Migrating a monolith to microservices? A dark energy, dark matter perspective

Chris Richardson

Founder of Eventuate.io

Founder of the original CloudFoundry.com

Author of POJOs in Action and Microservices Patterns

@crichardson

chris@chrisrichardson.net

adopt.microservices.io

Copyright © 2022. Chris Richardson Consulting, Inc. All rights reserved

Thriving in today's world

Volatile
Uncertain
Complex
Ambiguous

Businesses must be nimble, agile and innovate faster

Software is eating the world

Responsible for

Business critical application

You **must** deliver software rapidly, frequently, reliably and sustainably

Measured by DORA metrics

Goal

Your reality

Software delivery performance metric	High	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per month and once every 6 months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Between one day and one week	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one day	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	46%-60%

+ your monolith's technology stack is out of date

Adopting the microservice architecture will make everything wonderful, right?

Hint: it won't

Presentation goal

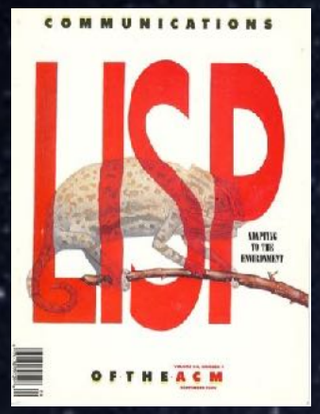
Using the

dark energy and dark matter forces

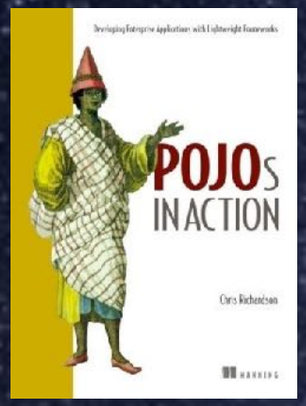
to decide whether to refactor a

monolith to microservices

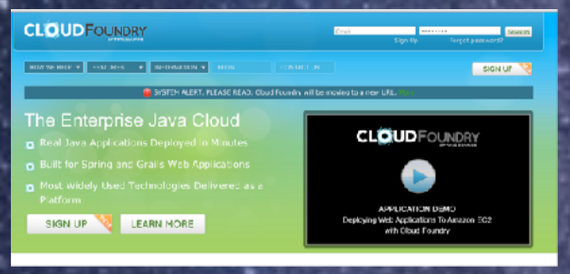
About Chris



Late 80s



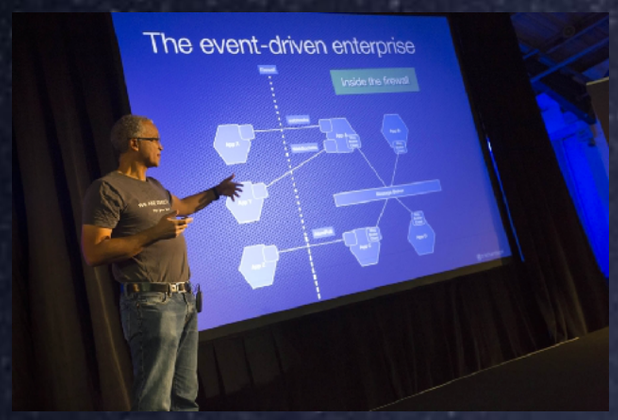
2006



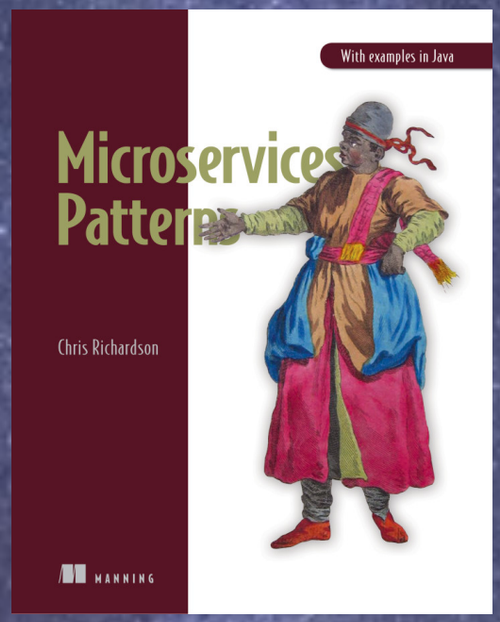
2008

2009

2012-



eventuate



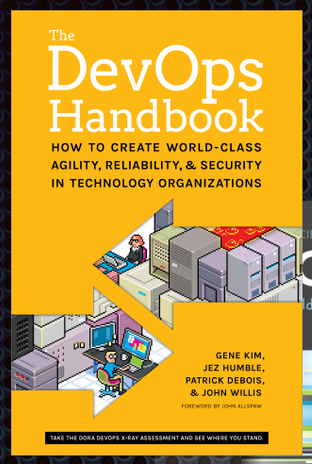
http://adopt.microservices.io

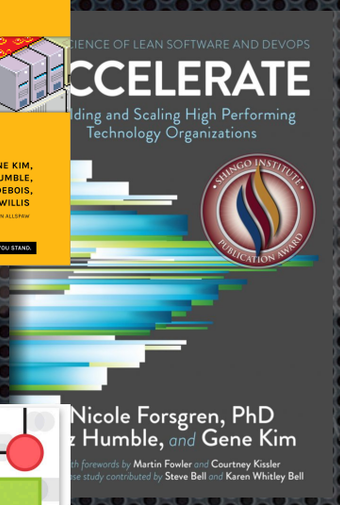
Agenda

- Architecting for modern software delivery
- Dark energy and dark matter: forces that drive the architecture
- Dark energy: encouraging decomposition
- Dark matter: resisting decomposition

The success triangle

Process: DevOps/Continuous Delivery & Deployment





S/W VUCA

Businesses must be nimble, agile and innovate faster

IT must deliver software rapidly, frequently, reliably and sustainably.

Measured by the DORA metrics

Supports

Organization:
Network of small,
loosely coupled, product teams

Supports

Architecture:
???

@crichardson

Required architectural quality attributes (.a.k.a. -ilities)

DevOps

Autonomous Teams

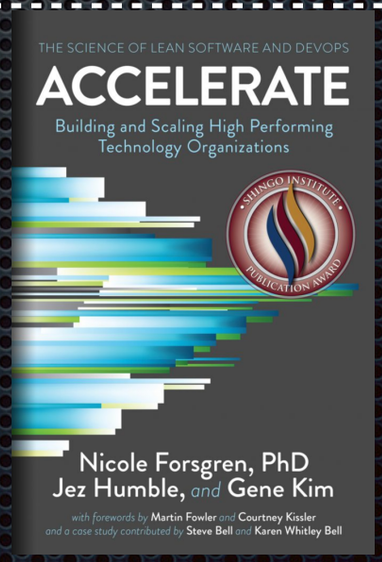
Long-lived applications

Testability

Deployability

Loose coupling

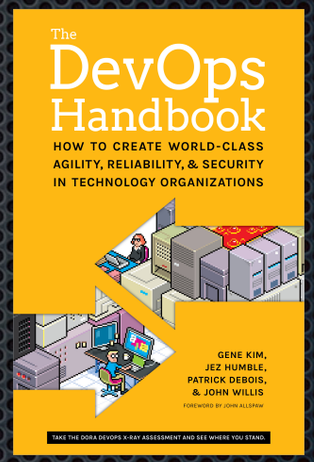
Evolvability



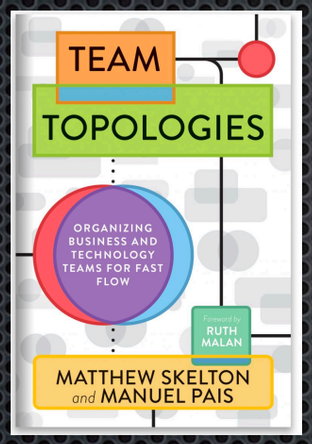
Enable incremental upgrades of technology stack

Make the most of the monolith

Process: **adopt DevOps and automate**



Success triangle



Organization:
Restructure and increase autonomy

Monolithic architecture:
Modularize and modernize

If and only if that is insufficient* **then** consider migrating to microservices

*Large, complex applications developed by a (usually) large team that need to be delivered rapidly, frequently, and reliably

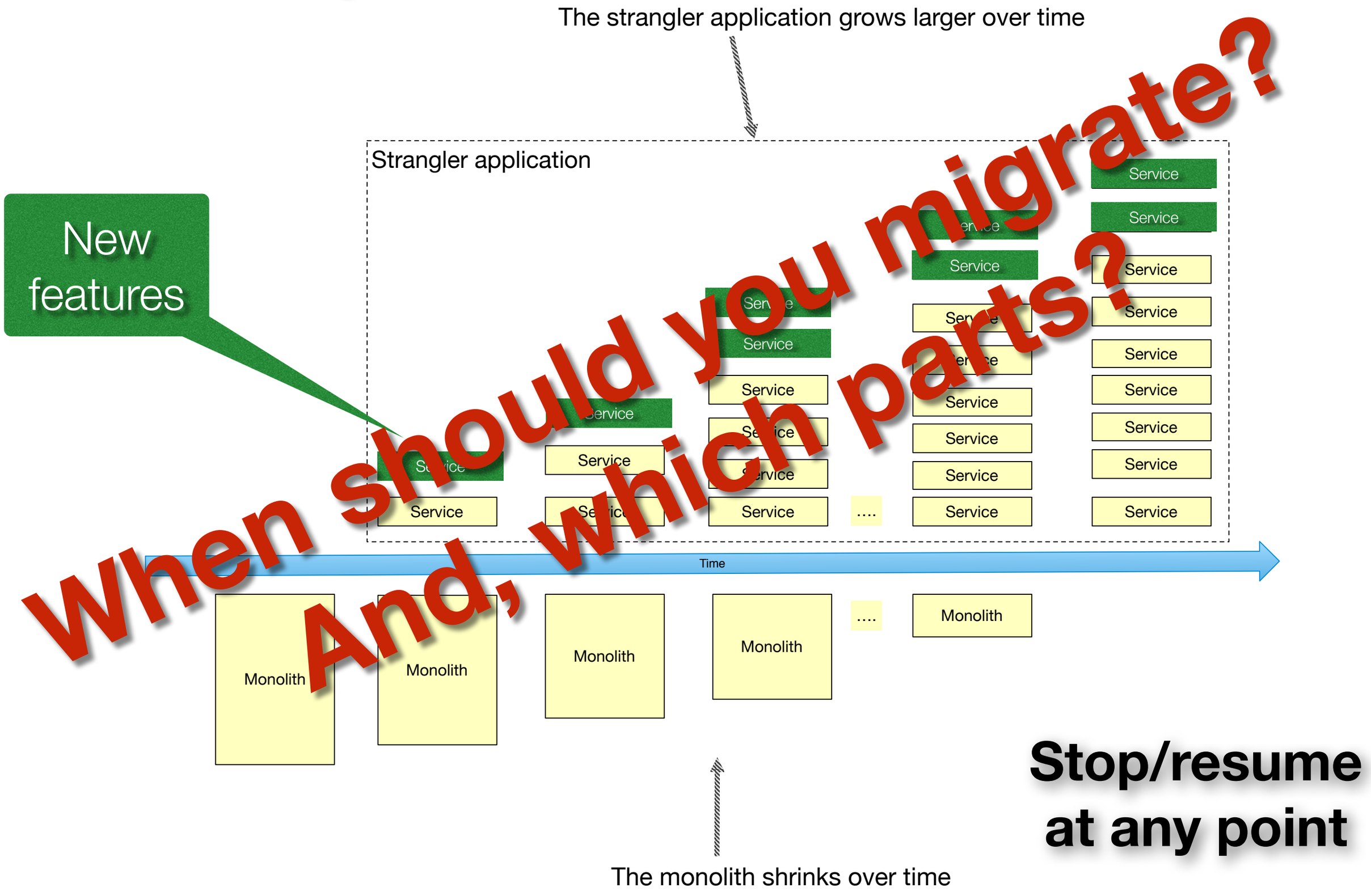
On the other hand:

Improving the monolith can take a while

THEREFORE

Implement urgent new features as services now

Strangler Fig Pattern: Incrementally refactoring a monolith to services



Agenda

- Architecting for modern software delivery
- Dark energy and dark matter: forces that drive the architecture
- Dark energy: encouraging decomposition
- Dark matter: resisting decomposition

How to define an Architecture…

Requirements

Functional requirements

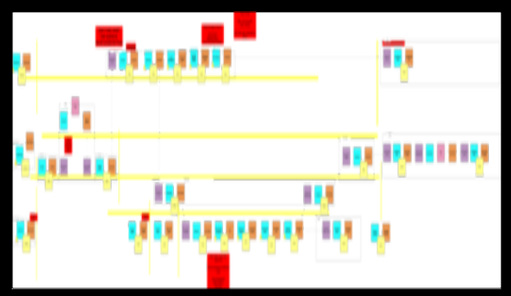
As a consumer
I want to place an Order
So that I can ….

As a Restaurant
I want to accept an Order
So that I can ….

System quality attributes

- SLA: Reliability/Latency
- Scalability
- …

Wireframe/UI mockups



Distill

The “requests” that the application implements

Have SLOs

About Subdomains

- Business capability/function/etc
- Logical view: packages and classes
- Team-sized
- Loosely coupled (Conways law)

Application

System operations

createCustomer()

createOrder()

findOrder()

findOrderHistory()

1

«subdomain»
Customer management

«aggregate»
Customer

2

«subdomain»
Order management

«aggregate»
Order

Customer Team

Order Team

@crichardson

… how to define an Architecture

createOrder()

Monolith or Microservices

System operations

createOrder()

<<subdomain>>
Order Management

Order

Group subdomains into services

3

Design collaborations for distributed operations

Collaboration

Order Service

<<subdomain>>
Order
Management

Delivery Service

<<subdomain>>
Delivery
Management

<<subdomain>>
Courier
Management

Kitchen Service

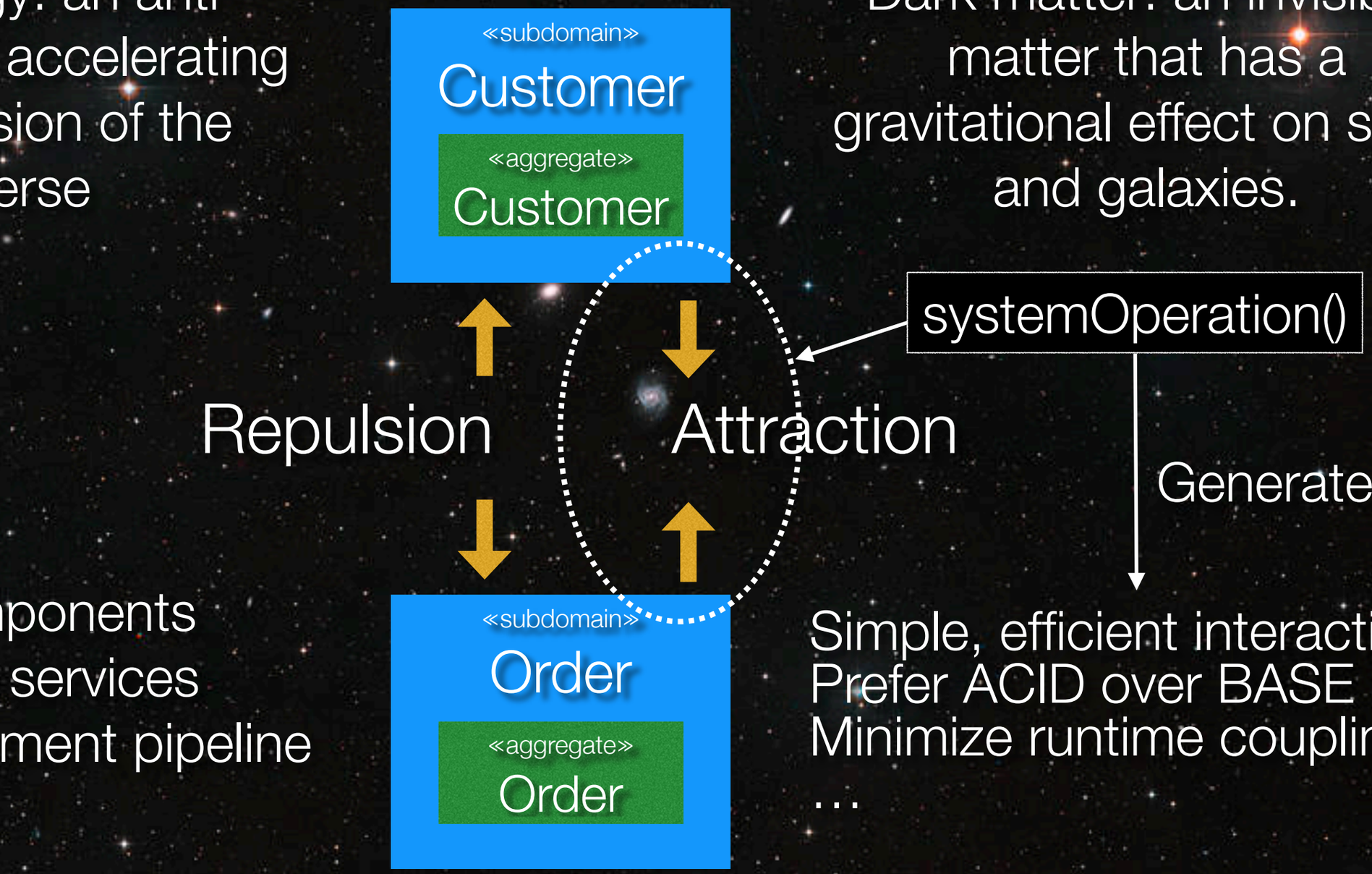
<<subdomain>>
Kitchen
Management

Application

Grouping subdomains into components: together or separate?

Dark energy: an anti-gravity that's accelerating the expansion of the universe

Dark matter: an invisible matter that has a gravitational effect on stars and galaxies.



Simple components
Team-sized services
Fast deployment pipeline
…

Simple, efficient interactions
Prefer ACID over BASE
Minimize runtime coupling
…

https://www.nasa.gov/feature/goddard/2020/new-hubble-data-explains-missing-dark-matter

https://chrisrichardson.net/post/microservices/2021/04/15/mucon-2021-dark-energy-dark-matter.html

Together or separate = Module vs. Service?

createOrder()

FTGO Monolith

- Order Management
- Delivery Management
- … Management

vs

Dark Matter

Cost & Feasibility

createOrder()

Collaboration

FTGO Monolith

- Order Management
- … Management

Dark Energy

Delivery Service

Benefits

Agenda

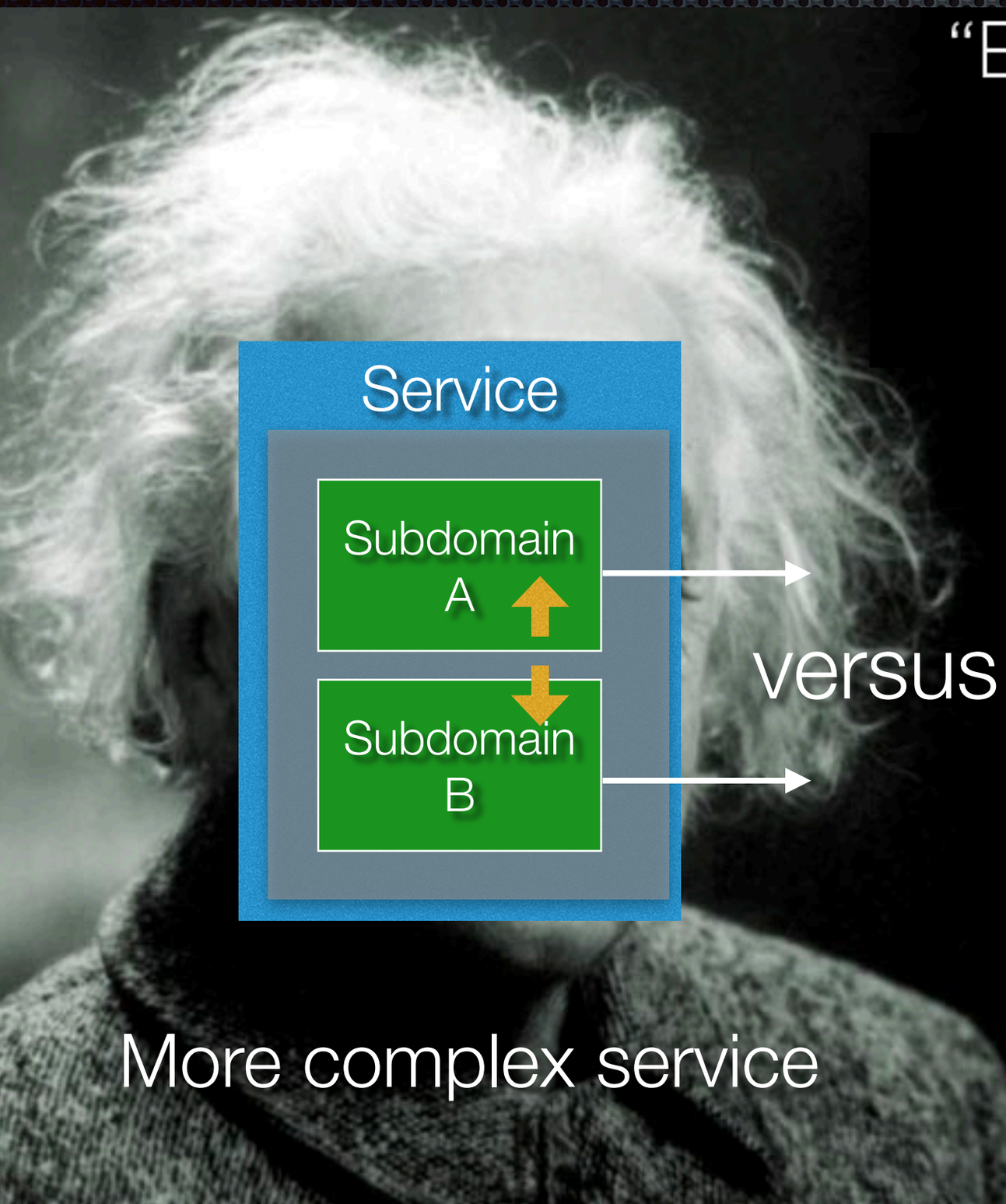
- Architecting for modern software delivery
- Dark energy and dark matter: forces that drive the architecture
- Dark energy: encouraging decomposition
- Dark matter: resisting decomposition

Repulsive forces ⇒ subdomains in different services

= reasons to migrate a module into a service

https://chrisrichardson.net/post/microservices/2021/04/15/mucon-2021-dark-energy-dark-matter.html

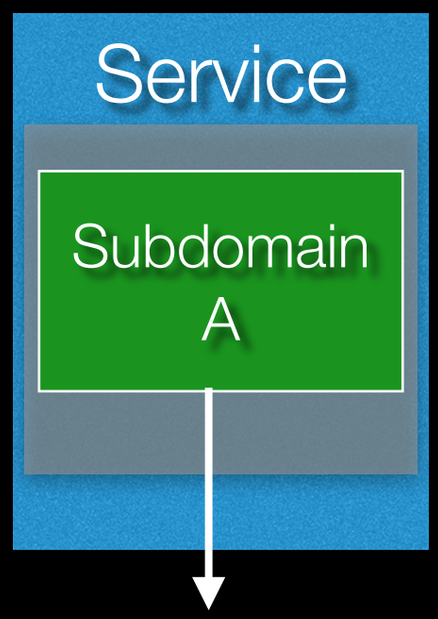
Simpler components/services

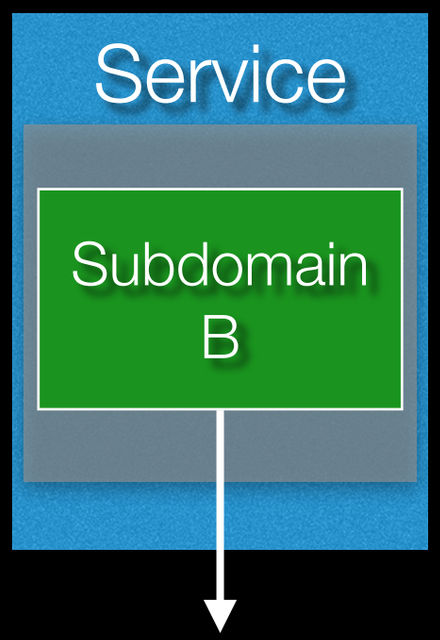


"Everything should be made as simple as possible. But not simpler."

Albert Einstein

versus



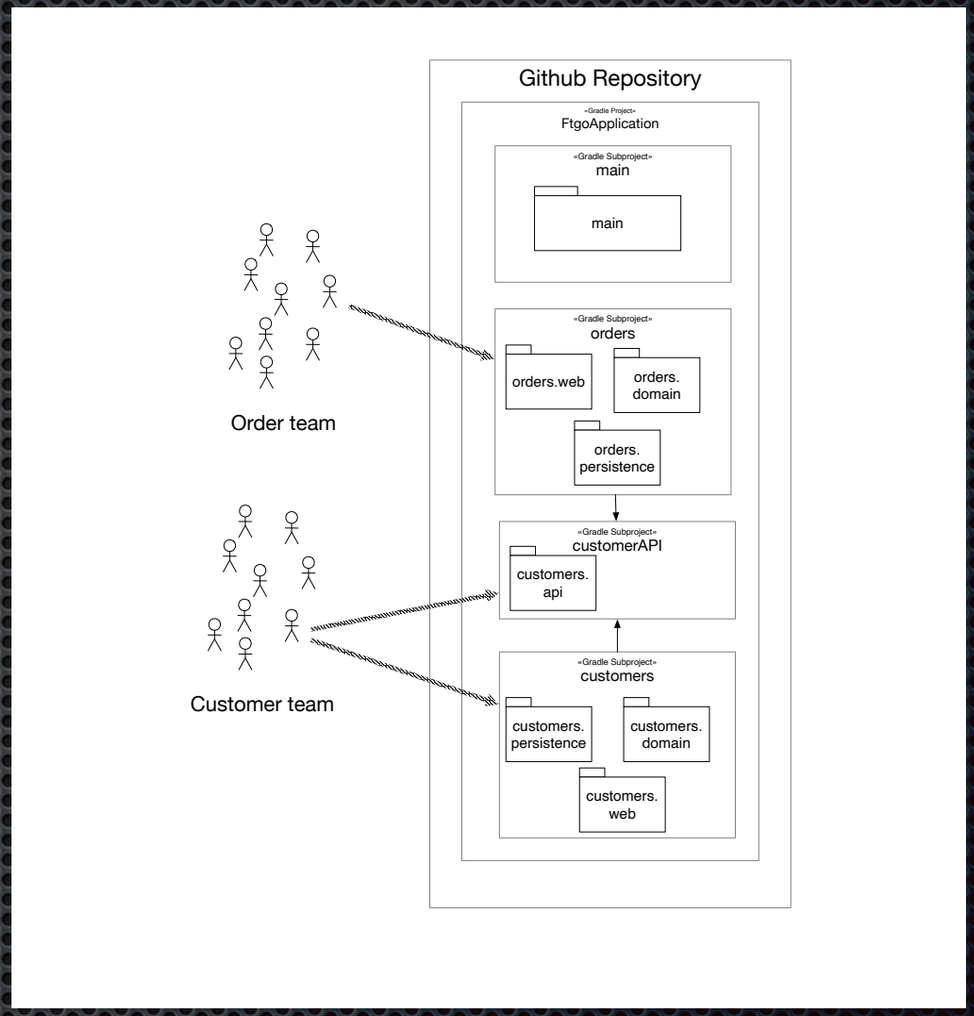


More complex service

Simpler services: easier to understand, develop, test, …

Simplifying a monolith

- Modularizing the monolith helps reduce complexity
- Developer can focus on their module



BUT

- Complexity $\propto$ Size

Module/New Feature → Service = simpler development

IF

- Module is actively being developed
- Monolith is large

Team autonomy = service per team

Team A

Team B

Service

Subdomain A

Subdomain B

Coordination required

vs.

Team A

Team B

Service

Subdomain A

Service

Subdomain B

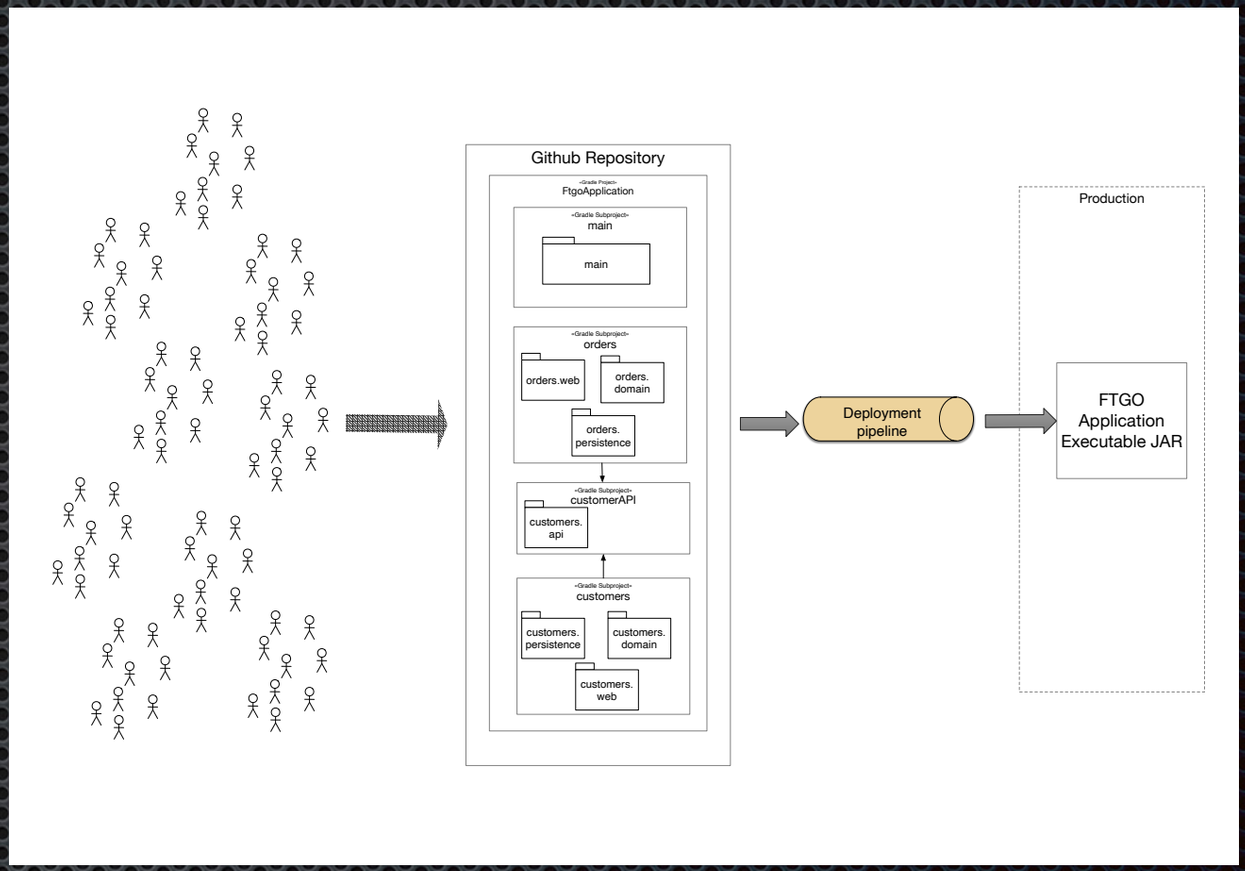
Build, test and deploy independently

Monoliths and team autonomy

- Modularization helps

BUT

- Single code base
- Autonomy ∝ 1/ # developers

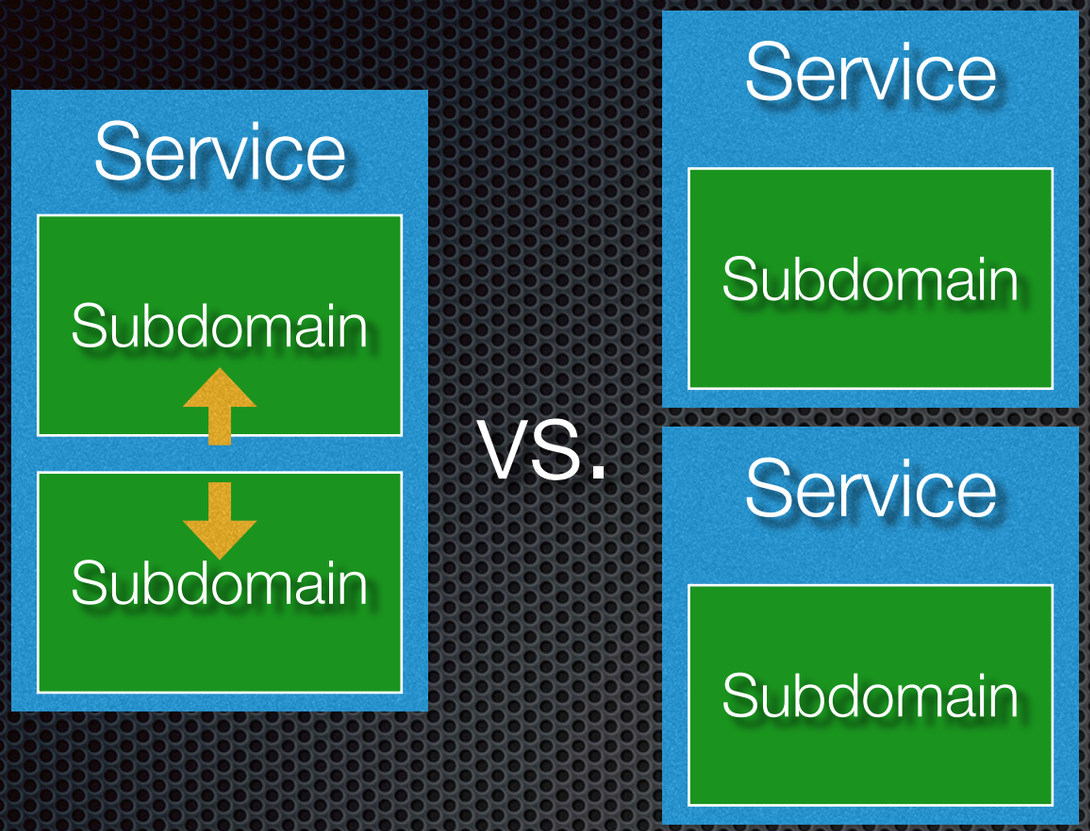


Module/New Feature → Service = increased autonomy

IF

- Module is actively being developed
- Many teams

Fast deployment pipeline



What does a good feedback loop look like?

1. Engineer merges a change to master
2. Any user-visible changes hidden safely behind a feature flag
3. Which triggers CI to run tests and build an artifact
4. The artifact is then deployed or canaried to production (or deployed to staging and later promoted to production).

Time elapsed: **<15min**

Notice:
- Only one changeset by one engineer per build
- No manual gates. No manual QA or variable times
- Feature flags to decouple deploys from releases

@mipsytipsy
https://speakerdeck.com/charity/cd?slide=17

Longer lead time

Shorter lead time

More complex build*

Simpler build

* Parallelizing building/testing partially helps

@crichardson

Optimizing a monolith's deployment pipeline

Use merge queue

```
$ git pull
$./gradlew test
$ git push
! [rejected] ….
```

Accelerate build/test:

- Incremental testing through DIP and ISP
- Parallelization/clustered builds
- Selective test execution

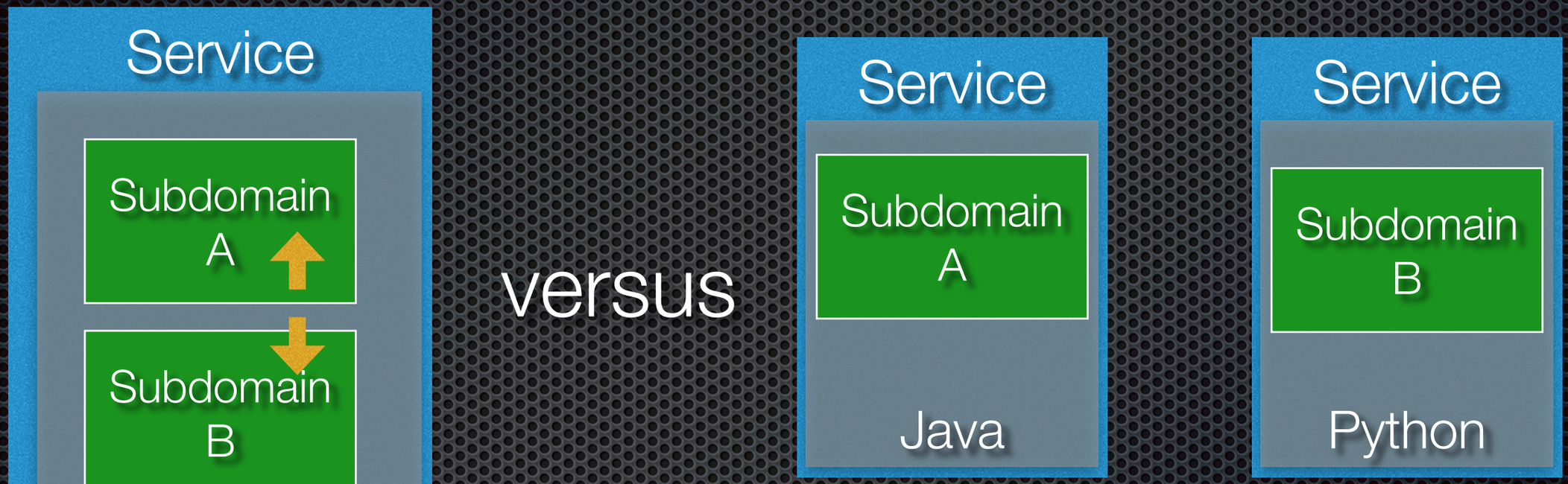
BUT if the application/team keeps growing
Then eventually the deployment pipeline = bottleneck

Module/New Feature → Service = faster deployment pipeline

IF

- Module is actively being developed
- Monolith is large
- Many teams

Support multiple technology stacks



Single technology stack

Upgrade together

Separate technology stacks

Right tool for the job

Upgrade independently

Experiment easily

Monolith = single technology stack

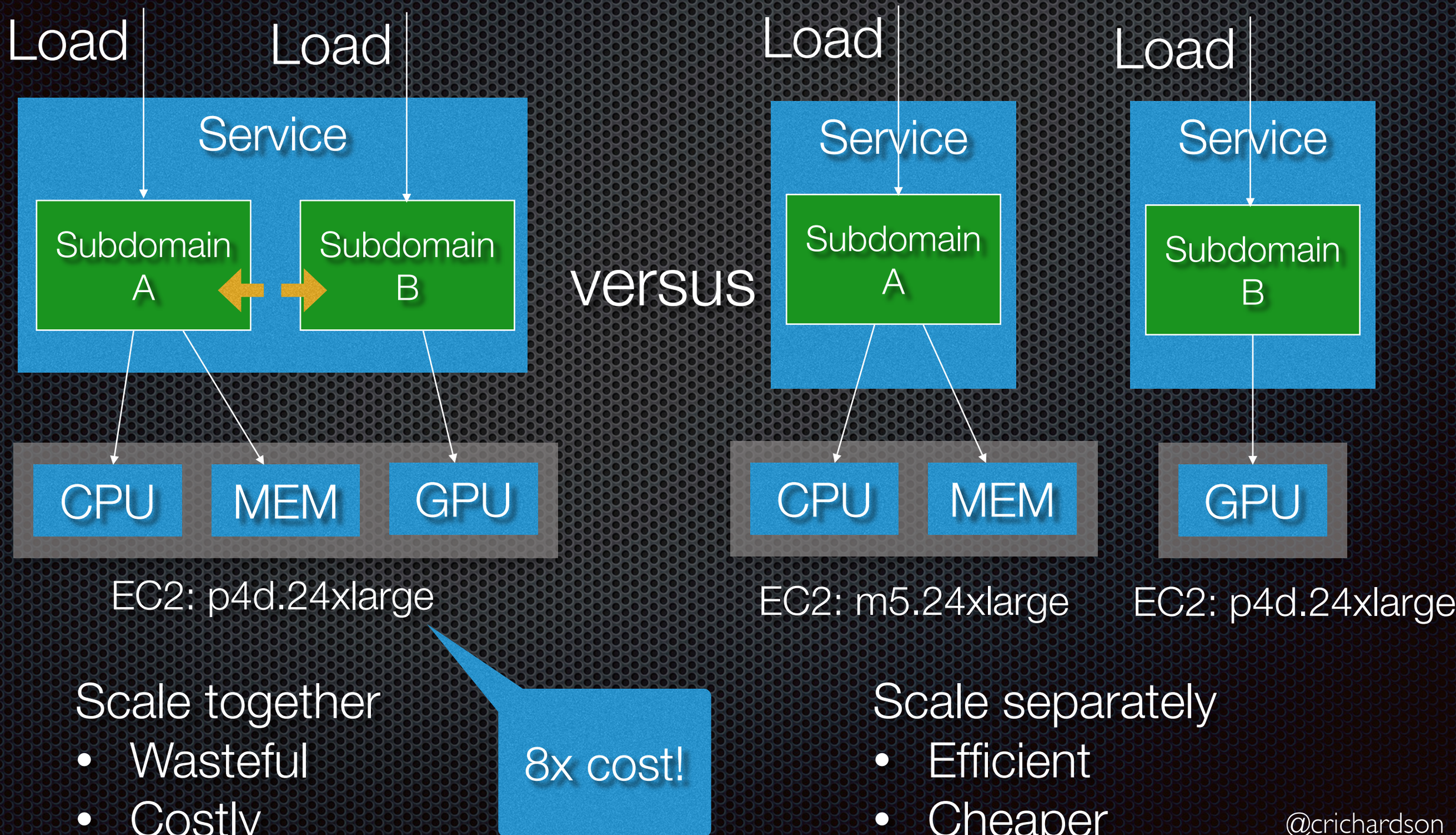
- Single class path unless using exotic technology, eg. Layrry
- Single version of each dependency => big bang upgrades
- No opportunity to experiment
- No possibility of using non-JVM technologies, e.g. Python

Module/New Feature → Service = multiple technology stacks

Separate subdomains by characteristics

Subdomain characteristic	Issue
Resource requirements	Cost-effective, scalability
Regulations, e.g. SaMD/ PCI	DevOps vs. Slower regulated process
Business criticality/tier	Maximize availability
Security, e.g. PII, …	Improve security
DDD core/supporting/ generic	Focus on being competitive

Cost effective scaling



Scale together
- Wasteful
- Costly

Scale separately
- Efficient
- Cheaper

Example: Segregate by business criticality

2.9% + 30c/ request

Revenue loss and penalties

chargeCard()

Service

Critical

Payment Processing

Important

Merchant management

vs.

chargeCard()

Service

Payment Processing

Service

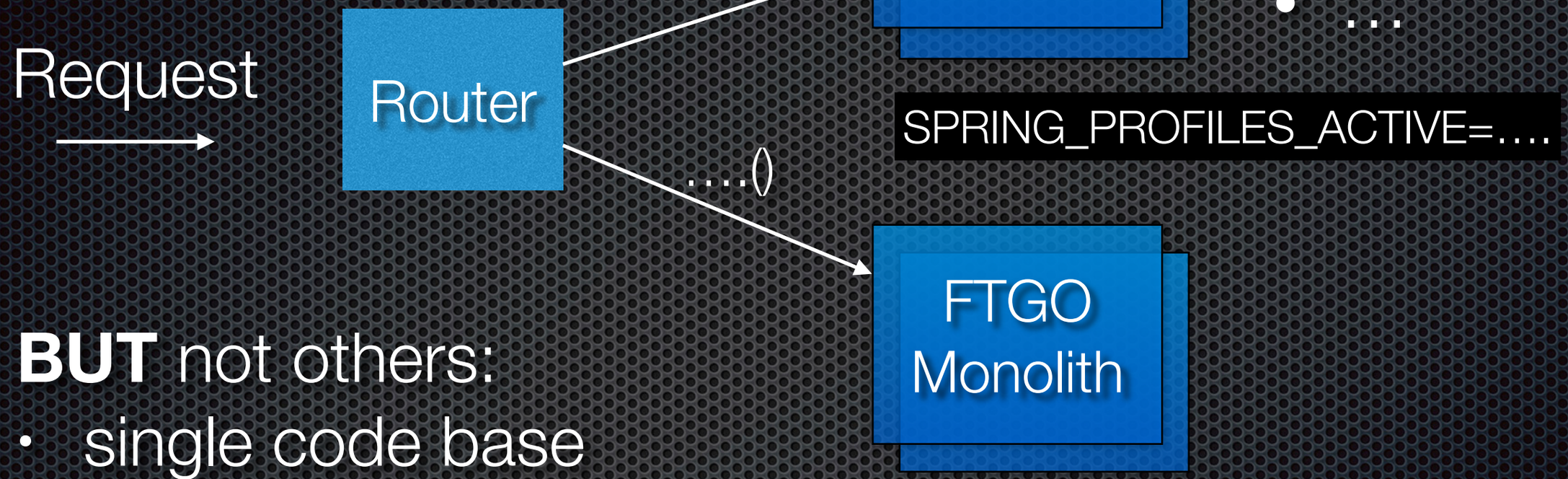
Merchant management

Shared infrastructure
Shared code base
Risk of interference

Separate infrastructure
Separate code base
Isolated

Multiple “deployments” can address some requirements

- Scalability
- Availability
- …



BUT not others:

- single code base
- ….

Module/New Feature → Service

=

Improves architecture via separation

Some parts of your monolith will benefit more from microservices

e.g. actively being developed

=>

Using dark energy to identify candidate services

Faster deployment pipeline

Python technology stack

Improved autonomy

Develops

ML-based Fraud detection Service

E-Commerce Monolith

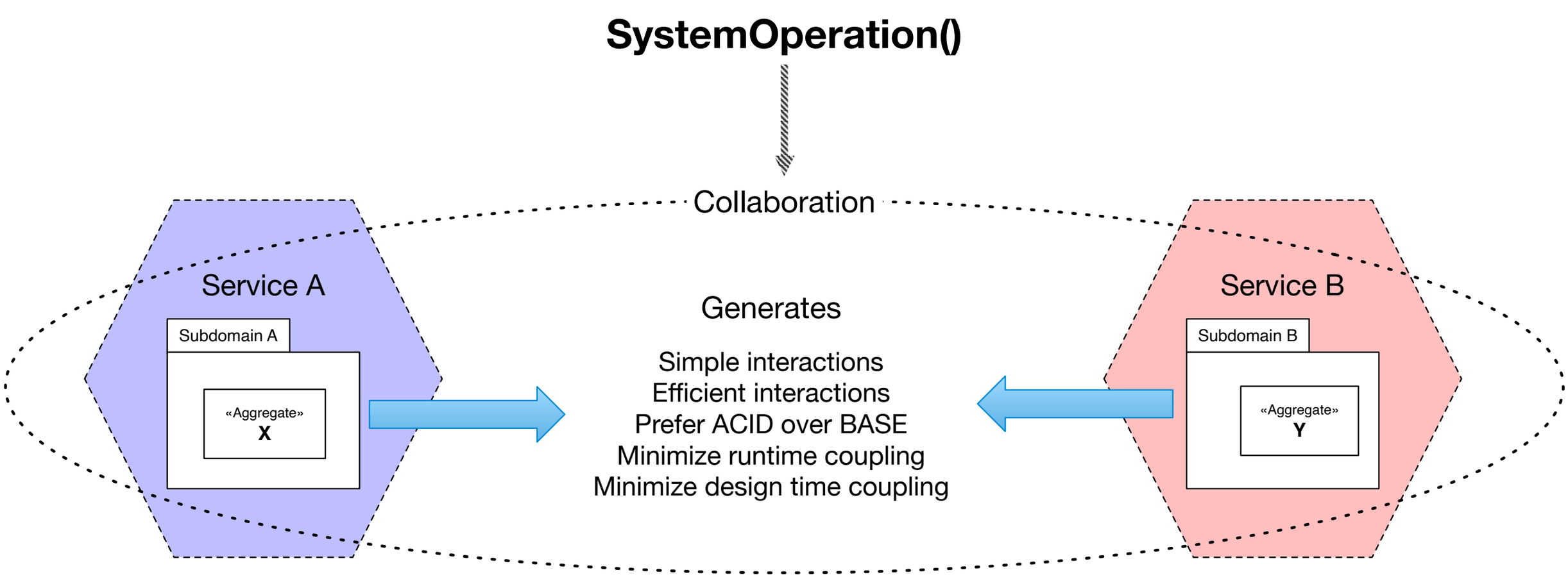
Data science team (Not “hardcore” devs)

Simplified development experience

Agenda

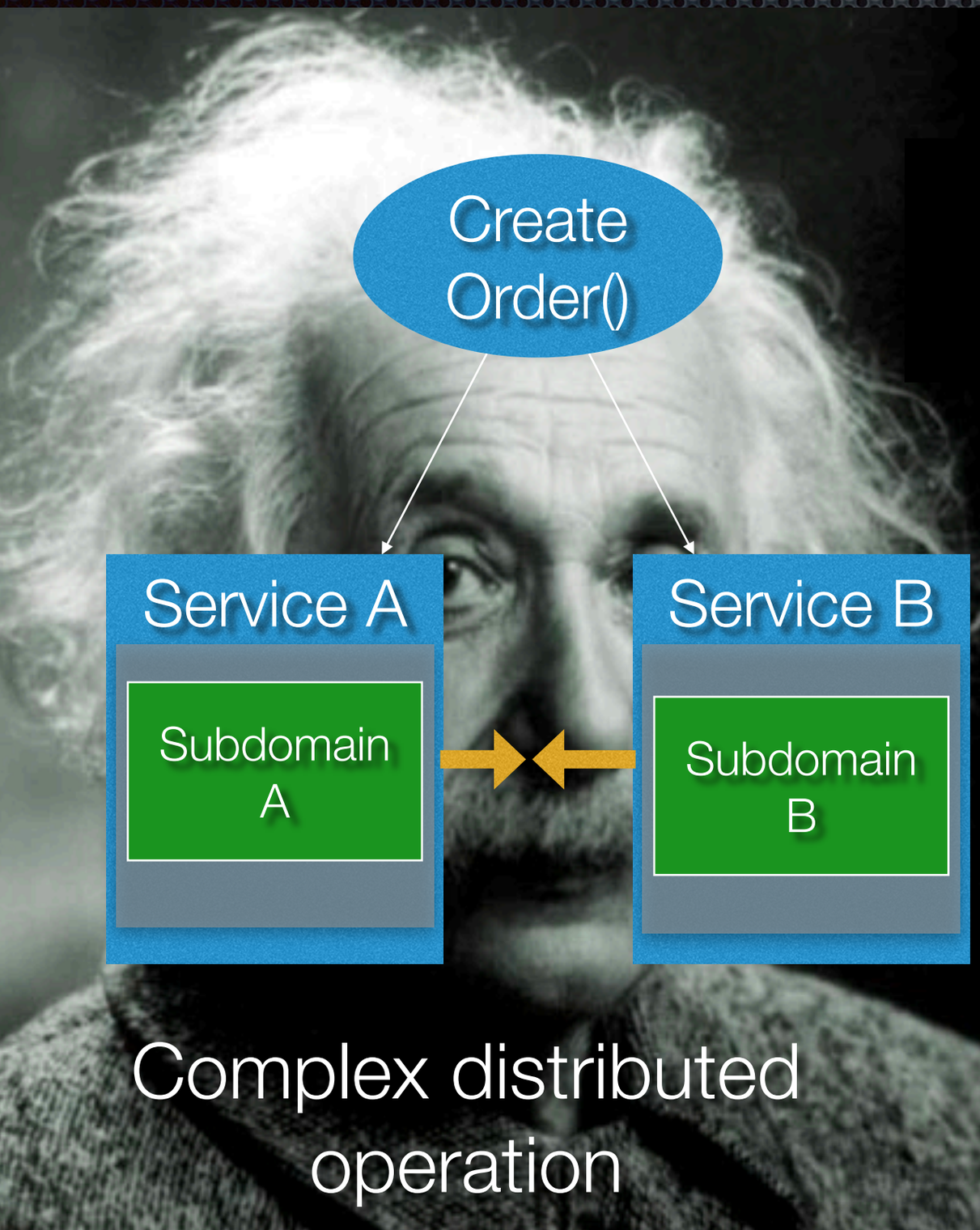
- Architecting for modern software delivery
- Dark energy and dark matter: forces that drive the architecture
- Dark energy: encouraging decomposition
- Dark matter: resisting decomposition

Attractive forces ⇒ subdomains in same service



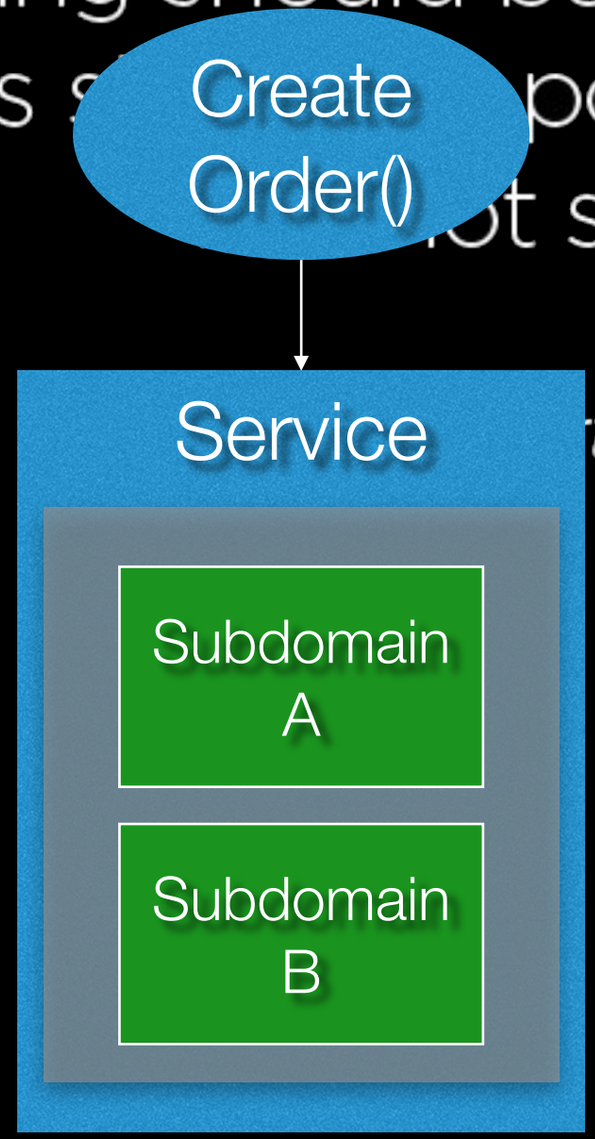
https://chrisrichardson.net/post/microservices/2021/04/15/mucon-2021-dark-energy-dark-matter.html

Simple interactions



Complex distributed operation

vs.



Simple local operation: easier to understand, troubleshoot, …

"Everything should be made as s possible. ot simpler."

t Einstein

Impact of extracting services on complexity

createOrder()

...

FTGO Monolith

Order Management

Delivery Management

... Management

Might increase complexity

createOrder()

...

Collaboration

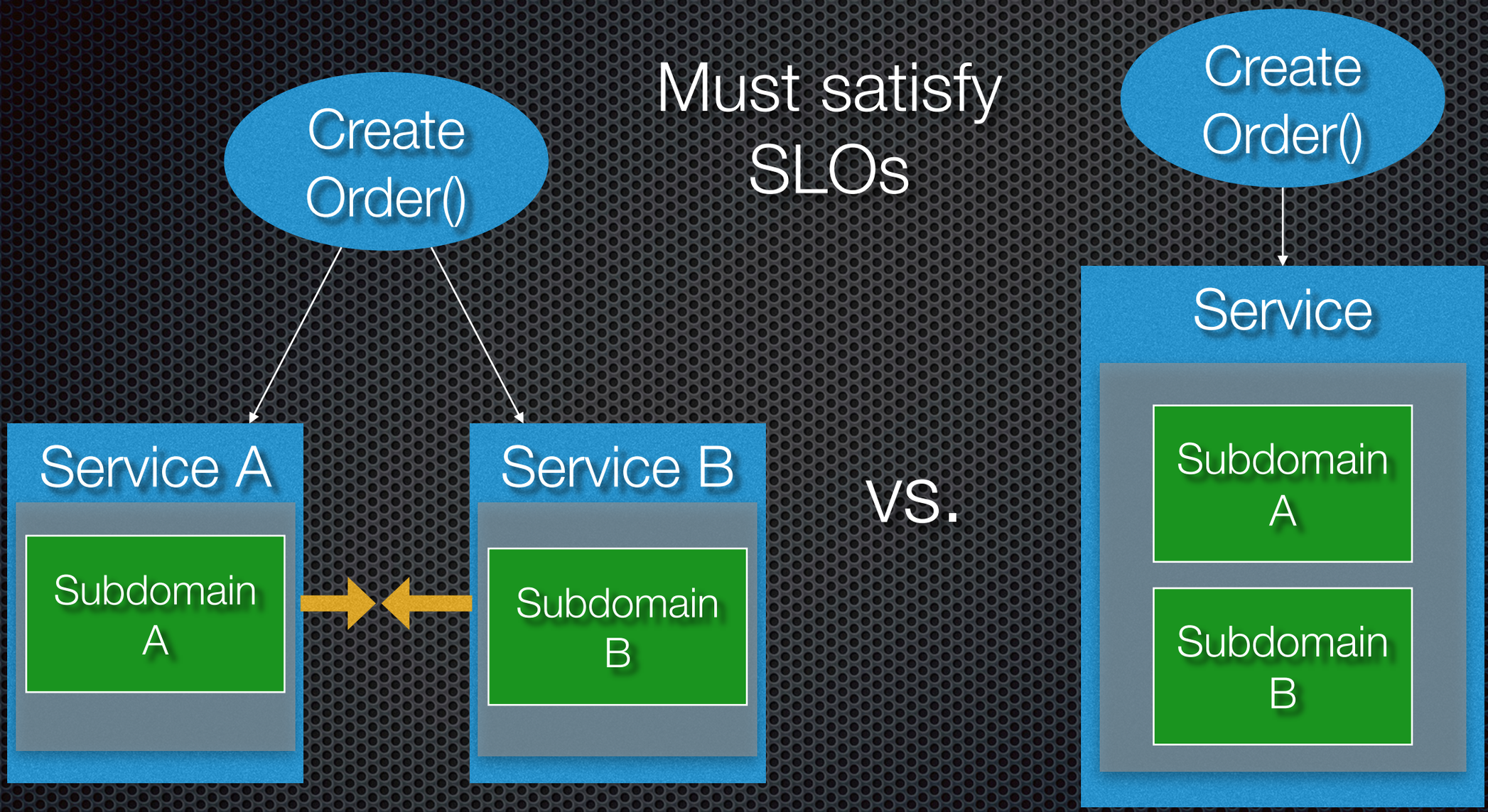
FTGO Monolith

Order Management

... Management

Delivery Service

Efficient interactions



Network latency, limited bandwidth

In-memory, fast!

Impact of extracting services on efficiency

createOrder()

...

FTGO Monolith

Order Management

Delivery Management

... Management

Might be too inefficient

createOrder()

...

Collaboration

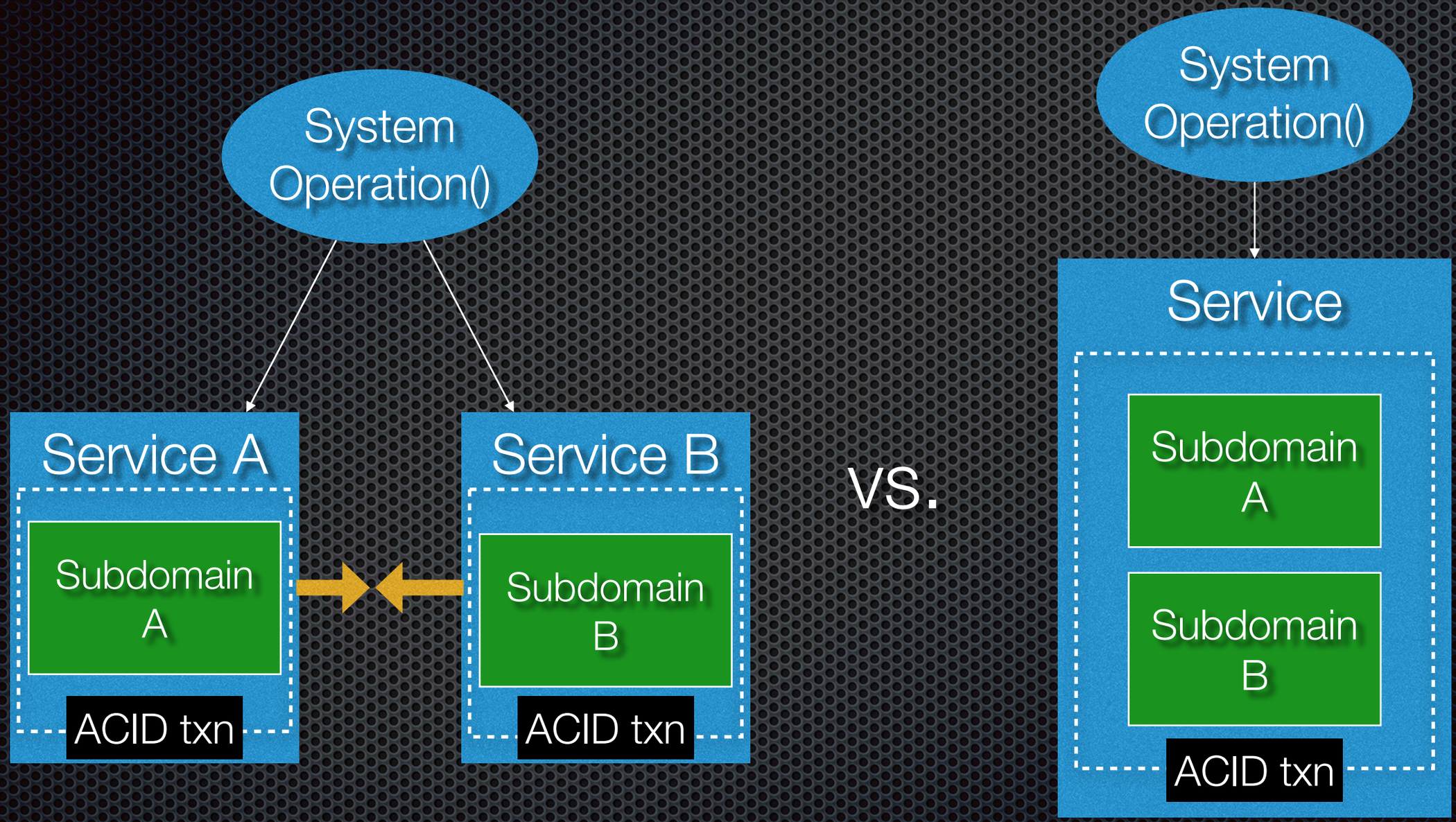
FTGO Monolith

Order Management

... Management

Delivery Service

Prefer ACID over BASE



Distributed, eventually consistent transaction

Simple, Local ACID transaction

Impact of extracting services on transactions

createOrder()

...

FTGO Monolith

Order Management

Delivery Management

... Management

Might need compensating transactions = complex changes to monolith

createOrder()

...

Saga

1. createOrder()

2. rejectOrder()

2. createDelivery() FAILS

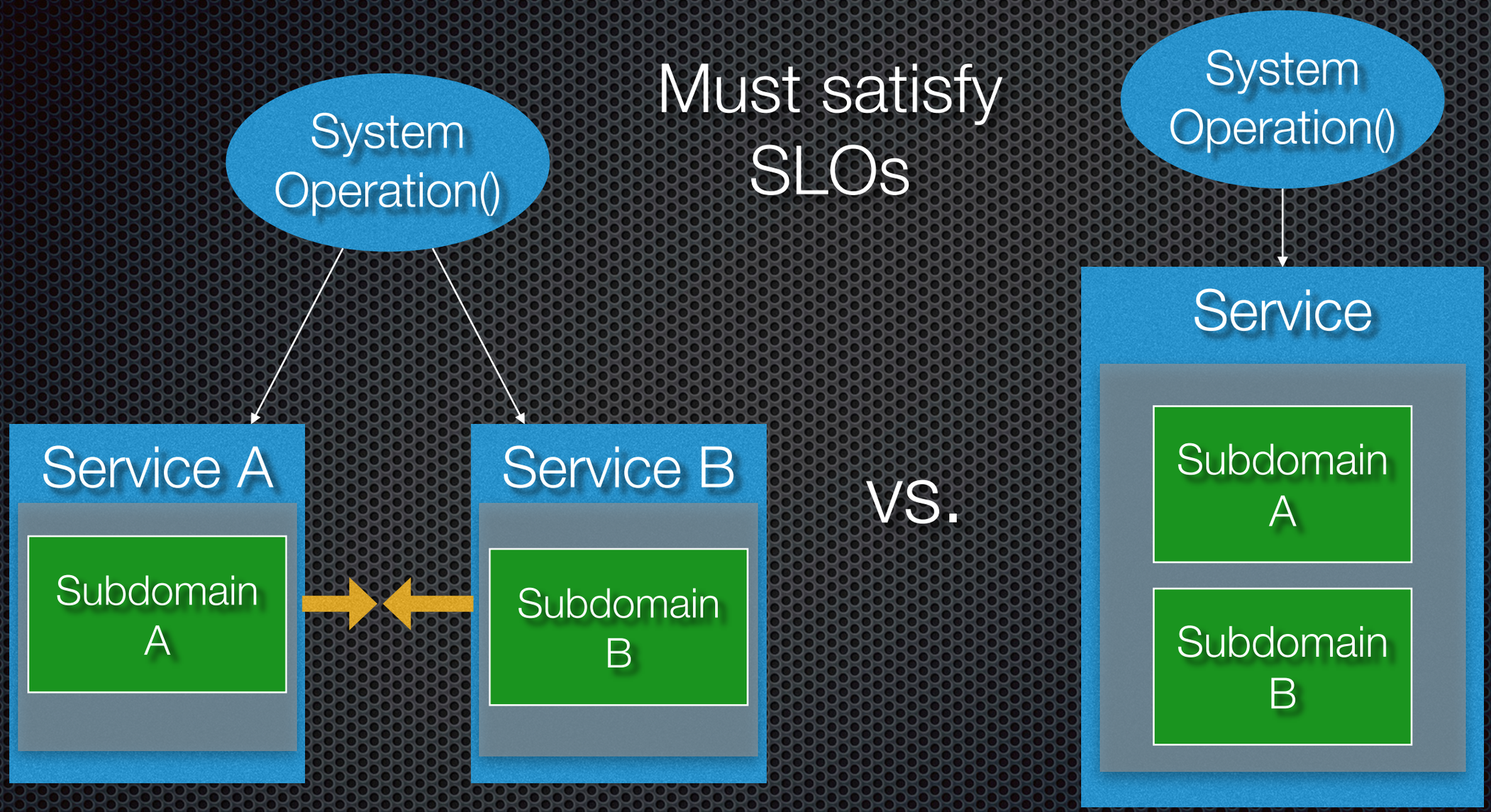
FTGO Monolith

Order Management

... Management

Delivery Service

Minimize runtime coupling



Risk of runtime coupling

No runtime coupling: higher availability, lower latency

Impact of extracting services on runtime coupling

createOrder()

...

FTGO Monolith

Order Management

Delivery Management

... Management

Might have excessive runtime coupling

createOrder()

...

Collaboration

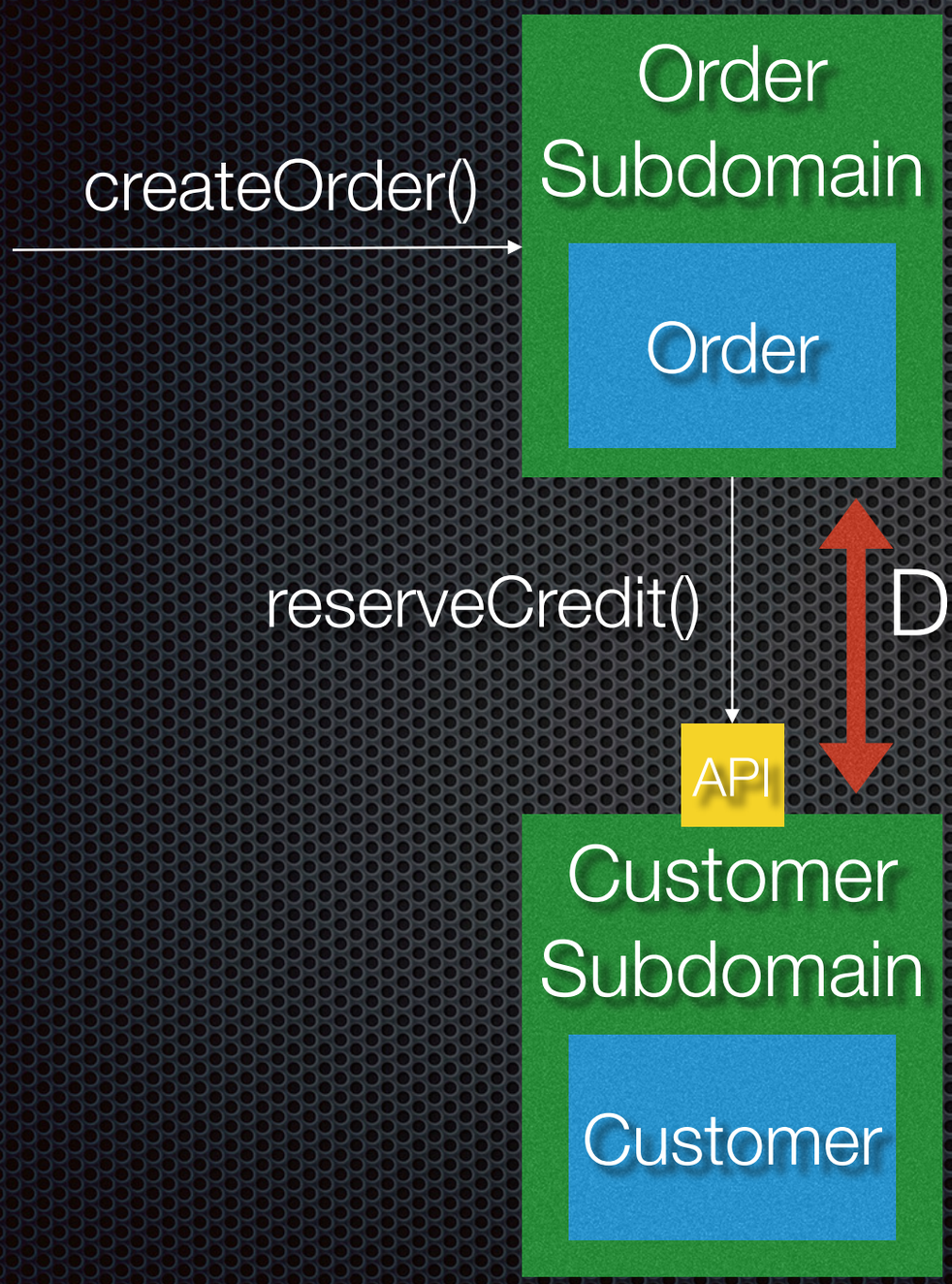
FTGO Monolith

Order Management

... Management

Delivery Service

Minimize design time coupling



Minimize with careful design

BUT

You can't always eliminate it

⇒

Risk of lock step changes

Risk proportional to:

- API instability
- API complexity
- …

Impact of extracting services on design-time coupling

- Monolith and new service might have excessive design-time coupling

BUT

- Experience with monolith = domain expertise = MonolithFirst
  - Increases likelihood of designing services with stable API
  - Reduced risk of accidentally creating design-time coupled services

Consequences of dark matter forces

Prefer ACID over BASE:

Might need expensive to implement compensating transactions in monolith

createOrder()

…

FTGO Monolith

Order Management

Delivery Management

… Management

Might not resolve dark matter forces:

Infeasible architecture

createOrder()

…

Collaboration

FTGO Monolith

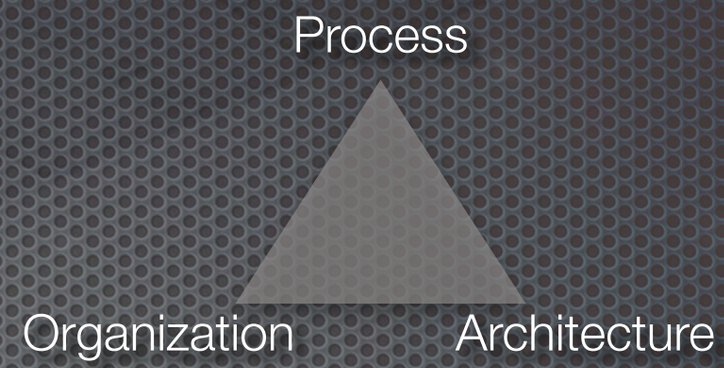
Order Management

… Management

Delivery Service

@crichardson

Summary



- Don’t automatically assume you need microservices:
  - Make the most of your monolith
  - Improve your process and organization

BUT

An application/organization can outgrow its monolithic architecture

THEREFORE

- Incrementally refactor to microservices
- Dark energy forces help identify candidate services
- Dark matter forces can make extracting a service infeasible or expensive

https://www.nasa.gov/feature/goddard/2019/nasa-s-james-webb-space-telescope-has-been-assembled-for-the-first-time

@crichardson chris@chrisrichardson.net

Questions?

adopt.microservices.io