redhat.

RED HAT® JBOSS®
MIDDLEWARE

# Refactoring your Java EE applications using Microservices and Containers

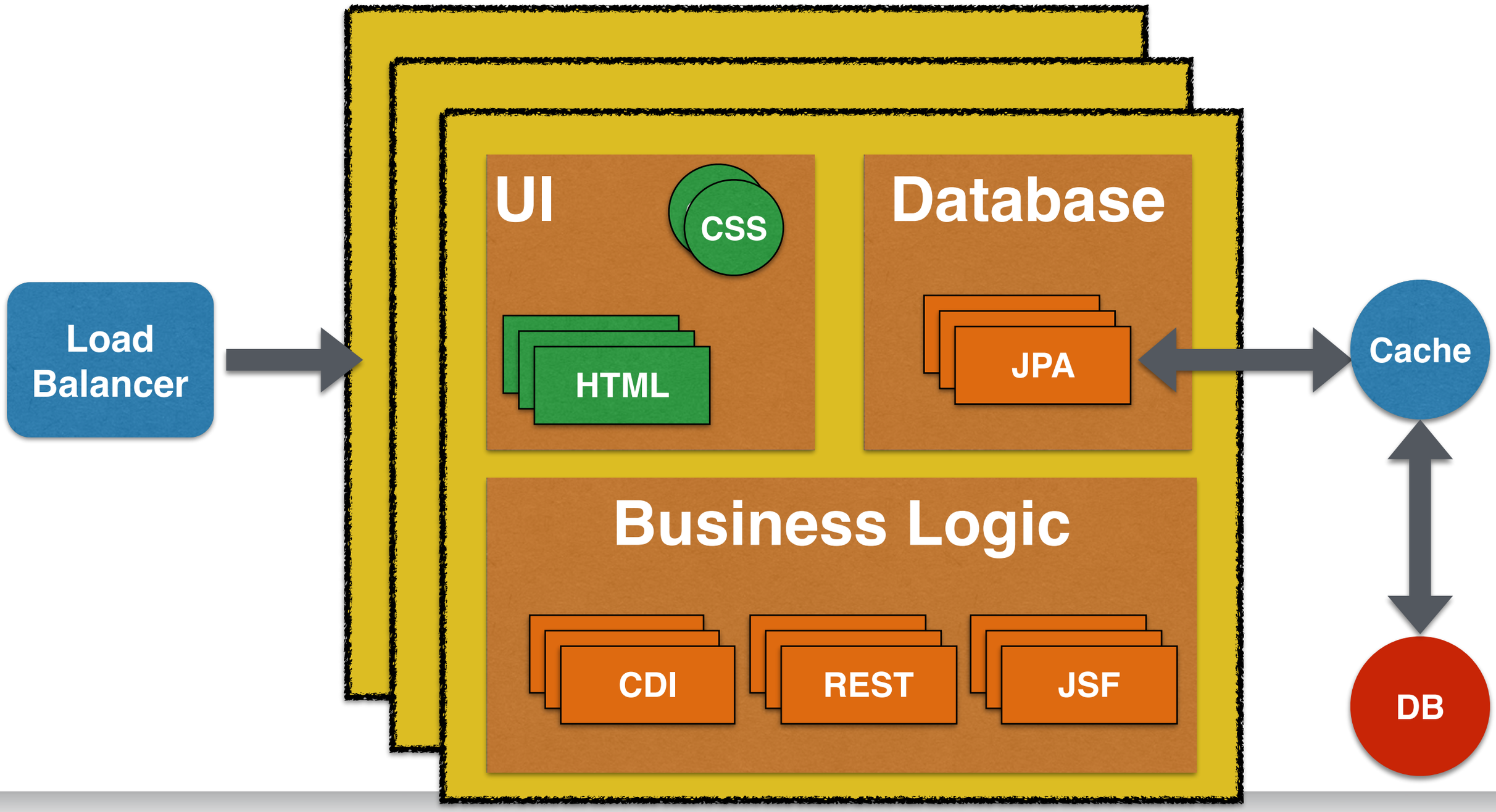
Arun Gupta, Red Hat

# Arun Gupta

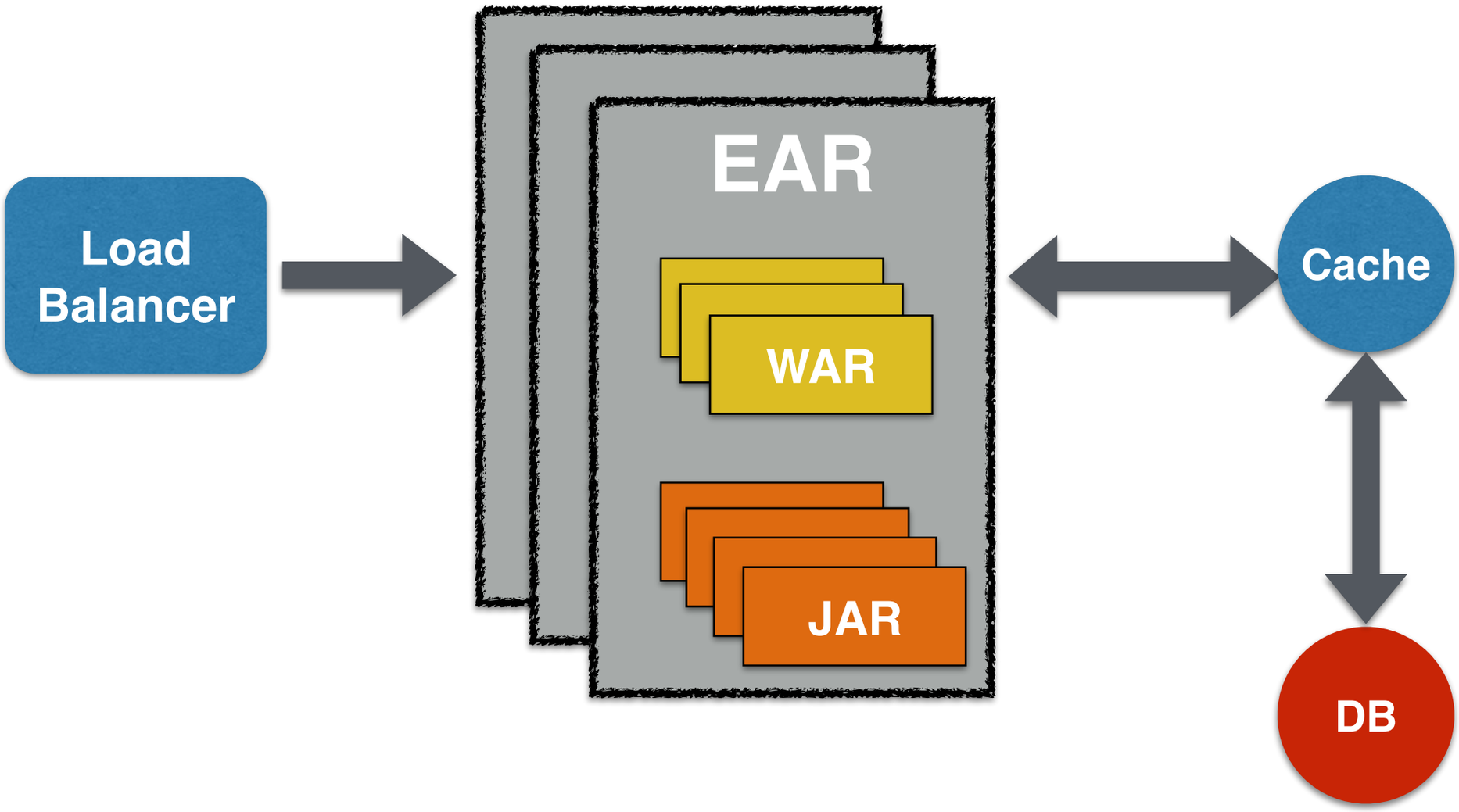
## Director, Developer Advocacy

@arungupta
blog.arungupta.me
arungupta@redhat.com

# Monolith Application



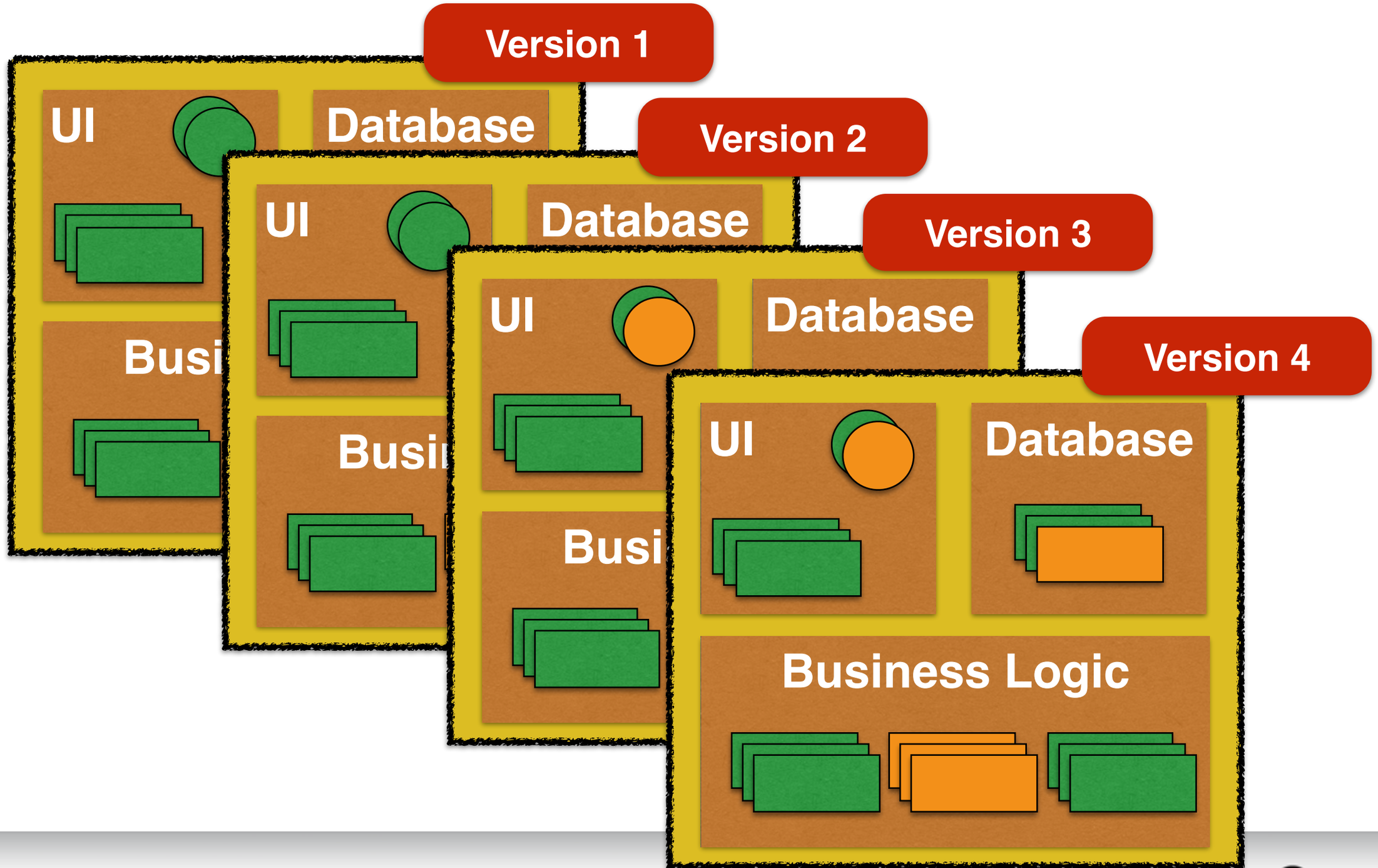
# Monolith Application



# Advantages of Monolith Application

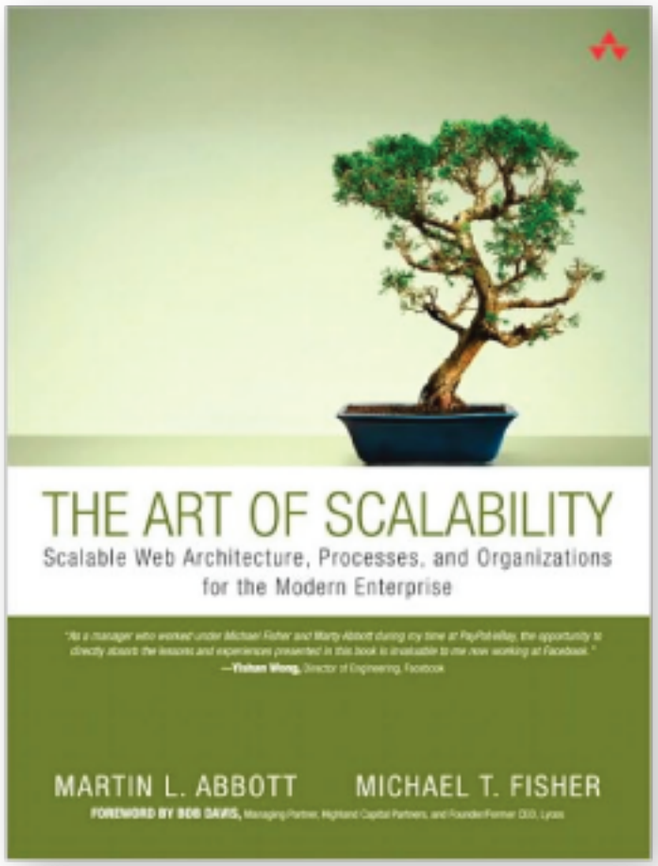
- Typically packaged in a single `.ear`
- Easy to test (all required services are up)
- Simple to develop

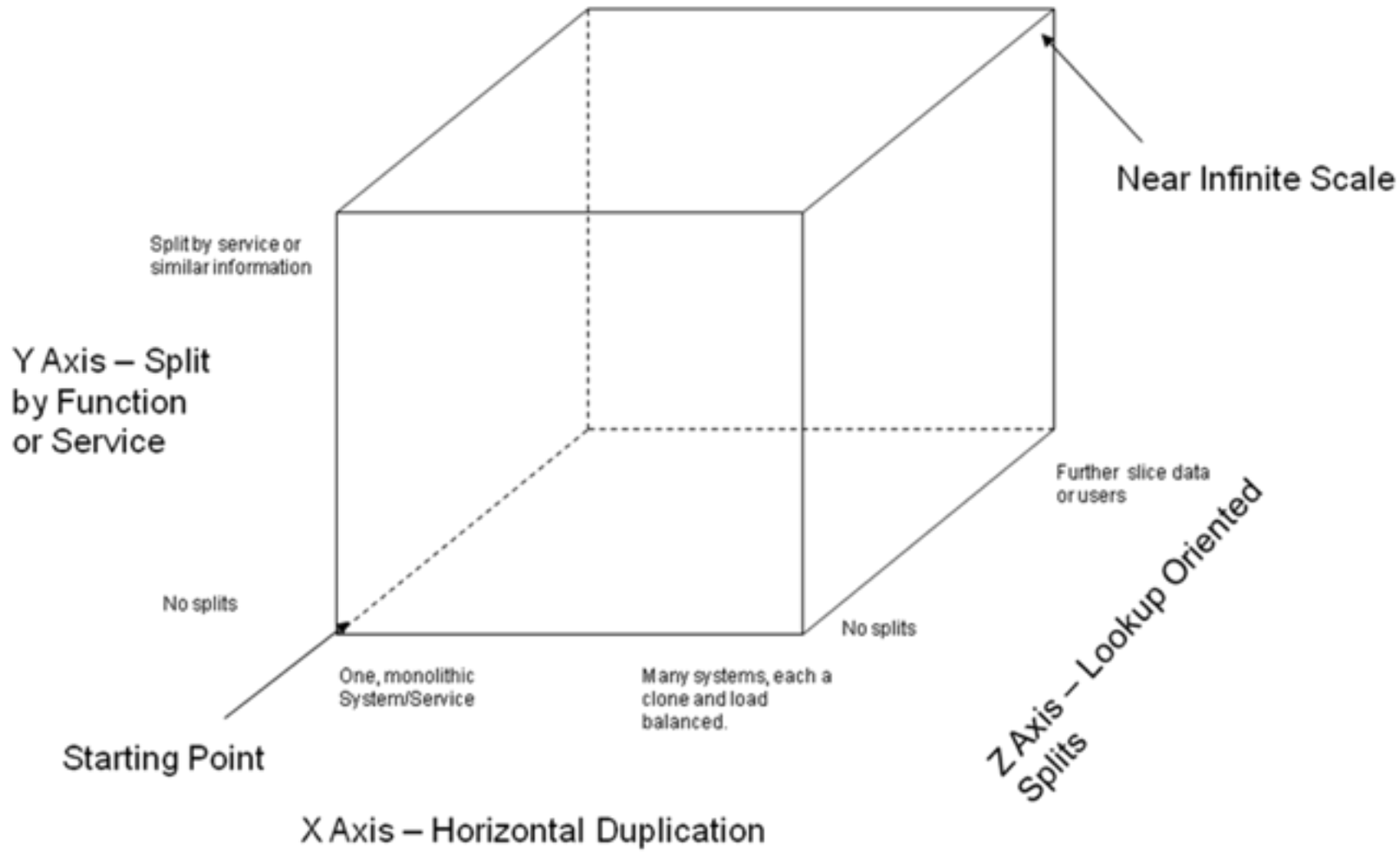
# Monolith Application



# Disadvantages of Monolith Application

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



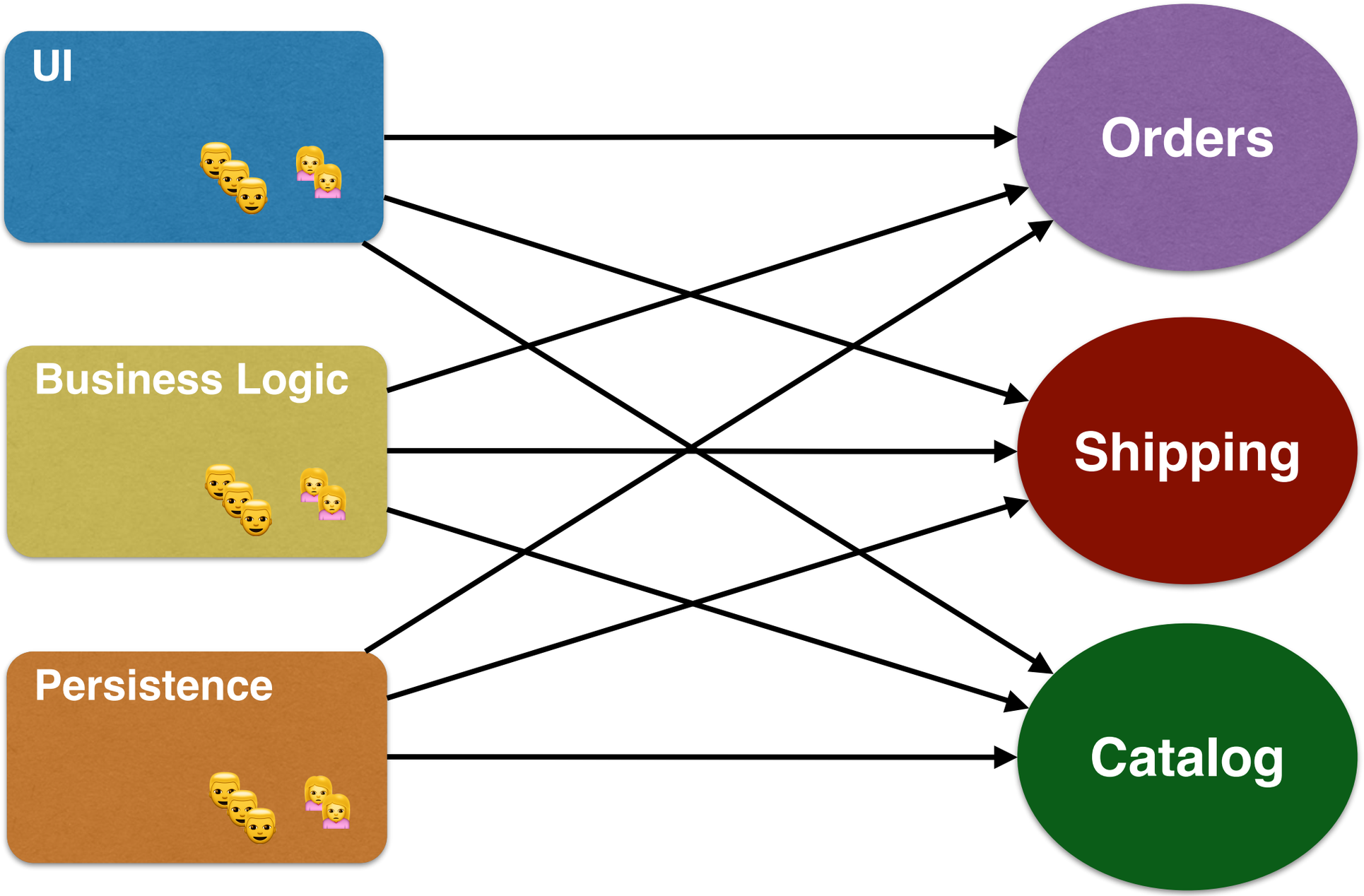

http://akfpartners.com/techblog/2008/05/08/splitting-applications-or-services-for-scale/

*"building applications as **suites of services**. As well as the fact that services are **independently deployable and scalable**, each service also provides a **firm module boundary**, even allowing for different services to be written in **different programming languages**. They can also be **managed by different teams**"*

–Martin Fowler

http://martinfowler.com/articles/microservices.html

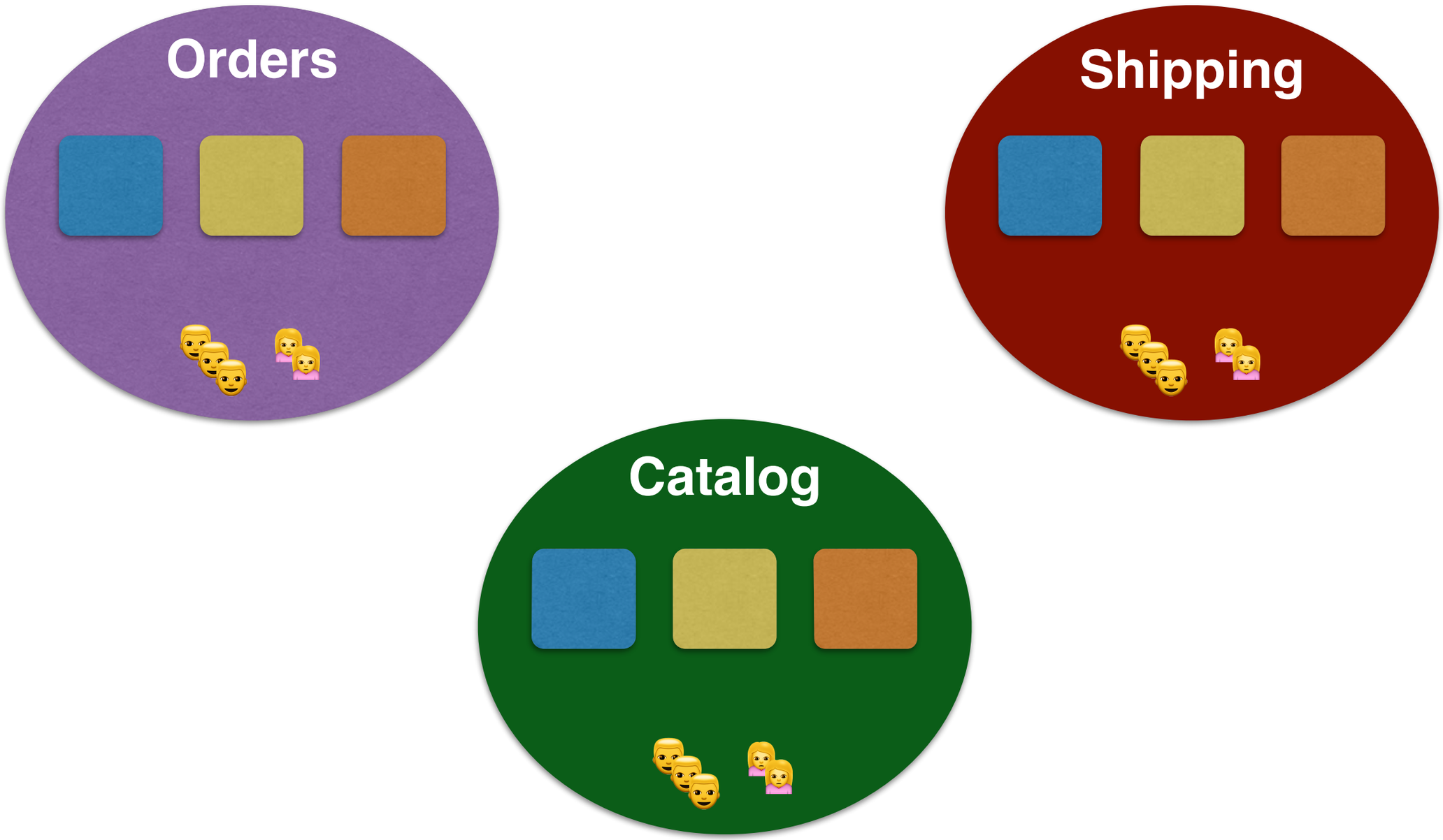

I DONT ALWAYS
BUILD EVERYTHING
BUT WHEN I DO I
USE MICRO SERVICES
imgflip.com




*"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
http://www.melconway.com/Home/Committees_Paper.html

Characteristics

# Teams around business capability



Characteristics

# Single Responsibility Principle

Characteristics

# Explicitly Published interface

Characteristics

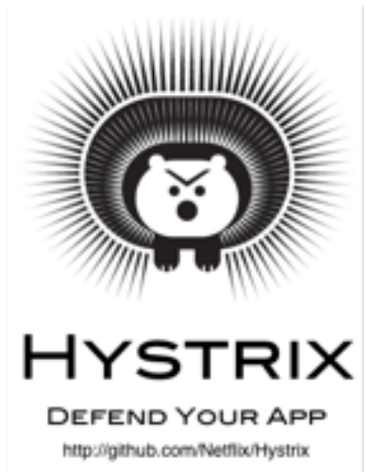
# Independently replaceable and upgradeable

Characteristics

With great power, comes great responsibility

“you build it, you run it!”

# Designed for failure


*Fault tolerance is a requirement, not a feature*

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

# Designed for failure



*Fault tolerance is a requirement, not a feature*

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

Characteristics













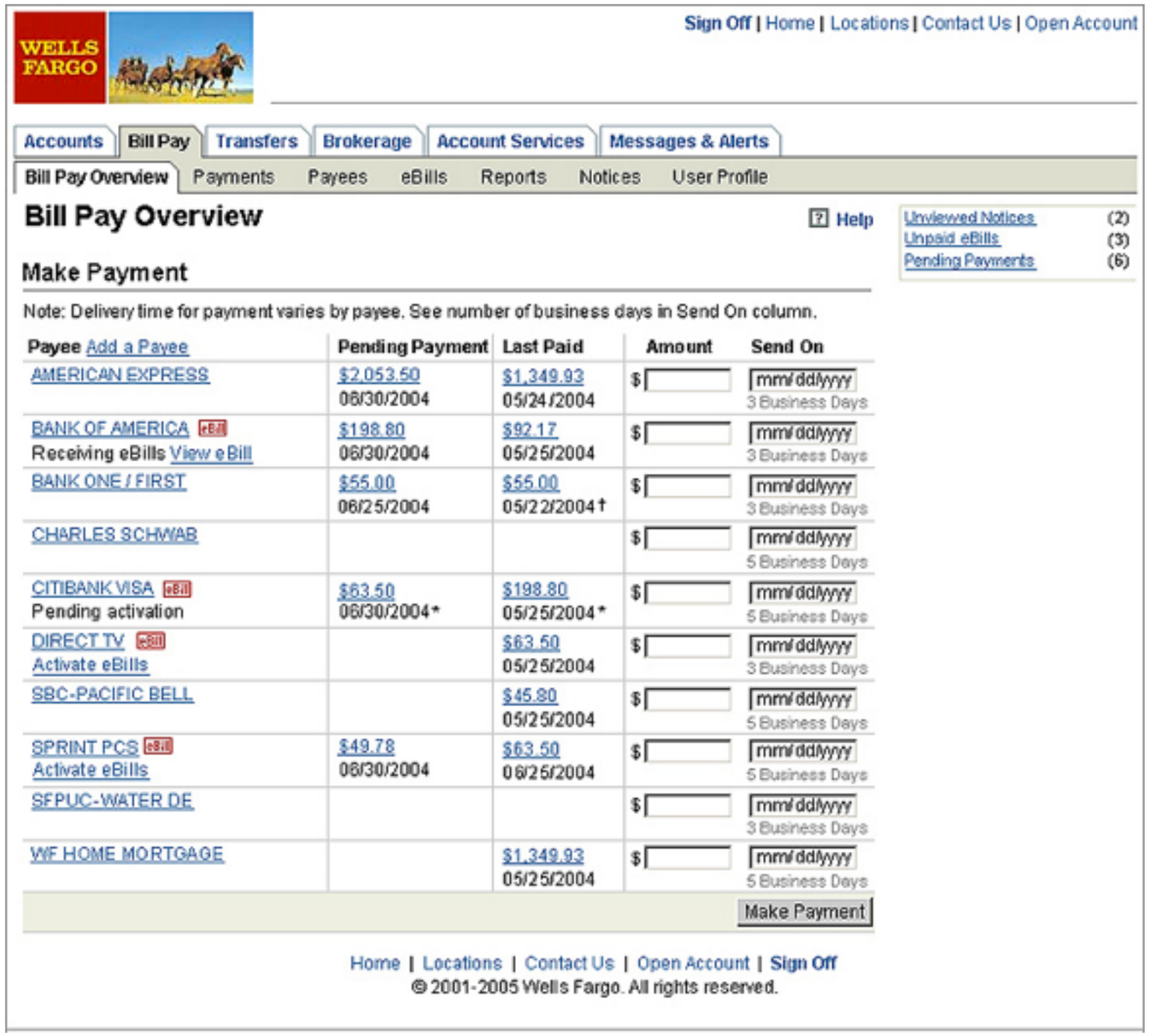






Characteristics

# 100% automated



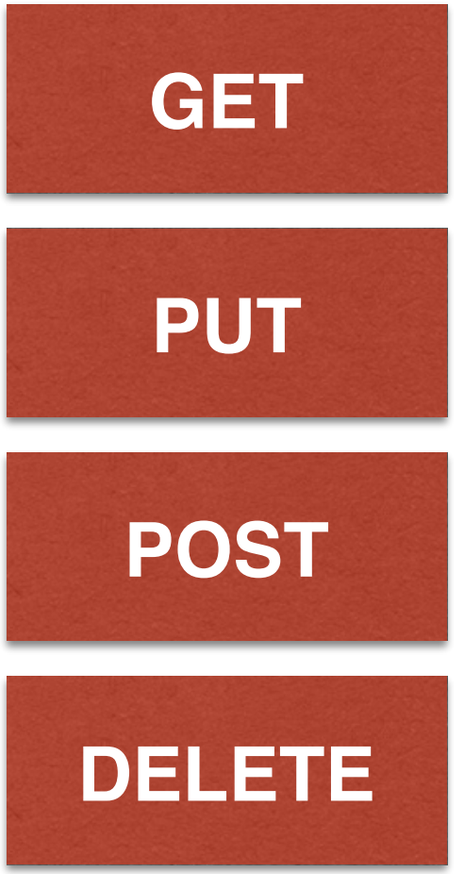

redhat

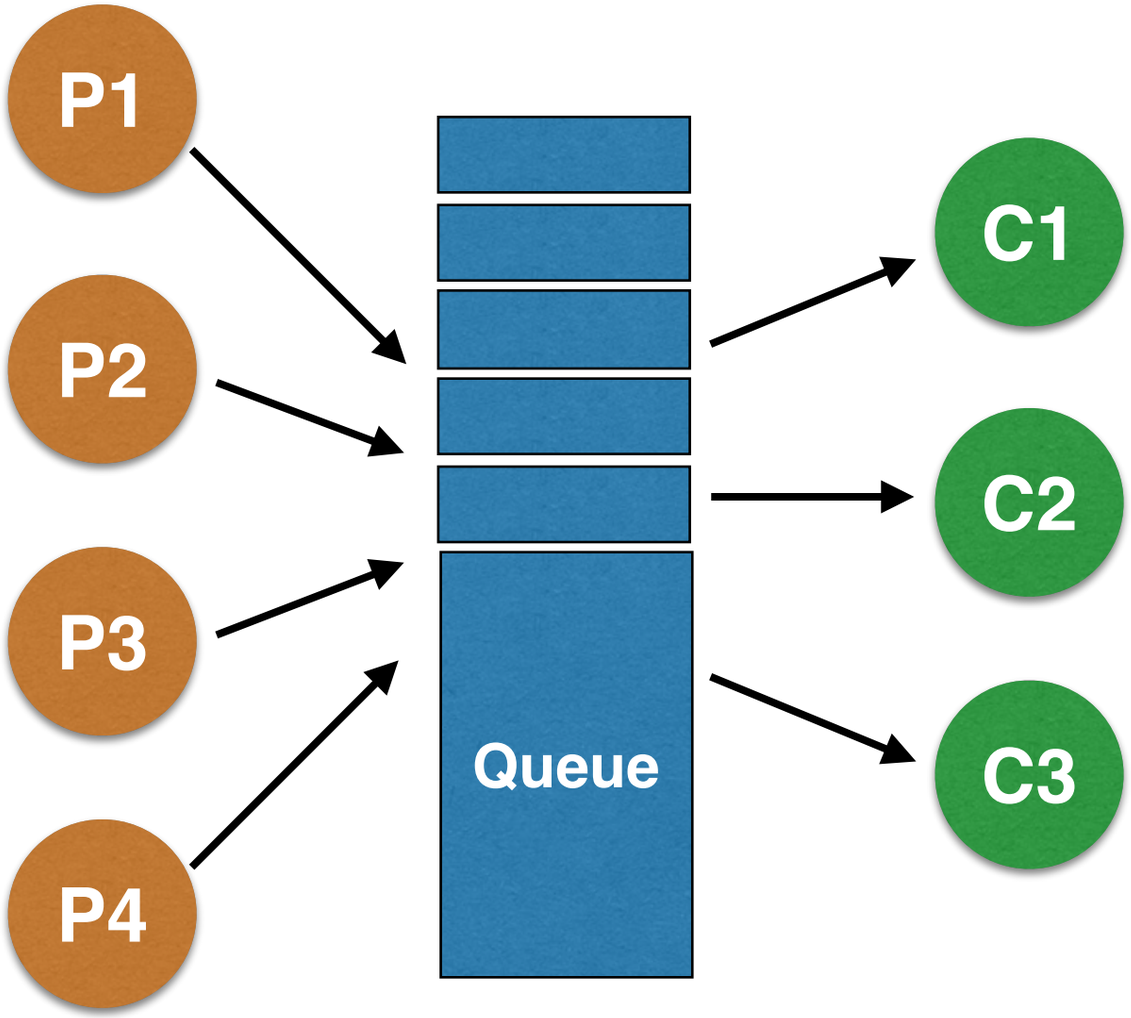
Characteristics

# Sync or Async Messaging


Characteristics

# REST vs Pub/Sub



Characteristics

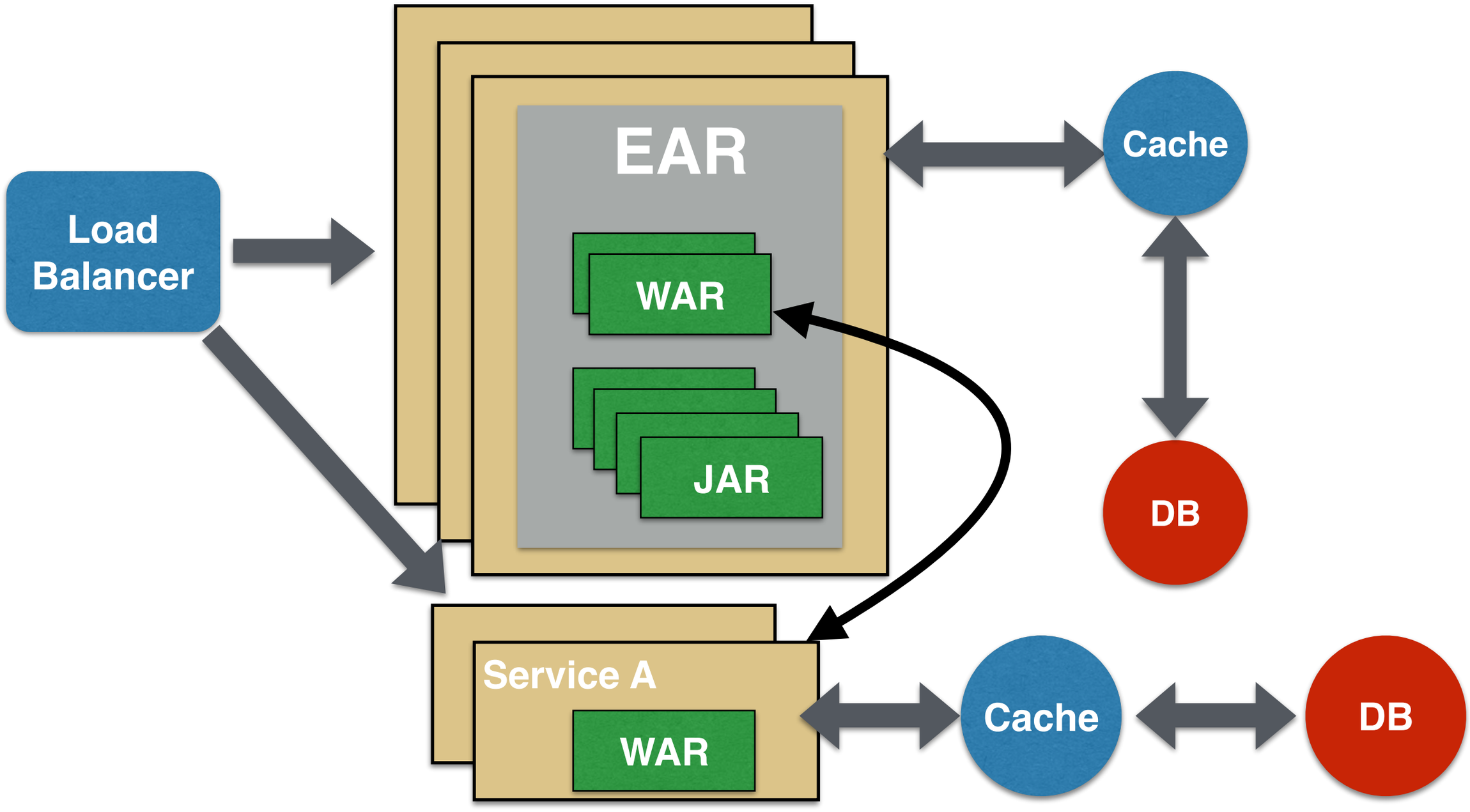
# “Smart endpoints Dumb pipes”

# Strategies for decomposing

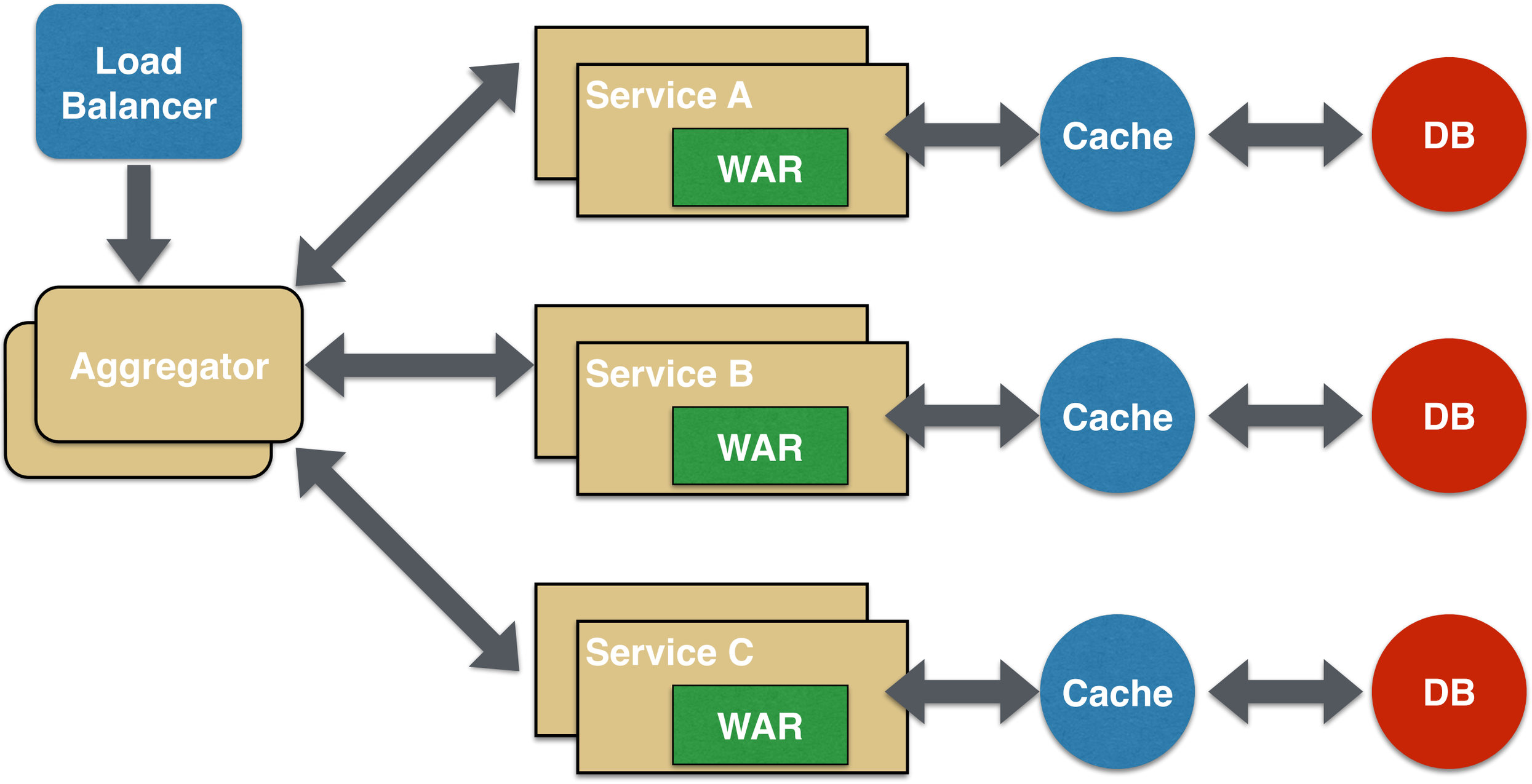
# Strategies for decomposing

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

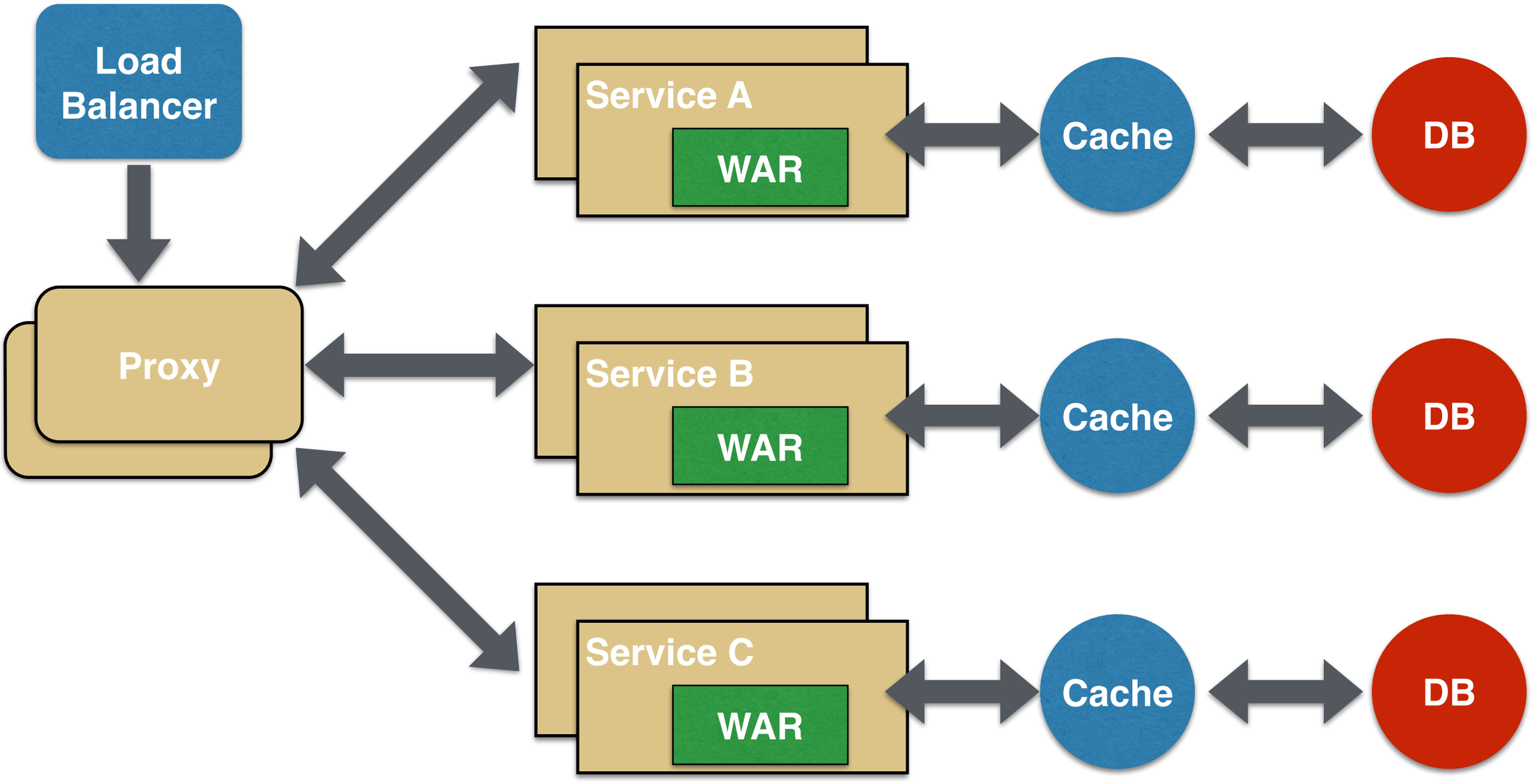
# Towards microservices



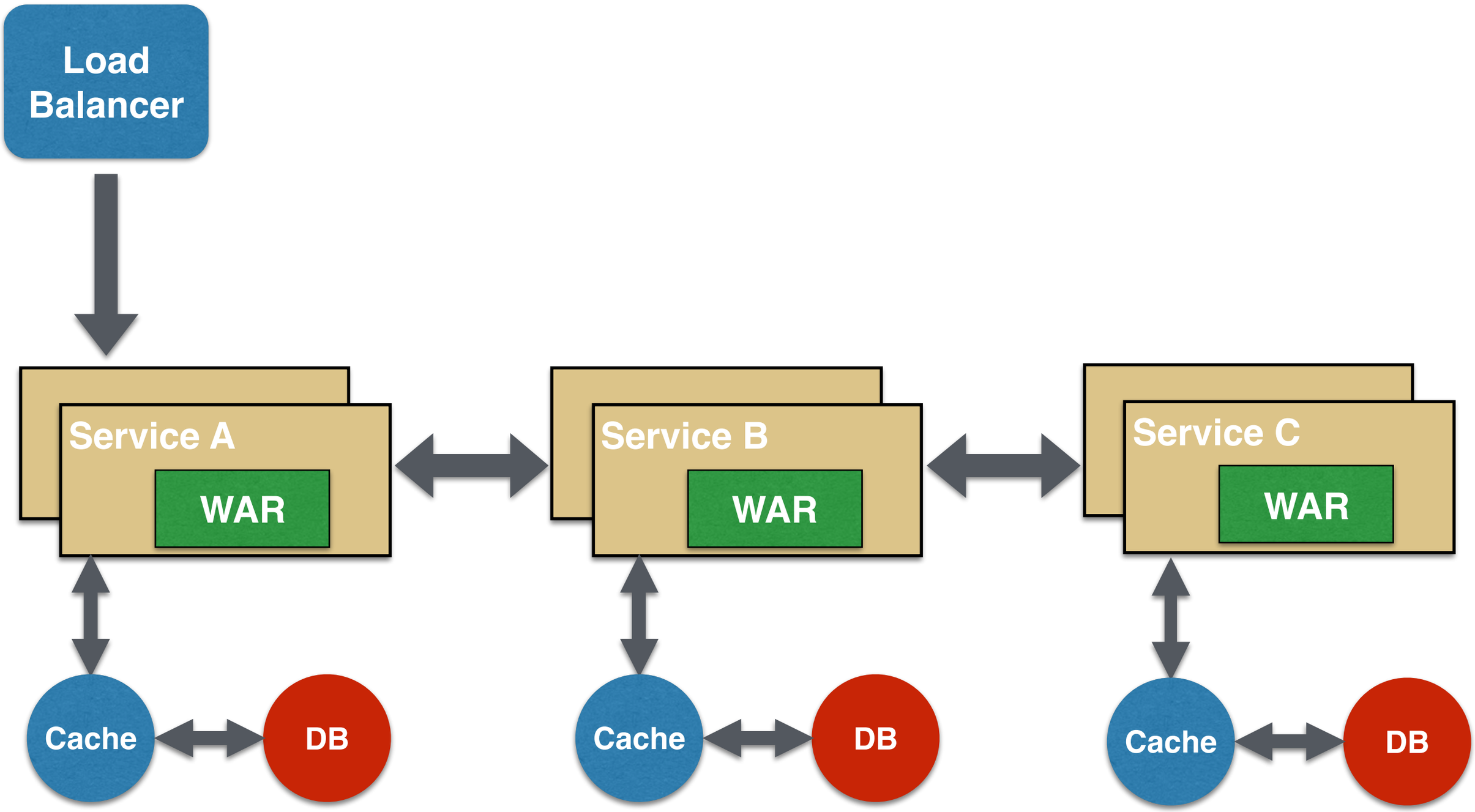
# Aggregator Pattern #1



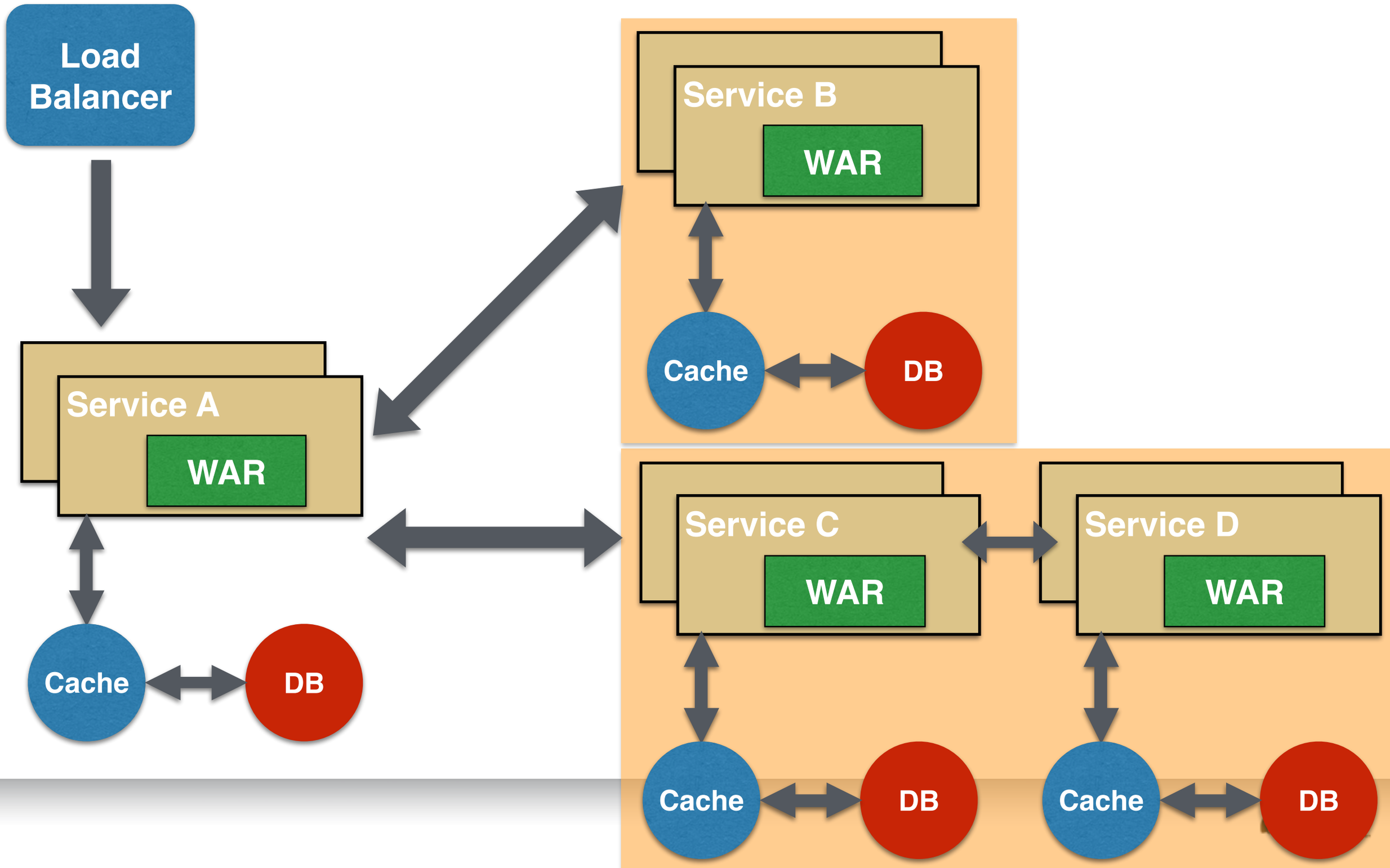
# Proxy Pattern #2



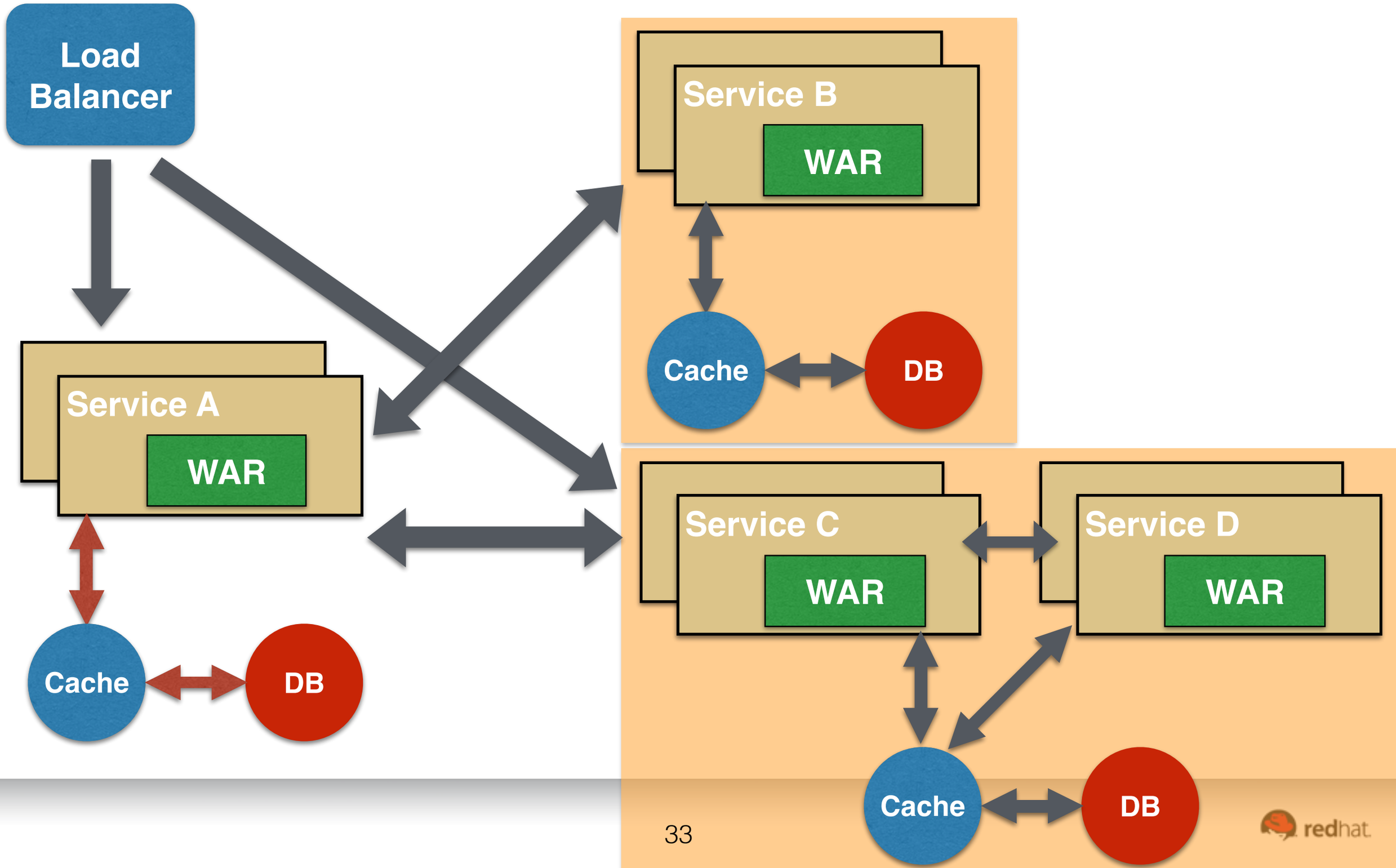
# Chained Pattern #3



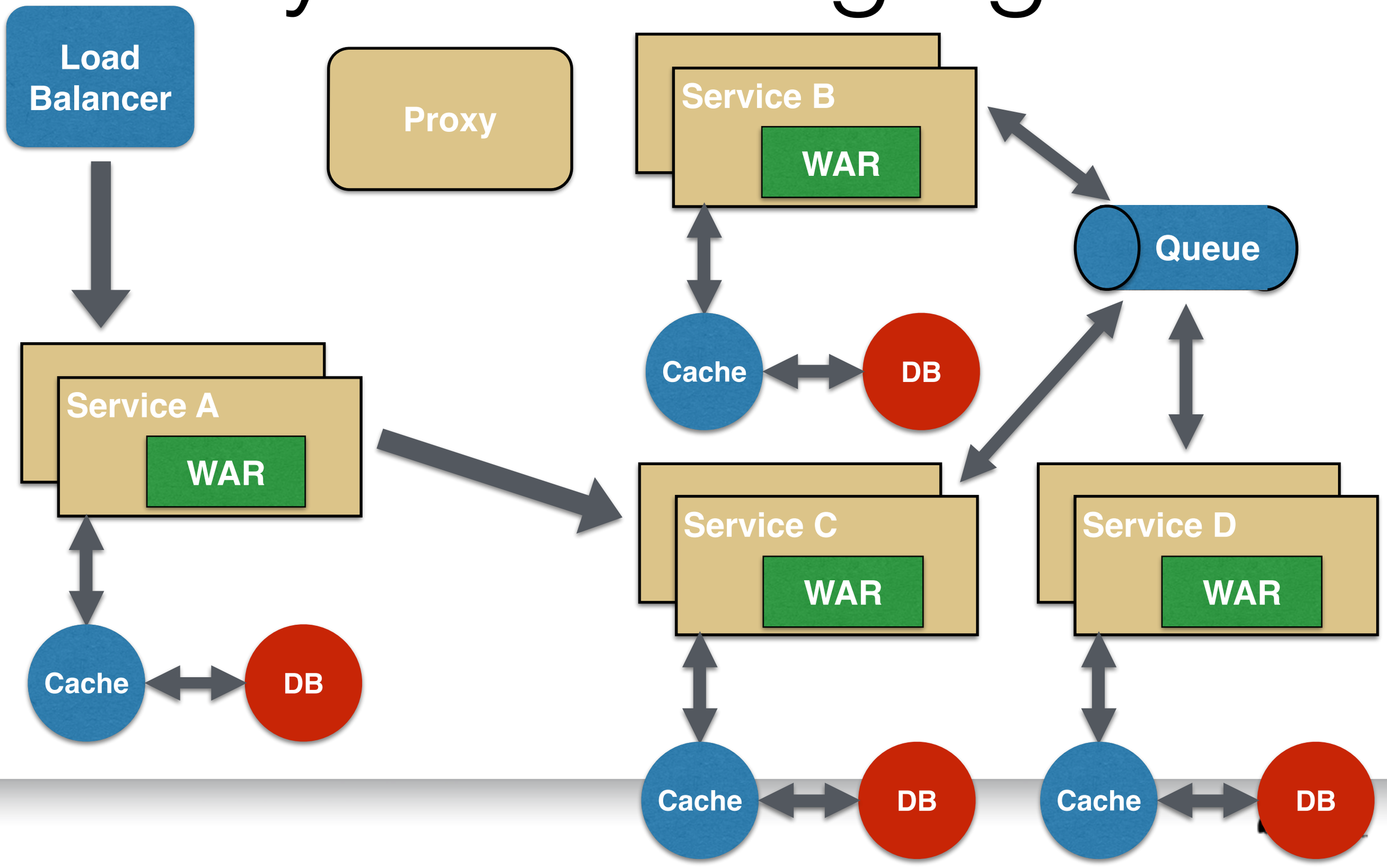
# Branch Pattern #4

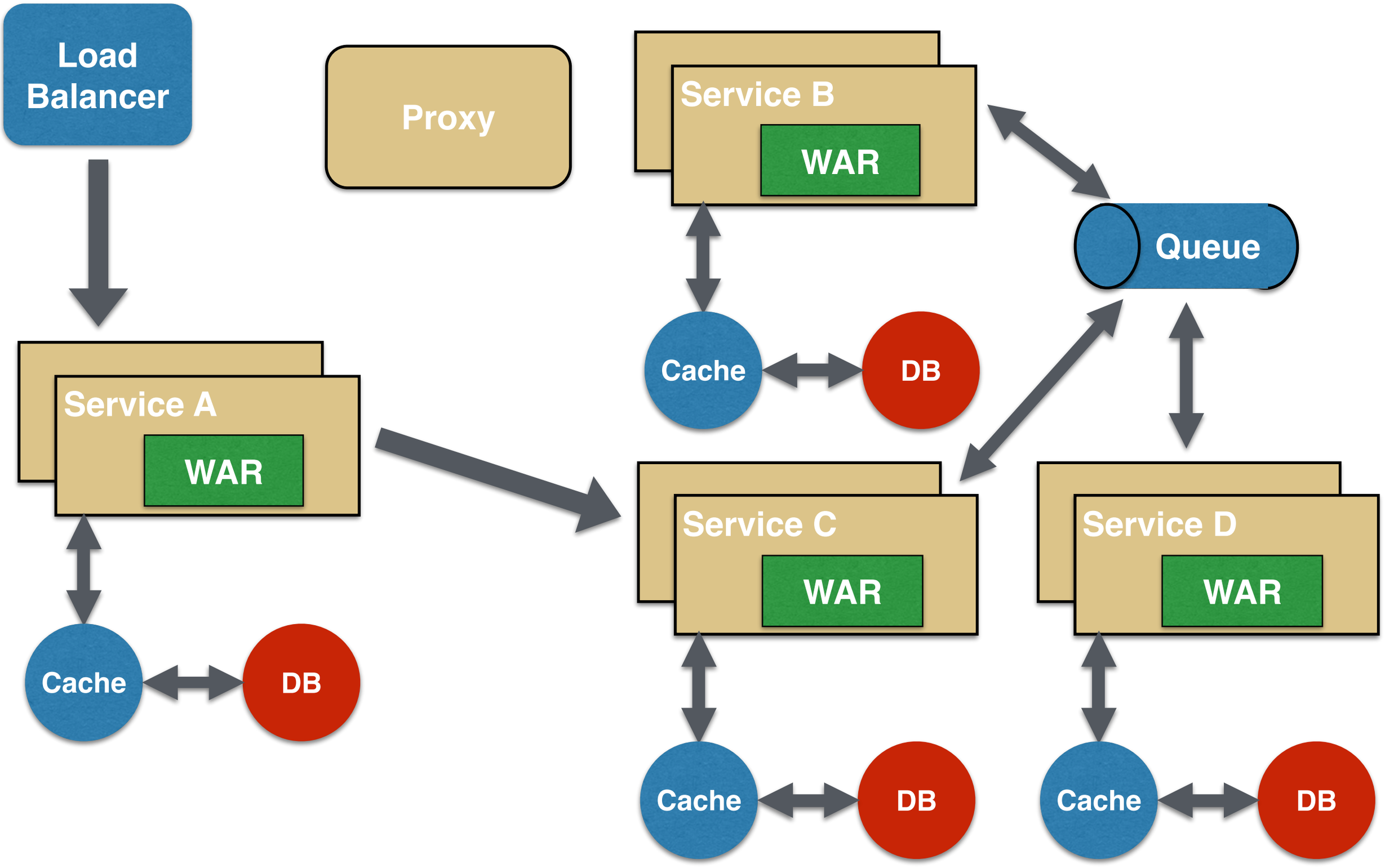


# Shared Resources #5



# Async Messaging #5




Load Balancer
Proxy
Service B
WAR
Queue
Cache
DB
Service A
WAR
Cache
DB
Service C
WAR
Service D
WAR
Cache
DB
Cache
DB

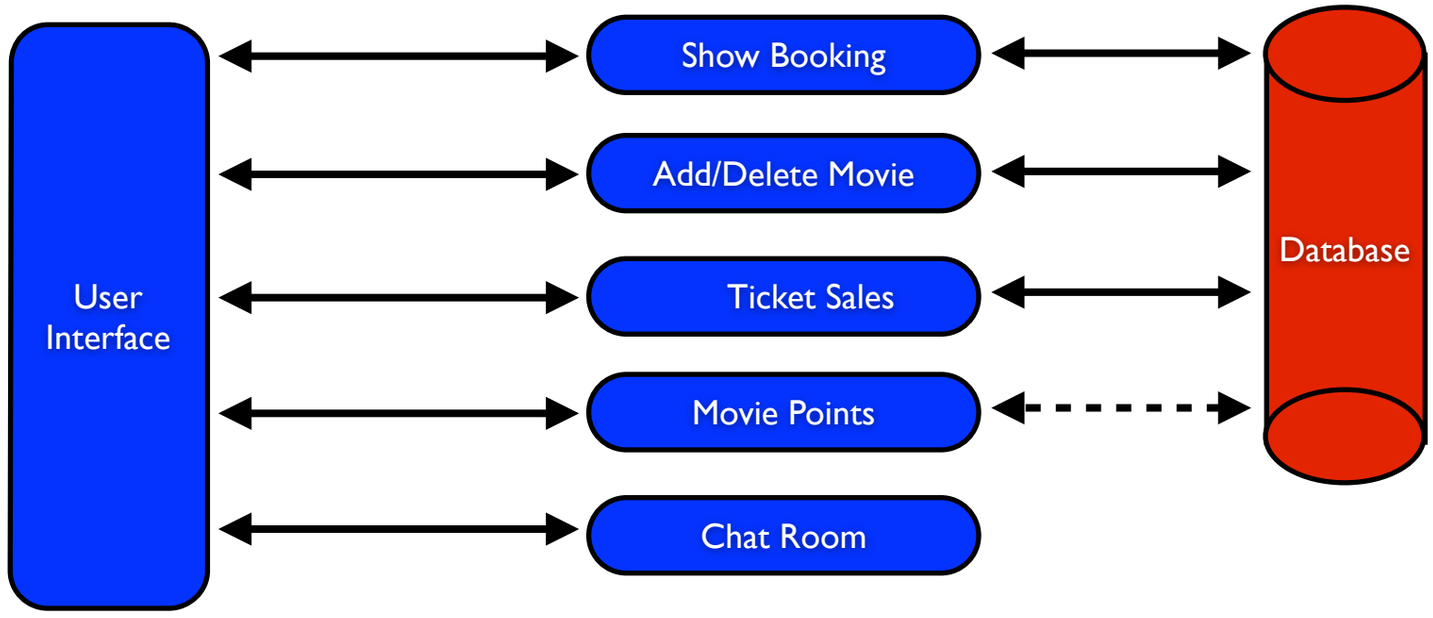

SAY MICROSERVICE
ONE MORE TIME
memegenerator.net

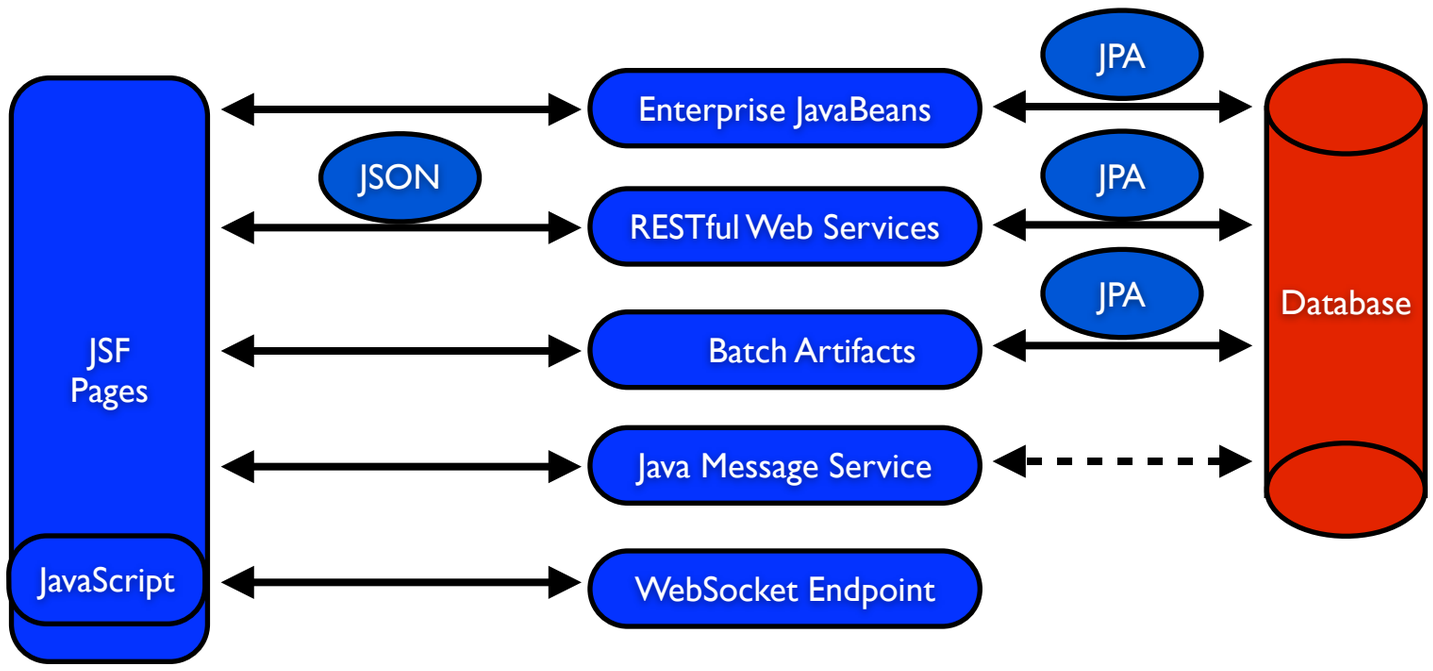
# Advantages of microservices

- 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

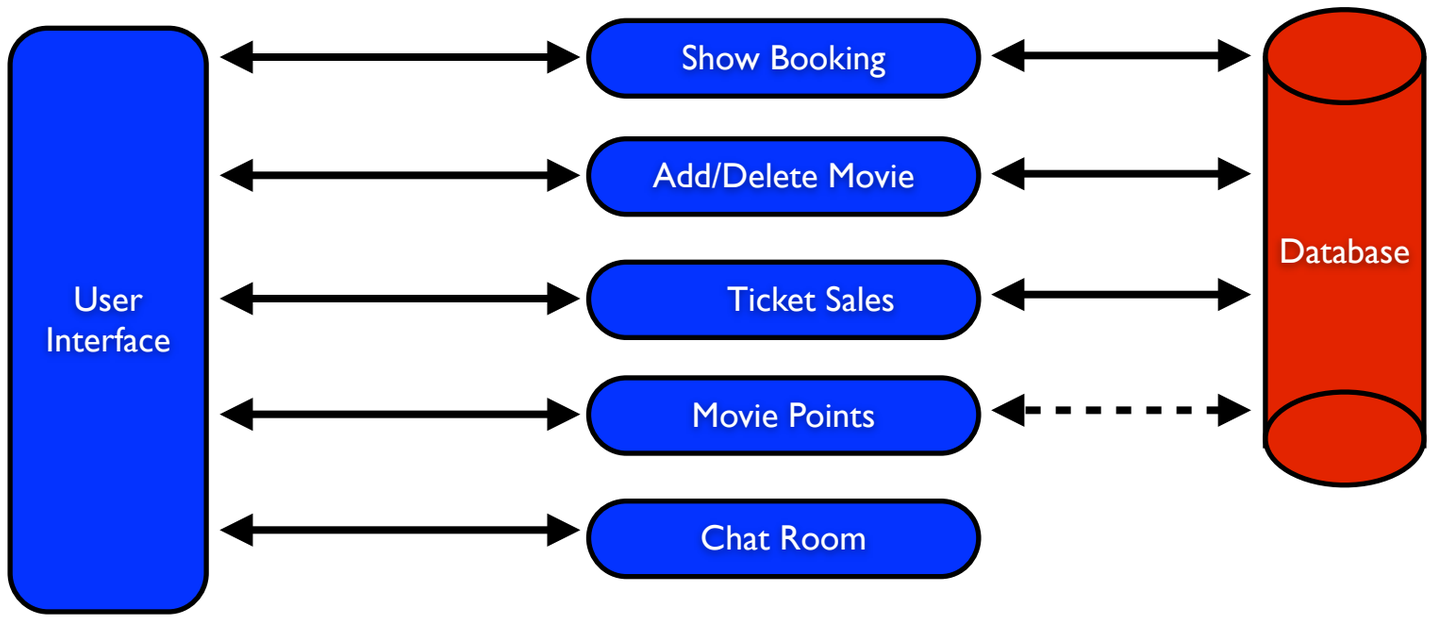
***“If you can't build a [well-structured] monolith, what makes you think microservices are the answer?”***

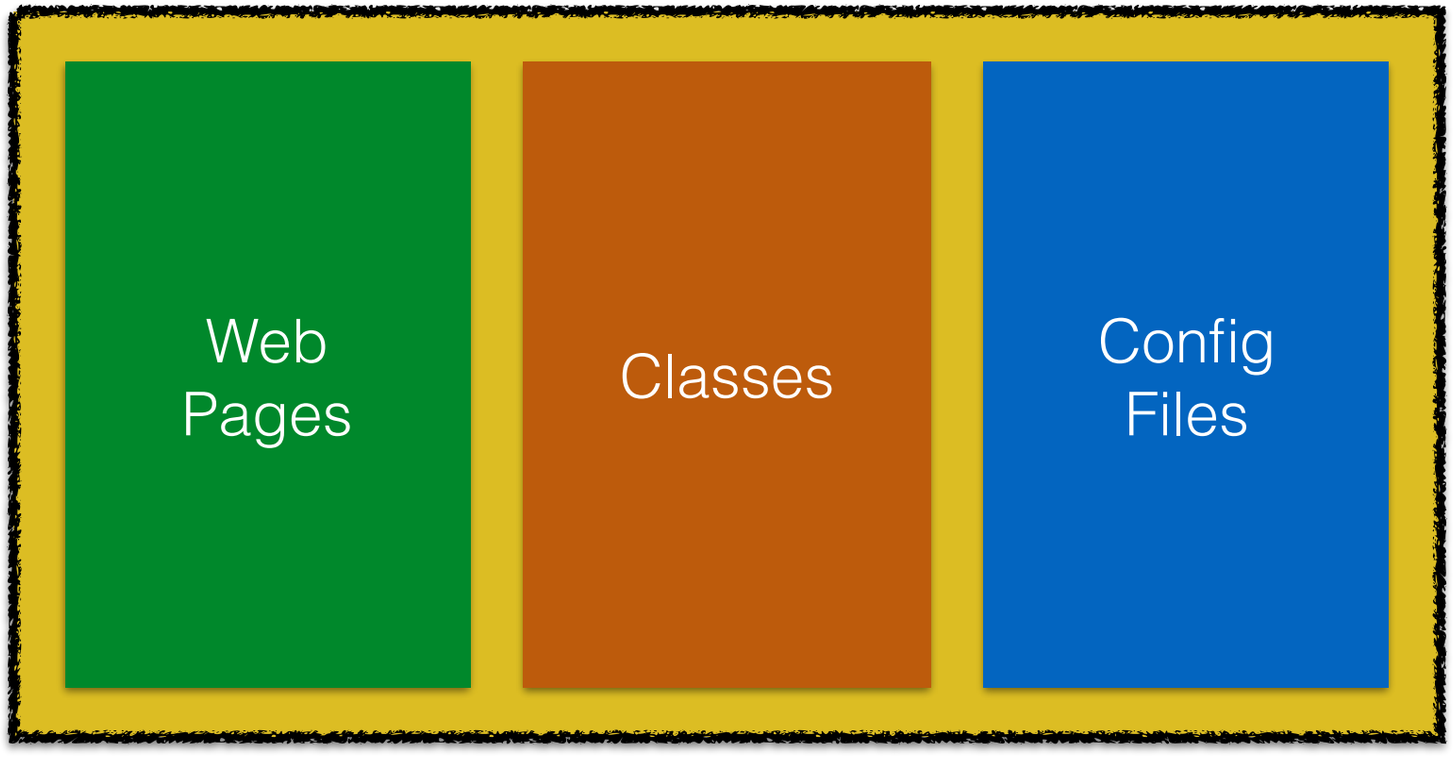
http://www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html


User Interface
Show Booking
Add/Delete Movie
Ticket Sales
Movie Points
Chat Room
Database

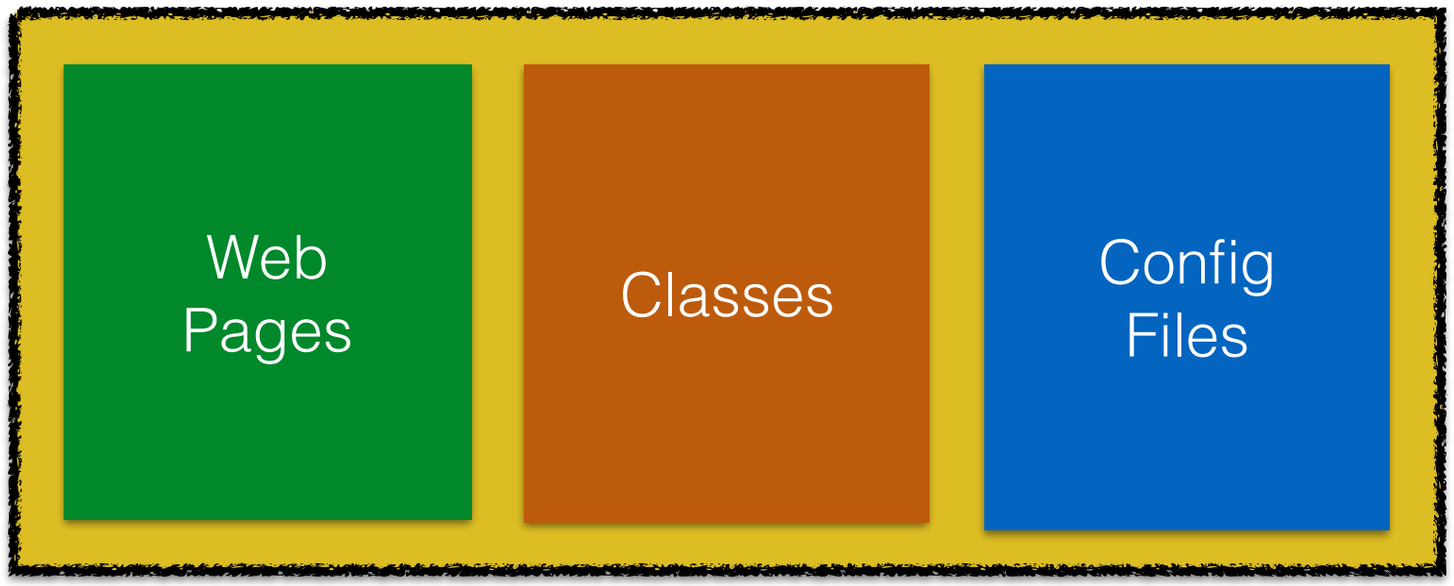

JSF Pages
JavaScript
JSON
Enterprise JavaBeans
RESTful Web Services
Batch Artifacts
Java Message Service
WebSocket Endpoint
JPA
JPA
JPA
Database

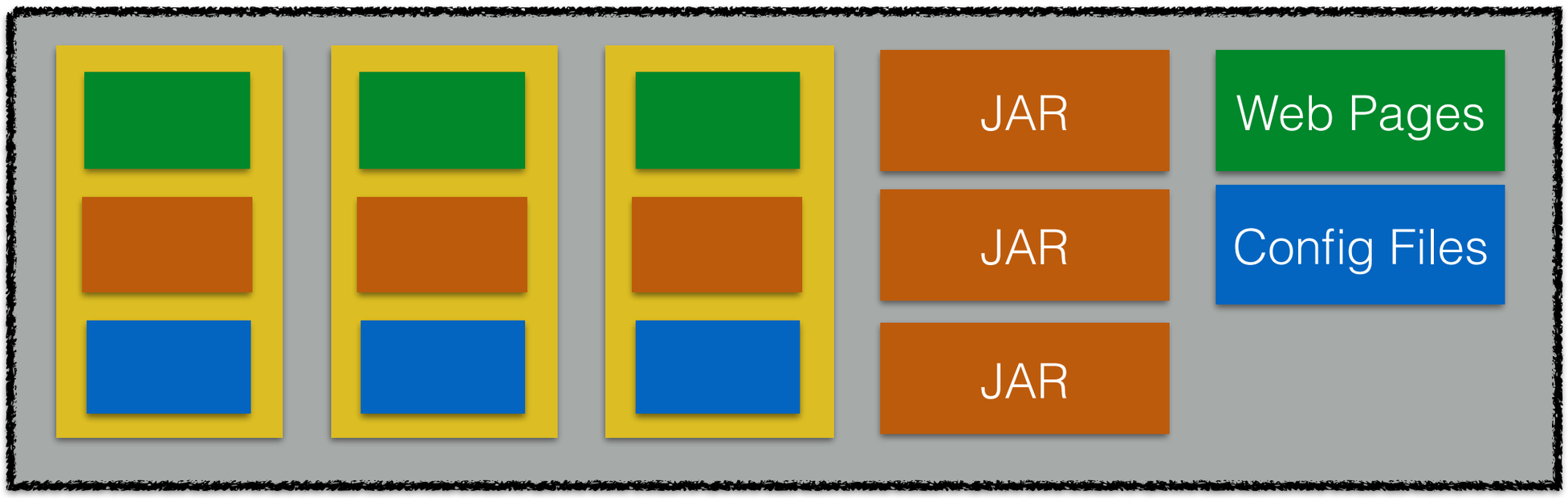
1

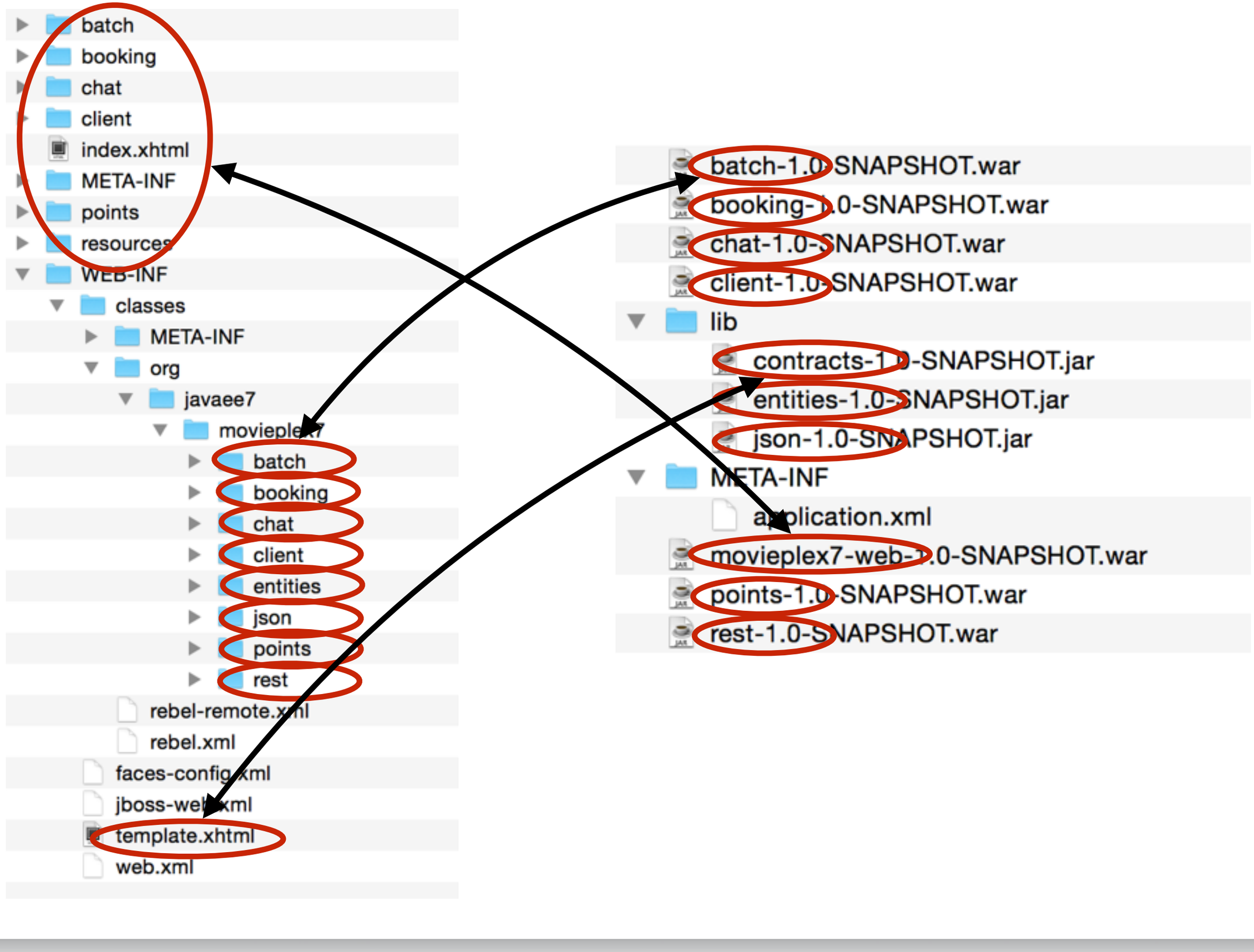

User Interface
Show Booking
Add/Delete Movie
Ticket Sales
Movie Points
Chat Room
Database

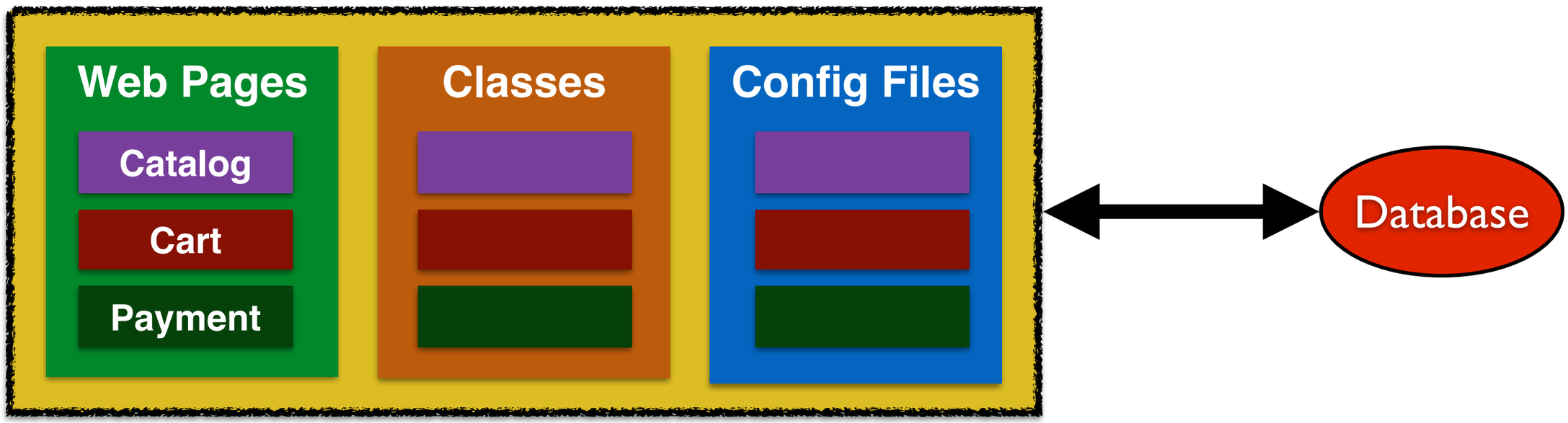

Web Pages
Classes
Config Files

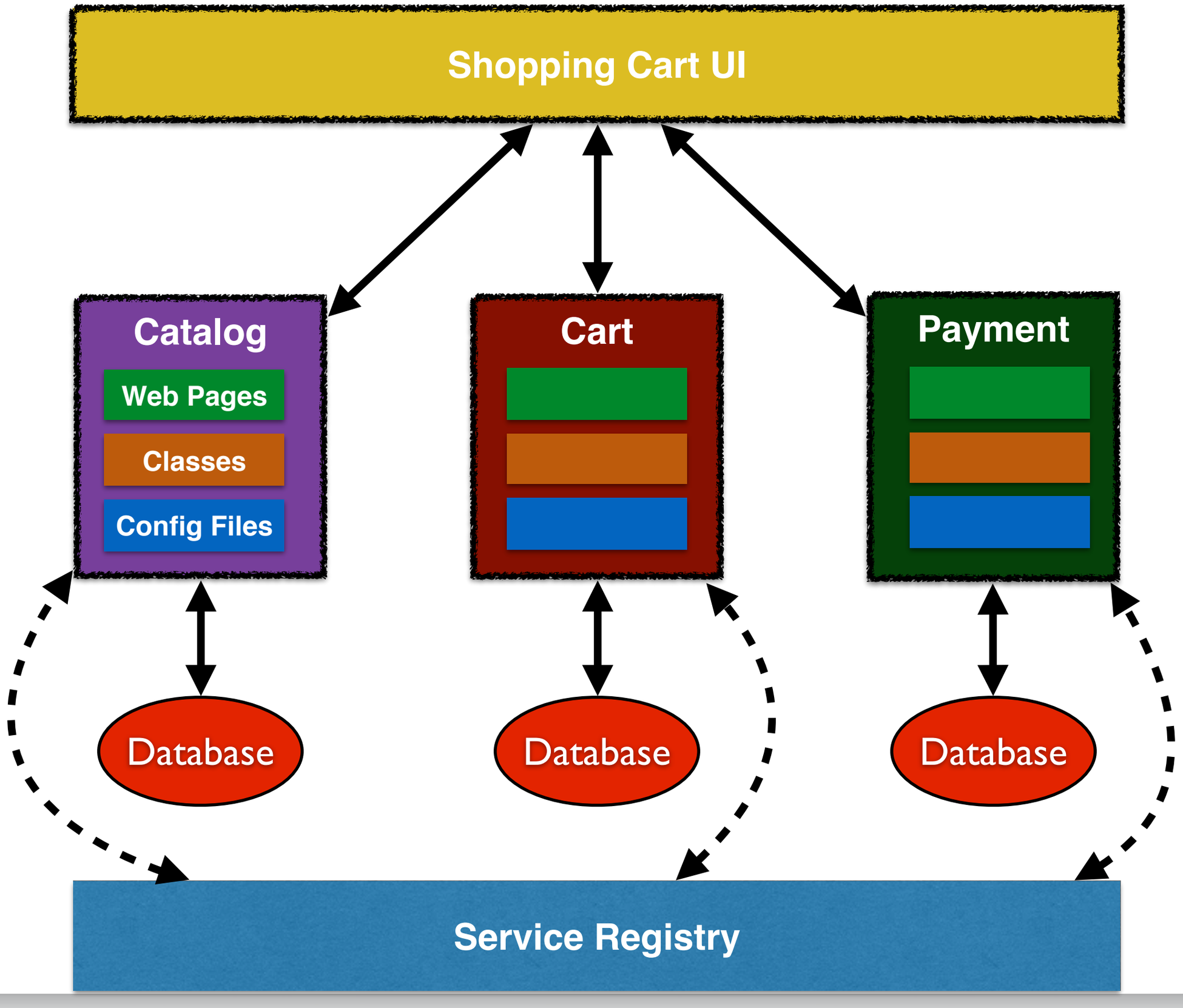
# 2 WAR → EAR

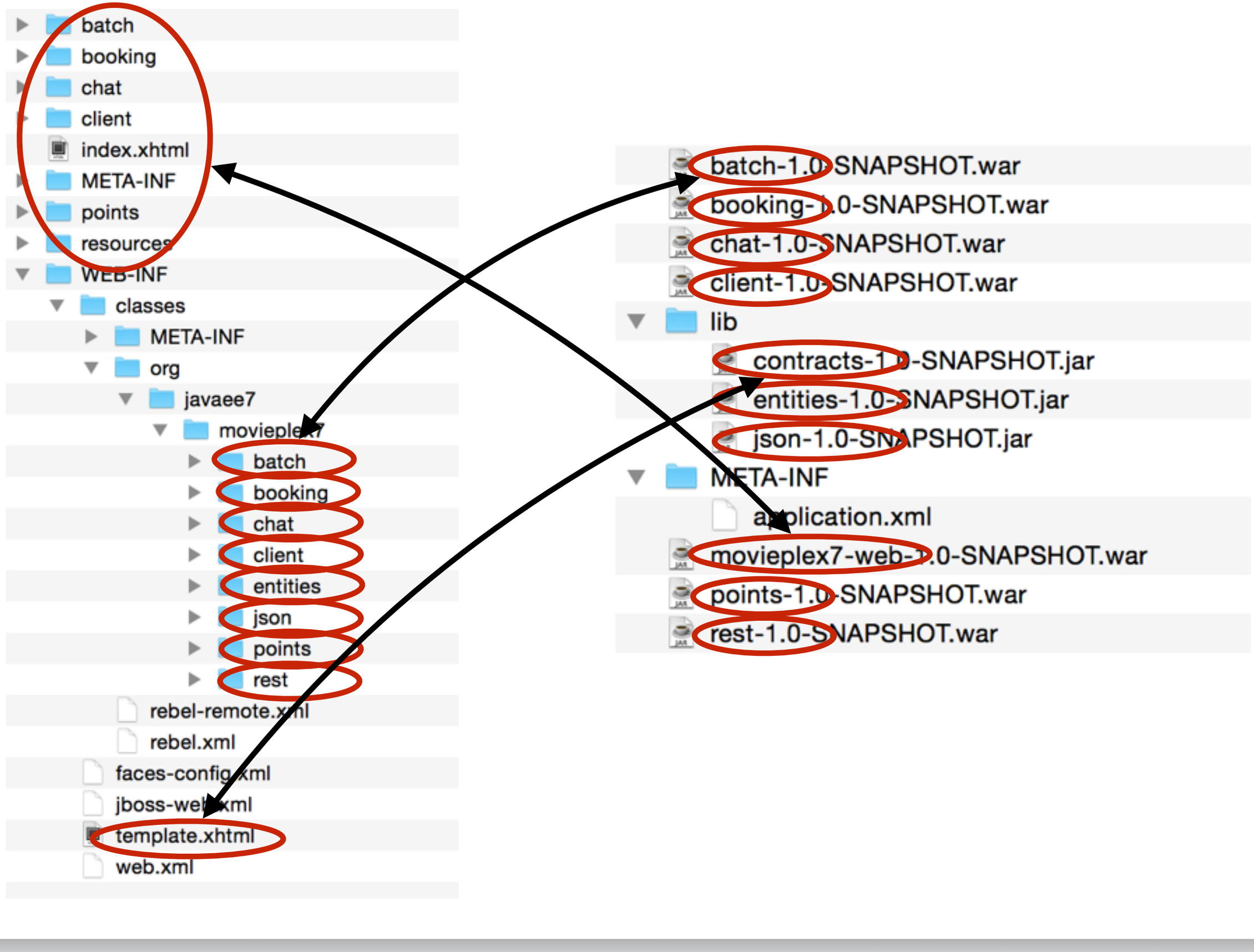

Web
Pages
Classes
Config
Files


JAR
JAR
JAR
Web Pages
Config Files

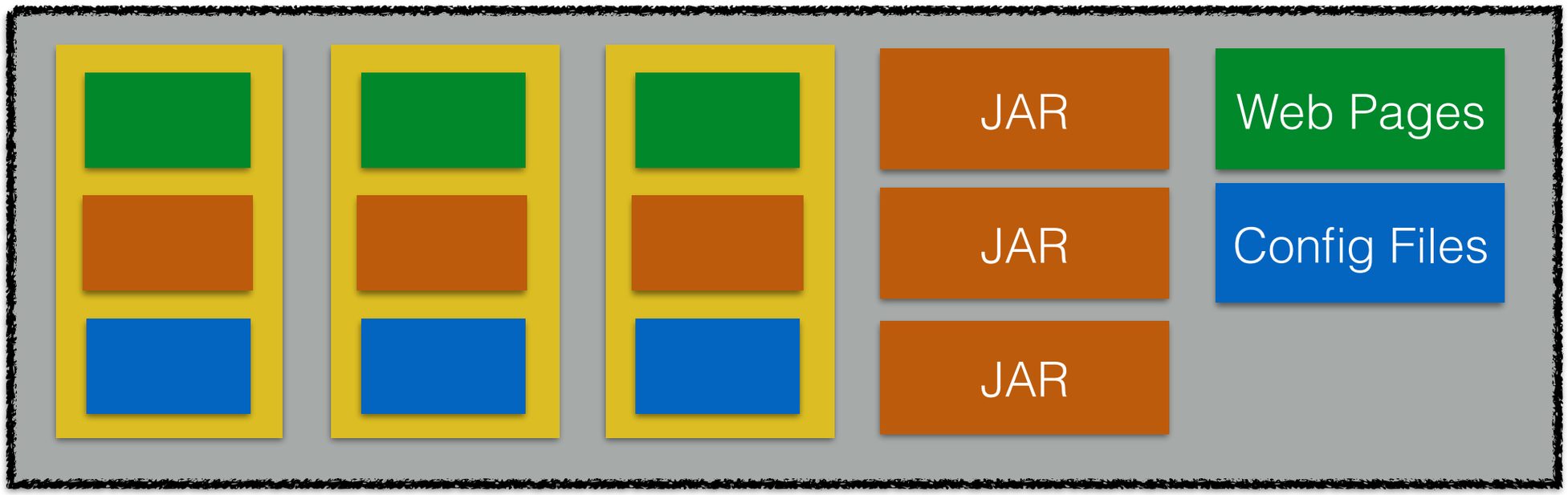

batch
booking
chat
client
index.xhtml
META-INF
points
resources
WEB-INF
classes
META-INF
org
javaee7
batch
booking
chat
client
entities
json
points
rest
rebel-remote.xml
rebel.xml
faces-config.xml
jboss-web.xml
template.xhtml
web.xml
batch-1.0-SNAPSHOT.war
booking-1.0-SNAPSHOT.war
chat-1.0-SNAPSHOT.war
client-1.0-SNAPSHOT.war
lib
contracts-1.0-SNAPSHOT.jar
entities-1.0-SNAPSHOT.jar
json-1.0-SNAPSHOT.jar
META-INF
application.xml
movieplex7-web-1.0-SNAPSHOT.war
points-1.0-SNAPSHOT.war
rest-1.0-SNAPSHOT.war

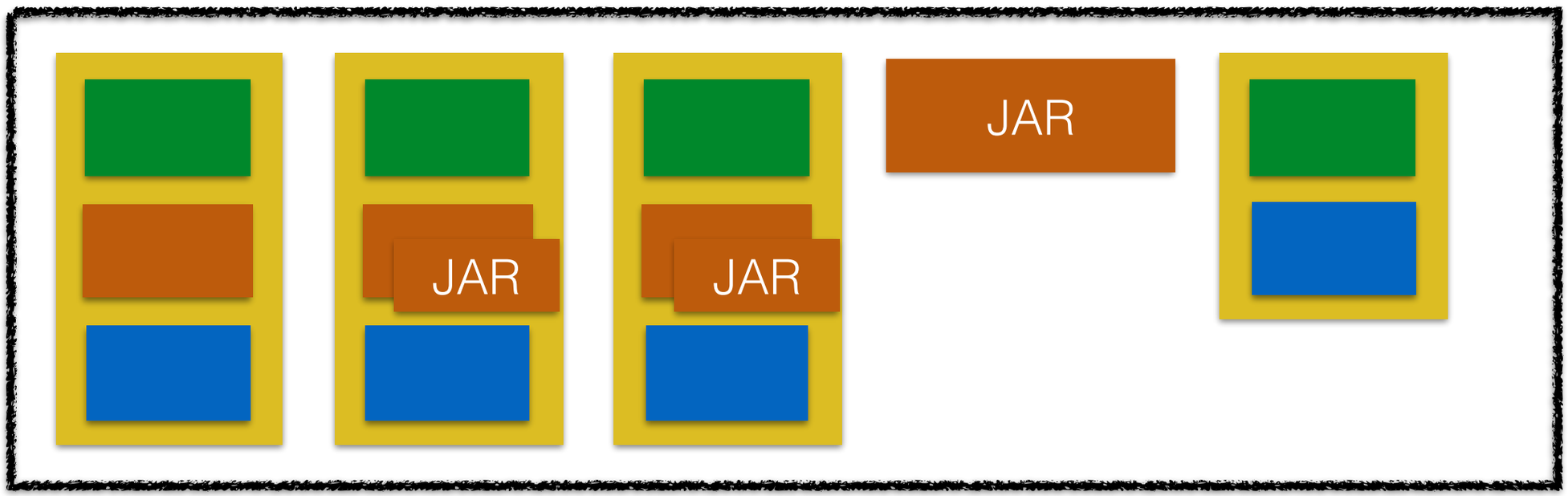

Web Pages
Catalog
Cart
Payment
Classes
Config Files
Database


Shopping Cart UI
Catalog
Web Pages
Classes
Config Files
Cart
Payment
Database
Database
Database
Service Registry

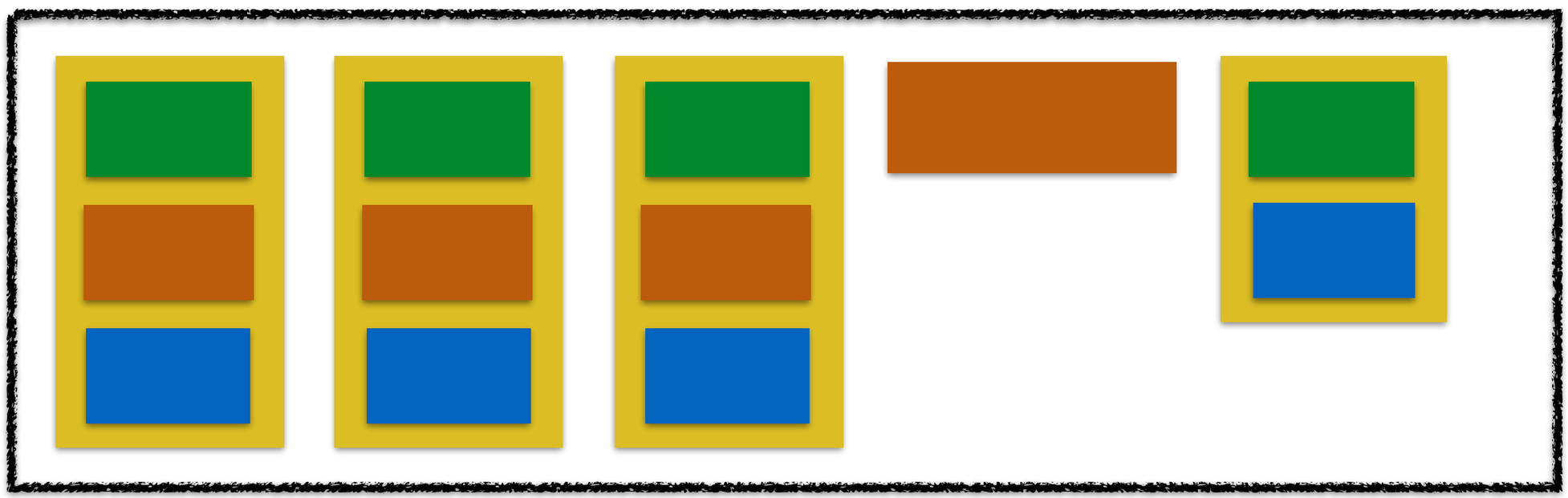

batch
booking
chat
client
index.xhtml
META-INF
points
resources
WEB-INF
classes
META-INF
org
javaee7
batch
booking
chat
client
entities
json
points
rest
rebel-remote.xml
rebel.xml
faces-config.xml
jboss-web.xml
template.xhtml
web.xml
batch-1.0-SNAPSHOT.war
booking-1.0-SNAPSHOT.war
chat-1.0-SNAPSHOT.war
client-1.0-SNAPSHOT.war
lib
contracts-1.0-SNAPSHOT.jar
entities-1.0-SNAPSHOT.jar
json-1.0-SNAPSHOT.jar
META-INF
application.xml
movieplex7-web-1.0-SNAPSHOT.war
points-1.0-SNAPSHOT.war
rest-1.0-SNAPSHOT.war

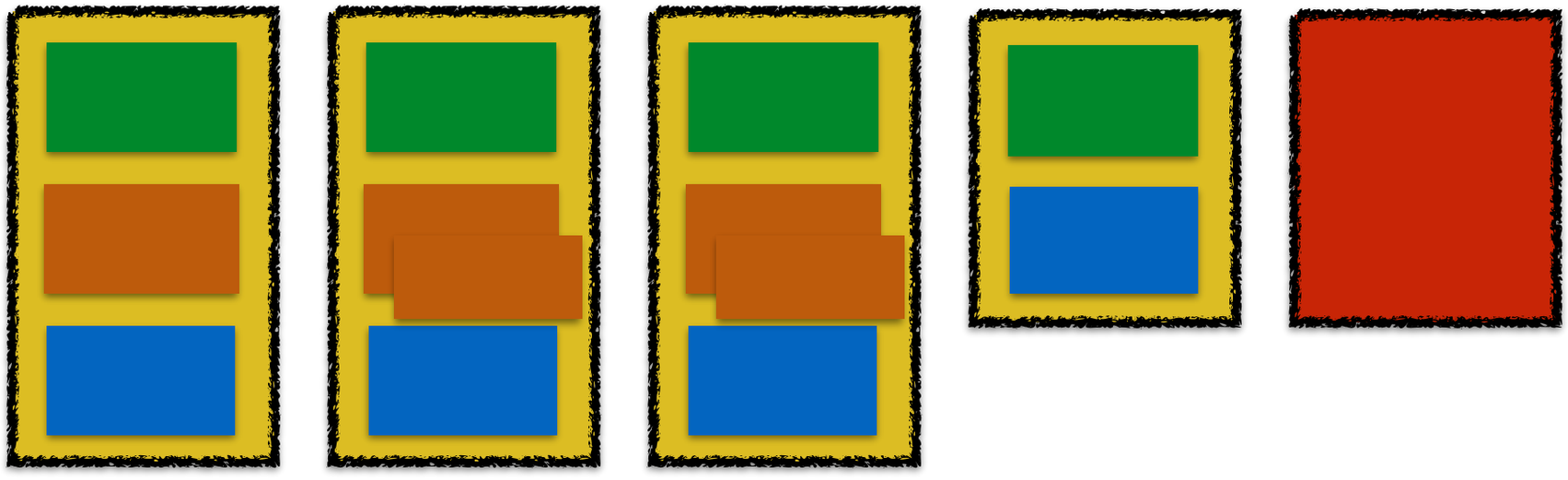
# 3 EAR → WAR



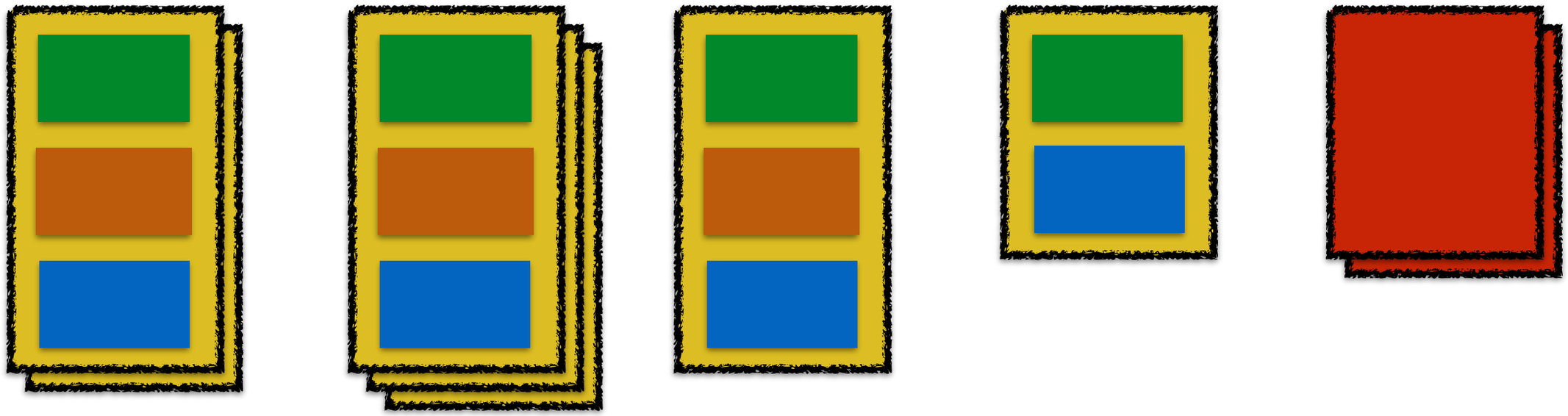



# 4 WAR in Container





# 5 WAR: Scale on X-axis or Y-Axis

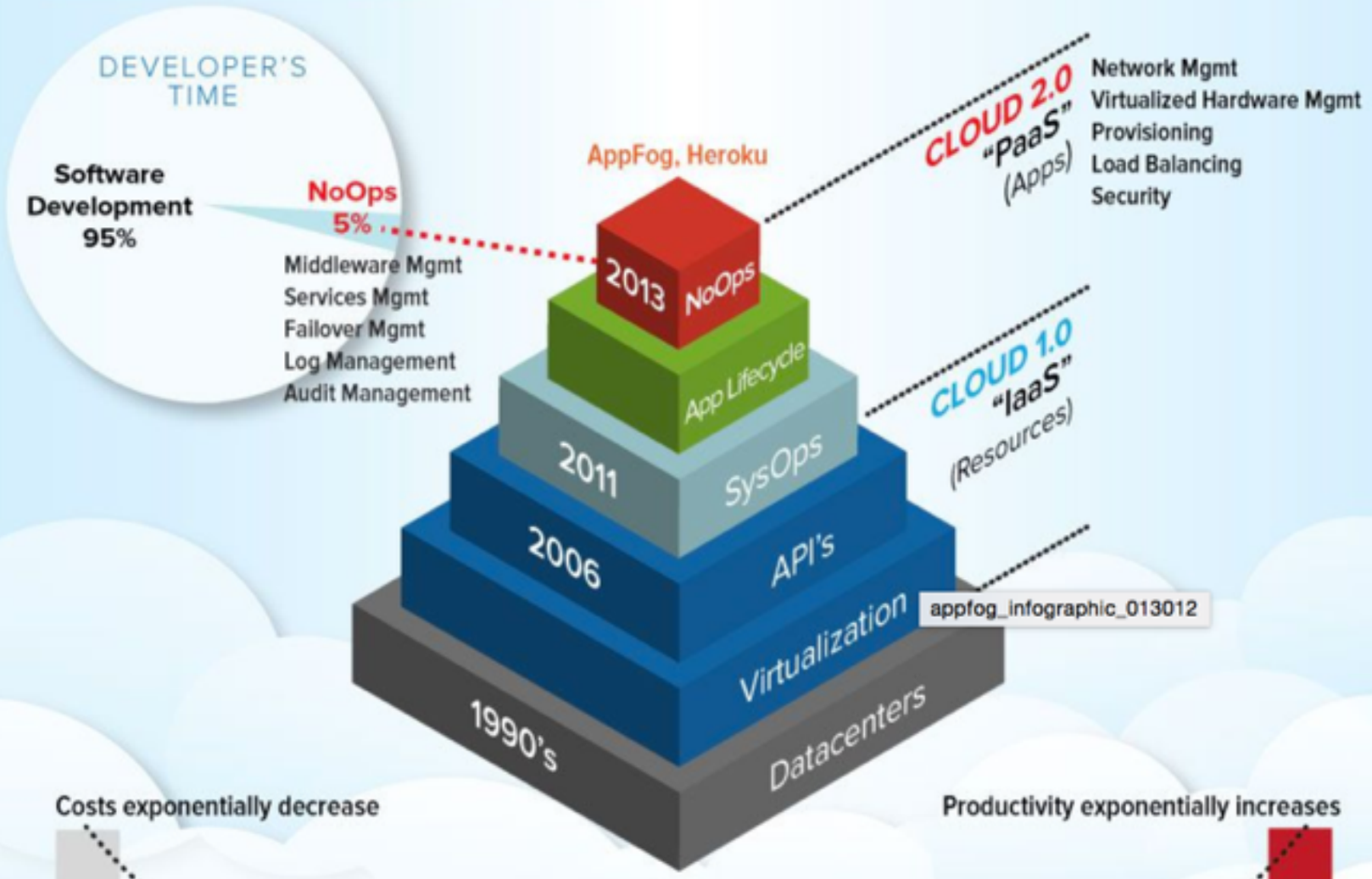


# NoOps

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

## 2013: A bright NoOps future

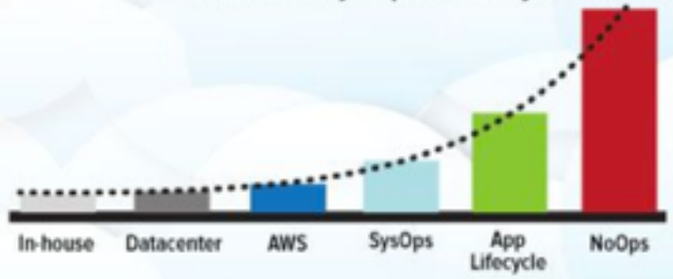
So where does this all lead? The end-game is NoOps. Where building and running an app is purely a developer process — and where developers are not having to spend time doing Ops work.




### Costs exponentially decrease


Sources: National Venture Capital Association and Center for Venture Research

### Productivity exponentially increases




Sources: National Venture Capital Association and Center for Venture Research

NoOps is the future. But it's a very near future. NoOps, typified by managed Platform-as-a-Service offerings, is the fully realized promise of the cloud. It has the most capex cost reduction of any cloud technology and combines it with the highest amount of productivity, velocity, and development agility.

NoOps takes all of the exponential gains from each layer of this Cloud 2.0 and passing them along to developers as one, rolled-up, massive reduction in cost, time and risk.

By freeing developers to focus on what they want to do — and on what they do best — we are enabling them to even more easily start companies and build products. And that is why AppFog is a *Start-up Enabler.*

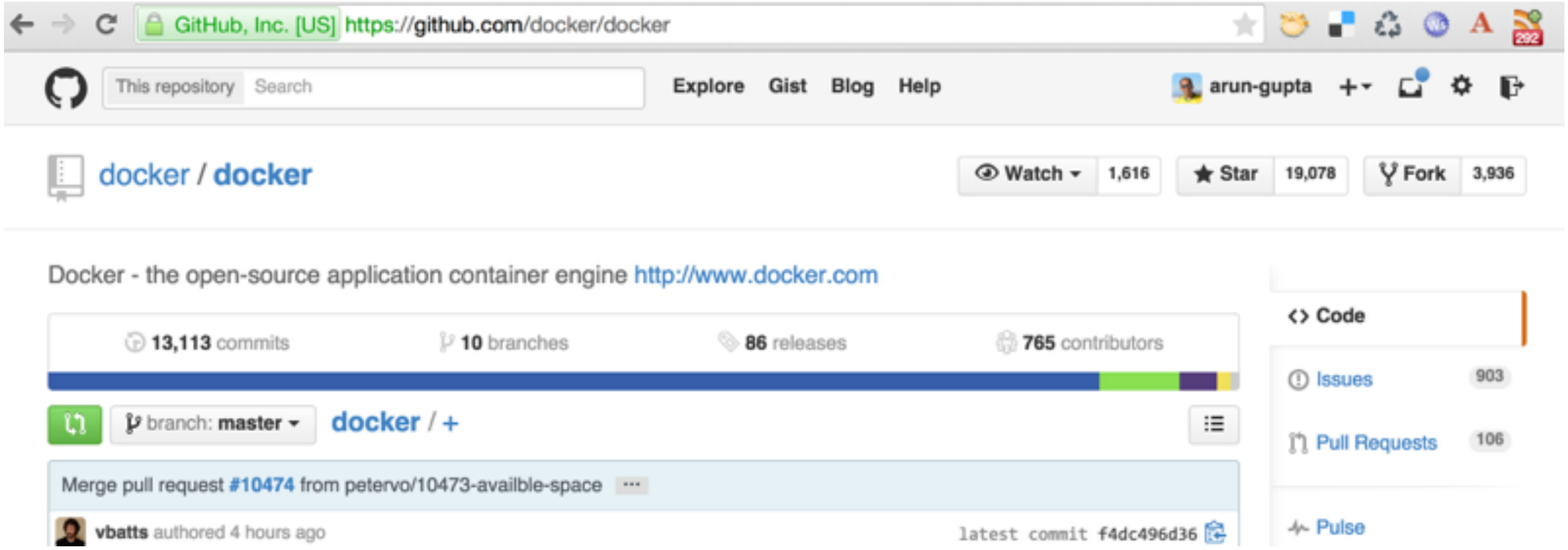
https://gigaom.com/2012/01/31/why-2013-is-the-year-of-noops-for-programmers-infographic/

# 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

# What is Docker?

- Open source project and company




- Used to create containers for software applications

- Package Once Deploy Anywhere (PODA)

# 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

- 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
  - Host-centric solution, not aware of anything
  - No higher-level provisioning
  - No usage tracking/reporting

# Kubernetes


- Open source orchestration system for Docker containers

- Provide declarative primitives for the “desired state”

- Self-healing

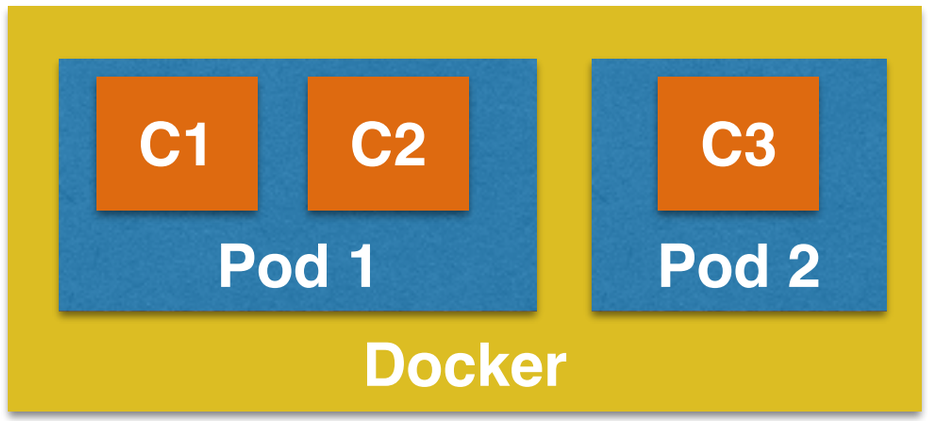
- Auto-restarting

- Schedule across hosts

- Replicating

# Concepts

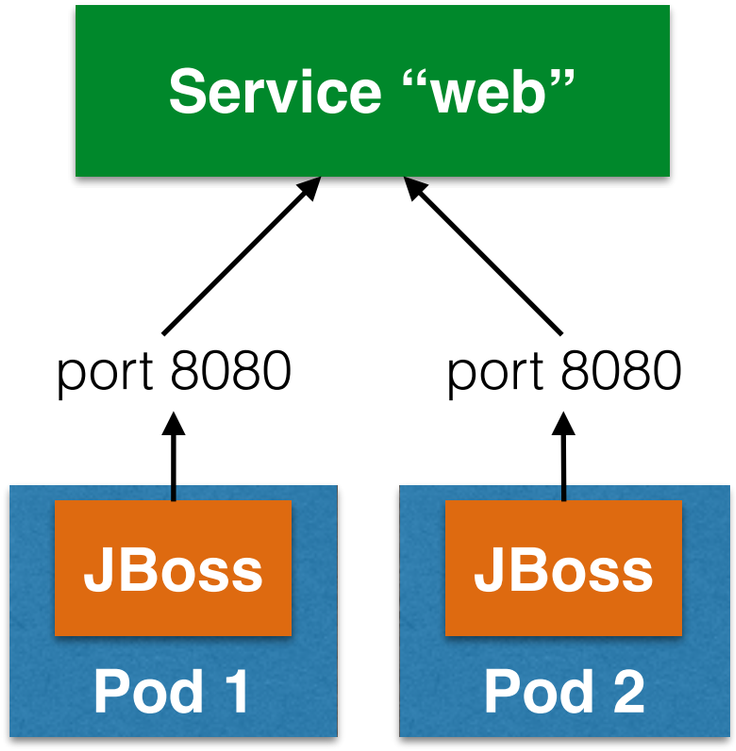
- **Pods**: collocated group of Docker containers that share an IP and storage volume




- **Service**: Single, stable name for a set of pods, also acts as LB

- **Replication Controller**: manages the lifecycle of pods and ensures specified number are running

- **Label**: used to organize and select group of objects



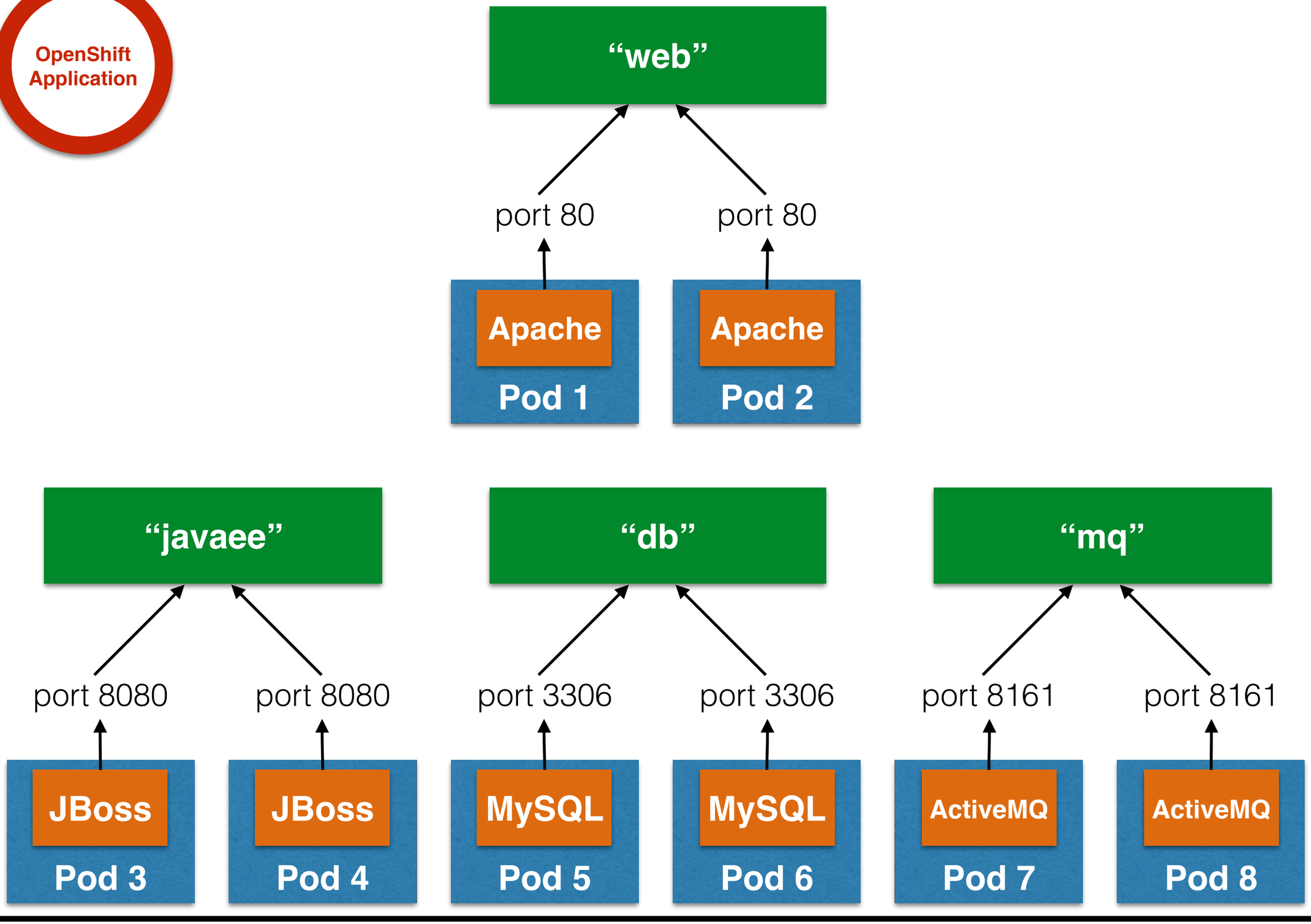
# Kubernetes: Pros and Cons

- 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

OpenShift Application






redhat.


# User Experience


# Containerized Services



# Orchestration


# Container


# Container Host


# jboss.org

JBoss Tools

RED HAT JBOSS DEVELOPER STUDIO

Arquillian

RESTEasy

WildFly

HIBERNATE

VERT.X

FEEDHENRY™

Infinispan

AeroGear

Apache Camel

Qhawtio

fabric8