



RED HAT® JBOSS®  
MIDDLEWARE

# Refactoring your Java EE applications using Microservices and Containers

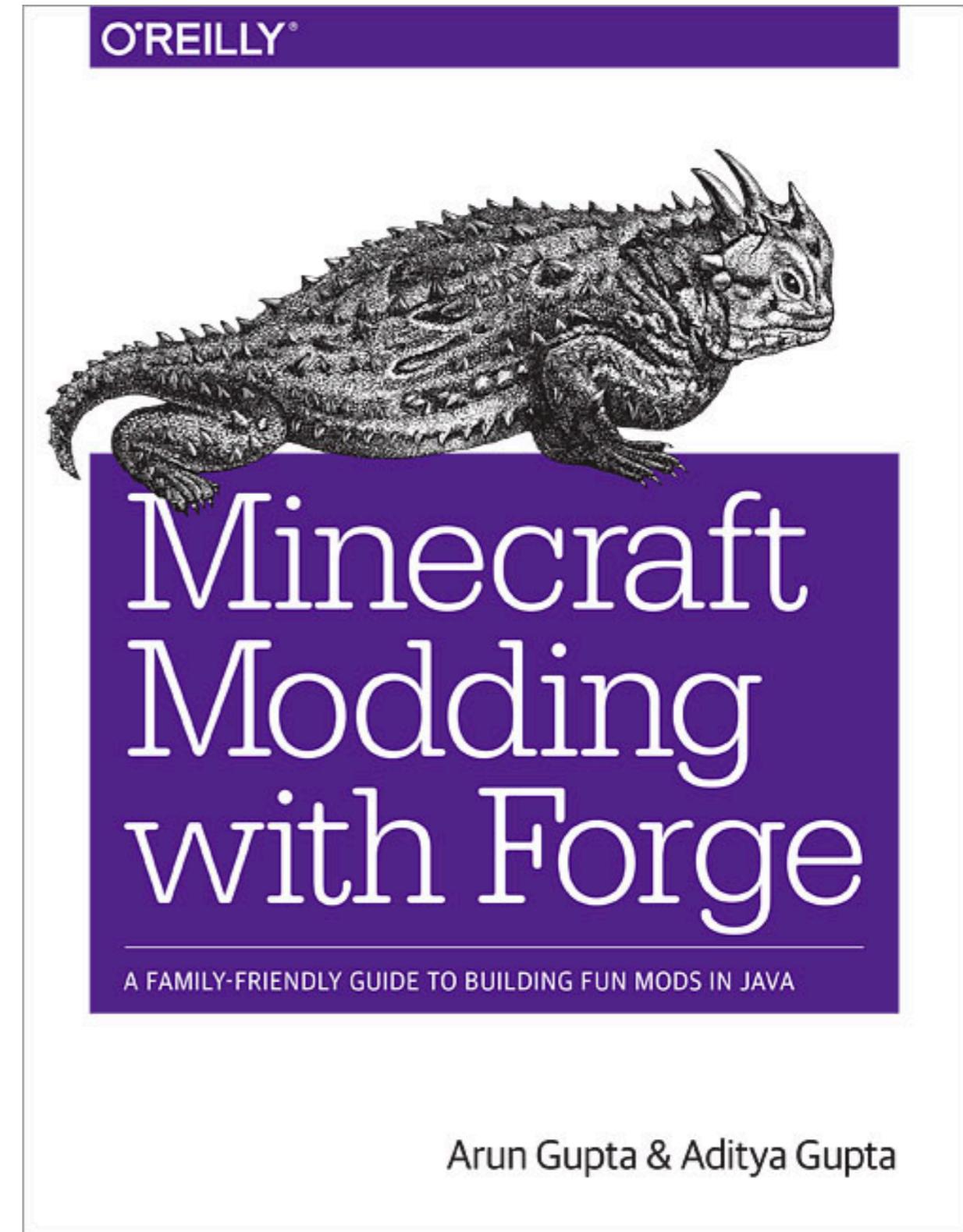
Arun Gupta, Red Hat



# Arun Gupta

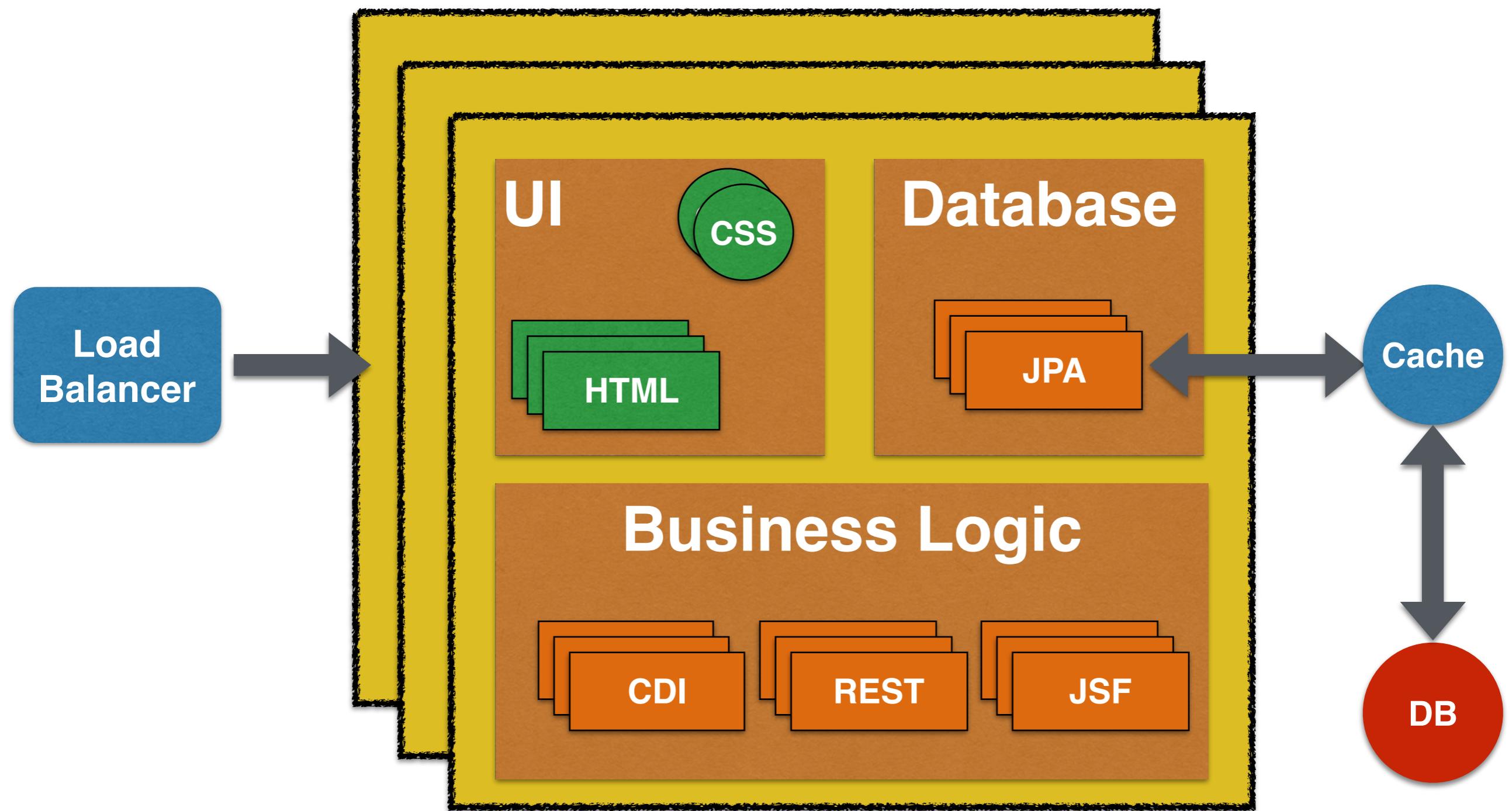
Director, Technical Marketing  
& Developer Advocacy

@arungupta  
[blog.arungupta.me](http://blog.arungupta.me)  
[arungupta@redhat.com](mailto:arungupta@redhat.com)

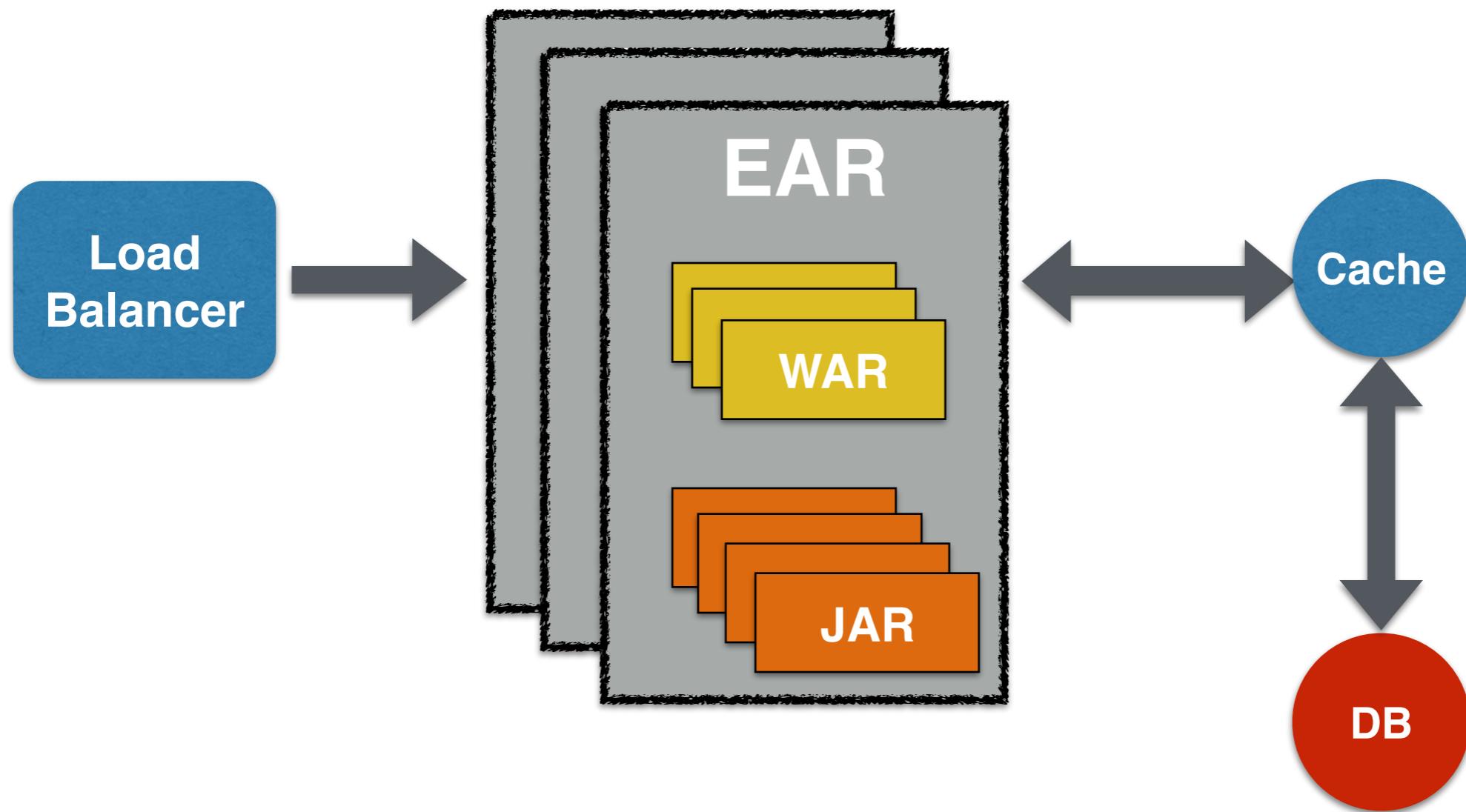


Arun Gupta & Aditya Gupta

# Monolith Application



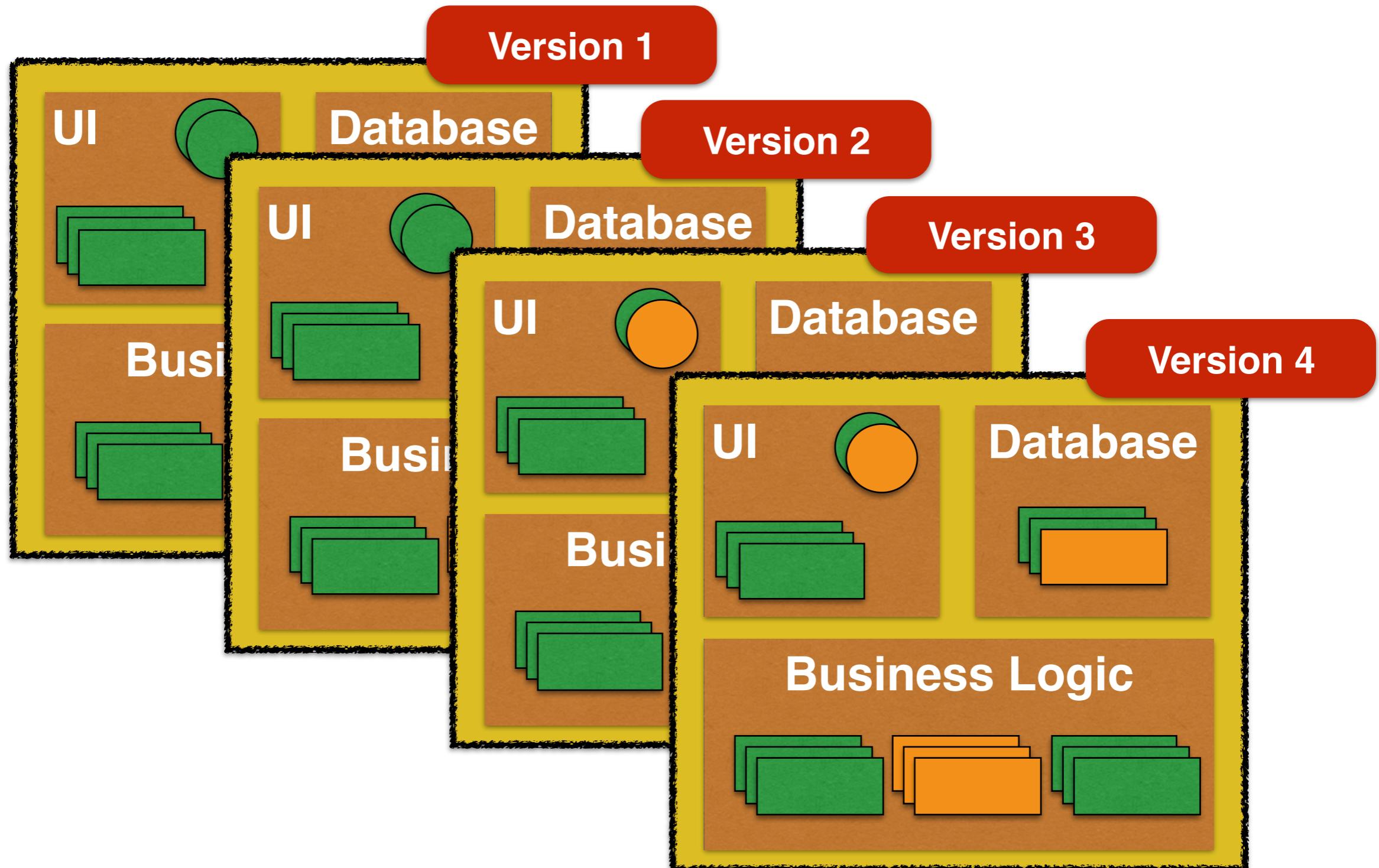
# Monolith Application



# Advantages of Monolith Application

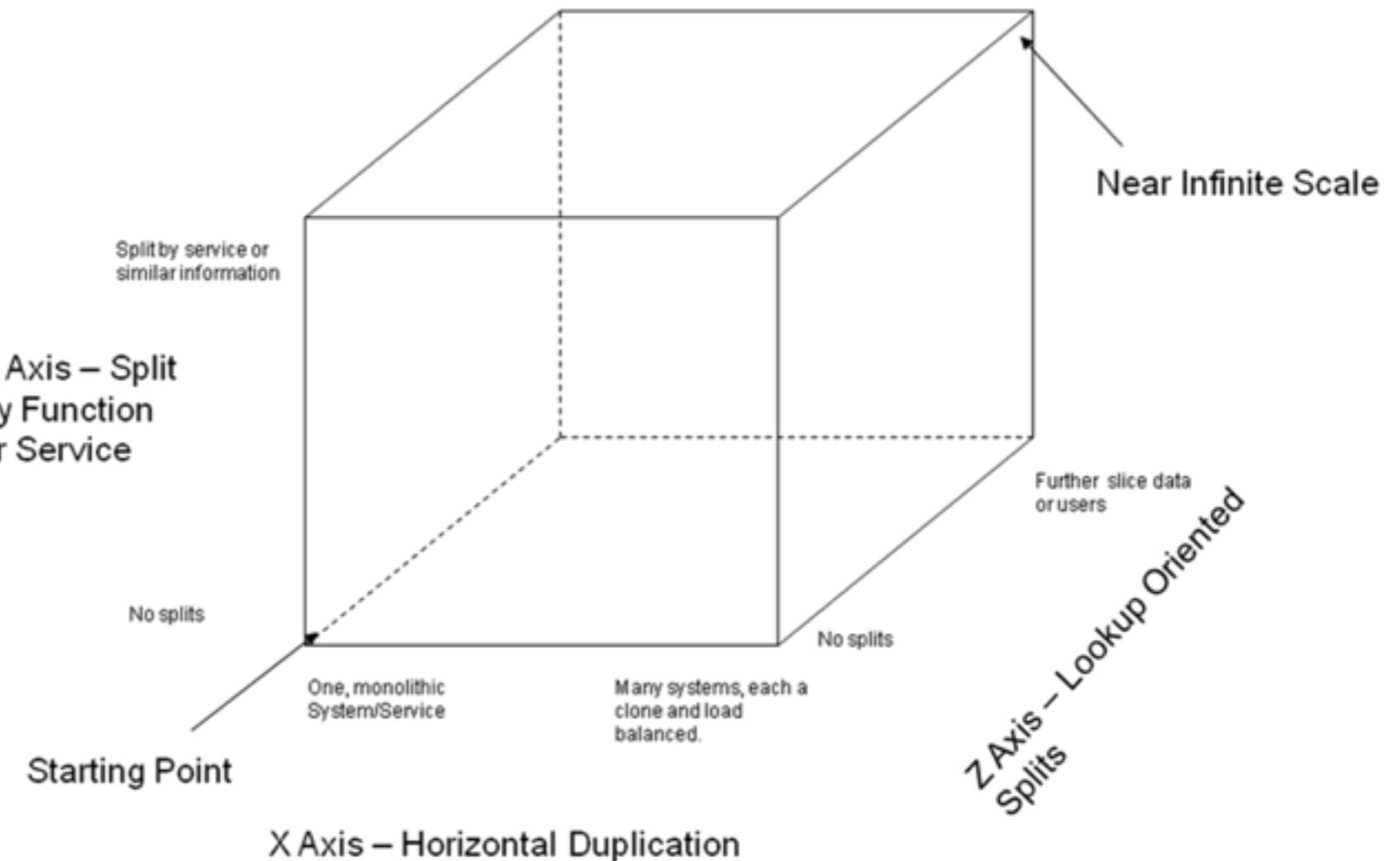
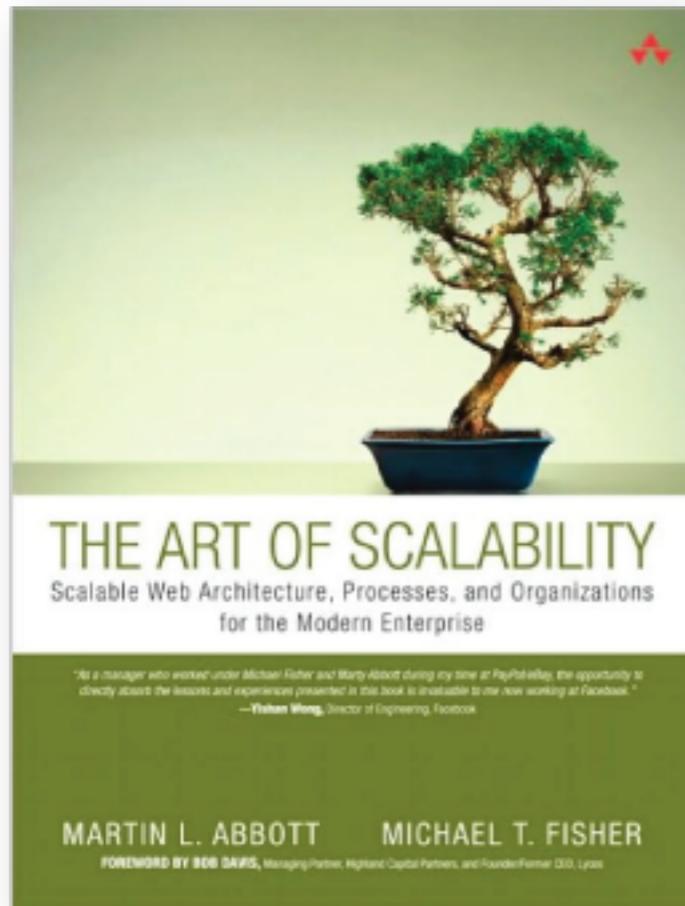
- Typically packaged in a single .ear
- Easy to test (all required services are up)
- Simple to develop

# Monolith Application



# Disadvantages of Monolith Application

- Difficult to deploy and maintain
- Obstacle to frequent deployments
- Makes it difficult to try out new technologies/ framework





An **architectural approach**, that emphasizes the **decomposition of applications** into **single-purpose, loosely coupled** services managed by **cross-functional teams**, for delivering and maintaining **complex software systems** with the velocity and quality required by today's **digital business**.

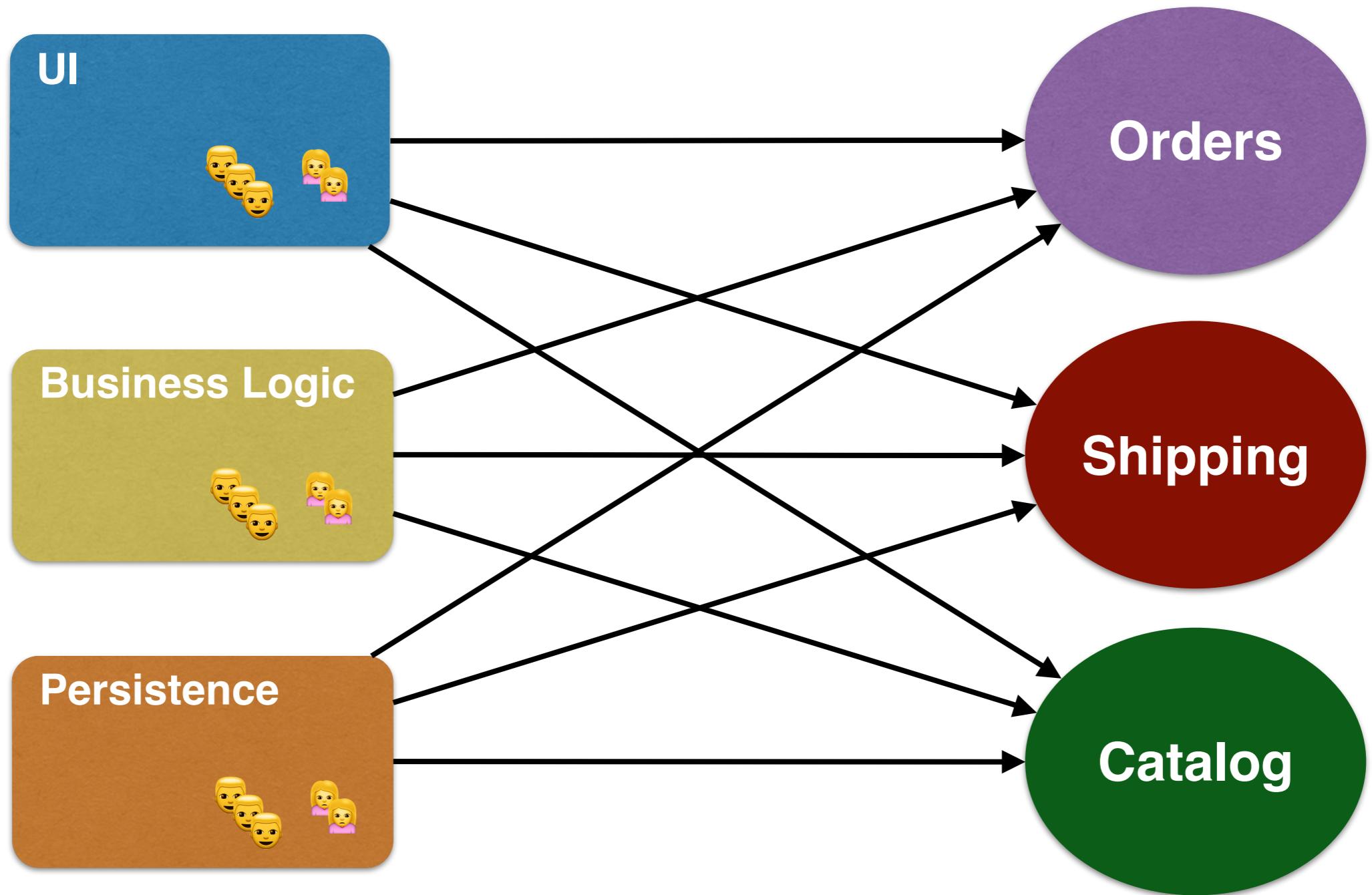


I DONT ALWAYS  
BUILD EVERYTHING



BUT WHEN I DO I  
USE MICRO SERVICES

imgflip.com

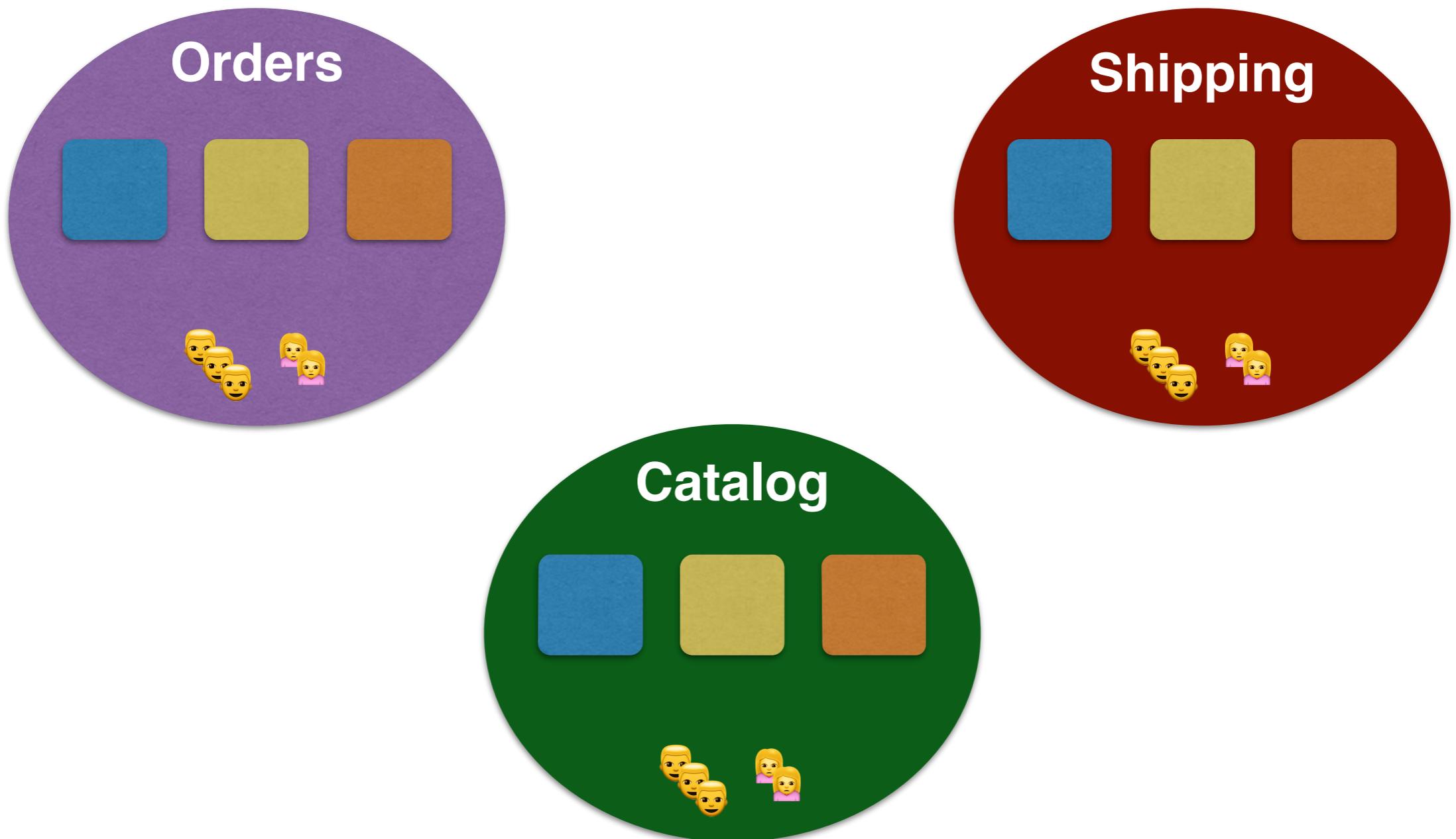




“Any ***organization*** that designs a system  
(defined more broadly here than just information  
systems) will inevitably produce a design  
whose structure is a ***copy of the***  
***organization's communication structure.***”

–Melvin Conway

# Teams around business capability



# Single Responsibility Principle

DO  
1  
THING

# Explicitly Published interface



# Independently replaceable and upgradeable





With great  
power, comes great  
responsibility



“you build it, you run it!”

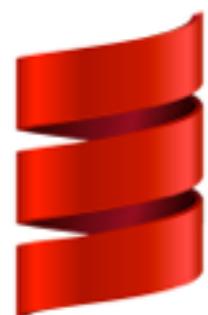
# Designed for failure



*Fault tolerance is a requirement, not a feature*



# Characteristics



Scala



ORACLE  
D A T A B A S E



PostgreSQL



Couchbase



redis



cassandra



# 100% automated

WELLS FARGO  

Sign Off | Home | Locations | Contact Us | Open Account

Accounts Bill Pay Transfers Brokerage Account Services Messages & Alerts

Bill Pay Overview Payments Payees eBills Reports Notices User Profile

**Bill Pay Overview** [Help](#) [Unviewed Notices \(2\)](#) [Unpaid eBills \(3\)](#) [Pending Payments \(6\)](#)

**Make Payment**

Note: Delivery time for payment varies by payee. See number of business days in Send On column.

Payee <a href="#">Add a Payee</a>	Pending Payment	Last Paid	Amount	Send On
<a href="#">AMERICAN EXPRESS</a>	\$2,053.50 06/30/2004	\$1,349.93 05/24/2004	\$ <input type="text"/>	mm/dd/yyyy 3 Business Days
<a href="#">BANK OF AMERICA </a> Receiving eBills <a href="#">View eBill</a>	\$198.80 06/30/2004	\$92.17 05/25/2004	\$ <input type="text"/>	mm/dd/yyyy 3 Business Days
<a href="#">BANK ONE / FIRST</a>	\$55.00 06/25/2004	\$55.00 05/22/2004†	\$ <input type="text"/>	mm/dd/yyyy 3 Business Days
<a href="#">CHARLES SCHWAB</a>			\$ <input type="text"/>	mm/dd/yyyy 5 Business Days
<a href="#">CITIBANK VISA </a> Pending activation	\$63.50 06/30/2004*	\$198.80 05/25/2004*	\$ <input type="text"/>	mm/dd/yyyy 5 Business Days
<a href="#">DIRECT TV </a> Activate eBills		\$63.50 05/25/2004	\$ <input type="text"/>	mm/dd/yyyy 3 Business Days
<a href="#">SBC-PACIFIC BELL</a>		\$45.80 05/25/2004	\$ <input type="text"/>	mm/dd/yyyy 5 Business Days
<a href="#">SPRINT PCS </a> Activate eBills	\$49.78 06/30/2004	\$63.50 06/26/2004	\$ <input type="text"/>	mm/dd/yyyy 5 Business Days
<a href="#">SFPUC-WATER DE</a>			\$ <input type="text"/>	mm/dd/yyyy 3 Business Days
<a href="#">WF HOME MORTGAGE</a>		\$1,349.93 05/25/2004	\$ <input type="text"/>	mm/dd/yyyy 5 Business Days

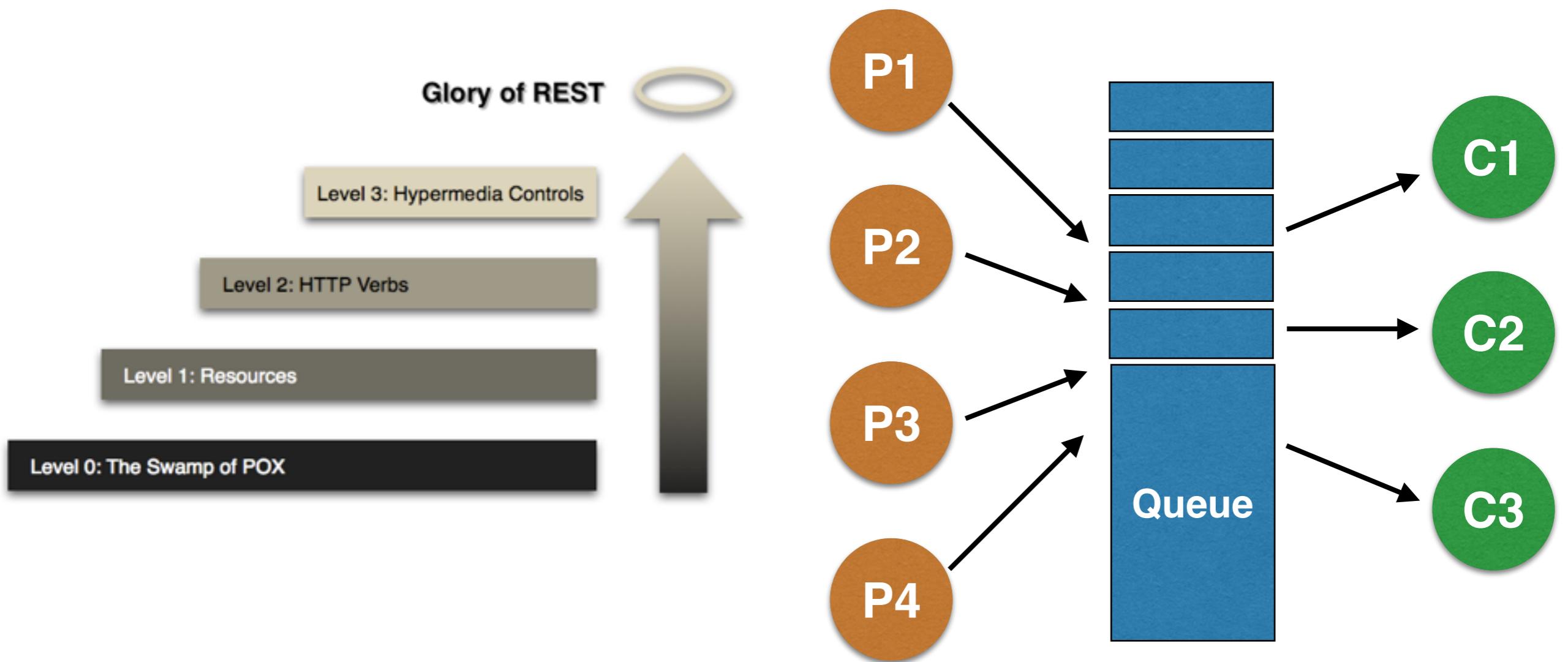
**Make Payment**

[Home](#) | [Locations](#) | [Contact Us](#) | [Open Account](#) | [Sign Off](#)  
© 2001-2005 Wells Fargo. All rights reserved.

# Sync or Async Messaging



# REST vs Pub/Sub



# “Smart endpoints Dumb pipes”



# SOA

- SOA 2.0
- Hipster SOA
- SOA done right
- SOA++

# SOA 2.0?



Arun Gupta  
@arungupta

- Conway's Law
- Service Discovery
- Immutable VM

Microservices = SOA -ESB -SOAP -  
Centralized governance/persistence -  
Vendors +REST/HTTP +CI/CD +DevOps  
+True Polyglot +Containers +PaaS WDYT?



RETWEETS FAVORITES  
**72** **63**



5:07 PM - 27 May 2015

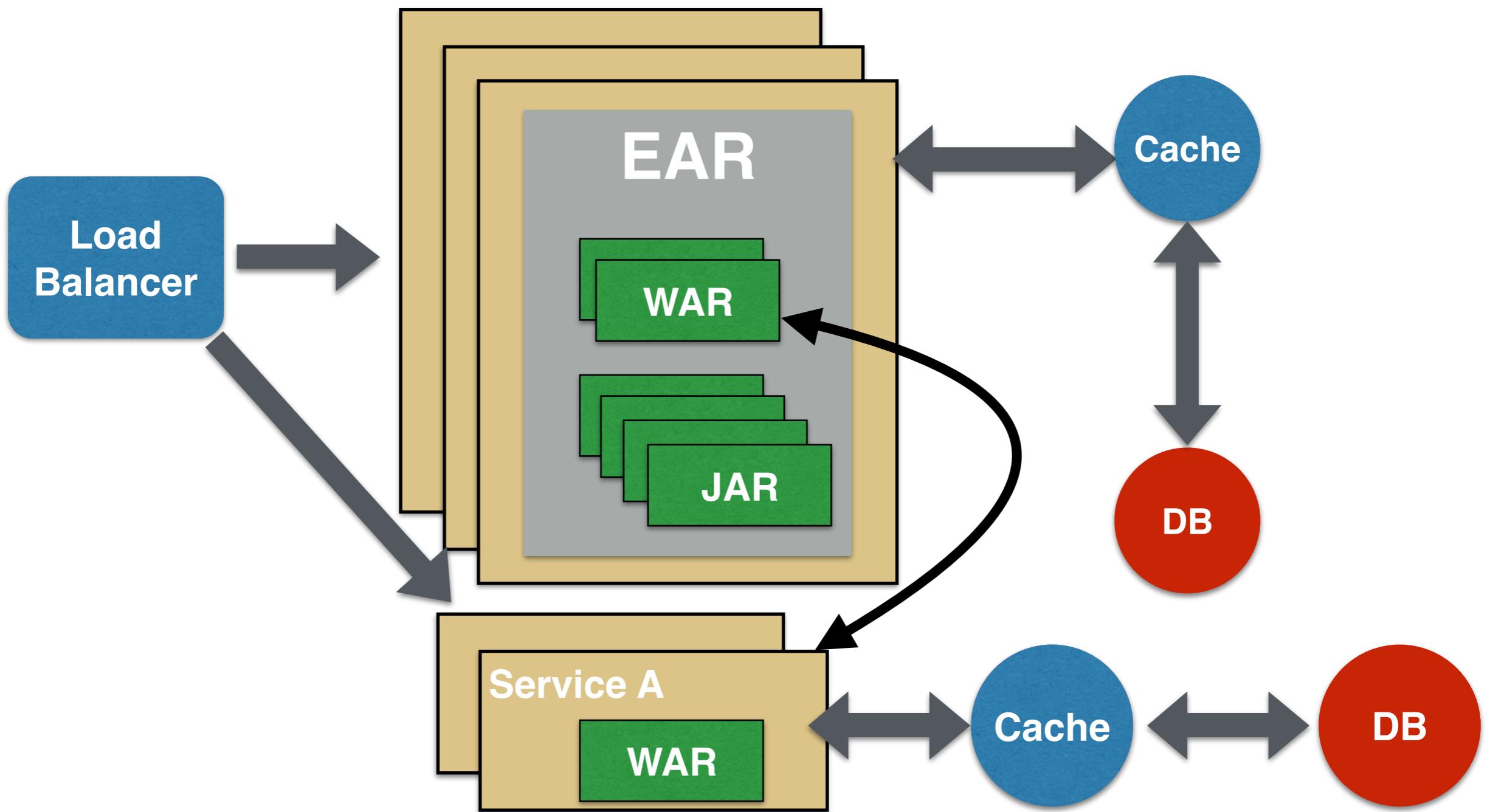
# Strategies for decomposing



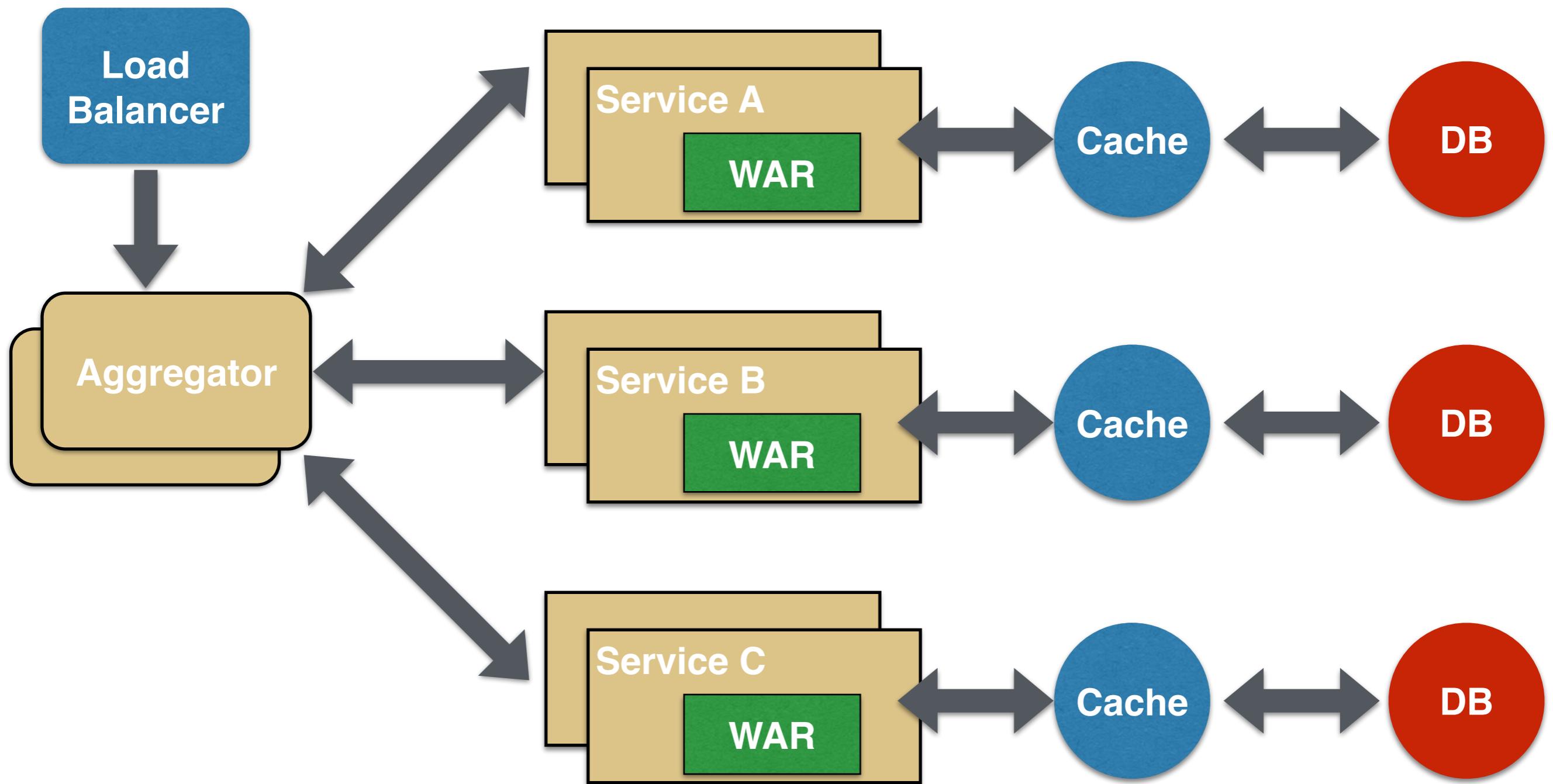
# Strategies for decomposing

- Verb or usecase - e.g. Checkout UI
- Noun - e.g. Catalog product service
- Single Responsible Principle - e.g. Unix utilities

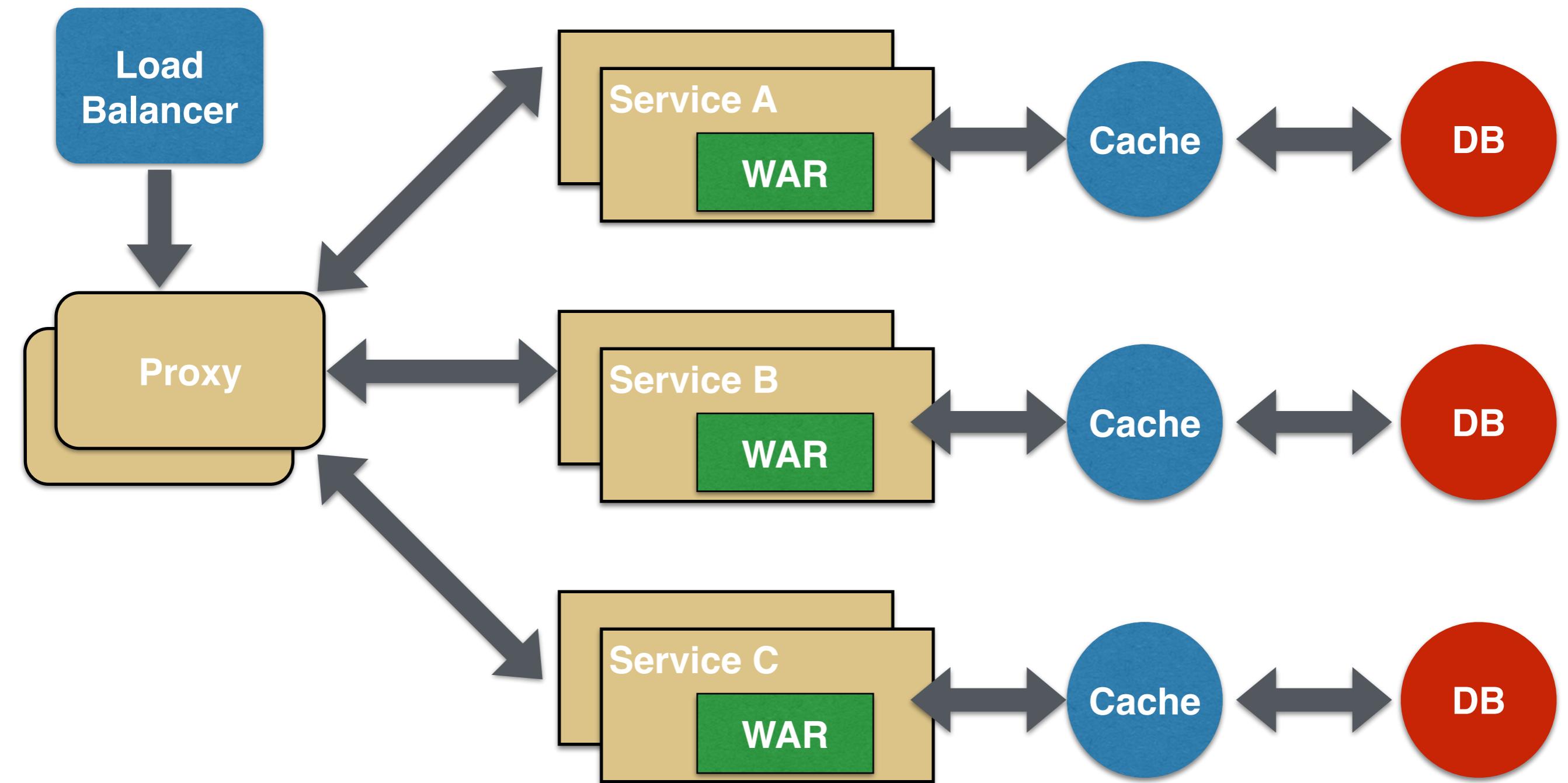
# Towards microservices



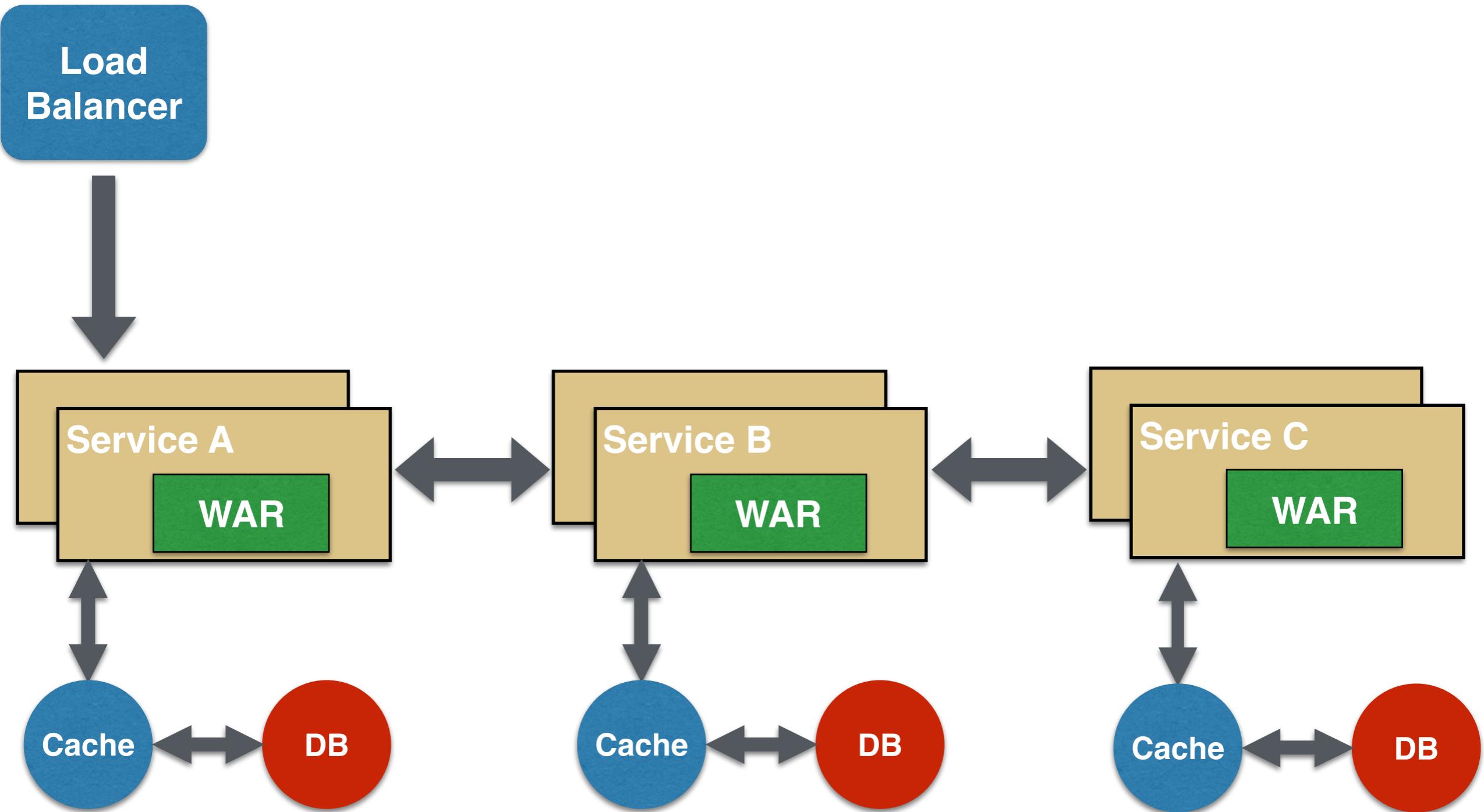
# Aggregator Pattern #1



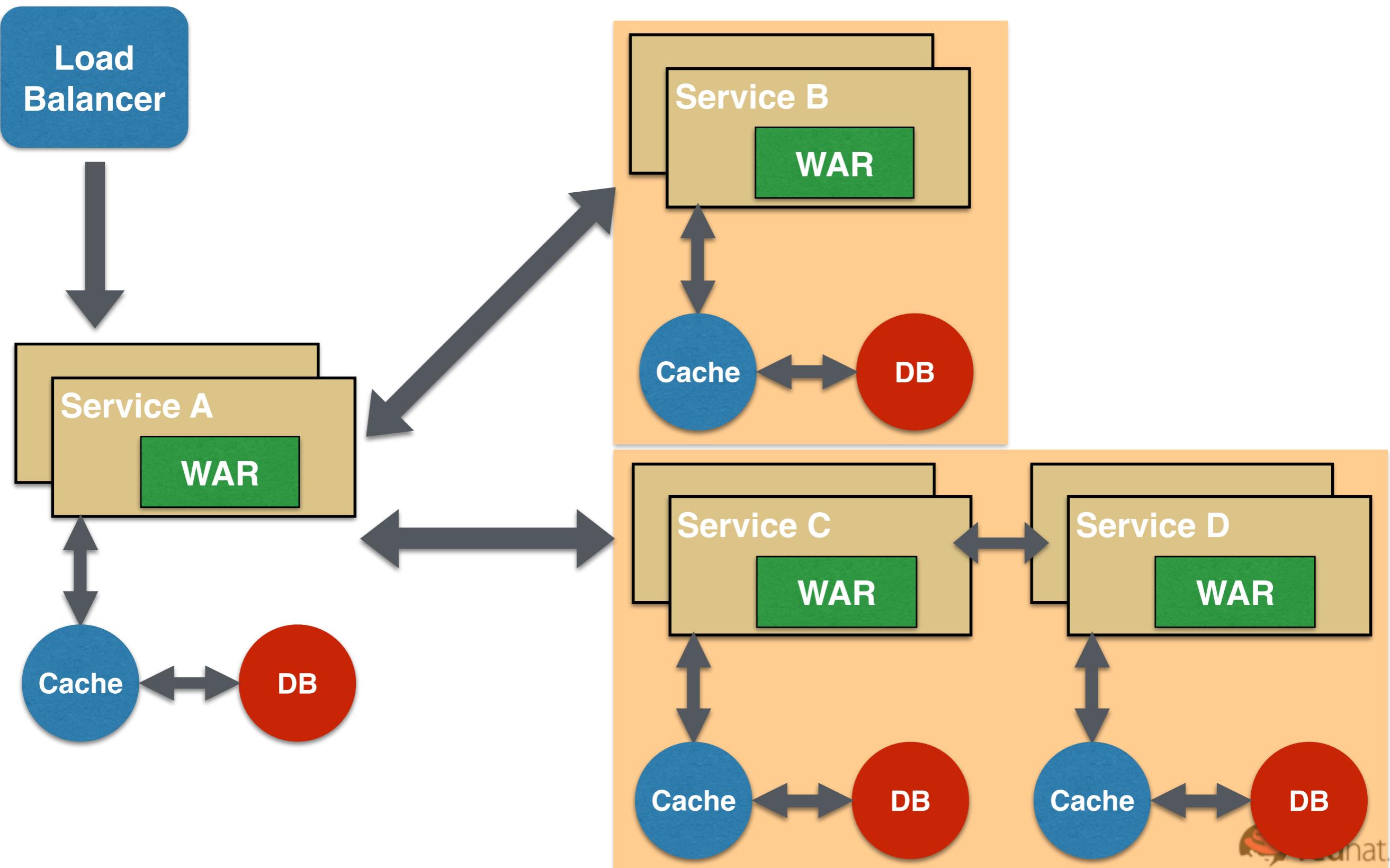
# Proxy Pattern #2



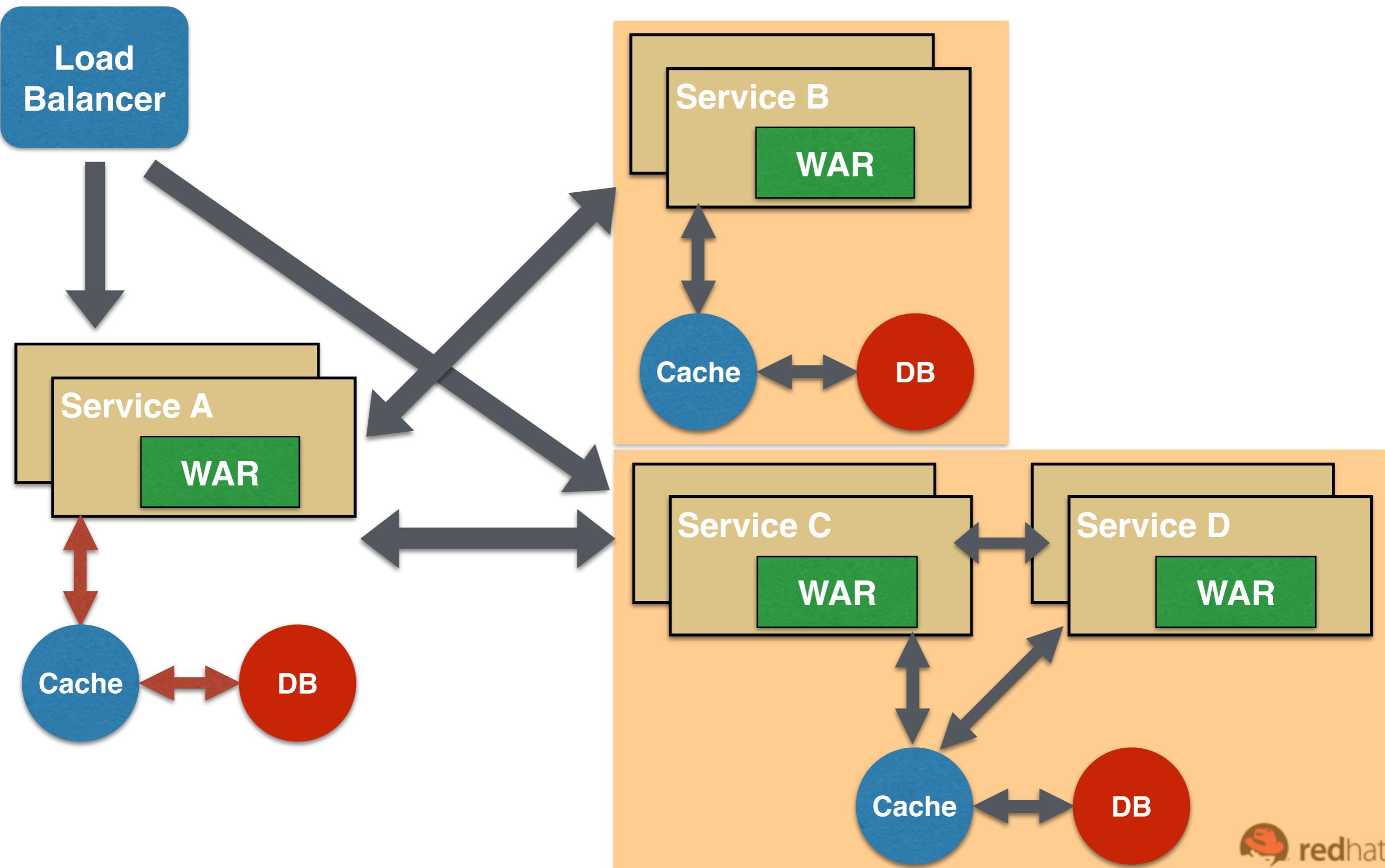
# Chained Pattern #3



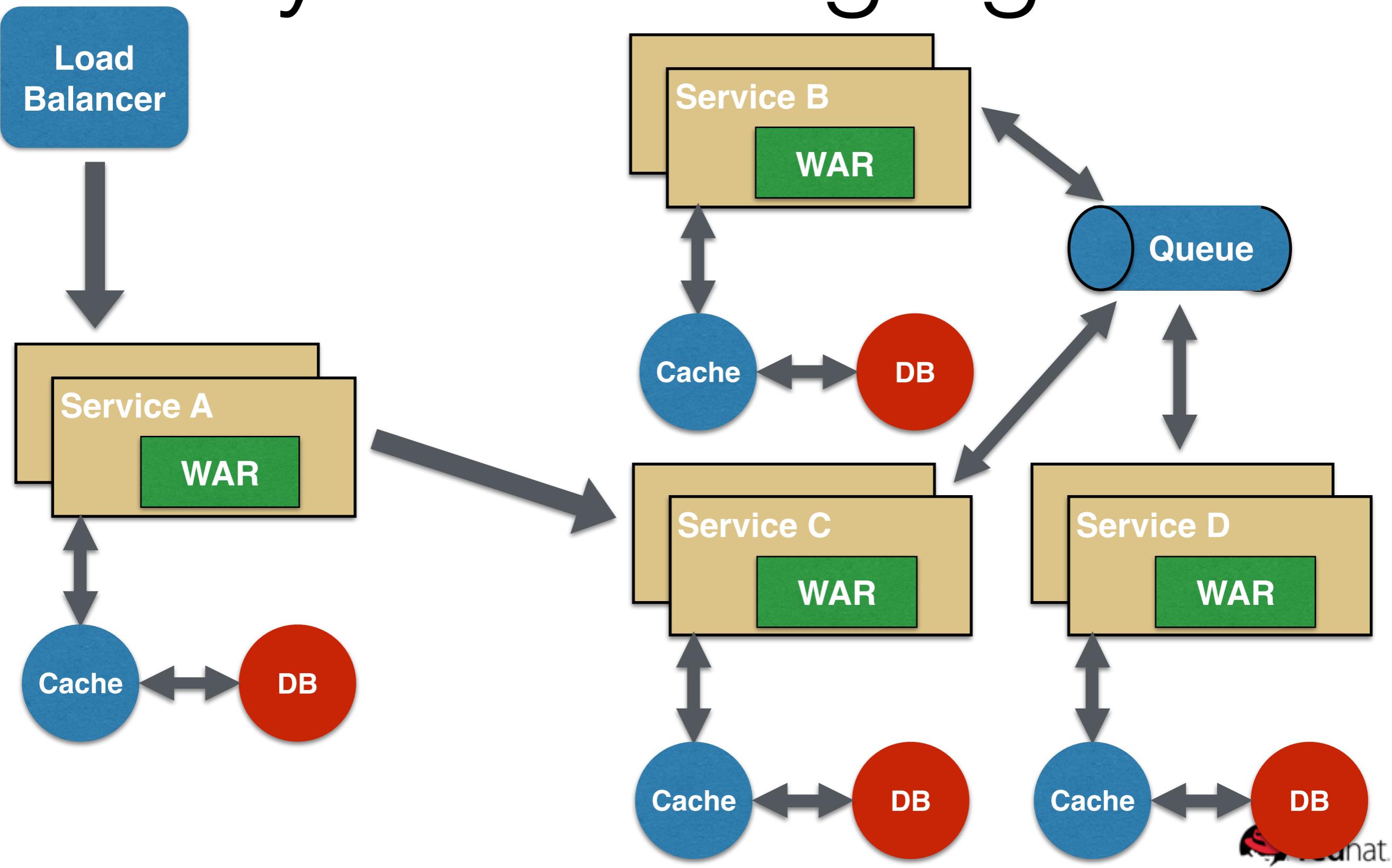
# Branch Pattern #4



# Shared Resources #5



# Async Messaging #5



# SAY MICROSERVICE



# ONE MORE TIME

memegenerator.net

# Advantages of microservices

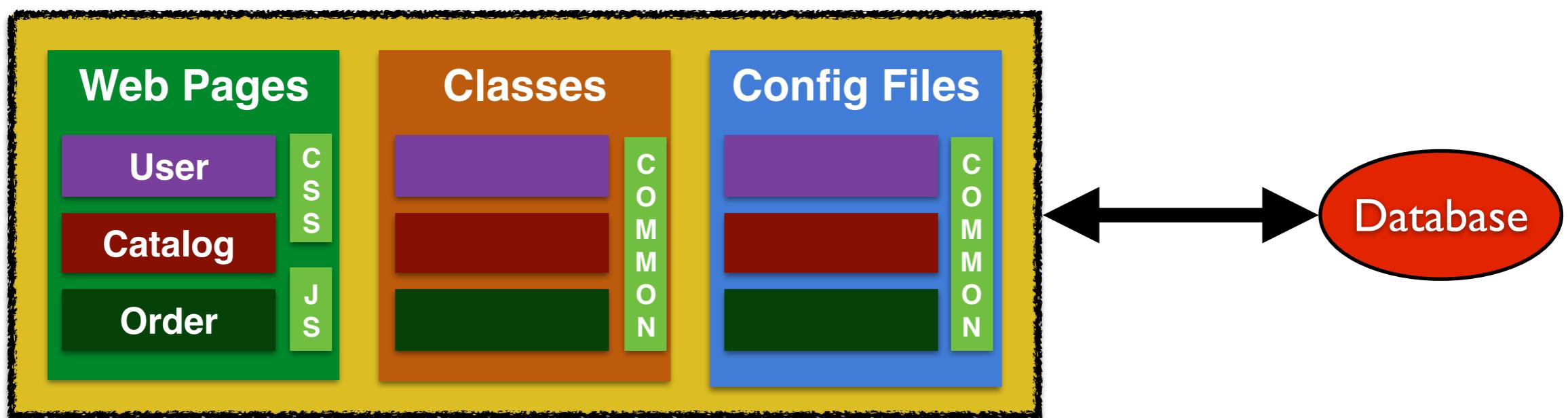
- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can be easily deployed, great enabler of CD
- Each service can scale on X- and Z-axis
- Improves fault isolation
- Eliminates any long-term commitment to a technology stack
- Freedom of choice of technology, tools, frameworks

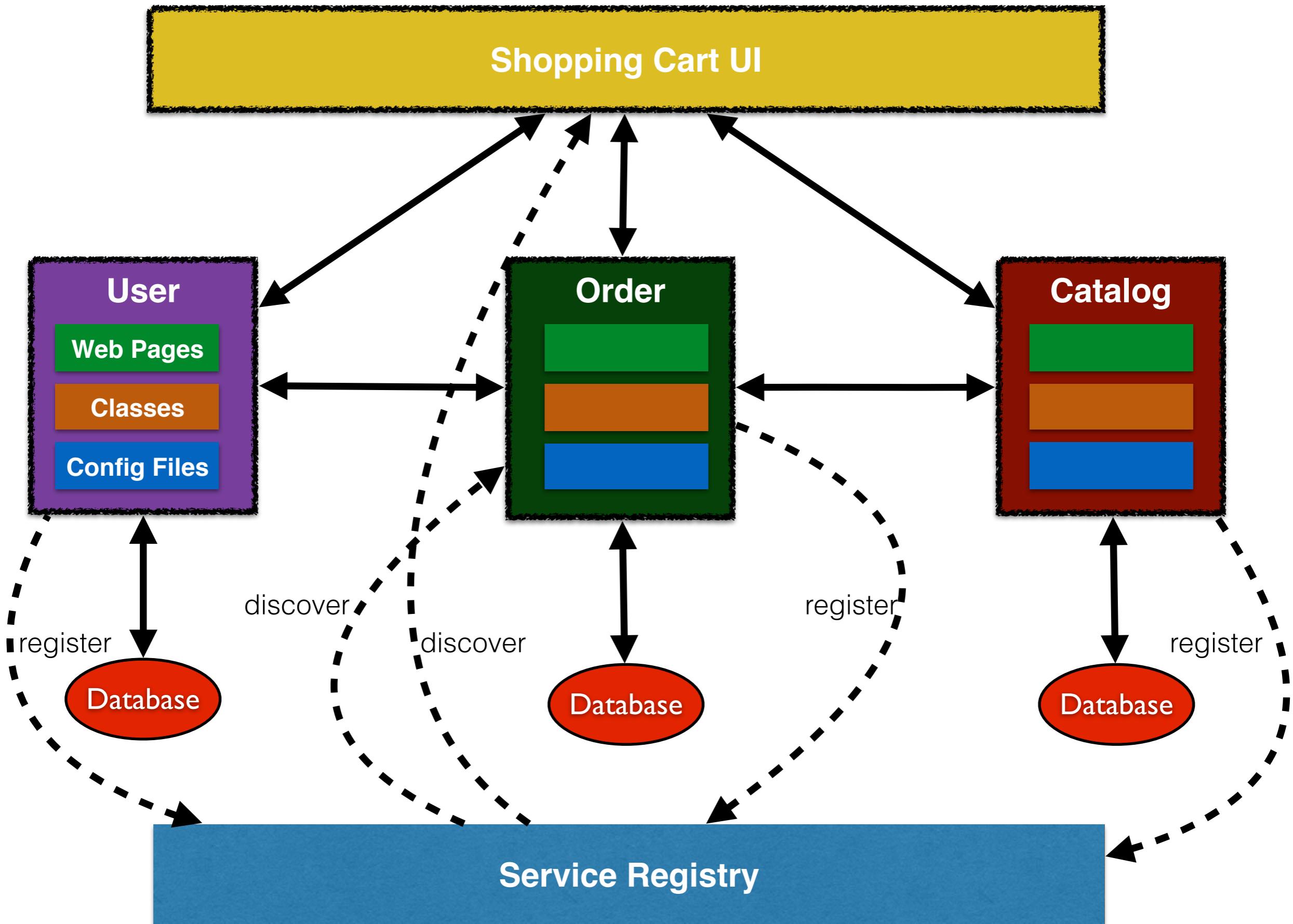
***“If you can't build a [well-structured] monolith, what makes you think microservices are the answer?”***



***“If your monolith is a big ball of mud, your microservice will be a bag of dirt”***

Arun Gupta





# Service Registry/Discovery

- ZooKeeper and Curator
- Snoop
- ...
- Kubernetes
- etcd
- Consul
- OSGi

# Monolith vs Microservice

	Monolith	Microservice
Archives	1	5 (Contracts, Order, User, Catalog, Web)
Web pages	8	8
Config Files	4 (persistence.xml, web.xml, load.sql, template.xhtml)	12 (3 per archive)
Classes	12	26 (Service registration/discovery, Application)
Archive Size	24 KB	~52 KB total

# Design Considerations

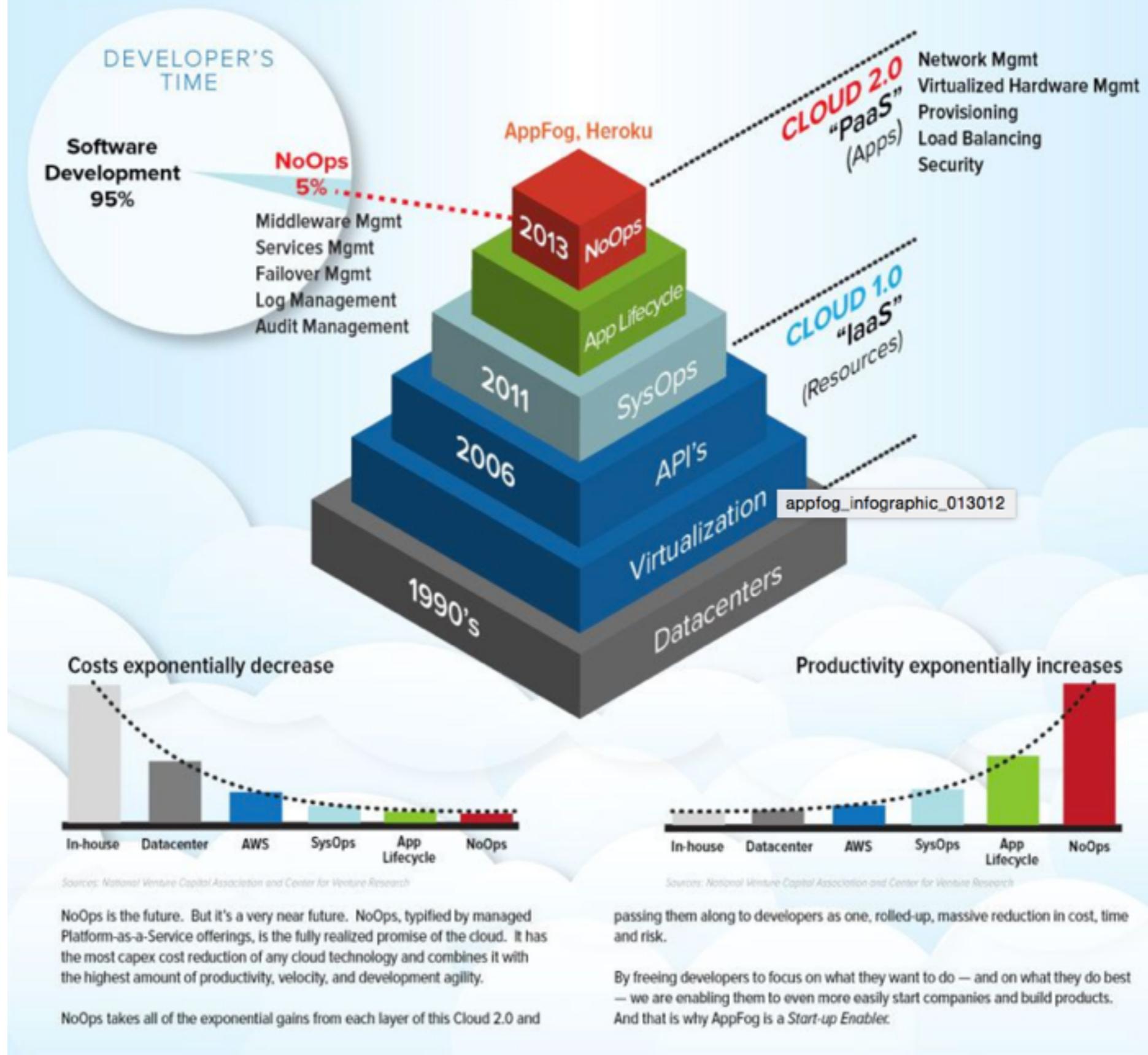
- UI and Full stack
  - Client-side composition (JavaScript?)
  - Server-side HTML generation (JSF?)
  - One service, one UI
- REST Services
- Compensating transactions instead of 2PC

# NoOps

- Service replication (Kubernetes)
- Dependency resolution (Nexus)
- Failover (Circuit Breaker)
- Resiliency (Circuit Breaker)
- Service monitoring, alerts and events (ELK)

## 2013: A bright NoOps future

So where does this all lead? The end-game is NoOps. Where building and running an app is purely a developer process — and where developers are not having to spend time doing Ops work.

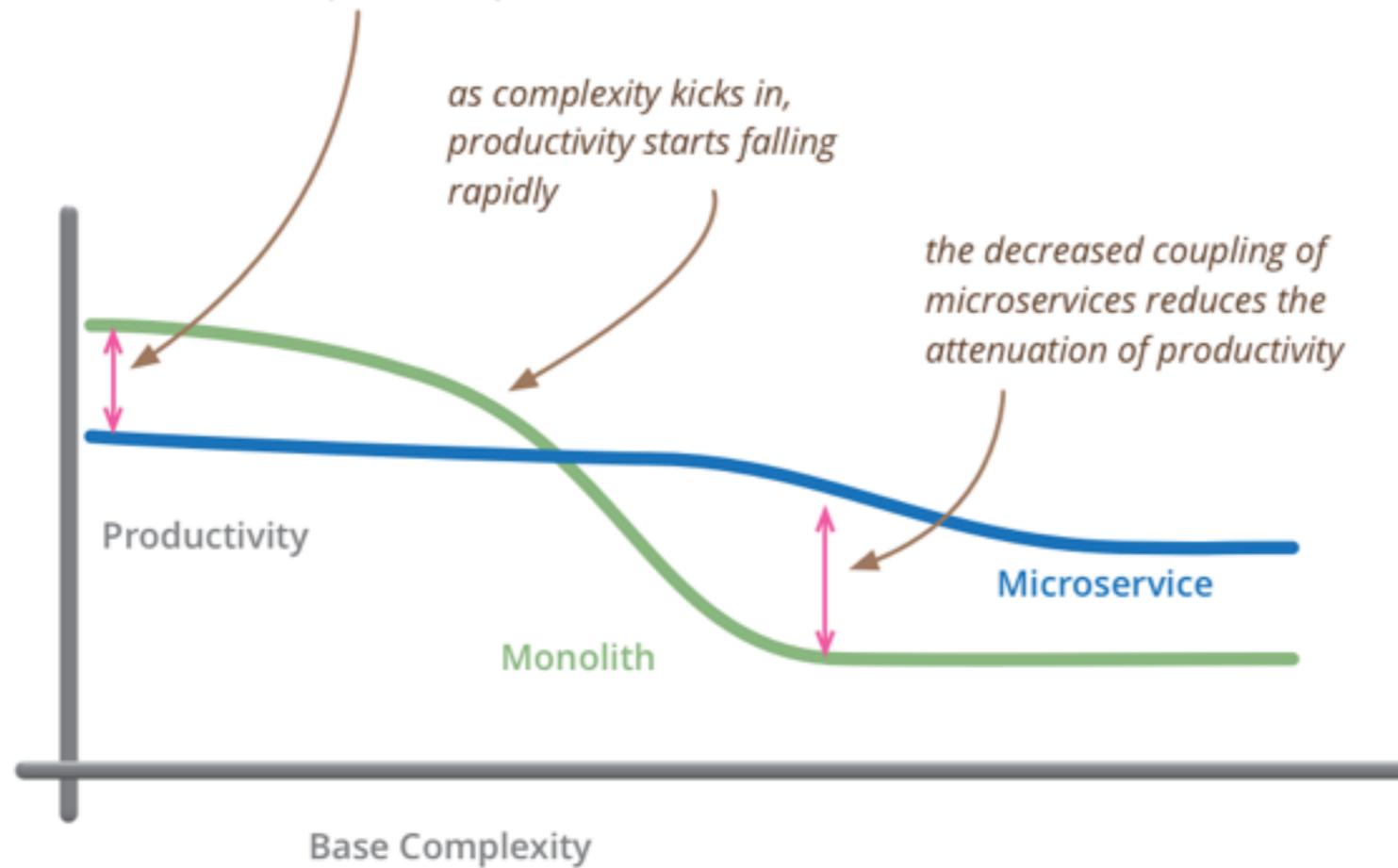


# Drawbacks of microservices

- Additional complexity of distributed systems
- Significant operational complexity, need high-level of automation
- Rollout plan to coordinate deployments
- Slower ROI, to begin with

# Microservice Premium

*for less-complex systems, the extra baggage required to manage microservices reduces productivity*



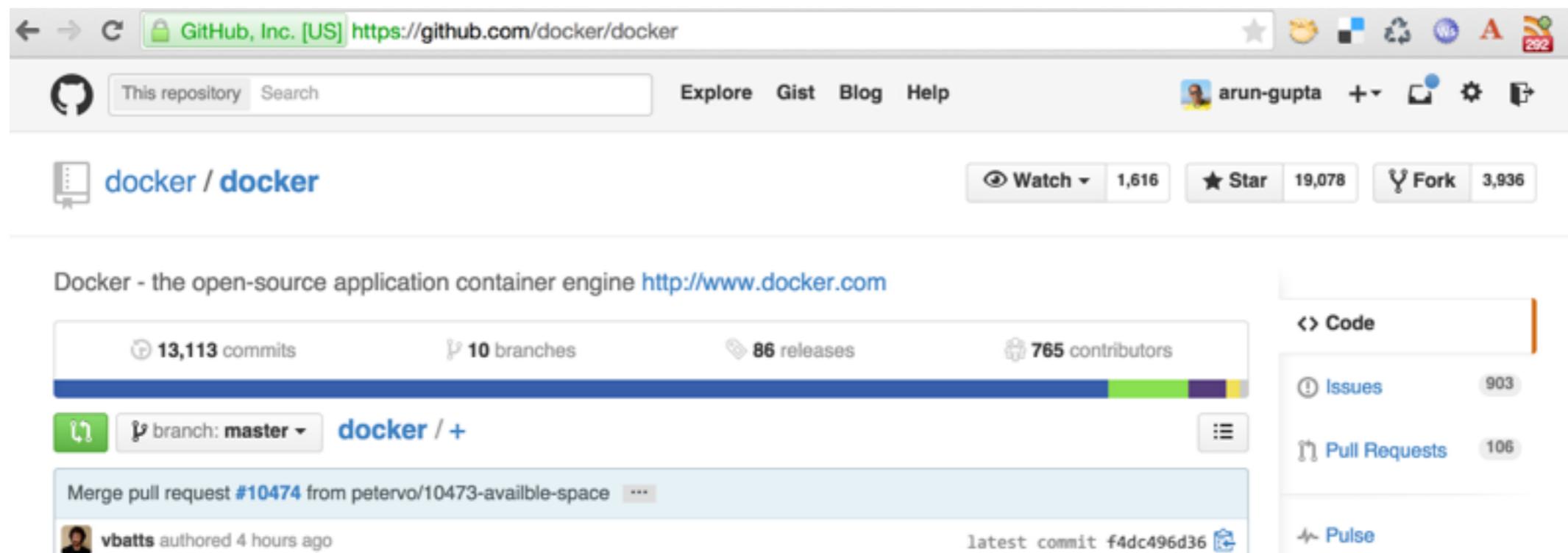
*“don’t even consider microservices unless you have a system that’s too complex to manage as a monolith”*

*but remember the skill of the team will outweigh any monolith/microservice choice*



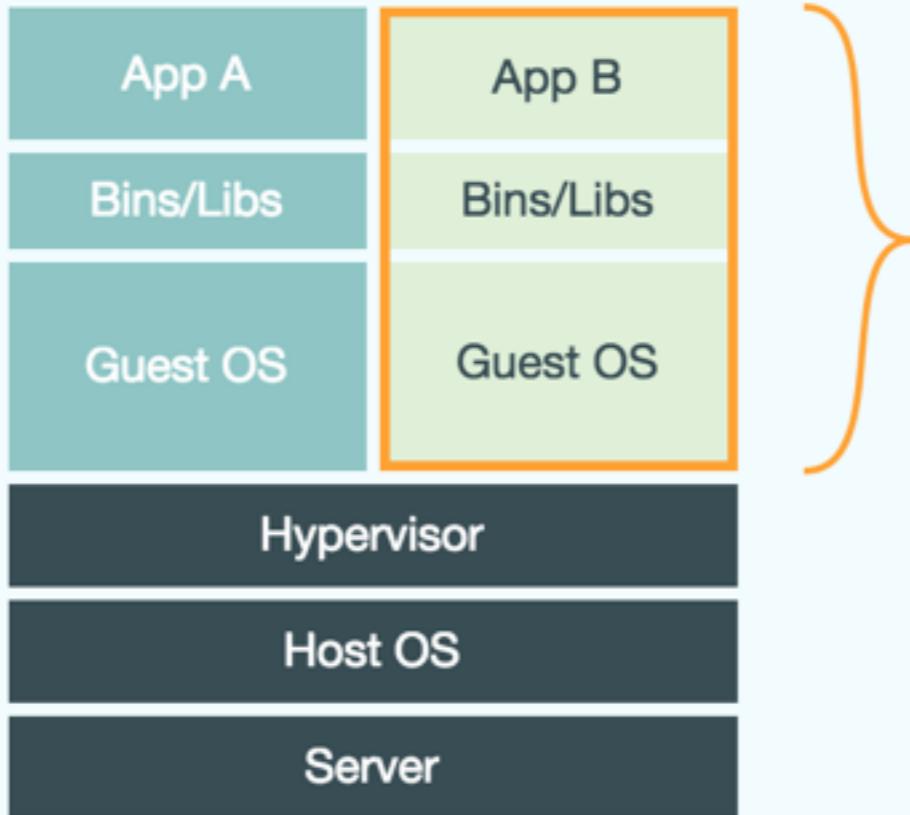
# What is Docker?

- Open source project and company



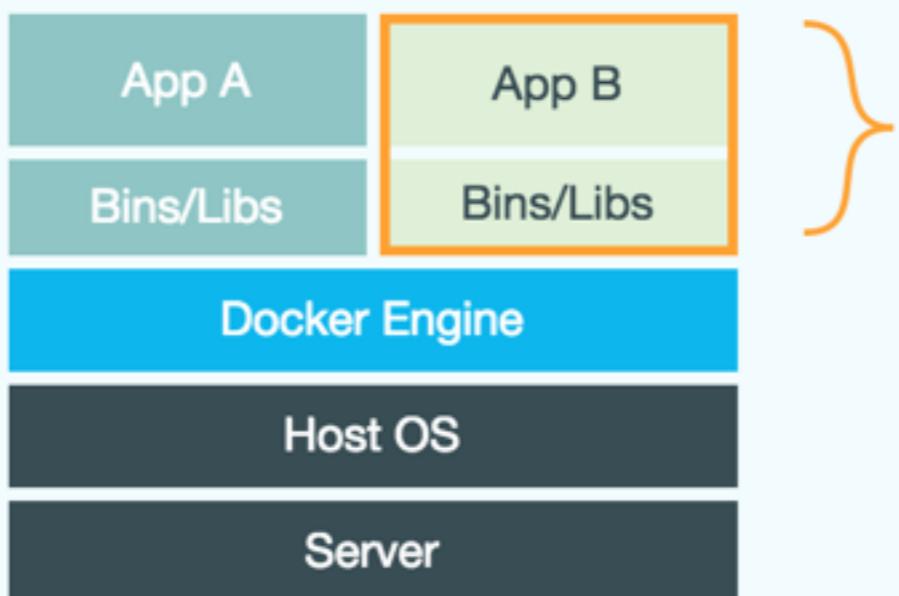
A screenshot of the Docker GitHub repository page (<https://github.com/docker/docker>). The page shows the repository's statistics: 13,113 commits, 10 branches, 86 releases, and 765 contributors. It also displays a merge pull request from petervo/10473-available-space, a commit by vbatts, and the latest commit f4dc496d36. The right sidebar shows code navigation, issues (903), pull requests (106), and a pulse summary.

- Used to create containers for software applications
- Package Once Deploy Anywhere (PODA)



## Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



## Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

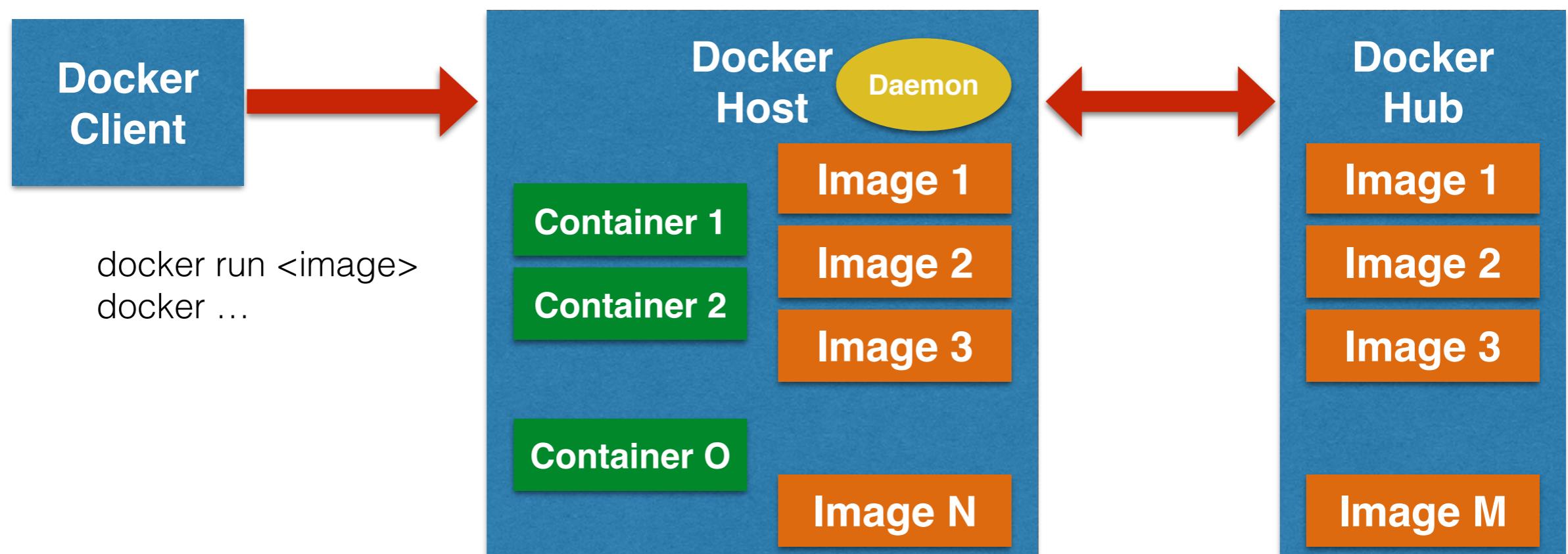
# Underlying Technology

- Written in Go



- Uses several Linux features
  - **Namespaces** to provide isolation
  - **Control groups** to share/limit hardware resources
  - **Union File System** makes it light and fast
  - **libcontainer** defines container format

# Docker Workflow





Build

Develop an app using Docker containers with  
any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image

```
FROM fedora:latest
```

```
CMD echo "Hello world"
```

```
FROM jboss/wildfly
```

```
RUN curl -L https://github.com/javaee-samples/javaee7-hol/raw/master/solution/  
movieplex7-1.0-SNAPSHOT.war -o /opt/jboss/wildfly/standalone/deployments/  
movieplex7-1.0-SNAPSHOT.war
```

# Advantages of Containers

- Faster deployments
- Isolation
- Portability - “it works on my machine”
- Snapshotting
- Security sandbox
- Limit resource usage
- Simplified dependency
- Sharing

# Docker: Pros and Cons

- PROS
  - Extreme application portability
  - Very easy to create and work with derivative
  - Fast boot on containers
- CONS
  - Machine, Swarm, Compose not ready for production
  - No usage tracking/reporting

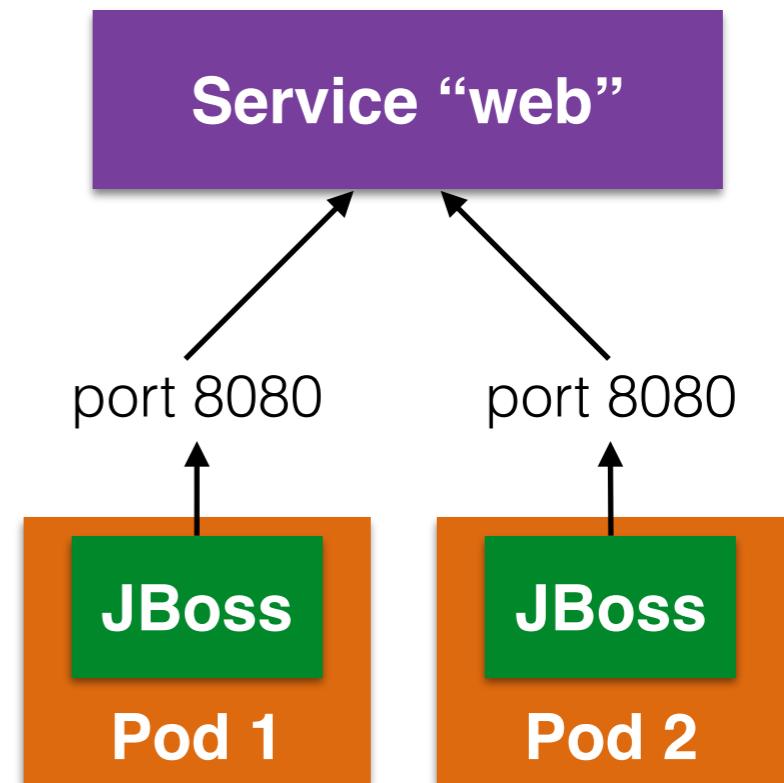
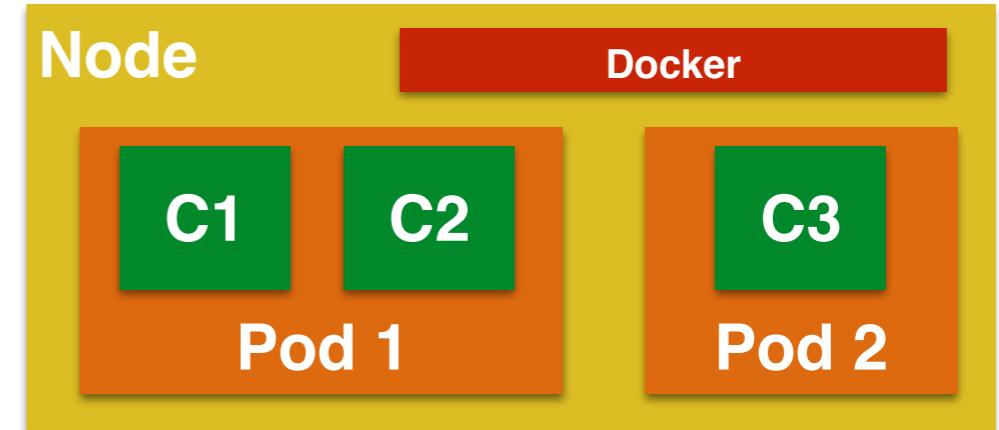
# Kubernetes



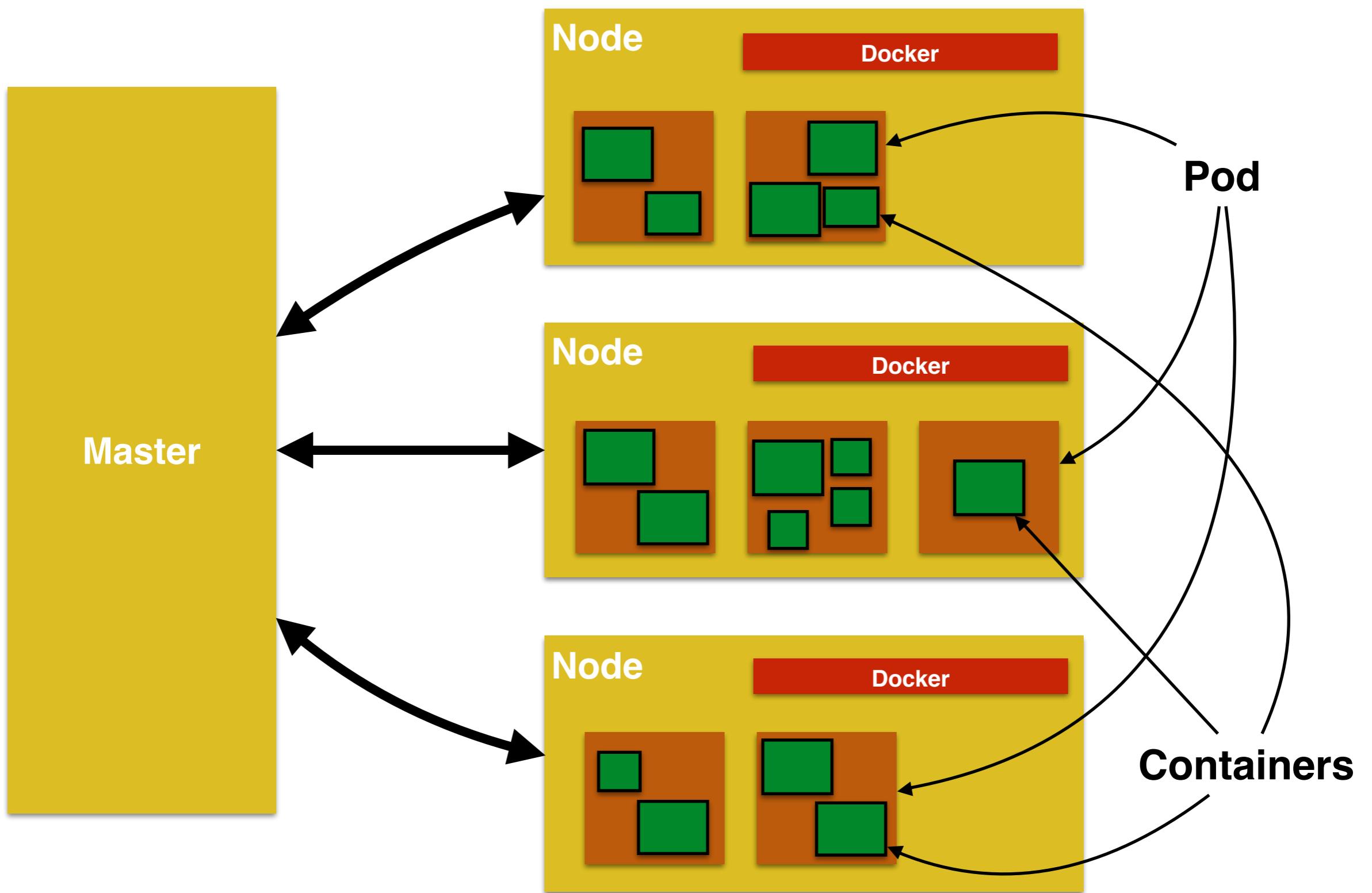
- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”
  - Self-healing
  - Auto-restarting
  - Schedule across hosts
  - Replicating

# Concepts

- **Pods**: collocated group of Docker containers that share an IP and storage volume
- **Service**: Single, stable name for a set of pods, also acts as LB
- **Label**: used to organize and select group of objects



# Kubernetes Architecture

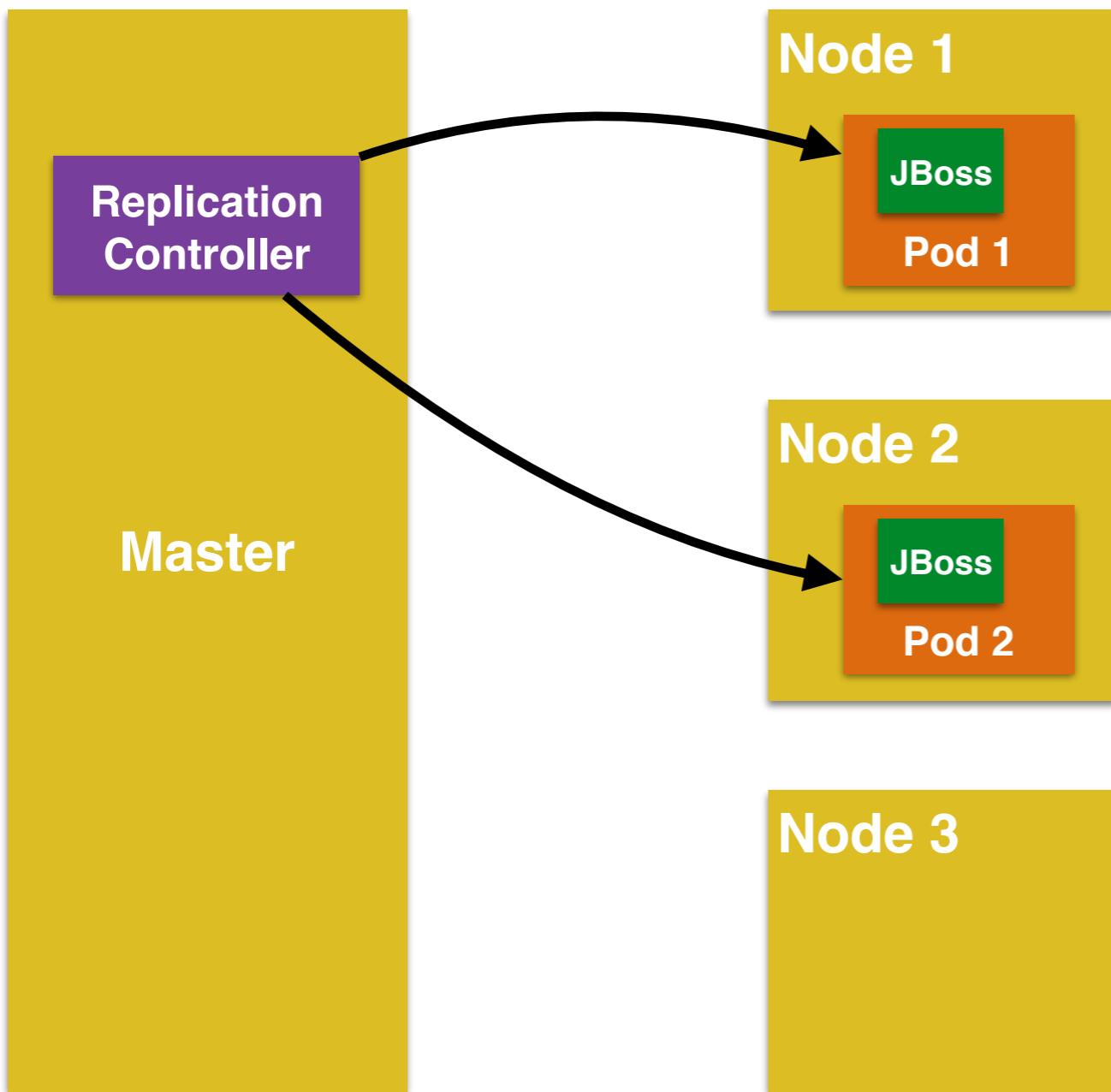




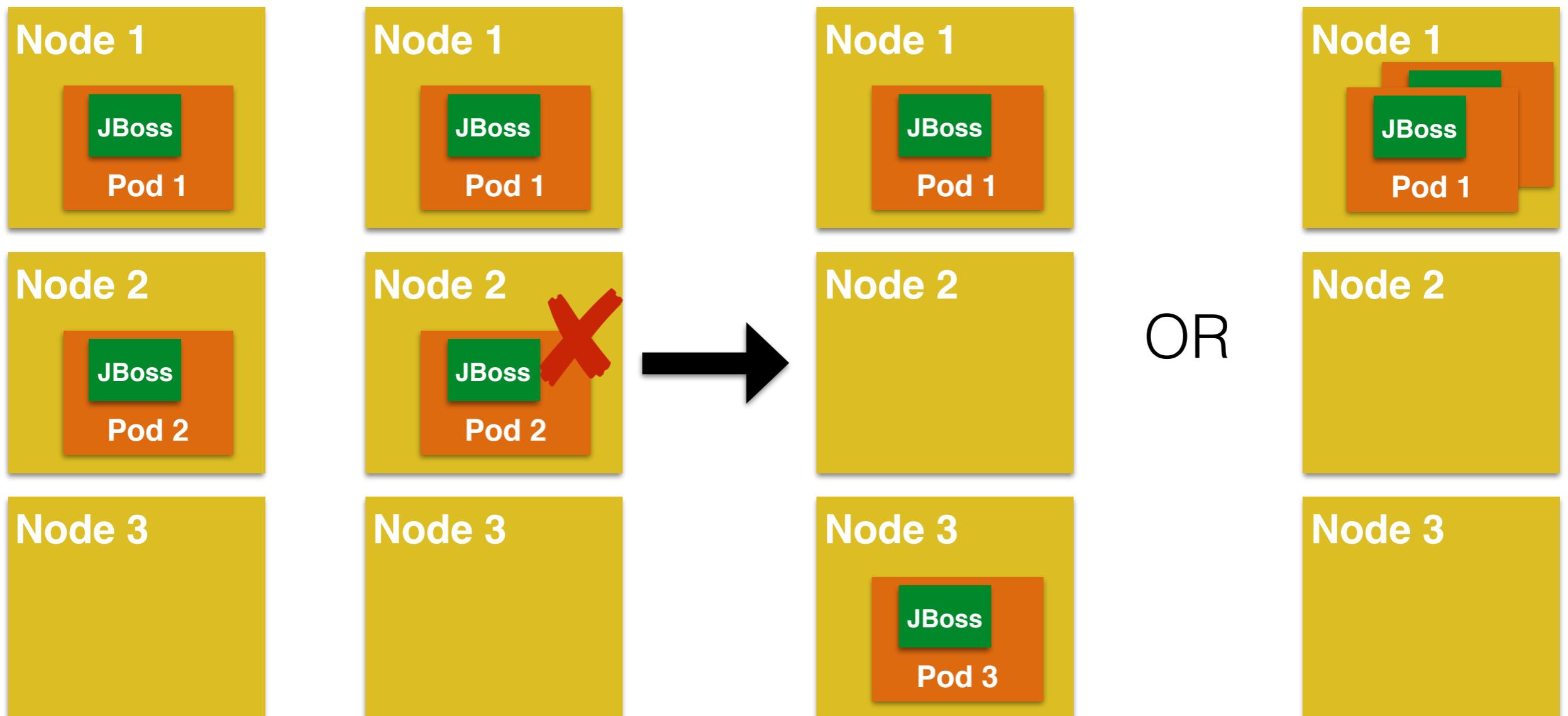
# Replication Controller

- Ensures that a specified number of pod "replicas" are running at any one time
- Recommended to wrap a Pod in a RC
- Only appropriate for Pods with **Restart=Always** policy (default)

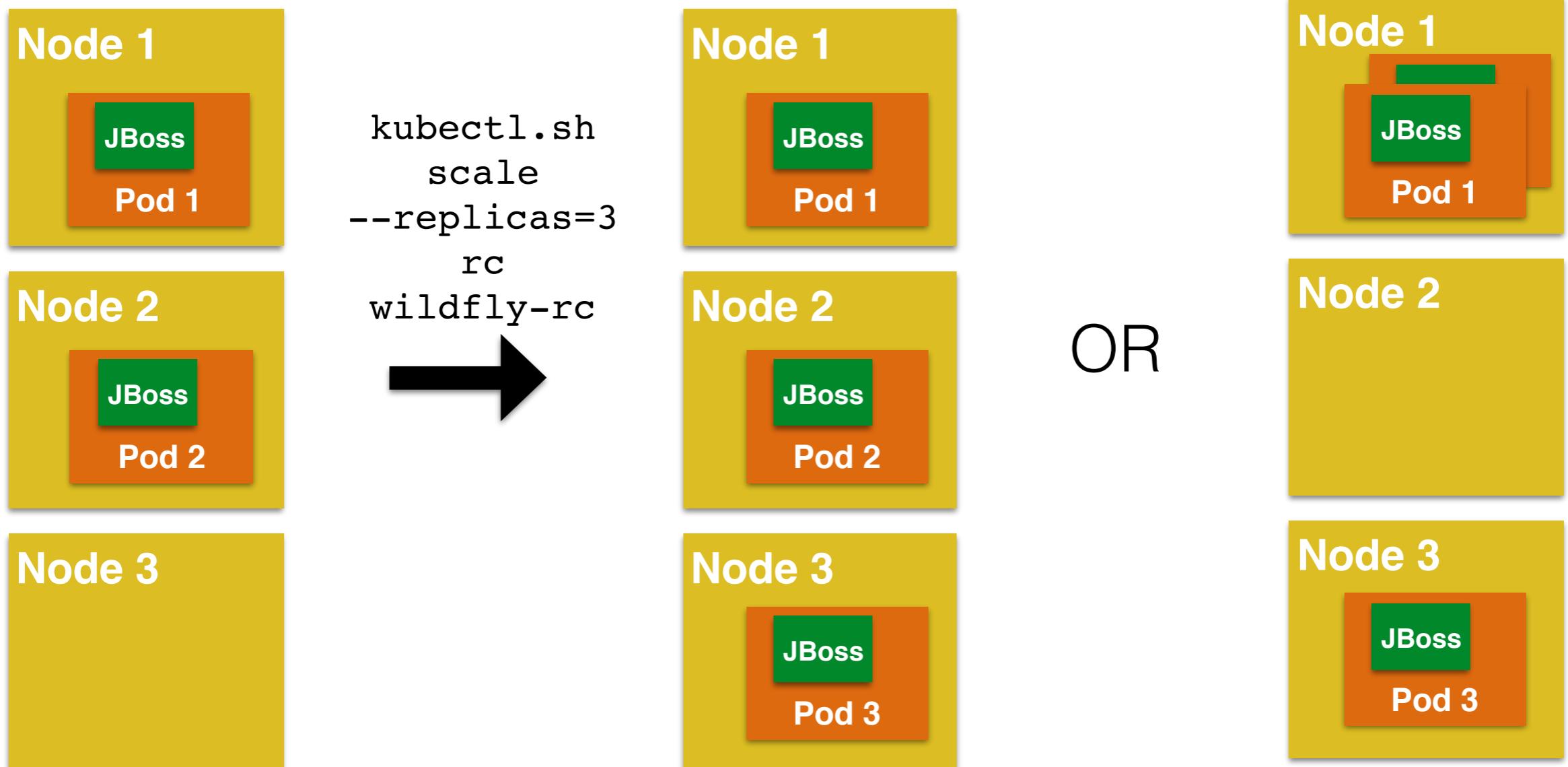
# Replication Controller



# Replication Controller: Automatic Rescheduling



# Replication Controller: Scaling



# Kubernetes Config

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: wildfly-pod
5   labels:
6     name: wildfly
7 spec:
8   containers:
9     - image: jboss/wildfly
10    name: wildfly-pod
11    ports:
12      - containerPort: 8080
```

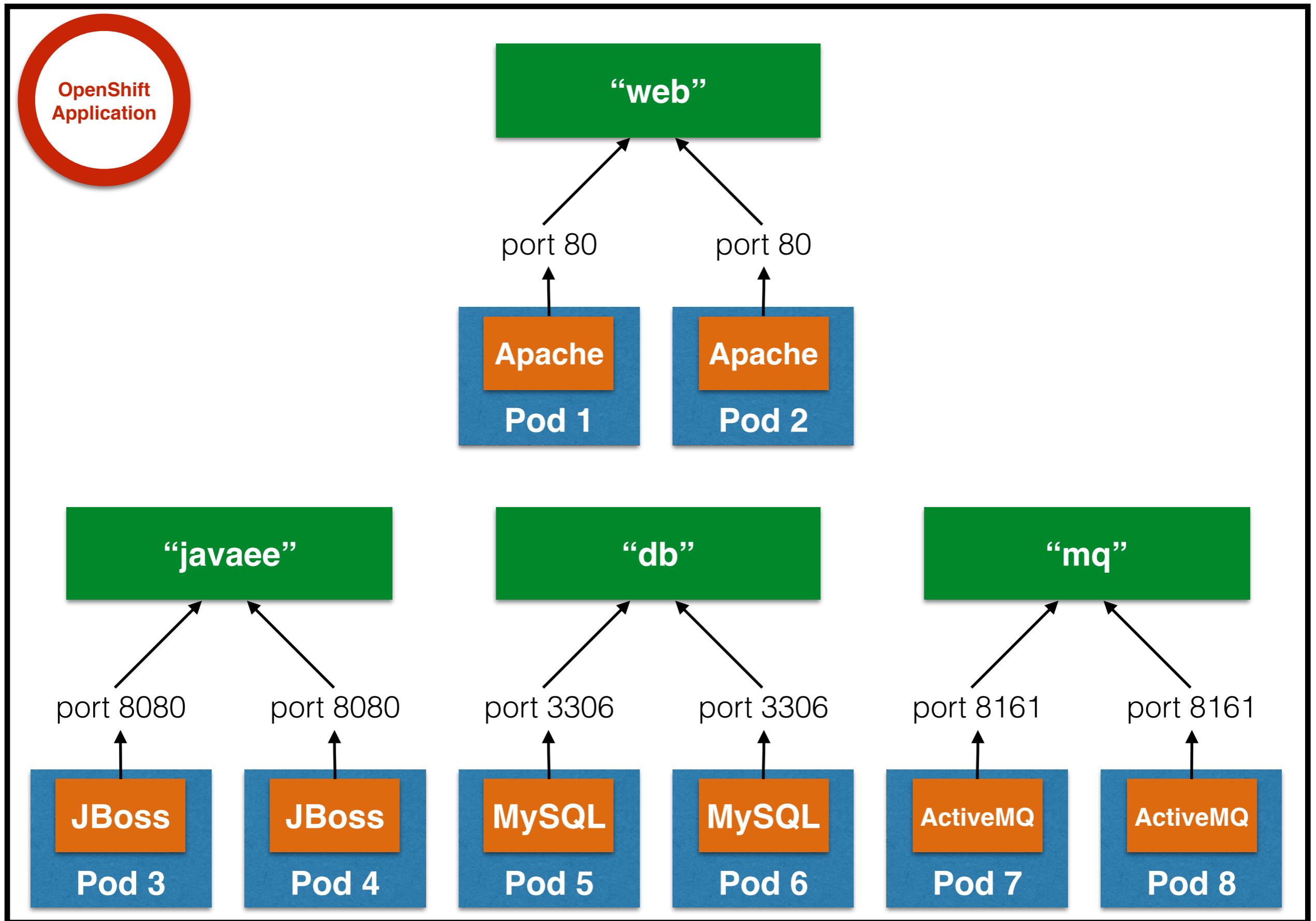
```
1 apiVersion: v1
2 kind: ReplicationController
3 metadata:
4   name: wildfly-rc
5   labels:
6     name: wildfly
7 spec:
8   replicas: 2
9   template:
10  metadata:
11    labels:
12      name: wildfly
13
14  spec:
15    containers:
16      - name: wildfly-rc-pod
17        image: jboss/wildfly
18        ports:
19          - containerPort: 8080
```

# Kubernetes: Pros and Cons

- PROS
  - Manage related Docker containers as a unit
  - Container communication across hosts
  - Availability and scalability through automated deployment and monitoring of pods and their replicas, across hosts

# Kubernetes: Pros and Cons

- CONS
  - Lifecycle of applications - build, deploy, manage, promote
  - Port existing source code to run in Kubernetes
  - DevOps: Dev -> Test -> Production
  - No multi-tenancy
  - On-premise (available on GCE)
    - Assumes inter-pod networking as part of infrastructure
    - Requires explicit load balancer





redhat.

## Applications

xPaaS  
VERT.X

RED HAT® JBOSS®  
MIDDLEWARE  
node.js™

## PaaS

openshift

## Containers & Orchestration

kubernetes

docker

## Container Host



RED HAT®  
ENTERPRISE LINUX®  
ATOMIC HOST

## IaaS

openstack™

jboss.org



VERT.X



FEEDHENRY™

Infinispan



Qhawtio

