

# MONOLITHS TO MICROSERVICES

Sam Newman

## Introduction

# Sam Newman & Associates

O'REILLY®

# Building Microservices

DESIGNING FINE-GRAINED SYSTEMS



Sam Newman



# Course Overview

# Course Overview

**Day One**

# Course Overview

## **Day One**

What & Why Of Microservices

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

## **Day Two**

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

## **Day Two**

Migrating a running system

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

## **Day Two**

Migrating a running system

Build & Deployment

# Course Overview

## **Day One**

What & Why Of Microservices

Planning A Transition

Splitting stateless and stateful services

## **Day Two**

Migrating a running system

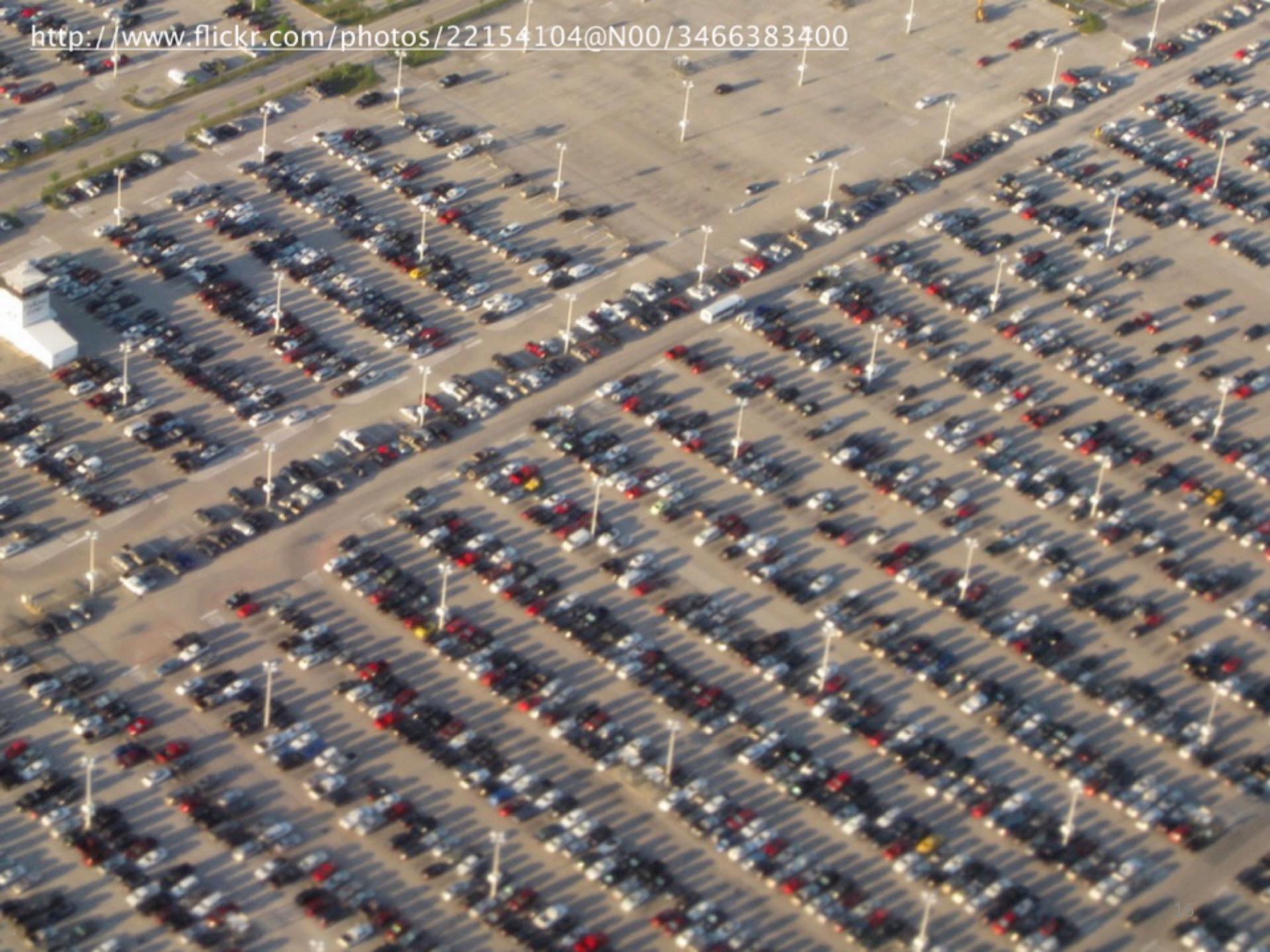
Build & Deployment

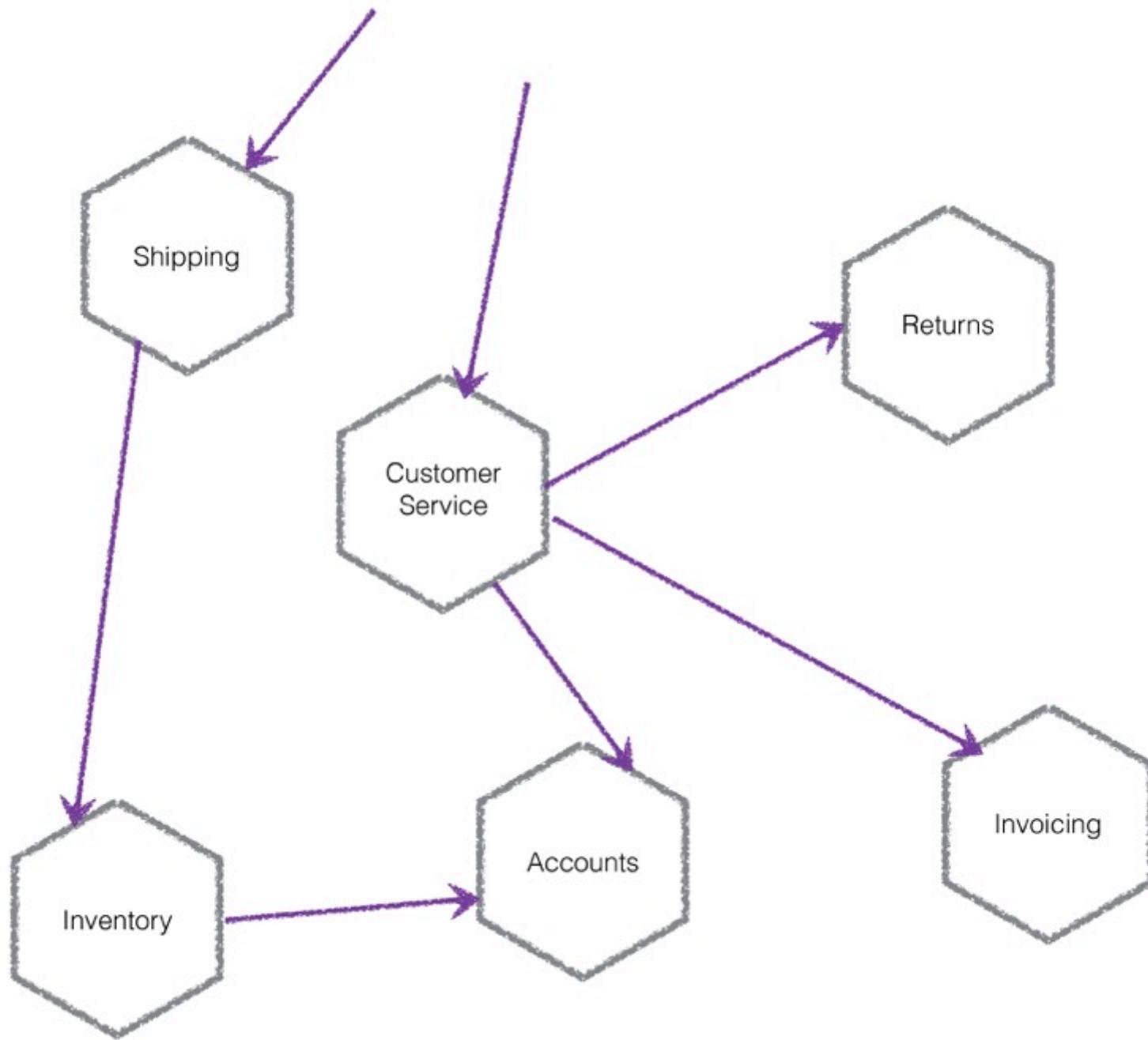
Monitoring



<http://www.flickr.com/photos/55255903@N07/6835060992>

<http://www.flickr.com/photos/22154104@N00/3466383400>



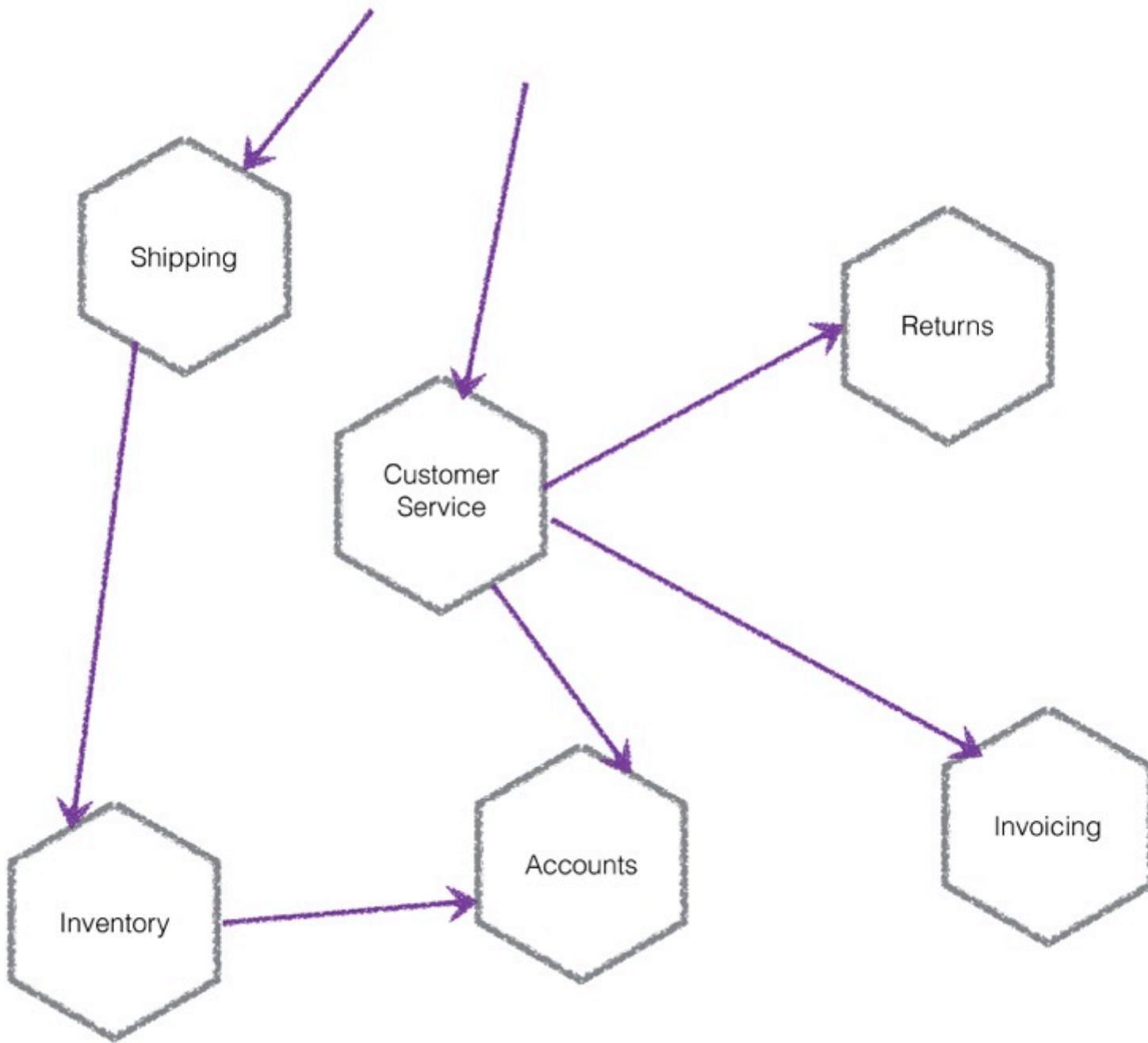


Small ***Independently Deployable*** services that ***work together***, modelled around a ***business domain***

Small?

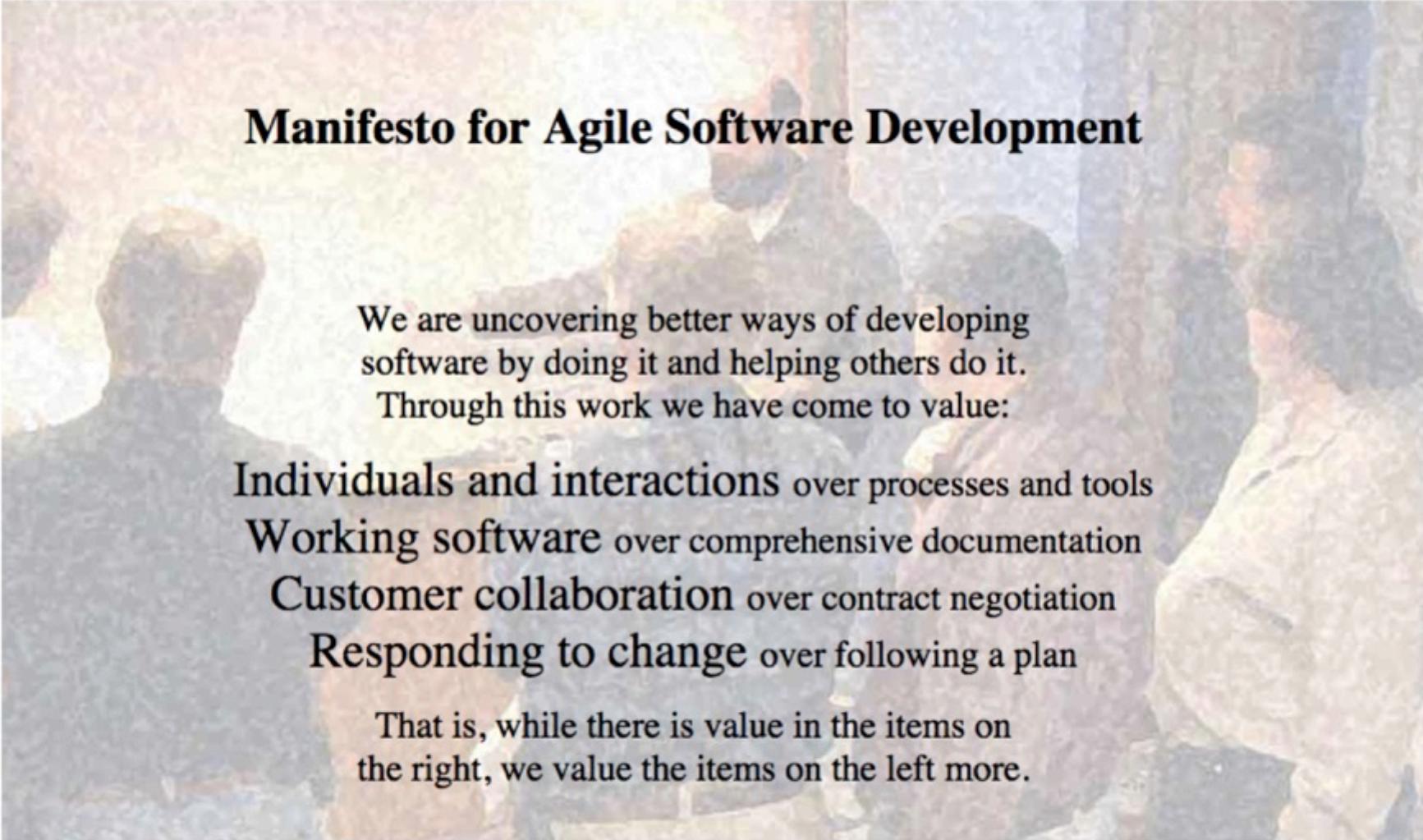
# Independently Deployable?

# Business Domain?



# Technology?

And SOA?



## **Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

**Individuals and interactions** over processes and tools

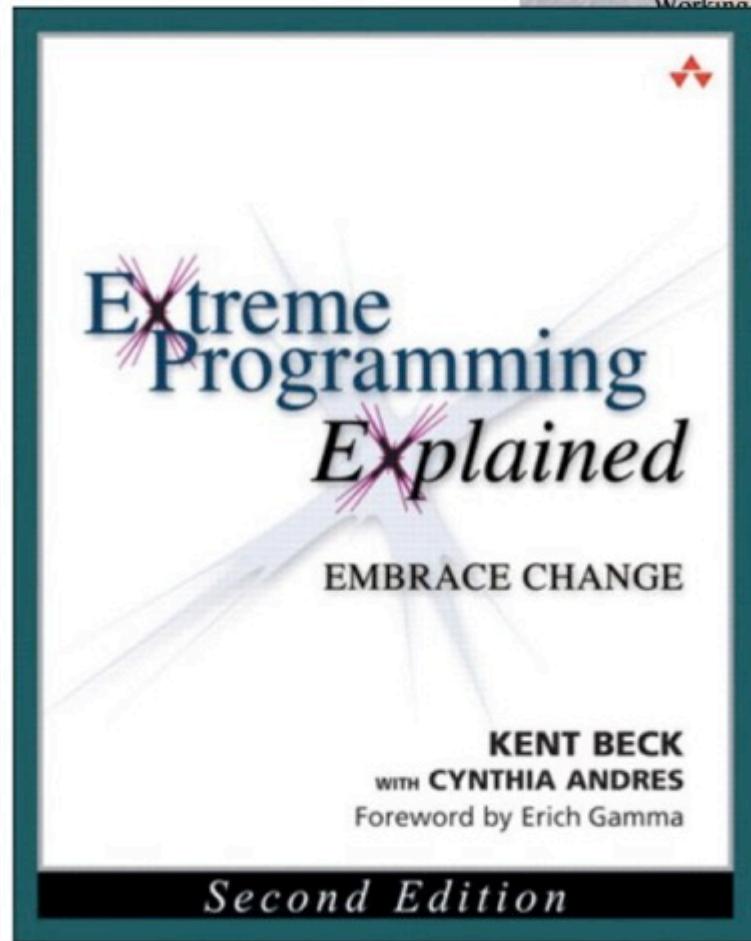
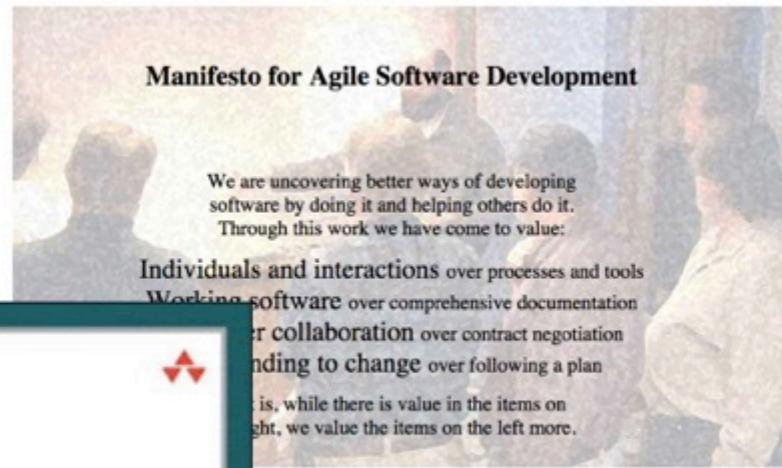
**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



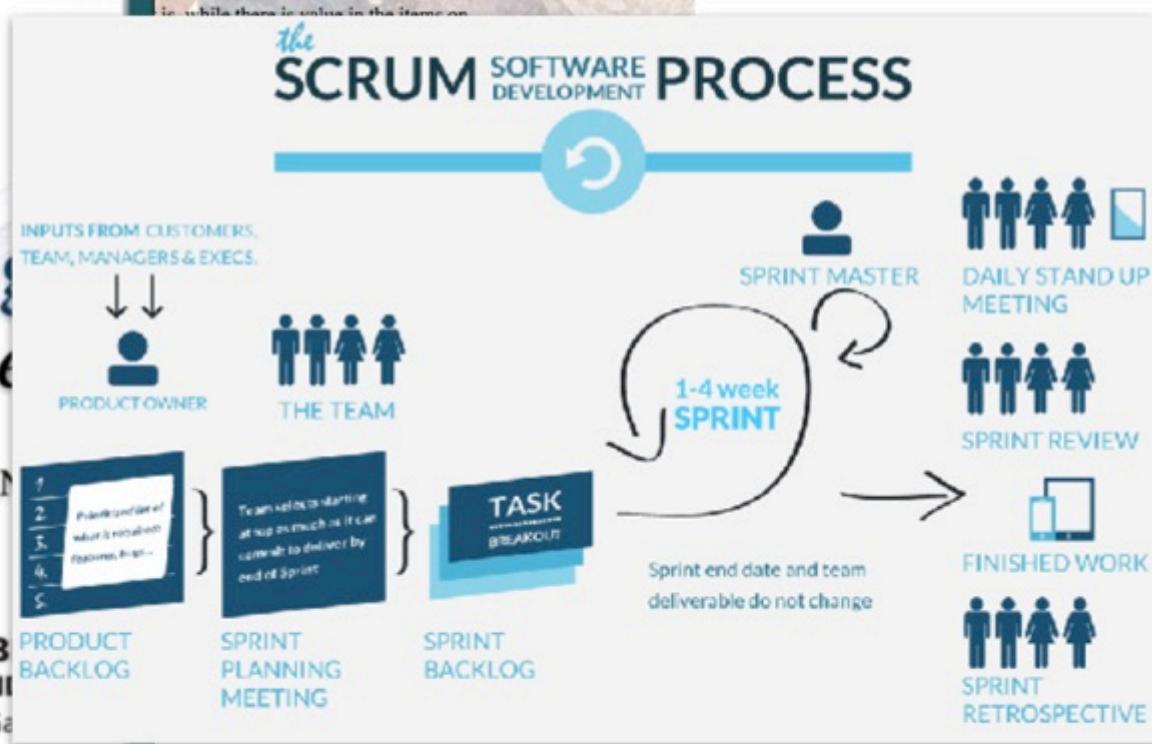


# ~~Extreme~~ Programming Explained

EMBRACE CHANGE

KENT B.  
with CYNTHIA ANI  
Foreword by Erich Gamma

*Second Edition*



# Why Use Them?

# **Autonomy**

# **Autonomy**

Go Faster

# **Autonomy**

Go Faster

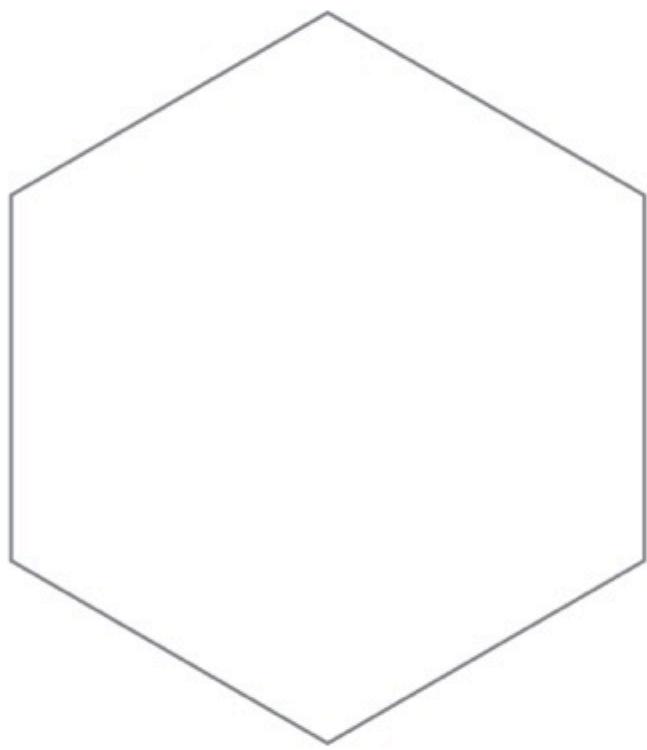
Scale Teams

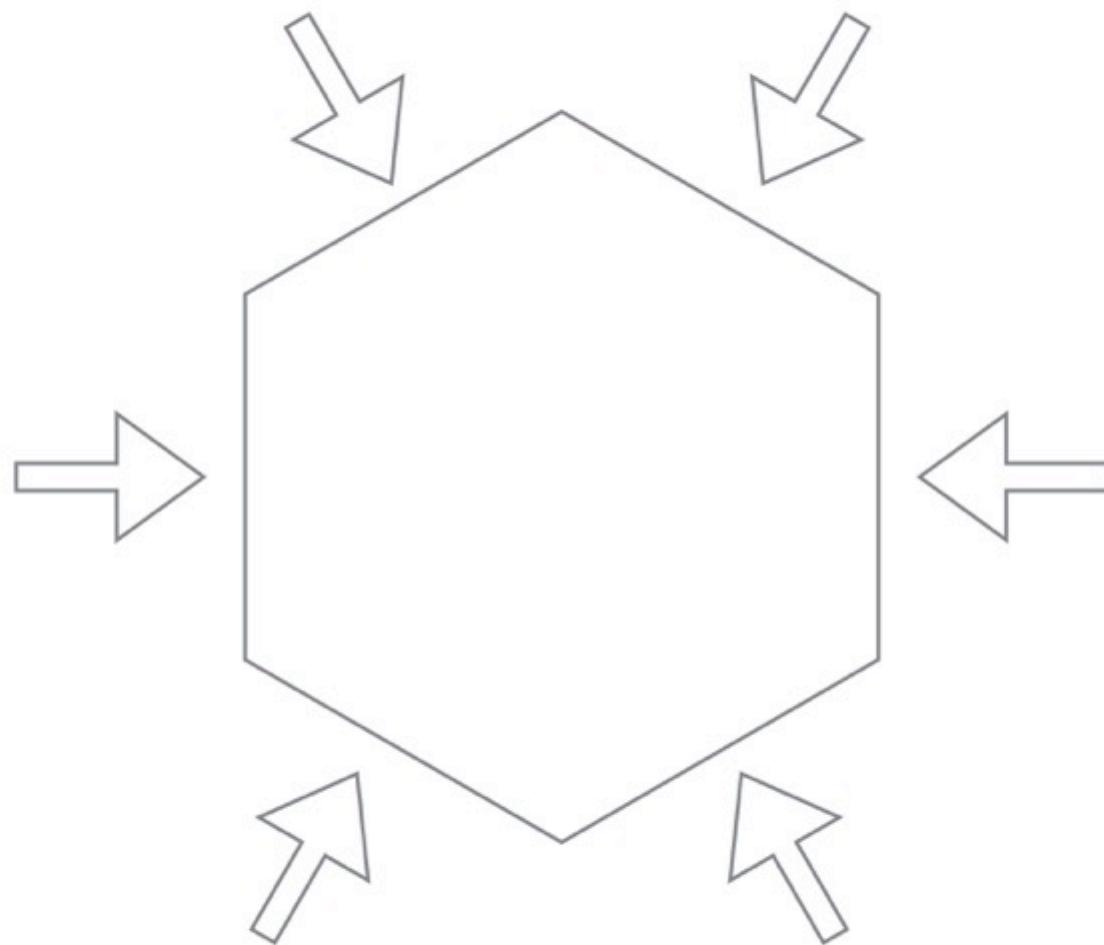
# **Autonomy**

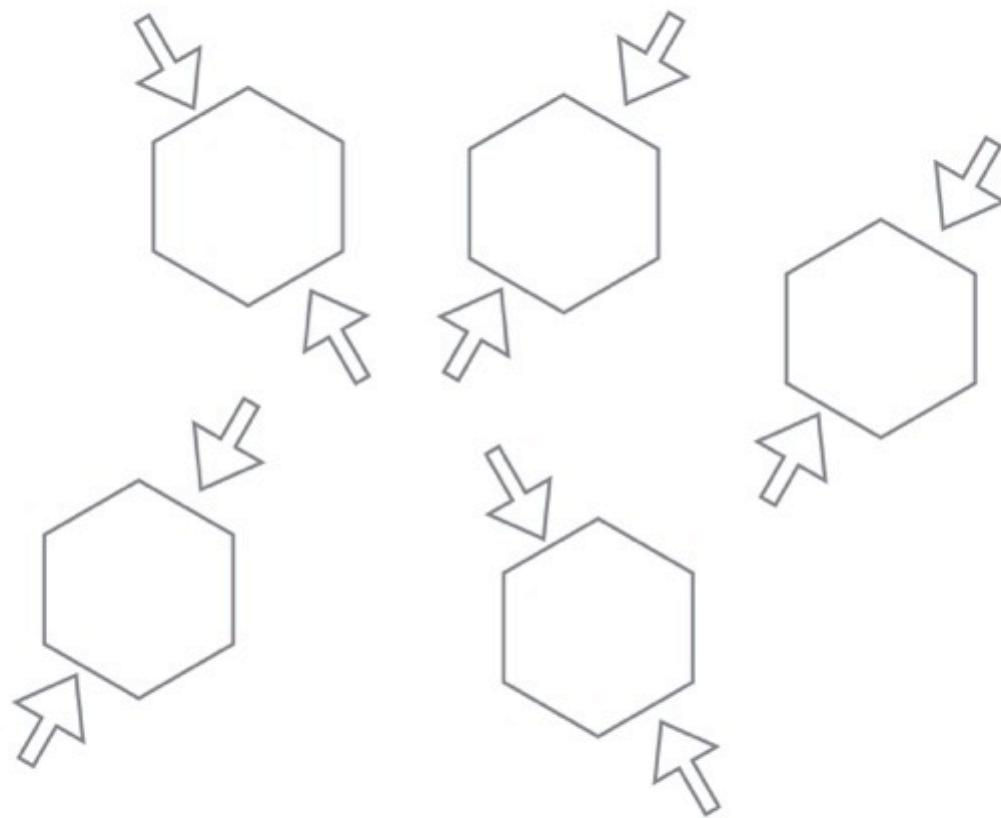
Go Faster

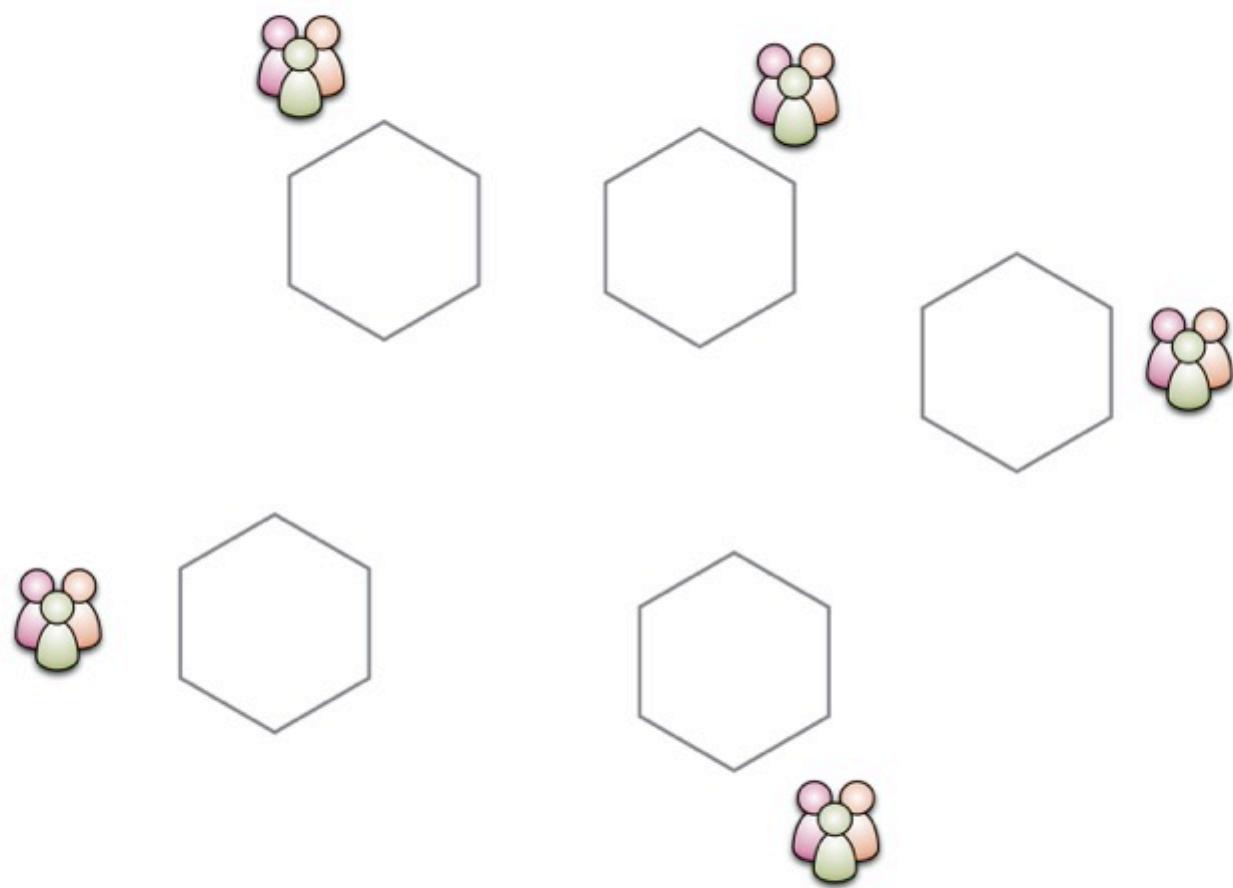
Scale Teams

Empowerment









# **And Perhaps...**

# **And Perhaps...**

New Technology

# **And Perhaps...**

New Technology

Security

# **And Perhaps...**

New Technology

Security

Scale Application

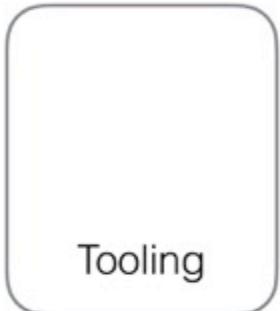
# What Are The Problems?

## COGNITIVE OVERLOAD

Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

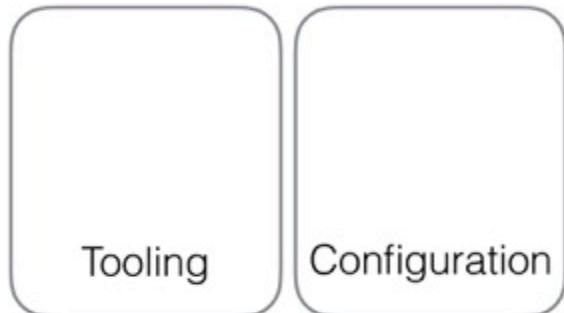
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

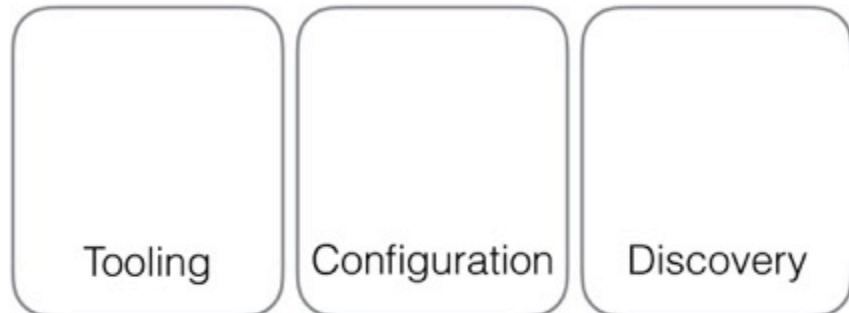
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

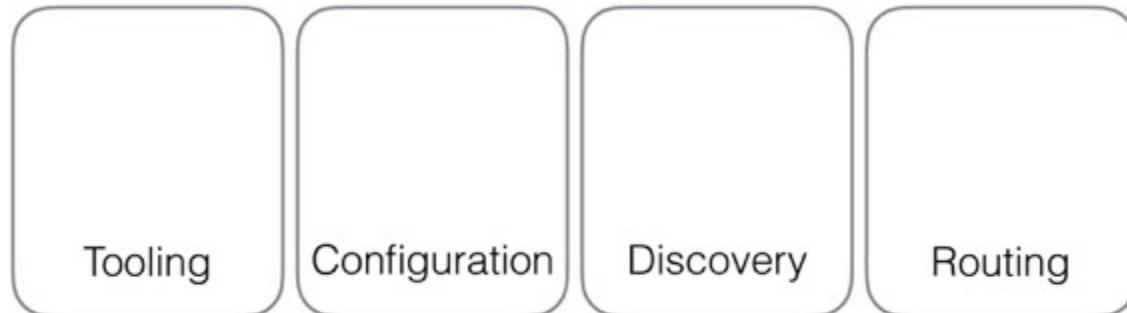
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

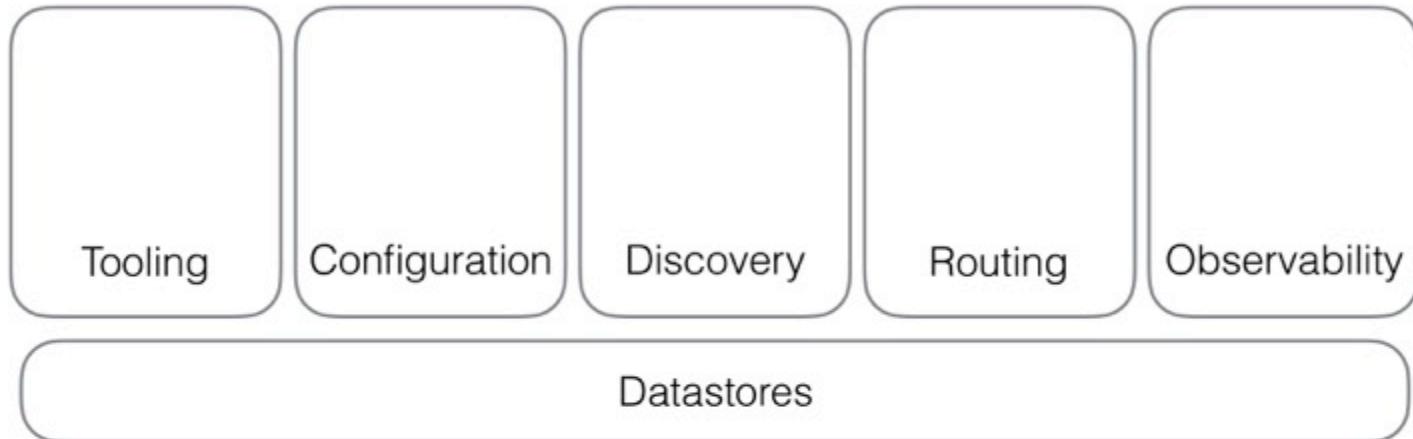
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

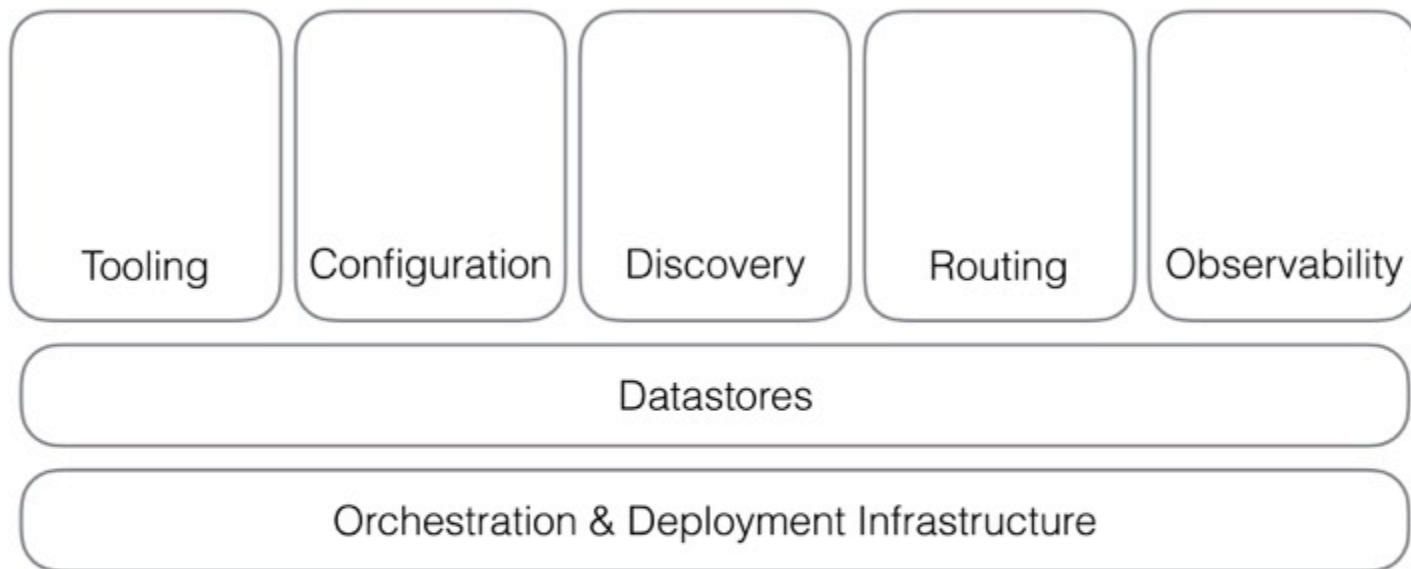
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

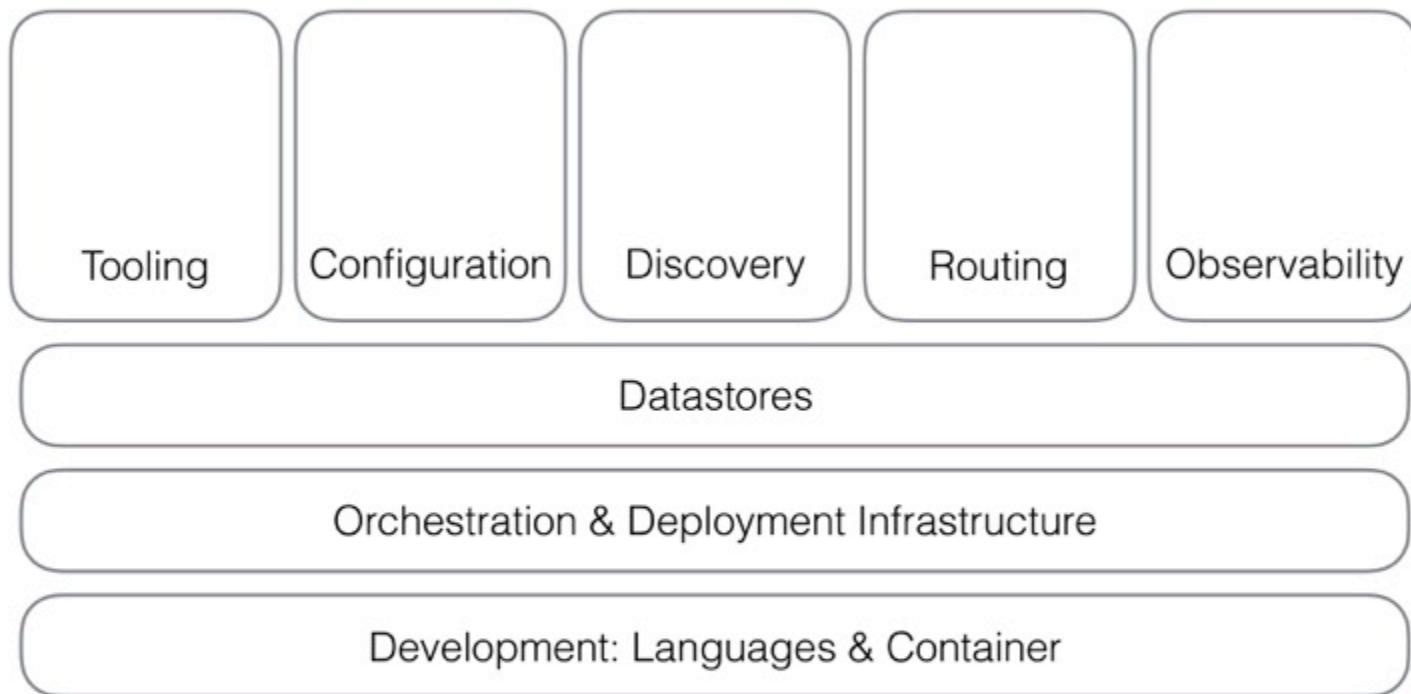
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

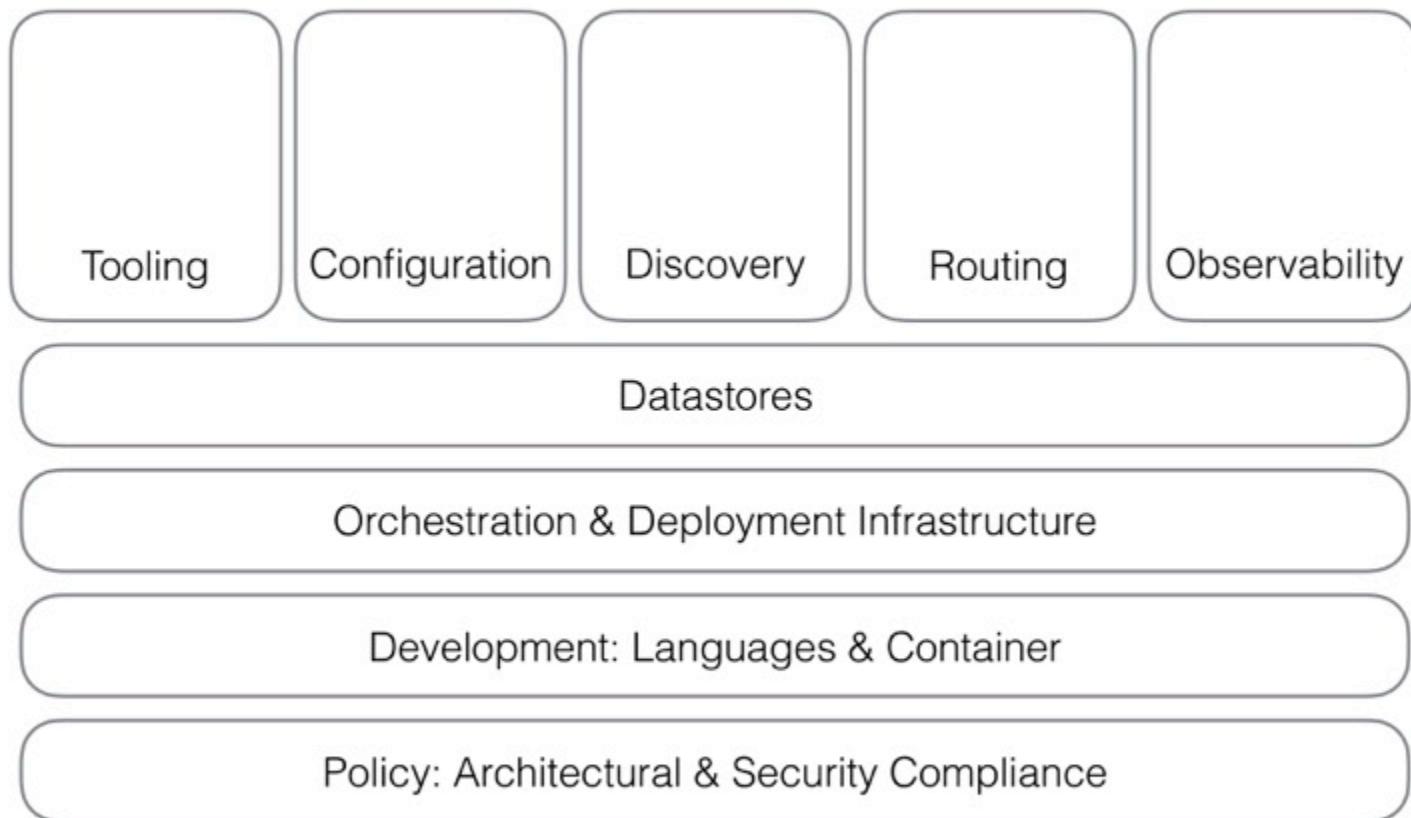
## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

## COGNITIVE OVERLOAD



Based on a model from Adrian Cockcroft:

<http://www.slideshare.net/adriancockcroft/evolution-of-microservices-craft-conference>

# **Distributed Systems**

# **Distributed Systems**

Transactions

# **Distributed Systems**

Transactions

CAP!

# Distributed Systems

Transactions

CAP!

Monitoring

# Distributed Systems

Transactions

CAP!

Monitoring

Testing

# Investment In New Tooling

# Co-ordination

Knowing where to start

What does good look like

How to manage a transition

# MONOLITHS TO MICROSERVICES

Sam Newman

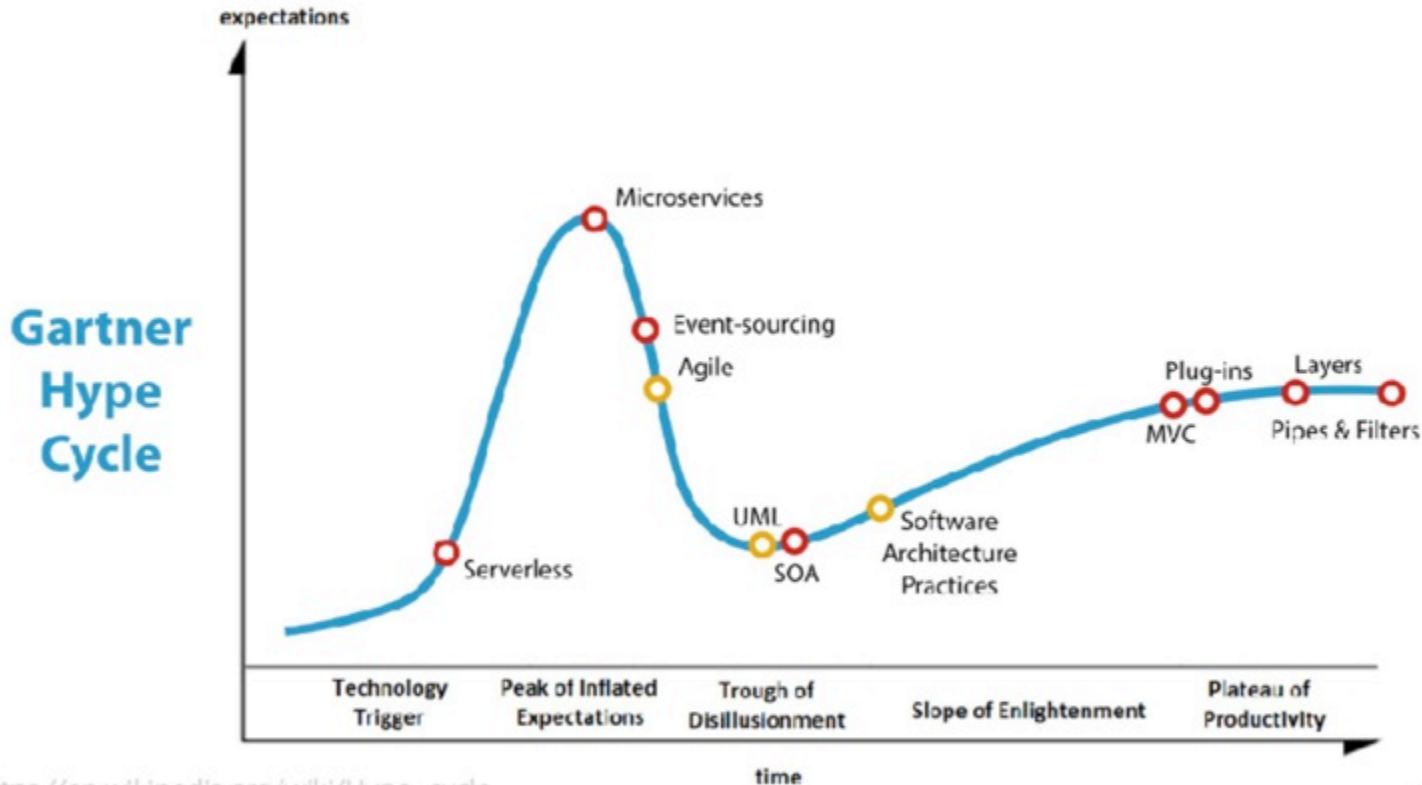
## Planning a Transition

How

# Why?

Because everyone else is!

## HYPE!



Hat tip @robsmallshire <https://twitter.com/robsmallshire/status/787936059581693952>



<https://www.flickr.com/photos/imprint777/6329796960/>

# Pros & Cons



# Can cannabis relieve pain and other ailments?

MPs have recommended medicinal marijuana for the relief of chronic pain and anxiety, but the law is against it



There is now good evidence for the medicinal use of cannabis. Photograph: Rick Wilking / Reuters/Reuters

Queen Victoria was prescribed cannabis for period pains, and 19th-century American doctors used it for everything from anorexia to sexual problems. And now the US is embracing medicinal cannabis again - it's

## Can cannabis relieve pain and other ailments?

MPs have recommended medicinal marijuana for the relief of chronic pain and anxiety, but the law is against it



*“Typical side-effects include drowsiness, nausea, paranoia, sweating and euphoria.”*



There is now good evidence for the medicinal use of cannabis. Photograph: Rick Wilking / Reuters/Reuters

Queen Victoria was prescribed cannabis for period pains, and 19th-century American doctors used it for everything from anorexia to sexual problems. And now the US is embracing medicinal cannabis again - it's

## Can cannabis relieve pain and other ailments?

MPs have recommended medicinal marijuana for the relief of chronic pain and anxiety, but the law is against it



*“Typical side-effects include drowsiness, nausea, paranoia, sweating and euphoria.”*



There is now good evidence for the medicinal use of cannabis. Photograph: Rick Wilking / Reuters/Reuters

Queen Victoria was prescribed cannabis for period pains, and 19th-century American doctors used it for everything from anorexia to sexual problems. And now the US is embracing medicinal cannabis again - it's



Organisational Alignment

Many Choices to Make

Release Functionality Faster

Have A Lead Time

Independent Scaling

Testing Is Harder

Easier to focus on security concerns

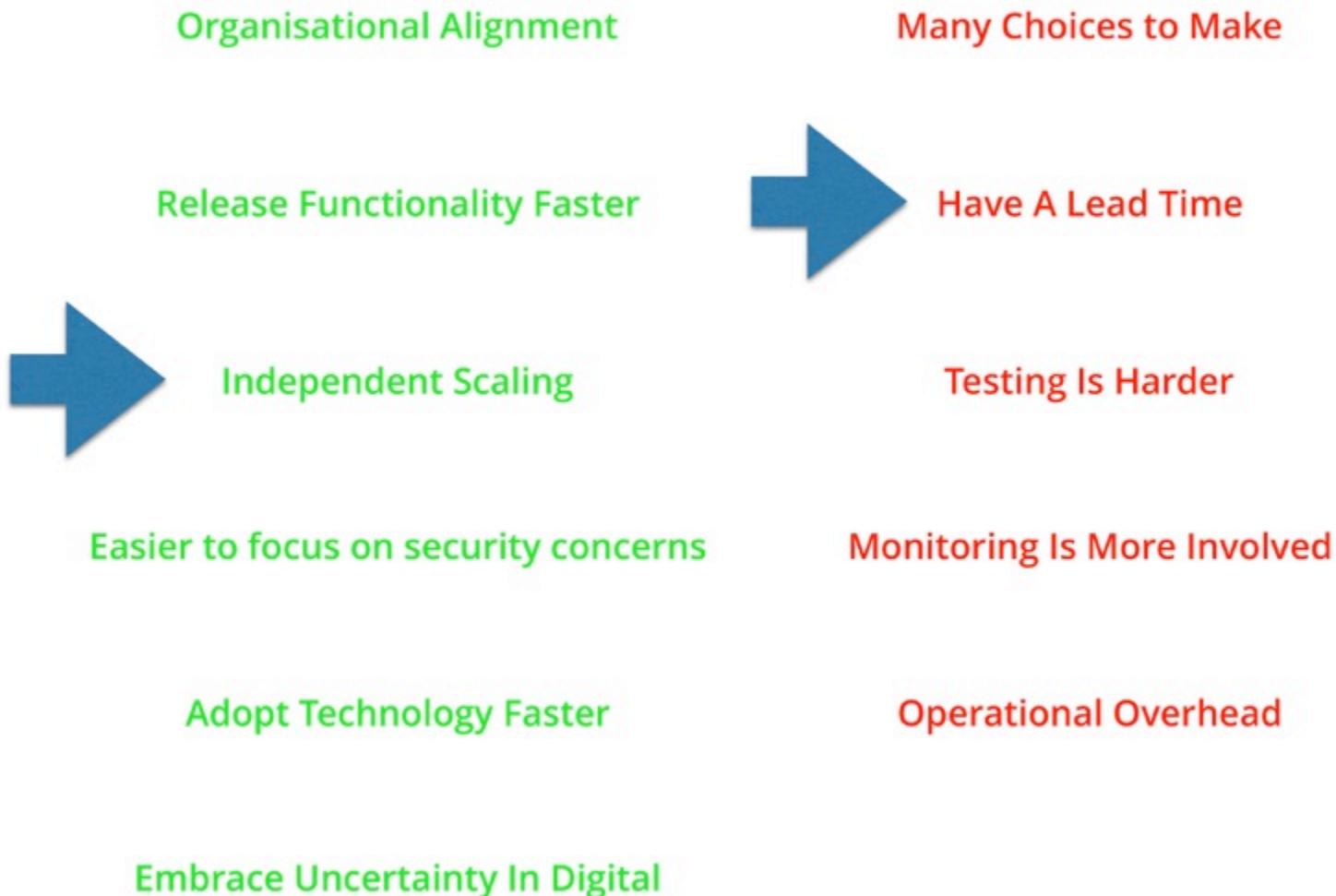
Monitoring Is More Involved

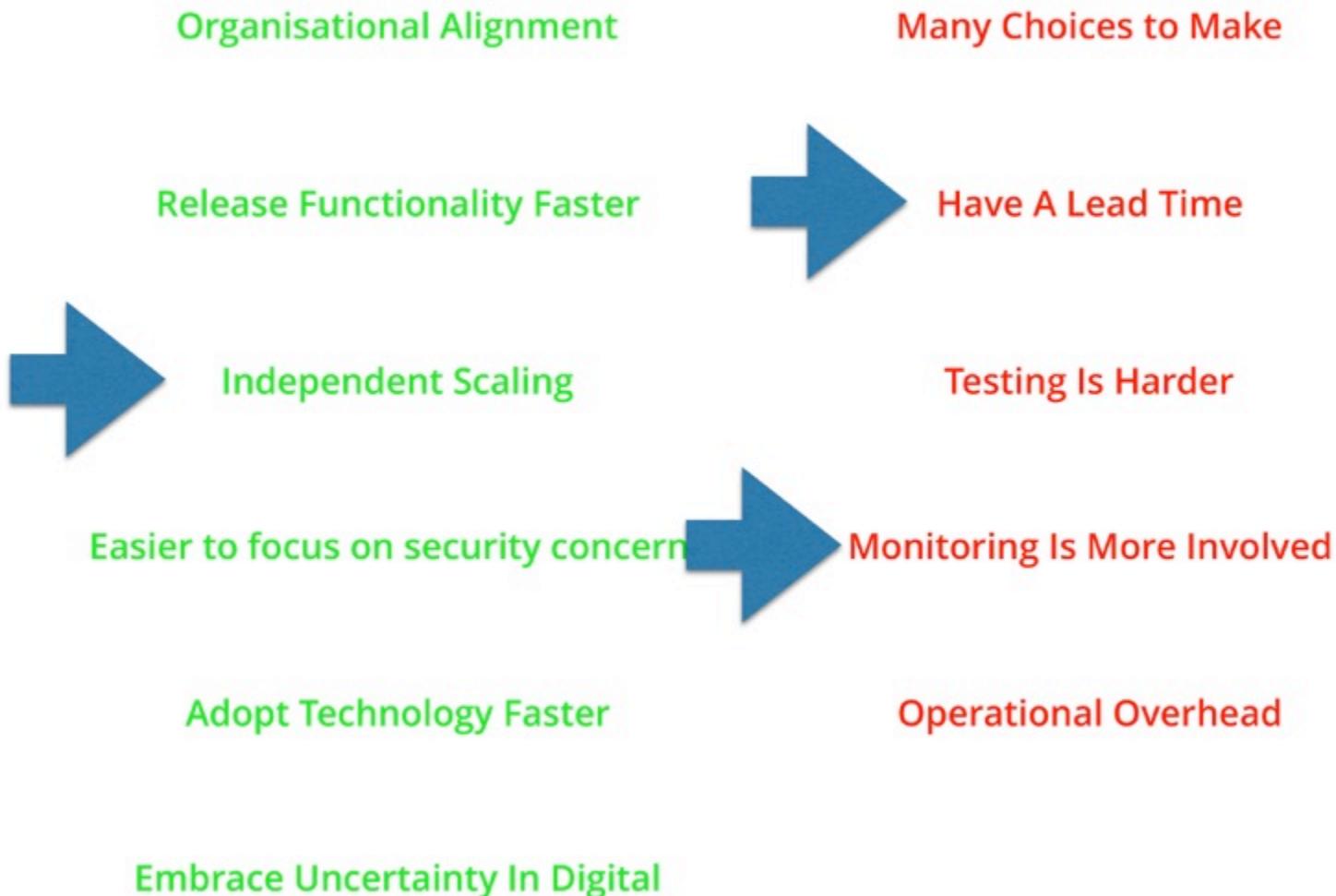
Adopt Technology Faster

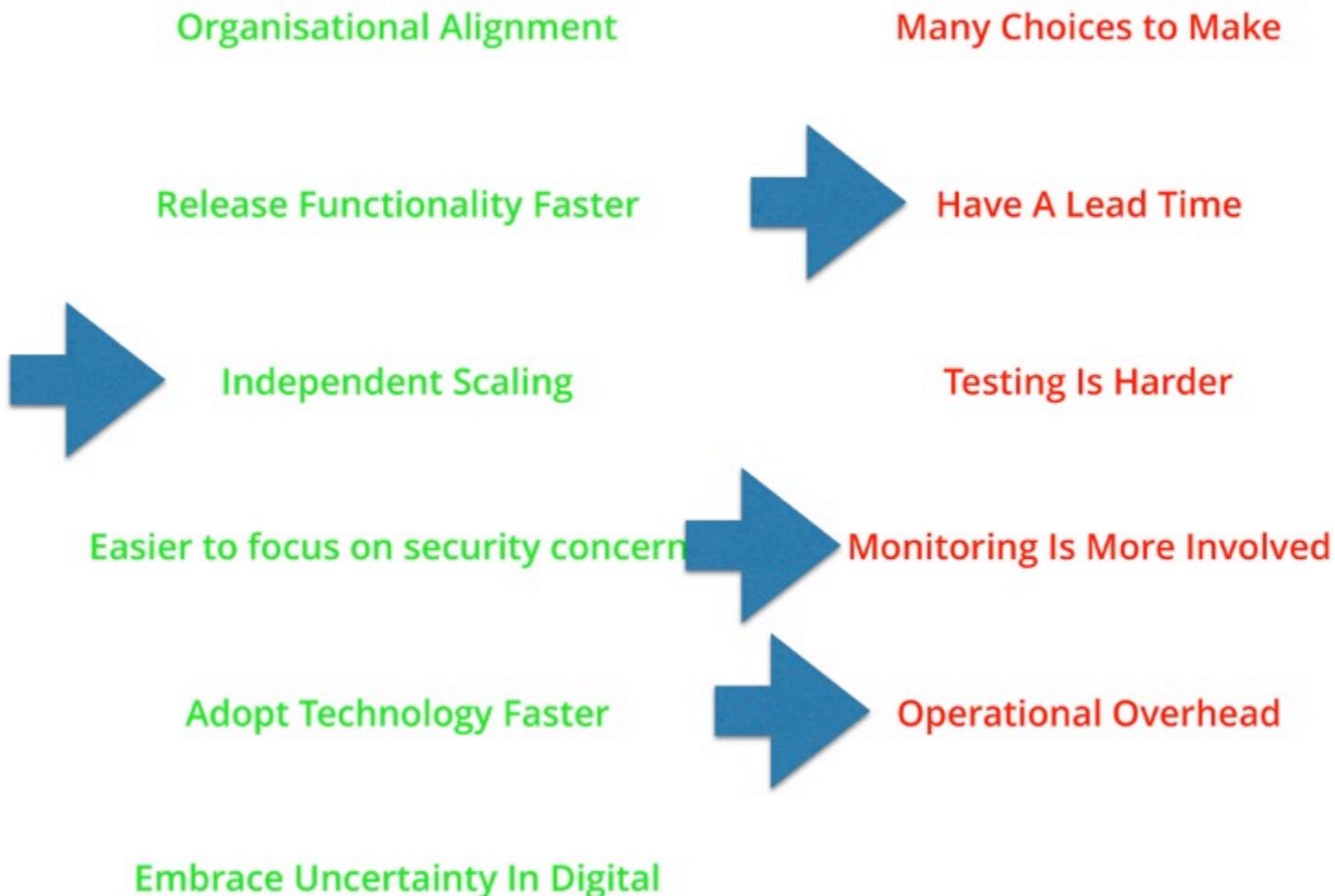
Operational Overhead

Embrace Uncertainty In Digital









# Know What You're After

Know What You're After

Know When To Stop

Know What You're After

Know When To Stop

And Know If There Is A  
Better Way

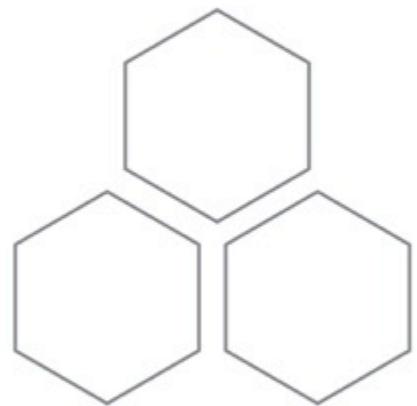


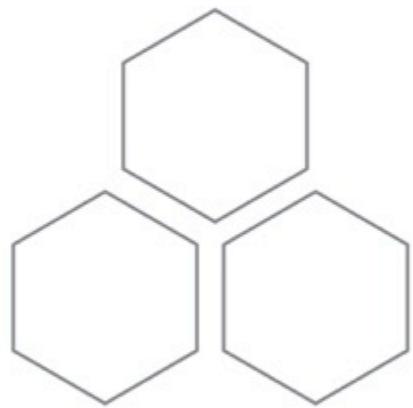


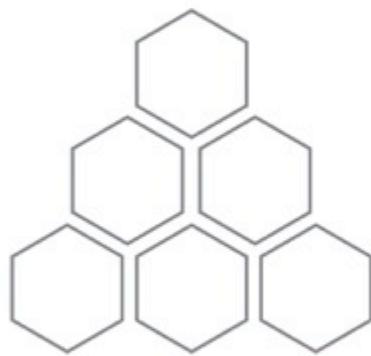
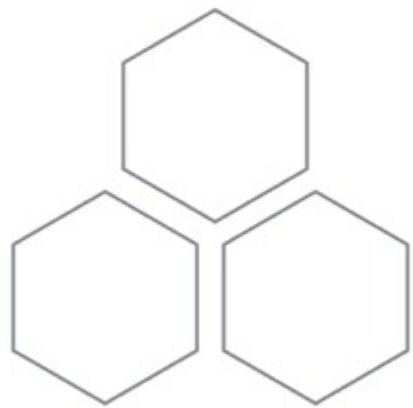
<https://www.flickr.com/photos/antonymayfield/8463026326/>

[https://www.flickr.com/photos/rafa\\_luque/20445309971/](https://www.flickr.com/photos/rafa_luque/20445309971/)







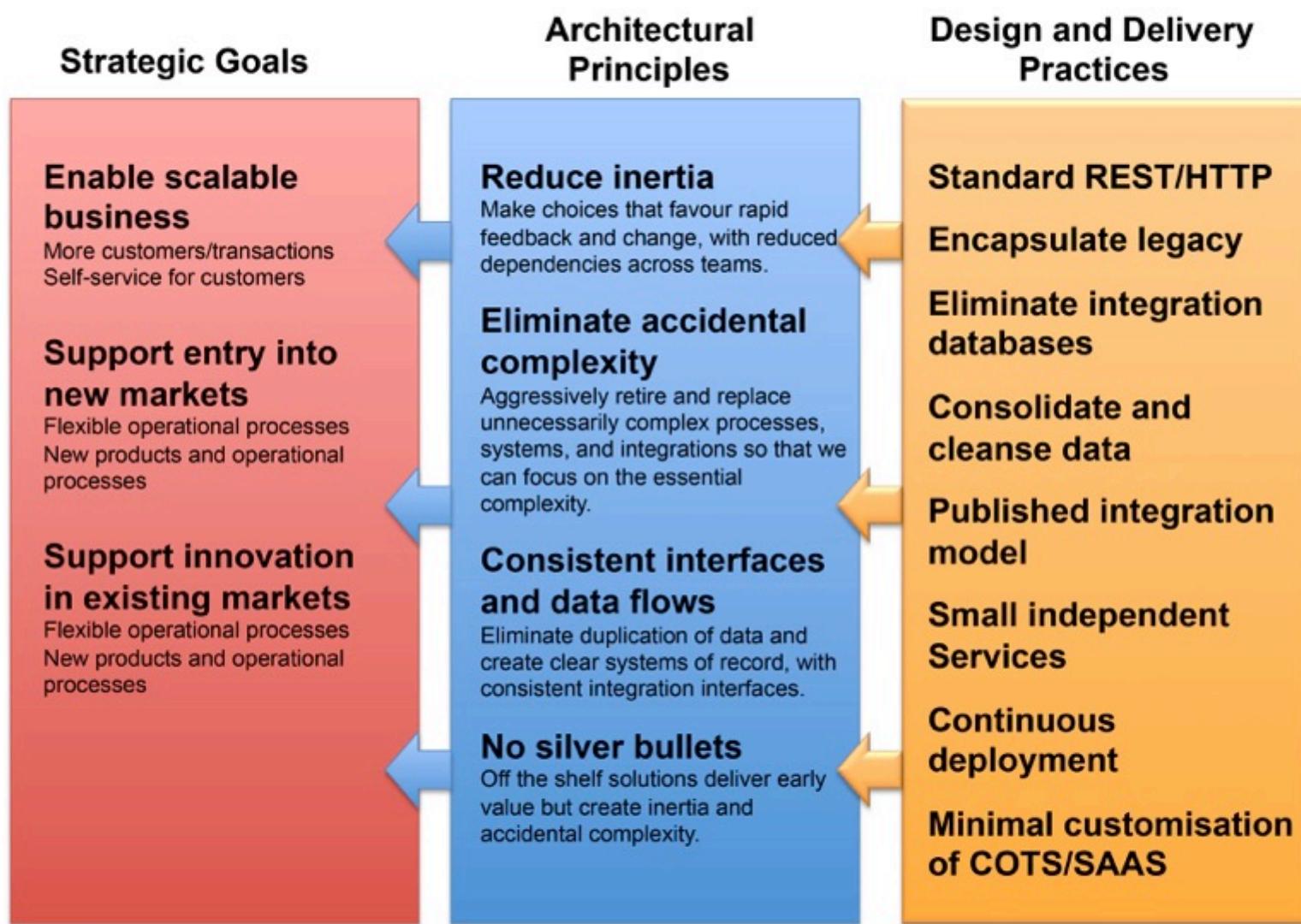


Microservices are not an end in and of themselves

Need to know why you are  
using them

Need to know why you are  
using them

If they are working



So what reasons  
might you pick?

# **2 Examples**

# **2 Examples**

Improved Autonomy

# 2 Examples

Improved Autonomy

Scale Your Application

# Improved Autonomy

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Meetings!

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Meetings!

## **How Else Could We Do It?**

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Meetings!

## **How Else Could We Do It?**

Team-provisioned Infrastructure

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Meetings!

## **How Else Could We Do It?**

Team-provisioned Infrastructure

More clear lines of ownership

# Improved Autonomy

Improve the freedom we give people to do the job at hand, reducing the amount of sign off and co-ordination needed with others...

...so that we can come up with new ideas and ship them to our customers faster

## **What Could We Track?**

# Of team-actioned deployments

Failure Rate

# Meetings!

## **How Else Could We Do It?**

Team-provisioned Infrastructure

More clear lines of ownership

Better testing

# Scale Your Application

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

## **How Else Could We Do It?**

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

## **How Else Could We Do It?**

Vertical Scaling

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

## **How Else Could We Do It?**

Vertical Scaling

Horizontal scaling

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

## **How Else Could We Do It?**

Vertical Scaling

Horizontal scaling

Data Partitioning

# Scale Your Application

Functionally decompose your monolith to allow latency/load sensitive parts of your application to be scaled independently

## **What Could We Track?**

Latency

Load

Failure Rate

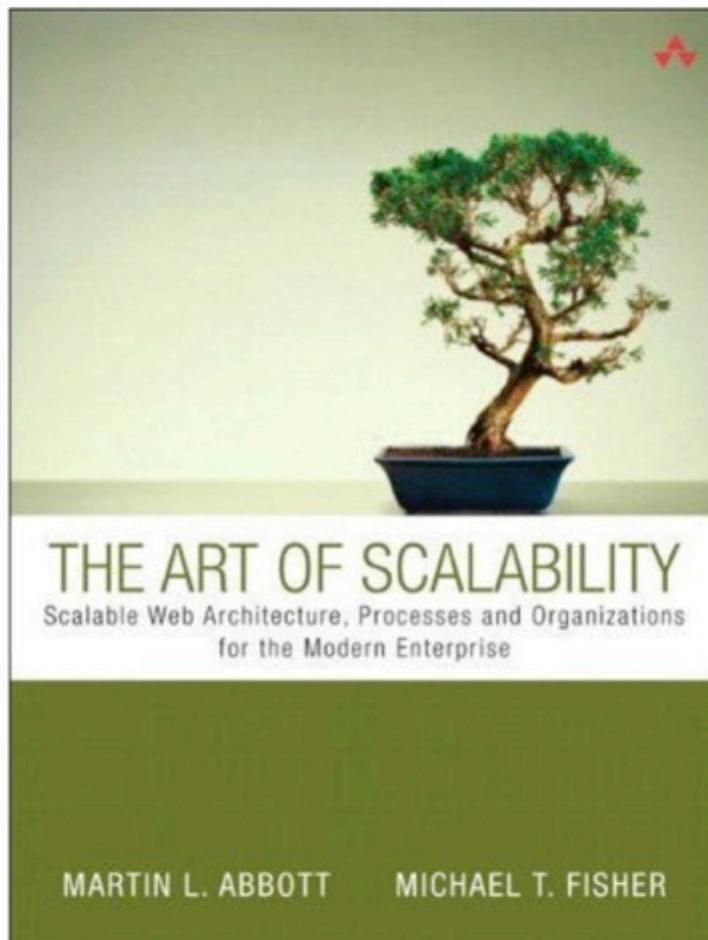
## **How Else Could We Do It?**

Vertical Scaling

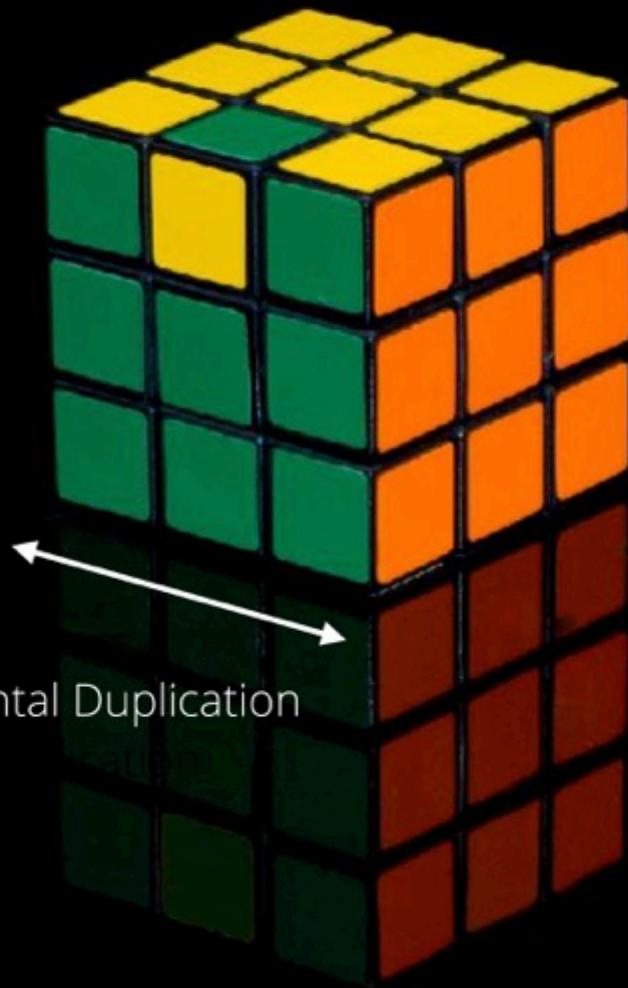
Horizontal scaling

Data Partitioning

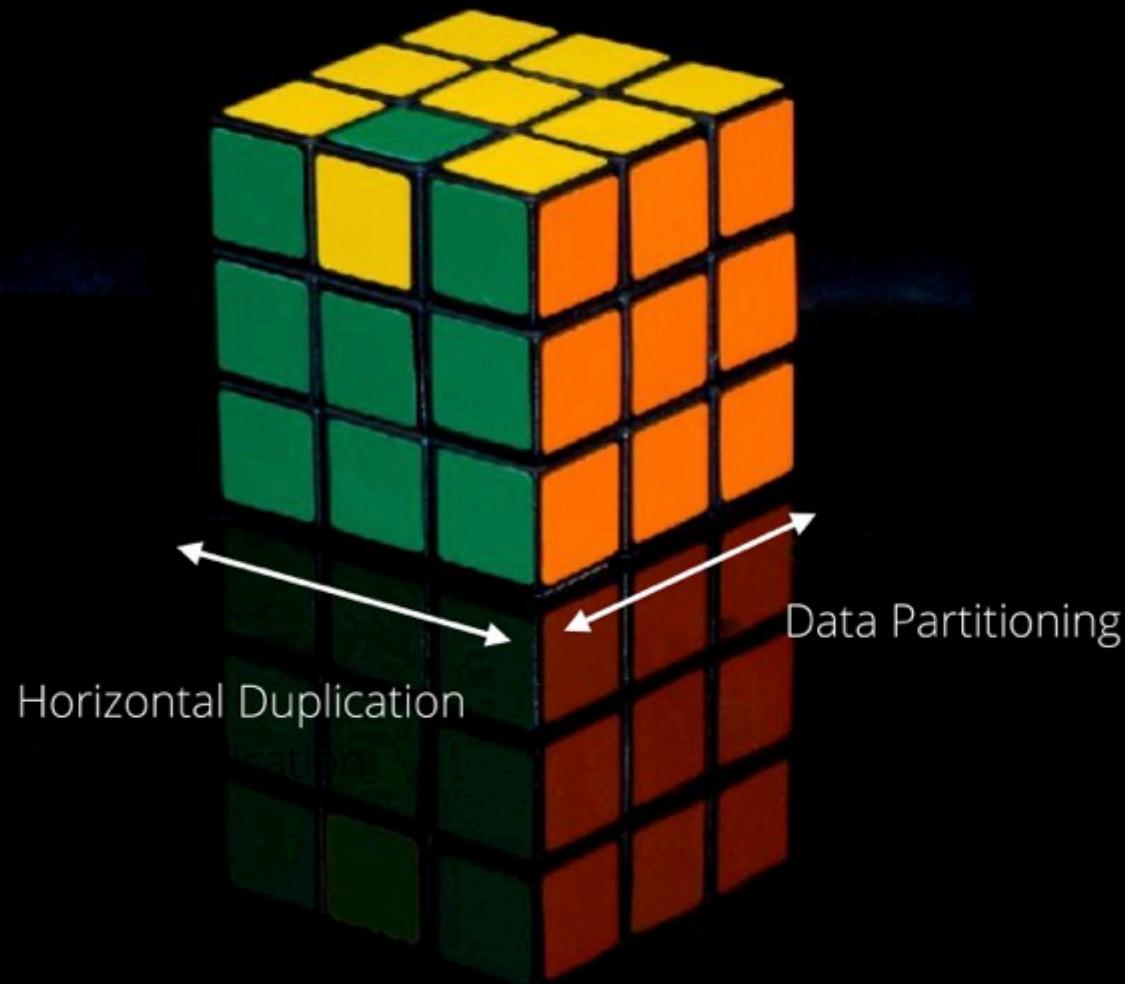
Reverse Proxies...

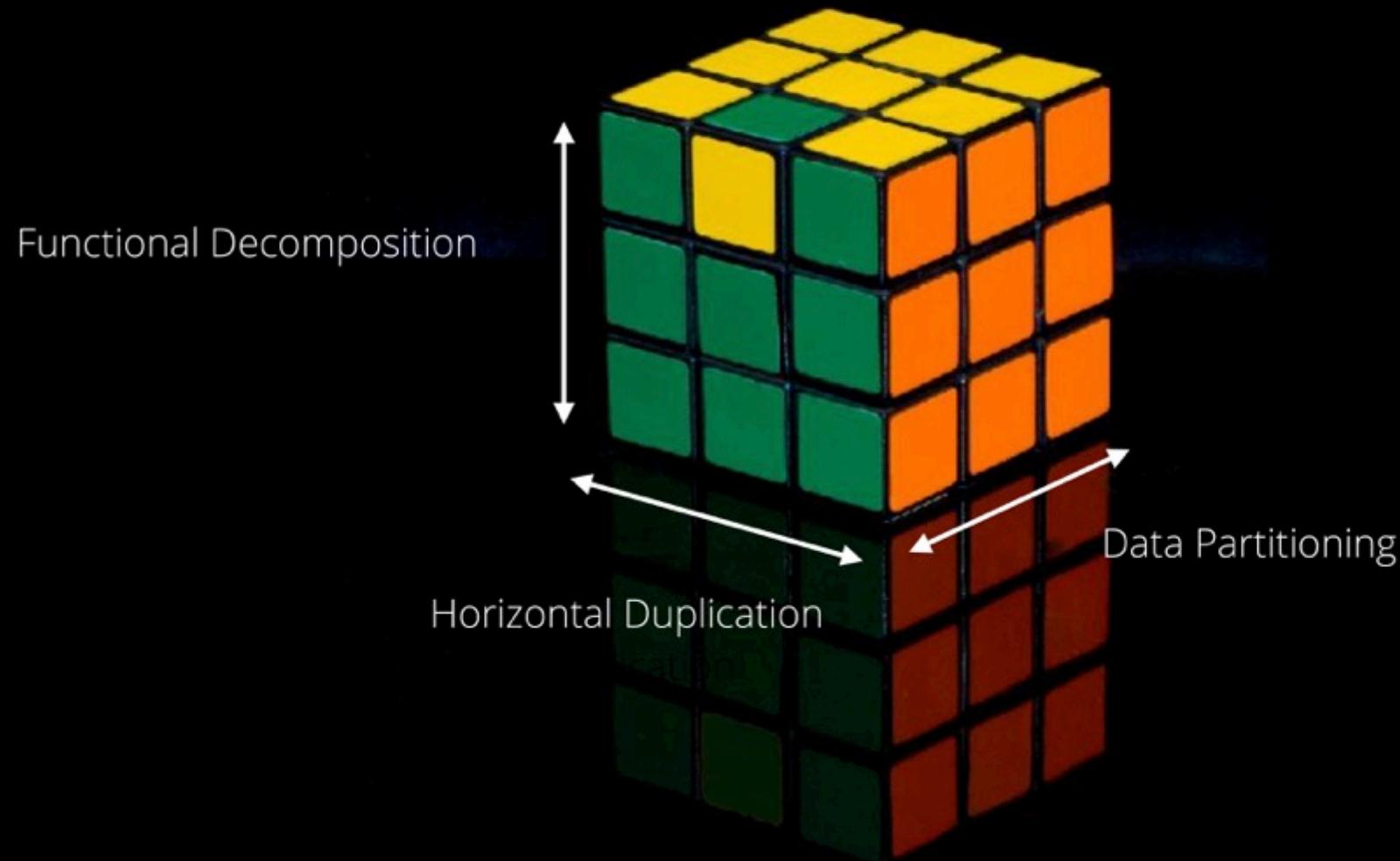






Horizontal Duplication







**astrid atkinson**

@shinynew\_oz



Follow

@bridgetkromhout software is made of feelings.

RETWEETS

3

LIKES

3



9:45 PM - 20 Feb 2015



3

3

...

[https://twitter.com/shinynew\\_oz/status/568889104273838080](https://twitter.com/shinynew_oz/status/568889104273838080)

Data isn't more important  
than feelings

# Have Periodic Checkpoints

# Have Periodic Checkpoints

- Restate the goal you're trying to achieve

# Have Periodic Checkpoints

- Restate the goal you're trying to achieve
- Reflect on what the data is showing you...

# Have Periodic Checkpoints

- Restate the goal you're trying to achieve
- Reflect on what the data is showing you...
- And how the team thinks about it

# Have Periodic Checkpoints

- Restate the goal you're trying to achieve
- Reflect on what the data is showing you...
- And how the team thinks about it
- Decide what, if anything, to change

How often should you have checkpoints?

# How often should you have checkpoints?

- Depends on scale of company/change

# How often should you have checkpoints?

- Depends on scale of company/change
  - For a single team, perhaps reflecting informally every week is good enough

# How often should you have checkpoints?

- Depends on scale of company/change
  - For a single team, perhaps reflecting informally every week is good enough
  - For larger change programs, you might want local review followed by some overall governance process

What about your  
customers?

# Big Bang



# Happy Coders

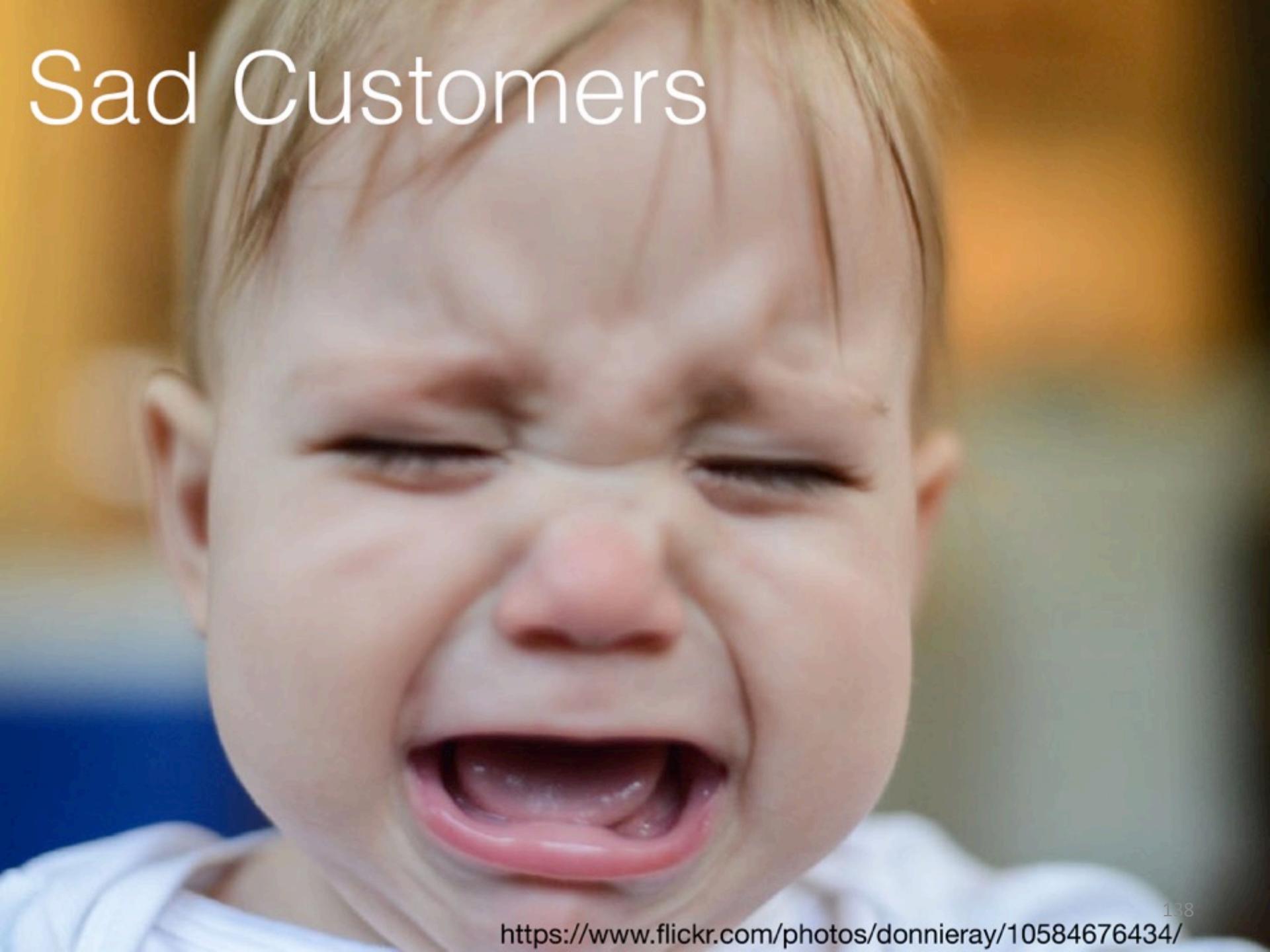


# No constraints!

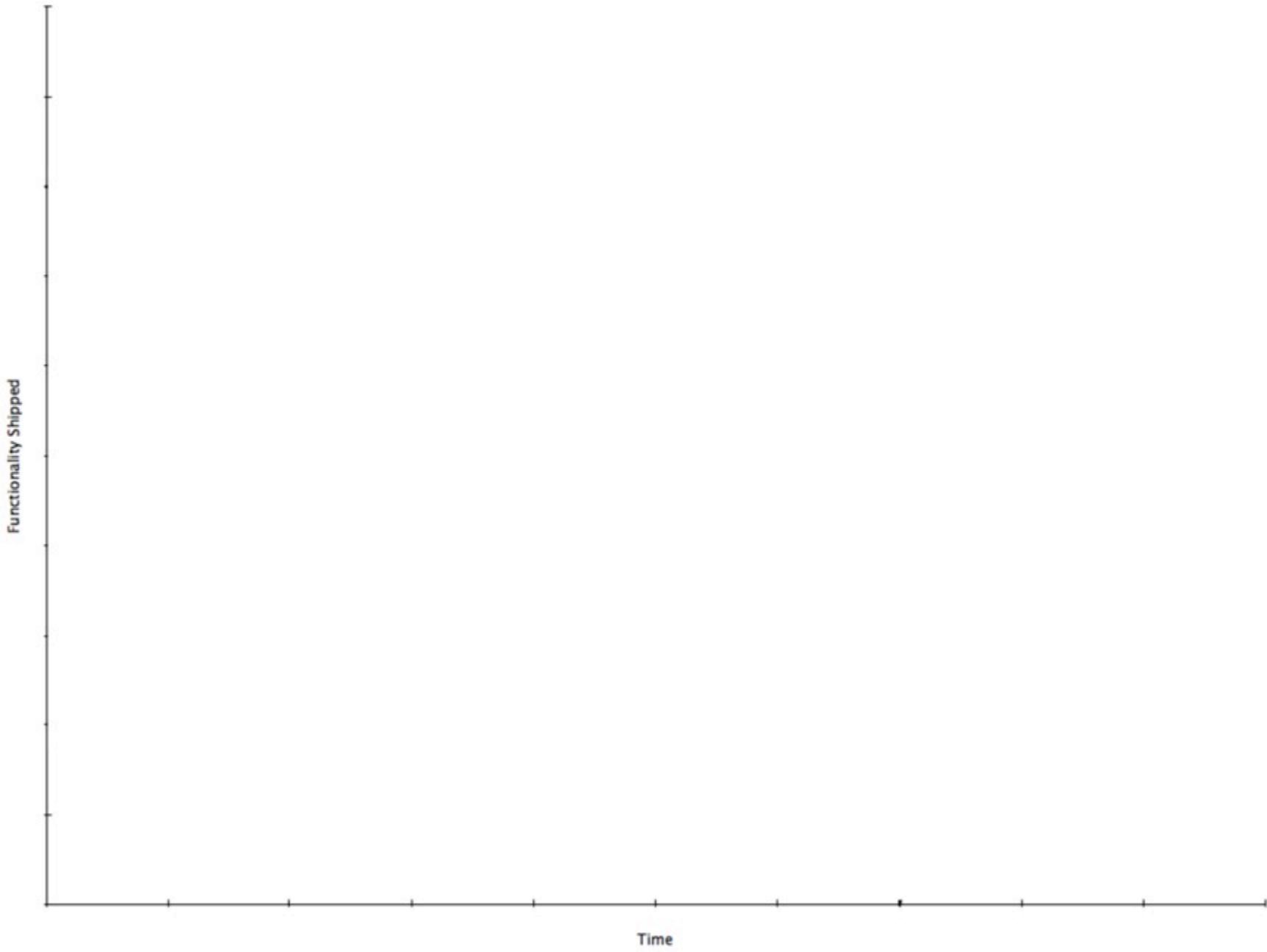
No constraints!

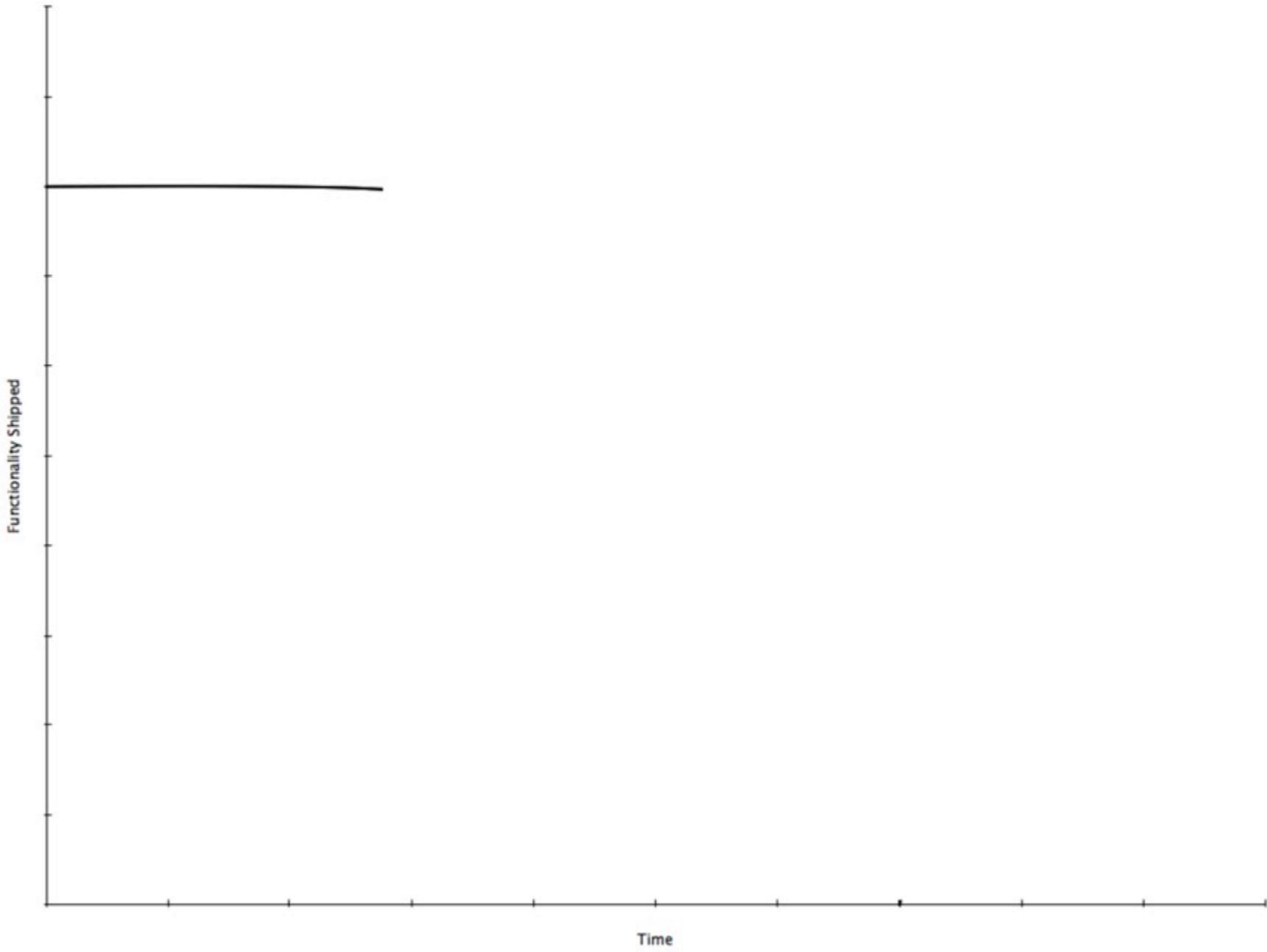
No new features!

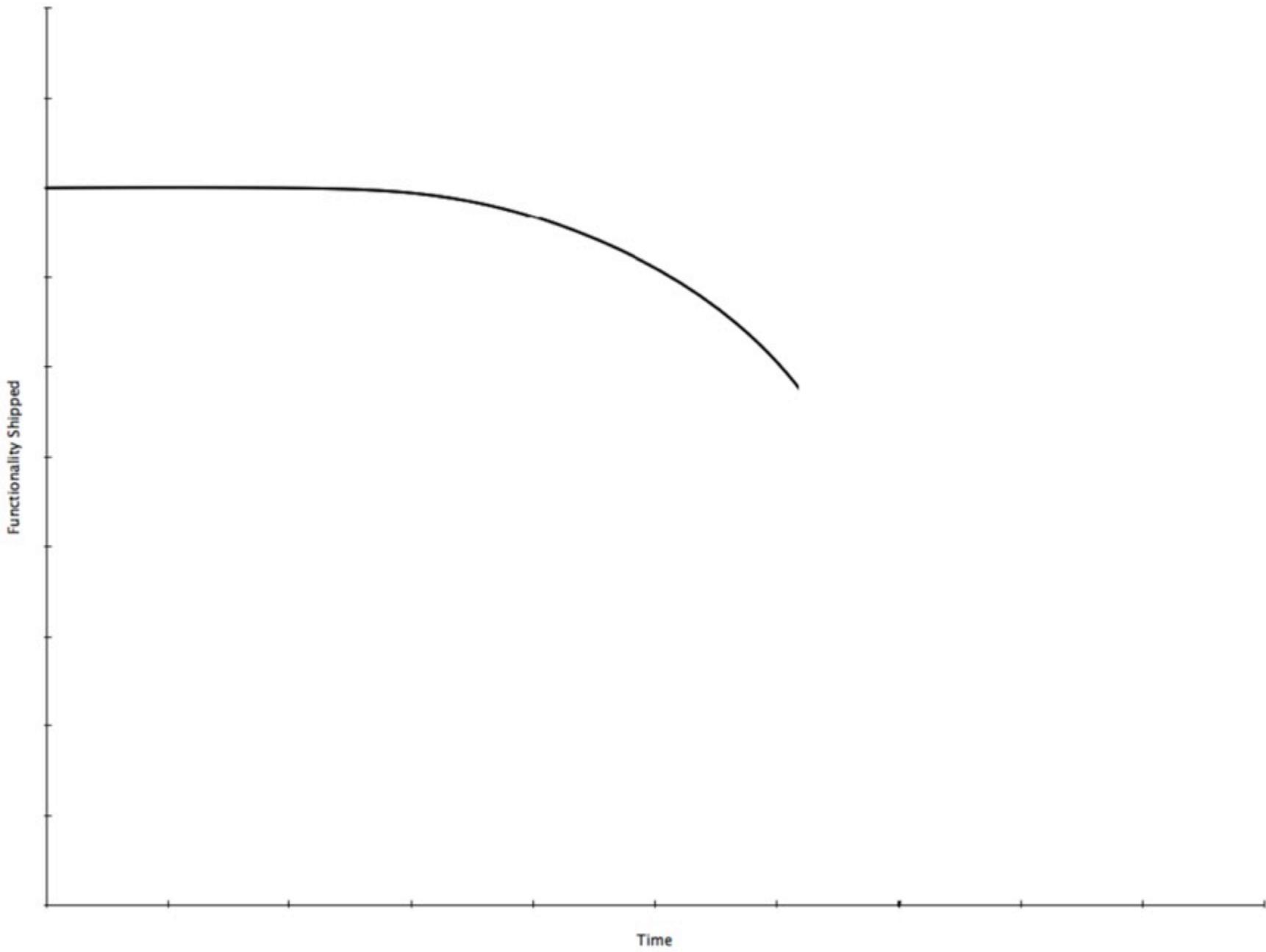
# Sad Customers

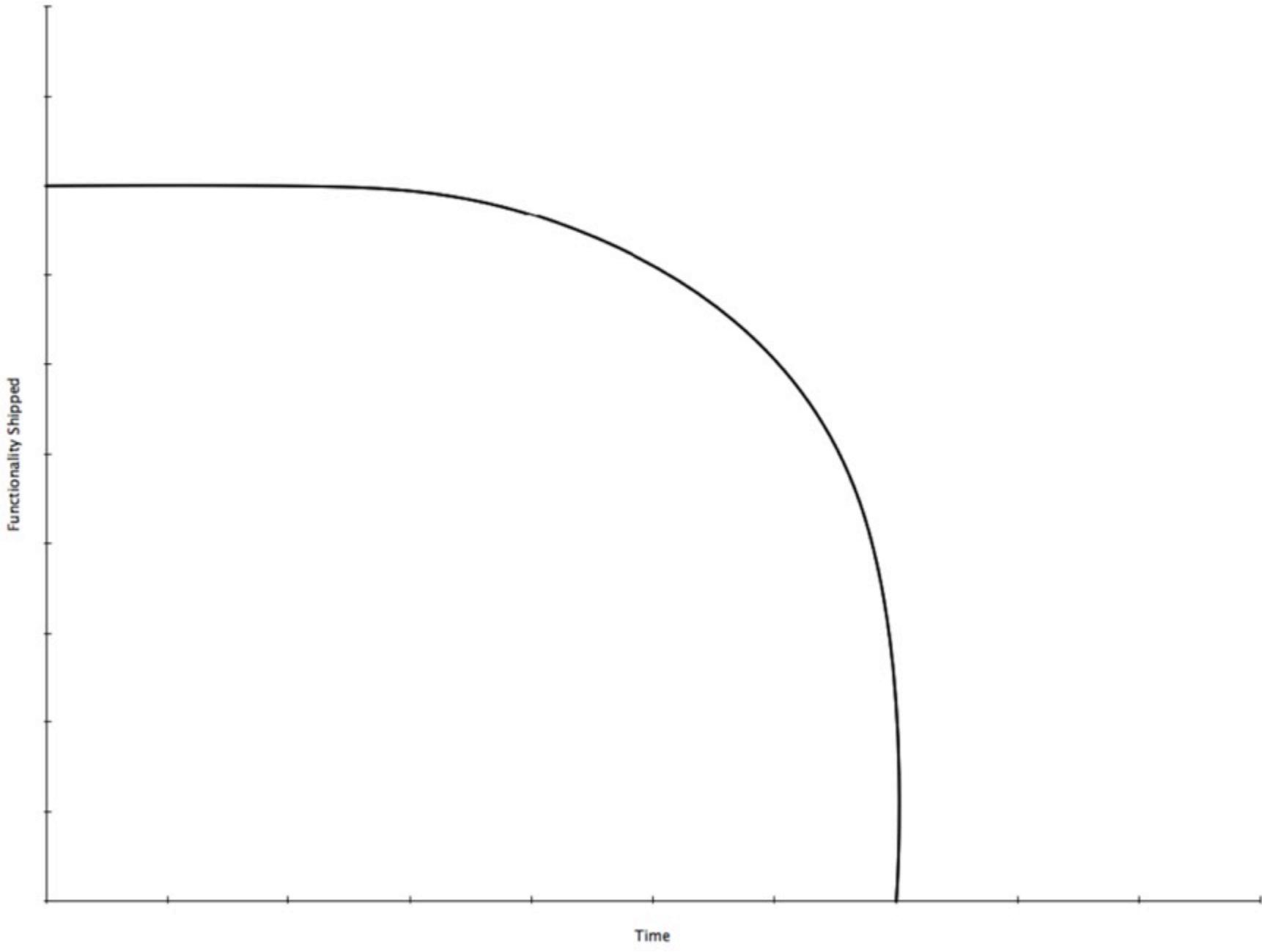


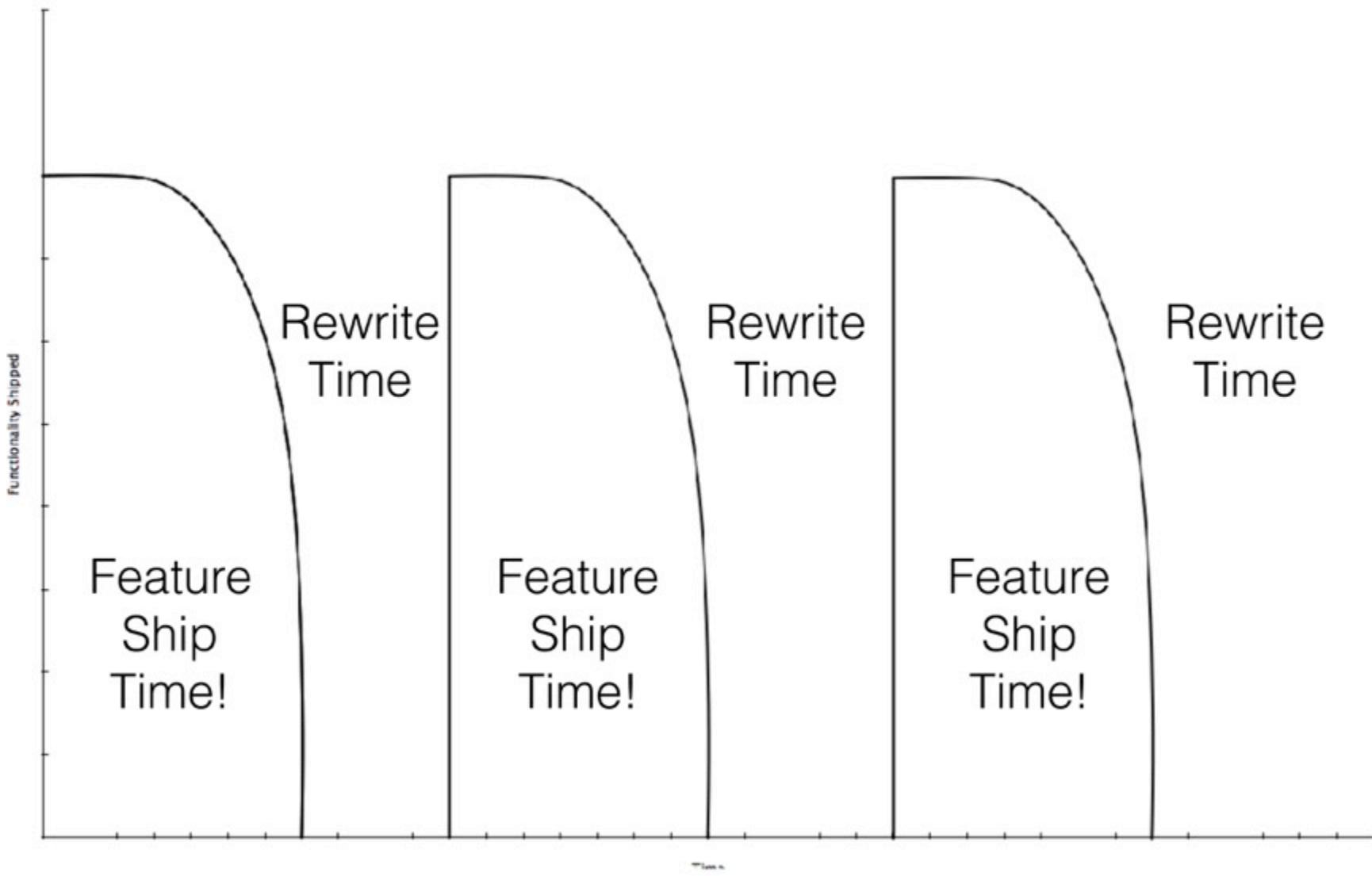
Change is inevitable











Features For Today

Platform For Tomorrow

You Need To Get Good At Incremental  
Changes To Your Architecture

# Your Homework!

# Your Homework!

- Write down why you want to use microservices

# Your Homework!

- Write down why you want to use microservices
- Brainstorm with your teams other ways you could achieve the same goals

# Your Homework!

- Write down why you want to use microservices
- Brainstorm with your teams other ways you could achieve the same goals
- Think about what you can track to see if it is working

# Your Homework!

- Write down why you want to use microservices
- Brainstorm with your teams other ways you could achieve the same goals
- Think about what you can track to see if it is working
- Think about how often you should have a review

*We want to use microservices because:*

*My customers will care because:*

*We could also consider these things to achieve  
the same goal:*

*We'll track these things to see how we're going:*

*We'll have a checkpoint every:*

Know why you're doing it

Know why you're doing it

Decide what to measure

Know why you're doing it

Decide what to measure

Track & adapt

Know why you're doing it

Decide what to measure

Track & adapt

Do this while shipping features

# **MONOLITHS**

## **TO MICROSERVICES**

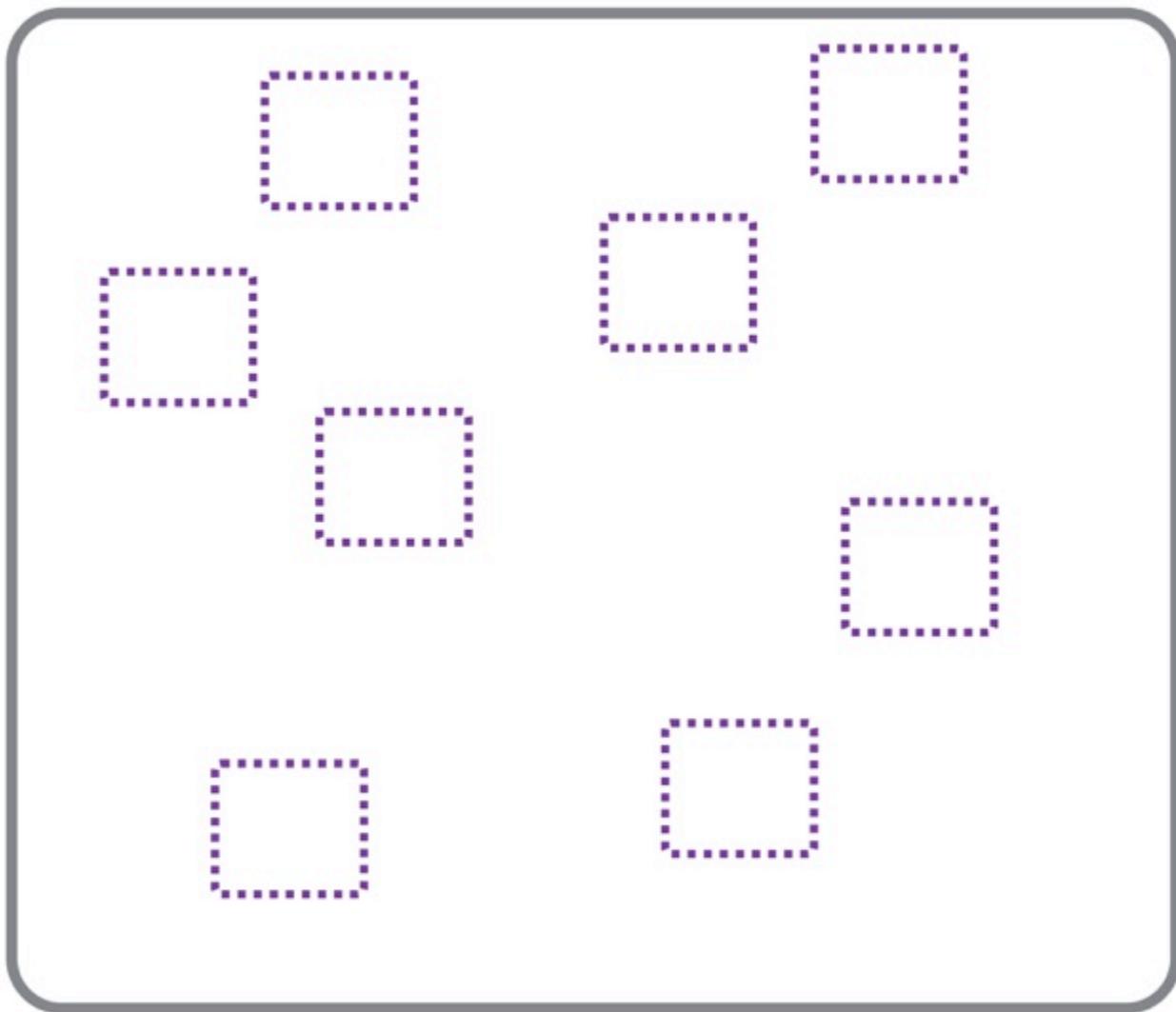
Sam Newman

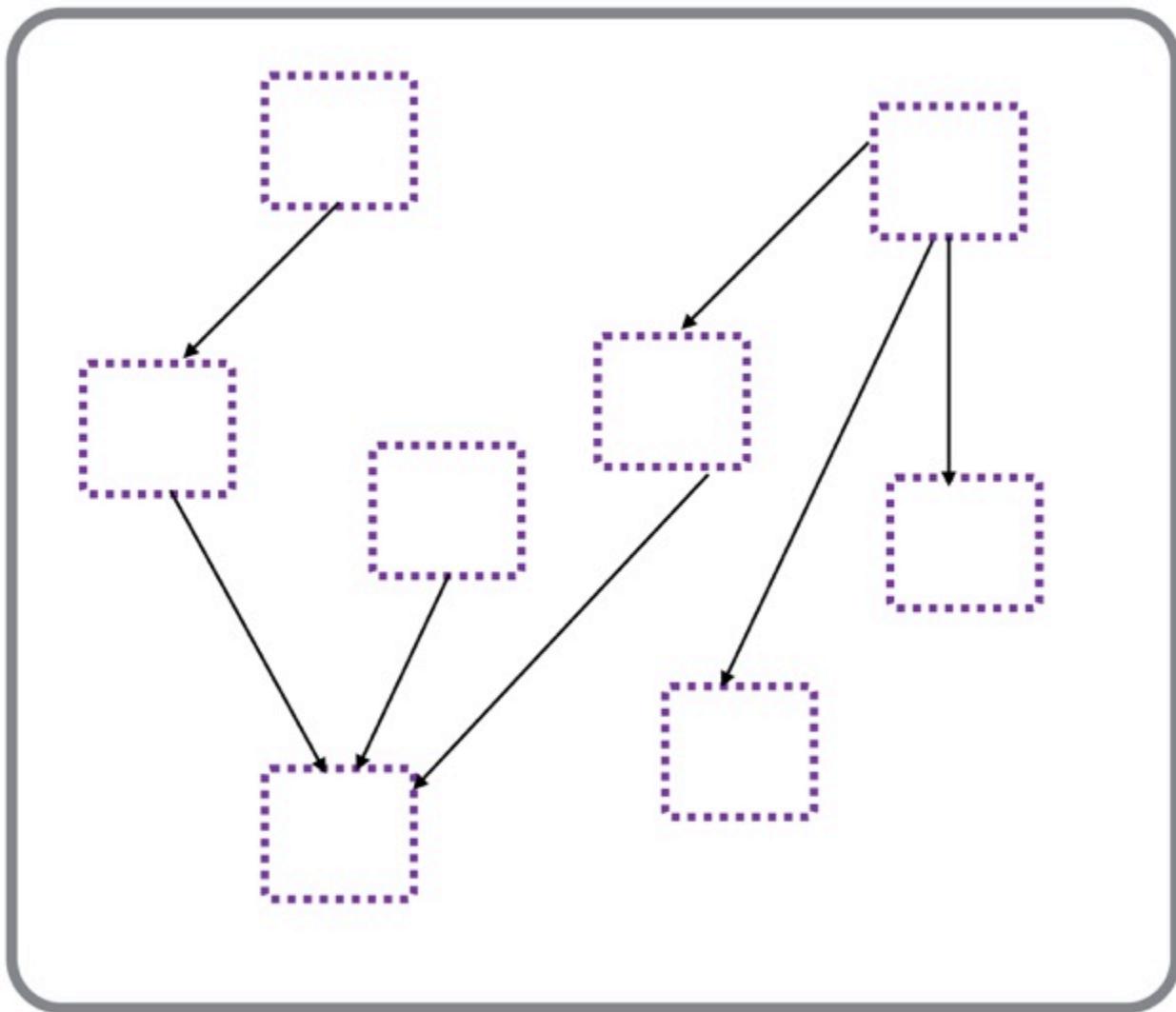
# Migrating A Running System

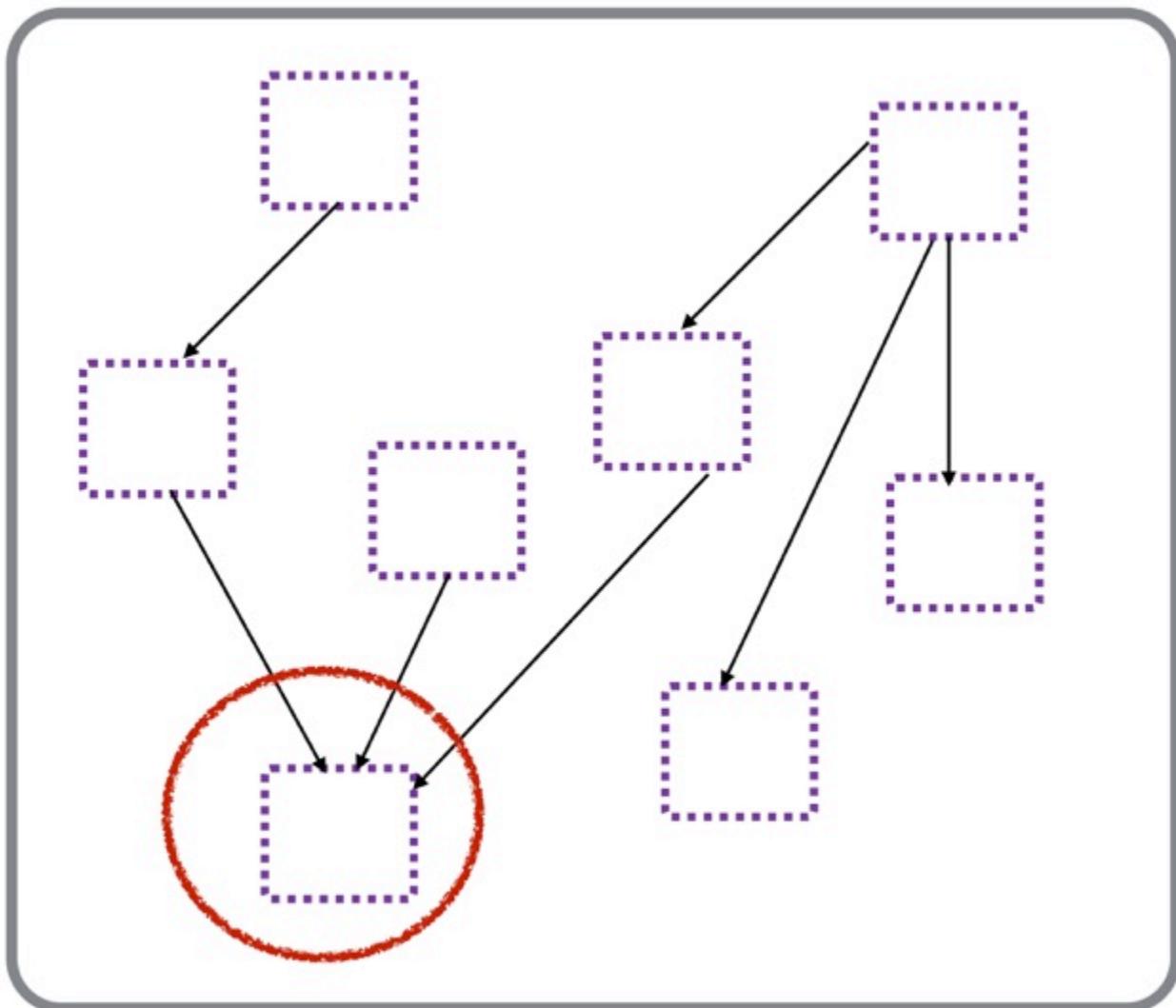


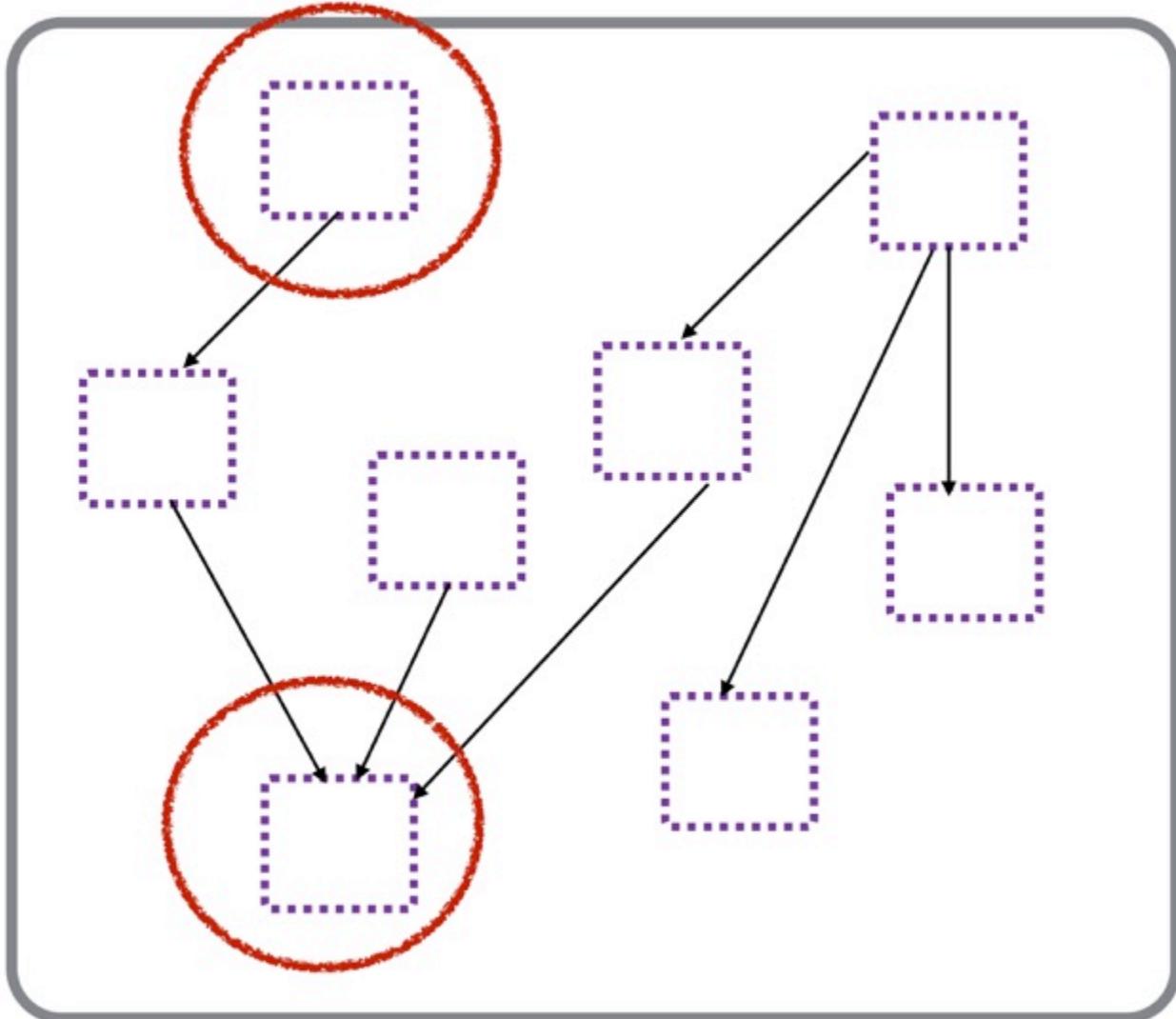


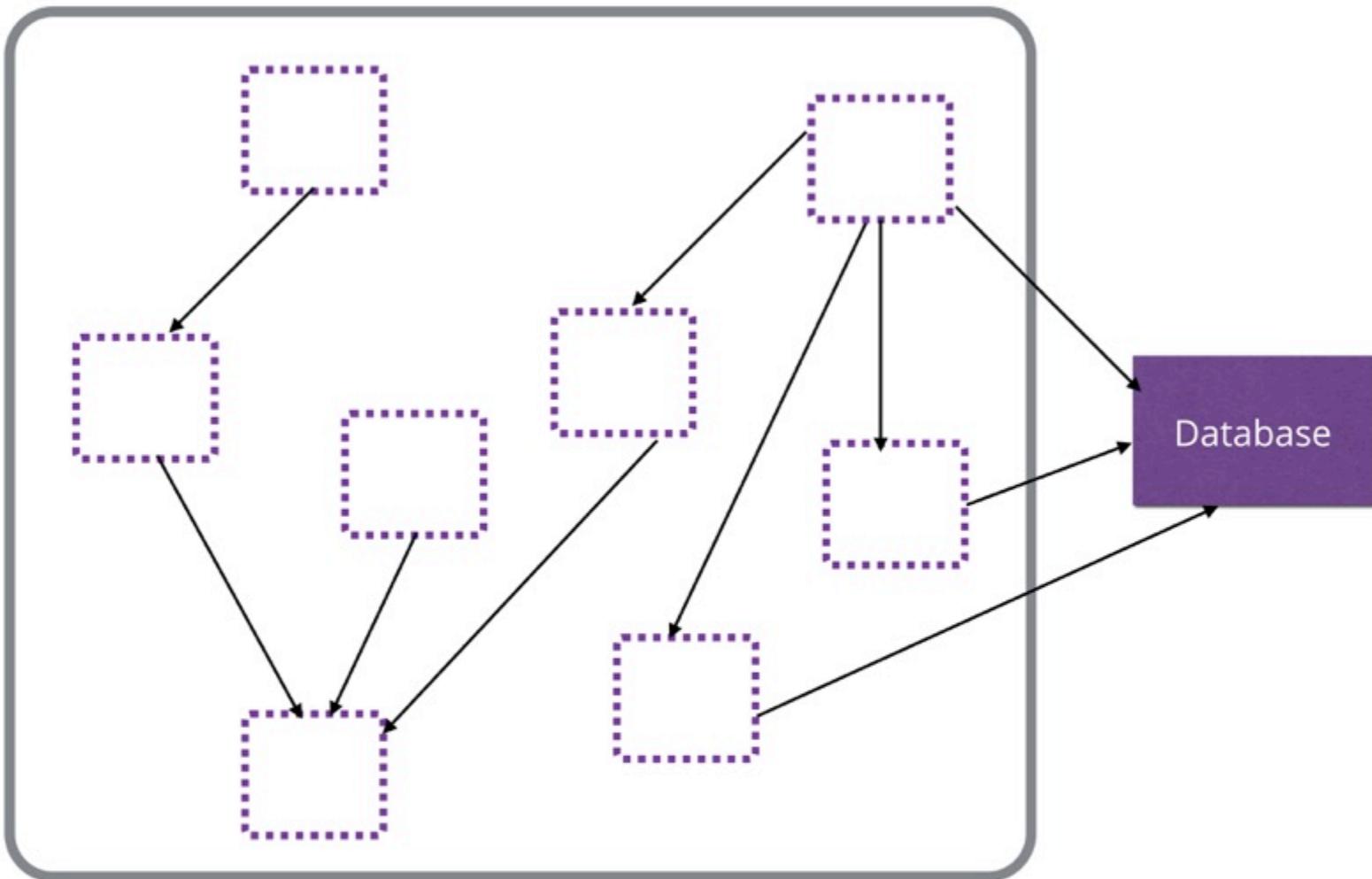
Where to start?

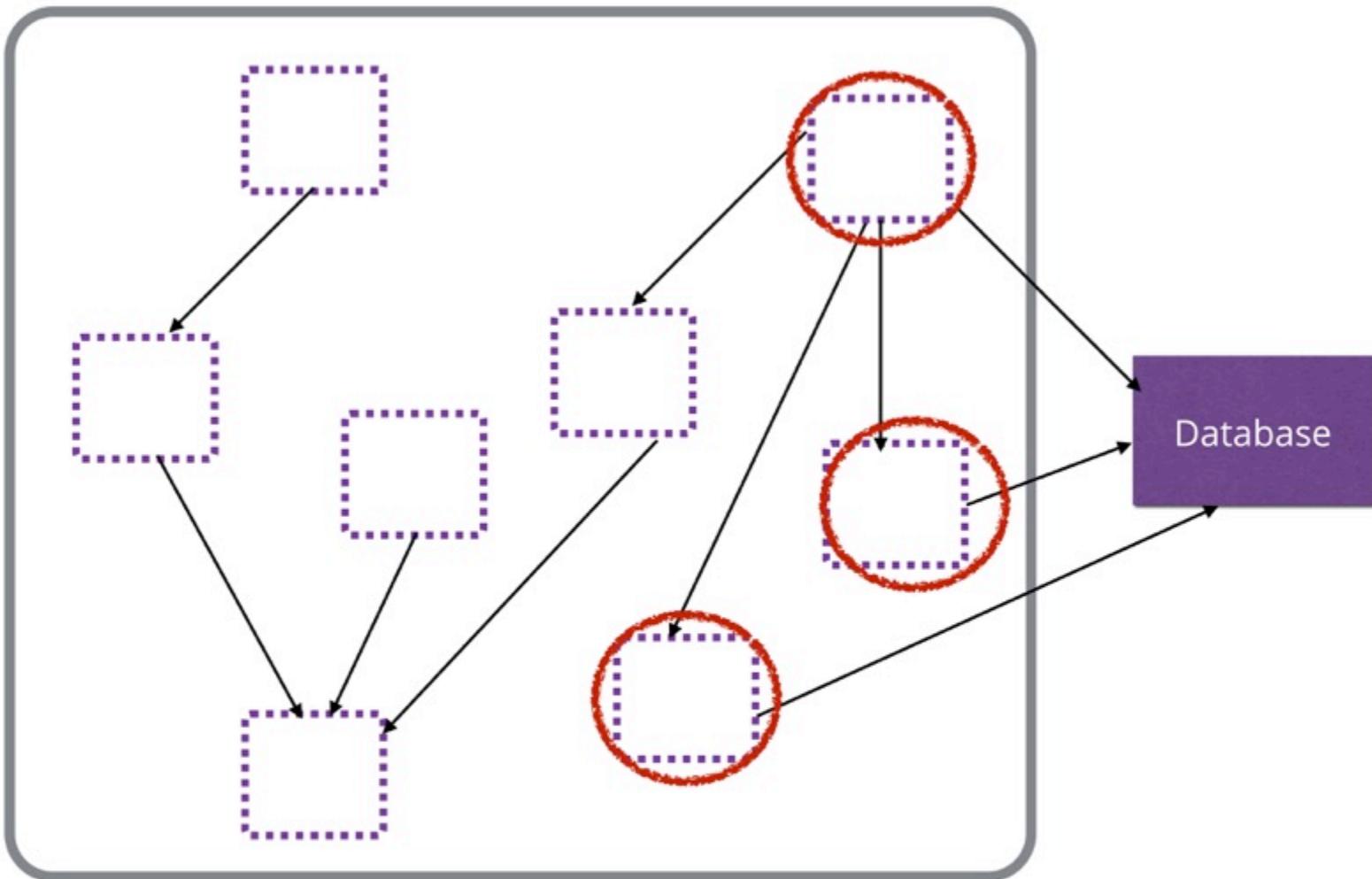


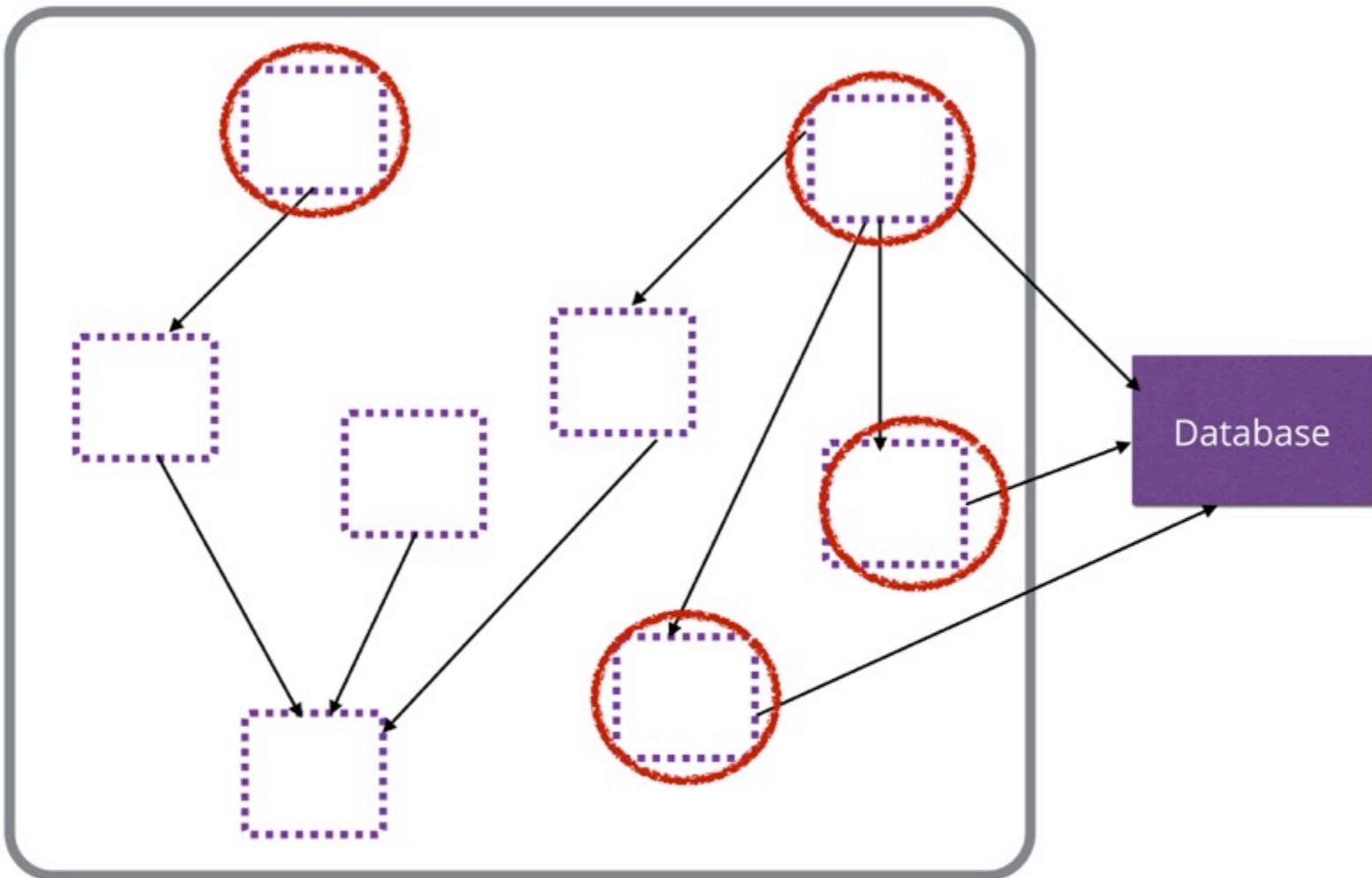












So all things being equal, look for:

So all things being equal, look for:

features/contexts with few  
inbound dependencies

So all things being equal, look for:

features/contexts with few  
inbound dependencies

which do not persist state

# Ease Of Decompositon

vs

# Benefits Of Decompositon

# Benefits Of Decomposition?

Why are you doing this?

For time to market?

For time to market?

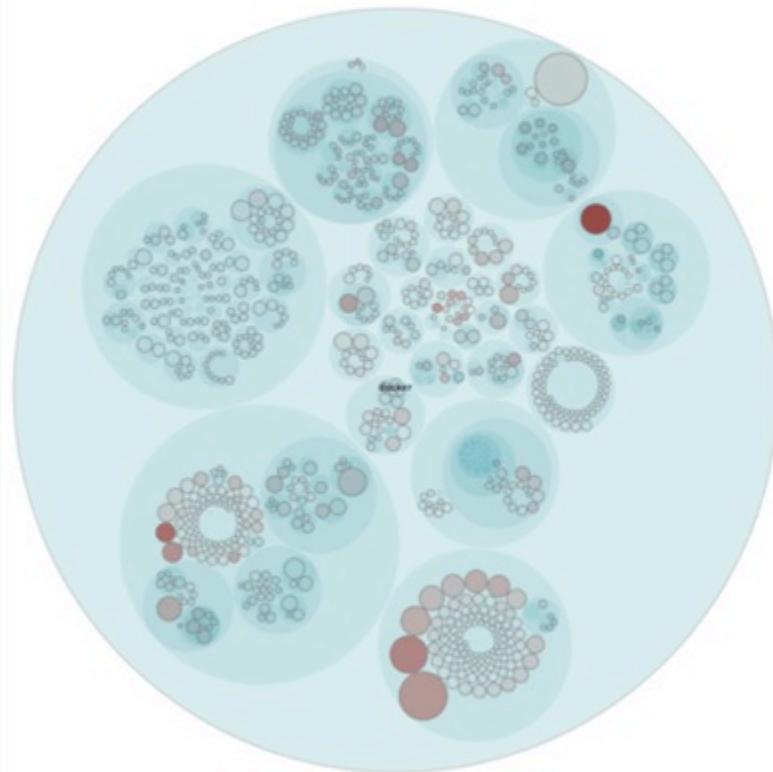
Look at code liveliness

## Hotspots

Main Suspects

Hotspots

Change Frequency Distribution



File	Change Frequency	Complexity/Size	Actions
docker/hack/vendor.sh	295	6	
docker/contrib/completion/zsh/_docker	282	2,355	
docker/daemon/daemon.go	161	978	
docker/integration-cli/docker_cli_run_test.go	127	3,658	
docker/daemon/daemon_unix.go	111	1,831	
docker/Dockerfile	109	221	
docker/integration-cli/docker_cli_build_test.go	184	6,866	

◀ 1 - 7 ▶

<https://codescene.io/>

Also check your backlog!

And measure your cycle  
time as you go!

For scale?

For scale?

Identify your bottlenecks



Industrialize your performance tests

# Our Enterprise Extension



Advanced reporting &  
Advanced automation

DISTRIBUTED MODE

NEW METRICS

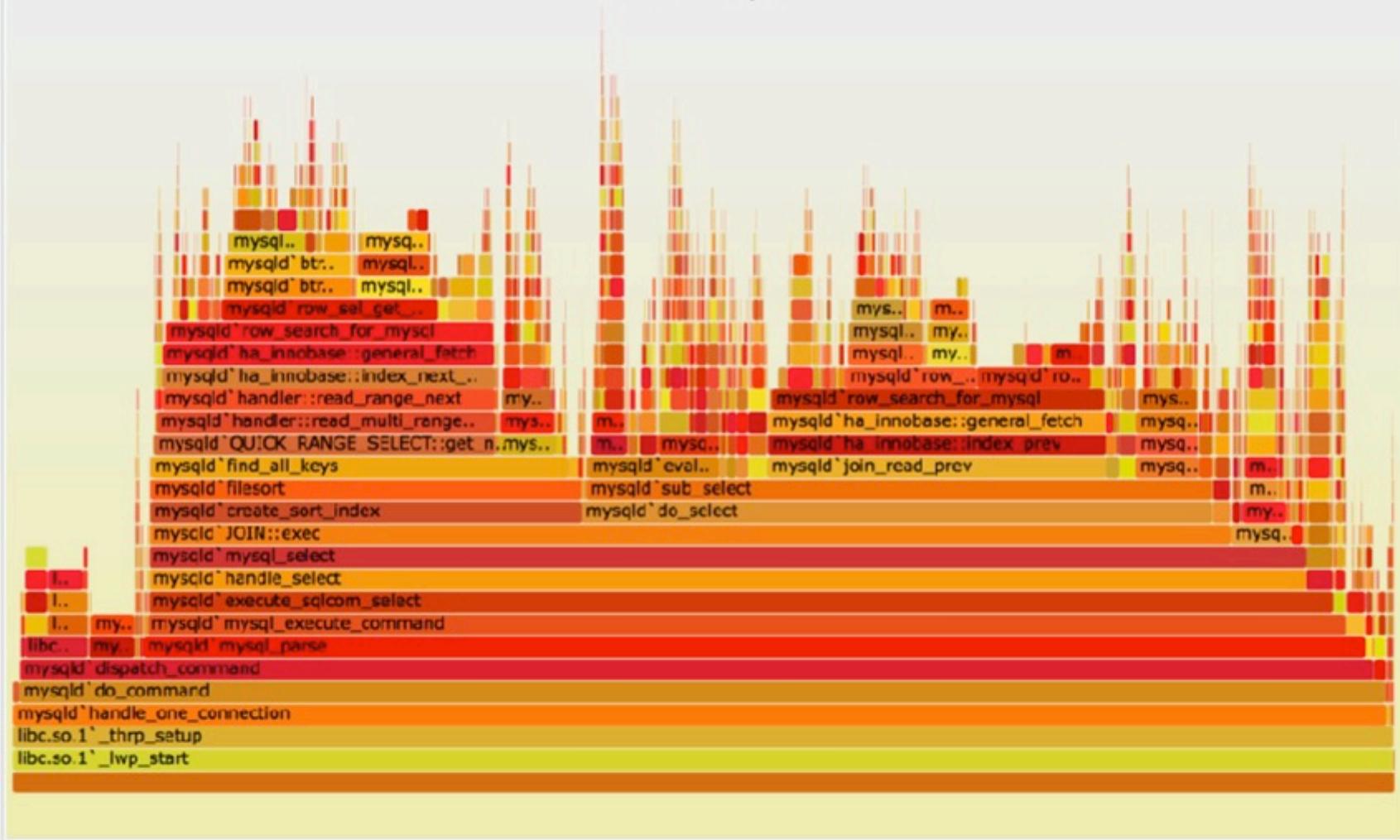
LIVE METRICS

TRENDS

<http://gatling.io>

## Flame Graph

Search

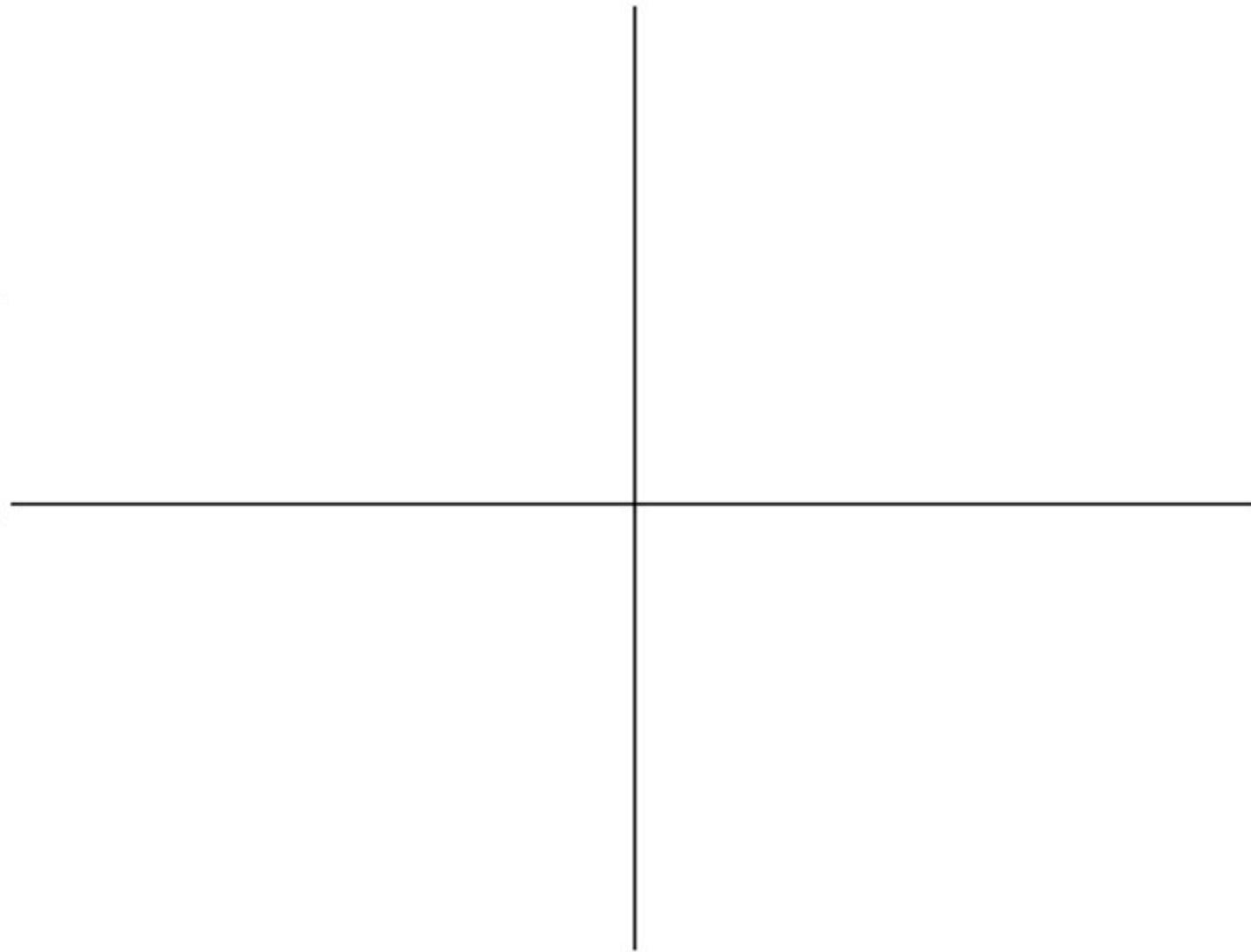


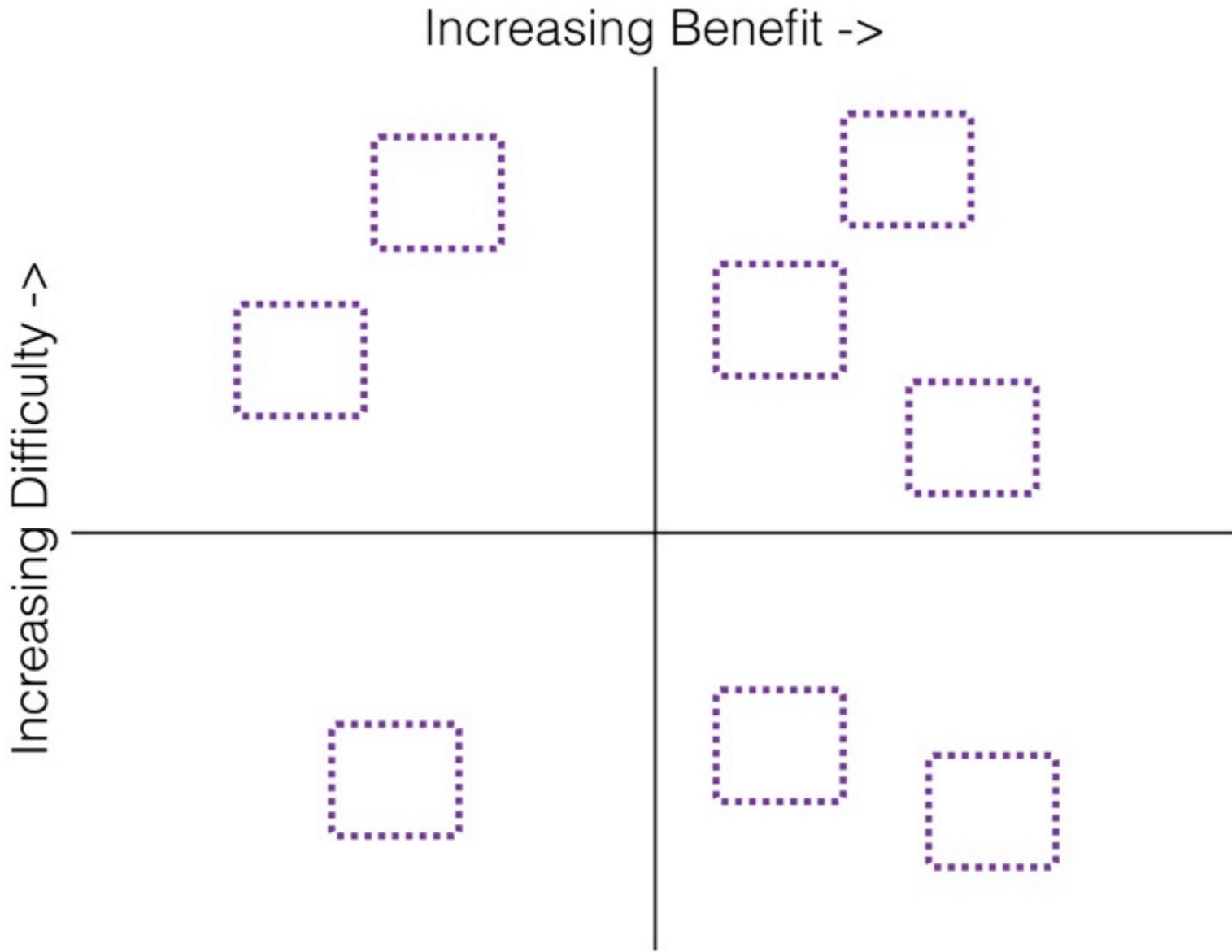
<http://www.brendangregg.com/flamegraphs.html>

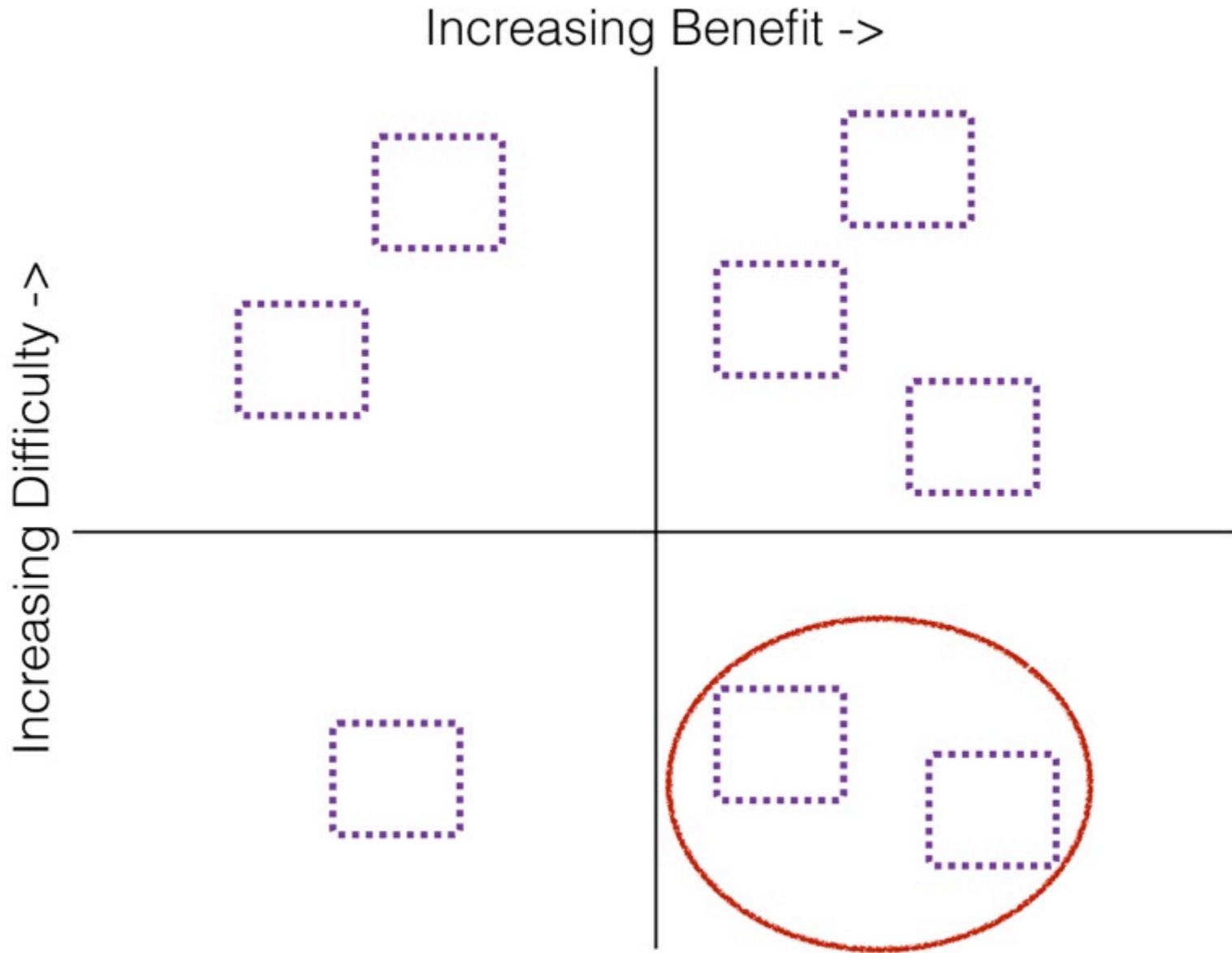
Continually measure

Increasing Difficulty ->

Increasing Benefit ->







# Incremental Change

“If you do a big bang rewrite, the only thing  
you’re certain of is a big bang”

- *Martin Fowler (paraphrased)*







VS



VS



Our system architecture needs to  
be as fluid as our business needs

But how do you incrementally  
break down a monolithic system?

Enter the strangler pattern

Enter the strangler pattern

Great pattern, terrible name



Implementing a strangler pattern?

Implementing a strangler pattern?

## **1. Asset capture:**

identify the functionality to move to a new microservice

Implementing a strangler pattern?

## **1. Asset capture:**

identify the functionality to move to a new microservice

## **2. Redirect calls**

Intercept calls to old functionality, and redirect to the new service

# Moving functionality?

Moving functionality?

It might be copy and paste

Moving functionality?

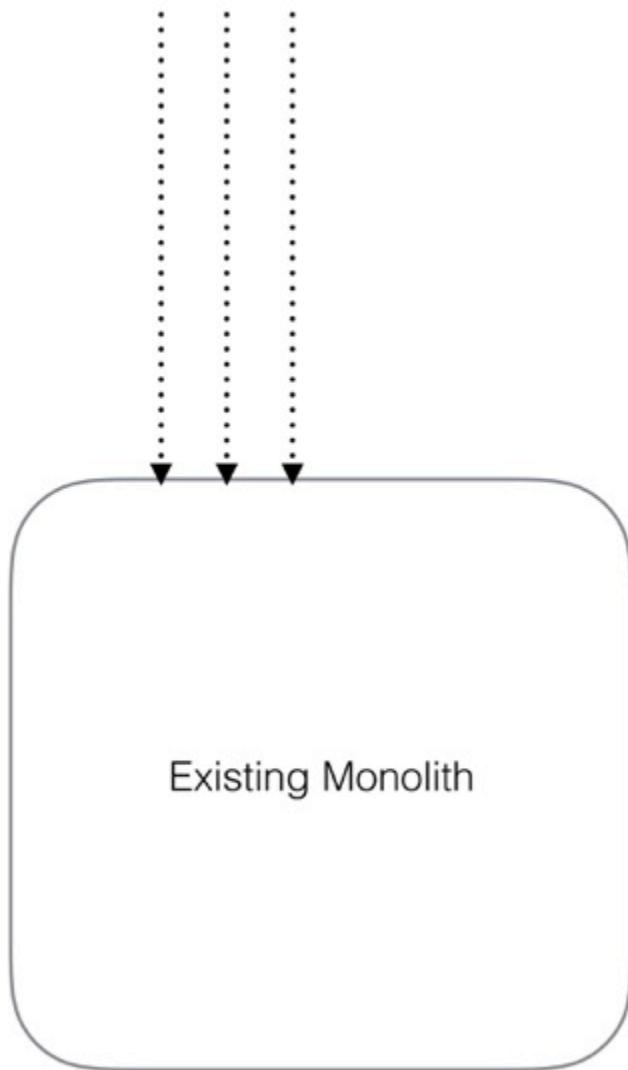
It might be copy and paste

More likely is a total or  
partial rewrite

## HTTP PROXY



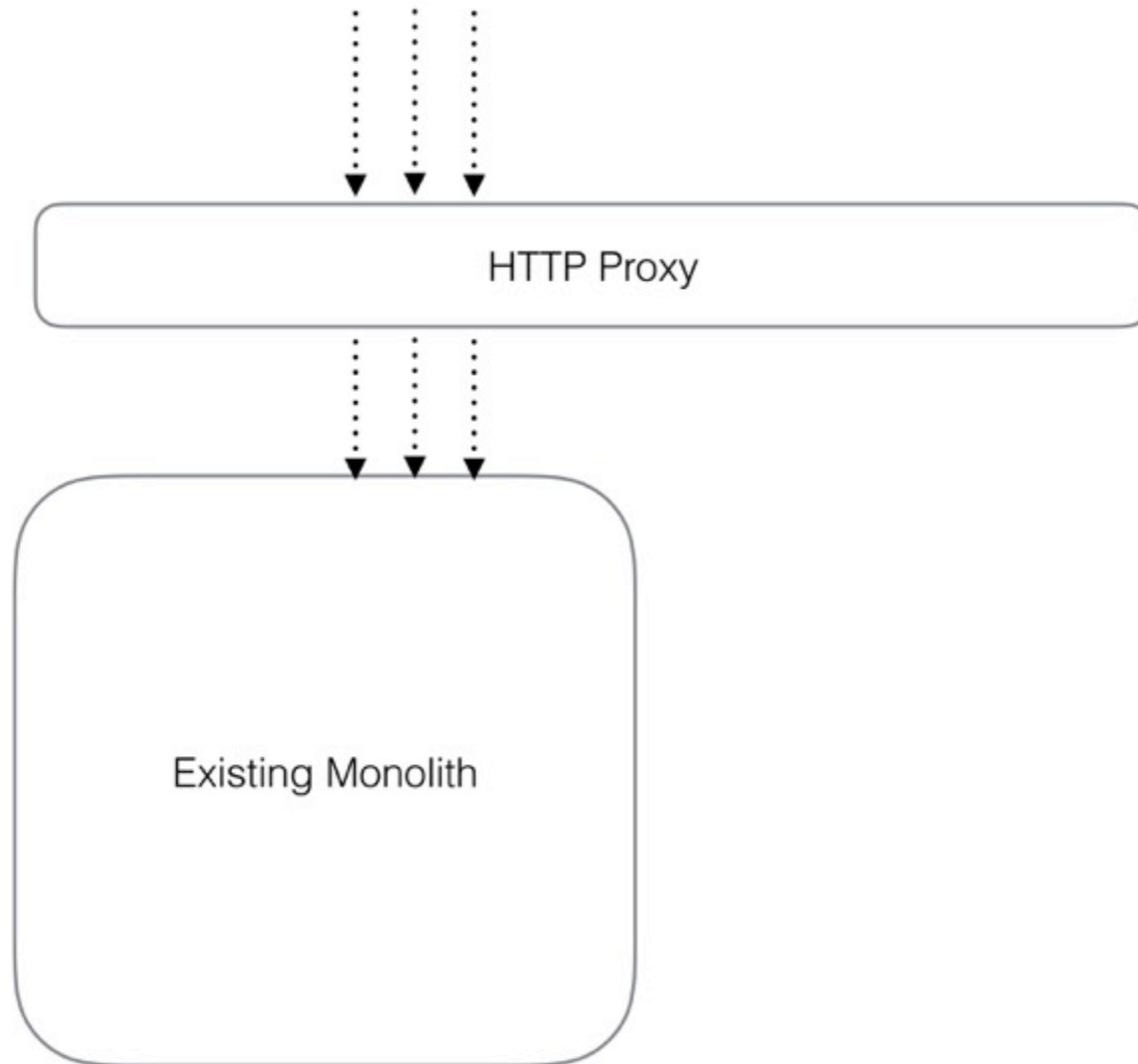
## HTTP PROXY



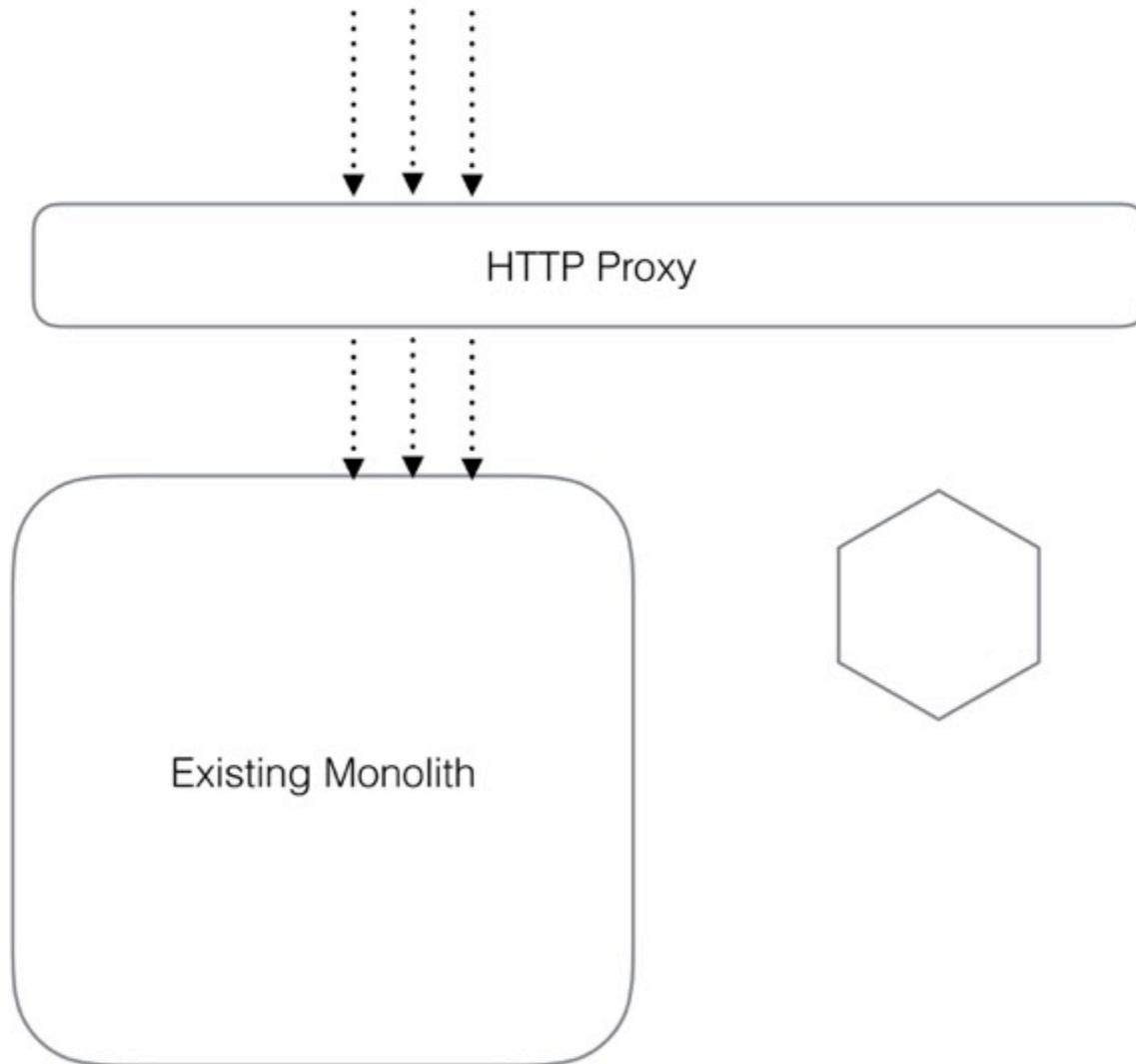
## HTTP PROXY



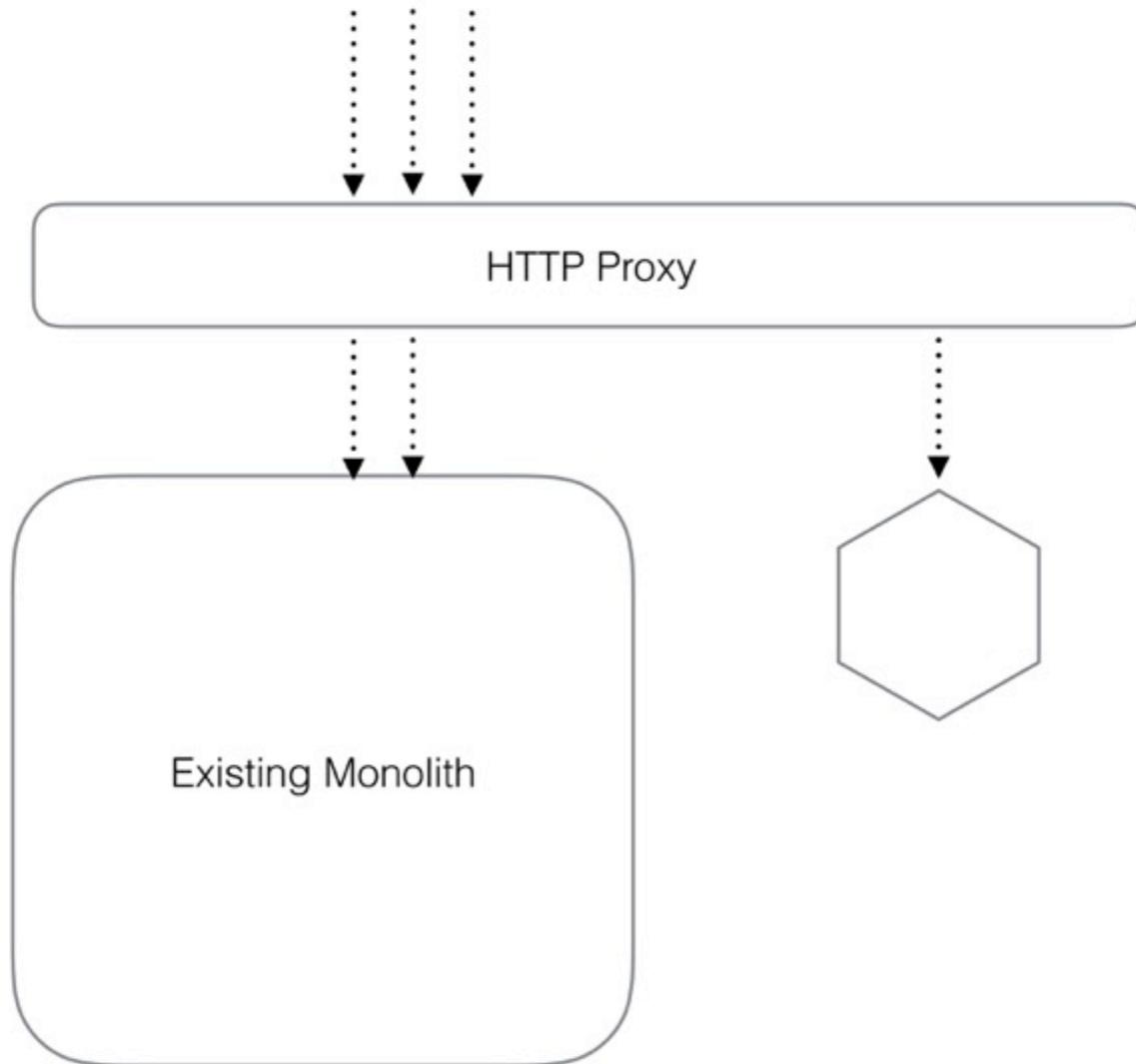
## HTTP PROXY



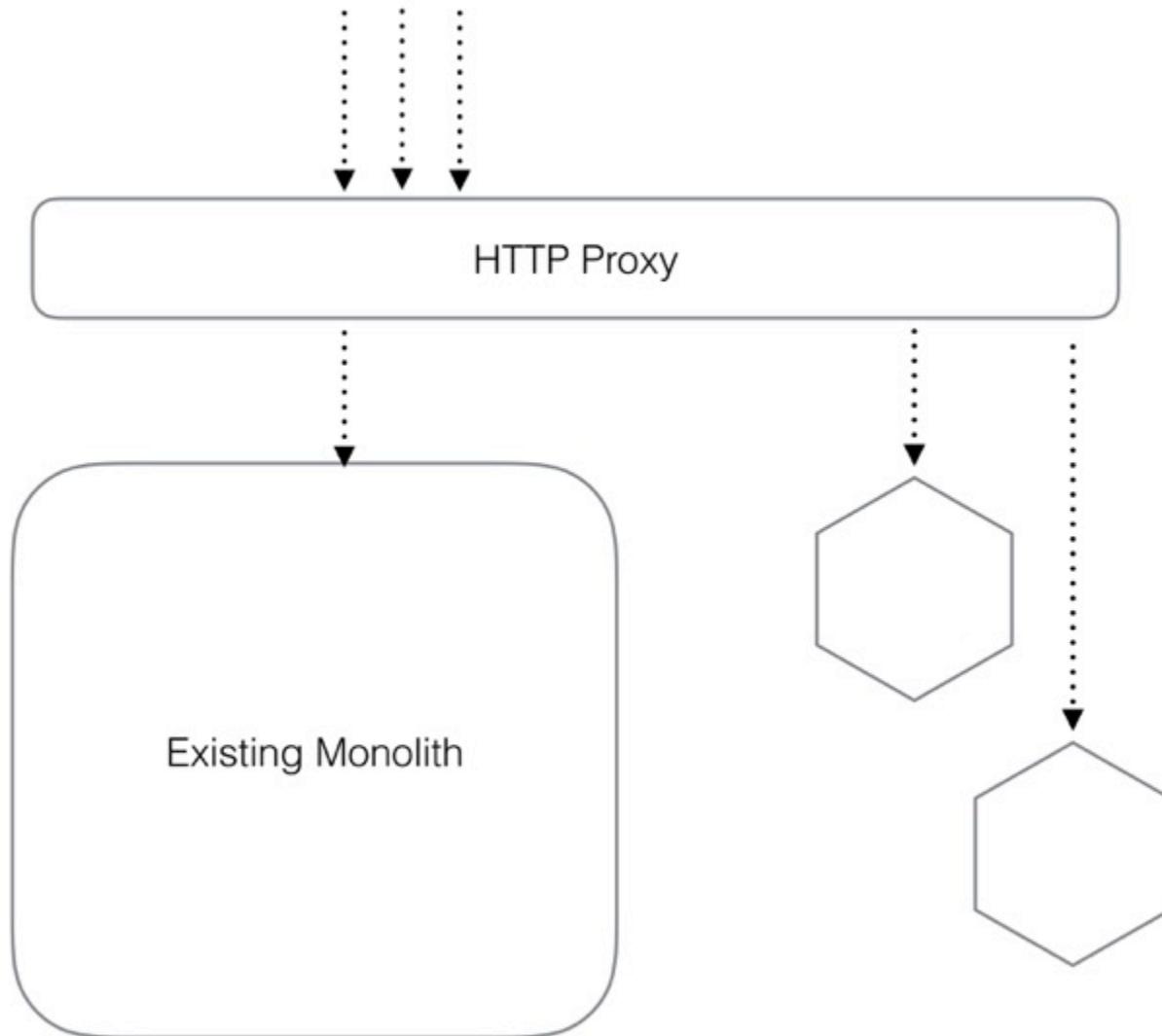
## HTTP PROXY



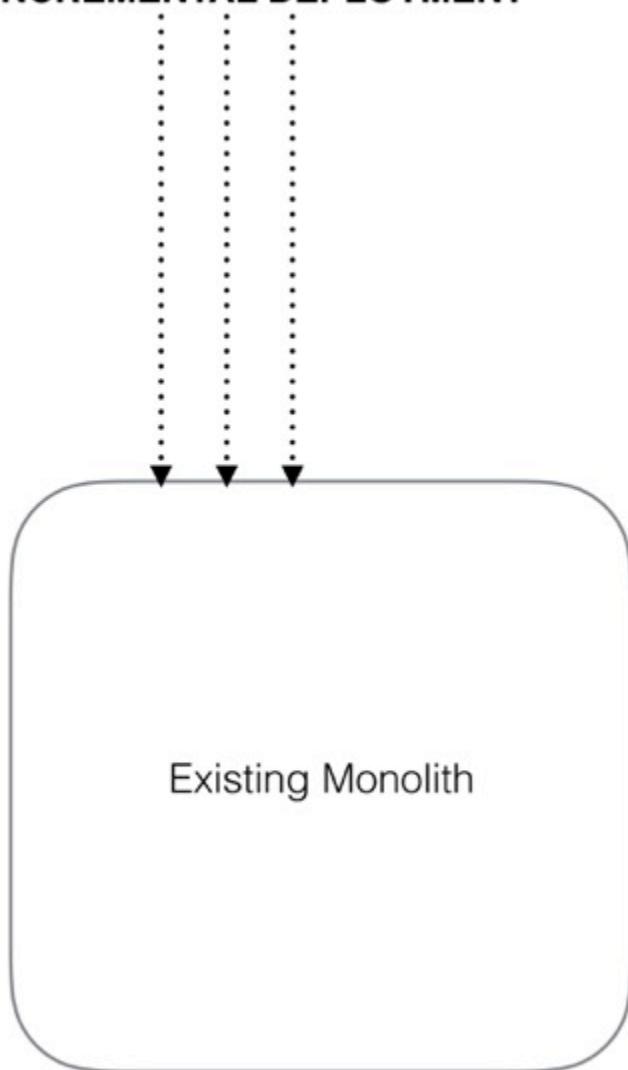
## HTTP PROXY



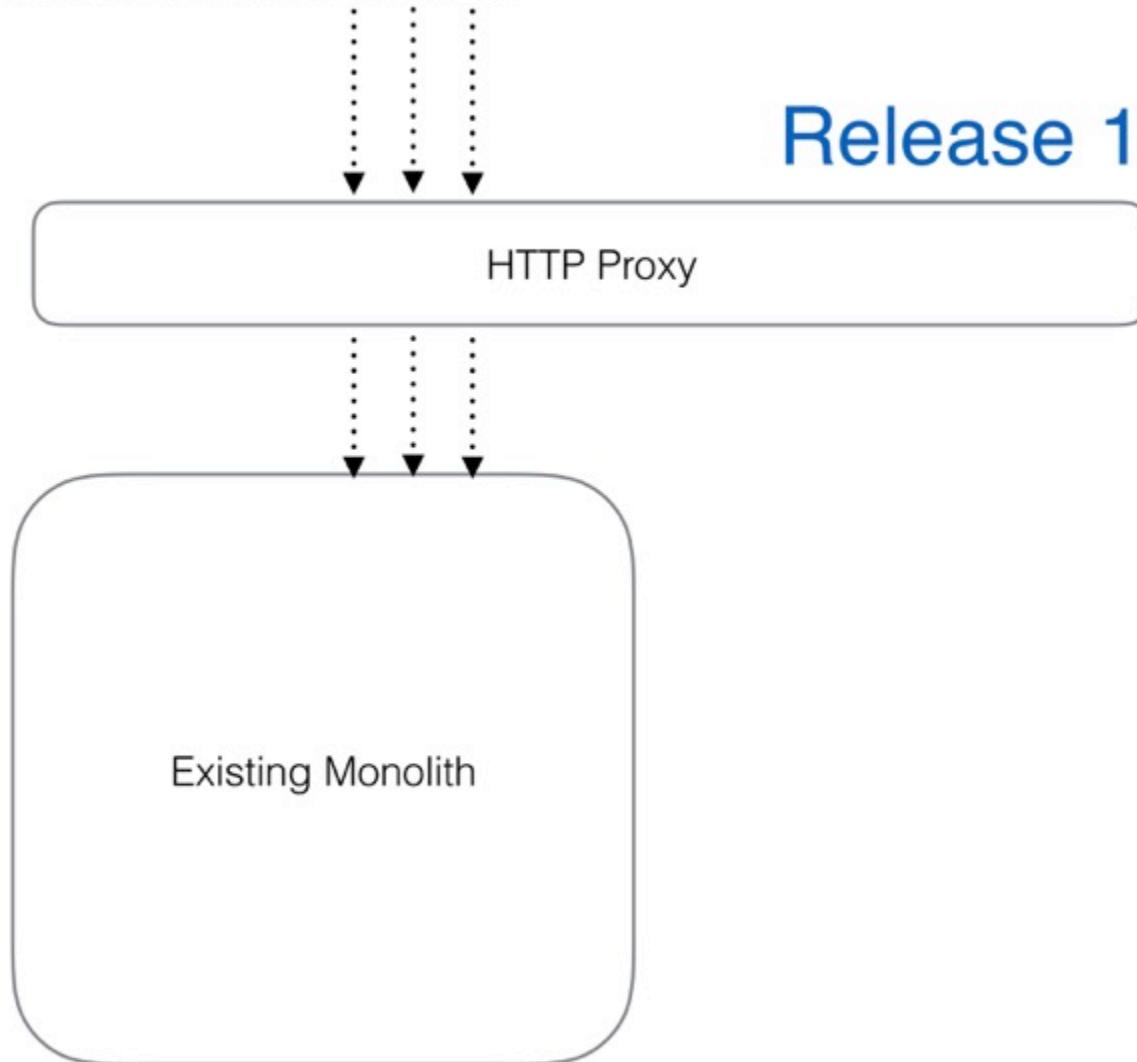
## HTTP PROXY



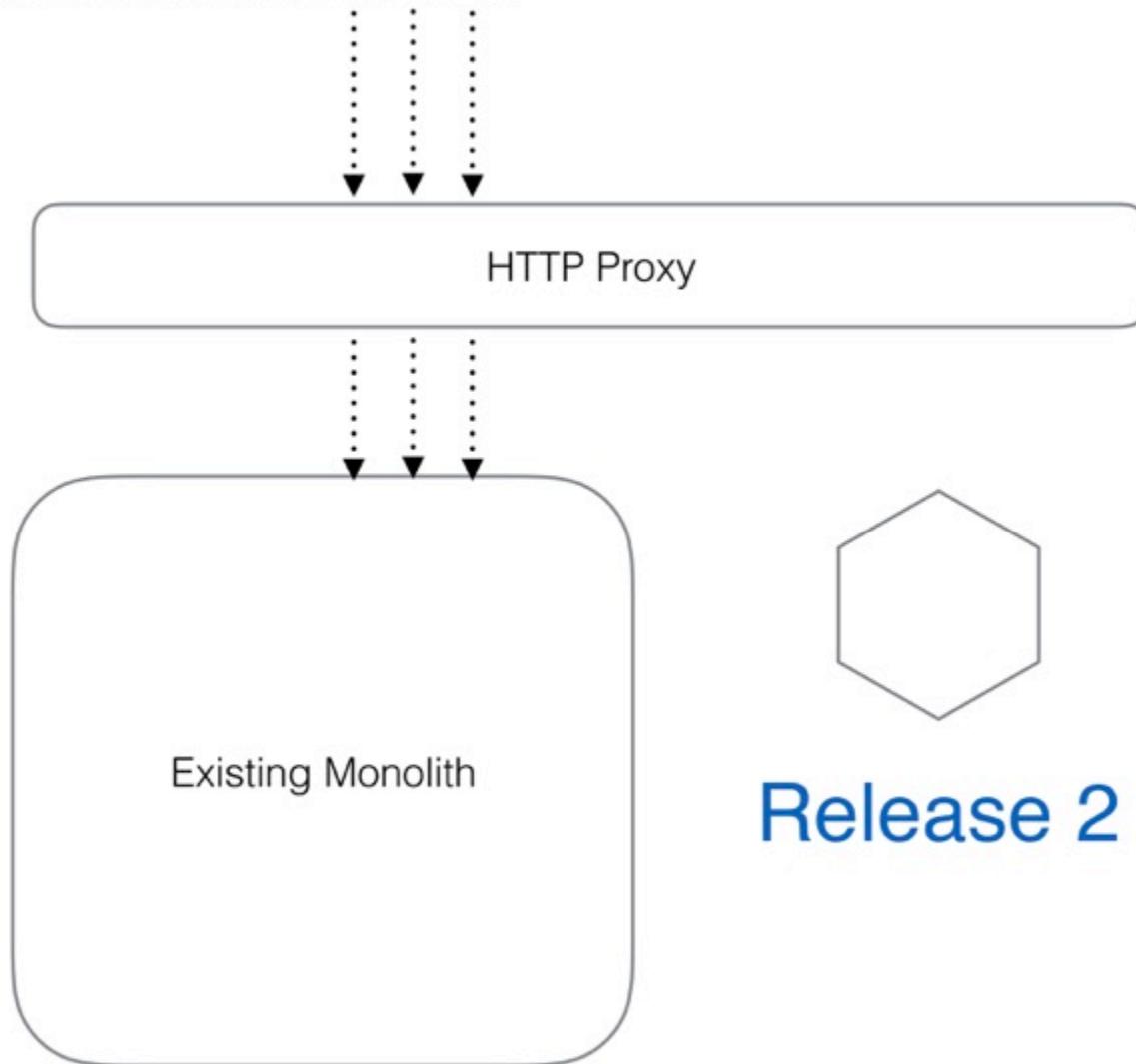
## HTTP PROXY - INCREMENTAL DEPLOYMENT



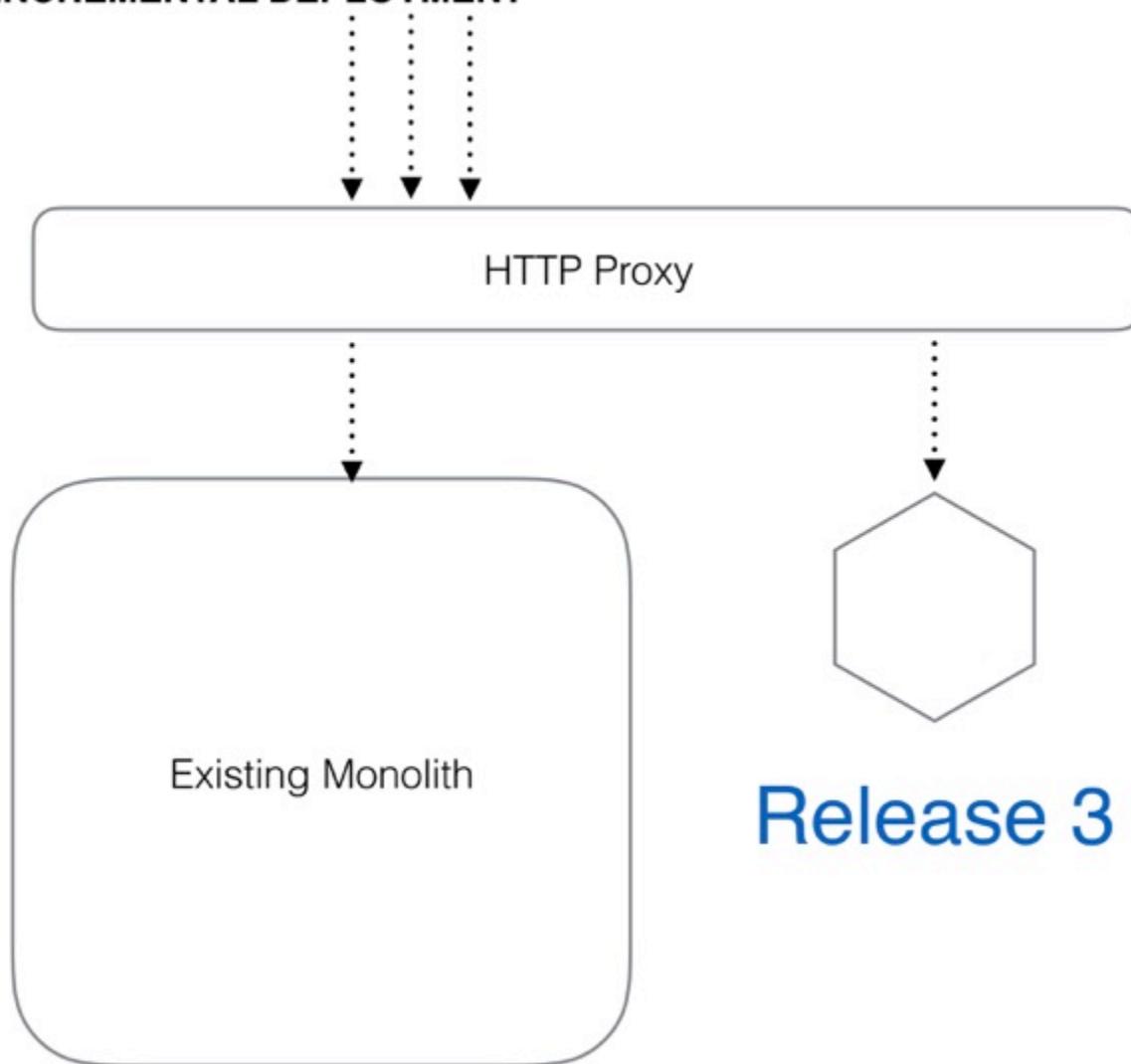
## HTTP PROXY - INCREMENTAL DEPLOYMENT



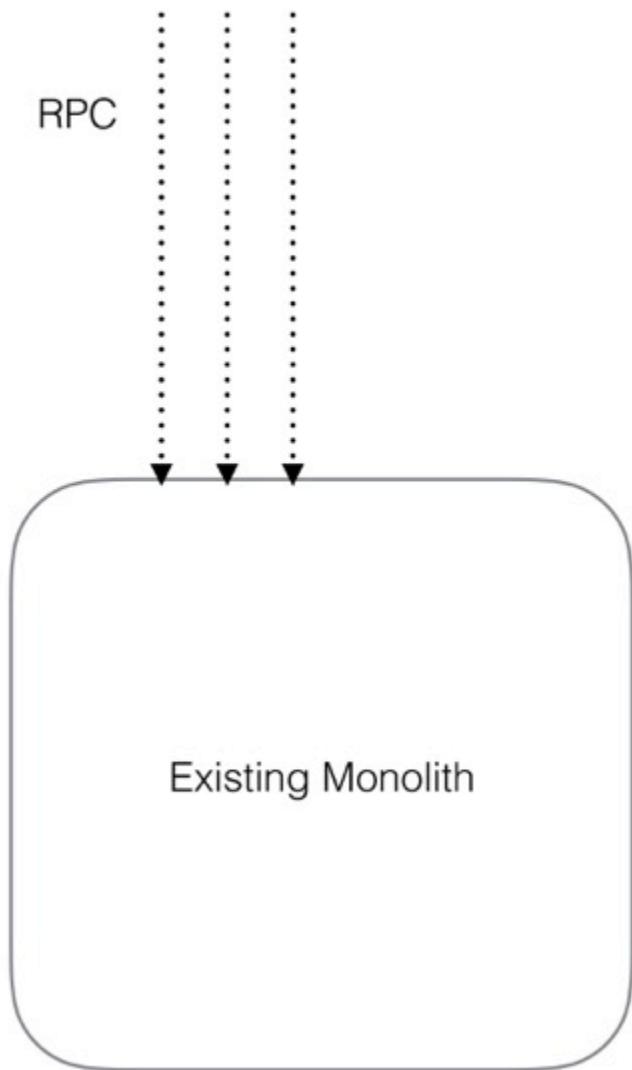
## HTTP PROXY - INCREMENTAL DEPLOYMENT



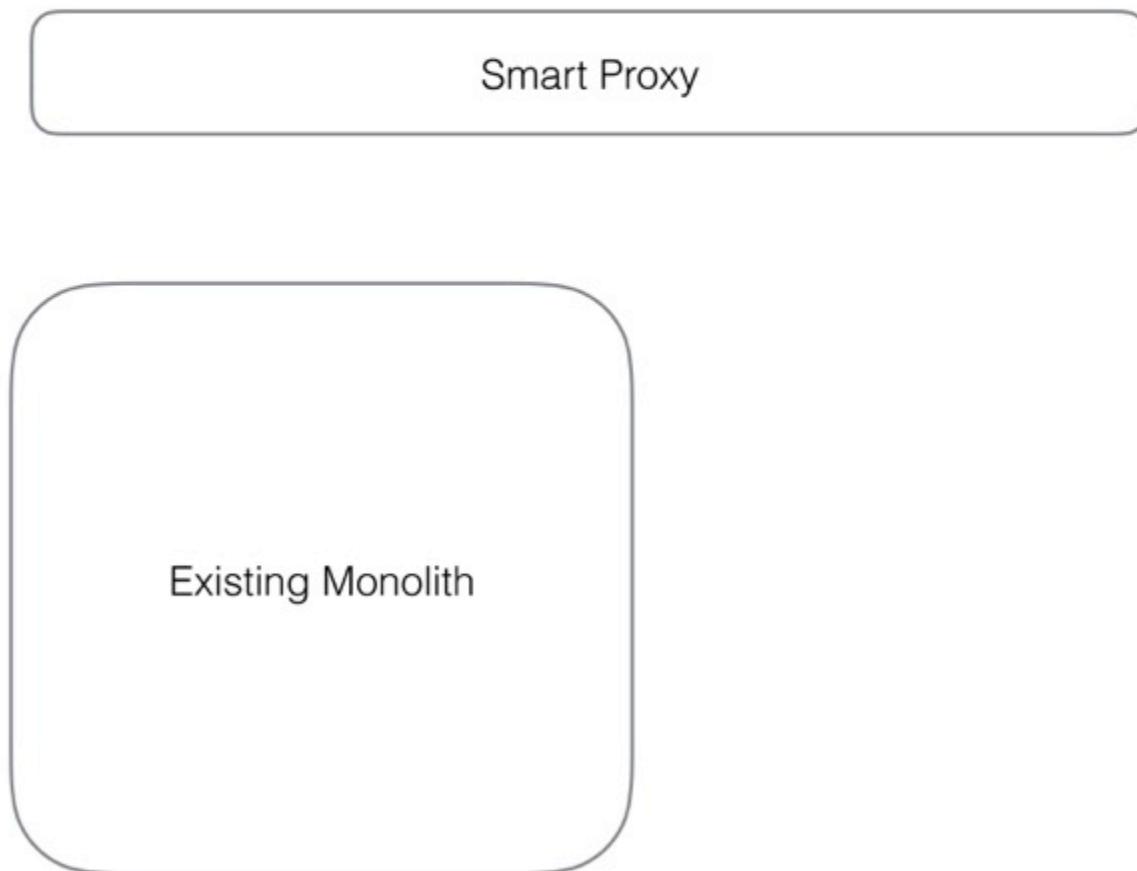
## HTTP PROXY - INCREMENTAL DEPLOYMENT



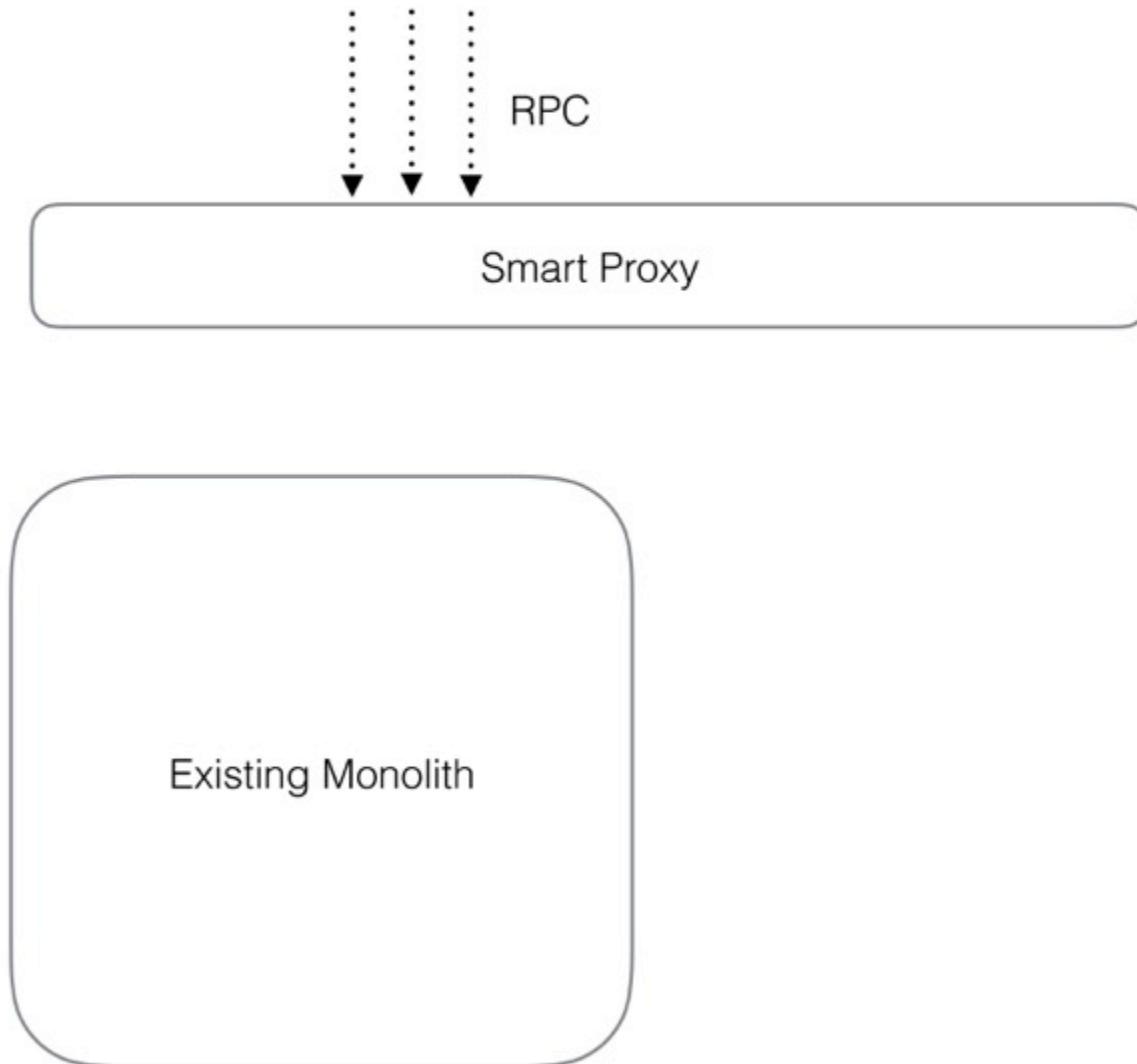
## PROTOCOL CHANGING PROXY



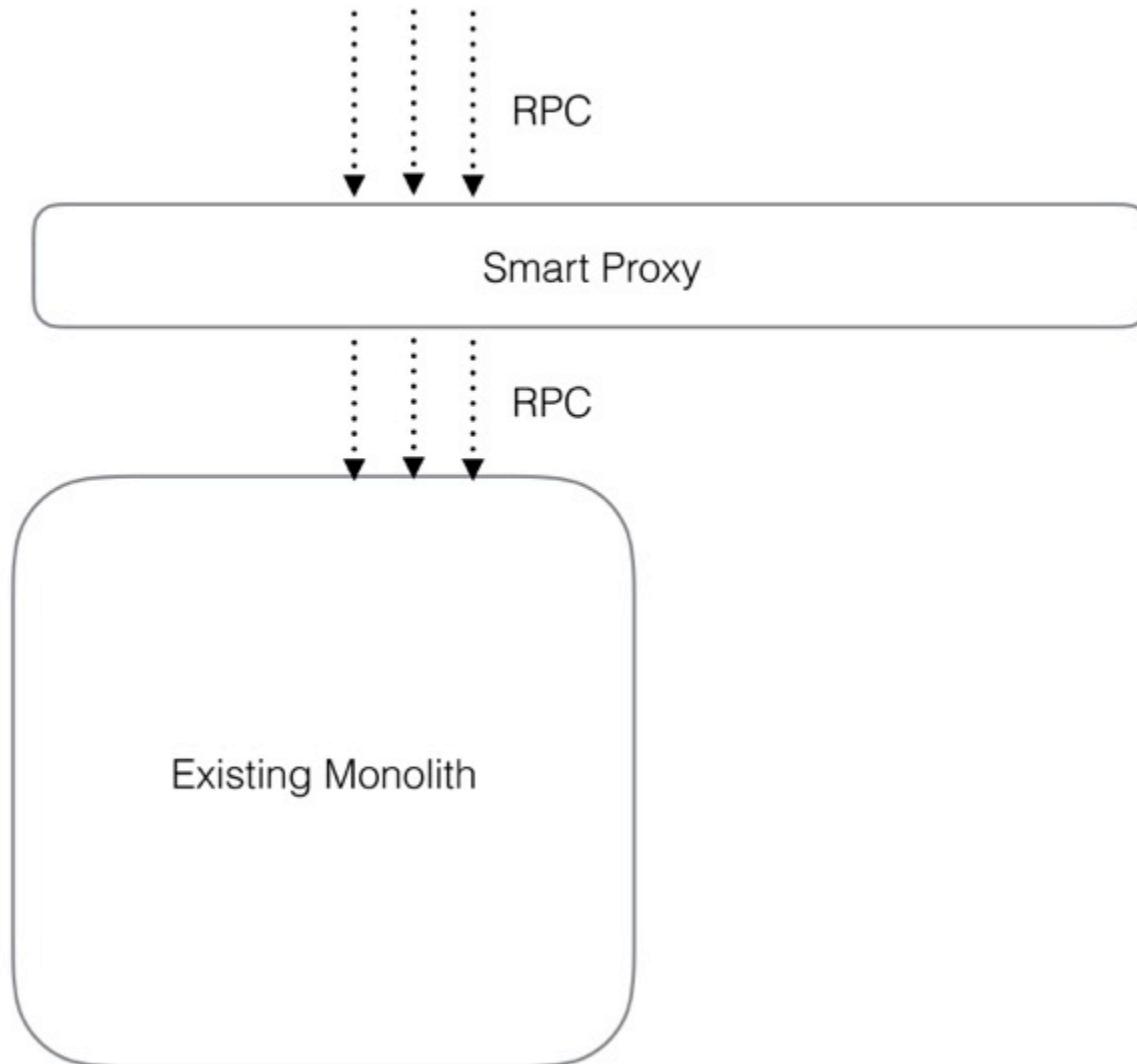
## PROTOCOL CHANGING PROXY



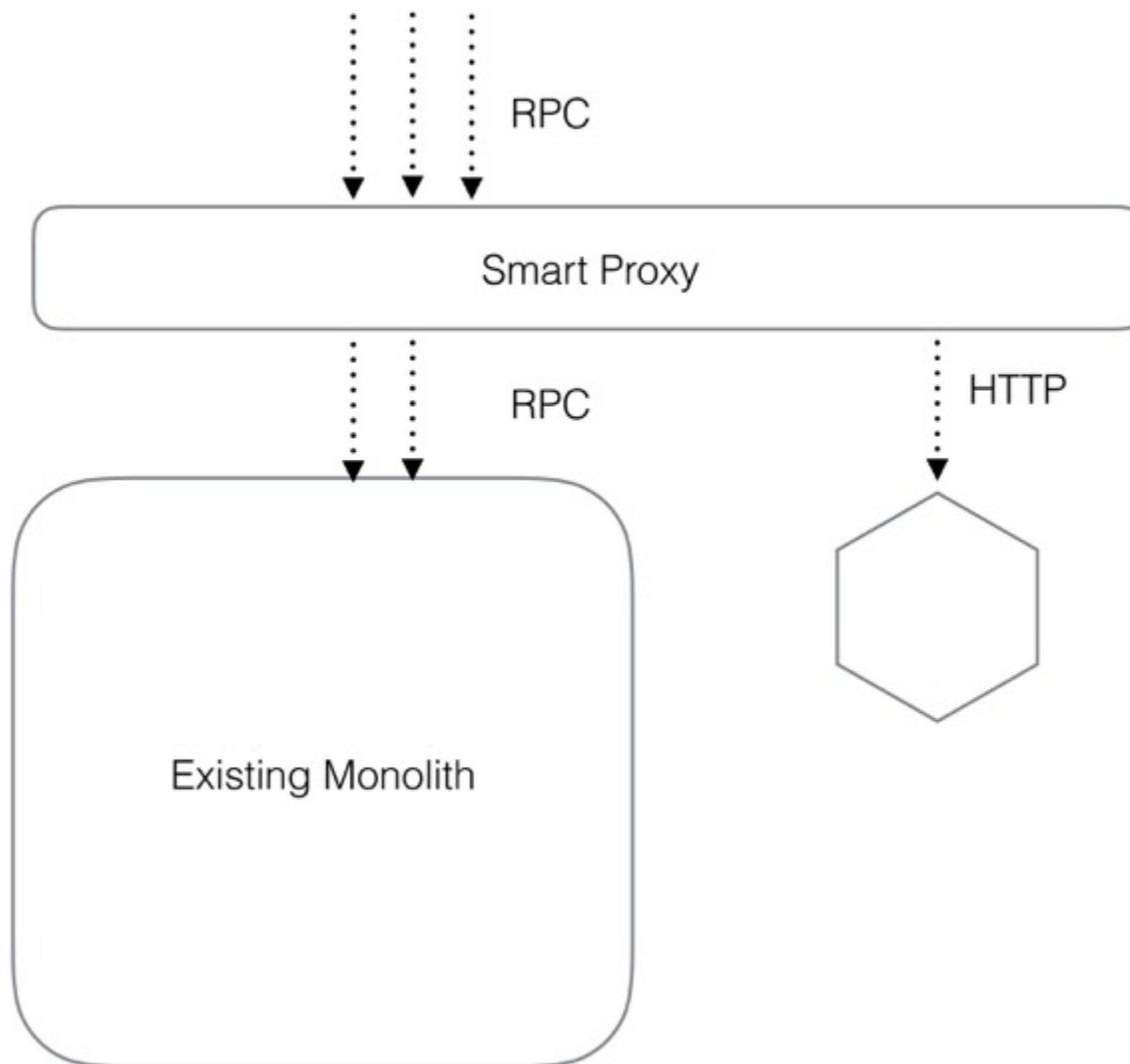
## PROTOCOL CHANGING PROXY



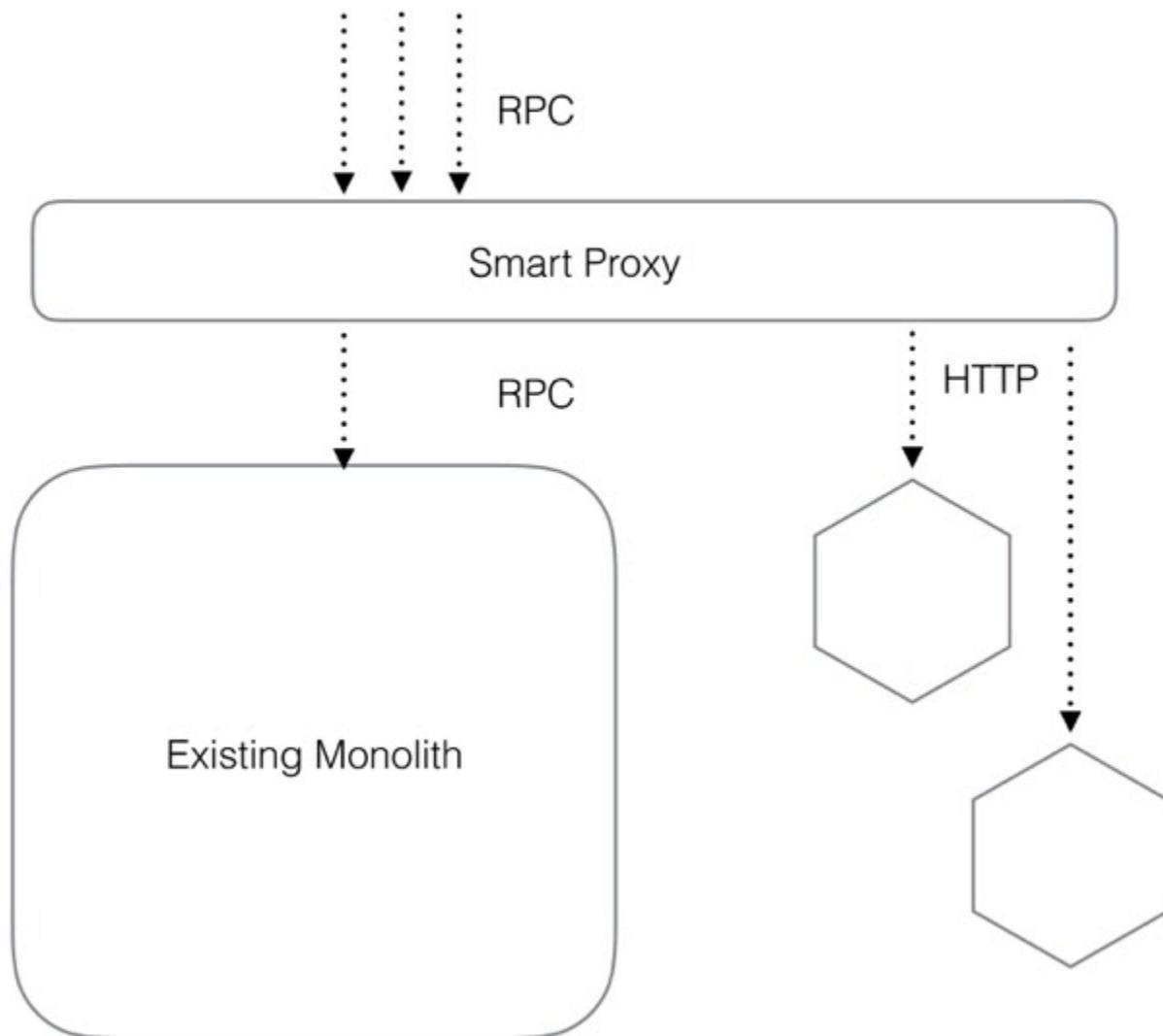
## PROTOCOL CHANGING PROXY



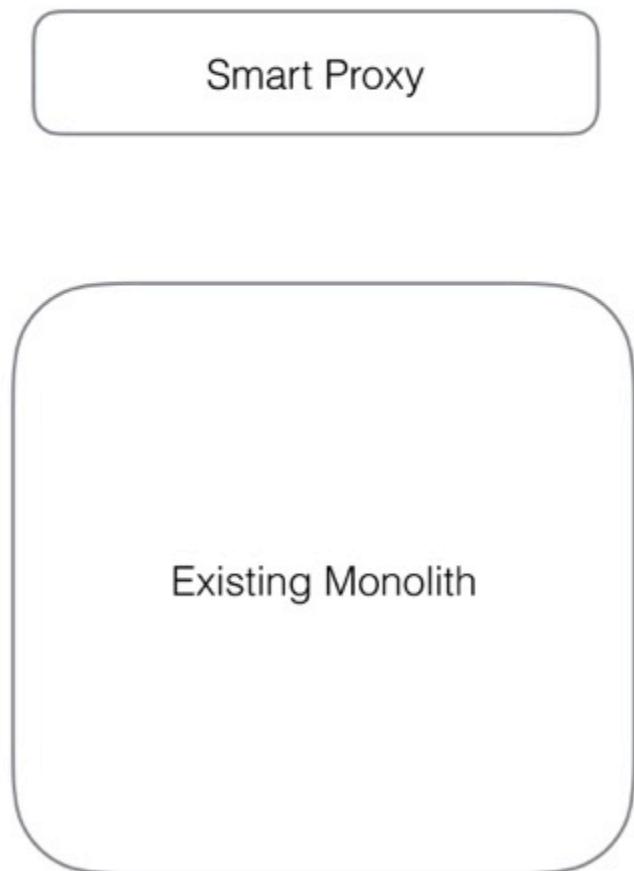
## PROTOCOL CHANGING PROXY



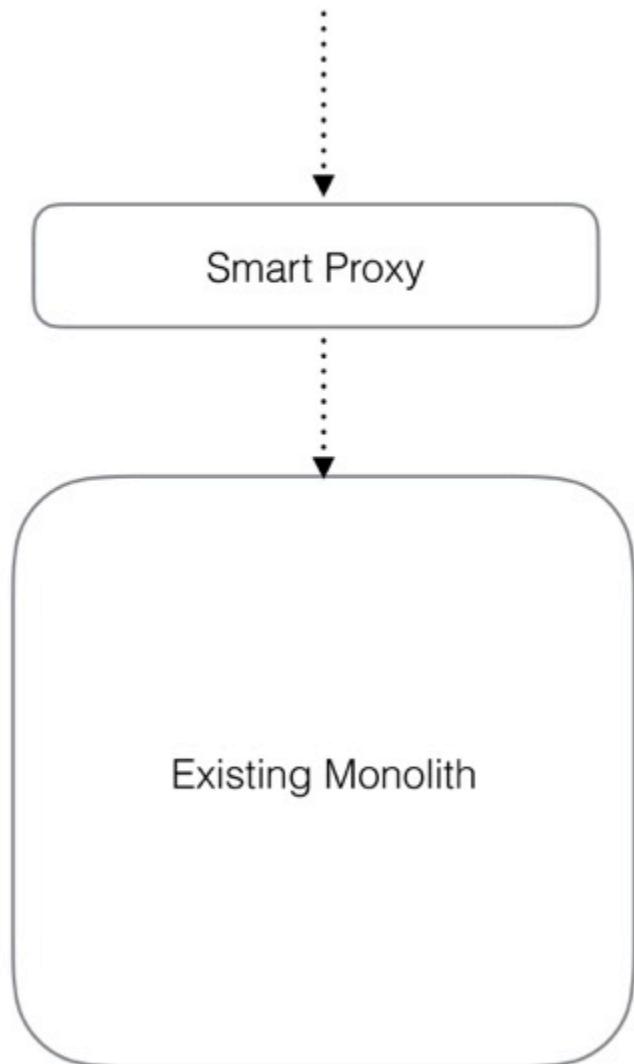
## PROTOCOL CHANGING PROXY



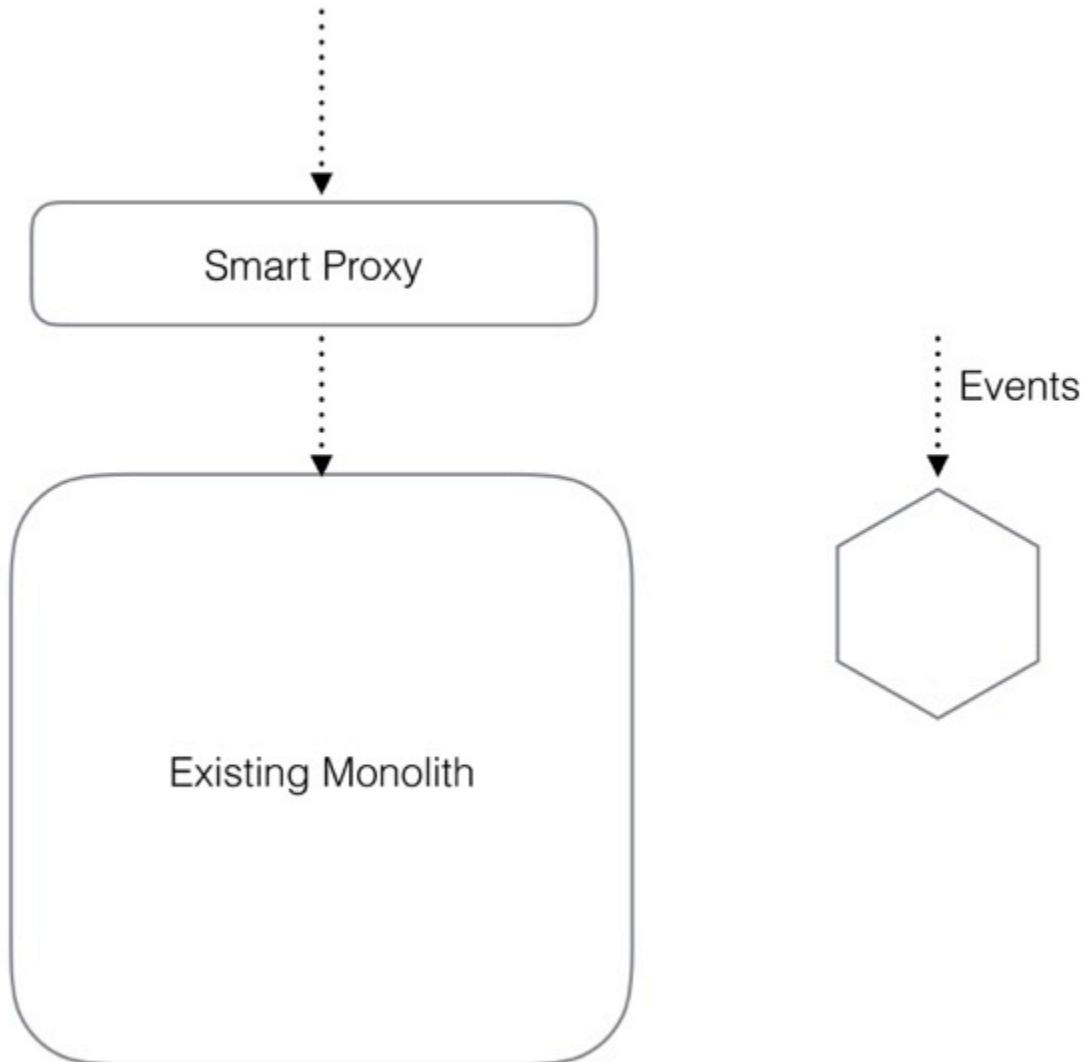
## EVENT DECORATING PROXY



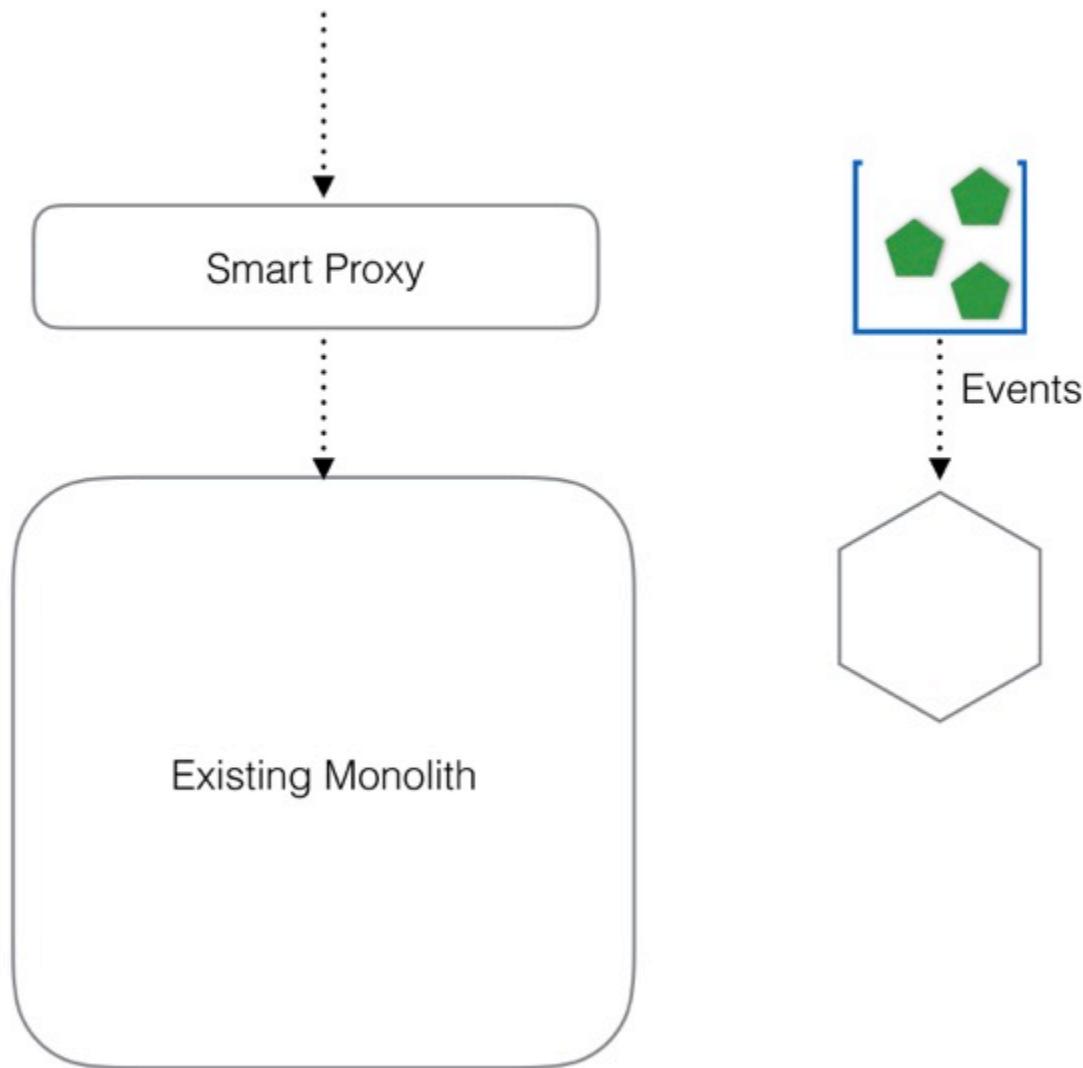
## EVENT DECORATING PROXY



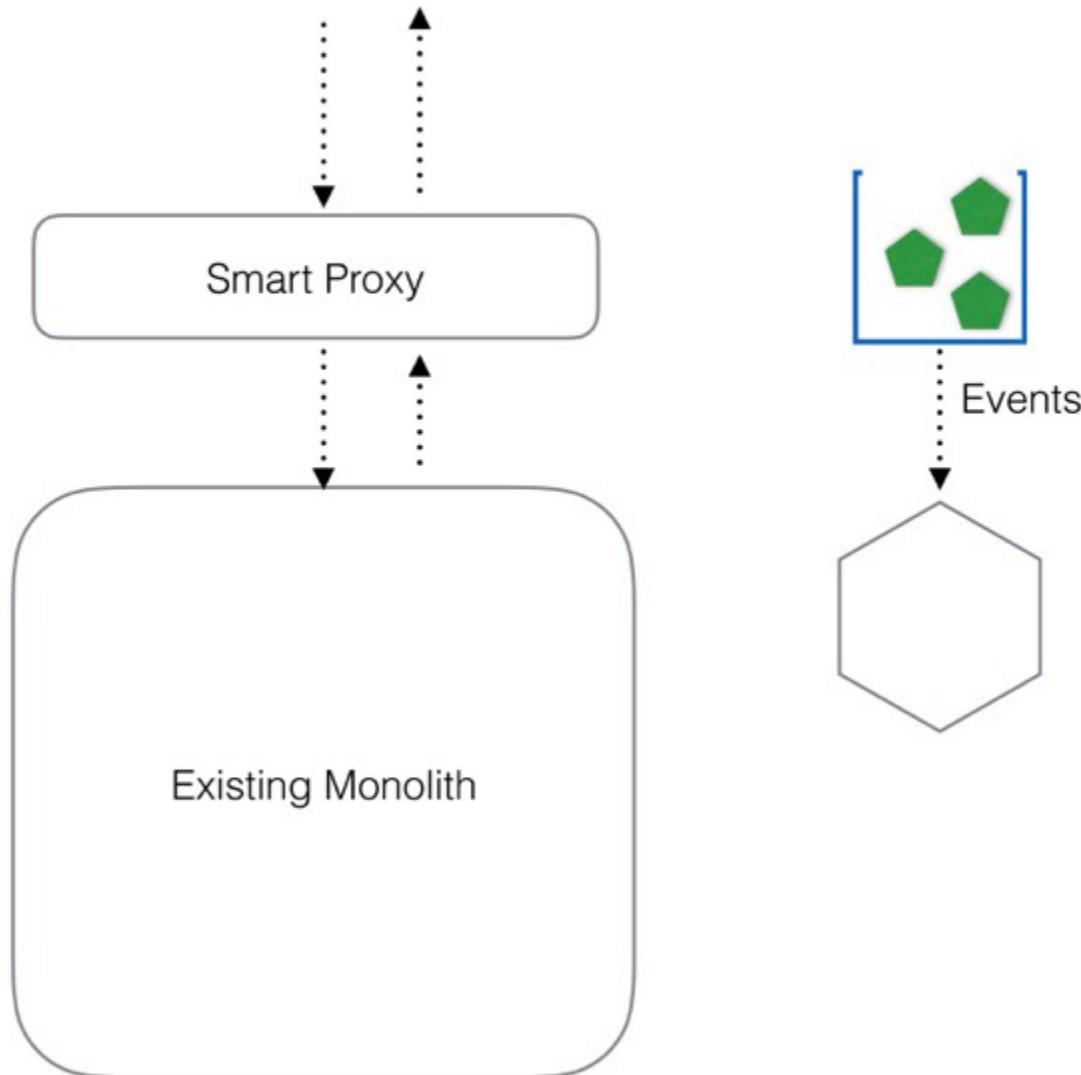
## EVENT DECORATING PROXY



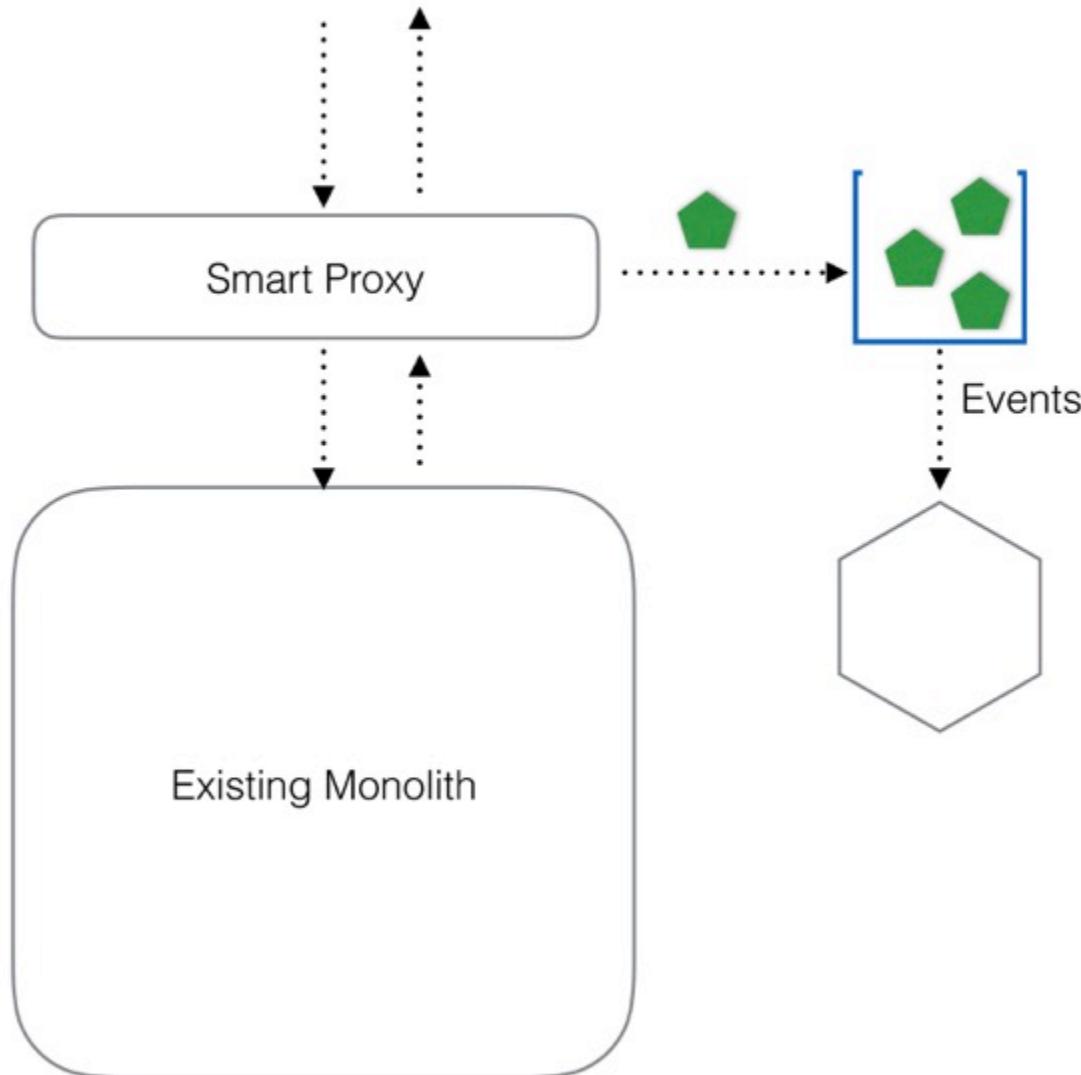
## EVENT DECORATING PROXY



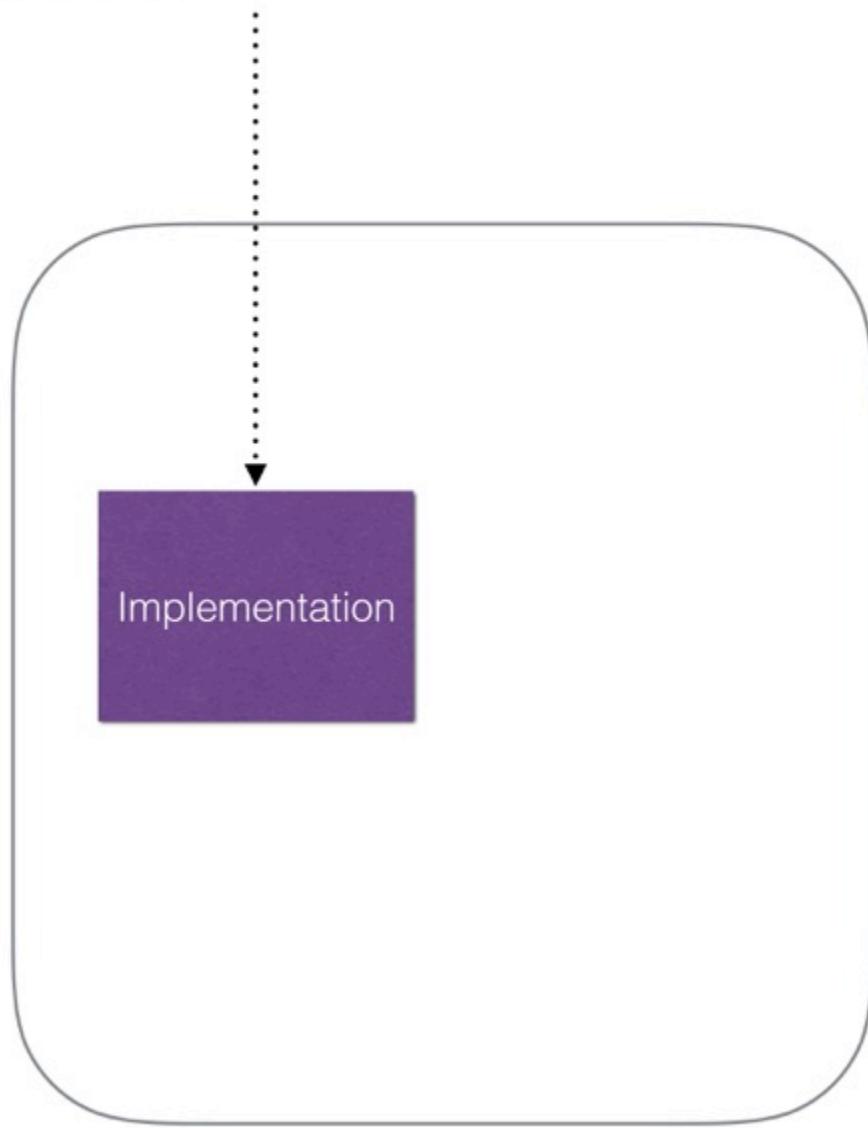
## EVENT DECORATING PROXY



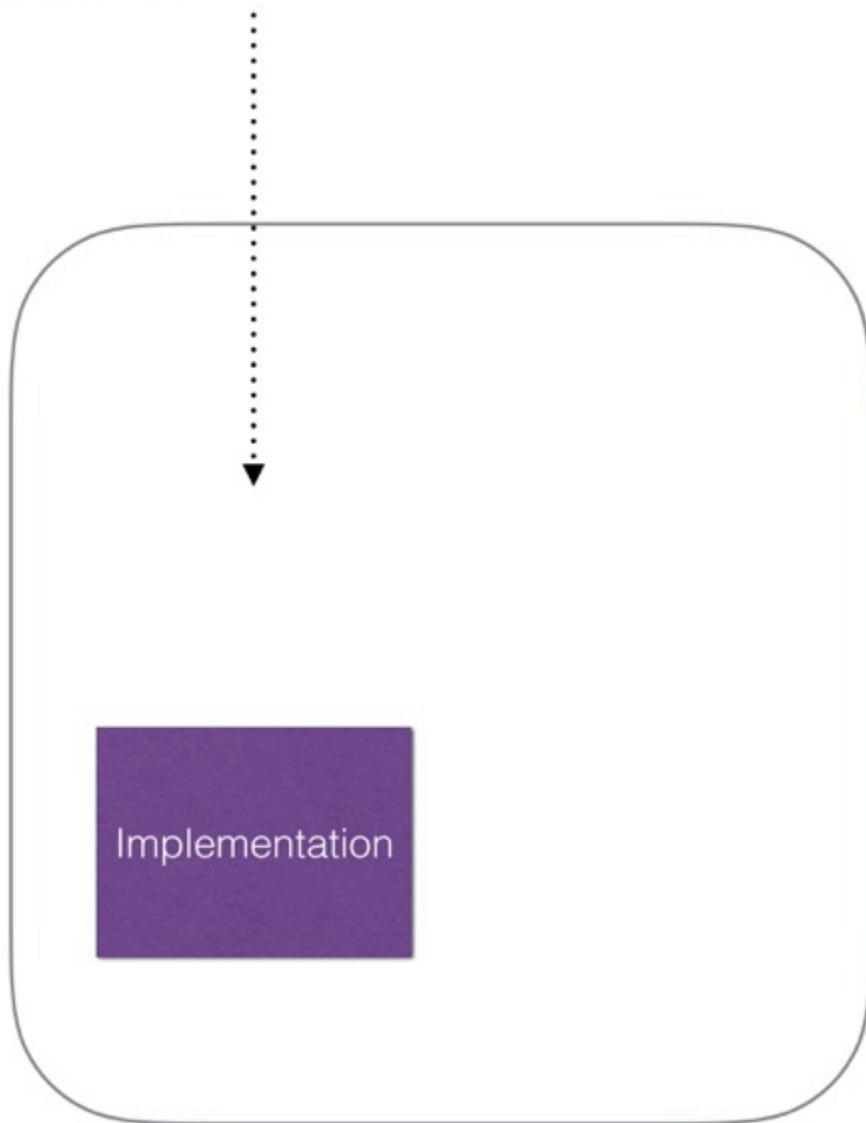
## EVENT DECORATING PROXY



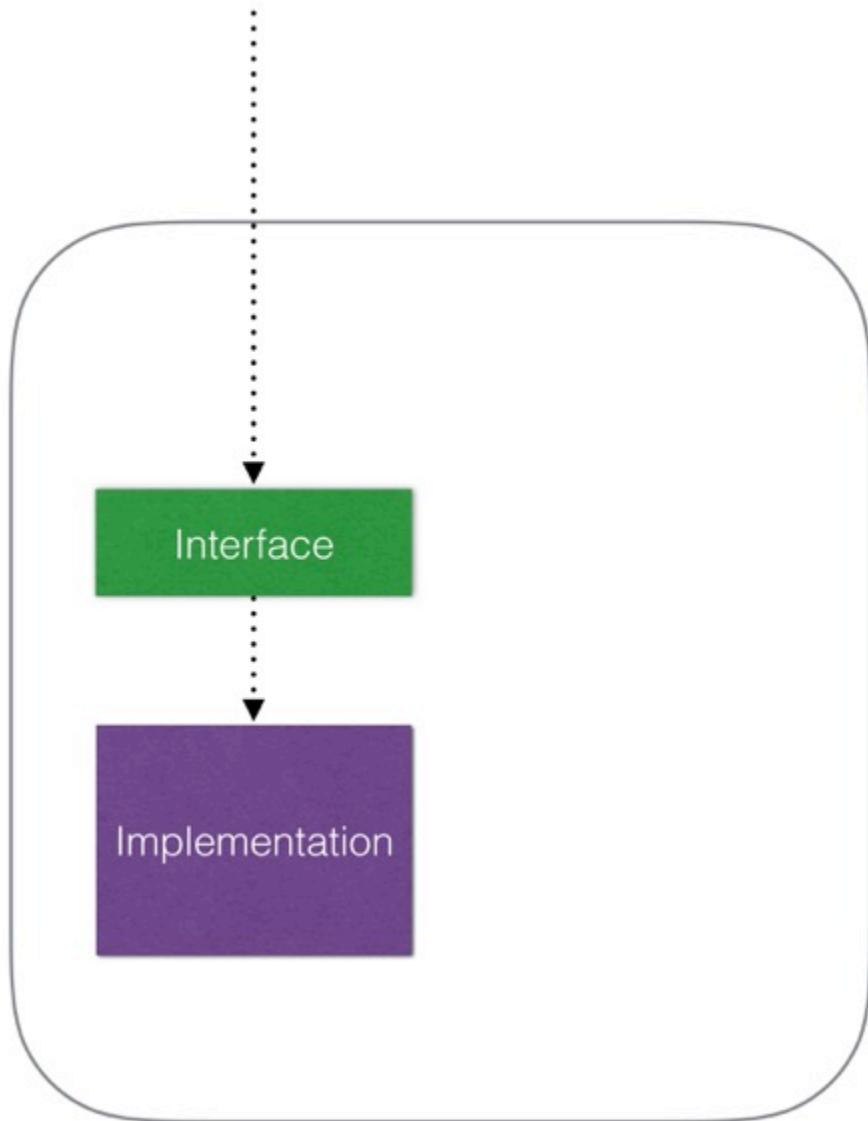
## INTERNAL ABSTRACTION



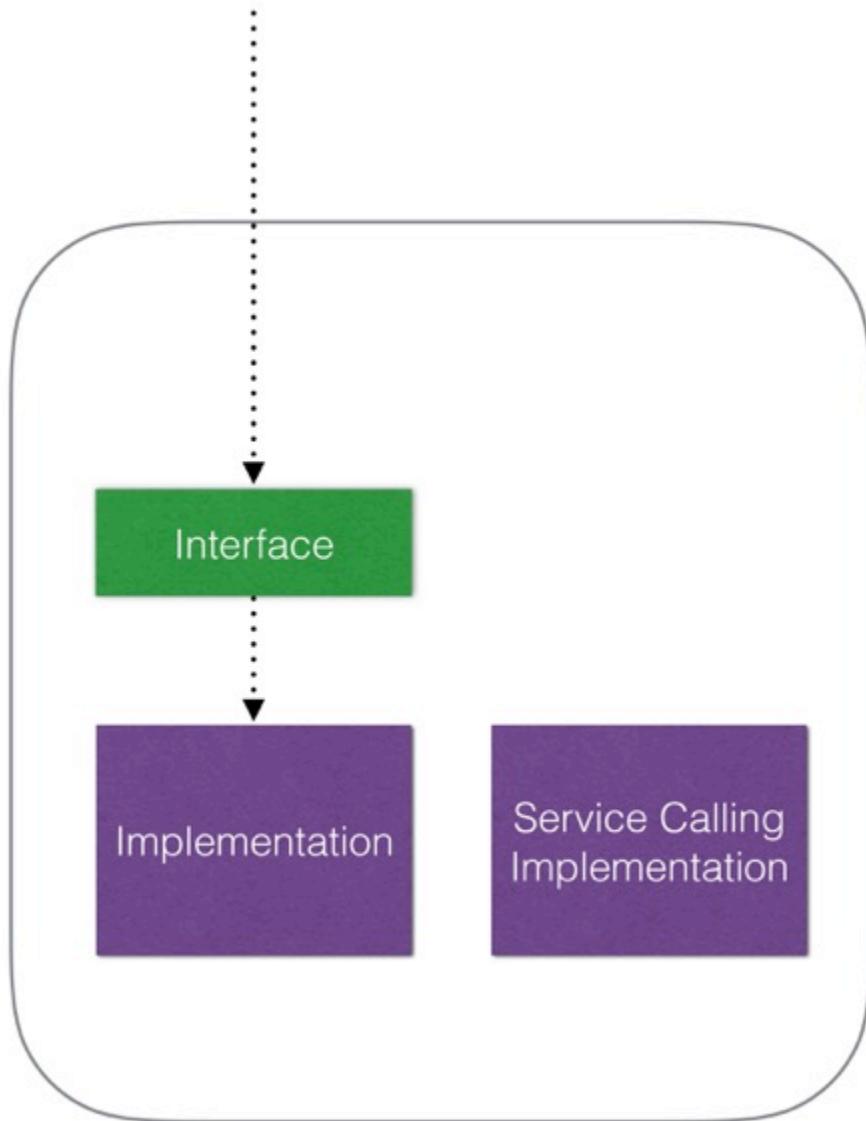
## INTERNAL ABSTRACTION



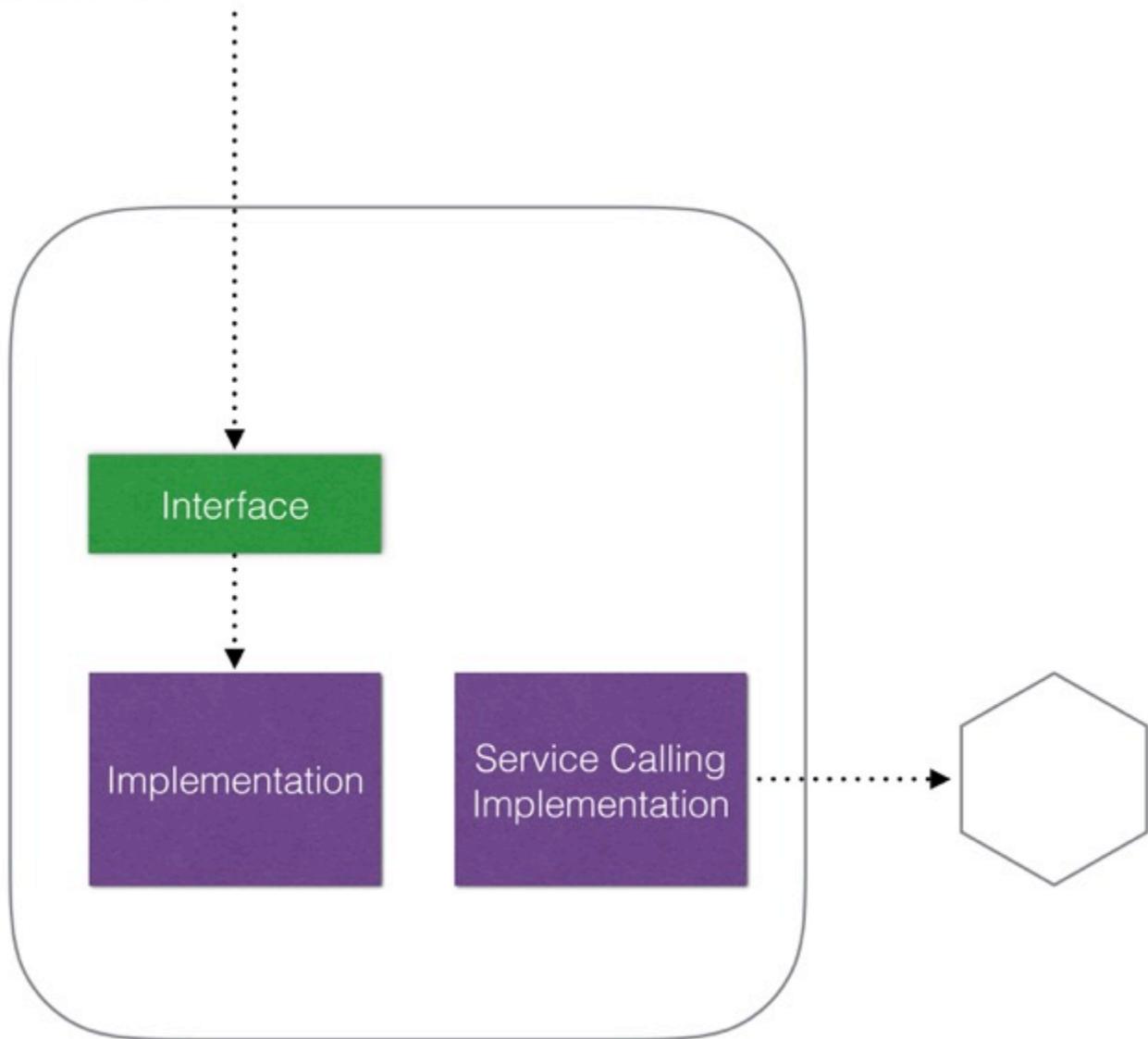
## INTERNAL ABSTRACTION



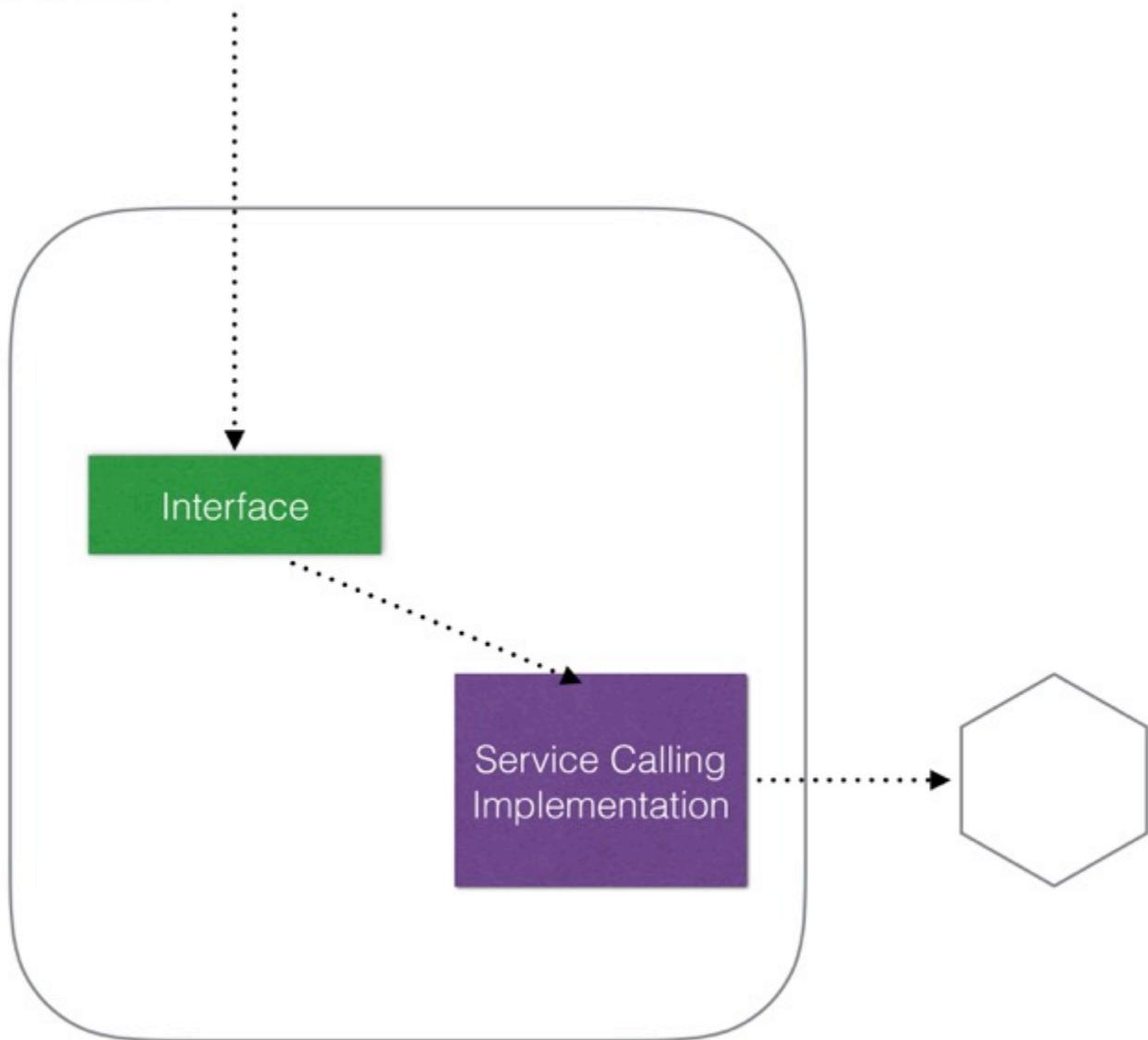
## INTERNAL ABSTRACTION



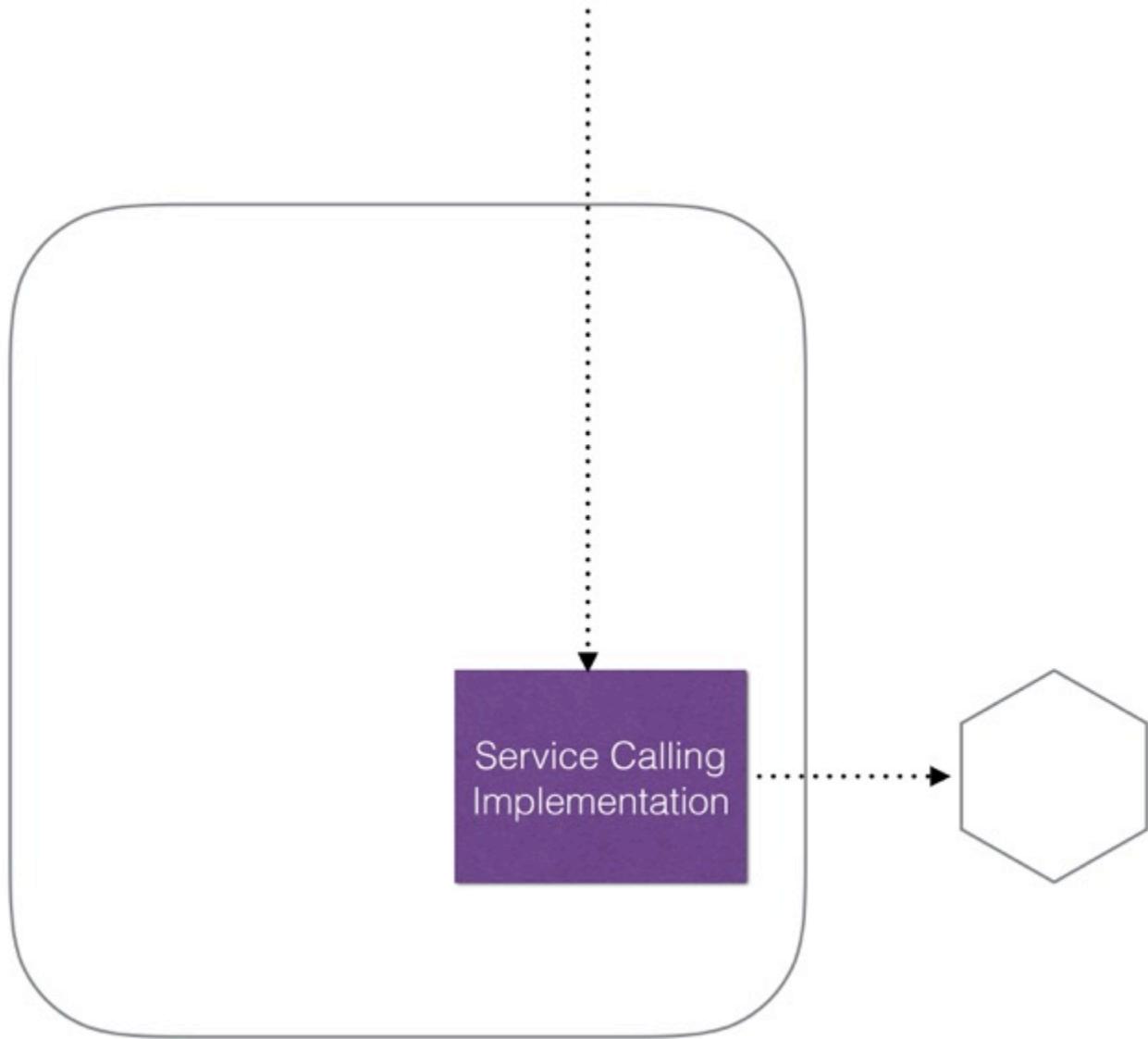
## INTERNAL ABSTRACTION



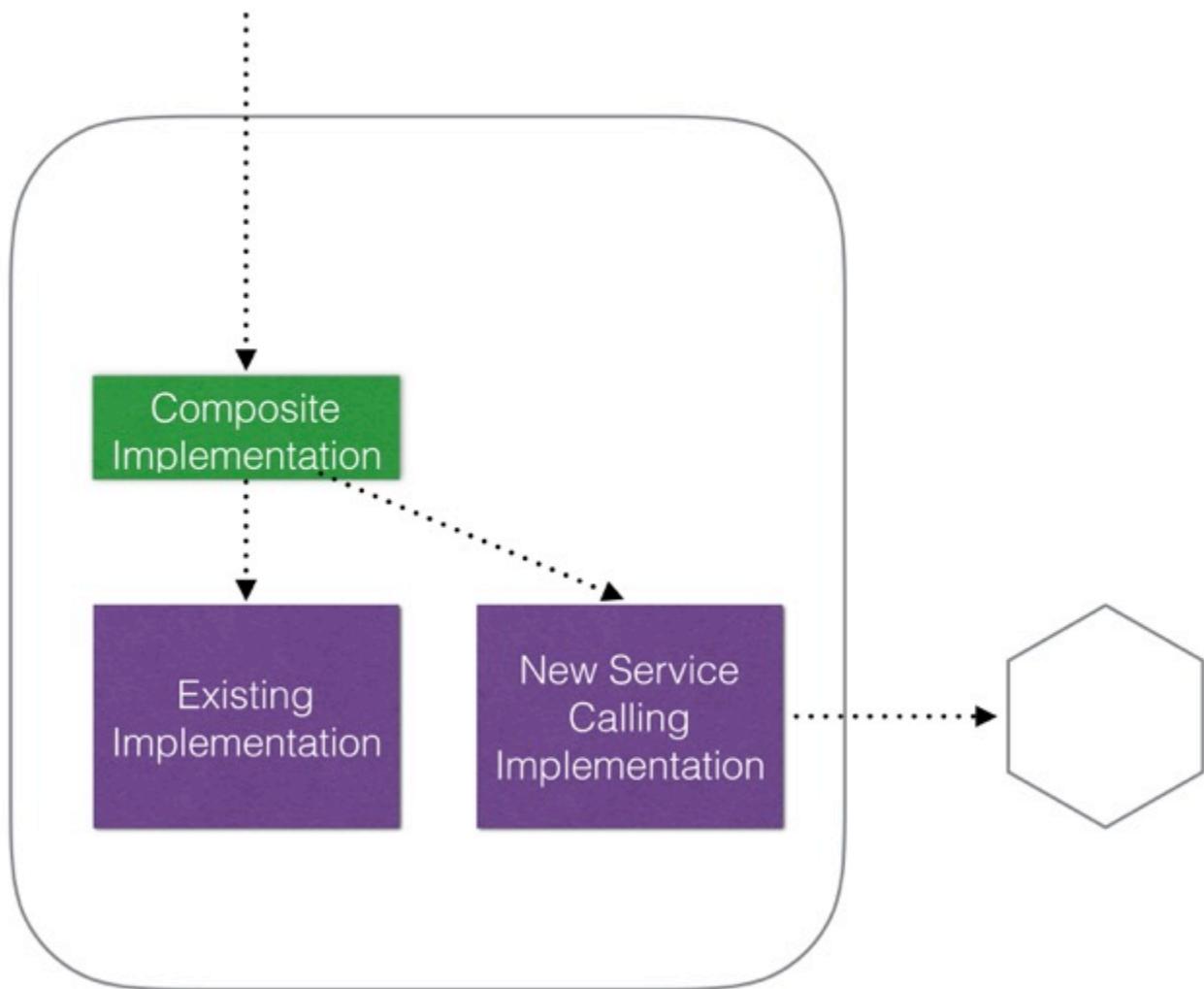
## INTERNAL ABSTRACTION



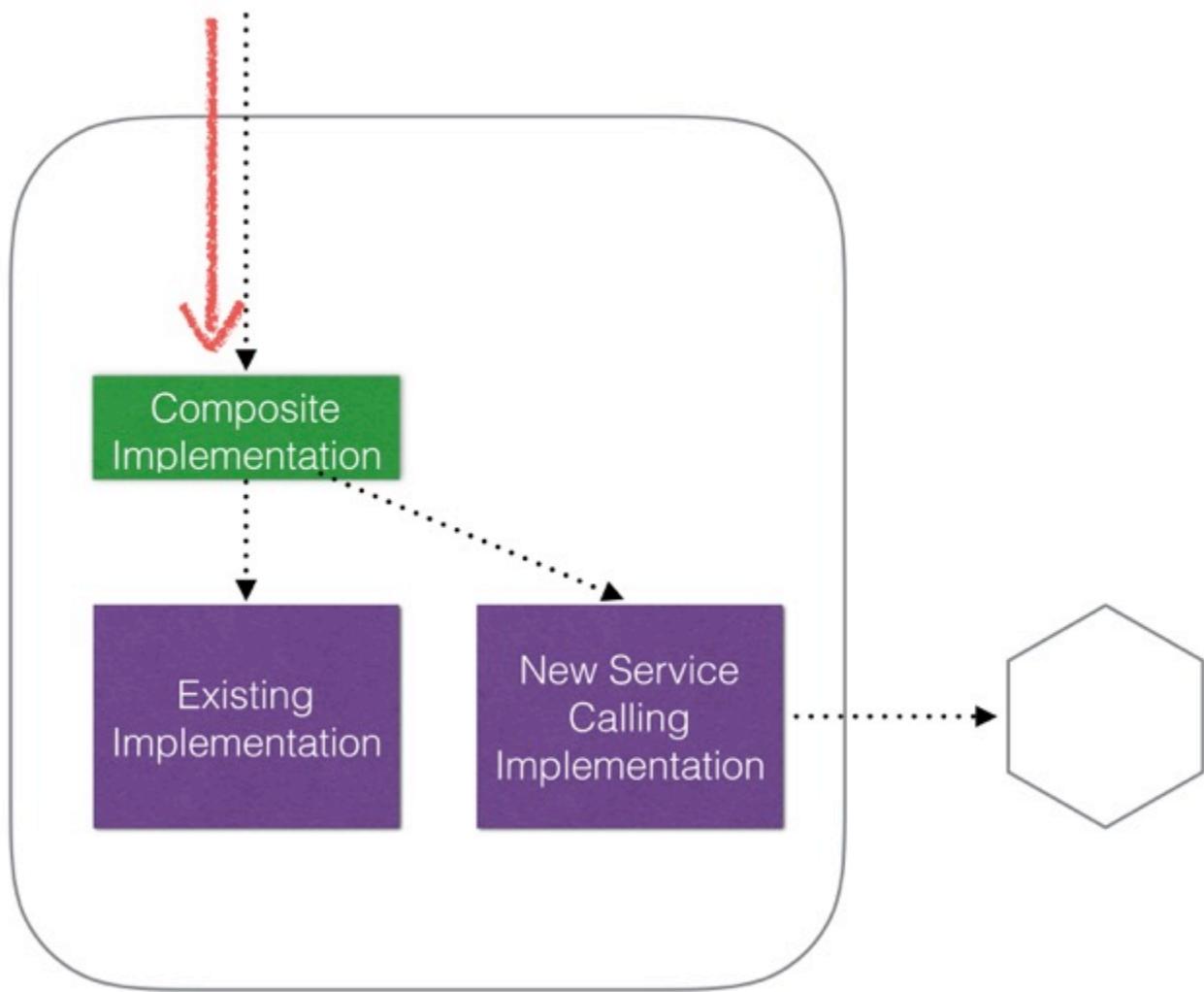
## INTERNAL ABSTRACTION



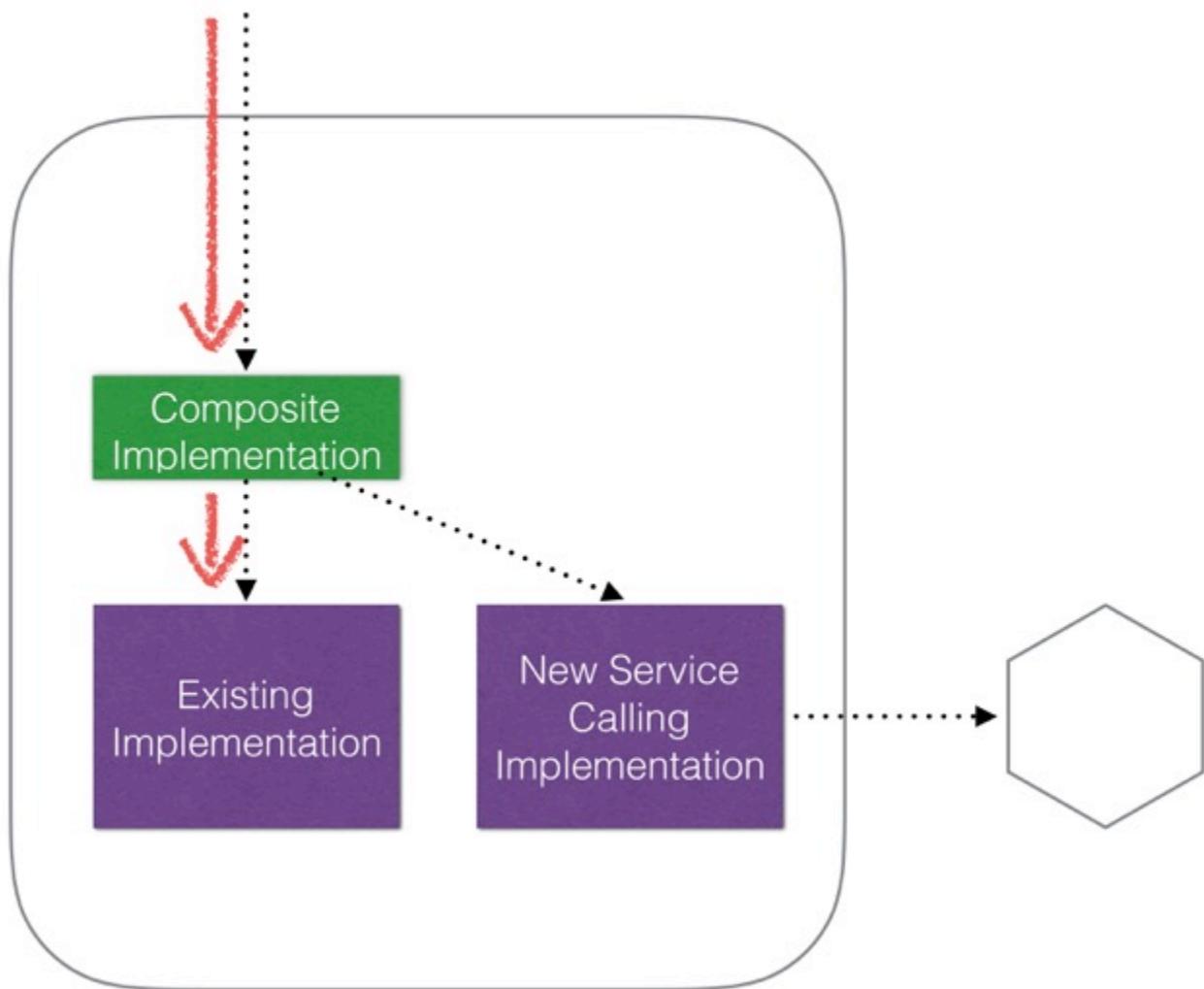
## CO-EXISTING INTERNAL ABSTRACTIONS



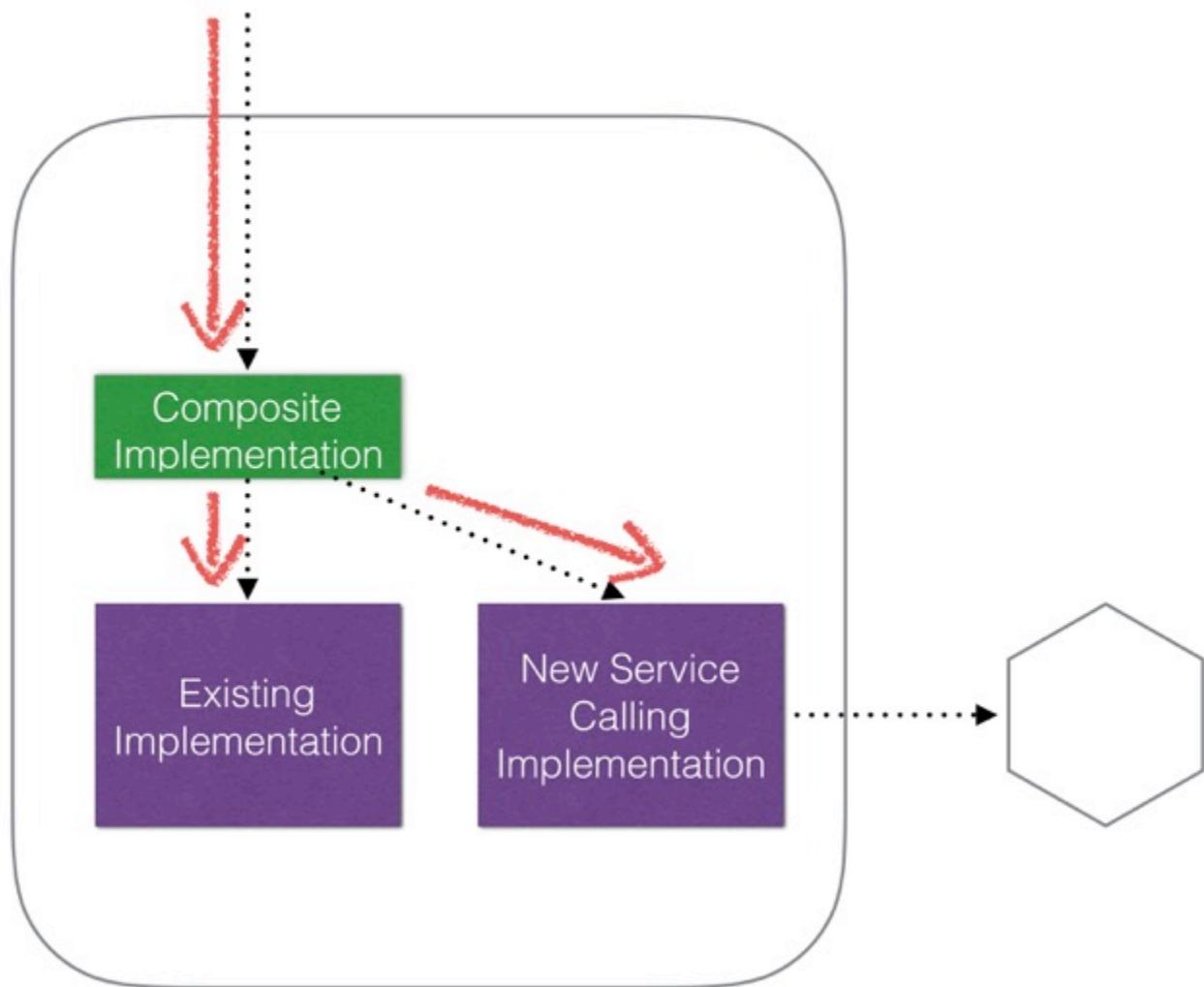
## CO-EXISTING INTERNAL ABSTRACTIONS



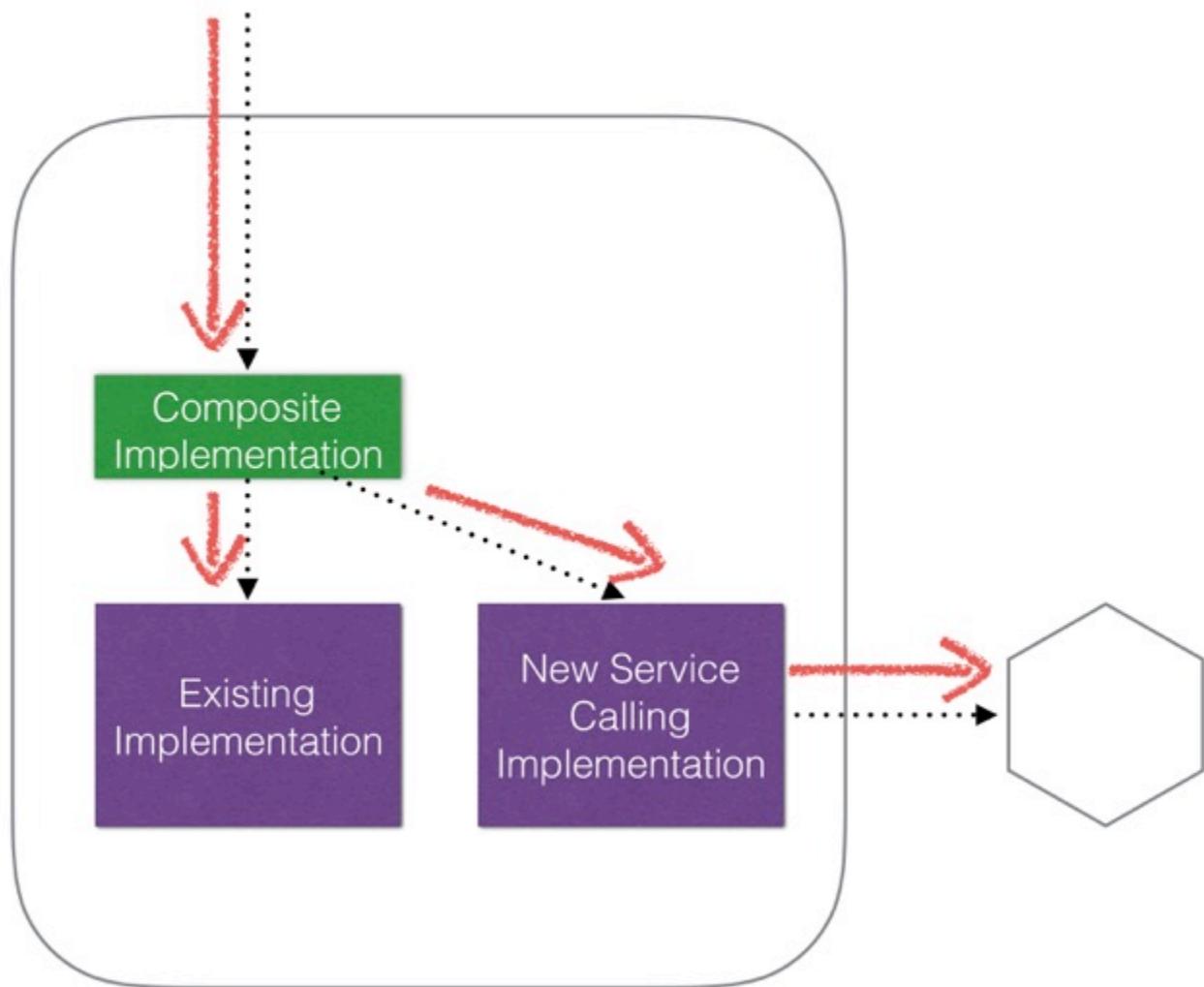
## CO-EXISTING INTERNAL ABSTRACTIONS



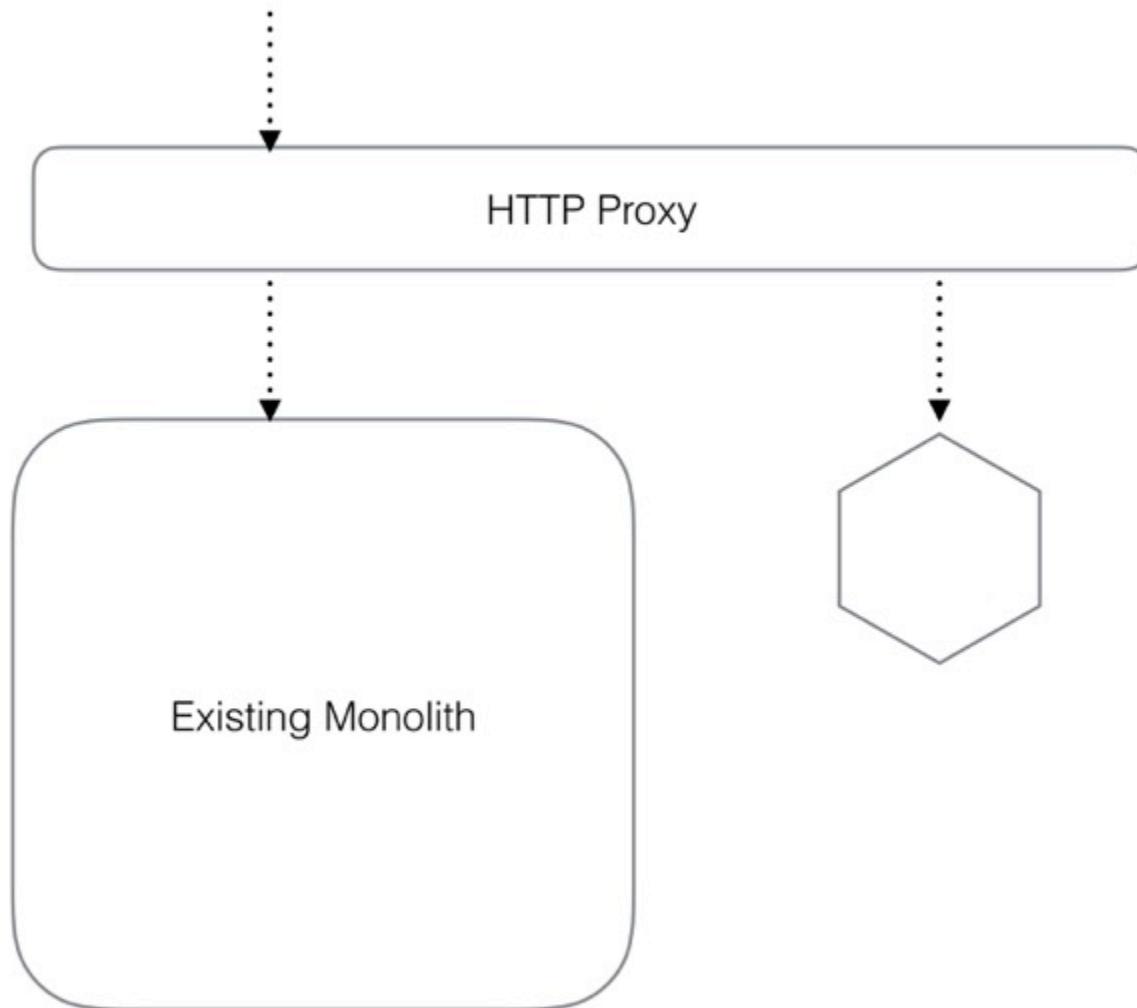
## CO-EXISTING INTERNAL ABSTRACTIONS



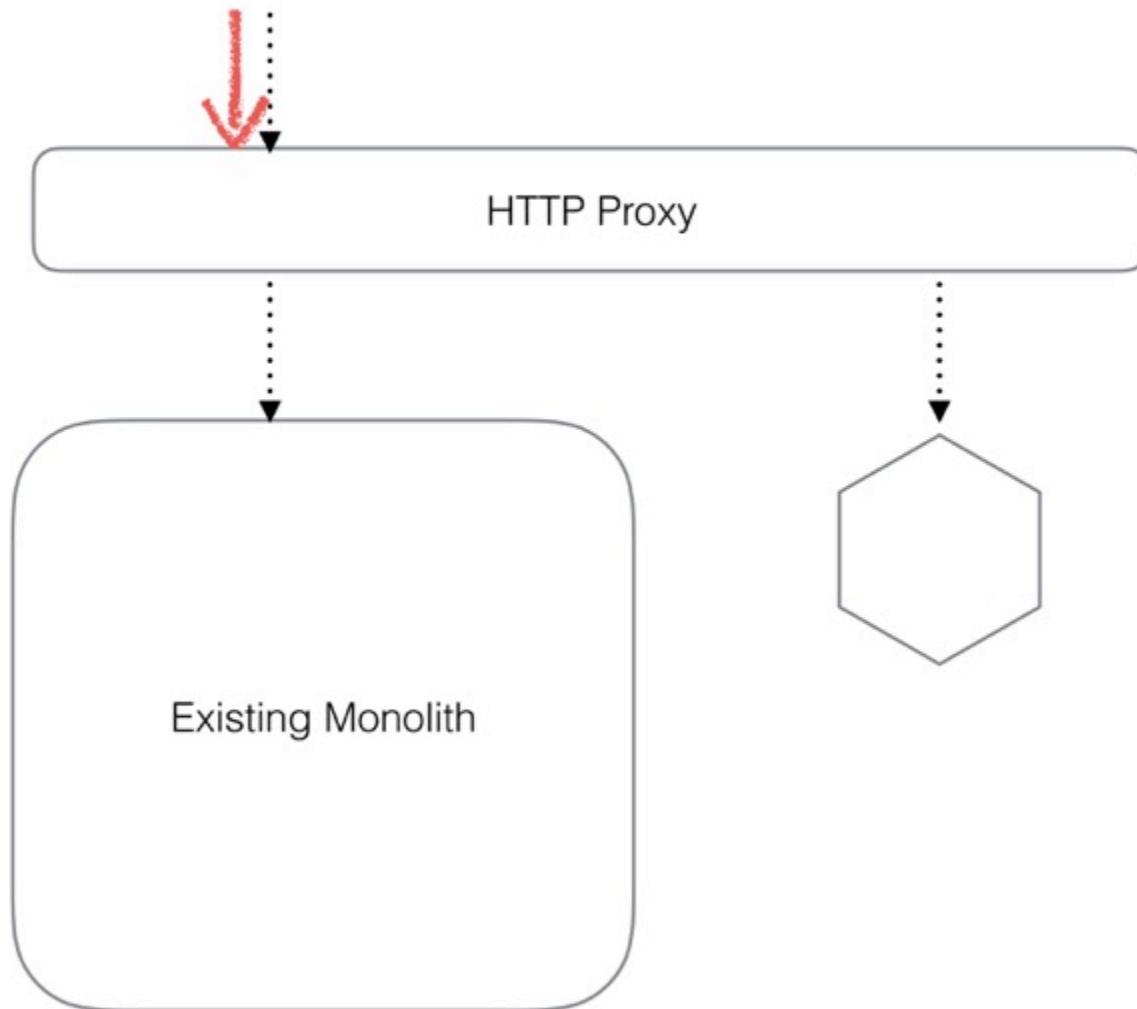
## CO-EXISTING INTERNAL ABSTRACTIONS



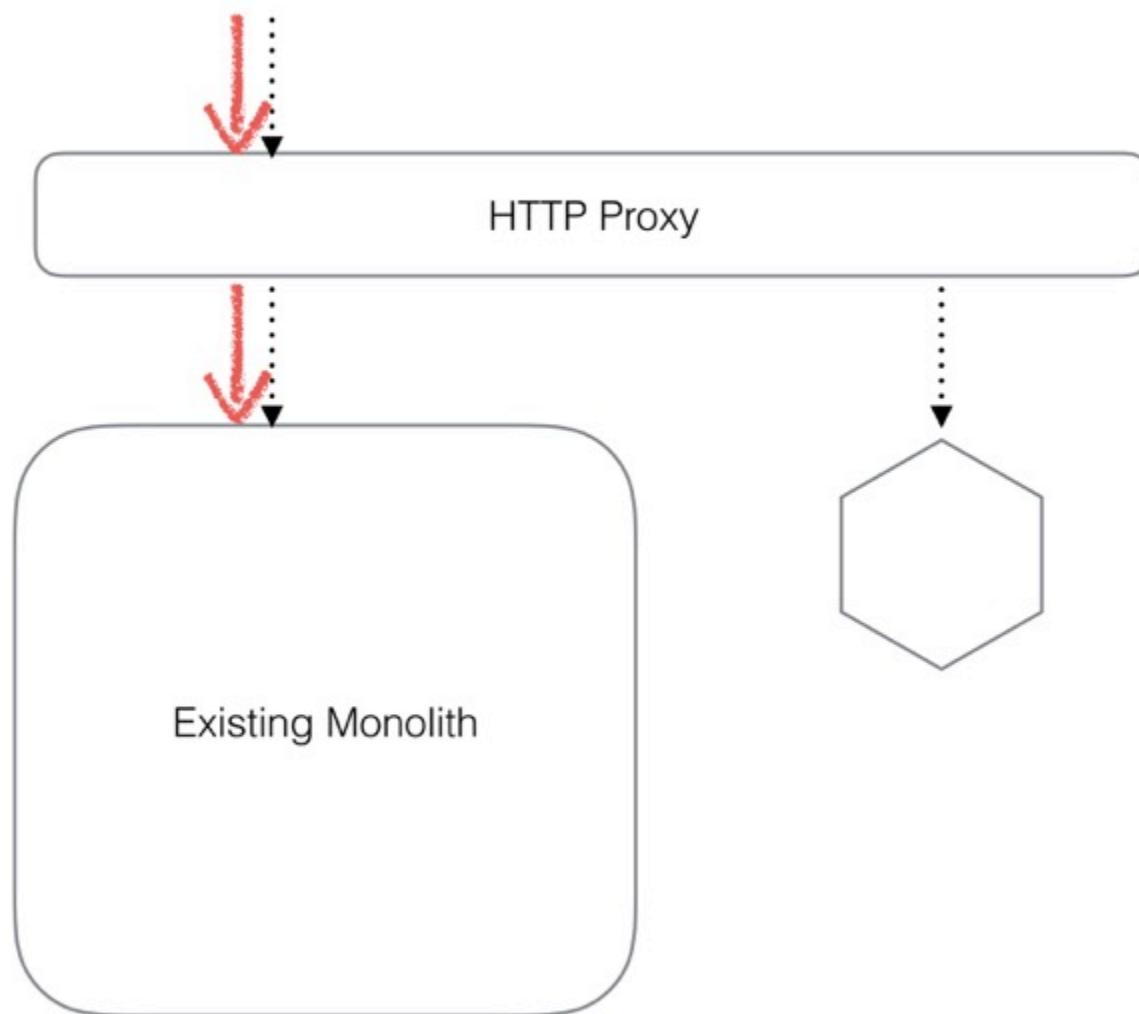
## CO-EXISTING PROXY IMPLEMENTATIONS



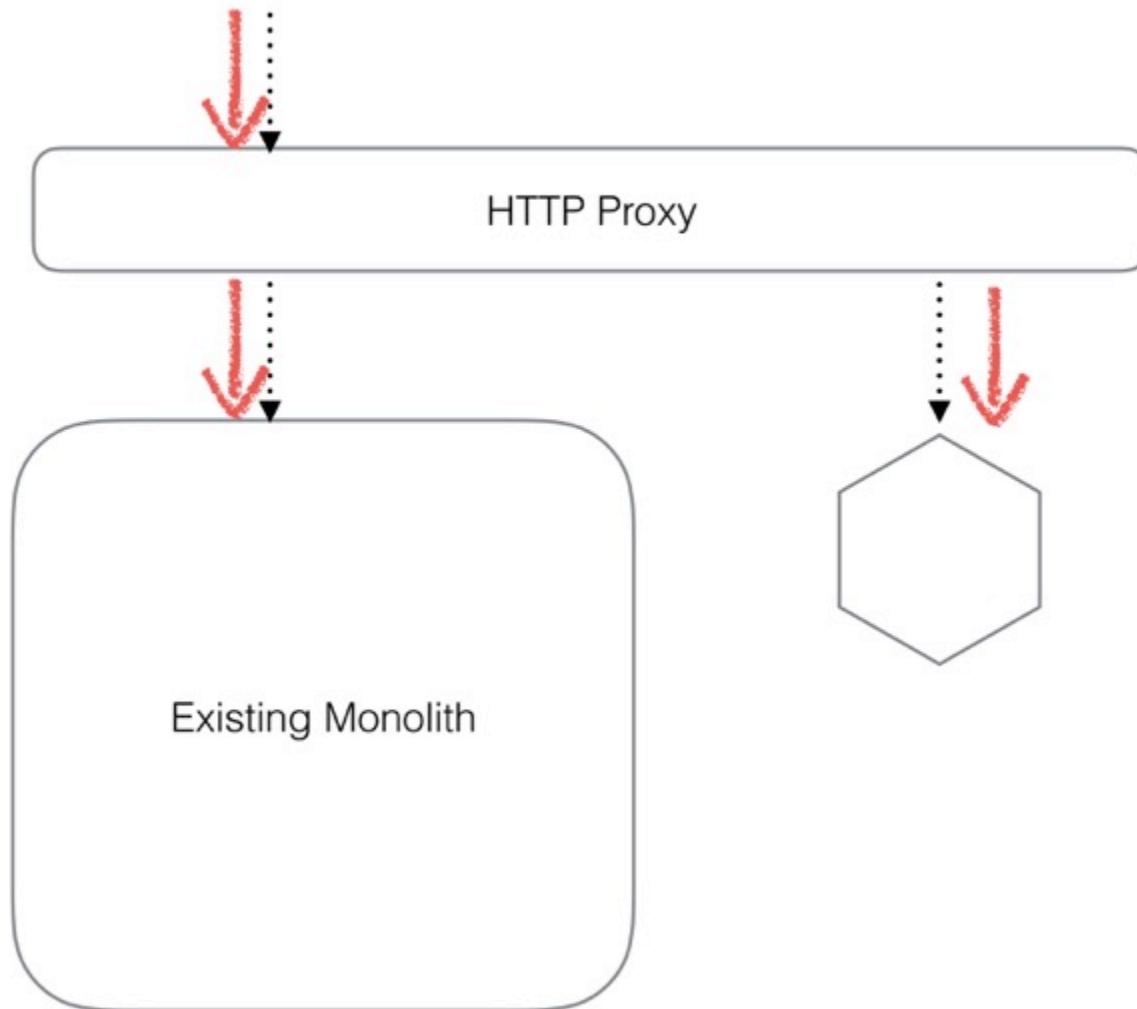
## CO-EXISTING PROXY IMPLEMENTATIONS



## CO-EXISTING PROXY IMPLEMENTATIONS



## CO-EXISTING PROXY IMPLEMENTATIONS



Call both implementations & compare

Divert proportion of traffic to test new implementation (canary release)

## CANARY RELEASING

### How release canaries can save your bacon - CRE life lessons

Friday, March 31, 2017

By Adrian Hilton, Customer Reliability Engineer

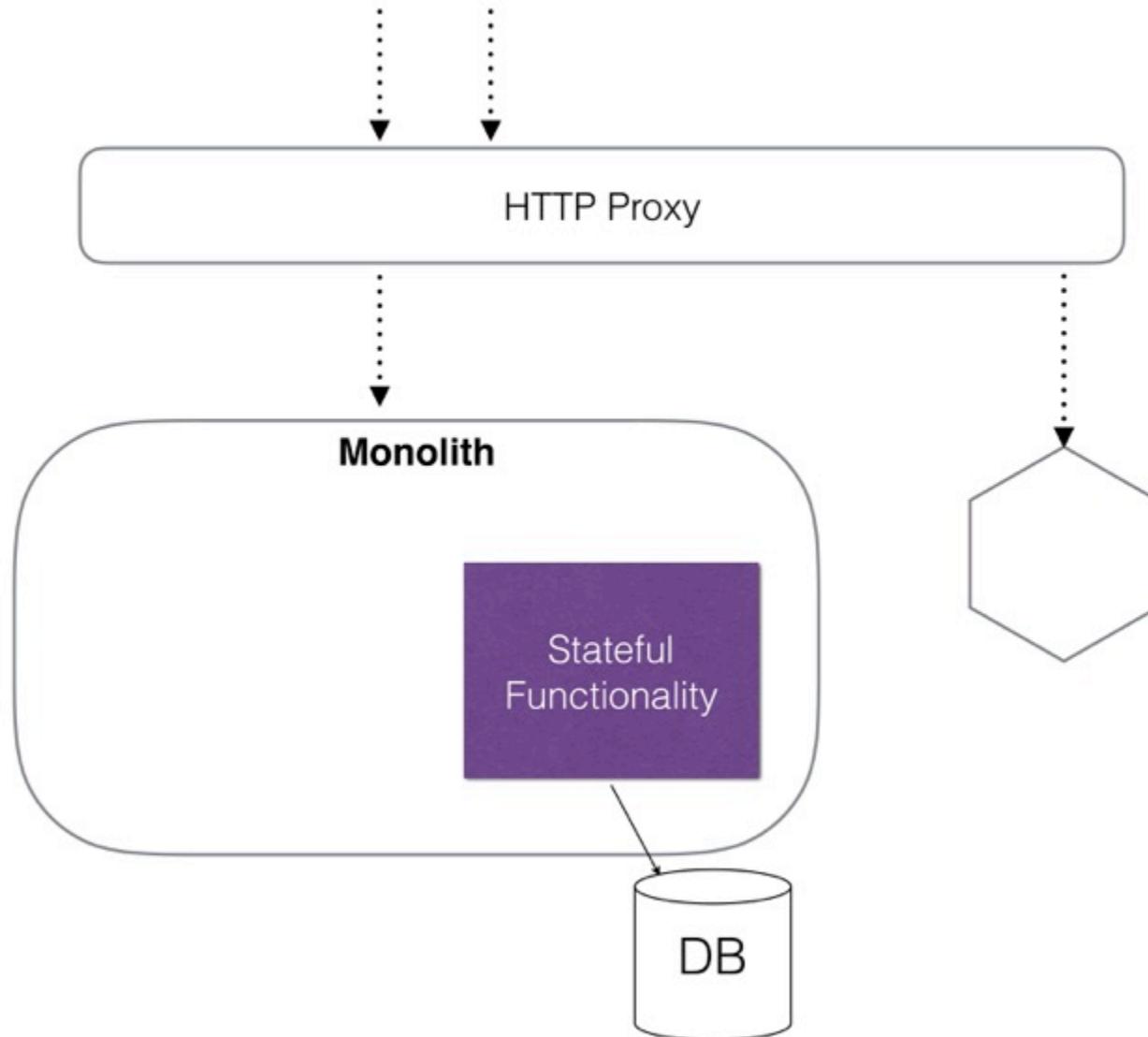
The first part of any reliable software release is being able to roll back if something goes wrong; we discussed how we do this at Google in last week's post, [Reliable releases and rollbacks](#). Once you have that under your belt, you'll want to understand how to detect that things are starting to go wrong in the first place, with canarying.



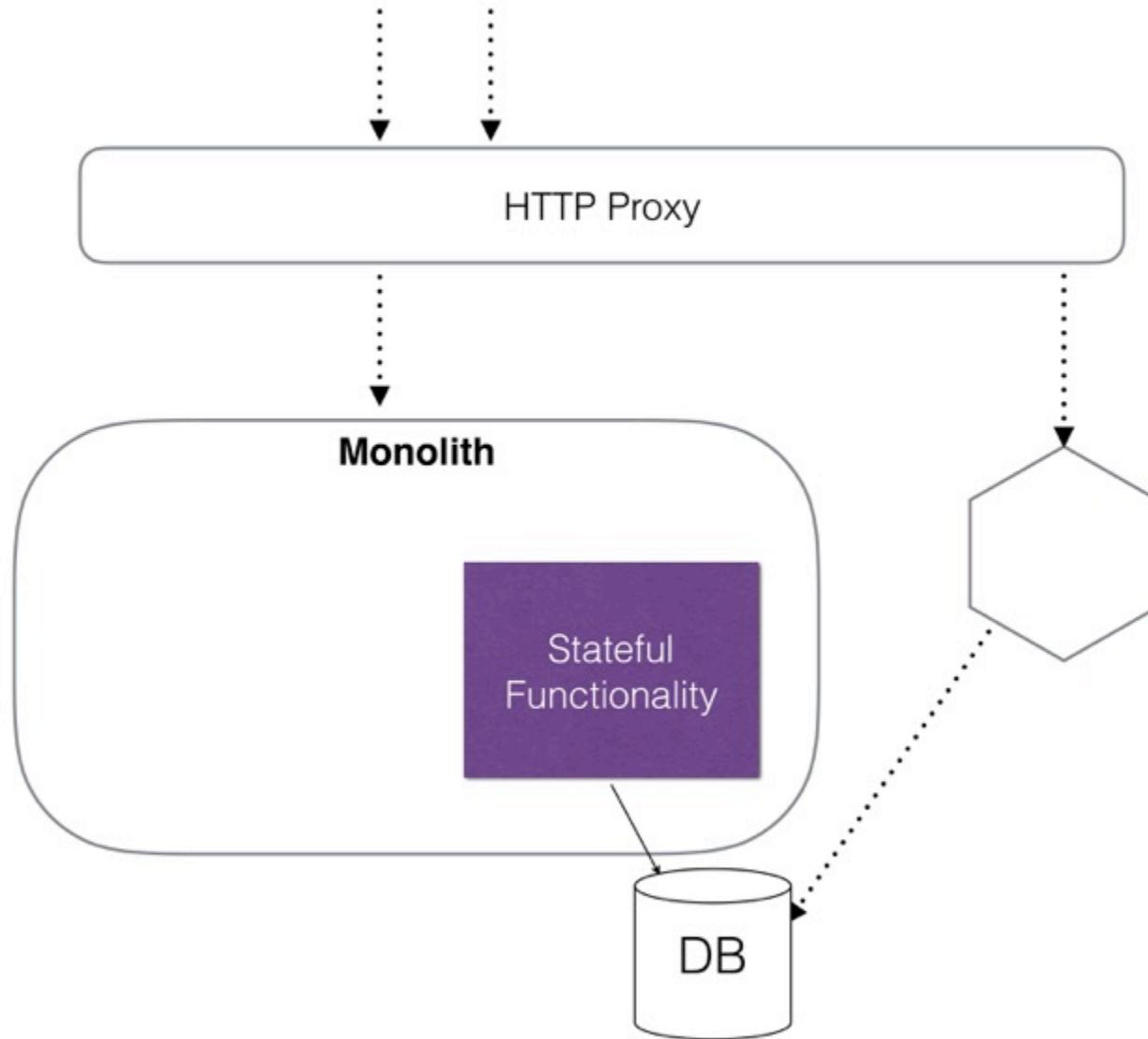
<https://cloudplatform.googleblog.com/2017/03/how-release-canaries-can-save-your-bacon-CRE-life-lessons.html>

# Err...Databases?

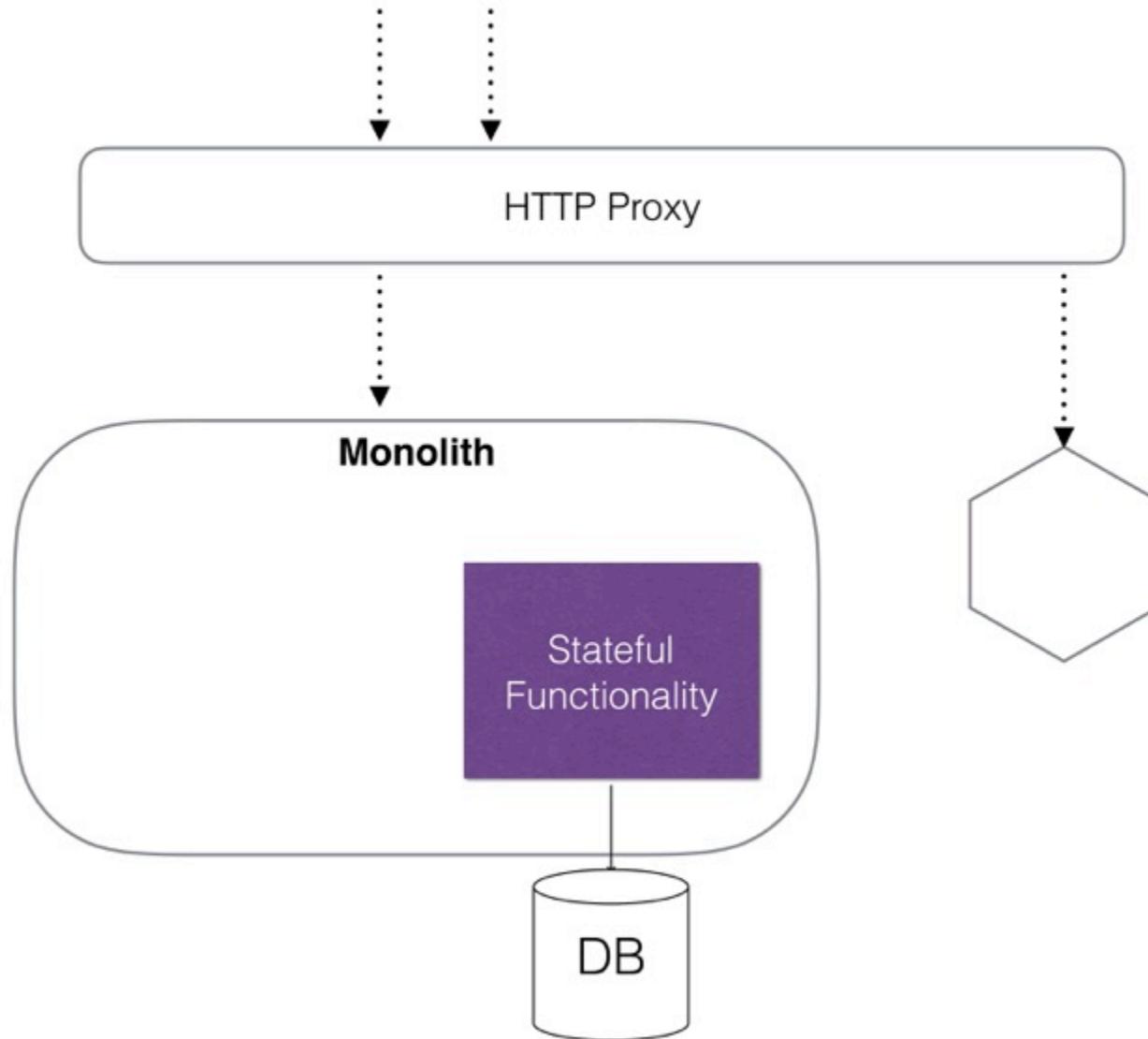
## TEMPORARY REUSE



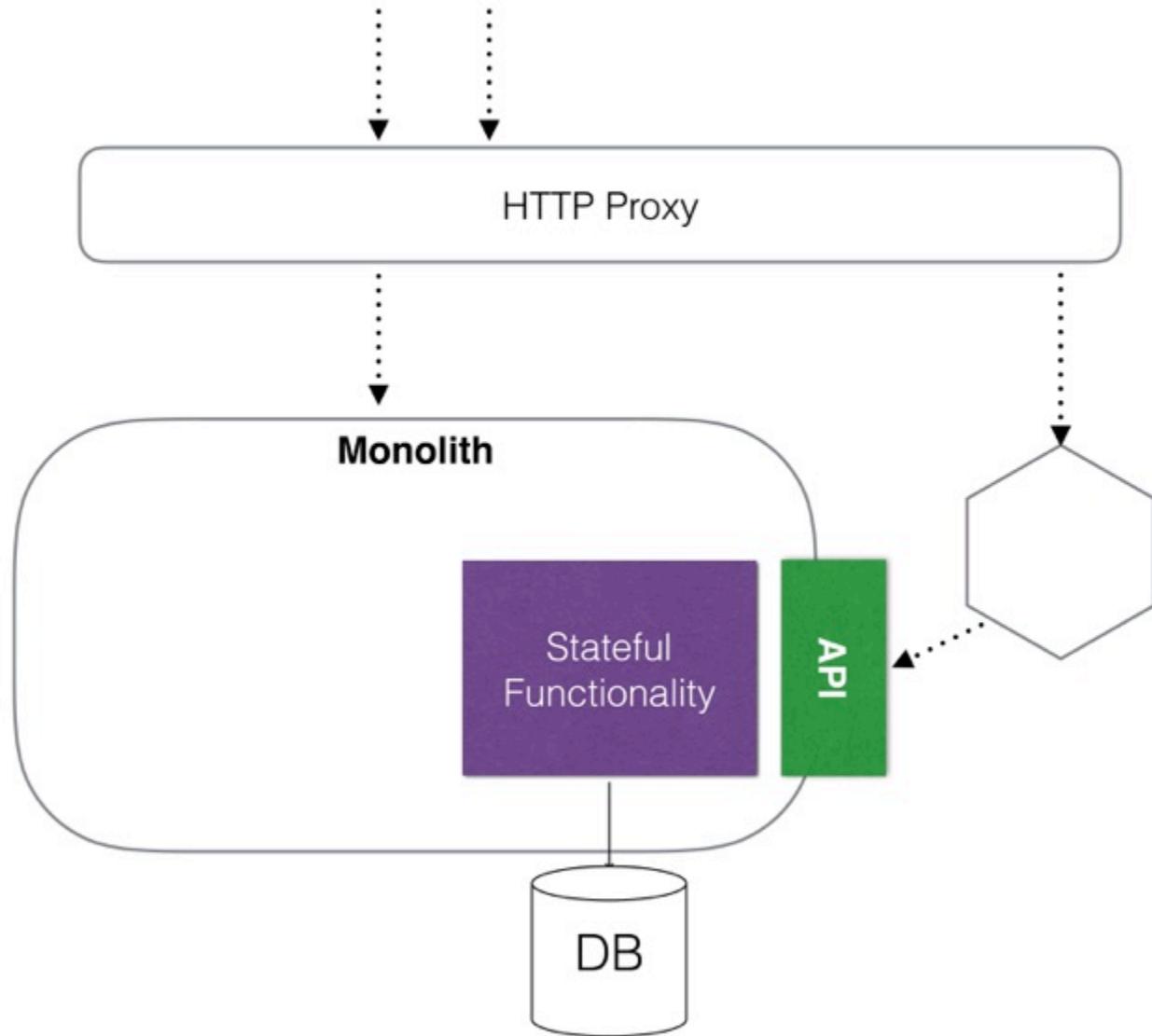
## TEMPORARY REUSE

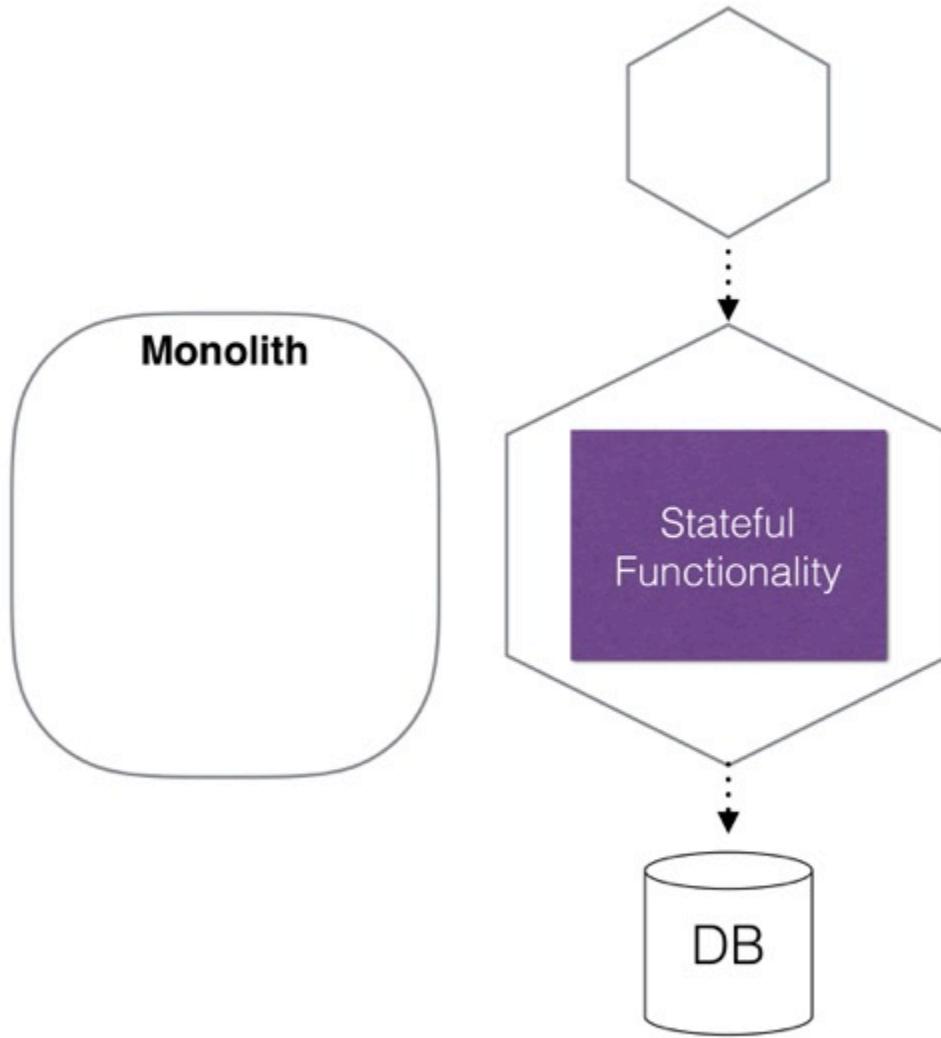


## REUSING STATEFUL FUNCTIONALITY



## REUSING STATEFUL FUNCTIONALITY



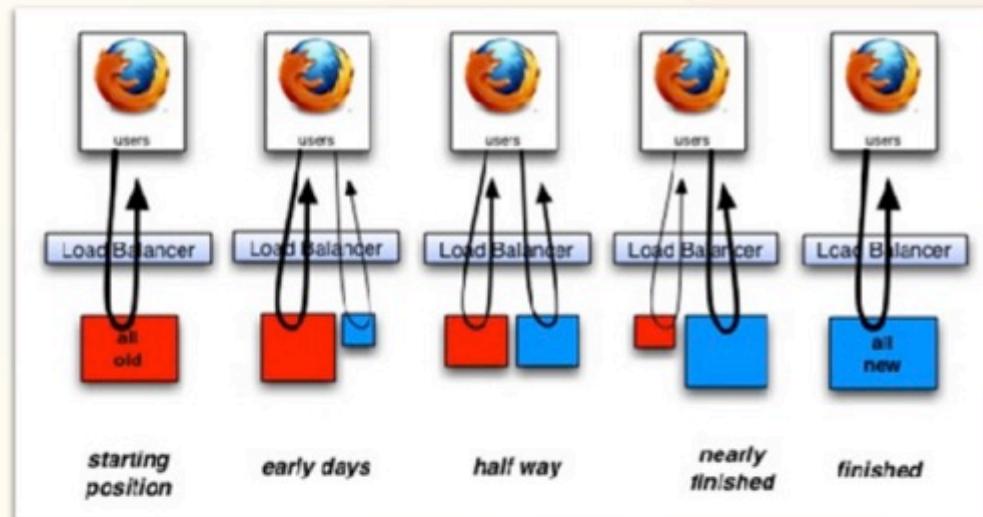


# Legacy Application Strangulation : Case Studies

## Strangler Applications

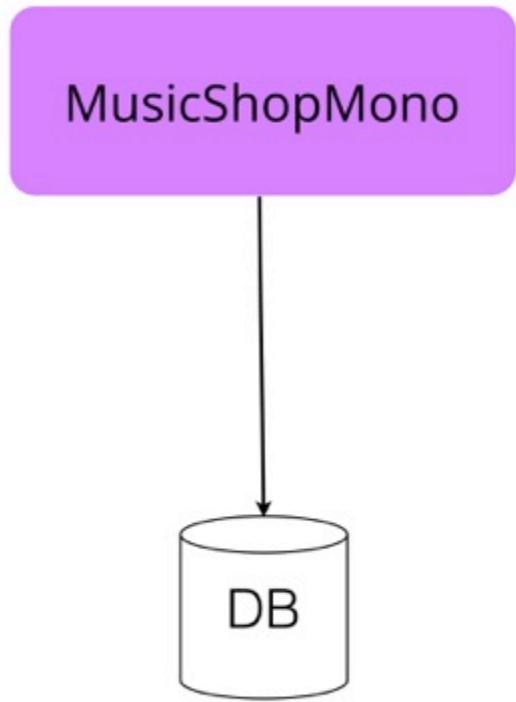
Martin Fowler wrote an [article](#) titled "Strangler Application" in mid 2004.

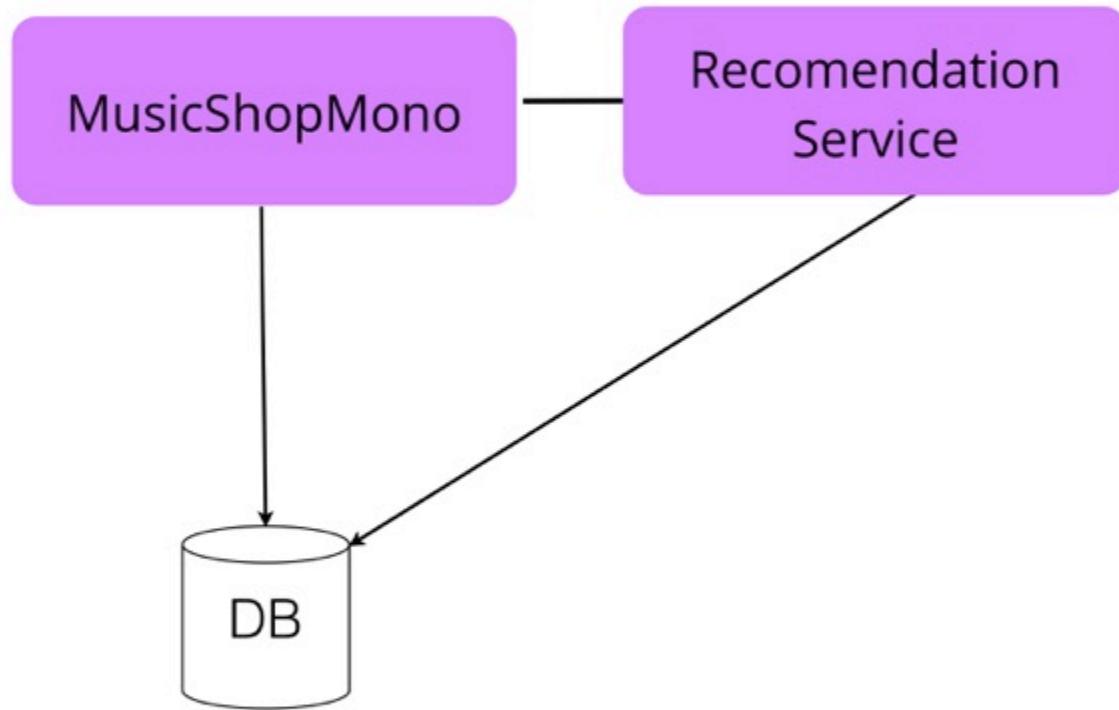
Strangulation of a legacy or undesirable solution is a safe way to phase one thing our for something better, cheaper, or more expandable. You make something new that obsoletes a small percentage of something old, and put them live together. You do some more work in the same style, and go live again (rinse, repeat). Here's a view of that (for web-apps):

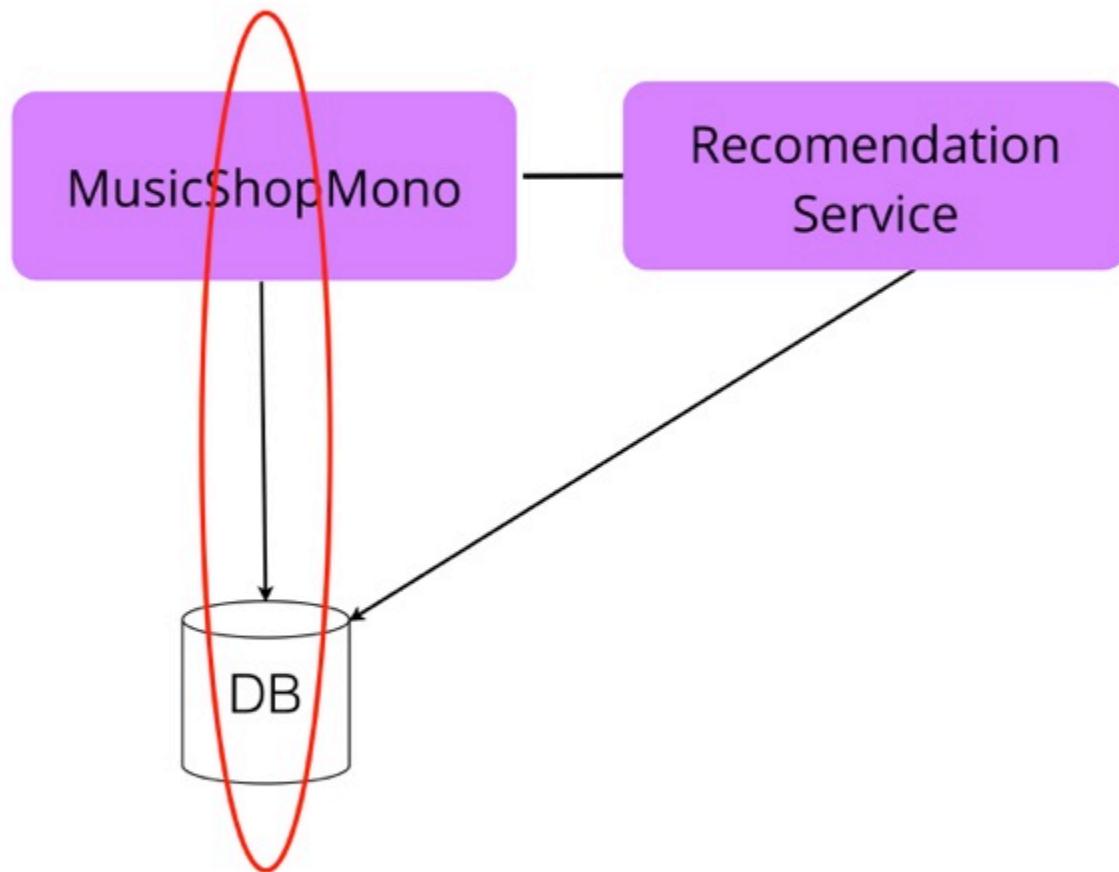


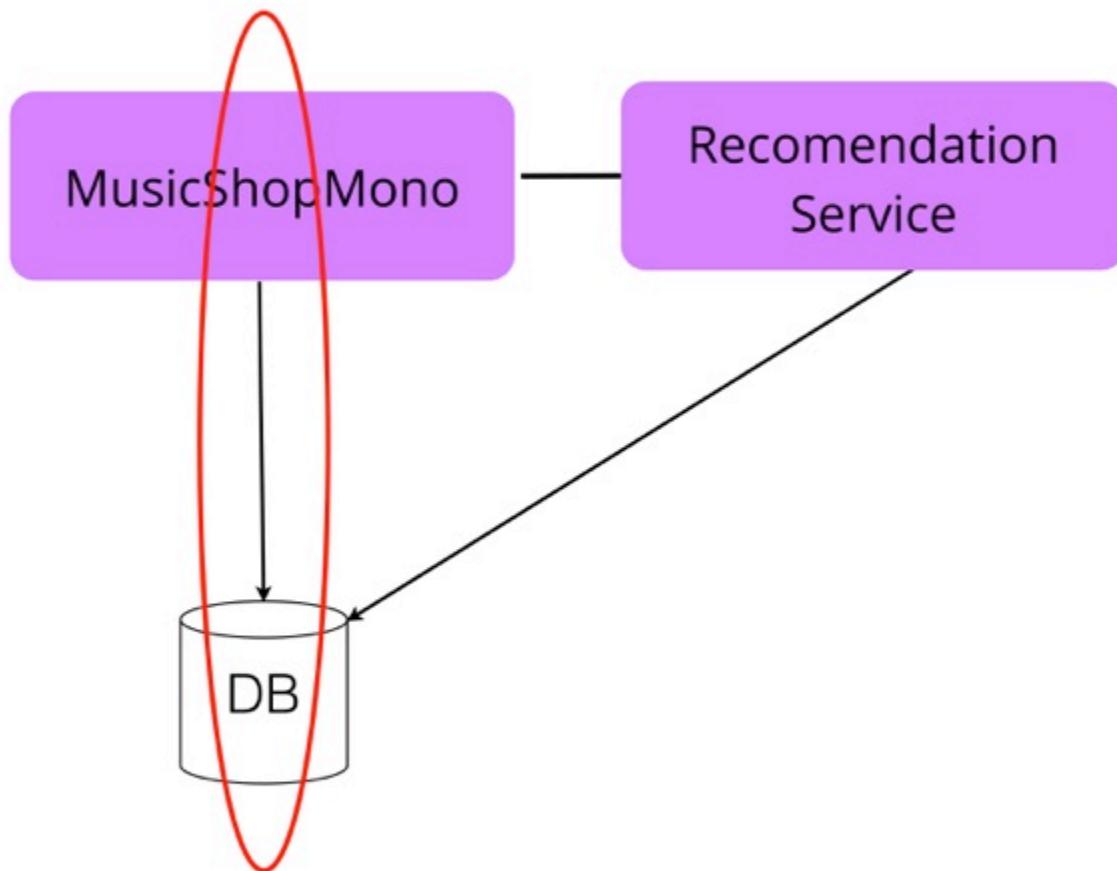
You could migrate all functionality from an old technology solution to new one in a series of releases that focussed on nothing else. Some companies will do that as there is a lot of sense to getting your house in order before doing anything else. However people outside the developer team may see that as a non-productive period, that could lengthen at any time, if it were asked for at all. People paying for that will notice, and may object. I mean execs, the board, or shareholders looking at the balance sheet.

# Databases

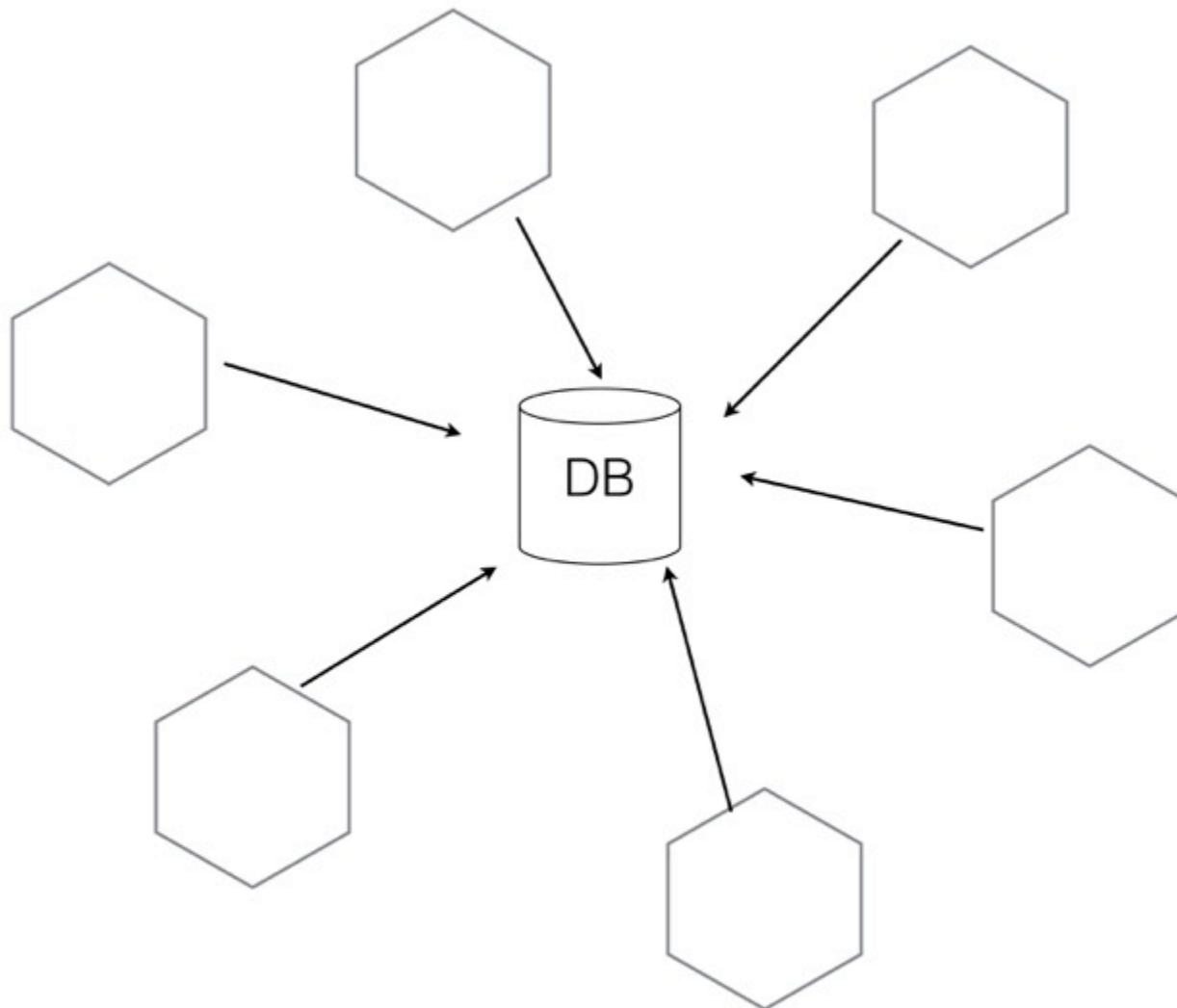


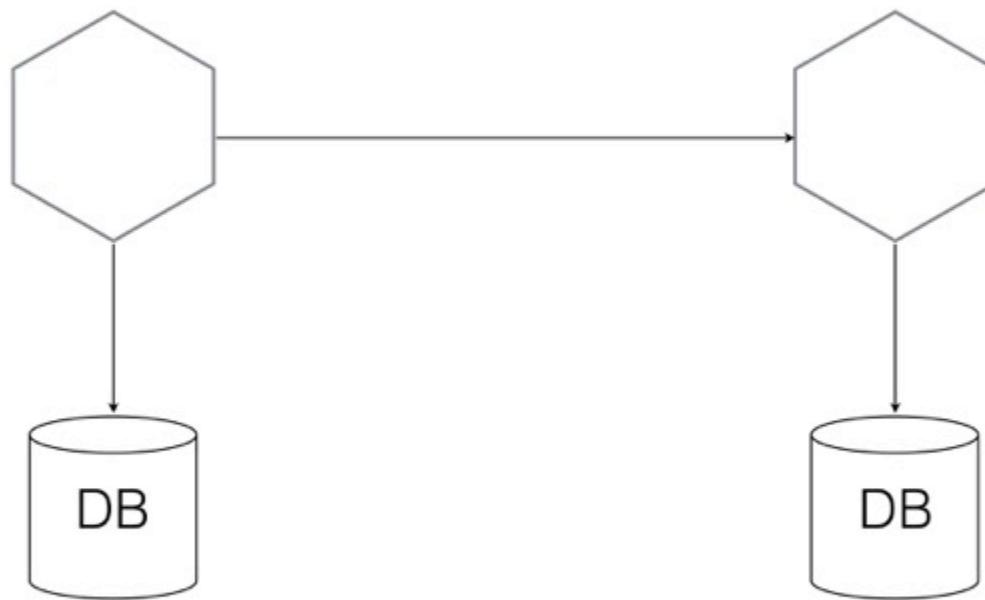




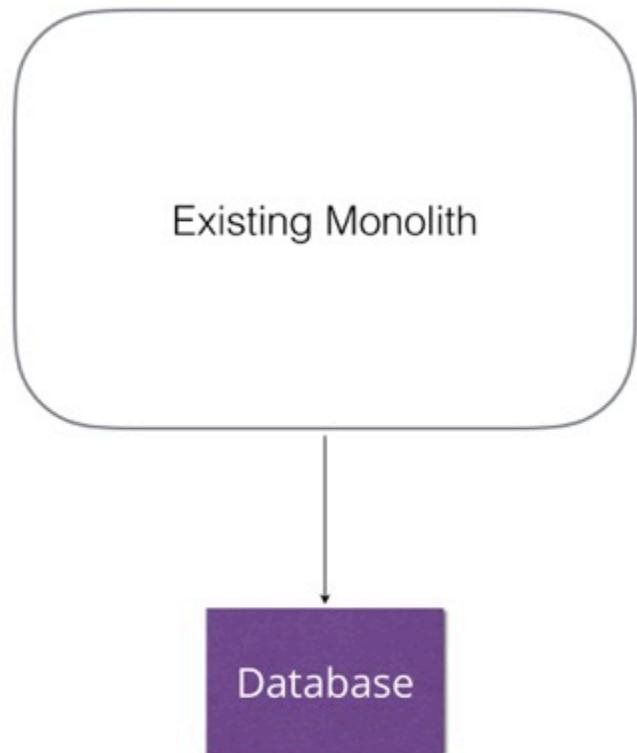


**No loose coupling!**

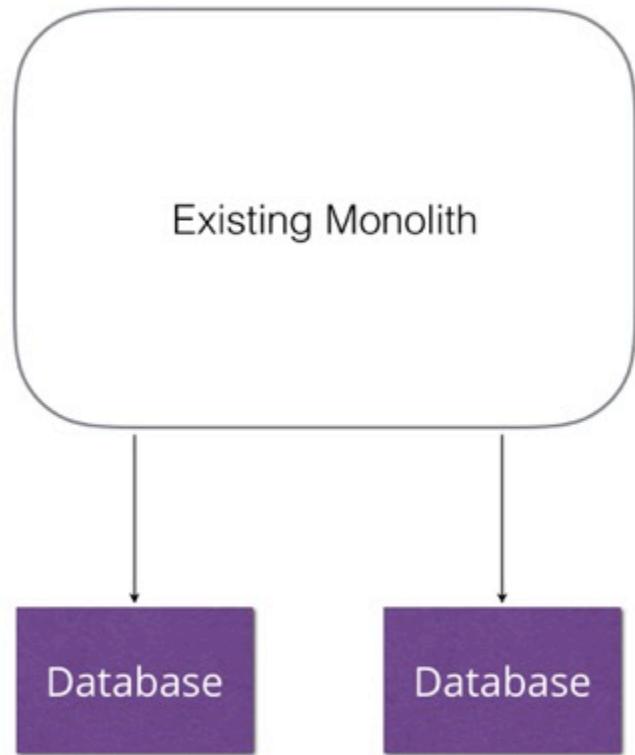




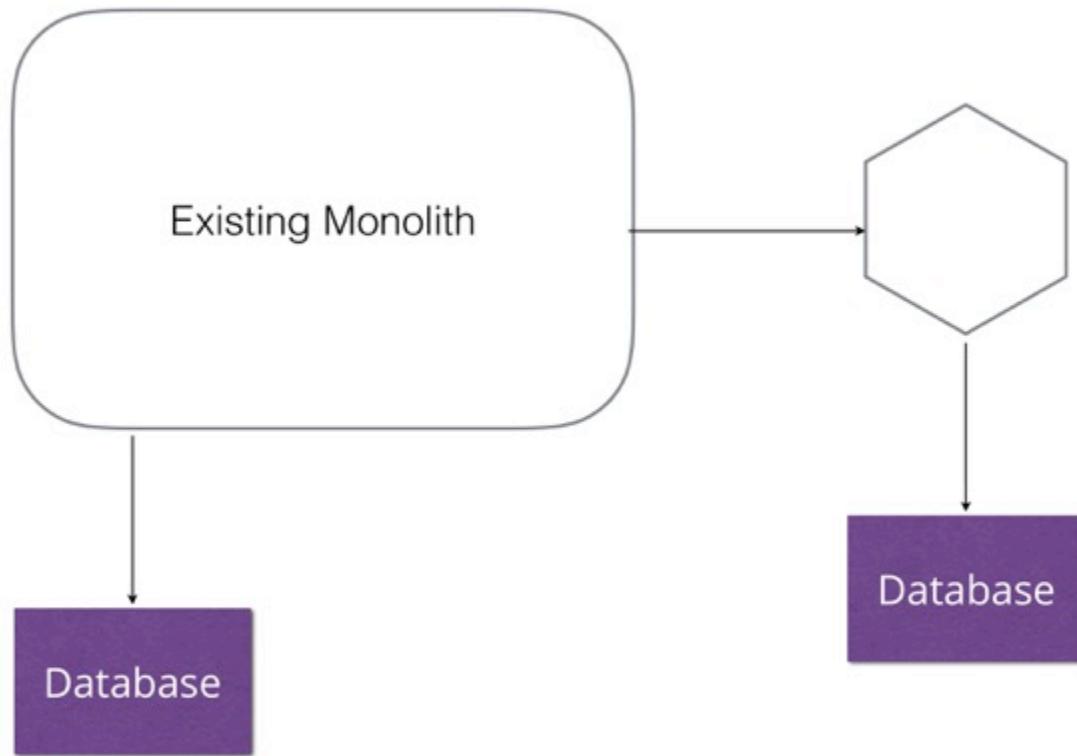
## SPLIT DATABASES FIRST?

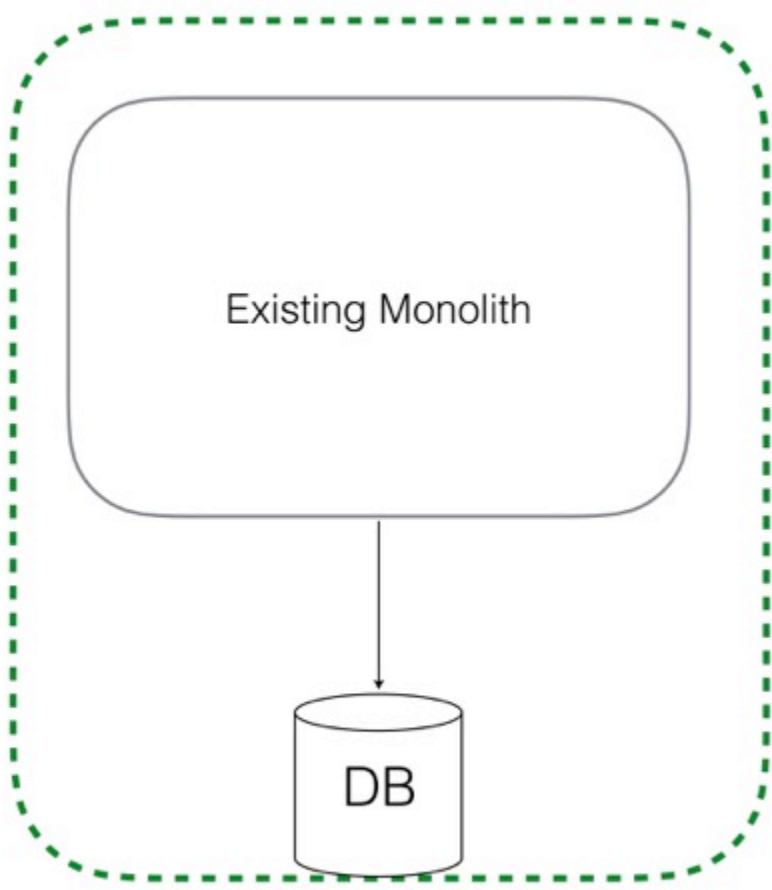


## SPLIT DATABASES FIRST?



## SPLIT DATABASES FIRST?

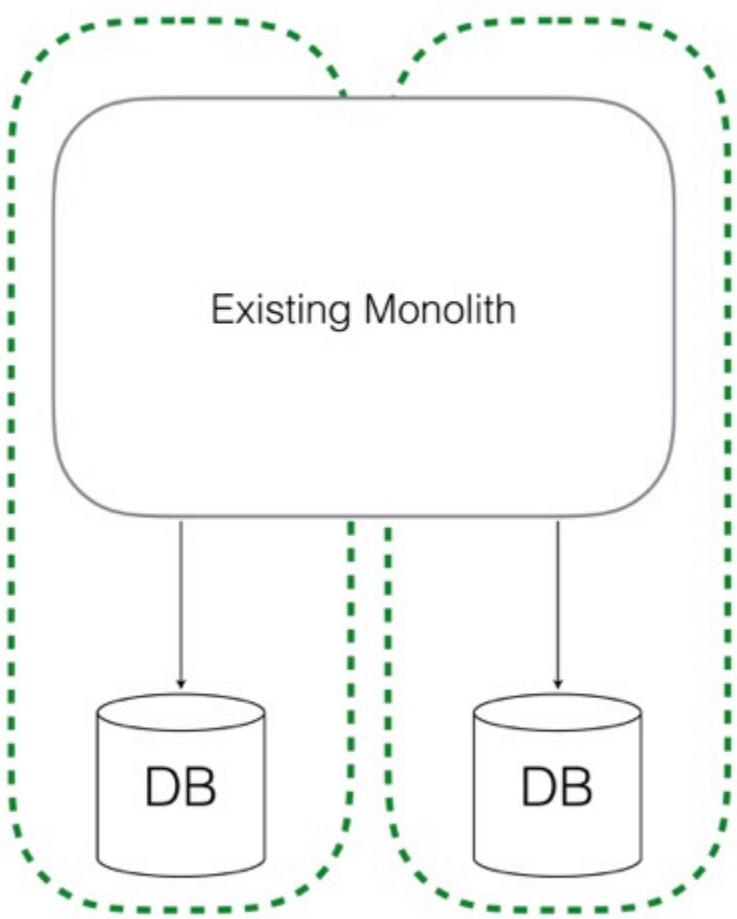




One Transactional  
Boundary

Single Database Call

Joins done in the DB



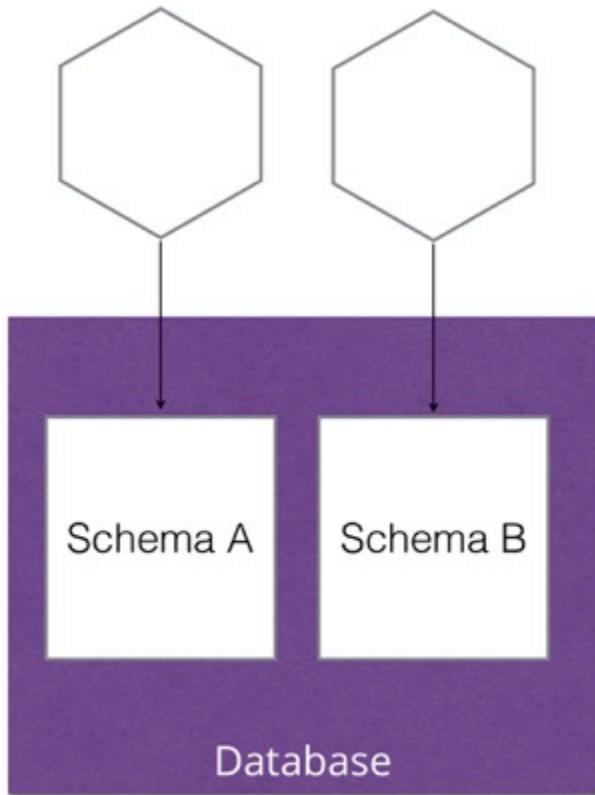
Broke transactional  
integrity

Two database calls

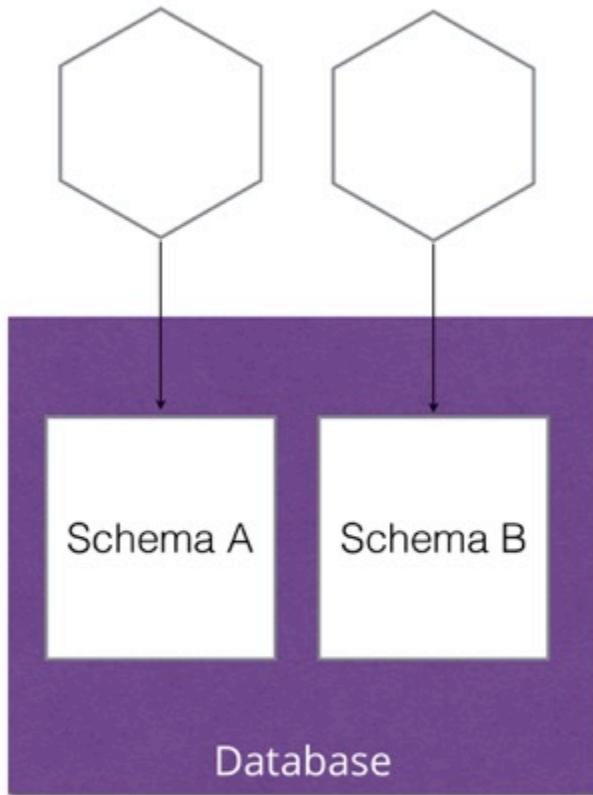
Some joins now done  
in application code

Understand the performance and transactional integrity issues early

## SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT

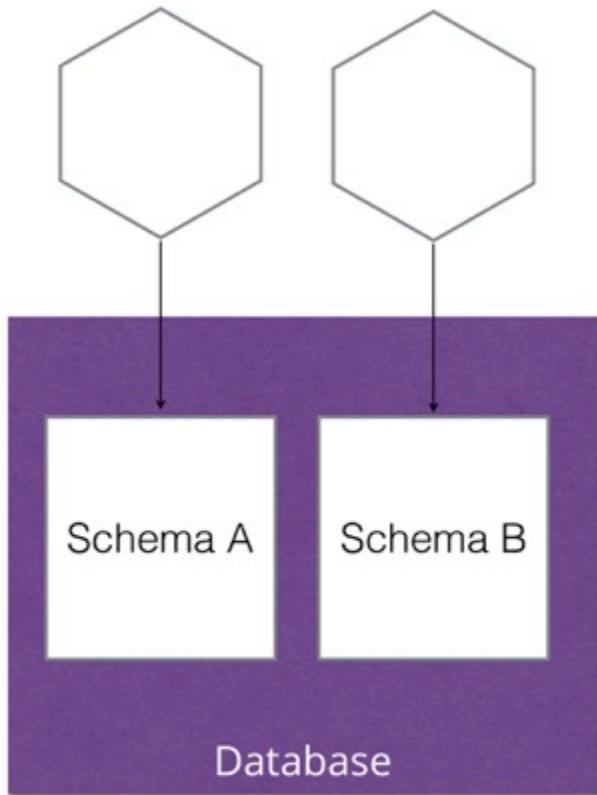


## SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT



Logically decoupled

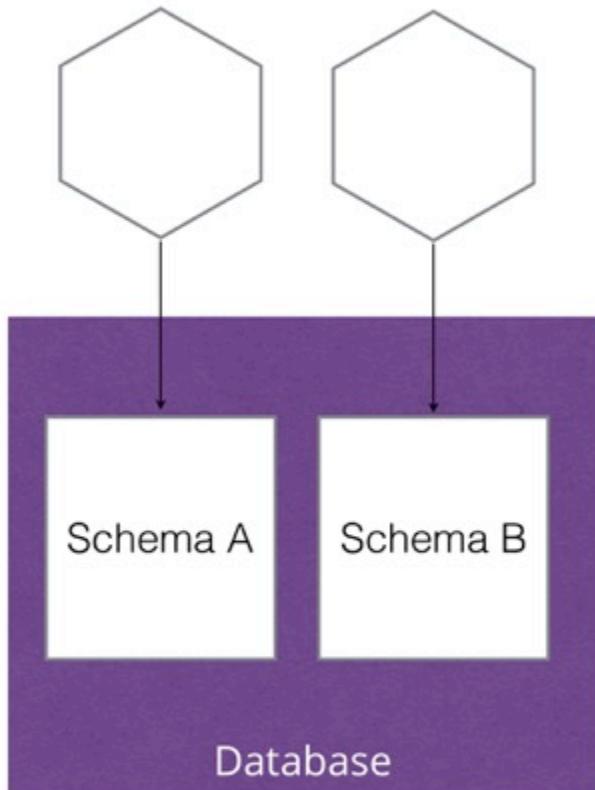
## SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT



Logically decoupled

Little or no extra infrastructure required

## SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT

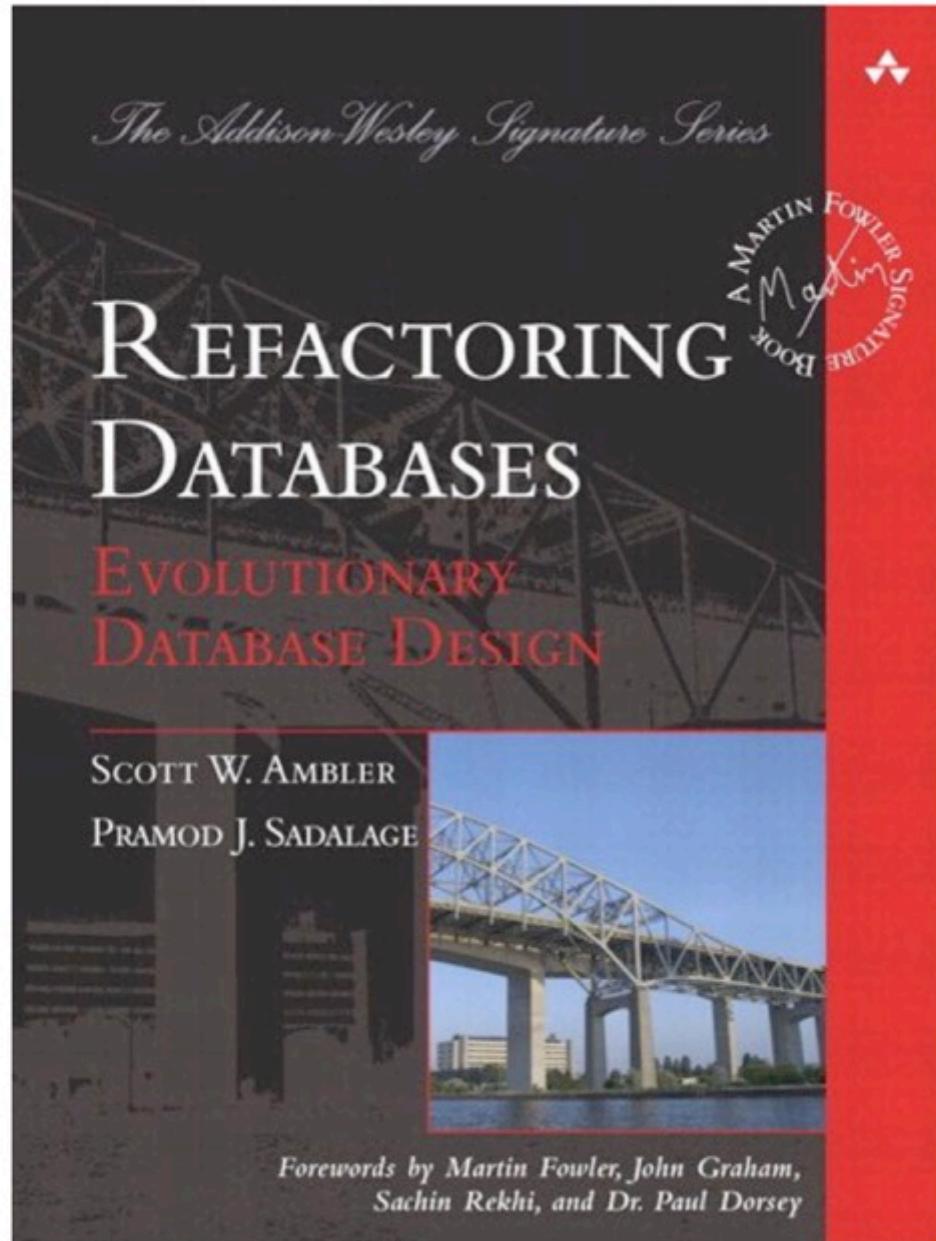


Logically decoupled

Little or no extra infrastructure required

Potential single point of failure

# DB Refactoring Patterns



# MusikShopMono

Catalog

Database

# MusikShopMono

Catalog

A diagram illustrating a data relationship. At the top, a rounded rectangle labeled "MusikShopMono" contains a pink rectangular box labeled "Catalog". A vertical arrow points downwards from the "Catalog" box to a table icon below. This table icon consists of a grid with 3 rows and 3 columns, representing "Line Items". The entire "Line Items" section is enclosed within a large purple rectangular frame labeled "Database" at the bottom.


Line Items

Database

# MusikShopMono

Catalog

Finance


Line Items

Database

# MusikShopMono

Catalog

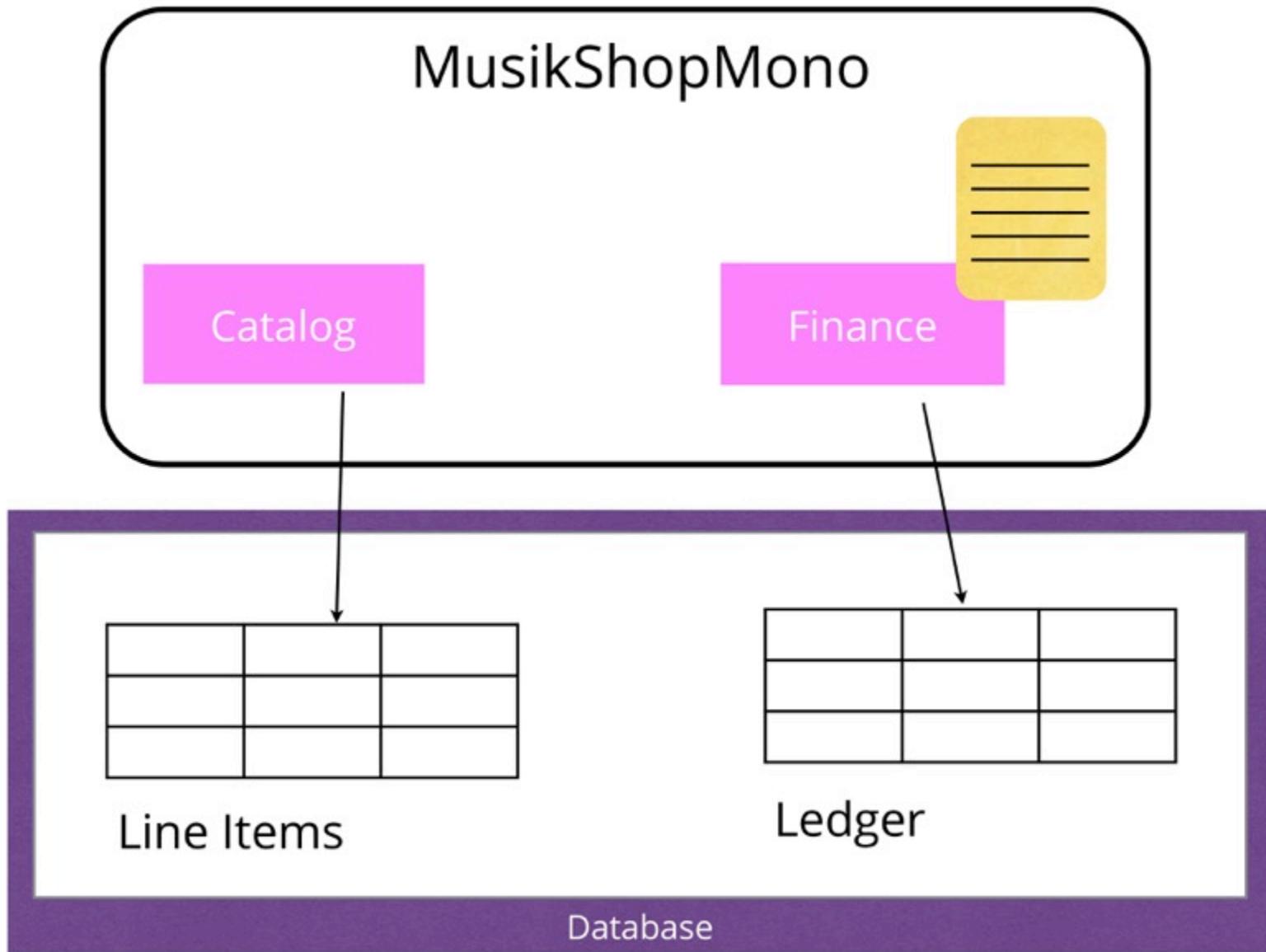
Finance


Line Items

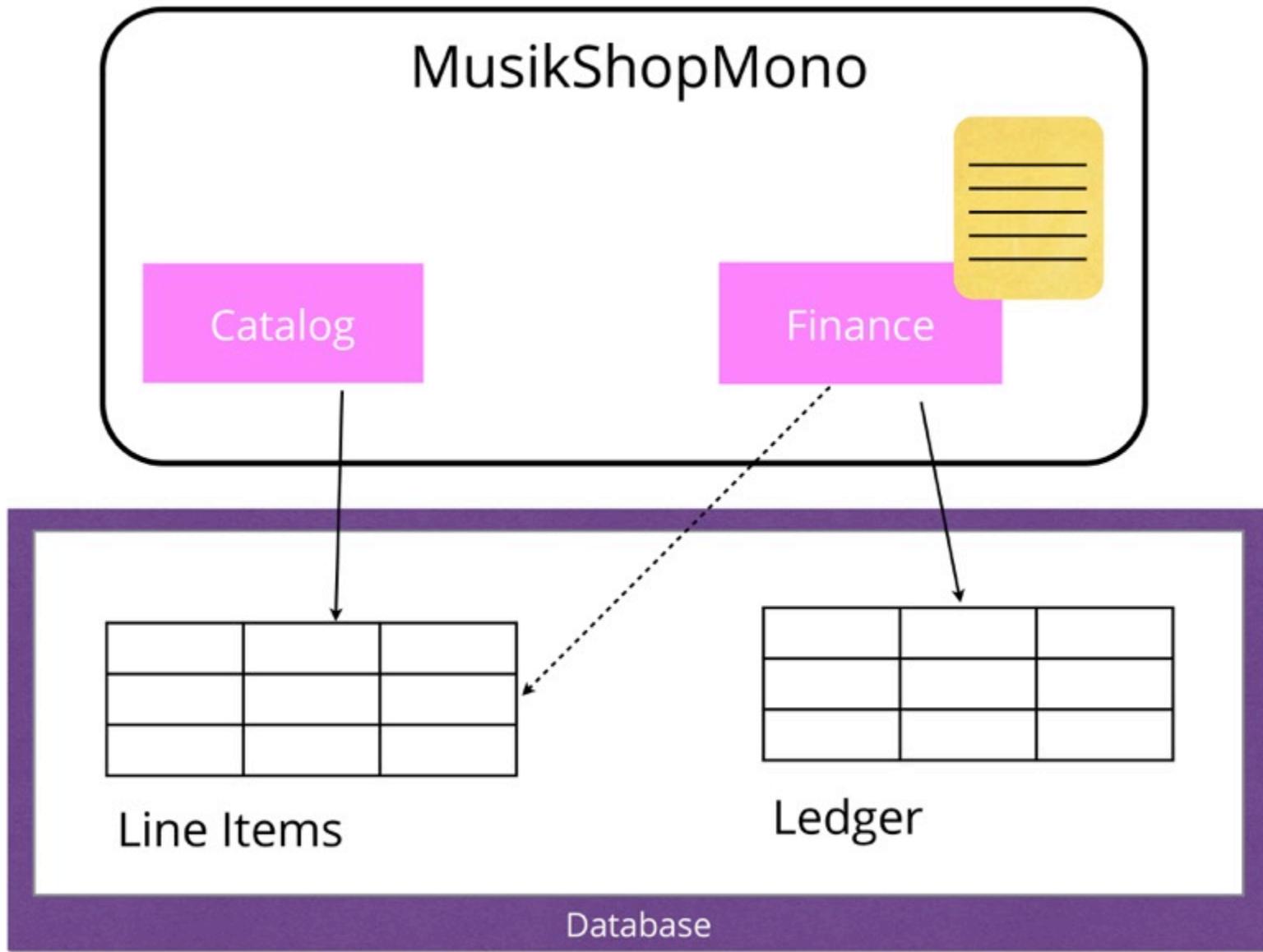

Ledger

Database

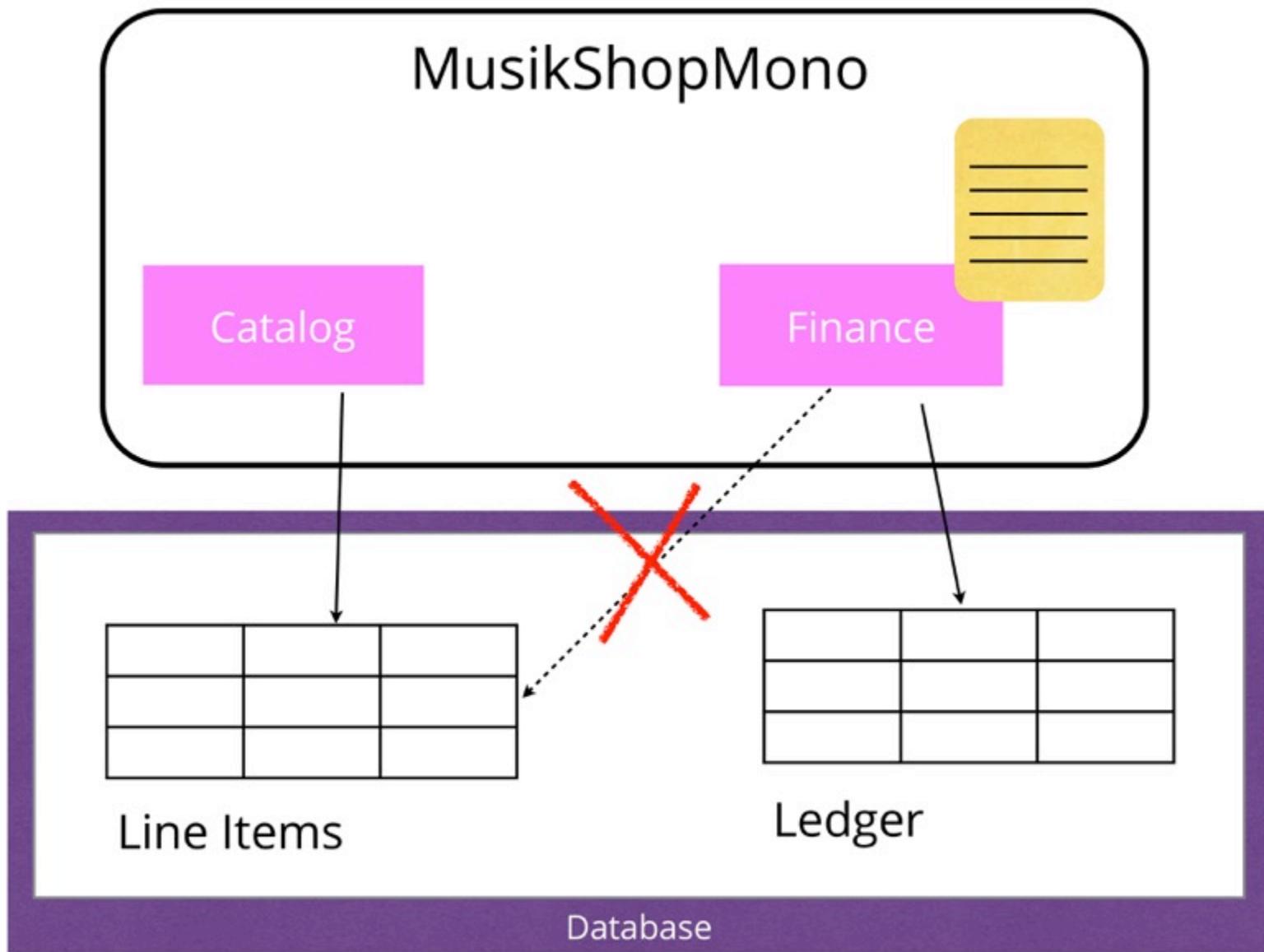
# MusikShopMono



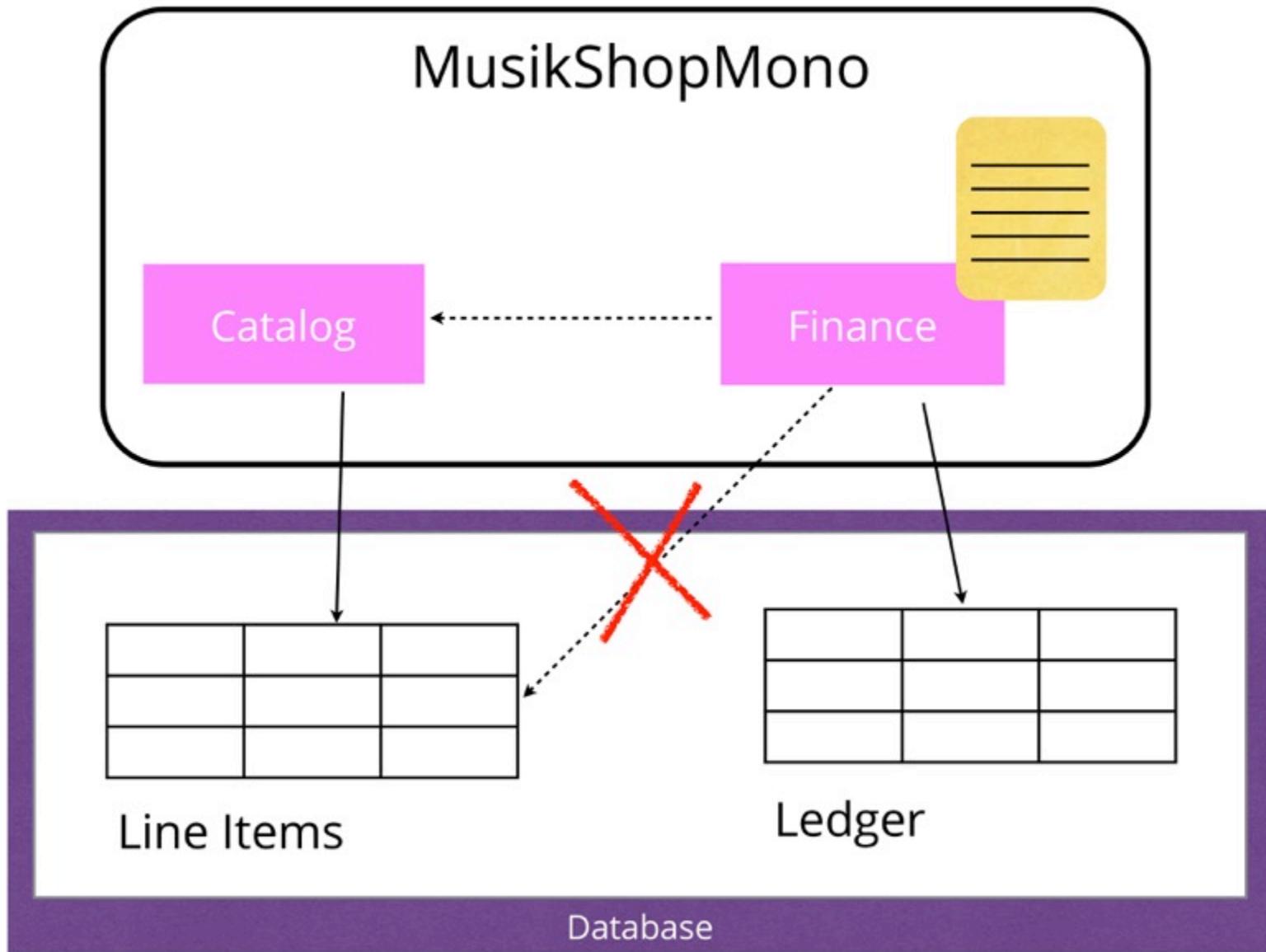
# MusikShopMono



# MusikShopMono



# MusikShopMono



# MusikShopMono

Catalog



Database



Database

# MusikShopMono

Catalog

Finance

Database

Database

# MusikShopMono

Catalog

Finance



Database



Database

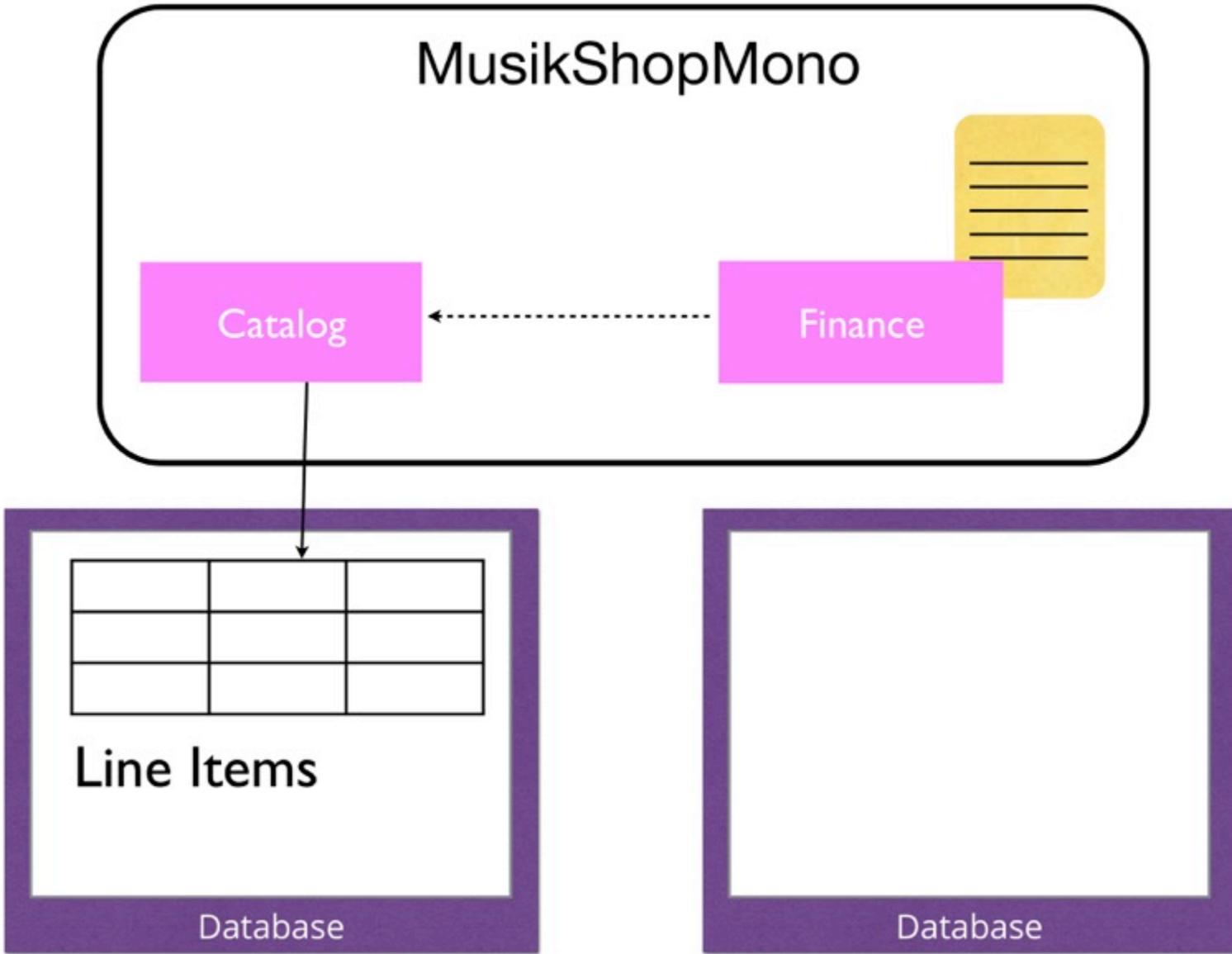
# MusikShopMono

Catalog

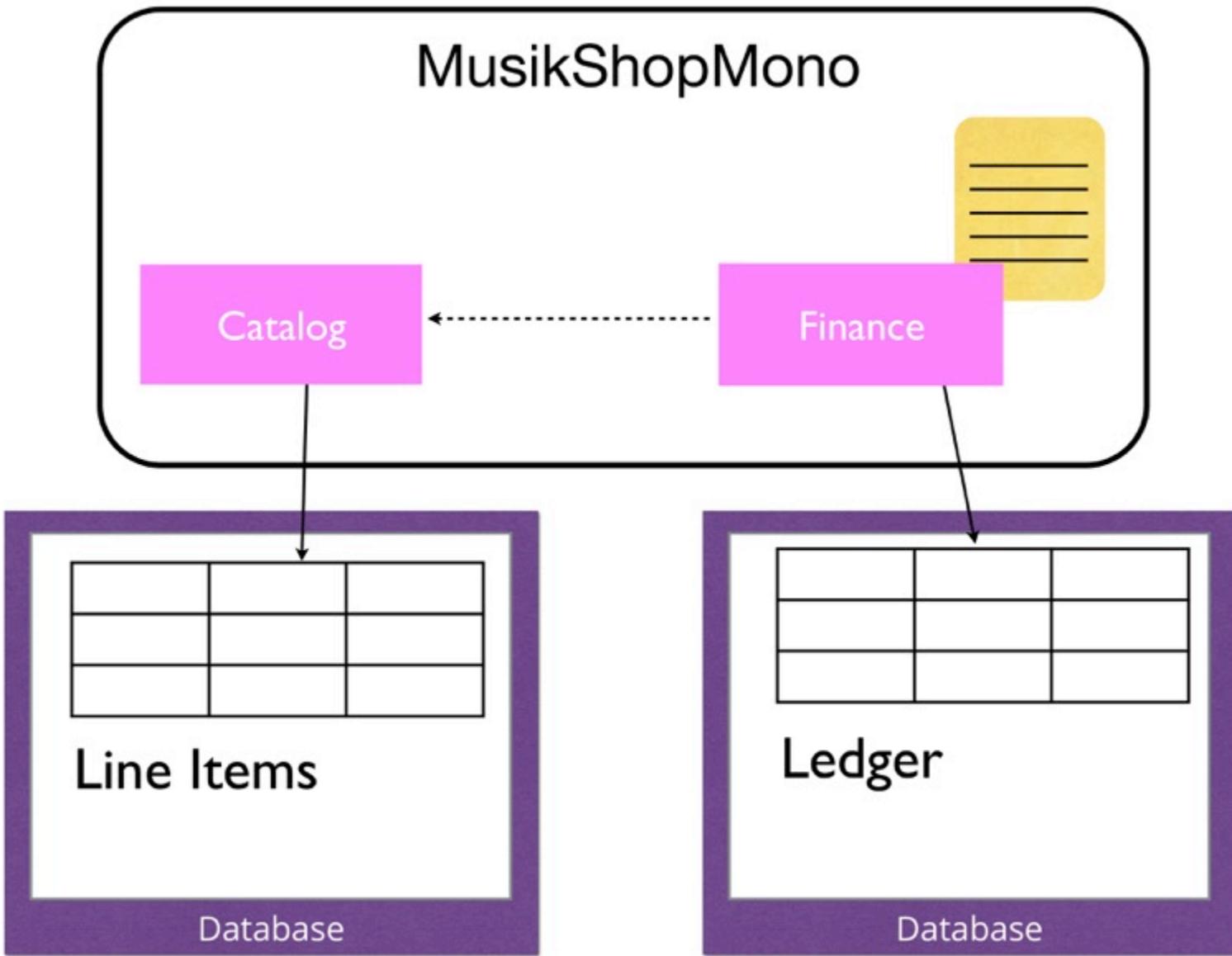
Finance

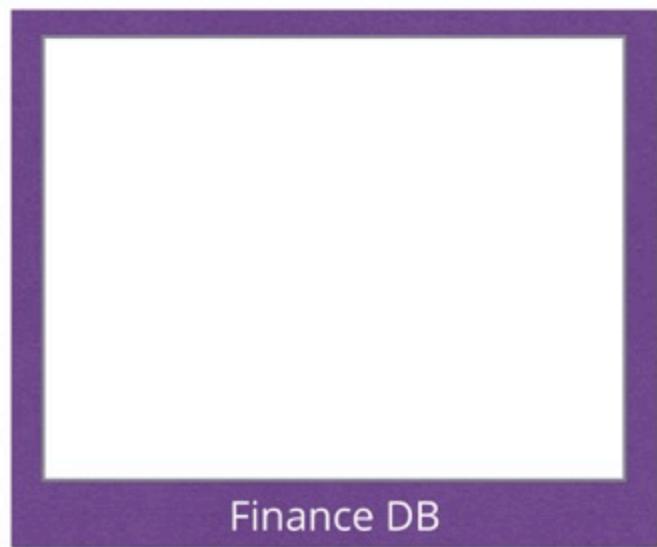
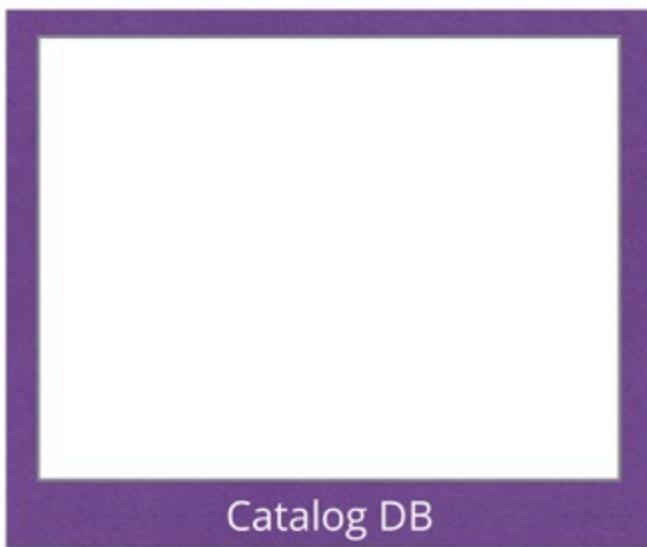


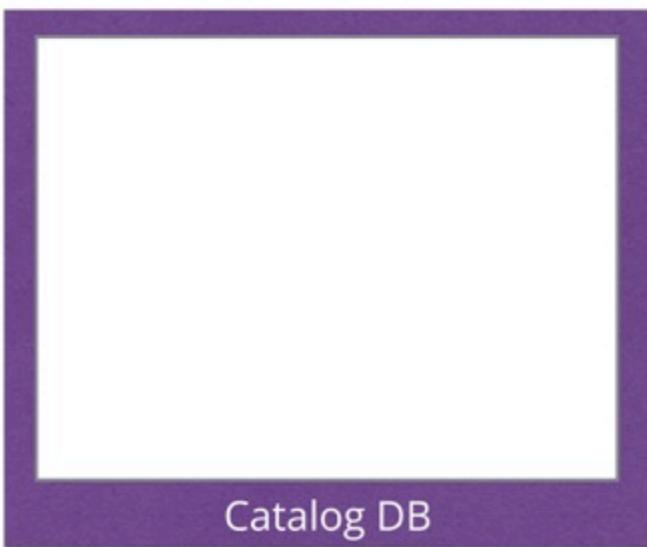
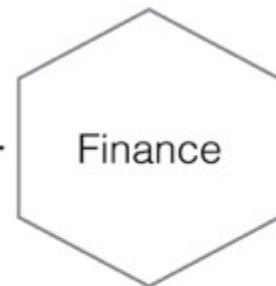
# MusikShopMono



# MusikShopMono



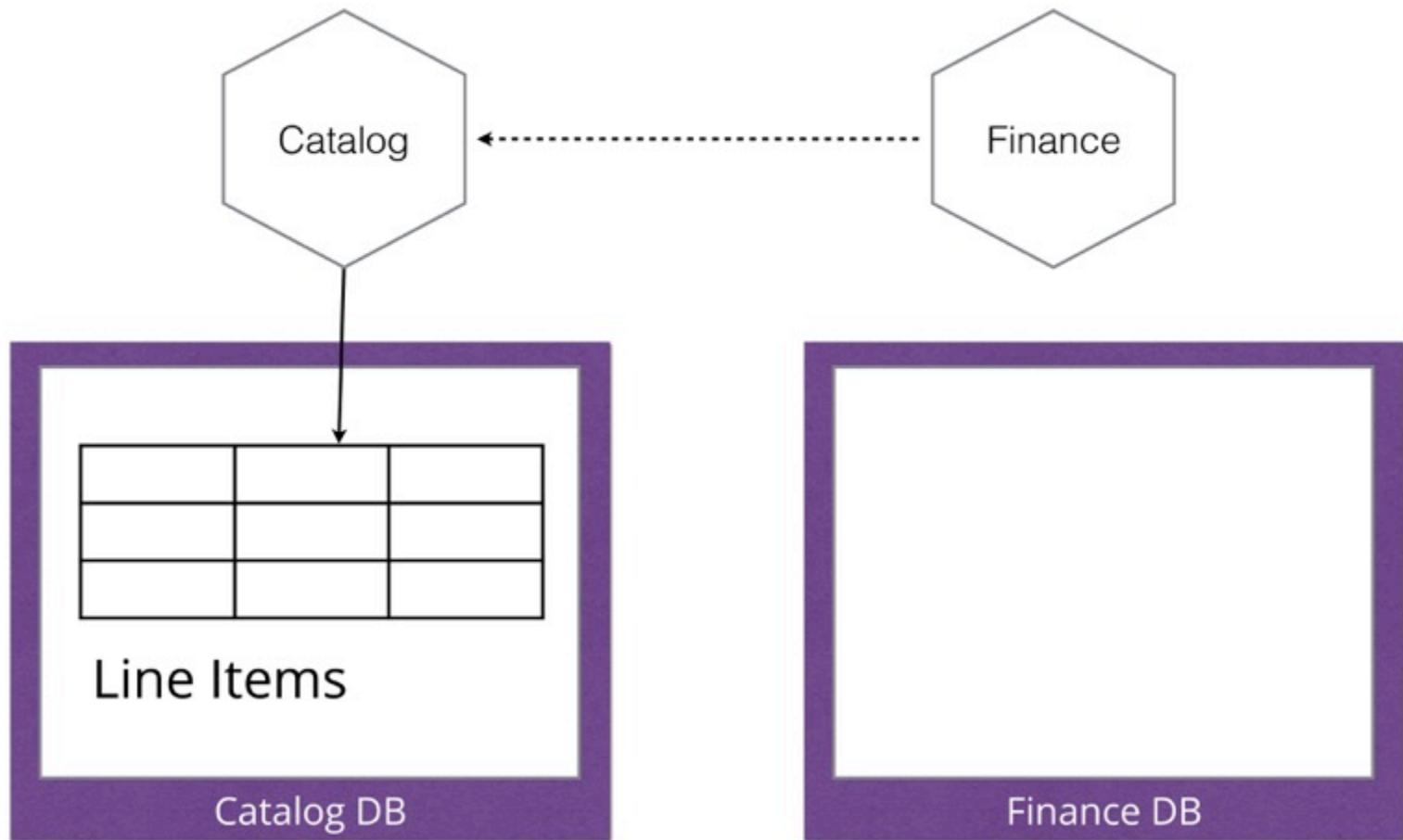


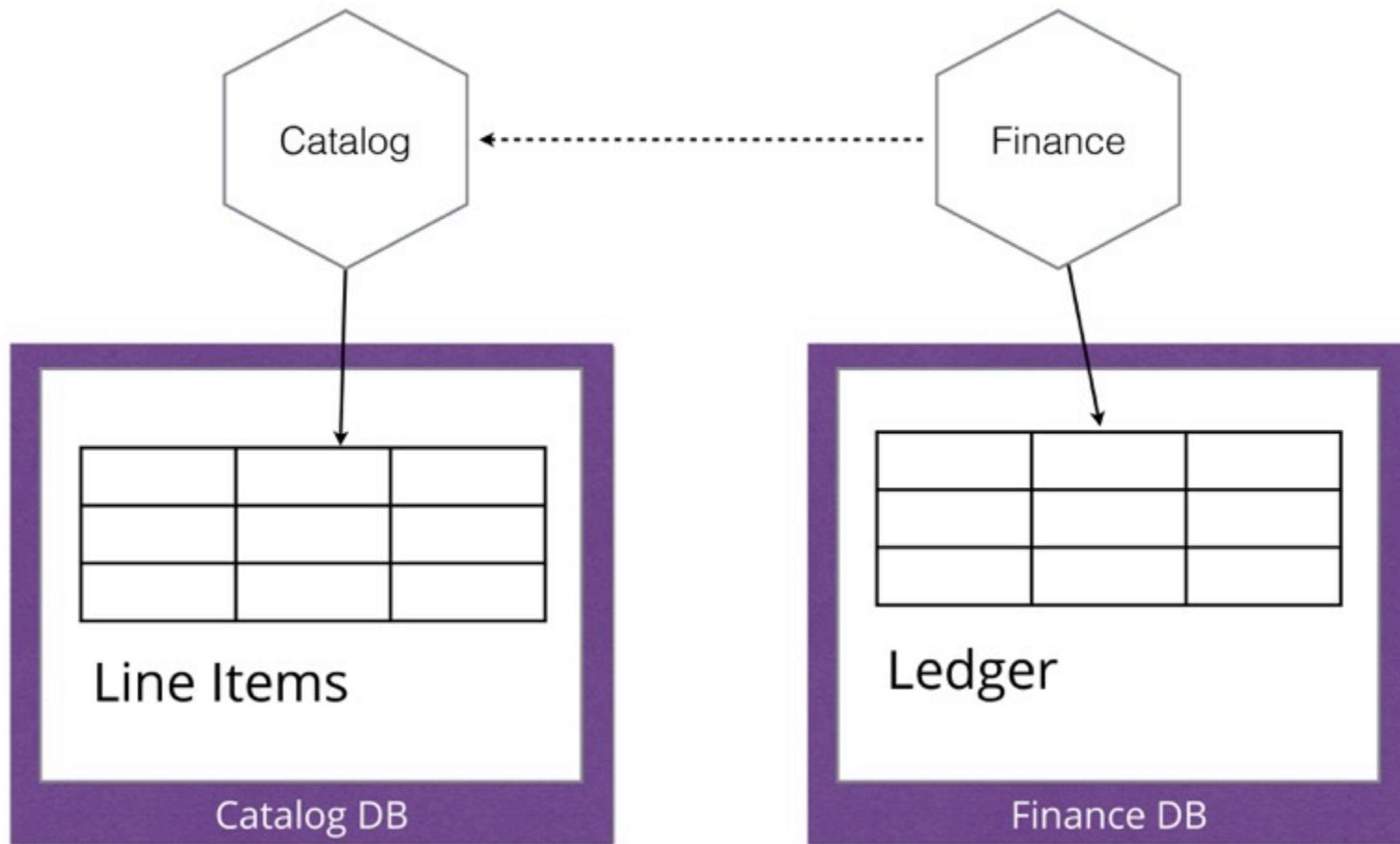


Catalog DB



Finance DB







Database

---

ID	Name	
123	Give Blood	

## Line Items

Database

---

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

---

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

.....

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

---

ID		
123		

Line Items

Catalog DB

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

---

ID		
123		

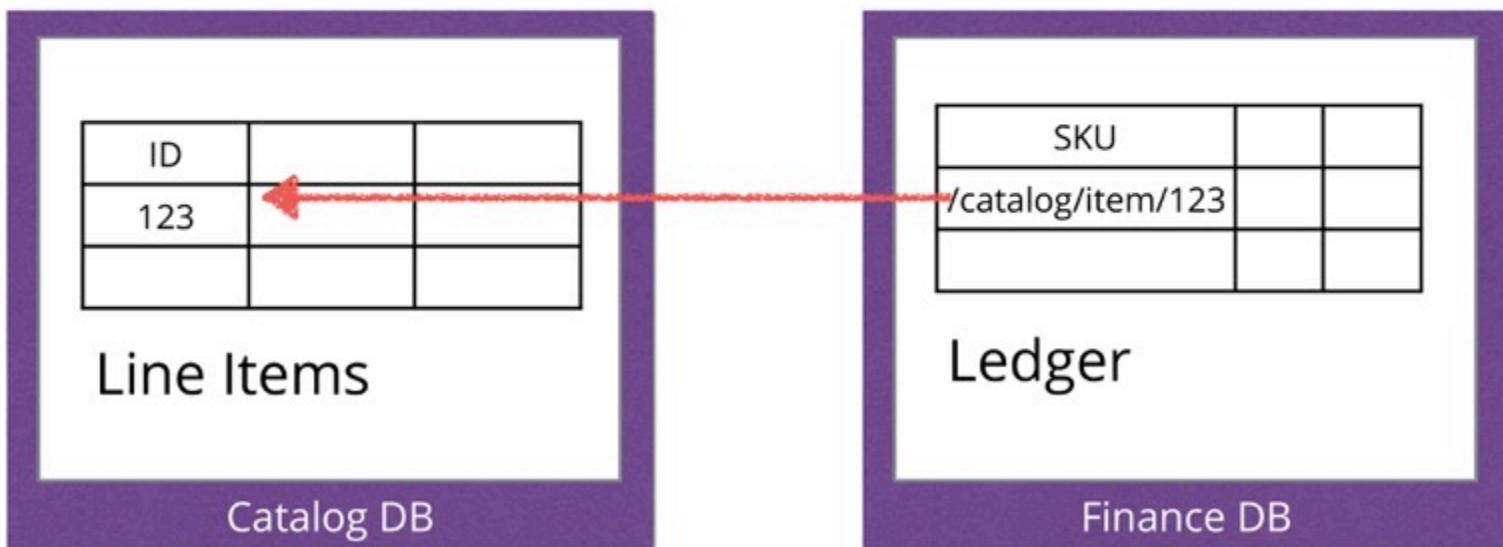
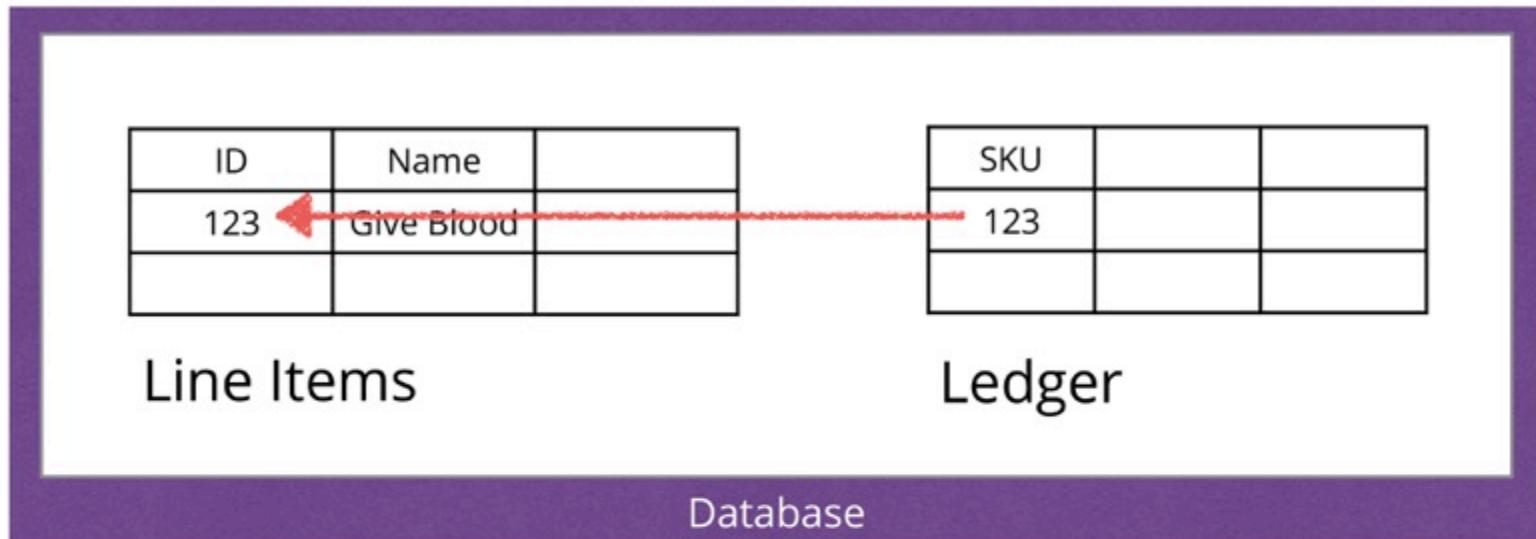
Line Items

Catalog DB

SKU		
/catalog/item/123		

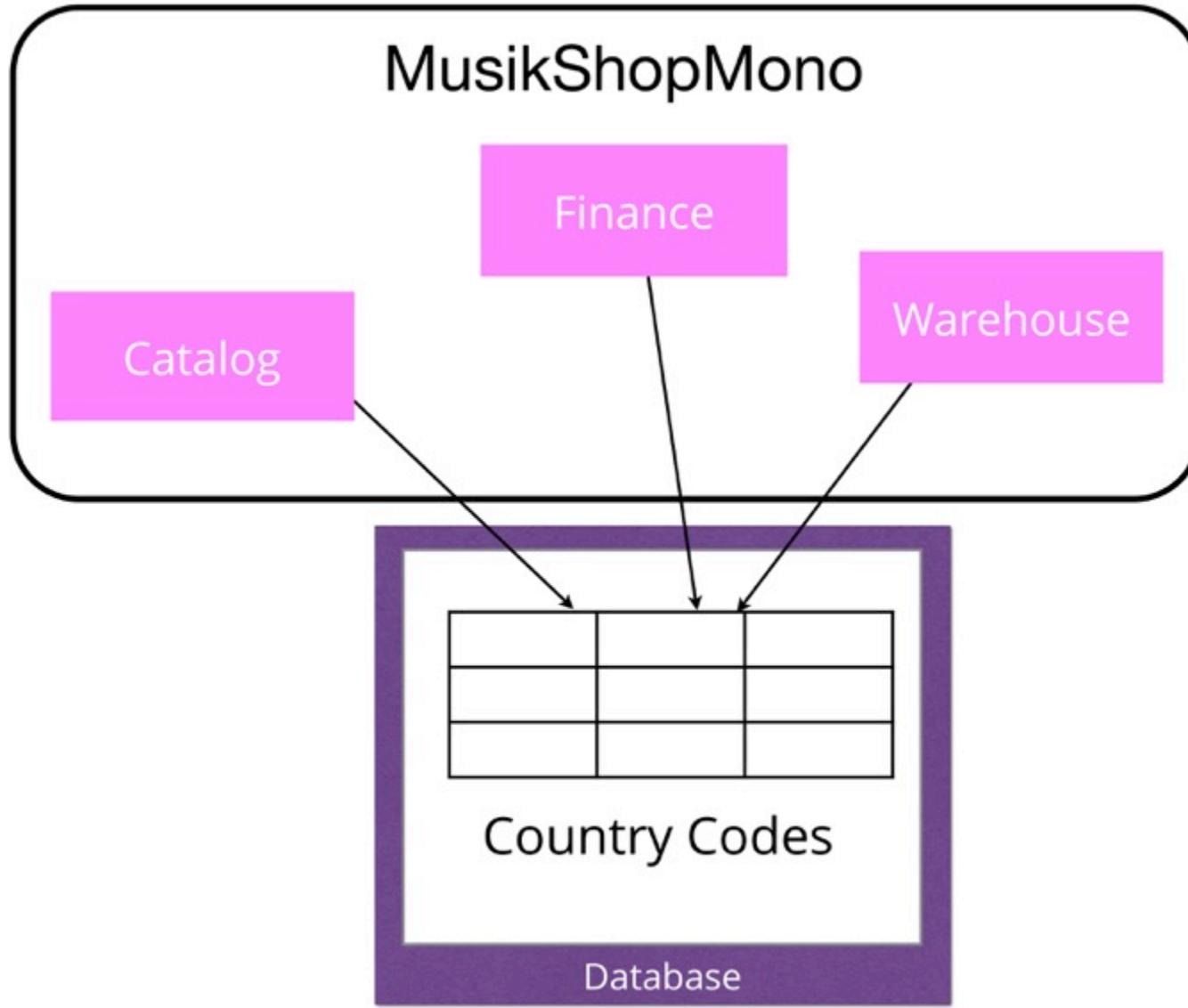
Ledger

Finance DB

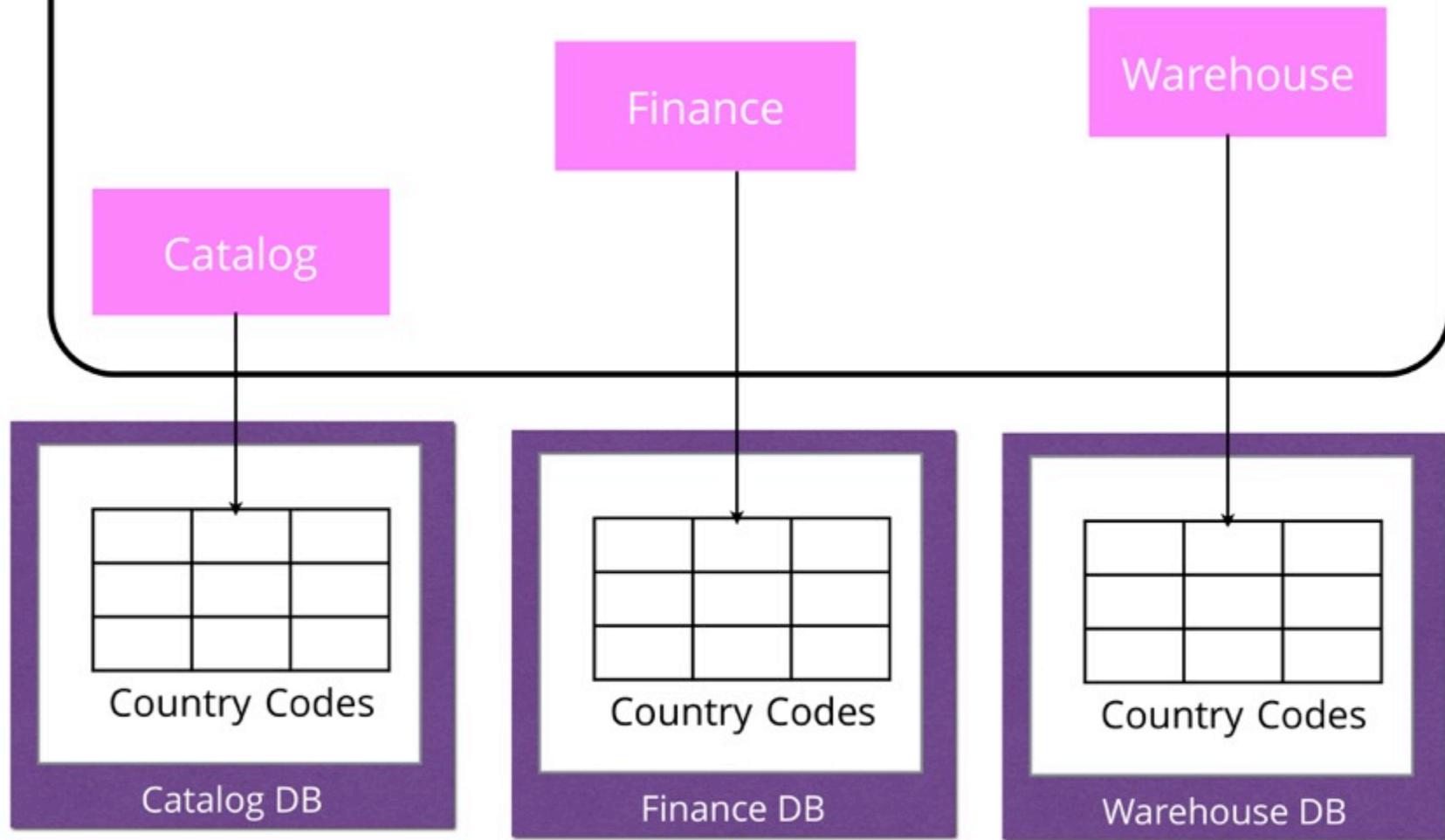


Breaking foreign key relationships can introduce inconsistency in your system

# MusikShopMono



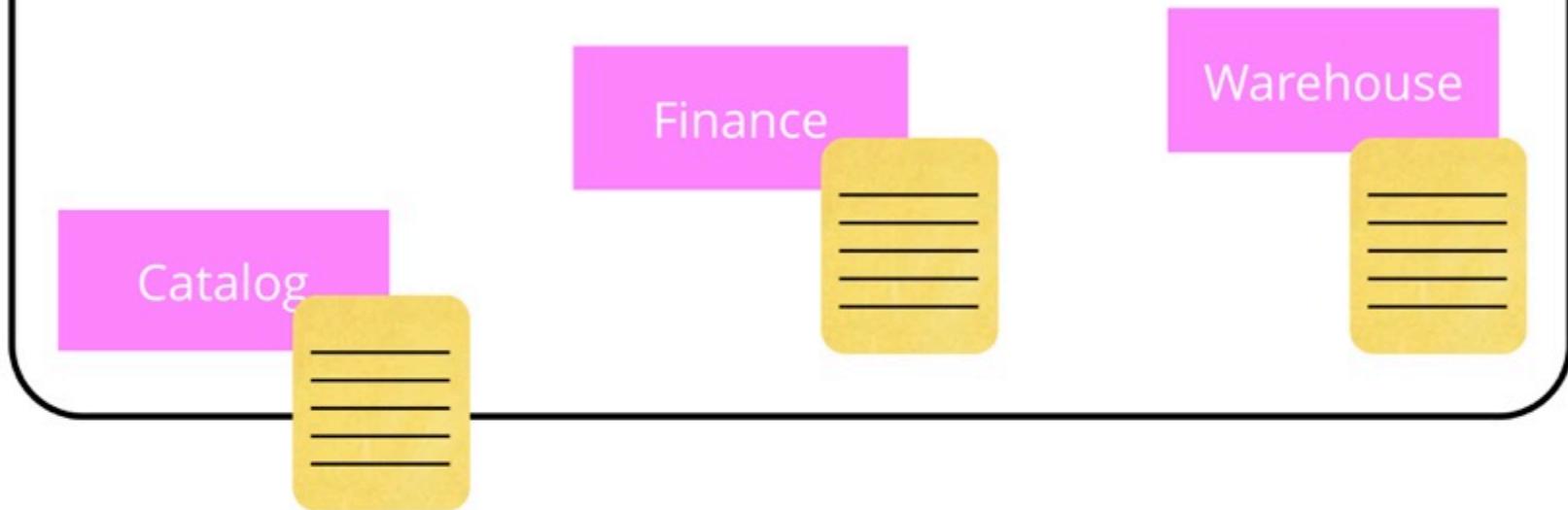
# MusikShopMono



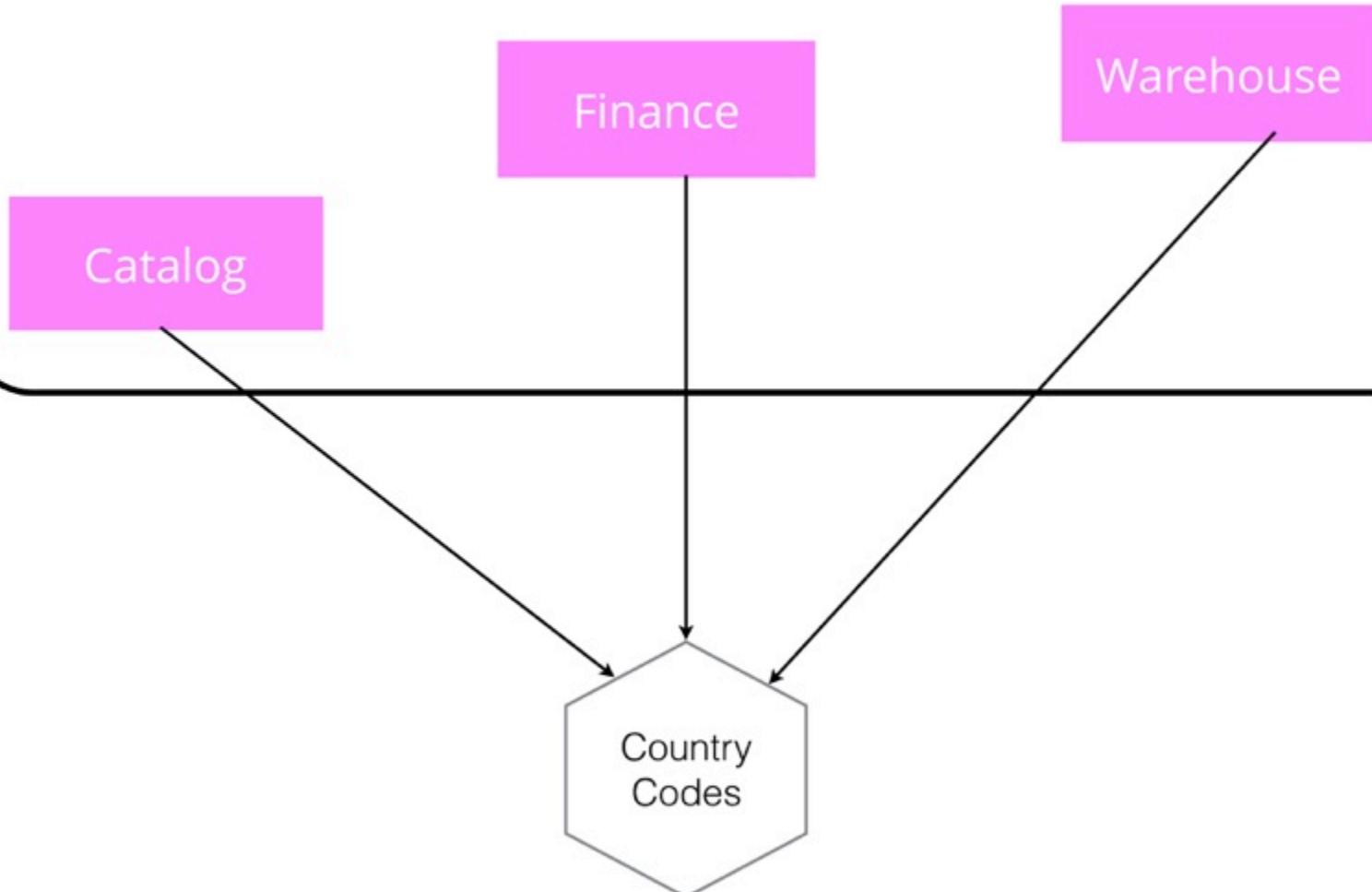
Duplication isn't  
always bad...

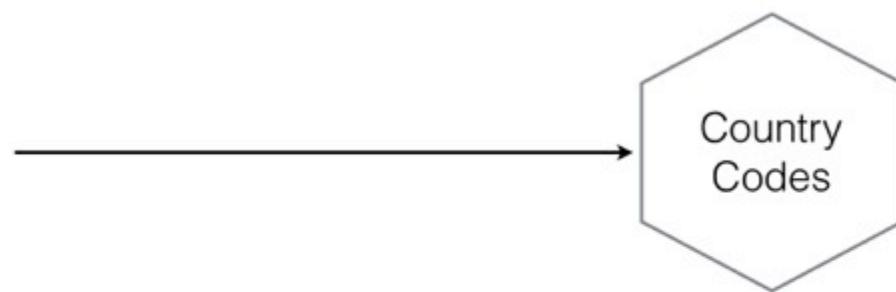
...but watch for  
inconsistency

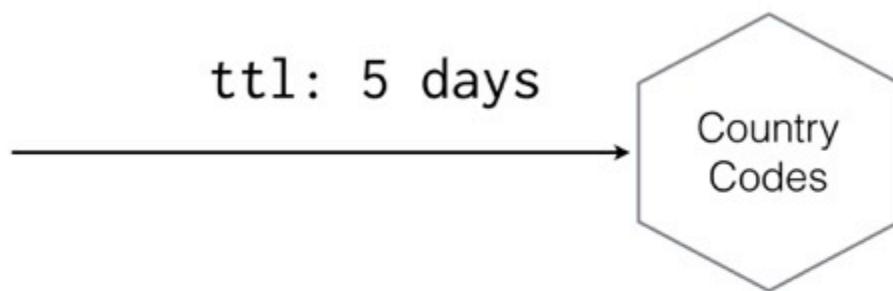
# MusikShopMono



# MusikShopMono







Can aggressively cache

What about stale  
data?

# Eventual Consistency

**Building reliable distributed systems  
at a worldwide scale demands trade-offs  
between consistency and availability.**

BY WERNER VOGELS

# Eventually Consistent

AT THE FOUNDATION of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost-effectiveness, and

transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.

One of the ways in which this manifests itself is in the type of data consistency that is provided, particularly when many widespread distributed systems provide an *eventual consistency* model in the context of data replication. When designing these large-scale systems at Amazon, we use a set of guiding principles and abstractions related to large-scale data replication and focus on the trade-offs between high availability and data consistency. Here, I present some of the relevant background that has informed our approach to delivering reliable distributed systems that must operate on a global scale. (An earlier version of this article appeared as a posting on the "All Things Distributed" Weblog and was greatly improved with the help of its readers.)

#### Historical Perspective

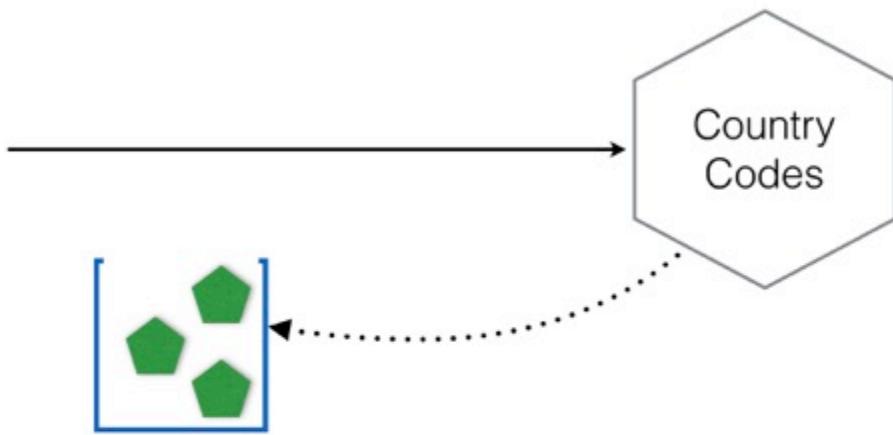
In an ideal world there would be only one consistency model: when an update is made all observers would see that update. The first time this surfaced as difficult to achieve was in the database systems of the late 1970s. The best "period piece" on this topic

<http://dl.acm.org/citation.cfm?doid=1435417.1435432>

Eventually consistent services can be easier to scale as they reduce contention



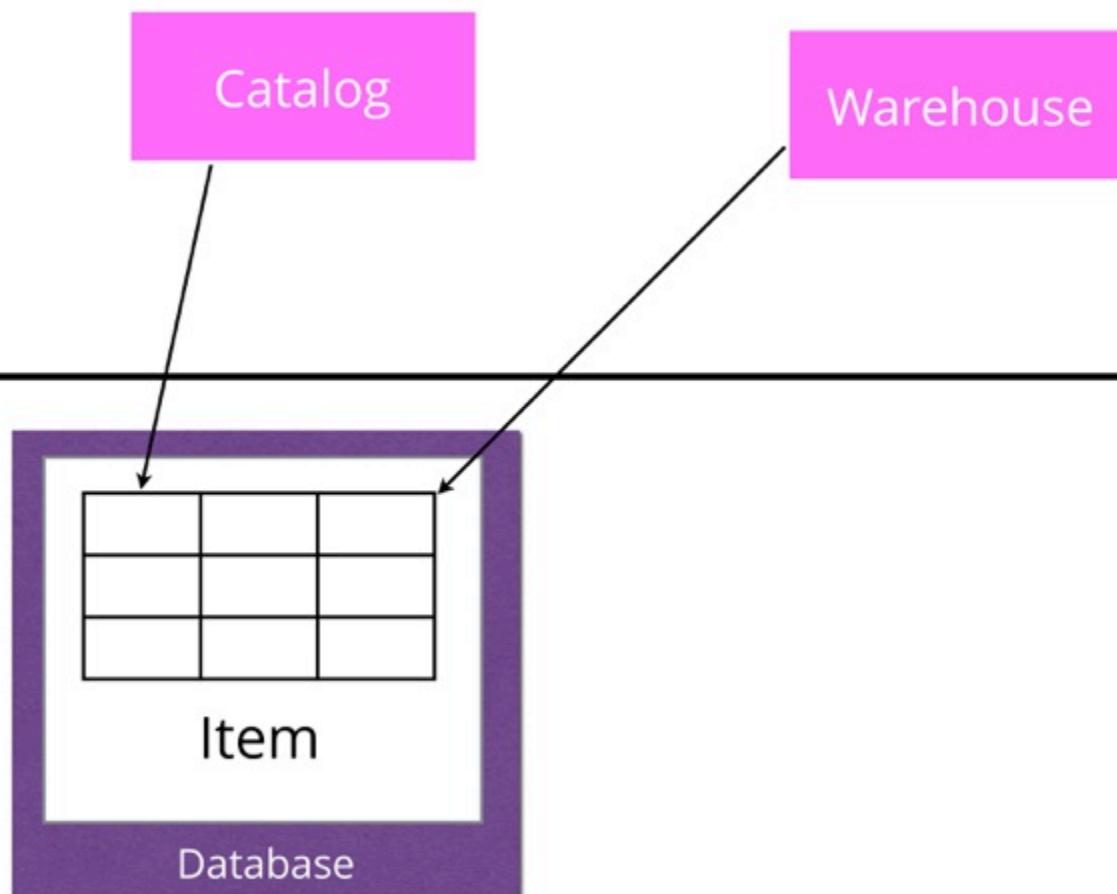
Use events to propagate changes  
Event propagation isn't immediate



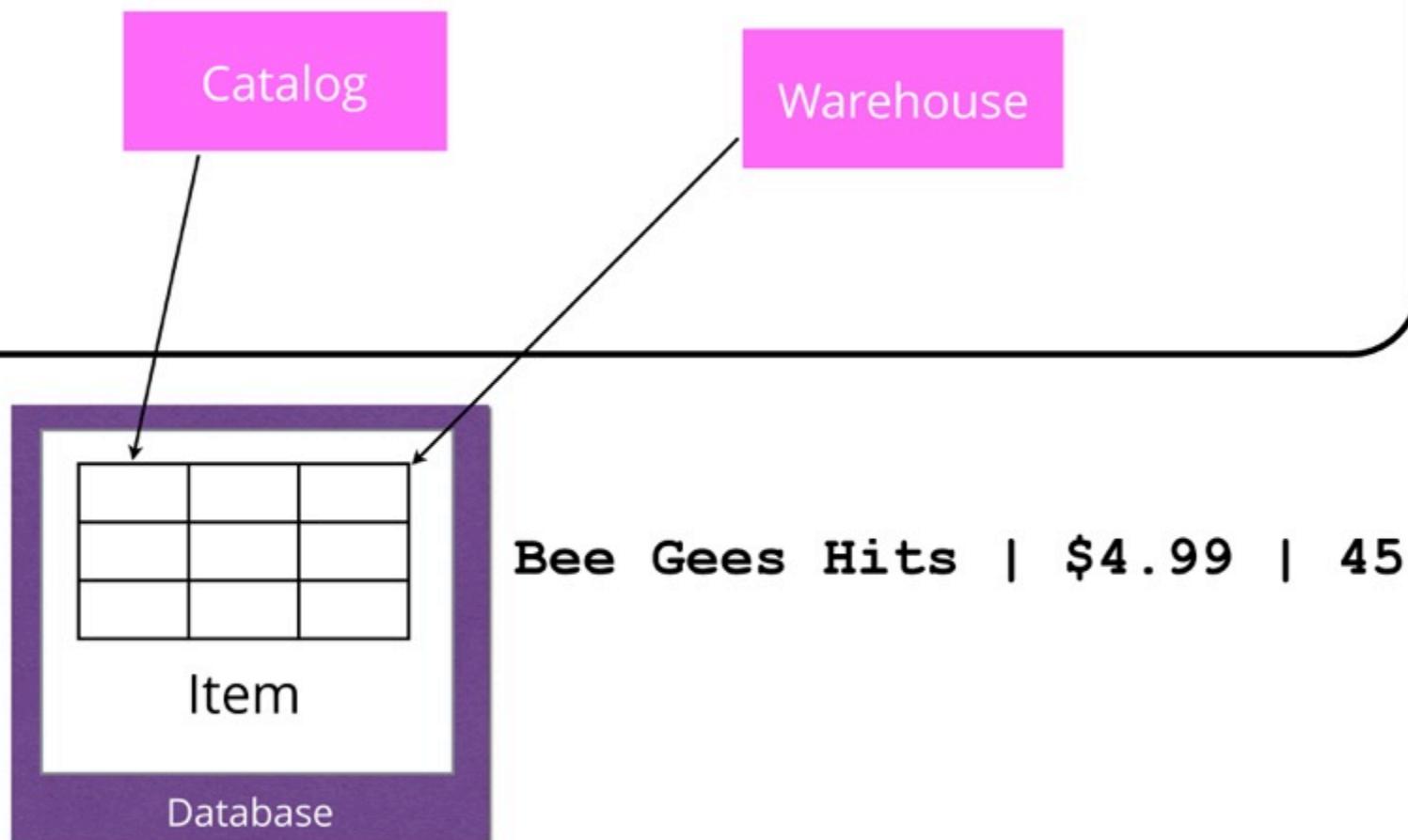
Use events to propagate changes

Event propagation isn't immediate

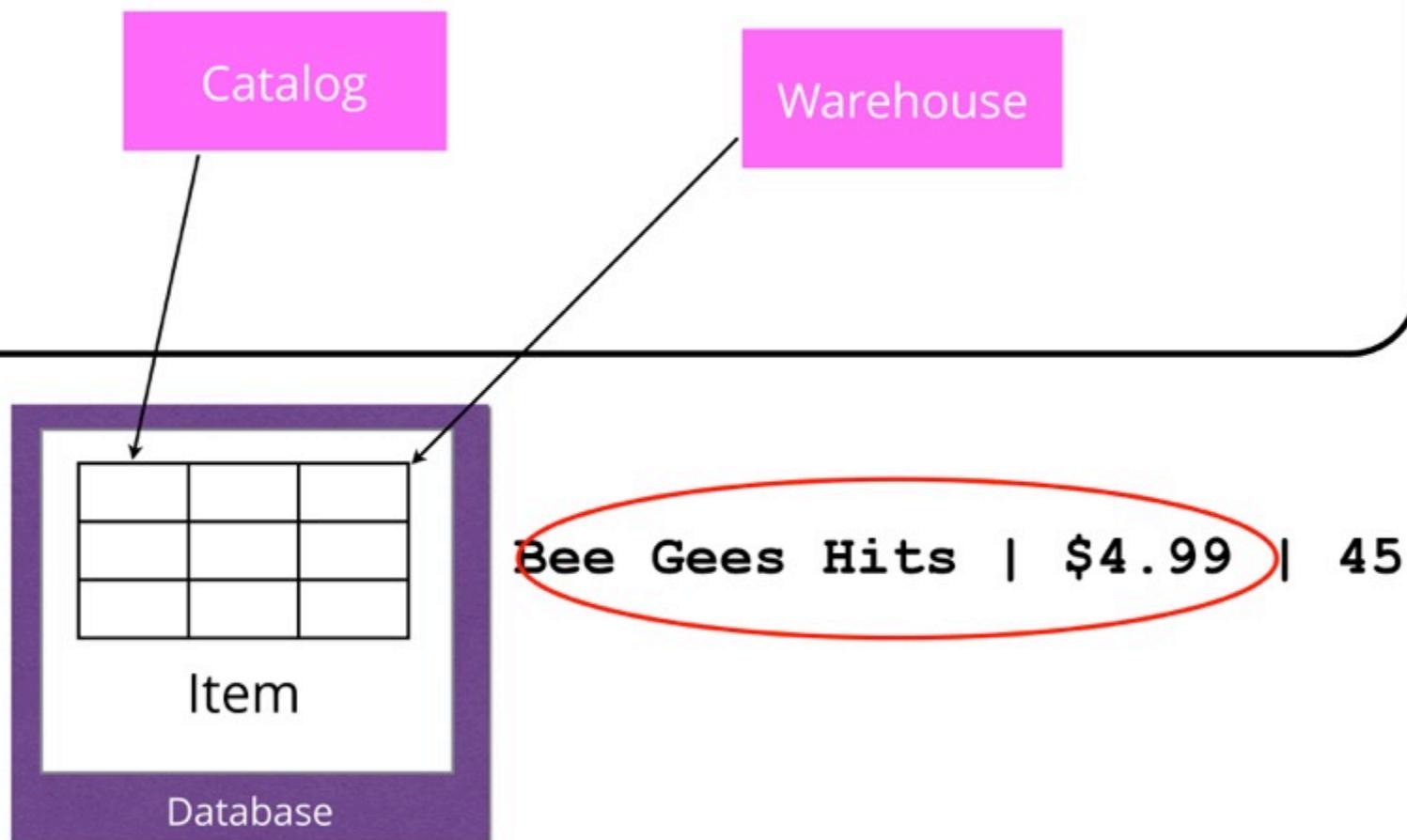
# MusikShop



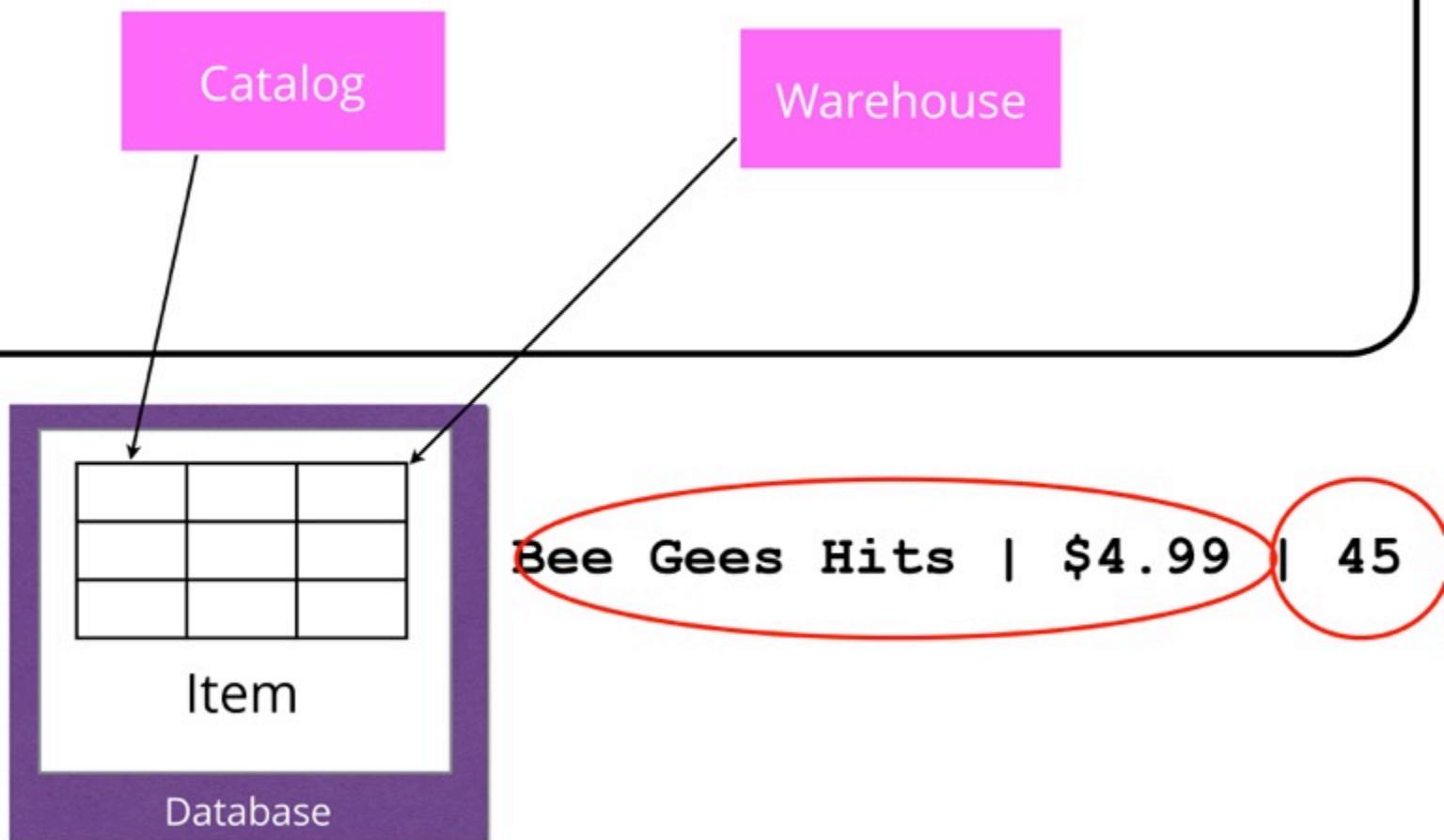
# MusikShop



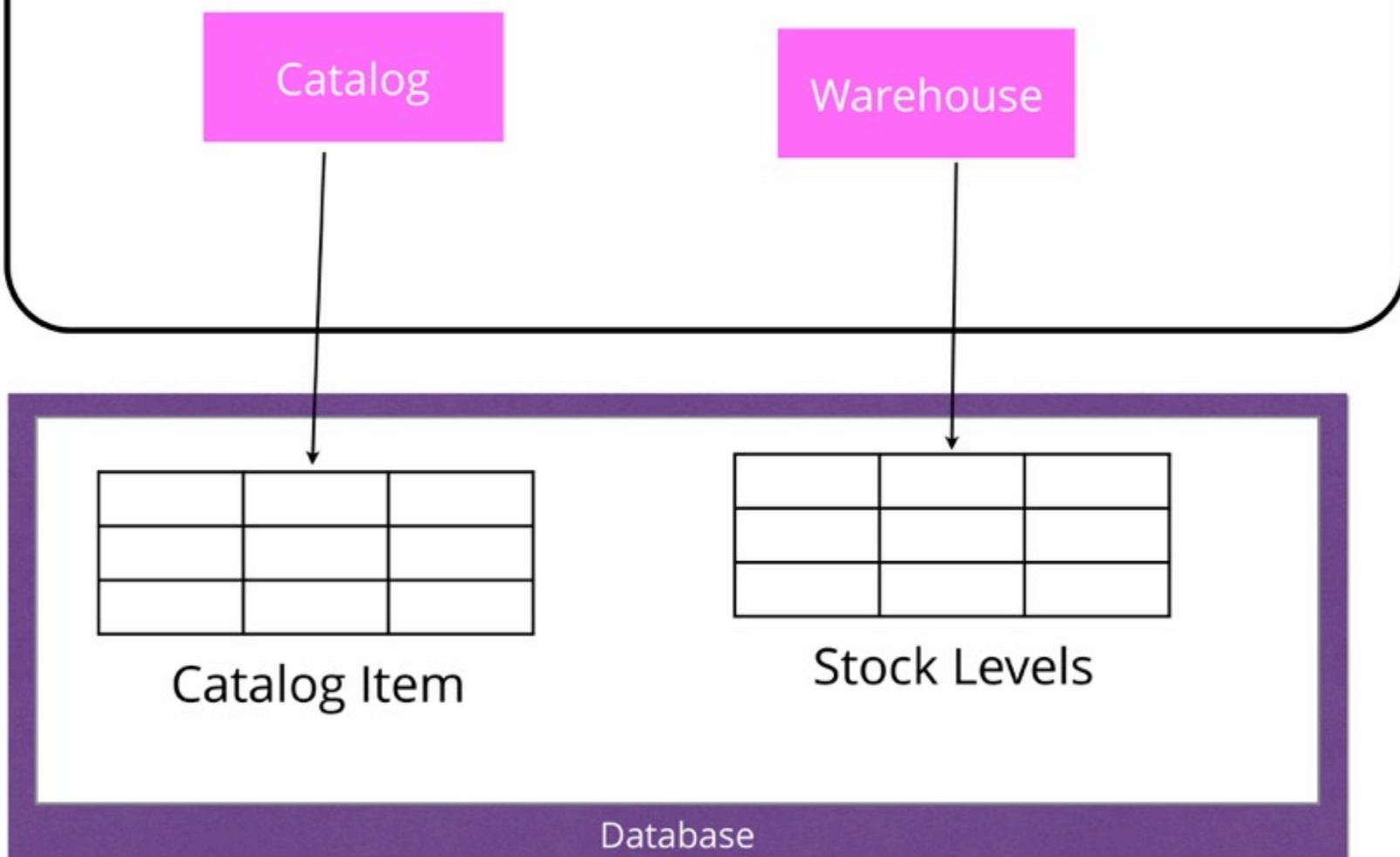
# MusikShop



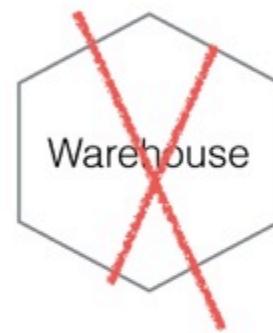
# MusikShop



# MusikShop



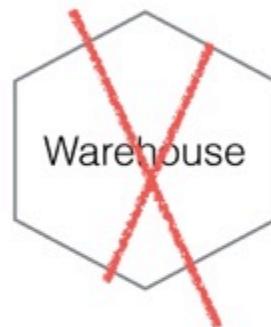




We can find out how  
much something costs...



We can find out how  
much something costs...



...and can take  
payment...

We can find out how  
much something costs...



...but can't check stock  
levels!



...and can take  
payment...

Should we take the money?

# CAP Theory

# CAP Theory

In a partition, you have to tradeoff between consistency and availability

Take the money?

Take the money?

Favouring availability over  
consistency

Don't take the money?

Don't take the money?

Favouring consistency over  
availability

This doesn't have to be  
black and white...

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago  
# Items: 100

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago     [Sell item](#)  
# Items: 100

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago      [Sell item](#)  
# Items: 100

Last view of stock: 1 hour ago  
# Items: 1

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago      [Sell item](#)  
# Items:                                100

Last view of stock: 1 hour ago      [Don't sell item](#)  
# Items:                                1

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago      [Sell item](#)  
# Items:                                100

Last view of stock: 1 hour ago      [Don't sell item](#)  
# Items:                                1

Last view of stock: 2 days ago  
# Items:                                100

This doesn't have to be  
black and white...

Last view of stock: 1 hour ago      [Sell item](#)  
# Items:                                100

Last view of stock: 1 hour ago      [Don't sell item](#)  
# Items:                                1

Last view of stock: 2 days ago      [Don't sell item](#)  
# Items:                                100

Distributed systems that require  
strong consistency are hard to  
scale

If you need strong consistency  
in parts of your system,  
consider offloading it to a  
system that supports it



## Turnable Consistency

<sup>1</sup>Consistency refers to how up-to-date and synchronized a row of Cassandra data is on all of its replicas. Cassandra extends the concept of [eventual consistency](#) by offering tunable consistency for any given read or write operation; the client application decides how consistent the requested data should be.

### Write Consistency Levels

The consistency level specifies the number of replicas on which the write must succeed before returning an acknowledgment to the client application.

Level	Description	Usage
ANY	A write must be written to at least one node. If all replica nodes for the given row key are down, the write can still succeed after a <a href="#">handed handoff</a> has been written. If all replica nodes are down at write time, an ANY write is not readable until the replica nodes for that row have recovered.	Provides low latency and a guarantee that a write never fails. Delivers the lowest consistency and highest availability compared to other levels.
ALL	A write must be written to the commit log and memory table on all replica nodes in the cluster for that row.	Provides the highest consistency and the lowest availability of all other levels.

[https://teddyma.gitbooks.io/learn cassandra/content/replication/turnable\\_consistency.html](https://teddyma.gitbooks.io/learn cassandra/content/replication/turnable_consistency.html)