redhat.

RED HAT® JBOSS®
MIDDLEWARE

Microservice Design Patterns for Java Applications

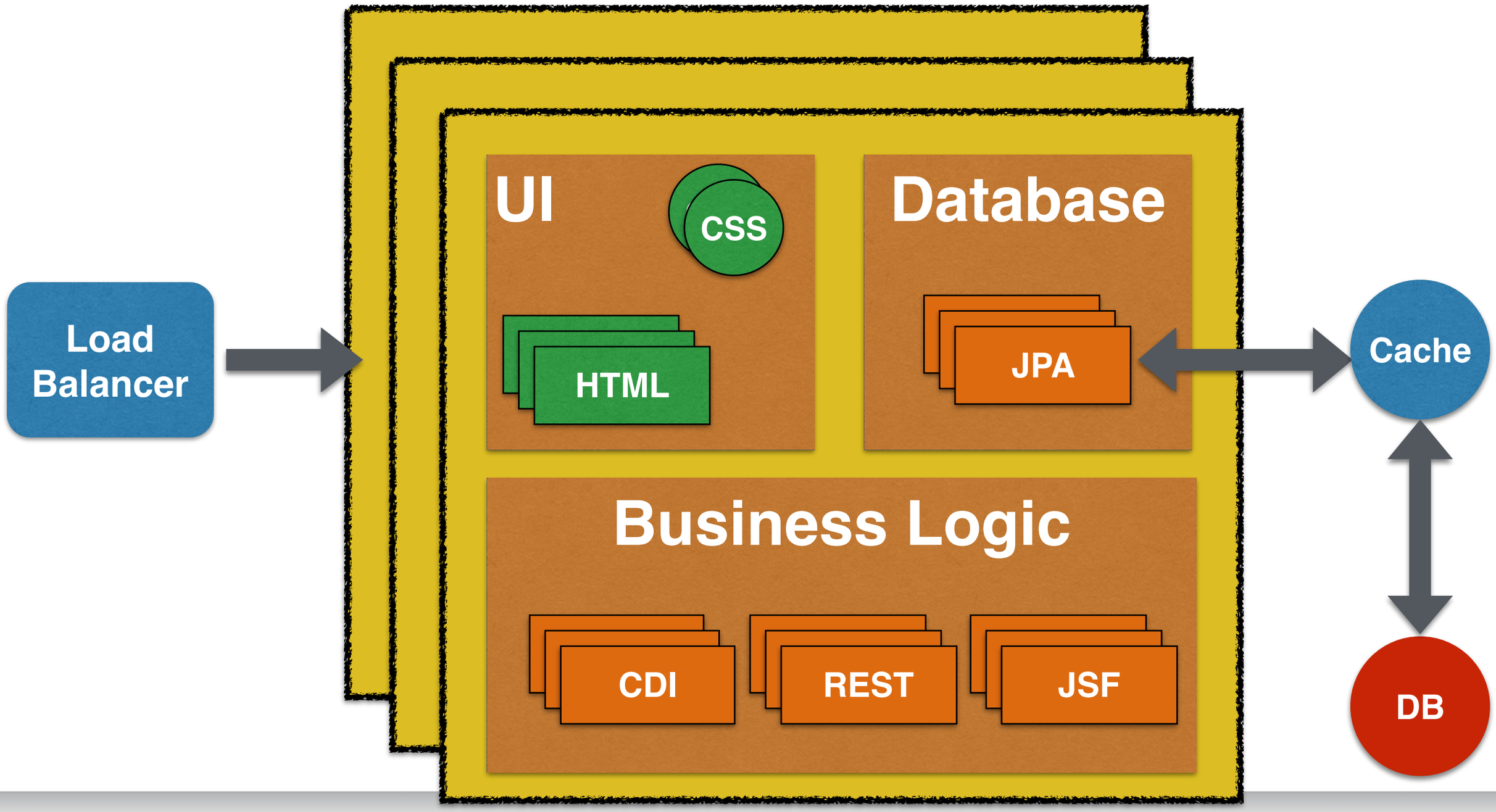
Arun Gupta, Red Hat

Arun Gupta

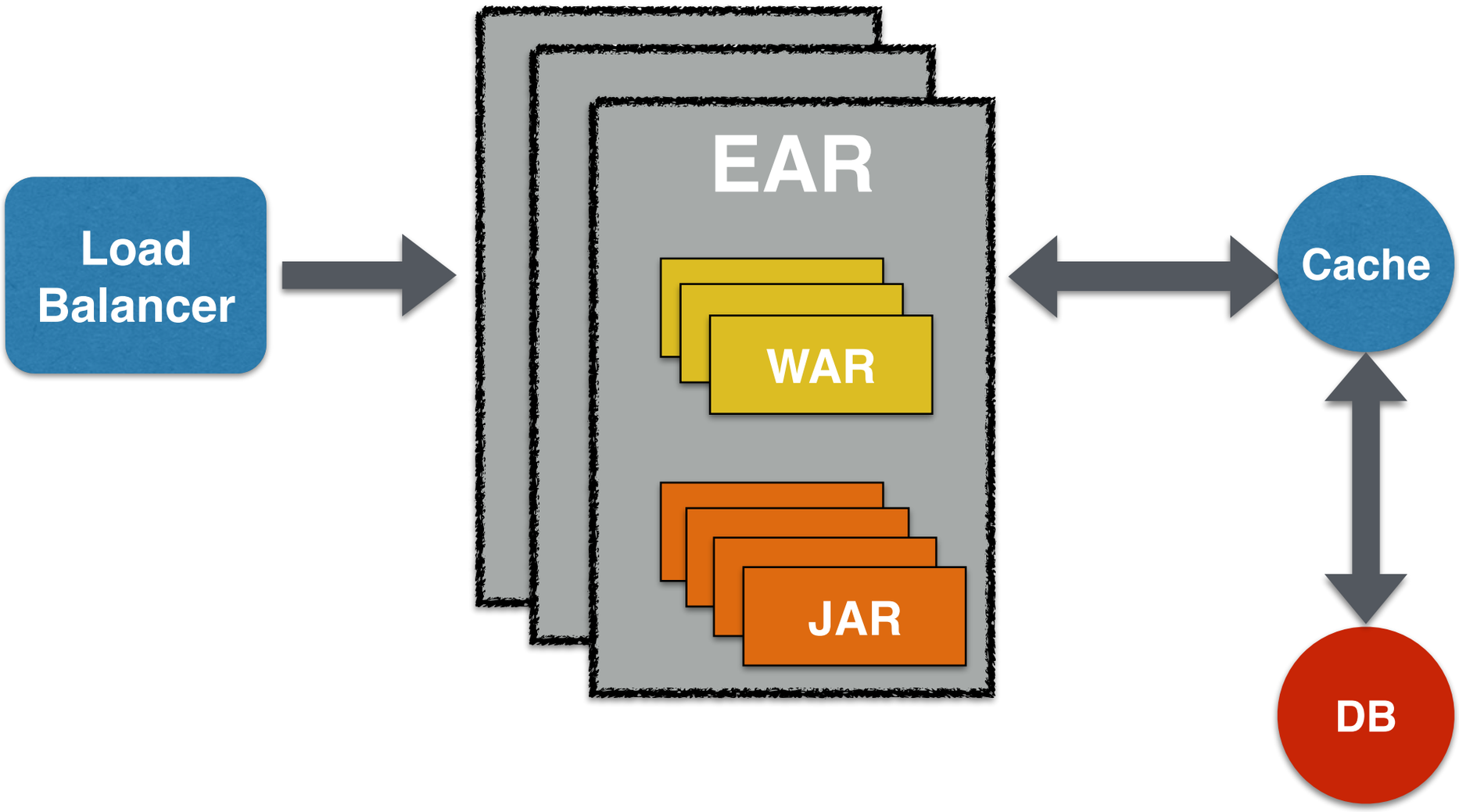
Director, Technical Marketing & Developer Advocacy

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

Monolith Application



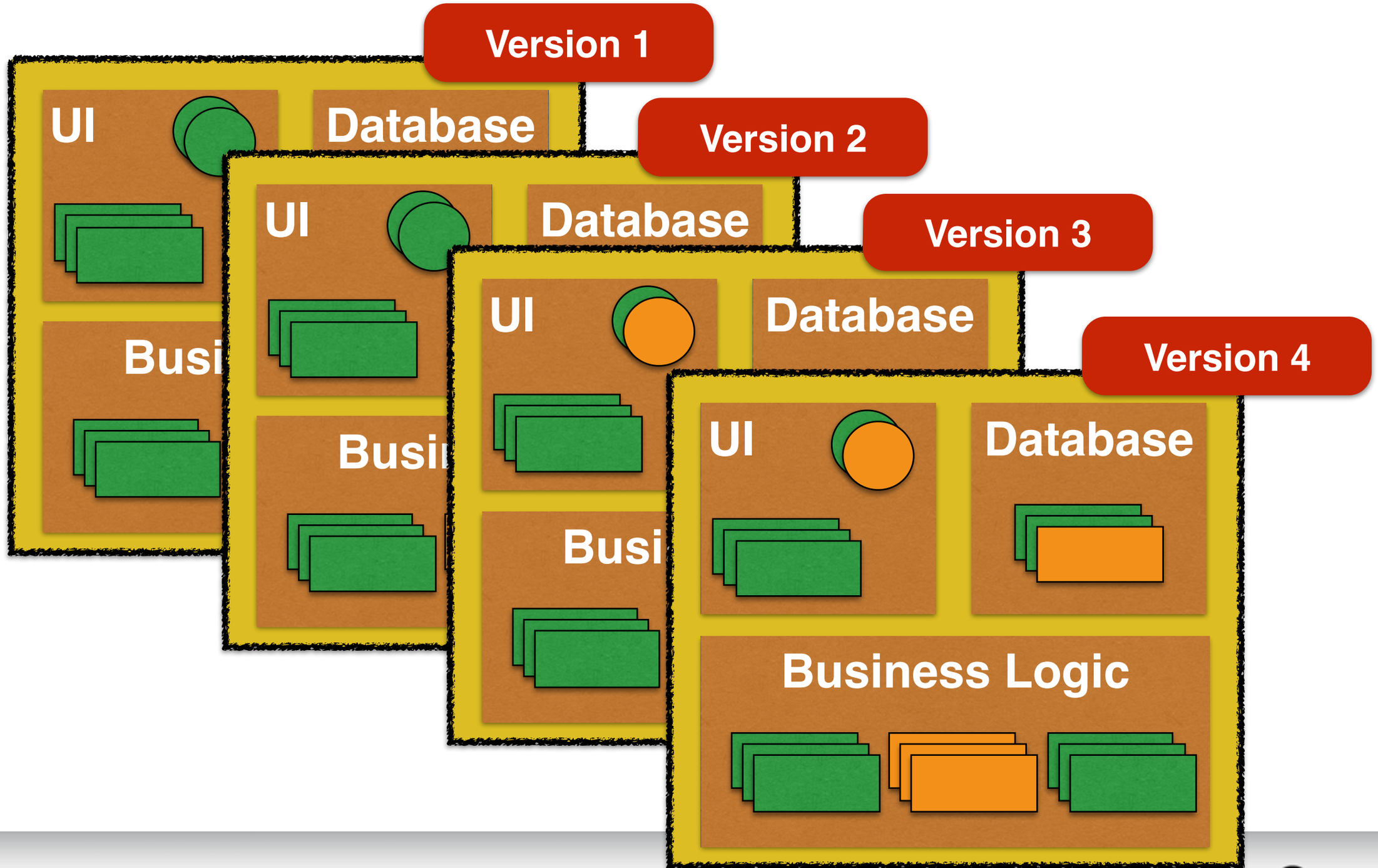
Monolith Application



Advantages of Monolith Application

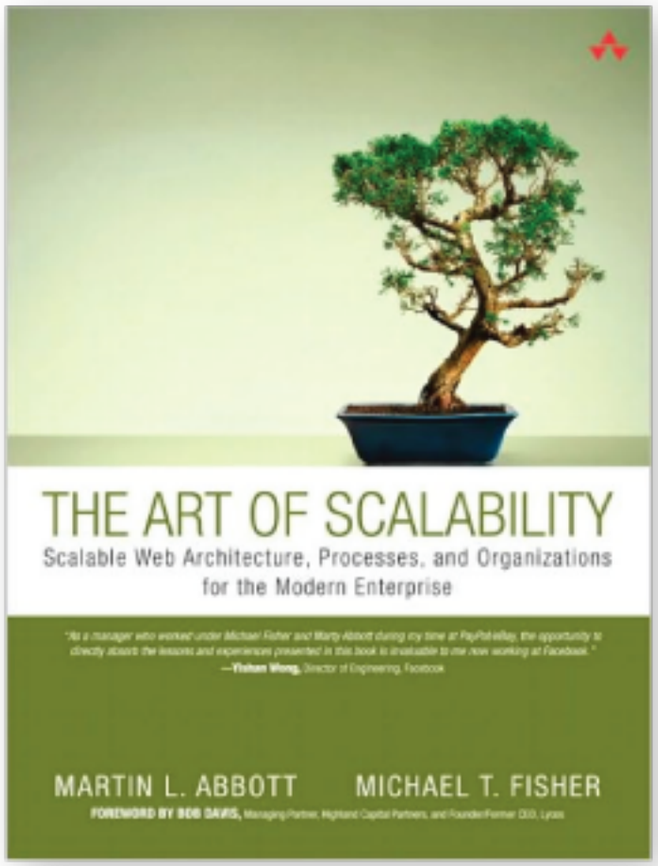
- Typically packaged in a single `.ear`
- Easy to test (all required services are up)
- Simple to develop

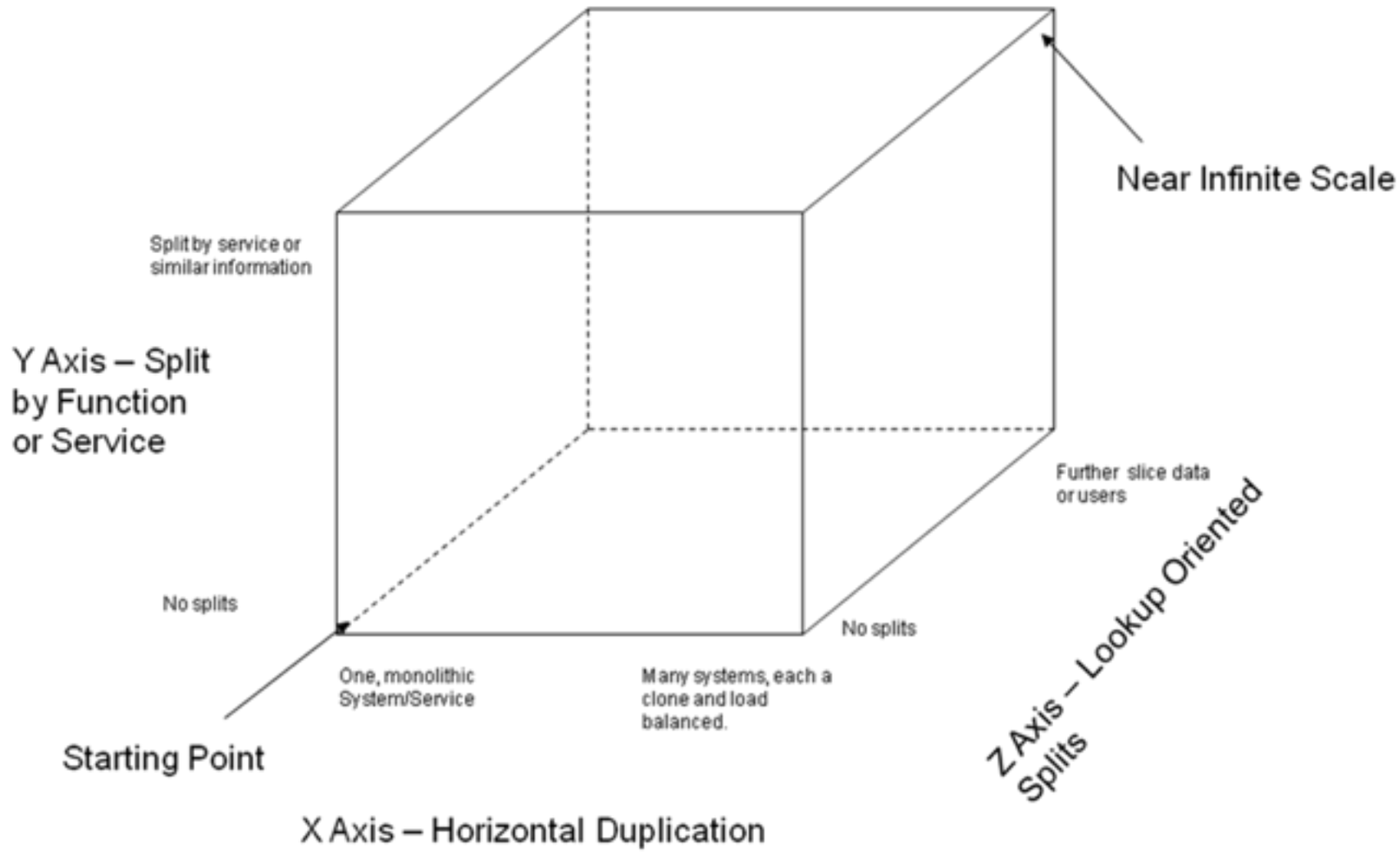
Monolith Application



Disadvantages of Monolith Application

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



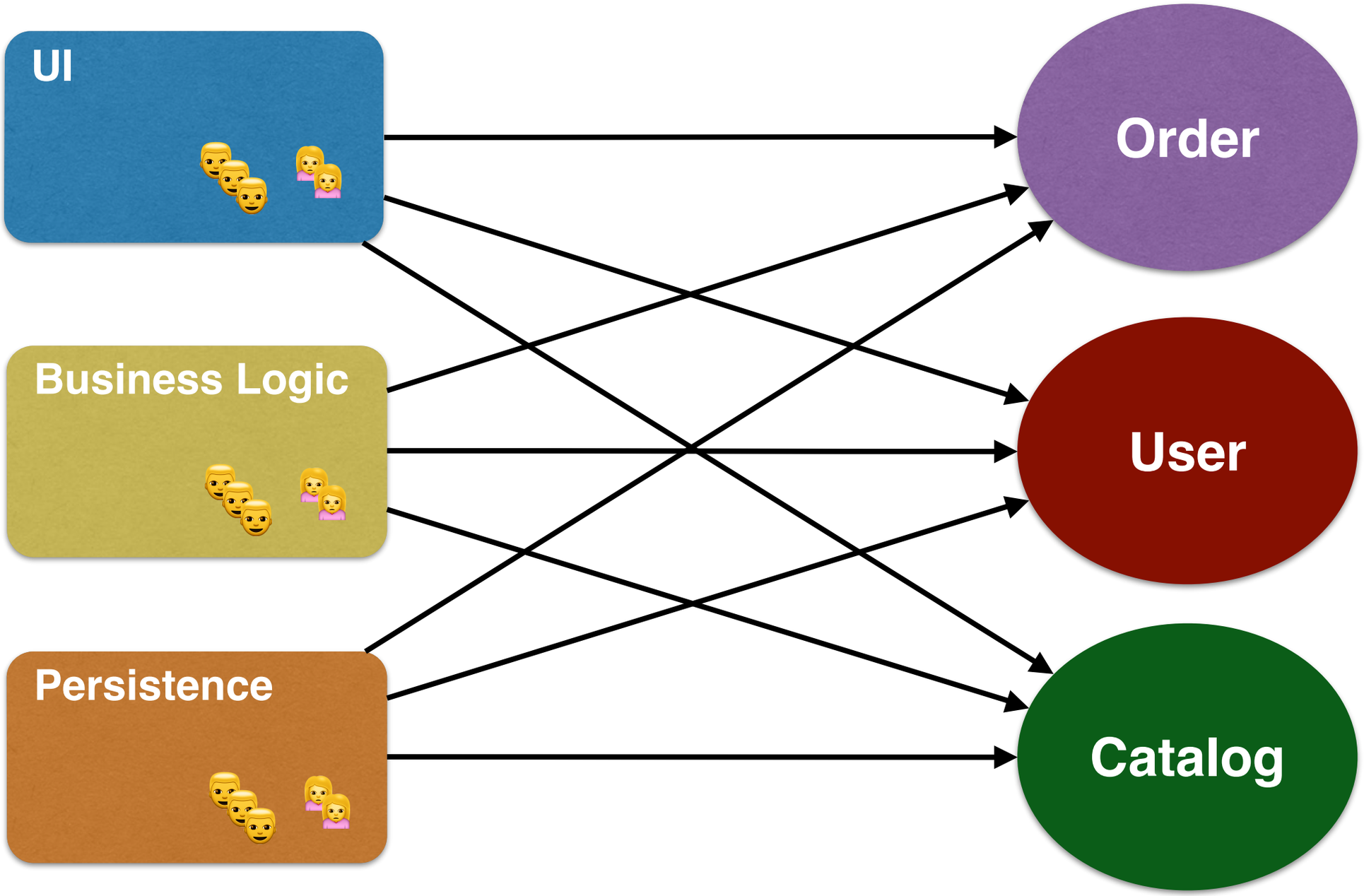


http://akfpartners.com/techblog/2008/05/08/splitting-applications-or-services-for-scale/

*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**.*

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

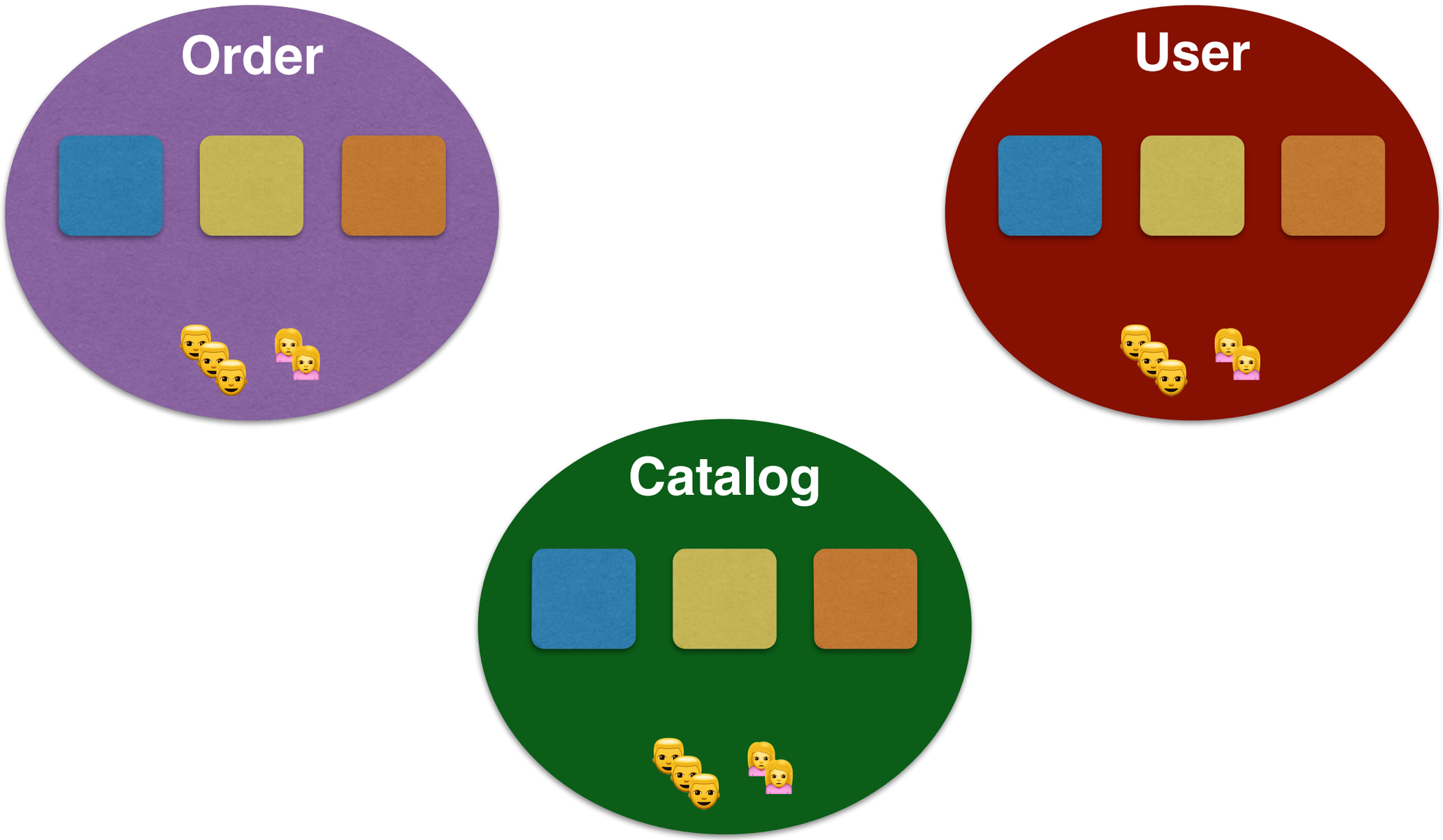

I DONT ALWAYS
BUILD EVERYTHING
BUT WHEN I DO I
USE MICRO SERVICES
imgflip.com




*"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

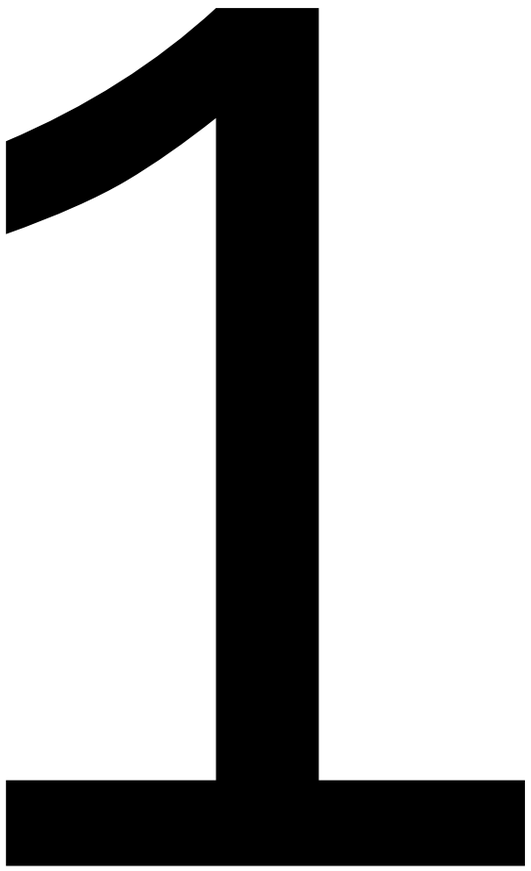
Teams around business capability



Characteristics

Single Responsibility Principle

DO



THING

Characteristics

Explicitly Published interface

Characteristics

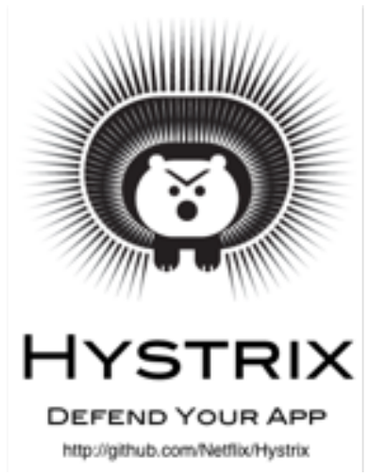
Independently replaceable and upgradeable

Characteristics

With great power, comes great responsibility

“you build it, you run it!”

Designed for failure



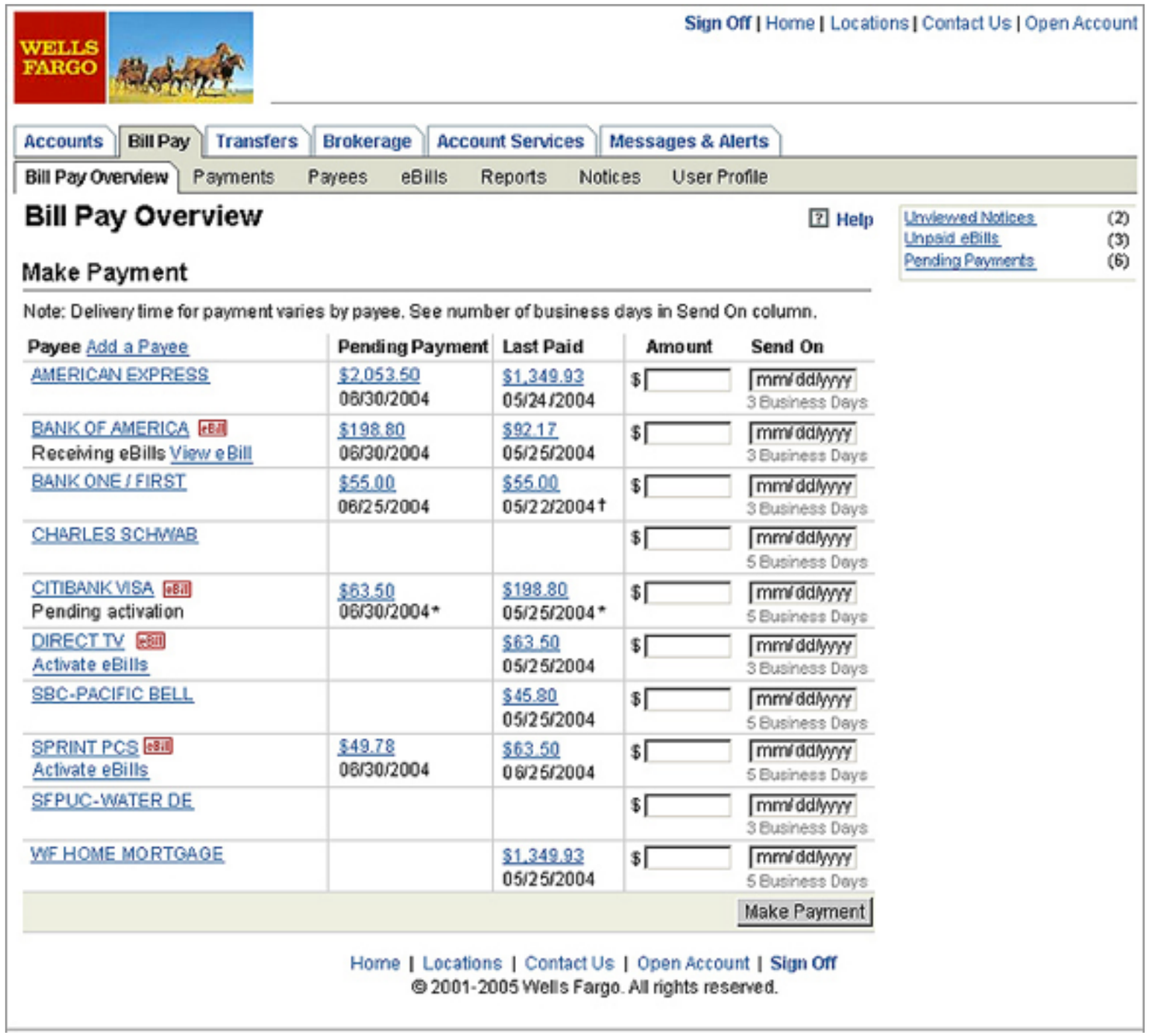
Fault tolerance is a requirement, not a feature

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

Characteristics

Characteristics

100% automated



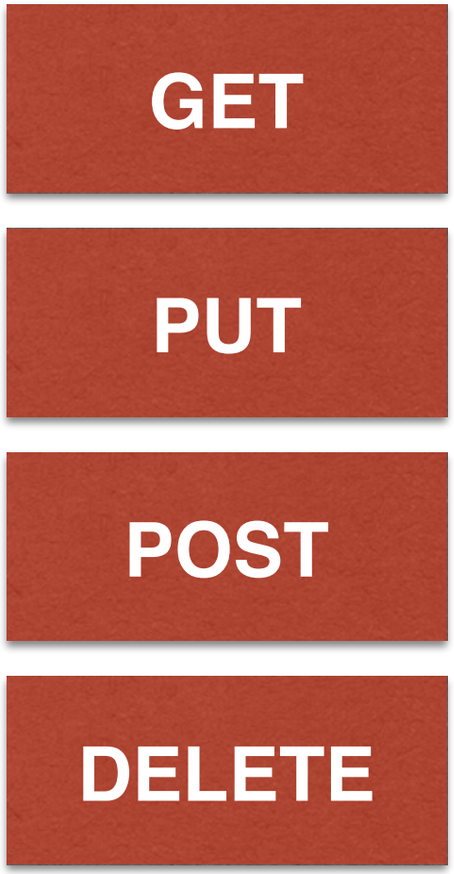
redhat

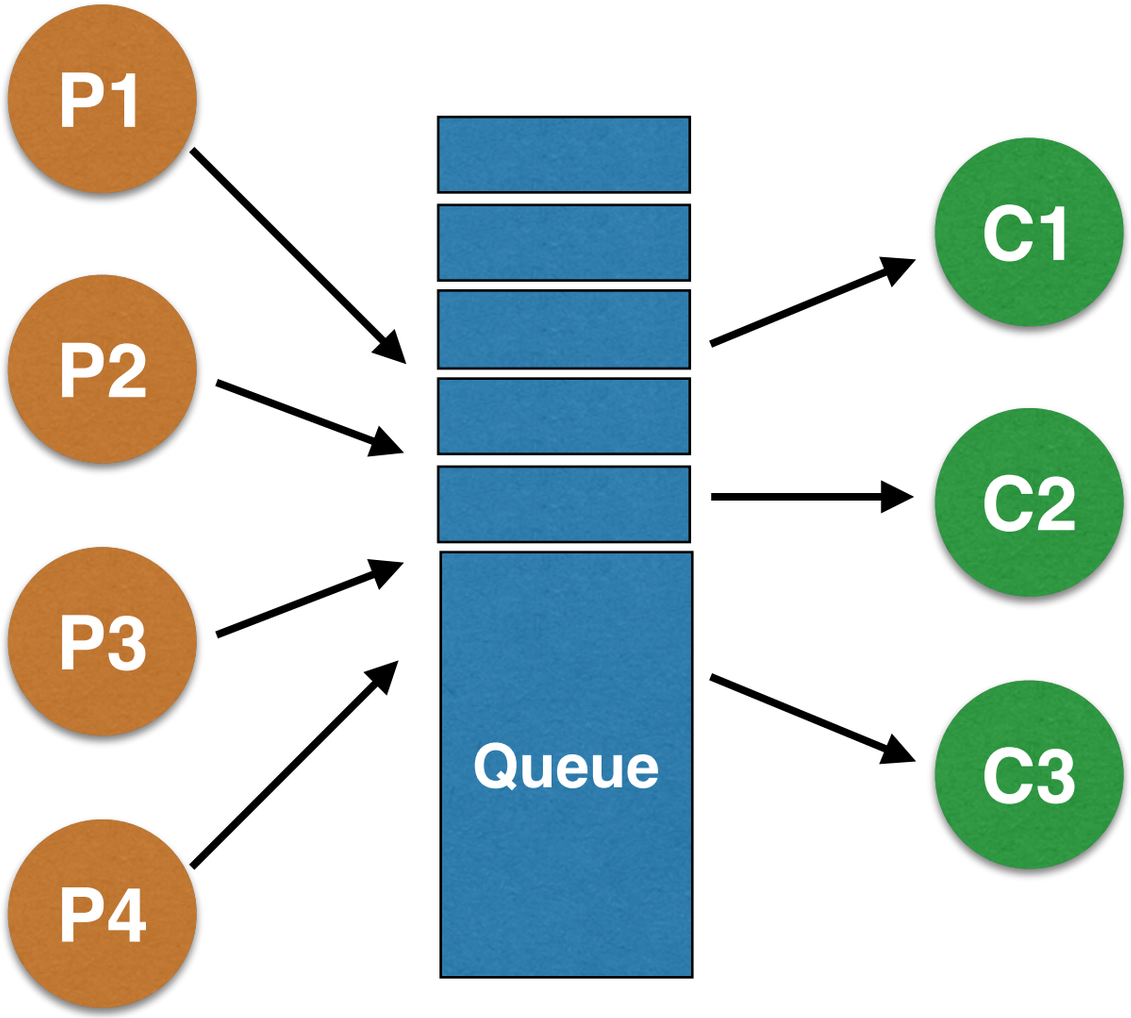
Characteristics

Sync or Async Messaging

Characteristics

REST vs Pub/Sub





Characteristics

“Smart endpoints Dumb pipes”

SOA

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

SOA 2.0?

- Conway's Law
- Service Discovery
- Immutable VM

Arun Gupta
@arungupta

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

RETWEETS 72 FAVORITES 63

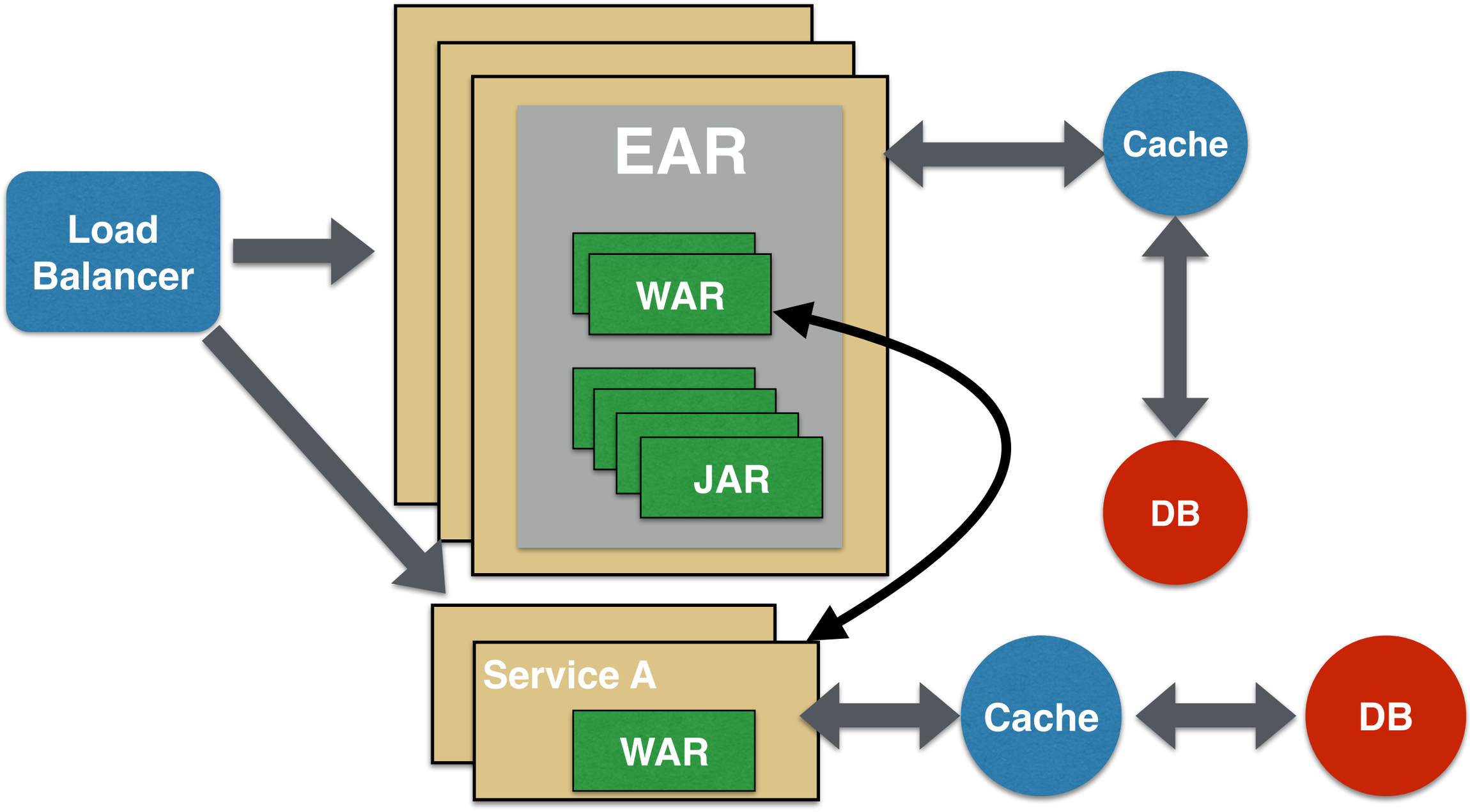
5:07 PM - 27 May 2015

Strategies for decomposing

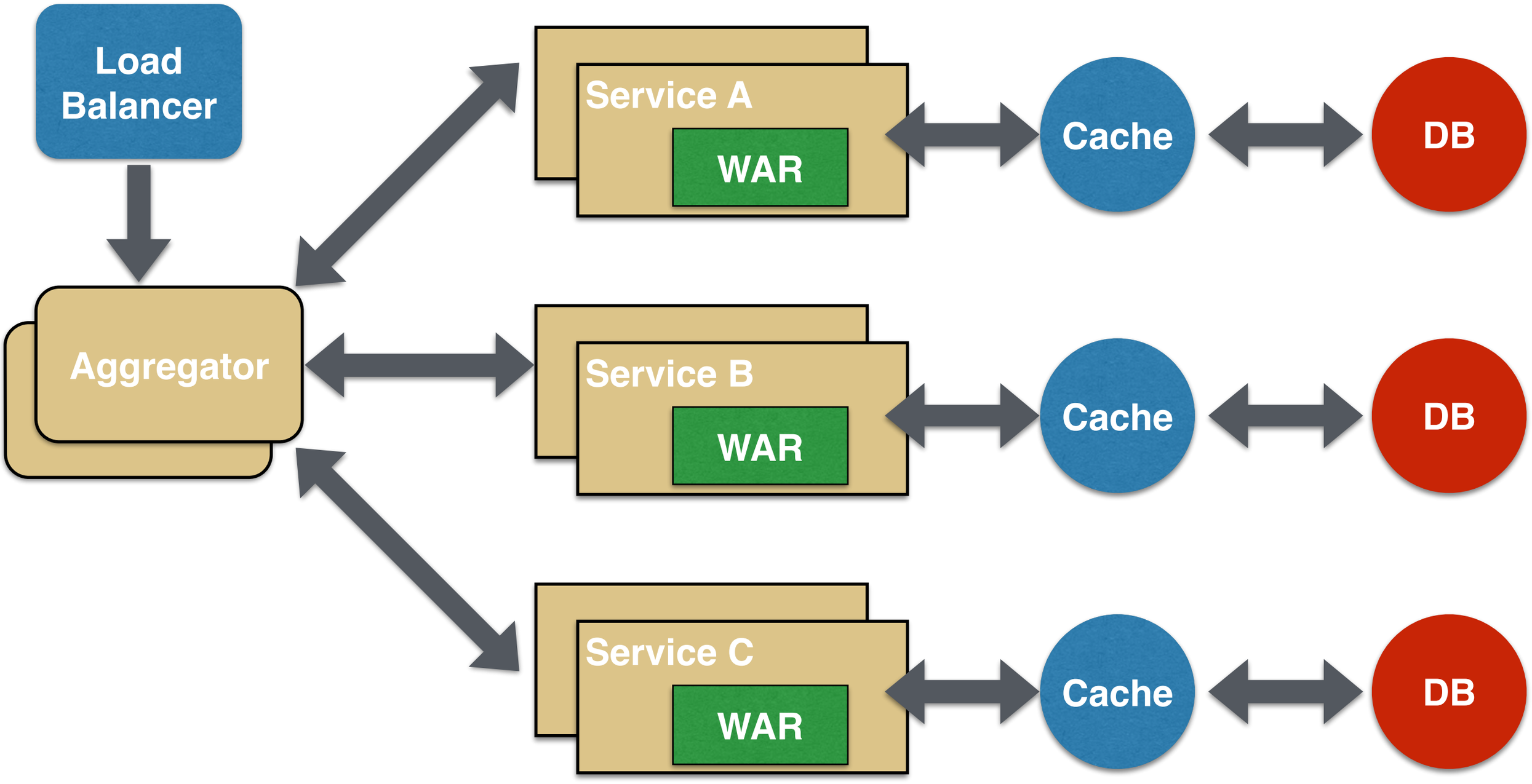
Strategies for decomposing

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

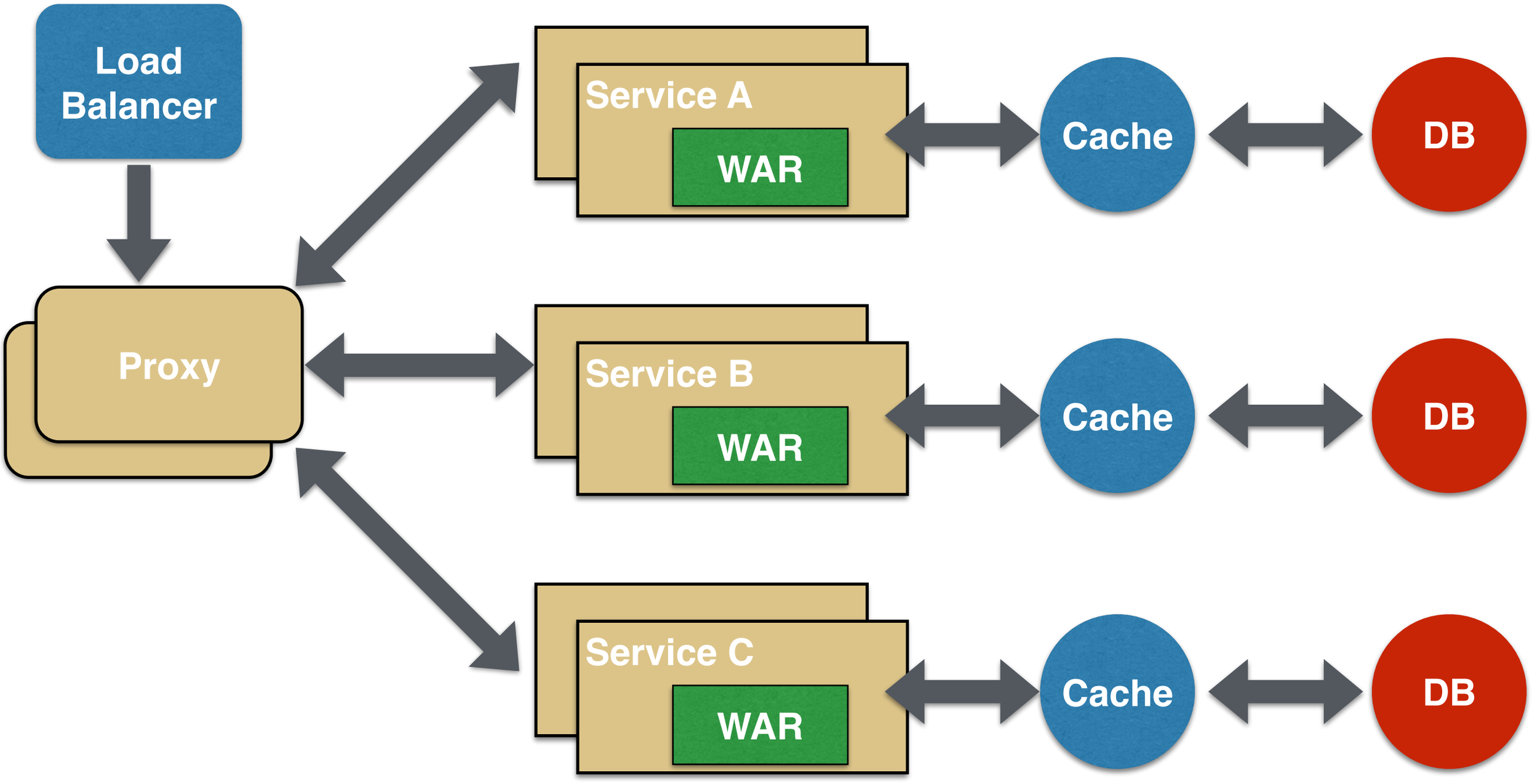
Towards microservices



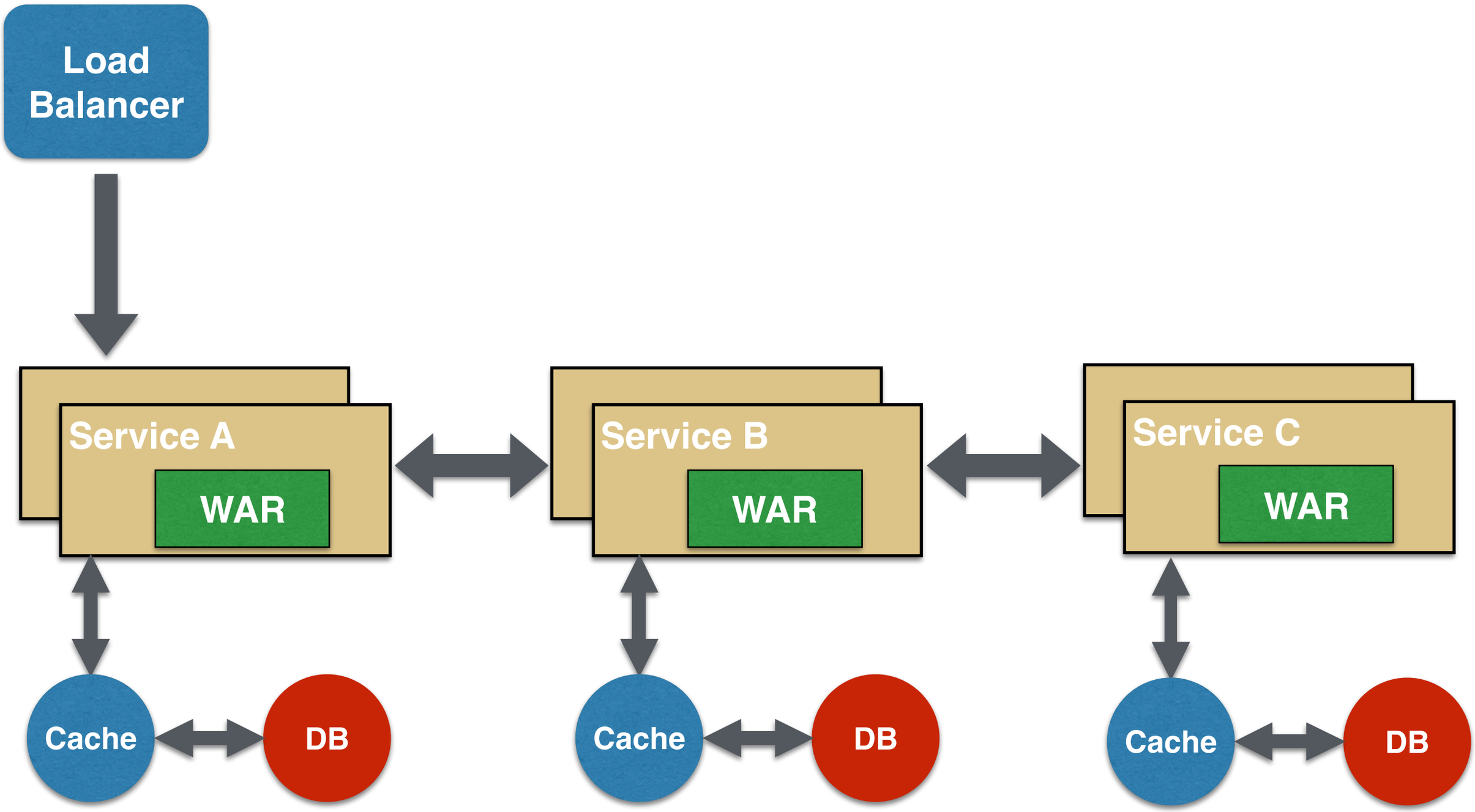
Aggregator Pattern #1



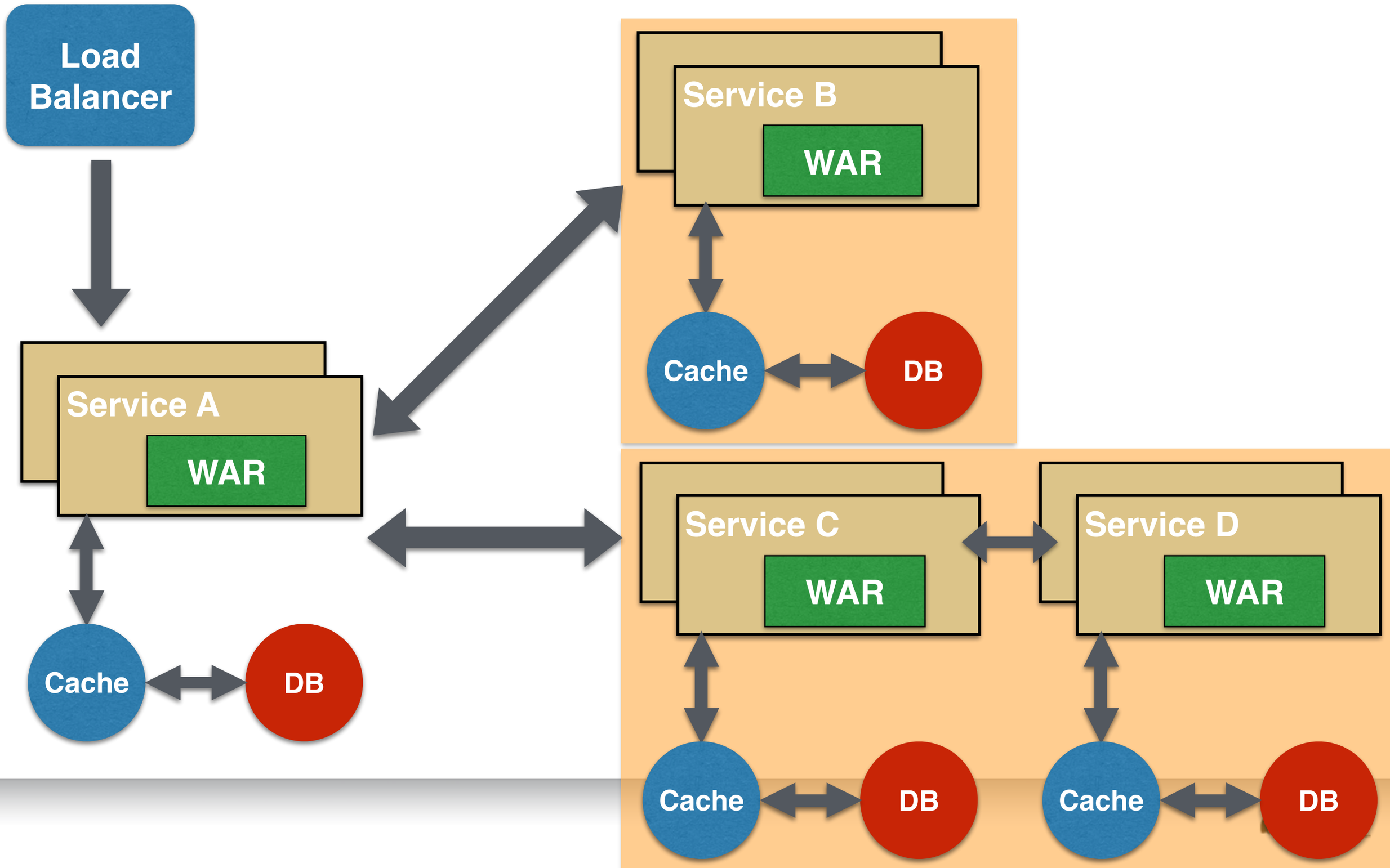
Proxy Pattern #2



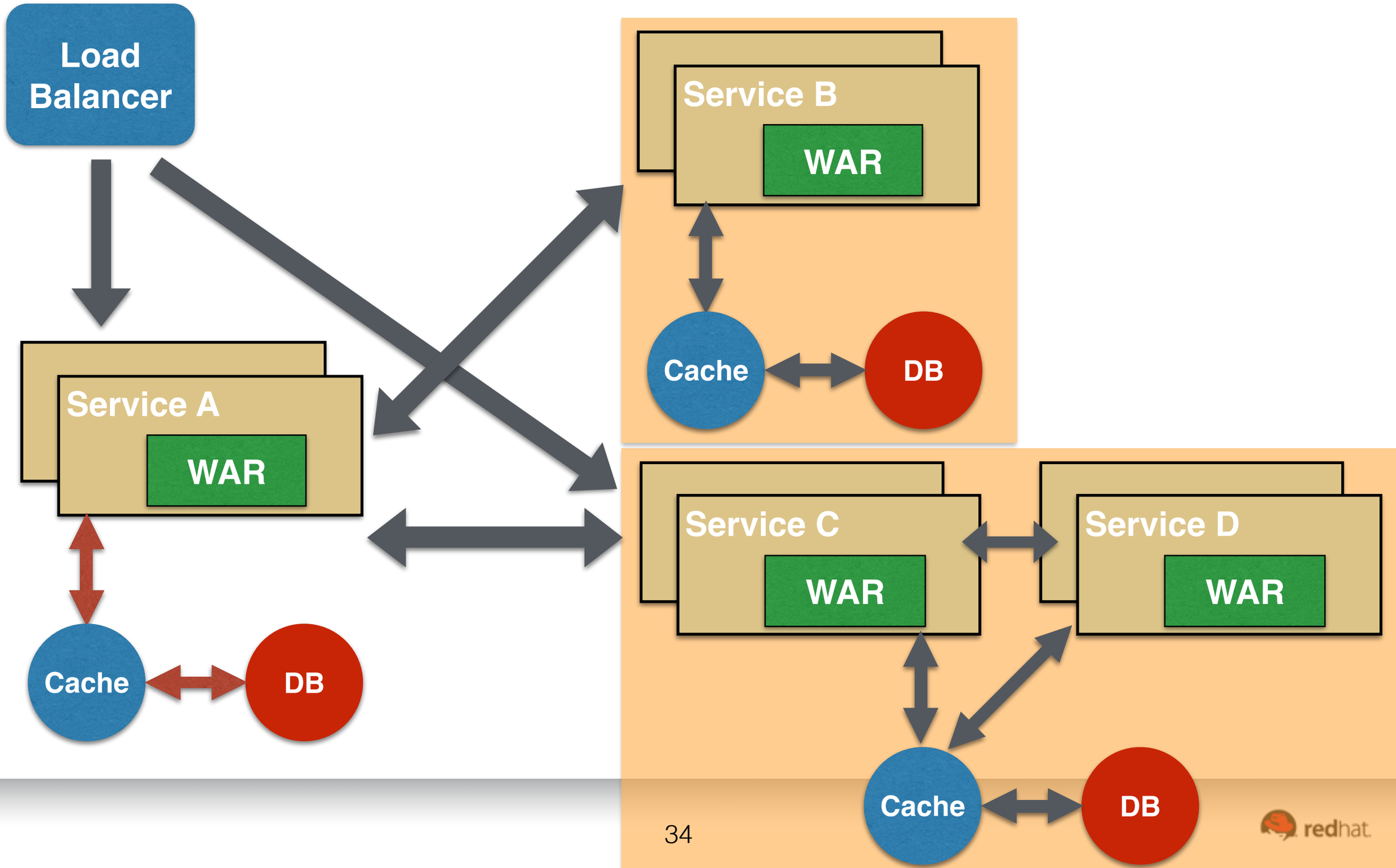
Chained Pattern #3



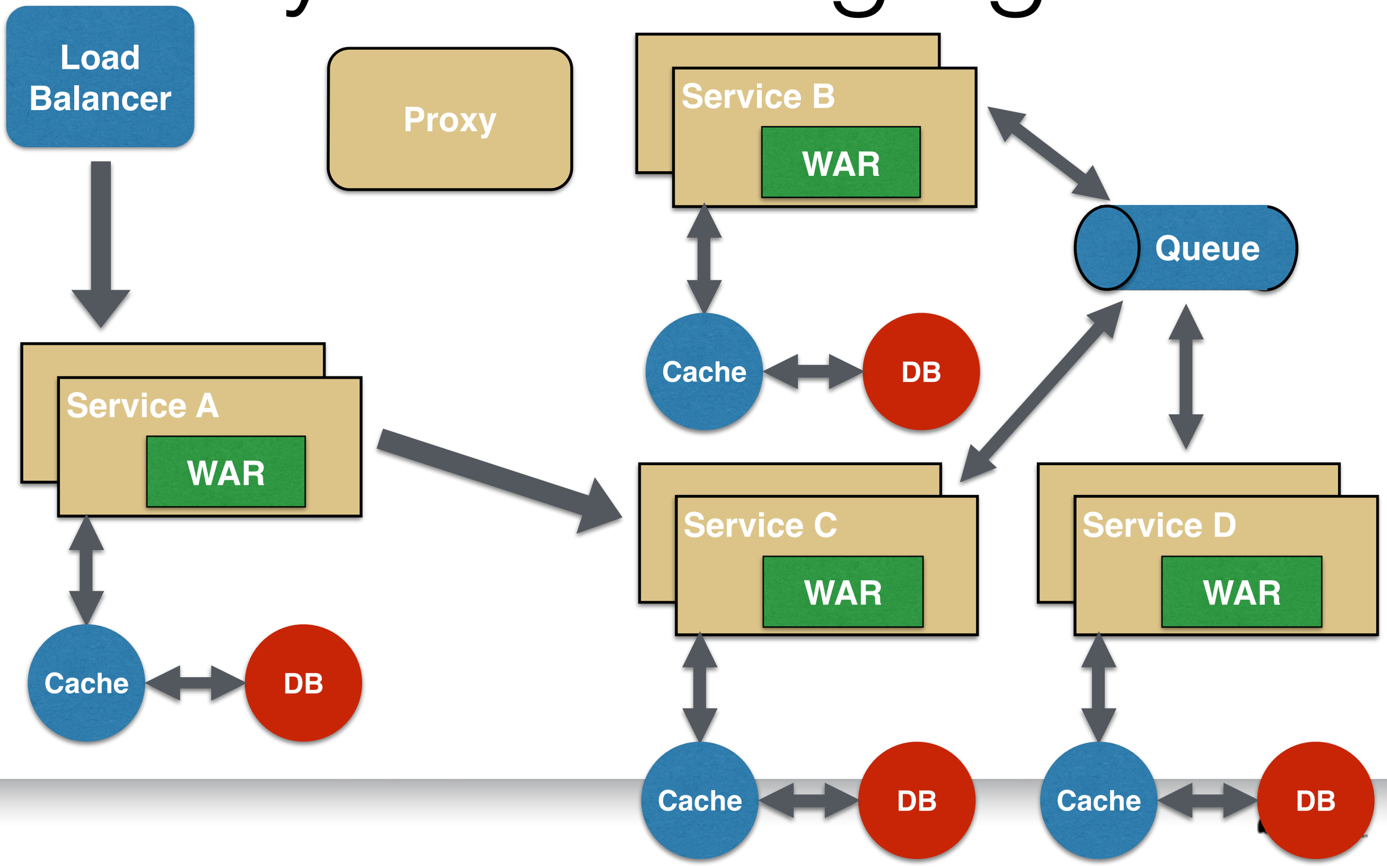
Branch Pattern #4



Shared Resources #5



Async Messaging #5



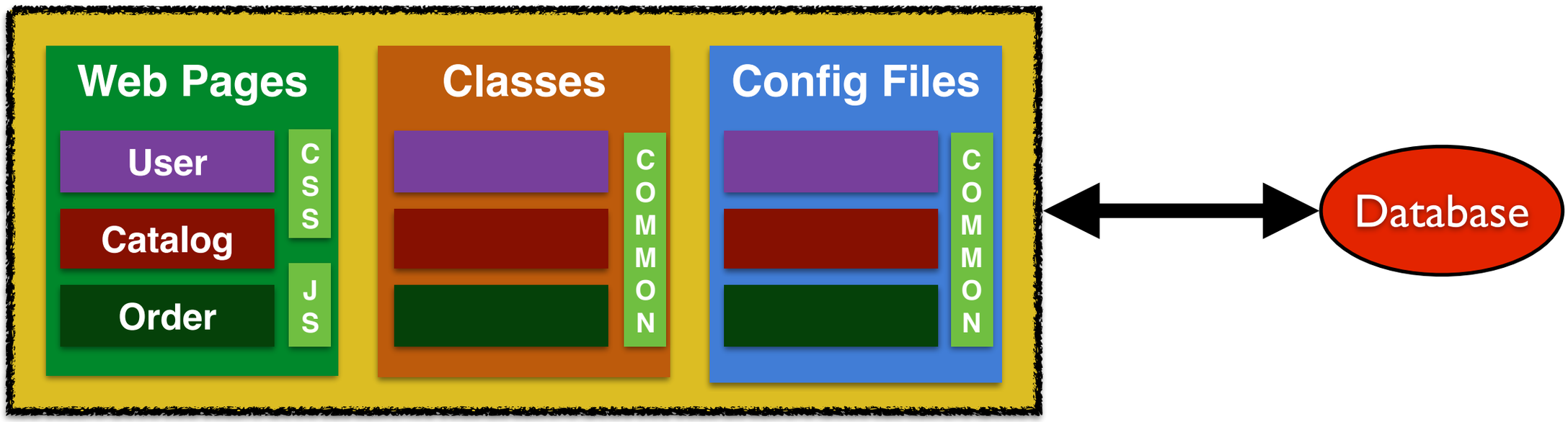

SAY MICROSERVICE
ONE MORE TIME
memegenerator.net

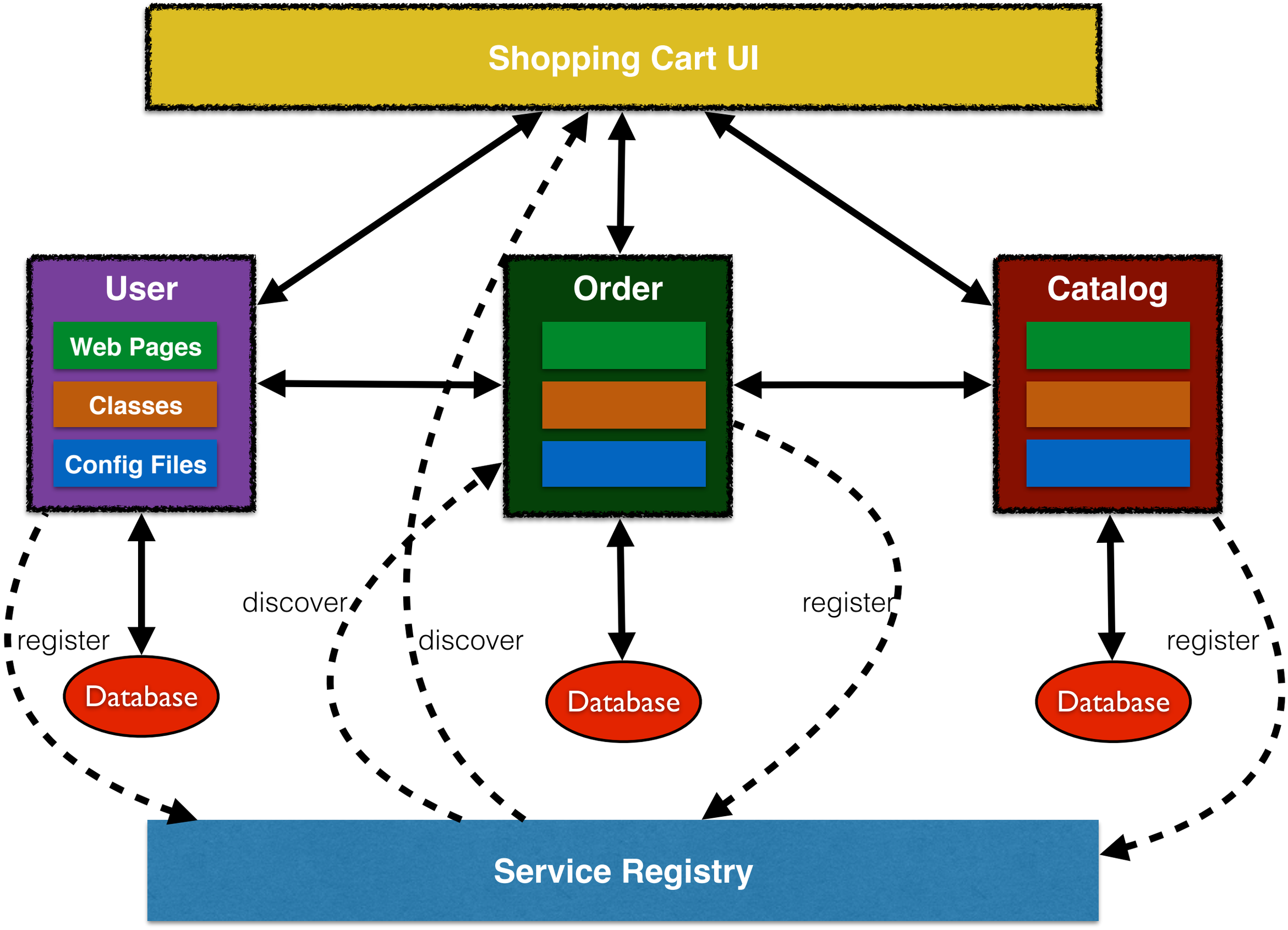
Advantages of microservices

- 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?”

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


Web Pages
User
Catalog
Order
CSS
JS
Classes
COMMON
Config Files
COMMON
Database


Shopping Cart UI
User
Web Pages
Classes
Config Files
Order
Catalog
register
discover
discover
register
register
Database
Database
Database
Service Registry

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)	3 per archive
Classes	12	26 (Service registration/ discovery, Application)
Archive Size	24 KB	~52 KB total

http://blog.arungupta.me/monolithic-microservices-refactoring-javaee-applications/

Design Considerations

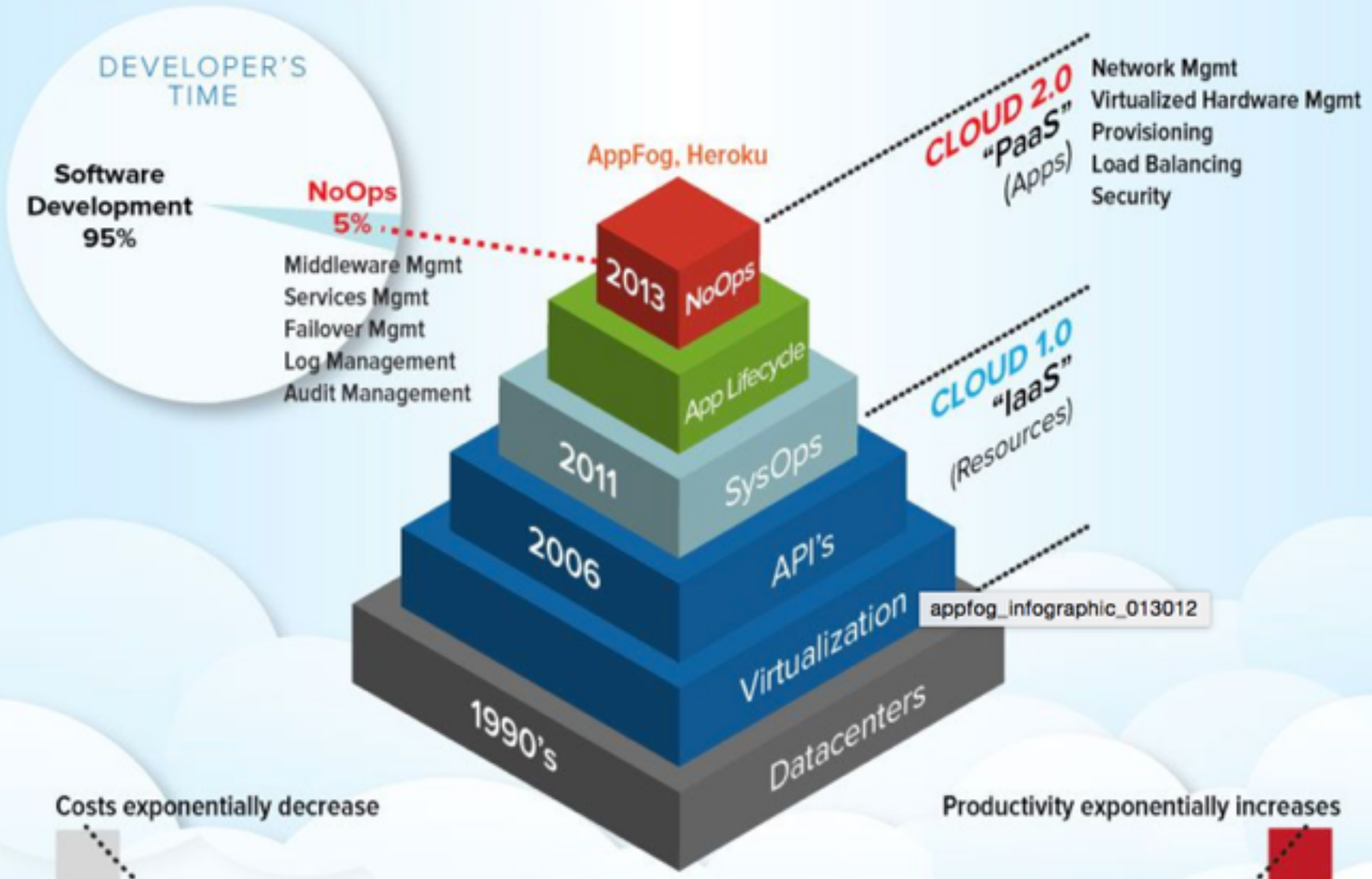
- UI and Full stack
  - Client-side composition (JavaScript?)
  - Server-side HTML generation (JSF?)
  - One service, one UI
- REST Services
- Compensating transactions instead of 2PC

NoOps

- Service replication (k8s, fabric8, etcd, ZK, …)
- Dependency resolution (Nexus, …)
- Failover (Circuit Breaker)
- Resiliency (Circuit Breaker)
- Service monitoring, alerts and events (New Relic, Log stash, …)

2013: A bright NoOps future

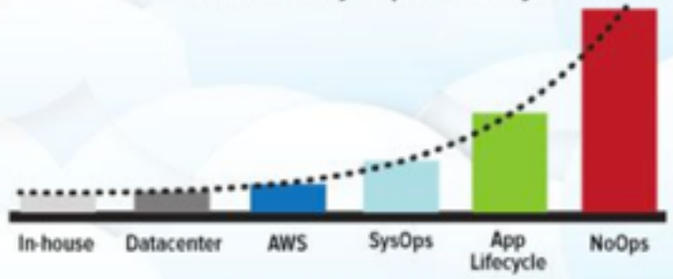
So where does this all lead? The end-game is NoOps. Where building and running an app is purely a developer process — and where developers are not having to spend time doing Ops work.



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.

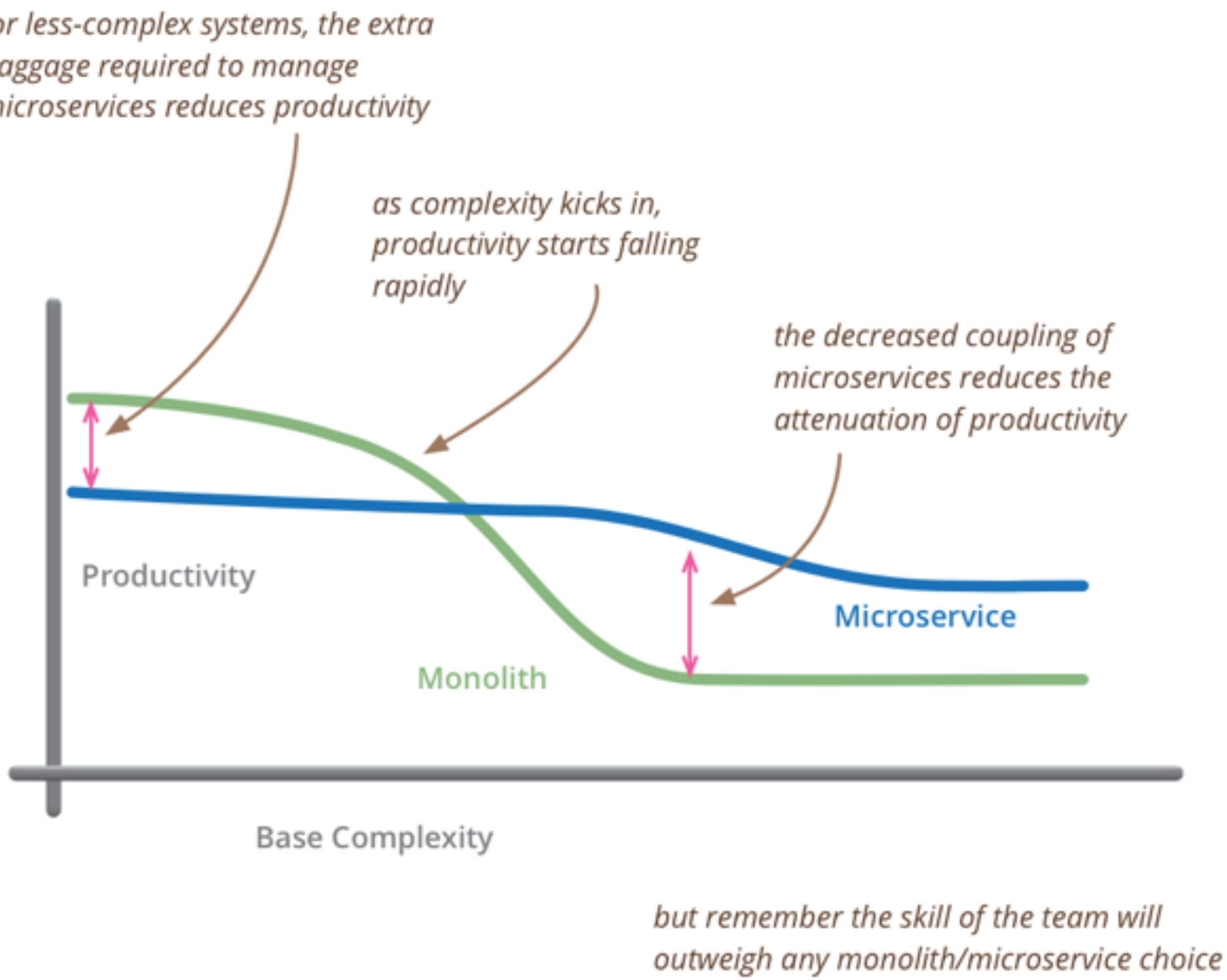
By freeing developers to focus on what they want to do — and on what they do best — we are enabling them to even more easily start companies and build products. And that is why AppFog is a *Start-up Enabler.*

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

Microservice Premium



"don't even consider microservices unless you have a system that's too complex to manage as a monolith"

http://martinfowler.com/bliki/MicroservicePremium.html

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

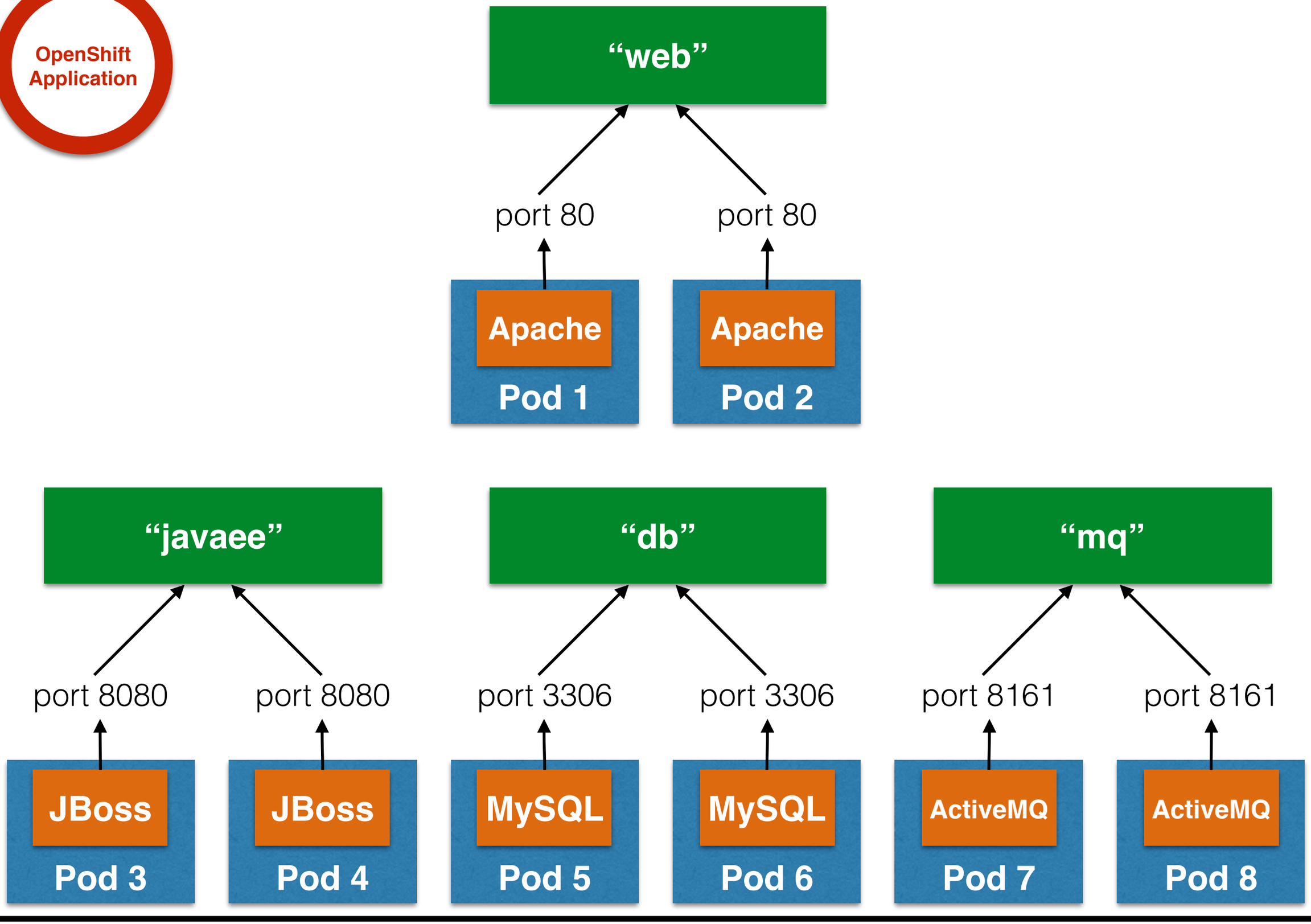
Kubernetes: Pros and Cons

- 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.

redhat.

Applications

PaaS

Containers & Orchestration

Container Host

IaaS

xPaaS

RED HAT JBOSS MIDDLEWARE

VERT.X

node

OPENSHIFT

kubernetes

docker

RED HAT ENTERPRISE LINUX ATOMIC HOST

openstack

jboss.org

JBoss Tools

RED HAT JBOSS DEVELOPER STUDIO

Arquillian

RESTEasy

WildFly

HIBERNATE

VERT.X

FEEDHENRY™

Infinispan

AeroGear

Apache Camel

Qhawtio

fabric8