

Microservices Architectures

Sébastien Mosser
01.10.2018

From Monolith to Microservices:

How to Scale Your Architecture



A pattern language for microservices

Chris Richardson

Founder of Eventuate.io

Founder of the original CloudFoundry.com

Author of POJOs in Action

➤ @crichardson

chris@chrisrichardson.net

<http://microservices.io>

<http://eventuate.io>

<http://plainoldobjects.com>

Developing applications with a microservice architecture

Chris Richardson

Author of POJOs in Action

Founder of the original CloudFoundry.com

➤ @crichardson

chris@chrisrichardson.net

<http://plainoldobjects.com>



ARAF KARSH HAMID
Co-Founder / CTO
MetaMagic Global Inc., NJ, USA

@arafkarsh
 arafkarsh

$$\sum_{b=1}^n f(b) = \sqrt[3]{desi^3r^2e} \text{ 3D}$$



Microservices Architecture

World Agile Testing & Automation
Novotel Techpark, Bangalore, June 22, 2018

<https://1point21gws.com/testingsummit/bangalore/>



ARAF KARSH HAMID
Co-Founder / CTO
MetaMagic Global Inc., NJ, USA

@arafkarsh
 arafkarsh

Domain Driven Design

Micro Services Architecture

Part 1 : Infrastructure Comparison &
Design Styles (DDD, Event Sourcing / CQRS, Functional Reactive Programming)
Araf Karsh Hamid – Co Founder / CTO, MetaMagic Global Inc., NJ, USA

A **Micro Service** will have its own Code Pipeline for build and deployment functionalities and it's scope will be defined by the Bounded Context focusing on the Business Capabilities and the interoperability between Micro Services will be achieved using message based communication.





The AirBnB Case Study

“Monorail”

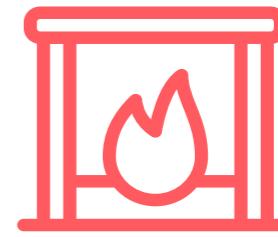
OUR MONOLITHIC RUBY ON RAILS APPLICATION



1 GitHub repo

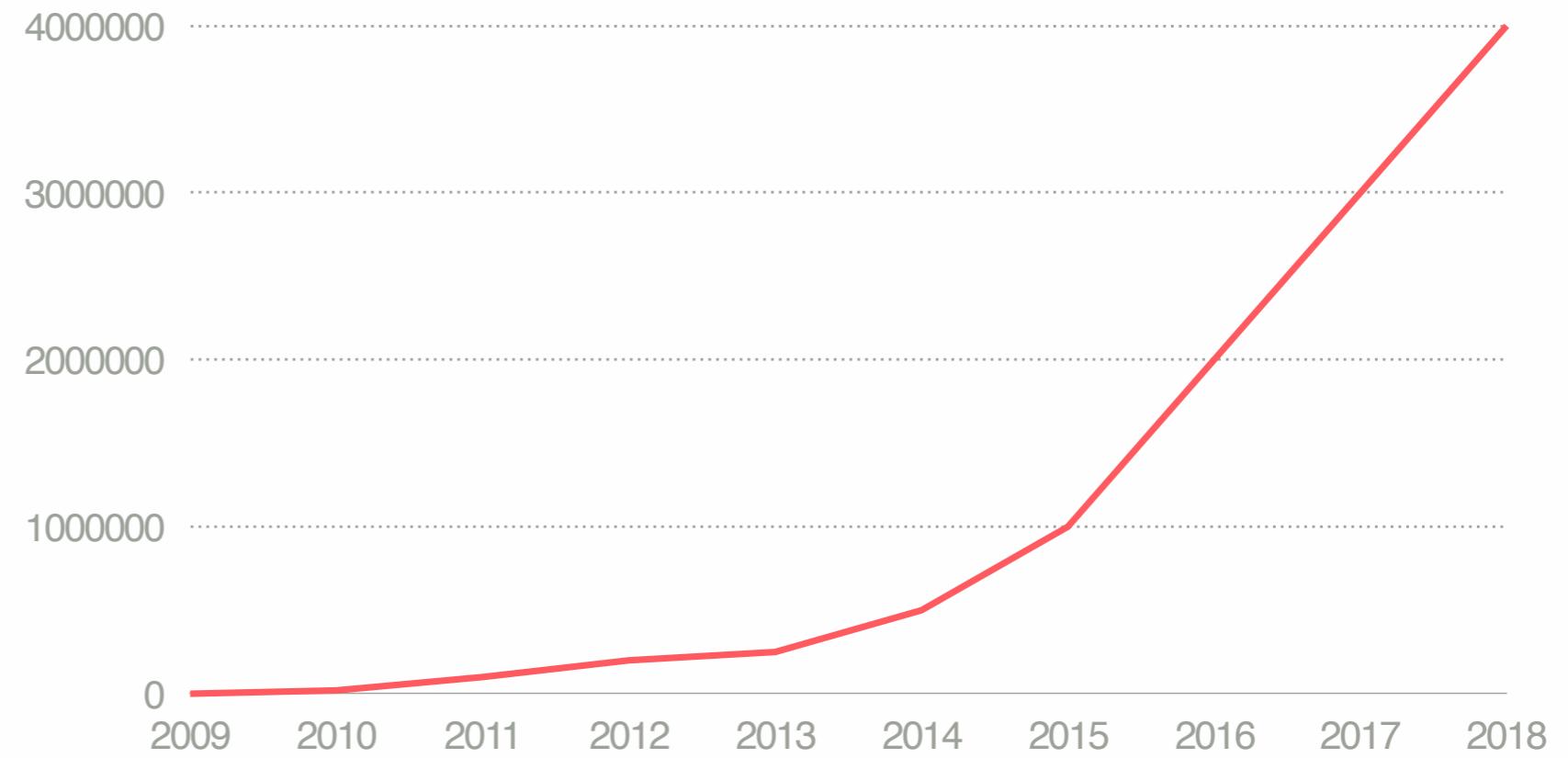


1 SRE



Sysops

**When it starts to
become a problem**
MONORAIL LOC



Democratic deploys

All developers are expected to “own” their features from implementation to production.

Each developer is expected to:

- deploy their changes
- monitor their changes
- roll back, abort, or revert if necessary

Deployboard ✓ Live Updating ✓ Merges and Deploys Unlocked Monorail 

Snapshots Deploy Targets Silhouettes master

Deploy 86441 of Monorail to Production
Started by john du at 1:24 pm.
Deploying snapshot #1574388 (05ad391)
Deploy is running! Please watch metrics!

Roll Back Abort

65 / 709

Deploy 86442 of Monorail to Next
Started by pengyu li at 1:25 pm.
Deploying snapshot #1574552 (887365c)
Deploy is running! Please watch metrics!

Roll Back Abort

0 / 11

Airbnb: Error rate (last 30 minutes)

Airbnb: Average response time, by tier (ms) (last 30 minutes)

Airbnb (Resque): Error rate (last 30 minutes)

New Relic Sentry Resque Synapse Dashboards Datadog Dashboards Hide graphs

Snapshot	SHA	Author	Pushed At (PDT)	Deployed	Compare To	Notes
<input type="checkbox"/> #1574636	b0a7345	feng.qian@ai...	Tue, Sep 27, 2016 1:11 PM		PREV HEAD C N P	 Deploy
<input type="checkbox"/> #1574564	085dc27	yat.choi@air...	Tue, Sep 27, 2016 1:05 PM		PREV HEAD C N P	 Deploy
<input type="checkbox"/> #1574560	8d1f378	nico.moscho...	Tue, Sep 27, 2016 1:04 PM		PREV HEAD C N P	 Deploy
<input type="checkbox"/> #1574552	887365c	tony.huang@...	Tue, Sep 27, 2016 1:03 PM		PREV HEAD C N P	 Deploy

Scaling Deploy Process

- builds, tests, and deploys must be fast and reliable
- ping authors when they need to take action
- automatic (and fast) reverts
- merge queue
- automatic rollbacks for new exceptions

Why Not Microservices?

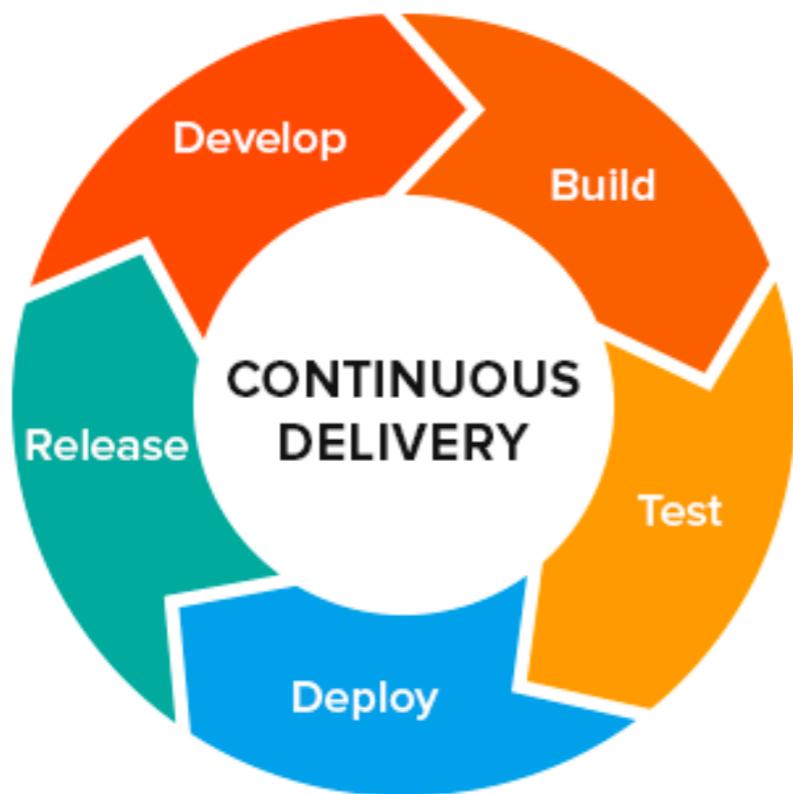
- Distributed systems are hard!
- Naming things is hard!
- Having to support different languages / frameworks

Why Microservices?

- Independent deployment
- Modular structure / Ownership
- Secure / Performant
- Upgrading dependencies

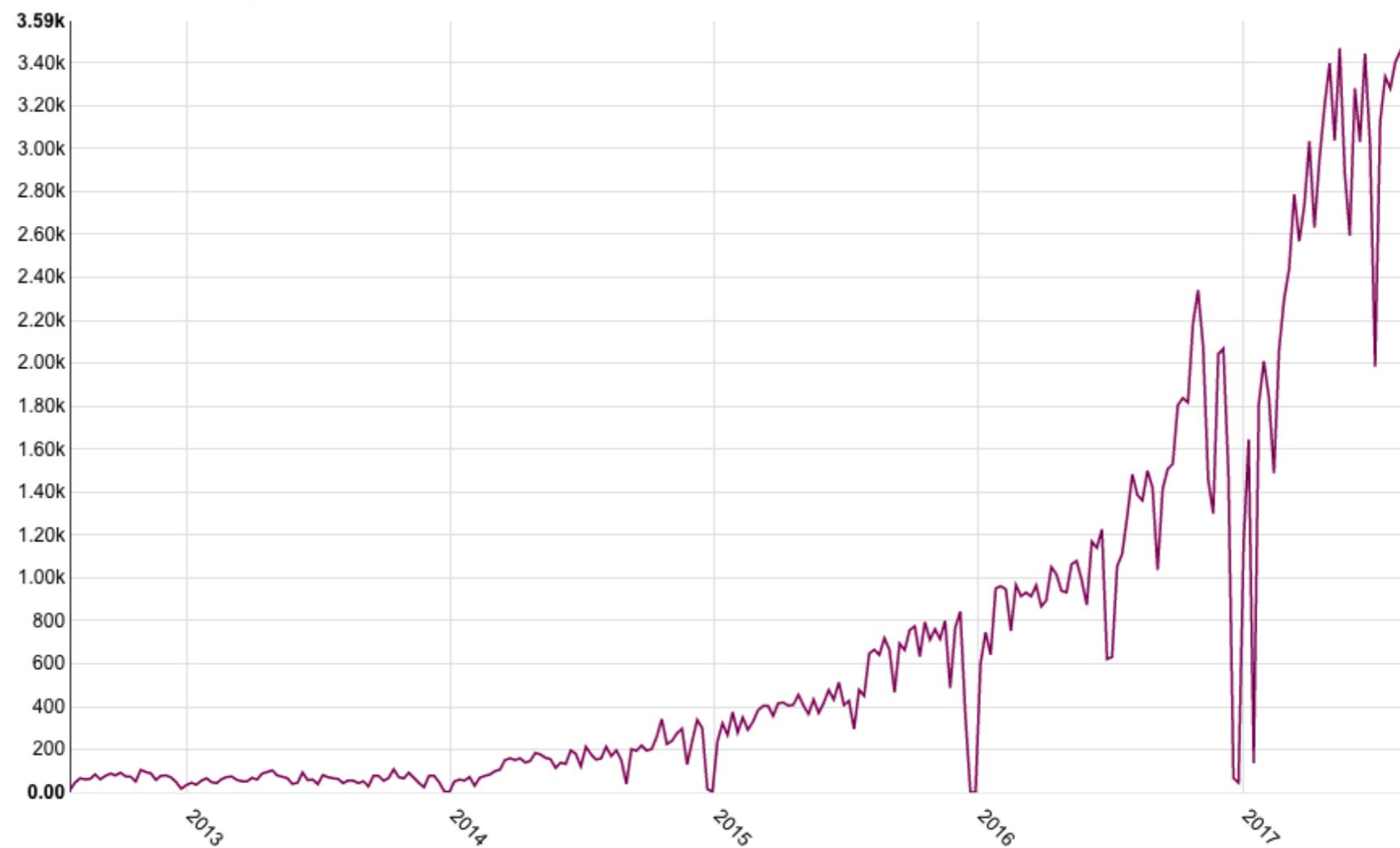
Why Microservices?

SCALING CONTINUOUS DELIVERY



Why Microservices?

Deploys per week (all apps, all environments)



Why Microservices?

75,000
production deploys
per year

Services

900
developers

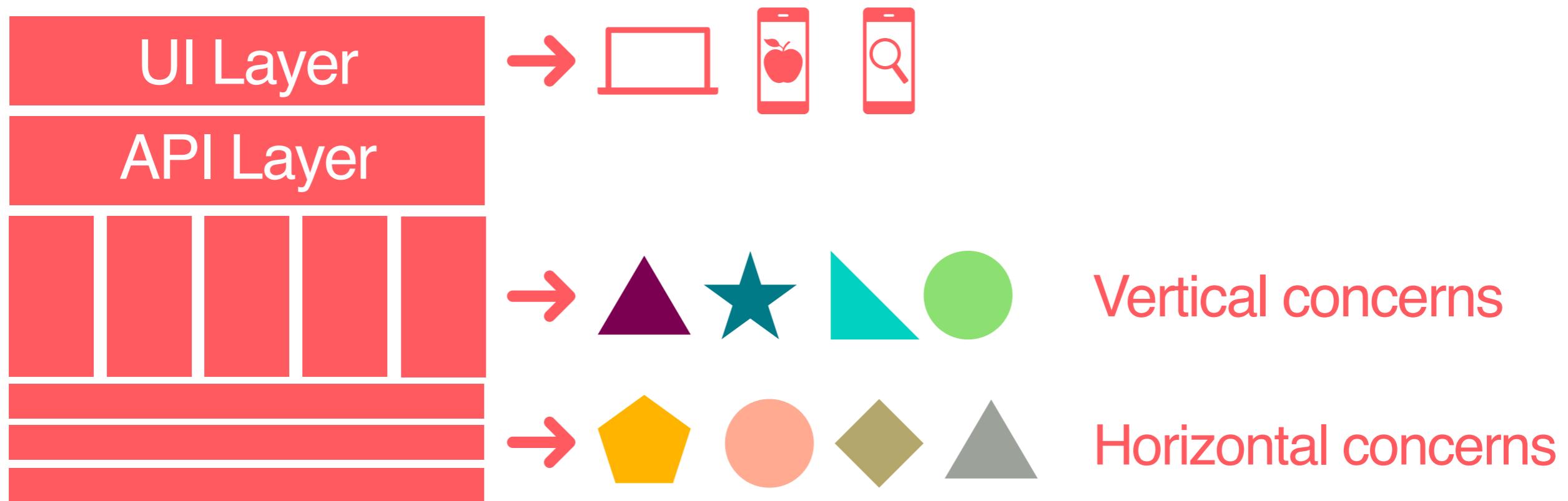
300
services

4k
deploys per week

Splitting the Monolith



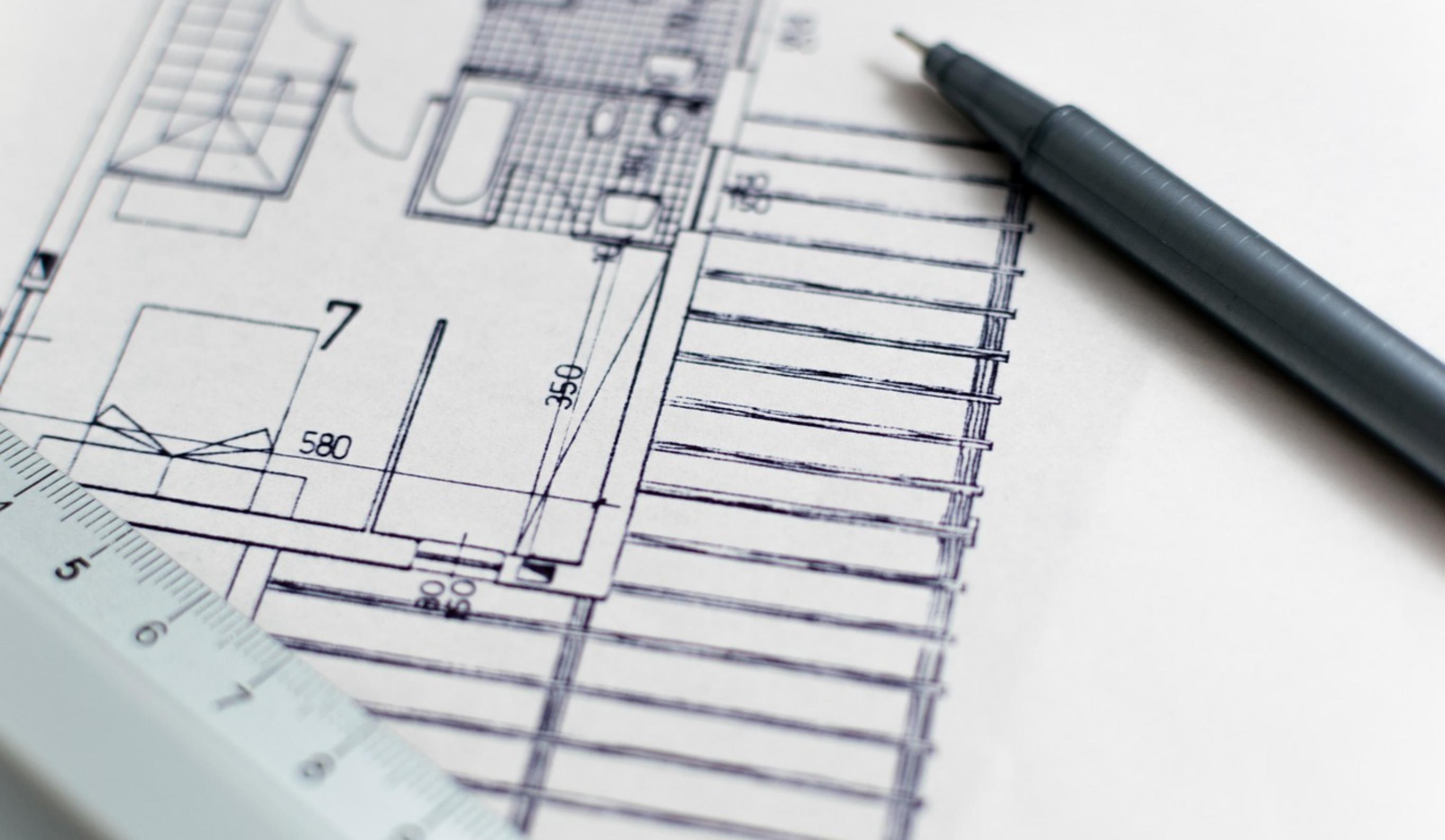
Splitting the Monolith



The important part

10 TAKEAWAYS

1. Monolith first
2. Devops culture
3. Configuration as code
4. Monitoring and Alerting
5. Continuous delivery
6. Automate what you can
7. Your first service is going to be bad
8. Services own their own data
9. Break apart the Monolith
10. Standardize service creation



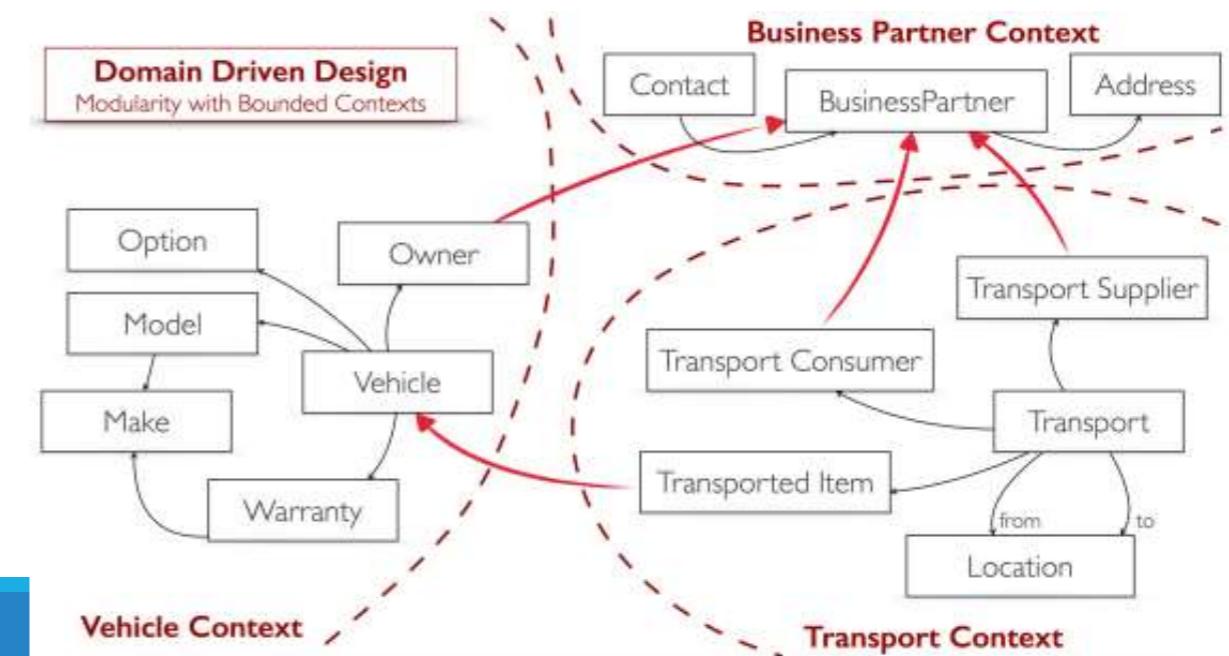
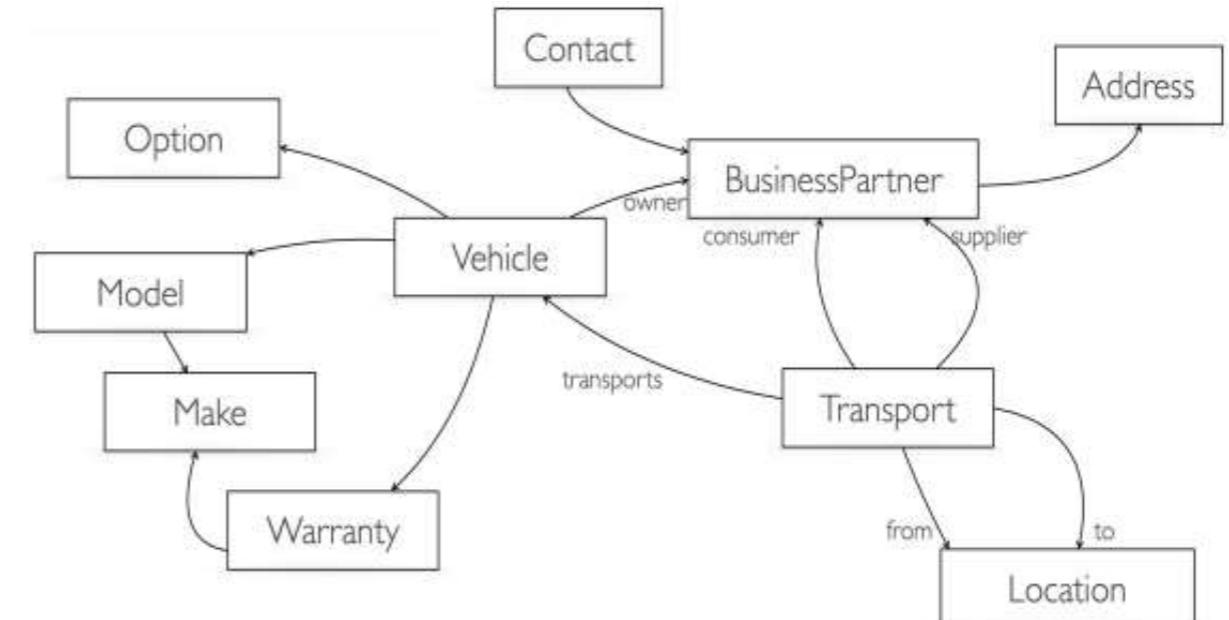
Elements of Domain-driven design

Understanding Requirement Analysis using DDD

Bounded Context

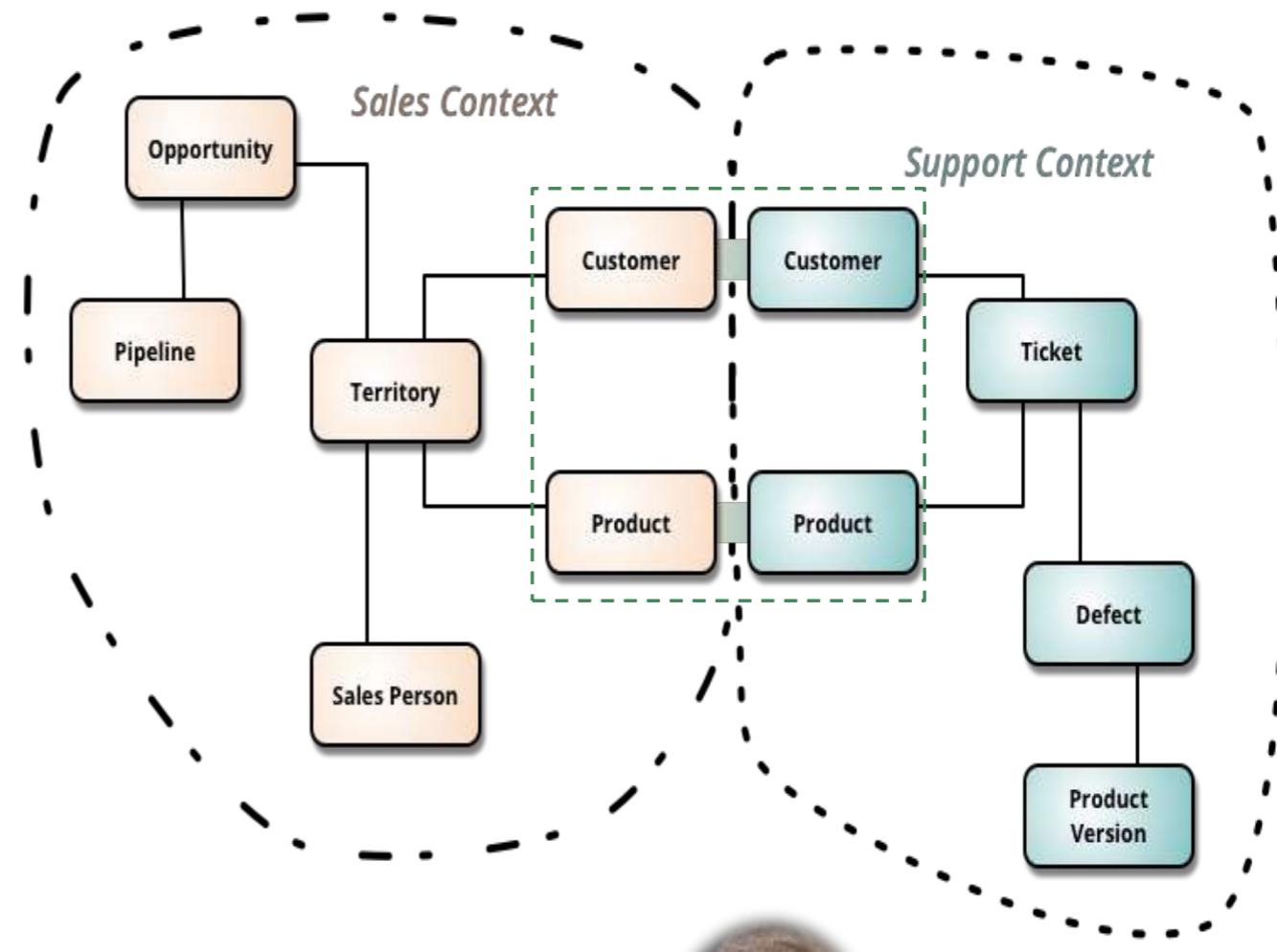
Areas of the domain treated independently

Discovered as you assess requirements and build language



DDD : Understanding Bounded Context

- DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their interrelationships.
- Bounded Contexts have both unrelated concepts
 - Such as a support ticket only existing in a customer support context
 - But also **share** concepts such as **products** and **customers**.
- Different contexts may have completely different models of common concepts with mechanisms to map between these polysemic concepts for integration.

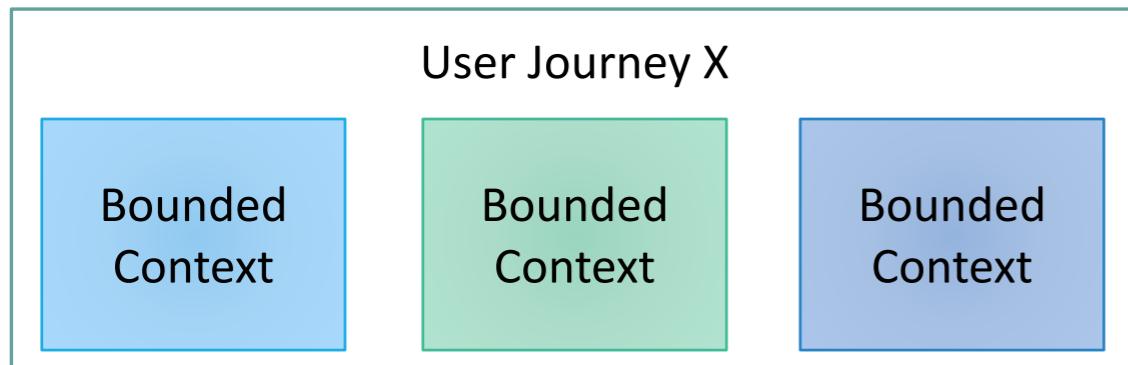


Source: BoundedContext By Martin Fowler :
<http://martinfowler.com/bliki/BoundedContext.html>



DDD: Bounded Context – Strategic Design

An App User's Journey can run across multiple Bounded Context / Micro Services.



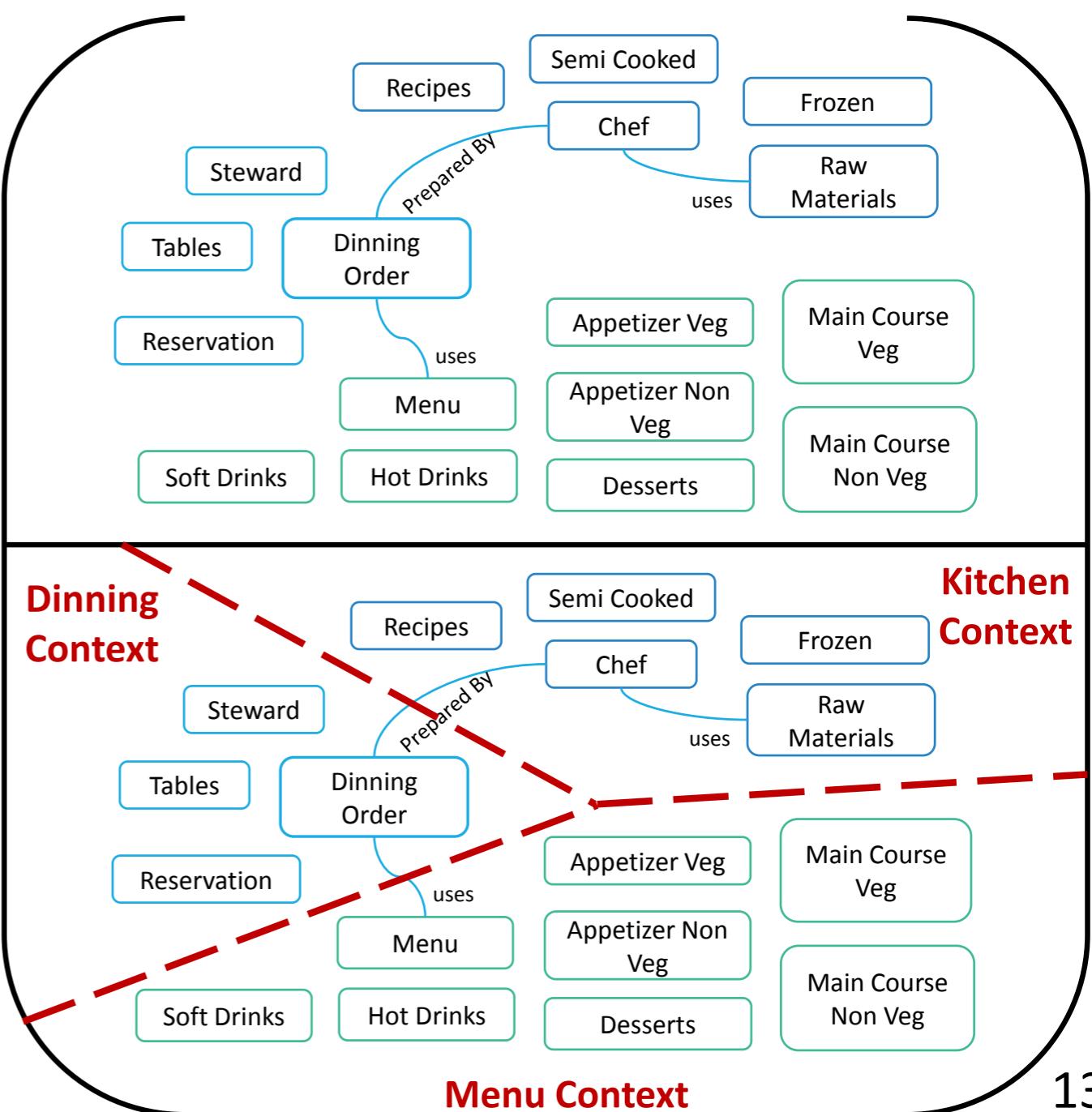
Areas of the domain treated independently

Discovered as you assess requirements and build language

Source: Domain-Driven Design
Reference by Eric Evans



Understanding Bounded Context (DDD) of a Restaurant App



DDD: Ubiquitous Language: Strategic Design

Ubiquitous Language

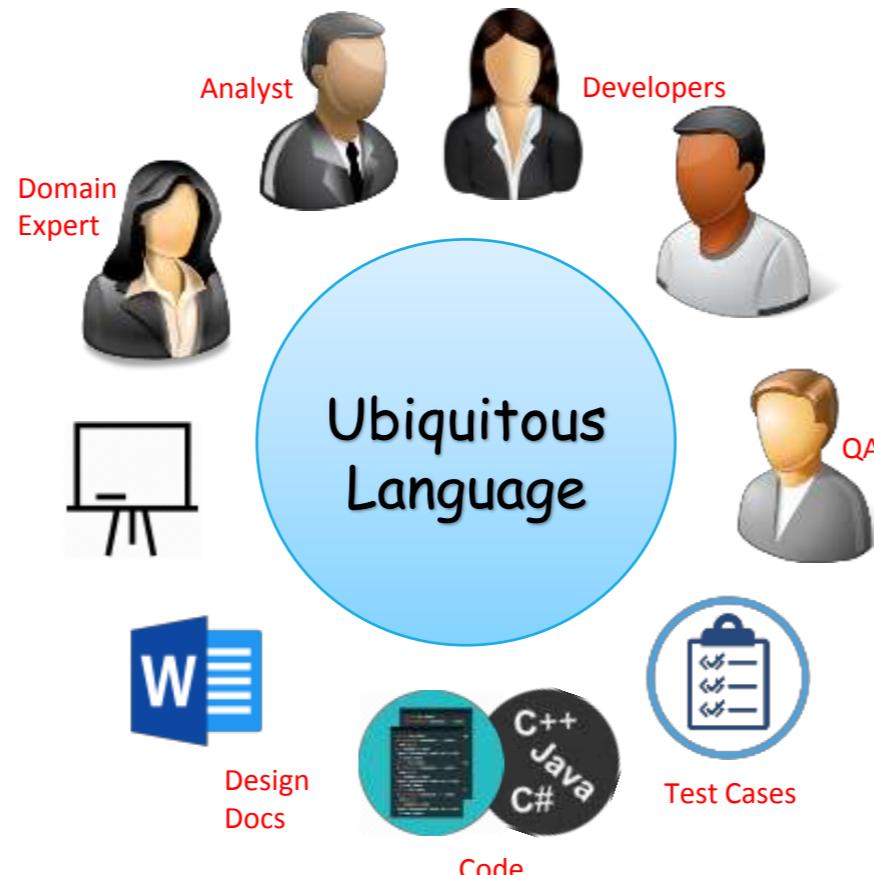
Vocabulary shared by all involved parties

Used in all forms of spoken / written communication

Restaurant Context – Food Item :

Eg. Food Item (Navrathnakurma) can have different meaning or properties depends on the context.

- In the Menu Context it's a Veg Dish.
- In the Kitchen Context it's a recipe.
- And in the Dining Context it will have more info related to user feed back etc.



Role-Feature-Reason Matrix

As an Restaurant Owner
I want to know who my Customers are
So that I can serve them better

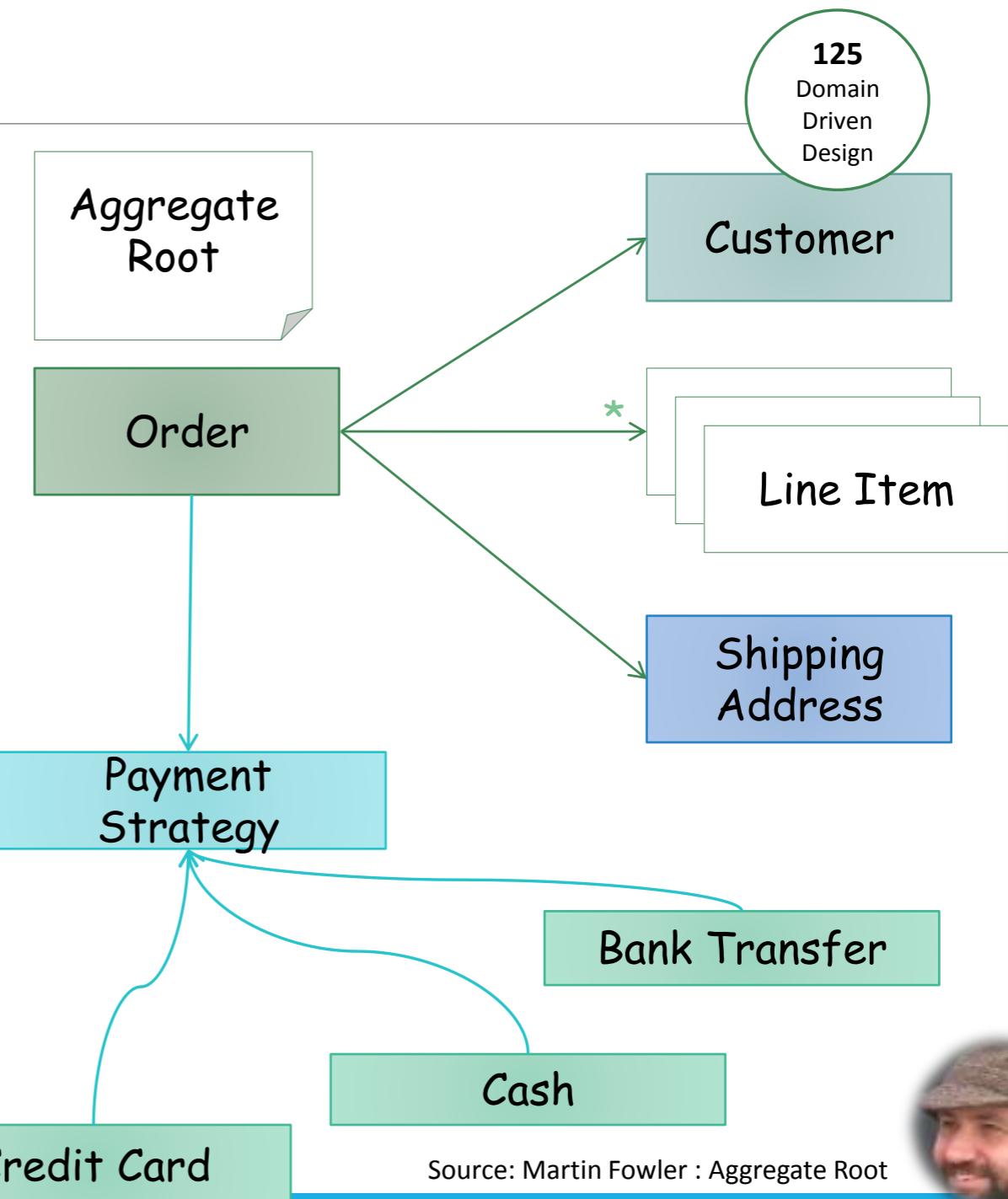
BDD Construct

Given	Customer John Doe exists
When	Customer orders food
Then	Assign customer preferences as Veg or Non Veg customer

BDD – Behavior Driven Development

Understanding Aggregate Root

- An aggregate will have one of its component objects be the aggregate root. **Any references from outside the aggregate should only go to the aggregate root.** The root can thus ensure the integrity of the aggregate as a whole.
- Aggregates are the basic element of transfer of data storage - you request to load or save whole aggregates. Transactions should not cross aggregate boundaries.
- Aggregates are sometimes confused with collection classes (lists, maps, etc.).
- Aggregates are **domain concepts** (order, clinic visit, **playlist**), while collections are generic. An aggregate will often contain multiple collections, together with simple fields.

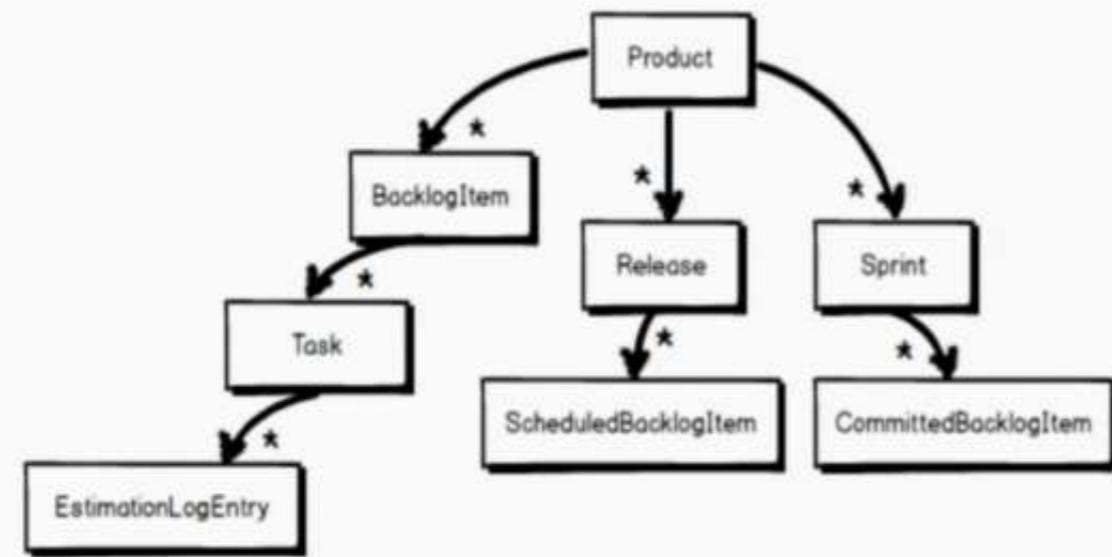


Source: Martin Fowler : Aggregate Root

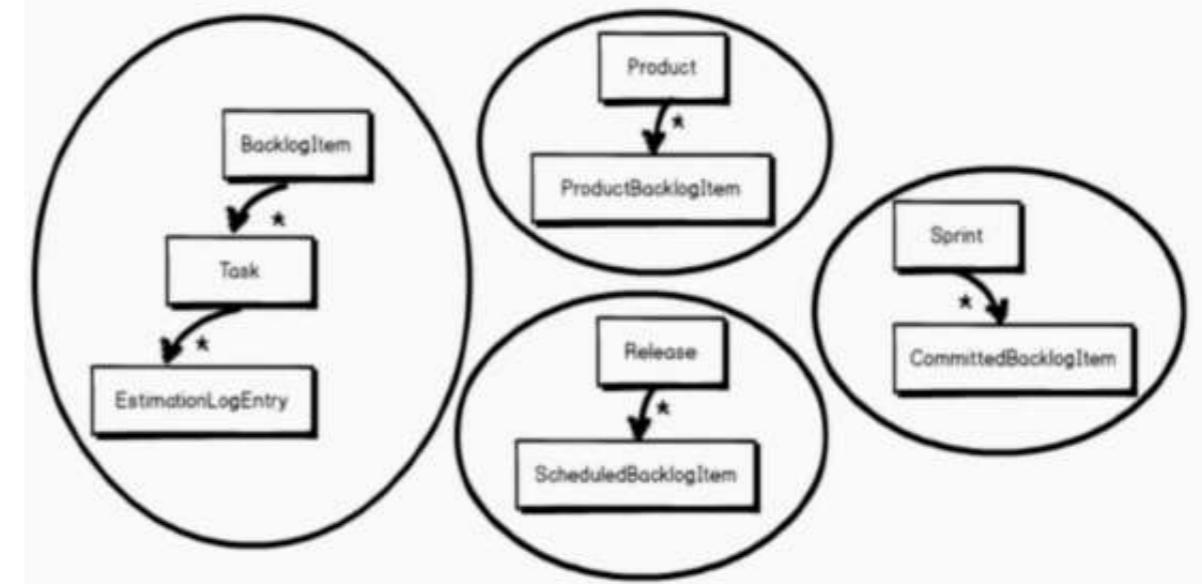


Designing and Fine Tuning Aggregate Root

Aggregate Root - #1



Aggregate Root - #2



Super Dense Single Aggregate Root
Results in Transaction concurrency issues.

Working on different design models helps the developers to come up with best possible design.

Super Dense Aggregate Root is split into 4 different smaller Aggregate Root in the 2nd Iteration.

Source : Effective Aggregate Design Part 1/2/3 : Vaughn Vernon
http://dddcommunity.org/wp-content/uploads/files/pdf_articles/Vernon_2011_1.pdf

Anemic Domain Model : Anti Pattern

- There are objects, many named after the nouns in the domain space, and these objects are connected with the rich relationships and structure that true domain models have.
- The catch comes when you look at the behavior, and you realize that there is **hardly any behavior** on these objects, making them little more than **bags of getters and setters**.
- The fundamental **horror** of this anti-pattern is that **it's so contrary to the basic idea of object-oriented design**; which is to combine data and process together.
- The **anemic domain model** is really just a **procedural style design**, exactly the kind of thing that object bigots like me (and Eric) have been fighting since our early days in Smalltalk.

```

1 package com.fusionfire.examples.commons.utils;
2
3 import java.util.ArrayList;
4
5 public class AnemicUser {
6
7     private String name;
8
9     private boolean isUserLocked;
10
11    private ArrayList<String> addresses;
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public boolean isUserLocked() {
22        return isUserLocked;
23    }
24
25    public void setUserLocked(boolean isUserLocked) {
26        this.isUserLocked = isUserLocked;
27    }
28
29    public ArrayList<String> getAddresses() {
30        return addresses;
31    }
32
33    public void setAddresses(ArrayList<String> addresses) {
34        this.addresses = addresses;
35    }
36
37}

```

- lockUser()
- unlockUser()
- addAddress(String address)
- removeAddress(String address)

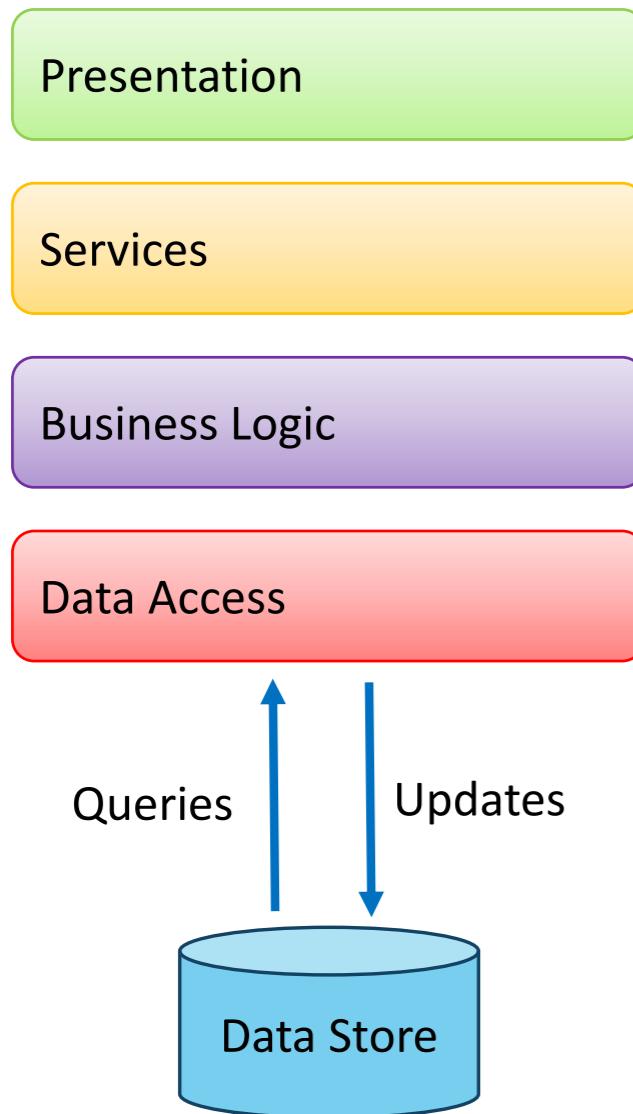


Source: Anemic Domain Model By Martin Fowler :
<http://martinfowler.com/bliki/AnemicDomainModel.html>

CRUD and CQRS

Source: <https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>
https://blogs.msdn.microsoft.com/maarten_mullender/2004/07/23/crud-only-when-you-can-afford-it-revisited

Traditional CRUD Architecture

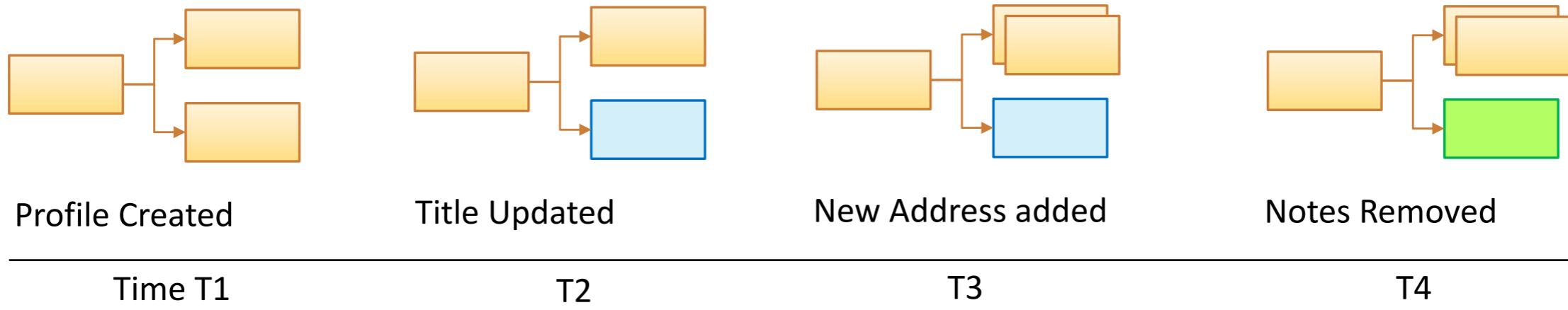


CRUD Disadvantages

- **A mismatch between the read and write representations of the data.**
- **It risks data contention when records are locked in the data store** in a collaborative domain, where multiple actors operate in parallel on the same set of data. These risks increase as the complexity and throughput of the system grows.
- **It can make managing security and permissions more complex** because each entity is subject to both read and write operations, which might expose data in the wrong context.

Event Sourcing Intro

Standard CRUD Operations – Customer Profile – Aggregate Root



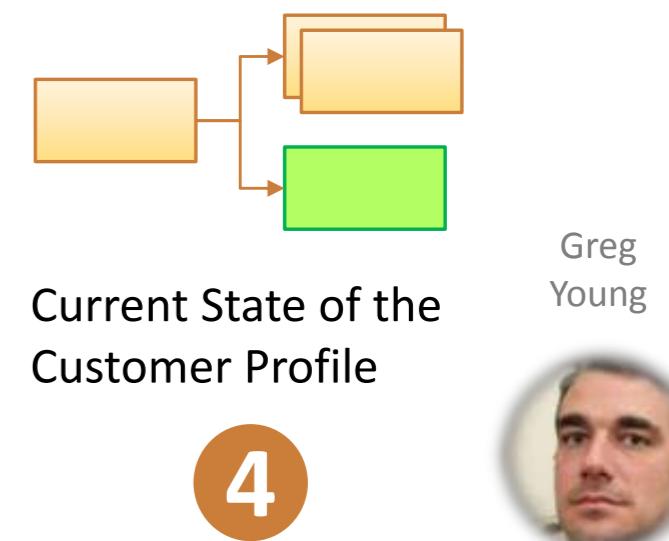
Event Sourcing and Derived Aggregate Root

- Commands**
1. Create Profile
 2. Update Title
 3. Add Address
 4. Delete Notes

- Events**
1. Profile Created Event
 2. Title Updated Event
 3. Address Added Event
 4. Notes Deleted Event

2

3



4

Restaurant Dining – Event Sourcing and CQRS

Processes



1

When people arrive at the Restaurant and take a table, a **Table** is **opened**. They may then **order drinks** and **food**. **Drinks** are **served** immediately by the table staff, however **food** must be **cooked** by a **chef**. Once the **chef prepared** the food it can then be **served**. **Table** is **closed** then the **bill** is prepared.

Customer Journey thru Dinning Processes

Commands

- Add Drinks
- Add Food
- Update Food

2

- Open Table
- Add Juice
- Add Soda
- Add Appetizer 1
- Add Appetizer 2
- Remove Soda
- Add Food 1
- Add Food 2
- Place Order
- Close Table

Food Menu



Dining



Kitchen



Order



Payment



Microservices

ES Aggregate

- Dinning Order
- Billable Order

4

Events

- Drinks Added
- Food Added
- Food Updated
- Food Discontinued

3

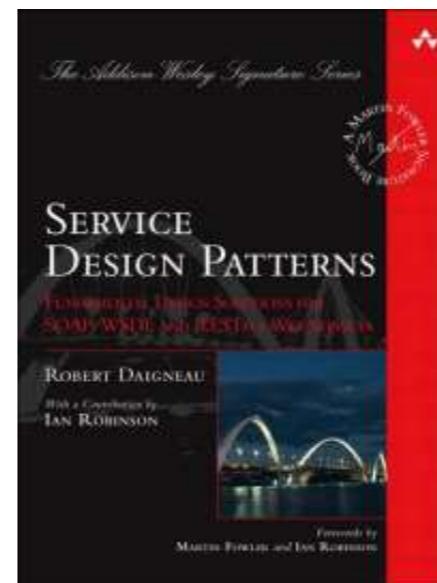
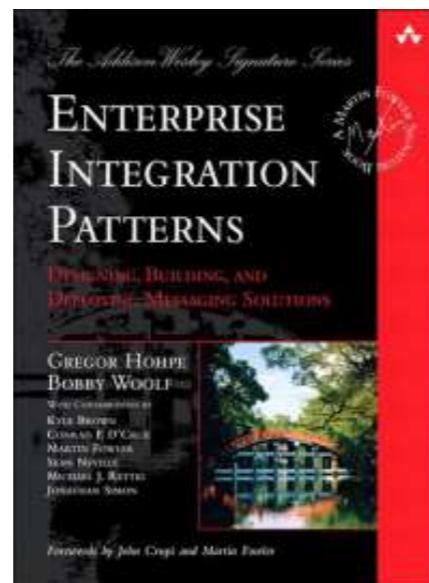
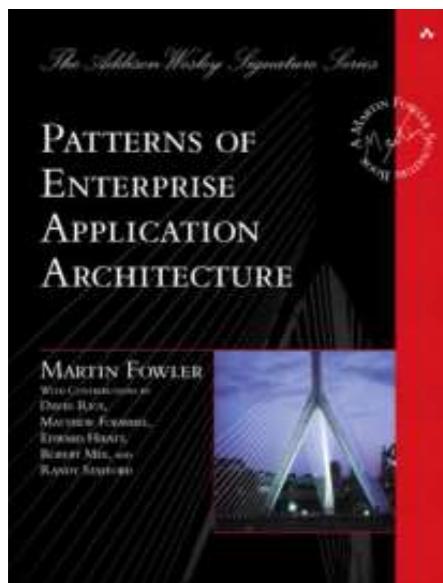
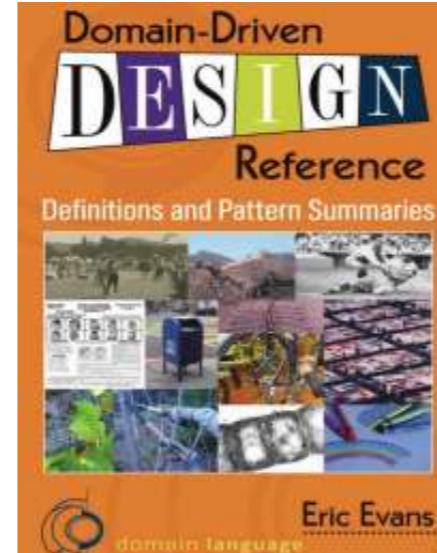
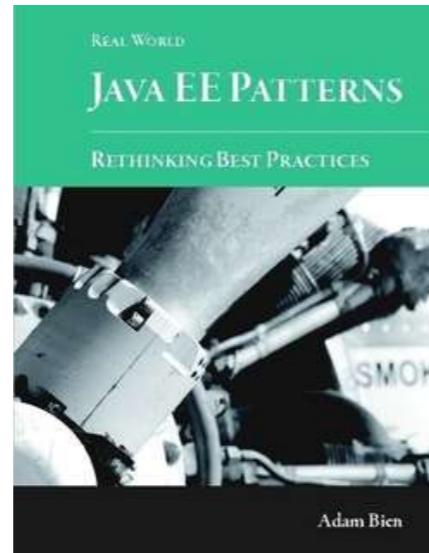
- Table Opened
- Juice Added
- Soda Added
- Appetizer 1 Added
- Appetizer 2 Added
- Remove Soda
- Food 1 Added
- Food 2 Added
- Order Placed
- Table Closed
- Juice Served
- Soda Served
- Appetizer Served
- Food Prepared
- Food Served
- Bill Prepared
- Payment Processed

Summary – Event Sourcing and CQRS

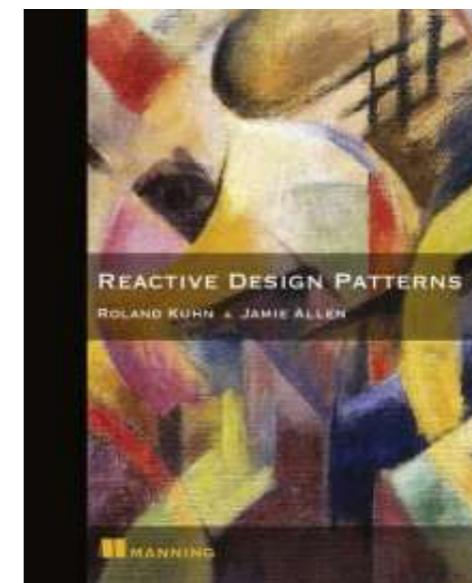


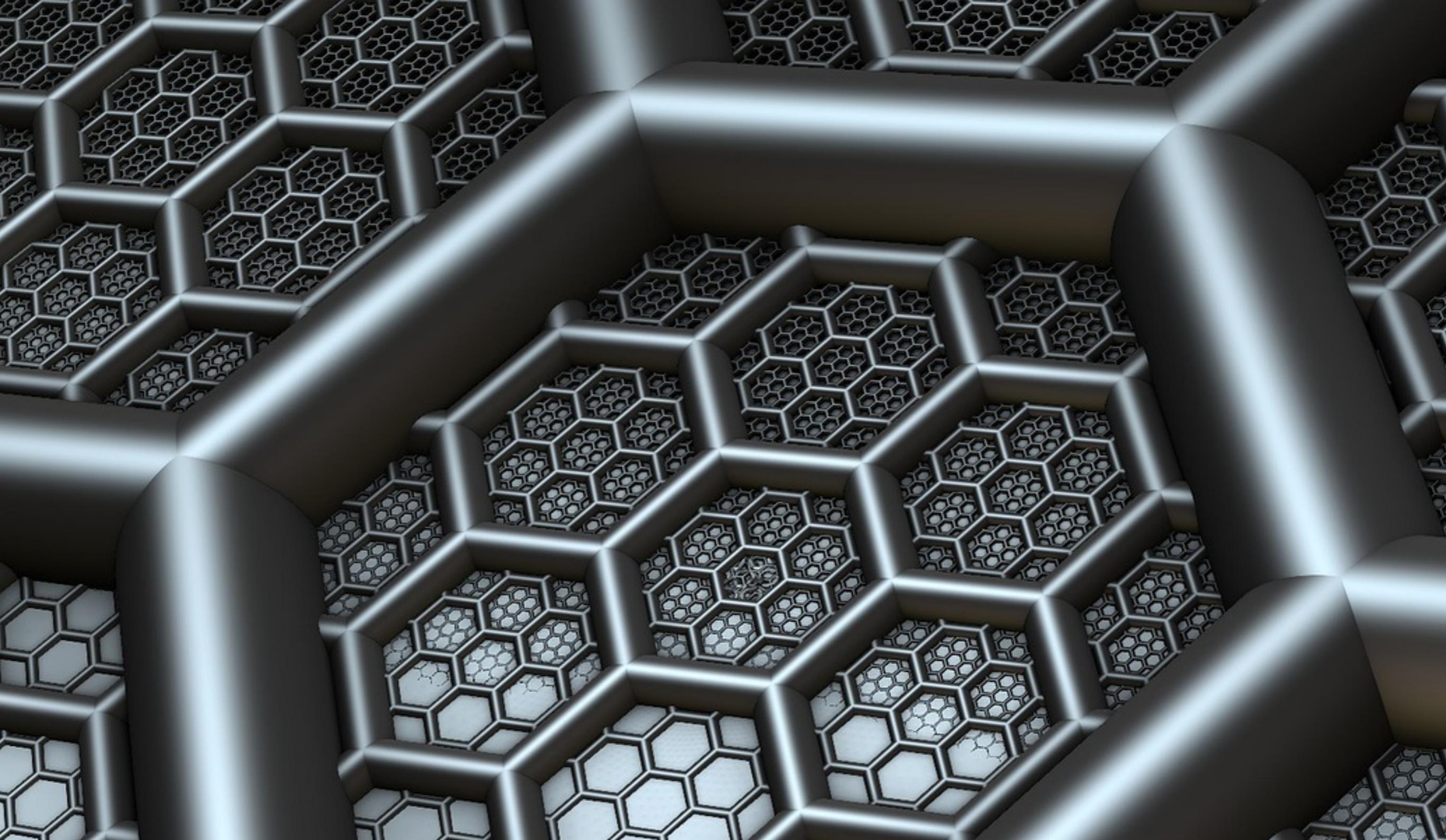
1. Immutable Events
2. Events represents the state change in Aggregate Root
3. Aggregates are Derived from a Collection of Events.
4. Separate Read and Write Models
5. Commands (originated from user or systems) creates Events.
6. Commands and Queries are always separated and possibly reads and writes using different data models.

Design Patterns – Holy Grail of Developers



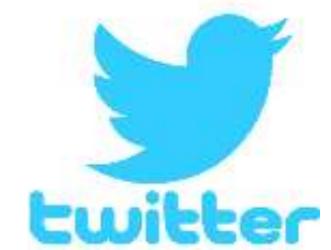
Design Patterns are solutions to general problems that software developers faced during software development.





Microservice Architecture (and hexagonal ones)

Pioneers in Microservices Implementation



New Entrants



Hexagonal Architecture

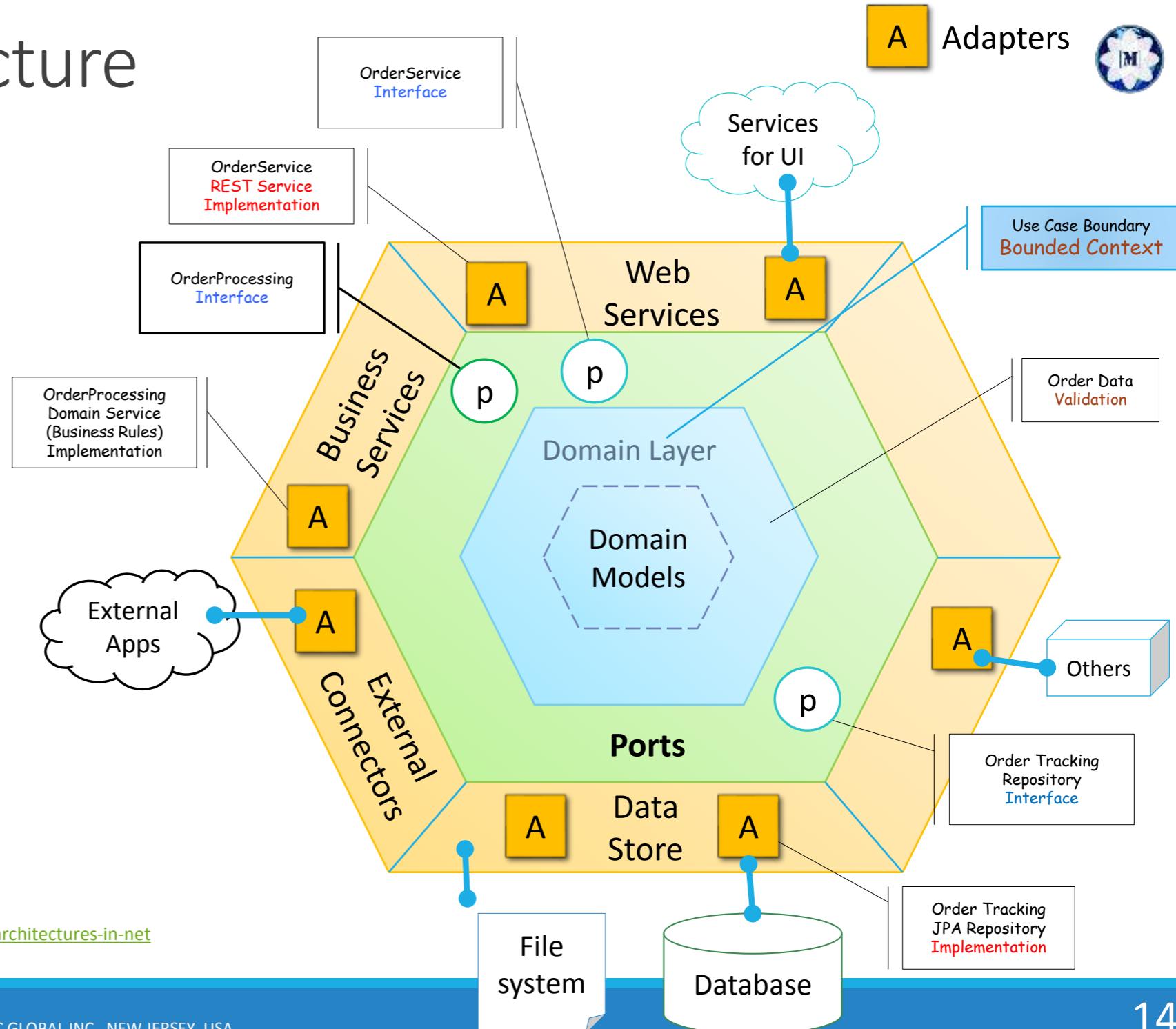
Ports & Adapters

The layer between the **Adapter** and the **Domain** is identified as the **Ports** layer. The Domain is inside the port, adapters for external entities are on the outside of the port.

The notion of a “port” invokes the OS idea that any device that adheres to a known protocol can be plugged into a port. Similarly many adapters may use the Ports.

- Reduces Technical Debt
- Dependency Injection
- Auto Wiring

Source : <http://alistair.cockburn.us/Hexagonal+architecture>
<https://skillsmatter.com/skillscasts/5744-decoupling-from-asp-net-hexagonal-architectures-in-net>



Micro Services Characteristics

By James Lewis and Martin Fowler



Components
via
Services



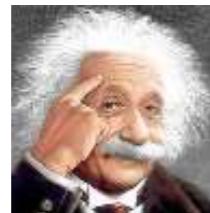
Organized around
**Business
Capabilities**



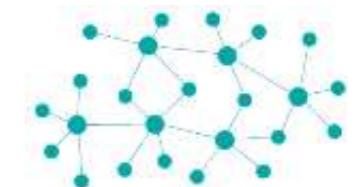
Products
NOT
Projects



**Smart
Endpoints**
& Dumb Pipes



Decentralized
Governance &
Data Management



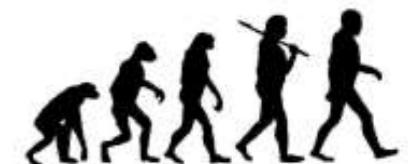
**Infrastructure
Automation**



**Design for
Failure**



Evolutionary
Design



The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be.

Alan Kay, 1998 email to the Squeak-dev list

Modularity ... is to a technological economy what the division of labor is to a manufacturing one.
W. Brian Arthur,
author of *e Nature of Technology*

We can scale our operation independently, maintain unparalleled system availability, and introduce new services quickly without the need for massive reconfiguration. —
Werner Vogels, CTO, Amazon Web Services

Pros and Cons



Pros

1. Robust
2. Scalable
3. Testable (Local)
4. Easy to Change and Replace
5. Easy to Deploy
6. Technology Agnostic

Cons

1. Adds Complexity
2. Skillset shortage
3. Confusion on getting the right size
4. Team need to manage end-to-end of the Service (From UI to Backend to Running in Production).

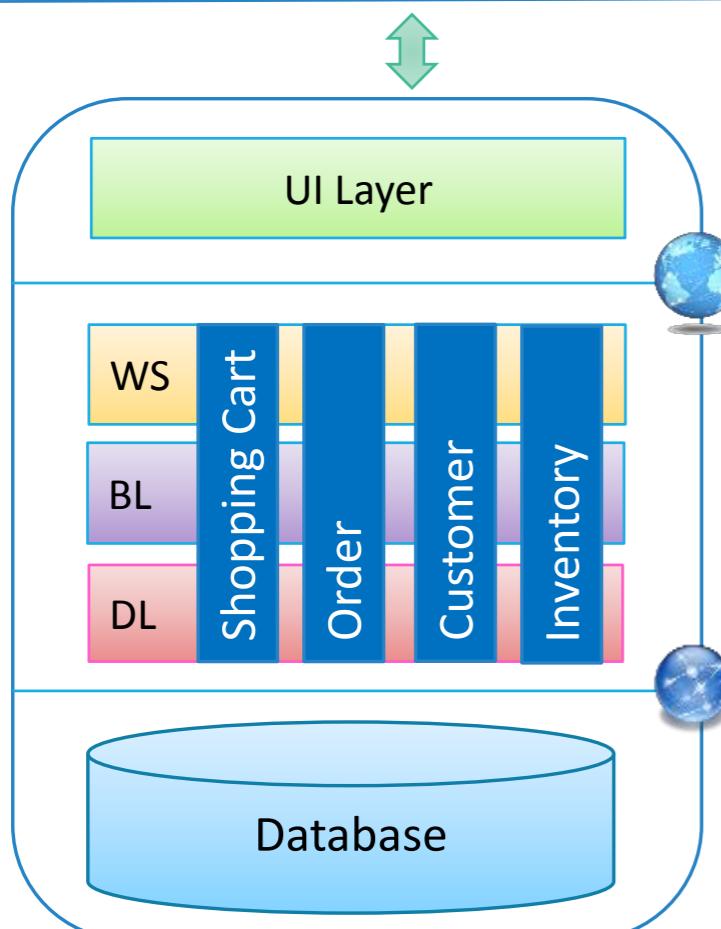
Monolithic vs. Micro Services Example



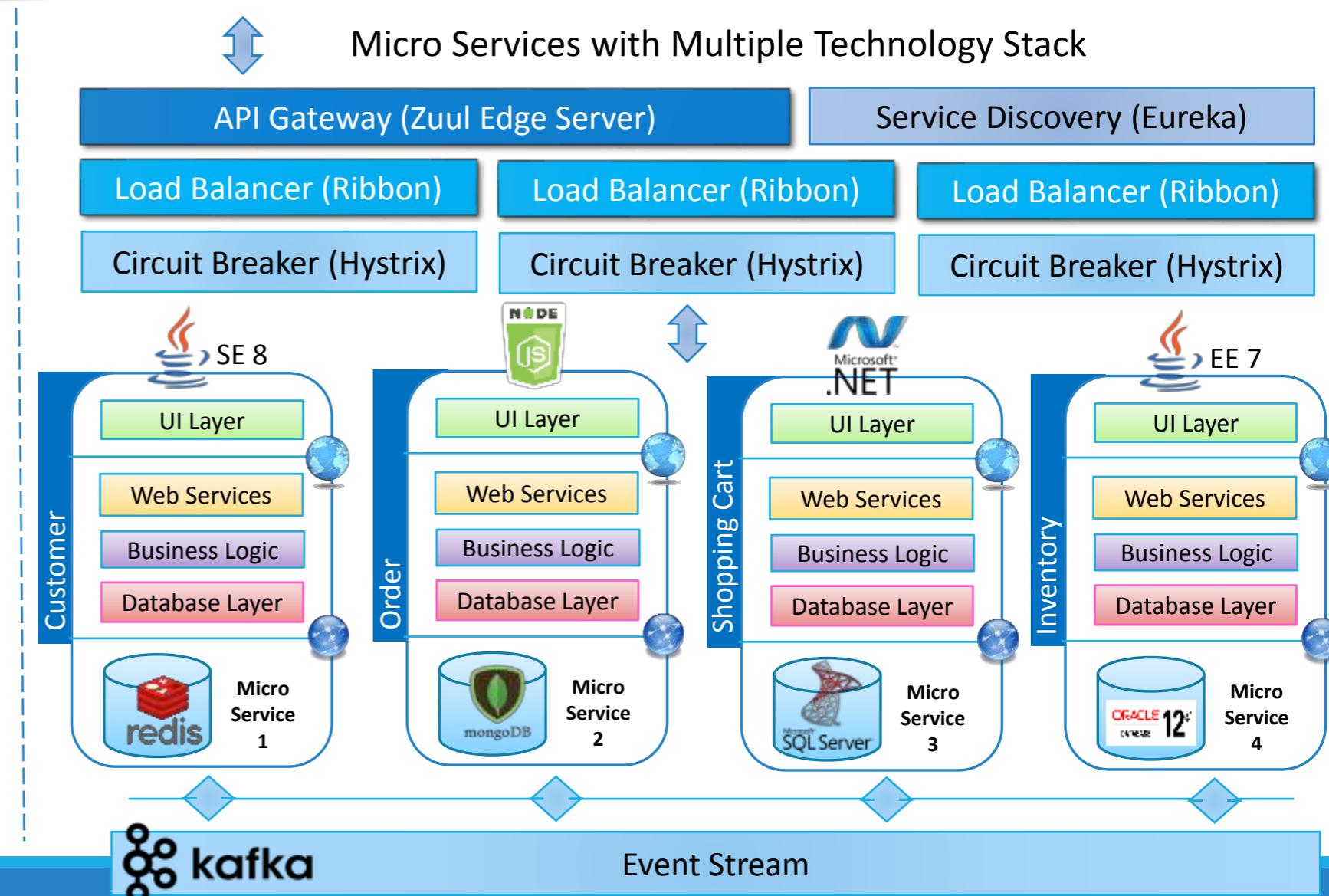
Existing aPaaS vendors creates Monolithic Apps.

This 3 tier model is obsolete now.

Source: Gartner Market Guide for Application Platforms Nov 23, 2016



Traditional Monolithic App using Single Technology Stack



SOA vs. Micro Services Example

