

A pattern language for microservices

Chris Richardson

Founder of Eventuate.io

Founder of the original CloudFoundry.com

Author of POJOs in Action

 @crichtson

chris@chrisrichardson.net

<http://microservices.io>

<http://eventuate.io>

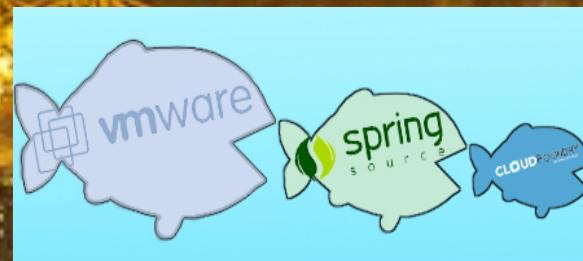
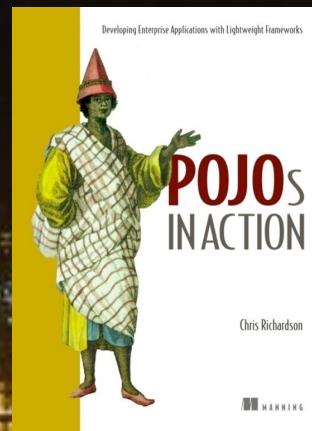
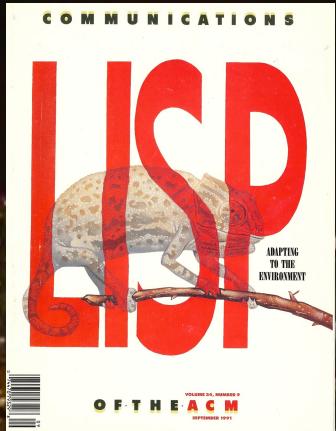
<http://plainoldobjects.com>

Presentation goal

Why patterns and pattern
languages?

A pattern language for
microservices

About Chris



About Chris

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

About Chris

Founder of Eventuate, Inc
Creating a platform that makes it
easy for application developers to
write microservices

(<http://eventuate.io>)



For more information

- <https://github.com/cer/event-sourcing-examples>
- <http://microservices.io>
- <http://plainoldobjects.com/>
- <https://twitter.com/crichardson>
- <http://eventuate.io/>

Agenda

- Why a pattern language for microservices?
- Core patterns
- Deployment patterns
- Communication patterns and more

In 1986...

No Silver Bullet —Essence and Accident in Software Engineering

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

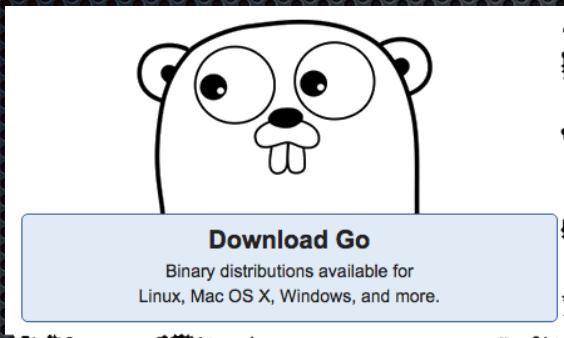
There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

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



Yet almost 30 years later
developers are still
passionately arguing over
“silver bullets”

Popular silver bullets



Functional programming



Suck/Rock Dichotomy

JavaScript vs. Java

Spring vs. Java EE

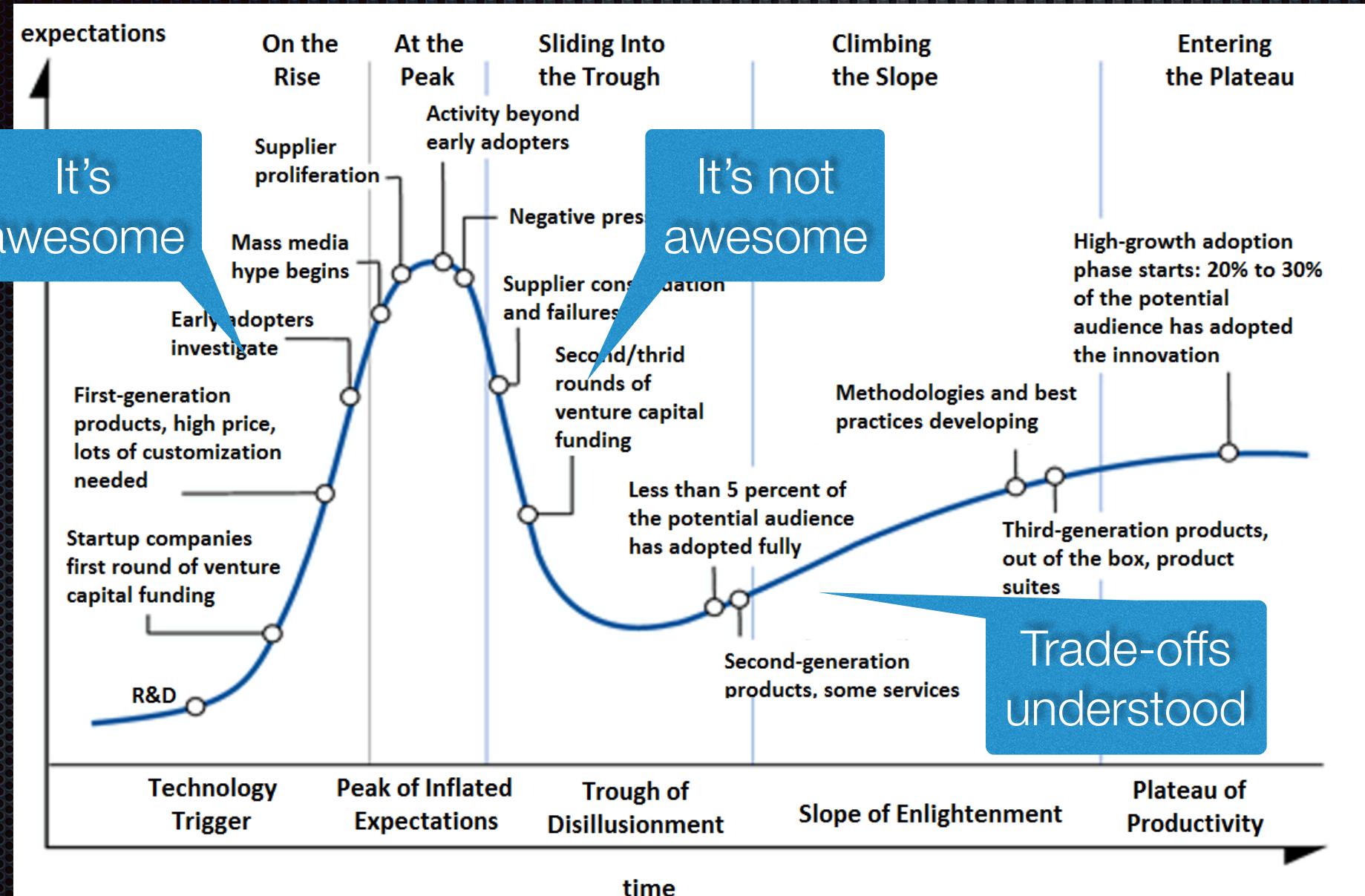
Functional programming vs. Object-oriented

Containers vs. Virtual Machines

<http://nealford.com/memeagora/2009/08/05/suck-rock-dichotomy.html>

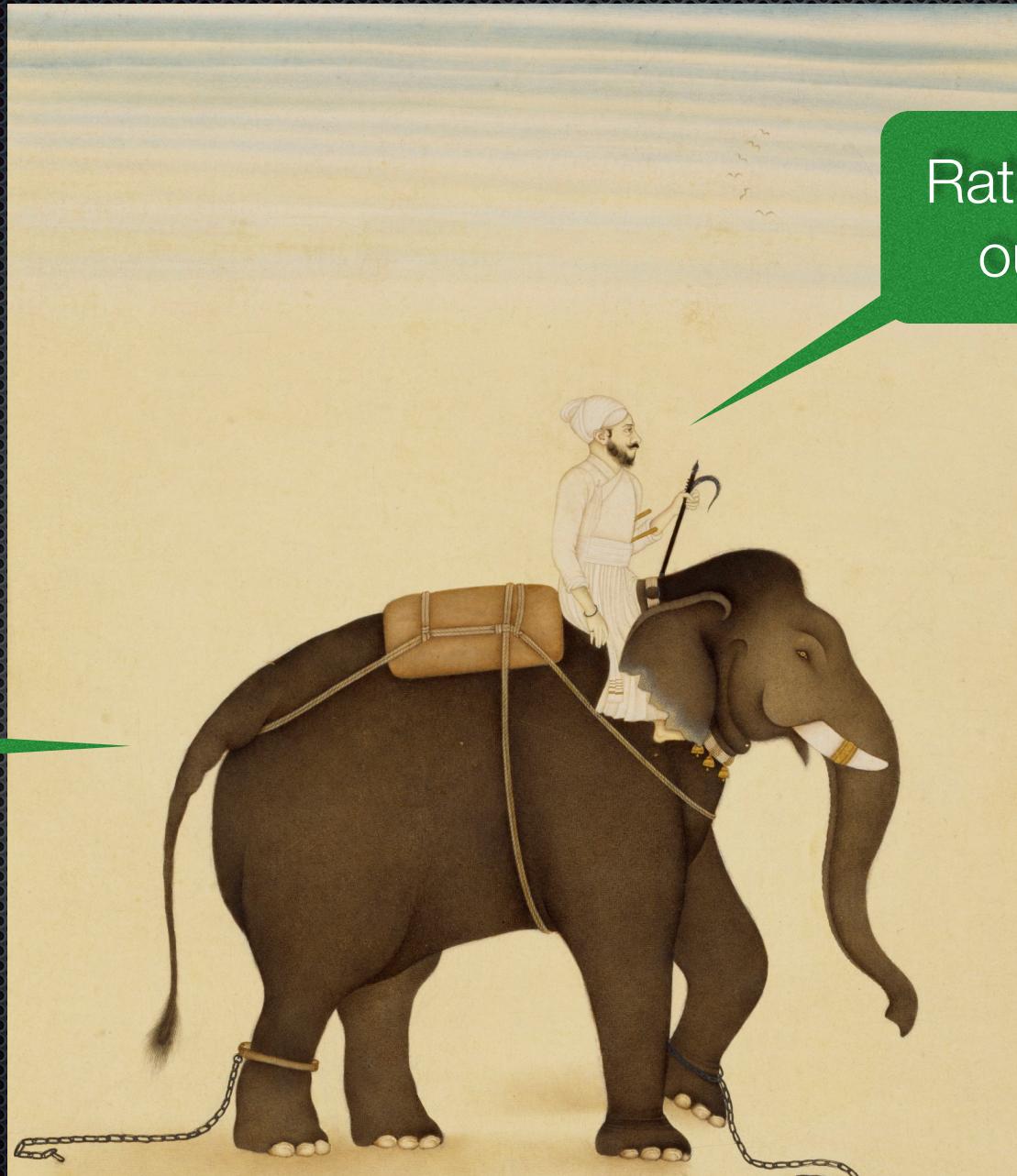
@crichardson

Gartner Hype Cycle



How we make decisions

Decide
using
emotions

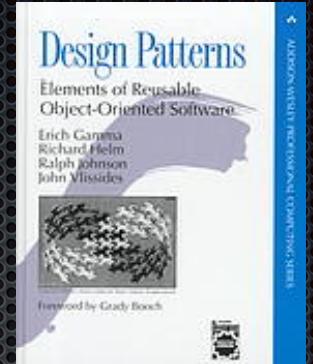


<http://en.wikipedia.org/wiki/Mahout>

@crichardson

We need a better way
to discuss and think
about technology

What's a pattern?



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

The structure of a pattern

=

Great framework for discussing
and thinking about technology

The structure of a pattern

Name

Context

aka the situation

Problem

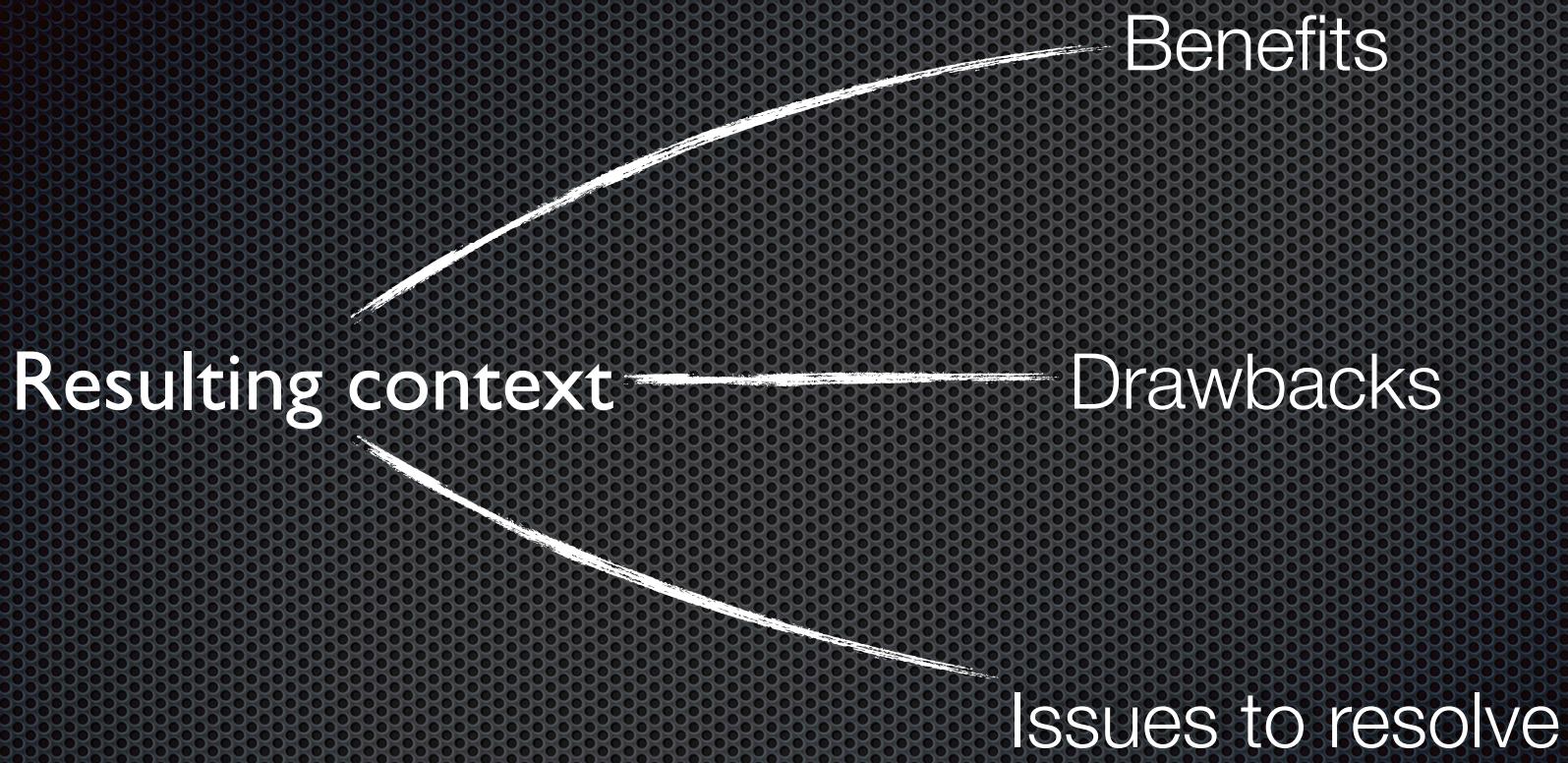
Forces

Solution

Resulting context

Related patterns

(conflicting) issues
etc to address



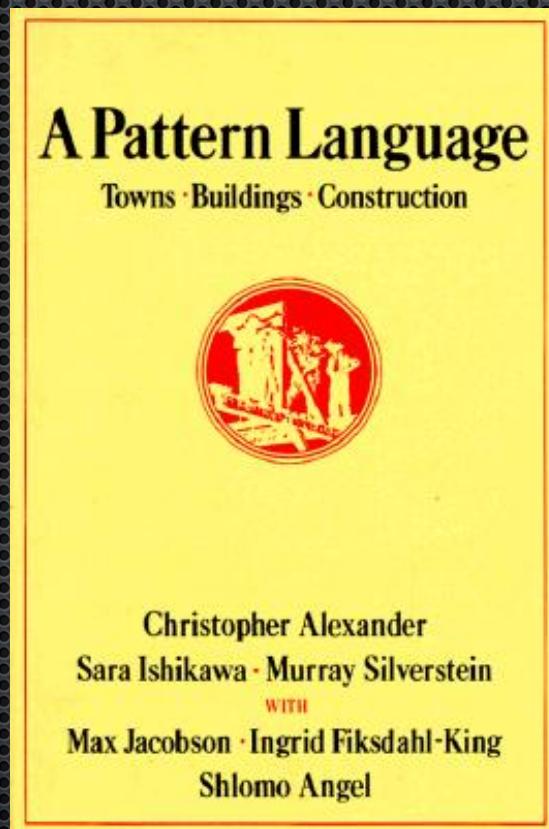
Related patterns

Alternative solutions

Solutions to problems
introduced by this pattern

Pattern language

- A collection of related patterns that solve problems in a particular domain
- Relationships
 - Pattern A results in a context that has a problem solved by Pattern B
 - Patterns A and B solve the same problem
 - Pattern A is a specialization of pattern B



Access to Water

Promenade

Local townhall

Intimacy gradient

Light on two sides

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

Meta-pattern

Problem: How to talk/reason about technology?

Context: Emotional software development culture

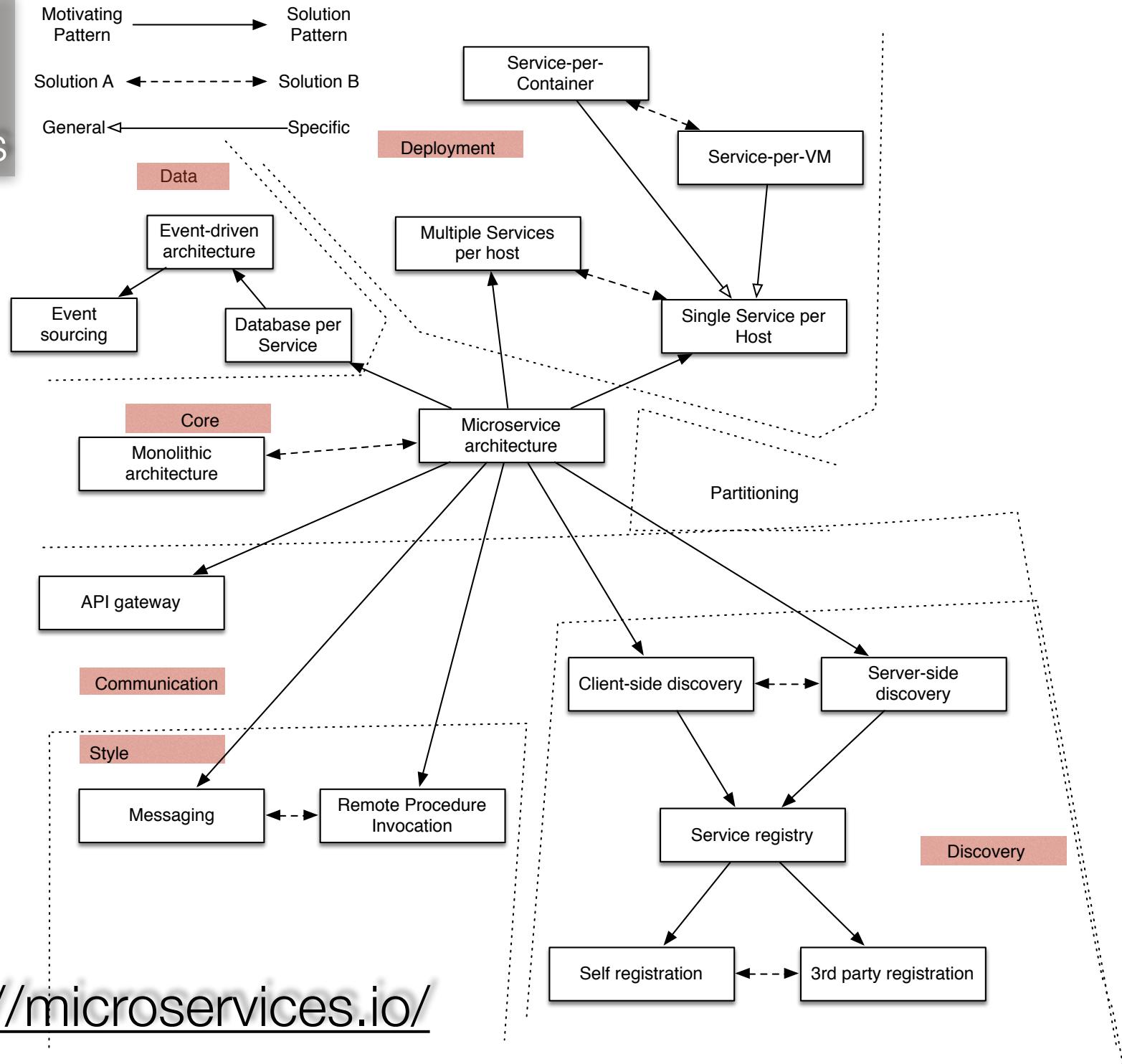
Solution: Use the pattern format

Benefit: More objective

Drawback: Less exciting

Related patterns: It's awesome!

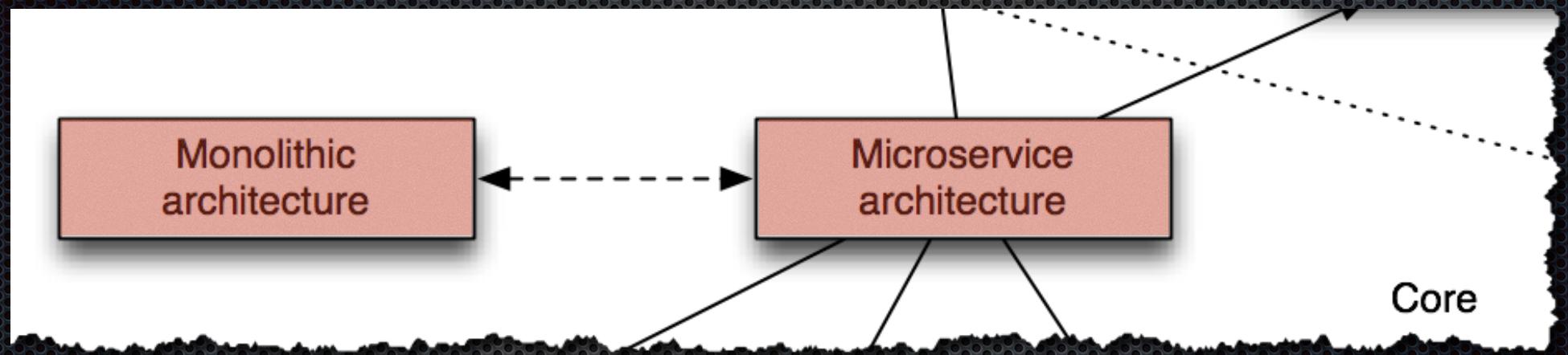
Work in progress



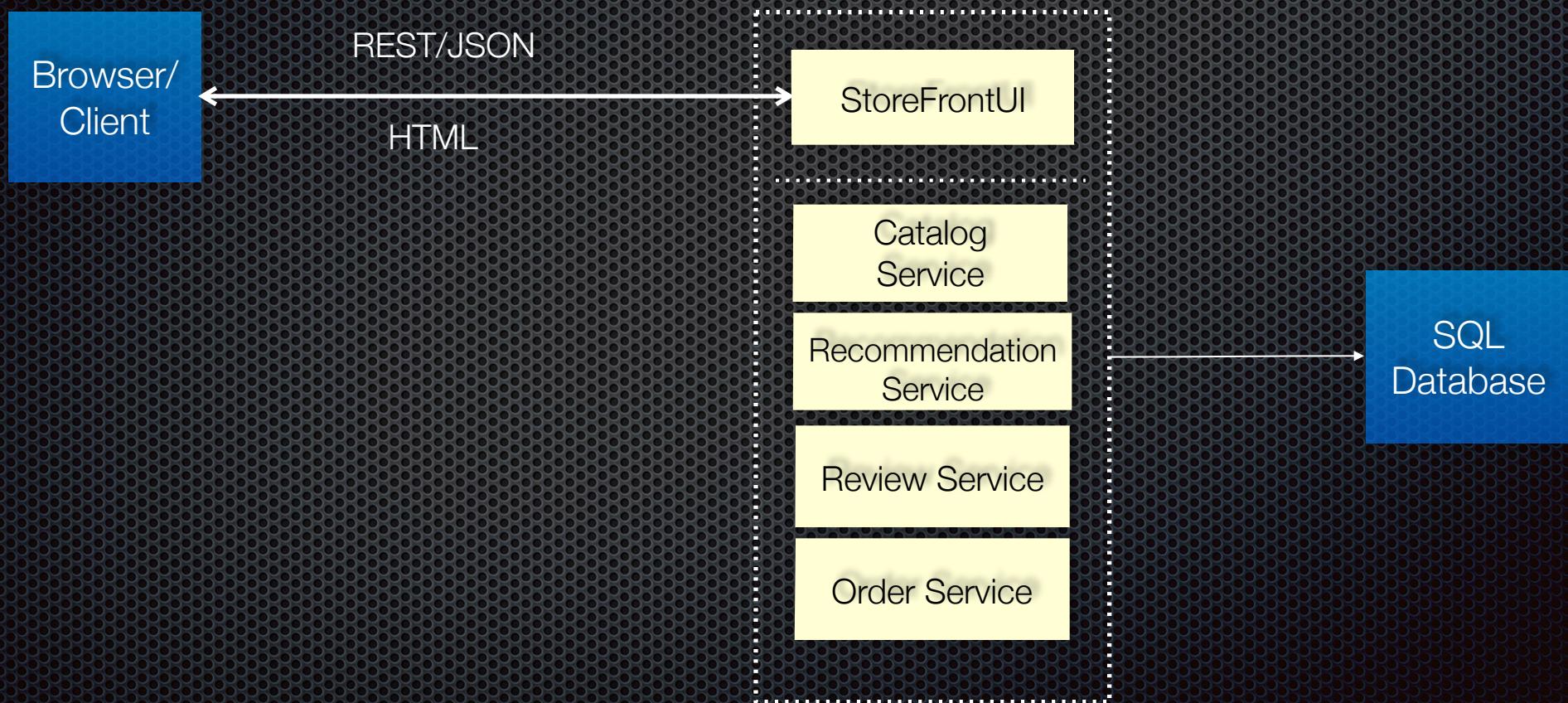
<http://microservices.io/>

Agenda

- Why a pattern language for microservices?
- Core patterns
- Deployment patterns
- Communication patterns and more



Let's imagine you are building an online store



Problem: what's the deployment architecture?

Forces

Businesses must innovate
faster

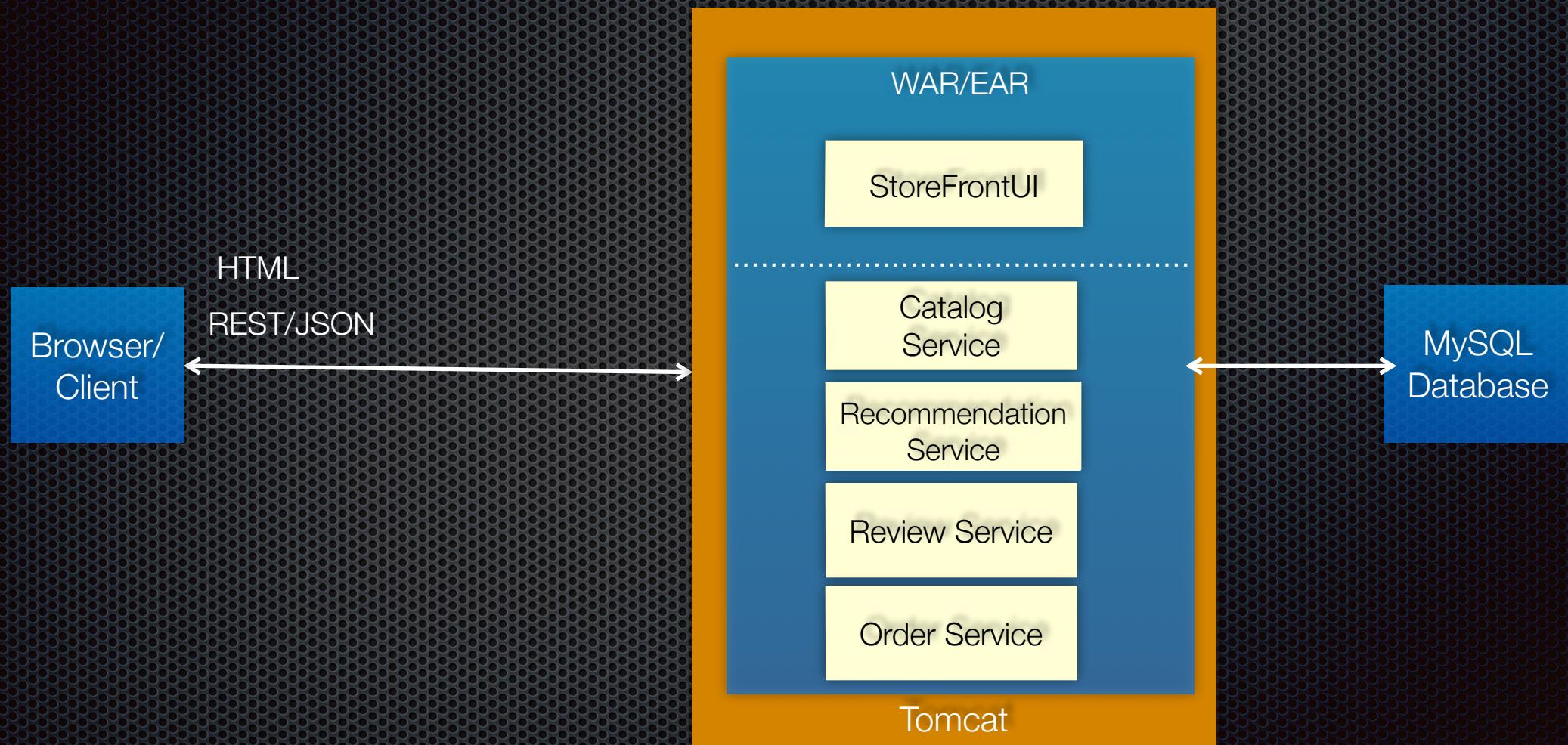


Develop more complex,
higher-quality software faster

Forces

- Developers must be productive
- The code must be easy to understand and modify
- Continuous deployment of applications
- Developers must leverage emerging technologies (frameworks, programming languages, etc)
- Deploy multiple application instances for scalability and availability

Pattern: Monolithic architecture



Examples everywhere



Simple to

Develop
Test
Deploy
Scale

Successful applications
have a habit of growing



Big, complex, monolithic
applications

Agile development and deployment
becomes impossible

Intimidates developers



Obstacle to frequent deployments

- Need to redeploy everything to change one component
- Interrupts long running background (e.g. Quartz) jobs
- Increases risk of failure

Eggs in
one basket

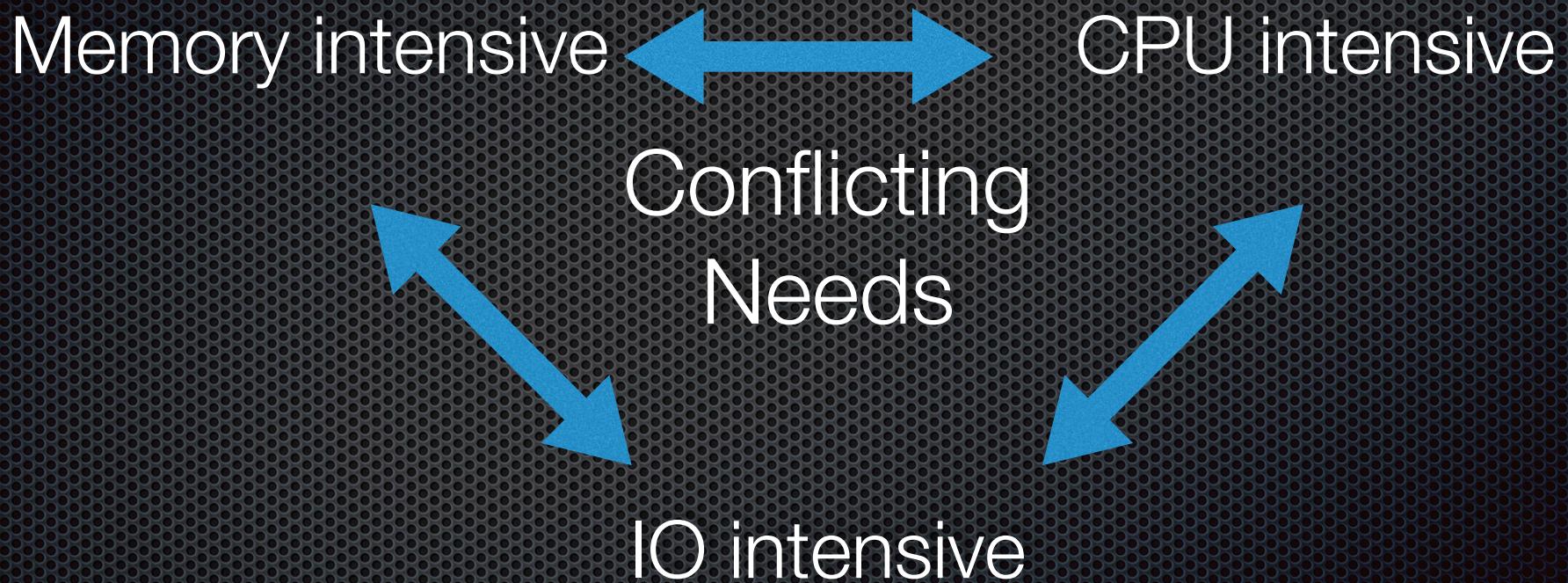


Fear of change



- Updates will happen less often - really long QA cycles
- e.g. Makes A/B testing UI really difficult

Scaling the application can be challenging



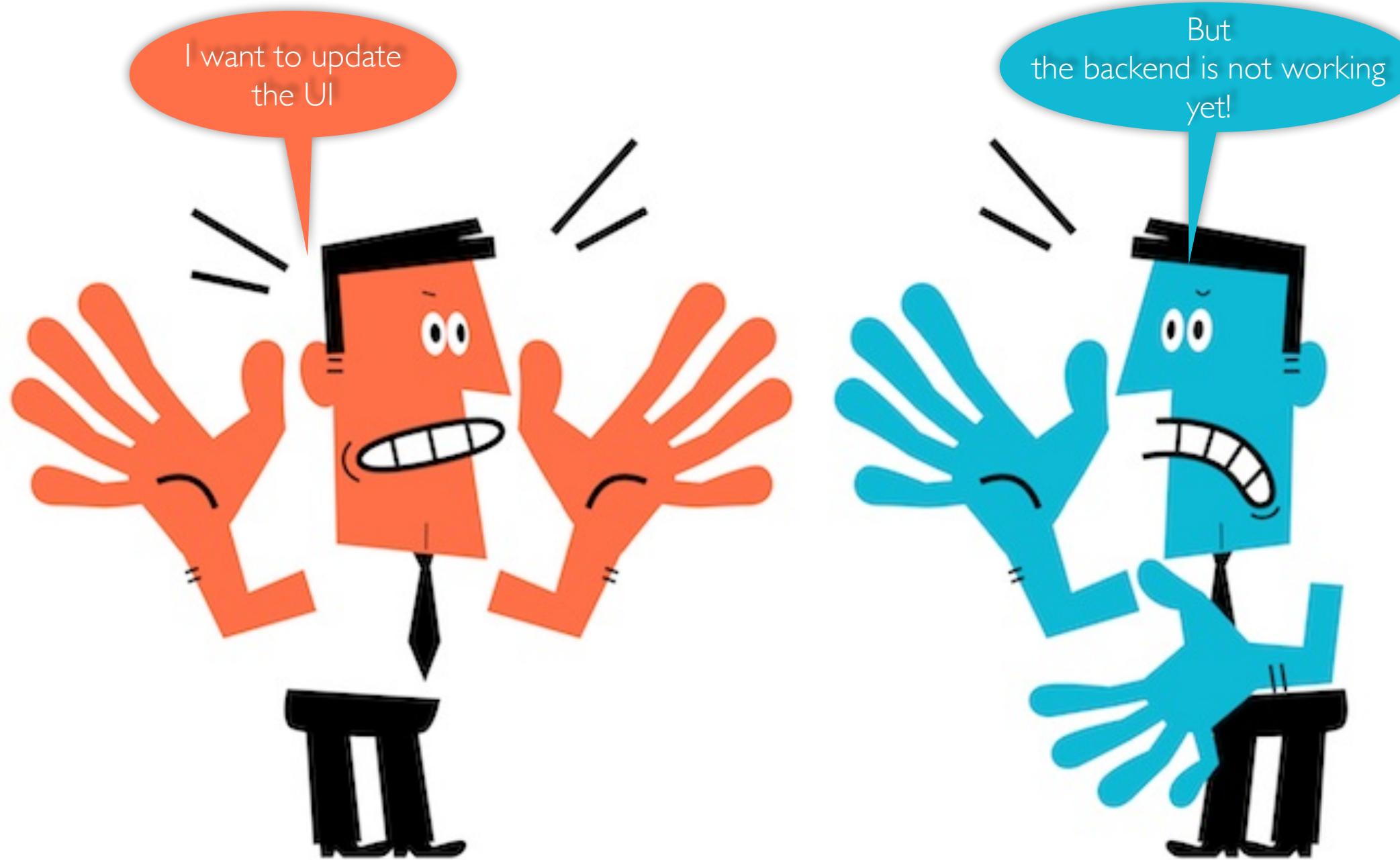
Overloads your IDE and container



Slows down development

@crichardson

Requires lots of coordination

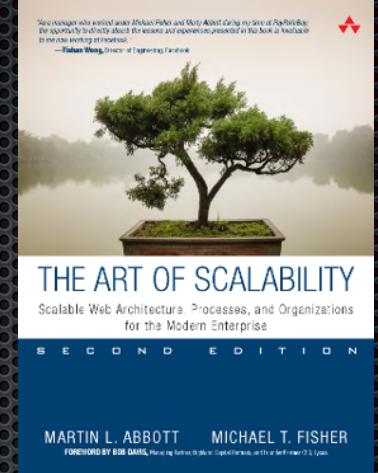
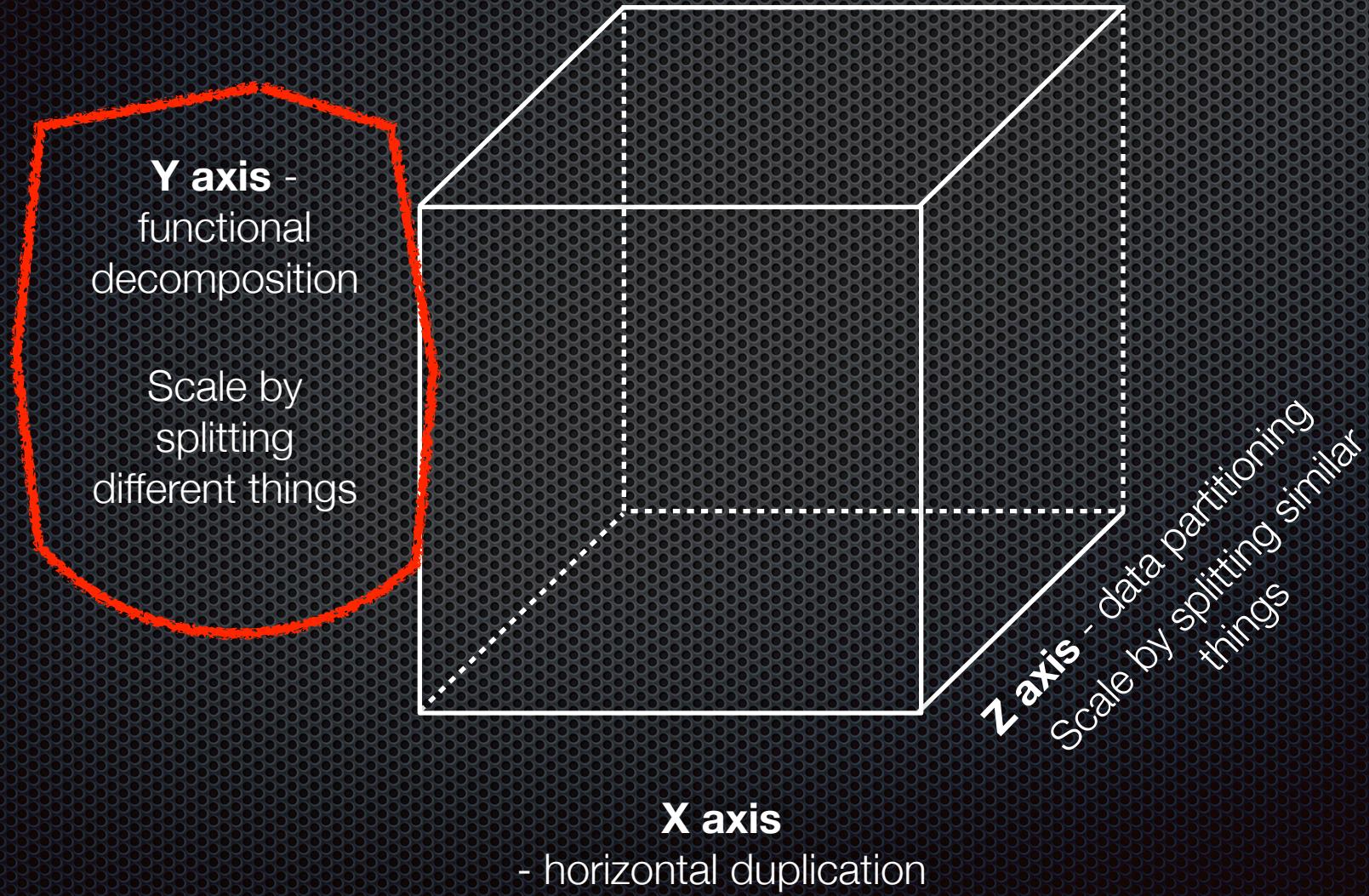


Requires long-term
commitment to a
technology stack

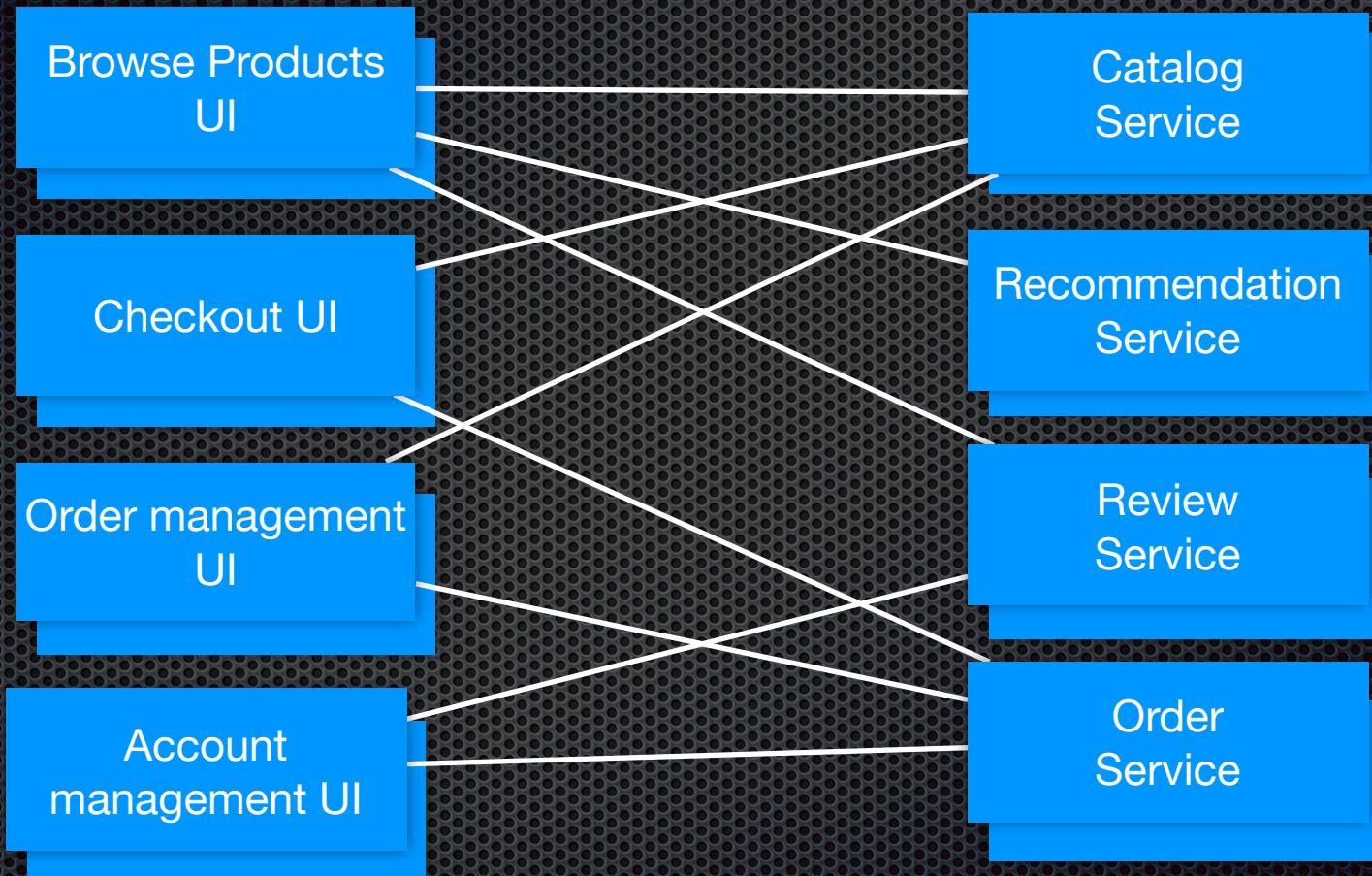


Pattern: Microservice architecture

Apply functional decomposition



Microservice architecture



Apply X-axis and Z-axis scaling
to each service independently

Goal of partitioning:

Most changes affect only a single service



Develop and deploy independently
in parallel with other changes

Goal of partitioning:

Separate components with
conflicting resource
requirements



Scale more easily

Examples of microservices



<http://techblog.netflix.com/>

~600 services



<http://highscalability.com/amazon-architecture>

100-150 services to build a page



<http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf>

<http://queue.acm.org/detail.cfm?id=1394128>

Examples of microservices



http://bit.ly/msexample_soundcloud



http://bit.ly/msexample_yelp



http://bit.ly/msexample_uber

Benefits

- Smaller, simpler apps
 - Easier to understand and develop
 - Less jar/classpath hell - who needs OSGI?
 - Faster to build and deploy
- Scales development: develop, deploy and scale each service independently
- Improves fault isolation
- Easier scaling
- Eliminates long-term commitment to a single technology stack
 - System level architecture vs. service level architecture
 - Easily and safely experiment with new technologies

Drawbacks

Complexity

Drawbacks

- ✖ Complexity of developing a distributed system
 - ✖ Implementing inter-process communication
 - ✖ Handling partial failures
- ✖ Implementing business transactions that span multiple databases (without 2PC)
- ✖ Complexity of testing a distributed system
- ✖ Complexity of deploying and operating a distributed system
- ✖ Managing the development and deployment of features that span multiple services

Fortunately solutions exists

The benefits typically
outweigh the drawbacks
for
large, complex applications

Issues to address

- ❖ **How to deploy the services?**
- ❖ **How do the services communicate?**
- ❖ **How do clients of the application communicate with the services?**
- ❖ How to partition the system into services?
- ❖ How to deal with distributed data management problems?
- ❖

Agenda

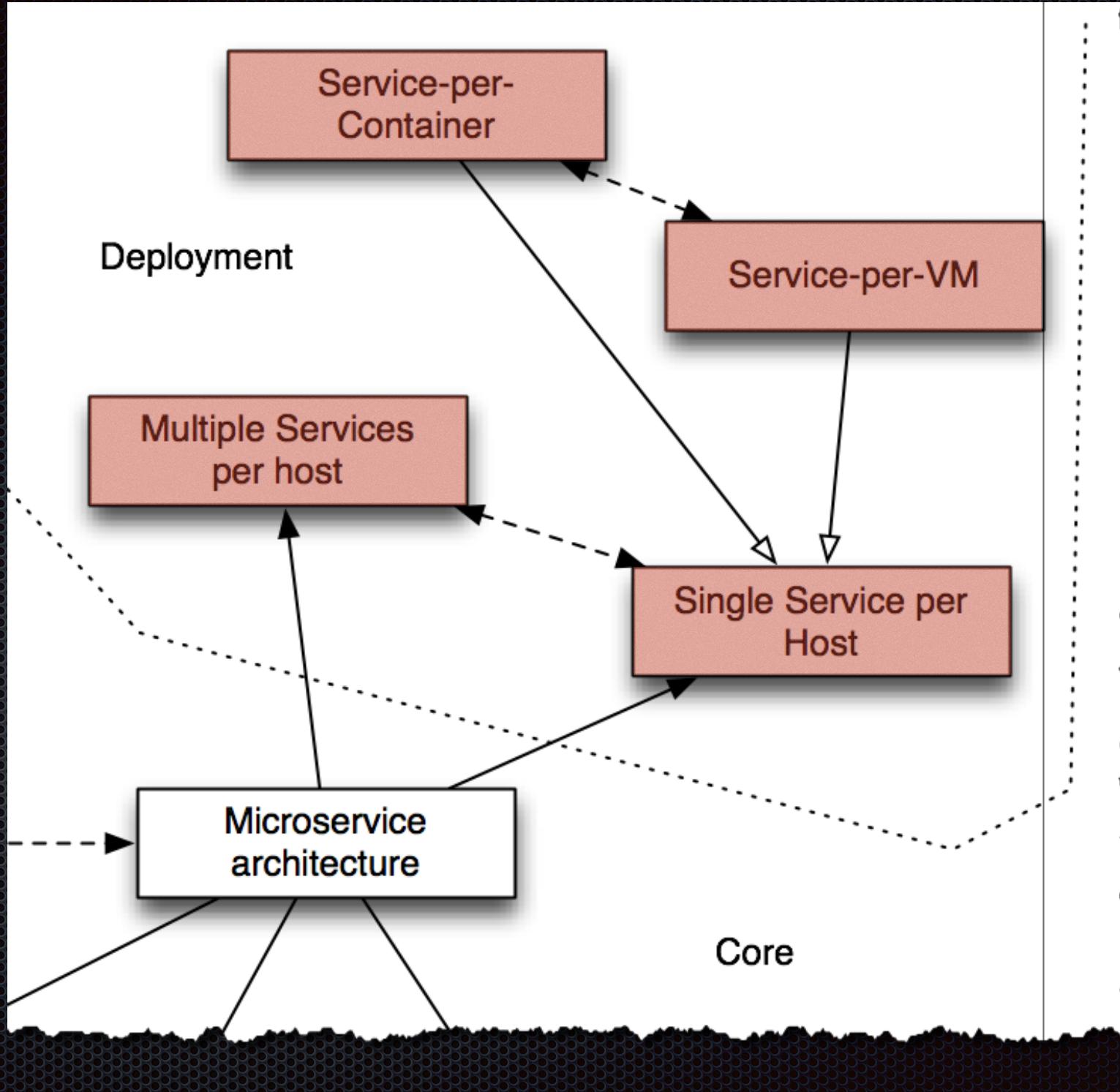
- Why a pattern language for microservices?
- Core patterns
- Deployment patterns
- Communication patterns and more

We have applied the
microservices pattern:

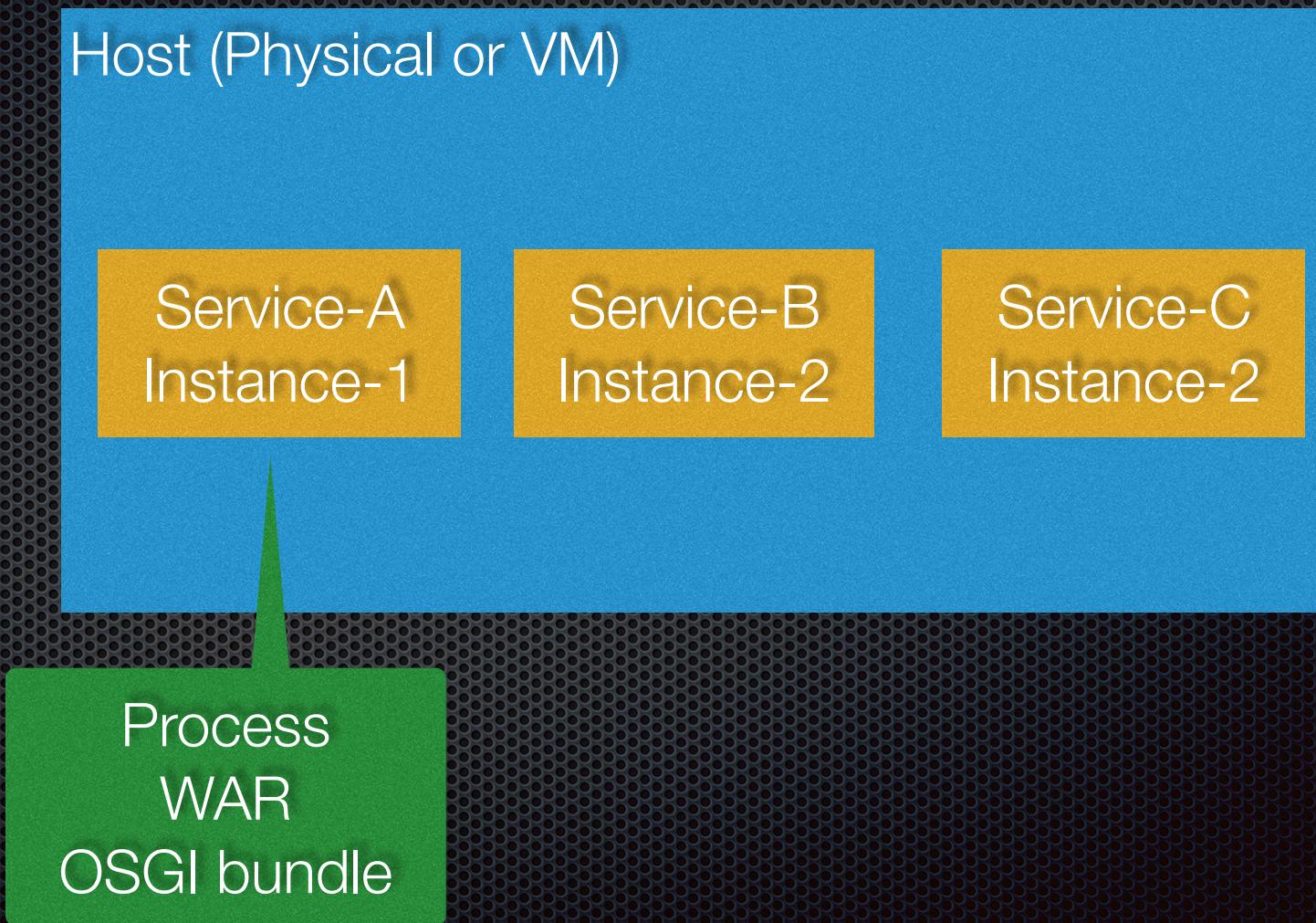
How to deploy the services?

Forces

- Services are written using a variety of languages, frameworks, and framework versions
- Each service consists of multiple service instances for throughput and availability
- Building and deploying a service must be fast
- Service must be deployed and scaled independently
- Service instances need to be isolated
- Resources consumed by a service must be constrained
- Deployment must be reliable
- Deployment must be cost-effective



Pattern: Multiple service instances per host



Benefits and drawbacks

Benefits

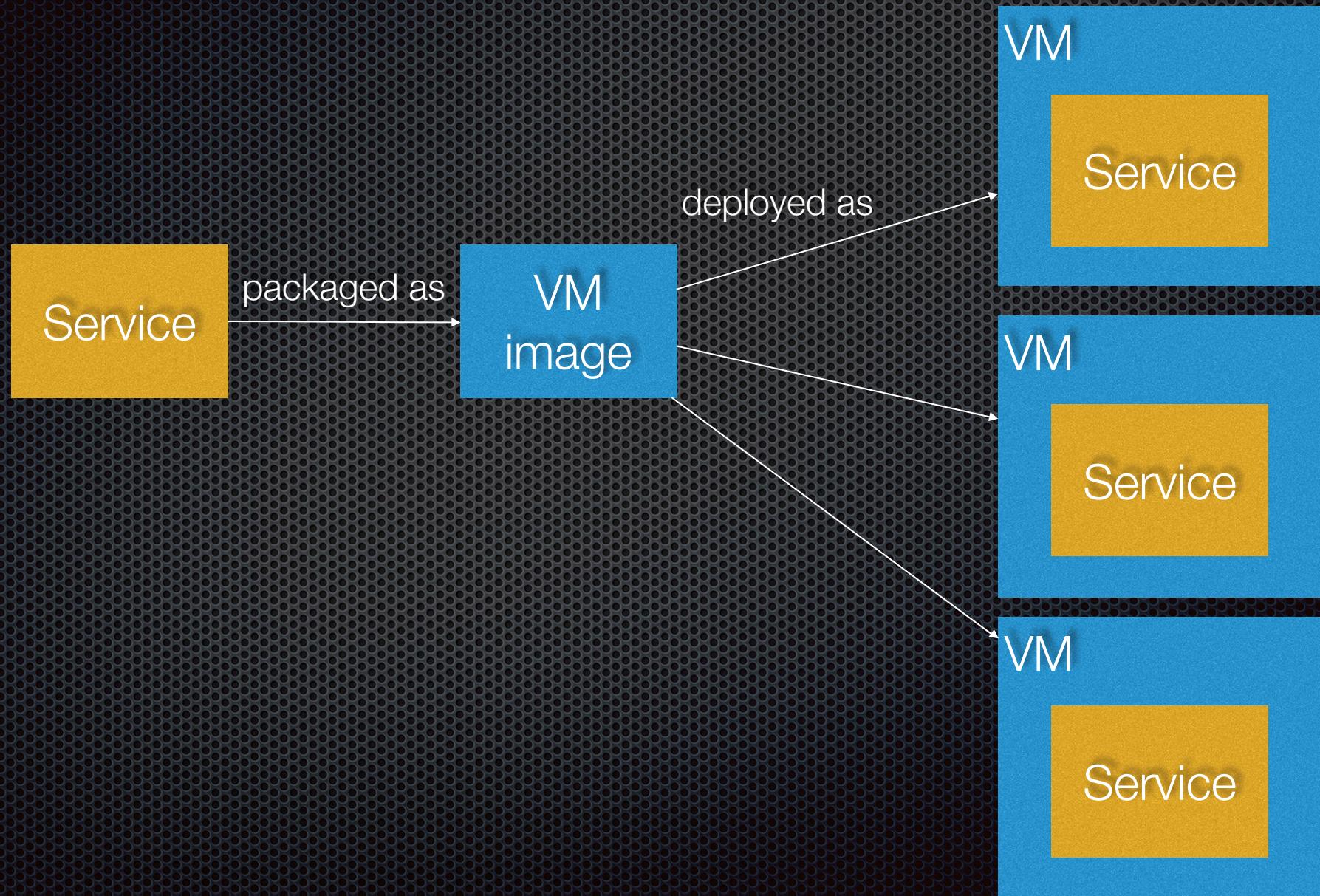
- ✖ Efficient resource utilization
- ✖ Fast deployment

Drawbacks

- ✖ Poor/Terrible isolation
- ✖ Poor visibility (with WAR/OSGI deployment)
- ✖ Difficult to limit resource utilization
- ✖ Risk of dependency version conflicts
- ✖ Poor encapsulation of implementation technology

Pattern: Service instance per host

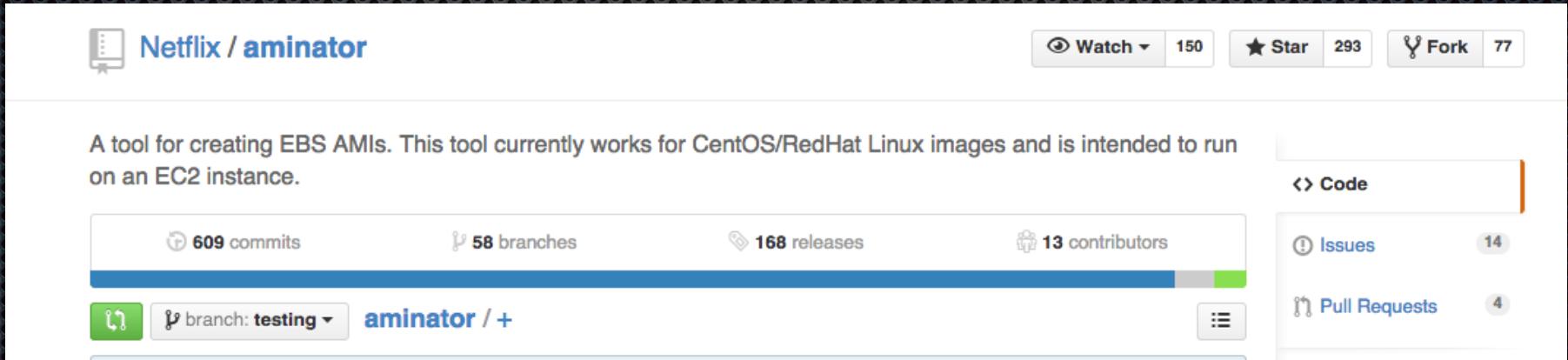
Pattern: Service per VM host



Example



<http://techblog.netflix.com/>
~600 services



A screenshot of a GitHub repository page for "Netflix / aminator". The page shows the repository's description: "A tool for creating EBS AMIs. This tool currently works for CentOS/RedHat Linux images and is intended to run on an EC2 instance." Below the description are statistics: 609 commits, 58 branches, 168 releases, and 13 contributors. A dropdown menu shows the current branch is "testing". On the right side, there are buttons for "Code", "Issues" (14), and "Pull Requests" (4). There are also "Watch" (150), "Star" (293), and "Fork" (77) buttons at the top right.

packer.io is a great tool



@crichton

Benefits and drawbacks

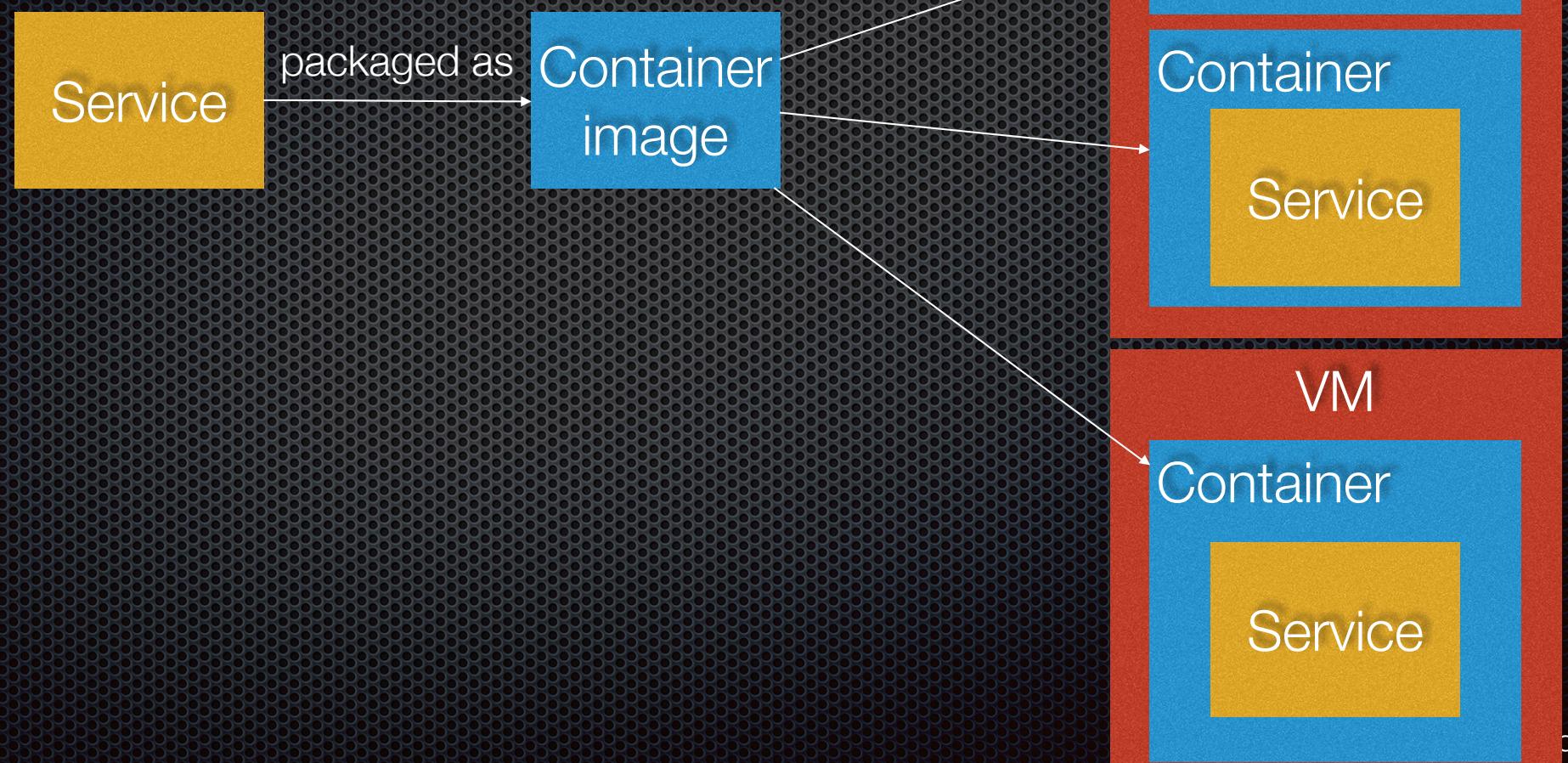
Benefits

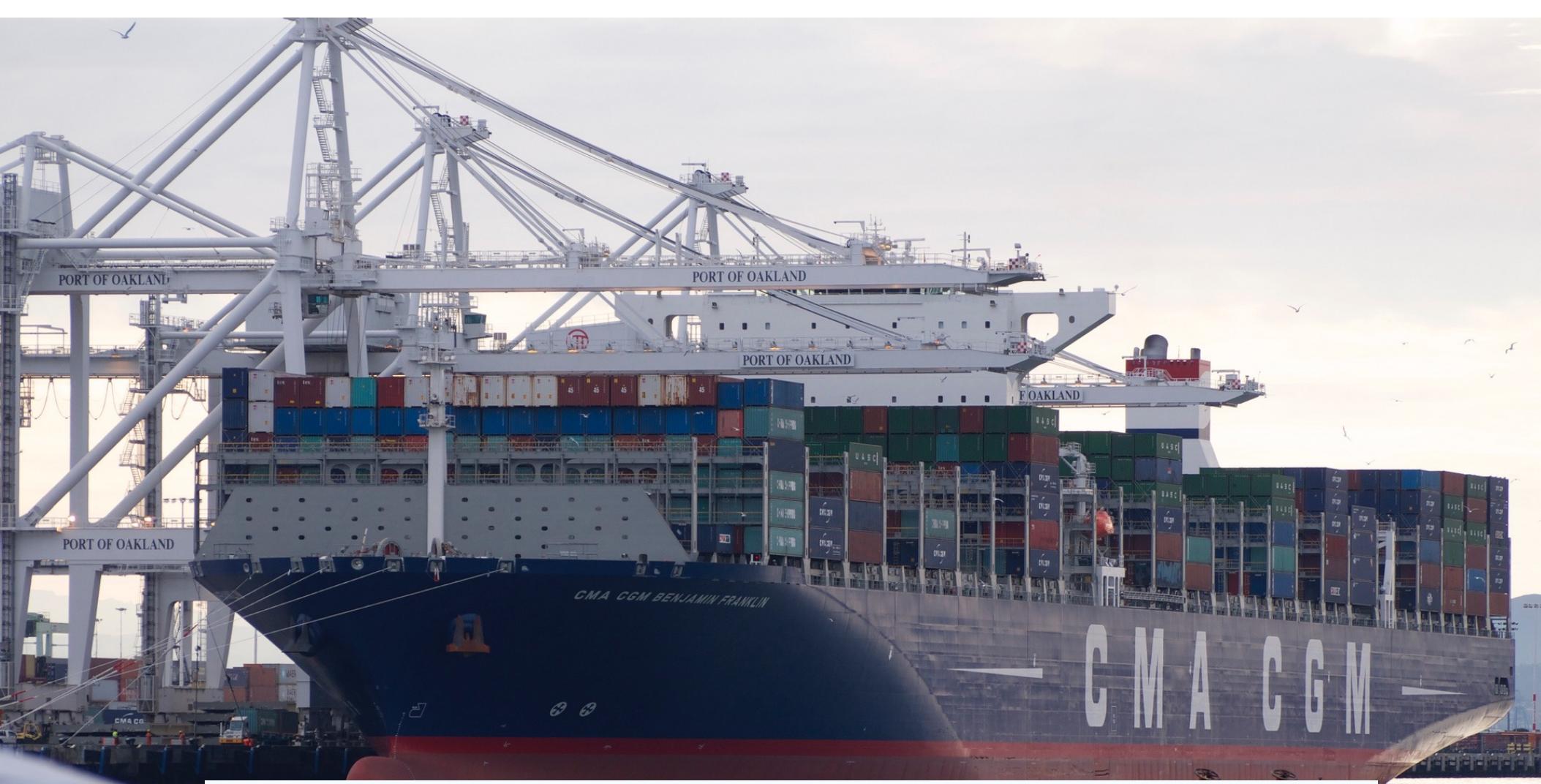
- Great isolation
- Great manageability
- VM encapsulates implementation technology
- Leverage AWS infrastructure for Autoscaling/Load balancing

Drawbacks

- Less efficient resource utilization
- Slow deployment

Pattern: Service per Container host



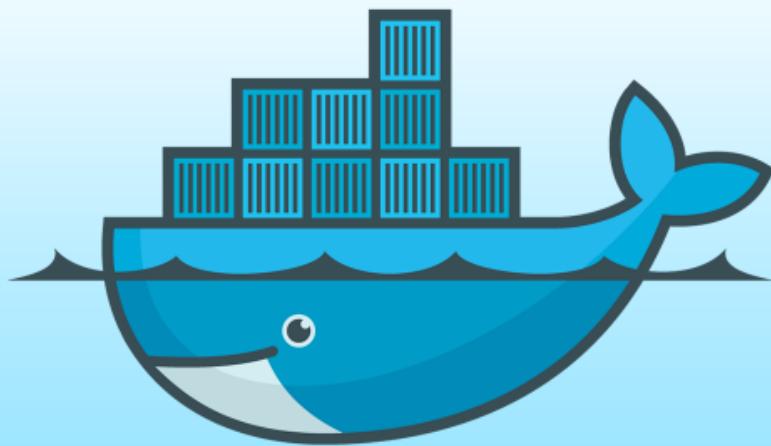


Build, Ship and Run Any App, Anywhere

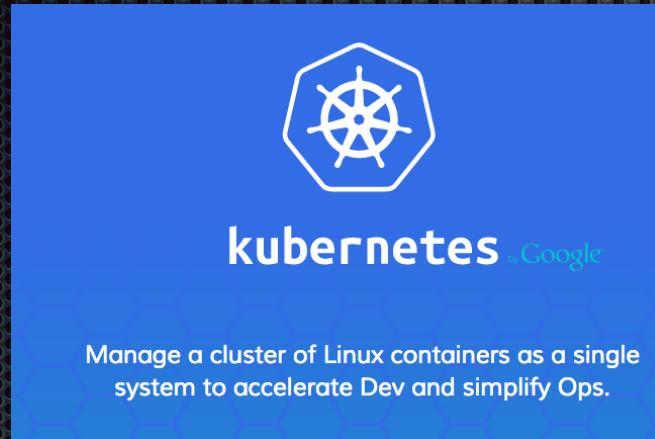
Docker - An open platform for distributed applications for developers and sysadmins.

[What is Docker?](#)

[Try It!](#)



Docker clustering solutions



The image shows the Marathon landing page. It has a dark grey header with the word "Marathon" in large white letters. Below the header, there is a white text area containing the slogan: "A cluster-wide init and control system for services in cgroups or Docker containers". At the bottom of this area is a green button with the text "Download Marathon v0.8.0".

The image shows the DCHQ landing page. It features a navigation bar at the top with links: Overview, Use Cases, Business Benefits, Features, Integration & Partners, Pricing, Contact Us, Download, and Log in. Below the navigation bar, there is a large image of the Golden Gate Bridge. Overlaid on the bridge image is the text "Advanced Platform for Automation of Container based Apps on any Cloud".

<https://www.dchq.io/landing/index.html>

@crichtson

Benefits and drawbacks

Benefits

- Great isolation
- Great manageability
- Container encapsulates implementation technology
- Efficient resource utilization
- Fast deployment

Drawbacks

- Immature infrastructure for deploying containers

Agenda

- Why a pattern language for microservices?
- Core patterns
- Deployment patterns
- Communication patterns and more

Communication issues

How do clients of the system interact with the services?

System Client

The System

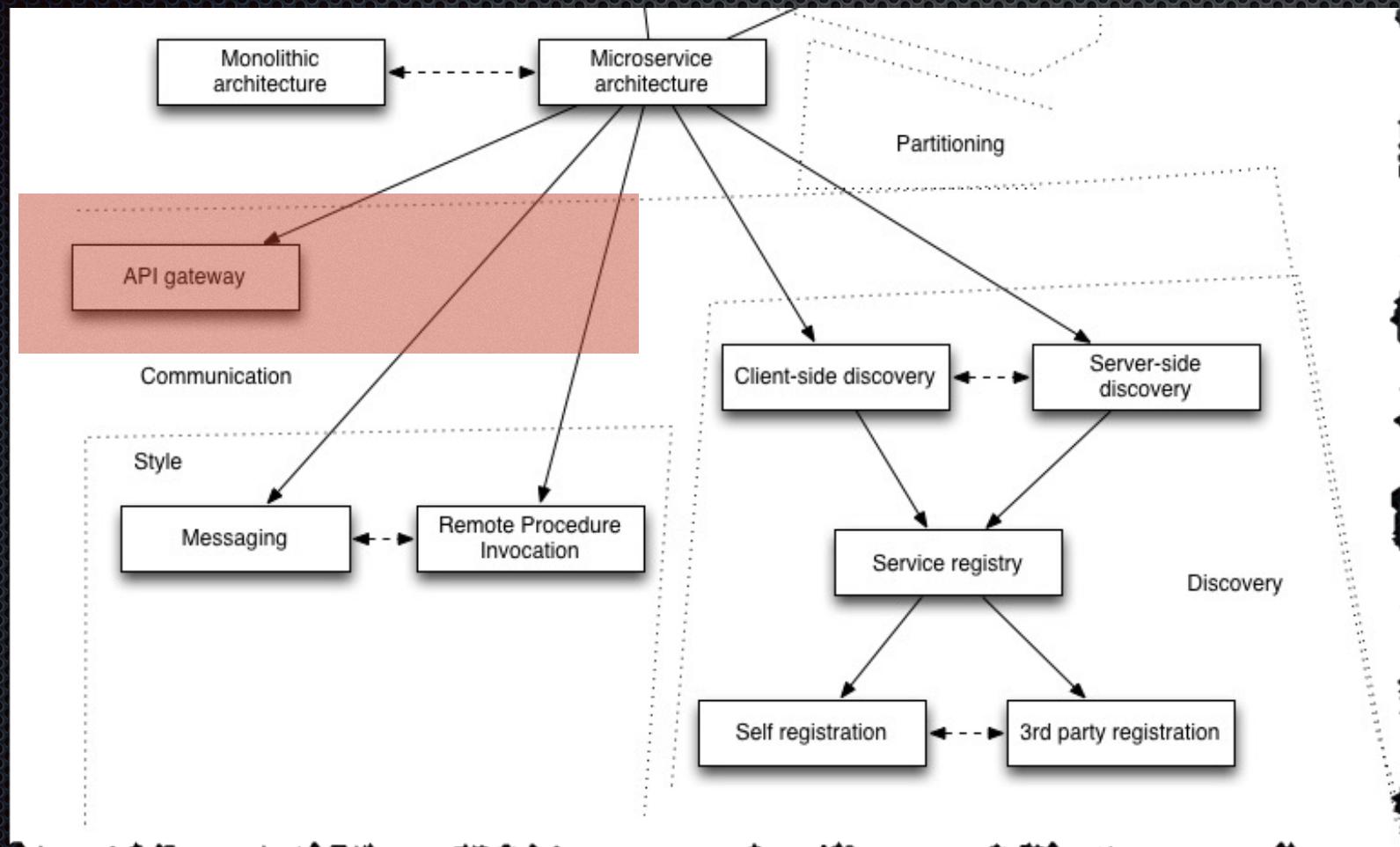
Service A

Service C

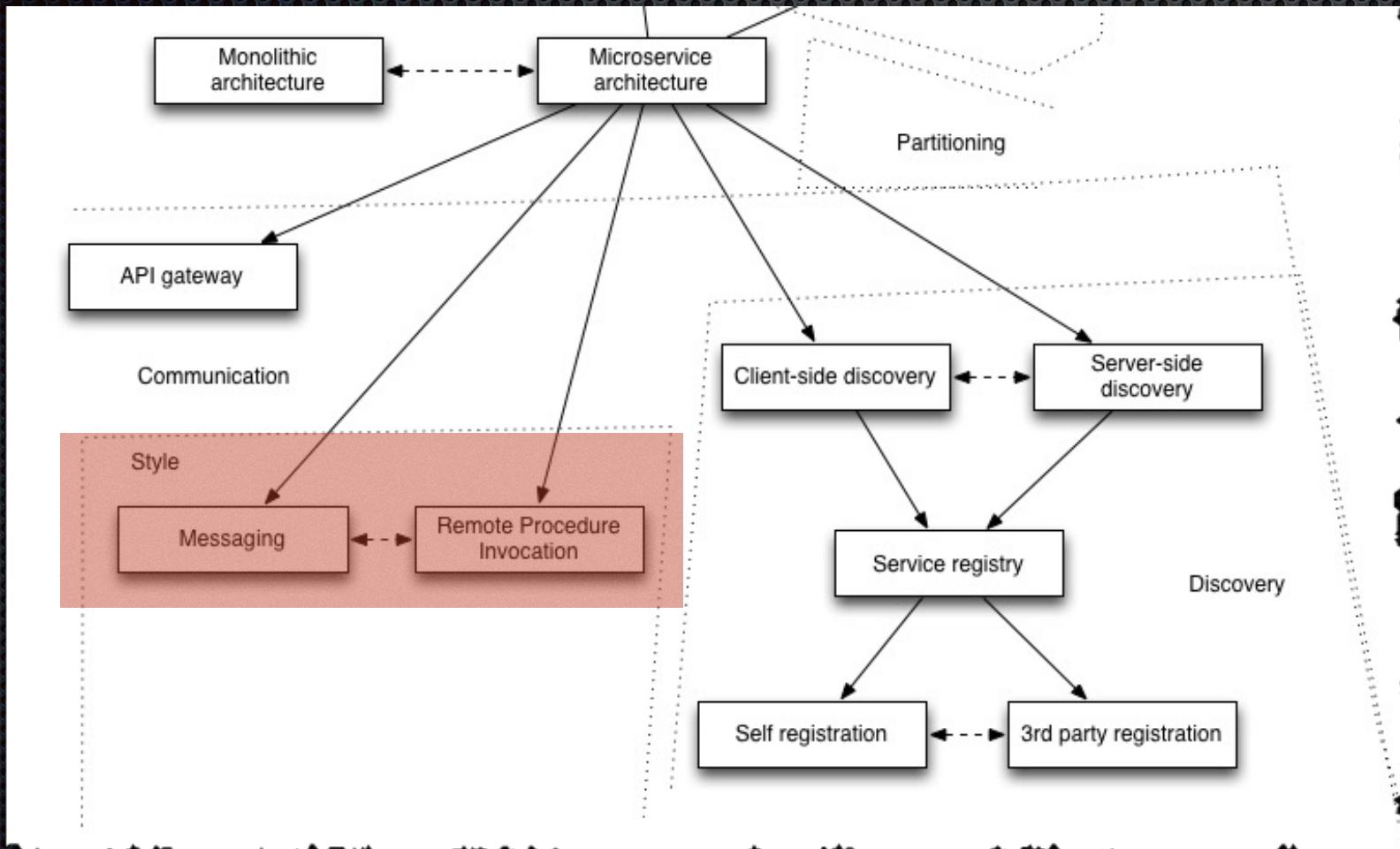
Service B

How do services within the system interact?

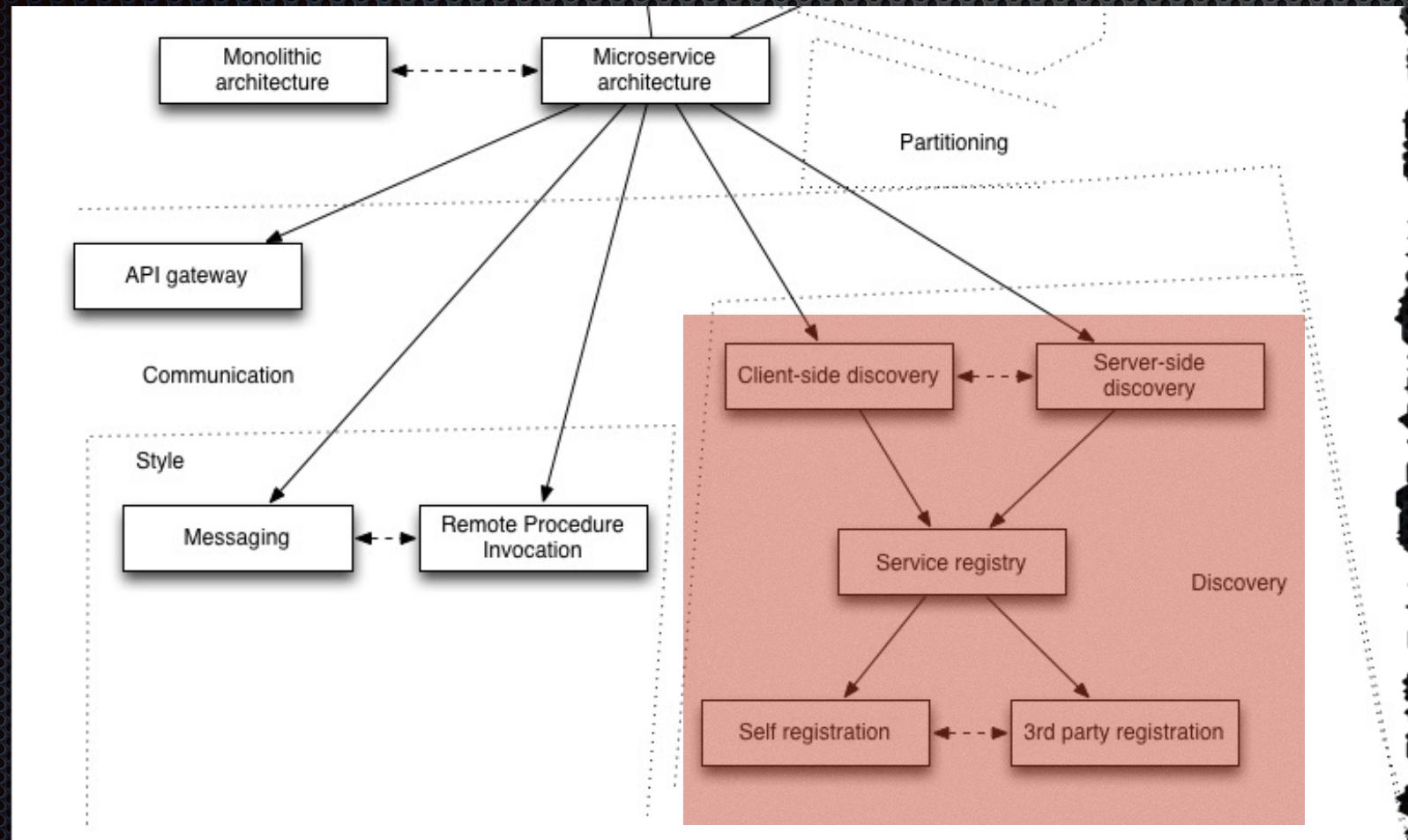
How do external clients interact with the microservices?



How do services communicate with each other?

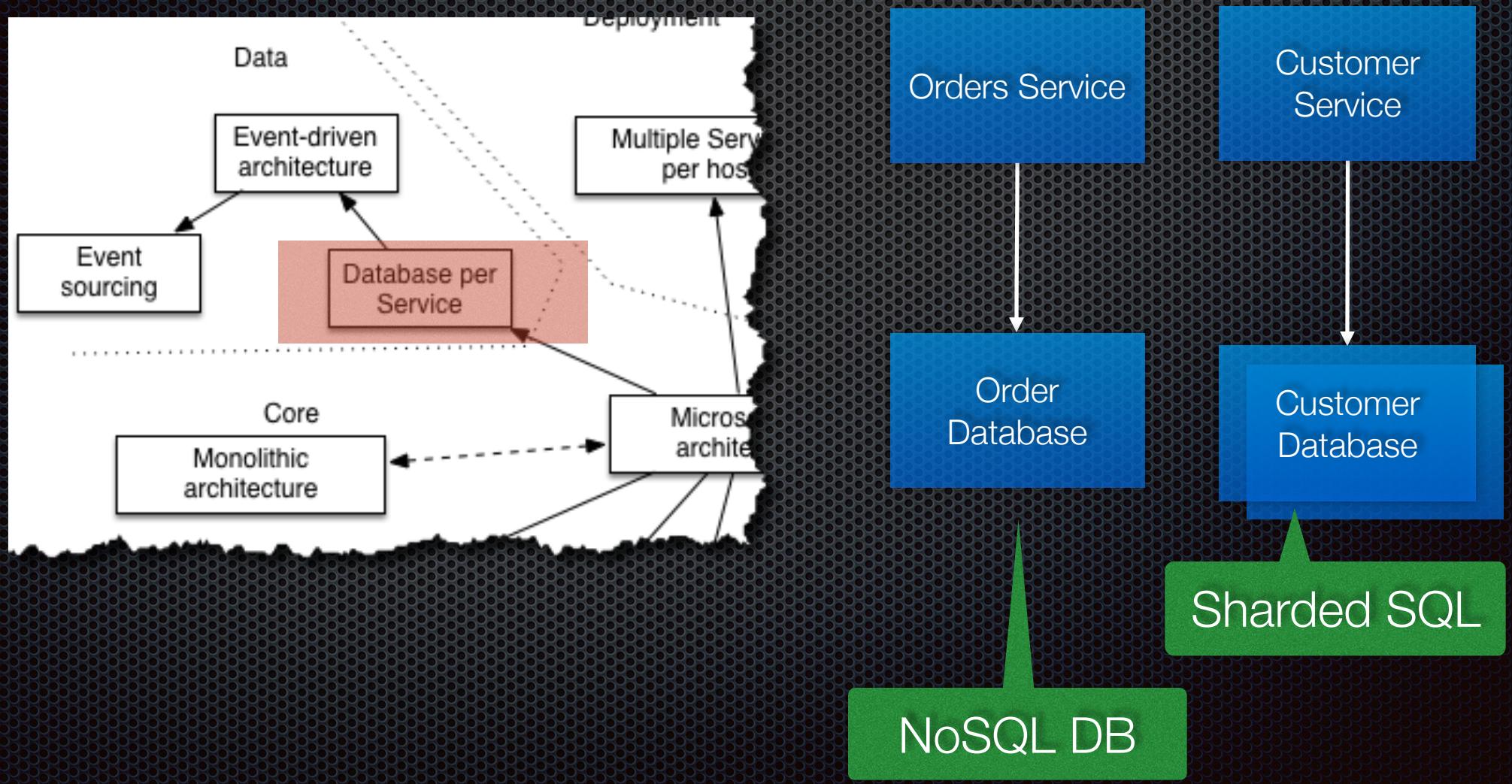


How does a client determine the network location of a service?

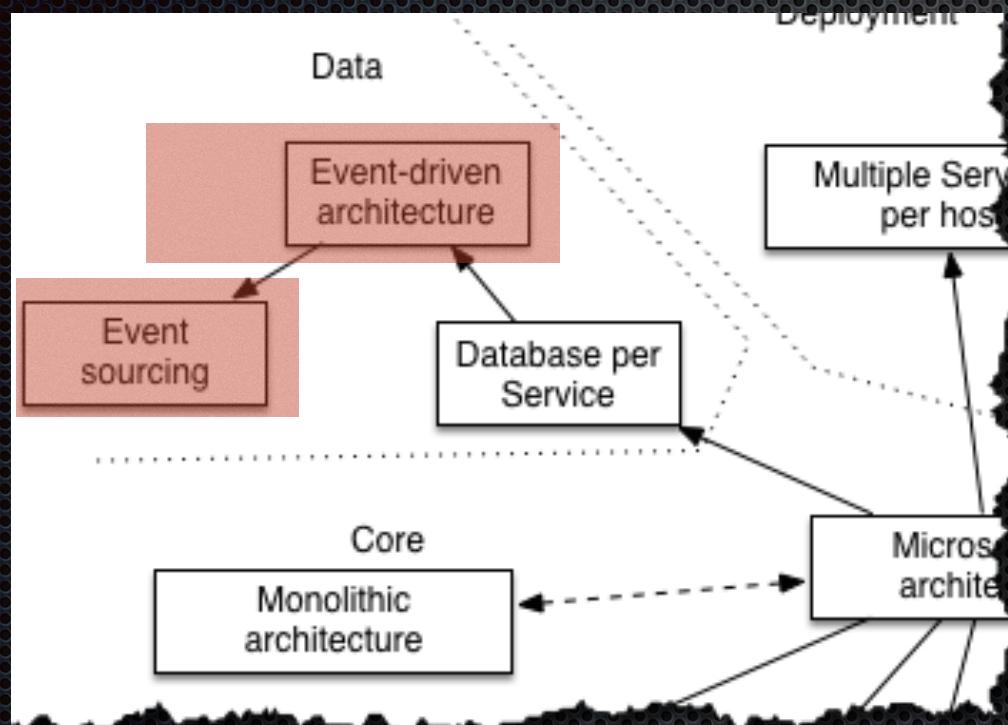


What's the database architecture?

Database per Service



How to solve distributed data management challenges?



- How to implement transactions that span multiple services?
- How to implement queries that span multiple services?

Summary: Patterns and pattern languages are a great way to ...

- Think about technology
- Discuss technology
- Apply technology

Summary: The Microservices pattern language is a great way to ...

- Think about microservices
- Discuss microservices
- Apply microservices (or not)

• @crichton chris@chrisrichardson.net

A close-up photograph of an emu's head, showing its large, dark brown eyes and hooked beak. The feathers around the eyes are dark and textured. The background is blurred green foliage.

Questions?

<http://plainoldobjects.com>

<http://microservices.io>

<http://eventuate.io>