Hot-or-Not Microservices
with James Lewis

Author: Philippe Dirkse, Senior Software Architect

March 7th, 2017

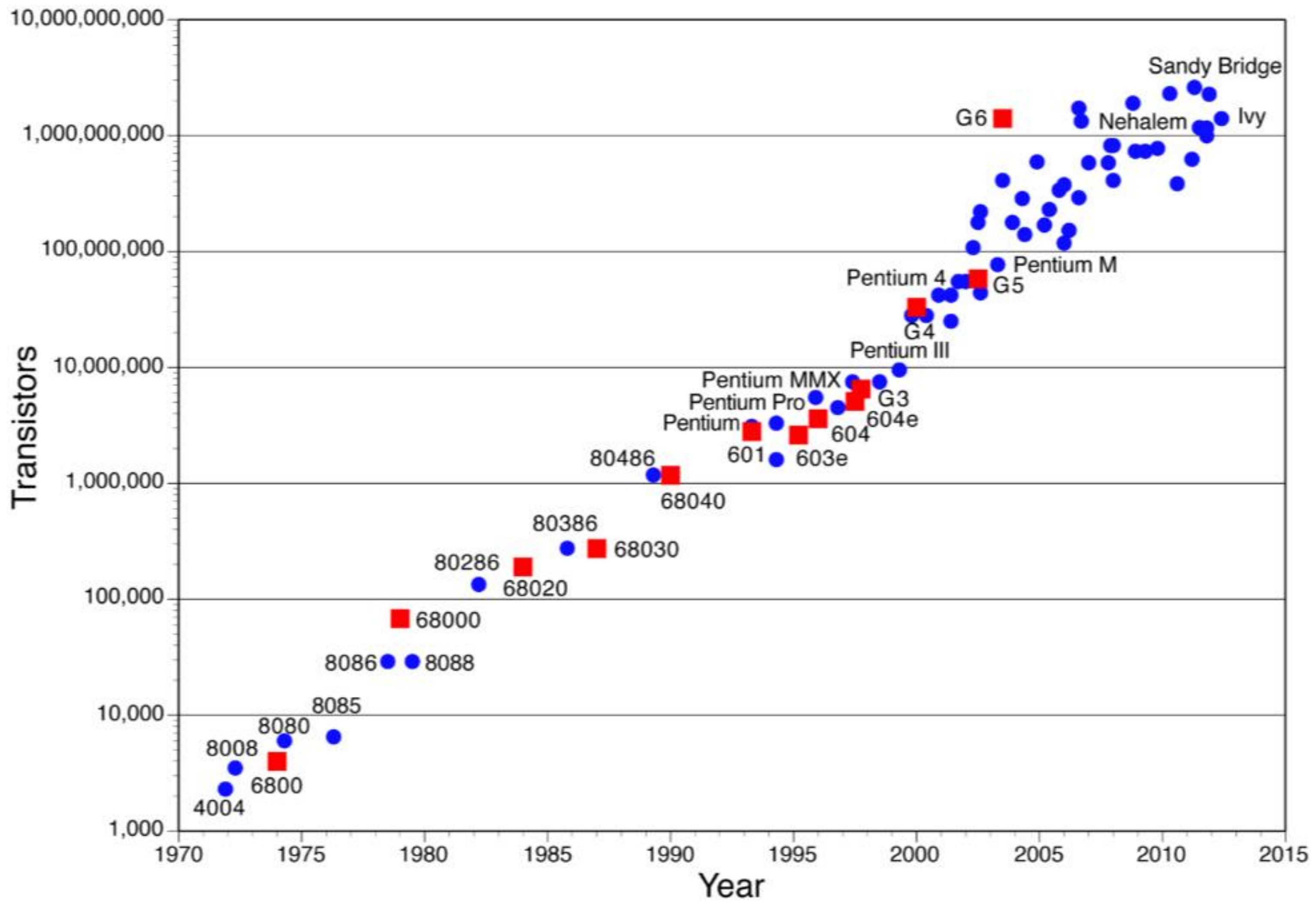
WeLcOme

Timetable

18:00h	Introduction
18:05h	Microservices, part 1
19:30h	Break
20:00h	Microservices, part 2
20:45h	Q & A
21:00h	Drinks

#End of Program

Familiar…?



Transistors
Year
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
4004
8008
6800
8080
8085
8086
8088
68000
80286
68020
80386
68030
80486
68040
Pentium
601
603e
Pentium Pro
604
Pentium MMX
604e
G3
Pentium III
G4
Pentium 4
G5
Pentium M
G6
Nehalem
Sandy Bridge
Ivy

Software expectations also double every year!

- Always available
- Scalable
- Responsive
- DevOps
- Zero downtime deployment

James Lewis will explain how Microservices can help elevate your software…

…to meet the expectations!

anitech
viny

The Floor is yours

ThoughtWorks®

FOUNDATIONS OF MICROSERVICES

jalewis@thoughtworks.com ***@boicy***

ThoughtWorks®

CONSULTANTS!
HELL YEAH!

ThoughtWorks®

A SOCIAL EXPERIMENT

Roy Singham founded ThoughtWorks in Chicago over 20 years ago with the aim of attracting and employing the best knowledge workers in the world – building a community based on attitude, aptitude and integrity.

ThoughtWorks®

jMock

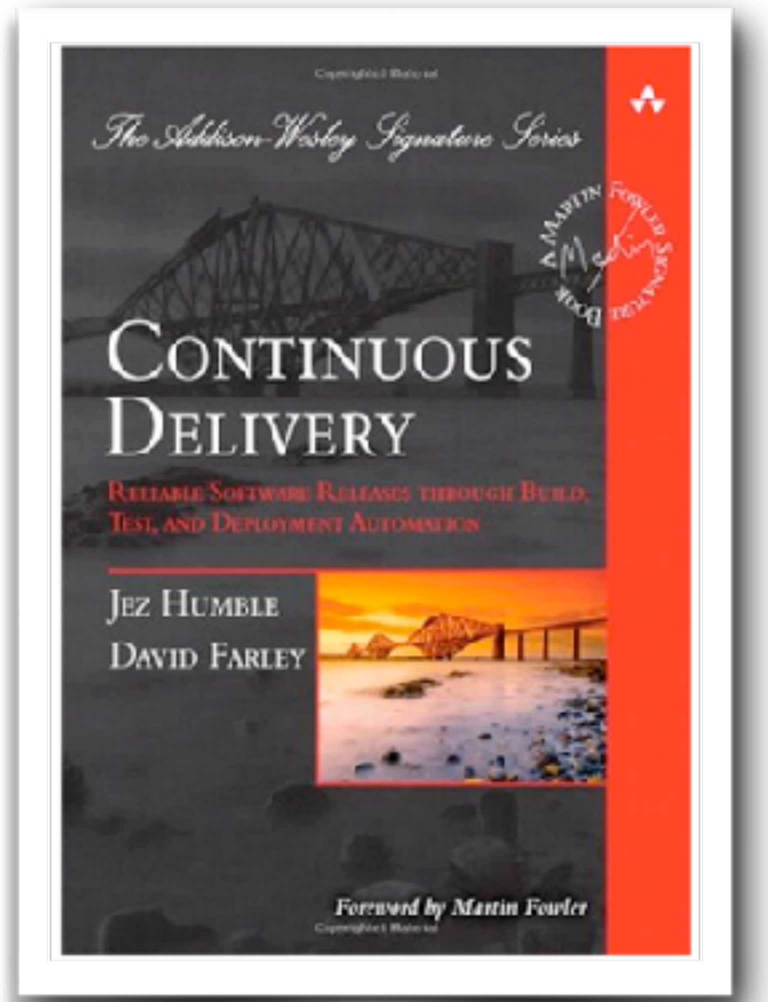
Se

go Continuous Delivery

cruisecontrol

pico container

BDD

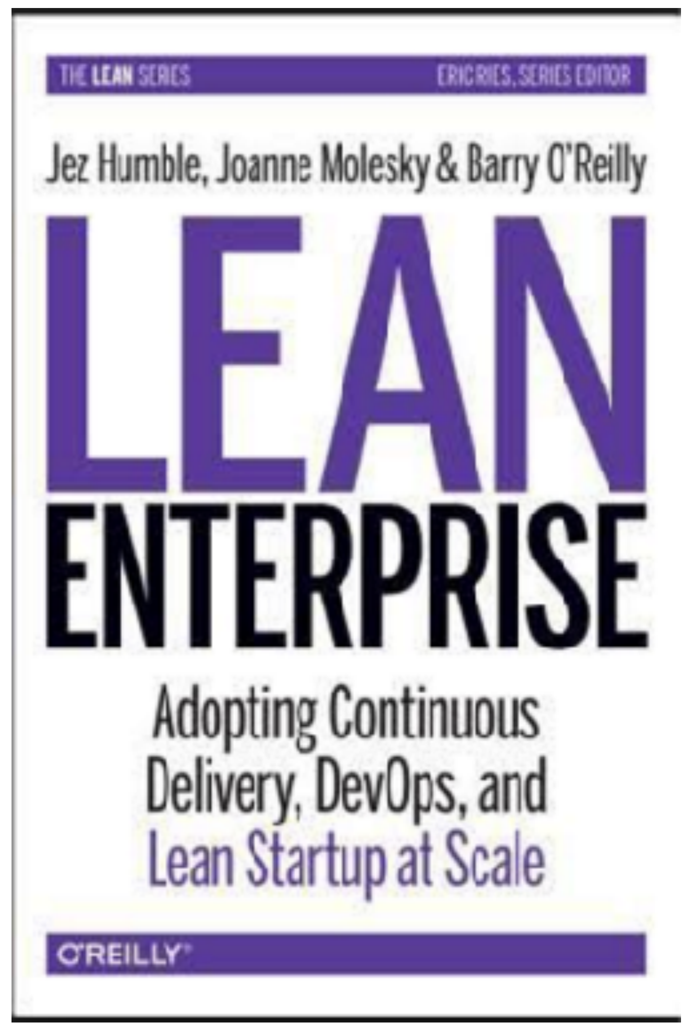

The Addison-Wesley Signature Series

Continuous Delivery

Reliable Software Releases through Build, Test, and Deployment Automation

Jez Humble
David Farley

Foreword by Martin Fowler

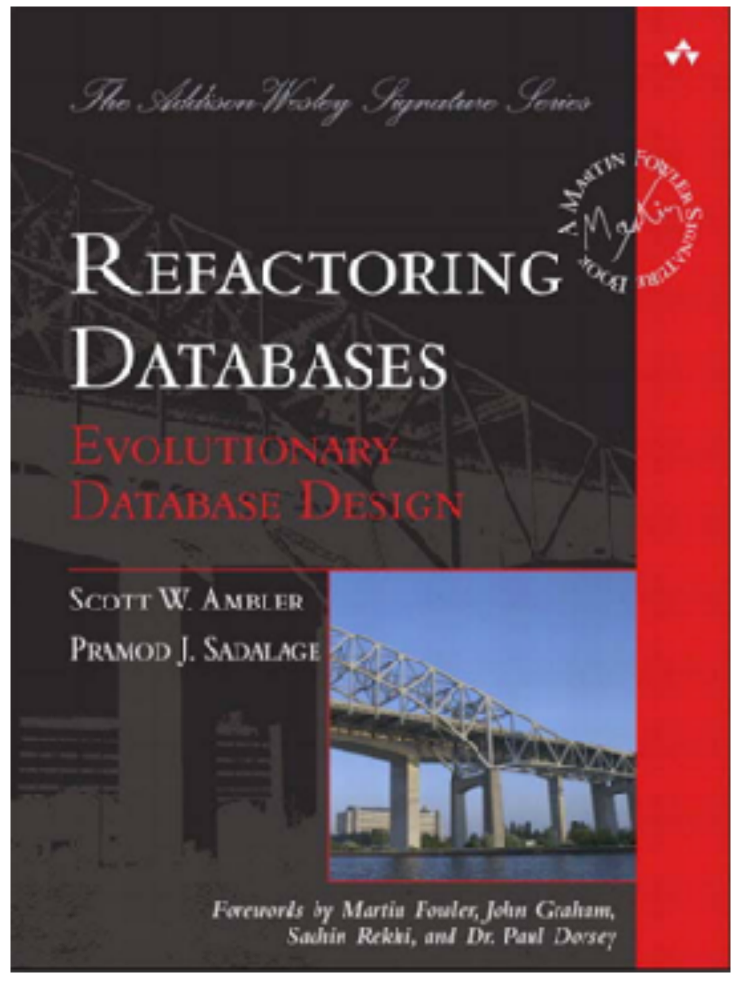

THE LEAN SERIES ERIC RIES, SERIES EDITOR

Jez Humble, Joanne Molesky & Barry O'Reilly

LEAN ENTERPRISE

Adopting Continuous Delivery, DevOps, and Lean Startup at Scale

O'REILLY


The Addison-Wesley Signature Series

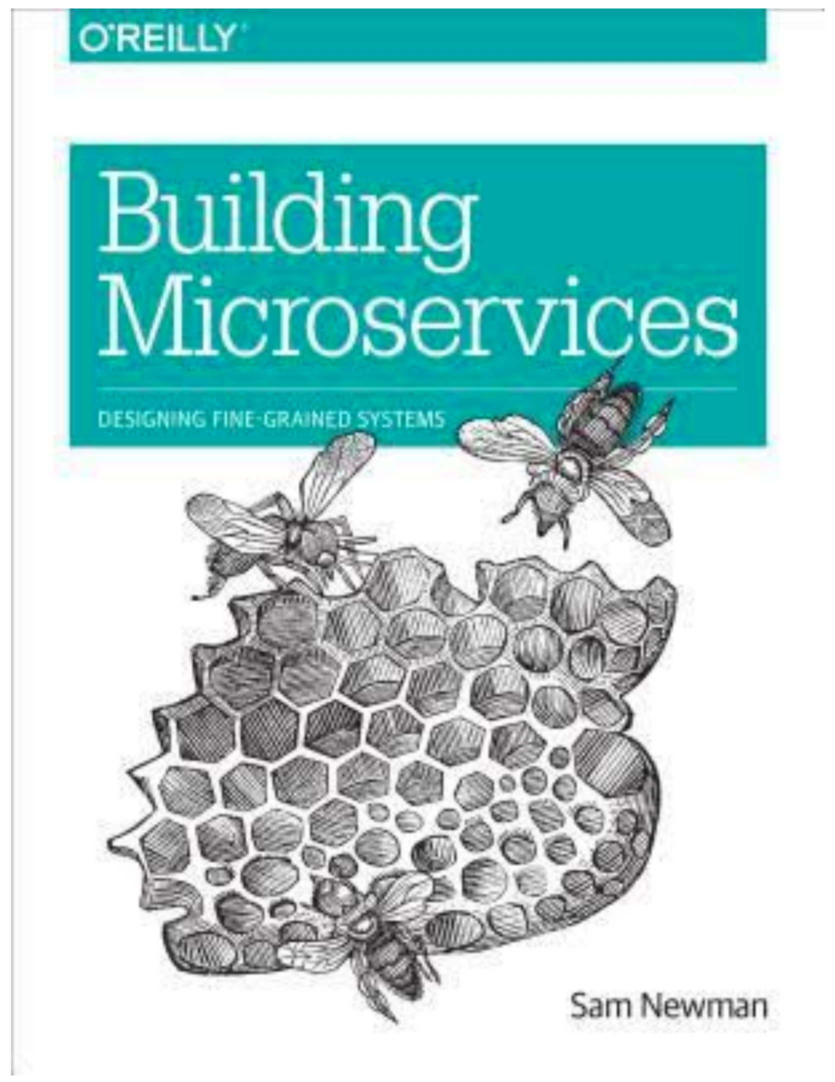
Refactoring Databases

Evolutionary Database Design

Scott W. Ambler
Pramod J. Sadalage

Forewords by Martin Fowler, John Graham, Sachin Rekhi, and Dr. Paul Dorsey

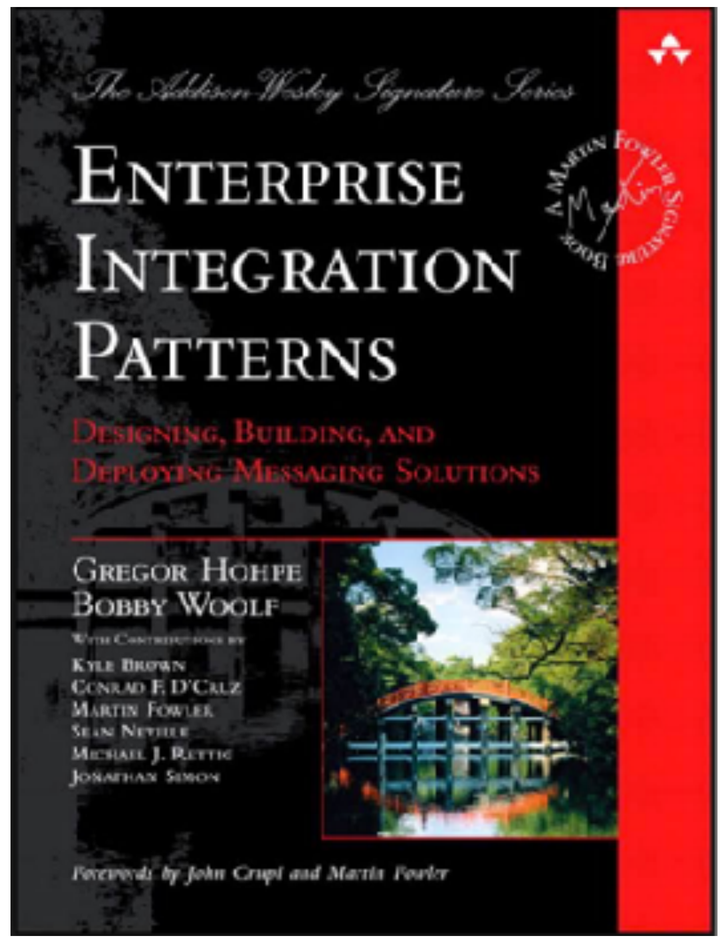
REST in Practice


O'REILLY

Building Microservices

DESIGNING FINE-GRAINED SYSTEMS

Sam Newman

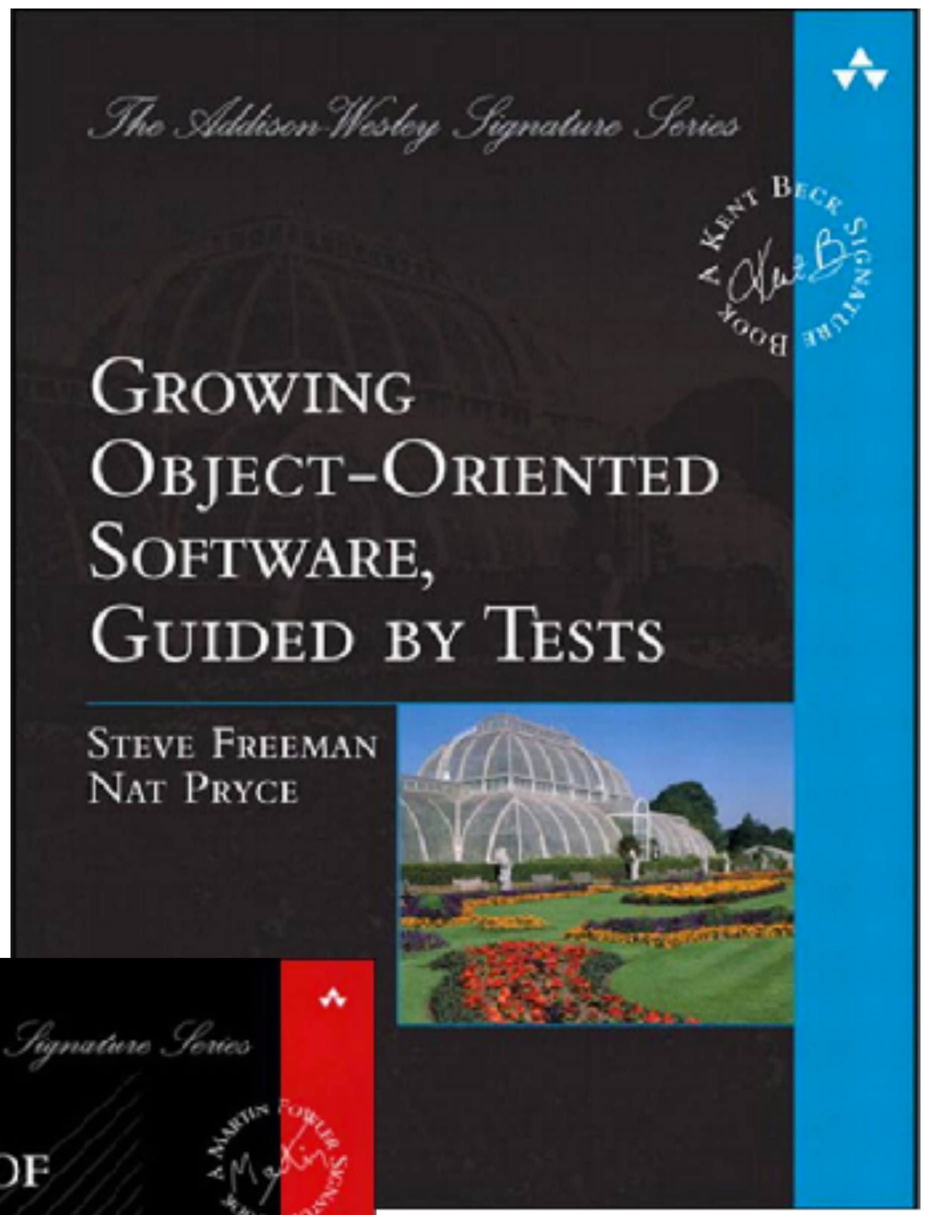

The Addison-Wesley Signature Series

Enterprise Integration Patterns

Designing, Building, and Deploying Messaging Solutions

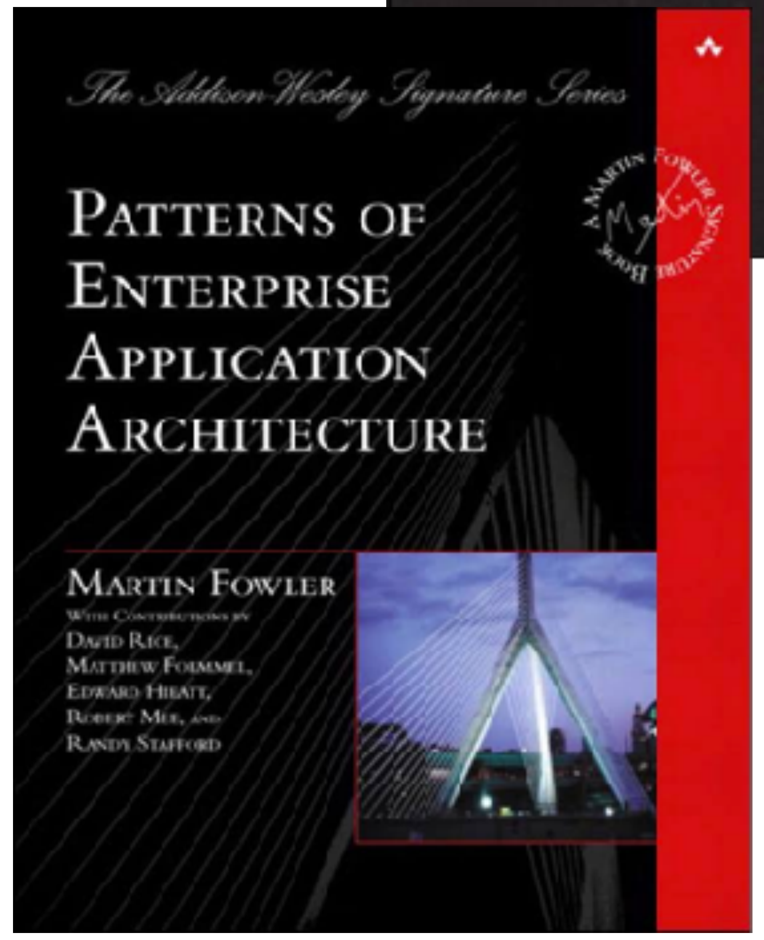
Gregor Hohpe
Bobby Woolf

Forewords by John Crupi and Martin Fowler


The Addison-Wesley Signature Series

Growing Object-Oriented Software, Guided by Tests

Steve Freeman
Nat Pryce


The Addison-Wesley Signature Series

Patterns of Enterprise Application Architecture

Martin Fowler

With Contributions by David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford

TECHNOLOGY RADAR

Search A-Z FAQs

Techniques | Tools | Platforms | Languages & Frameworks

● ADOPT

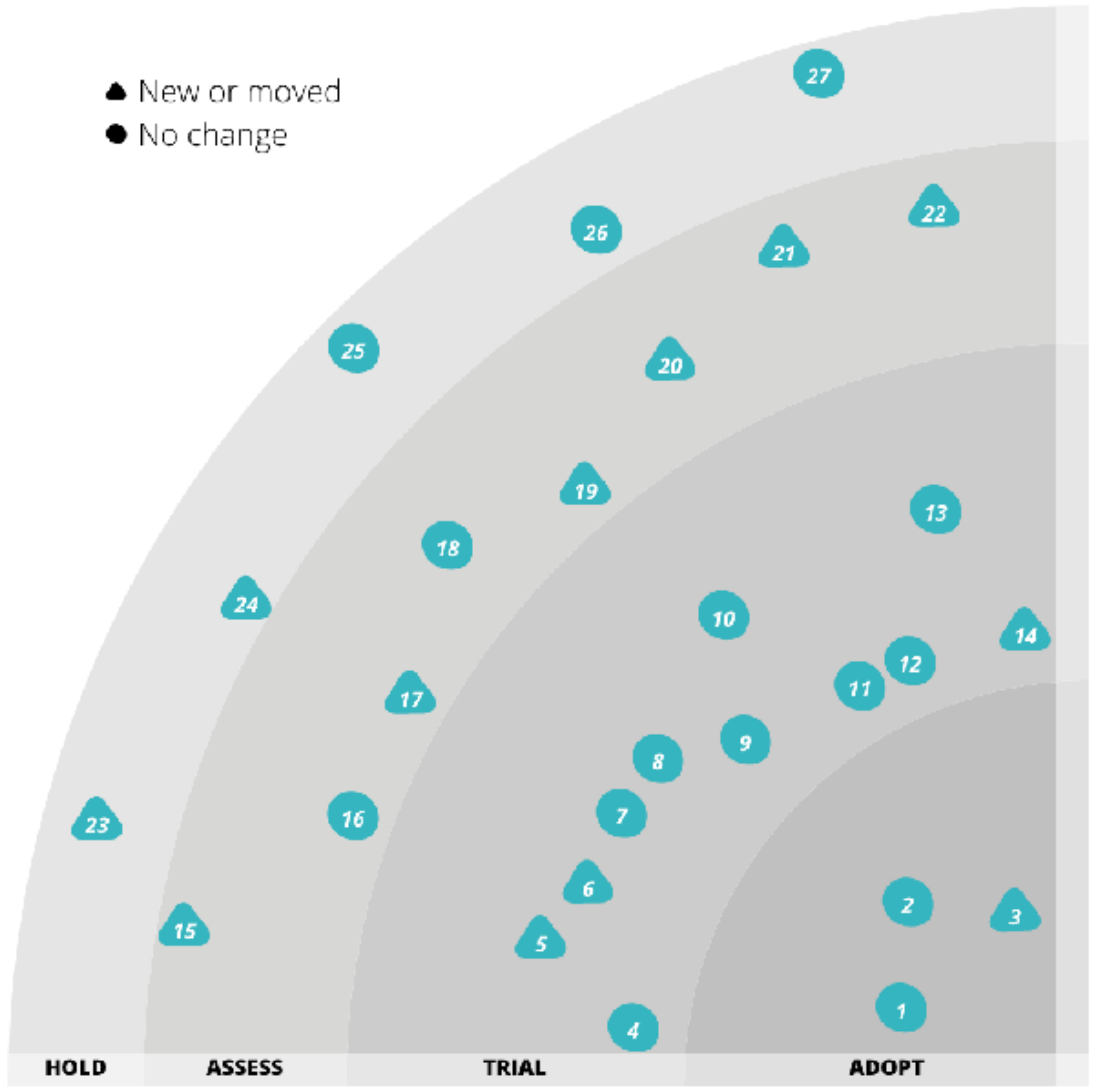
1. Decoupling deployment from release
2. Products over projects
3. Threat Modeling

● TRIAL

4. BFF - Backend for frontends
5. Bug bounties
6. Data Lake
7. Event Storming
8. Flux
9. Idempotency filter
10. iFrames for sandboxing
11. NPM for all the things
12. Phoenix Environments
13. QA in production
14. Reactive architectures

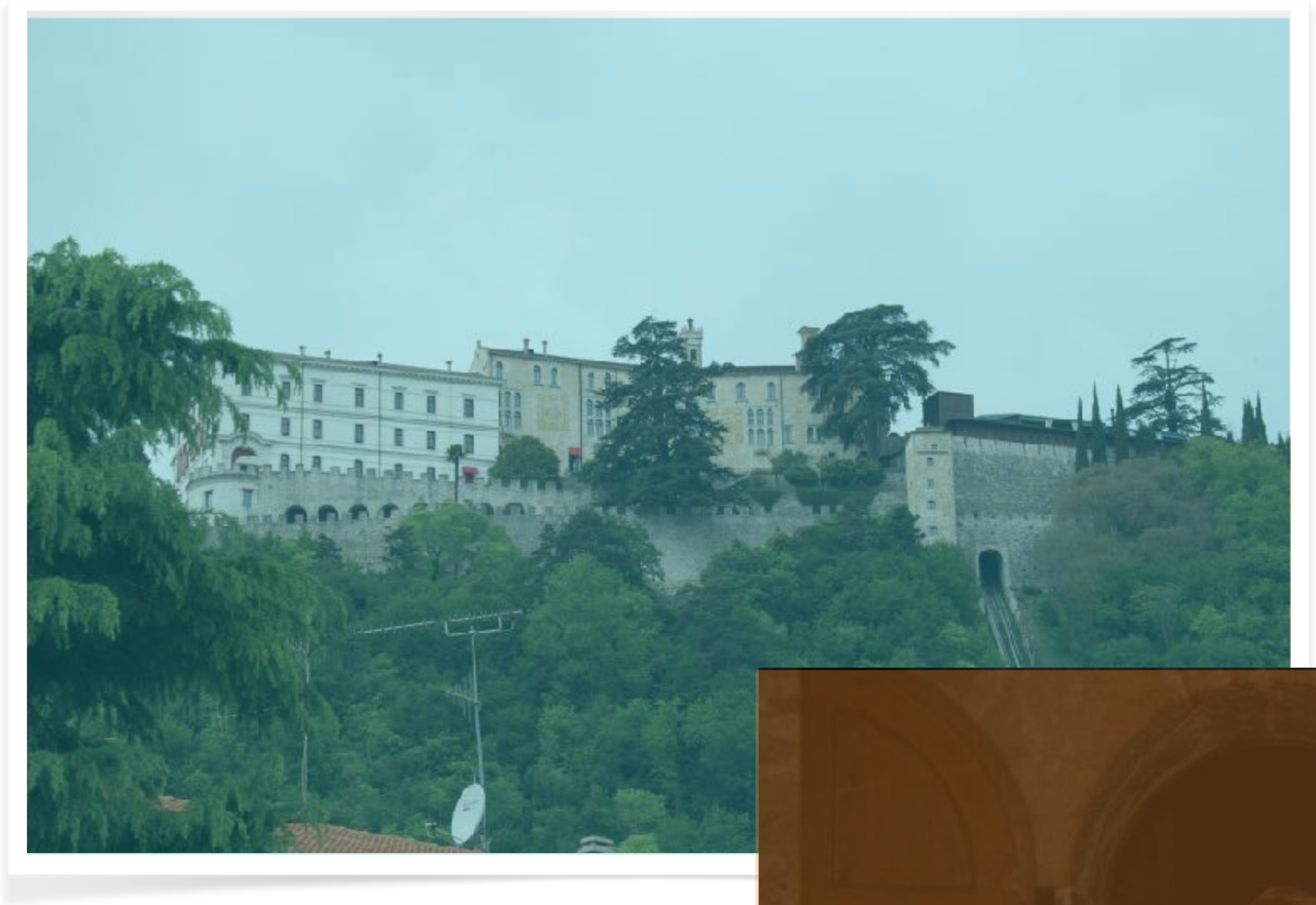
● ASSESS

15. Content Security Policies new
16. Hosted IDE's
17. Hosting PII data in the EU new
18. Monitoring of invariants
19. OWASP ASVS new
20. Serverless architecture new
21. Unikernels new
22. VR beyond gaming new

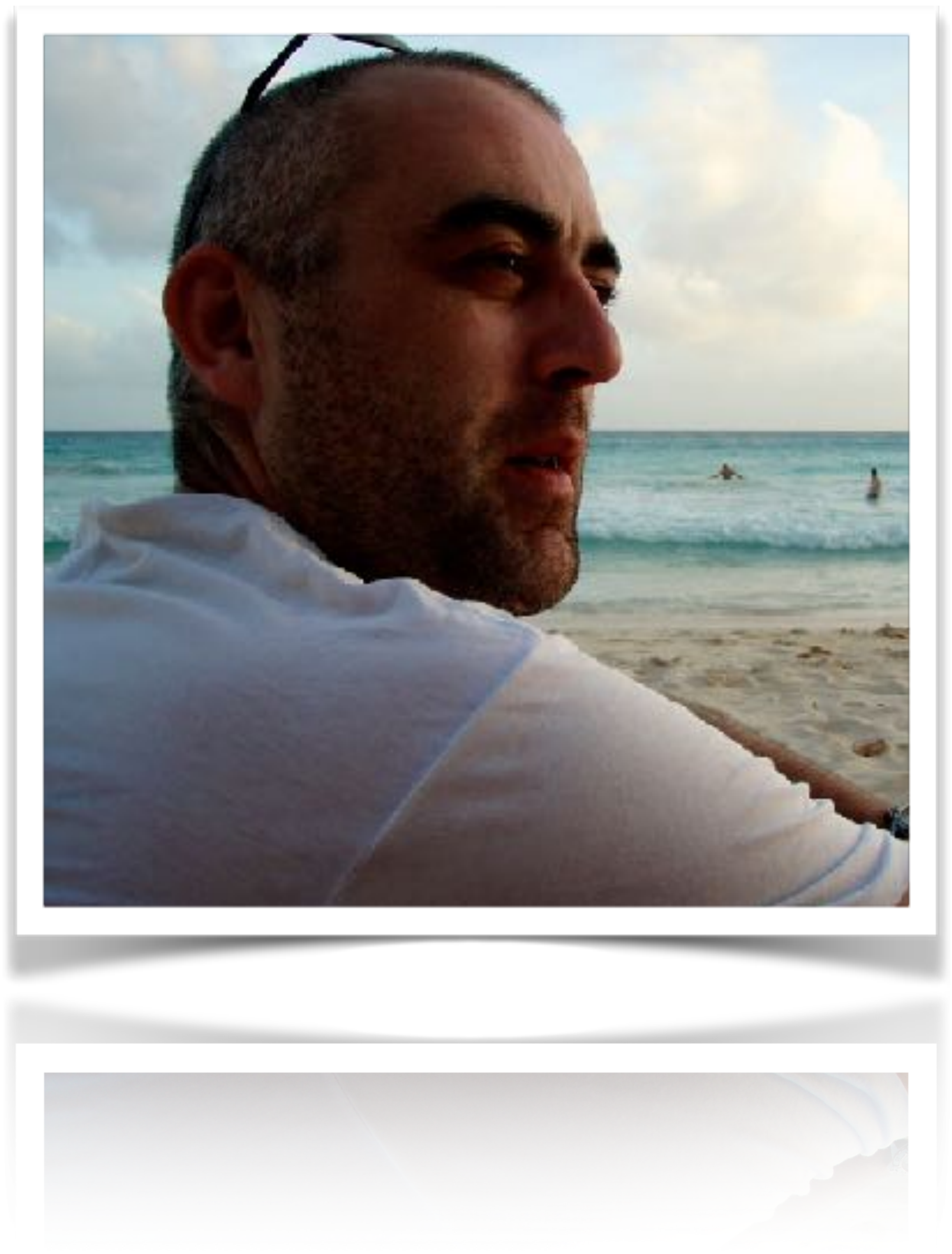


Unable to find something you expected to see? Your item may have been on a previous radar »

SAW 2011



TIME PASSES...



Microservices

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

25 March 2014

James Lewis

James Lewis is a Principal Consultant at ThoughtWorks and member of the Technology Advisory Board. James' interest in building applications out of small collaborating services stems from a background in integrating enterprise systems at scale. He's built a number of systems using microservices and has been an active participant in the growing community for a couple of years.

Martin Fowler

Martin Fowler is an author, speaker, and general loud-mouth on software development. He's long been puzzled by the problem of how to componentize

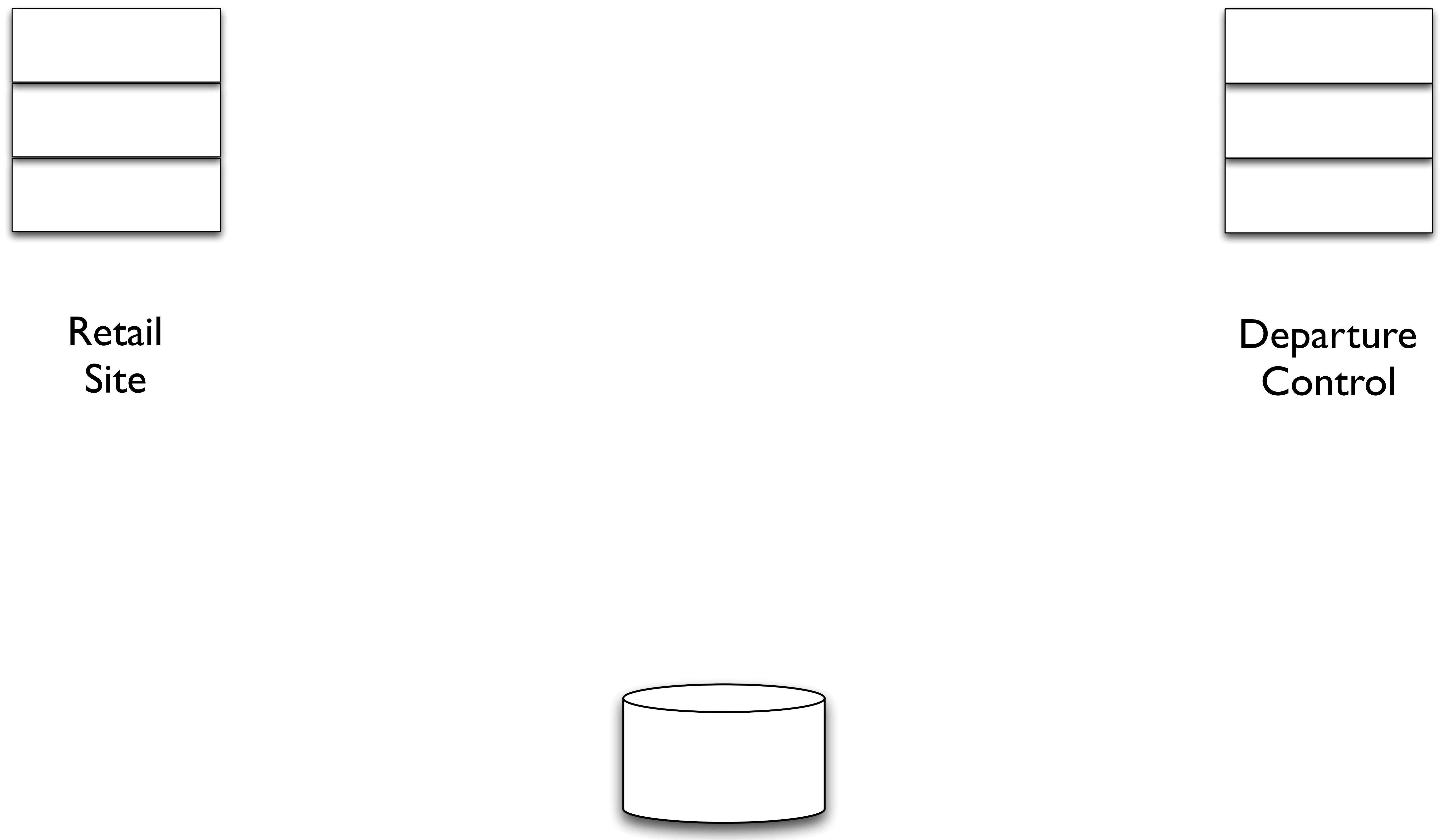
Contents

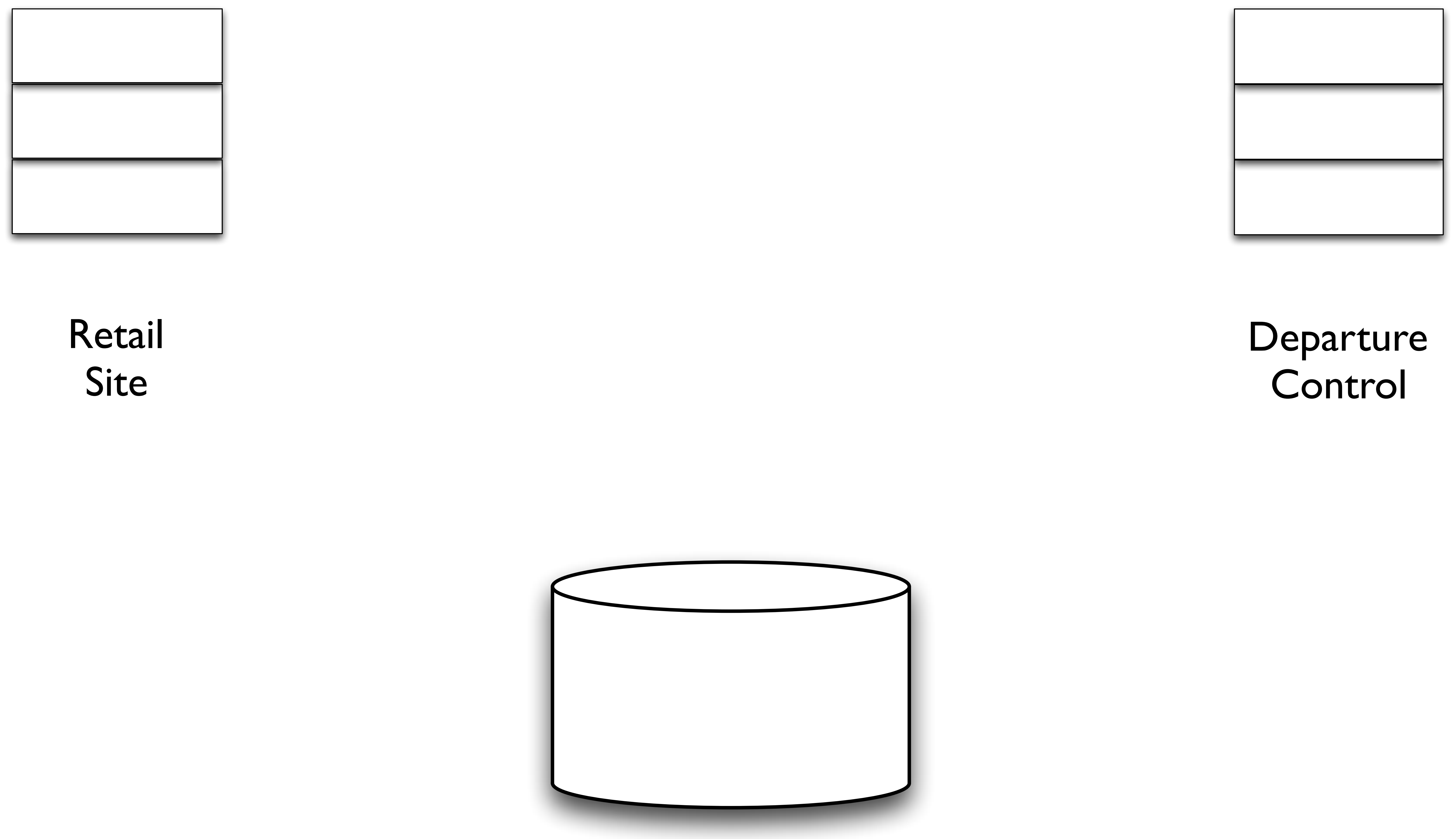
Part the Second

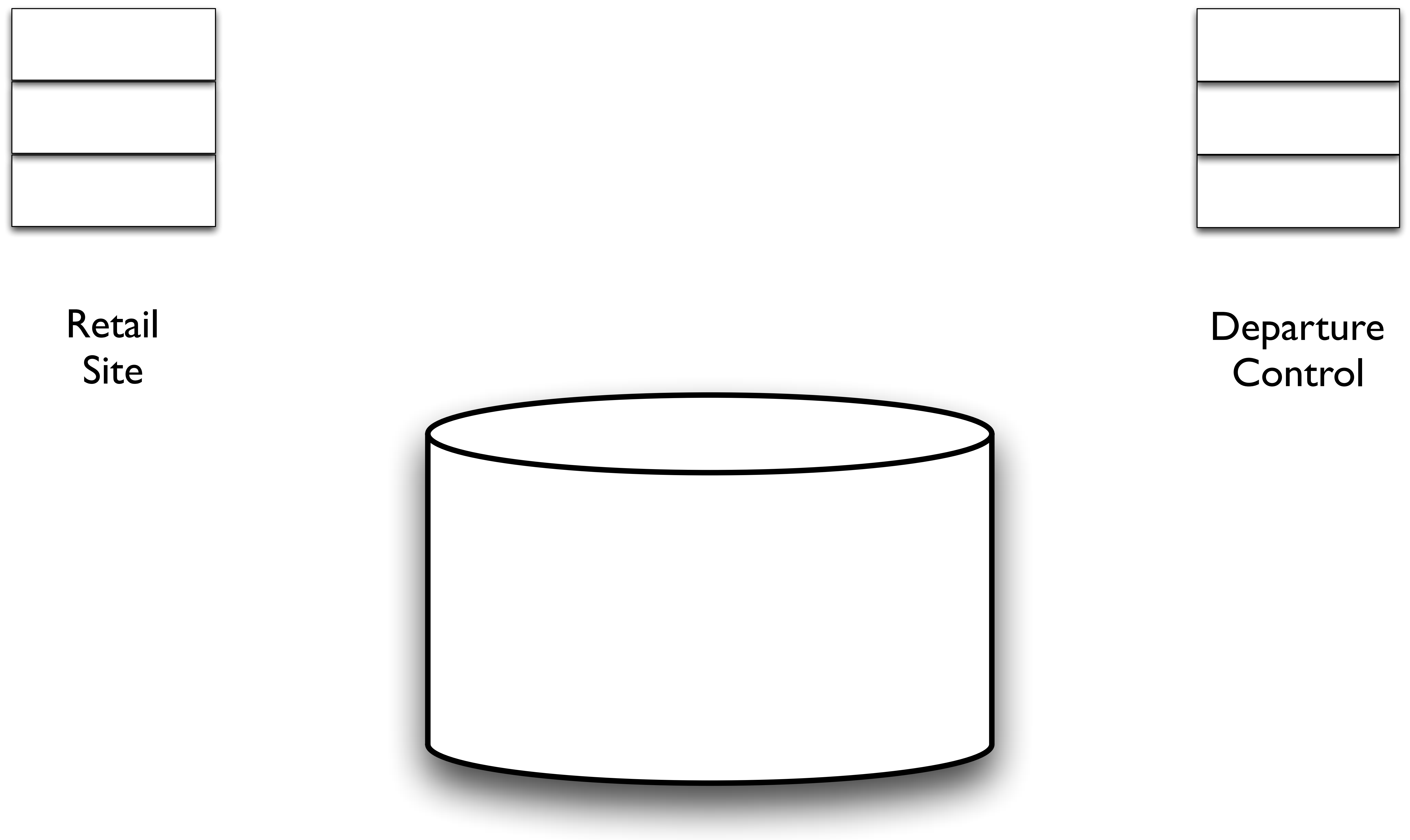
Why Microservices

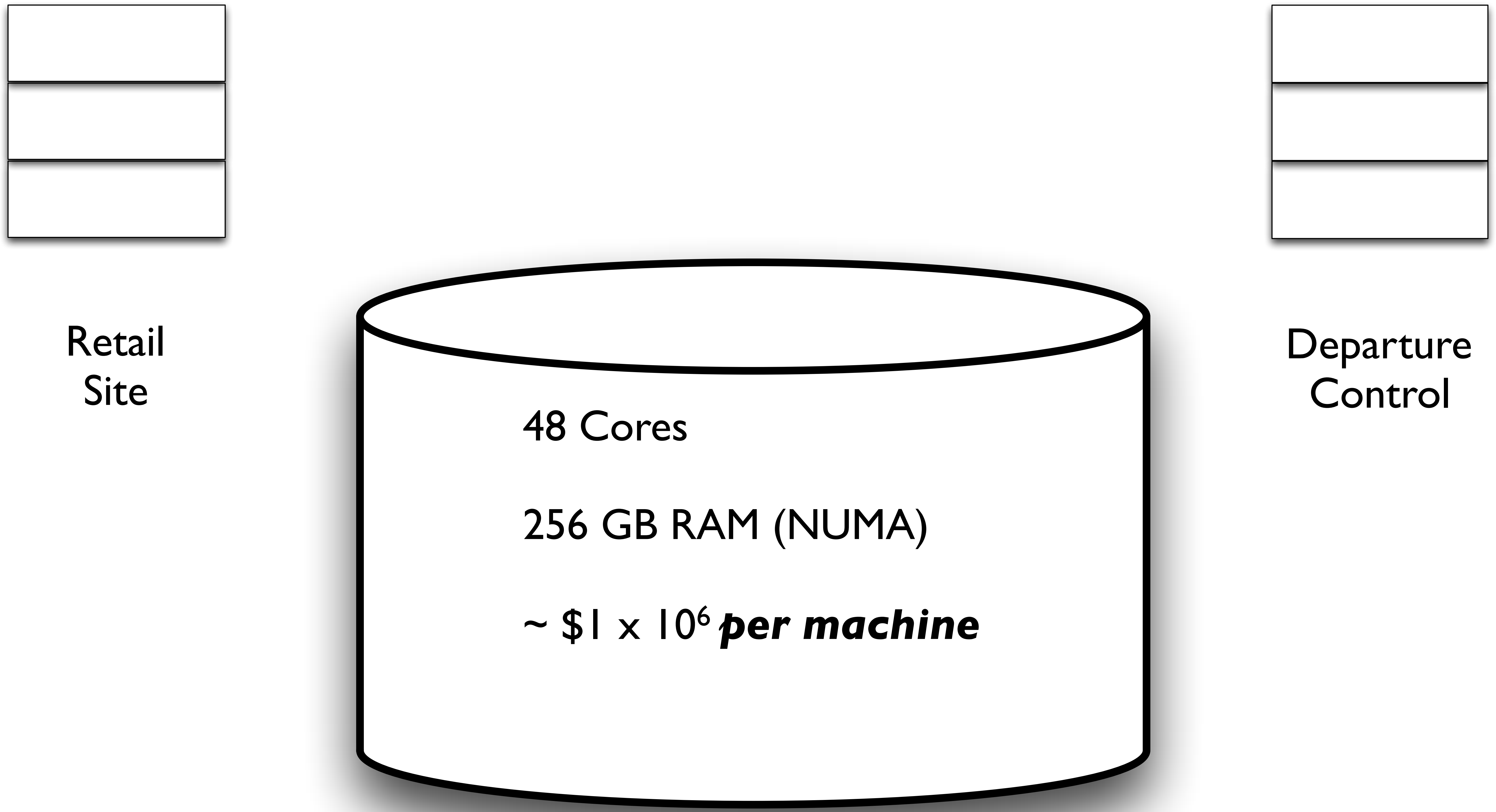
"The Bellman himself they all praised to the skies—
Such a carriage, such ease and such grace!
Such solemnity, too! One could see he was wise,
The moment one looked in his face! "

Airline problems: monolithic databases ~ 2010


Retail
Site
Departure
Control

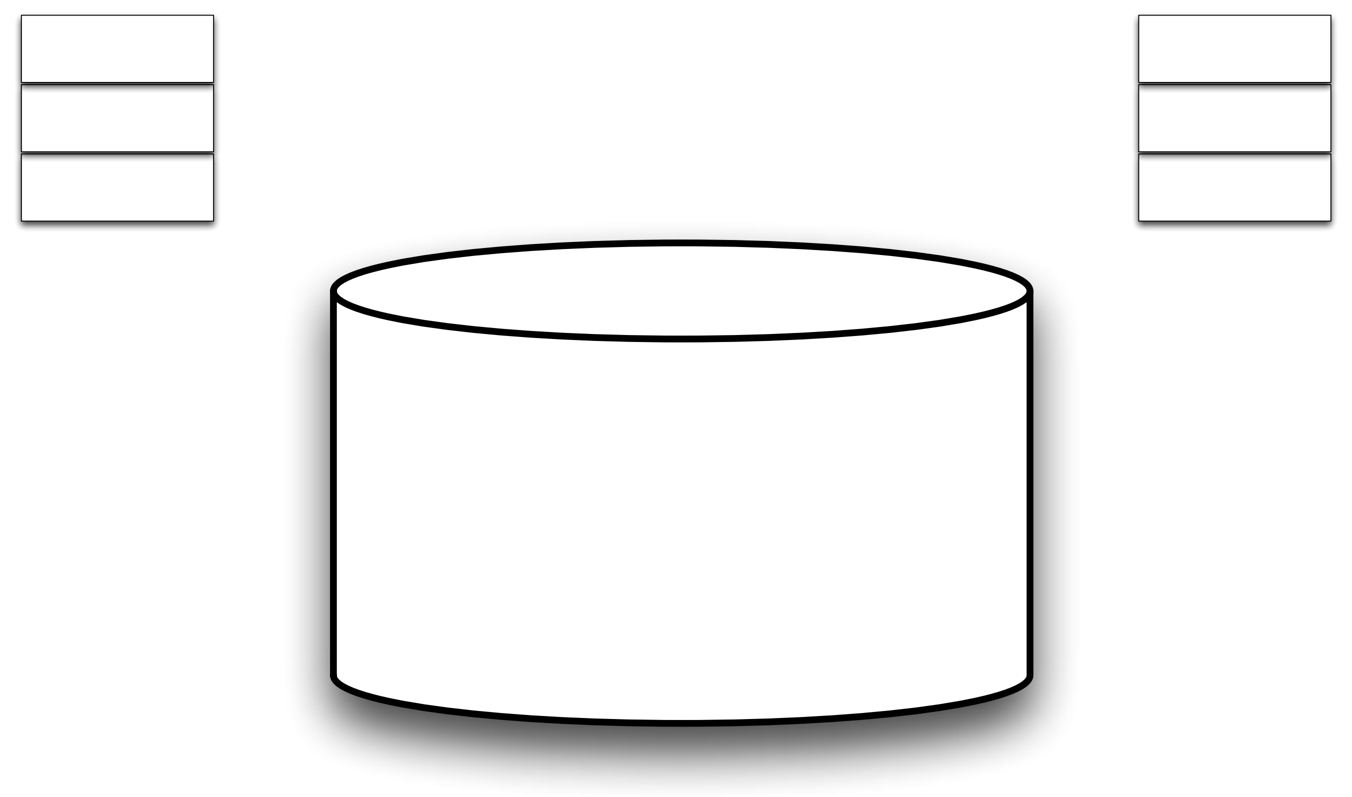

Retail
Site
Departure
Control


Retail
Site
Departure
Control


Retail Site
48 Cores
256 GB RAM (NUMA)
~ $1 x 10^6 ***per machine***
Departure Control

UNITED STATES OF AMERICA
ONE DOLLAR

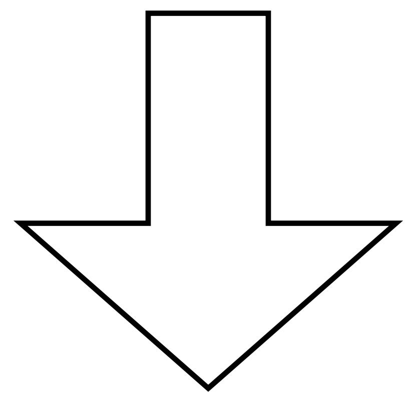
Airline



Tightly coupled

Single point of scaling

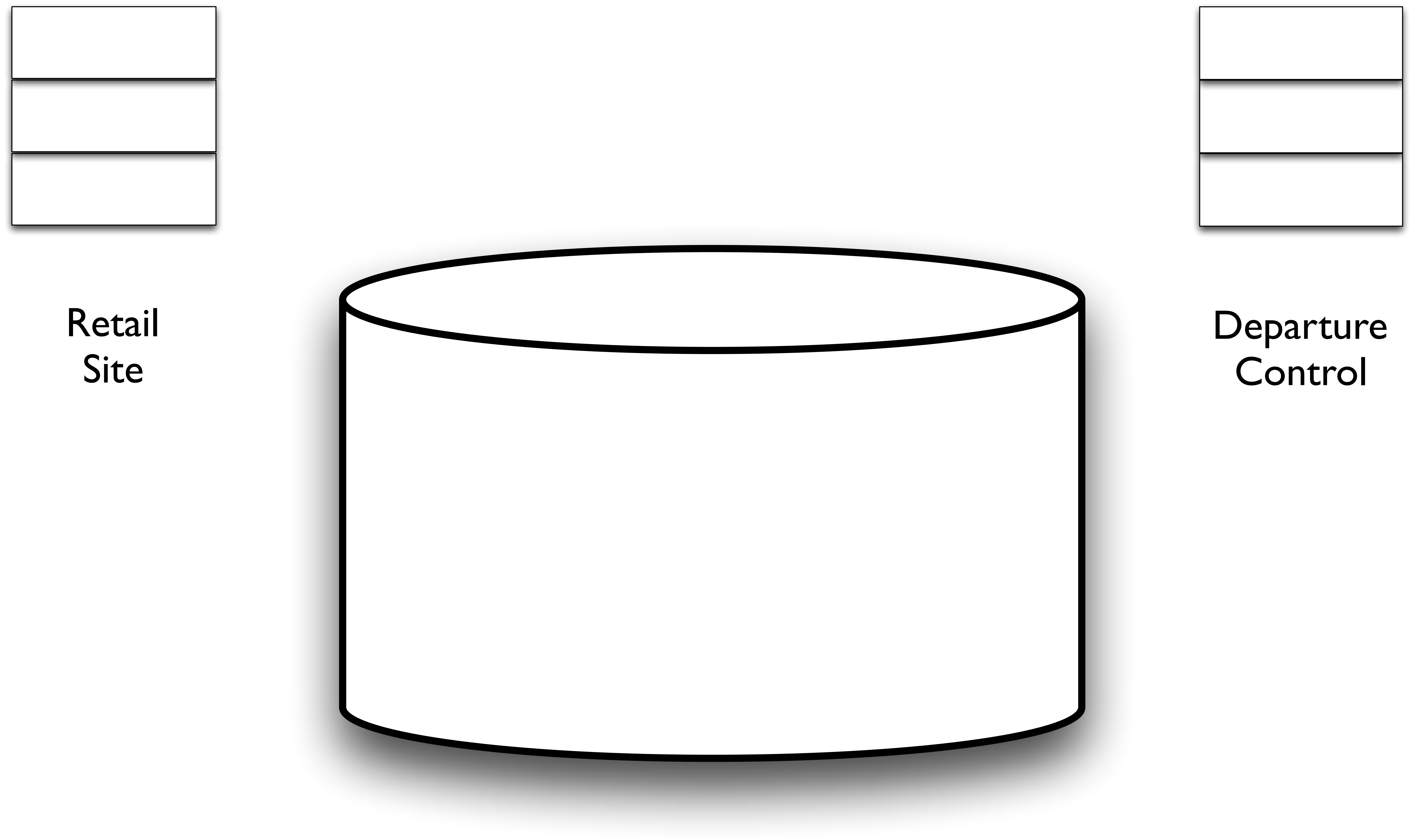
Single point of failure

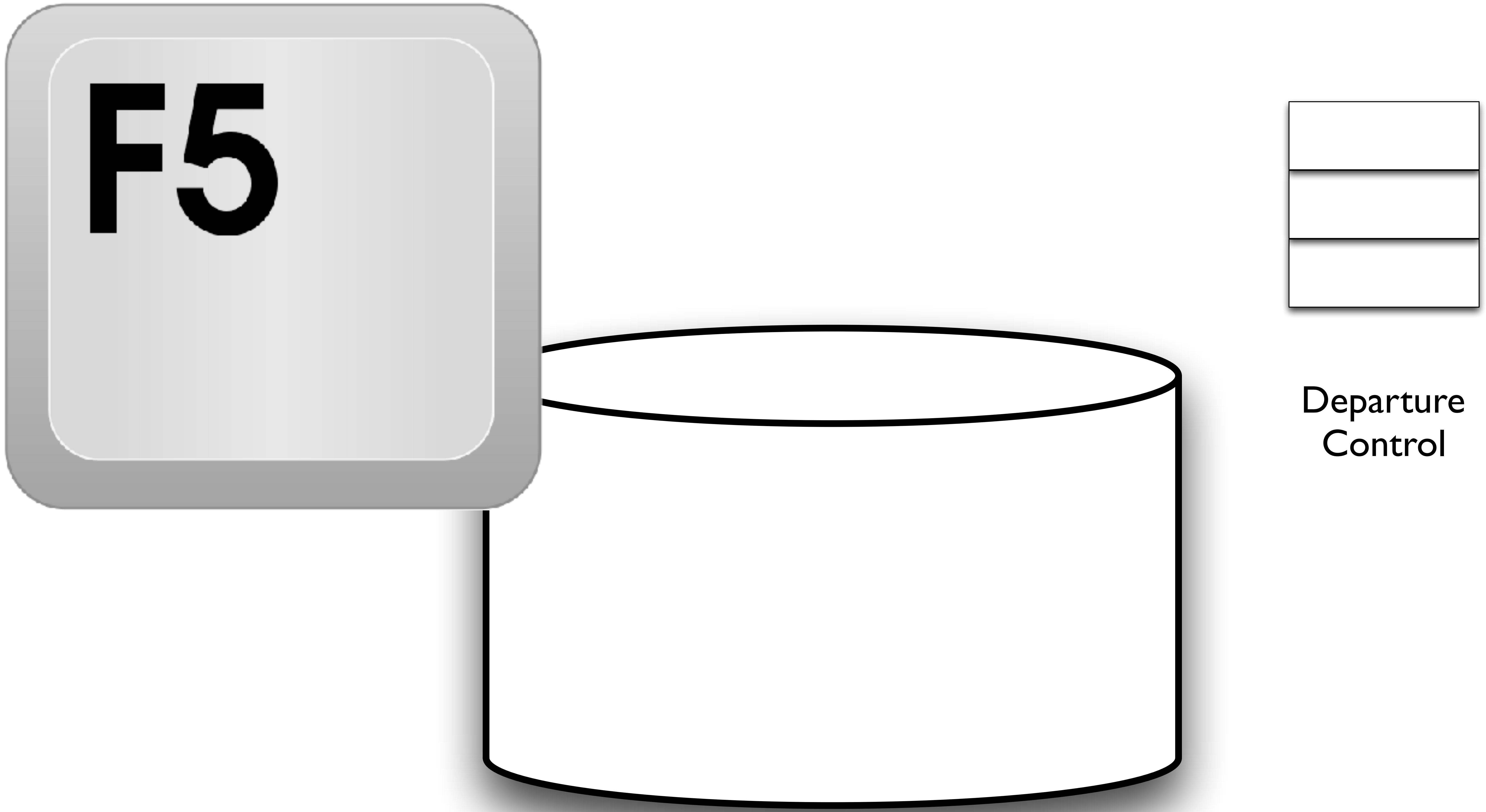


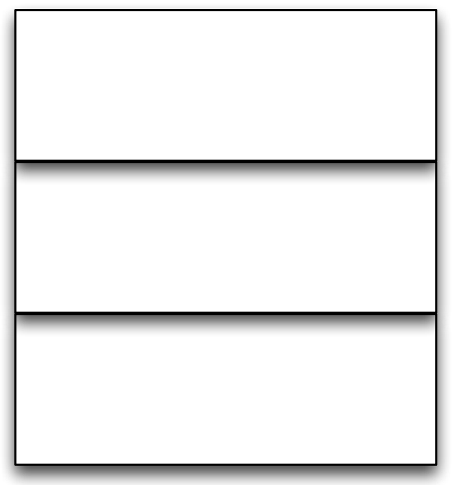
Expensive to change

High operational cost

High cost of failure

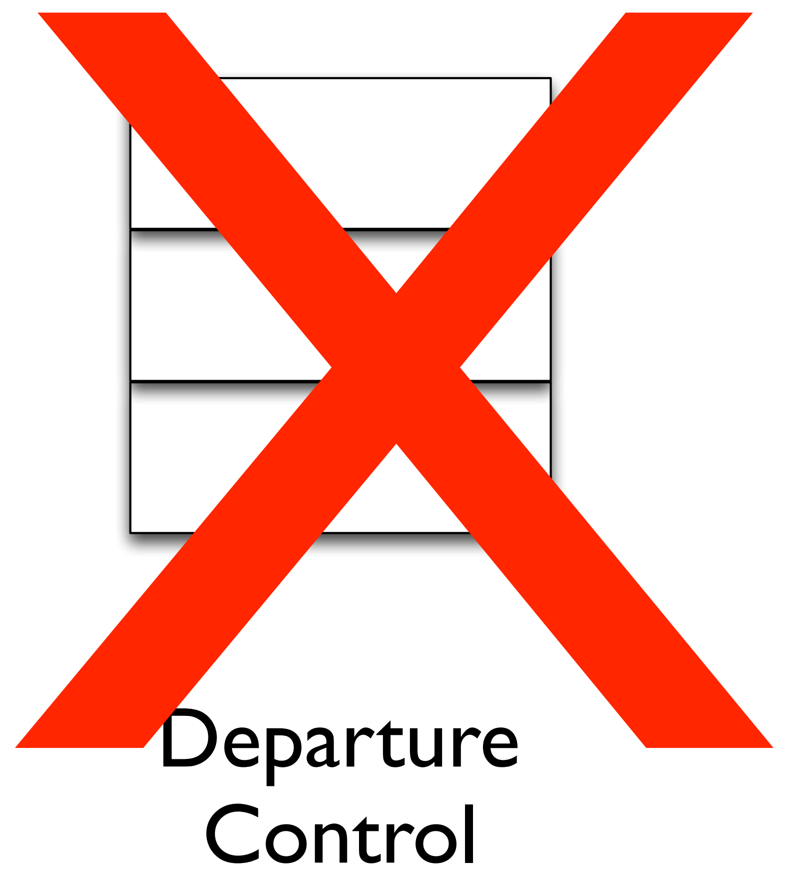

Retail
Site
Departure
Control


F5
Departure Control



Departure
Control

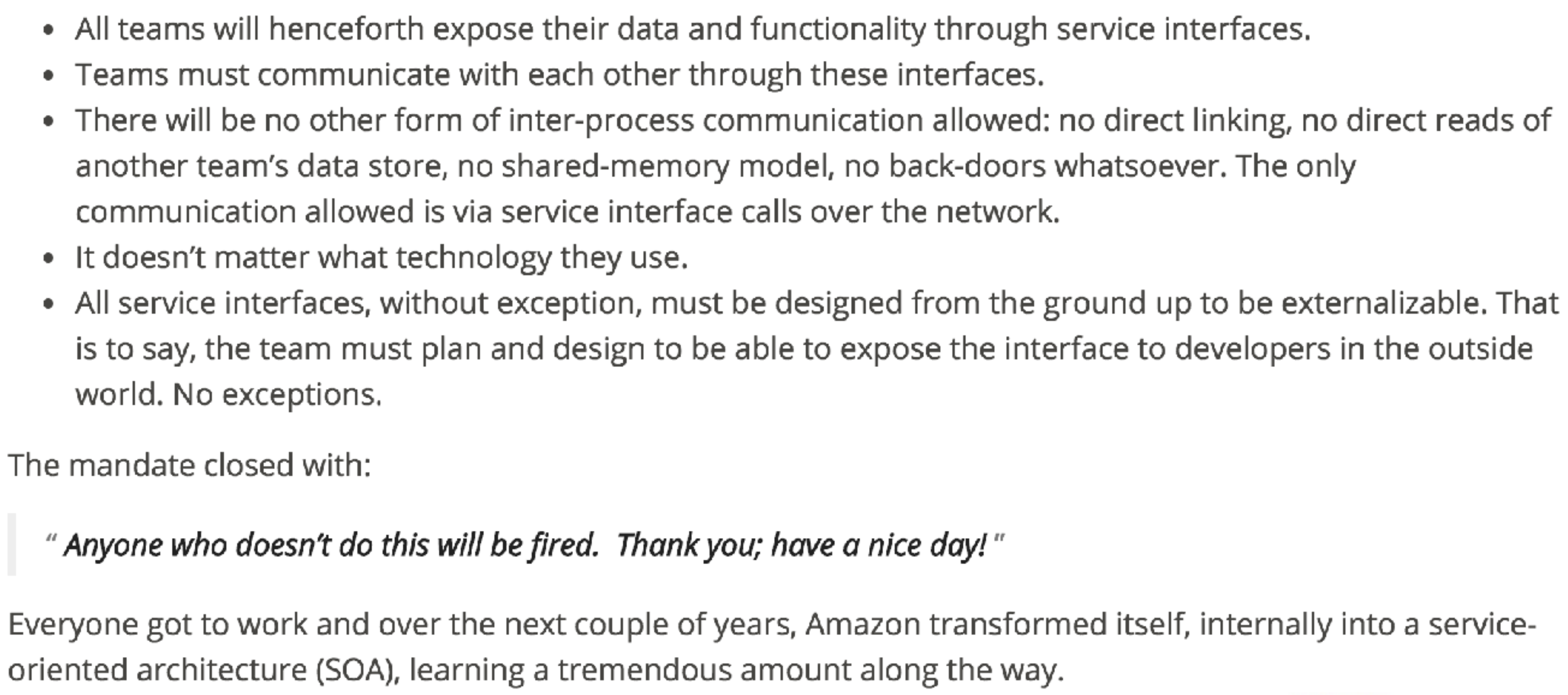
F5


Departure
Control

BACK IN 2004 (ISH)

amazon.com®

BACK IN 2004 (ISH)



- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they use.
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

The mandate closed with:

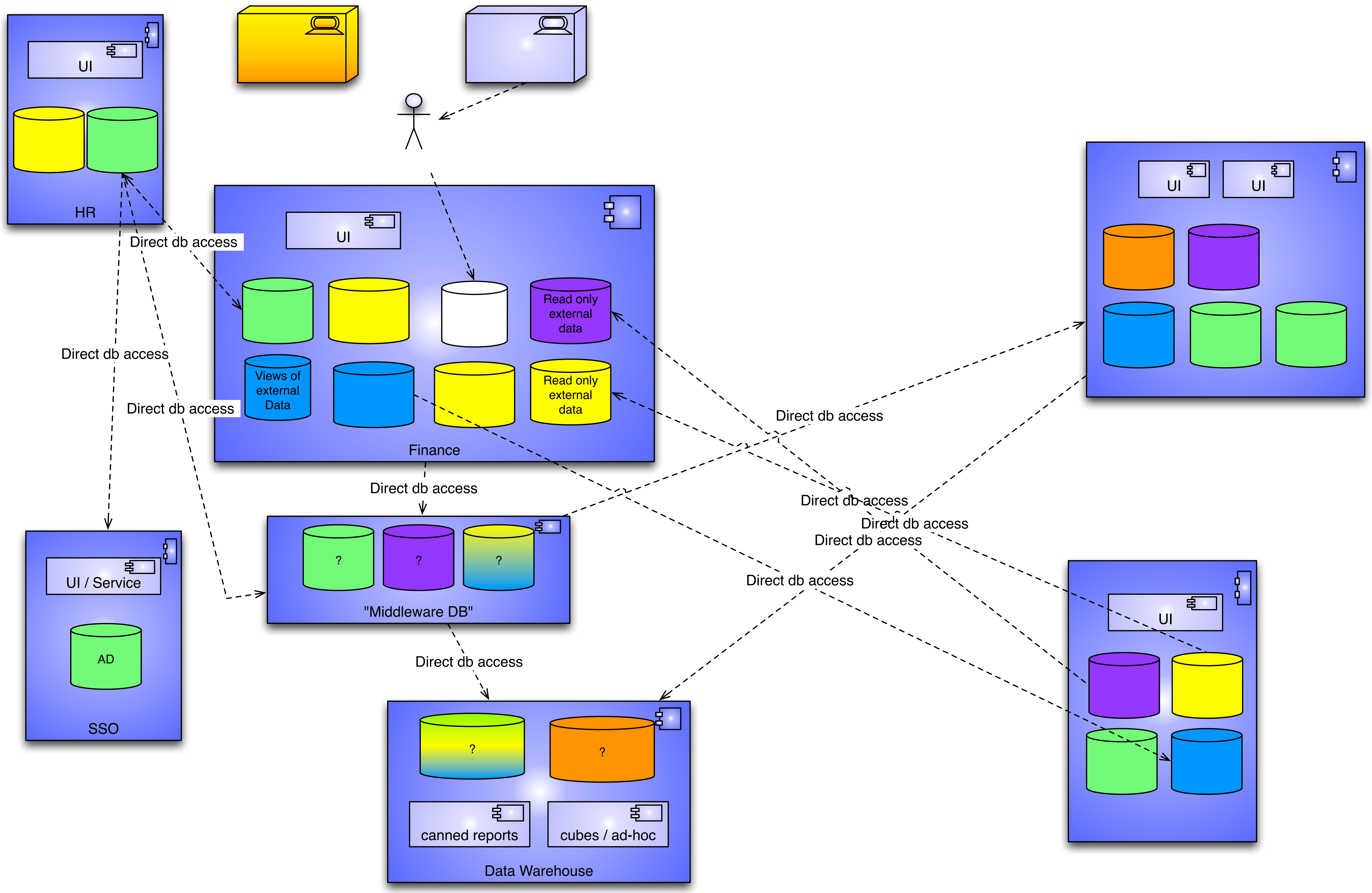
> *" Anyone who doesn't do this will be fired. Thank you; have a nice day! "*

Everyone got to work and over the next couple of years, Amazon transformed itself, internally into a service-oriented architecture (SOA), learning a tremendous amount along the way.

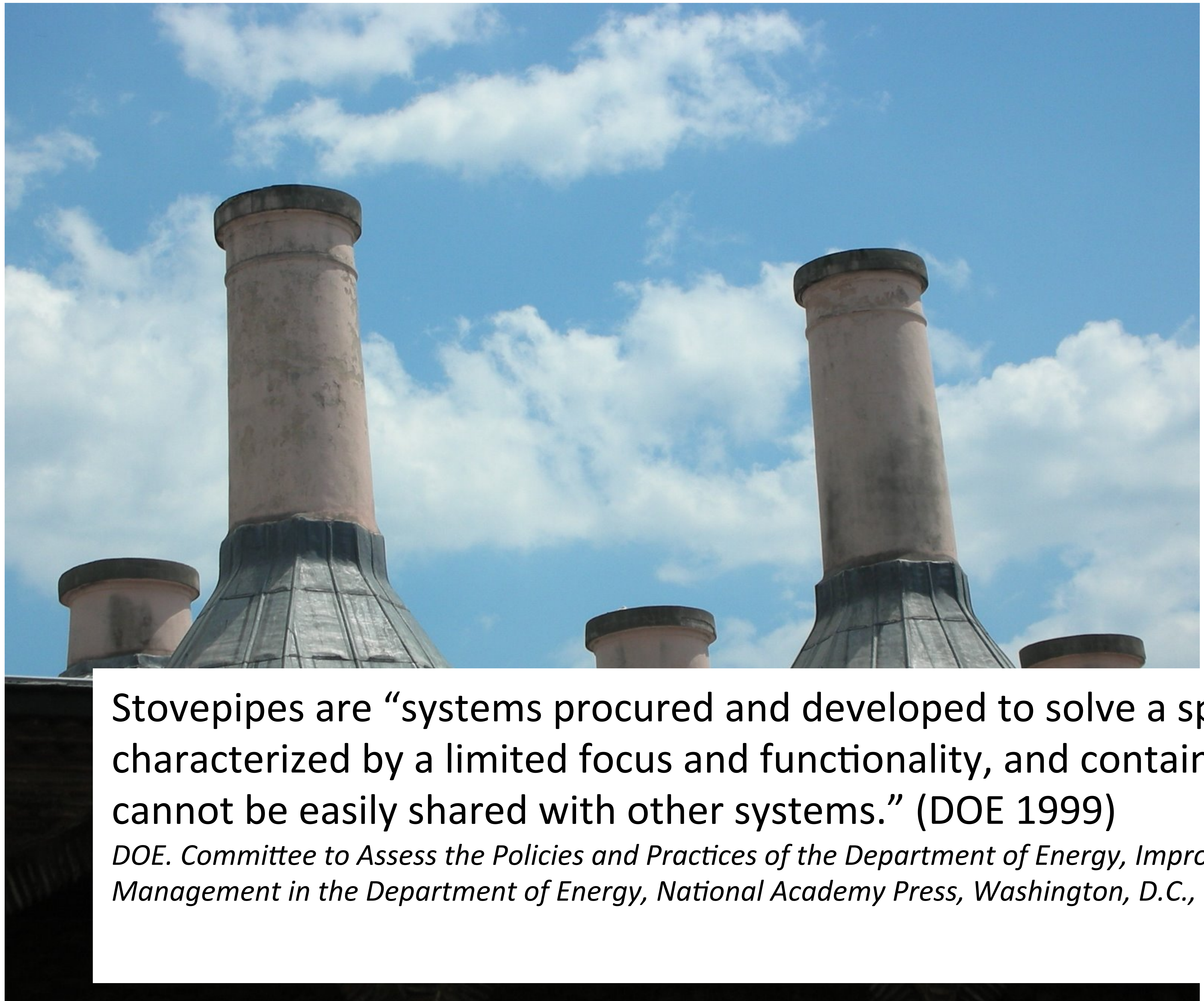
amazon.com®

ThoughtWorks®

<embarrassing>

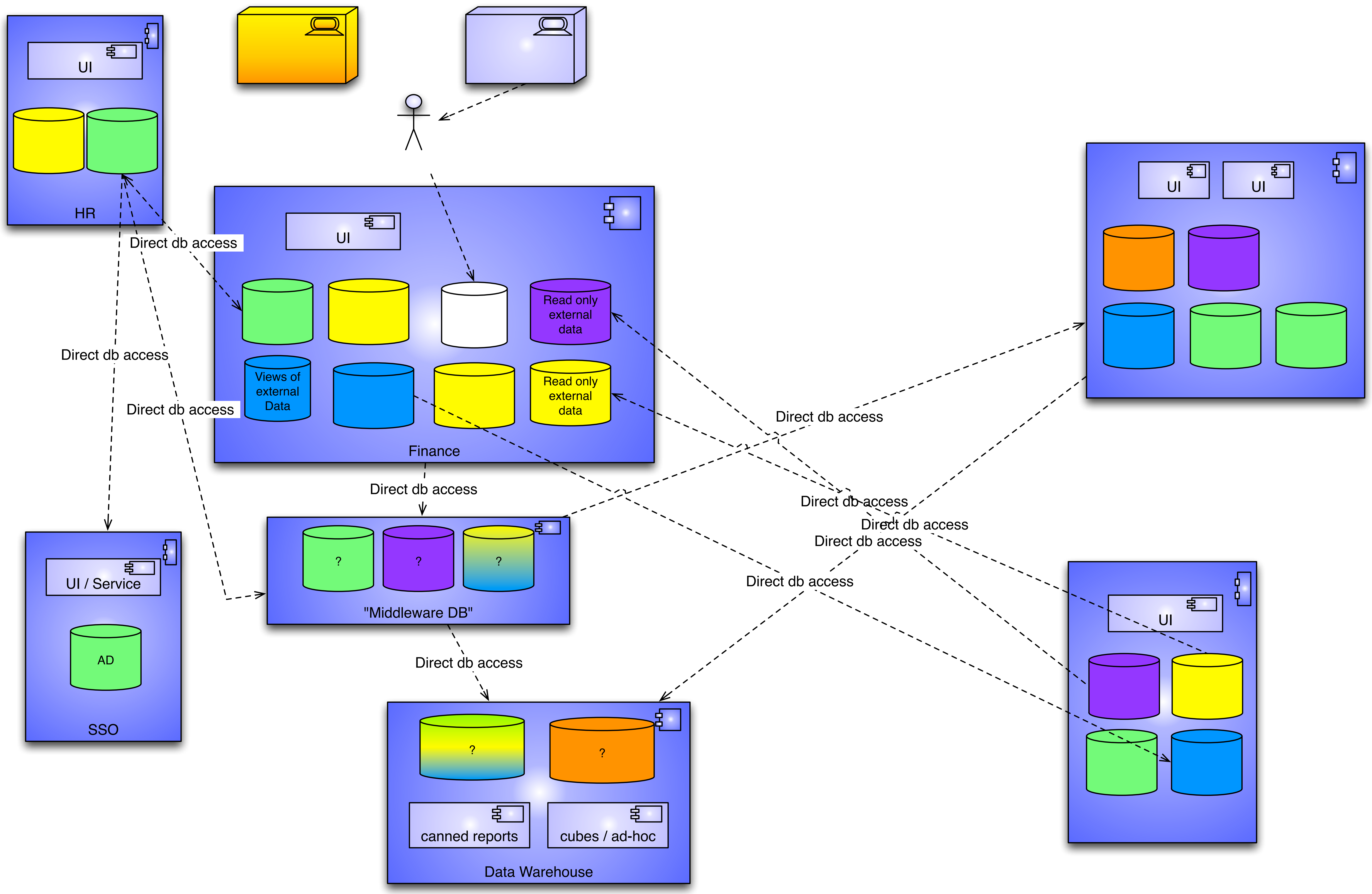

UI
HR
Direct db access
UI
Read only external data
Views of external Data
Read only external data
Finance
Direct db access
Direct db access
UI
UI
Direct db access
Direct db access
Direct db access
Direct db access
Direct db access
Direct db access
UI / Service
AD
SSO
?
?
?
"Middleware DB"
Direct db access
UI
?
?
canned reports
cubes / ad-hoc
Data Warehouse

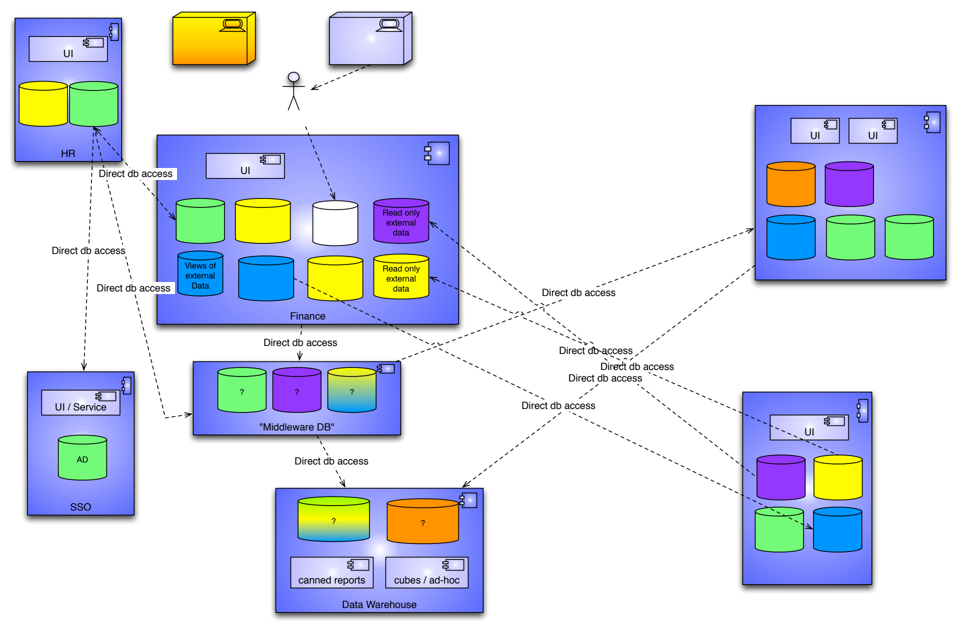
The stovepipe enterprise

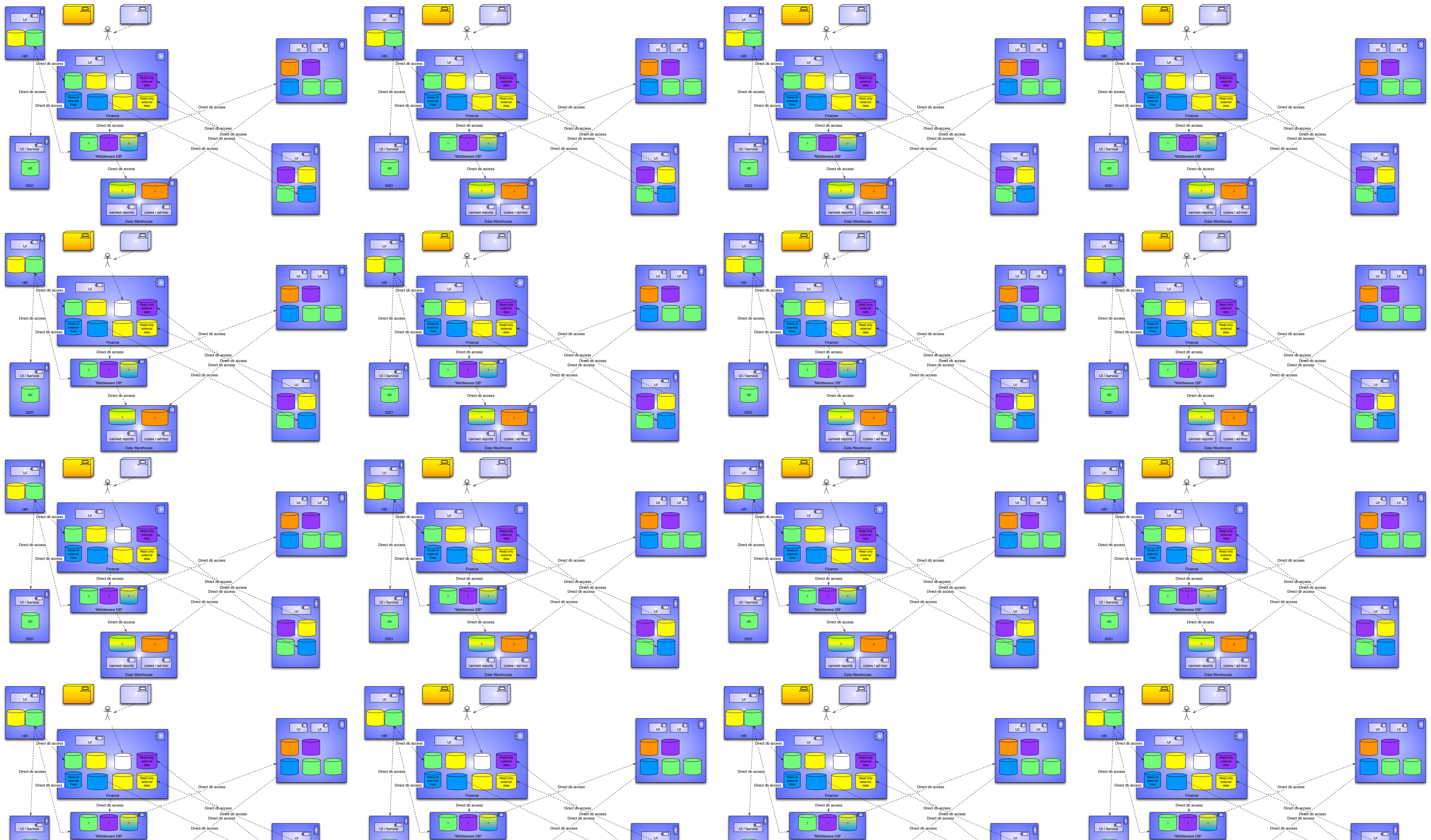


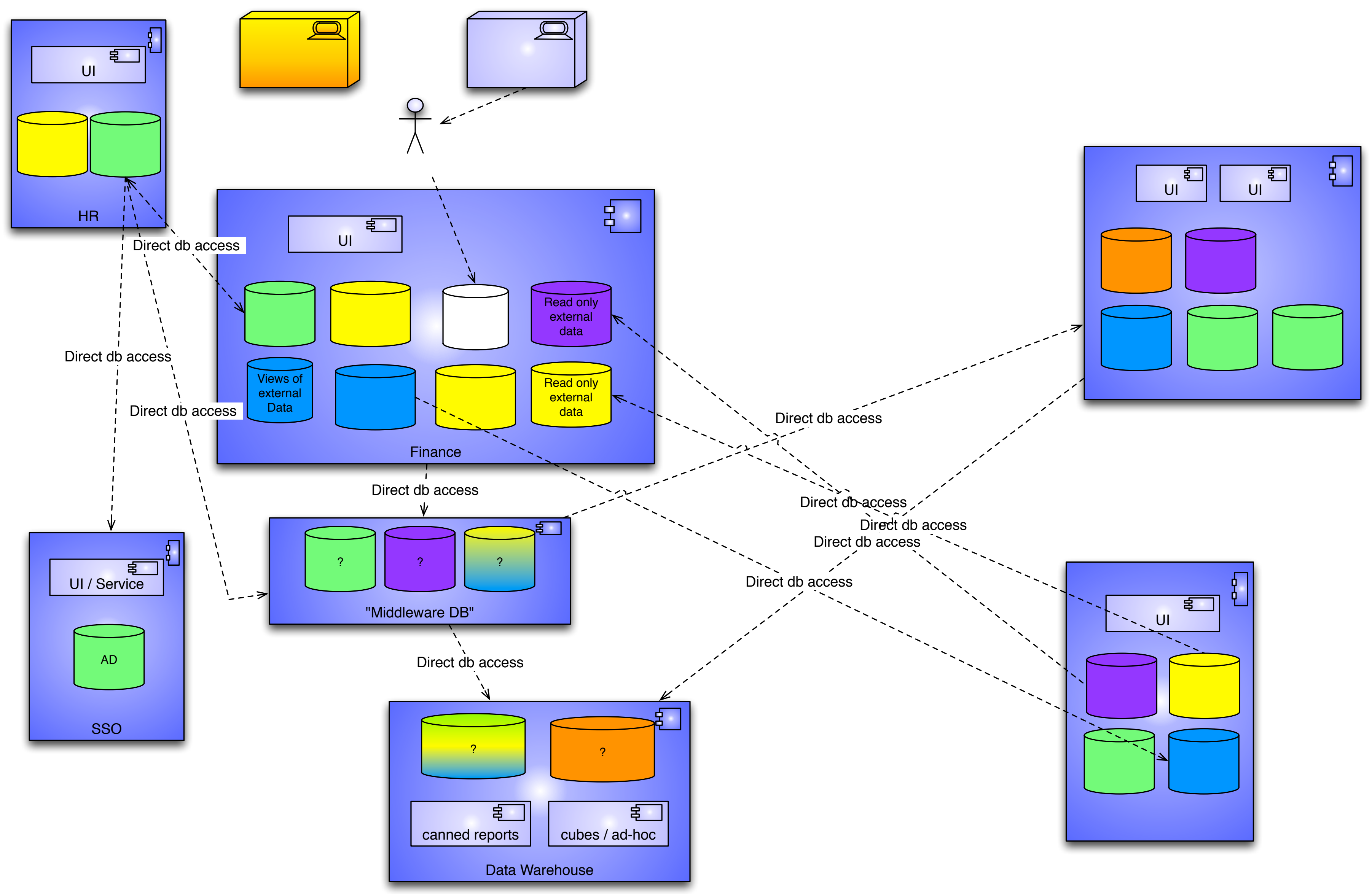
Stovepipes are “systems procured and developed to solve a specific problem, characterized by a limited focus and functionality, and containing data that cannot be easily shared with other systems.” (DOE 1999)

DOE. Committee to Assess the Policies and Practices of the Department of Energy, Improving Project Management in the Department of Energy, National Academy Press, Washington, D.C., 1999, page 133.


UI
HR
Direct db access
Direct db access
Direct db access
UI
Read only external data
Views of external Data
Read only external data
Finance
Direct db access
?
?
?
"Middleware DB"
UI / Service
AD
SSO
Direct db access
?
?
canned reports
cubes / ad-hoc
Data Warehouse
UI
UI
Direct db access
Direct db access
Direct db access
Direct db access
Direct db access
UI


UI
HR
Direct db access
Direct db access
Direct db access
UI
Read only external data
Views of external Data
Read only external data
Finance
Direct db access
UI / Service
AD
SSO
?
?
?
"Middleware DB"
Direct db access
?
?
canned reports
cubes / ad-hoc
Data Warehouse
Direct db access
Direct db access
Direct db access
Direct db access
Direct db access
UI
UI
UI


UI
HR
Direct db access
Direct db access
Direct db access
UI
Read only external data
Views of external Data
Read only external data
Finance
Direct db access
UI / Service
AD
SSO
?
?
?
"Middleware DB"
Direct db access
?
?
canned reports
cubes / ad-hoc
Data Warehouse
UI
UI
Direct db access
Direct db access
Direct db access
Direct db access
Direct db access
UI



Logic scattered all over the place

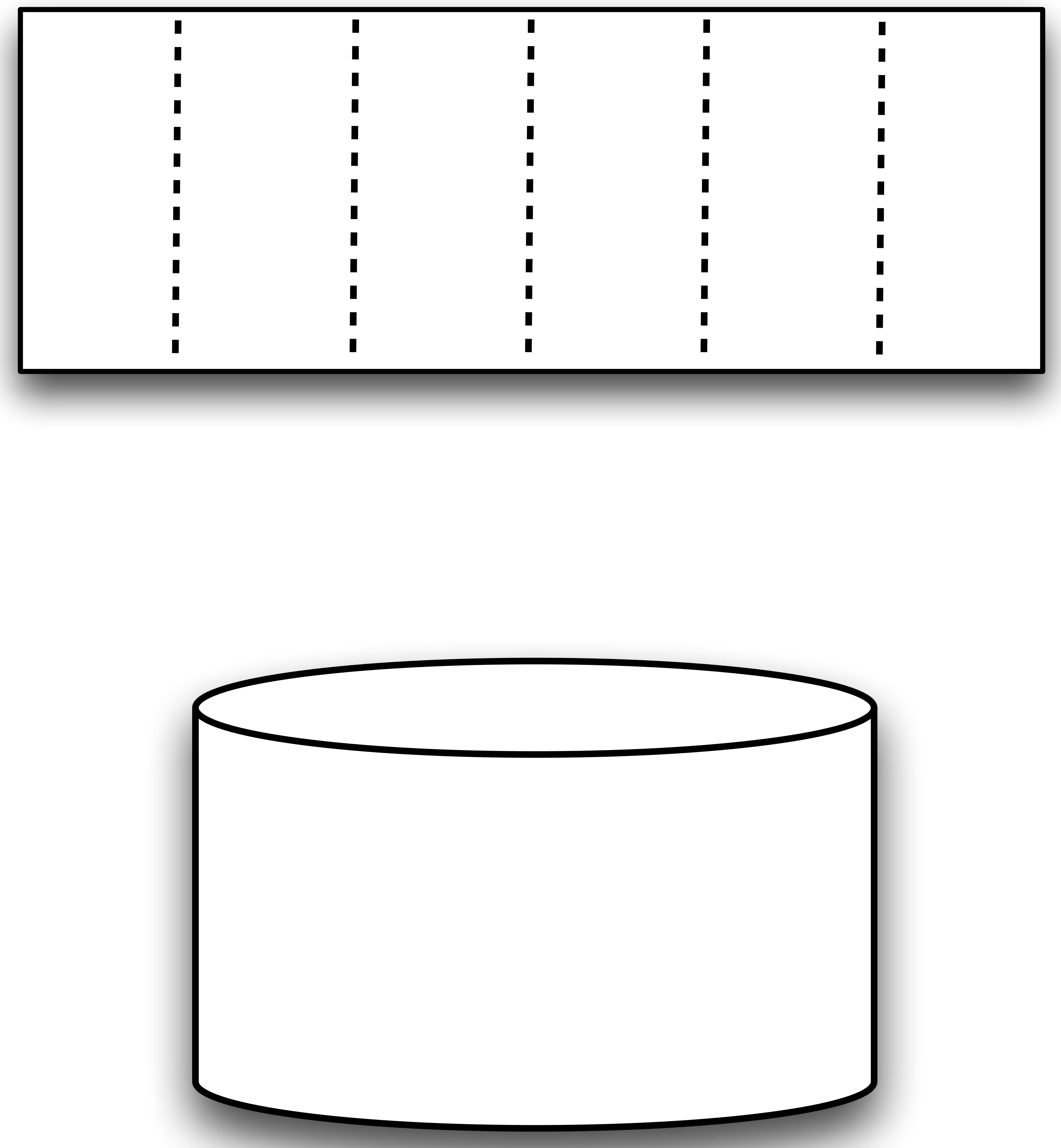
Data scattered all over the place

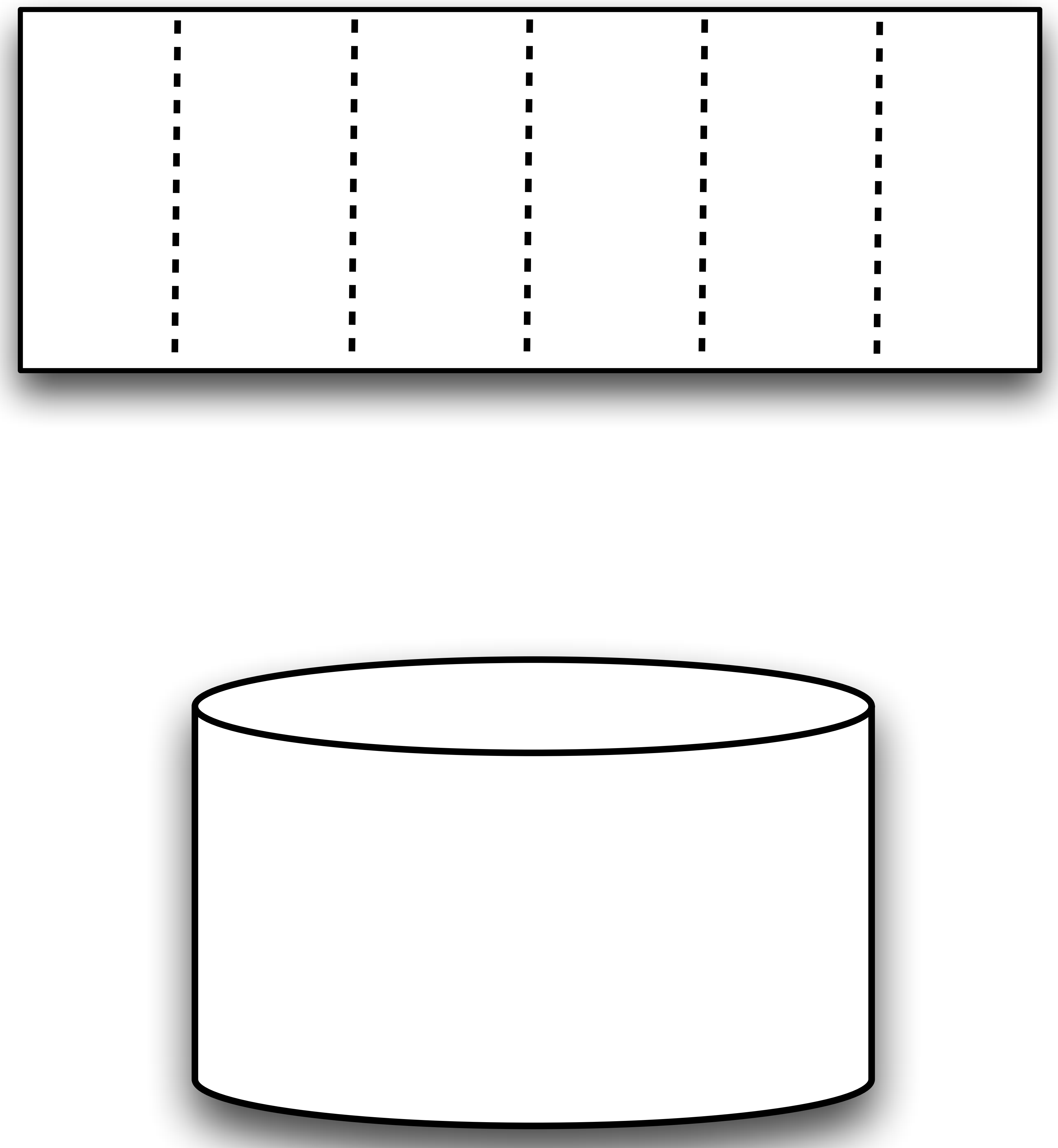
Difficult to predict the effect of changes

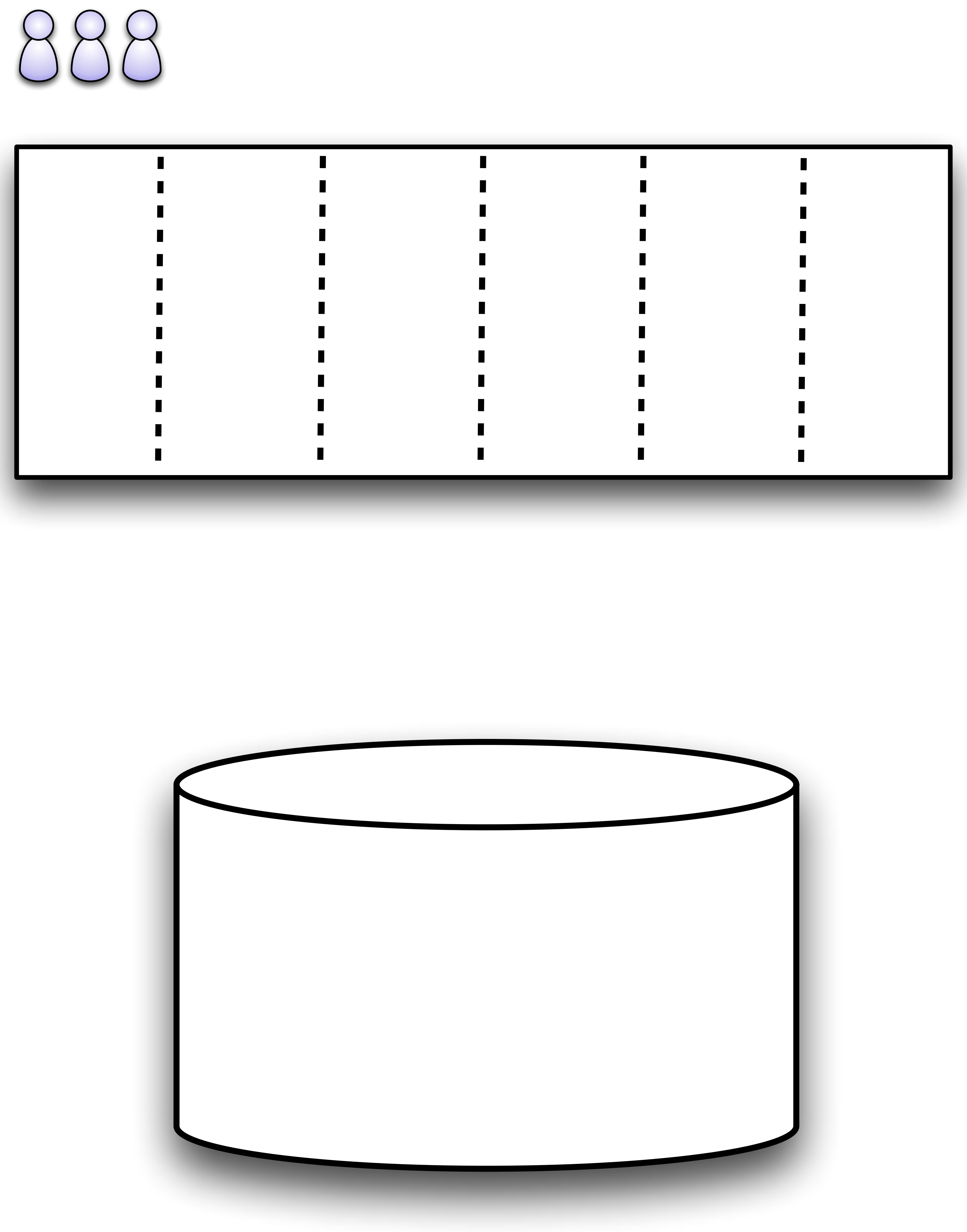
Where are the sources of truth?

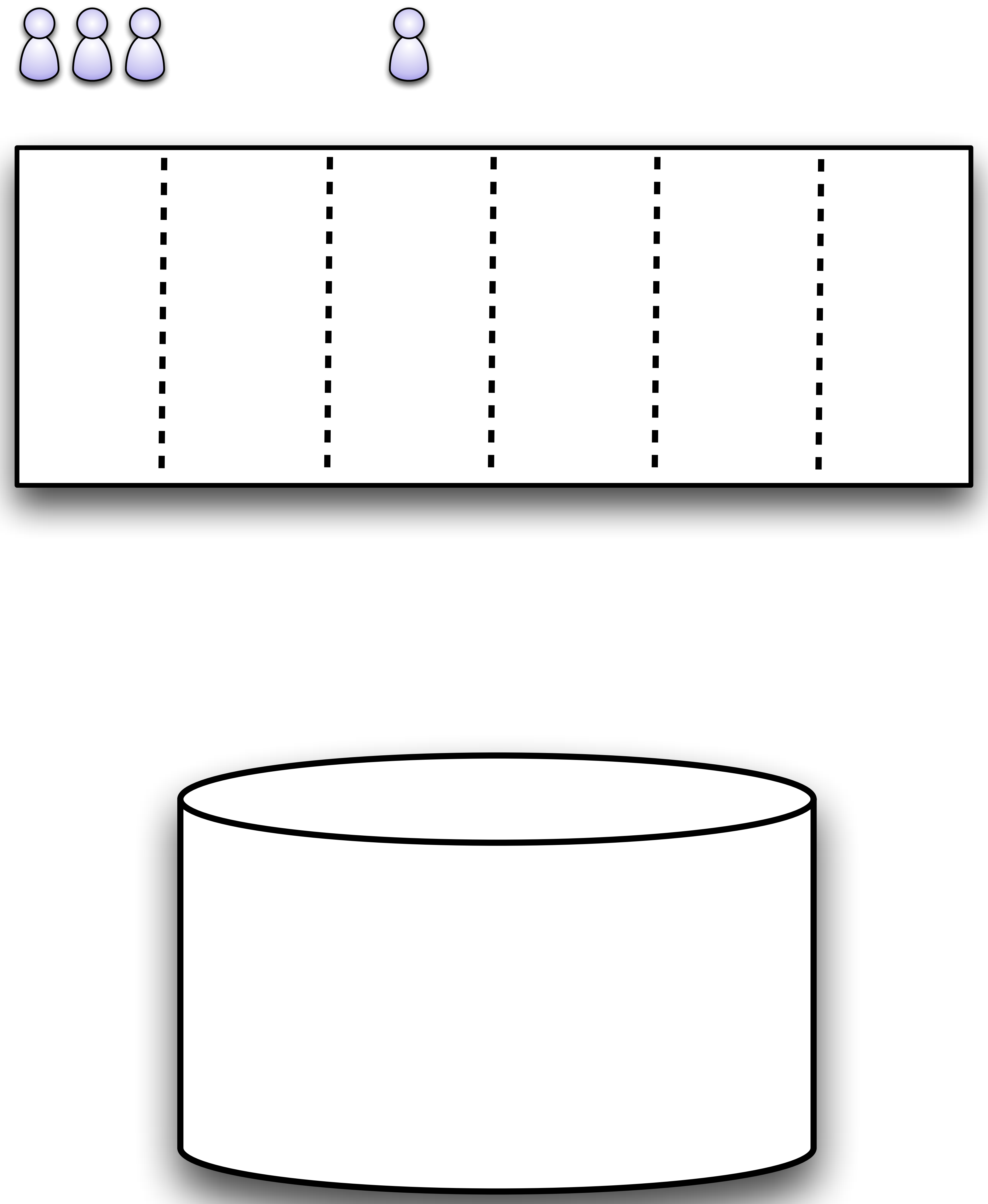
BI / MI almost impossible to get at

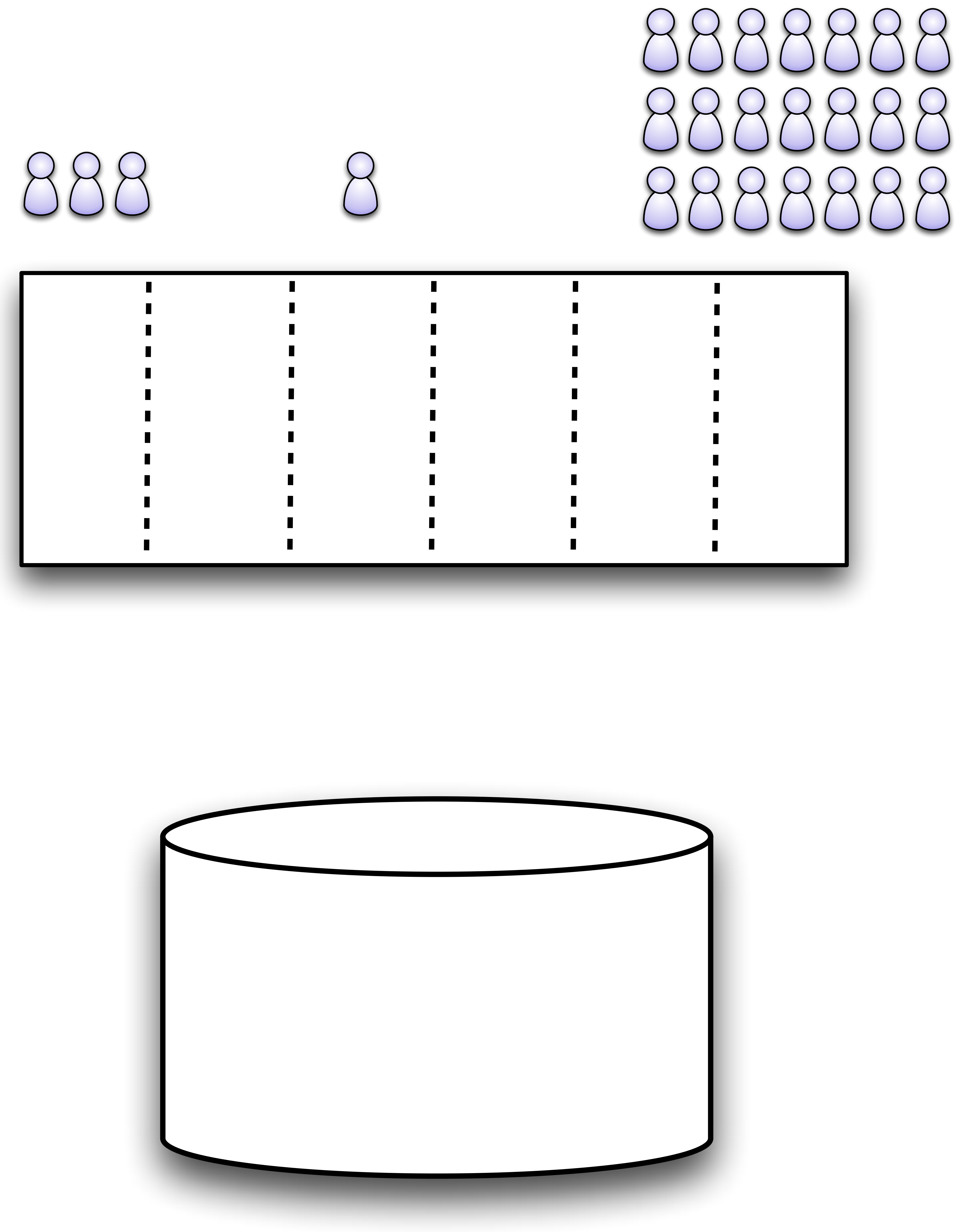
Insurance - 2011

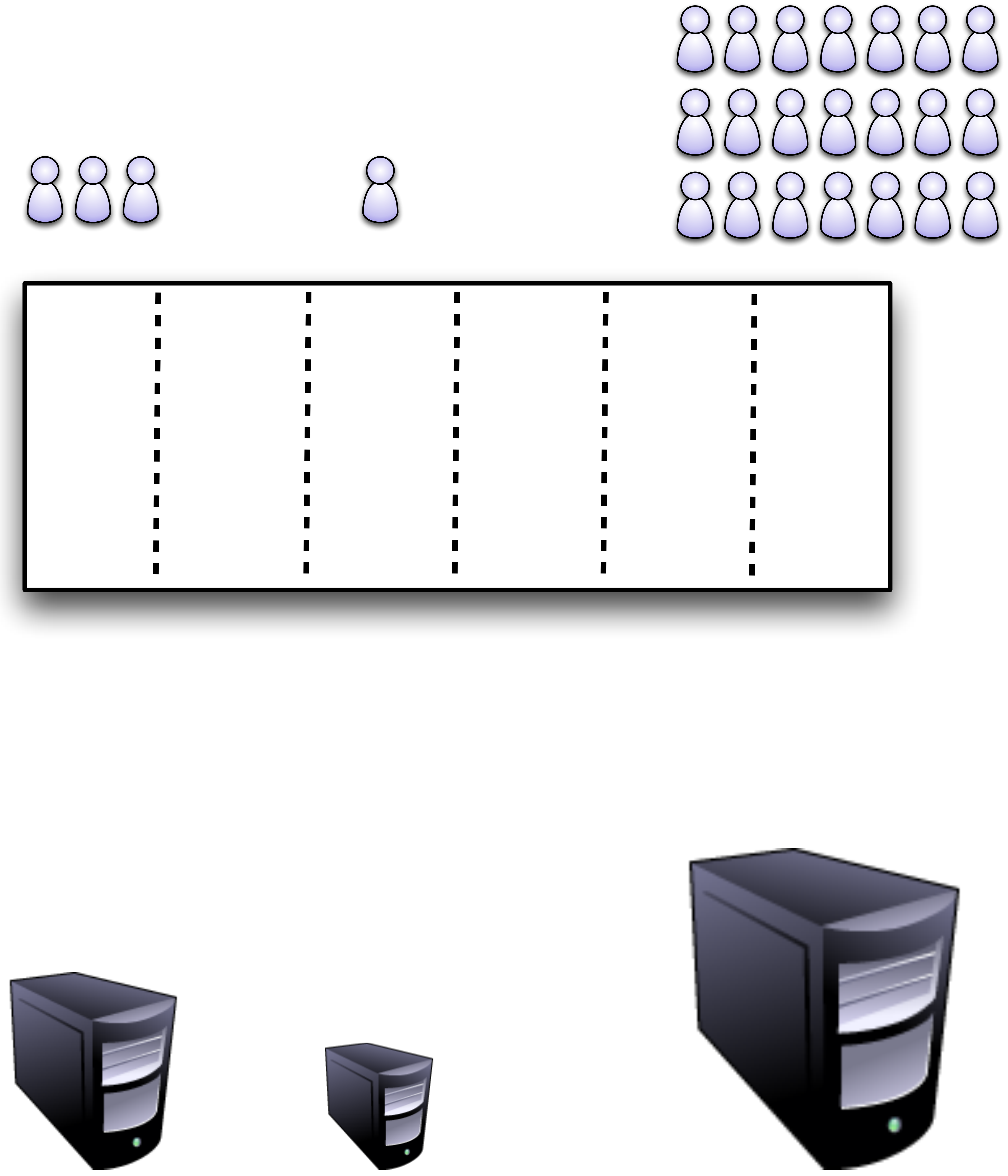


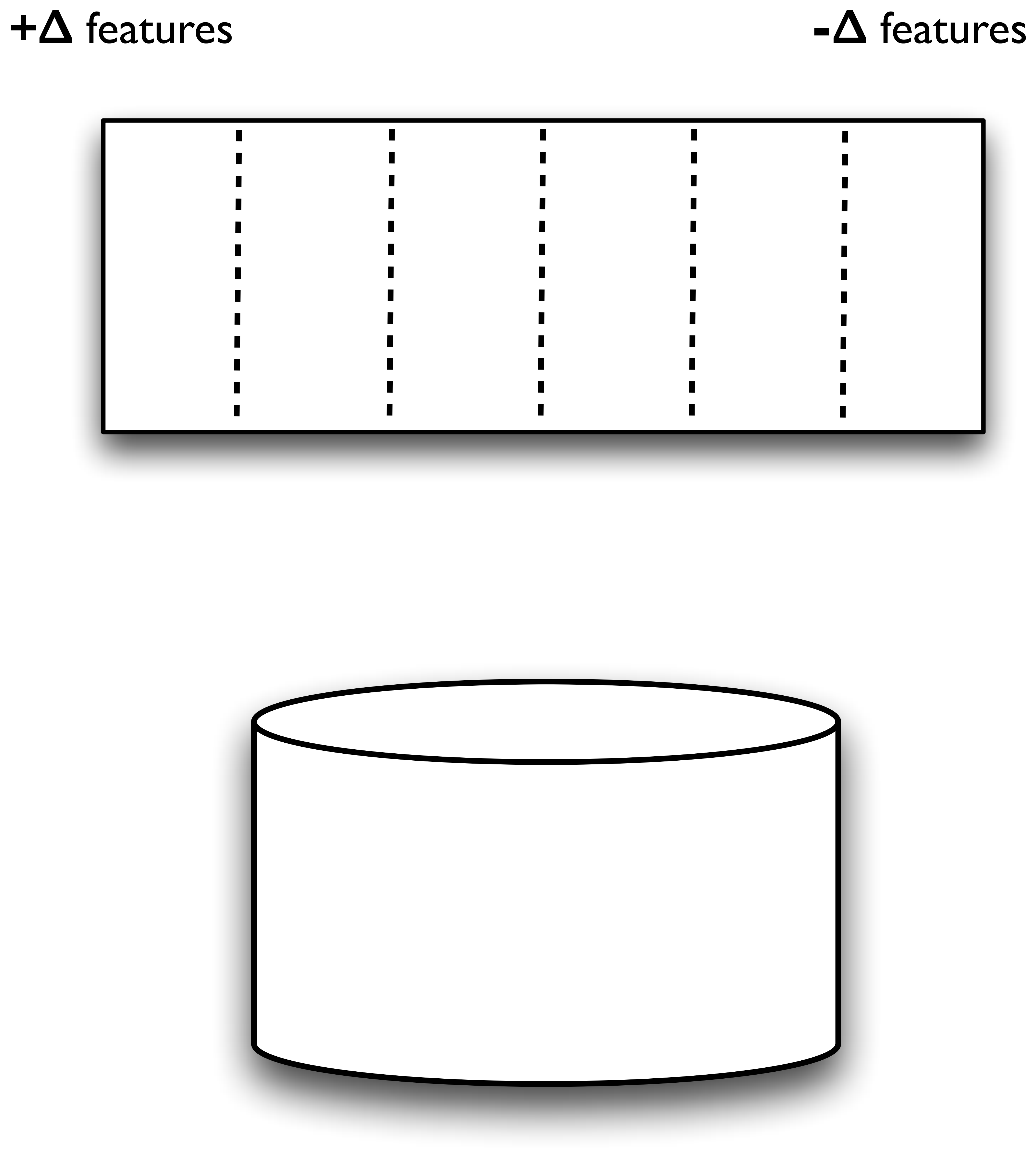




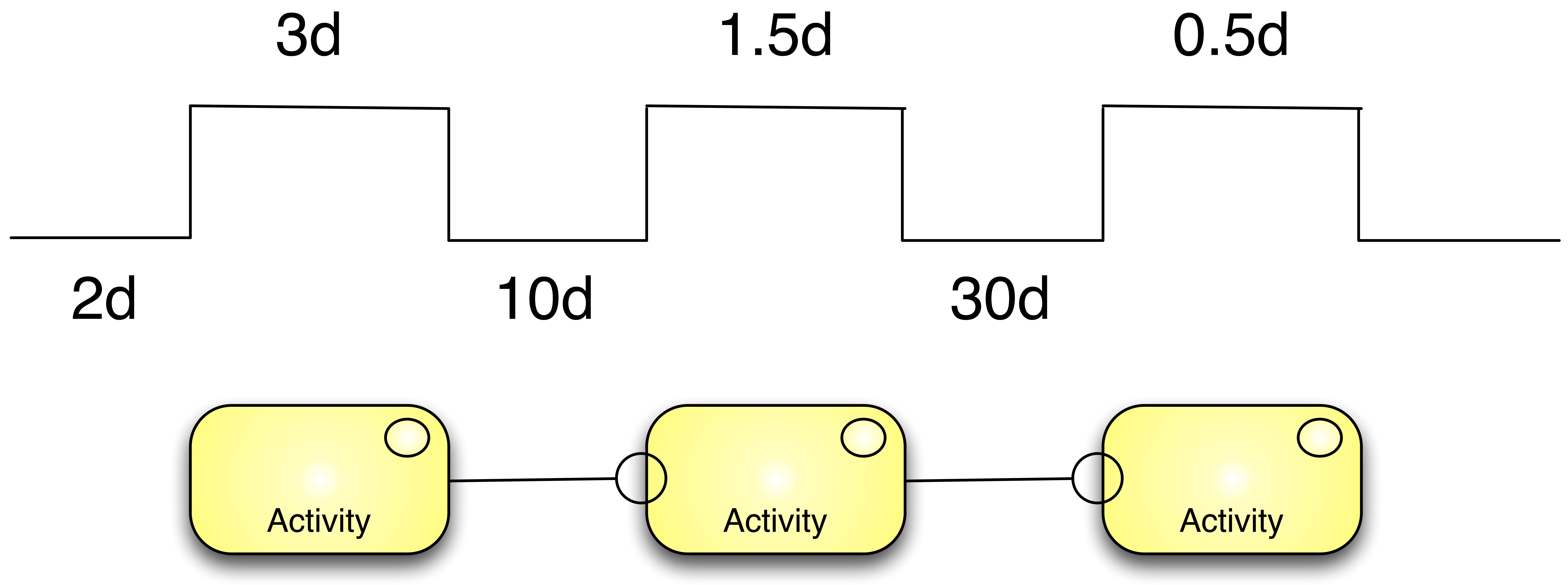







+Δ features
-Δ features

extremely long lead times to return on capital



Isn’t there a better way of spending my money?

Can’t we build systems that are:

cheap to replace

deployable on demand

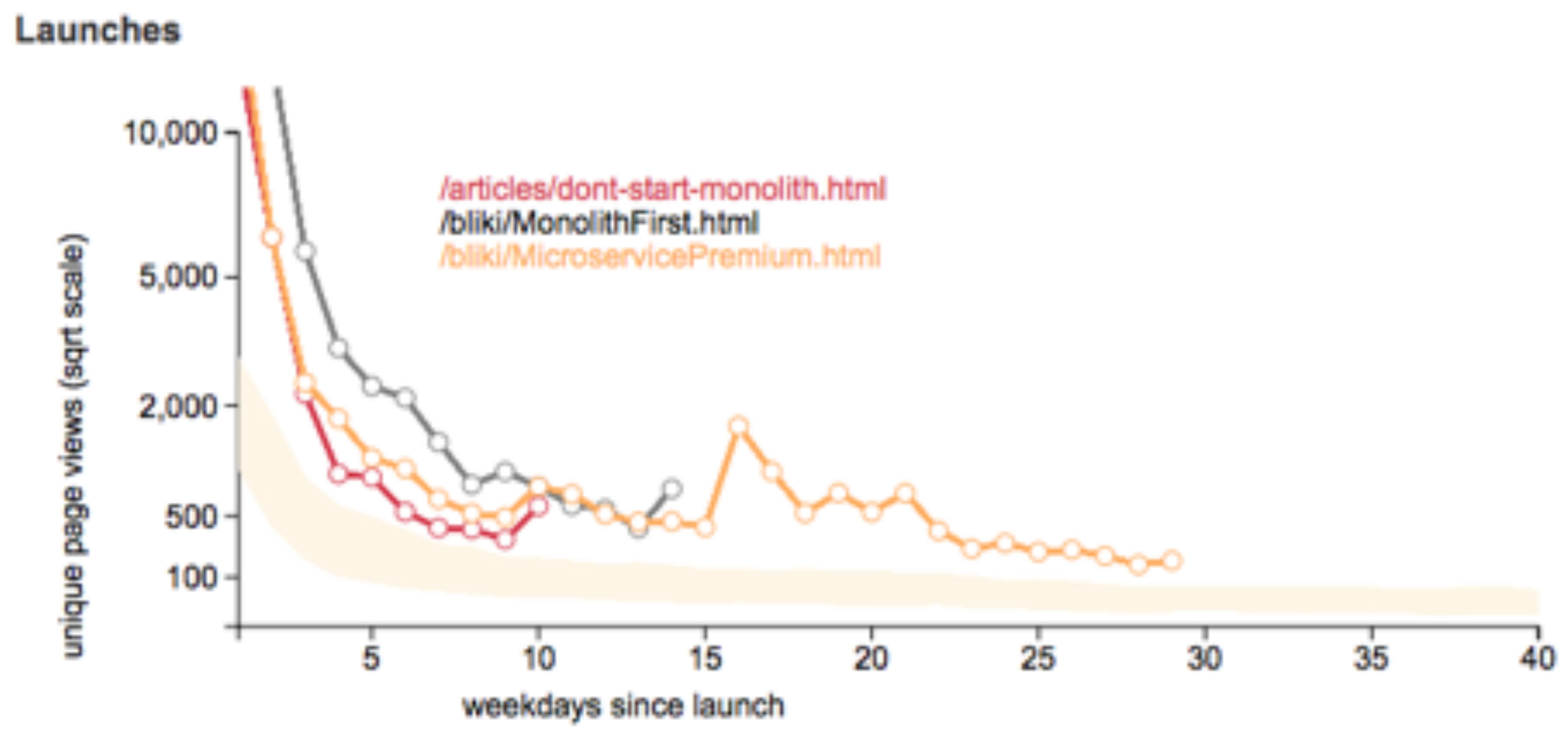
resilient on imperfect networks

Part the Third

The bankers nightmare

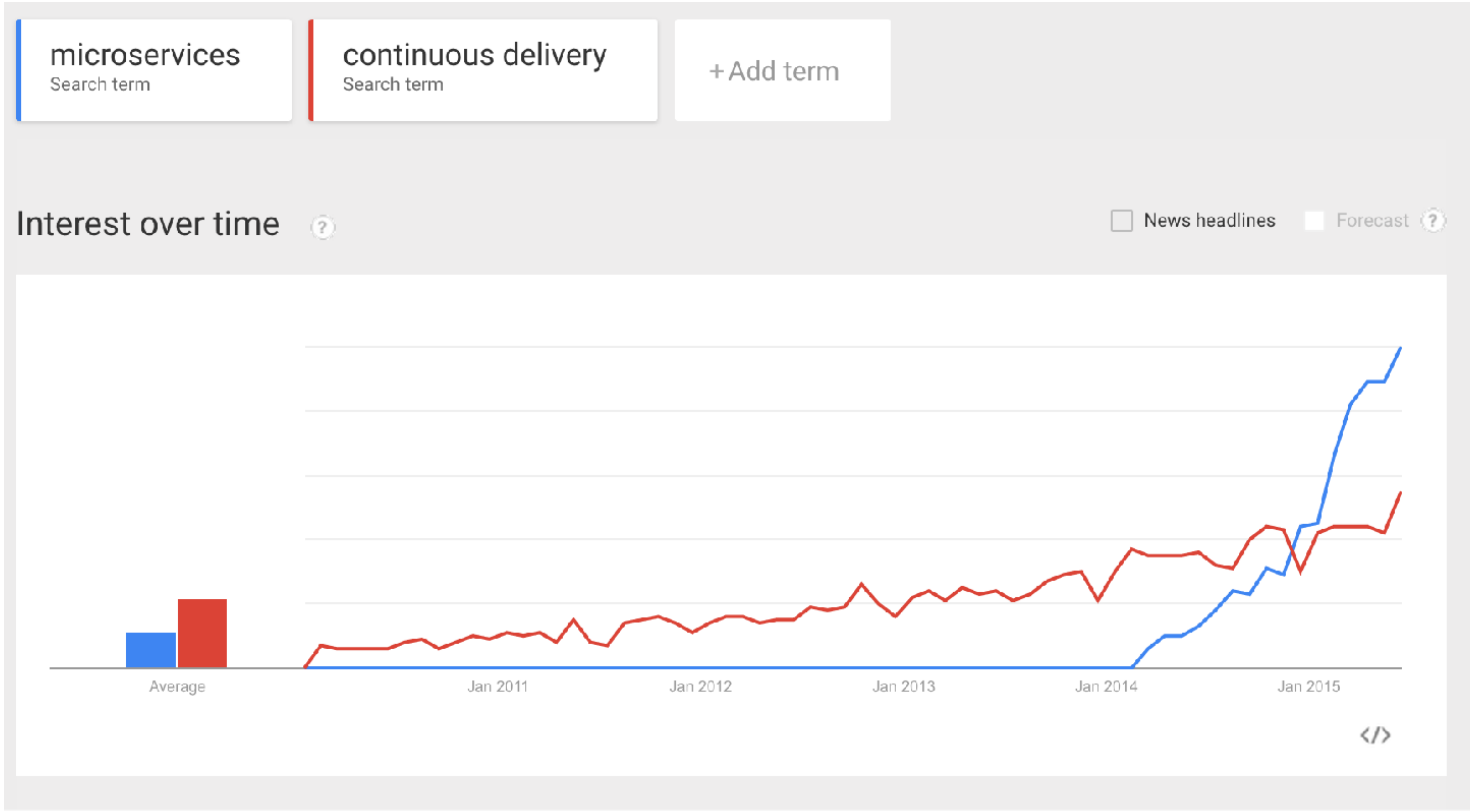
“They roused him with muffins—they roused him with ice—
They roused him with mustard and cress—
They roused him with jam and judicious advice—
They set him conundrums to guess.

“So our core microservice article got 45,144 unique page views last month, and is currently running at 1837 per day” @martin



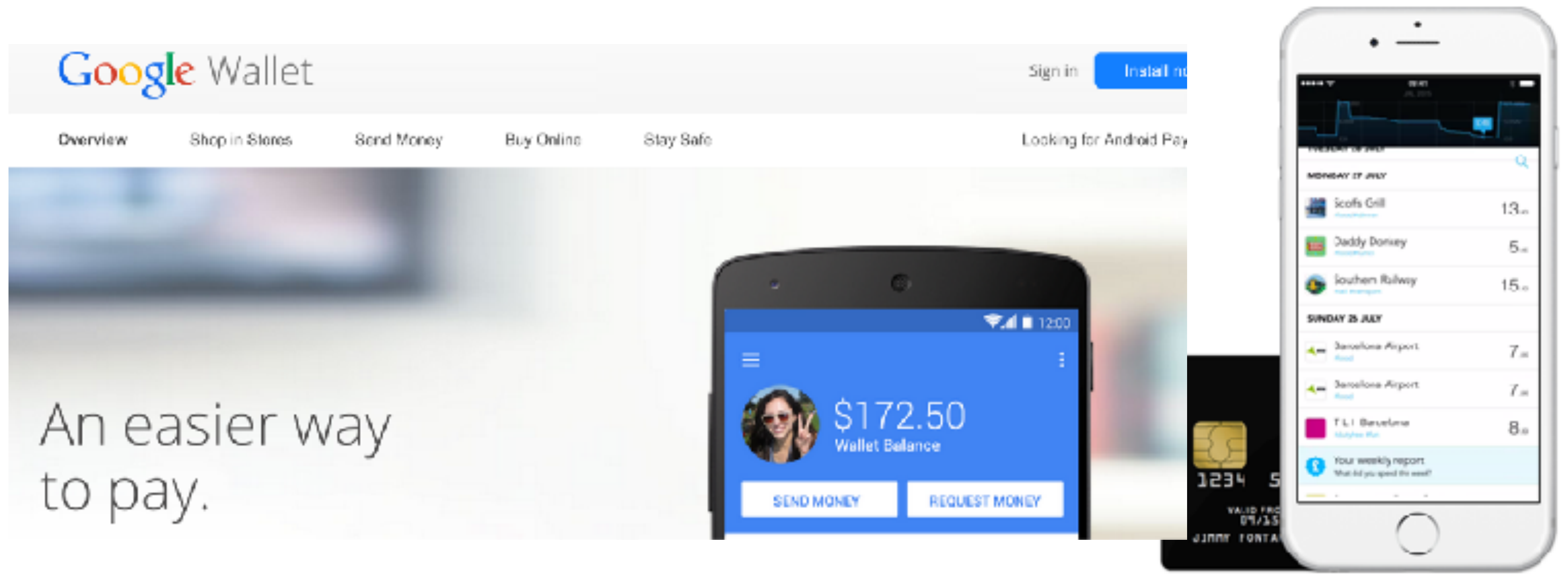
clear

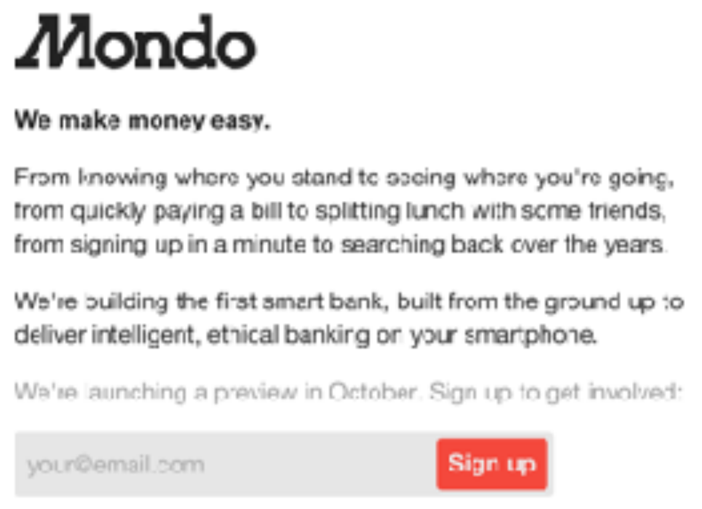
path	plot	date	total 7 days	total 28 days	peak day	recent median
/articles/doctor-who.html	plot	2015-06-19			1346	1346
/articles/tor-for-technologists.html	plot	2015-06-15	8378		4121	786
/articles/dont-start-monolith.html	plot	2015-06-09	24870		13573	399
/bliki/MonolithFirst.html	plot	2015-06-03	67681		39092	602
/bliki/Yagni.html	plot	2015-05-26	50841	63239	28326	299
/bliki/MicroservicePremium.html	plot	2015-05-13	29873	42180	16292	229
/bliki/CodeAsDocumentation.html	plot	2015-03-25	5618	8778	2860	19


microservices
Search term
continuous delivery
Search term
+ Add term
Interest over time
News headlines
Forecast
Average
Jan 2011
Jan 2012
Jan 2013
Jan 2014
Jan 2015

Why?

FINTECH IS COMING



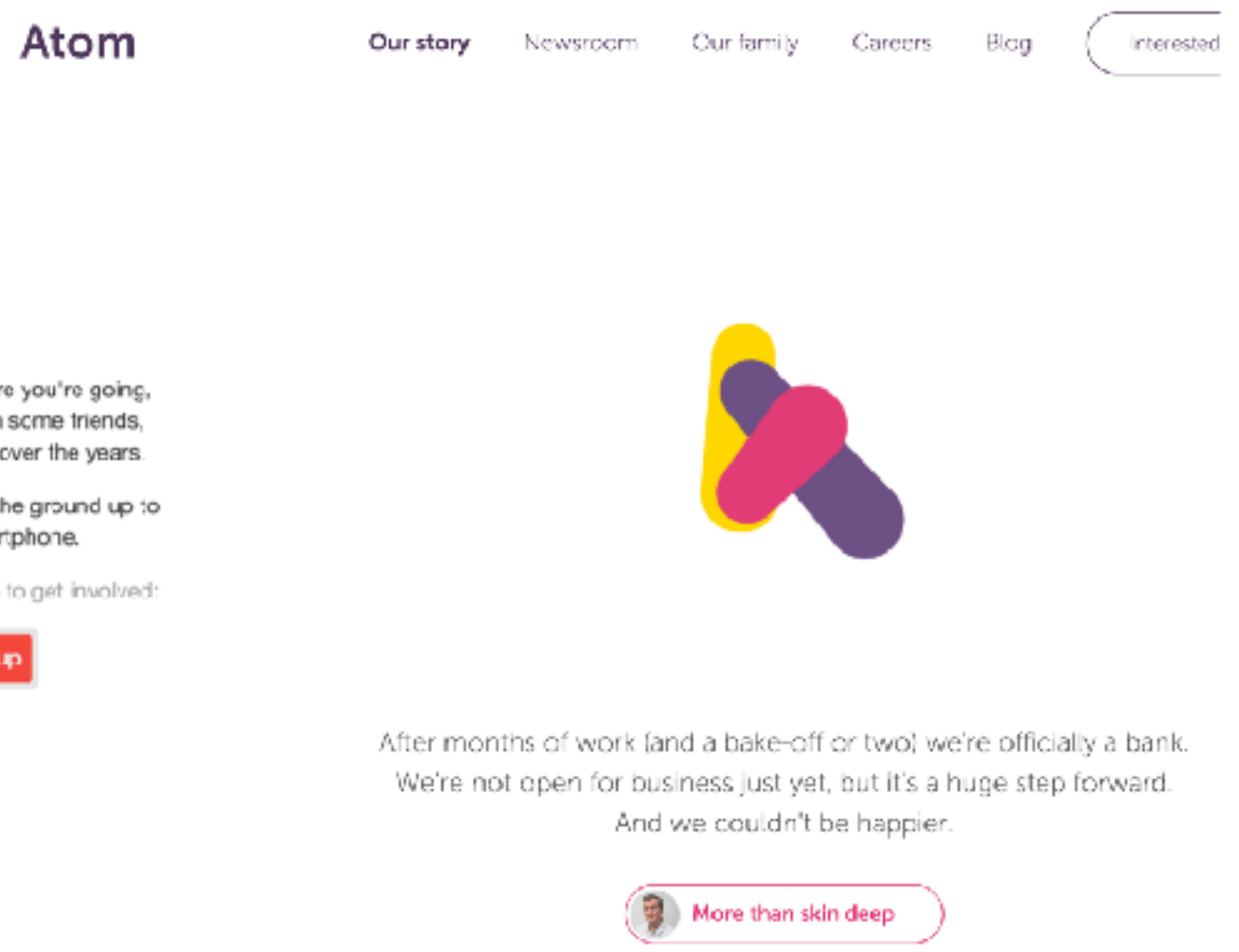

Mondo

We make money easy.

From knowing where you stand to seeing where you're going, from quickly paying a bill to splitting lunch with some friends, from signing up in a minute to searching back over the years.

We're building the first smart bank, built from the ground up to deliver intelligent, ethical banking on your smartphone.

We're launching a preview in October. Sign up to get involved:



After months of work (and a bake-off or two) we're officially a bank. We're not open for business just yet, but it's a huge step forward. And we couldn't be happier.

Pay

Microservice envy

HOLD ?

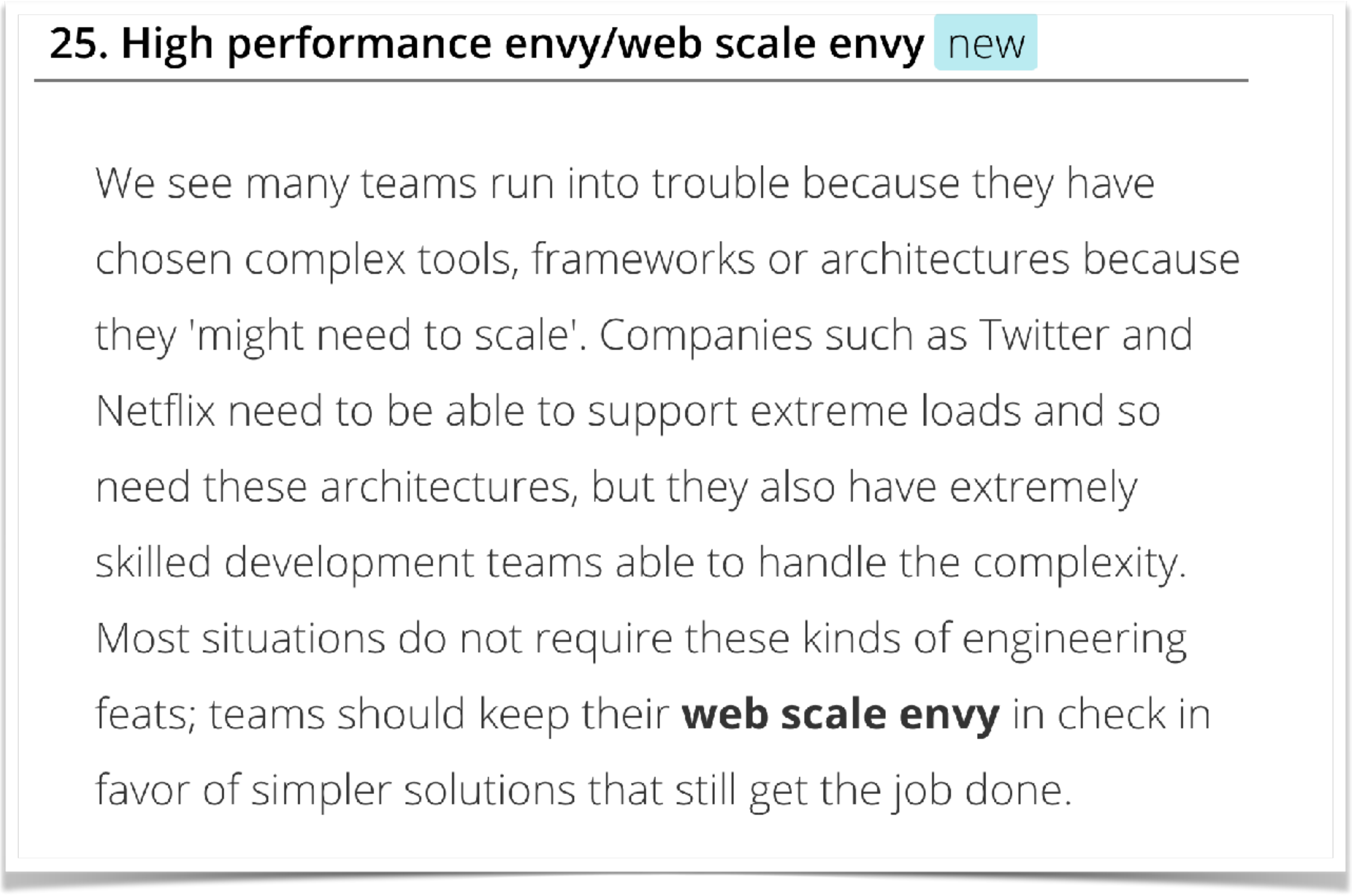
We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.

Techniques

Microservice envy

HOLD

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.



25. High performance envy/web scale envy new

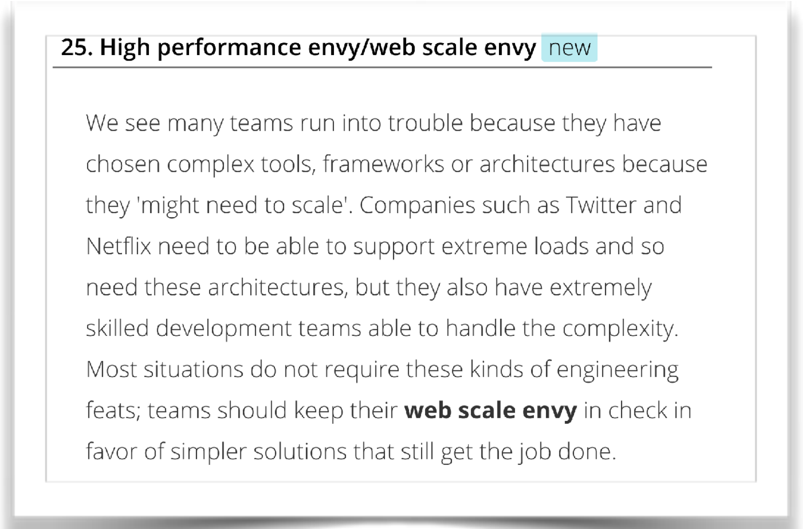
We see many teams run into trouble because they have chosen complex tools, frameworks or architectures because they 'might need to scale'. Companies such as Twitter and Netflix need to be able to support extreme loads and so need these architectures, but they also have extremely skilled development teams able to handle the complexity. Most situations do not require these kinds of engineering feats; teams should keep their **web scale envy** in check in favor of simpler solutions that still get the job done.

Techniques

Microservice envy

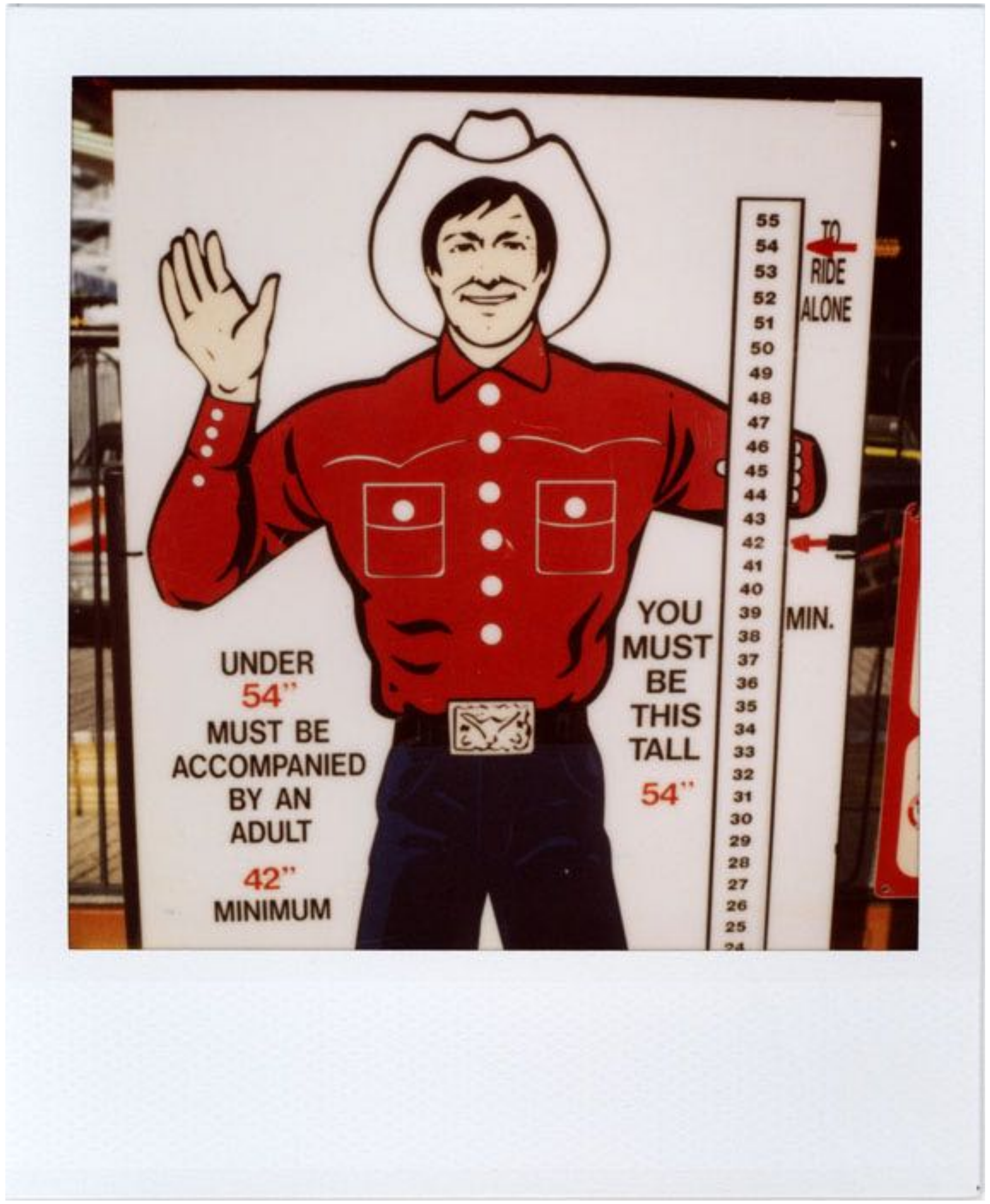
HOLD

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.



25. High performance envy/web scale envy new

We see many teams run into trouble because they have chosen complex tools, frameworks or architectures because they 'might need to scale'. Companies such as Twitter and Netflix need to be able to support extreme loads and so need these architectures, but they also have extremely skilled development teams able to handle the complexity. Most situations do not require these kinds of engineering feats; teams should keep their **web scale envy** in check in favor of simpler solutions that still get the job done.



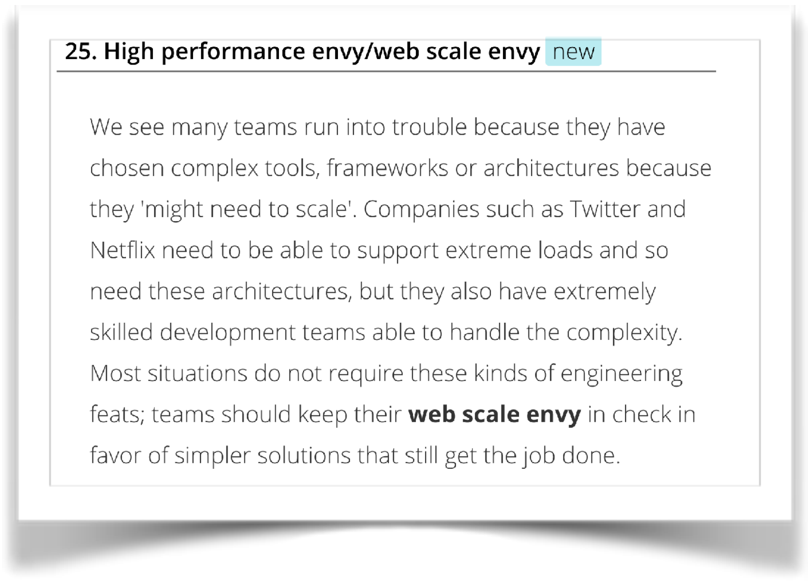
https://www.flickr.com/photos/futurowoman/2923992303

Techniques

Microservice envy

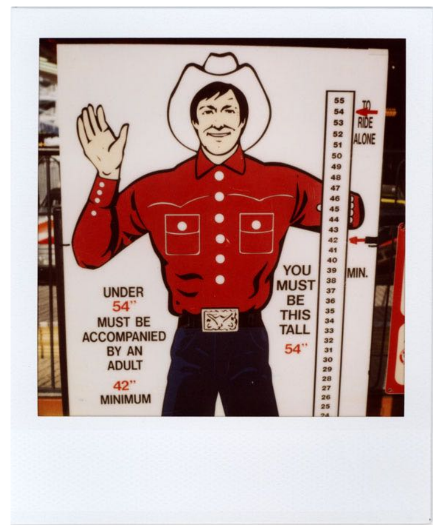
HOLD

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.


25. High performance envy/web scale envy new

We see many teams run into trouble because they have chosen complex tools, frameworks or architectures because they 'might need to scale'. Companies such as Twitter and Netflix need to be able to support extreme loads and so need these architectures, but they also have extremely skilled development teams able to handle the complexity. Most situations do not require these kinds of engineering feats; teams should keep their **web scale envy** in check in favor of simpler solutions that still get the job done.

DOCKER DOCKER DOCKER

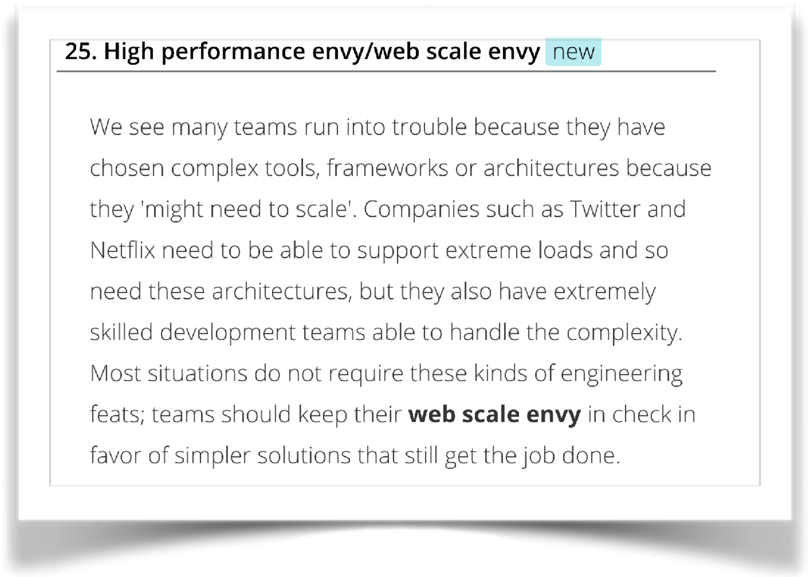

UNDER 54" MUST BE ACCOMPANIED BY AN ADULT 42" MINIMUM
YOU MUST BE THIS TALL 54"
TO RIDE ALONE
MIN.

Techniques

Microservice envy

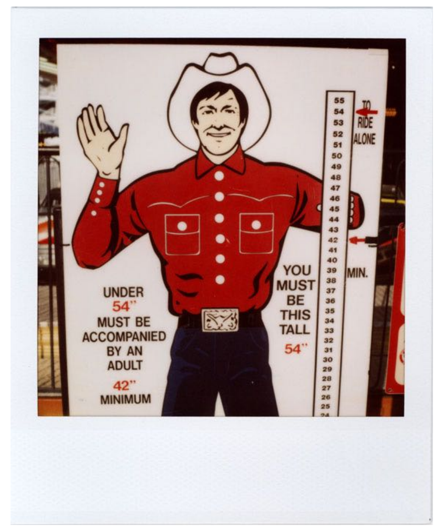
HOLD

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.



25. High performance envy/web scale envy new

We see many teams run into trouble because they have chosen complex tools, frameworks or architectures because they 'might need to scale'. Companies such as Twitter and Netflix need to be able to support extreme loads and so need these architectures, but they also have extremely skilled development teams able to handle the complexity. Most situations do not require these kinds of engineering feats; teams should keep their **web scale envy** in check in favor of simpler solutions that still get the job done.


UNDER 54" MUST BE ACCOMPANIED BY AN ADULT 42" MINIMUM
YOU MUST BE THIS TALL 54"

DOCKER DOCKER DOCKER

Monitoring

Organisational Structure

Deployment

Integration

Testing

Architectural Safety

DOCKER DOCKER DOCKER DOCKER DOCKER D

DOCKER DOCKER DOCKER DOCKER DOCKER D

DOCKER DOCKER DOCKER DOCKER DOCKER D

DOCKER DOCKER DOCKER DOCKER DOCKER D

DOCKER DOCKER DOCKER DOCKER DOCKER D

DOCKER DOCKER DOCKER DOCKER DOCKER D

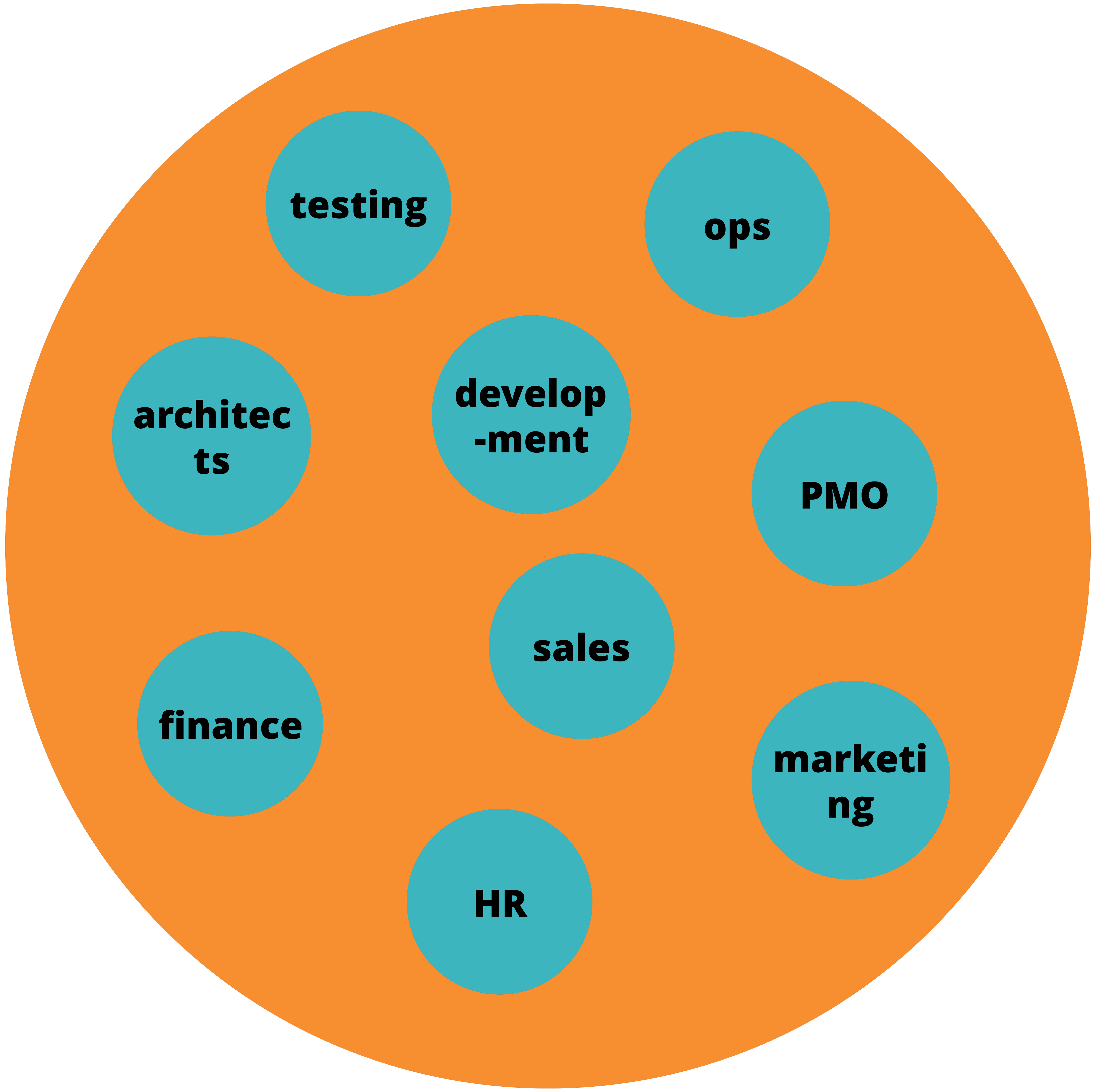
DOCKER DOCKER DOCKER DOCKER DOCKER D

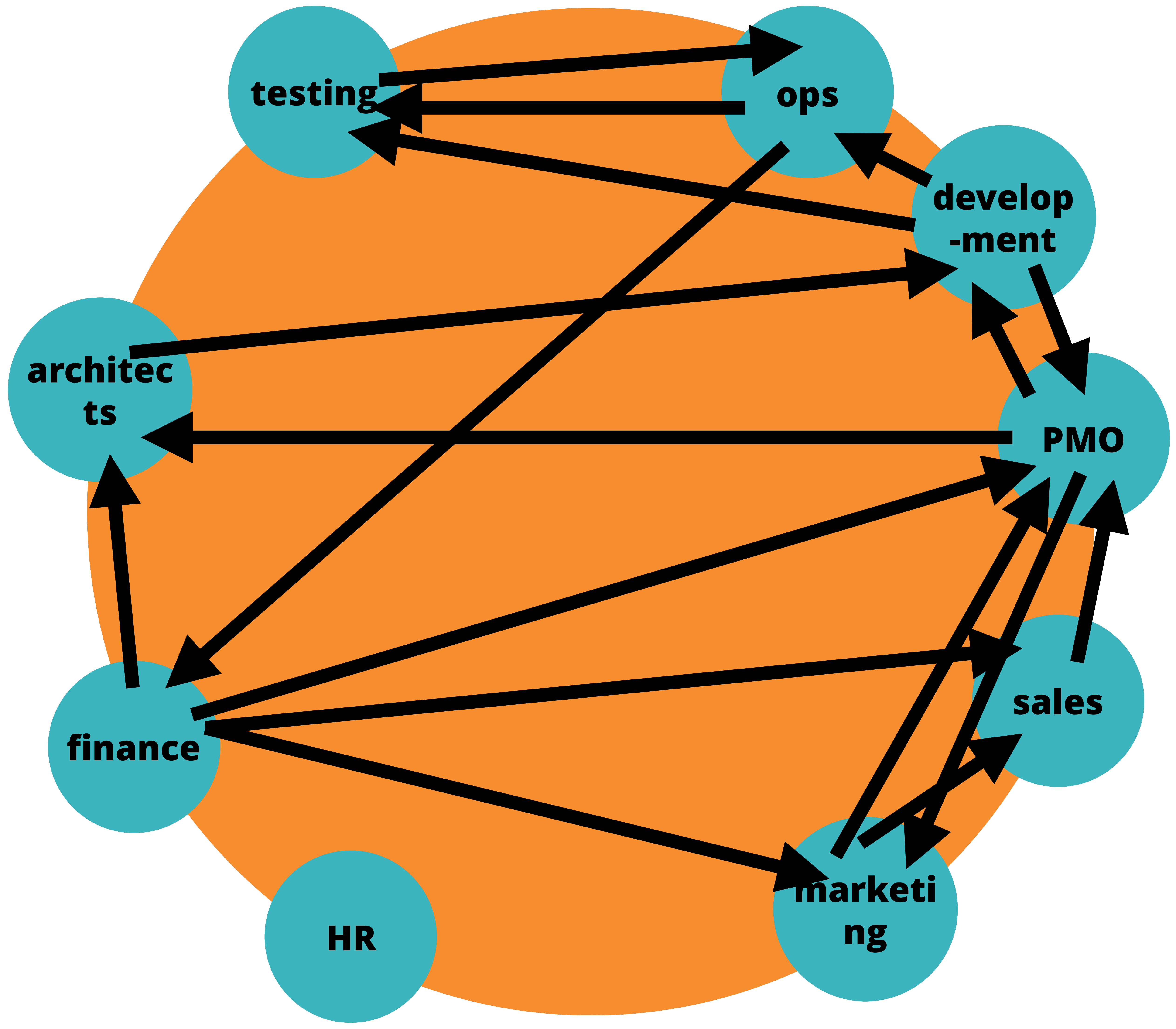
most organisations aren't setup to do this effectively

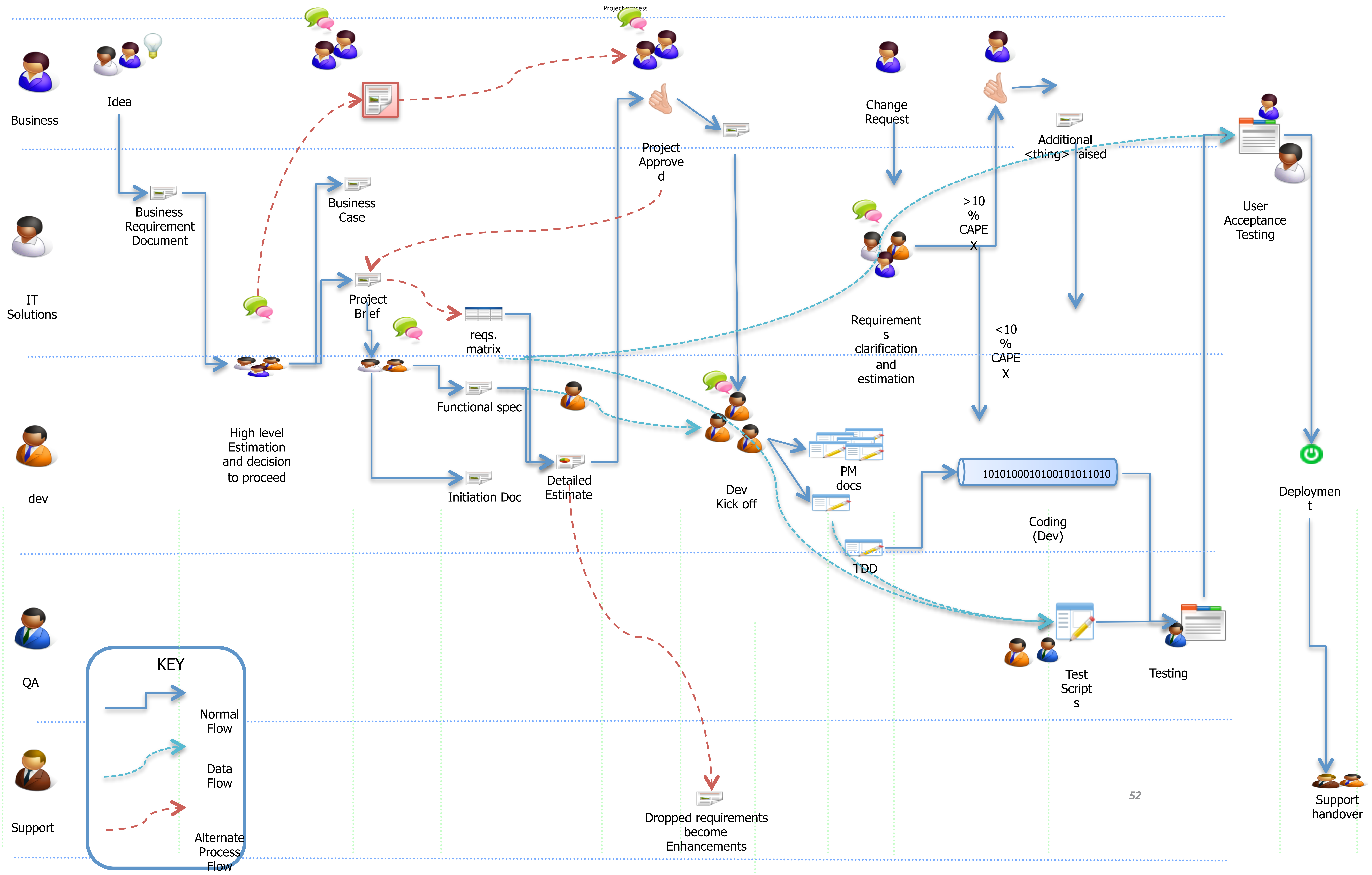
Part the Fourth

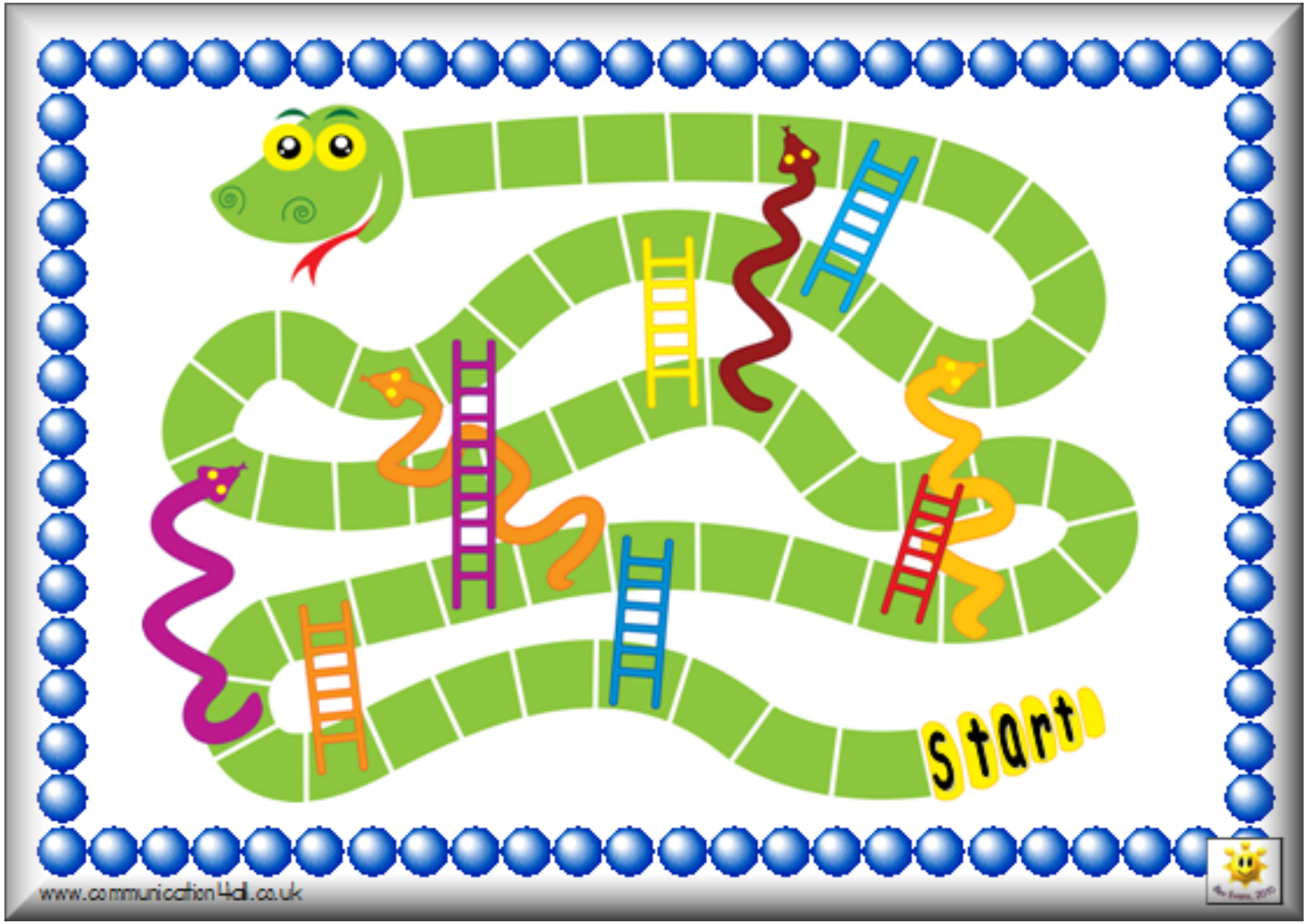
The Project Managers Tale

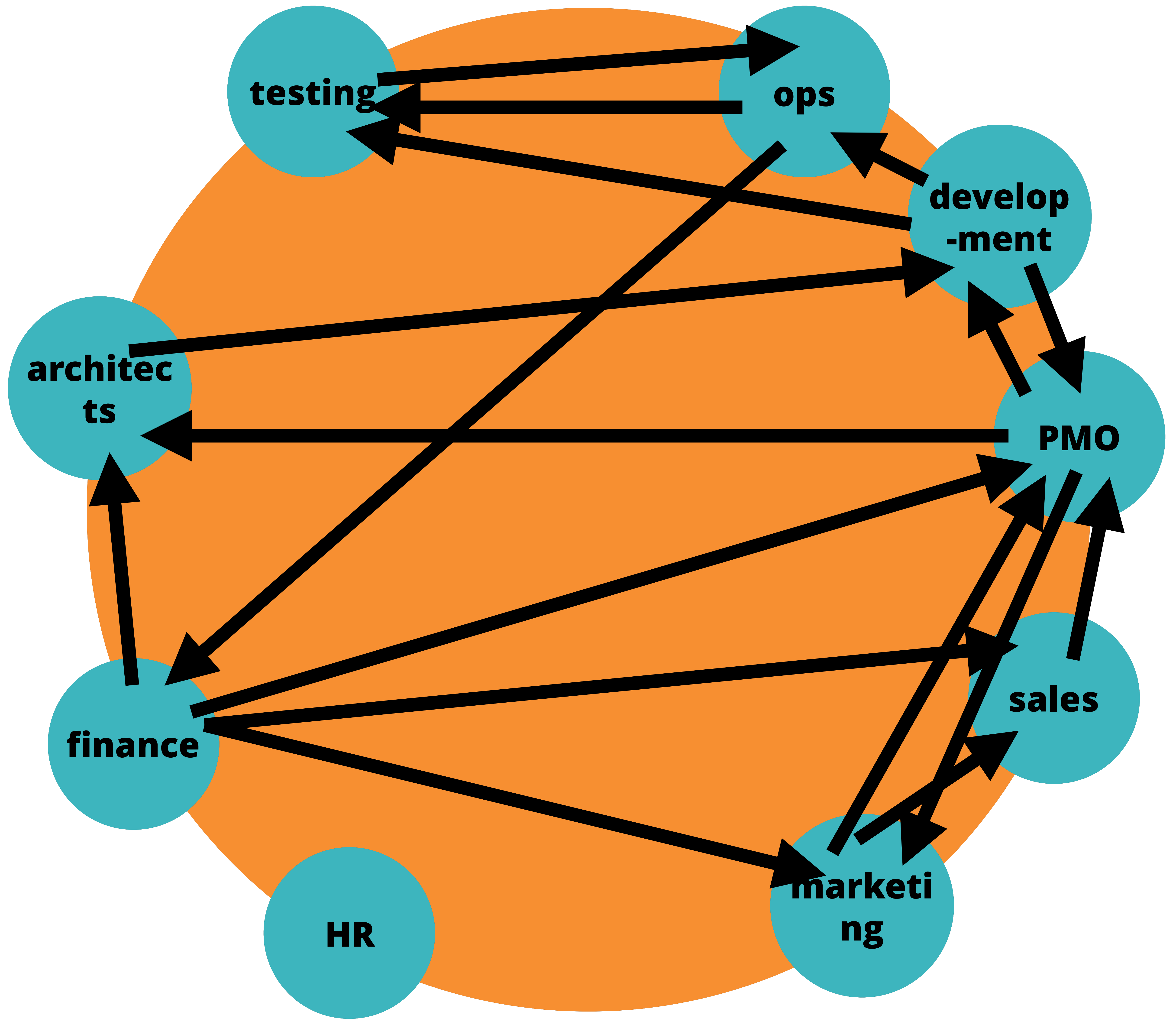
“The Bellman looked uffish, and wrinkled his brow.
"If only you'd spoken before!
It's excessively awkward to mention it now,
With the Snark, so to speak, at the door!”

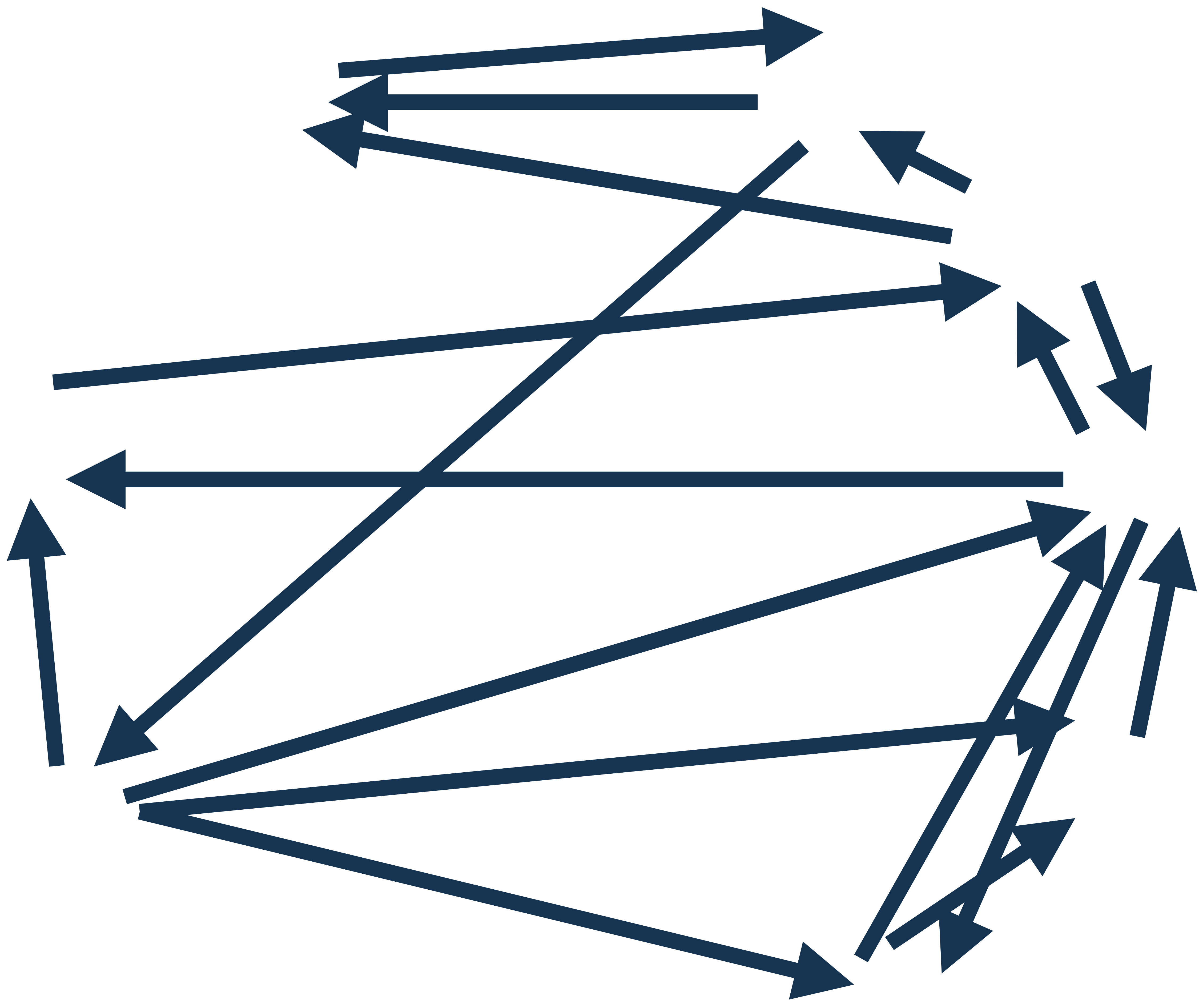

testing
ops
architec
ts
develop
-ment
PMO
sales
finance
marketi
ng
HR


testing
ops
develop
-ment
architec
ts
PMO
finance
sales
marketi
ng
HR

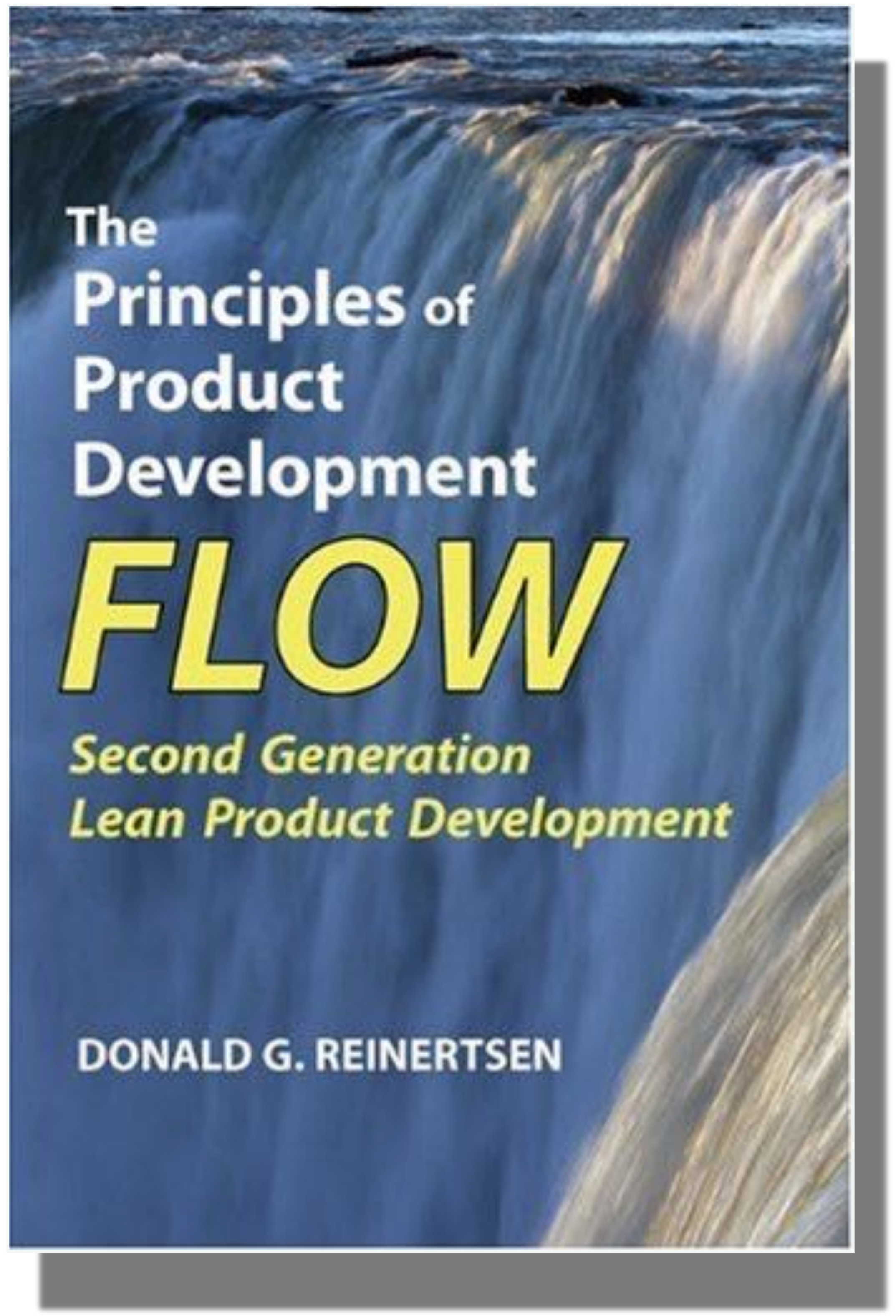

Project process
Business
Idea
IT Solutions
dev
QA
Support
Business Requirement Document
High level Estimation and decision to proceed
Business Case
Project Brief
reqs. matrix
Functional spec
Initiation Doc
Detailed Estimate
Project Approve d
Dev Kick off
PM docs
TDD
Change Request
Requirement s clarification and estimation
>10 % CAPE X
<10 % CAPE X
Additional <thing> raised
Coding (Dev)
1010100010100101011010
Test Script s
Testing
User Acceptance Testing
Deploymen t
Support handover
Dropped requirements become Enhancements
KEY
Normal Flow
Data Flow
Alternate Process Flow
52

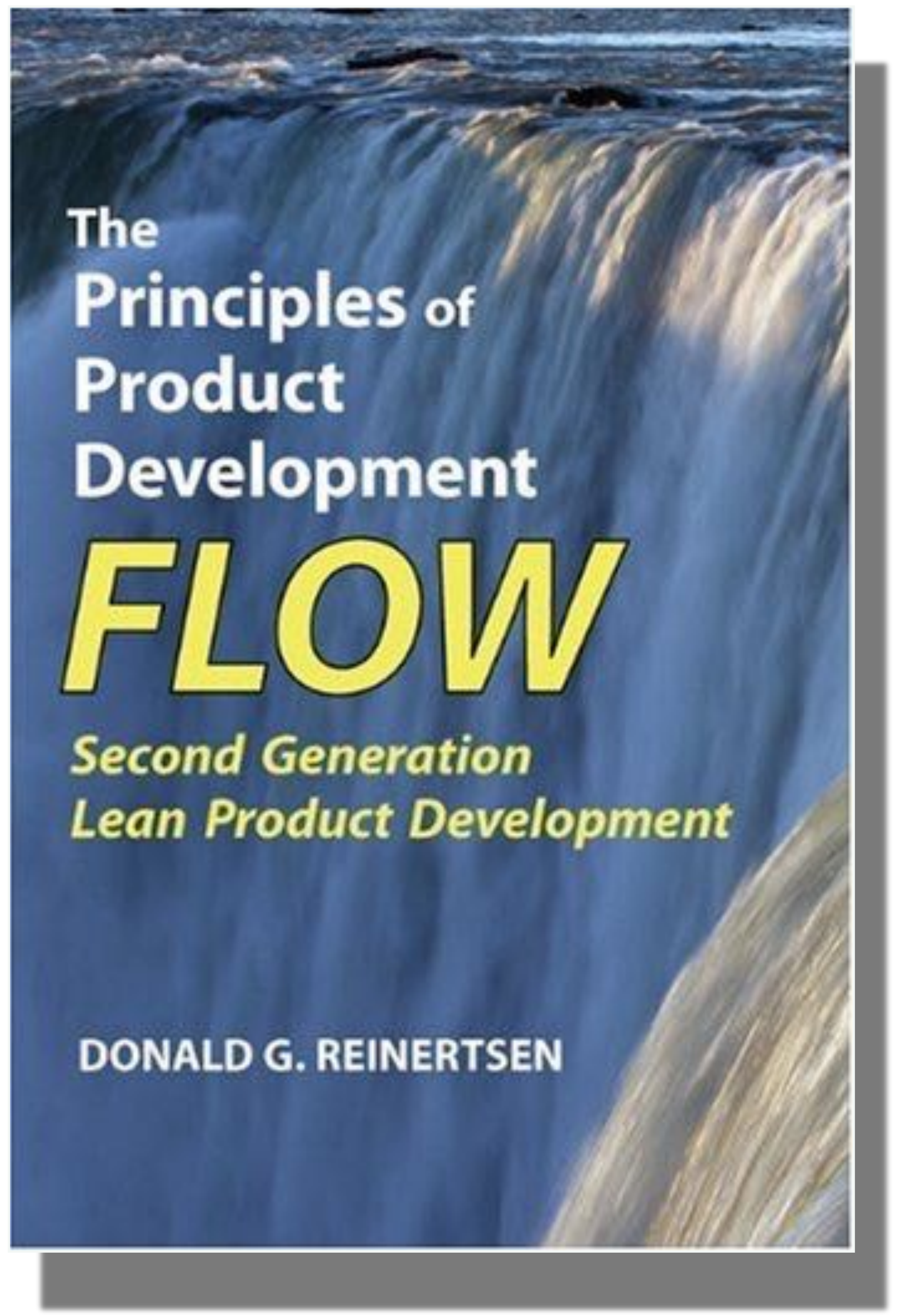

Start
www.communication4all.co.uk


testing
ops
develop
-ment
architec
ts
PMO
finance
sales
marketi
ng
HR



each of these chords represents a queue


The Principles of Product Development
FLOW
Second Generation Lean Product Development
DONALD G. REINERTSEN


The Principles of Product Development
FLOW
Second Generation Lean Product Development
DONALD G. REINERTSEN

The Effect of Queues

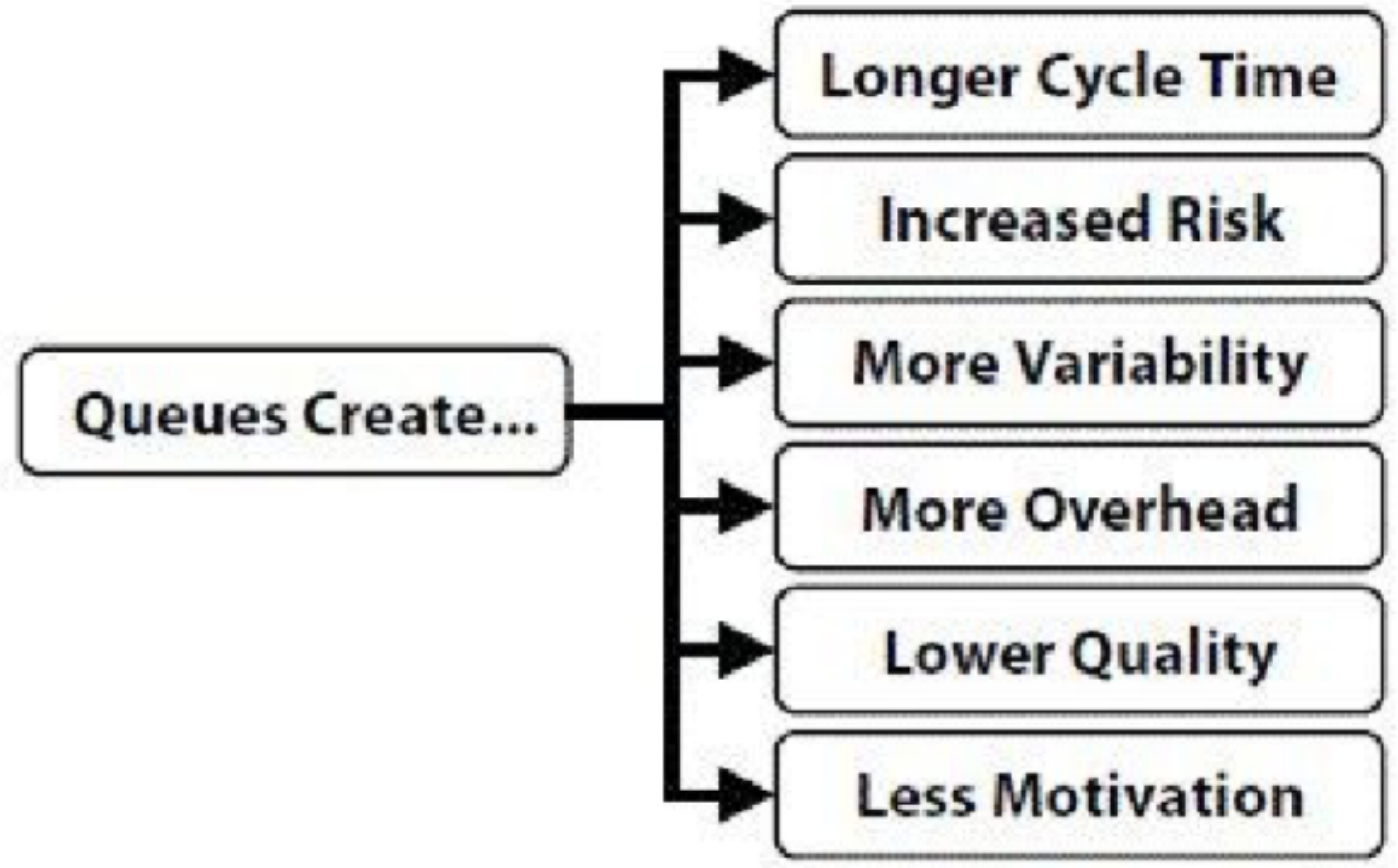


Figure 3-1 Queues are the underlying cause of many economic problems in product development. Yet, they remain unmeasured at 98 percent of product developers.

“There is nothing so **useless** as doing **efficiently** that which should **not be done** at all”

Peter Drucker

The first order factor influencing most organisations building software

is the number of queues in the development process

and the size of the batches of work flowing through them

Part the Fifth

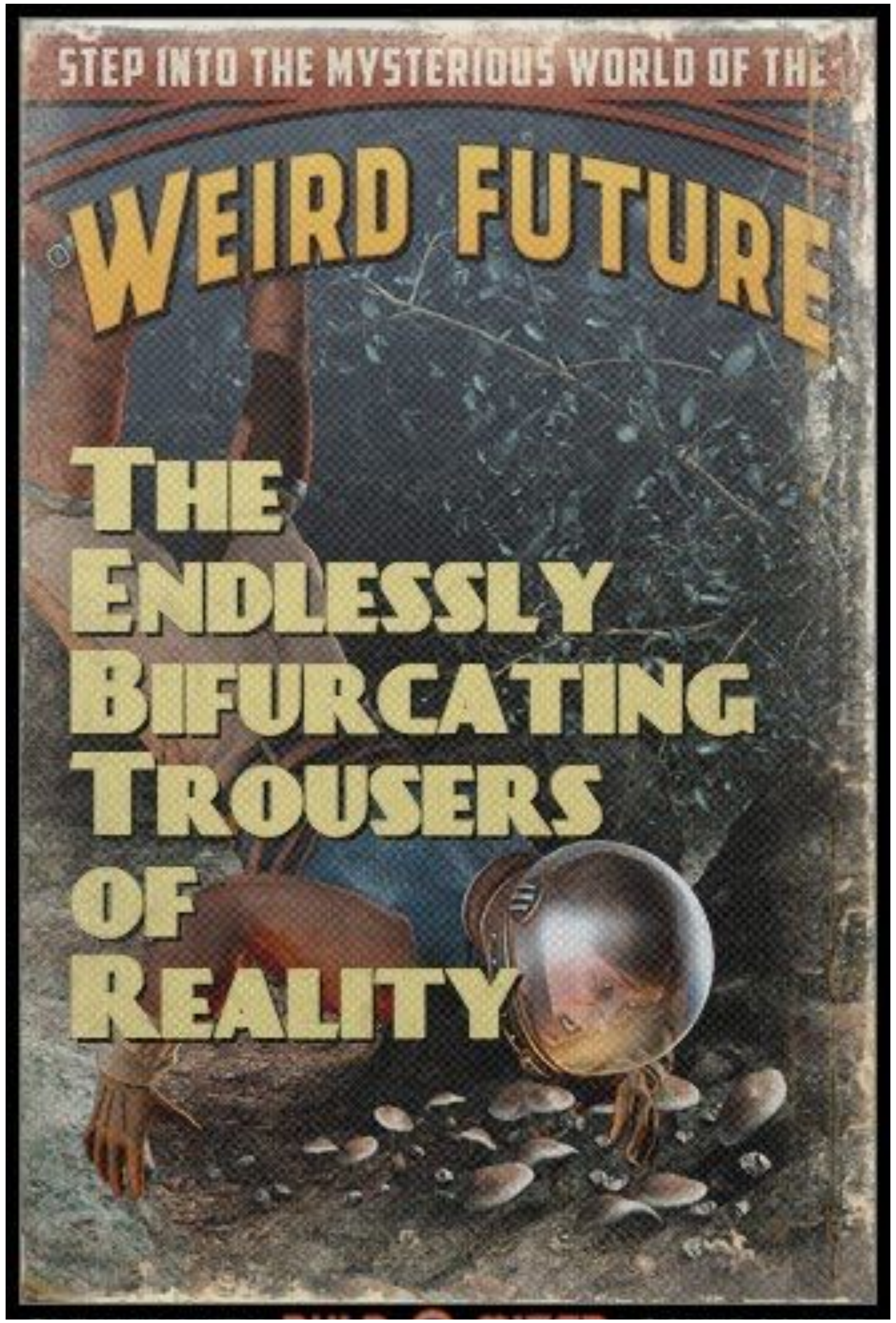
The Architect's dream

"They sought it with thimbles, they sought it with care;
They pursued it with forks and hope;
They threatened its life with a railway-share;
They charmed it with smiles and soap."

this is the problem

"It is perfectly true, as philosophers say, that life must be understood backwards. But they forget the other proposition, that it must be lived forwards."

Søren Kierkegaard

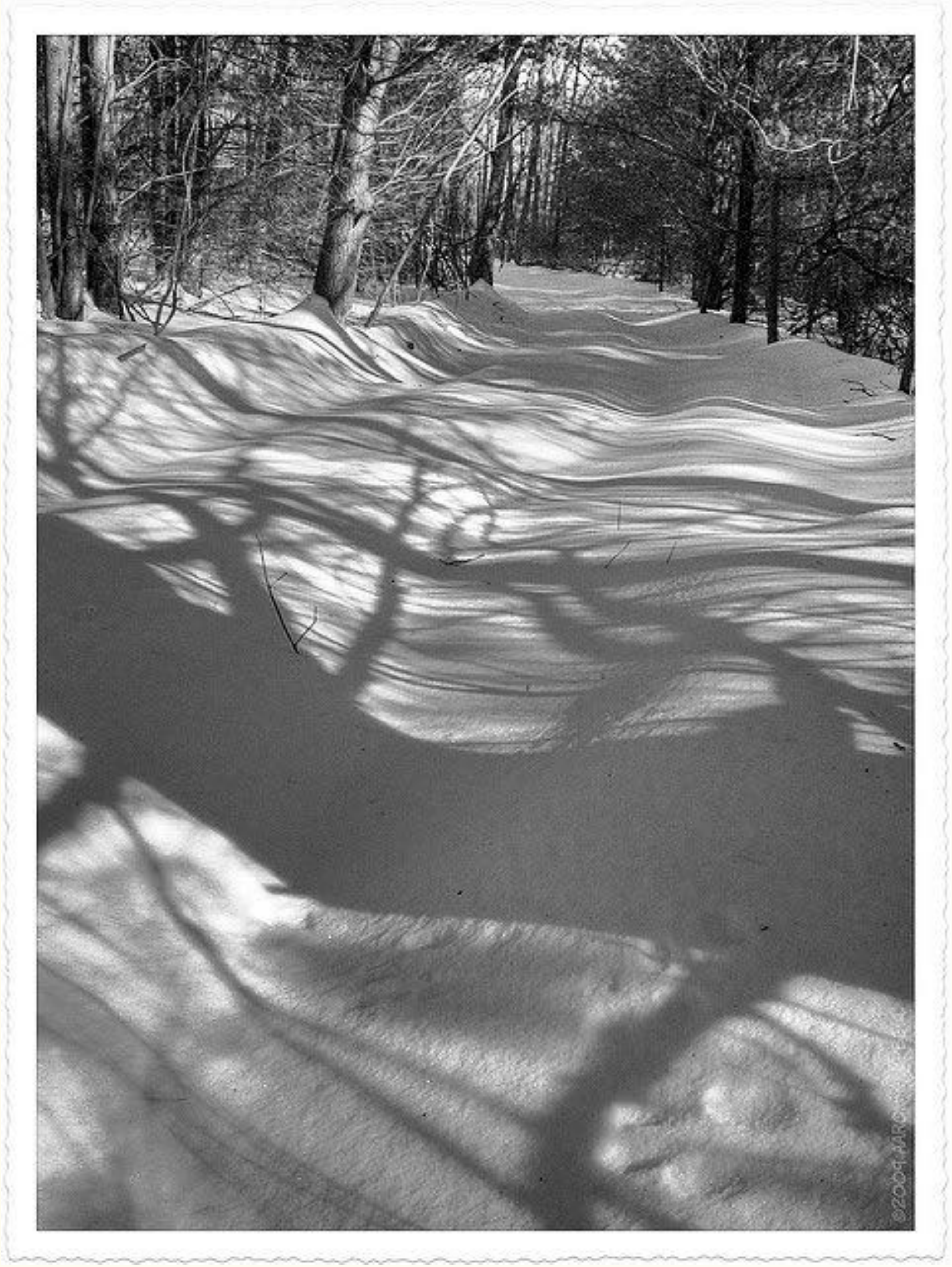

STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE
THE ENDLESSLY BIFURCATING TROUSERS OF REALITY

History

The lawful good product owners of the publishing house had long lived in awe and fear of their publishing systems.

In awe, for they had made a tremendous amount of Gold, but in fear of the time taken to change them, their slowness and their fragility.

A messenger was sent to fetch help from a distant land famed for it’s mighty wizards. You have taken up the challenge…



1.

You must save the product owners by rebuilding their content delivery system. You start off the project. In the course of discussions you discover that your goals are three fold:

1. improve availability
2. improve performance
3. reduce the cost of delay

An Enterprise Architect approaches and addresses you.

You may use:

Summon Walking Skeleton	turn to 4
Analysis Paralysis	turn to 3

If you have none of these you will have to draw your sword and fight (turn to **178**)

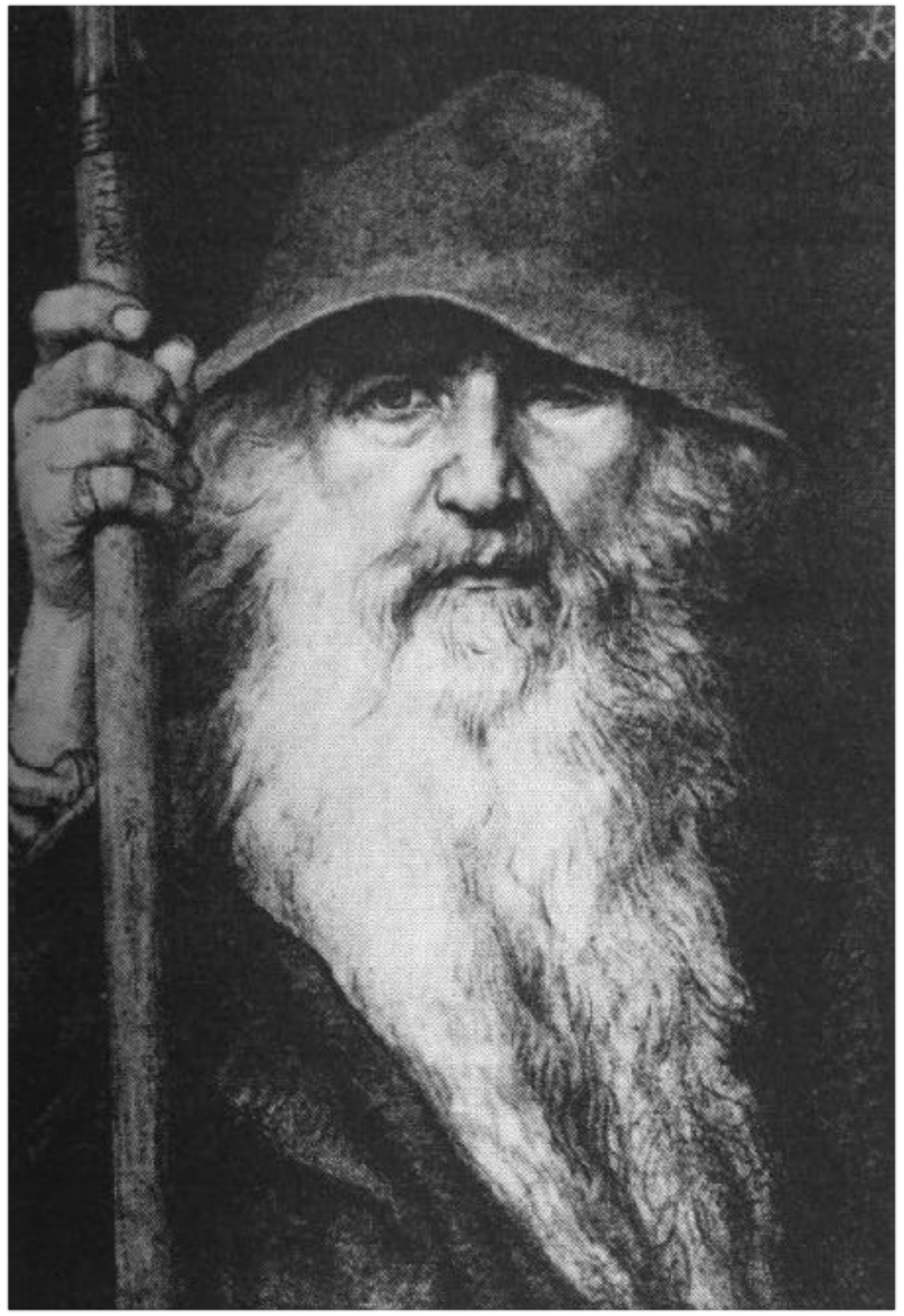
3.

You cast Analysis Paralysis at the Enterprise Architect.

“Foolish young adventurer” says the architect, “we follow the evolutionary school of architecture and we shall have none of the lawful-evil ways of waterfall”.

The last thing you see before everything goes dark is the architect incanting in a strange voice.

You have died. Turn to page 1.



1.

You must save the product owners by rebuilding their website. You start off the project. In the course of discussions you discover that your goals are three fold:

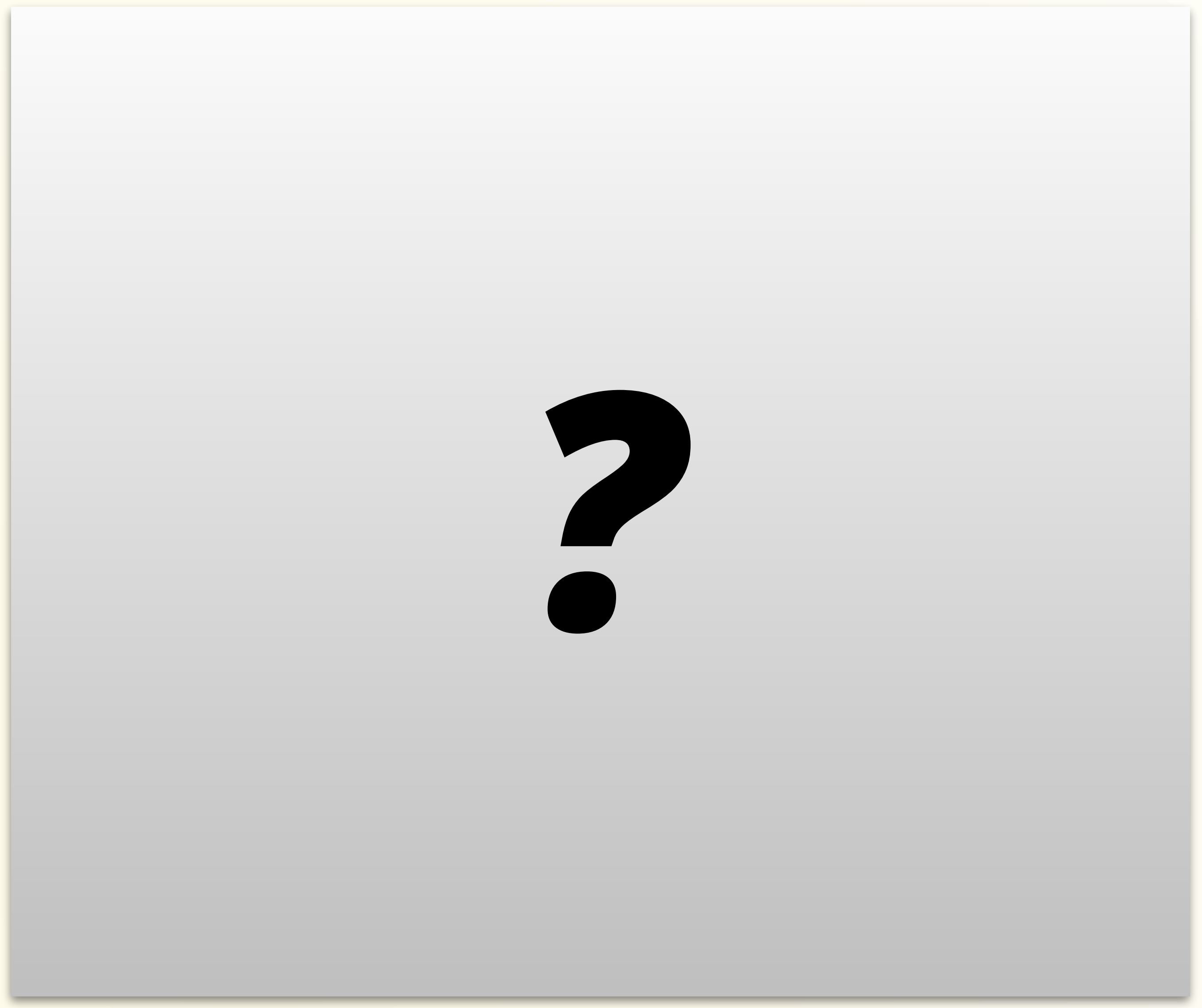
1. improve availability
2. improve performance
3. reduce the cost of delay

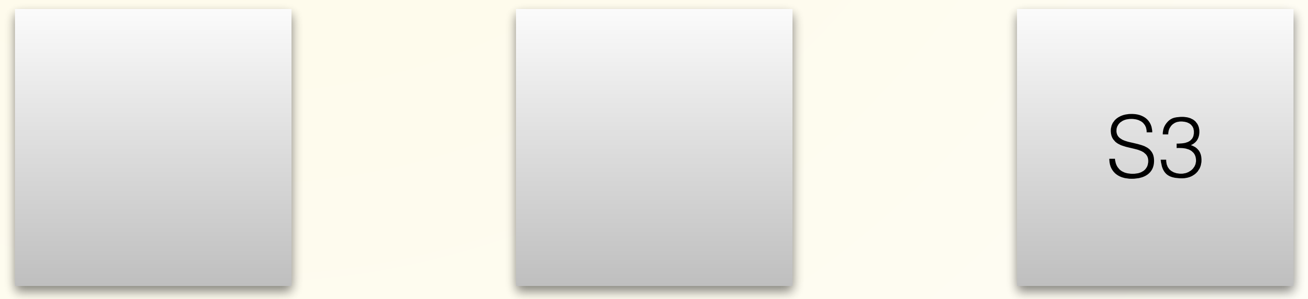
An Enterprise Architect approaches and addresses you.

You may use:

Summon Walking Skeleton	<u>turn to 4</u>
Analysis Paralysis	<u>turn to 3</u>

If you have none of these you will have to draw your sword and fight (turn to **178**)





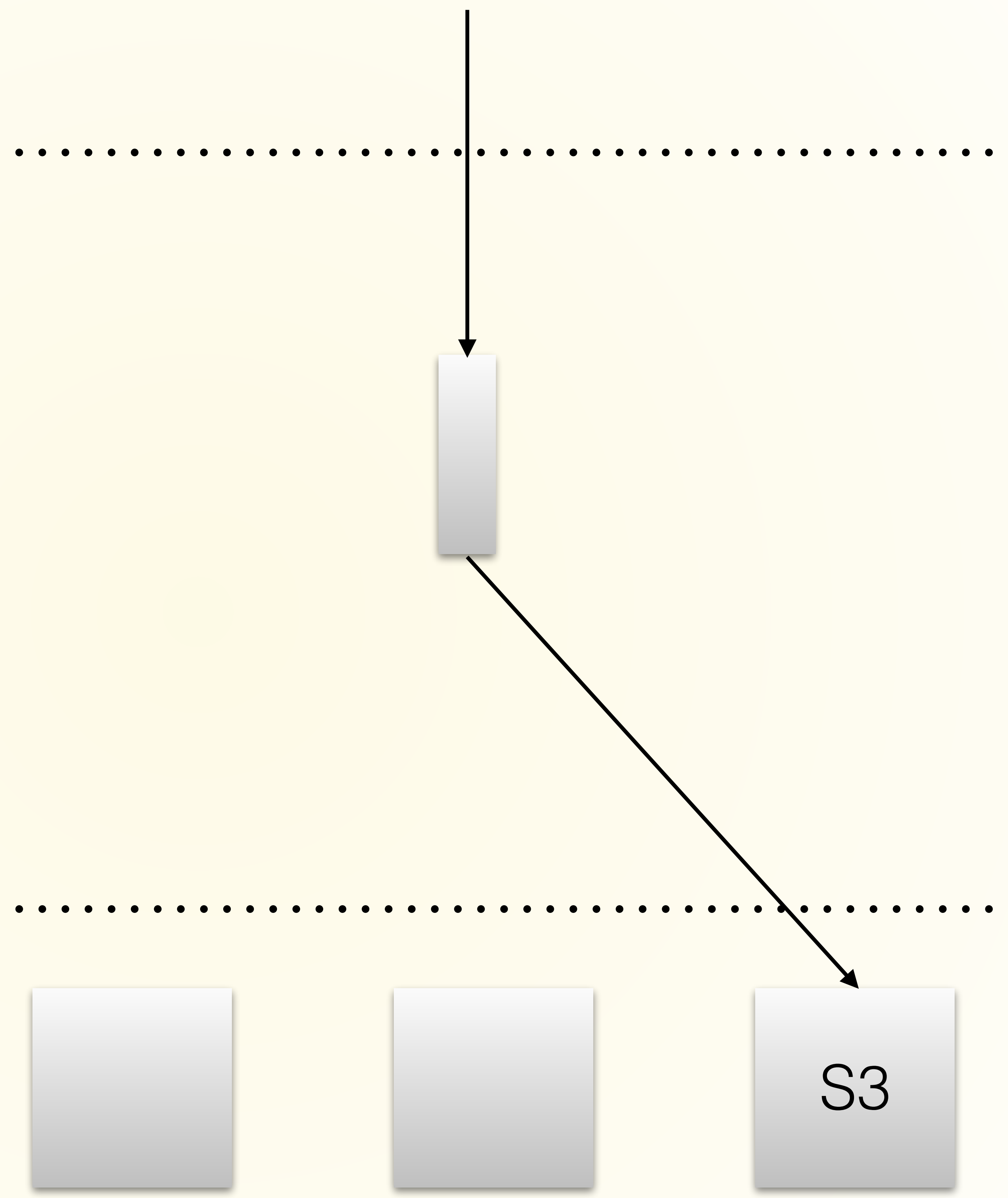
4.

Your walking skeleton coalesces in a cloud of noxious gasses and solidifies as a java dropwizard application.

You reach into your backpack and deploy the content store. Your walking skeleton reaches out it's skeletal arms and grabs armfuls of raw xml.

Would you like to:

Transform the xml inside the skeleton	turn to 6
Use a magic box	turn to 5



5.

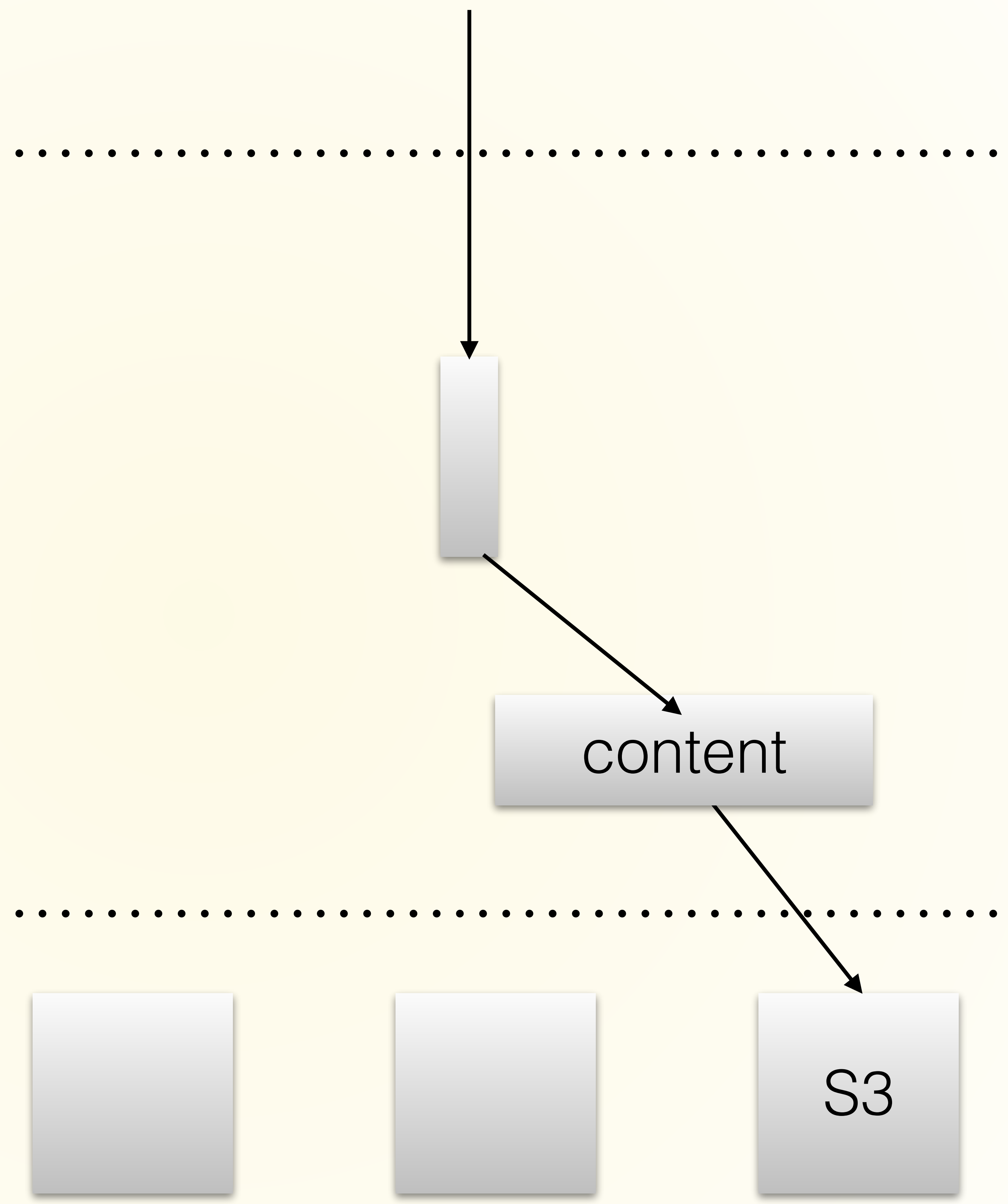
You throw the magic box in between the walking skeleton and the content store.

A villager approaches and exclaims: “this beautiful content I see in front of me seems to take an awful long time to get here”

You must somehow make the content arrive faster.

If you have a http cache in your inventory, you may use it now.

Cache in between S3 and content	turn to 10
Cache in between skeleton and content	turn to 33



6.

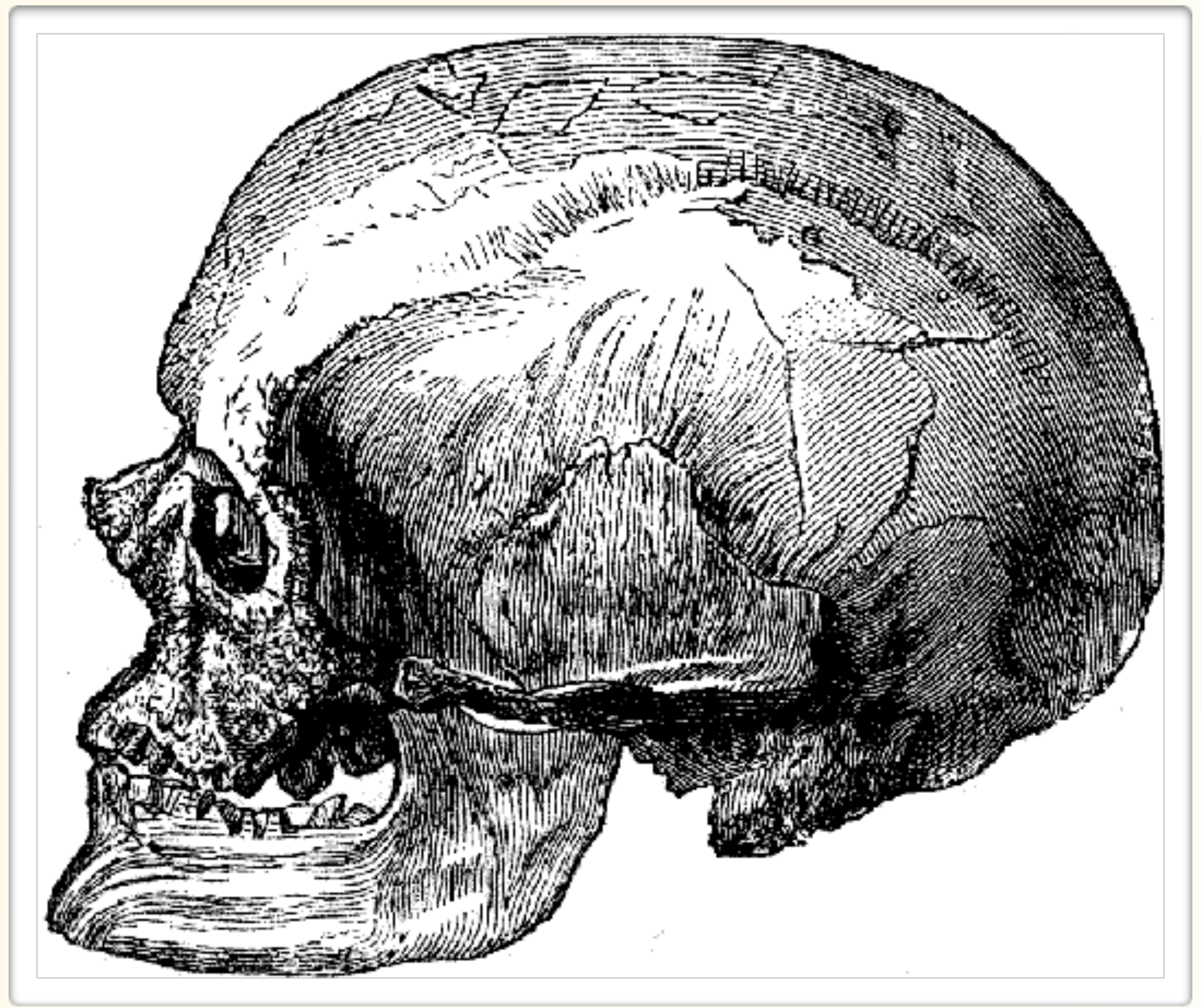
The skeleton gurgles, grunts and then doubles in size.

A villager approaches and exclaims: “this beautiful content I see in front of me seems to take an awful long time to get here”

You try to add a cache into the skeleton’s bony skull. First you cast sticky sessions. With a splash it rebounds, soaking you in the stench of the unscalable.

Desperately, you try terracotta and then the oracle of coherence. Nothing seems to work. The murky substances overwhelm you.

You have died. turn to page 1.



10.

The cache causes the content load times to drop from 300ms to 150ms.

The villager says “this wonderful content is now arriving more swiftly than even the knight-messengers of the Empress”.

The villagers are happy but all too soon, all is not well for the content has a long tail. You must work out how to refresh the content when it changes.

You can either:

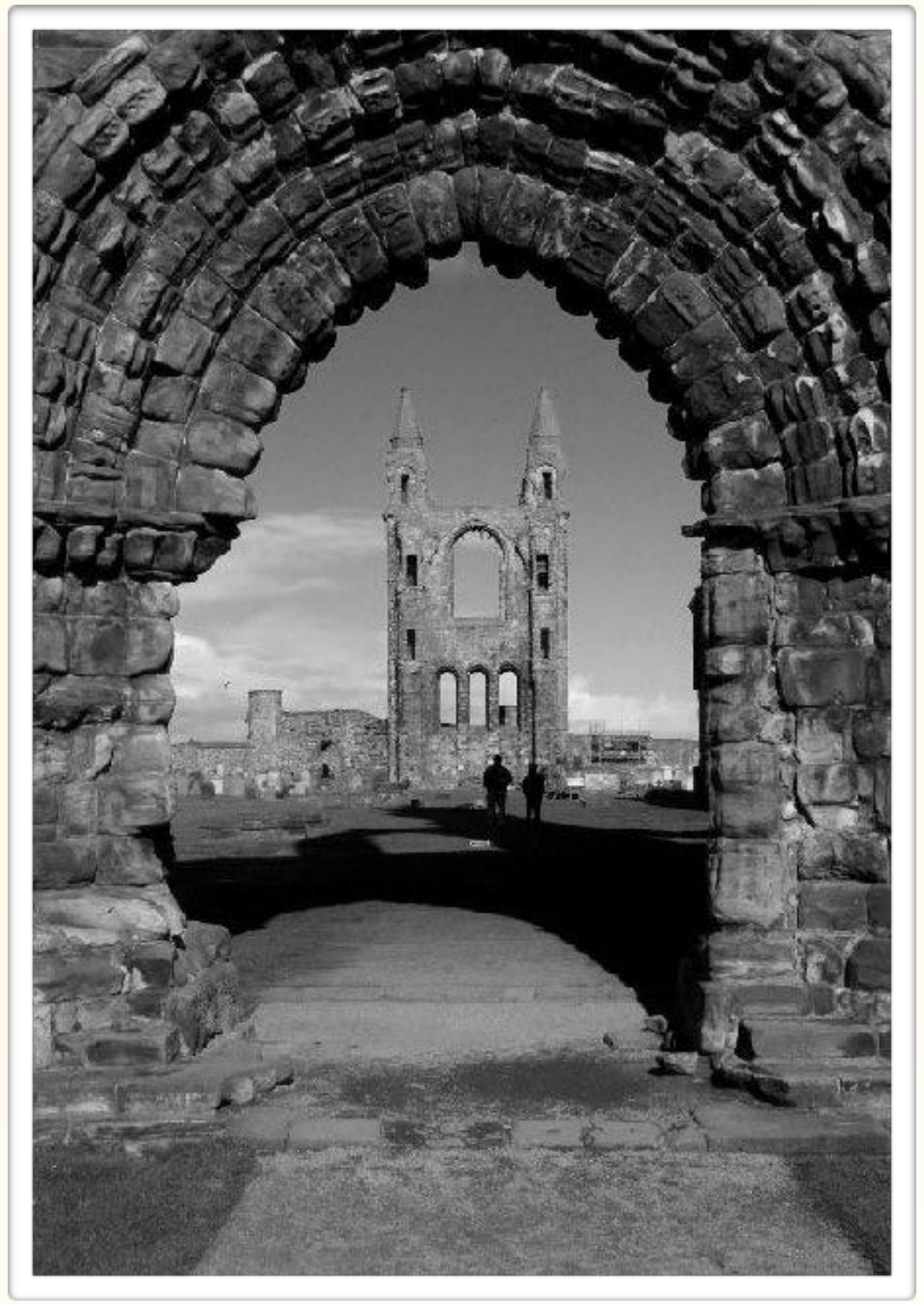
Refresh the content when it appears from the ether	turn to 150
Trust that it will be fast enough on first view	turn to 22

22.

The tail is just too long. When villagers or merchants try to use the content it is just too slow to arrive.

The amount of Gold diminishes and over the years the village fades into a forgotten hamlet, then to a legend and a myth.

You have died, turn to page 1.



150.

Content trickles into the store. You keep up by listening for the new content and casting “wget” on the cache to keep it refreshed.

New types of content appears - content the villagers have never seen before. Content the walking skeleton is unable to combat.

Fortunately, through Continuous Delivery you are able to keep up with the changed content but the cache doesn’t. The cache becomes stale.

How will you keep your delivery continuous?

cast cache shards	turn to **255**

If you are unable to shard the cache turn to page **48**

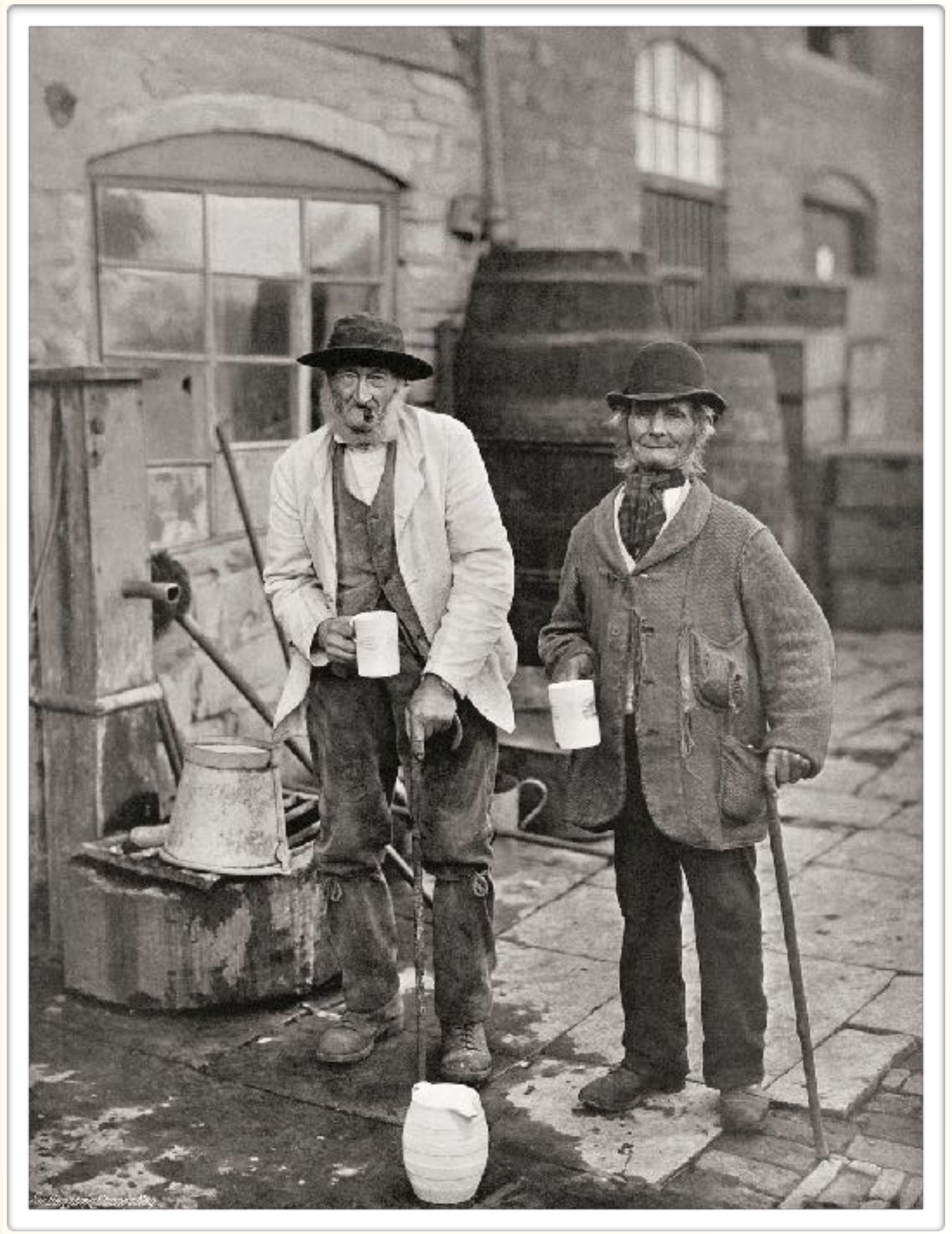
33.

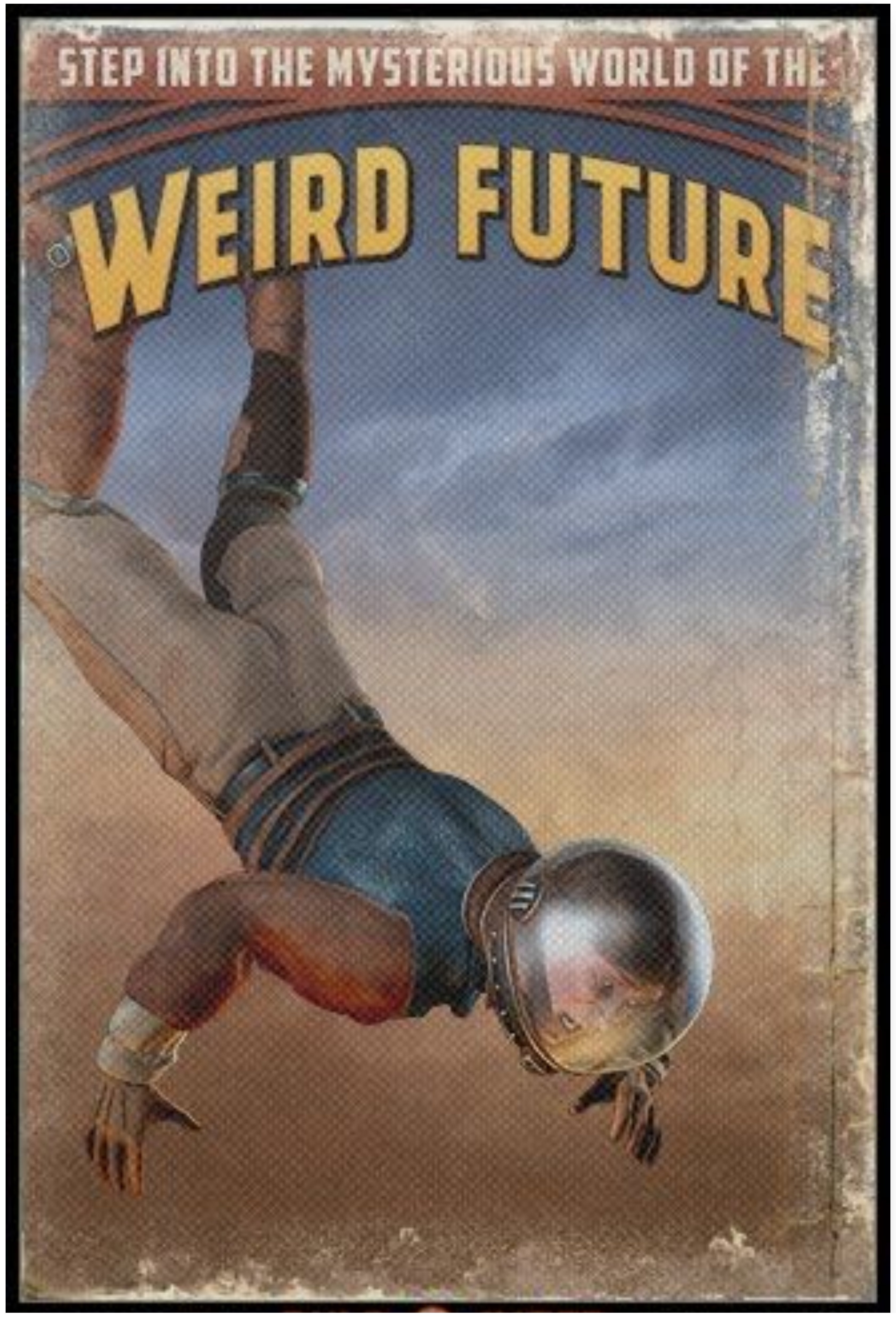
The HTTP cache has an instant effect. Latency drops from 300ms to 10ms.

Changes to the content mount up. Every time one of the lawful-good researches publishes something, the cache must be refreshed. Every time the skeleton changes it’s appearance, the cache must be refreshed.

The villagers need you to do something. Will you:

Suffer the long tail	turn to 22
Refresh the cache on API and content changes	turn to 150




STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE

Evolutionary Architecture is a fundamental concern

and it’s really hard.

It requires us to be comfortable with uncertainty

Part the Sixth

The System Administrators Fate

“They sought it with thimbles, they sought it with care;
They pursued it with forks and hope;
They threatened its life with a railway-share;
They charmed it with smiles and soap.“

microservices **should** *allow us to go as* ***"fast as possible"***

be cheap to replace

be deployable on demand

be resilient on imperfect networks

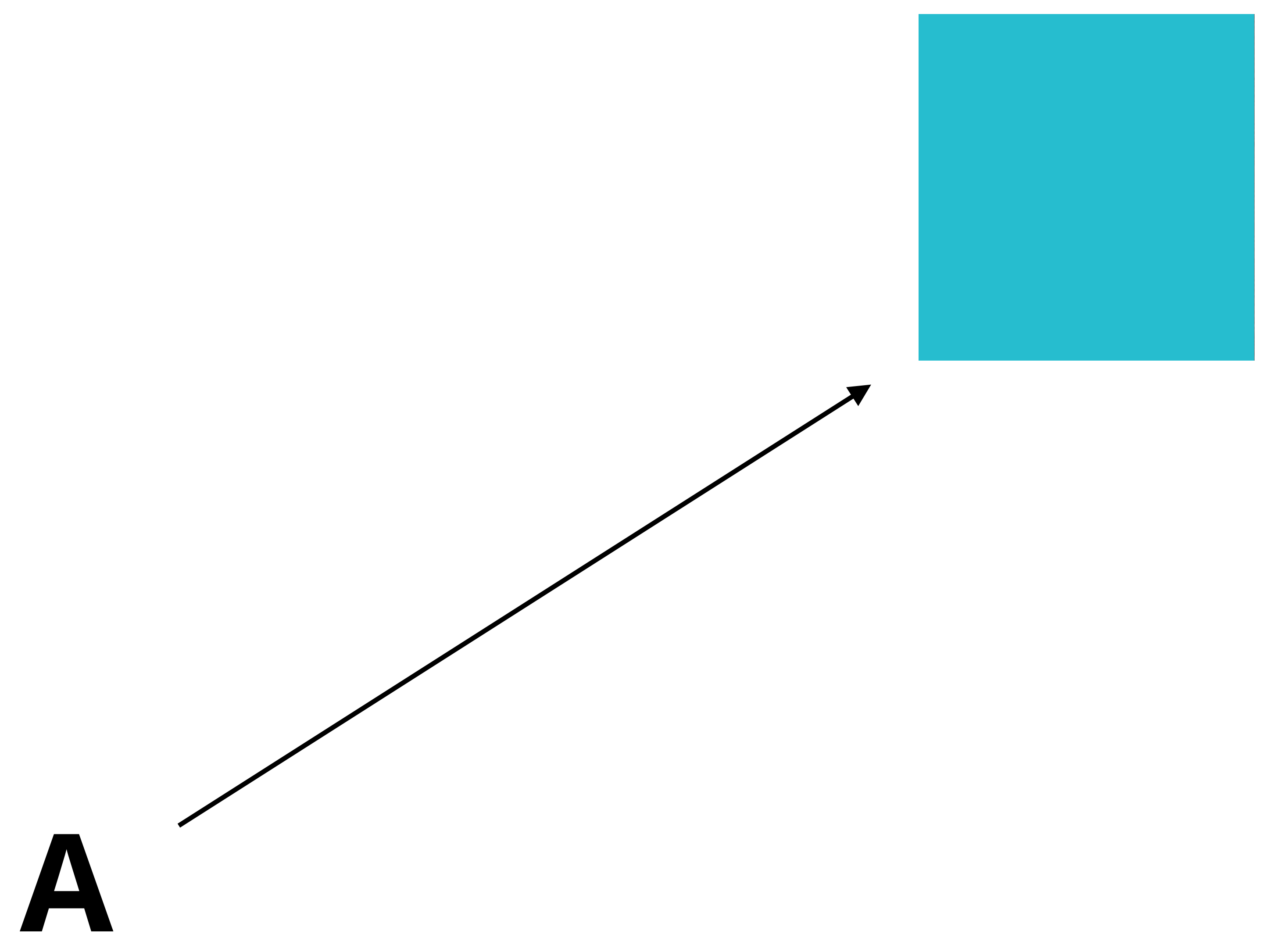
How **big** *are they?*

~~*How* **big** *are they?*~~

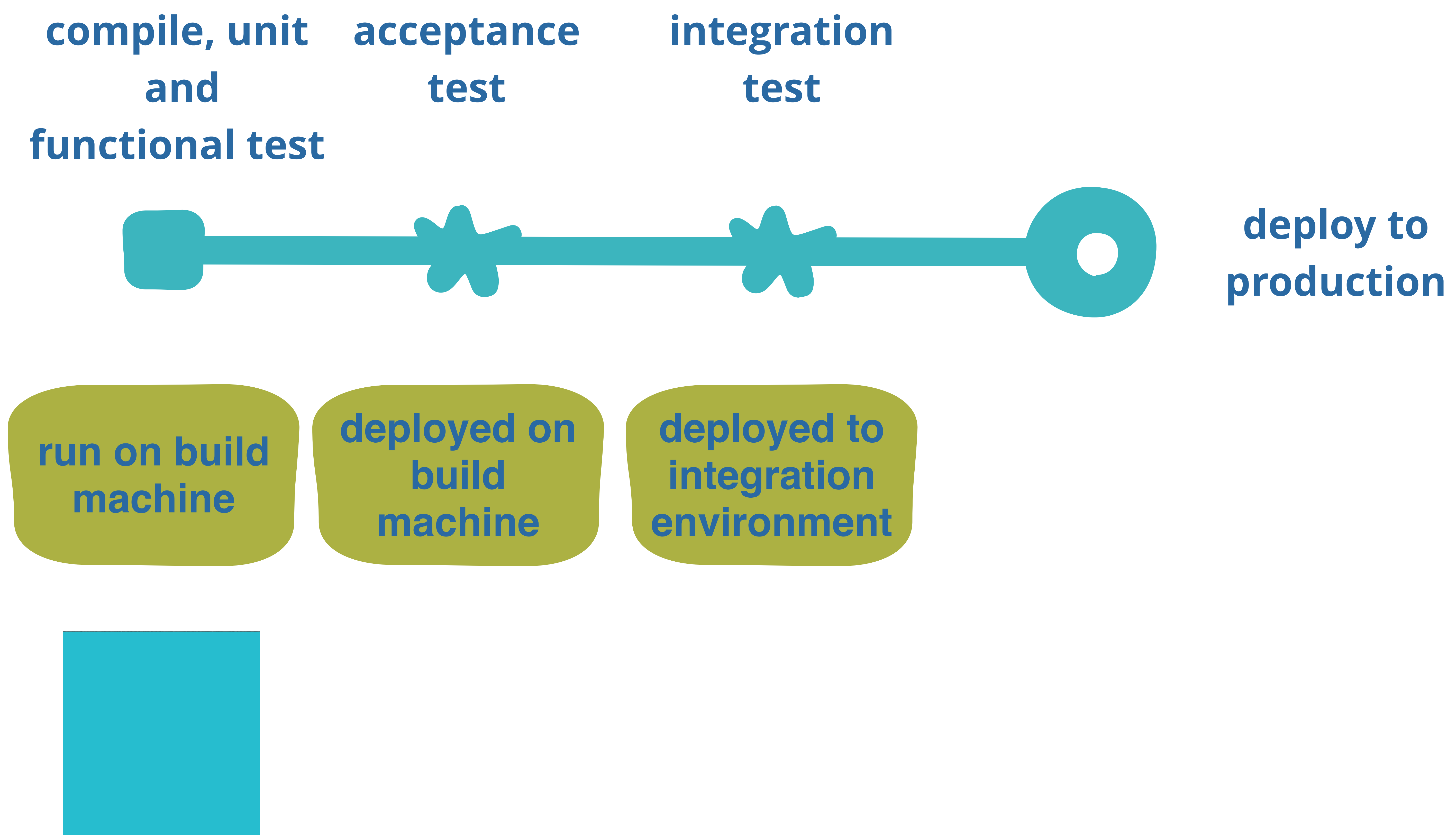
~~*How* **big** *are they?*~~

How **many** *can you support?*

Consider a single application - its a website, lets call it A

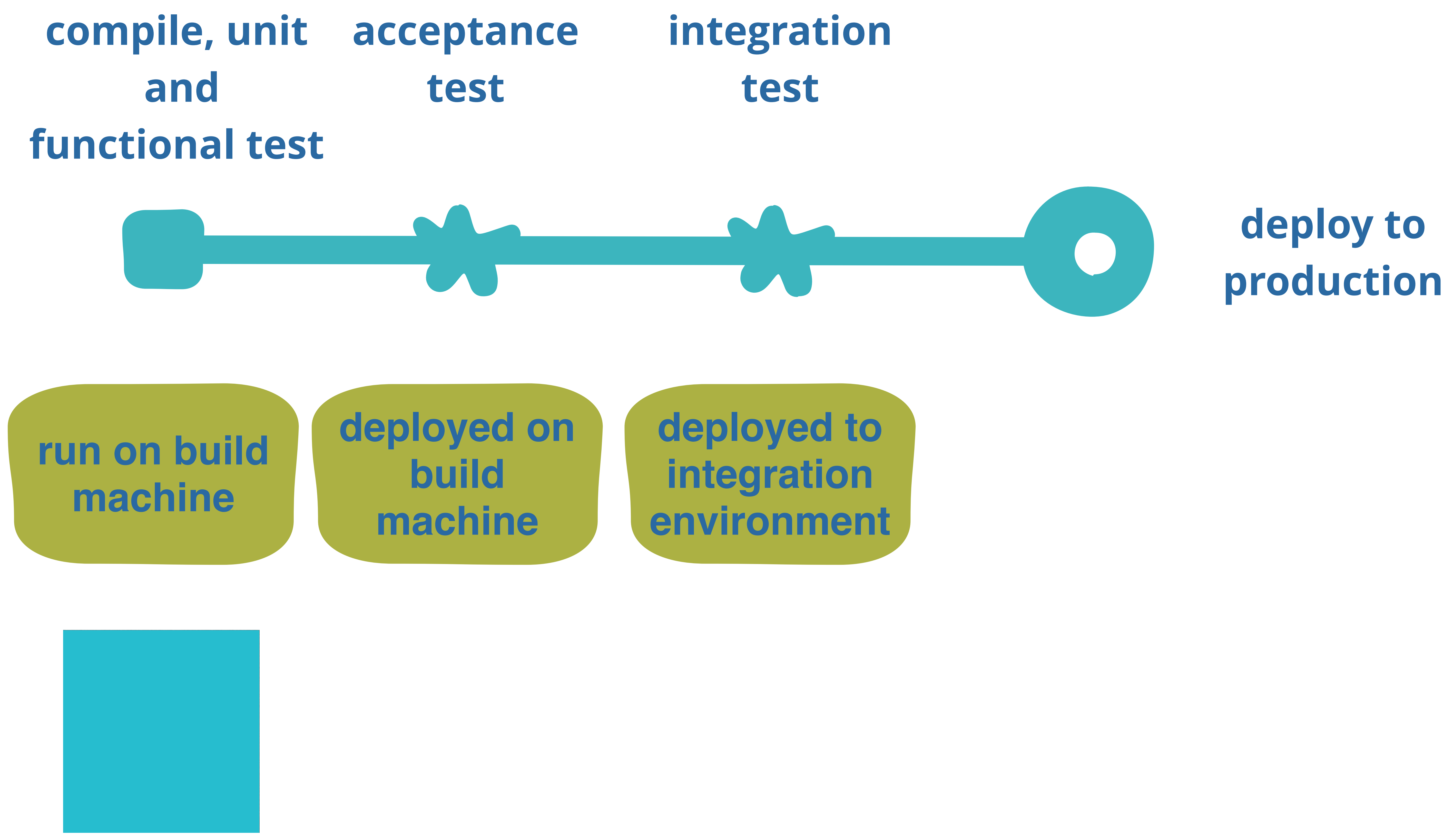


we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline


compile, unit and functional test
acceptance test
integration test
deploy to production
run on build machine
deployed on build machine
deployed to integration environment

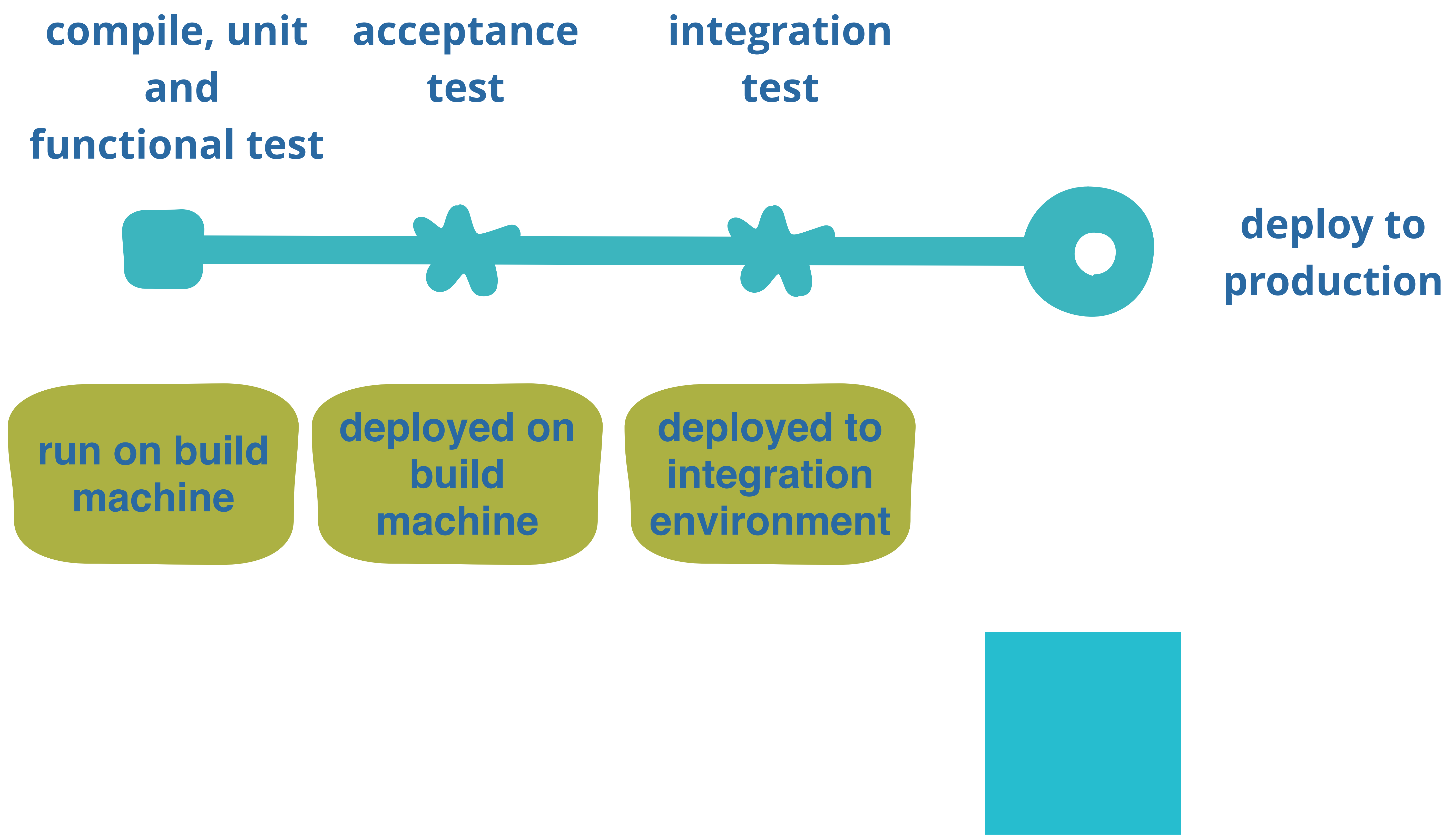
we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline

Tappety tap


compile, unit and functional test
acceptance test
integration test
deploy to production
run on build machine
deployed on build machine
deployed to integration environment

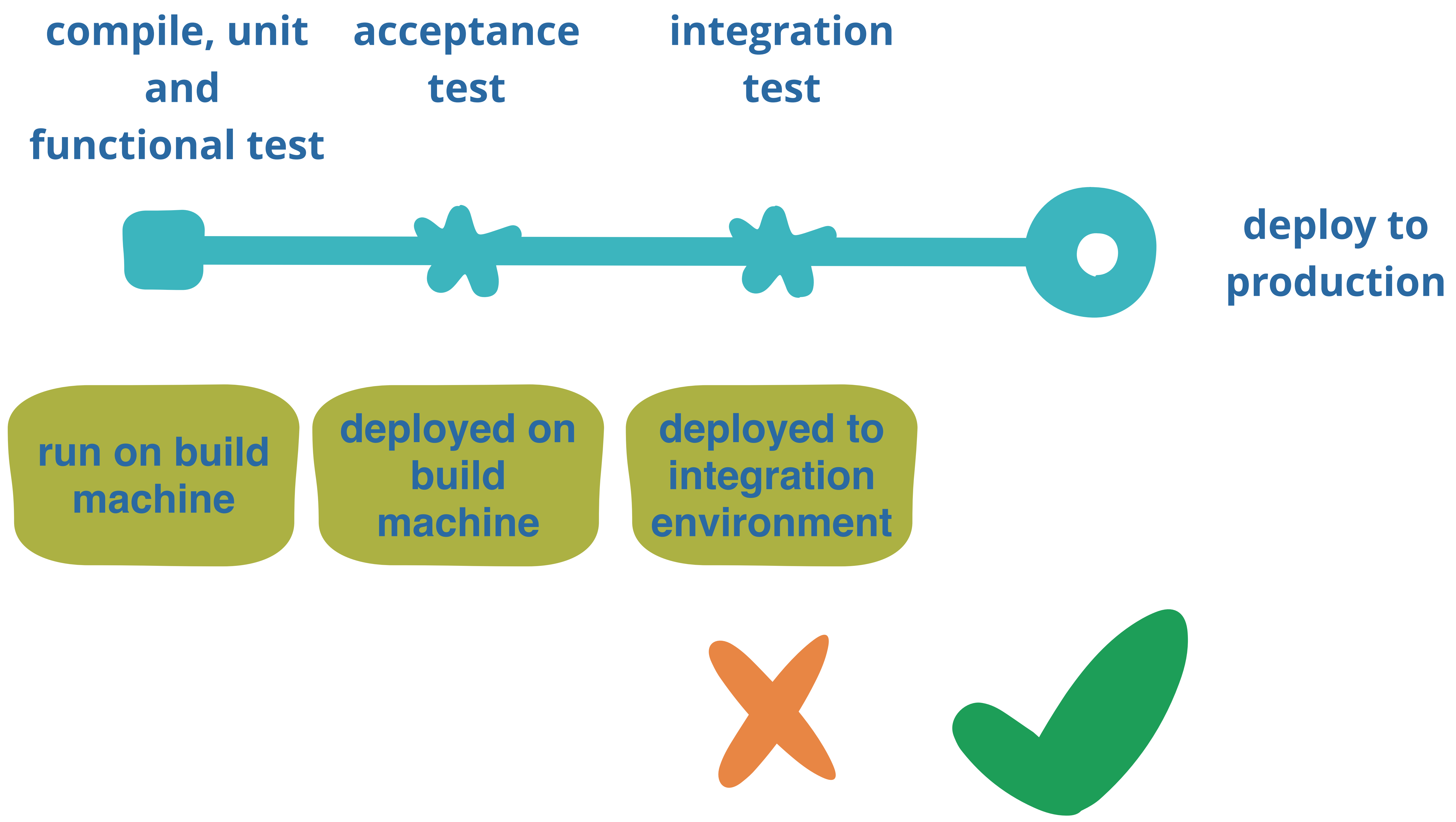
we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline

Tappety tap


compile, unit and functional test
acceptance test
integration test
deploy to production
run on build machine
deployed on build machine
deployed to integration environment

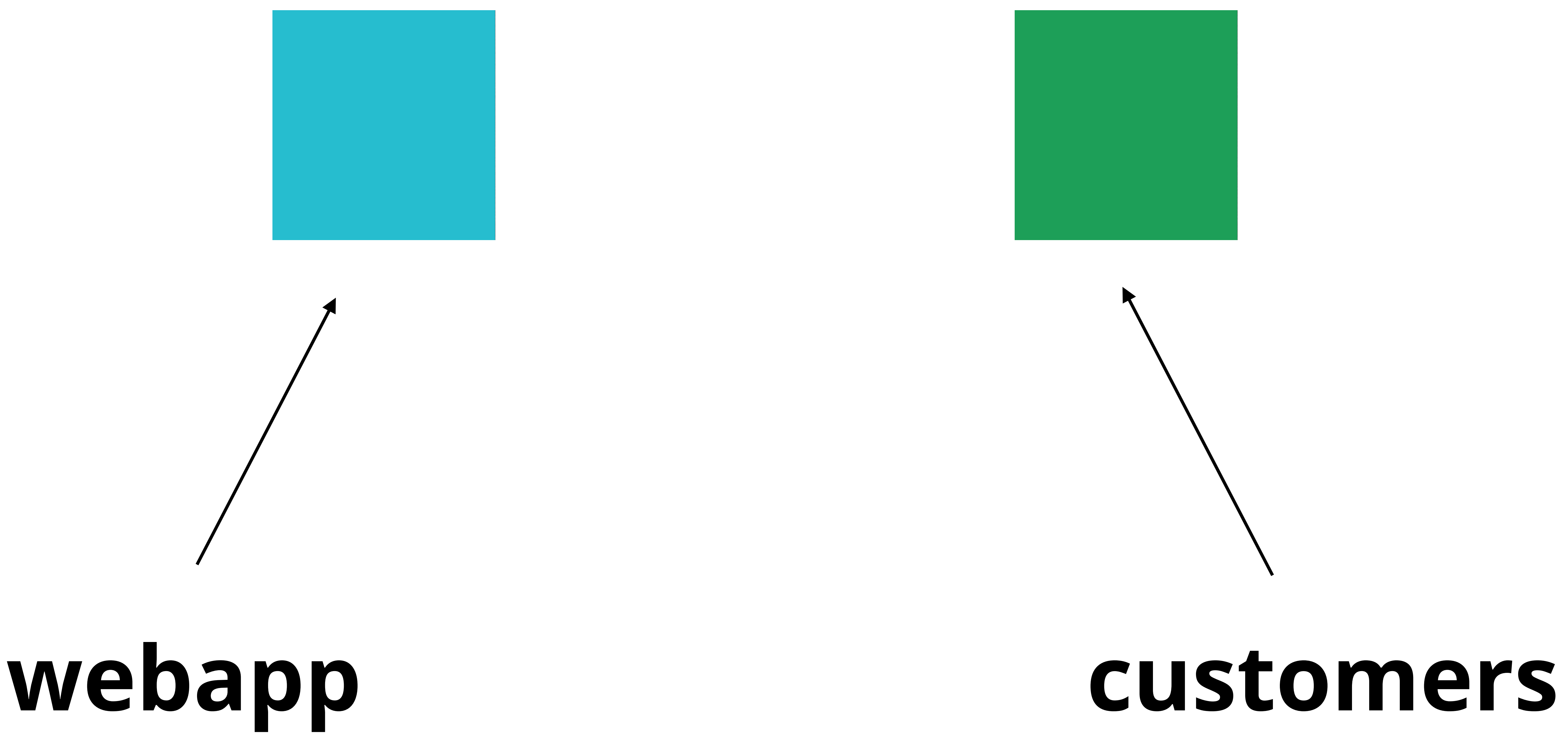
How many environments do we need?

How many environments do we need?

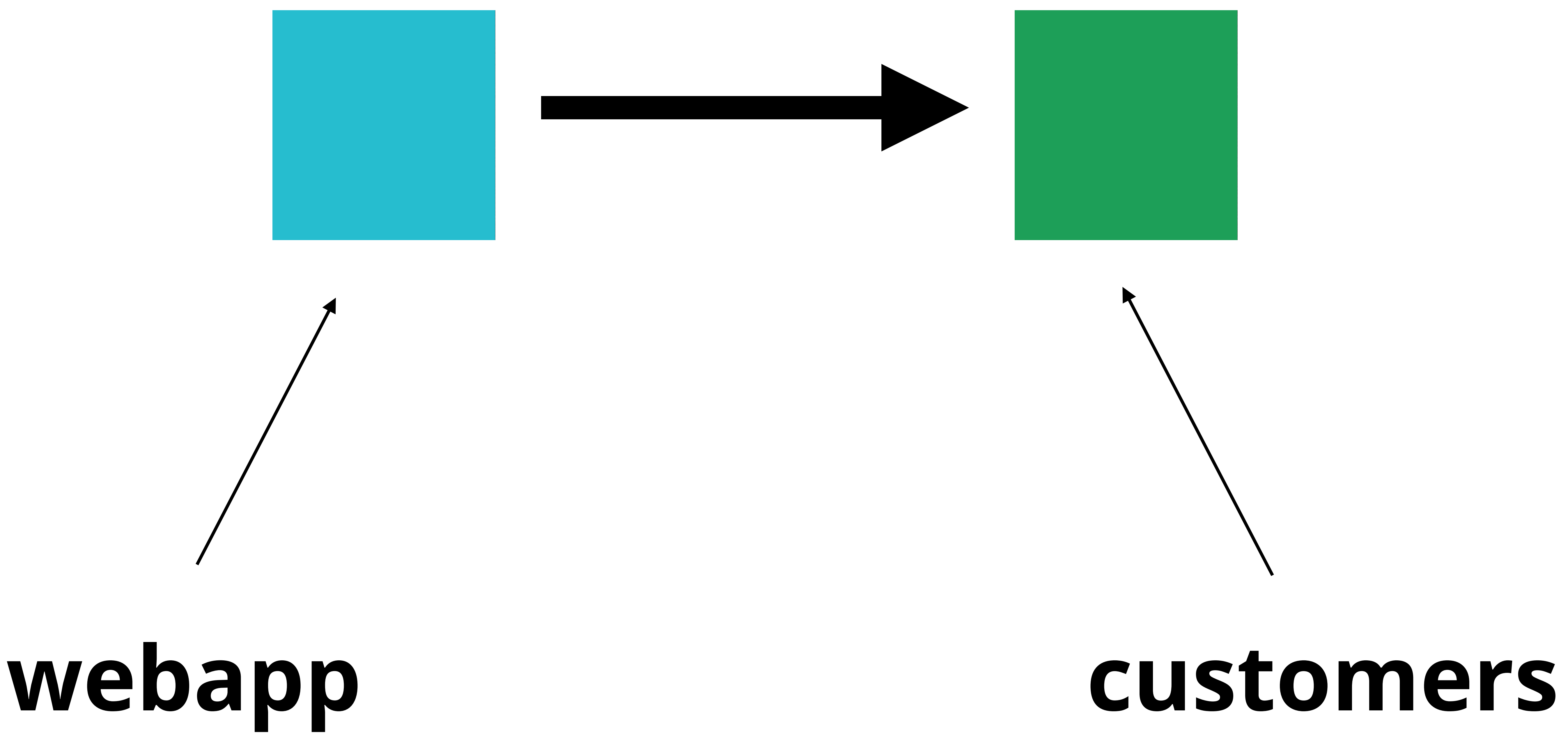

compile, unit and functional test
acceptance test
integration test
deploy to production
run on build machine
deployed on build machine
deployed to integration environment

OK, so we are going to be cool and use microservices

and we might as well call them something interesting

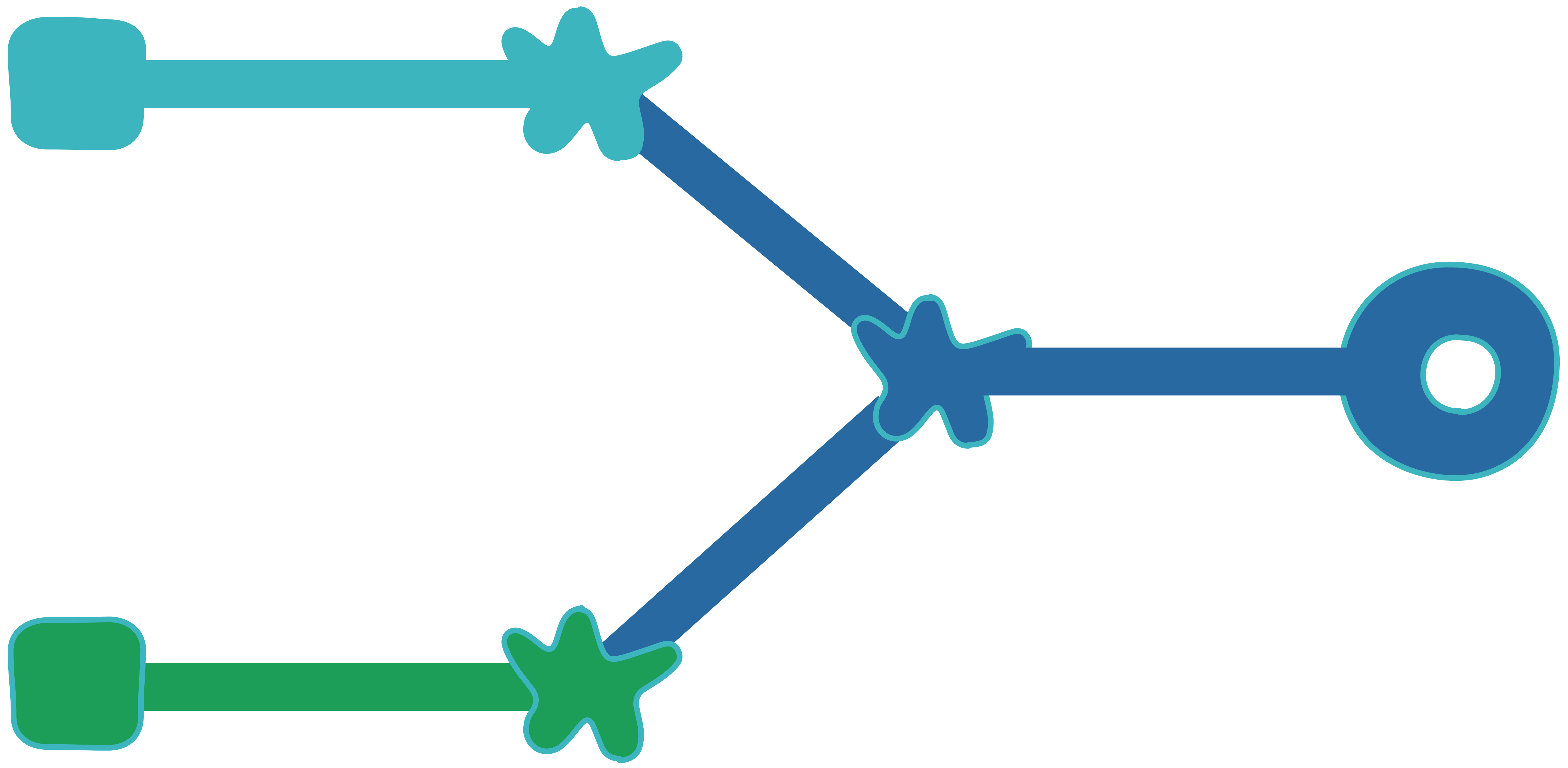


and they have a dependency on one another…



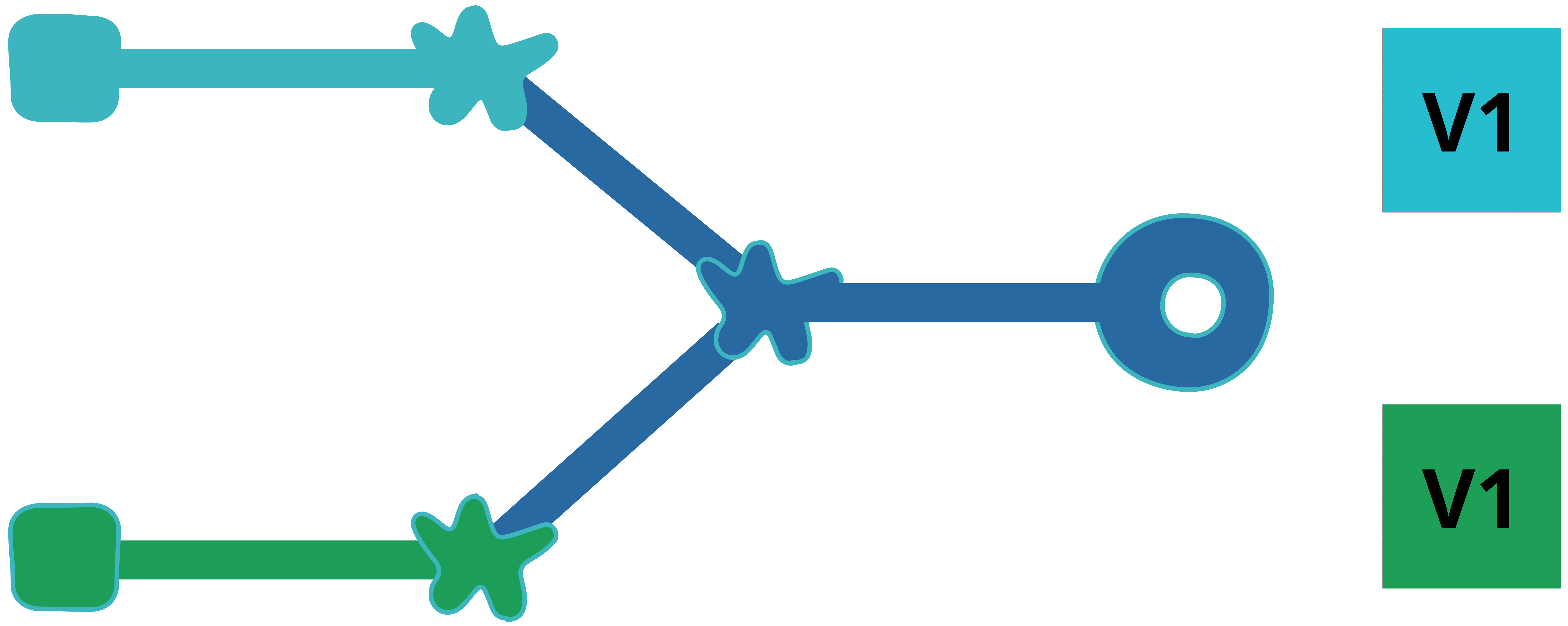
How do we traditionally make sure that new versions of the services work with each other?

Let me illustrate this

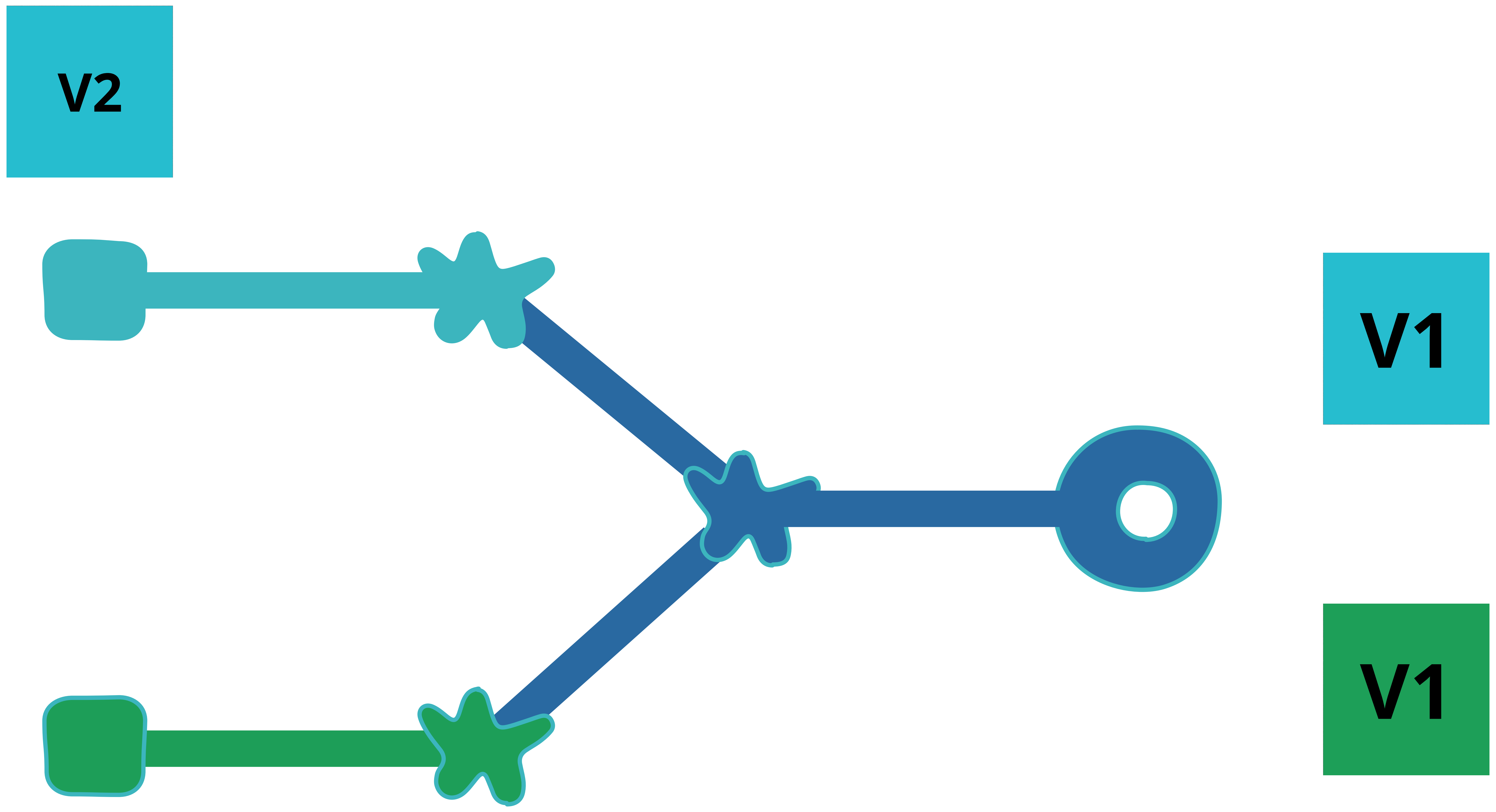


V1
V1

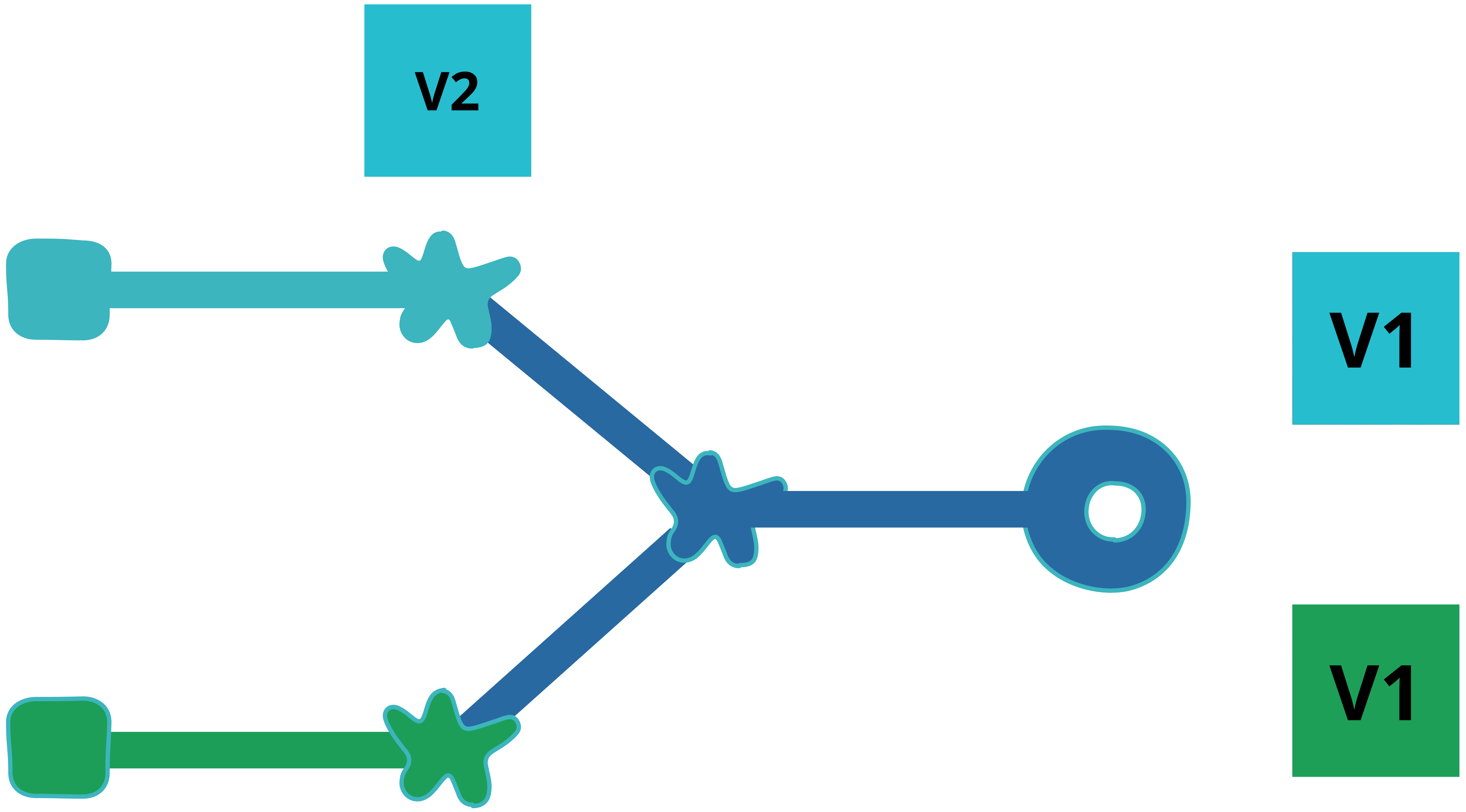
git push origin master



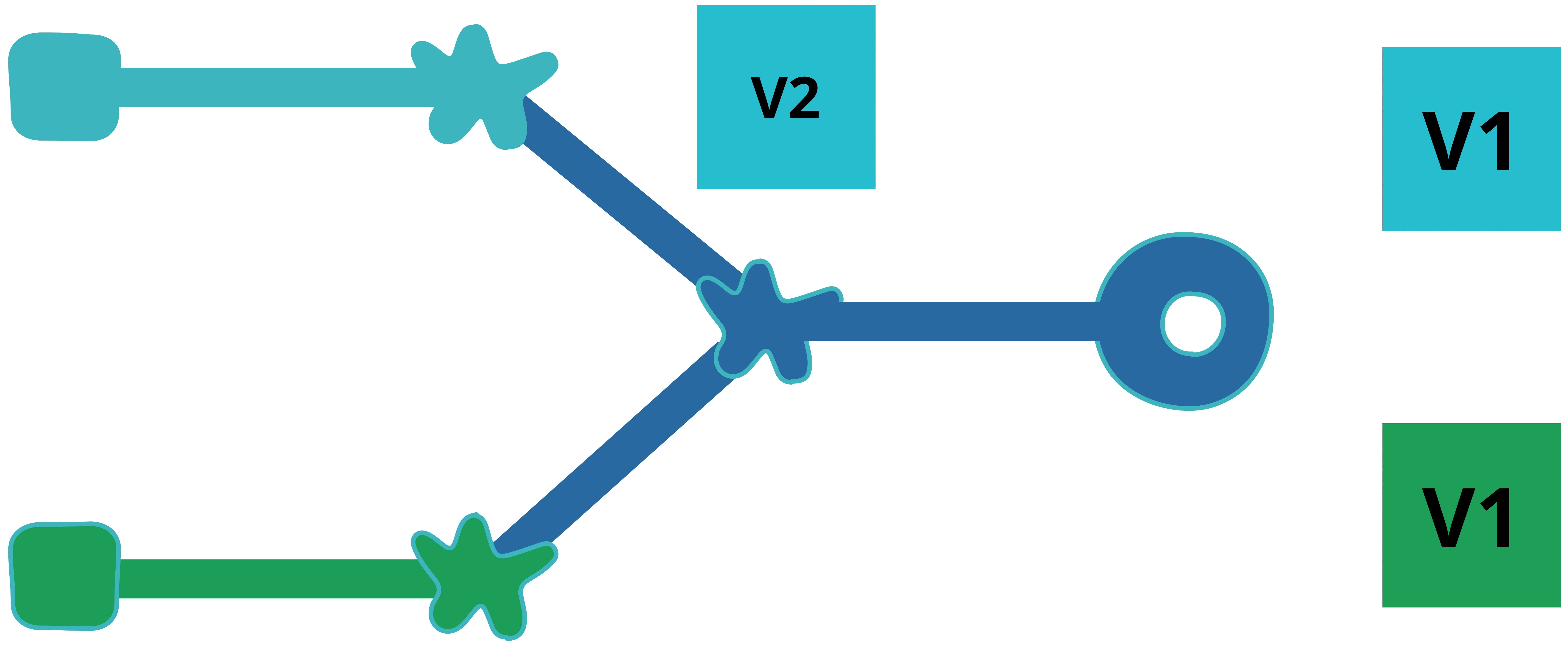
git push origin master



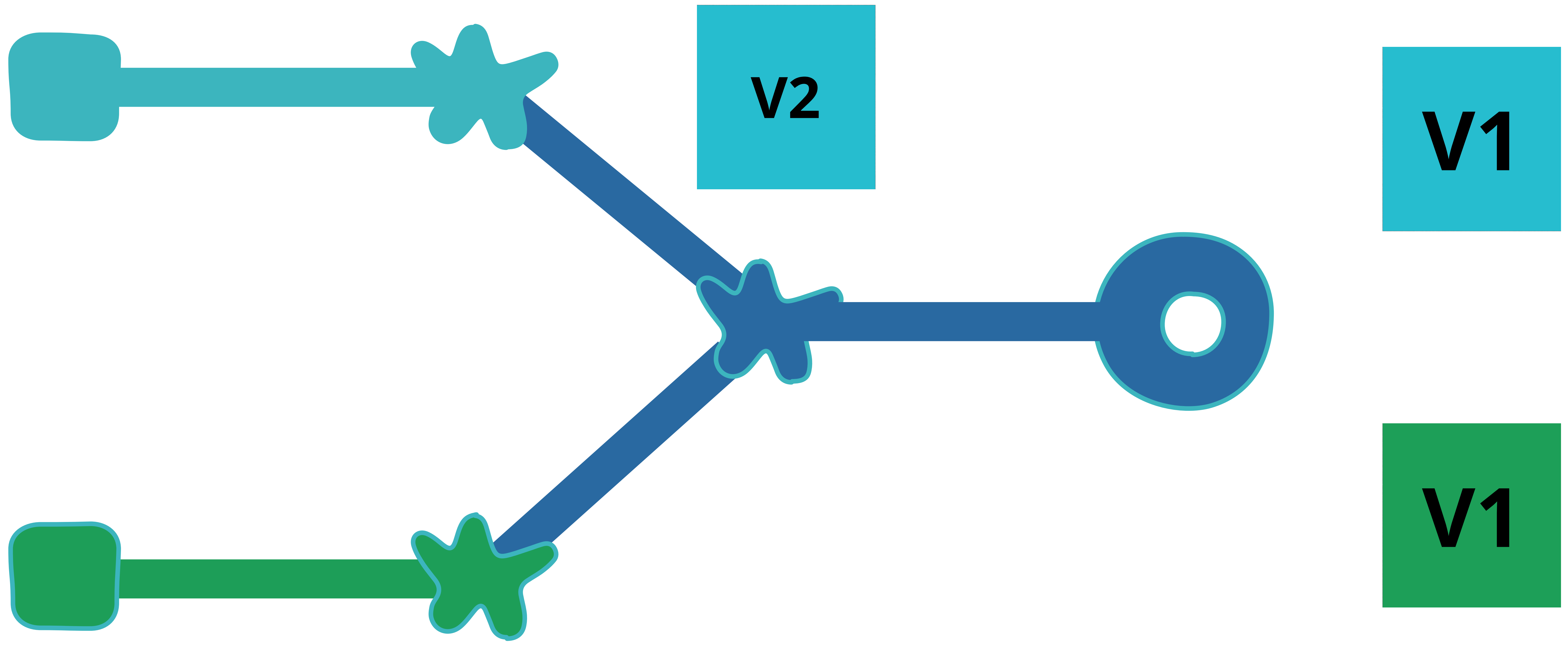
git push origin master



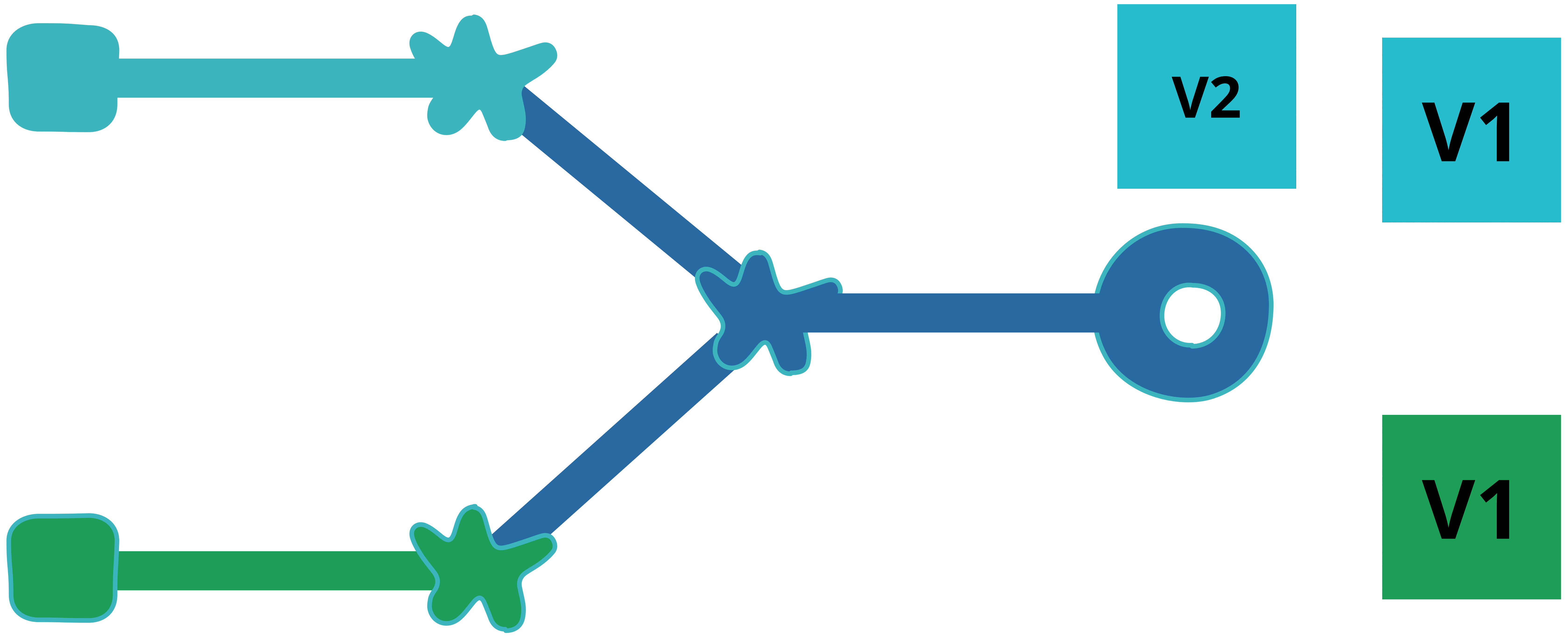
git push origin master

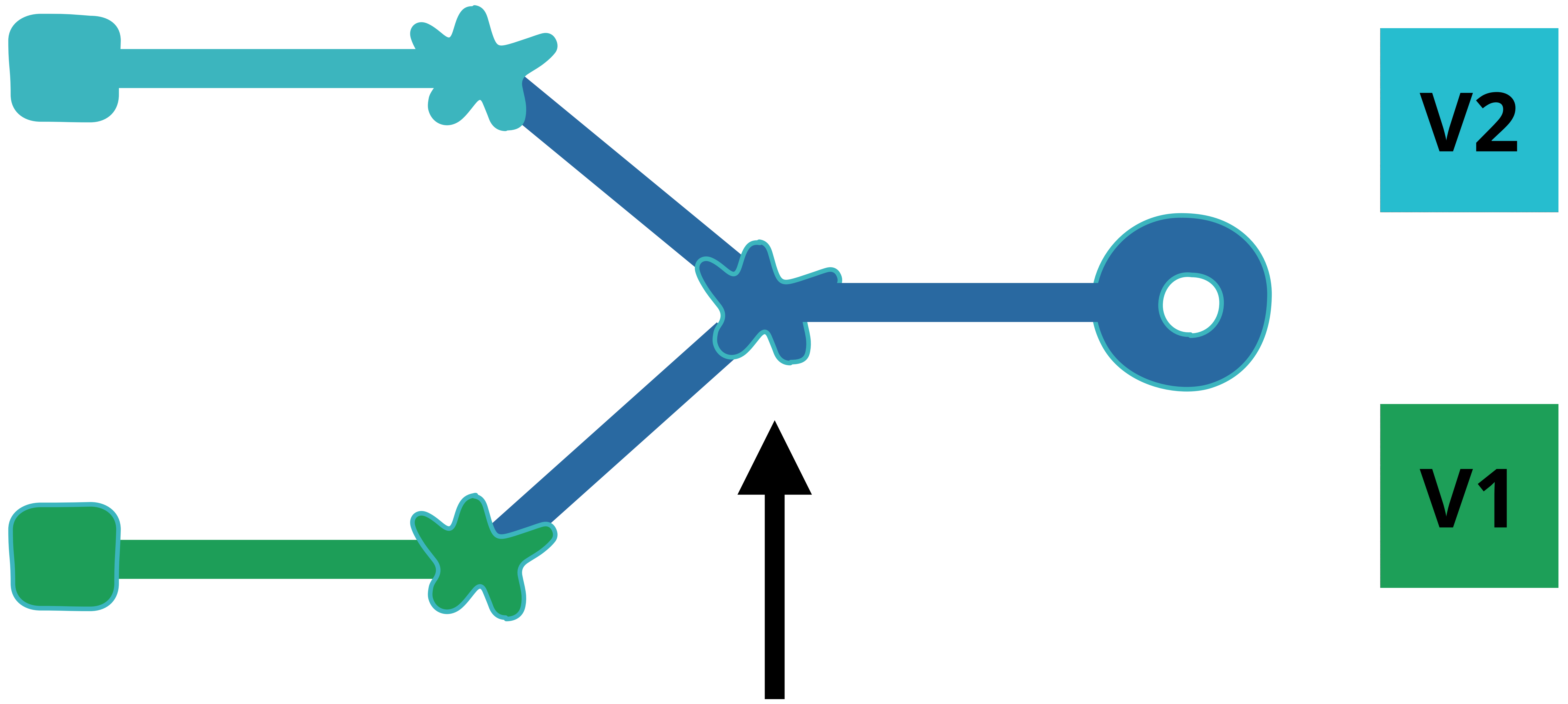


git push origin master



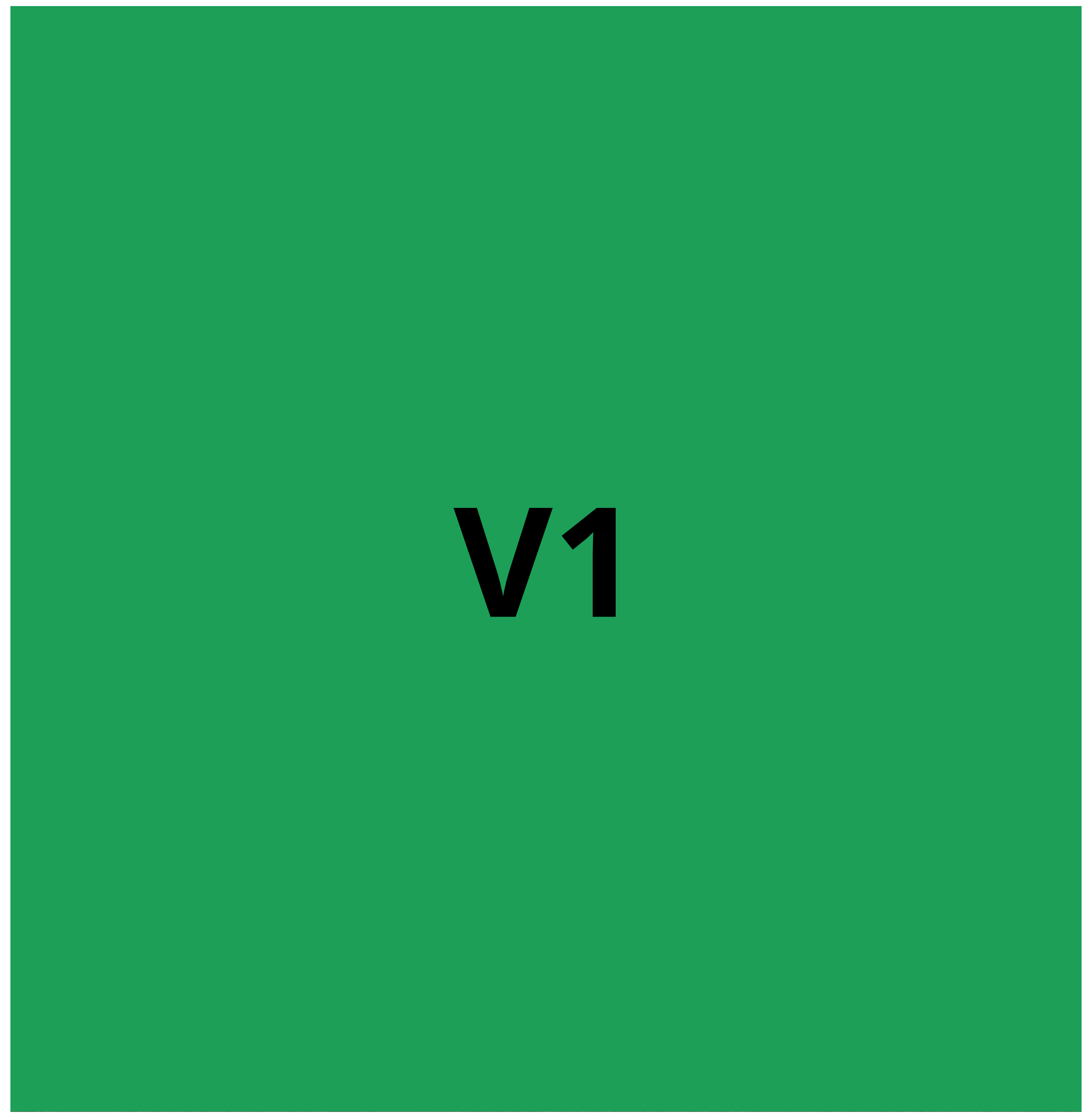
git push origin master

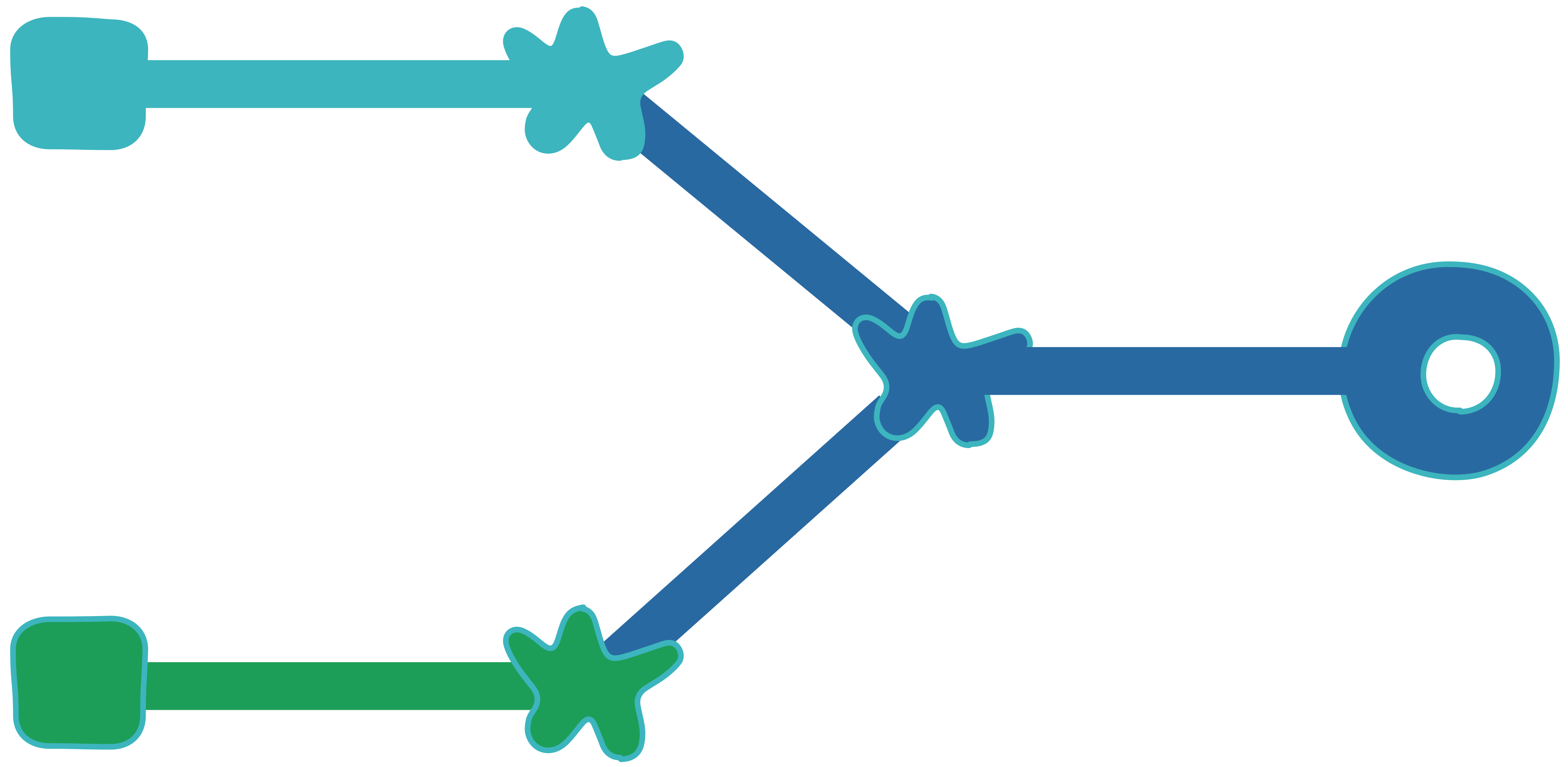




What should V2 of the blue app be tested against here

This is in production, so presumably we should test against this?

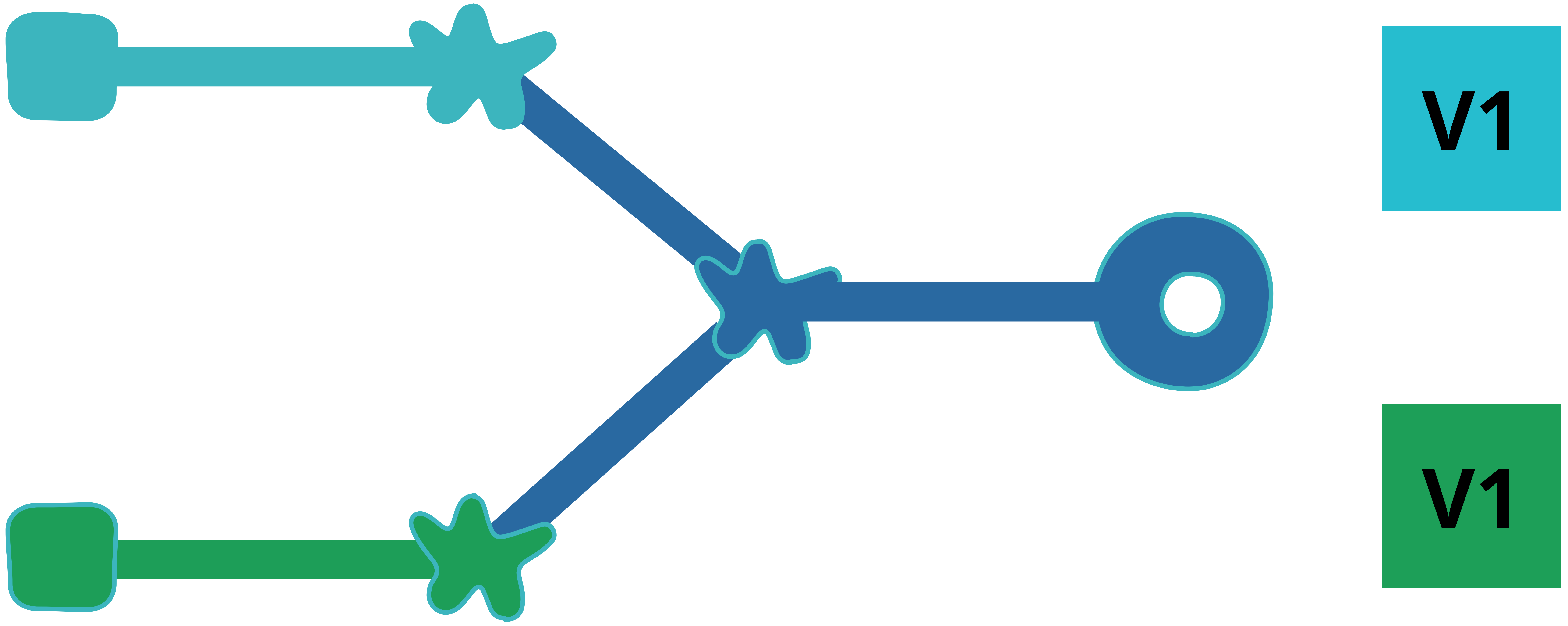




V1

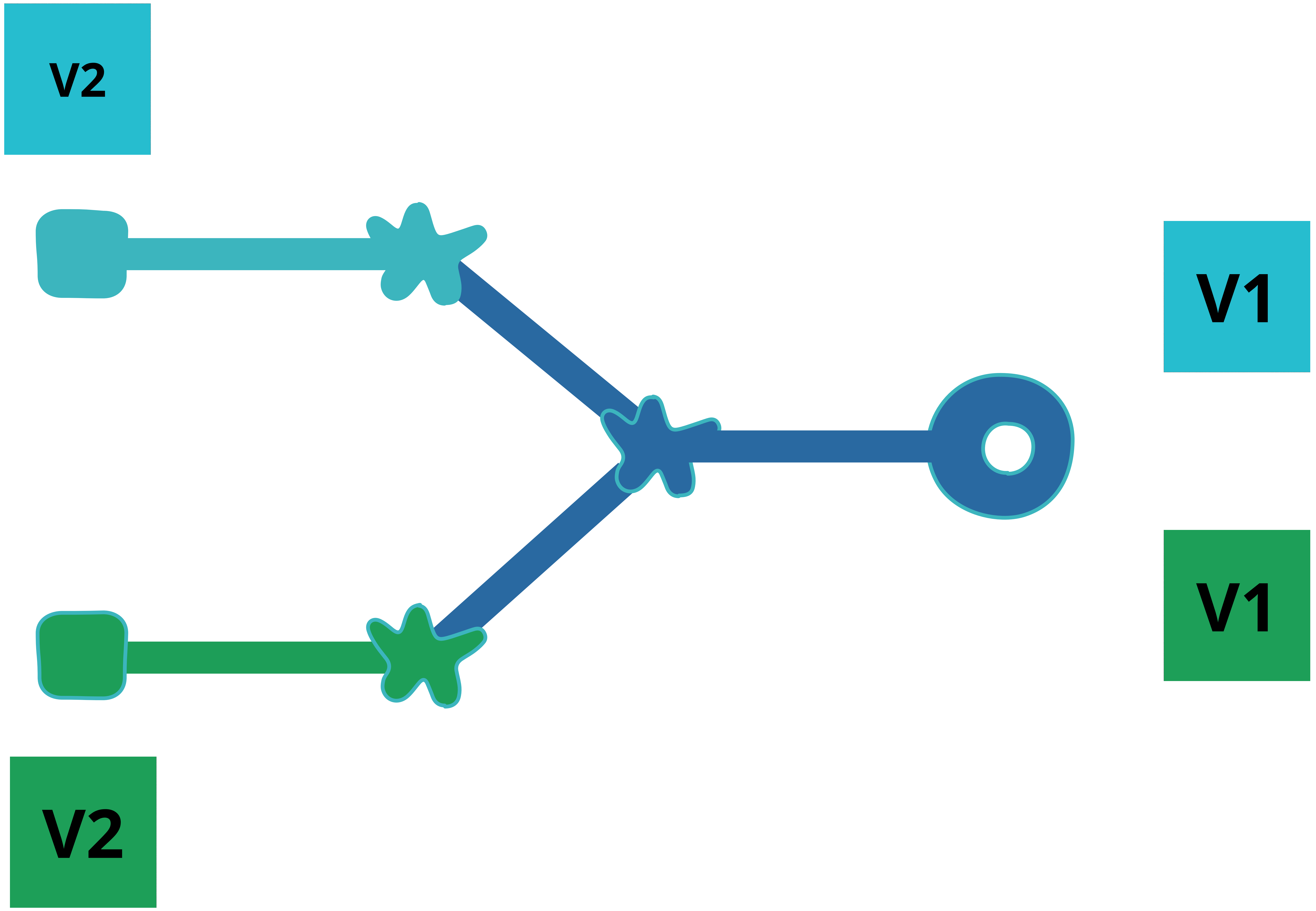
V1

```
git push origin master
```



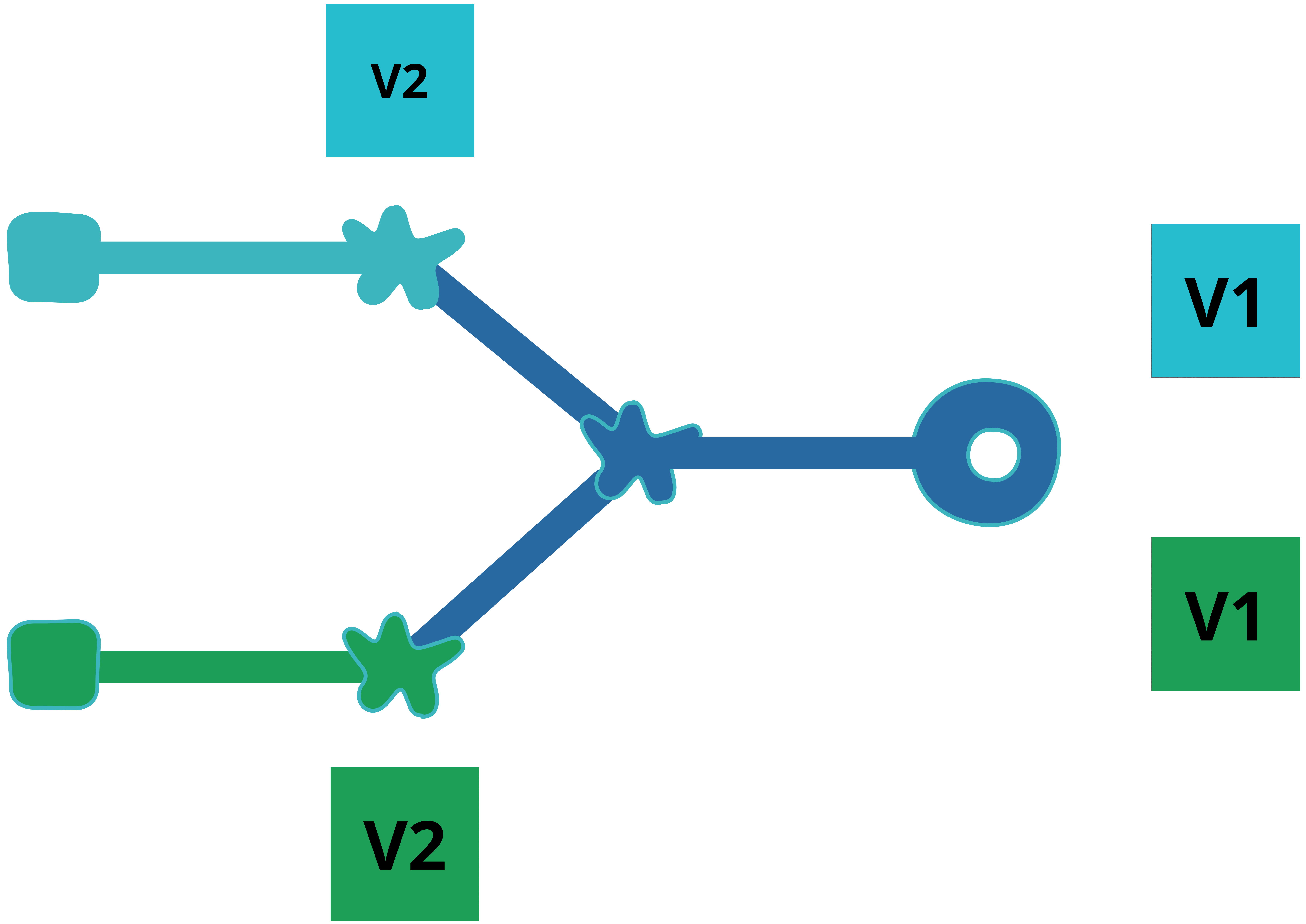
```
git push origin master
```

git push origin master



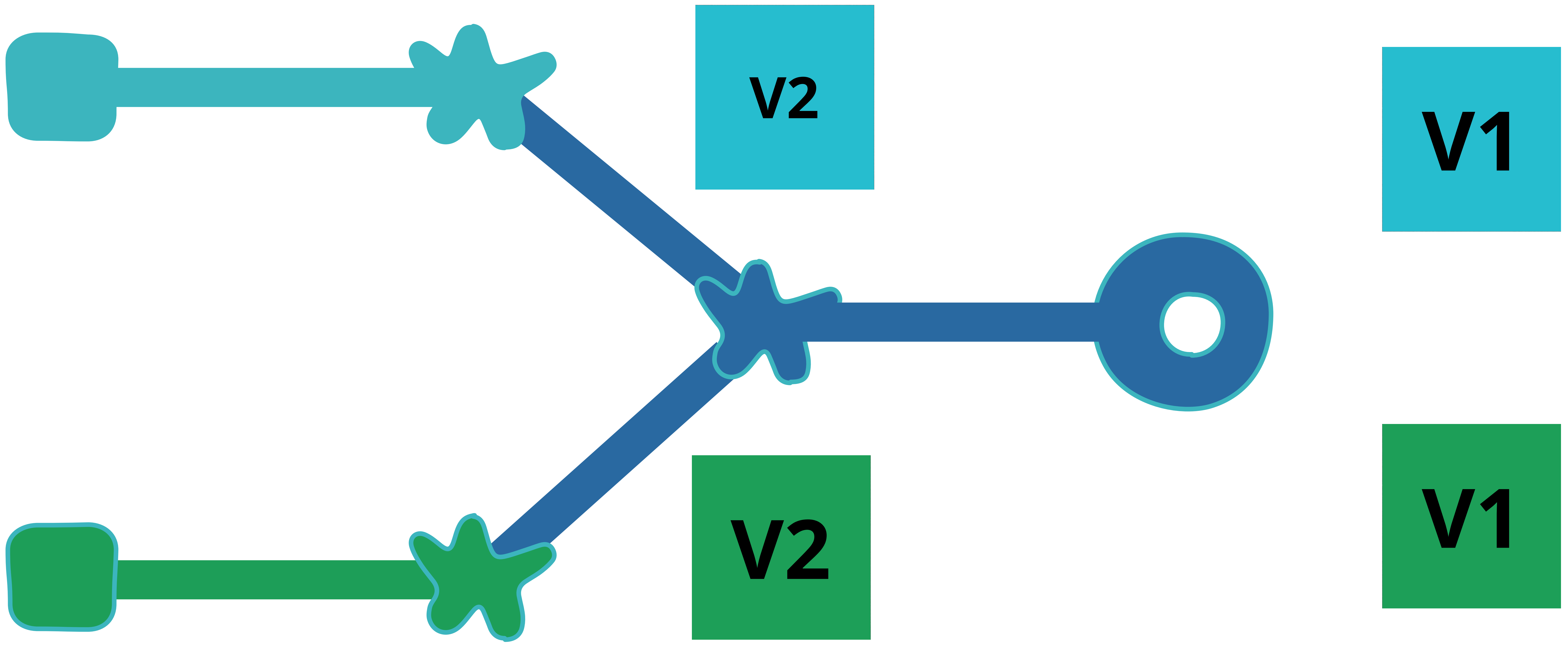
git push origin master

```
git push origin master
```



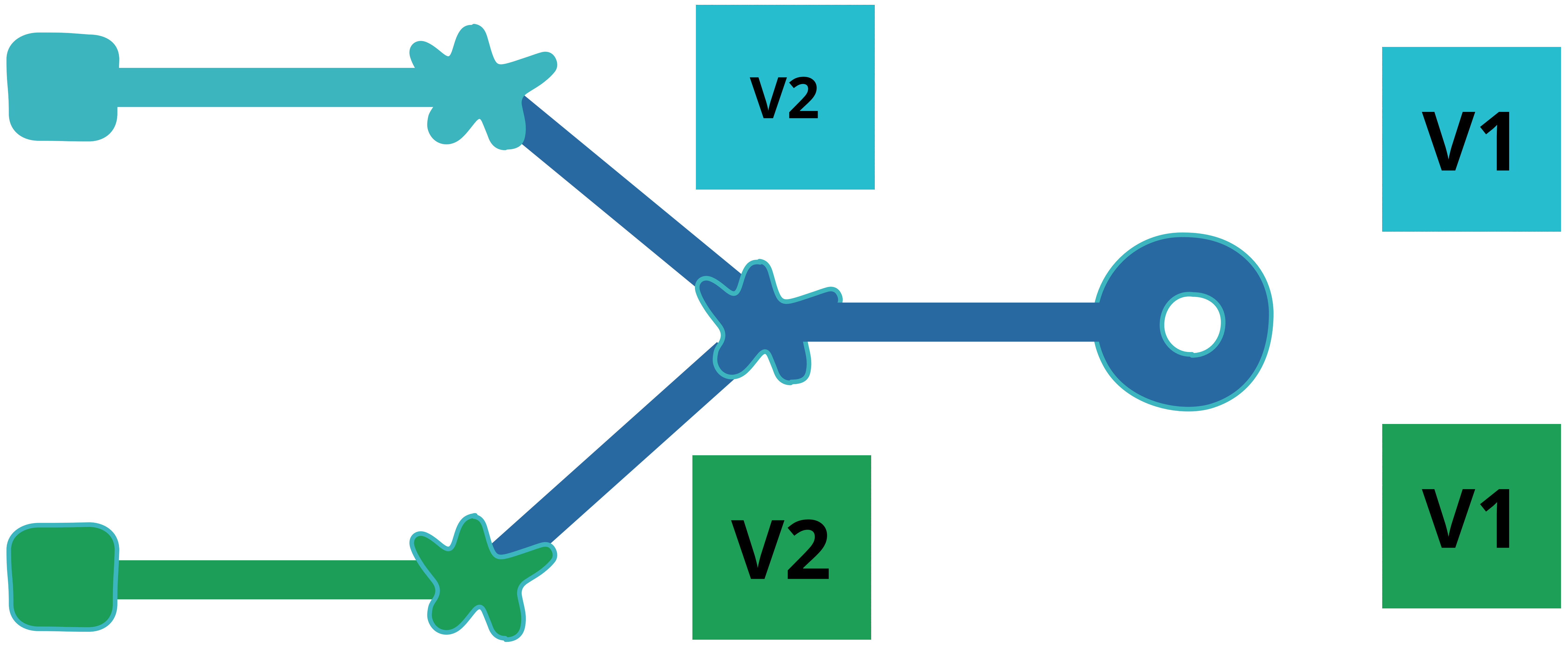
```
git push origin master
```

git push origin master



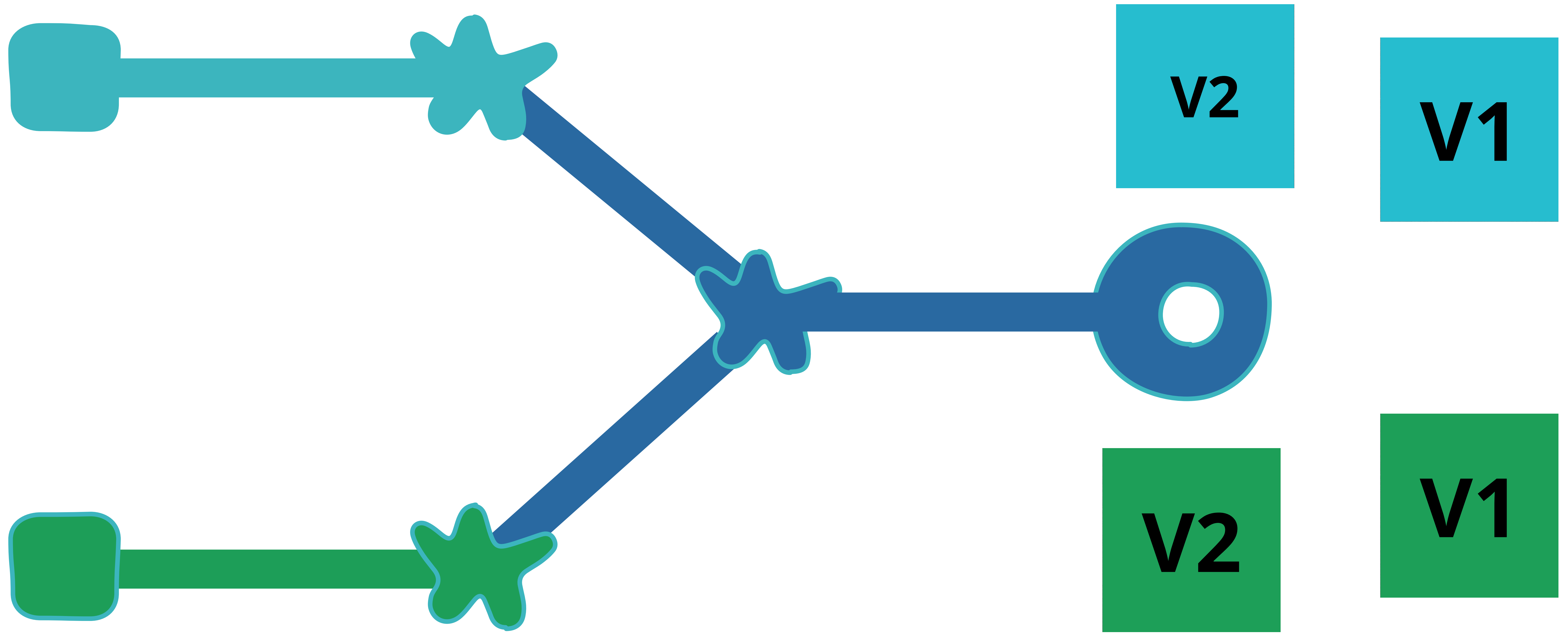
git push origin master

git push origin master



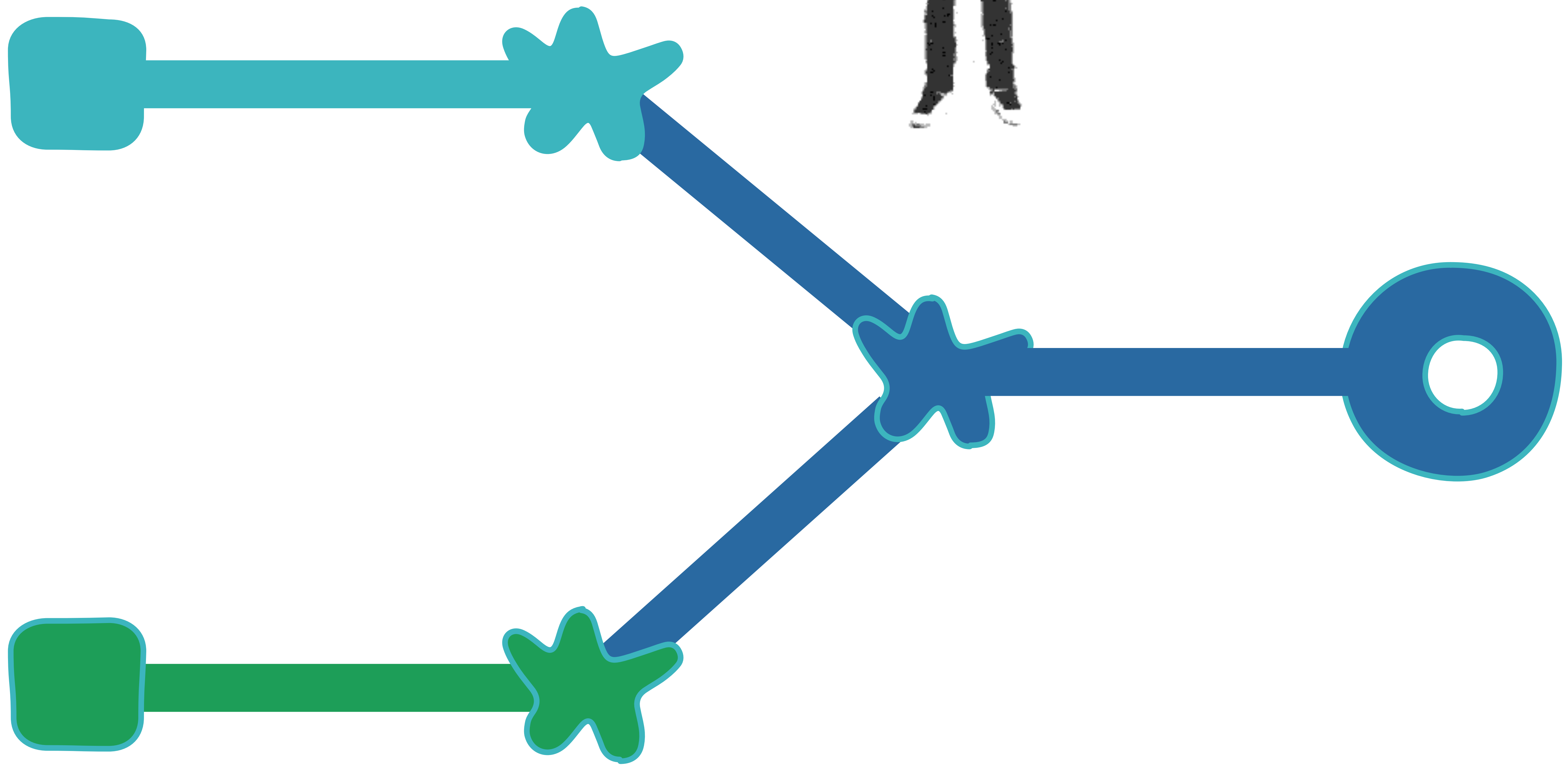
git push origin master

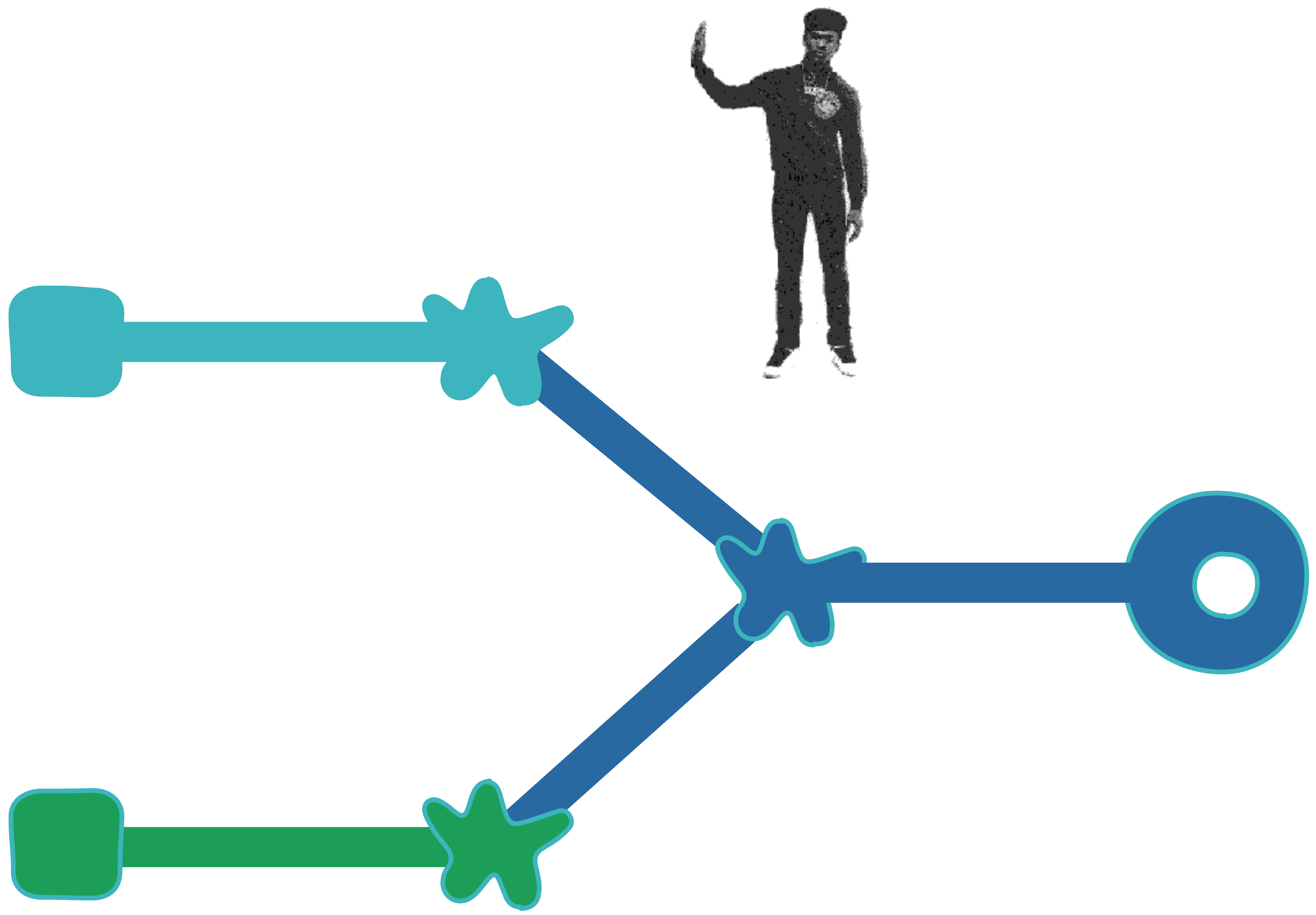
```
git push origin master
```



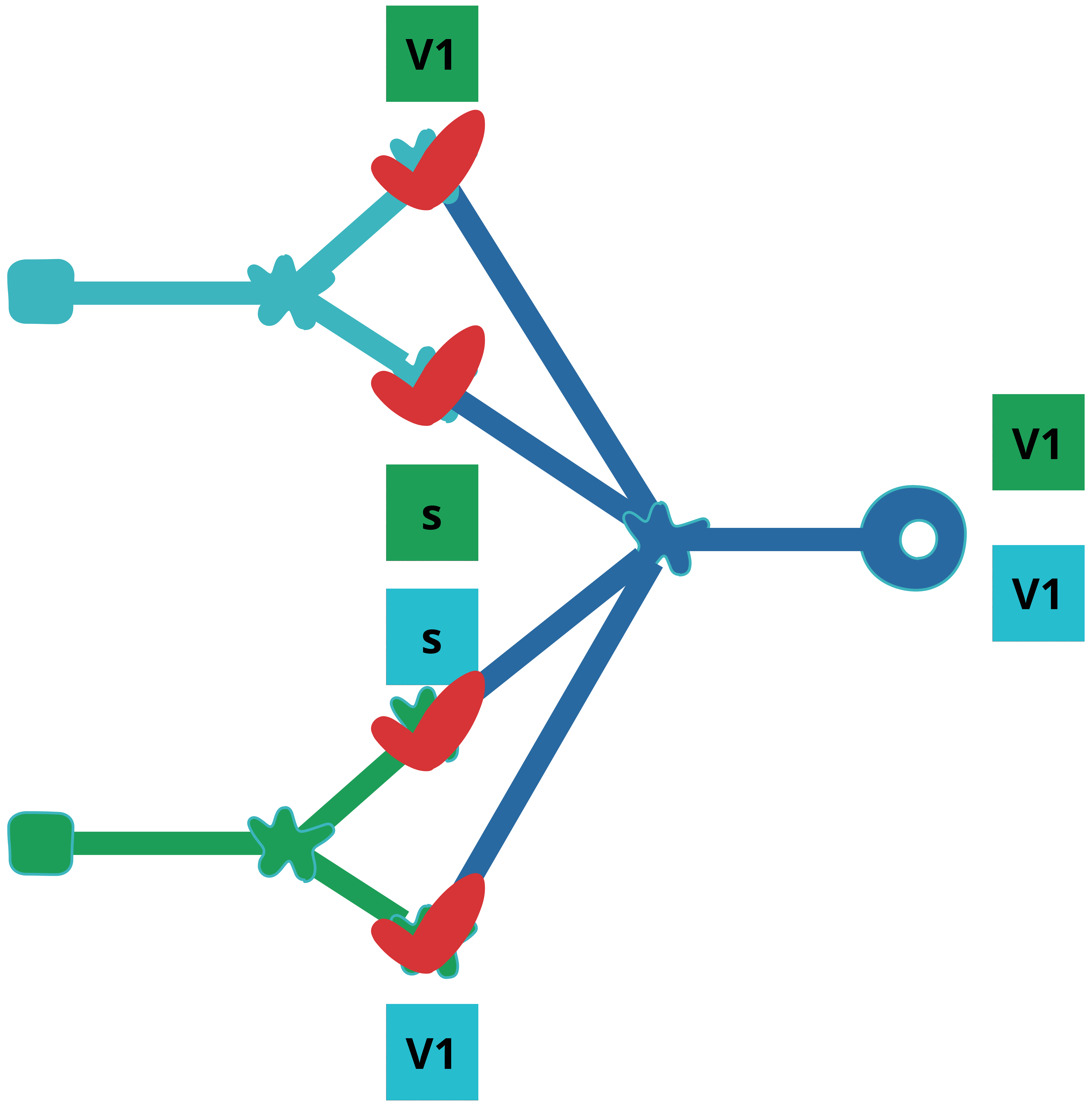
```
git push origin master
```

I’m sorry Dave, I can’t let you do that





Locks == Delay


V1
s
s
V1
V1
V1

amazon
web services™

2 services

$\Downarrow$

4 environments

2 services

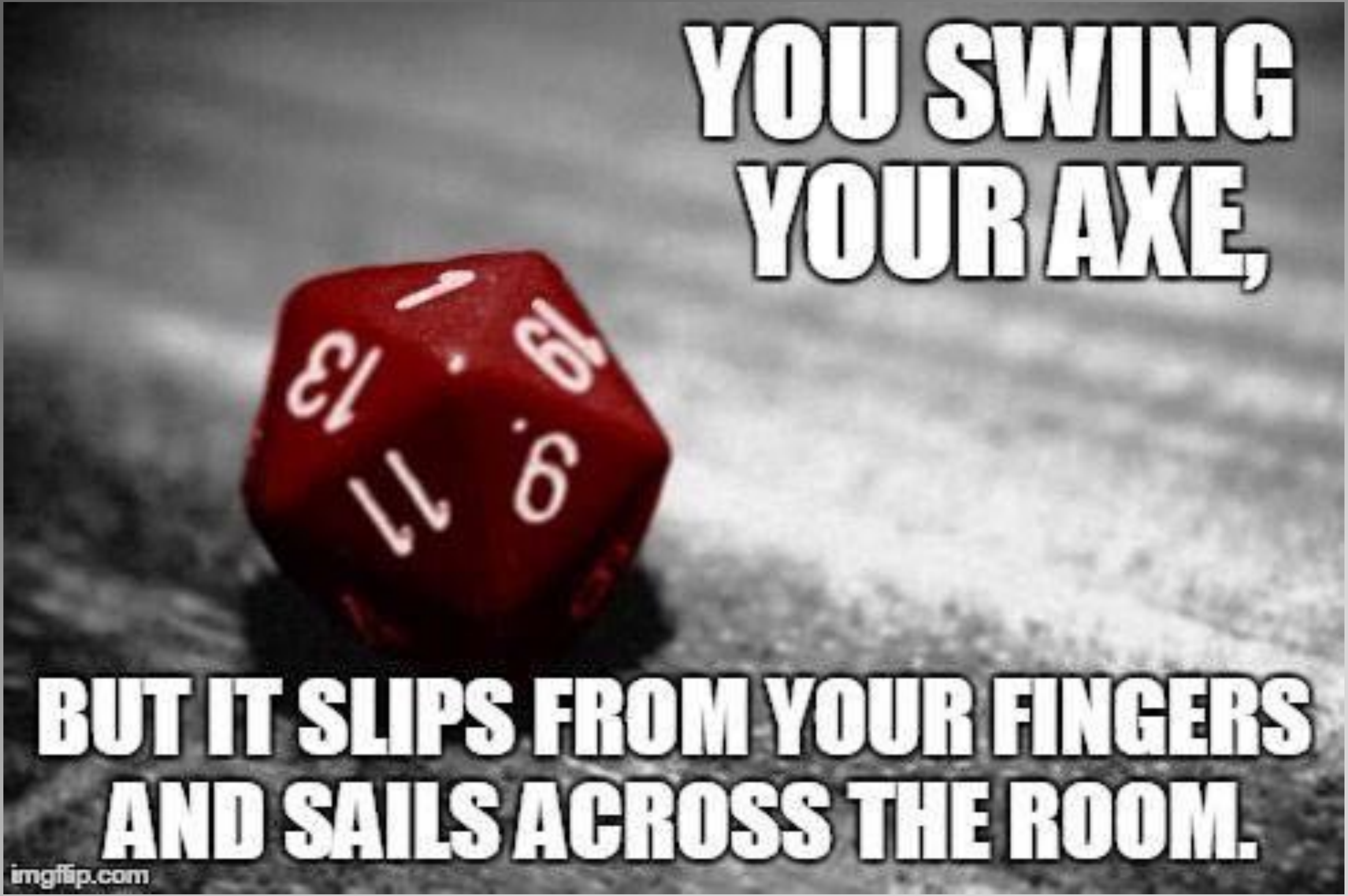
2 services

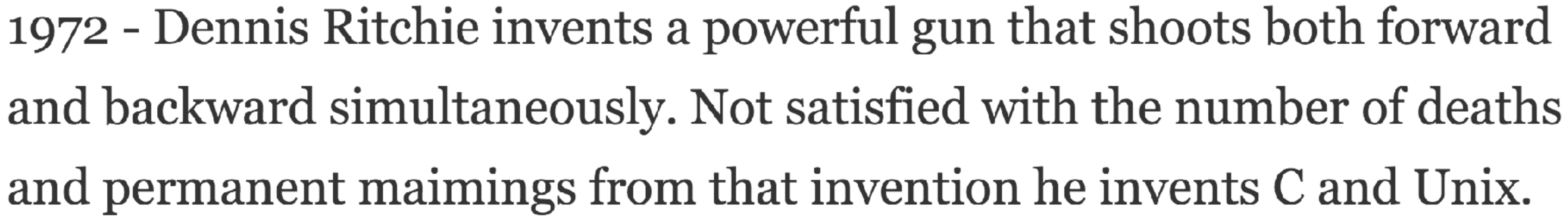
4

2 services

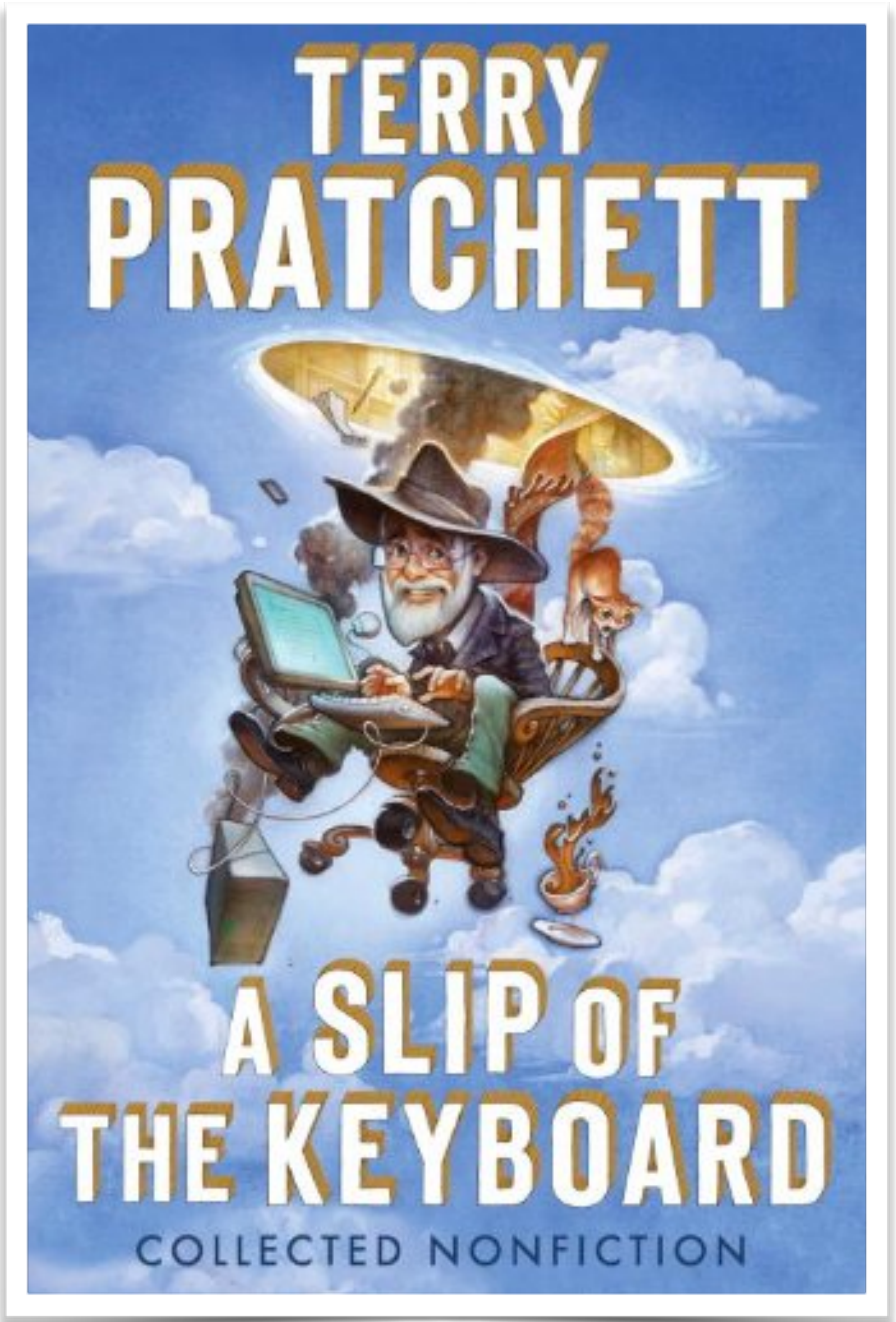
4

>600


YOU SWING YOUR AXE,
BUT IT SLIPS FROM YOUR FINGERS AND SAILS ACROSS THE ROOM.
imgflip.com

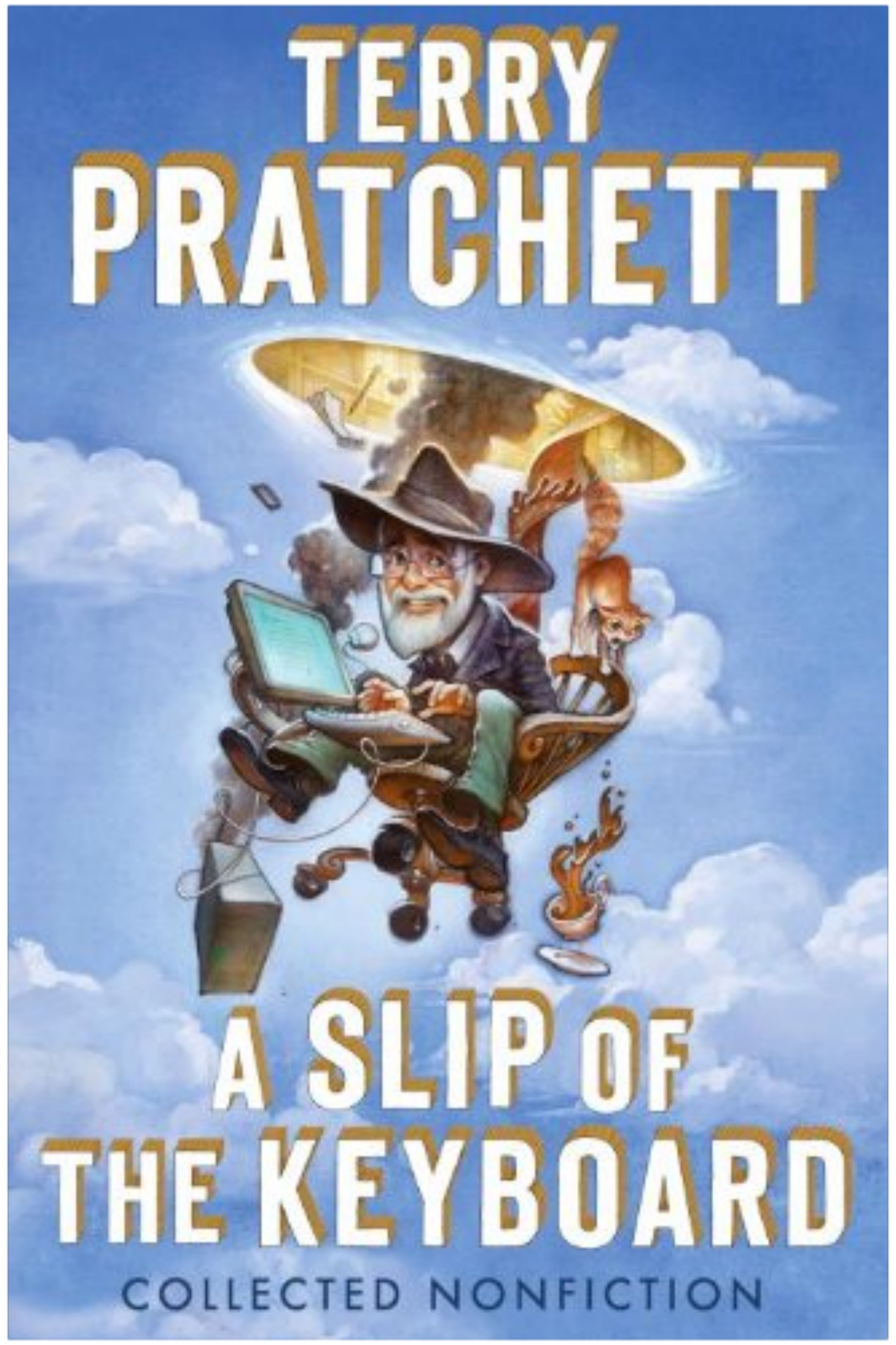

1972 - Dennis Ritchie invents a powerful gun that shoots both forward and backward simultaneously. Not satisfied with the number of deaths and permanent maimings from that invention he invents C and Unix.

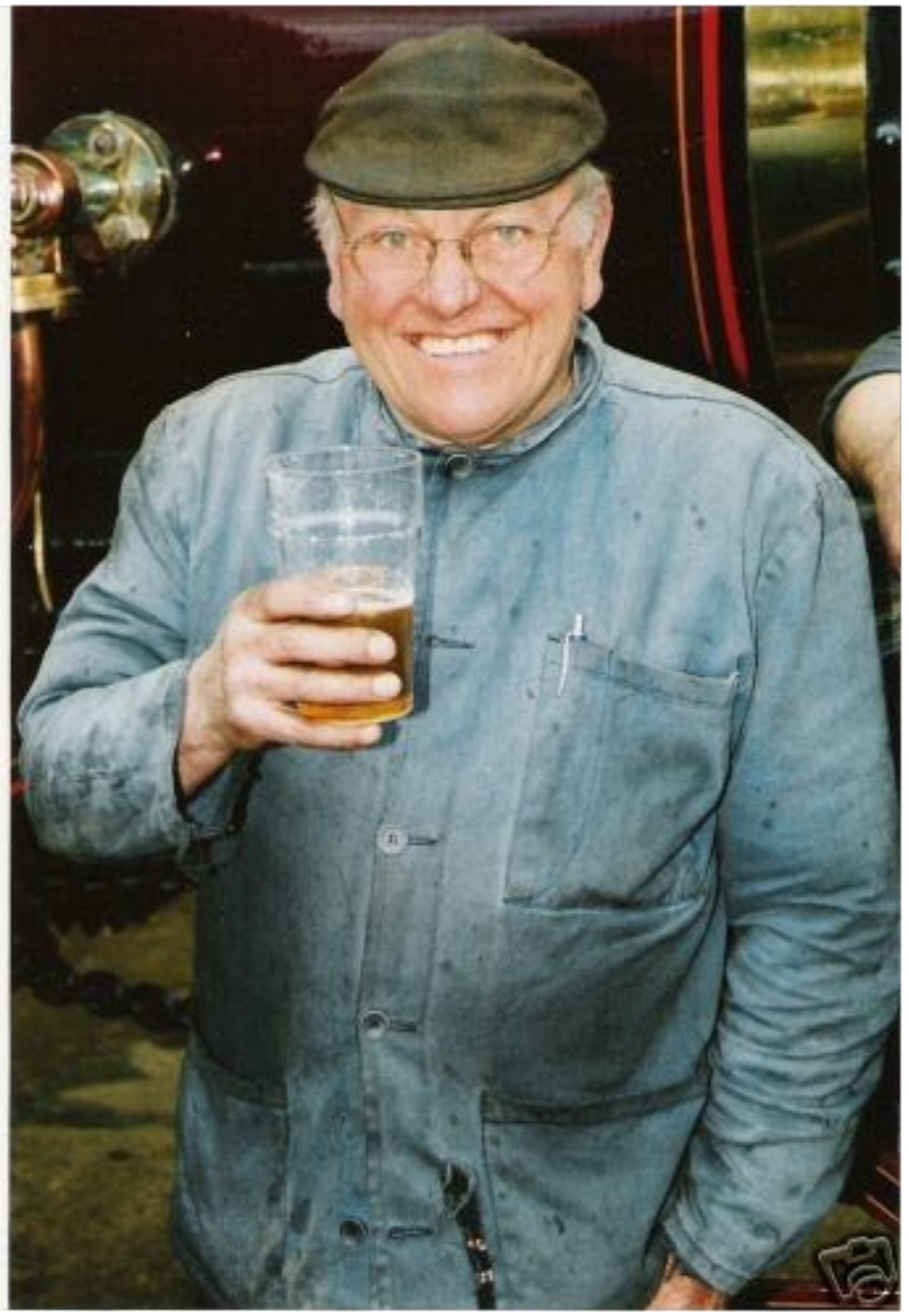
http://james-iry.blogspot.com.au/2009/05/brief-incomplete-and-mostly-wrong.html

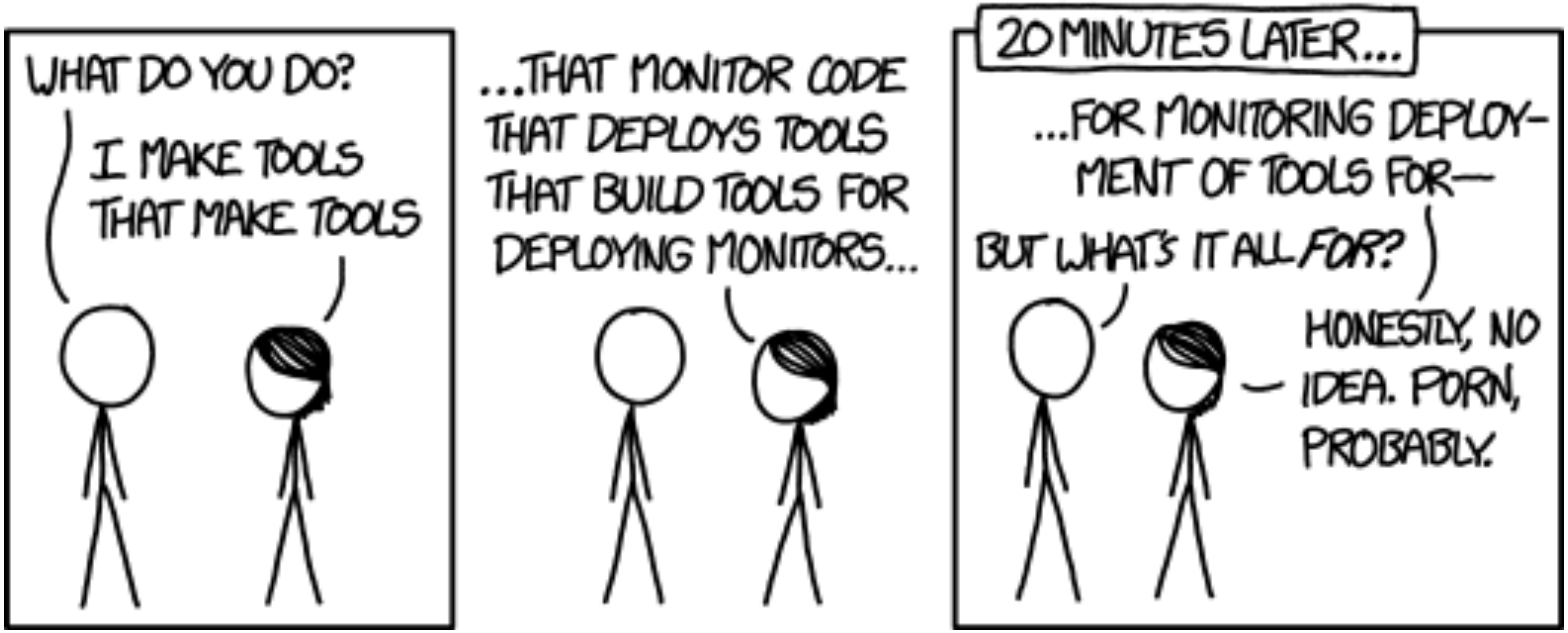

TERRY PRATCHETT

A SLIP OF THE KEYBOARD

COLLECTED NONFICTION


TERRY PRATCHETT
A SLIP OF THE KEYBOARD
COLLECTED NONFICTION




WHAT DO YOU DO?
I MAKE TOOLS THAT MAKE TOOLS
...THAT MONITOR CODE THAT DEPLOYS TOOLS THAT BUILD TOOLS FOR DEPLOYING MONITORS...
20 MINUTES LATER...
...FOR MONITORING DEPLOYMENT OF TOOLS FOR—
BUT WHAT'S IT ALL *FOR*?
HONESTLY, NO IDEA. PORN, PROBABLY.

https://xkcd.com/1629/

Part the Seventh

The Developers Fear

“But while he was seeking with thimbles and care,
A Bandersnatch swiftly drew nigh
And grabbed at the Banker, who shrieked in despair,
For he knew it was useless to fly. “

if:

if:

Integration testing

if:

Integration testing

Independent deployment

if:

Integration testing

Independent deployment

Service versioning / evolution

if:

Integration testing

Independent deployment

Service versioning / evolution

is hard

What about some of our other sacred cows?

What about some of our other sacred cows?

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

emergent design

GoF

Continuous Delivery

TDD

XP

KISS

Refactoring

GRASP

World of Warcraft

SOLID

agile

DRY

YAGNI

BDD

ergent desig

G

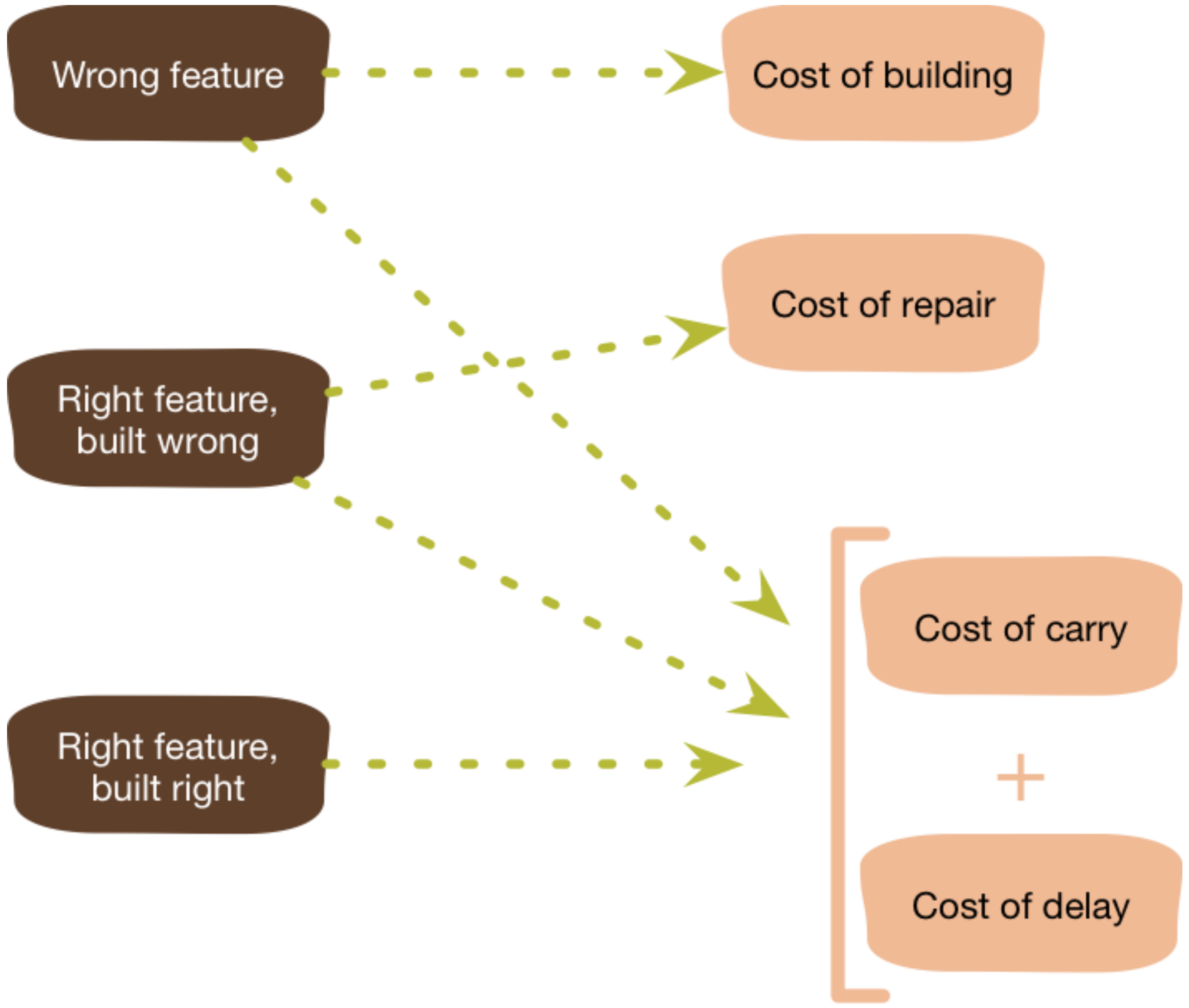
Continuous Delivery

TDD

XP

KISS

Refactoring



http://martinfowler.com/bliki/Yagni.html

build out services as you need them

minimise holding cost and batch size

YAGNI

World of Warcraft

SOLID

agile

DRY

GRASP

BDD

emergent design

GoF

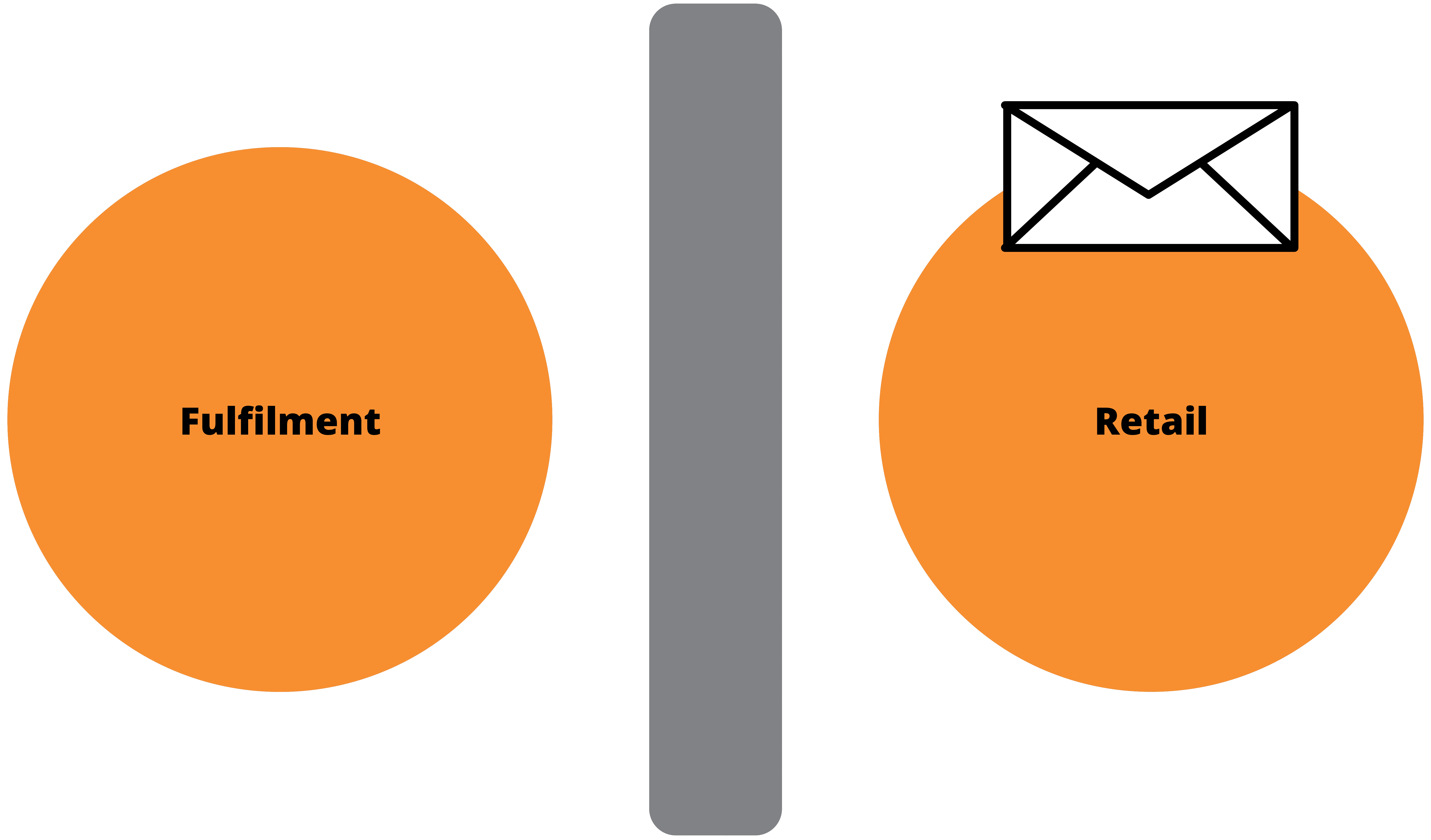
Continuous Delivery

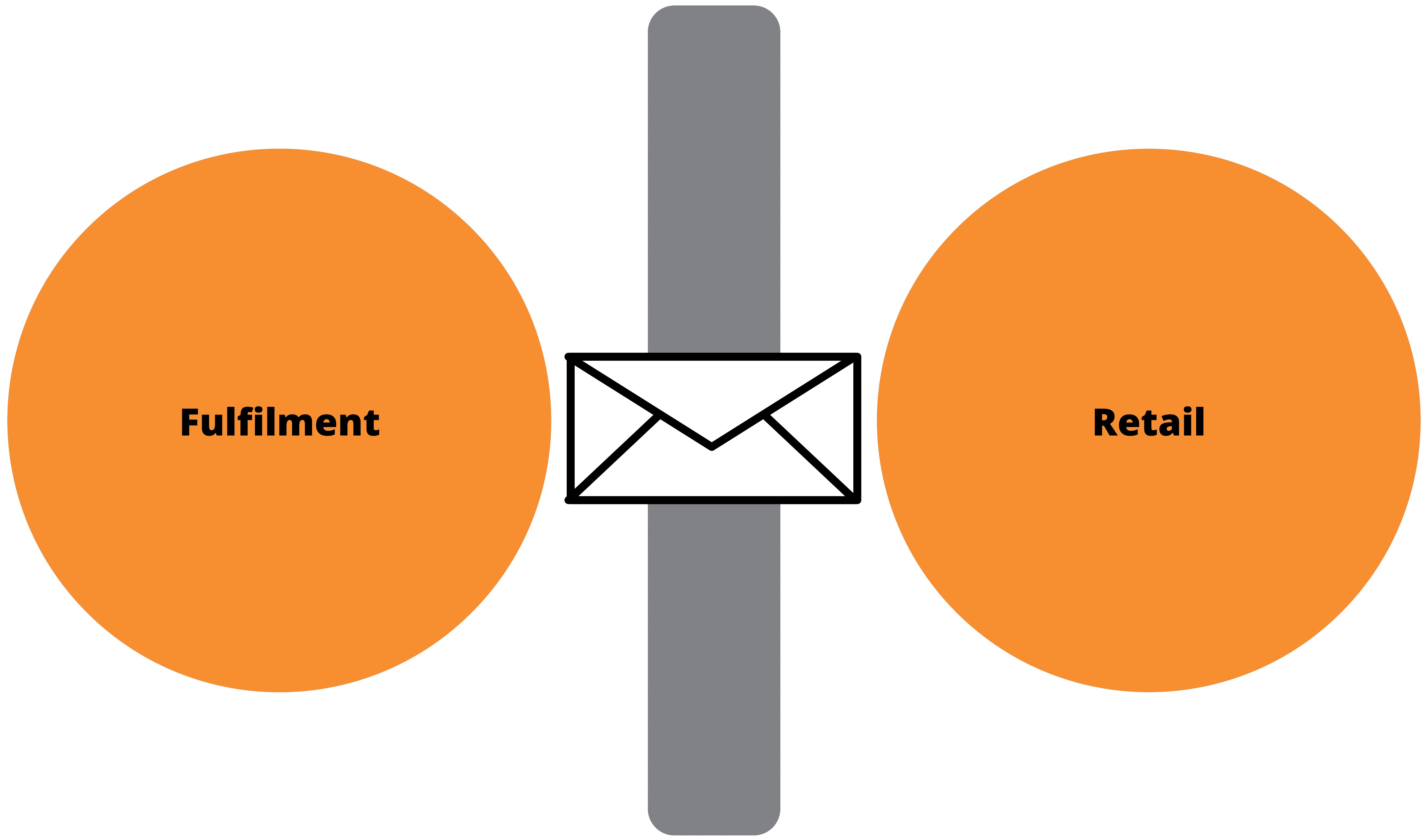
TDD

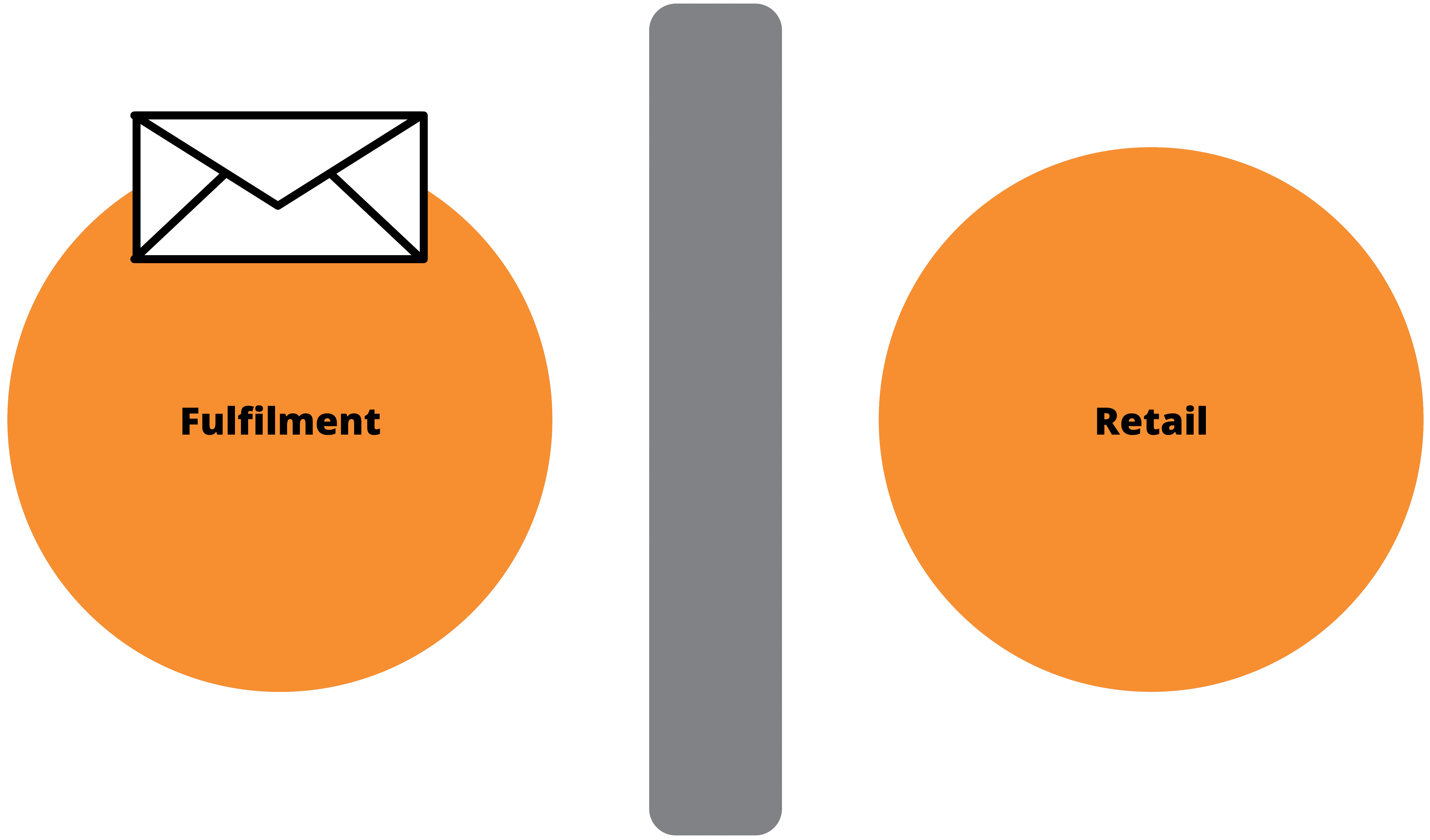
XP

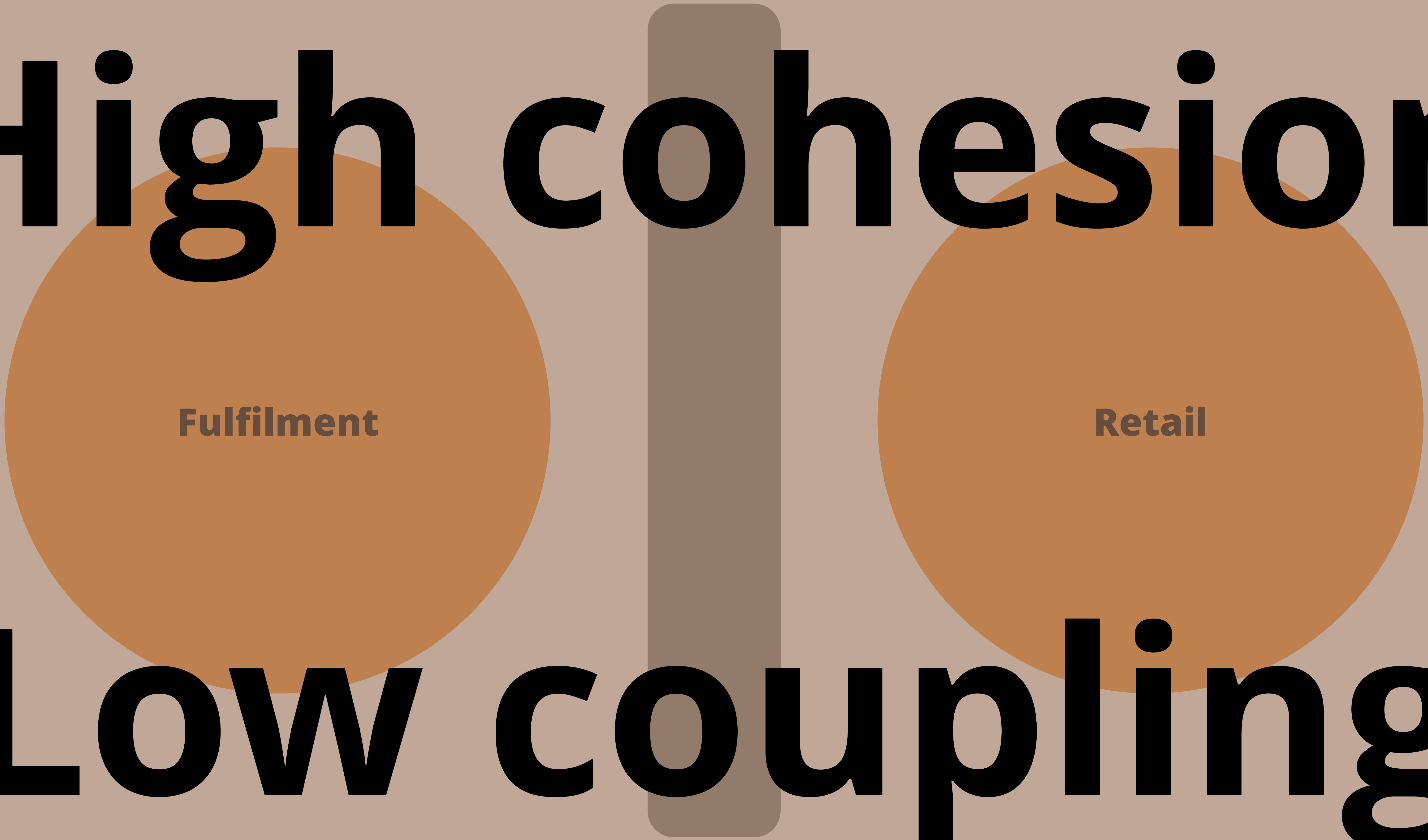
KISS

Refactoring


Fulfilment
Retail


Fulfilment
Retail


Fulfilment
Retail



High cohesion

Low coupling

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

e[illegible] design

GoF

Continuous Delivery

TDD

XP

KISS

Refactoring

“Every piece of knowledge must have a single, unambiguous, authoritative representation within a system”

Andrew Hunt, Dave Thomas: The Pragmatic Programmer. 1999-10-01. ISBN: 978-0-2016-1622-4
https://pragprog.com/book/tpp/the-pragmatic-programmer

STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE
THE ENDLESSLY BIFURCATING TROUSERS OF REALITY
CREATED WITH PULP-O-MIZER COVER MAKER

DRY holds across services

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

TDD

BDD

emerg des n

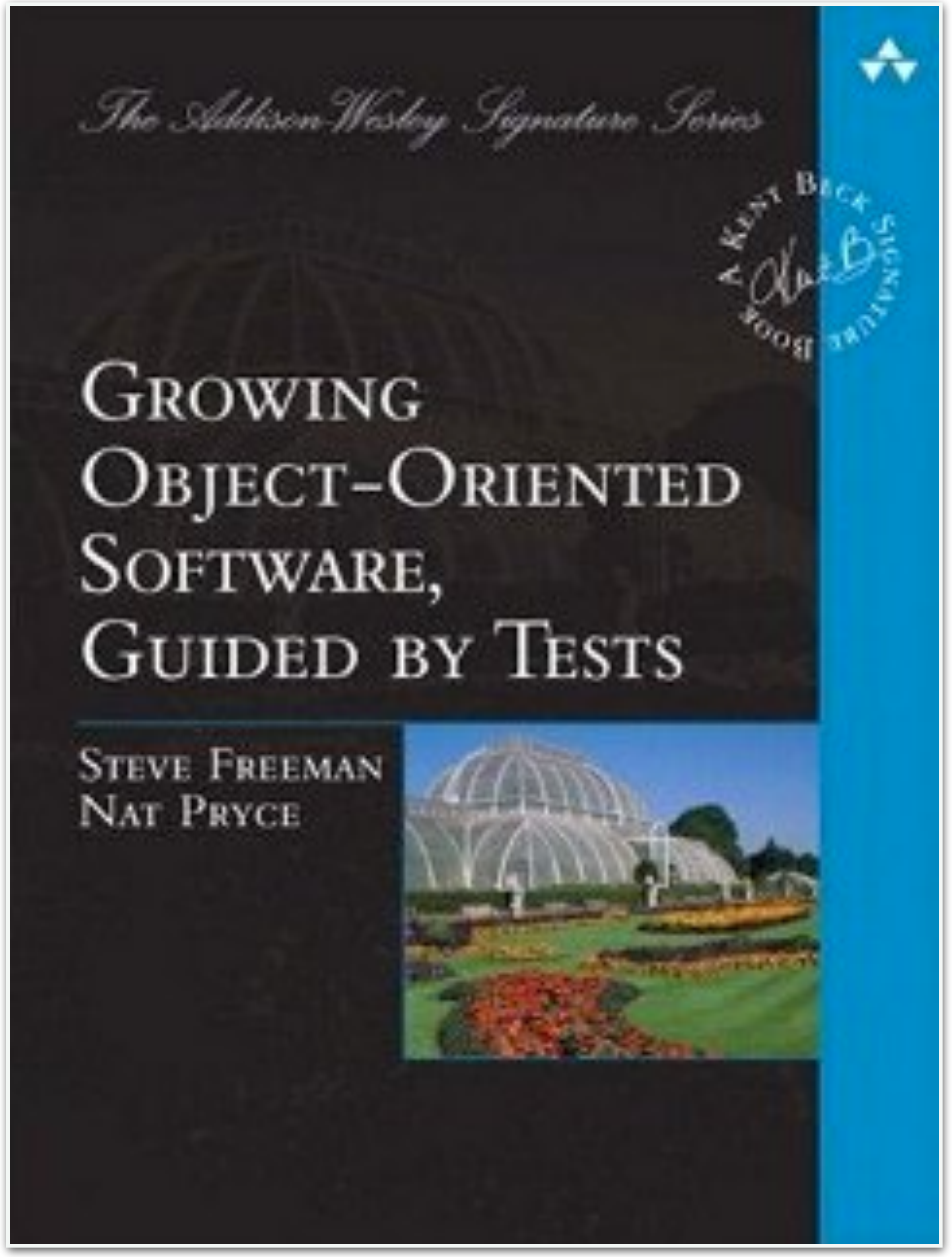
GoF

Continuous Delivery

XP

KISS

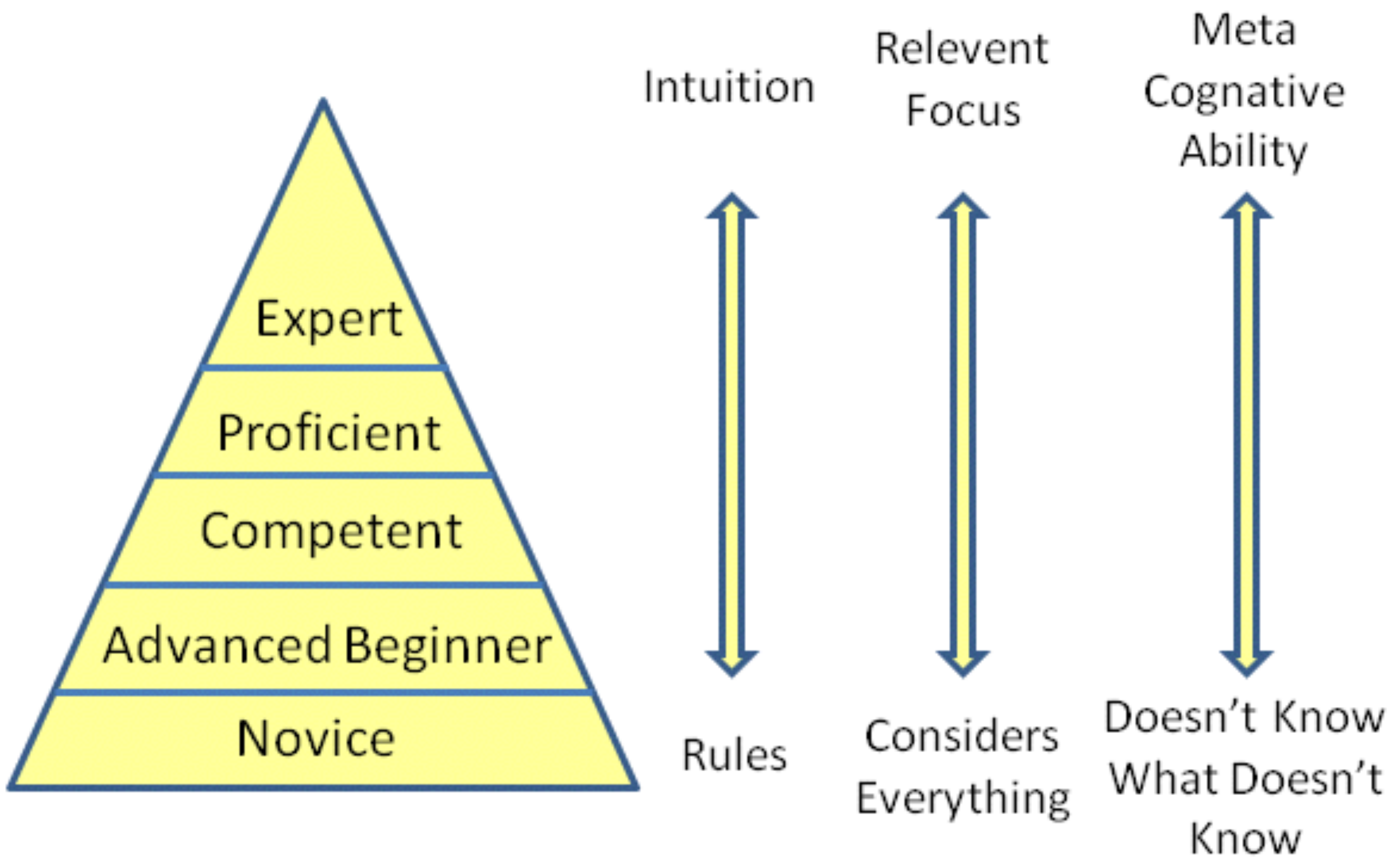
Refactoring


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

"The London school of Test Driven Development"

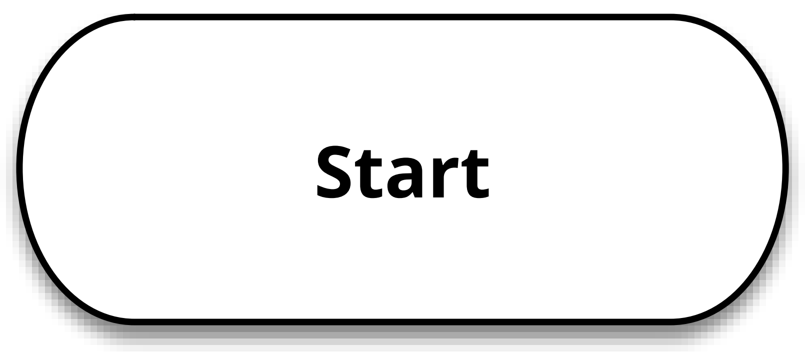
Mike Feathers

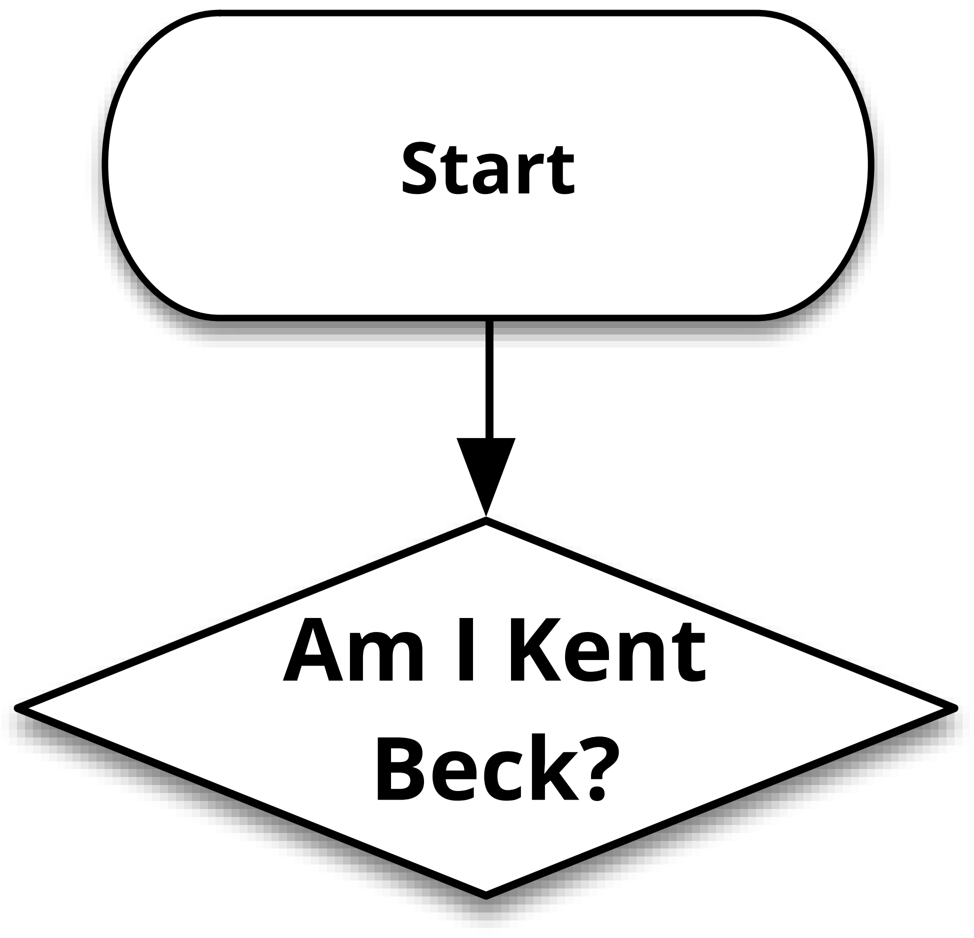
should we bother with test driving our code if we are going to throw it away?

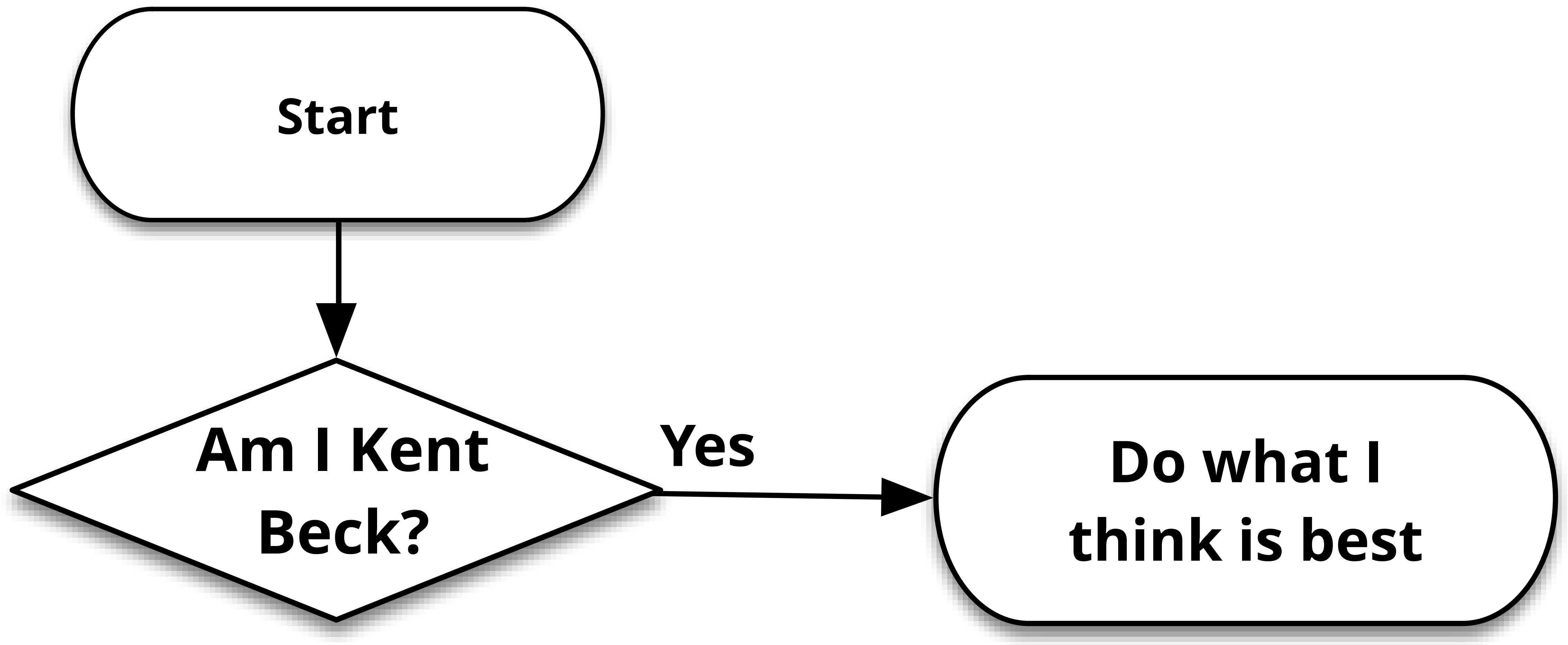


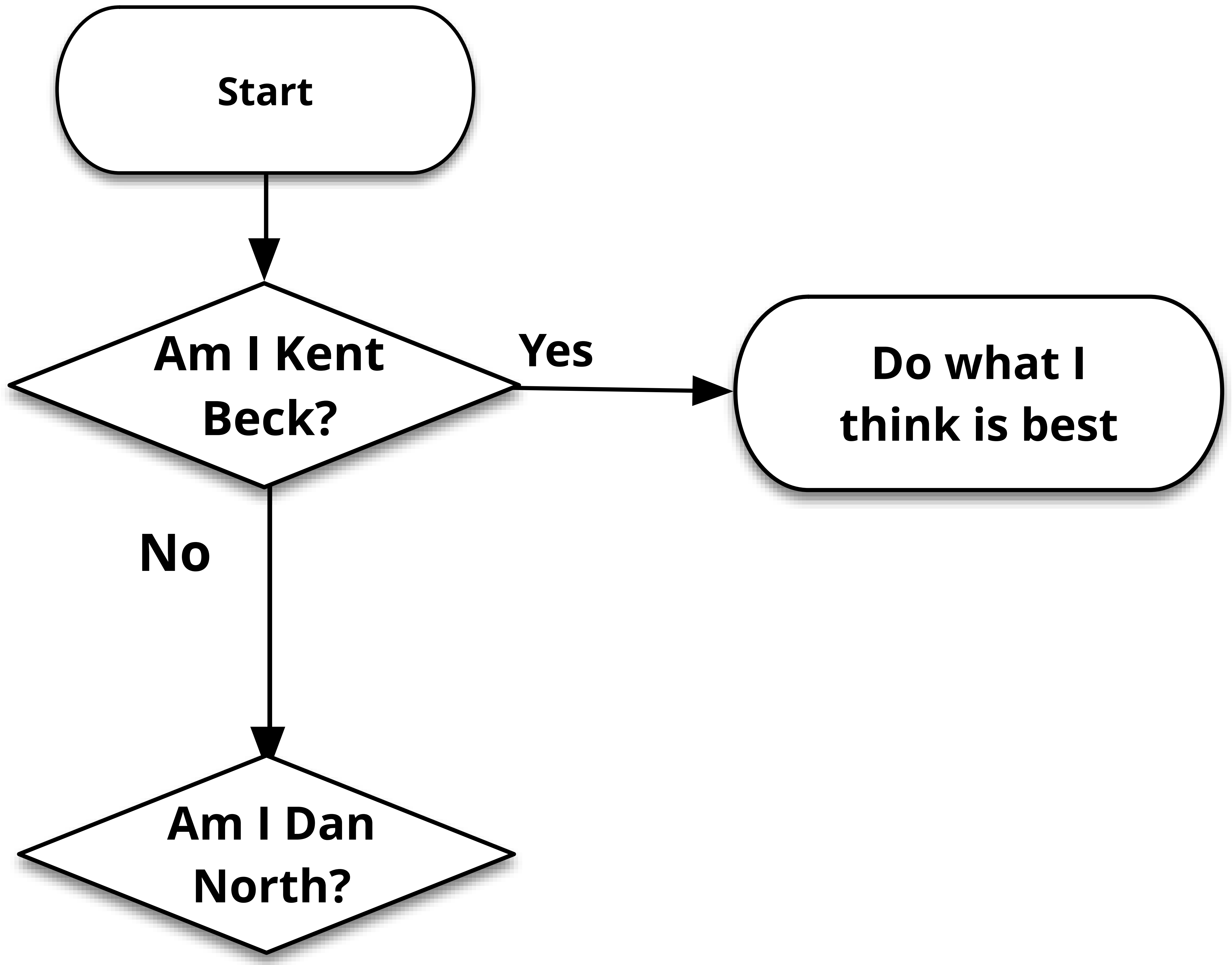
http://moleseyhill.com/blog/2009/08/27/dreyfus-model/

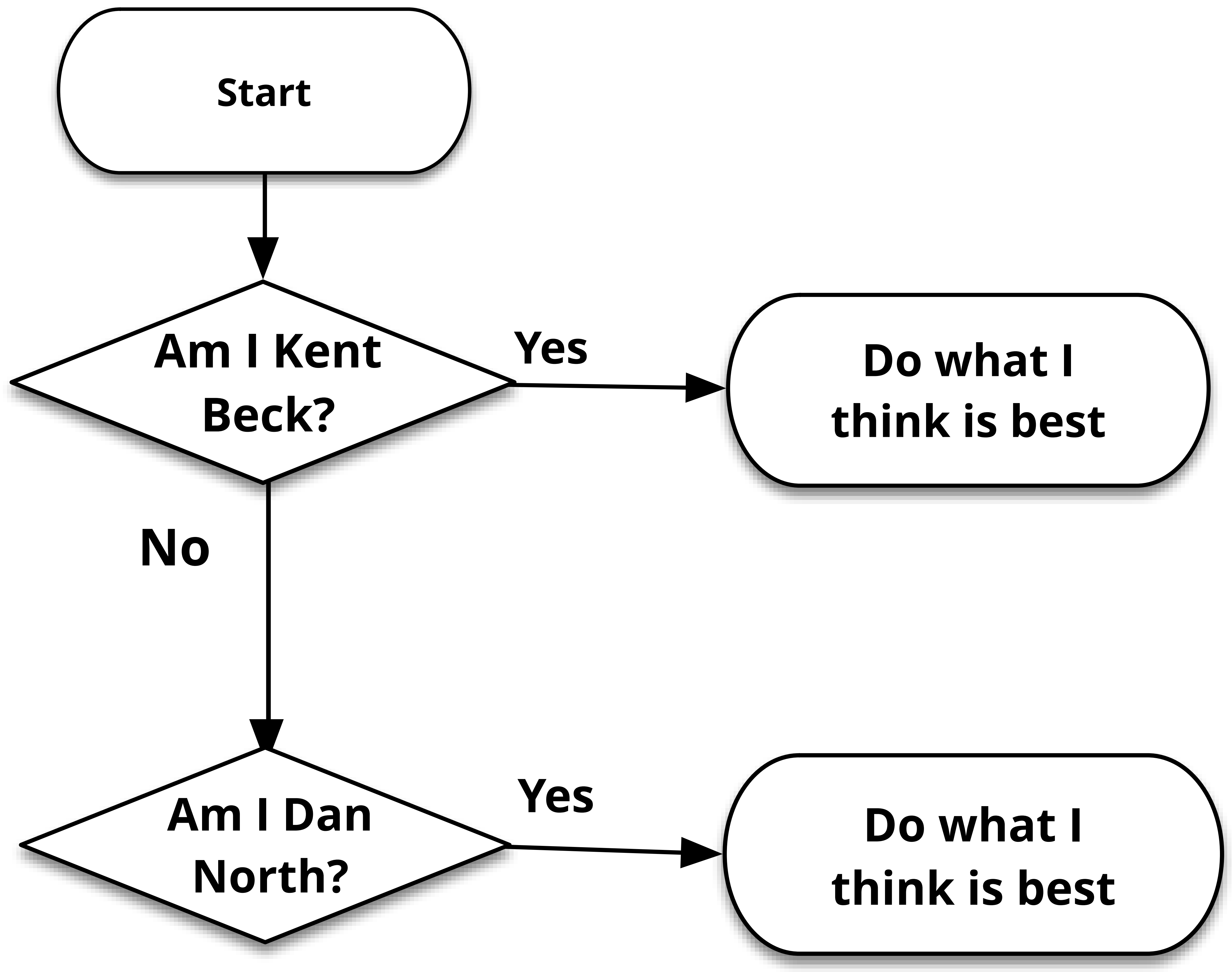
All rules are contextual

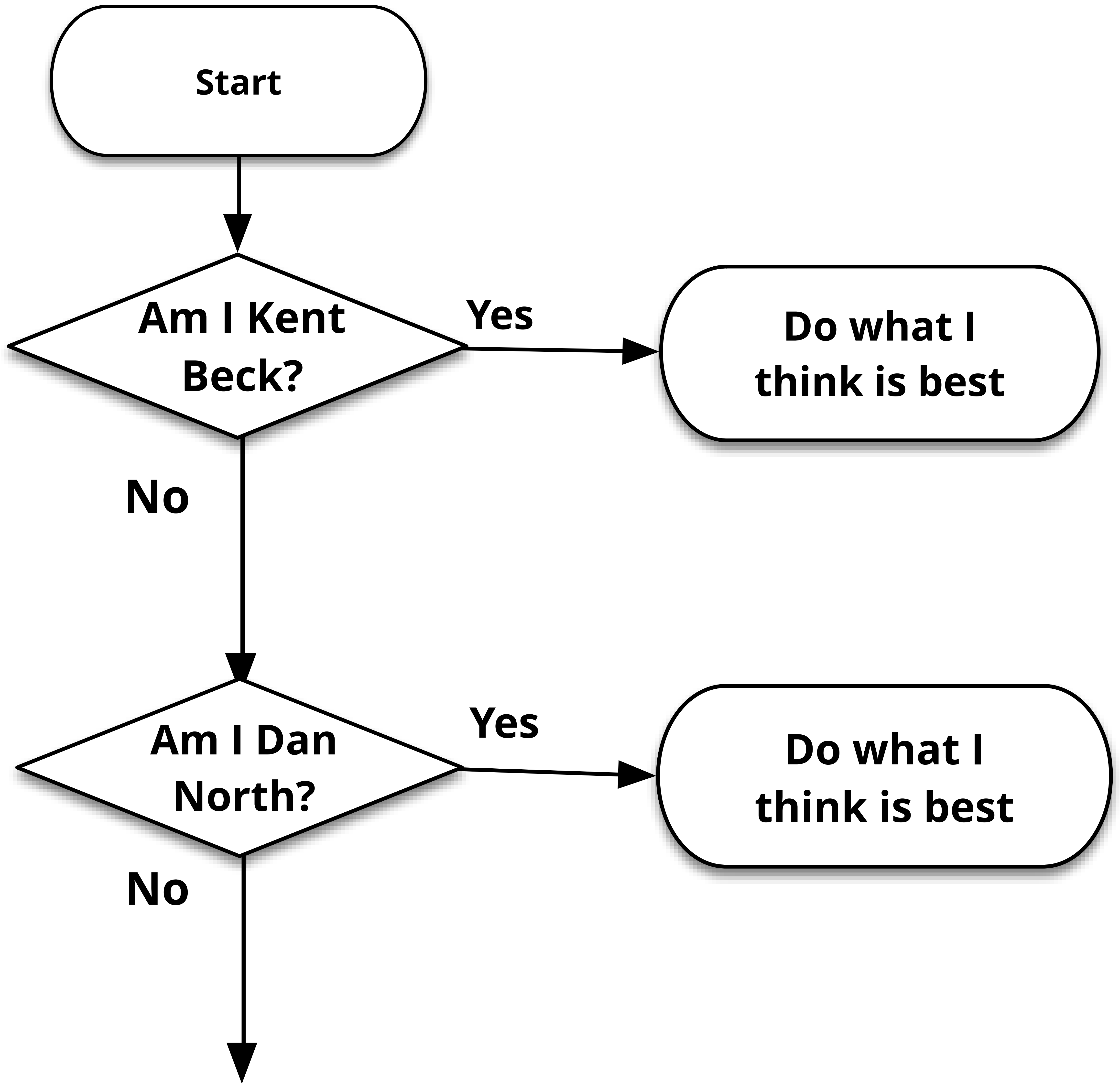

Start

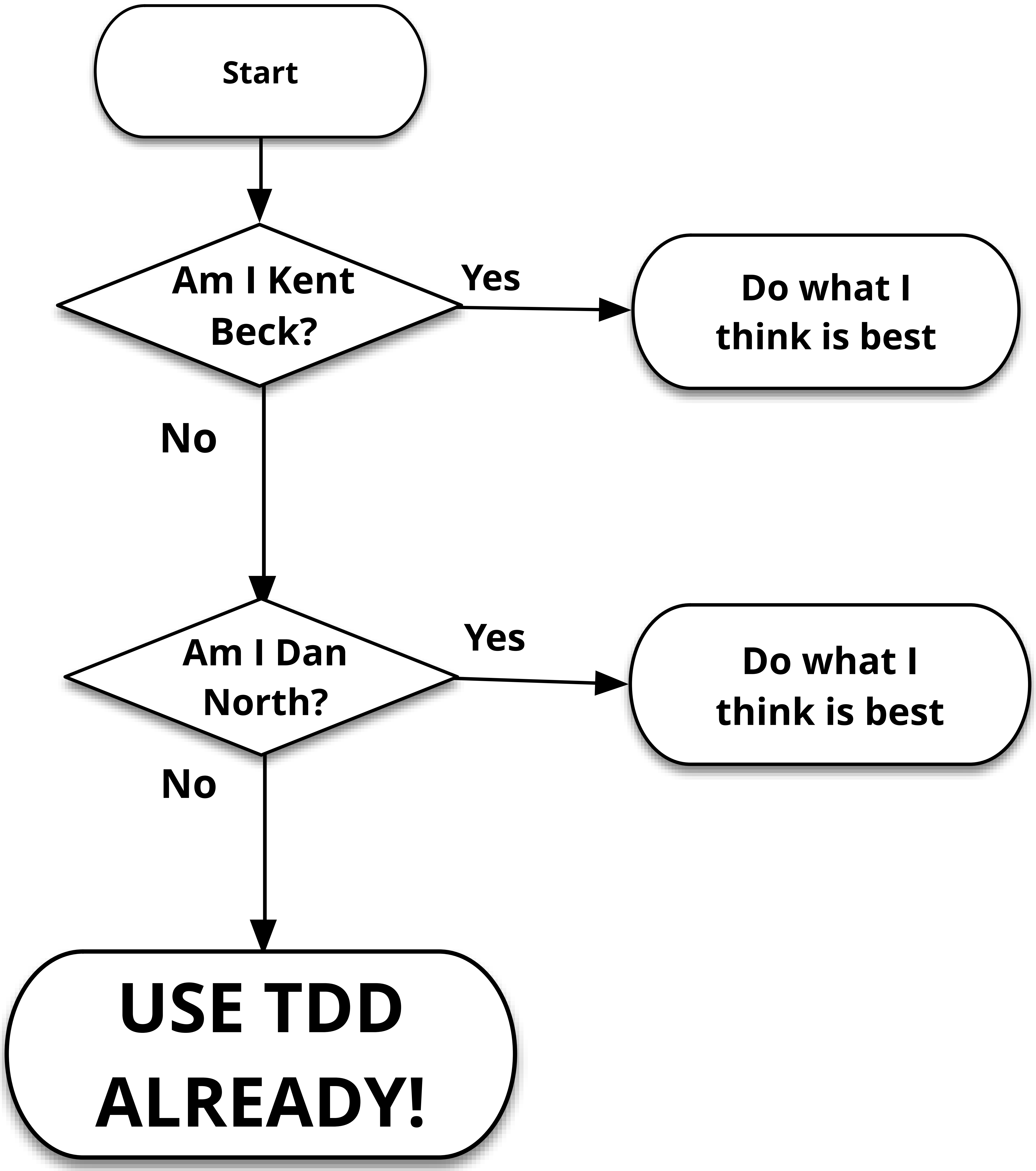

Start
Am I Kent Beck?


Start
Am I Kent Beck?
Yes
Do what I think is best


Start
Am I Kent Beck?
Yes
Do what I think is best
No
Am I Dan North?


Start
Am I Kent Beck?
Yes
Do what I think is best
No
Am I Dan North?
Yes
Do what I think is best


Start
Am I Kent Beck?
Yes
Do what I think is best
No
Am I Dan North?
Yes
Do what I think is best
No


Start
Am I Kent Beck?
Yes
Do what I think is best
No
Am I Dan North?
Yes
Do what I think is best
No
USE TDD ALREADY!

should we use TDD?

Yes. But.

GRASP

YAGNI

World of Warcraft

agile

DRY

SRP

BDD

nt des

GoF

Continuous Delivery

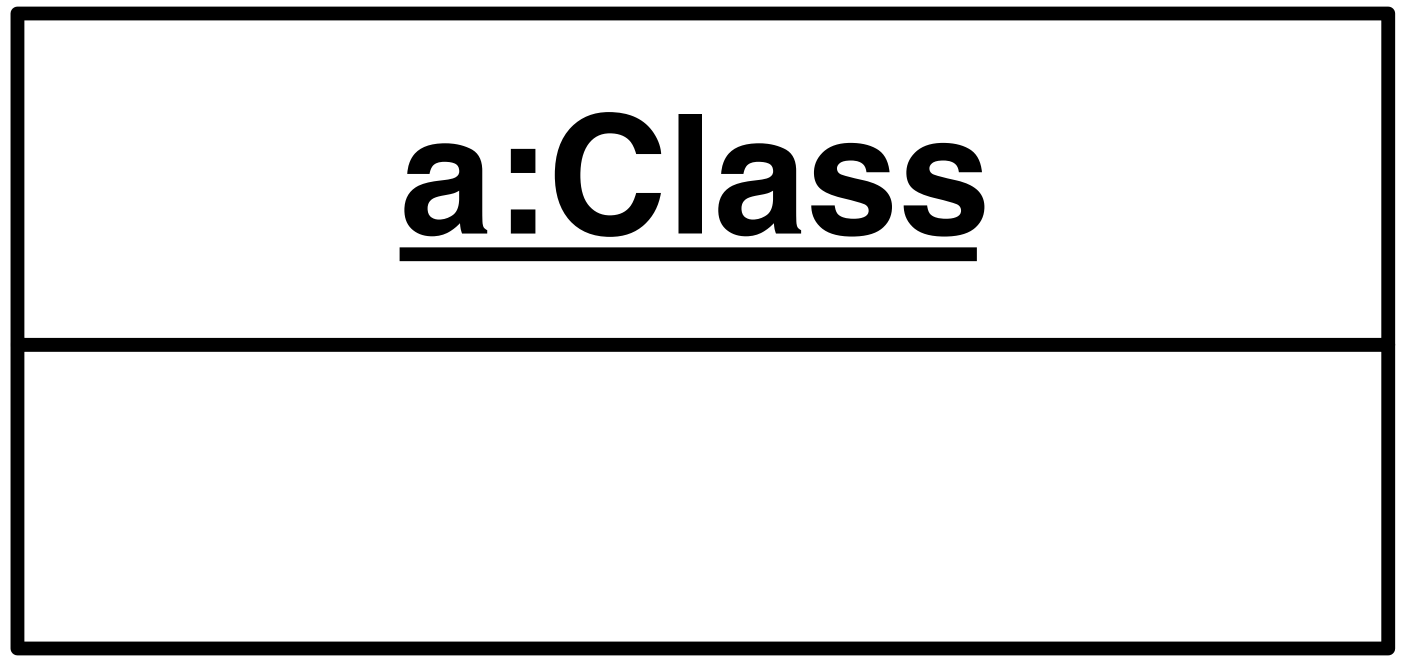
TDD

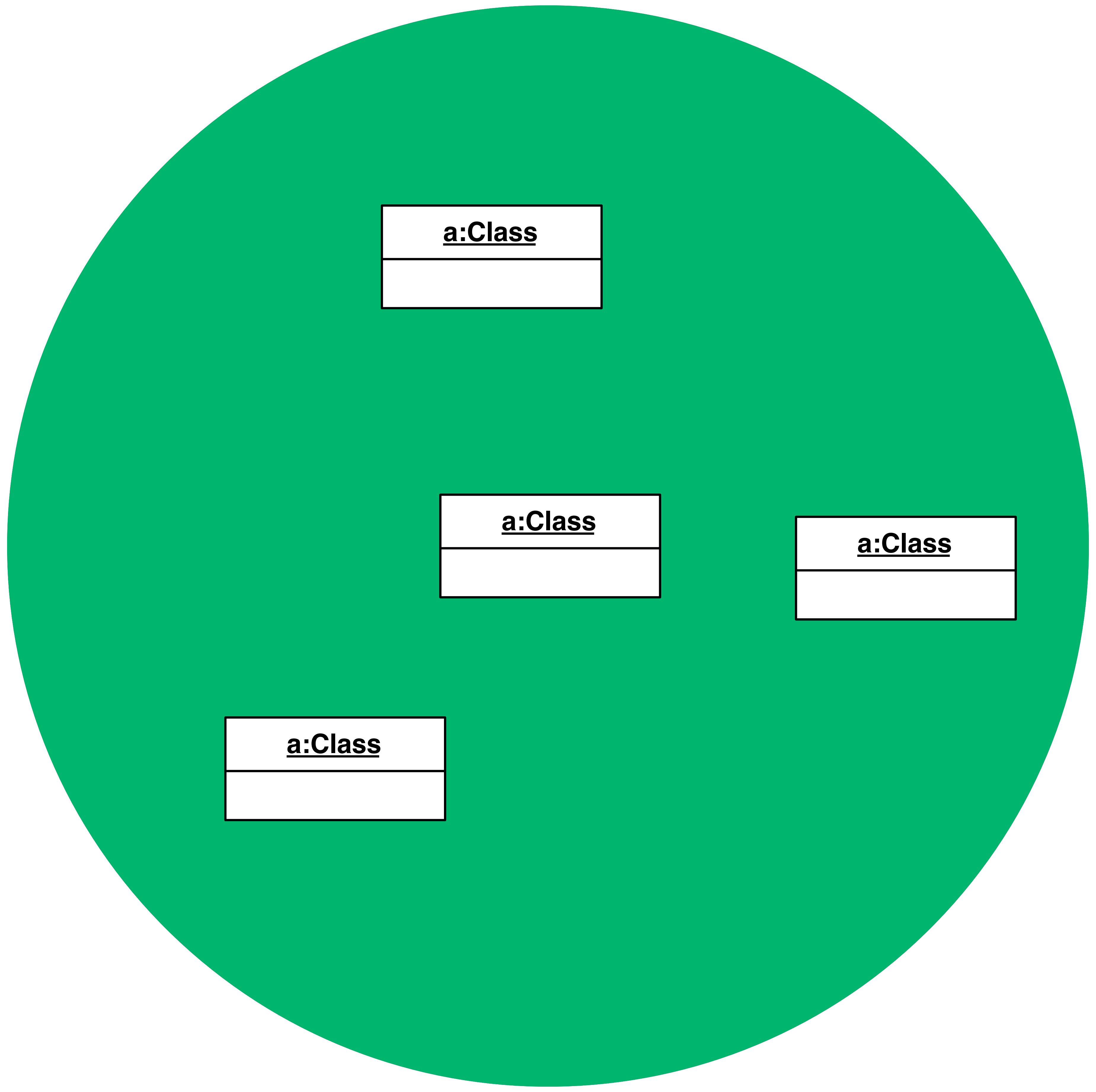
XP

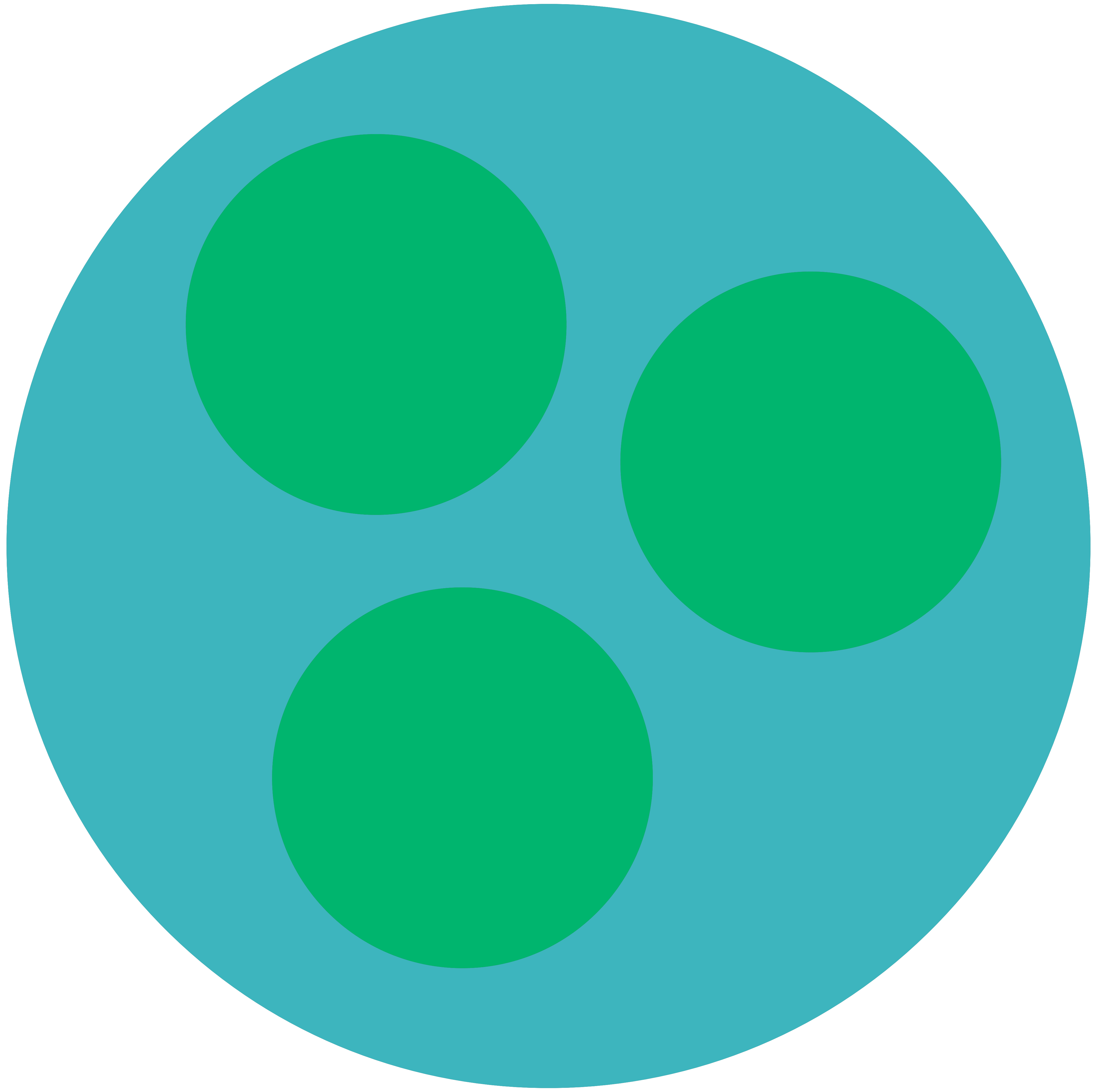
KISS

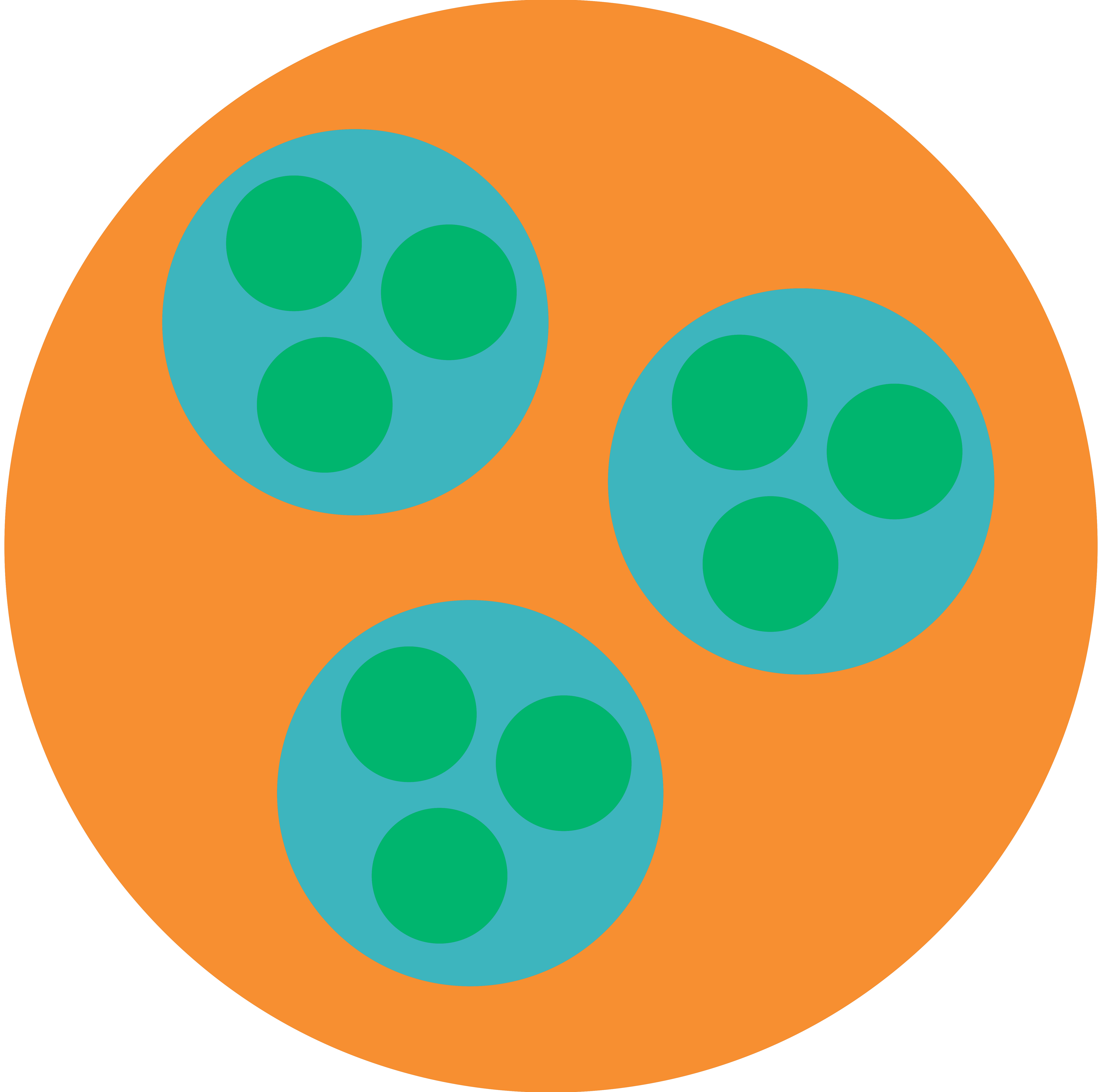
Refactoring

a class should be no bigger than my head


a:Class


a:Class
a:Class
a:Class
a:Class





SRP

a service should be no bigger than my head

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

KISS

BDD

merg esign

GoF

Continuous Delivery

TDD

XP

Refactoring

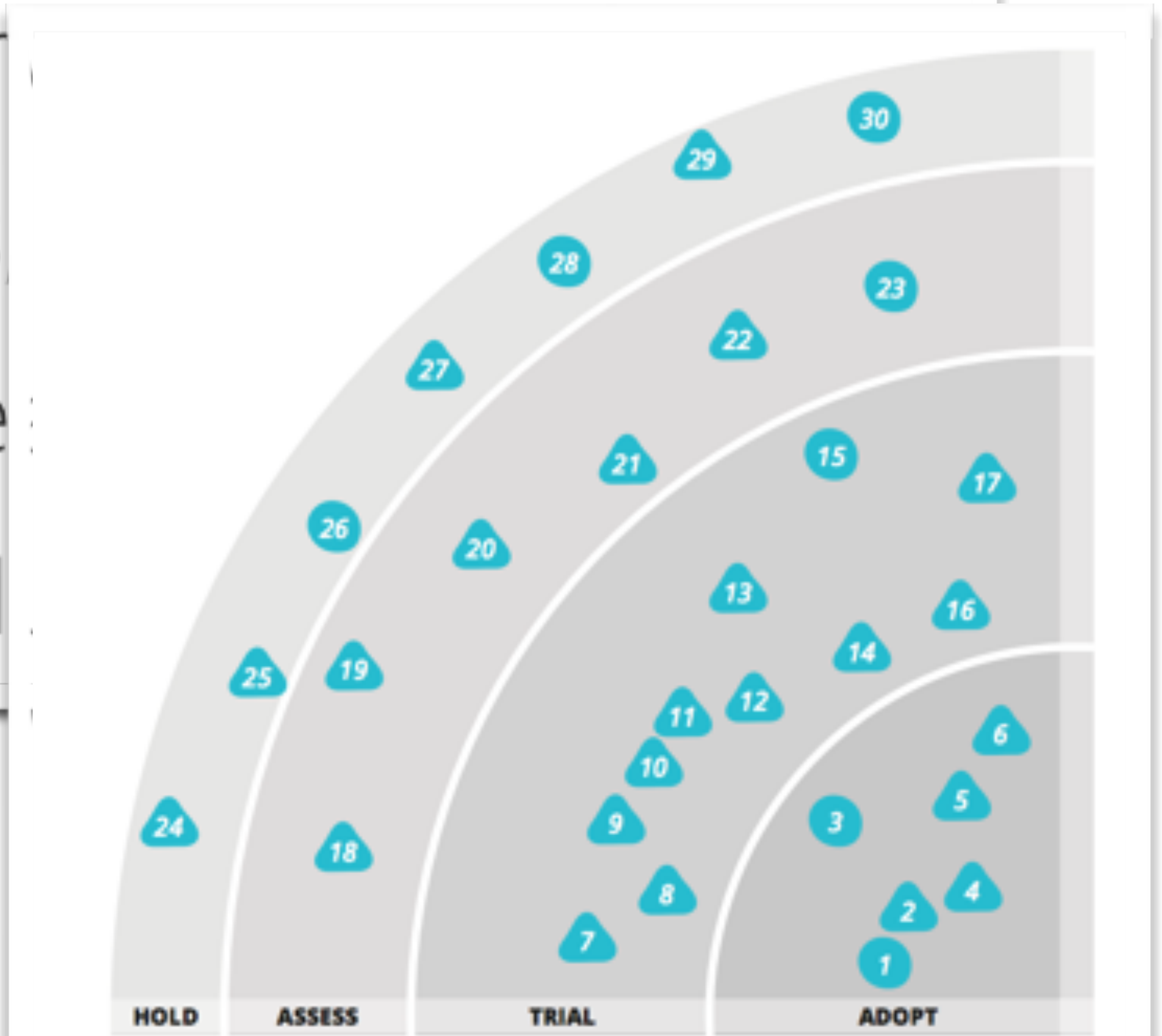
“Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.”

Edsger W. Dijkstra

● HOLD

45.Application Servers new

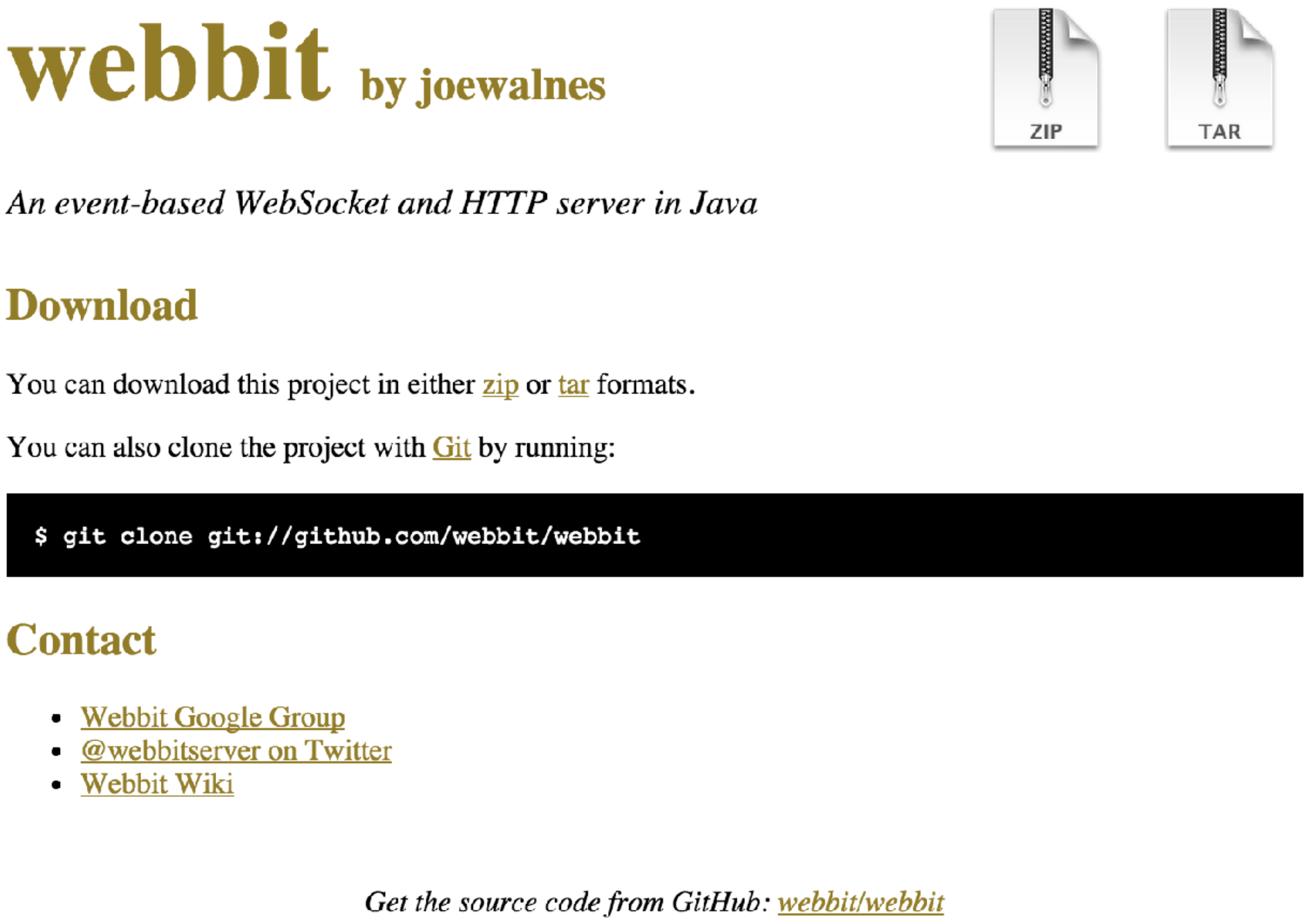
The rise of containers, phoenix servers and continuous delivery has seen a move away from the usual approach to deploying web applications. Traditionally we have built an artifact and then installed that artifact into an application ser
result was long feedback loops for changes,
build times and the not insignificant overhe
managing these application servers in prod



WWJD?

WWJD?

(what would Joe do?)



webbit by joewalnes

ZIP TAR

An event-based WebSocket and HTTP server in Java

Download

You can download this project in either zip or tar formats.

You can also clone the project with Git by running:

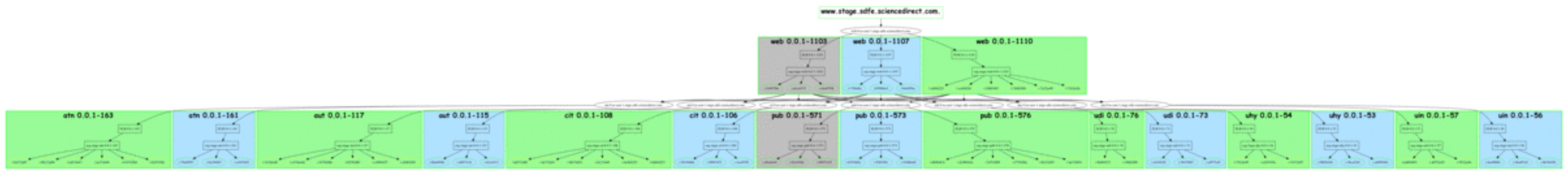
```
$ git clone git://github.com/webbit/webbit
```

Contact

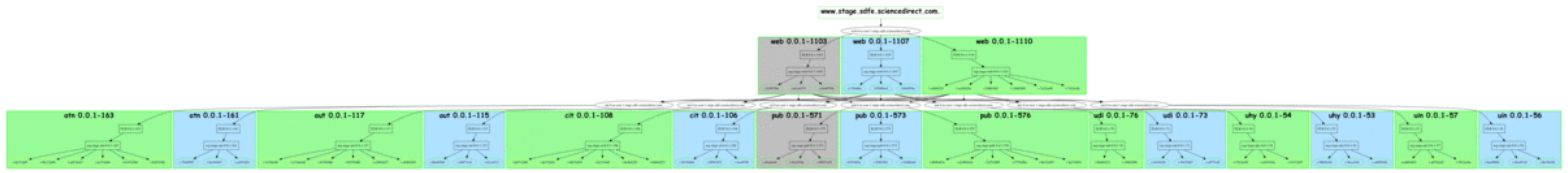
- Webbit Google Group
- @webbitserver on Twitter
- Webbit Wiki

Get the source code from GitHub: webbit/webbit

cron, python, boto, pydot, graphviz



cron, python, boto, pydot, graphviz



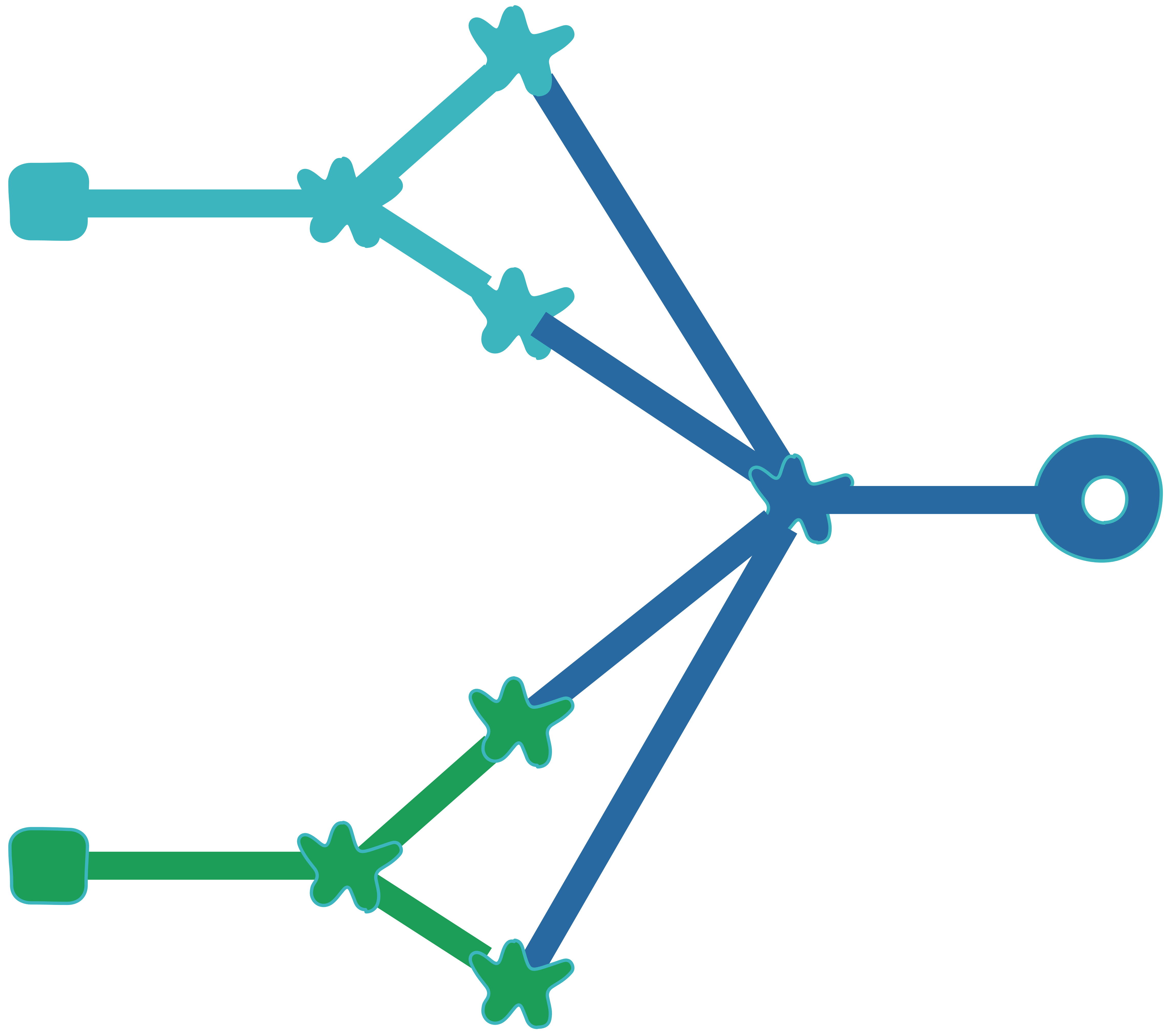
cron, python, boto, pydot, graphviz

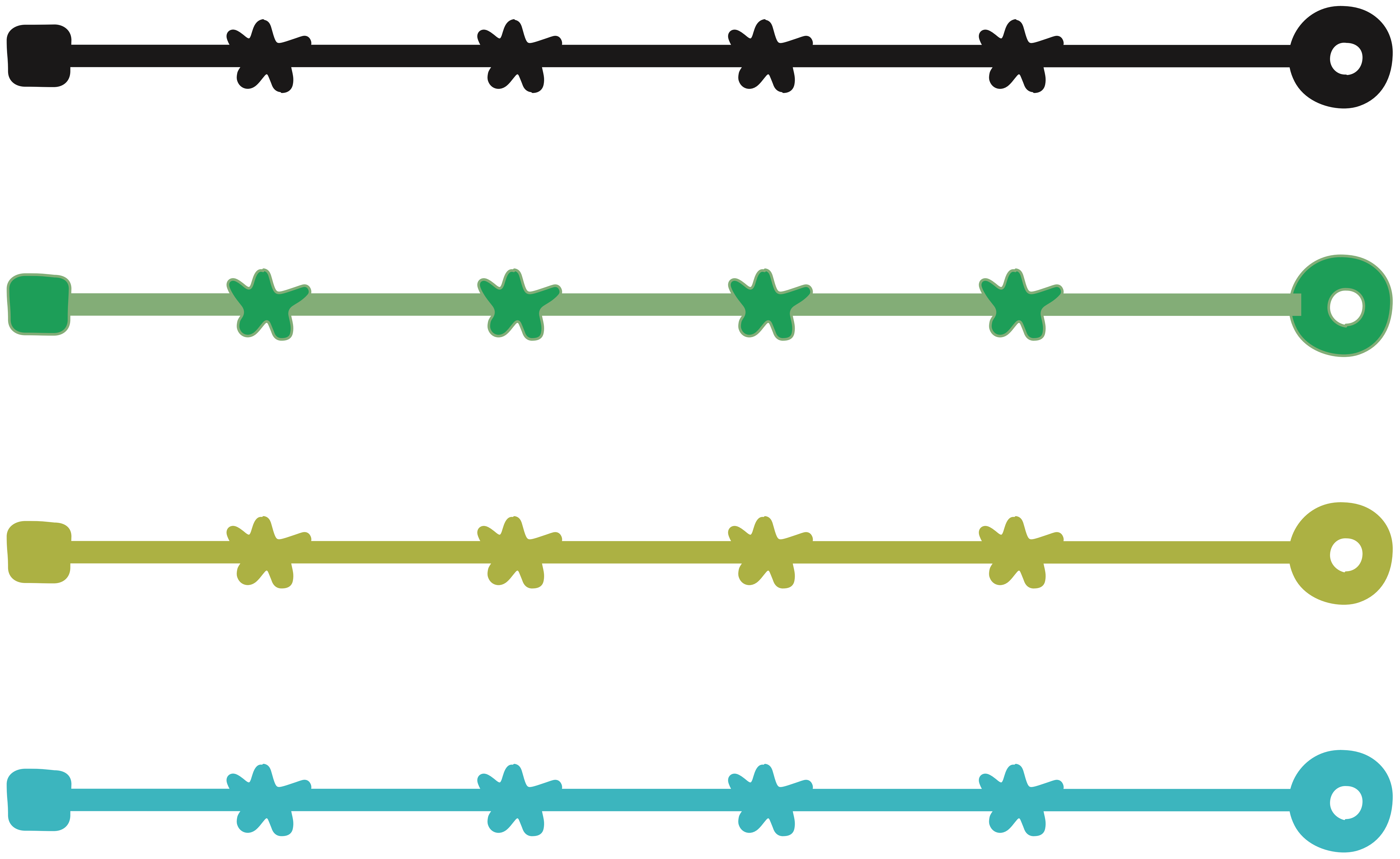
Do the simplest thing possible

What about our integration testing practices?

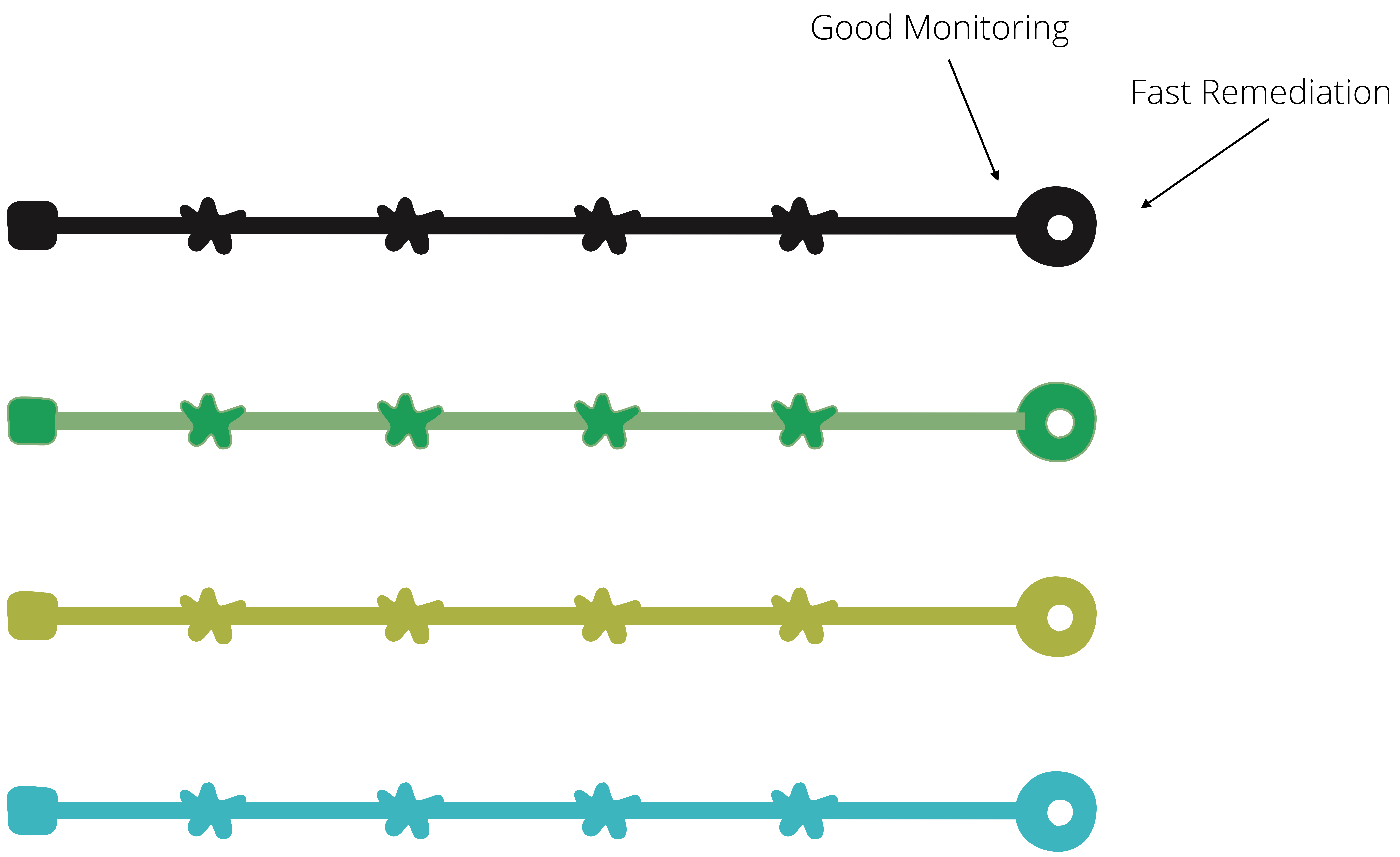
“There is nothing so **useless** as doing **efficiently** that which should **not be done** at all”

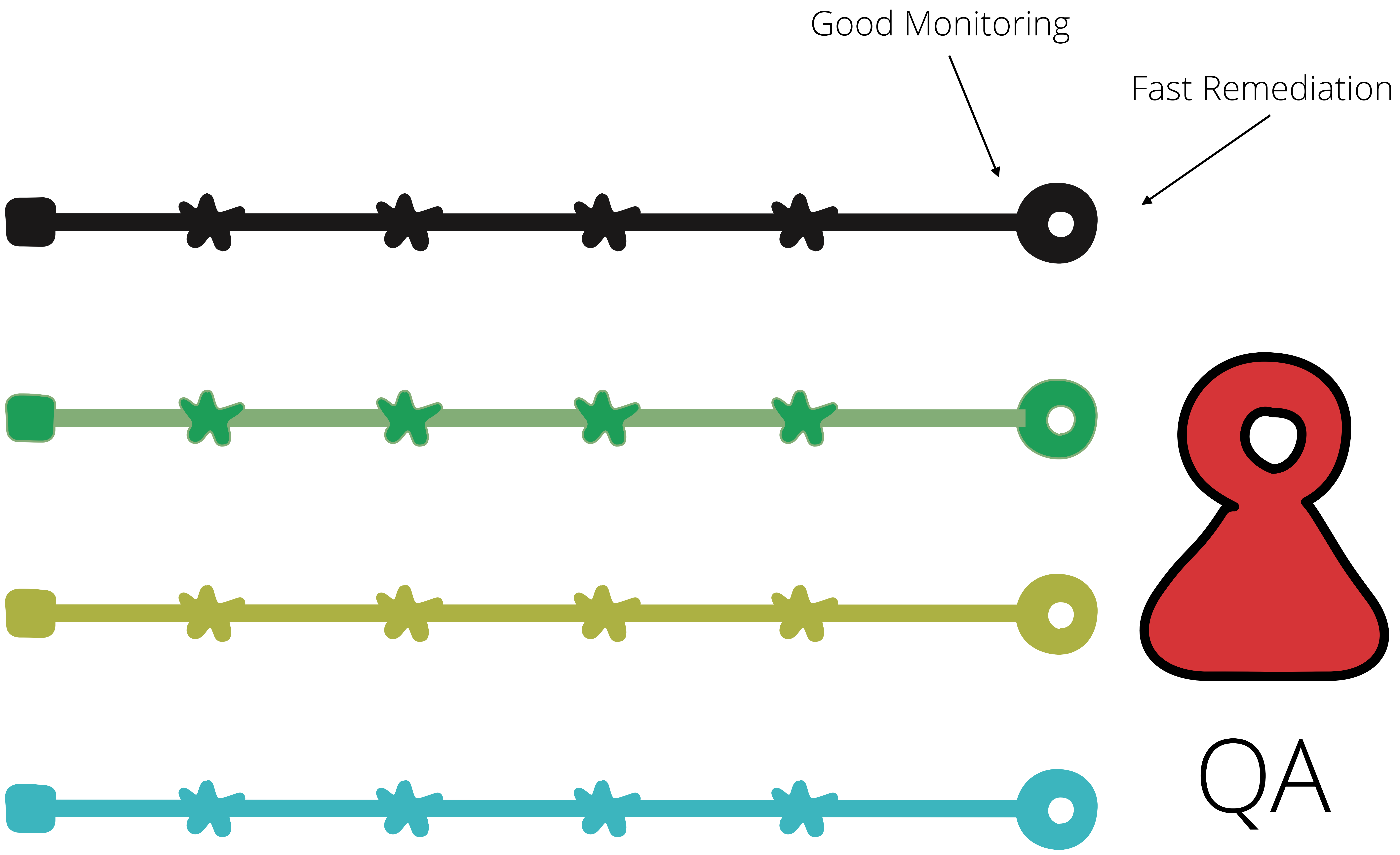
Peter Drucker

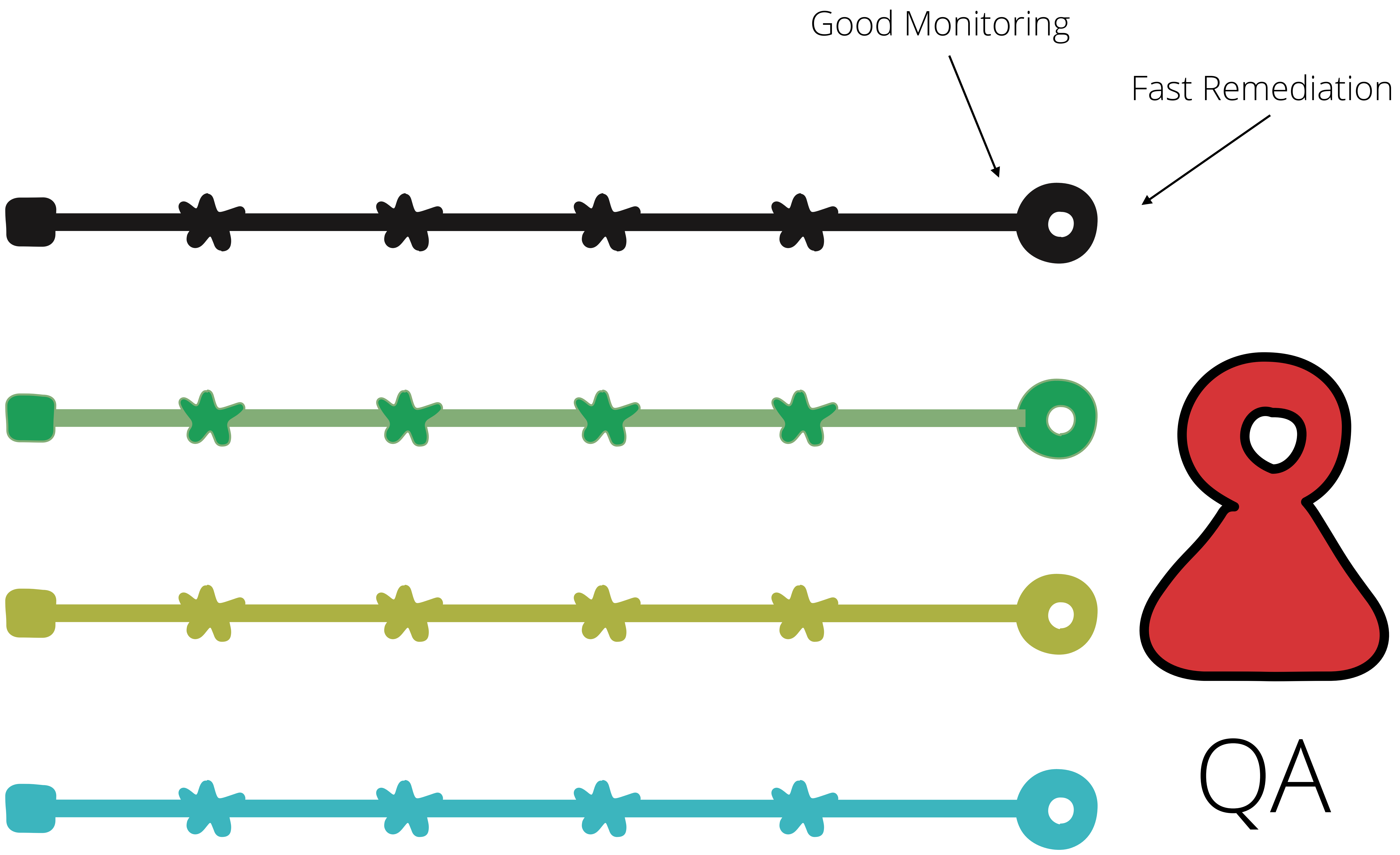




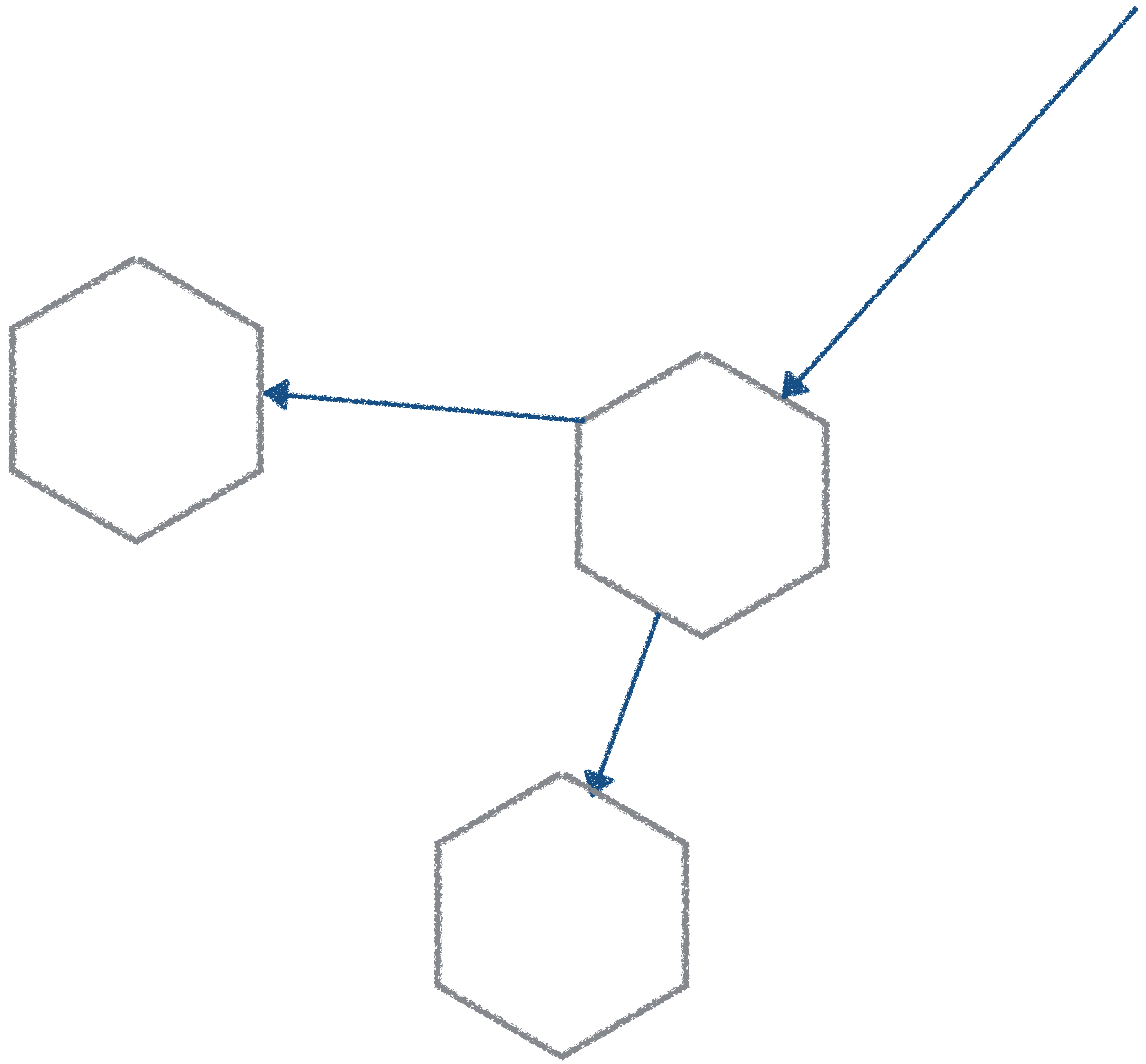
Good Monitoring

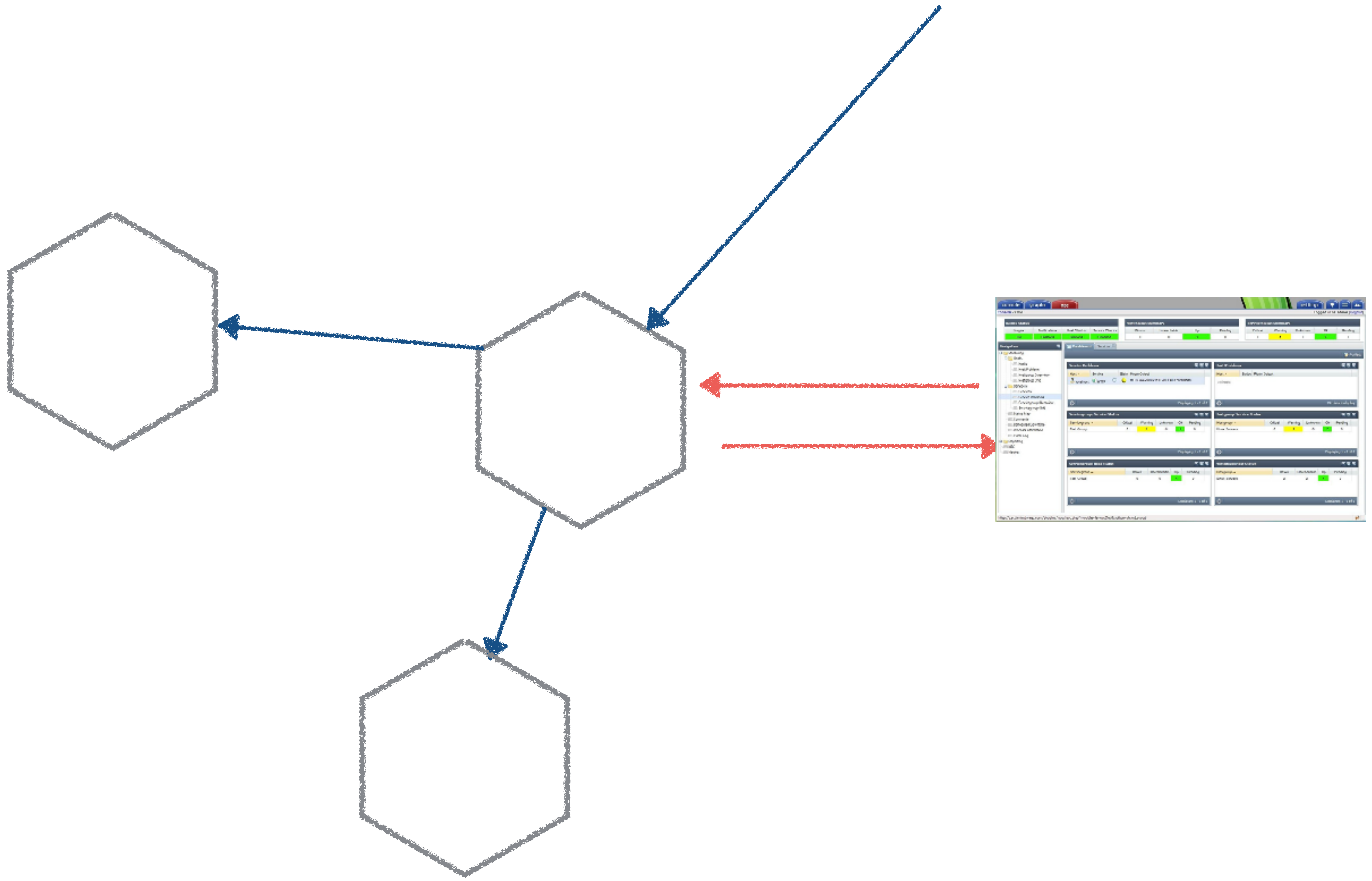

Good Monitoring
Fast Remediation


Good Monitoring
Fast Remediation
QA

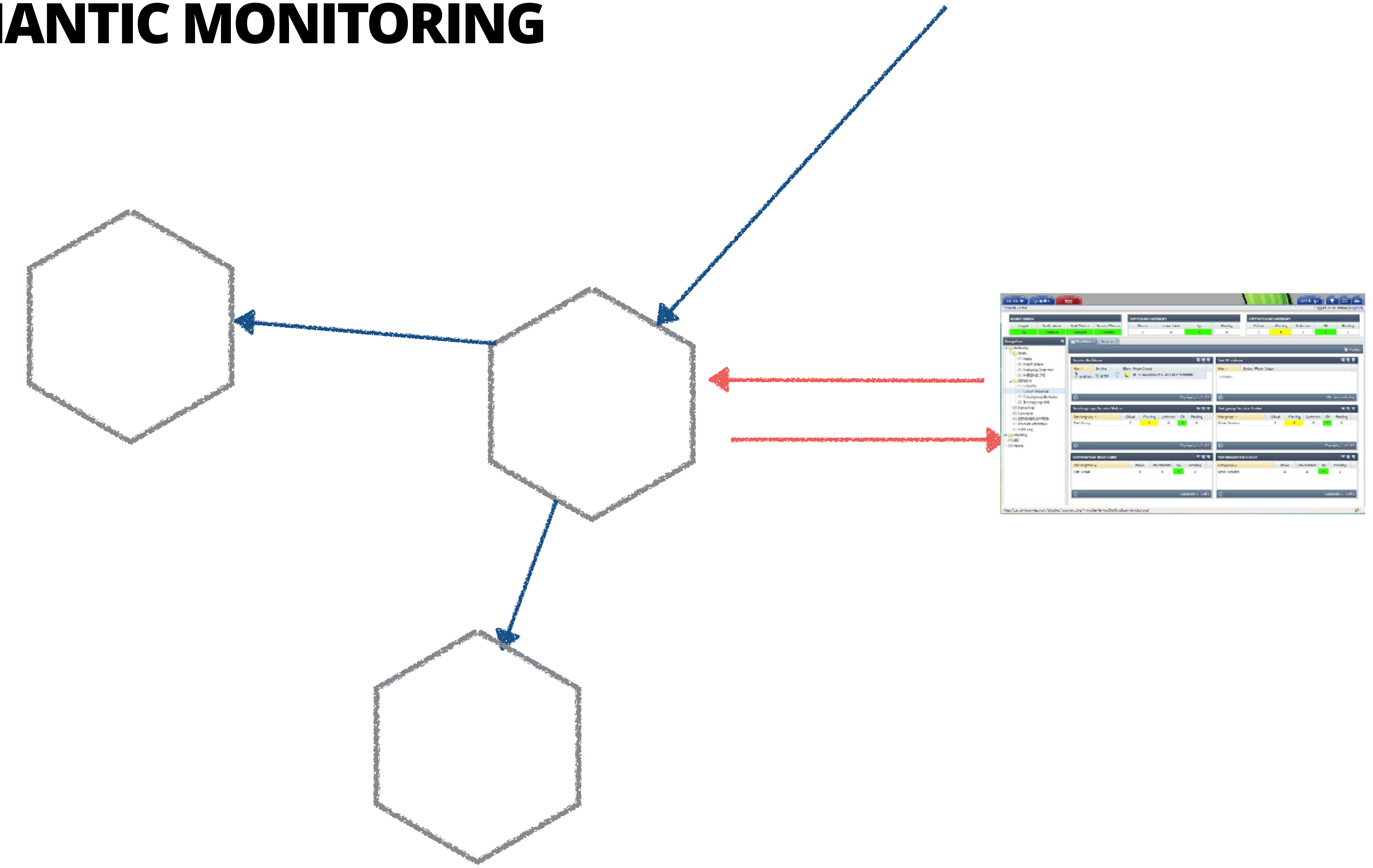


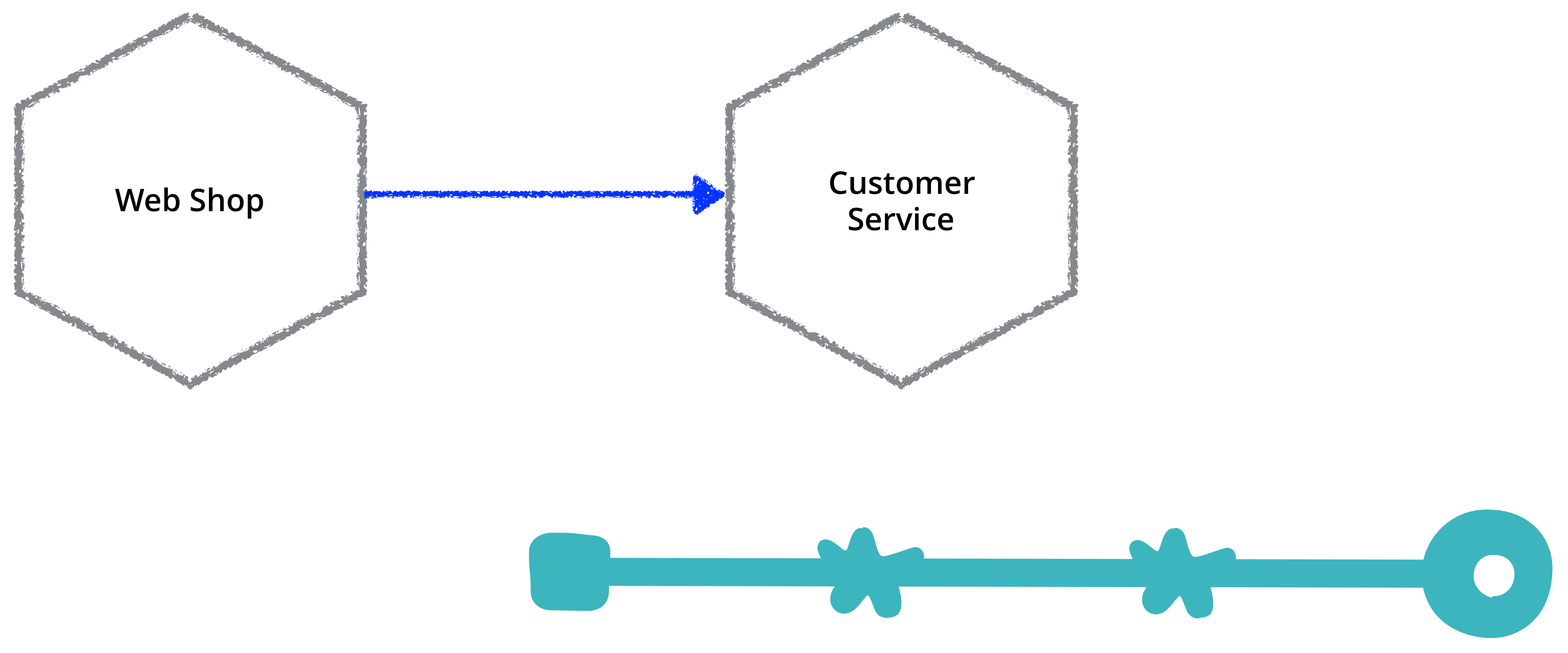
Test in production

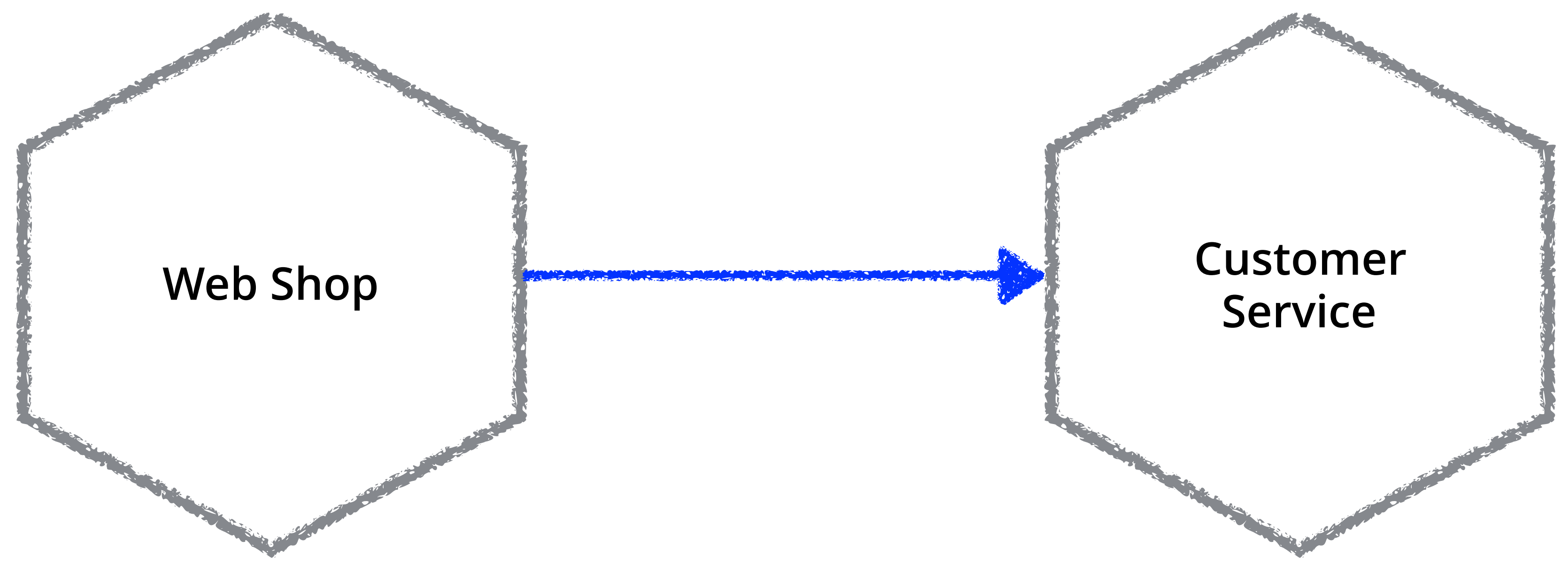




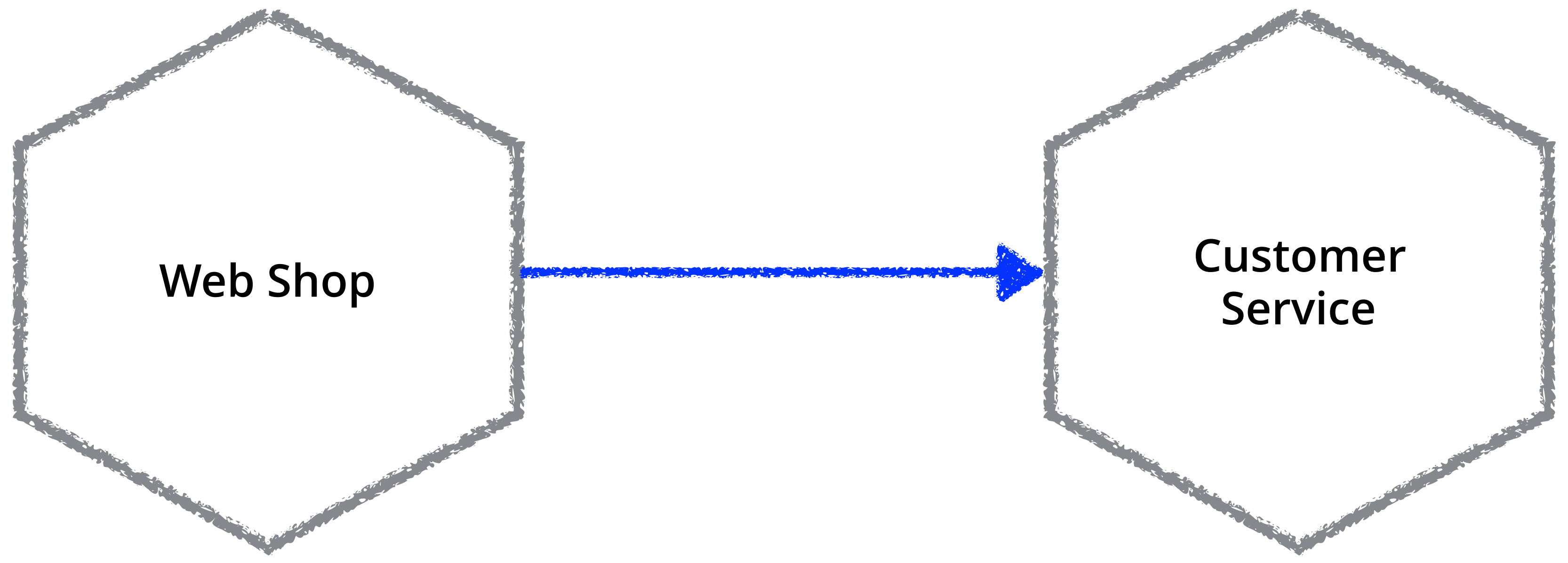
SEMANTIC MONITORING



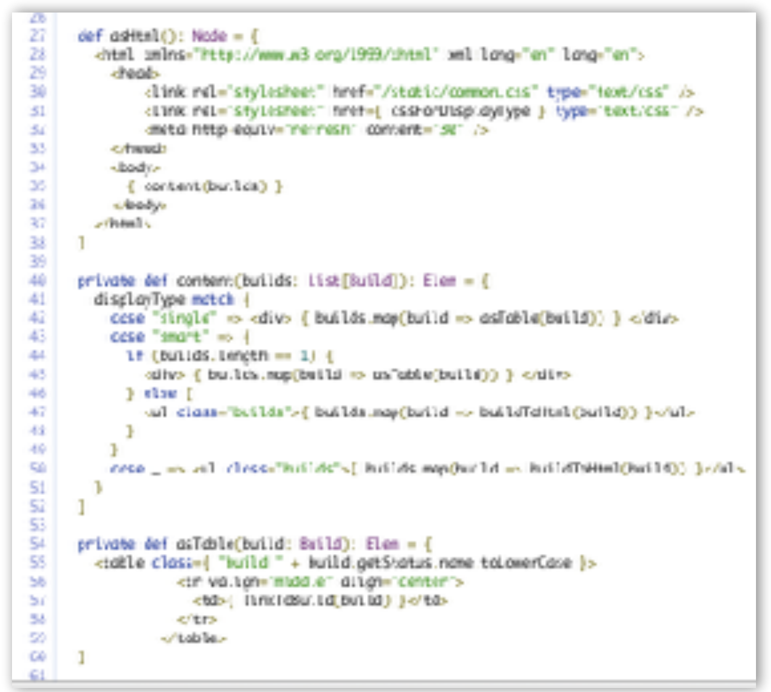

Web Shop
Customer Service



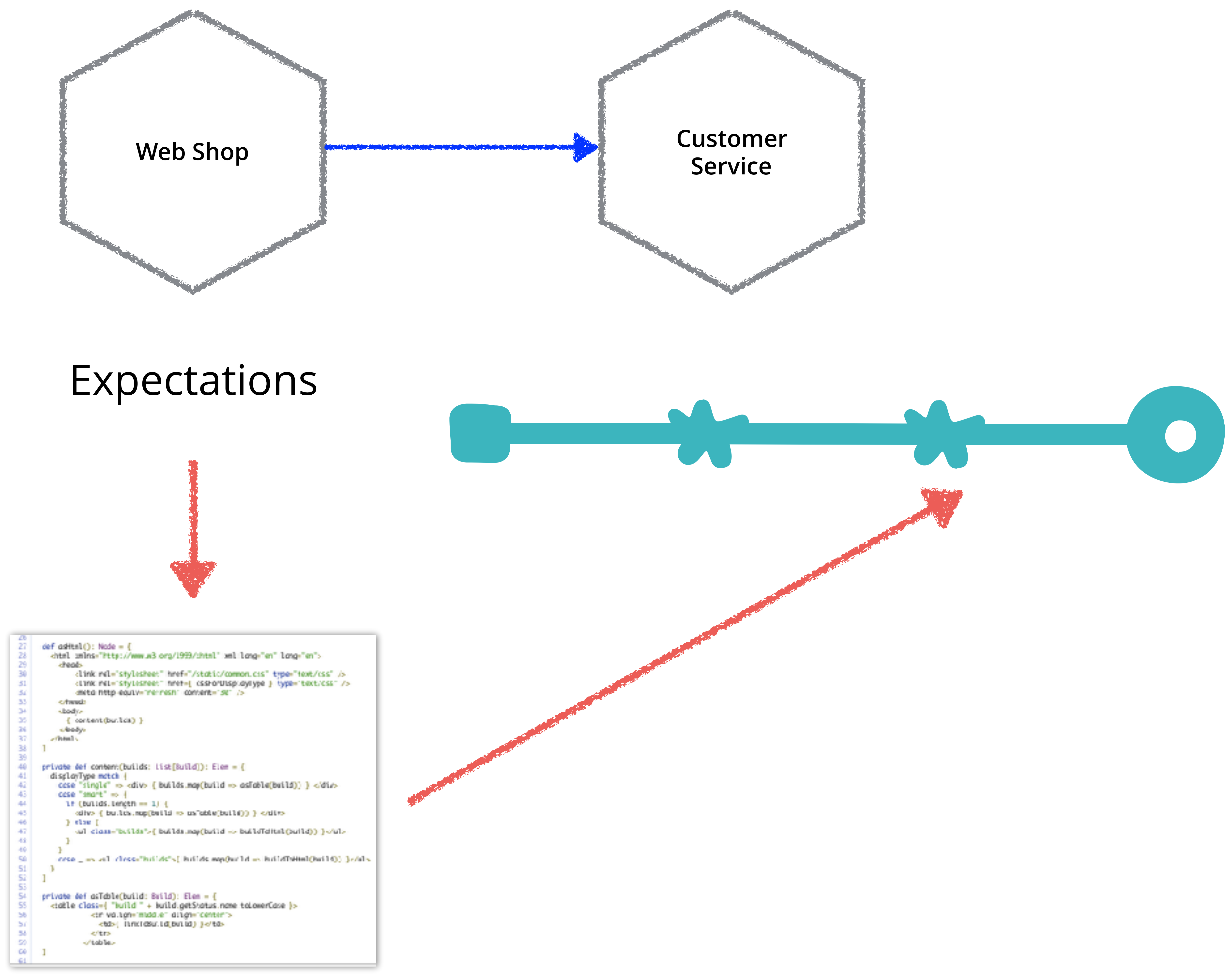
Expectations



Expectations

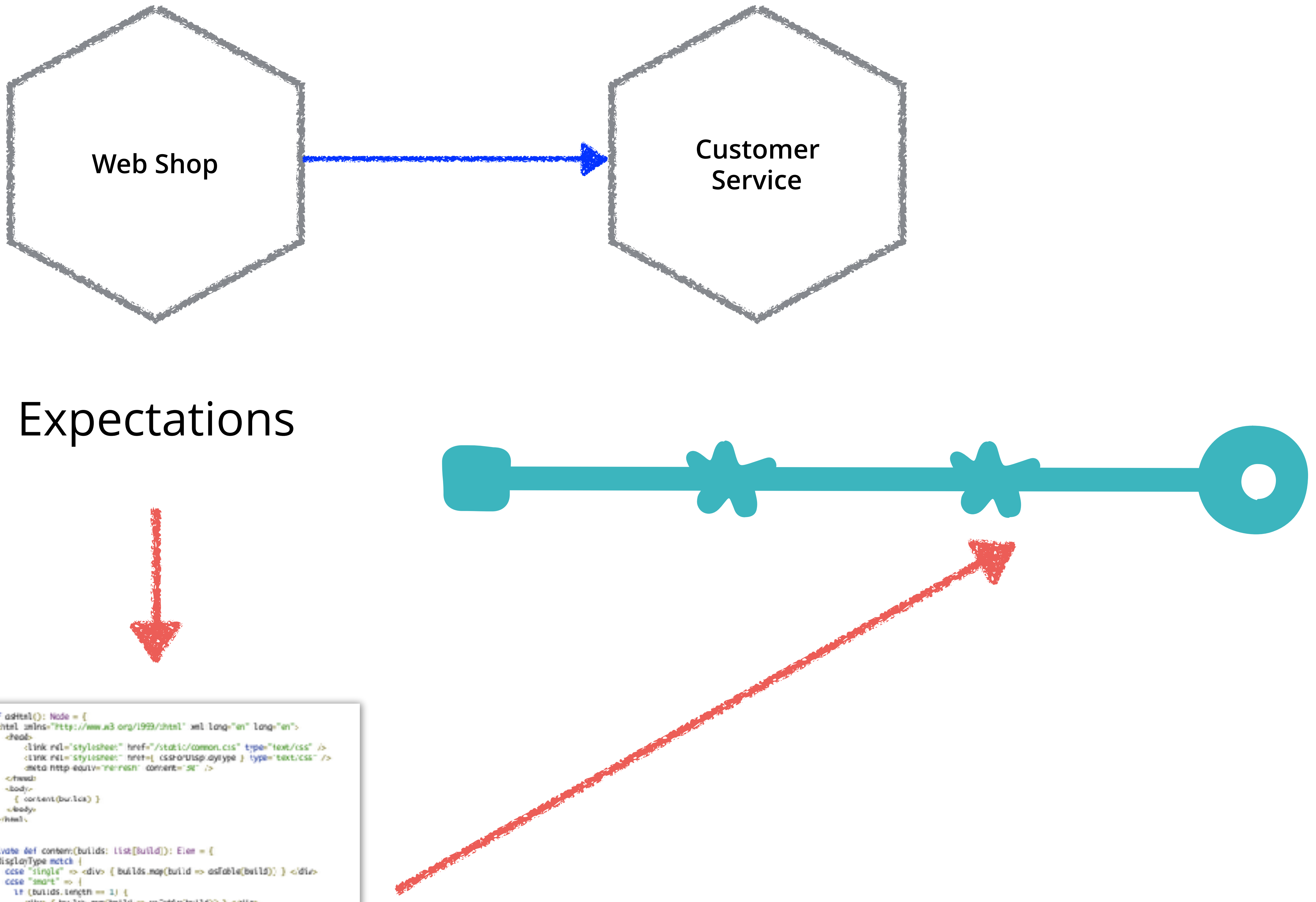


```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name toLowerCase }>
            <tr valign="middle" align="center">
              <td>{ link(build.id, build) }</td>
            </tr>
          </table>
}
```



```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
            <tr valign="middle" align="center">
              <td>{ link(build.id, build) }</td>
            </tr>
          </table>
}
```

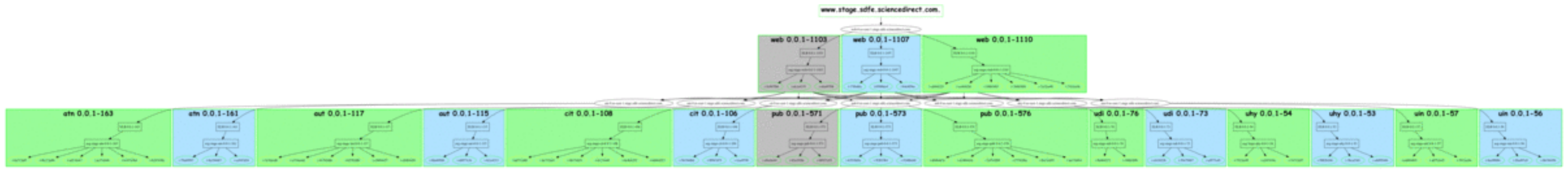
Consumer Driven Contracts

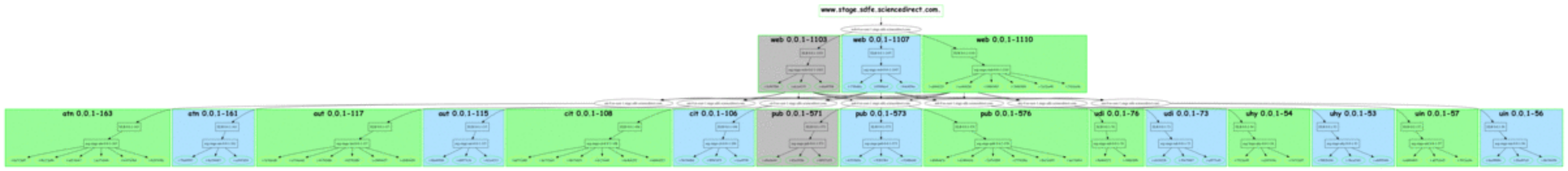


```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name toLowerCase }>
          <tr valign="middle" align="center">
            <td>{ link(build.id, build) }</td>
          </tr>
        </table>
}
```

the death of the integration environment

production != live


www.stage.sdfe.sciencedirect.com.
web 0.0.1-1103
web 0.0.1-1107
web 0.0.1-1110
atn 0.0.1-163
atn 0.0.1-161
aut 0.0.1-117
aut 0.0.1-115
cit 0.0.1-108
cit 0.0.1-106
pub 0.0.1-571
pub 0.0.1-573
pub 0.0.1-576
udi 0.0.1-76
udi 0.0.1-73
uhy 0.0.1-54
uhy 0.0.1-53
uin 0.0.1-57
uin 0.0.1-56


www.stage.sdfe.sciencedirect.com.
web 0.0.1-1103
web 0.0.1-1107
web 0.0.1-1110
atn 0.0.1-163
atn 0.0.1-161
aut 0.0.1-117
aut 0.0.1-115
cit 0.0.1-108
cit 0.0.1-106
pub 0.0.1-571
pub 0.0.1-573
pub 0.0.1-576
udi 0.0.1-76
udi 0.0.1-73
uhy 0.0.1-54
uhy 0.0.1-53
uin 0.0.1-57
uin 0.0.1-56

Part the Eighth

The hunting of the snark!

"Leave him here to his fate—it is getting so late!"
The Bellman exclaimed in a fright.
"We have lost half the day. Any further delay,
And we sha'n't catch a Snark before night!"

characteristics of microservices

componentisation via services

organised around business capabilities

decentralised data management

products not projects

decentralised governance

smart endpoints and dumb pipes

evolutionary design

infrastructure automation

designed for failure

http://martinfowler.com/articles/microservices.html

It turns out, it’s not all about componentisation via services

“…organizations which design systems … are constrained to produce designs which are copies of the communication structure of those organizations”

Melvyn Conway, 1968

The mirroring phenomenon is consistent with two rival causal mechanisms. First, designs may evolve to reflect their development environments. In **tightly-coupled organizations**, dedicated teams employed by a single firm and located at a single site develop the design. Problems are solved by face-to-face interaction, and performance “tweaked” by taking advantage of the access that module developers have to information and solutions developed in other modules. **Even if not an explicit managerial choice, the design naturally becomes more tightly-coupled**.

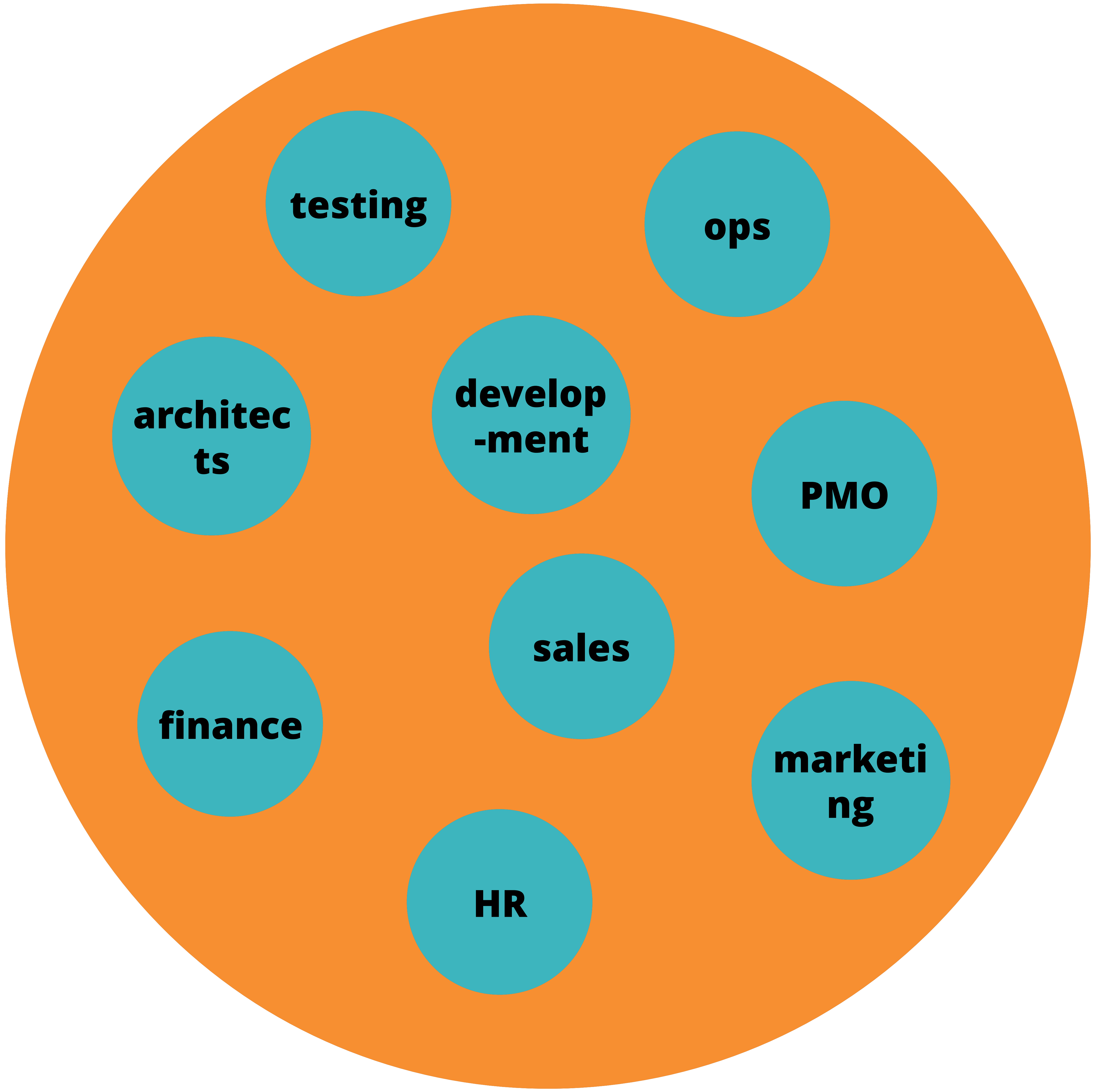
By contrast, in **loosely-coupled organizations**, a large, distributed team of volunteers develops the design. Face-to-face communications are rare given most developers never meet. Hence fewer connections between modules are established. **The architecture that evolves is more modular** as a result of the limitations on communication between developers.

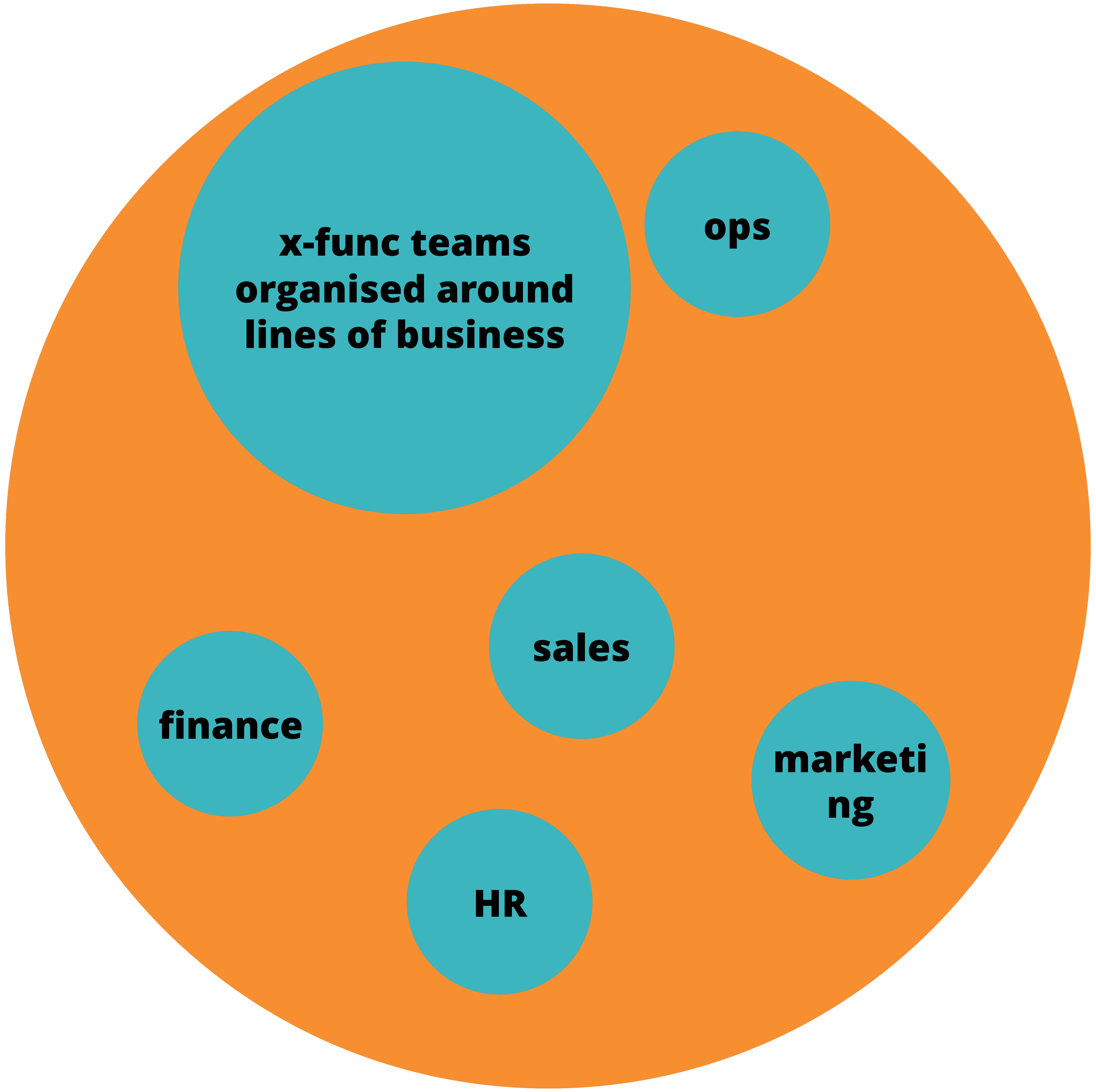
"Exploring the Duality between Product and Organizational Architectures : A Test of the “Mirroring” Hypothesis"

http://www.hbs.edu/faculty/Publication%20Files/08-039_1861e507-1dc1-4602-85b8-90d71559d85b.pdf

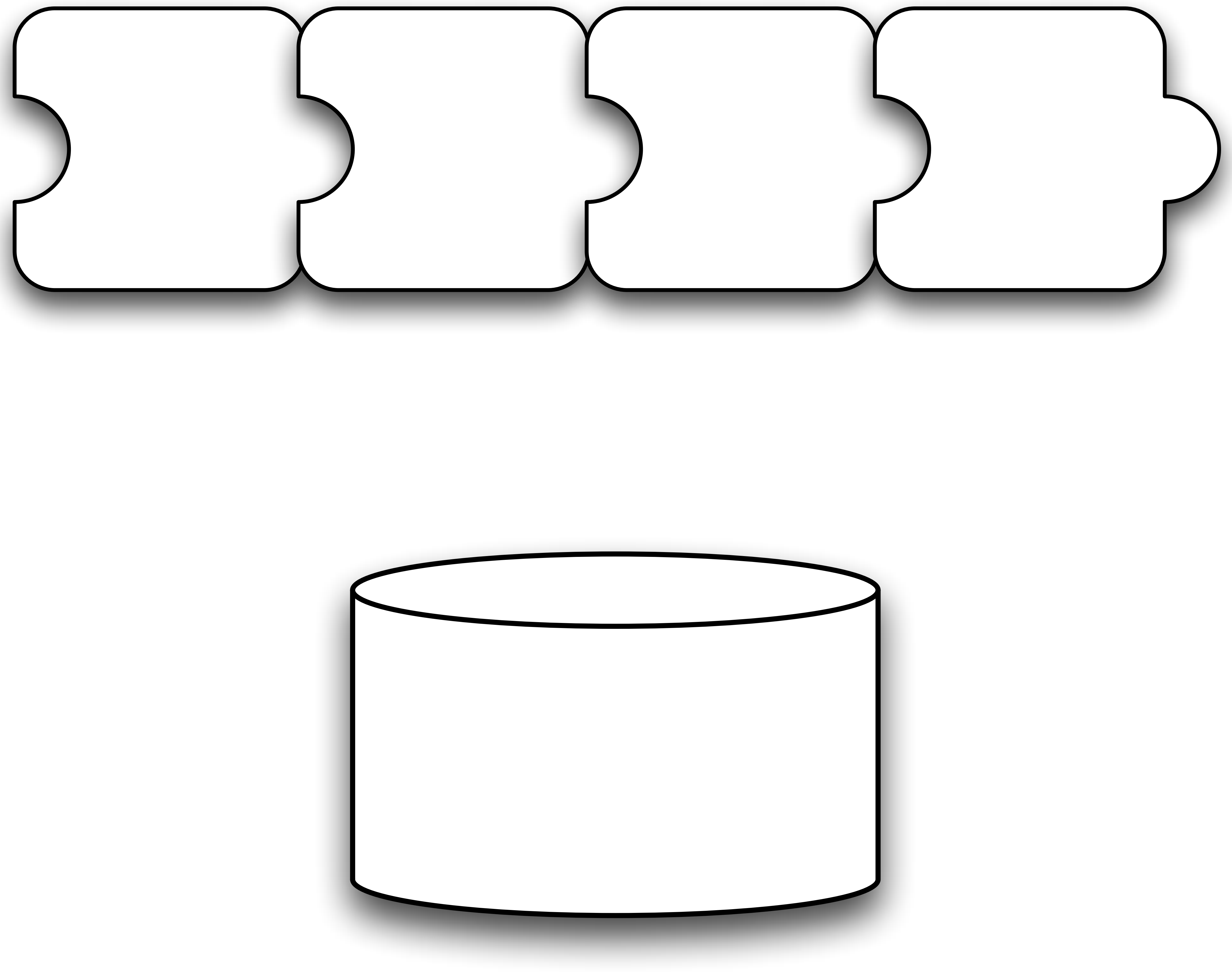
tightly-coupled **organizations ⇒ the design becomes more** ***tightly-coupled***.

loosely-coupled **organizations ⇒ the architecture is more** ***modular***

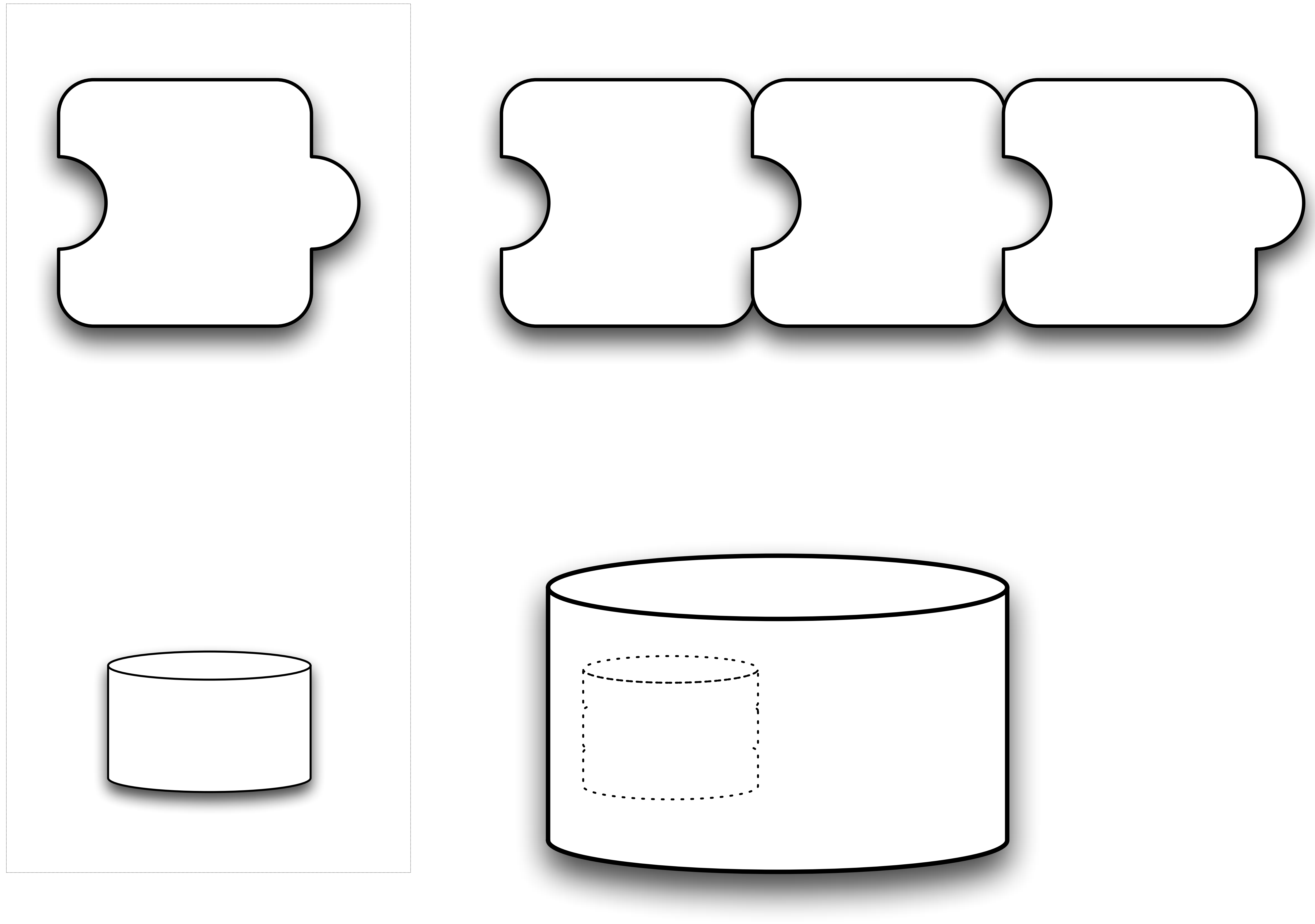

testing
ops
architec
ts
develop
-ment
PMO
sales
finance
marketi
ng
HR


x-func teams organised around lines of business
ops
sales
finance
marketi ng
HR

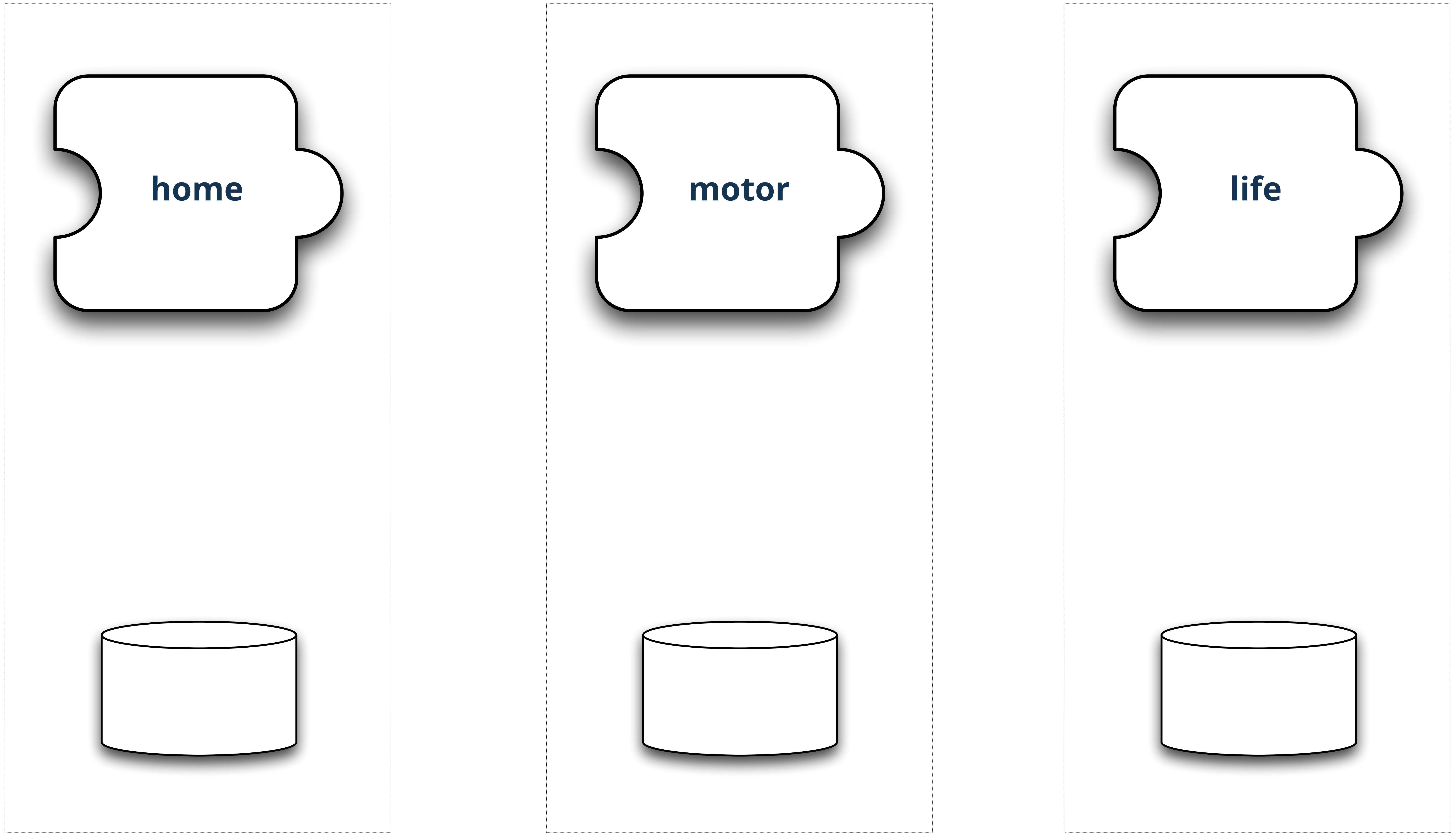
insurance company



insurance company



separate lines of business



separate lines of business

home

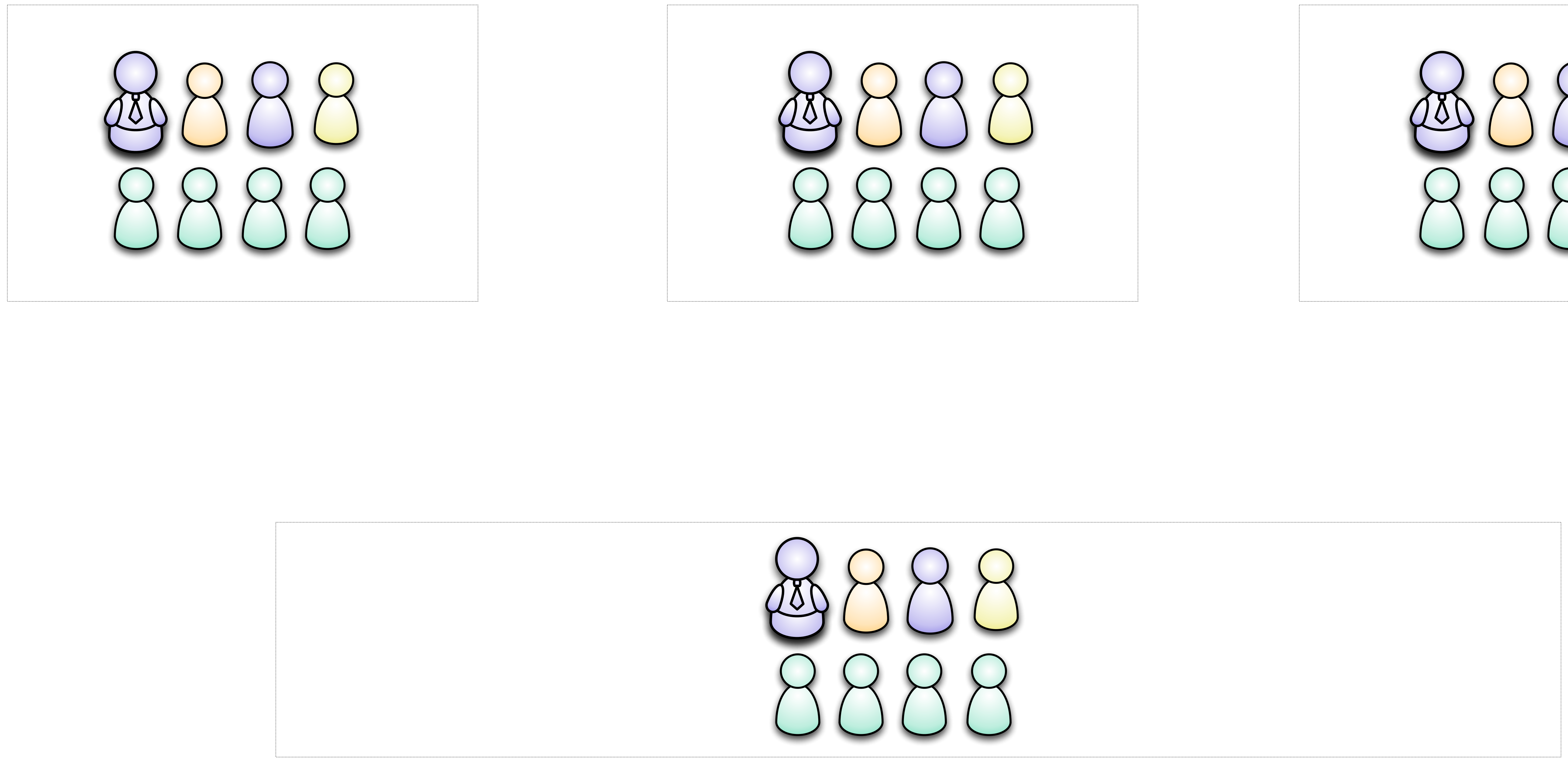
motor

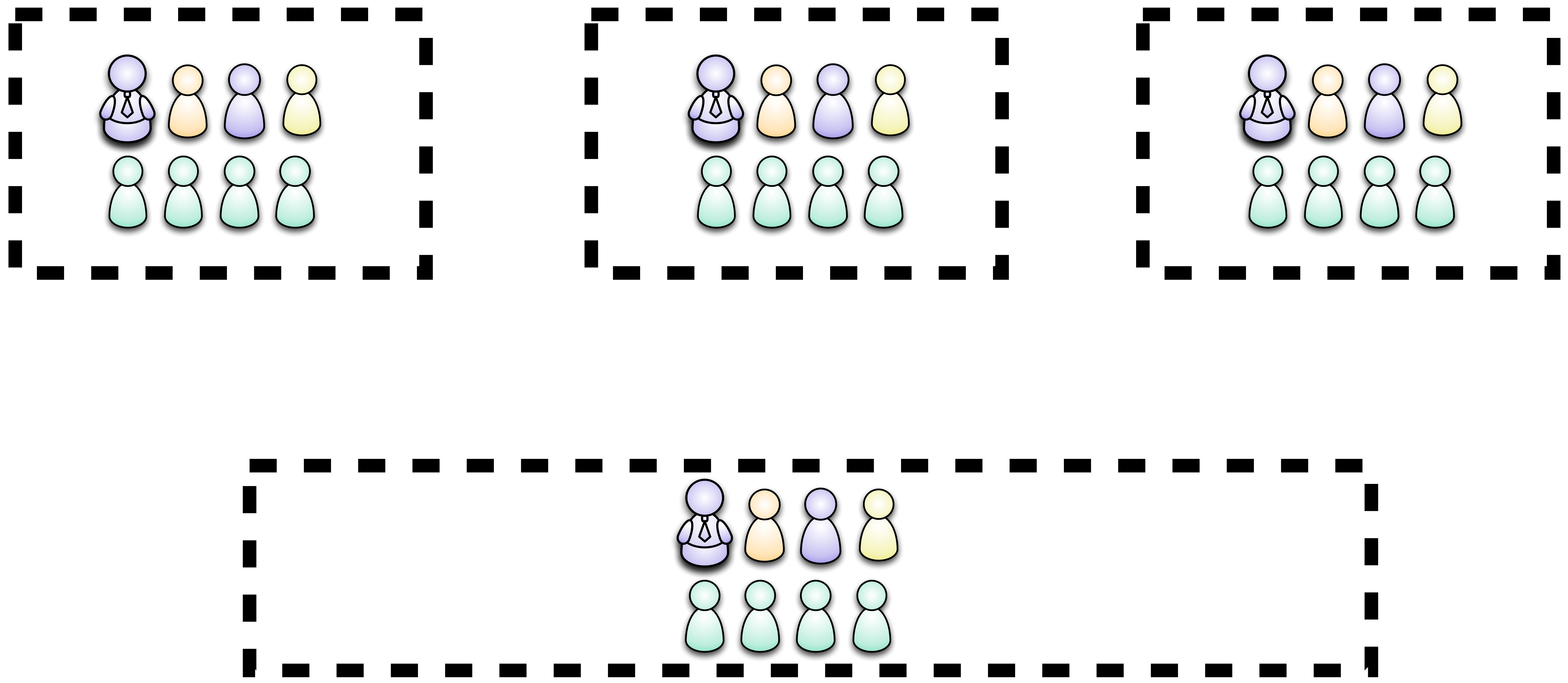
life

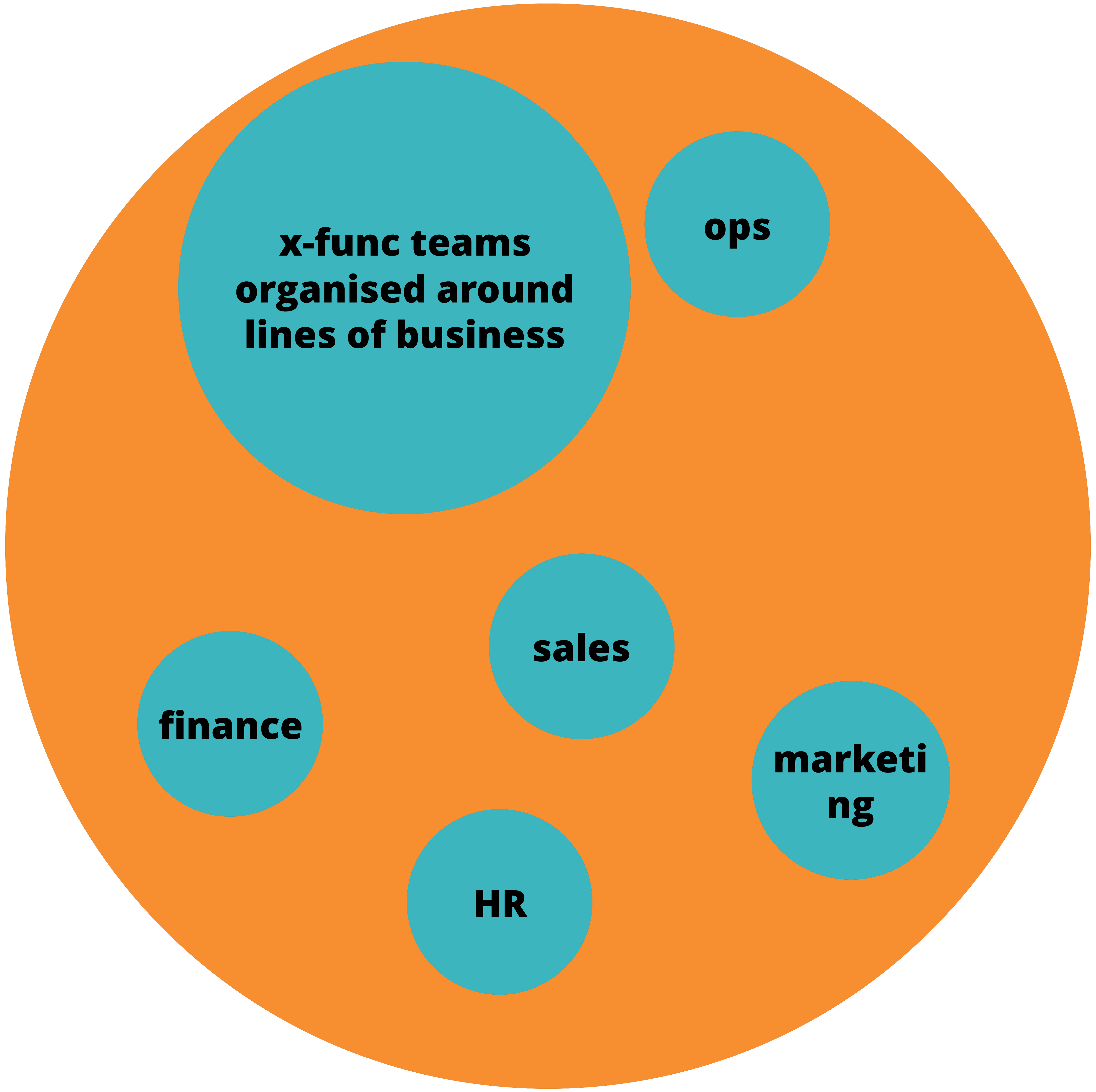
and cross-cutting capabilities

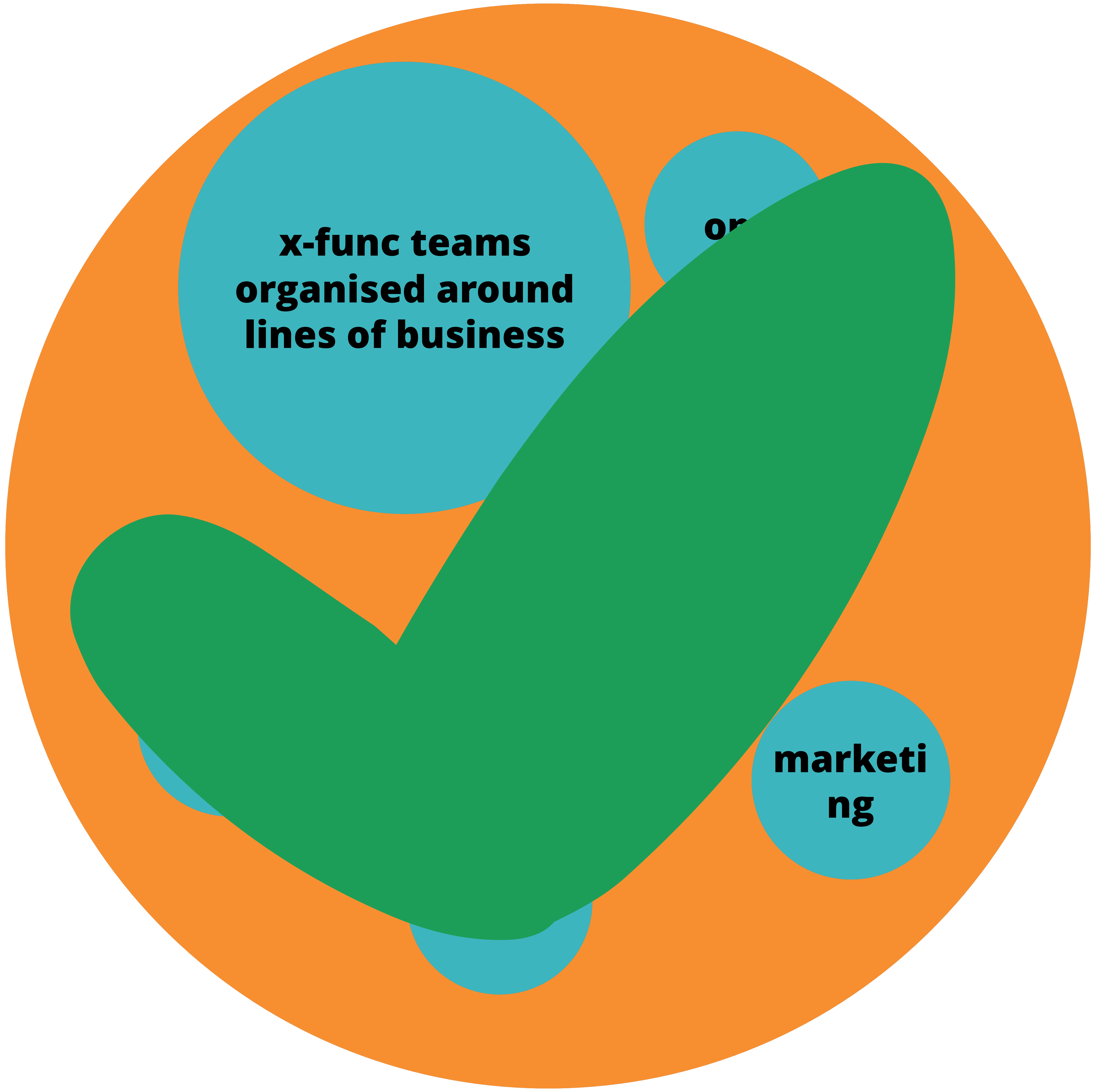
my account

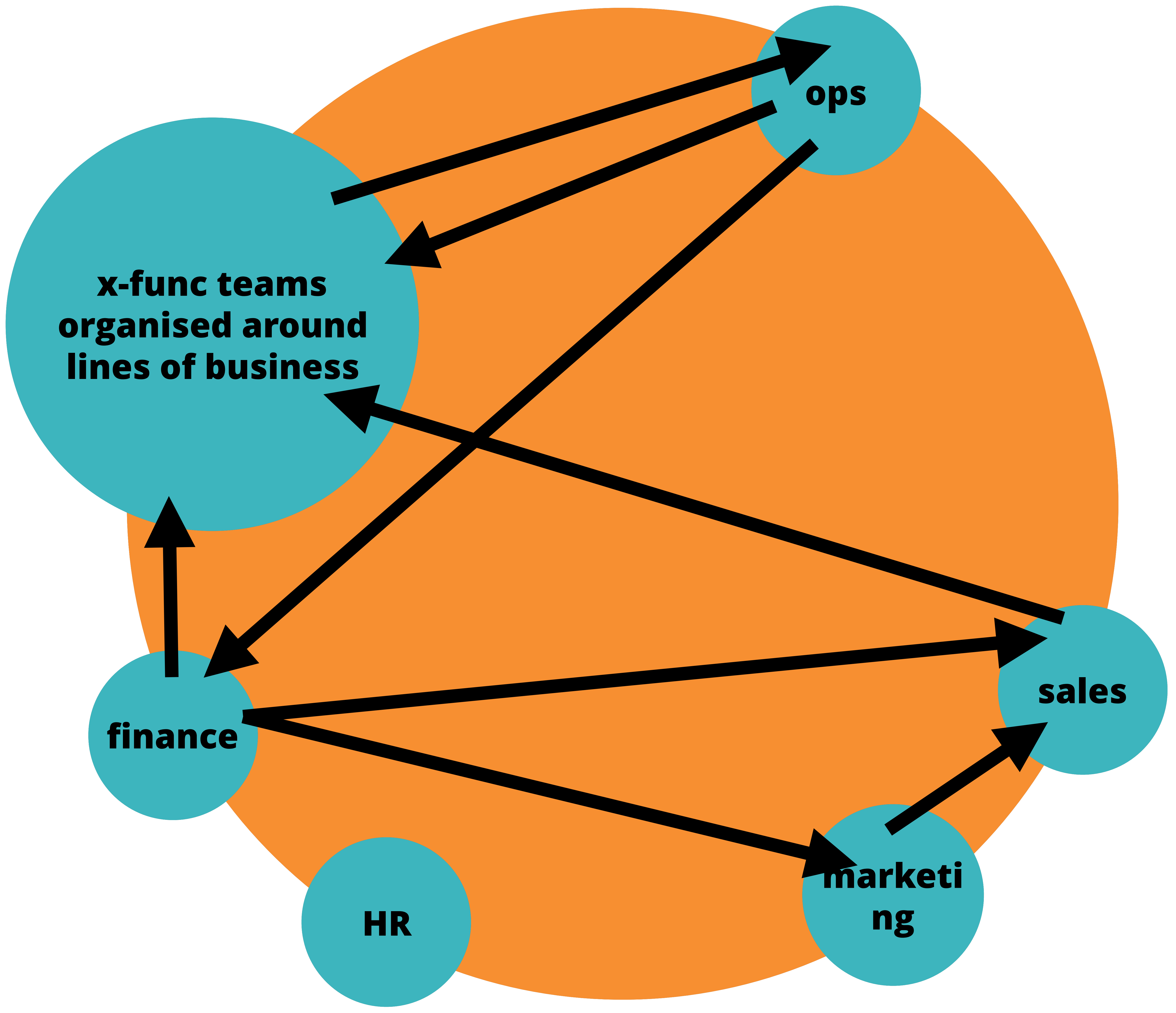
cross-functional teams delivering lines of business






x-func teams organised around lines of business
ops
sales
finance
marketi ng
HR


x-func teams organised around lines of business
on
marketi ng


ops
x-func teams organised around lines of business
finance
sales
HR
marketi ng

Jez Humble

This card ranks 14 if only Fruits were committed to this iteration. Otherwise, this card ranks 0.

note: this counts only during iterations

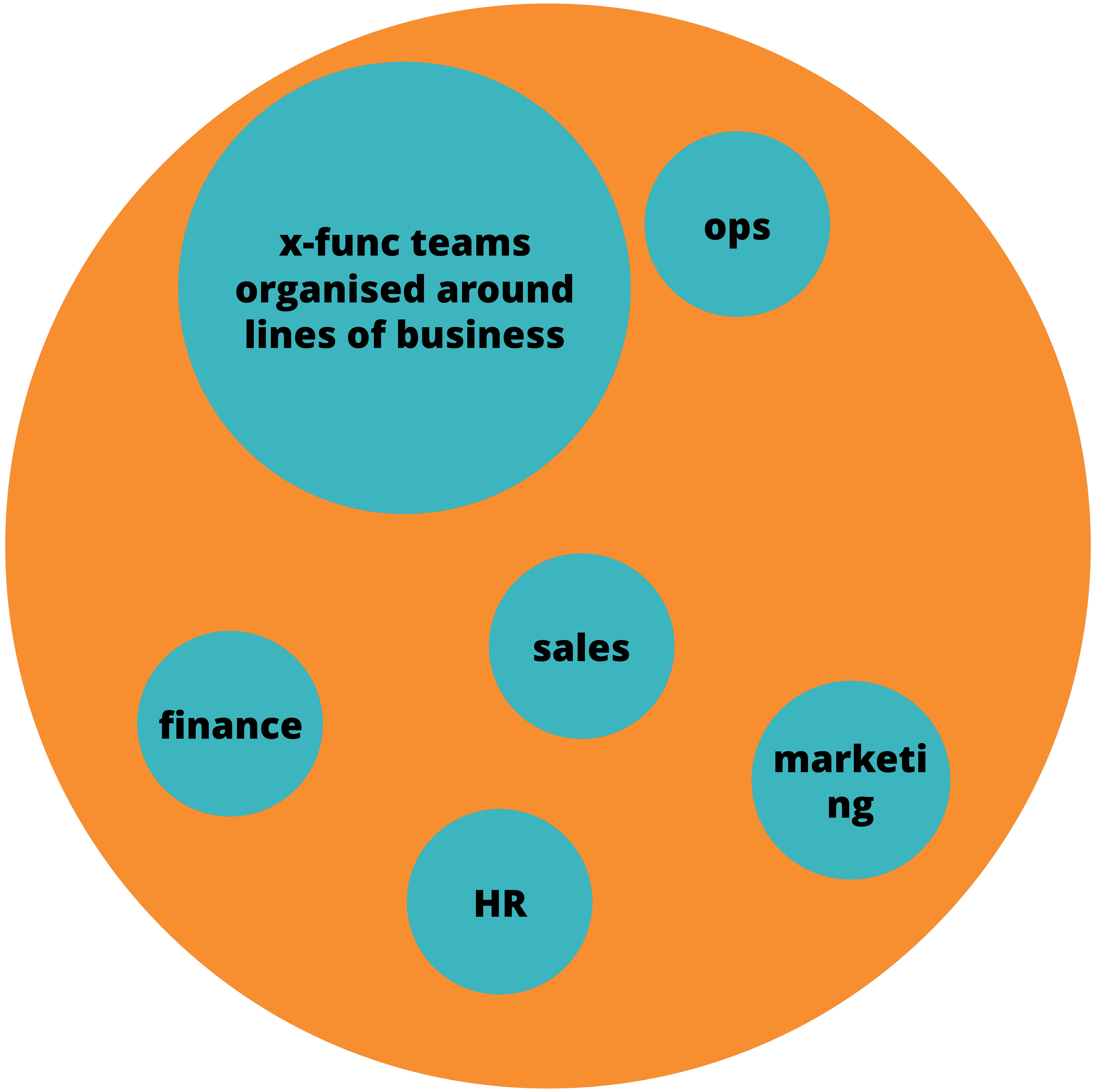
10

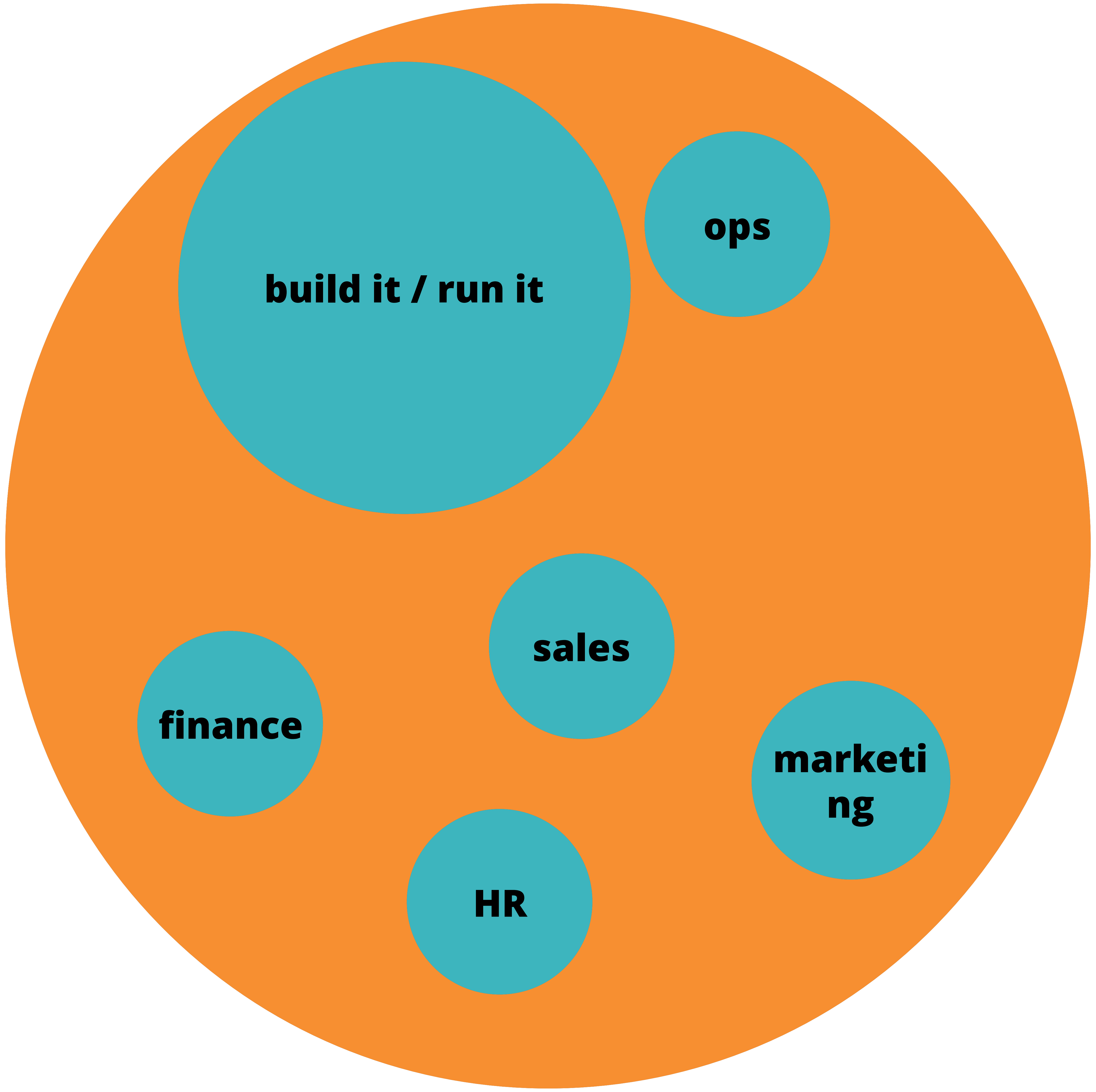
© 2014 Inedo R!34 inedo.com/release

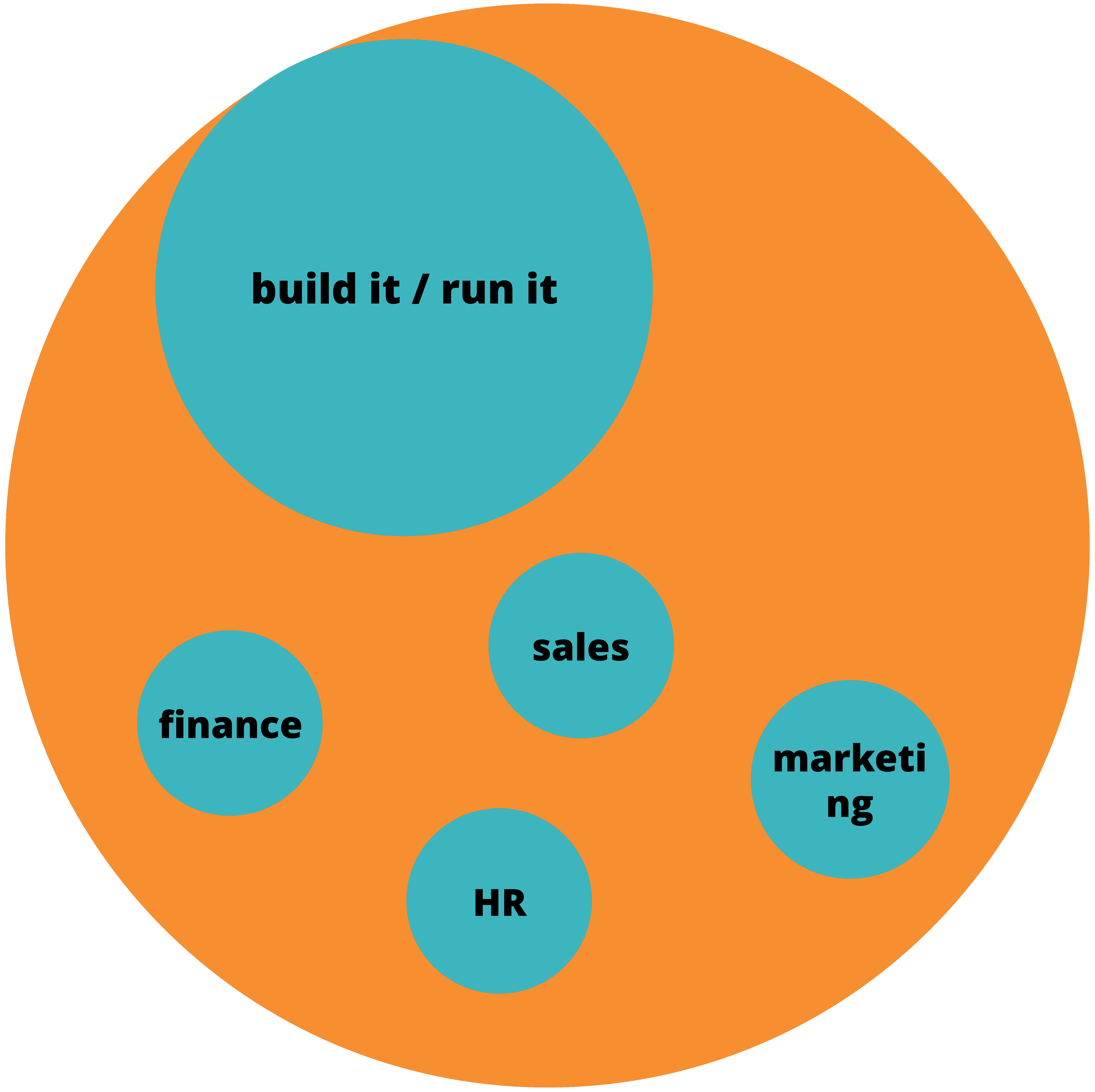
DEVOPS!!!

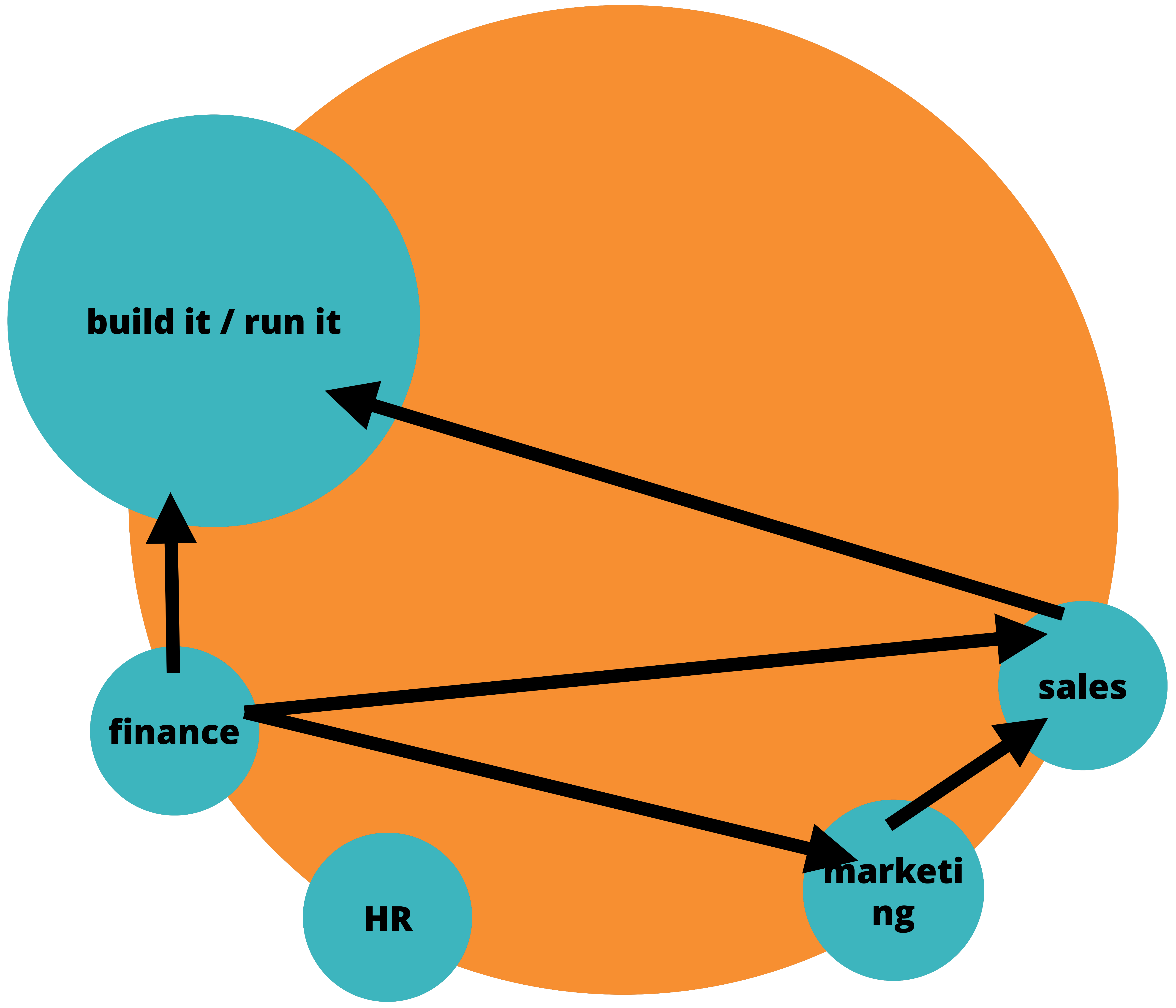
DEVOPS!!!

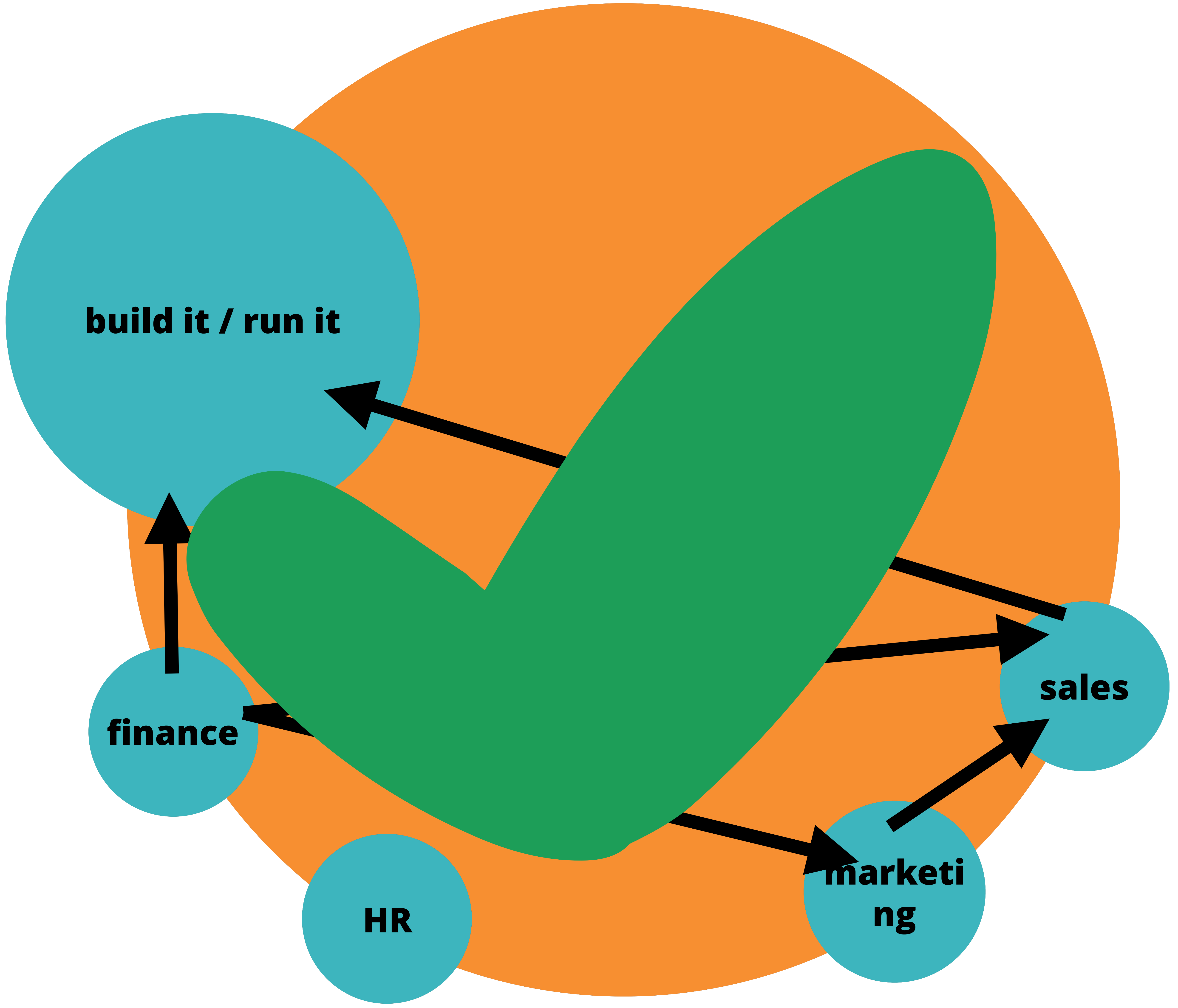
Well, build it, run it…


x-func teams organised around lines of business
ops
sales
finance
marketi ng
HR

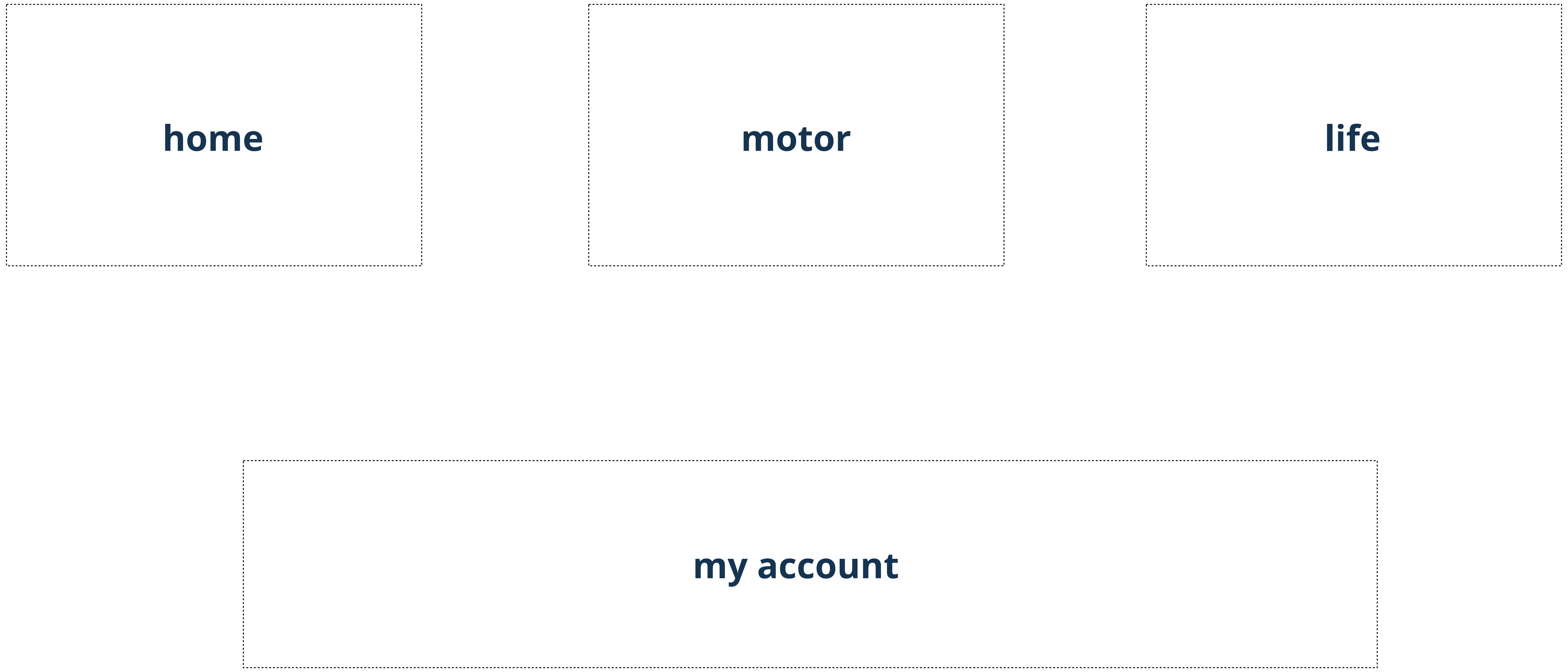

build it / run it
ops
sales
finance
marketi
ng
HR

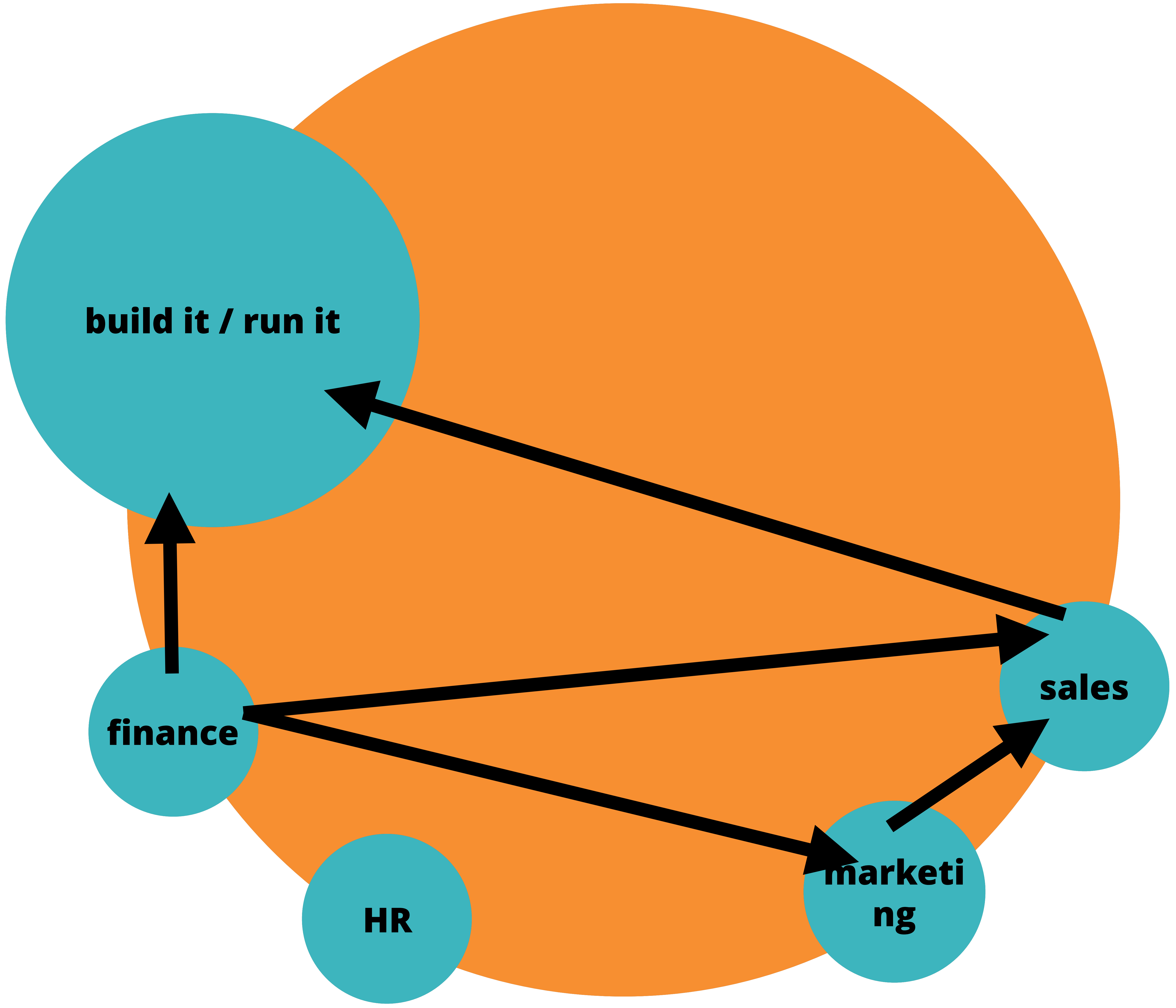

build it / run it
sales
finance
marketi
ng
HR


build it / run it
sales
finance
marketi
ng
HR

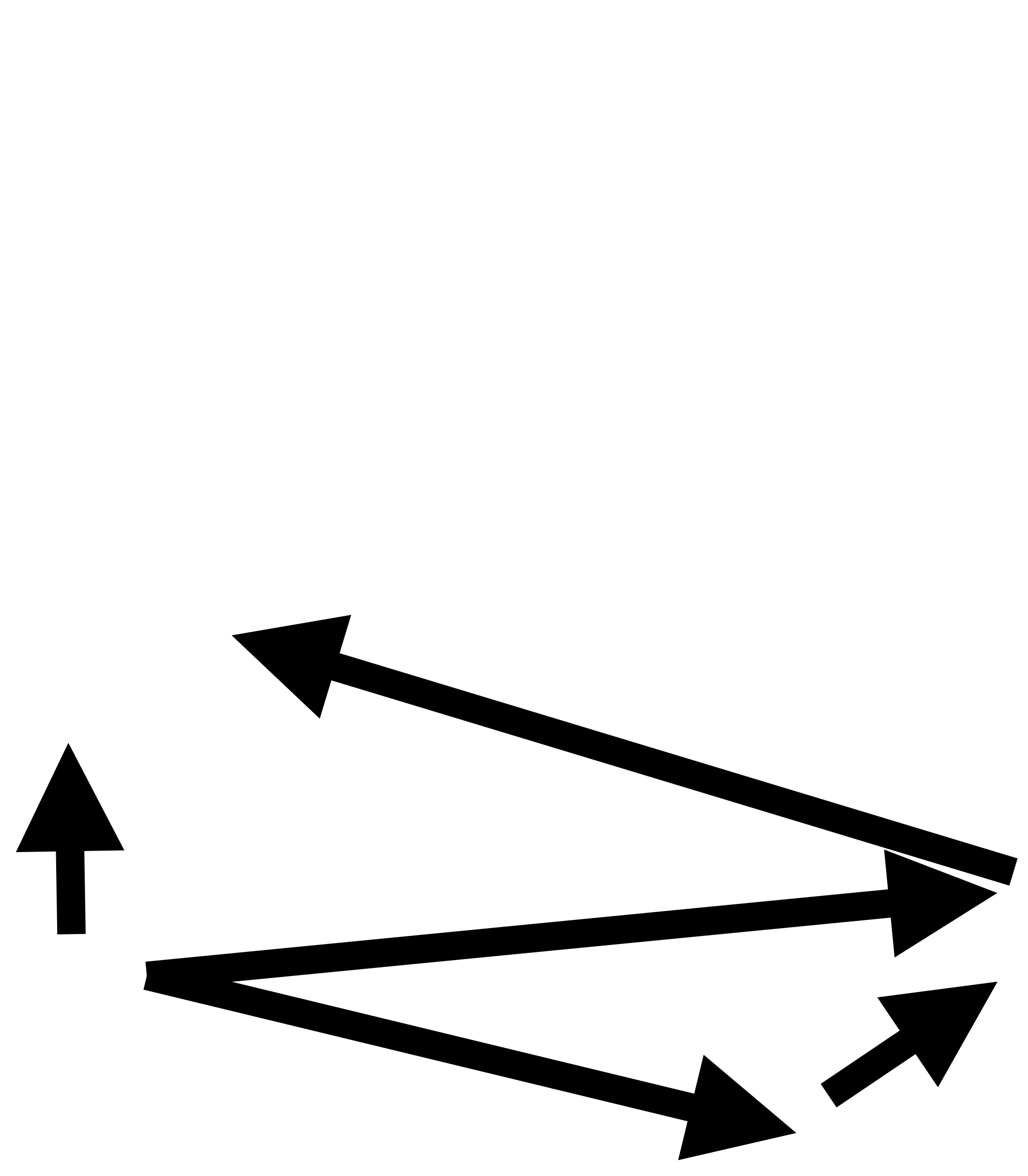

build it / run it
finance
HR
sales
marketi
ng

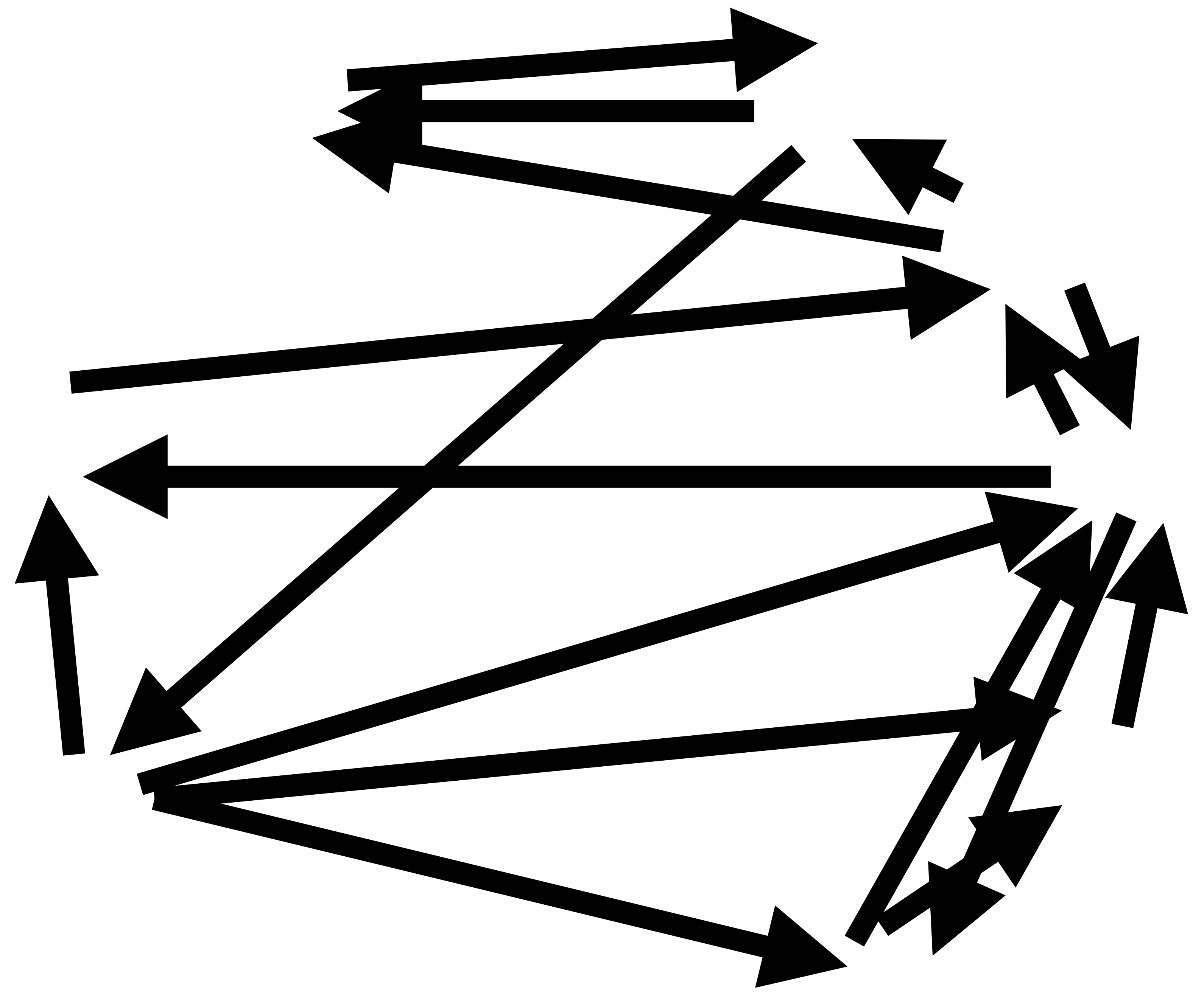
each of these capabilities can be tested and deployed independently



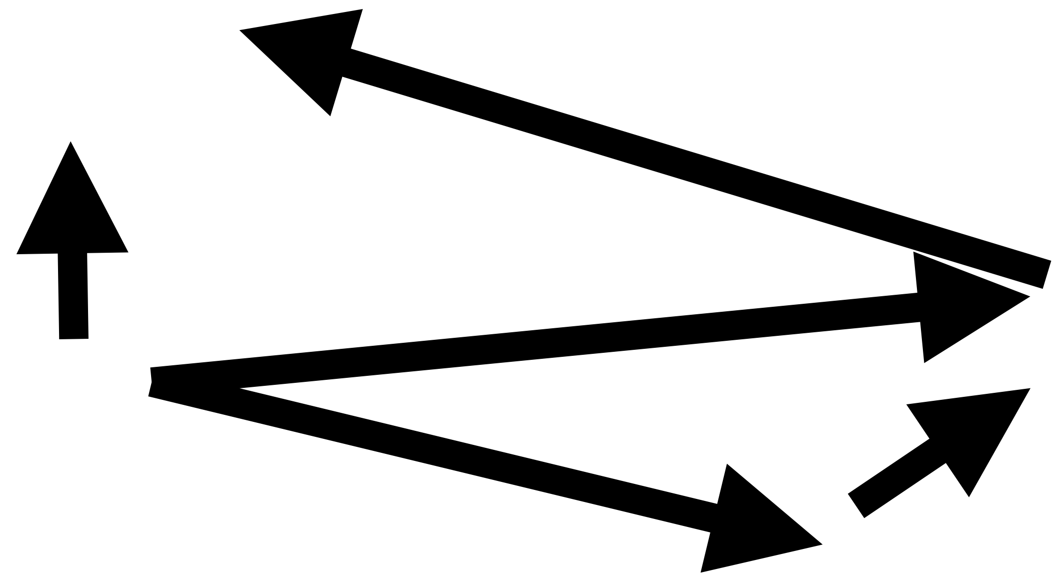

build it / run it
finance
sales
HR
marketi
ng

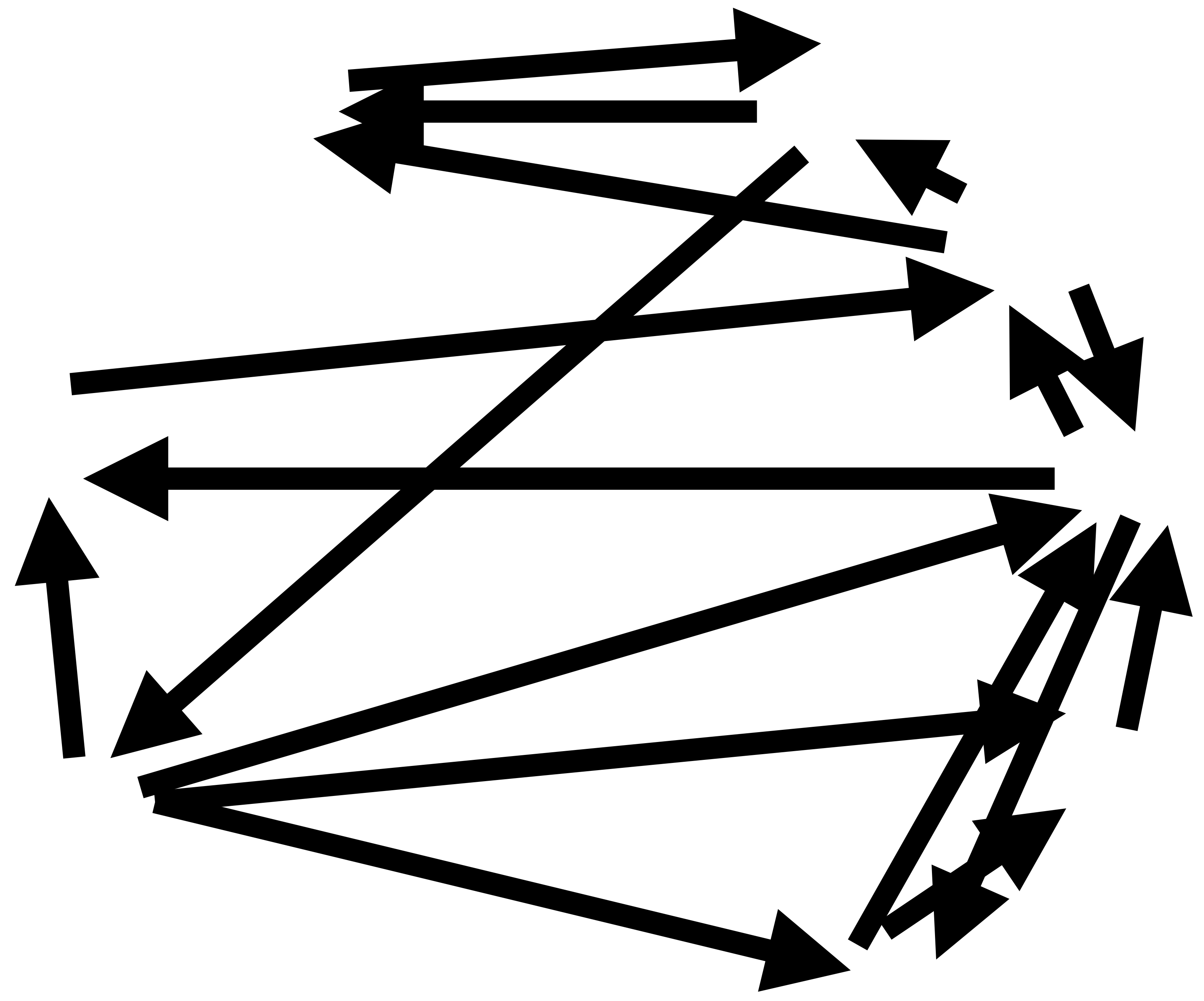
getting there...





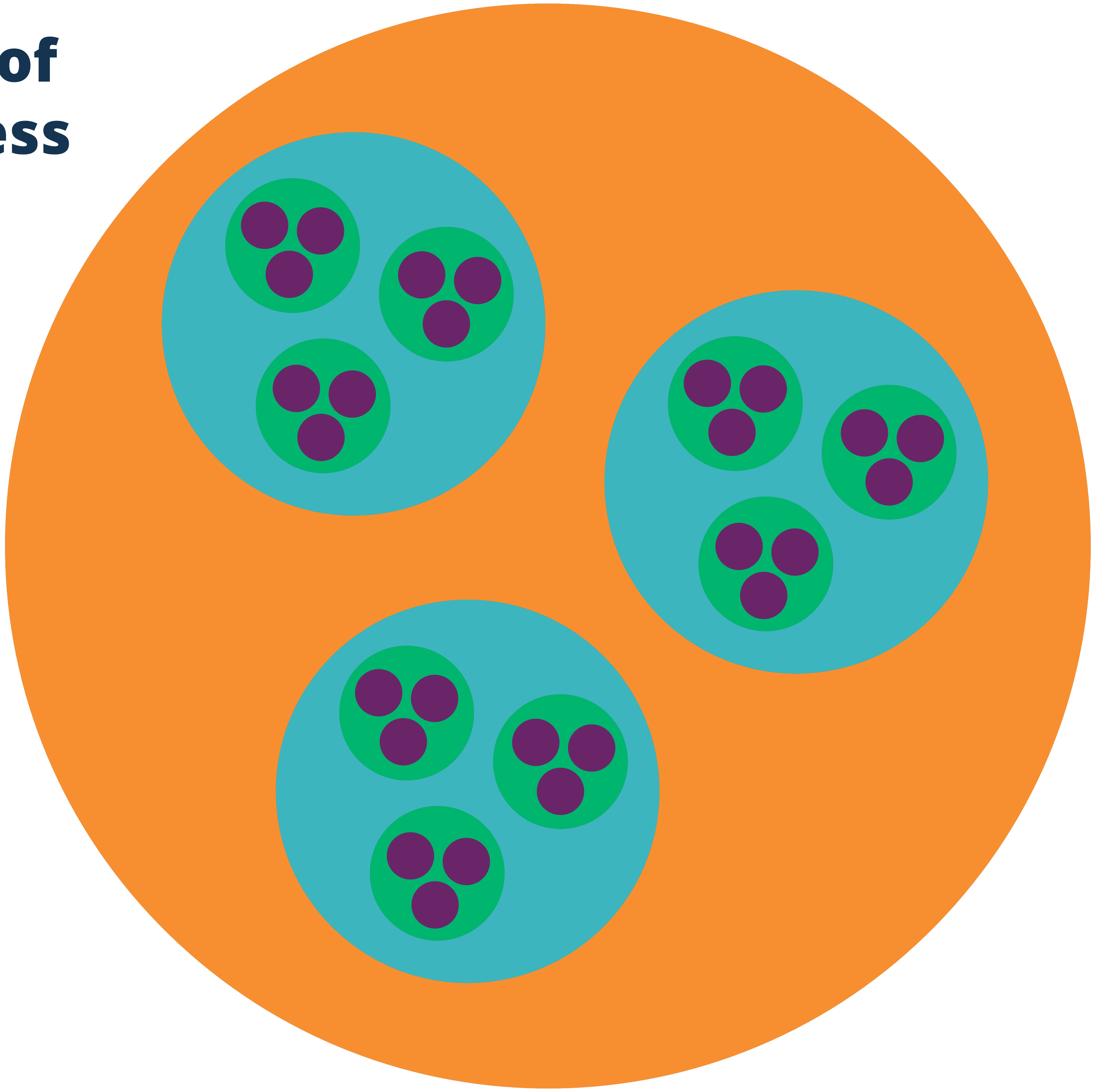
getting there...



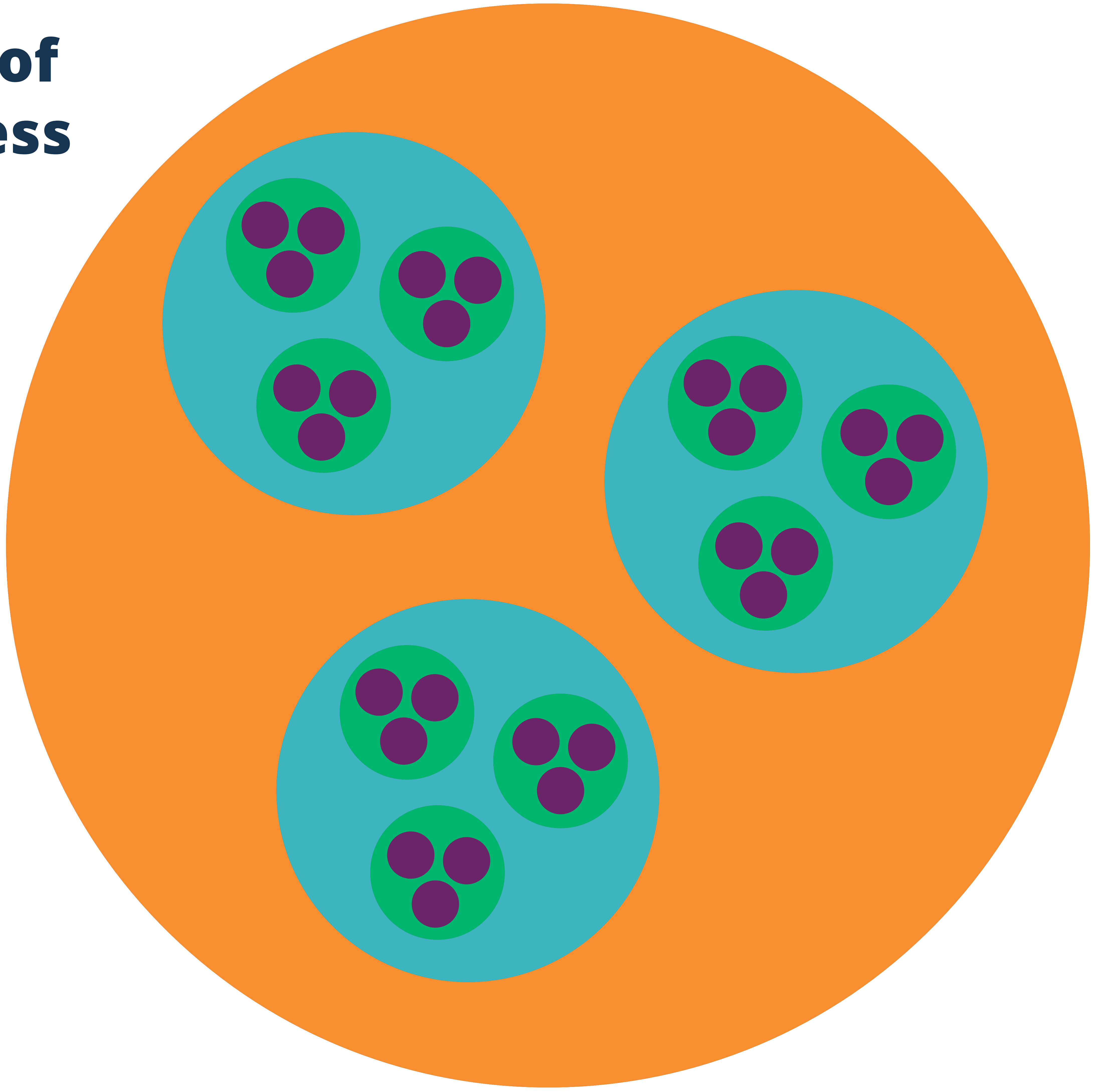


But can we go further?

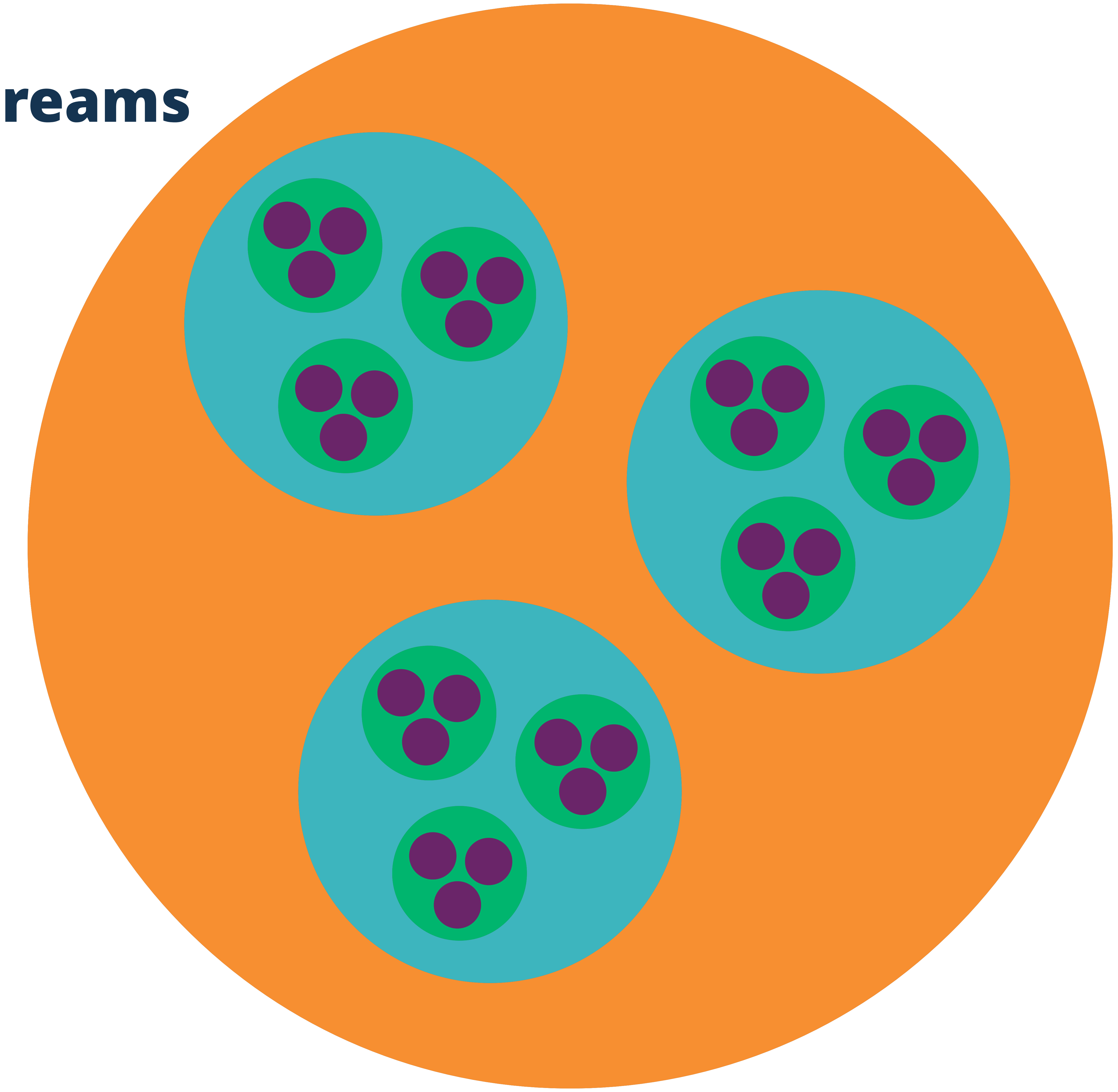
Lines of business



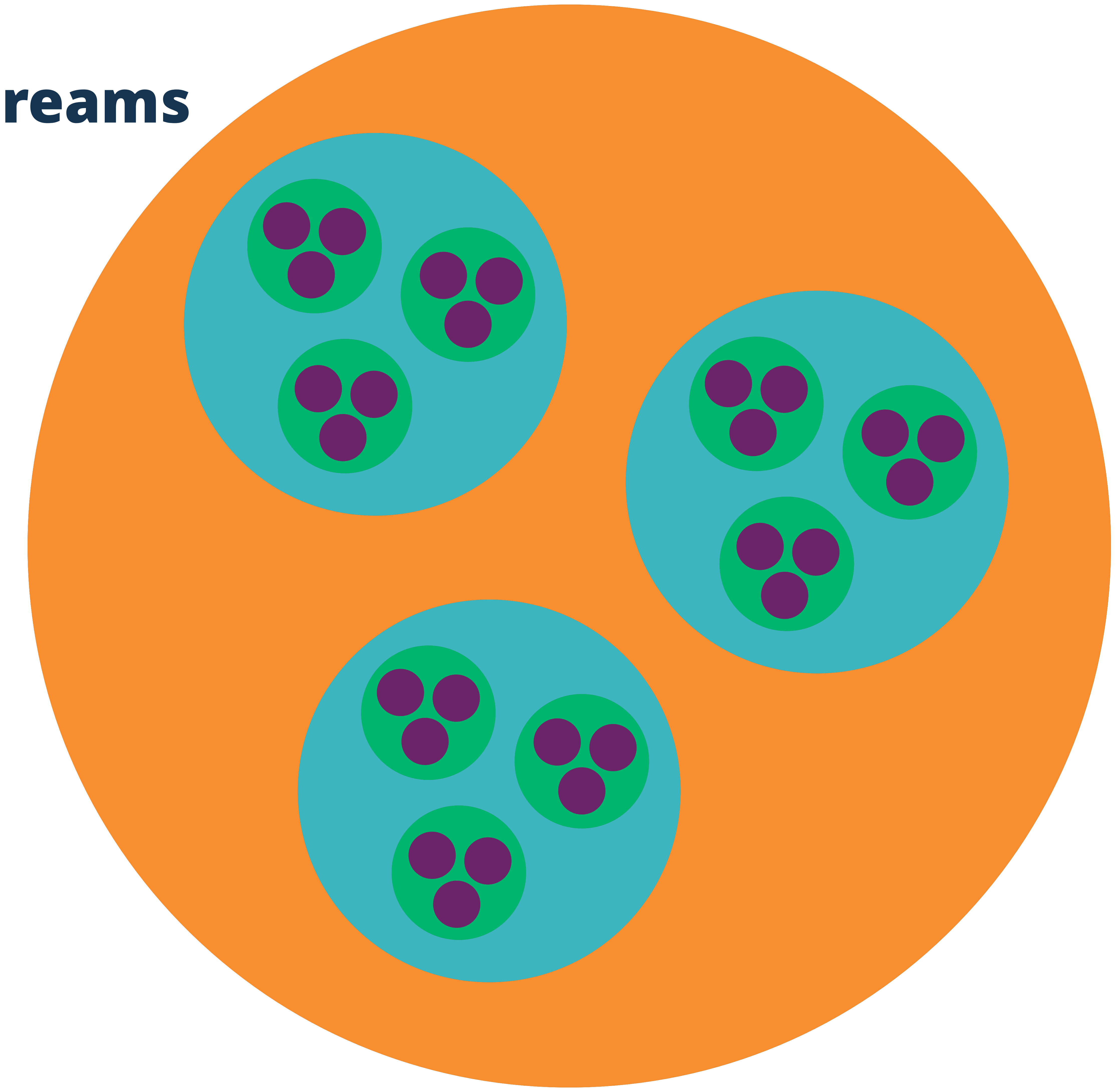
Lines of business



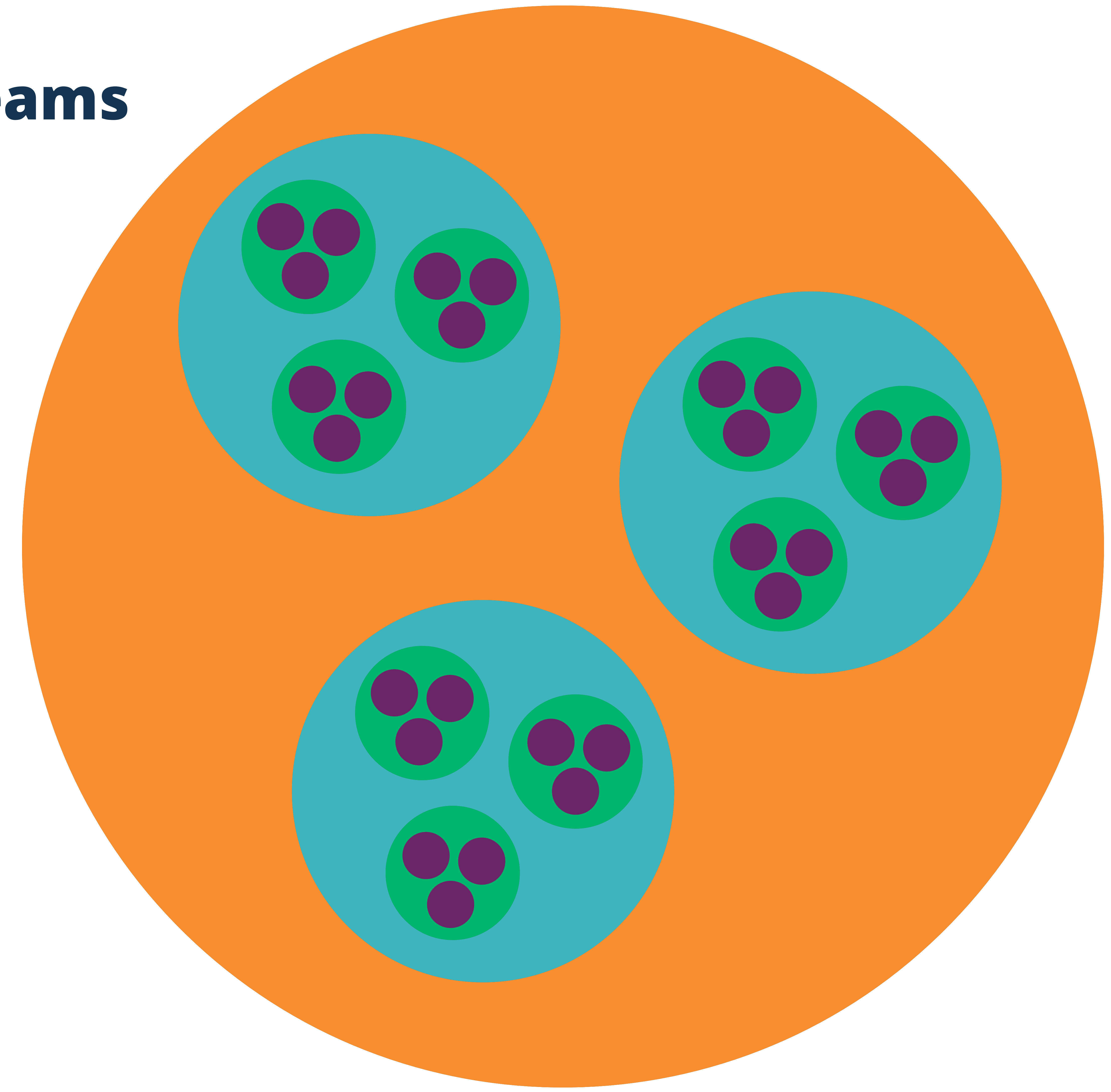
Value streams



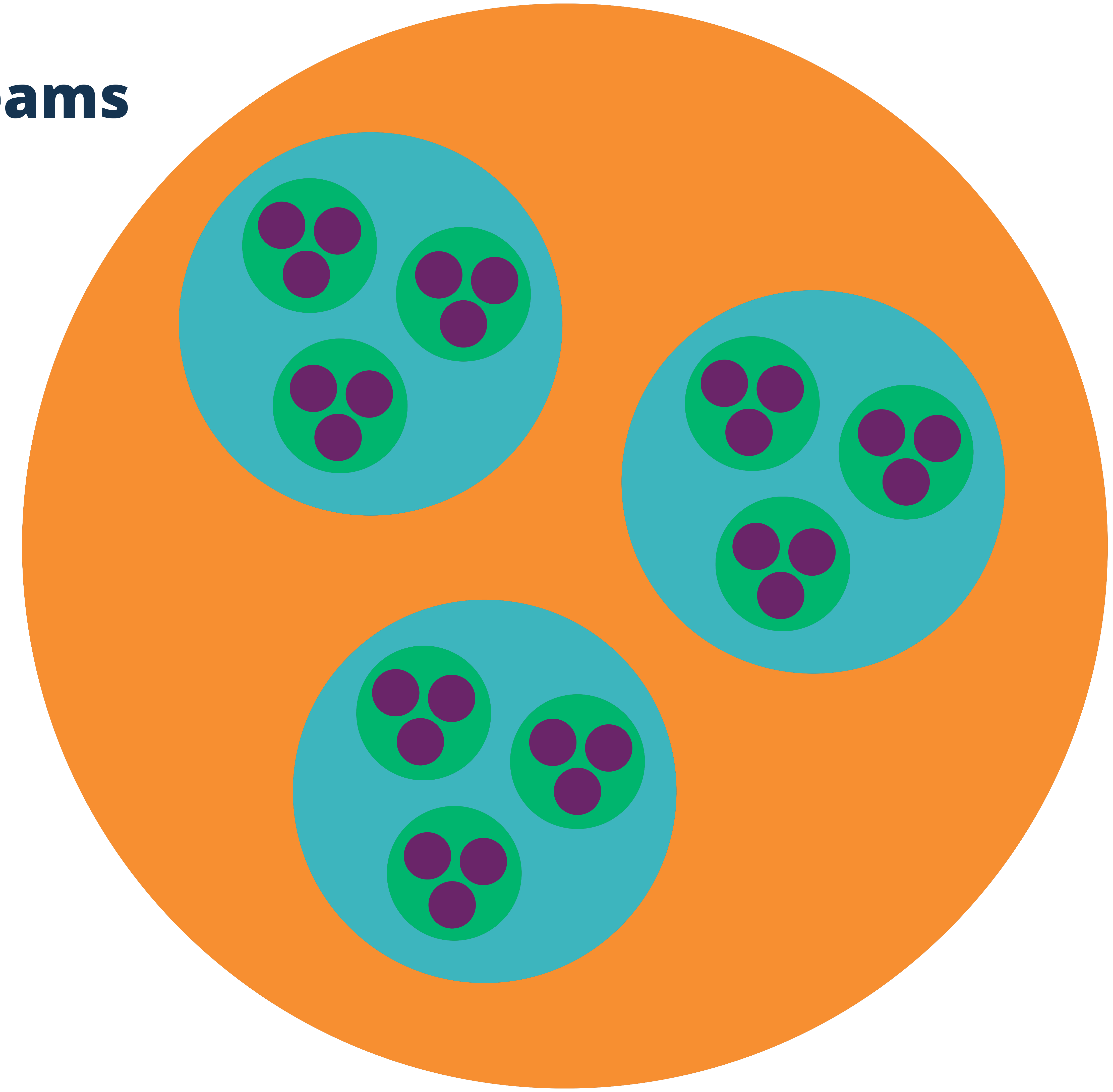
Value streams



teams



teams



each team

owns* one *or* more *services

~10-20

~10-20

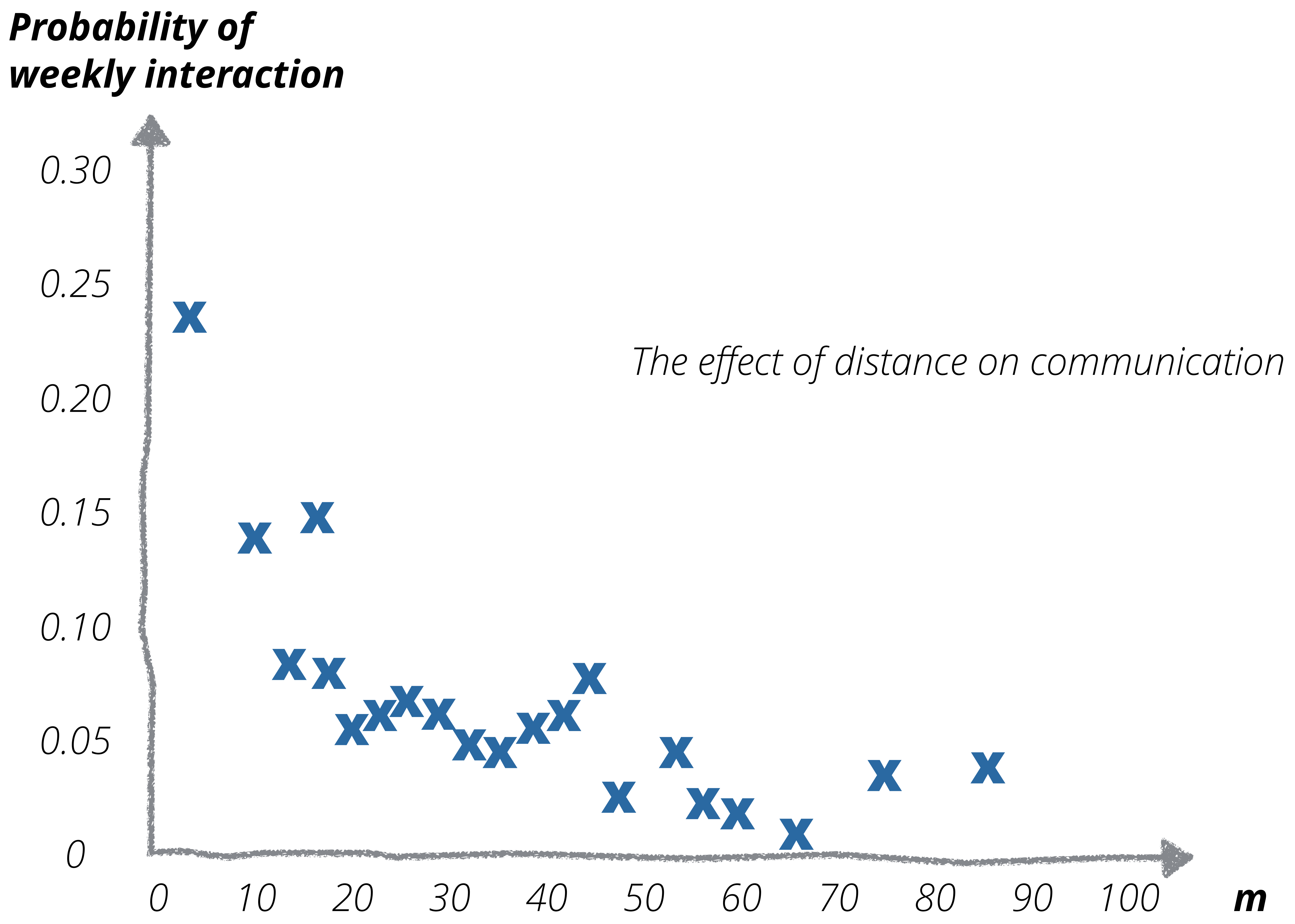
~160-200

~10-20

~160-200

multiples thereof

Thomas J. Allen, 1977

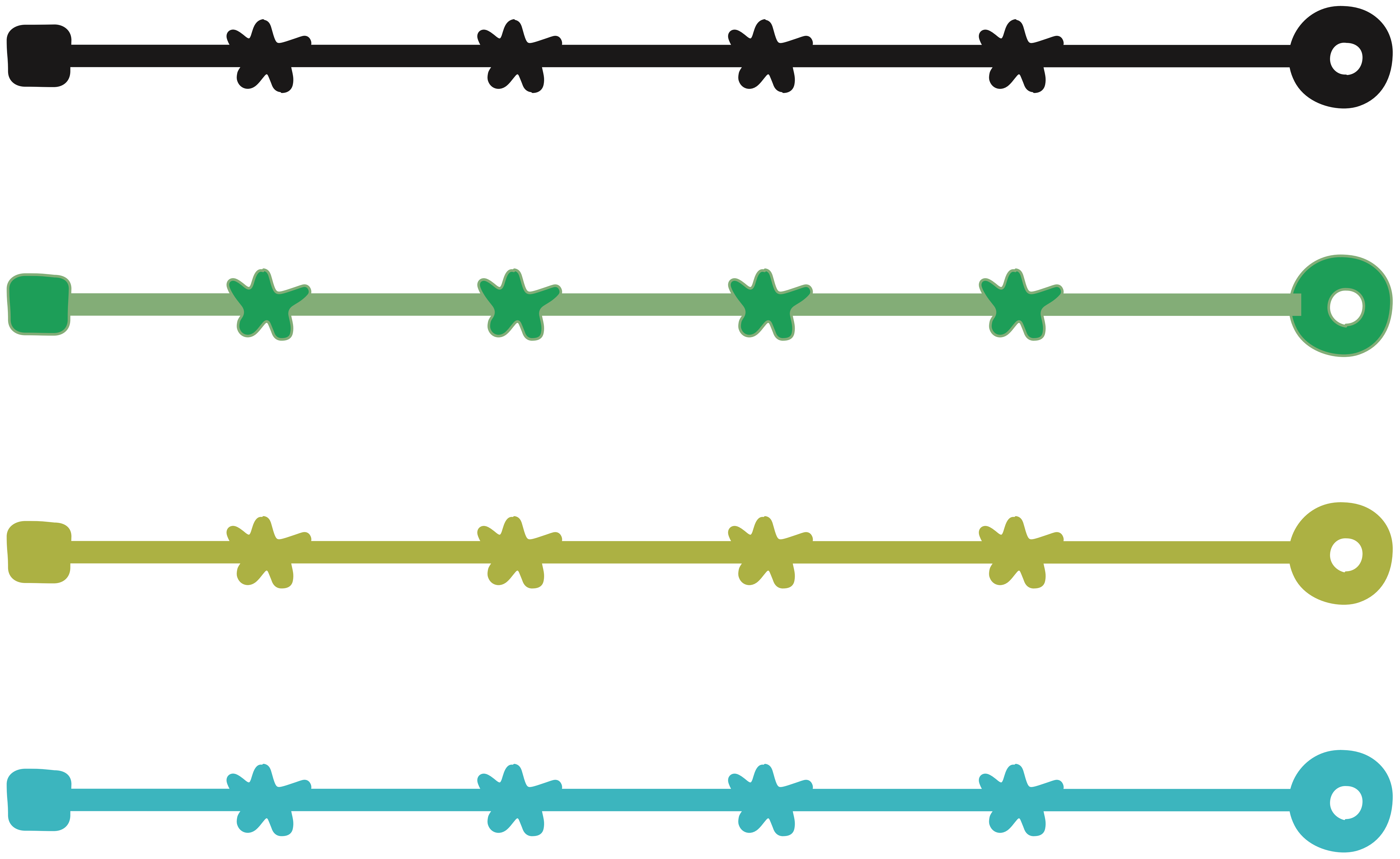


The effect of distance on communication

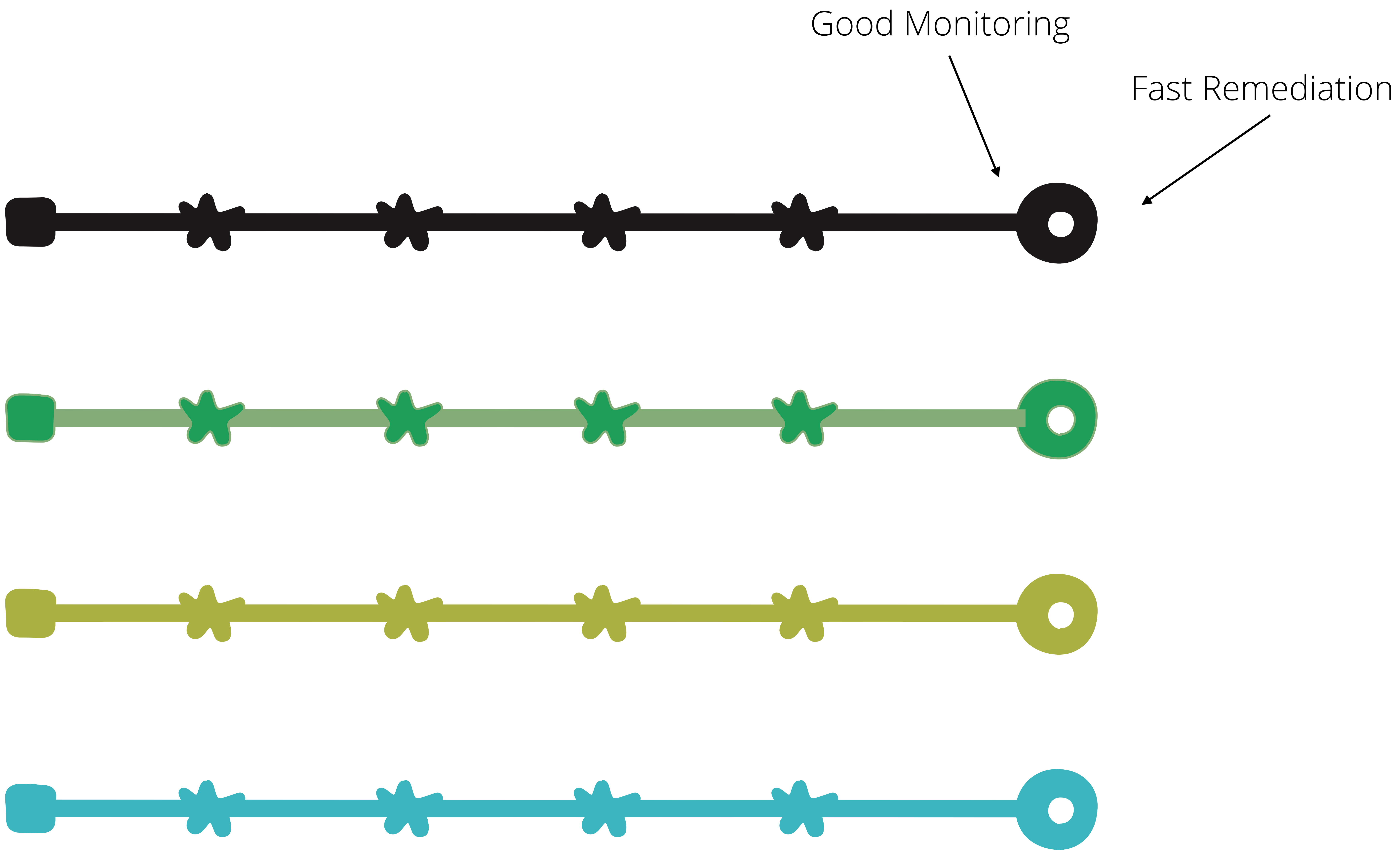
co-locate as much as possible

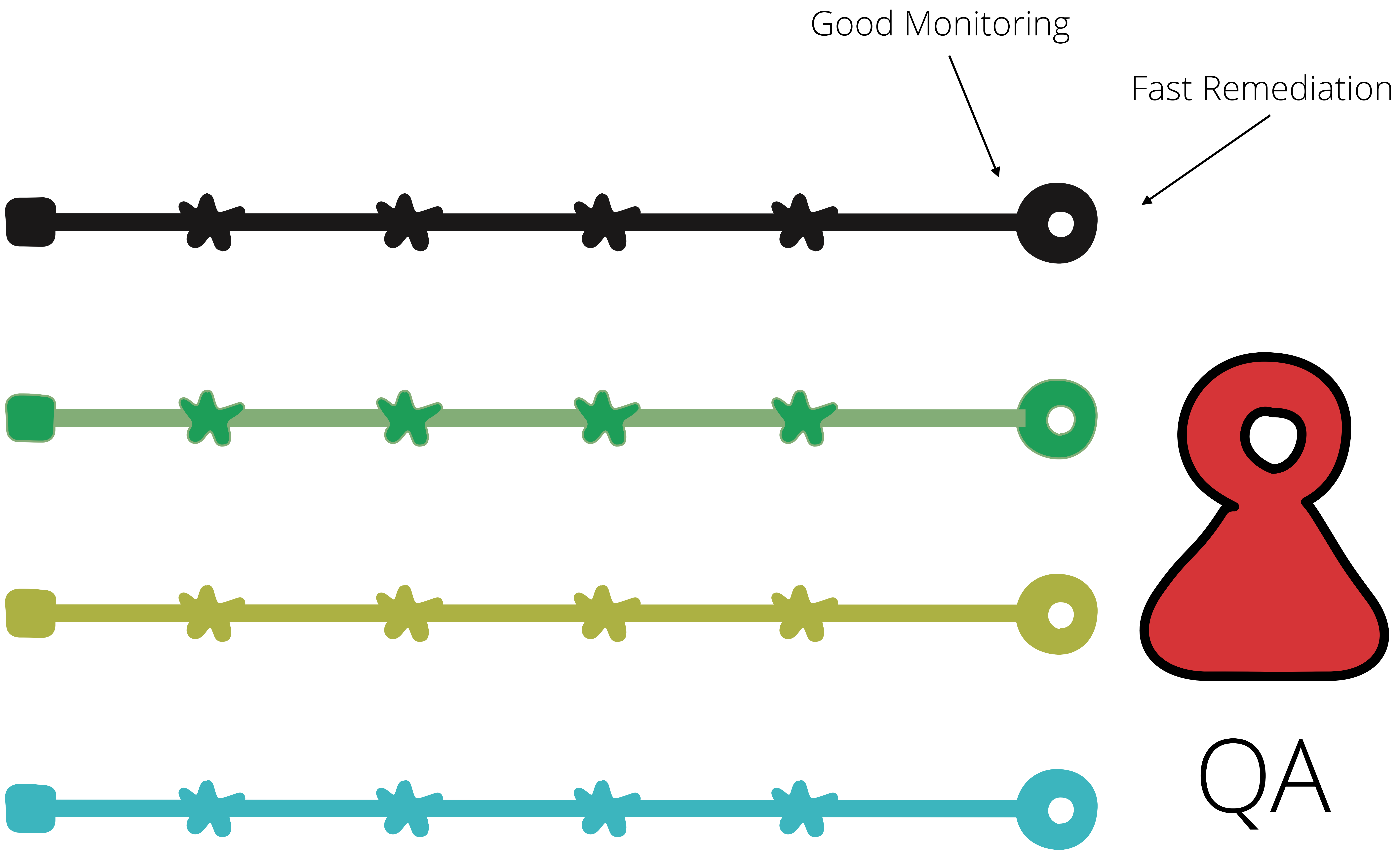
take advantage of serendipitous conversations

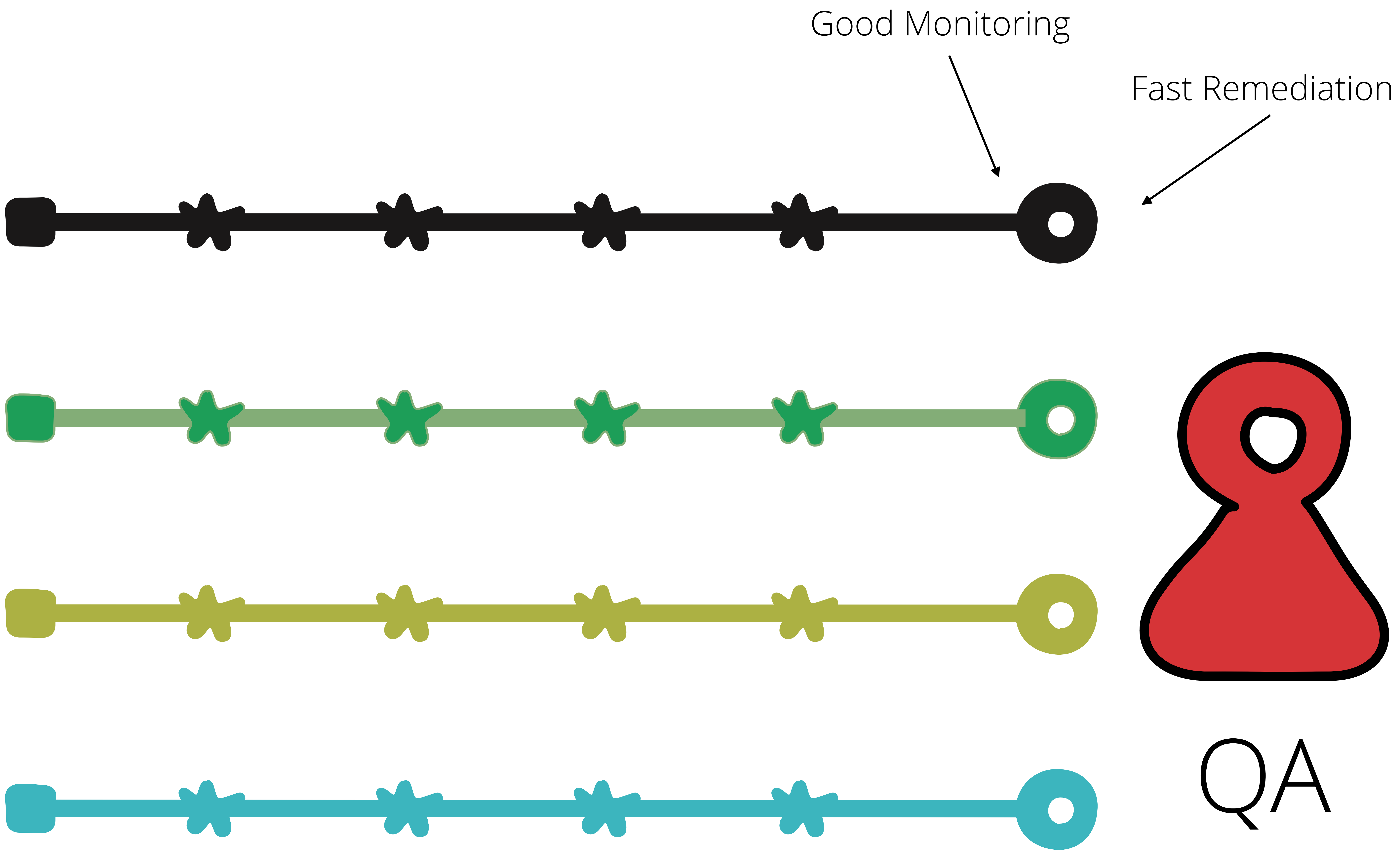
we aren’t in Kansas anymore



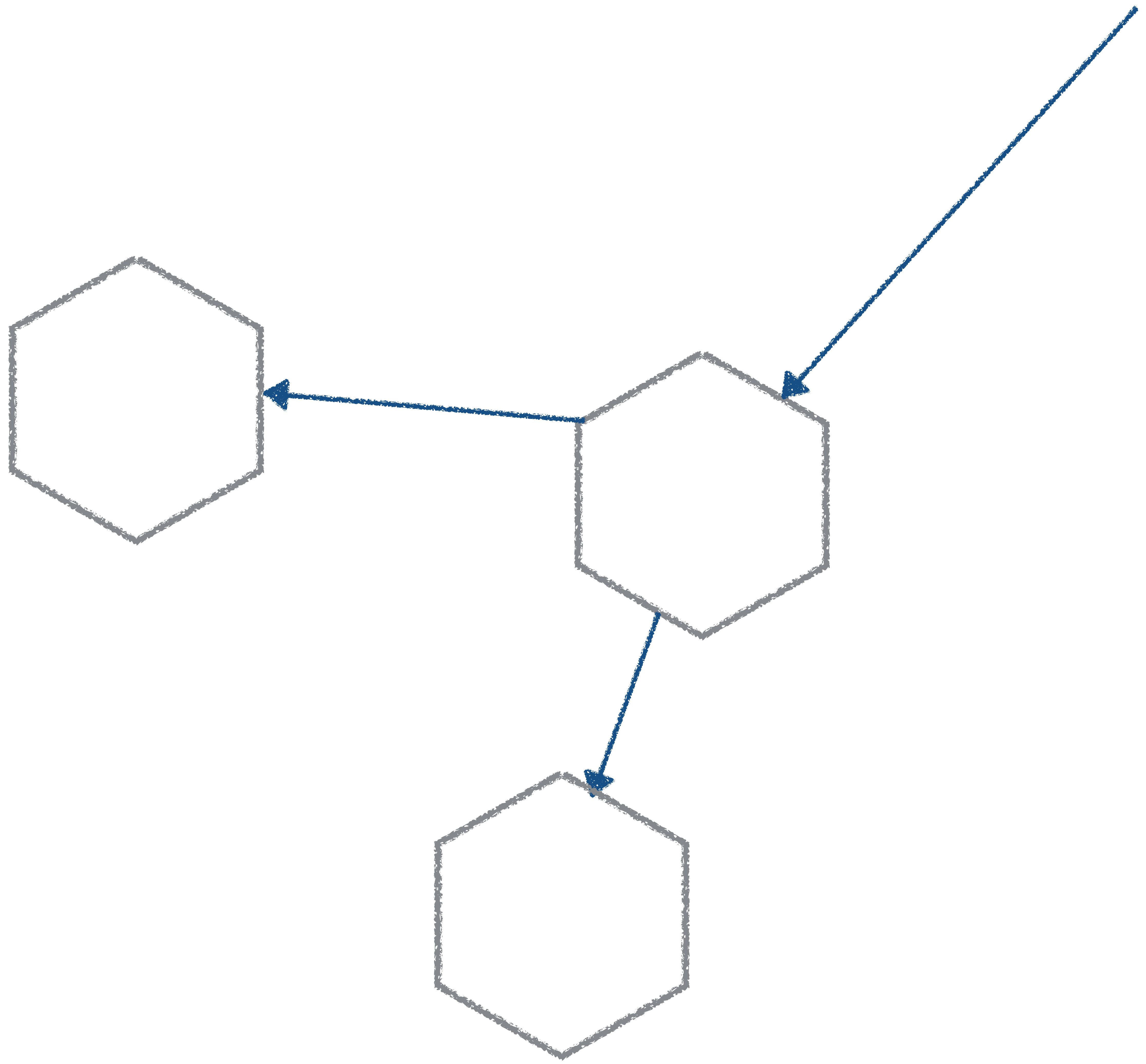
Good Monitoring

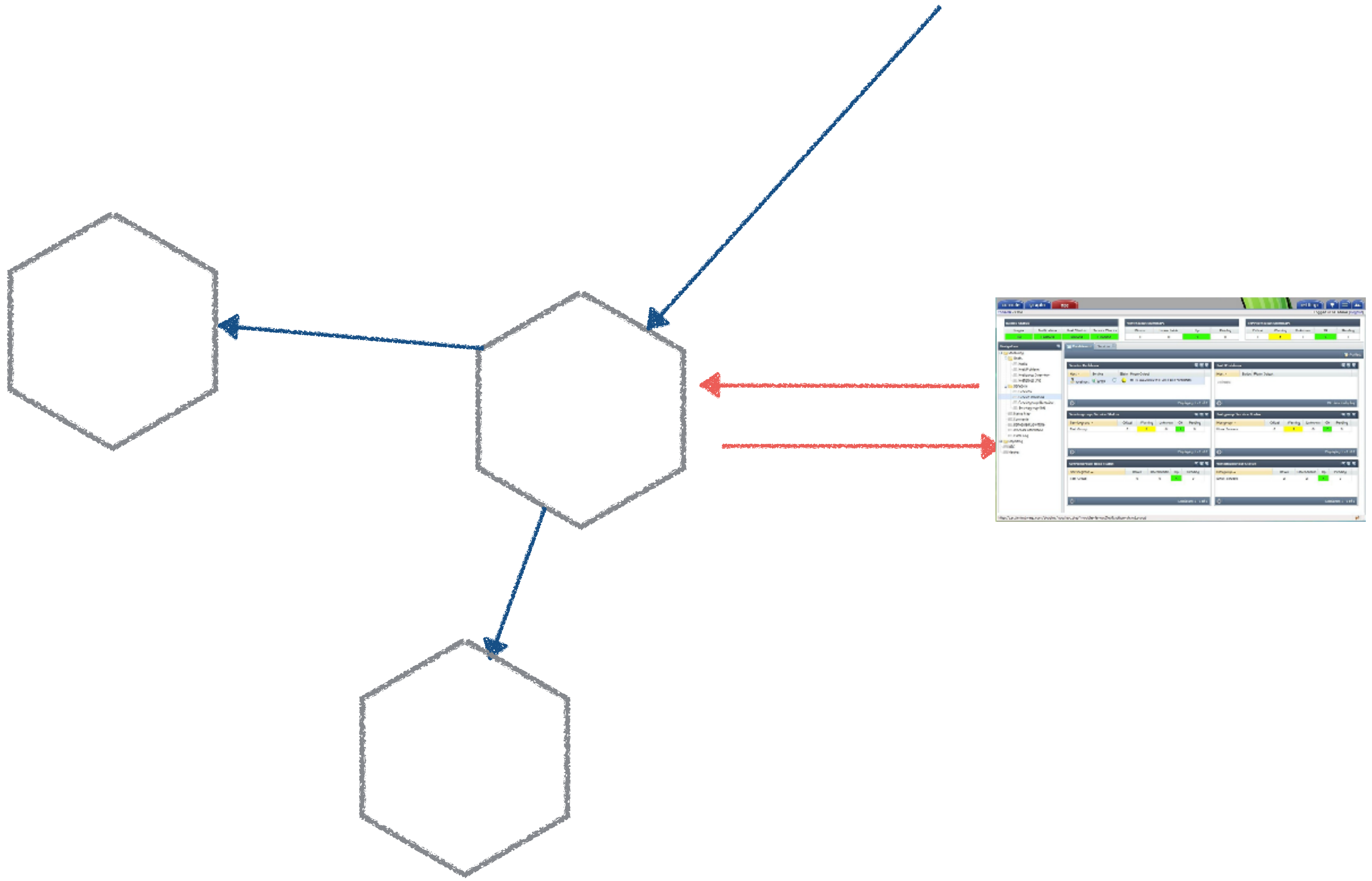

Good Monitoring
Fast Remediation


Good Monitoring
Fast Remediation
QA

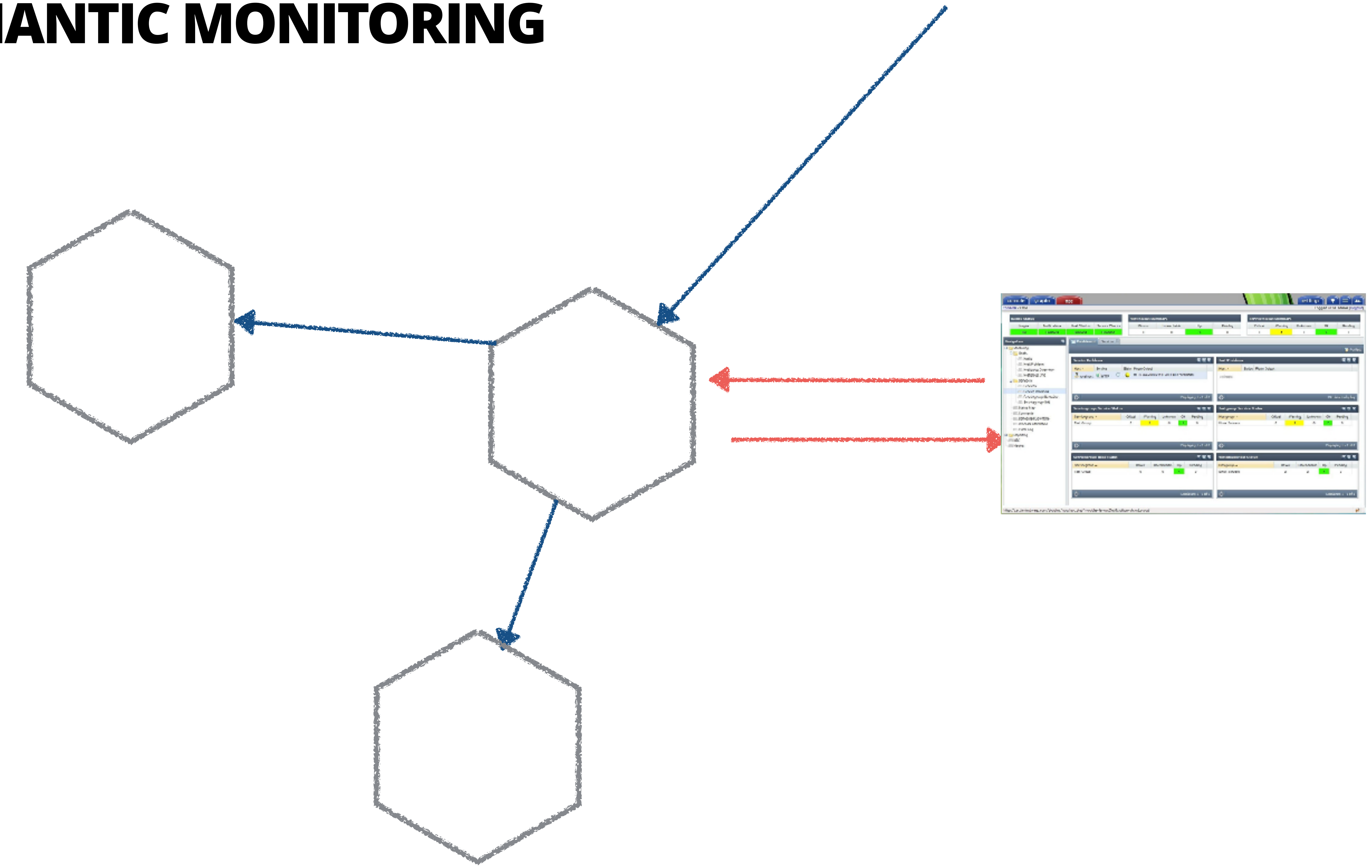


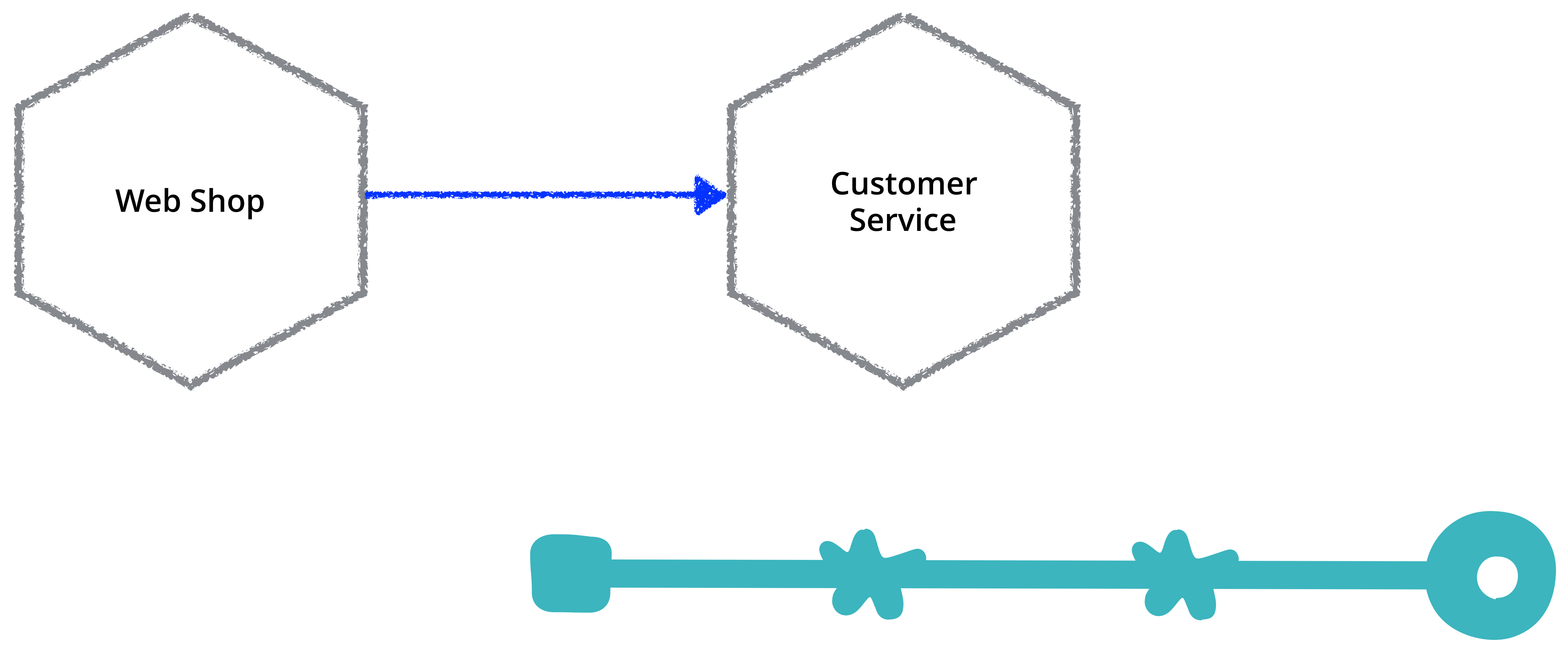
QA in production

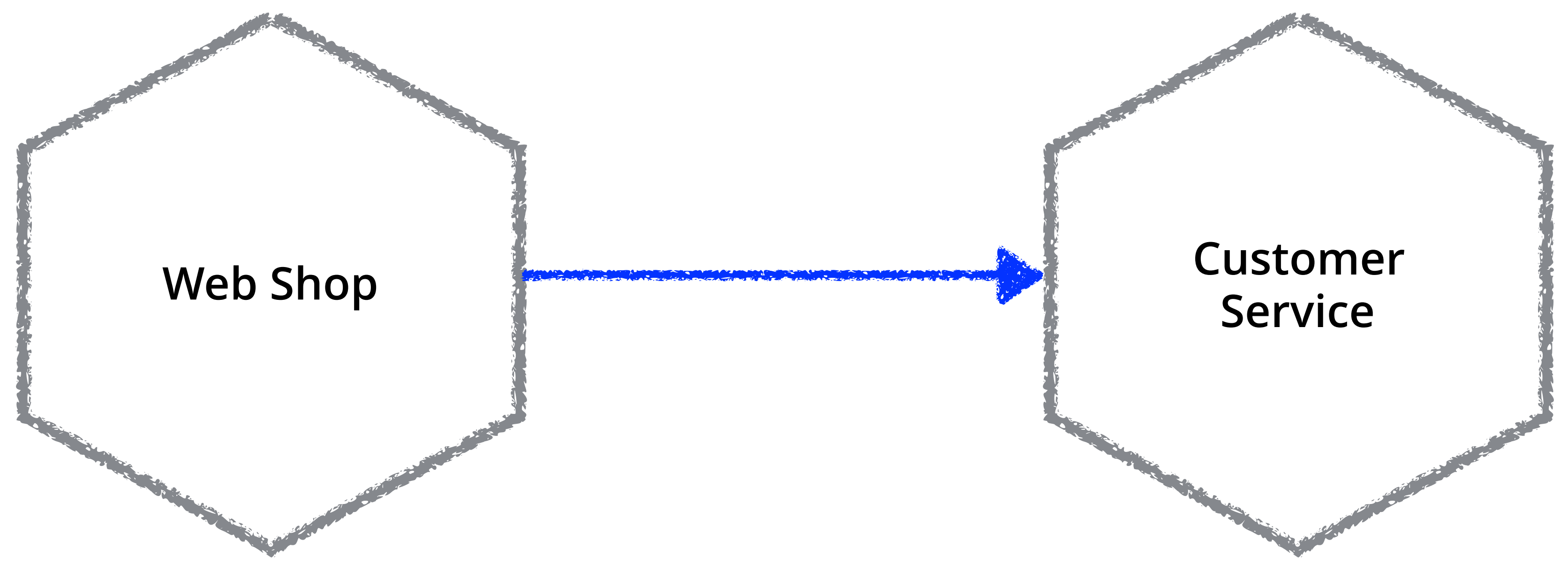




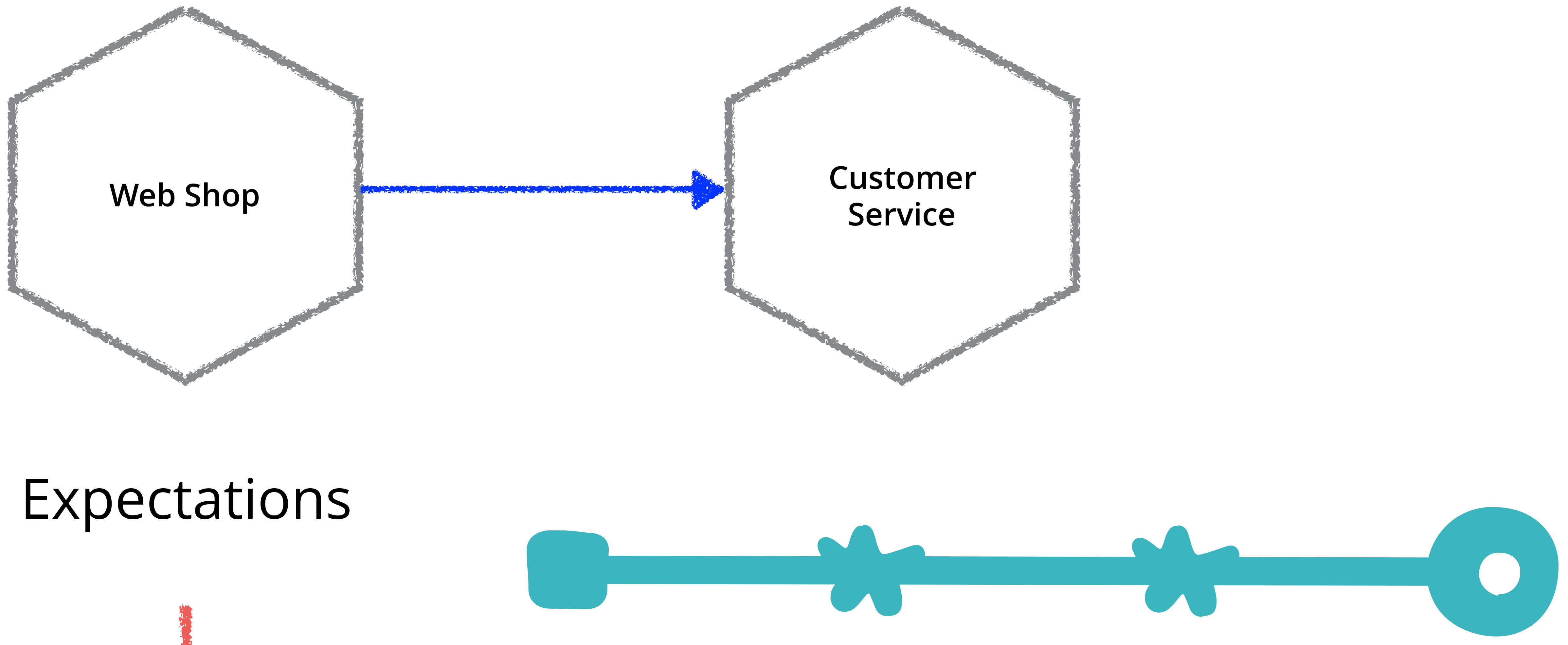
SEMANTIC MONITORING




Web Shop
Customer Service

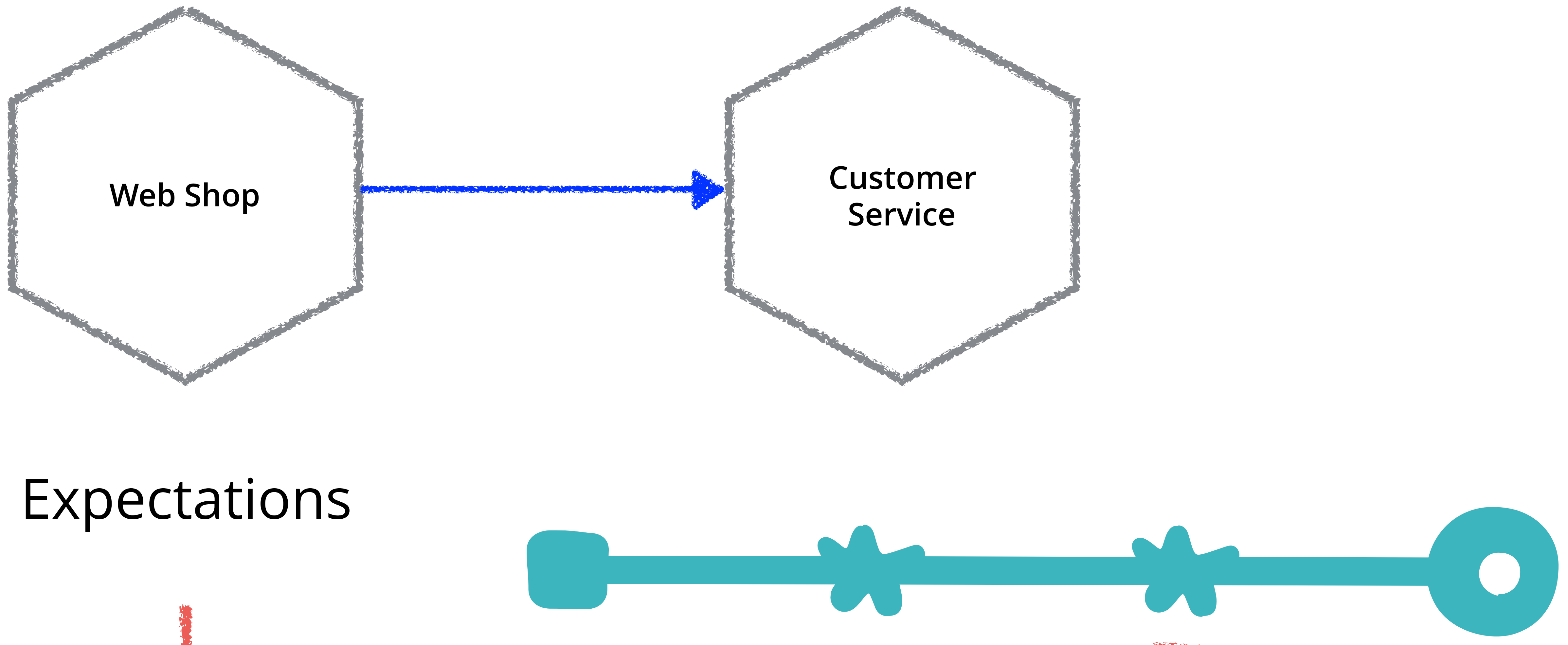


Expectations



Expectations

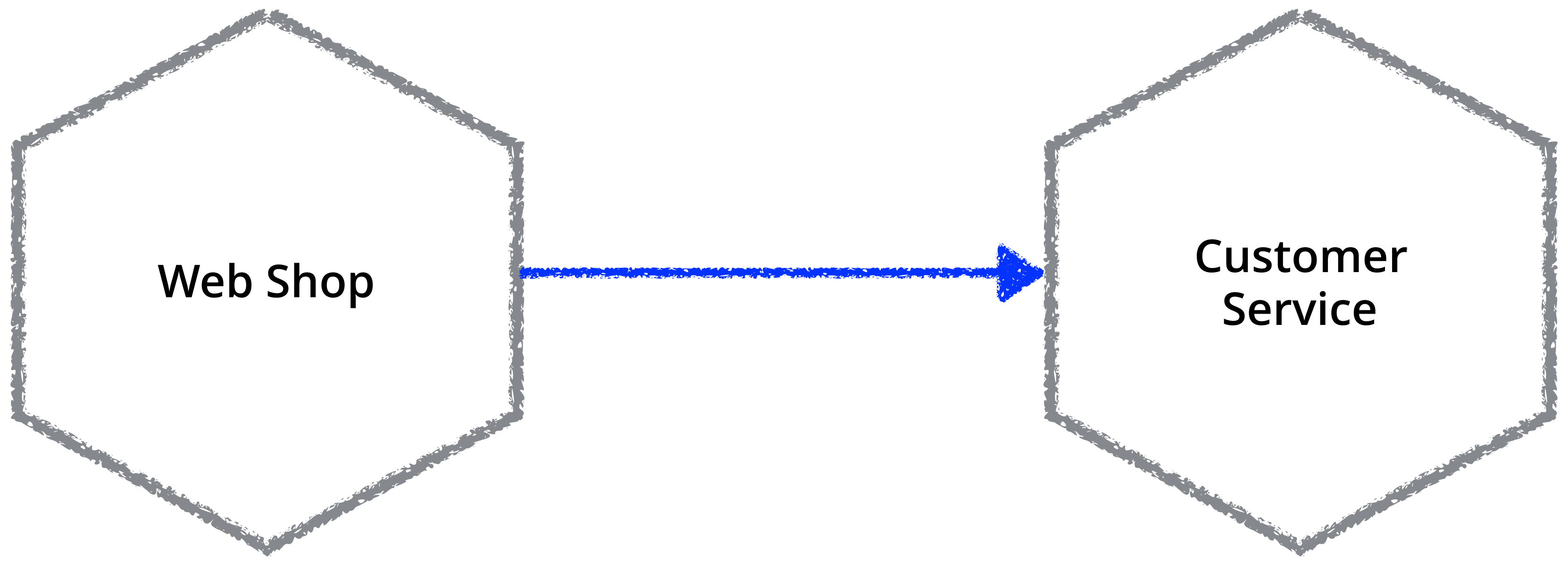
```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
          <tr valign="middle" align="center">
            <td>{ link(build.id, build) }</td>
          </tr>
        </table>
}
```



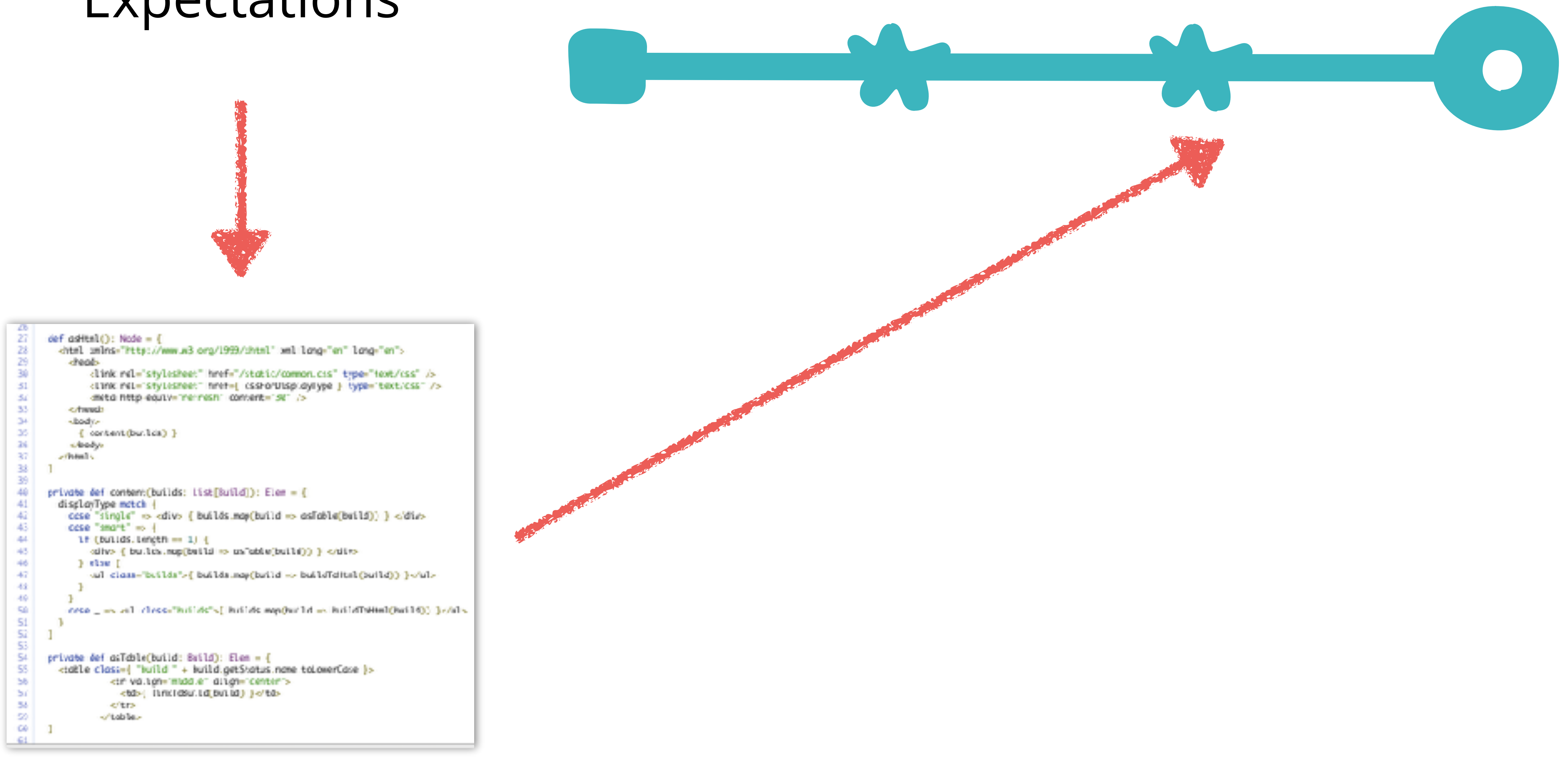
Expectations

```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name toLowerCase }>
            <tr valign="middle" align="center">
              <td>{ link(build.id, build) }</td>
            </tr>
          </table>
}
```

Consumer Driven Contracts

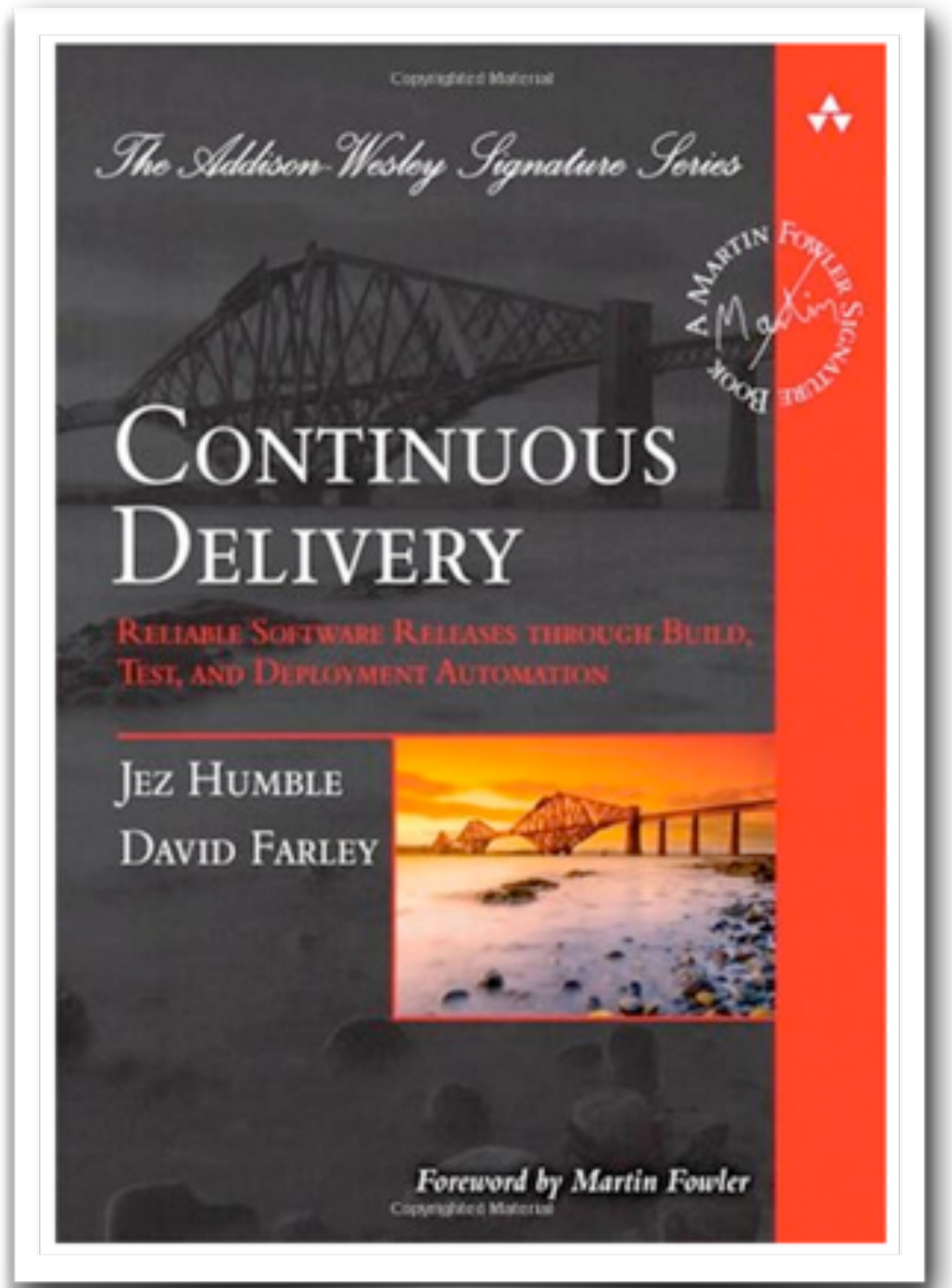


Expectations



```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssFor(displayType) } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
          <tr valign="middle" align="center">
            <td>{ link(build.id, build) }</td>
          </tr>
        </table>
}
```

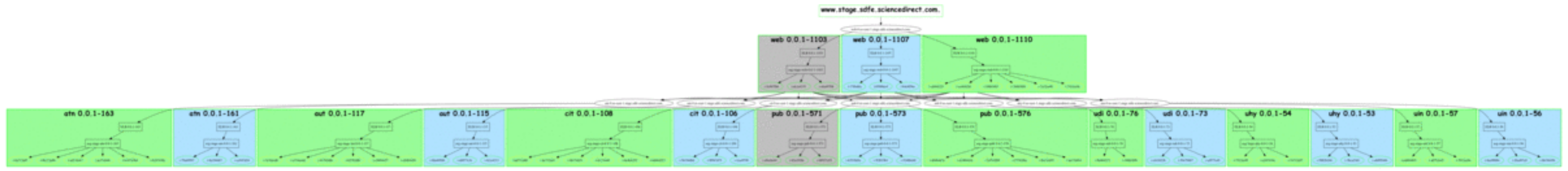
production != live

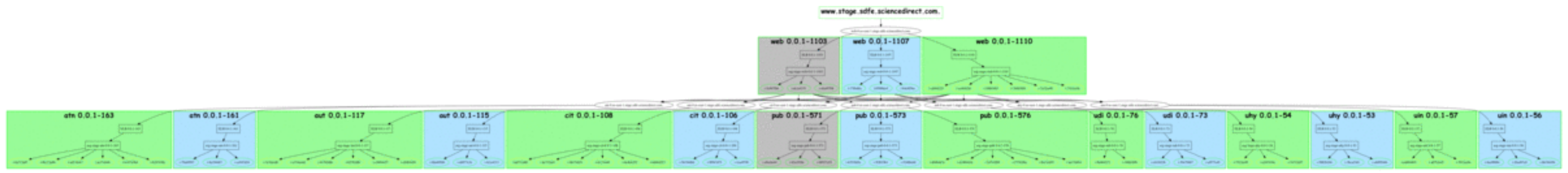

The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

blue / green deploys

canary releases

infrastructure as code


www.stage.sdfe.sciencedirect.com.
web 0.0.1-1103
web 0.0.1-1107
web 0.0.1-1110
atn 0.0.1-163
atn 0.0.1-161
aut 0.0.1-117
aut 0.0.1-115
cit 0.0.1-108
cit 0.0.1-106
pub 0.0.1-571
pub 0.0.1-573
pub 0.0.1-576
udi 0.0.1-76
udi 0.0.1-73
uhy 0.0.1-54
uhy 0.0.1-53
uin 0.0.1-57
uin 0.0.1-56


www.stage.sdfe.sciencedirect.com.
web 0.0.1-1103
web 0.0.1-1107
web 0.0.1-1110
atn 0.0.1-163
atn 0.0.1-161
out 0.0.1-117
out 0.0.1-115
cit 0.0.1-108
cit 0.0.1-106
pub 0.0.1-571
pub 0.0.1-573
pub 0.0.1-576
udi 0.0.1-76
udi 0.0.1-73
uhy 0.0.1-54
uhy 0.0.1-53
uin 0.0.1-57
uin 0.0.1-56

Plus, my hypothesis* is that you can use organisational boundaries to reason about which testing patterns to apply and which integration patterns to use

**Disclaimer IANAS*

The “chunking up from microservices to teams to value streams to lines of business to organisations" practice onion*

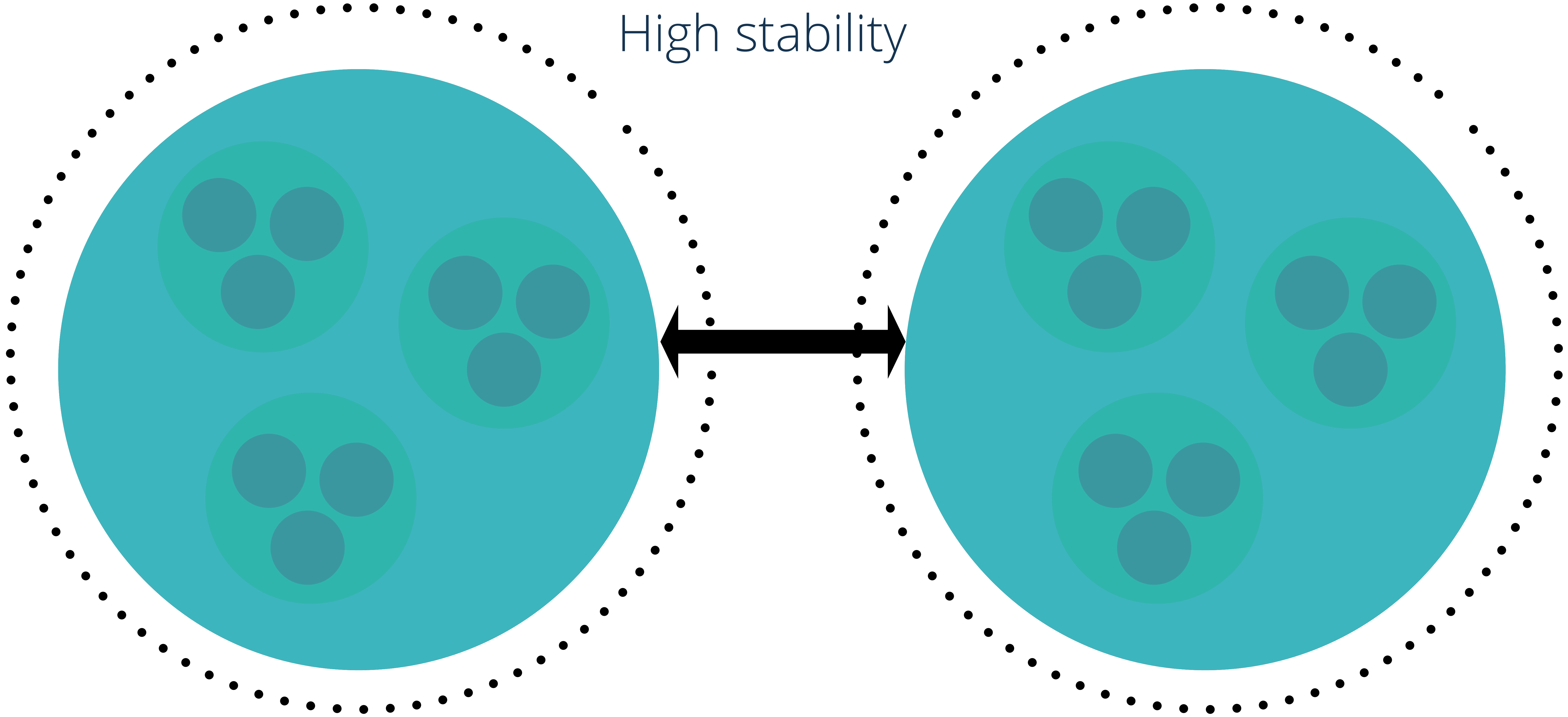
*I might need a better name for this

between organisational boundaries

typically requires:

Low change rate

High stability



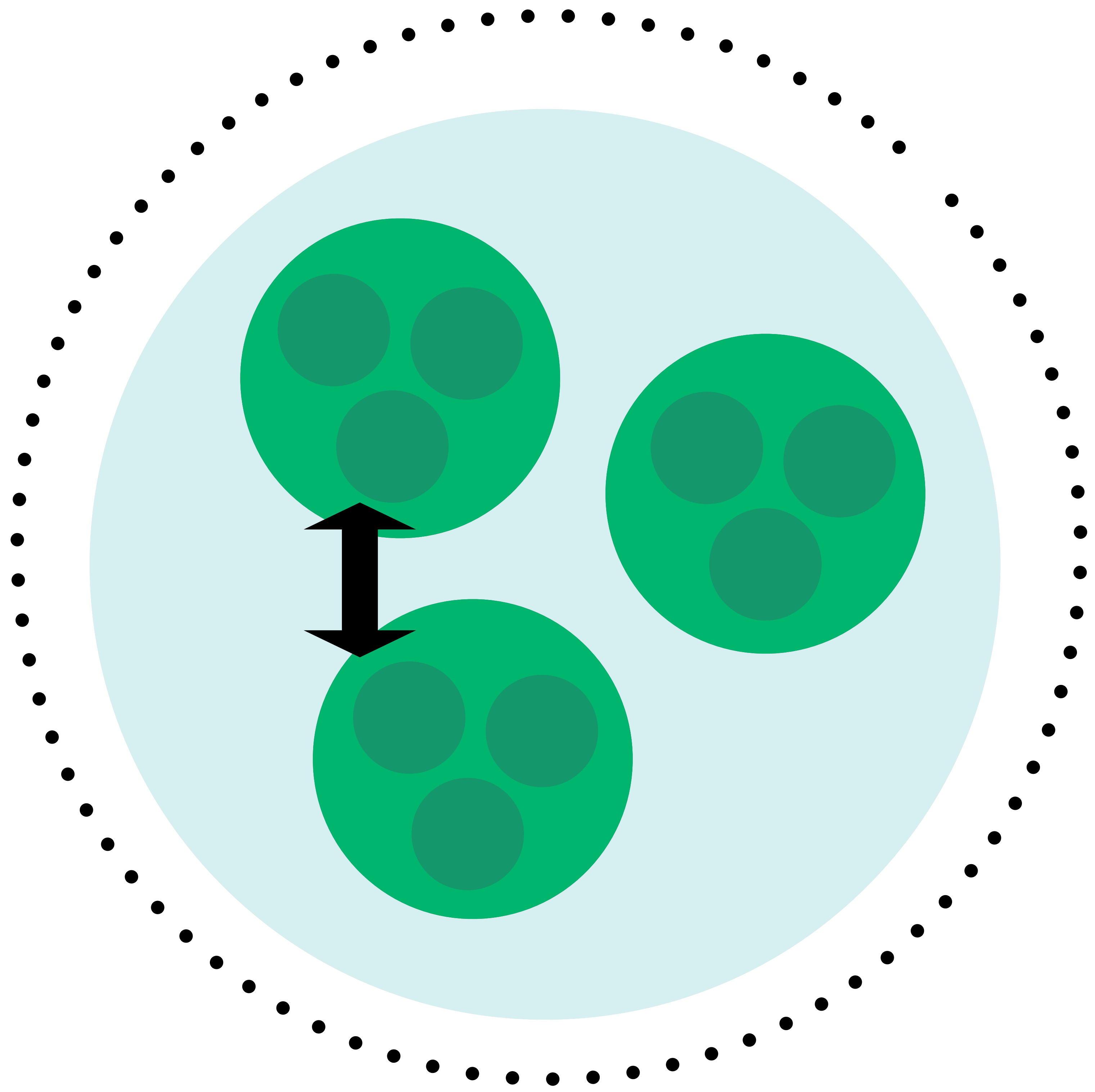
Semantic Versioning

Tolerant Reader

between business capabilities

Higher change rate

Lower stability



Semantic Versioning

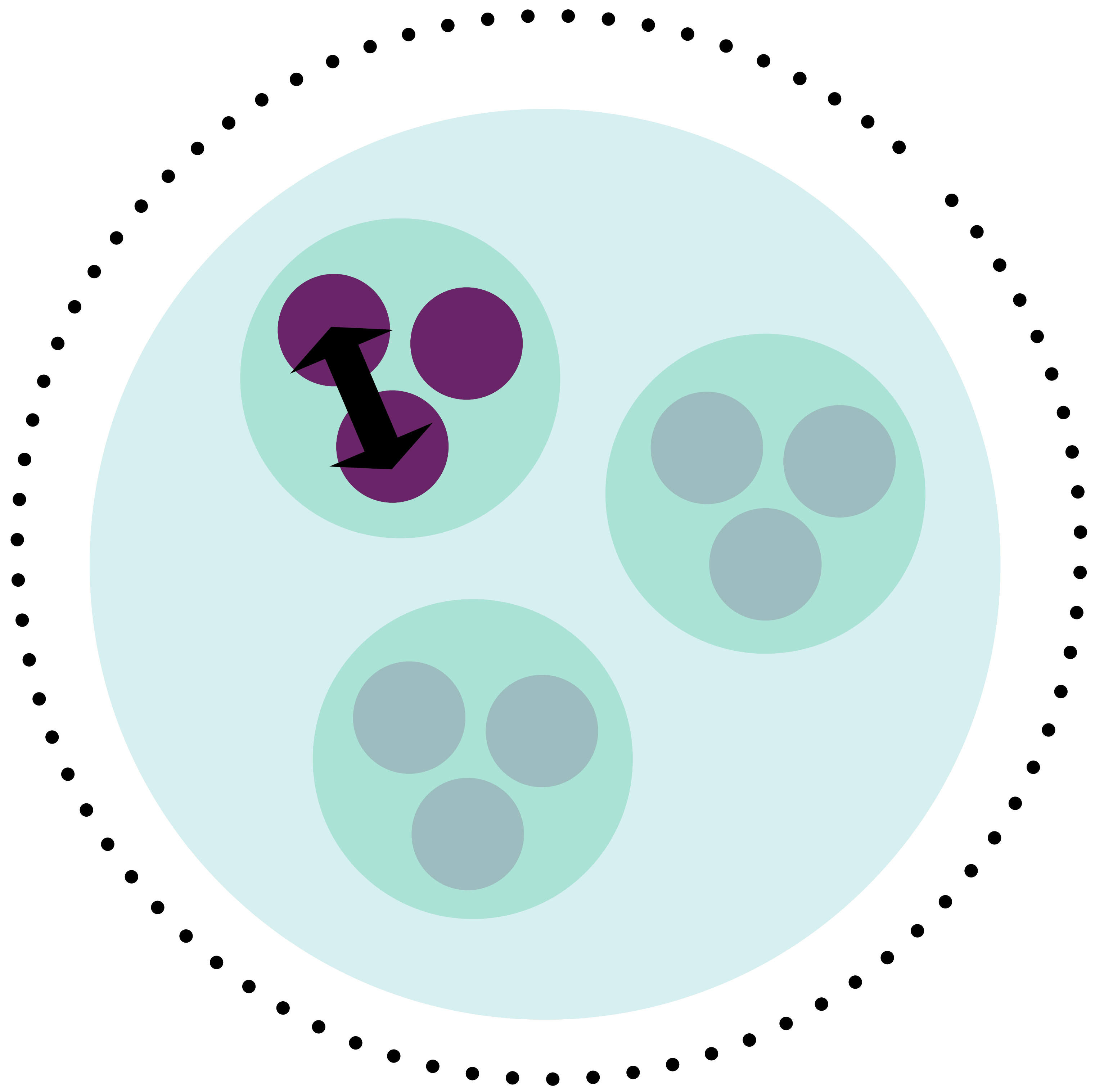
Contract Testing

Tolerant Reader

between teams

Higher rate of change

Lower stability



Semantic Versioning

Contract Testing

Tolerant Reader

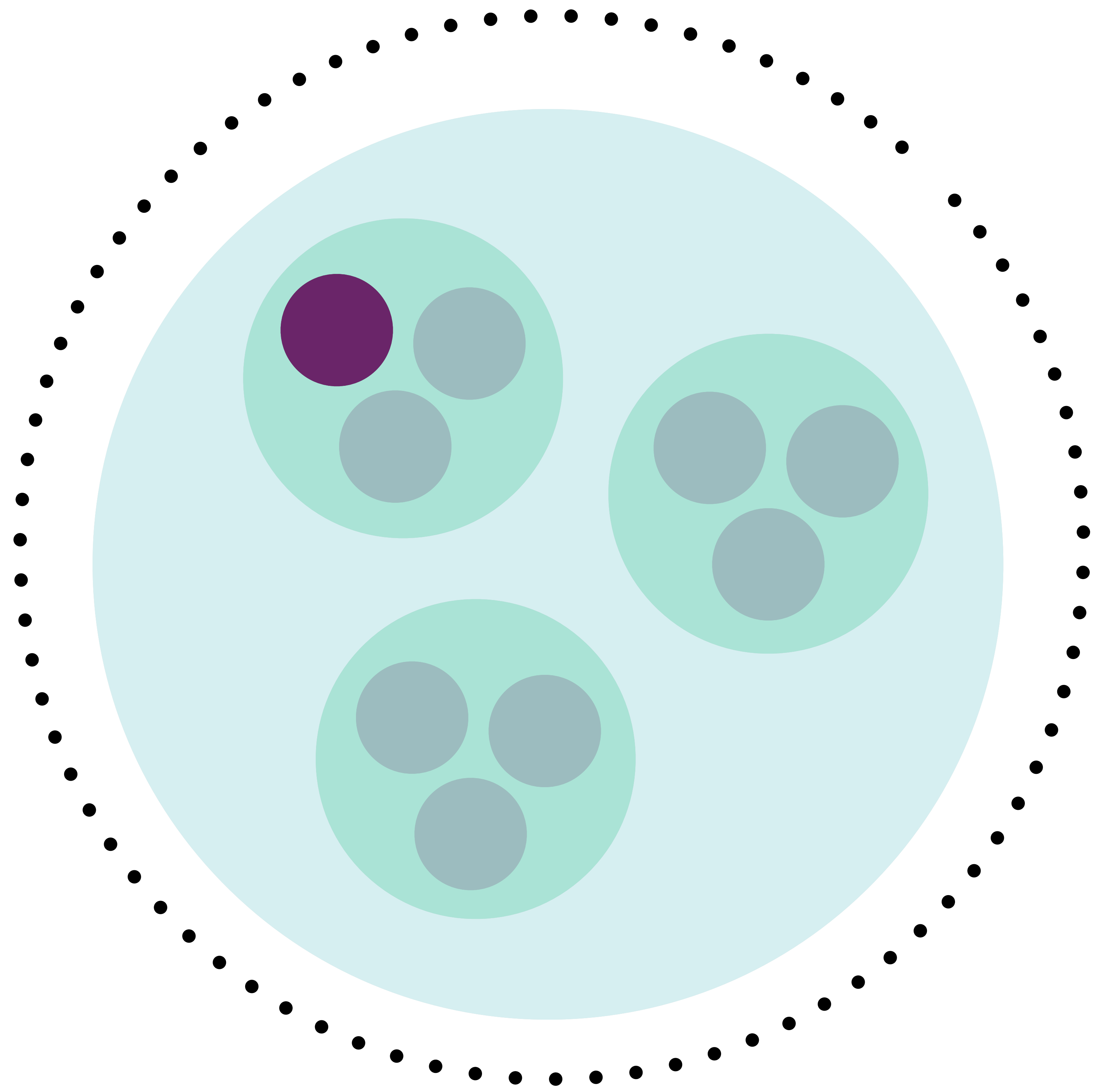
within teams

Highest rate of change

Lower stability

Conversational change

Tolerant Reader



Part the Ninth

The hunting of the snark!

"It's a Snark!" was the sound that first came to their ears,
And seemed almost too good to be true.
Then followed a torrent of laughter and cheers:
Then the ominous words "It's a Boo—"

microservices are not just about componentisation

- *componentisation via services*
- *organised around business capabilities*
- *decentralised data management*
- *products not projects*
- *decentralised governance*
- *smart endpoints and dumb pipes*
- *evolutionary design*
- *infrastructure automation*
- *designed for failure*

the characteristics may seem familiar?

1. Rule of **Modularity**: Write simple parts connected by clean interfaces.
2. Rule of **Clarity**: Clarity is better than cleverness.
3. Rule of **Composition**: Design programs to be connected to other programs.
4. Rule of **Separation**: Separate policy from mechanism; separate interfaces from engines.
5. Rule of **Simplicity**: Design for simplicity; add complexity only where you must.
6. Rule of **Parsimony**: Write a big program only when it is clear by demonstration that nothing else will do.
7. Rule of **Transparency**: Design for visibility to make inspection and debugging easier.
8. Rule of **Robustness**: Robustness is the child of transparency and simplicity.
9. Rule of **Representation**: Fold knowledge into data so program logic can be stupid and robust.
10. Rule of **Least Surprise**: In interface design, always do the least surprising thing.
11. Rule of **Silence**: When a program has nothing surprising to say, it should say nothing.
12. Rule of **Repair**: When you must fail, fail noisily and as soon as possible.
13. Rule of **Economy**: Programmer time is expensive; conserve it in preference to machine time.
14. Rule of **Generation**: Avoid hand-hacking; write programs to write programs when you can.
15. Rule of **Optimization**: Prototype before polishing. Get it working before you optimize it.
16. Rule of **Diversity**: Distrust all claims for “one true way”.
17. Rule of **Extensibility**: Design for the future, because it will be here sooner than you think.

The Art of Unix Programming *https://g.co/kgs/S960WG*

What do we need when hunting the Snark?

What do we need when hunting the Snark?

Business and Architecture Isomorphism

What do we need when hunting the Snark?

Business and Architecture Isomorphism

Infrastructure as a Service / Phoenix

What do we need when hunting the Snark?

Business and Architecture Isomorphism

Infrastructure as a Service / Phoenix

Continuous Delivery and deployment

What do we need when hunting the Snark?

Business and Architecture Isomorphism

Infrastructure as a Service / Phoenix

Continuous Delivery and deployment

System designs that support rapid change

What do we need when hunting the Snark?

Business and Architecture Isomorphism

Infrastructure as a Service / Phoenix

Continuous Delivery and deployment

System designs that support rapid change

Comfortable with ambiguity of evolutionary architecture

What do we need when hunting the Snark?

Business and Architecture Isomorphism

Infrastructure as a Service / Phoenix

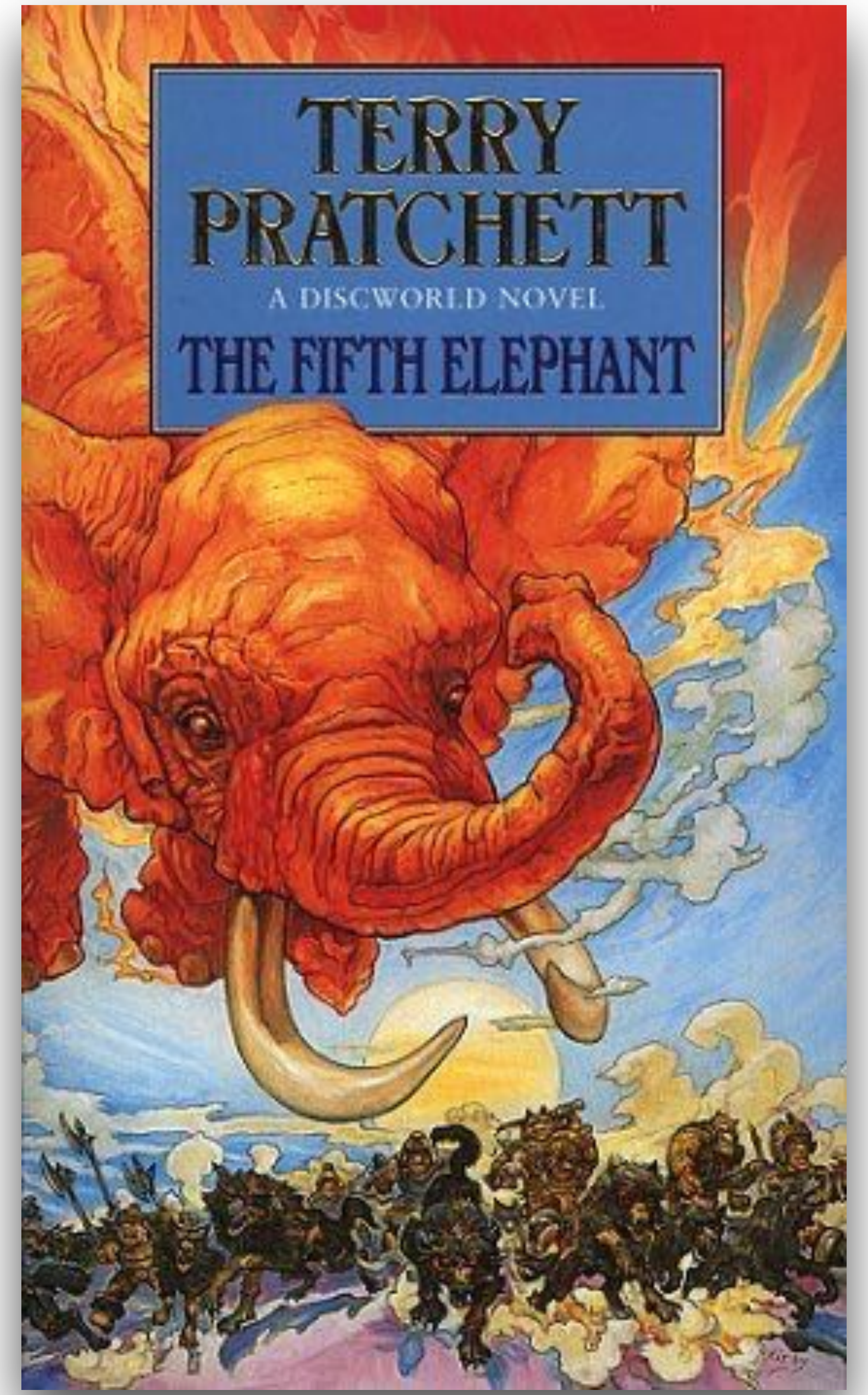
Continuous Delivery and deployment

System designs that support rapid change

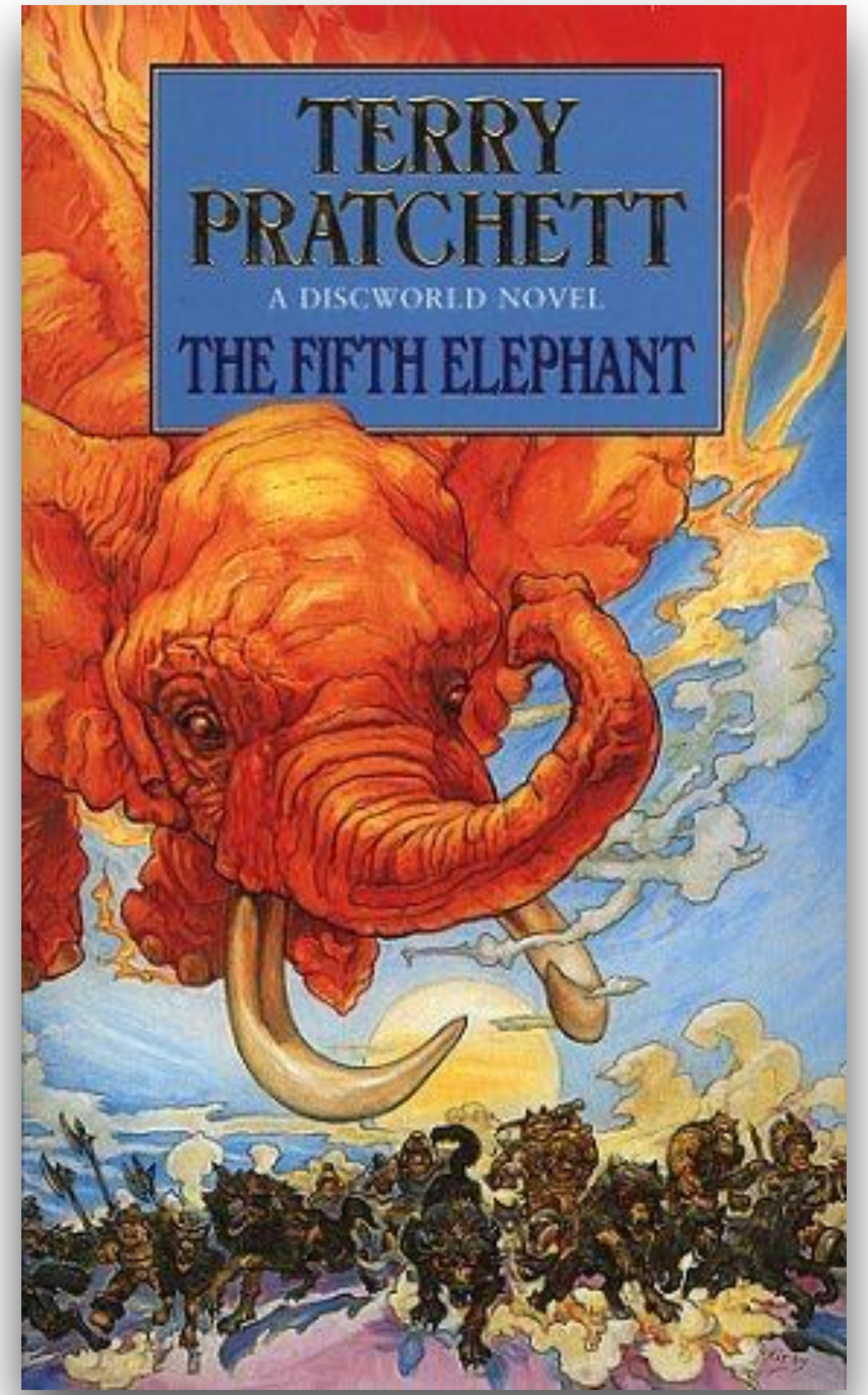
Comfortable with ambiguity of evolutionary architecture

QA in prod / rapid remediation / semantic monitoring

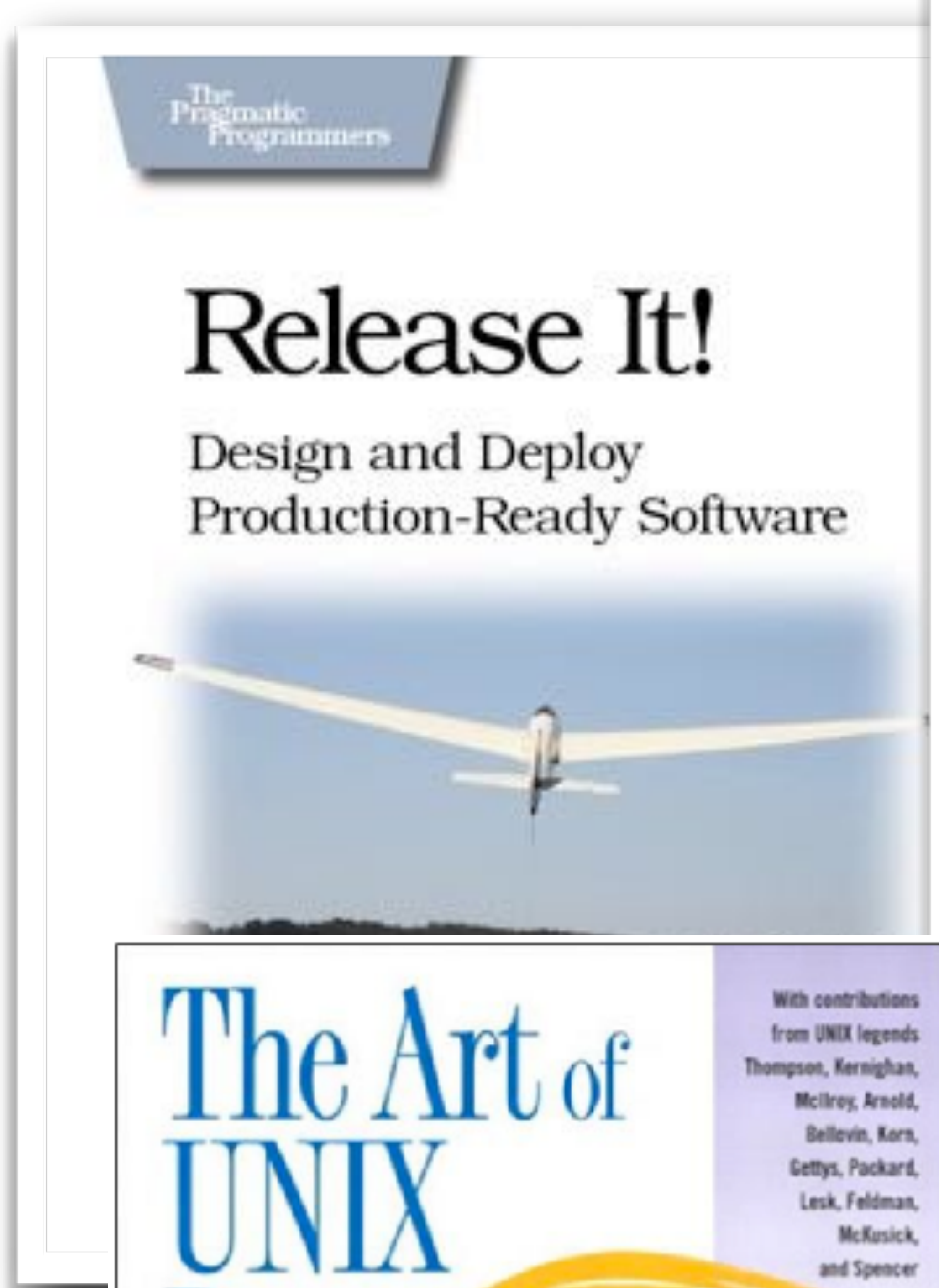
never done



never done



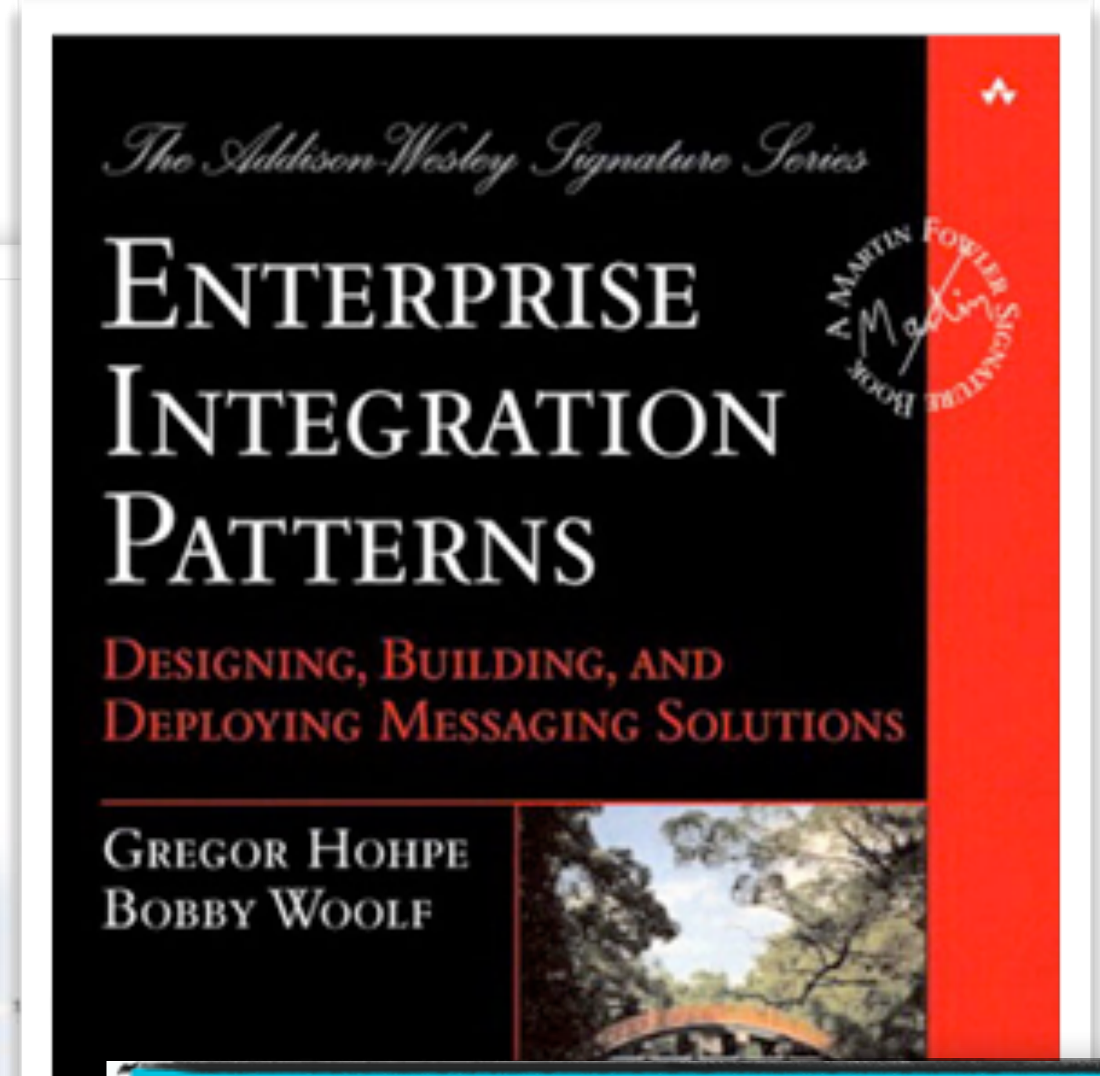
“This, milord, is my **family's axe**. We have owned it for almost nine hundred years, see. Of course, sometimes it needed a **new blade**. And sometimes it has required a **new handle**, new designs on the metalwork, a little refreshing of the ornamentation . . . but **is this not** the nine hundred-year-old axe of my family? And because it has changed gently over time, it is still a pretty good axe, y'know. Pretty good.”



The Pragmatic Programmers

Release It!

Design and Deploy Production-Ready Software



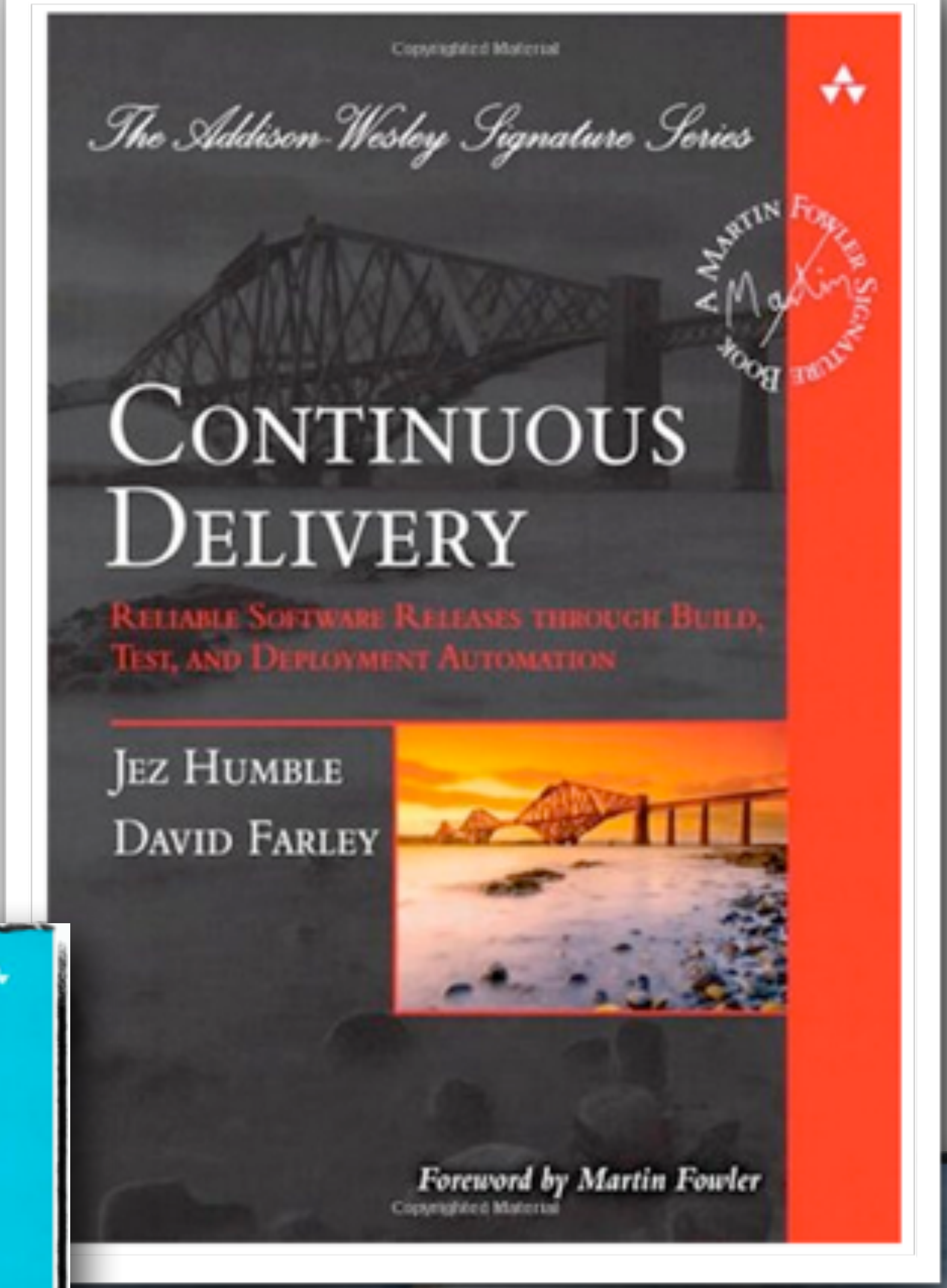
The Addison-Wesley Signature Series

ENTERPRISE INTEGRATION PATTERNS

DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS

GREGOR HOHPE
BOBBY WOOLF

A MARTIN FOWLER SIGNATURE BOOK



The Addison-Wesley Signature Series

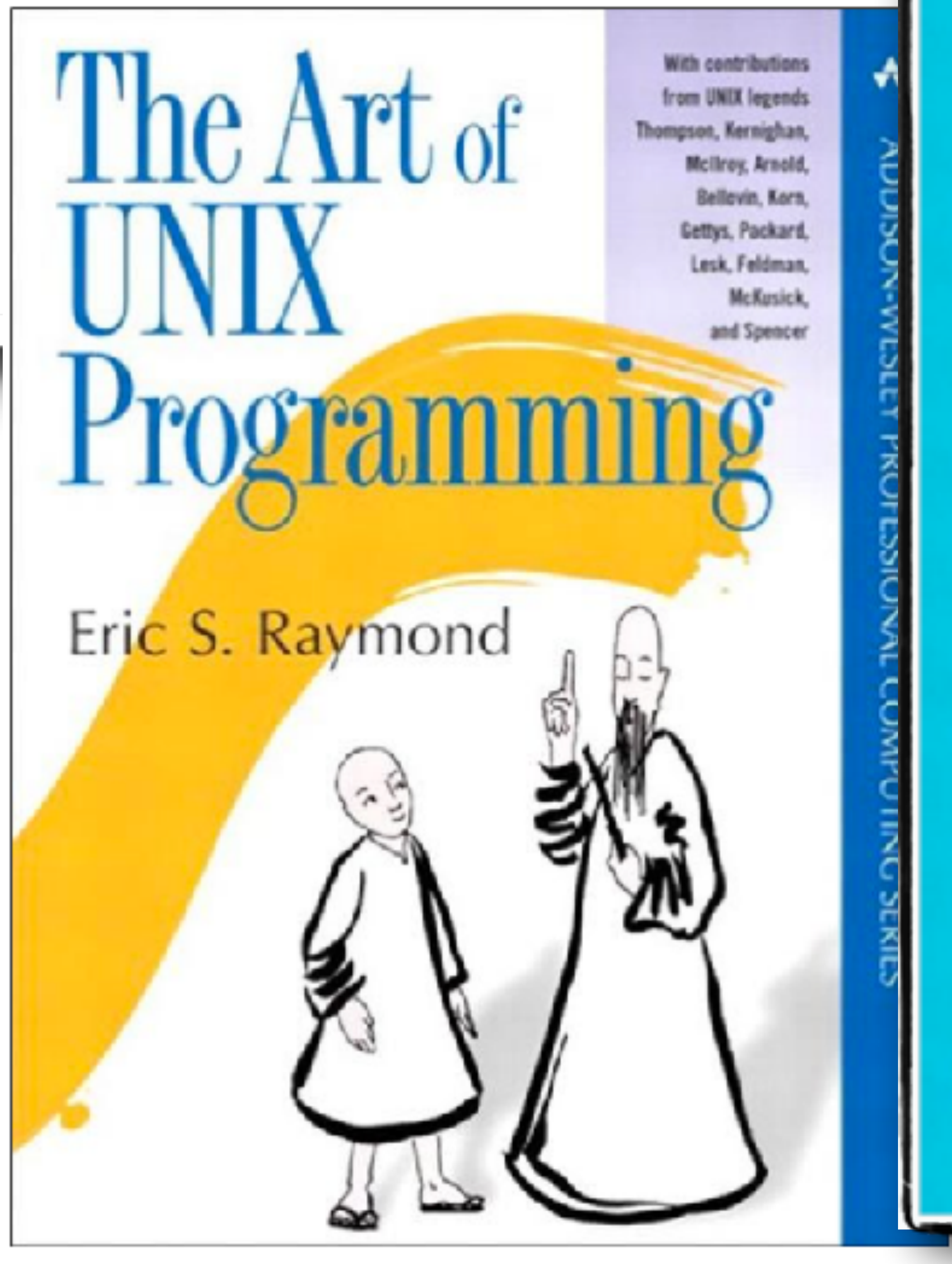
CONTINUOUS DELIVERY

RELIABLE SOFTWARE RELEASES THROUGH BUILD, TEST, AND DEPLOYMENT AUTOMATION

JEZ HUMBLE
DAVID FARLEY

A MARTIN FOWLER SIGNATURE BOOK

Foreword by Martin Fowler

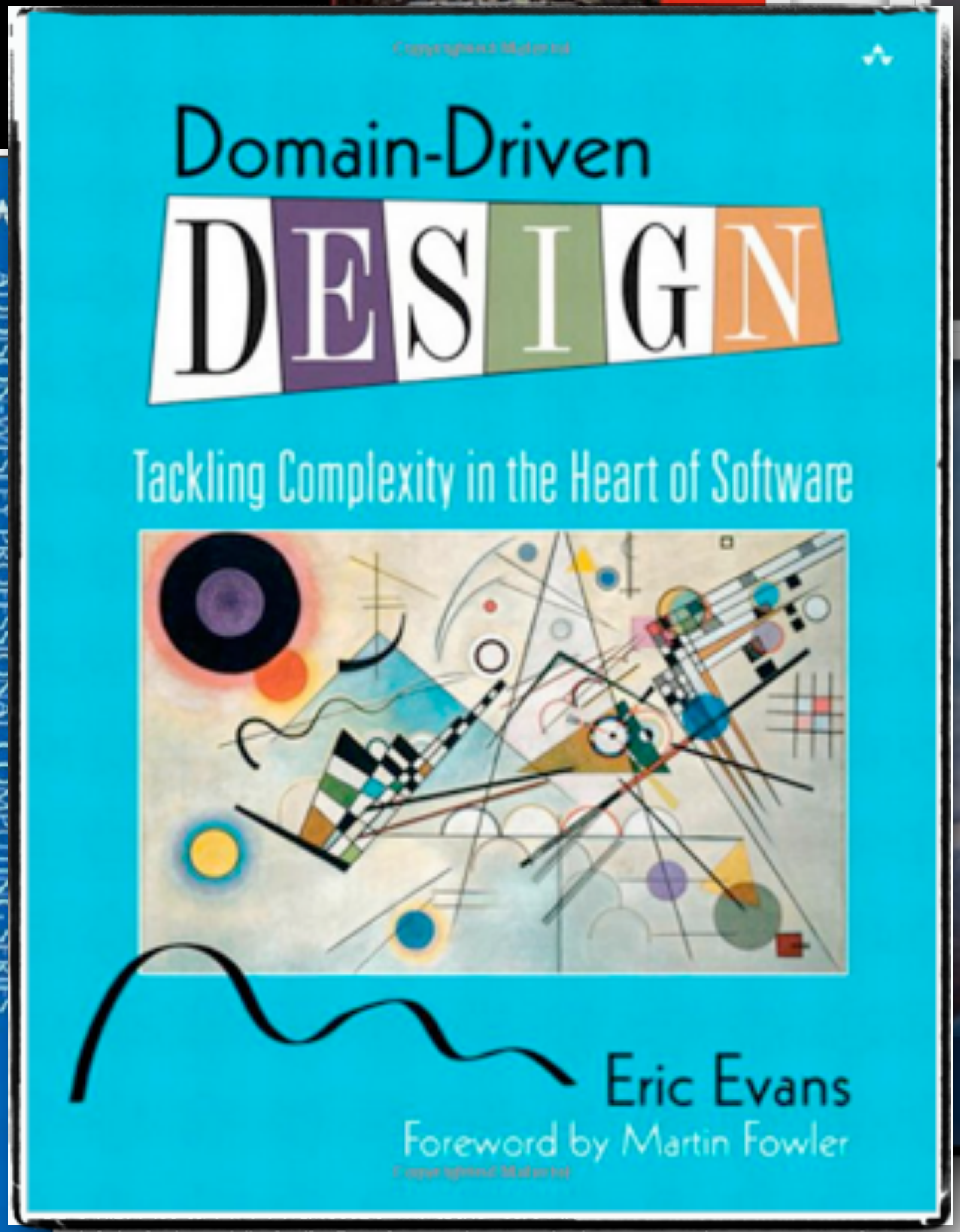


The Art of UNIX Programming

With contributions from UNIX legends Thompson, Kernighan, McIlroy, Arnold, Bellovin, Korn, Gettys, Packard, Lesk, Feldman, McKusick, and Spencer

Eric S. Raymond

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

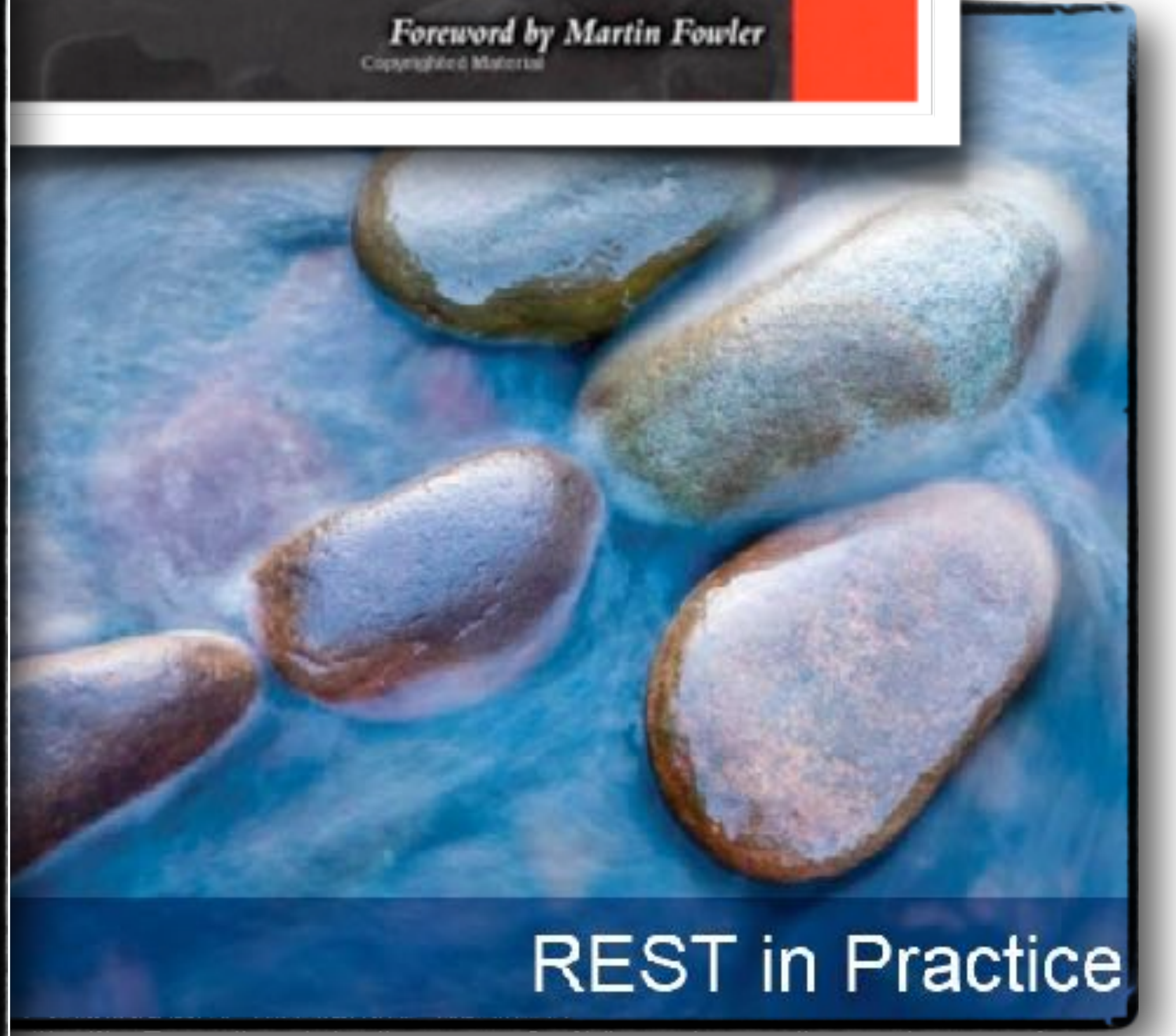


Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

Eric Evans

Foreword by Martin Fowler



REST in Practice

Part the Tenth

The vanishing

“In the midst of the word he was trying to say,
In the midst of his laughter and glee,
He had softly and suddenly vanished away—
For the Snark was a Boojum, you see.“

Always have ten parts

thanks!

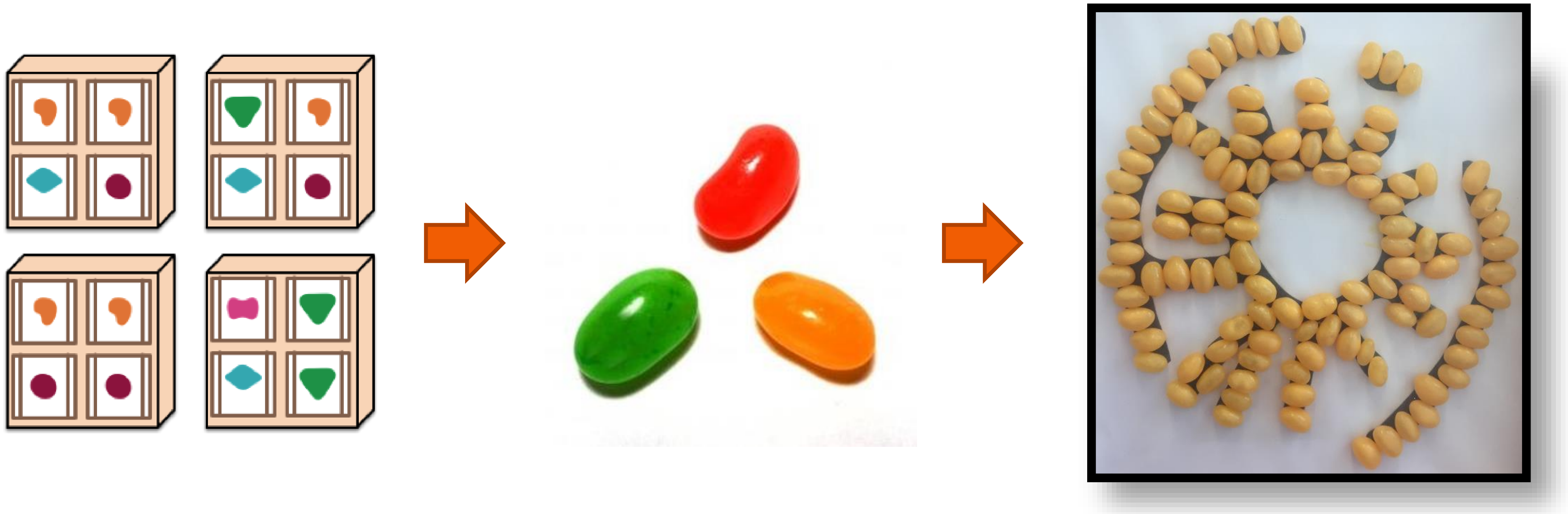
jalewis@thoughtworks.com **@boicy**

ThoughtWorks®

“Applause”

Microservices - Hot-or-Not?

Thank you!



More Hot-or-Not, more Sioux

March 8&9	**>**	**Premium Course “Microservices”**
March 21	**>**	**“Proefzitten” (open house)**
June 2016	**>**	**Hot-or-Not “Elixer”**
Q3 2016	**>**	**Hot-or-Not “Artificial Intelligence”**
Q4 2016	**>**	**Hot-or-Not The Next Generation**

Goto www.sioux.eu for more information.

DRINKS

Source of
your technology