There is no such thing as a microservice!

Chris Richardson

Founder of Eventuate.io

Founder of the original CloudFoundry.com

Author of POJOs in Action


@crichardson

chris@chrisrichardson.net

http://microservices.io

http://eventuate.io

http://plainoldobjects.com

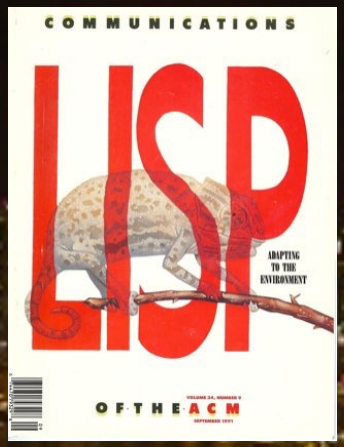
SATURN 2017

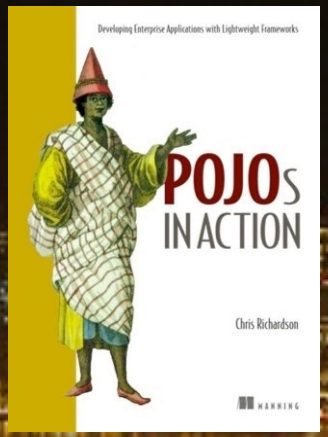
Presentation goal

Define the Microservice Architecture as an architectural style

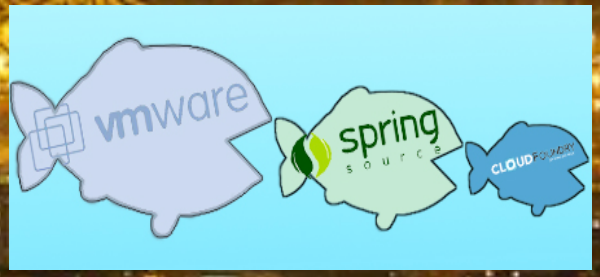
Explain what that means and why it matters

About Chris


COMMUNICATIONS
LISP
ADAPTING TO THE ENVIRONMENT
OF THE ACM


Developing Enterprise Applications with Lightweight Frameworks
POJOs IN ACTION
Chris Richardson
MANNING


CLOUDFOUNDRY
SIGN IN
SIGN UP
SYSTEM ALERT. PLEASE READ: Cloud Foundry will be moving to a new URL.
The Enterprise Java Cloud
Real Java Applications Deployed in Minutes
Built for Spring and Grails Web Applications
Most Widely Used Technologies Delivered as a Platform
SIGN UP
LEARN MORE
APPLICATION DEMO
Deploying Web Applications To Amazon EC2 with Cloud Foundry


vmware
spring
source
CLOUDFOUNDRY

About Chris

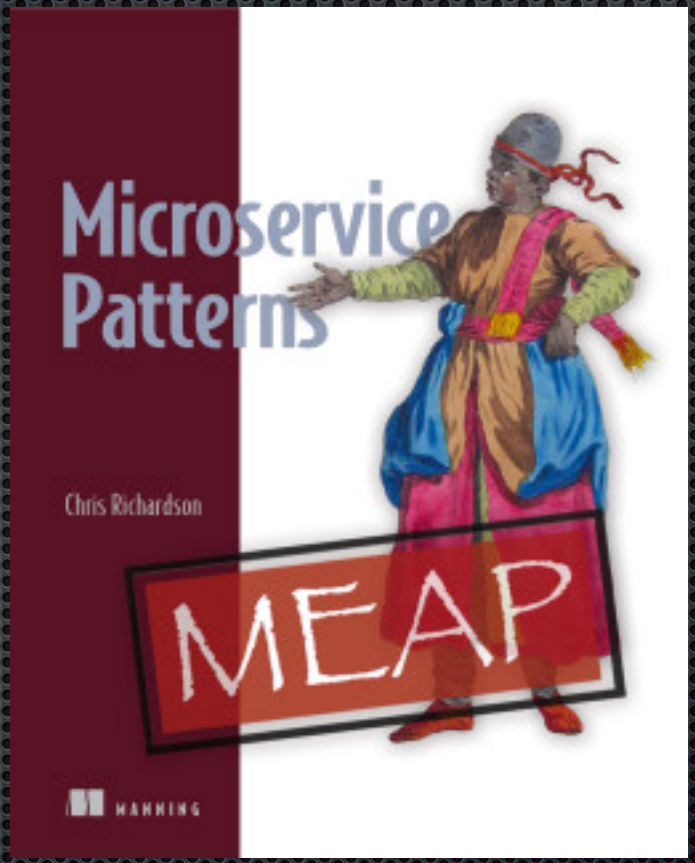
Consultant and trainer focusing on modern application architectures including microservices (http://www.chrisrichardson.net/)

About Chris

Founder of a startup that is creating an open-source/SaaS platform that simplifies the development of transactional microservices

(http://eventuate.io)

About Chris



https://www.manning.com/books/microservice-patterns

For more information

http://learnmicroservices.io

Agenda

- A brief refresher on software architecture
- From monolith to microservices
- Microservices != silver bullet
- Managing distributed data

About software architecture

"The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

Documenting Software Architectures, Bass et al

Architecture

=

(elements, relations, properties)

Architecture

=

Boxes and lines 😄

Architecture is multi-dimensional

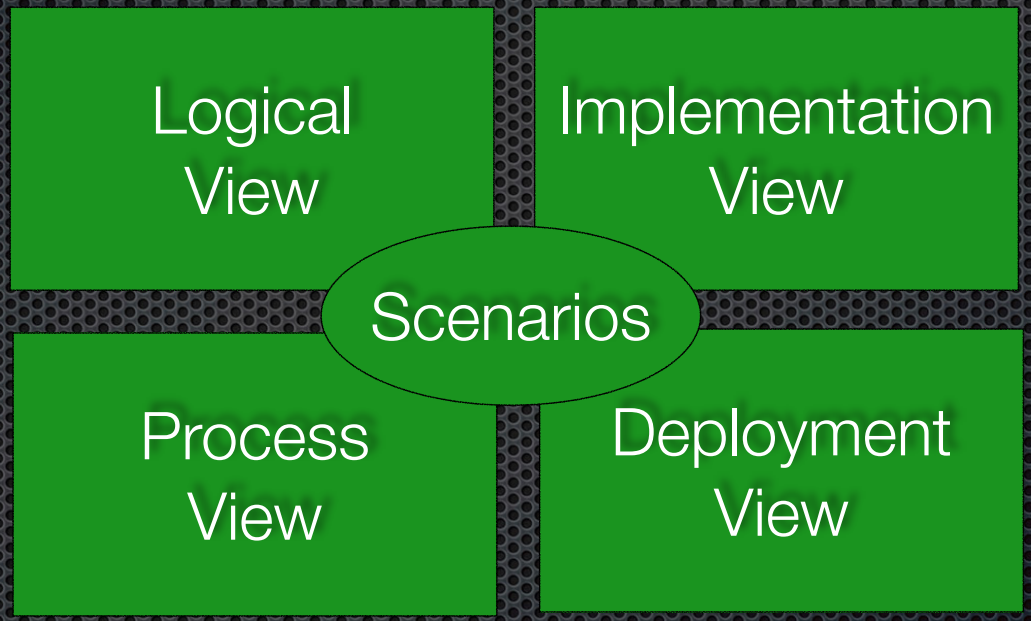
e.g. Structural, electrical, plumbing, mechanical

⇒

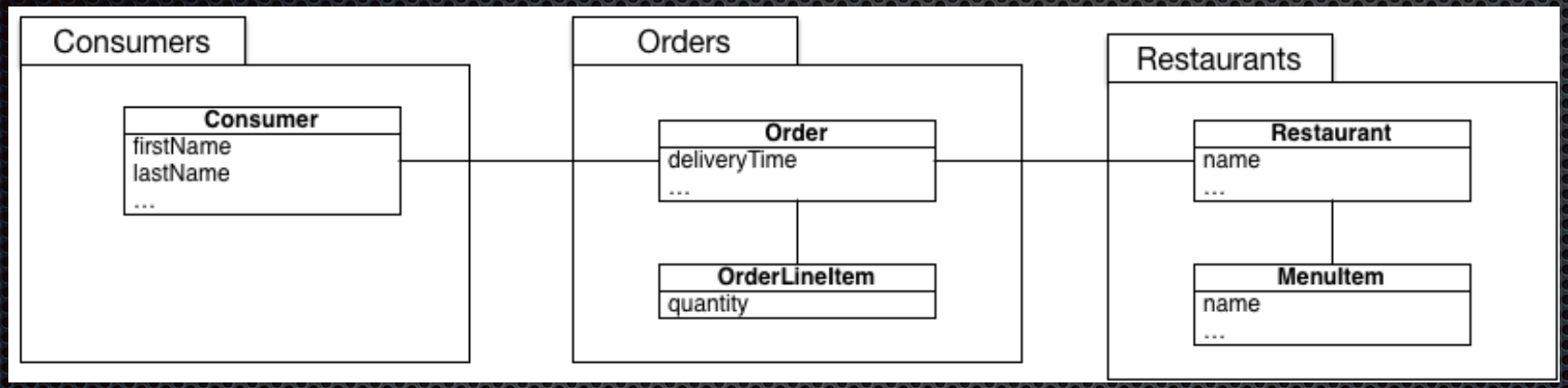
Described by multiple views

View = (elements, relations, properties)

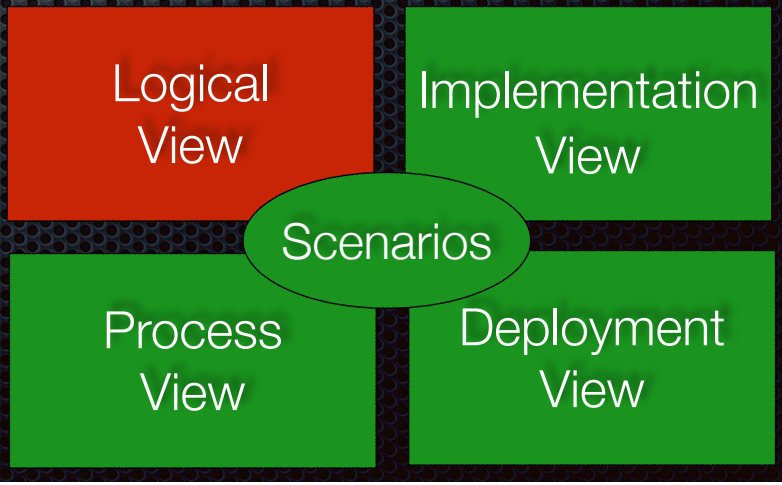
4+1 view model



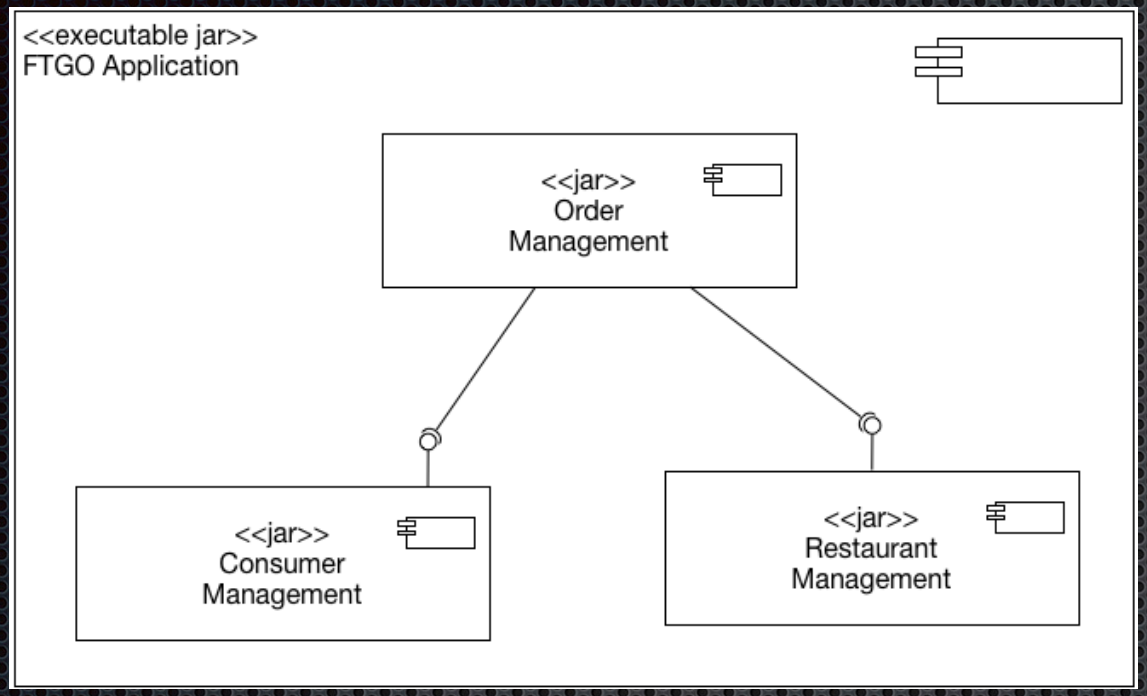
4+1 Logical view



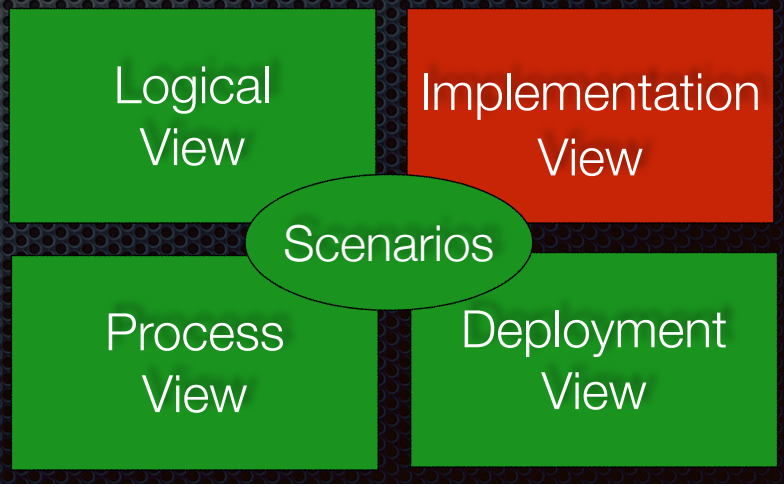
Elements: classes and packages
Relations: inheritance, associations, …



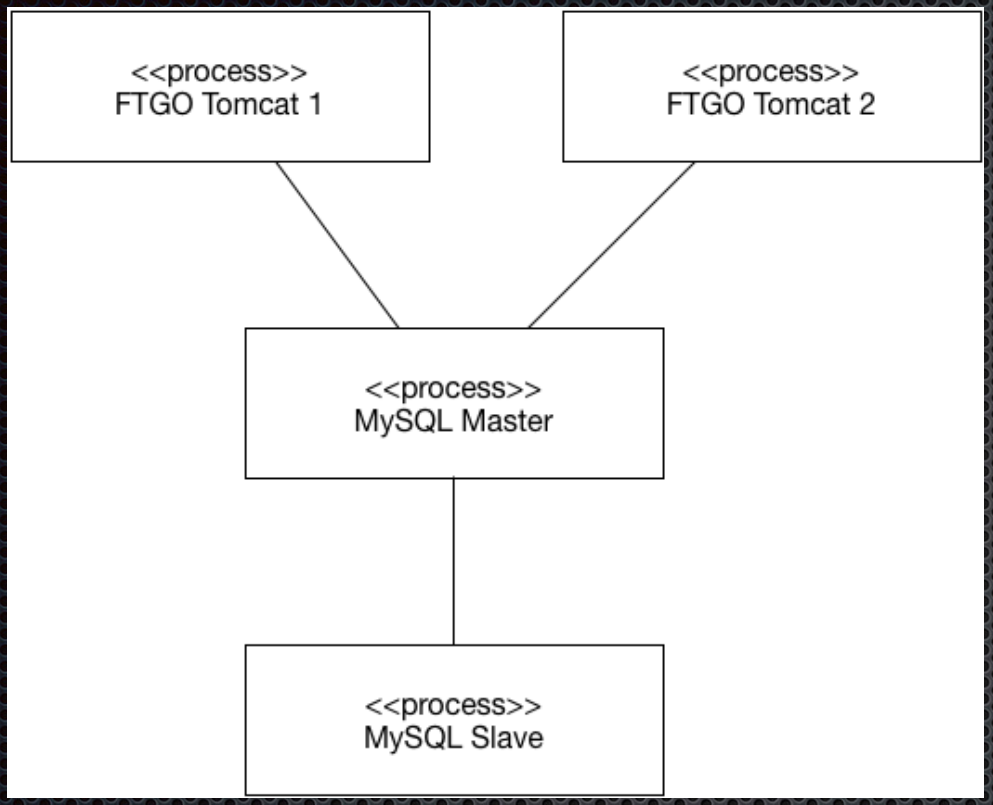
4+1 Implementation view



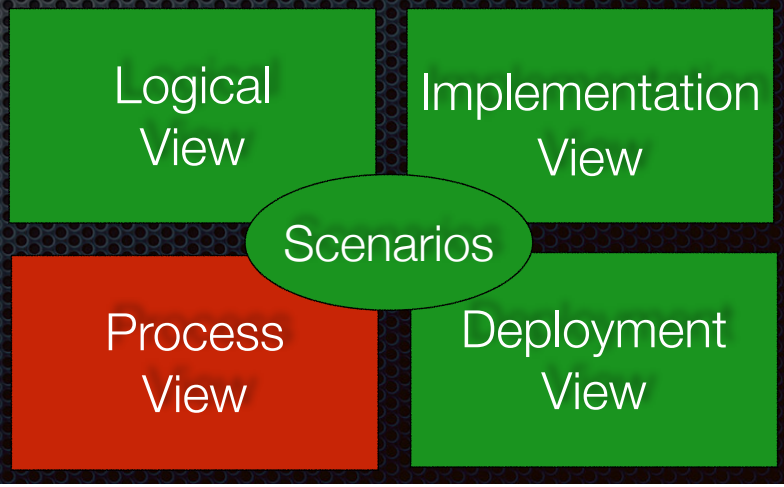
Elements: modules and components
Relations: dependencies



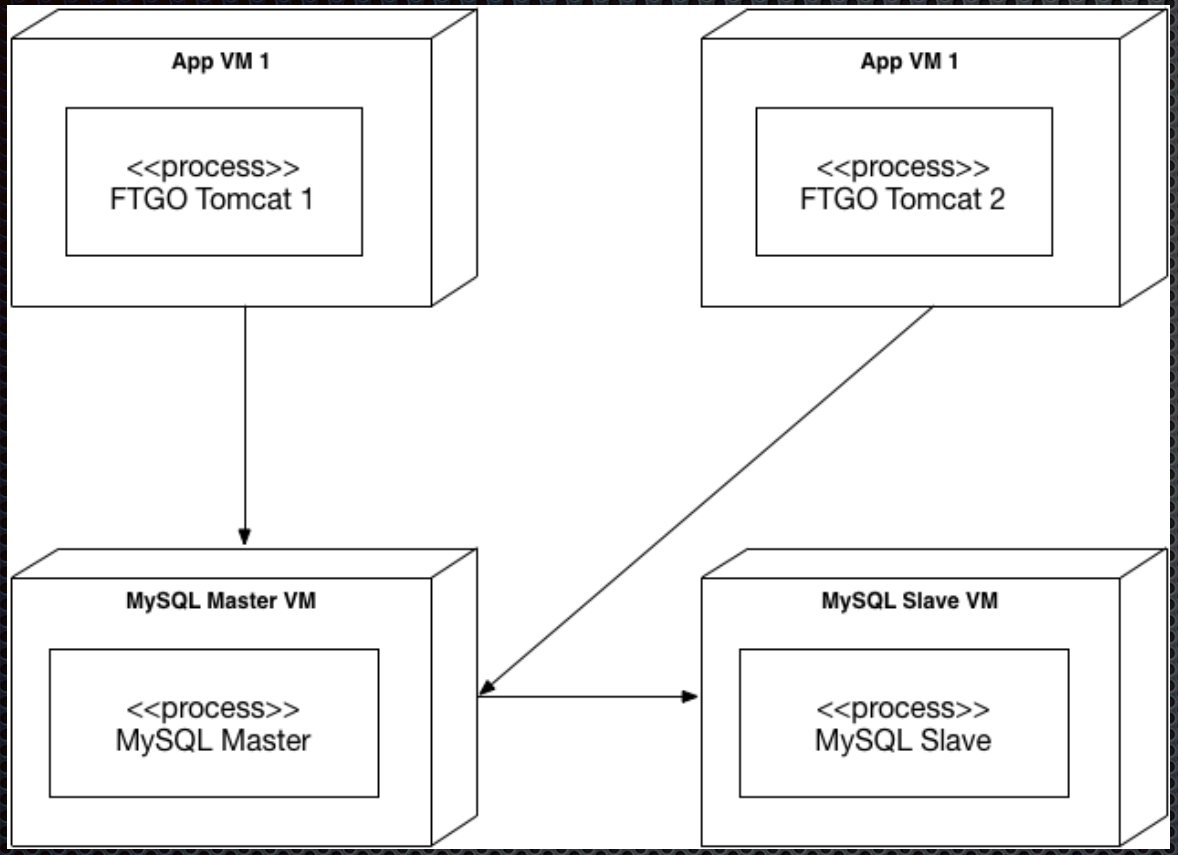
4+1 Process view



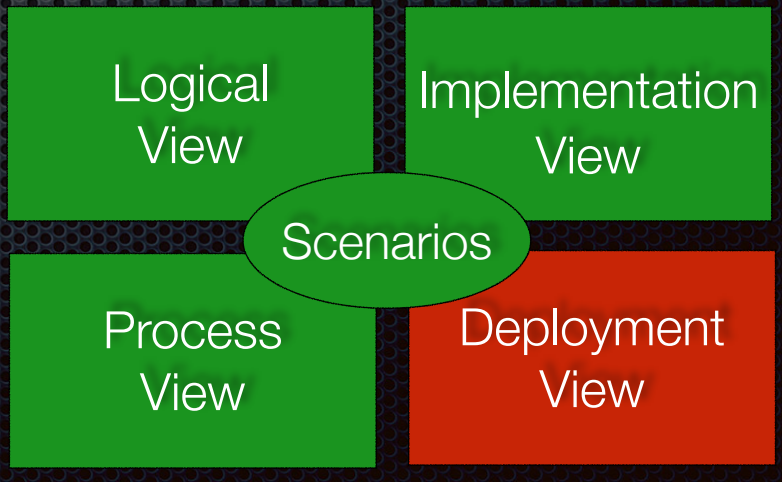
Elements: processes
Relations: IPC



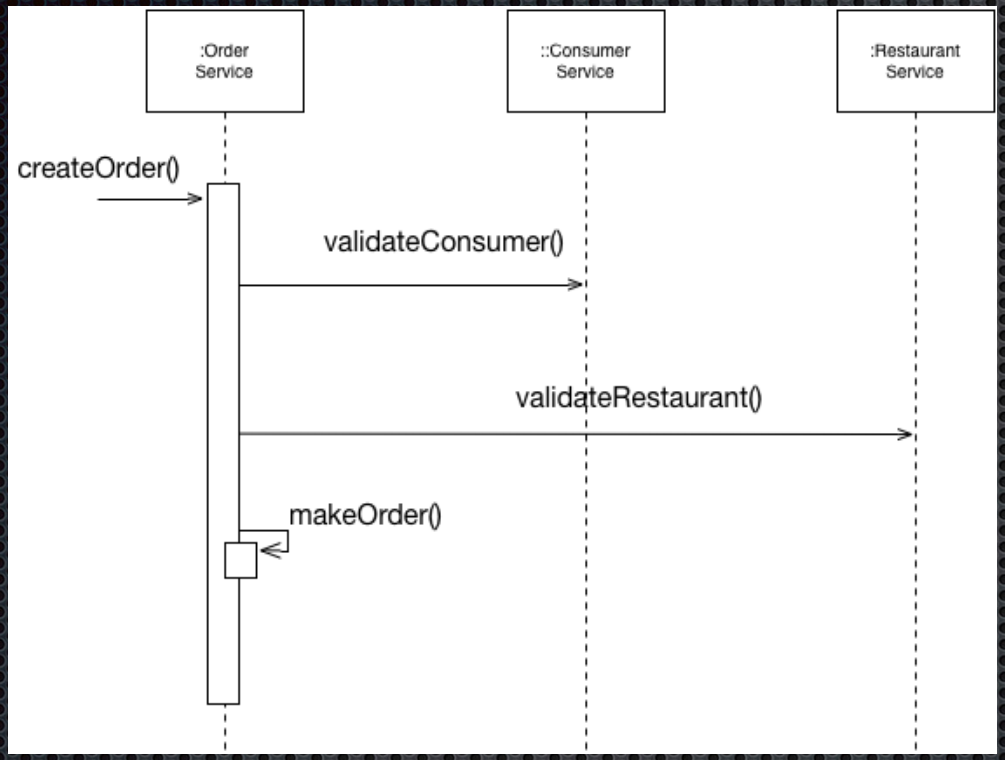
4+1 Deployment view



Elements: “machines”
Relations: networking

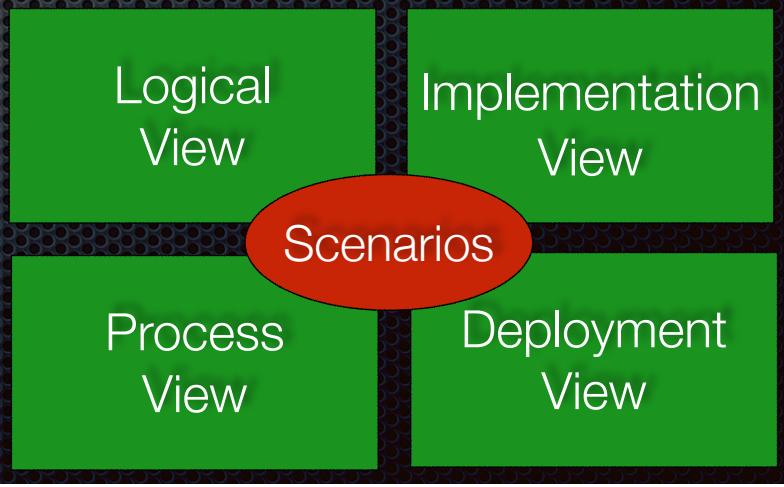


4+1 Scenarios



Derived from use cases/stories

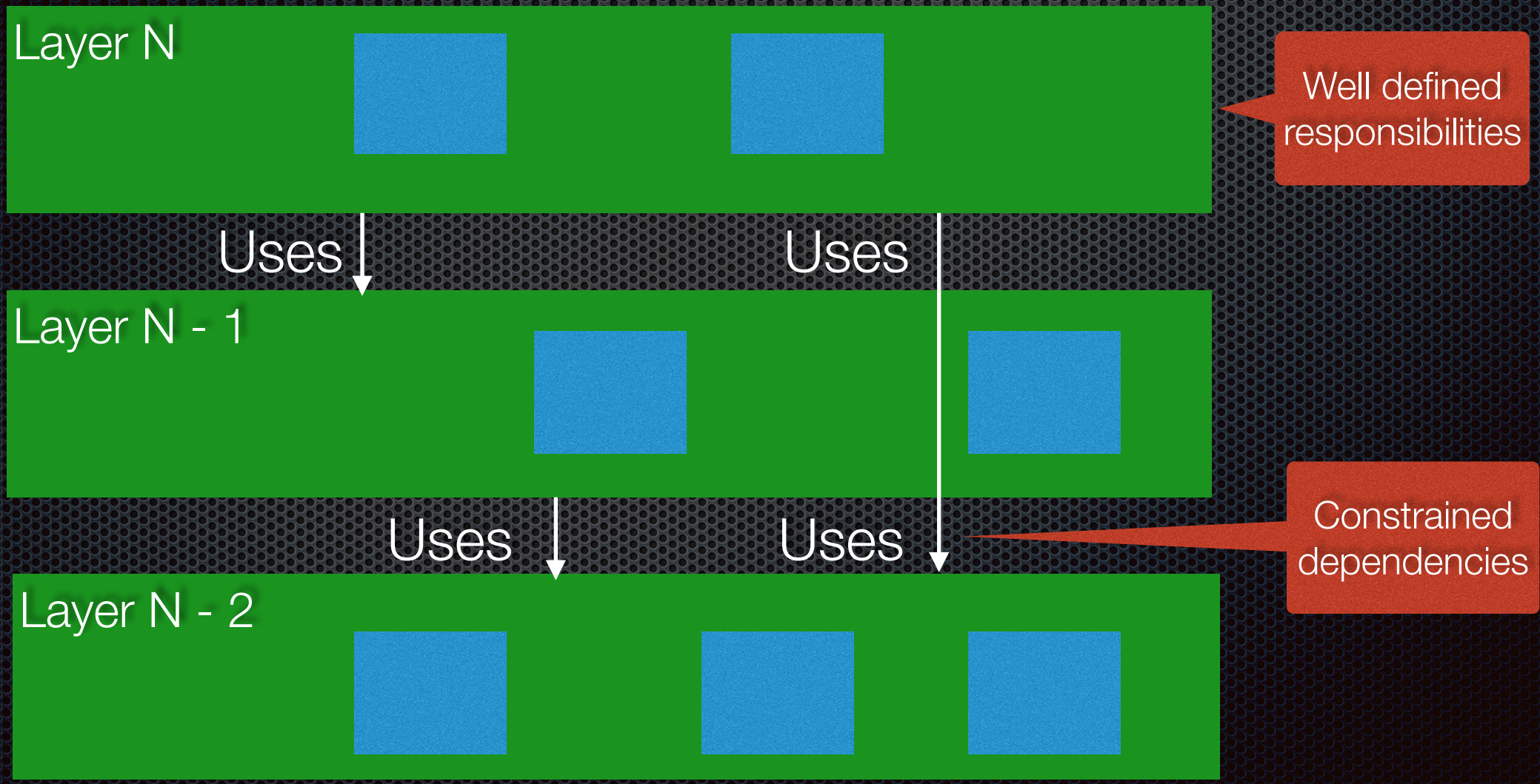
Animate the views



"“… An **architectural style** determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined…..”"

David Garlan and Mary Shaw, An Introduction to Software Architecture

Layered architectural style



The role of architecture

Requirements

=

Functional requirements

+

Non-functional requirements (-ilities)

Domain knowledge

Architecture

Important -ilities

- Maintainability
- Evolvability
- Testability
- Deployability
- Scalability
- Security
- Reliability
- …

Development velocity

https://en.wikipedia.org/wiki/Non-functional_requirement

Businesses must innovate faster

⇒

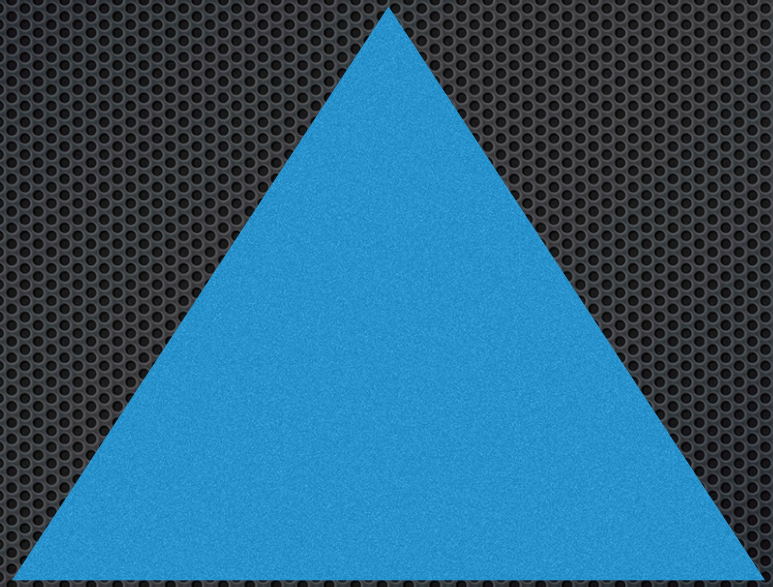
Build better software faster

Reducing lead time

Increasing deployment frequency

Modern software development

Process: Continuous delivery/deployment



Organization:

Small, autonomous teams

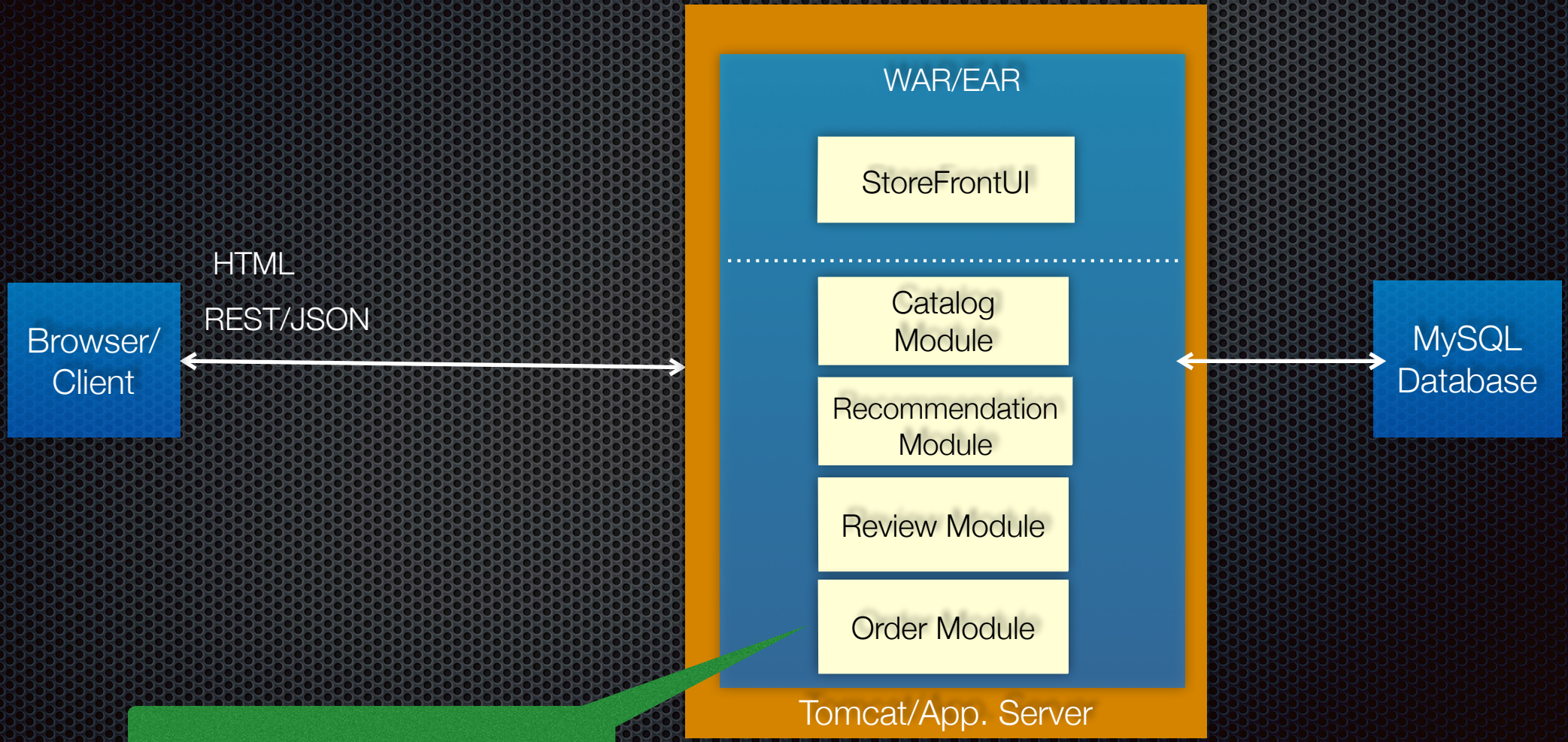
Architecture:

???

Agenda

- A brief refresher on software architecture
- From monolith to microservices
- Microservices != silver bullet
- Managing distributed data

Traditional: Monolithic architecture



The monolithic architecture is an **architectural style** that structures the **application** as a **single executable component**

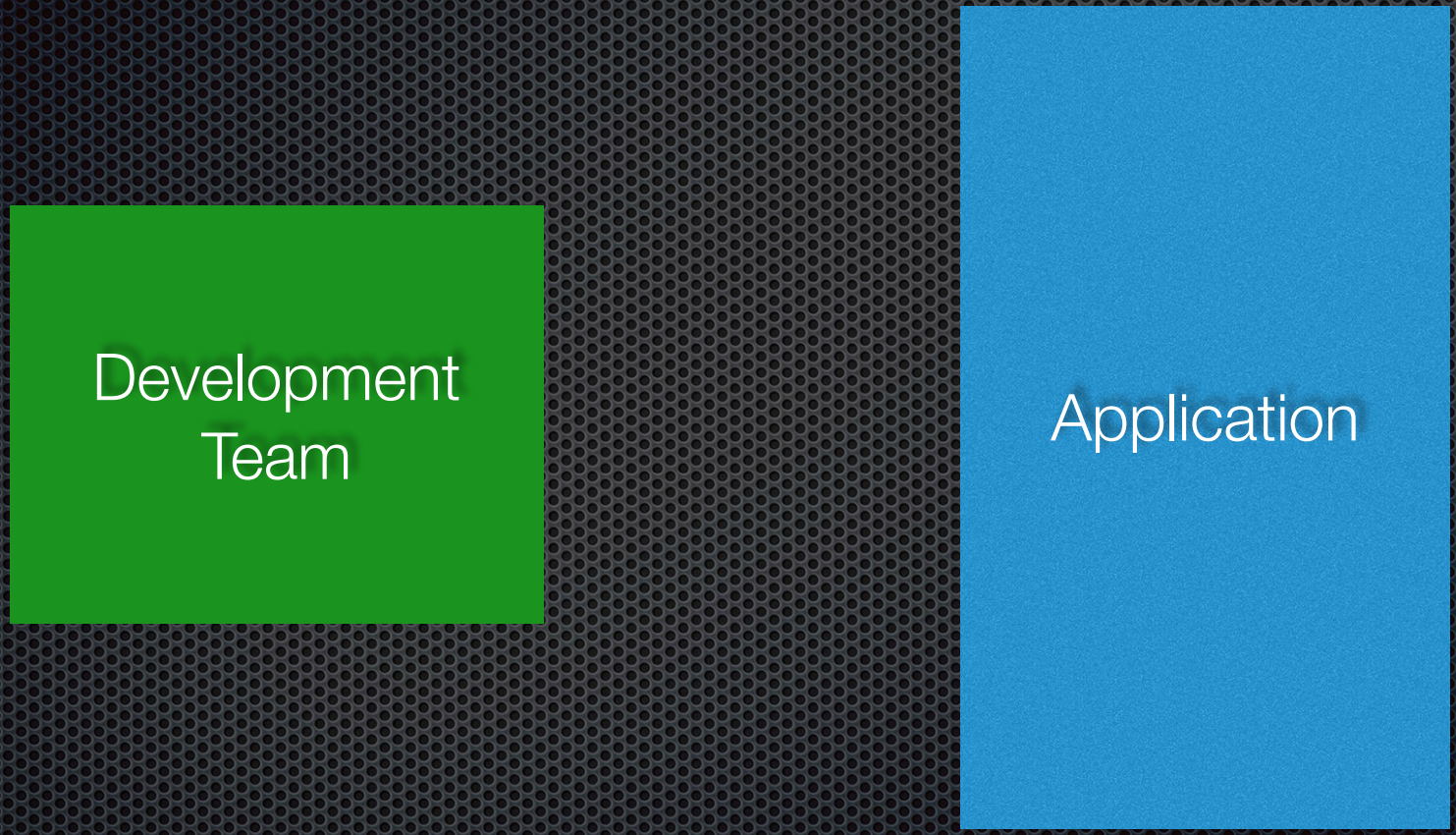
Implementation view

-ilities of small monoliths

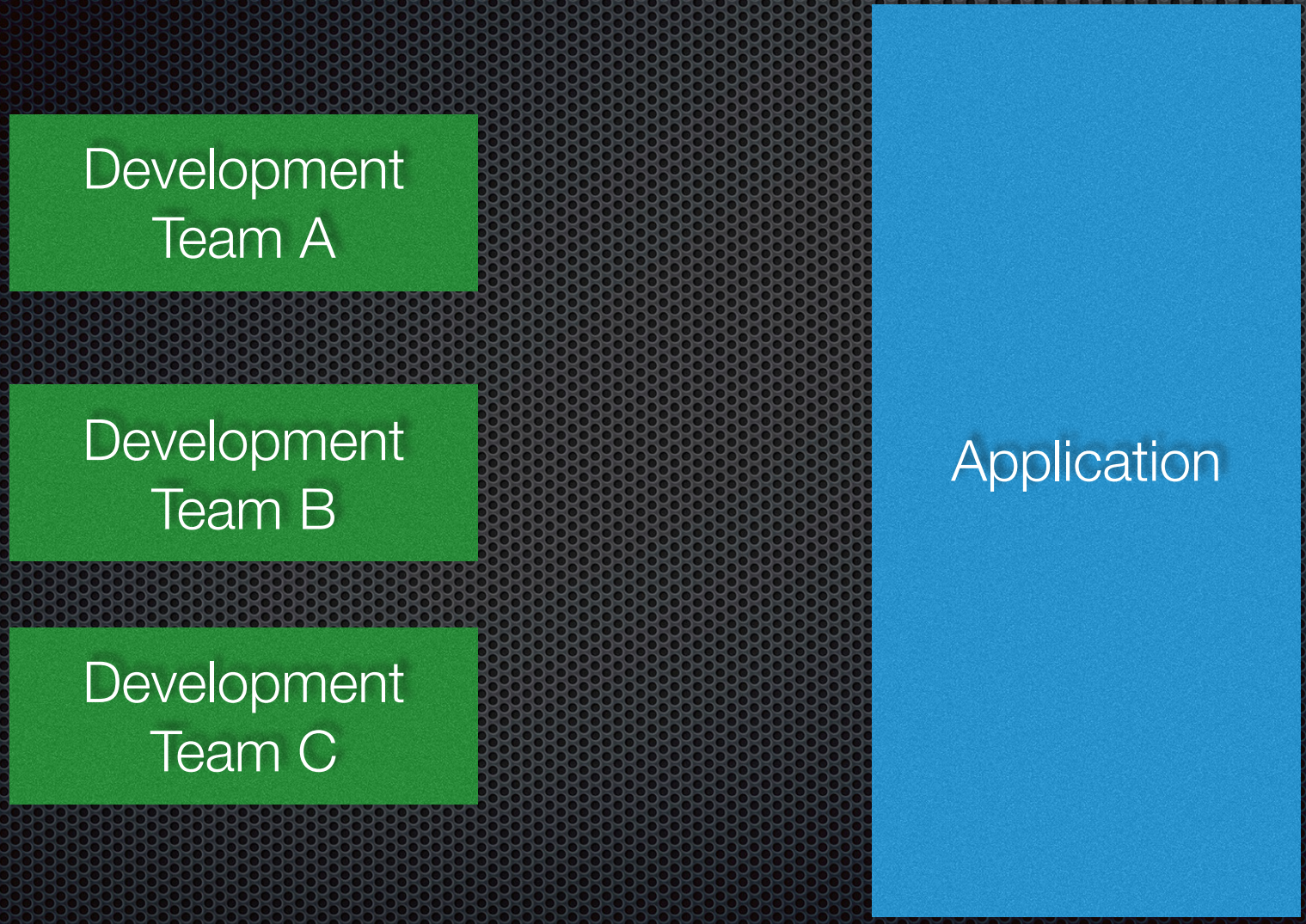
- Maintainability
- Evolvability
- Testability
- Deployability
- Scalability
- …

https://en.wikipedia.org/wiki/Non-functional_requirement

But successful applications keep growing....



… and growing


Development Team A
Development Team B
Development Team C
Application

Eventually:

agile development and deployment becomes impossible

=

monolithic hell

Technology stack becomes increasingly obsolete

BUT

A rewrite is not feasible

-ilities of **large** monoliths

- Maintainability
- Evolvability
- Testability
- Deployability
- …

https://en.wikipedia.org/wiki/Non-functional_requirement

The microservice architecture is an **architectural style** that **structures** an application as a **set of loosely coupled, services** organized around **business capabilities**

Two levels of architecture

Application-level

Services
Inter-service glue: interfaces and communication mechanisms
Slow changing

Service-level

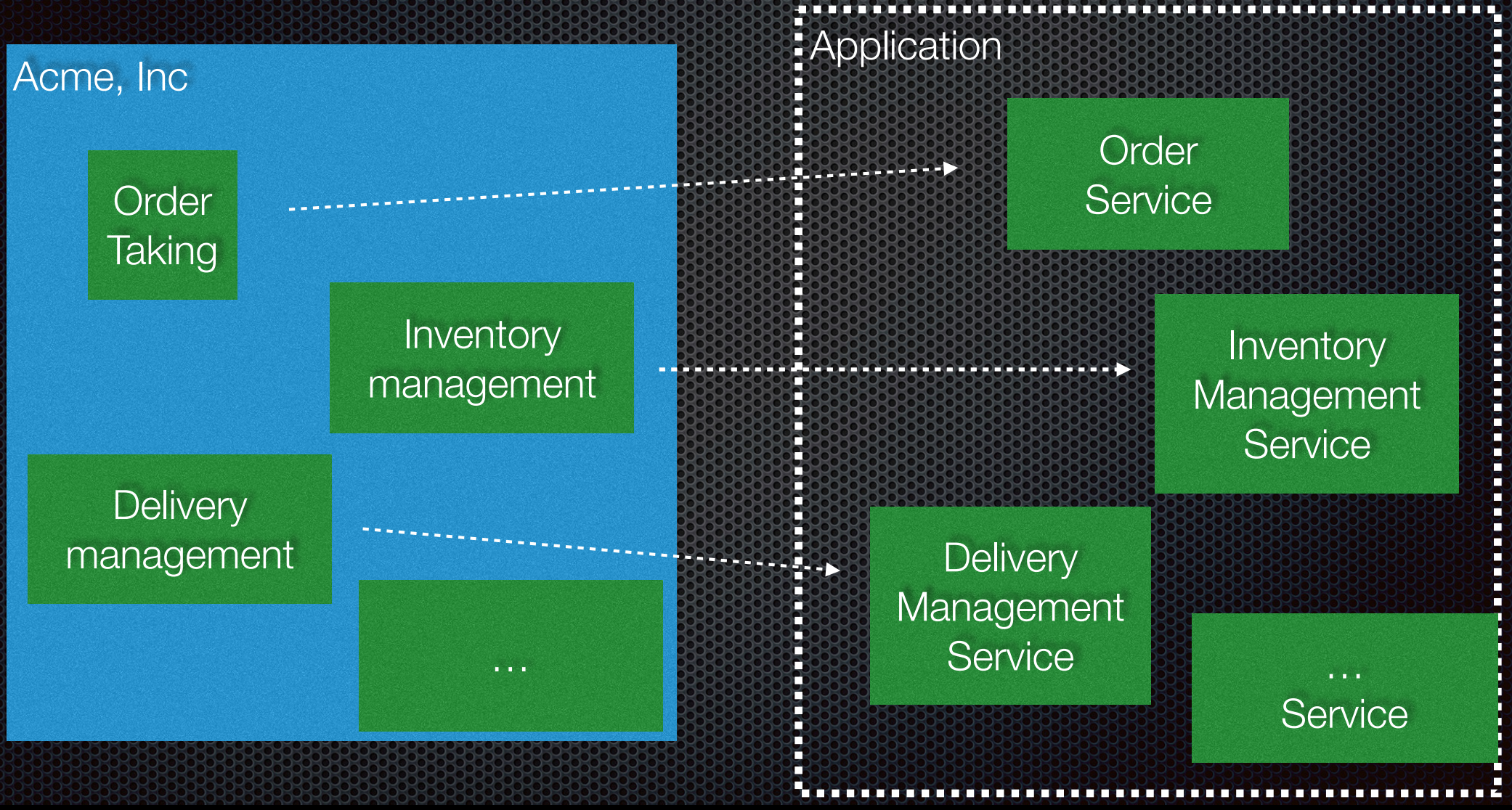
Internal architecture of each service
Each service could use a different technology stack
Pick the best tool for the job
Rapidly evolving

Services vs. Service instances

Service = Implementation view: executable/deployable component

Service Instance = Process view: Process

Service = business capability



Business Capability = something a business does to deliver value

Service size is secondary

~~micro~~service architecture

Service:

Meaningful business functionality

Developed by a **small** team

Minimal lead time/high deployment frequency

Microservices

=

Microservice architecture

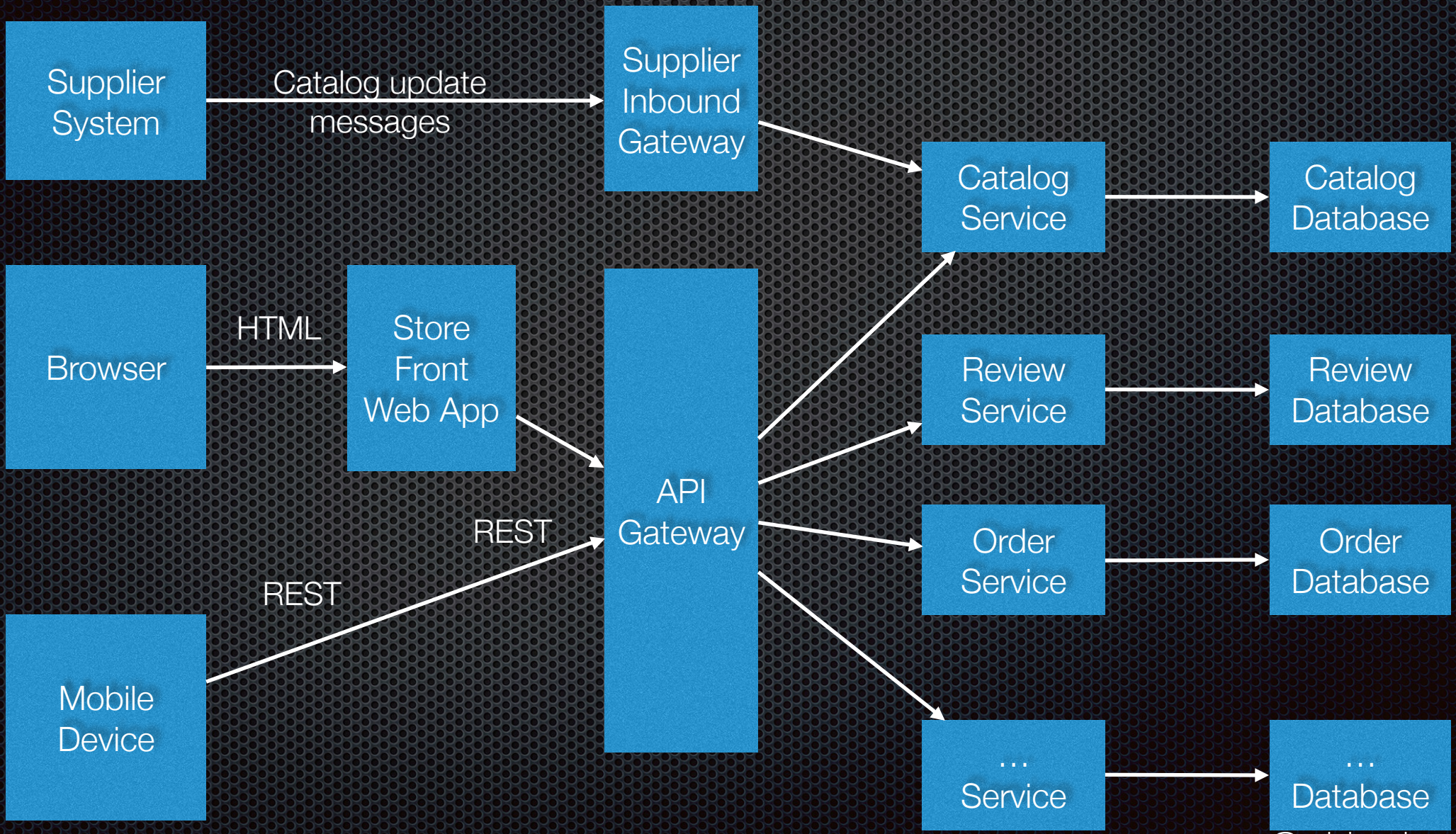
=

Application architecture

⇒

No such thing as a microservice

Example microservice architecture

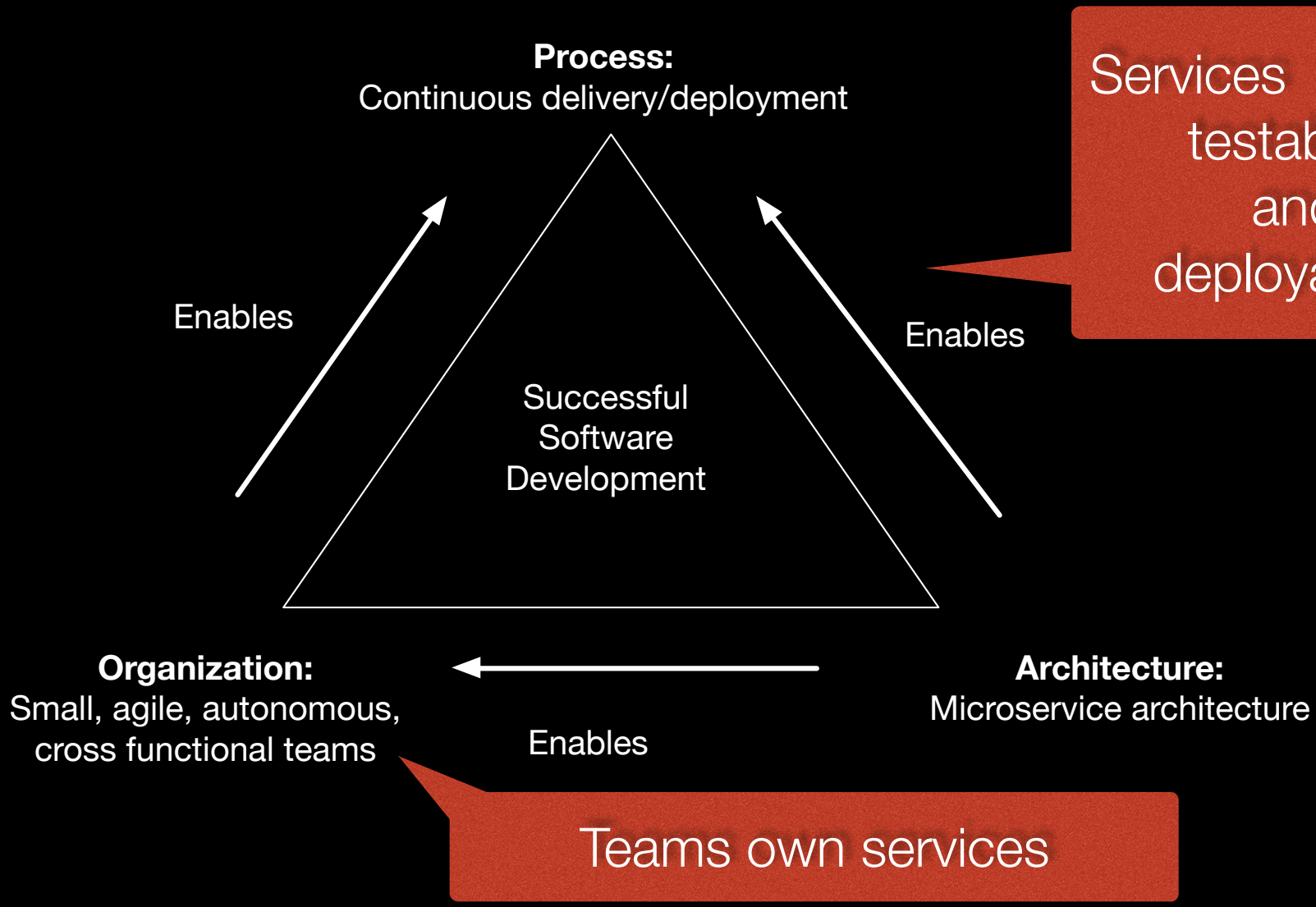


-ilities of a microservice architecture

- Maintainability
- Evolvability
- Testability
- Deployability
- Scalability
- …

https://en.wikipedia.org/wiki/Non-functional_requirement

Microservices enable continuous delivery/deployment



Evolve the technology stack

- Pick a new technology when
  - Writing a new service
  - Making major changes to an existing service
- Let's you experiment and fail safely

Agenda

- A brief refresher on software architecture
- From monolith to microservices
- Microservices != silver bullet
- Managing distributed data

No silver bullets

No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

http://en.wikipedia.org/wiki/Fred_Brooks

Drawbacks of microservices

Complexity

Development: IPC, partial failure, distributed data

Testing: Integration, end to end, …

Deployment

…

Are microservices a good fit for my application?

When using microservices:

How to decompose an application into services?

How to deploy an application's services?

How to handle cross cutting concerns?

Which communication mechanisms to use?

How do external clients communicate with the services?

How does a client discover the network location of a service instance?

How to prevent a network or service failure from cascading to other services?

How to maintain data consistency and implement queries?

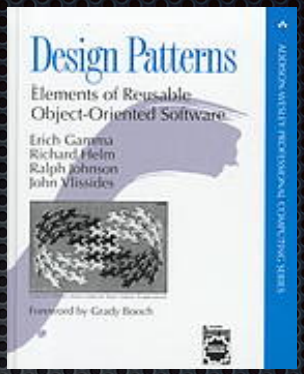
How to understand the behavior of an application and troubleshoot problems?

How to make testing easier?

How to implement a UI screen or page that displays data from multiple services?

It depends!

What's a pattern?



Reusable **solution**
to a **problem**
occurring
in a particular **context**

The structure of a pattern

Name

Context

aka the situation

Problem

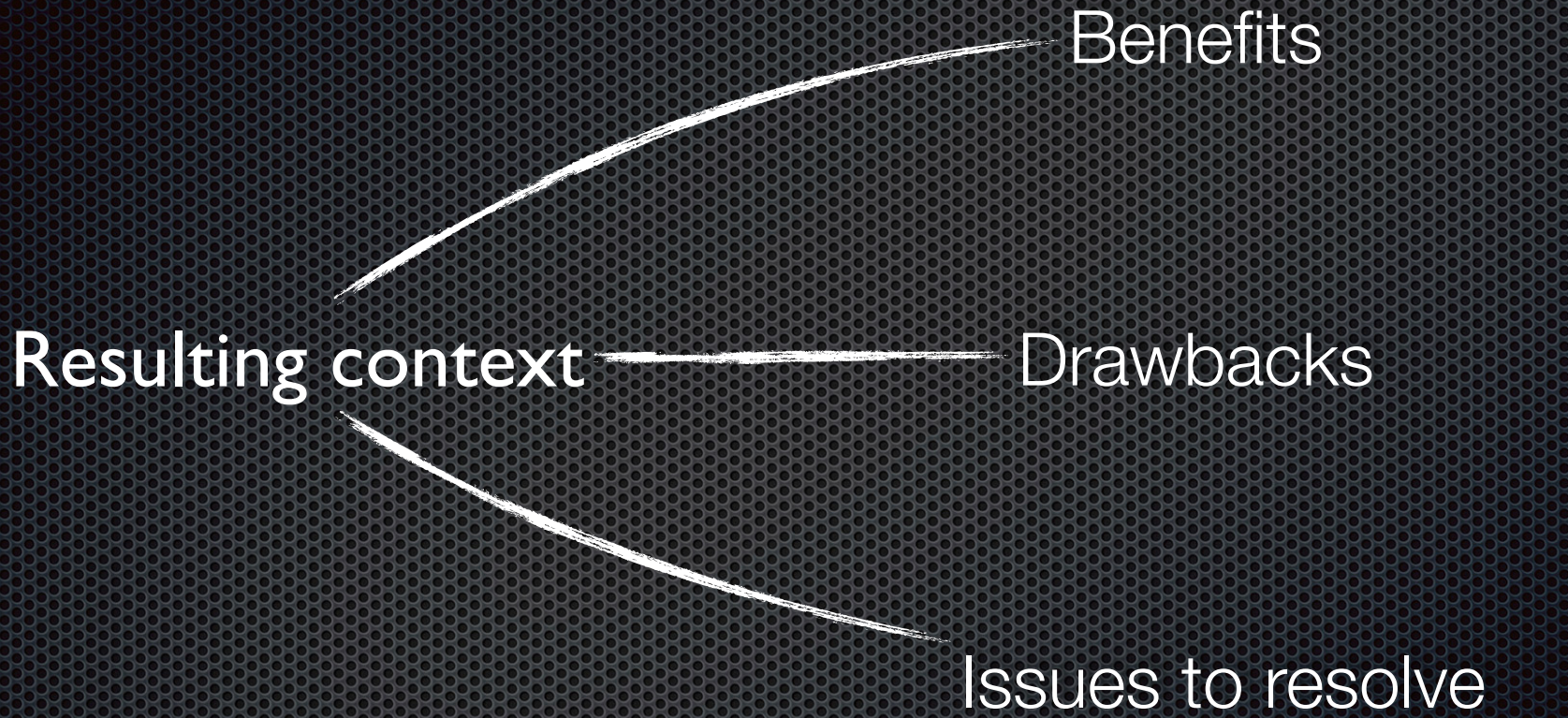
(conflicting) issues etc to address

Forces

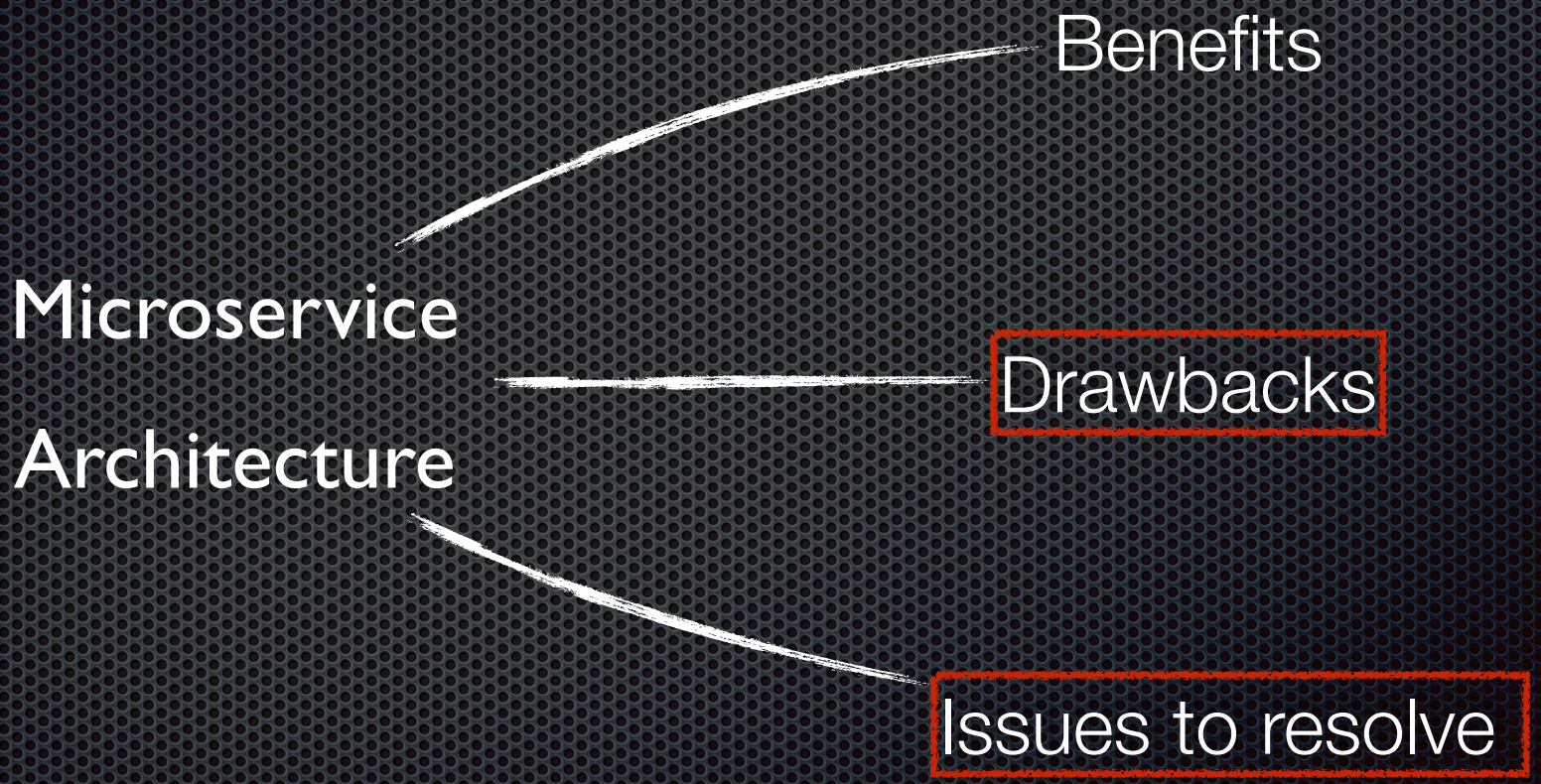
Solution

Resulting context

Related patterns


Benefits
Resulting context
Drawbacks
Issues to resolve

Example resulting context

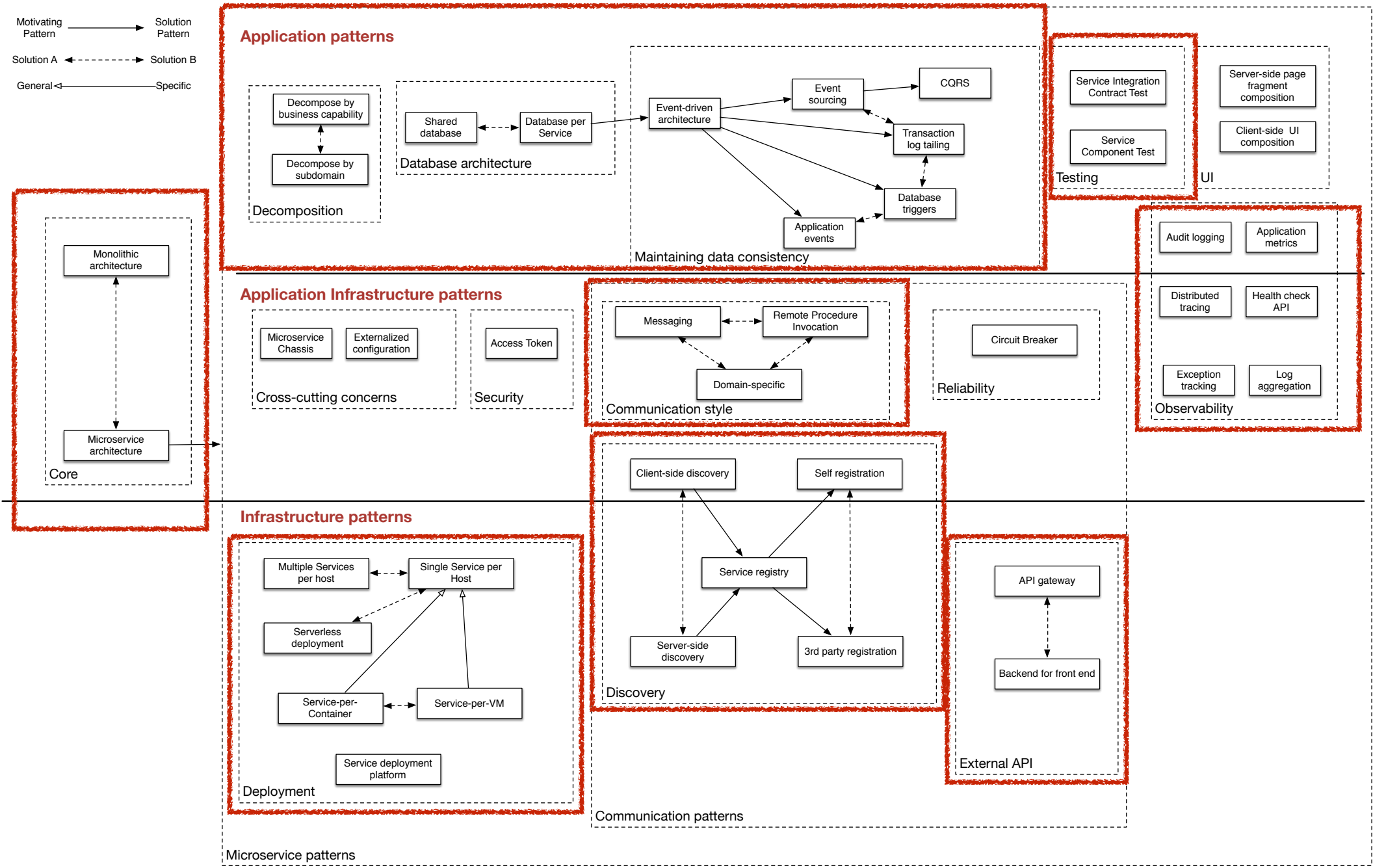


Alternative solutions

Related patterns

Solutions to problems
introduced by this pattern

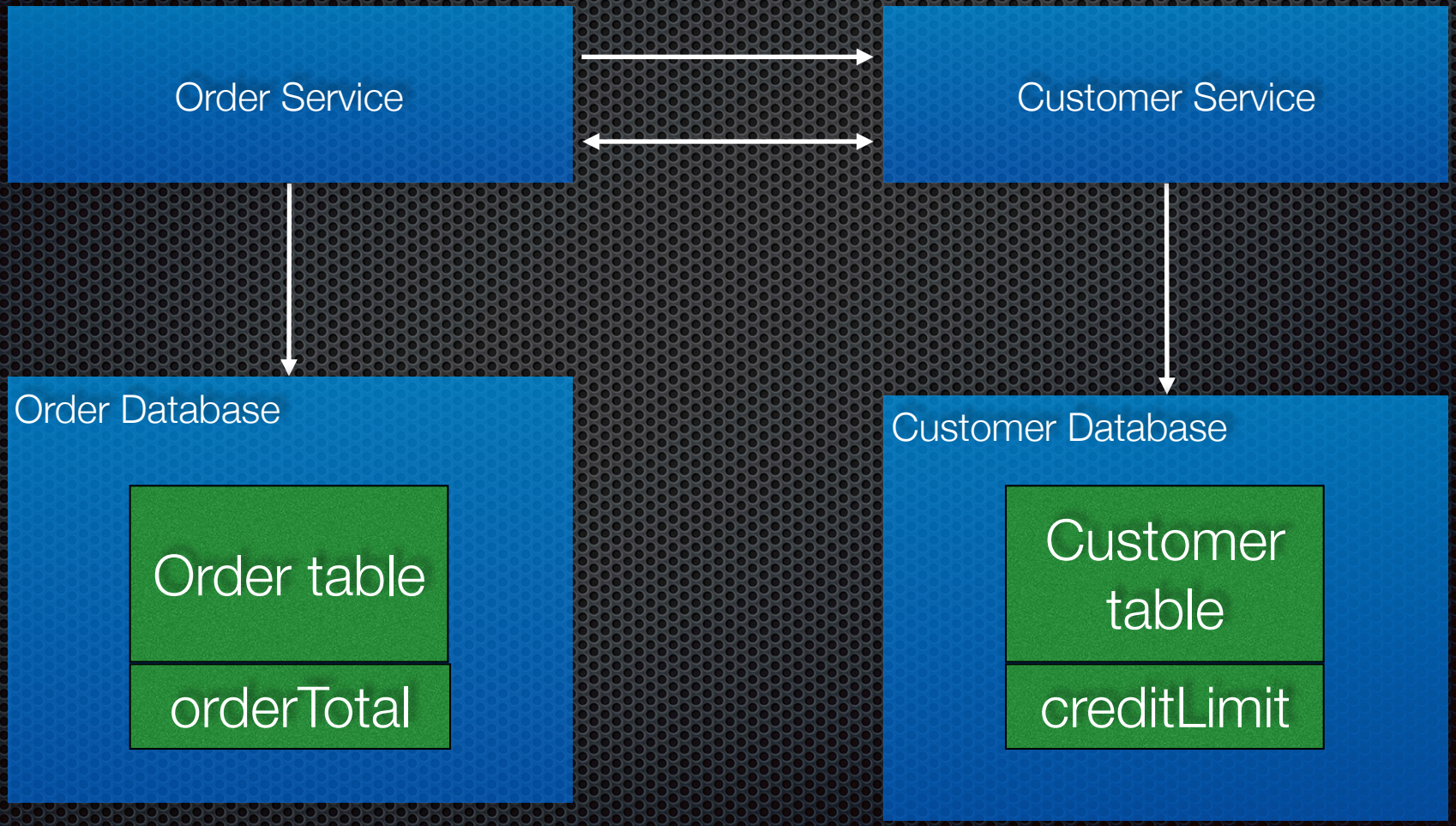
Microservices pattern language: http://microservices.io



Agenda

- A brief refresher on software architecture
- From monolith to microservices
- Microservices != silver bullet
- Managing distributed data

Loose coupling = encapsulated data



How to enforce credit limit?

sum(order.total) <= customer.creditLimit

Cannot use ACID transactions

Distributed transactions

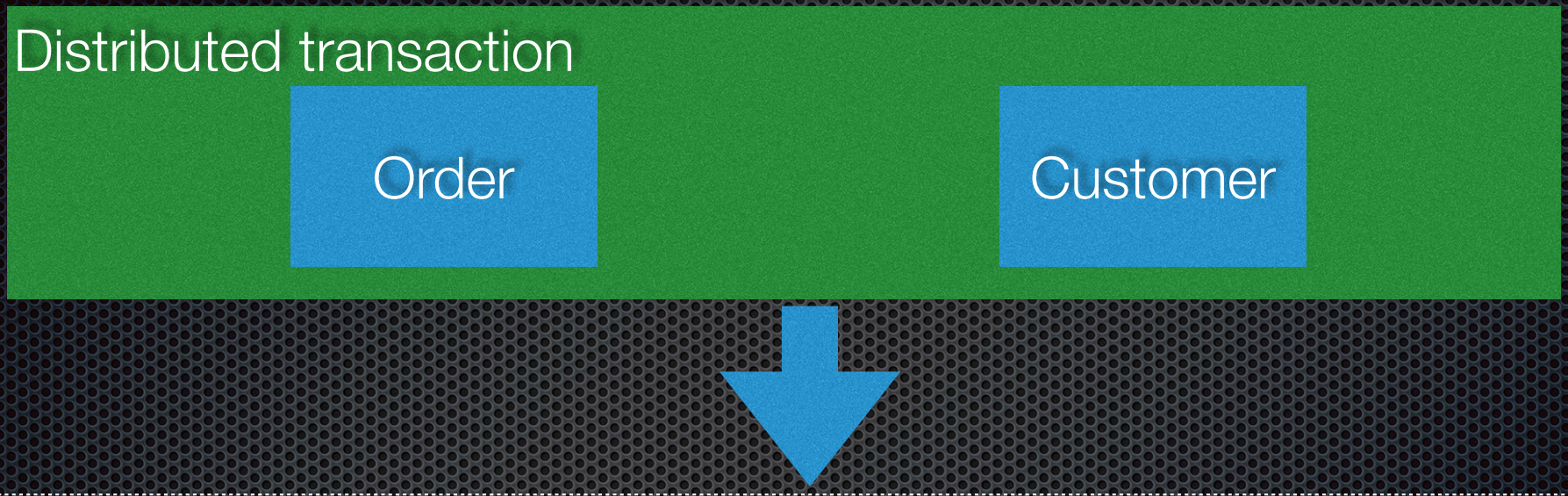
```
BEGIN TRANSACTION
...
SELECT ORDER_TOTAL
 FROM ORDERS WHERE CUSTOMER_ID = ?
...
SELECT CREDIT_LIMIT
FROM CUSTOMERS WHERE CUSTOMER_ID = ?
...
INSERT INTO ORDERS ...
...
COMMIT TRANSACTION
```

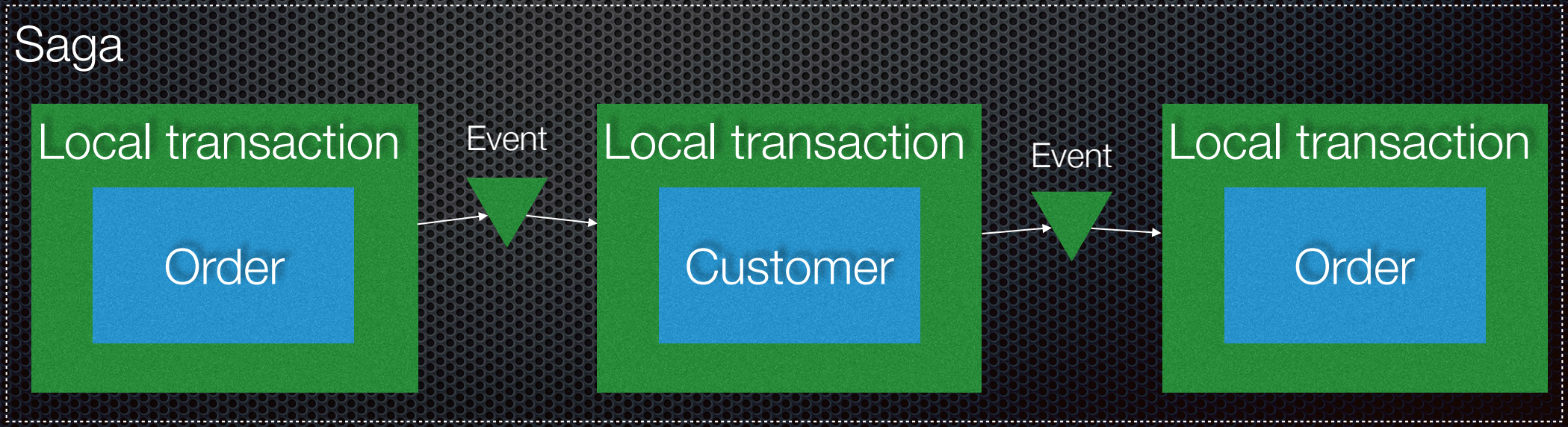
Private to the Order Service

Private to the Customer Service

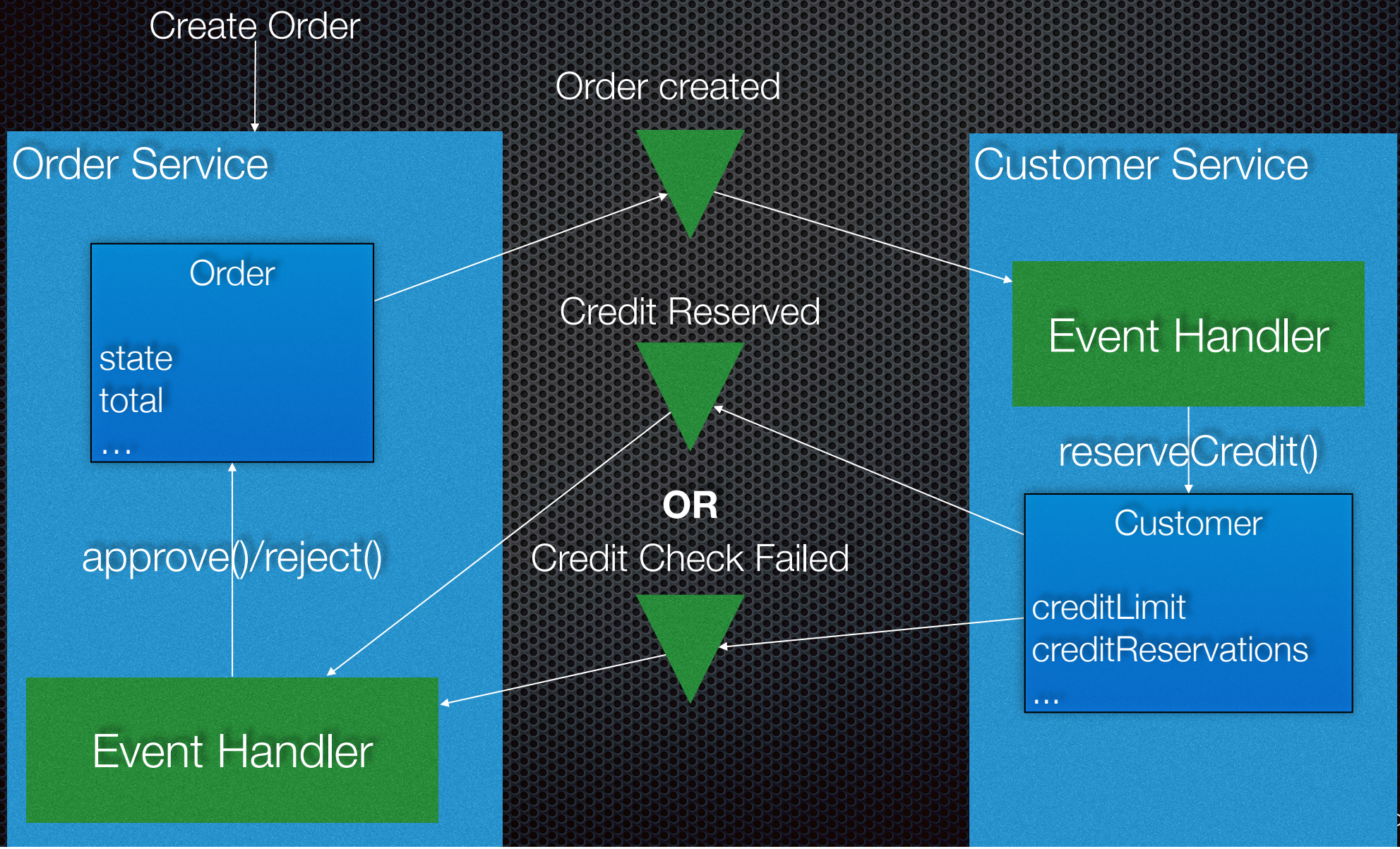
2PC is not an option

Instead: use event-driven sagas





Saga-based, eventually consistent order processing



But how to reliably
update the database
and
publish events
without 2PC?

Event sourcing: event-centric persistence

Event table

Entity id	Entity type	Event id	Event type	Event data
101	Order	901	OrderCreated	…
101	Order	902	OrderApproved	…
101	Order	903	OrderShipped	…

How to find recent, valuable customers?

Customer Service

Order Service

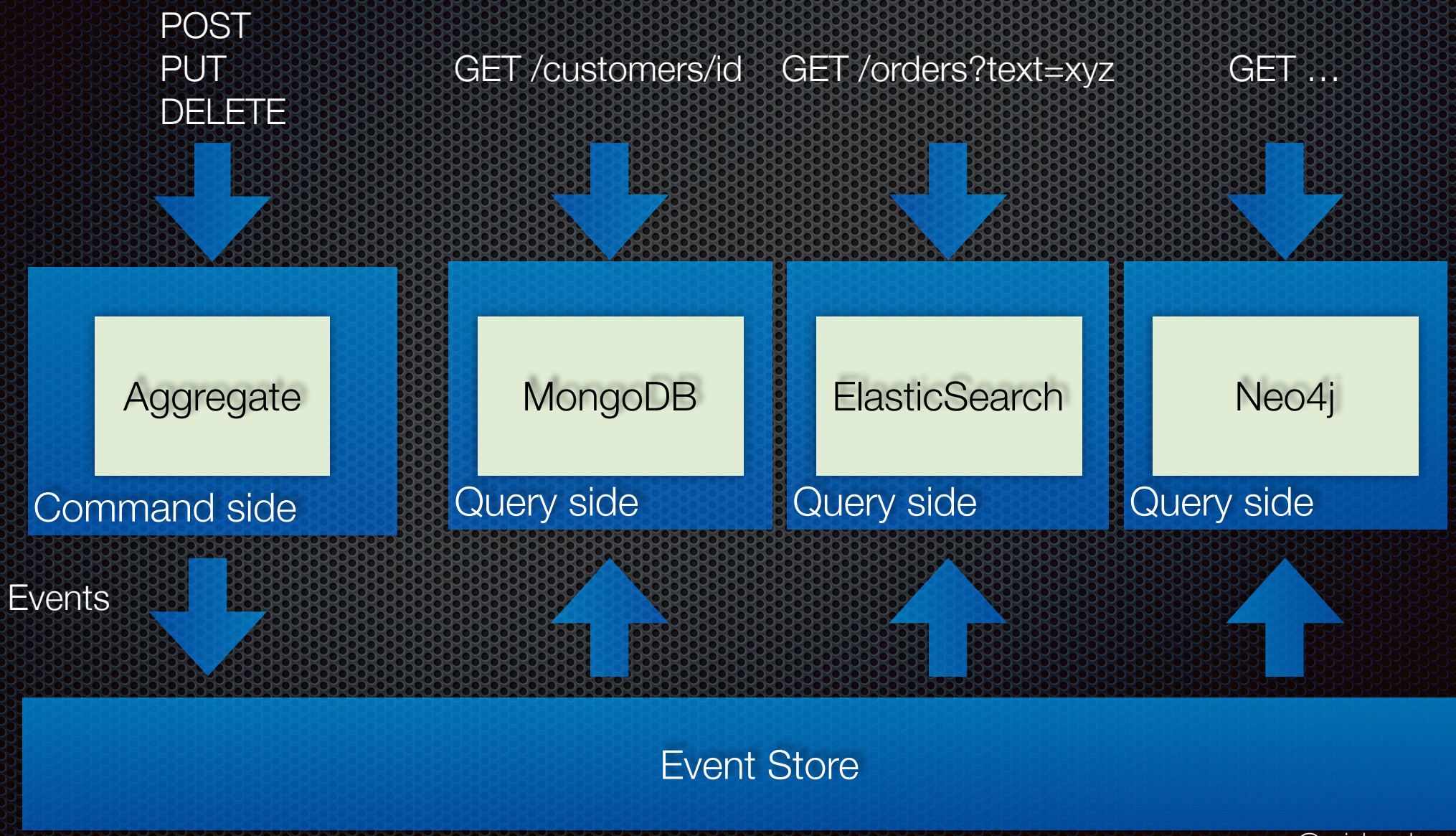
```
SELECT *
FROM CUSTOMER c, ORDER o
WHERE
    c.id = o.ID
      AND o.ORDER_TOTAL > 100000
      AND o.STATE = 'SHIPPED'
      AND c.CREATION_DATE > ?
```

What if event sourcing is used?

.... is no longer easy

Command Query Responsibility Segregation (CQRS)

Queries ⇒ database (type)



Summary

- The goal of architecture is to satisfy non-functional requirements
- Use the appropriate architectural style
  - Small applications ⇒ Monolithic architecture
  - Complex applications ⇒ Microservice architecture
- Use the pattern language to guide your decision making
- Use an event-driven architecture for transactions and queries


@crichardson chris@chrisrichardson.net
Thank you!
http://learnmicroservices.io