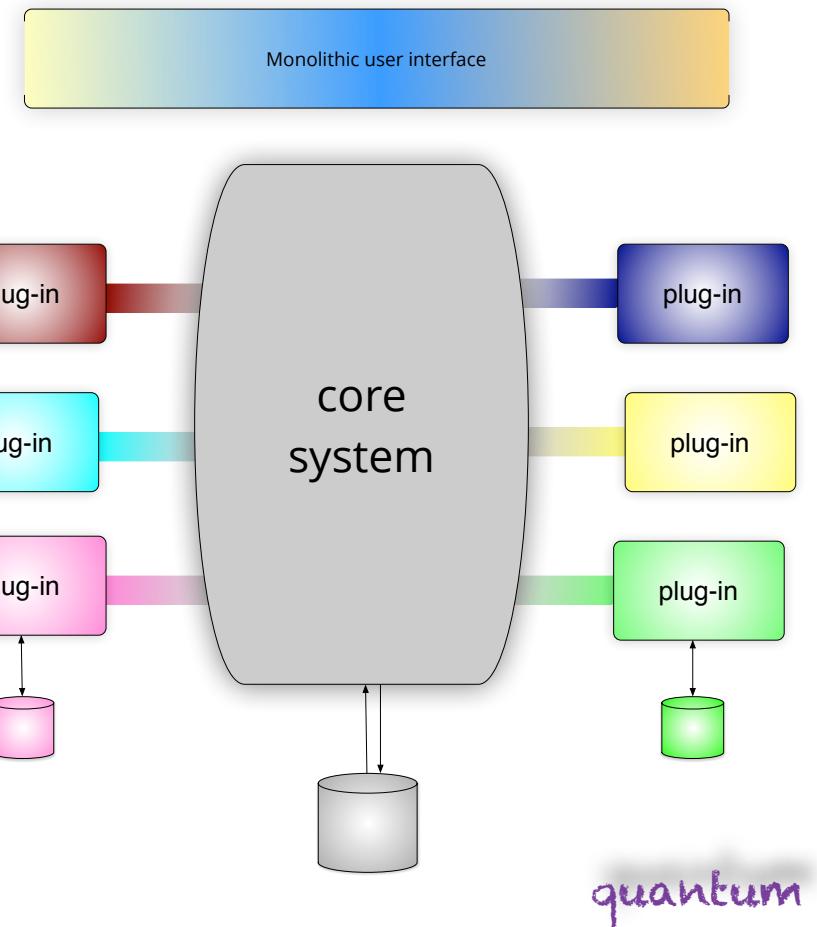
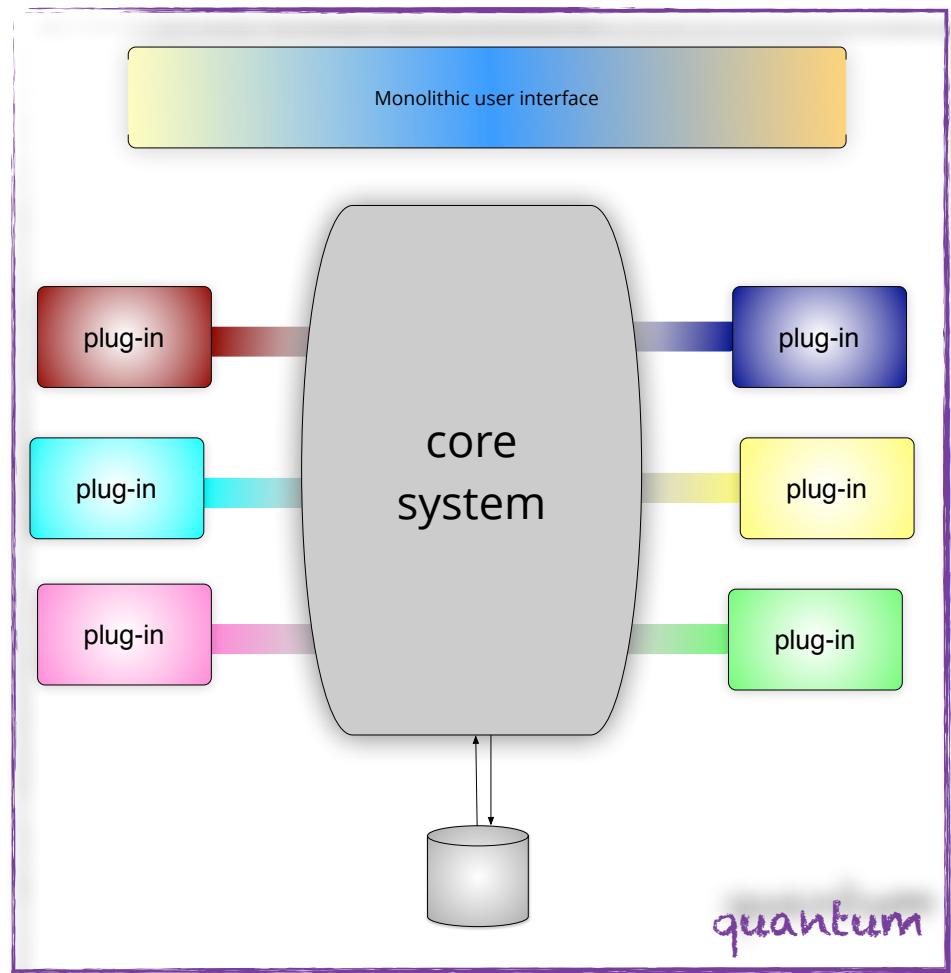
Microkernel: Data



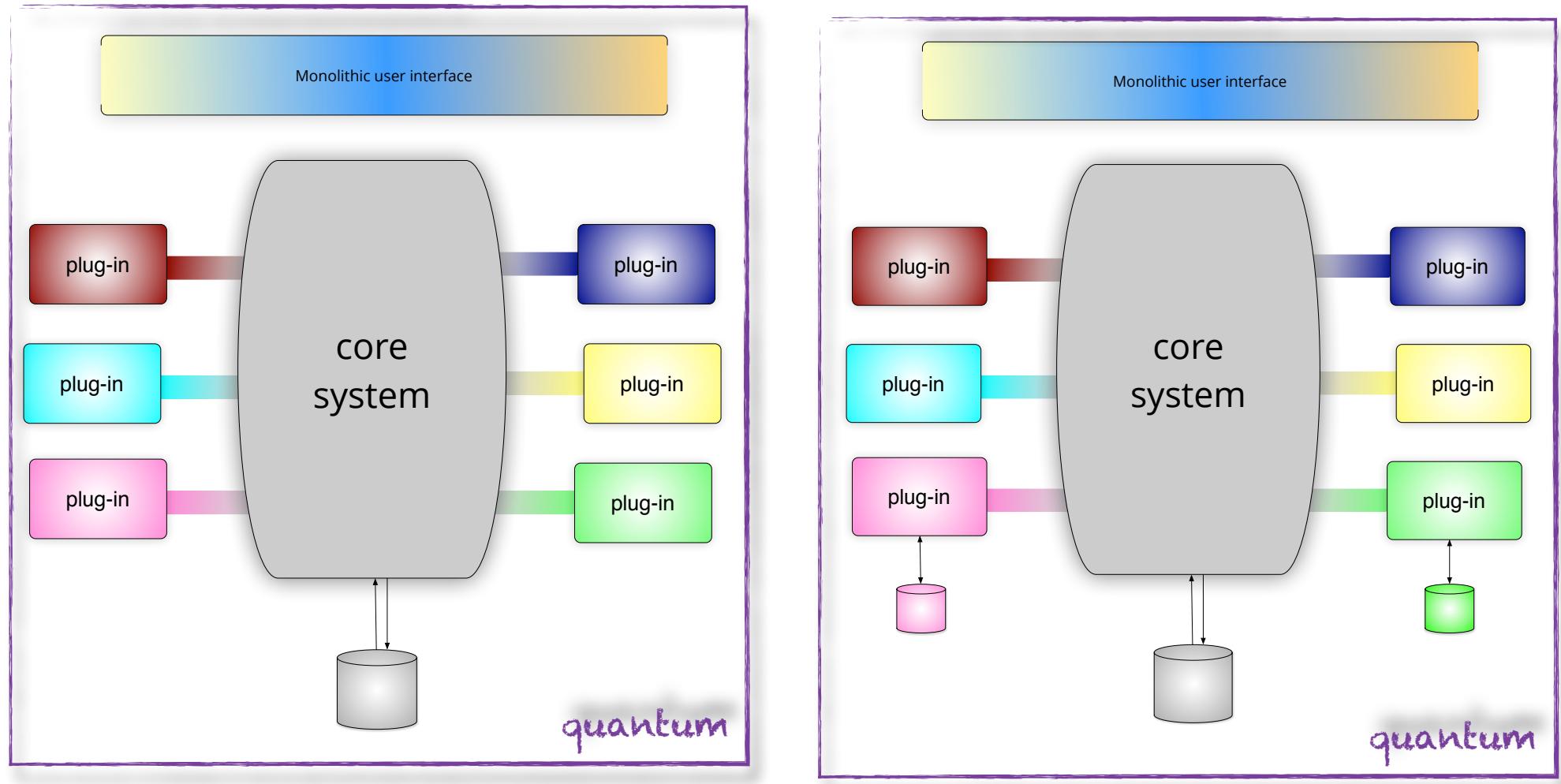
Microkernel: Quanta



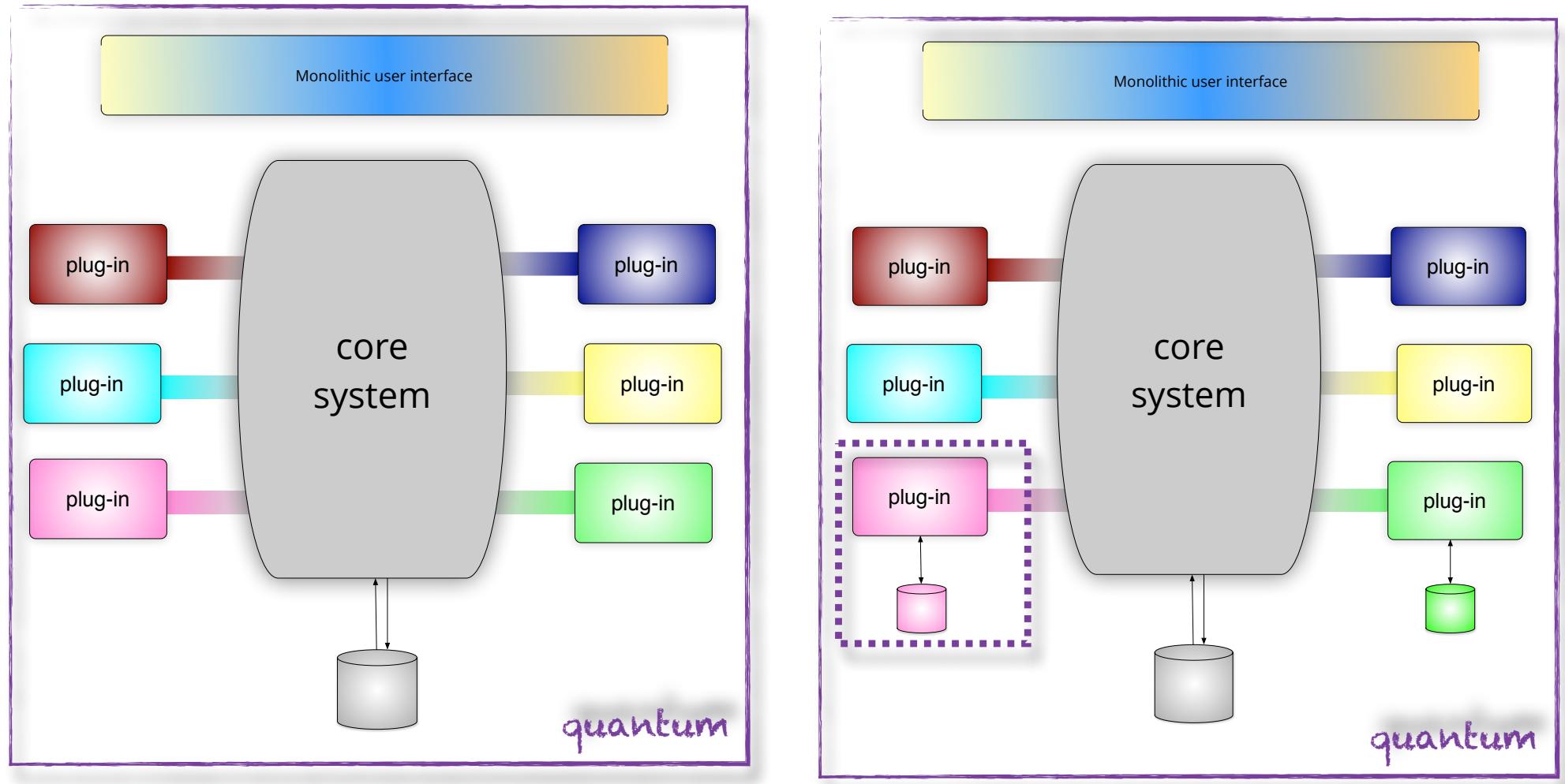
Microkernel: Quanta



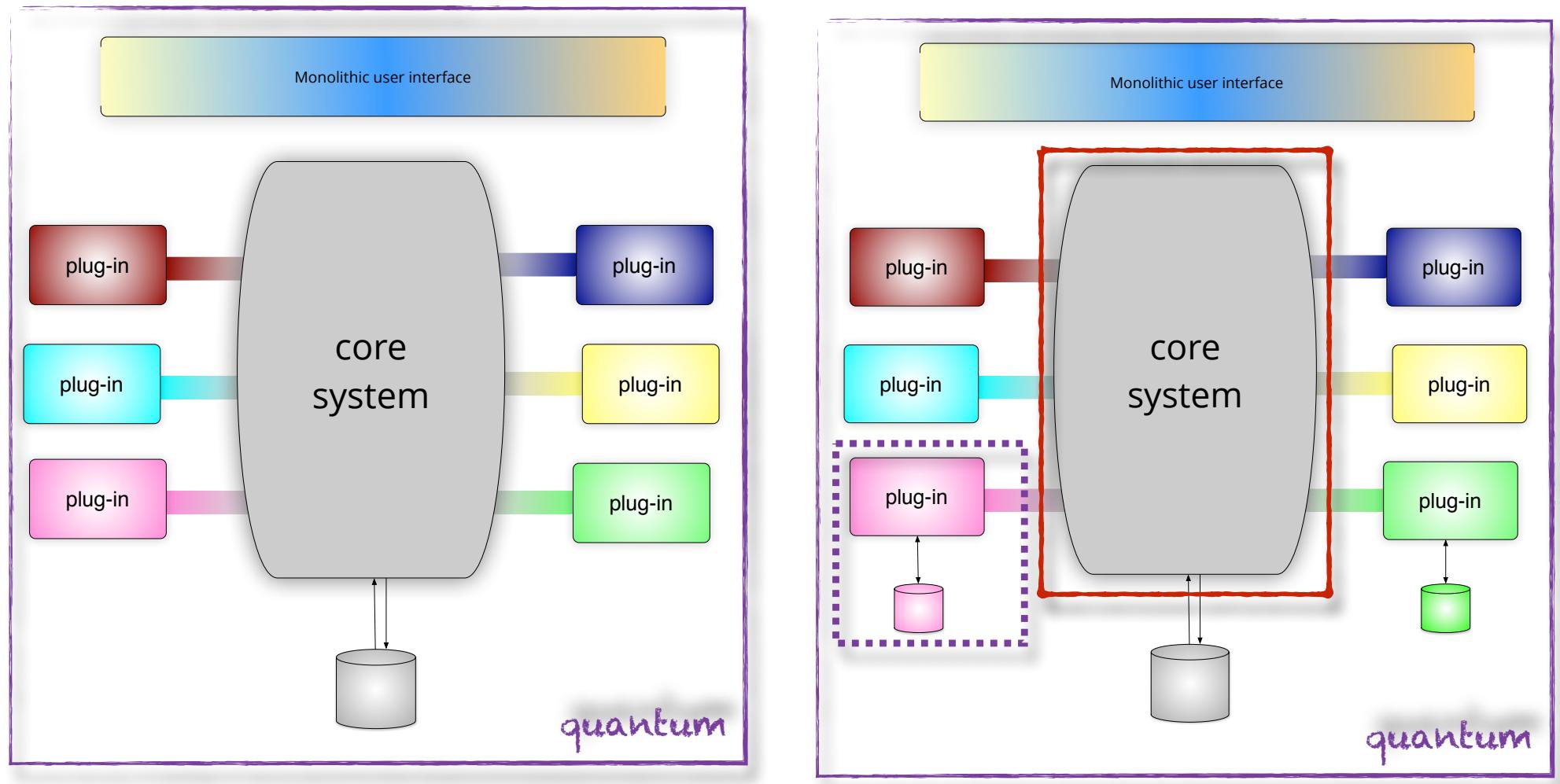
Microkernel: Quanta



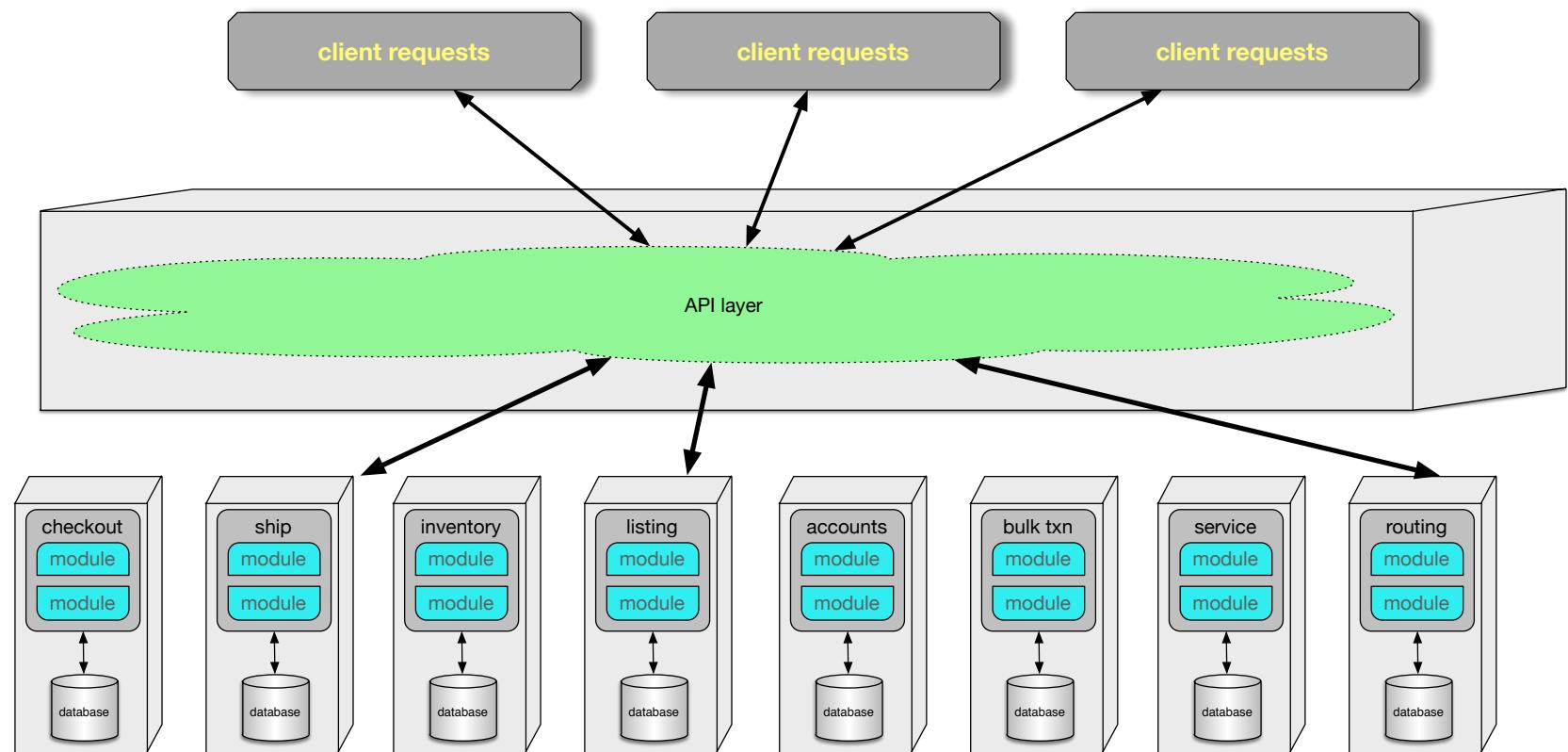
Microkernel: Quanta



Microkernel: Quanta



Microservices



What makes microservices so evolvable?

extremely loose coupling

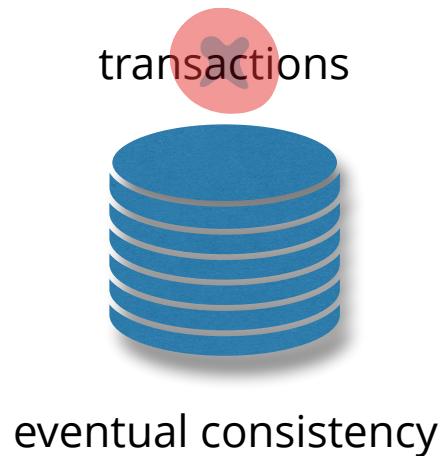
What makes microservices so evolvable?

extremely loose coupling



What makes microservices so evolvable?

extremely loose coupling



What makes microservices so evolvable?

extremely loose coupling

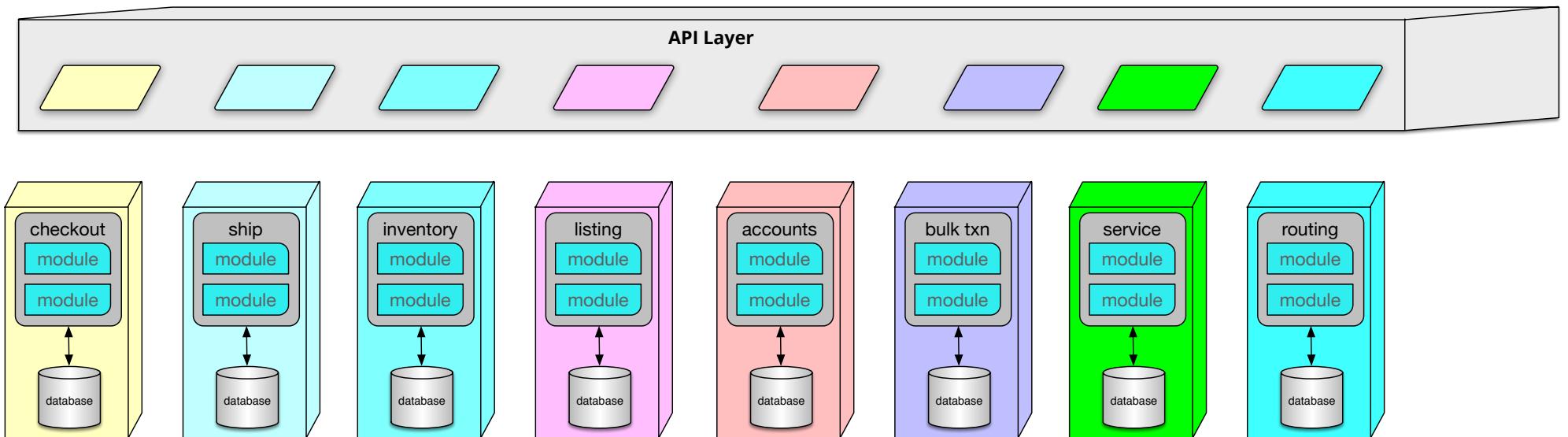


What makes microservices so evolvable?

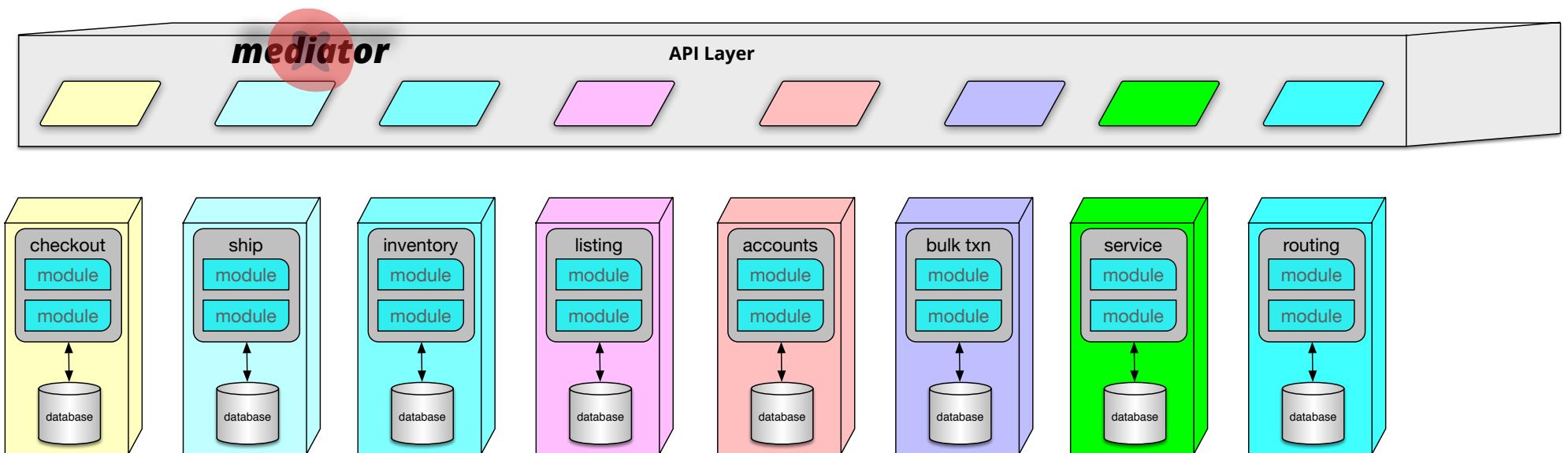
extremely loose coupling



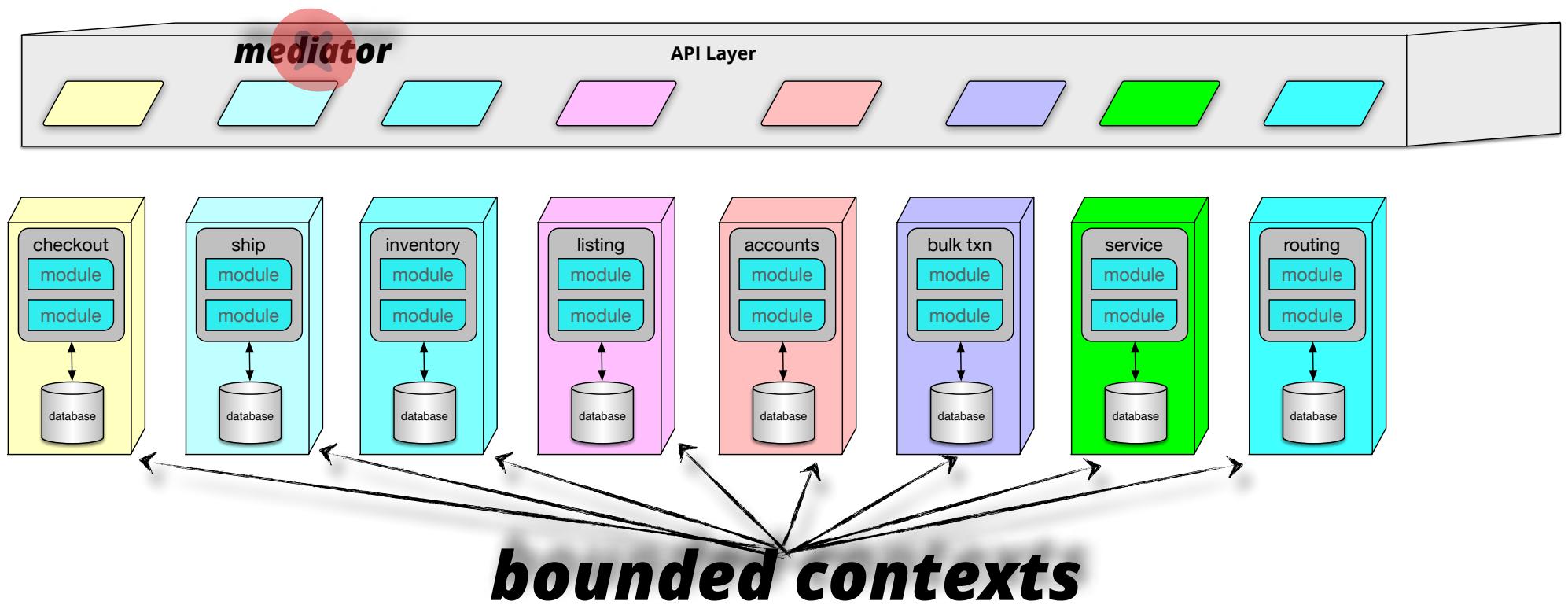
Microservices



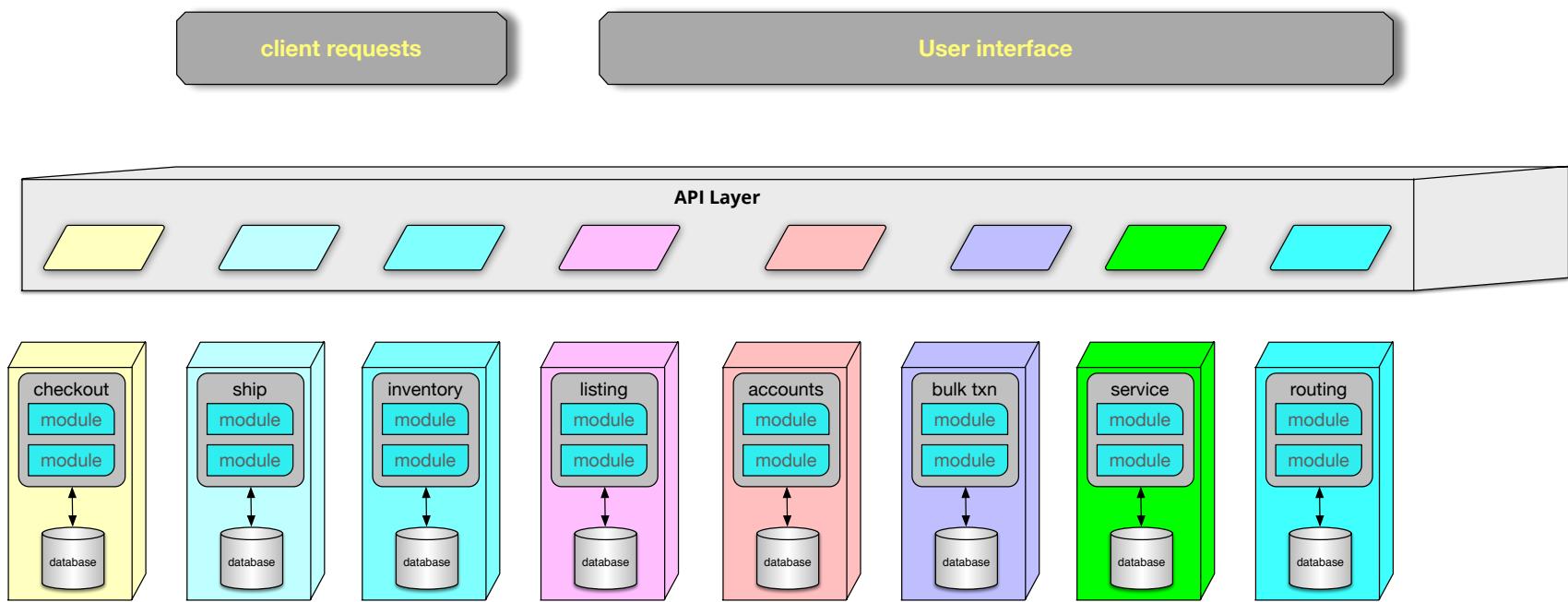
Microservices



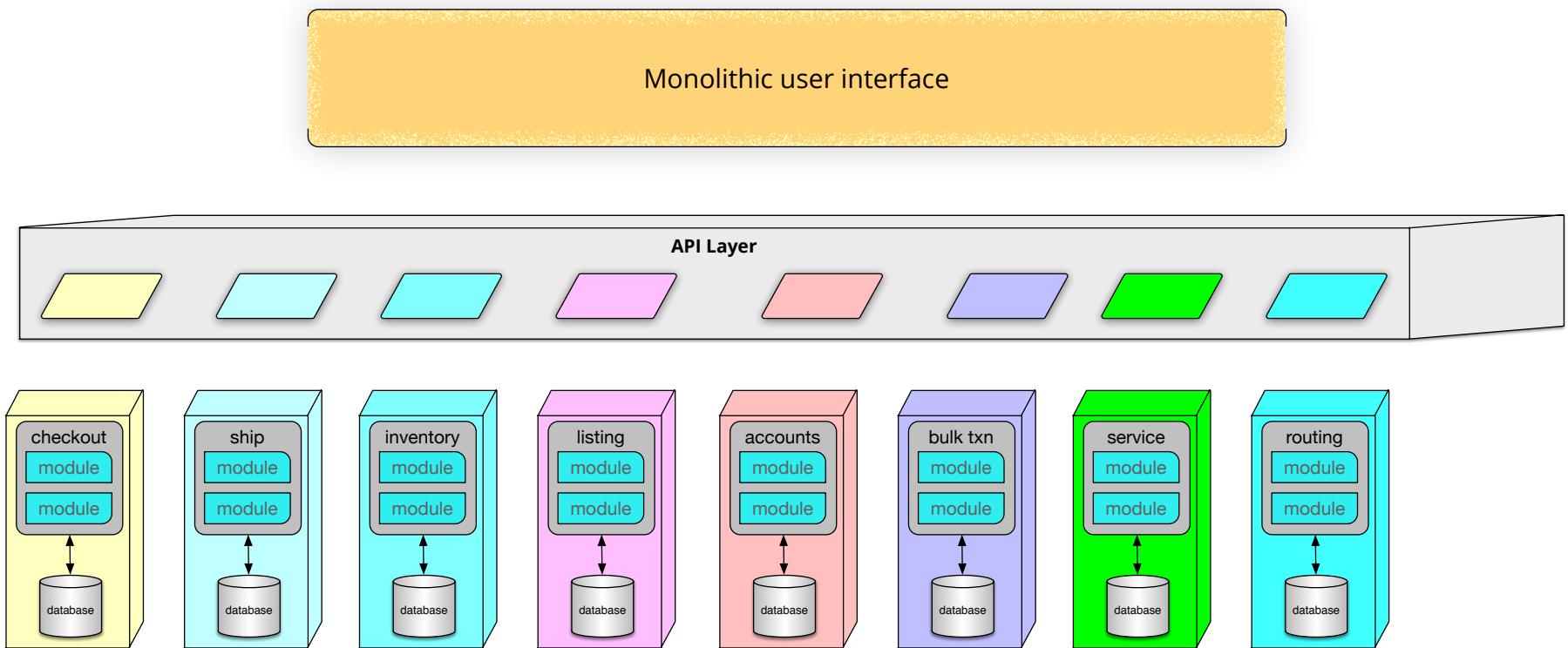
Microservices



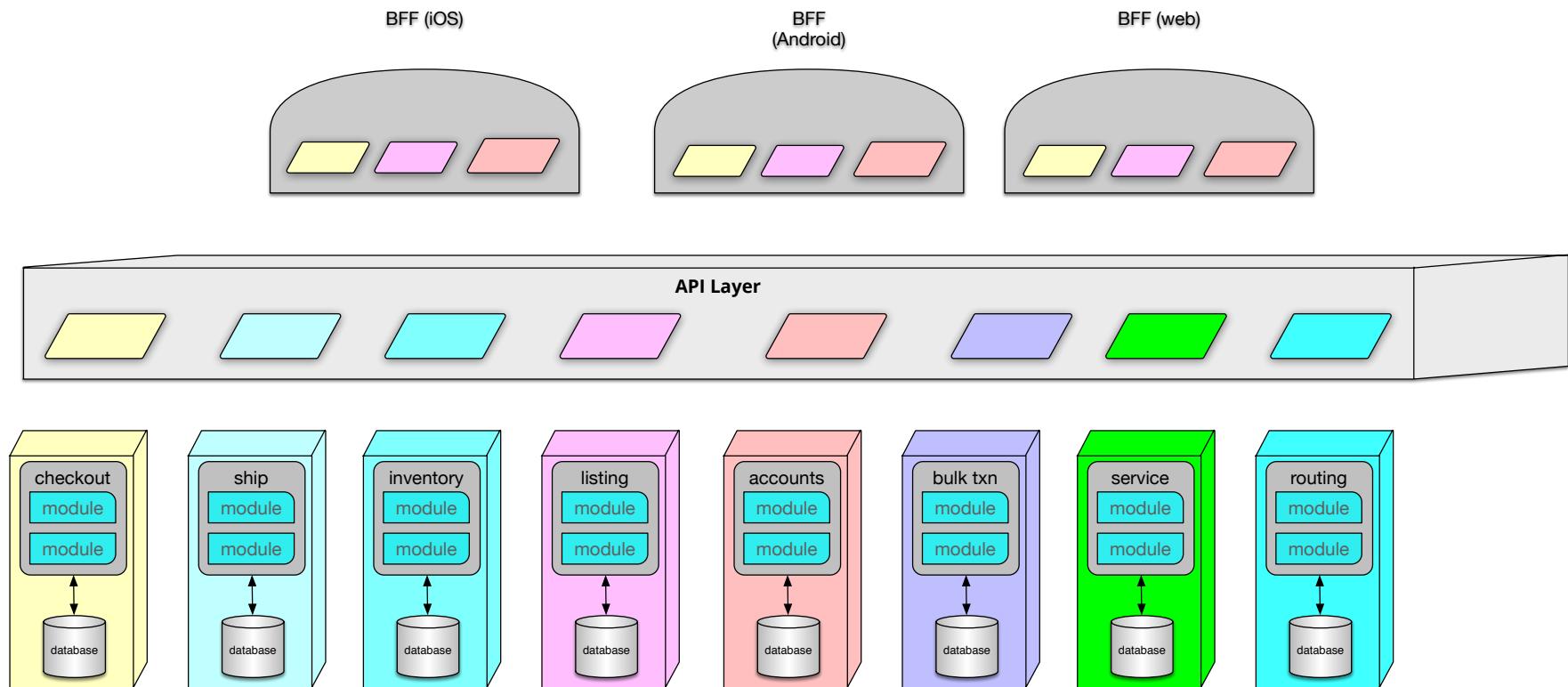
Microservices



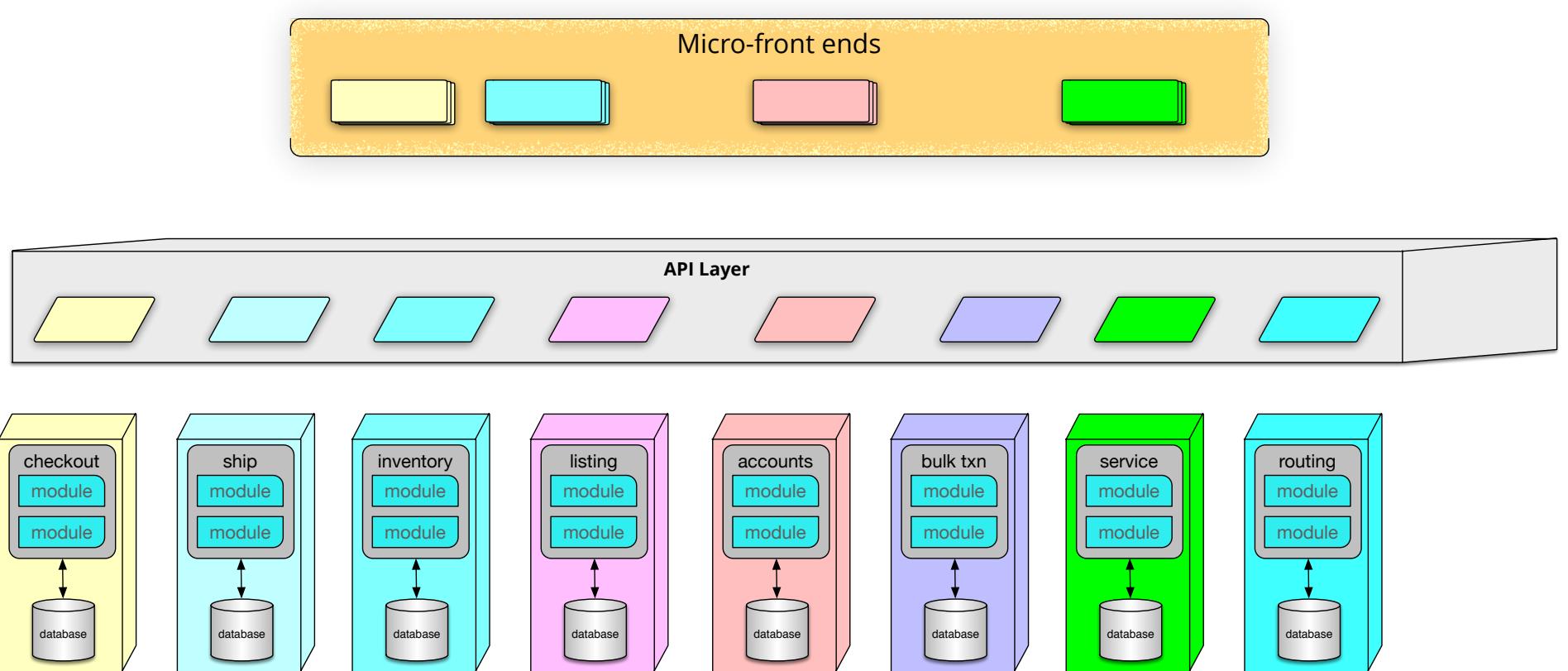
Microservices: Monolithic UI



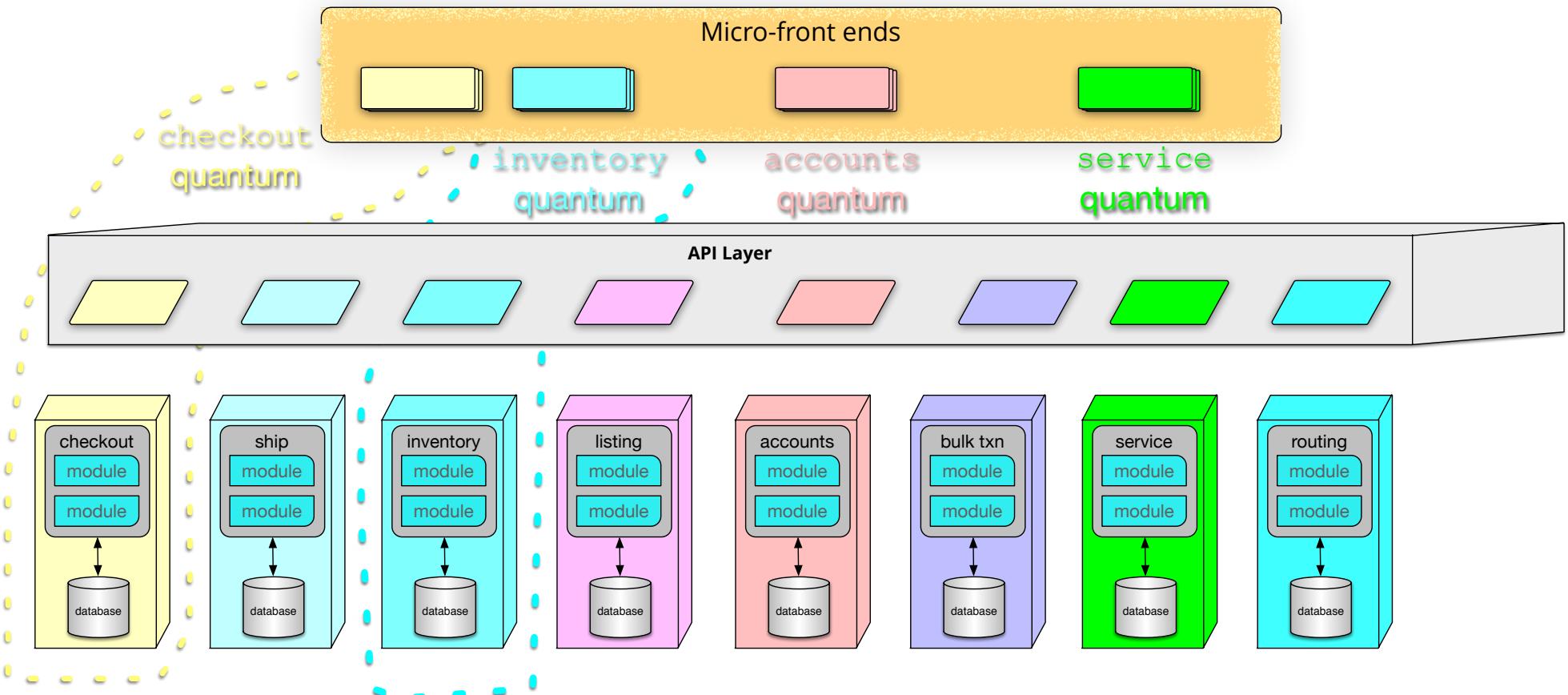
Microservices :BFF



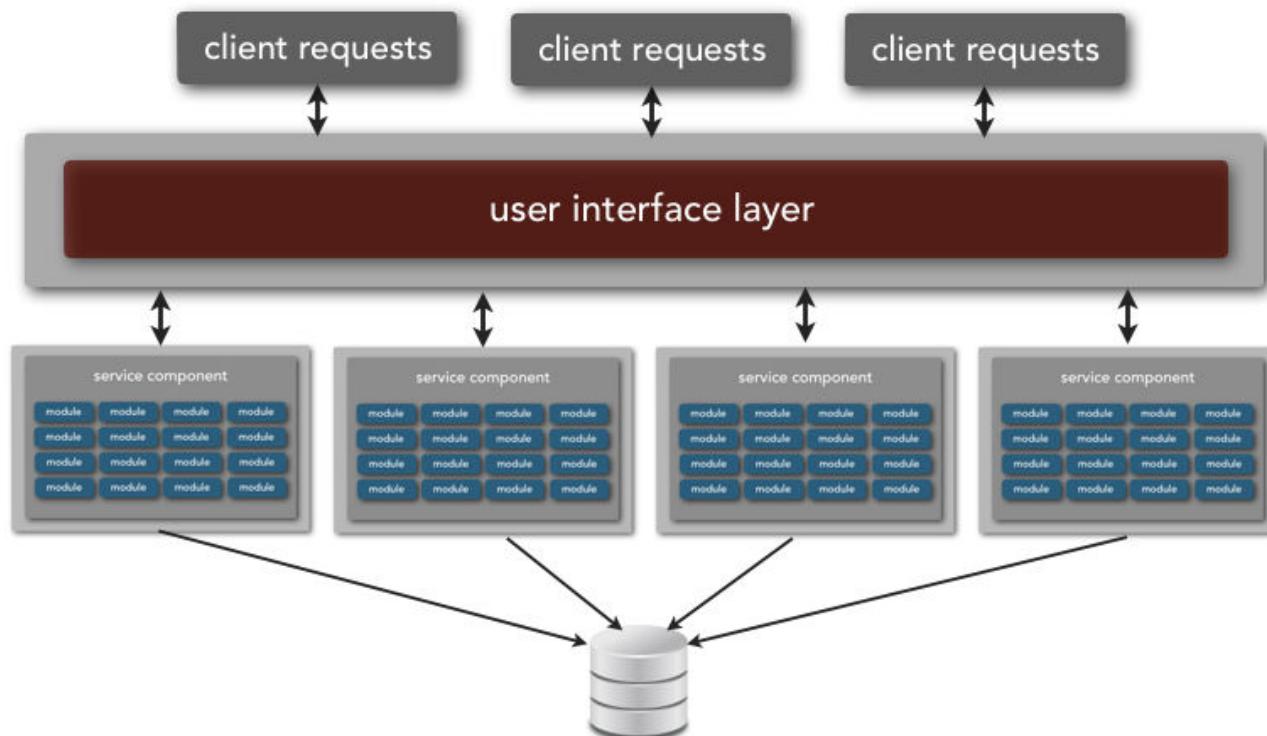
Microservices



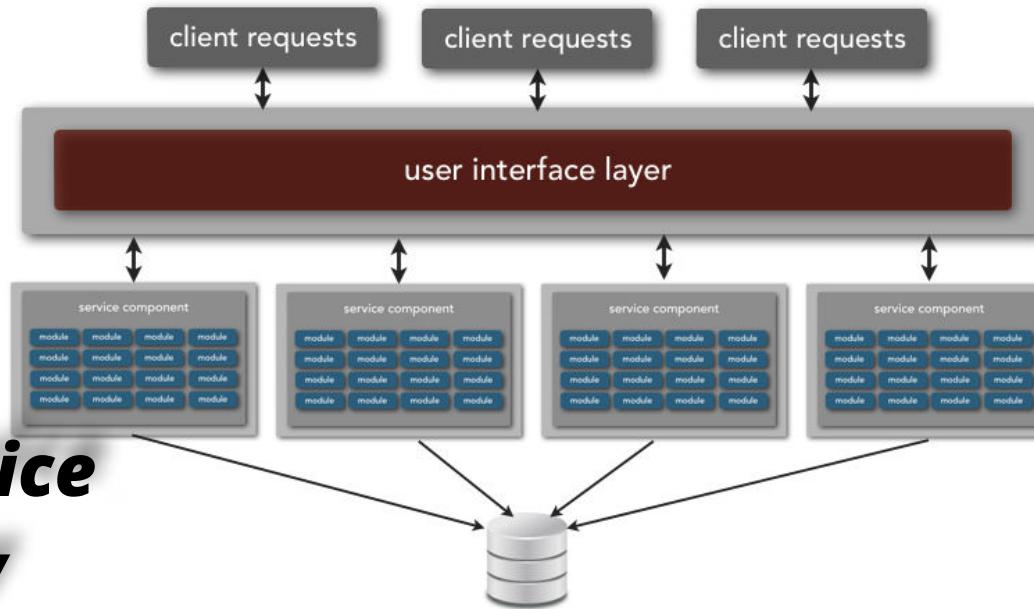
Microservices



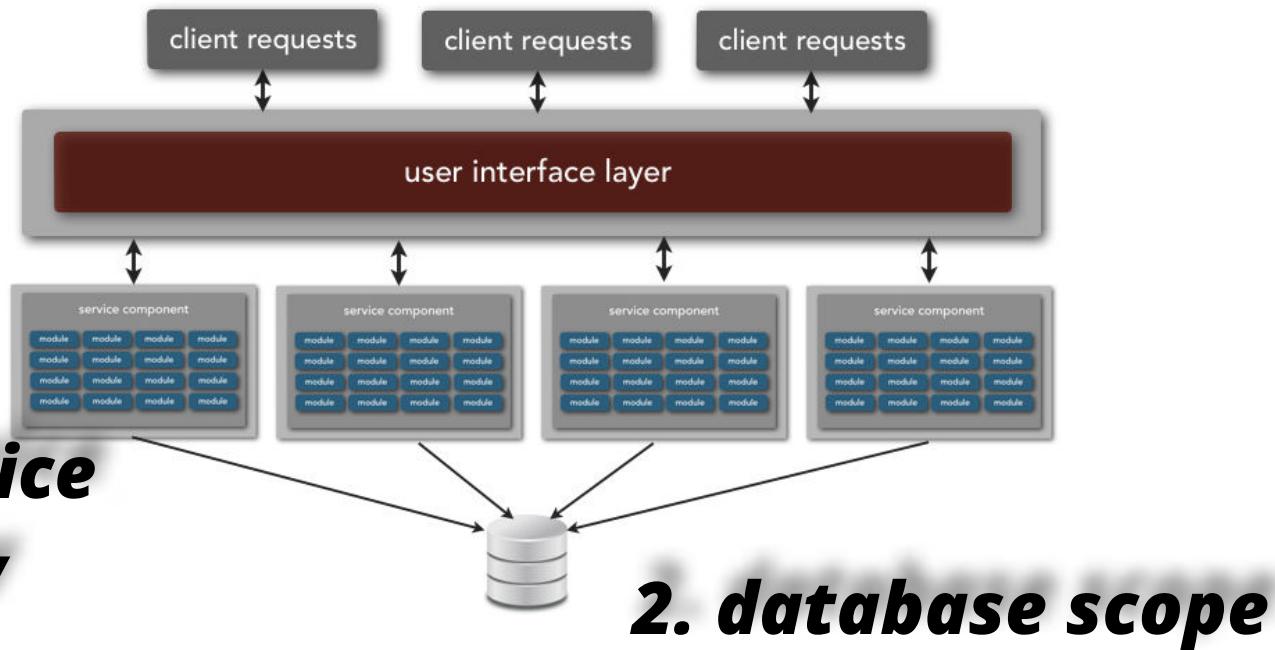
Service-based Architectures



Service-based Architectures

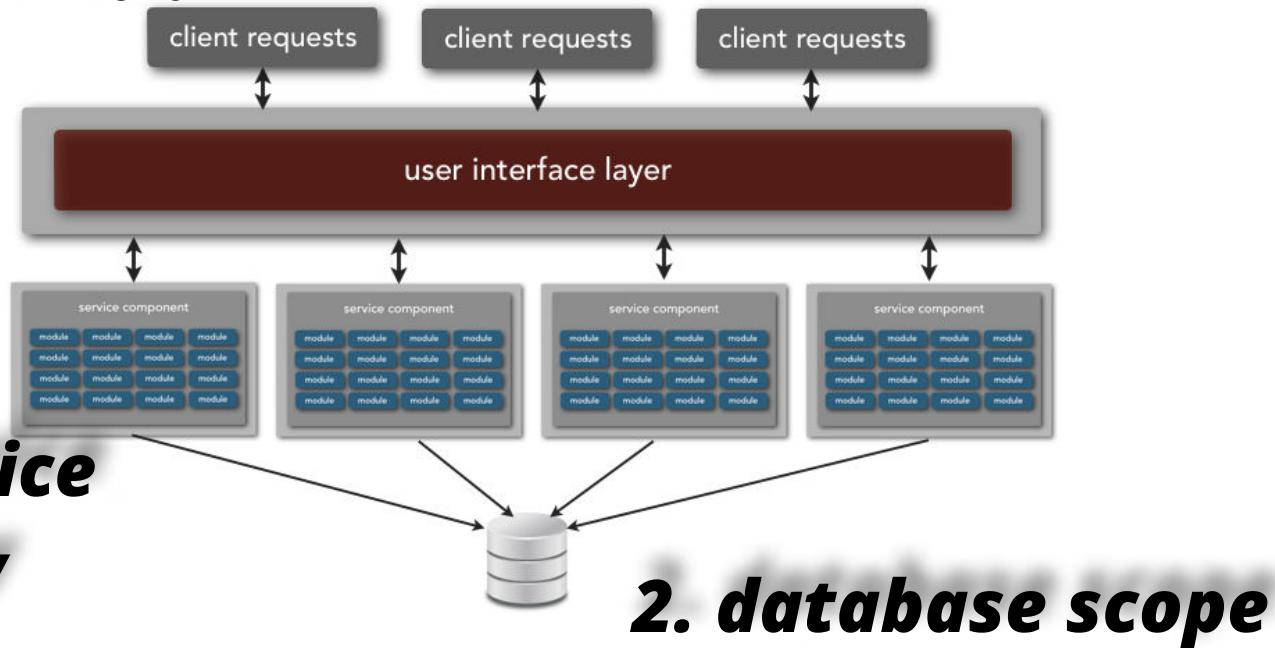


Service-based Architectures

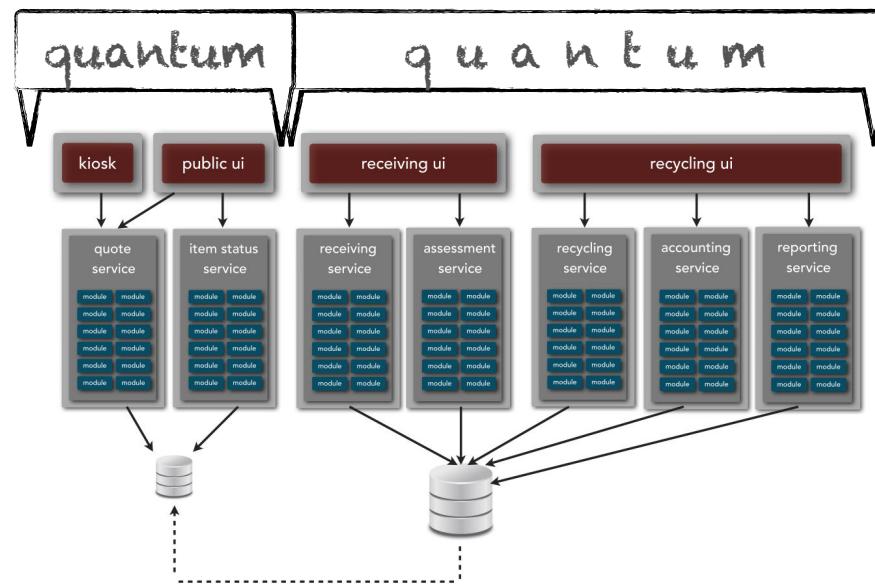


Service-based Architectures

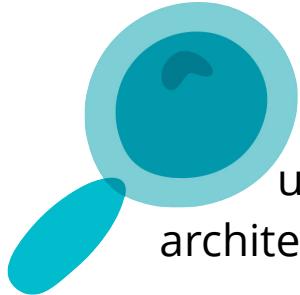
3. use of service bus as integration hub



Reducing Quanta Size



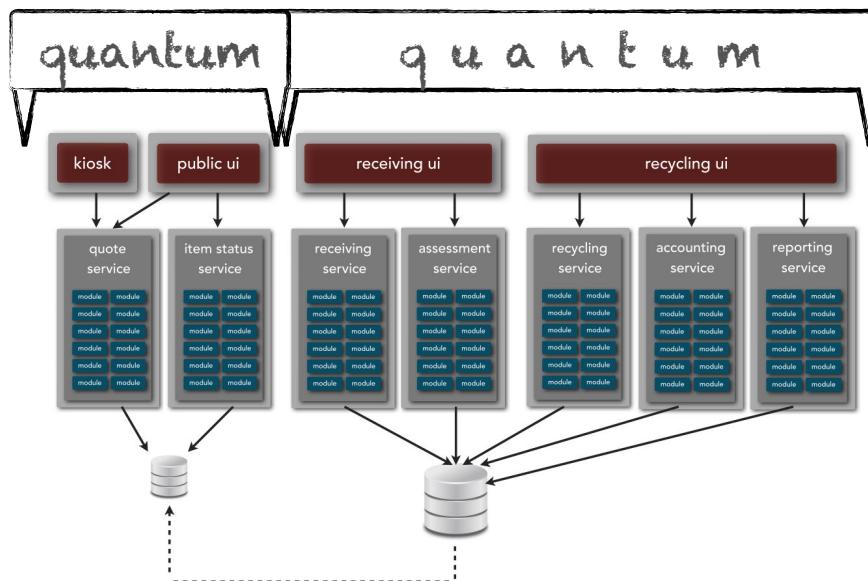
Reducing Quanta Size



useful for
architectural analysis



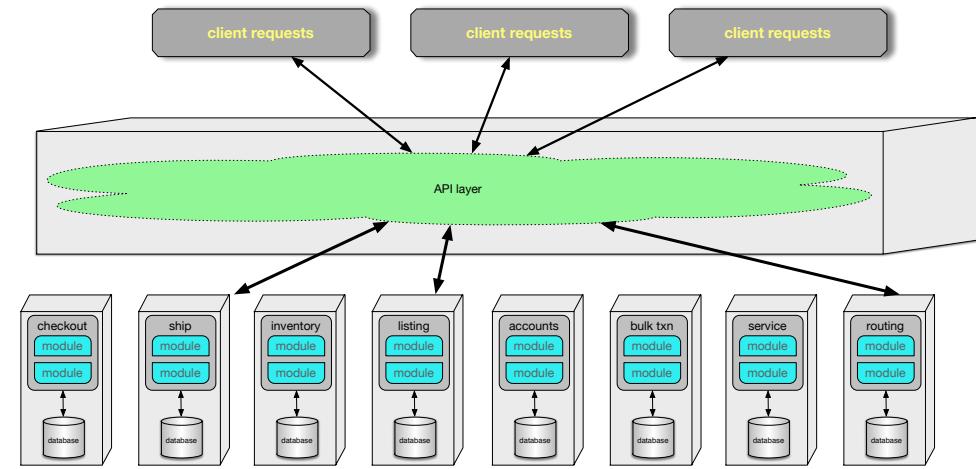
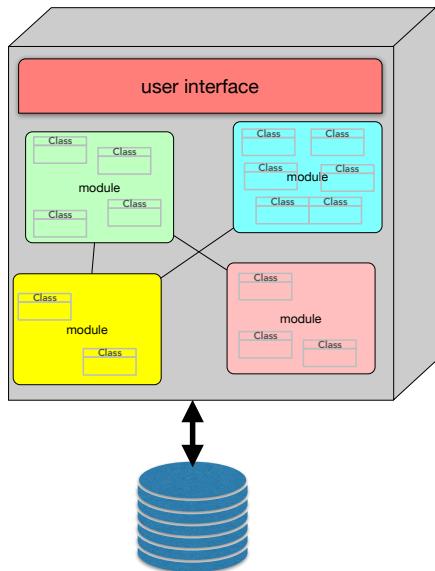
*helps analyze
coupling*



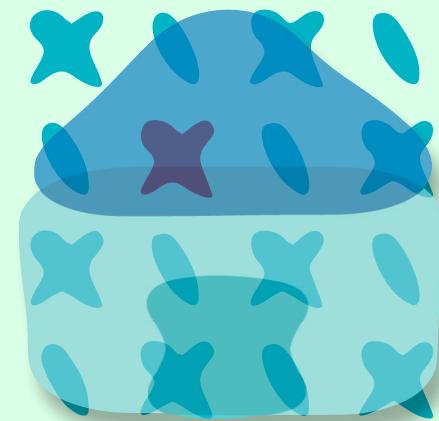
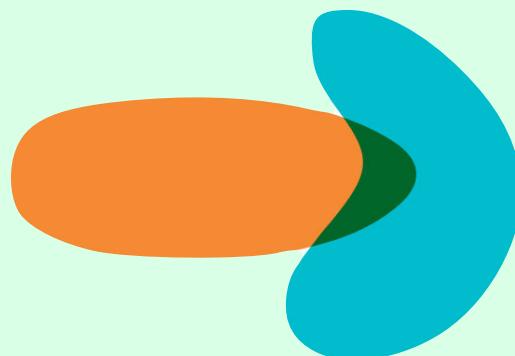
Smaller Quanta Size

△
≡

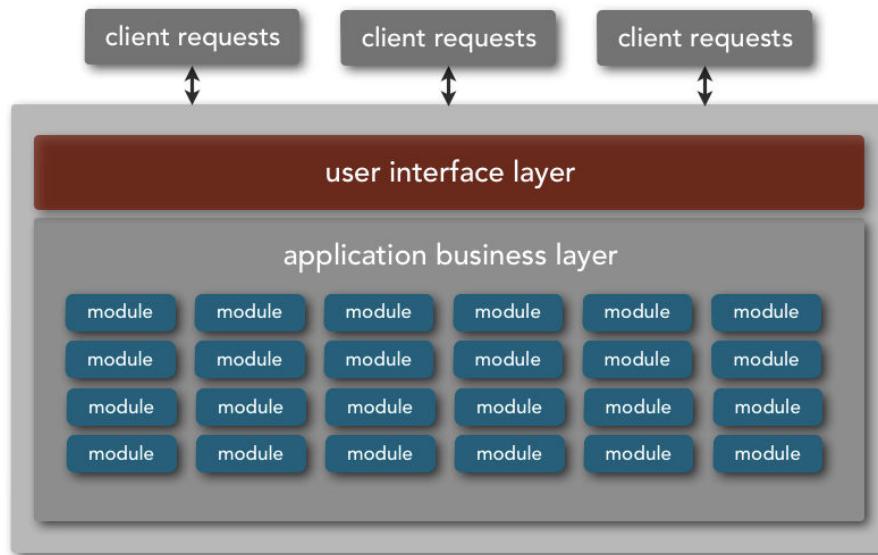
Evolutionary



Migrating Architectures

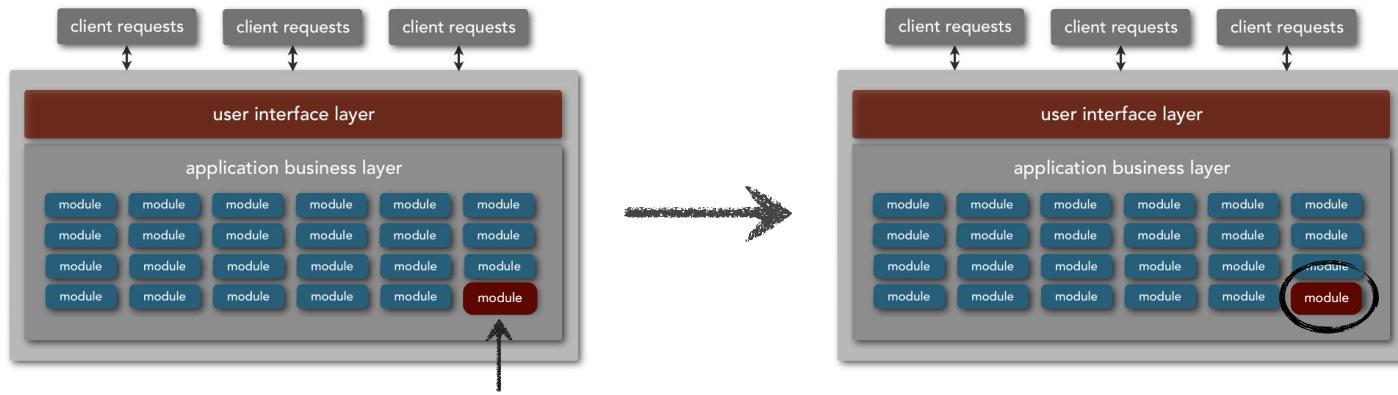


monolithic application issues



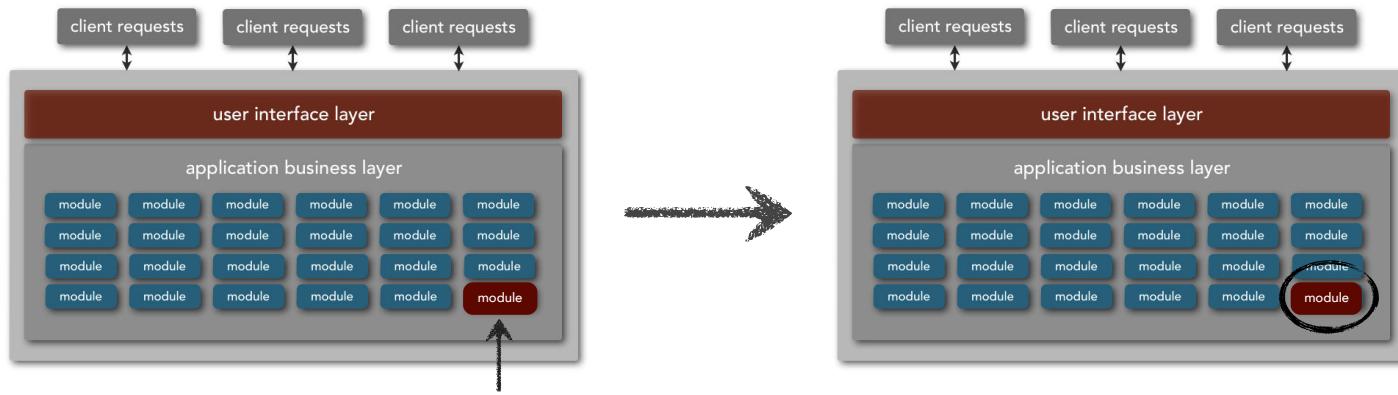
monolithic application issues

deployment



monolithic application issues

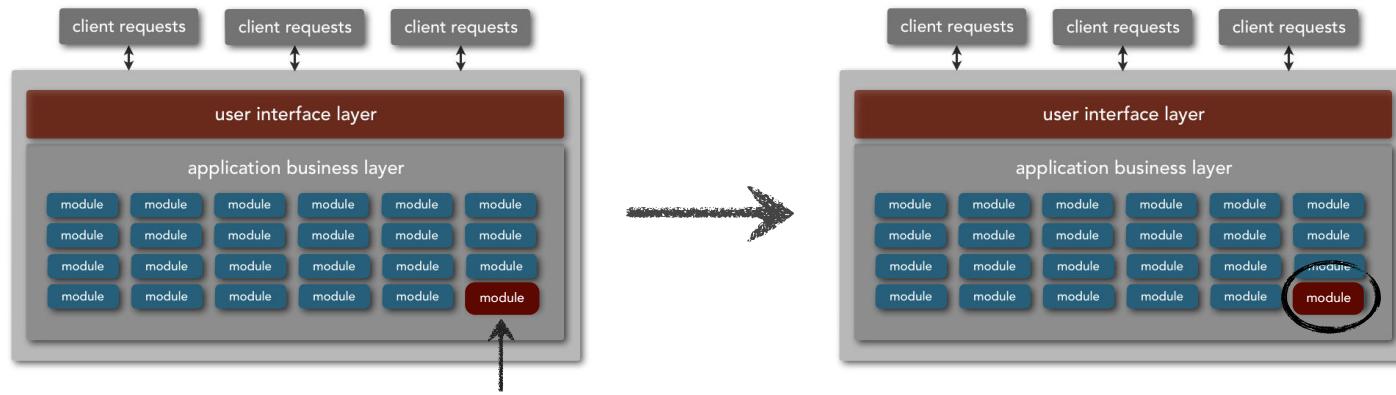
deployment



entire application must be deployed for a small change

monolithic application issues

deployment

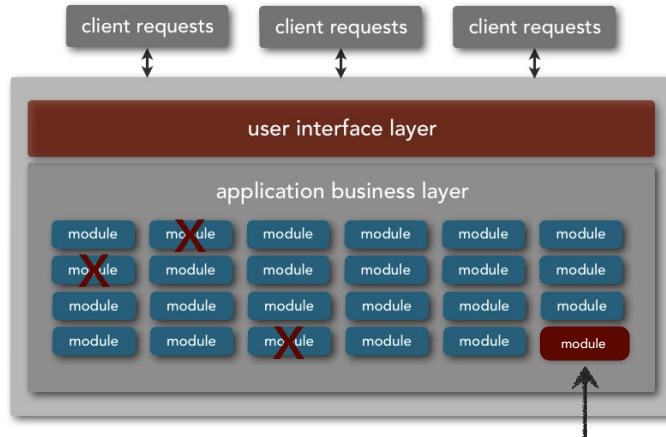


entire application must be deployed for a small change

scheduling and coordination challenges can impact deployment

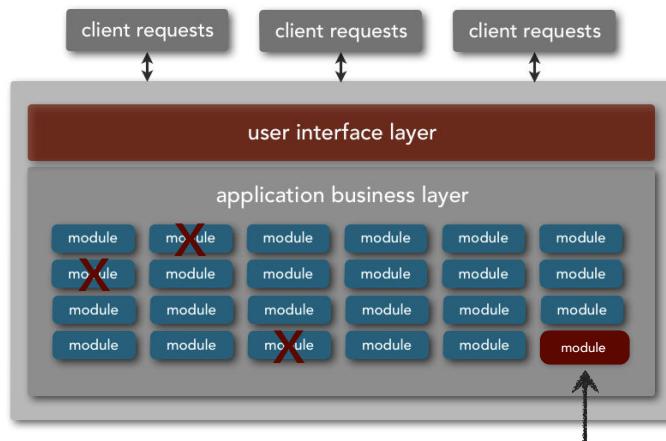
monolithic application issues

reliability and robustness



monolithic application issues

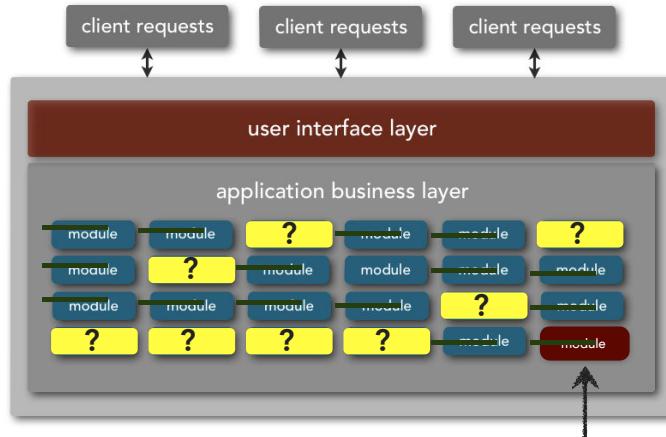
reliability and robustness



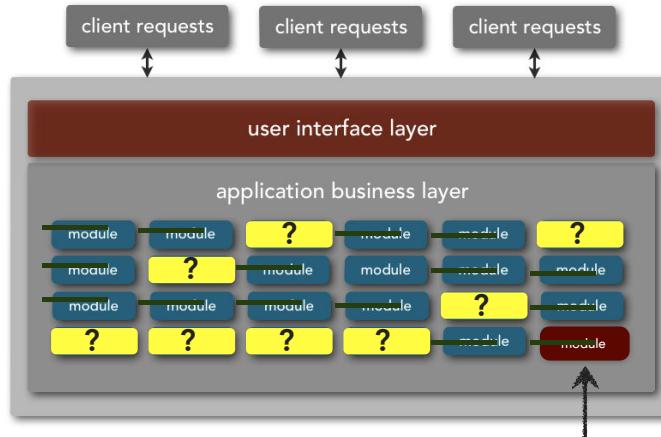
due to tight coupling, changes in one area of the application adversely affects other areas

monolithic application issues

testability



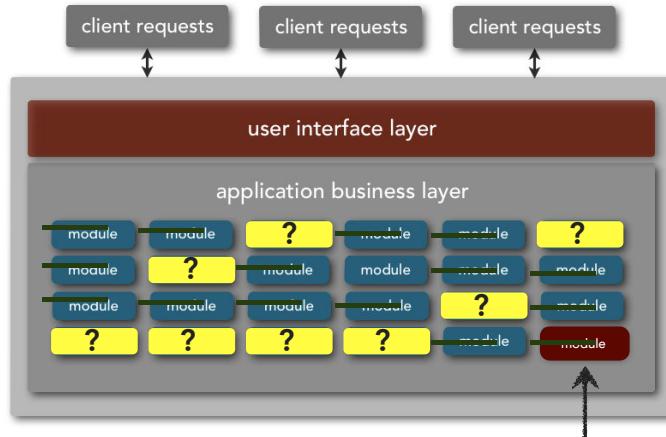
monolithic application issues testability



poor test coverage for entire application

monolithic application issues

testability

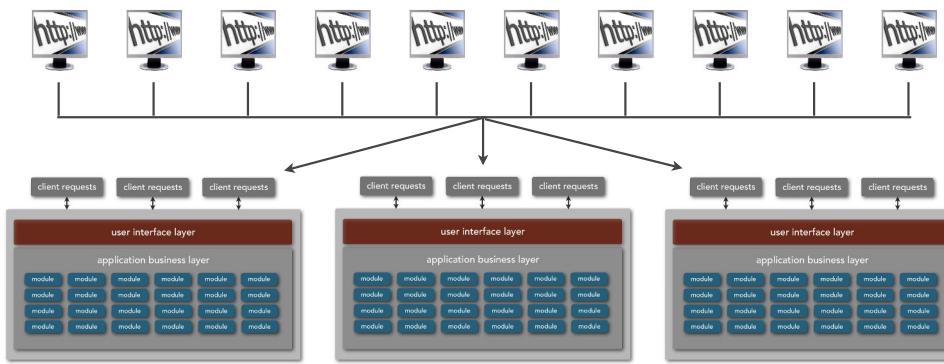


poor test coverage for entire application

low confidence level that nothing else is impacted by the change

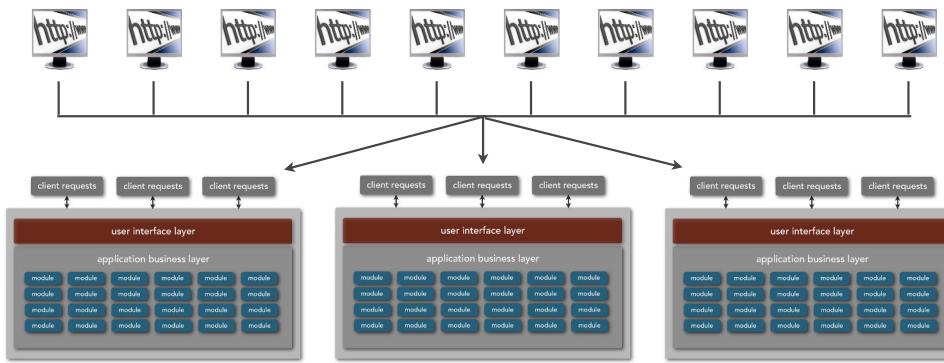
monolithic application issues

scalability



monolithic application issues

scalability



scaling monolithic applications requires you to scale the entire application, which can be both difficult and costly

monolithic application issues

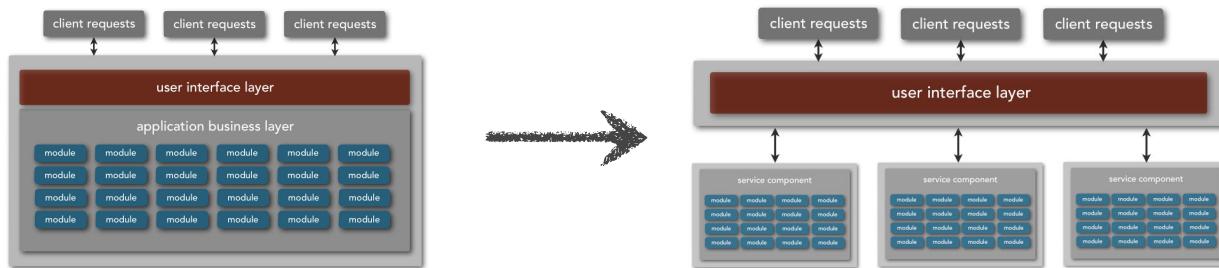
application size



monolithic applications can grow to quickly and consume all available resources

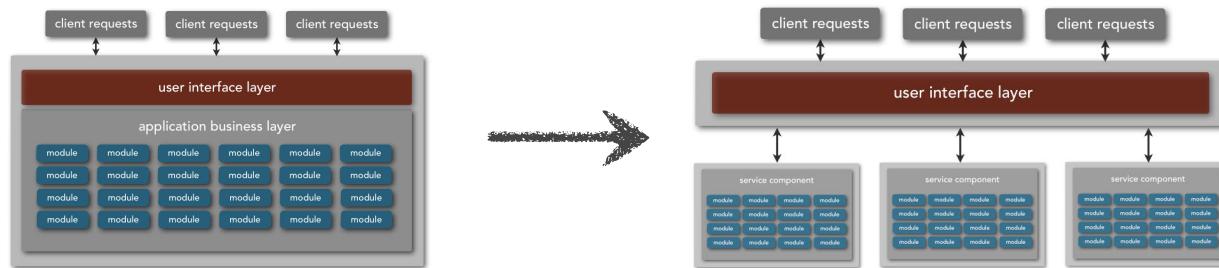
migration paths

monolithic application to service-based architecture



migration paths

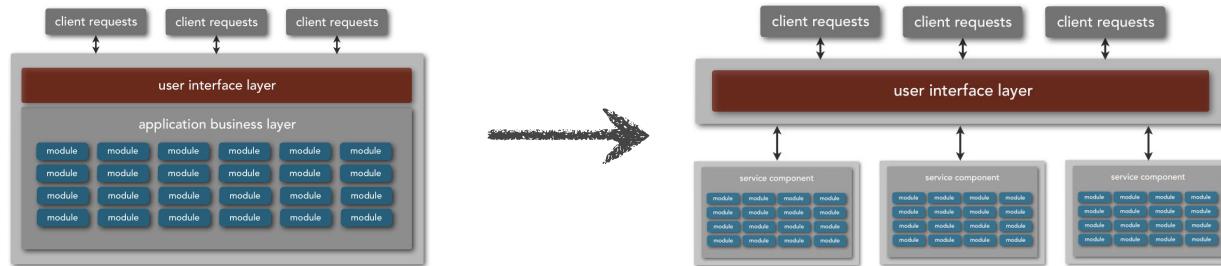
monolithic application to service-based architecture



—deployment

migration paths

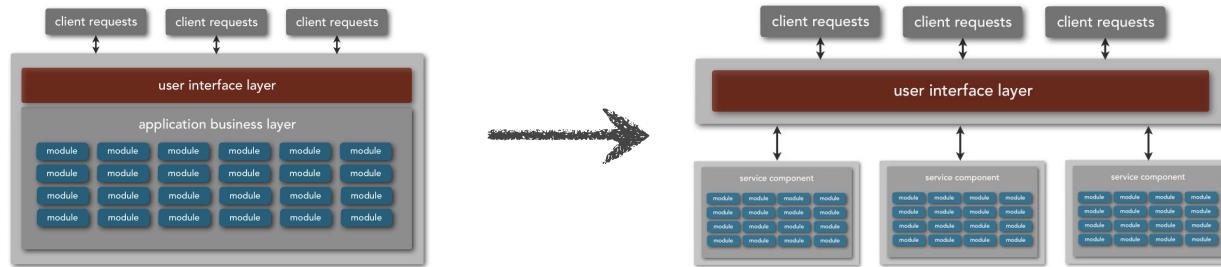
monolithic application to service-based architecture



— deployment
— reliability and robustness

migration paths

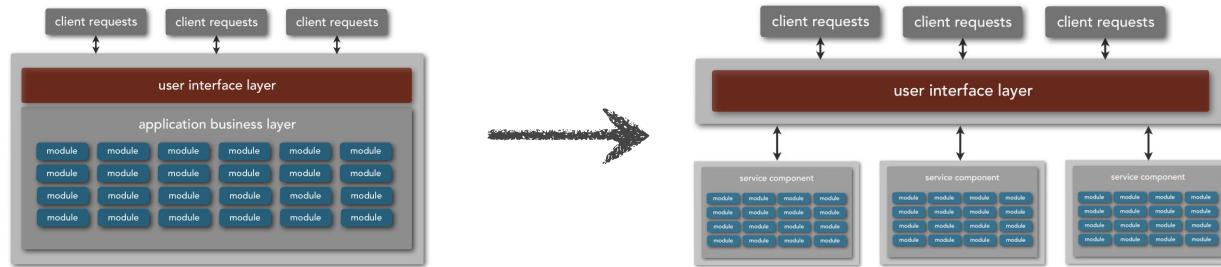
monolithic application to service-based architecture



- deployment
- reliability and robustness
- testability

migration paths

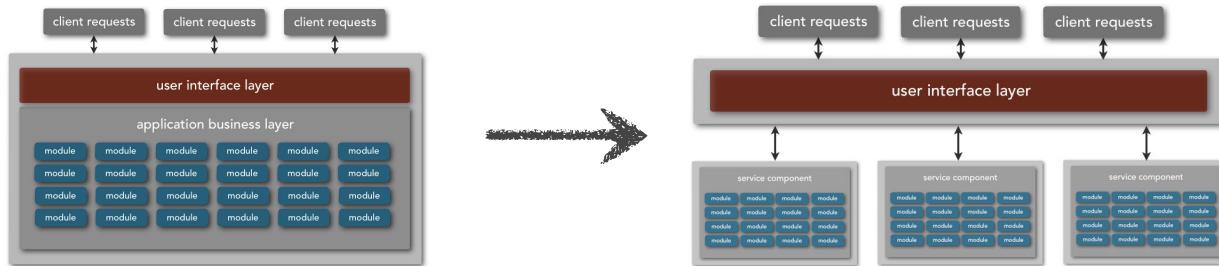
monolithic application to service-based architecture



- deployment
- reliability and robustness
- testability
- scalability

migration paths

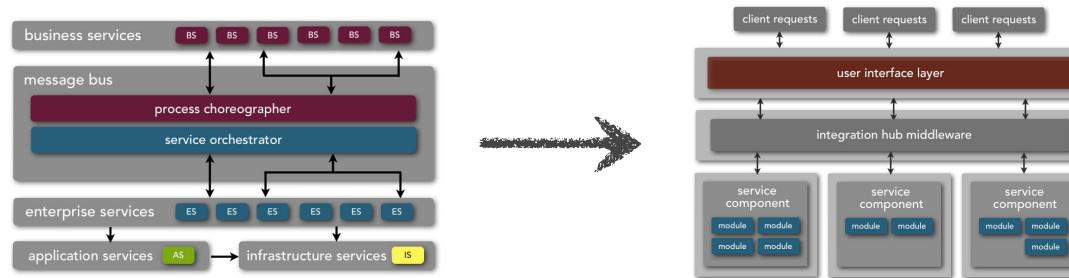
monolithic application to service-based architecture



- deployment
- reliability and robustness
- testability
- scalability
- application size

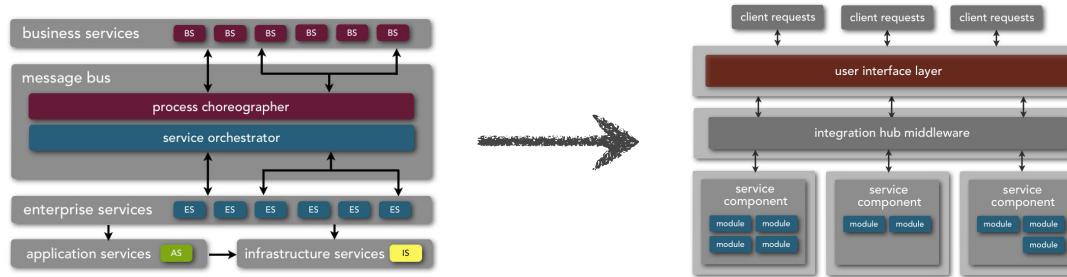
migration paths

soa to service-based architecture



migration paths

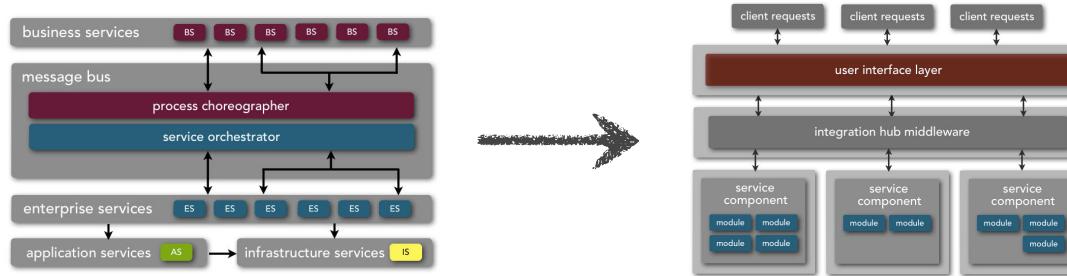
soa to service-based architecture



—complexity

migration paths

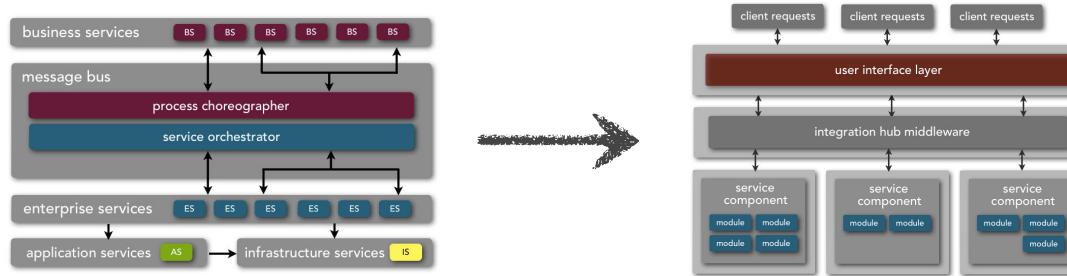
soa to service-based architecture



complexity
business involvement

migration paths

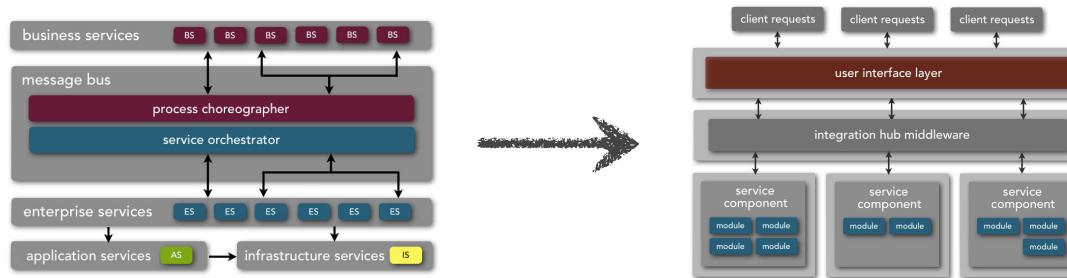
soa to service-based architecture



- complexity
- business involvement
- cost

migration paths

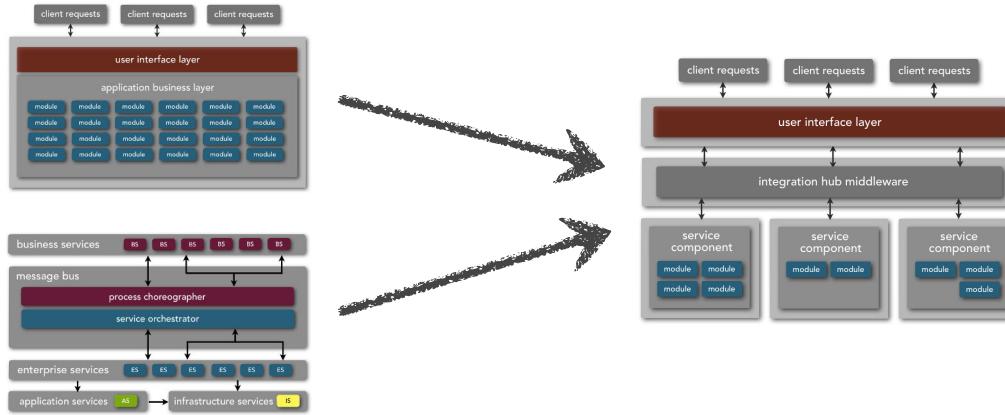
soa to service-based architecture



- complexity
- business involvement
- cost
- overkill

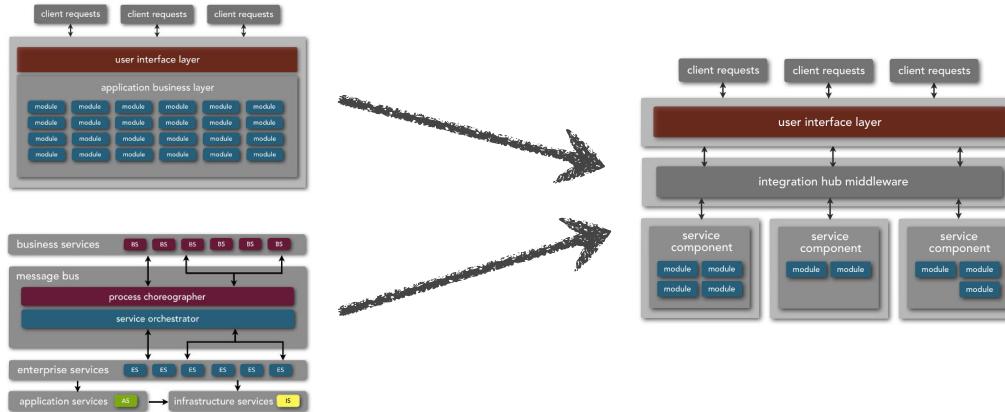
migration paths

business drivers



migration paths

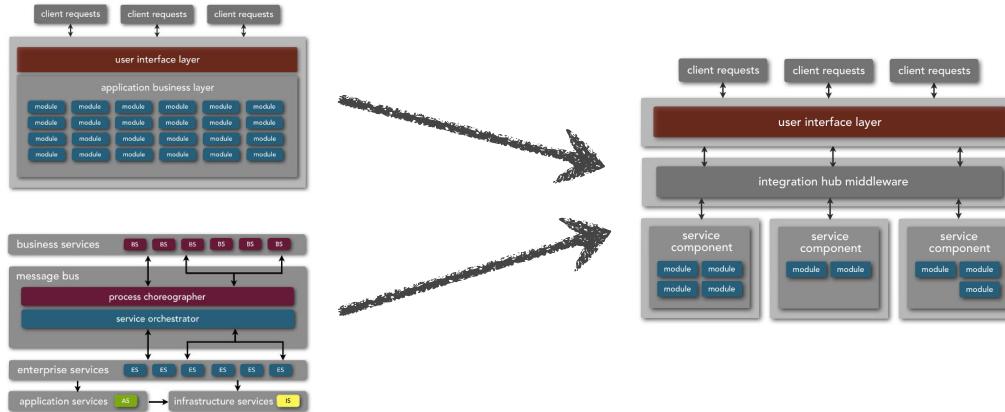
business drivers



— faster time to market

migration paths

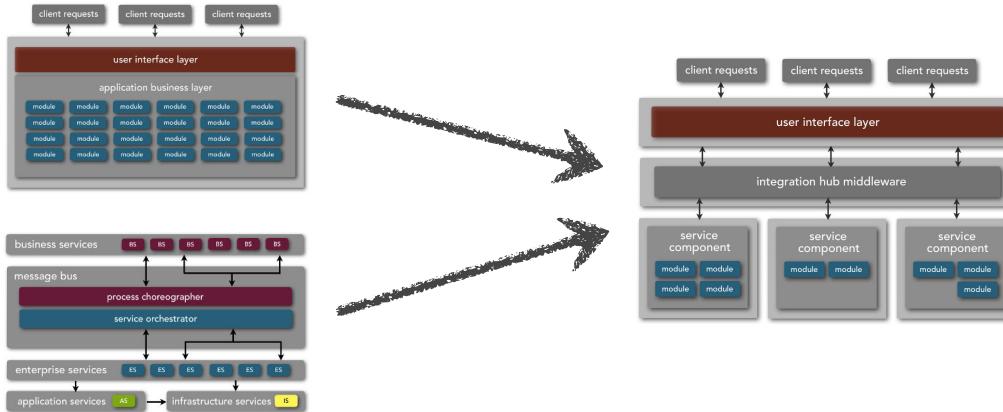
business drivers



— faster time to market
— improved reliability and quality

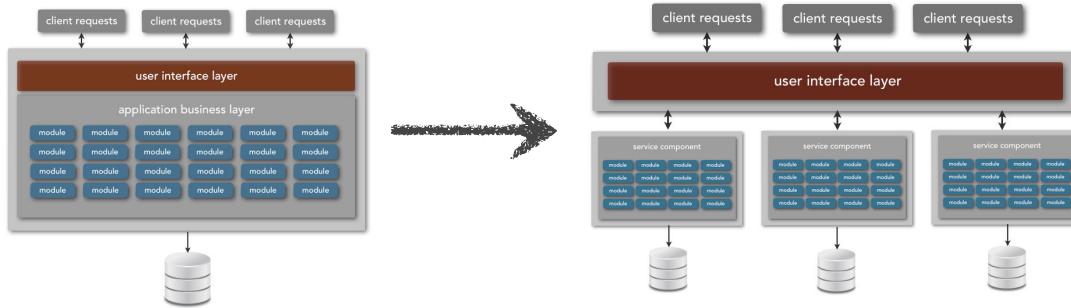
migration paths

business drivers



- faster time to market
- improved reliability and quality
- reduced costs

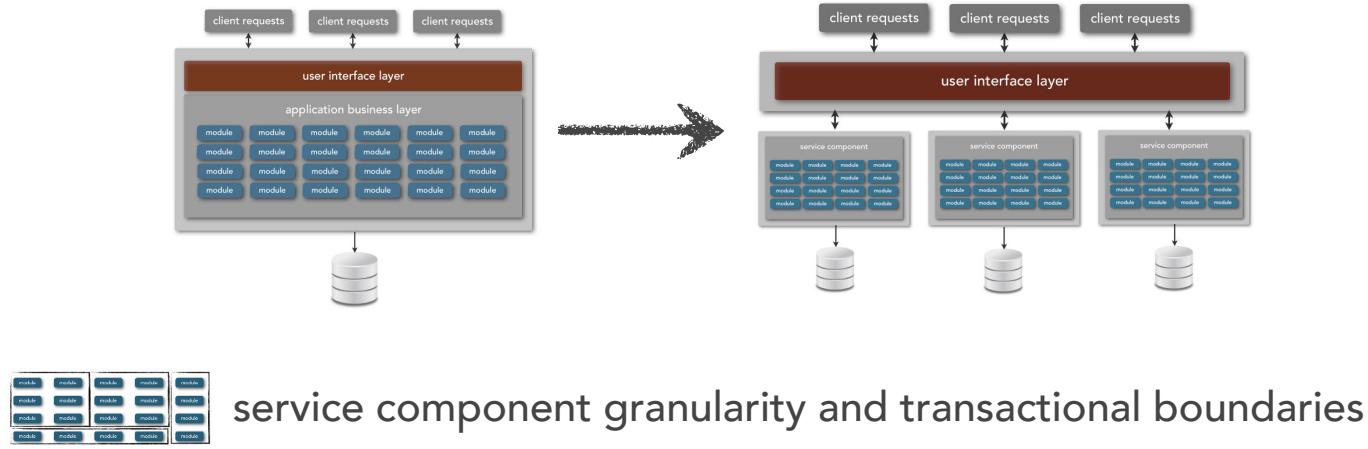
migration challenges



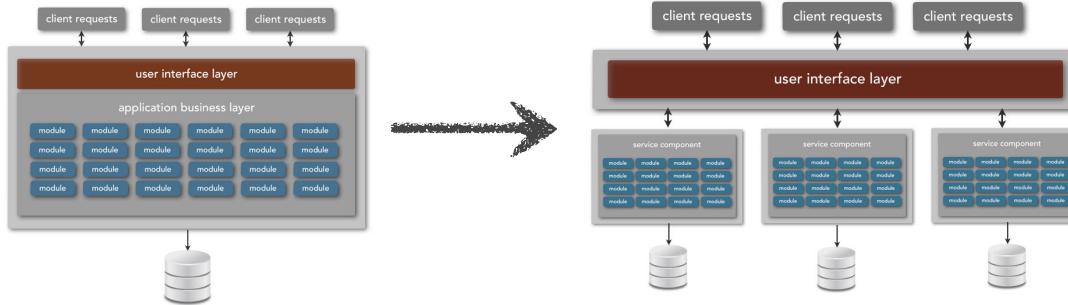
share everything
architecture

share as little as
possible architecture

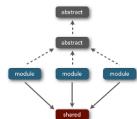
migration challenges



migration challenges

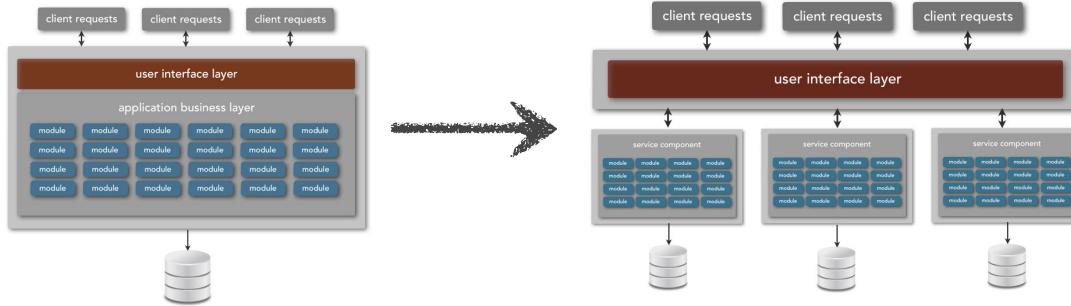


service component granularity and transactional boundaries

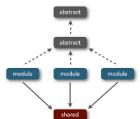


shared services, modules, and object hierarchies

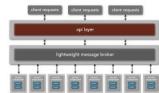
migration challenges



service component granularity and transactional boundaries

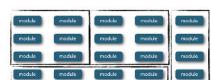
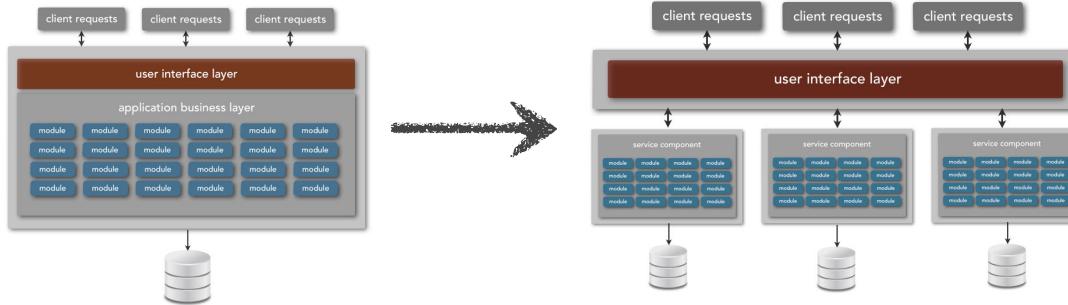


shared services, modules, and object hierarchies

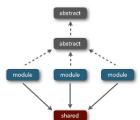


distributed architecture and remote service issues

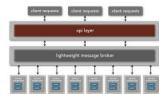
migration challenges



service component granularity and transactional boundaries



shared services, modules, and object hierarchies



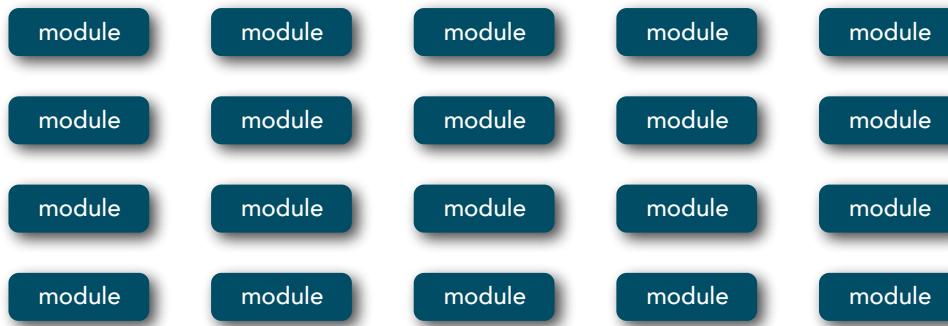
distributed architecture and remote service issues



large shared relational database

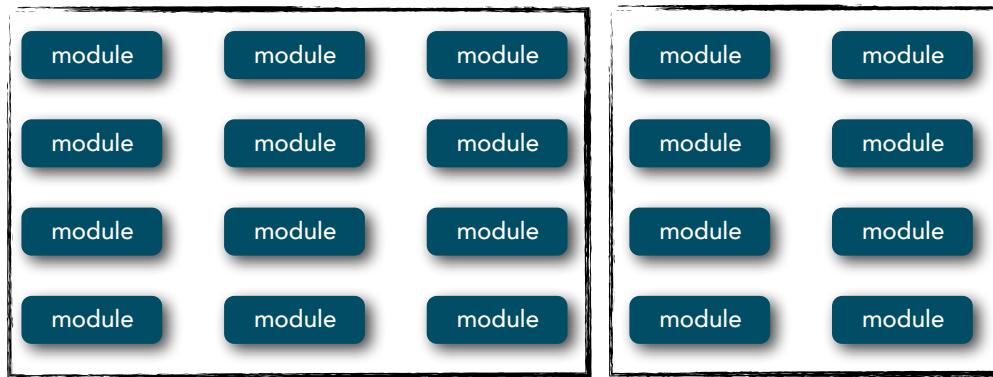
migration challenges

service component granularity



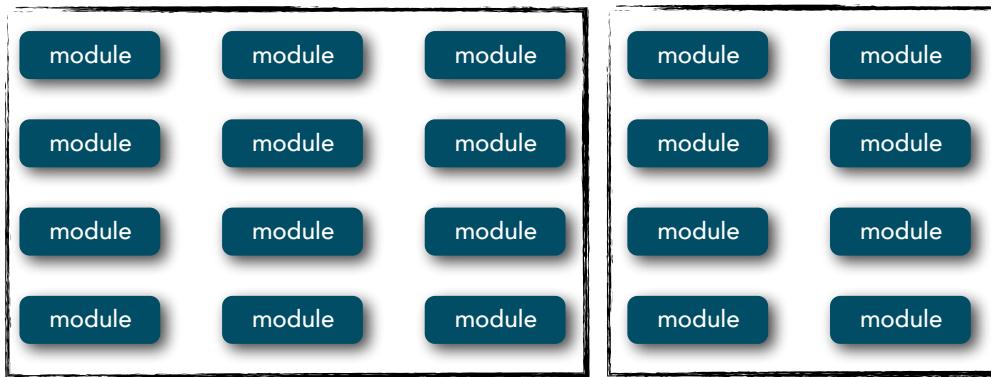
migration challenges

service component granularity



migration challenges

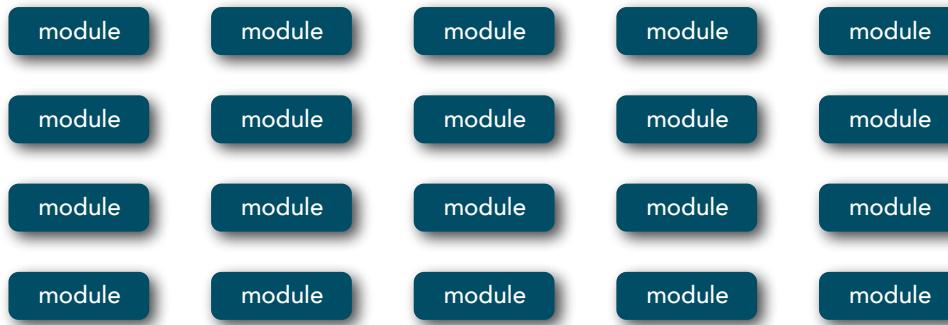
service component granularity



coarse-grained service components address transactional issues but may not achieve your desired goals

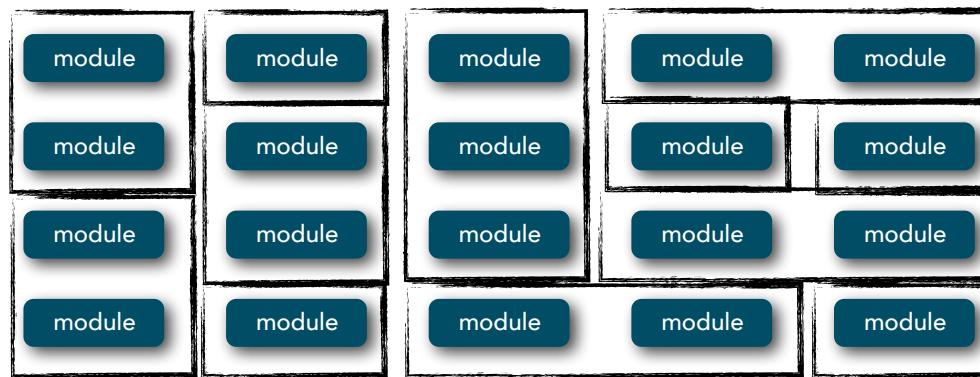
migration challenges

service component granularity



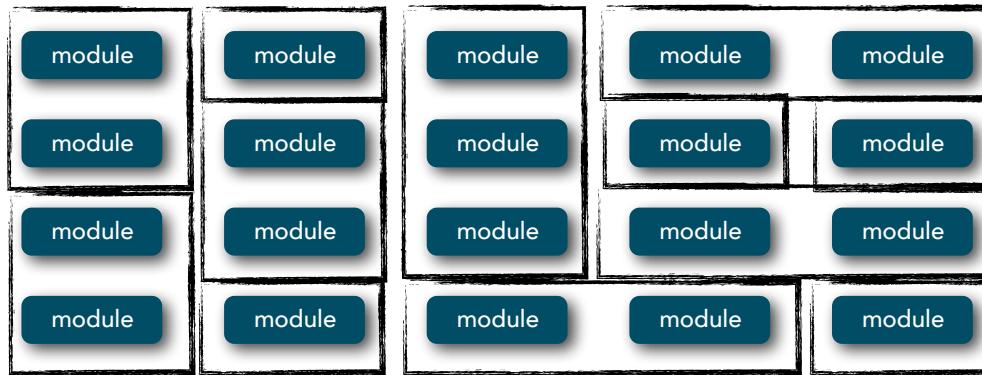
migration challenges

service component granularity



migration challenges

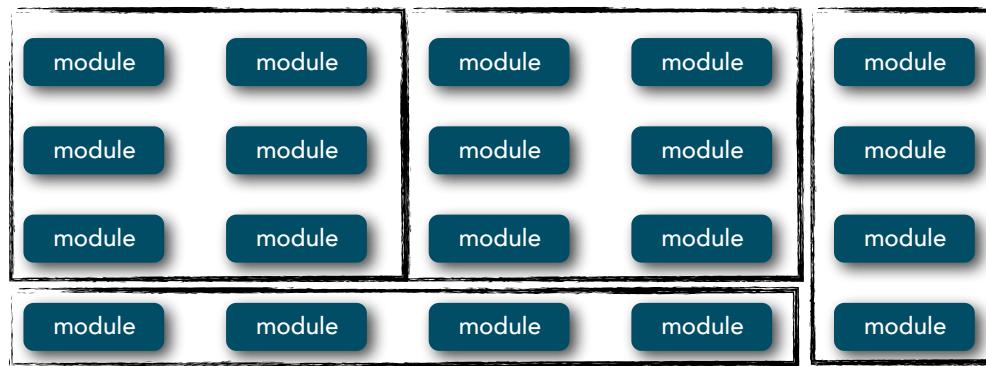
service component granularity



fine-grained service components may lead to
too much orchestration and inter-dependency
between service components

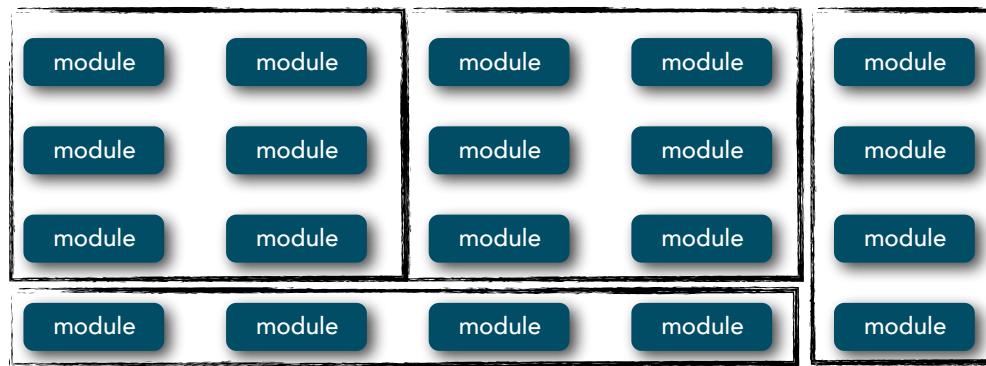
migration challenges

service component granularity



migration challenges

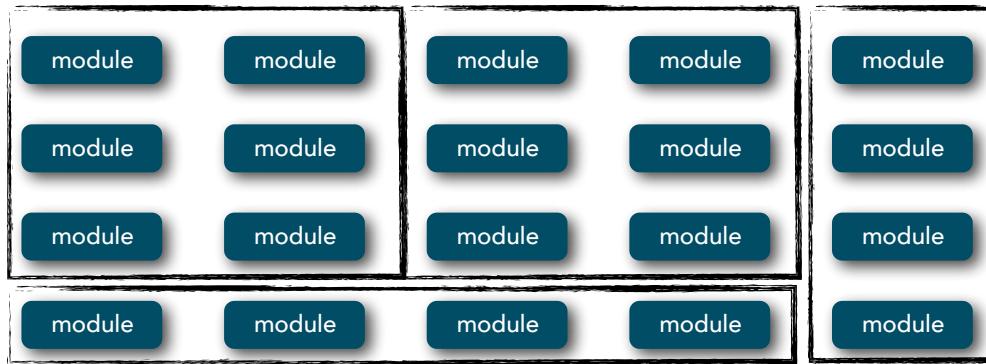
service component granularity



business functionality groupings

migration challenges

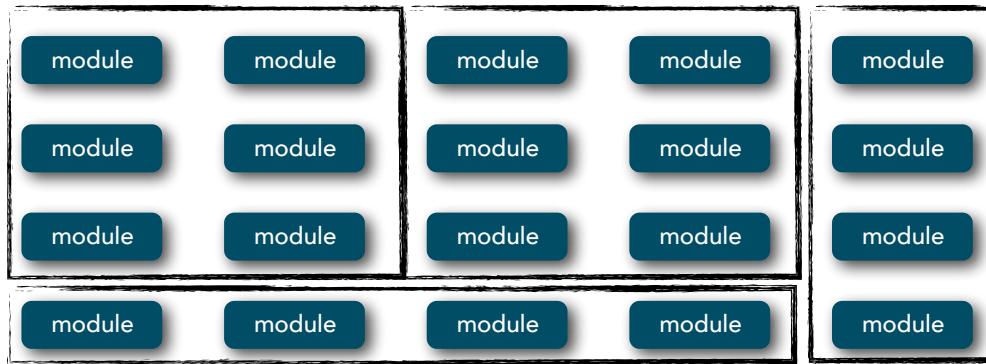
service component granularity



business functionality groupings
transactional boundaries

migration challenges

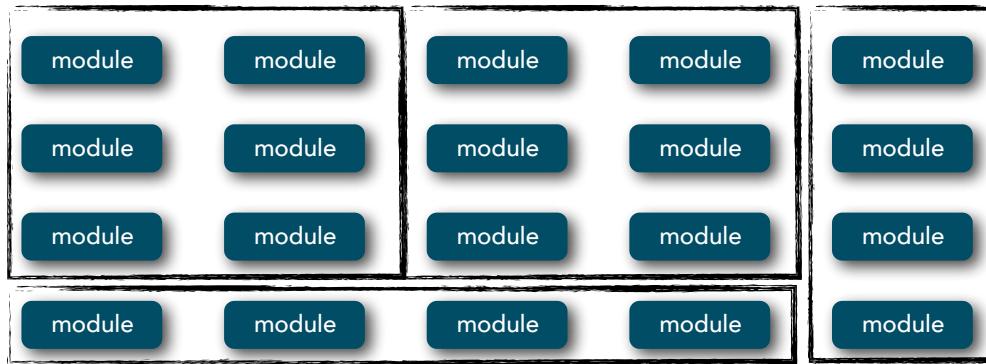
service component granularity



business functionality groupings
transactional boundaries
deployment goals

migration challenges

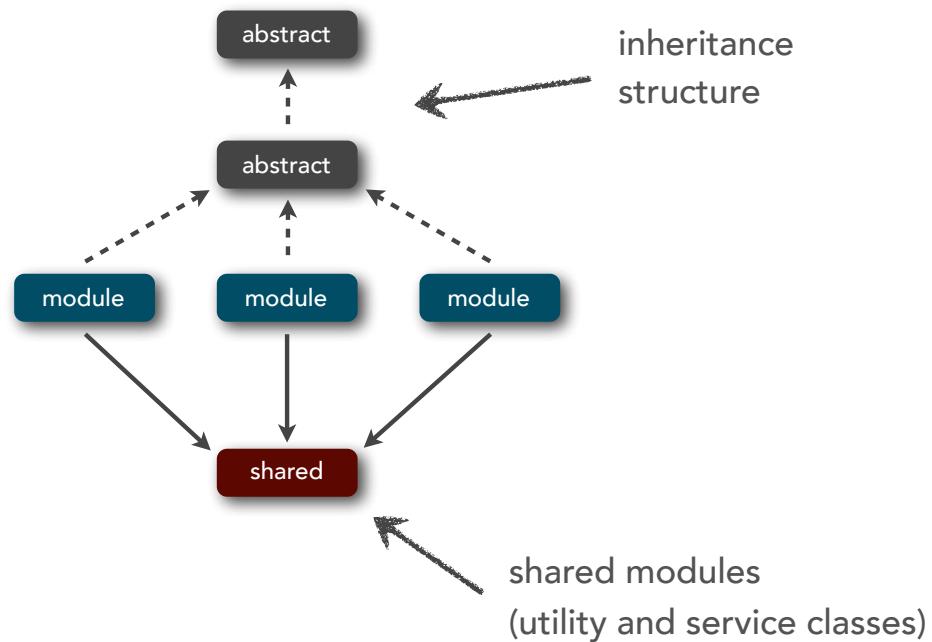
service component granularity



business functionality groupings
transactional boundaries
deployment goals
scalability needs

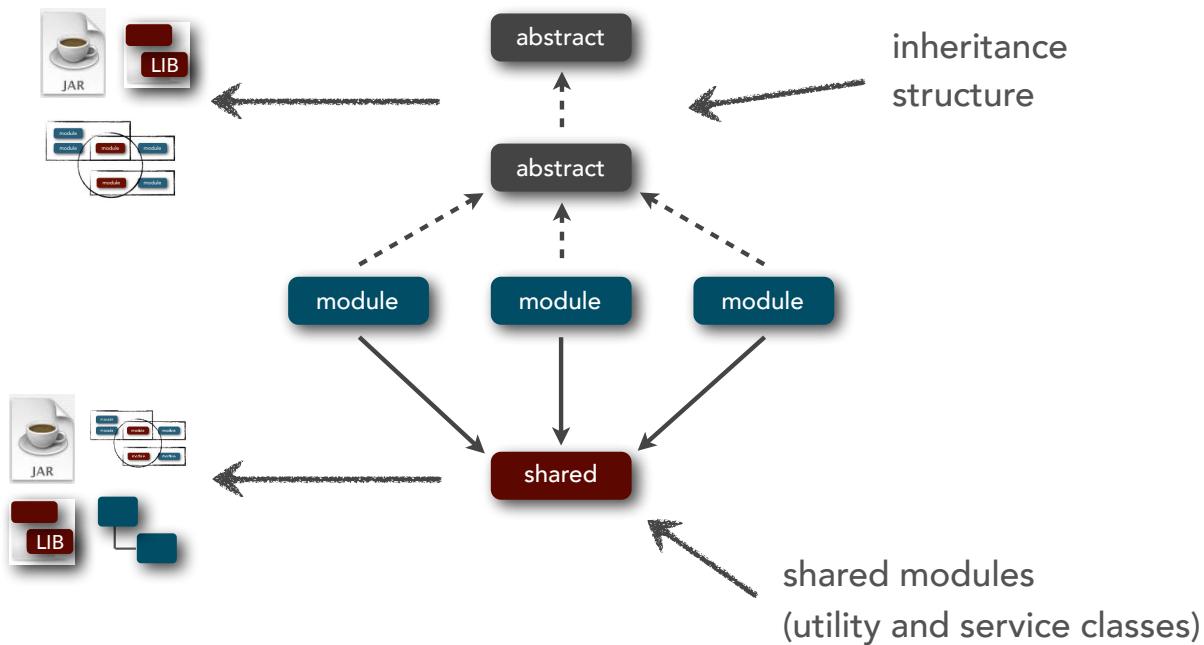
migration challenges

shared components and object hierarchies



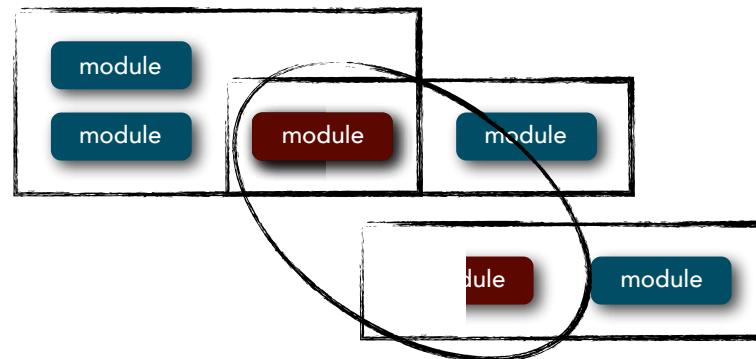
migration challenges

shared components and object hierarchies



migration challenges

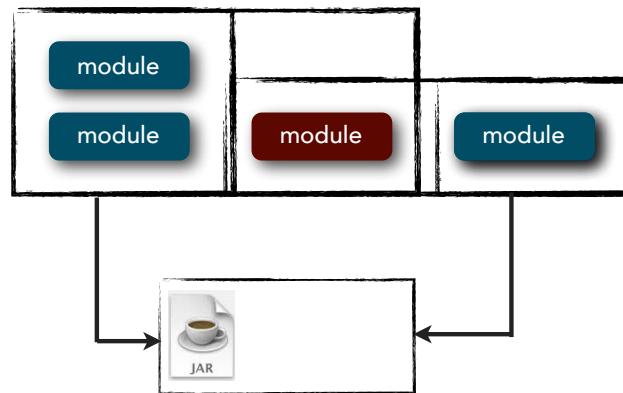
shared components and object hierarchies



modules can be split....

migration challenges

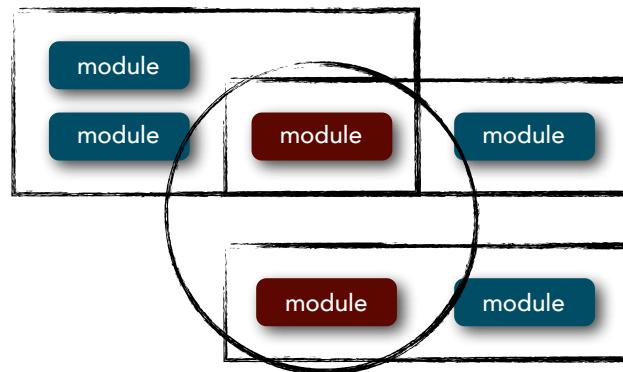
shared components and object hierarchies



or moved into a shared library or jar file...

migration challenges

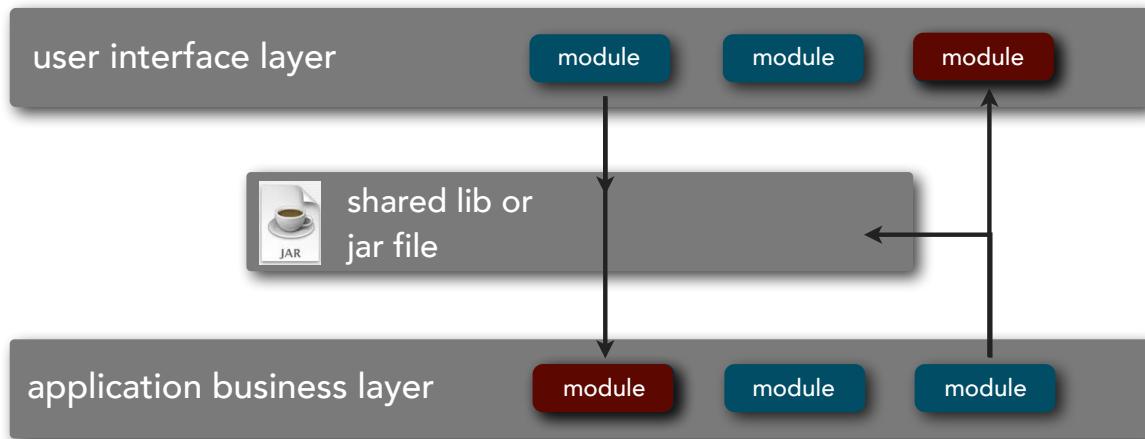
shared components and object hierarchies



or replicated in each service component

migration challenges

shared components and object hierarchies



migration challenges

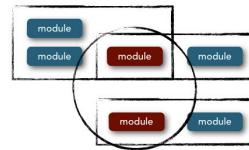
shared component techniques



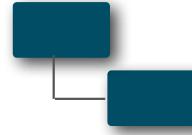
jar /dll
(compile or runtime)



shared library
(compile time)



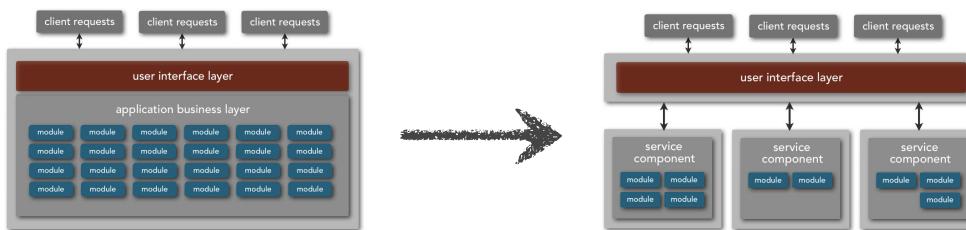
code replication



remote services

migration steps and techniques

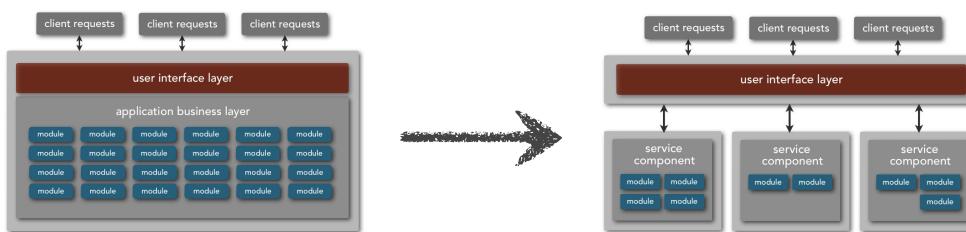
basic approach



focus on the business layer first; do the ui layer later if needed

migration steps and techniques

basic approach

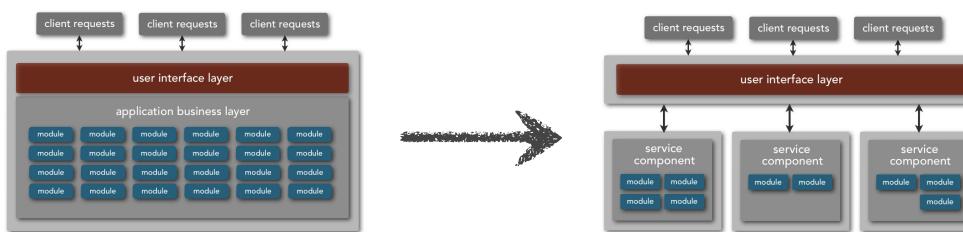


focus on the business layer first; do the ui layer later if needed

approach in small, incremental changes

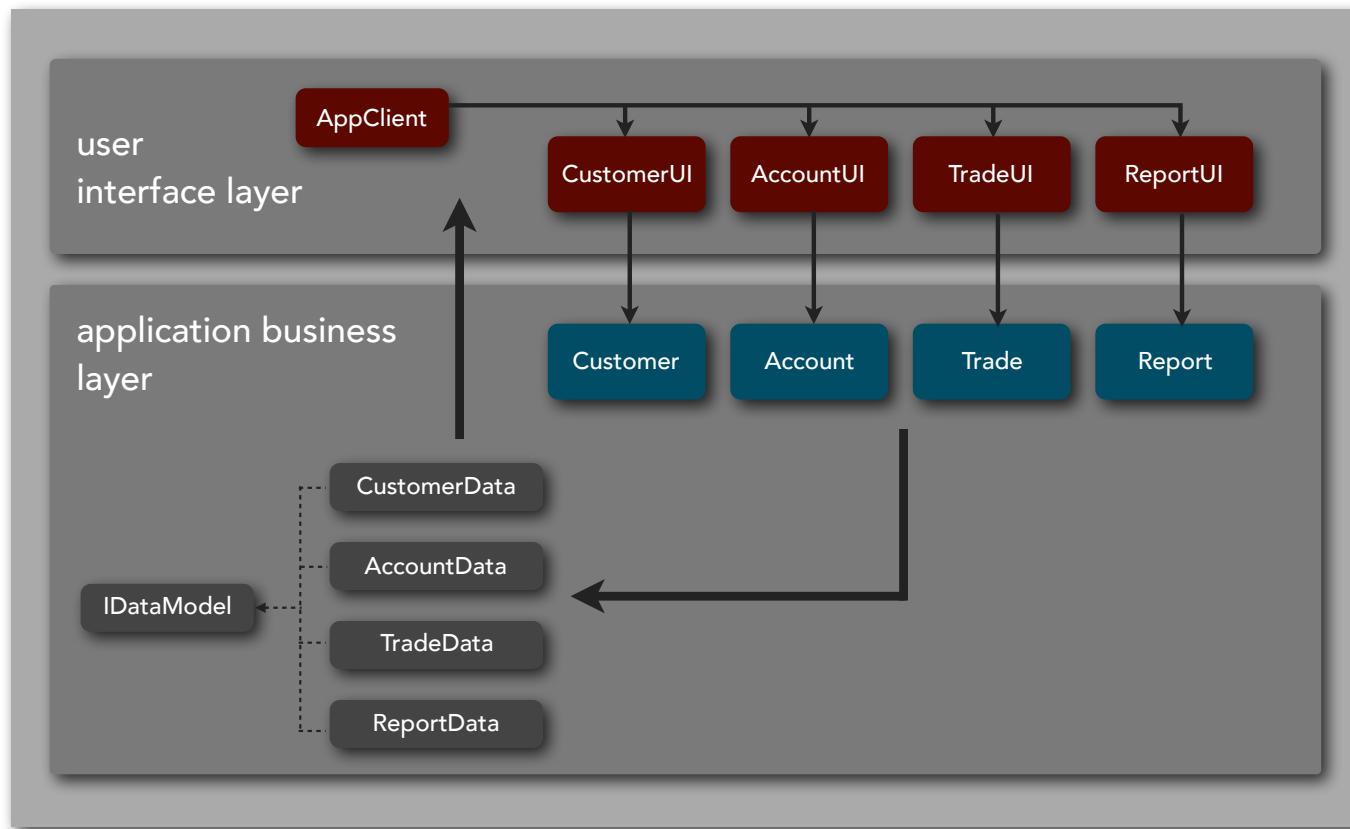
migration steps and techniques

basic approach

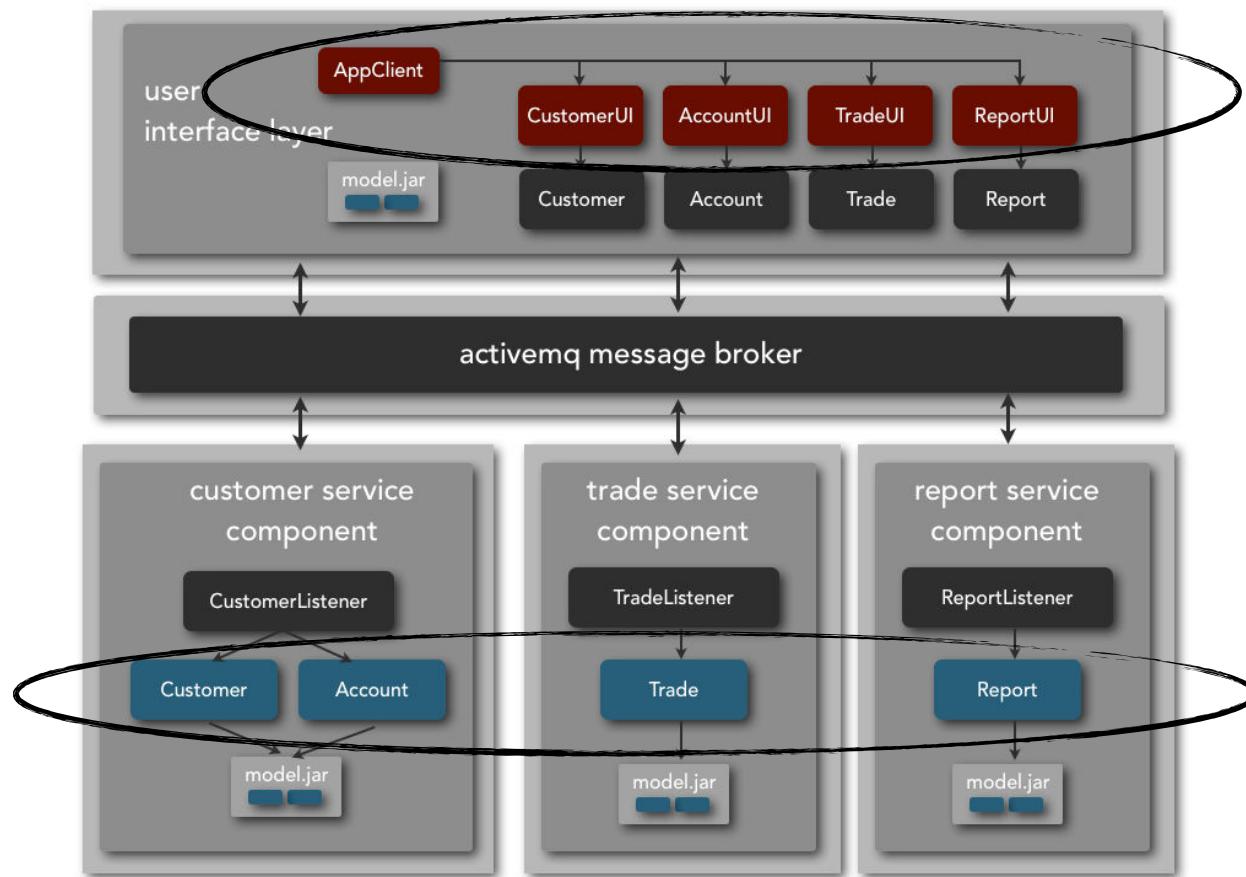


focus on the business layer first; do the ui layer later if needed
approach in small, incremental changes
evolve the service components through refactoring and migration

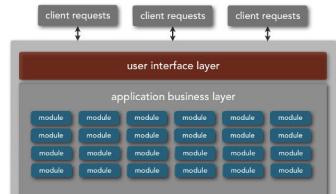
migration steps and techniques



migration steps and techniques

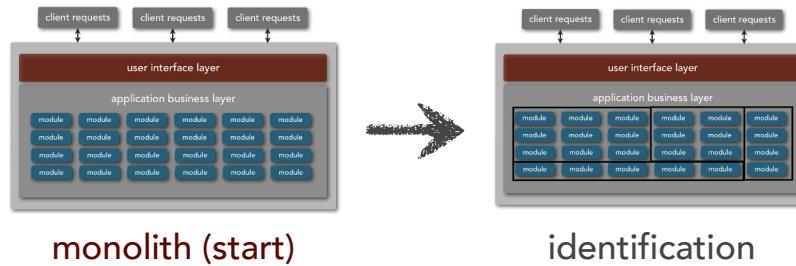


migration steps and techniques

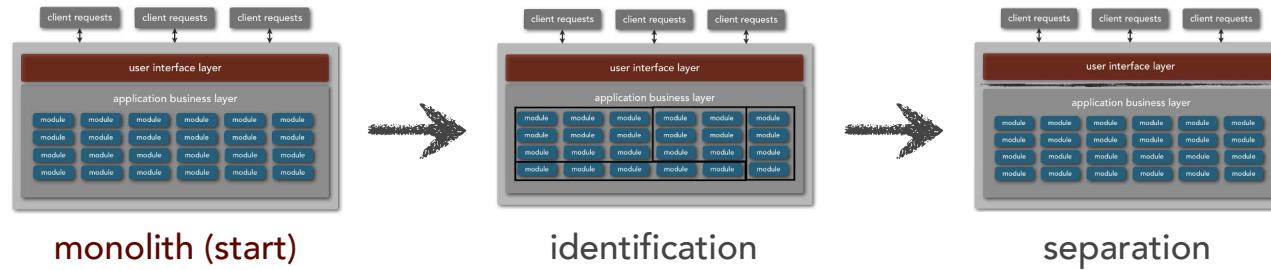


monolith (start)

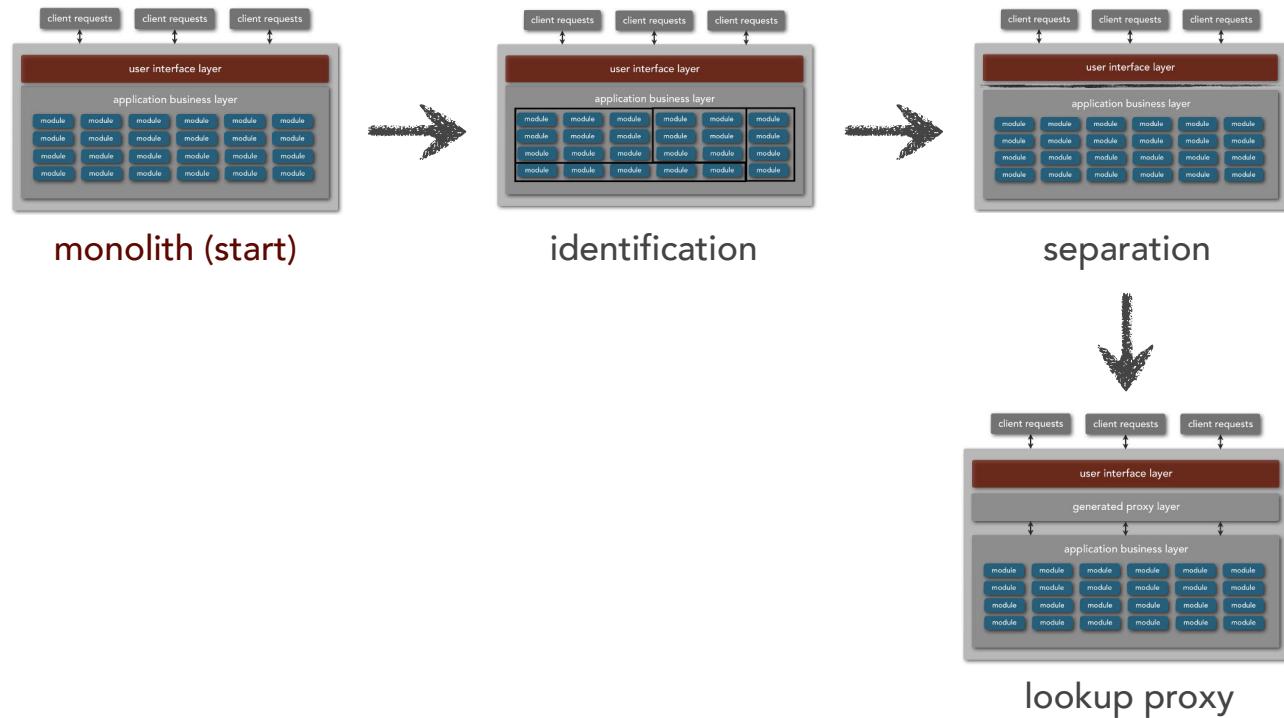
migration steps and techniques



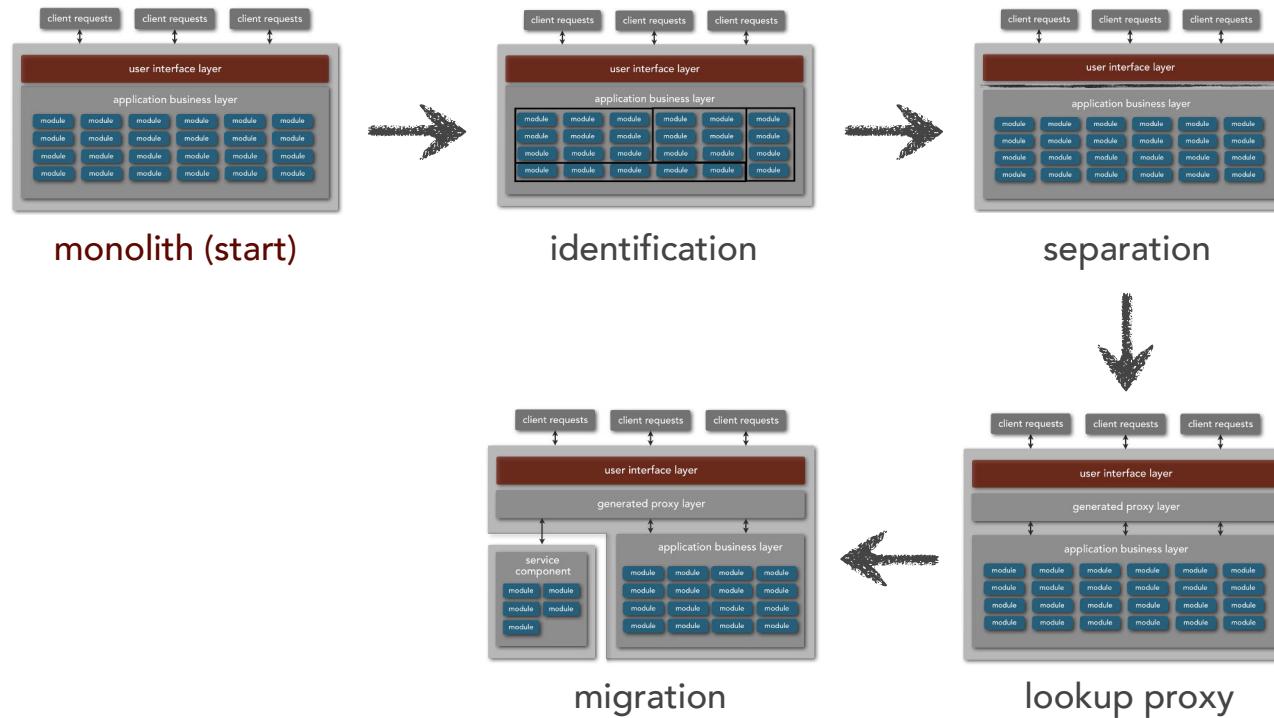
migration steps and techniques



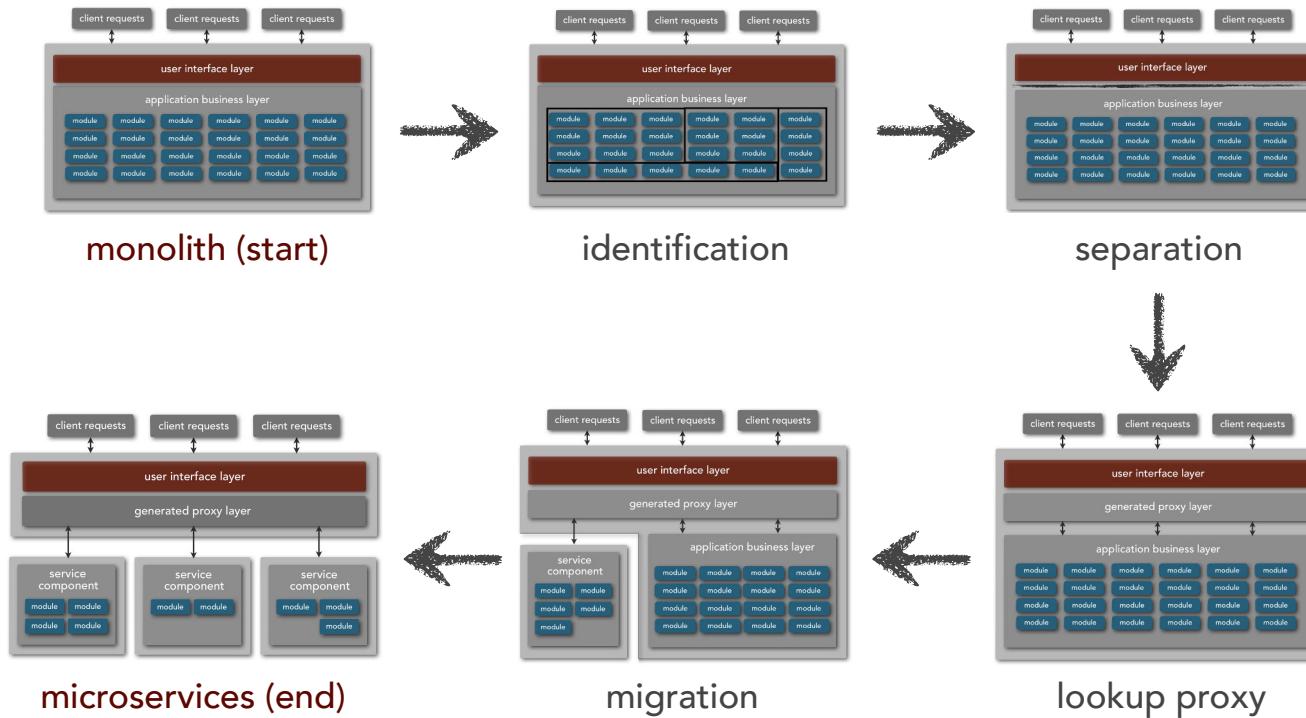
migration steps and techniques



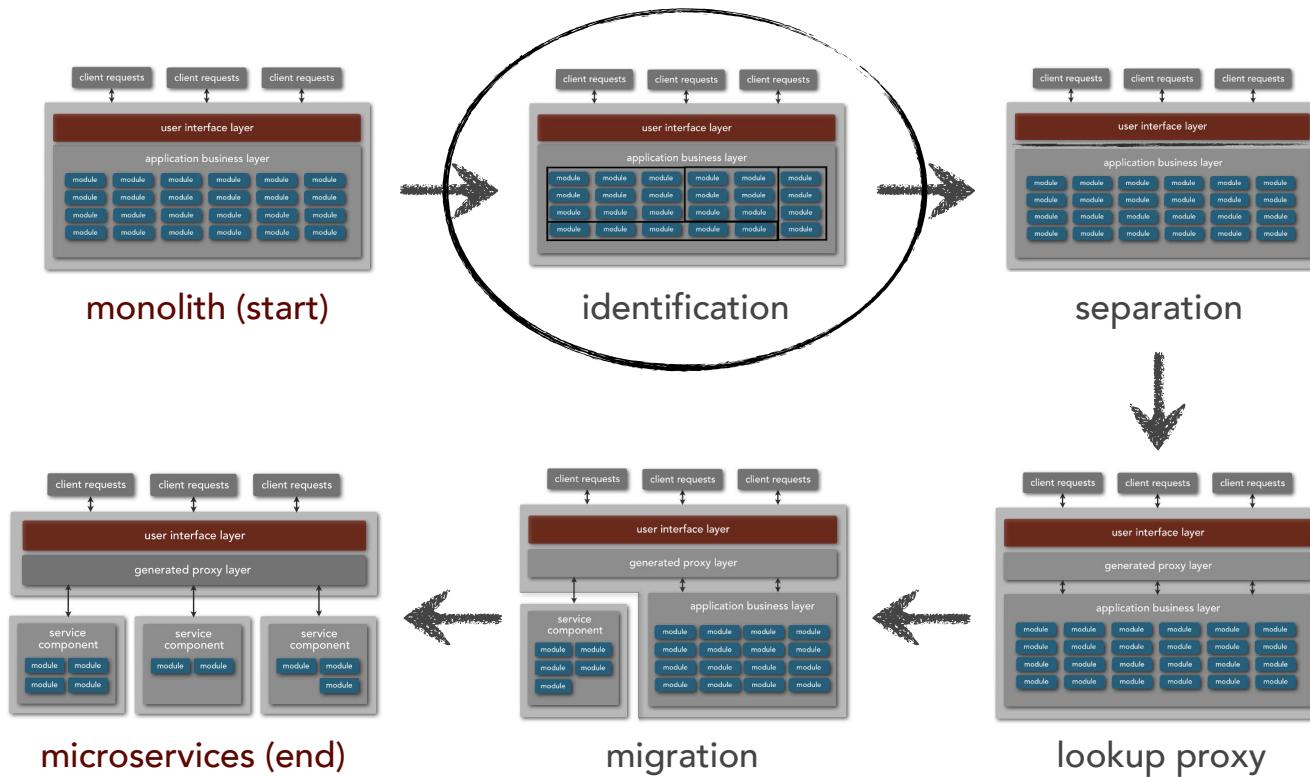
migration steps and techniques



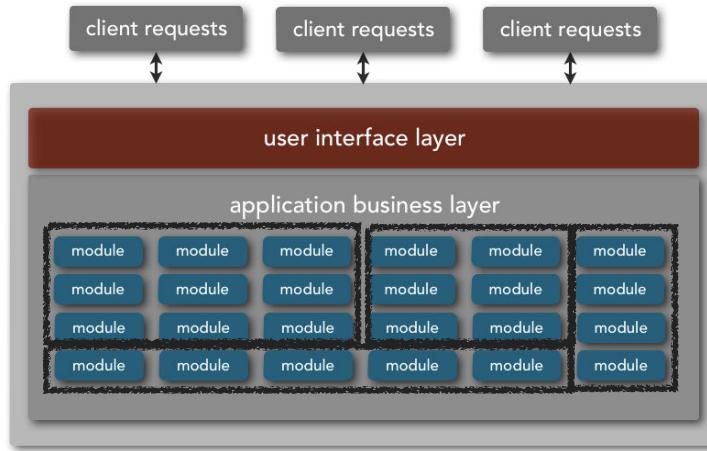
migration steps and techniques



migration steps and techniques

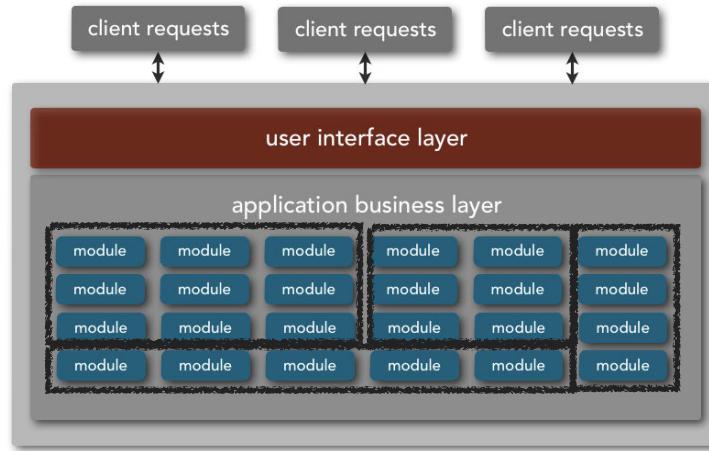


migration steps - identification



identify logical areas of the application that would form the service components

migration steps - identification

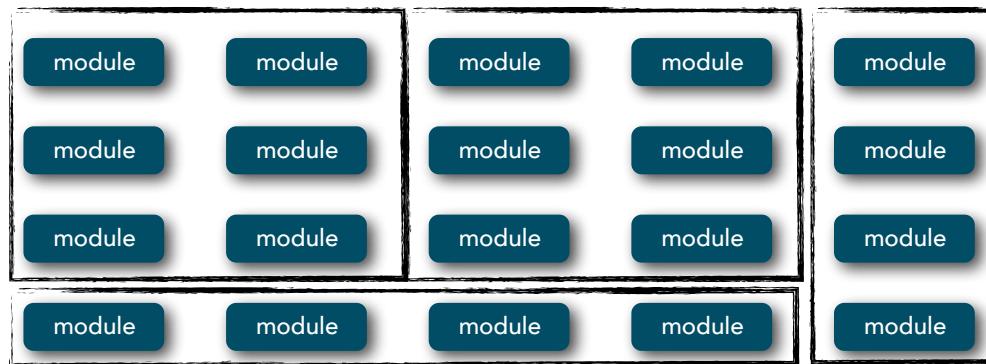


identify logical areas of the application that would form the service components

identify shared components and base classes

migration steps - identification

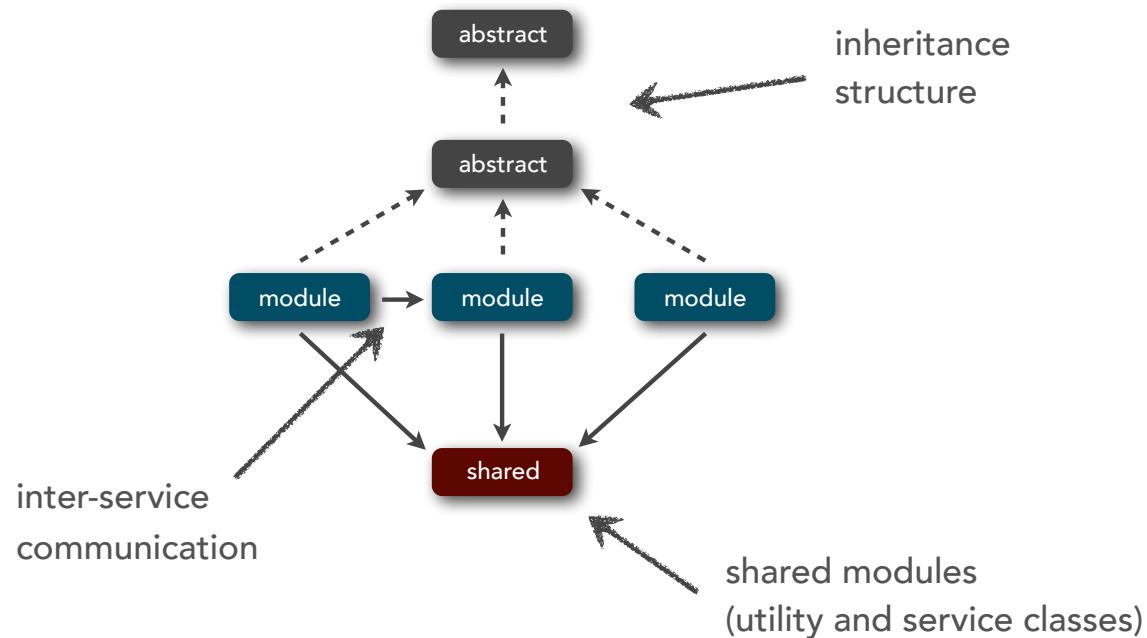
service component granularity



business functionality groupings
transactional boundaries
deployment goals
scalability needs

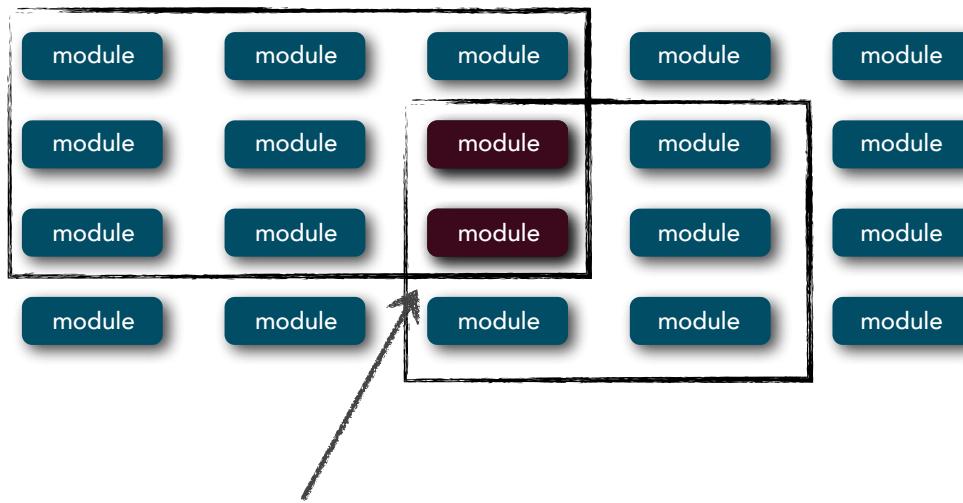
migration steps - identification

shared components



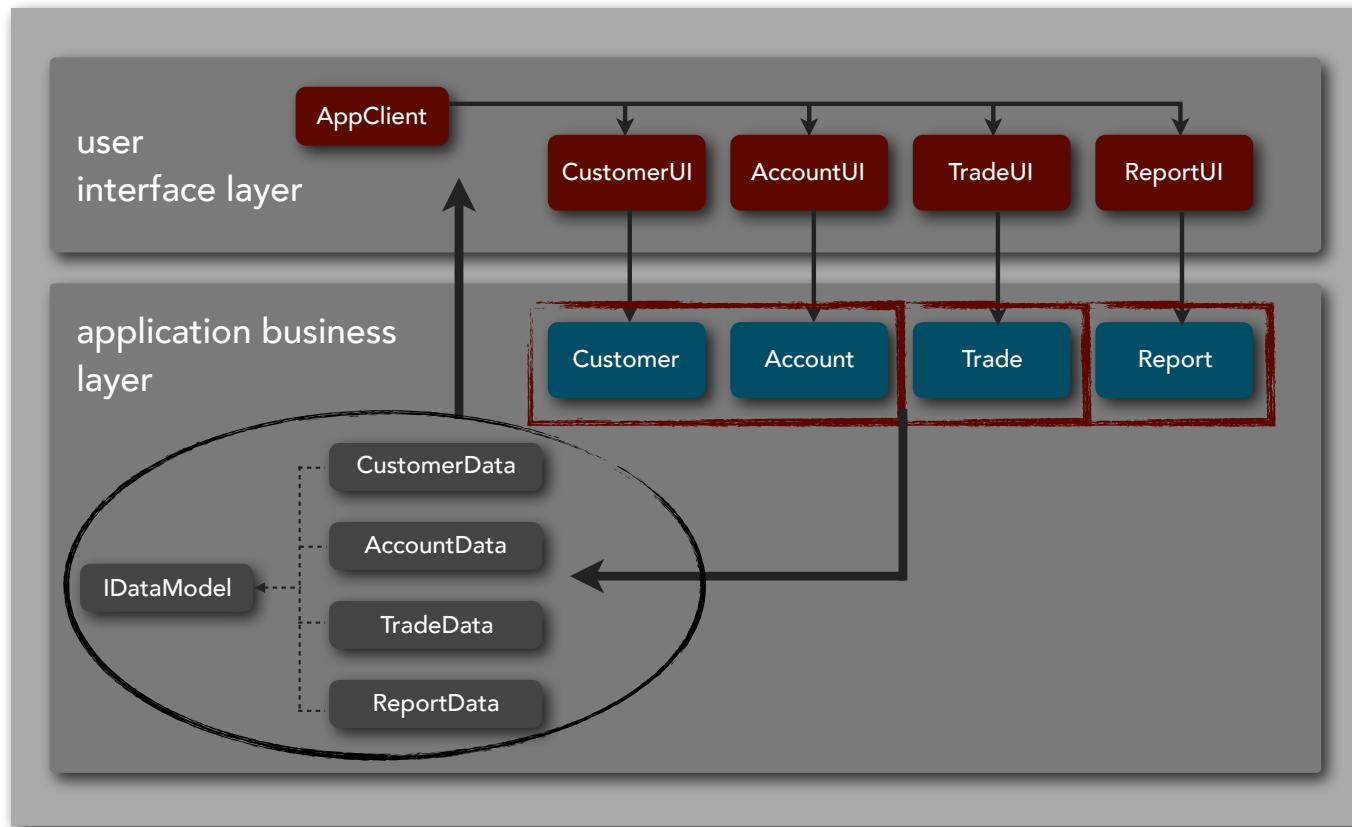
migration steps - identification

overlapping functionality

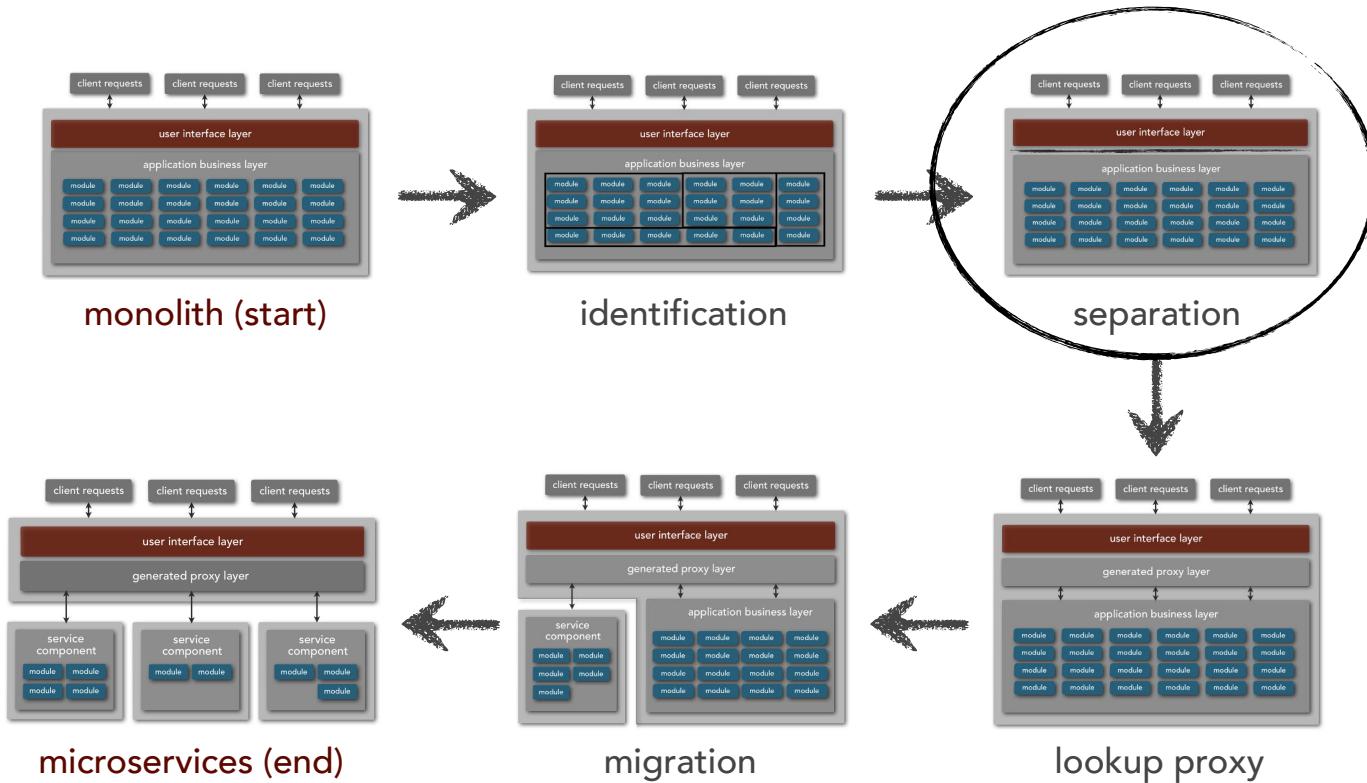


look for sharing among **business objects** that cross over component boundaries

migration steps - identification

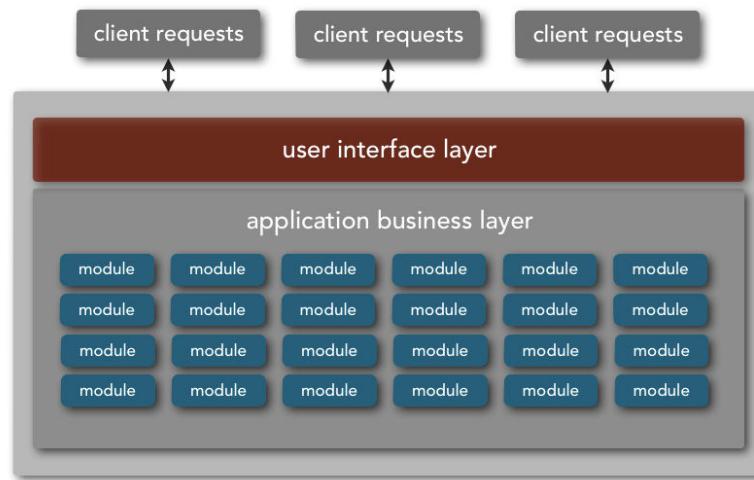


migration steps - separation



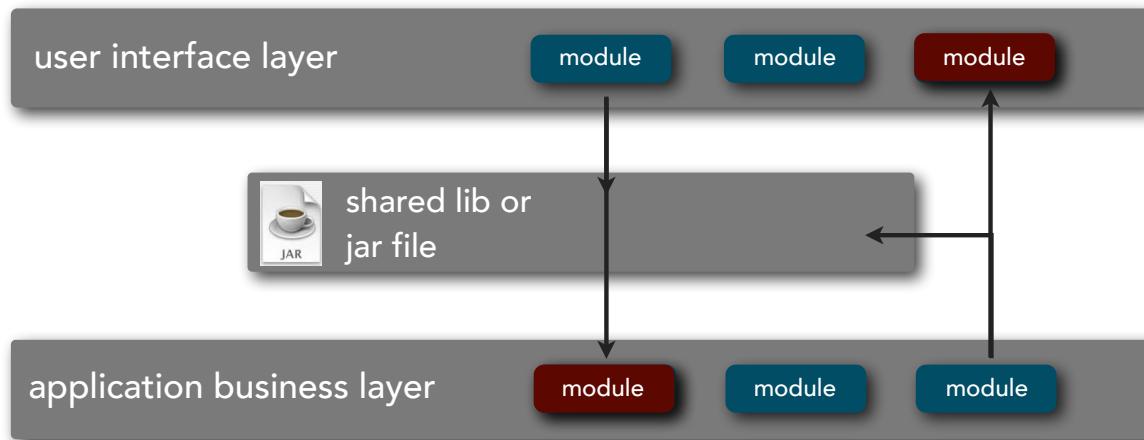
migration steps - separation

user interface separation

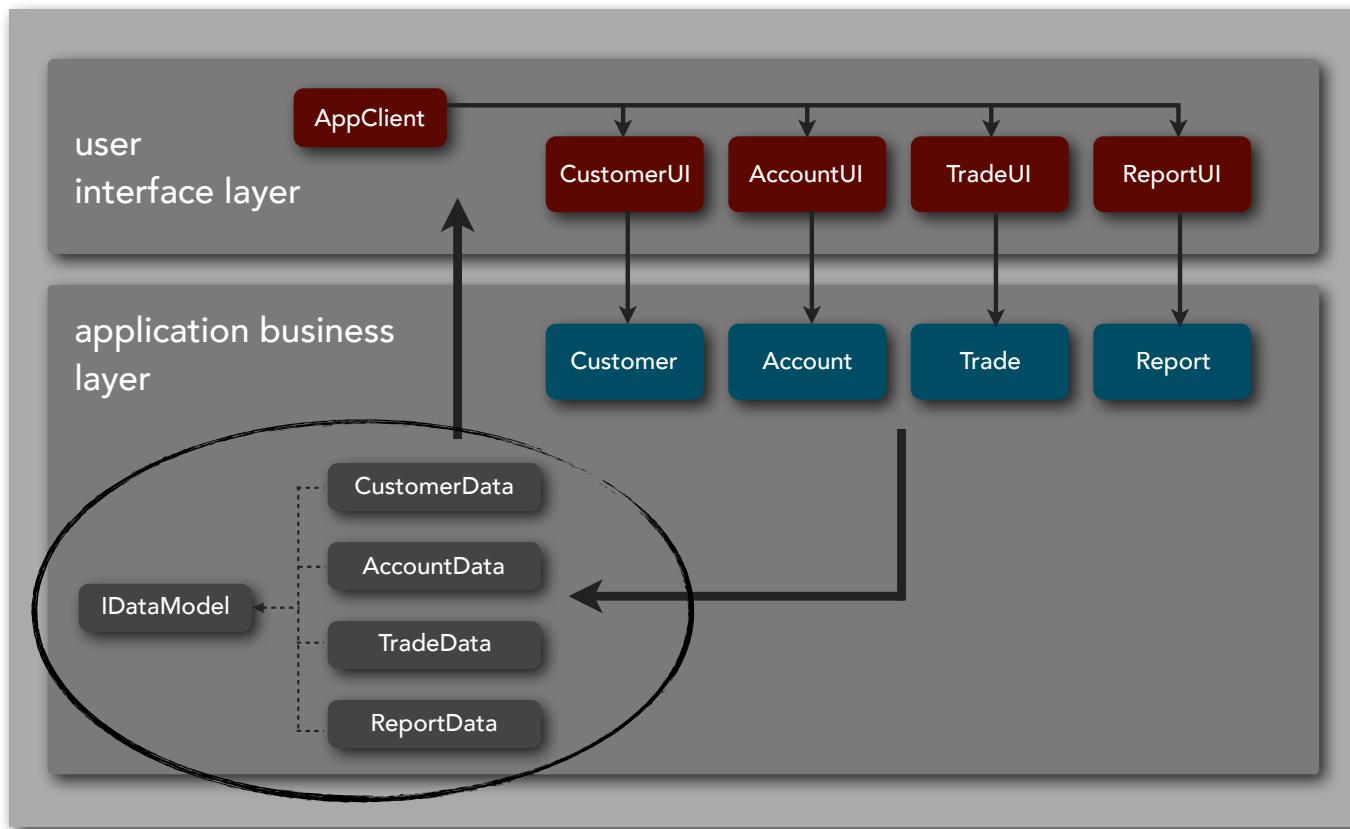


form a logical or physical separation between the user interface layer and the business layer

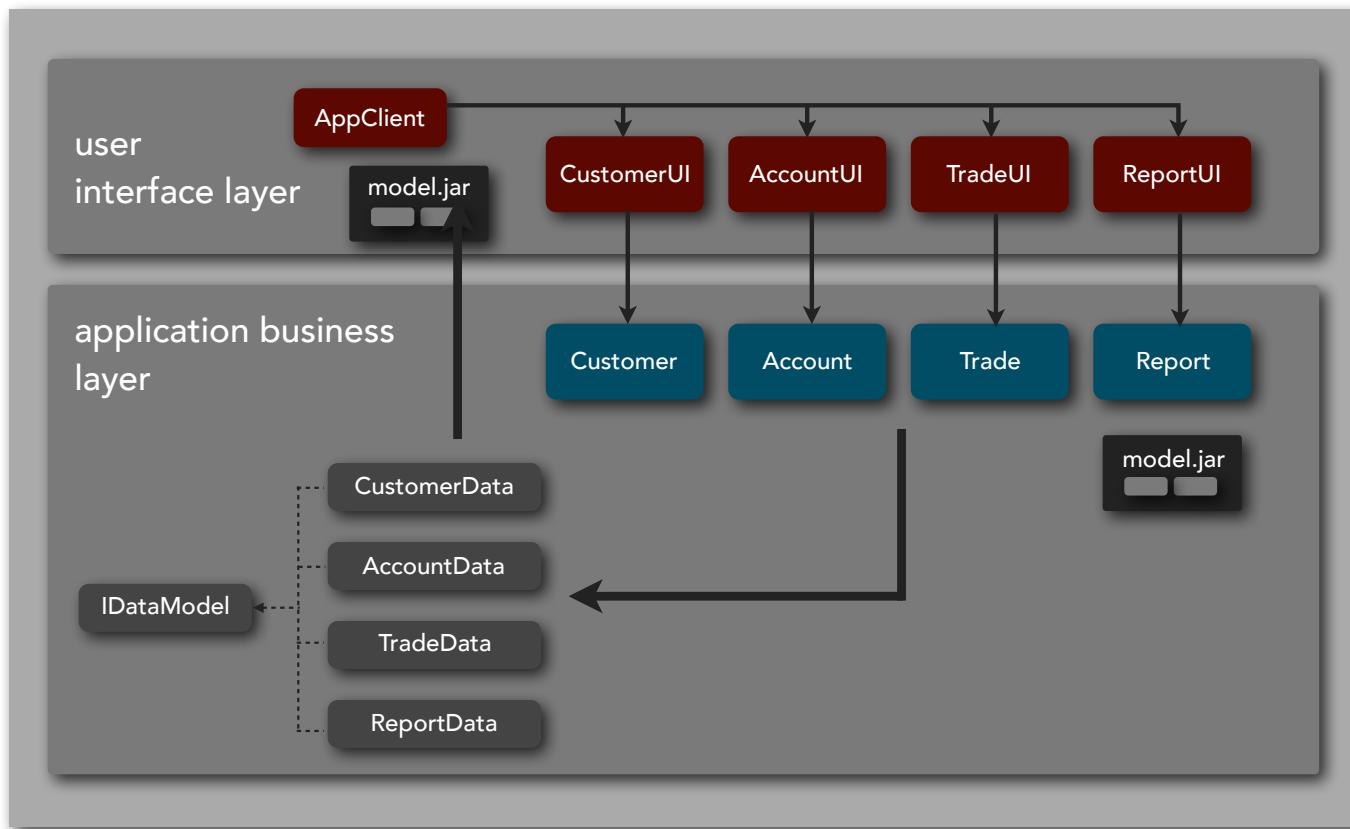
migration steps - separation



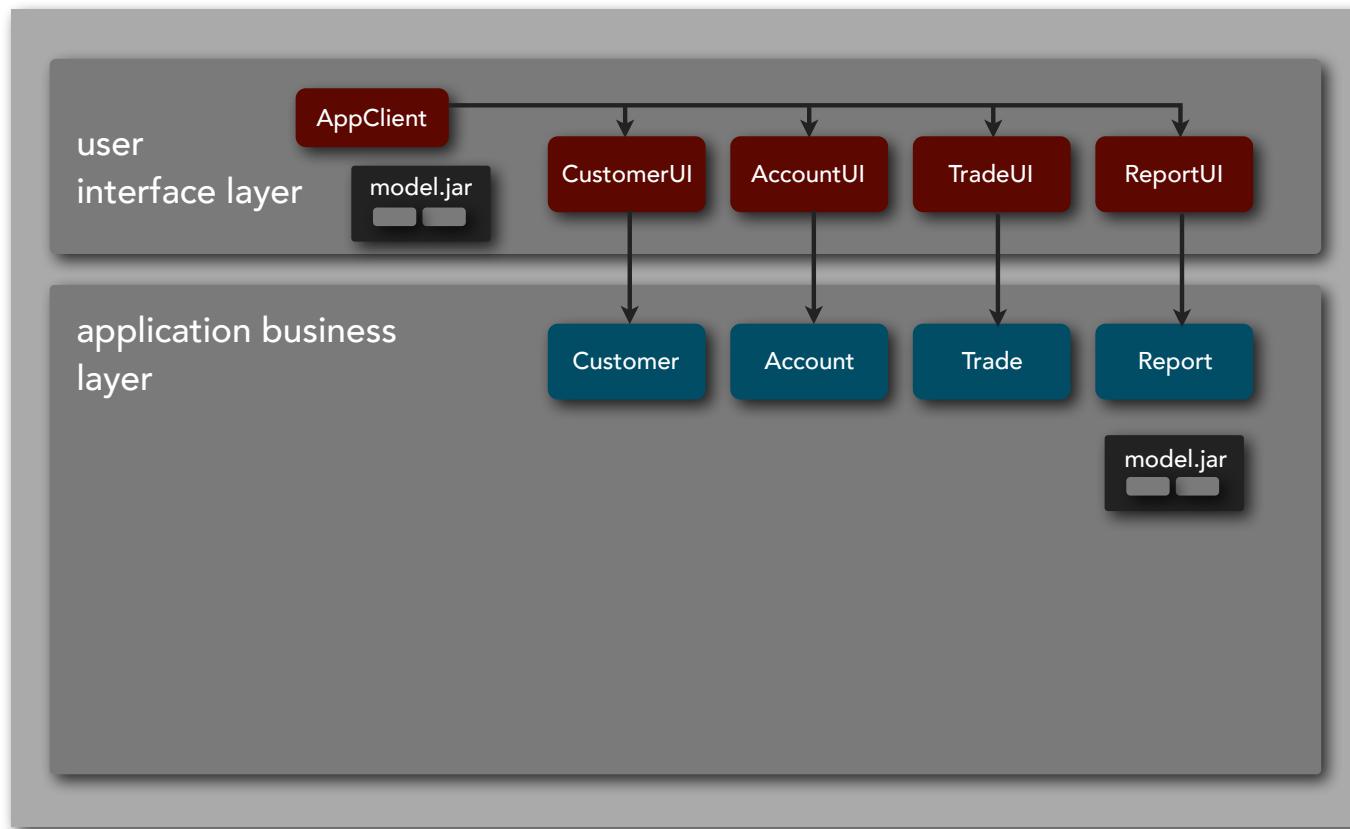
migration steps - separation



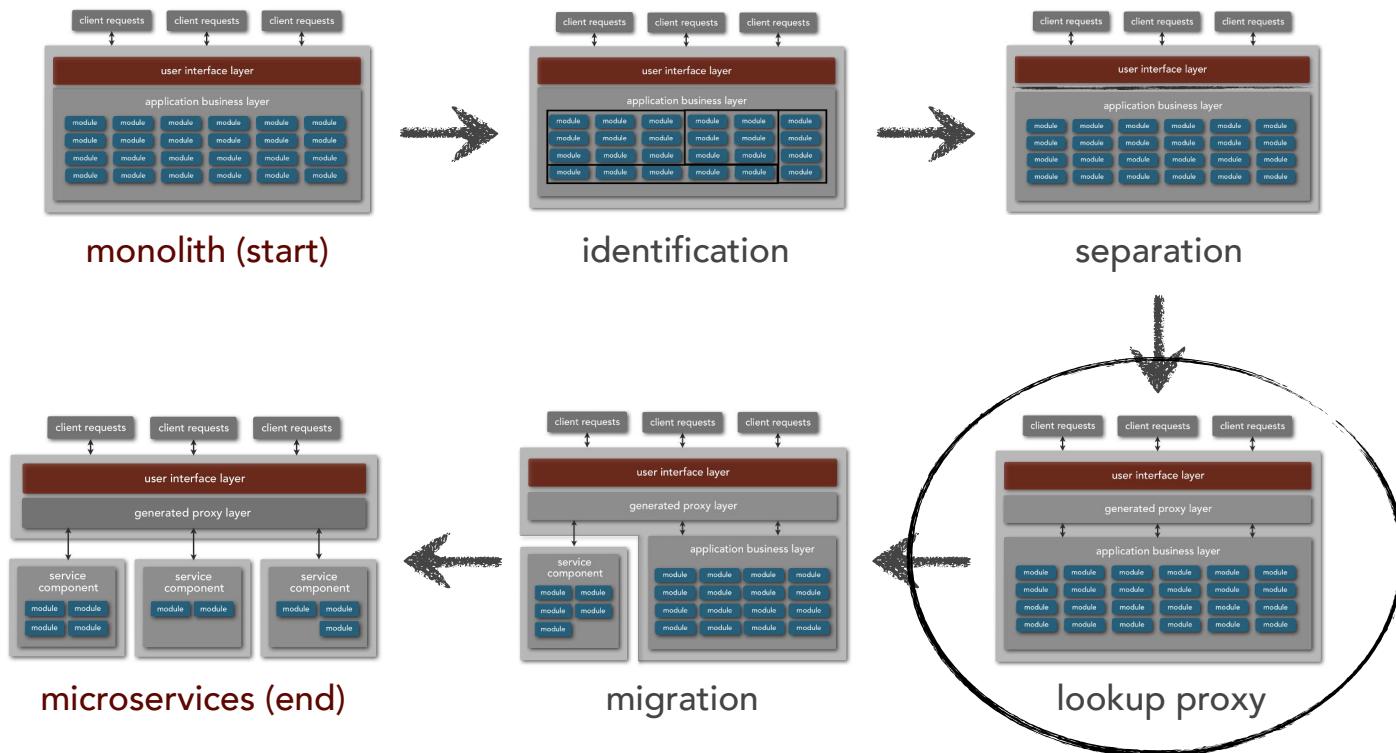
migration steps - separation



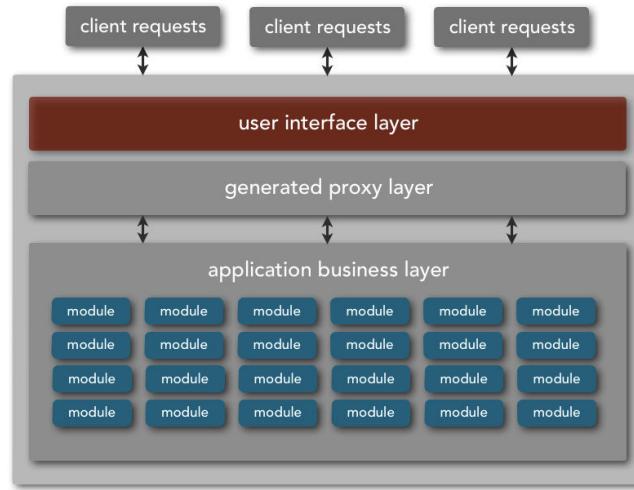
migration steps - proxy layer



migration steps - proxy layer

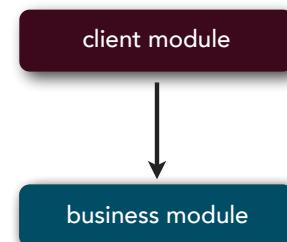


migration steps - proxy layer

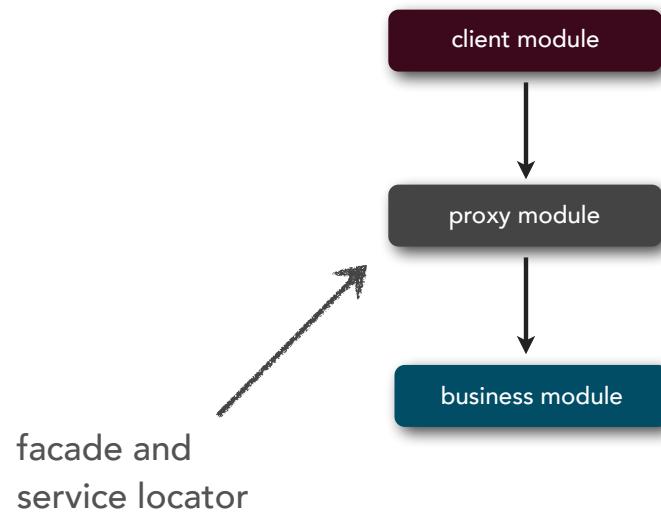


add a proxy layer that acts as both a facade and service locator to isolate changes during the migration effort

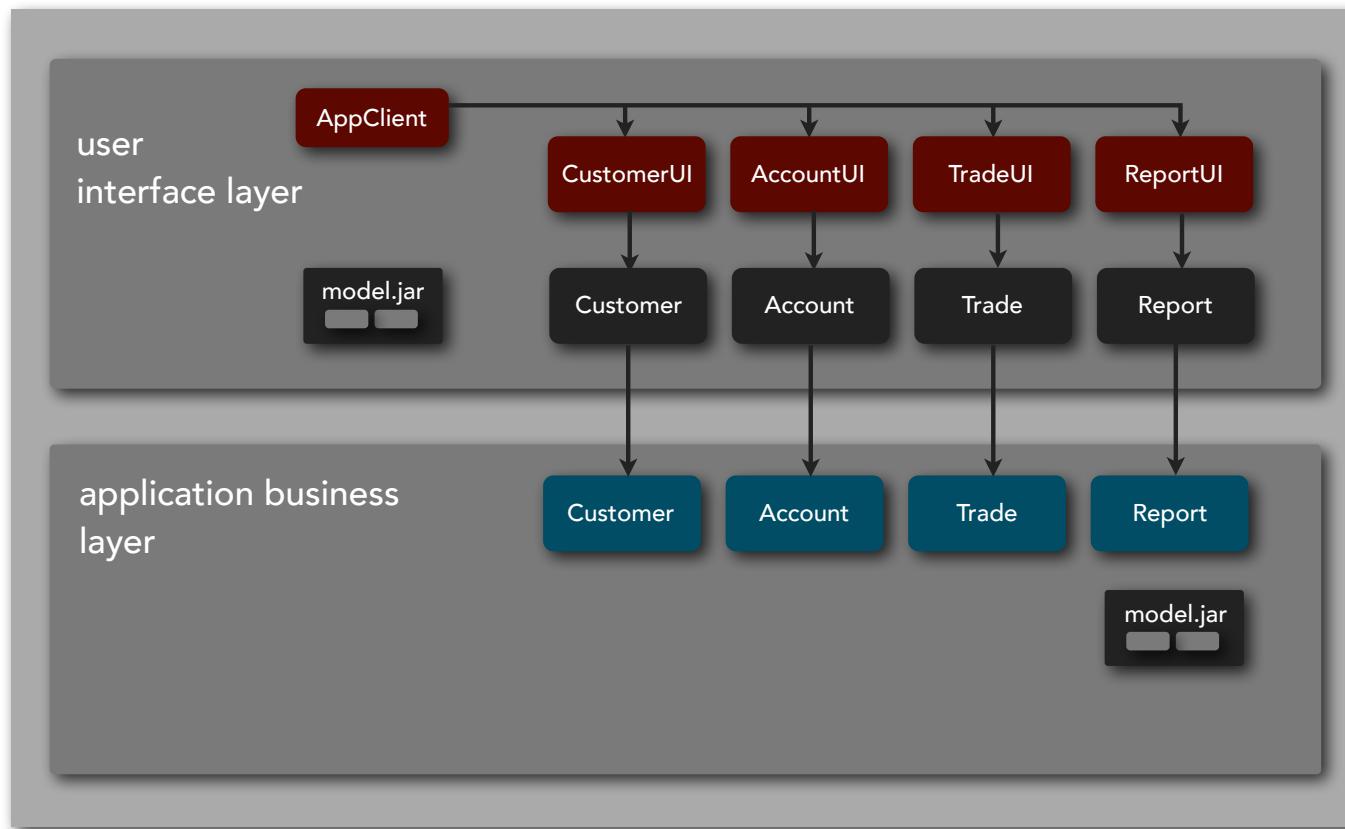
migration steps - proxy layer



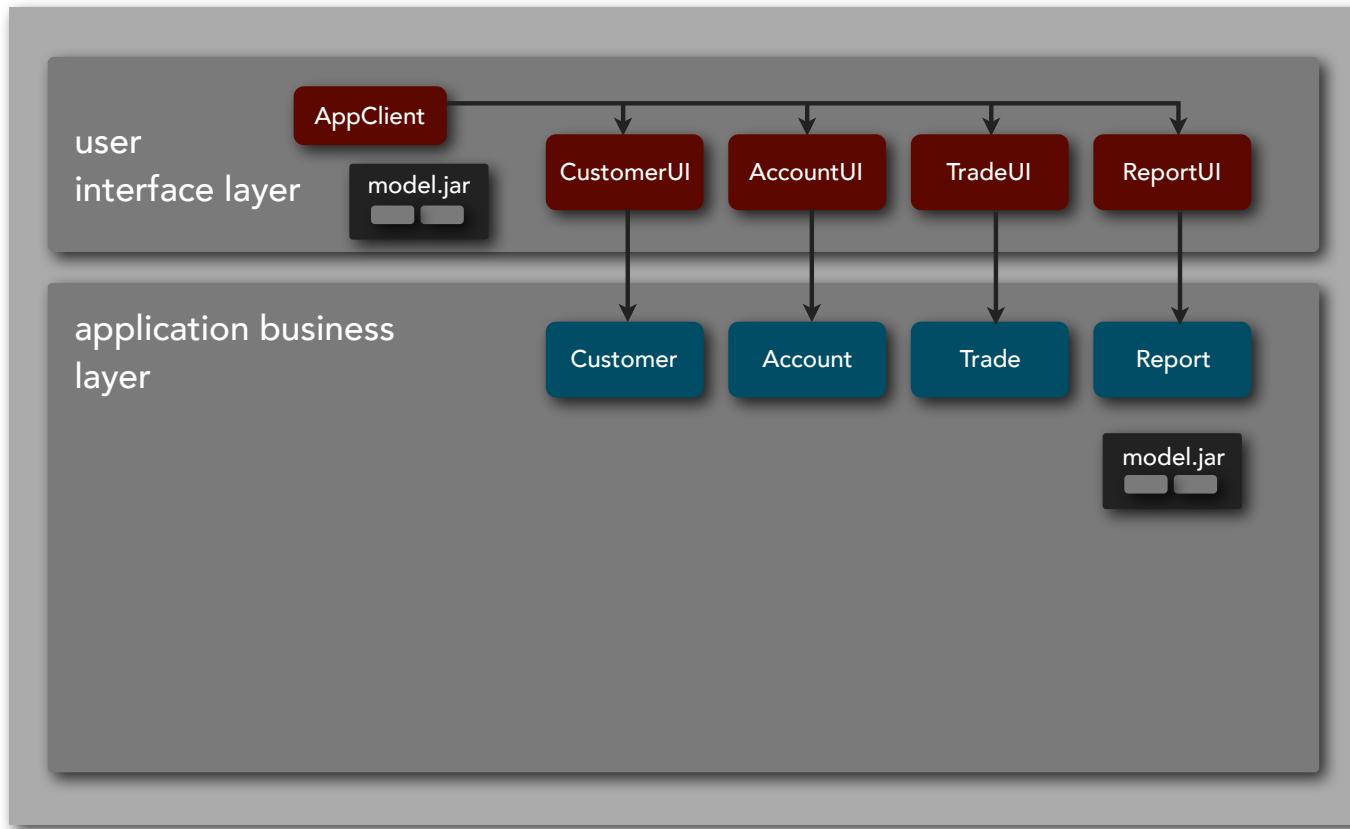
migration steps - proxy layer



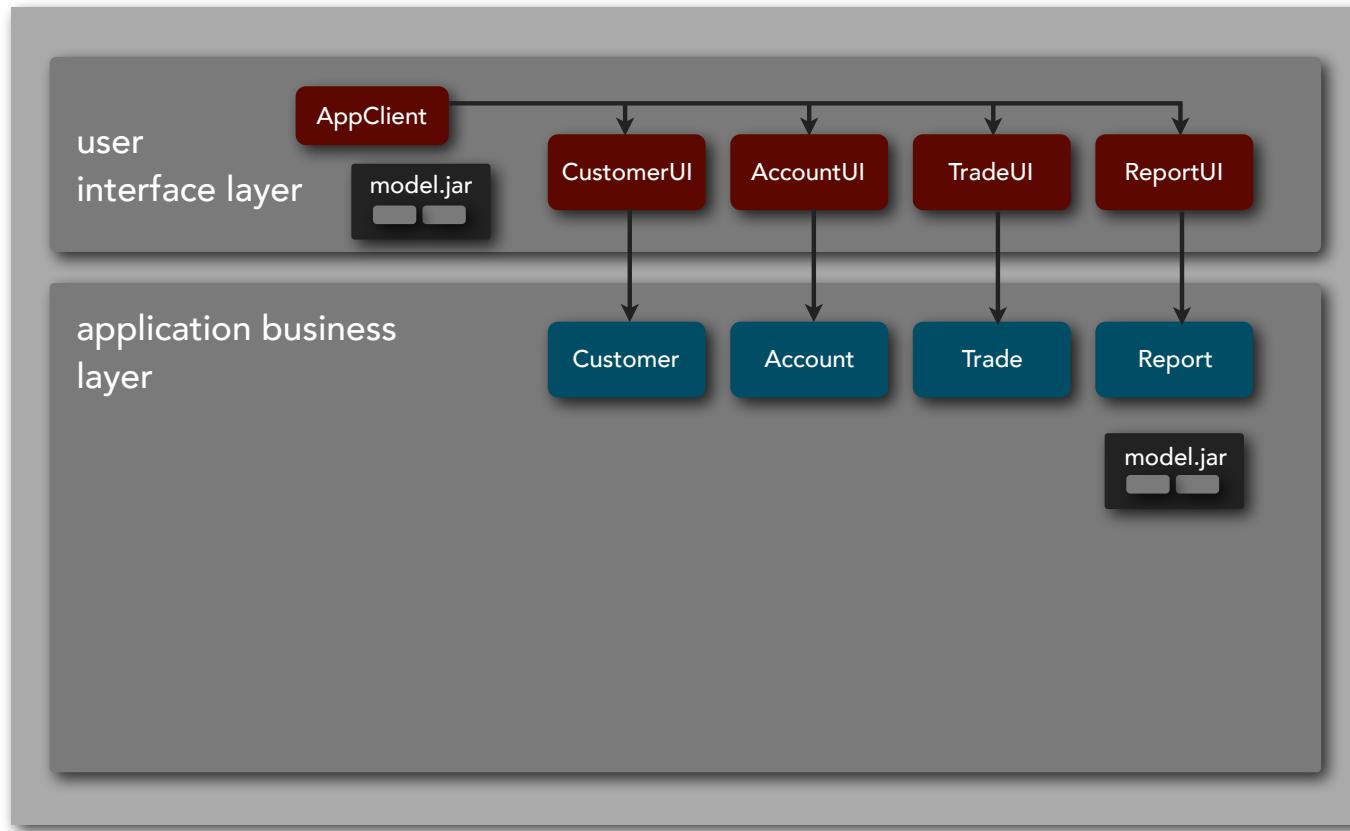
migration steps - proxy layer



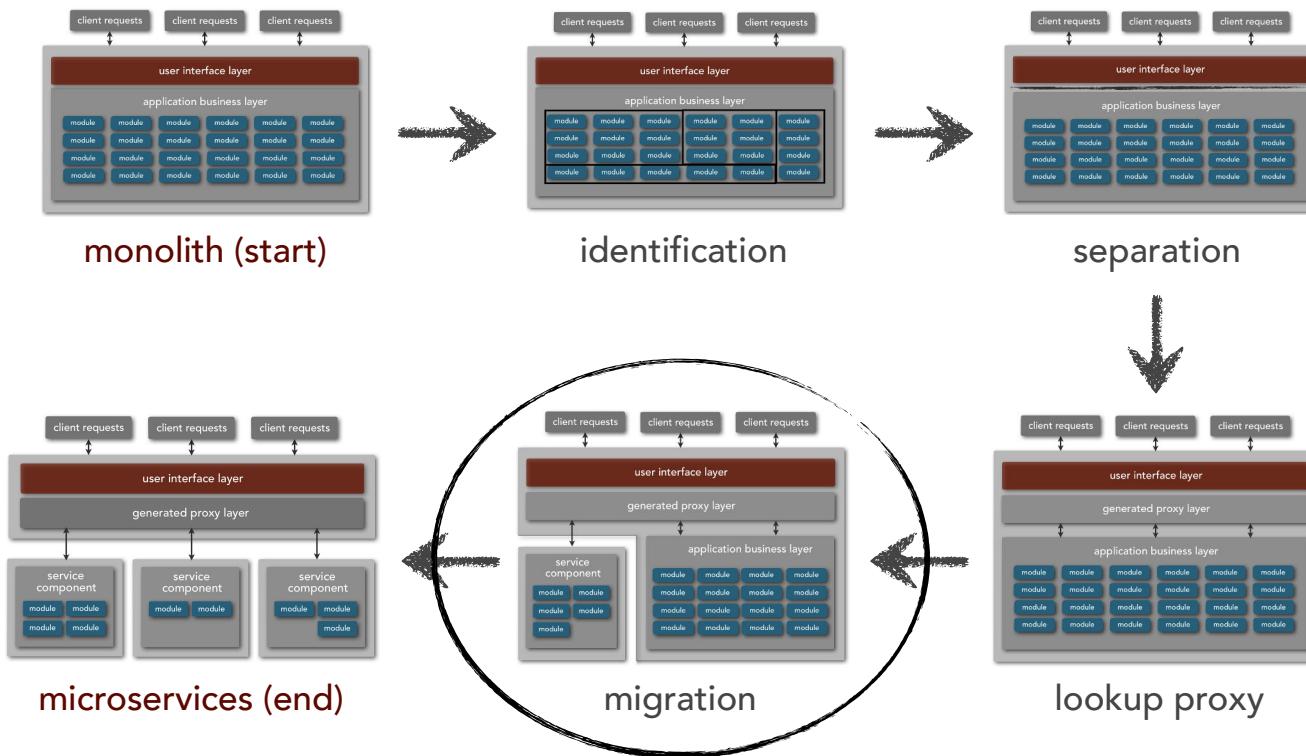
migration steps - separation



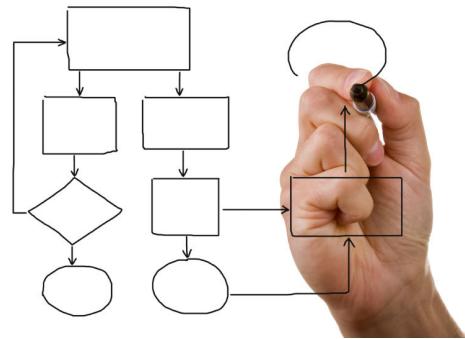
migration steps - refactoring



migration steps - refactoring

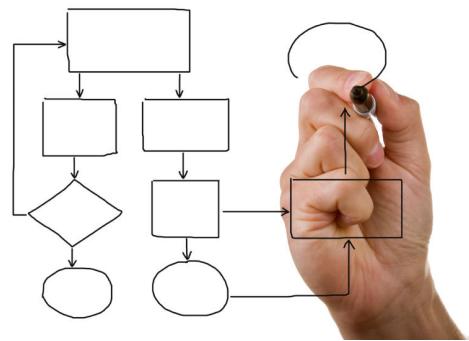


migration steps - refactoring



work in small iterations

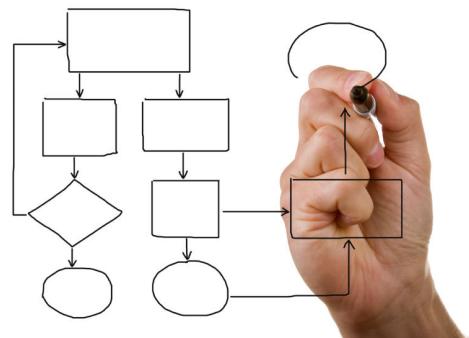
migration steps - refactoring



work in small iterations

identify the technical and business
value expected at each iteration

migration steps - refactoring

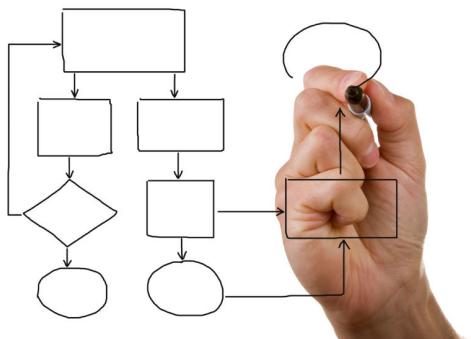


work in small iterations

identify the technical and business value expected at each iteration

use a playbook approach to outline the architecture transformations

migration steps - refactoring



work in small iterations

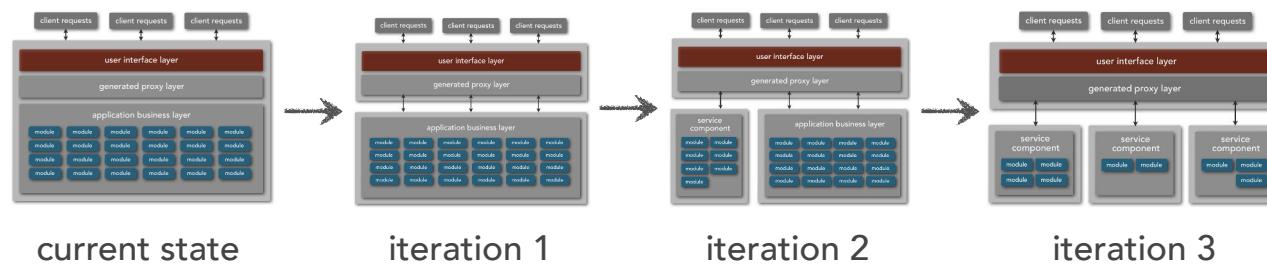
identify the technical and business value expected at each iteration

use a playbook approach to outline the architecture transformations

decide on migration vs. adaptation (or a combination of both)

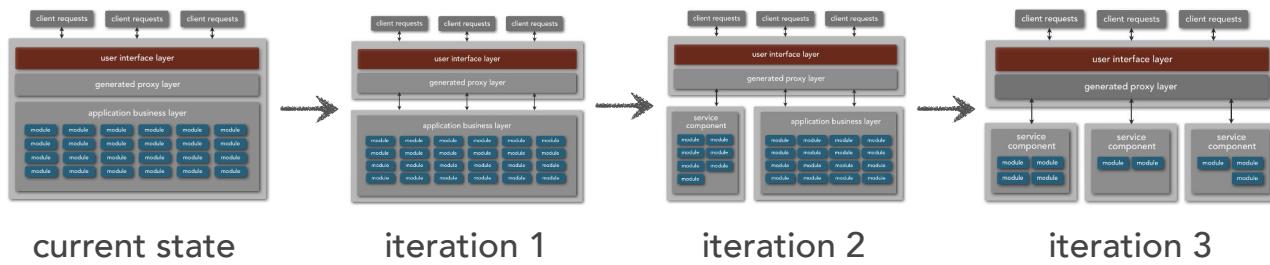
migration steps - refactoring playbook approach

each iteration should clearly illustrate the changes to the architecture each step along the way



migration steps - refactoring

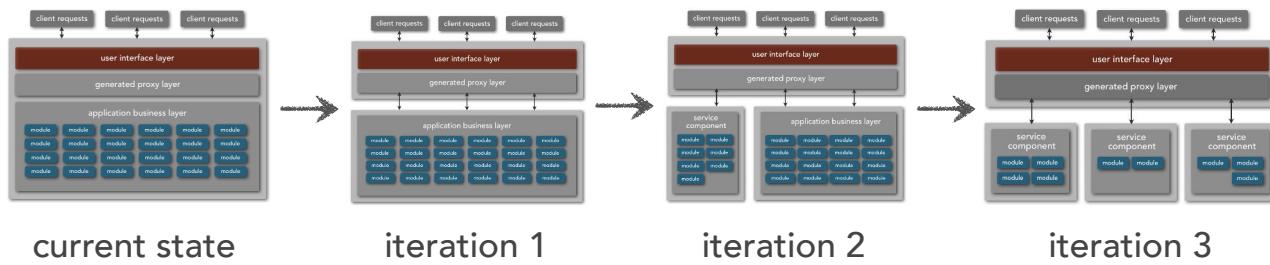
playbook approach



identify the purpose behind each iteration

migration steps - refactoring

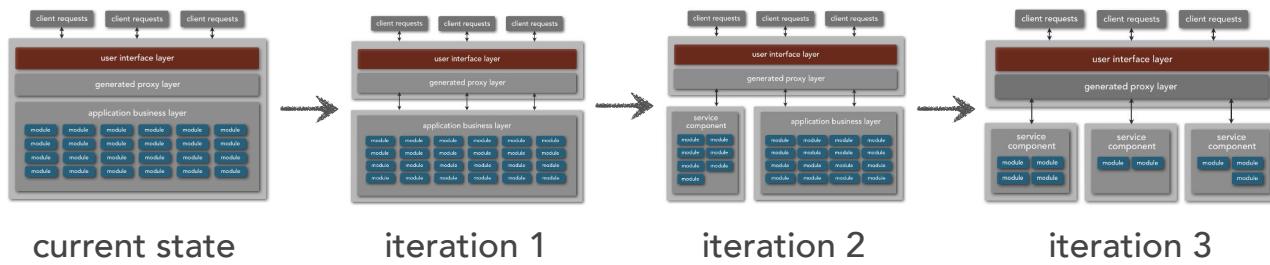
playbook approach



identify the purpose behind each iteration

identify the technical and business value for each iteration

migration steps - refactoring playbook approach

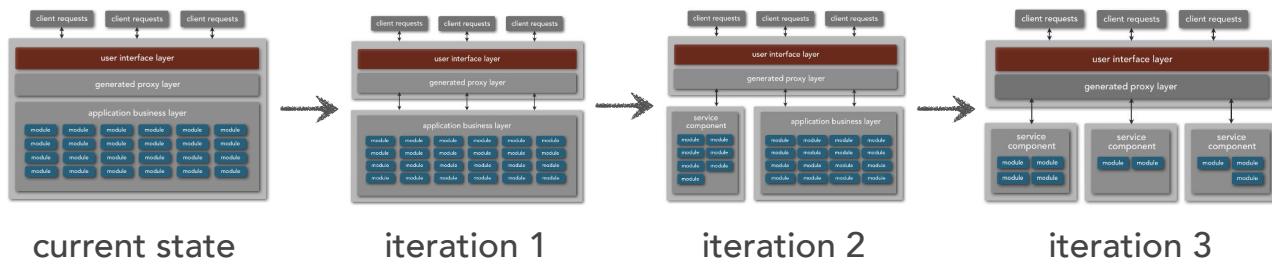


identify the purpose behind each iteration

identify the technical and business value for each iteration

try to minimize "staging iterations"

migration steps - refactoring playbook approach



identify the purpose behind each iteration

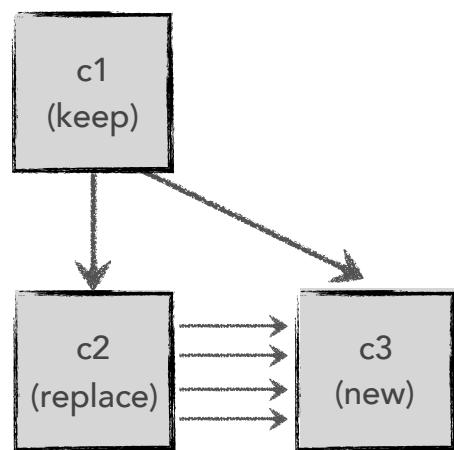
identify the technical and business value for each iteration

try to minimize "staging iterations"

keep iterations as small as possible while still providing enough technical and business value

migration steps - refactoring

migration vs. adaptation

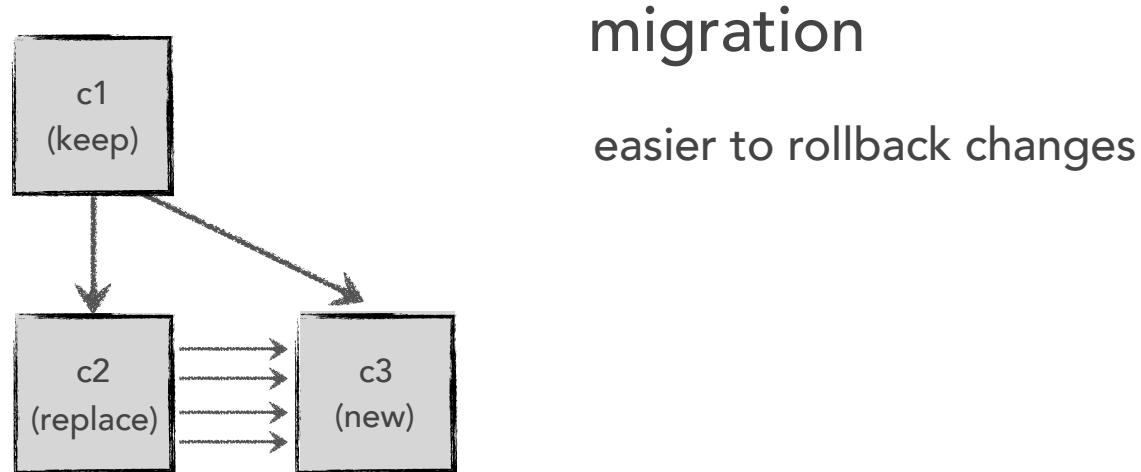


migration

*the replacement of old components
with new ones through migration
over time*

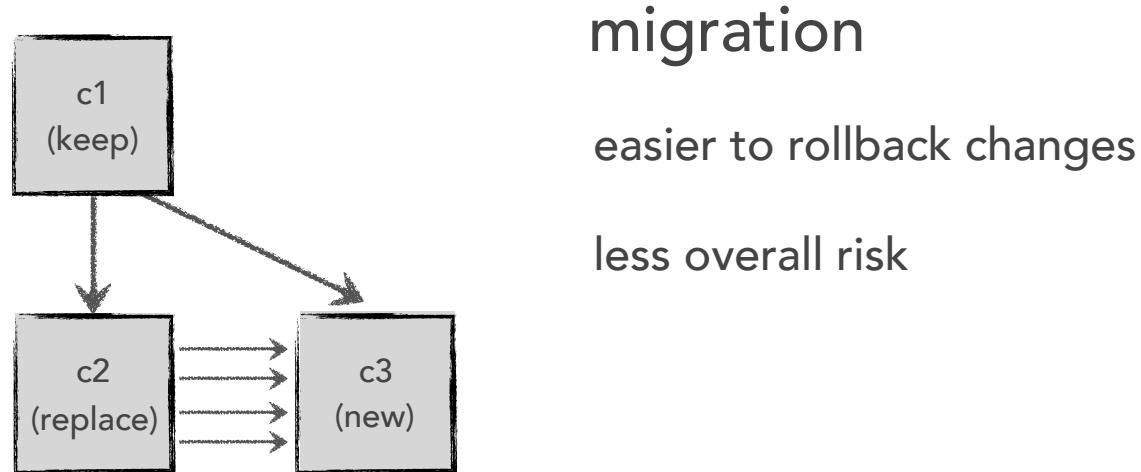
migration steps - refactoring

migration vs. adaptation



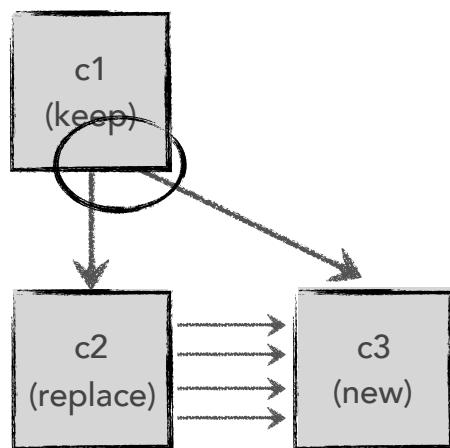
migration steps - refactoring

migration vs. adaptation



migration steps - refactoring

migration vs. adaptation



migration

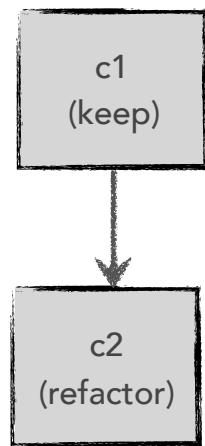
easier to rollback changes

less overall risk

requires switching logic in calling components

migration steps - refactoring

migration vs. adaptation

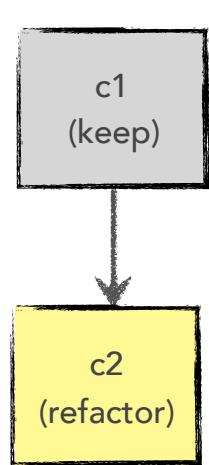


adaptation

*refactoring of existing components
into new functionality*

migration steps - refactoring

migration vs. adaptation

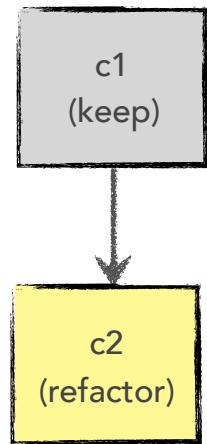


adaptation

refactor vs. replacement

migration steps - refactoring

migration vs. adaptation



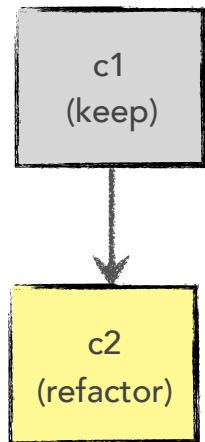
adaptation

refactor vs. replacement

harder to rollback changes

migration steps - refactoring

migration vs. adaptation



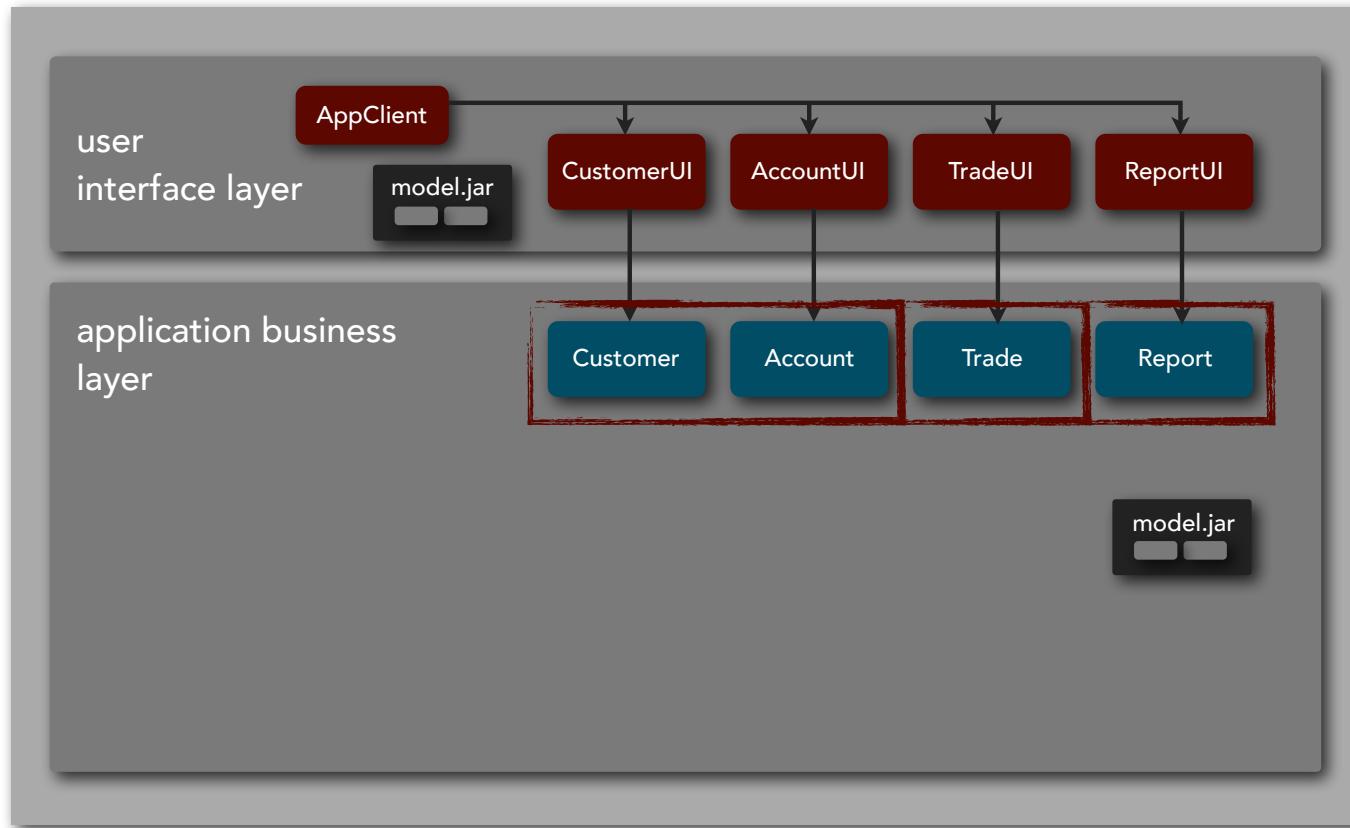
adaptation

refactor vs. replacement

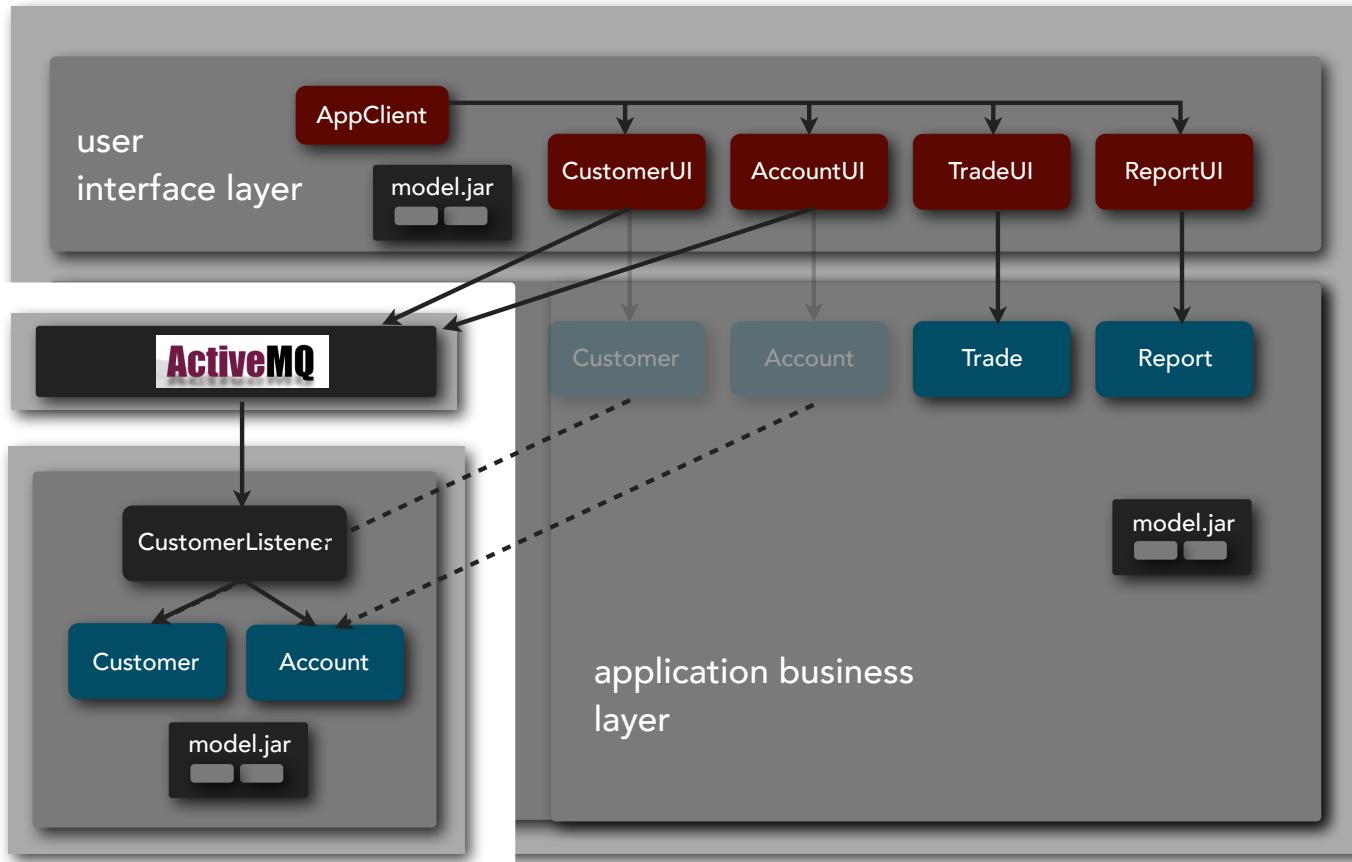
harder to rollback changes

no changes to calling components

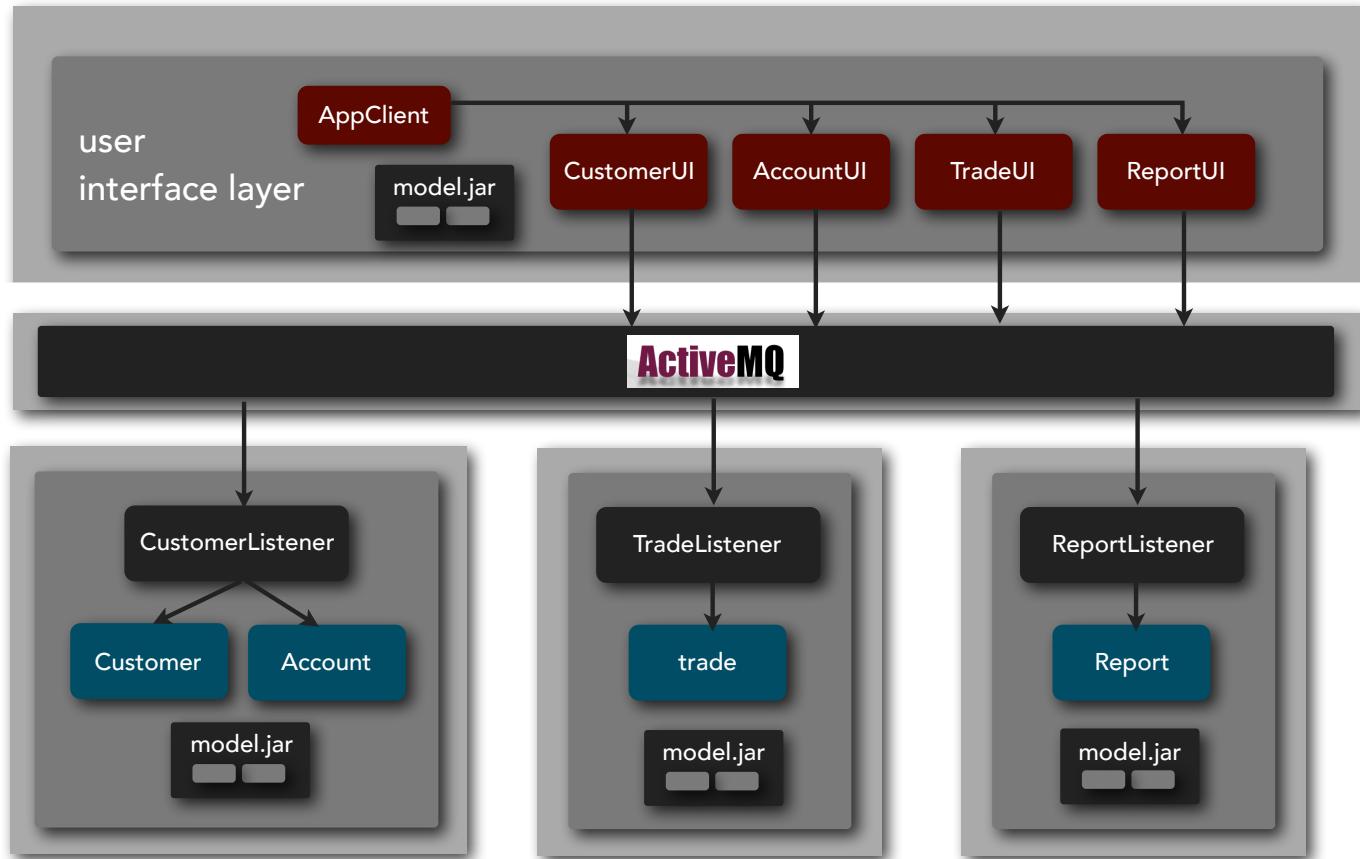
migration steps - refactoring



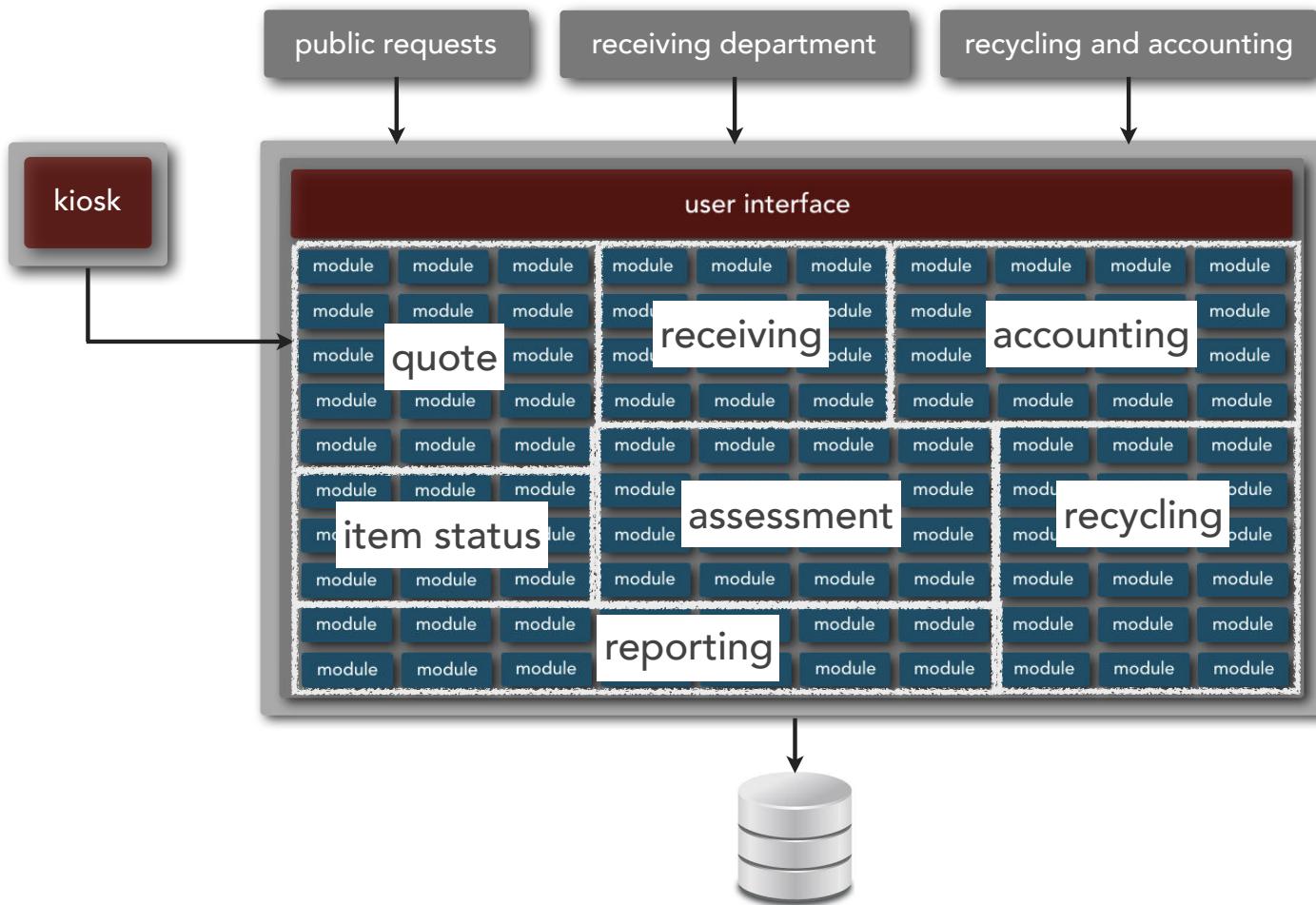
migration steps - refactoring



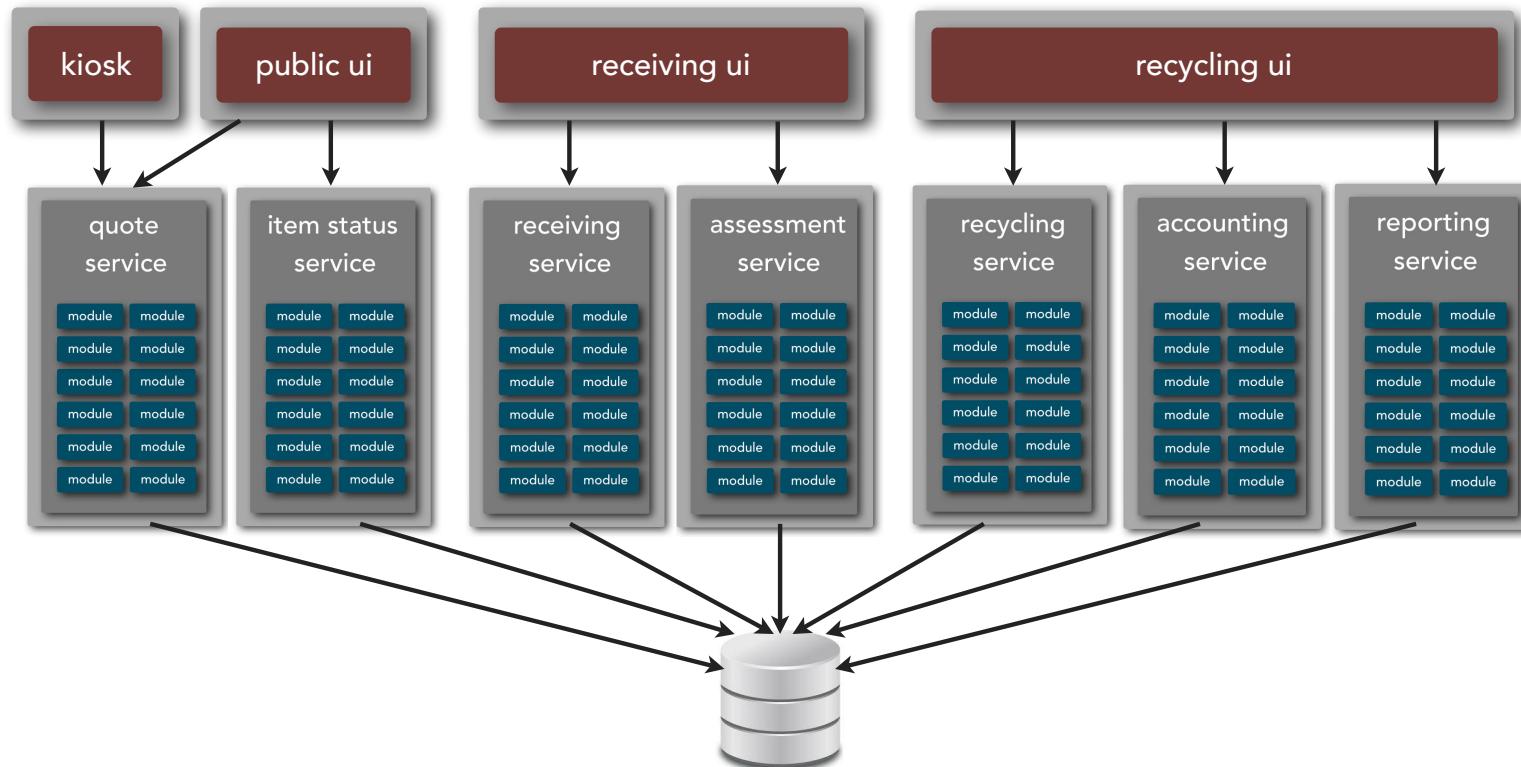
migration steps - refactoring



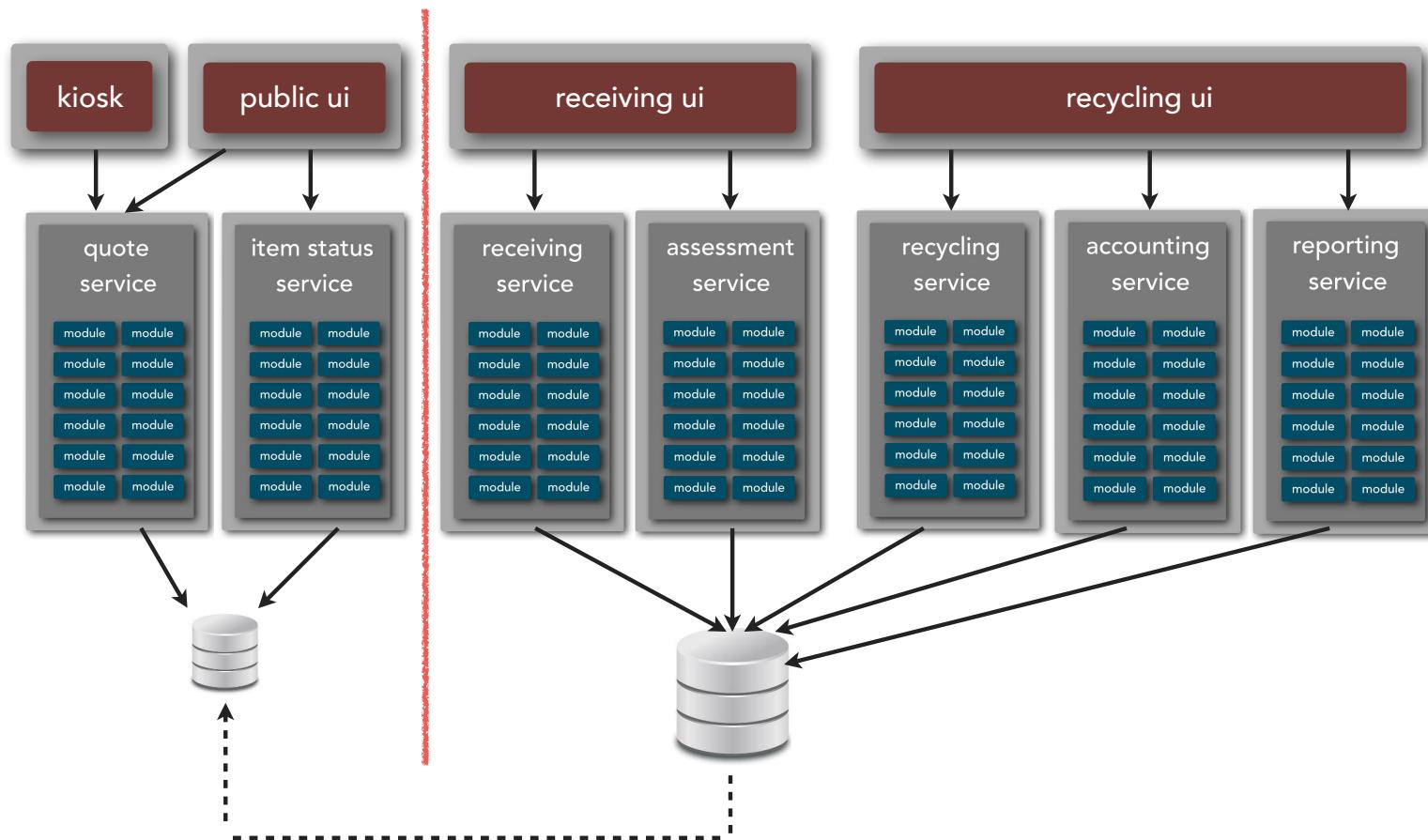
electronics recycling application



electronics recycling application

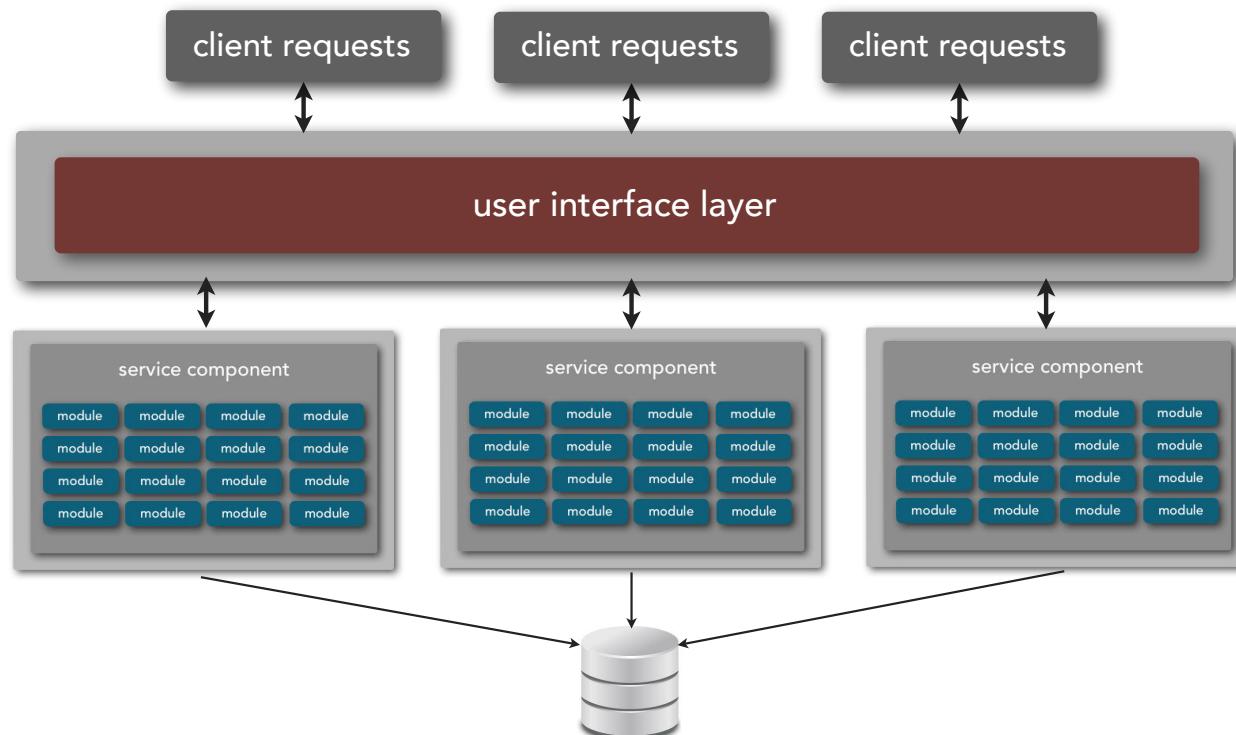


electronics recycling application



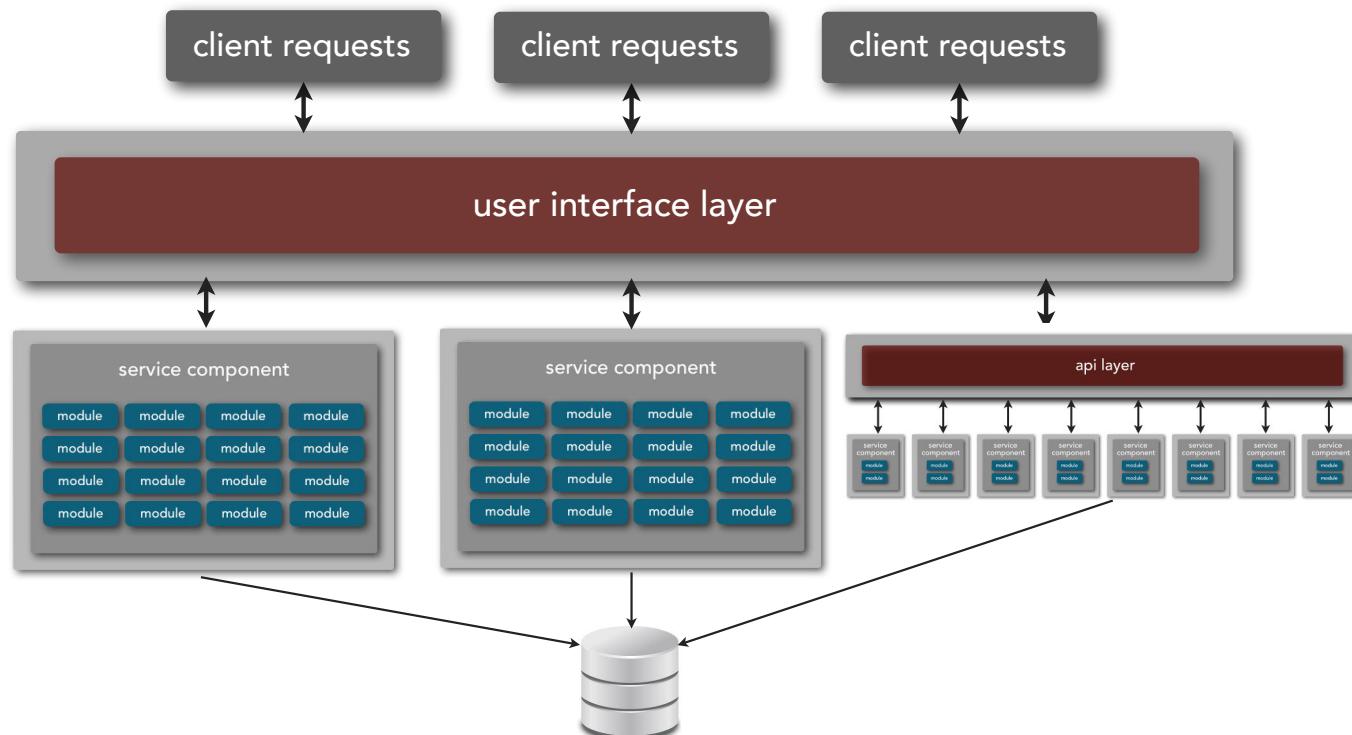
service-based architecture

adding microservices



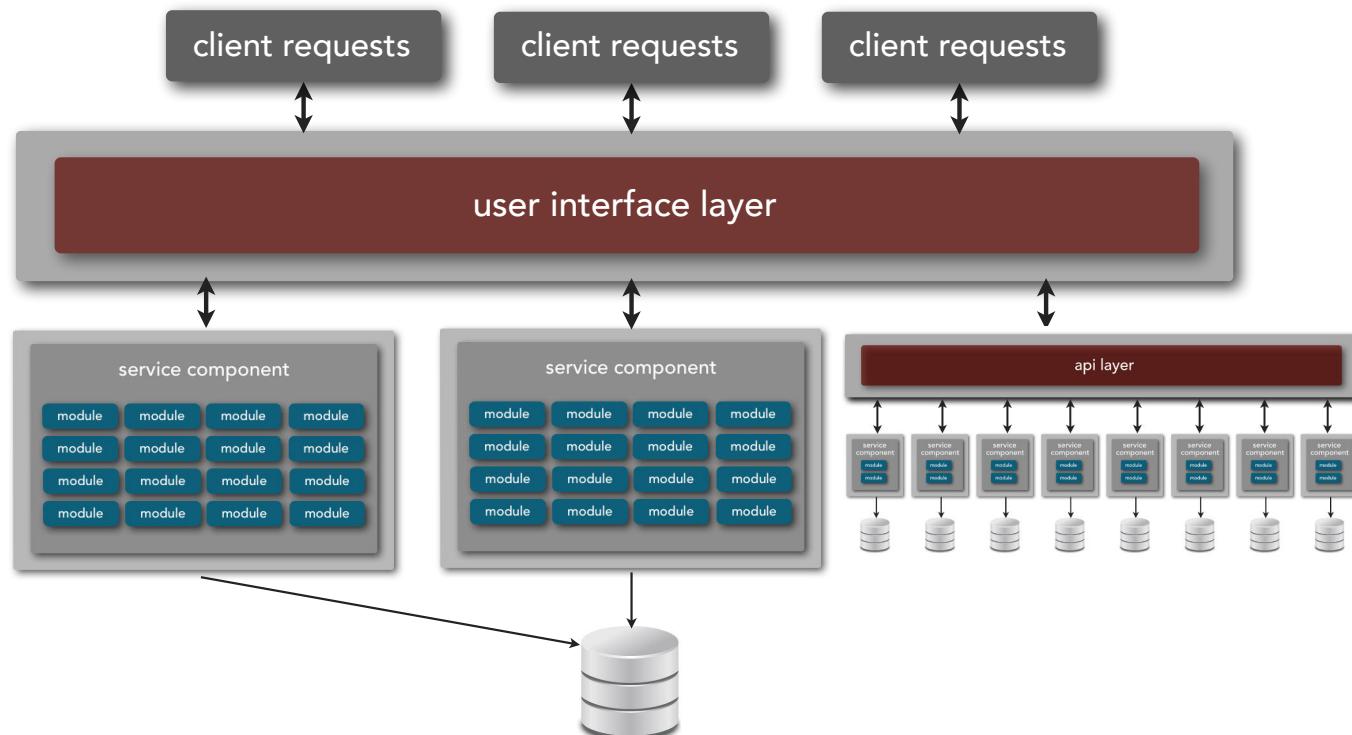
service-based architecture

adding microservices

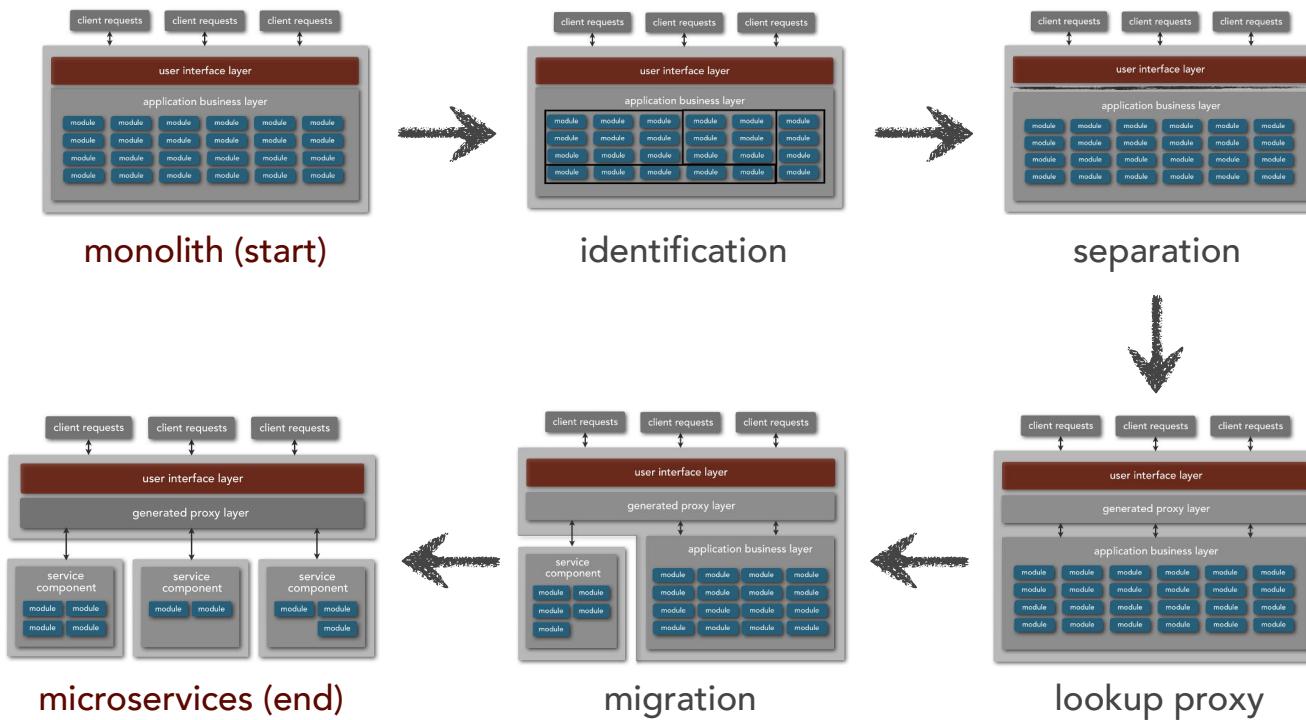


service-based architecture

adding microservices



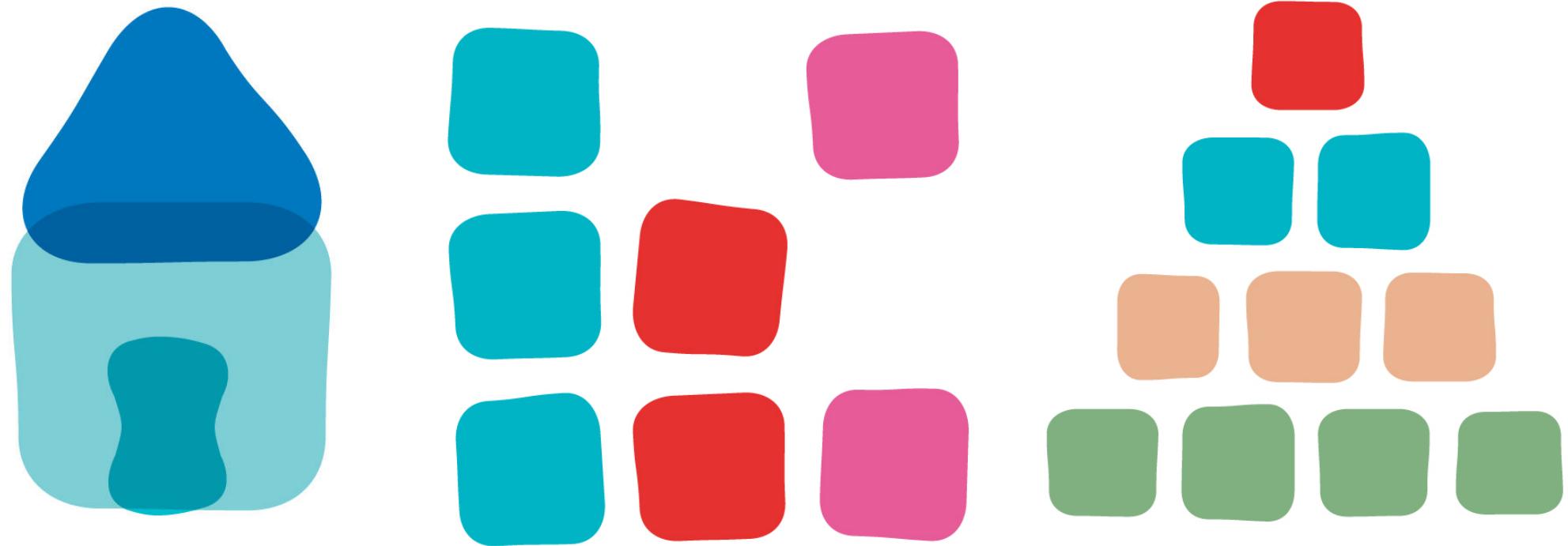
migration steps - refactoring



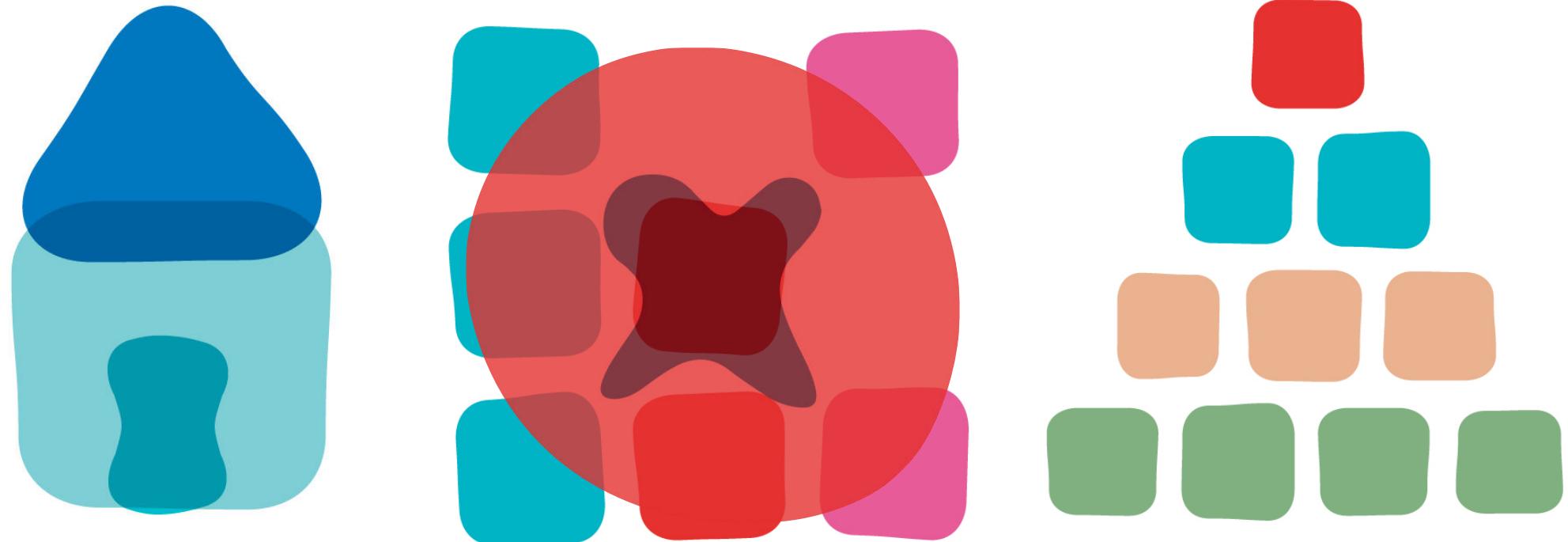
Domain/Architecture Isomorphism



Domain/Architecture Isomorphism



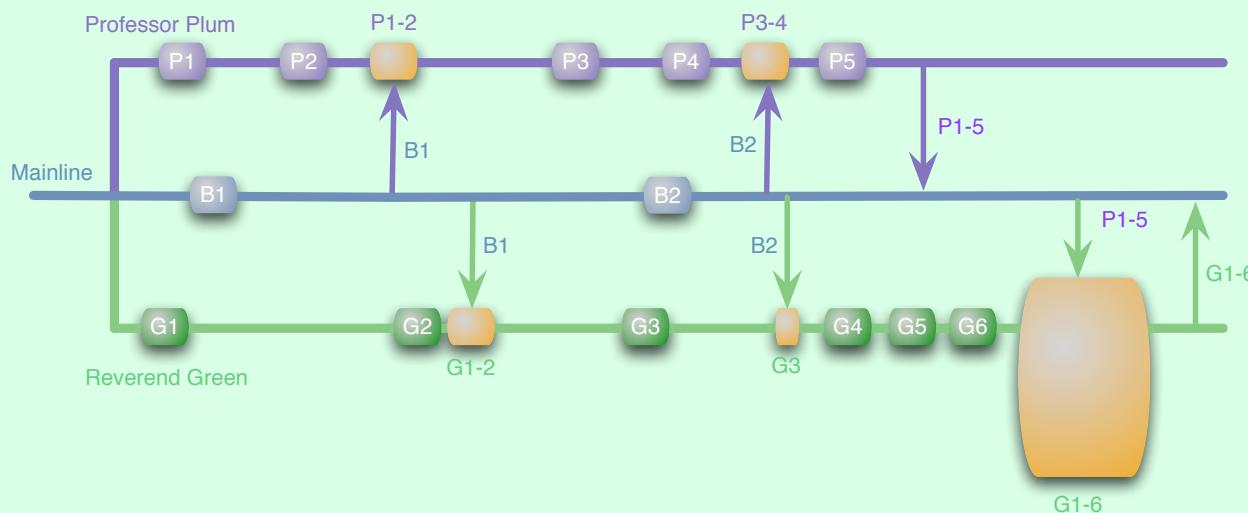
Domain/Architecture Isomorphism

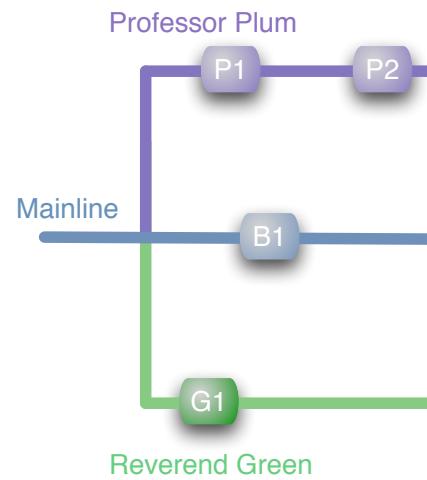


aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



trunk-based development versus feature branching

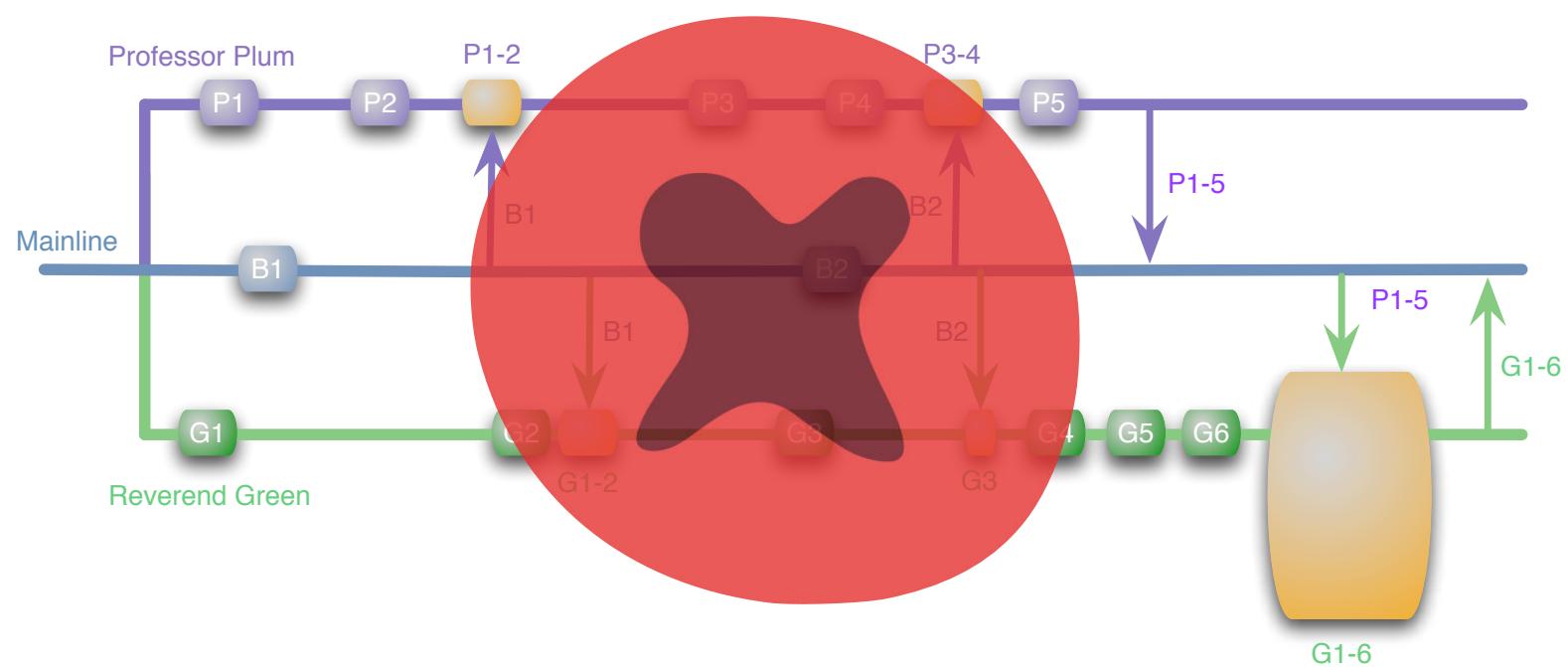




***merge
ambush!***
Feature Branching



trunk-based development



Trunk Based Development: Introduction

The screenshot shows the homepage of the Trunk Based Development website. The header reads "Trunk Based Development: Introduction". On the left, there's a sidebar with a navigation menu. The main content area has a large blue arrow pointing right labeled "The trunk". Above the arrow, it says "A shared branch" and "The whole team develops here ...". Below the arrow, it says "... not here". The sidebar includes sections like "Introduction", "One line summary", "Caveats", "History", "This site", "Context", "Five minute overview", "Deciding factors", "VCS features", "VCS choices", "Feature flags", "Branch by Abstraction", "Branch for release", "Release from trunk", "Continuous Integration", "Continuous Code Review", "Continuous Delivery", "Concurrent development of con...", "Strangulation", "Observed habits", "Doing it wrong", "Alternative branching models", "Monorepos", "Expanding Contracting Monoreps...", "Game Changers", "Publications", and "Site Source on GitHub". The "Introduction" section contains a diagram illustrating the trunk-based development model.



Paul Hammant

trunk-based development

<https://trunkbaseddevelopment.com>

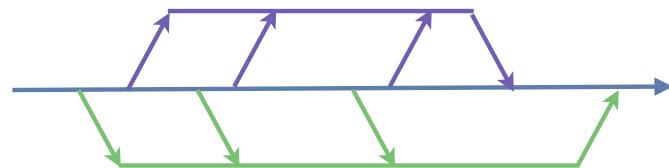


trunk-based development



<https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext>

Feature Branching



Big Scary Merge

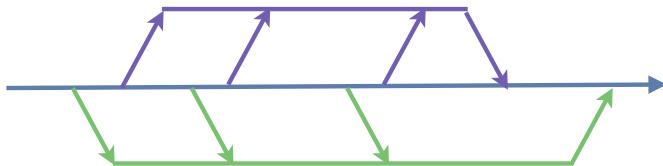
Cherry Picking



*Untrusted
Contributors*

Continuous Integration

Feature Branching



Big Scary Merge

Discouraging refactoring

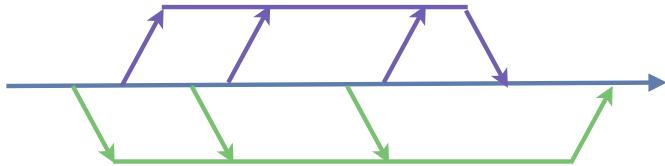
Cherry Picking



*Untrusted
Contributors*

Continuous Integration

Feature Branching



- 1 !
- 2 !
- 3 !

Big Scary Merge

Discouraging refactoring

Hard to combine features

Cherry Picking



*Untrusted
Contributors*

Continuous Integration



<http://martinfowler.com/articles/feature-toggles.html>

The screenshot shows a Mac OS X browser window displaying the Martin Fowler website at martinfowler.com. The page title is "Feature Toggles". The main content discusses Feature Toggles as a powerful technique for modifying system behavior without changing code. It features a bio of Pete Hodgson, a photo of him, and a sidebar with a "Contents" section and an "expand" button. Below the main content, there's a summary of what Feature Toggling is and its benefits.

Feature Toggles

Feature toggles are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

08 February 2016

 **Pete Hodgson**

Pete Hodgson is a consultant at ThoughtWorks, where he's spent the last few years helping teams become awesome at sustainable delivery of high-quality software. He's particularly passionate about web-based mobile, ruby, agile, functional approaches to software, and beer.

Find [similar articles](#) to this by looking at these tags: [delivery](#) · [application architecture](#) · [continuous integration](#)

"Feature Toggling" is a set of patterns which can help a team to deliver new functionality to users rapidly but safely. In this article on Feature Toggling we'll start off with a short story showing some typical scenarios where Feature Toggles are helpful. Then we'll dig into the details, covering specific patterns and practices which will help a team succeed with Feature Toggles.

A Toggling Tale

Picture the scene. You're on one of several teams working on a sophisticated town planning simulation game. Your team is responsible for the core simulation engine. You have been tasked with increasing the efficiency of the Spline Reticulation algorithm. You know this will require a fairly large overhaul of the implementation which will take several weeks. Meanwhile other members of your team will need to

before

```
function reticulateSplines(){
    // current implementation lives here
}
```

these examples all use JavaScript ES2015

after

```
function reticulateSplines(){
    var useNewAlgorithm = false;
    // useNewAlgorithm = true; // UNCOMMENT IF YOU ARE WORKING ON THE NEW SR ALGORITHM

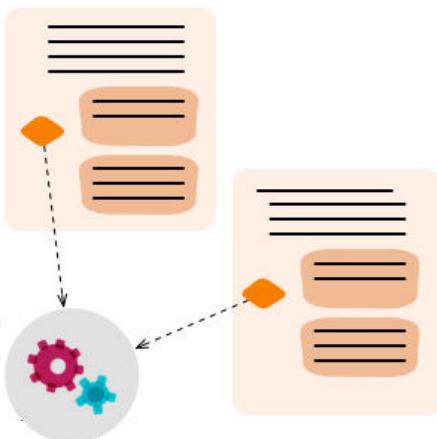
    if( useNewAlgorithm ){
        return enhancedSplineReticulation();
    }else{
        return oldFashionedSplineReticulation();
    }
}

function oldFashionedSplineReticulation(){
    // current implementation lives here
}

function enhancedSplineReticulation(){
    // TODO: implement better SR algorithm
}
```

```
function reticulateSplines(){
  if( featureEnabled("use-new-SR-algorithm") ){
    return enhancedSplineReticulation();
  }else{
    return oldFashionedSplineReticulation();
  }
}
```

make toggle dynamic



```
function createToggleRouter(featureConfig){
  return {
    setFeature(featureName, isEnabled){
      featureConfig[featureName] = isEnabled;
    },
    featureEnabled(featureName){
      return featureConfig[featureName];
    }
  };
}
```

note that we're using ES2015's method shorthand

```
describe( 'spline reticulation', function(){
  let toggleRouter;
  let simulationEngine;

  beforeEach(function(){
    toggleRouter = createToggleRouter();
    simulationEngine = createSimulationEngine({toggleRouter:toggleRouter});
  });

  it('works correctly with old algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",false);

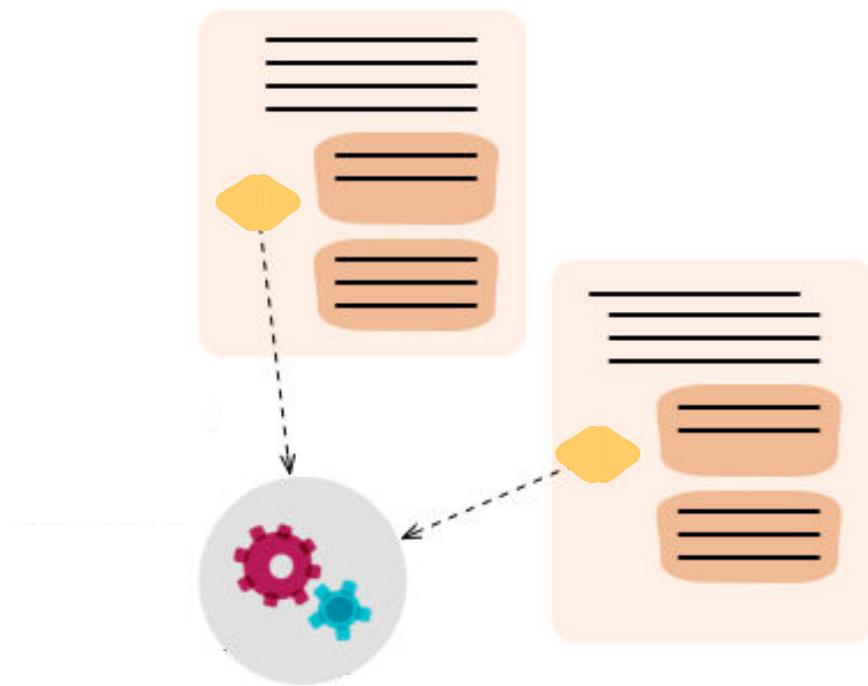
    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

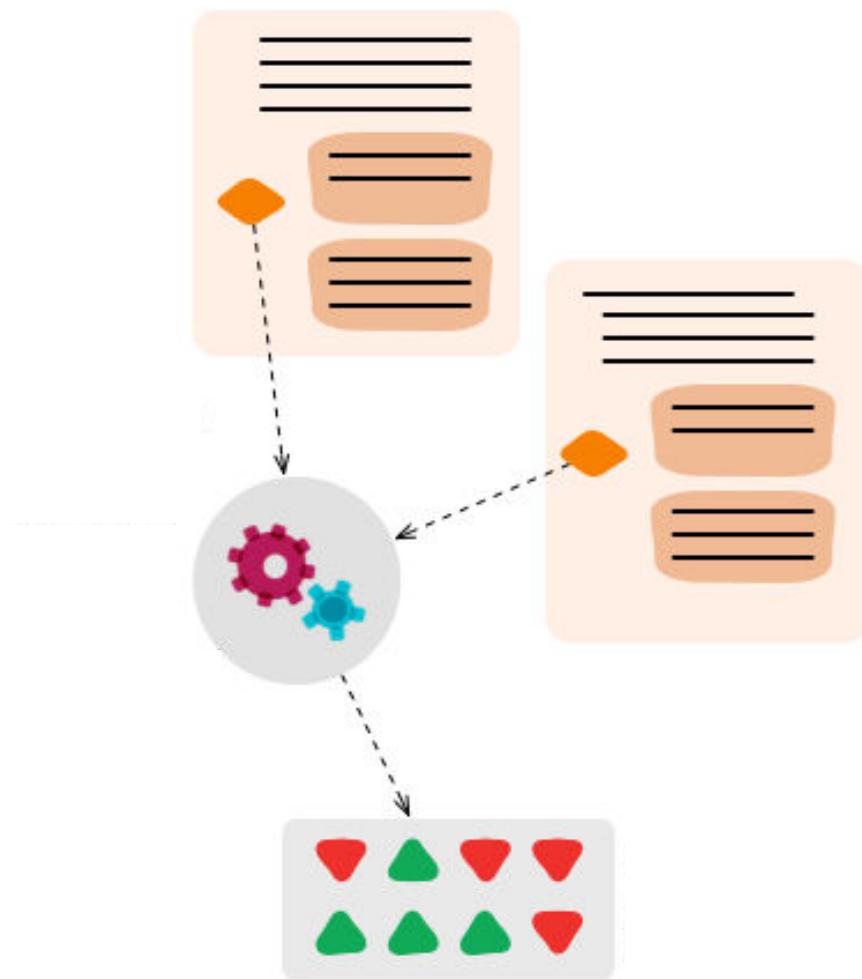
    // Then
    verifySplineReticulation(result);
  });

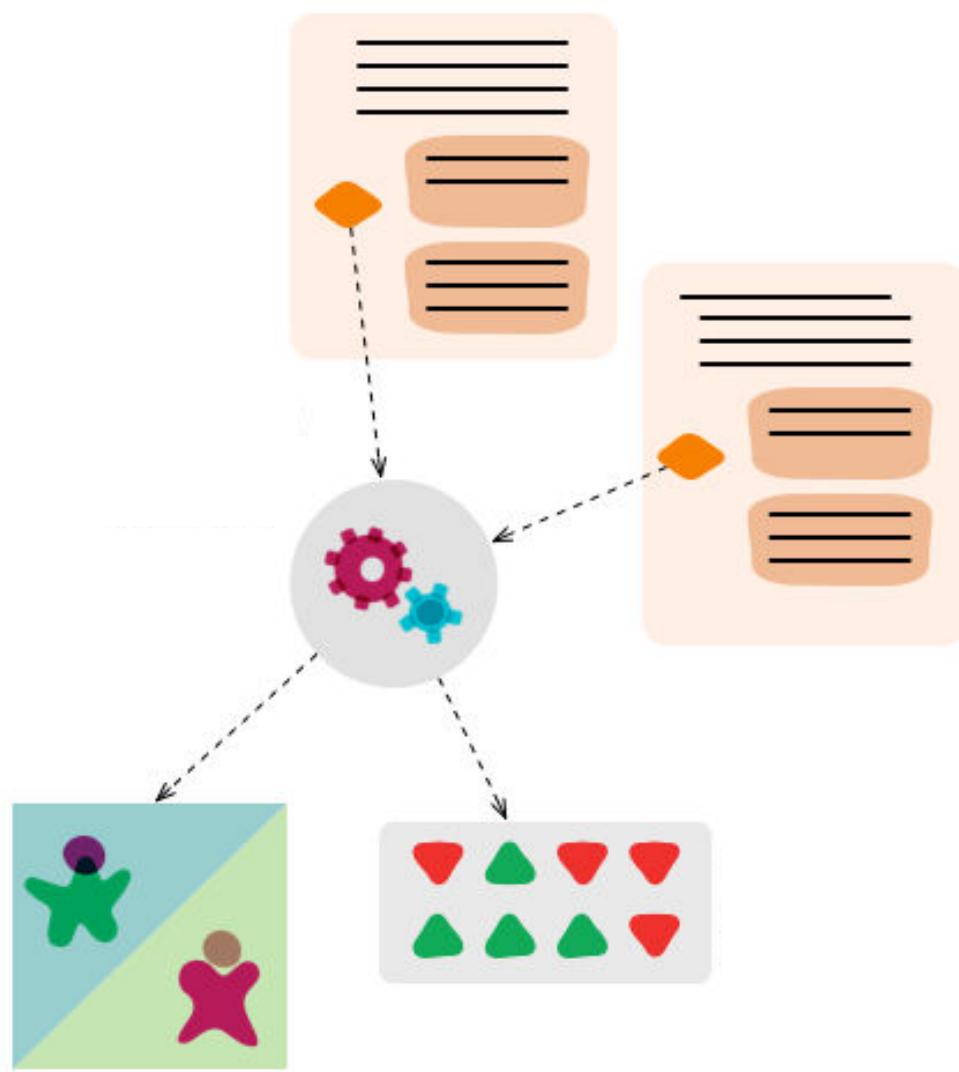
  it('works correctly with new algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",true);

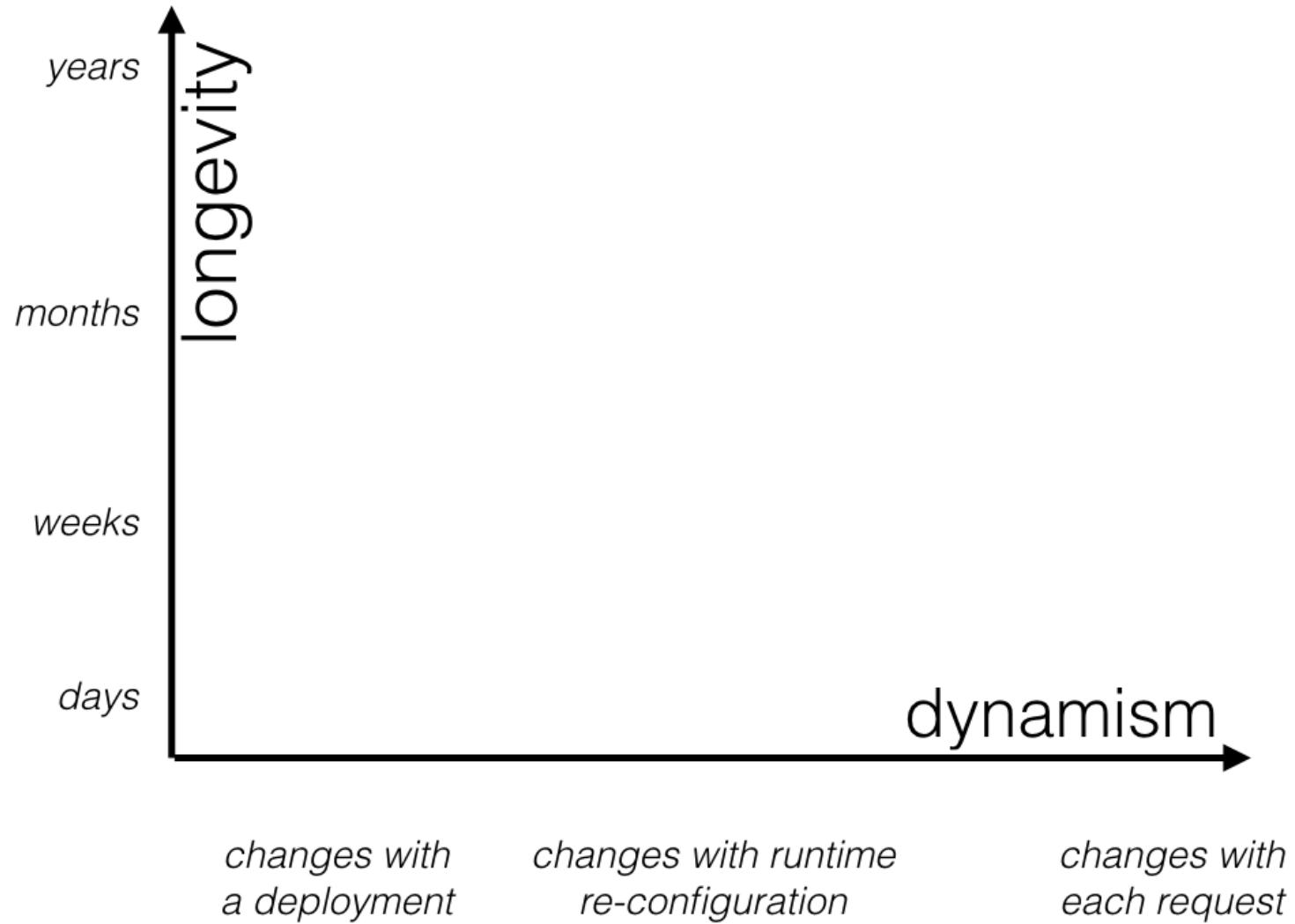
    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

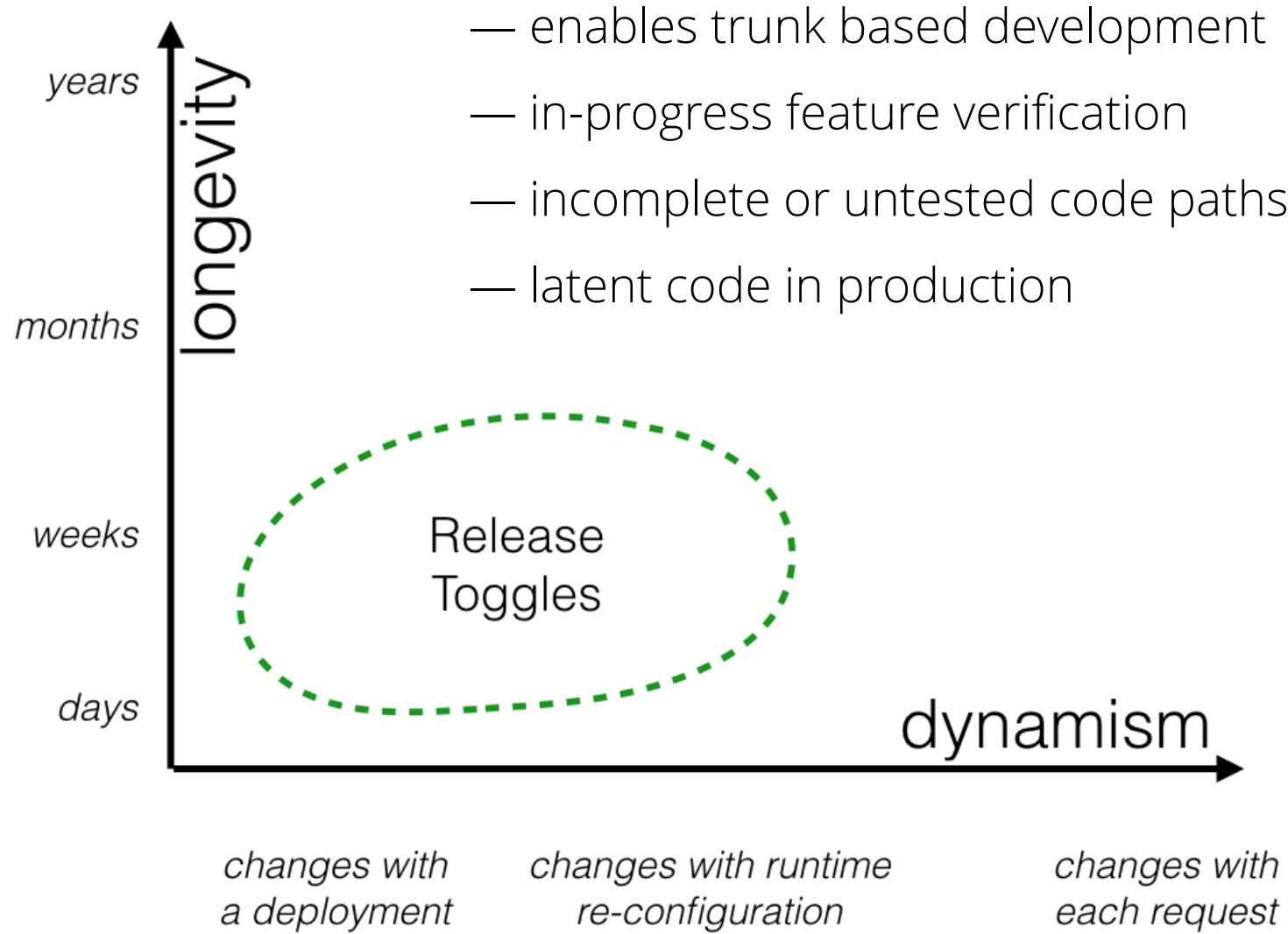
    // Then
    verifySplineReticulation(result);
  });
});
```

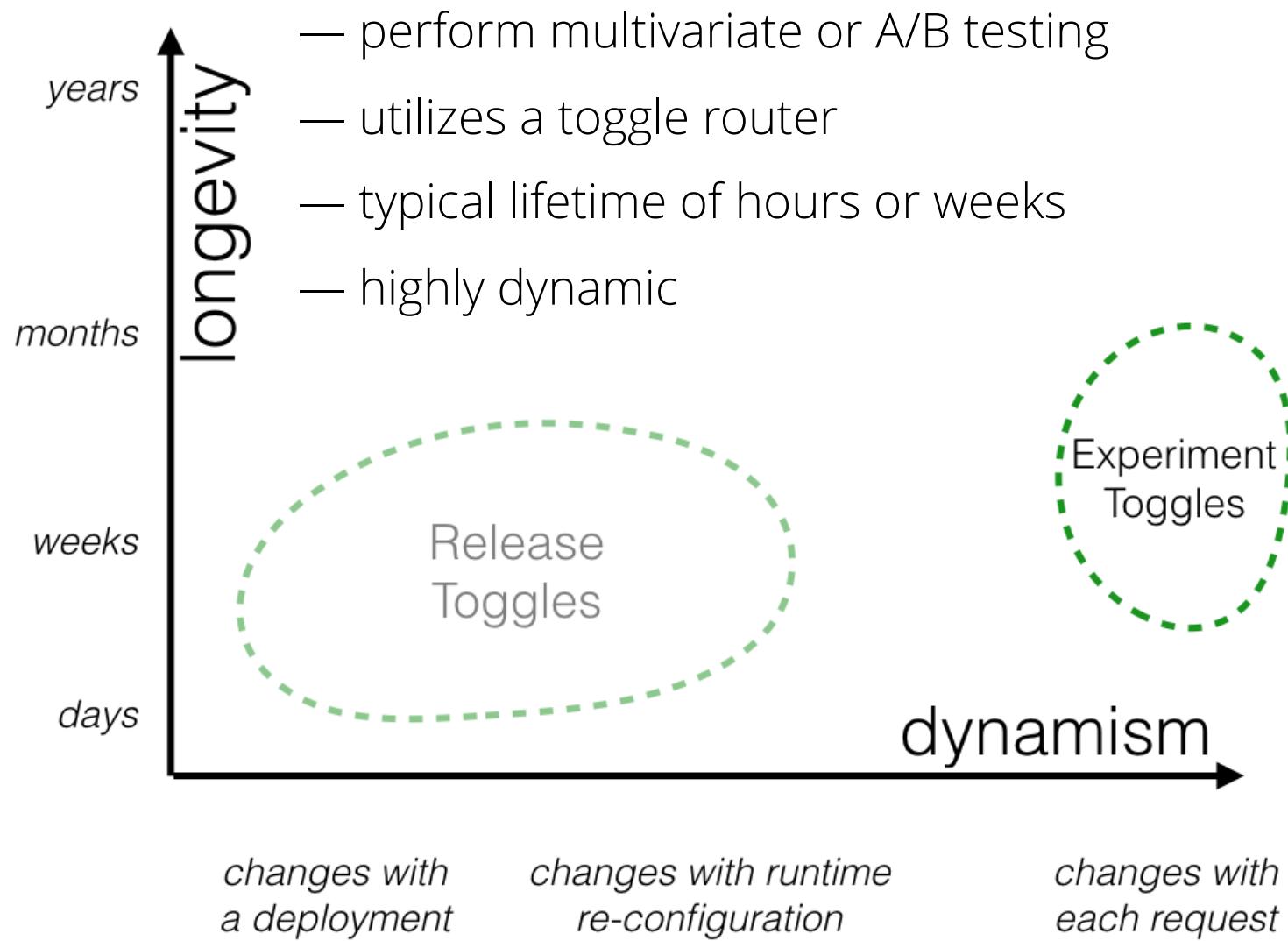


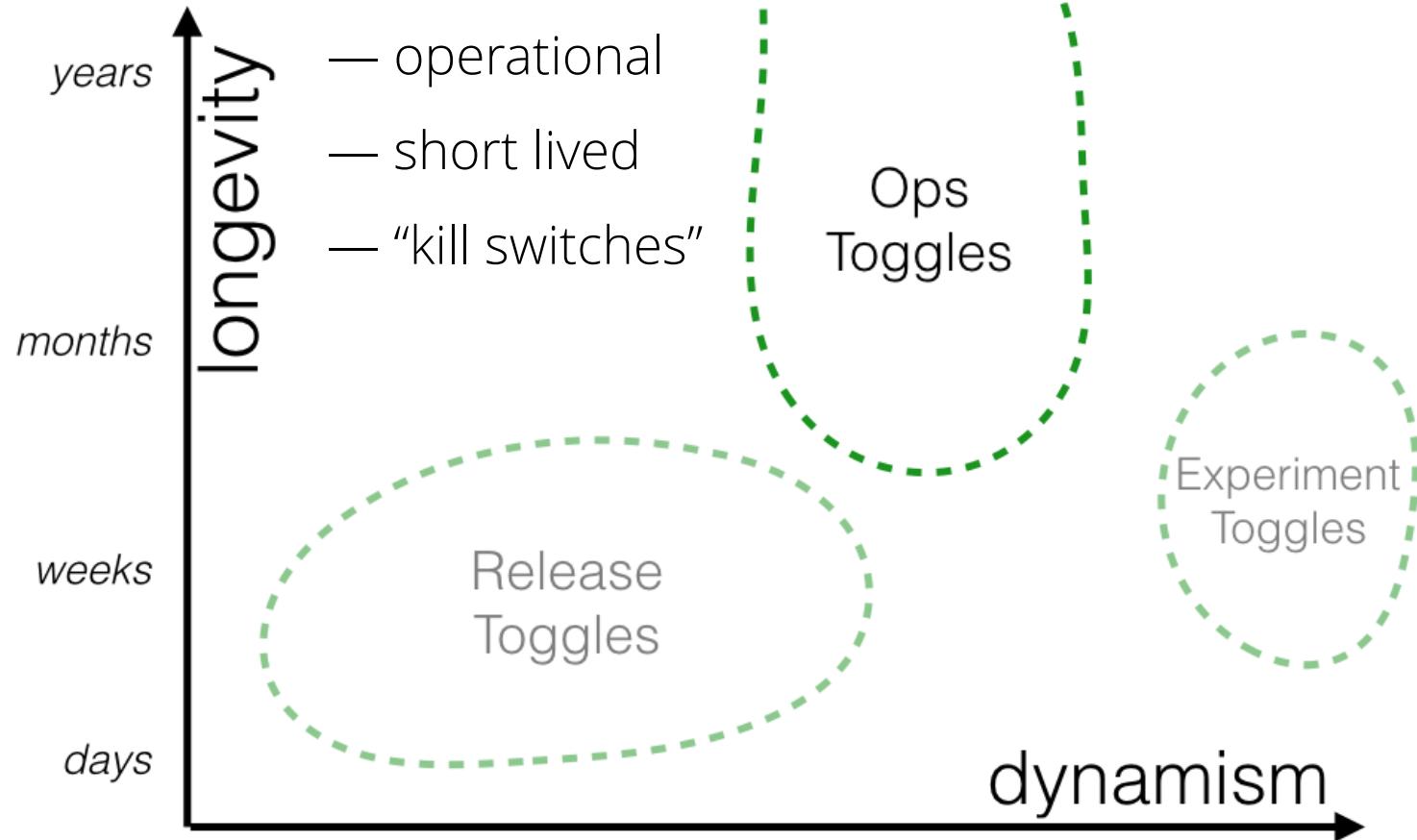








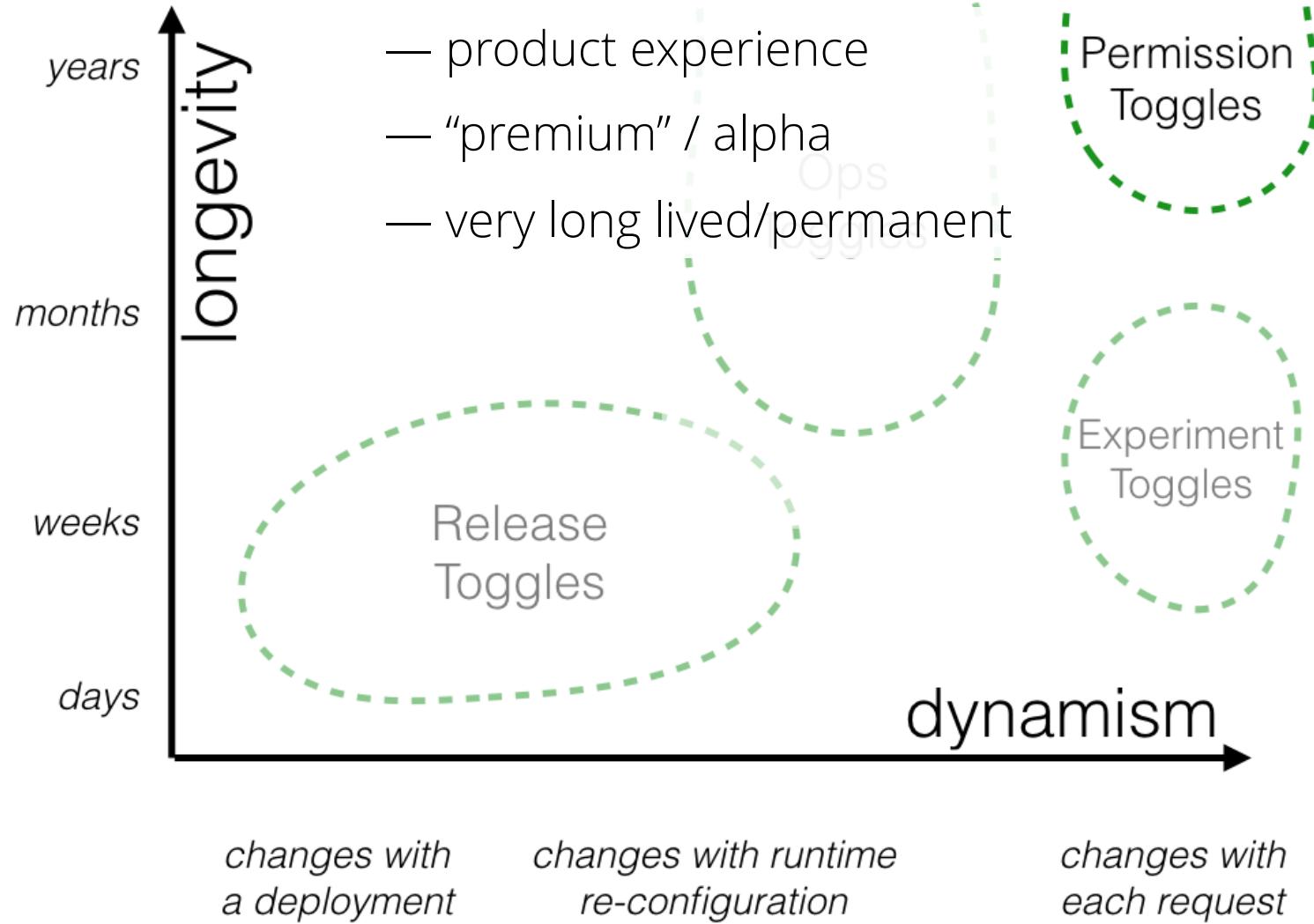




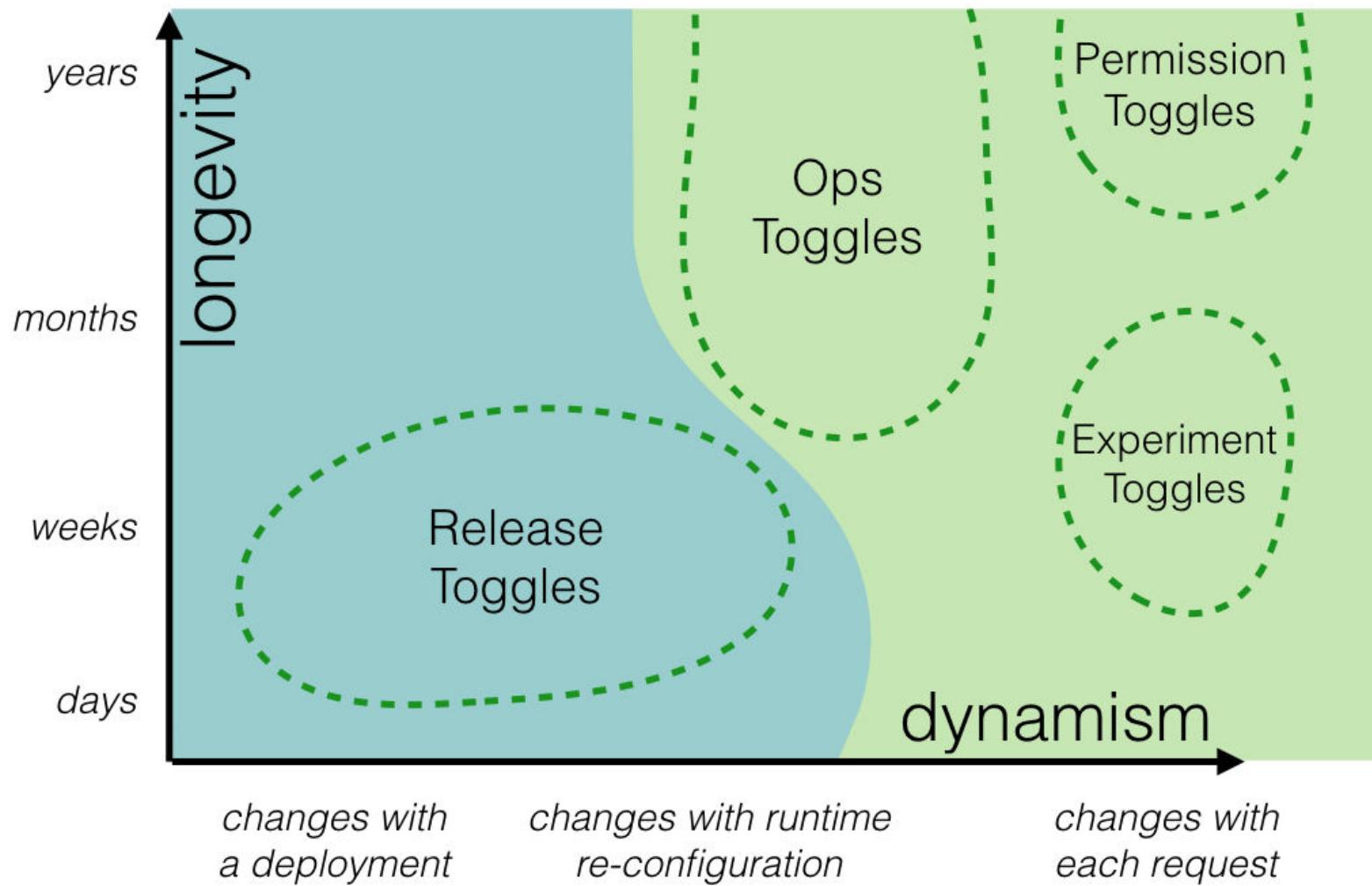
*changes with
a deployment*

*changes with runtime
re-configuration*

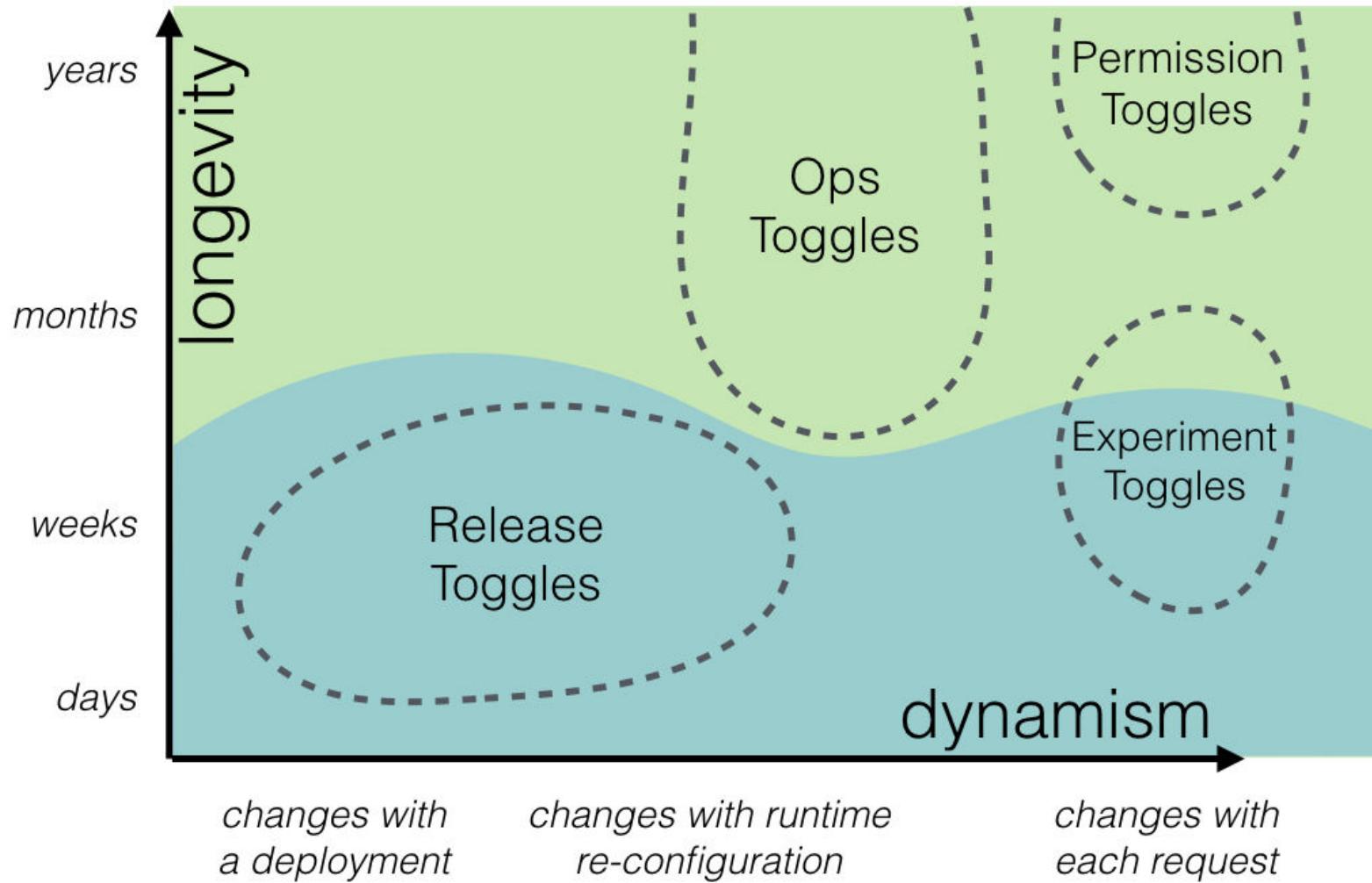
*changes with
each request*



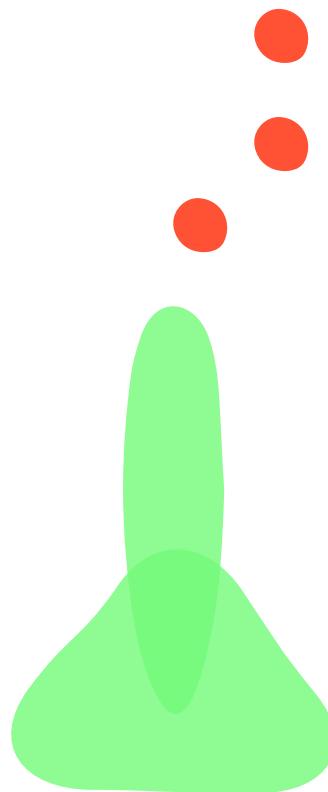
Static versus Dynamic Toggles

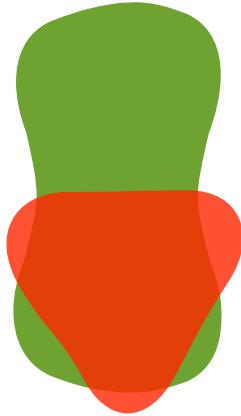


Long-lived versus Transient Toggles



Experimental Toggles





removed as soon as feature decision is resolved

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



The screenshot shows a web browser window with the DZone DevOps Zone website loaded. The page title is "Feature Toggles are one of the Worst kinds of Technical Debt". Below the title, a sub-headline reads: "Technical debt is pretty bad, and feature toggles are some of the most horrible examples of technical debt." The author is Jim Bird, and the date is Aug. 11, 14. The article has 9,288 views. A sidebar on the right features a New Relic advertisement for a global bank case study.

Feature Toggles are one of the Worst kinds of Technical Debt

Technical debt is pretty bad, and feature toggles are some of the most horrible examples of technical debt.

by Jim Bird · Aug. 11, 14 · DevOps Zone

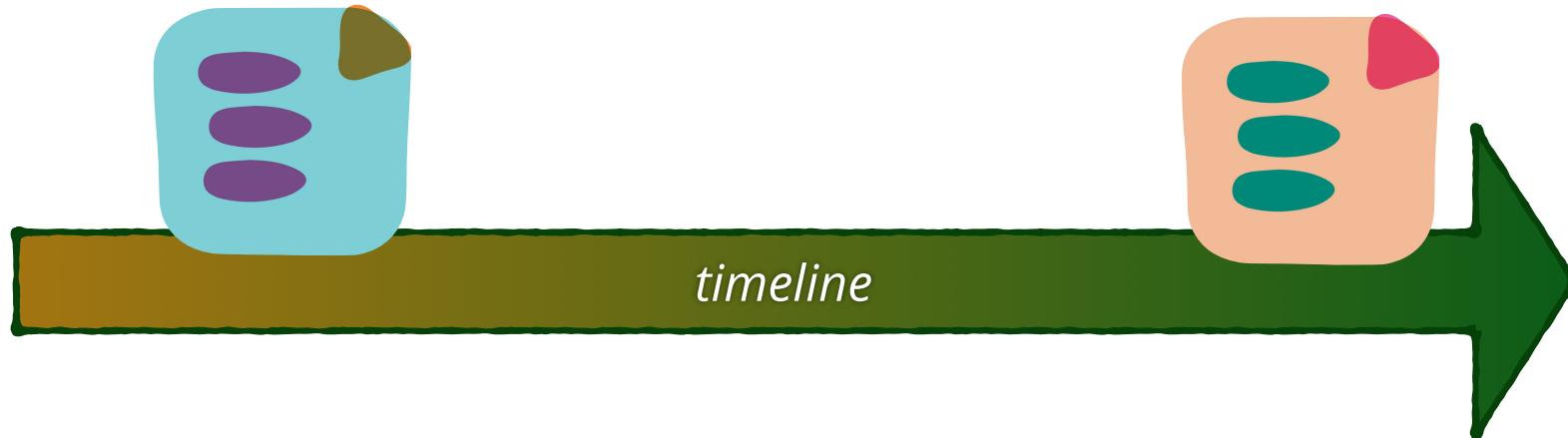
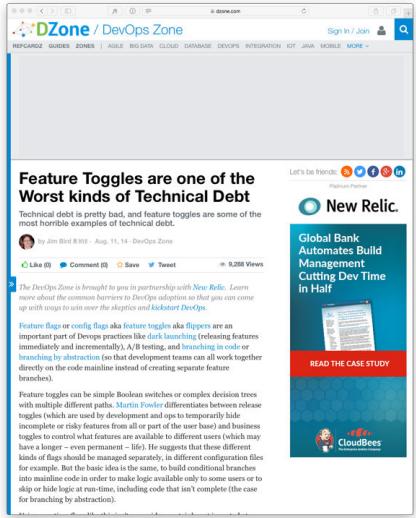
Like (0) Comment (0) Save Tweet 9,288 Views

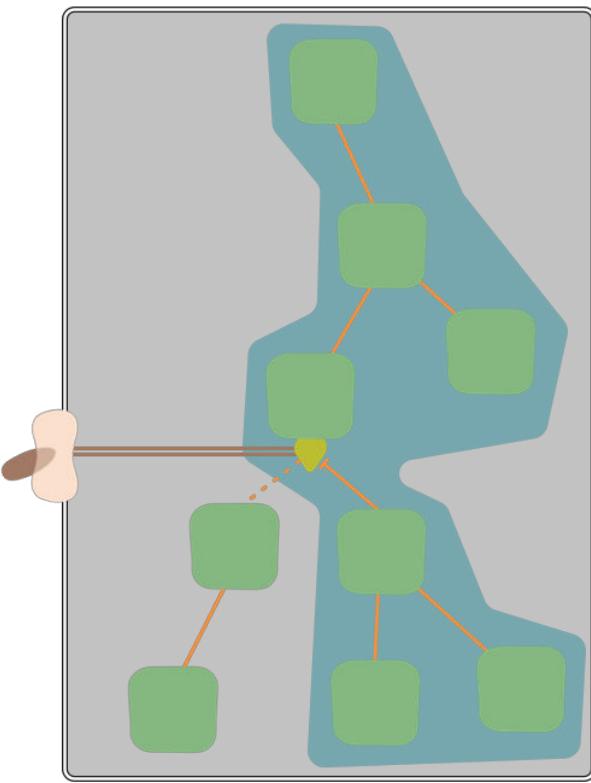
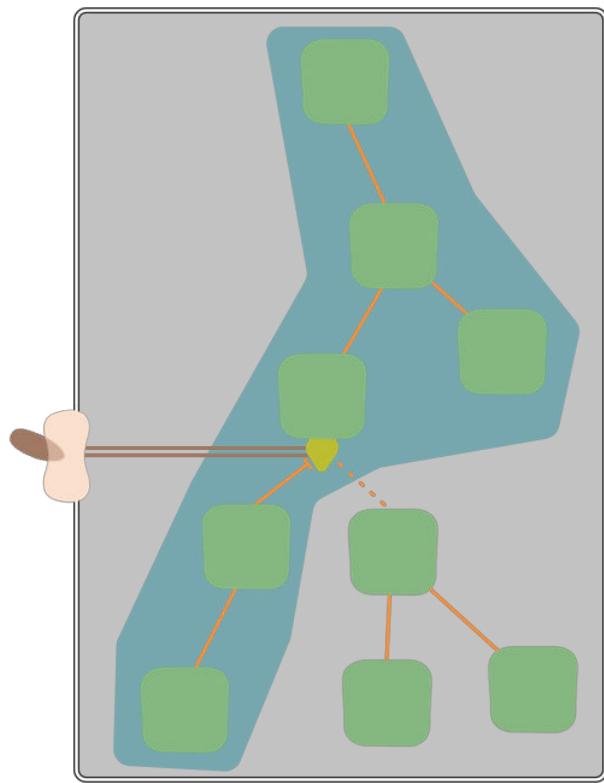
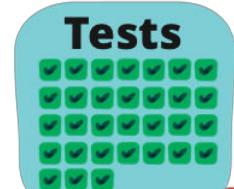
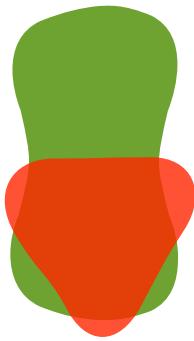
» The DevOps Zone is brought to you in partnership with New Relic. Learn more about the common barriers to DevOps adoption so that you can come up with ways to win over the skeptics and kickstart DevOps.

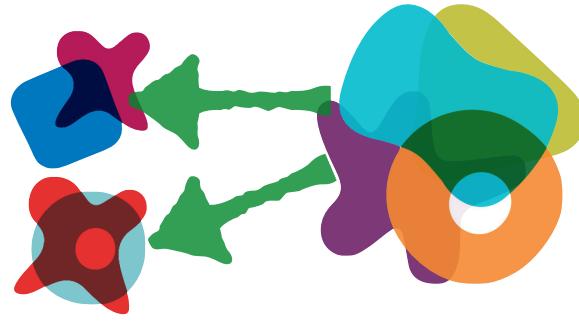
Feature flags or config flags aka feature toggles aka flippers are an important part of Devops practices like dark launching (releasing features immediately and incrementally), A/B testing, and branching in code or branching by abstraction (so that development teams can all work together directly on the code mainline instead of creating separate feature branches).

Feature toggles can be simple Boolean switches or complex decision trees with multiple different paths. Martin Fowler differentiates between release toggles (which are used by development and ops to temporarily hide incomplete or risky features from all or part of the user base) and business toggles to control what features are available to different users (which may have a longer – even permanent – life). He suggests that these different kinds of flags should be managed separately, in different configuration files for example. But the basic idea is the same, to build conditional branches into mainline code in order to make logic available only to some users or to skip or hide logic at run-time, including code that isn't complete (the case for branching by abstraction).

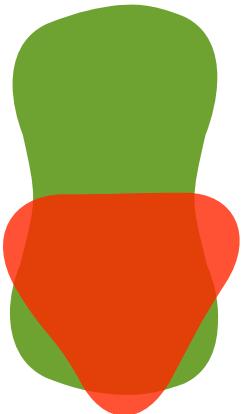
<https://dzone.com/articles/feature-toggles-are-one-worst>

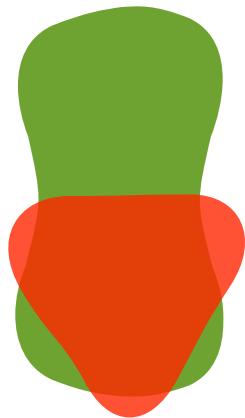
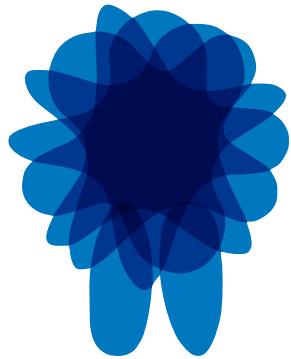




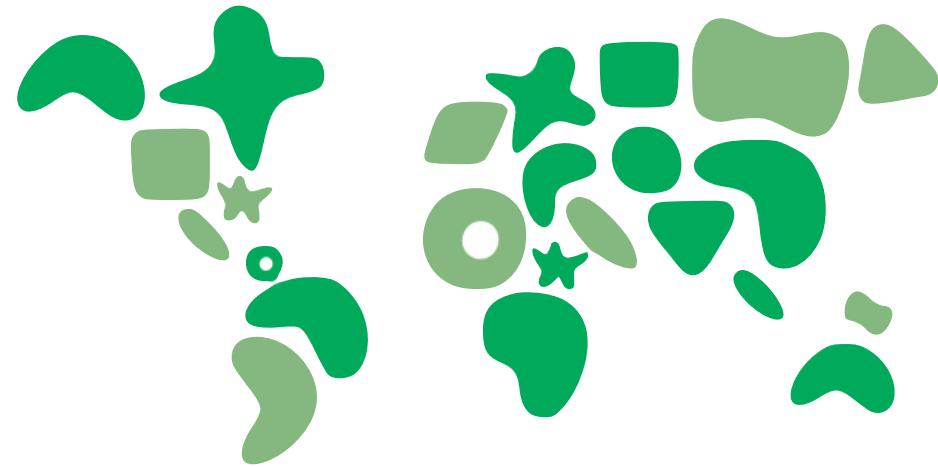


don't create toggle that
depend on other toggles.

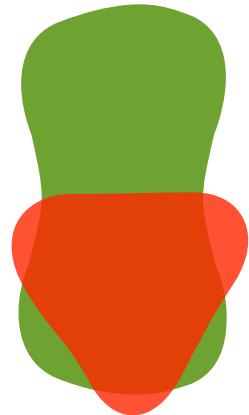




works on all platforms &
technology stacks



ubiquitous



Togglz – Features flag for Java

www.togglz.org

Reader

The screenshot shows a web browser window displaying the Togglz website. The title bar reads "Togglz – Features flag for Java". The address bar shows the URL "www.togglz.org". A "Reader" button is visible in the top right corner. The main content area features the Togglz logo (blue "togg" followed by a silver "l" with a red "on" switch) and the text "Feature Flags for the Java platform". On the left, a sidebar contains navigation links under four categories: MAIN (Home, Downloads, Source Code, Forums, Issue Tracker, stackoverflow.com, Continuous Integration, License), REFERENCE (What's new?, Getting Started, Javadocs 2.0.0.Final, Javadocs 1.1.0.Final, Javadocs 1.0.0.Final, Updating Notes), and DOCUMENTATION. The main content area includes sections for "Togglz" (introduction), "What is it about?" (description of Feature Toggles), "News" (announcing Togglz 2.0.0.Final release), and "Usage Examples" (links to usage examples and quickstart guide).

togglz

Feature Flags for the Java platform

Togglz

What is it about?

Togglz is an implementation of the [Feature Toggles](#) pattern for Java. Feature Toggles are a very common agile development practices in the context of continuous deployment and delivery. The basic idea is to associate a toggle with each new feature you are working on. This allows you to enable or disable these features at application runtime, even for individual users.

Want to learn more? Have a look at an [usage example](#) or check the [quickstart guide](#).

News

01-Jul-2013

Togglz 2.0.0.Final released

I'm very happy to announce the release of Togglz 2.0.0.Final. This new version is the result of many months of hard work. Many core concepts of Togglz have been revised to provide much more flexibility.

The most noteworthy change in Togglz 2.0.0.Final is the new extendible feature activation mechanism that allows to implement custom strategies for activating features. Beside that there are many other updates.

www.togglz.org

Togglz - Features flag for Java

www.togglz.org/documentation/activation-strategies.html

Reader

The screenshot shows a web browser displaying the Togglz documentation. The title bar reads "Togglz - Features flag for Java" and the URL is "www.togglz.org/documentation/activation-strategies.html". A "Reader" button is visible in the top right corner. The main content area features the Togglz logo with a "on" switch icon and the text "Feature Flags for the Java platform". On the left, a sidebar contains navigation links under four categories: MAIN, REFERENCE, and DOCUMENTATION. The MAIN category includes links for Home, Downloads, Source Code, Forums, Issue Tracker, stackoverflow.com, Continuous Integration, and License. The REFERENCE category includes What's new?, Getting Started, Javadocs 2.0.0.Final, Javadocs 1.1.0.Final, Javadocs 1.0.0.Final, Updating Notes, and a link to the Admin Console. The DOCUMENTATION category includes Overview, Installation, Configuration, Usage, and Admin Console. The main content area starts with a section titled "Activation Strategies" which defines activation strategies and lists default ones like Username, Gradual rollout, Release date, Client IP, Server IP, and ScriptEngine. It also mentions custom strategies and the Username strategy in detail, explaining how it compares usernames case-insensitively.

Activation Strategies

Togglz defines the concept of *activation strategies*. They are responsible to decide whether an enabled feature is active or not. Activation strategies can for example be used to activate features only for specific users, for specific client IPs or at a specified time.

Togglz ships with the following default strategies:

- Username
- Gradual rollout
- Release date
- Client IP
- Server IP
- ScriptEngine

The following sections will describe each strategy in detail. The last section [custom strategies](#) describes how to build you own strategies.

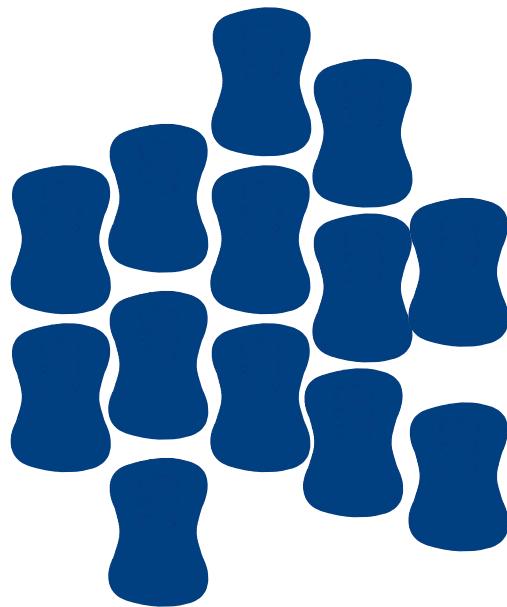
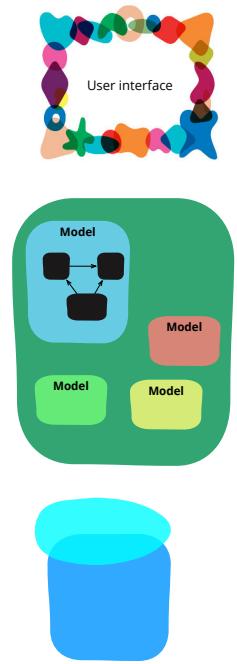
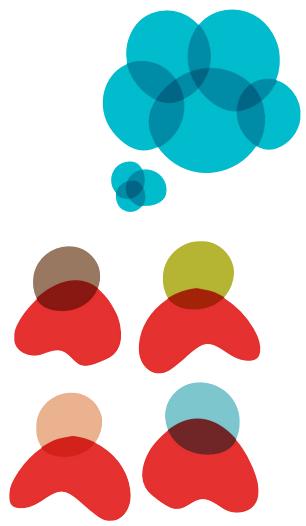
Username

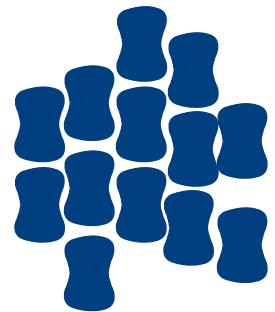
Enabling features for specific users was already supported in very early versions of Togglz, even before the activation strategy concept was introduced in Togglz 2.0.0.

If you select this strategy for a feature, you can specify an comma-separated list of users for which the feature should be active. Togglz will use the [UserProvider](#) you configured for the FeatureManager to determine the current user and compare it to that list.

Please note that Togglz will take case into account when comparing the usernames. So the users `admin` and `Admin` are NOT the same.

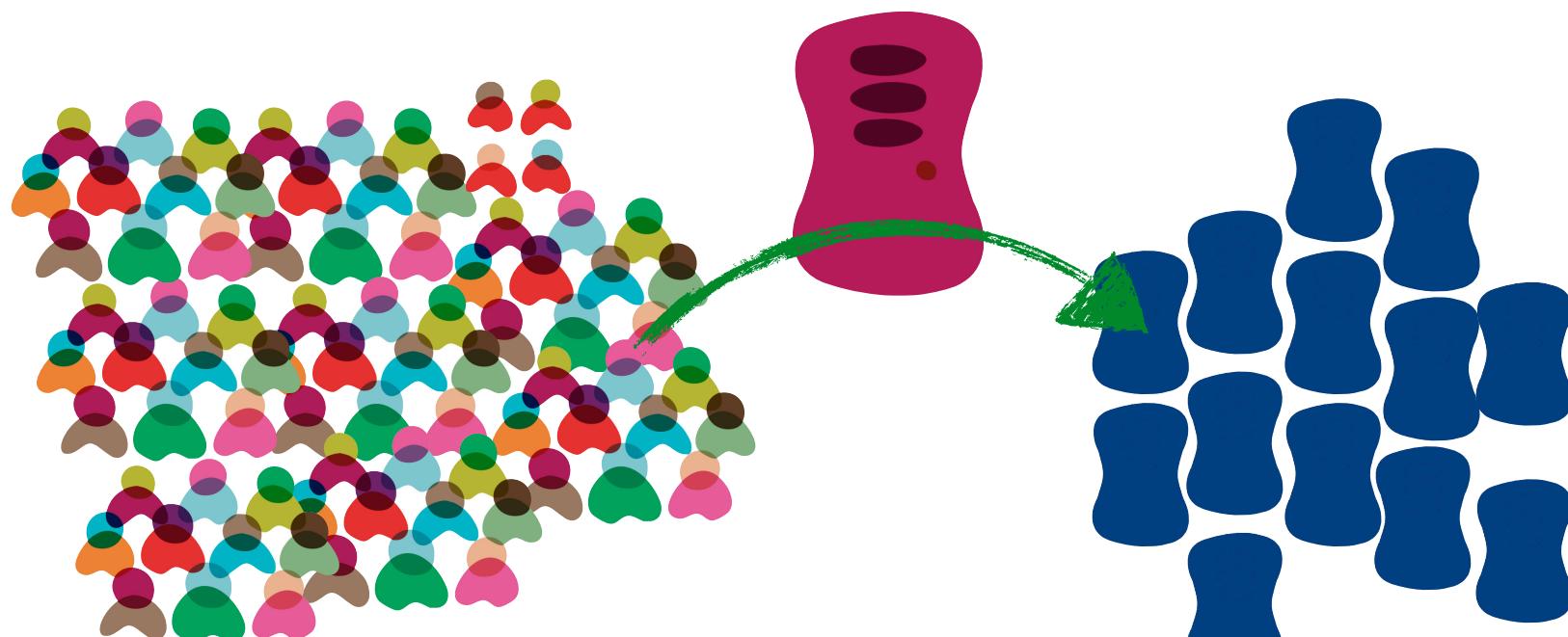
www.togglz.org



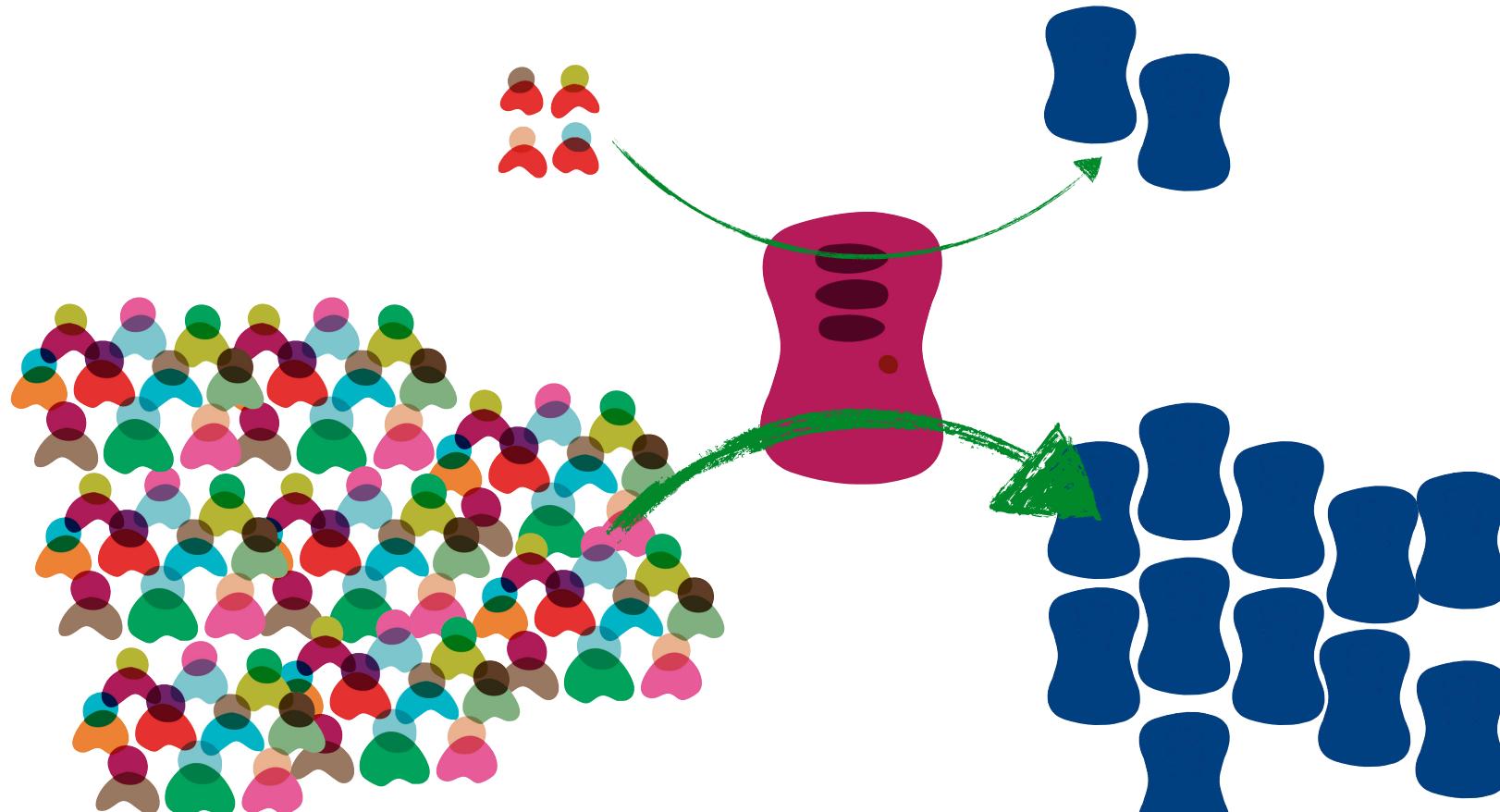


Canary Releasing

Canary Releasing

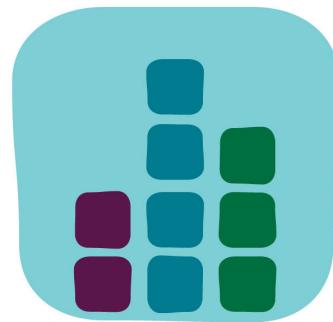


Canary Releasing

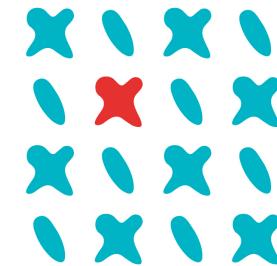


Canary Releasing

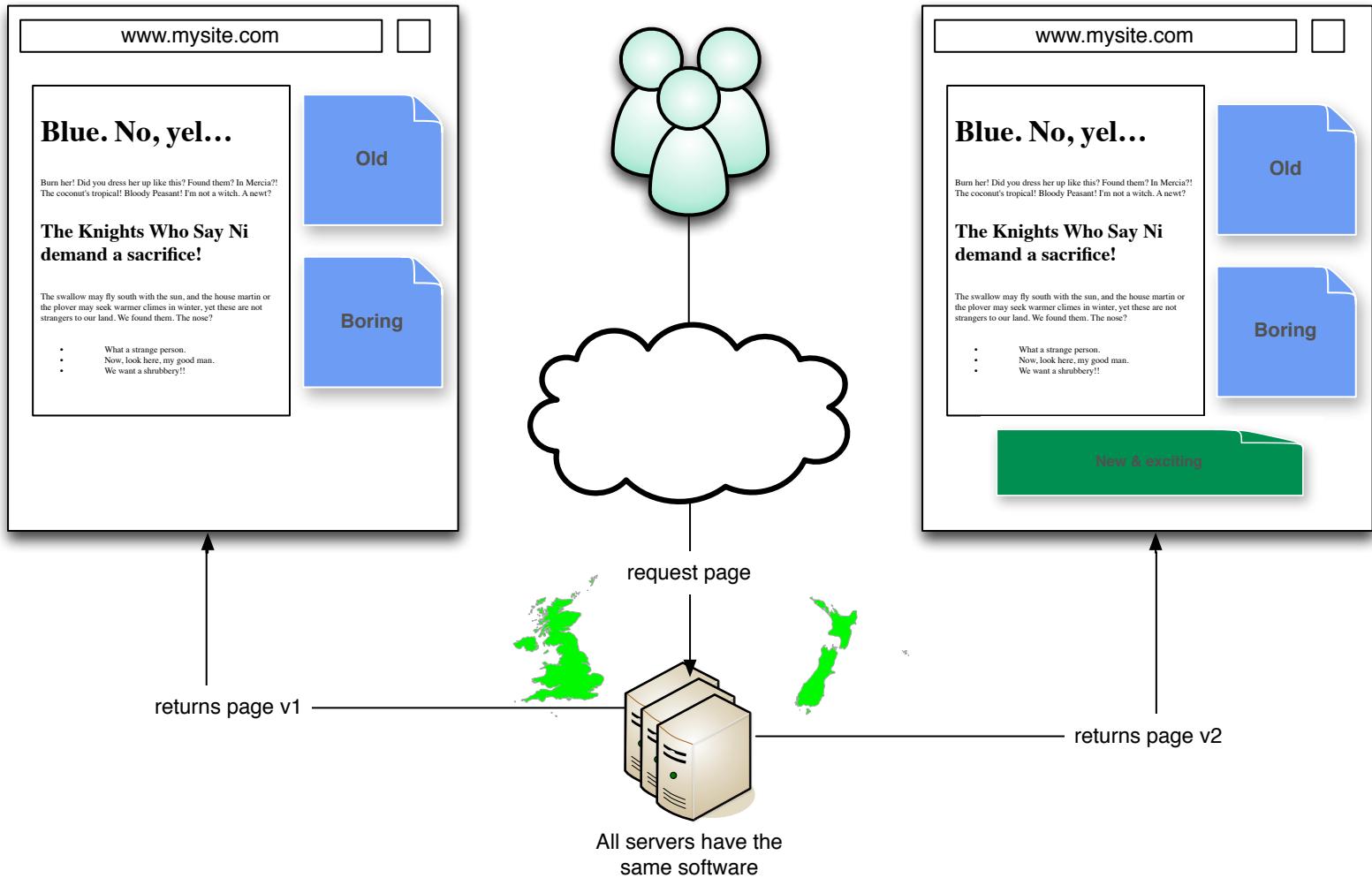
reduce risk of release



performance testing



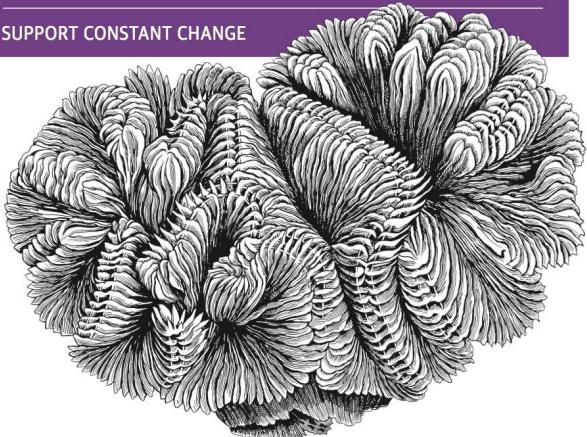
multi-variant testing



O'REILLY®

Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE



Neal Ford, Rebecca Parsons & Patrick Kua

 @neal4d
[http://
nealford.com](http://nealford.com)

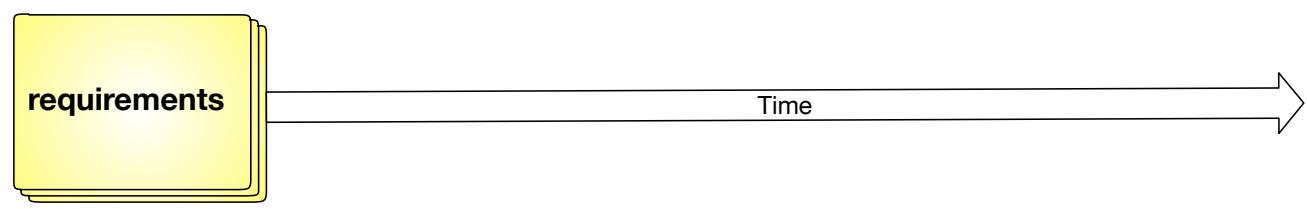


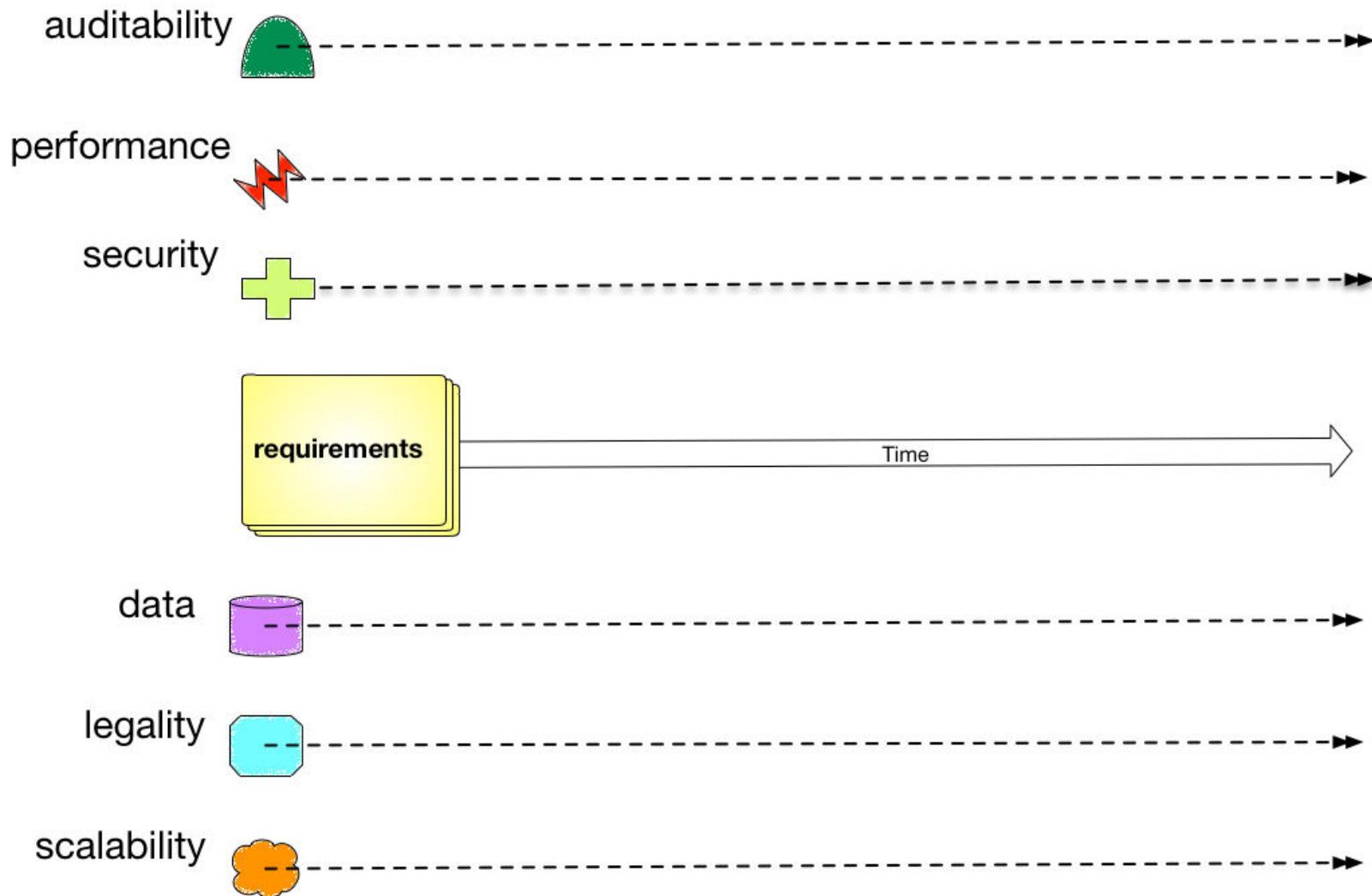
 @rebeccaparsons



 @patkua







accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

ility

failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
productivity
provability
recoverability
relevance
reliability
extensibility
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness

evolvability

determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

orthogonality
portability
precision
predictability
process capabilities
productability
provability
recoverability
relevance

tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

evolvability

•
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

•
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

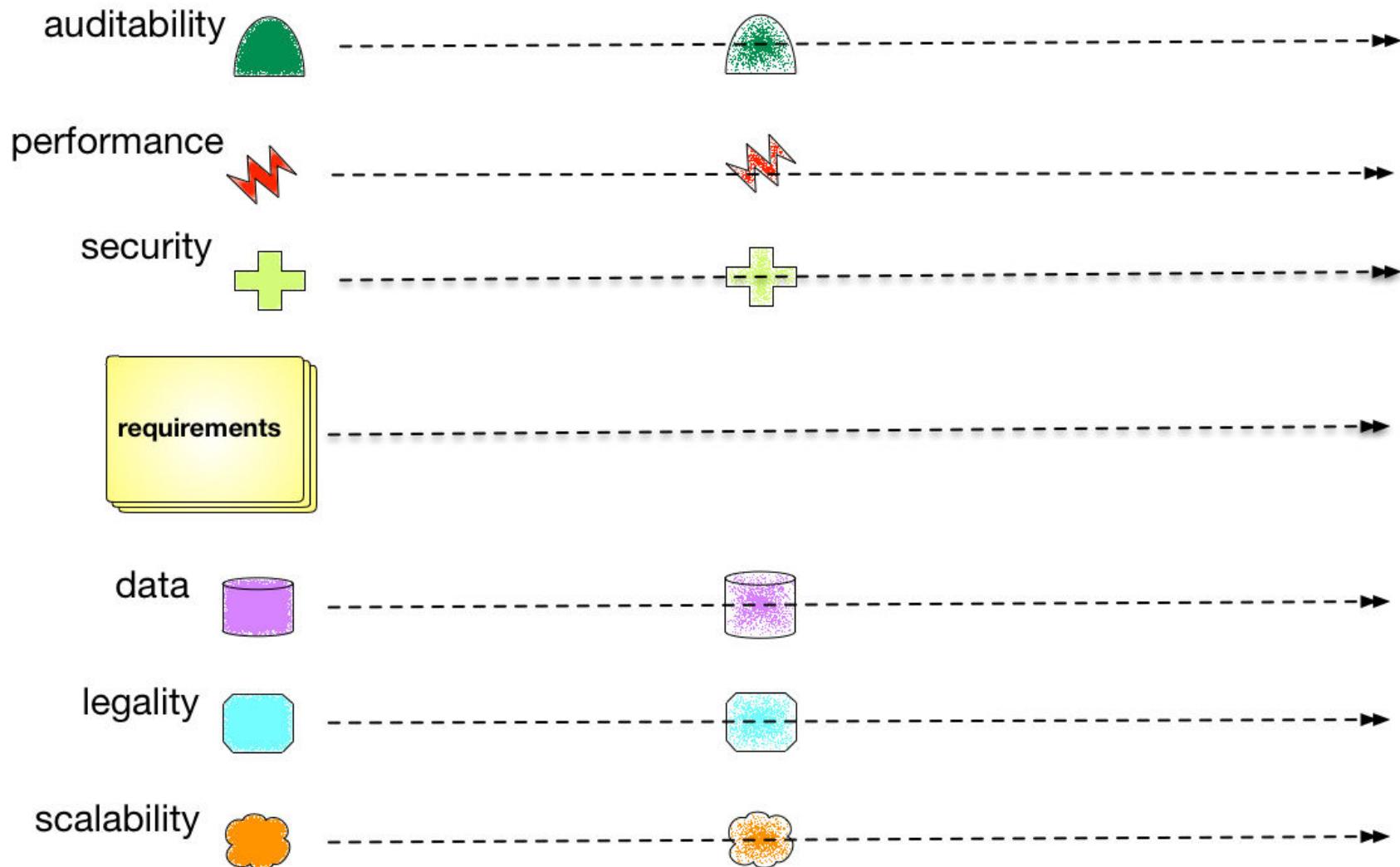
•
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy

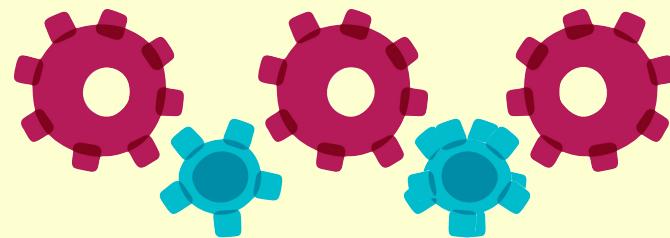
reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness

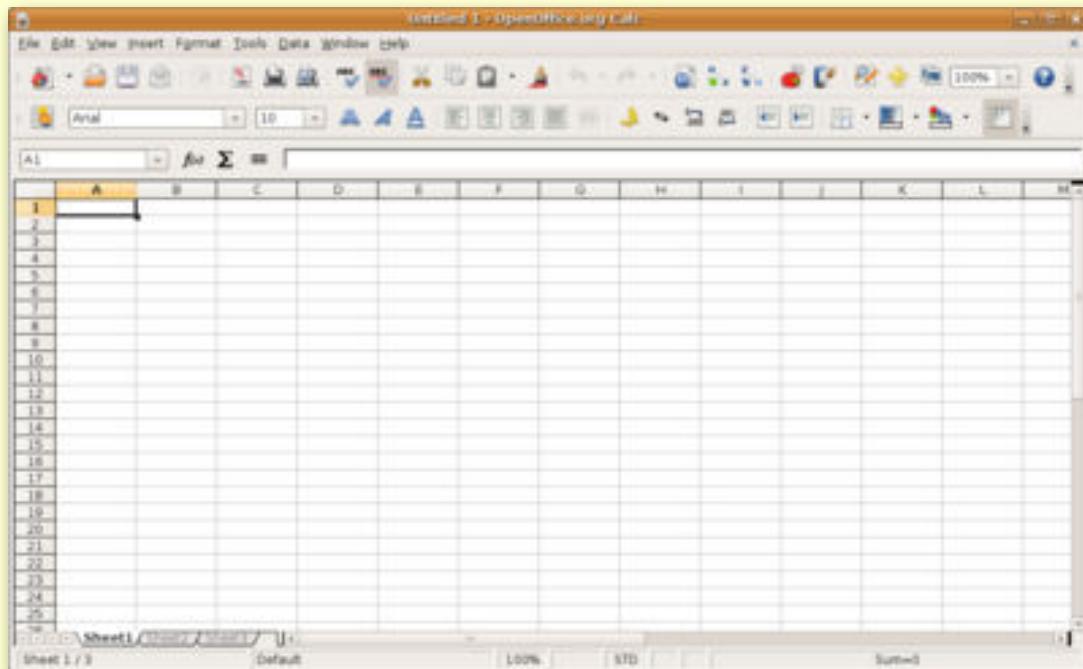
**How is long term planning
possible when things constantly
change in unEXpECtEd ways?**



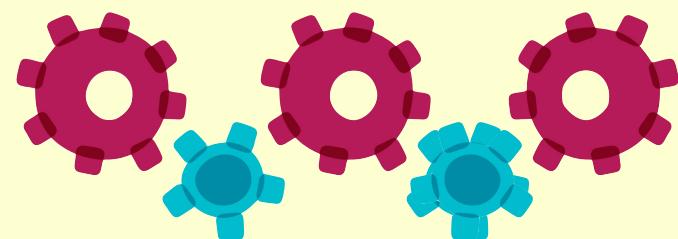
Penultima ↑ e



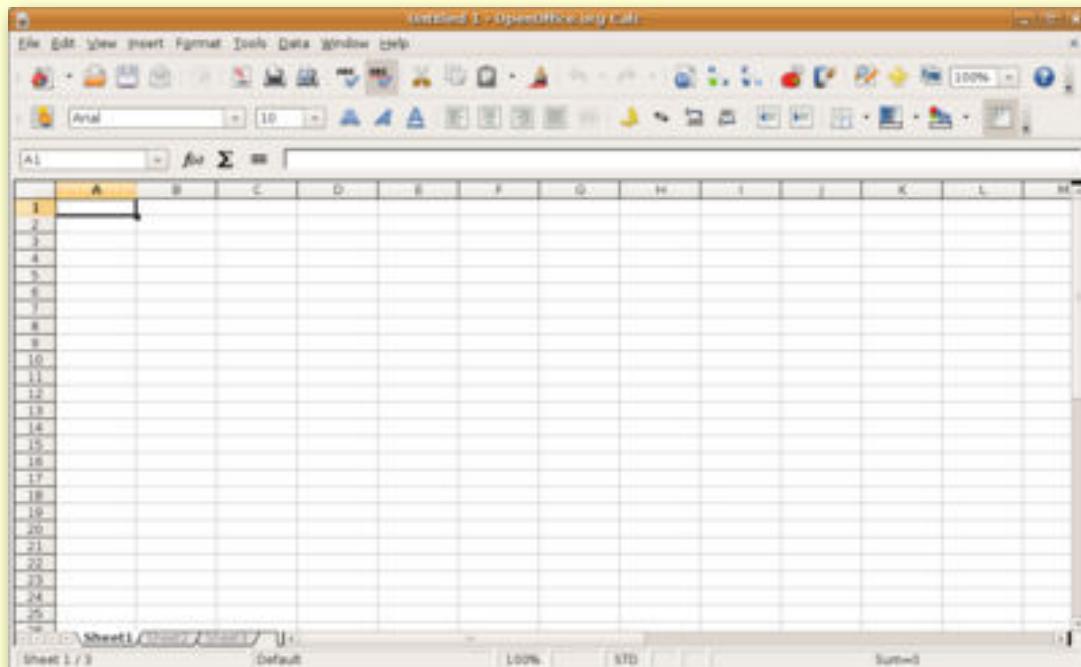
EA Spreadsheet



Penultima ↑ e

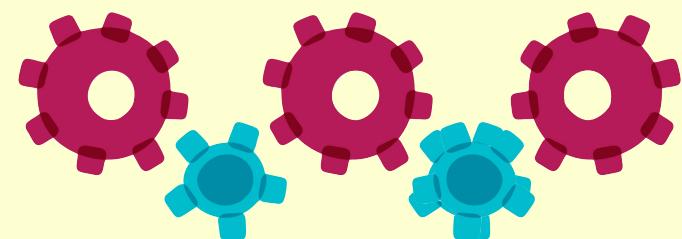


EA Spreadsheet

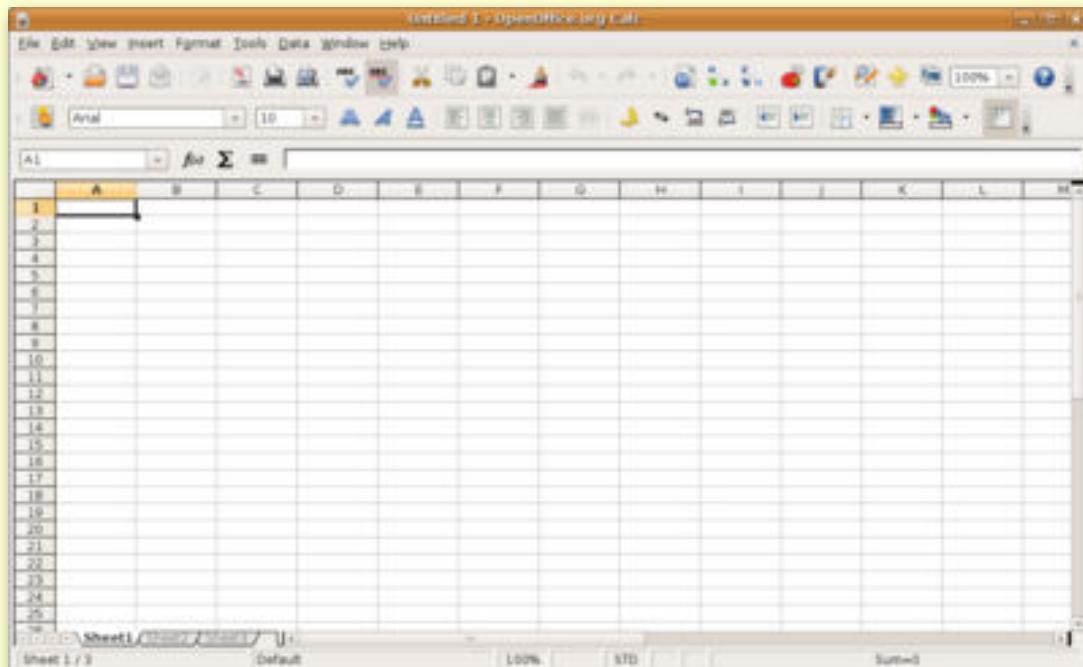


✓ definition

Penultima ↑ e



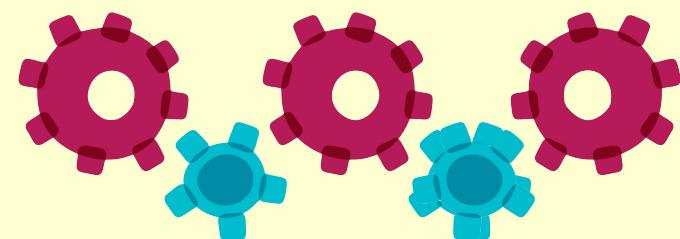
EA Spreadsheet



✓ definition

! verification

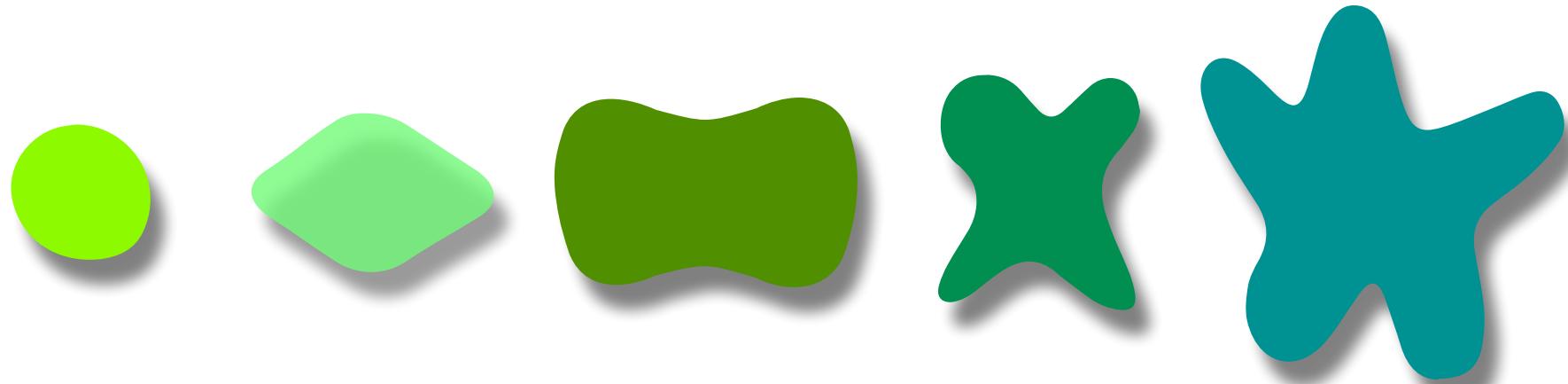
Penultima ↑ e



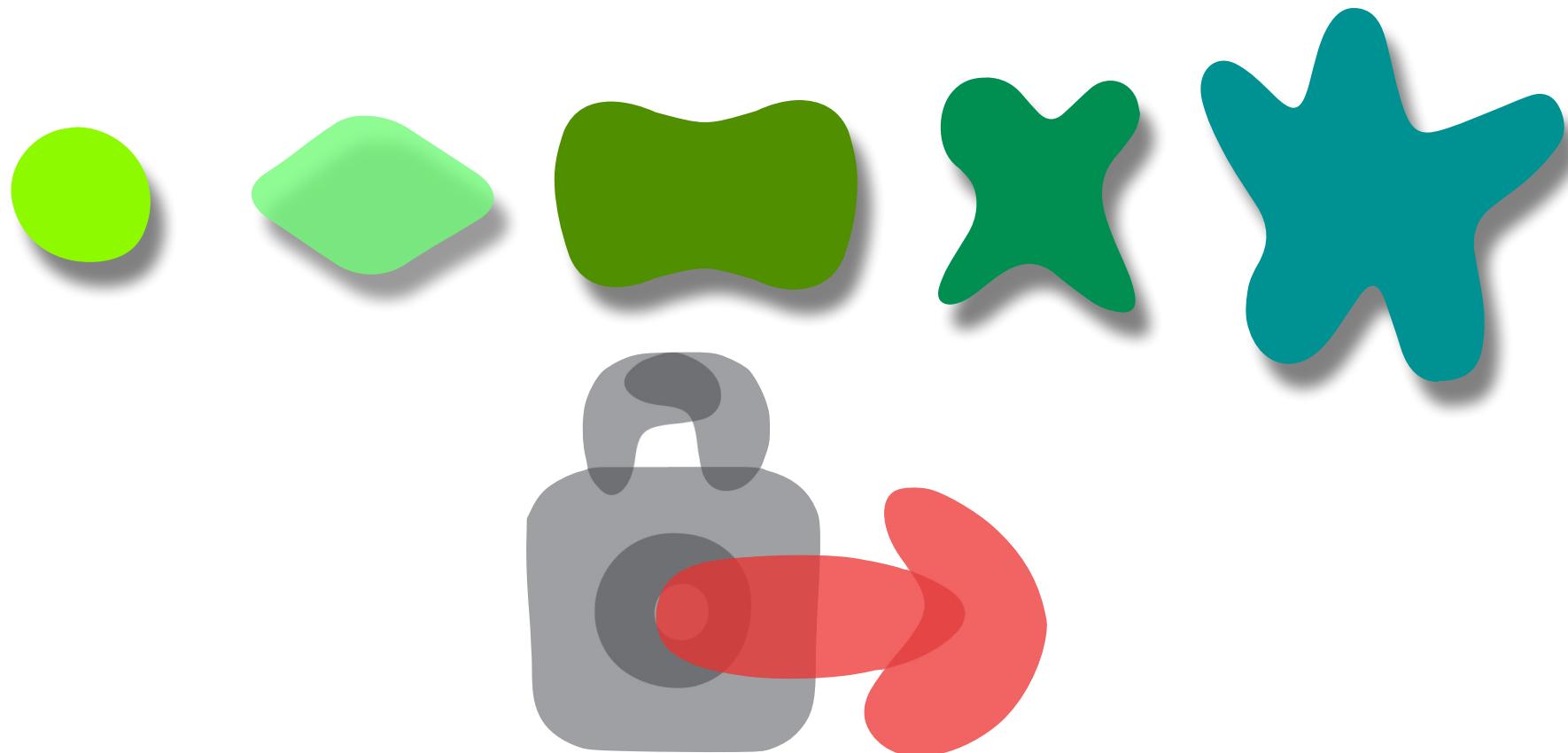
**Once I've built an architecture,
how can I prevent it from
gradually dEgRADiNg over time?**

sEcONd-ORdeR eFfEcT

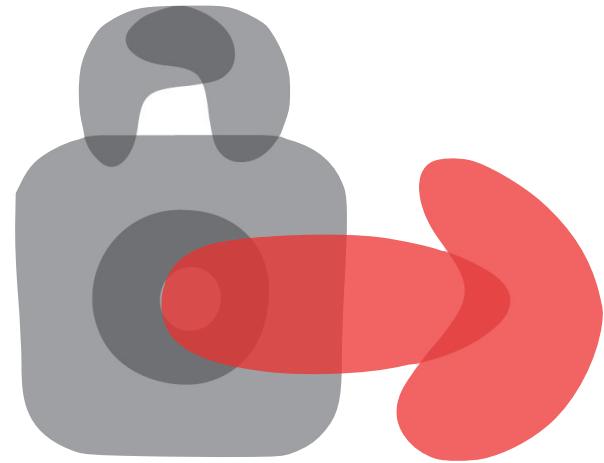
sEcOND-ORdeR eFfEcT



sEcOND-ORdeR eFfEcT



governance



Evolutionary Architecture

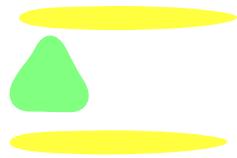
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change
across multiple dimensions.



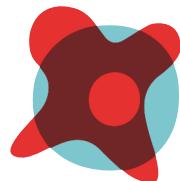


guided

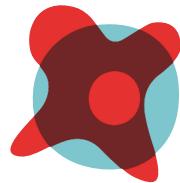
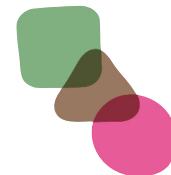
evolutionary computing fitness function:

a particular type of objective function that is used to summarize...how close a given design solution is to achieving the set aims.

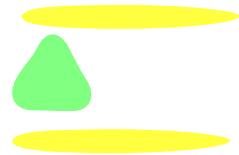
Traveling Salesman Problem



Traveling Salesman Problem



fitness function = length of route

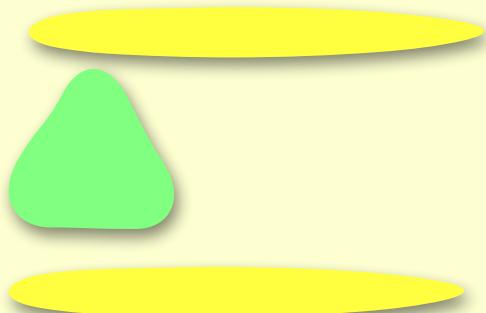


guided

architectural fitness function:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

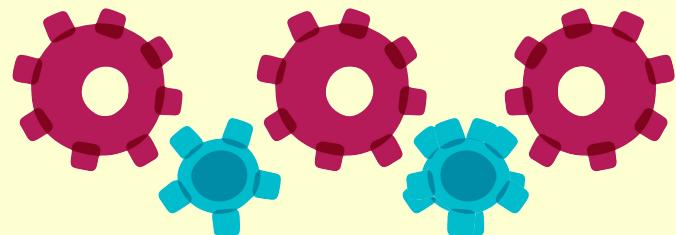
EA Spreadsheet



✓ definition

✓ *verification*

Penultima ↑ e



Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



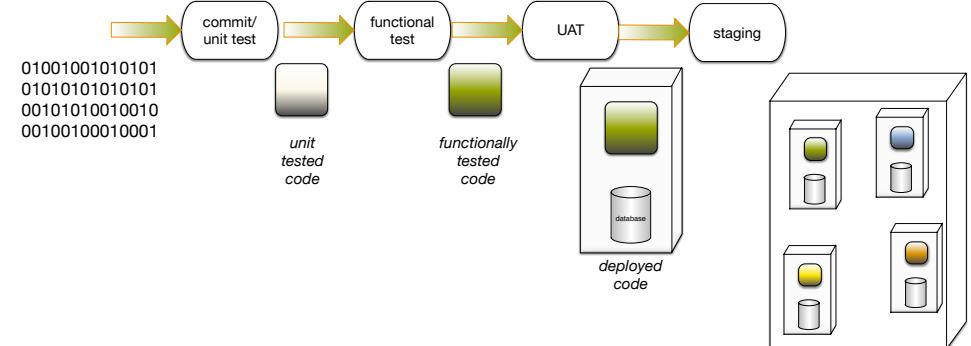
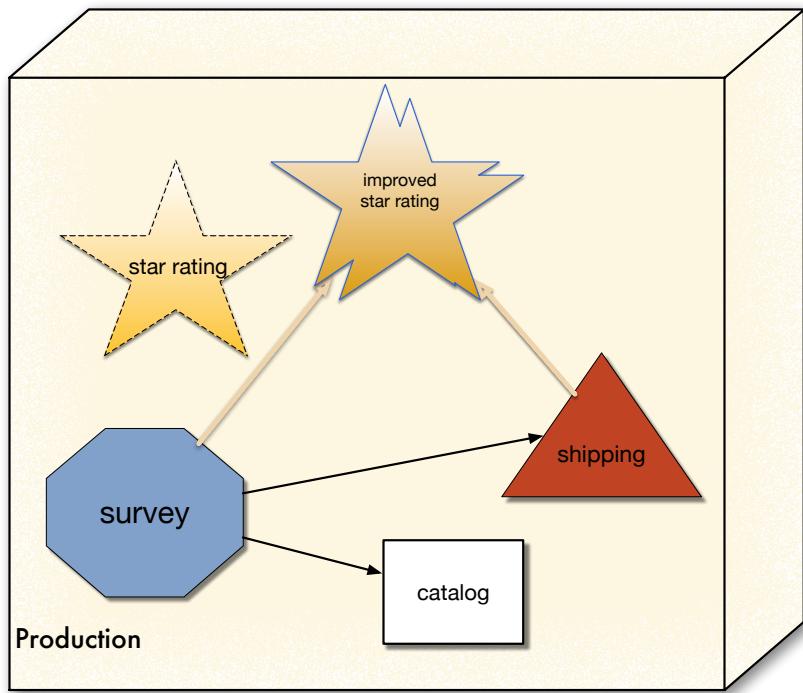
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.





incremental



Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



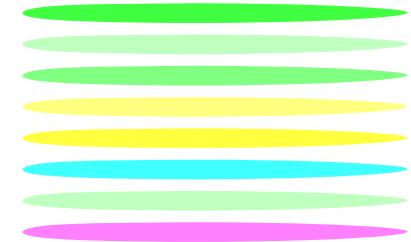
Evolutionary Architecture

An evolutionary architecture supports
guided,

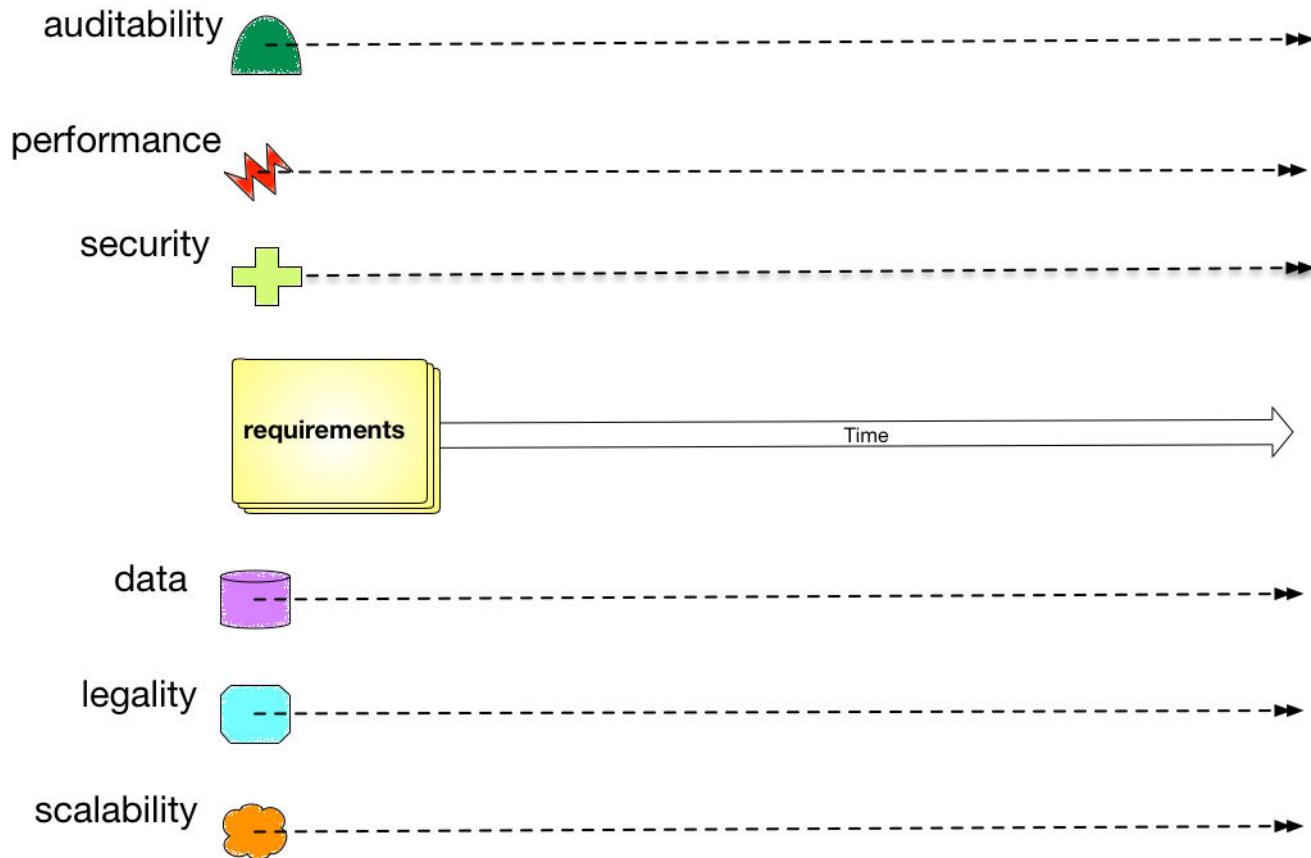
incremental change

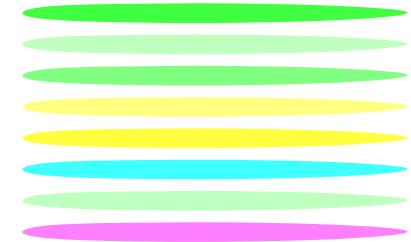
across ***multiple dimensions***



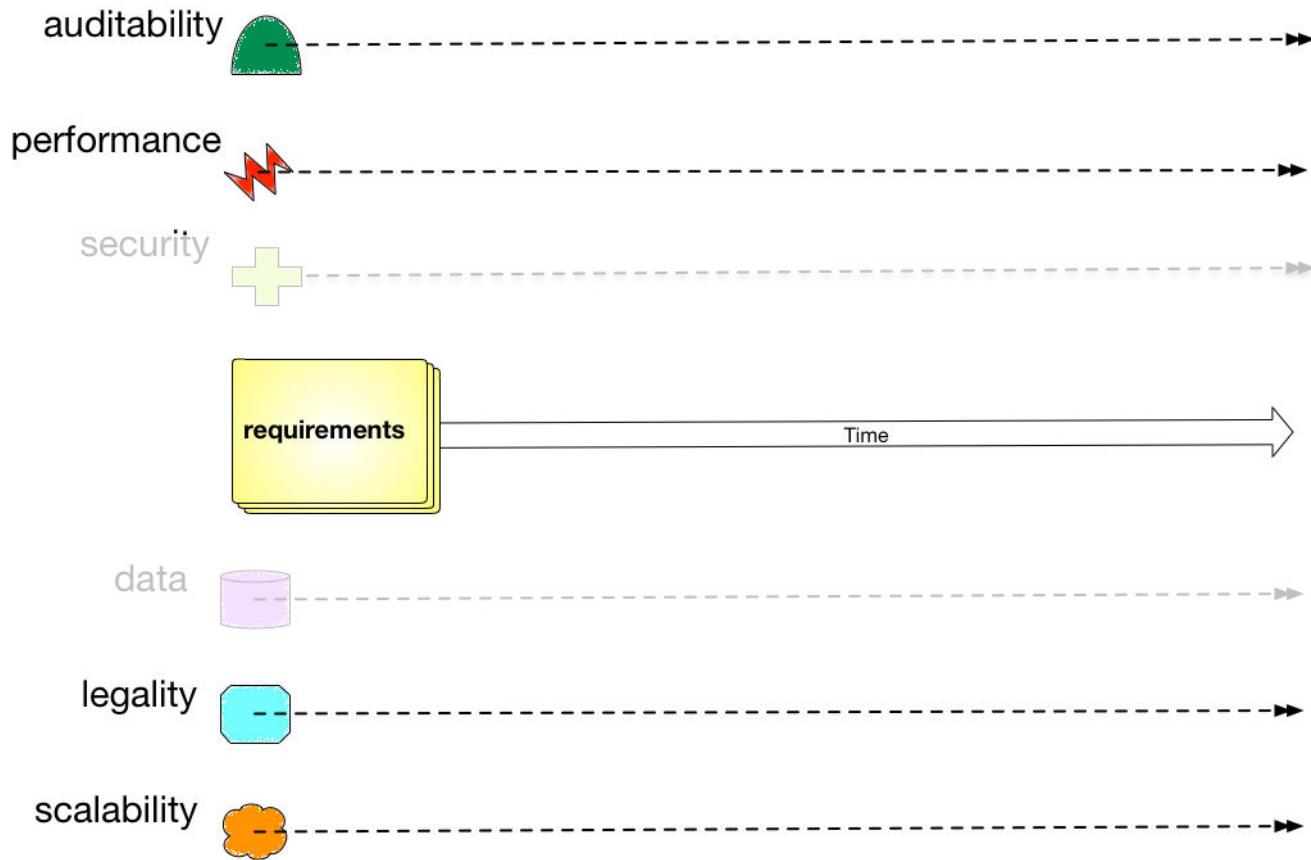


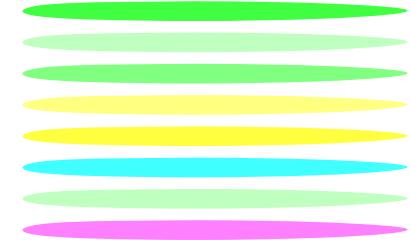
multiple dimensions



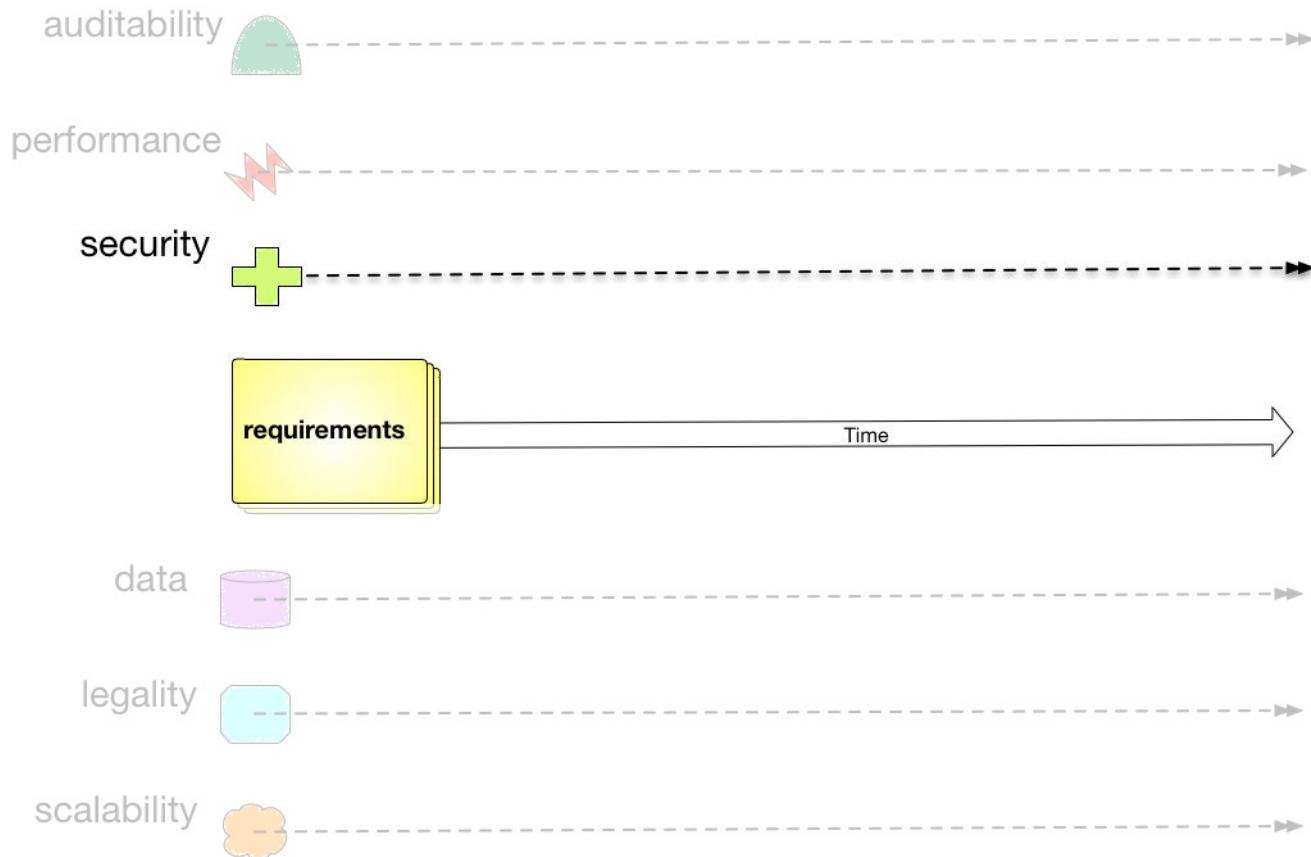


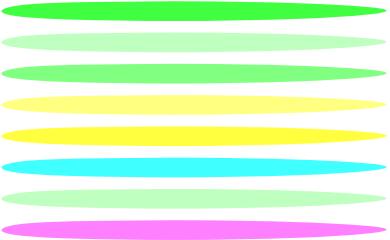
multiple dimensions



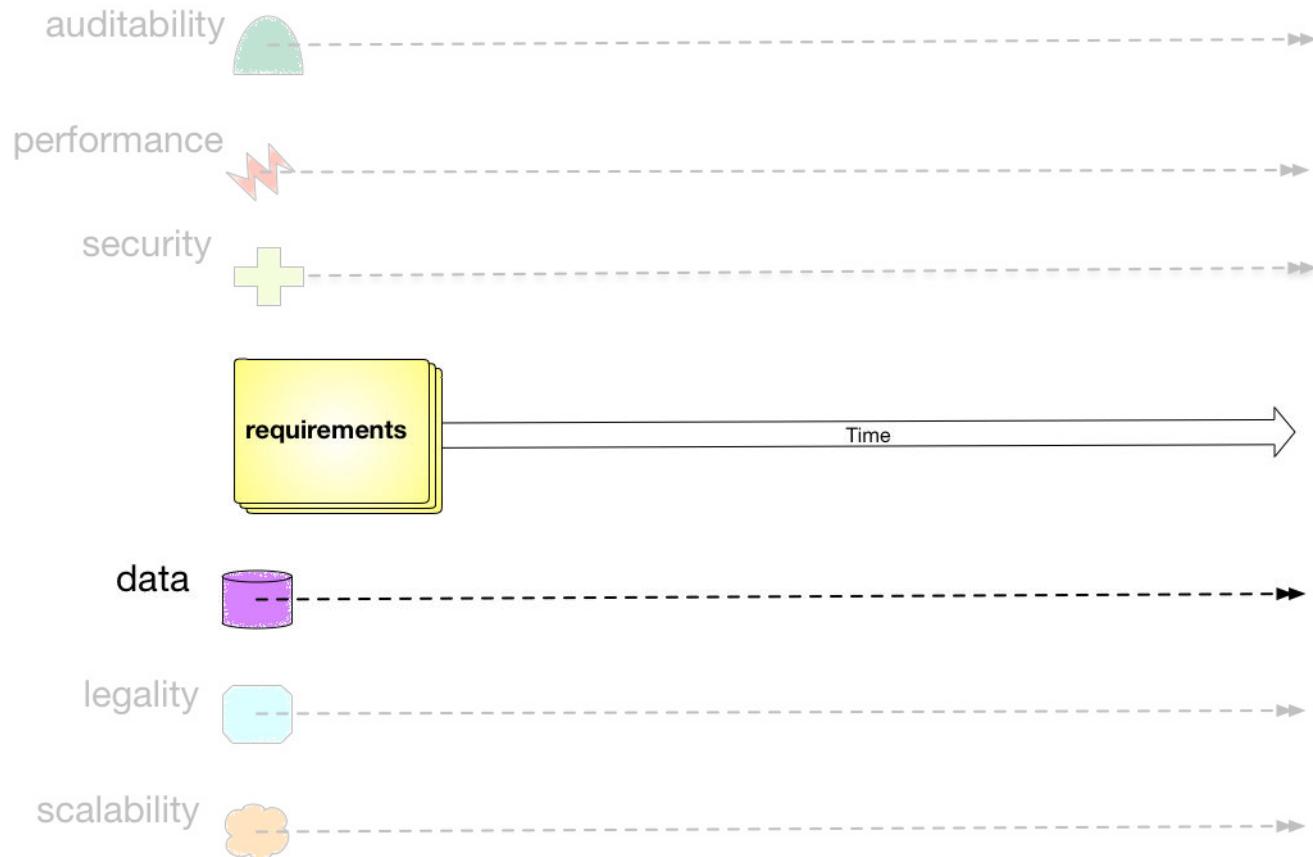


multiple dimensions





multiple dimensions



Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change

across ***multiple dimensions***



Evolutionary Architecture

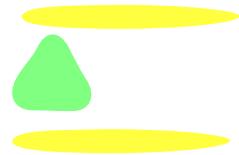
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change
across multiple dimensions.



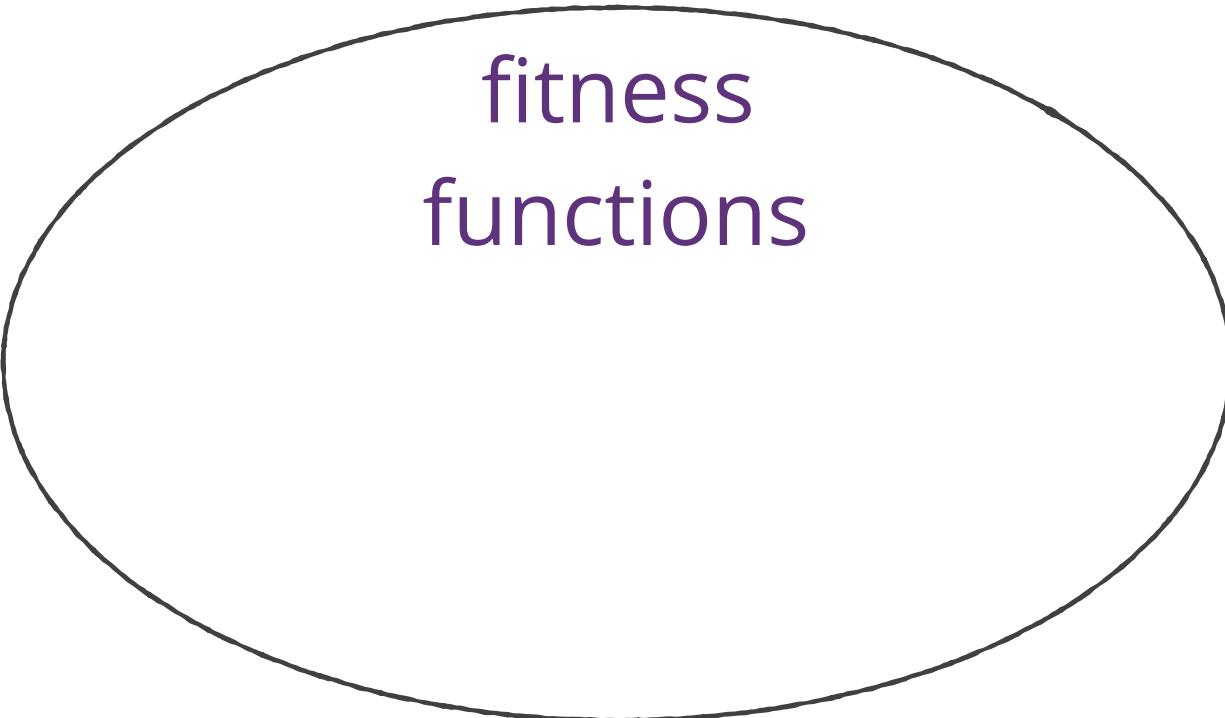


guided

architectural fitness function:

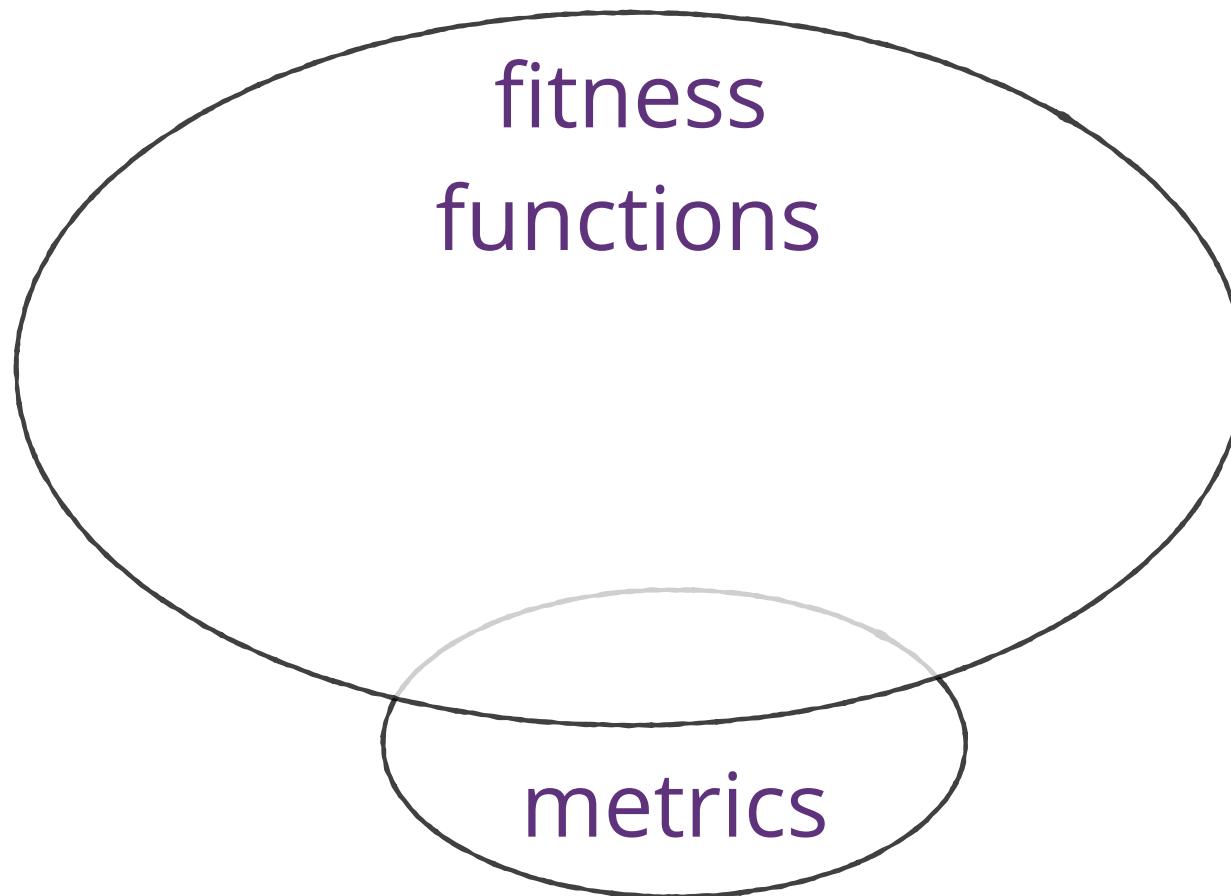
An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Fitness Functions

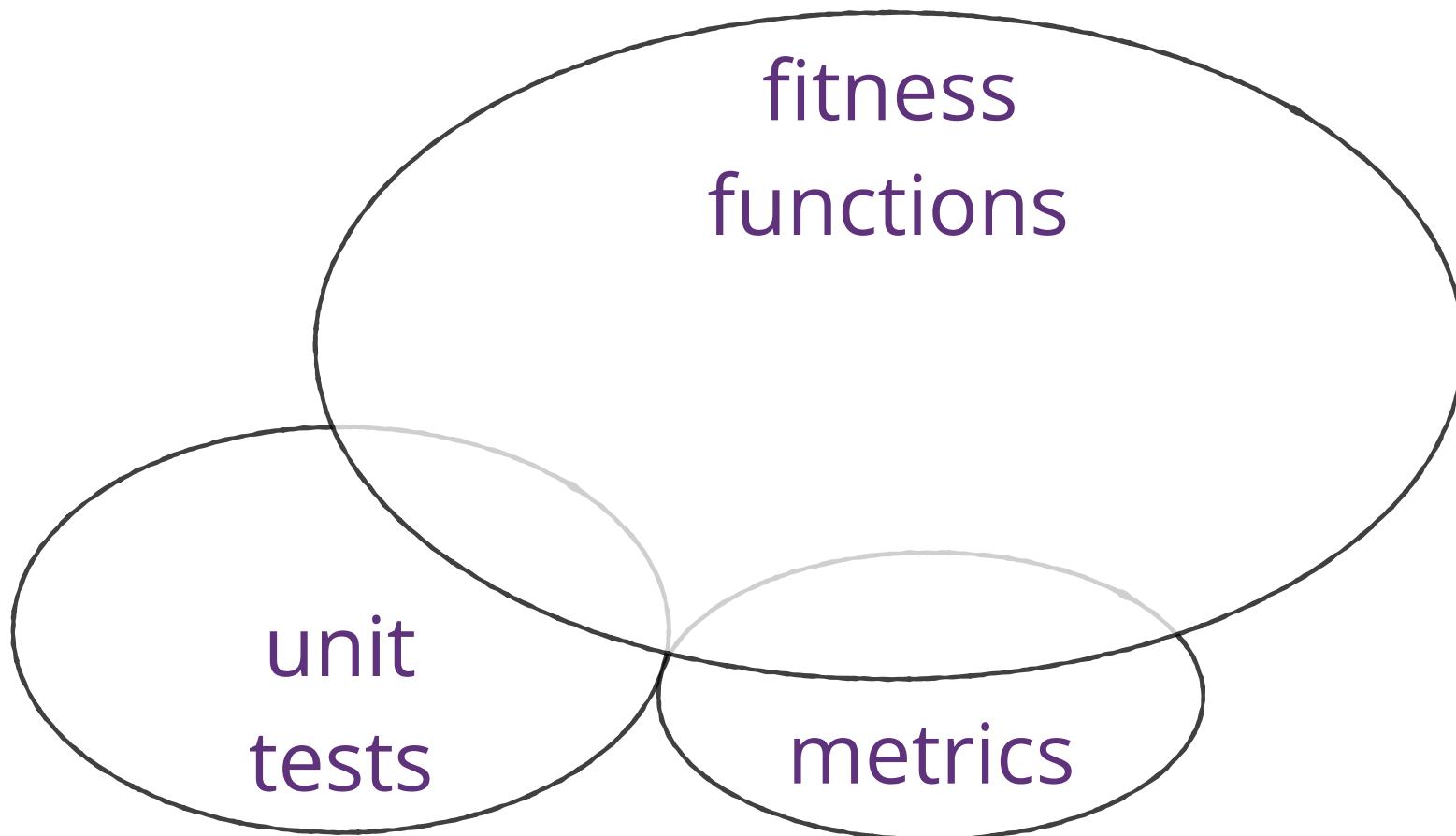


fitness
functions

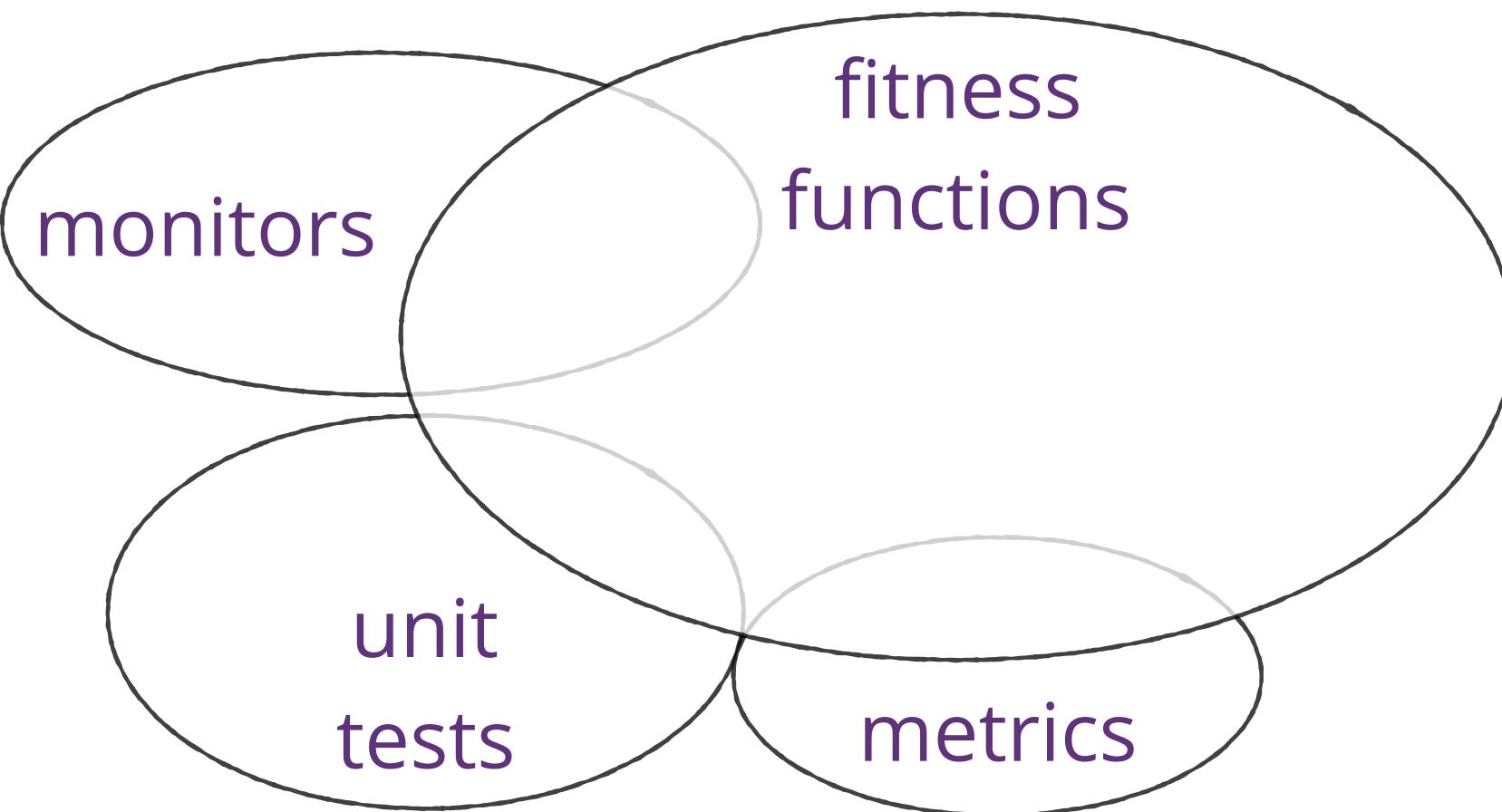
Fitness Functions



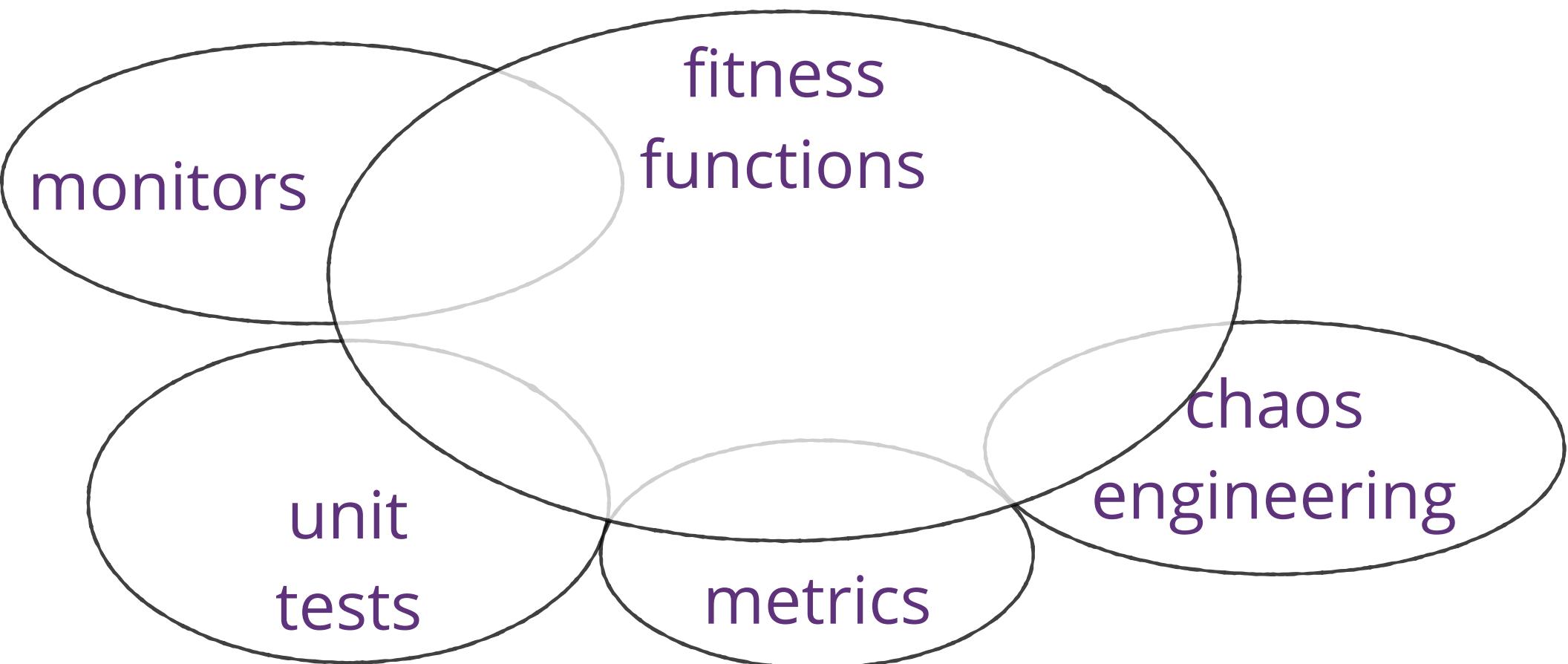
Fitness Functions



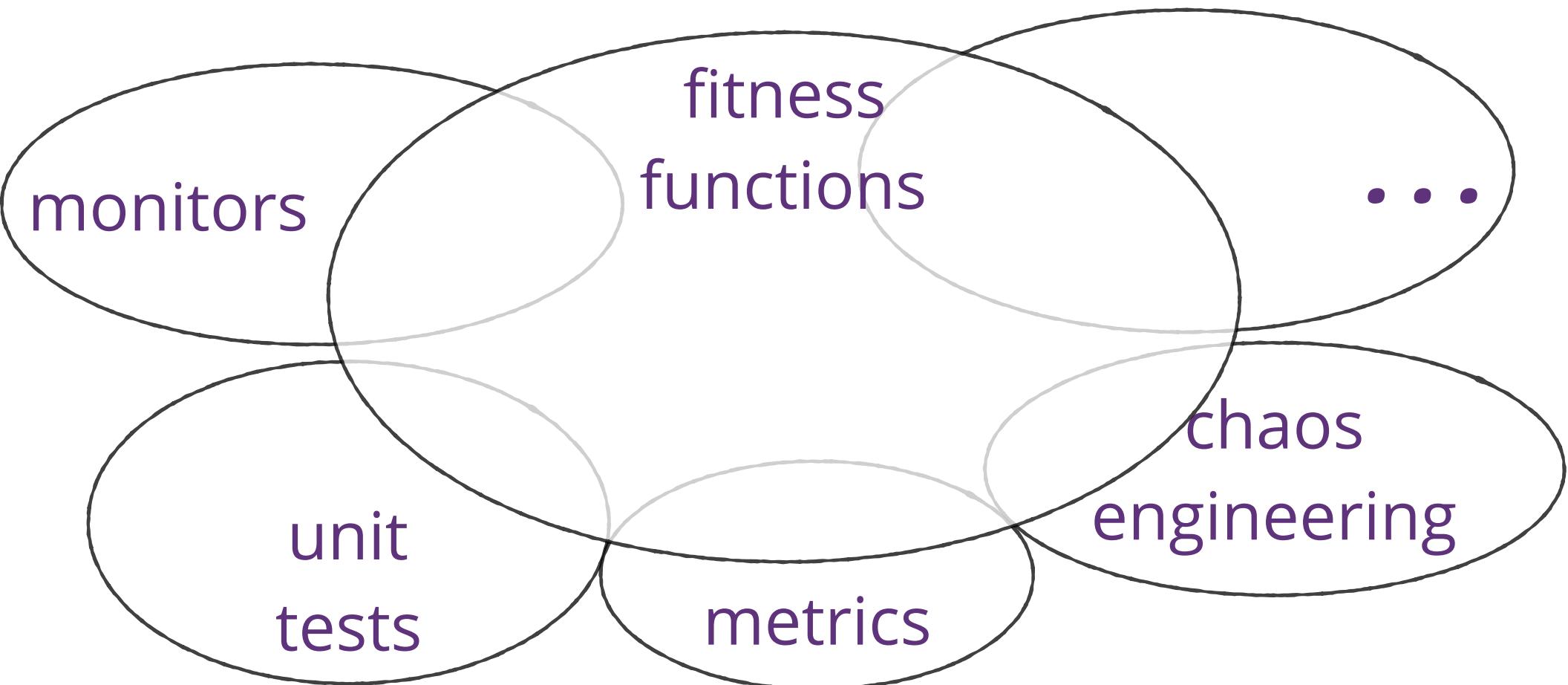
Fitness Functions



Fitness Functions

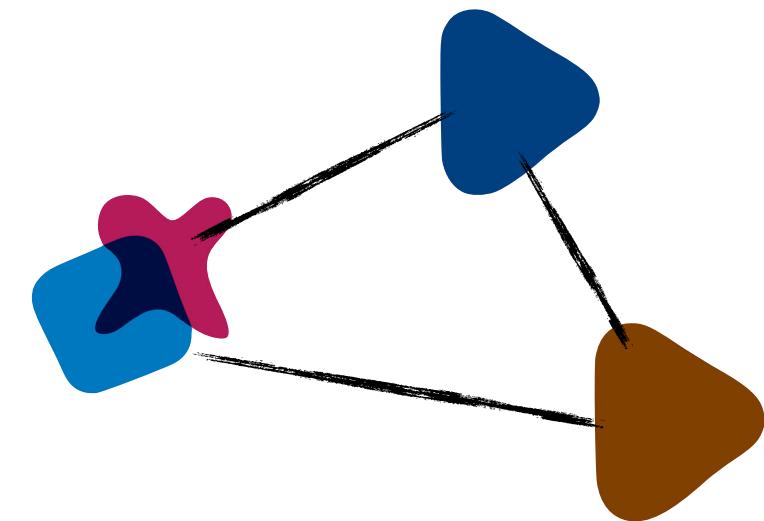


Fitness Functions

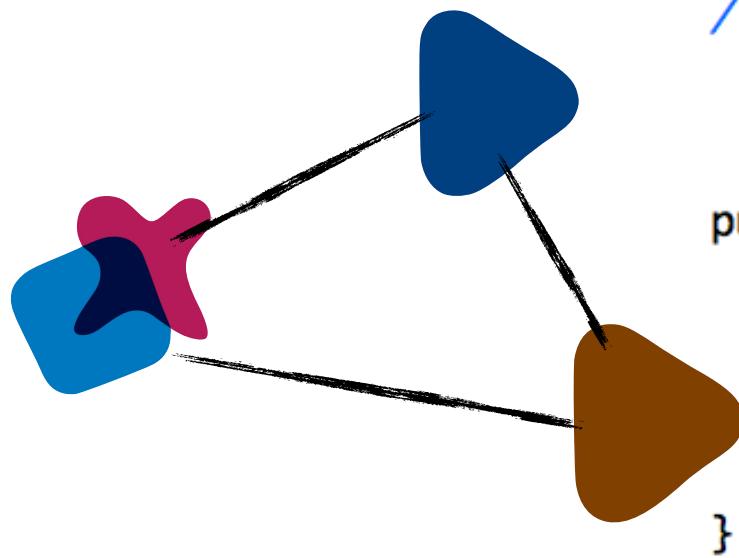


Cyclic Dependency Function

Cyclic Dependency Function



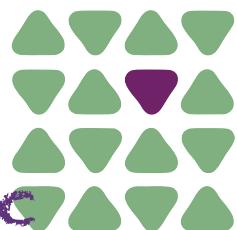
Cyclic Dependency Function



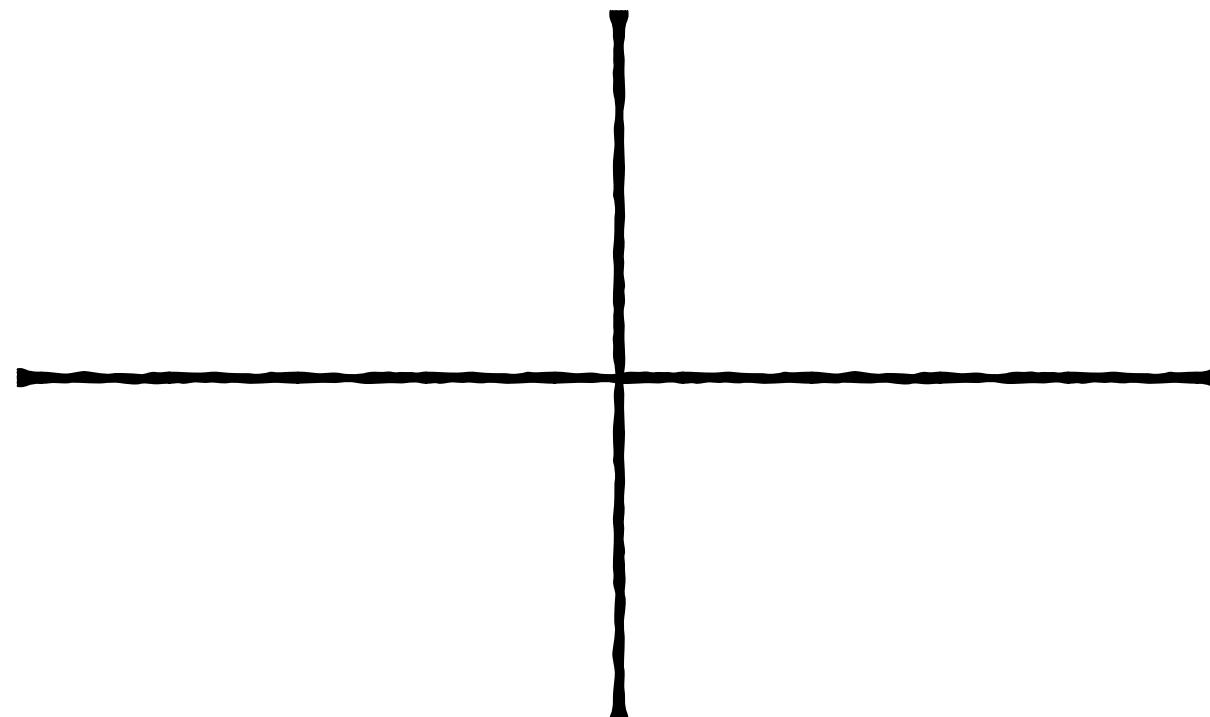
```
/**  
 * Tests that a package dependency cycle does not  
 * exist for any of the analyzed packages.  
 */  
public void testAllPackages() {  
  
    Collection packages = jdepend.analyze();  
  
    assertEquals("Cycles exist",  
                false, jdepend.containsCycles());  
}
```

Fitness Function

atomic



holistic



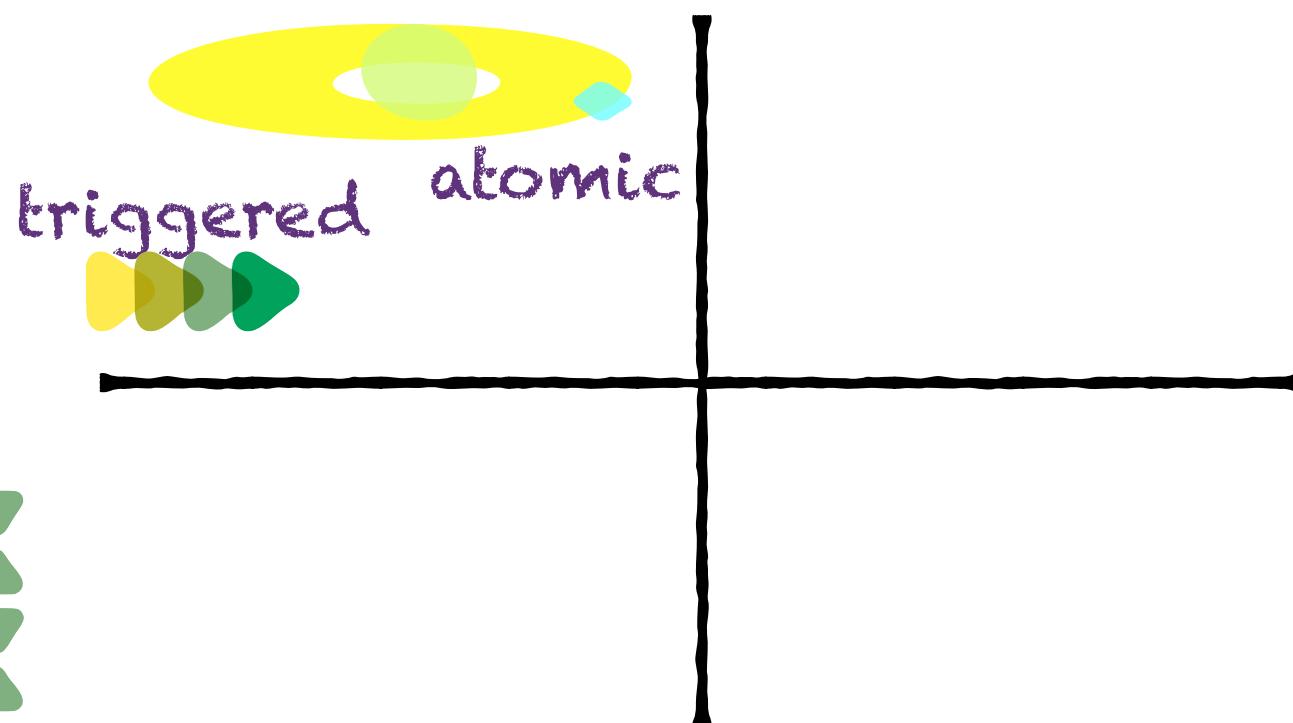
triggered



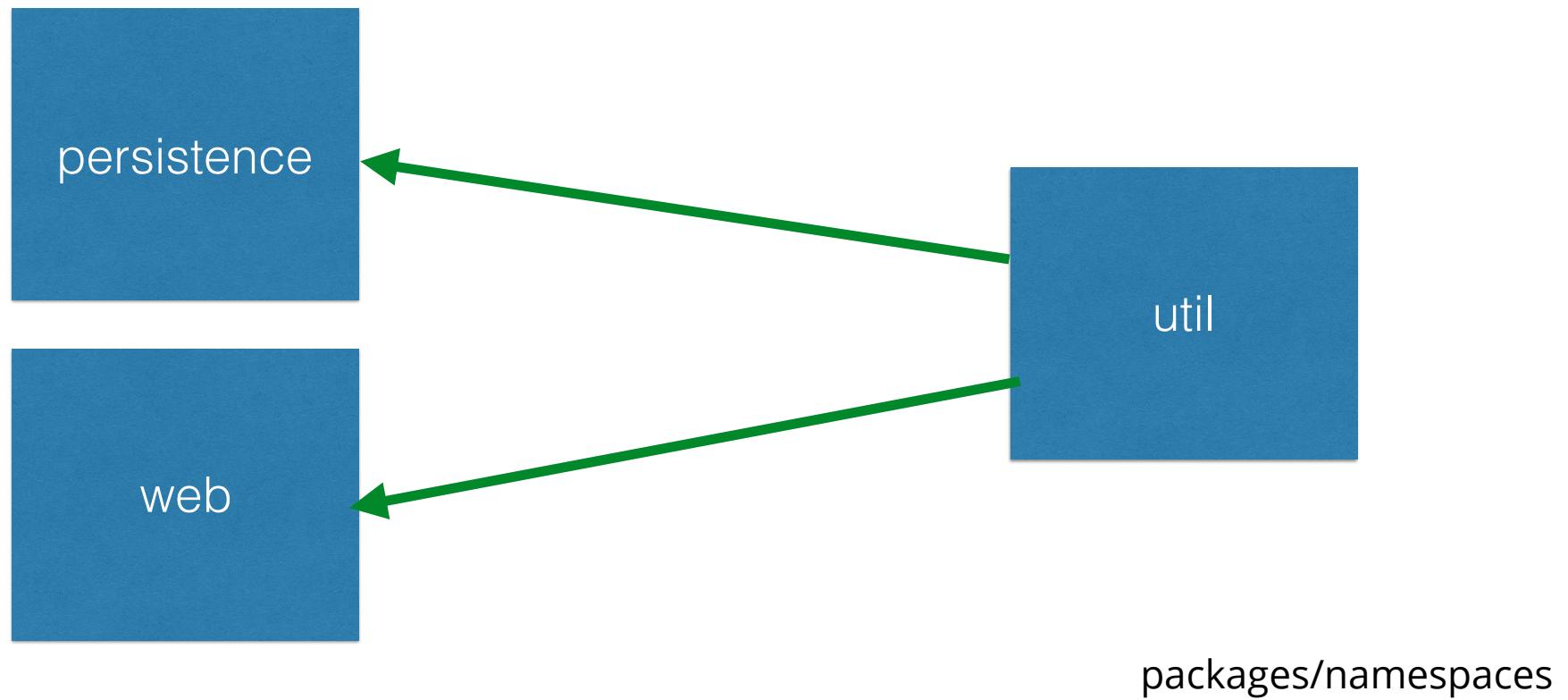
continuous



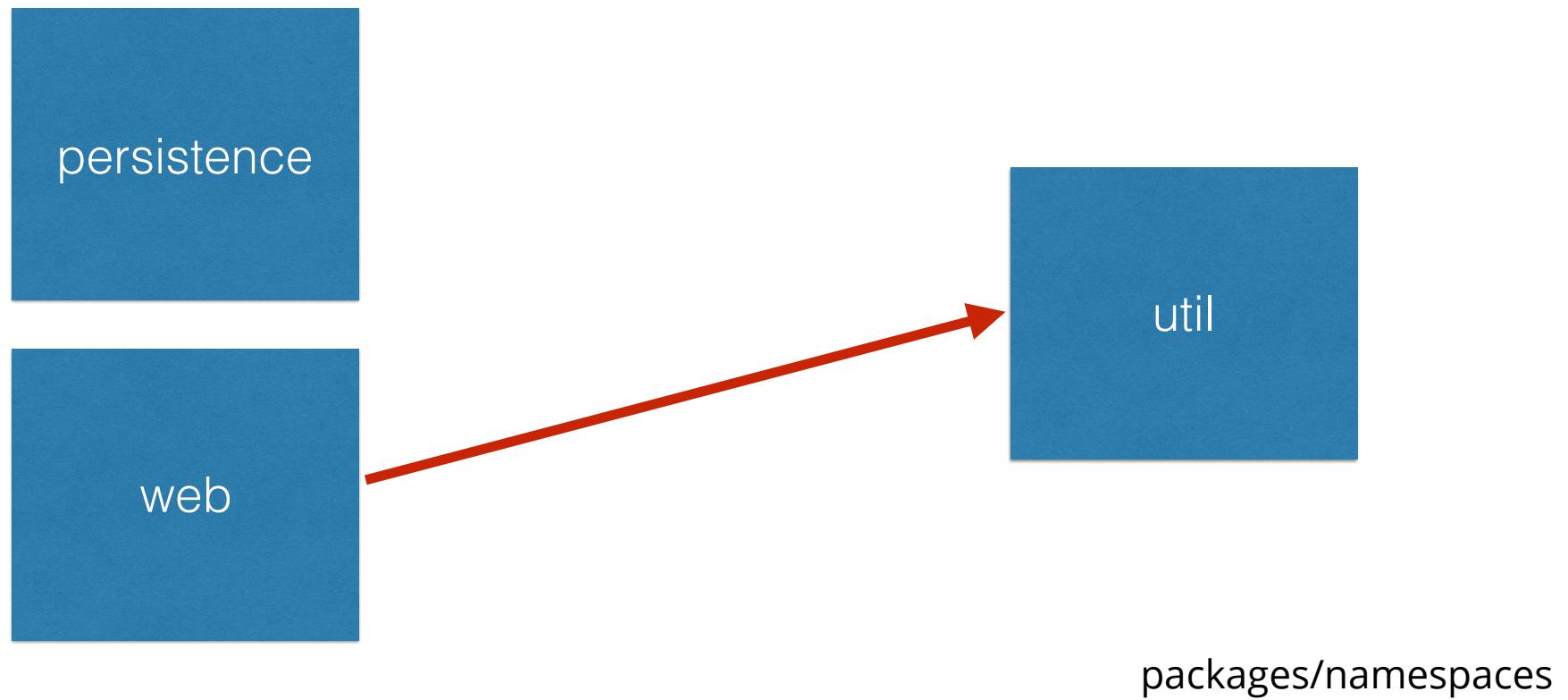
Fitness Function



Directionality of Imports



Directionality of Imports



Coupling Fitness Function

```
public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();

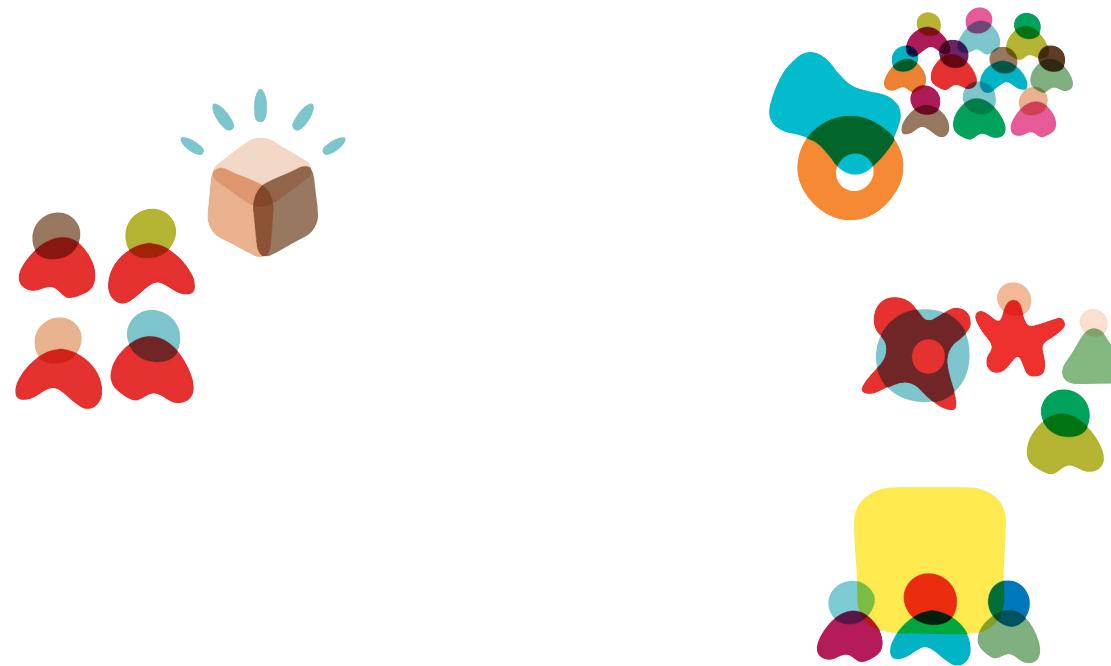
    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");
    JavaPackage web = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    persistence.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

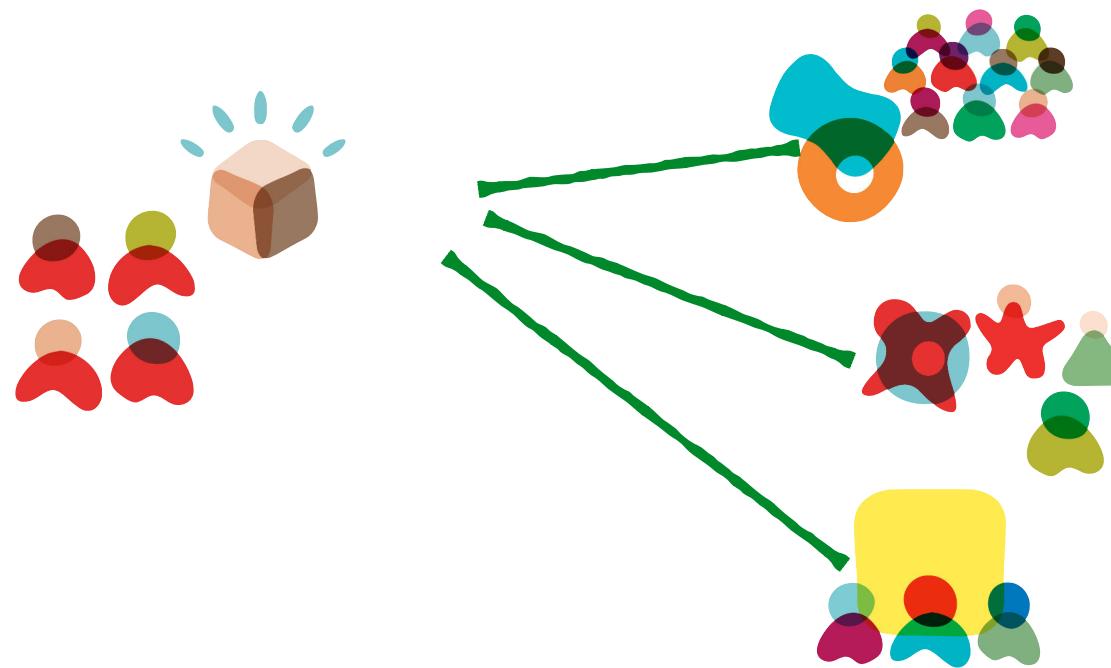
    assertEquals("Dependency mismatch",
                true, jdepend.dependencyMatch(constraint));
}
```

Consumer Driven Contracts



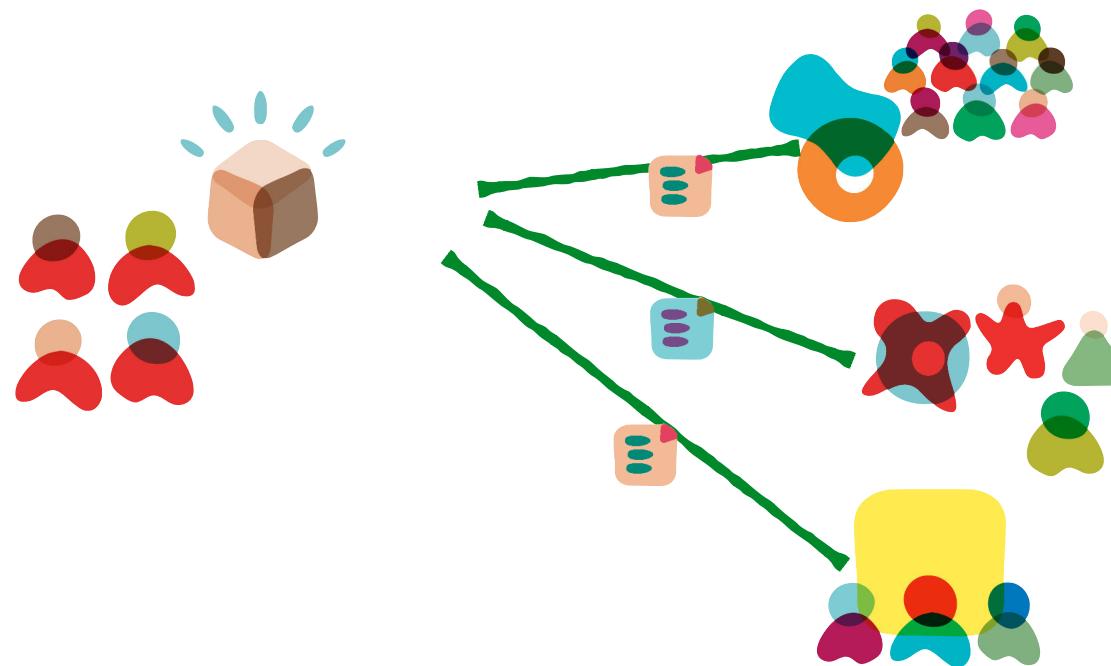
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



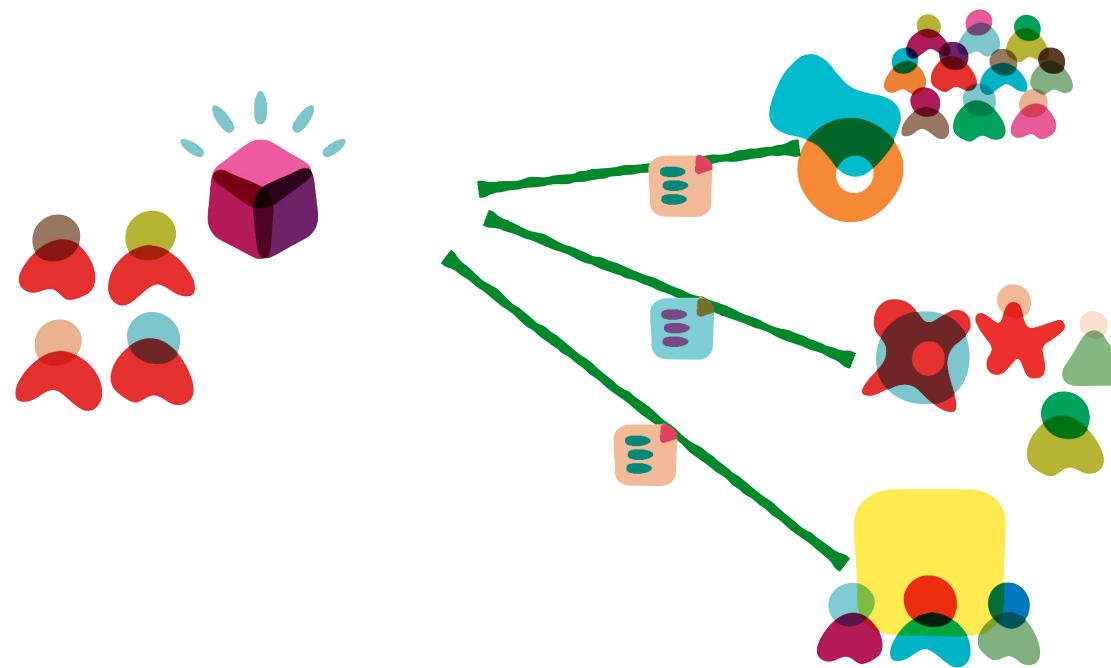
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



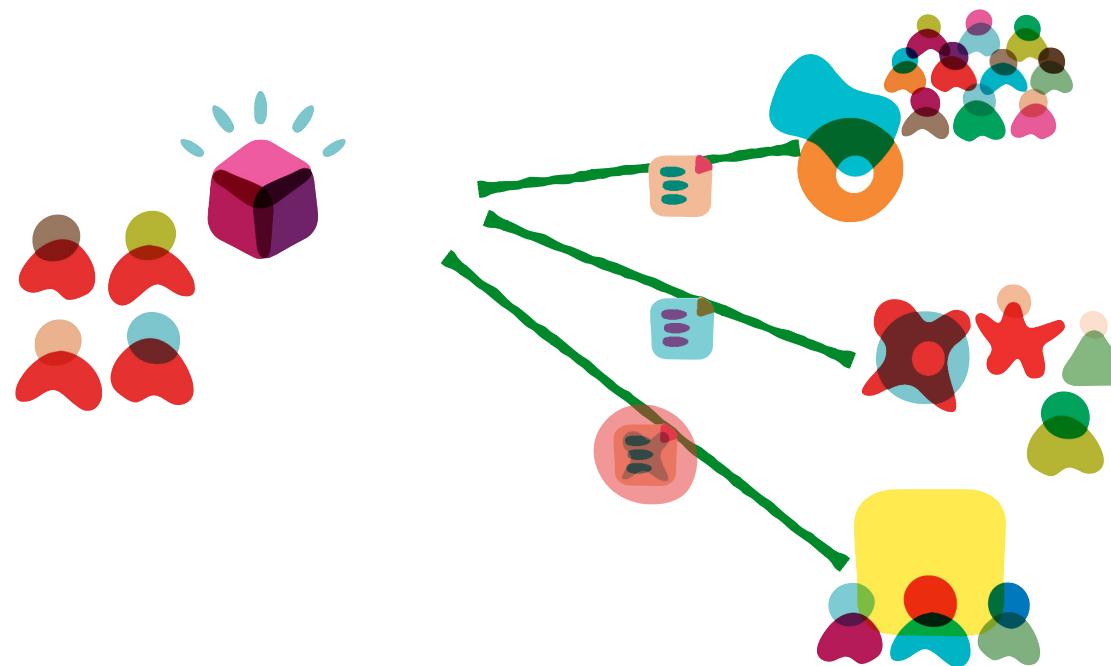
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



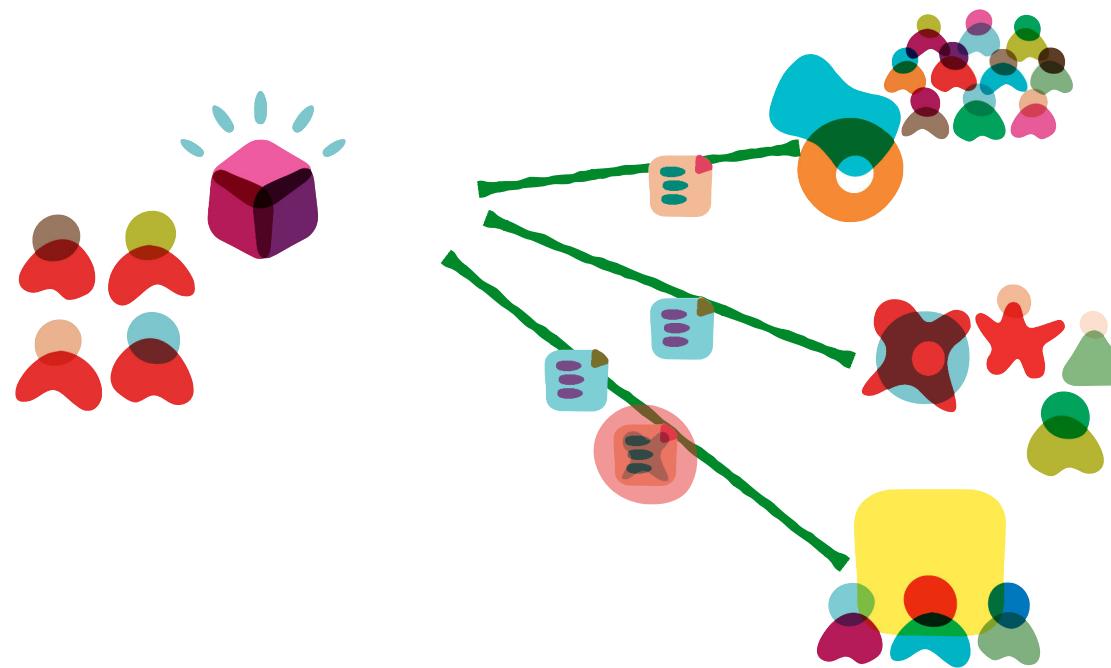
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



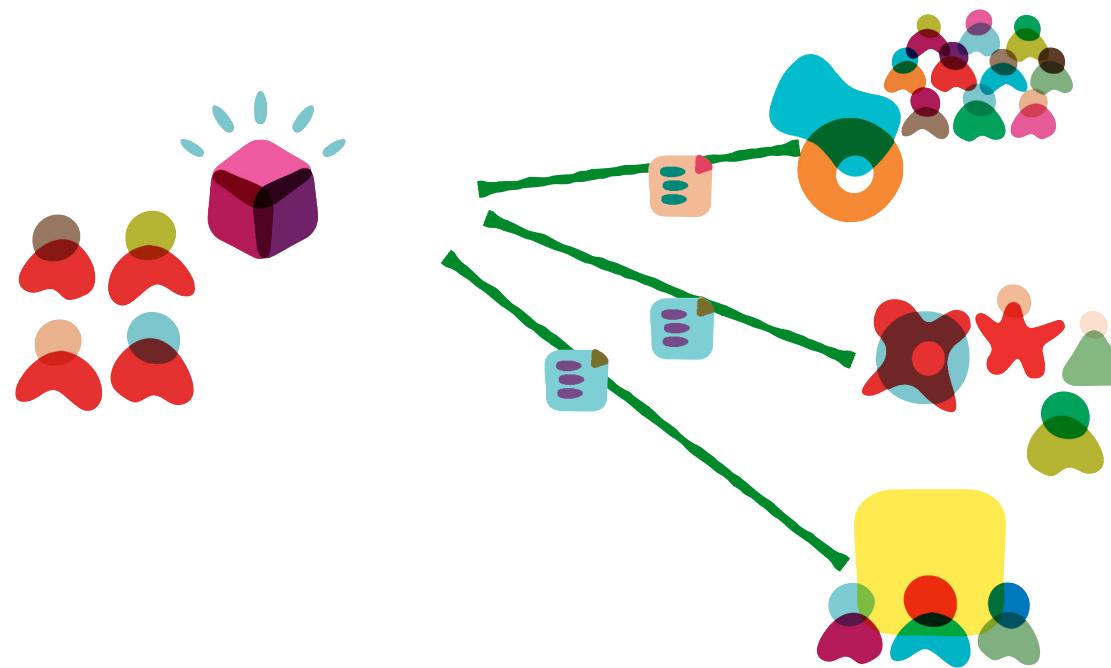
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



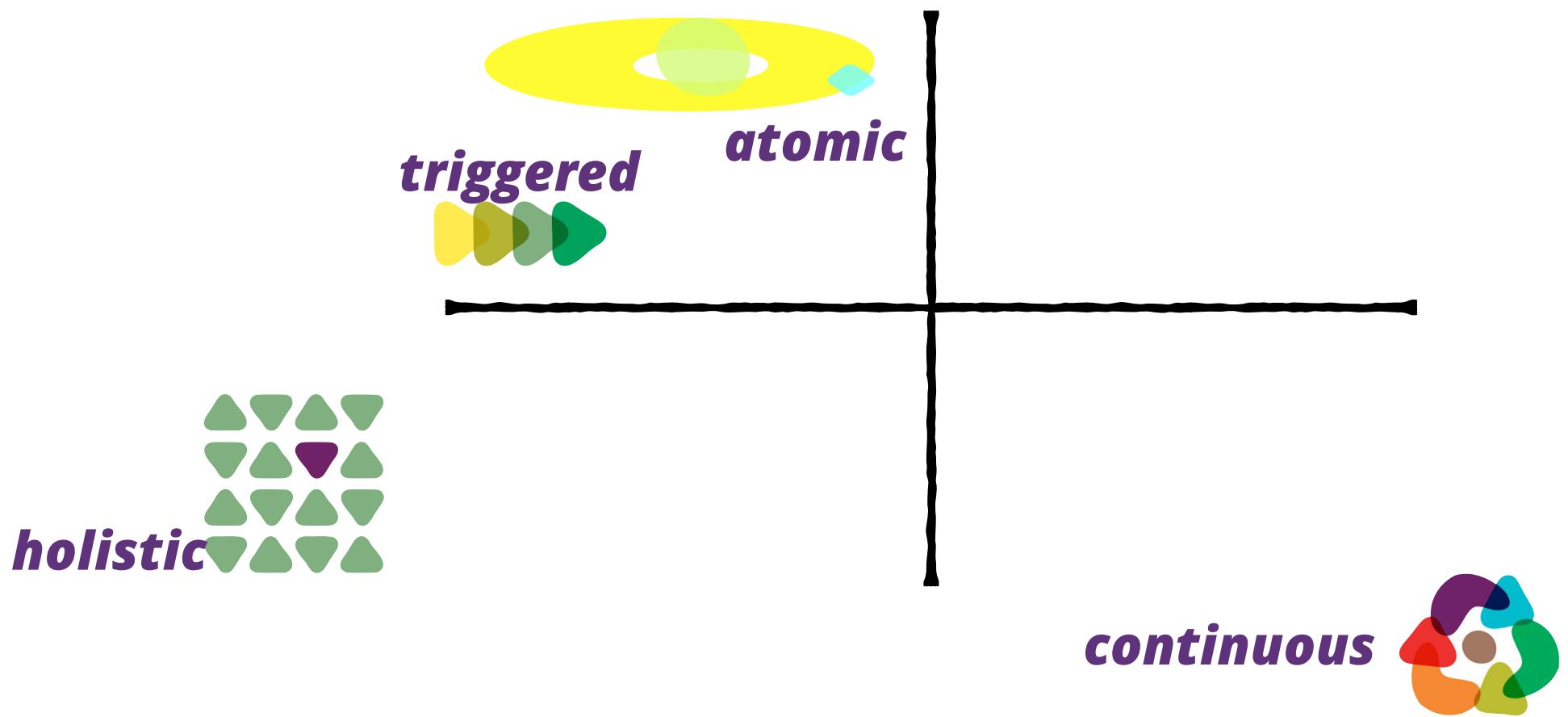
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts

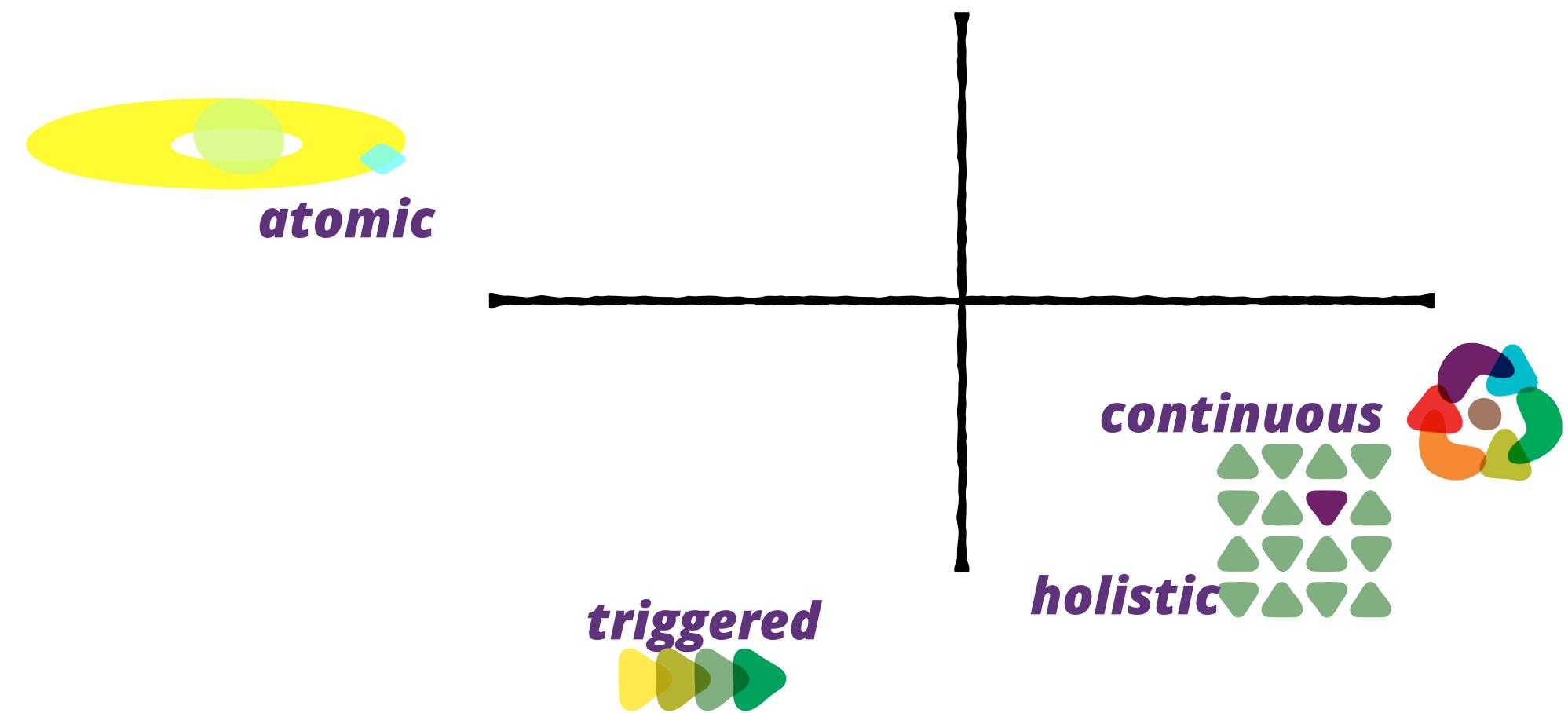


martinfowler.com/articles/consumerDrivenContracts.html

Fitness Function



Fitness Function





amazon
web services





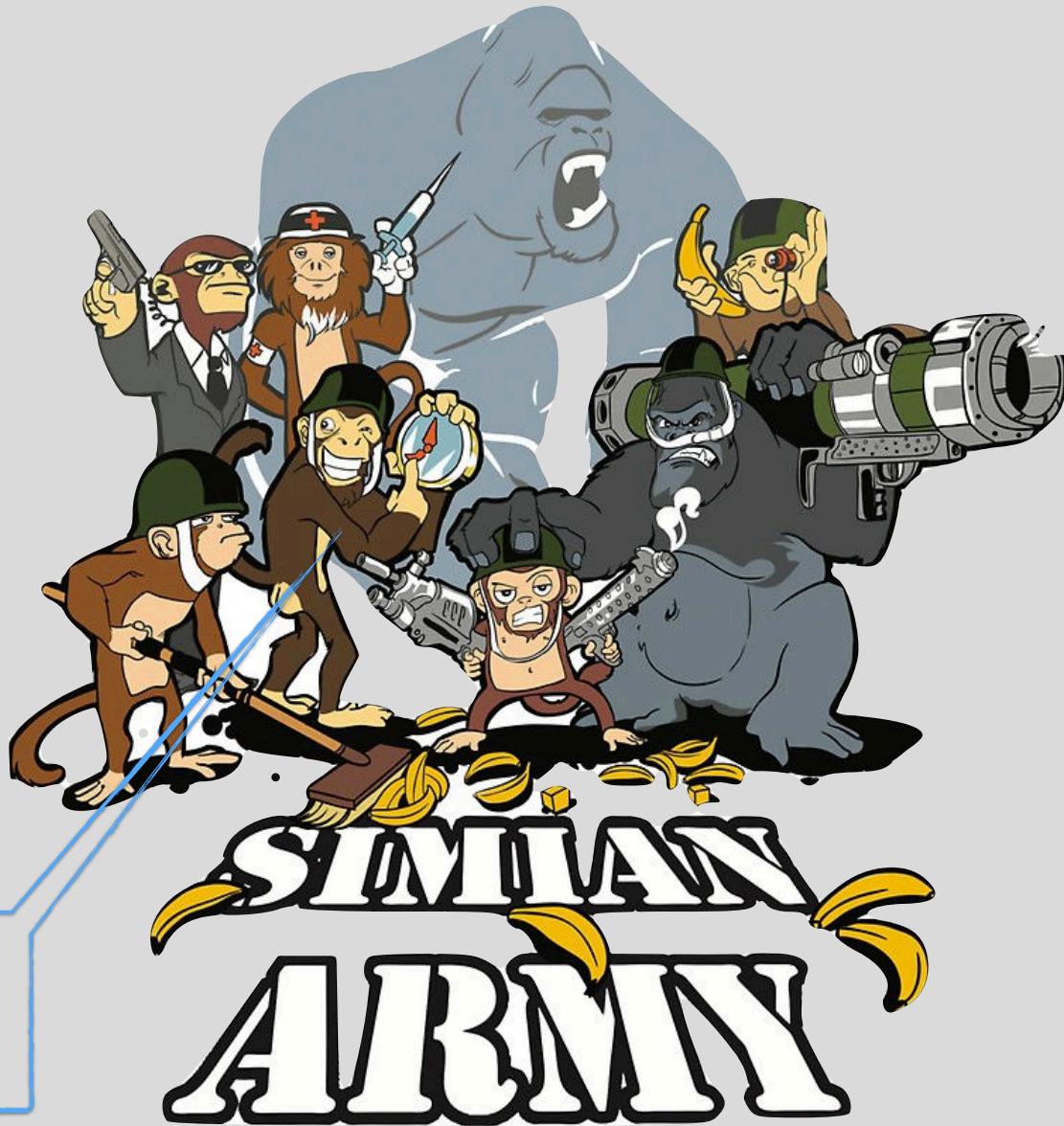
chaos monkey

SIMIAN ARMY



SIMIAN ARMY

chaos
gorilla



Latency
monkey

doctor
monkey



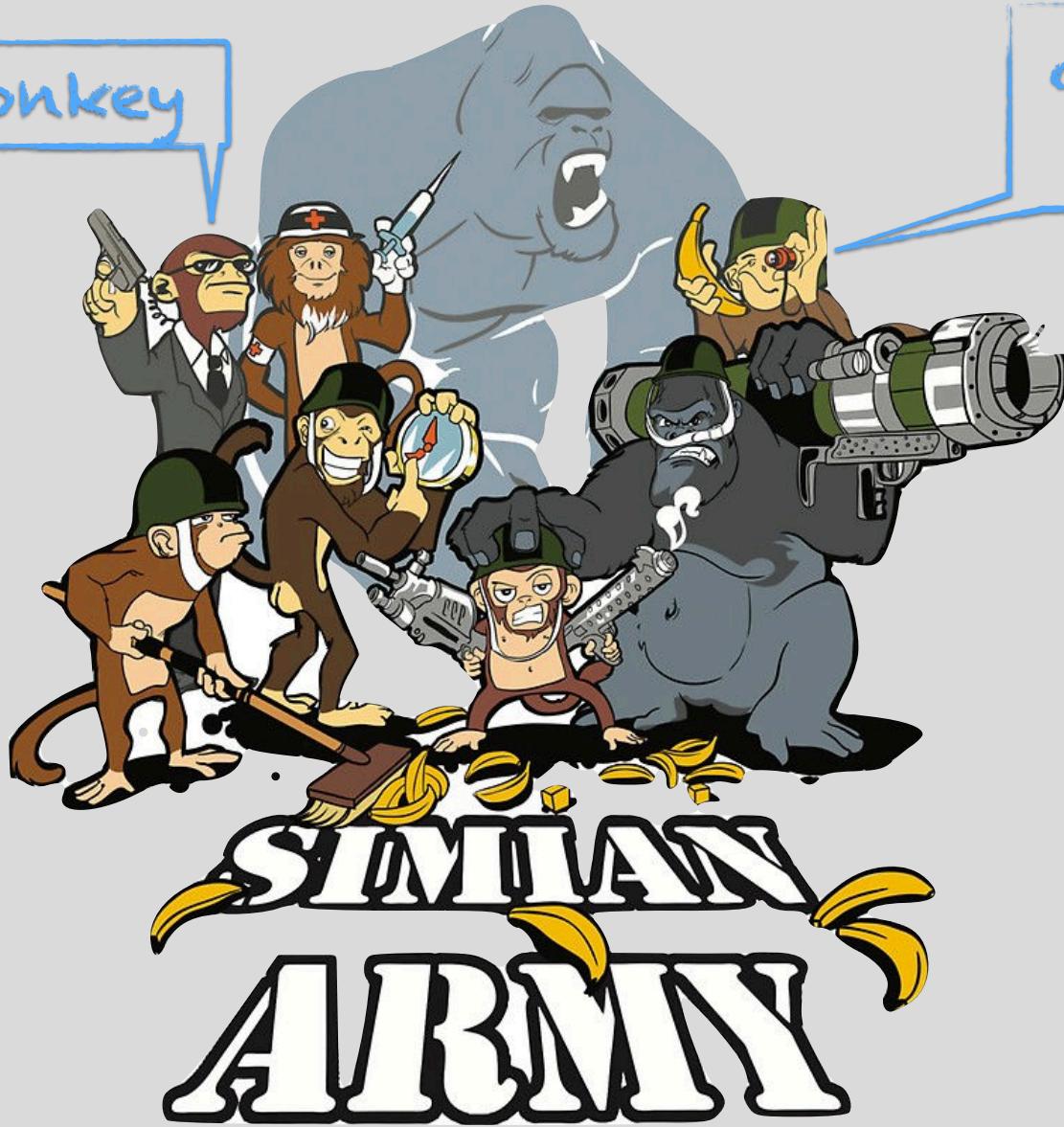


janitor
monkey



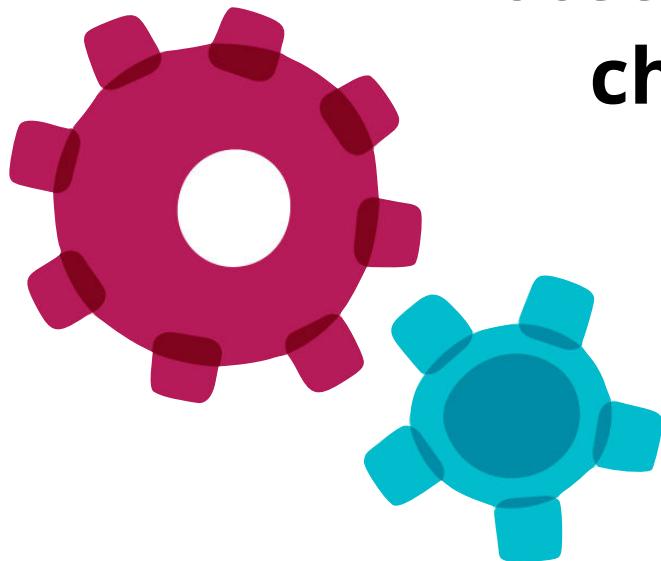
security monkey

conformity
monkey



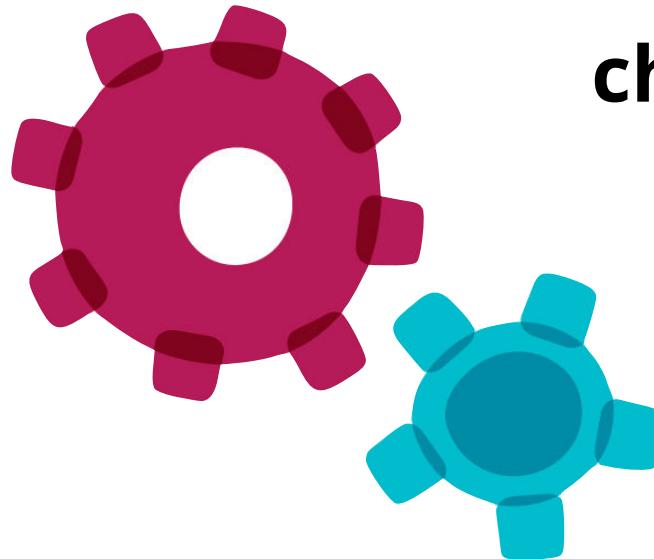
Implementing Fitness Functions

**Protecting architectural
characteristics**

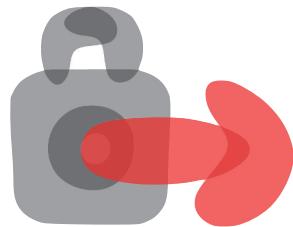


Implementing Fitness Functions

Protecting architectural
characteristics



Automating governance



maintainable?

maintainable?

Cyclomatic complexity < 50 for all projects

maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

maintainable?

(incoming/outgoing)

Controlled afferent/efferent coupling

Cyclomatic complexity < 50 for all projects

Naming conventions

immutability

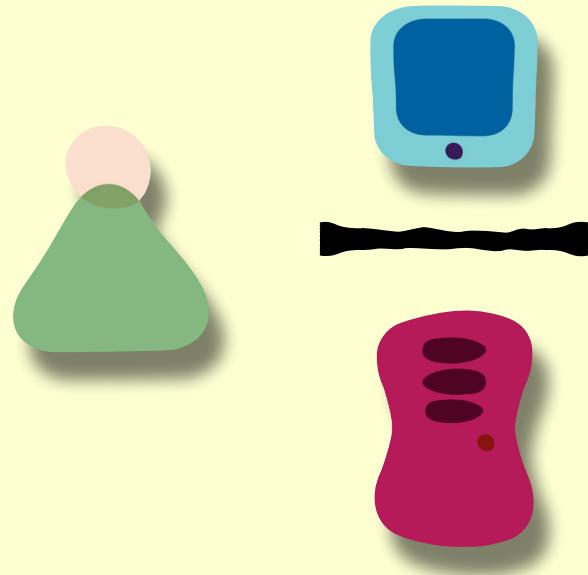
maintainable?

(incoming/outgoing)

Controlled afferent/efferent coupling

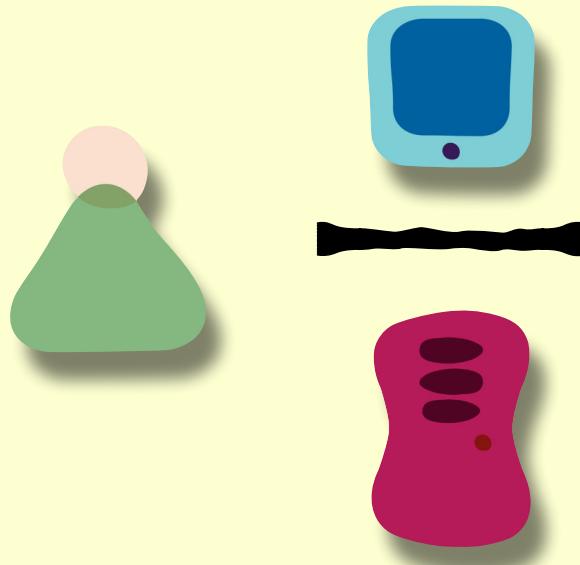


Governing Code Quality





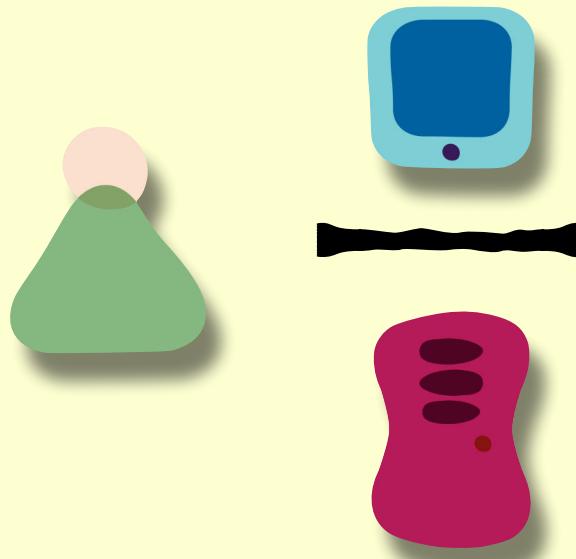
Governing Code Quality



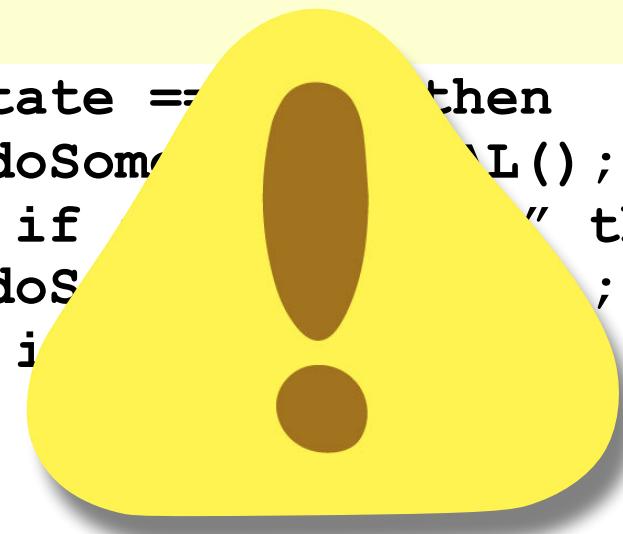
```
if state == "AL" then  
    doSomethingForAL();  
else if state == "GA" then  
    doSomethingForGA();  
else if ...
```



Governing Code Quality

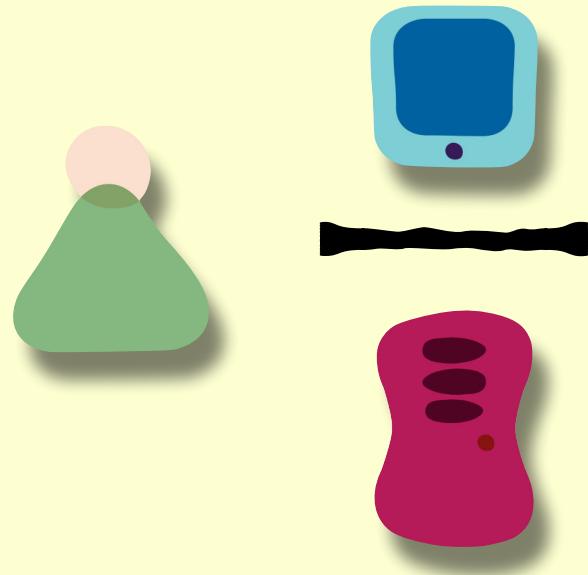


```
if state == "A" then
    doSomethingA();
else if state == "B" then
    doSomethingB();
else if state == "C" then
    doSomethingC();
```





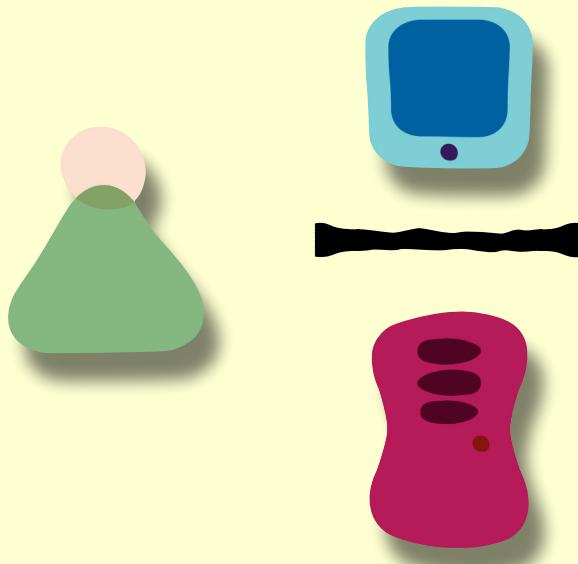
Governing Code Quality





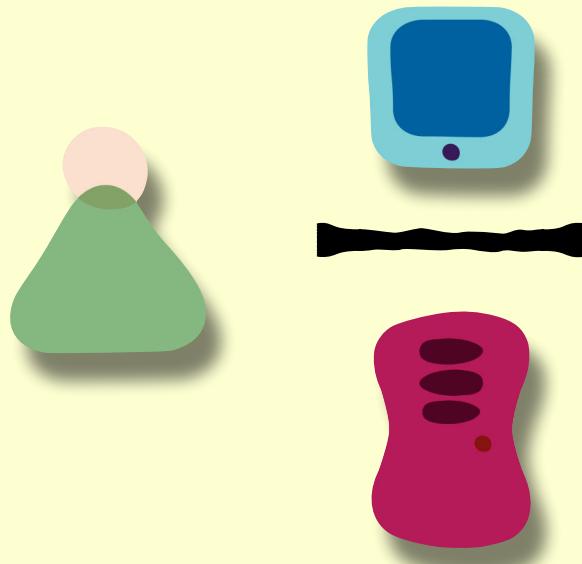
Governing Code Quality

```
if state == "AL" then  
    doSomethingForAL();  
else if state == "GA" then  
    doSomethingForGA();  
else if ...
```

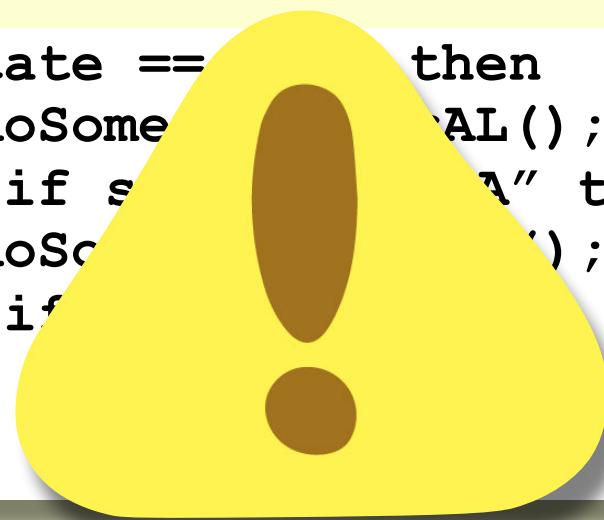




Governing Code Quality

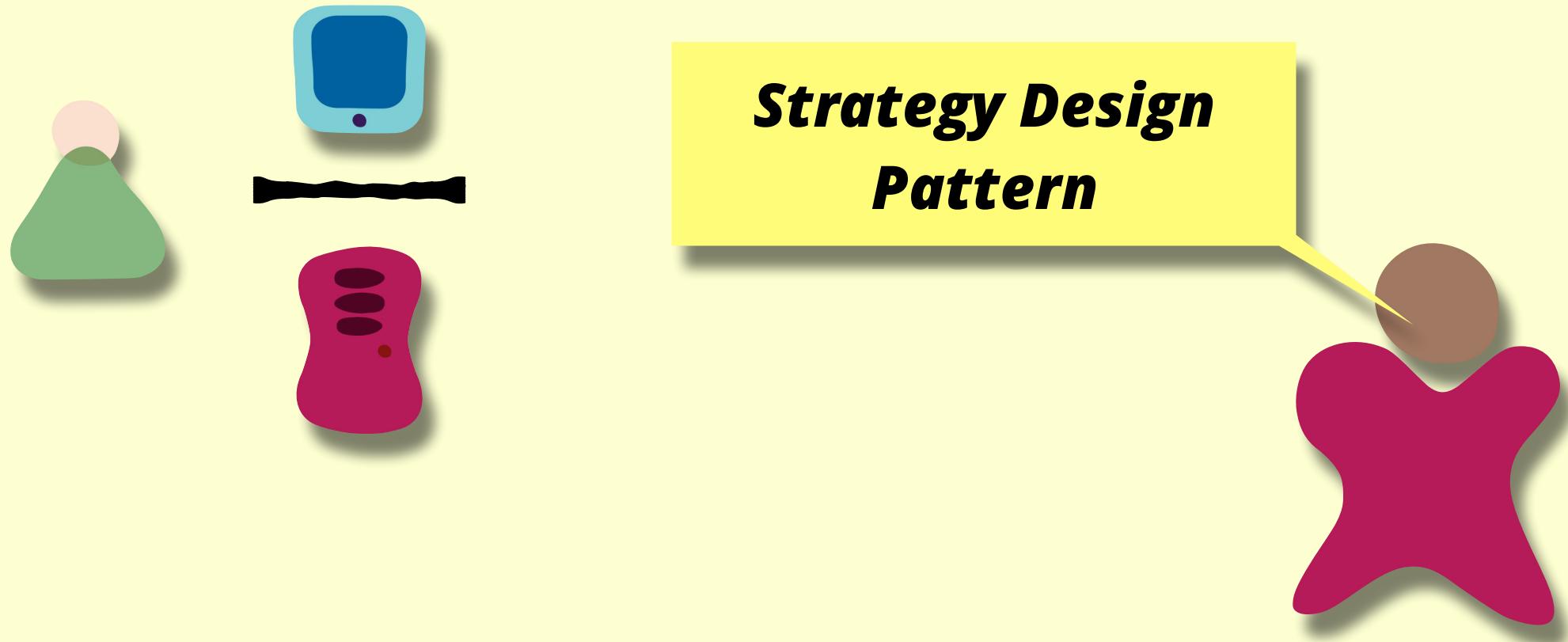


```
if state == "A" then  
    doSomethingA();  
else if state == "B" then  
    doSomethingB();  
else if state == "C" then  
    doSomethingC();  
else if state == "D" then  
    doSomethingD();  
else if state == "E" then  
    doSomethingE();  
else if state == "F" then  
    doSomethingF();  
else if state == "G" then  
    doSomethingG();  
else if state == "H" then  
    doSomethingH();  
else if state == "I" then  
    doSomethingI();  
else if state == "J" then  
    doSomethingJ();  
else if state == "K" then  
    doSomethingK();  
else if state == "L" then  
    doSomethingL();  
else if state == "M" then  
    doSomethingM();  
else if state == "N" then  
    doSomethingN();  
else if state == "O" then  
    doSomethingO();  
else if state == "P" then  
    doSomethingP();  
else if state == "Q" then  
    doSomethingQ();  
else if state == "R" then  
    doSomethingR();  
else if state == "S" then  
    doSomethingS();  
else if state == "T" then  
    doSomethingT();  
else if state == "U" then  
    doSomethingU();  
else if state == "V" then  
    doSomethingV();  
else if state == "W" then  
    doSomethingW();  
else if state == "X" then  
    doSomethingX();  
else if state == "Y" then  
    doSomethingY();  
else if state == "Z" then  
    doSomethingZ();  
end if
```





Governing Code Quality



The screenshot shows a Safari browser window with the URL blog.jdriven.com in the address bar. The page content is as follows:

jdriven

[← Previous](#) [Next →](#)

Search

Implementing architectural fitness functions using Gradle, JUnit and code-assert

Posted on October 6, 2017 by Rob Brinkman [Tweet](#)

Architectural fitness functions

Inspired by Neal Ford's presentation at our [Change is the Only constant event](#) I started experimenting with architectural fitness functions. An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

If you want to take a deeper dive into evolutionary architectures including fitness functions take look at Neals book: [Building Evolutionary Architectures: Support Constant Change](#).

Neal's [slides](#) contained an example of verifying package dependencies from a Unit Test using [JDepend](#).

Verifying code modularity

In this blog post we'll elaborate on that approach and create a Unit Test that verifies that our code complies to the chosen packaging strategies using an

JDriven
blog.jdriven.com
www.jdriven.com

Featured Posts

- Spring Sweets: Add (Extra)
- Build Information To Info
- Endpoint
- Angular2 and Spring Boot: Getting Started
- Het ontstaan van de passie voor het moderne maken
- Securing your application landscape with Spring Cloud
- Security – Part 1
- Spicy Spring : Dynamically create your own BeanDefinition

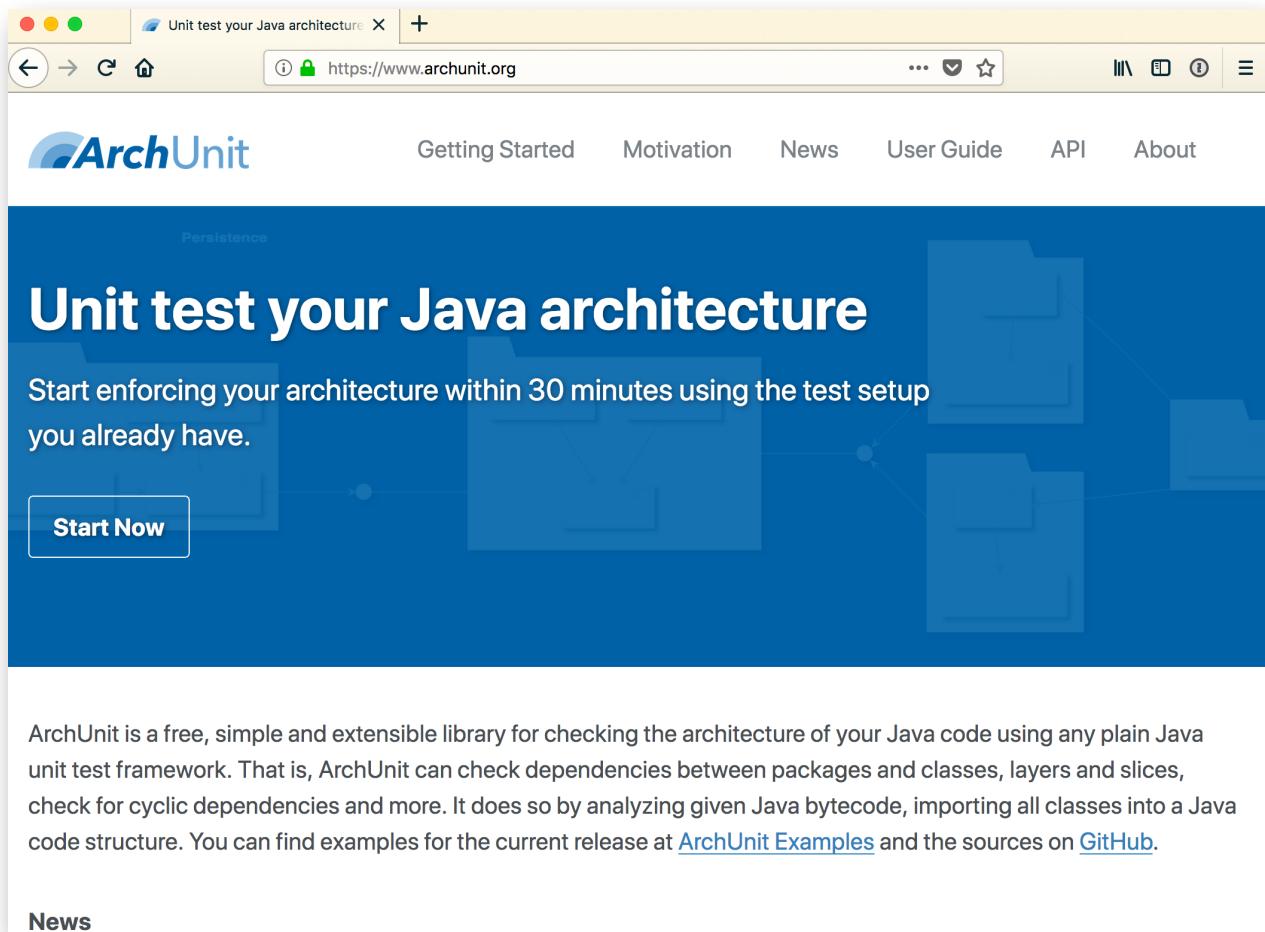
Recent Posts

- Awesome Ascidoctor: Grouping Floating Images
- Awesome Ascidoctor: Using Tab Separated Data In A Table
- Implementing architectural fitness functions using Gradle, JUnit and code-assert
- 6 Steps to help you debug your application

<https://blog.jdriven.com/2017/10/implementing-architectural-fitness-functions-using-gradle-junit-code-assert/>

ArchUnit

<https://www.archunit.org/>



The screenshot shows the homepage of the ArchUnit website. At the top, there's a navigation bar with links for "Getting Started", "Motivation", "News", "User Guide", "API", and "About". Below the navigation, a large banner features the text "Unit test your Java architecture" and a subtext: "Start enforcing your architecture within 30 minutes using the test setup you already have." A prominent "Start Now" button is located on the left side of the banner. The background of the banner is a blue-toned diagram of interconnected code structures. At the bottom of the page, there's a "News" section.

Persistence

Unit test your Java architecture

Start enforcing your architecture within 30 minutes using the test setup you already have.

Start Now

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code using any plain Java unit test framework. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. You can find examples for the current release at [ArchUnit Examples](#) and the sources on [GitHub](#).

News

ArchUnit

<https://www.archunit.org/>

coding rules

```
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

public class CodingRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void classes_should_not_access_standard_streams_defined_by_hand() {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @Test
    public void classes_should_not_access_standard_streams_from_library() {
        NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
    }

    @Test
    public void classes_should_not_throw_generic_exceptions() {
        NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
    }

    @Test
    public void classes_should_not_use_java_util_logging() {
        NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
    }
}
```

ArchUnit

@Test

<https://www.archunit.org/>

```
public void classes_should_not_access_standard_streams_from_library() {  
    NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);  
}
```

@Test

```
public void classes_should_not_throw_generic_exceptions() {  
    NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);  
}
```

@Test

```
public void classes_should_not_use_java_util_logging() {  
    NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);  
}
```

coding rules

ArchUnit

<https://www.archunit.org/>

```
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
            SomeBusinessInterface.class,
            SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
            SomeBusinessInterface.class,
            SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);
    }

    @Test
    public void interfaces_must_not_be_placed_in_implementation_packages() {
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);

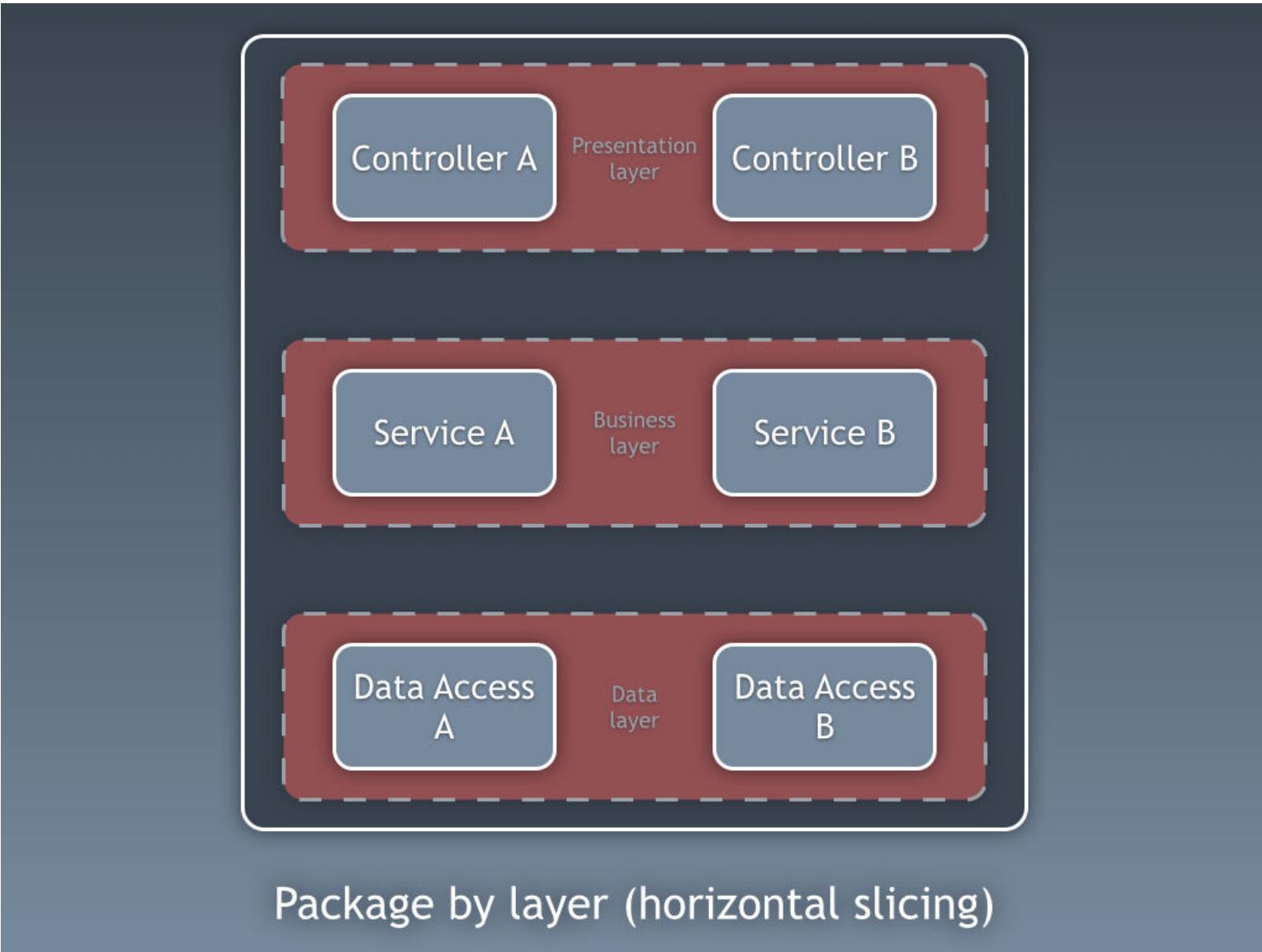
        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);
    }
}
```

interface rules

ArchUnit

<https://www.archunit.org/>

```
public class InterfaceRules {  
  
    @Test  
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {  
        JavaClasses classes = new ClassFileImporter().importClasses(  
            SomeBusinessInterface.class,  
            SomeDao.class  
        );  
  
        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface")  
    }  
  
    @Test  
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word
```



ArchUnit

<https://www.archunit.org/>

```
public class LayerDependencyRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void services_should_not_access_controllers() {
        noClasses().that().resideInAPackage("..service..")
            .should().accessClassesThat().resideInAPackage("..controller..").check(classes);
    }

    @Test
    public void persistence_should_not_access_services() {
        noClasses().that().resideInAPackage("..persistence..")
            .should().accessClassesThat().resideInAPackage("..service..").check(classes);
    }

    @Test
    public void services_should_only_be_accessed_by_controllers_or_other_services() {
        classes().that().resideInAPackage("..service..")
            .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);
    }
}
```

layer dependency

```
-----  
private JavaClasses classes;  
  
@Before  
public void setUp() throws Exception {  
    classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);  
}  
  
@Test  
public void services_should_not_access_controllers() {  
    noClasses().that().resideInAPackage("..service..")  
        .should().accessClassesThat().resideInAPackage("..controller..").check(classes);  
}  
  
@Test  
public void persistence_should_not_access_services() {  
    noClasses().that().resideInAPackage("..persistence..")  
        .should().accessClassesThat().resideInAPackage("..service..").check(classes);  
}
```

ArchUnit

<https://www.archunit.org/>

```
@Test
public void third_party_class_should_only_be_instantiated_via_workaround() {
    classes().should(notCreateProblematicClassesOutsideOfWorkaroundFactory()
        .as(THIRD_PARTY_CLASS_RULE_TEXT))
        .check(classes);
}

private ArchCondition<JavaClass> notCreateProblematicClassesOutsideOfWorkaroundFactory() {
    DescribedPredicate<JavaCall<?>> constructorCallOfThirdPartyClass =
        target(is(constructor())).and(targetOwner(is(assignableTo(ThirdPartyClassWithProblem.class))));

    DescribedPredicate<JavaCall<?>> notFromWithinThirdPartyClass =
        originOwner(is(not(assignableTo(ThirdPartyClassWithProblem.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> notFromWorkaroundFactory =
        originOwner(is(not(equivalentTo(ThirdPartyClassWorkaroundFactory.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> targetIsIllegalConstructorOfThirdPartyClass =
        constructorCallOfThirdPartyClass.
            and(notFromWithinThirdPartyClass).
            and(notFromWorkaroundFactory);

    return never(callCodeUnitWhere(targetIsIllegalConstructorOfThirdPartyClass));
}
```

governance

Legality of Open Source Libraries



Penultima ↑


Legality of Open Source Libraries



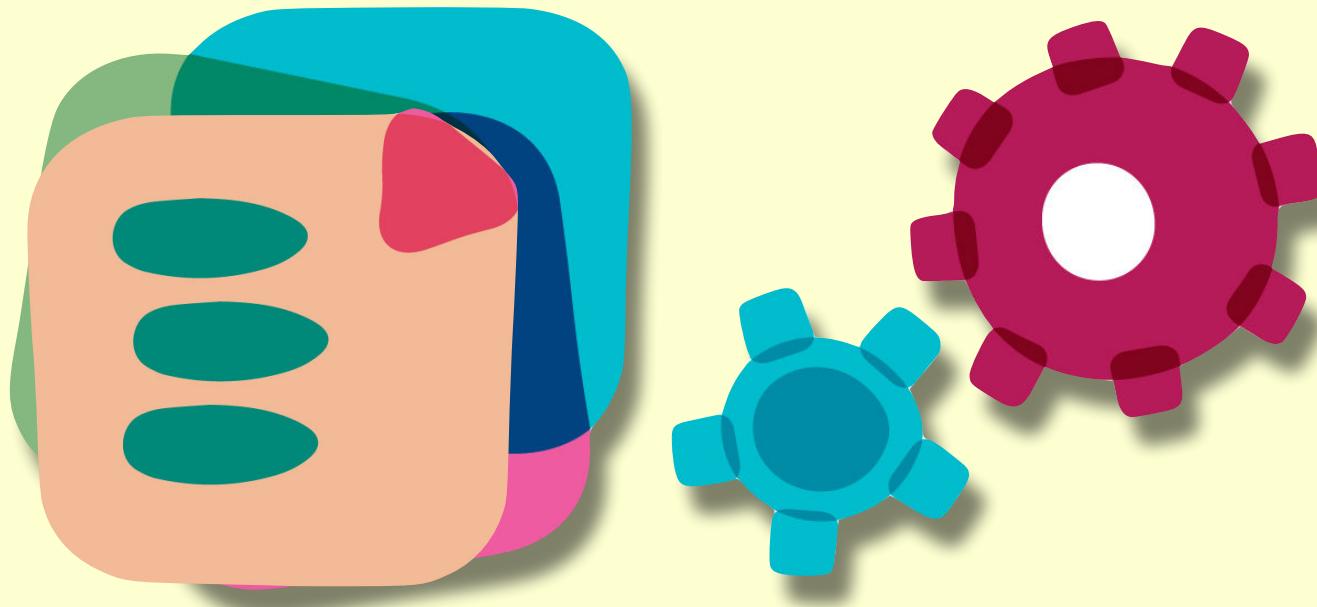
Penultima ↑
A small cluster of four interlocking gears in red, teal, and magenta colors, arranged in a semi-circle with one gear pointing upwards.

Legality of Open Source Libraries



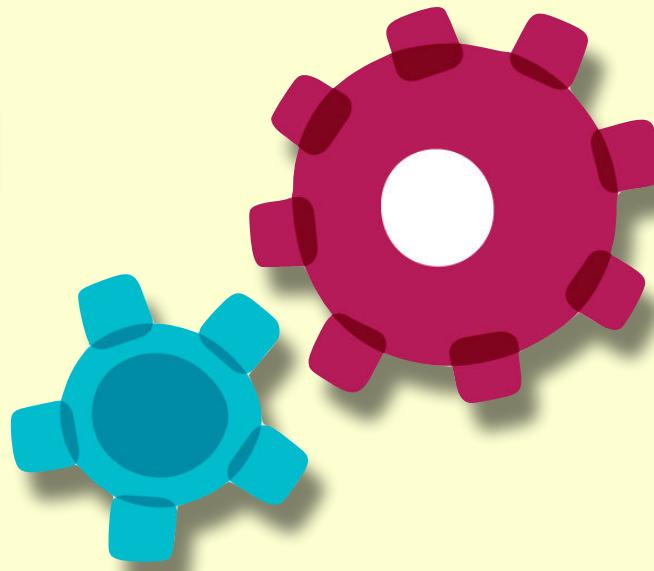
Penultima ↑


Legality of Open Source Libraries



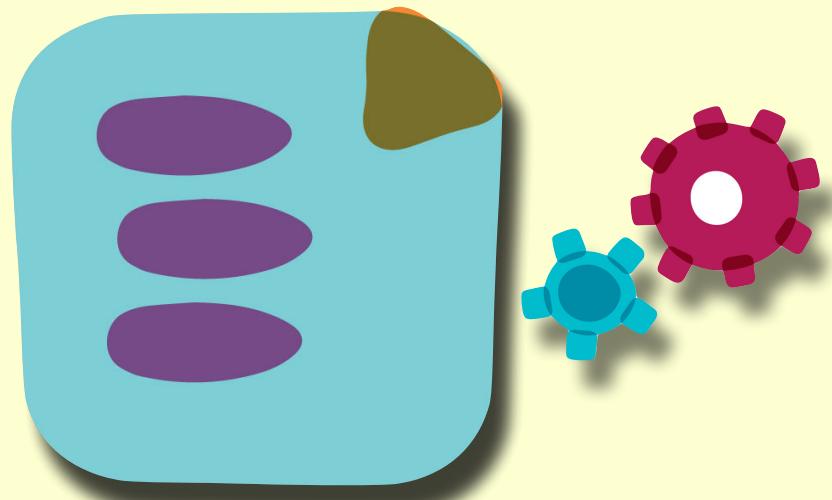
Penultima
↑
A small cluster of three gears, one blue and two red, with an upward-pointing arrow above them.

Legality of Open Source Libraries



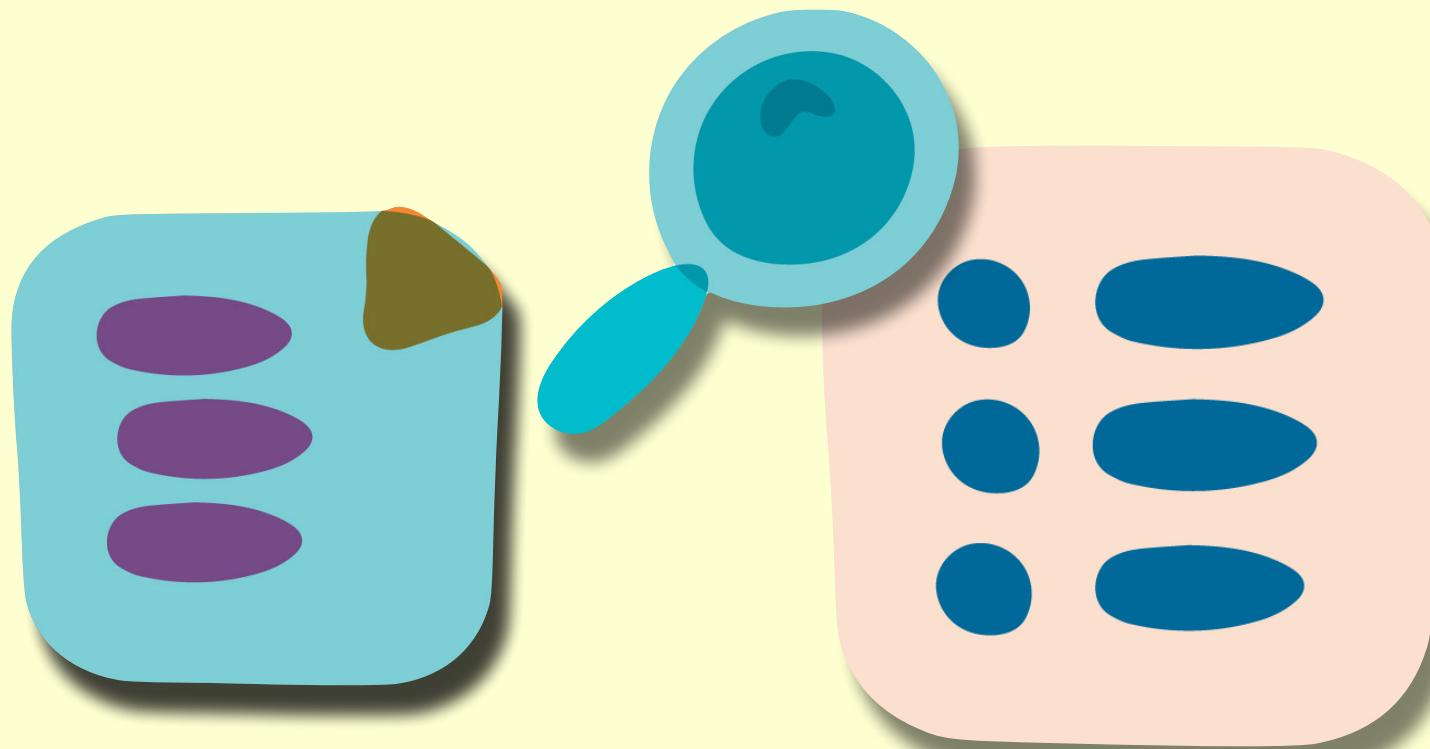
Penultima ↑


Legality of Open Source Libraries



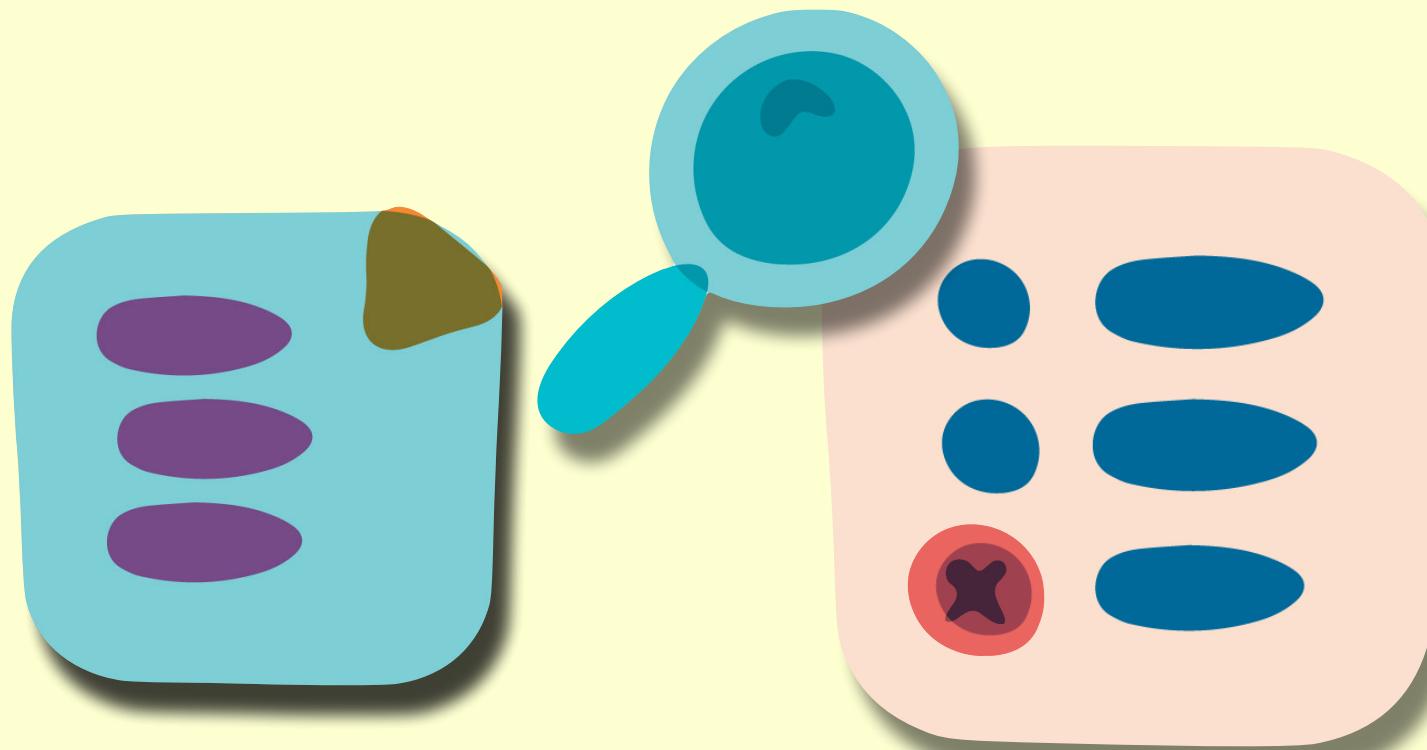
Penultimate
↑
Four small gears in a row, with the last one being red.

Legality of Open Source Libraries



Penultima ↑
A row of four small, semi-transparent gears. From left to right, they are colored red, teal, magenta, and teal. An upward-pointing arrow is positioned above the third gear.

Legality of Open Source Libraries

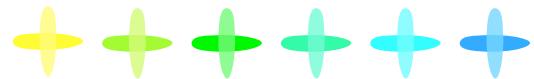


Penultima ↑
 Cyan Magenta Cyan Magenta

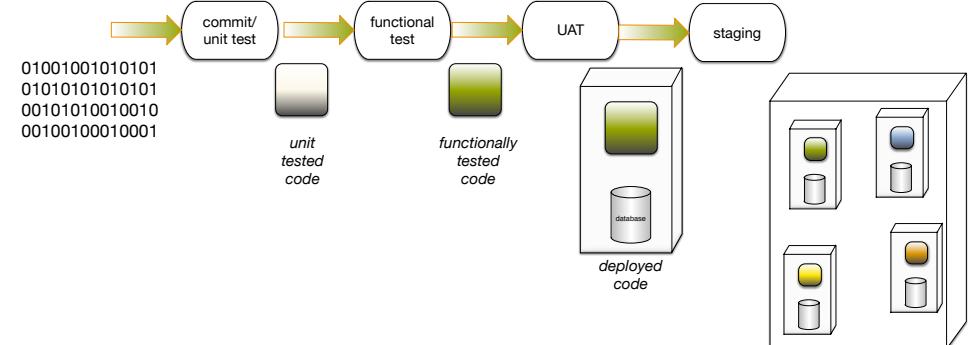
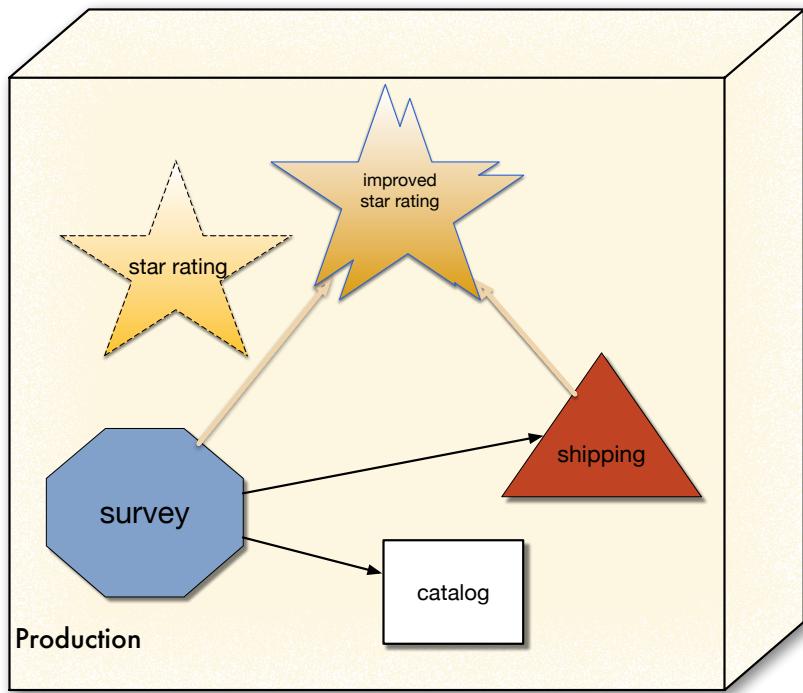
Legality of Open Source Libraries



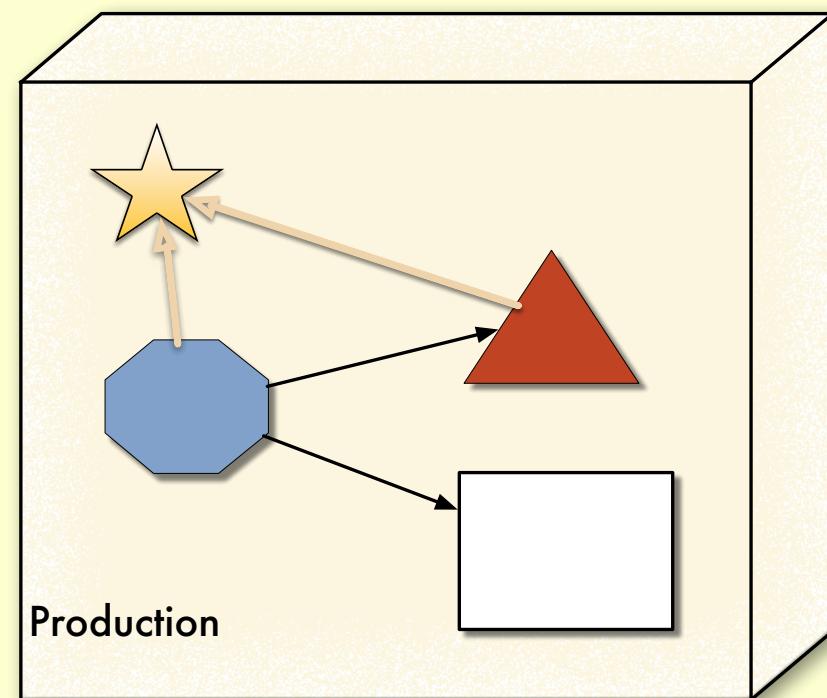
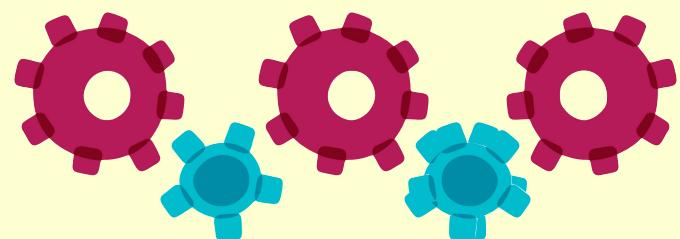
Penultima ↑
 Cyan



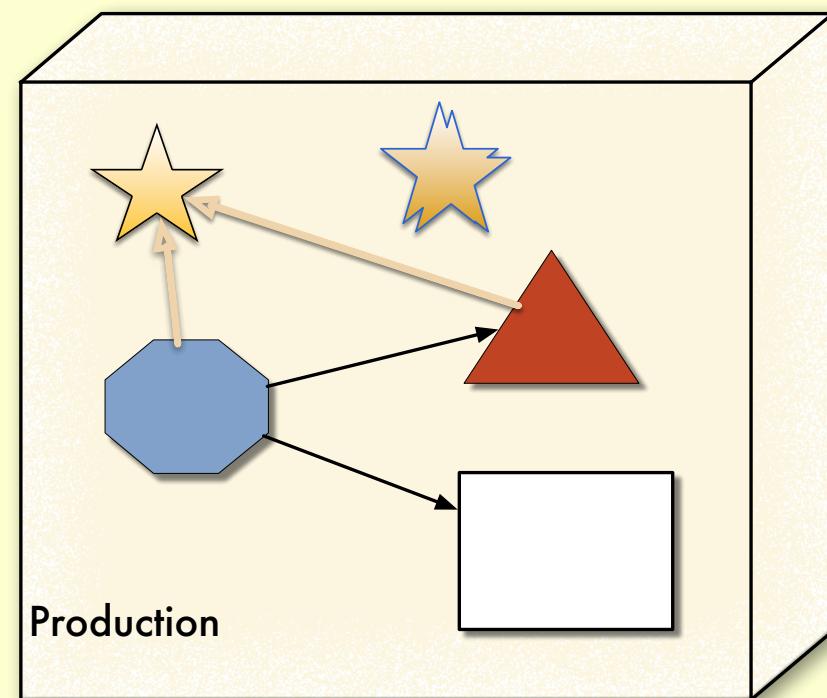
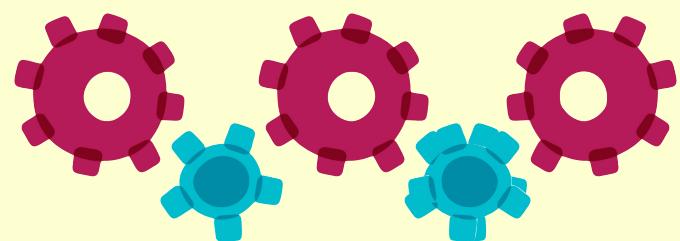
incremental



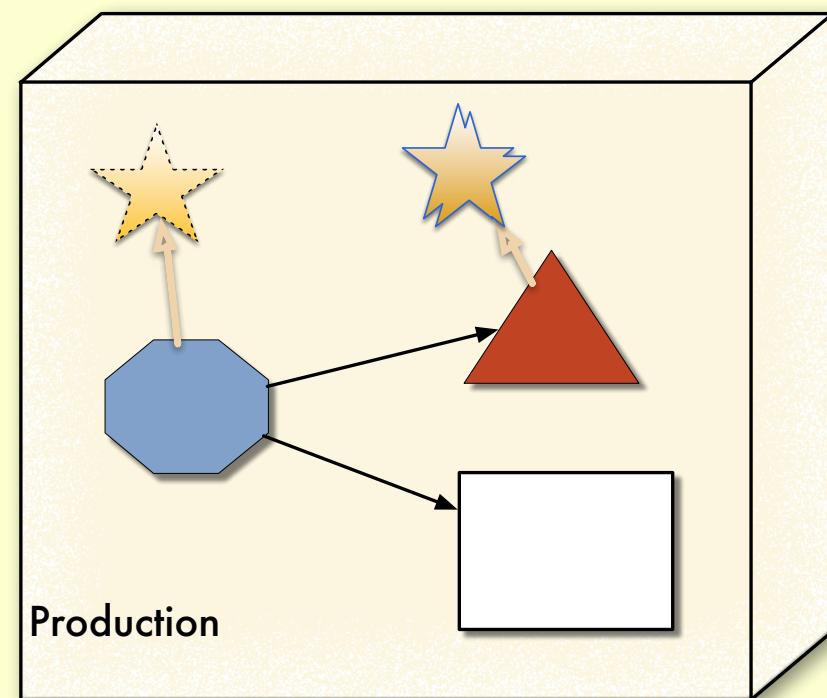
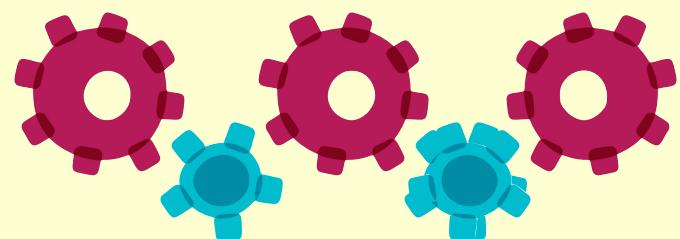
Penultima ↑ e



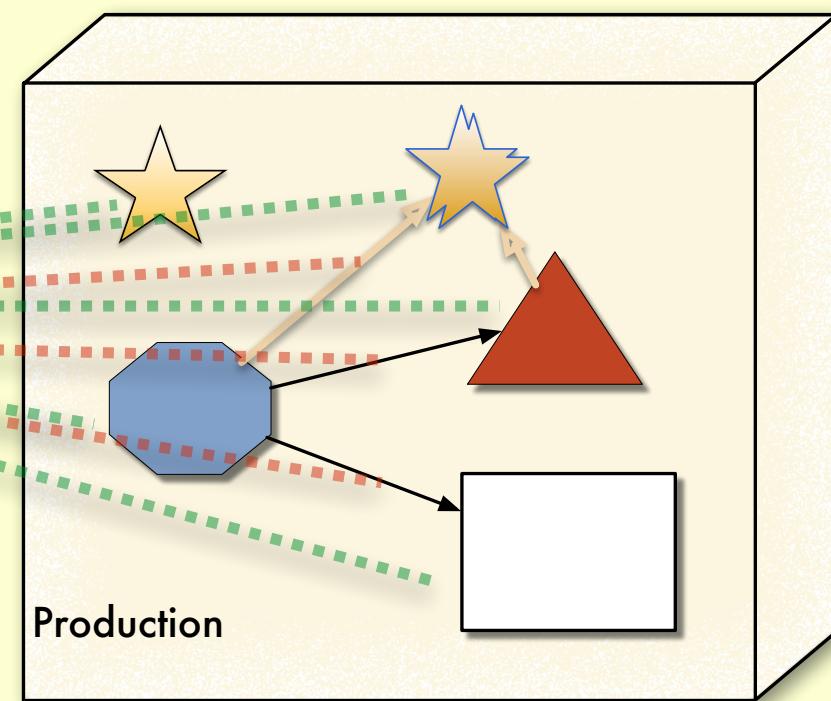
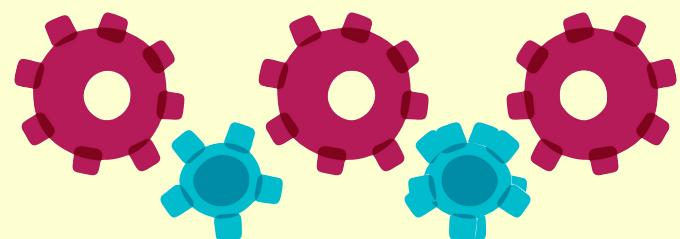
Penultima ↑ e



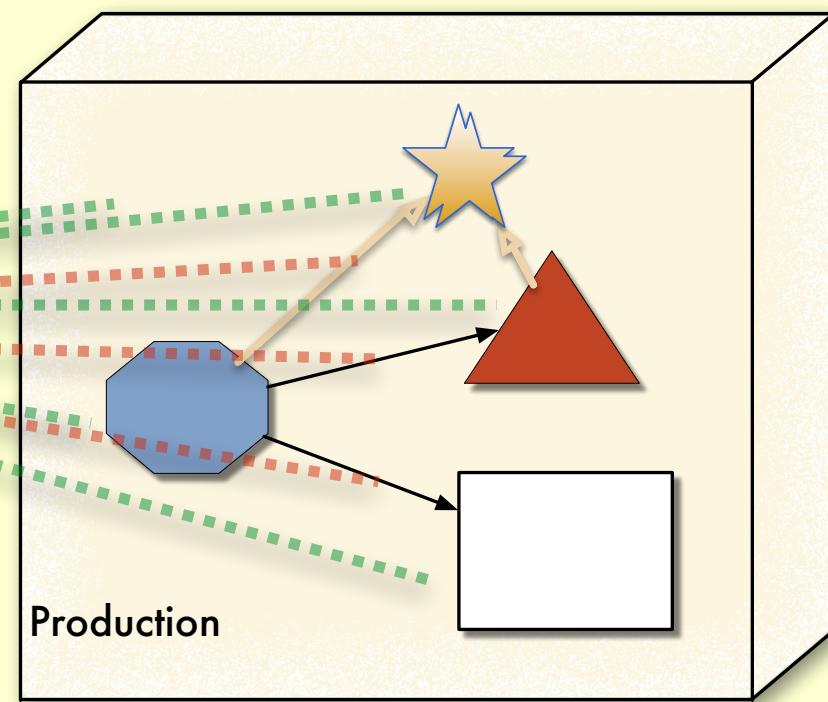
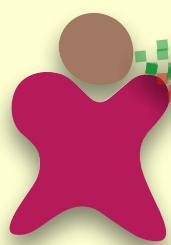
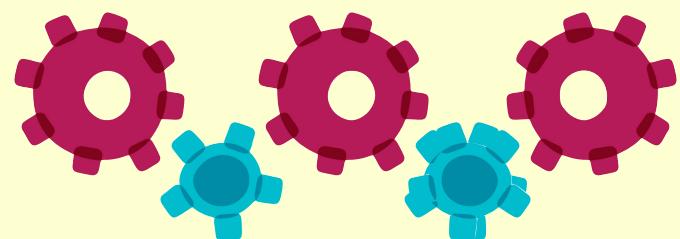
Penultima ↑ e



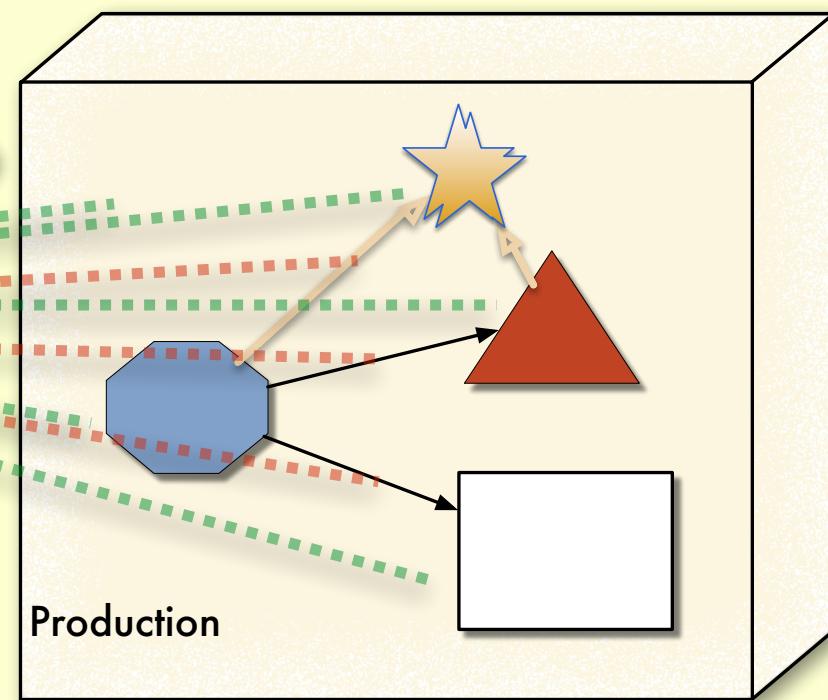
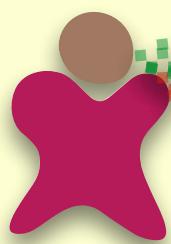
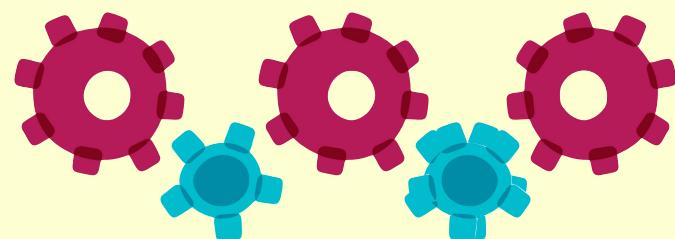
Penultima↑e



Penultima↑e



Penultima↑e



The screenshot shows a web browser displaying a blog post from GitHub Engineering. The title of the post is "Move Fast and Fix Things". The author is listed as "vmg" and the date is "December 15, 2015". The post discusses technical debt and how GitHub has been addressing it over the years. It includes a section titled "Merges in Git" which explains how GitHub's storage model for repositories works. The post also mentions a workaround developed by GitHub for a specific limitation related to merge strategies.

Anyone who has worked on a large enough codebase knows that technical debt is an inescapable reality: The more rapidly an application grows in size and complexity, the more technical debt is accrued. With GitHub's growth over the last 7 years, we have found plenty of nooks and crannies in our codebase that are inevitably below our very best engineering standards. But we've also found effective and efficient ways of paying down that technical debt, even in the most active parts of our systems.

At GitHub we try not to brag about the "shortcuts" we've taken over the years to scale our web application to more than 12 million users. In fact, we do quite the opposite: we make a conscious effort to study our codebase looking for systems that can be rewritten to be cleaner, simpler and more efficient, and we develop tools and workflows that allow us to perform these rewrites efficiently and reliably.

As an example, two weeks ago we replaced one of the most critical code paths in our infrastructure: the code that performs merges when you press the Merge Button in a Pull Request. Although we routinely perform these kind of refactorings throughout our web app, the importance of the merge code makes it an interesting story to demonstrate our workflow.

Merges in Git

We've [talked at length in the past](#) about the storage model that GitHub uses for repositories in our platform and our Enterprise offerings. There are many implementation details that make this model efficient in both performance and disk usage, but the most relevant one here is the fact that repositories are always stored "*bare*".

This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside [packfiles](#).

Because of this, performing a merge in a production environment is a nontrivial endeavour. Git knows several [merge strategies](#), but the recursive merge strategy that you'd get by default when using `git merge` to merge two branches in a local repository assumes the existence of a working tree for the repository, with all the files checked out on it.

The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default `git-merge-recursive` strategy, we wrote our own merge strategy based on the original one that Git used back in the day: `git-merge-resolve`. With some tweaking, the old strategy can be adapted to not require an actual checkout of the files on disk.

To accomplish this, we wrote a shell script that sets up a temporary working directory, in

Move
Fast
&
Fix
Things



```
def create_merge_commit(base, head, author, commit_message)
  base = resolve_commit(base)
  head = resolve_commit(head)
  commit_message = Rugged.prettyify_message(commit_message)

  merge_base = rugged.merge_base(base, head)
  return [nil, "already_merged"] if merge_base == head.oid

  ancestor_tree = merge_base && Rugged::Commit.lookup(rugged, merge_base).tree
  merge_options = {
    :fail_on_conflict => true,
    :skip_revc => true,
    :no_recursive => true,
  }
  index = base.tree.merge(head.tree, ancestor_tree, merge_options)
  return [nil, "merge_conflict"] if (index.nil? || index.conflicts?)

  options = {
    :message => commit_message,
    :committer => author,
    :author => author,
    :parents => [base, head],
    :tree => index.write_tree(rugged)
  }

  [Rugged::Commit.create(rugged, options), nil]
end
```

```
def create_merge_commit(author, base, head, options = {})
  commit_message = options[:commit_message] || "Merge #{head} into #{base}"
  now = Time.current

  science "create_merge_commit" do |e|
    e.context :base => base.to_s, :head => head.to_s, :repo => repository.repo
    e.use { create_merge_commit_git(author, now, base, head, commit_message) }
    e.try { create_merge_commit_rugged(author, now, base, head, commit_message) }
  end
end
```

A screenshot of a GitHub repository page for 'scientist'. The page shows the repository's overview, including 71 commits, 1 branch, 6 releases, and 16 contributors. It features a timeline of commits, a 'Scientist!' section with a note about changing permissions, and a 'How do I science?' section with a code snippet for refactoring. The URL at the bottom is https://github.com/github/scientist.

<https://github.com/github/scientist>



```
require "scientist"

class MyWidget
  def allows?(user)
    experiment = Scientist::Default.new "widget-permissions"
    experiment.use { model.check_user?(user).valid? } # old way
    experiment.try { user.can?(:read, model) } # new way

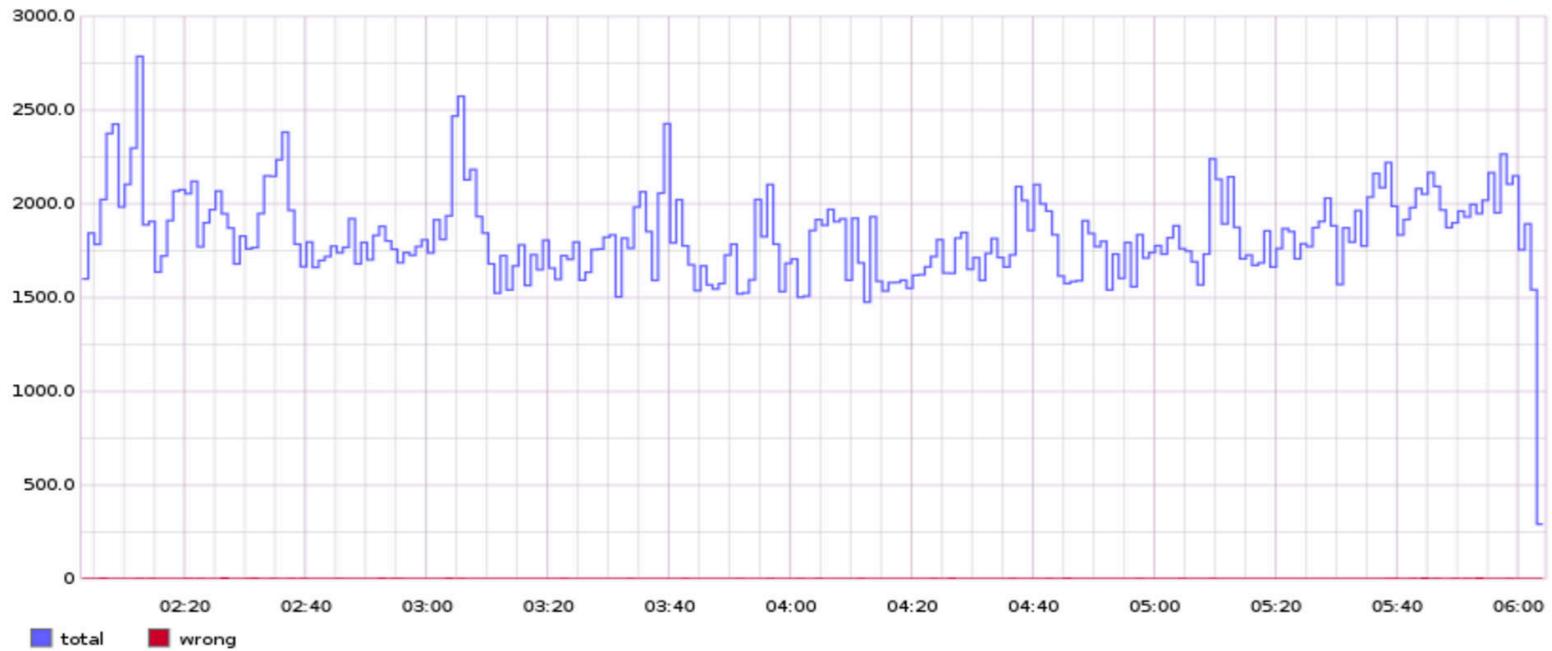
    experiment.run
  end
end
```

- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.



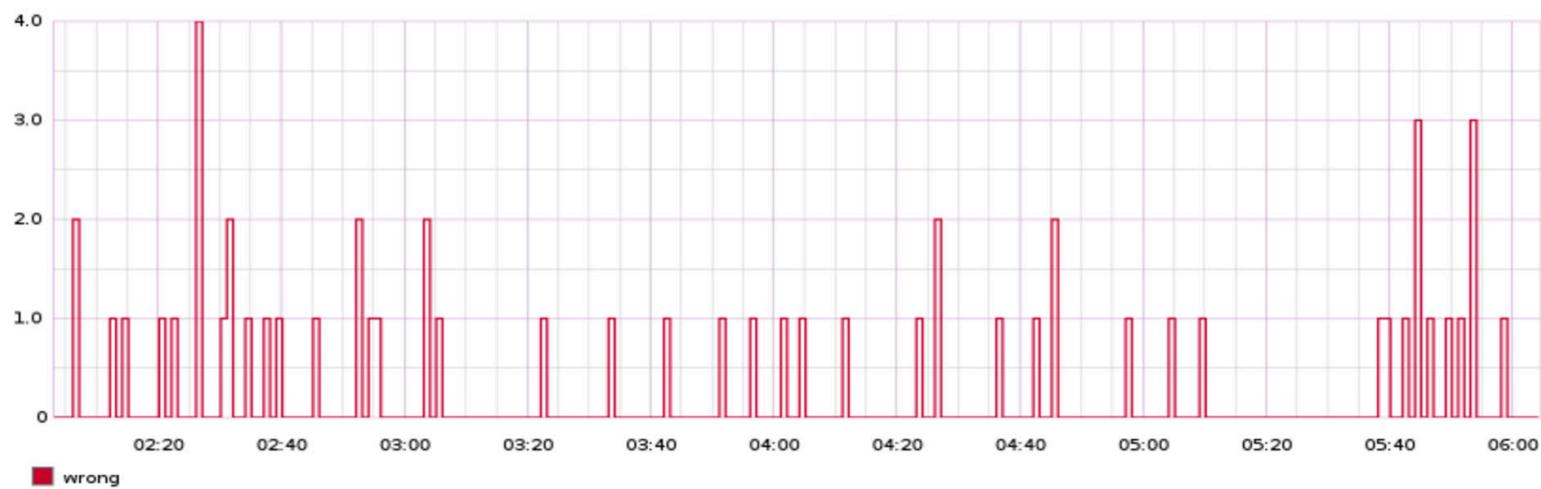
Accuracy

The number of times that the candidate and the control agree or disagree. [View mismatches](#)



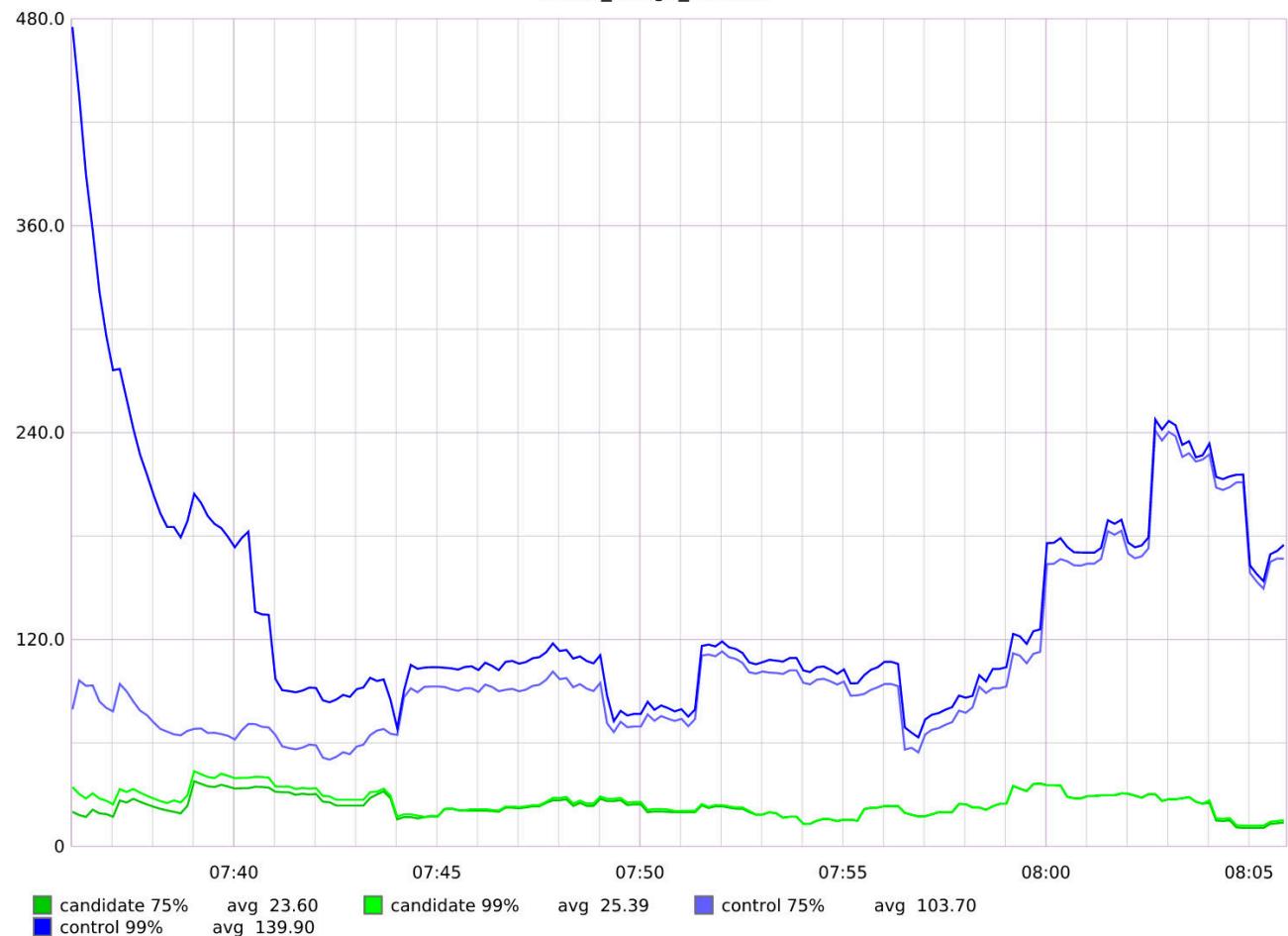


The number of incorrect/ignored only.





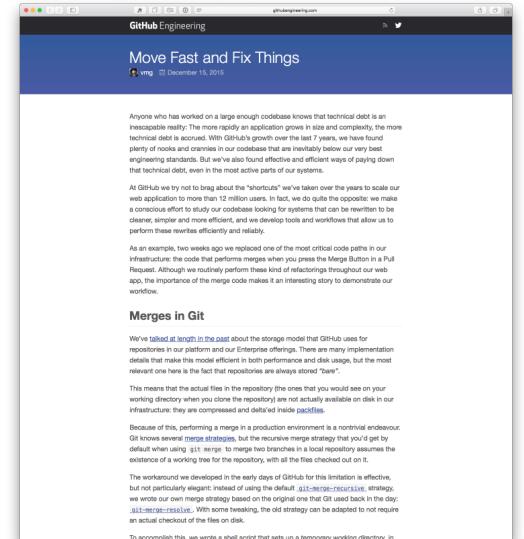
create_merge_commit



4 days

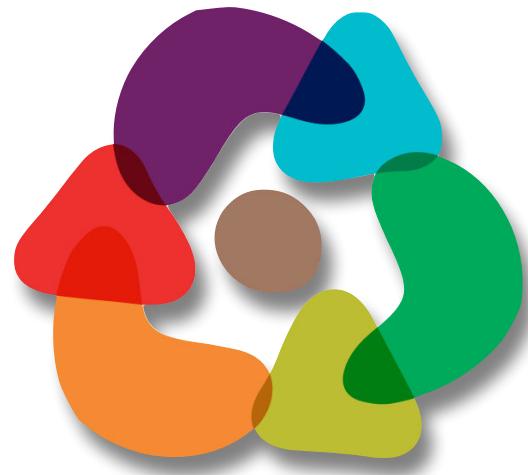
**24 hours/
no mismatches or slow cases**

**> 10,000,000
comparisons**



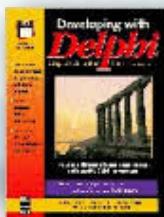
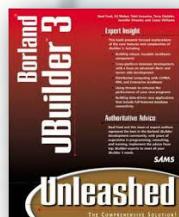
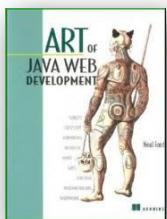
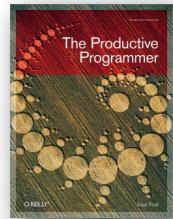
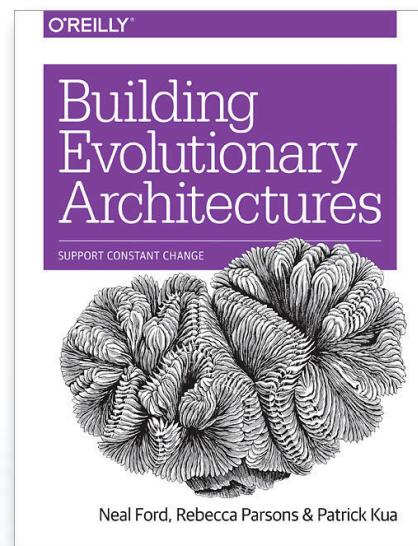
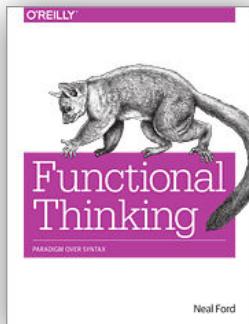
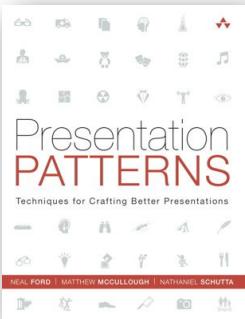


aRChiTeCtuRe
fOr
cONTiNuoUs DeLiVeRy



aRChiTeCtuRe aNd cONTiNuoUs DeLiVeRy





nealford.com/books

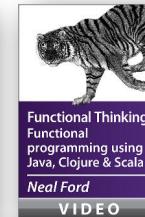
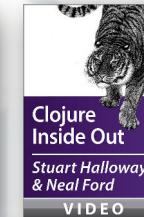
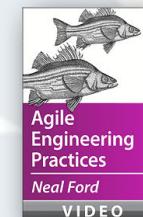


ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler

nealford.com/videos



O'REILLY®

SOFTWARE ARCHITECTURE SERIES

www.oreilly.com/software-architecture-video-training-series.html

