ThoughtWorks®

# PRINCIPLES OF MICROSERVICES

---

*Sam Newman*

*Velocity Santa Clara, May 2015*

ThoughtWorks®

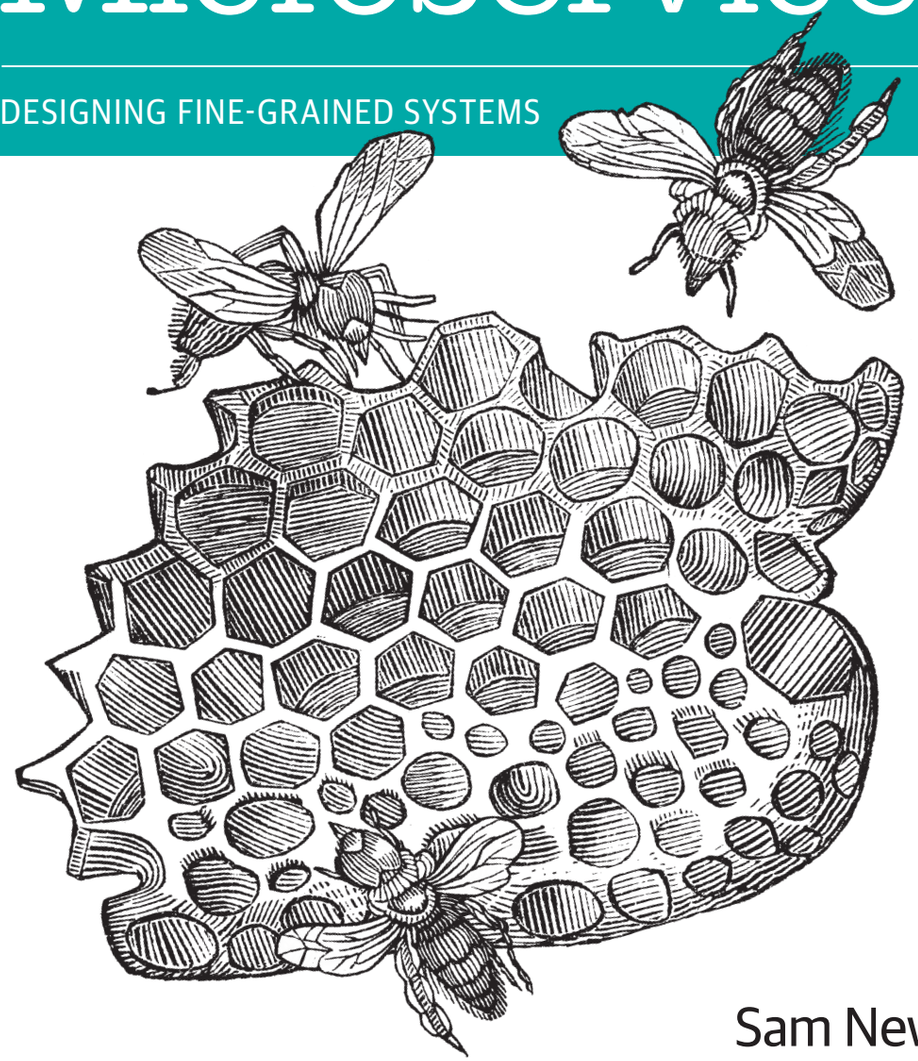
@velocityconf

@samnewman

O'REILLY®

# Building Microservices

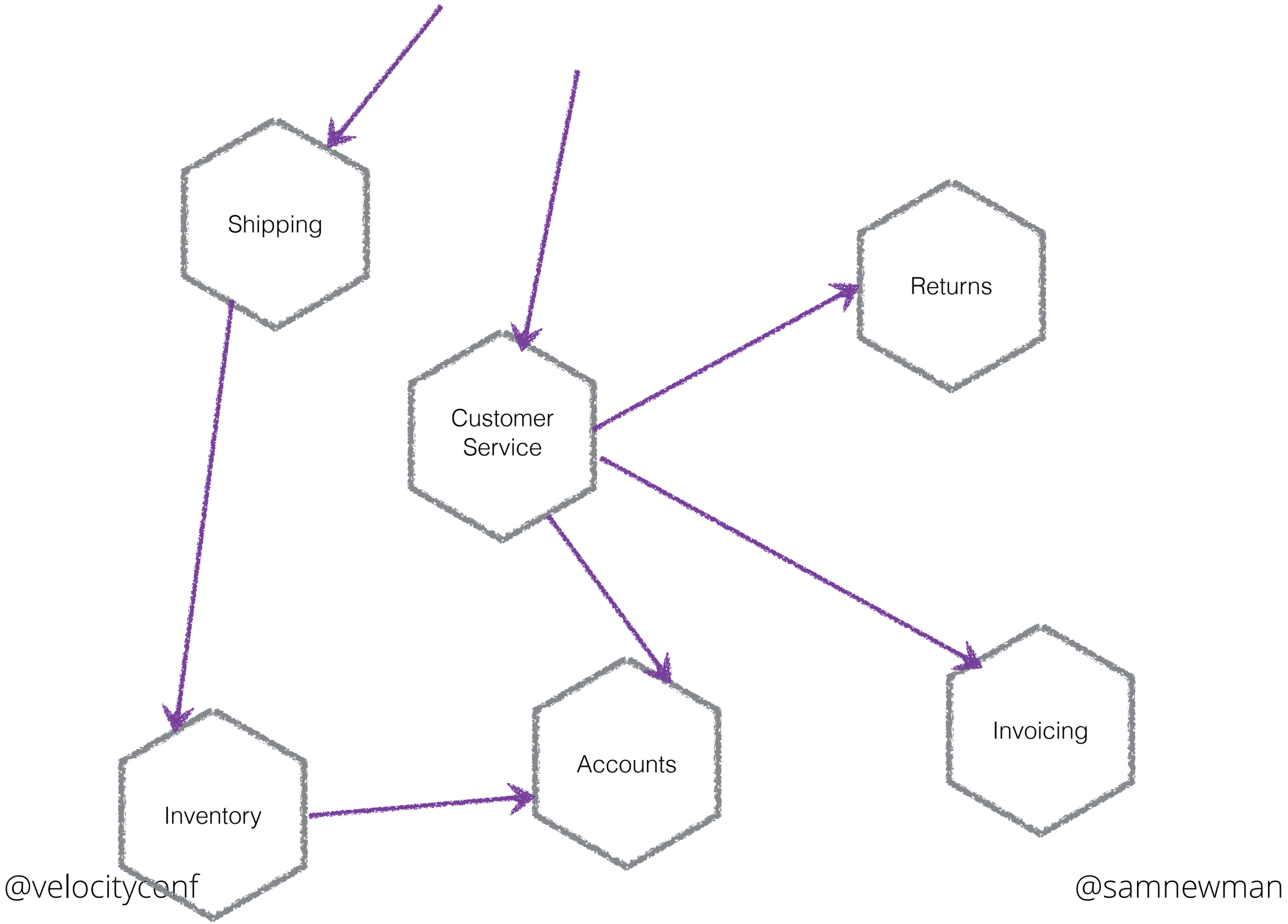
DESIGNING FINE-GRAINED SYSTEMS



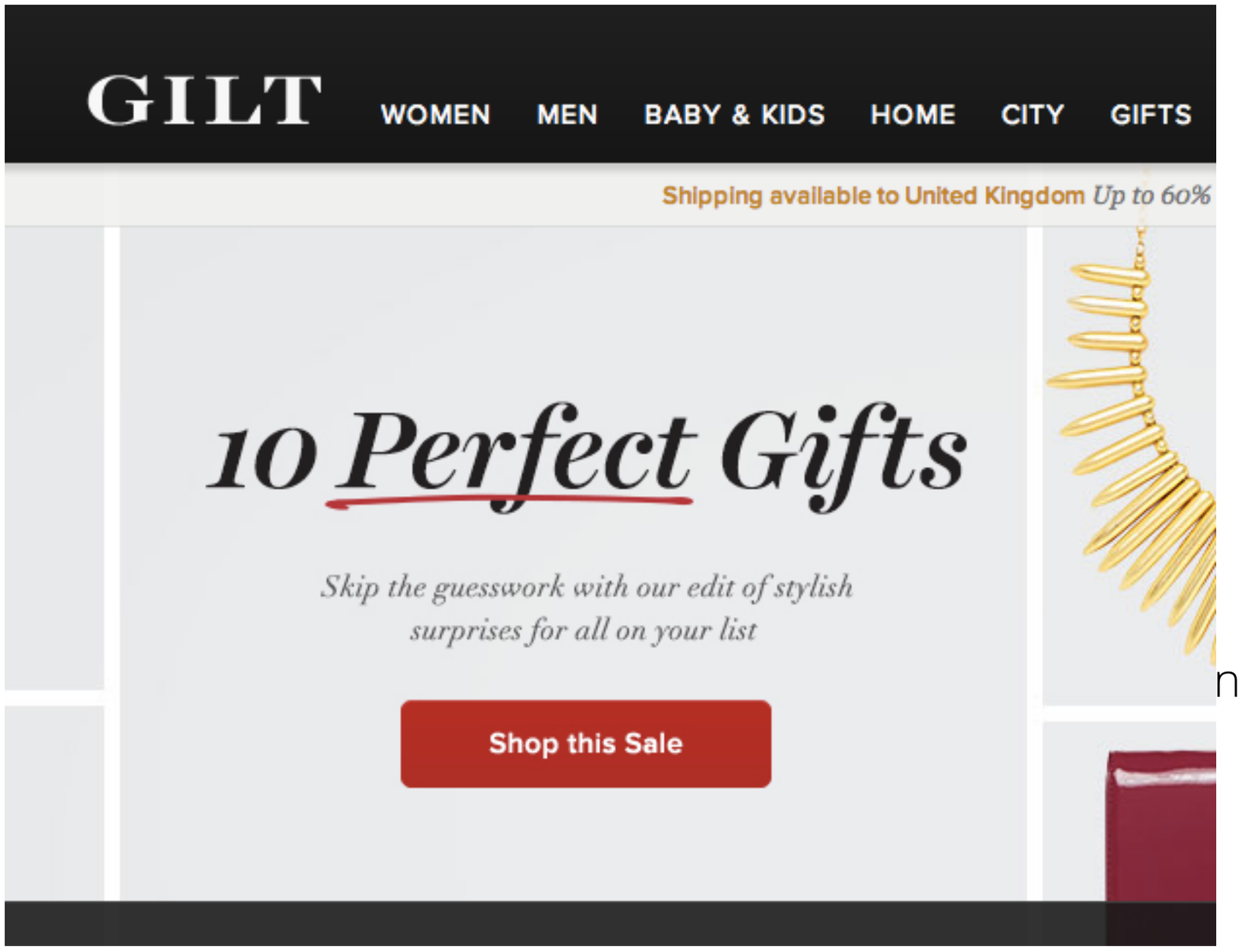
Sam Newman

@velocityconf

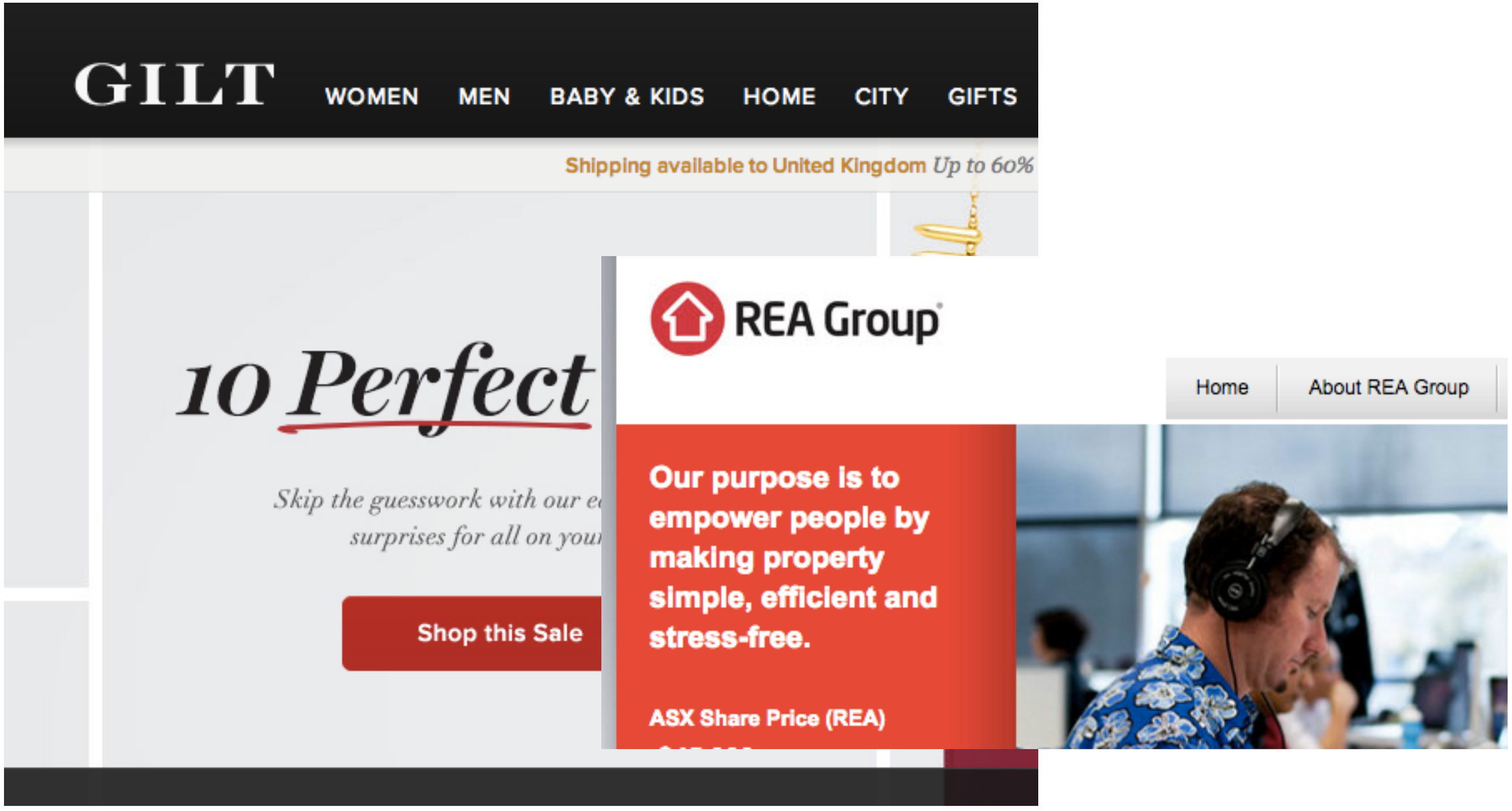
@samnewman

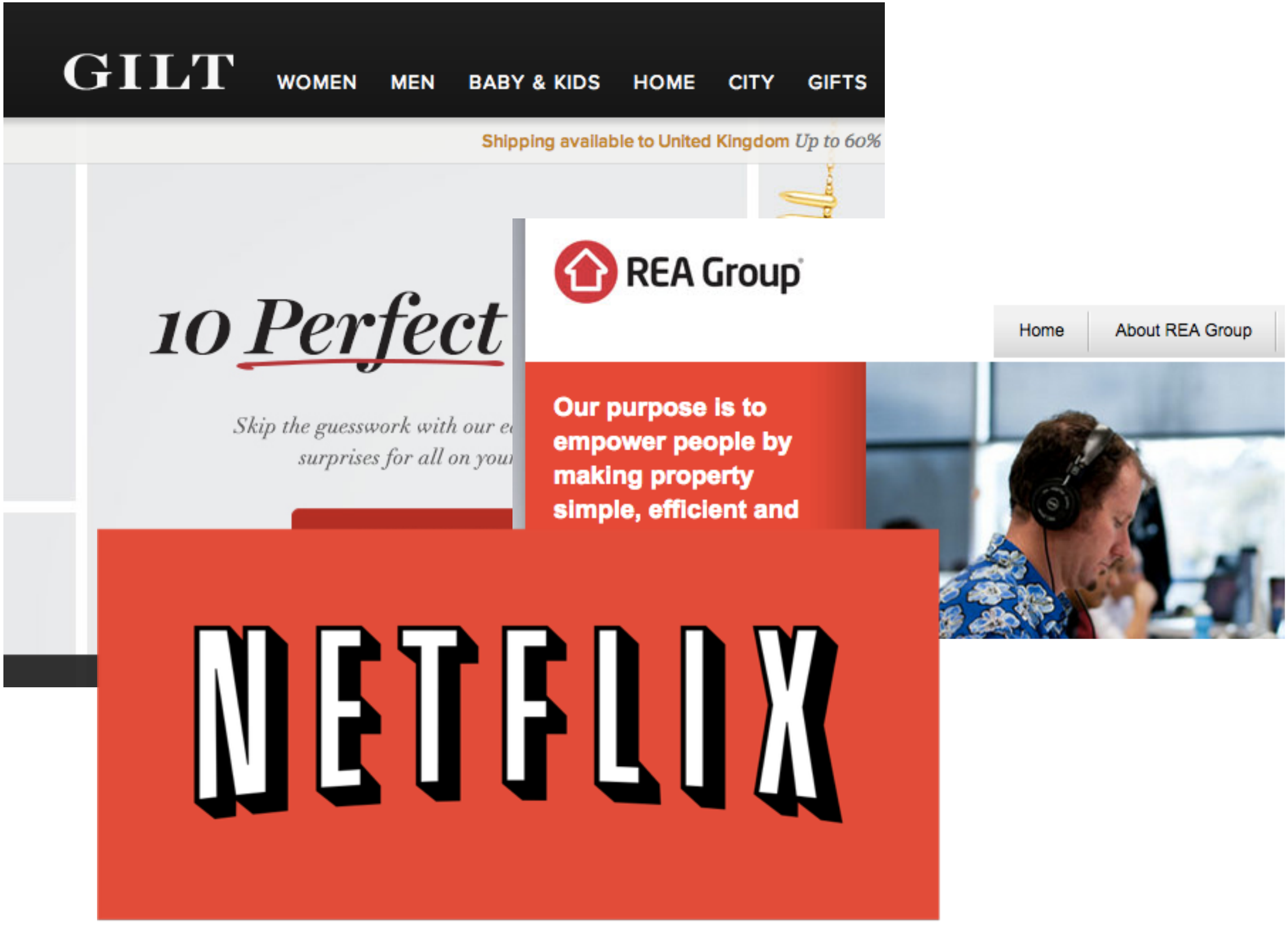

Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


# Small ***Autonomous*** services that ***work together***


GILT
WOMEN
MEN
BABY & KIDS
HOME
CITY
GIFTS
Shipping available to United Kingdom Up to 60%
10 Perfect Gifts
Skip the guesswork with our edit of stylish
surprises for all on your list
Shop this Sale

n


GILT
WOMEN
MEN
BABY & KIDS
HOME
CITY
GIFTS
Shipping available to United Kingdom Up to 60%
10 Perfect
Skip the guesswork with our e
surprises for all on you
Shop this Sale
REA Group
Home
About REA Group
Our purpose is to empower people by making property simple, efficient and stress-free.
ASX Share Price (REA)


GILT
WOMEN
MEN
BABY & KIDS
HOME
CITY
GIFTS
Shipping available to United Kingdom Up to 60%
10 Perfect
Skip the guesswork with our e
surprises for all on you
REA Group
Home
About REA Group
Our purpose is to empower people by making property simple, efficient and
NETFLIX

# The Twelve Factors

**I. Codebase**
One codebase tracked in revision control, many deploys

**II. Dependencies**
Explicitly declare and isolate dependencies

**III. Config**
Store config in the environment

**IV. Backing Services**
Treat backing services as attached resources

**V. Build, release, run**
Strictly separate build and run stages

**VI. Processes**
Execute the app as one or more stateless processes

**VII. Port binding**
Export services via port binding

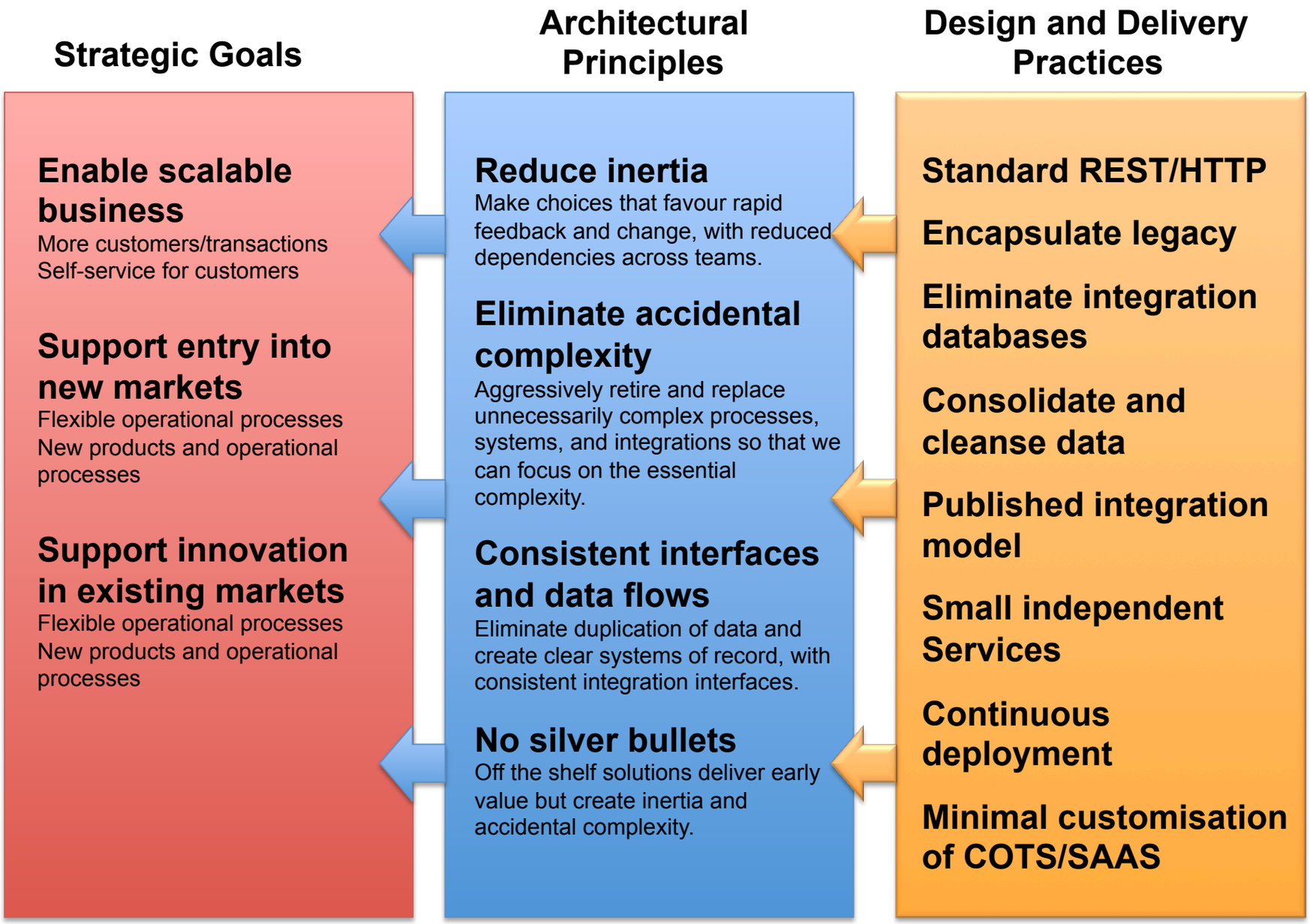
**VIII. Concurrency**
Scale out via the process model

**IX. Disposability**
Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**
Keep development, staging, and production as similar as possible

**XI. Logs**
Treat logs as event streams

**XII. Admin processes**
Run admin/management tasks as one-off processes

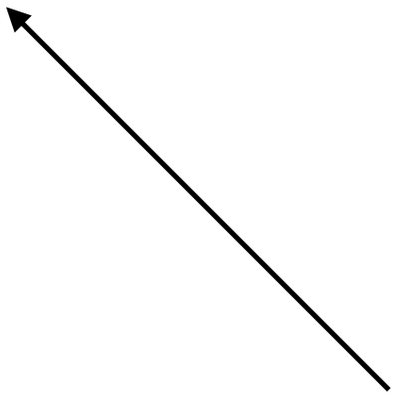

Strategic Goals
Enable scalable business
More customers/transactions
Self-service for customers
Support entry into new markets
Flexible operational processes
New products and operational processes
Support innovation in existing markets
Flexible operational processes
New products and operational processes
Architectural Principles
Reduce inertia
Make choices that favour rapid feedback and change, with reduced dependencies across teams.
Eliminate accidental complexity
Aggressively retire and replace unnecessarily complex processes, systems, and integrations so that we can focus on the essential complexity.
Consistent interfaces and data flows
Eliminate duplication of data and create clear systems of record, with consistent integration interfaces.
No silver bullets
Off the shelf solutions deliver early value but create inertia and accidental complexity.
Design and Delivery Practices
Standard REST/HTTP
Encapsulate legacy
Eliminate integration databases
Consolidate and cleanse data
Published integration model
Small independent Services
Continuous deployment
Minimal customisation of COTS/SAAS

Small ***Autonomous*** services

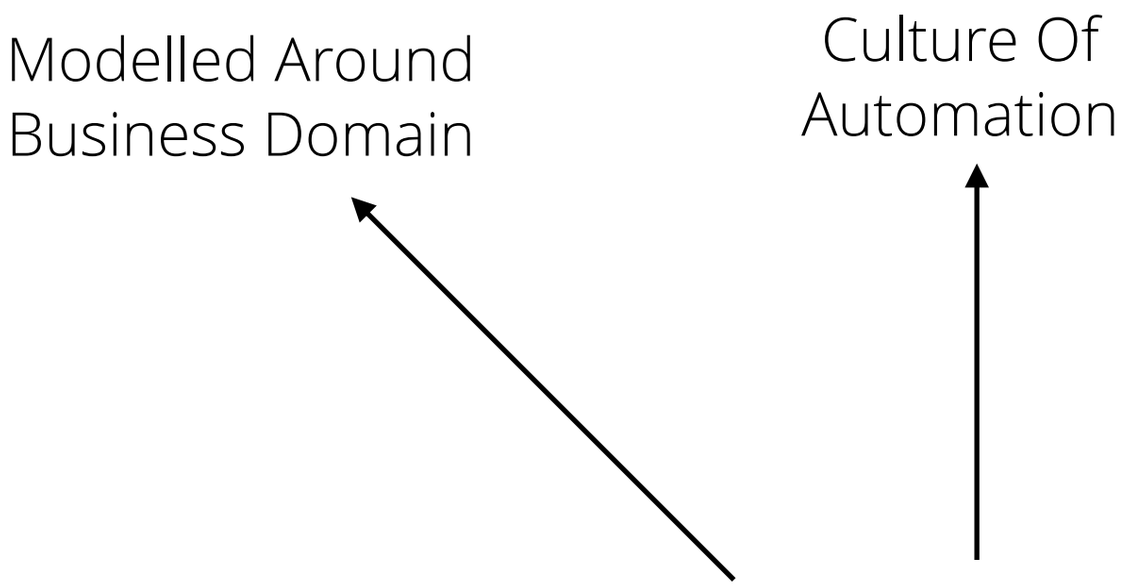
that ***work together***

# Principles Of Microservices

Modelled Around
Business Domain



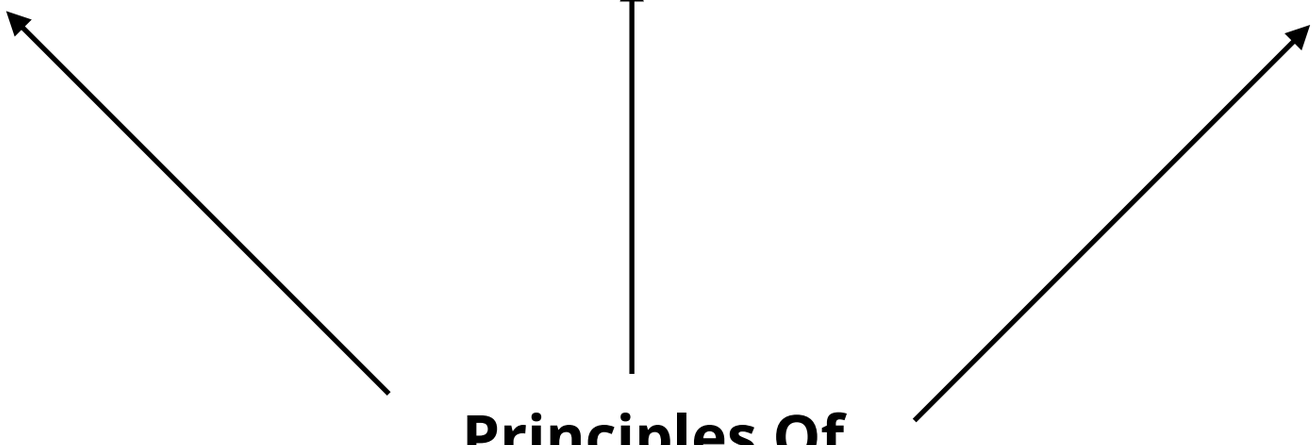
**Principles Of Microservices**


Modelled Around
Business Domain
Culture Of
Automation
Principles Of
Microservices

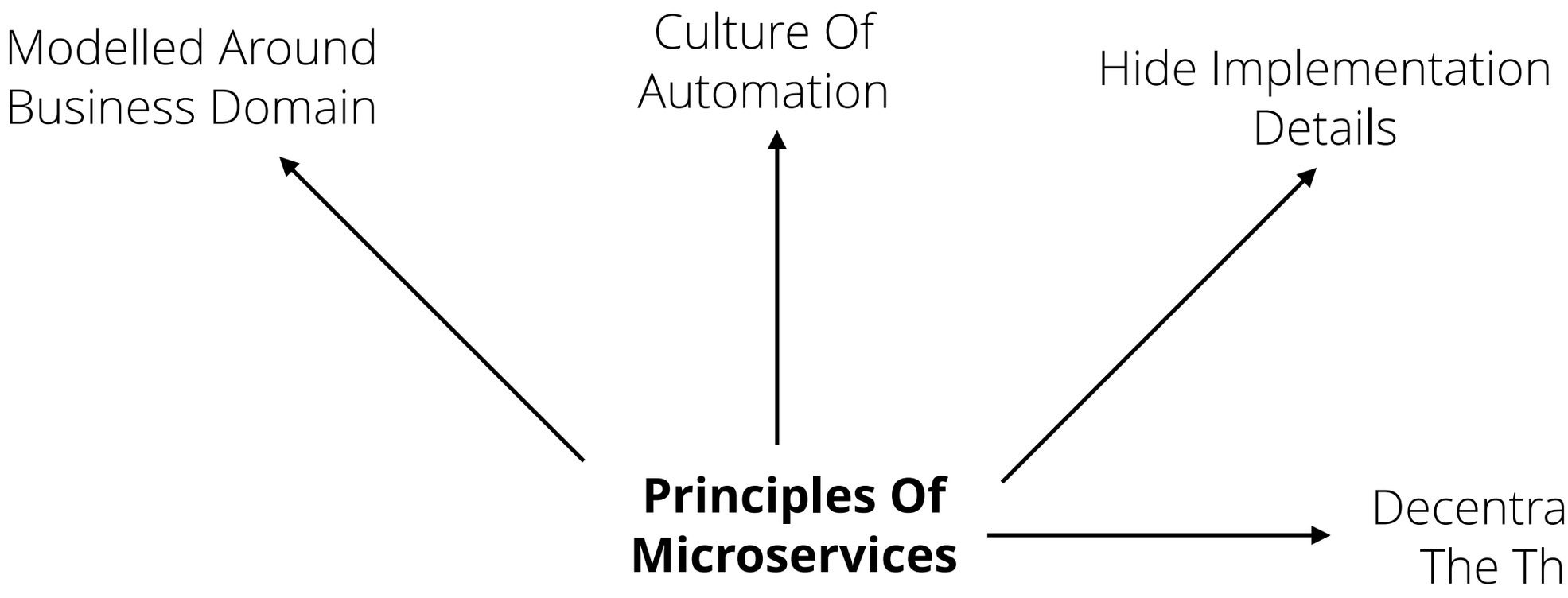
Modelled Around Business Domain

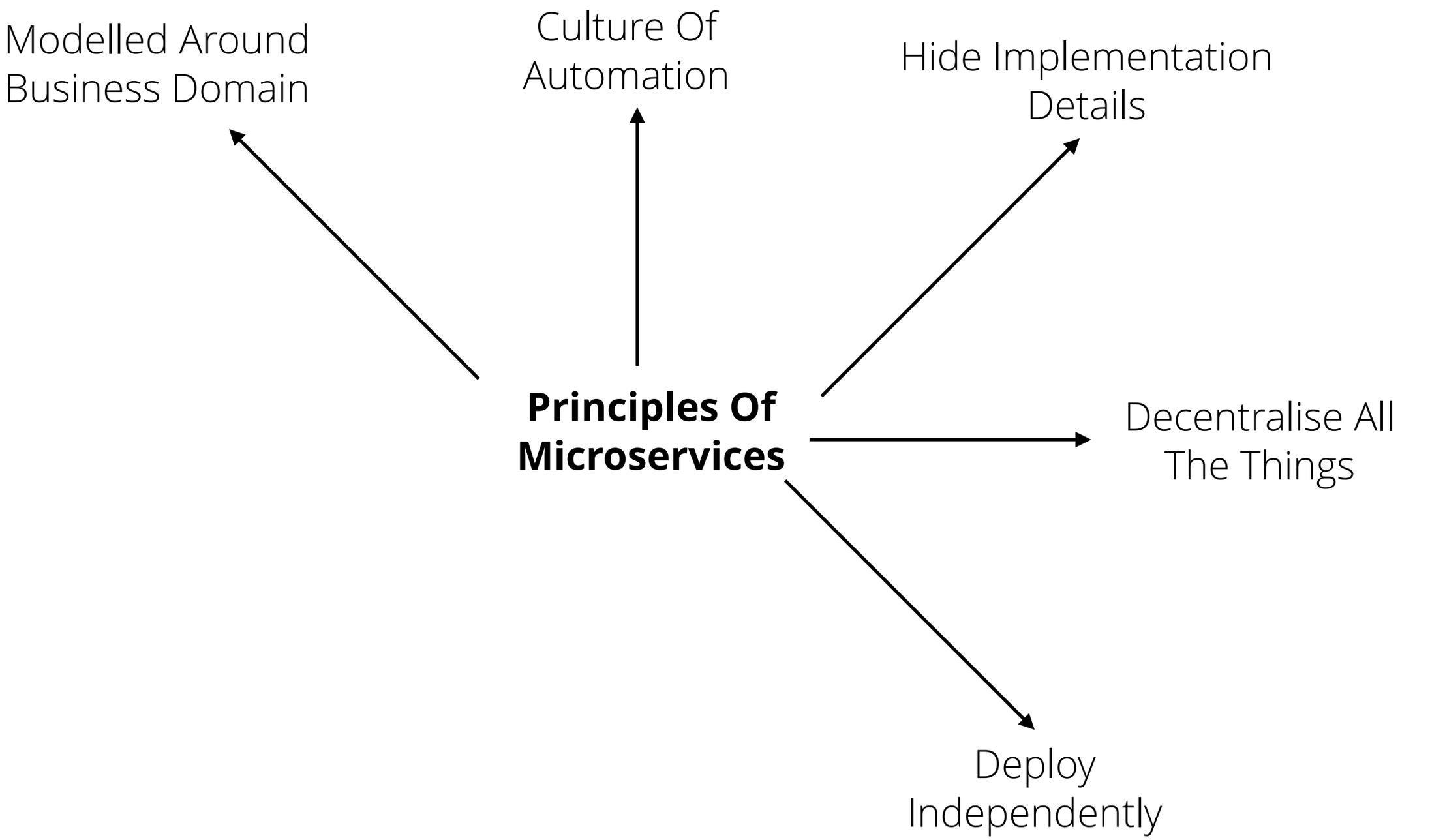
Culture Of Automation

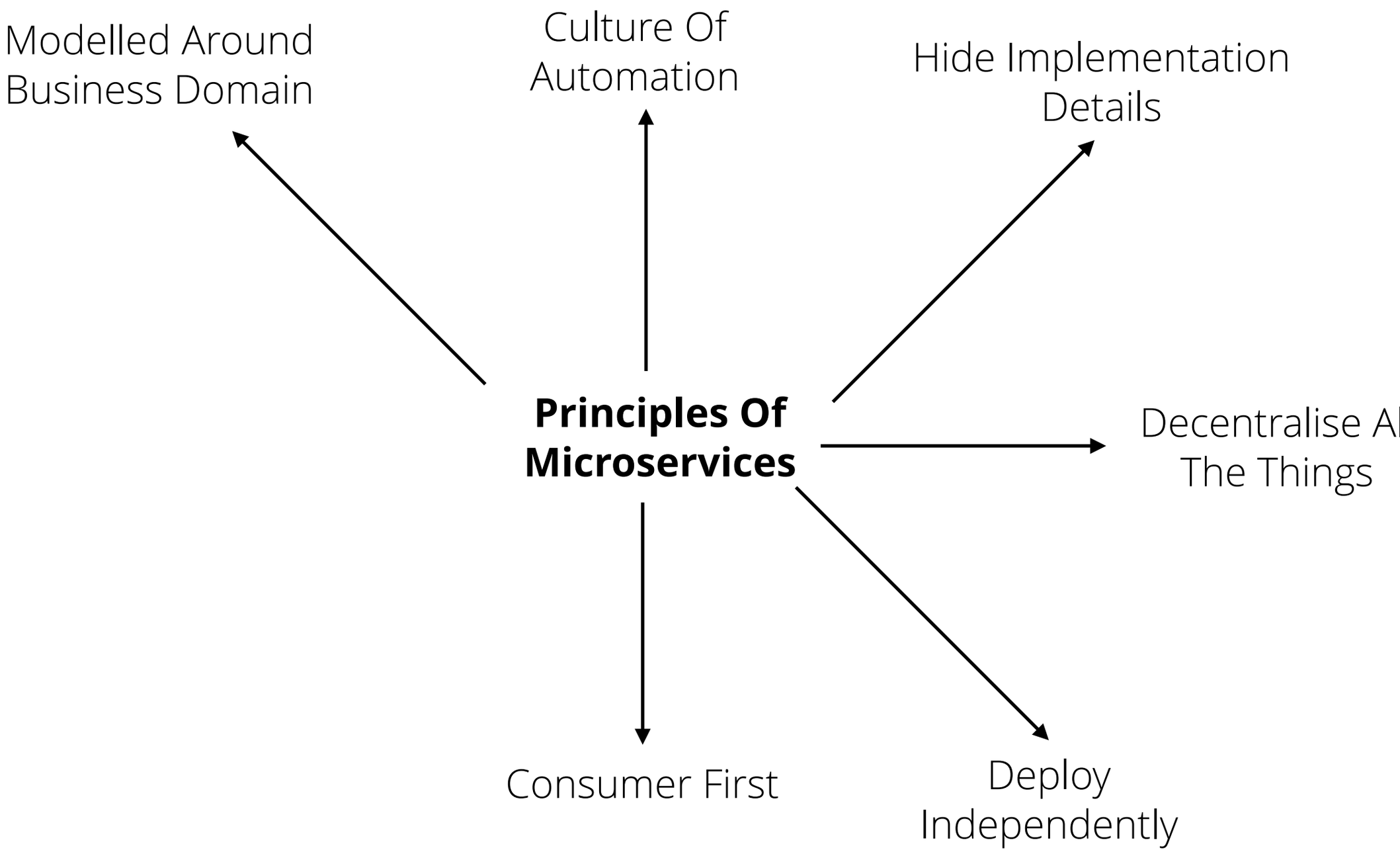
Hide Implementation Details



**Principles Of Microservices**


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things
Deploy Independently


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things
Consumer First
Deploy Independently

Modelled Around Business Domain

Culture Of Automation

Hide Implementation Details

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

Modelled Around Business Domain

Culture Of Automation

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

Modelled Around Business Domain

Culture Of Automation

Hide Implementation Details

Highly Observable

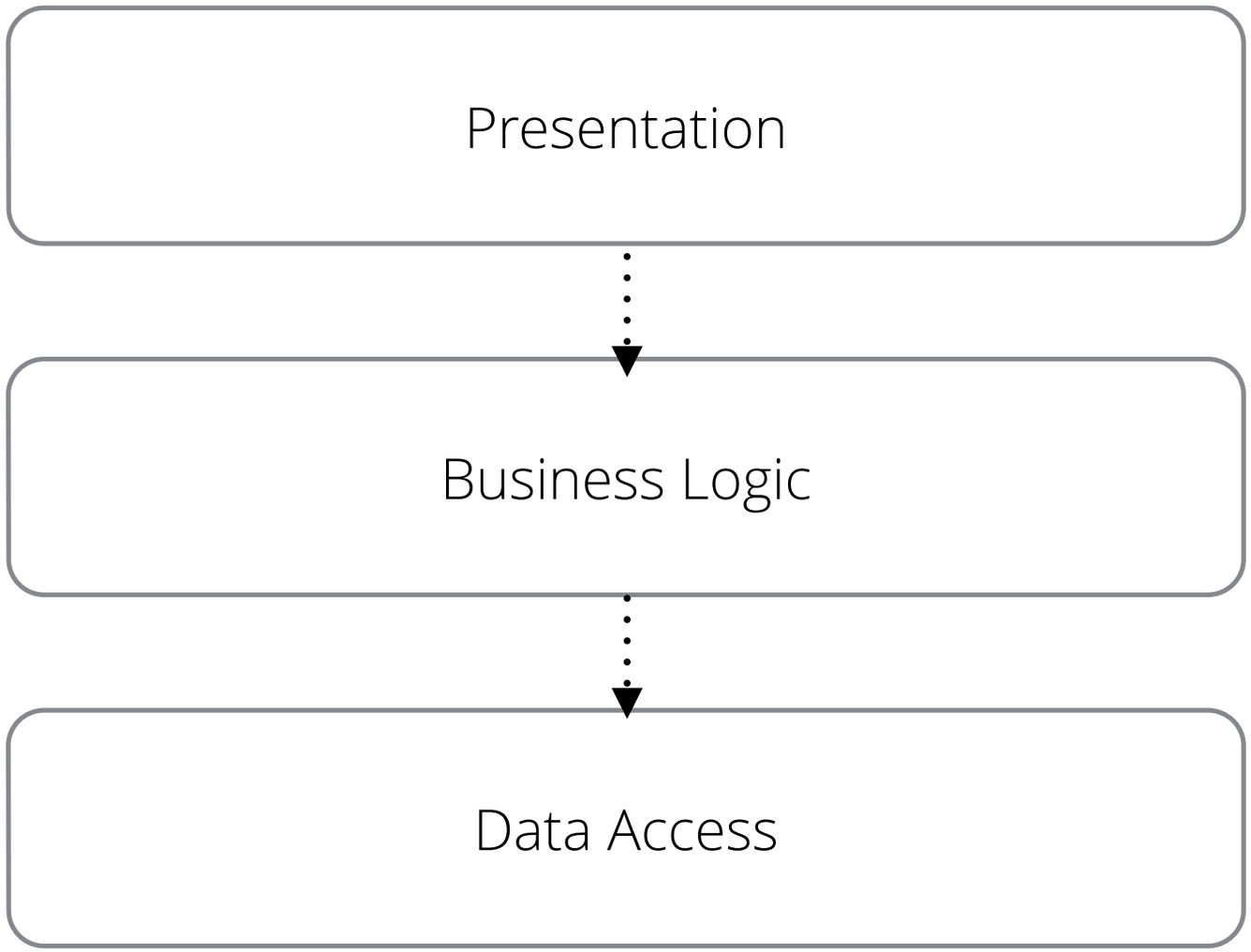
**Principles Of Microservices**

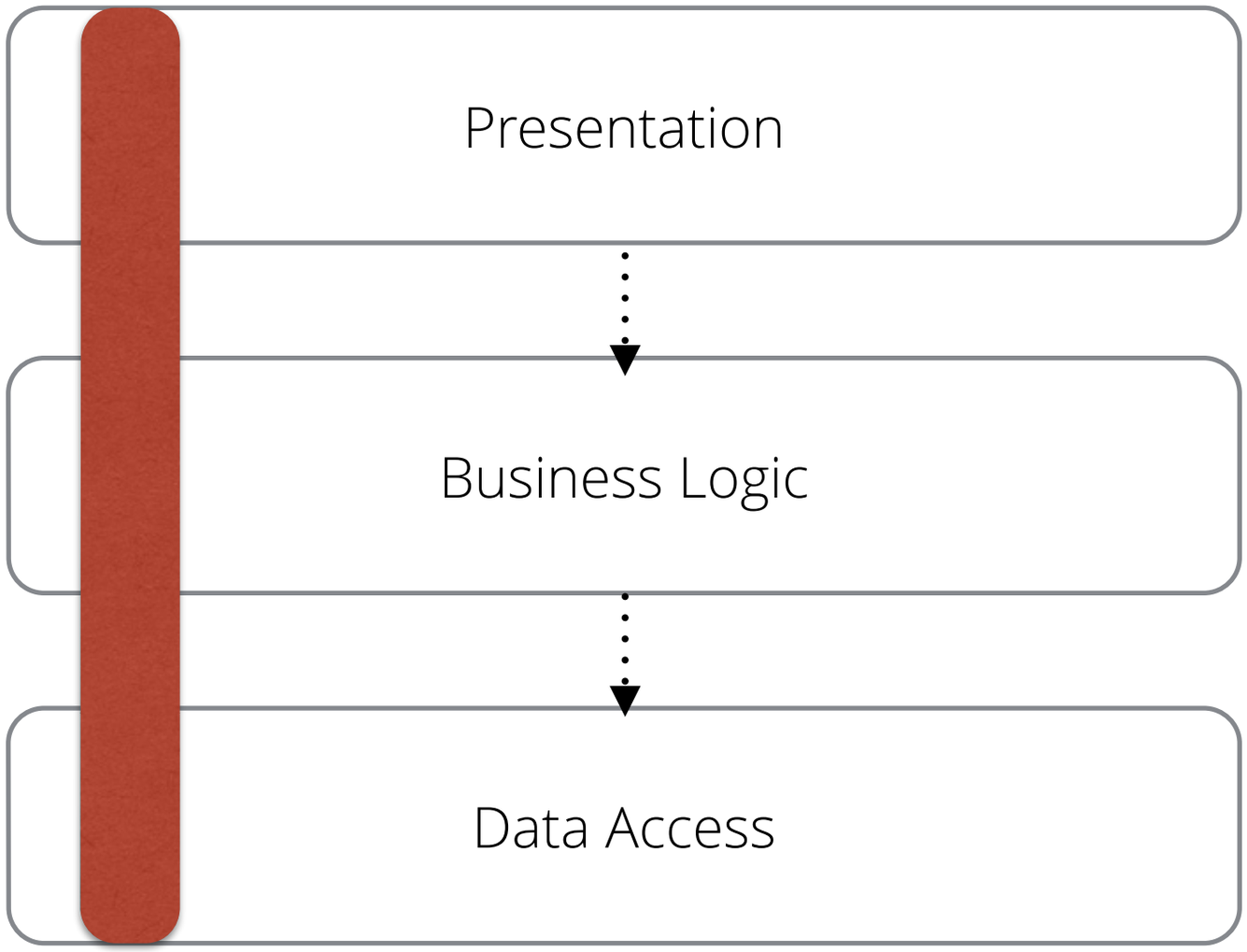
Decentralise All The Things

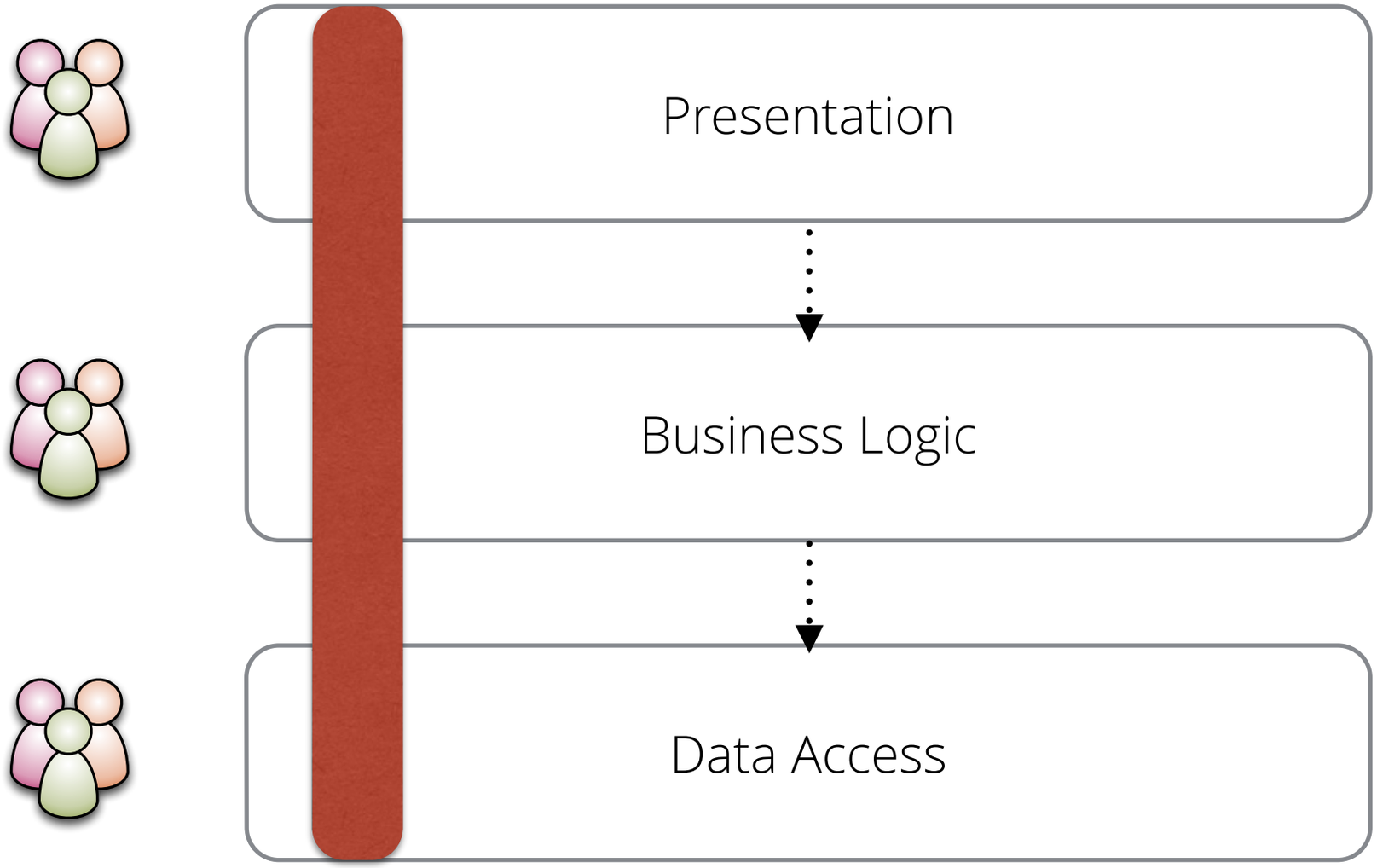
Isolate Failure

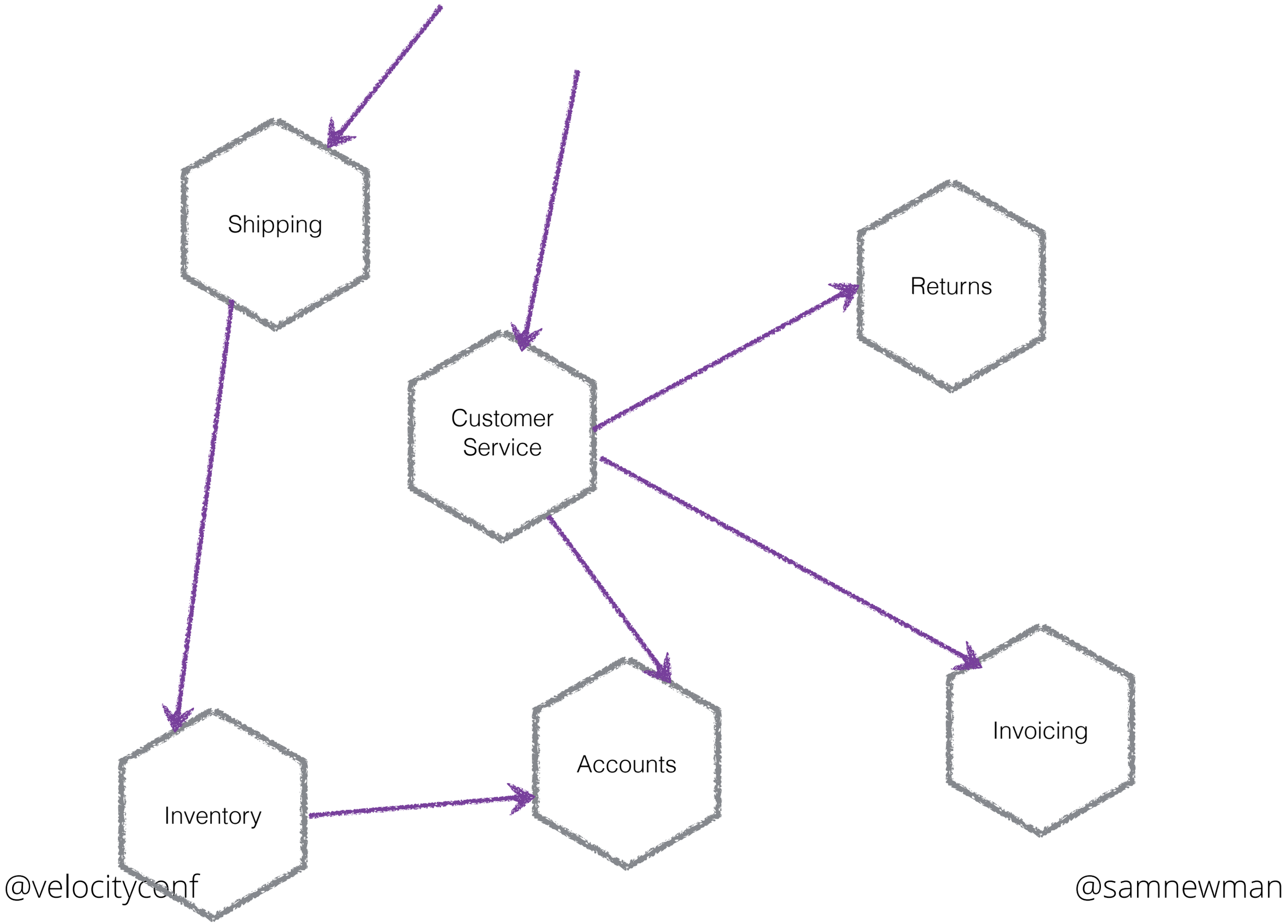
Consumer First

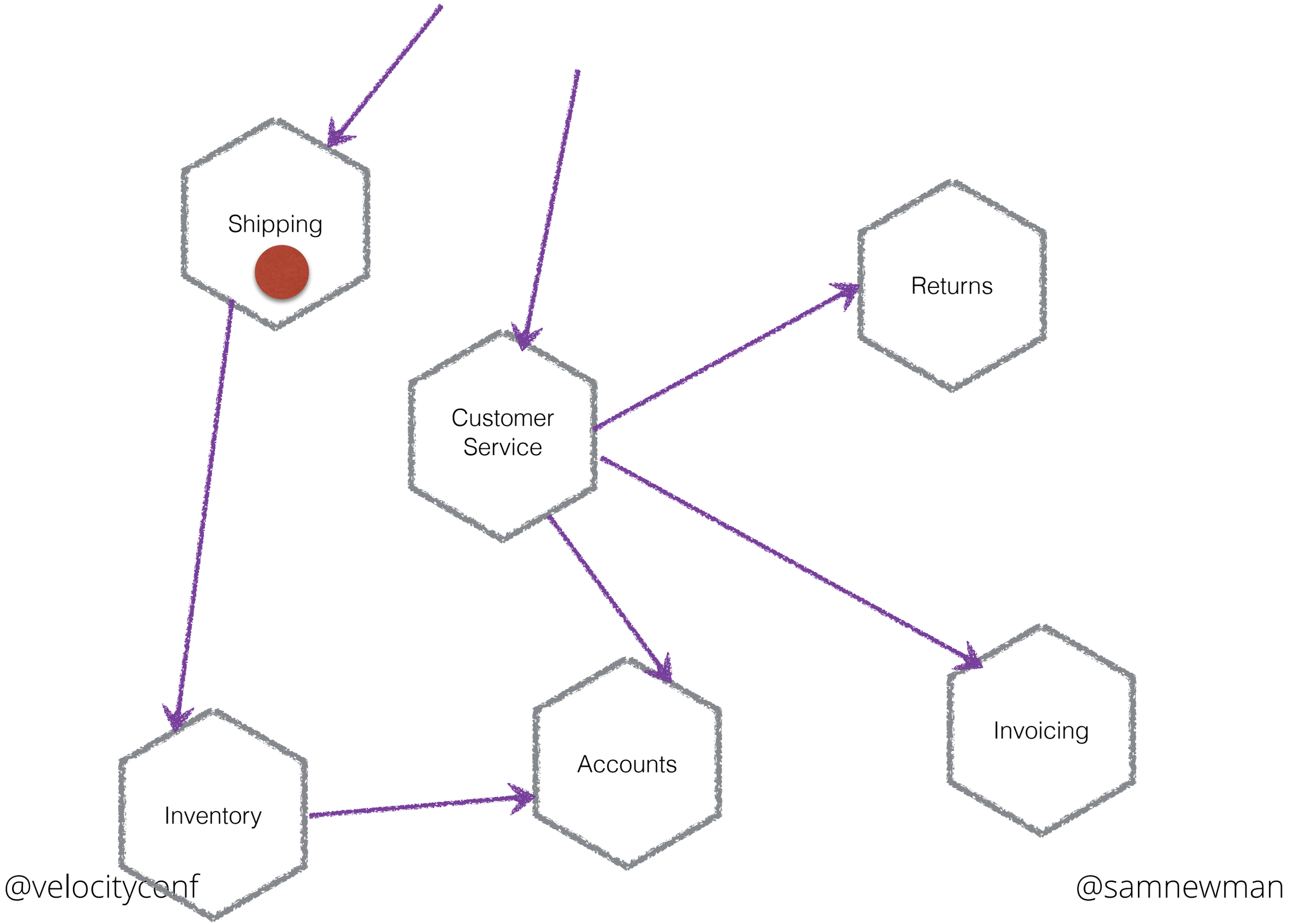
Deploy Independently

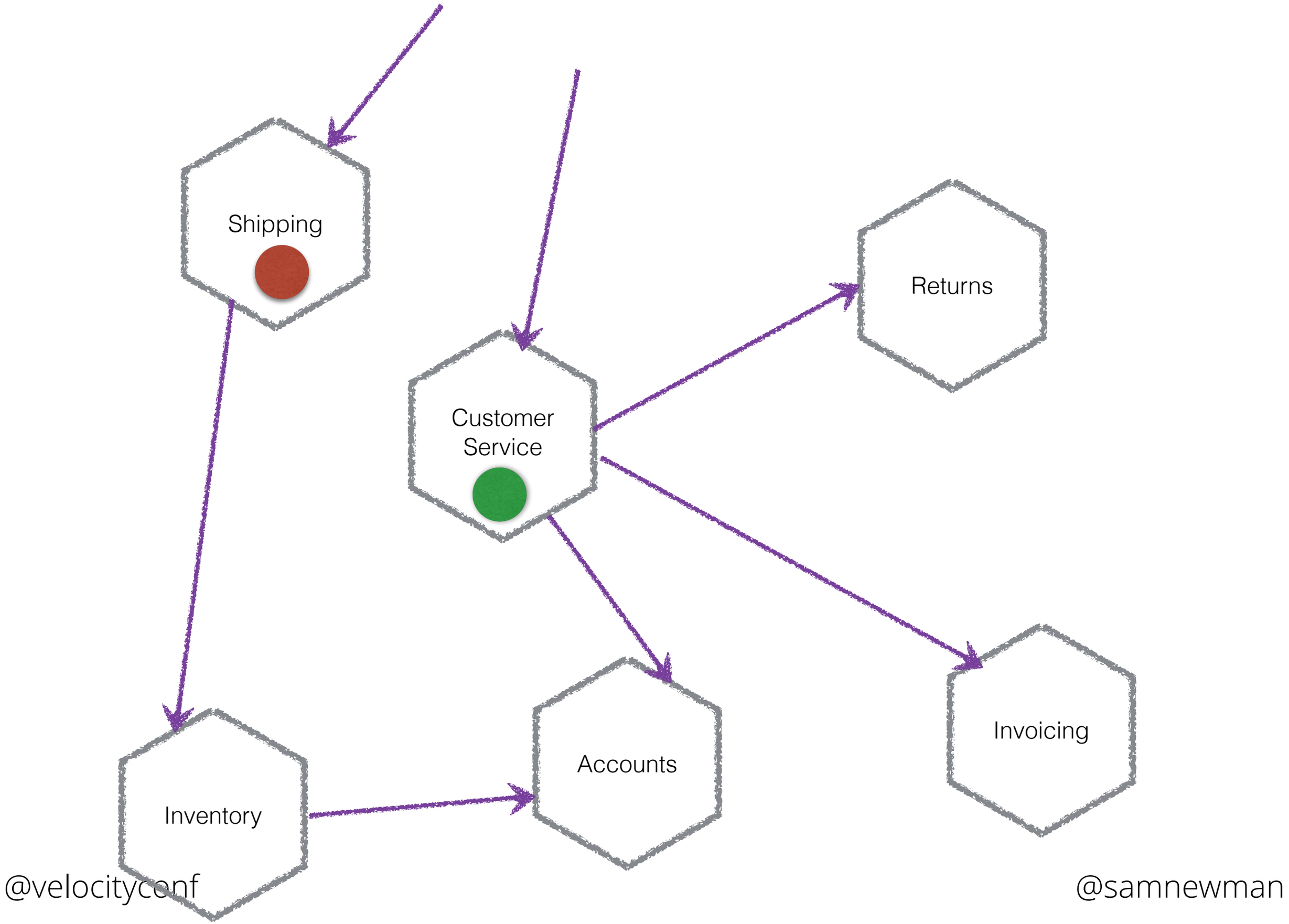

Presentation
Business Logic
Data Access

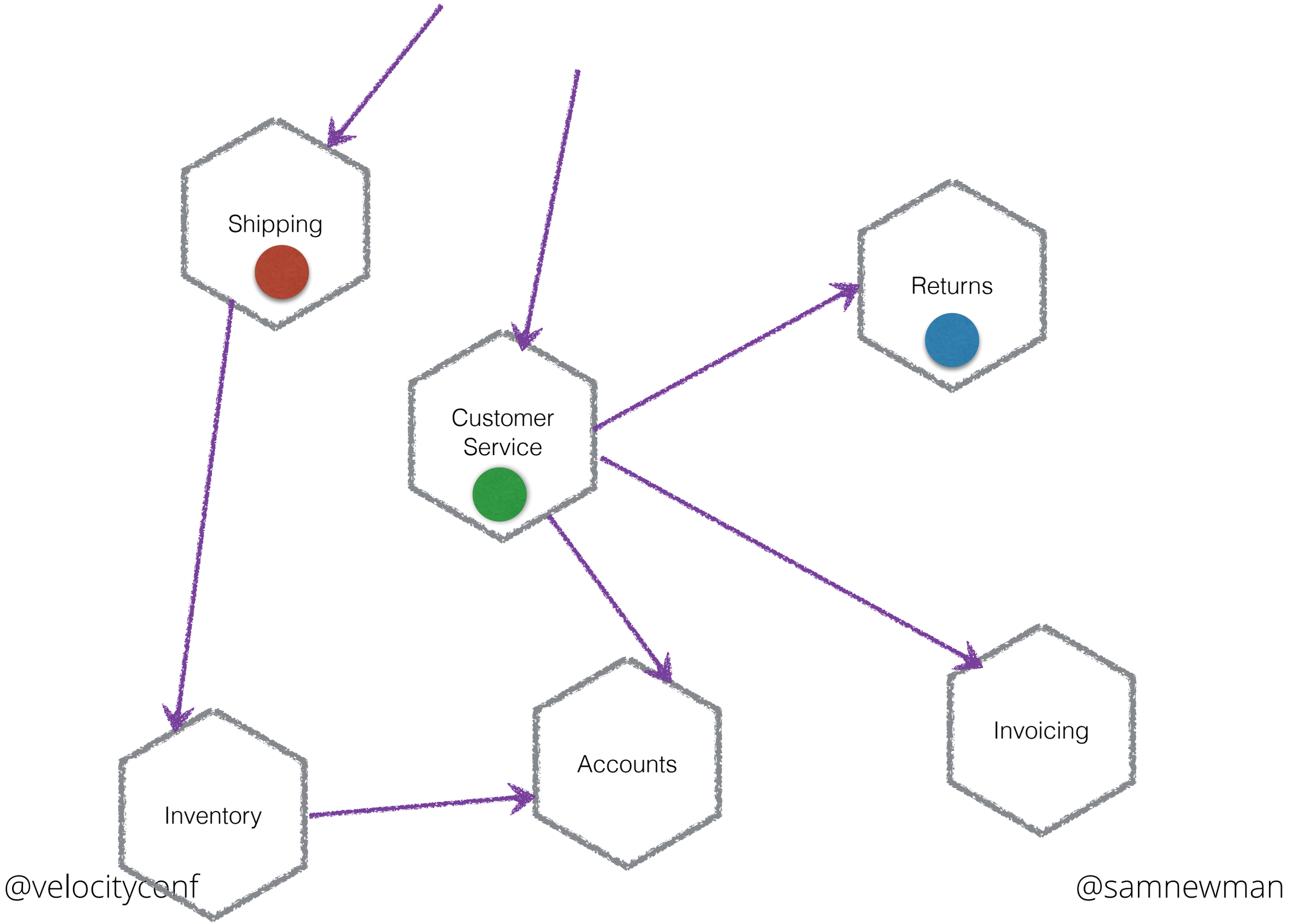

Presentation
Business Logic
Data Access

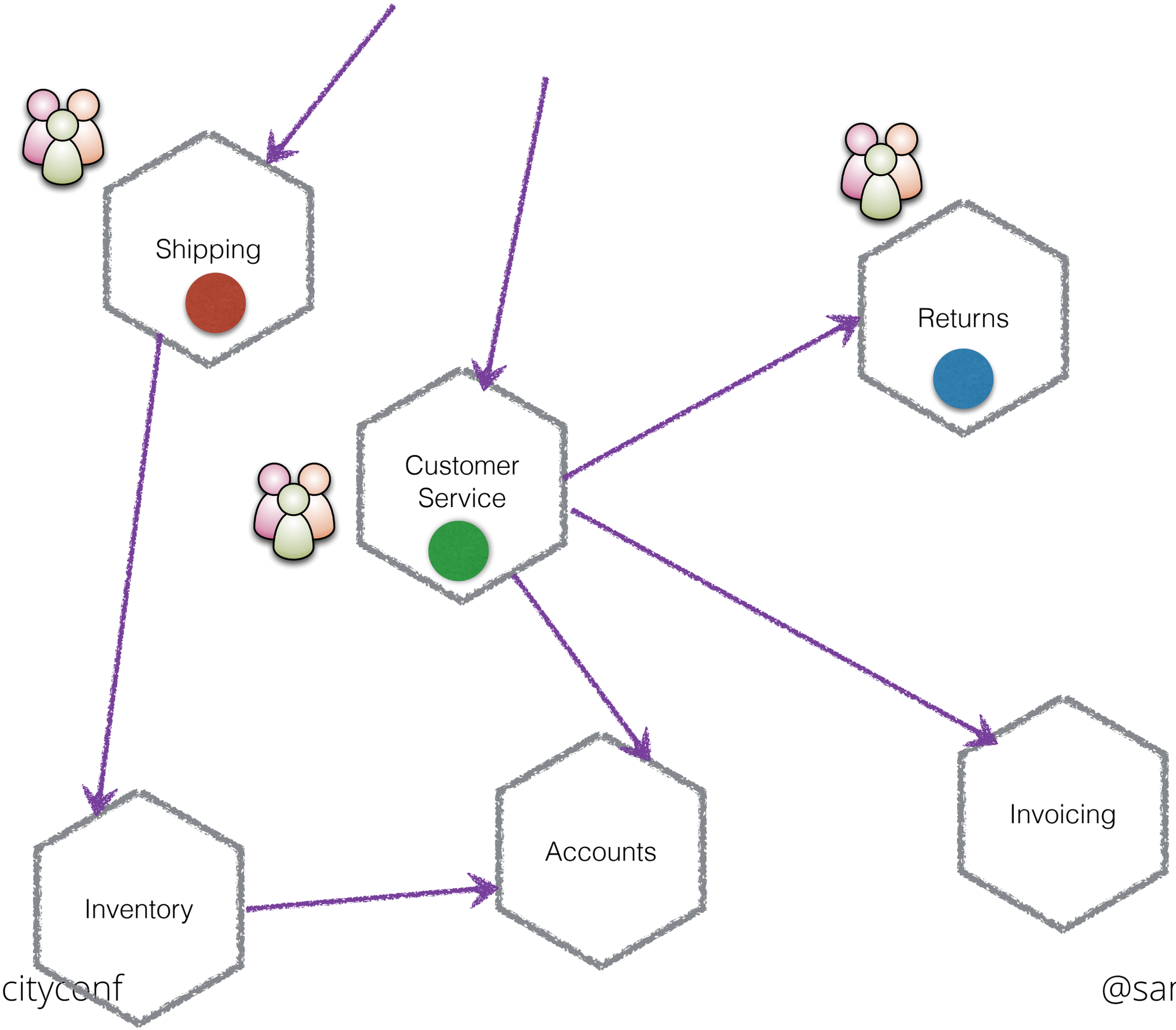

Presentation
Business Logic
Data Access


Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


Shipping
Customer Service
Returns
Inventory
Accounts
Invoicing


https://www.flickr.com/photos/glimeend/5692801655/

# Register a new customer

Register a new customer

Oil Change

Register a new customer

Oil Change

Emission Test

Register a new customer

Oil Change

Emission Test

Replace Tire

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Send an invoice

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Make a repair

Send an invoice

Register a new customer

Oil Change

Emission Test

Replace Tire

Contact a customer

Order parts

Send an invoice

Make a repair

Register a new customer

Contact a customer

Send an invoice

Oil Change

Make a repair

Replace Tire

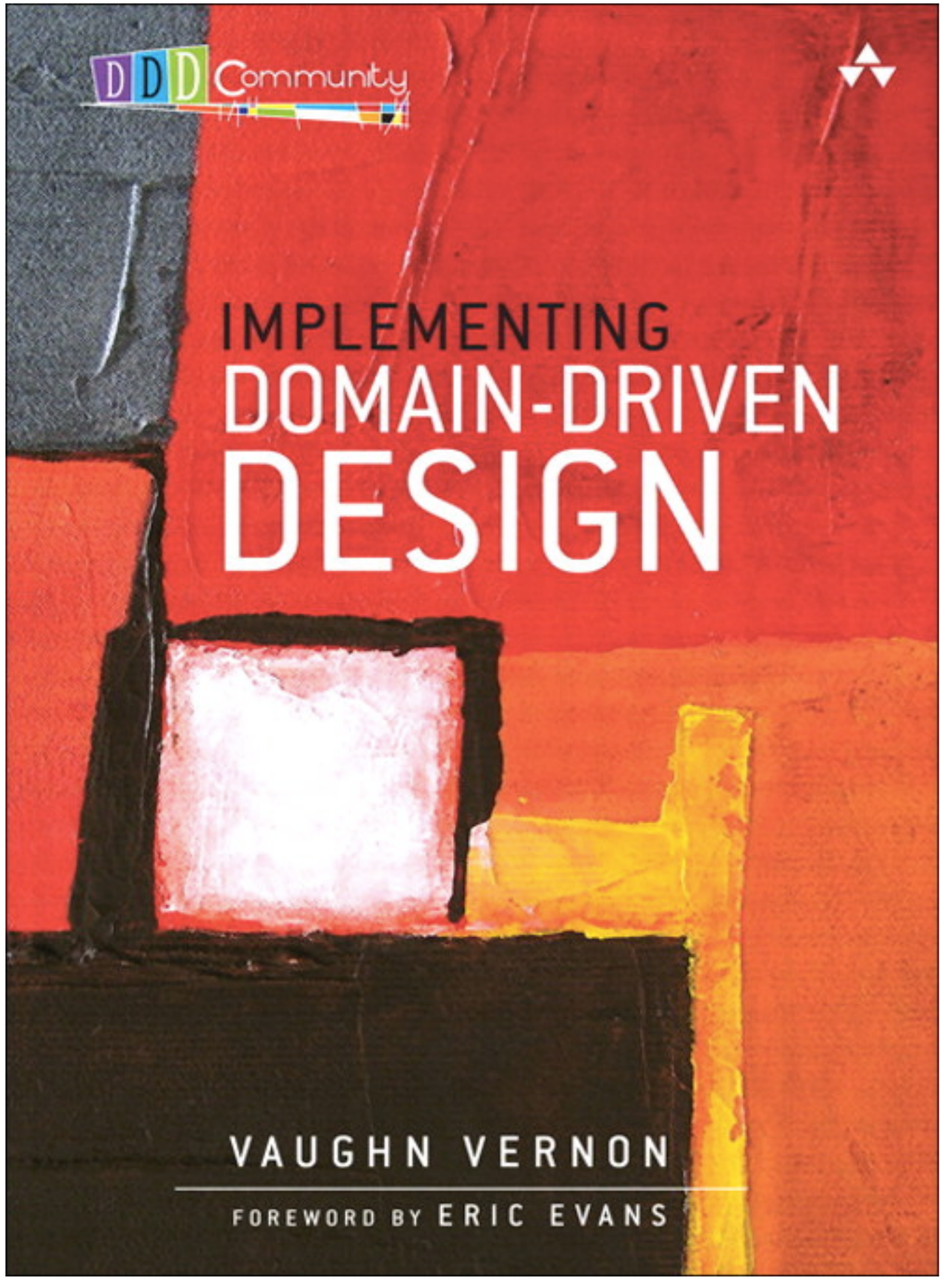
Emission Test

Order parts

Customer Management

Maintenance

Inventory


DDD Community
IMPLEMENTING
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON
FOREWORD BY ERIC EVANS

✓ Modelled Around Business Domain

Culture Of Automation

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

Culture Of Automation

✓ Modelled Around Business Domain

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

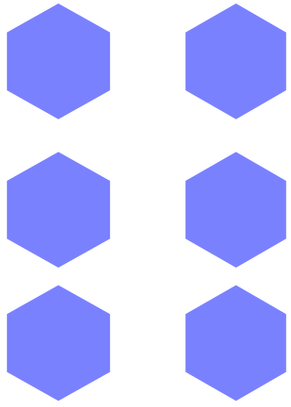
***2 Microservices***

**3 Months**

***10 Microservices***

***2 Microservices***

3 Months

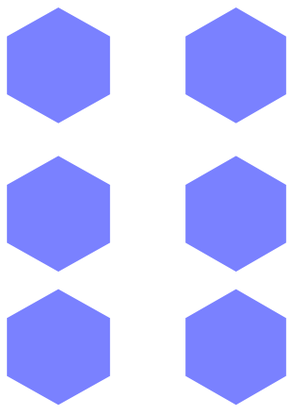


12 Months

***2 Microservices***

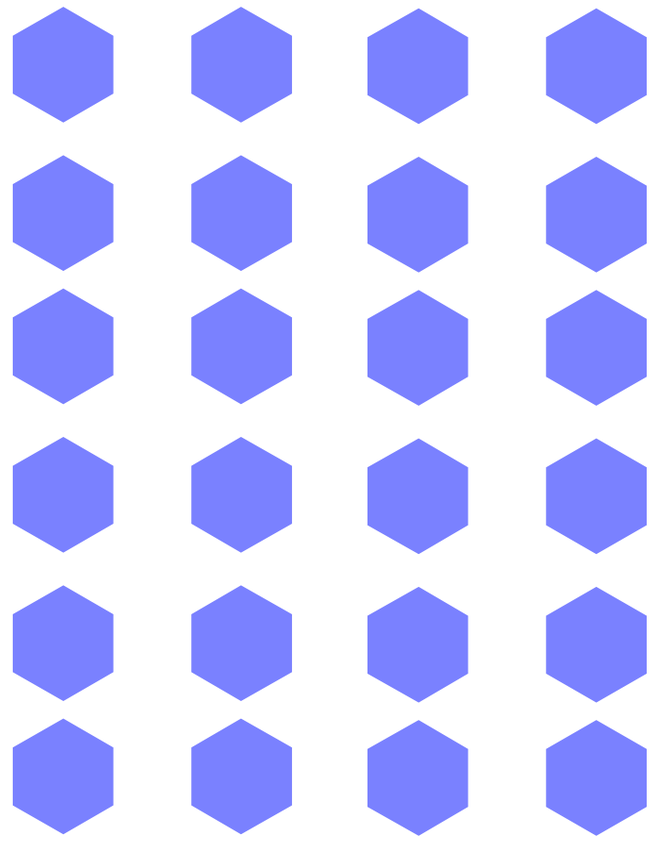
**3 Months**

***10 Microservices***



**12 Months**

***60 Microservices***



**18 Months**

# Infrastructure Automation

@velocityconf

@samnewman

# Infrastructure Automation

## Automated Testing

@velocityconf

@samnewman

Infrastructure Automation

Automated Testing

Continuous Delivery

@velocityconf

@samnewman

✓ Modelled Around Business Domain

✓ Culture Of Automation

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

✓ Modelled Around Business Domain

✓ Culture Of Automation

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

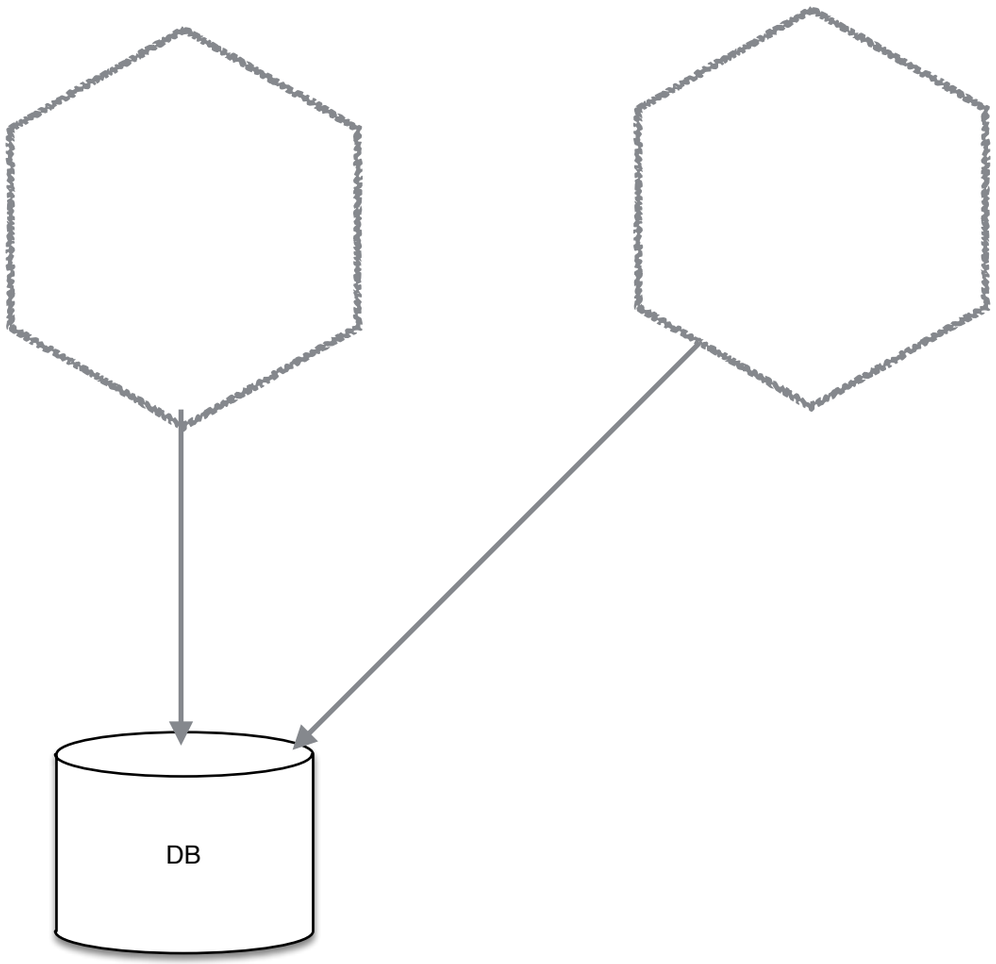
Decentralise All The Things

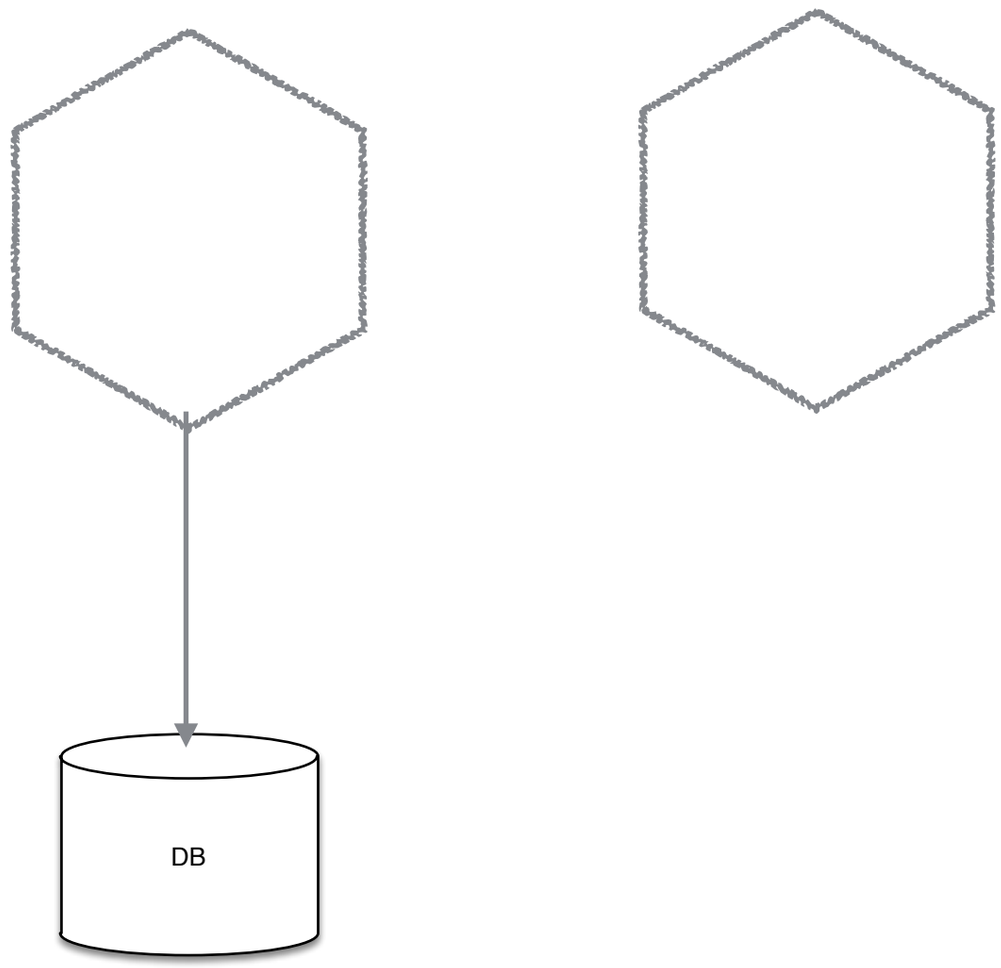
Isolate Failure

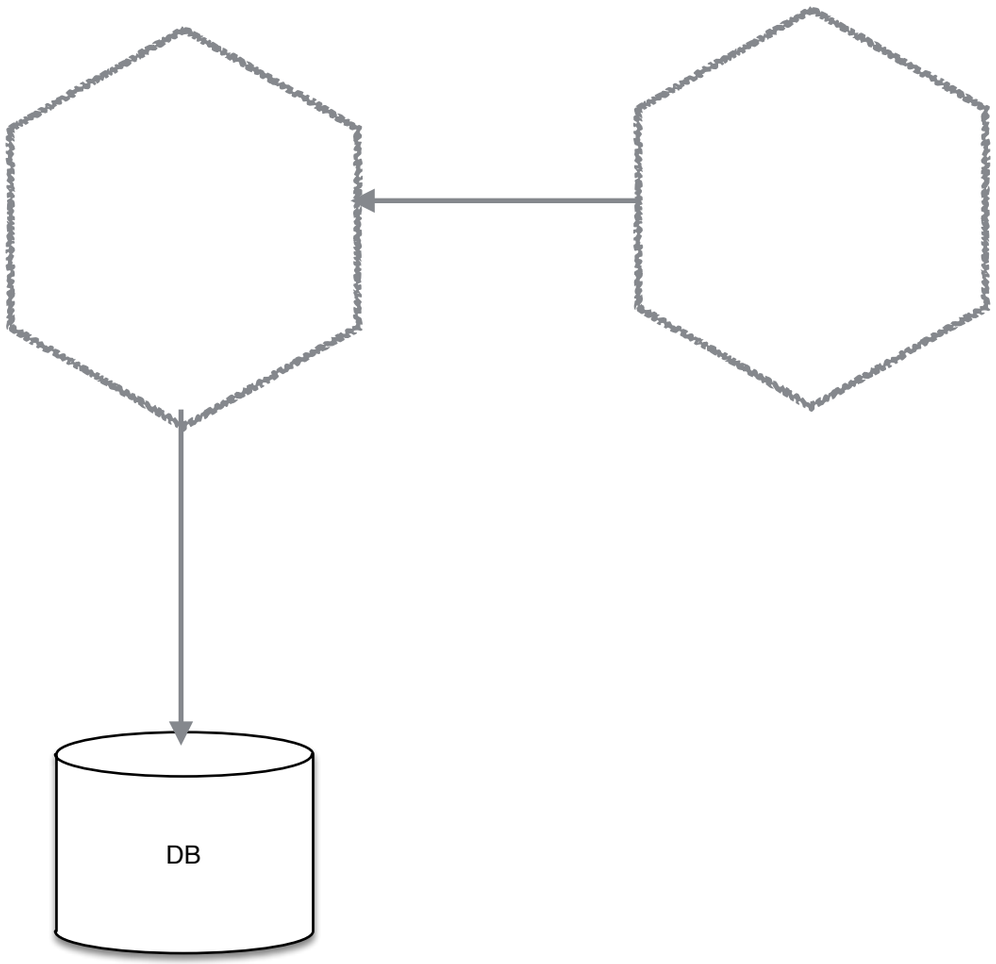
Consumer First

Deploy Independently

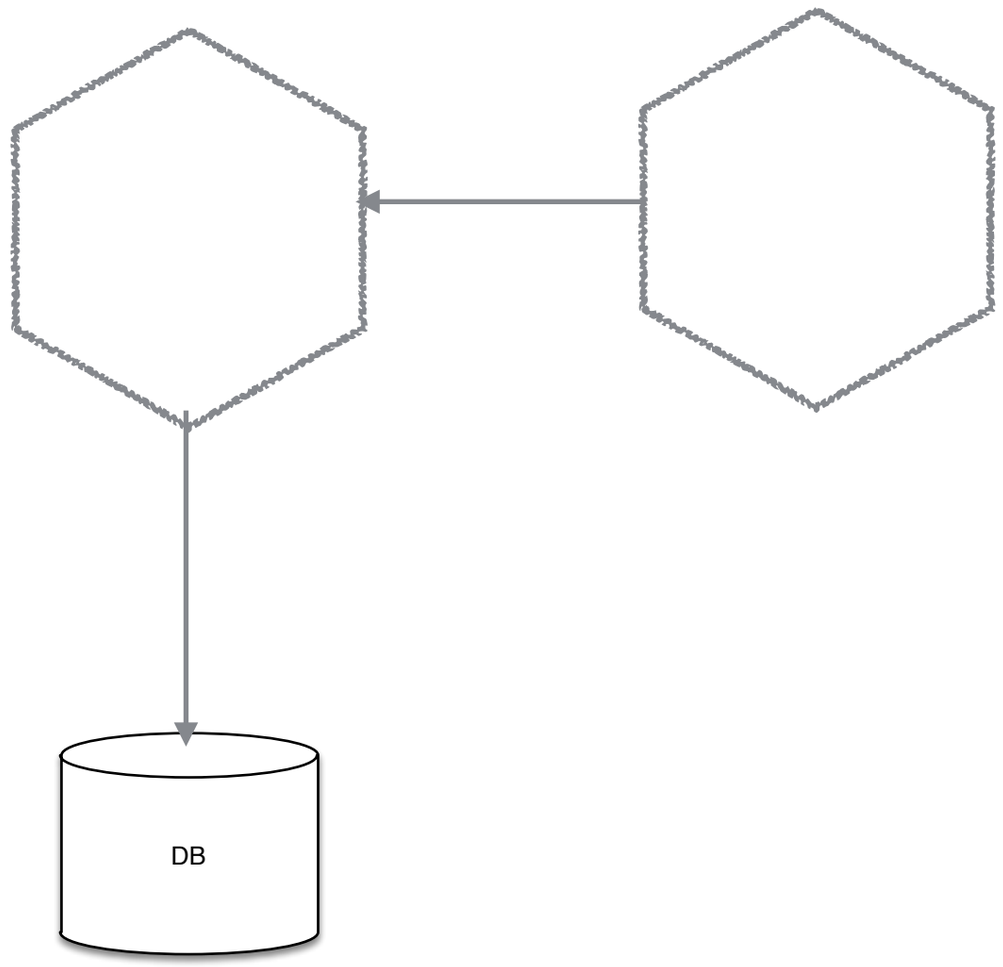

DB

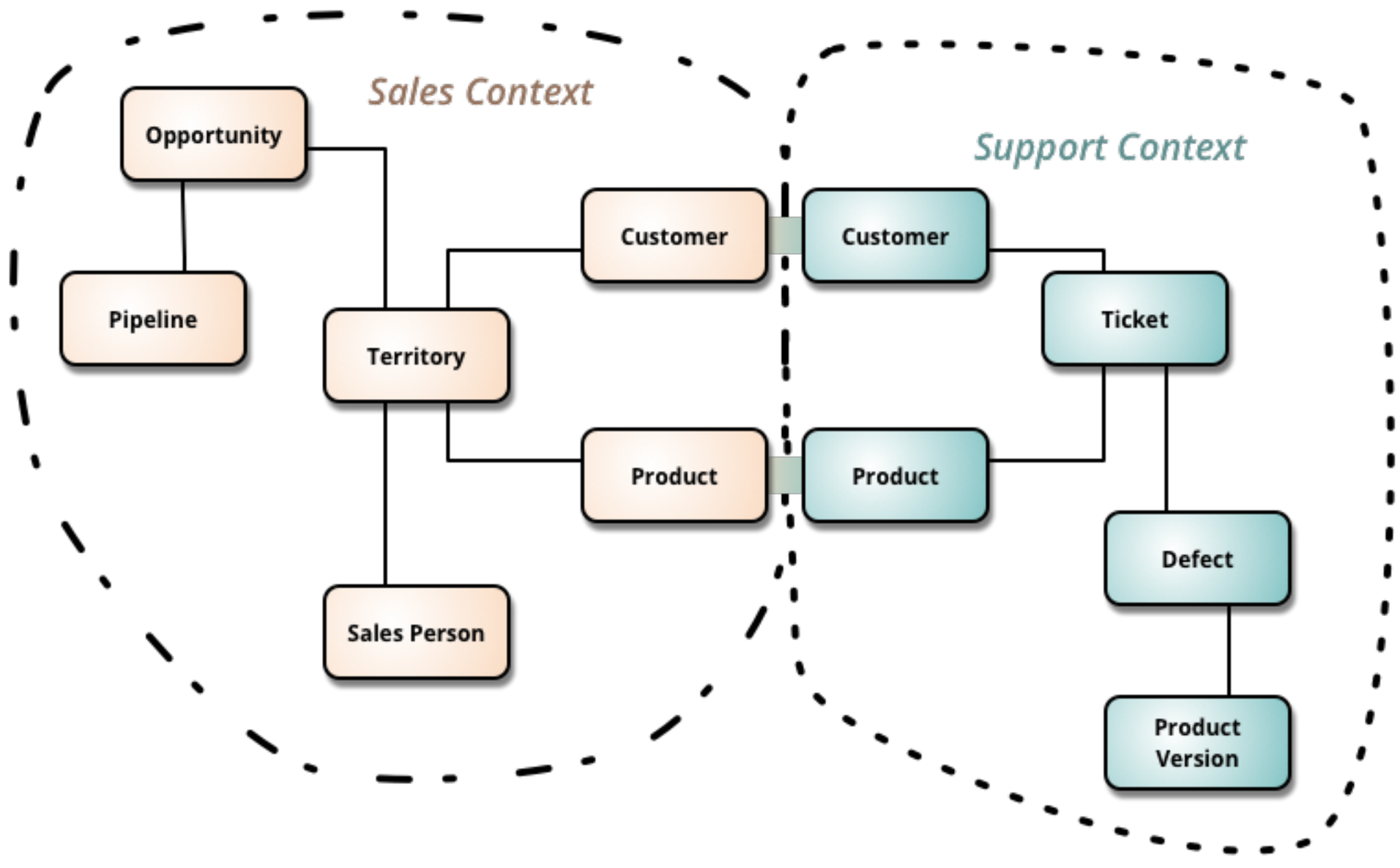

DB


DB


DB

# HIDE YOUR DATABASE

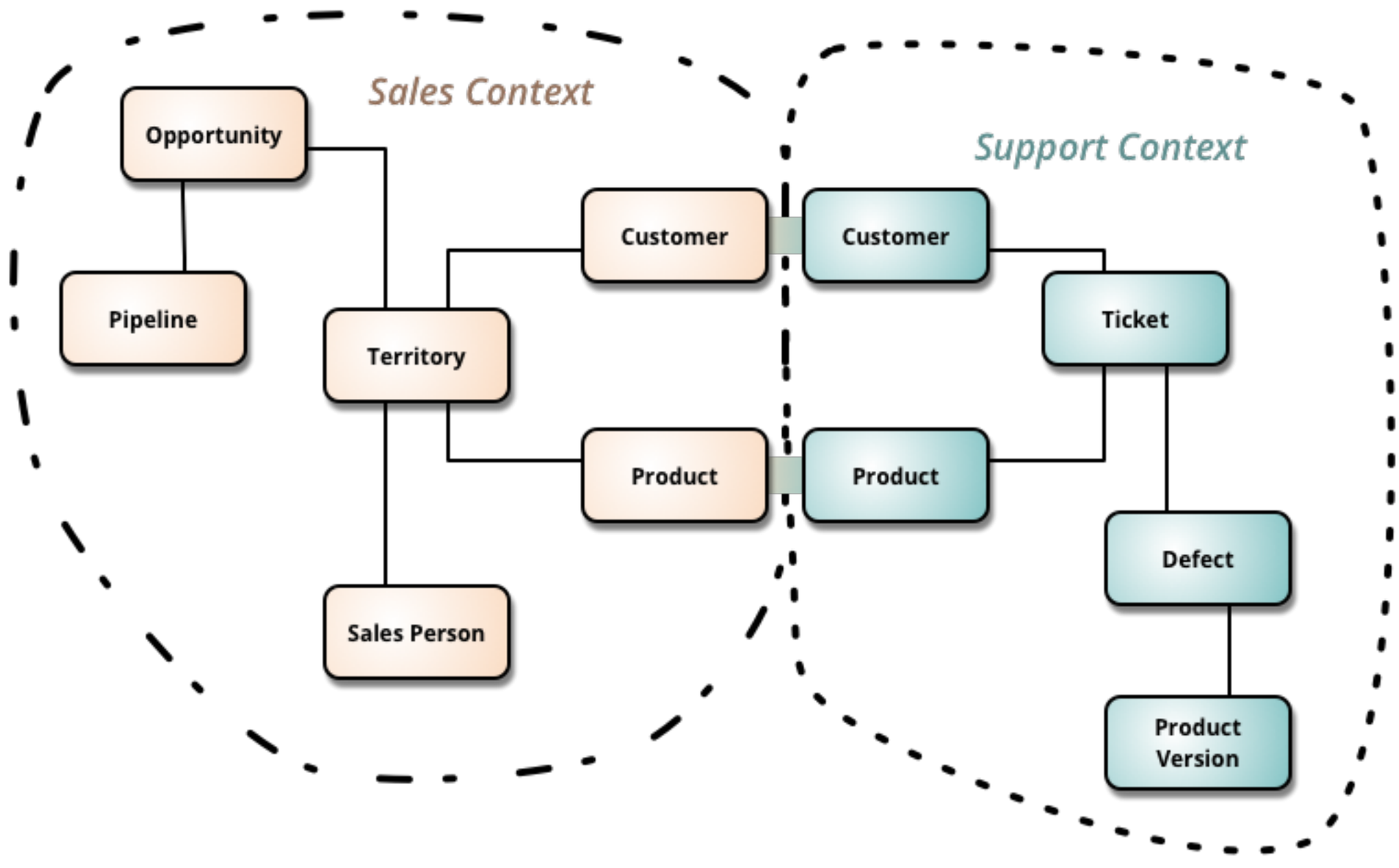





@velocityconf @samnewman

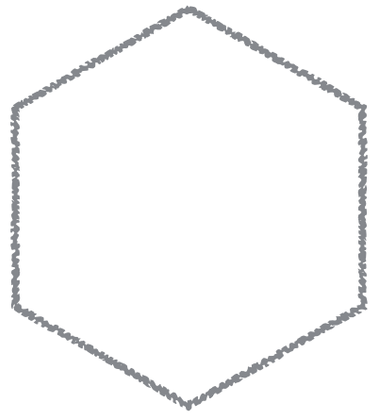
http://martinfowler.com/bliki/images/boundedContext/sketch.png

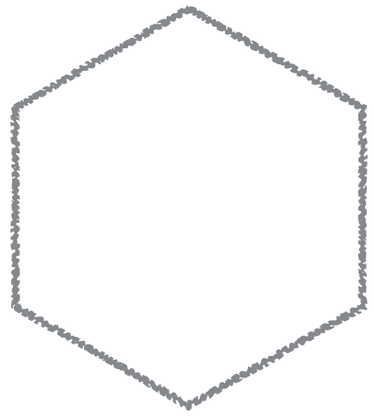
# BOUNDED CONTEXTS

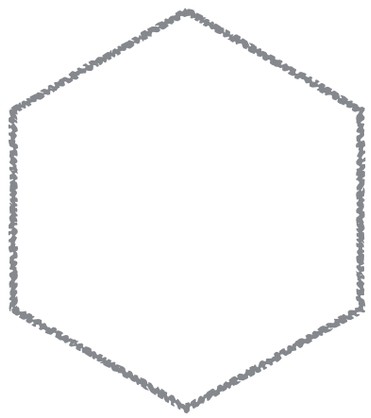


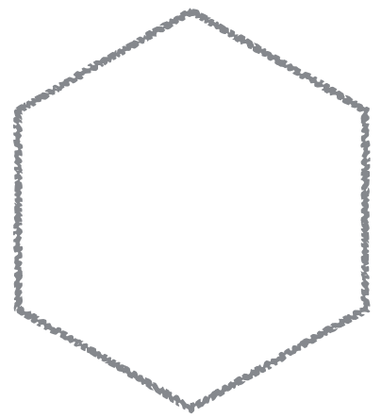


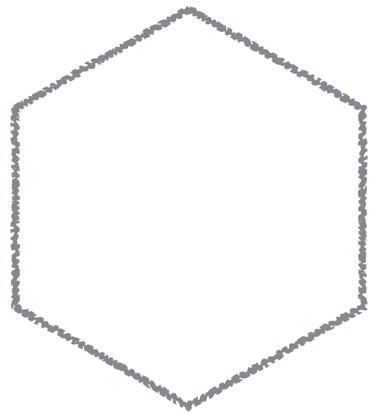

http://martinfowler.com/bliki/images/boundedContext/sketch.png

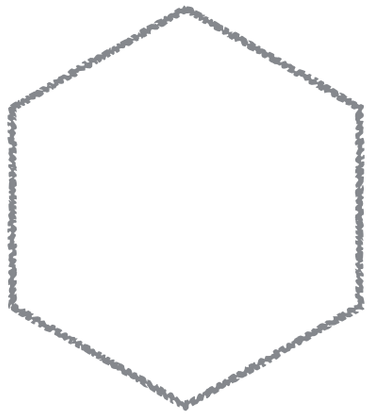




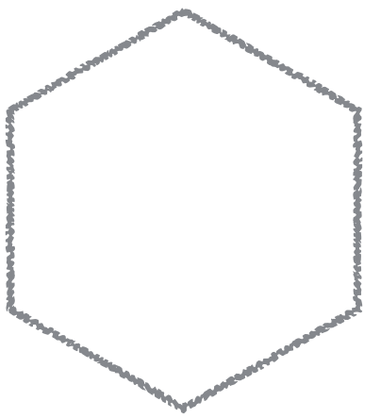


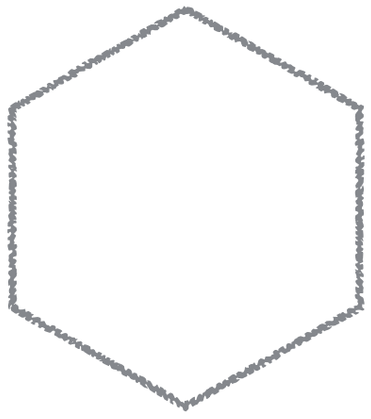


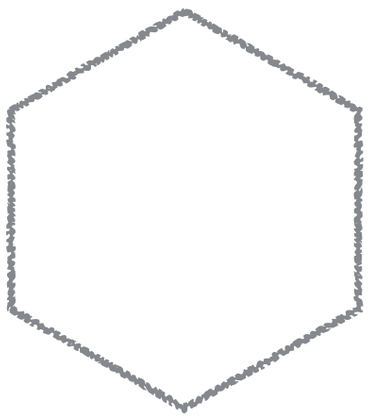


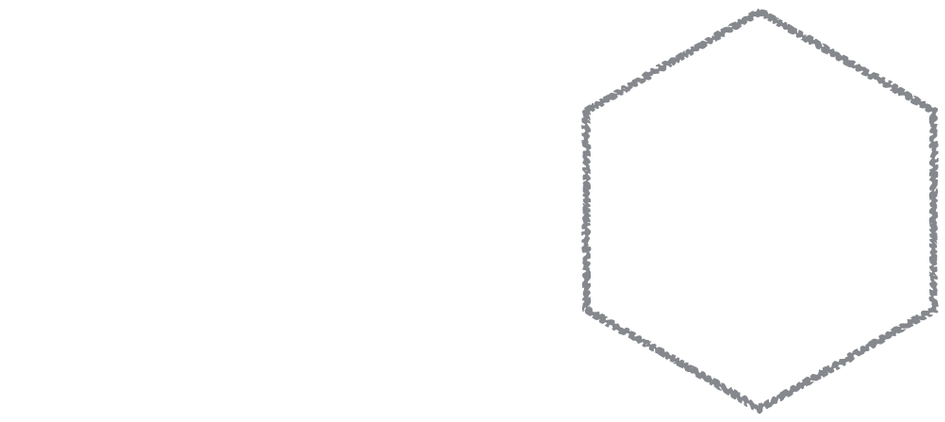


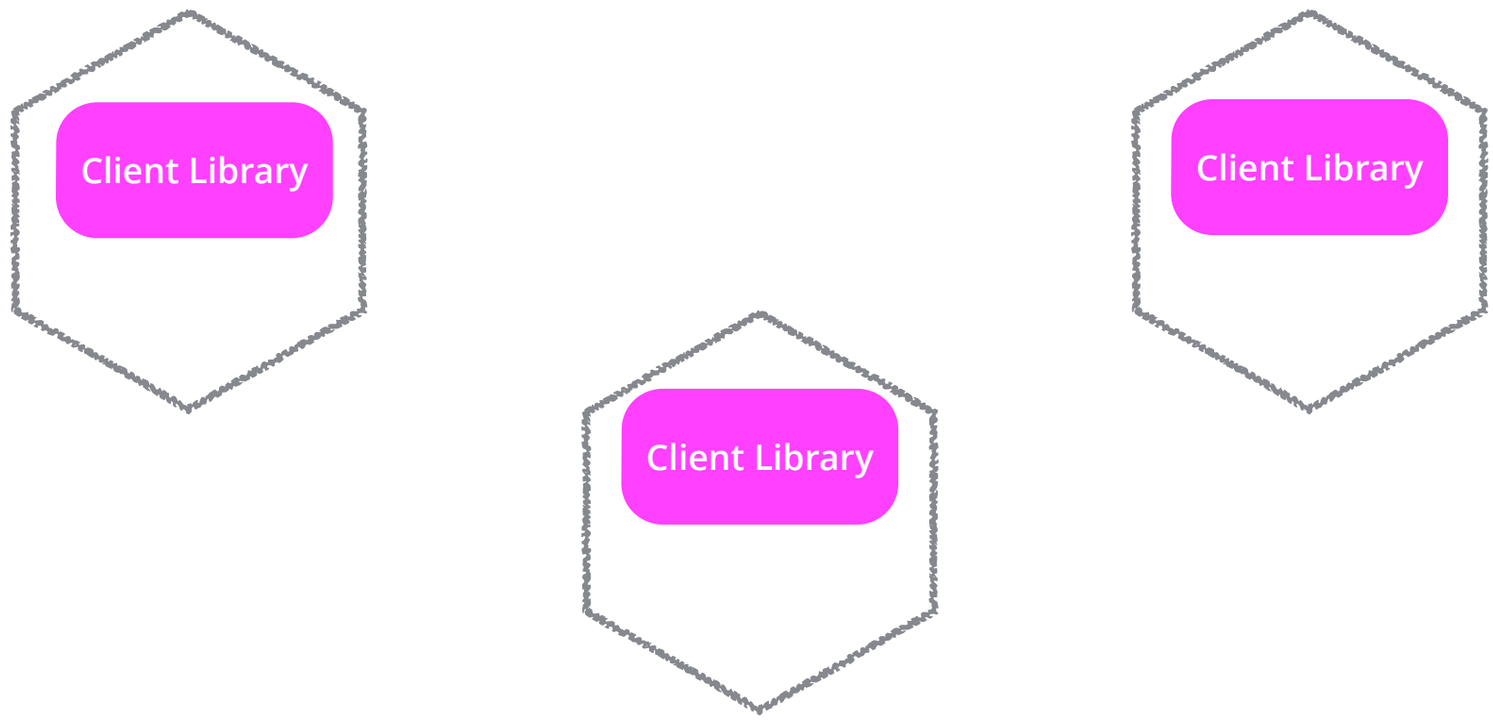
Client Library

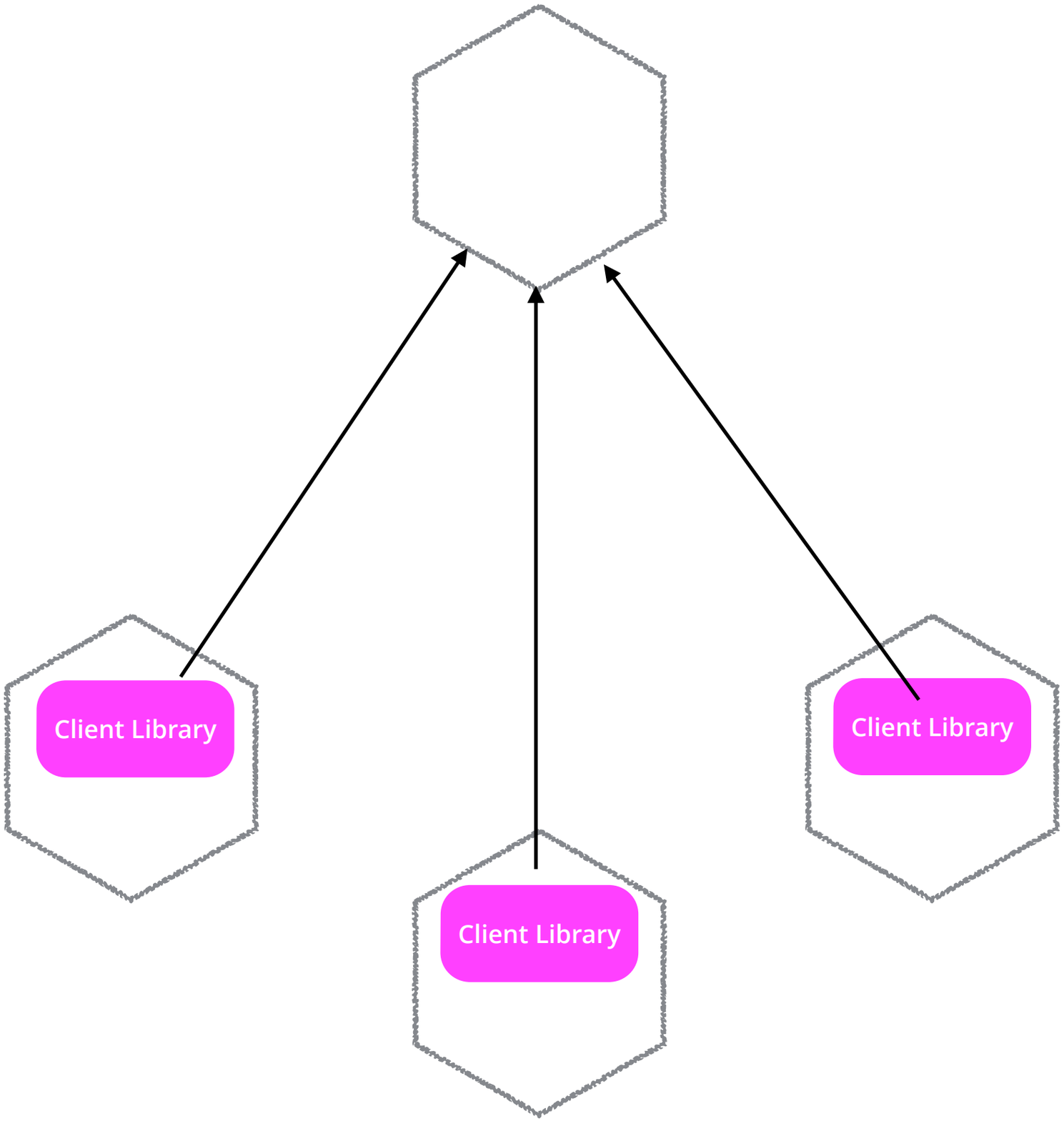




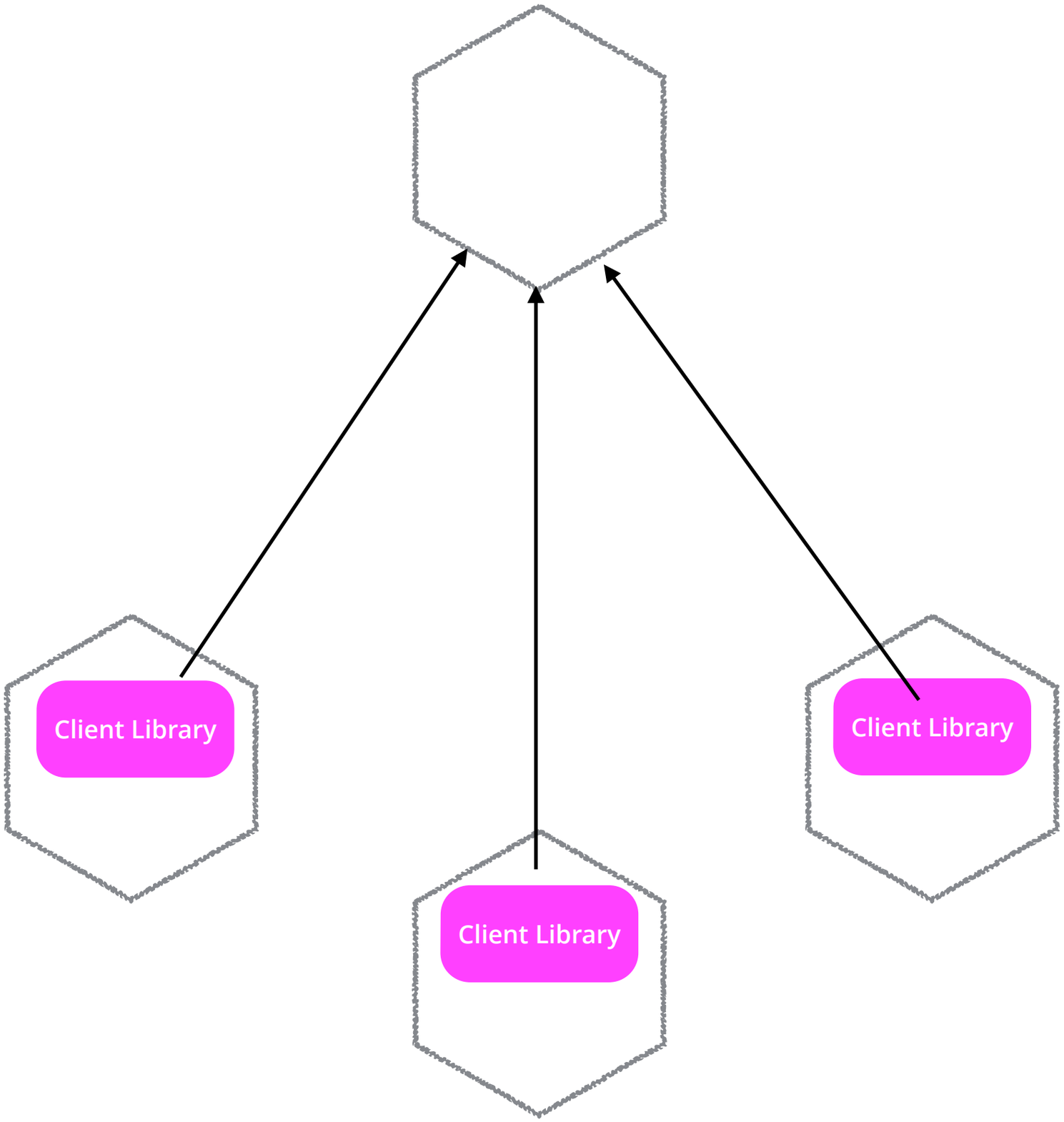





Client Library
Client Library
Client Library


Client Library
Client Library
Client Library

# BE CAREFUL OF CLIENT LIBRARIES



✓ Modelled Around Business Domain

✓ Culture Of Automation

✓ Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently

✓ Modelled Around Business Domain

✓ Culture Of Automation

✓ Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

Consumer First

Deploy Independently


DECENTRALIZE
ALL THE THINGS
memegenerator.net

# What is autonomy?

# What is autonomy?

Giving people as much freedom as possible to do the job at hand

# What is autonomy?

Giving people as much freedom as possible to do the job at hand


DEVOLUTION?

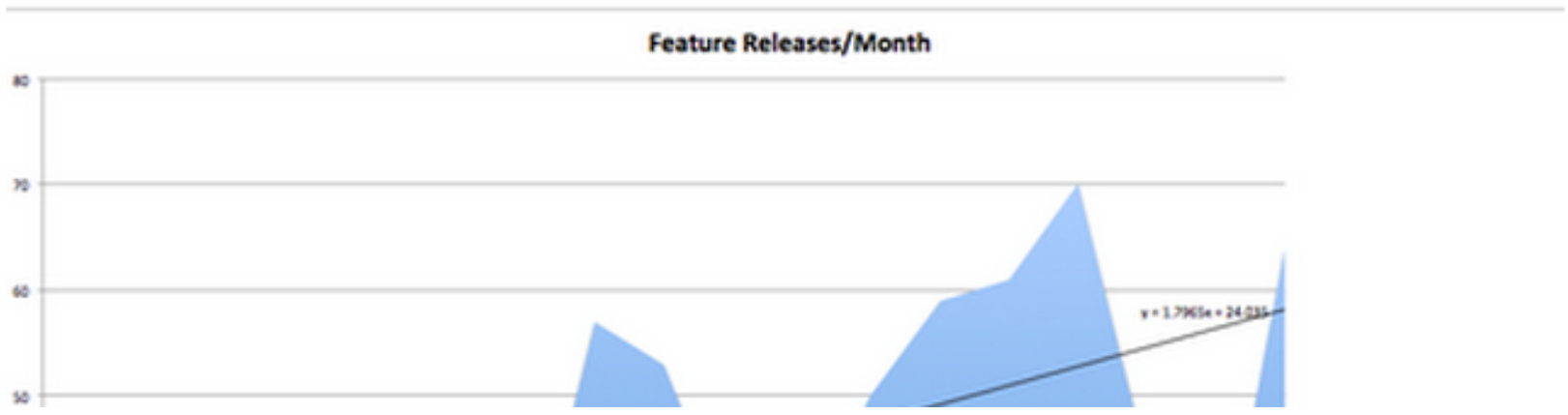
# SELF-SERVICE

https://www.flickr.com/photos/katsrcool/15184711908/

GILT TECH

Search the blog

# Making Architecture Work in Microservice Organizations

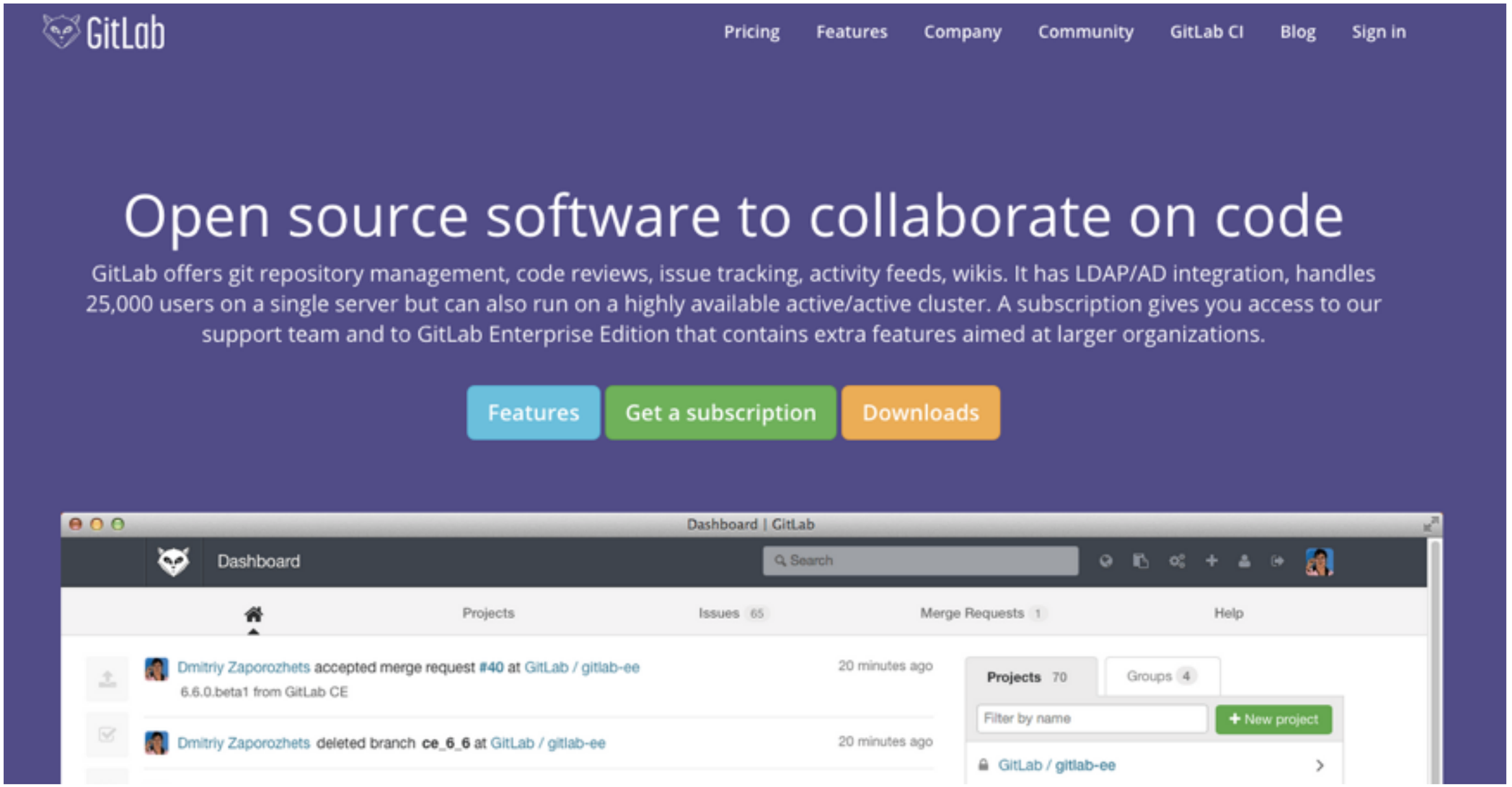



http://tech.gilt.com/post/102628539834/making-architecture-work-in-microservice

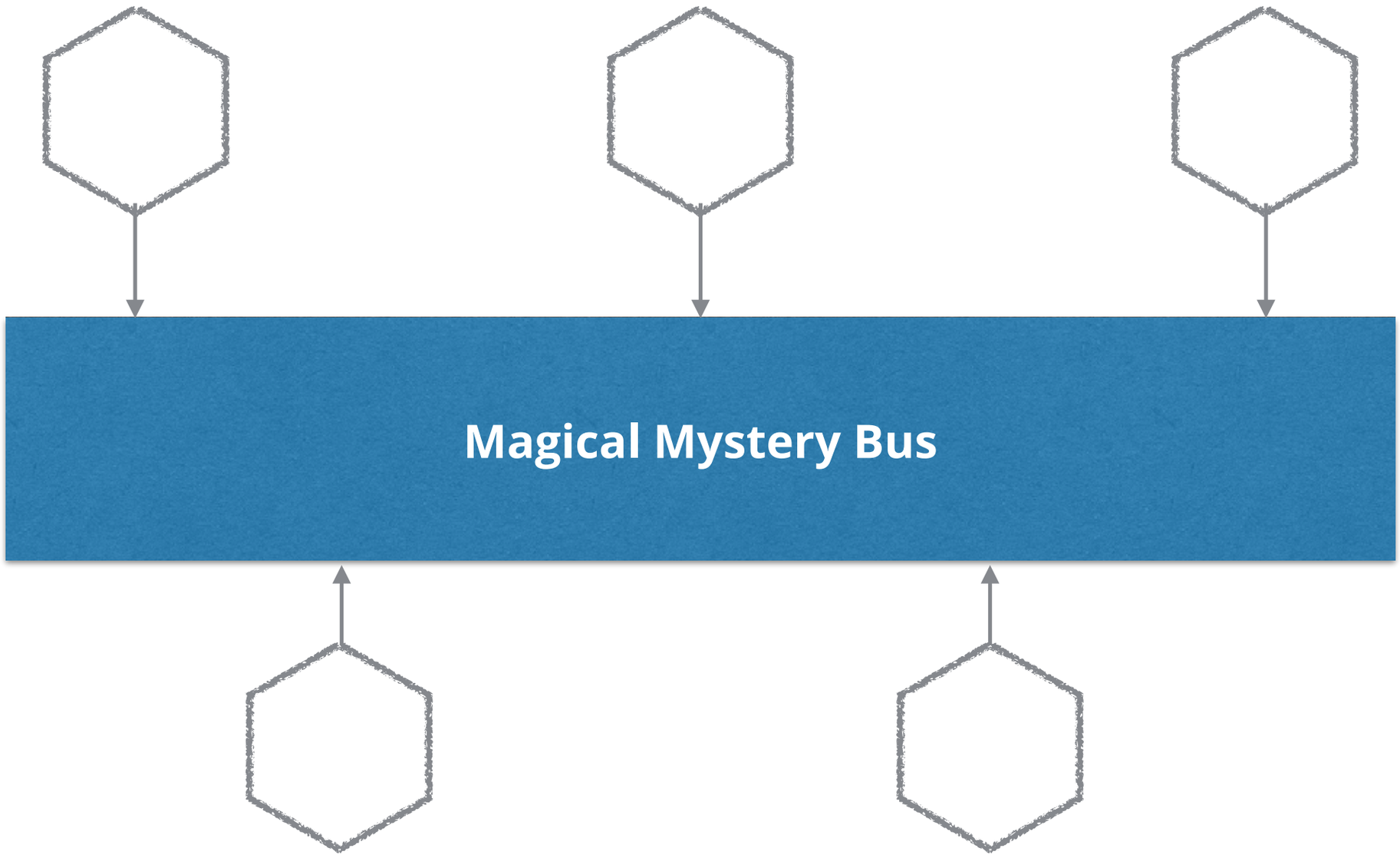
# OWNER-OPERATOR


https://www.flickr.com/photos/stevendepolo/5939055612

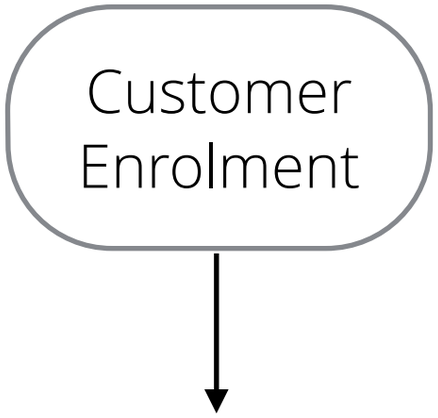
# INTERNAL OPEN SOURCE

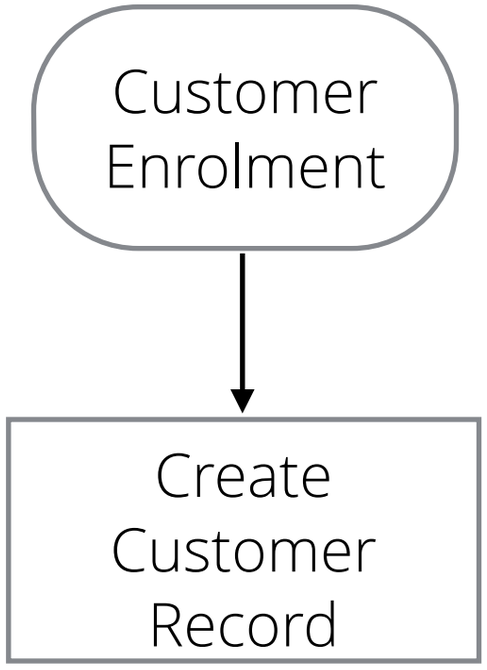


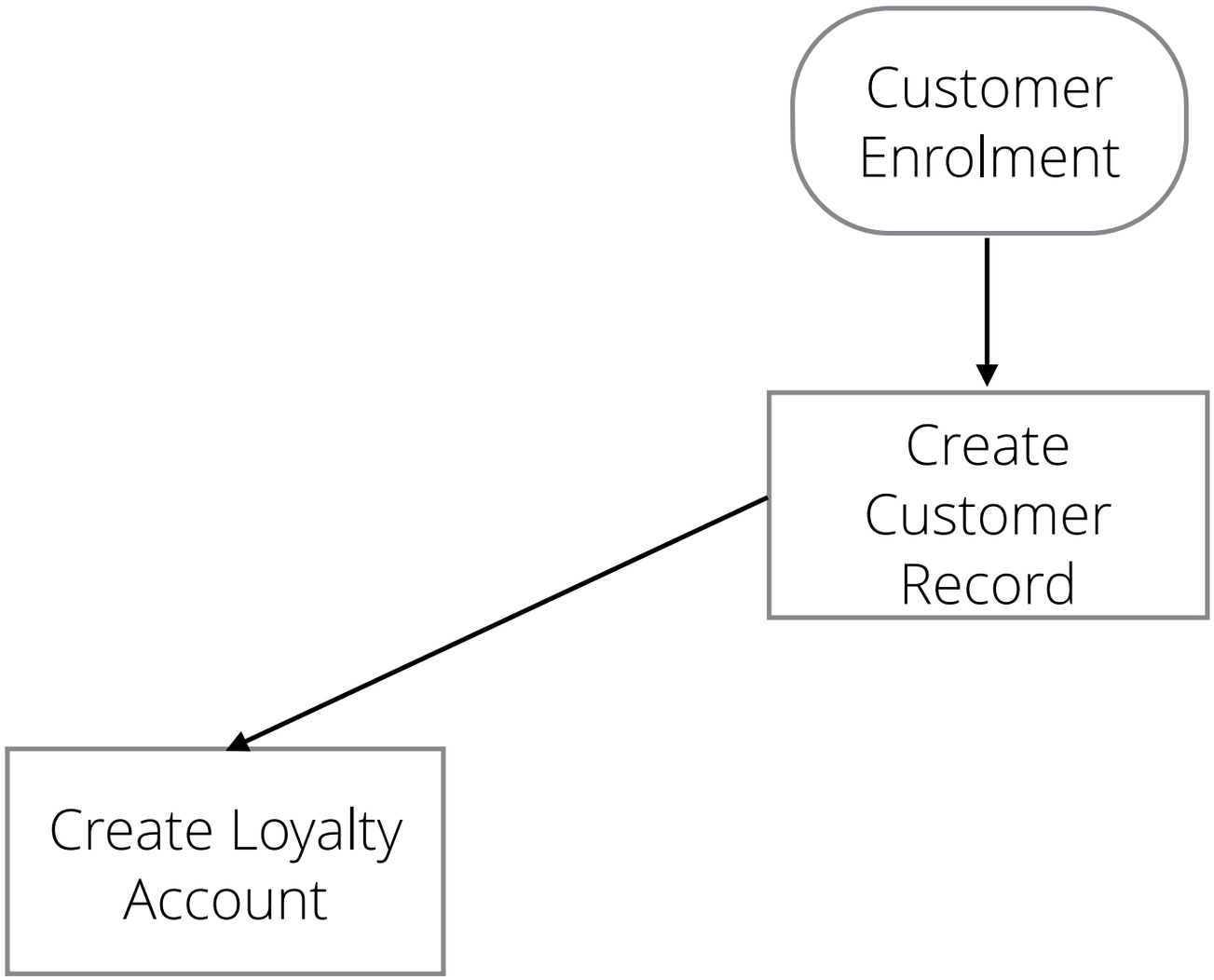
# DUMB-PIPES, SMART ENDPOINTS

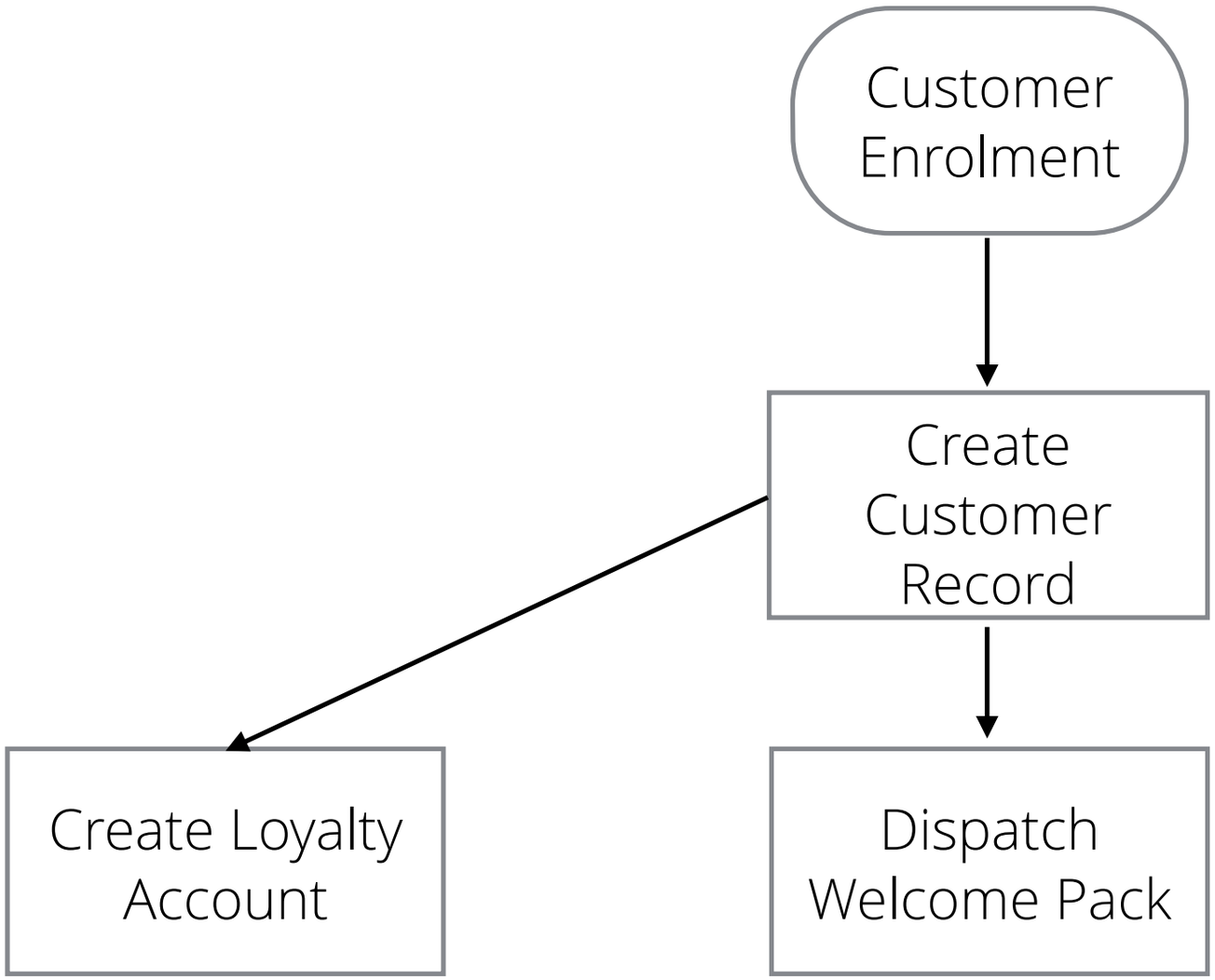


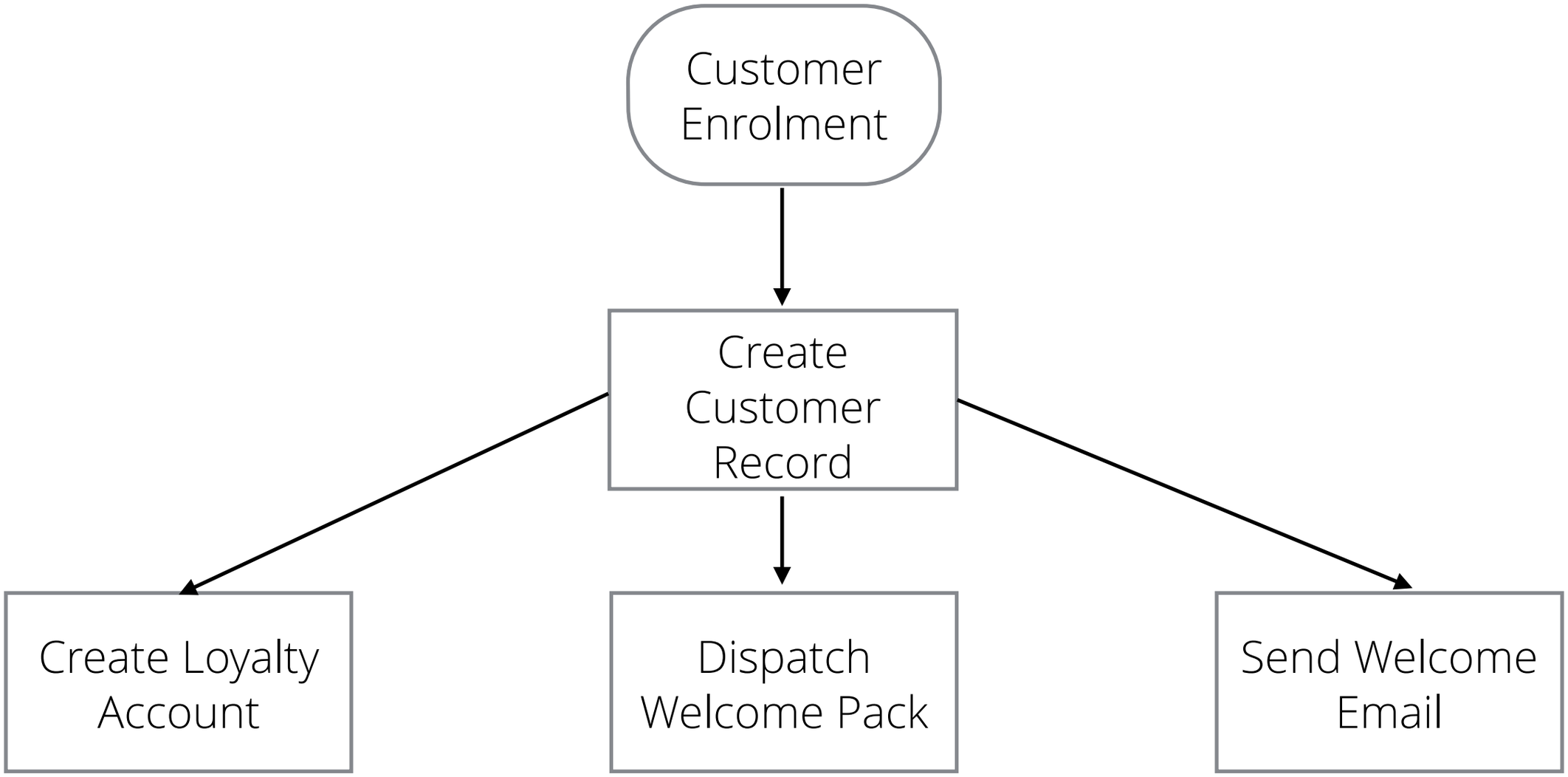
# Magical Mystery Bus

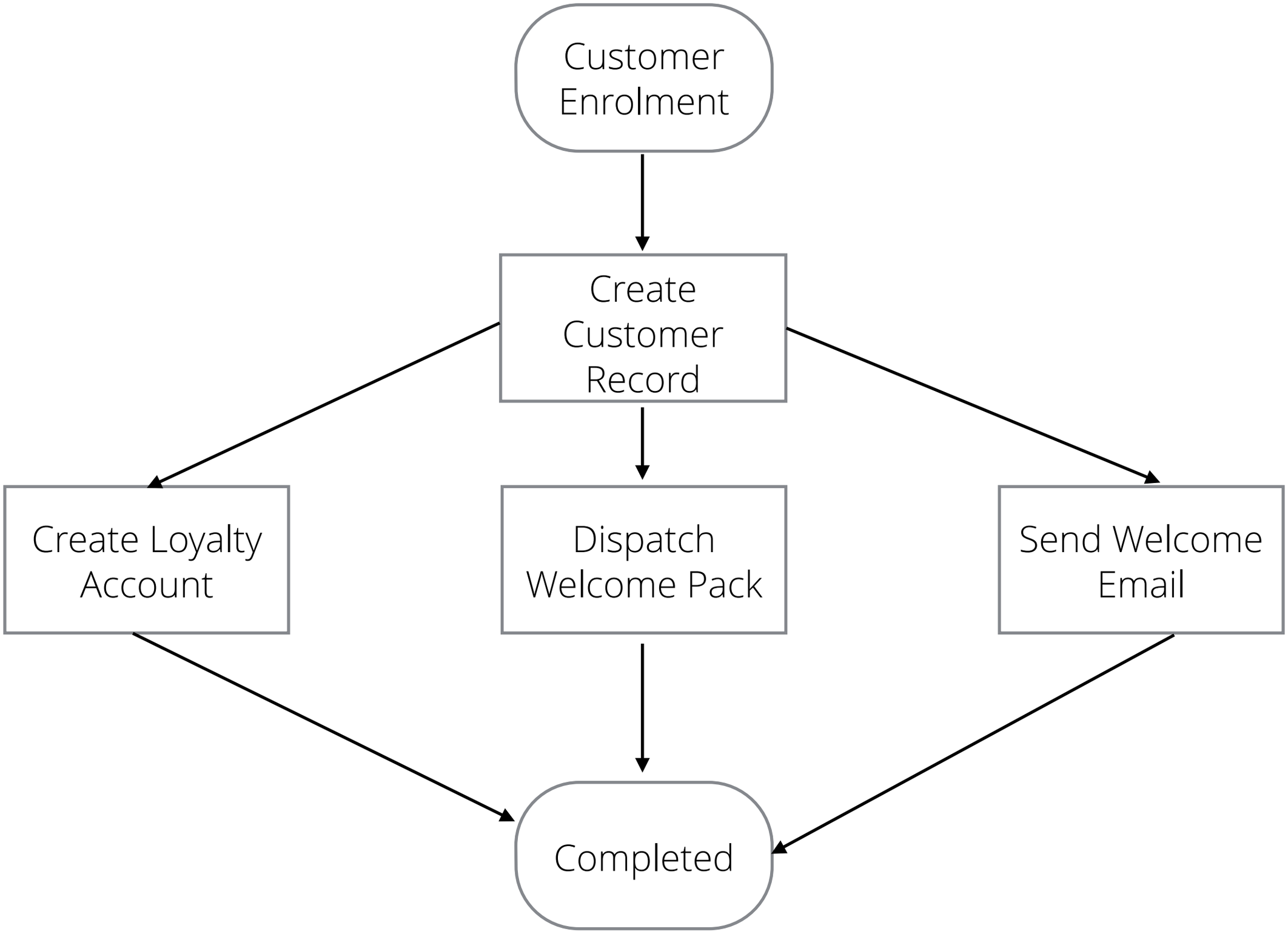

Customer
Enrolment


Customer Enrolment
Create Customer Record


Customer Enrolment
Create Customer Record
Create Loyalty Account

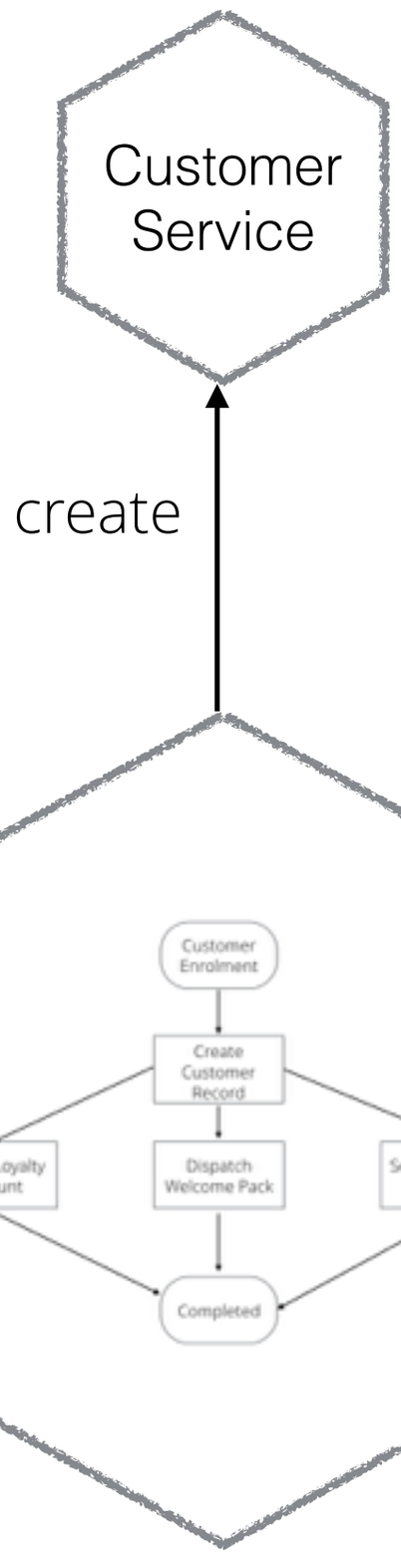

Customer Enrolment
Create Customer Record
Create Loyalty Account
Dispatch Welcome Pack


Customer Enrolment
Create Customer Record
Create Loyalty Account
Dispatch Welcome Pack
Send Welcome Email

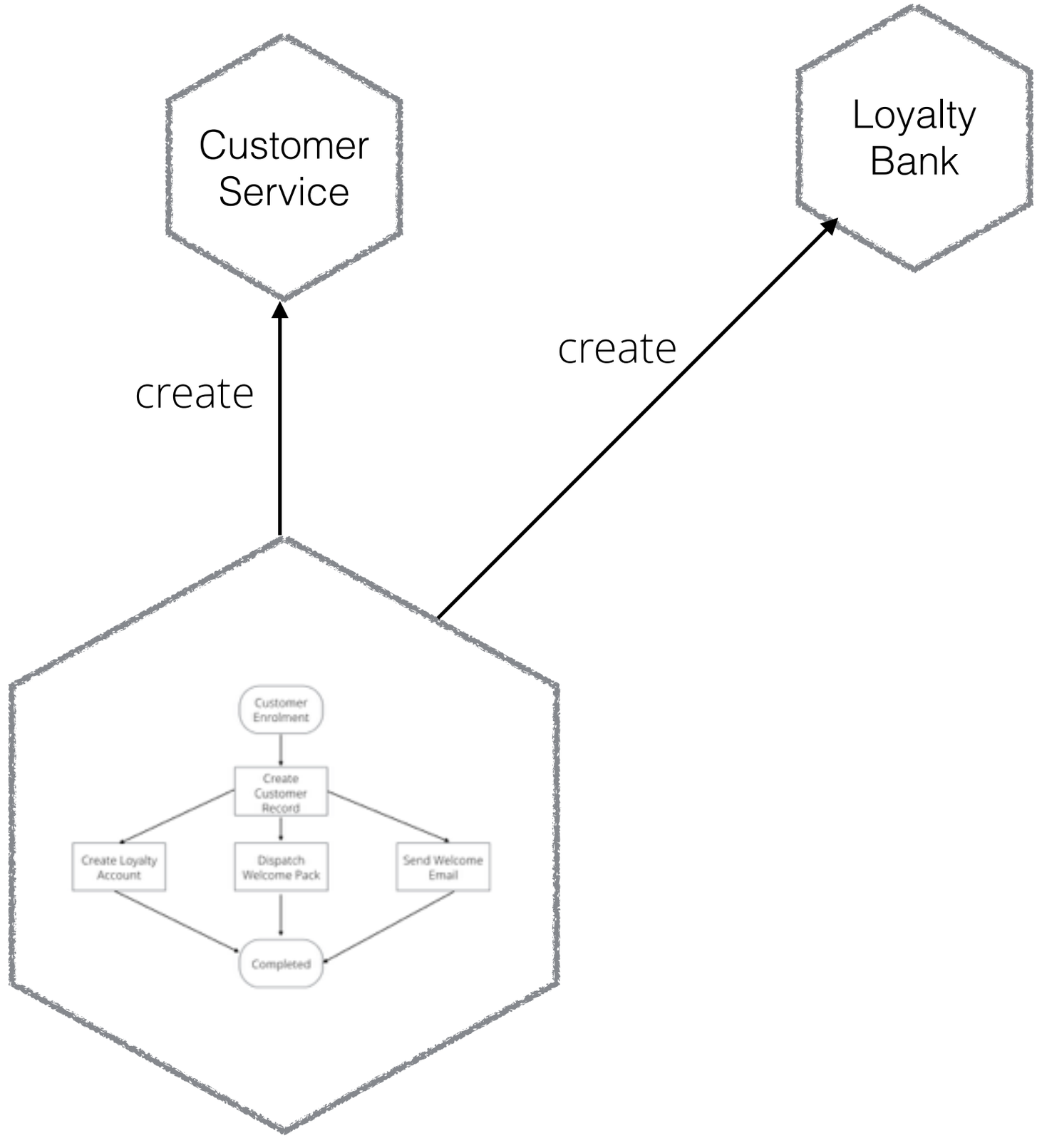

Customer Enrolment
Create Customer Record
Create Loyalty Account
Dispatch Welcome Pack
Send Welcome Email
Completed

# ORCHESTRATION

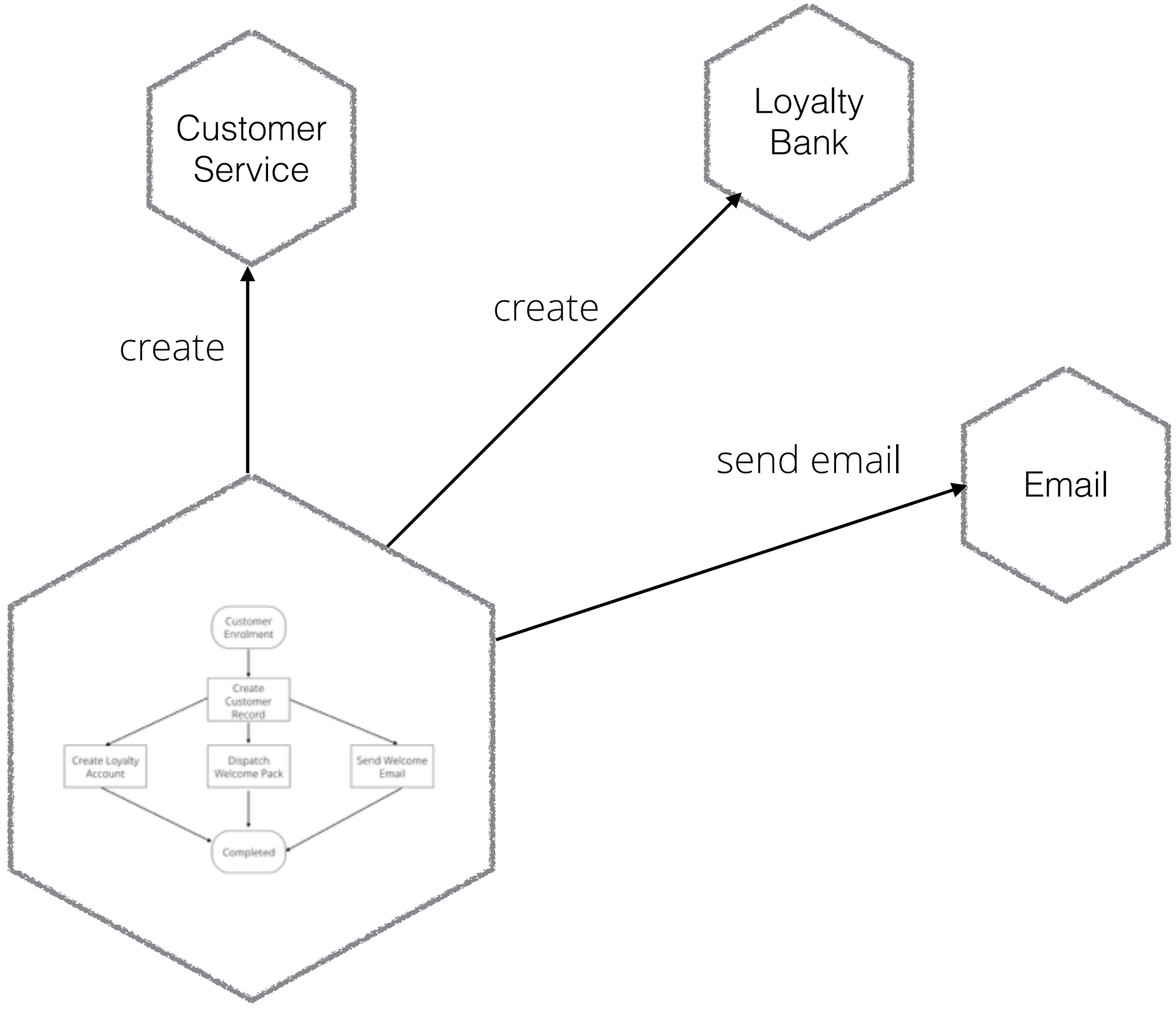
# ORCHESTRATION



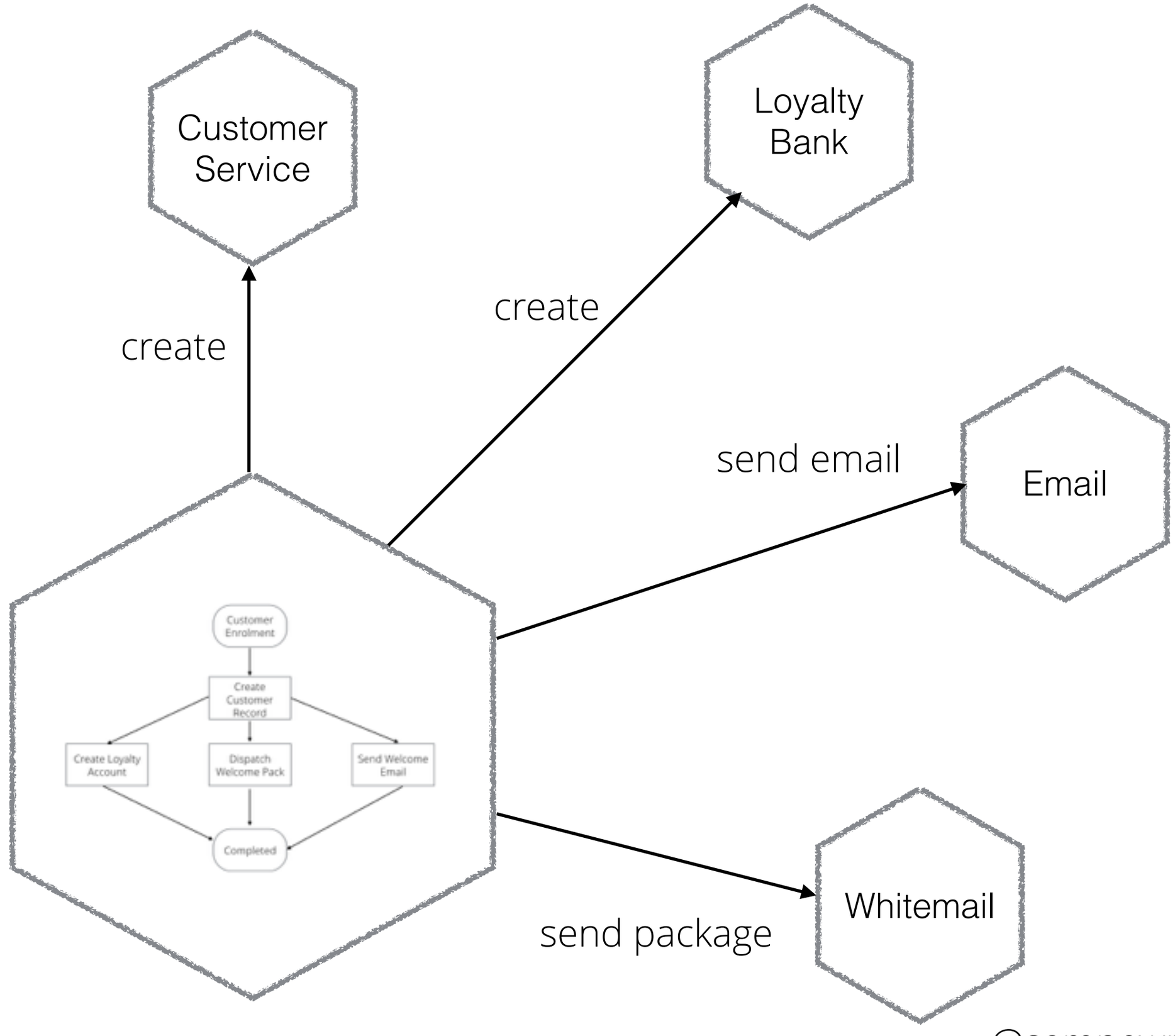
# ORCHESTRATION



# ORCHESTRATION

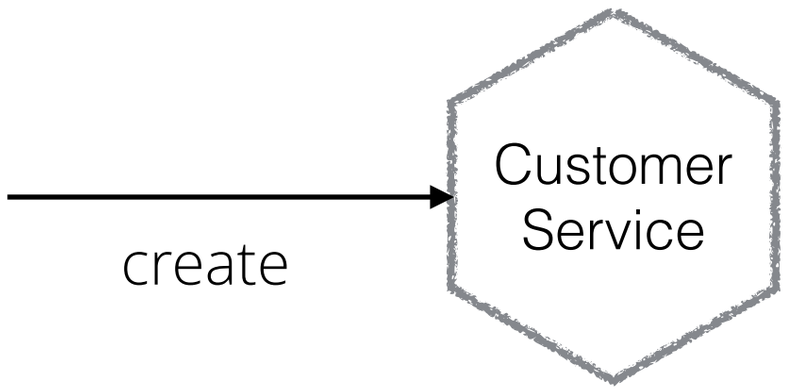


# ORCHESTRATION

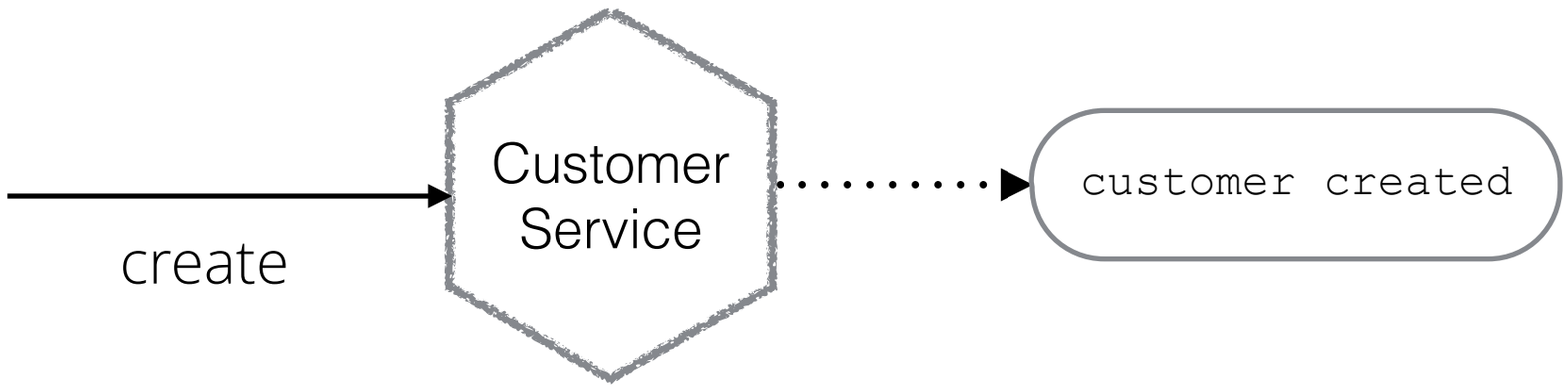


# CHOREOGRAPHED

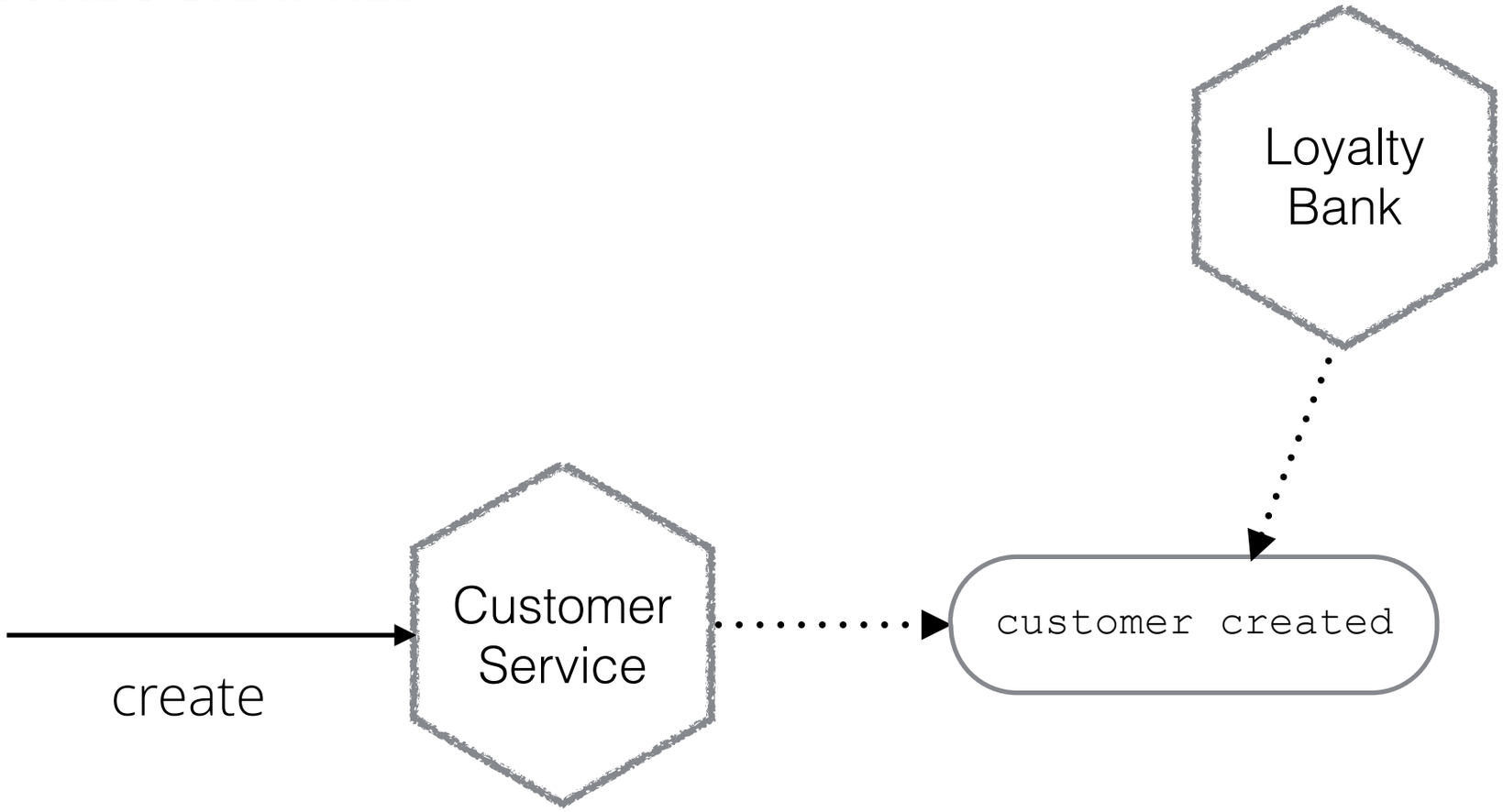
# CHOREOGRAPHED



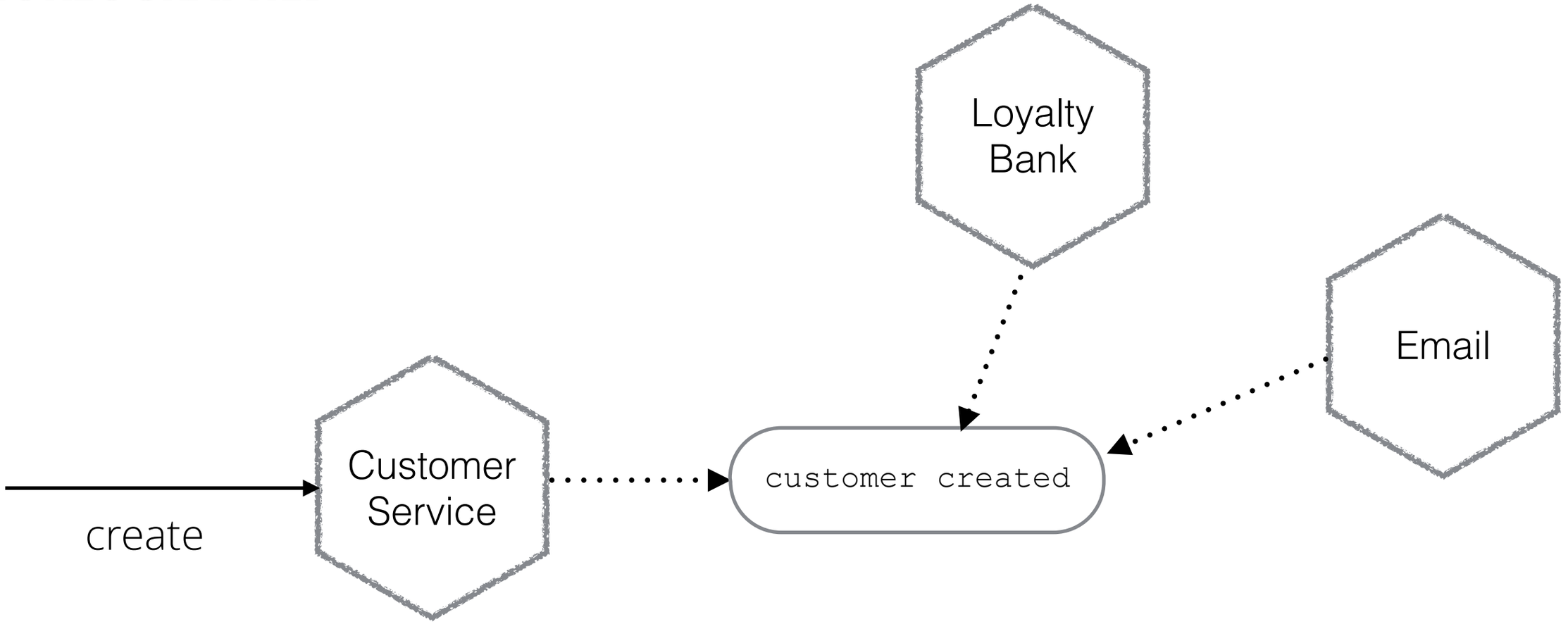
# CHOREOGRAPHED



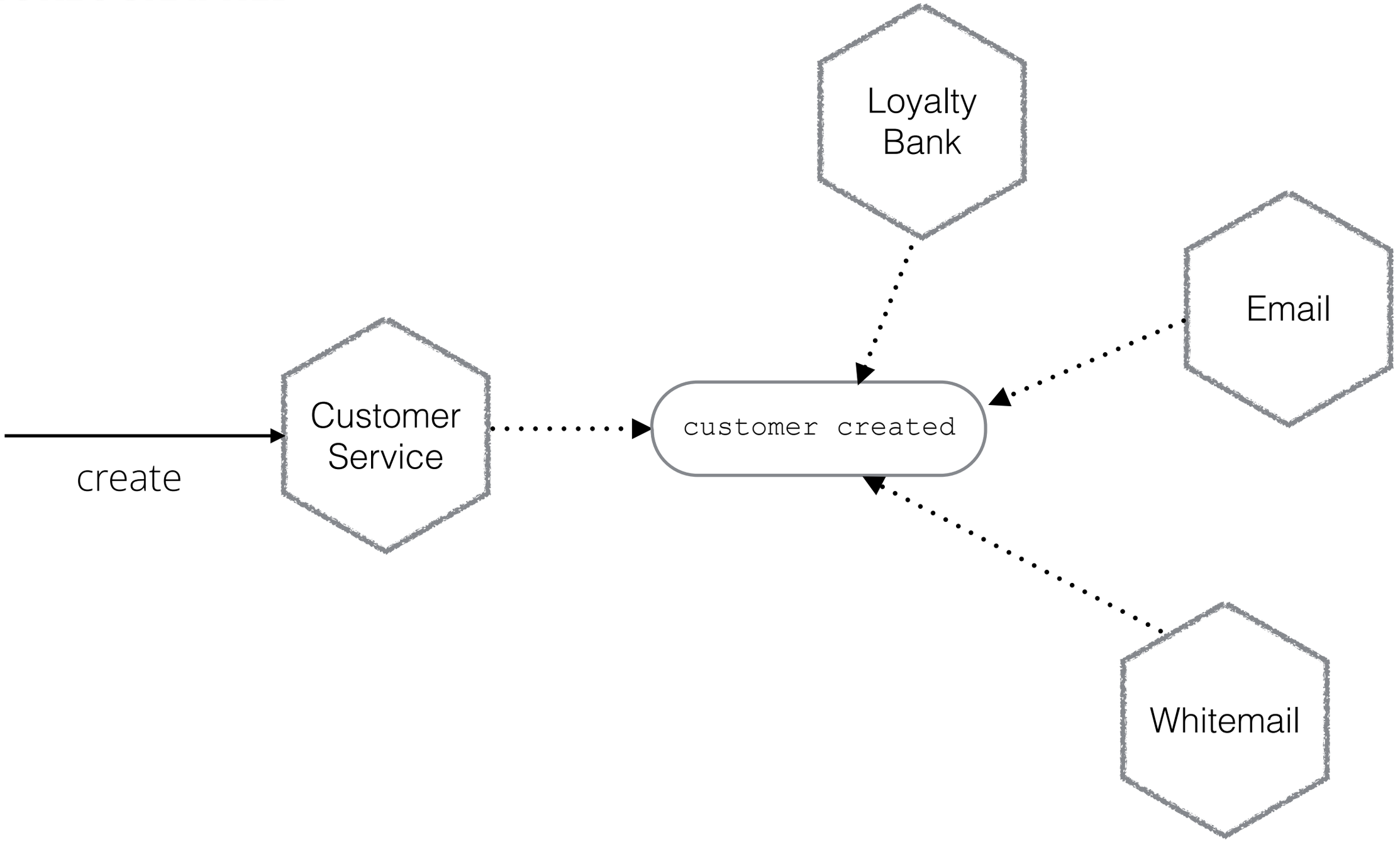
# CHOREOGRAPHED



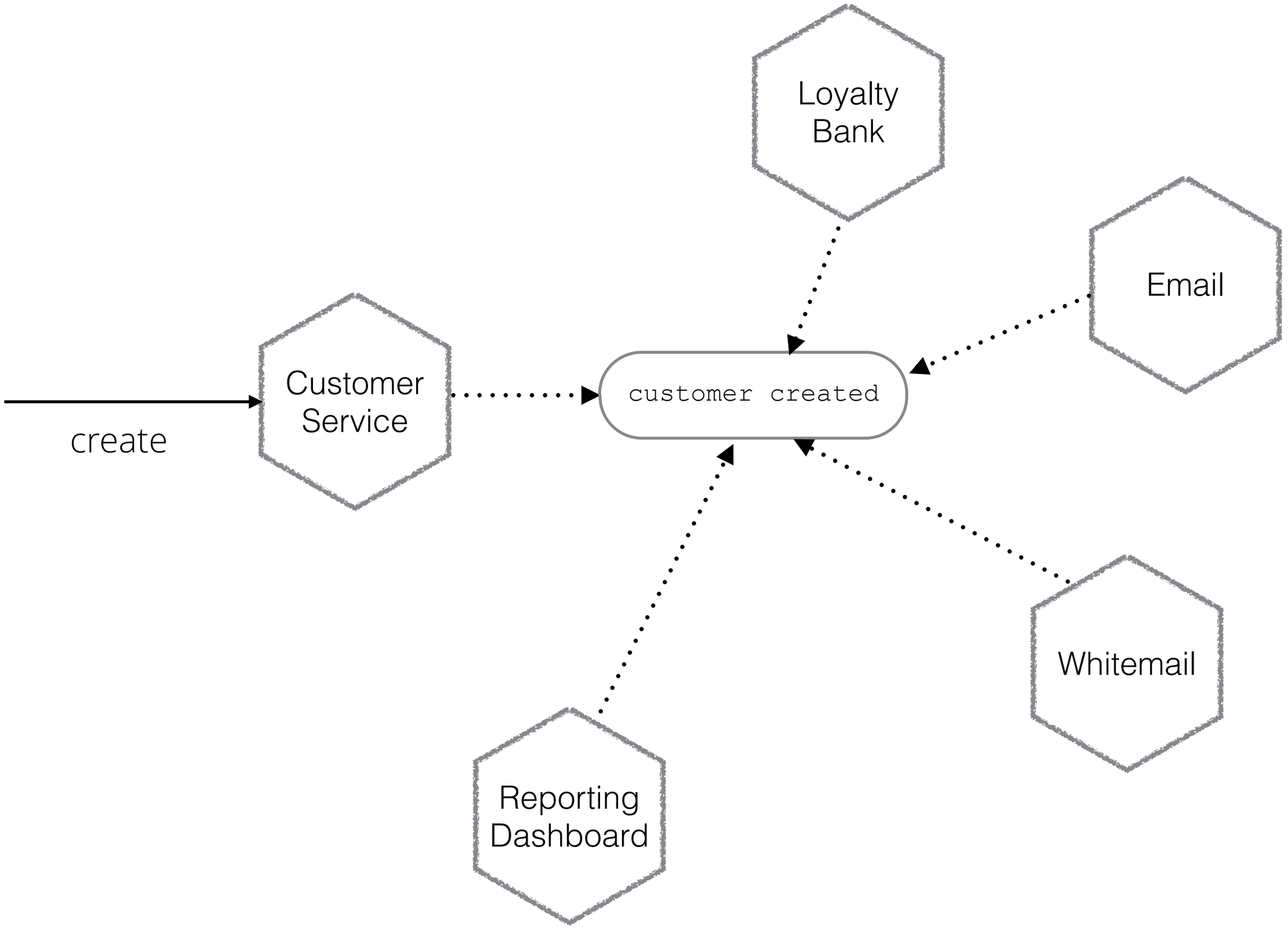
# CHOREOGRAPHED



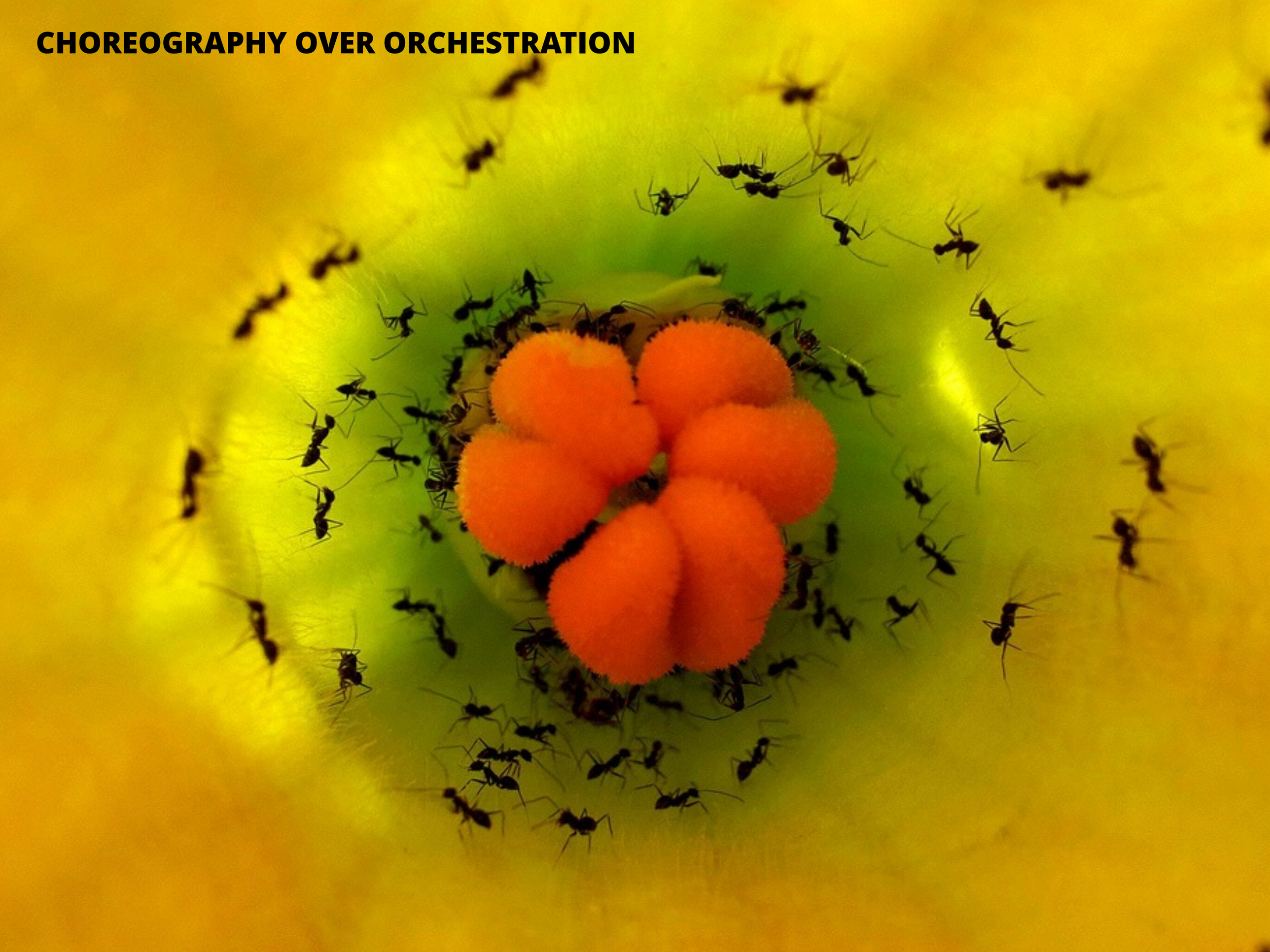
# CHOREOGRAPHED

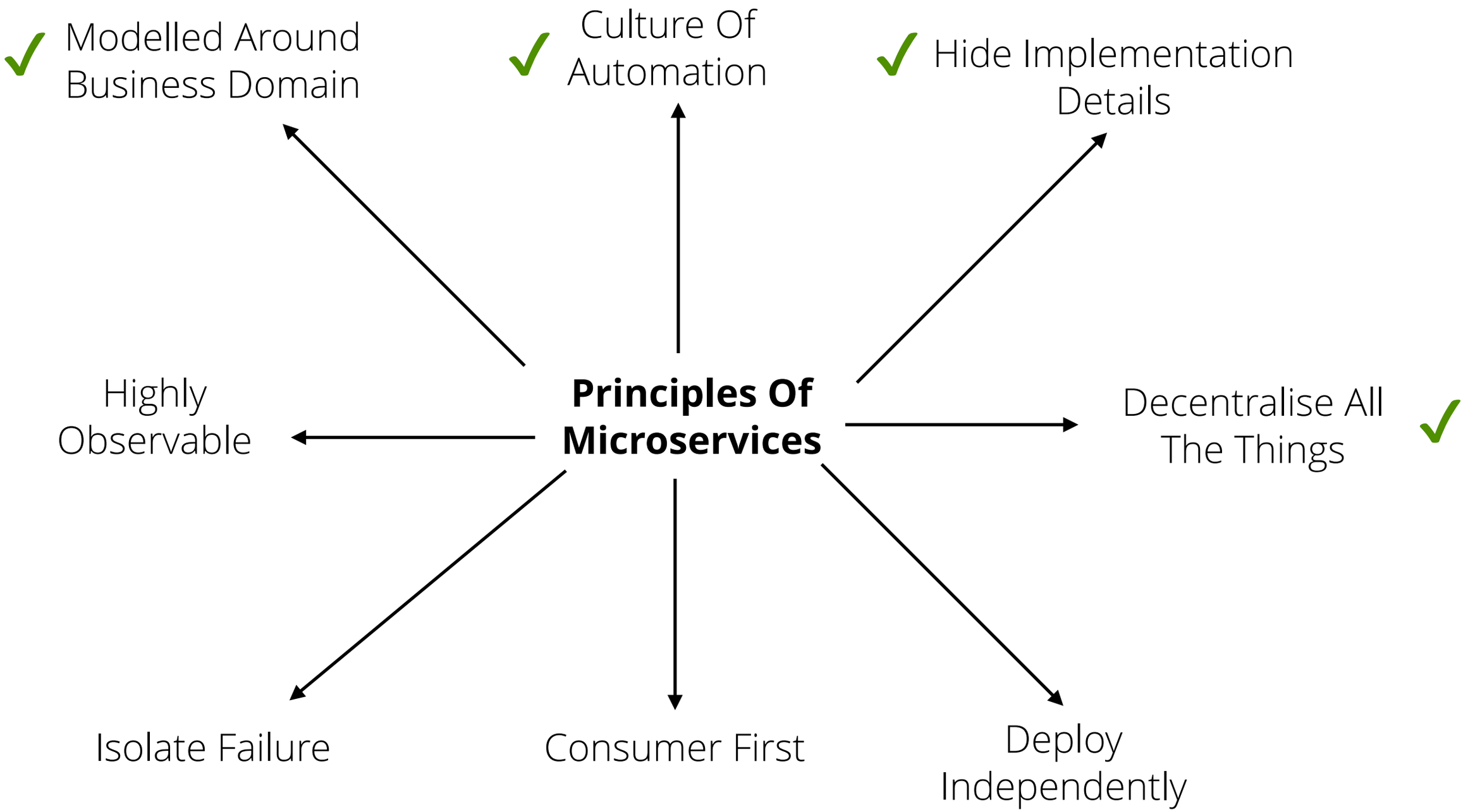


# CHOREOGRAPHED



# CHOREOGRAPHY OVER ORCHESTRATION




✓ Modelled Around Business Domain
✓ Culture Of Automation
✓ Hide Implementation Details
Highly Observable
Principles Of Microservices
Decentralise All The Things ✓
Isolate Failure
Consumer First
Deploy Independently

✓ Modelled Around Business Domain

✓ Culture Of Automation

✓ Hide Implementation Details

Highly Observable

**Principles Of Microservices**

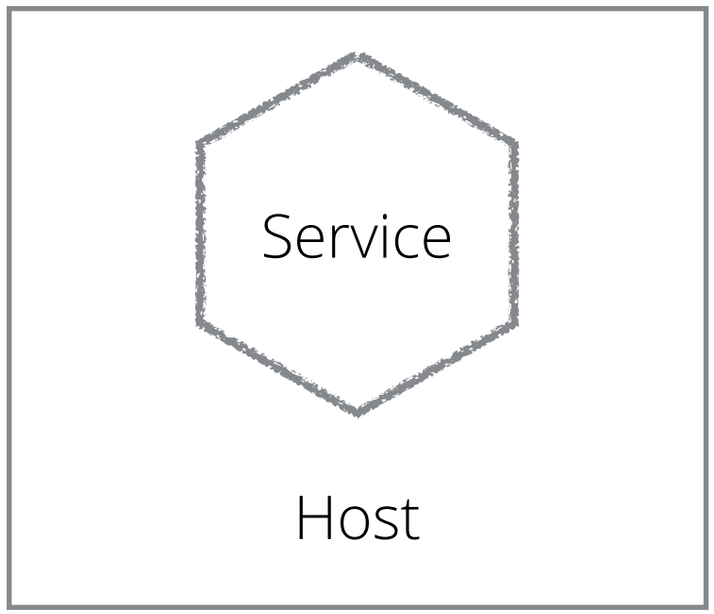
Decentralise All The Things ✓

Isolate Failure

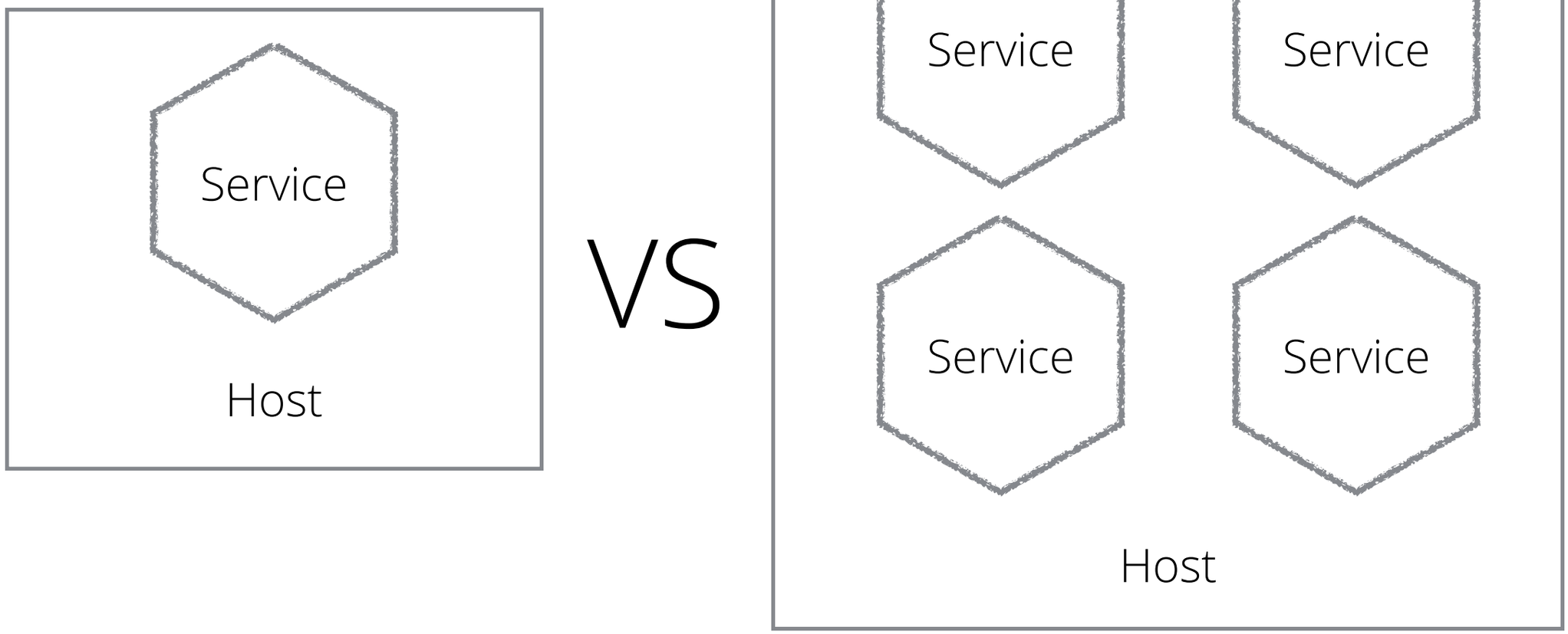
Consumer First

Deploy Independently

# ONE SERVICE PER-HOST



# ONE SERVICE PER-HOST



# CONSUMER-DRIVEN CONTRACTS

# CONSUMER-DRIVEN CONTRACTS

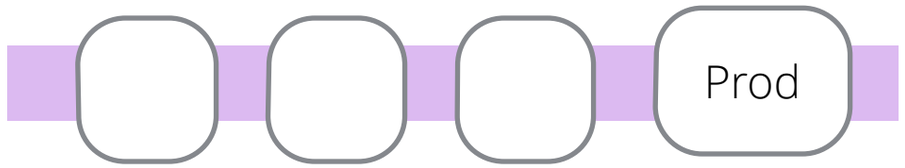

Expectations

# CONSUMER-DRIVEN CONTRACTS


Expectations

# CONSUMER-DRIVEN CONTRACTS


Expectations




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

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

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

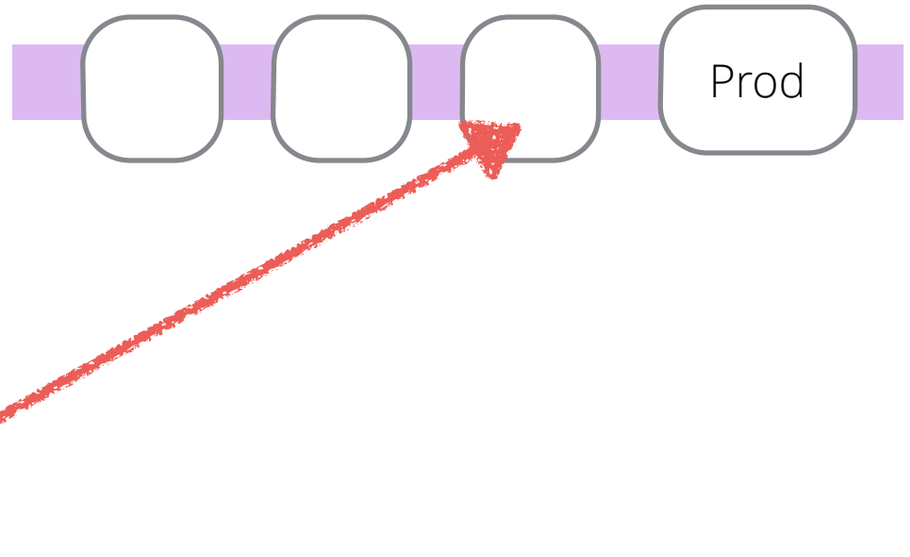
# CONSUMER-DRIVEN CONTRACTS


Expectations

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

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

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



README.md

# Pact

Define a pact between service consumers and providers, enabling "consumer driven contract" testing.

Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to provide a mock service provider. The interactions are recorded, and played back in the service provider specs to ensure the service provider actually does provide the response the consumer expects.

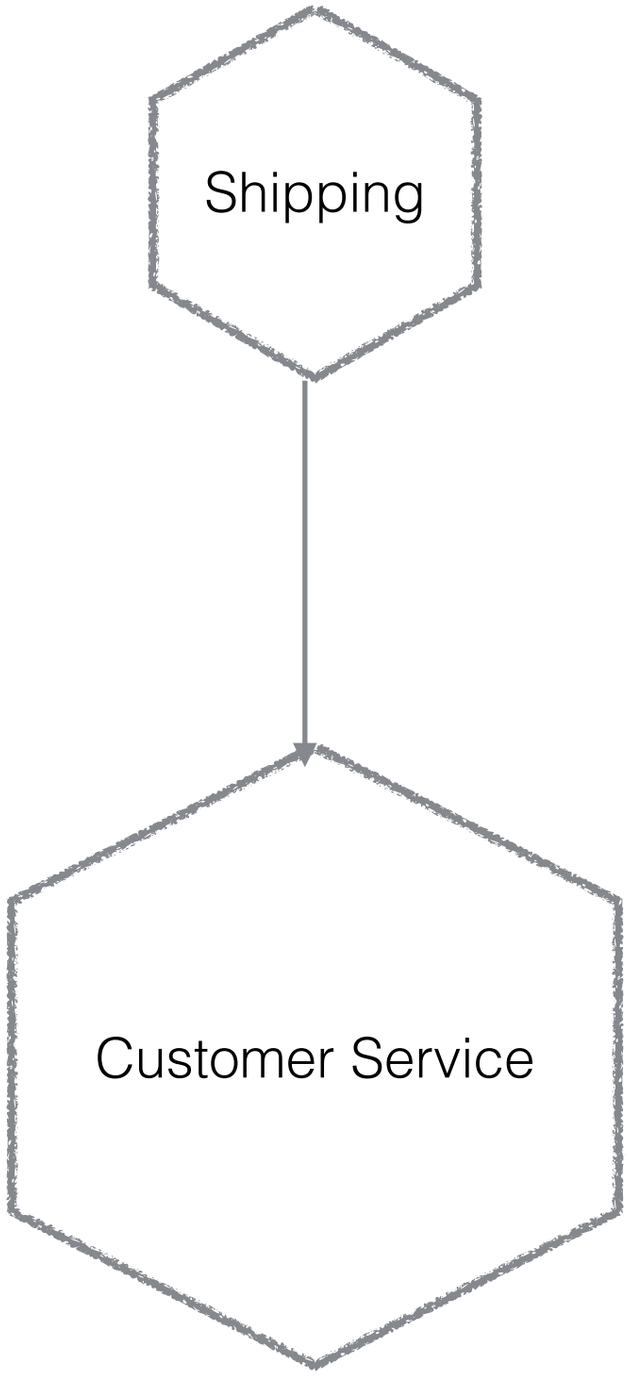
This allows testing of both sides of an integration point using fast unit tests.

This gem is inspired by the concept of "Consumer driven contracts". See http://martinfowler.com/articles/consumerDrivenContracts.html for more information.

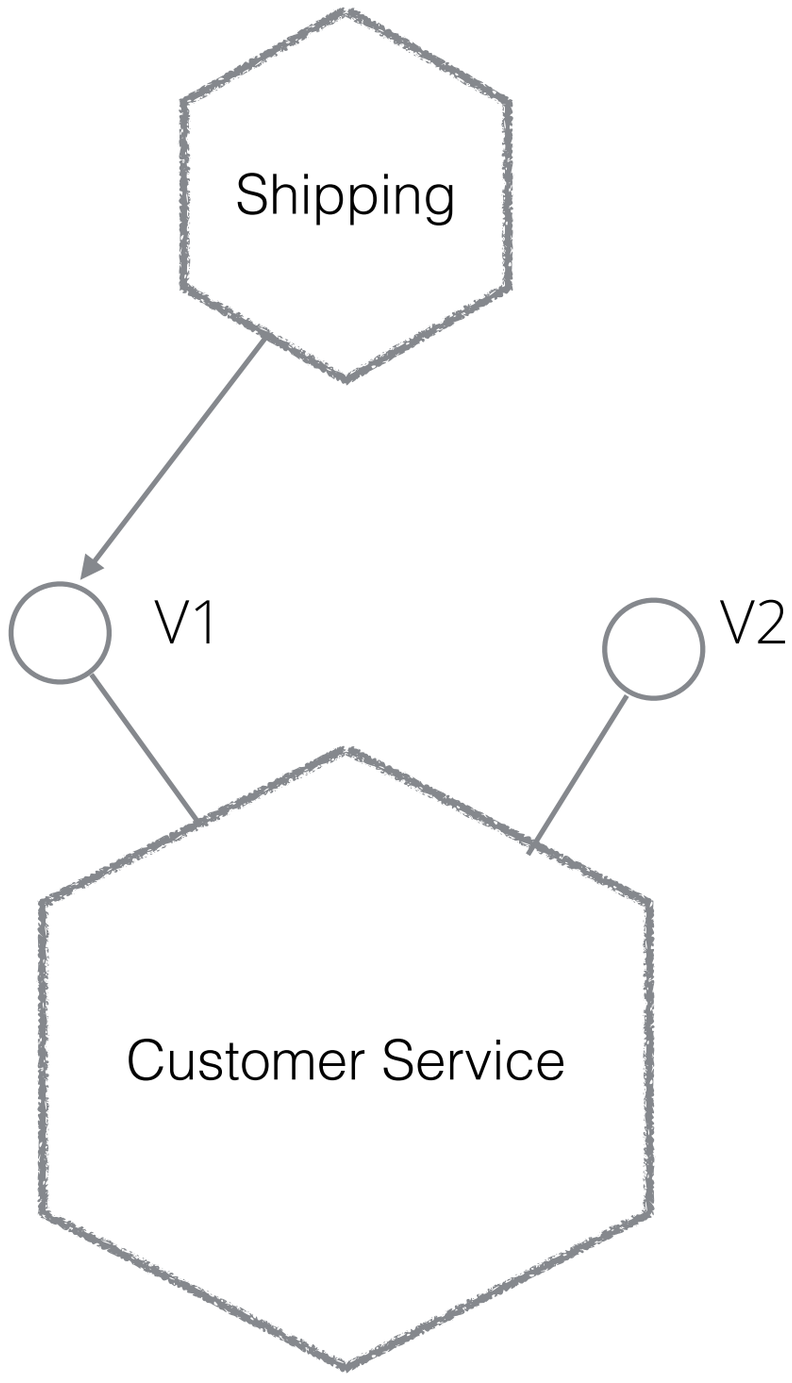
Travis CI Status: build passing

https://github.com/realestate-com-au/pact

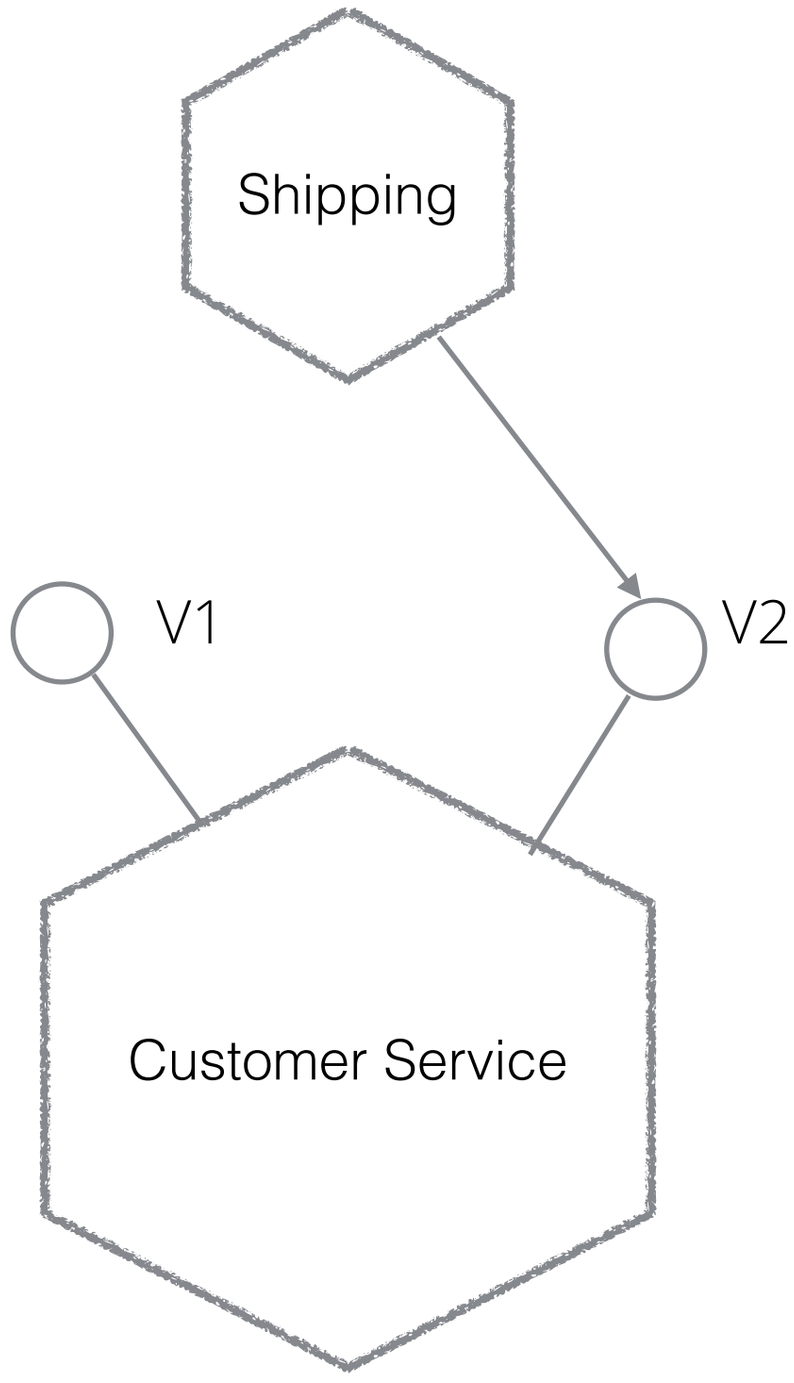
# CO-EXIST ENDPOINTS



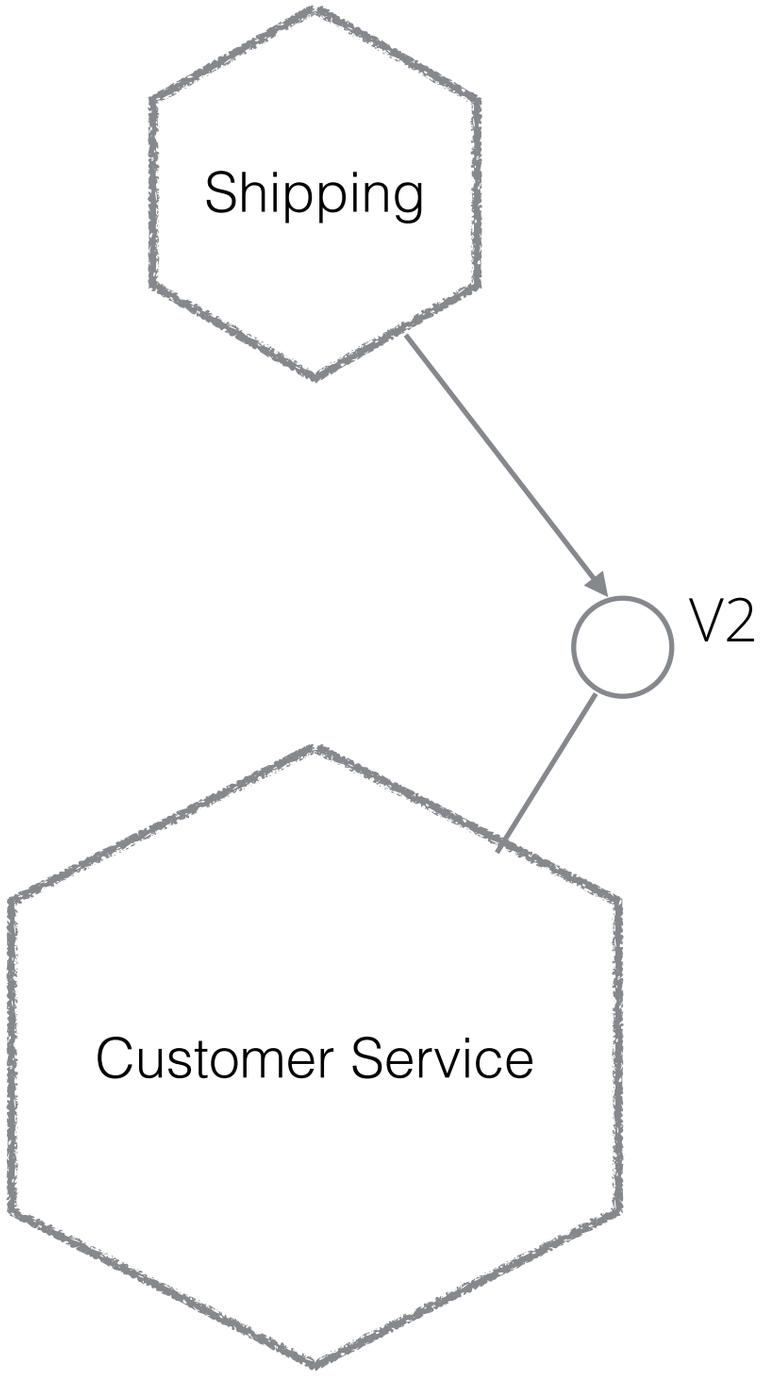
# CO-EXIST ENDPOINTS

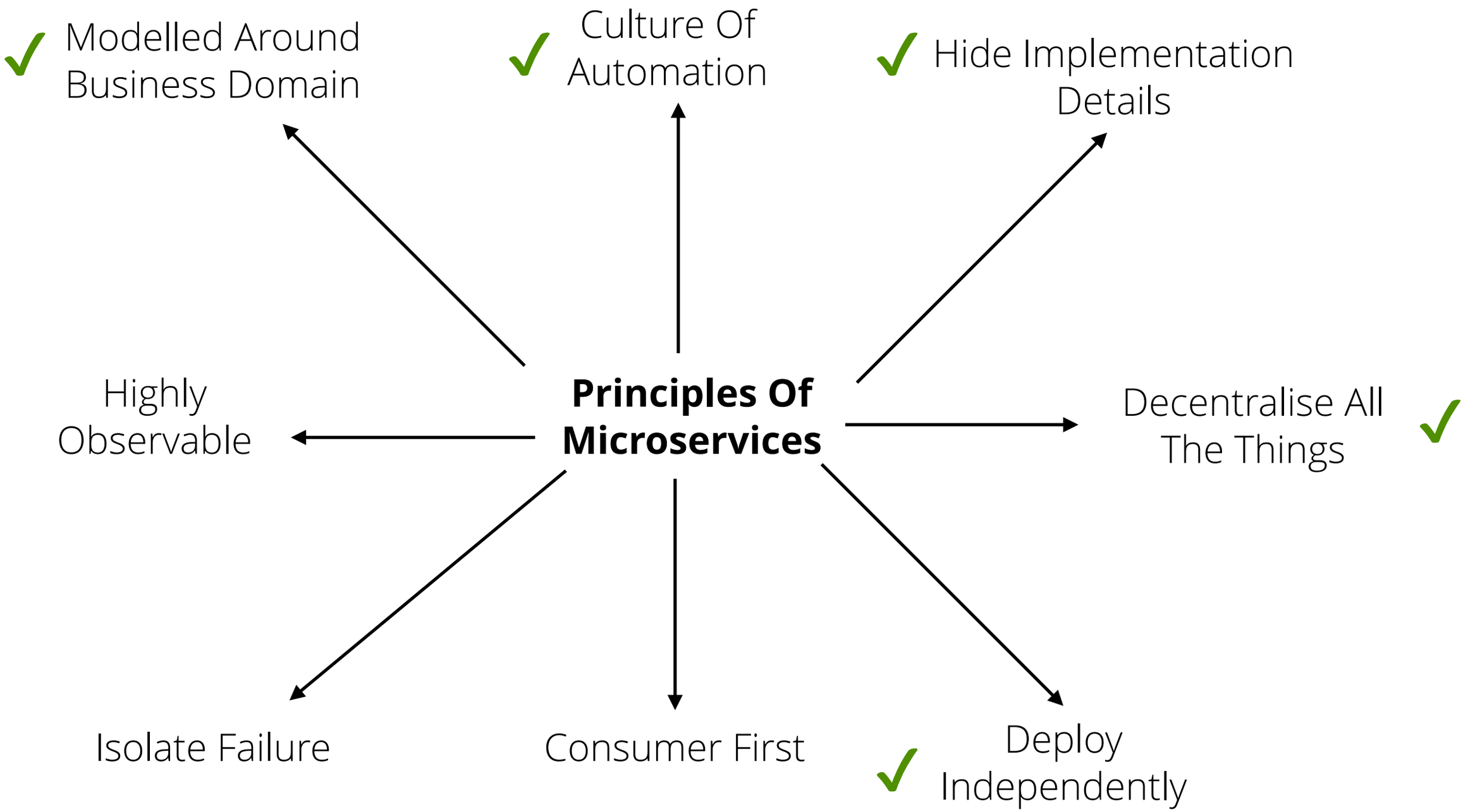


# CO-EXIST ENDPOINTS



# CO-EXIST ENDPOINTS




✓ Modelled Around Business Domain
✓ Culture Of Automation
✓ Hide Implementation Details
Highly Observable
Principles Of Microservices
Decentralise All The Things ✓
Isolate Failure
Consumer First
✓ Deploy Independently

✓ Modelled Around Business Domain

✓ Culture Of Automation

✓ Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things ✓

Isolate Failure

Consumer First ✓

Deploy Independently

**DOCUMENTATION**


## A POWERFUL INTERFACE TO YOUR API

Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.

Response Content Type application/json

Try it out!

POST /store/order — Place an order for a pet

Response Class (Status 200)

Model | Model Schema


```
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2015-05-11T05:03:29.879Z",
  "status": "placed",
  "complete": true
}
```

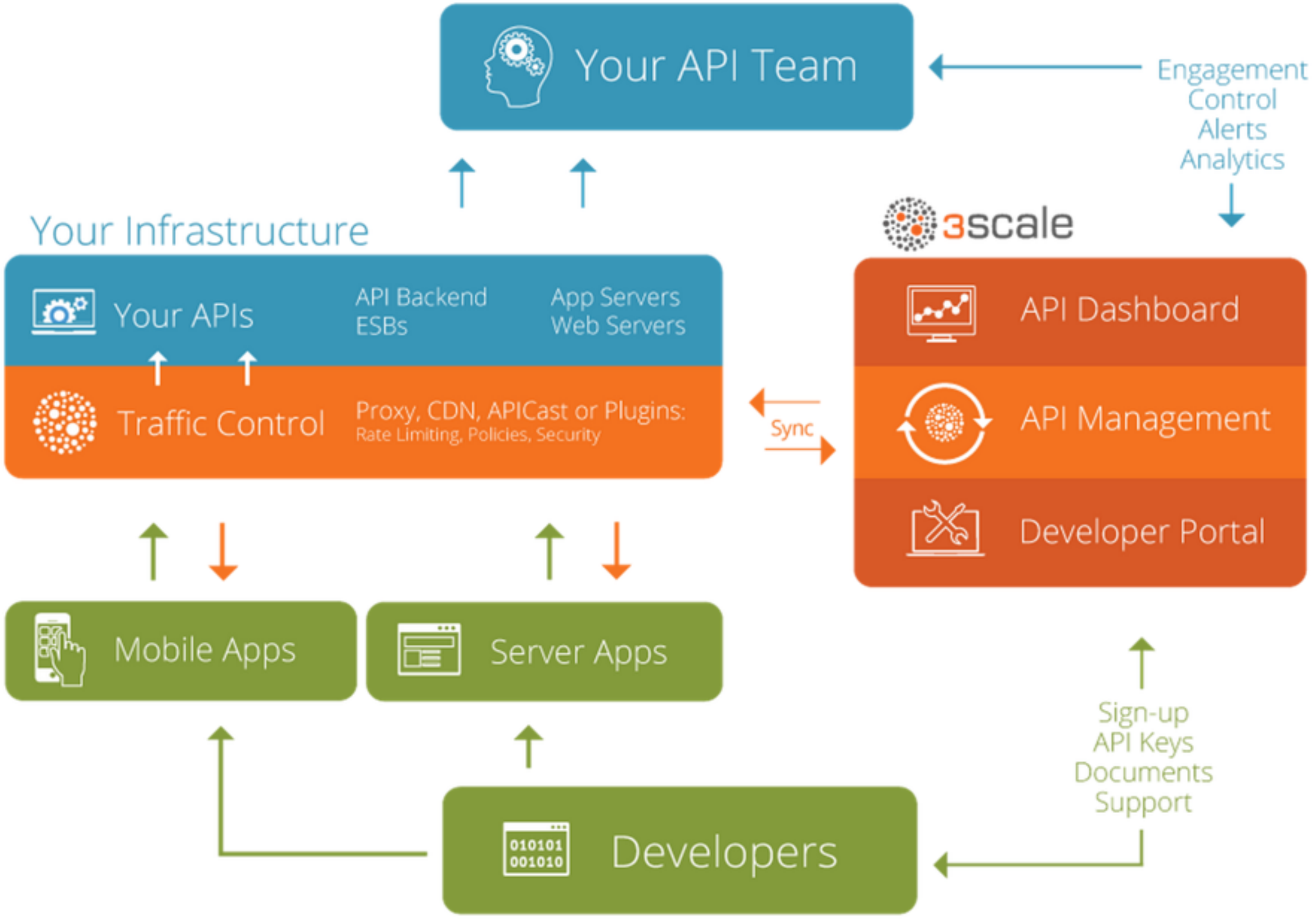

Response Content Type application/xml

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| body | Parameter content type: application/json | order placed for purchasing the pet | body | Model \| Model Schema |


```
{
  "id": 0,
  "petId": 0,
  "quantity": 0,
  "shipDate": "2015-05-11T05:03:29.800Z",
  "status": "placed",
  "complete": true
}
```

# API GATEWAYS



# SERVICE DISCOVERY

# SERVICE DISCOVERY


# SERVICE DISCOVERY



# HUMANE REGISTRIES

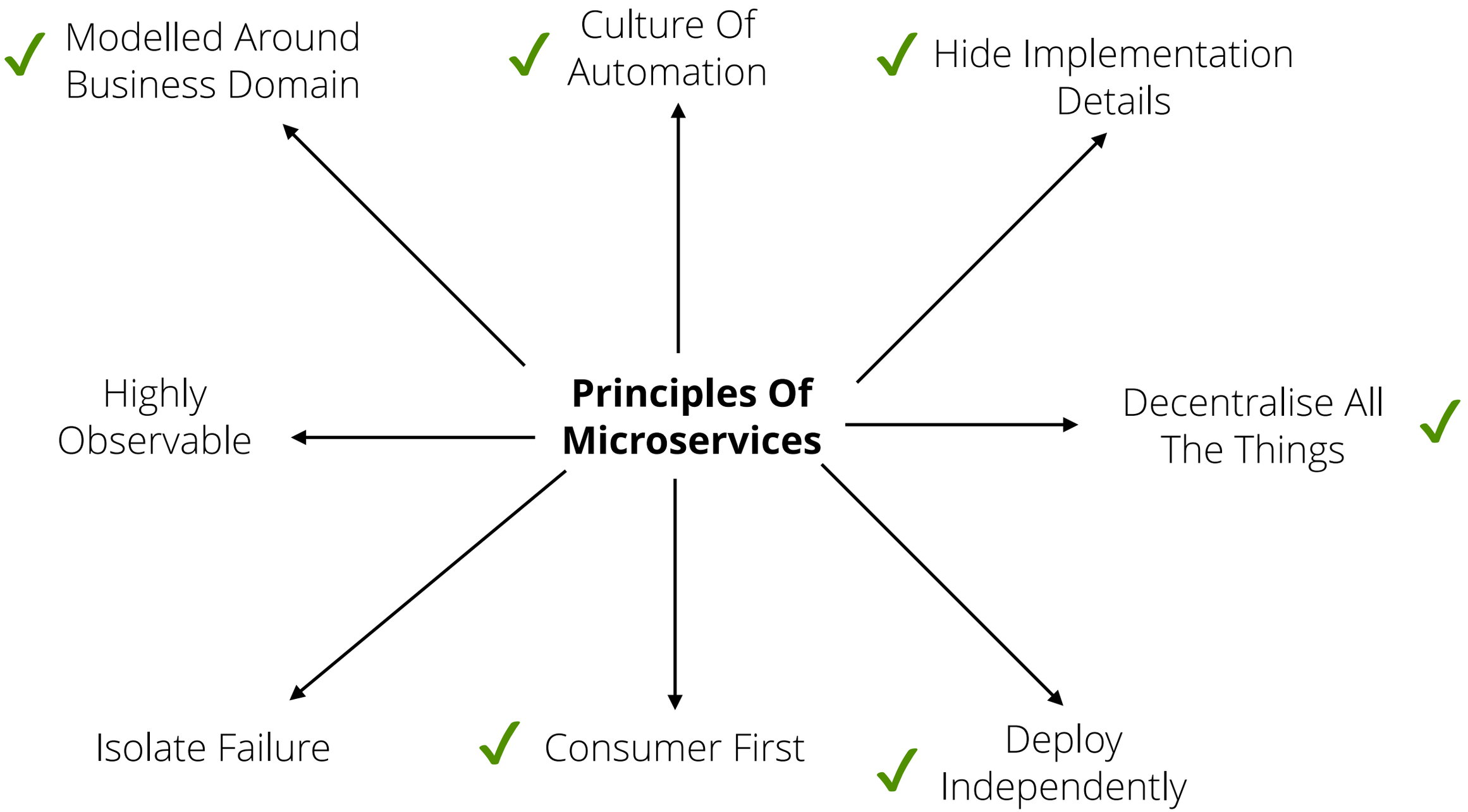
## HumaneRegistry

*Martin Fowler*
*1 December 2008*

One of the features of the new world of services that SOA-gushers promoted was the notion of registries. Often this was described in terms of automated systems that would allow systems to automatically look up useful services in a registry and bind and consume those services all by themselves.

Well computers may look clever occasionally, but I didn't particularly buy that idea. While there might the be odd edge case for automated service lookup, I reckon twenty-two times out of twenty it'll be a human programmer who is doing the looking up.

I was chatting recently to my colleague Erik Dörnenburg about a project he did with Halvard Skogsrud to build a service registry that was designed for humans to use and maintain. The organization was already using ServiceCustodians to manage the development on the project, so the registry needed to work in that context. This led to the following principles:


✓ Modelled Around Business Domain
✓ Culture Of Automation
✓ Hide Implementation Details
Highly Observable
Principles Of Microservices
Decentralise All The Things
✓
Isolate Failure
✓ Consumer First
✓ Deploy Independently

✓ Modelled Around Business Domain

✓ Culture Of Automation

✓ Hide Implementation Details

Highly Observable

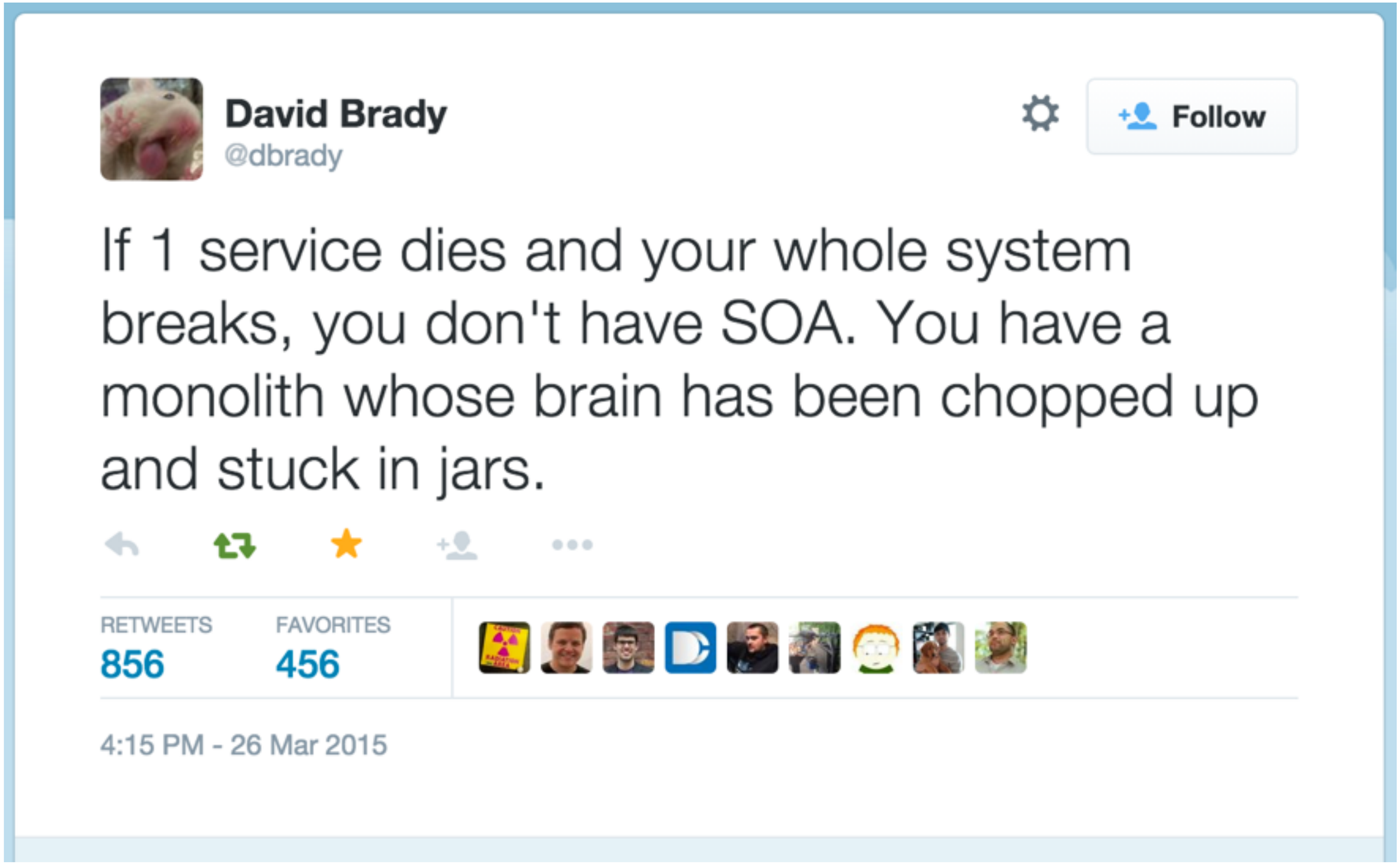
**Principles Of Microservices**

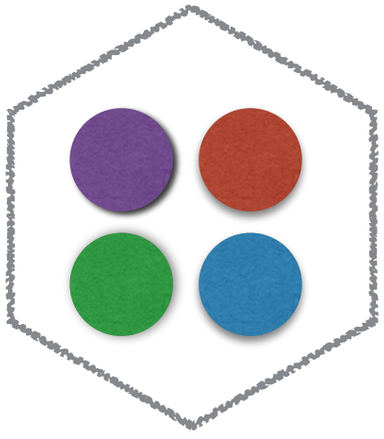
Decentralise All The Things ✓

Isolate Failure

✓ Consumer First

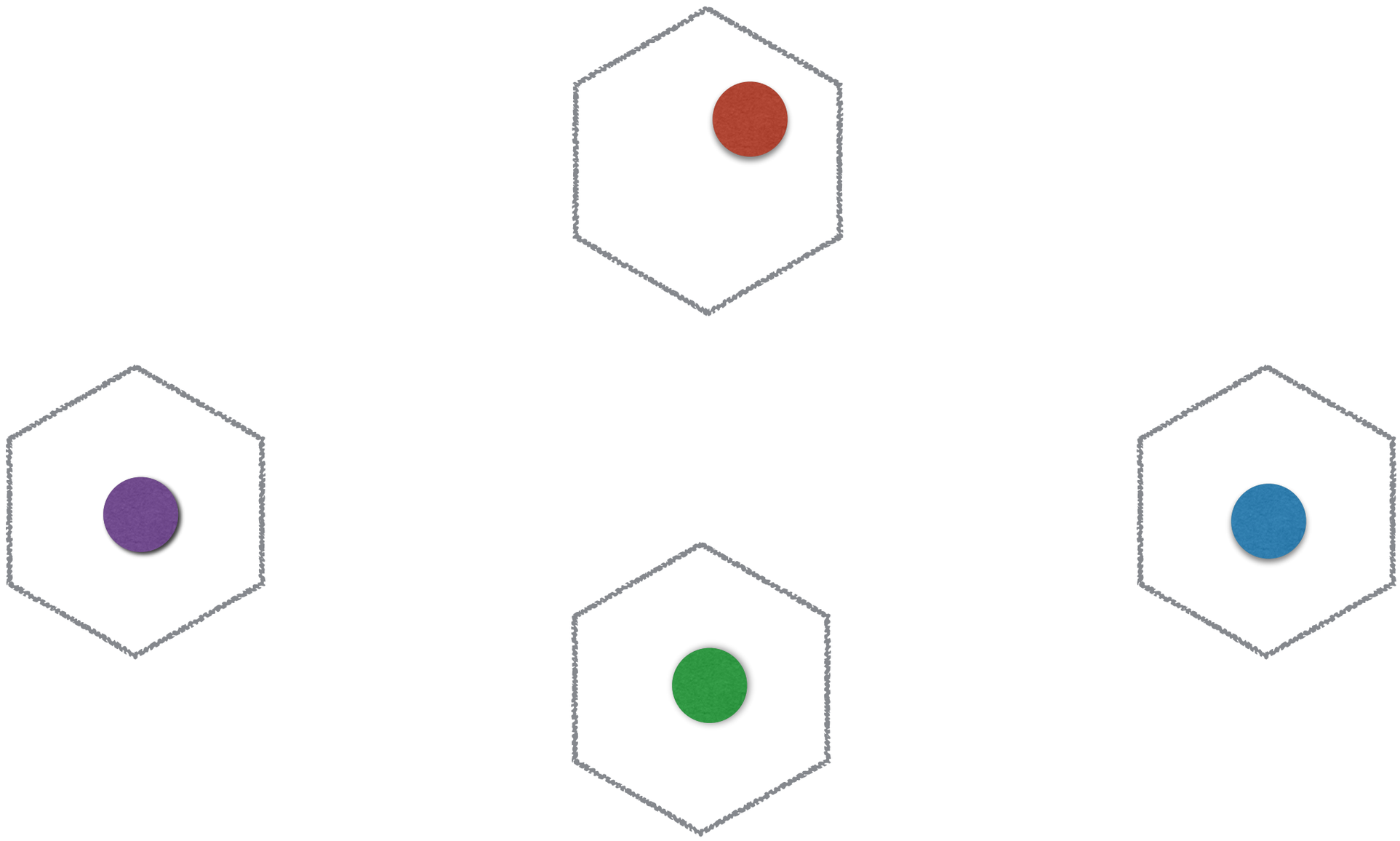
✓ Deploy Independently

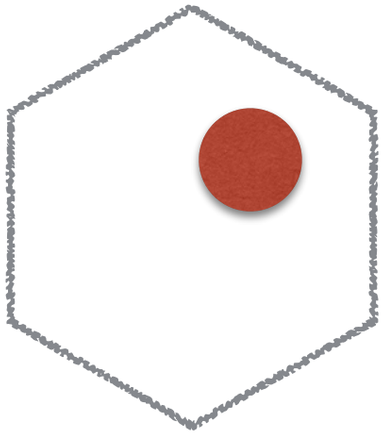

David Brady
@dbrady
Follow
If 1 service dies and your whole system breaks, you don't have SOA. You have a monolith whose brain has been chopped up and stuck in jars.
RETWEETS
856
FAVORITES
456
4:15 PM - 26 Mar 2015

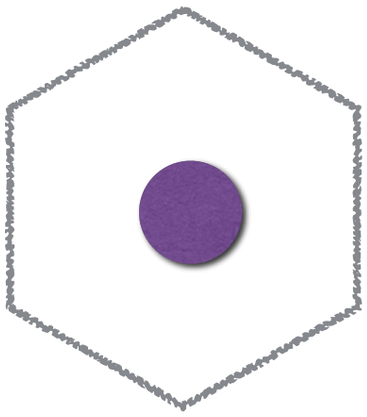


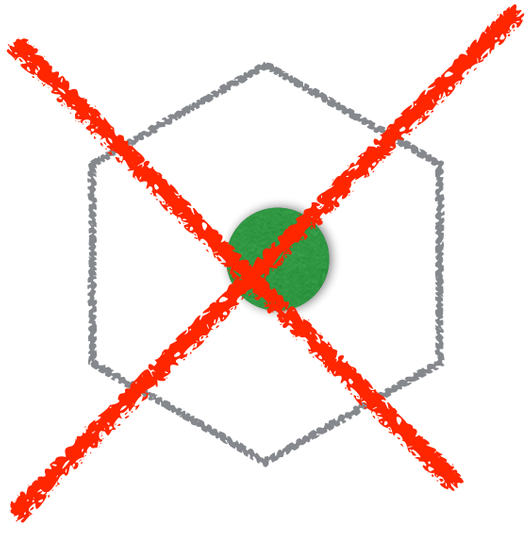
@velocityconf

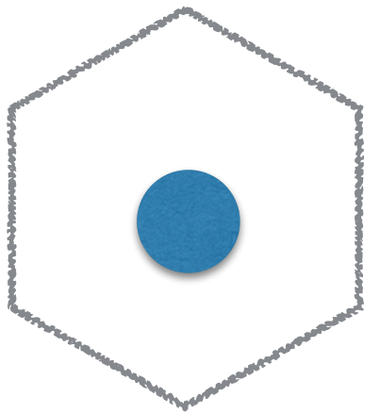
@samnewman

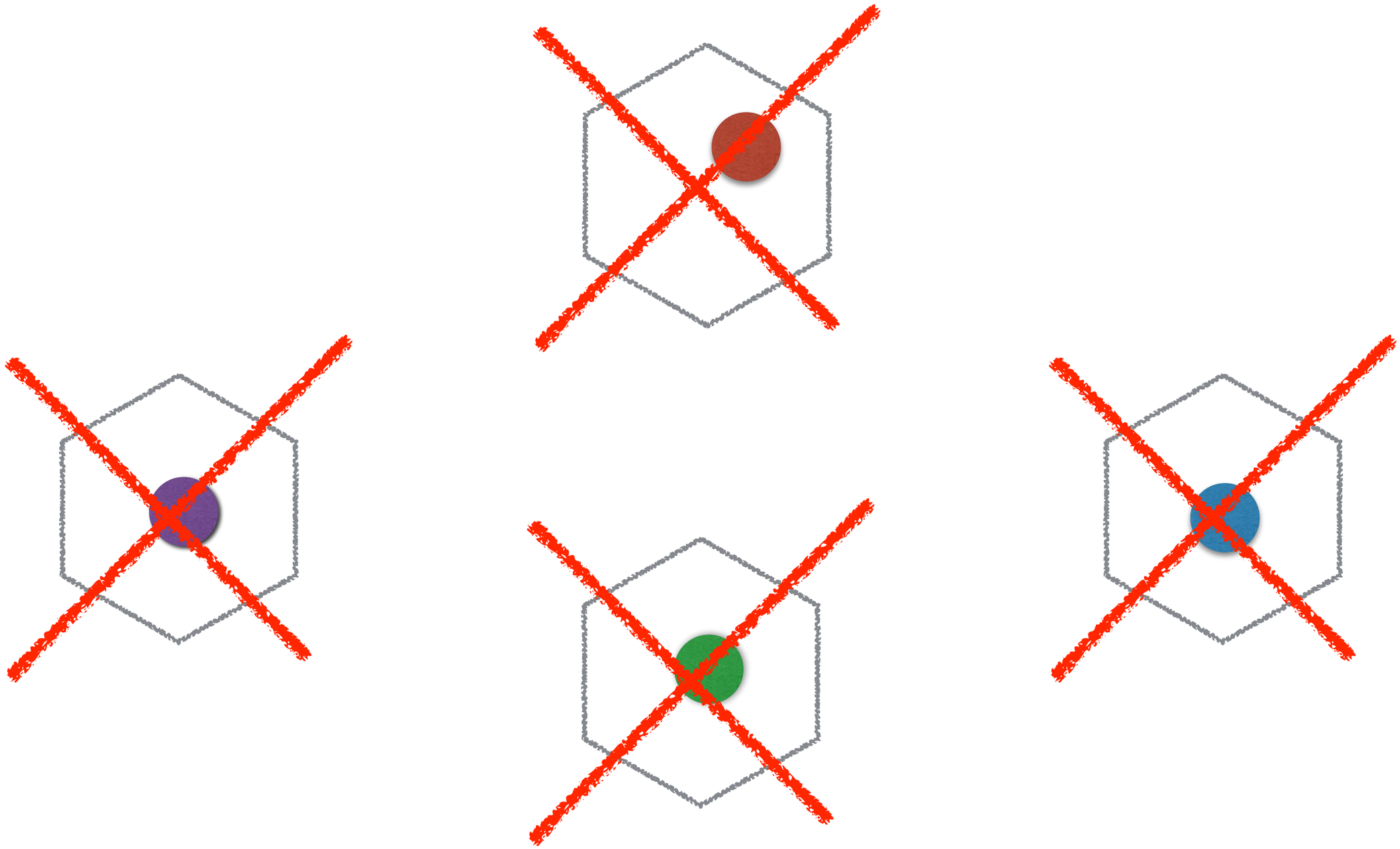




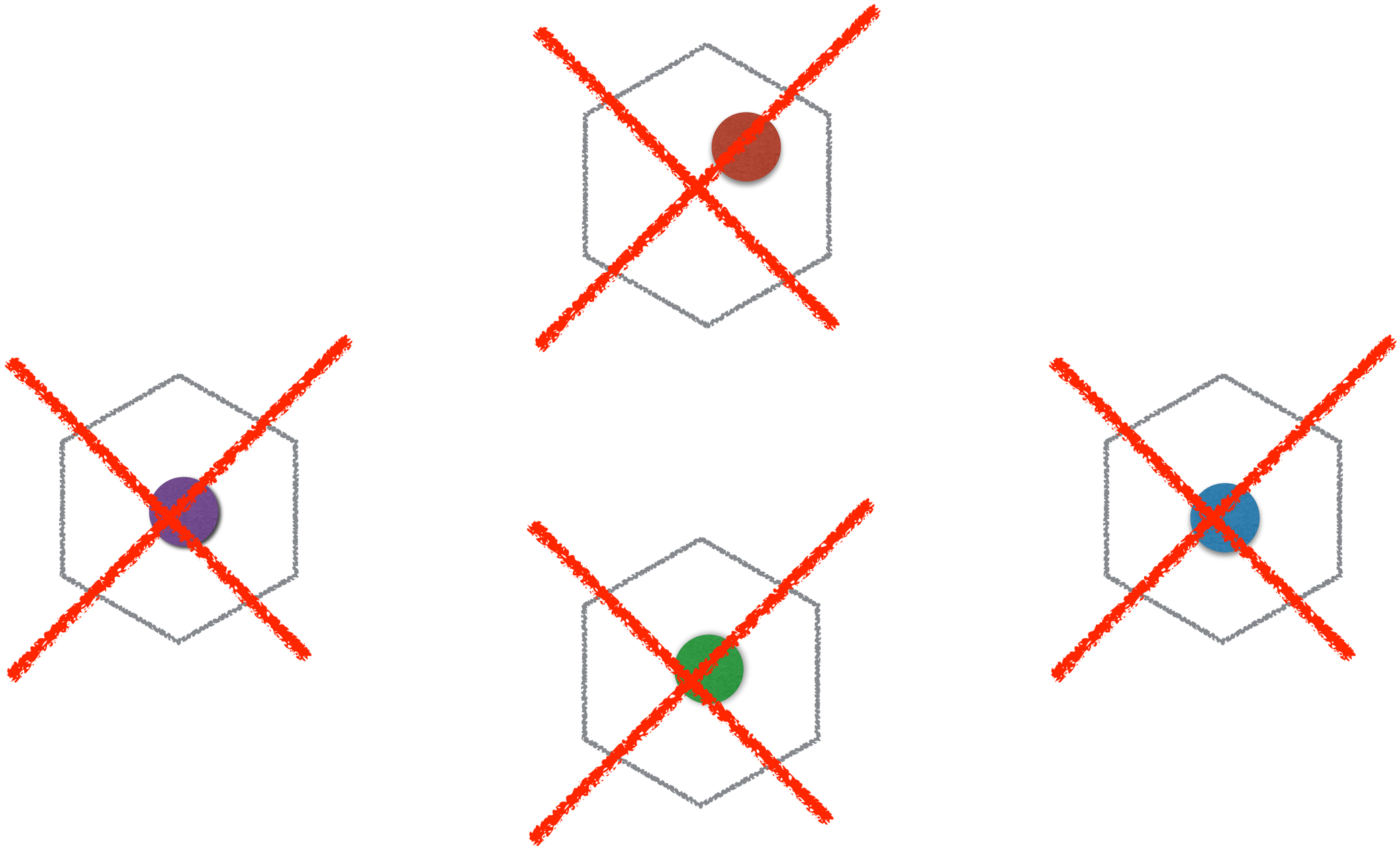








# AVOID THE DISTRIBUTED SINGLE POINT OF FAILURE!



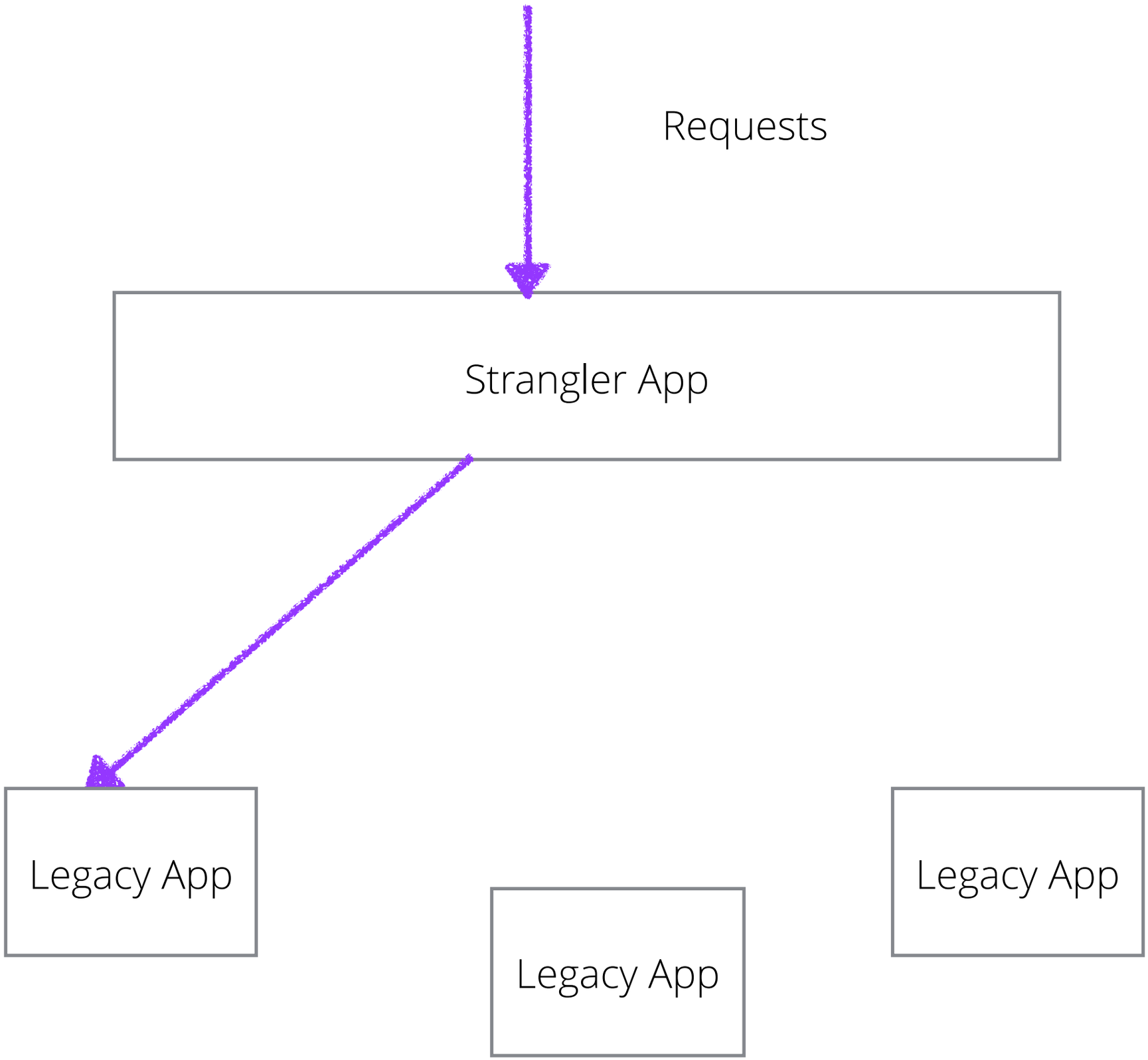
Strangler App

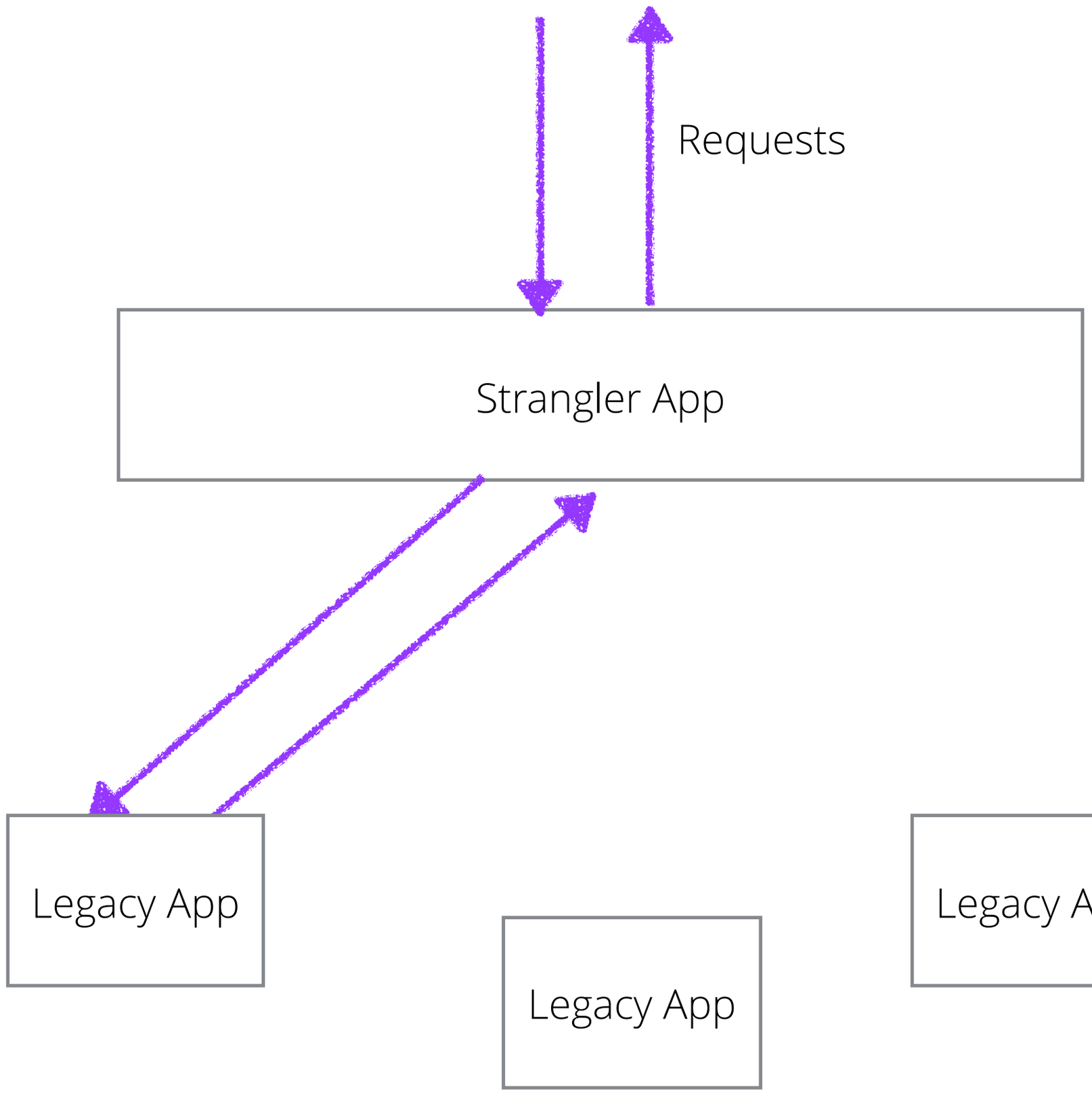
Strangler App

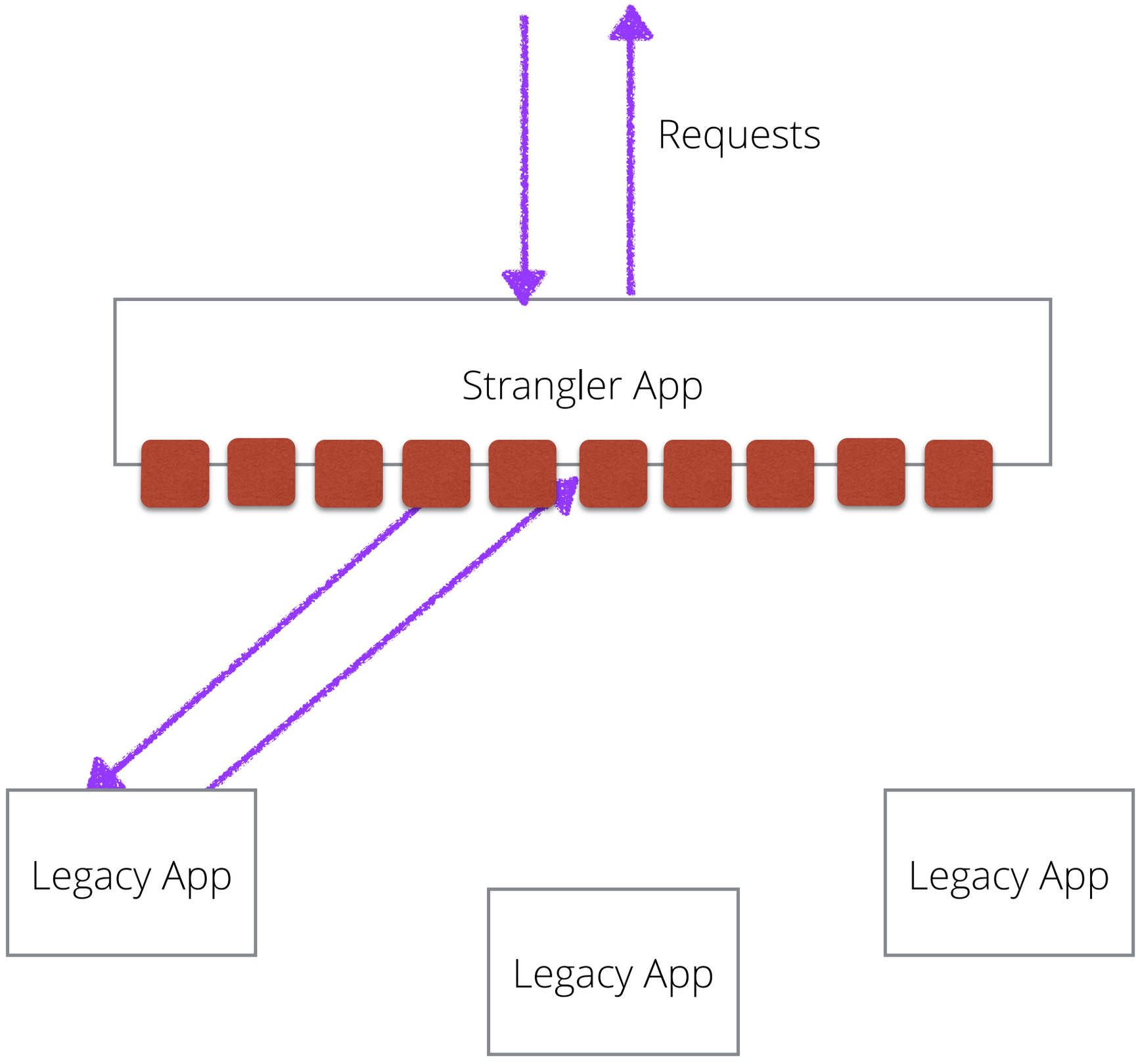
Legacy App

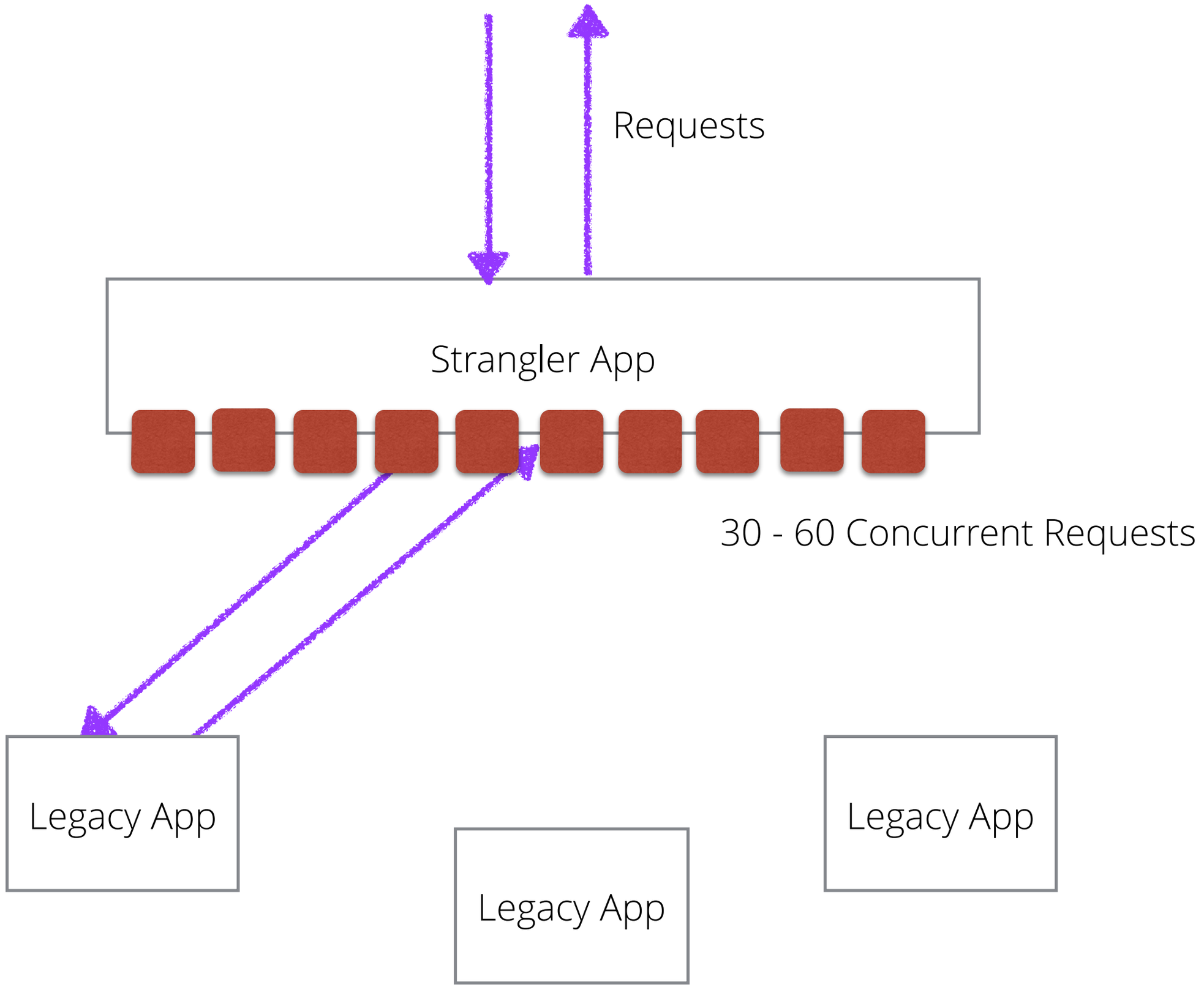
Legacy App

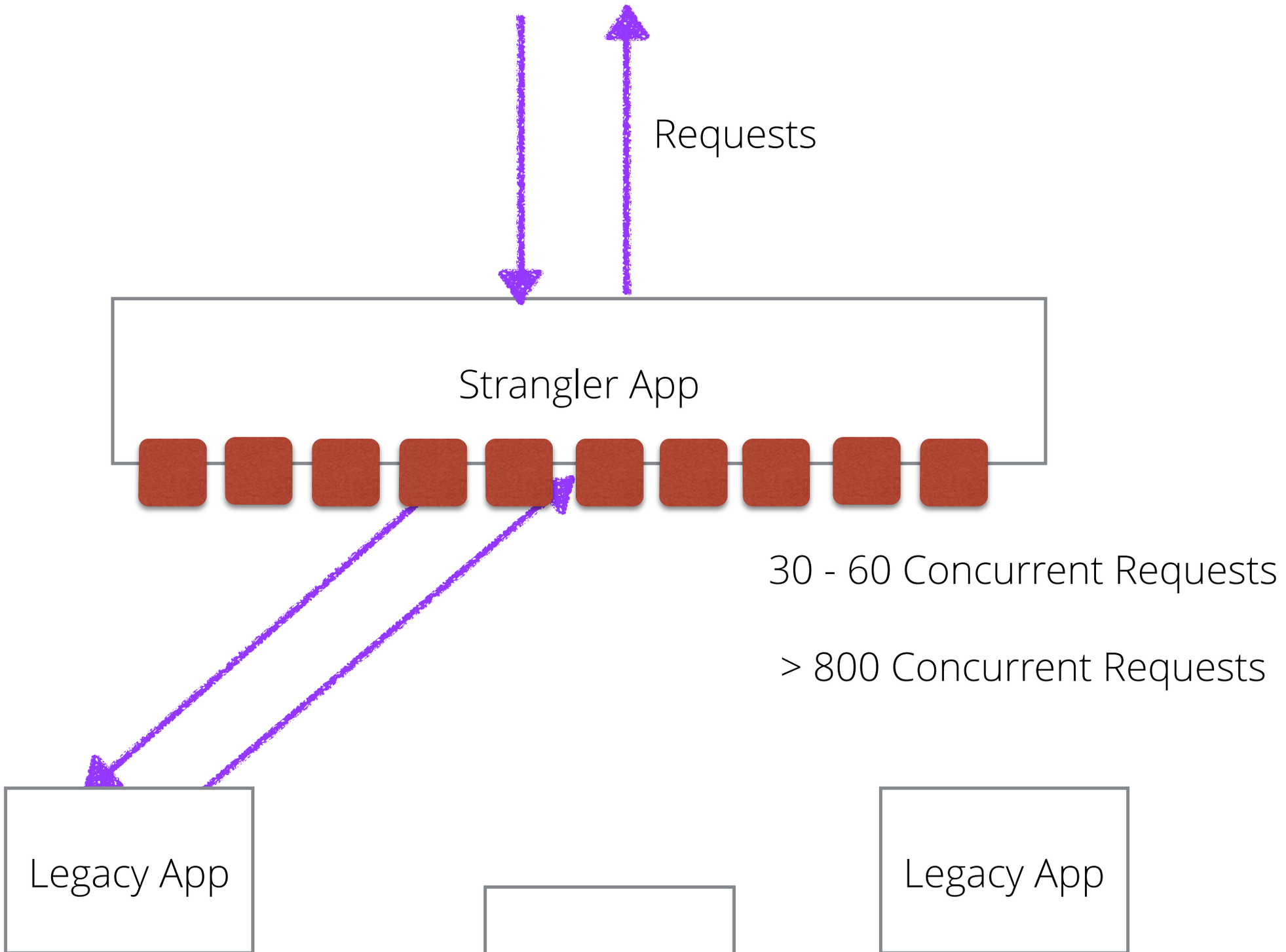
Legacy App


Requests
Strangler App
Legacy App
Legacy App
Legacy App


Requests
Strangler App
Legacy App
Legacy App
Legacy App

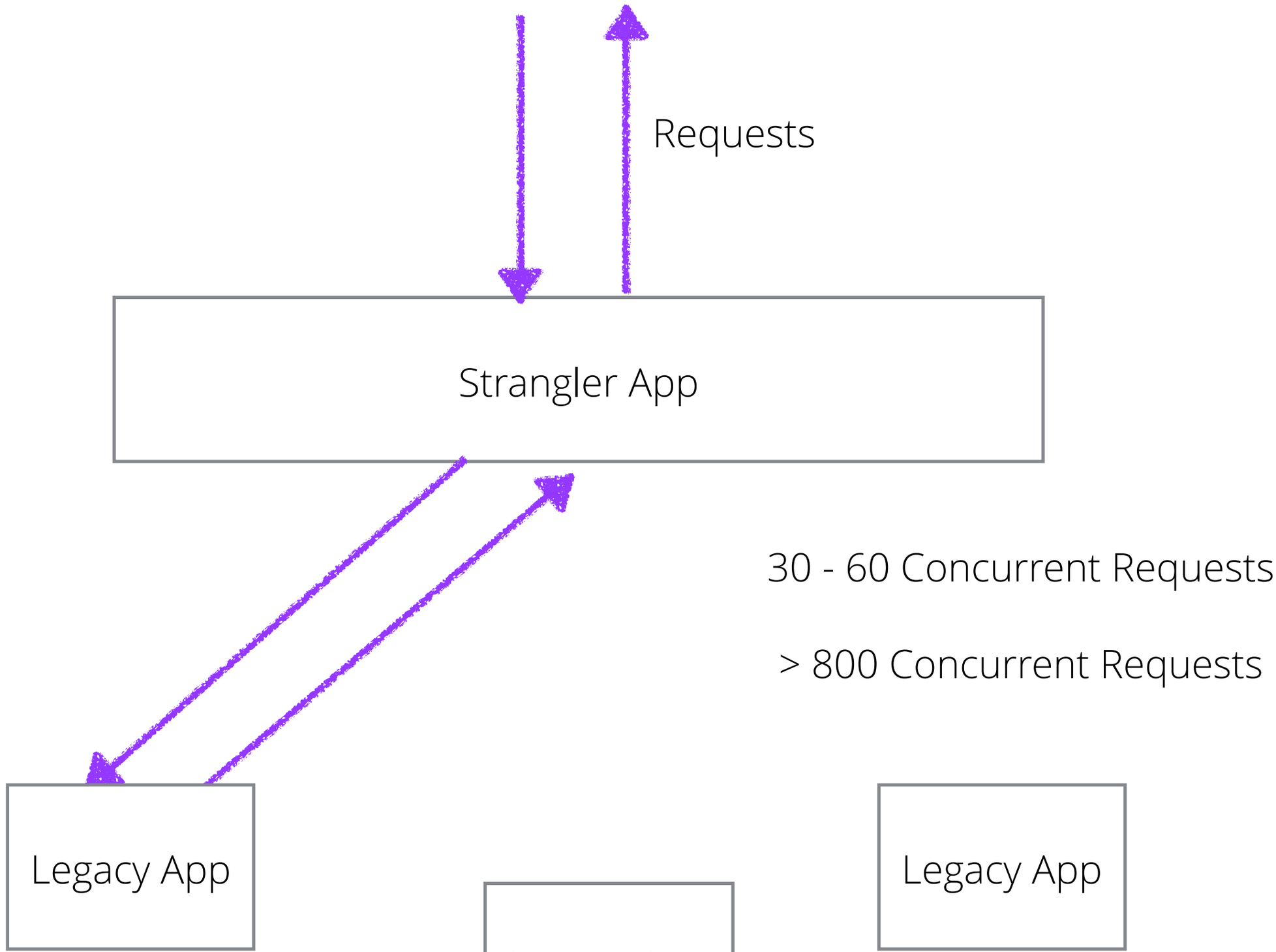

Requests
Strangler App
Legacy App
Legacy App
Legacy App


Requests
Strangler App
30 - 60 Concurrent Requests
Legacy App
Legacy App
Legacy App




30 - 60 Concurrent Requests

> 800 Concurrent Requests

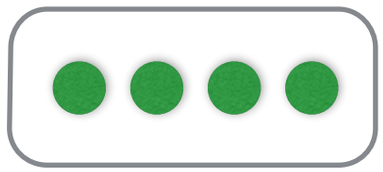



30 - 60 Concurrent Requests

> 800 Concurrent Requests

Strangler App

*Thread Pool*



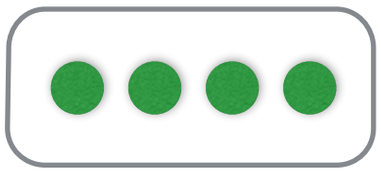
Legacy App

Legacy App

Legacy App

Strangler App

*Thread Pool*

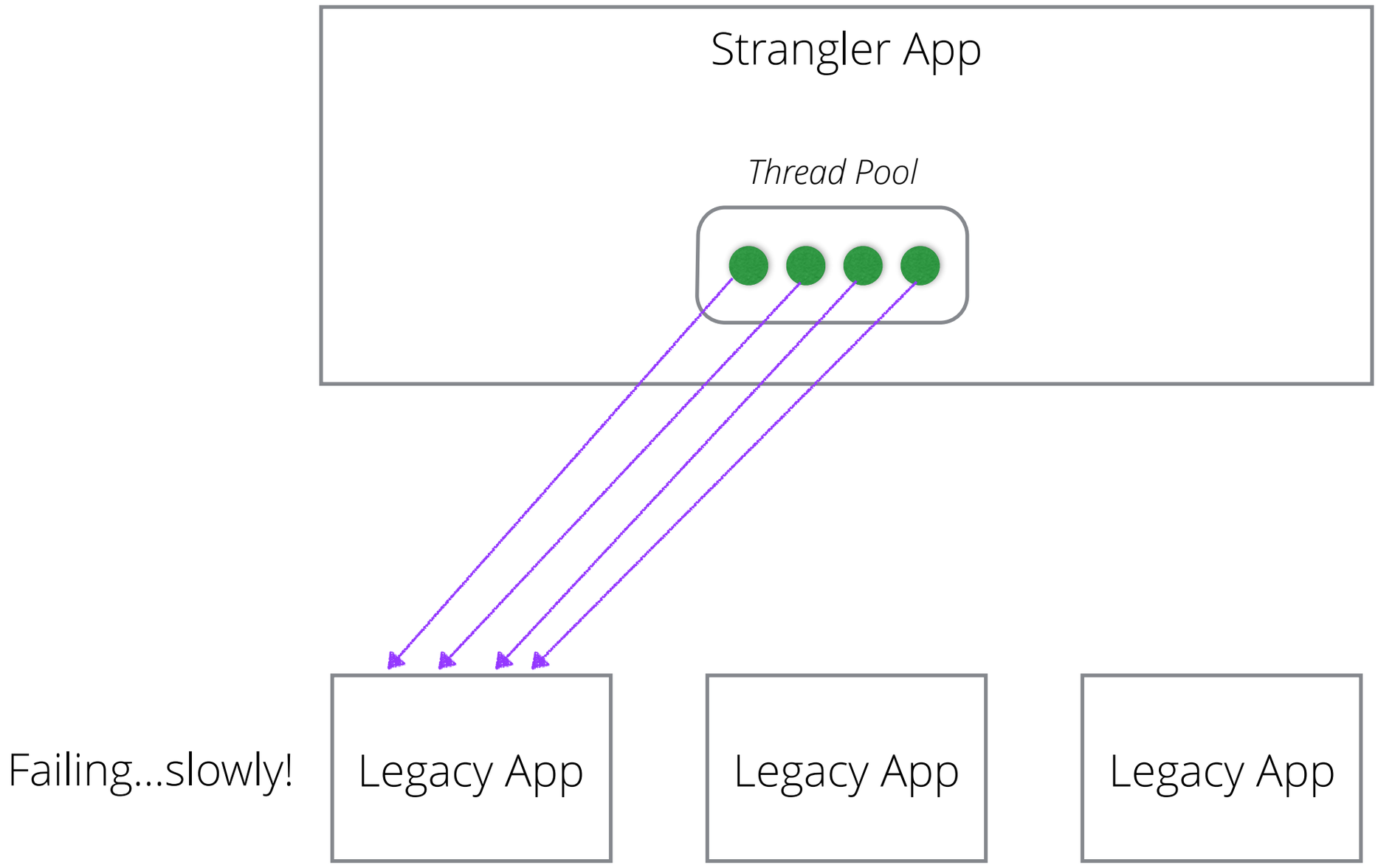


Failing...slowly!

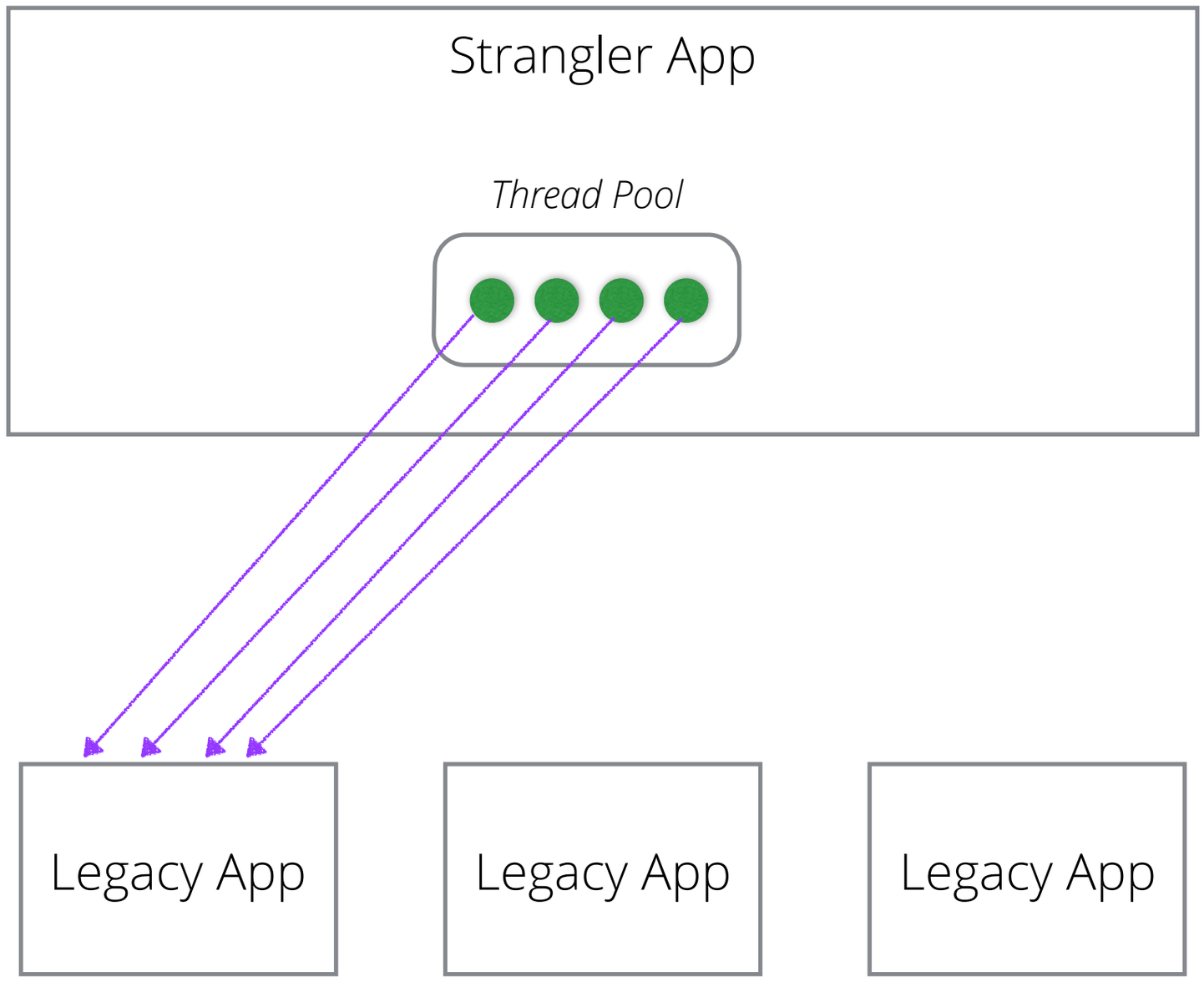
Legacy App

Legacy App

Legacy App

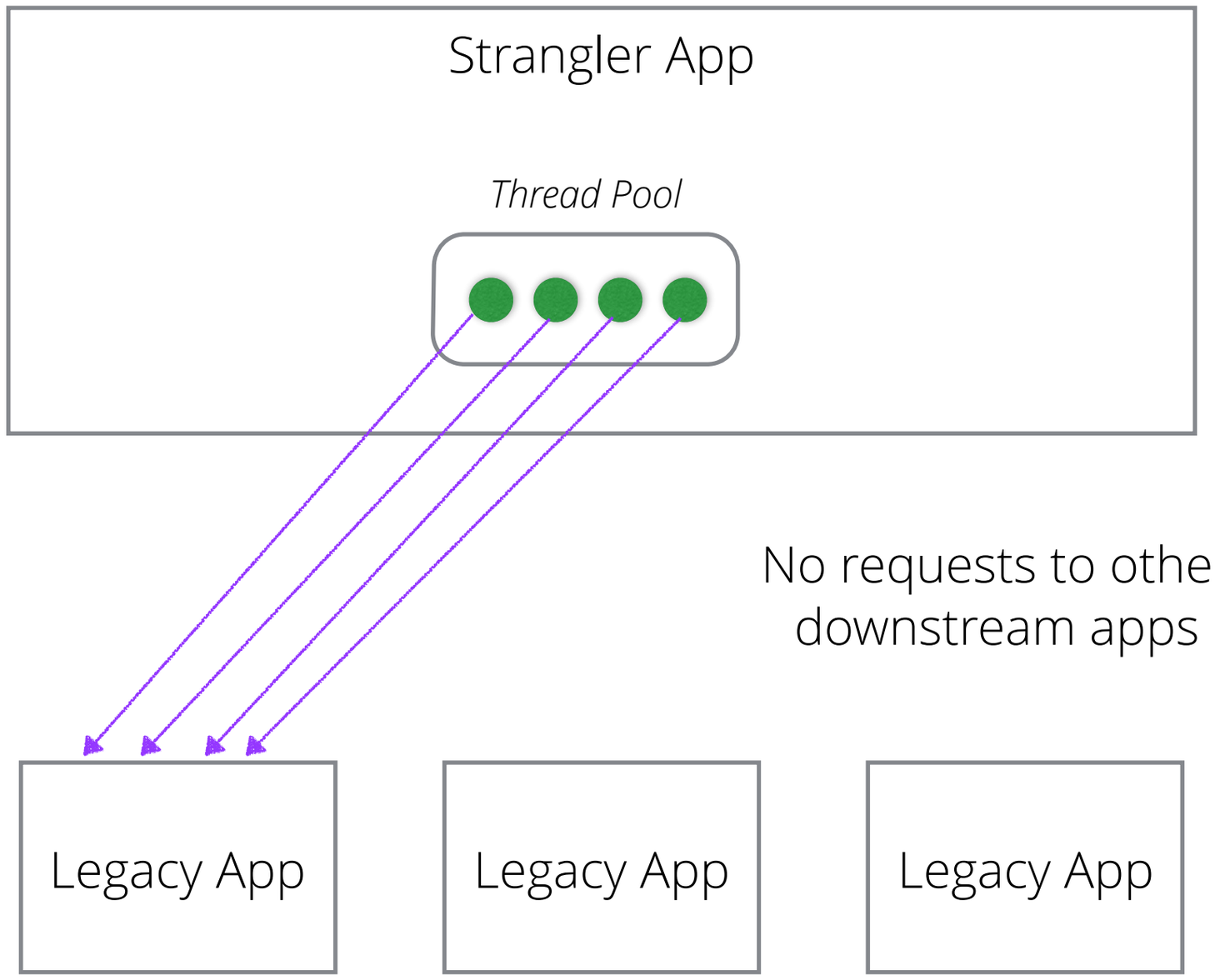

Strangler App
Thread Pool
Failing...slowly!
Legacy App
Legacy App
Legacy App

Thread-pool exhausted

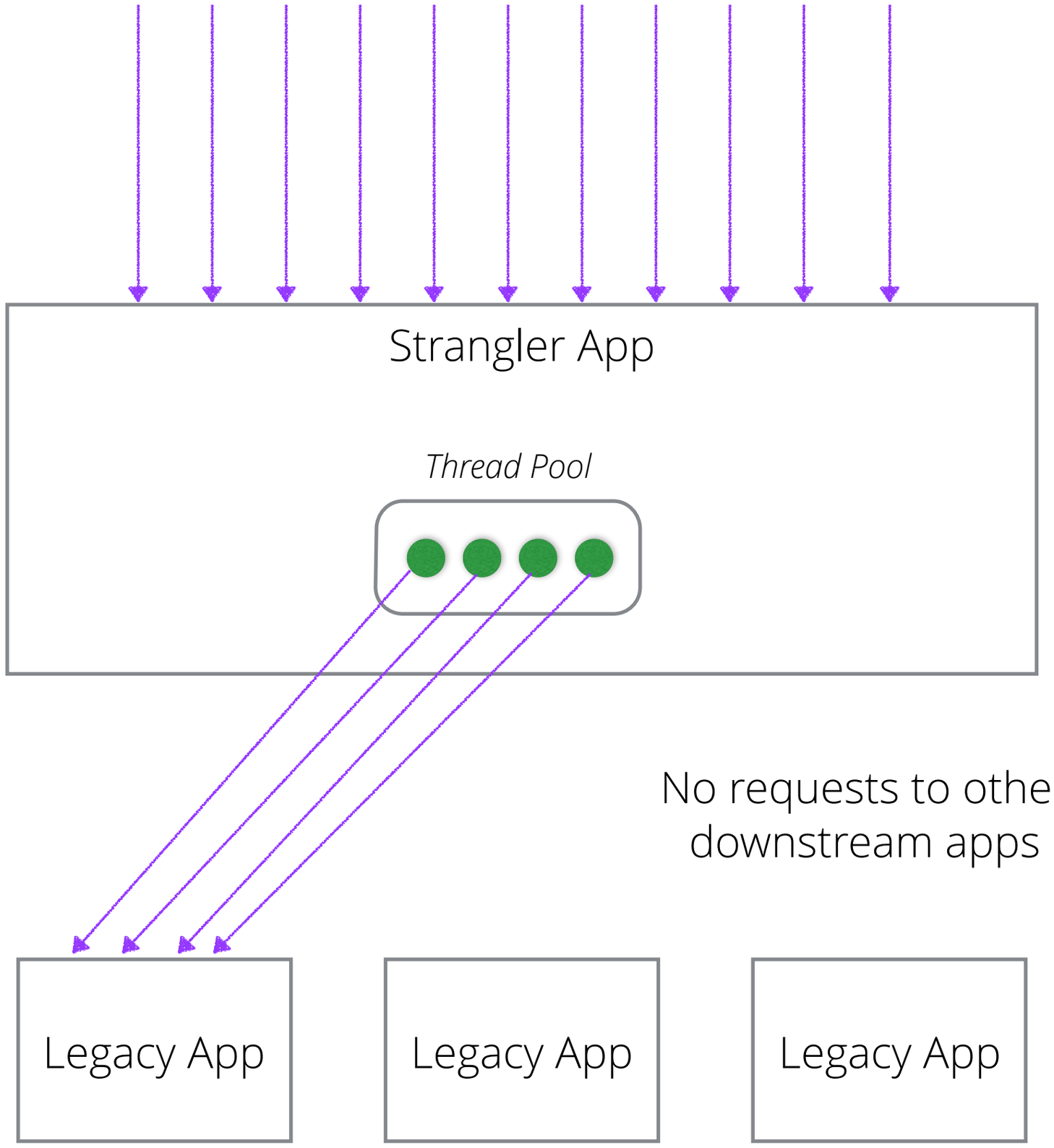



Failing...slowly!

Thread-pool exhausted



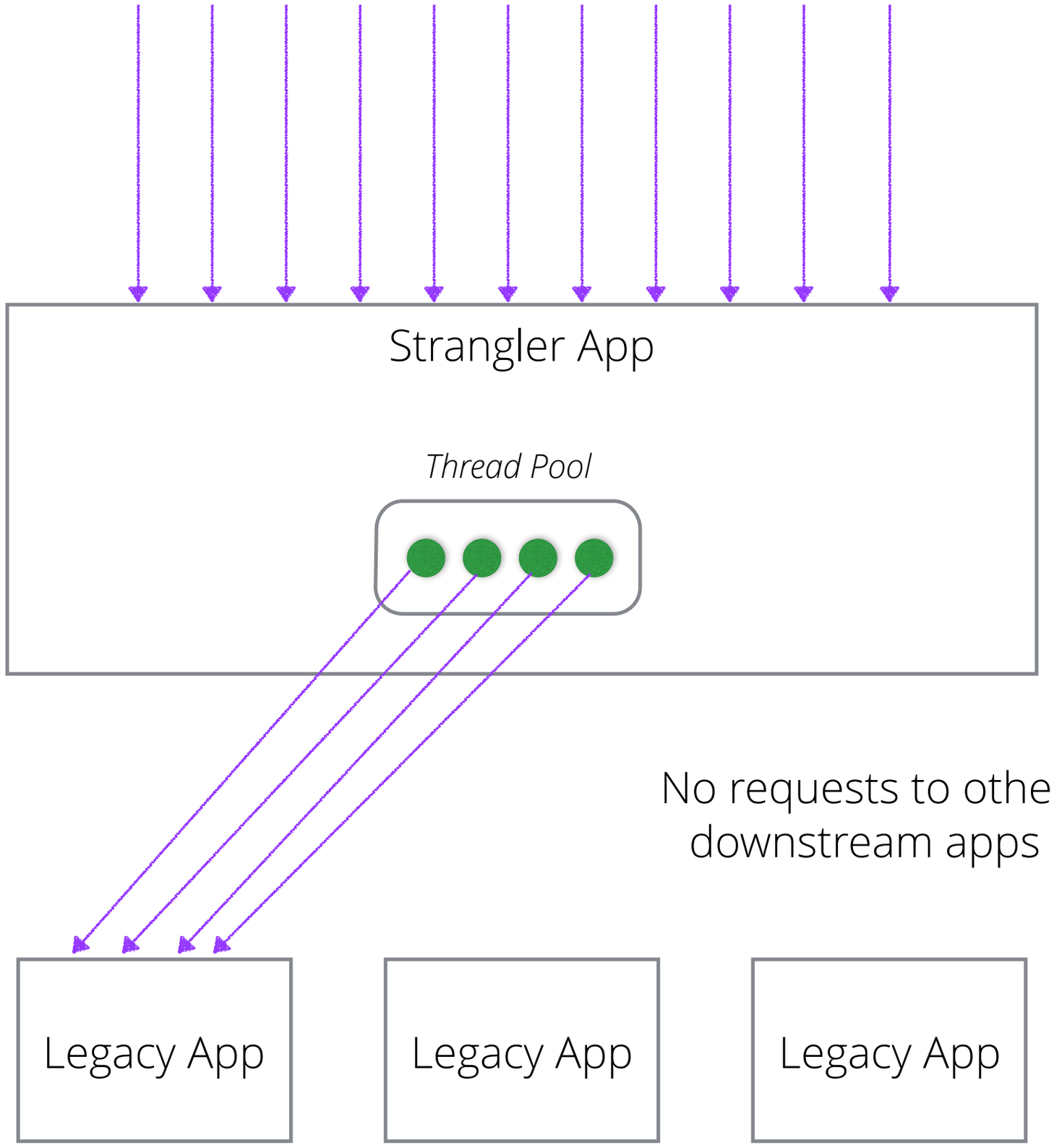
Thread-pool exhausted




No requests to other downstream apps

Failing...slowly!

Requests Building Up



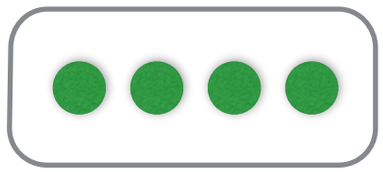

Thread-pool exhausted

No requests to other downstream apps

Failing...slowly!

Strangler App

*Thread Pool*

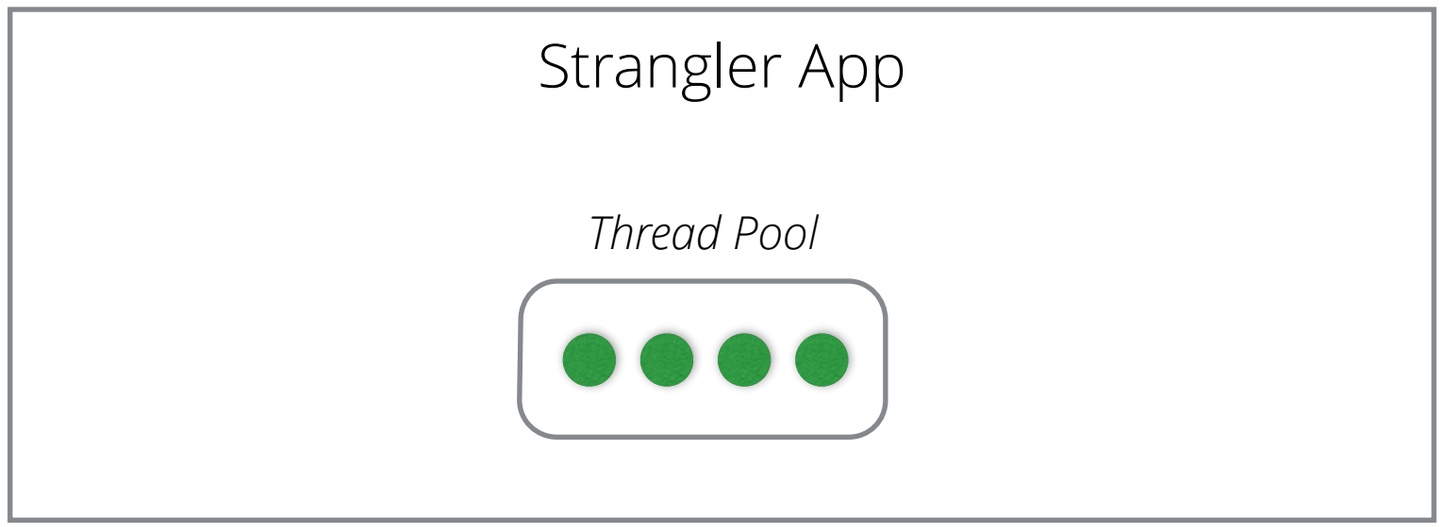


Legacy App

Legacy App

Legacy App

Fix **Timeouts**

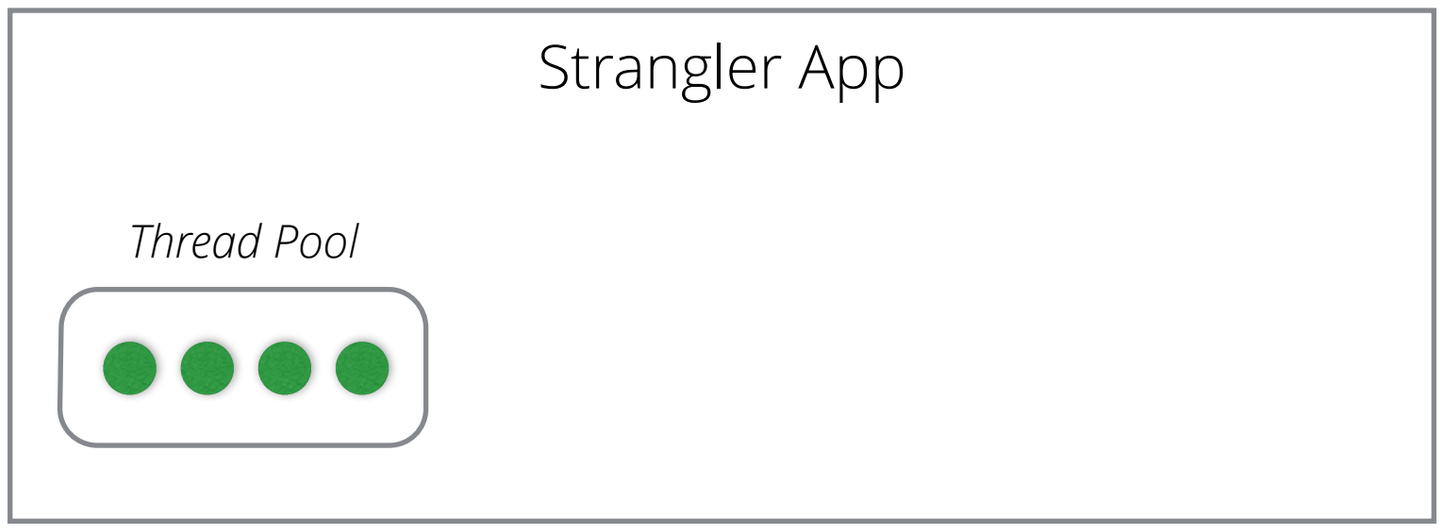



Legacy App

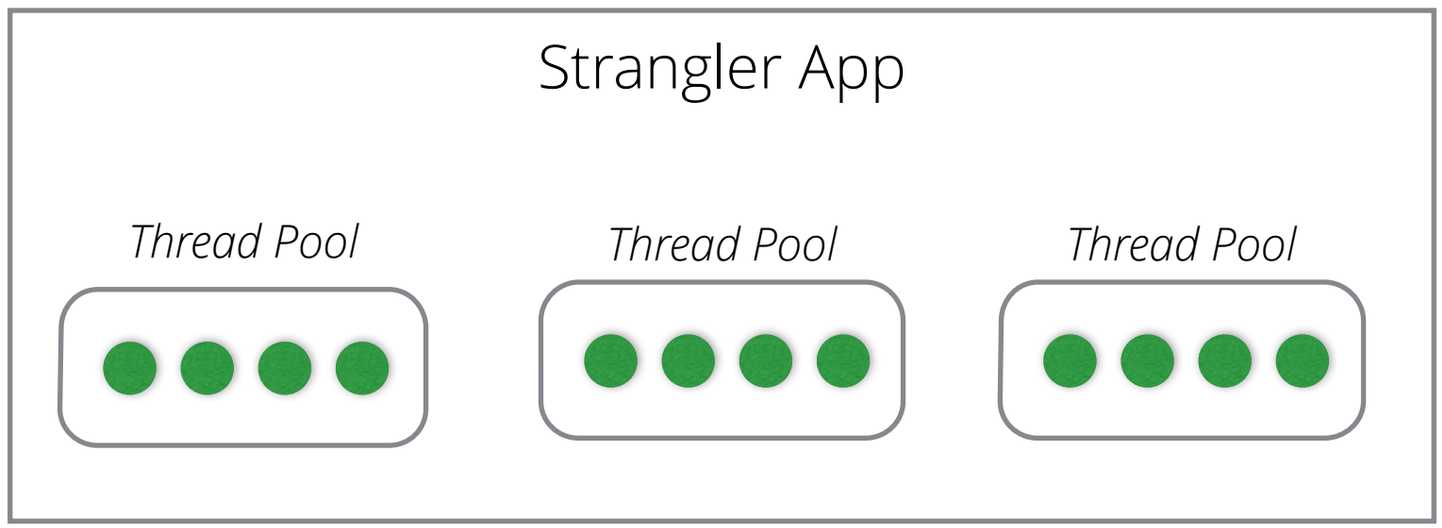
Legacy App

Legacy App

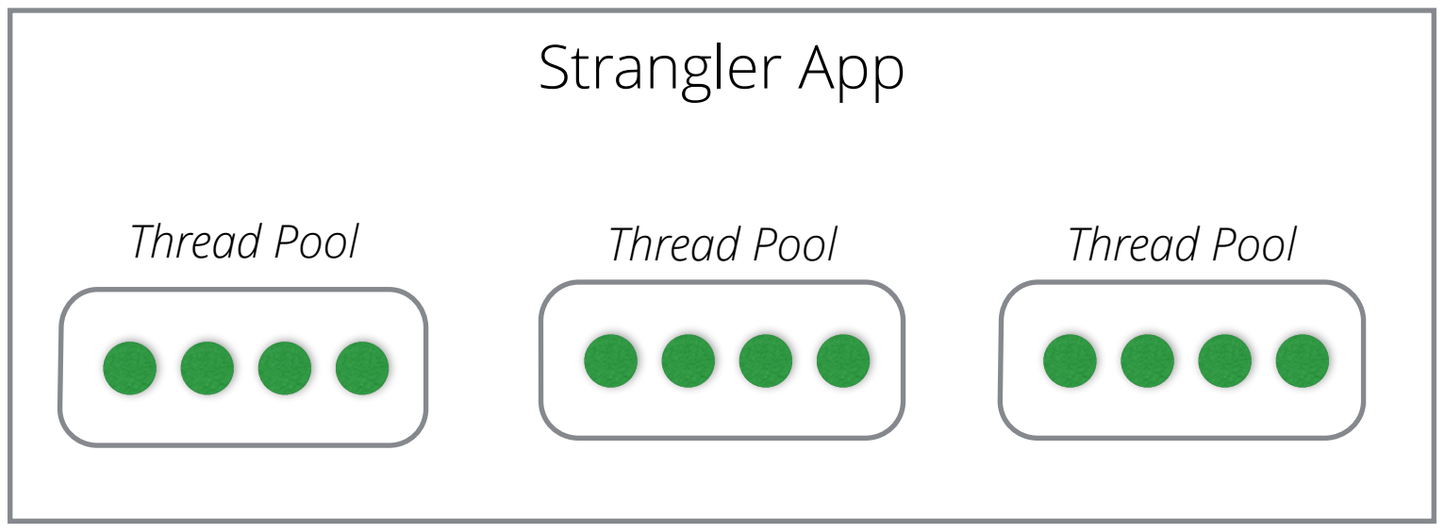
Fix **Timeouts**




Fix **Timeouts**

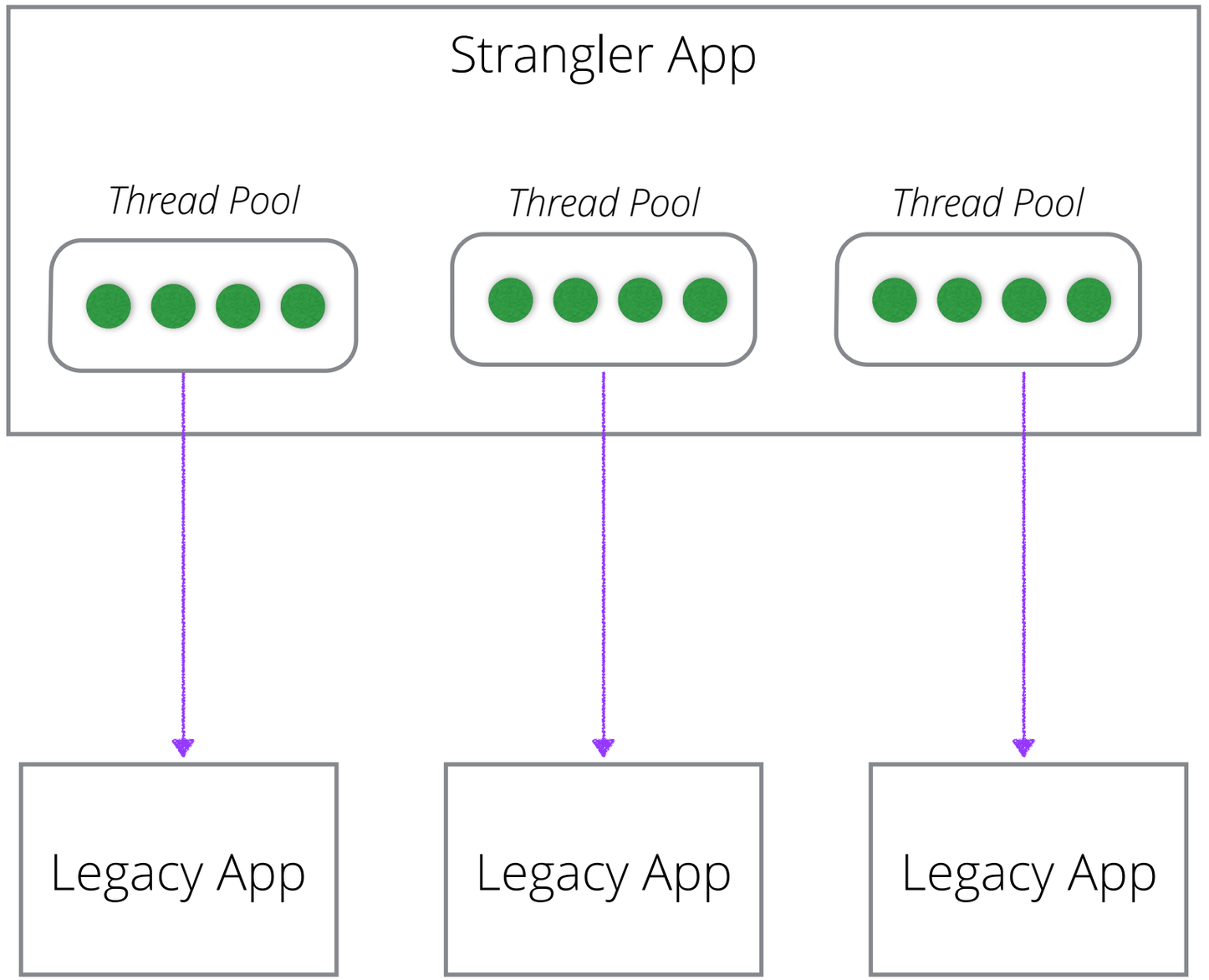



Fix **Timeouts**



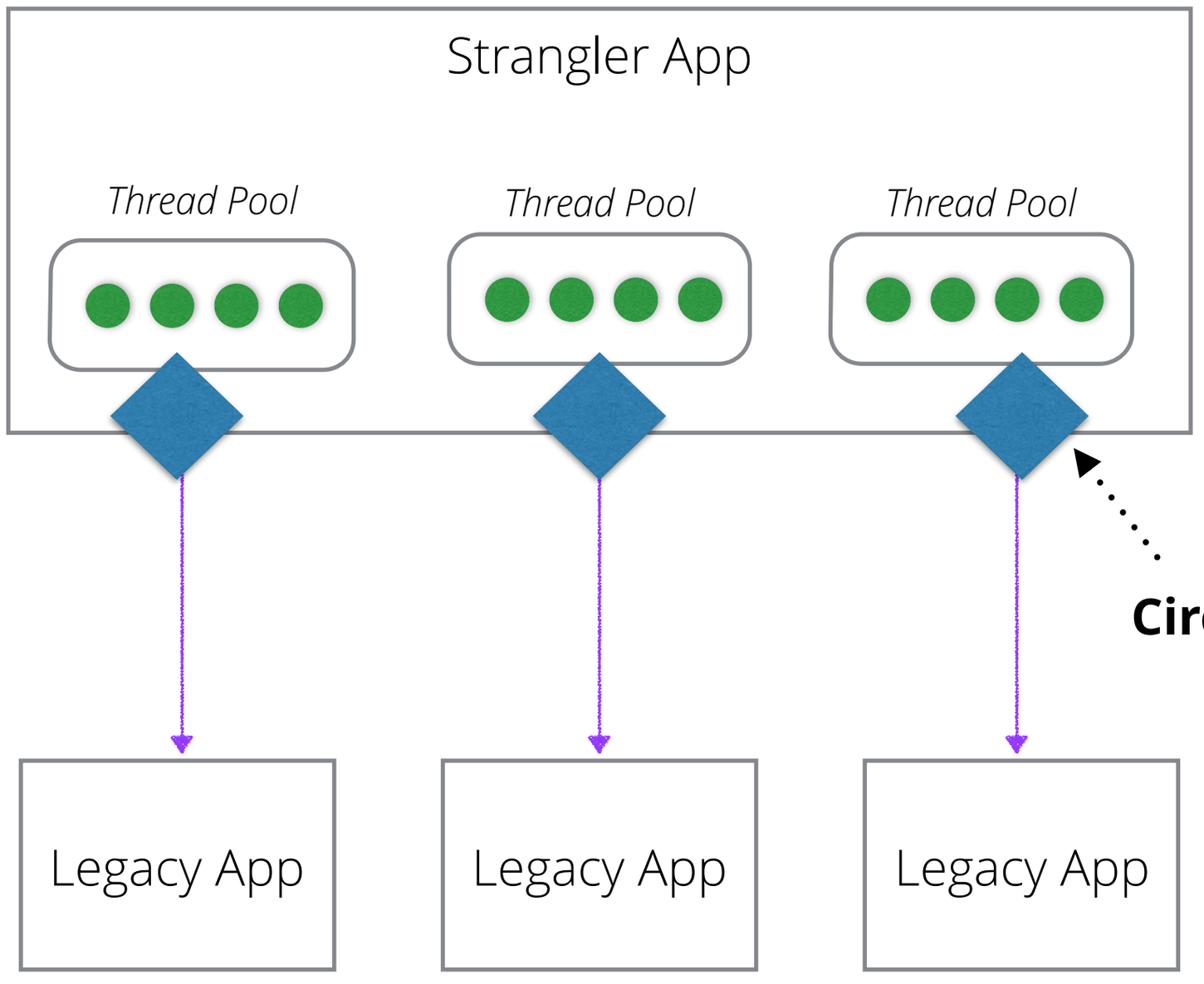

**Bulkhead**
Downstream
Connections

Fix **Timeouts**



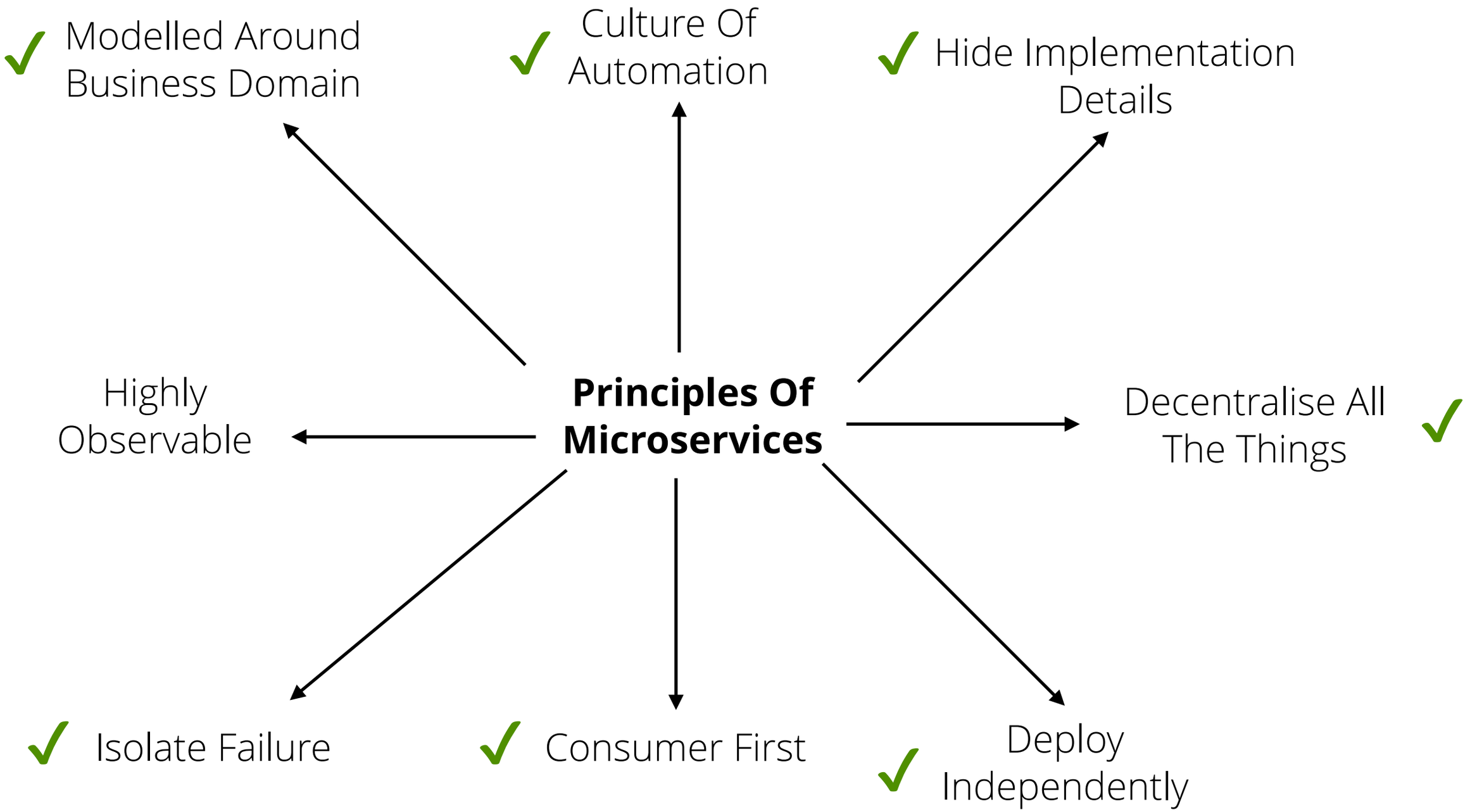

**Bulkhead**
Downstream
Connections

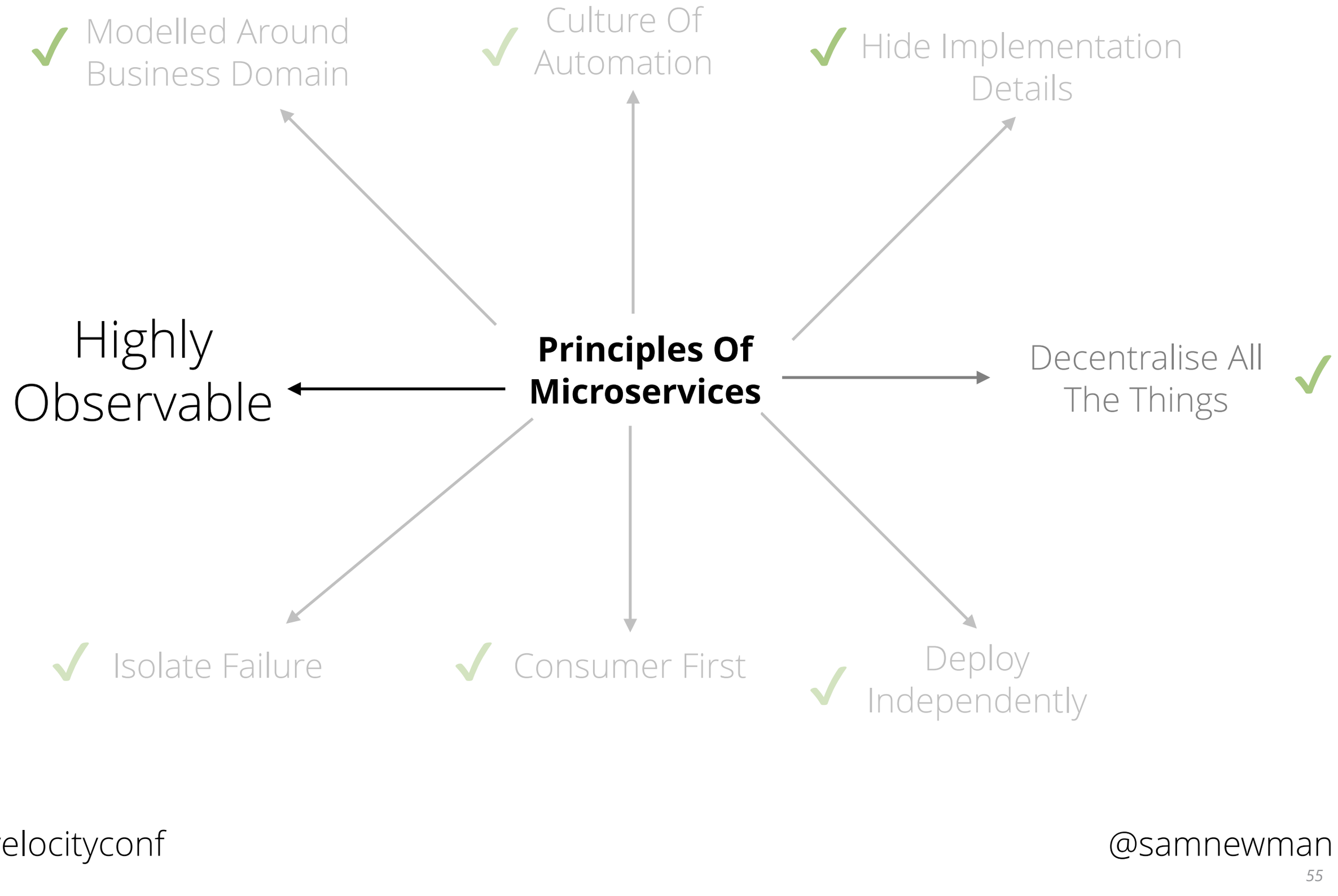
Fix **Timeouts**



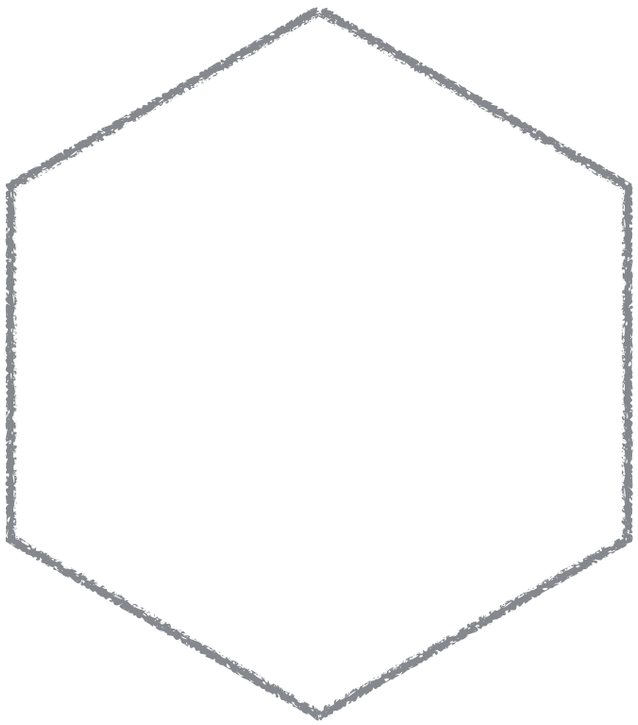

**Bulkhead**
Downstream
Connections

**Circuit Breakers**

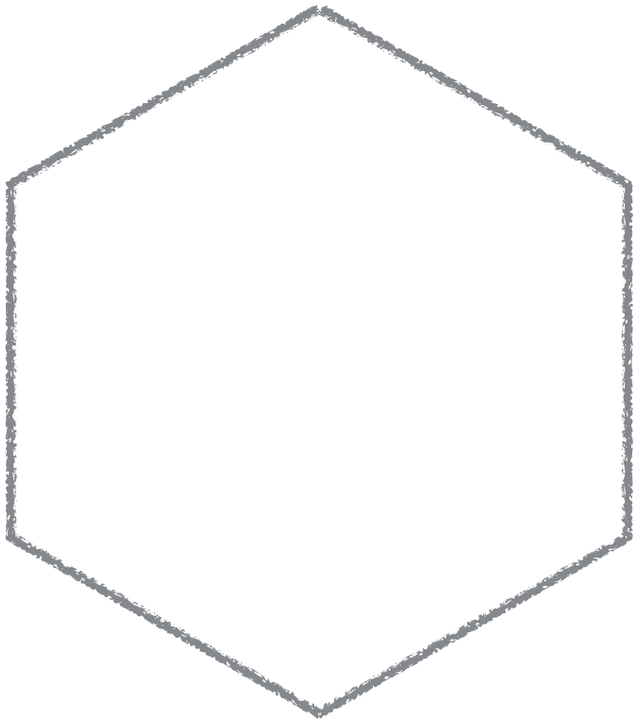

✓ Modelled Around Business Domain
✓ Culture Of Automation
✓ Hide Implementation Details
Highly Observable
Principles Of Microservices
Decentralise All The Things
✓
✓ Isolate Failure
✓ Consumer First
✓ Deploy Independently


✓ Modelled Around Business Domain
✓ Culture Of Automation
✓ Hide Implementation Details
Highly Observable
Principles Of Microservices
Decentralise All The Things
✓
✓ Isolate Failure
✓ Consumer First
✓ Deploy Independently

# STATS PAGES

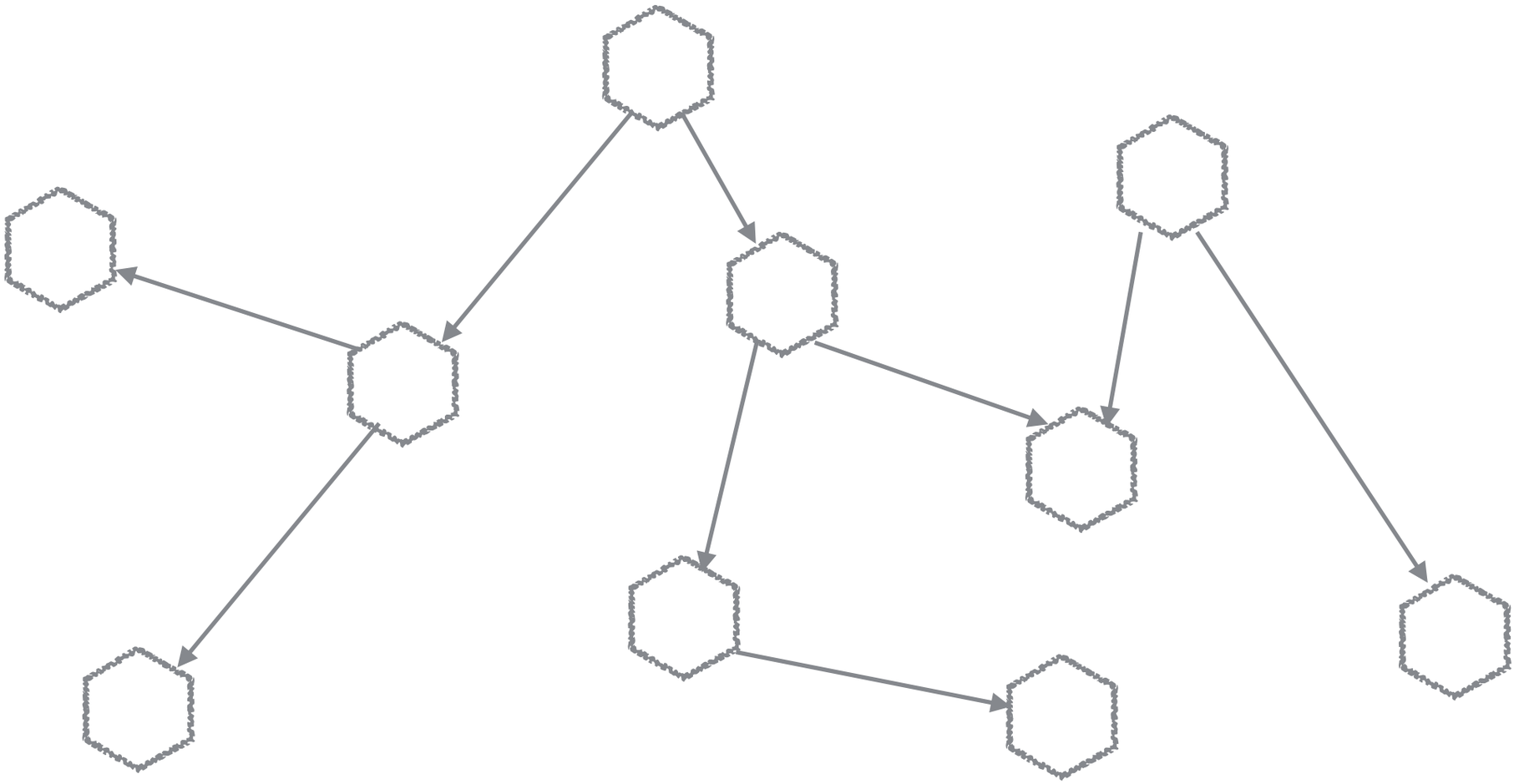


# STATS PAGES

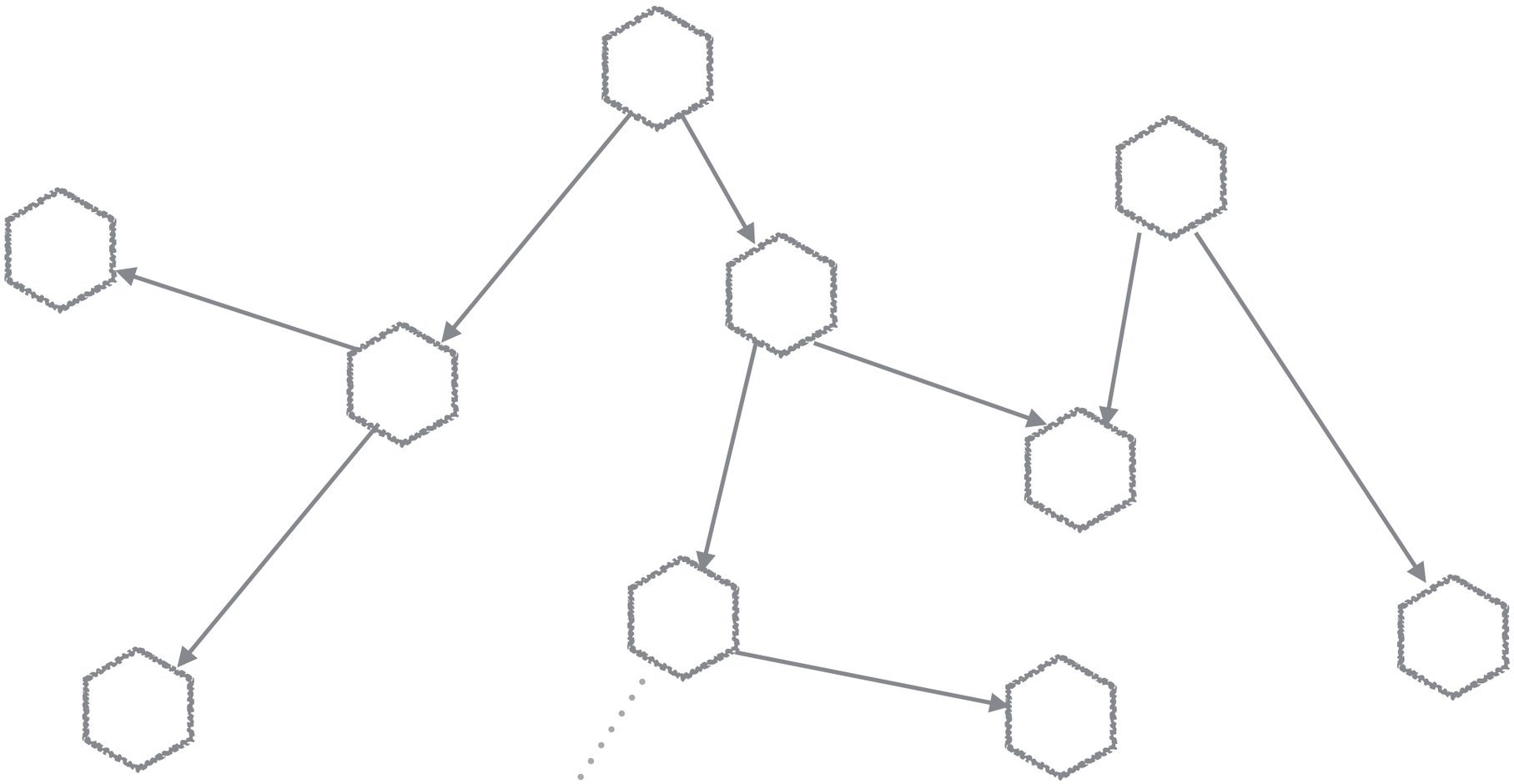



numberOfApplicationErrors
57
numberOfServicedRequestsWithResponse200
136711
numberOfServicedRequestsWithResponse304
27782
numberOfServicedRequestsWithResponse404
303
numberOfServicedRequestsWithResponse500
141
totalNumberOfServicedRequests
172383

# AGGREGATION

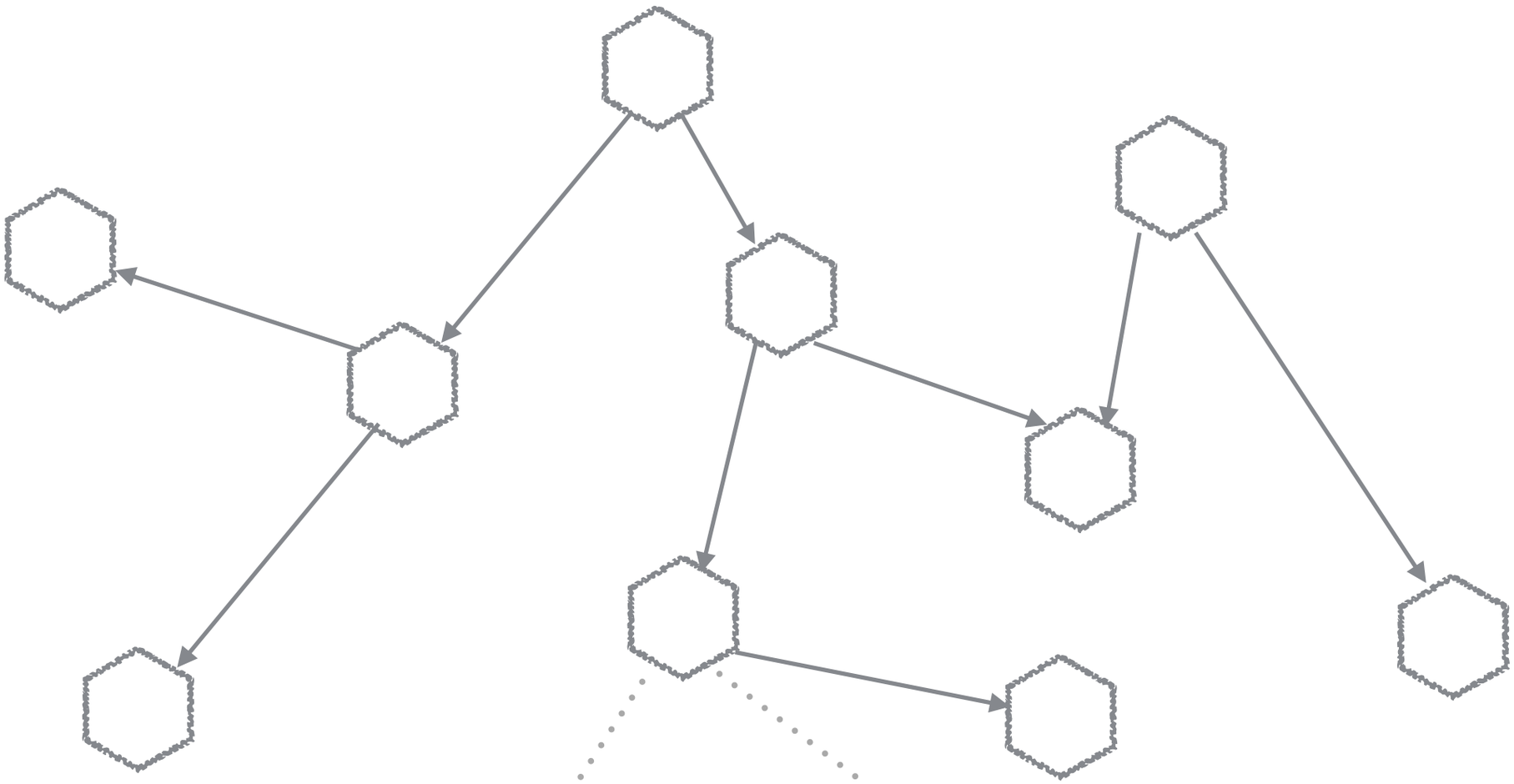


# AGGREGATION



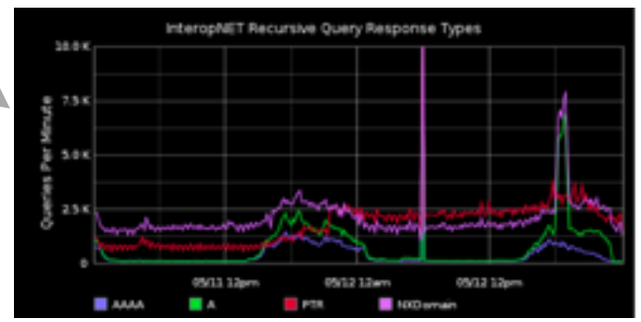
**LOGS**

# AGGREGATION

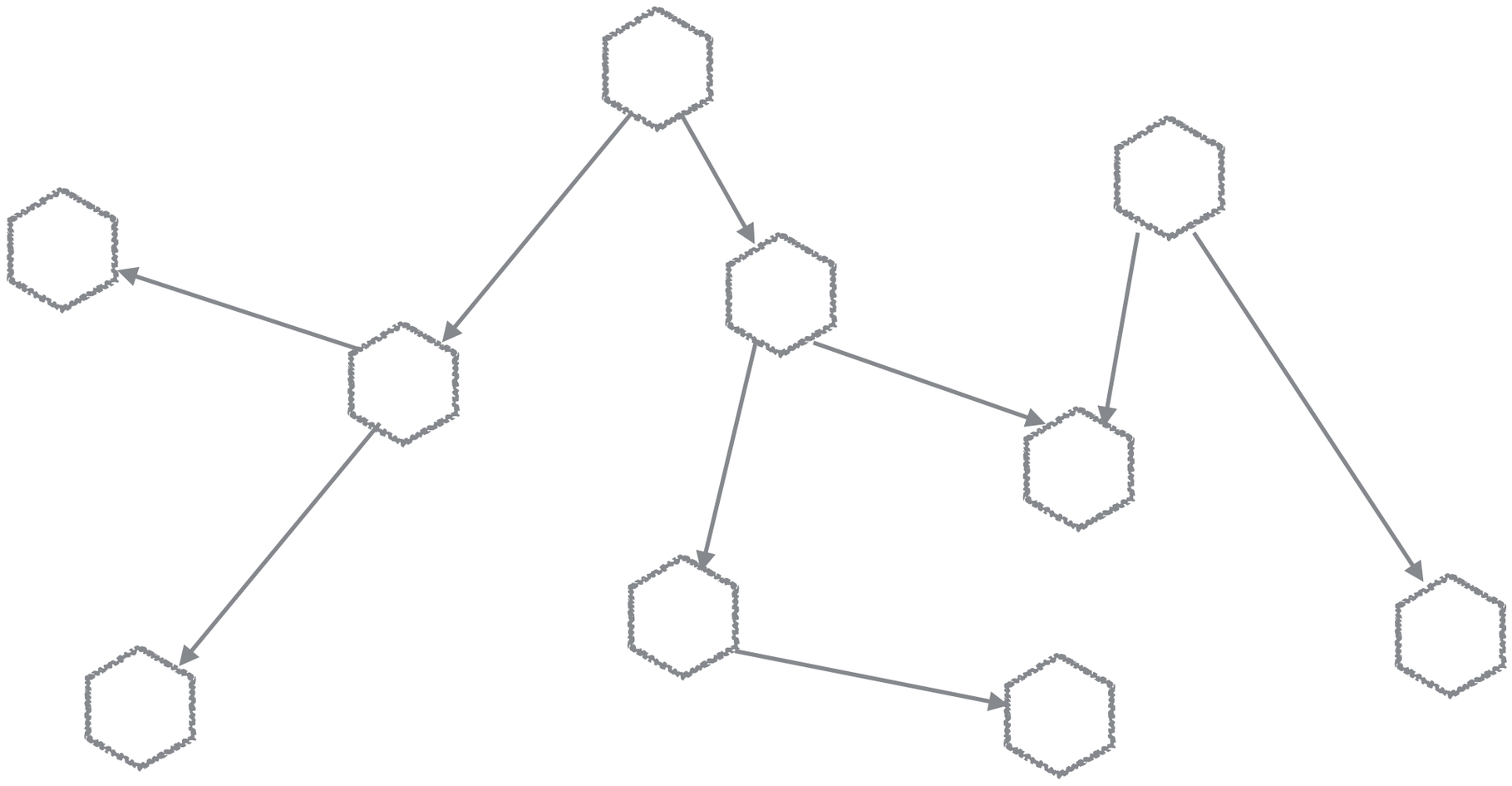


**LOGS**

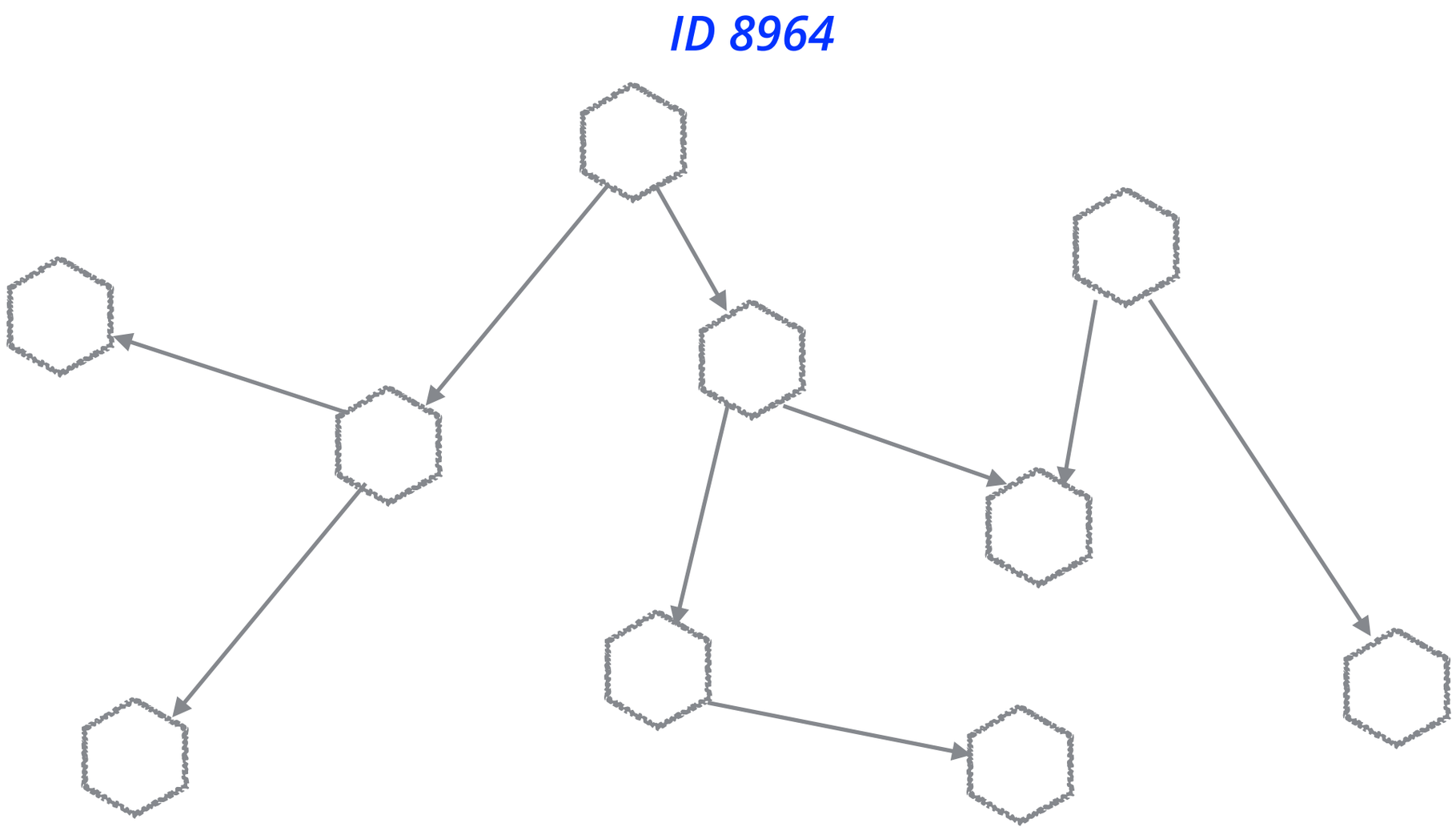

**STATS**



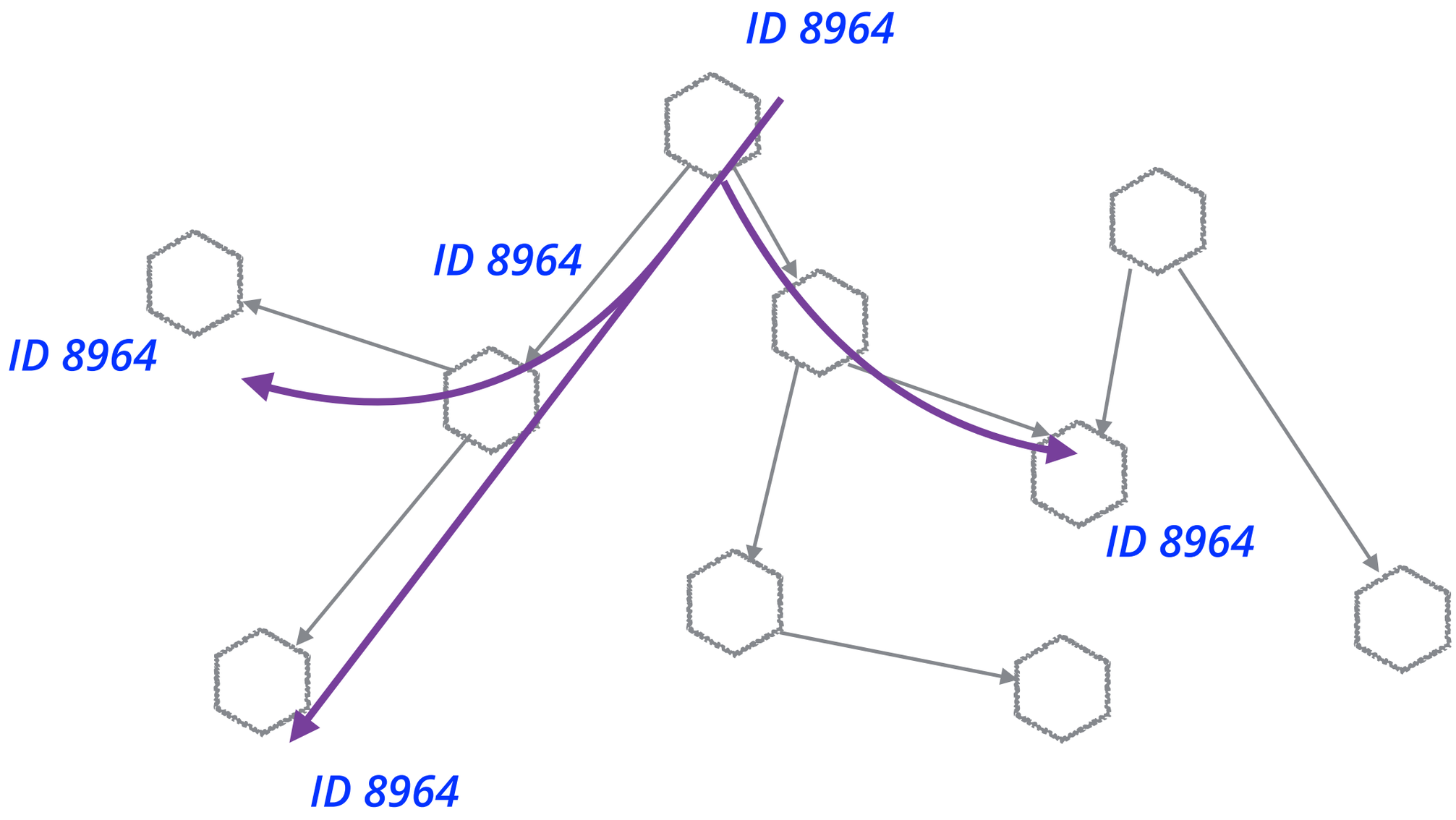
# CORRELATION IDS

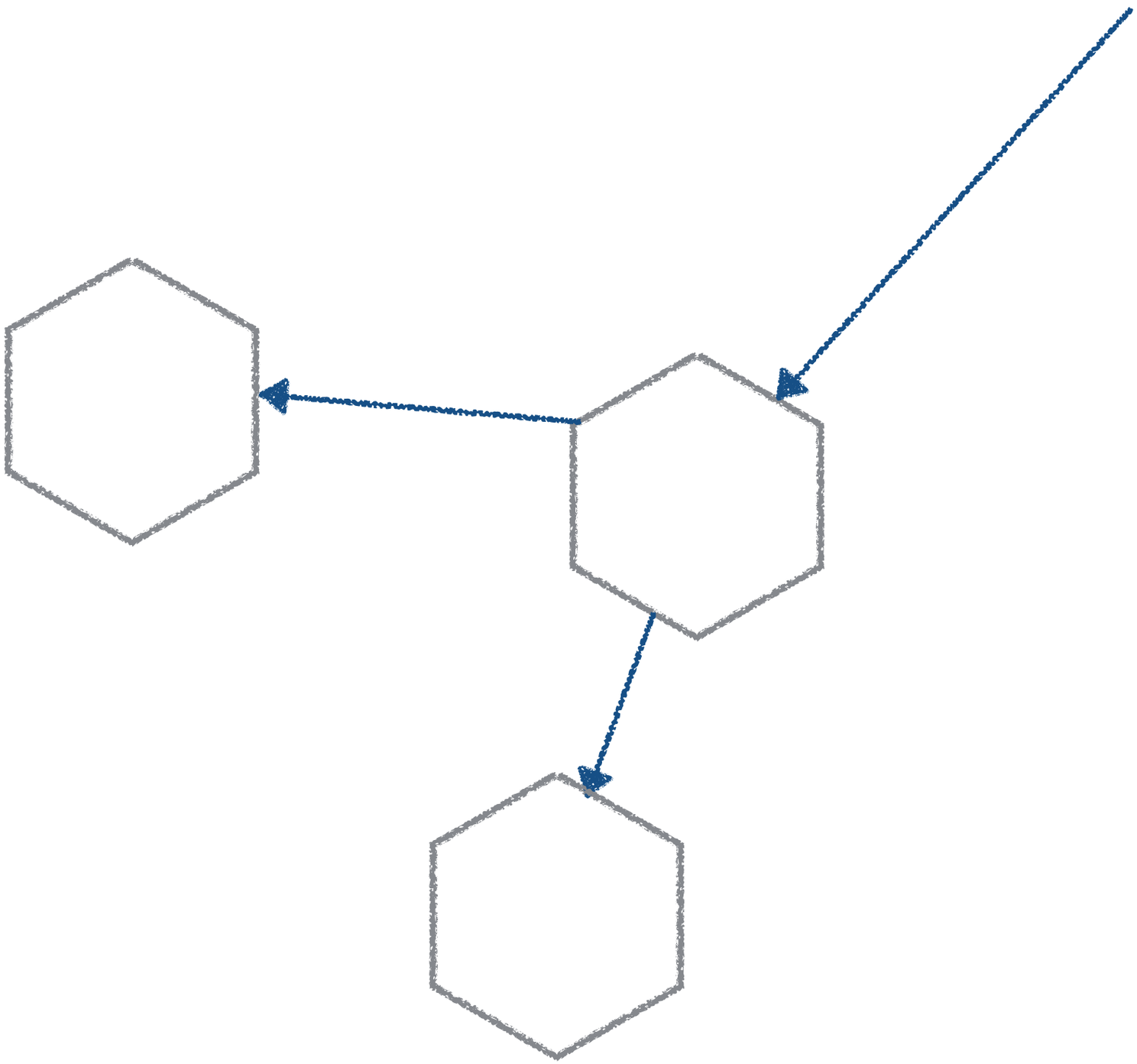


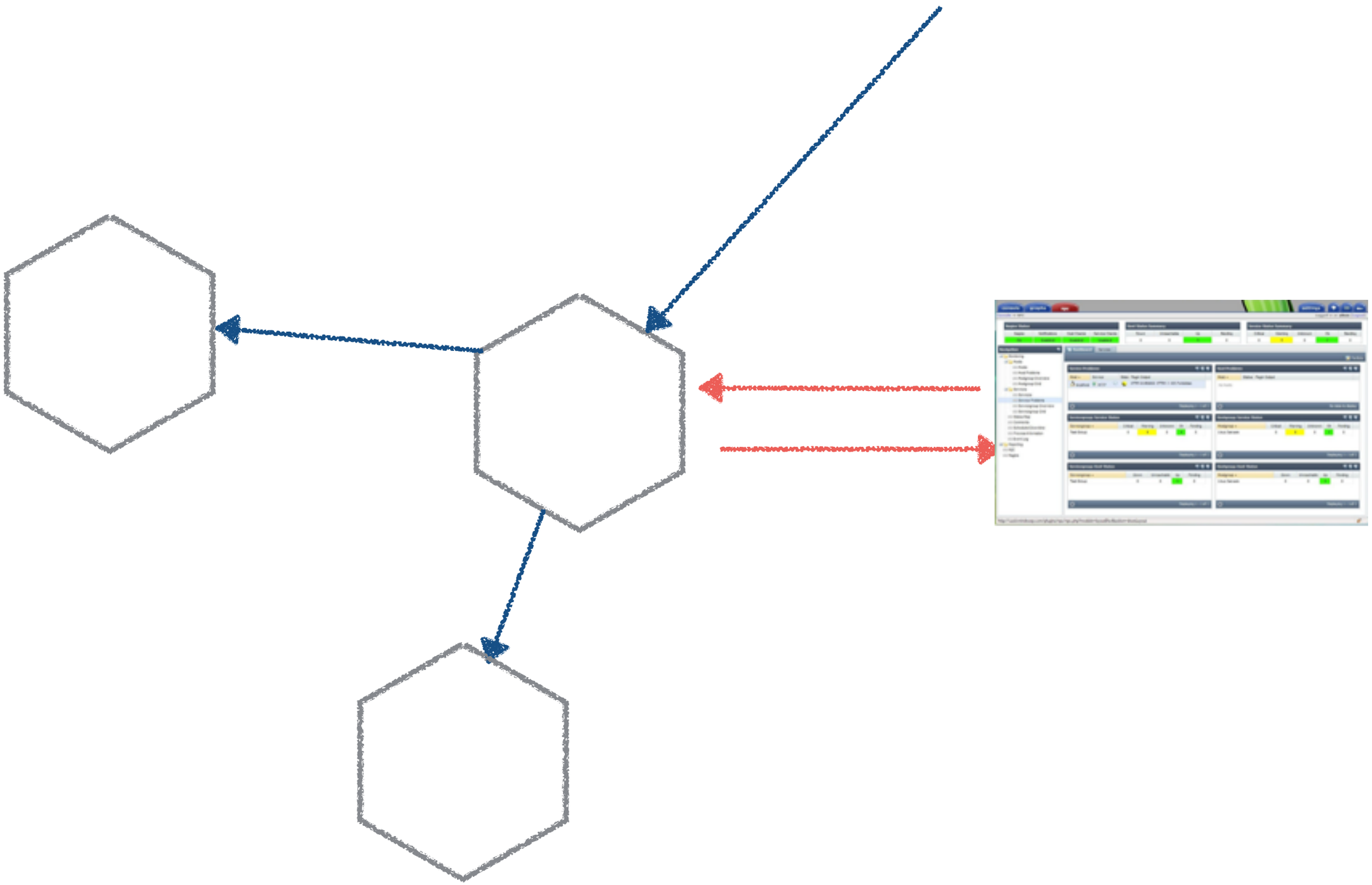
# CORRELATION IDS

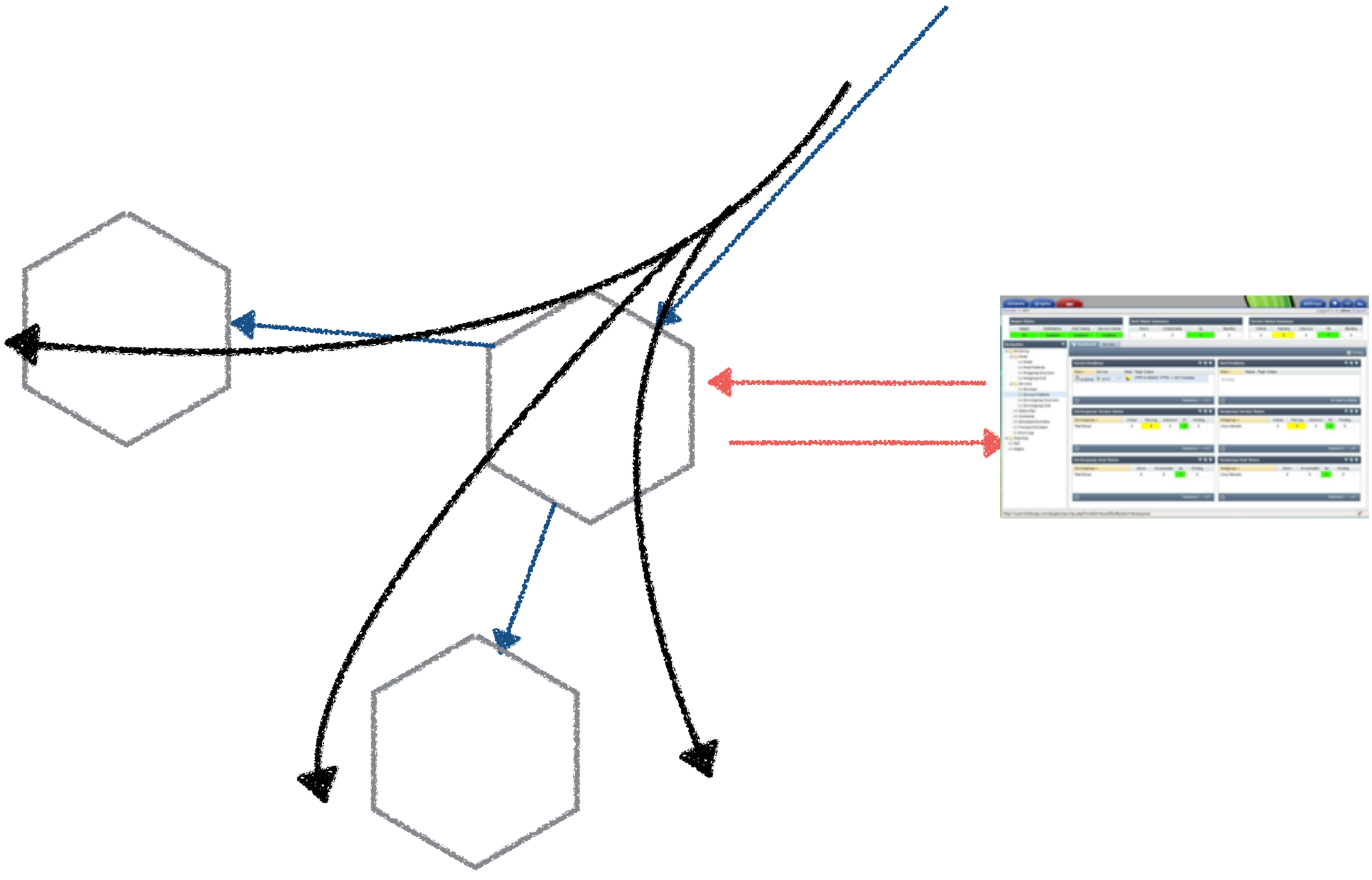


# CORRELATION IDS

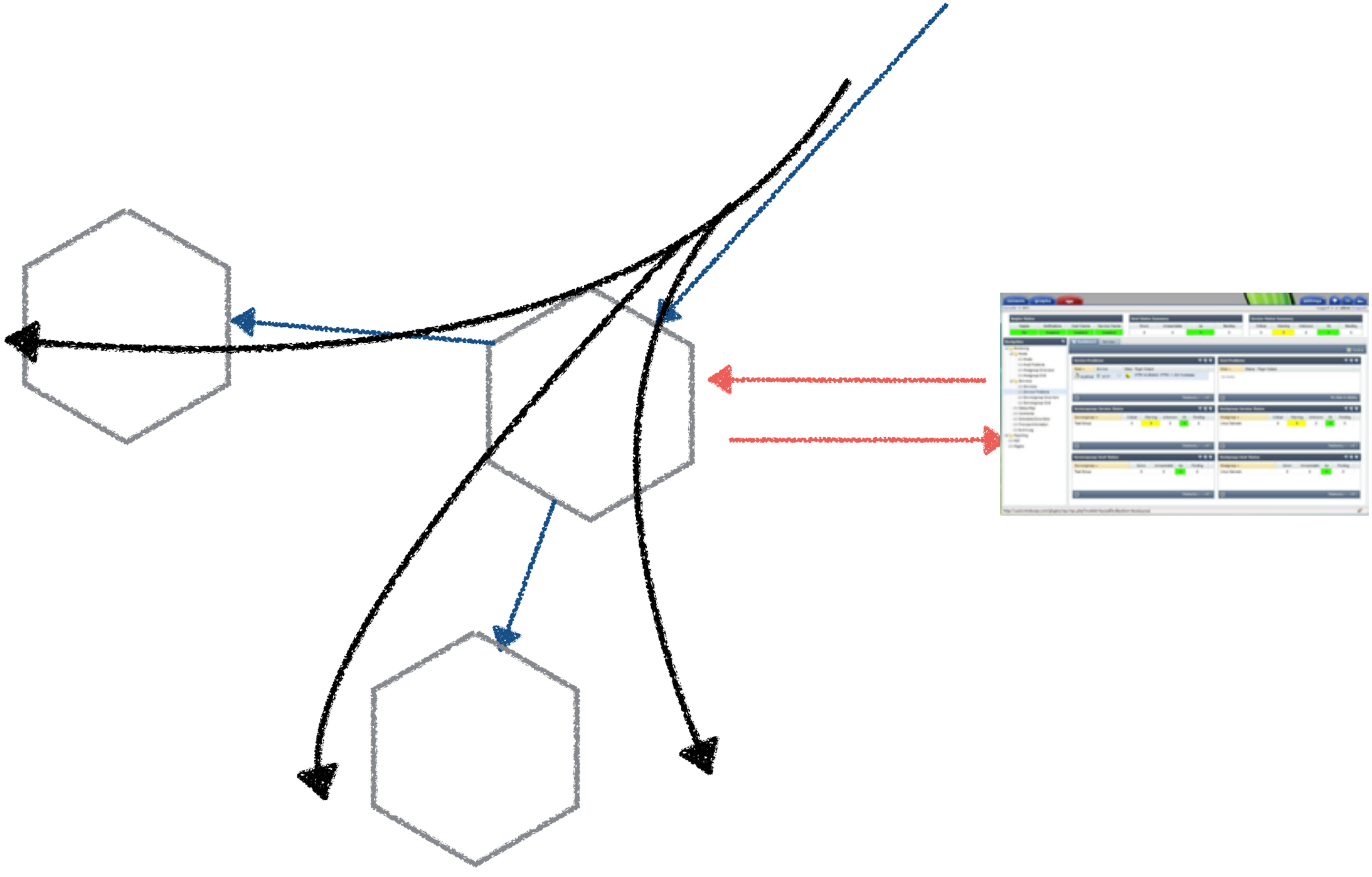








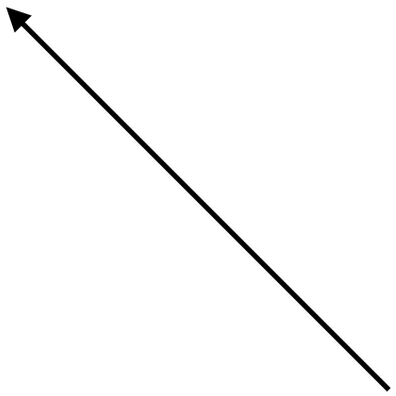
# SEMANTIC MONITORING



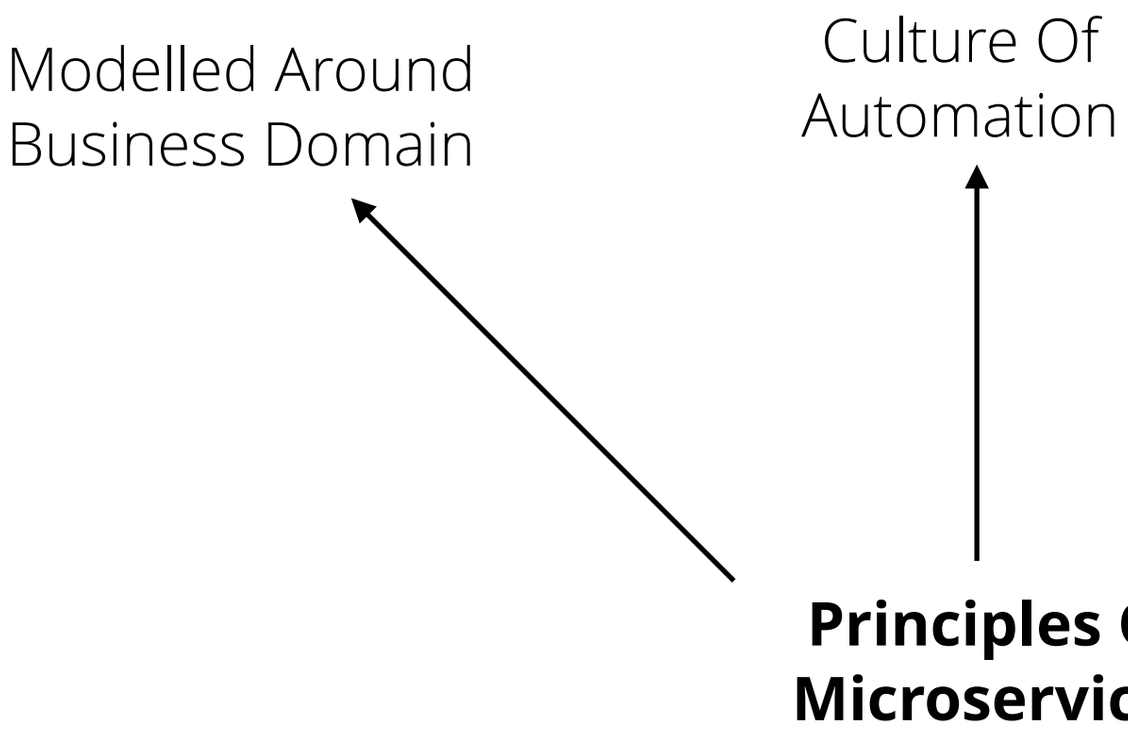

This Product Can Be Found On The Promotional End
BEKO
£249
AN EASY WAY TO PAY

# Principles Of Microservices

Modelled Around
Business Domain



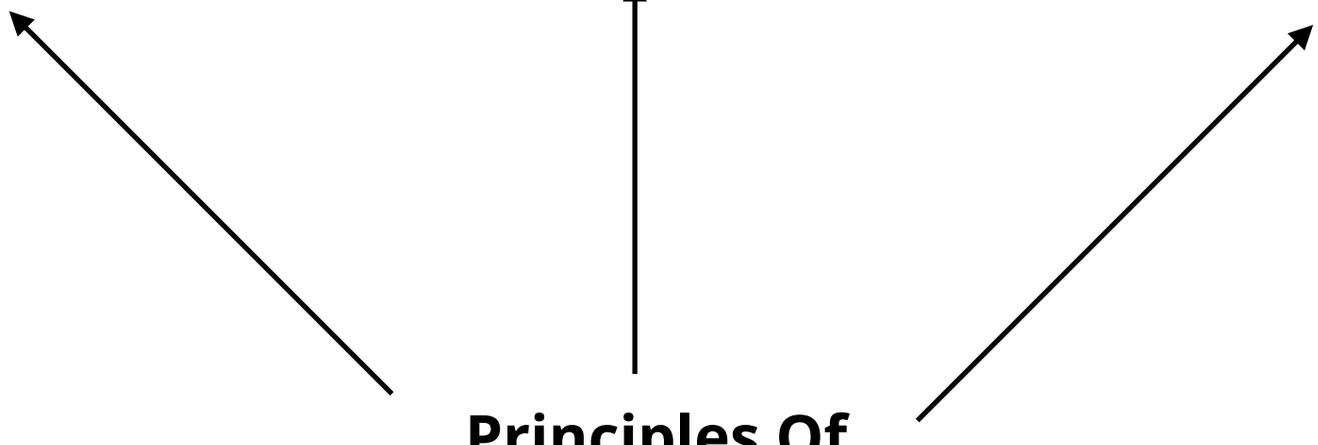
**Principles Of Microservices**


Modelled Around
Business Domain
Culture Of
Automation
Principles Of
Microservices

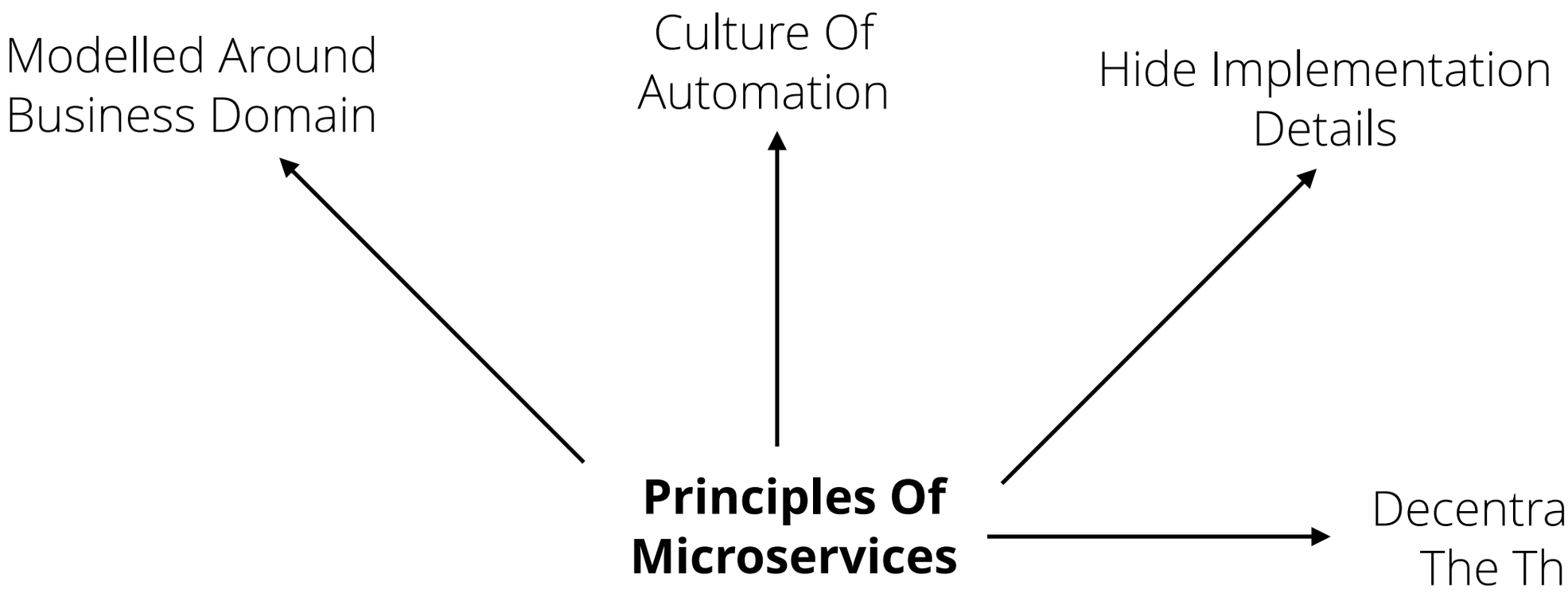
Modelled Around Business Domain

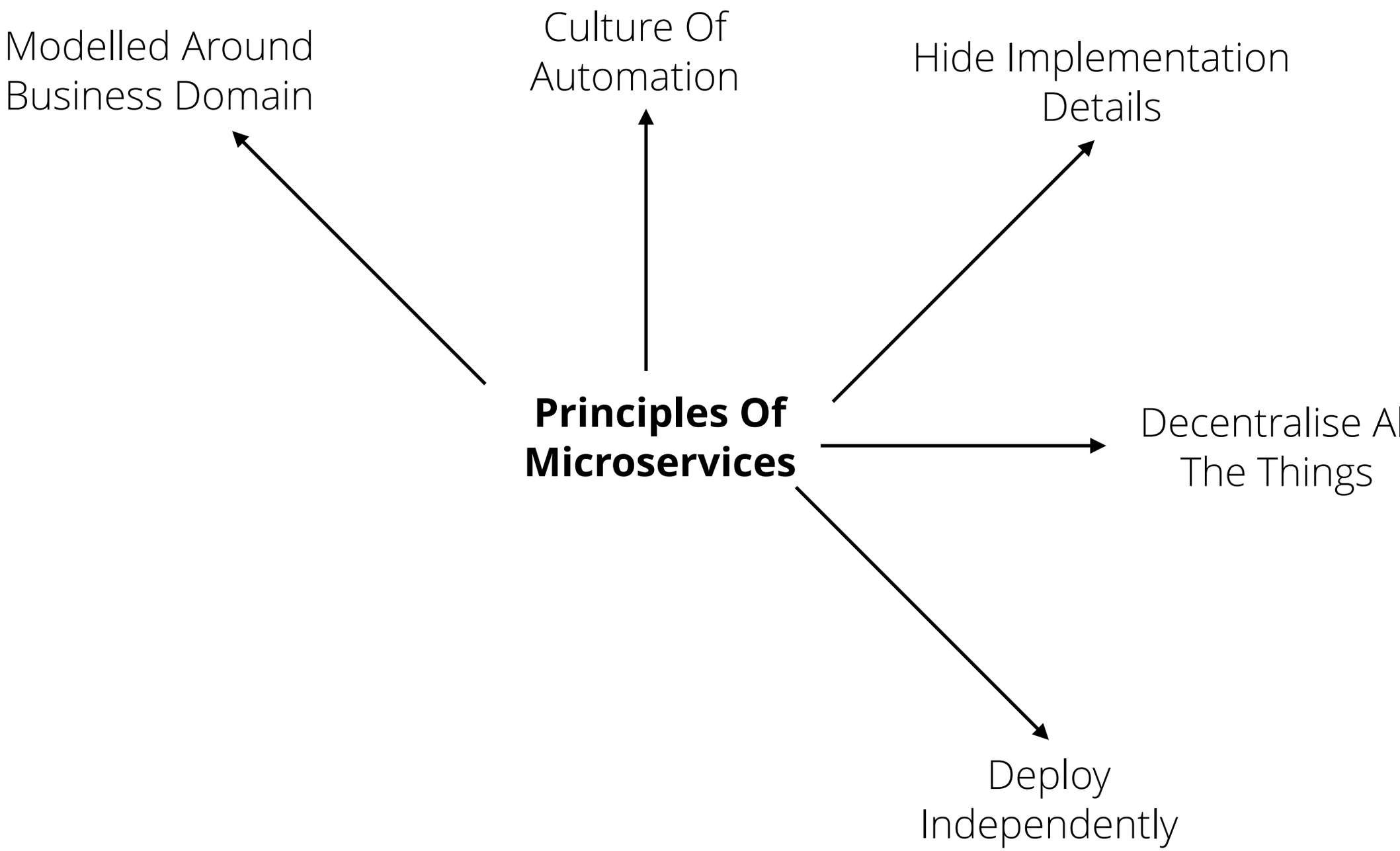
Culture Of Automation

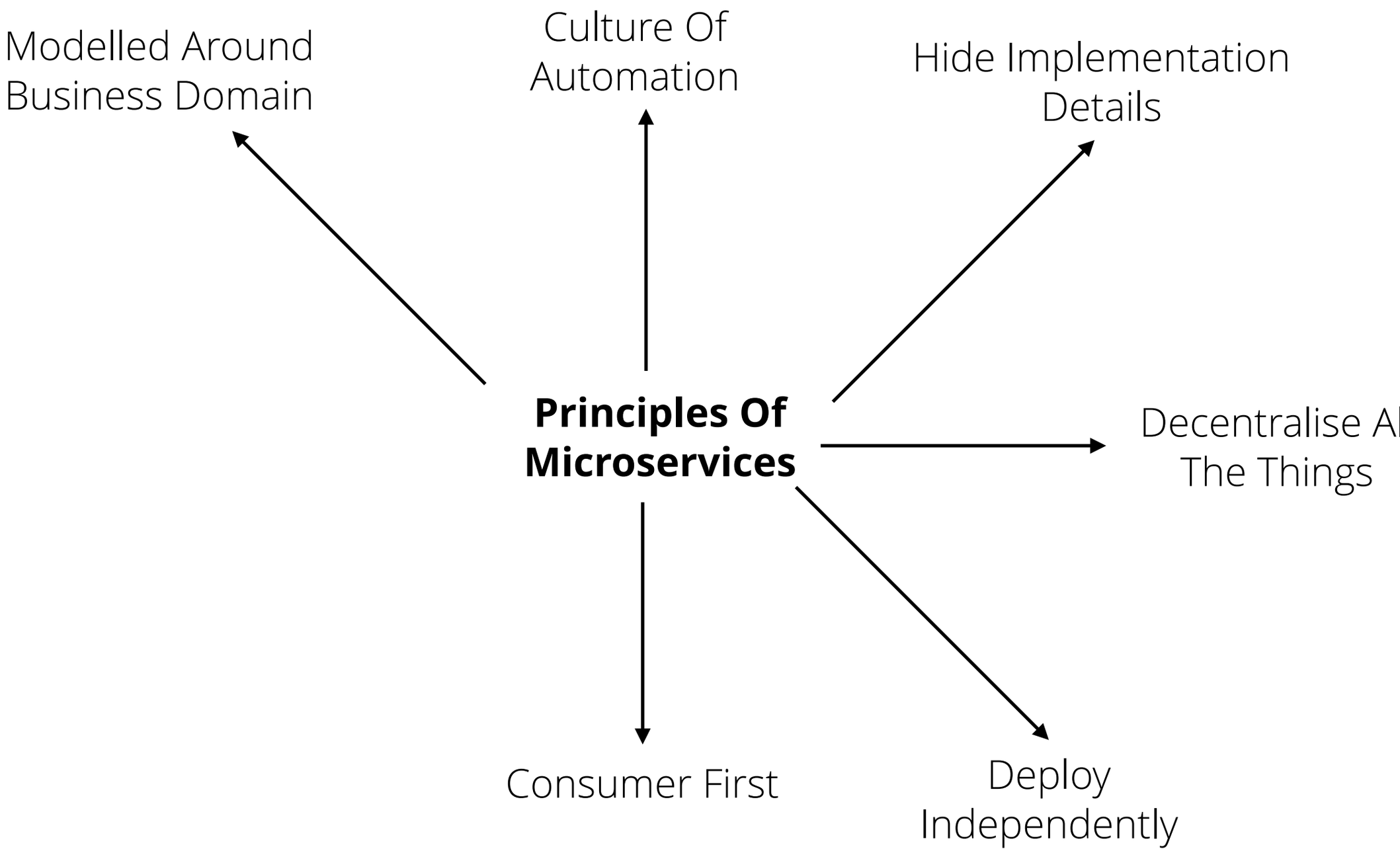
Hide Implementation Details

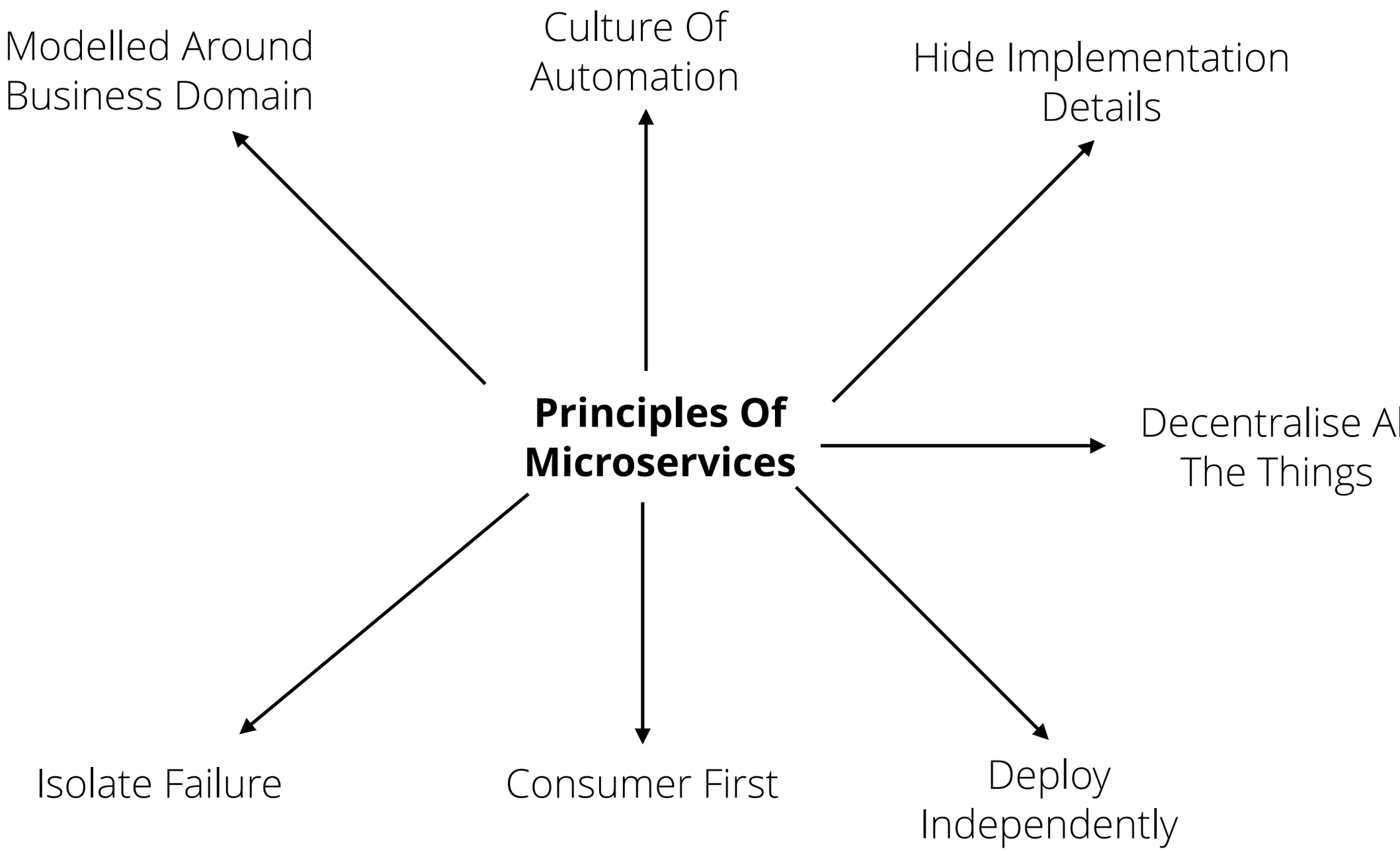


**Principles Of Microservices**


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things
Deploy Independently


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things
Consumer First
Deploy Independently


Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Principles Of Microservices
Decentralise All The Things
Isolate Failure
Consumer First
Deploy Independently

Modelled Around Business Domain

Culture Of Automation

Hide Implementation Details

Highly Observable

**Principles Of Microservices**

Decentralise All The Things

Isolate Failure

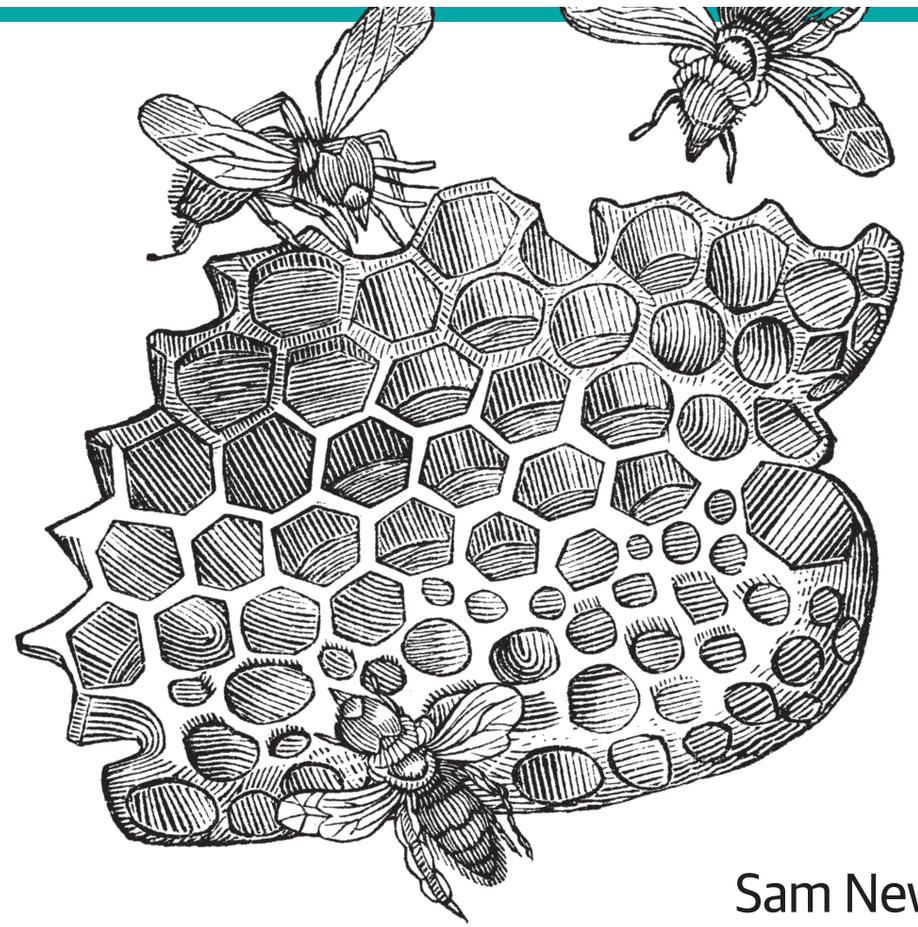
Consumer First

Deploy Independently

O'REILLY®

# Building Microservices

http://buildingmicroservices.com/



Sam Newman

@velocityconf

@samnewman

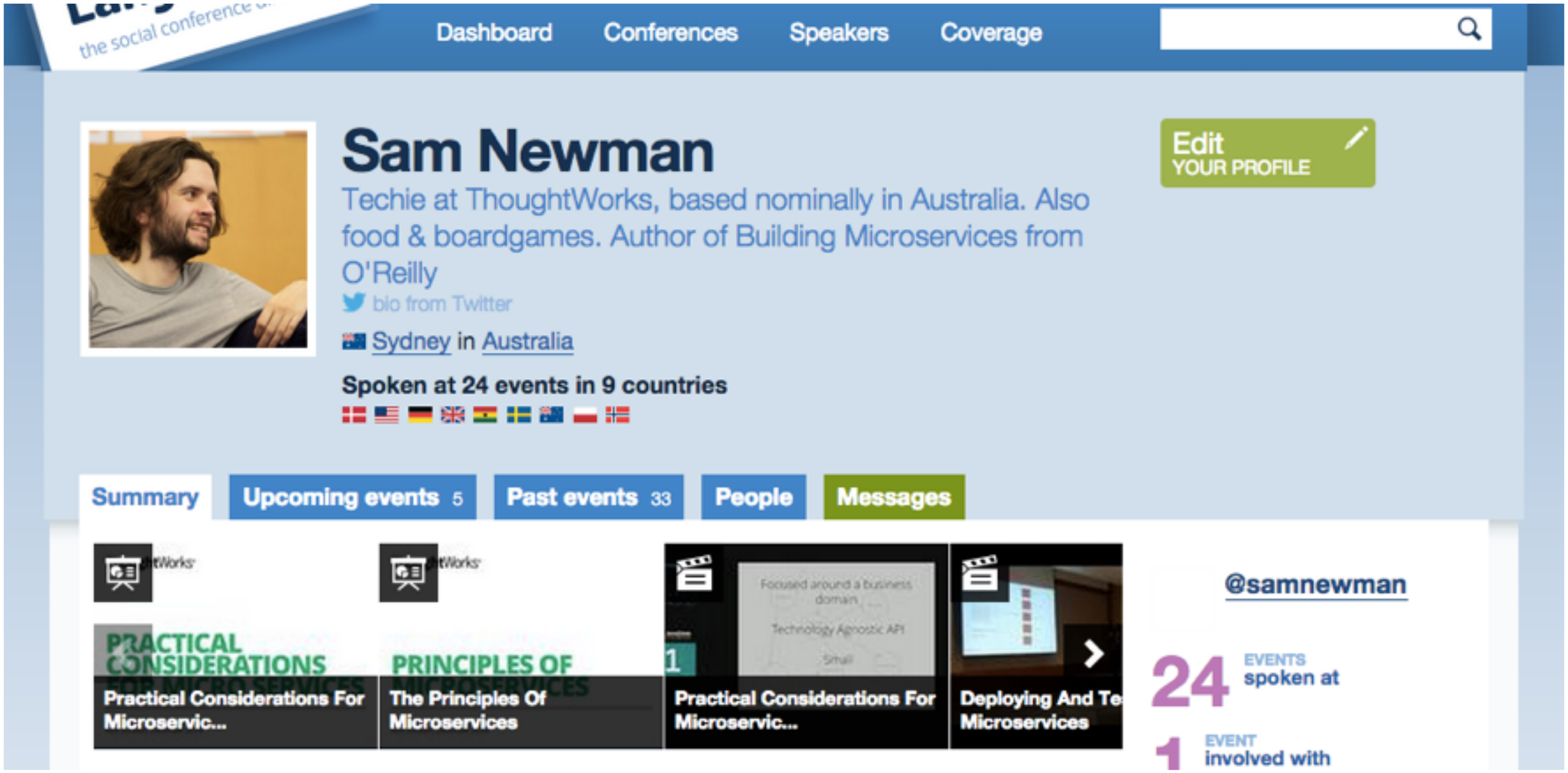
# Office Hours

# Office Hours
# (well, 30mins)

# Office Hours

(well, 30mins)

12.30pm on Thursday




http://lanyrd.com/profile/samnewman/

# THANKS!

*Sam Newman*
*@samnewman*

ThoughtWorks®