Transformation From Monolith to Micro service Application Architecture.

Sabari Balaji – TLS

sabaribalaji.parthasarathy@techmahindra.com

Definition From The Experts

@ Developing a single application as a suite of small Services, each running in it own process and communicating With lightweight mechanisms, often an HTTP resource API
- Martin Fowler

@ Fine-grained SOA **– Adrian Cockcroft - Netflix**

*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***

Current Trends

- Twitter moved from Ruby/Rails monolith to Microservices
- Facebook moved from PHP monolith to Microservices
- Netflix moved from JAVA monolith to Microservices

Monolith Example

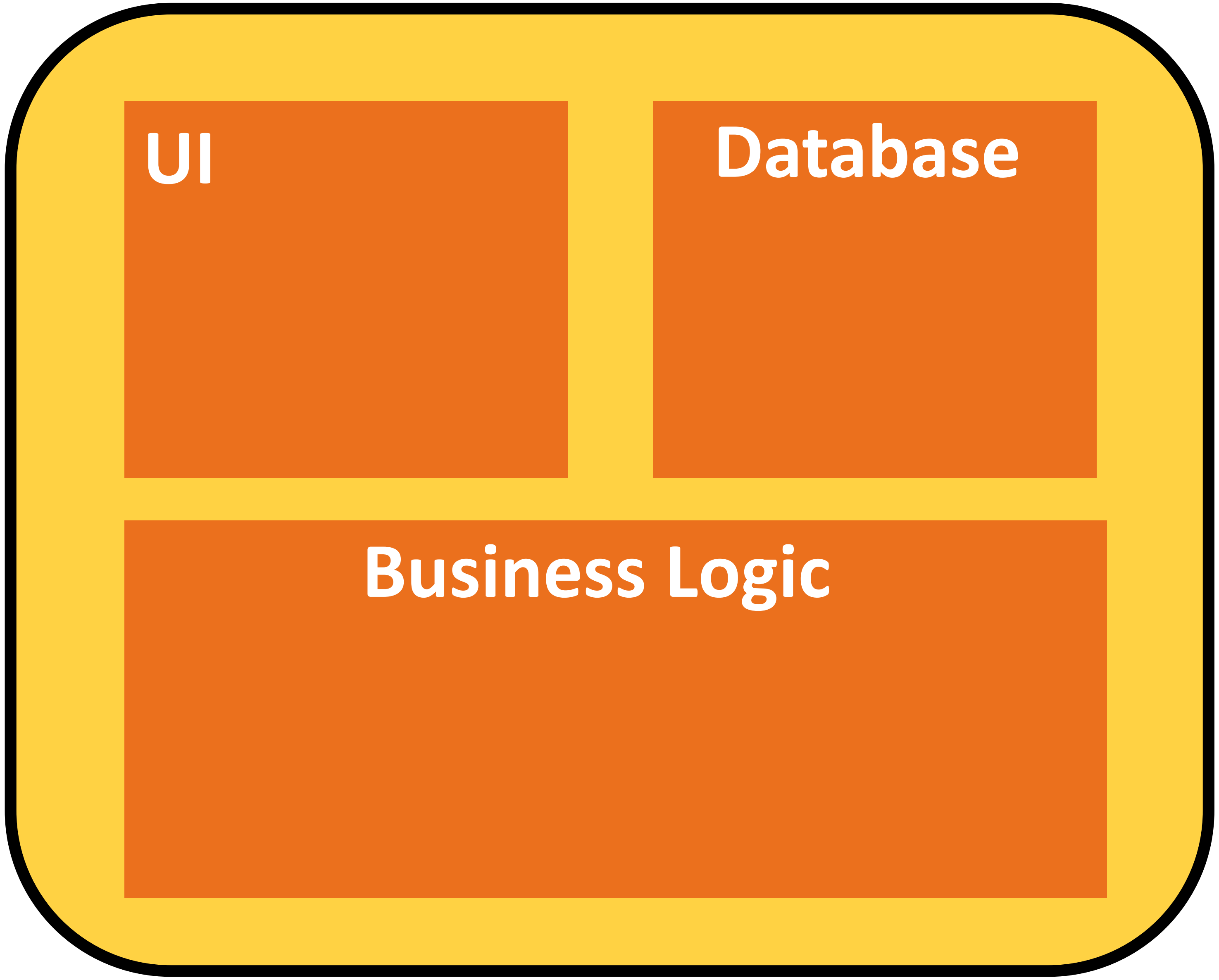
# Consider a monolithic shopping cart application:

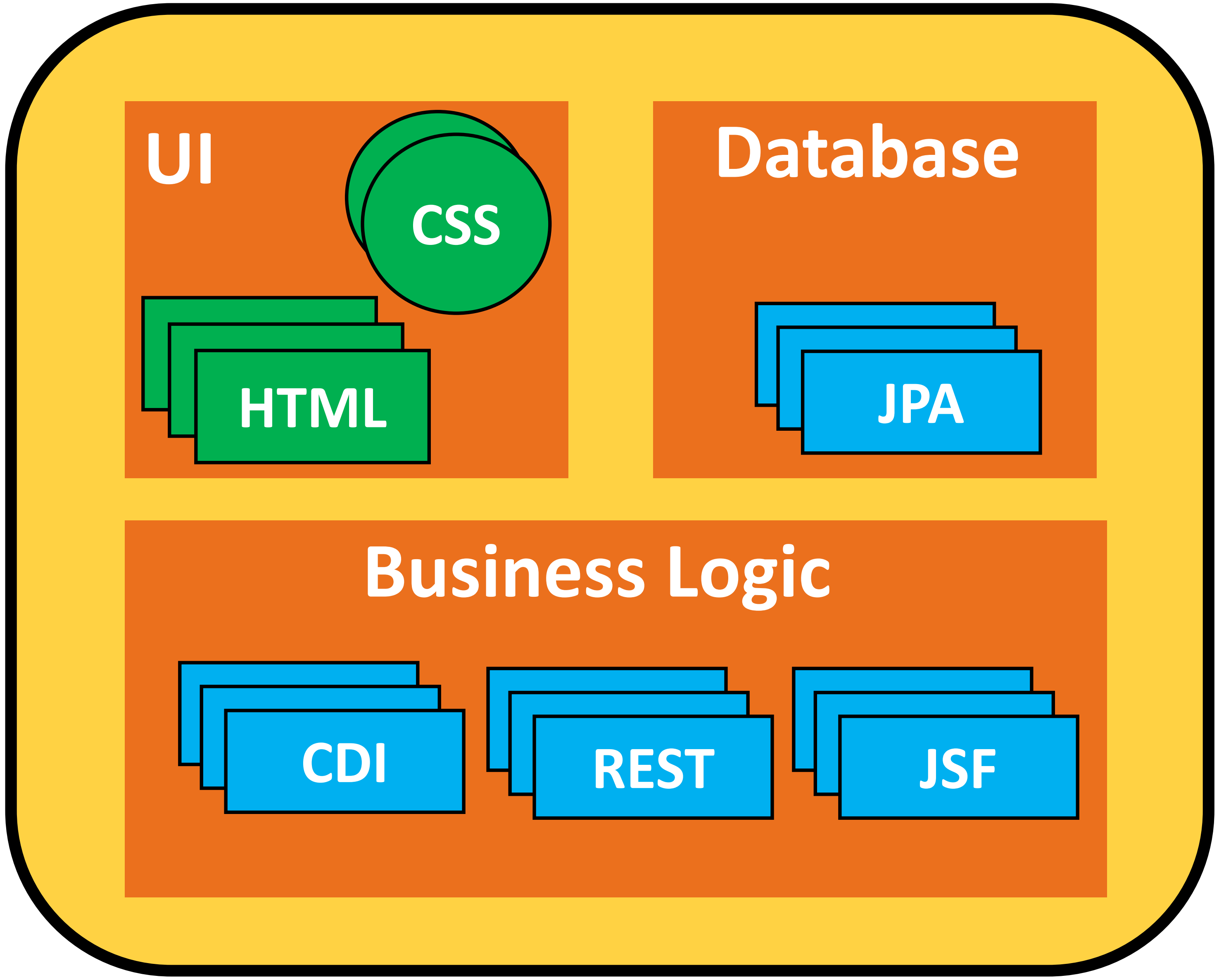
(Web / mobile interfaces)

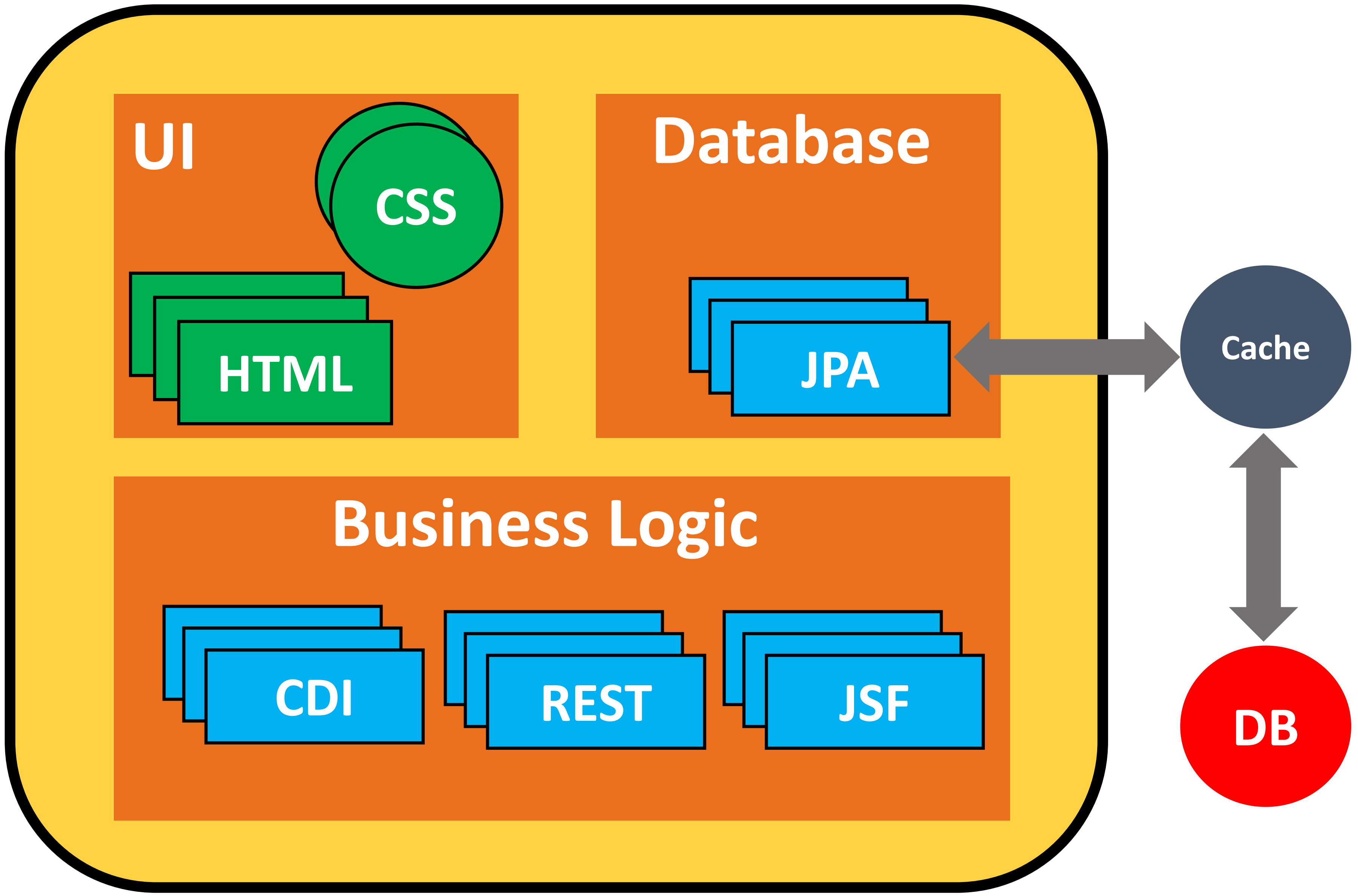
Functions for:

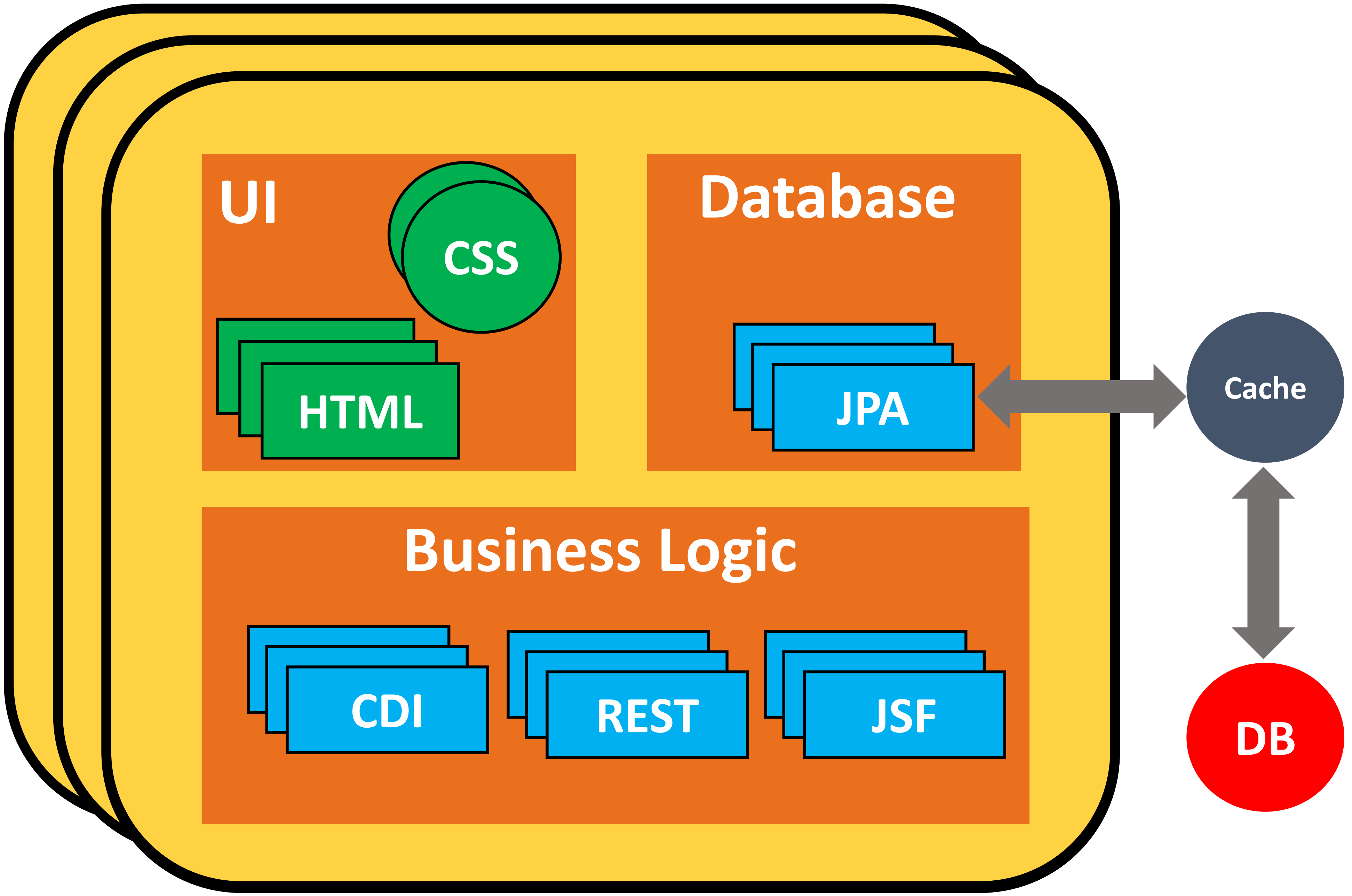
- # Searching for products
- # Product catalog
- # Inventory Management
- # Shopping cart
- # Checkout
- # Fullfilment

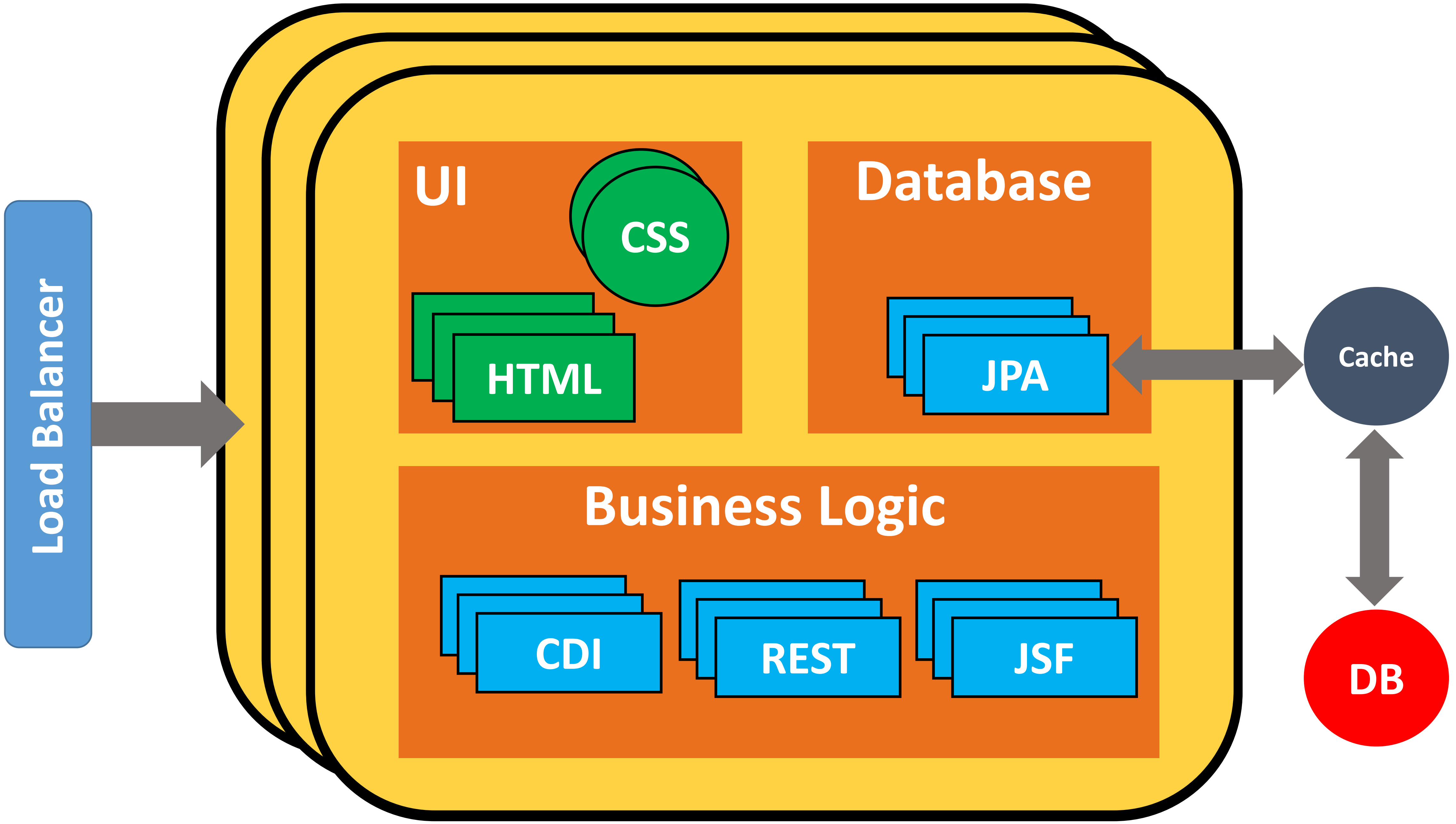
How would this look with microservices ?

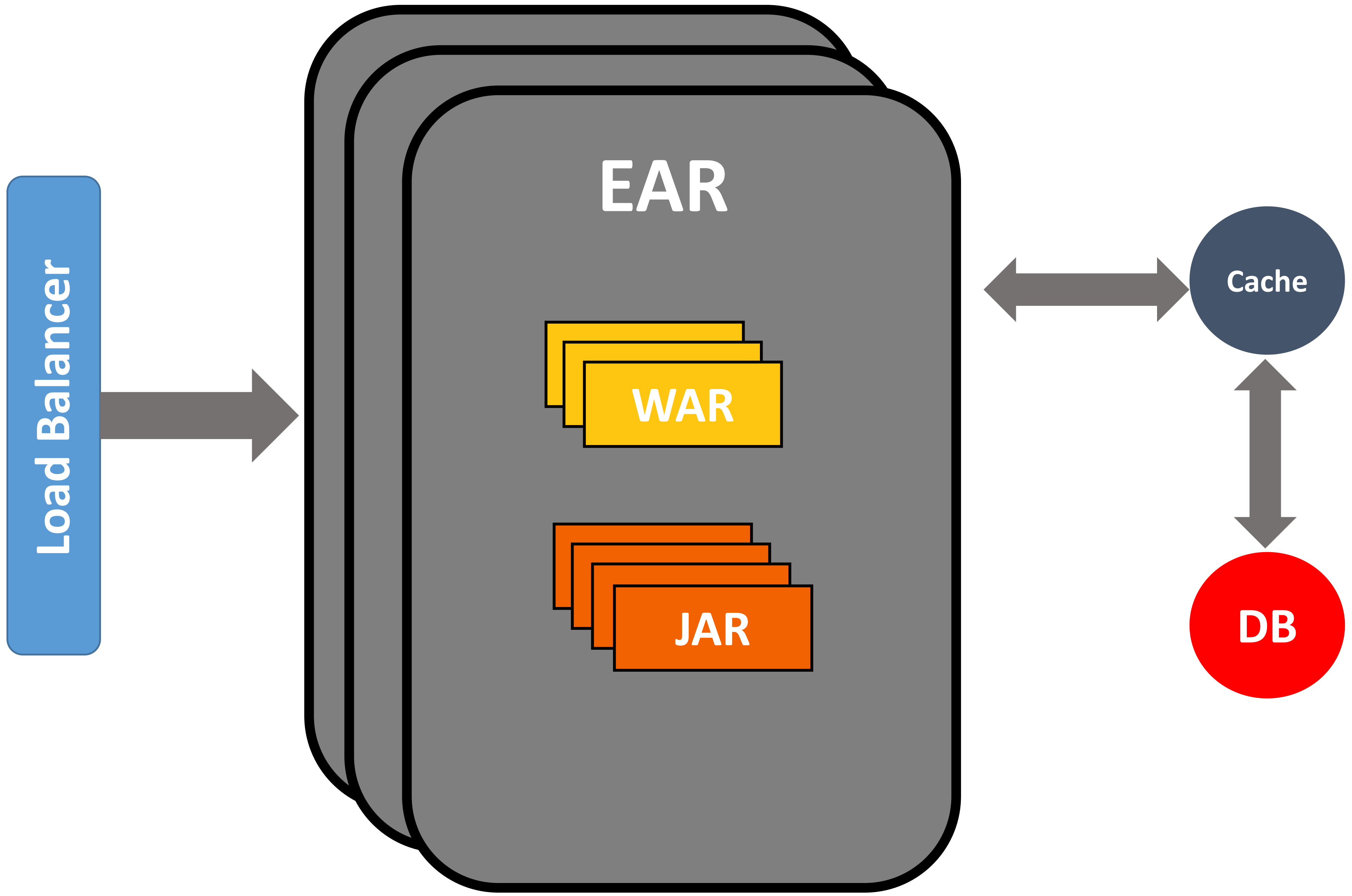

UI
Database
Business Logic

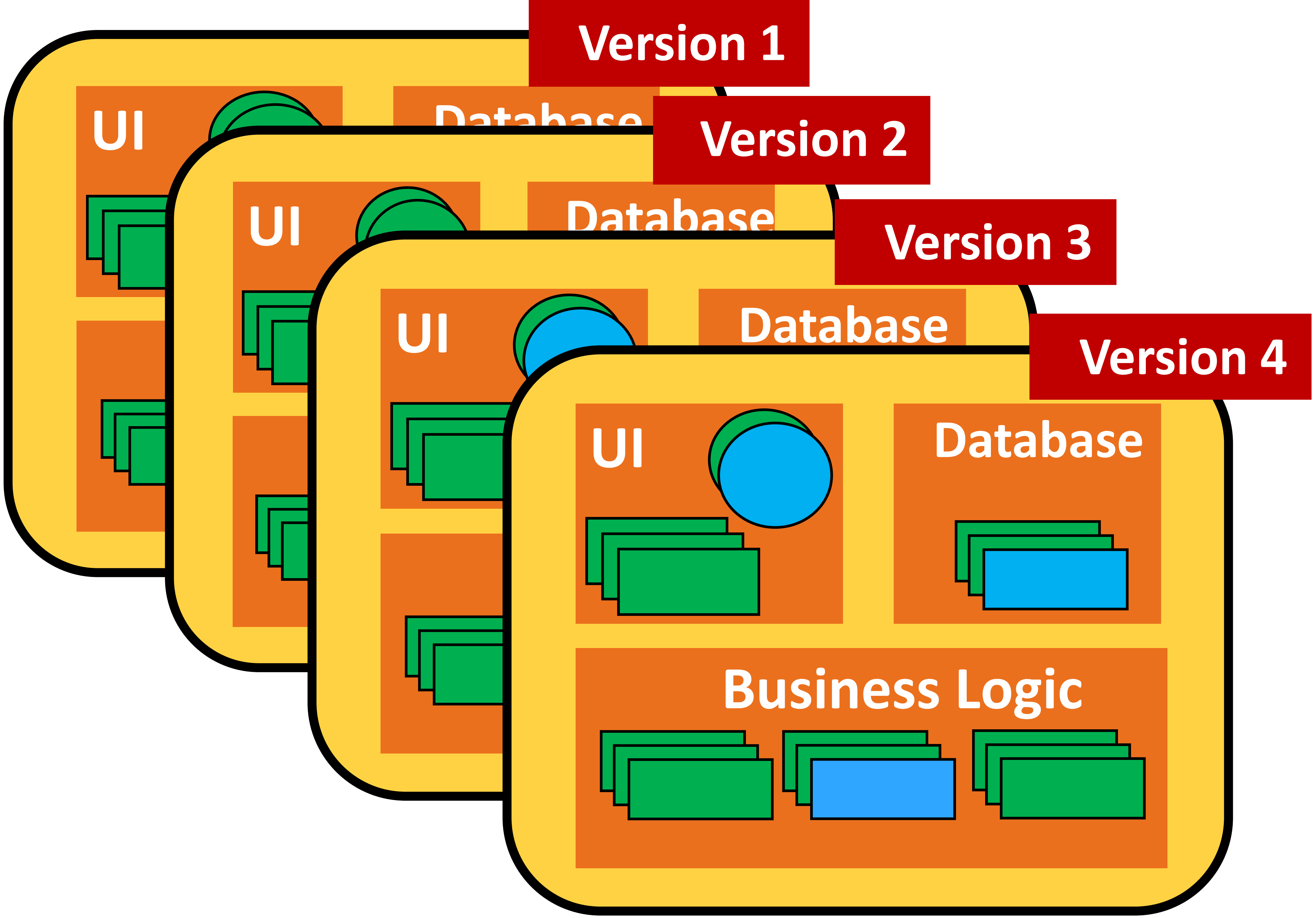

UI
CSS
HTML
Database
JPA
Business Logic
CDI
REST
JSF

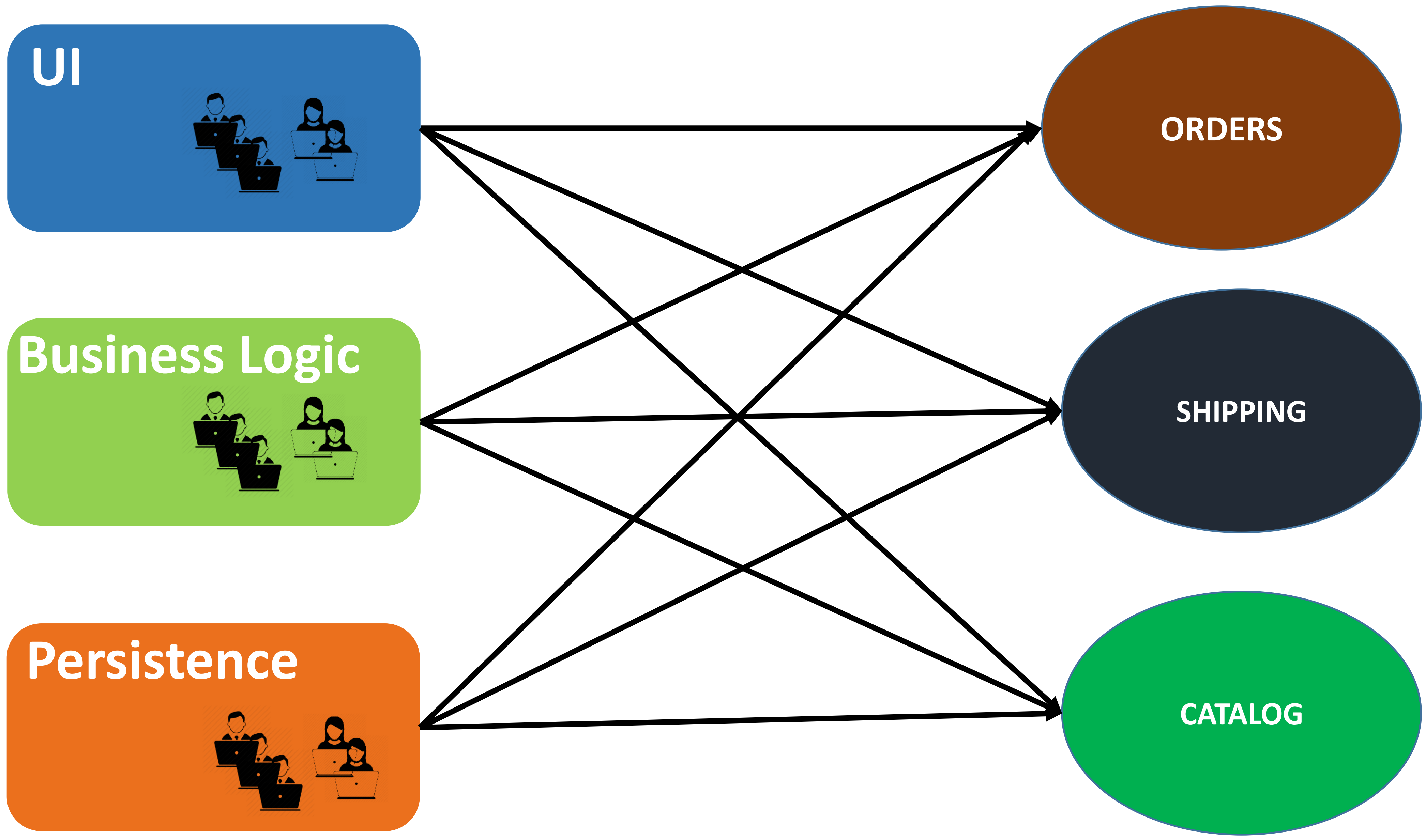

UI
CSS
HTML
Database
JPA
Cache
Business Logic
CDI
REST
JSF
DB


UI
CSS
HTML
Database
JPA
Cache
DB
Business Logic
CDI
REST
JSF

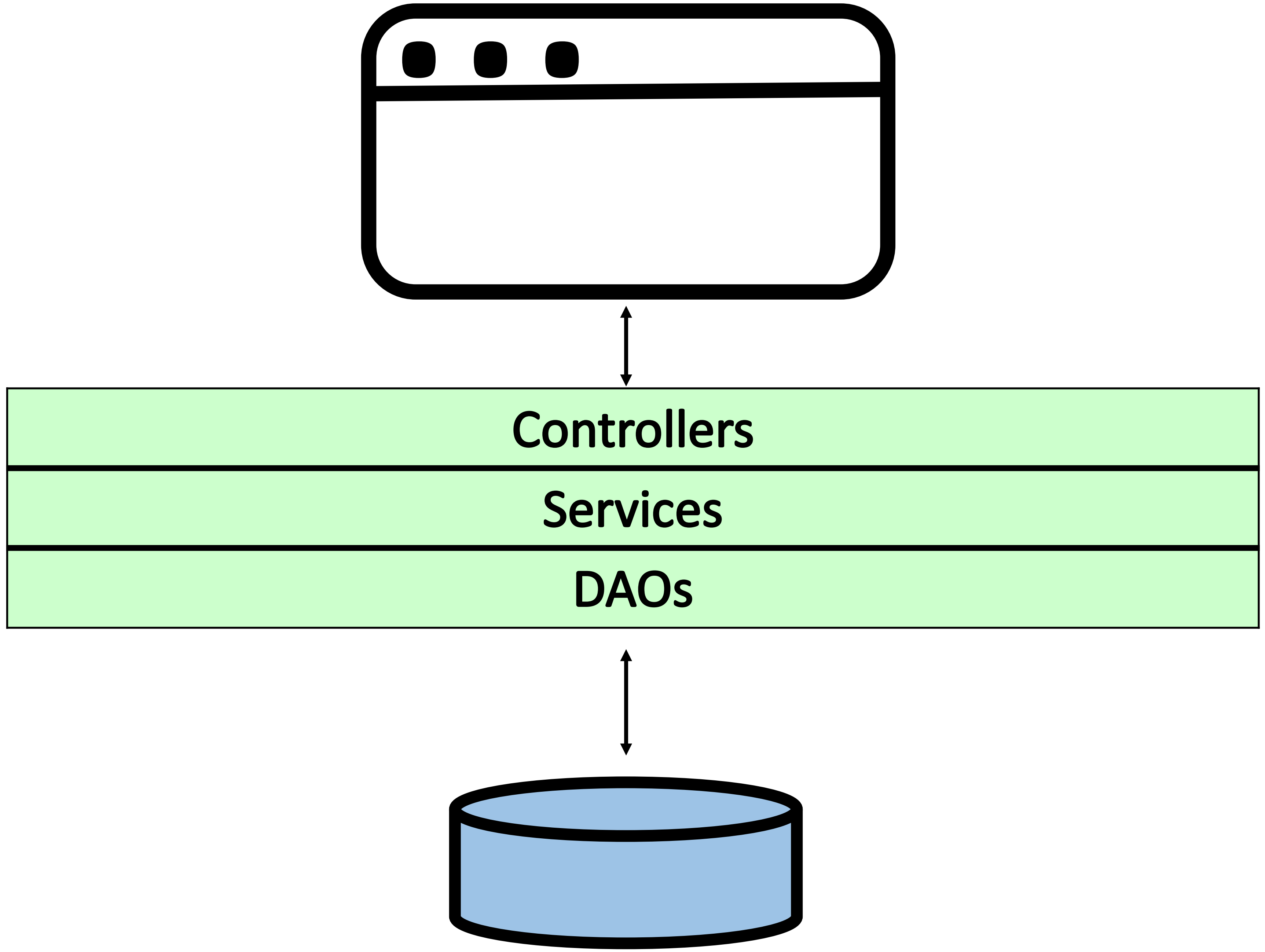

Load Balancer
UI
CSS
HTML
Database
JPA
Business Logic
CDI
REST
JSF
Cache
DB


Load Balancer
EAR
WAR
JAR
Cache
DB

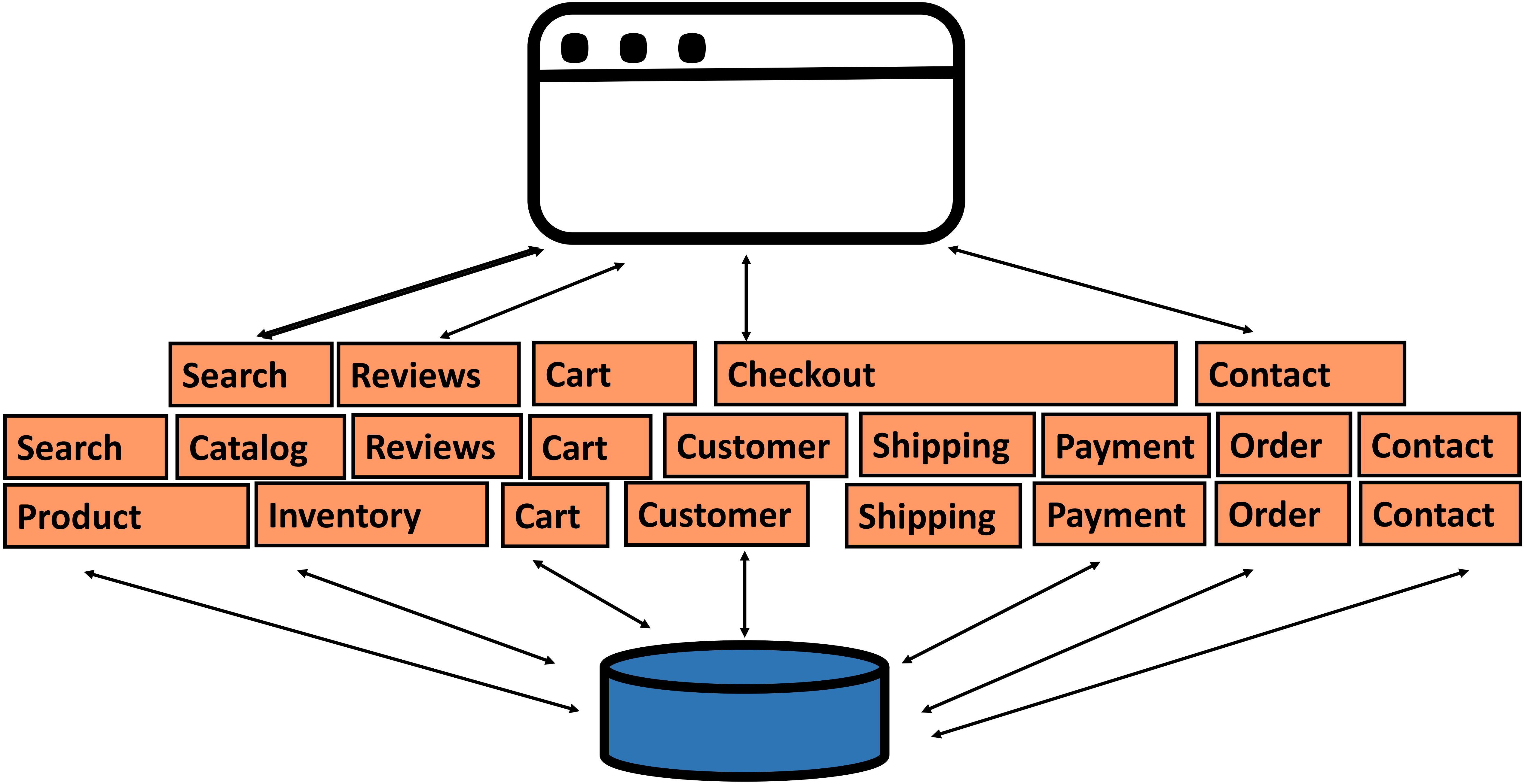

Version 1
Version 2
Version 3
Version 4
UI
Database
UI
Database
UI
Database
UI
Database
Business Logic


UI
Business Logic
Persistence
ORDERS
SHIPPING
CATALOG

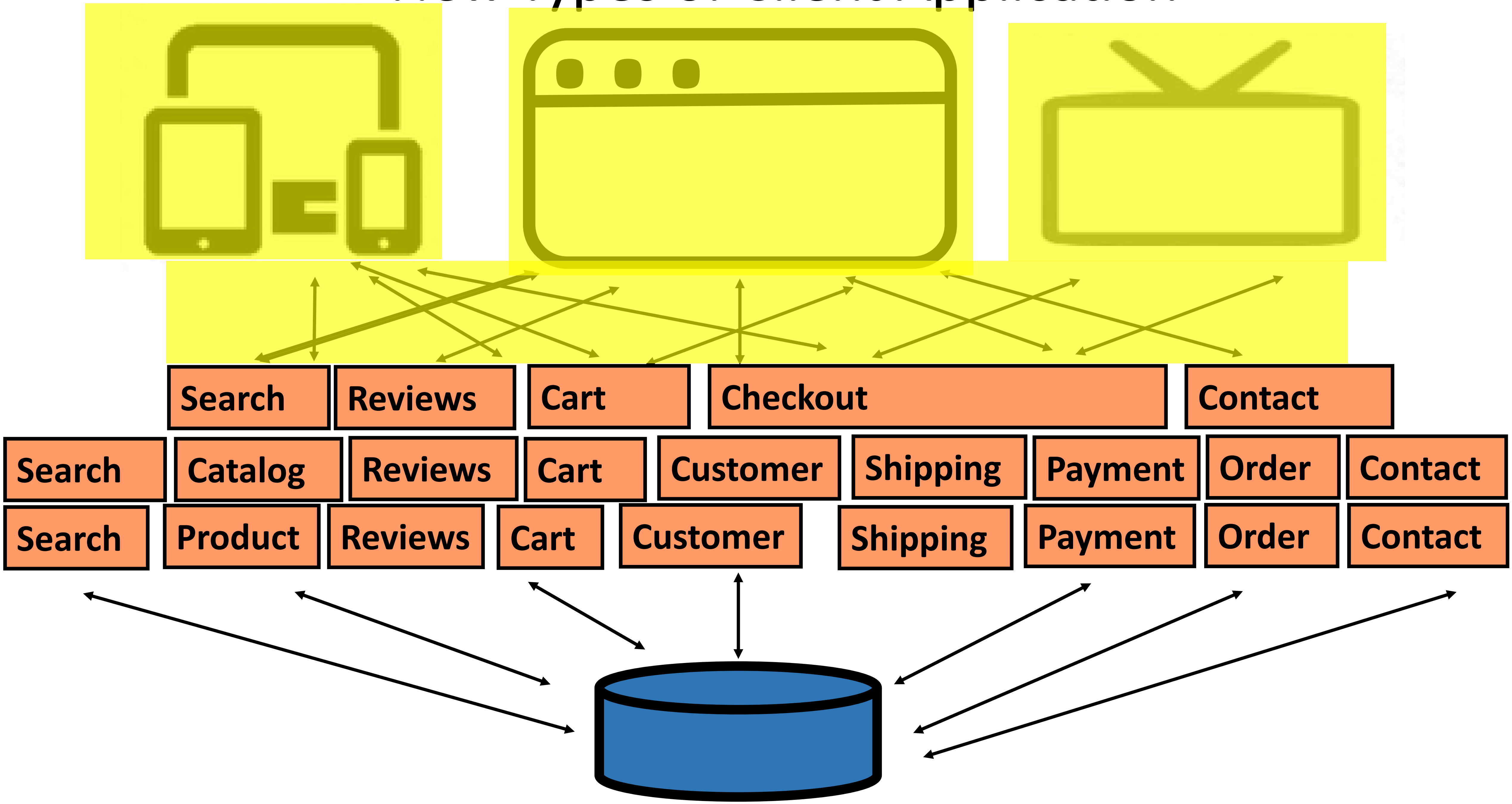
Monolithic Application Example



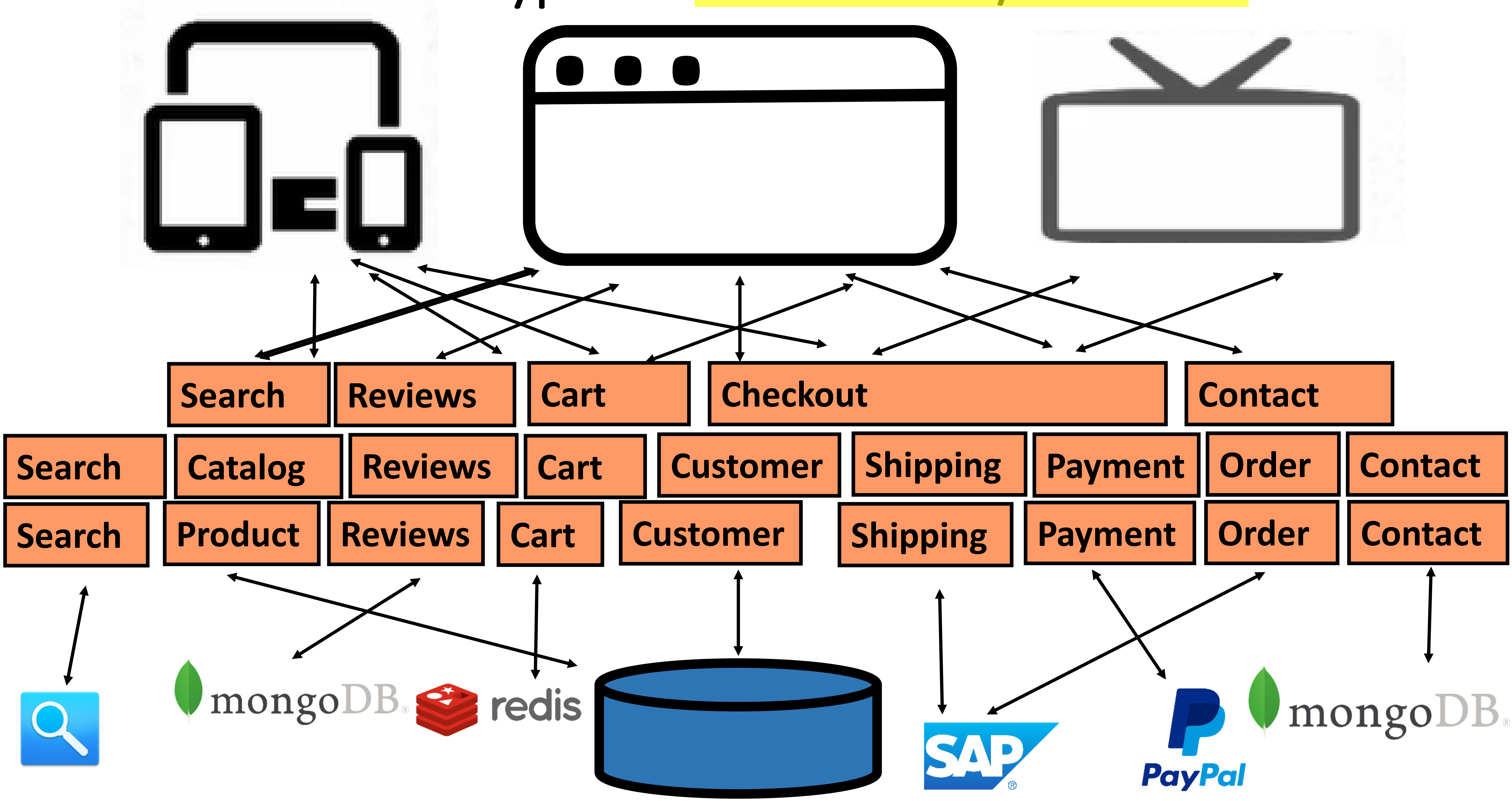
Monolithic Application Example



New Types of Client Application

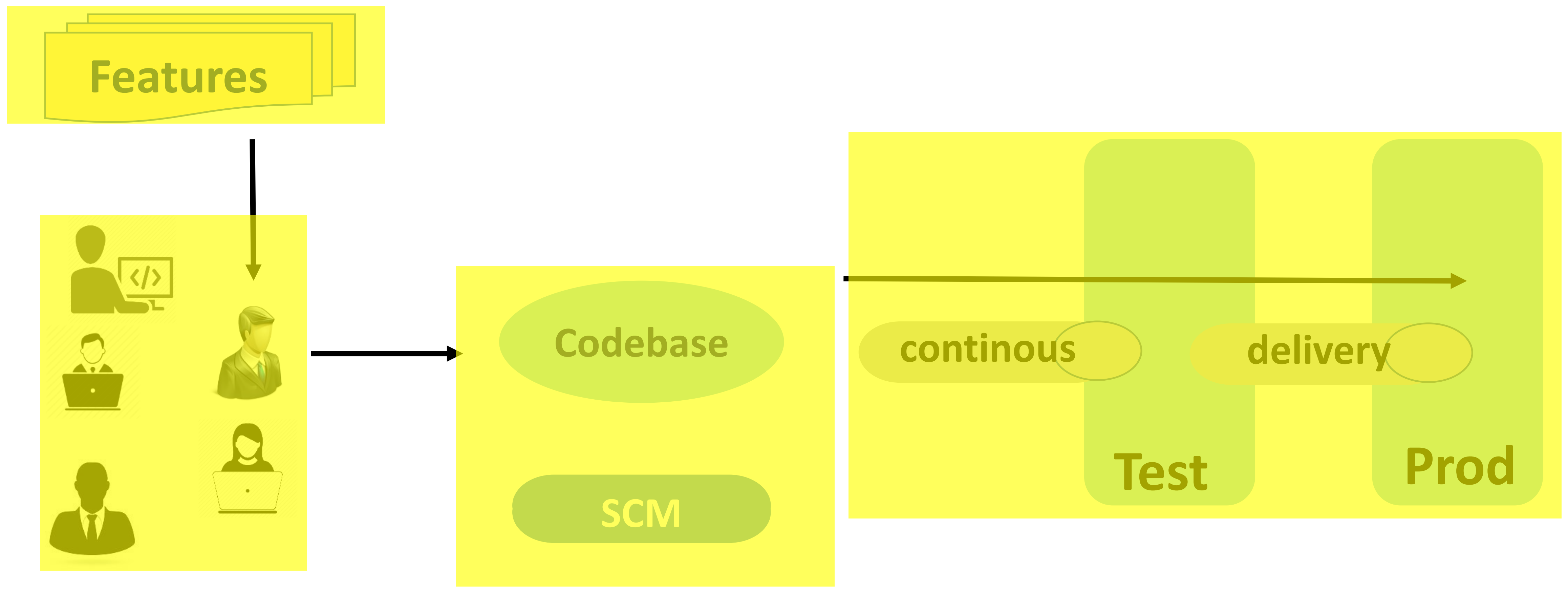


New Types of Persistence / Service



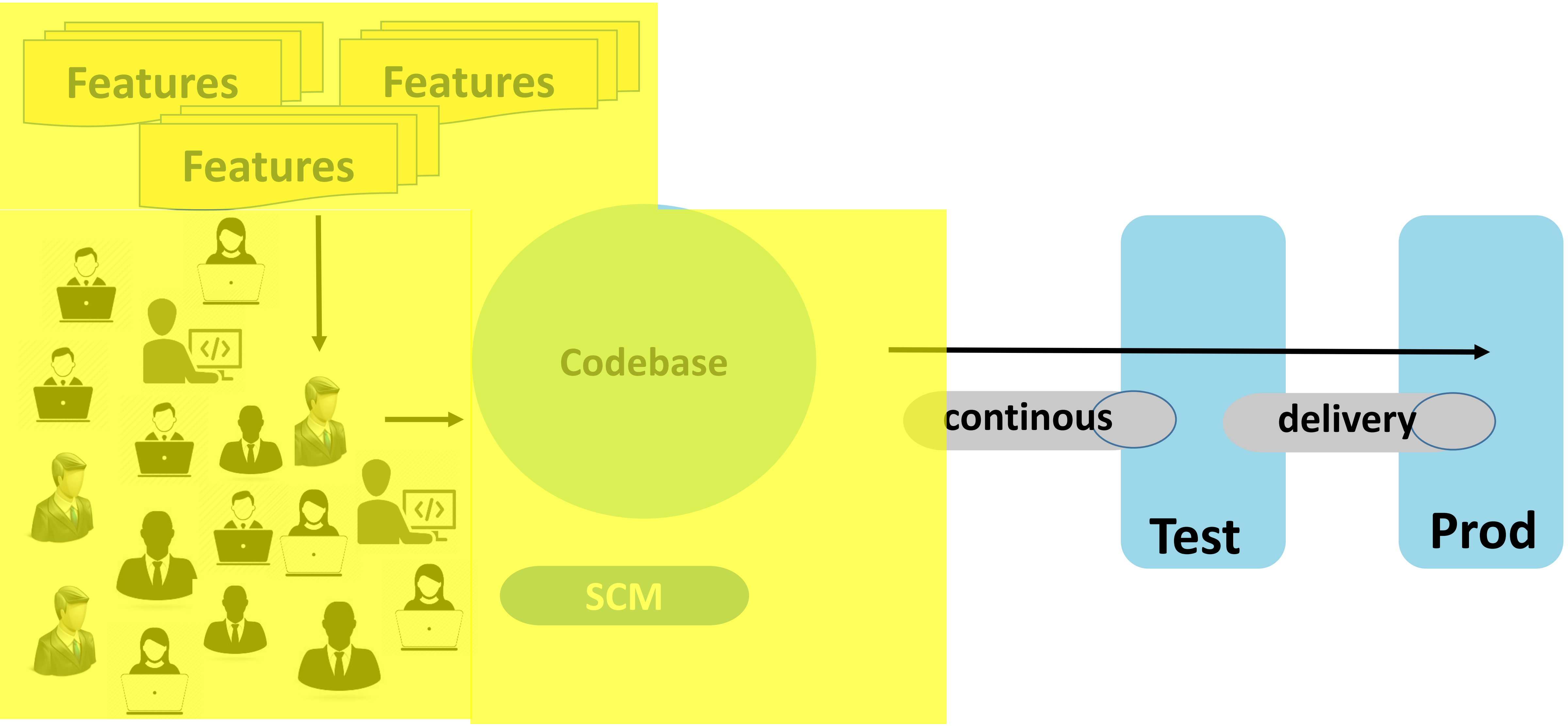
Monolith Challenges – Broader VIEW

# Single Codebase, Deployment, Versioning, Team Size



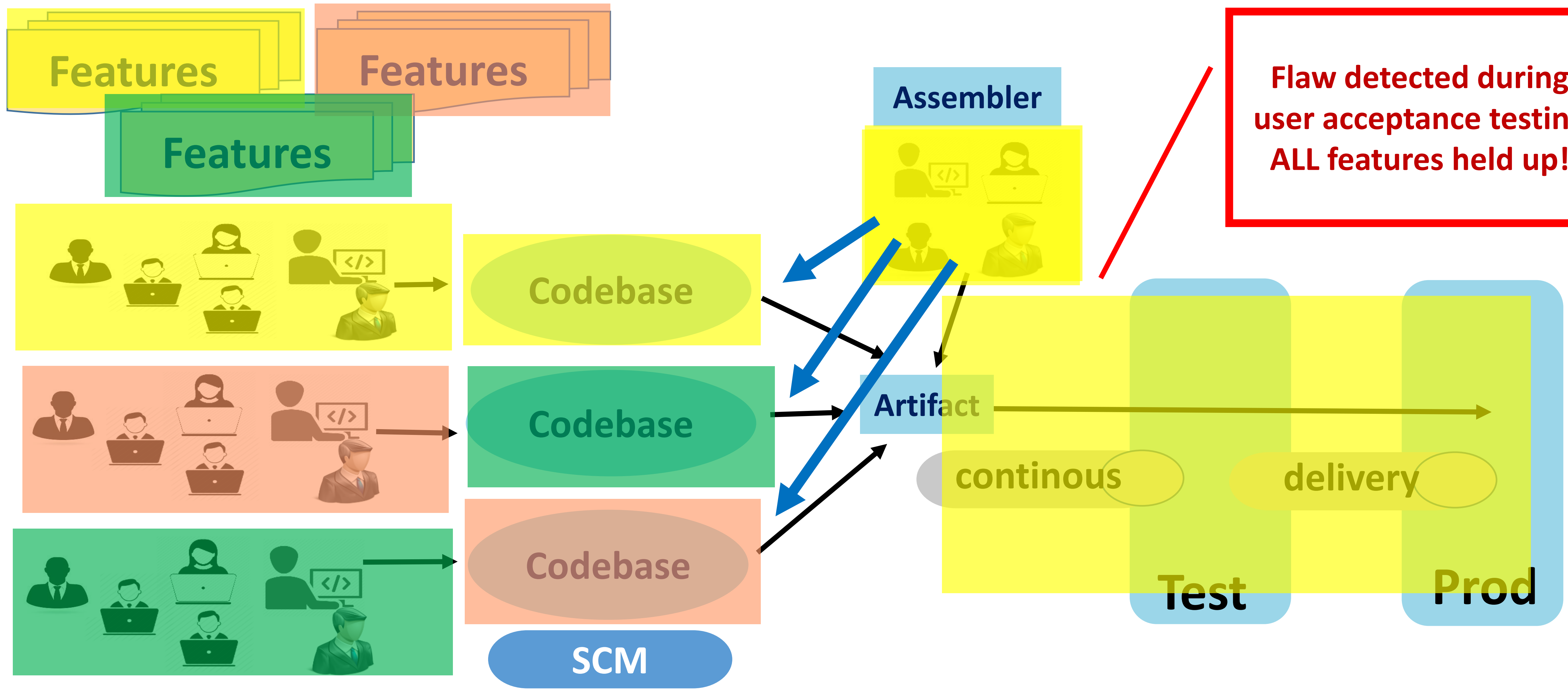
Monolith Challenges – Larger System

# Single Codebase, Deployment, Versioning, Team Size



Monolith Challenges – Larger System

# Single Codebase, Deployment, Versioning, Team Size



Flaw detected during user acceptance testing ALL features held up!

Understanding The Monolithic Application

- # Single application executable
    - # Easy to comprehend, but not to digest
    - # Must be written in a single language
- # Modularity based on Program Language
    - # Using the constructs available in that language (packages, classes, functions, namespaces, frameworks)
    - # Various storage / service technologies used
        - # RDBMS, Messaging, eMail, etc.

Design Principle For Monolith

# **DDD** = Domain Driven Design

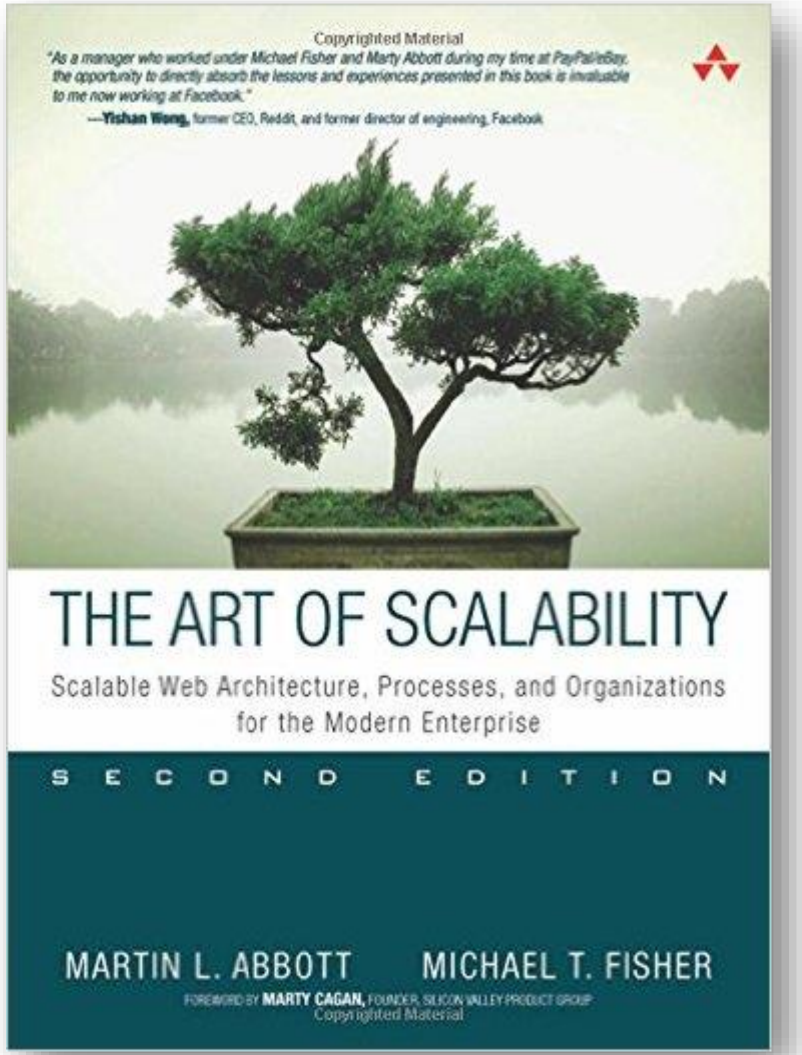
# **SoC** = Separation Of Concern Using **MVC**

# **High Cohesion, low coupling**

# **DRY** = Don’t Repeat Yourselve

# **CoC** = Convention over Configuration

# **YAGNI** = You aren't gonna need it


Copyrighted Material

"As a manager who worked under Michael Fisher and Marty Abbott during my time at PayPal/eBay, the opportunity to directly absorb the lessons and experiences presented in this book is invaluable to me now working at Facebook."

—Yishan Wong, former CEO, Reddit, and former director of engineering, Facebook

THE ART OF SCALABILITY

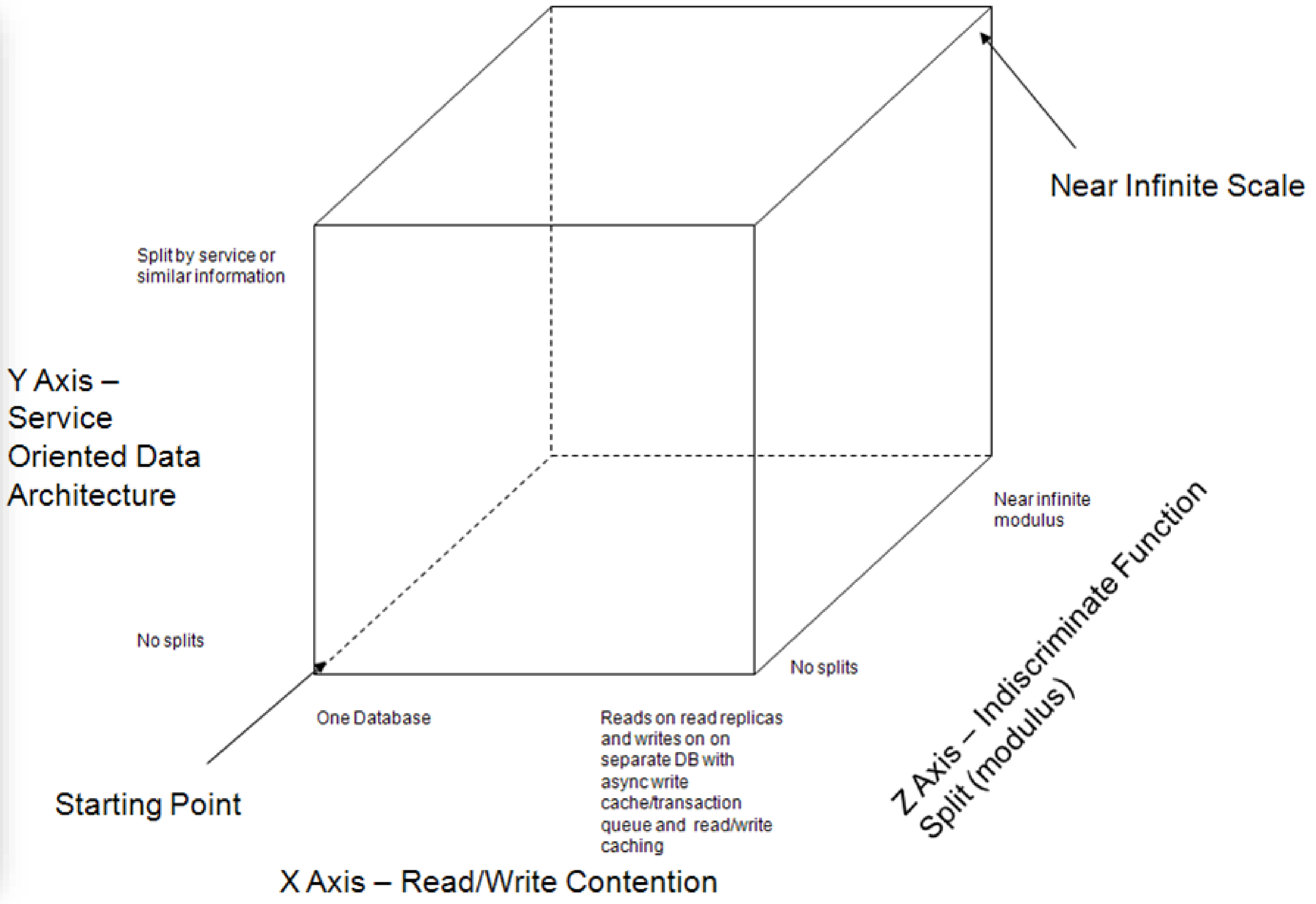
Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise

SECOND EDITION

MARTIN L. ABBOTT MICHAEL T. FISHER

FOREWORD BY MARTY CAGAN, FOUNDER, SILICON VALLEY PRODUCT GROUP

Copyrighted Material


Near Infinite Scale
Split by service or similar information
Y Axis – Service Oriented Data Architecture
No splits
Near infinite modulus
No splits
Z Axis – Indiscriminate Function Split (modulus)
One Database
Reads on read replicas and writes on on separate DB with async write cache/transaction queue and read/write caching
Starting Point
X Axis – Read/Write Contention

Software Scaling

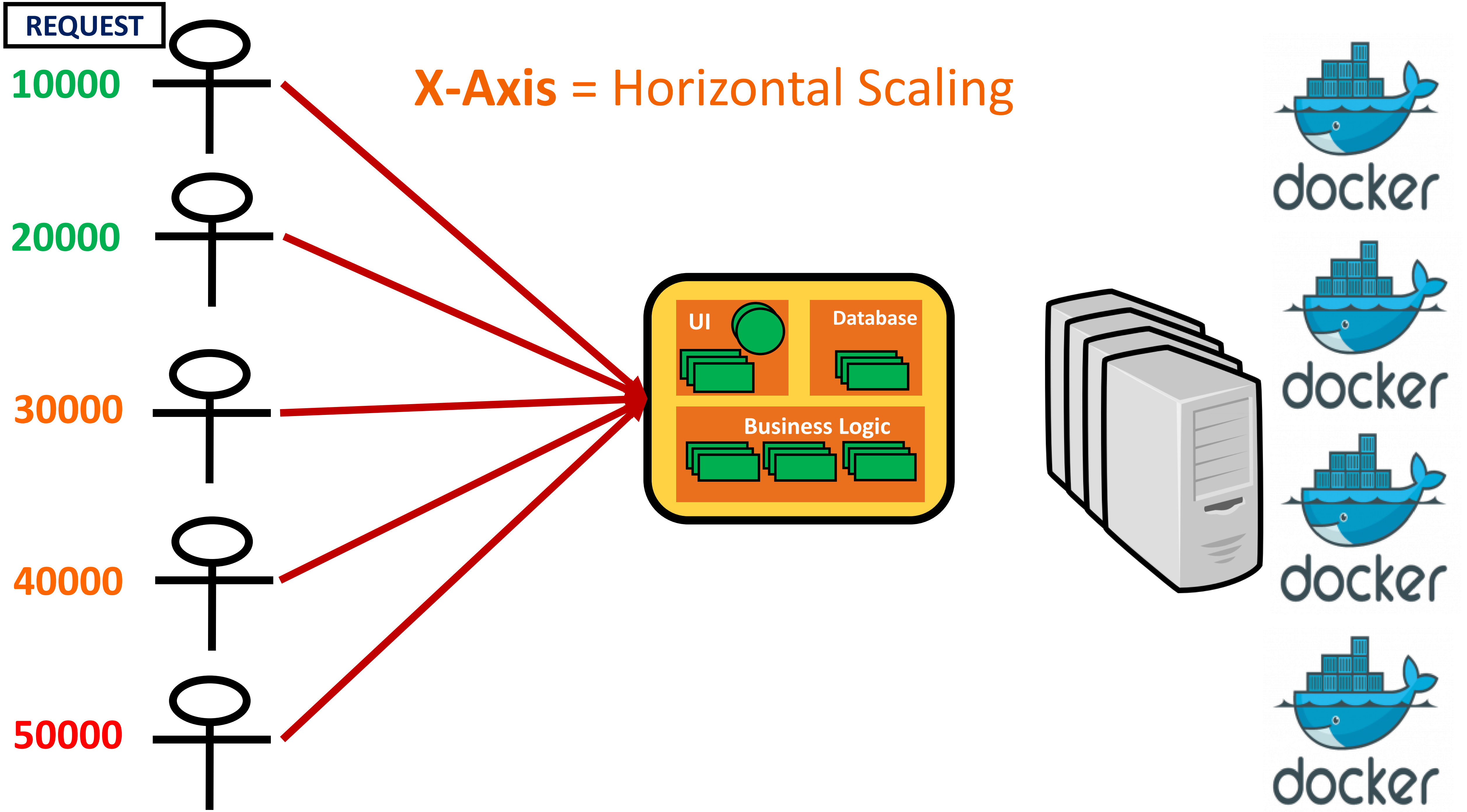
# THREE Dimension a software can typically SCALE

X-Axis = Horizontal Scaling

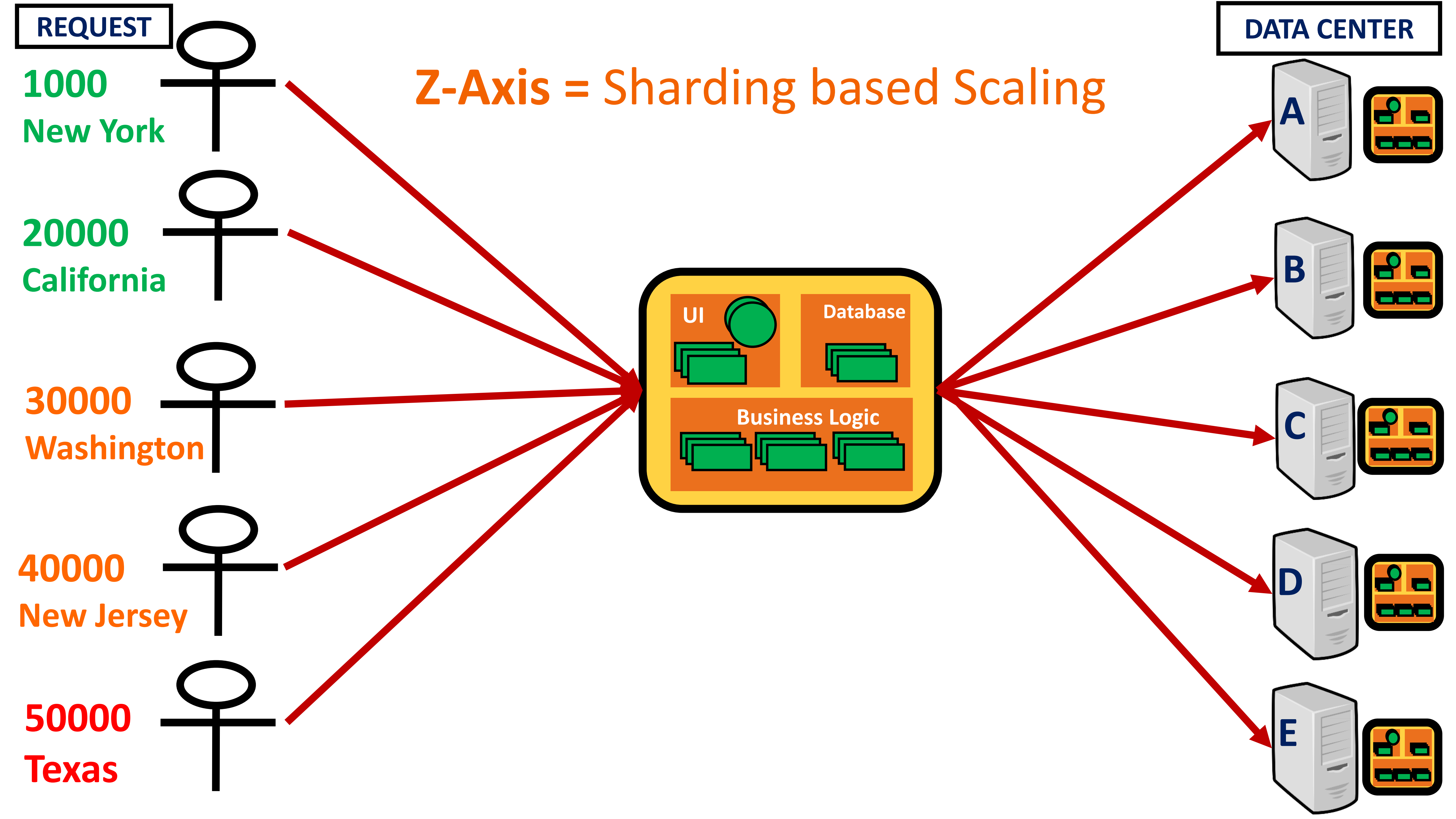
Z-Axis = Sharding based Scaling

Y-Axis = Infinite Scaling

X-Axis = Horizontal Scaling



Z-Axis = Sharding based Scaling



Advantages Of Monolith

# Typically packaged in a single **.ear**

# Easy to test (all required steps are up)

# Simple to Develop

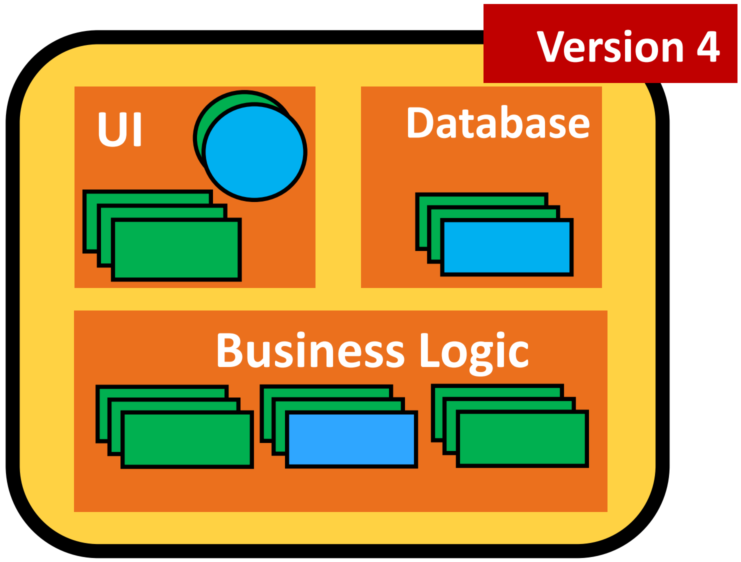
Disadvantages Of Monolith

# Difficult to deploy & maintain

# Obstacles to frequent deployment

# Dependency between unrelated features

# Makes it difficult to try out new tech/frameworks



Microservices – Working Definition

@ Composing a single application using a suite of small services

  # **(rather than a single monolith application)**

@ …each running as independent process

  # **(not merely modules / components with in a single exec)**

@ …intercommunicating via open protocols

  # **(Like HTTP/REST, or messaging AMQP)**

@ Separately written, deployed, scaled and maintained

  # **(potentially in different language)**

@ Services are independently replaceable and upgradable

## Microservices are not:

@ The same as SOA

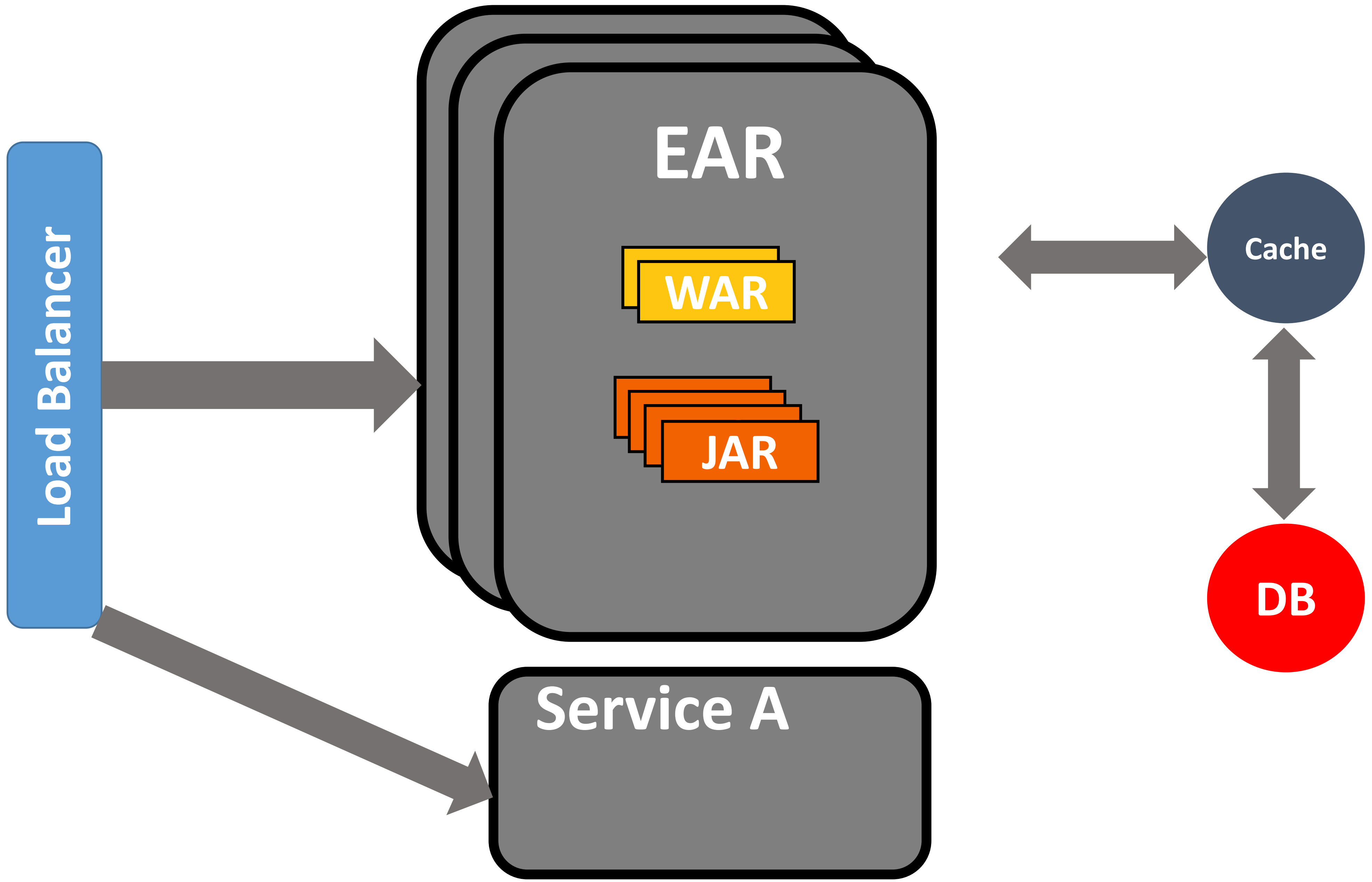
  # SOA is about integrating various enterprise application

  # Microservices are mainly about decomposing single app

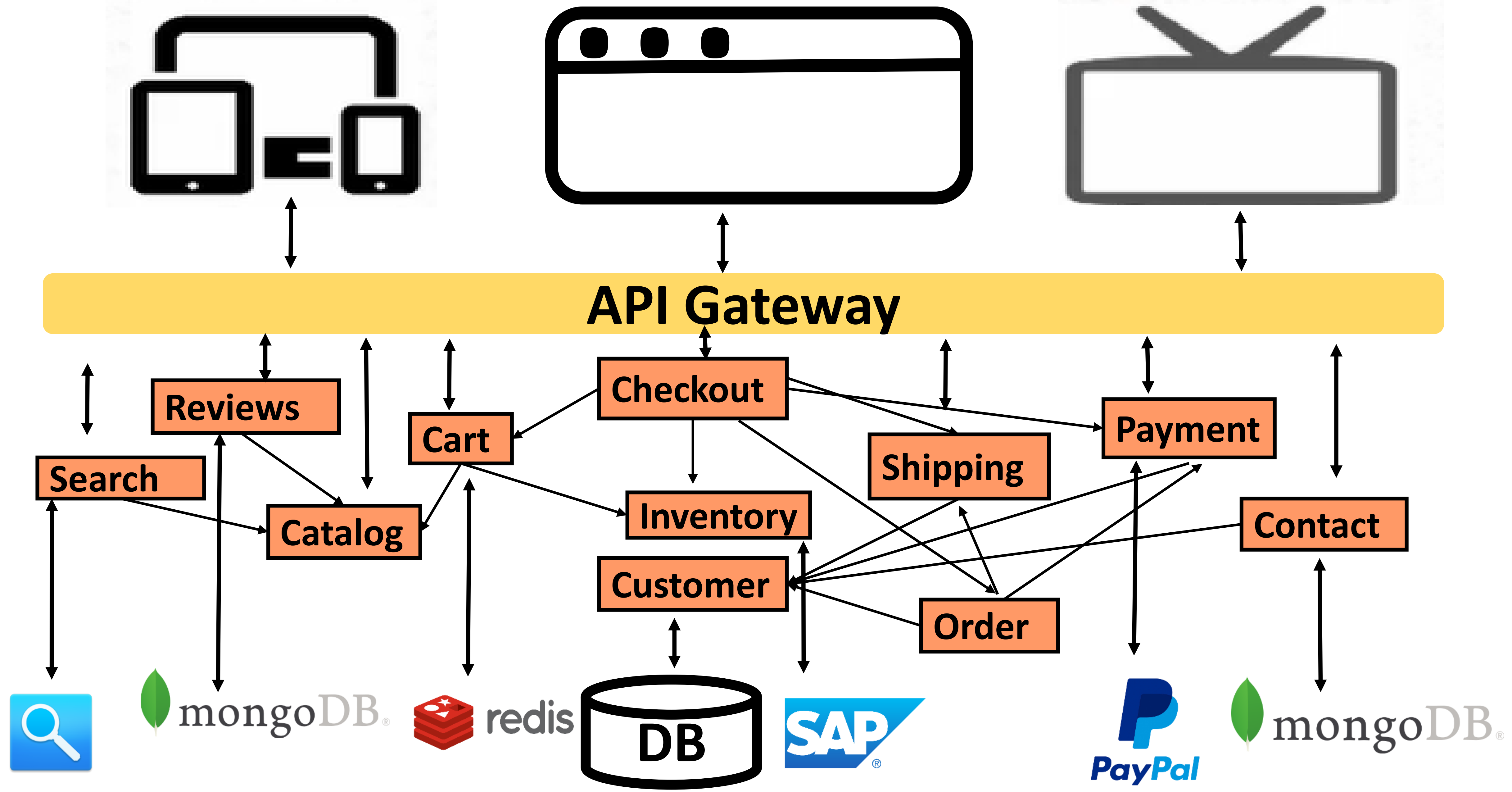
@ A Silver bullet

  # The microservices approach involves drawbacks & risks

@ New! You may be using microservices now and not Know it!


Load Balancer
EAR
WAR
JAR
Service A
Cache
DB

Enter Microservices Architecture



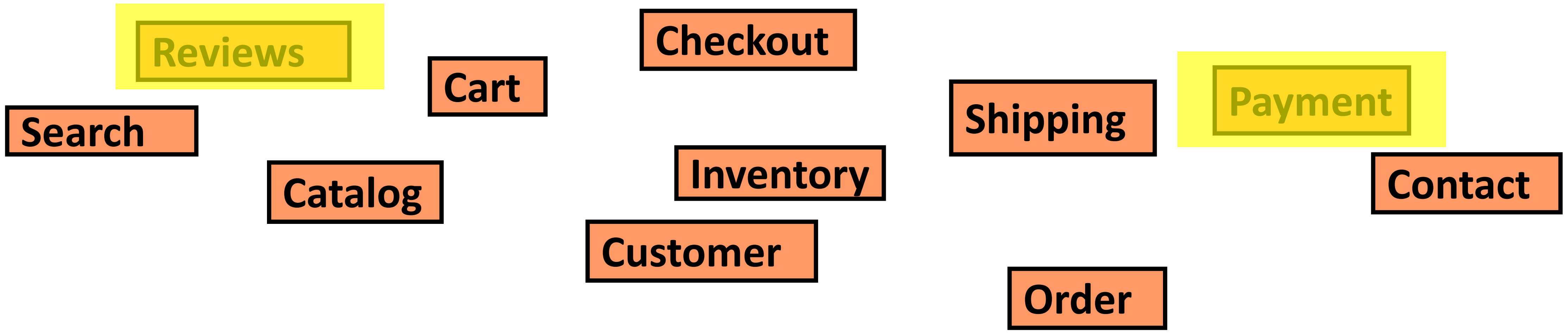
Componentization via Services

# NOT language constructs

# Where services are small, independently deployable applications

# Forces the design of clear interfaces

# Changes scoped to their affected services

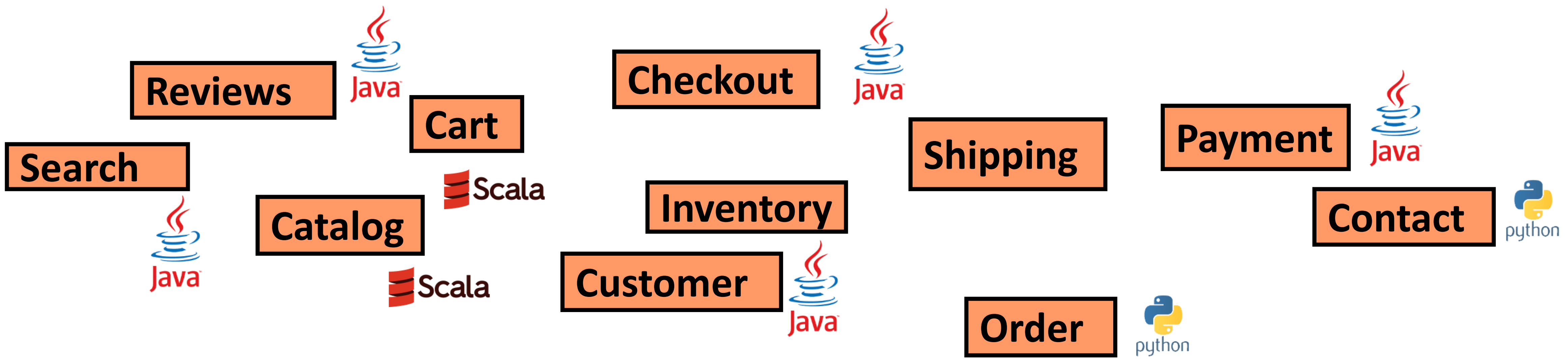


Microservices:

Composed using suite of small services

# Services are small independently deployable applications

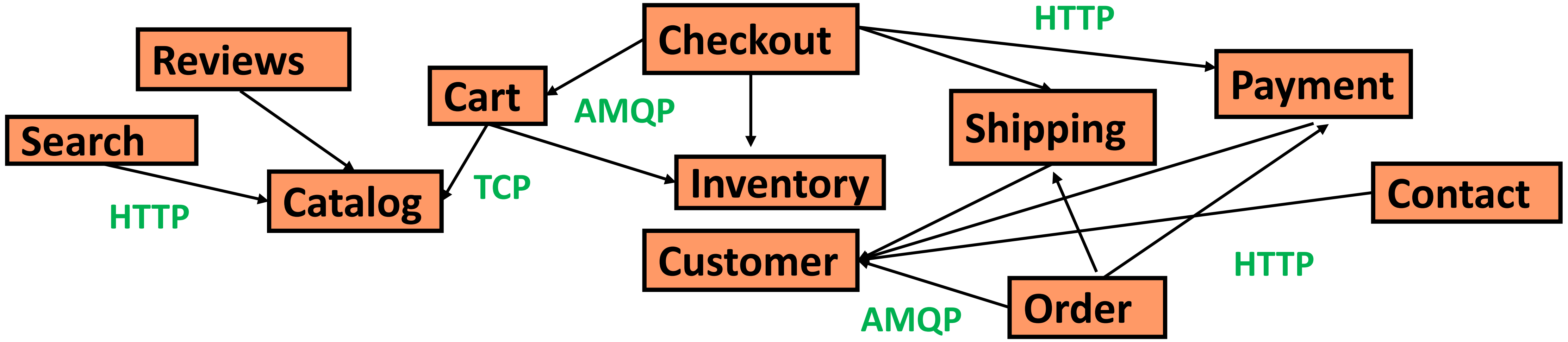
- $ Not a single codebase
- $ Not (necessarily) a single language/framework
- $ Modularization not based on language / framework cosntructs



Microservices:

Communication based on lightweight protocols

- # HTTP, TCP, UDP, Messaging, etc.
  - Payloads: JSON, BSON, XML, Protocol Buffers, etc.
- $ Forces the design of clear interfaces
- $ Netflix's cloud native architecture communicate via API's
  - Not Common Database



Microservices:

Services encapsulate business capabilities

# Not based on technology stack

# Vertical slices by business function (i.e. cart, catalog, checkout)

# ….Though technology chunk also practical (email service)

# Suitable for cross-functional teams

Search

PUT /search

Reviews

GET /review/123
POST /review

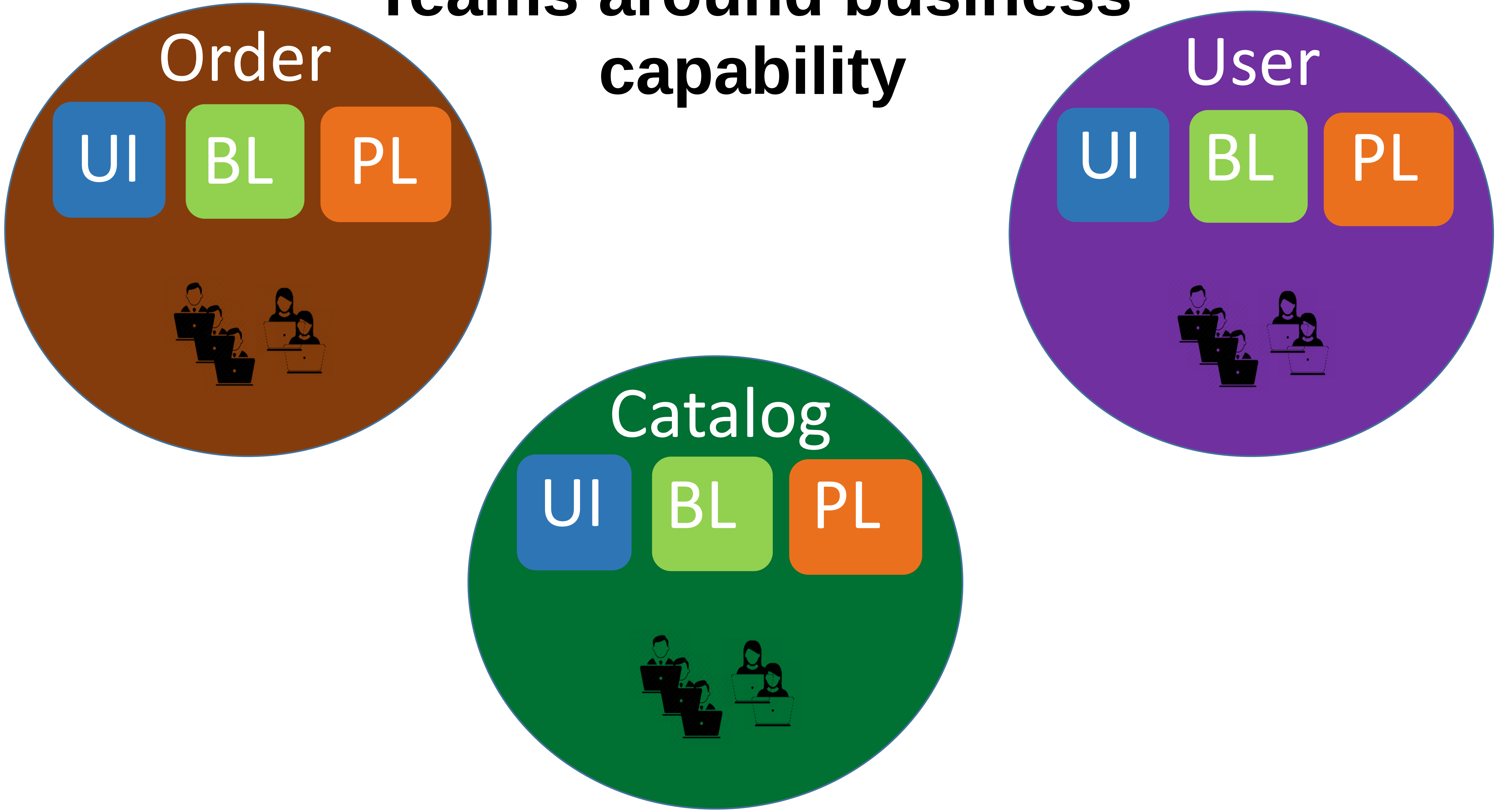
Cart

POST /cart
GET /cart/123
POST /cart/123/item
DELETE /cart/123
PUT /cart/123/item/1
DELETE /cart/123/item/1

Contact

GET /post/123
POST /post

Teams around business capability

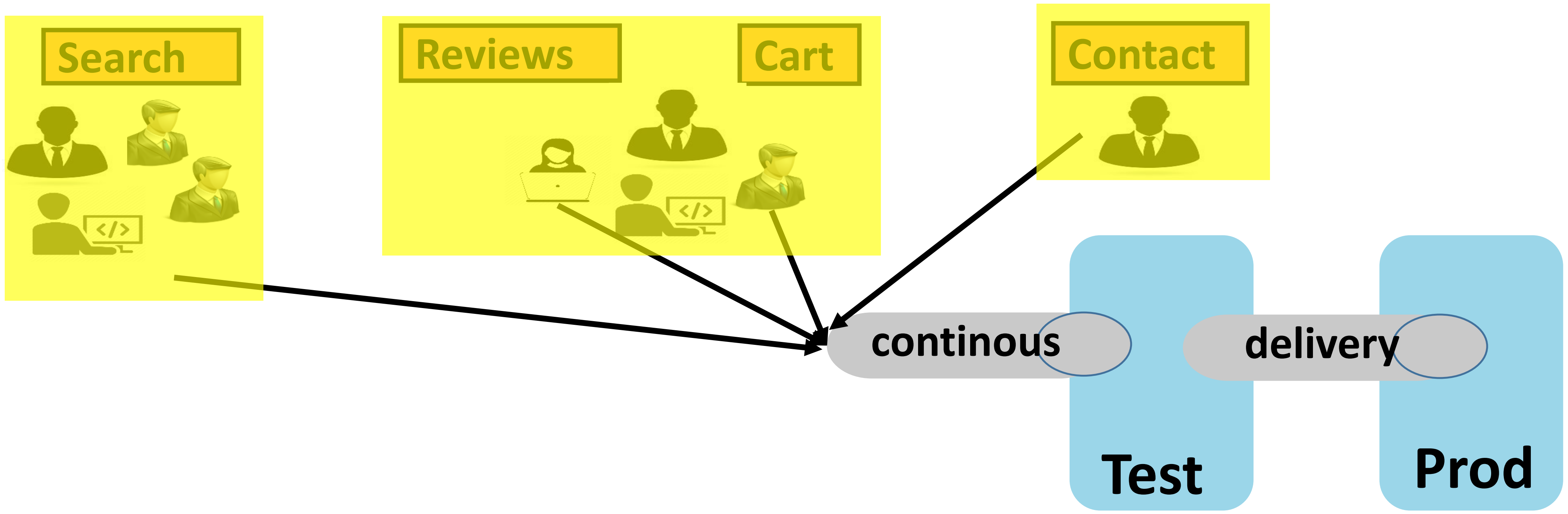


Microservices:

Services easily managed

# Easy to comprehend, alter, test, version, deploy, manage, overhaul, replace

# By small, cross-functional teams (or even individuals)



Microservices Advantages

# Easy to digest each services (difficult to comprehend whole)

# VERY easy to test, deploy, manage, version and scale single services

# Change cycle decoupled

# Easier to scale staff

# No Language/ Framework lock.

Advantages of Microservices

# Easier to develop, understand and 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

How do you Break a Monolith into Microservices?

# Primary Consideration: Business Functionality

- **Noun-based** (catalog, cart, customer)

- **Verb-based** (search, checkout, shipping)

- **Single** responsibility principle

  $ http://programmer.97things.oreilly.com/wiki/index.php/The_Single_Responsibility_Principle

- **Bounded** Context

  $ http://martinfowler.com/bliki/BoundedContext.html

How Micro is Micro?

# Size is not a compelling factor

- Small enough for an individual developer to digest
- Small enough to be built and managed by small team

  Amazon’s two pizza rule
- Documentation small enough to understand
- Dozens of Secret, not hundreds
- Predictable. Easy to experiment with

Differences with SOA

# **SOA** addresses integration between systems

- **Microservices** address individual application

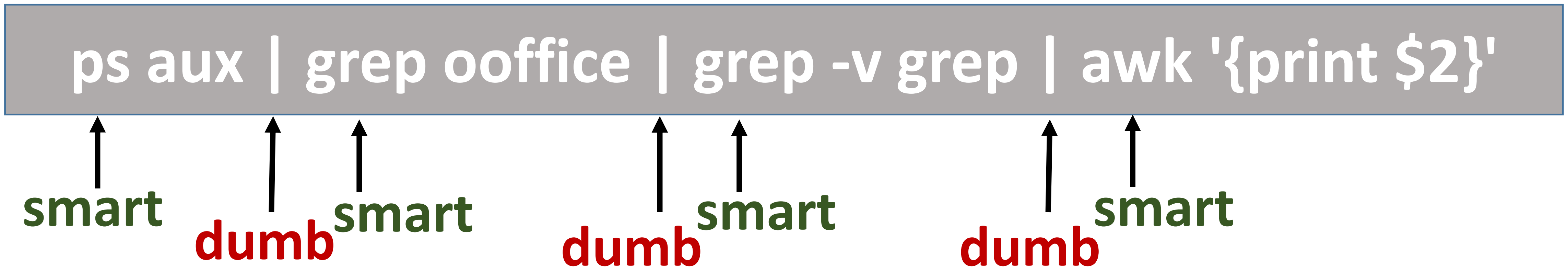
# **SOA** relies on orchestration

- **Microservices** rely on choreography

# **SOA** relies on smart integration technology, dumb services

- **Microservices** rely on smart services, dumb integration technology

- **Consider:** Linux commands, pipes and filters

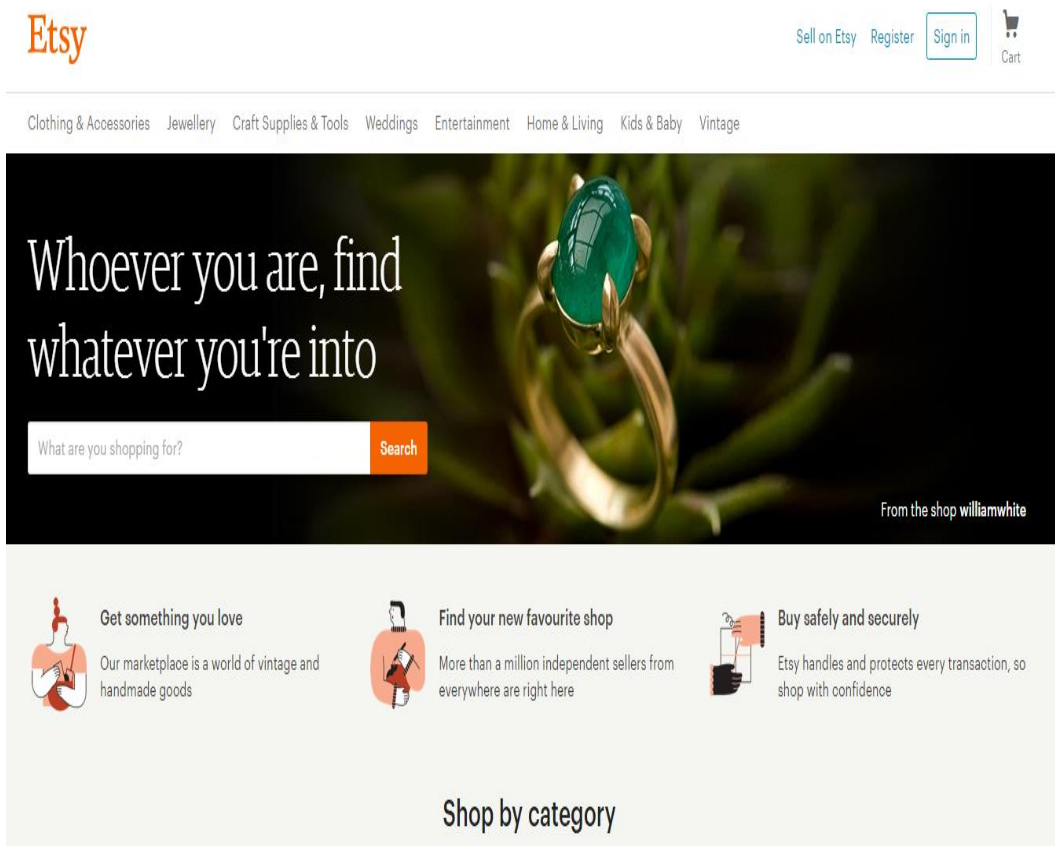


```
ps aux | grep ooffice | grep -v grep | awk '{print $2}'
```

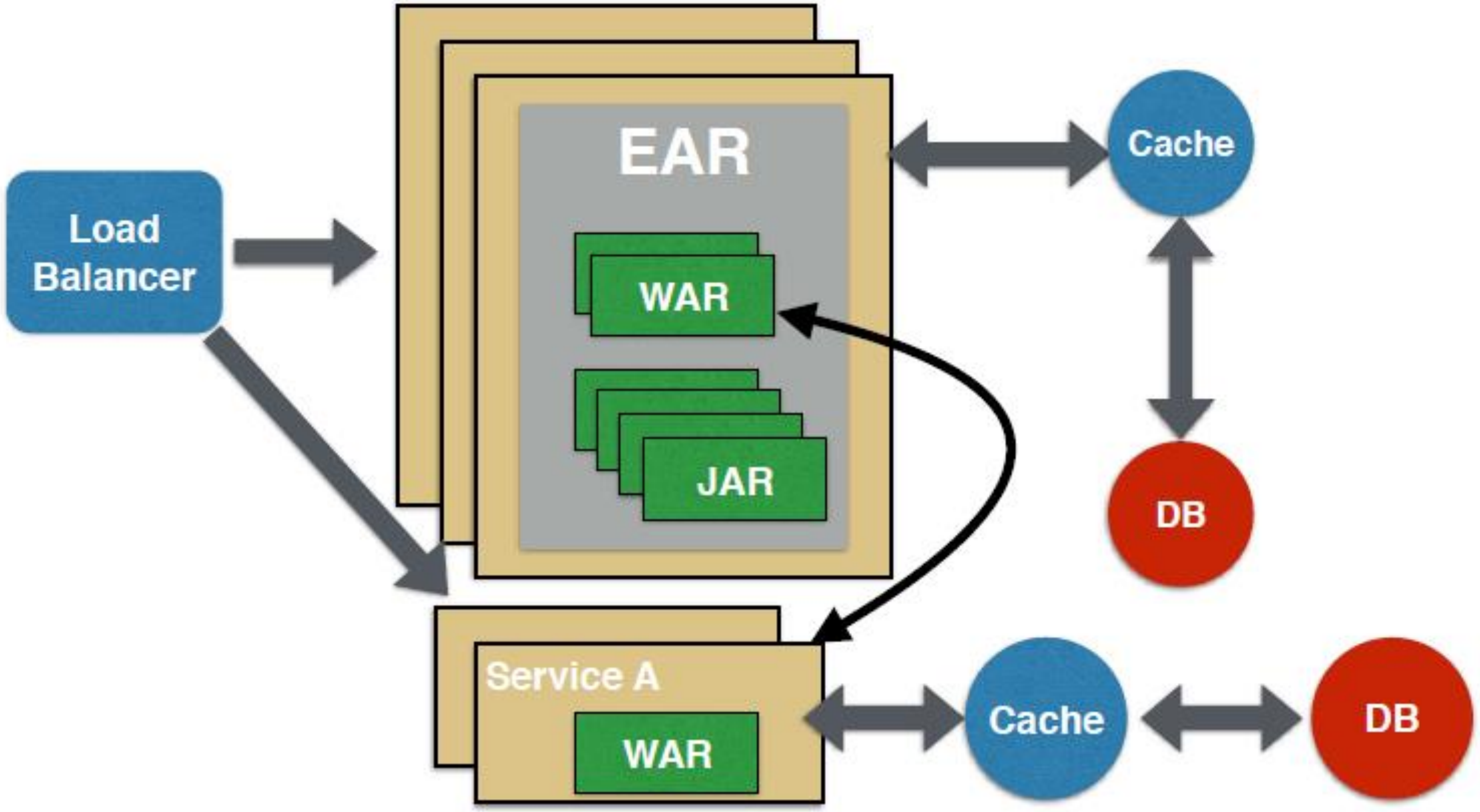
Are Monoliths Always BAD?

# Consider etsy.com

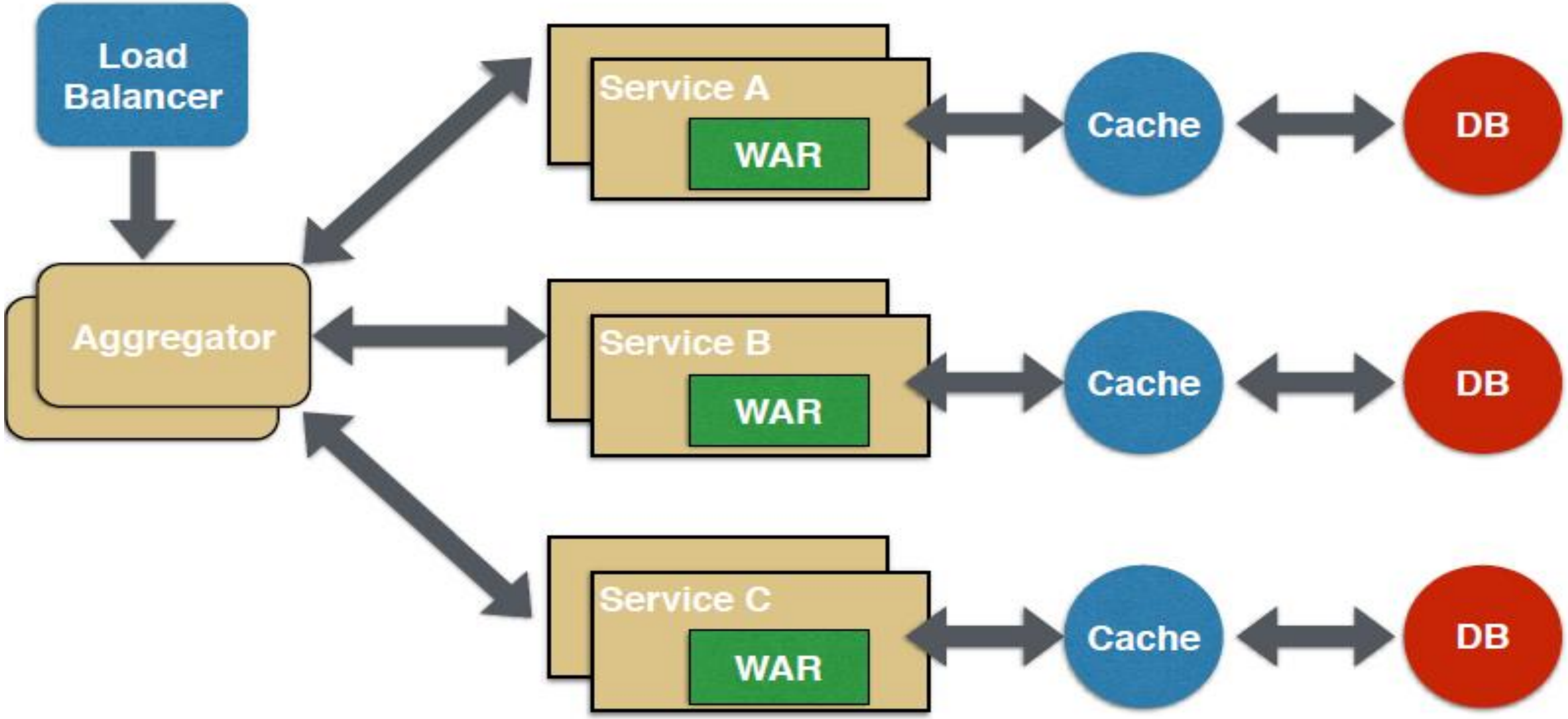
- As of February 2013: 1.49 billion page views, 4,215,169 items sold, $94.7 million goods sold, 22+ million members
- 150 developers deploy single WAR 60 times a day
- Practices: CI; push button deployment; good monitoring; developers deploy to the site on first day; VMs per developer; GitHub; Chef; IRC to control releases; dashboards; no source control branches


Etsy
Sell on Etsy
Register
Sign in
Cart
Clothing & Accessories
Jewellery
Craft Supplies & Tools
Weddings
Entertainment
Home & Living
Kids & Baby
Vintage
Whoever you are, find whatever you're into
What are you shopping for?
Search
From the shop williamwhite
Get something you love
Our marketplace is a world of vintage and handmade goods
Find your new favourite shop
More than a million independent sellers from everywhere are right here
Buy safely and securely
Etsy handles and protects every transaction, so shop with confidence
Shop by category

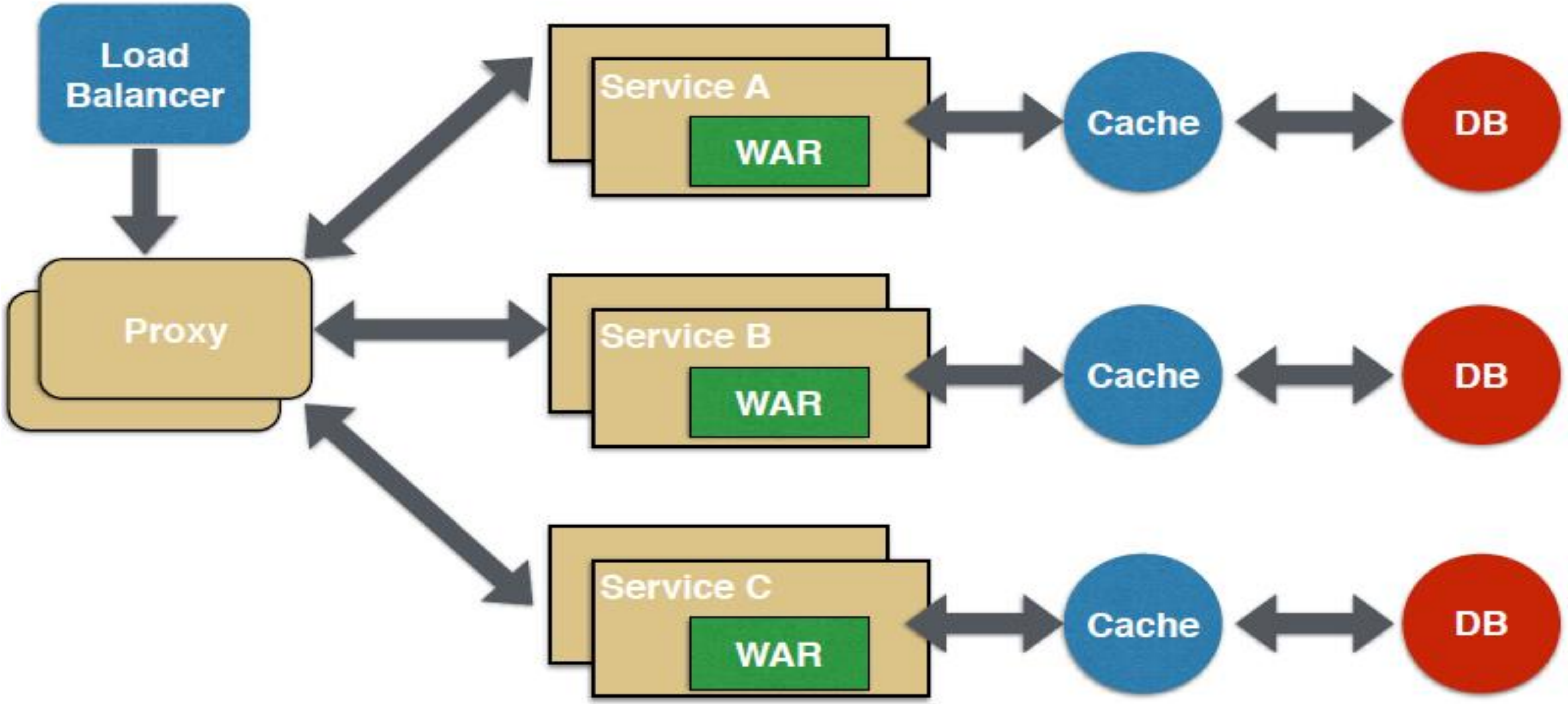
Towards microservices



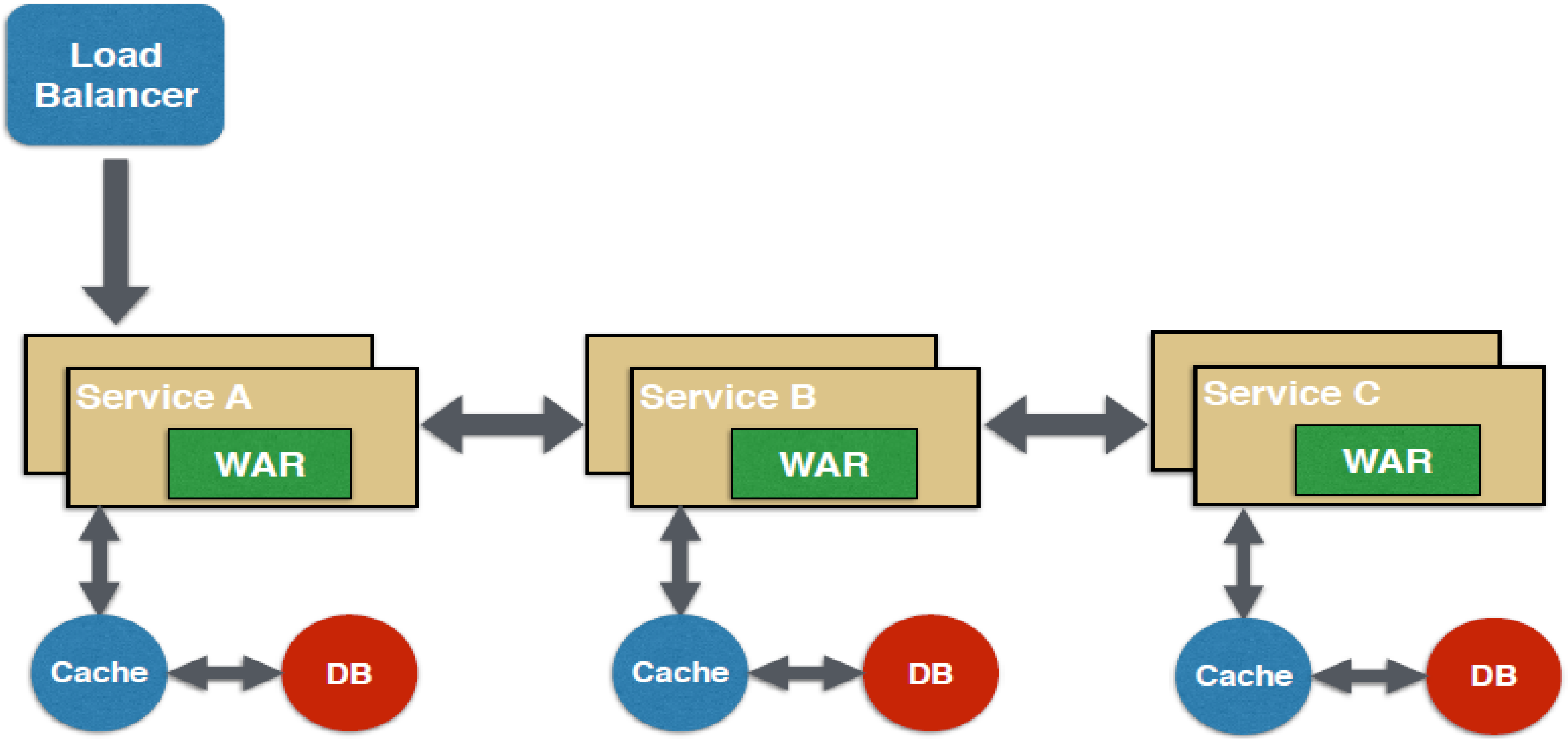
Aggregator Pattern #1



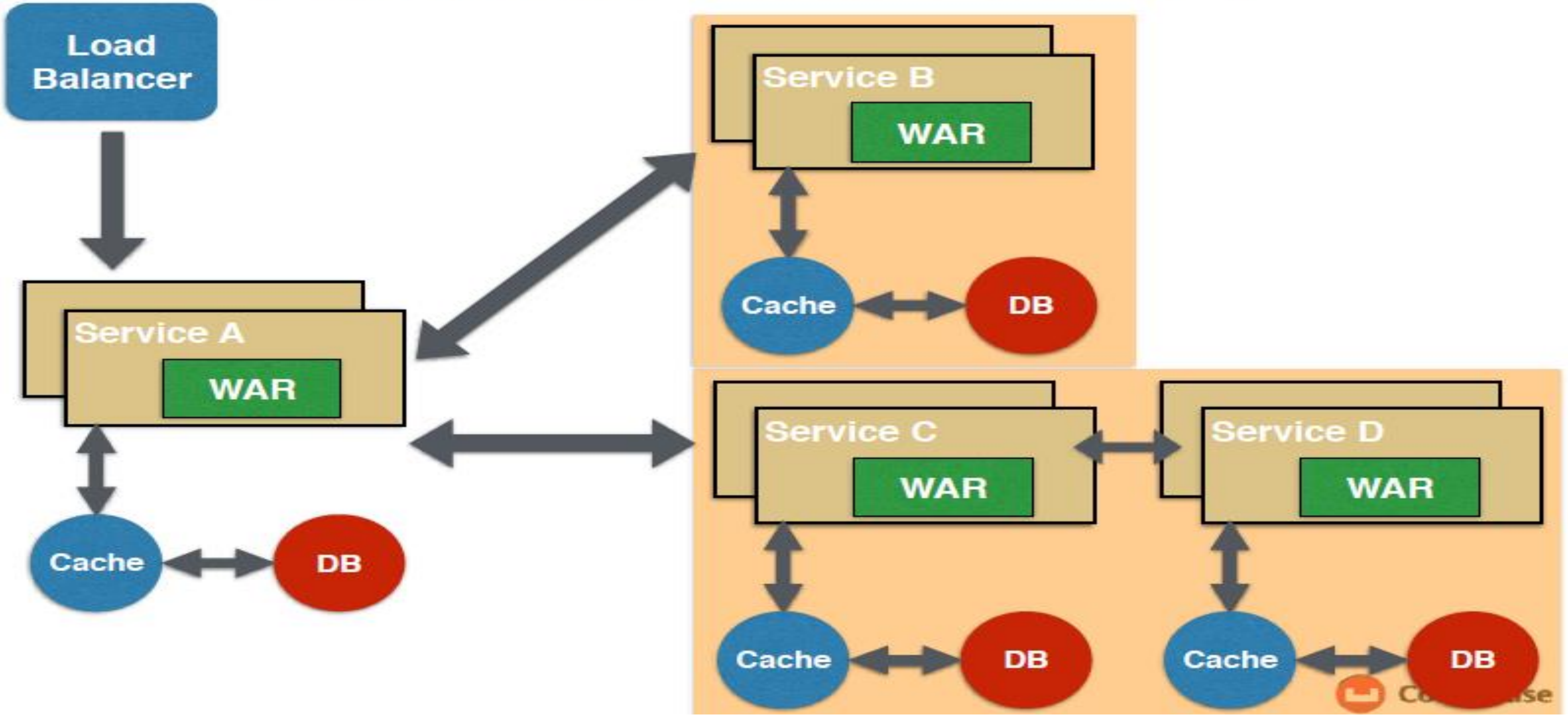
Proxy Pattern #2



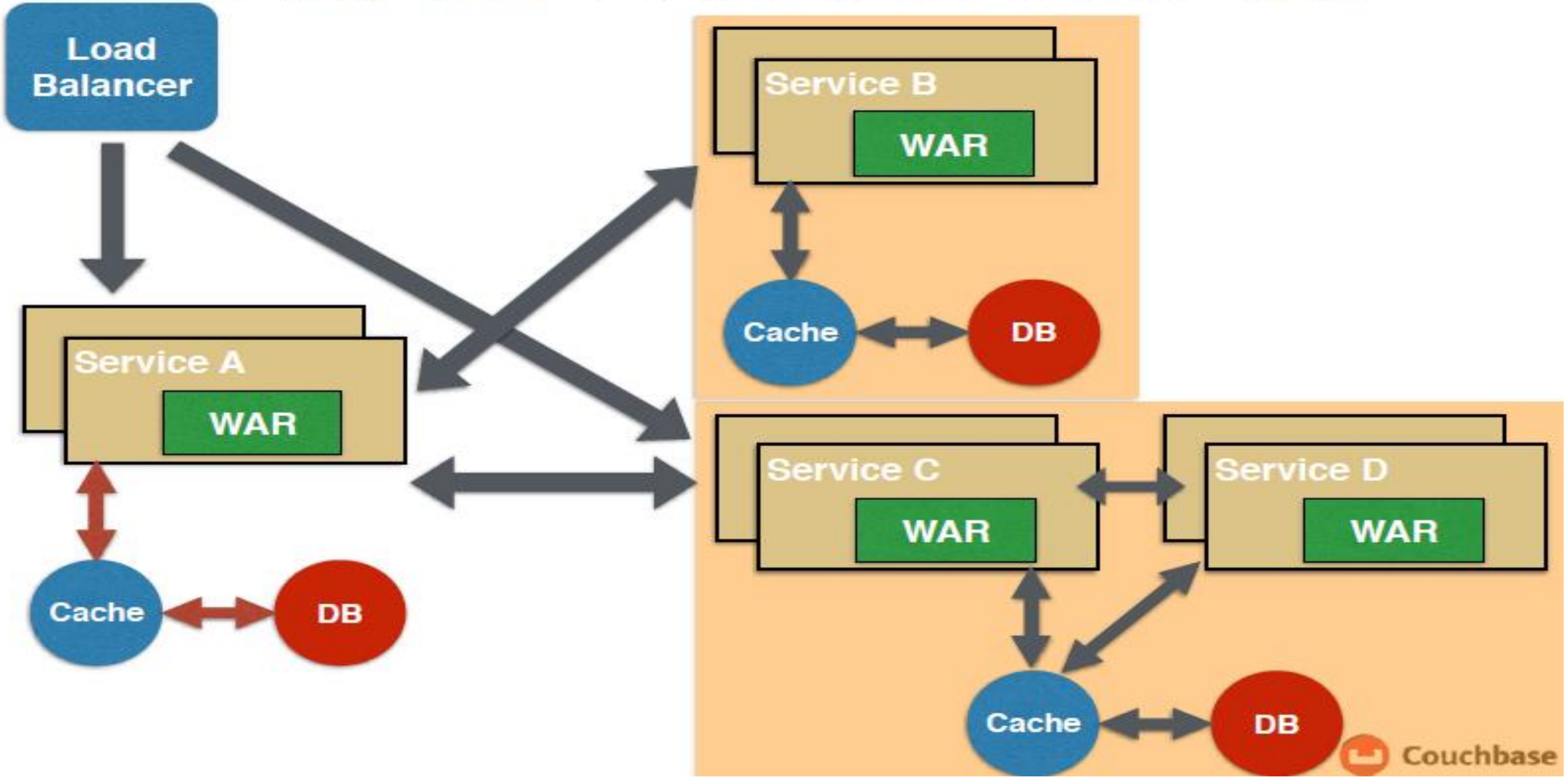
Chained Pattern #3



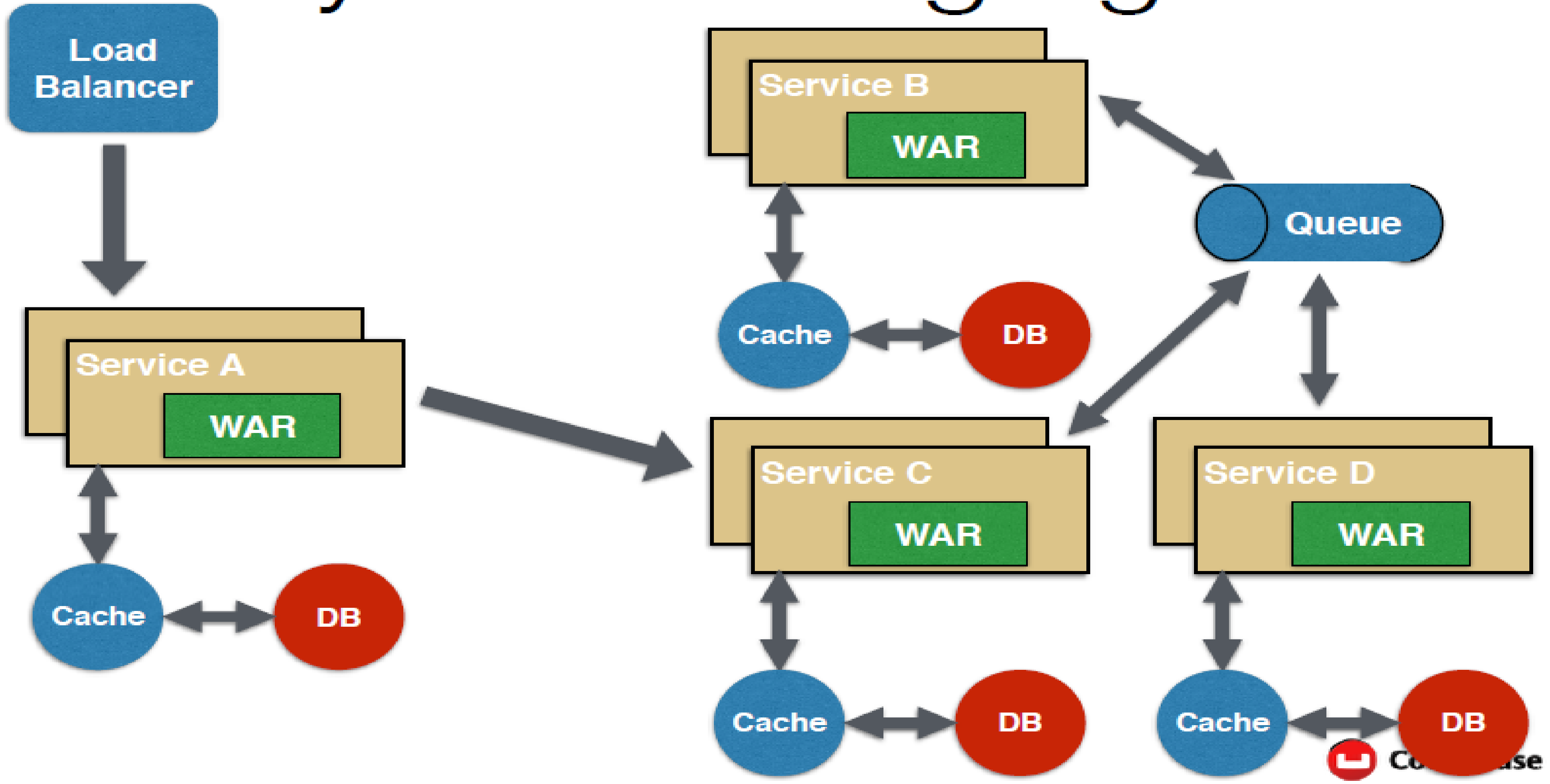
Branch Pattern #4



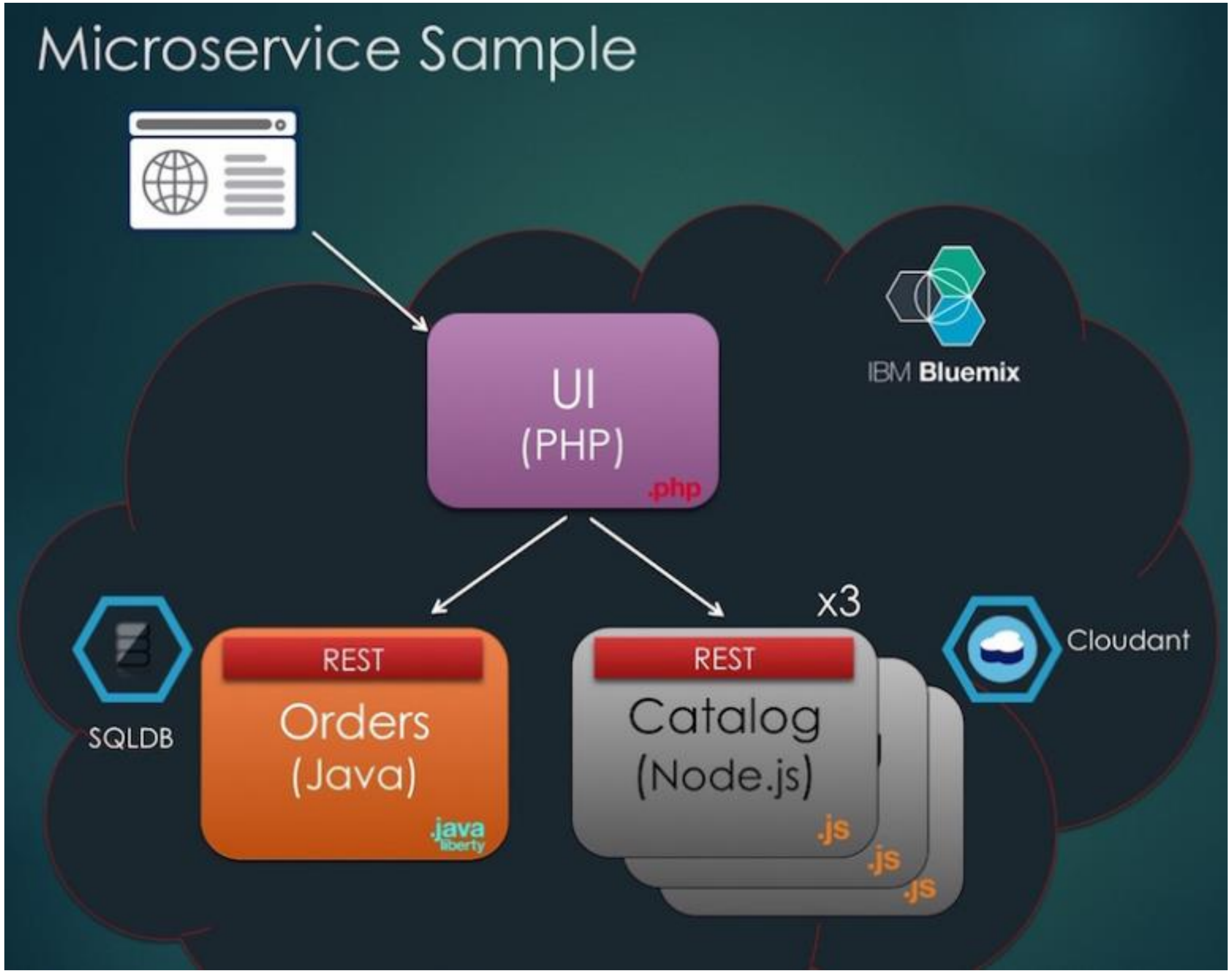
Shared Resources #5



Async Messaging #5



Simple microservices online e-commerce sample



Simple microservices online e-commerce sample

What you will need ?

# A Bluemix account

# Cloud Foundry Command Line Interface

#1 GB of memory in your Bluemix Dashboard.

# Space for two services in your Bluemix Dashboard.

Simple microservices online e-commerce sample

Applications Overview

Application 1: Catalog API – A backed RESTful API to keep track of all the items in the store. We will use Node.js with Express framework. The catalog of items will be persisted in a Cloudant database.

Simple microservices online e-commerce sample

Applications Overview

Application 2: Orders API – Another backend RESTful API to keep track of all store orders. We will implement this using Java JAX-RS and use JPA to store the orders in a SQL Database.

Simple microservices online e-commerce sample

Applications Overview

Application 1 – Catalog API

Simple microservices online e-commerce sample

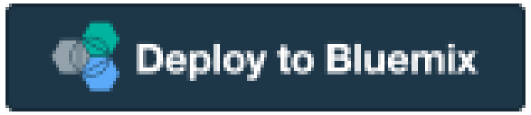
Application 3: UI – A simple UI that displays all the items in the store catalog, and can create orders. This UI will call both REST APIs provided by the applications above. We will use PHP to write this part.

Simple microservices online e-commerce sample

Follow the steps below to deploy your Node.js API back-end.

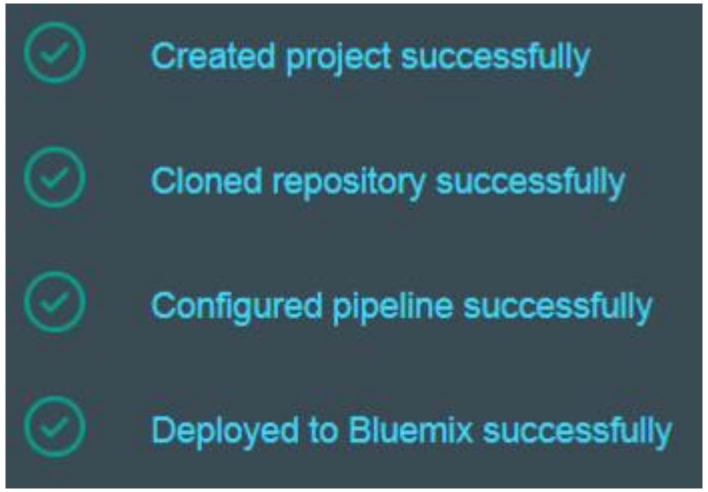
1. Sign up or log into Bluemix

2. Click **Deploy to Bluemix** button below:



Deploy to Bluemix

3. All fields on the next page will be pre-filled for you. Select **Deploy**.
4. Once the application is done deploying to Bluemix, select View your App
   (keep your app open as you will need the route for application 3, not the git url)



✓ Created project successfully

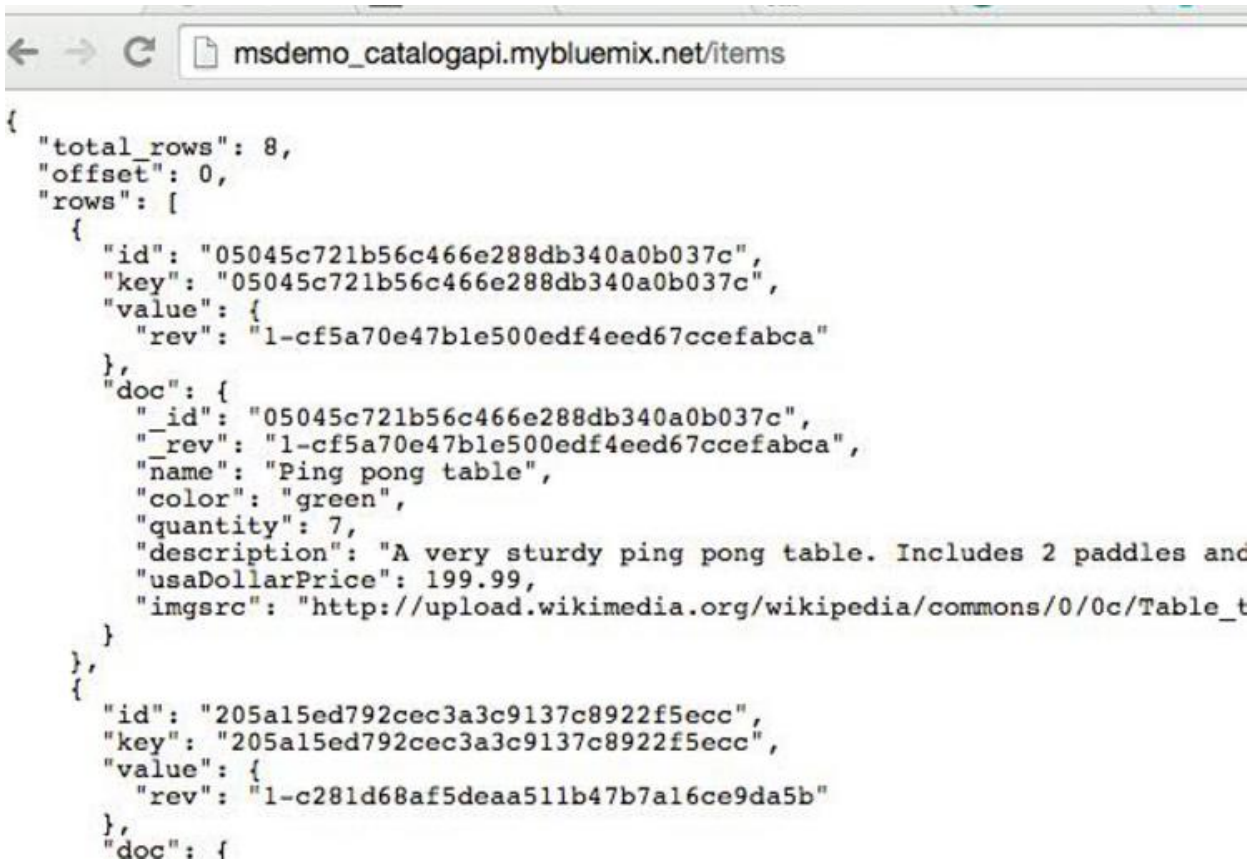
✓ Cloned repository successfully

✓ Configured pipeline successfully

✓ Deployed to Bluemix successfully

Simple microservices online e-commerce sample

5. To see the items in your catalog follow the instructions on the homepage. Note that the format will be in JSON. We will make a UI application later, which can display these items in a more user-friendly way.



msdemo_catalogapi.mybluemix.net/items

```
{
  "total_rows": 8,
  "offset": 0,
  "rows": [
    {
      "id": "05045c721b56c466e288db340a0b037c",
      "key": "05045c721b56c466e288db340a0b037c",
      "value": {
        "rev": "1-cf5a70e47b1e500edf4eed67ccefabca"
      },
      "doc": {
        "_id": "05045c721b56c466e288db340a0b037c",
        "_rev": "1-cf5a70e47b1e500edf4eed67ccefabca",
        "name": "Ping pong table",
        "color": "green",
        "quantity": 7,
        "description": "A very sturdy ping pong table. Includes 2 paddles and
        "usaDollarPrice": 199.99,
        "imgsrc": "http://upload.wikimedia.org/wikipedia/commons/0/0c/Table_t
      }
    },
    {
      "id": "205a15ed792cec3a3c9137c8922f5ecc",
      "key": "205a15ed792cec3a3c9137c8922f5ecc",
      "value": {
        "rev": "1-c281d68af5deaa511b47b7a16ce9da5b"
      },
      "doc": {
```

Simple microservices online e-commerce sample

Applications Overview

Application 2 – Orders API

Simple microservices online e-commerce sample

Follow the steps below to create the Java Orders back-end.

Due to the nature of this application handling customer orders and potential billing information, failures are not acceptable for this service and security is our utmost importance. We have selected to use **Java EE** because this application needs a technology stack that is robust, well supported, standards driven, and can provide transactional support.

The WebSphere Application Server Liberty buildpack on Bluemix offers us an enterprise-grade Java EE application server while still being extremely light weight and dynamic.

The jax-rs feature is provided by default and allows us to quickly create RESTful end points with simple annotations.

Simple microservices online e-commerce sample

Click **Deploy to Bluemix** button below:

All fields on the next page will be pre filled for you. Select **Deploy**.

Once the application is done deploying to Bluemix, select **View your App** (keep this tab open as you will need the route for application 3, not the git URL).

Simple microservices online e-commerce sample

Applications Overview

Application 3 – UI

Simple microservices online e-commerce sample

This application requires 2 user-provided services. One for you Catalog API and one for your Orders API. Start by opening a terminal and type: **cf login.**

Enter your Email and password. These are associated with your Bluemix Account.

Next, paste this command in your terminal to create our first user-provided service for the Catalog API: cf cups catalogAPI -p "host" and hit enter. This will prompt you for the host. This is the URL for your Catalog API application we just deployed. For example, http://ms-catalogAPI-0123abc.mybluemix.net

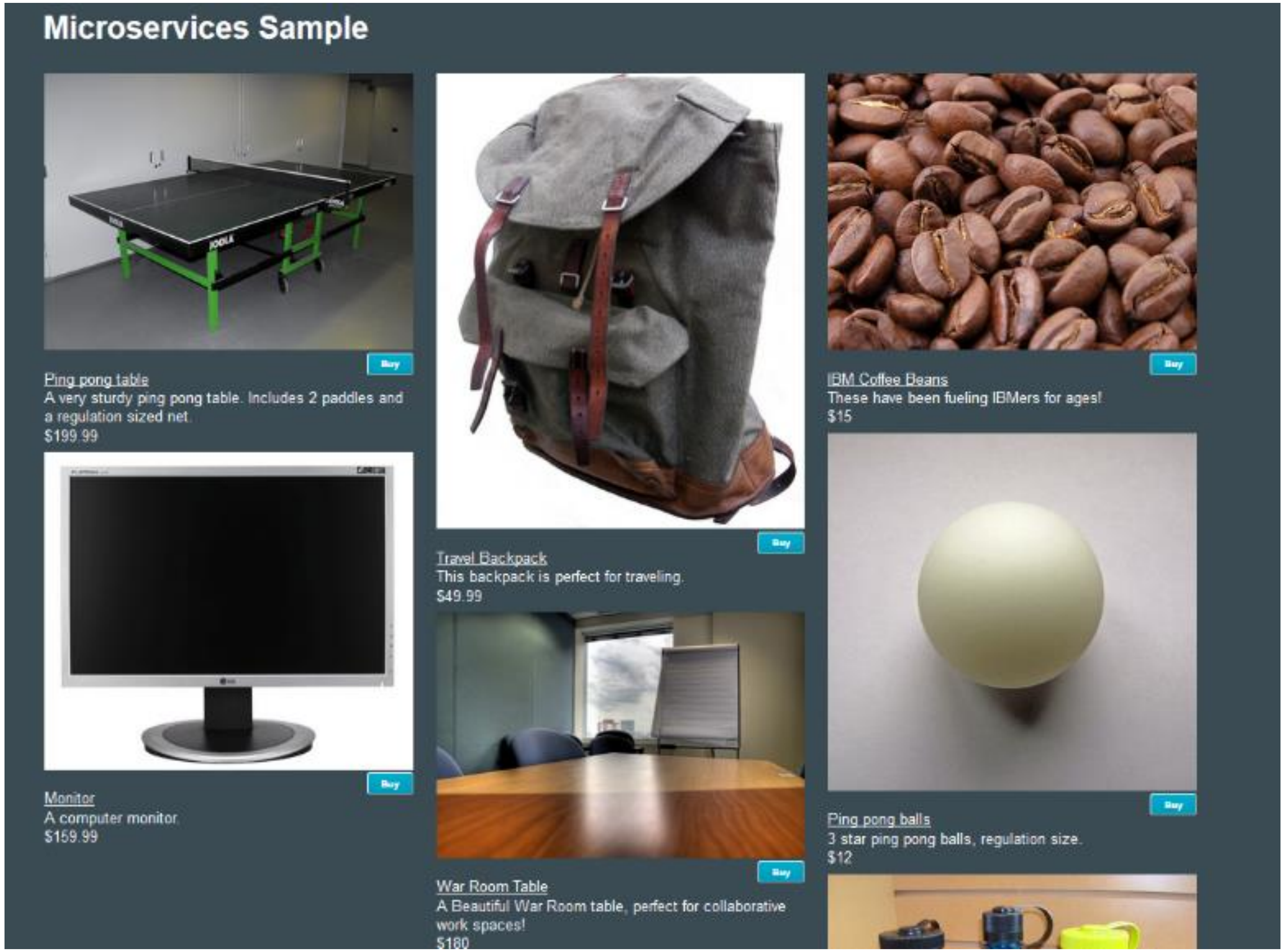
We are going to do the same as the step above, but for our Orders API. Go ahead and paste this command into your terminal cf cups ordersAPI -p "host" and hit enter. When it prompts you for the host, enter the URL for your Orders API application. For example, http://ms-ordersAPI-abc123.mybluemix.net

Now that we have our services created, it's time to Deploy to Bluemix by selecting the button below:

Deploy to Bluemix

Once your application has finished deploying, click the View Your App button to see your PHP UI communicating with your Node.js Catalog API and your Java Orders API!

Simple microservices online e-commerce sample



Microservices Sample

Buy

Ping pong table
A very sturdy ping pong table. Includes 2 paddles and a regulation sized net.
$199.99

Buy

Travel Backpack
This backpack is perfect for traveling.
$49.99

Buy

IBM Coffee Beans
These have been fueling IBMers for ages!
$15

Buy

Monitor
A computer monitor.
$159.99

Buy

War Room Table
A Beautiful War Room table, perfect for collaborative work spaces!
$180

Buy

Ping pong balls
3 star ping pong balls, regulation size.
$12

Simple microservices online e-commerce sample

Congrats! You now have a full microservices sample consisting of a Node.js Catalog back-end, a Java Orders back-end, and a PHP UI.