



Decomposing applications for deployability and scalability

Chris Richardson,

Author of POJOs in Action, Founder of the original CloudFoundry.com

🐦 @crichardson crichardson@vmware.com <http://plainoldobjects.com/>

Presentation goal

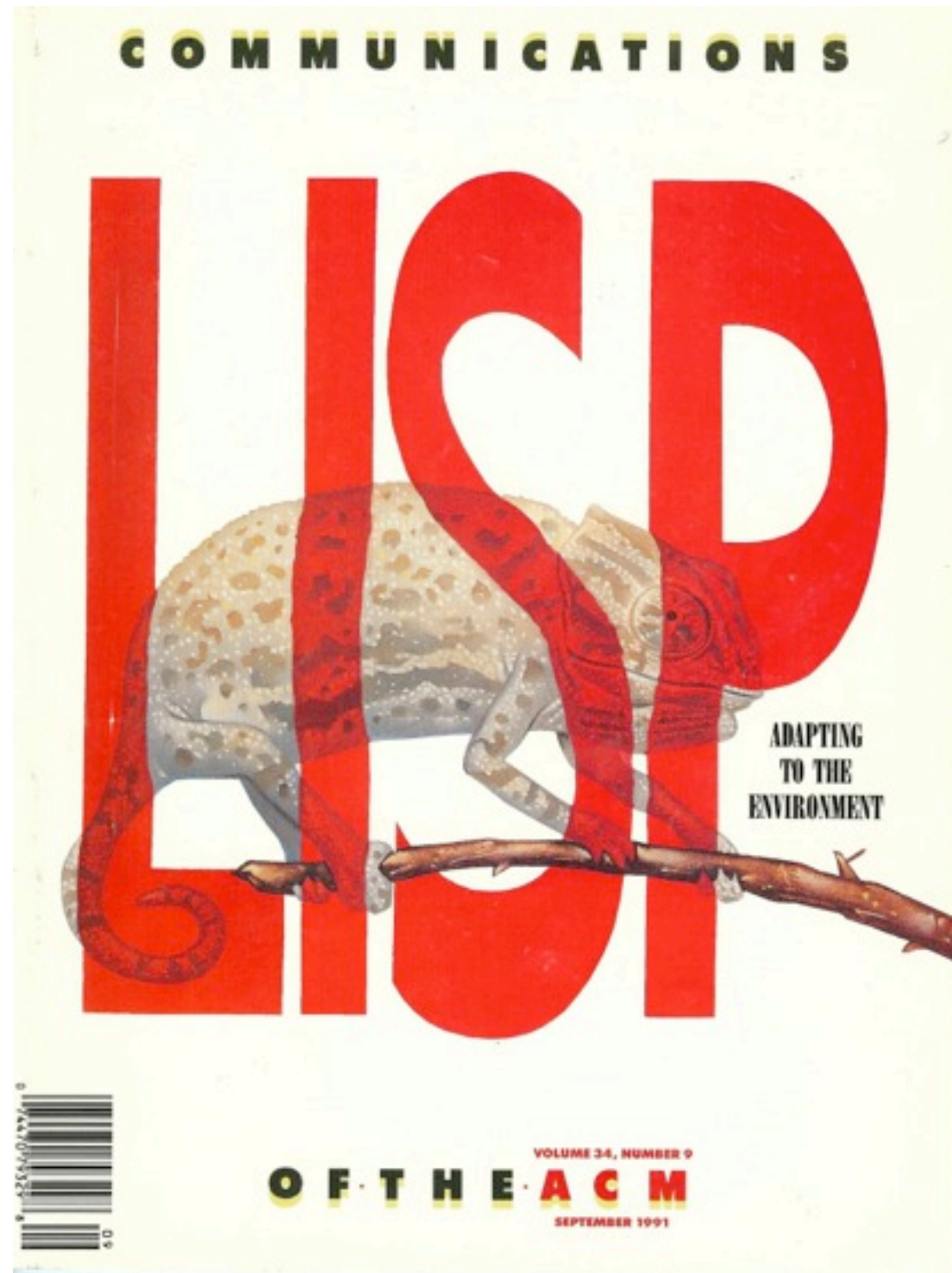
How decomposing applications
improves deployability and
scalability
and

How Cloud Foundry helps

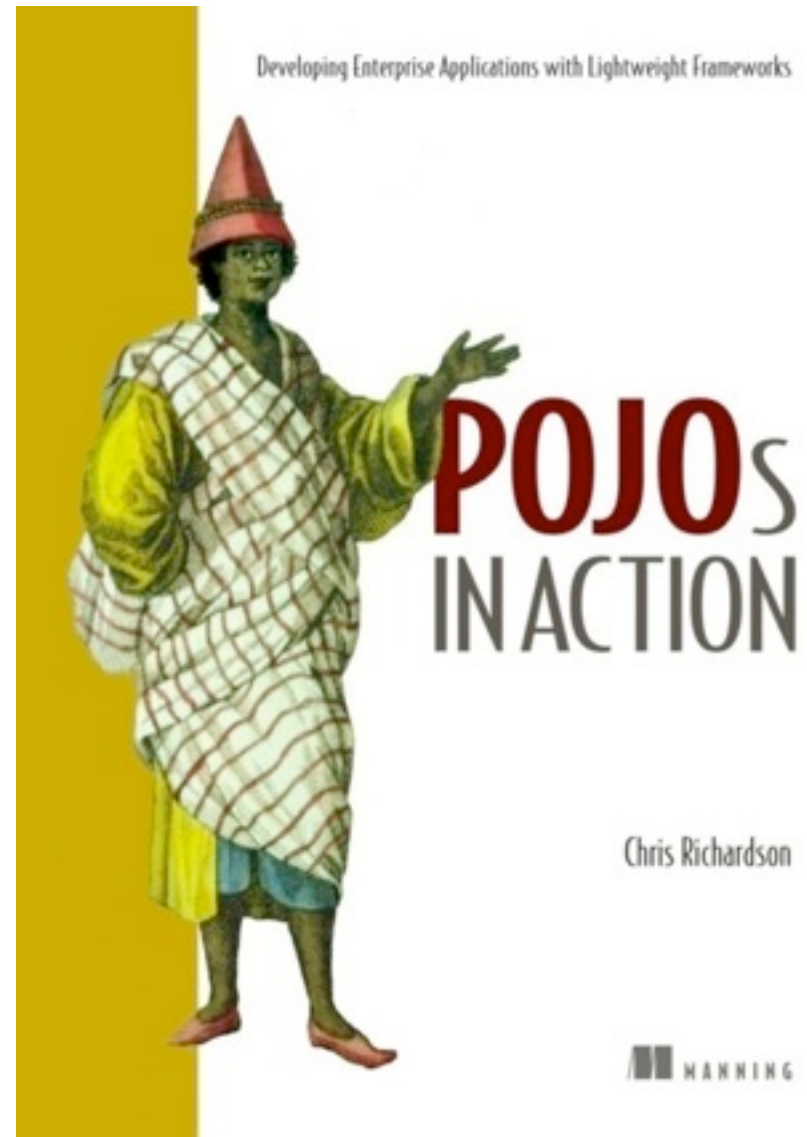
About Chris



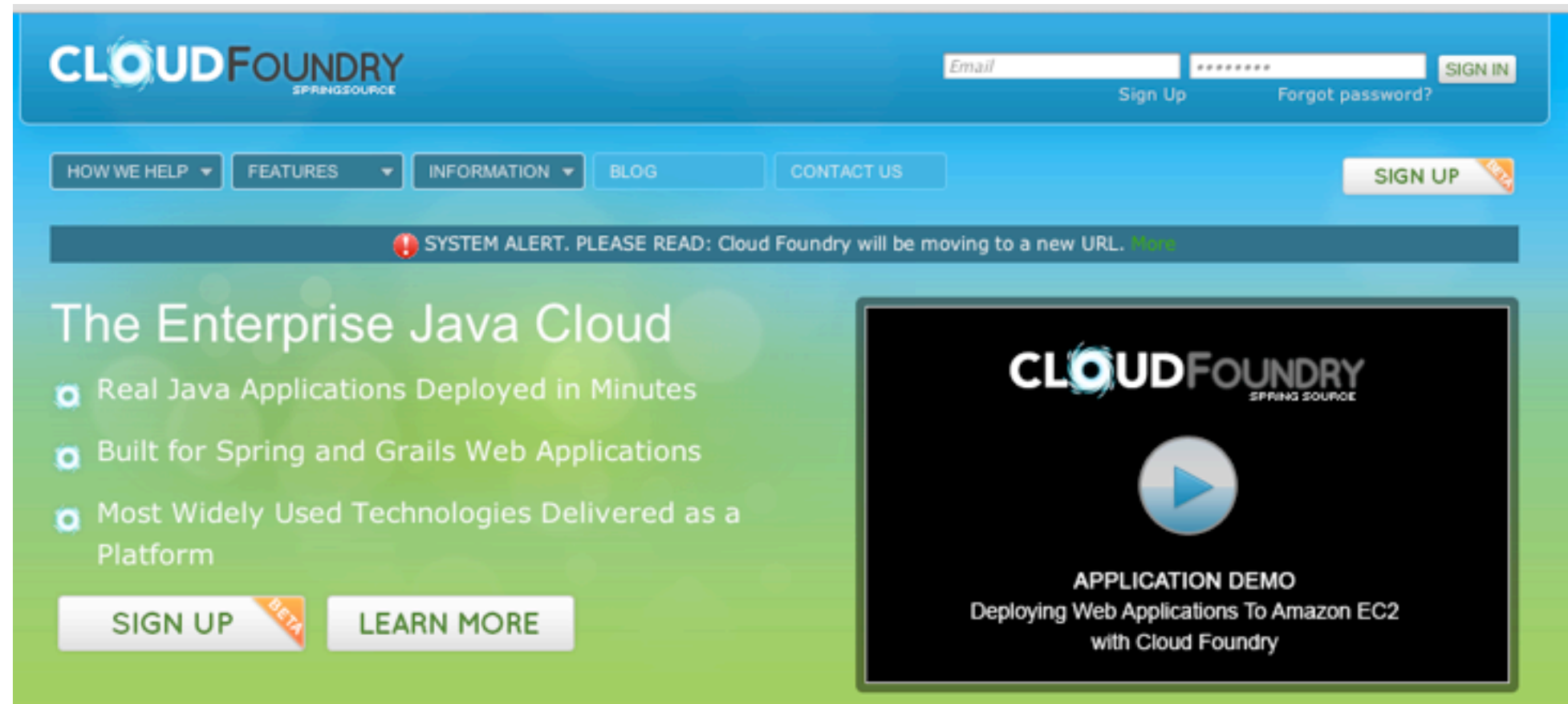
(About Chris)



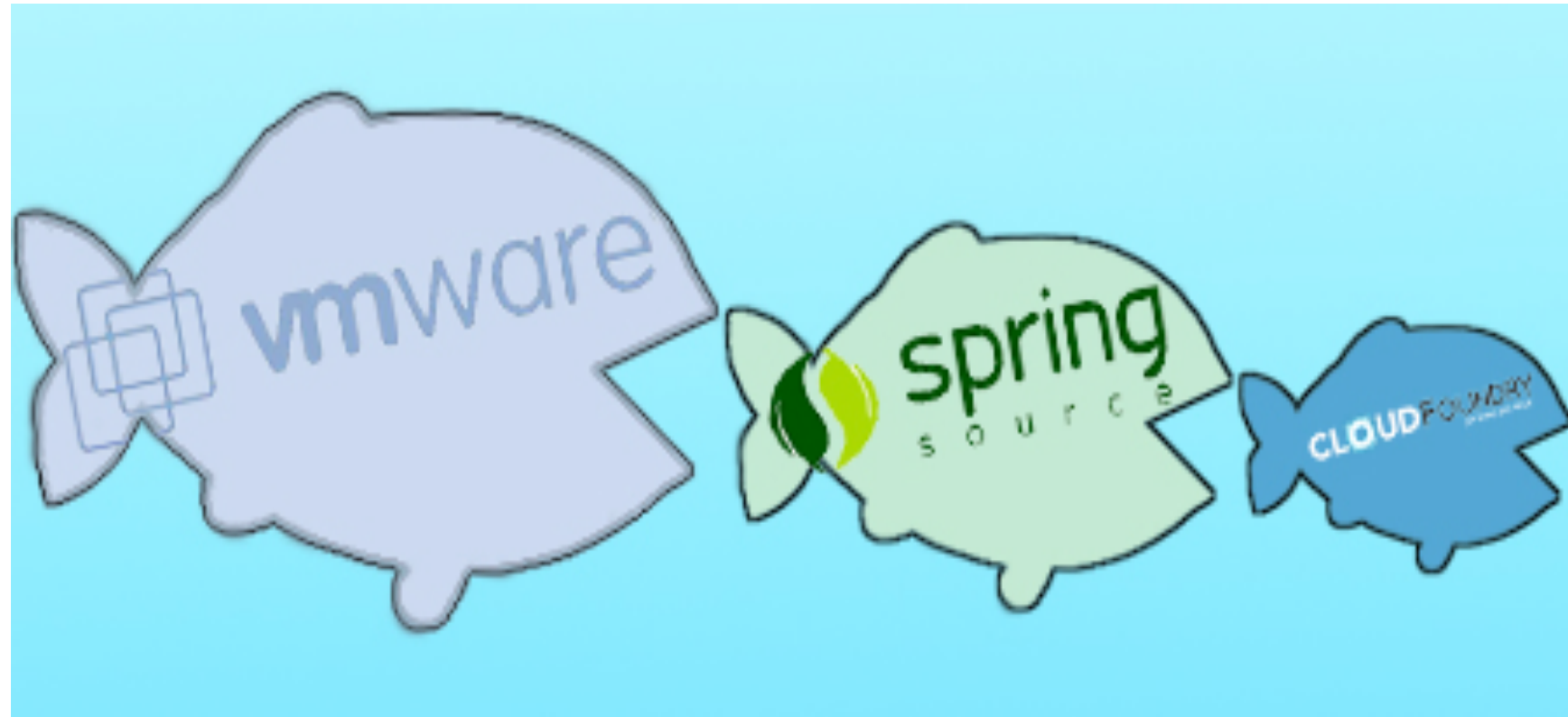
About Chris()



About Chris



About Chris



http://www.theregister.co.uk/2009/08/19/springsource_cloud_foundry/

vmc push About-Chris

Developer Advocate



vmc push About-Chris

Developer Advocate



vmc push About-Chris

Developer Advocate



Signup at <http://cloudfoundry.com>

Agenda

- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps

**Let's imagine you are building
an e-commerce application**

Traditional web application architecture

Traditional web application architecture



StoreFrontUI

Traditional web application architecture

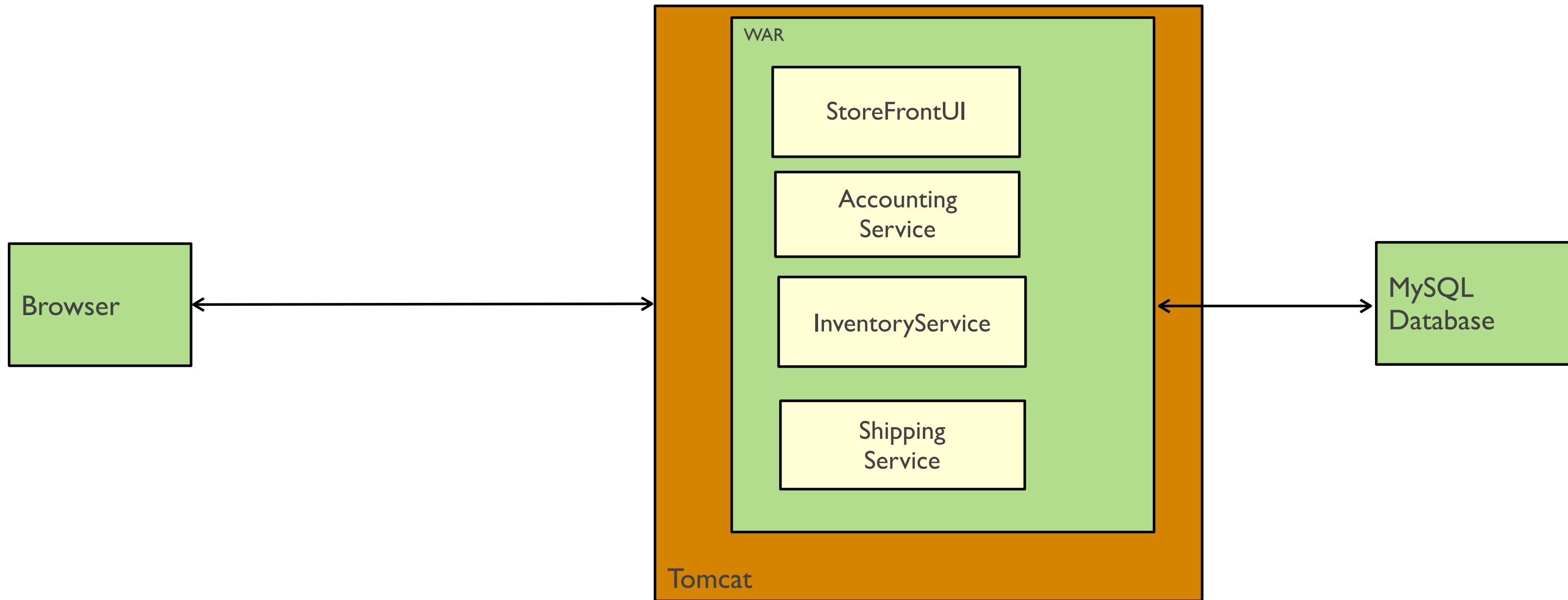
StoreFrontUI

Accounting
Service

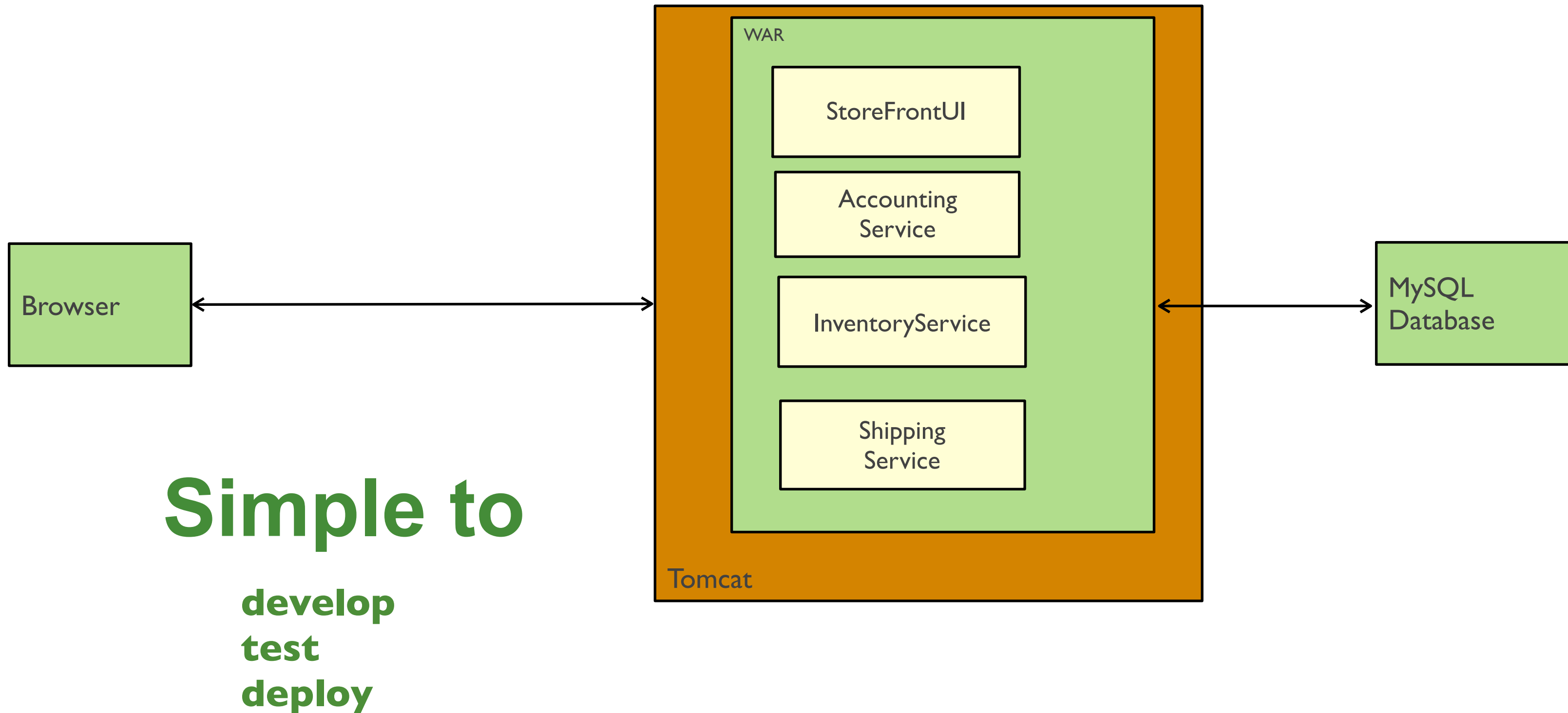
InventoryService

Shipping
Service

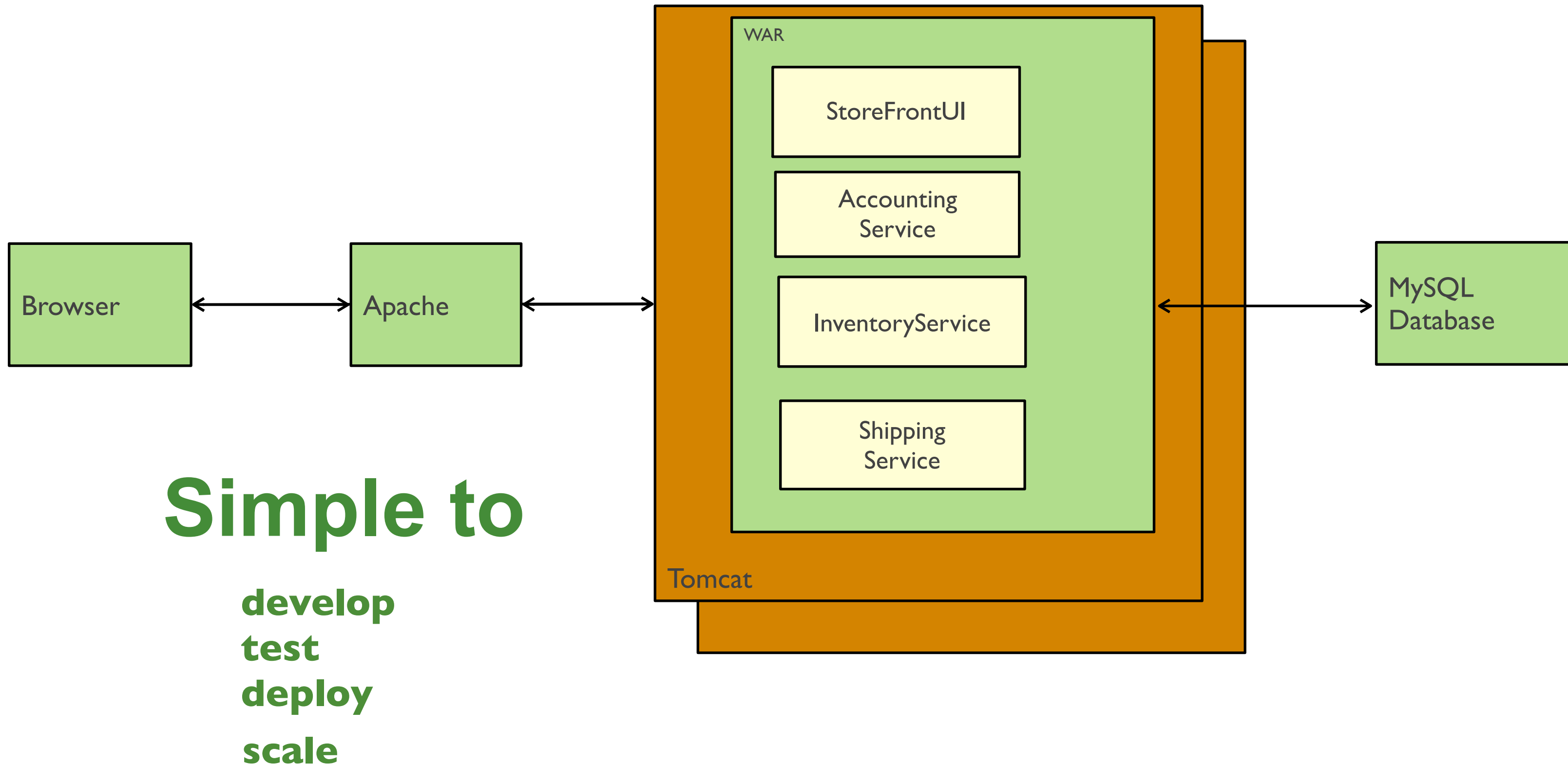
Traditional web application architecture



Traditional web application architecture

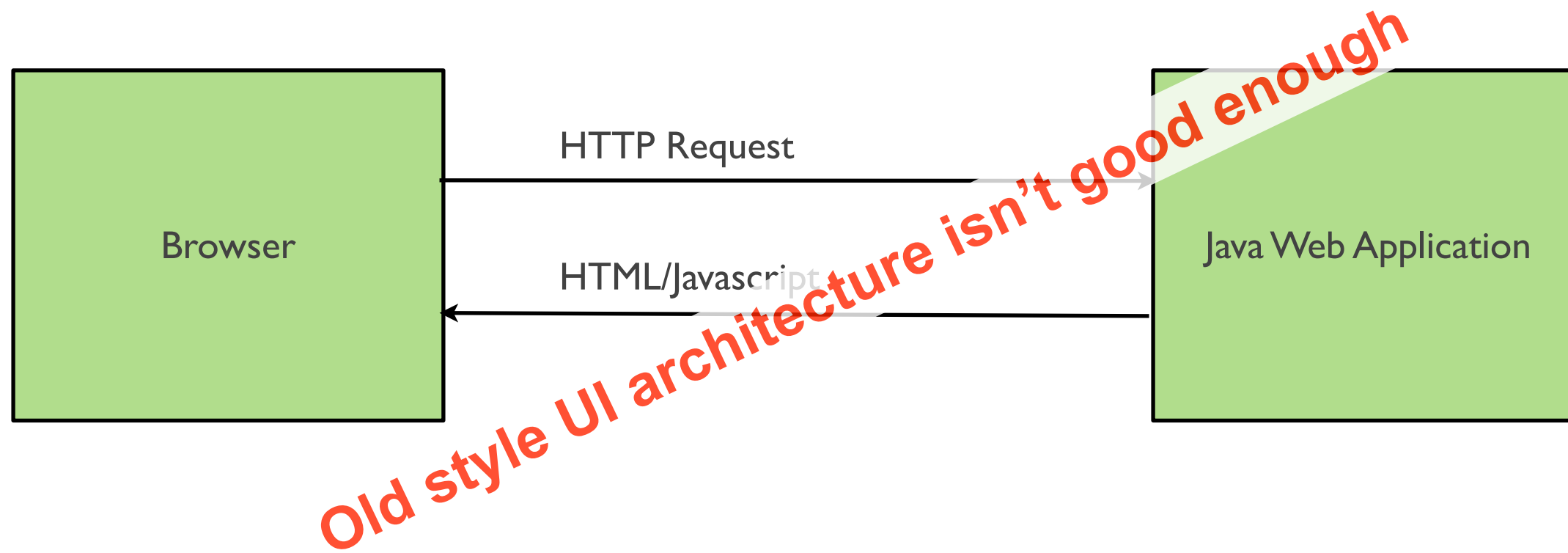


Traditional web application architecture

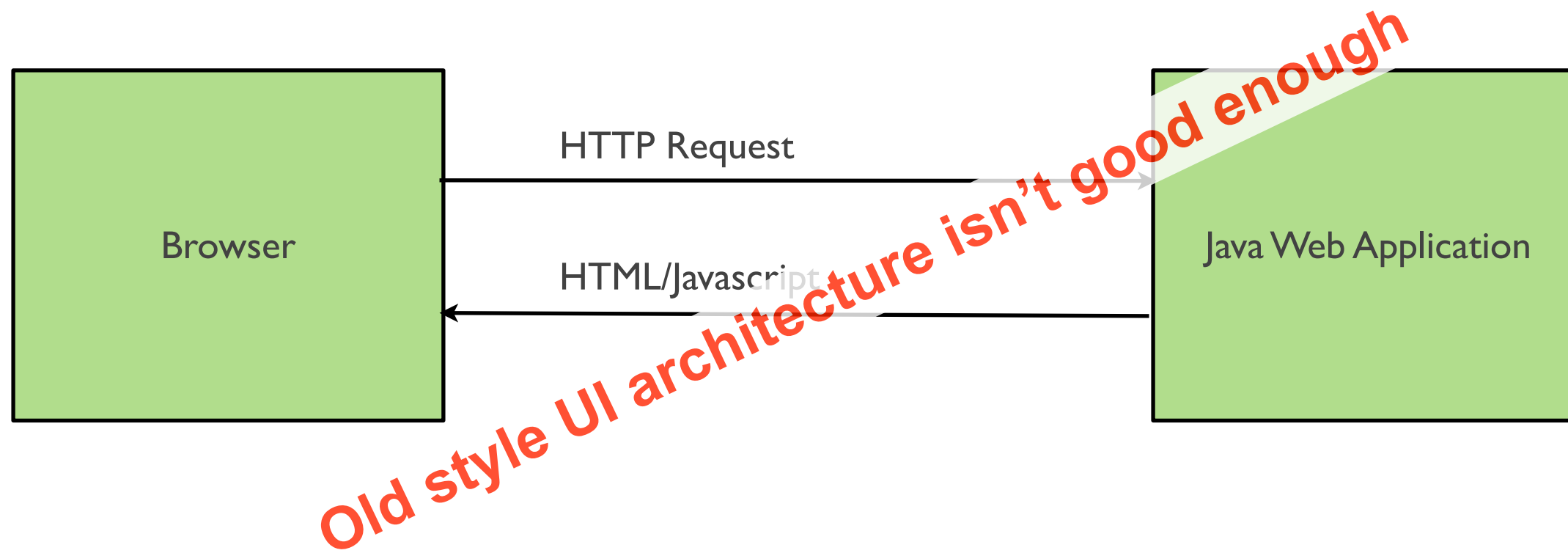


**But there are problems with a
monolithic architecture**

Users expect a rich, dynamic and interactive experience



Users expect a rich, dynamic and interactive experience



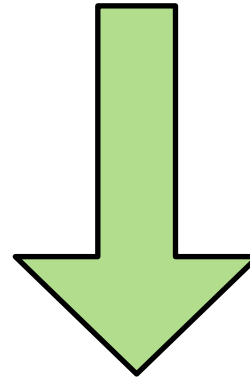
Real-time web \cong NodeJS

Intimidates developers

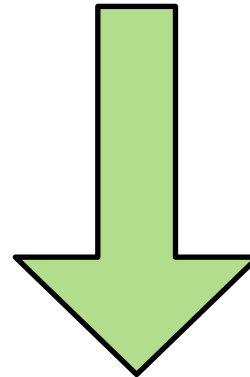


Obstacle to frequent deployments

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



Fear of change



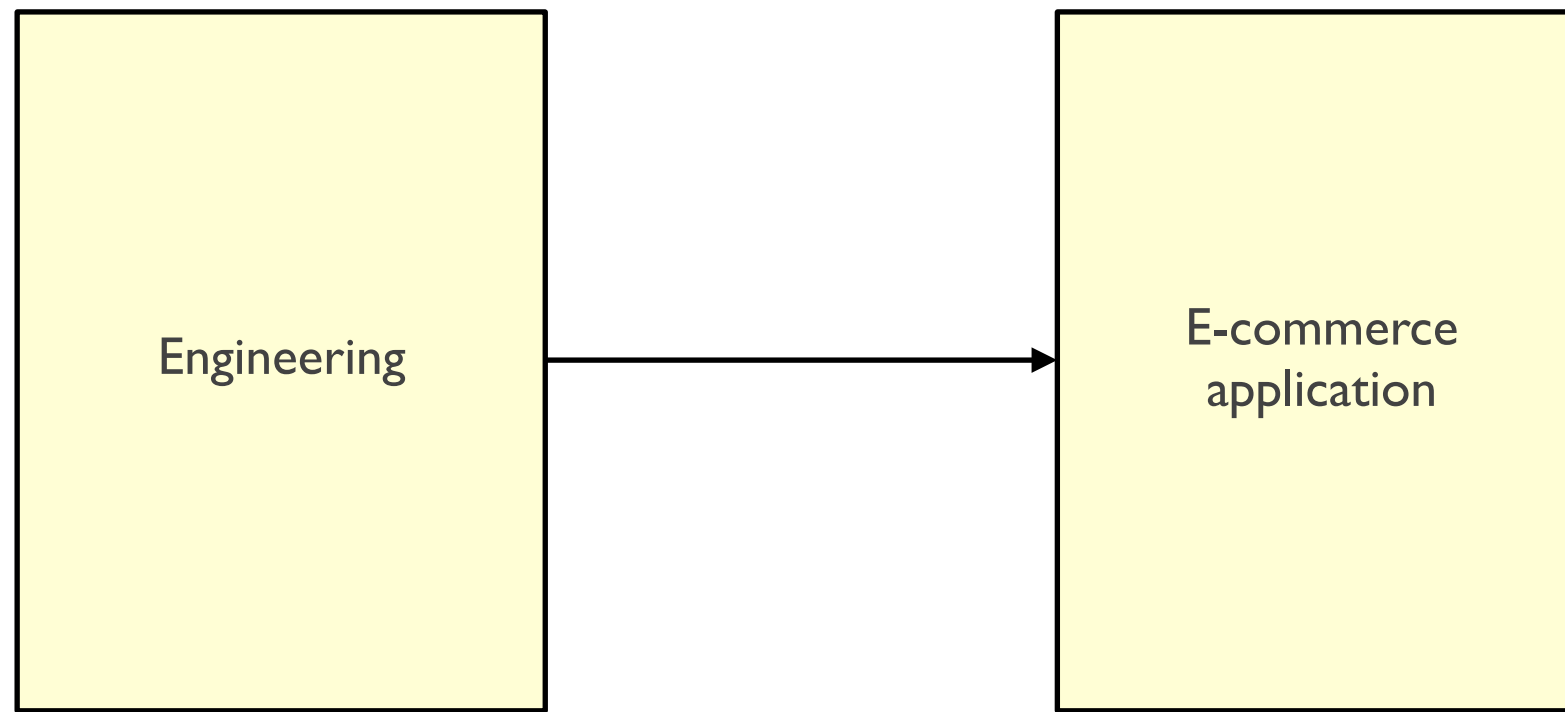
- Updates will happen less often
- e.g. Makes A/B testing UI really difficult

Overloads your IDE and container

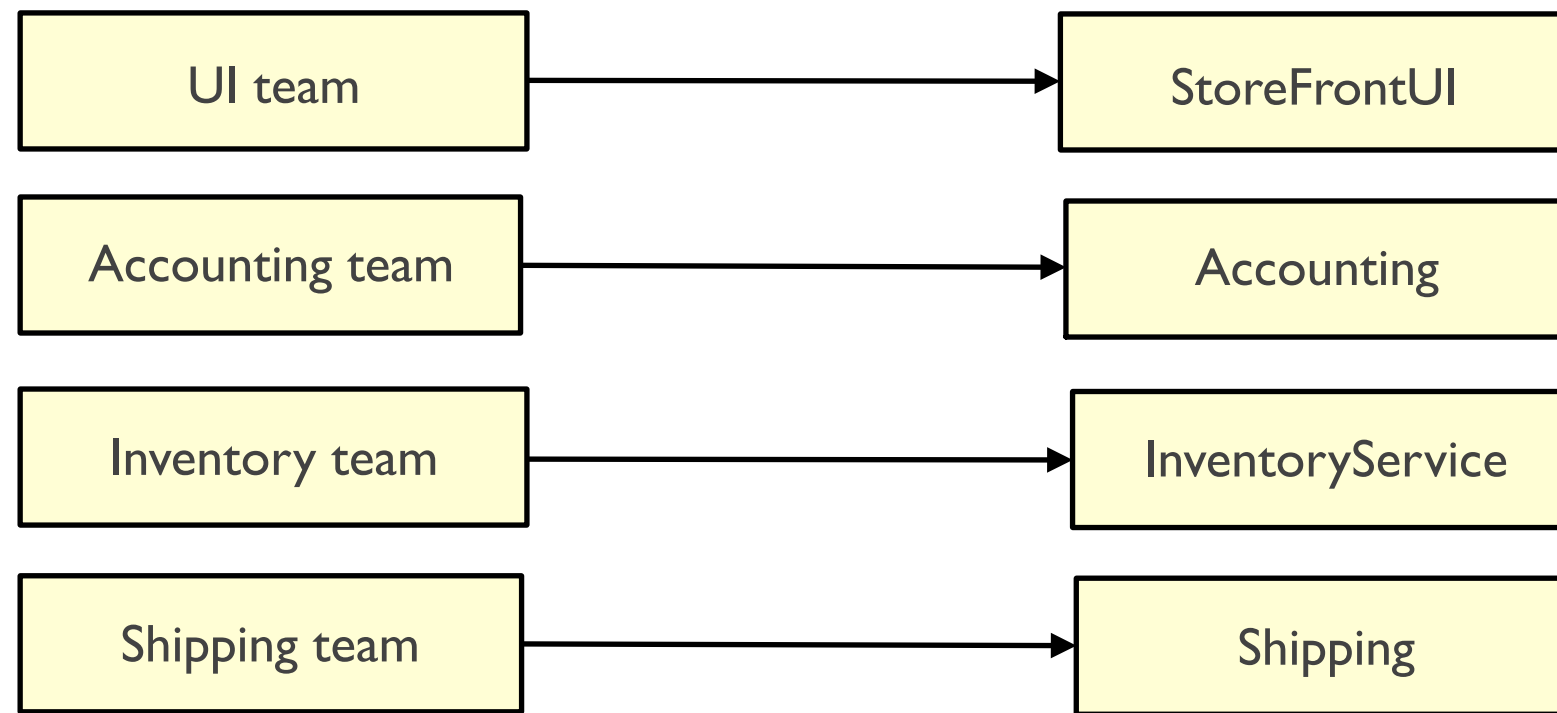


Slows down development

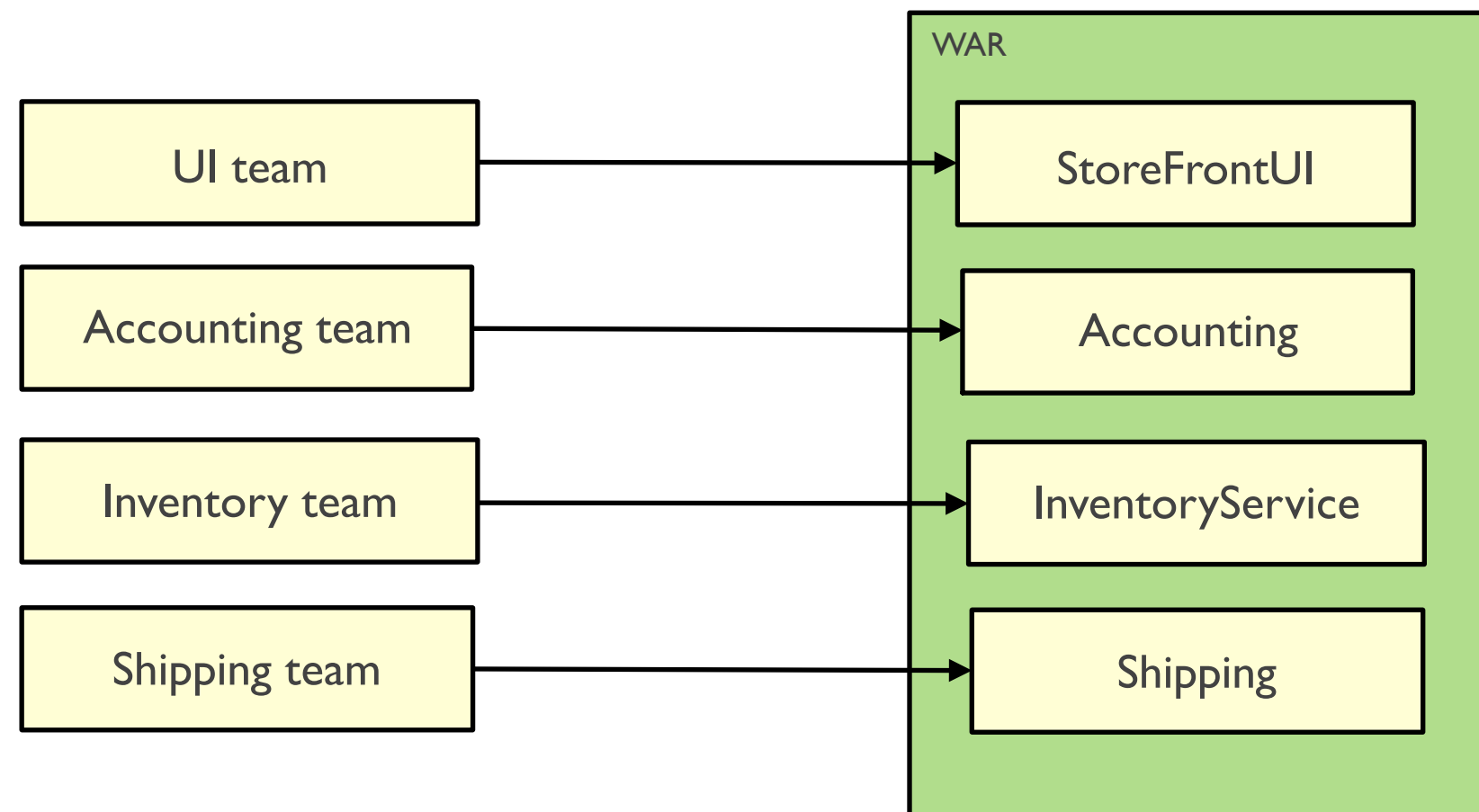
Obstacle to scaling development



Obstacle to scaling development



Obstacle to scaling development



Obstacle to scaling development



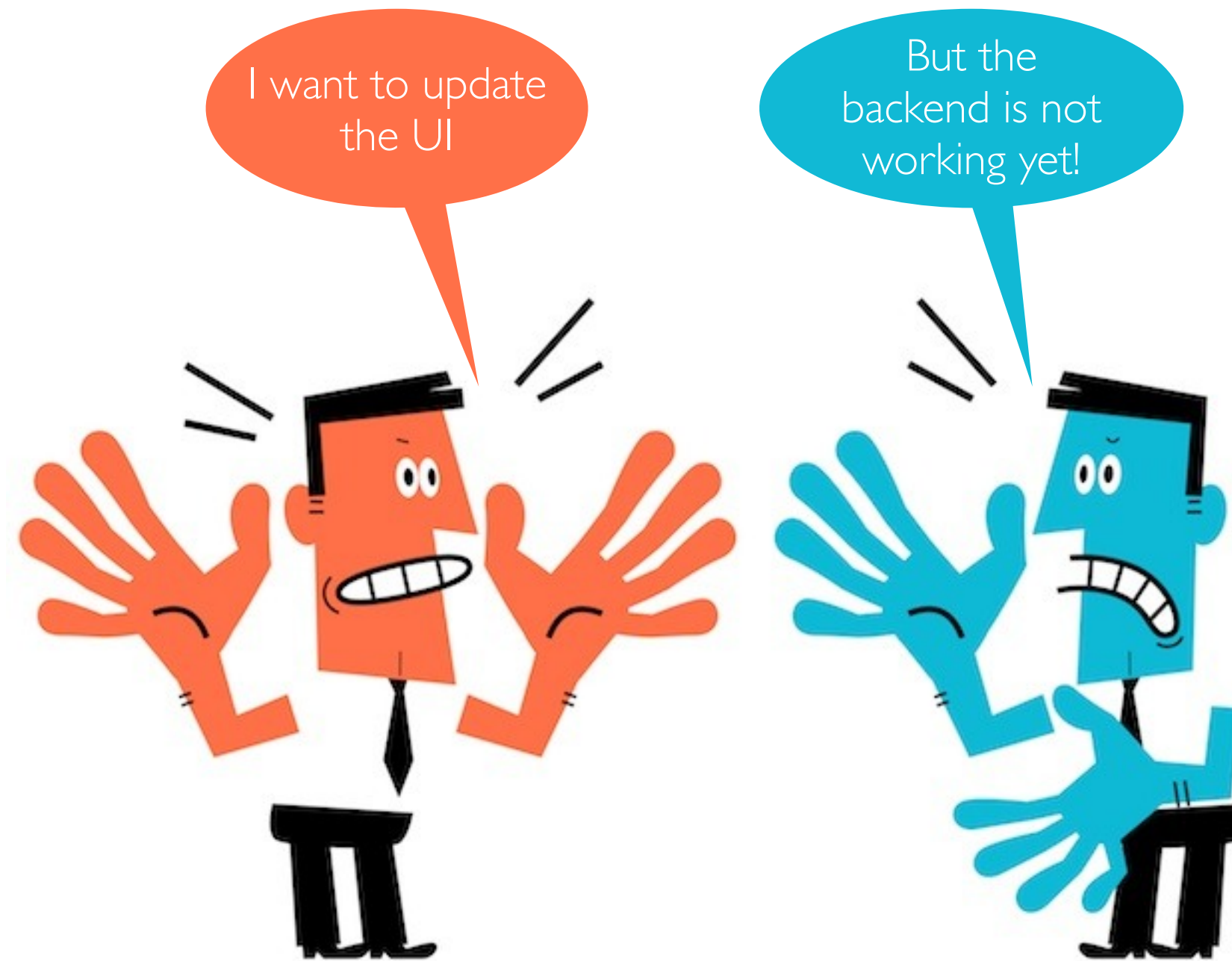
Lots of coordination and communication
required

Obstacle to scaling development



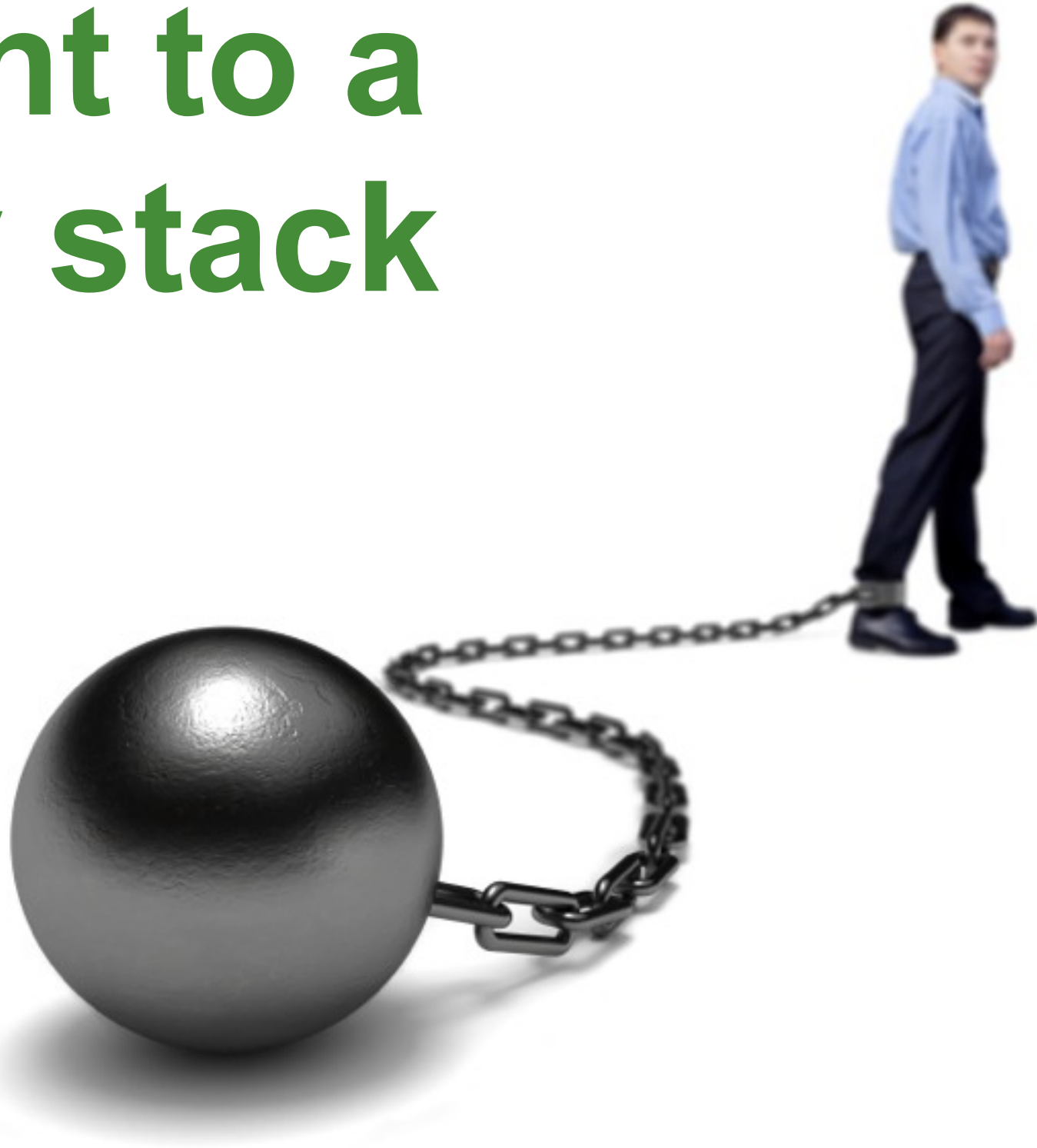
Lots of coordination and communication
required

Obstacle to scaling development



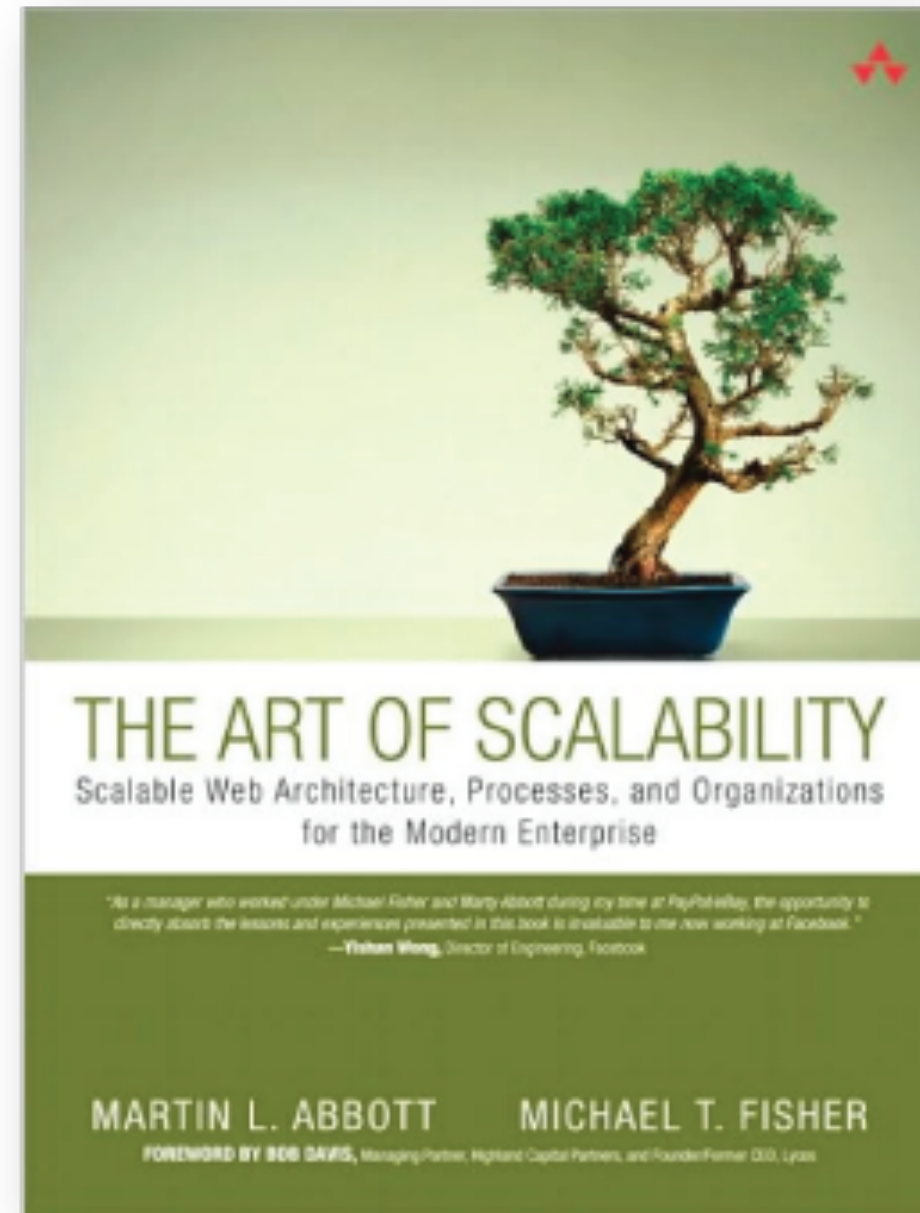
Lots of coordination and communication
required

Requires long-term commitment to a technology stack

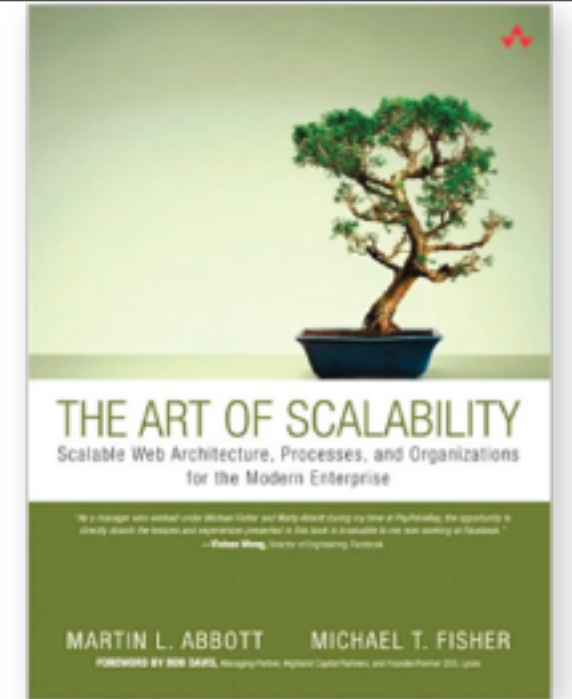
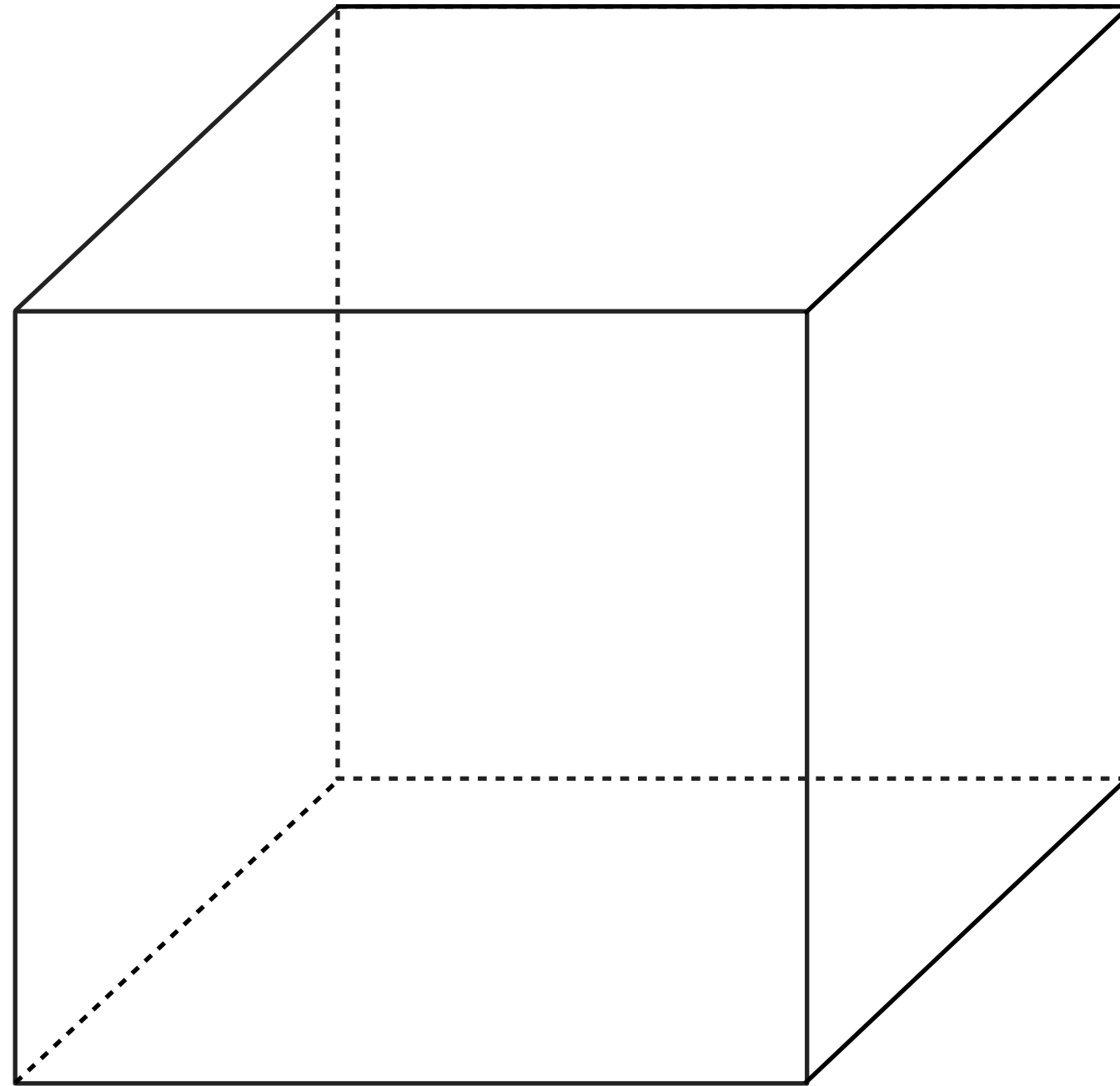


Agenda

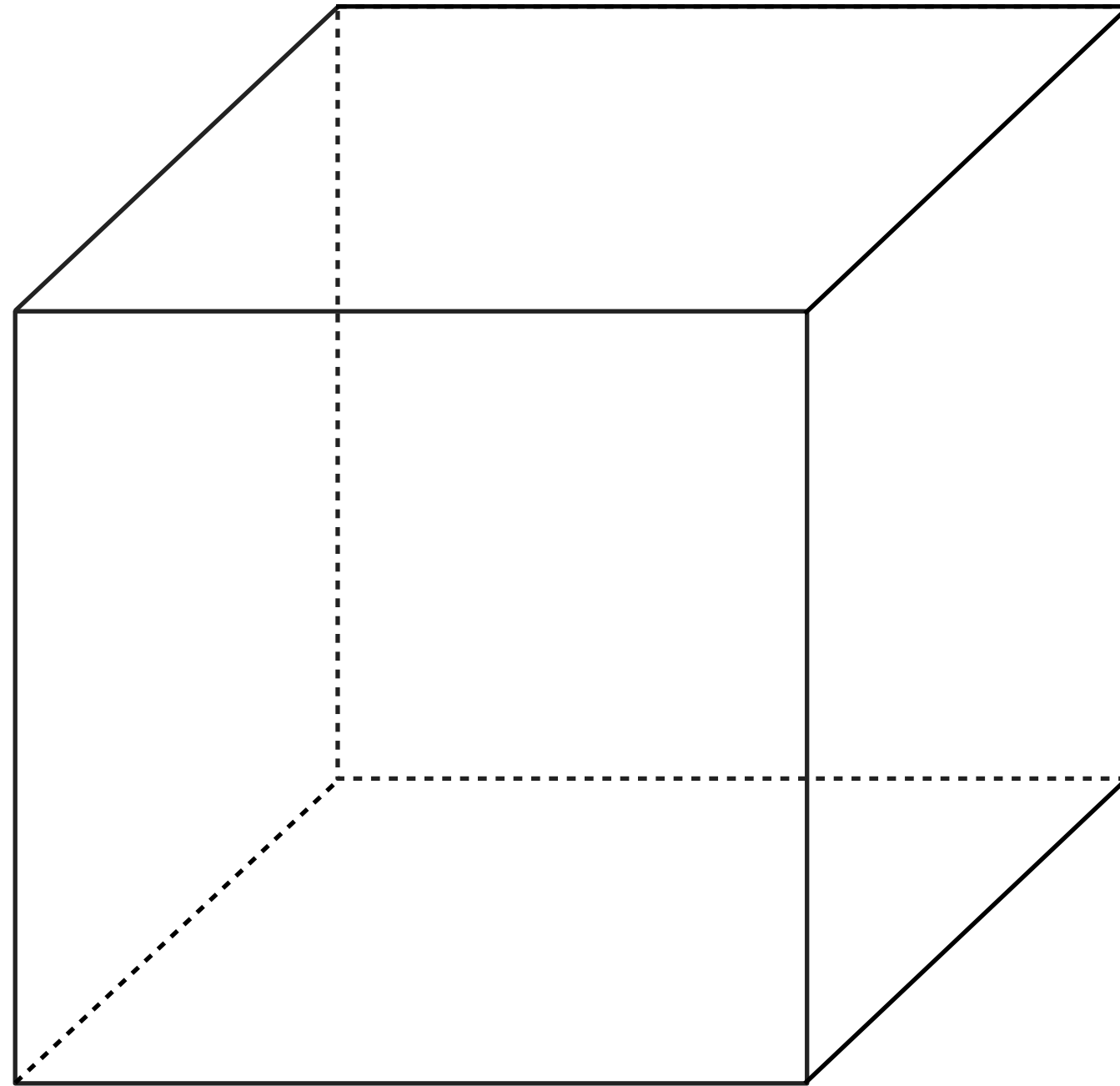
- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps



The scale cube

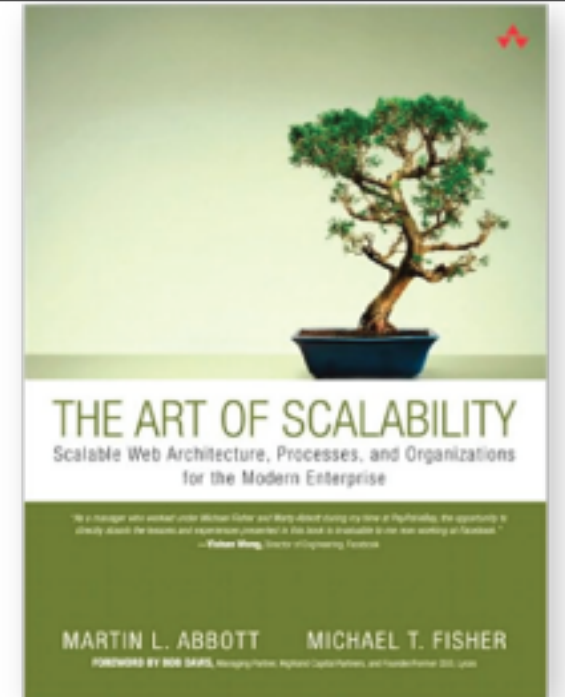


The scale cube

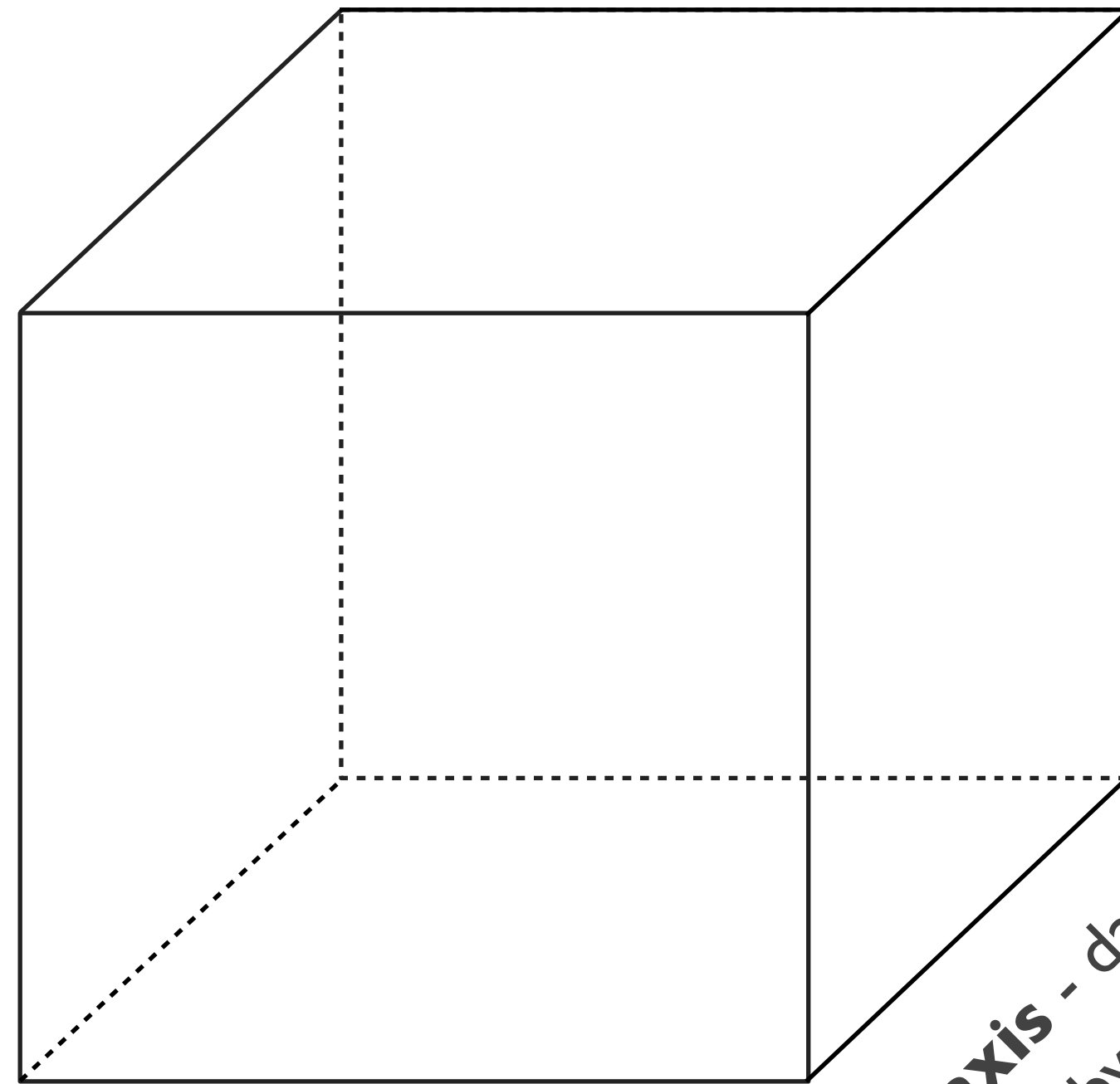


X axis

- horizontal duplication

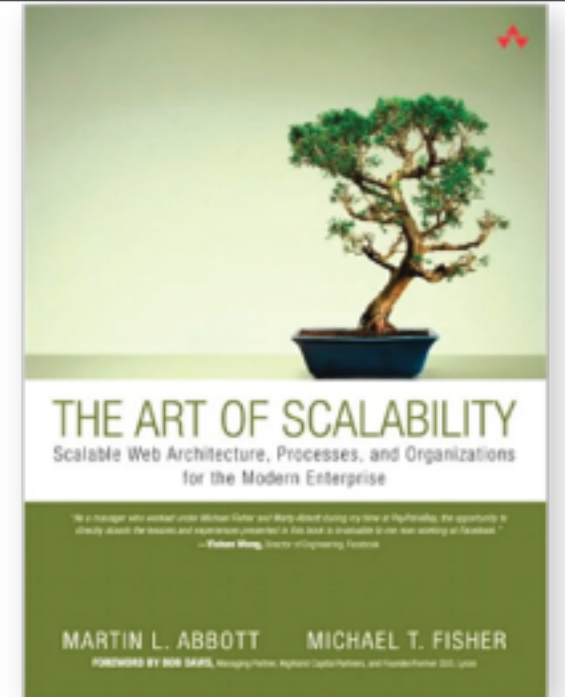


The scale cube



X axis
- horizontal duplication

Z axis - data partitioning
Scale by splitting similar
things

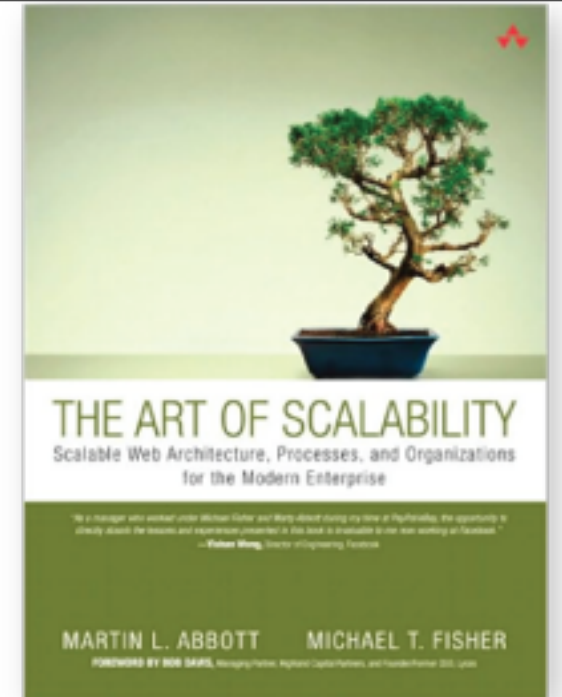


The scale cube

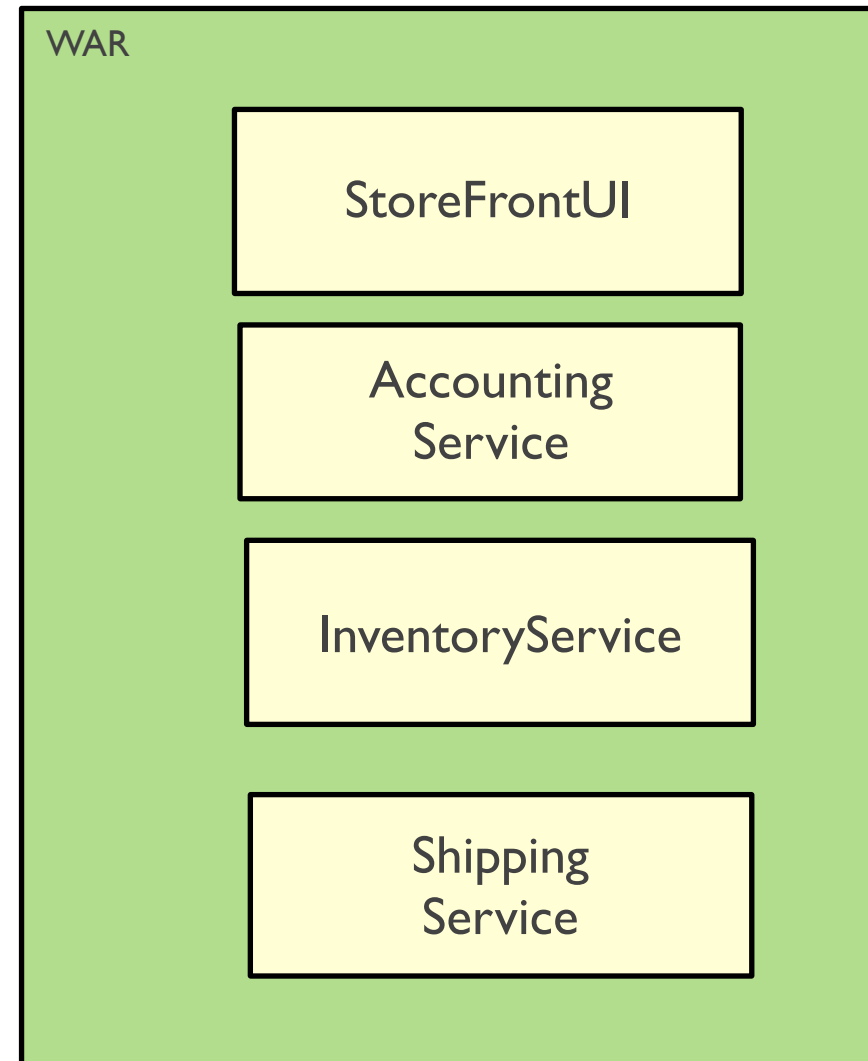
Y axis -
functional
decomposition
Scale by
splitting
different things

X axis
- horizontal duplication

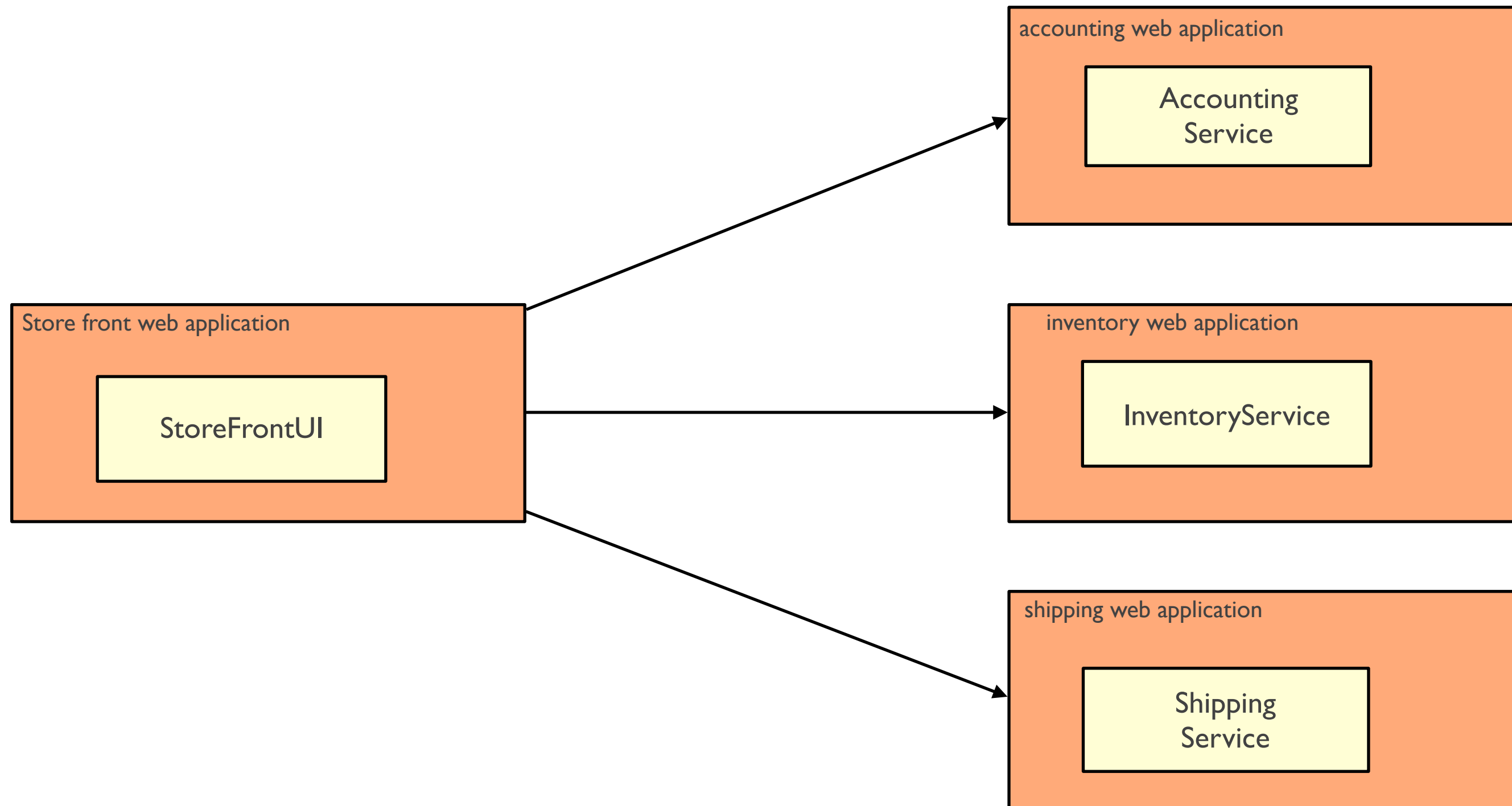
Z axis - data partitioning
Scale by splitting similar
things



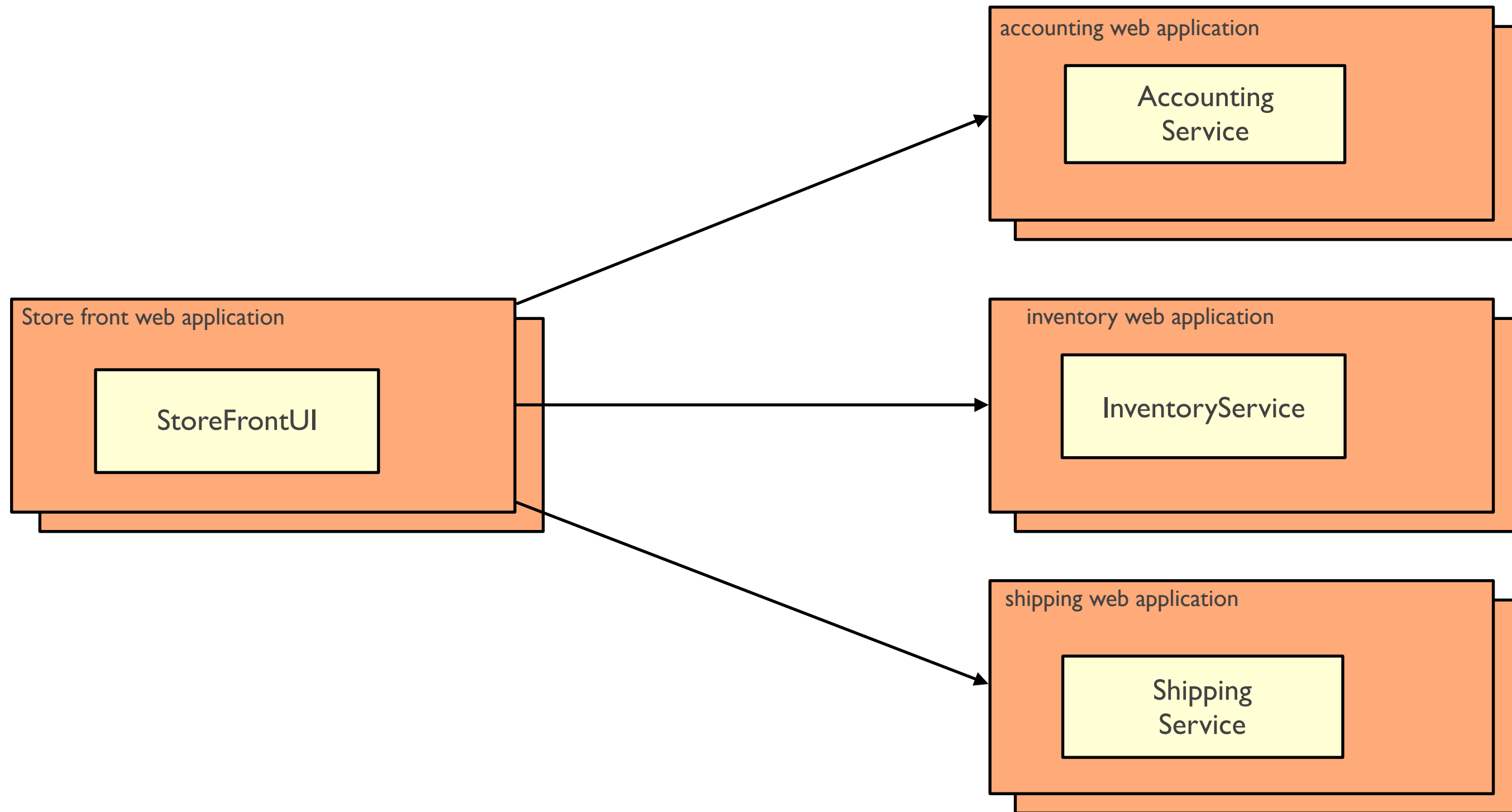
Y-axis scaling - application level



Y-axis scaling - application level



Y-axis scaling - application level



Apply X axis cloning and/or Z axis partitioning to each service

Partitioning strategies

Partitioning strategies

- Partition by verb, e.g. shipping service

Partitioning strategies

- Partition by verb, e.g. shipping service
- Partition by noun, e.g. inventory service

Partitioning strategies

- Partition by verb, e.g. shipping service
- Partition by noun, e.g. inventory service
- Single Responsibility Principle

Partitioning strategies

- Partition by verb, e.g. shipping service
- Partition by noun, e.g. inventory service
- Single Responsibility Principle
- Unix utilities - do one focussed thing well

Partitioning strategies

- Partition by verb, e.g. shipping service
- Partition by noun, e.g. inventory service
- Single Responsibility Principle
- Unix utilities - do one focussed thing well

Something of an art

Real world examples



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



Between 100-150 services are accessed to build a page.

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



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

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

There are drawbacks

Complexity

Complexity

See Steve Yegge's Google Platforms Rant re
Amazon.com

**Multiple databases
=
Transaction management
challenges**

When to use it?

When to use it?

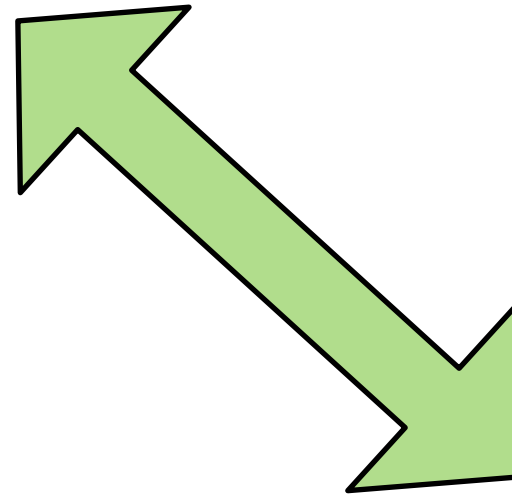
In the beginning:

- You don't need it
- It will slow you down

When to use it?

In the beginning:

- You don't need it
- It will slow you down



Later on:

- You need it
- Refactoring is painful

But there are many benefits

But there are many benefits

- Scales development: develop, deploy and scale each service independently

But there are many benefits

- Scales development: develop, deploy and scale each service independently
- Update UI independently

But there are many benefits

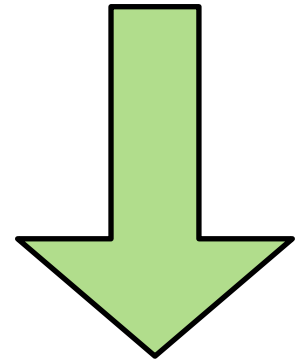
- Scales development: develop, deploy and scale each service independently
- Update UI independently
- Improves fault isolation

But there are many benefits

- Scales development: develop, deploy and scale each service independently
- Update UI independently
- Improves fault isolation
- Eliminates long-term commitment to a single technology stack

But there are many benefits

- Scales development: develop, deploy and scale each service independently
- Update UI independently
- Improves fault isolation
- Eliminates long-term commitment to a single technology stack



Modular, polyglot, multi-framework
applications

Two levels of architecture

System-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

If services are small...

If services are small...

- Regularly rewrite using a better technology stack

If services are small...

- Regularly rewrite using a better technology stack
- Adapt system to changing requirements and better technology without a total rewrite

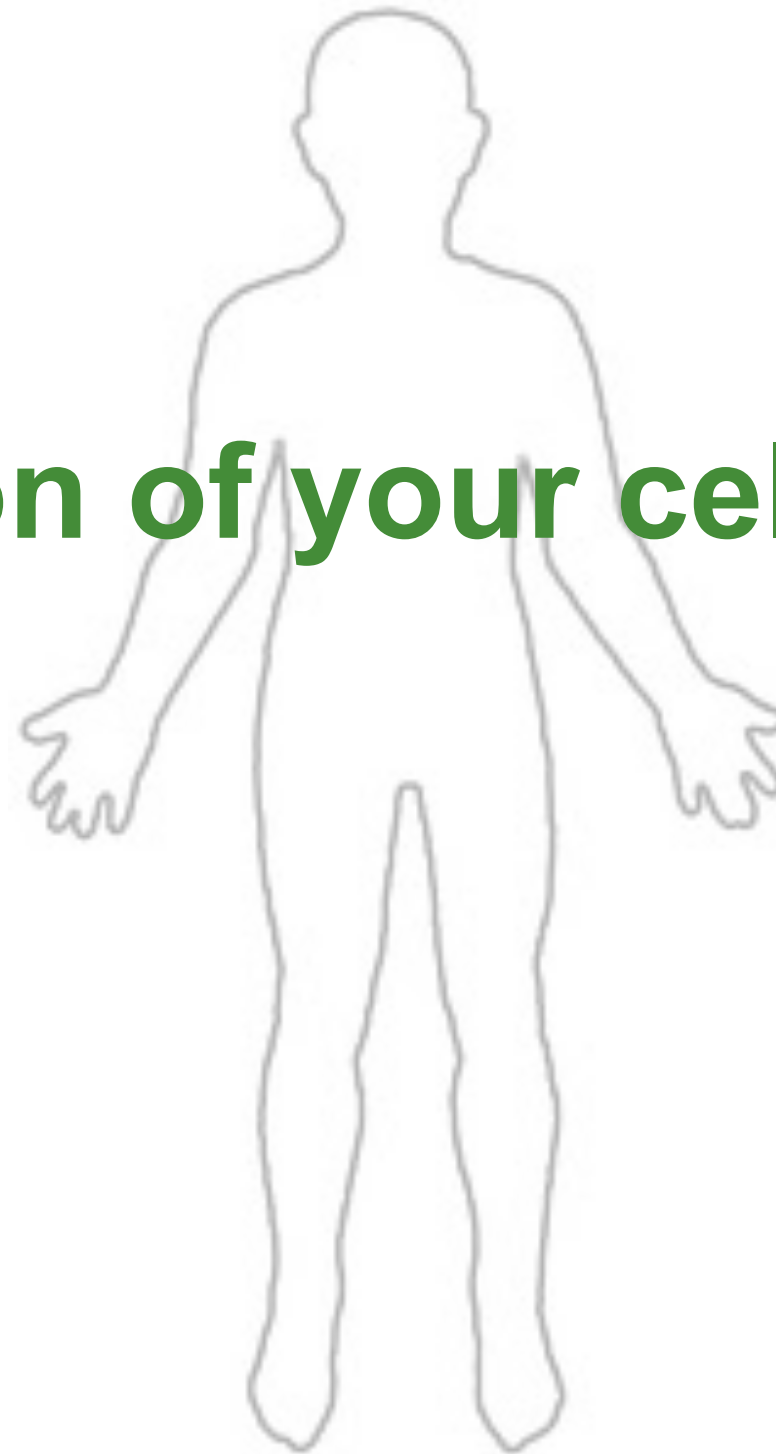
If services are small...

- Regularly rewrite using a better technology stack
- Adapt system to changing requirements and better technology without a total rewrite
- Pick the **best developers** rather than **best <pick a language> developers** \Rightarrow polyglot culture

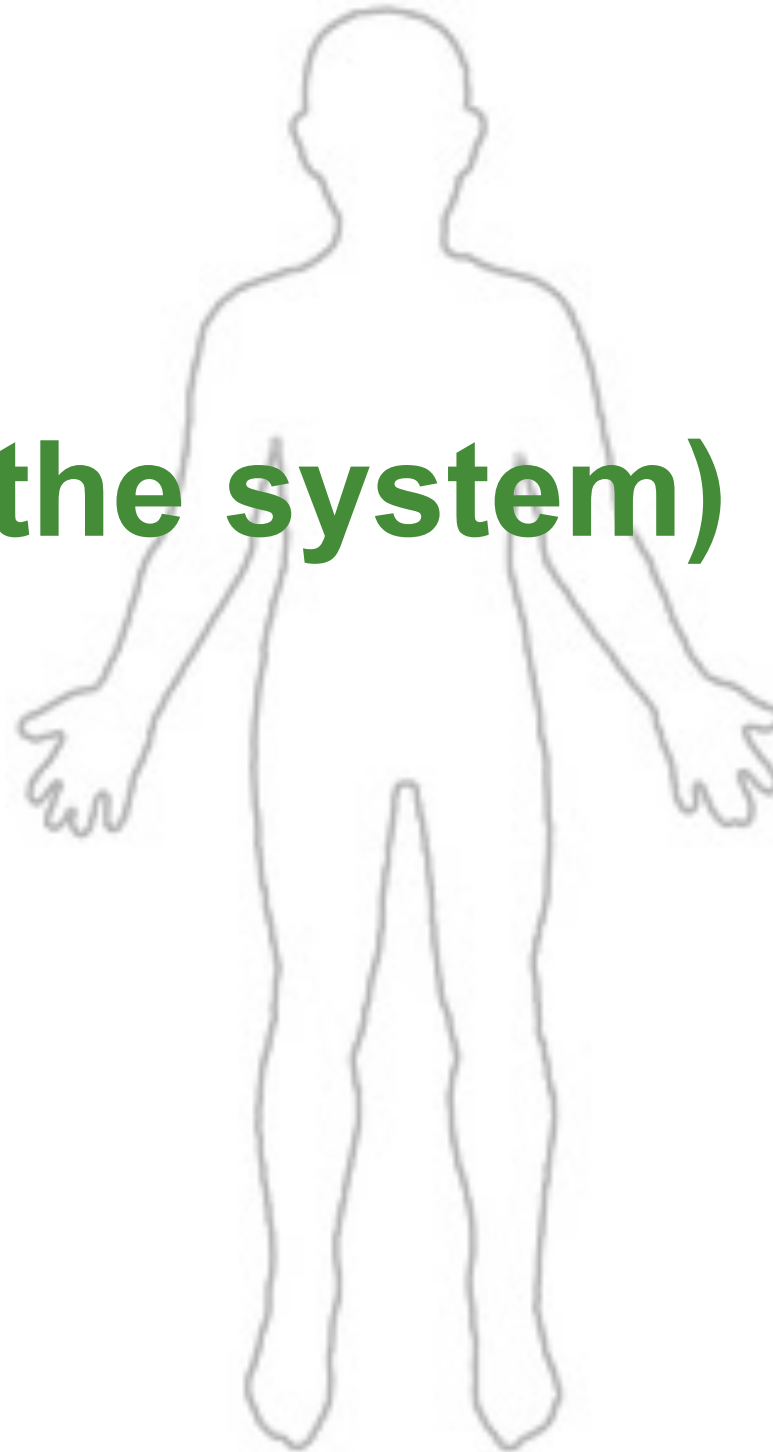
The human body as a system



50 to 70 billion of your cells die each day



Yet you (the system) remain you



Can we build software systems with these characteristics?

Can we build software systems with these characteristics?



[http://dreamsongs.com/Files/
DesignBeyondHumanAbilitiesSimp.pdf](http://dreamsongs.com/Files/DesignBeyondHumanAbilitiesSimp.pdf)

<http://dreamsongs.com/Files/WhitherSoftware.pdf>

Agenda

- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps

Inter-service communication options

- Synchronous HTTP \Leftrightarrow asynchronous AMQP
- Formats: JSON, XML, Protocol Buffers, Thrift, ...
- Even via the database

Inter-service communication options

- Synchronous HTTP \Leftrightarrow asynchronous AMQP
- Formats: JSON, XML, Protocol Buffers, Thrift, ...
- Even via the database

Asynchronous is preferred

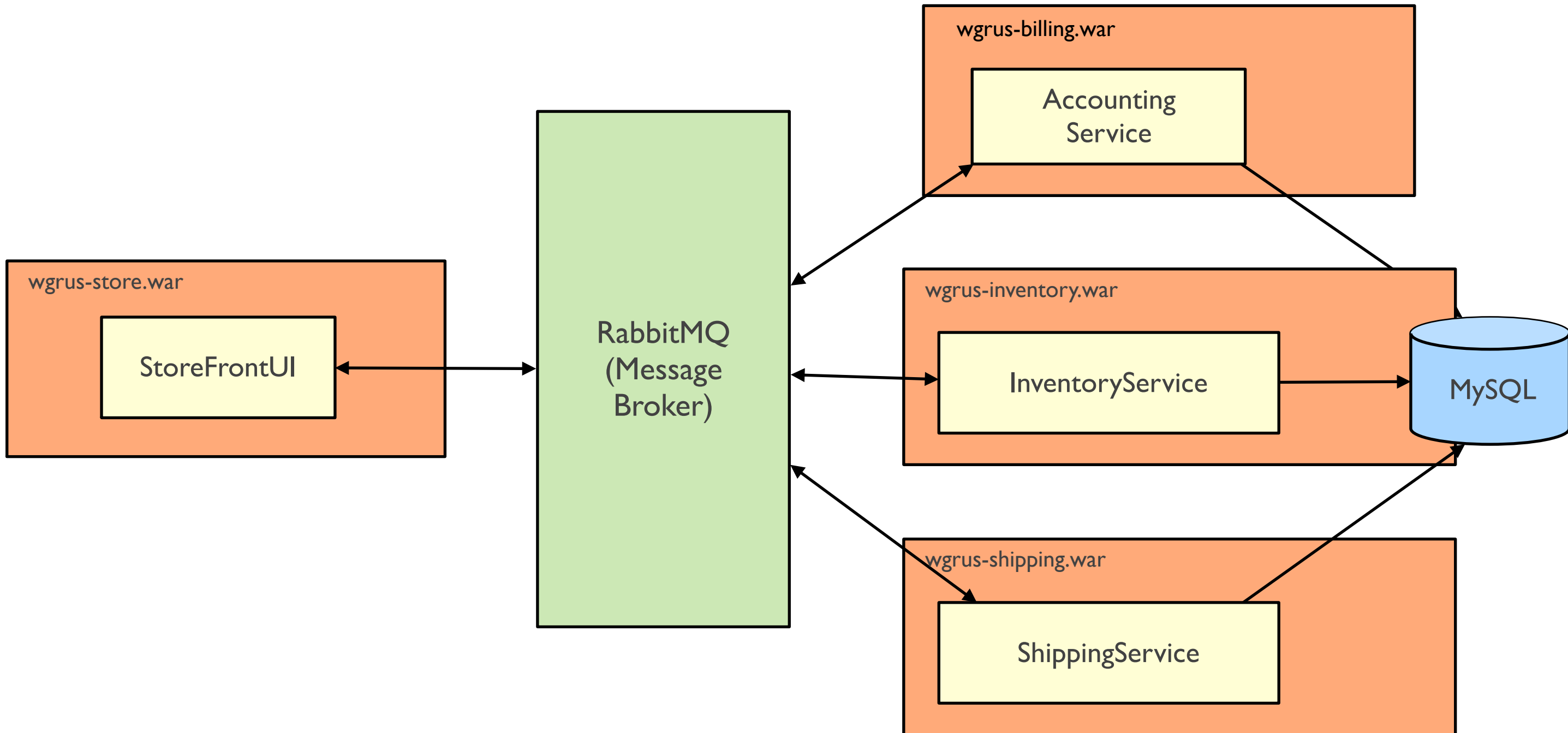
Inter-service communication options

- Synchronous HTTP \Leftrightarrow asynchronous AMQP
- Formats: JSON, XML, Protocol Buffers, Thrift, ...
- Even via the database

Asynchronous is preferred

JSON is fashionable but binary format is more efficient

Asynchronous message-based communication



Benefits

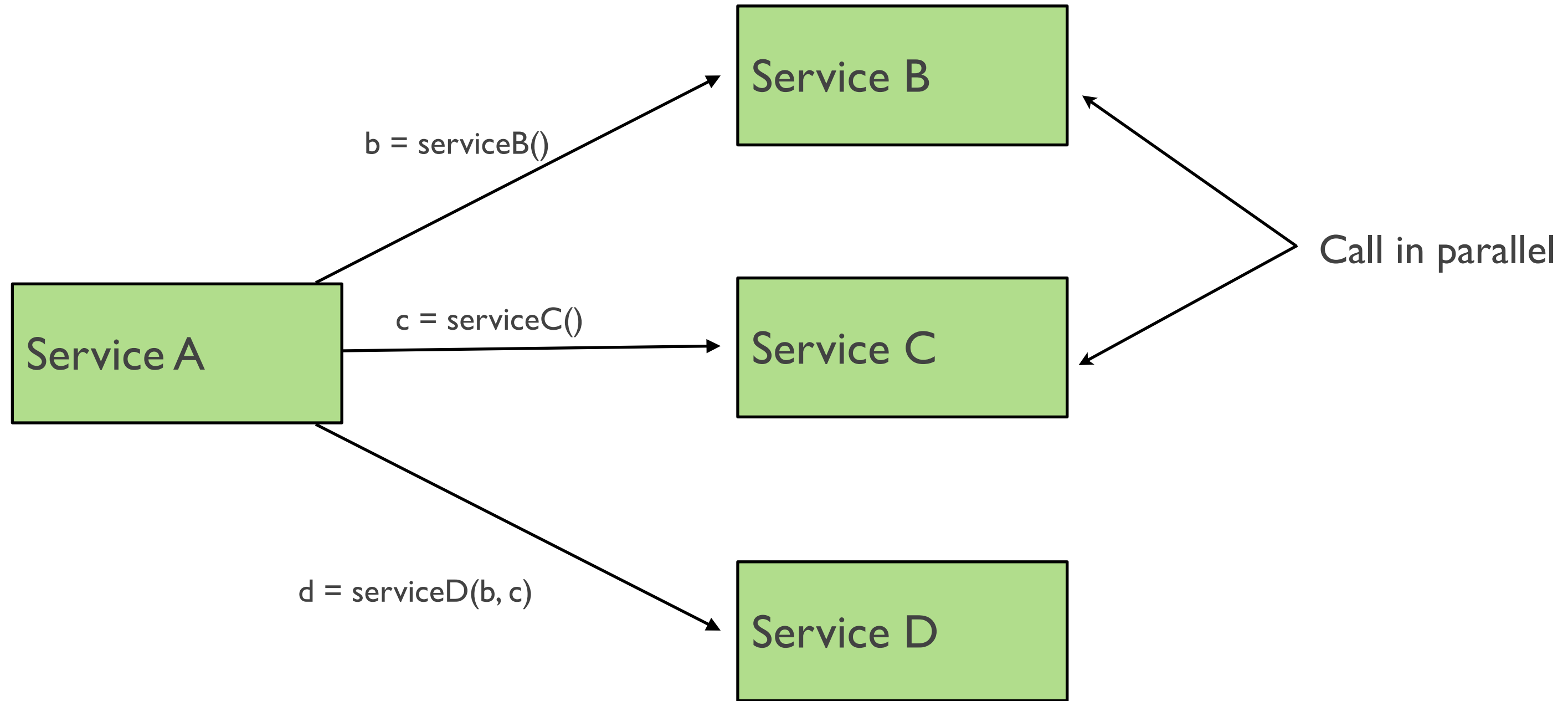
- Decouples caller from server
- Caller unaware of server's coordinates (URL)
- Message broker buffers message when server is down/slow
- Supports a variety of communication patterns, e.g. point-to-point, pub-sub, ...

Drawbacks

- Additional complexity of message broker
- Request/reply-style communication is more complex

Writing code that calls services

The need for parallelism



Java Futures are a great concurrency abstraction

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

Using Java Futures

```
public class Client {

    private ExecutorService executorService;
    private RemoteServiceProxy remoteServiceProxy;

    public void doSomething() throws ... {
        Future<Integer> result =
            executorService.submit(new Callable<Integer>() {
                @Override
                public Integer call() throws Exception {
                    return remoteServiceProxy.invokeRemoteService();
                }
            });

        /// Do other things

        int r = result.get(500, TimeUnit.MILLISECONDS);

        System.out.println(r);

    }
```

Using Java Futures

```
public class Client {  
  
    private ExecutorService executorService;  
    private RemoteServiceProxy remoteServiceProxy;  
  
    public void doSomething() throws ... {  
        Future<Integer> result =  
            executorService.submit(new Callable<Integer>() {  
                @Override  
                public Integer call() throws Exception {  
                    return remoteServiceProxy.invokeRemoteService();  
                }  
            });  
  
        /// Do other things  
  
        int r = result.get(500, TimeUnit.MILLISECONDS);  
  
        System.out.println(r);  
  
    }  
}
```

Eventually contains result

Using Java Futures

```
public class Client {  
  
    private ExecutorService executorService;  
    private RemoteServiceProxy remoteServiceProxy;  
  
    public void doSomething() throws ... {  
        Future<Integer> result =  
            executorService.submit(new Callable<Integer>() {  
                @Override  
                public Integer call() throws Exception {  
                    return remoteServiceProxy.invokeRemoteService();  
                }  
            });  
  
        /// Do other things  
  
        int r = result.get(500, TimeUnit.MILLISECONDS);  
  
        System.out.println(r);  
    }  
}
```

Eventually contains result

When needed wait for result

**Akka's composable futures are
even better**

Composable Futures

```
val f1 = Future { ... ; 1 }  
val f2 = Future { ... ; 2 }  
  
val f4 = f2.map(_ * 2)  
assertEquals(4, Await.result(f4, 1 second))  
  
val fzip = f1 zip f2  
assertEquals((1, 2), Await.result(fzip, 1 second))
```

Composable Futures

```
val f1 = Future { ... ; 1 }  
val f2 = Future { ... ; 2 }  
  
val f4 = f2.map(_ * 2)  
assertEquals(4, Await.result(f4, 1 second))  
  
val fzip = f1 zip f2  
assertEquals((1, 2), Await.result(fzip, 1 second))
```



Transforms Future

Composable Futures

```
val f1 = Future { ... ; 1 }  
val f2 = Future { ... ; 2 }
```

```
val f4 = f2.map(_ * 2)  
assertEquals(4, Await.result(f4, 1 second))
```

Transforms Future

```
val fzip = f1 zip f2  
assertEquals((1, 2), Await.result(fzip, 1 second))
```

Combines two futures

Using Akka futures

```
def callB() : Future[...] = ...
def callC() : Future[...] = ...
def callD() : Future[...] = ...

val future = for {
  (b, c) <- callB() zip callC();
  d <- callD(b, c)
} yield d

val result = Await.result(future, 1 second)
```

<http://doc.akka.io/docs/akka/2.0.1/scala/futures.html>

Using Akka futures

```
def callB() : Future[...] = ...  
def callC() : Future[...] = ...  
def callD() : Future[...] = ...
```

```
val future = for {  
  (b, c) <- callB() zip callC();  
  d <- callD(b, c)  
} yield d
```

```
val result = Await.result(future, 1 second)
```

Two calls execute in parallel

<http://doc.akka.io/docs/akka/2.0.1/scala/futures.html>

Using Akka futures

```
def callB() : Future[...] = ...  
def callC() : Future[...] = ...  
def callD() : Future[...] = ...
```

Two calls execute in parallel

```
val future = for {  
  (b, c) <- callB() zip callC();  
  d <- callD(b, c)  
} yield d
```

And then invokes D

```
val result = Await.result(future, 1 second)
```

<http://doc.akka.io/docs/akka/2.0.1/scala/futures.html>

Using Akka futures

```
def callB() : Future[...] = ...  
def callC() : Future[...] = ...  
def callD() : Future[...] = ...
```

Two calls execute in parallel

```
val future = for {  
  (b, c) <- callB() zip callC();  
  d <- callD(b, c)  
} yield d
```

And then invokes D

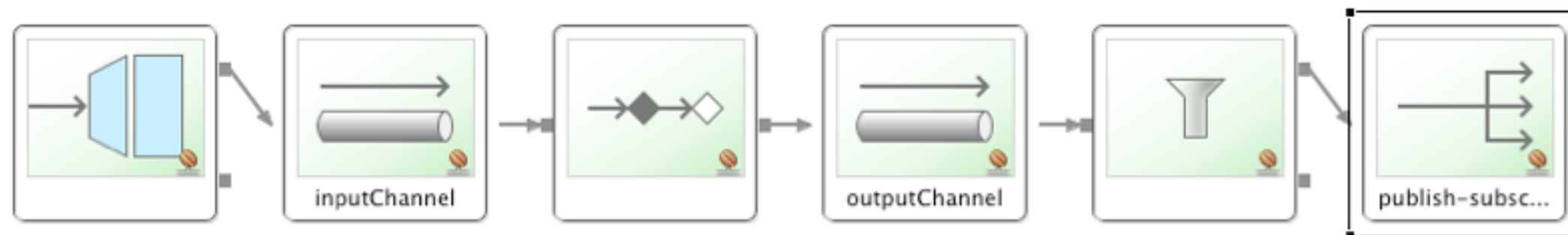
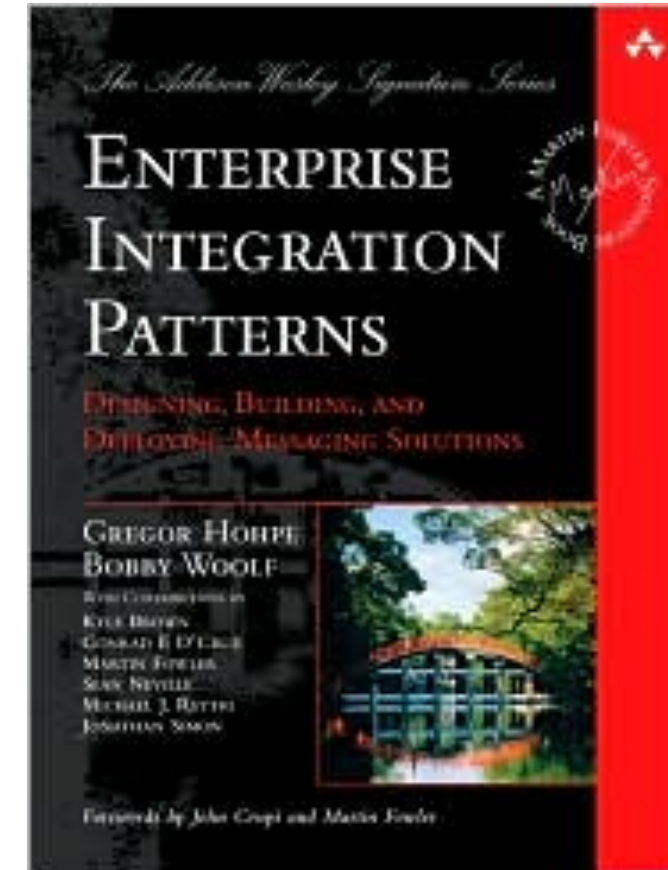
```
val result = Await.result(future, 1 second)
```

Get the result of D

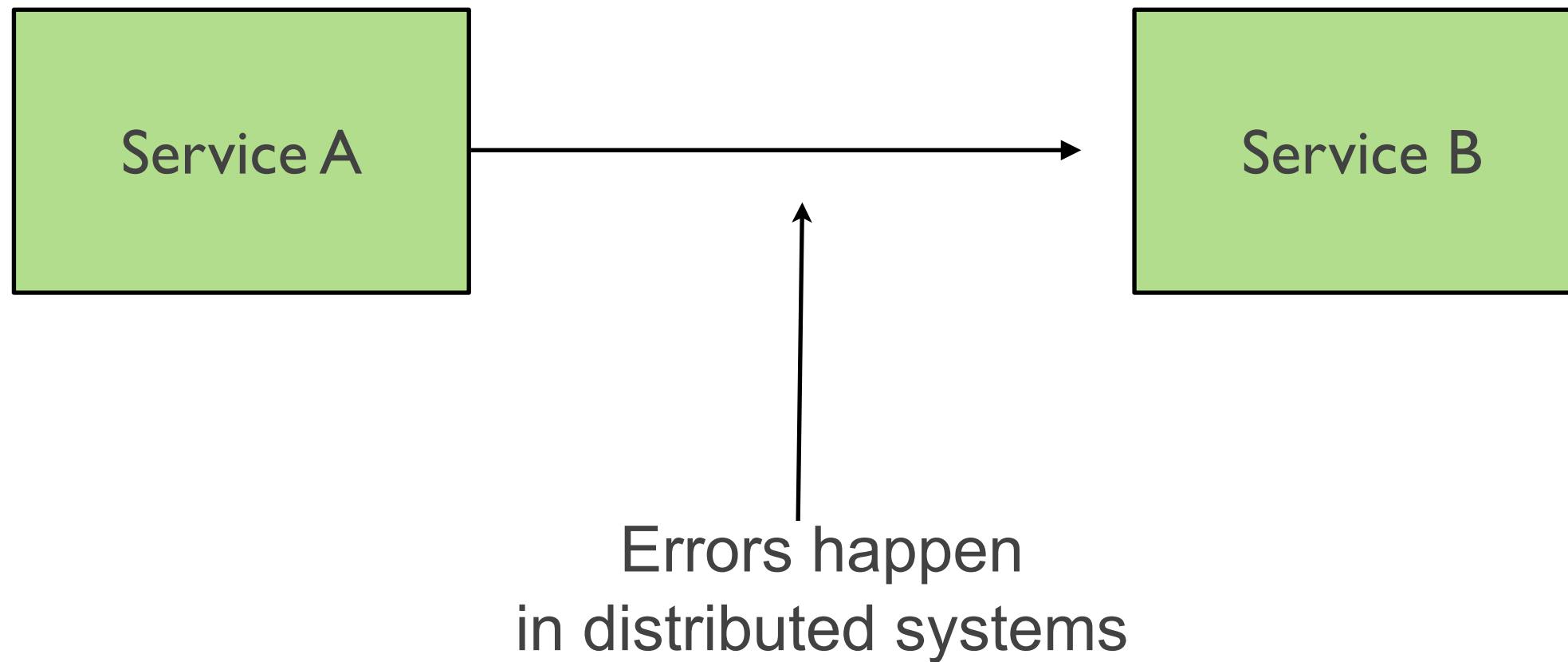
<http://doc.akka.io/docs/akka/2.0.1/scala/futures.html>

Spring Integration

- Provides the building blocks for a pipes and filters architecture
- Enables development of application components that are
 - loosely coupled
 - insulated from messaging infrastructure
- Messaging defined declaratively



Handling failure



About Netflix

About Netflix

> 1B API calls/day

About Netflix

> 1B API calls/day

1 API call \Rightarrow average 6 service calls

About Netflix

> 1B API calls/day

1 API call \Rightarrow average 6 service calls

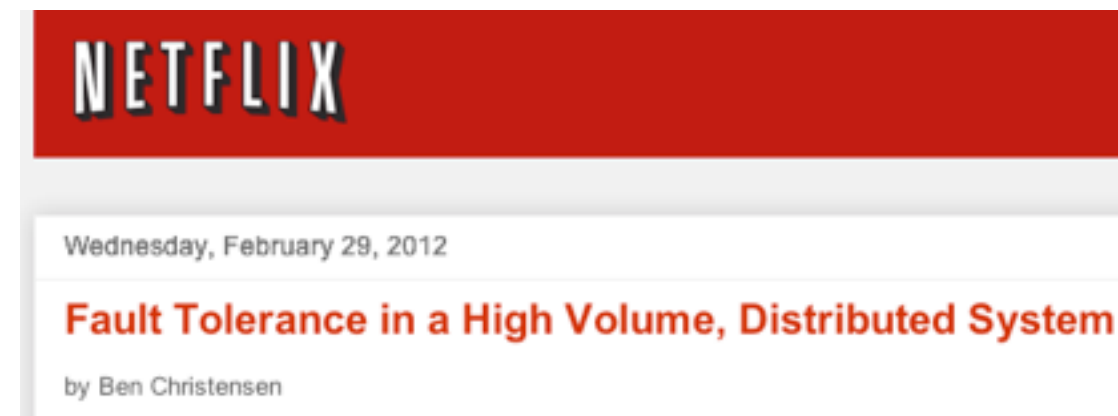
Fault tolerance is essential

About Netflix

> 1B API calls/day

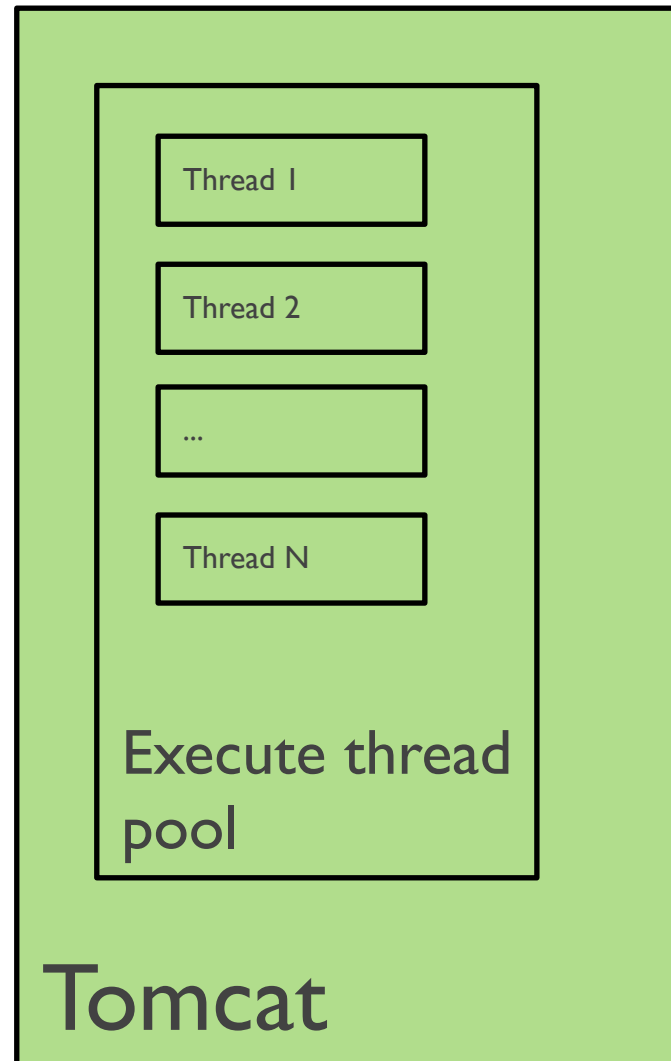
1 API call \Rightarrow average 6 service calls

Fault tolerance is essential

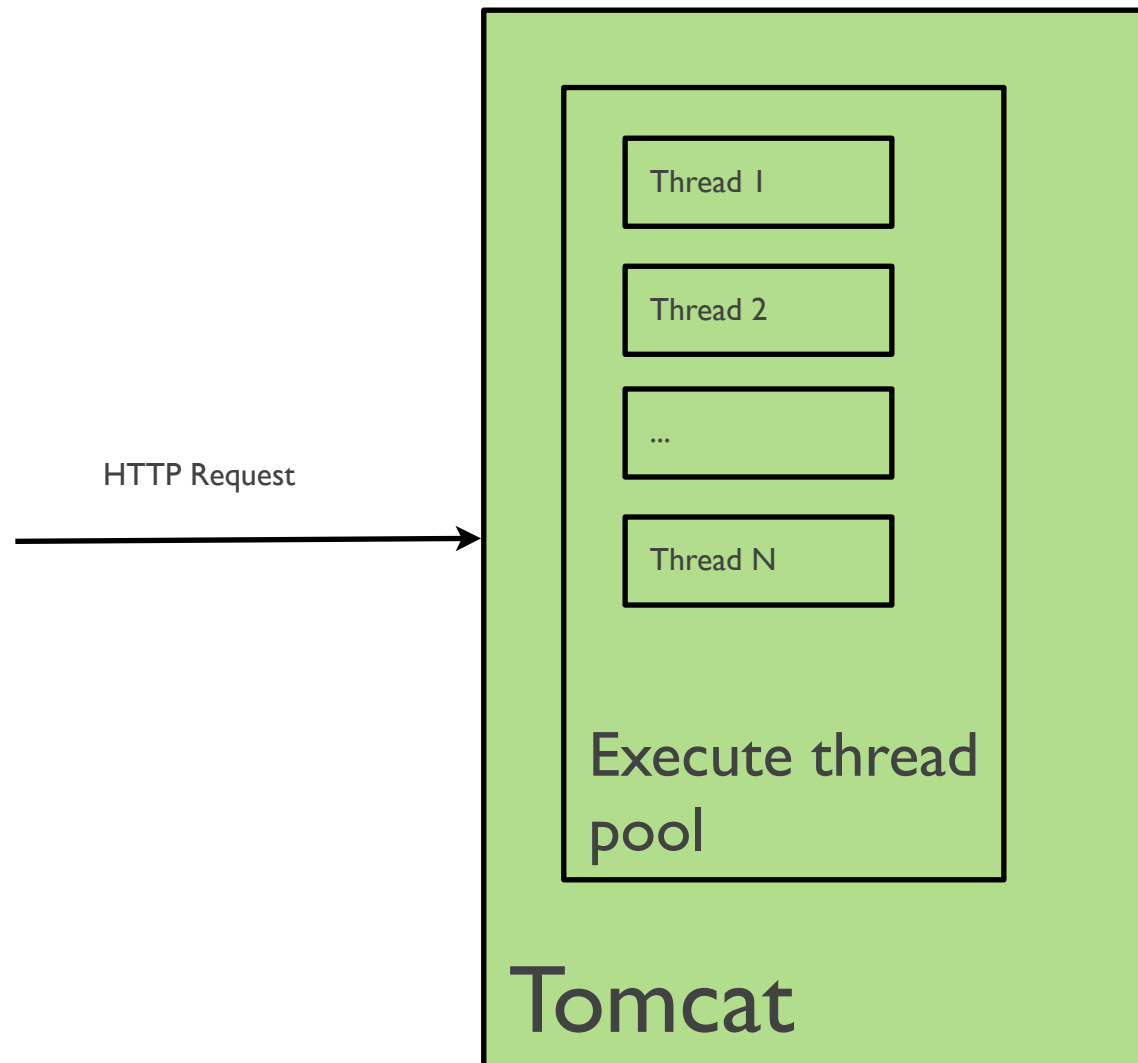


<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

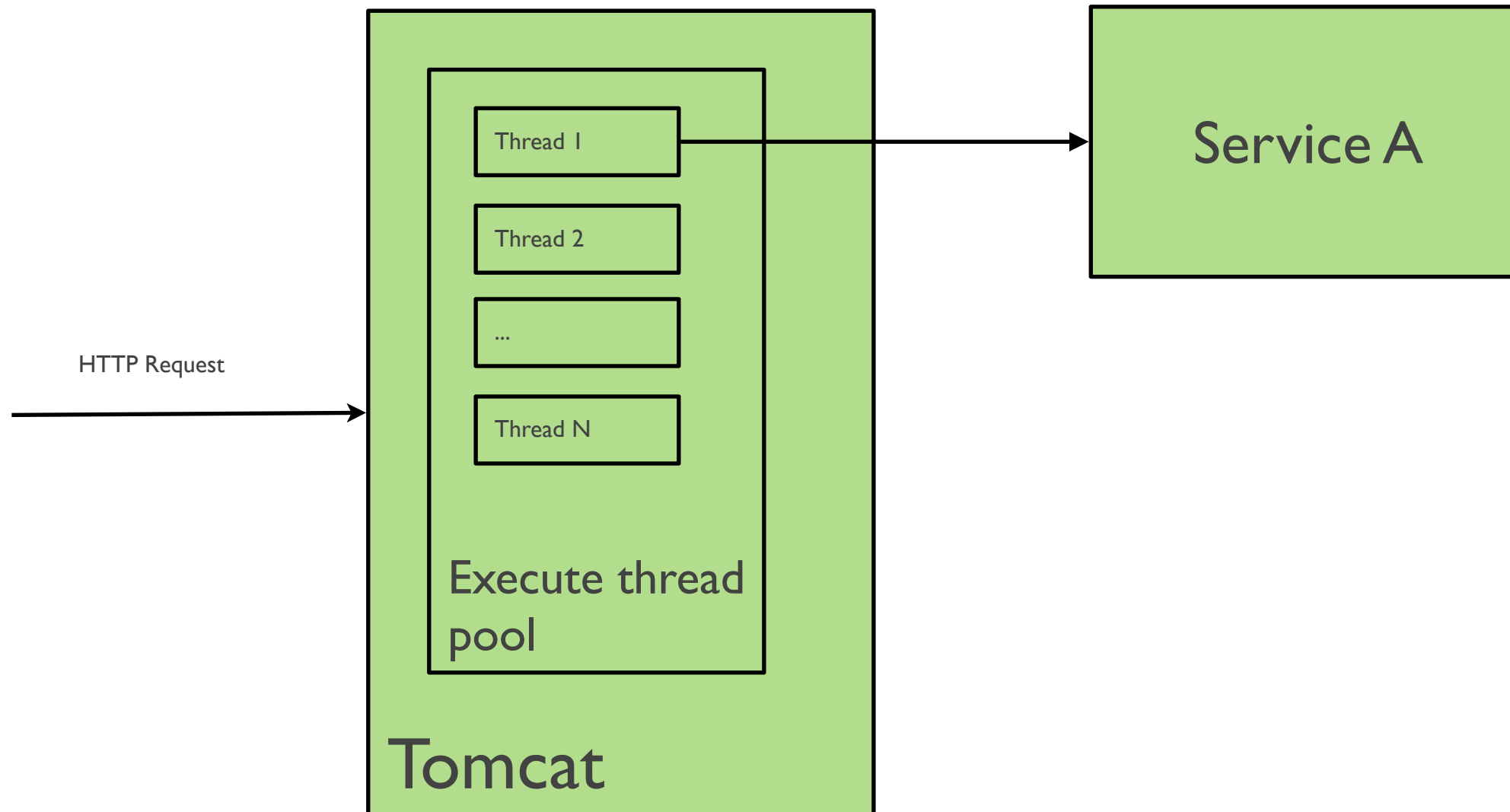
How to run out of threads



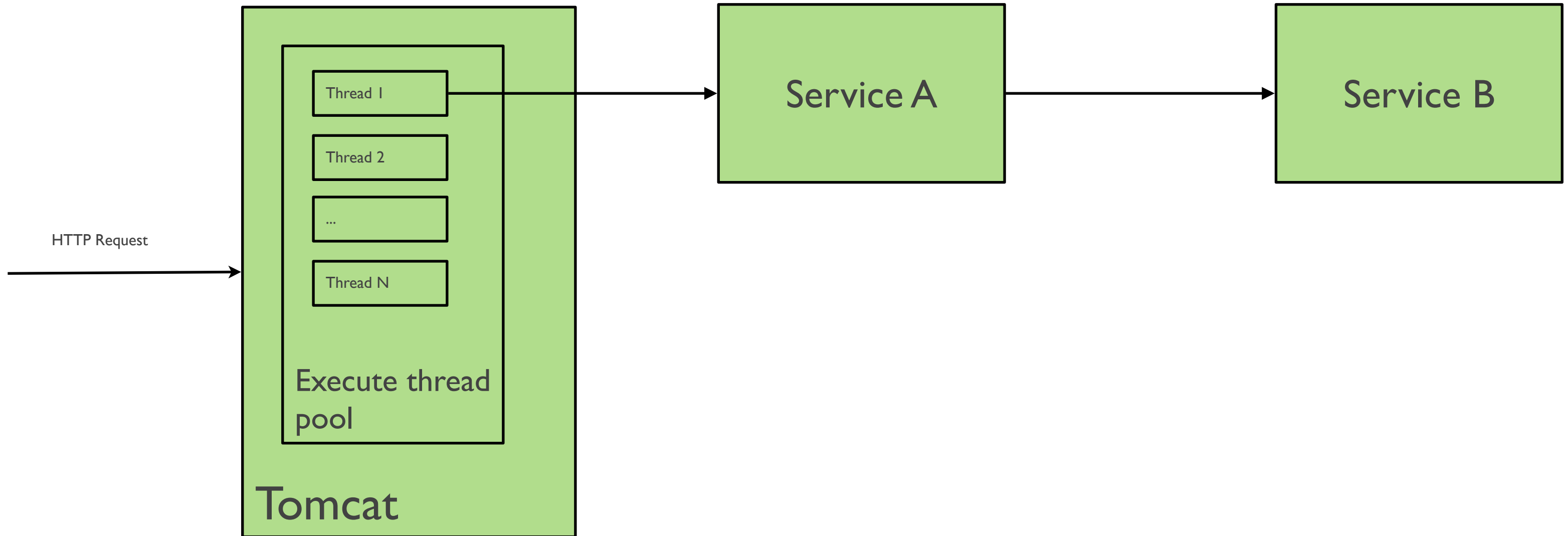
How to run out of threads



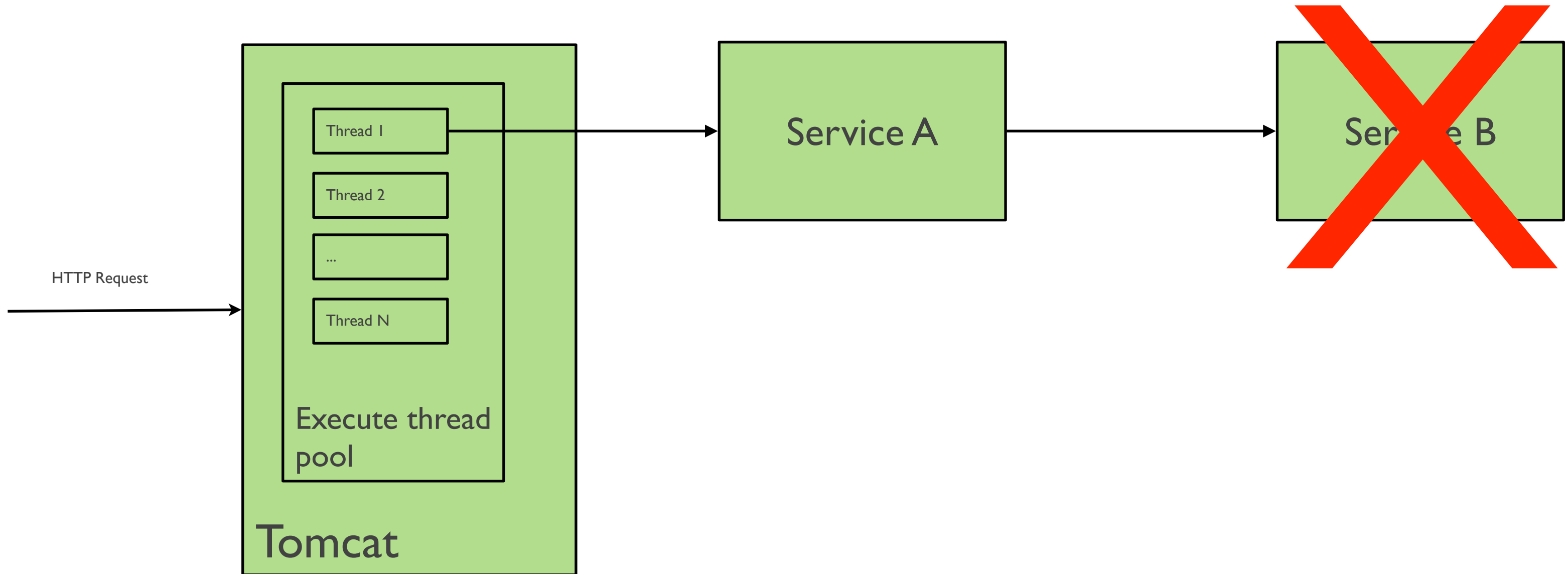
How to run out of threads



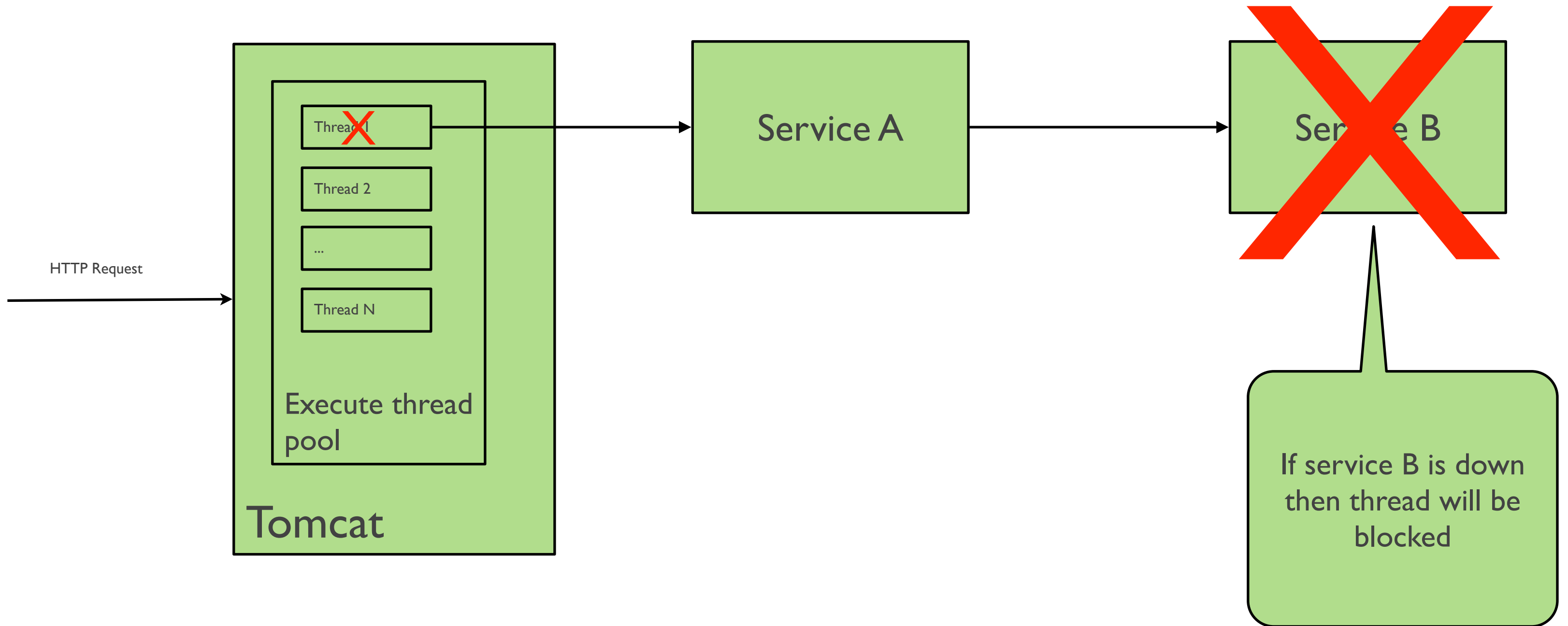
How to run out of threads



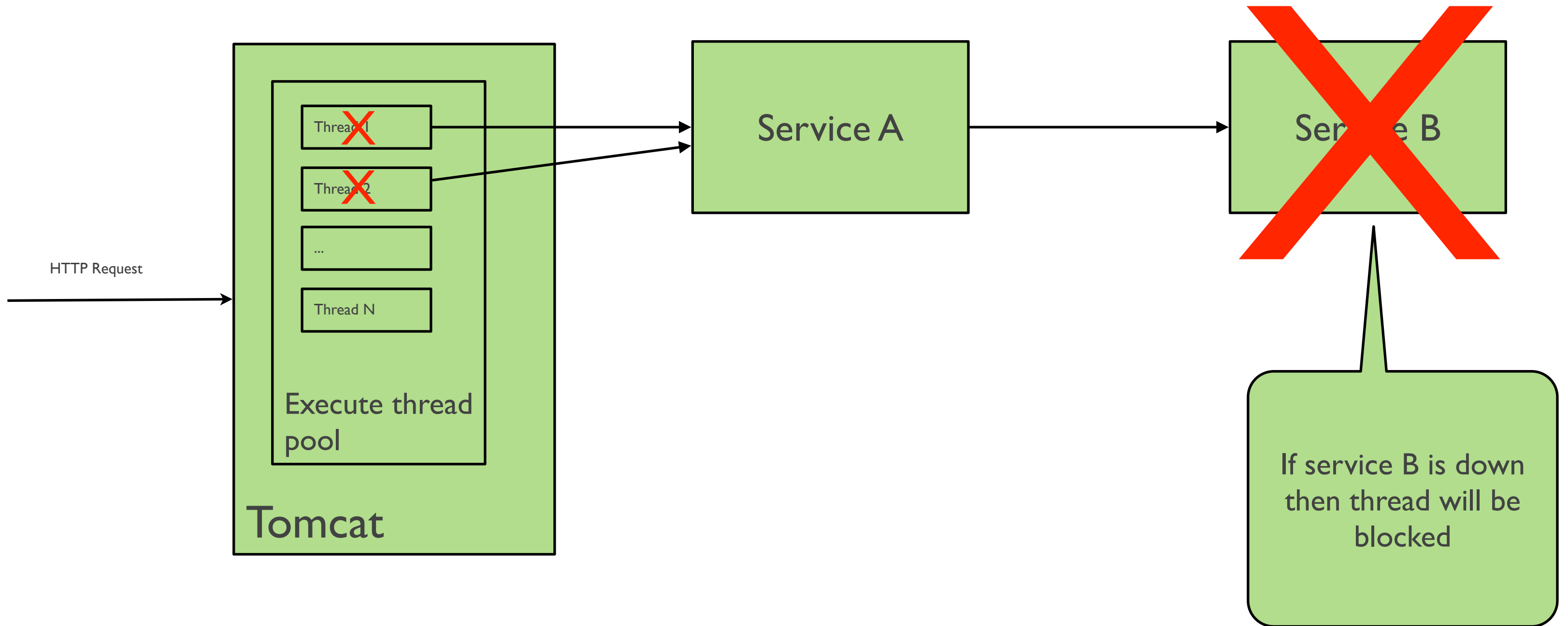
How to run out of threads



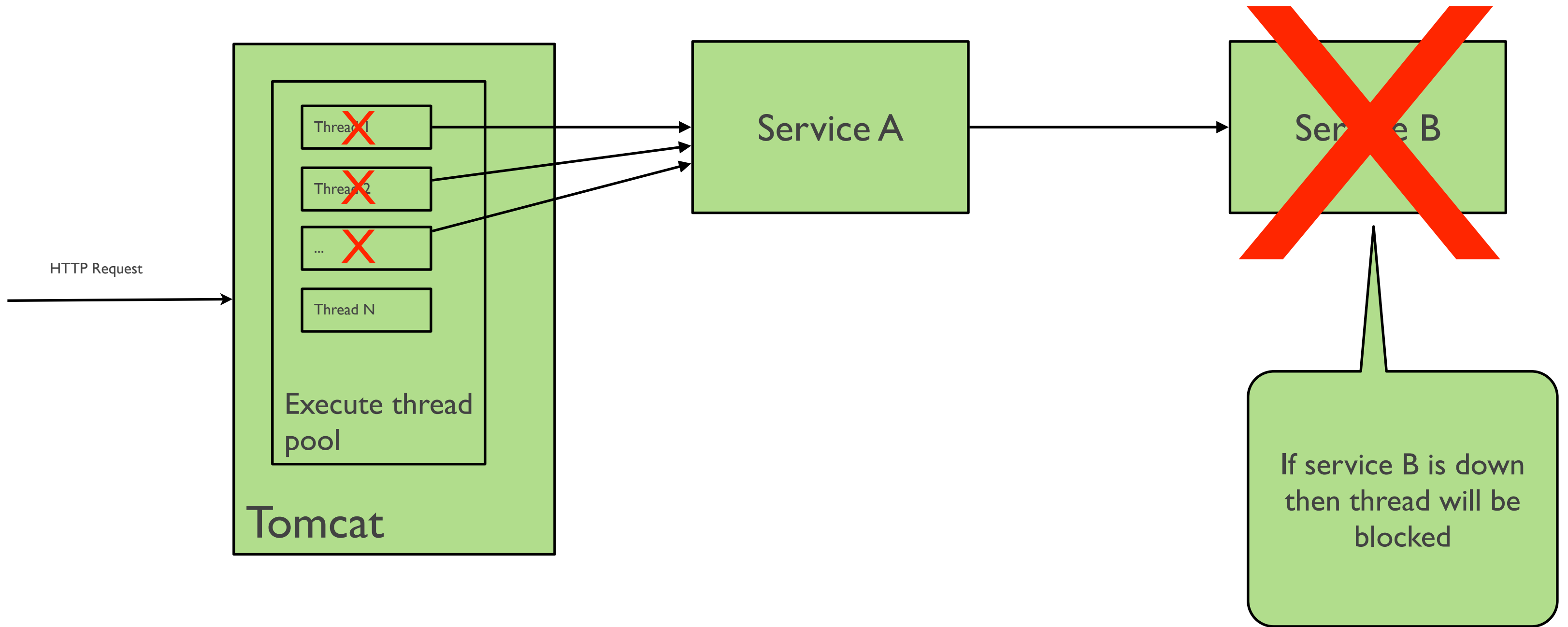
How to run out of threads



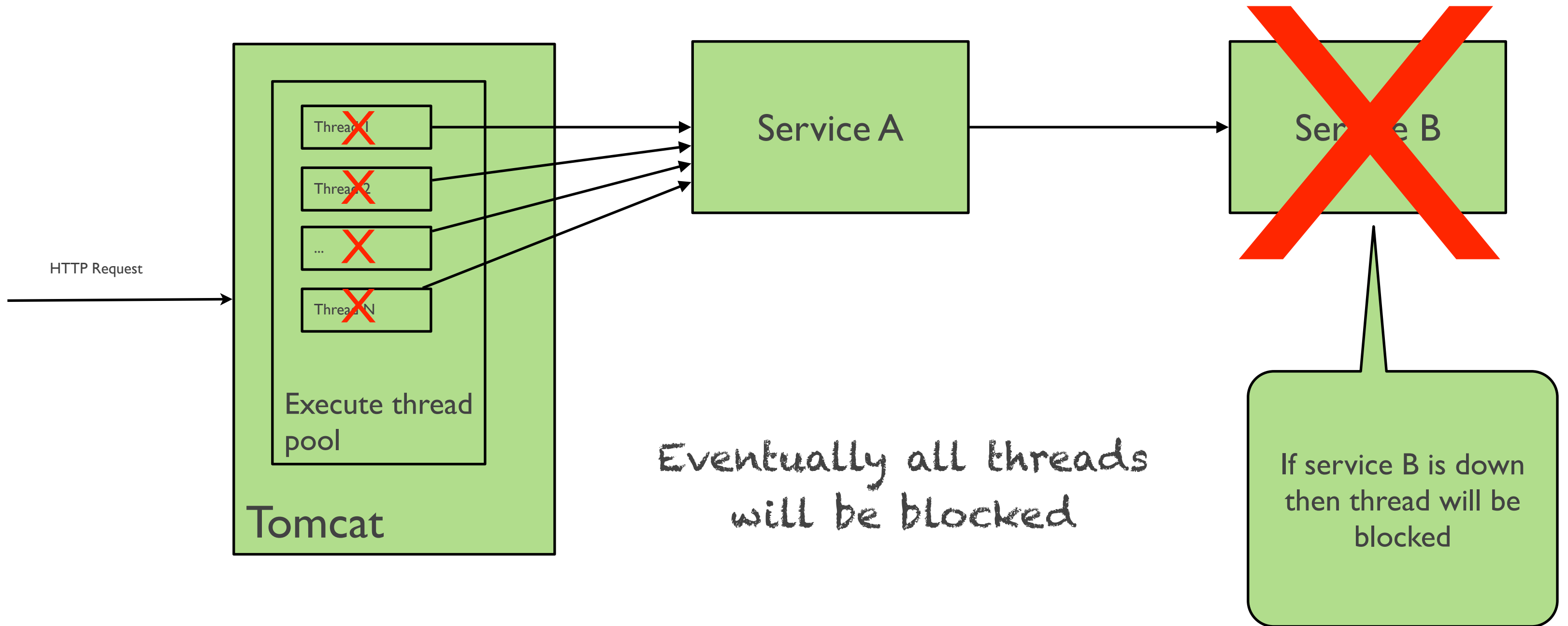
How to run out of threads



How to run out of threads



How to run out of threads



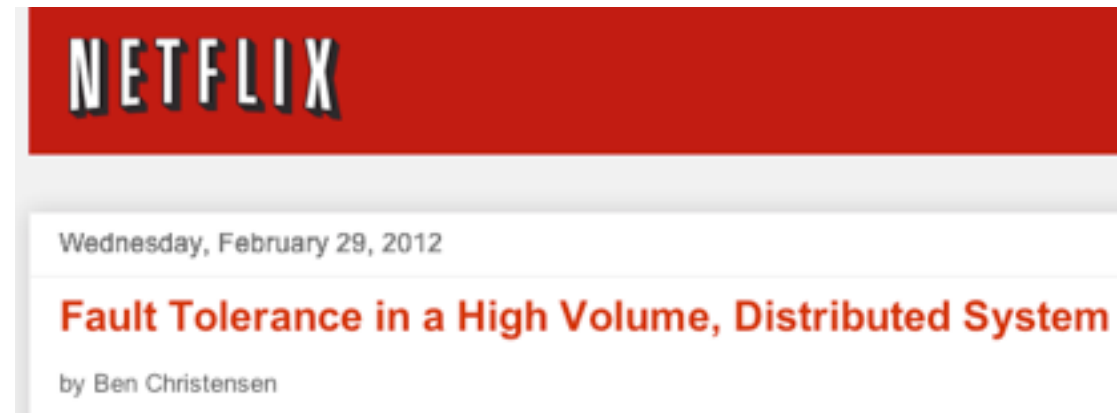
Use timeouts and retries



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Use timeouts and retries

Never wait forever



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Use timeouts and retries

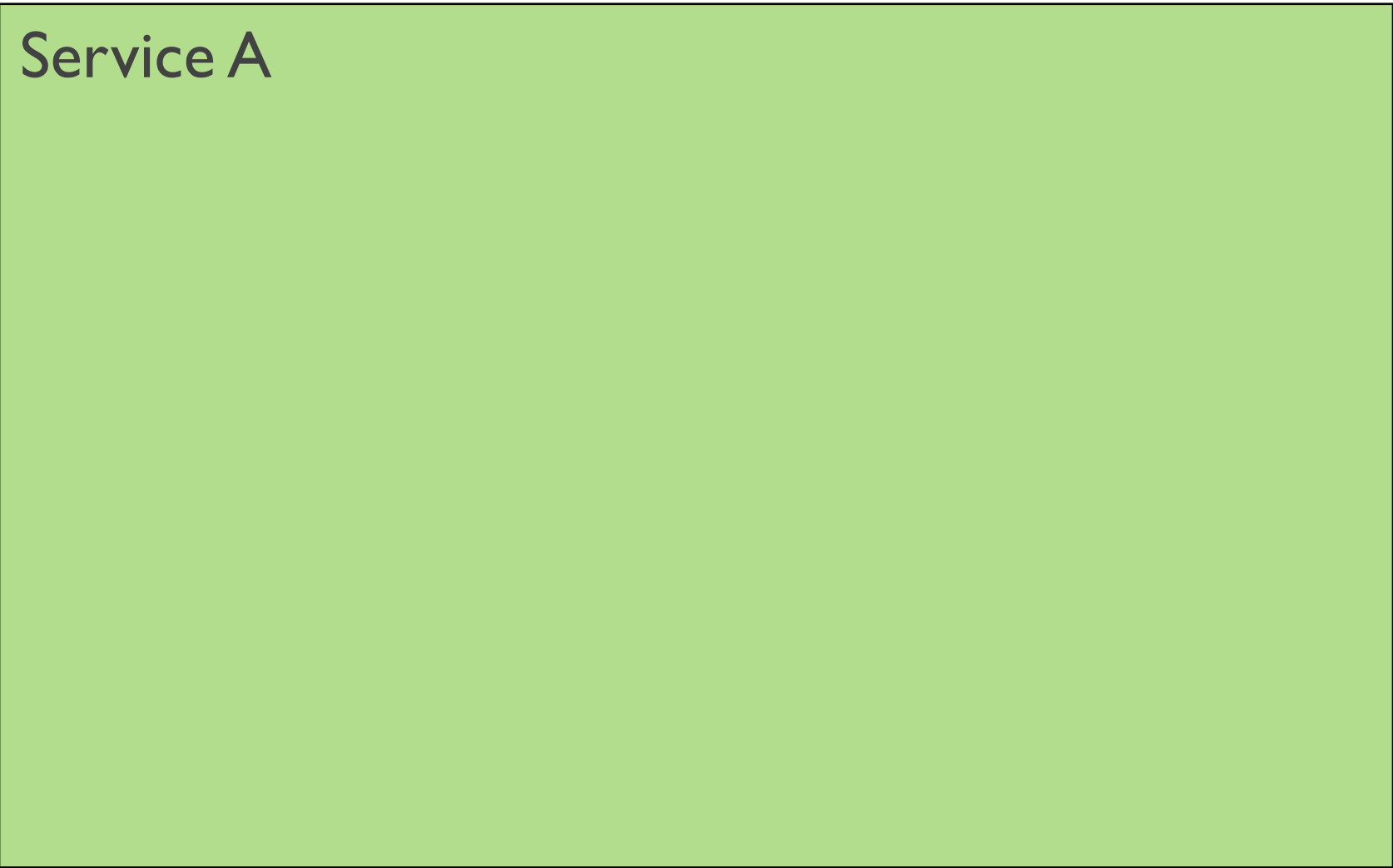
Never wait forever

Errors can be transient \Rightarrow retry

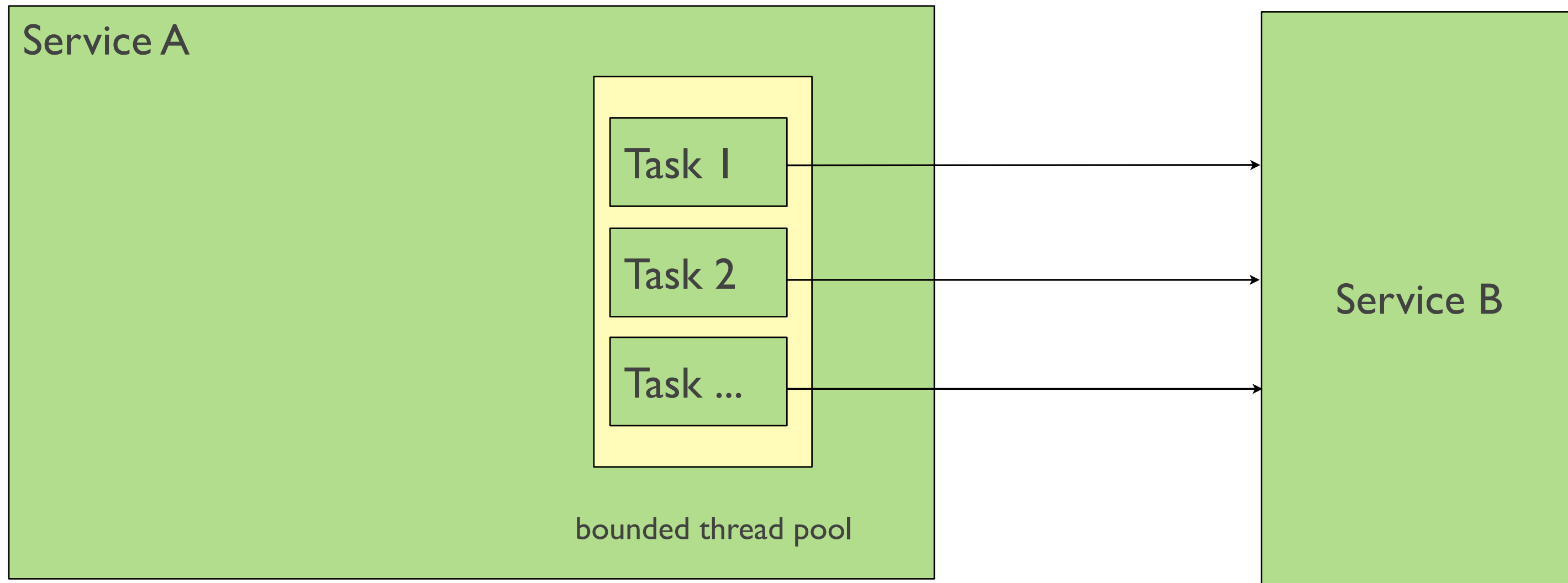


<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

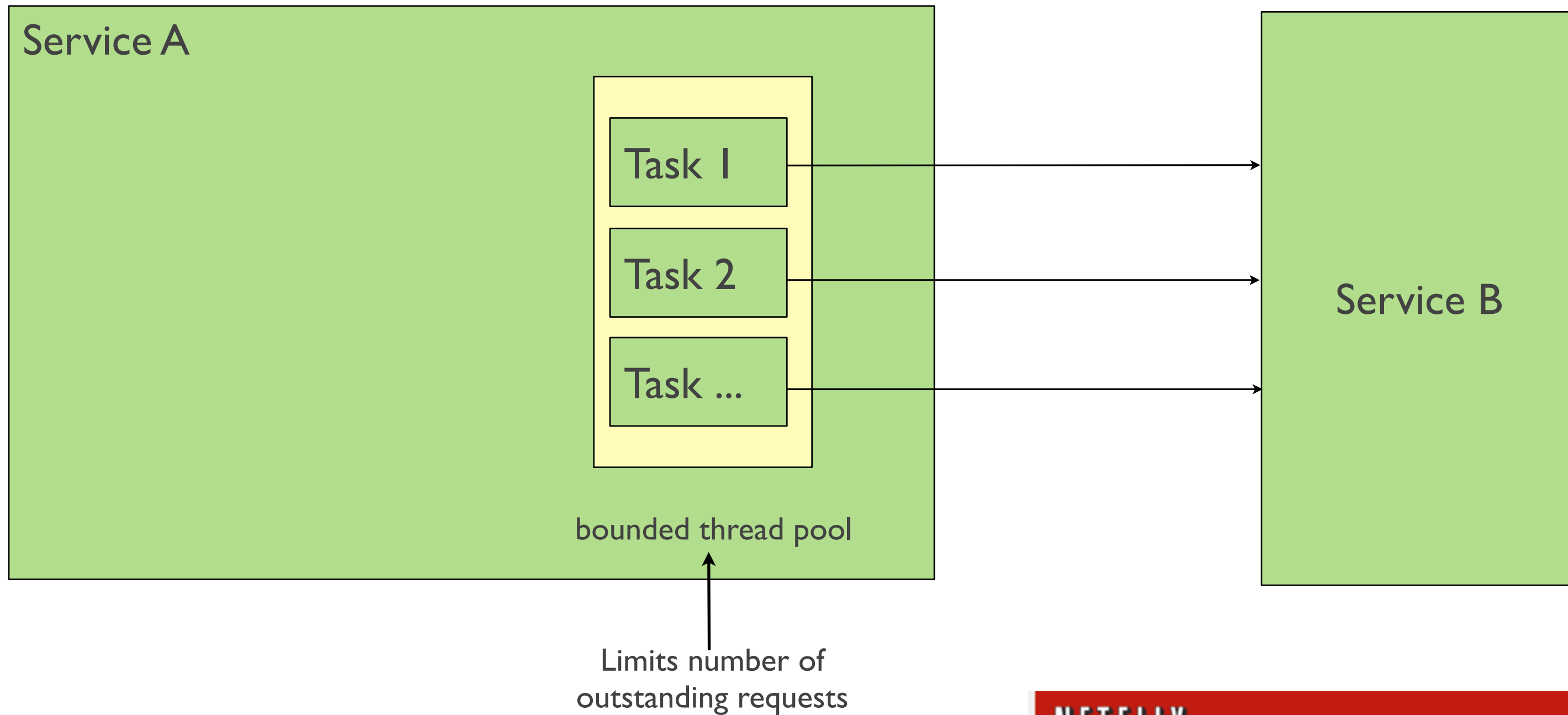
Use per-dependency bounded thread pool



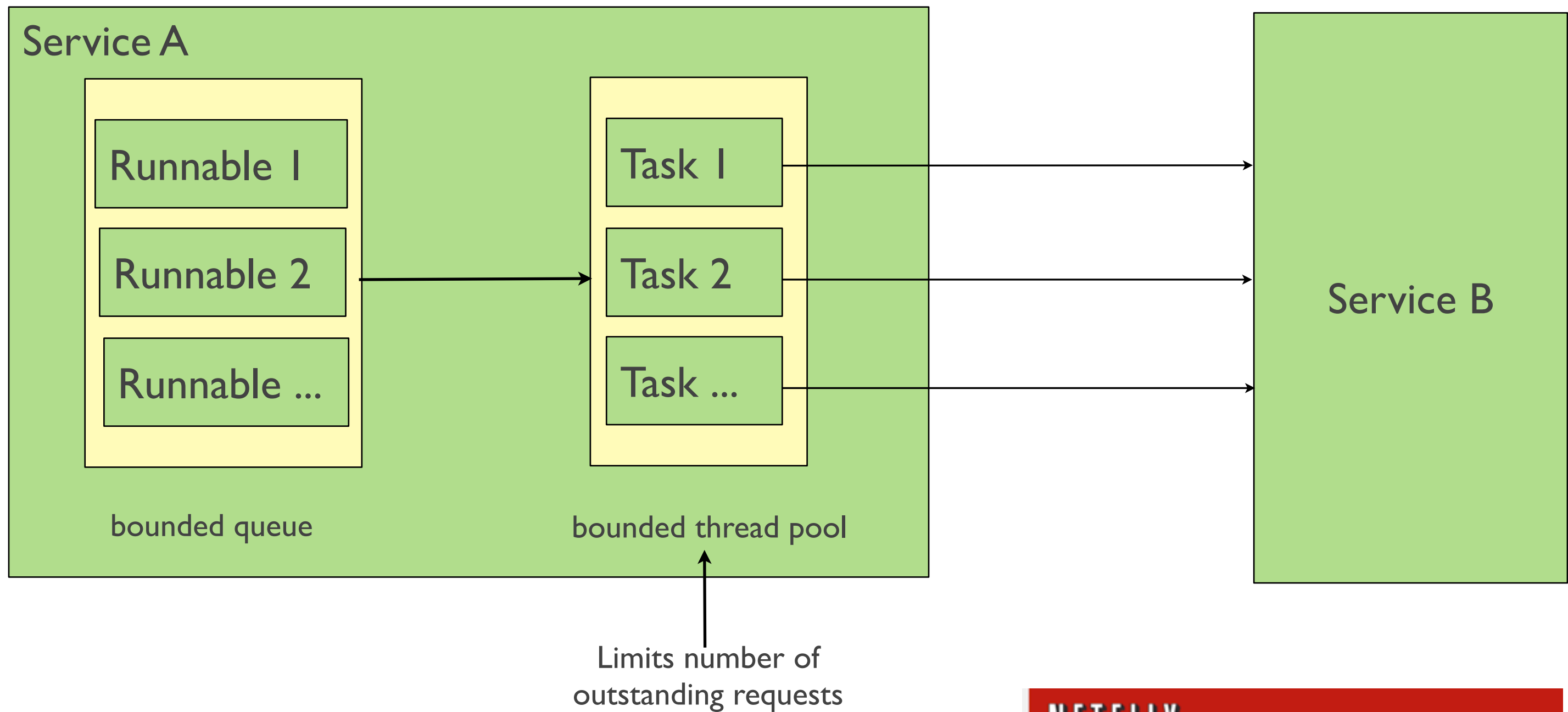
Use per-dependency bounded thread pool



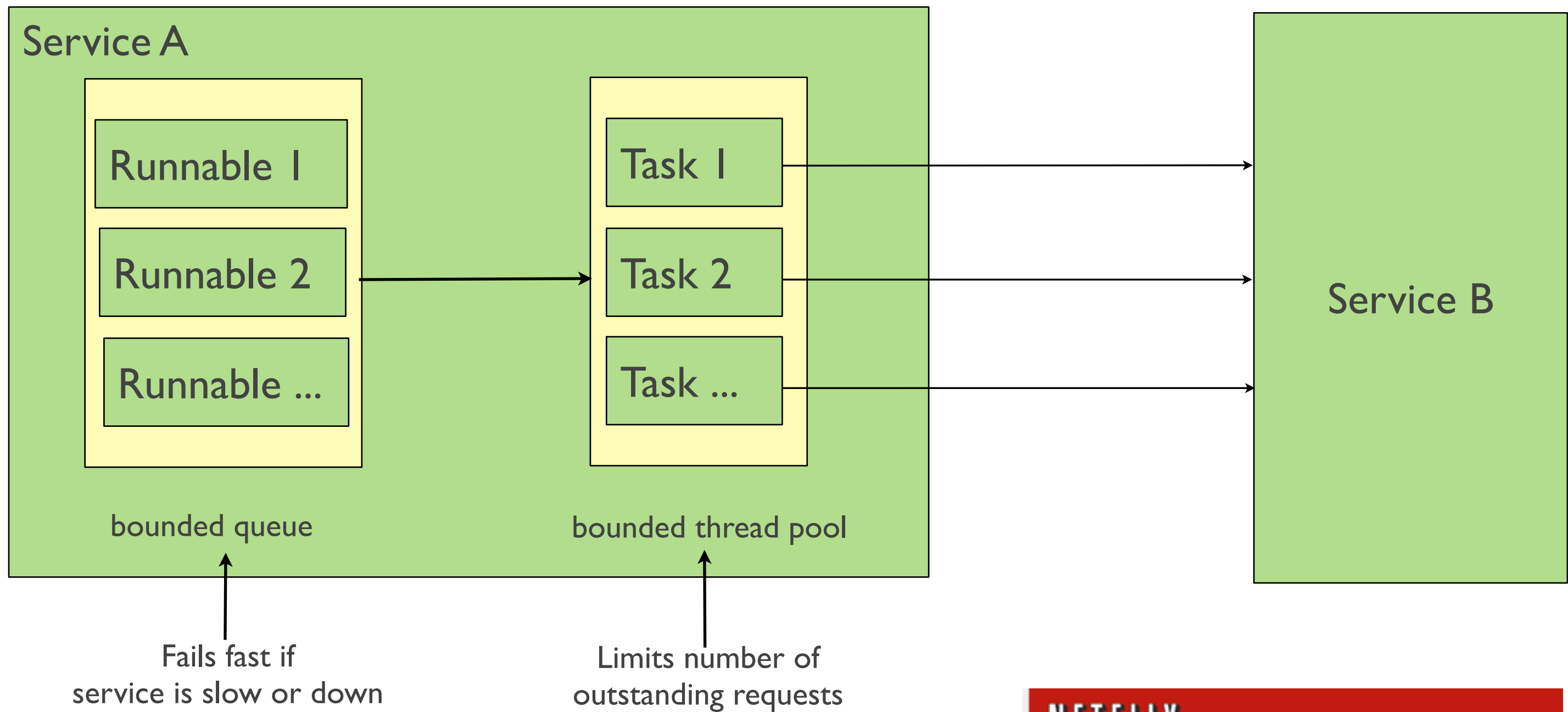
Use per-dependency bounded thread pool



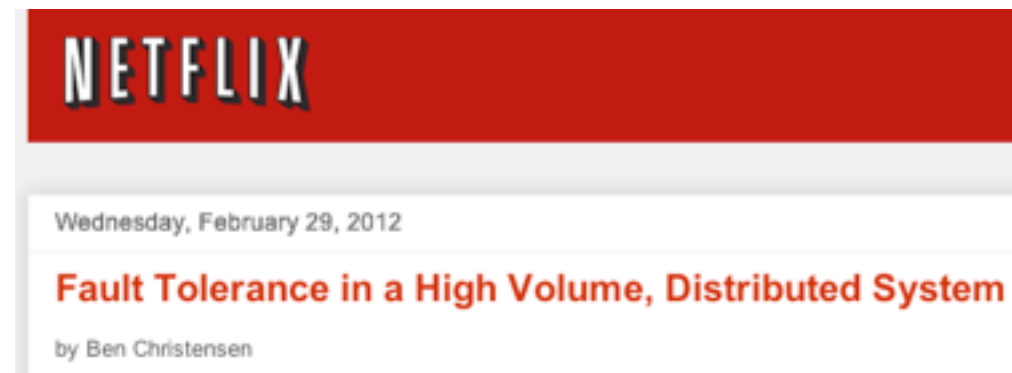
Use per-dependency bounded thread pool



Use per-dependency bounded thread pool



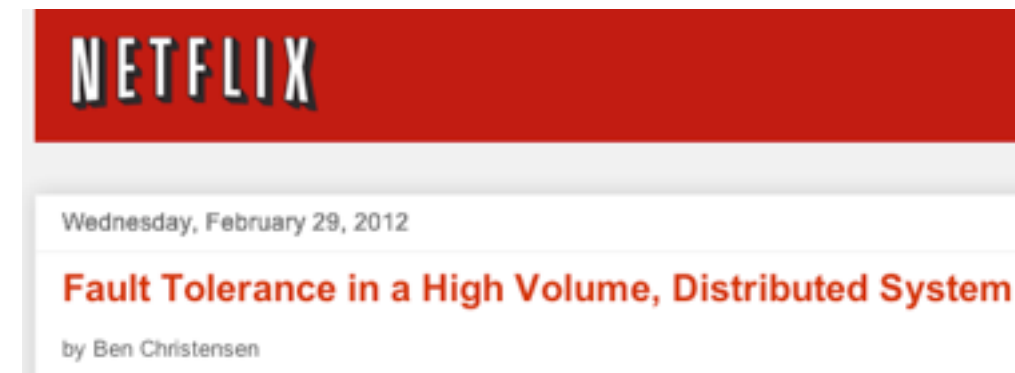
Use a circuit breaker



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Use a circuit breaker

High error rate \Rightarrow stop calling temporarily

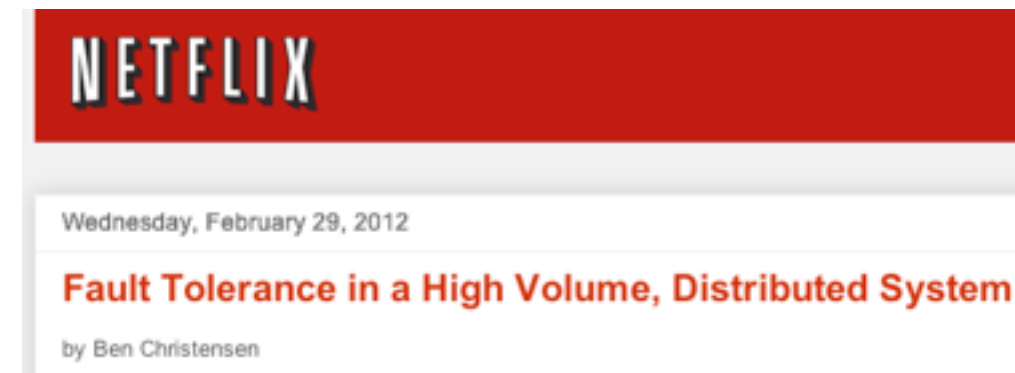


<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Use a circuit breaker

High error rate \Rightarrow stop calling temporarily

Down \Rightarrow wait for it to come back up



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Use a circuit breaker

High error rate \Rightarrow stop calling temporarily

Down \Rightarrow wait for it to come back up

Slow \Rightarrow gives it a chance to recover



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

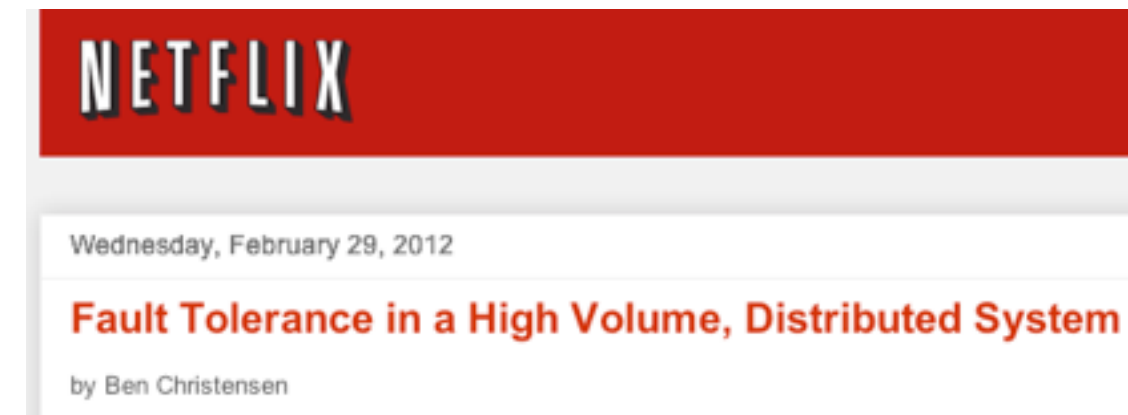
On failure



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

On failure

Return cached data

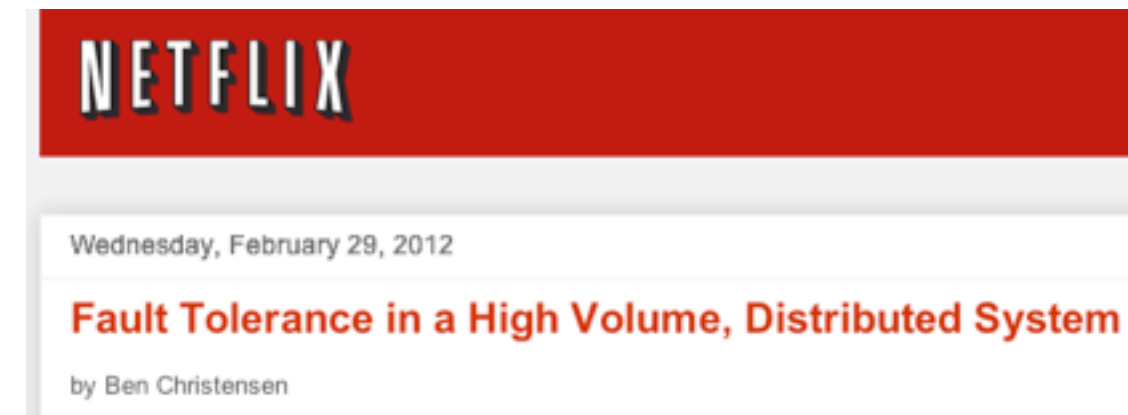


<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

On failure

Return cached data

Return default data



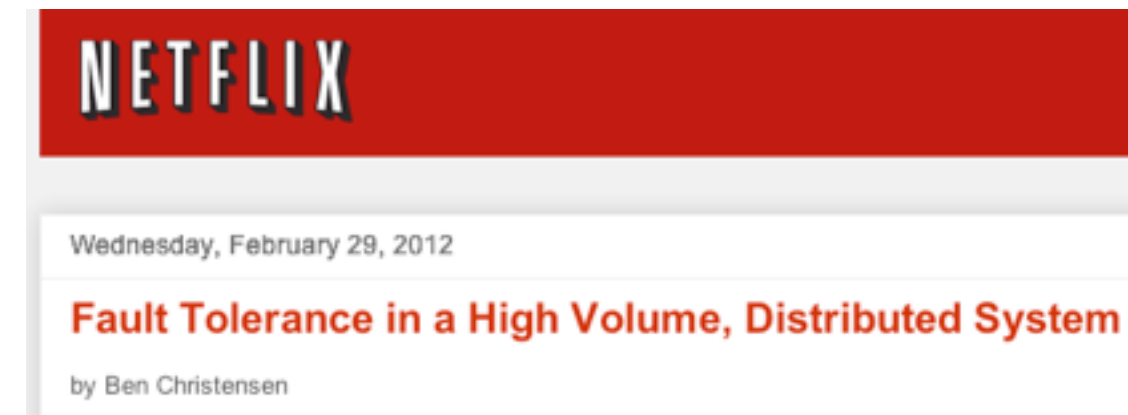
<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

On failure

Return cached data

Return default data

Fail fast



<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

On failure

Avoid
Failing

Return cached data

Return default data

Fail fast

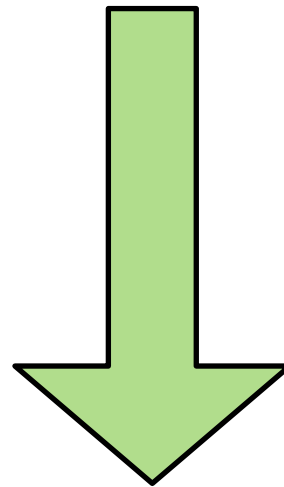


<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

Agenda

- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps

Modular application



Choice of presentation layer technology

NodeJS is the fashionable technology



Why NodeJS?



Node.js is a platform built on [Chrome's JavaScript runtime](#) for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

[DOWNLOAD](#)[DOCS](#)

v0.6.15

Why NodeJS?

- Familiar Javascript



Node.js is a platform built on [Chrome's JavaScript runtime](#) for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

[DOWNLOAD](#)[DOCS](#)

v0.6.15

Why NodeJS?

- Familiar Javascript
- High-performance, scalable event-driven, non-blocking I/O model



Why NodeJS?

- Familiar Javascript
- High-performance, scalable event-driven, non-blocking I/O model
- Compact runtime



Why NodeJS?

- Familiar Javascript
- High-performance, scalable event-driven, non-blocking I/O model
- Compact runtime
- Over 17,000 modules developed by the community



Why NodeJS?

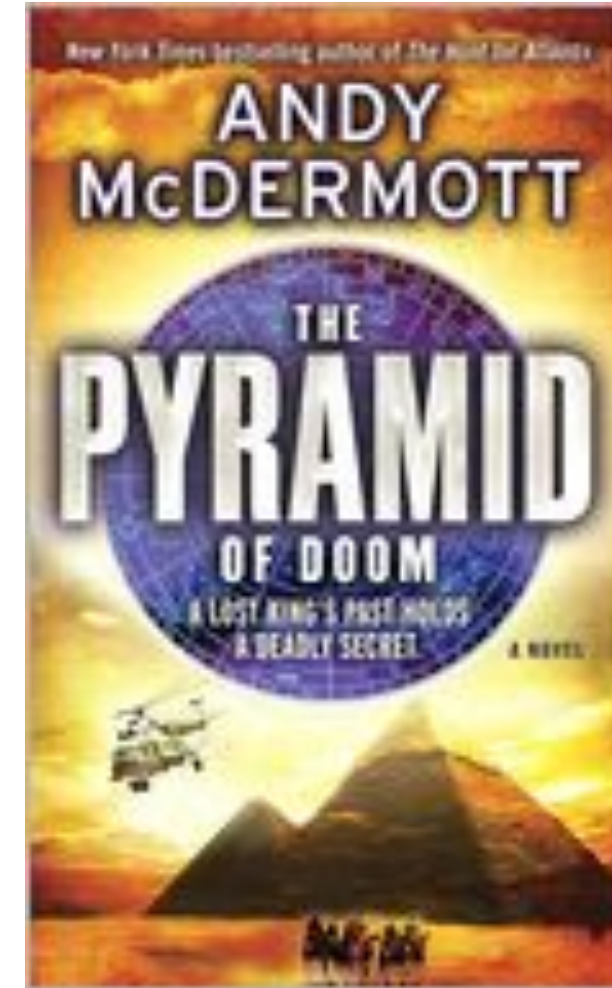
- Familiar Javascript
- High-performance, scalable event-driven, non-blocking I/O model
- Compact runtime
- Over 17,000 modules developed by the community
- Many JavaScript client frameworks have a NodeJS counterpart, e.g. socket.io and SockJS



Why not NodeJS?

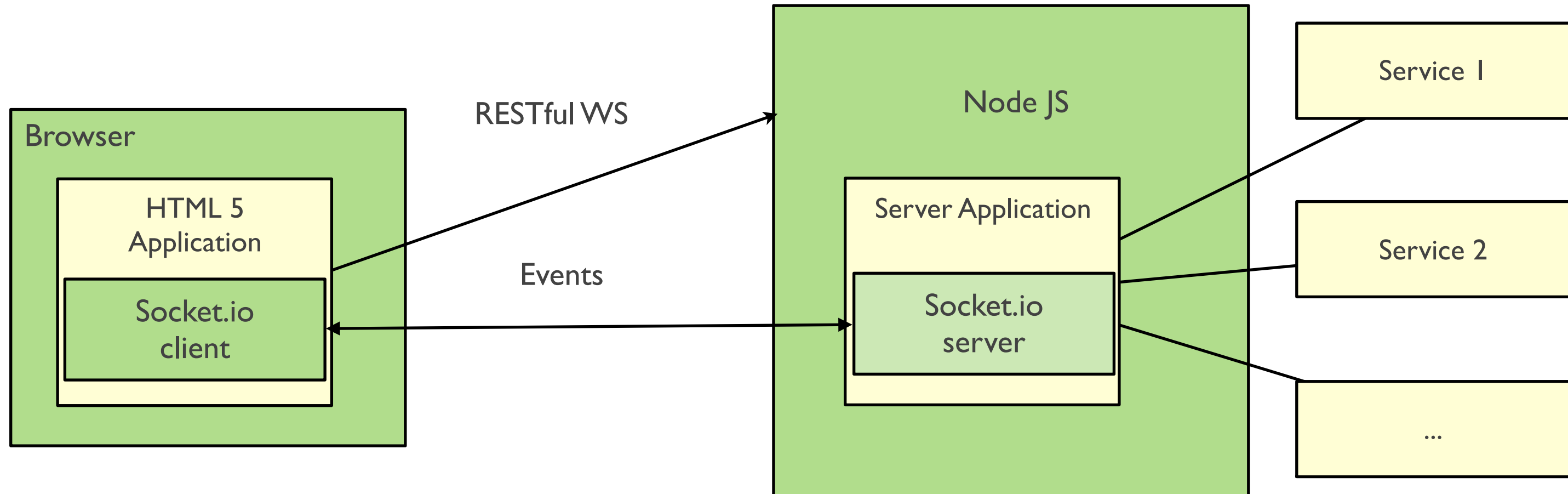


Why not NodeJS?

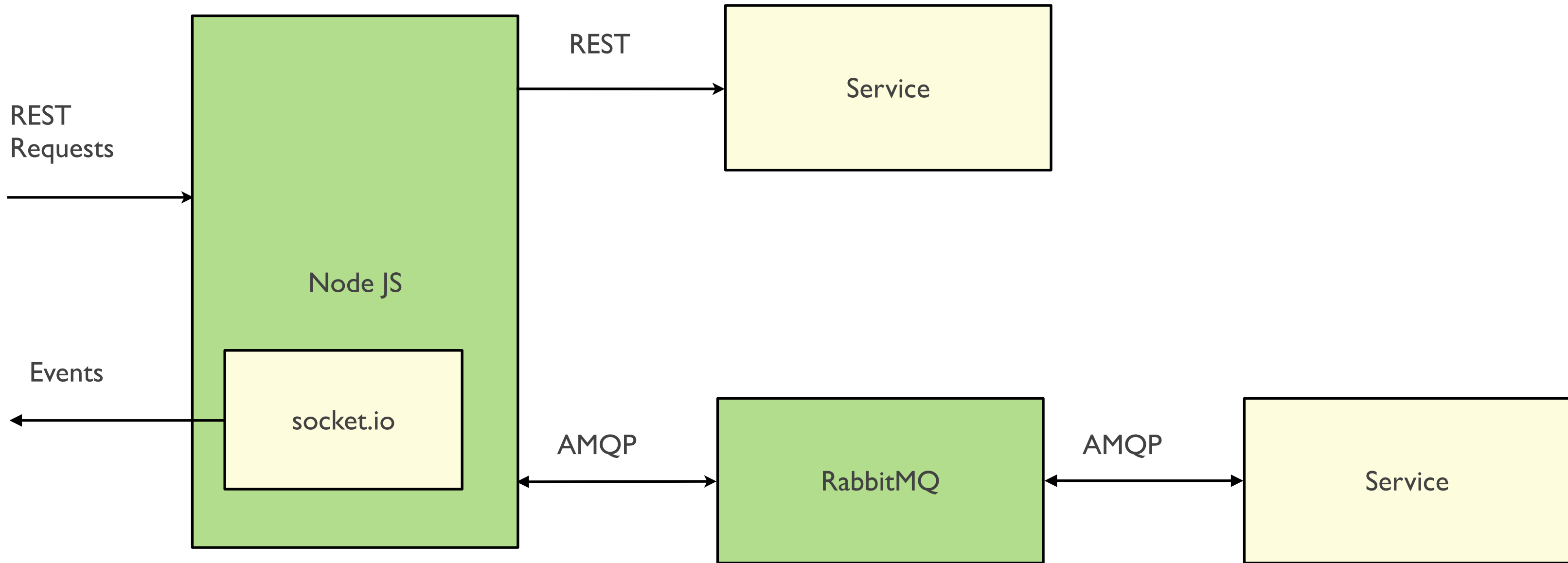


a.k.a. callback hell

A modern web application



NodeJS - using RESTful WS and AMQP



Socket.io server-side

```
var express = require('express')
  , http = require('http')
  , amqp = require('amqp')
  ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
    var m = JSON.parse(message.data.toString());
    socket.emit('tick', m);
  };
  amqpCon.queue("", {},
    function(queue) {
      queue.bind("myExchange", "");
      queue.subscribe(amqpMessageHandler);
    });
});
```


Socket.io server-side

```
var express = require('express')
  , http = require('http')
  , amqp = require('amqp')
  ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
    var m = JSON.parse(message.data.toString());
    socket.emit('tick', m);
  };
  amqpCon.queue("", {},
    function(queue) {
      queue.bind("myExchange", "");
      queue.subscribe(amqpMessageHandler);
    });
});
```

Handle socket.io
connection


Socket.io server-side

```
var express = require('express')
  , http = require('http')
  , amqp = require('amqp')
  ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
    var m = JSON.parse(message.data.toString());
    socket.emit('tick', m);
  };
  amqpCon.queue("", {},
    function(queue) {
      queue.bind("myExchange", "");
      queue.subscribe(amqpMessageHandler);
    });
});
```



Subscribe to AMQP
queue

Socket.io server-side

```
var express = require('express')
  , http = require('http')
  , amqp = require('amqp')
  ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
    var m = JSON.parse(message.data.toString());
    socket.emit('tick', m);
  };
  amqpCon.queue("", {},
    function(queue) {
      queue.bind("myExchange", "");
      queue.subscribe(amqpMessageHandler);
    });
});
```

Republish as
socket.io event

Socket.io - client side

```
<html>
<body>
```

The event is ``

```
<script src="/socket.io/socket.io.js"></script>
<script src="/knockout-2.0.0.js"></script>
<script src="/clock.js"></script>

</body>
</html>
```

clock.js

```
var socket = io.connect(location.hostname);

function ClockModel() {
  self.ticker = ko.observable(1);
  socket.on('tick', function (data) {
    self.ticker(data);
  });
};

ko.applyBindings(new ClockModel());
```

Socket.io - client side

```
<html>  
<body>
```

The event is ``

```
<script src="/socket.io/socket.io.js"></script>  
<script src="/knockout-2.0.0.js"></script>  
<script src="/clock.js"></script>
```

```
</body>  
</html>
```

Bind to model

clock.js

```
var socket = io.connect(location.hostname);
```

```
function ClockModel() {  
  self.ticker = ko.observable(1);  
  socket.on('tick', function (data) {  
    self.ticker(data);  
  });  
};
```

```
ko.applyBindings(new ClockModel());
```

Socket.io - client side

```
<html>  
<body>
```

The event is ``

```
<script src="/socket.io/socket.io.js"></script>  
<script src="/knockout-2.0.0.js"></script>  
<script src="/clock.js"></script>
```

```
</body>  
</html>
```

Connect to
socket.io server

clock.js

```
var socket = io.connect(location.hostname);
```

```
function ClockModel() {  
  self.ticker = ko.observable(1);  
  socket.on('tick', function (data) {  
    self.ticker(data);  
  });  
};
```

```
ko.applyBindings(new ClockModel());
```

Socket.io - client side

```
<html>  
<body>
```

The event is ``

```
<script src="/socket.io/socket.io.js"></script>  
<script src="/knockout-2.0.0.js"></script>  
<script src="/clock.js"></script>  
  
</body>  
</html>
```

clock.js

```
var socket = io.connect(location.hostname);
```

```
function ClockModel() {  
  self.ticker = ko.observable(1);  
  socket.on('tick', function (data) {  
    self.ticker(data);  
  });  
};
```

```
ko.applyBindings(new ClockModel());
```

Subscribe to tick
event

Socket.io - client side

```
<html>  
<body>
```

The event is ``

```
<script src="/socket.io/socket.io.js"></script>  
<script src="/knockout-2.0.0.js"></script>  
<script src="/clock.js"></script>
```

```
</body>  
</html>
```

clock.js

```
var socket = io.connect(location.hostname);
```

```
function ClockModel() {  
  self.ticker = ko.observable(1);  
  socket.on('tick', function (data) {  
    self.ticker(data);  
  });  
};
```

```
ko.applyBindings(new ClockModel());
```

Update model

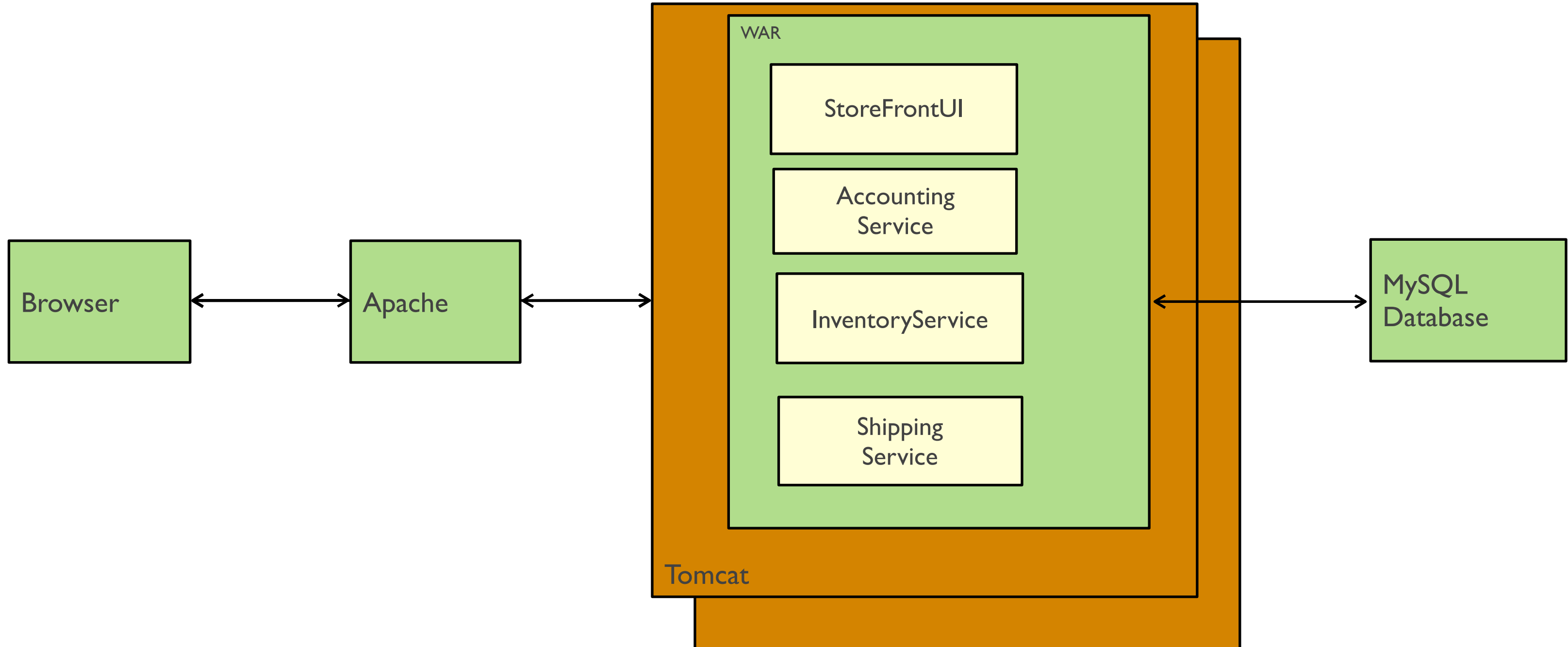
Agenda

- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps

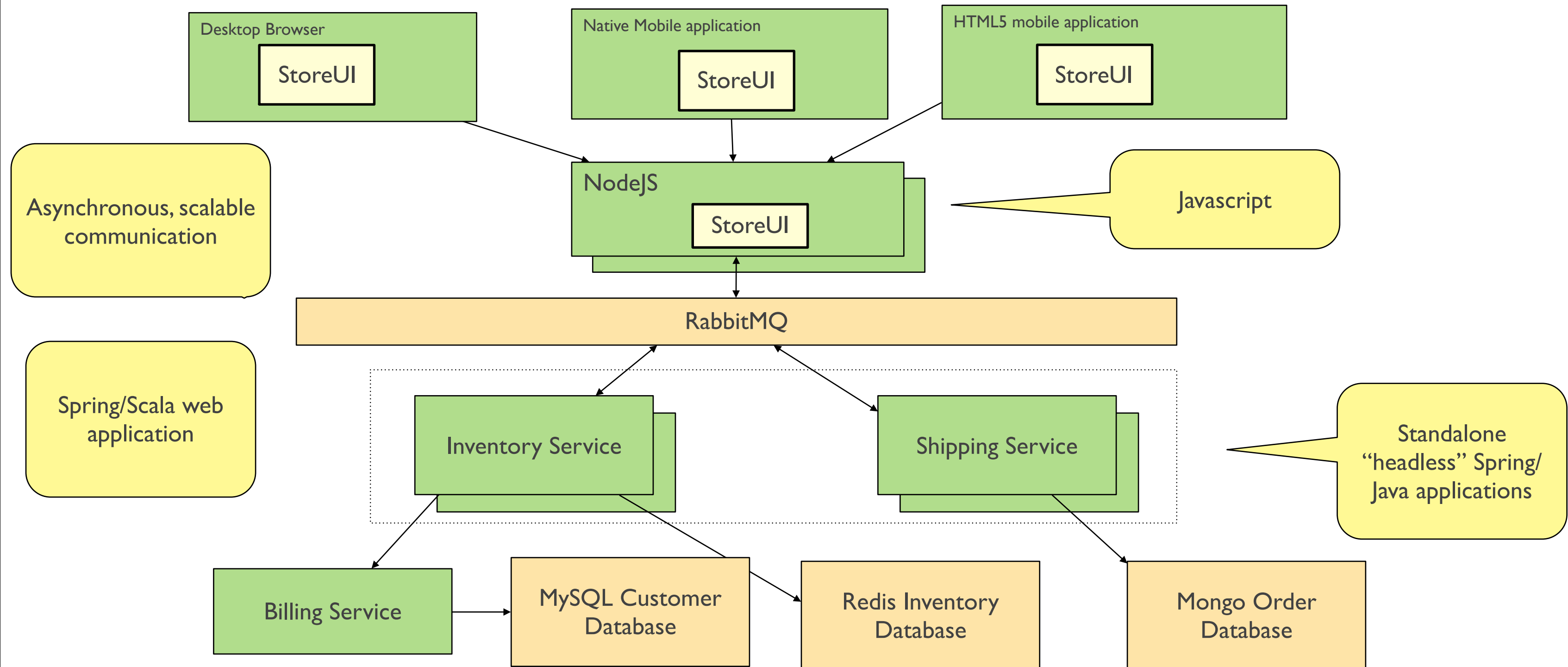
Agenda

- The (sometimes evil) monolith
- Decomposing applications into services
- How do services communicate?
- Presentation layer design
- How Cloud Foundry helps

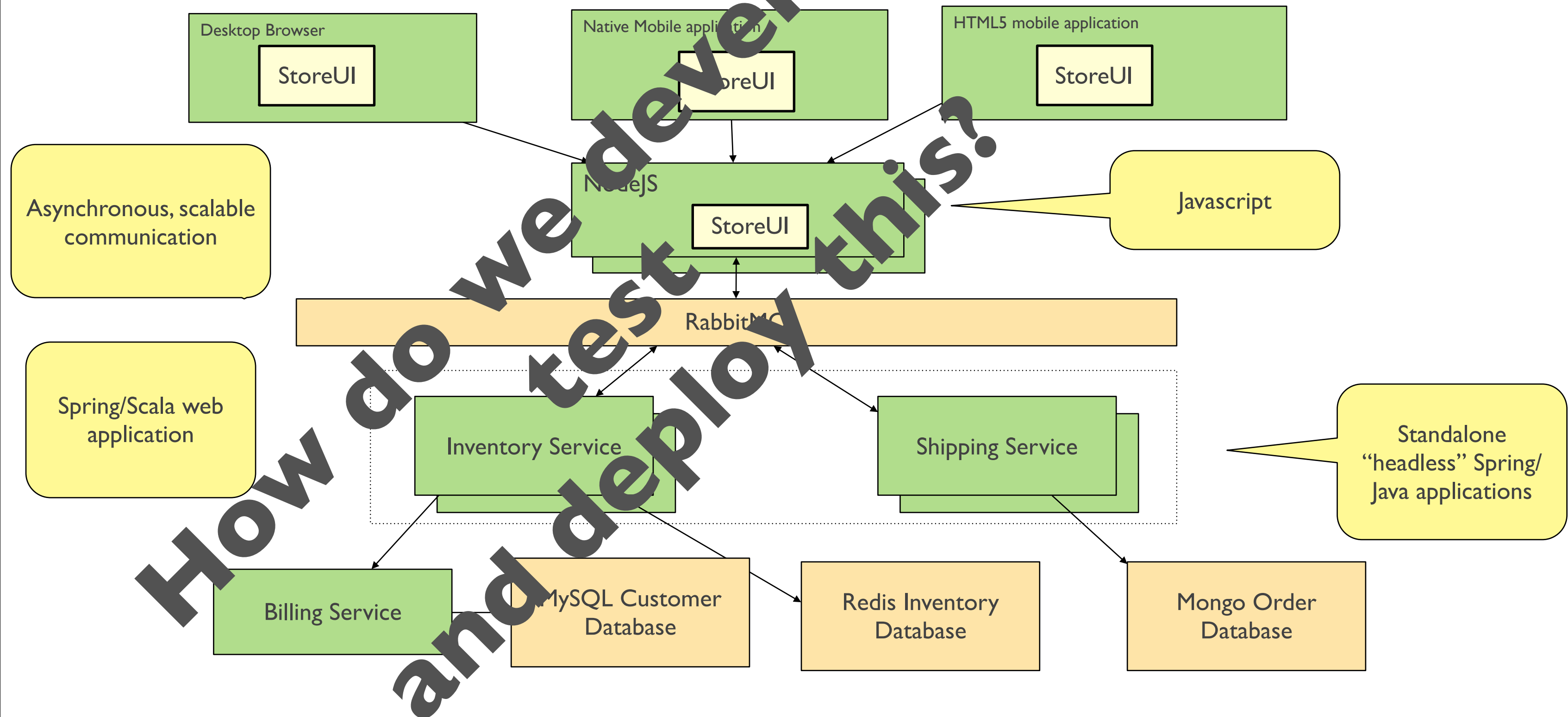
Original architecture



Modern architecture



Modern architecture



Traditional tools: monolithic applications



Web development that doesn't hurt

Ruby on Rails® is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.



Developing modular apps is more difficult

Developing modular apps is more difficult

- Many more moving parts to manage

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code
- Who is going to setup the environments:

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code
- Who is going to setup the environments:
 - the developer sandbox?

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code
- Who is going to setup the environments:
 - the developer sandbox?
 - ...

Developing modular apps is more difficult

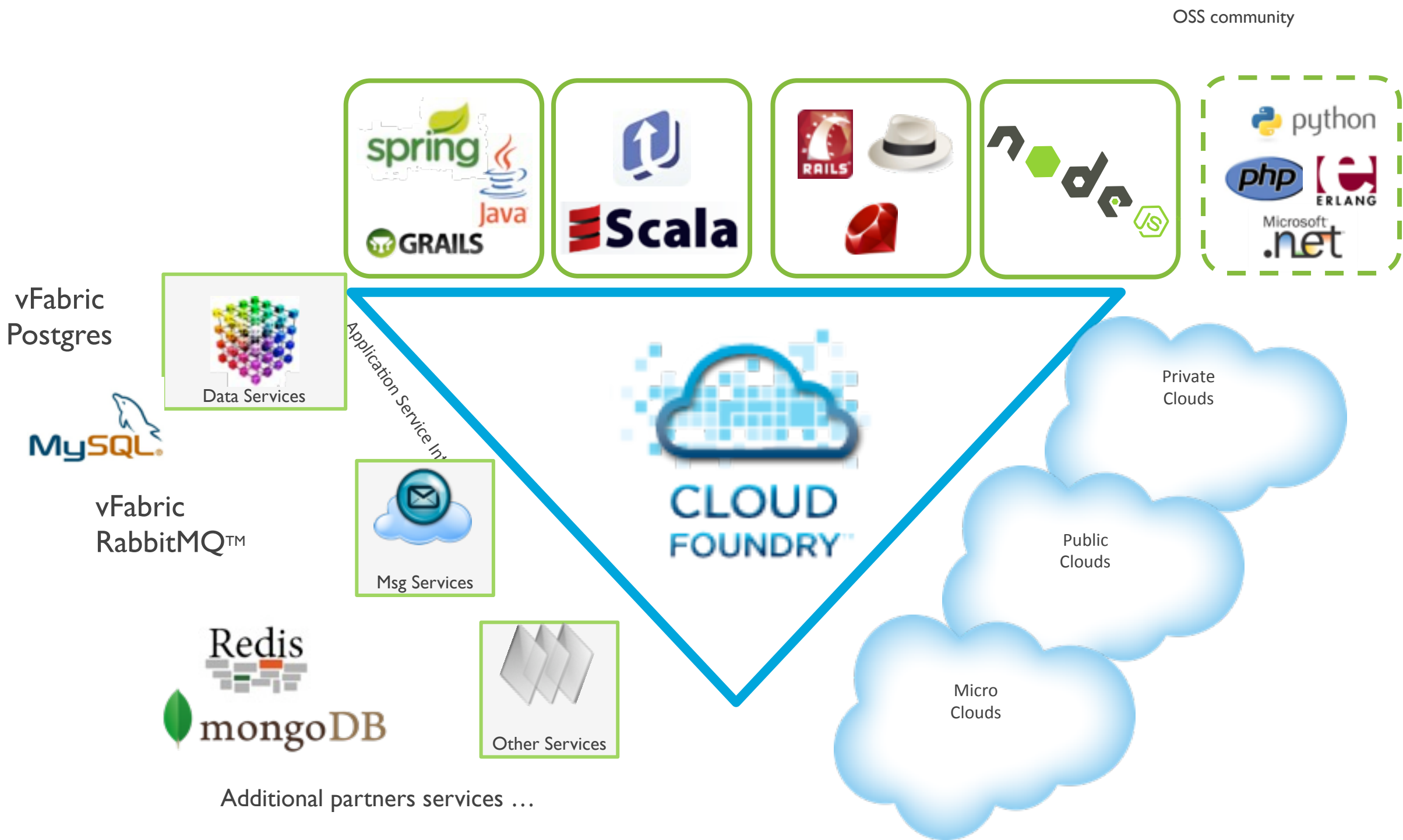
- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code
- Who is going to setup the environments:
 - the developer sandbox?
 - ...
 - QA environments?

Developing modular apps is more difficult

- Many more moving parts to manage
 - **Platform services:** SQL, NoSQL, RabbitMQ
 - **Application services:** your code
- Who is going to setup the environments:
 - the developer sandbox?
 - ...
 - QA environments?

But Cloud Foundry helps...

Easy polyglot application deployment and service provisioning



Creating a platform service instance

```
$ vmc create-service mysql --name mysql1
```

```
Creating Service: OK
```

```
$ vmc services
```

```
.....
```

```
===== Provisioned Services =====
```

+-----+-----+
Name Service
+-----+-----+
mysql1 mysql
+-----+-----+

Multi-application manifest - part 1

applications:

inventory/target:

name: inventory

url: cer-inventory.chrisr.cloudfoundry.me

framework:

name: spring

info:

mem: 512M

description: Java SpringSource Spring Application

exec:

mem: 512M

instances: 1

services:

si-rabbit:

type: :rabbitmq

si-mongo:

type: :mongodb

si-redis:

type: :redis

Path to application

Required platform services

Multi-application manifest - part 2

store/target:

name: store

url: cer-store.chrisr.cloudfoundry.me

framework:

name: spring

info:

mem: 512M

description: Java SpringSource Spring Application

exec:

mem: 512M

instances: 1

services:

si-mongo:

type: :mongodb

si-rabbit:

type: :rabbitmq

Path to application

Required platform services

One command to create platform services and deploy application

\$ vmc push

Would you like to deploy from the current directory? [Yn]:

Pushing application 'inventory'...

Creating Application: OK

Creating Service [si-rabbit]: OK

Binding Service [si-rabbit]: OK

Creating Service [si-mongo]: OK

Binding Service [si-mongo]: OK

Creating Service [si-redis]: OK

Binding Service [si-redis]: OK

Uploading Application:

Checking for available resources: OK

Processing resources: OK

Packing application: OK

Uploading (12K): OK

Push Status: OK

Staging Application 'inventory': OK

Starting Application 'inventory': OK

Pushing application 'store'...

Creating Application: OK

Binding Service [si-mongo]: OK

Binding Service [si-rabbit]: OK

Uploading Application:

Checking for available resources: OK

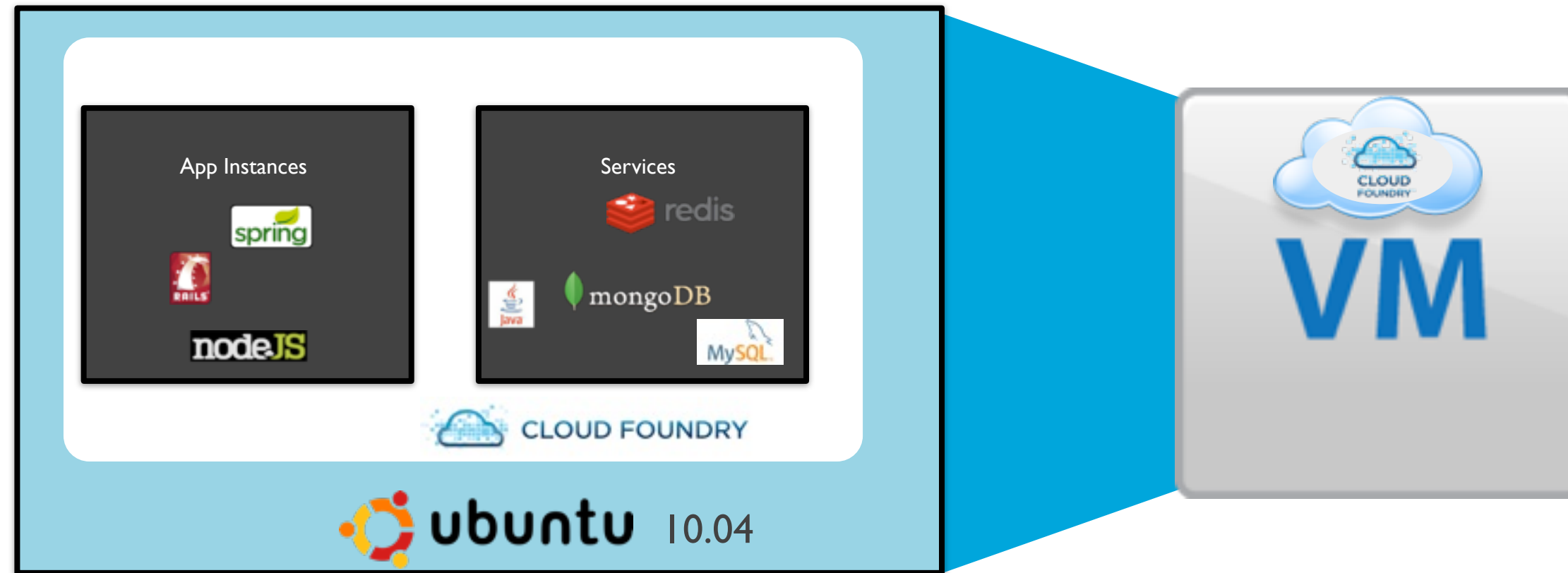
Processing resources: OK

Packing application: OK

vmc push:

- Reads the manifest file
- Creates the required platform services
- Deploys all the applications

Micro Cloud Foundry: new developer sandbox



A PaaS packaged as a VMware Virtual Machine

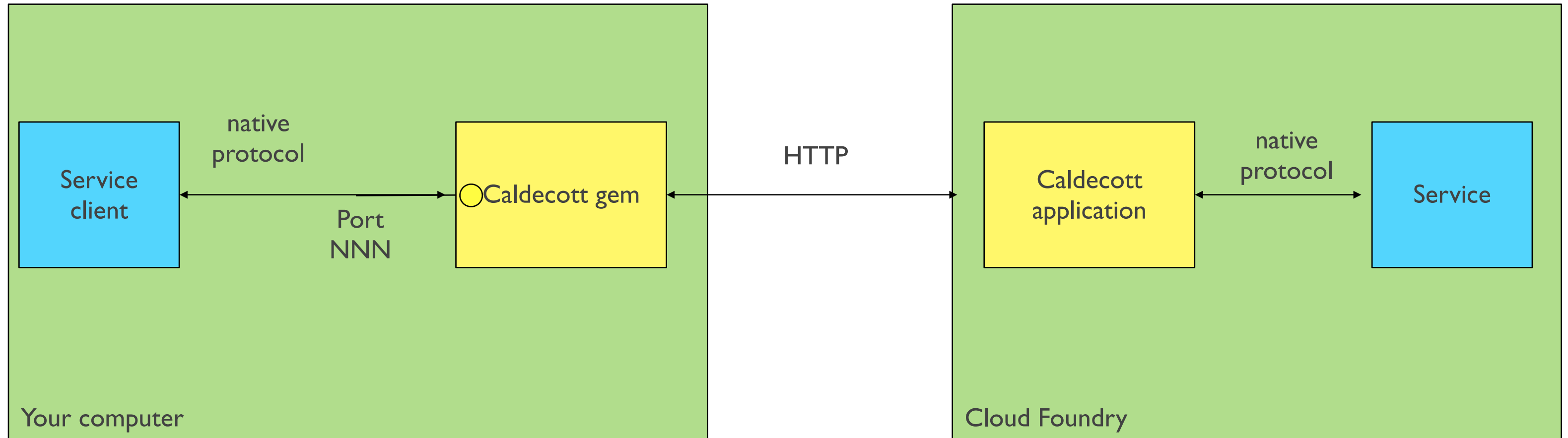
Use as a developer sandbox

- Use the services from Junit integration tests
- Deploy your application for functional testing
- Remote debugging from STS

Using Caldecott to tunnel into your services



Caldecott = TCP over HTTP



Using Caldecott...

```
$ vmc tunnel
1: mysql-135e0
2: mysql1
Which service to tunnel to?: 2
Password: *****
Stopping Application: OK
Redeploying tunnel application 'caldecott'.
Uploading Application:
  Checking for available resources: OK
  Packing application: OK
  Uploading (1K): OK
Push Status: OK
Binding Service [mysql1]: OK
Staging Application: OK
Starting Application: OK
Getting tunnel connection info: OK

Service connection info:
  username : uMe6Apgw00AhS
  password : pKcD76PcZR7GZ
  name      : d7cb8afb52f084f3d9bdc269e7d99ab50

Starting tunnel to mysql1 on port 10000.
1: none
2: mysql
Which client would you like to start?: 2
```


...Using Caldecott

```
Launching 'mysql --protocol=TCP --host=localhost --port=10000 --user=uMe6Apgw00AhS --  
password=pKcD76PcZR7GZ d7cb8afb52f084f3d9bdc269e7d99ab50'
```

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 10944342

Server version: 5.1.54-rel12.5 Percona Server with XtraDB (GPL), Release 12.5, Revision 188

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

Running JUnit test with Caldecott

Configure your test code to use port + connection info



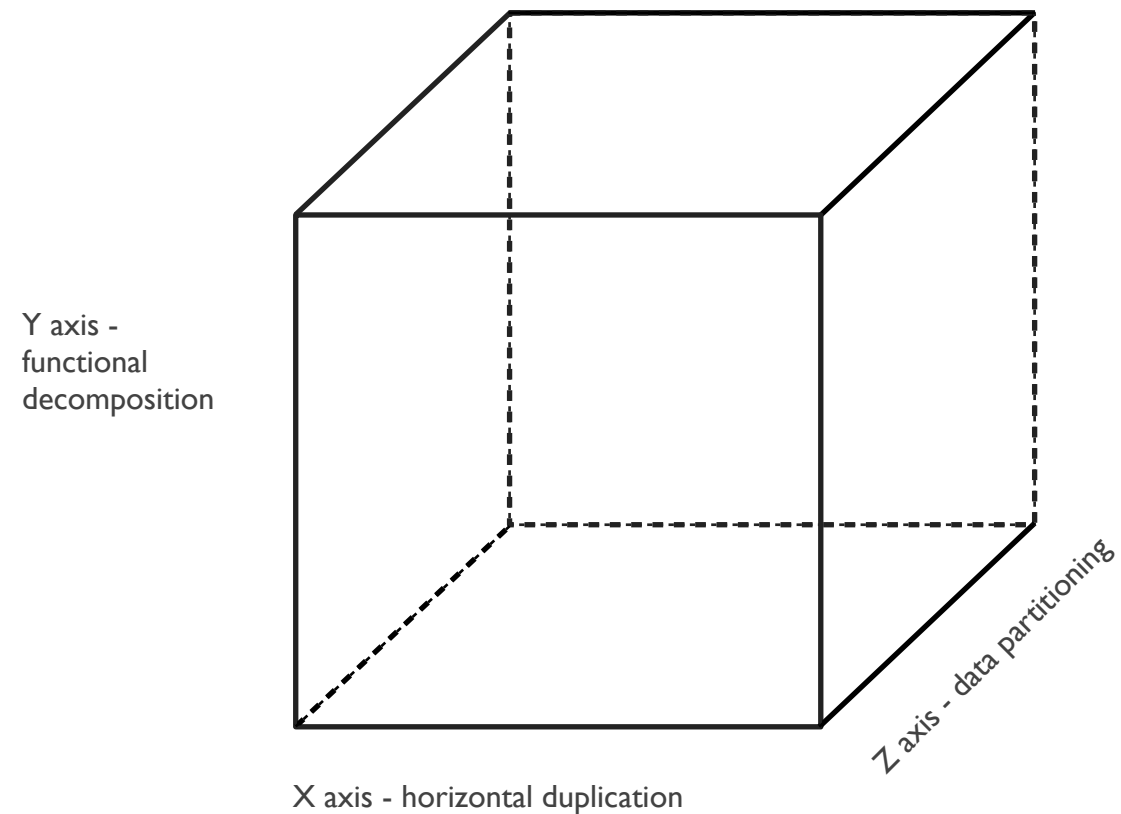
```
Service connection info:  
  username : uFZpMHcVgMyjN  
  password : pMYfxETX3dcxA  
  name      : da285916ae4234b91a6ceadb638aa8365  
  
Starting tunnel to survey-mysql on port 10000.
```

Summary

**Monolithic applications are
simple to develop and deploy**

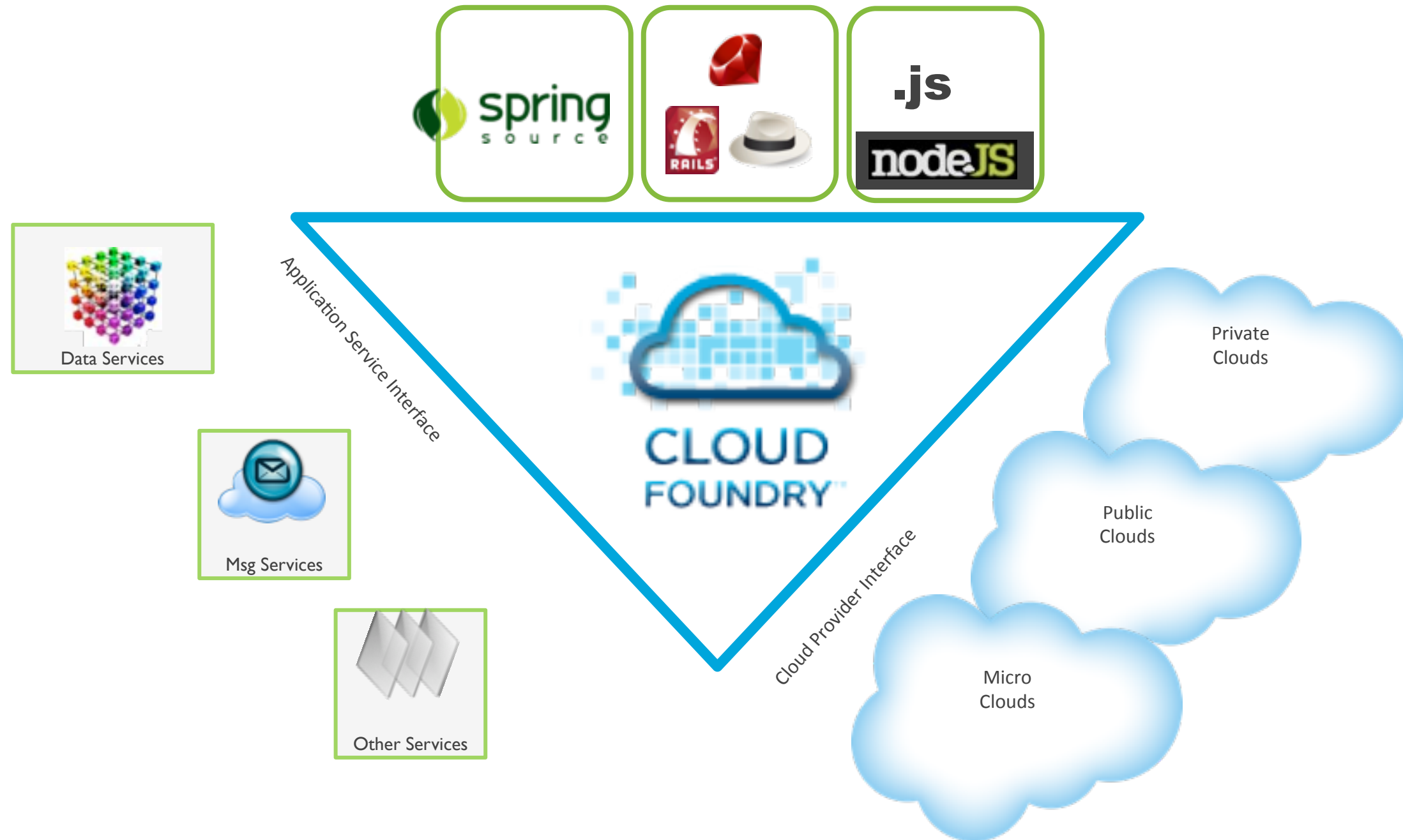
BUT have significant drawbacks

Apply the scale cube



- Modular, polyglot, and scalable applications
- Services developed, deployed and scaled independently

Cloud Foundry helps



 **@crichardson crichardson@vmware.com**
<http://plainoldobjects.com/presentations/>



Questions?

www.cloudfoundry.com @cloudfoundry

Cloud Foundry 启动营

在www.cloudfoundry.com注册账号并成功上传应用程序,
即可于12月8日中午后凭账号ID和应用URL到签到处换取Cloud Foundry主题卫衣一件。



iPhone5 等你拿

第二天大会结束前，请不要提前离开，将填写完整的意见反馈表投到签到处抽奖箱内，即可参与“iPhone5”抽奖活动。



Birds of a Feather 专家面对面

所有讲师都会在课程结束后，到紫兰厅与来宾讨论课程上的问题

