ThoughtWorks®

microservices

THE HUNTING OF THE SNARK

Chapter 0

The “London school of software engineering”

ThoughtWorks®


CONSULTANTS!
HELL YEAH!

ThoughtWorks®

ThoughtWorks®

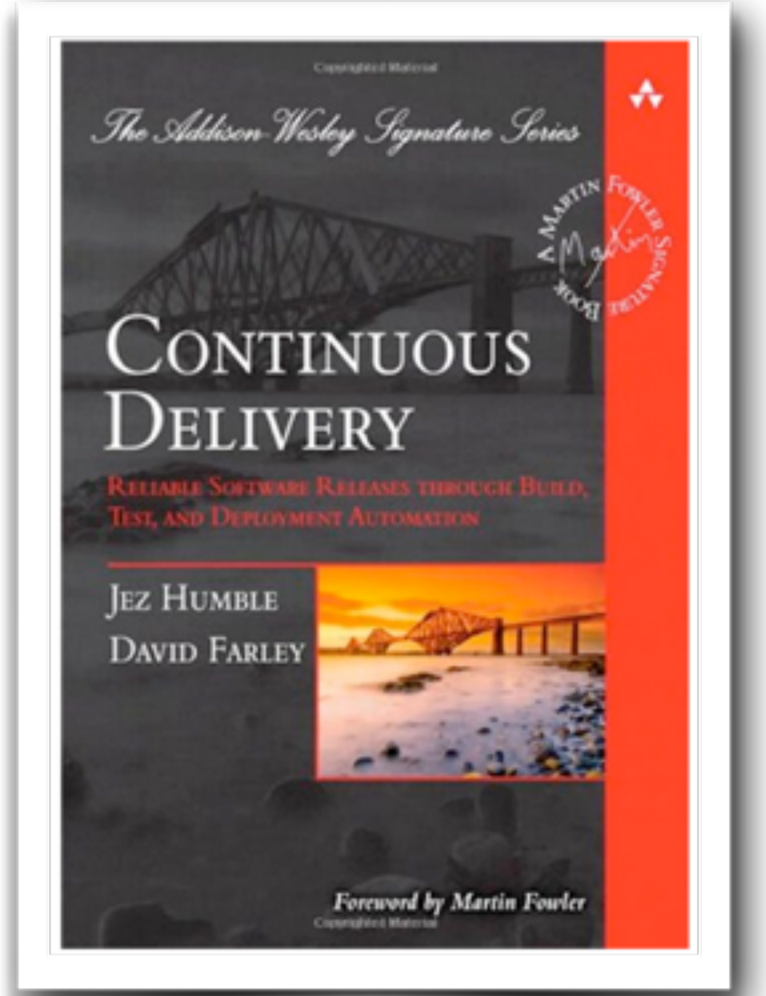
jMock

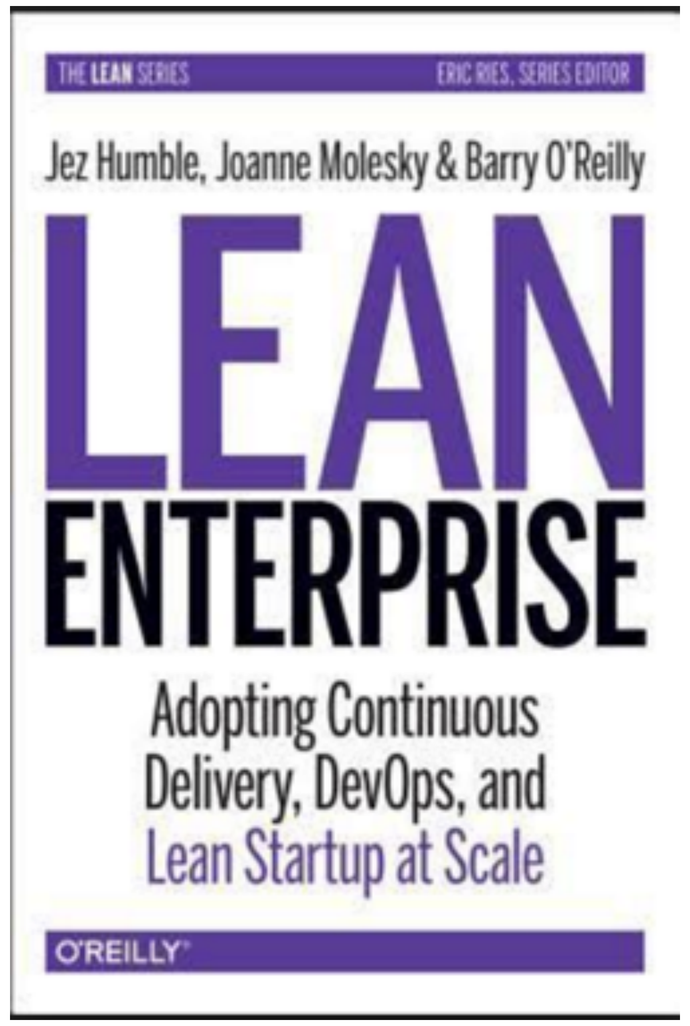
go
Continuous Delivery

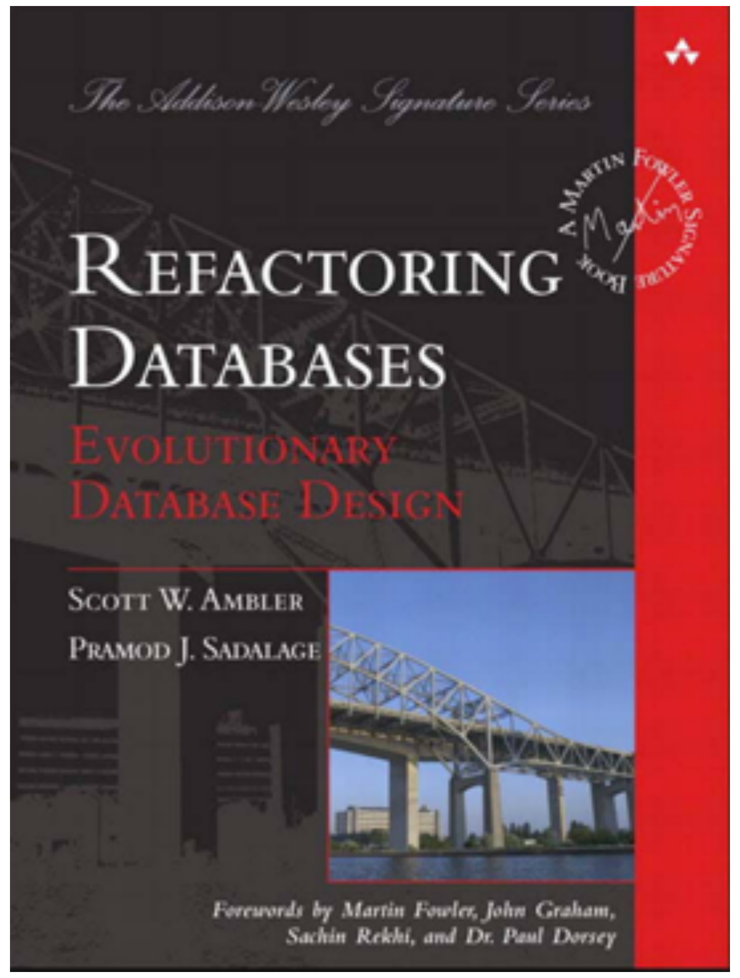
cruisecontrol

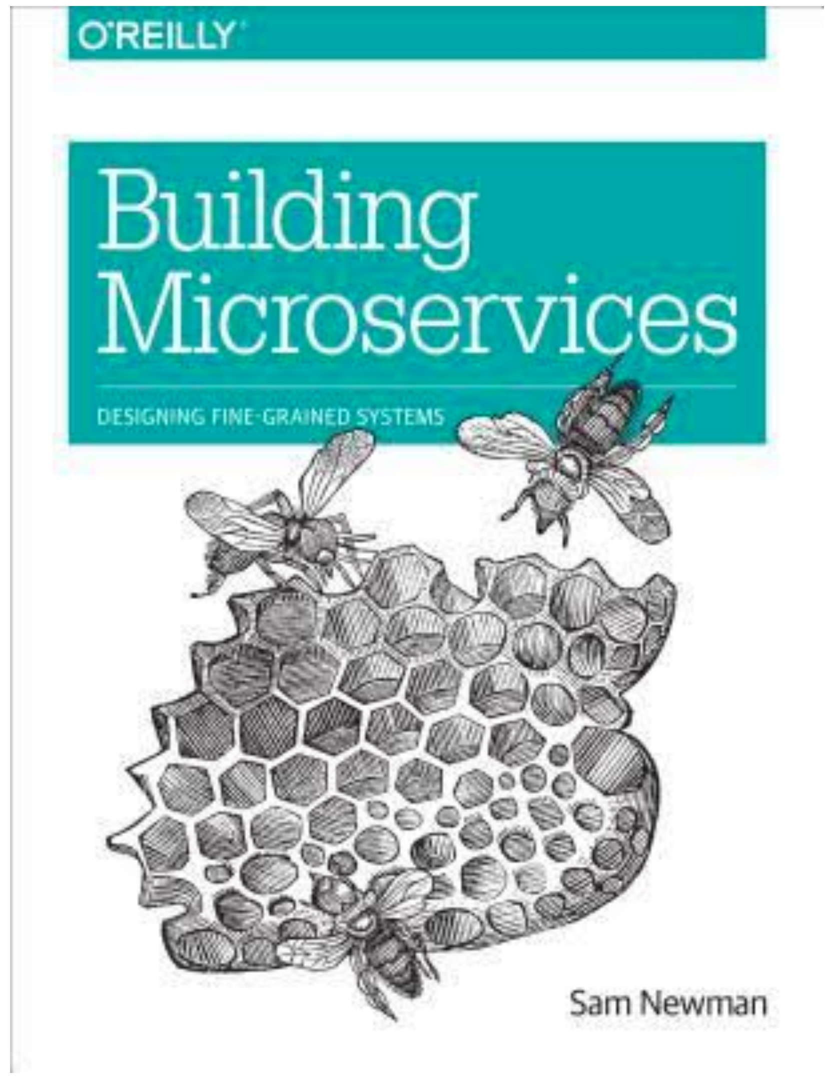
BDD

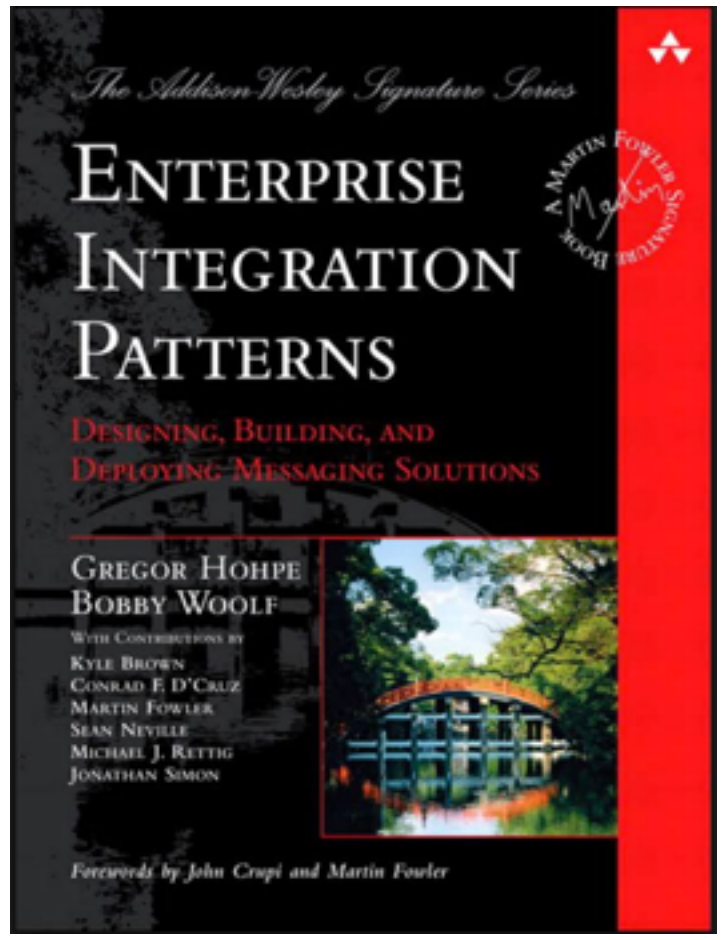
pico
container

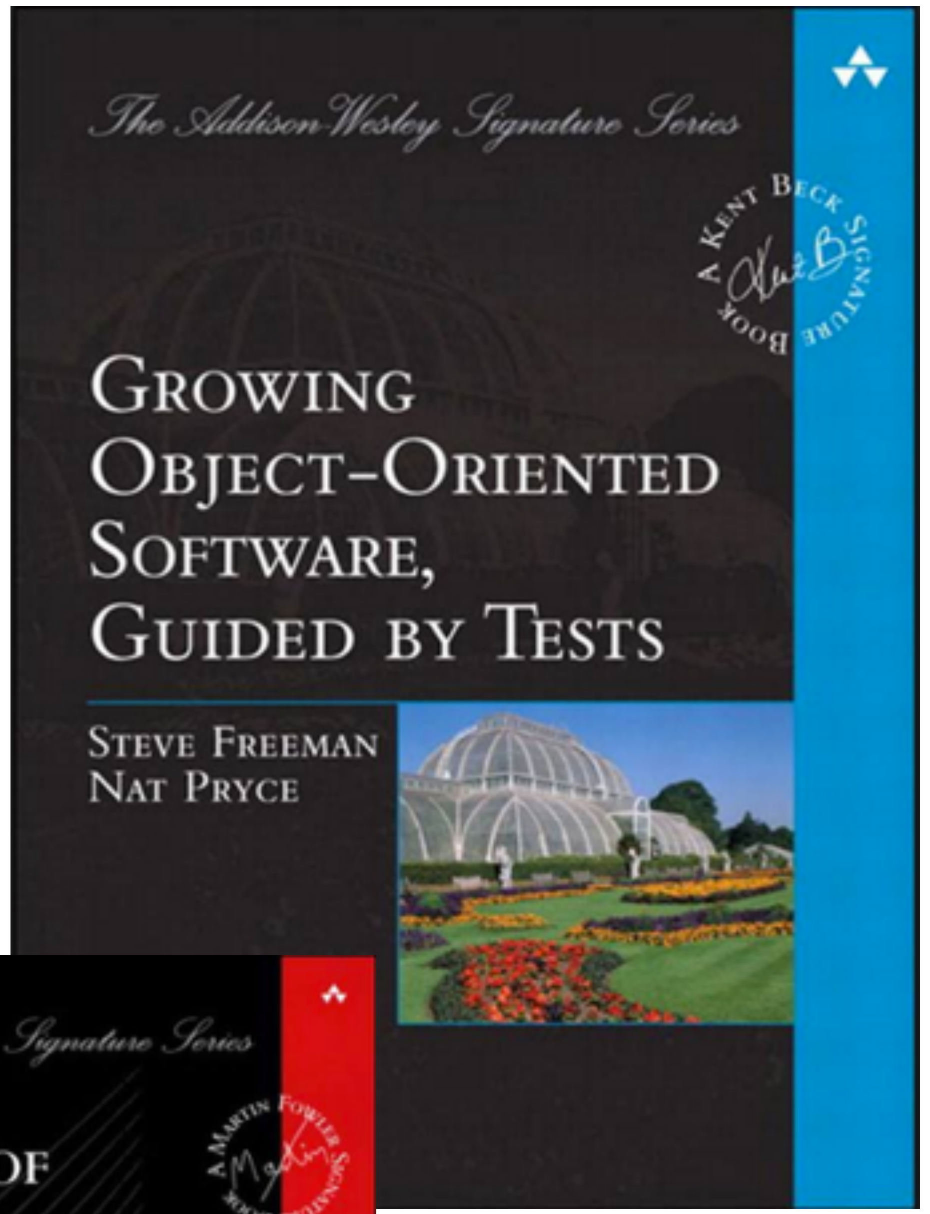


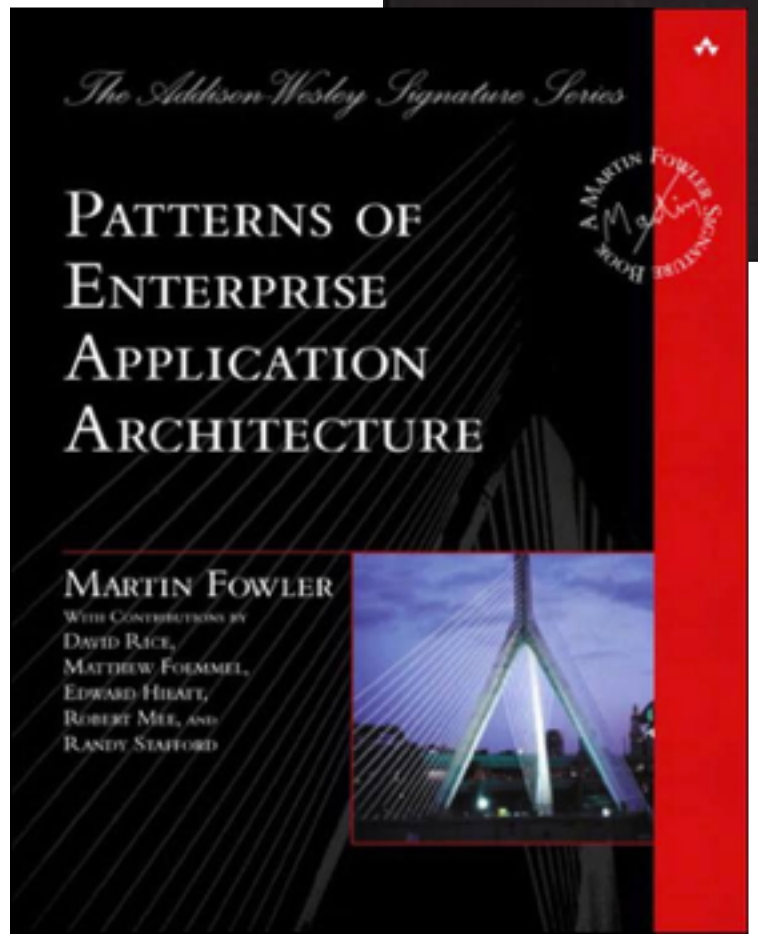












TECHNOLOGY RADAR

Search A-Z FAQs

Techniques | Tools | Platforms | Languages & Frameworks

ADOPT

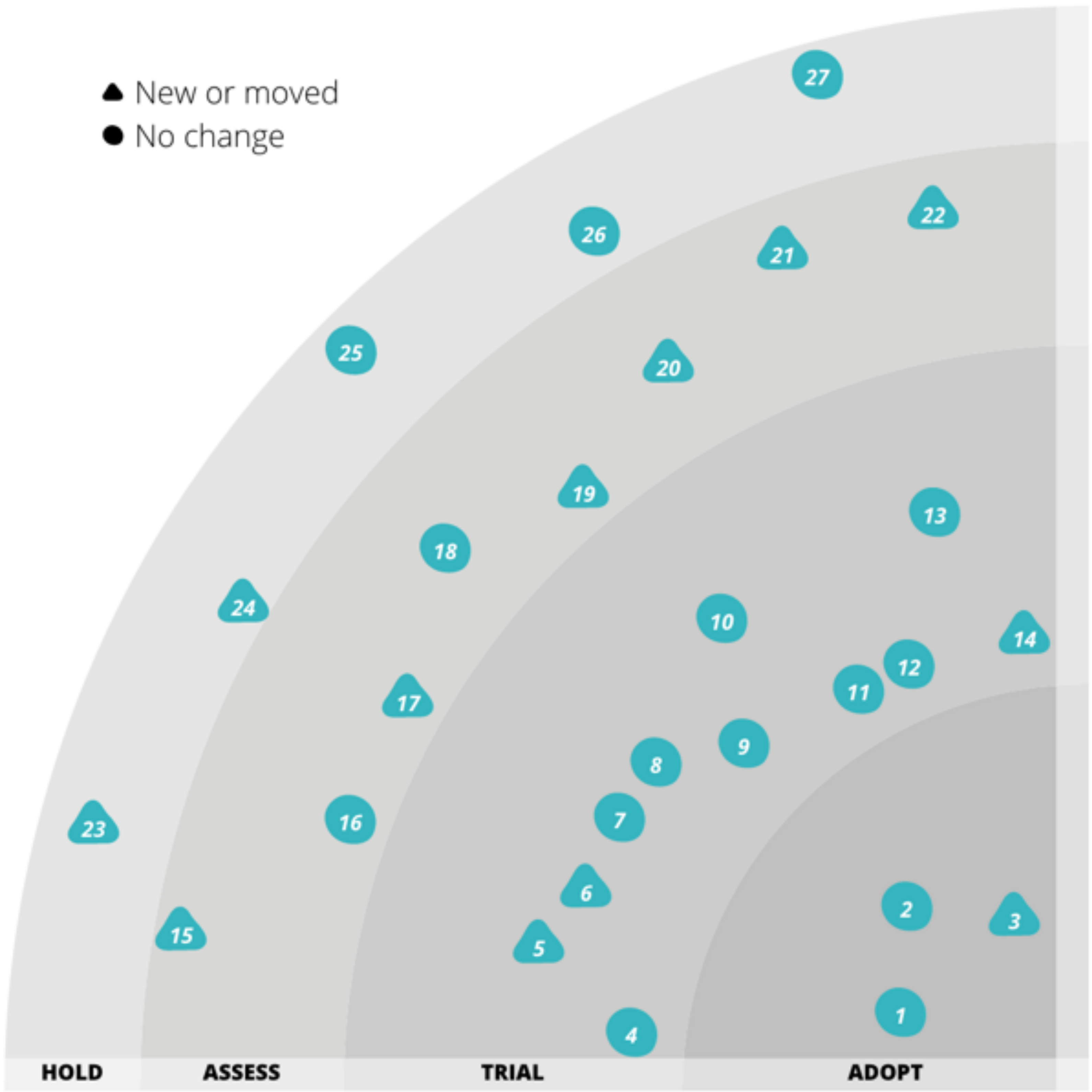
1. Decoupling deployment from release
2. Products over projects
3. Threat Modeling

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 »

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

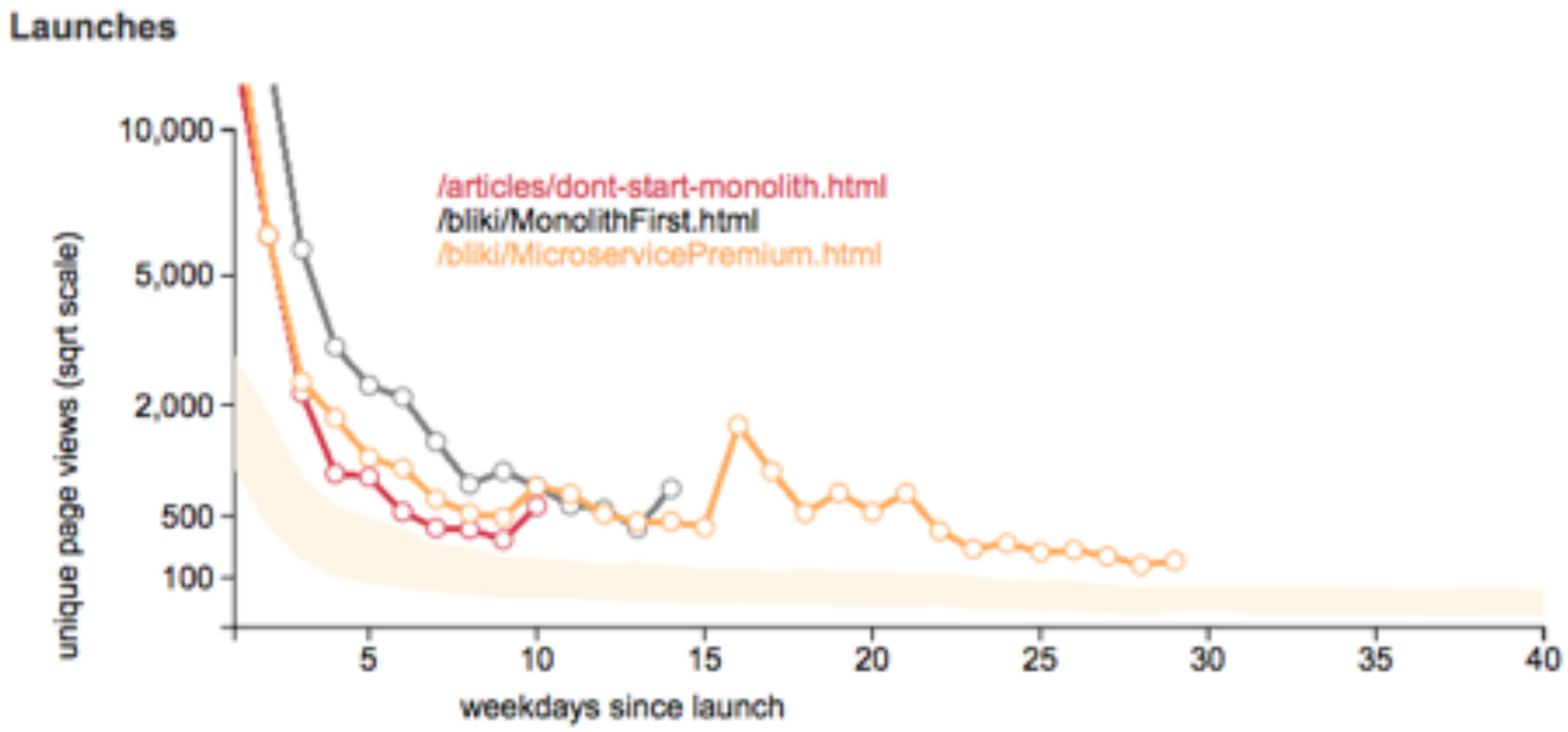
Characteristics of a Microservice Architecture
- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Are Microservices the Future?

Sidebars

How big is a microservice?
Microservices and SOA
Many languages, many options
Battle-tested standards and enforced standards
Make it easy to do the right thing
The circuit breaker and production ready code
Synchronous calls considered harmful

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



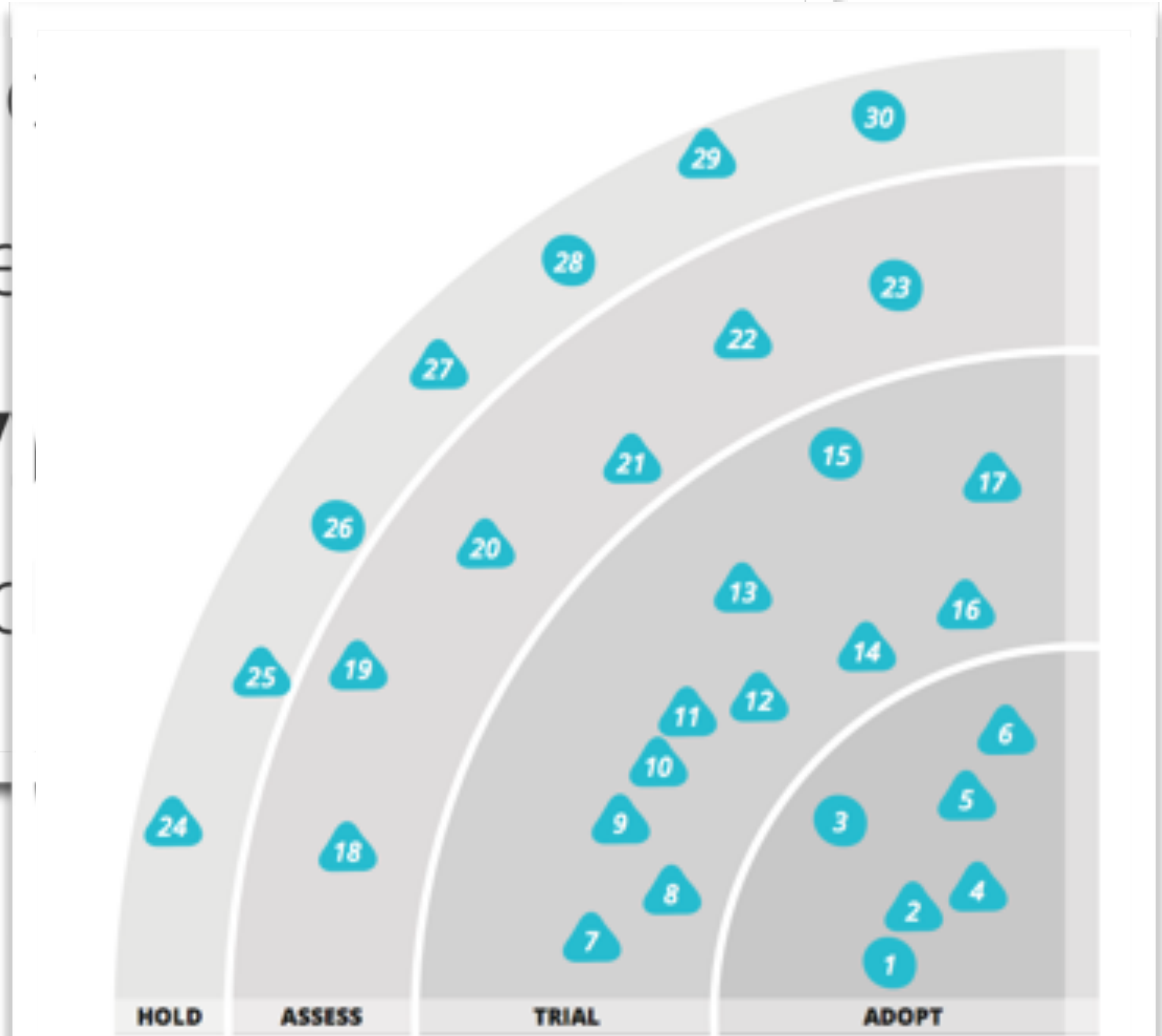
clear

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

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
Most situations do not require these kinds of
feats; teams should keep their **web scale env**
favor of simpler solutions that still get the job

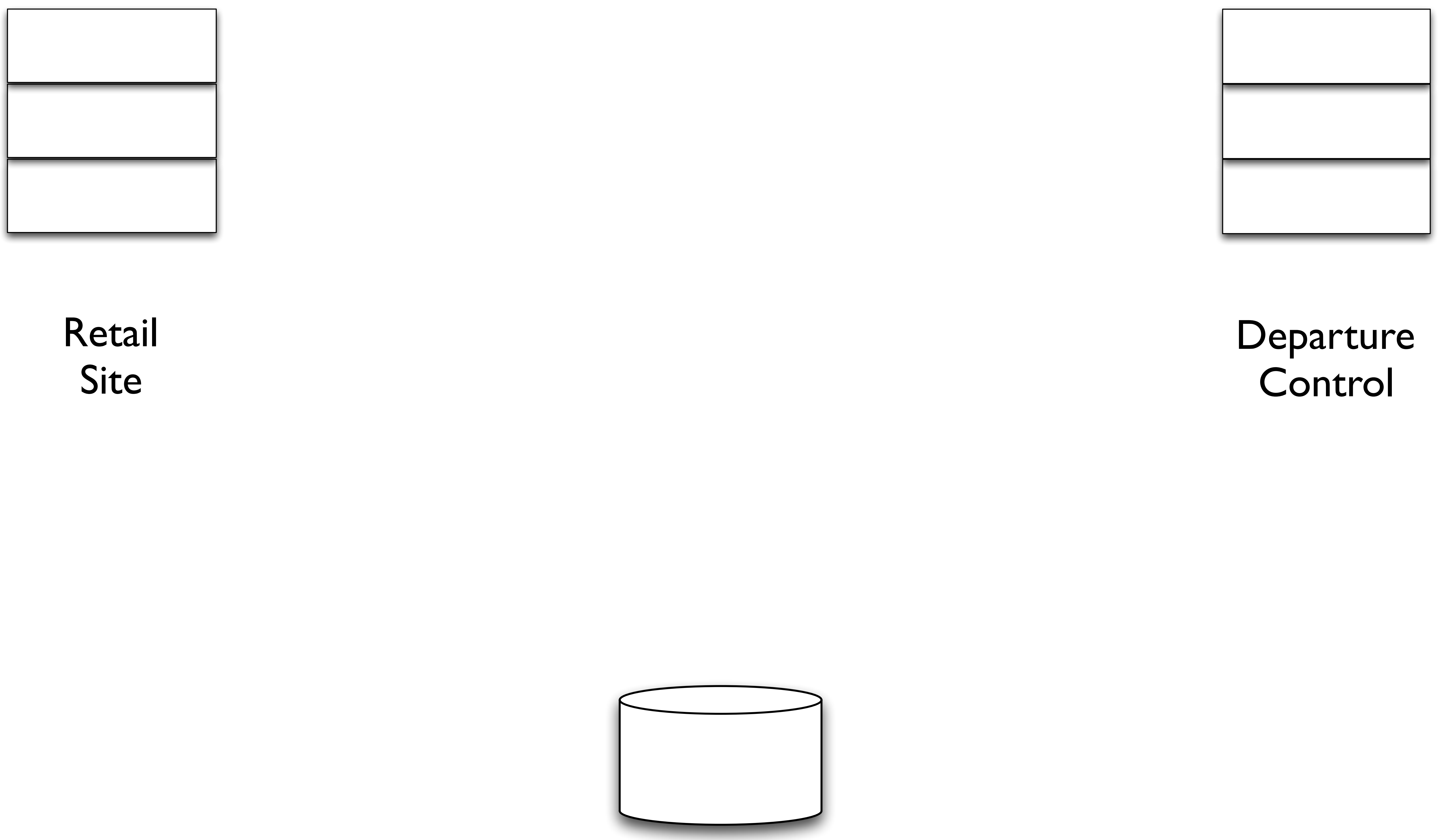


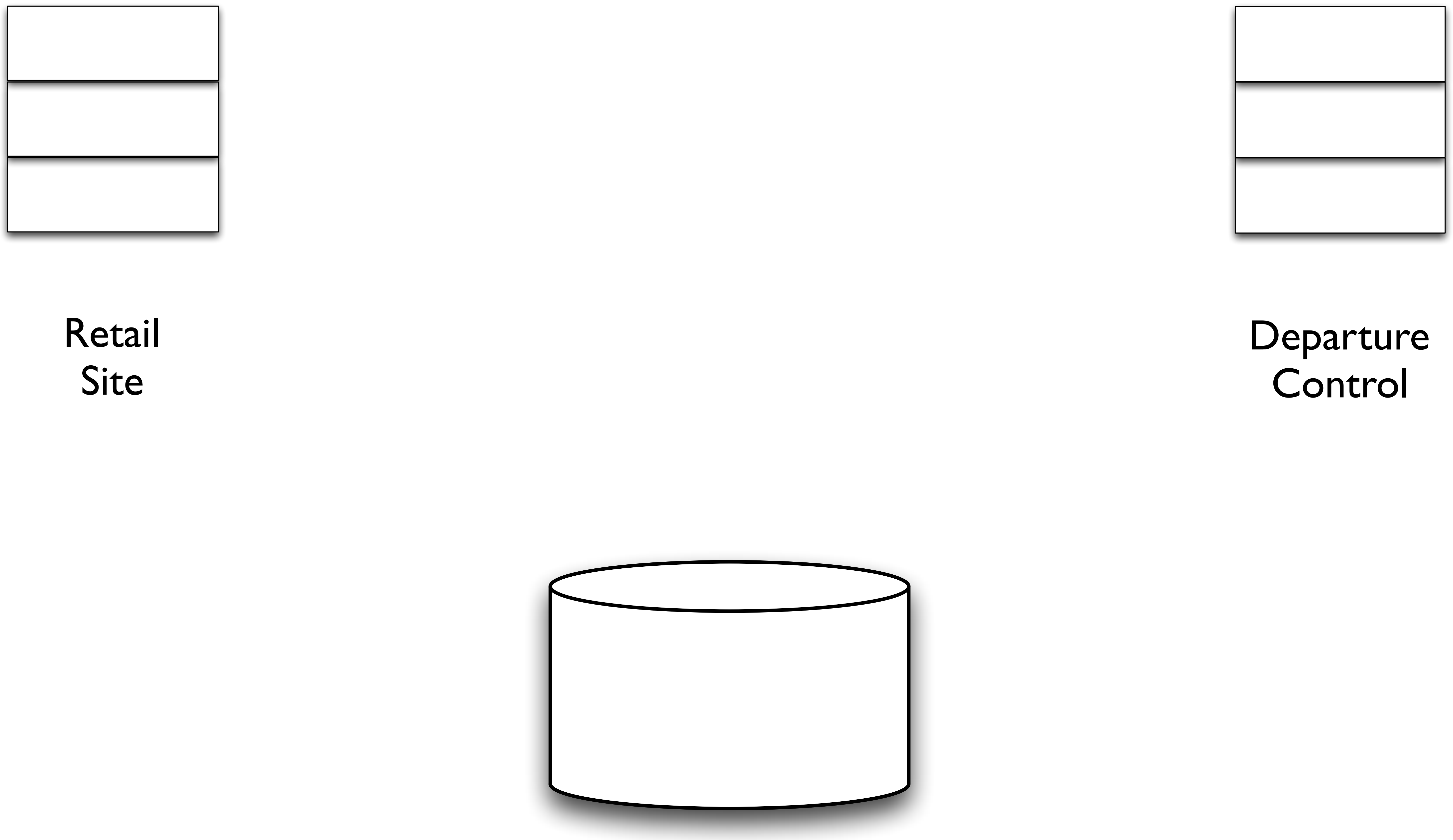
https://www.thoughtworks.com/radar/

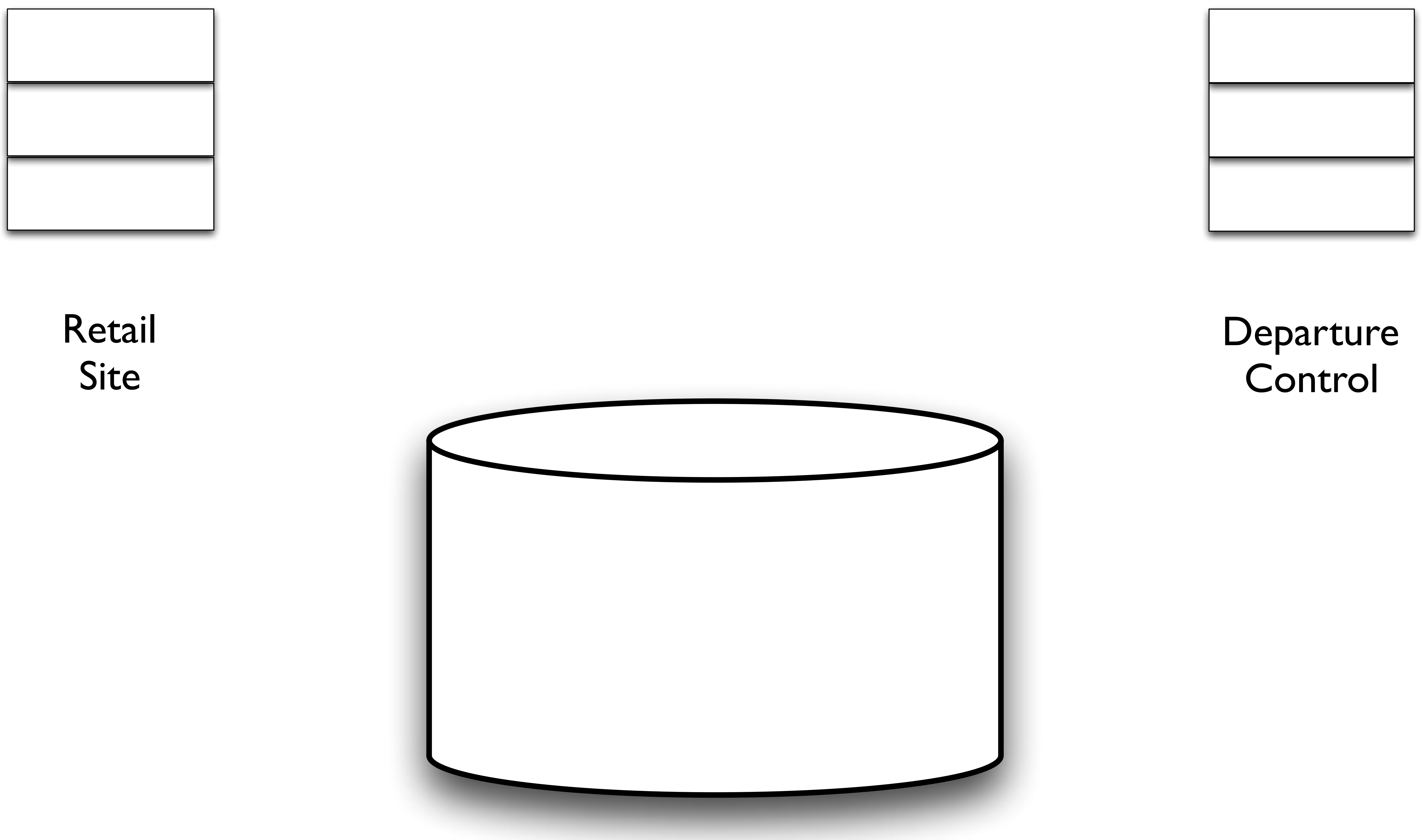
Chapter I

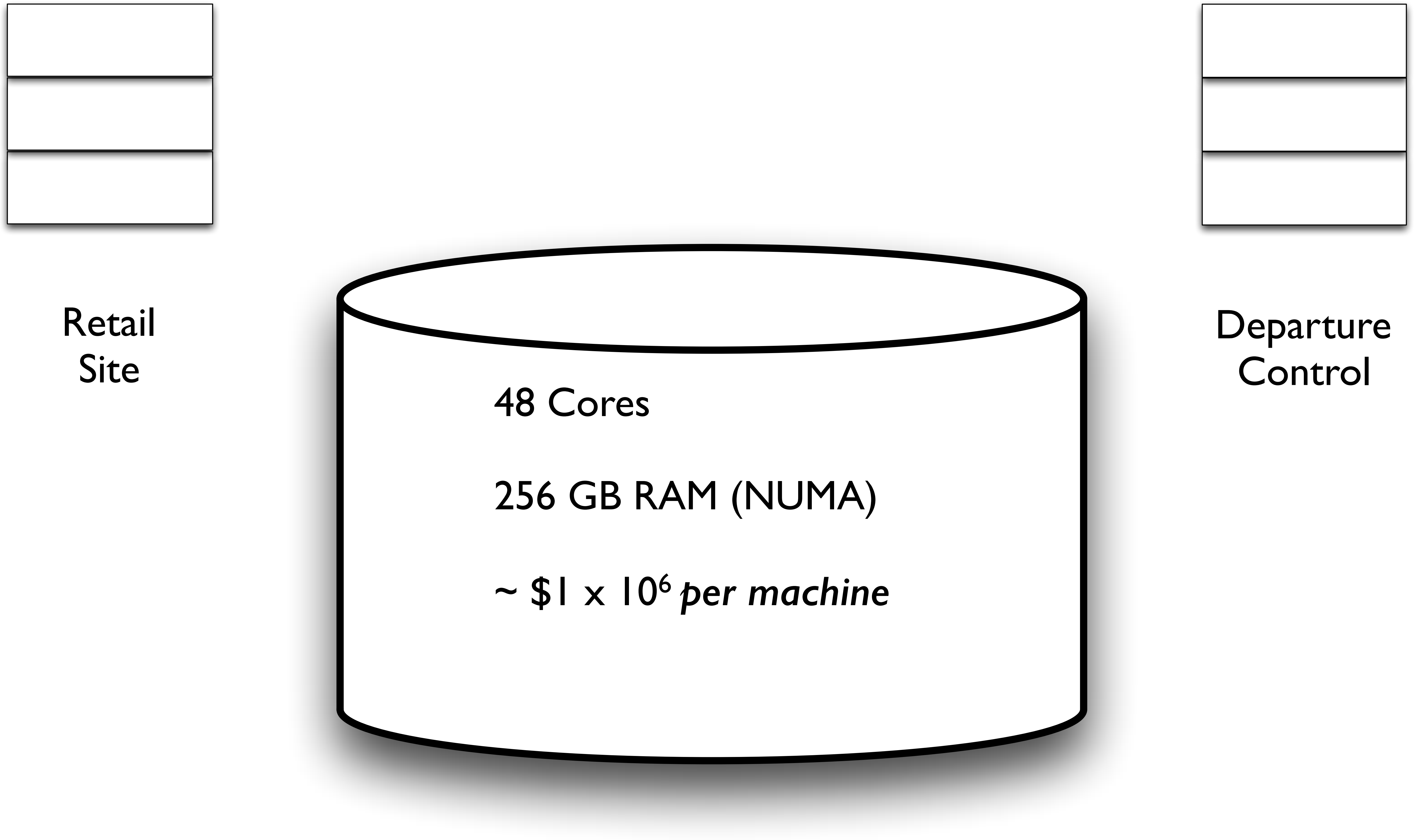
The architects dream

Airline problems with: monolithic databases ~ 2010

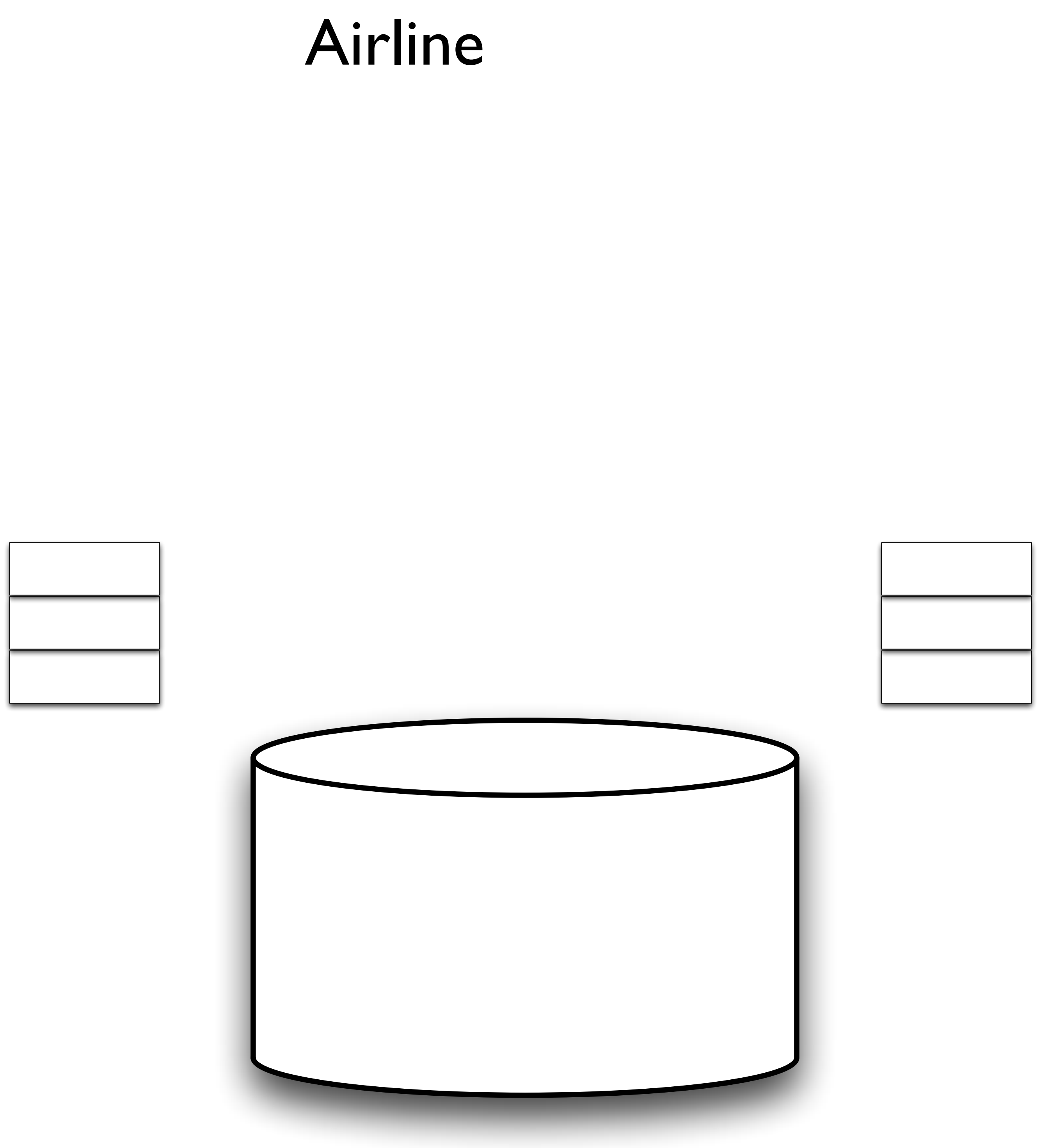

Retail
Site
Departure
Control


Retail
Site
Departure
Control


Retail
Site
Departure
Control


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


ONE DOLLAR
G 27286156 A



Tightly coupled

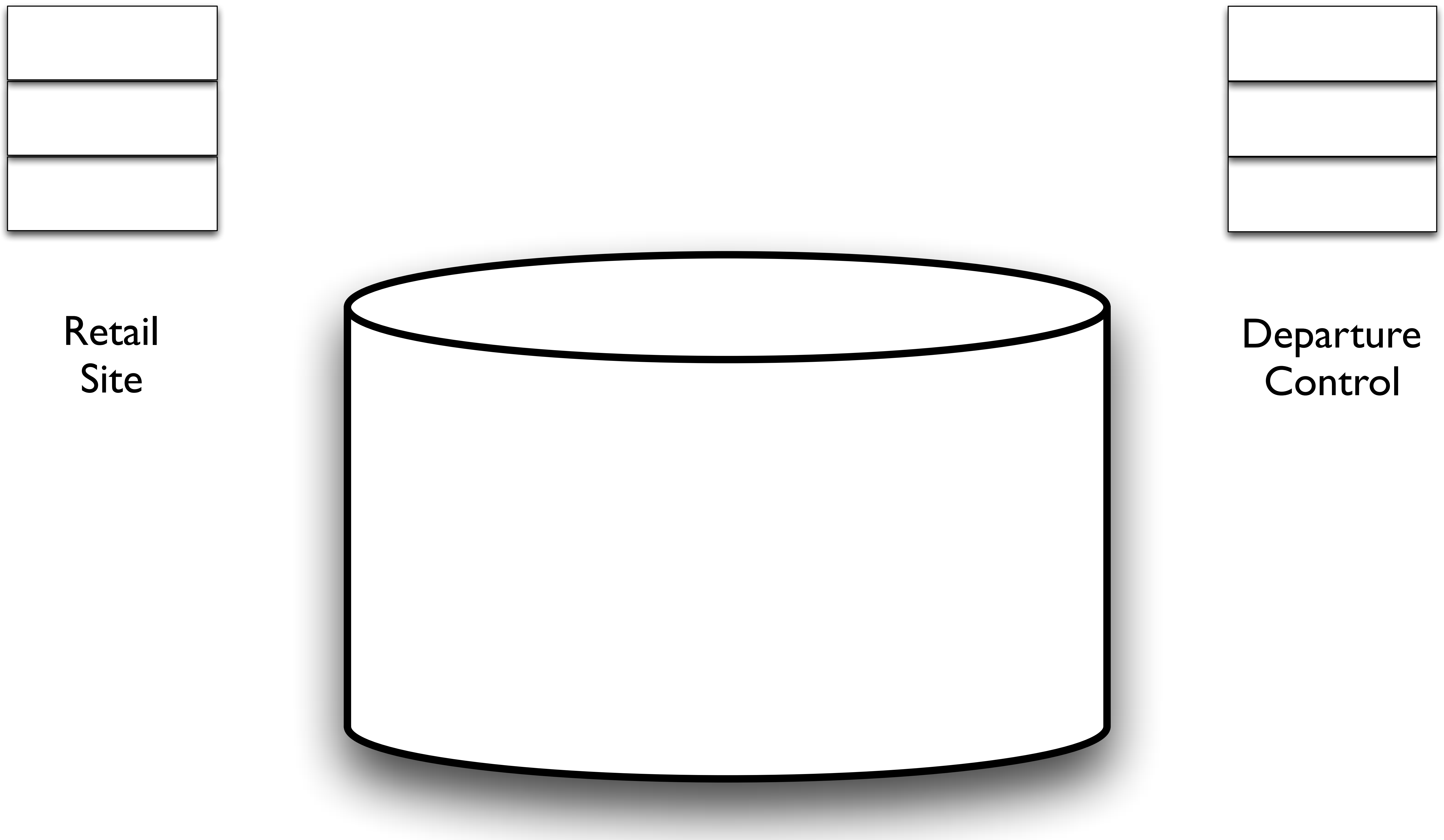
Single point of scaling

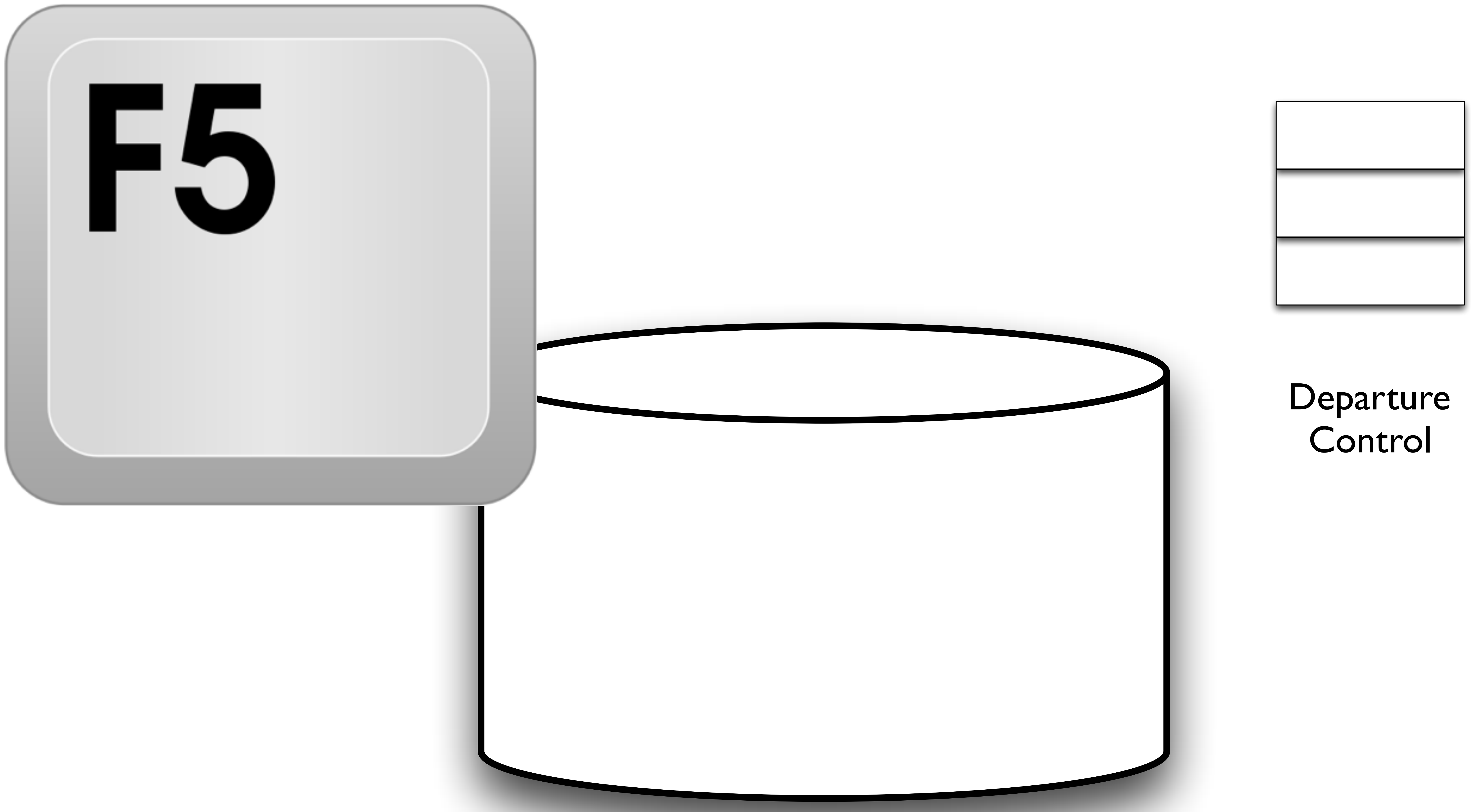
Single point of failure

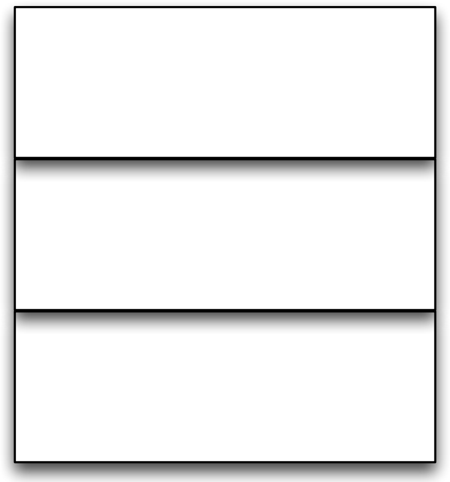
Expensive to change

High operational cost

High cost of failure

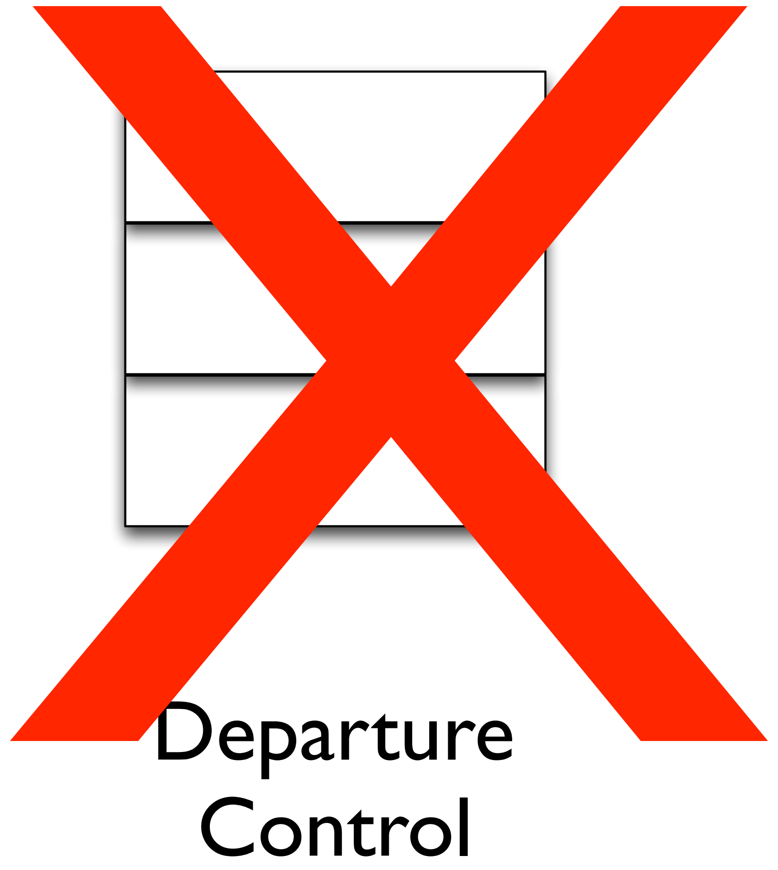

Retail
Site
Departure
Control


F5
Departure
Control



Departure
Control


F5


Departure
Control

BACK IN 2004 (ISH)

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.

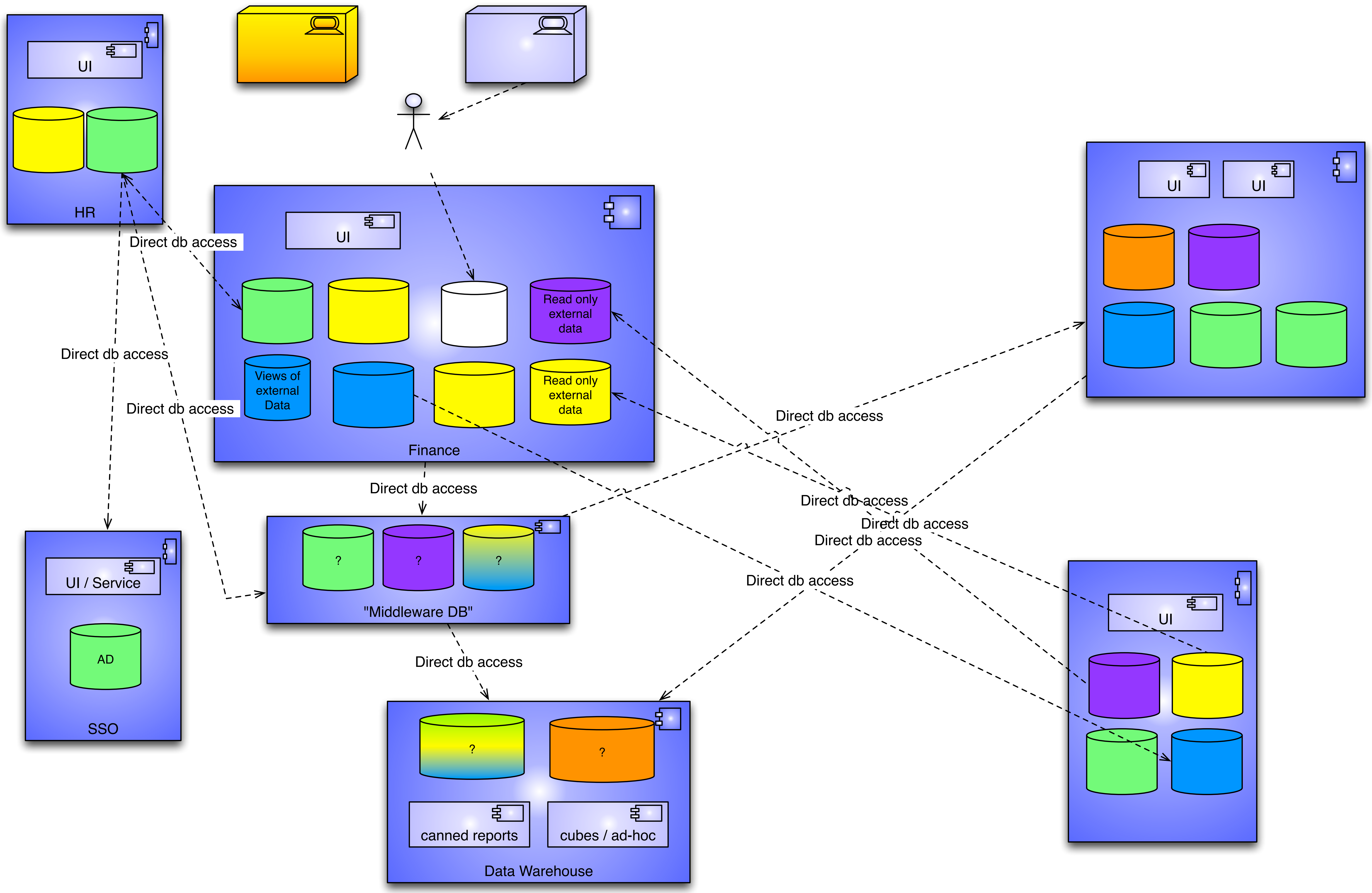
The mandate closed with:

> " *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®

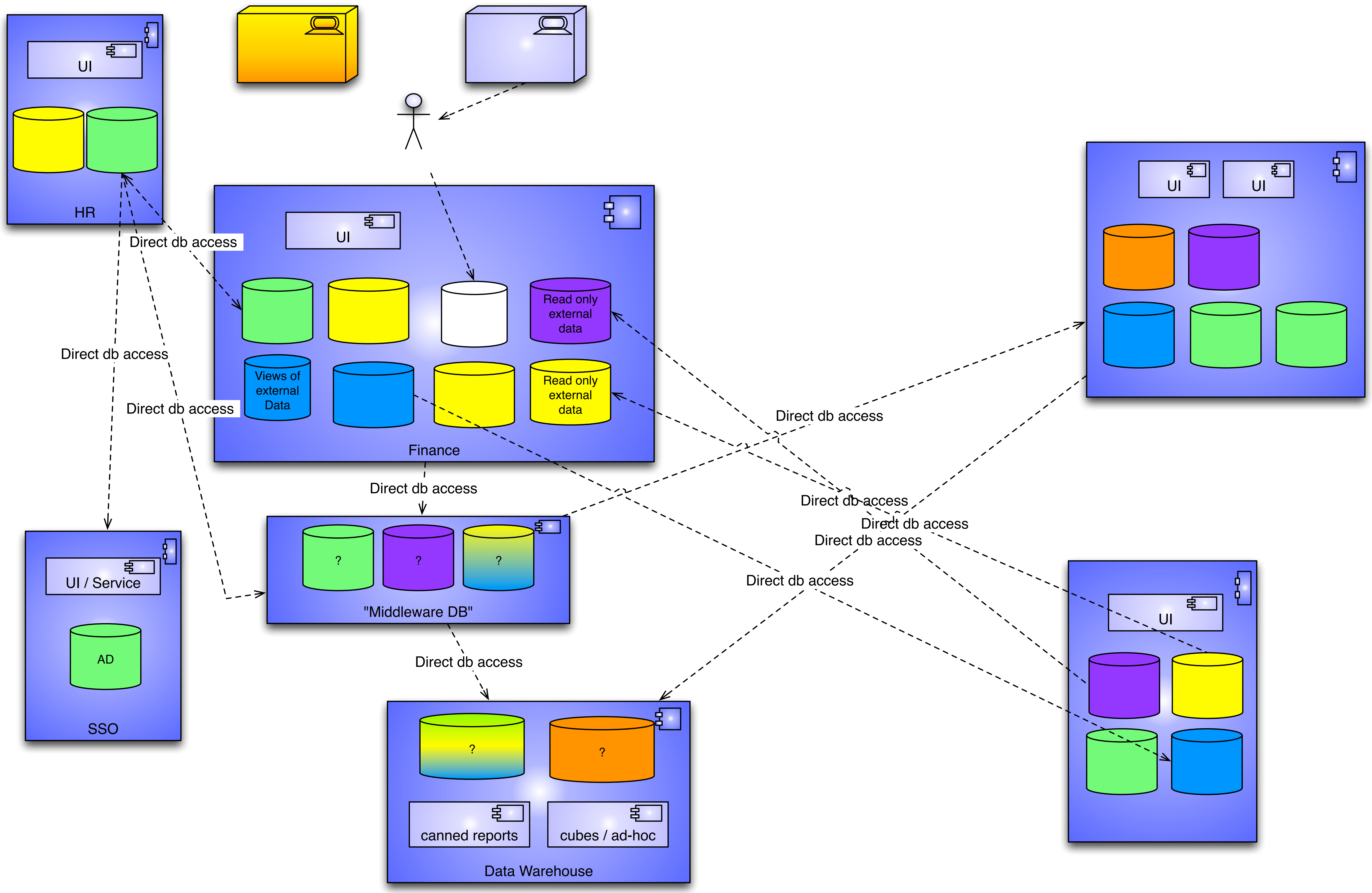


UI
HR
UI
Read only
external
data
Views of
external
Data
Read only
external
data
Finance
Direct db access
Direct db access
Direct db access
Direct db access
UI / Service
AD
SSO
?
?
?
"Middleware DB"
Direct db access
?
?
canned reports
cubes / ad-hoc
Data Warehouse
UI
UI
Direct db access
Direct db access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

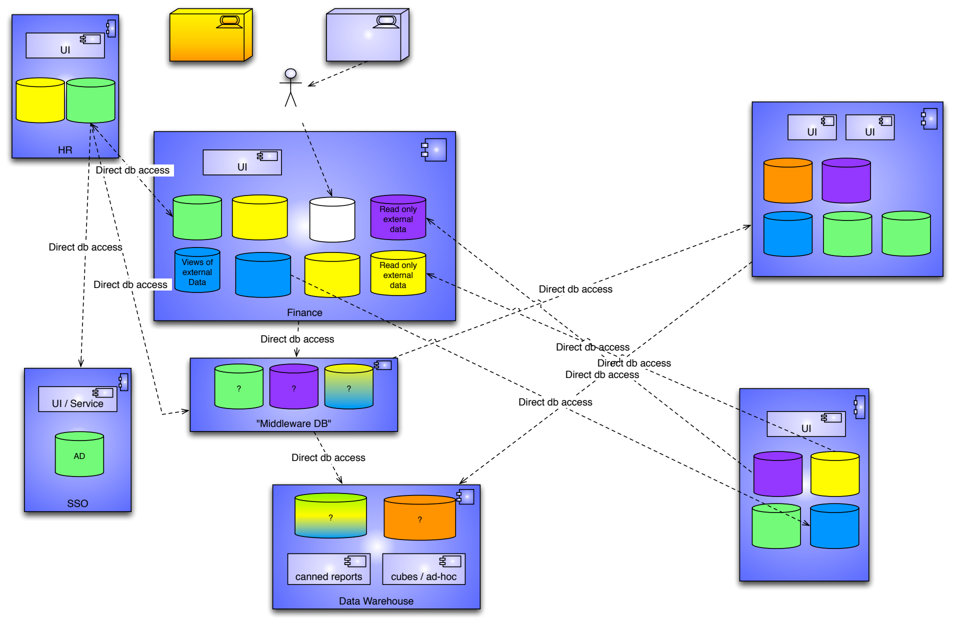
The stovepipe enterprise

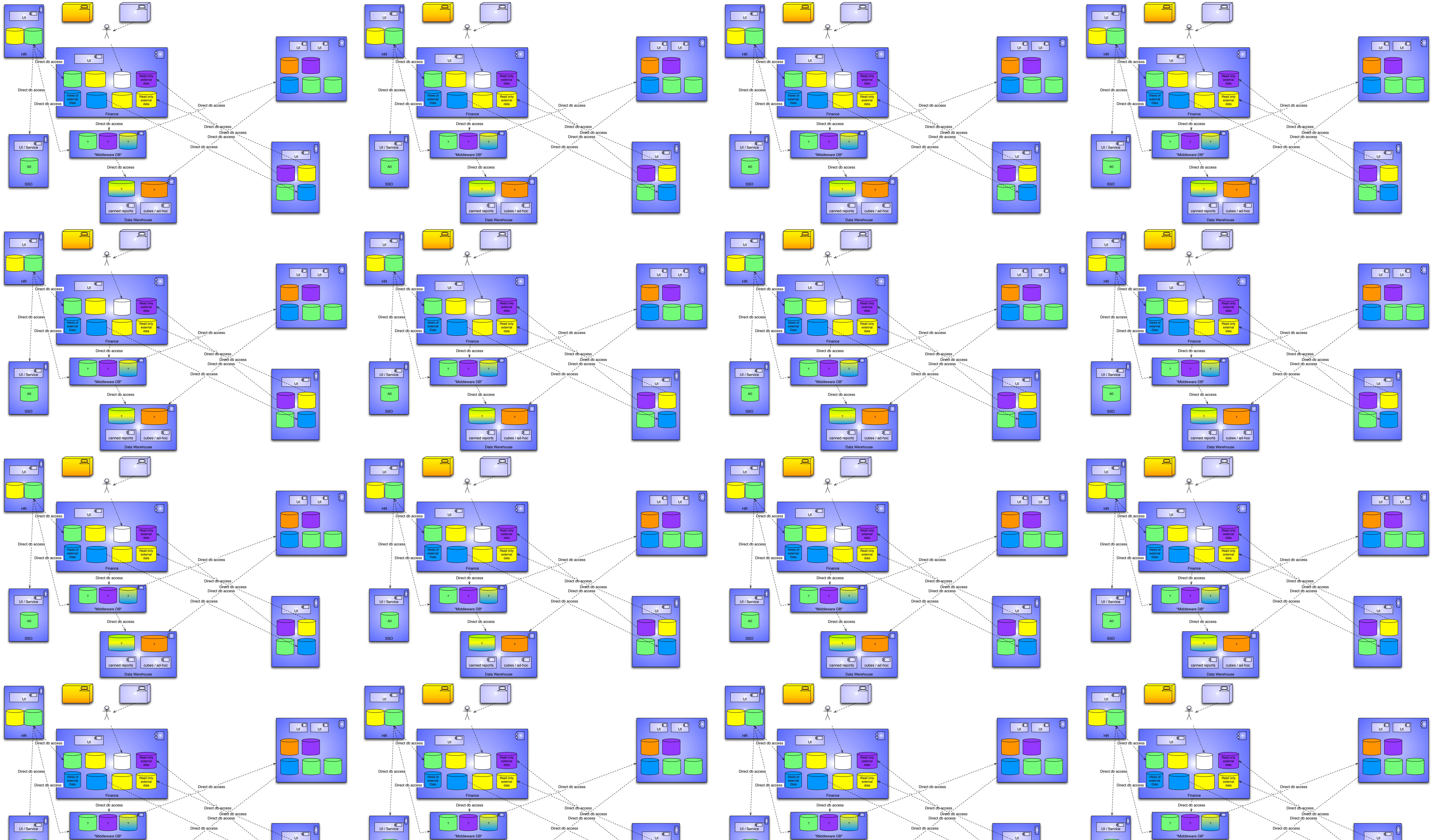
Stovepipes are “systems procured and developed to solve a specific problem, characterized by a limited focus and functionality, and containing data that cannot be easily shared with other systems.” (DOE 1999)

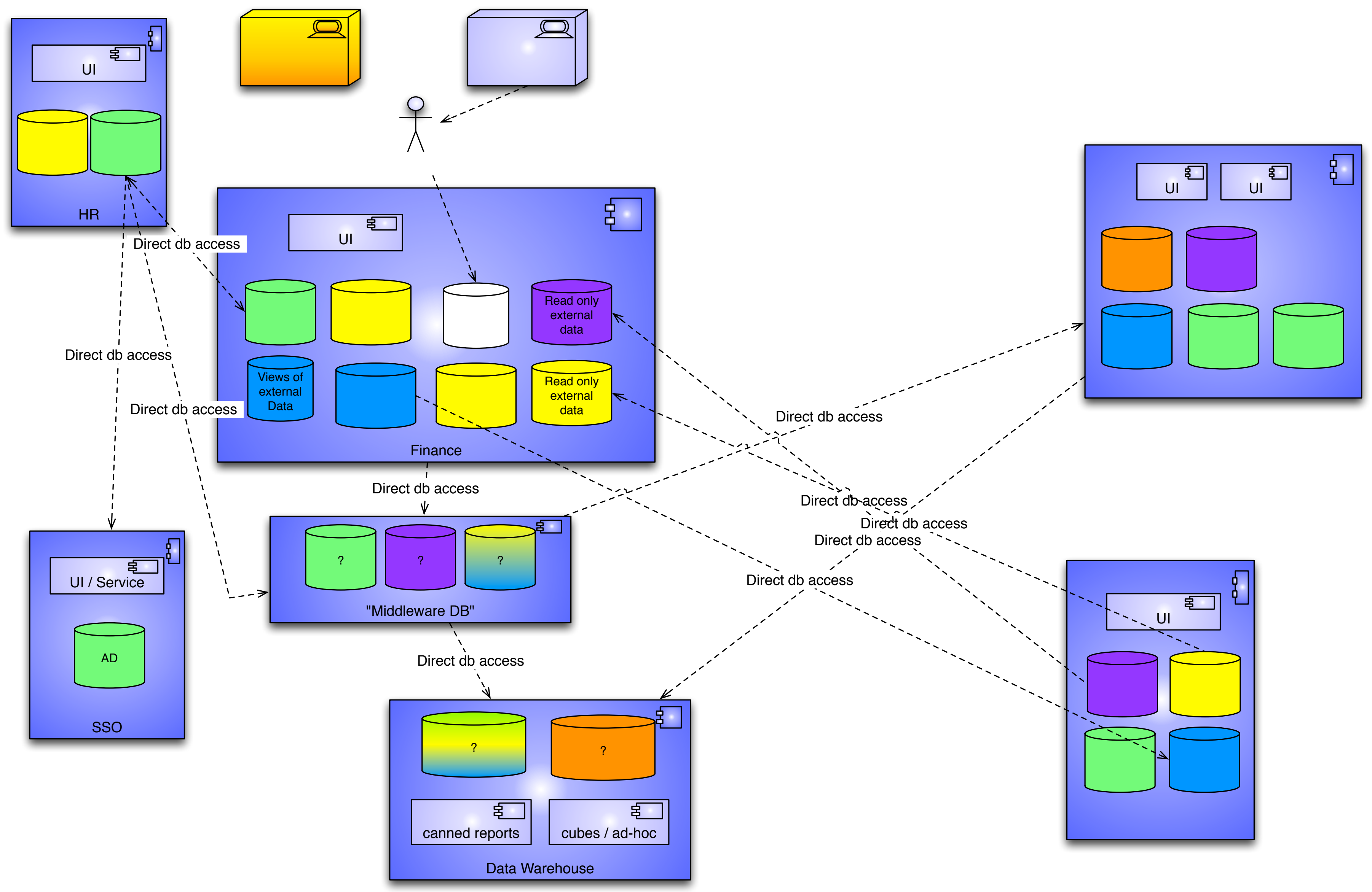
DOE. Committee to Assess the Policies and Practices of the Department of Energy, Improving Project Management in the Department of Energy, National Academy Press, Washington, D.C., 1999, page 133.



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
UI
Read only external data
Views of external Data
Read only external data
Finance
Direct db access
Direct db access
Direct db access
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


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



Logic scattered all over the place

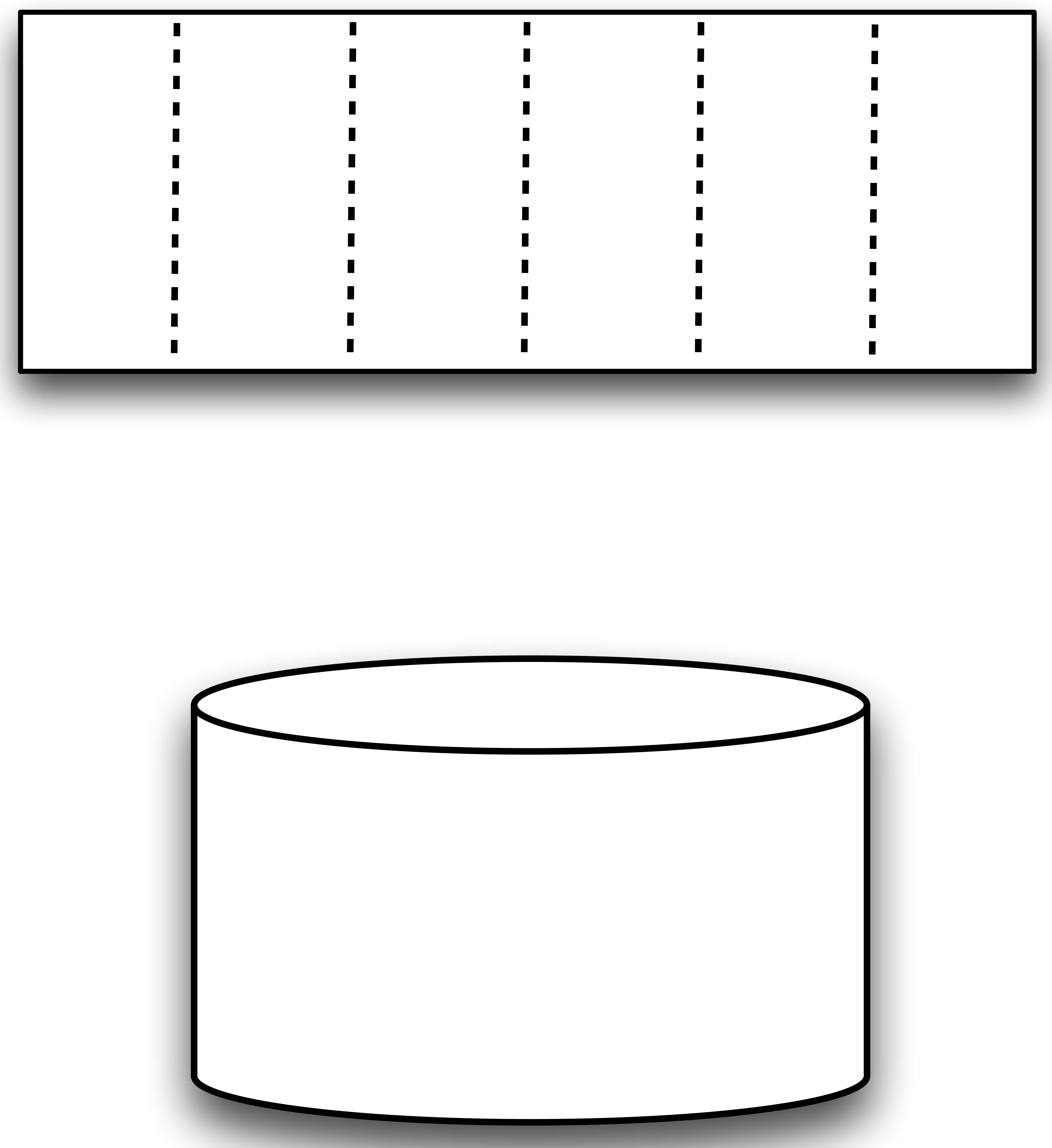
Data scattered all over the place

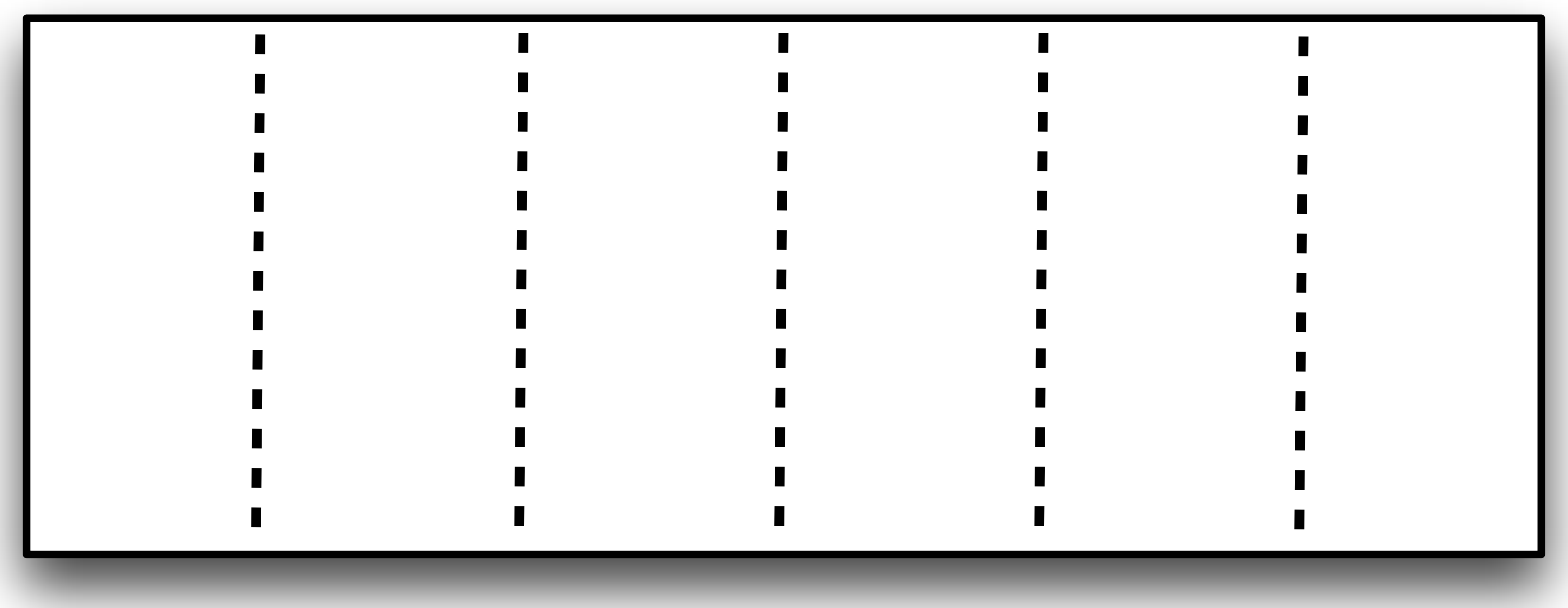
Difficult to predict the effect of changes

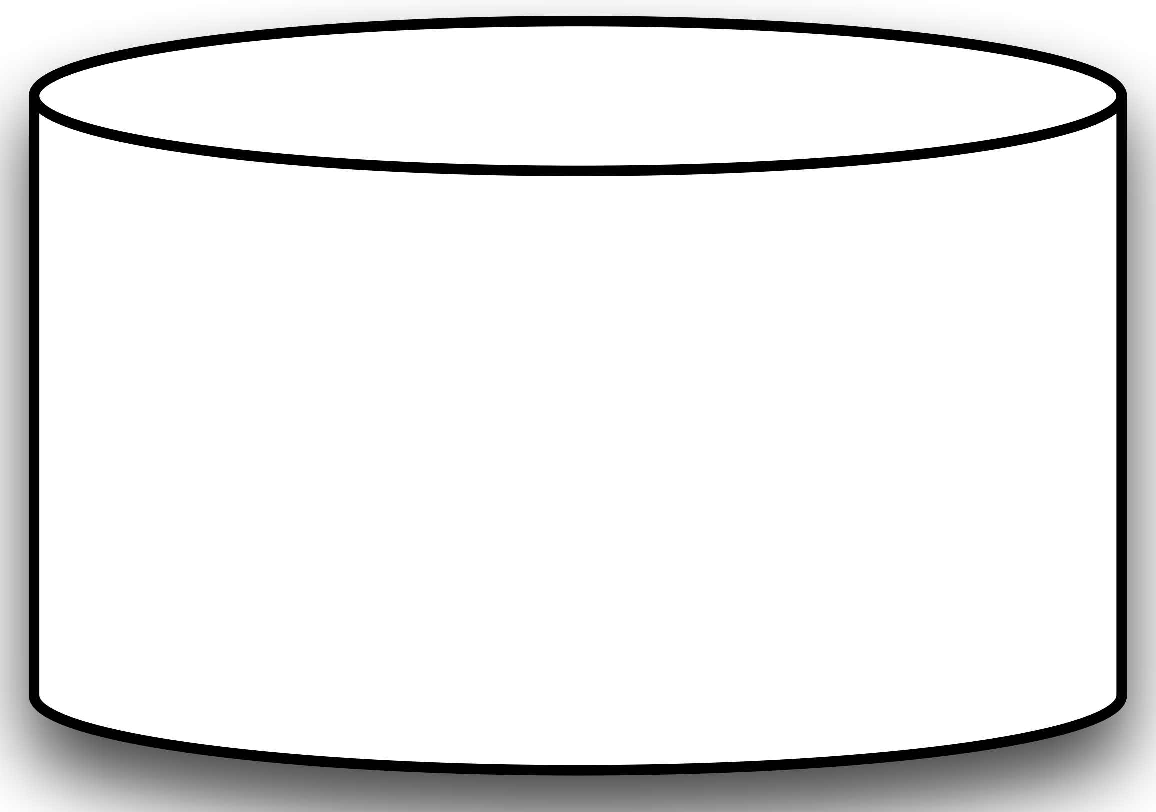
Where are the sources of truth?

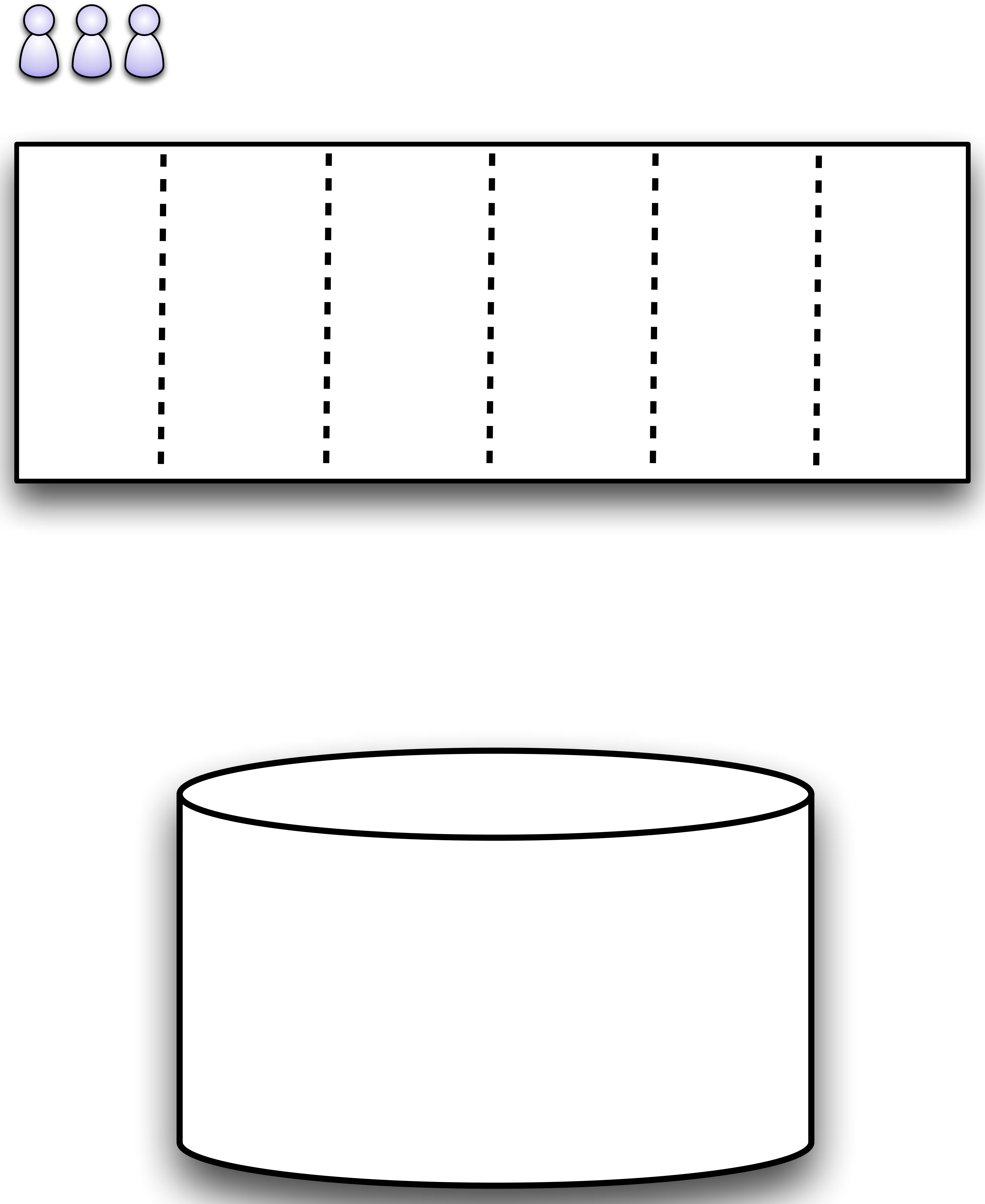
BI / MI almost impossible to get at

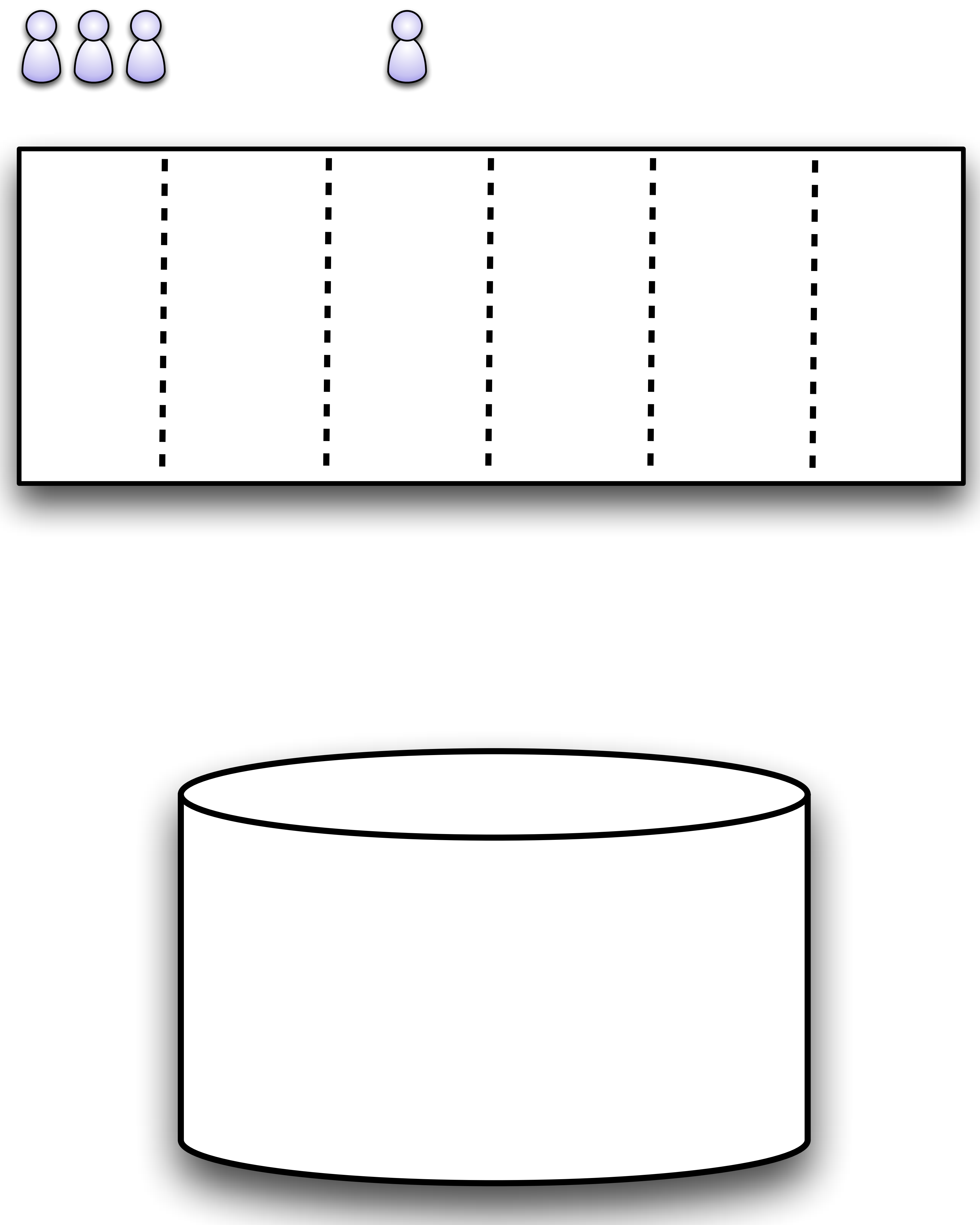
Insurance - 2011

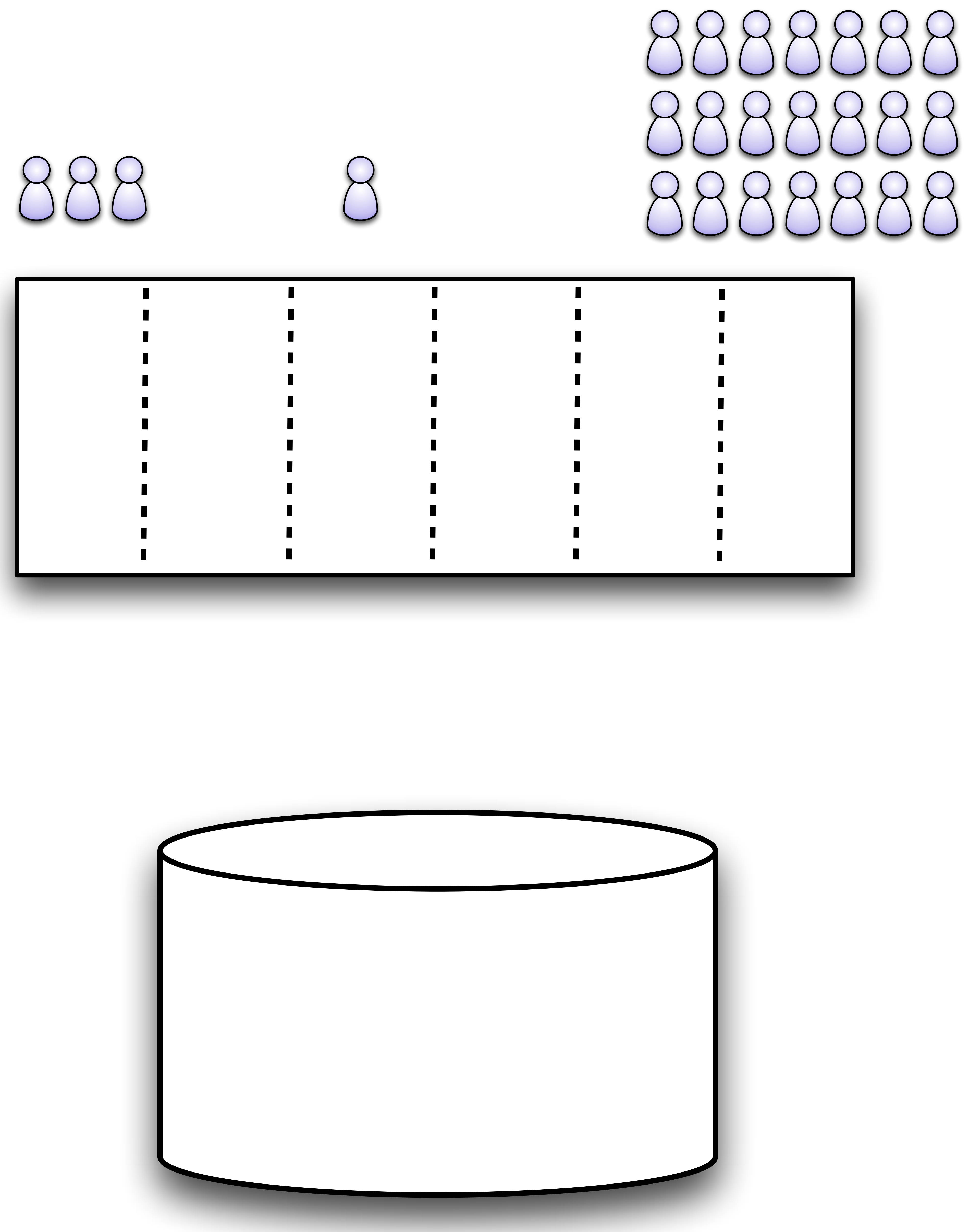


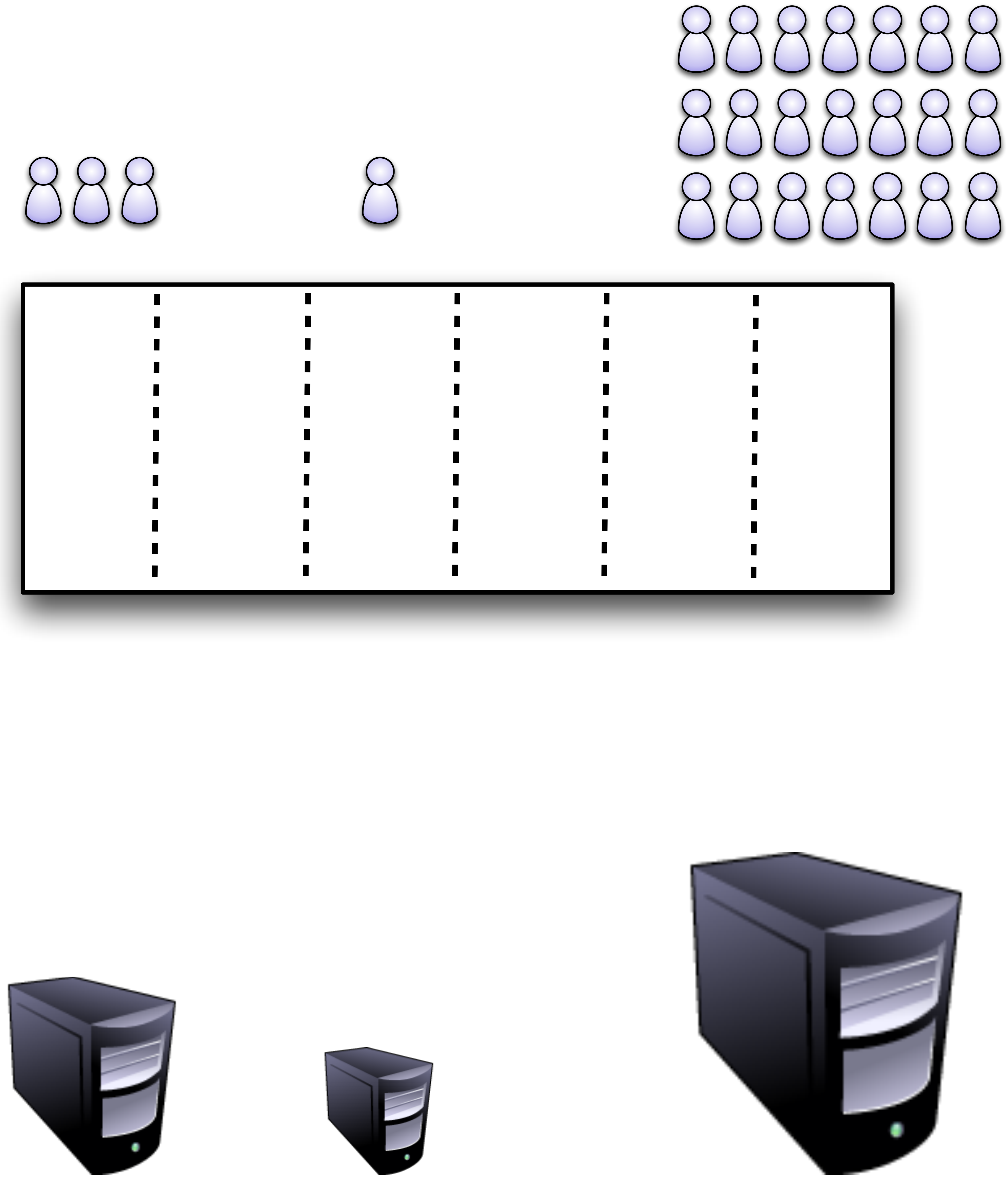


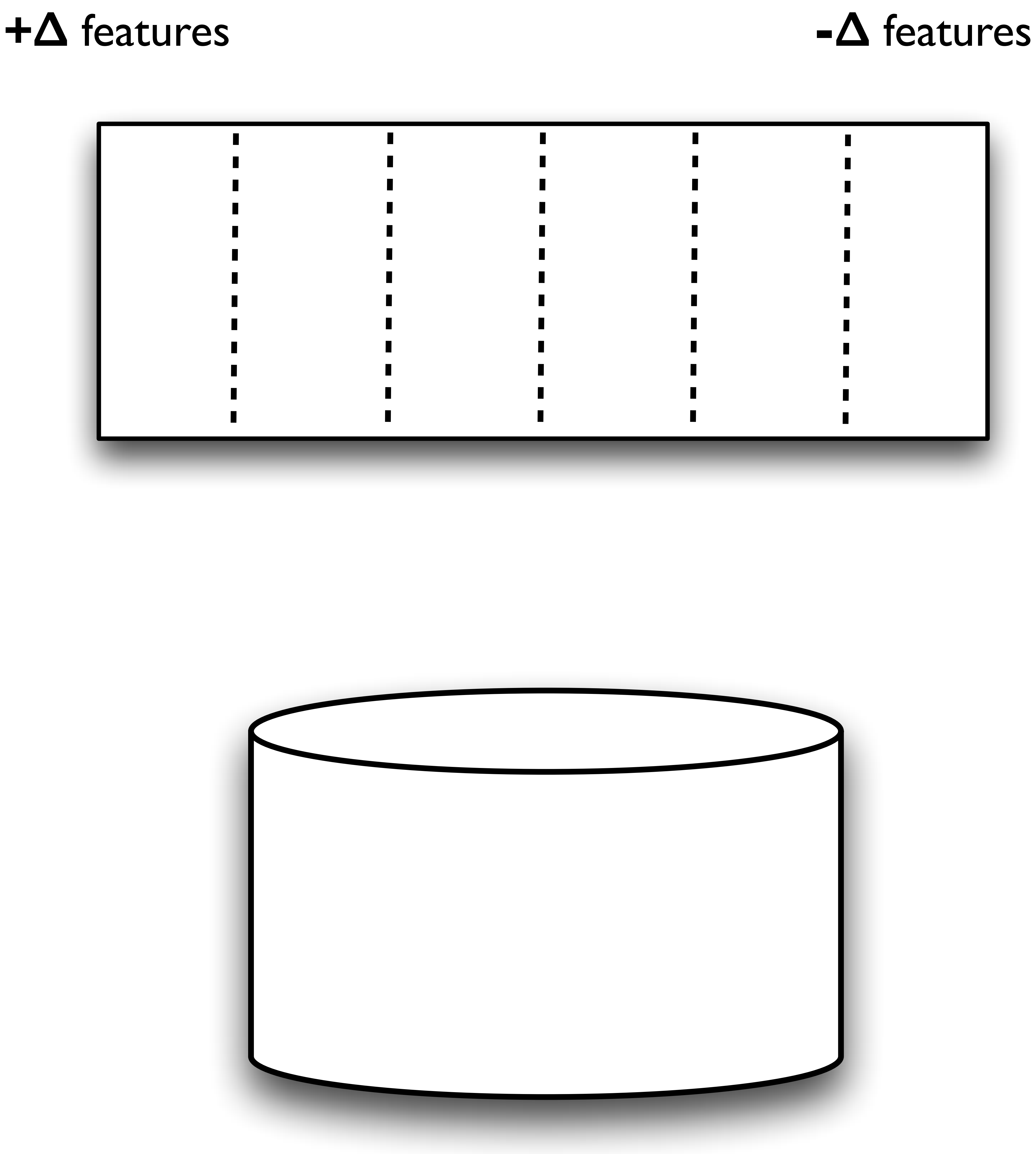






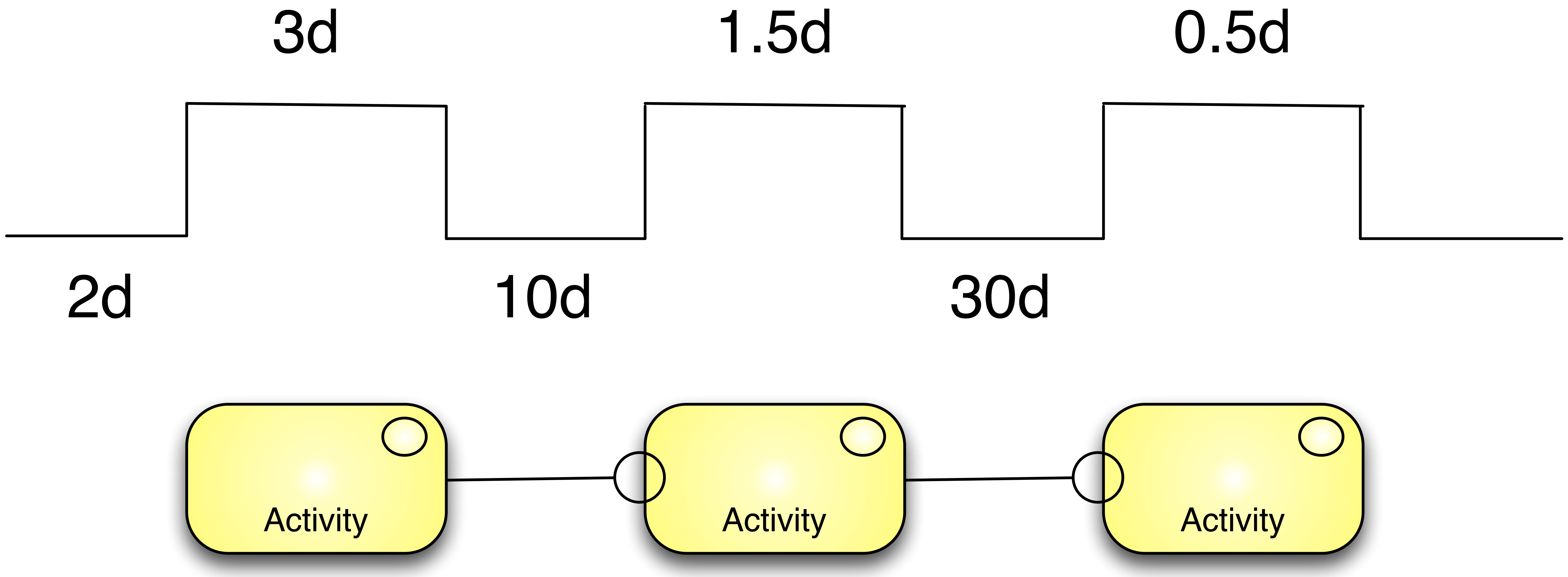






+Δ features
-Δ features

extremely high cost of delay

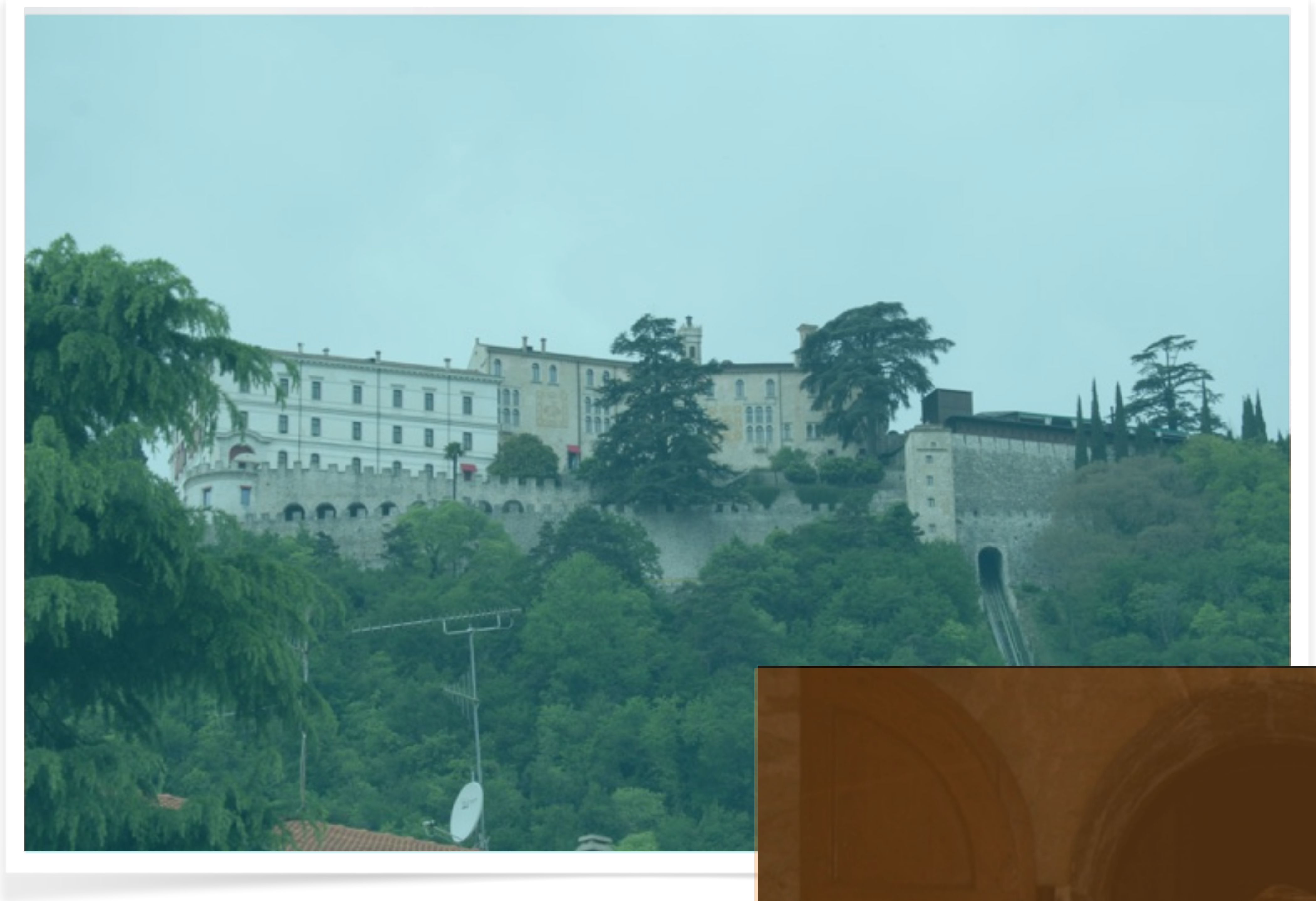


Summary of CTM

- basically
  - –High cost of delay:
    - long lead times
      - Project teams
      - Long lived branches and awful merges
- (very coupled so stamping over each others code)

problems thereof

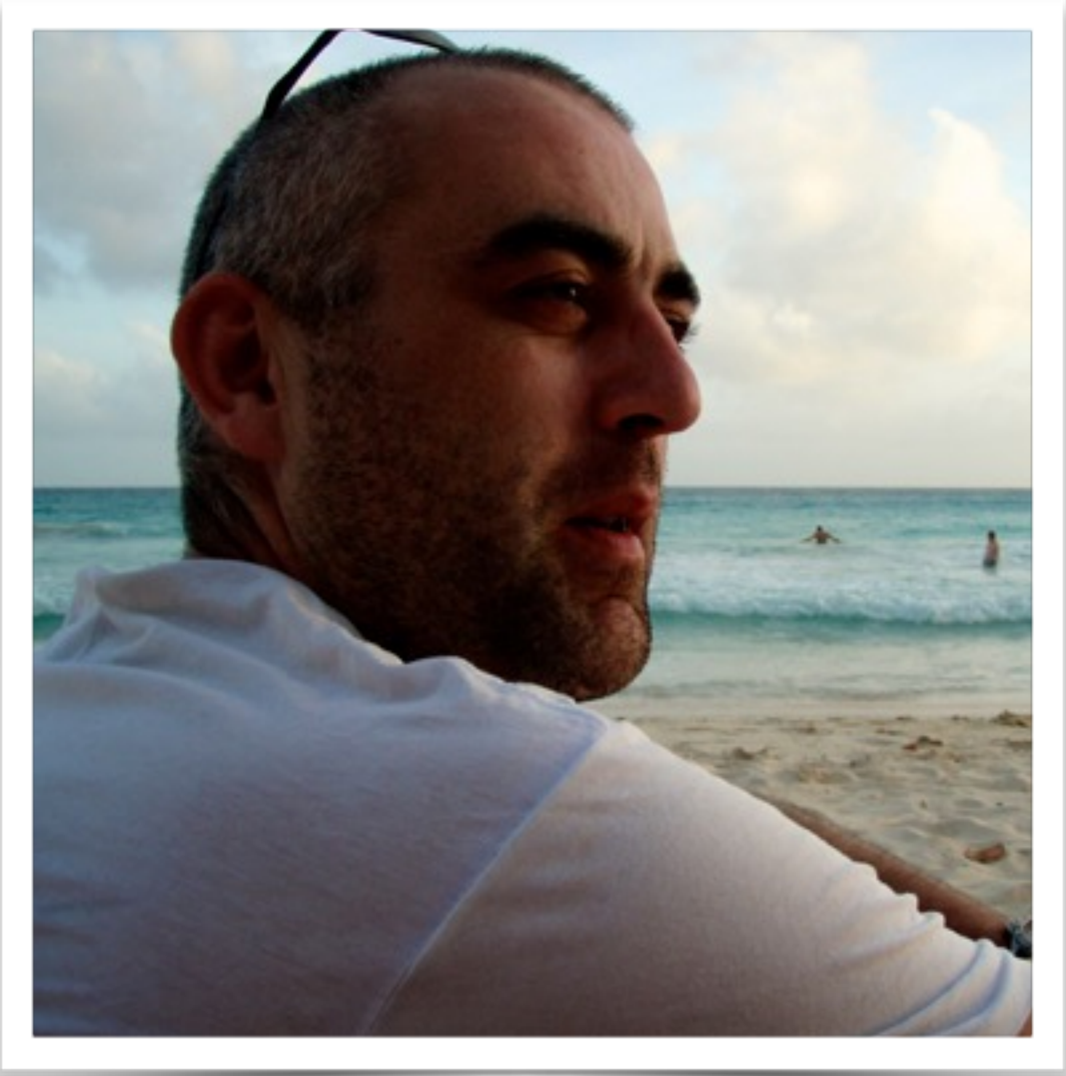
- High cost of change
  - –sufficiently complex systems become more coupled over time
    - Martin’s tech debt quadrant
- High operational cost
- Lead time to business impact



SAW 2011

TIME PASSES...

programmer anarchy



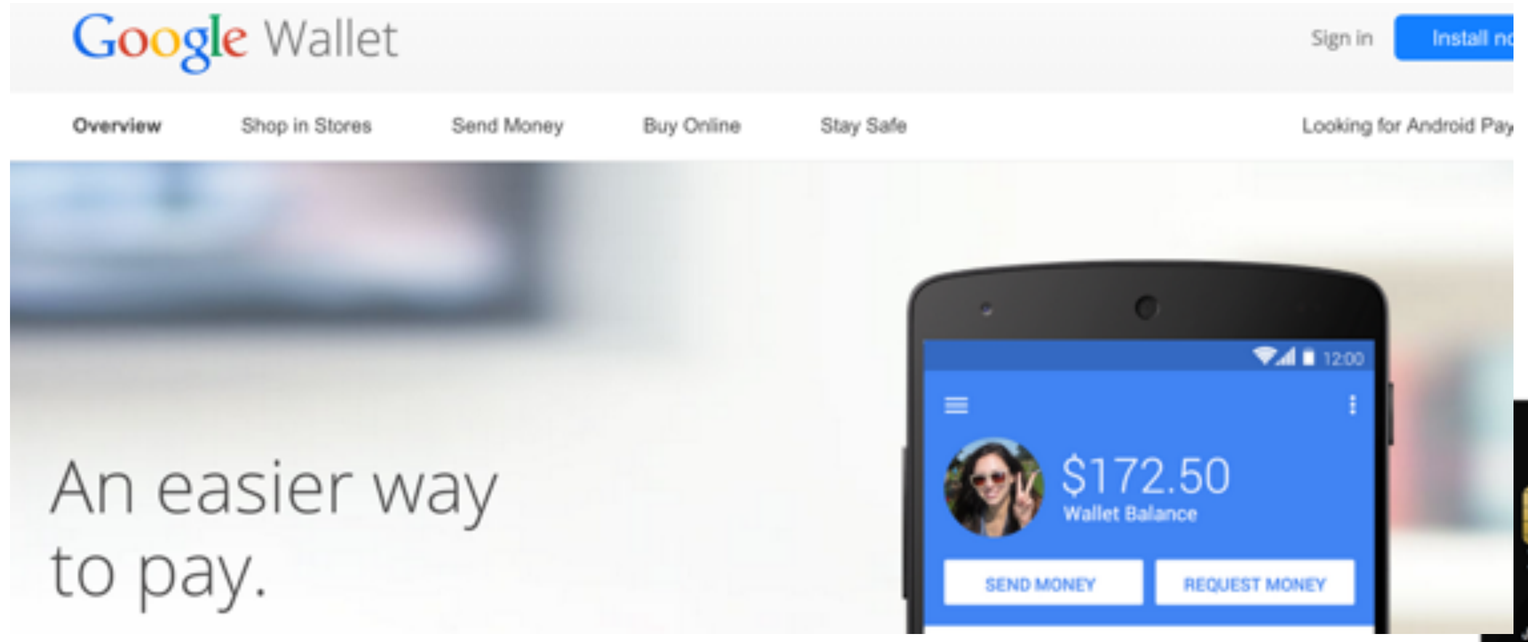
Java the UNIX way

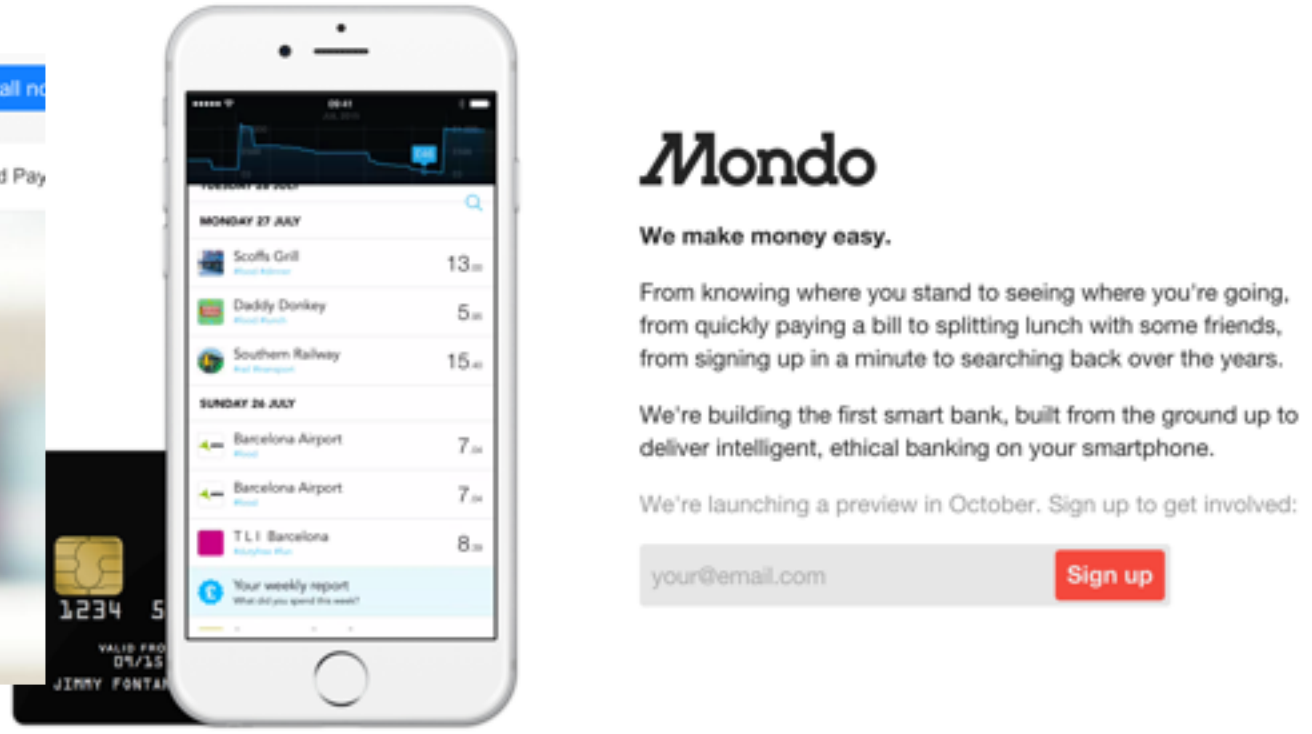
fine-grained SOA

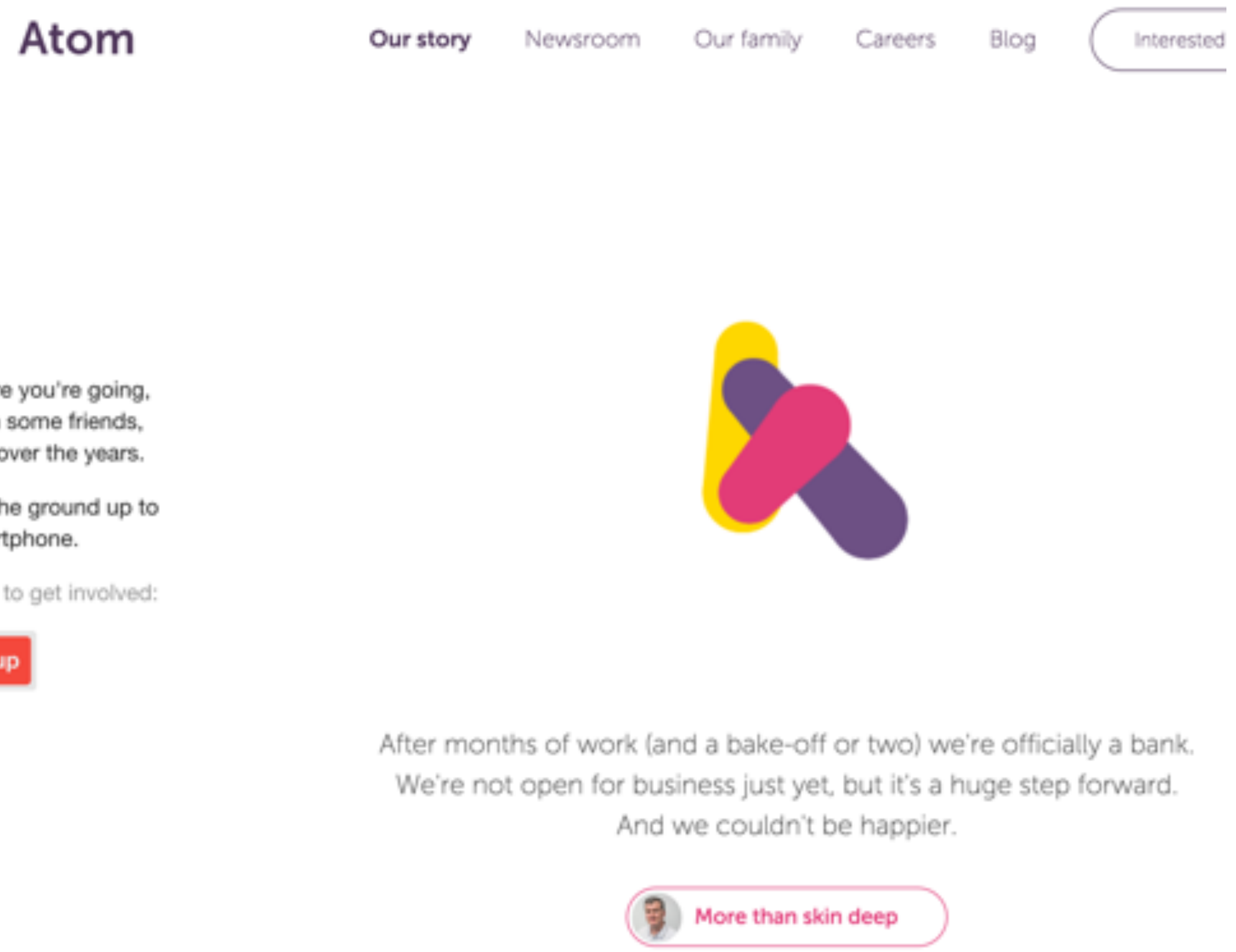
MORE TIME PASSES...

MORE TIME PASSES…

EVEN FINANCE IS NOT IMMUNE







Chapter 2

What are Microservices?

"loosely coupled service oriented architecture with bounded contexts"

Adrian Cockcroft, GOTO Aarhus 2014

“the first post-devops architectural style”

Neal Ford

replaceable component architectures

Dan North

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

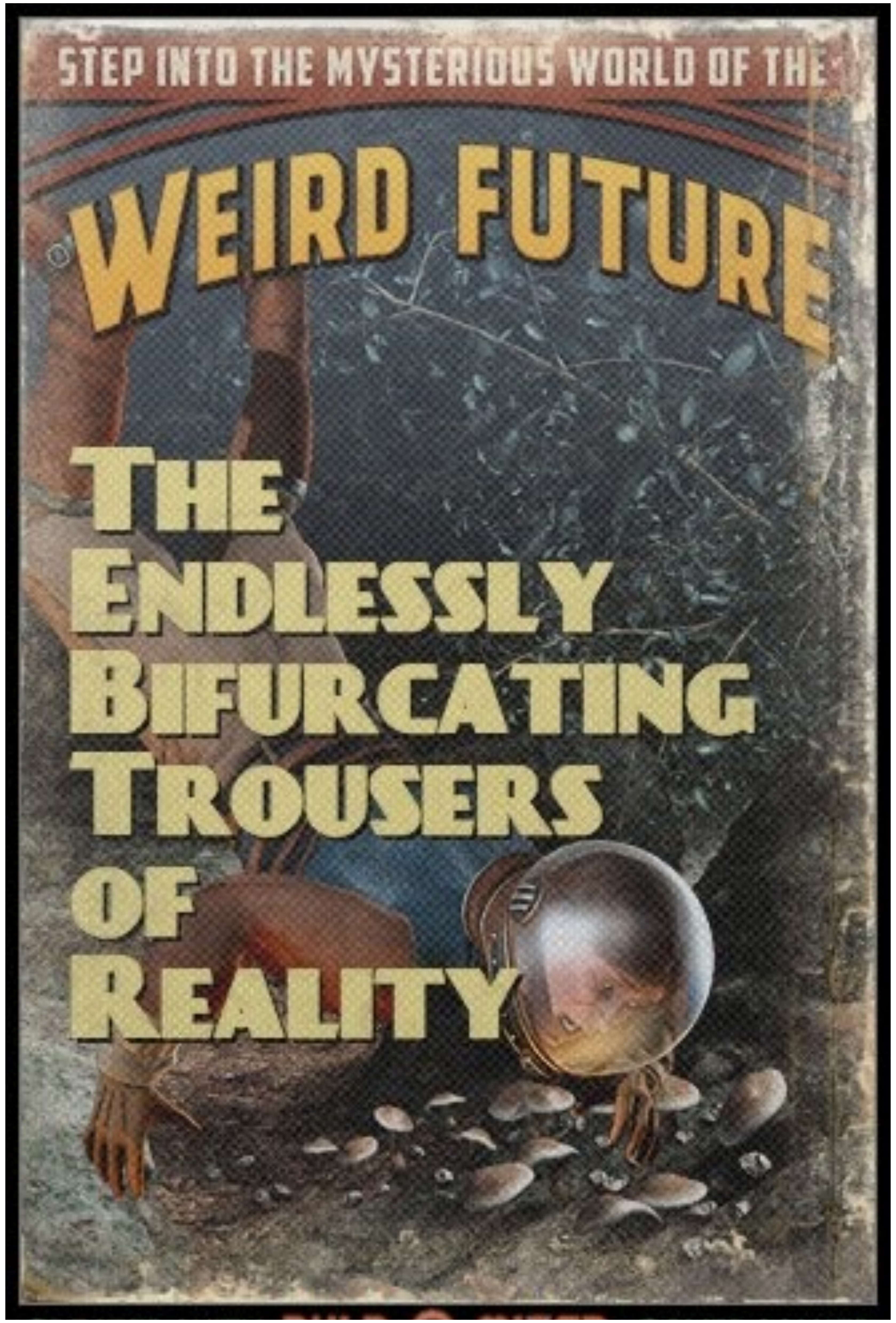
characteristics of microservices

- *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*

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	<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**)

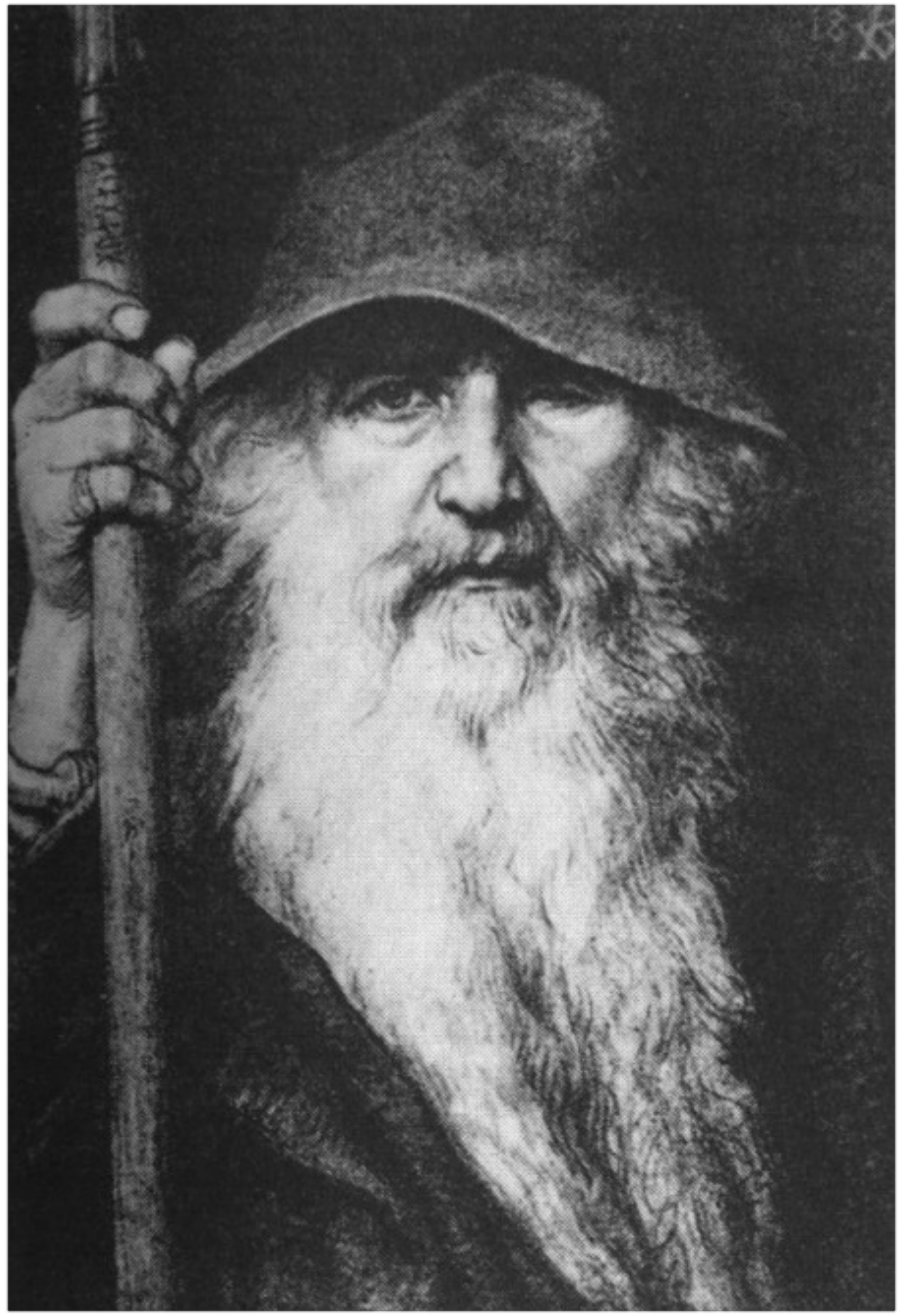
3.

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:

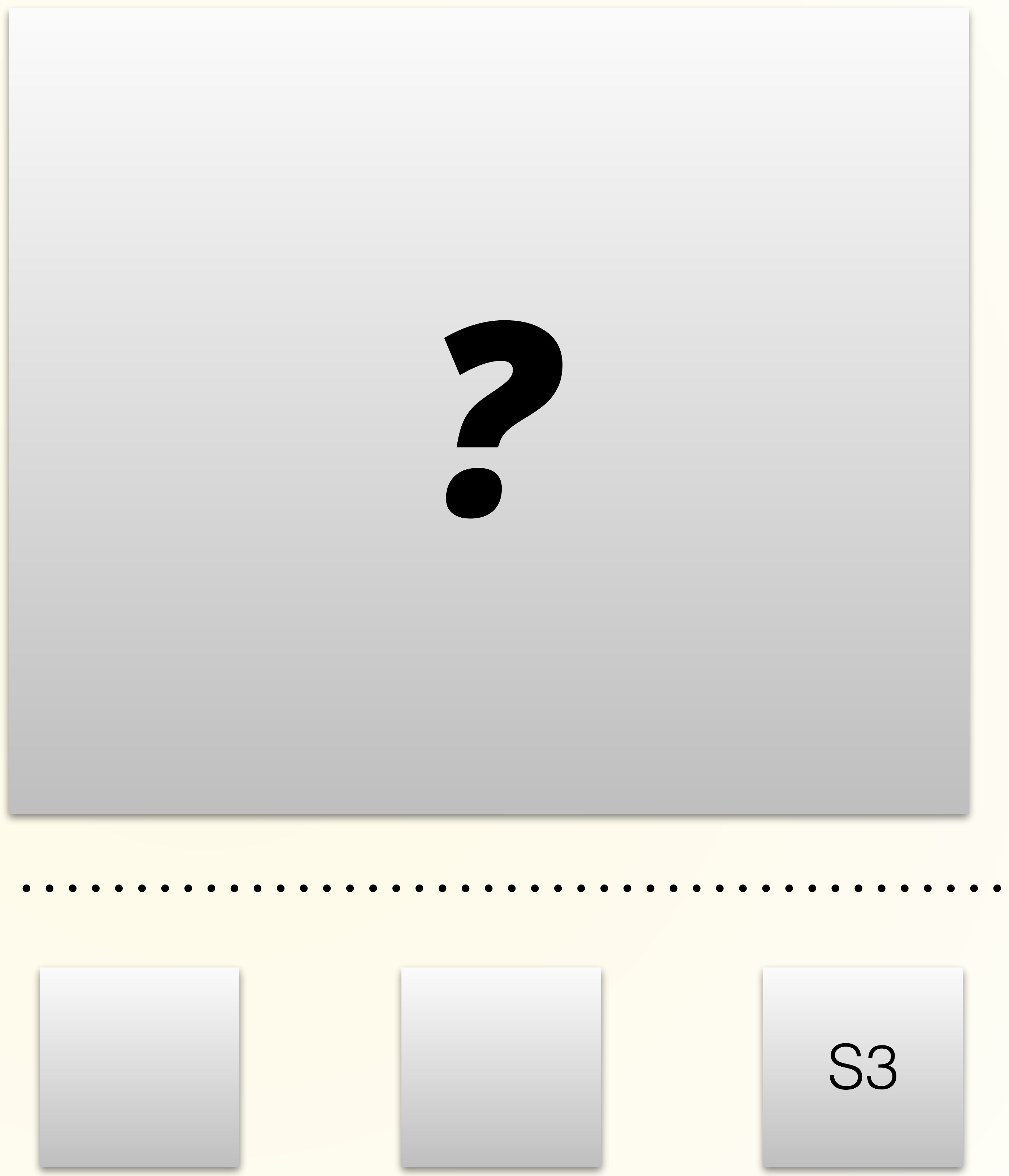
1. improve availability
2. improve performance
3. reduce the cost of delay

An Enterprise Architect approaches and addresses you.

You may use:

Summon Walking Skeleton	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**)



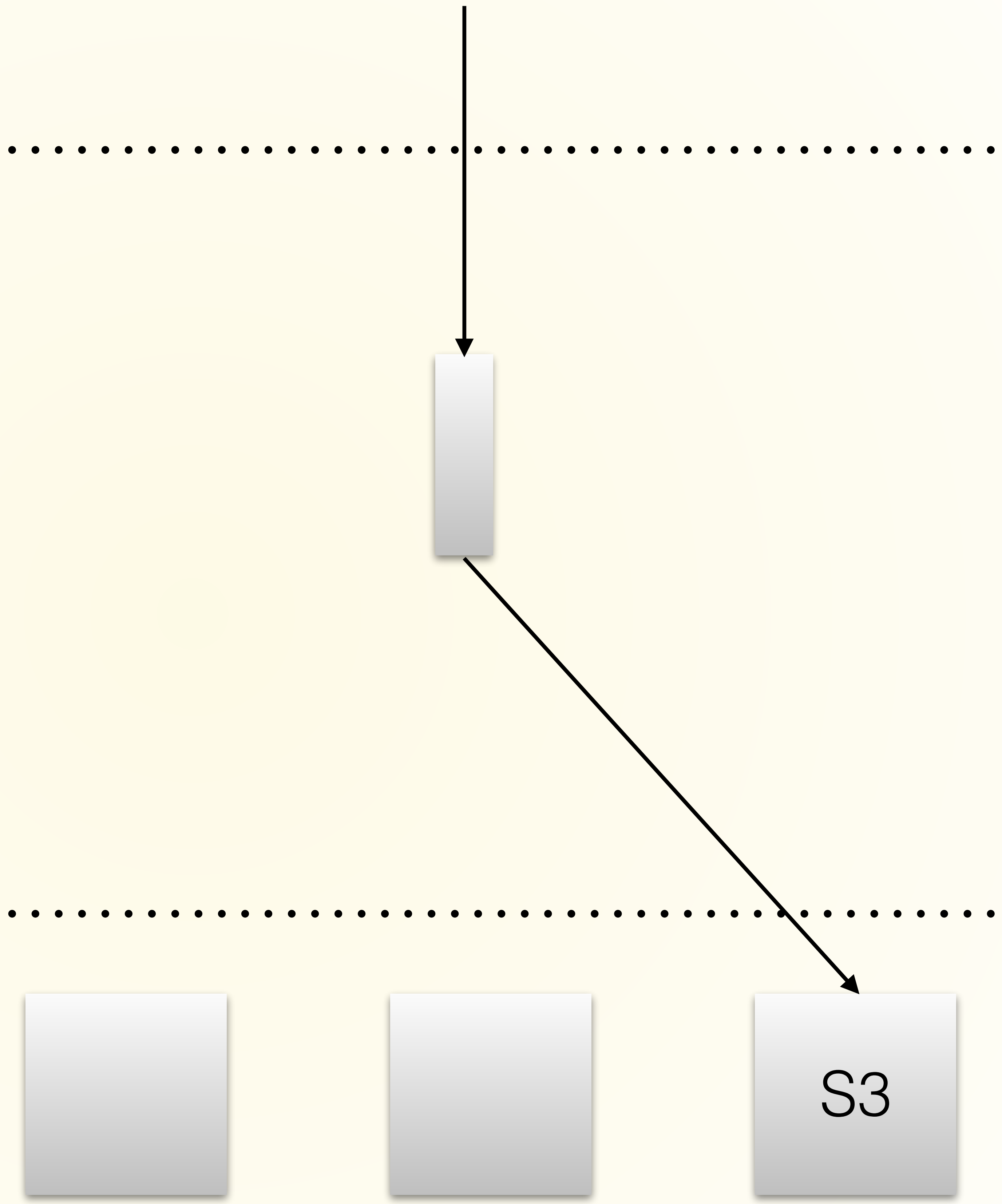
4.

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.

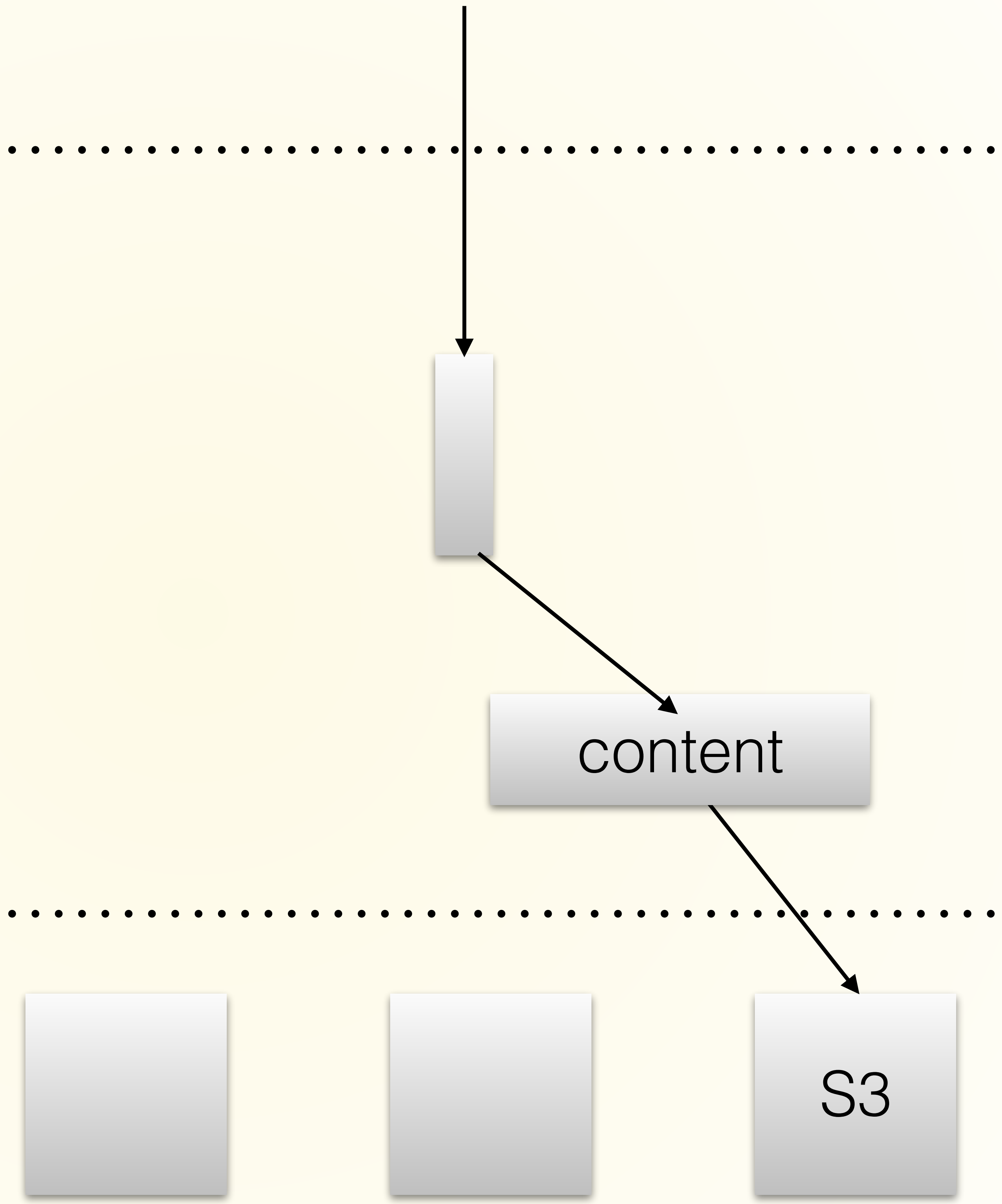
A villager approaches and exclaims: “this beautiful content I see in front of me seems to take an awful long time to get here”

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.

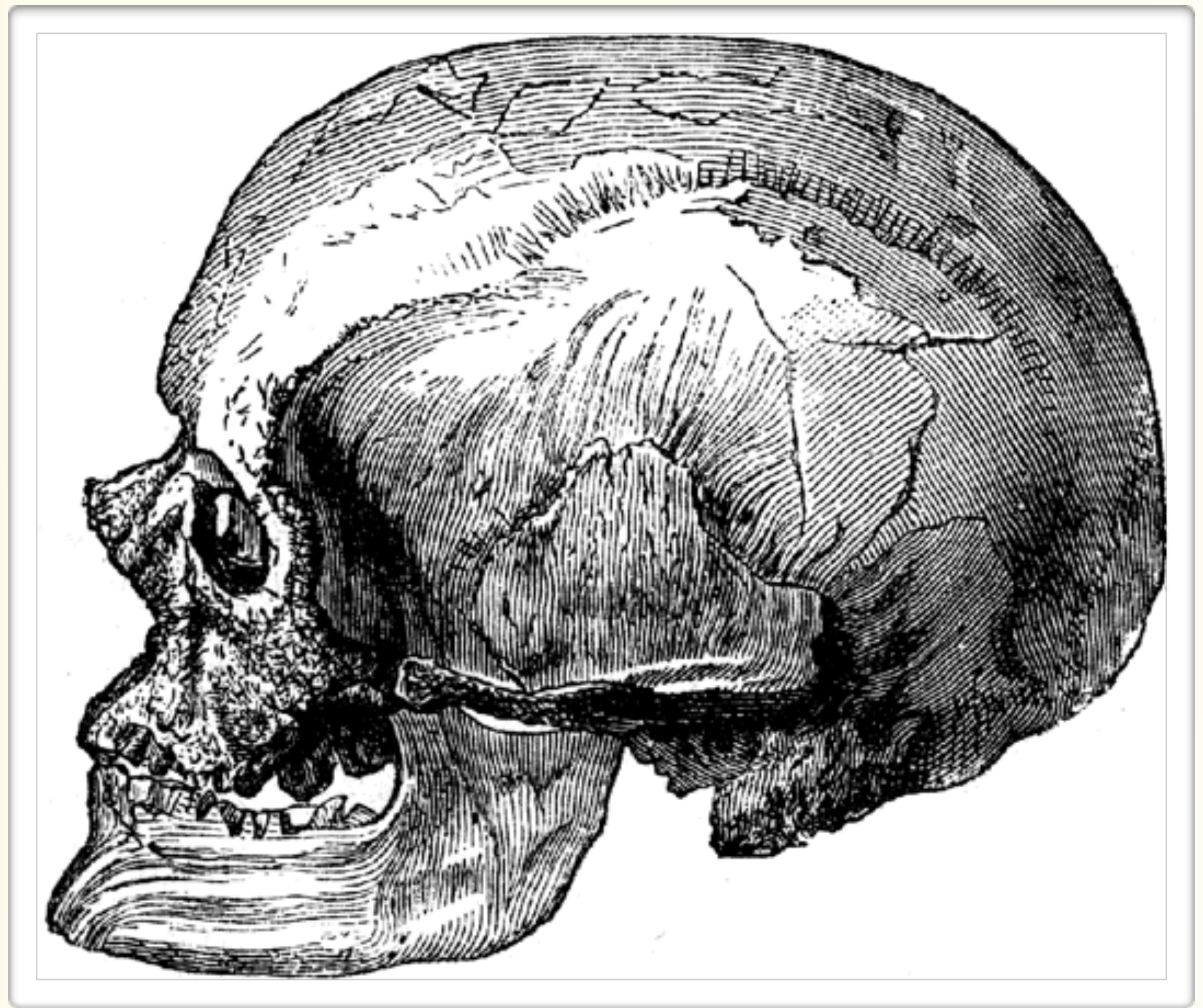
The skeleton gurgles, grunts and then doubles in size.

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

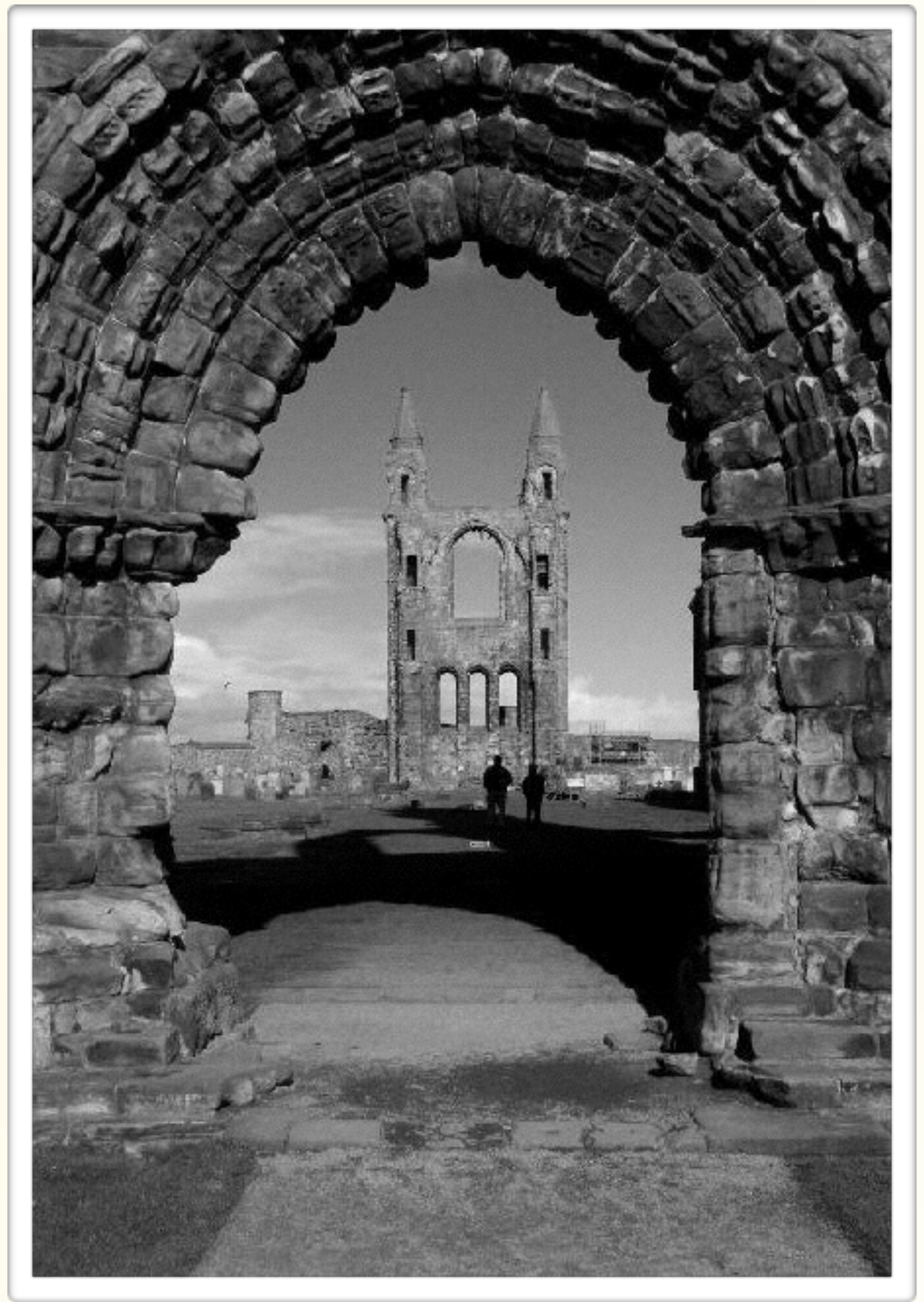
Trust that it will be fast enough on first view turn to 22

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.

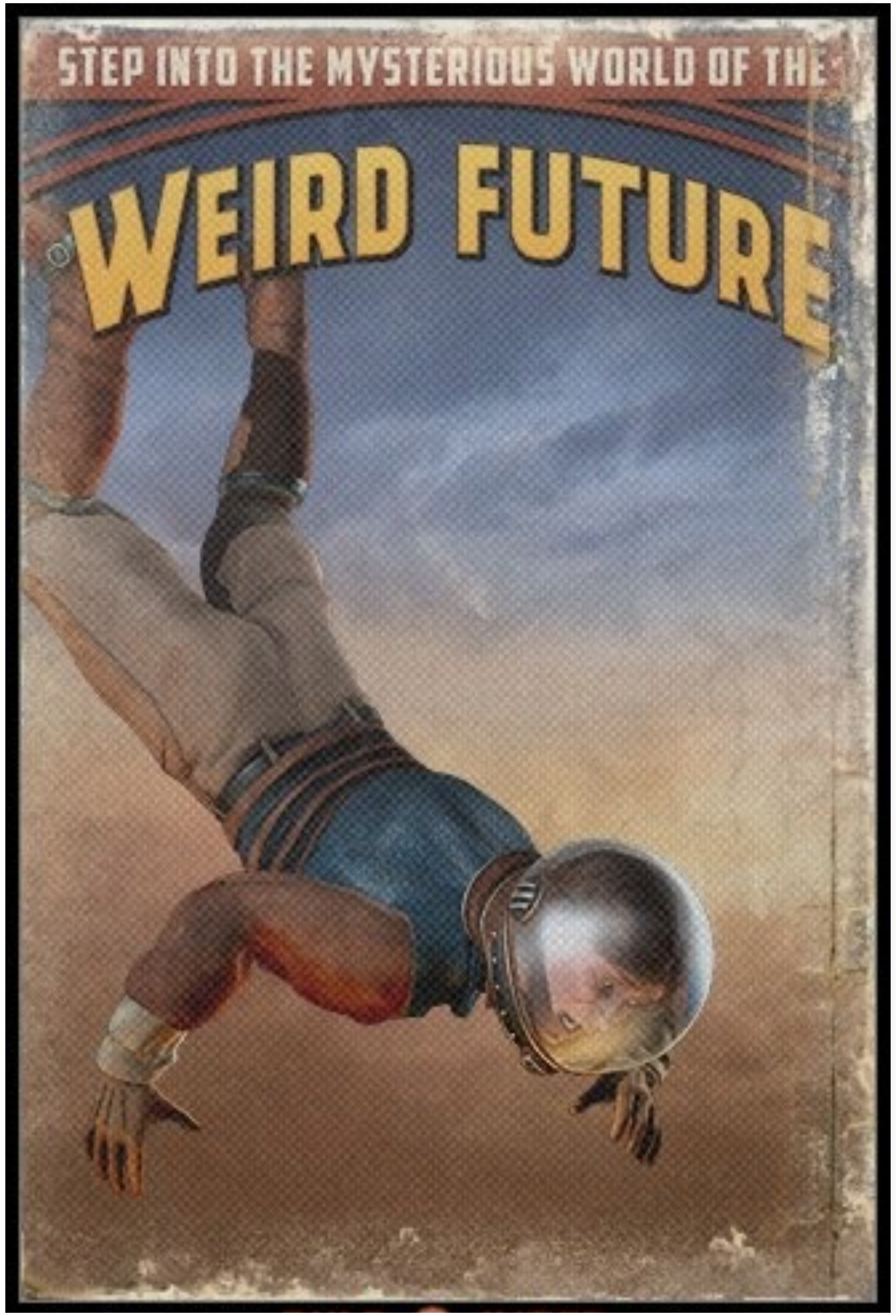
The HTTP cache has an instant effect. Latency drops from 300ms to 10ms.

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

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

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*

A **capability** is a combination of people, processes, systems that provides value to customers (internal or external)

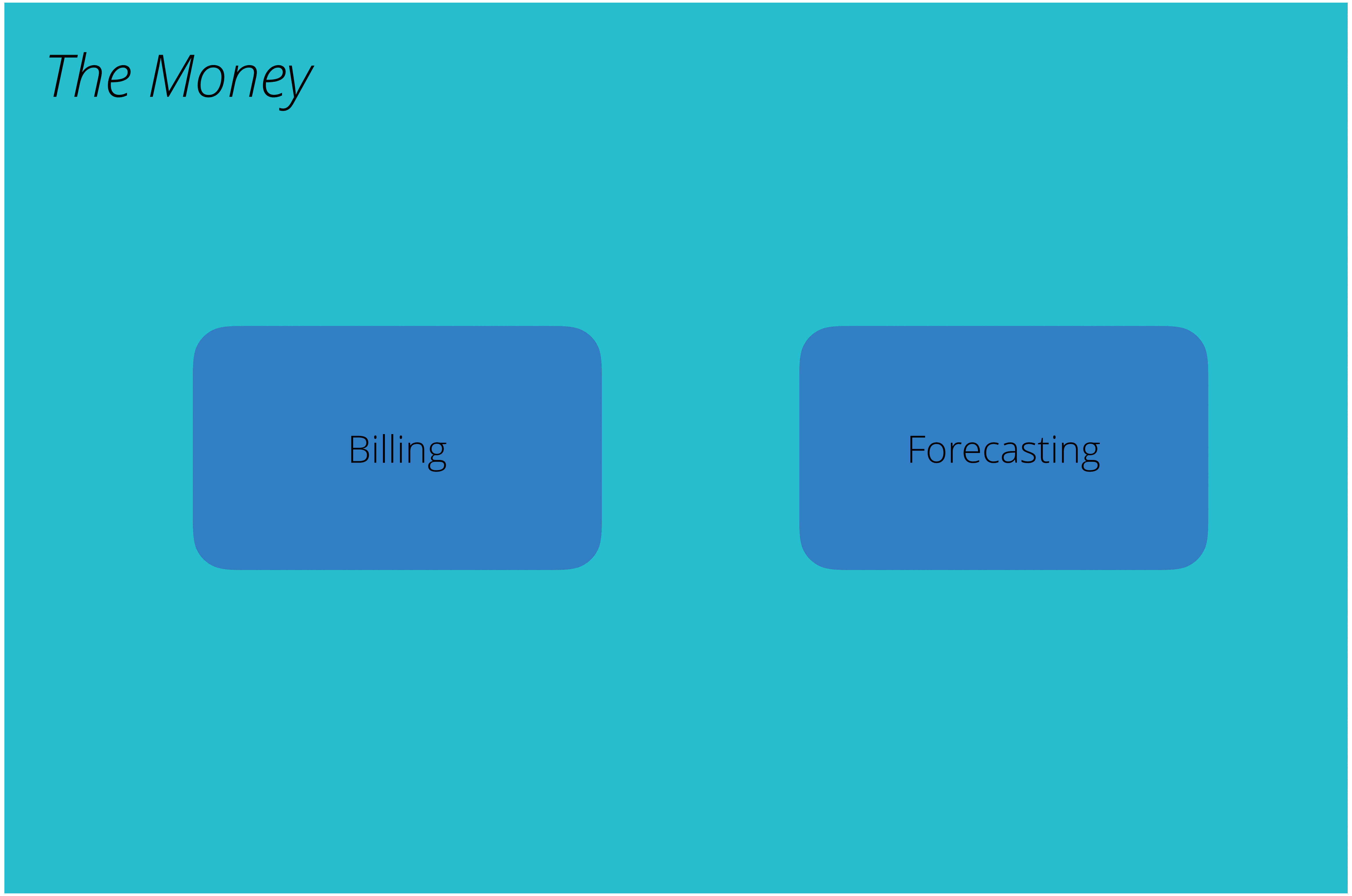
The **what** of the business, not the **how**

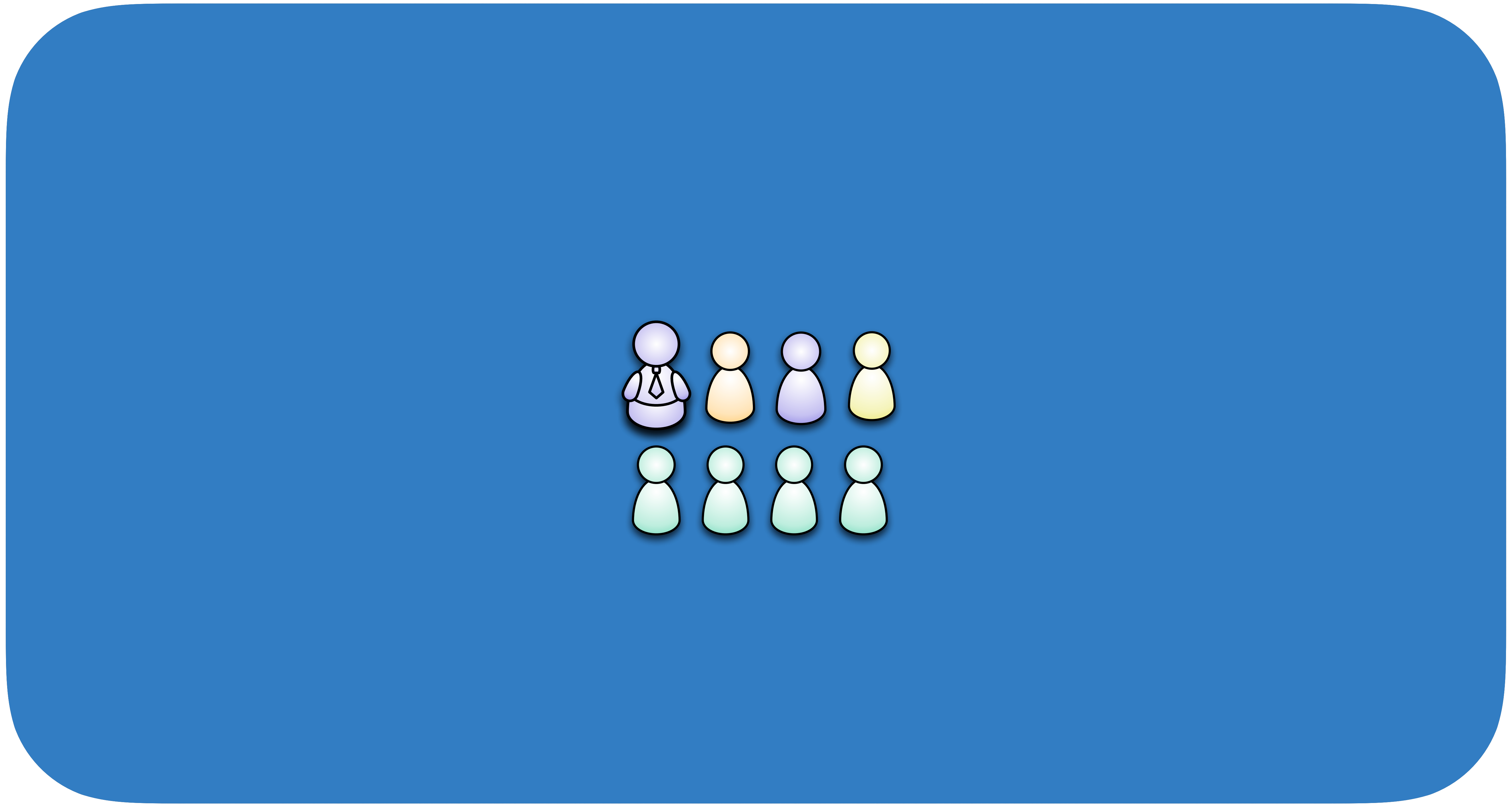
property company

Commercial property

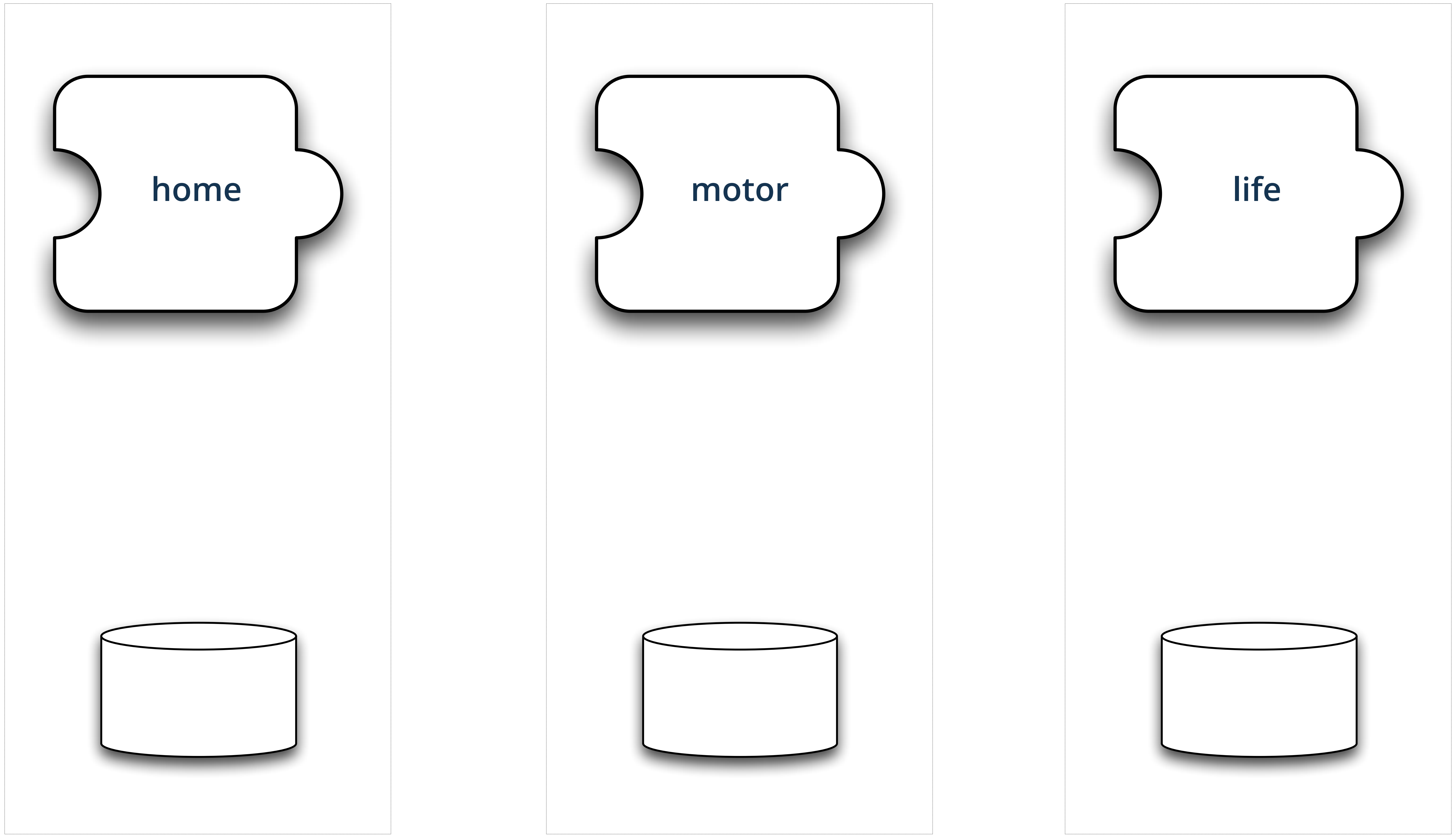
Residential property

The Money


The Money
Billing
Forecasting



insurance company



insurance company

home

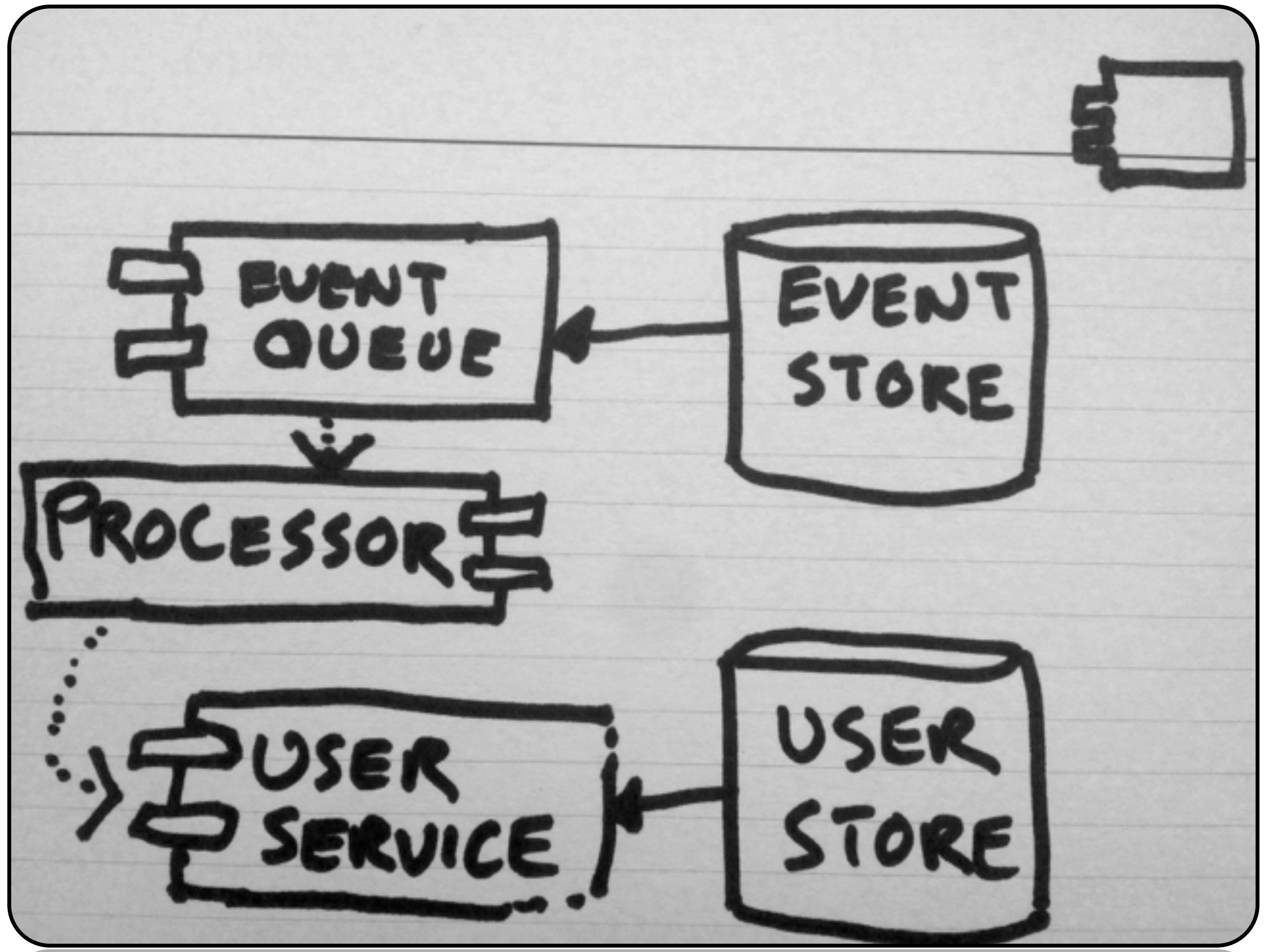
motor

life

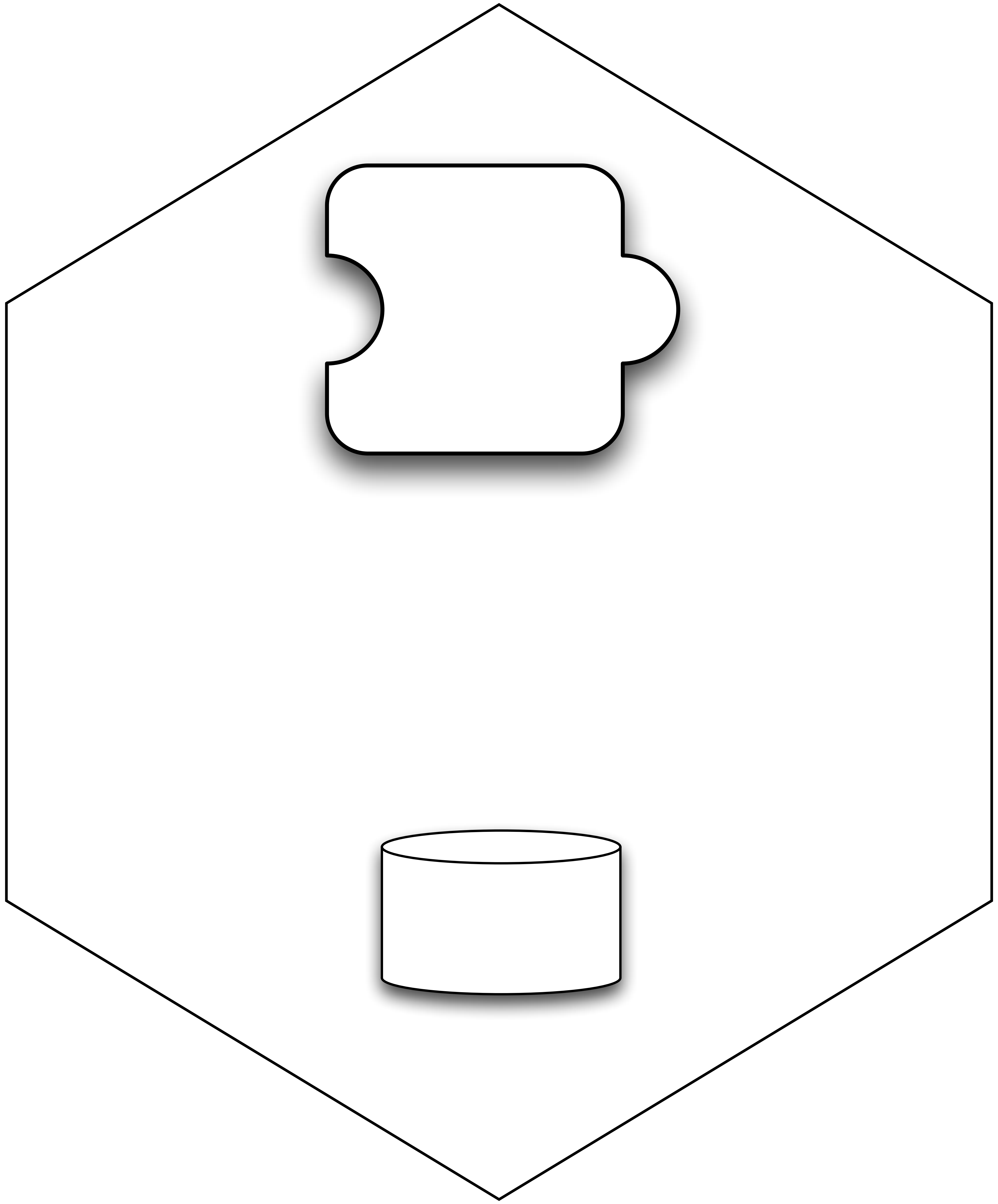
and cross-cutting capabilities

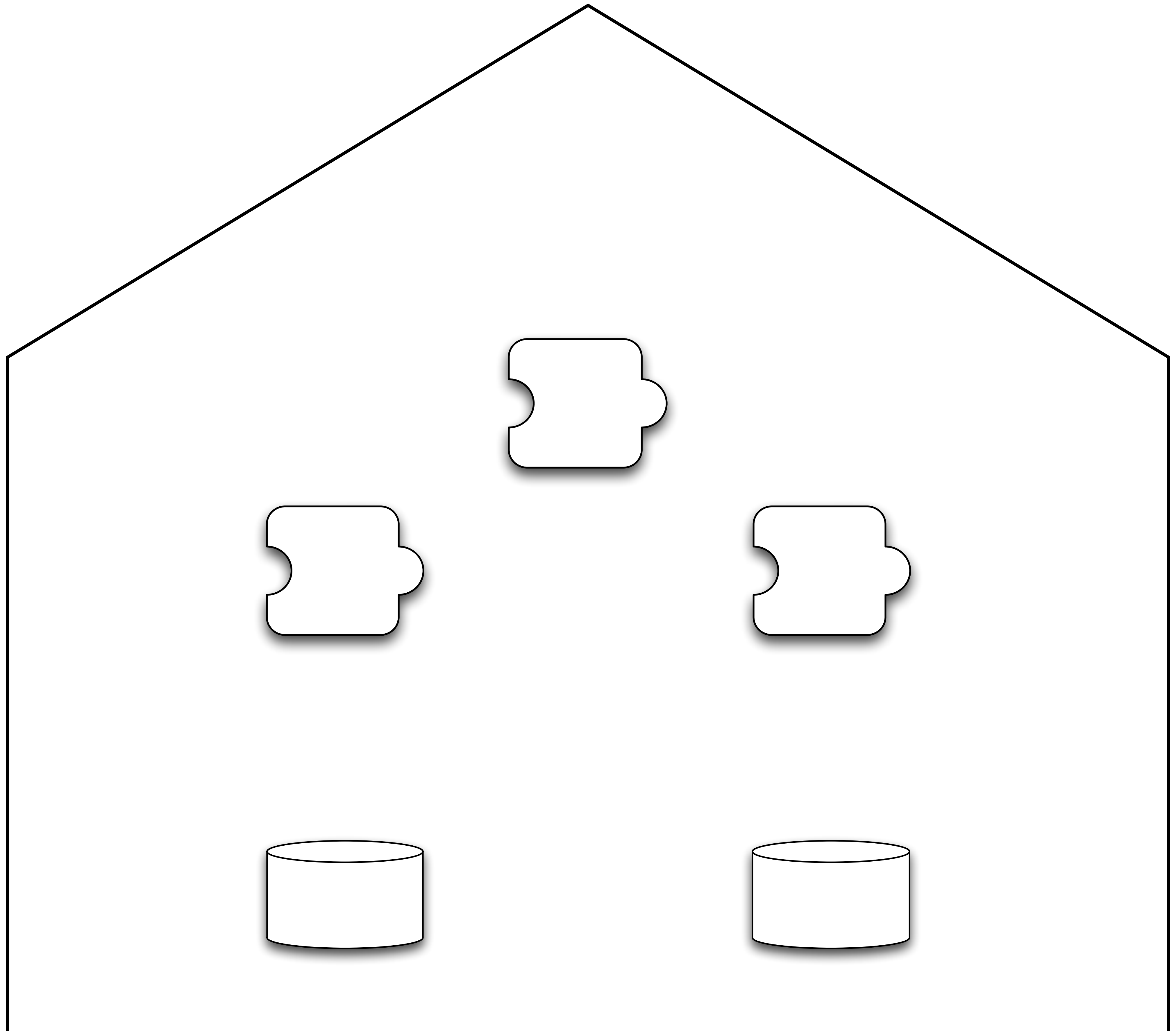
my account

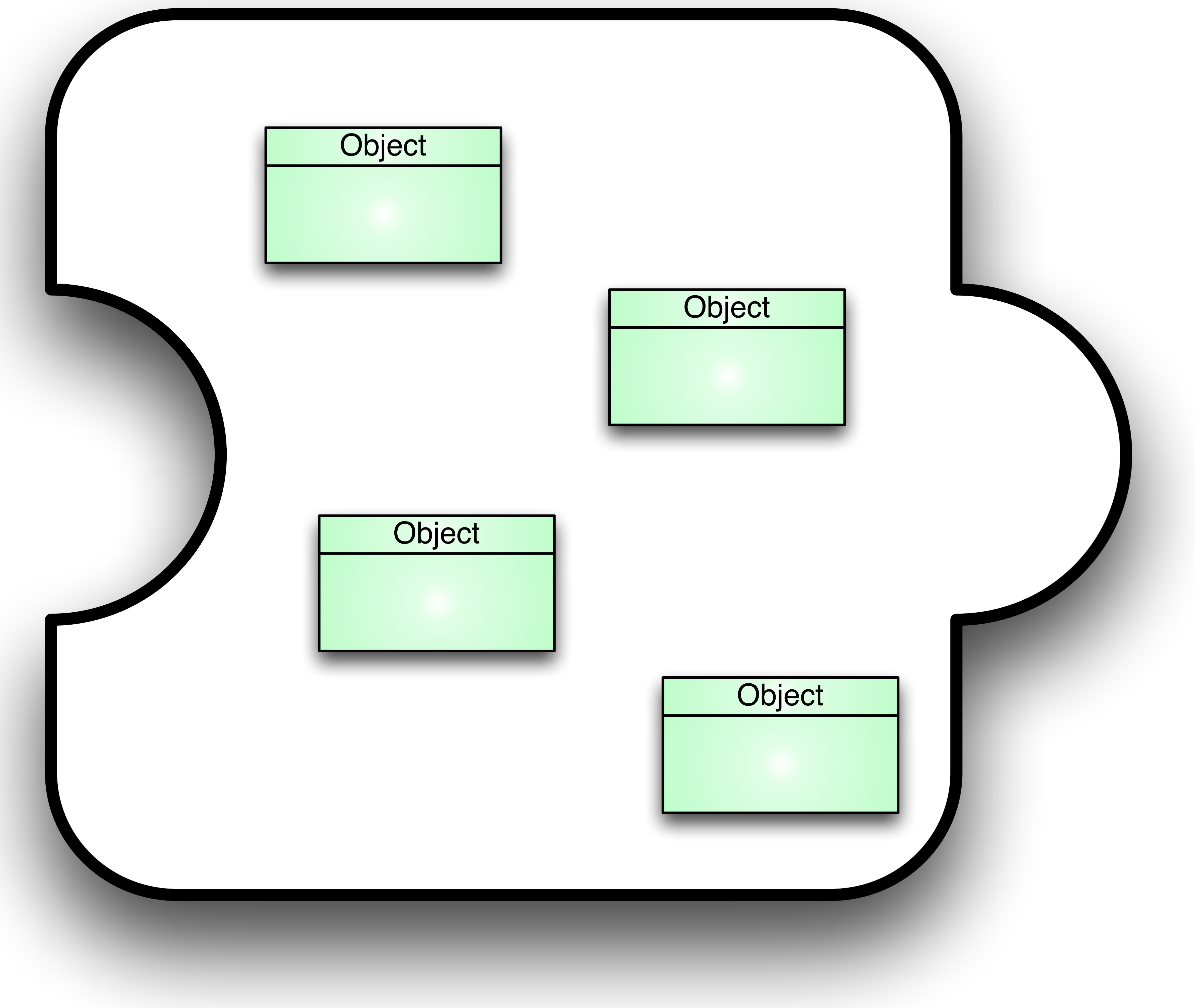
Each capability decomposed into smaller sub-domains based on your functional and cross-functional needs

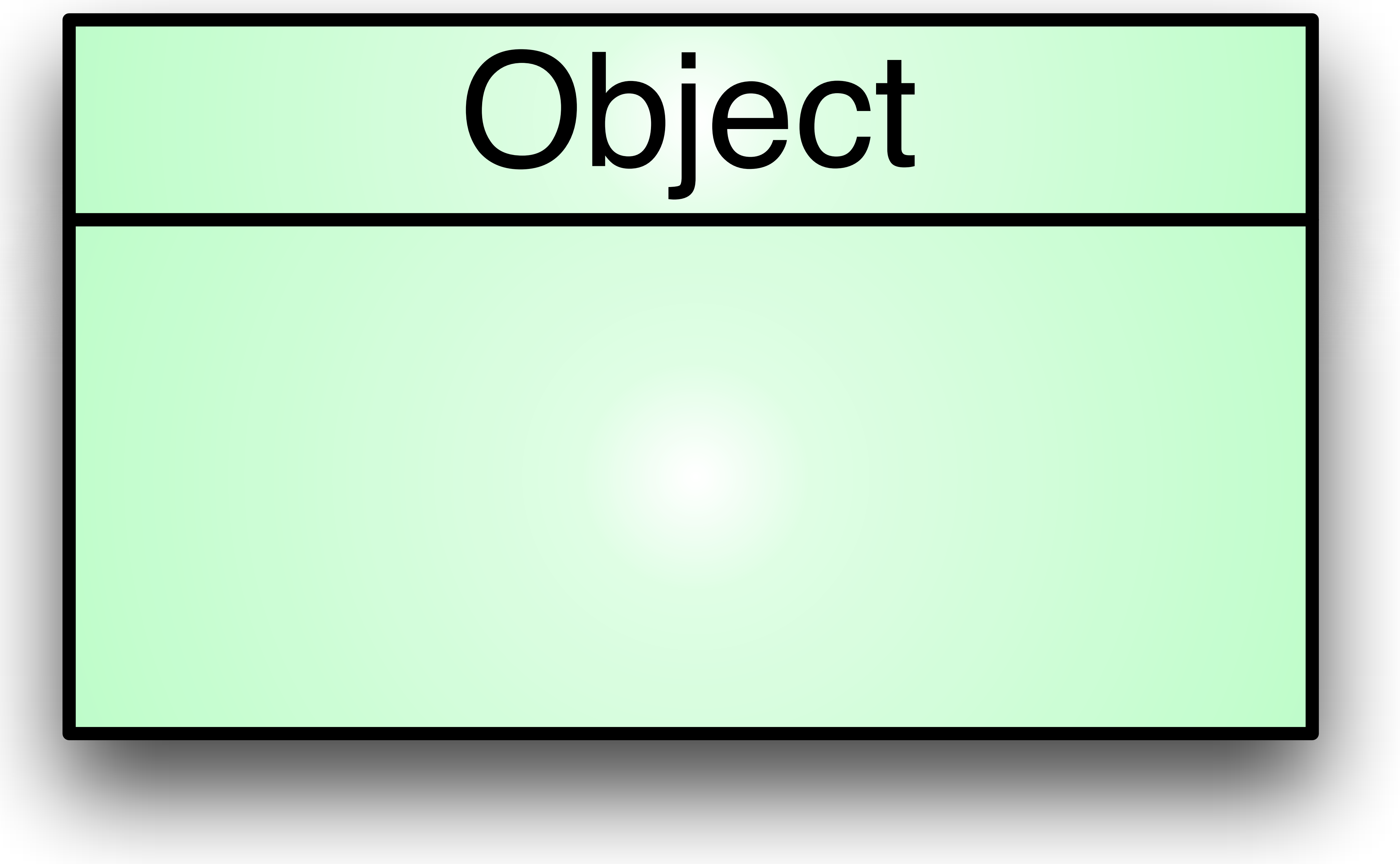


How **big** *are they?*

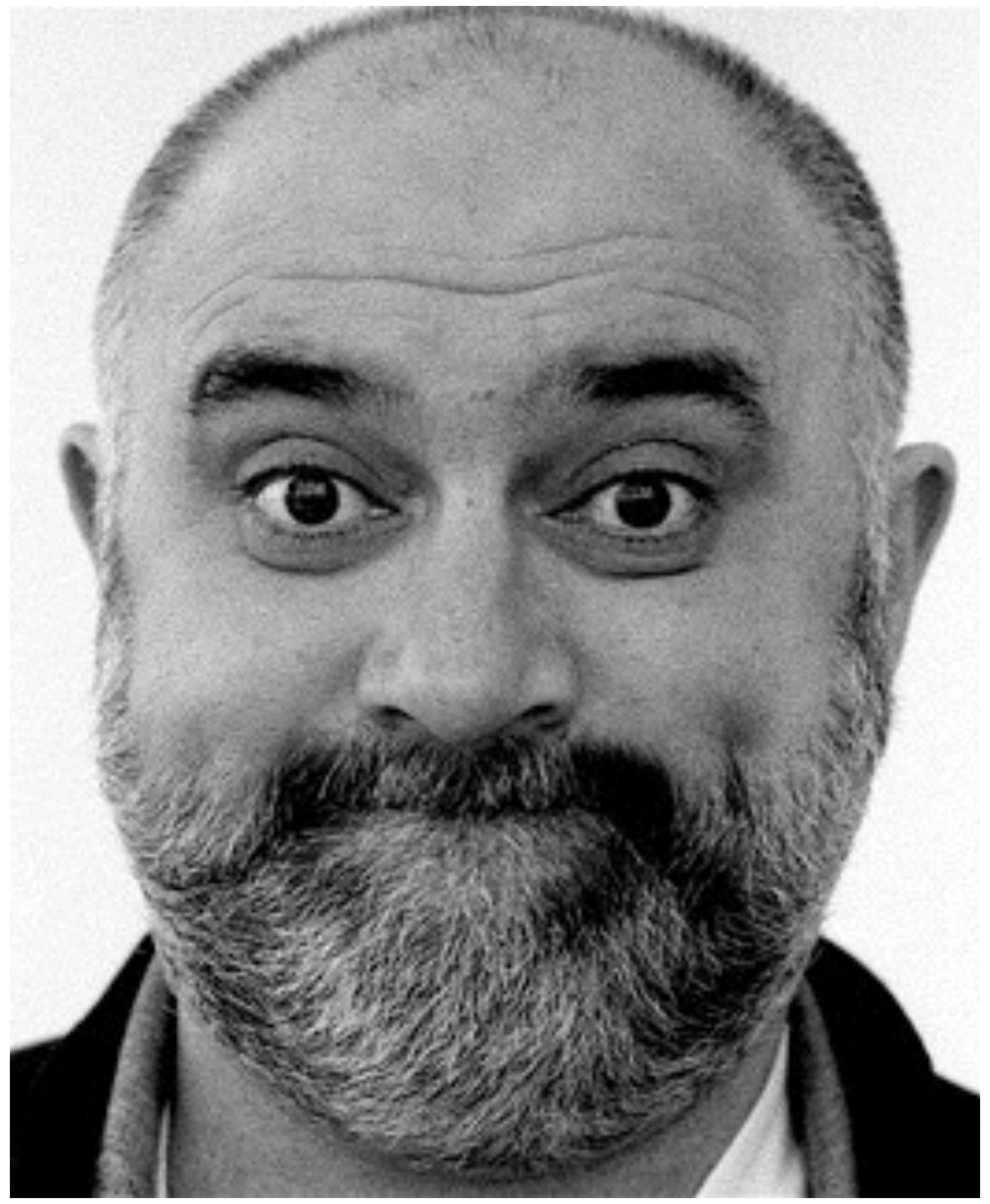




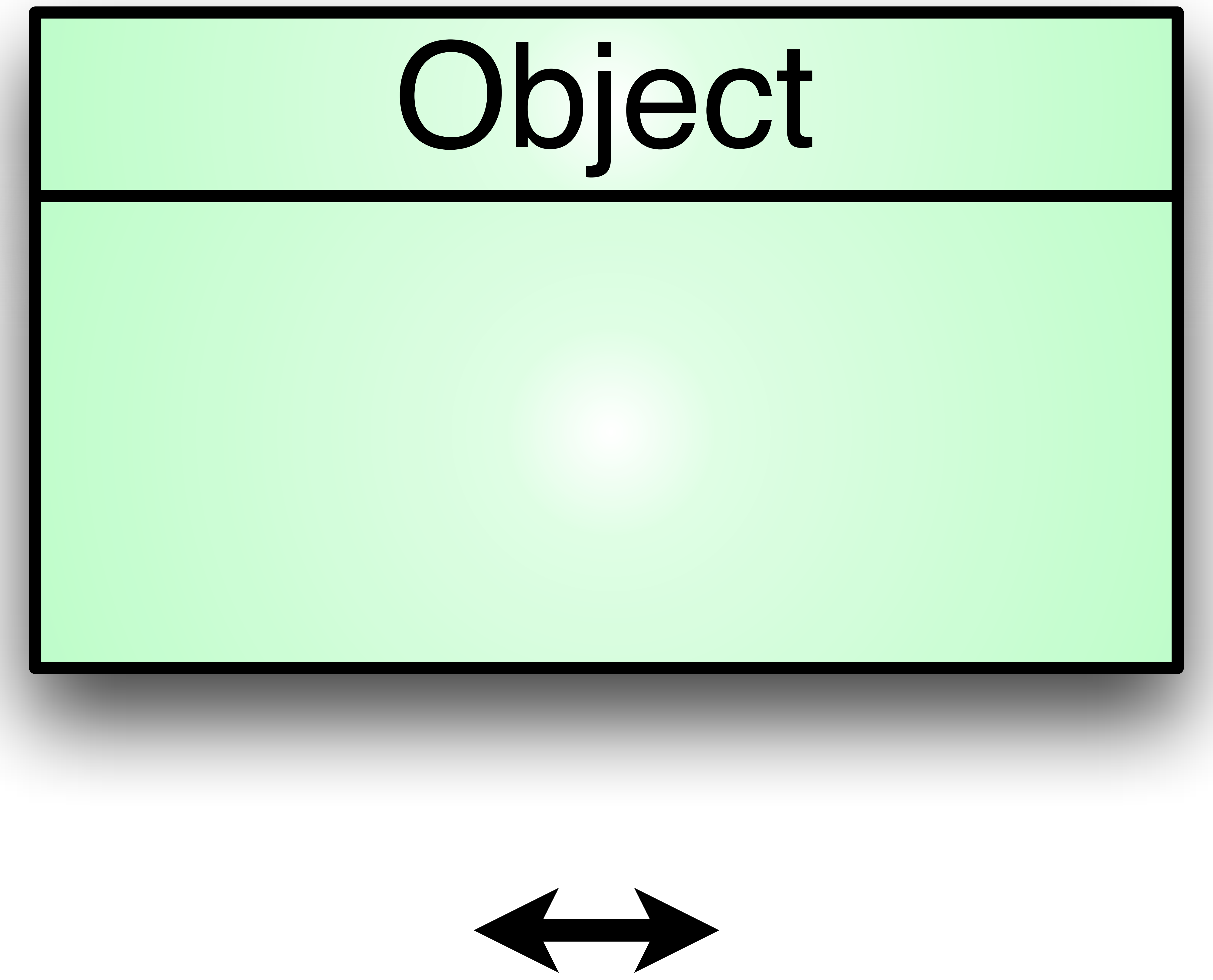

Object
Object
Object
Object

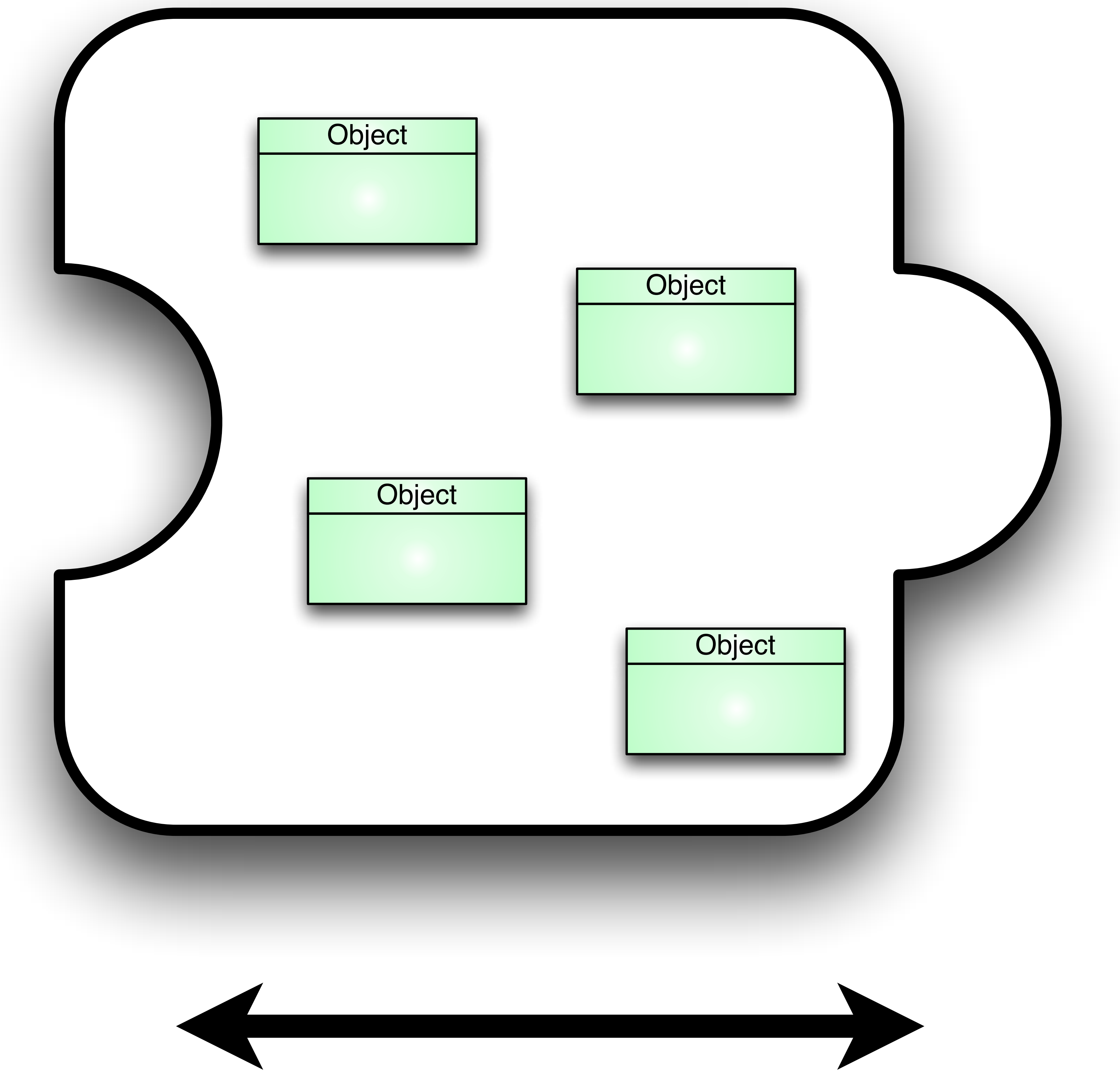


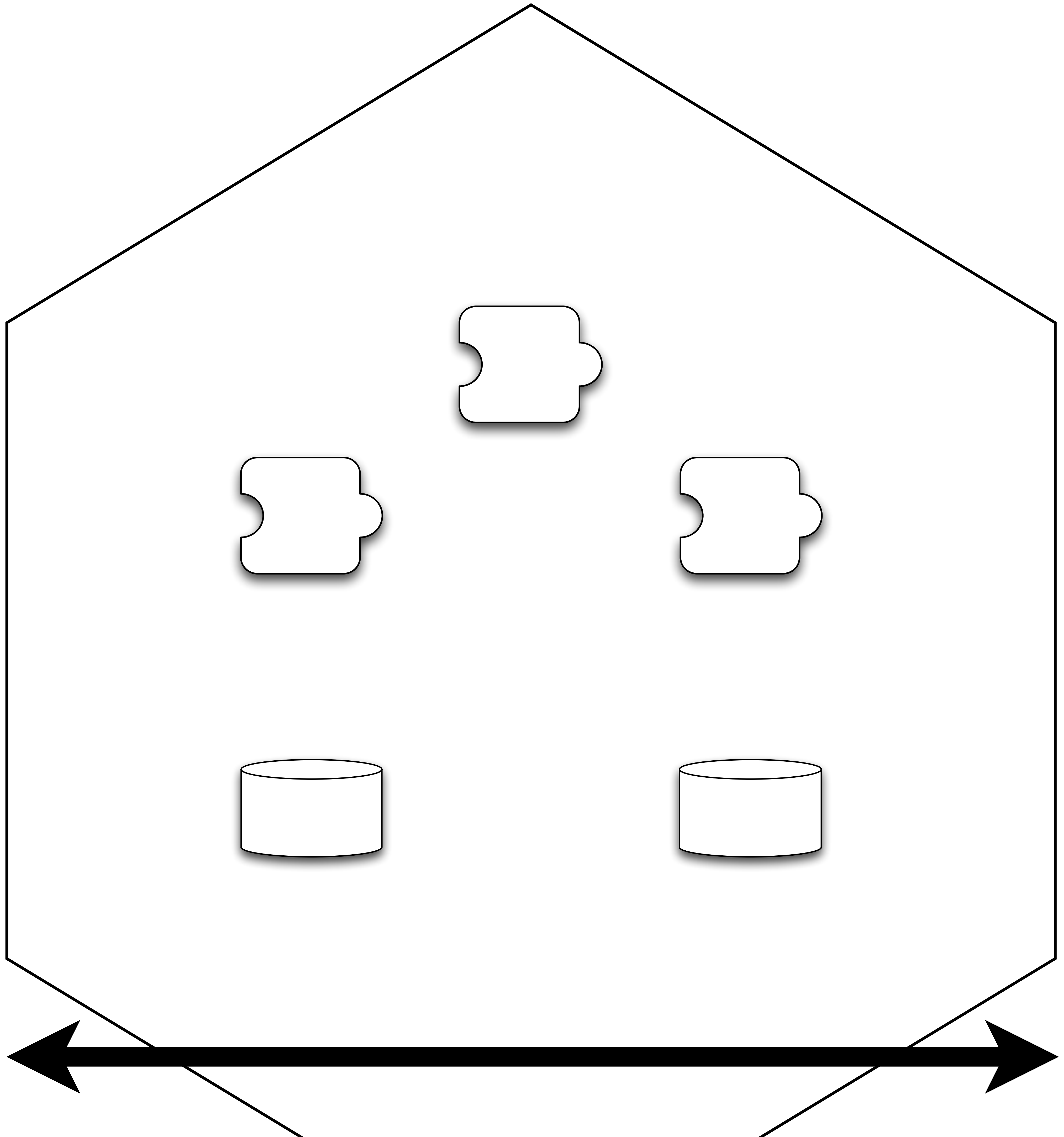
“objects should be no bigger than my head”

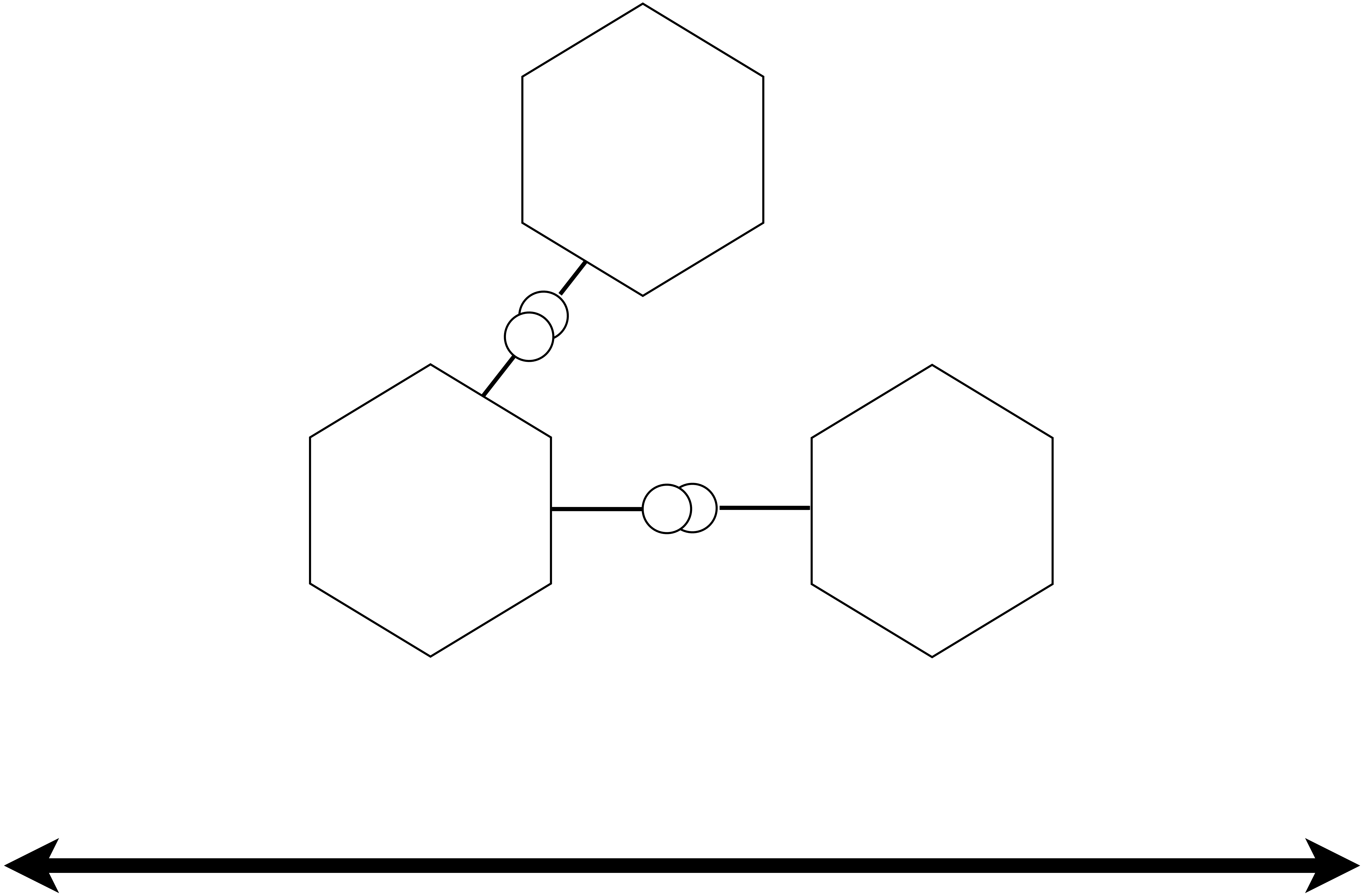


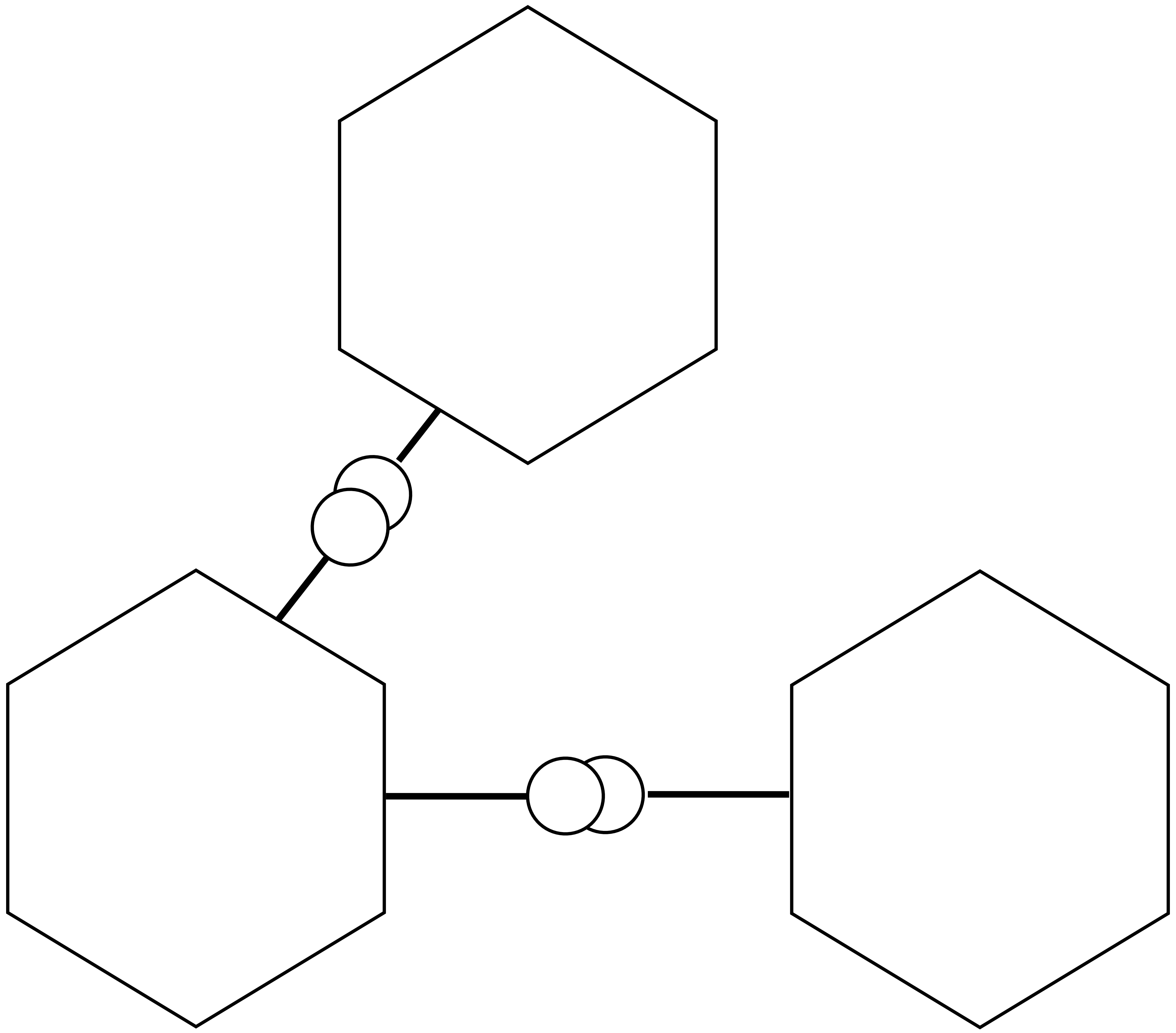
AND WHILE I HAVE A GIANT HEAD, ITS NOT FULL OF MUCH STUFF SO THATS OK...


Object


Object
Object
Object
Object







AS WE CHUNK UP DOMAINS, EACH DOMAIN SHOULD BE SMALL ENOUGH TO FIT IN MY HEAD

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

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://www.flickr.com/photos/photophilde/4527076709/

The Web

“be of the web, not behind the web”

Ian Robinson, author, REST in Practice

characteristics of microservices

componentisation via services

organised around business capabilities

decentralised data management

products not projects

decentralised governance

smart endpoints and dumb pipes

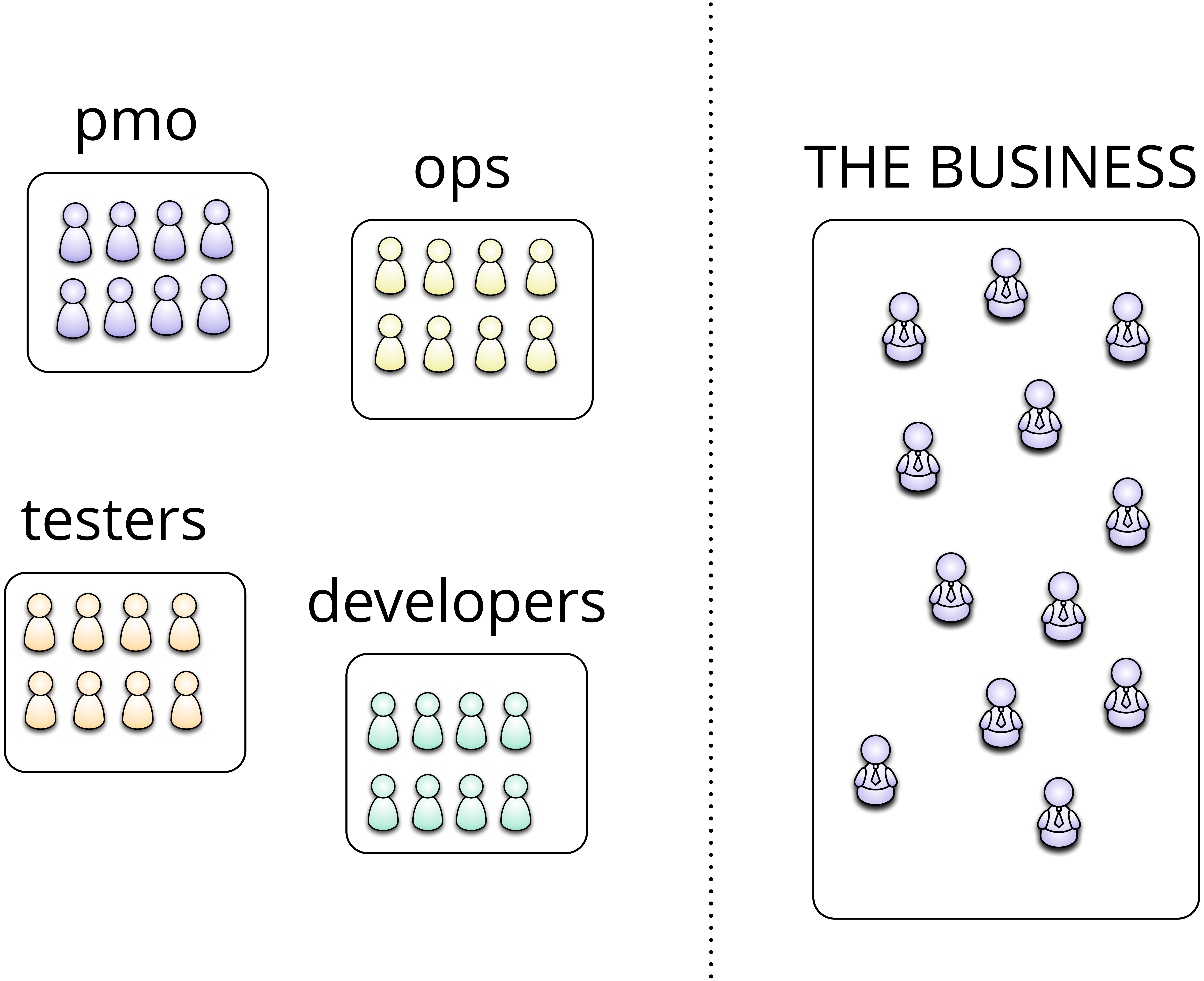
evolutionary design

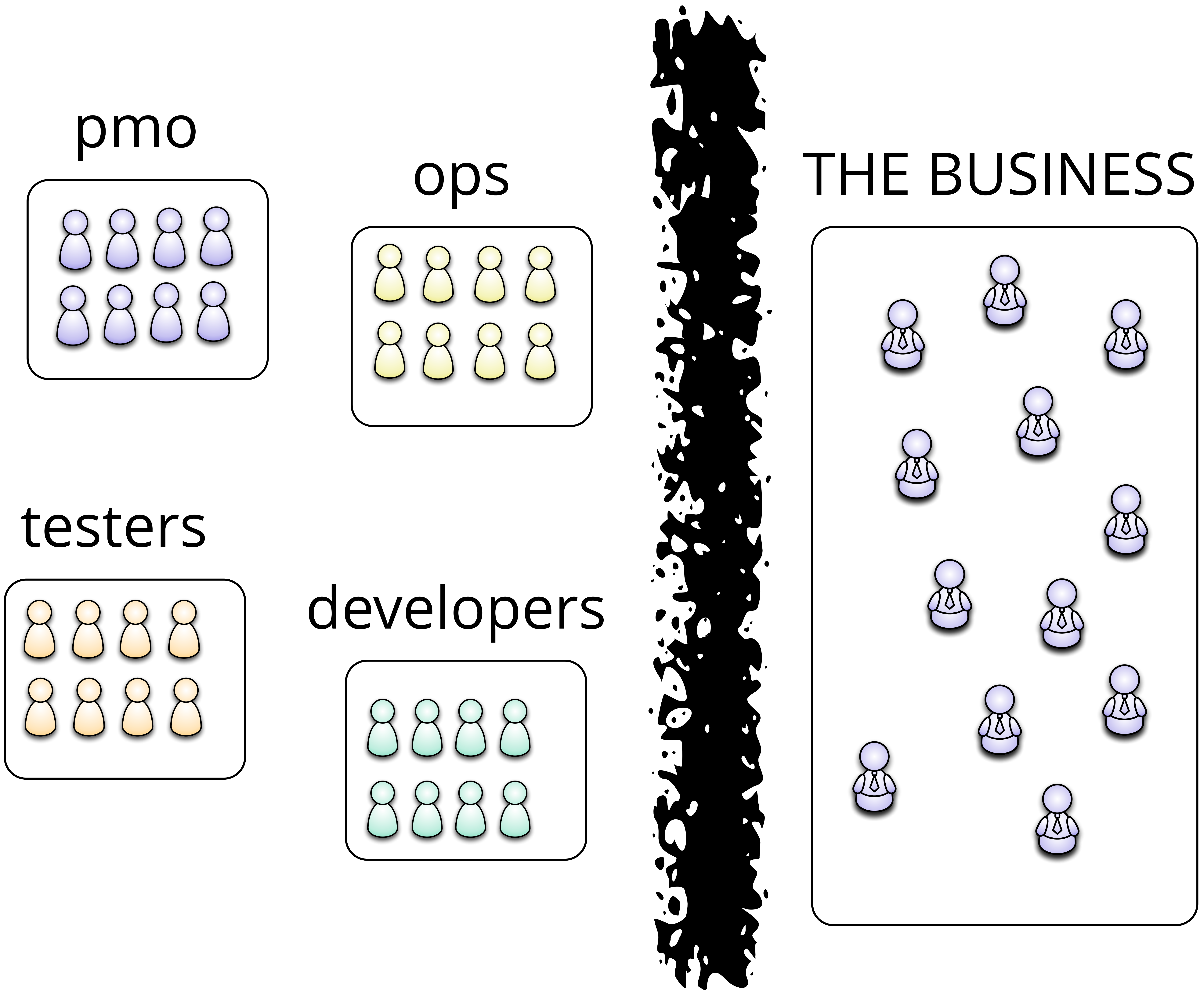
infrastructure automation

designed for failure

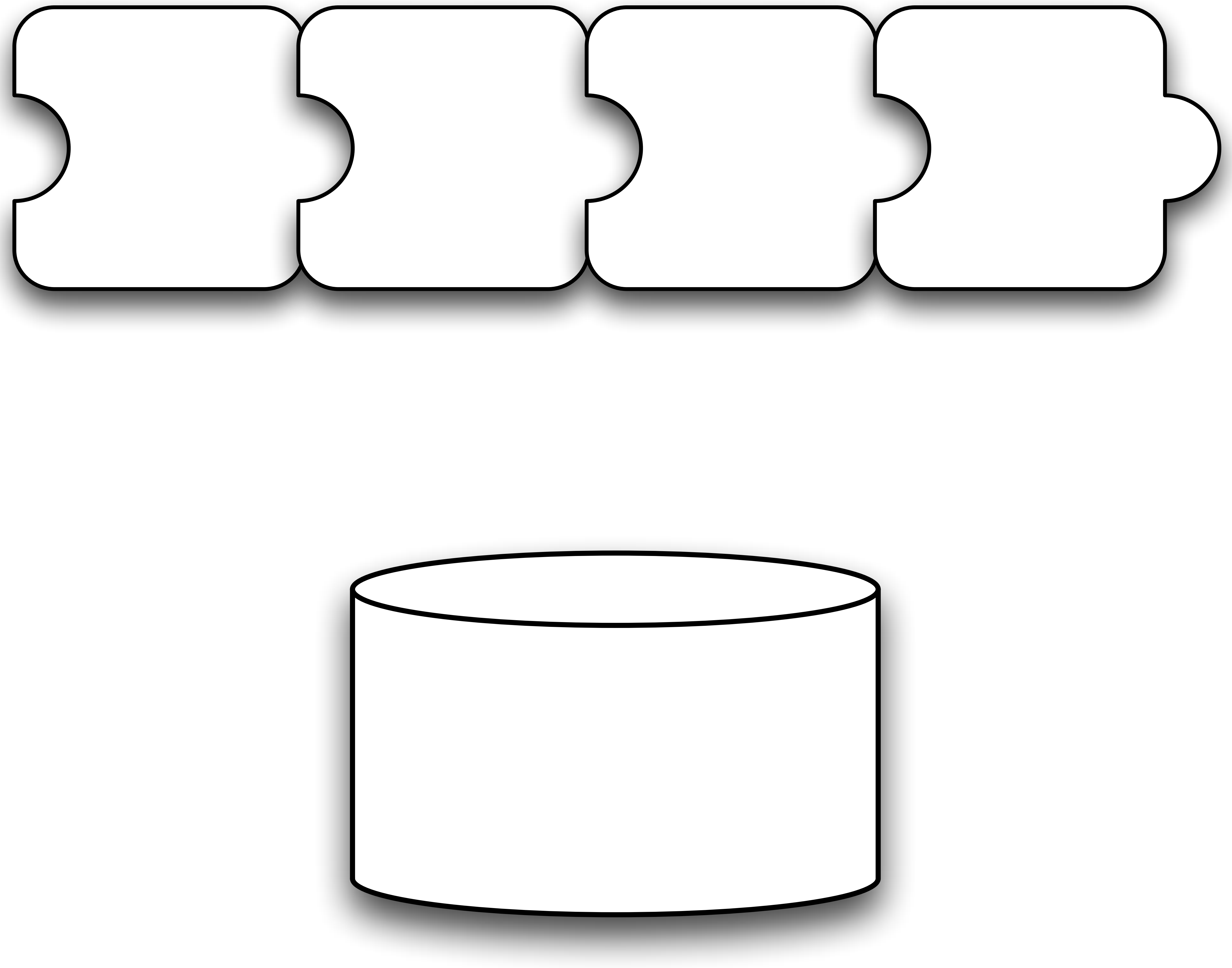
characteristics of microservices

- *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*

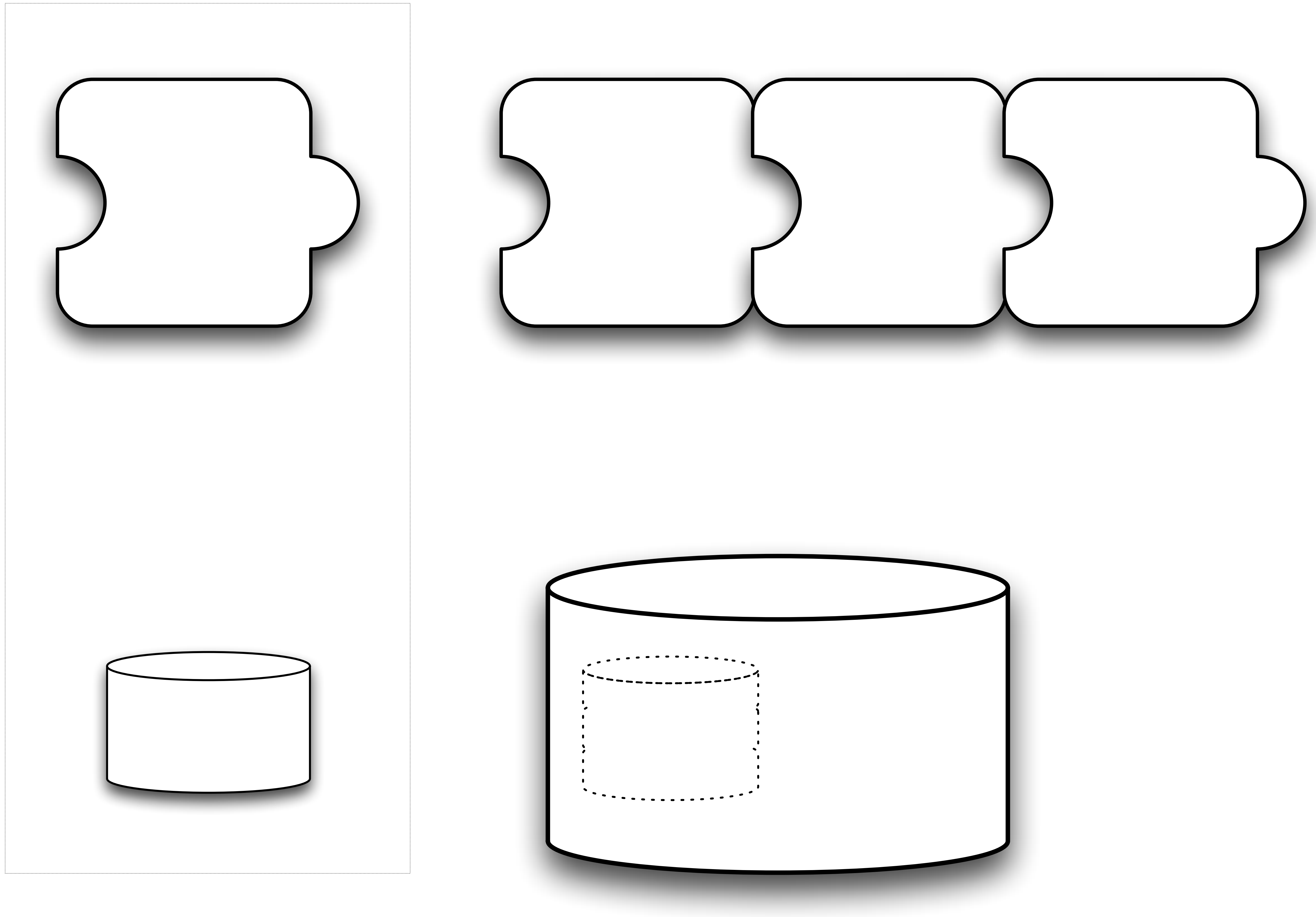

pmo
ops
testers
developers
THE BUSINESS


pmo
ops
THE BUSINESS
testers
developers

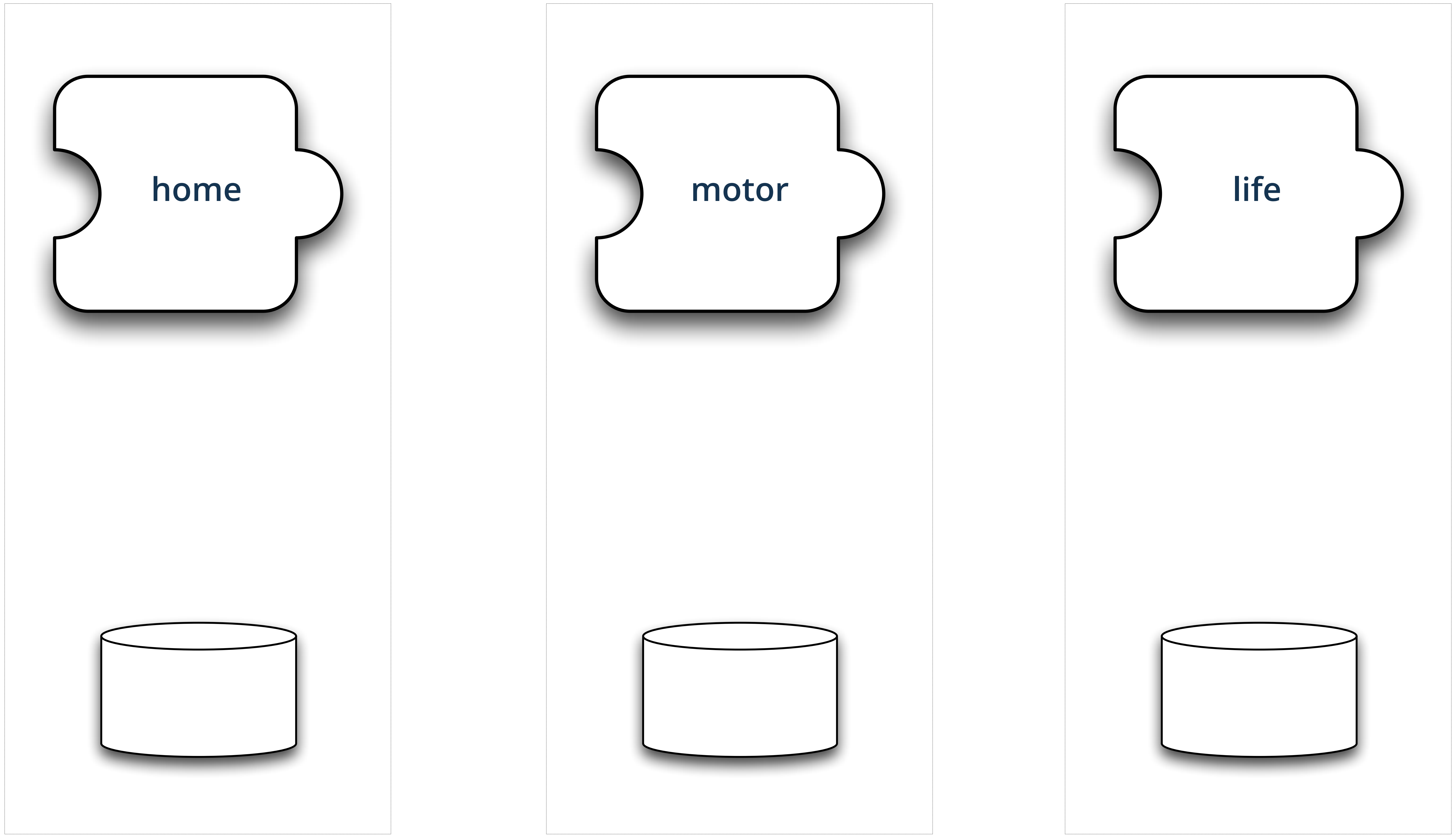
insurance company



insurance company



separate lines of business



separate lines of business

home

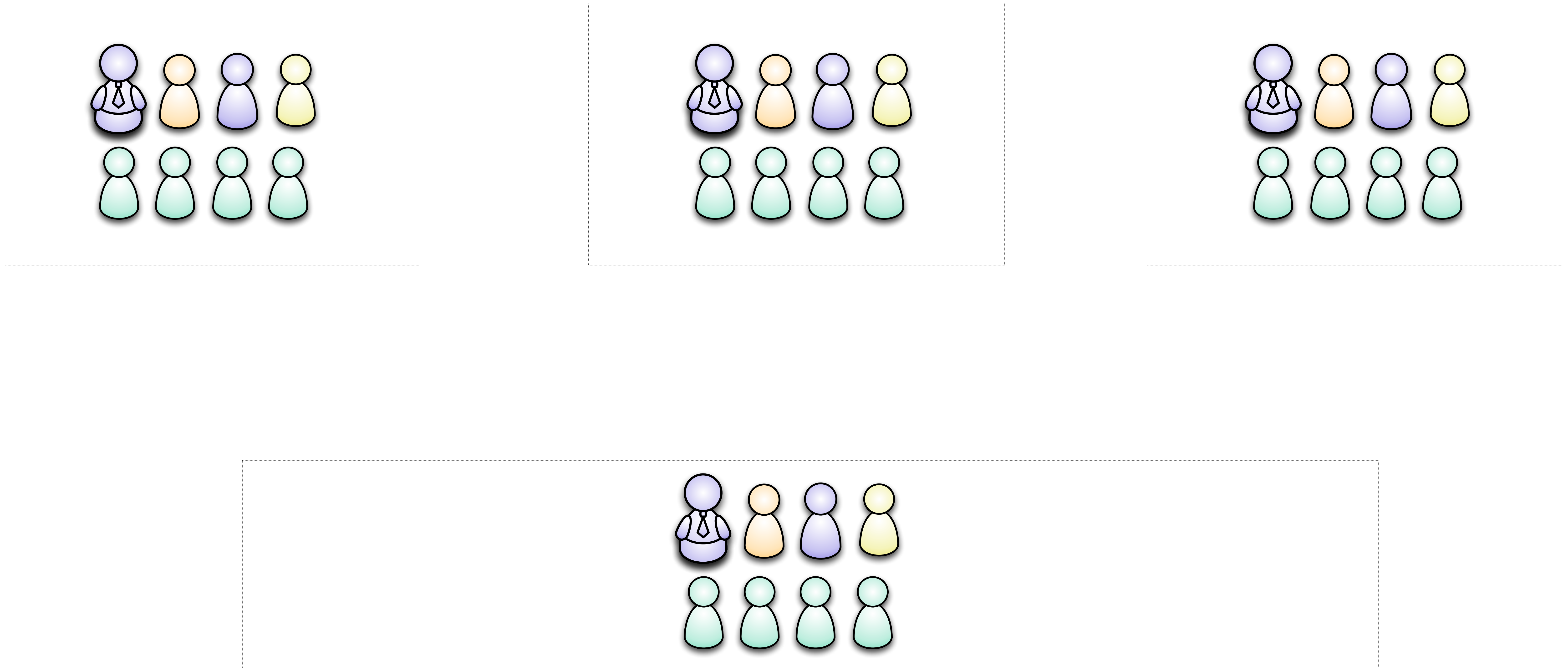
motor

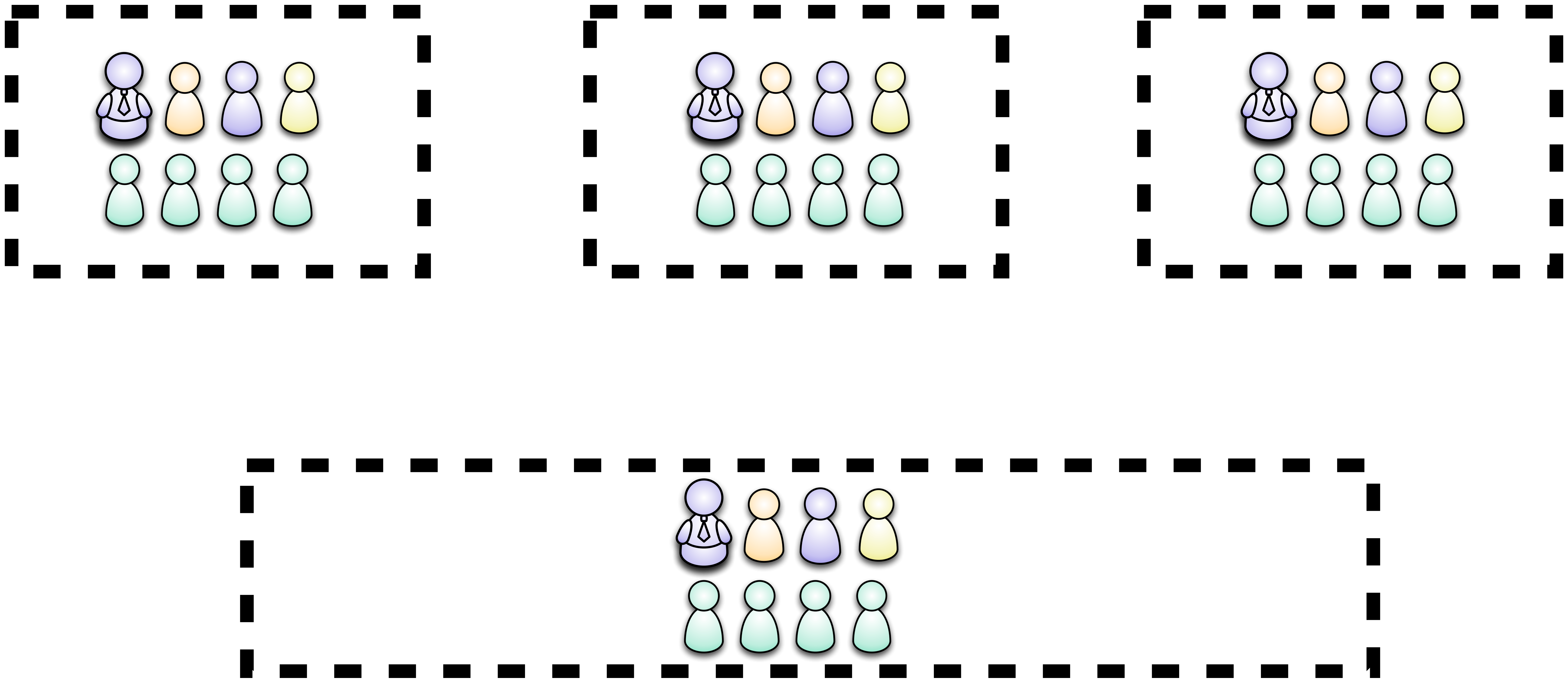
life

and cross-cutting capabilities

my account

cross-functional teams delivering lines of business





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

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

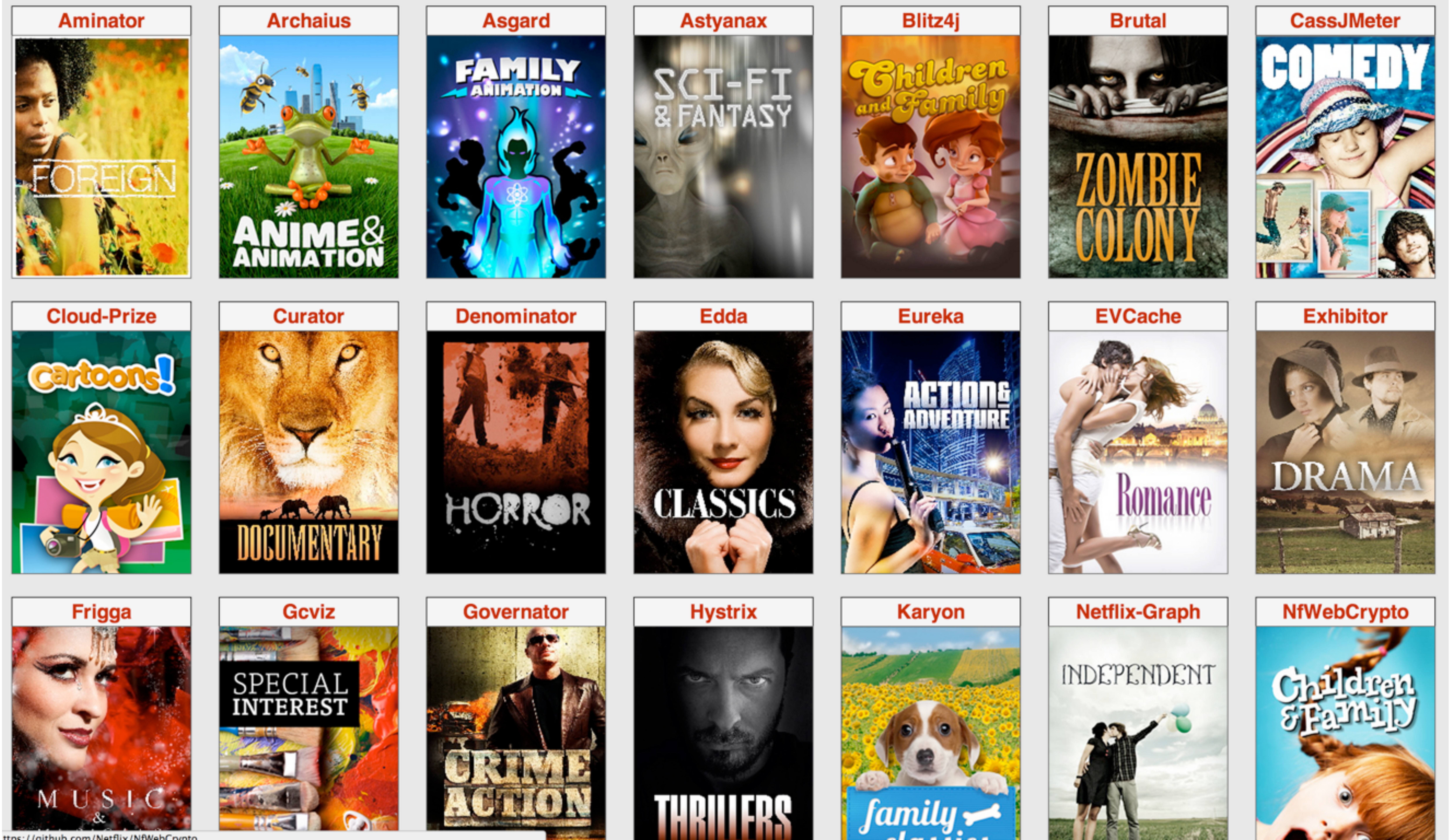
infrastructure automation

designed for failure

military

Share tools, don’t enforce standards

MAKE IT EASY TO DO THE RIGHT THING


Aminator
FOREIGN
Archaius
ANIME & ANIMATION
Asgard
FAMILY ANIMATION
Astyanax
SCI-FI & FANTASY
Blitz4j
Children and Family
Brutal
ZOMBIE COLONY
CassJMeter
COMEDY
Cloud-Prize
Cartoons!
Curator
DOCUMENTARY
Denominator
HORROR
Edda
CLASSICS
Eureka
ACTION & ADVENTURE
EVCache
Romance
Exhibitor
DRAMA
Frigga
MUSIC
Gcviz
SPECIAL INTEREST
Governator
CRIME ACTION
Hystrix
THRILLERS
Karyon
family
Netflix-Graph
INDEPENDENT
NfWebCrypto
Children & Family
ttps://github.com/Netflix/NfWebCrypto

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*

characteristics of microservices

componentisation via services

organised around business capabilities

decentralised data management

products not projects

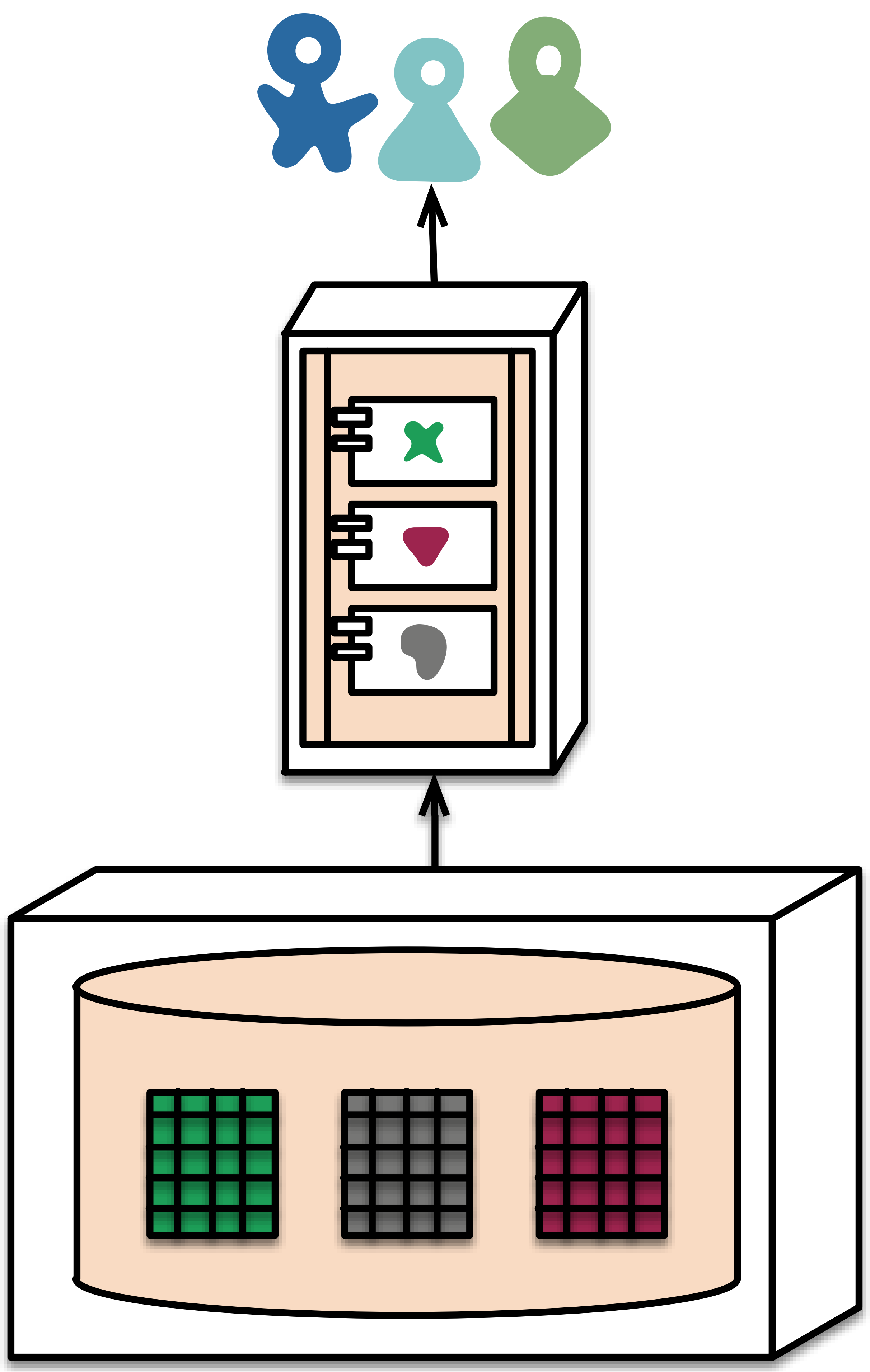
decentralised governance

smart endpoints and dumb pipes

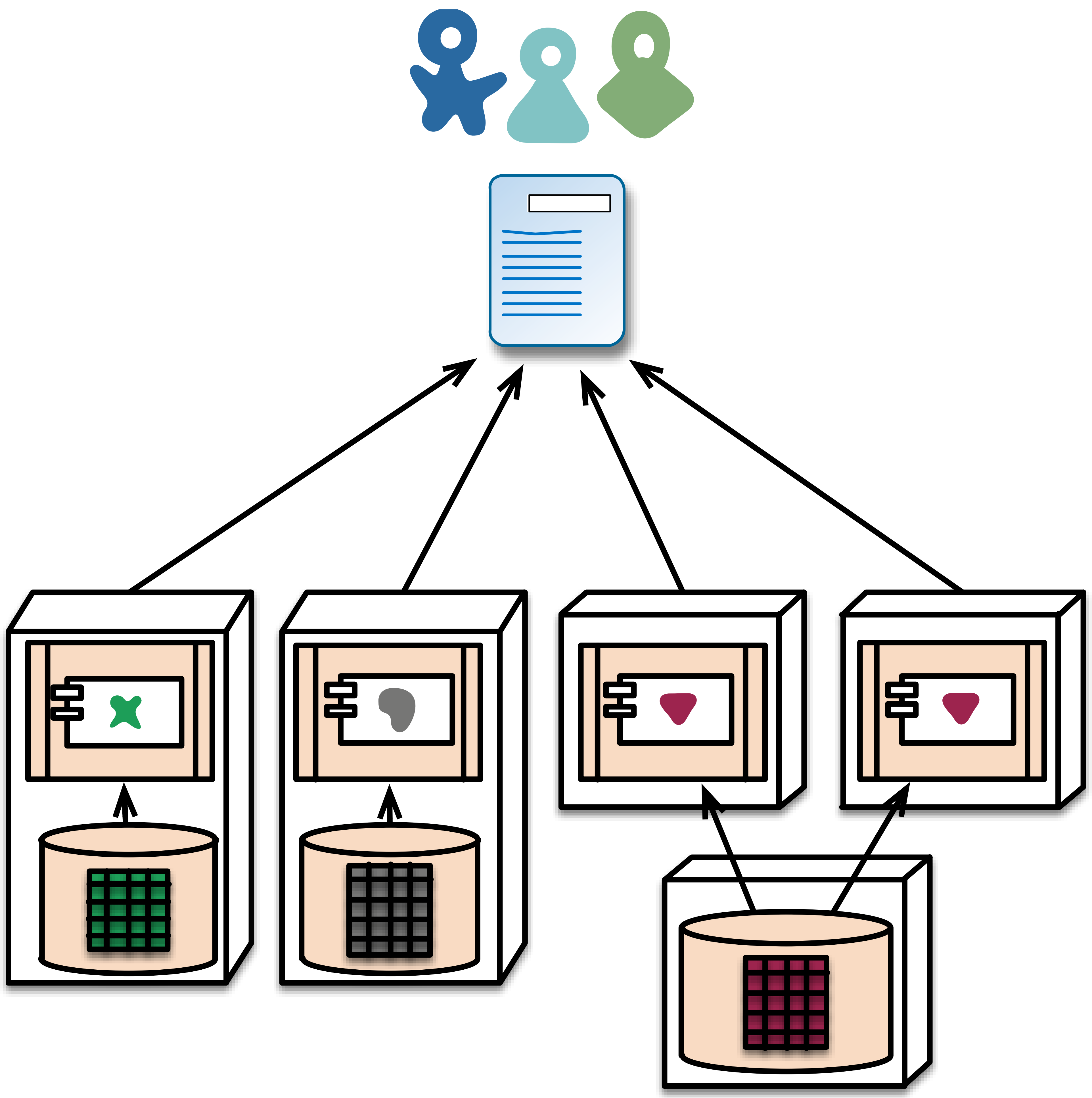
evolutionary design

infrastructure automation

designed for failure



monolith - single database



microservices - application databases

Capabilities own their own data

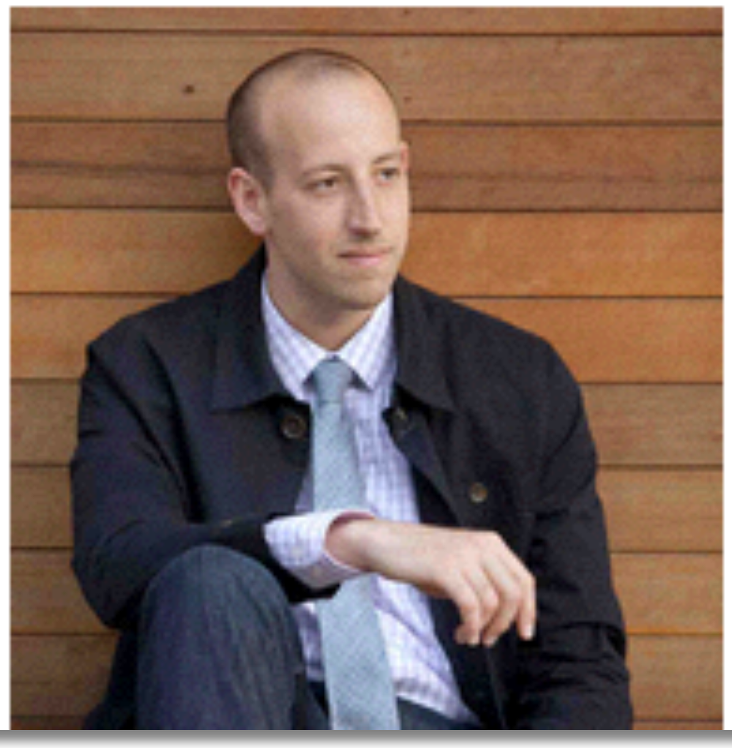
(can be cached elsewhere with appropriate policies)

what about transactions in this model?

Blog About Speaking Archives Contact

Follow Nathan on

- Twitter
- GitHub
- LinkedIn
- Blog RSS



« Early access edition of my book is available | Main | My talks at POSSCON »

How to beat the CAP theorem

THURSDAY, OCTOBER 13, 2011

The CAP theorem states a database cannot guarantee consistency, availability, and partition-tolerance at the same time. But you can't sacrifice partition-tolerance (see here and here), so you must make a tradeoff between availability and consistency. Managing this tradeoff is a central focus of the NoSQL movement.

Consistency means that after you do a successful write, future reads will always take that write into account. Availability means that you can always read and write to the system. During a partition, you can only have one of these properties.

Systems that choose consistency over availability have to deal with some awkward issues. What do you do when the database isn't available? You can try buffering writes for later, but you risk losing those writes if you lose the machine with the

http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

“As bad as anything else”

Beating the CAP Theorem Checklist

```
Your ( ) tweet ( ) blog post ( ) marketing material ( ) online comment
advocates a way to beat the CAP theorem. Your idea will not work. Here is why
it won't work:

( ) you are assuming that software/network/hardware failures will not happen
( ) you pushed the actual problem to another layer of the system
( ) your solution is equivalent to an existing one that doesn't beat CAP
( ) you're actually building an AP system
( ) you're actually building a CP system
( ) you are not, in fact, designing a distributed system

Specifically, your plan fails to account for:

( ) latency is a thing that exists
( ) high latency is indistinguishable from splits or unavailability
( ) network topology changes over time
( ) there might be more than 1 partition at the same time
( ) split nodes can vanish forever
( ) a split node cannot be differentiated from a crashed one by its peers
( ) clients are also part of the distributed system
```

http://ferd.ca/beating-the-cap-theorem-checklist.html

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

characteristics of microservices

componentisation via services

organised around business capabilities

decentralised data management

products not projects

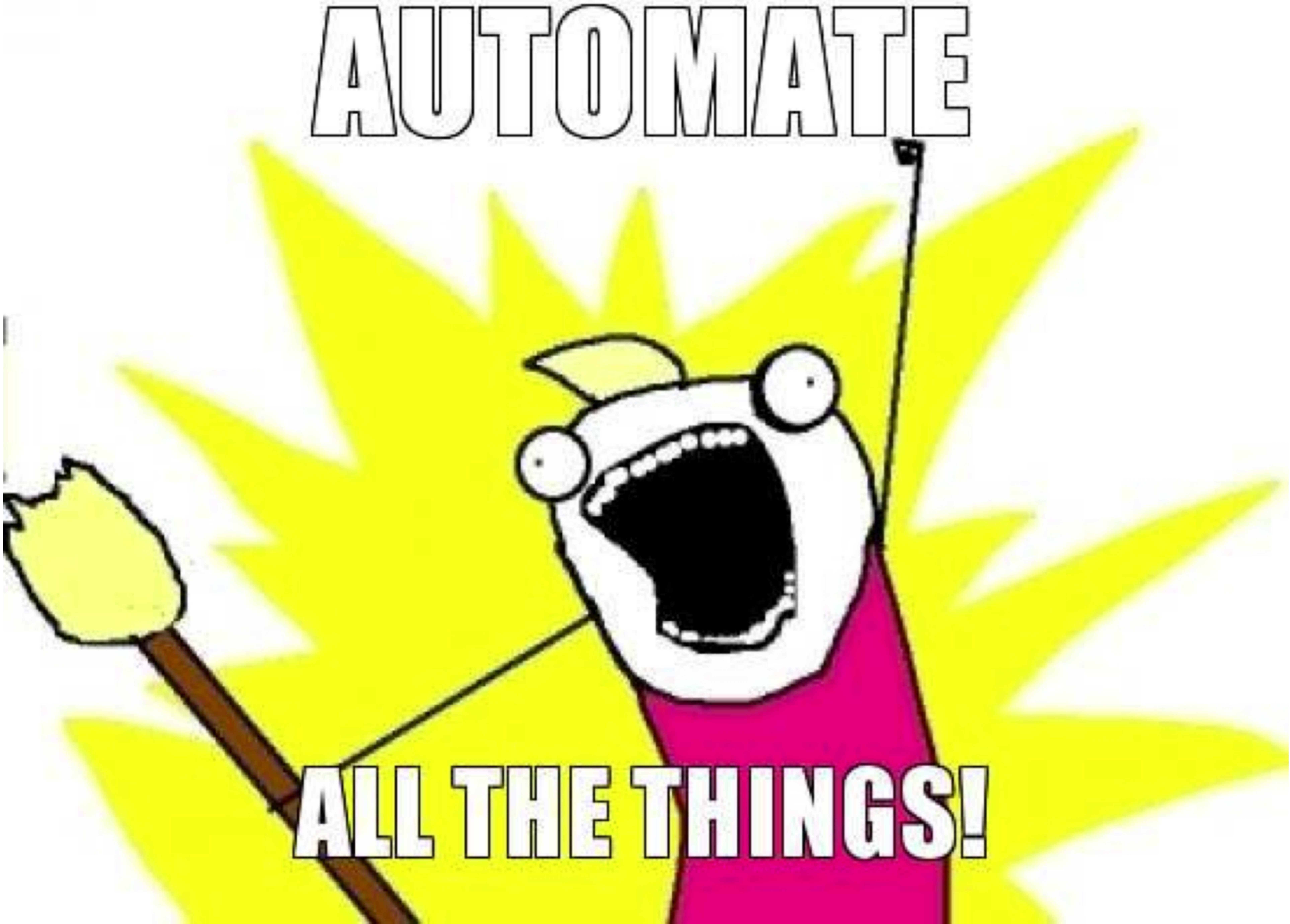
decentralised governance

smart endpoints and dumb pipes

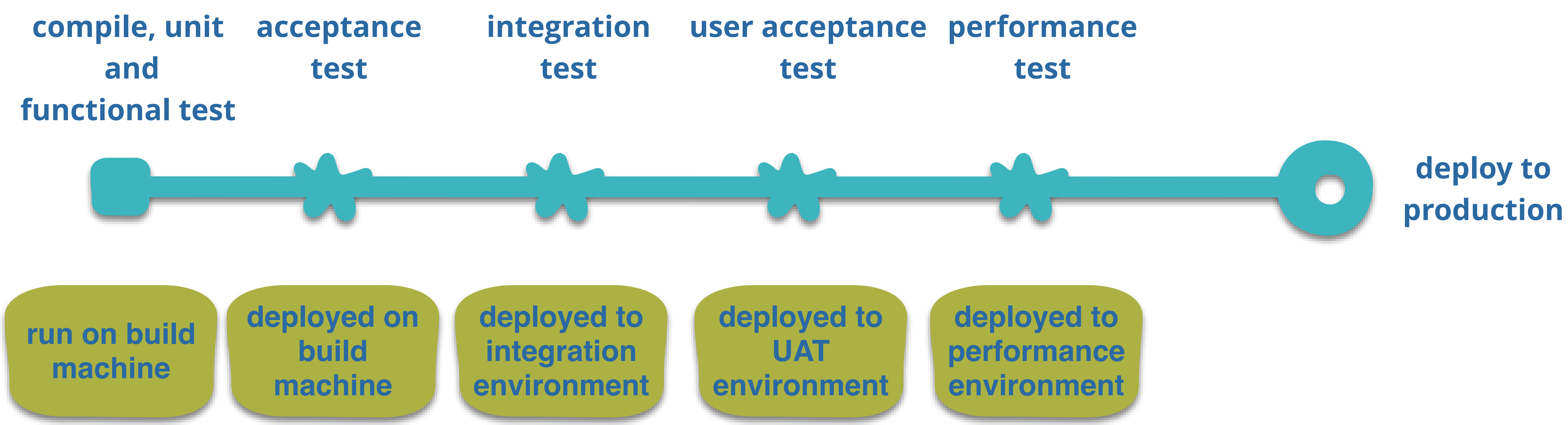
evolutionary design

infrastructure automation

designed for failure


AUTOMATE
ALL THE THINGS!

Fast Feedback



More Confidence

phoenix infrastructure

characteristics of microservices

componentisation via services

organised around business capabilities

decentralised data management

products not projects

decentralised governance

smart endpoints and dumb pipes

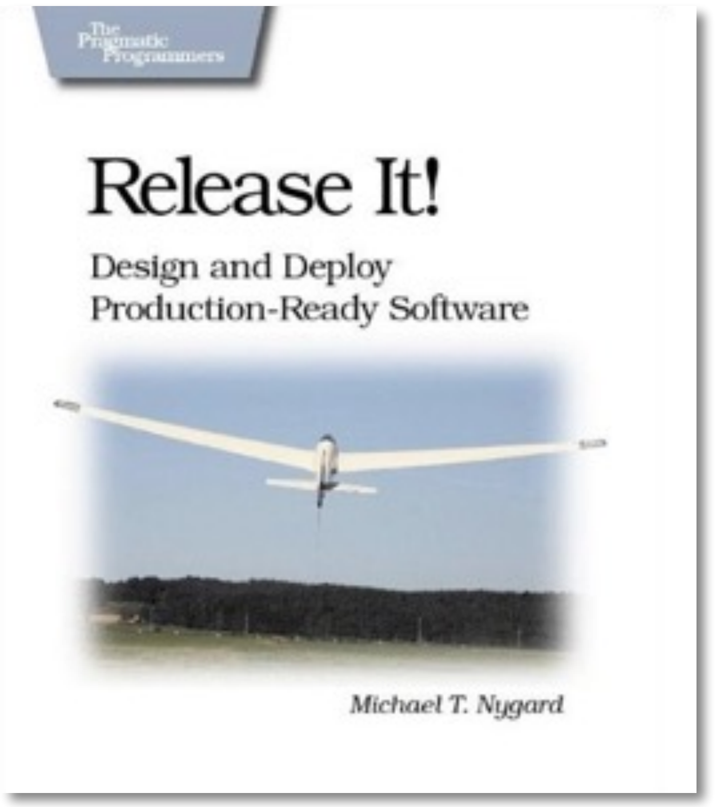
evolutionary design

infrastructure automation

designed for failure

“it pushes the accidental complexity into the infrastructure”

Martin Fowler



"Every socket, process, pipe, or remote procedure call can and will hang. Even database calls [...]"

M. Nygard, "Release It"

Chapter 3

Some (un)expected consequences

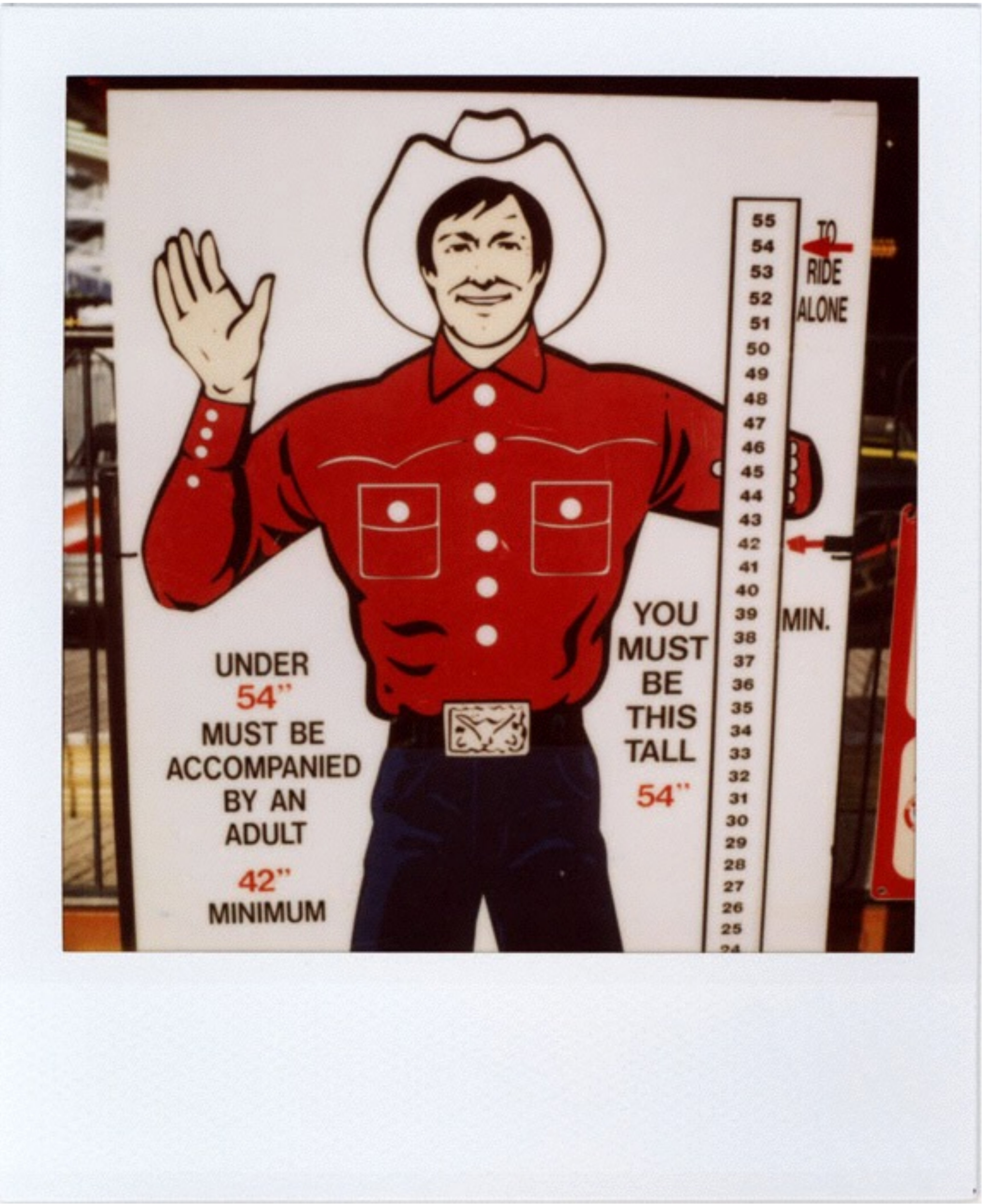
microservices **should** *allow us to go as*
"fast as possible"

be cheap to replace

be deployable on demand

be resilient on imperfect networks

but it's not as simple as that



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

Monitoring

Organisational Structure

Deployment

Integration

Testing

Architectural Safety

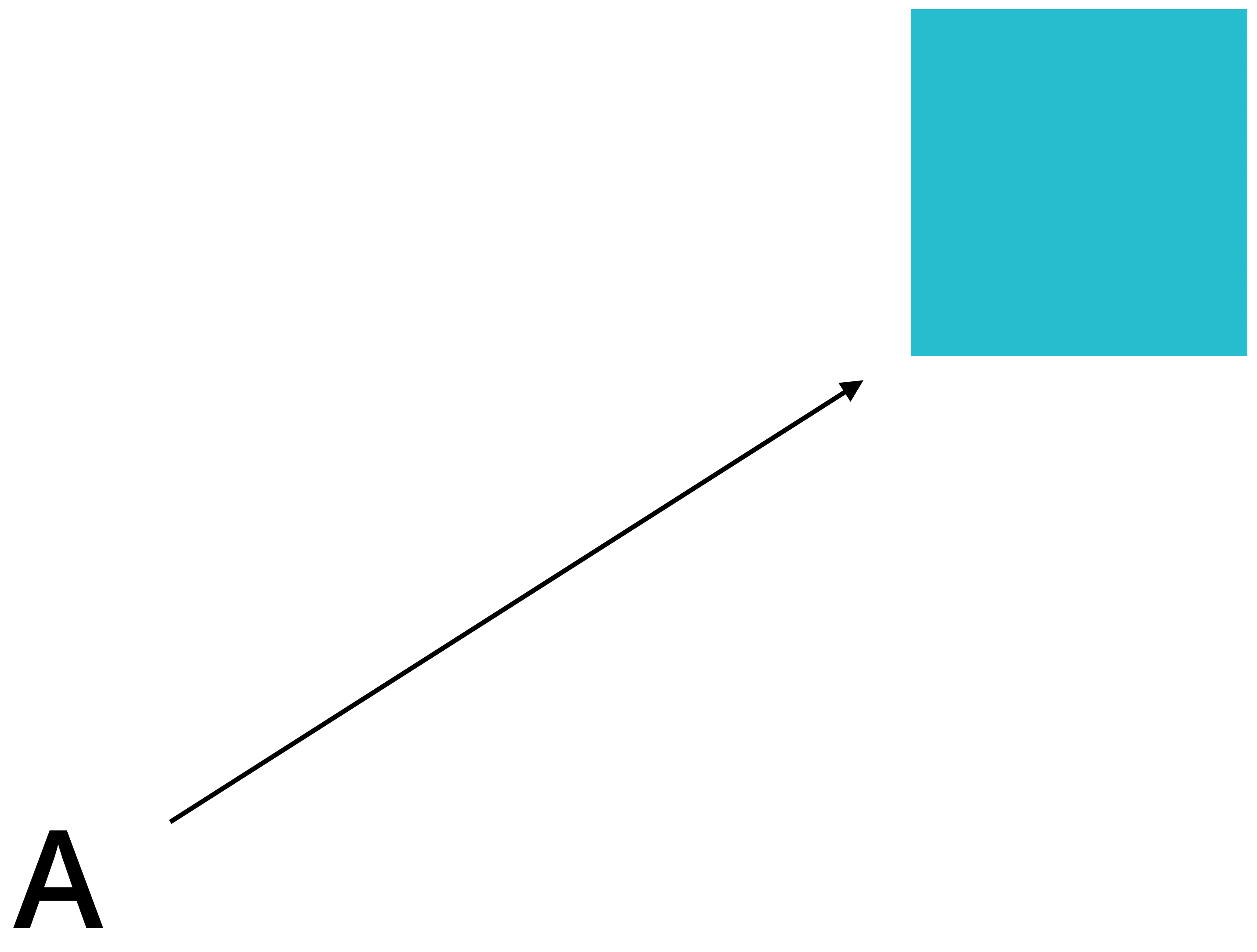
How **big** *are they?*

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

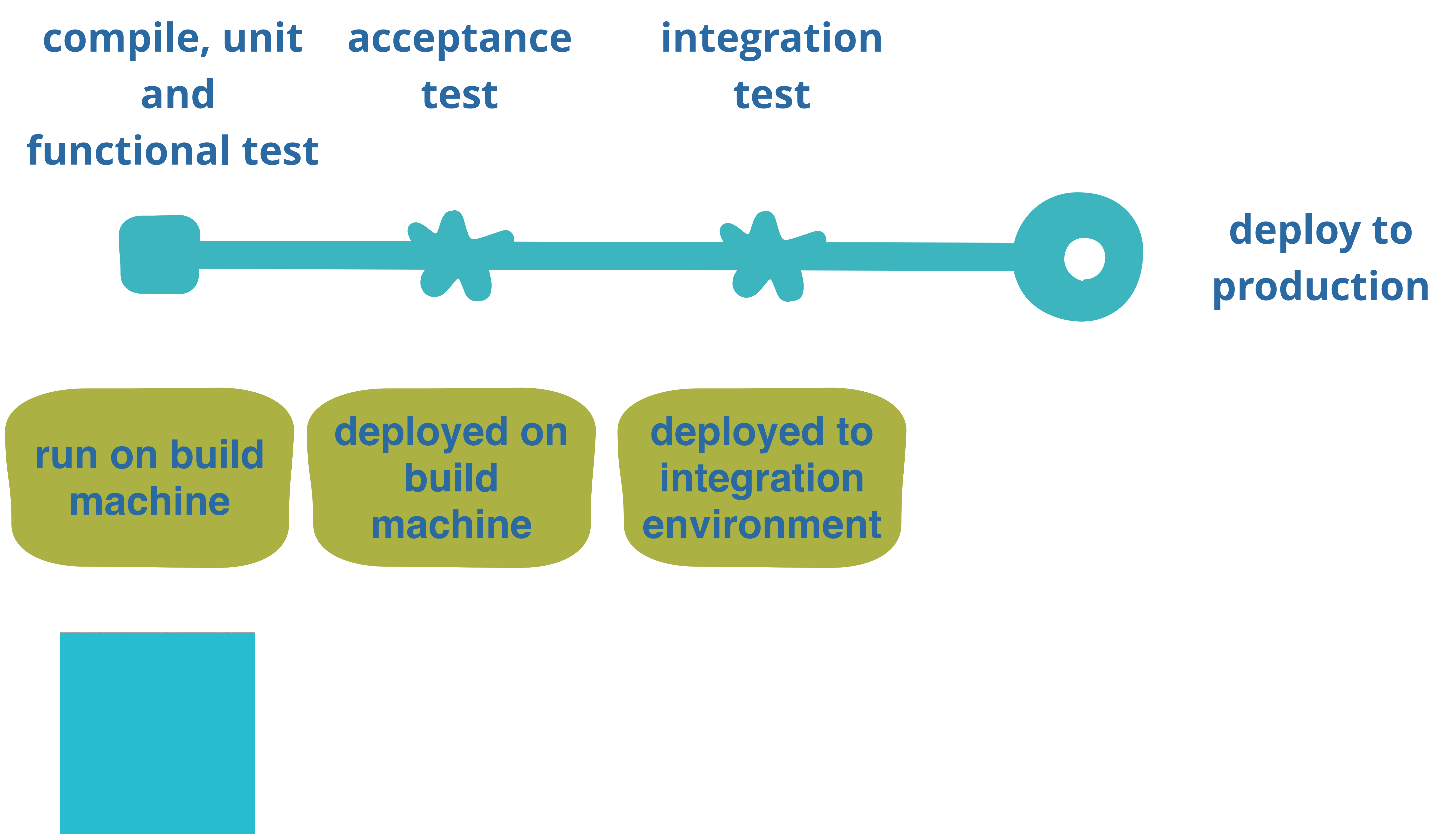
~~*How* **big** *are they?*~~

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

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

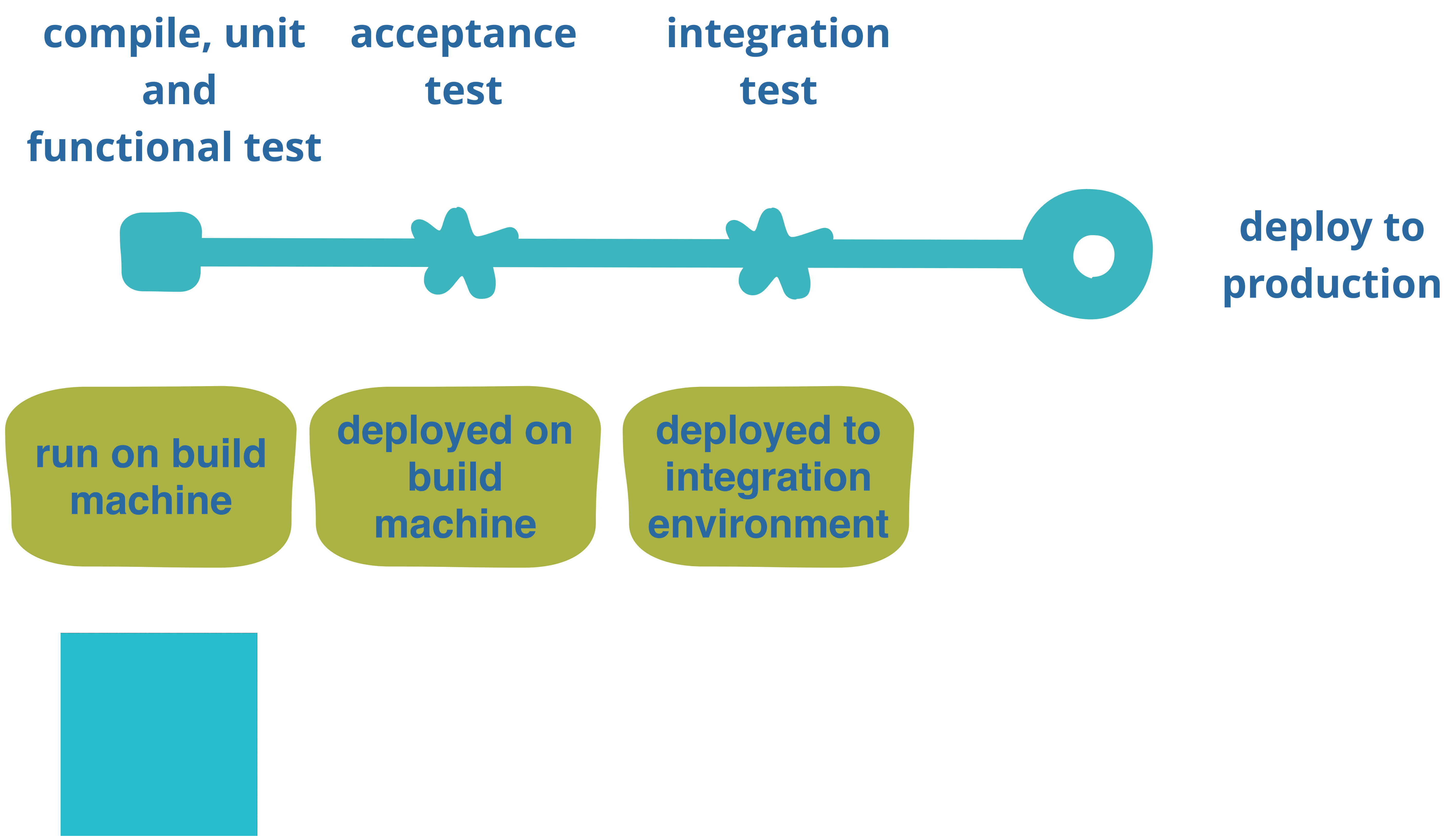


we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline



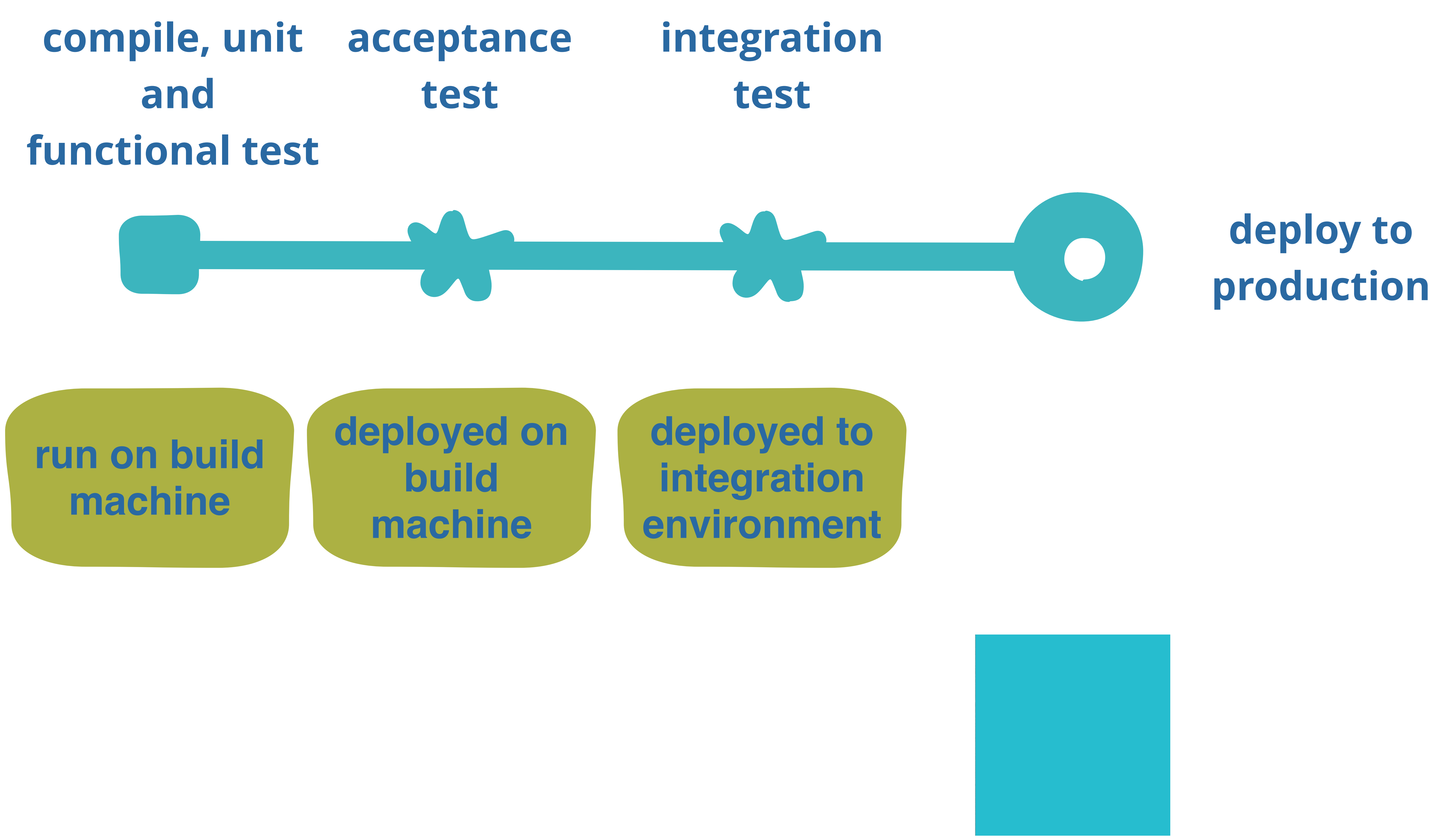
we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline

Tappety tap



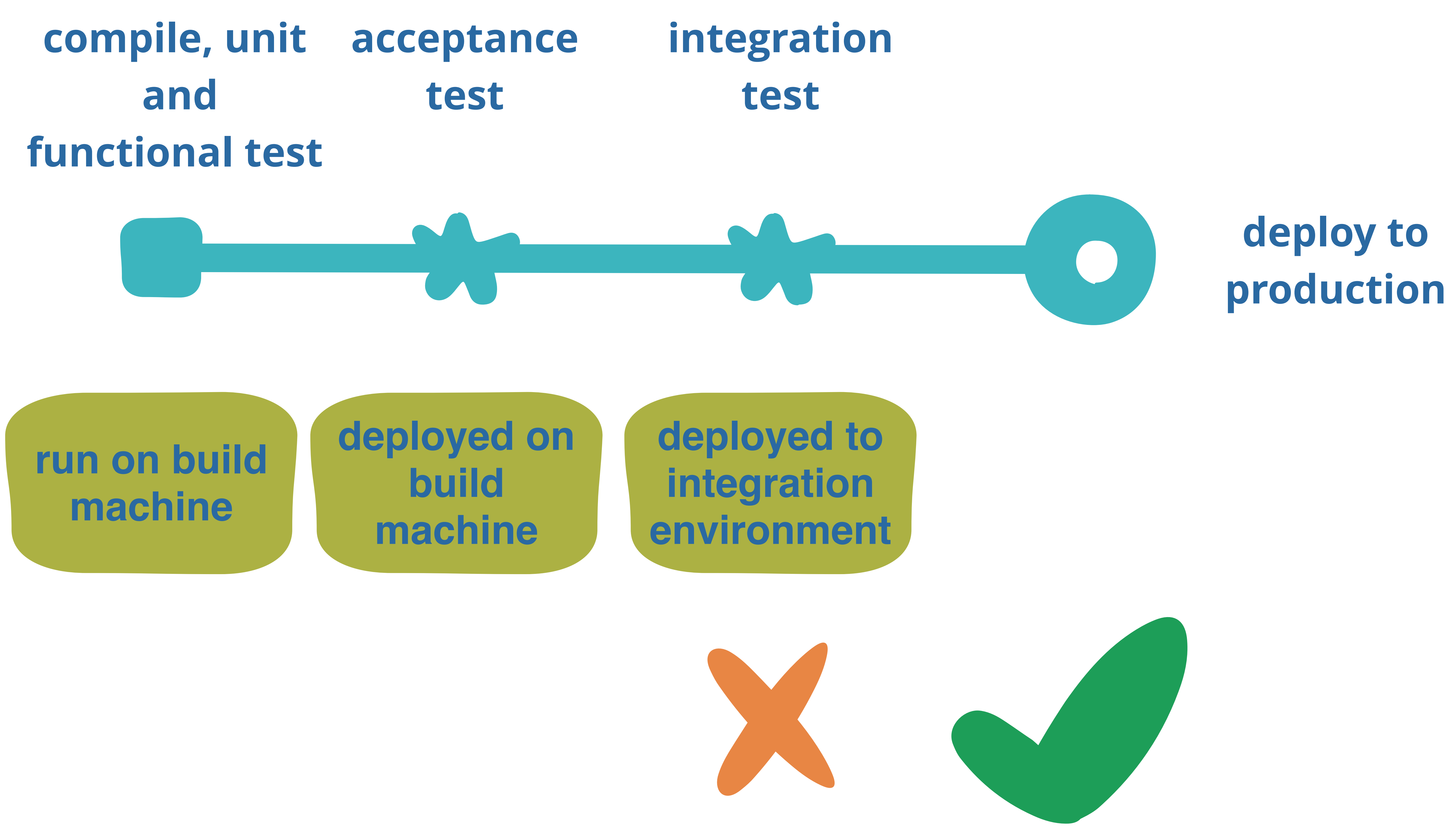
we want to get A into production and since we are hipsters we are going to practice continuous delivery - we will have a full automated build pipeline

Tappety tap



How many environments do we need?

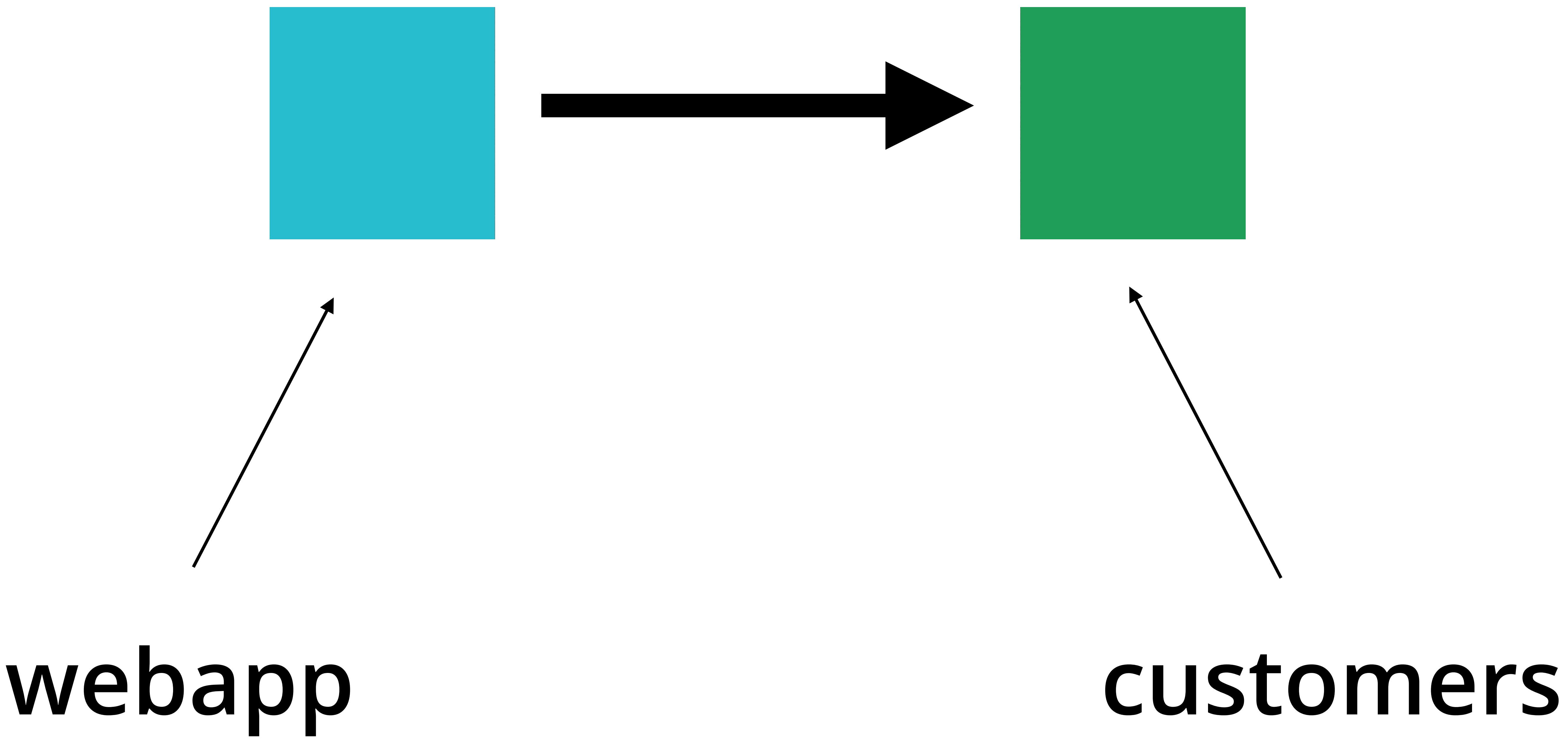
How many environments do we need?



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

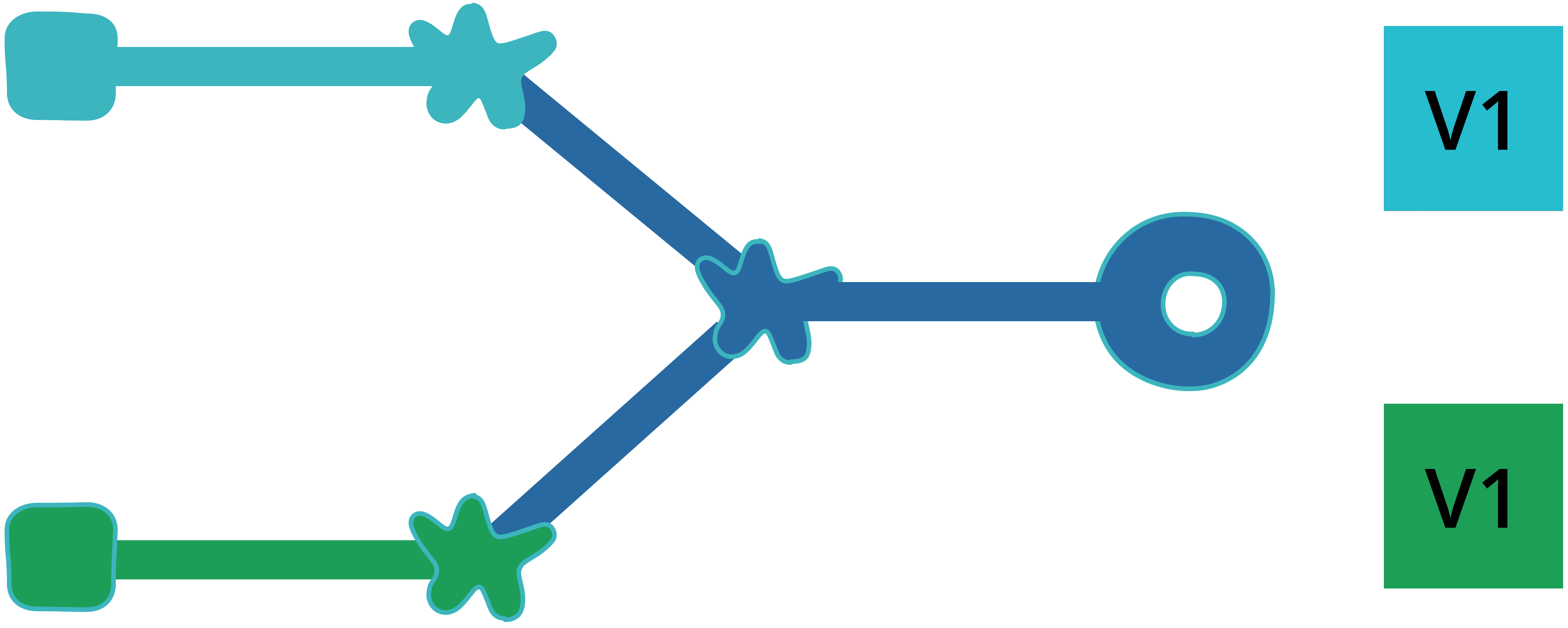
and we might as well call them something interesting

and they have a dependency on one another…

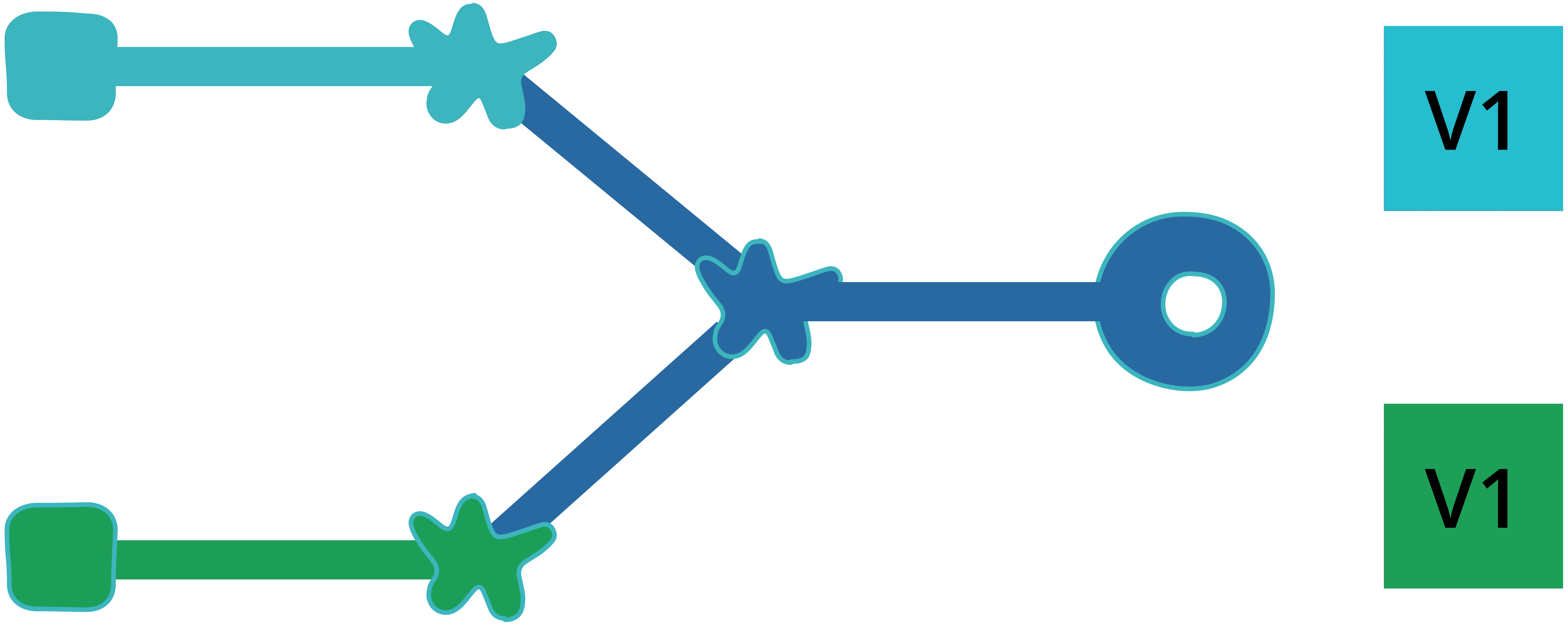


How do we traditionally make sure that new versions of the services work with each other?

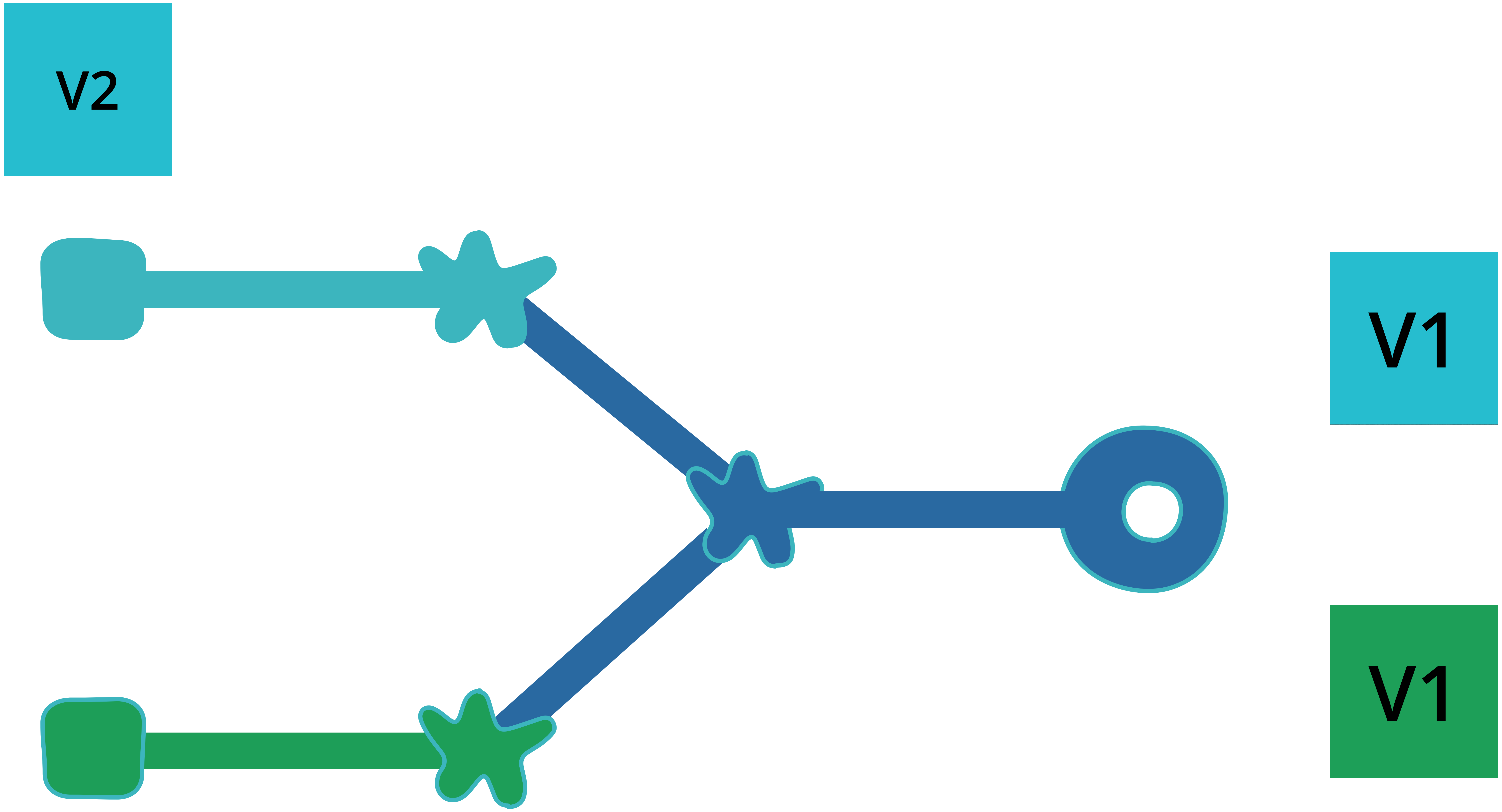
Let me illustrate this


V1
V1

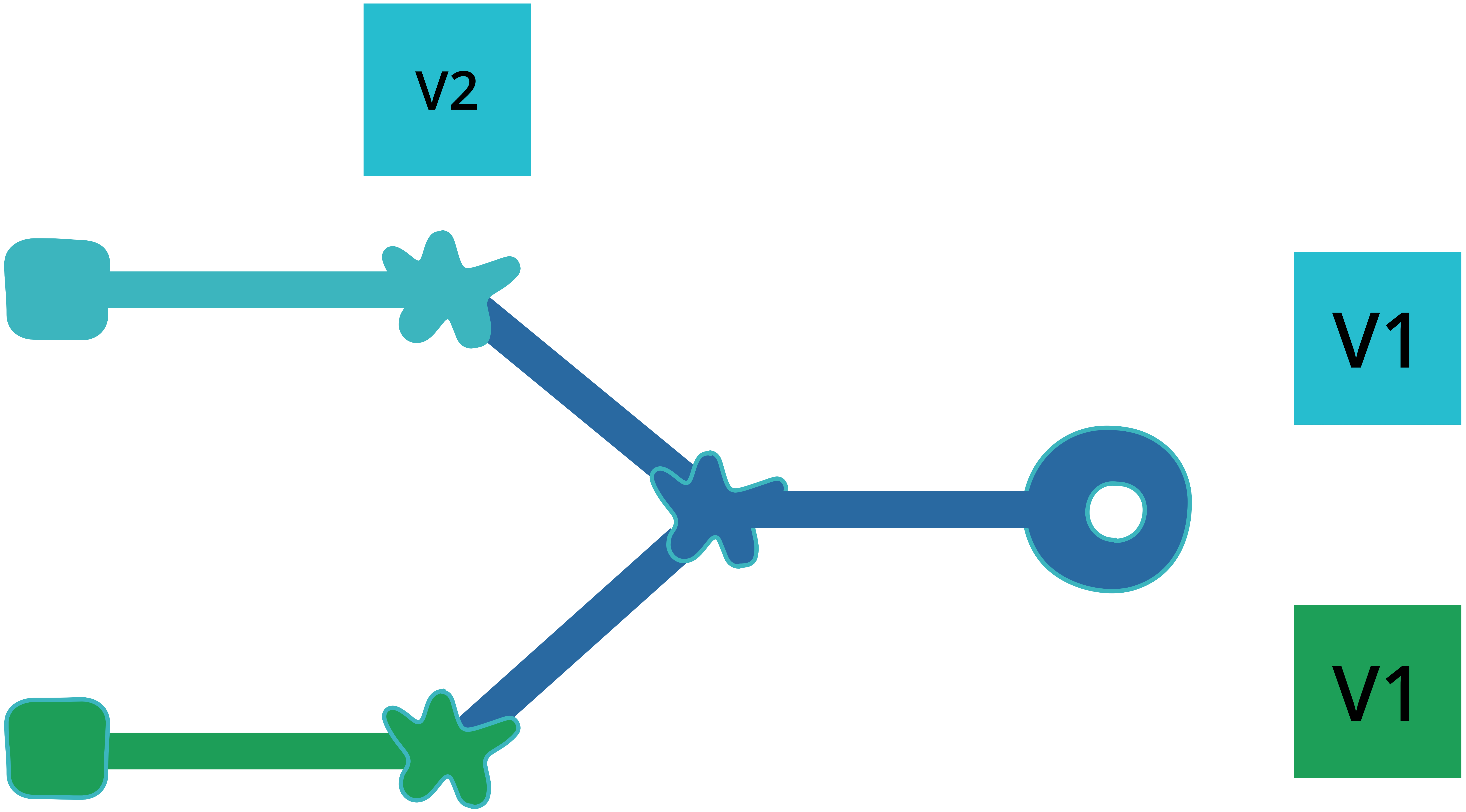
git push origin master



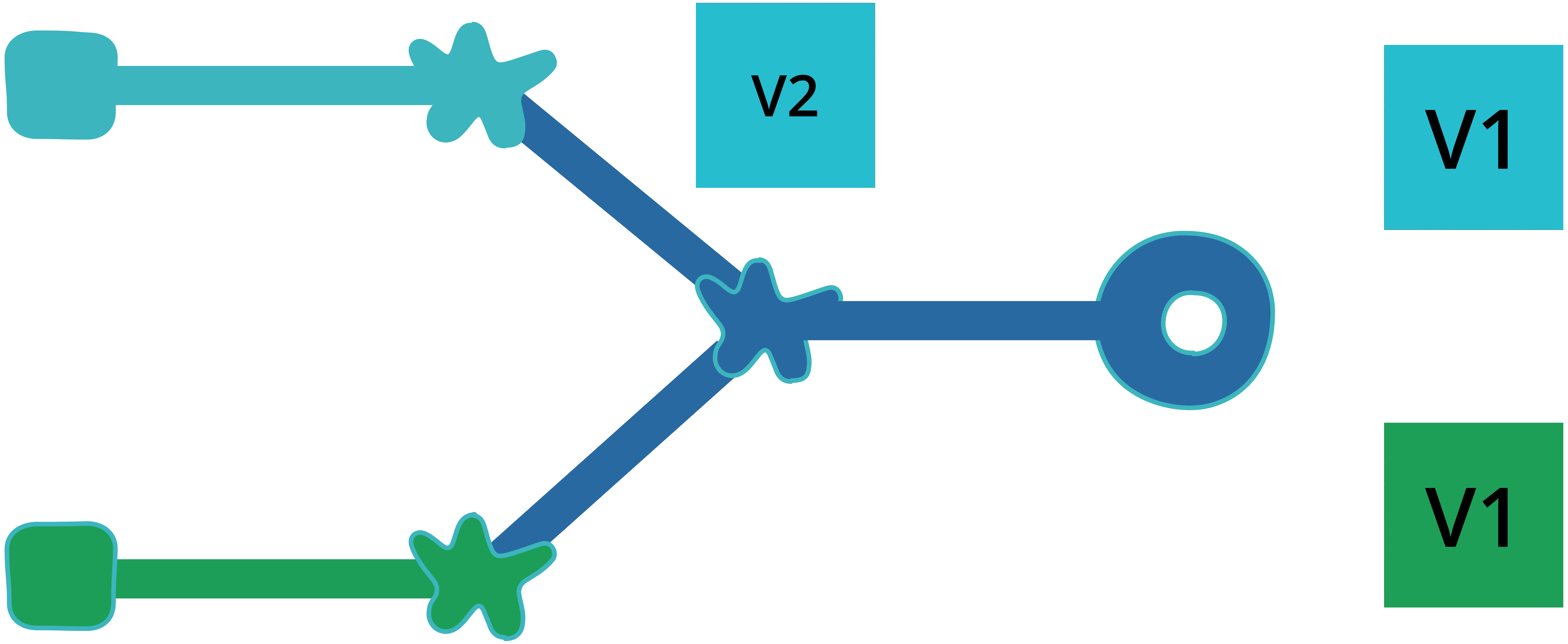
git push origin master



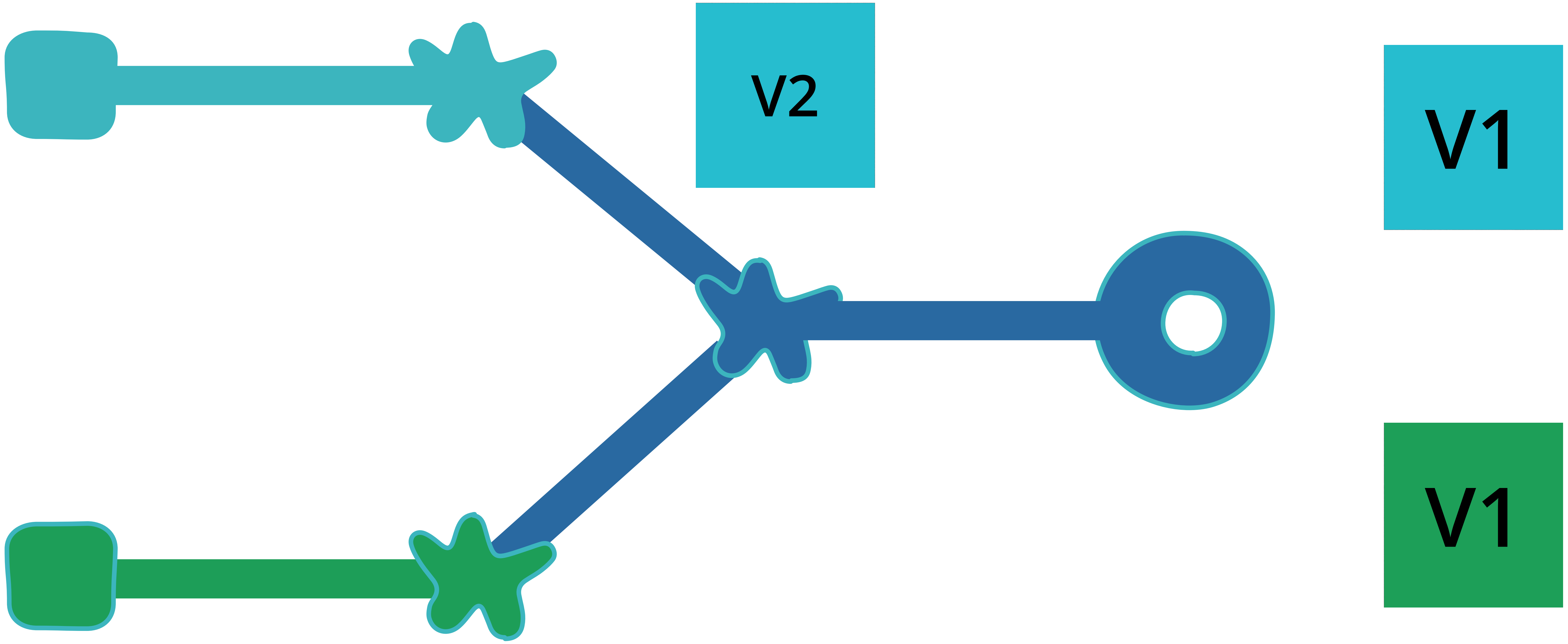
git push origin master



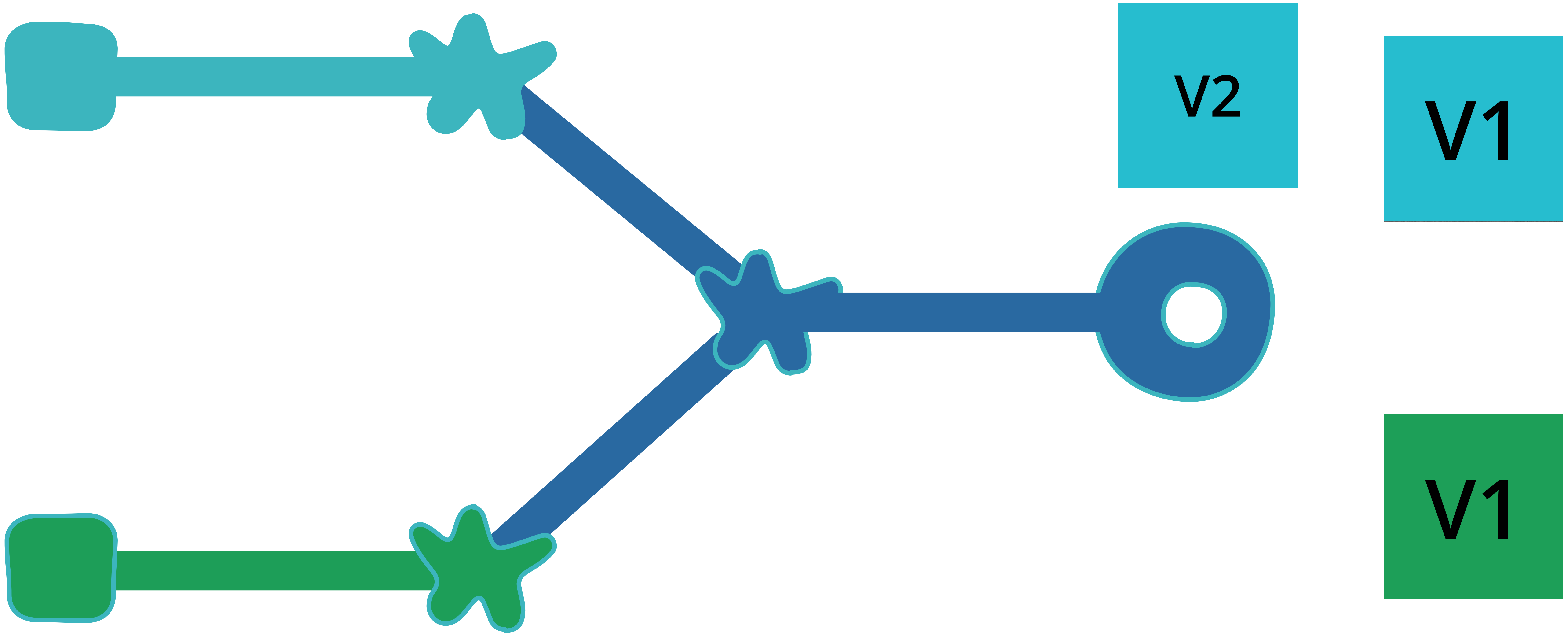
git push origin master

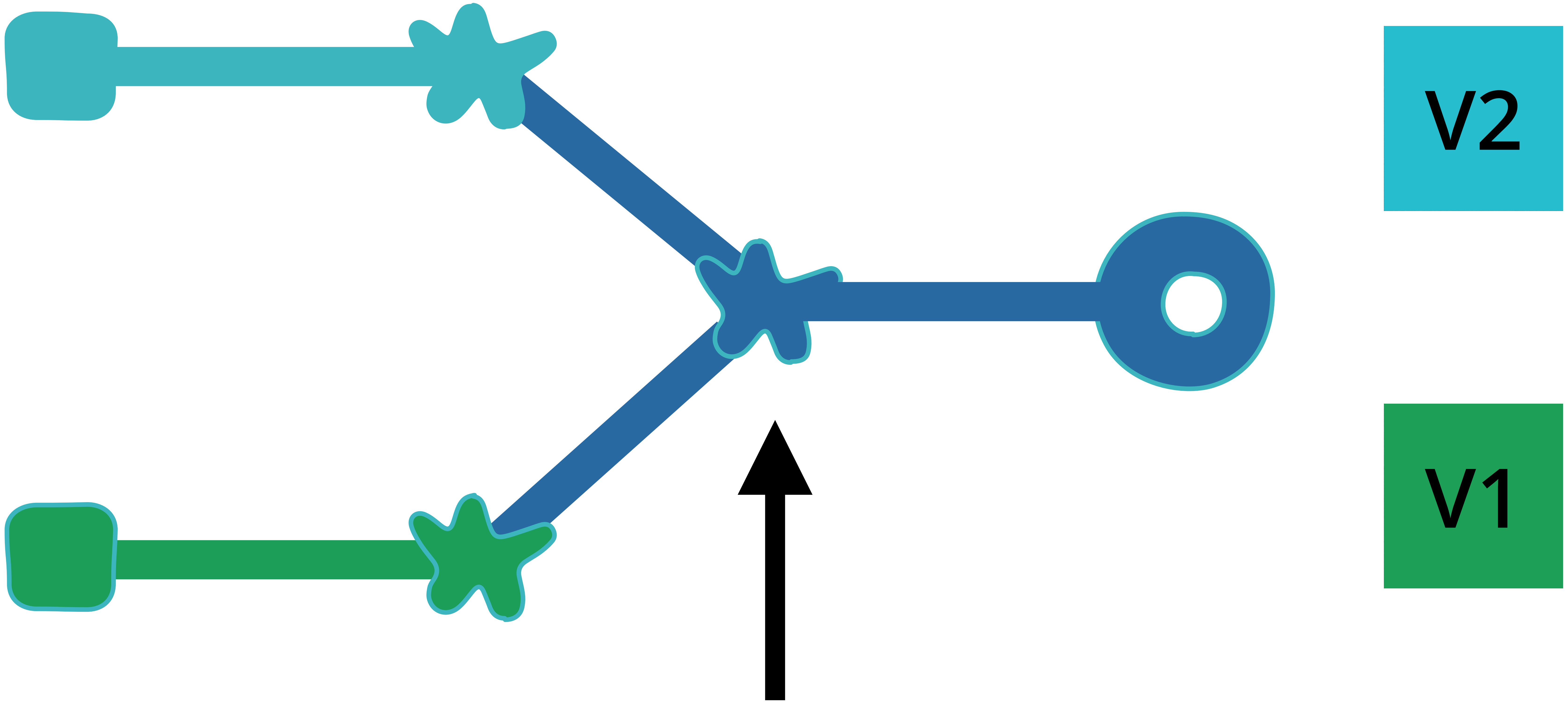


git push origin master



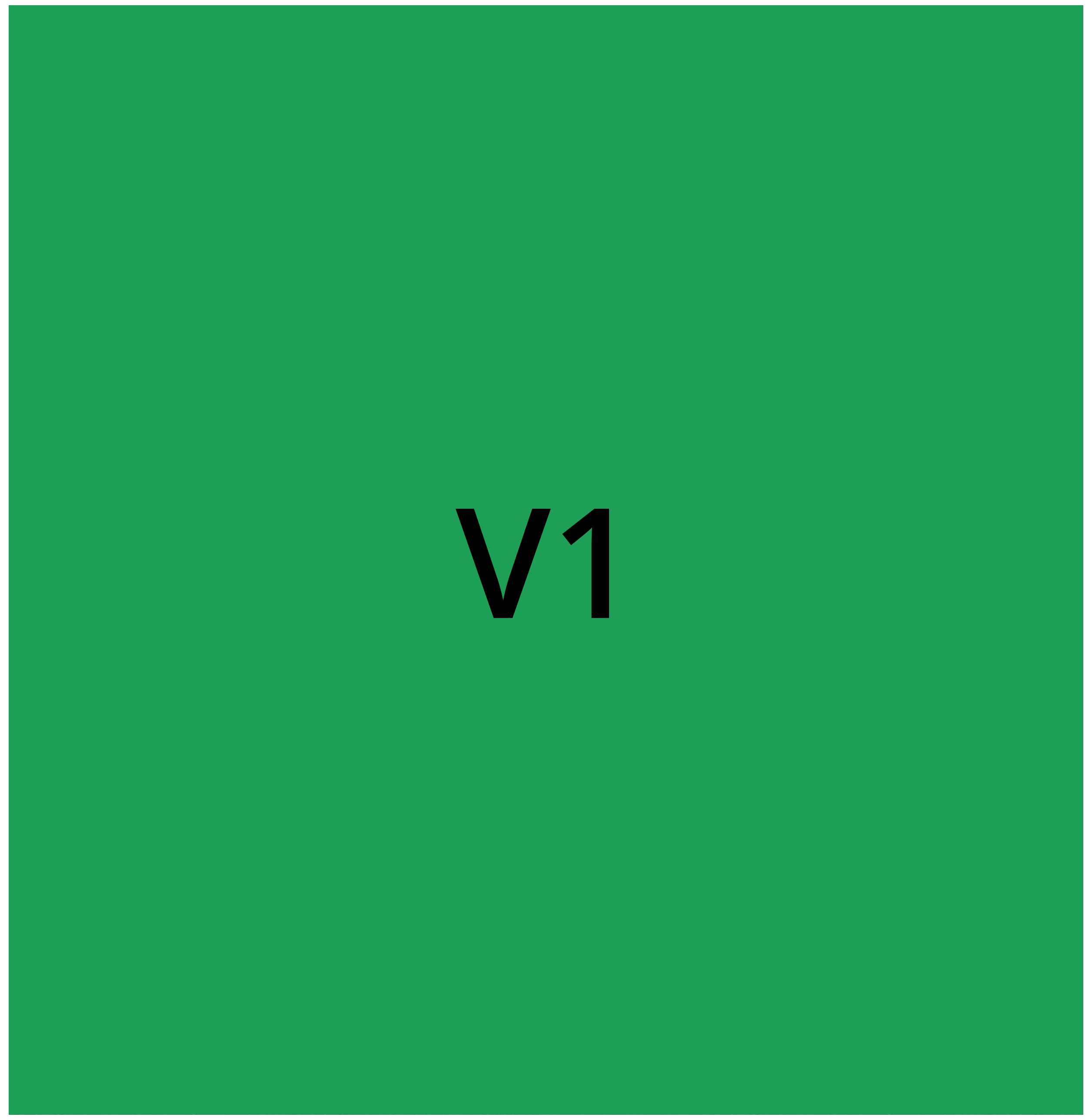
git push origin master

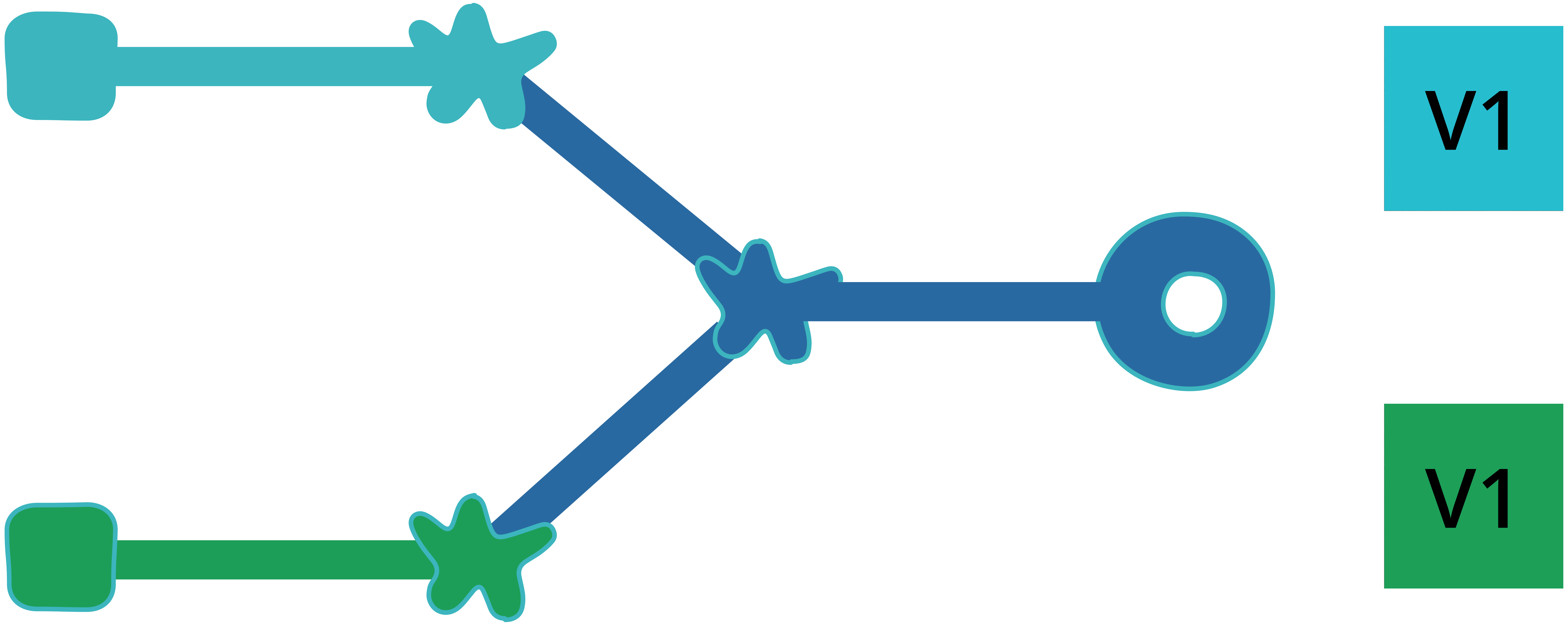




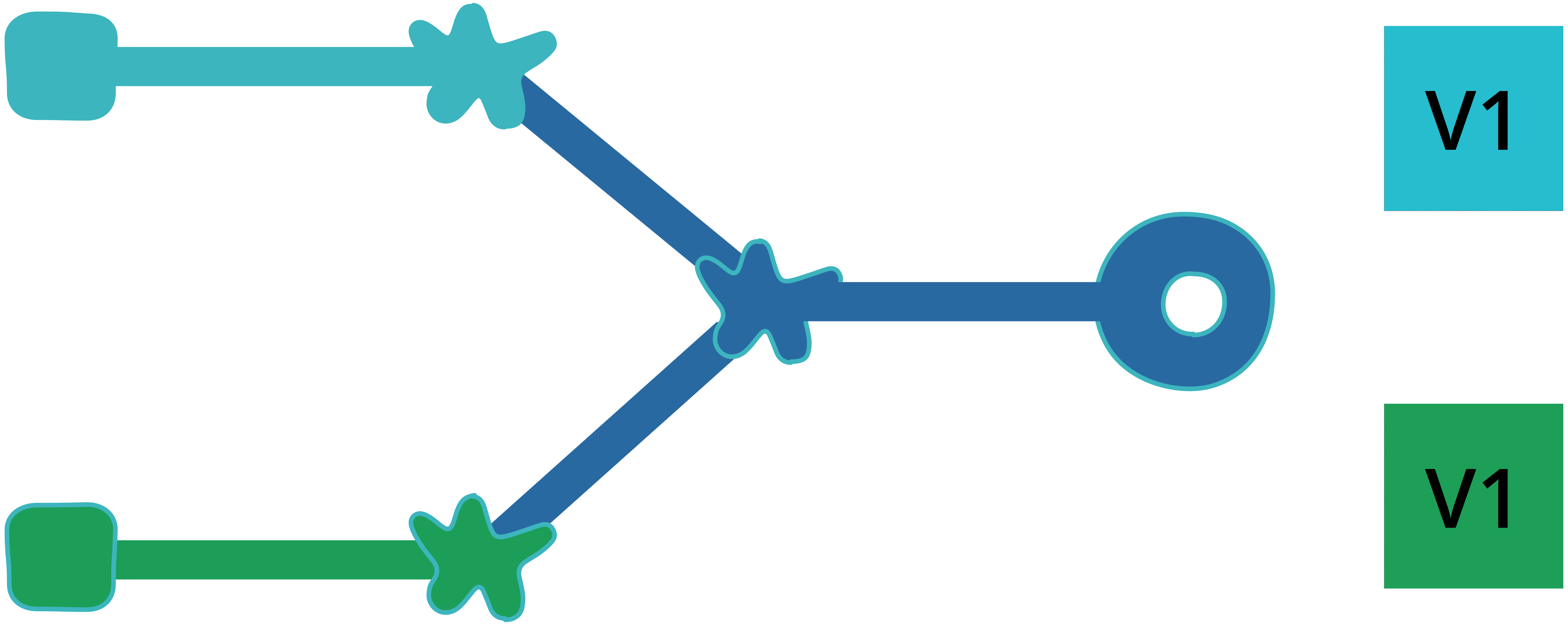
What should V2 of the blue app be tested against here

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



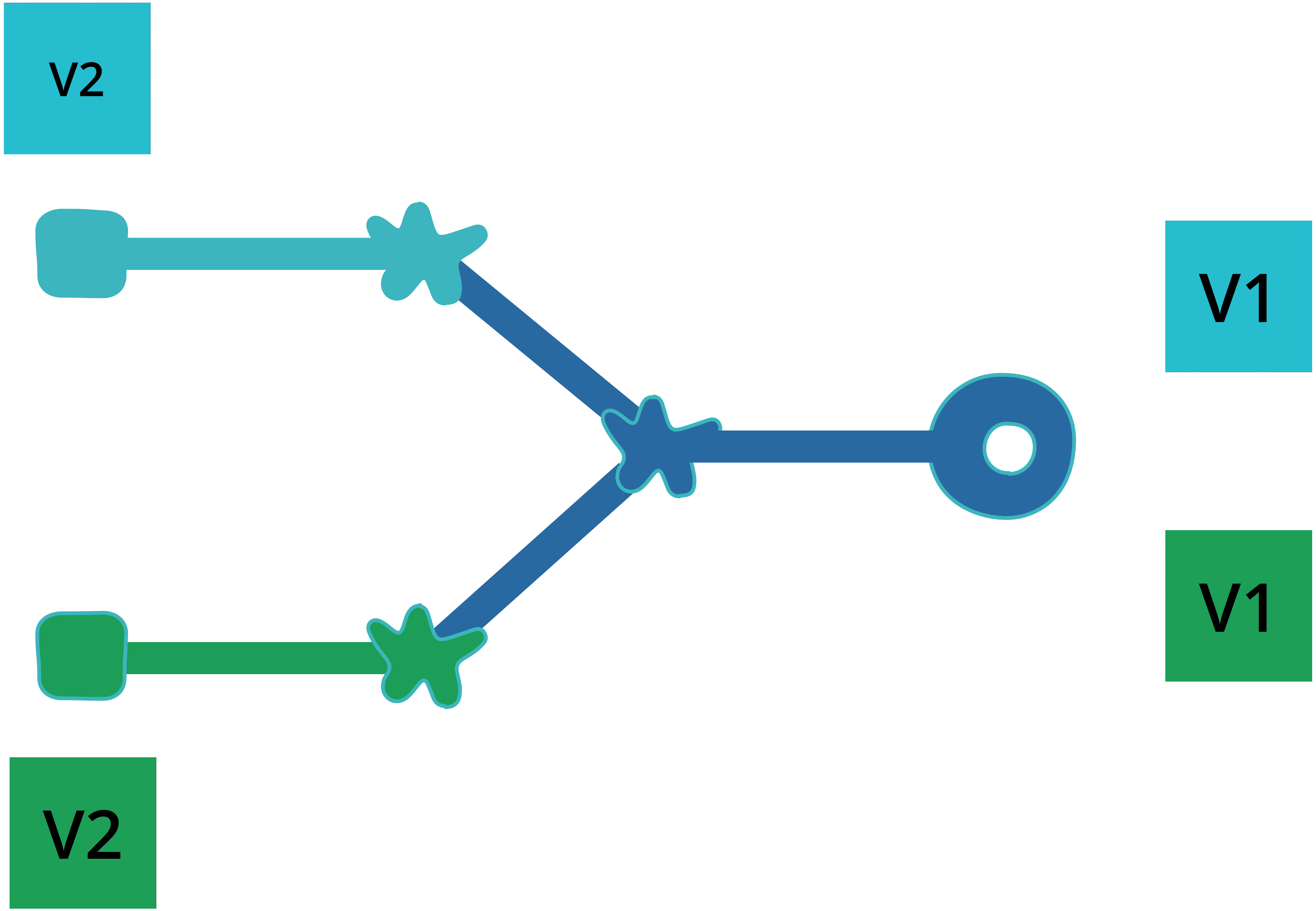

V1
V1

git push origin master



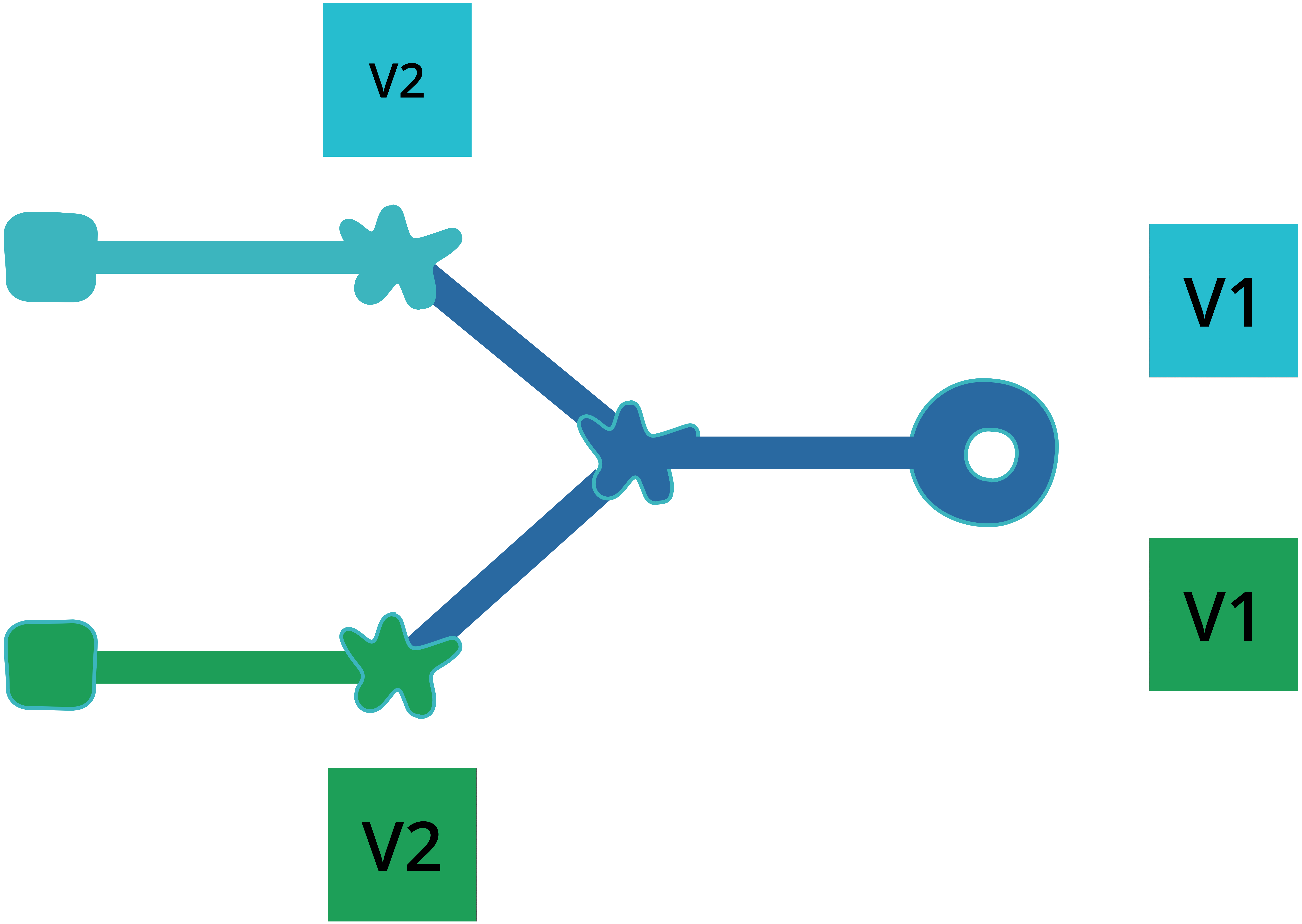
git push origin master

git push origin master



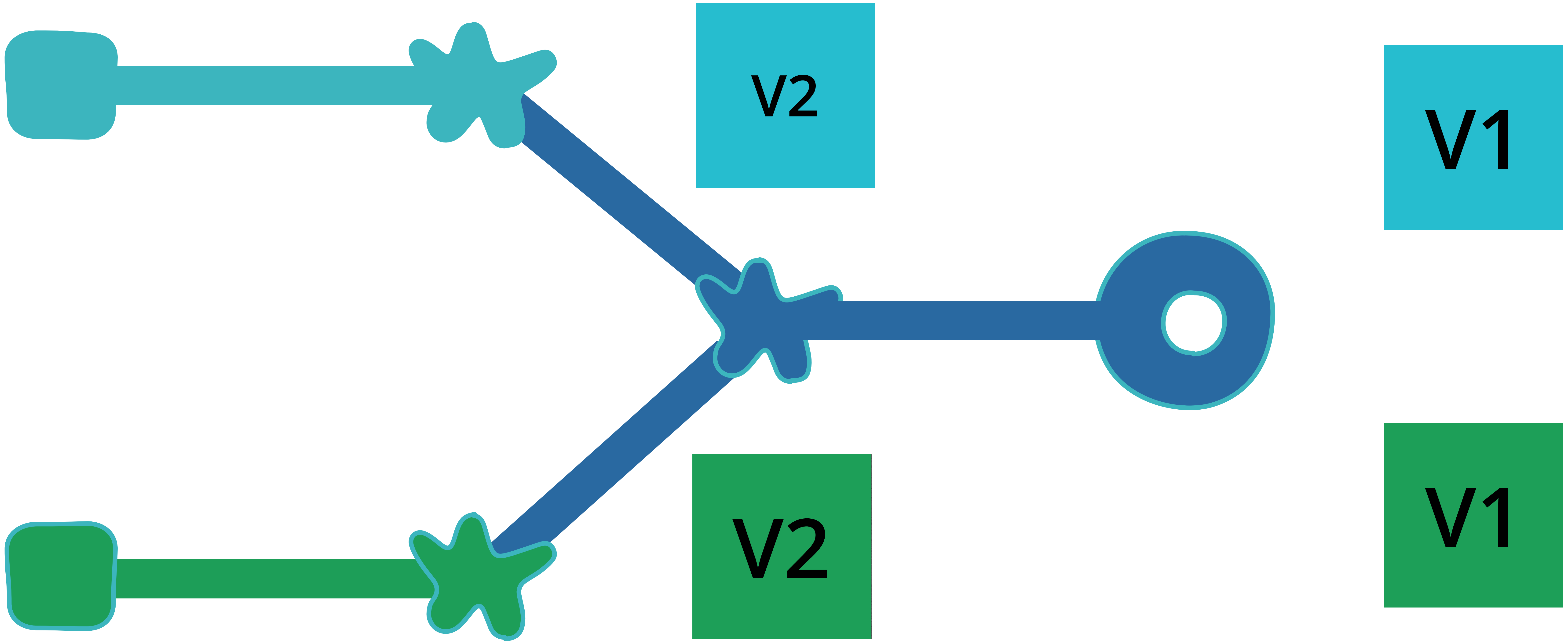
git push origin master

git push origin master



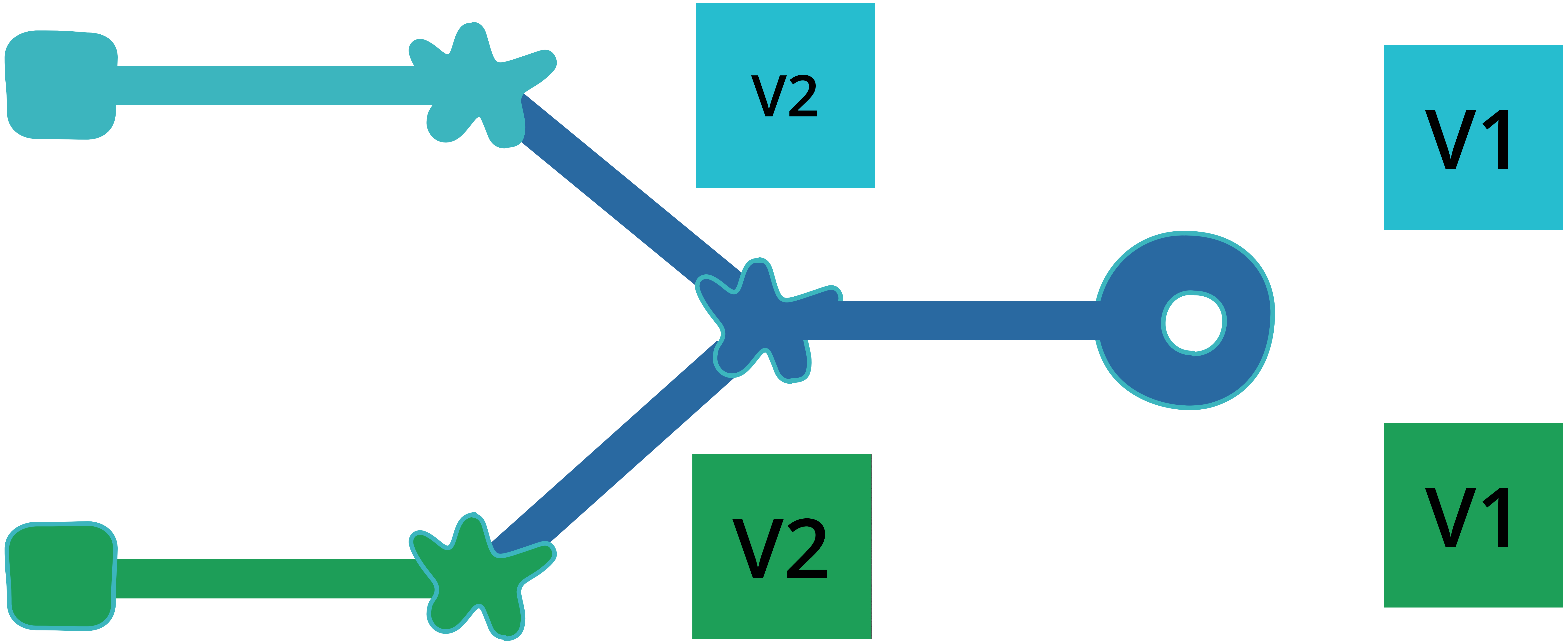
git push origin master

git push origin master



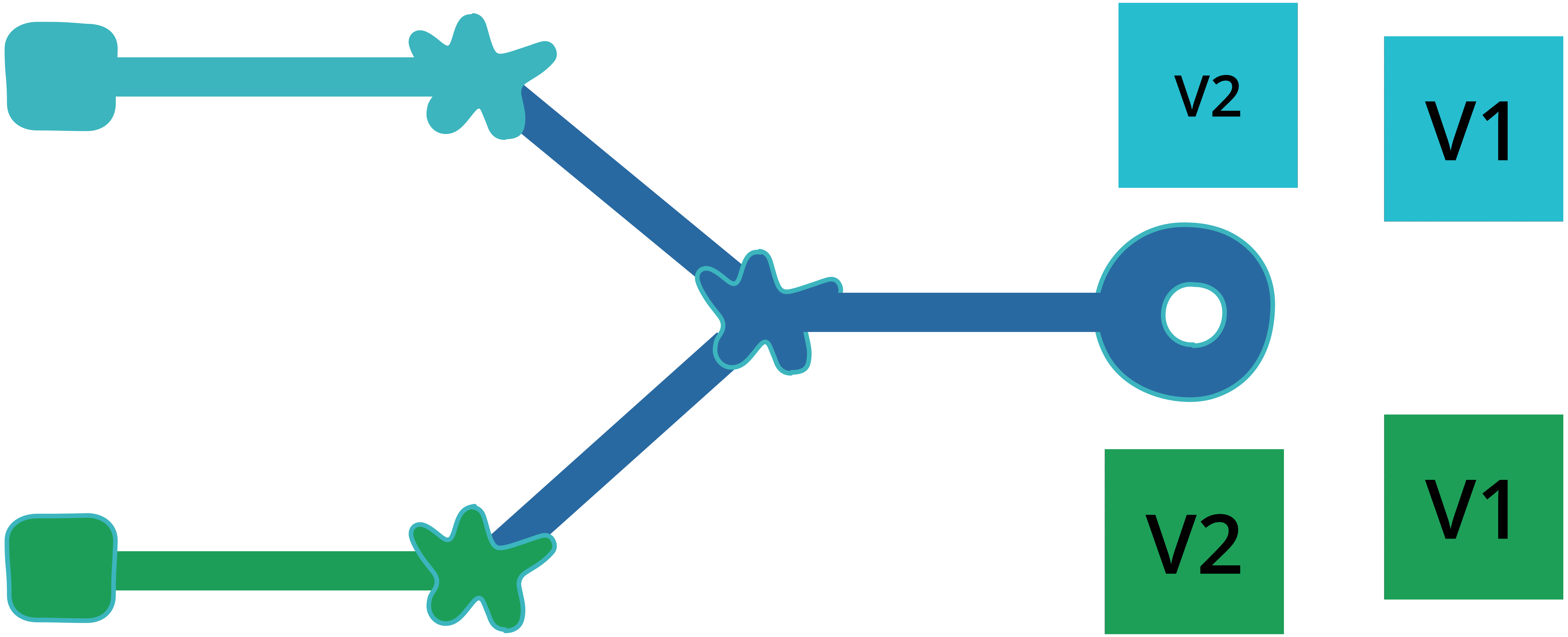
git push origin master

git push origin master



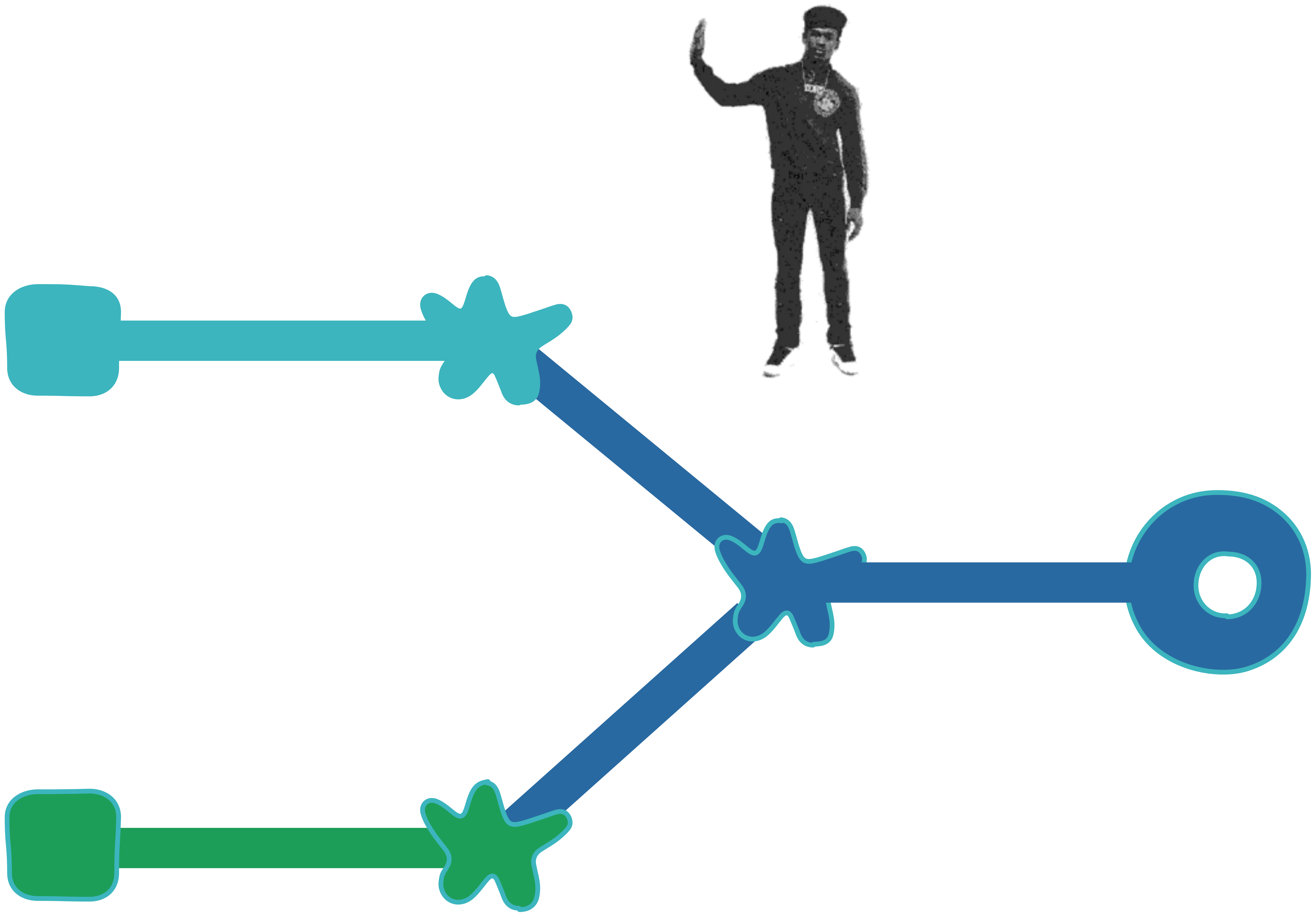
git push origin master

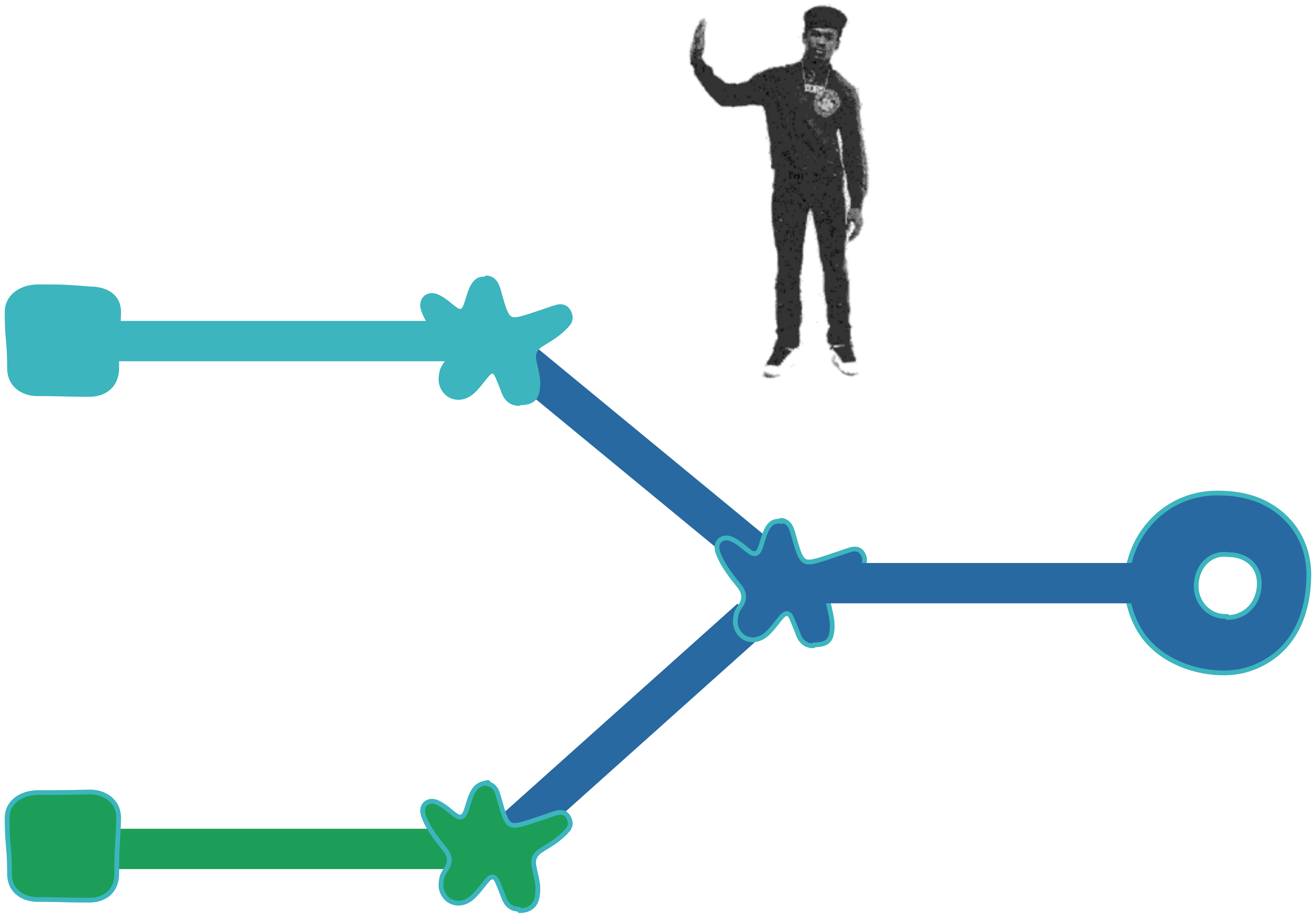
git push origin master



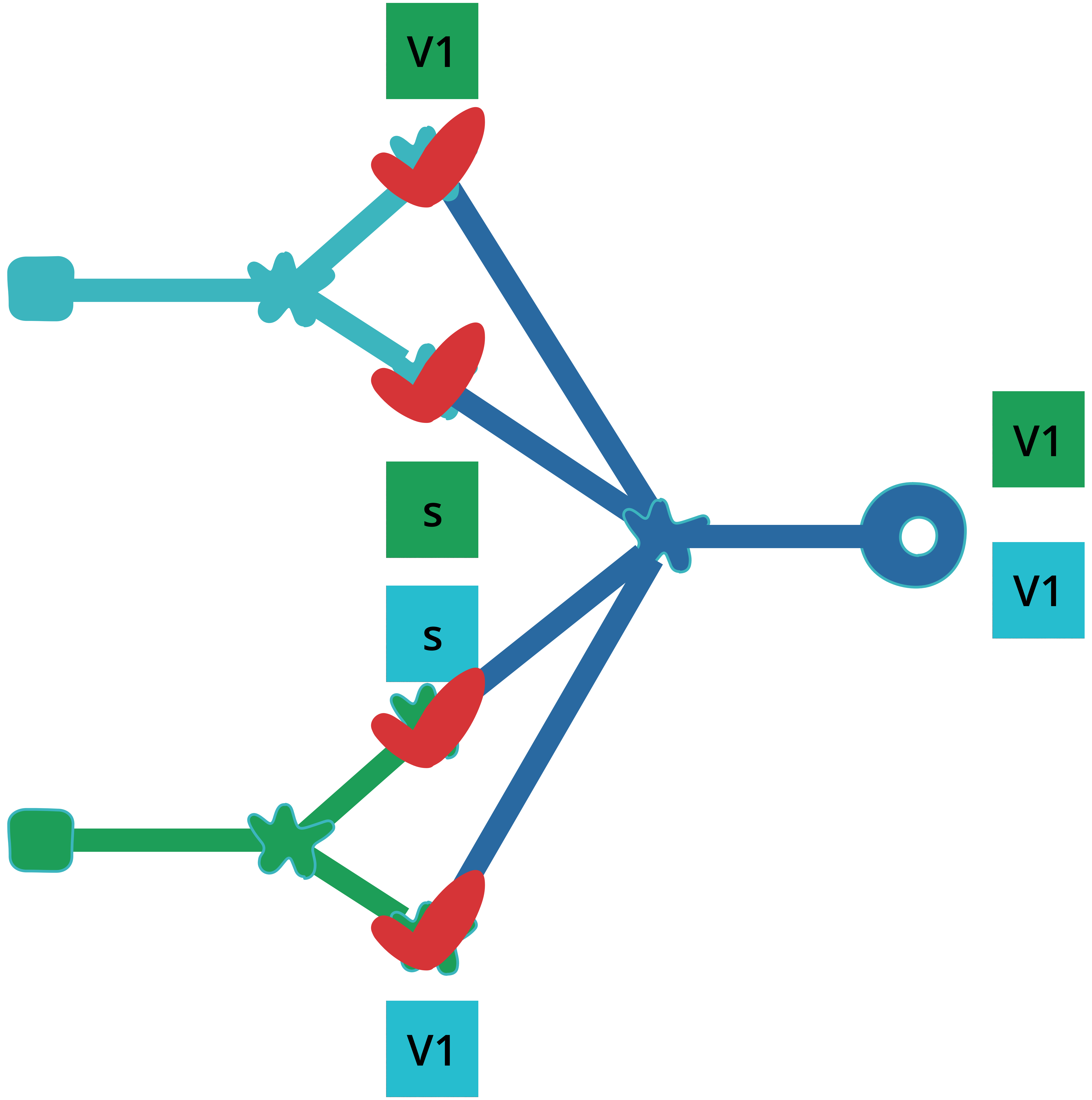
git push origin master


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





Locks == Delay


V1
s
s
V1
V1
V1

amazon
web services™

2 services

⇓

4 environments

2 services

2 services

4

2 services

4

>600

the death of the integration environment

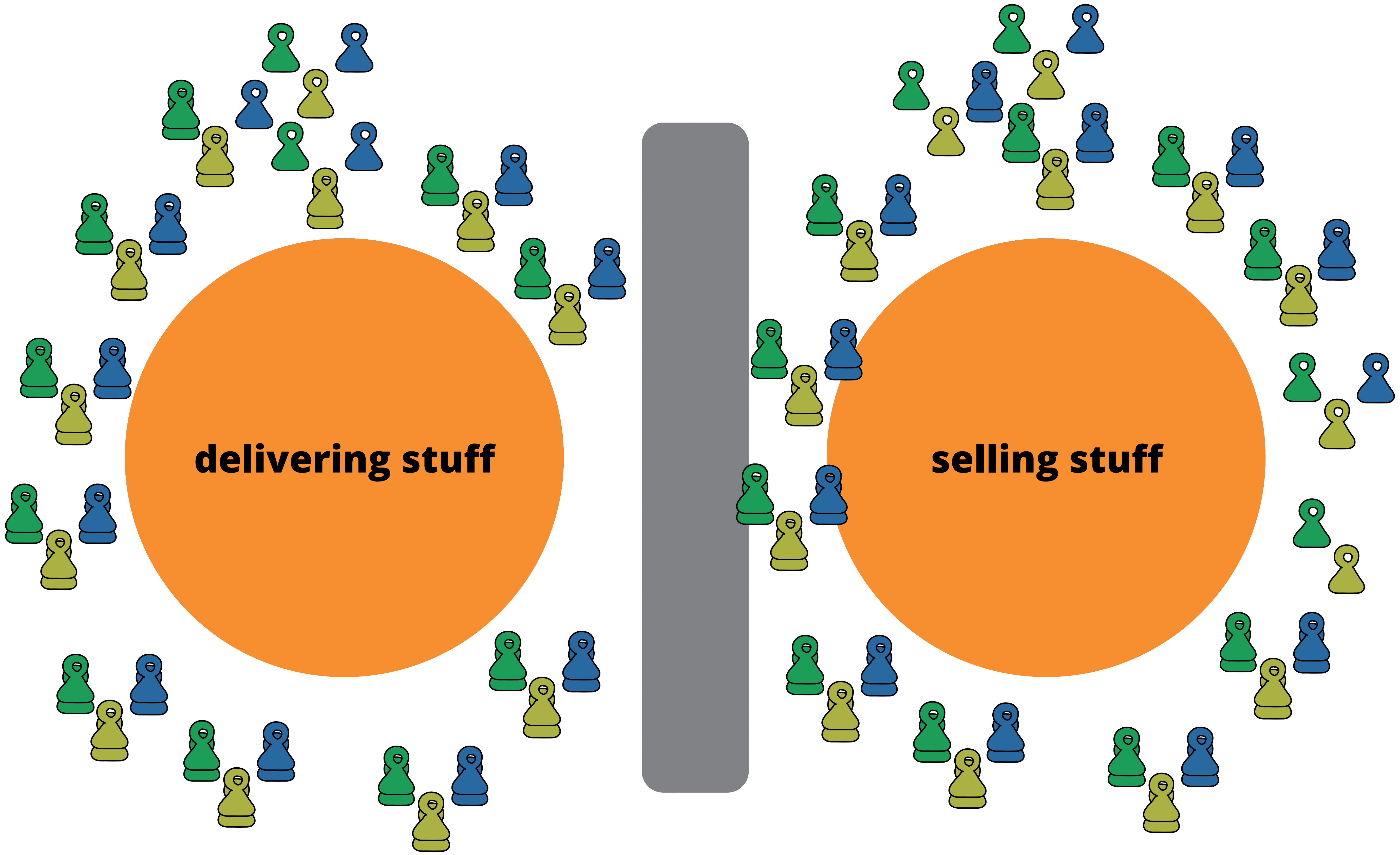
1972 - Dennis Ritchie invents a powerful gun that shoots both forward and backward simultaneously. Not satisfied with the number of deaths and permanent maimings from that invention he invents C and Unix.

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

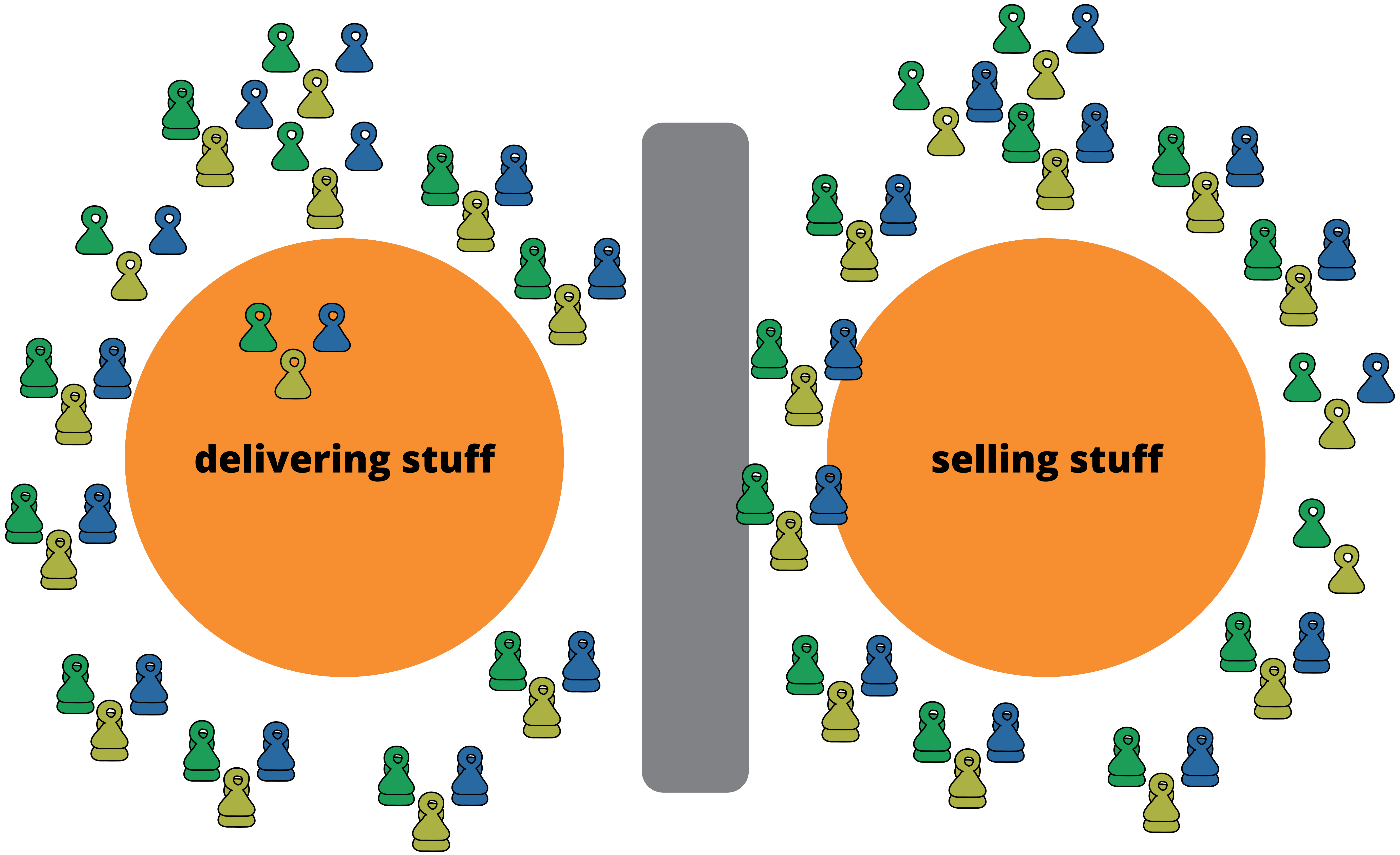
A **capability** is a combination of people, processes, systems that provides value to customers (internal or external)

The **what** of the business, not the **how**

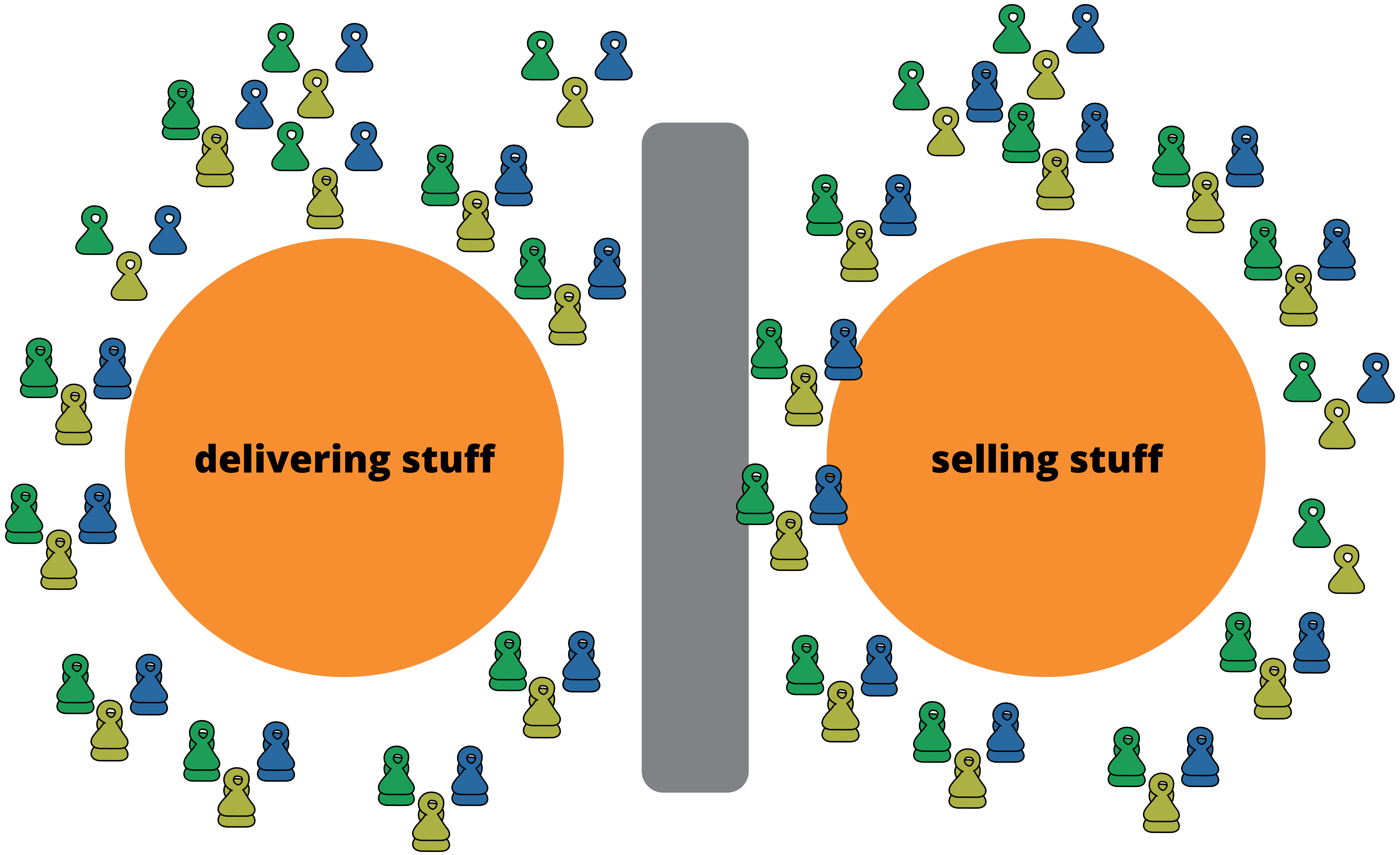
The problem with projects



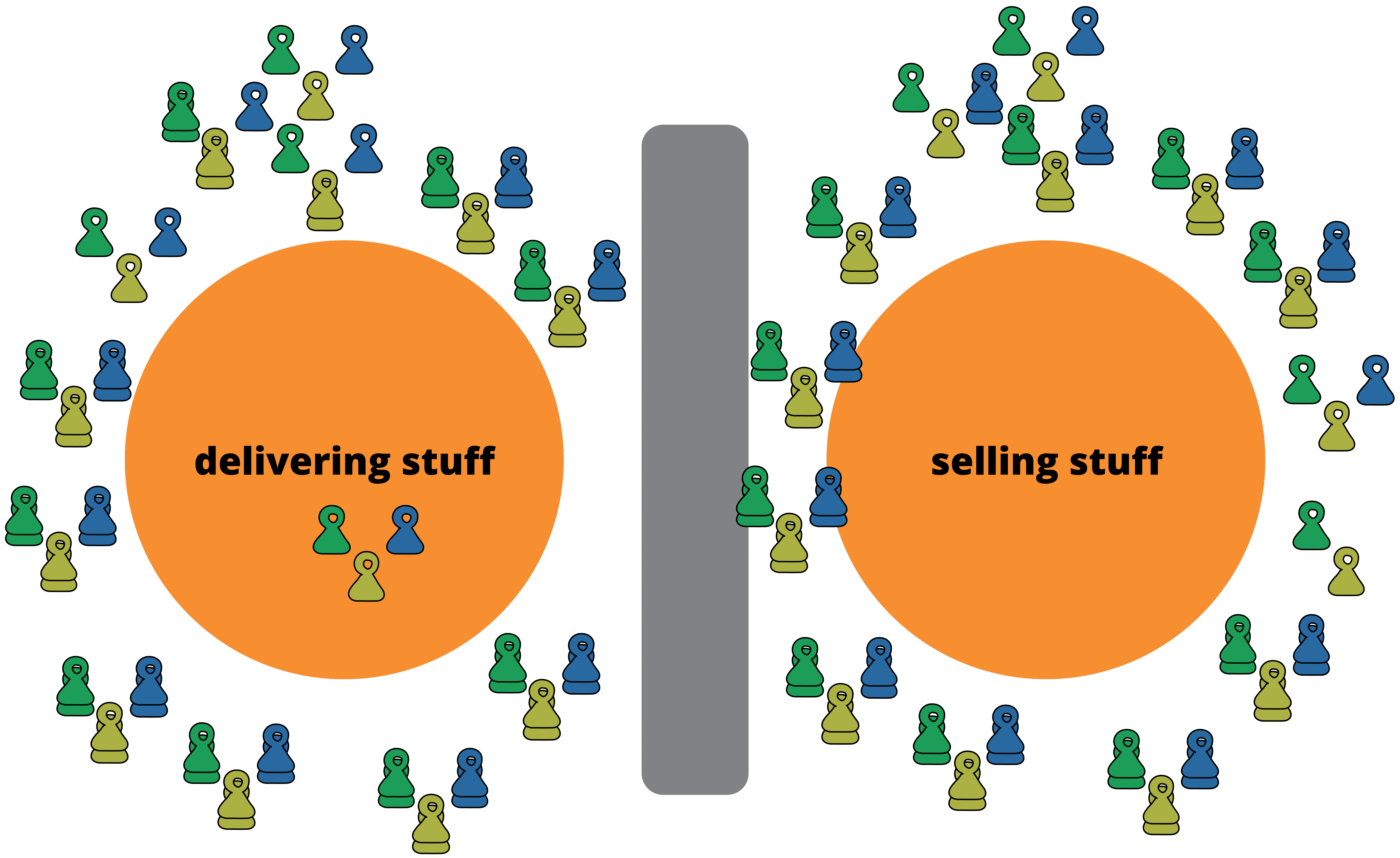
The problem with projects



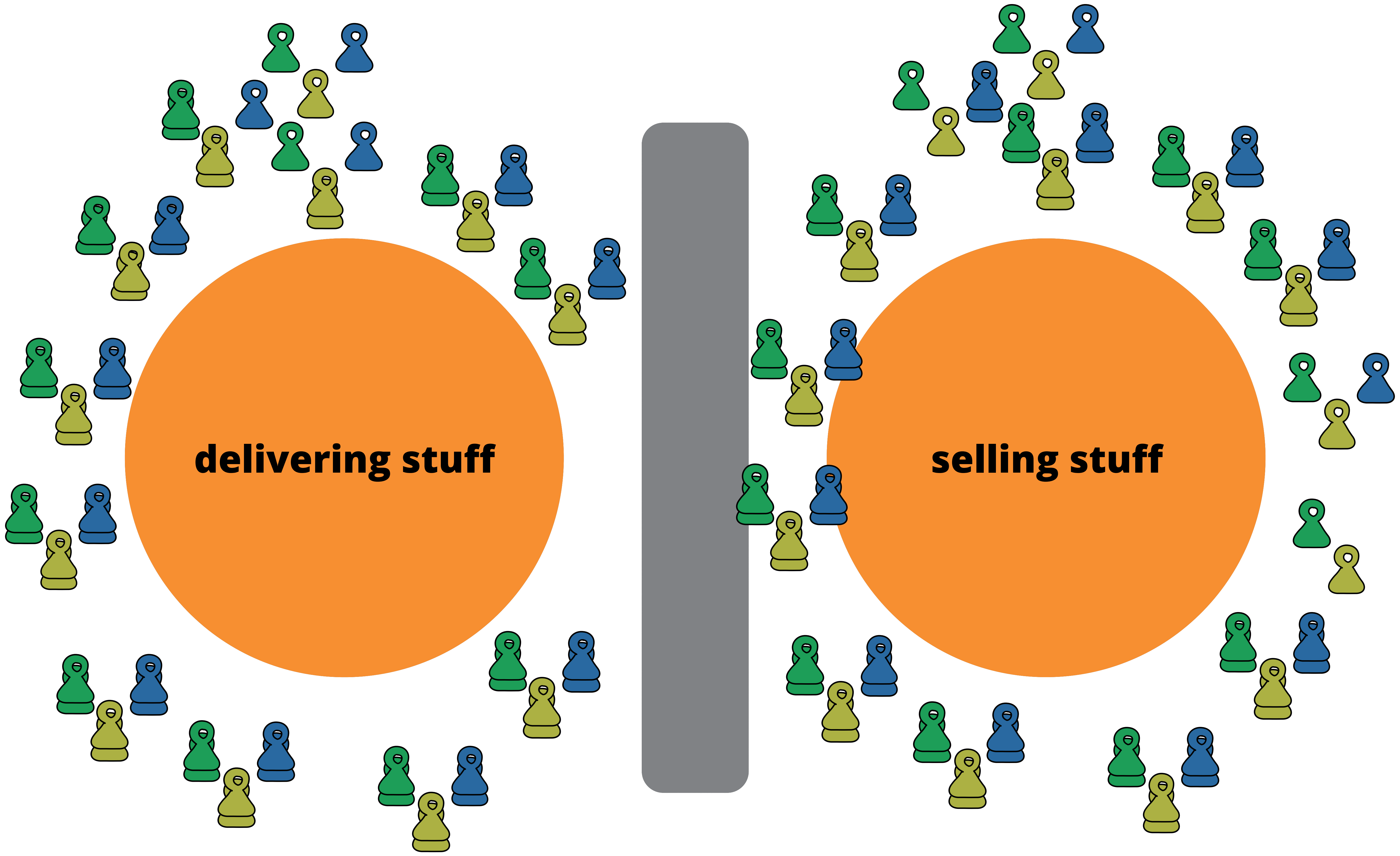
The problem with projects



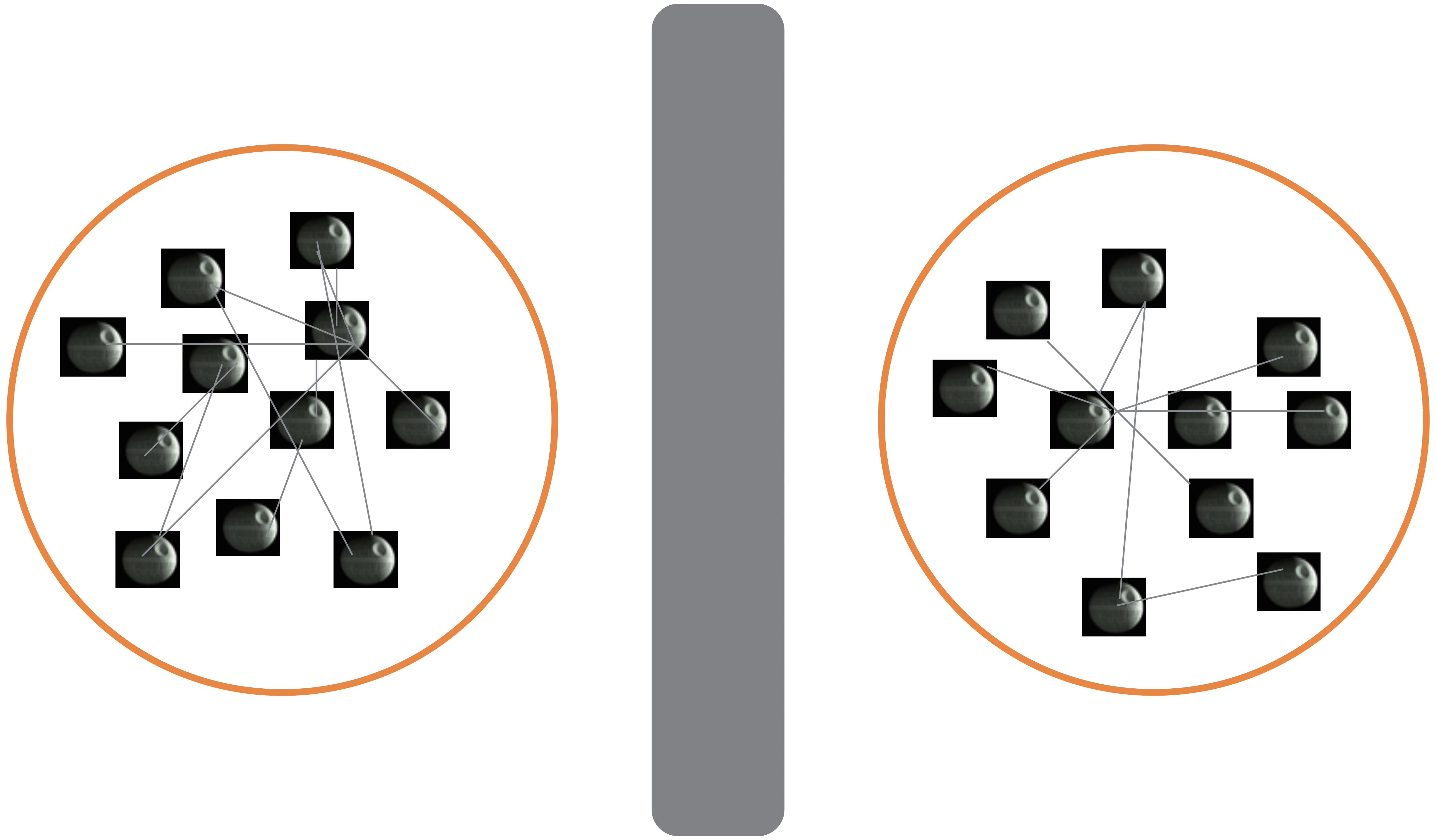
The problem with projects



The problem with projects

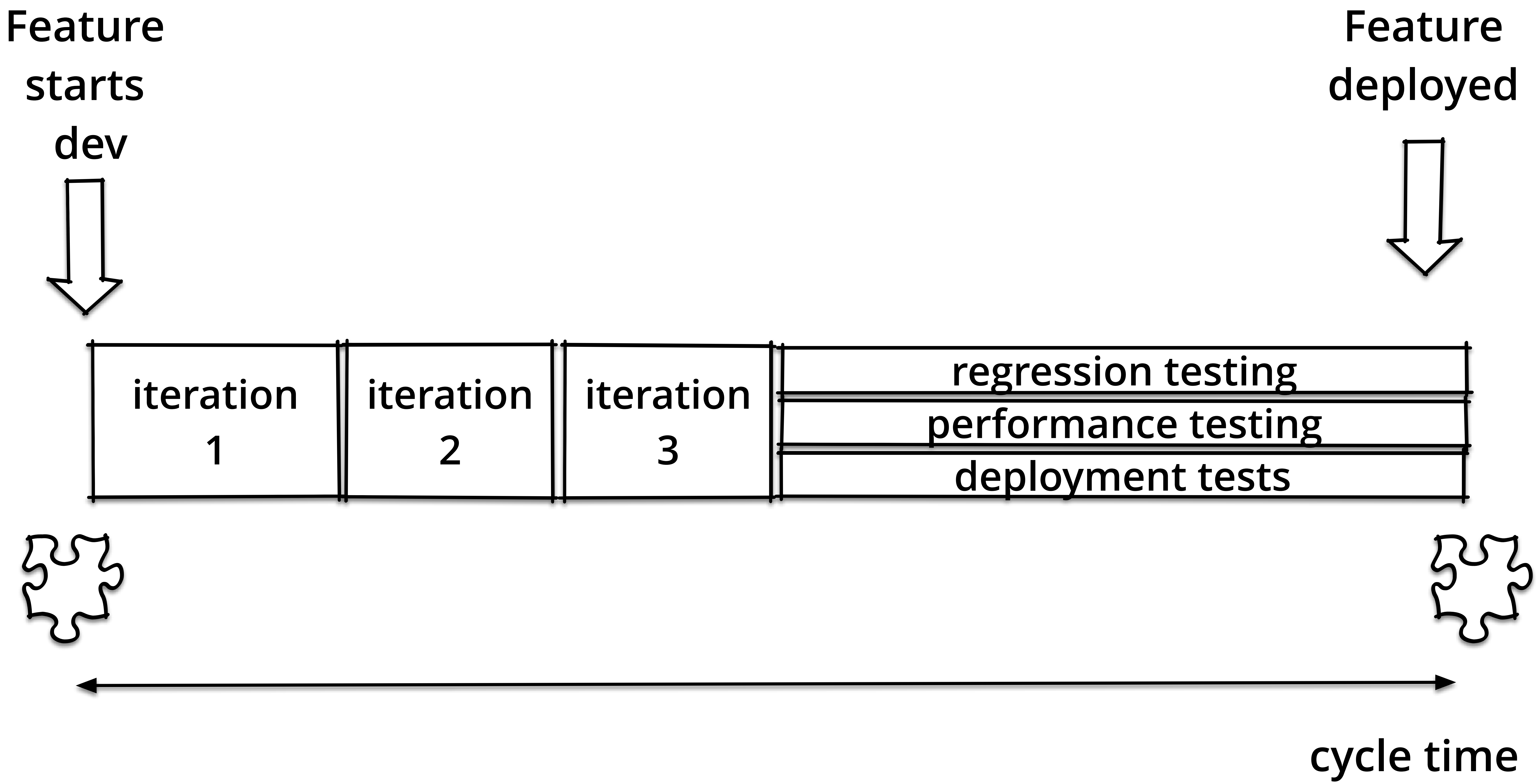


If you aren’t *really* careful with your API design

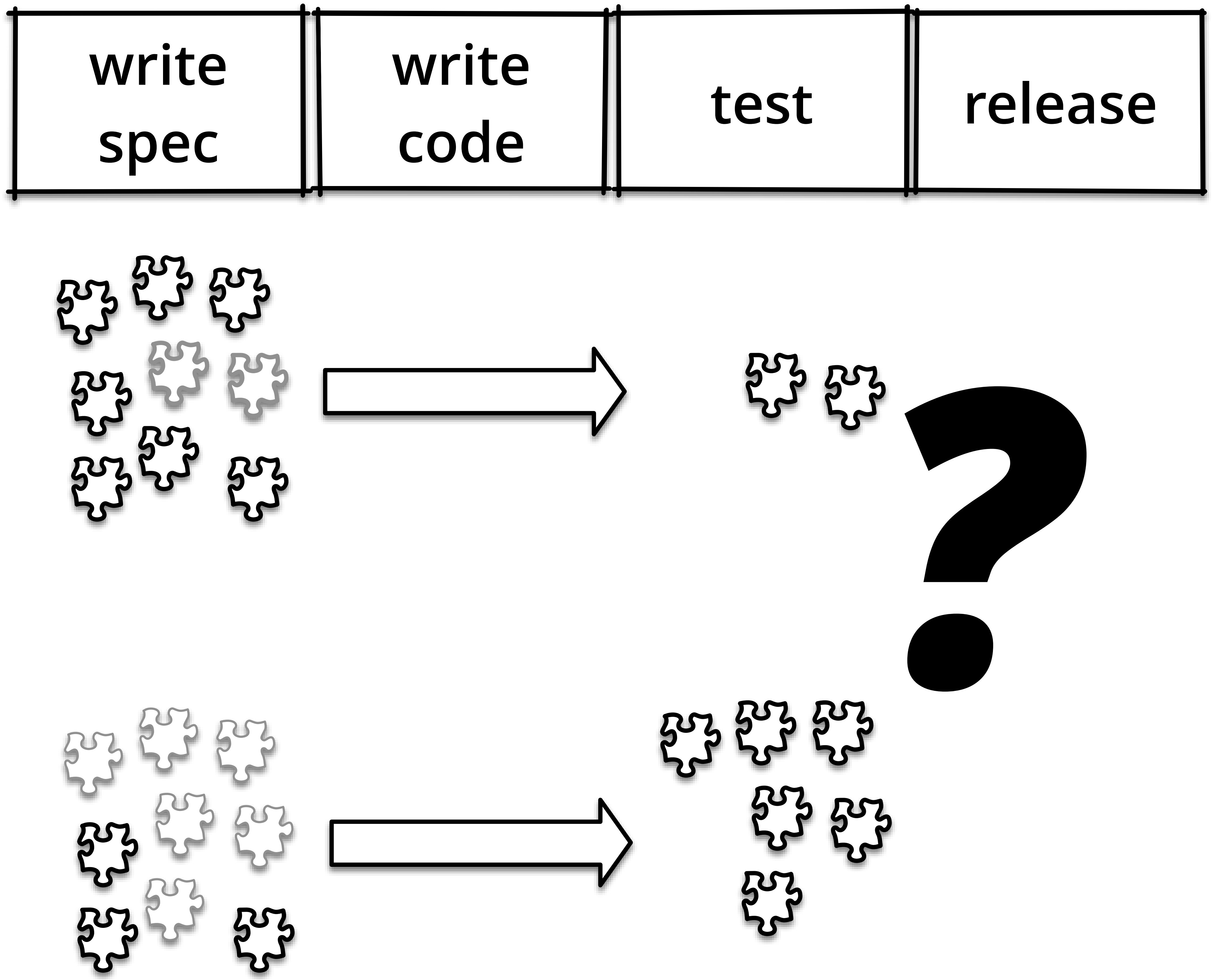


your beautiful microservices end up in a tangled, coupled mess

6 week periods of "hardening"

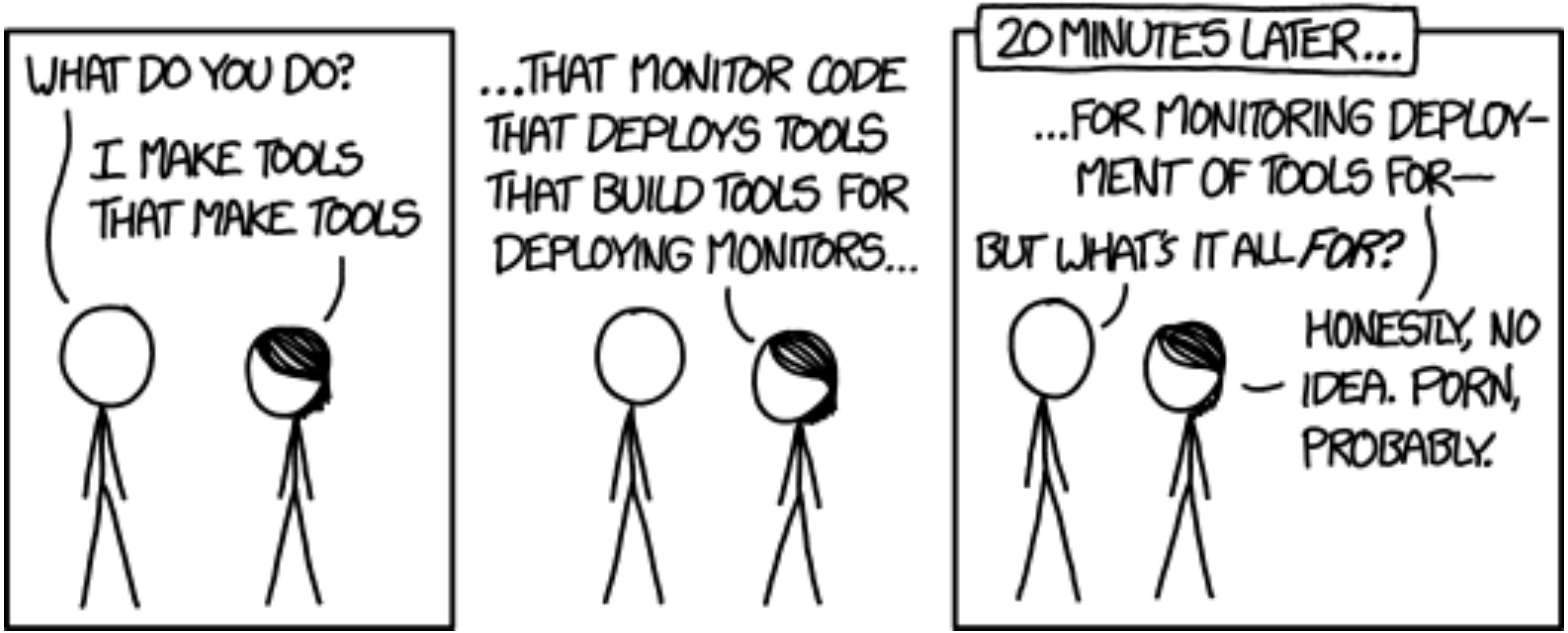


deploying everything all at once

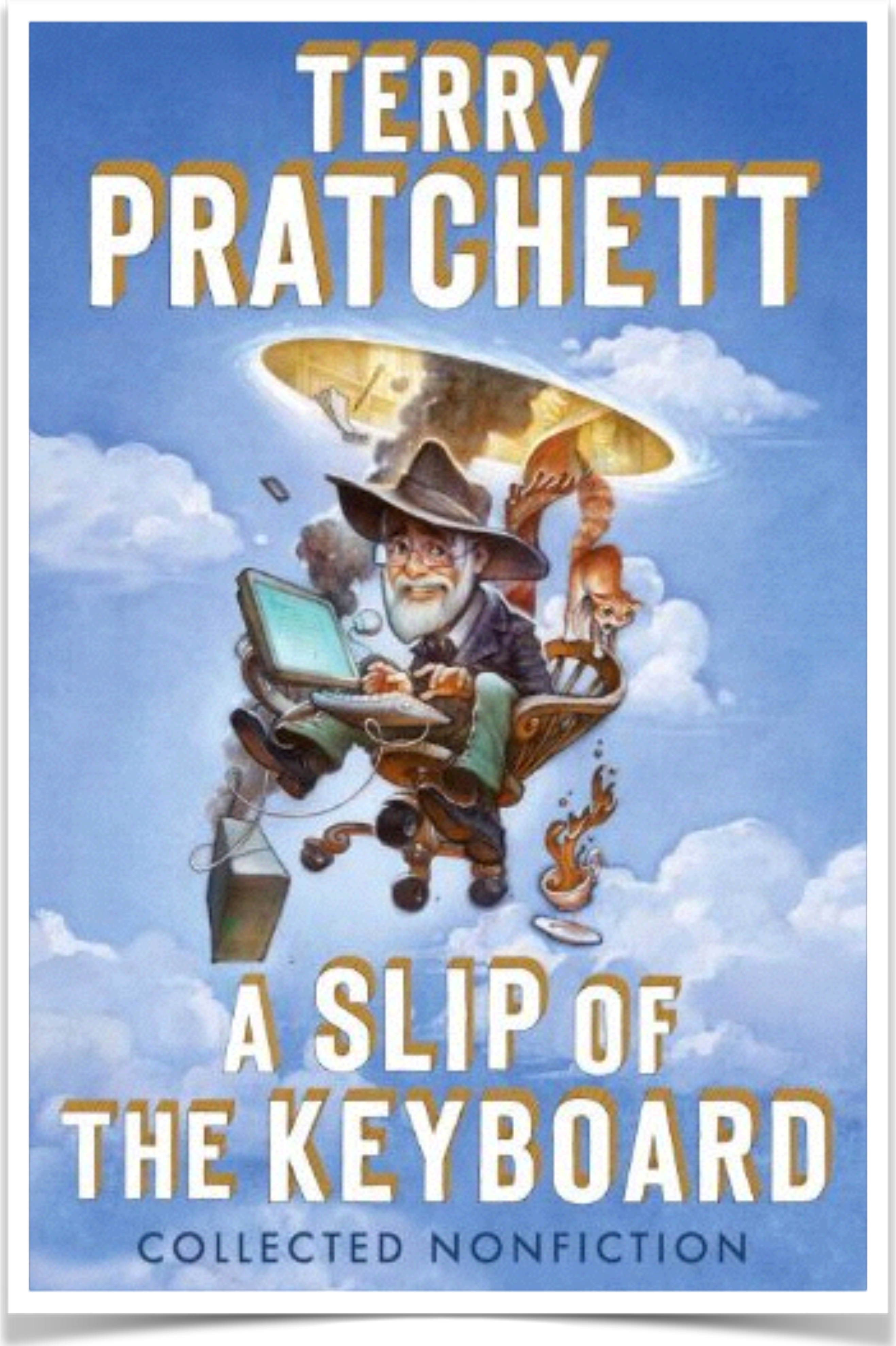


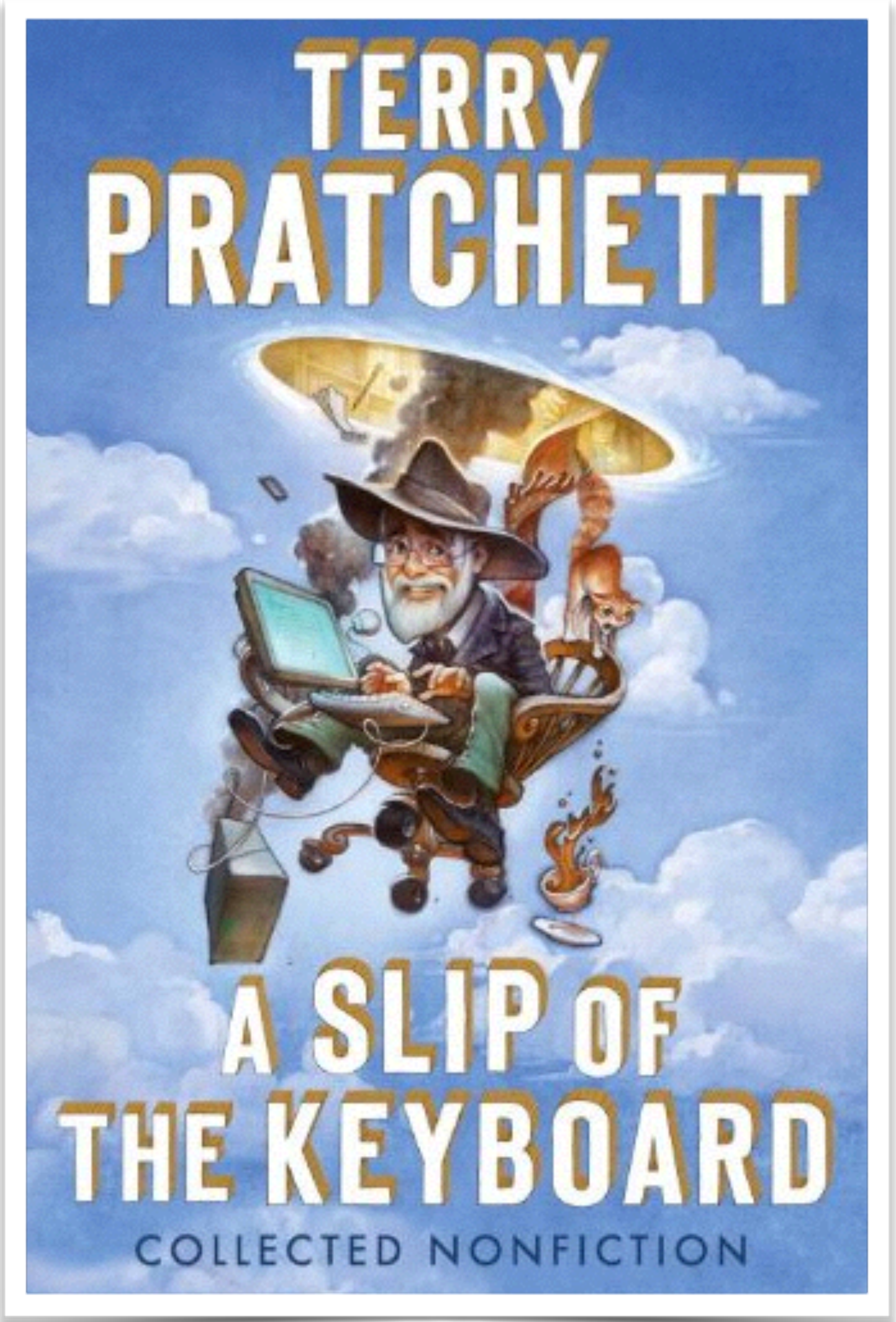
Without deploying into production, inventory is built up - inventory costs money and the more we have the more risky our deployments

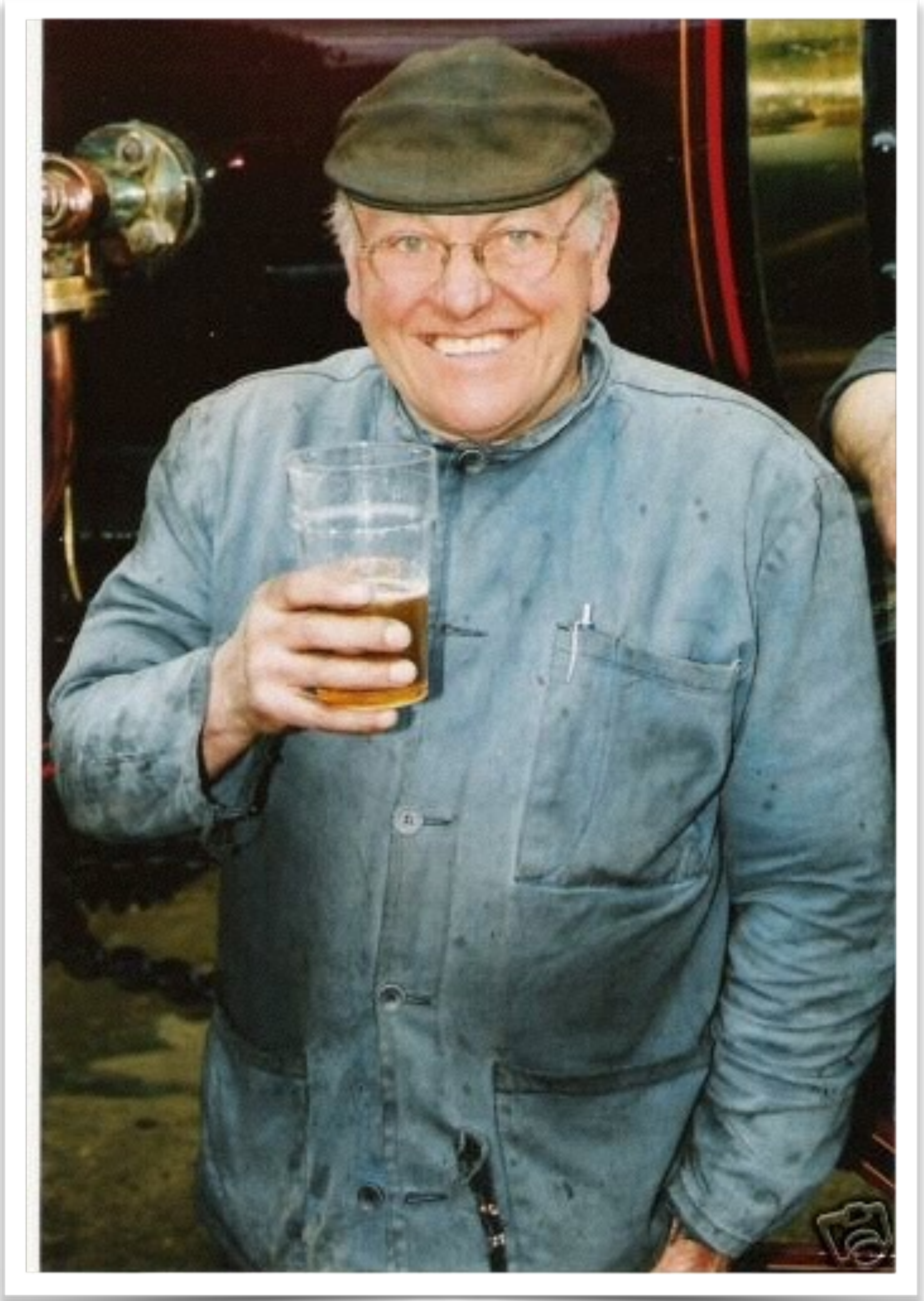
beware the distributed monolith!



https://xkcd.com/1629/


TERRY
PRATCHETT
A SLIP OF
THE KEYBOARD
COLLECTED NONFICTION


TERRY
PRATCHETT
A SLIP OF
THE KEYBOARD
COLLECTED NONFICTION



The Boojam!

Large organisations tend to be “functionally split” for efficiency reasons

Many use scrummerfall or project based teams

Throw away integration testing? Are you made?

We suddenly need a whole new set of skills

Design for failure

Architectural safety

Zookeeper?!?

Chapter 4

Facing these consequences

It is not possible to get the benefits
of microservices without serious organisational change

“...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**.

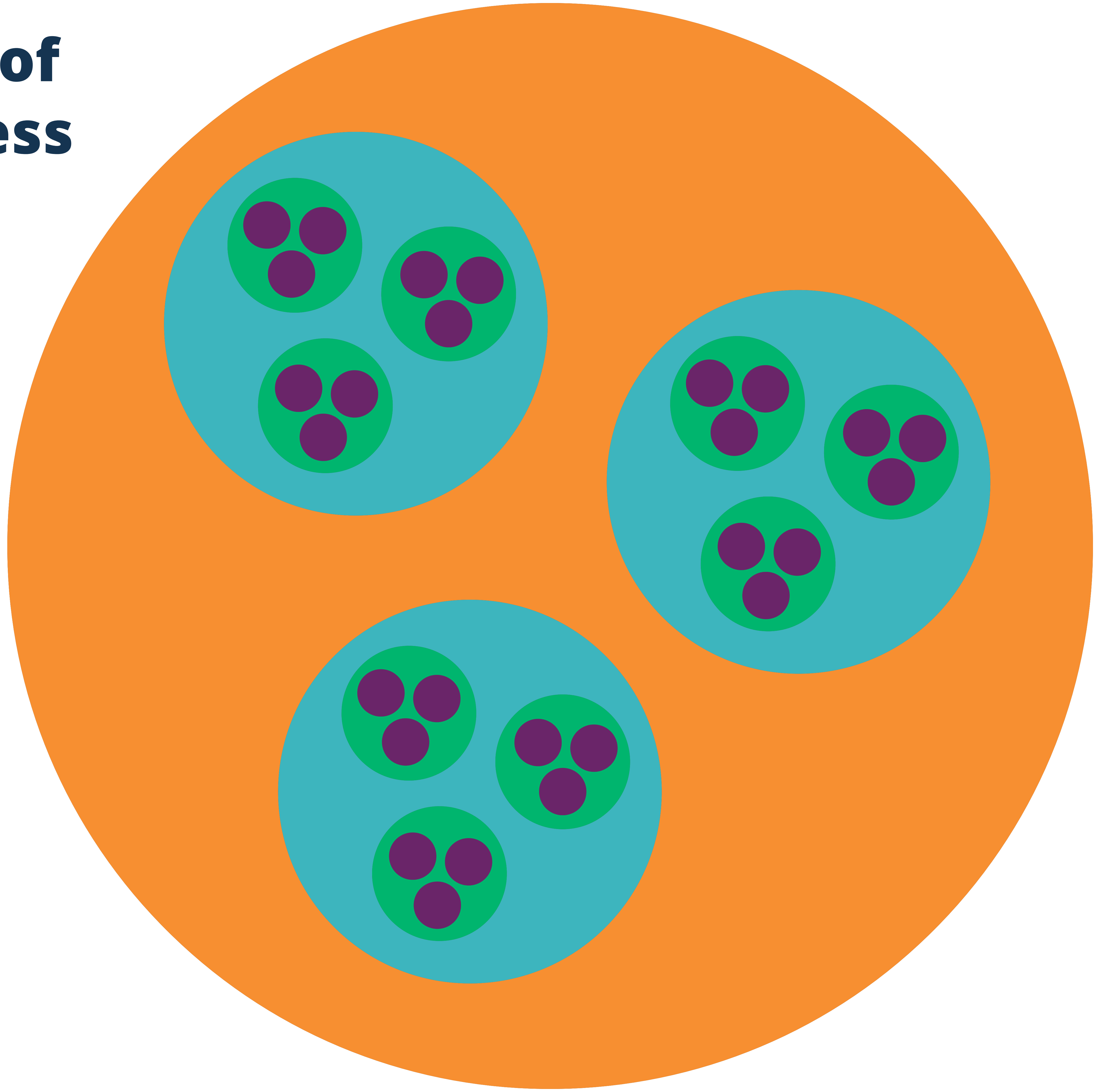
By contrast, in **loosely-coupled organizations**, a large, distributed team of volunteers develops the design. Face-to-face communications are rare given most developers never meet. Hence fewer connections between modules are established. **The architecture that evolves is more modular** as a result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

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

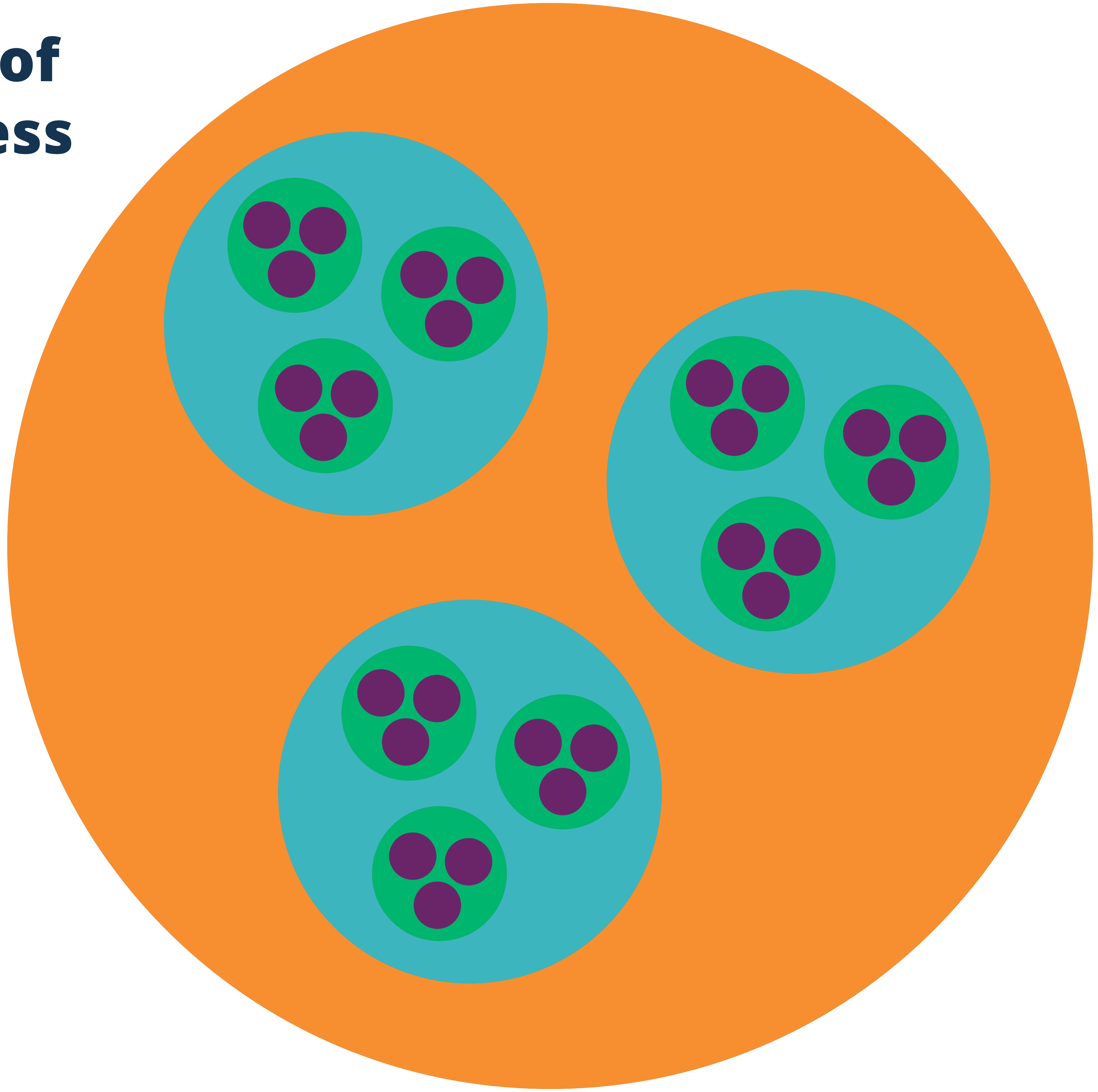
http://www.hbs.edu/faculty/Publication%20Files/08-039_1861e507-1dc1-4602-85b8-90d71559d85b.pdf *tightly-coupled* organizations ⇒ the design becomes more *tightly-coupled*.

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

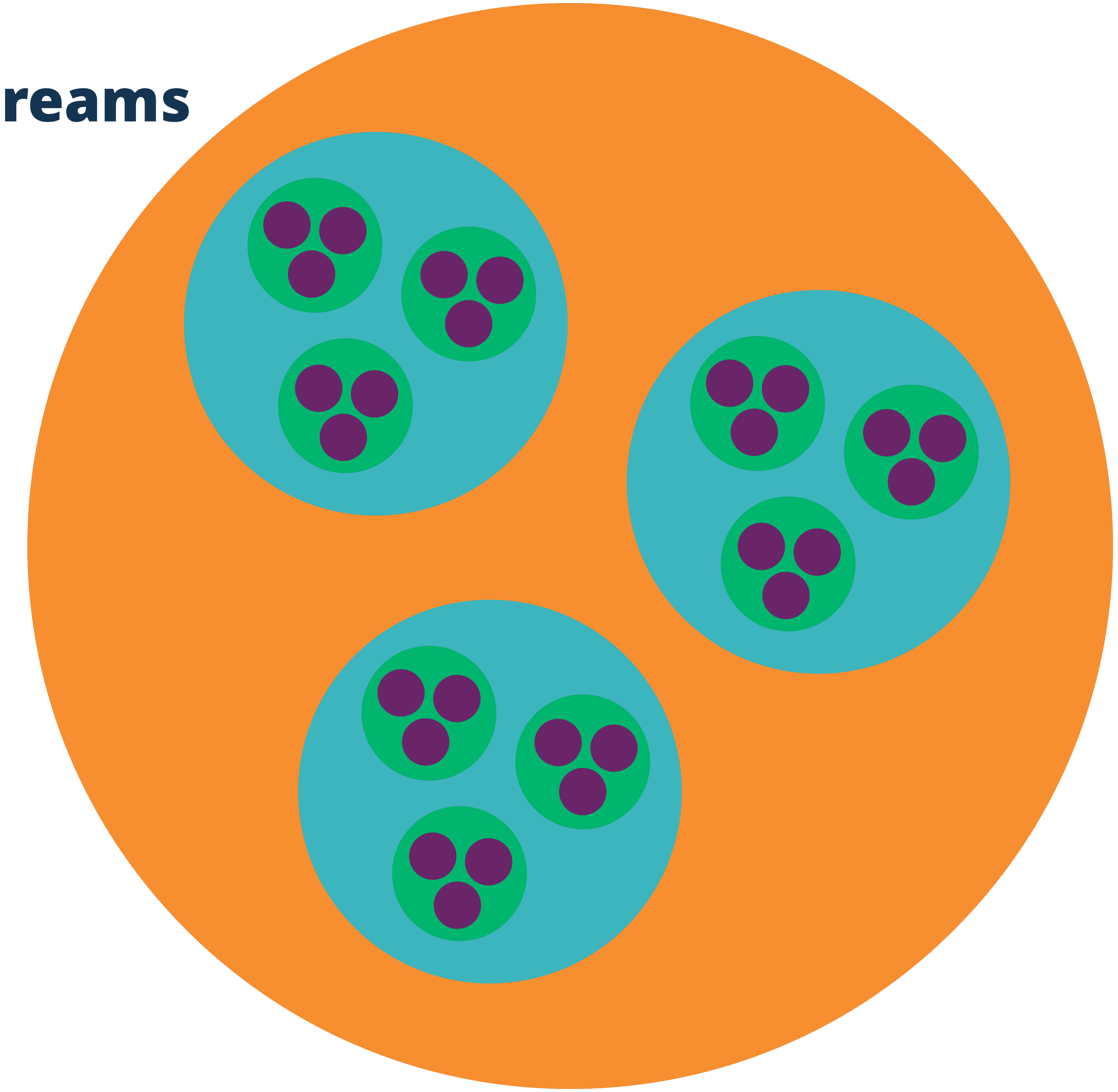
Lines of business



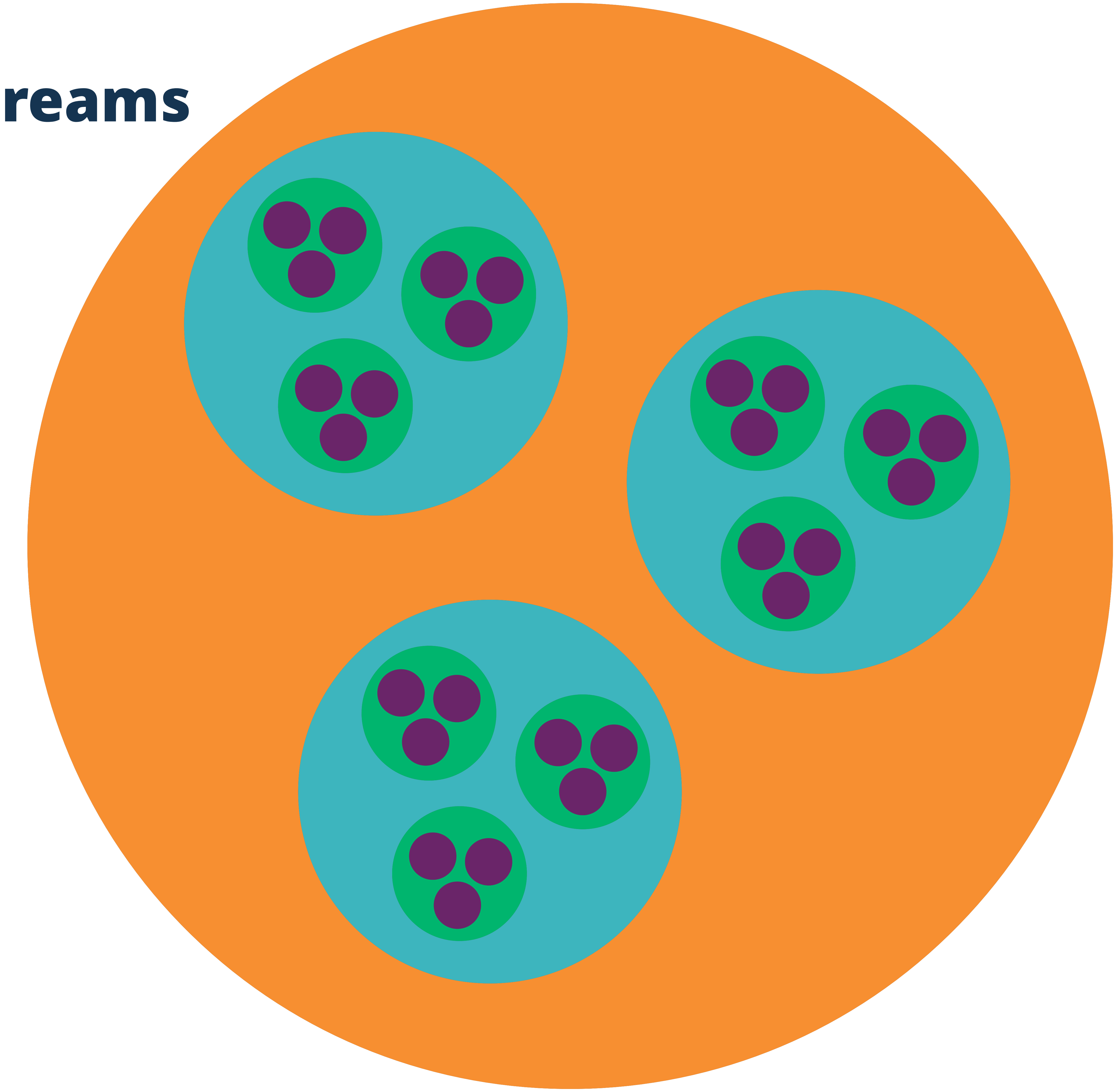
Lines of business



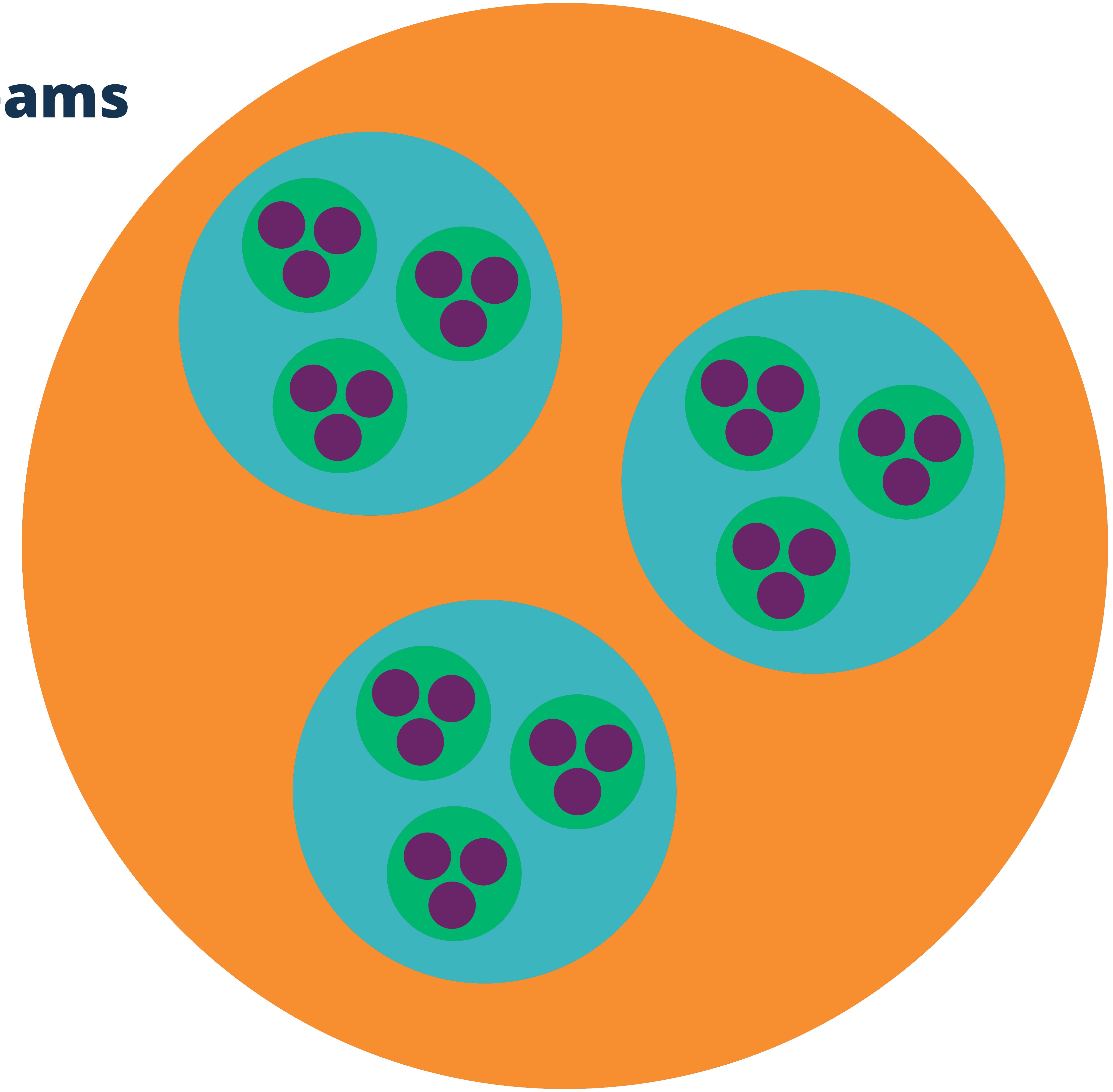
Value streams



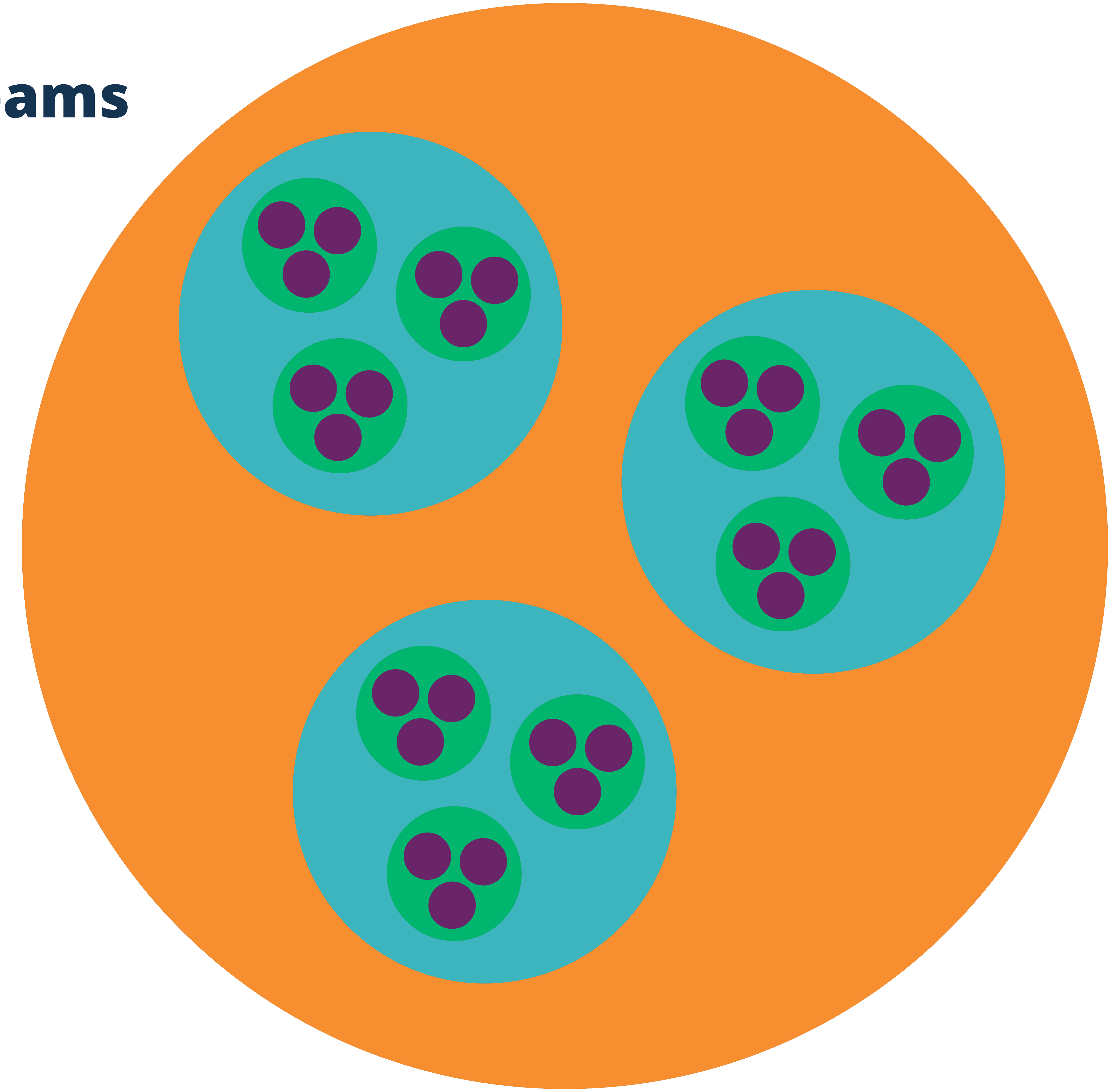
Value streams



teams



teams



each team

owns **one *or* more *services***

~10-20

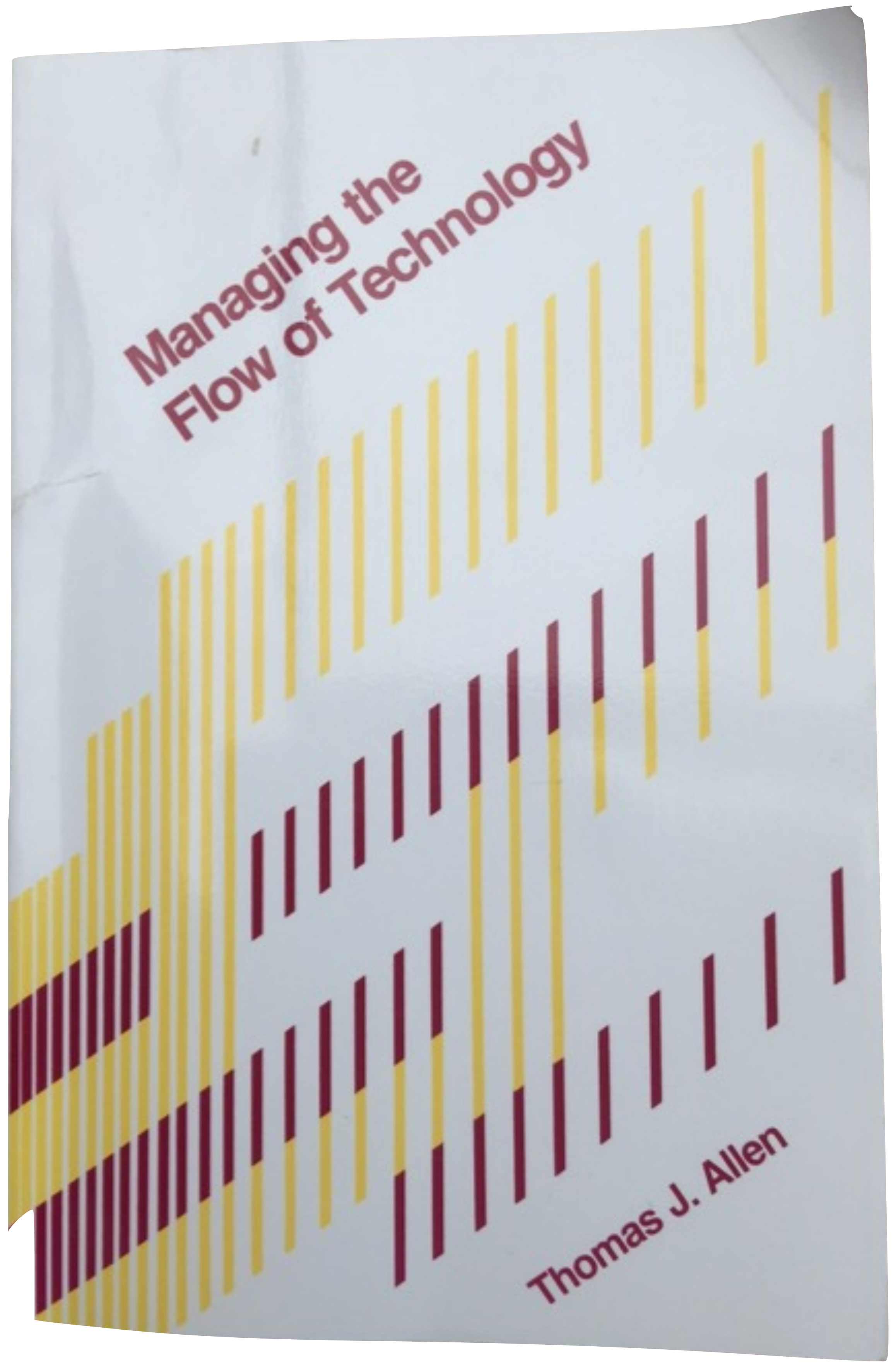
~10-20

~160-200

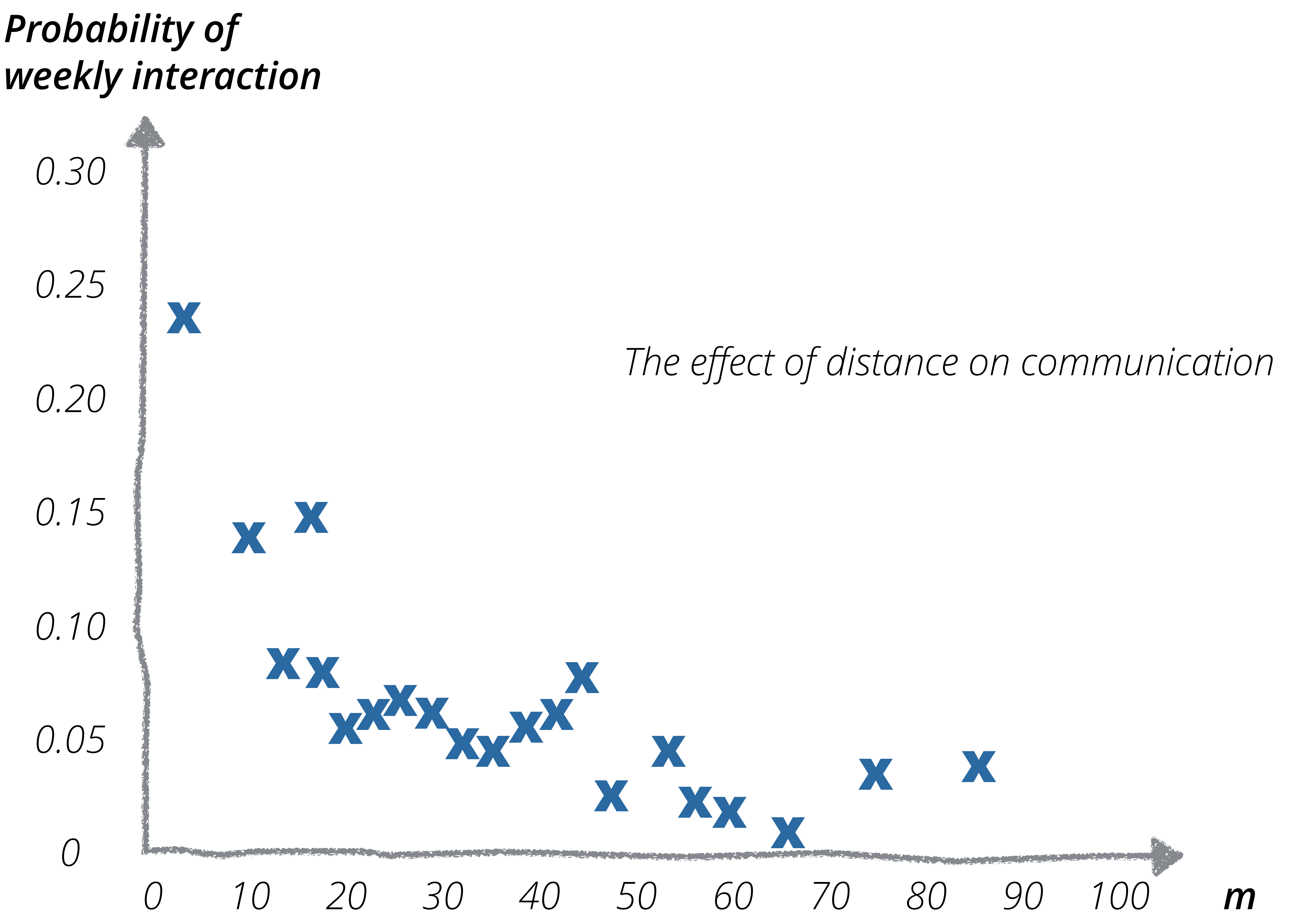
~10-20

~160-200

multiples thereof

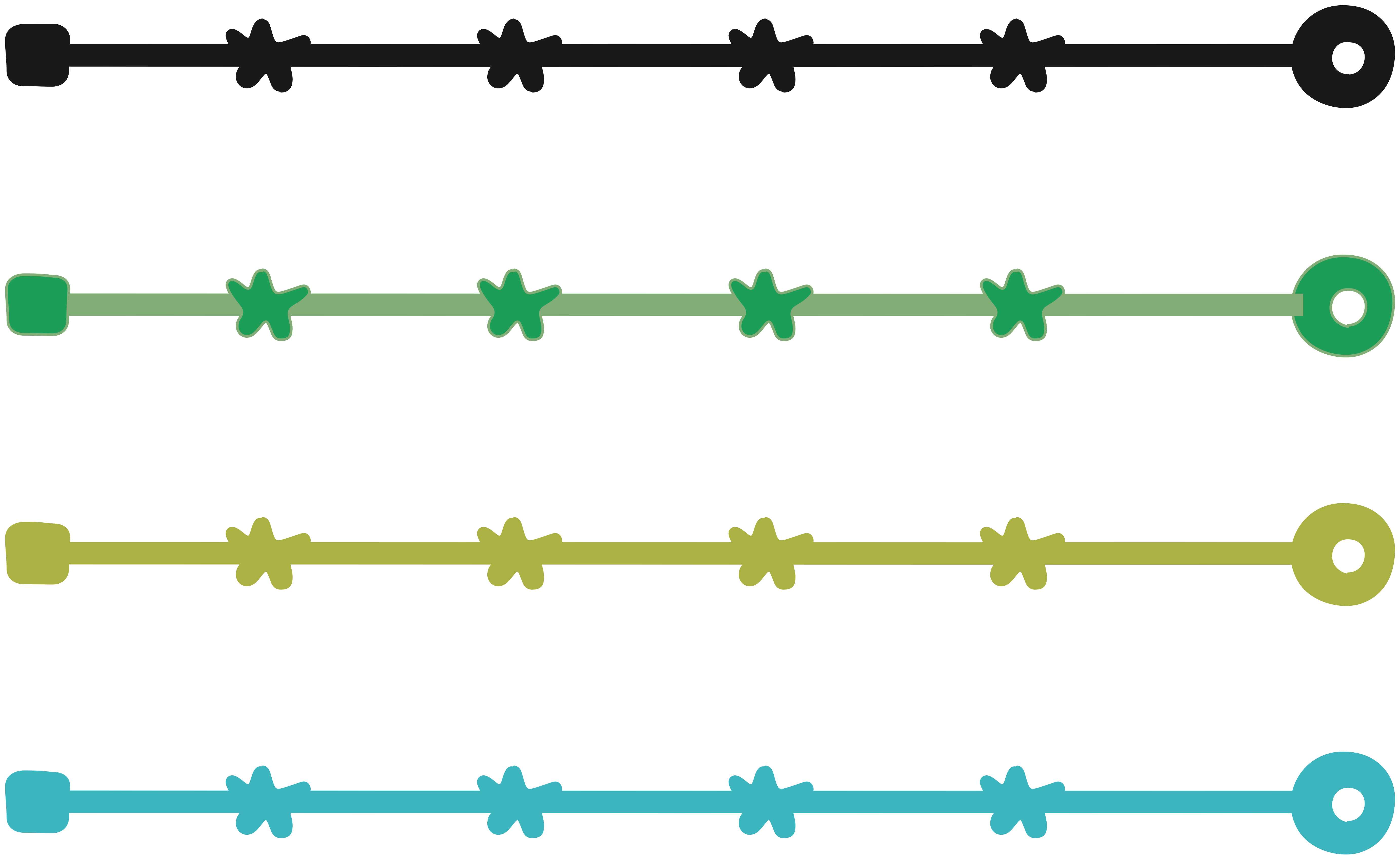


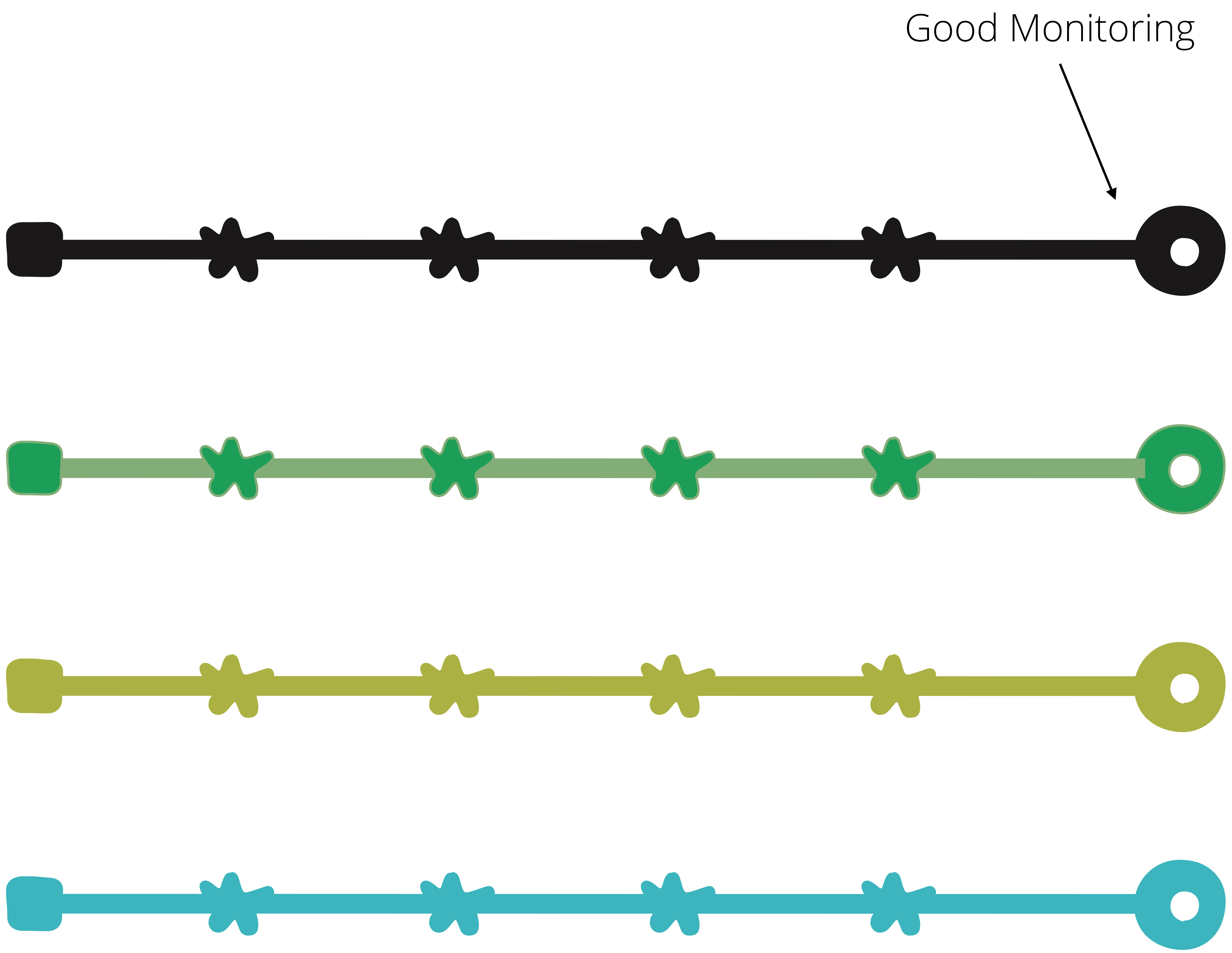
Thomas J. Allen, 1977

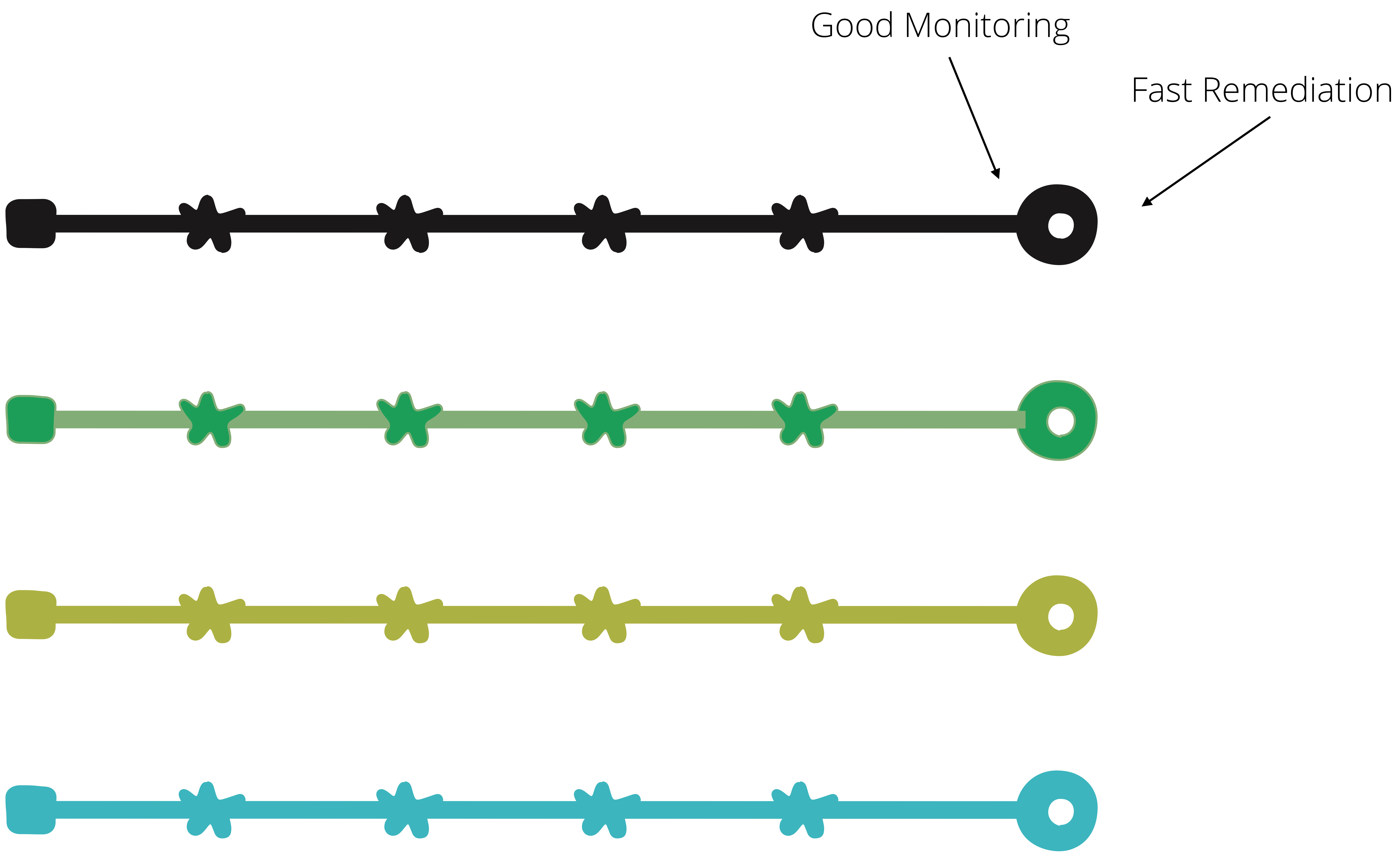

Probability of
weekly interaction
0.30
0.25
0.20
0.15
0.10
0.05
0
The effect of distance on communication
0
10
20
30
40
50
60
70
80
90
100
m

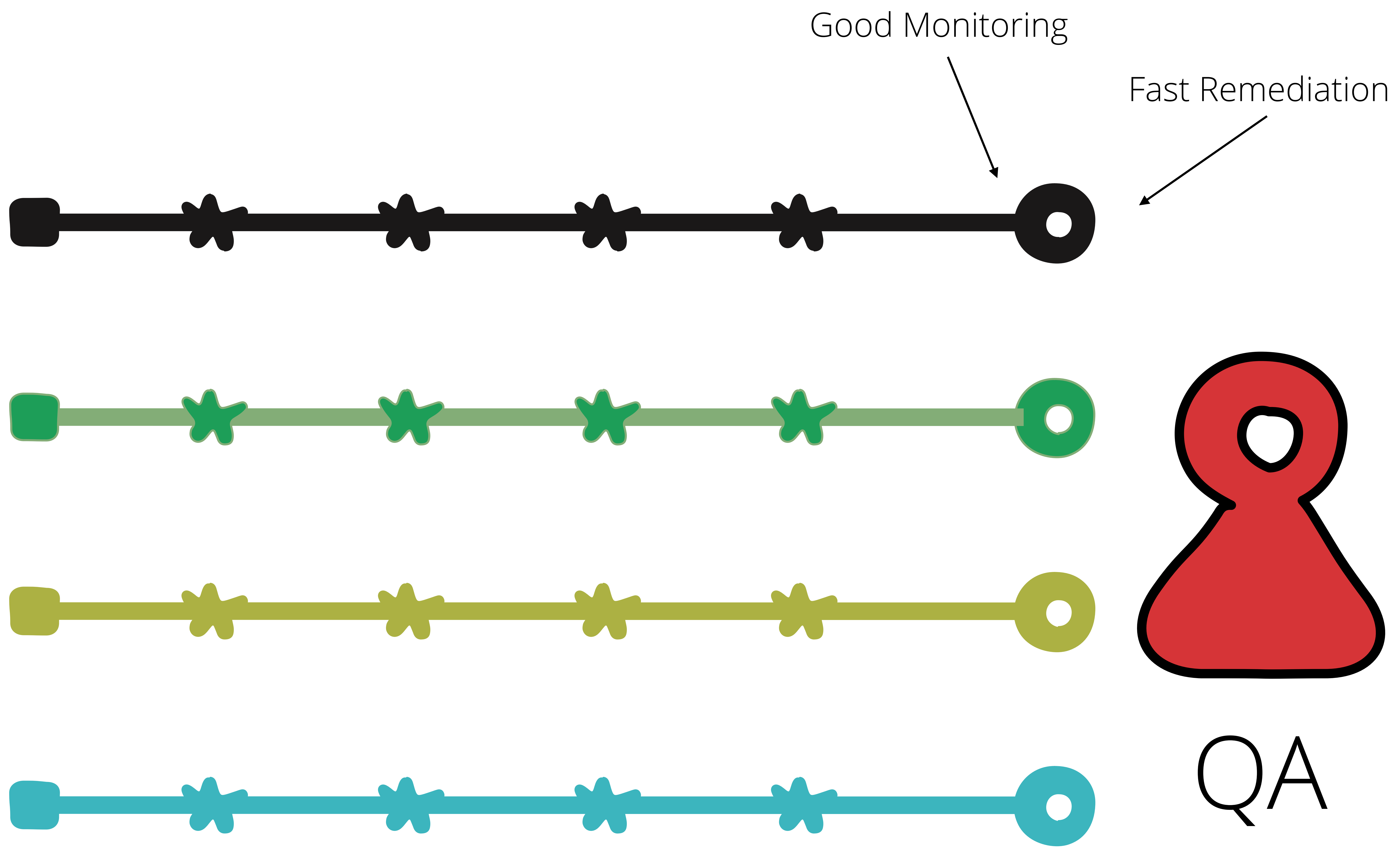
co-locate as much as possible

take advantage of serendipitous conversations

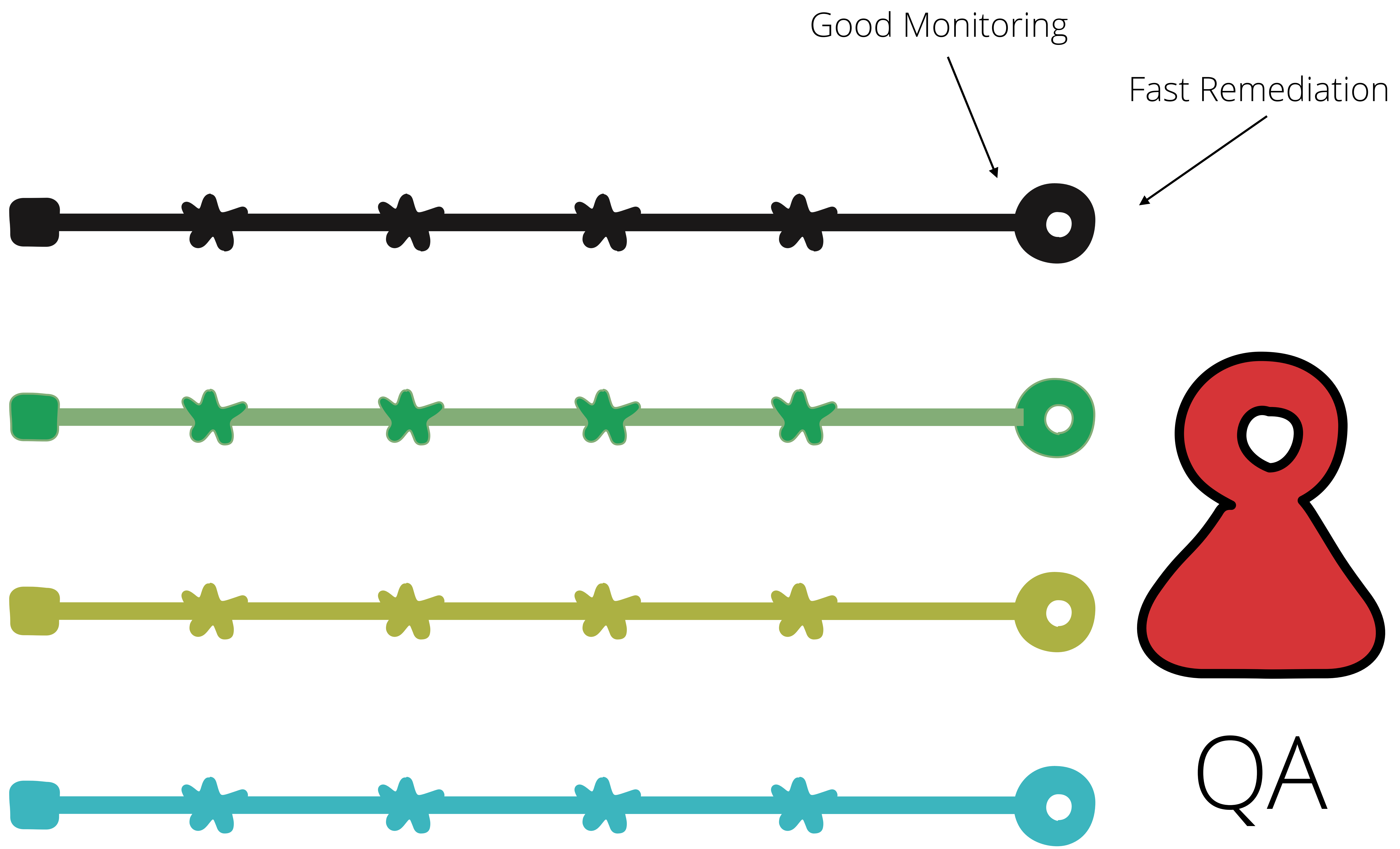



Good Monitoring

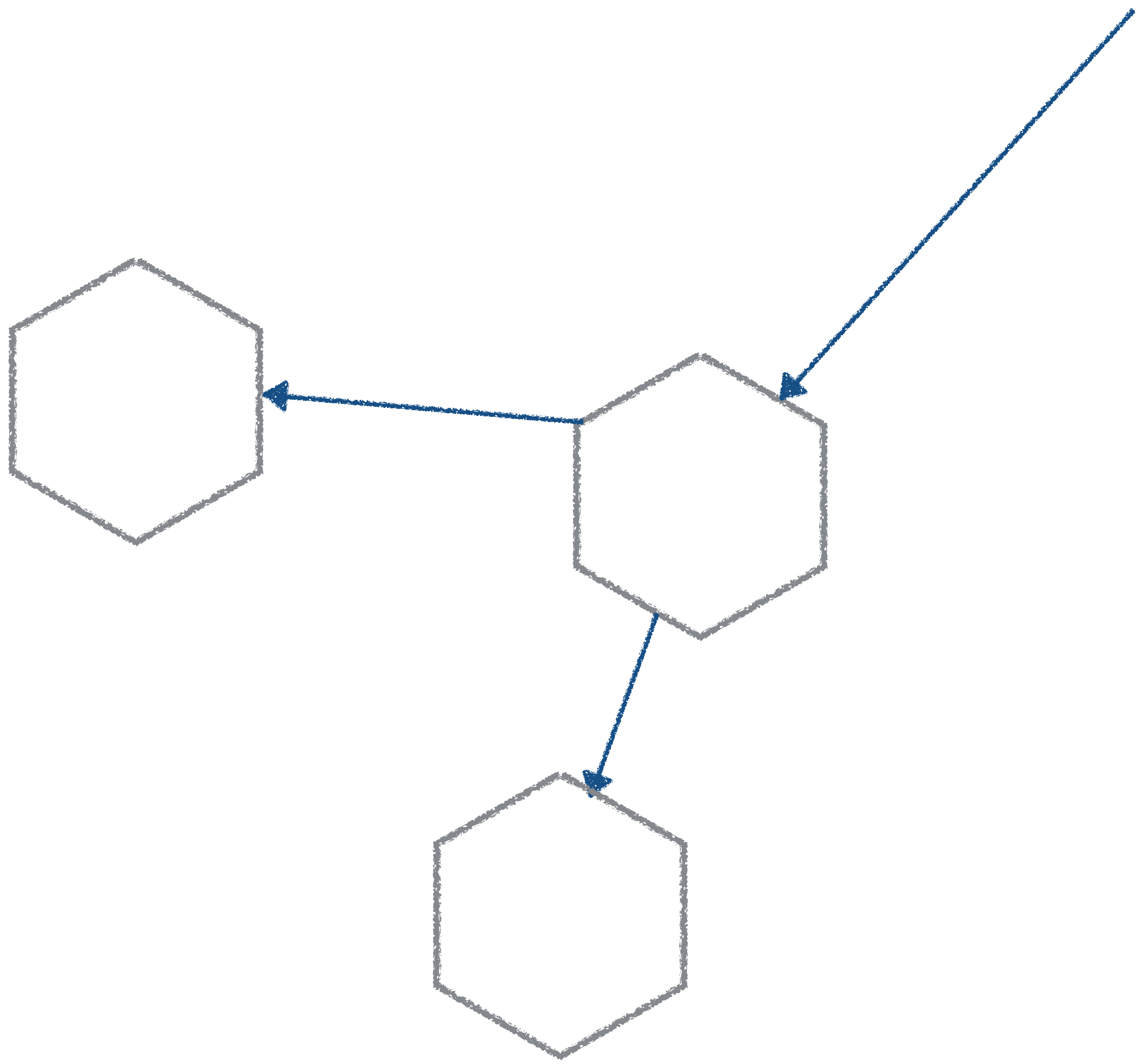

Good Monitoring
Fast Remediation

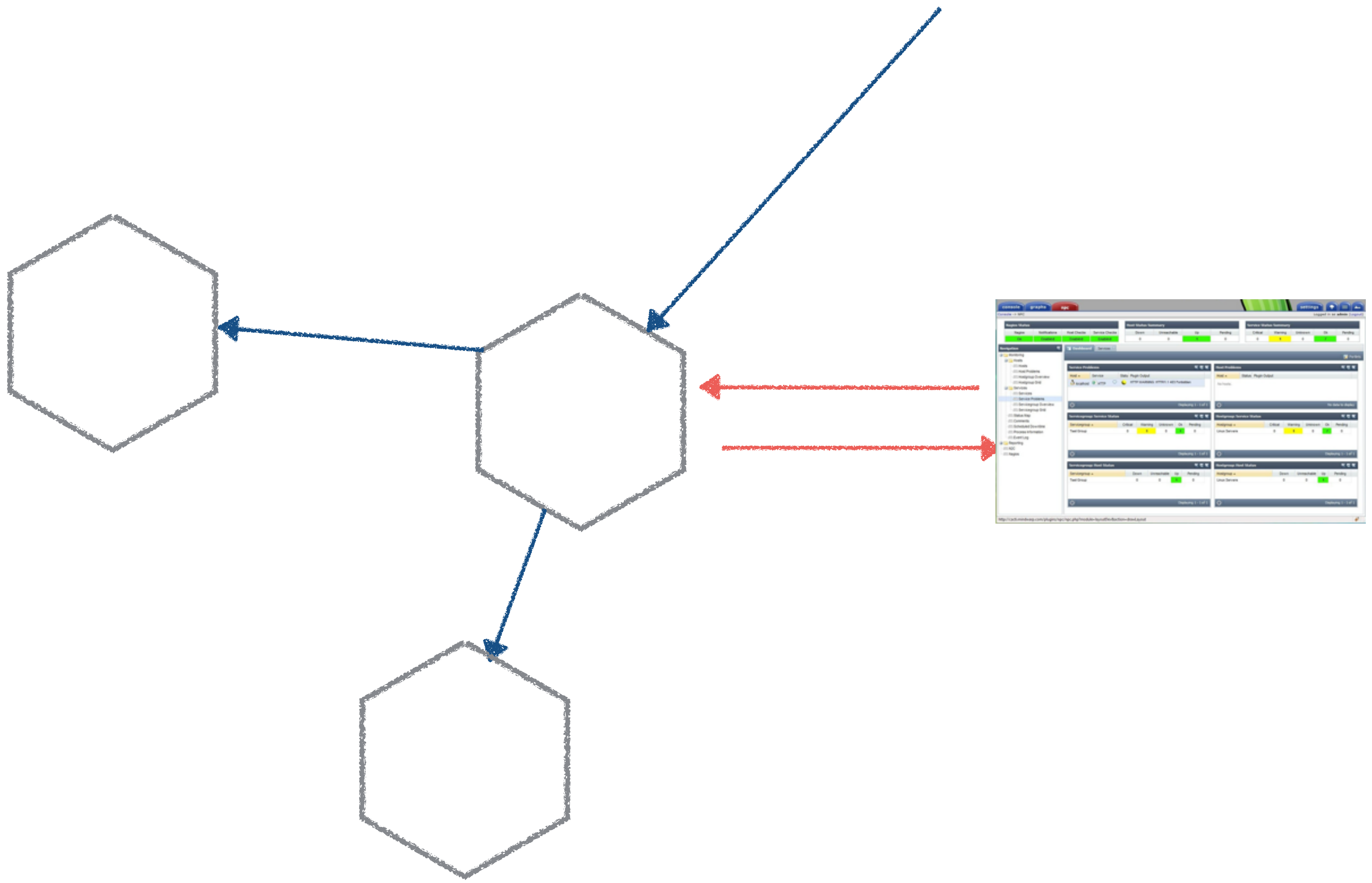


Good Monitoring
Fast Remediation
QA

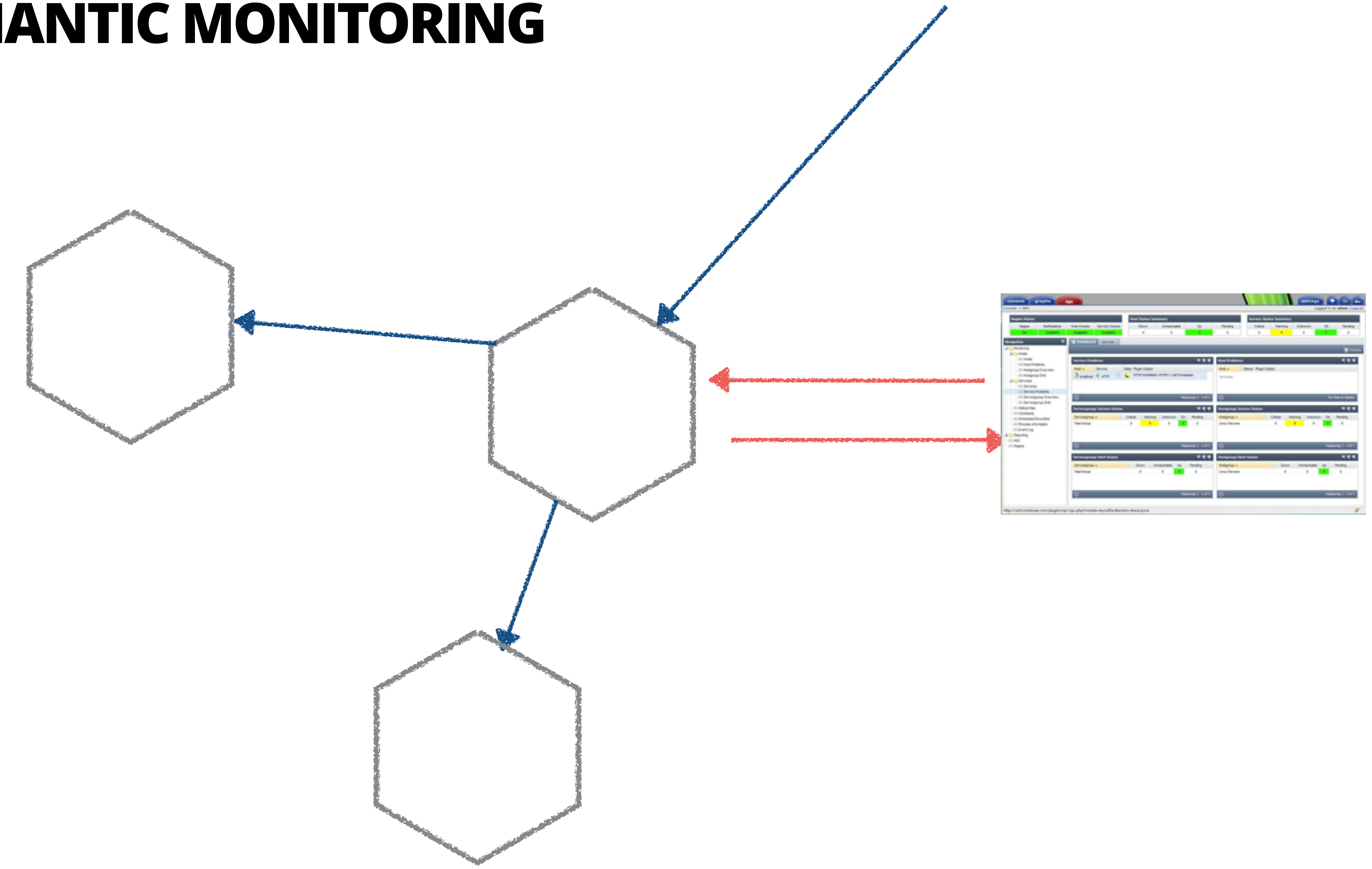


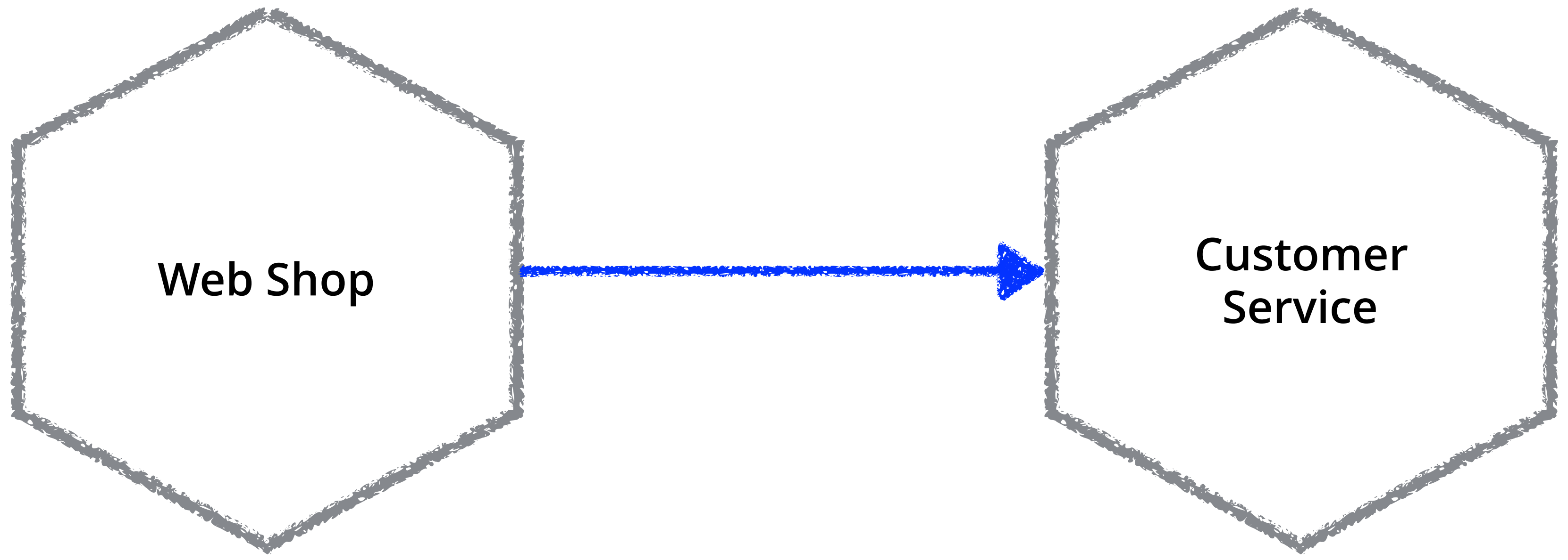
Test in production

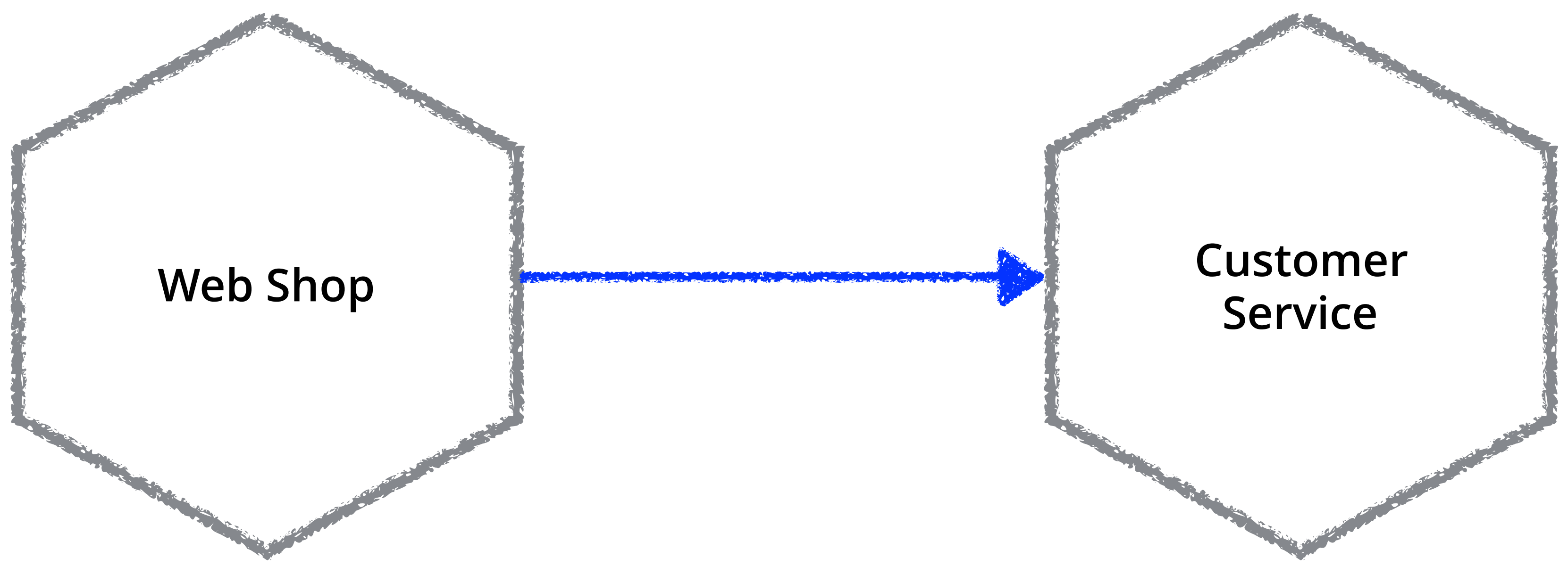




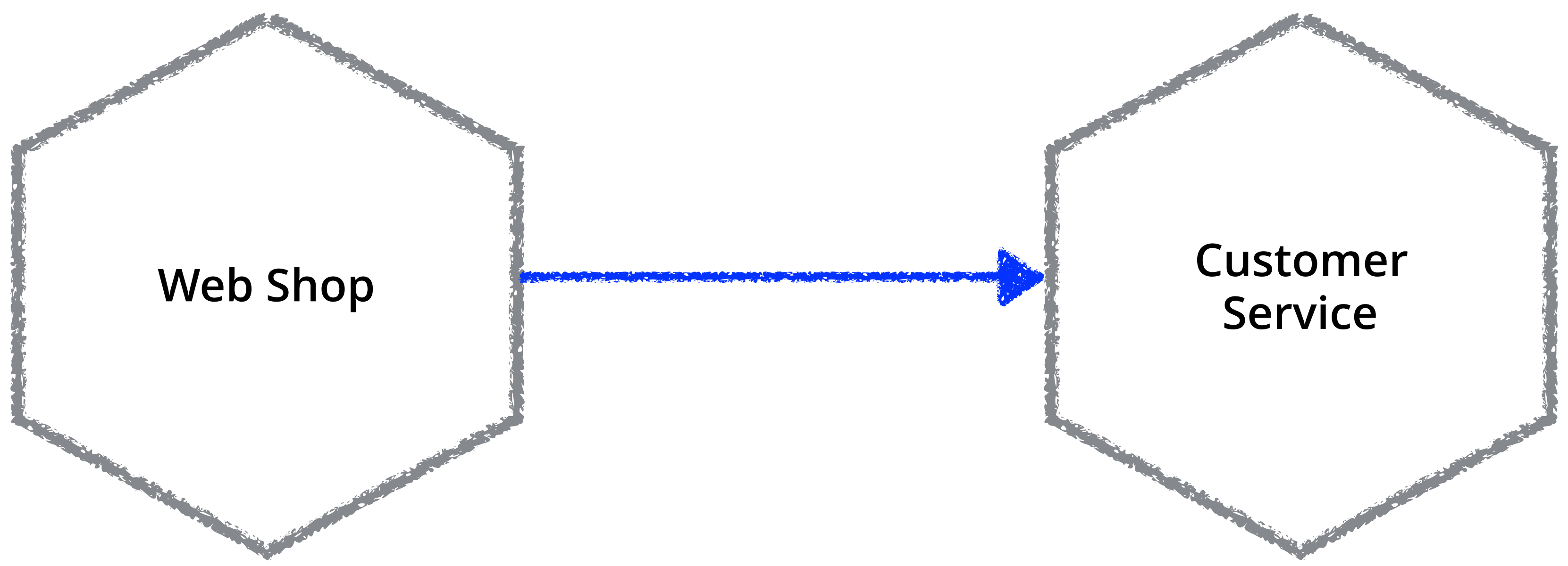
SEMANTIC MONITORING



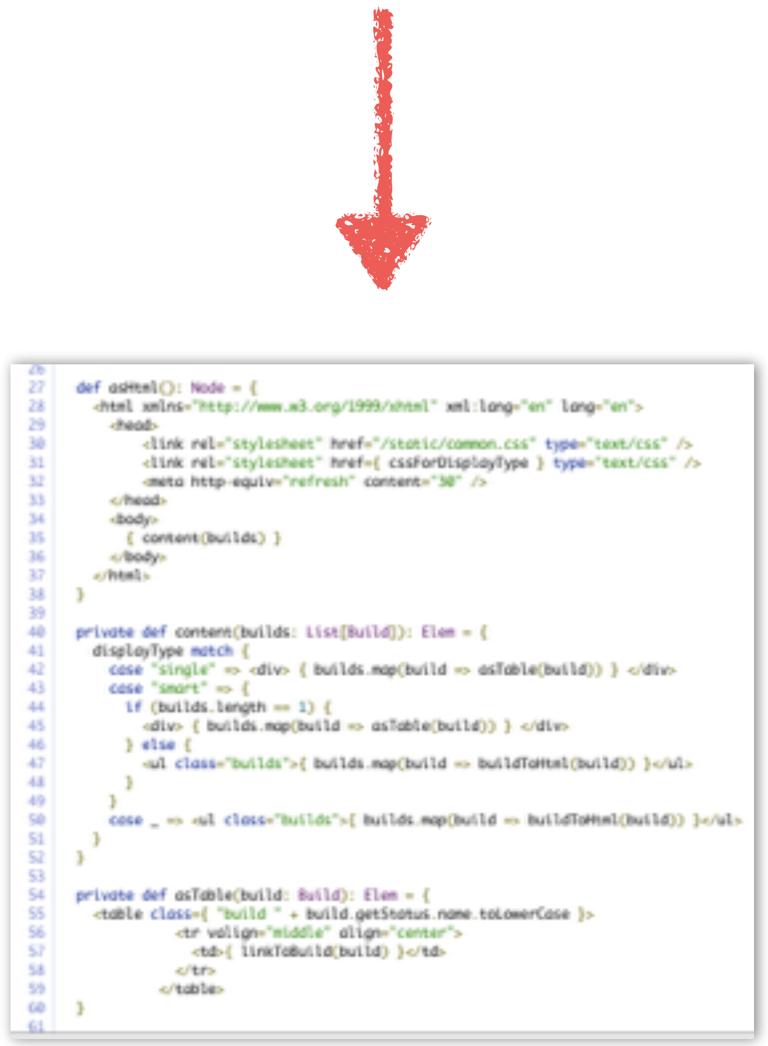

Web Shop
Customer Service

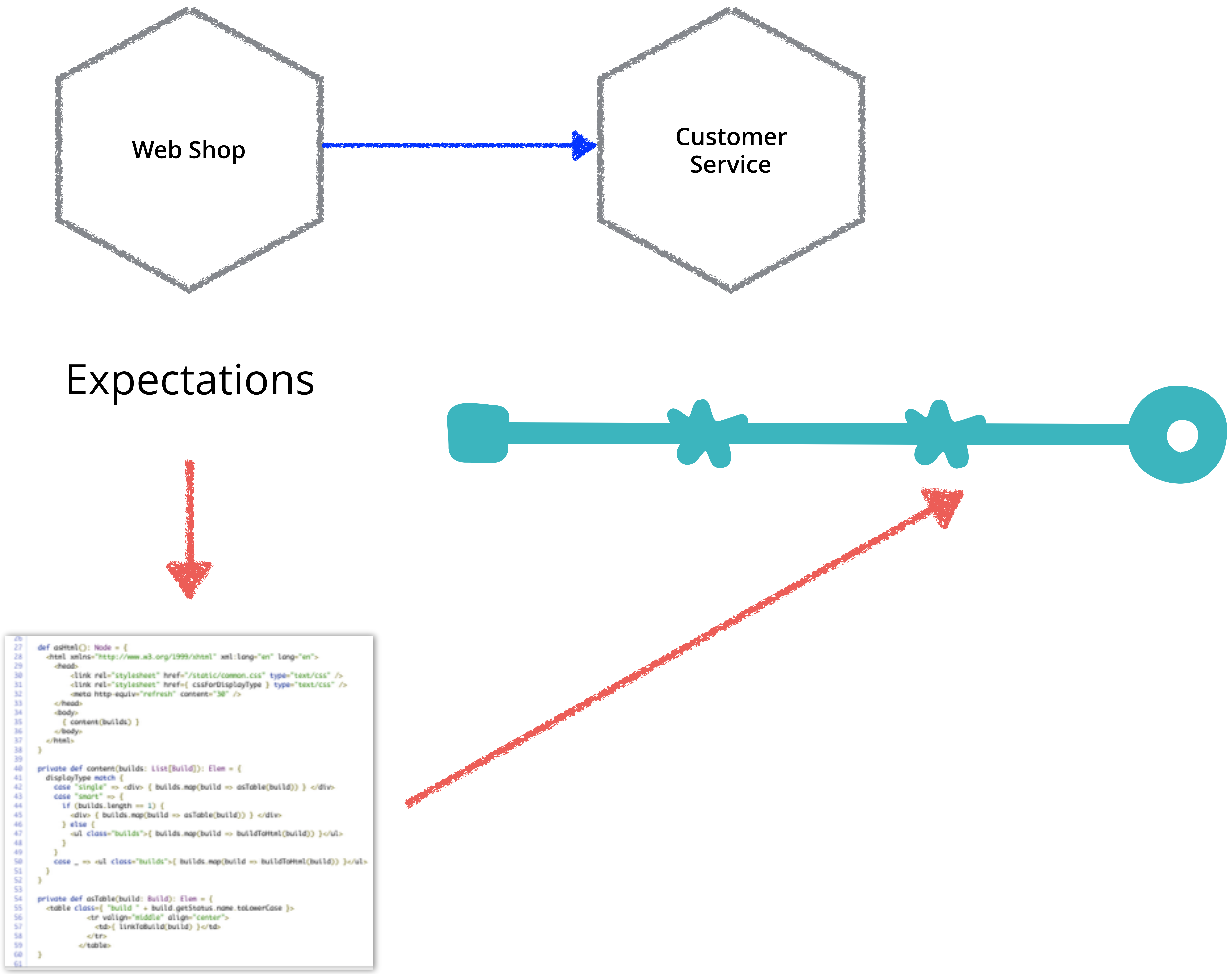


Expectations

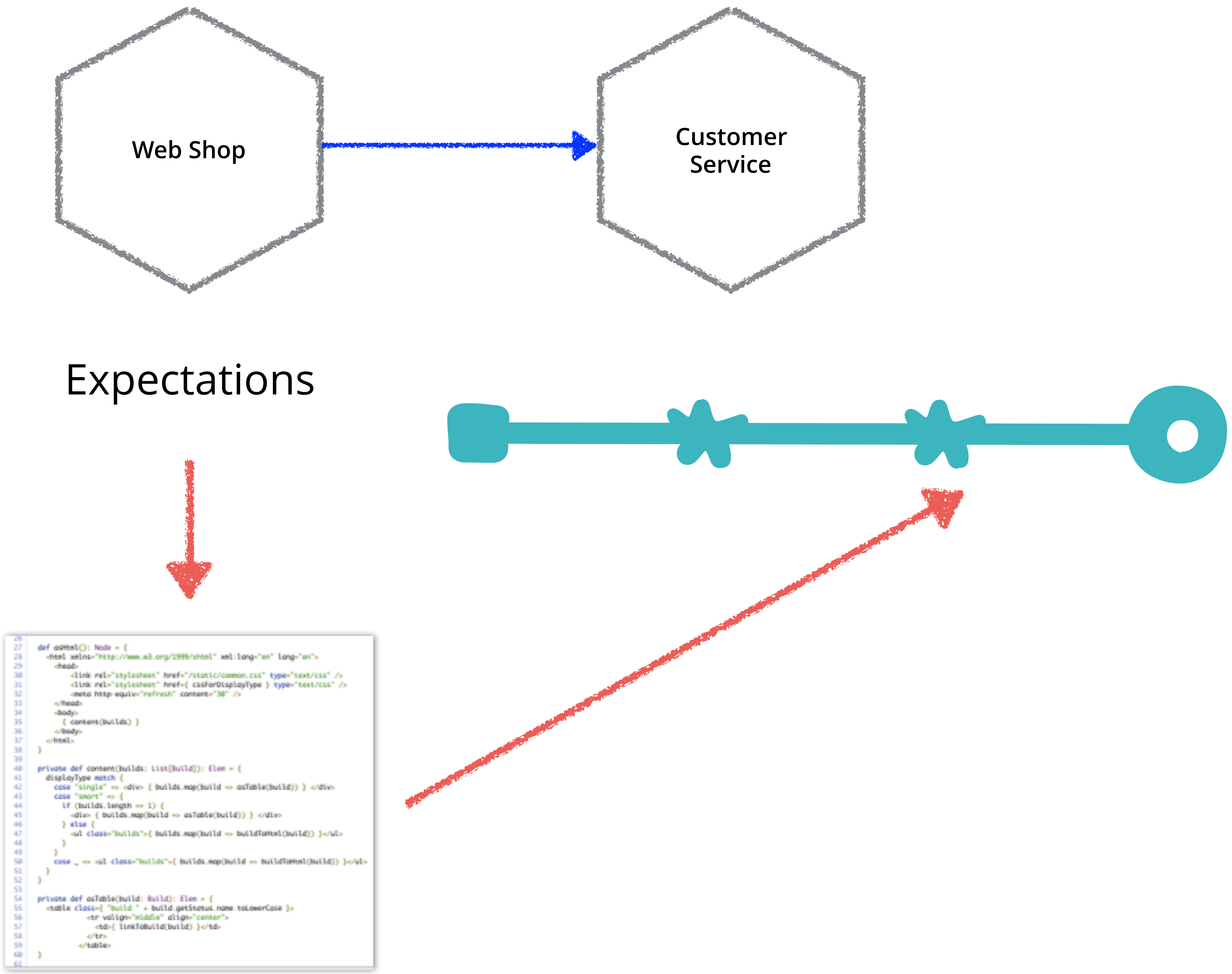


Expectations

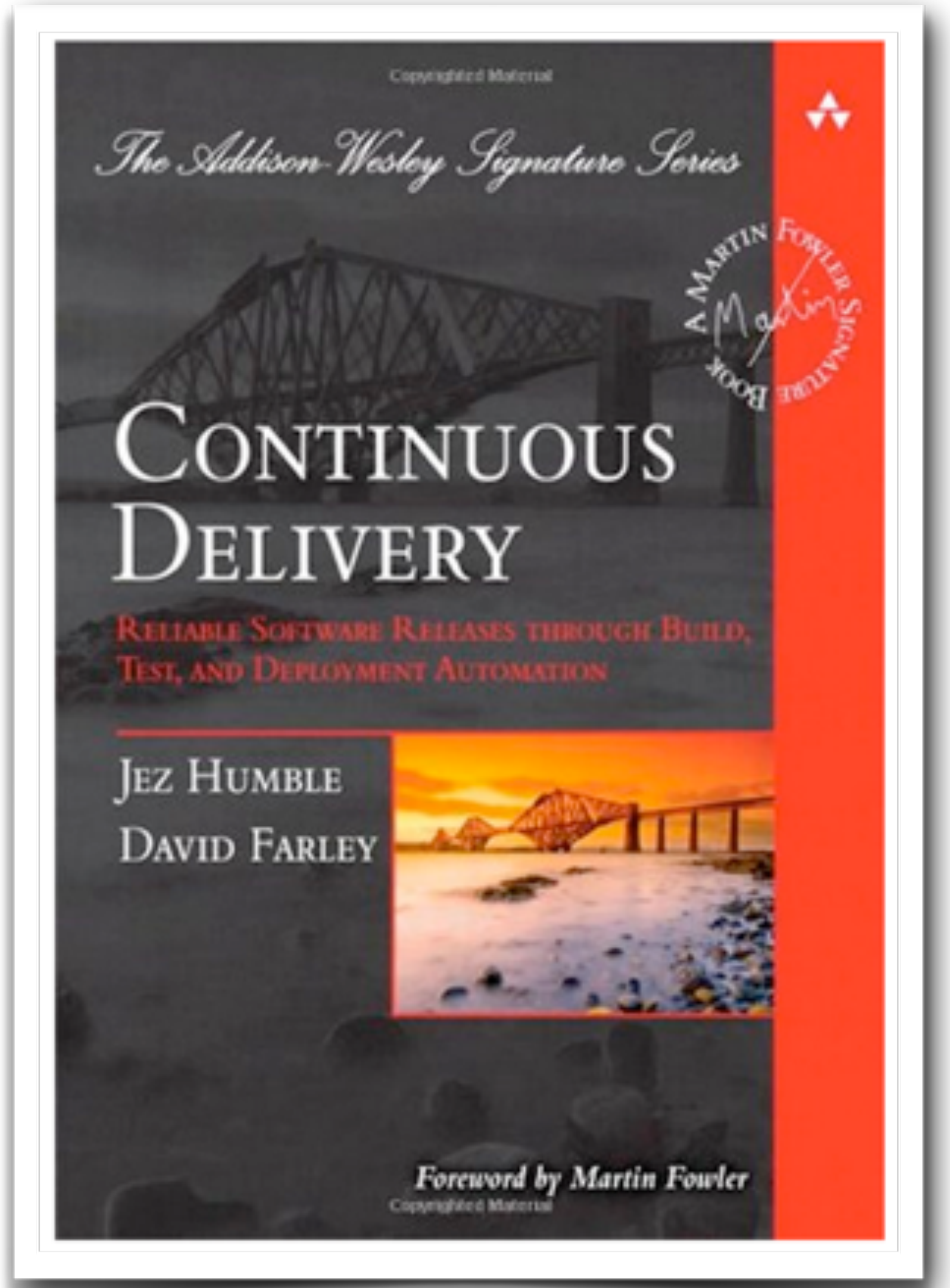



Web Shop
Customer Service
Expectations

Consumer Driven Contracts



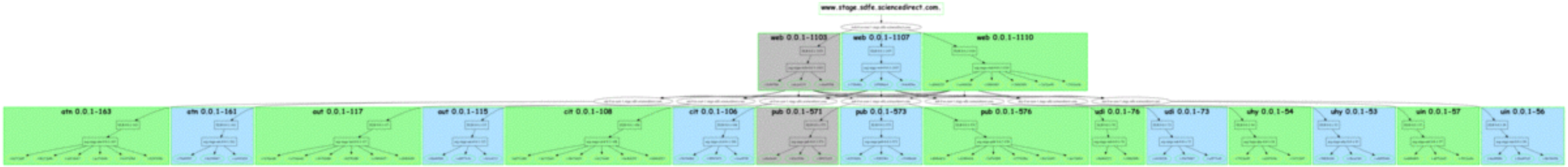
production != live



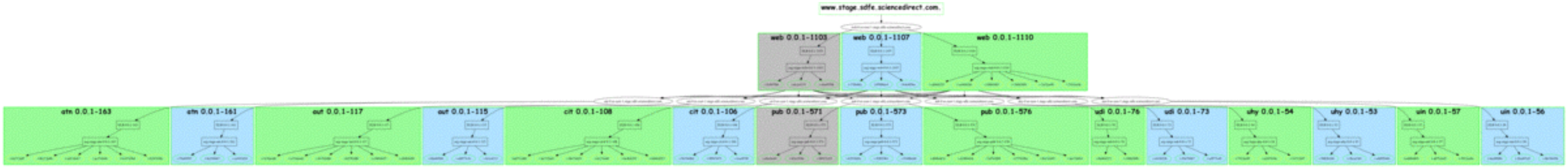
blue / green deploys

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

My hypothesis is that you can use organisational boundaries to reason about which testing patterns to apply and which integration patterns to use

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

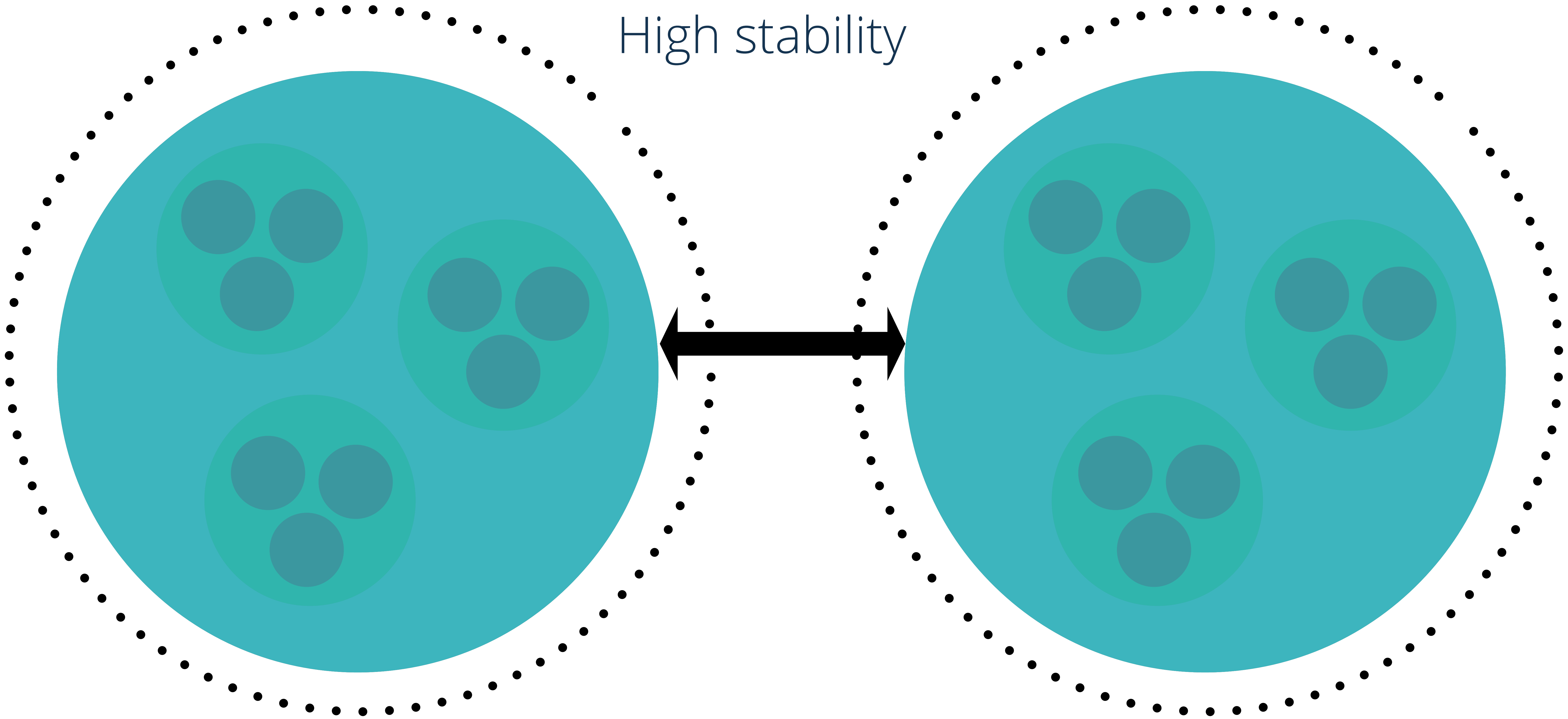
*I might need a better name for this

between organisational boundaries

typically requires:

Low change rate

High stability



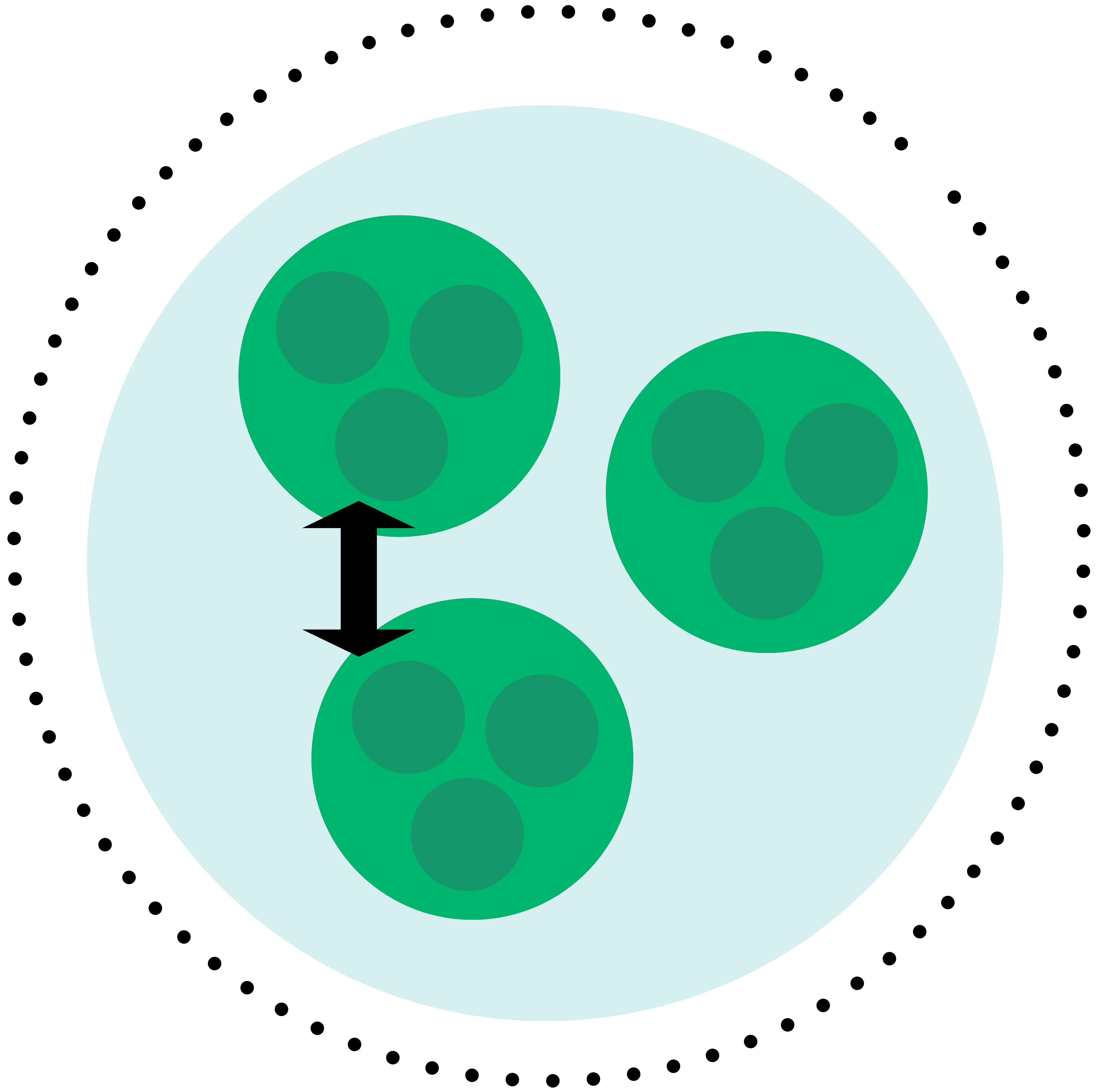
Semantic Versioning

Tolerant Reader

between business capabilities

Higher change rate

Lower stability



Semantic Versioning

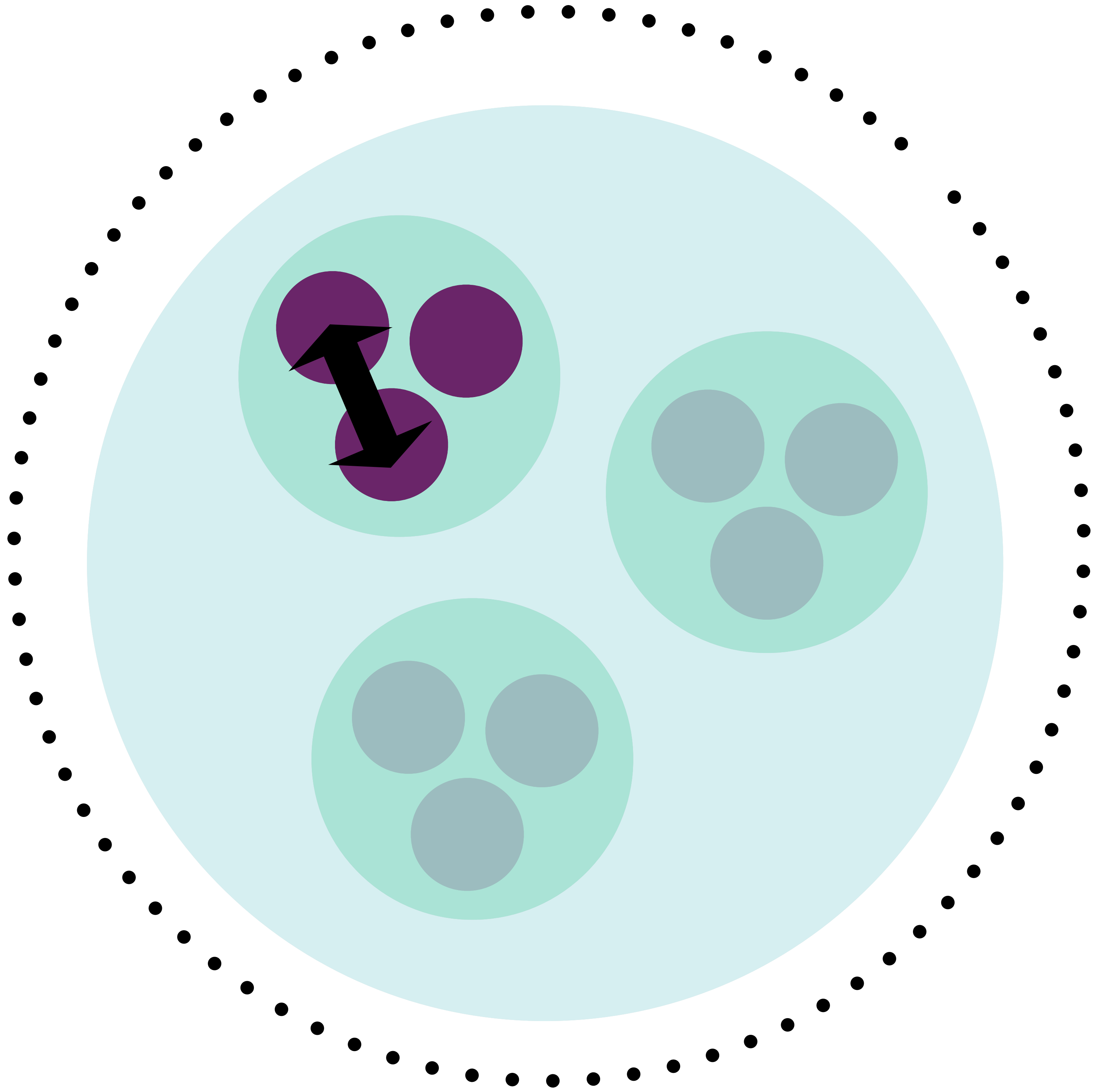
Contract Testing

Tolerant Reader

between teams

Higher rate of change

Lower stability



Semantic Versioning

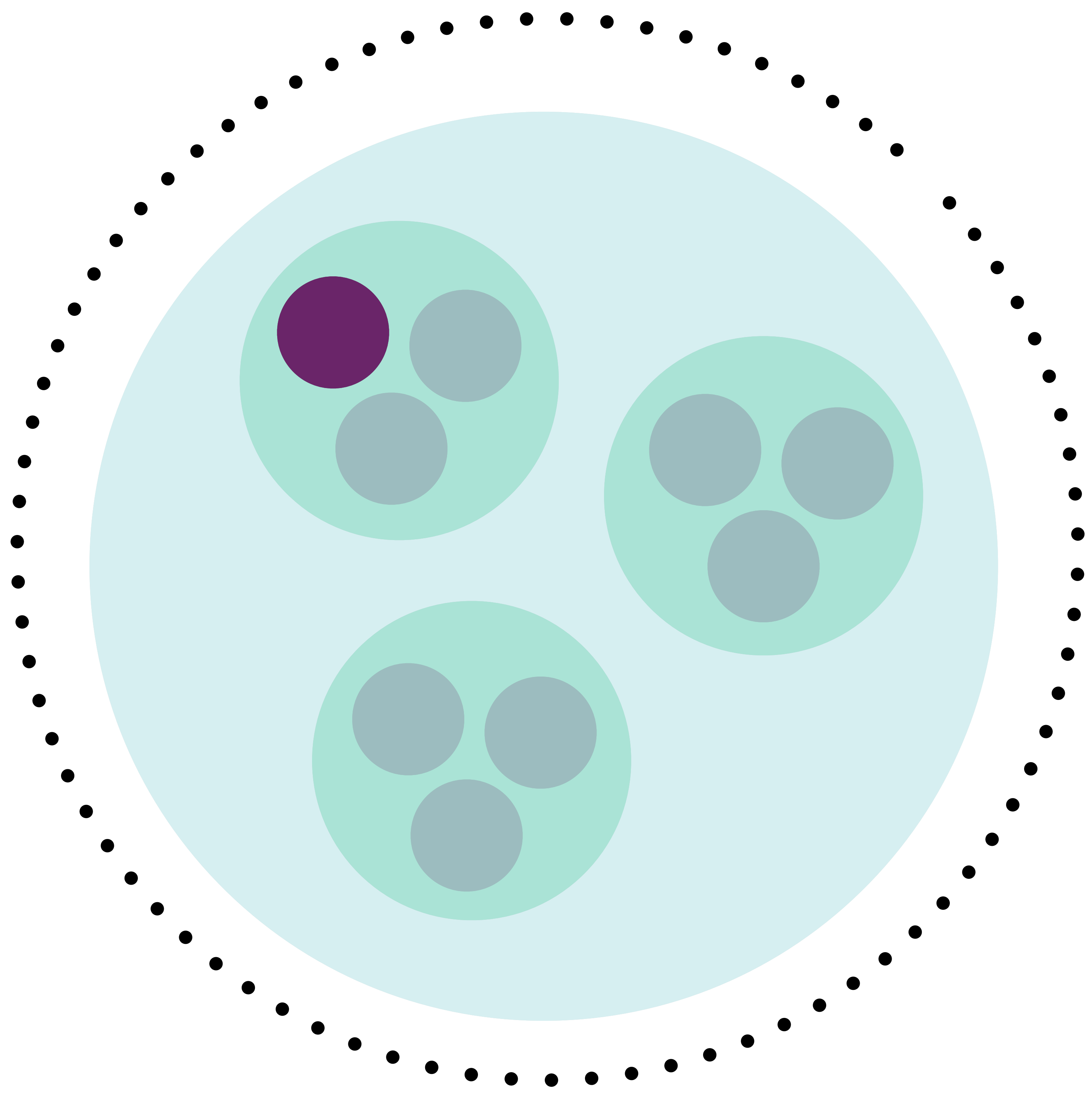
Contract Testing

Tolerant Reader

within teams

Highest rate of change

Lower stability



Conversational change

Tolerant Reader

FINAL THOUGHTS

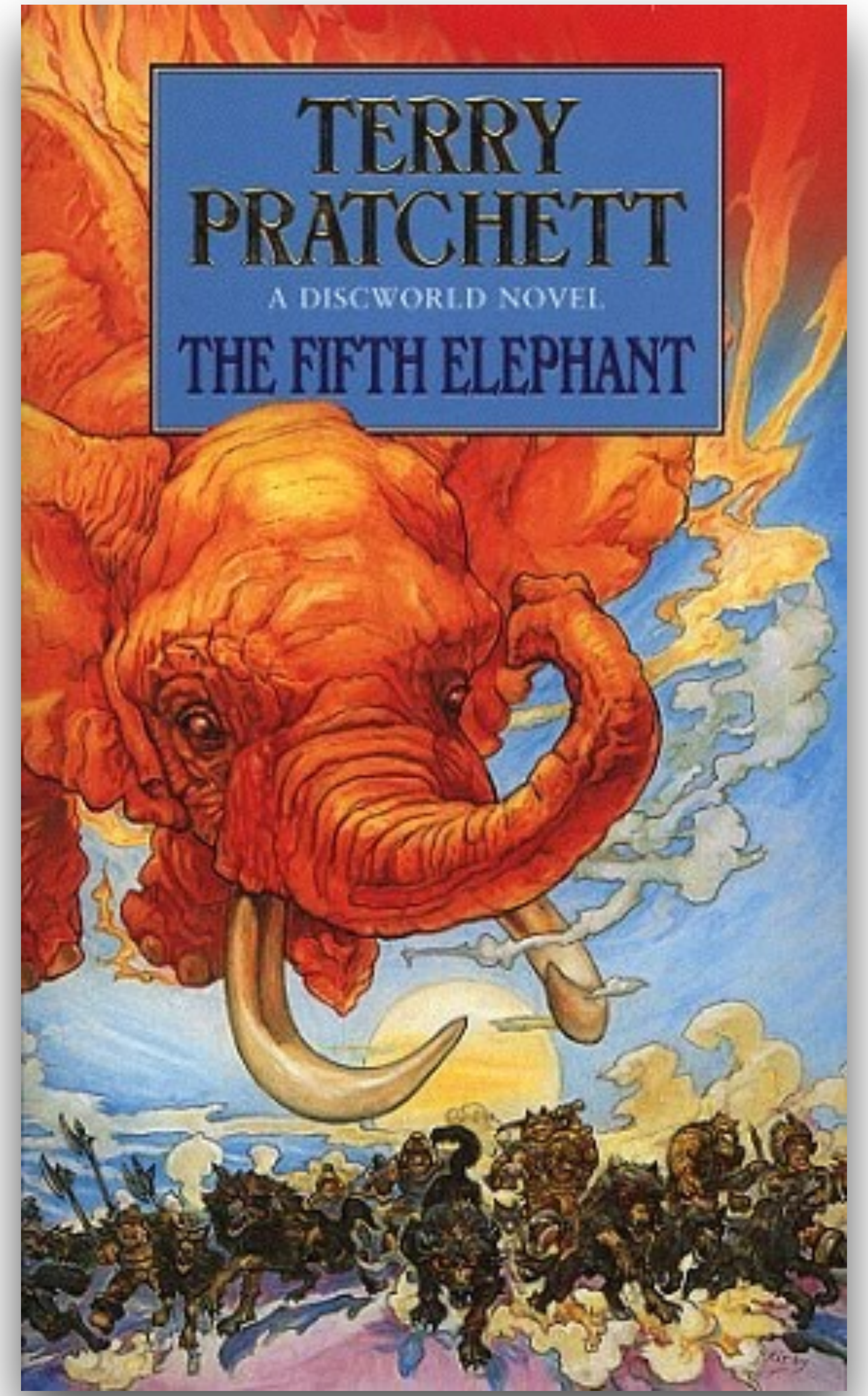
1. Rule of Modularity: Write simple parts connected by clean interfaces.
2. Rule of Clarity: Clarity is better than cleverness.
3. Rule of Composition: Design programs to be connected to other programs.
4. Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
5. Rule of 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 17 rules of UNIX programming

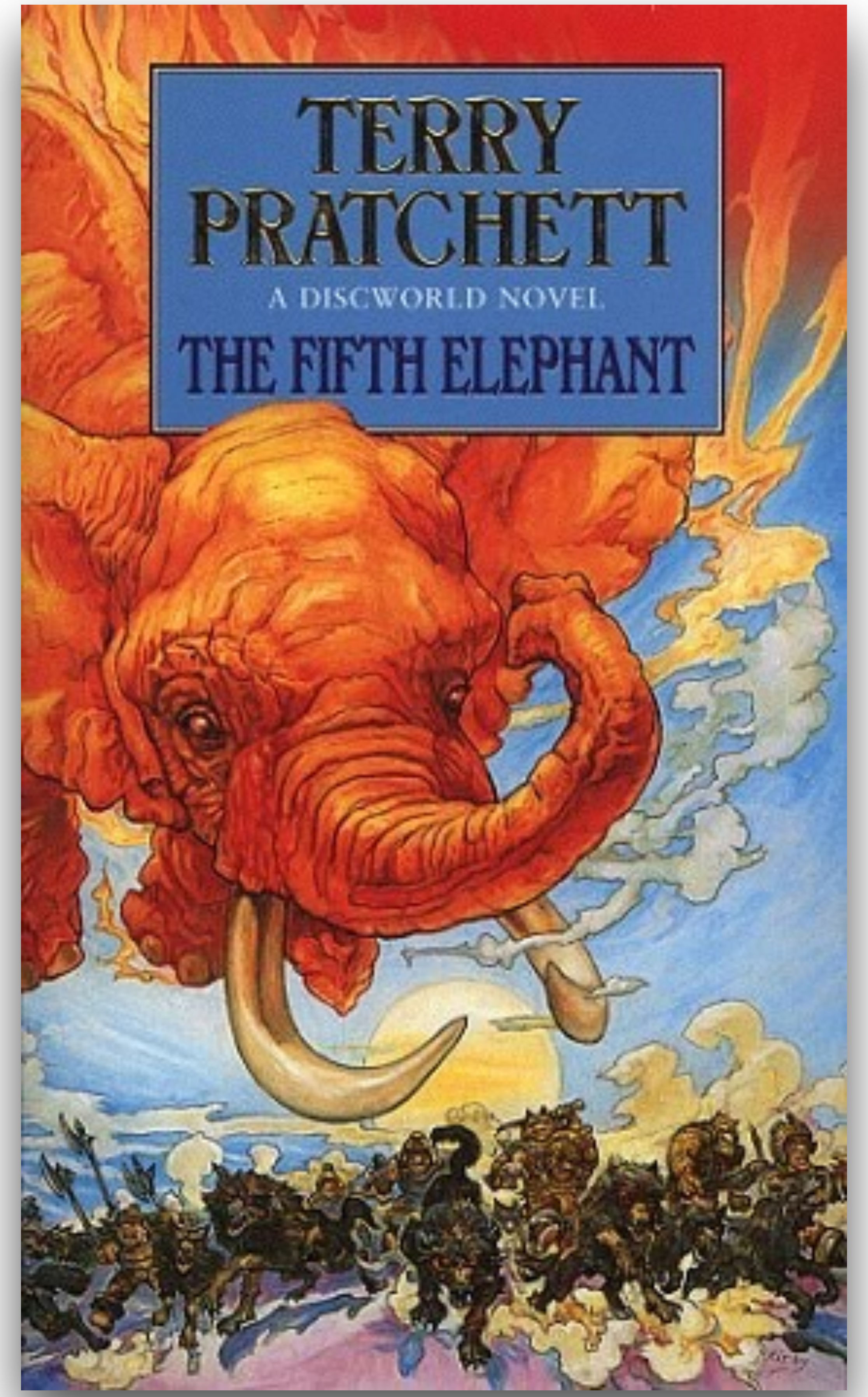
1. Rule of Modularity: Write simple parts connected by clean interfaces.
2. Rule of Clarity: Clarity is better than cleverness.
3. Rule of Composition: Design programs to be connected to other programs.
4. Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
5. Rule of 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 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
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

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 16th rule of unix programming

The Rule of Diversity

DISTRUST ALL CLAIMS FOR "ONE TRUE WAY"

Thanks!

jalewis@thoughtworks.com

@boicy

ThoughtWorks®