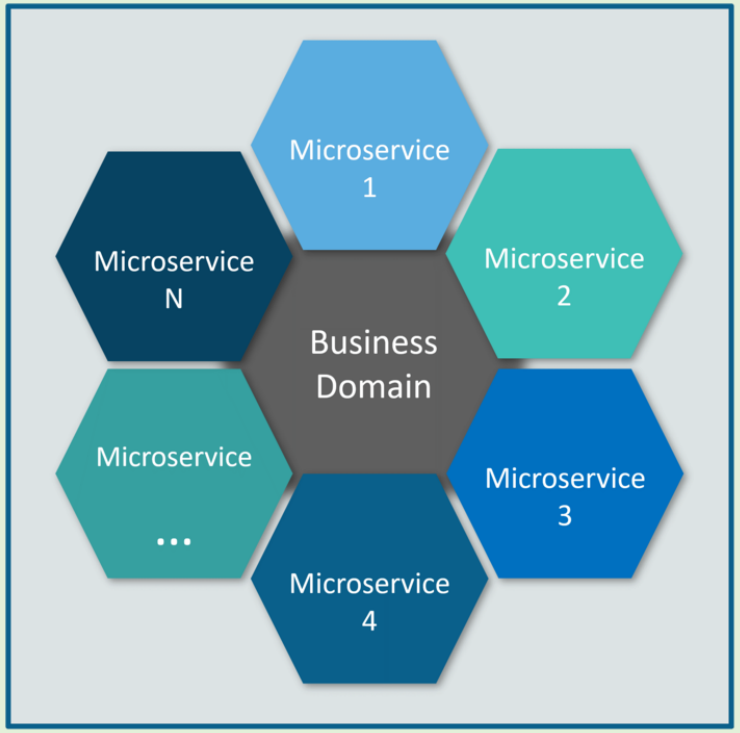


Microservices as a Product

Vidya Vrat Agarwal

Principal Architect T-Mobile

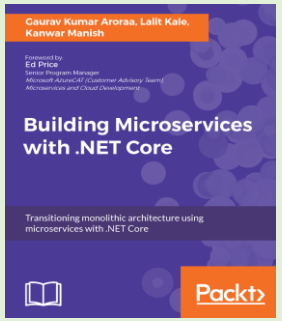
www.MyPassionFor.Net | **@dotNetAuthor**

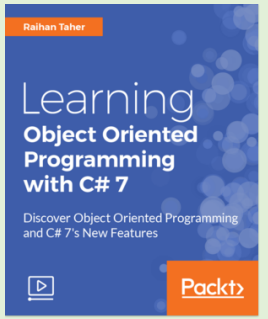
https://www.linkedin.com**/in/vidyavrat/**

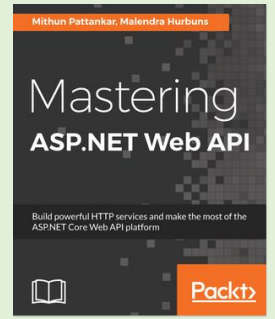
https://github.com/**vidyavrat**

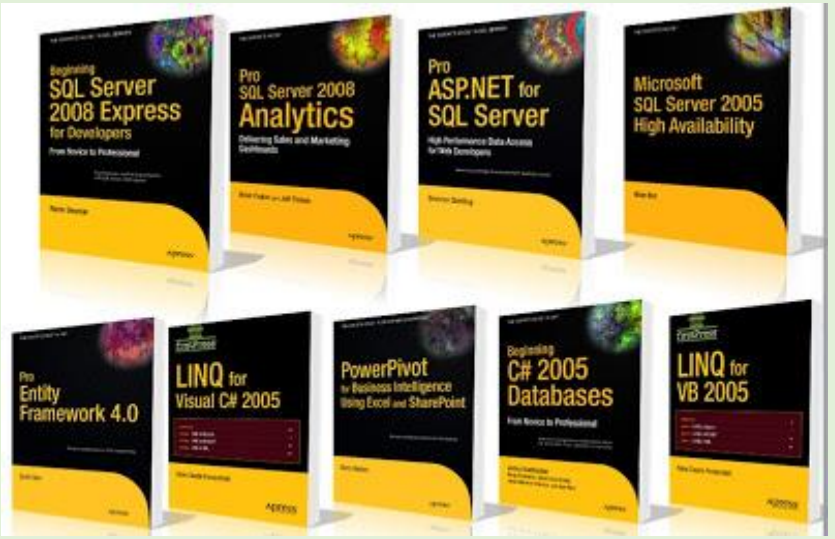
About Me

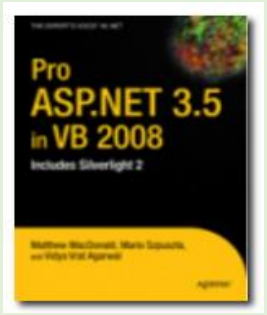
- 20+ years of industry experience
- Principal Architect with T-Mobile
- Microsoft MVP | C# Corner MVP
- TOGAF Certified Architect
- Certified Scrum Master (CSM)
- Microsoft Certified (MCT, MCSD / MCAD .NET, MCTS etc.)
- Published Author (5), and Technical Reviewer (over a dozen)

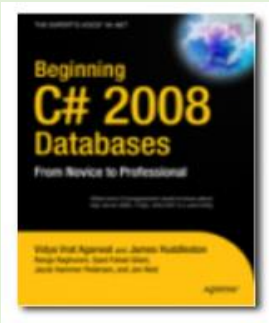


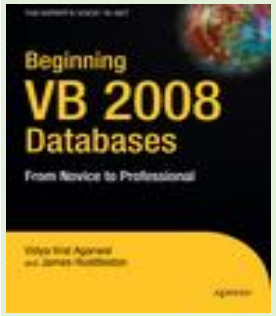


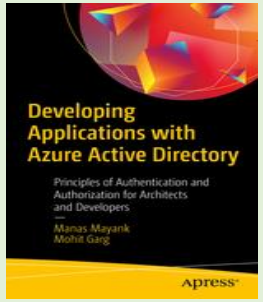




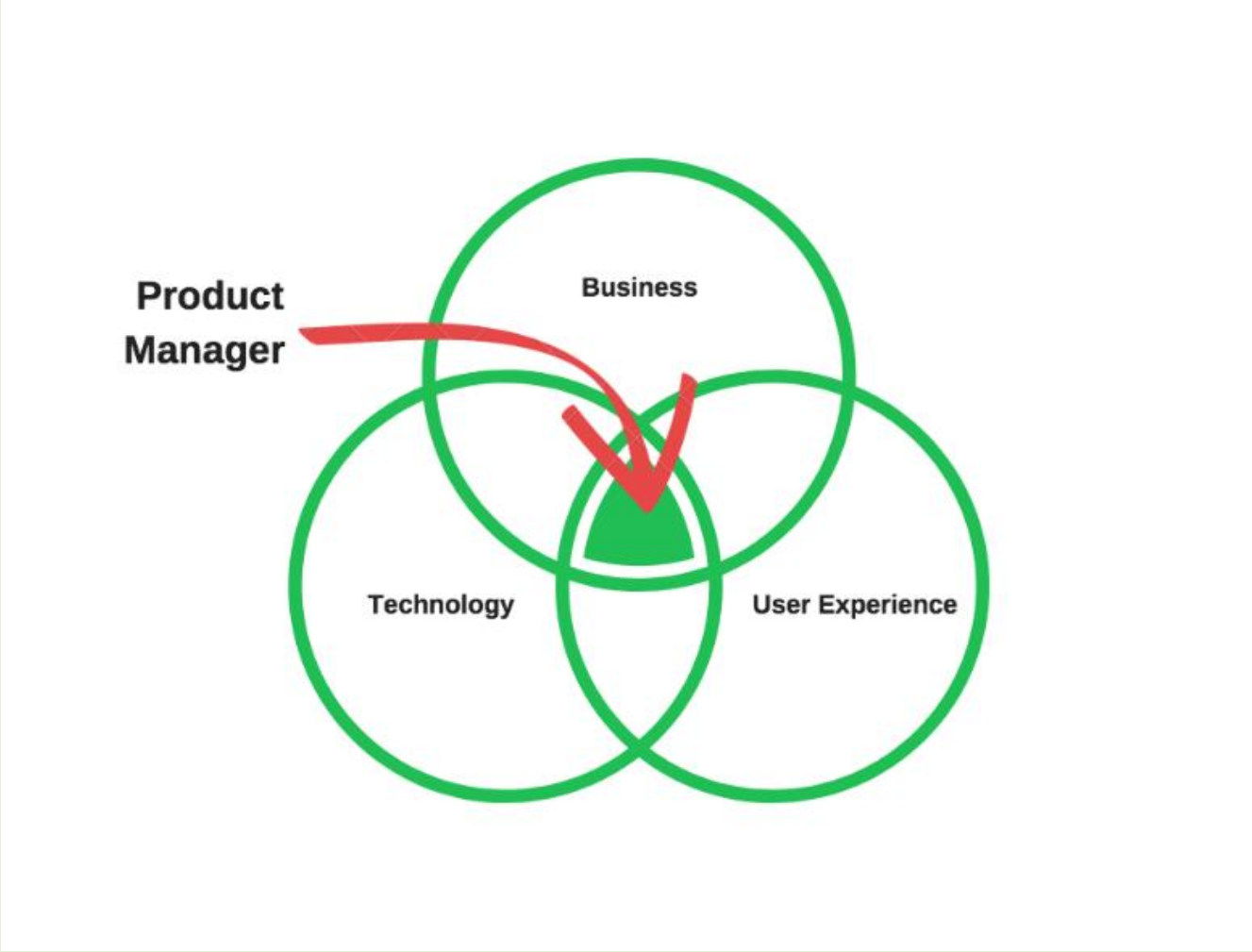








Product Management



3 Key Takeaways from this session

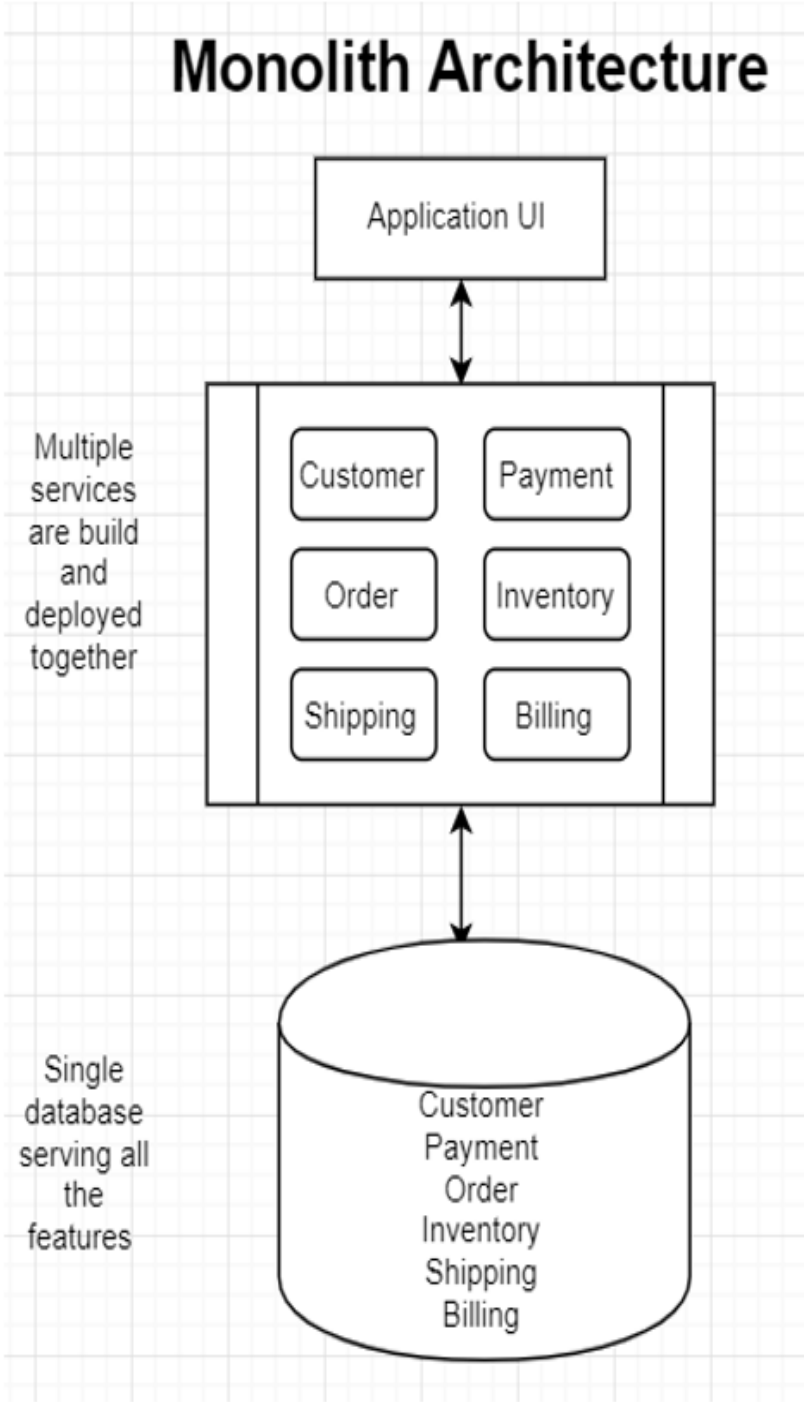
1. Walk out with a "*clear understanding*" of Microservices architecture

2. "Lessons Learned" / "Dealing with ambiguity" while working with cross-domain/functional teams.

3. "*How*" some real-world problems are being solved with Microservices.

Monolith /SOA

- An application which is developed, build and deployed as a single unit, and a single data store is used for the application.



Microservice

- A “Microservice is a small autonomous, tightly scoped, loosely coupled, independently deployable, and independently scalable application component.”

- Global Microservice Architecture Market anticipated accreting to US$ 33 Billion by 2023.

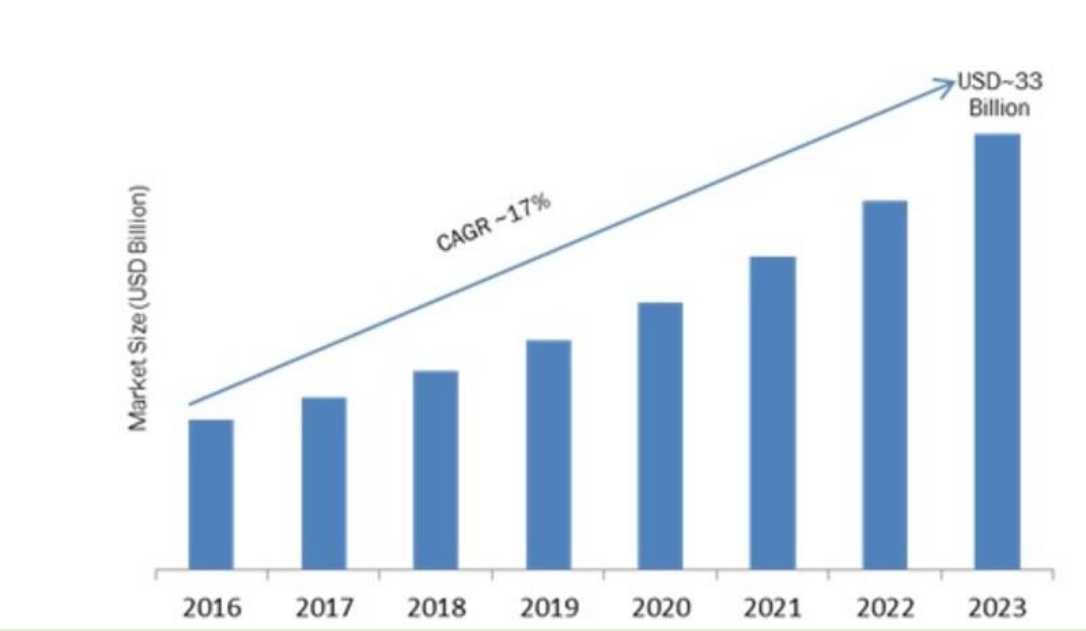
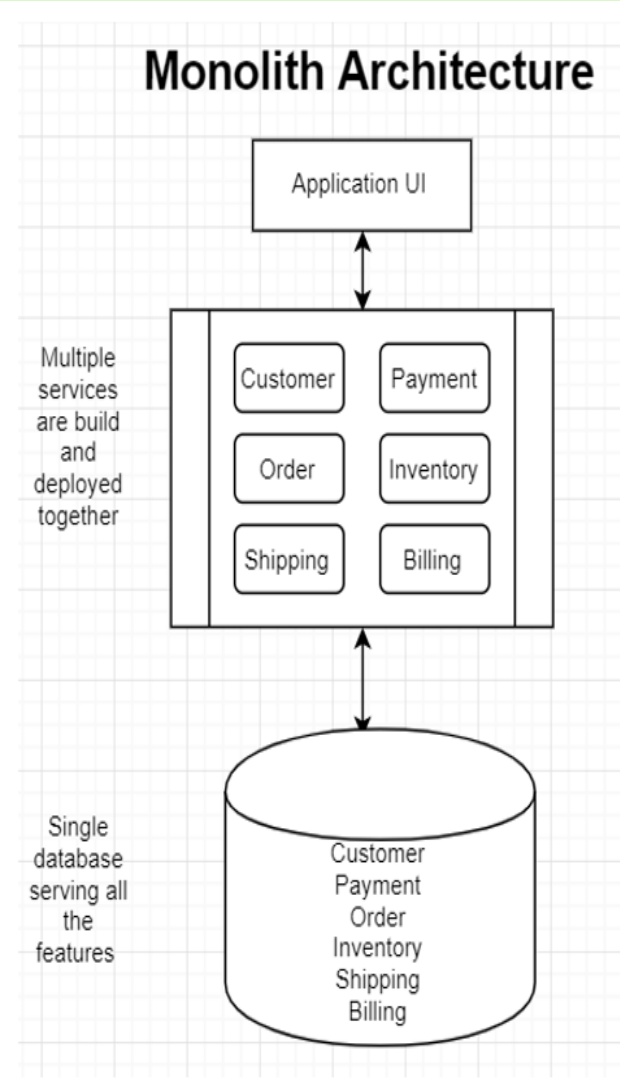


Image Reference - https://www.marketresearchfuture.com/reports/microservices-architecture-market-3149

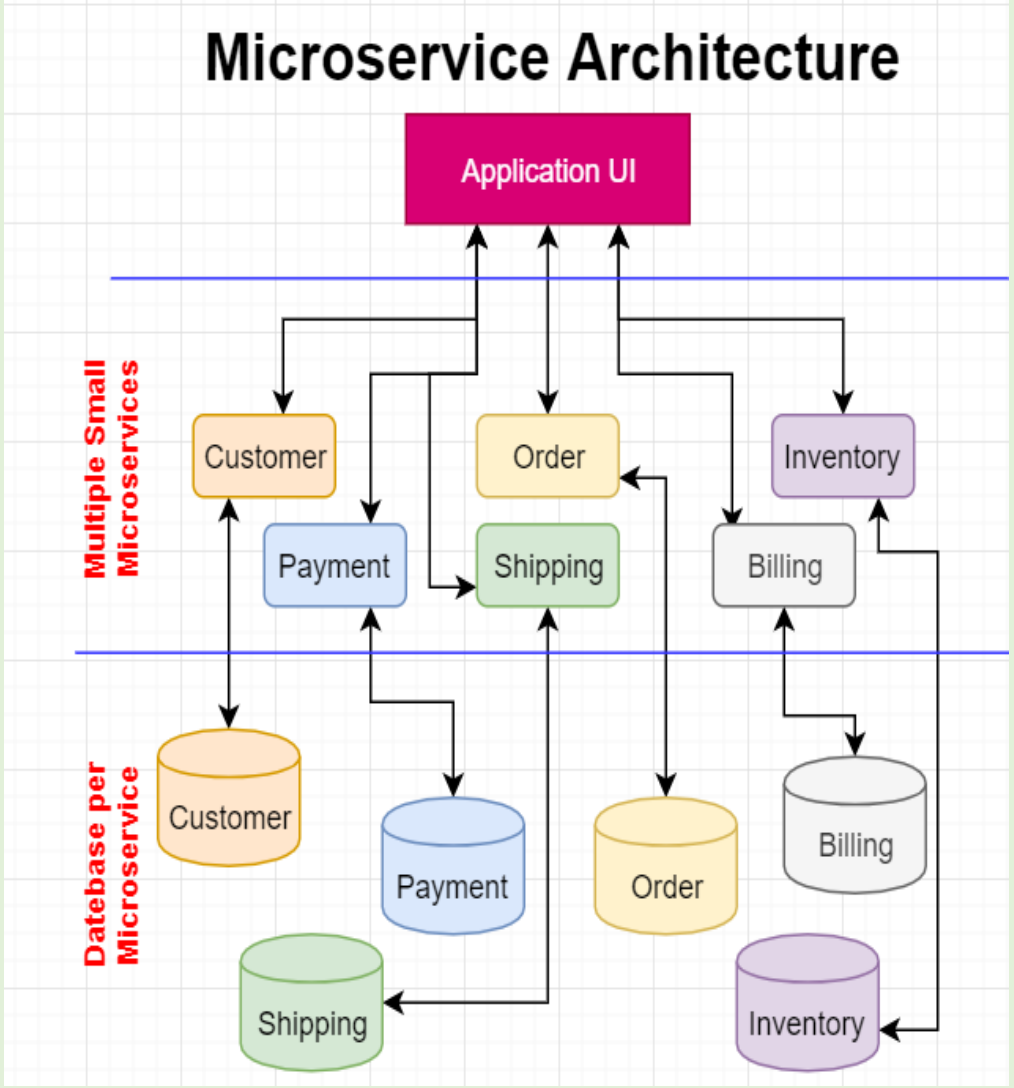
Finding a Bounded Context

A bounded context is an explicit boundary within which a domain model exists.



Microservice Architecture Violation

- Billing can support payment info
- Billing can support customer data
- Order can support inventory
- Payment can support T&Cs
- Order can generate receipt

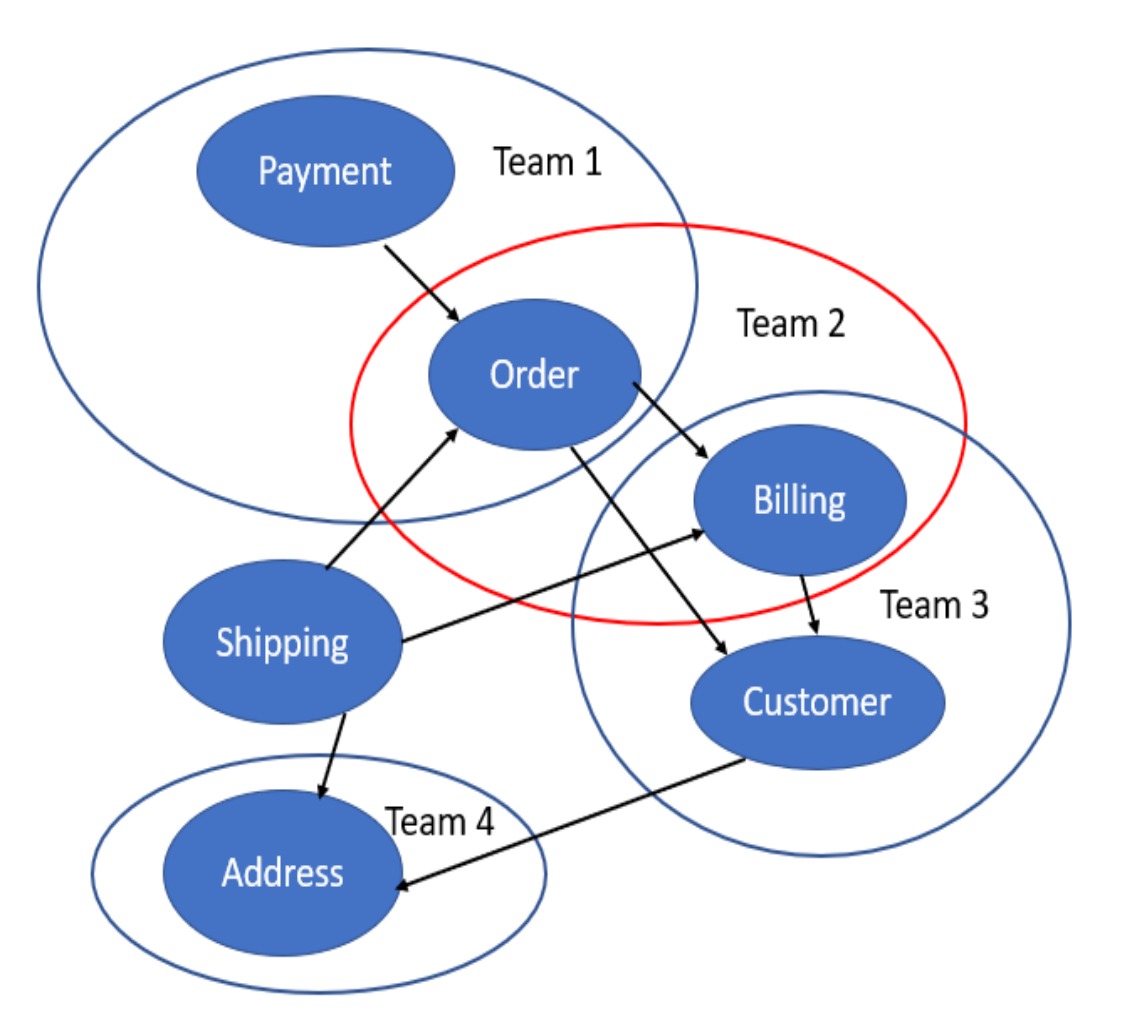


Motivation & Pain Points of Microservices Architecture

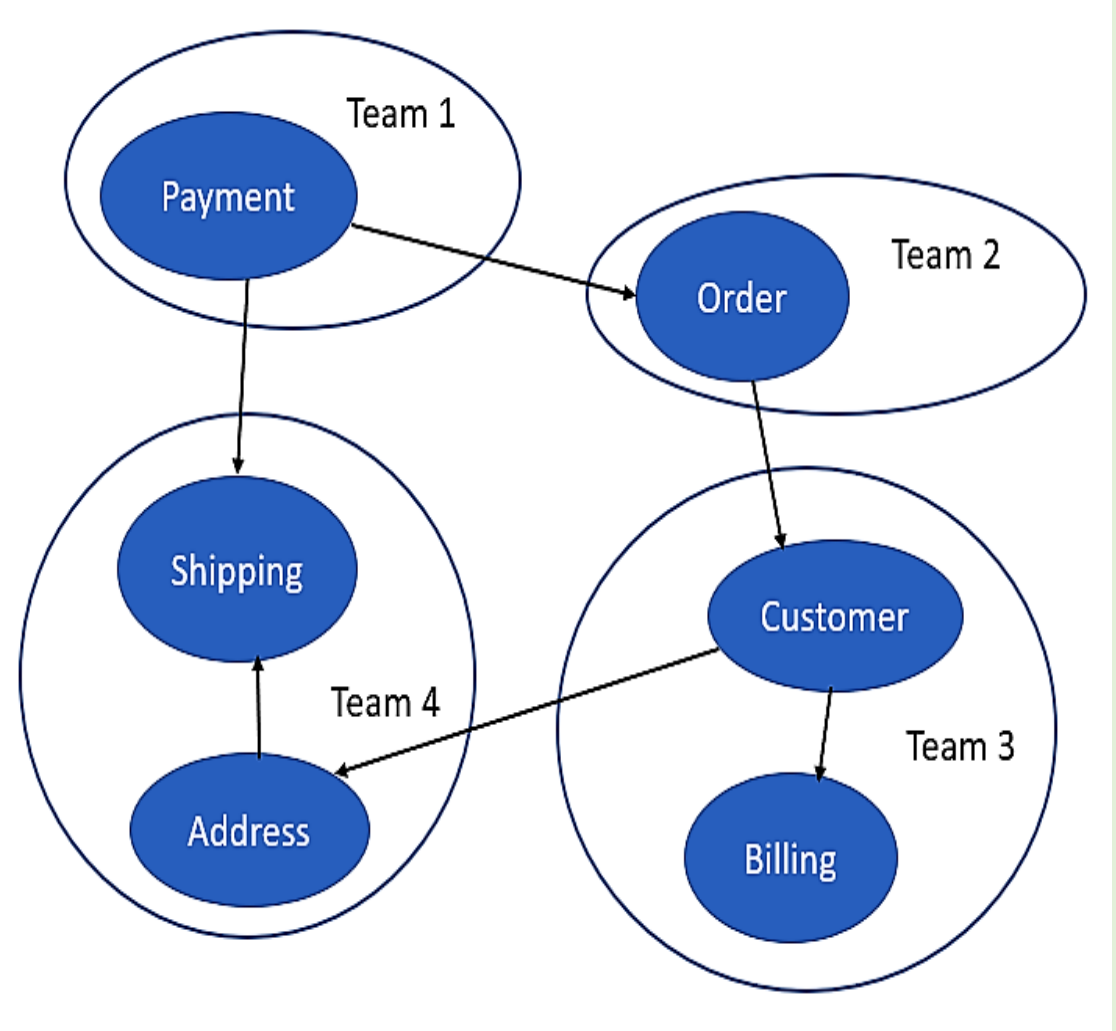
- Single Responsibility
- Speed – Time to market
- Scalability
- Easier Deployments
- Problem Isolation
- Polyglot Programming

- Cultural Change
- More Expensive
- Increased Complexity
- Less Productivity

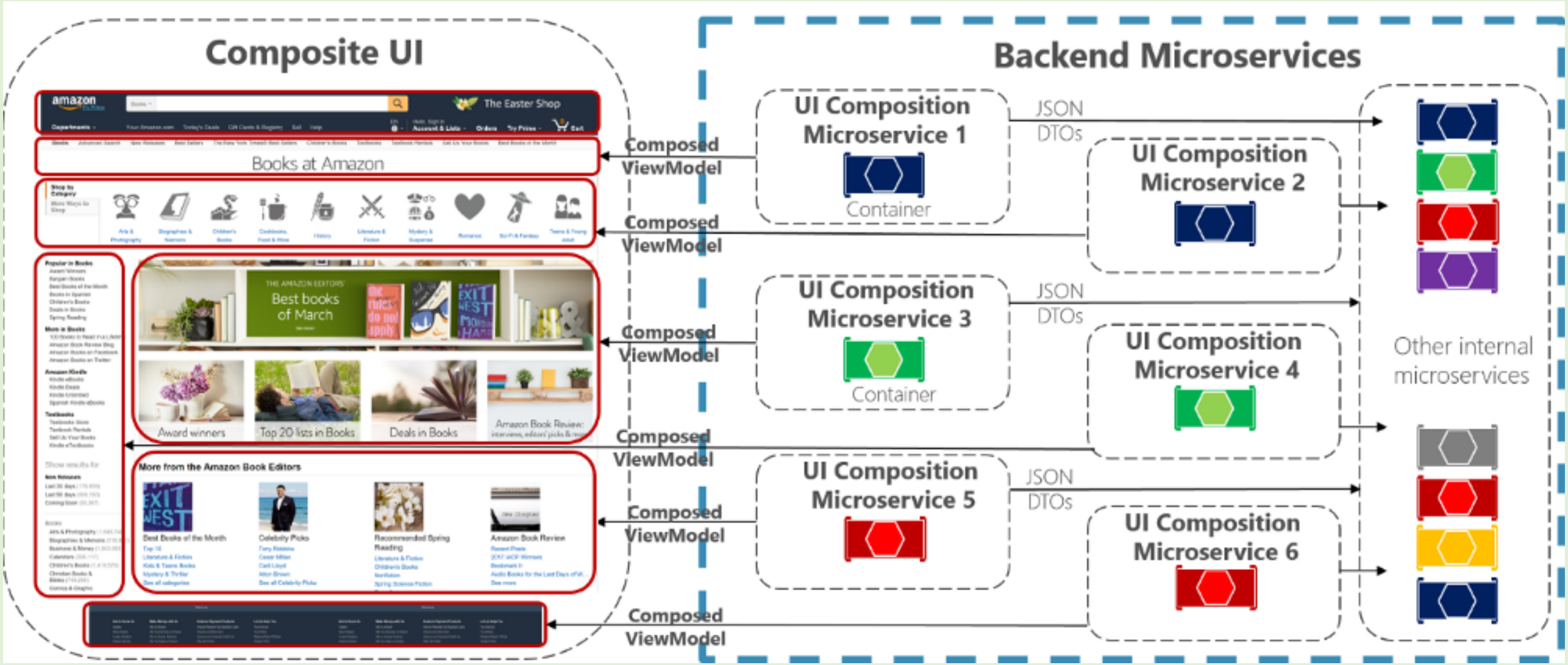
Service Ownership

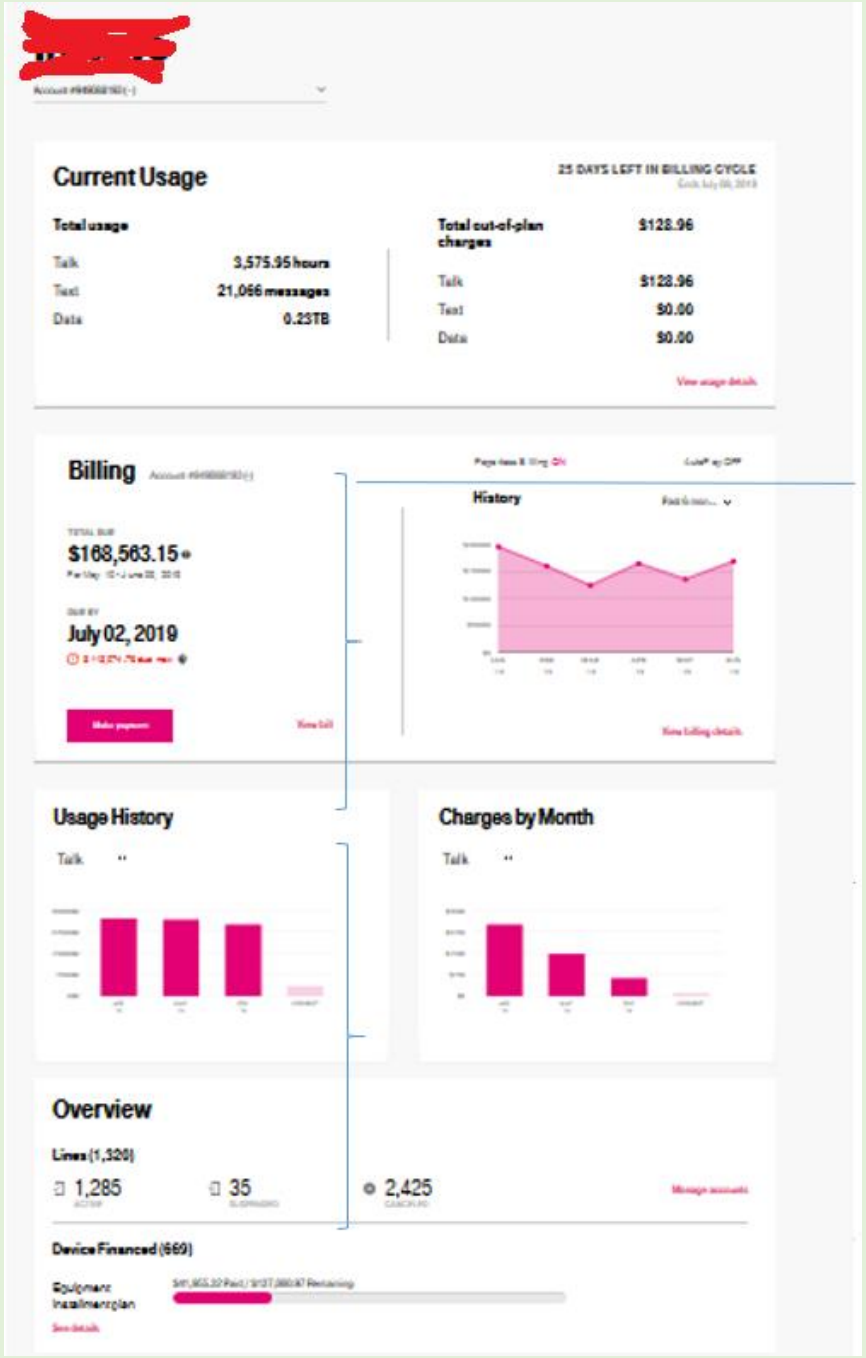


STOSA (Single Team Owned Service Architecture)

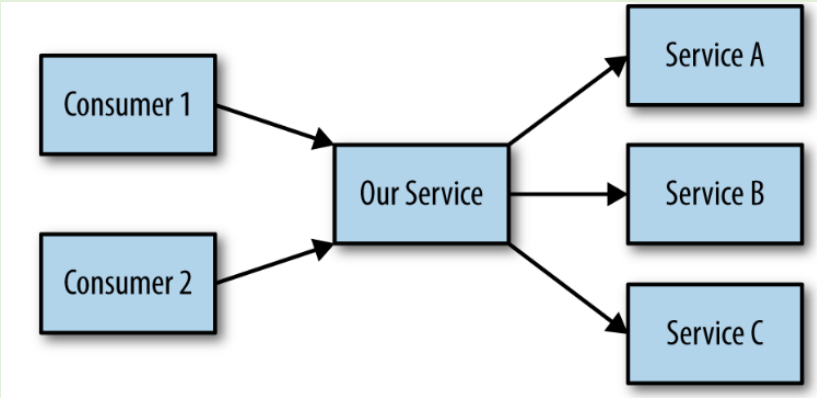


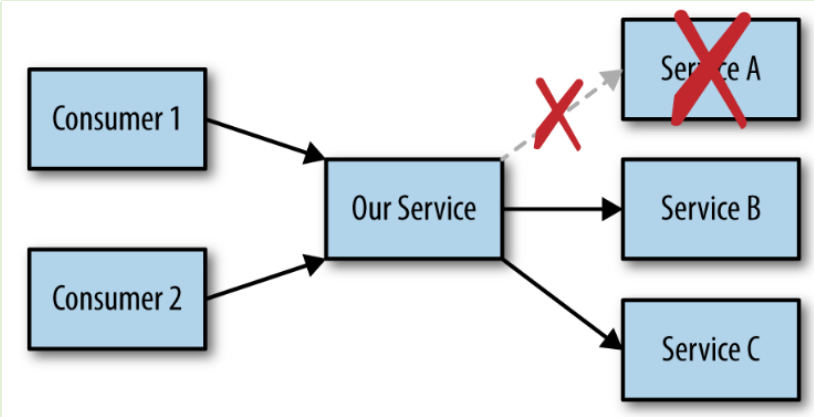
Microservices In Action




Current Usage
25 DAYS LEFT IN BILLING CYCLE
Total usage
Talk 3,575.95 hours
Text 21,066 messages
Data 0.2STB
Total out-of-plan charges $128.96
Talk $128.96
Text $0.00
Data $0.00
Billing
$168,563.15
July 02, 2019
History
Usage History
Talk
Charges by Month
Talk
Overview
Lines (1,320)
1,285
35
2,425
Device Financed (669)
Equipment Installment plan

Dealing With Service Failures





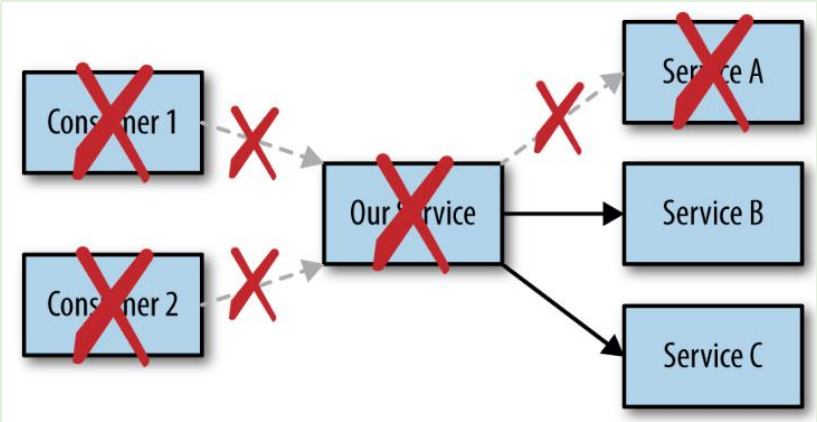


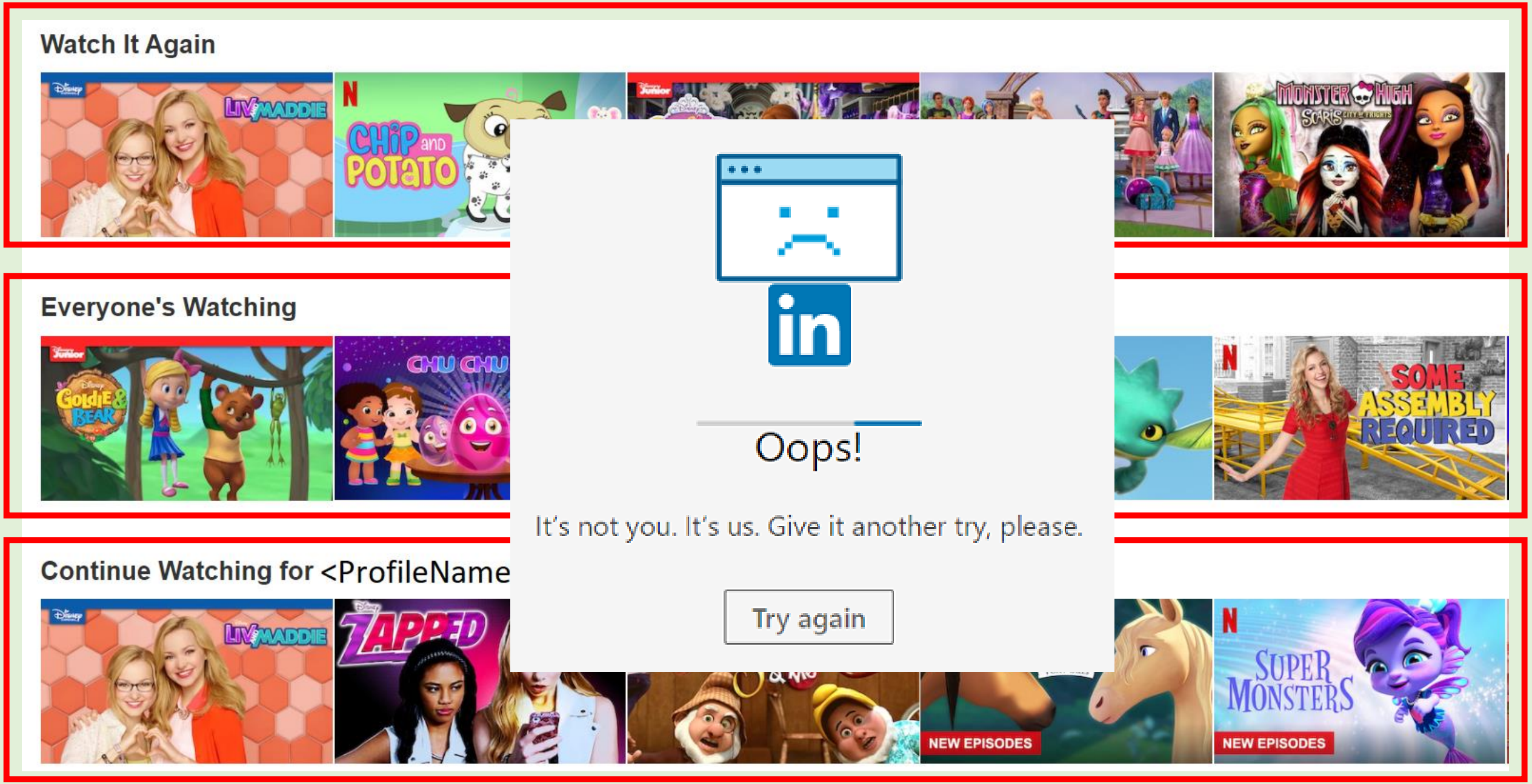
Image credit Lee A, 13

Responding to Service Failures

Response to a failure must be:

- **Understandable –** Adhere to the API Contract. Don't violate API Contract with consumers.
- **Reasonable -** Indicative to what actually happening – 2+2 shouldn't return Red, instead "Please, try again later".
- **Predictable –** Planned response for given a set of circumstances – 2+2 =4, 2/0 = Invalid Request, not 687343439898 or any random number

Handling Unpredictable Failures



2 Second Rule

Amazon.com Marketplace <payments-messages@amazon.com>
To: vidya_mct@yahoo.com

Oct 2 at 10:14 PM

amazon

Your Orders | Your Account | Amazon.com

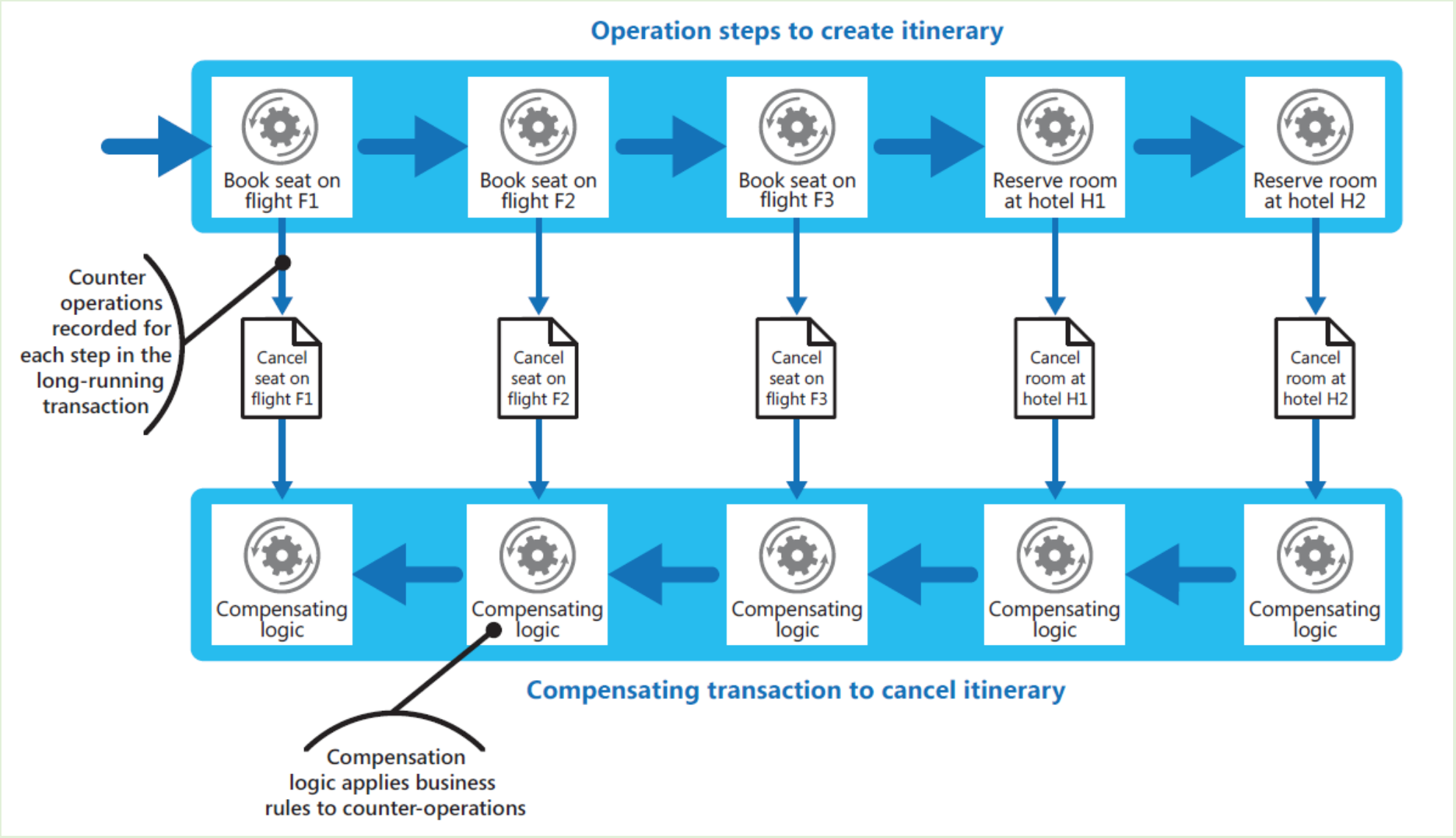
Order Cancellation

Order #113-4827284-6541824

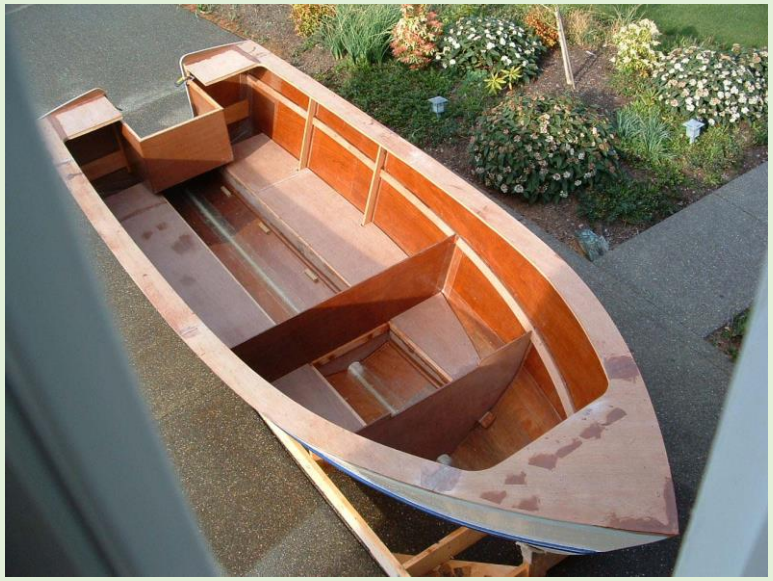
Hello vidyavrat,

We're writing to inform you that your order from Book Depository US has been canceled because the item you purchased is out of stock. Please return and place your order again at a later time. We're sorry for the inconvenience this has caused. In most cases, you pay for items when we ship them to you, so you won't be charged for items that are canceled.*

Pattern: Compensating Transaction



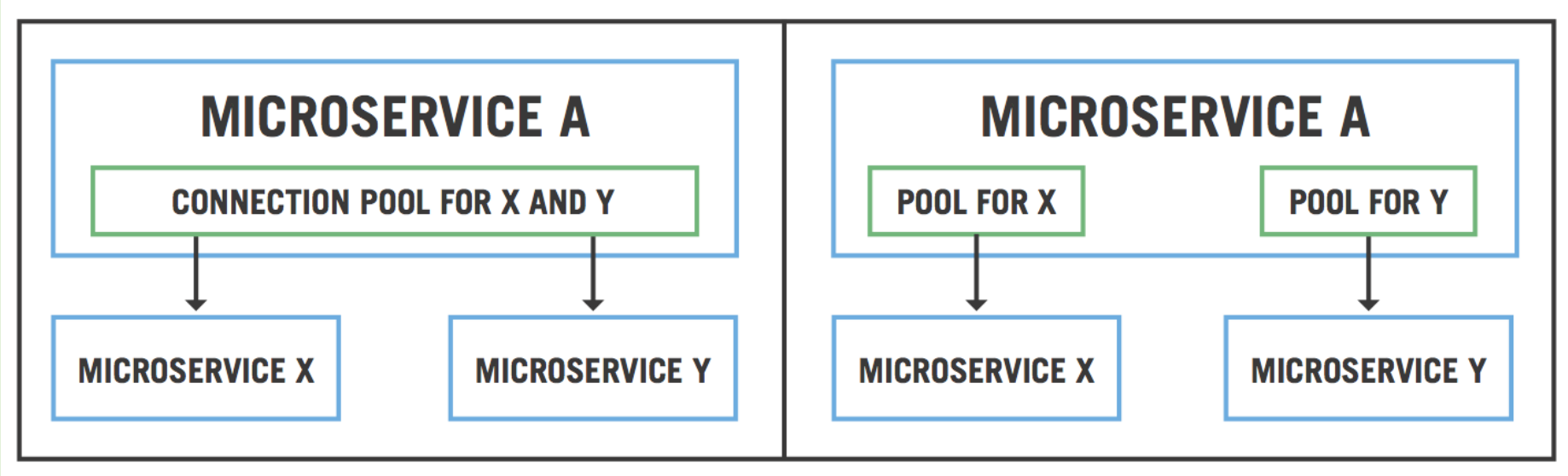
Pattern: Bulkhead



The Bulkhead pattern is a type of application design that is “**tolerant of failure**". In a bulkhead architecture, elements of an application are isolated into pools so that if one fails, the others will continue to function.

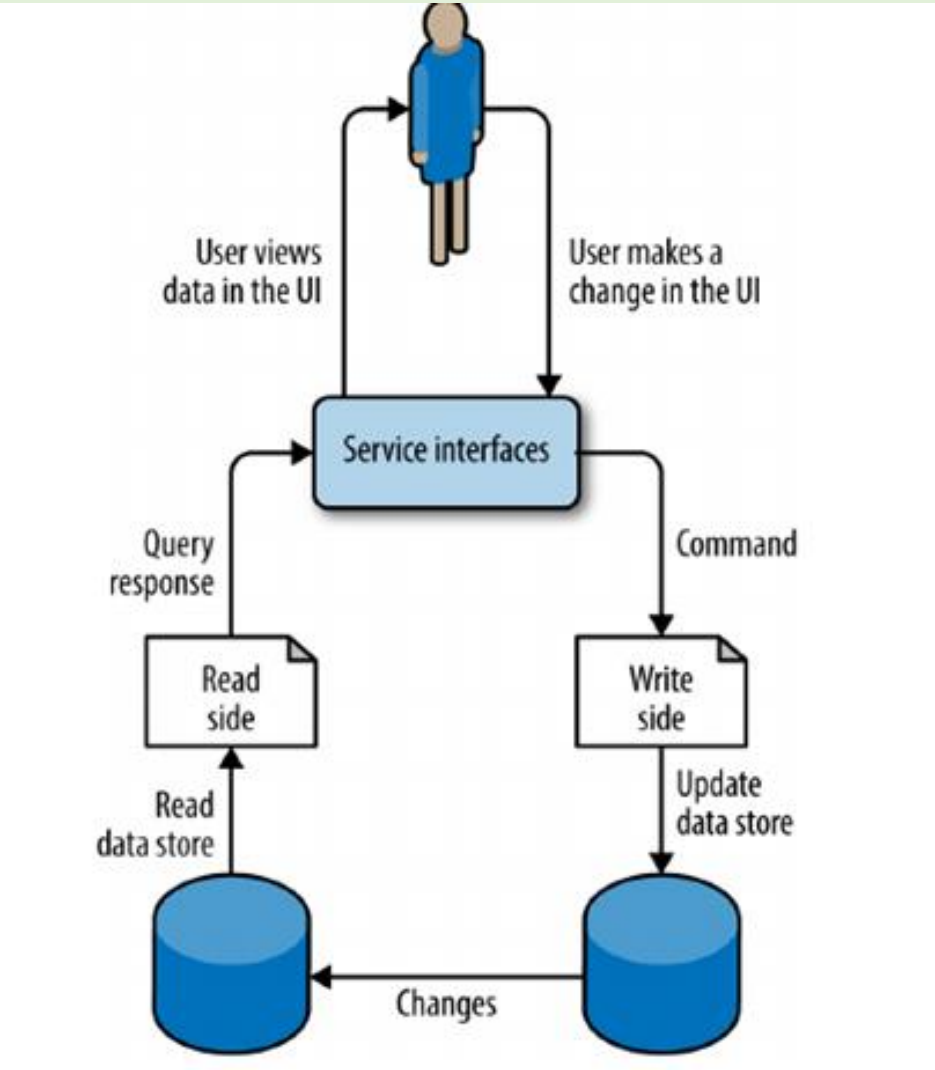
Prevents against:

- **Propagation of Failure**
- **Noisy Neighbors**
- **Unusual Demand**

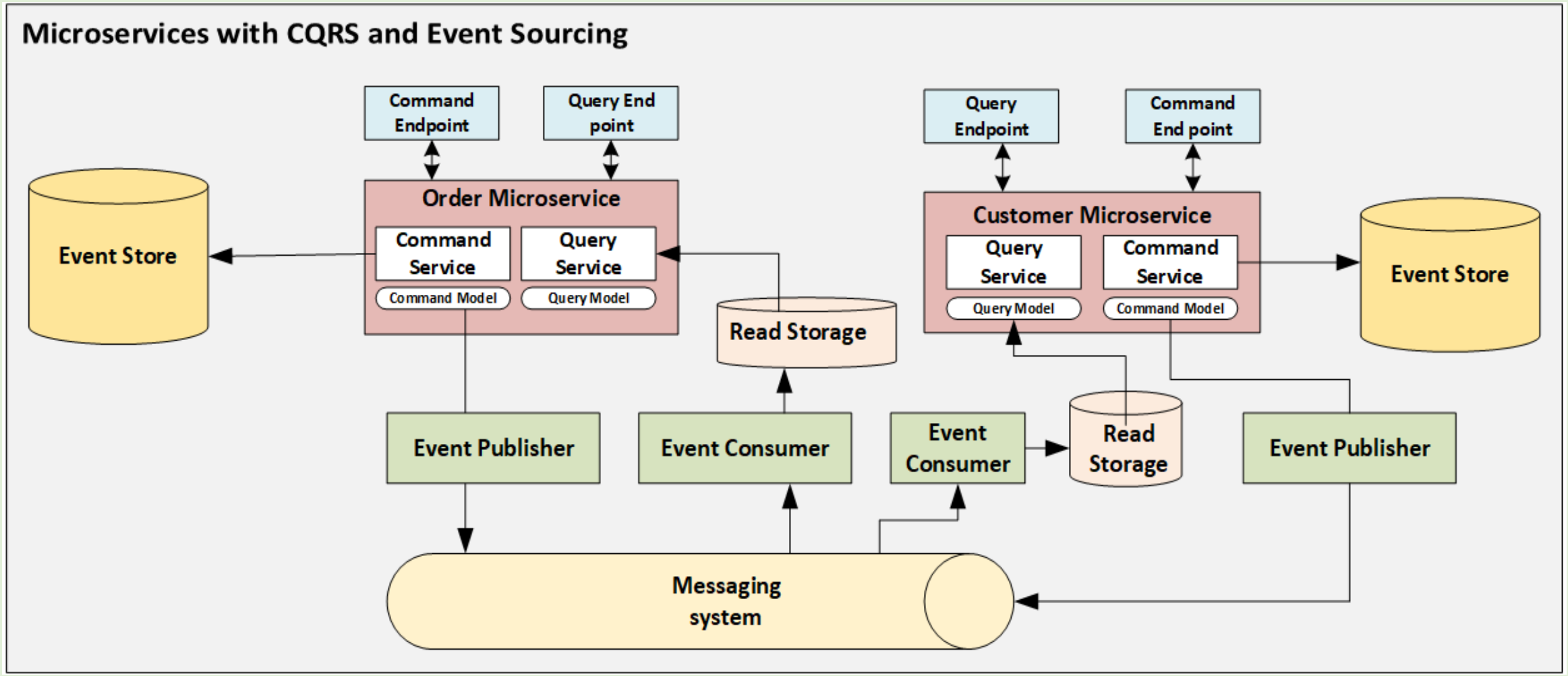


Pattern: CQRS

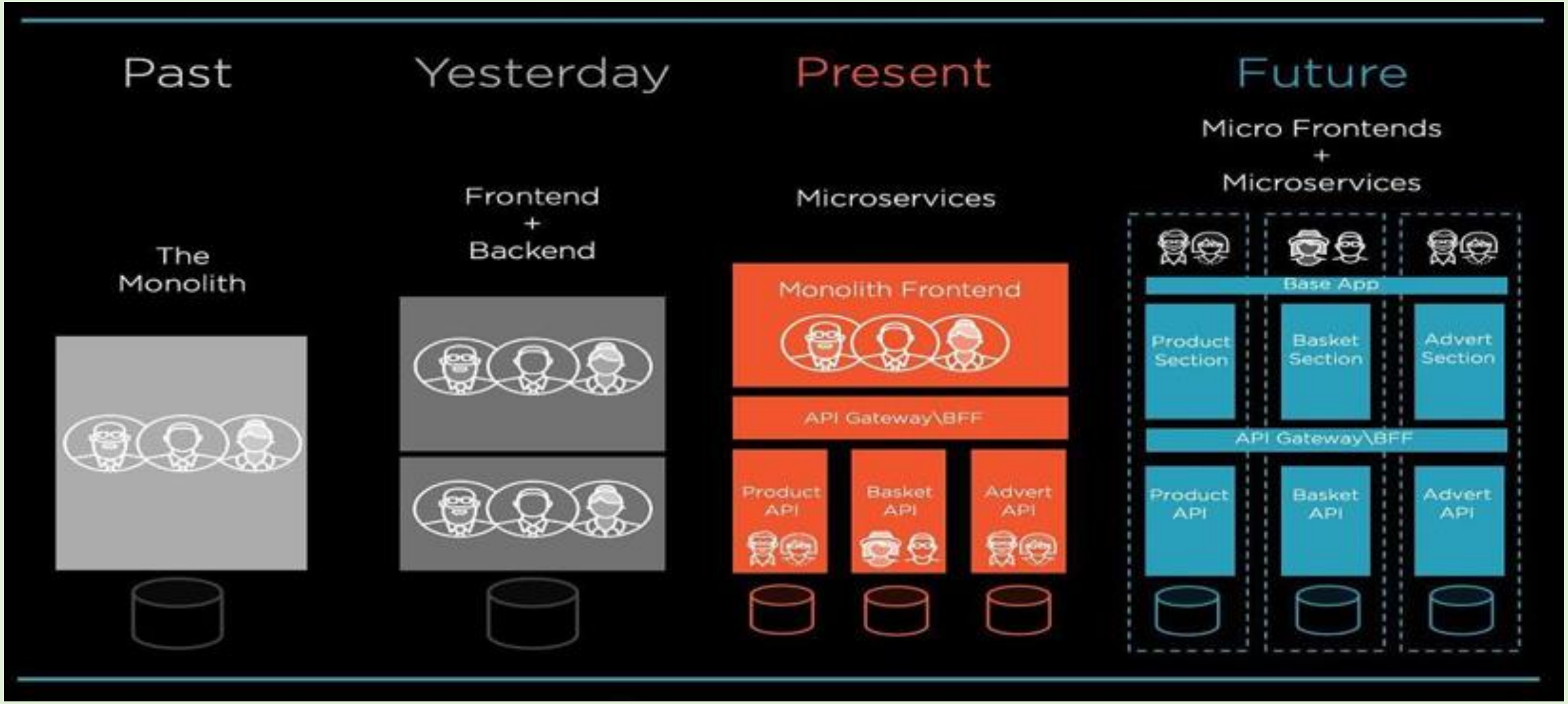
Command Query Responsibility Segregation



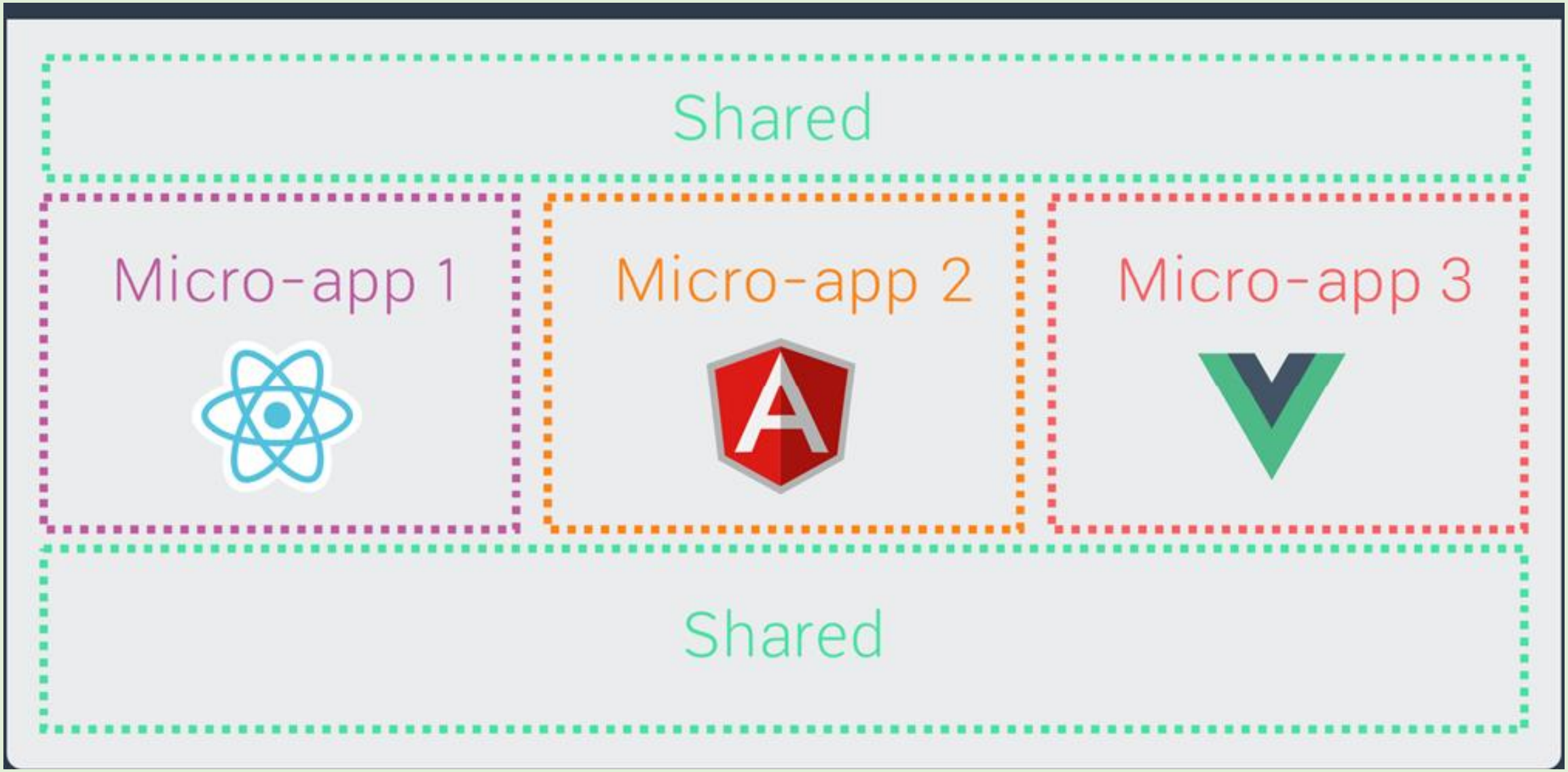
CQRS + Event Sourcing



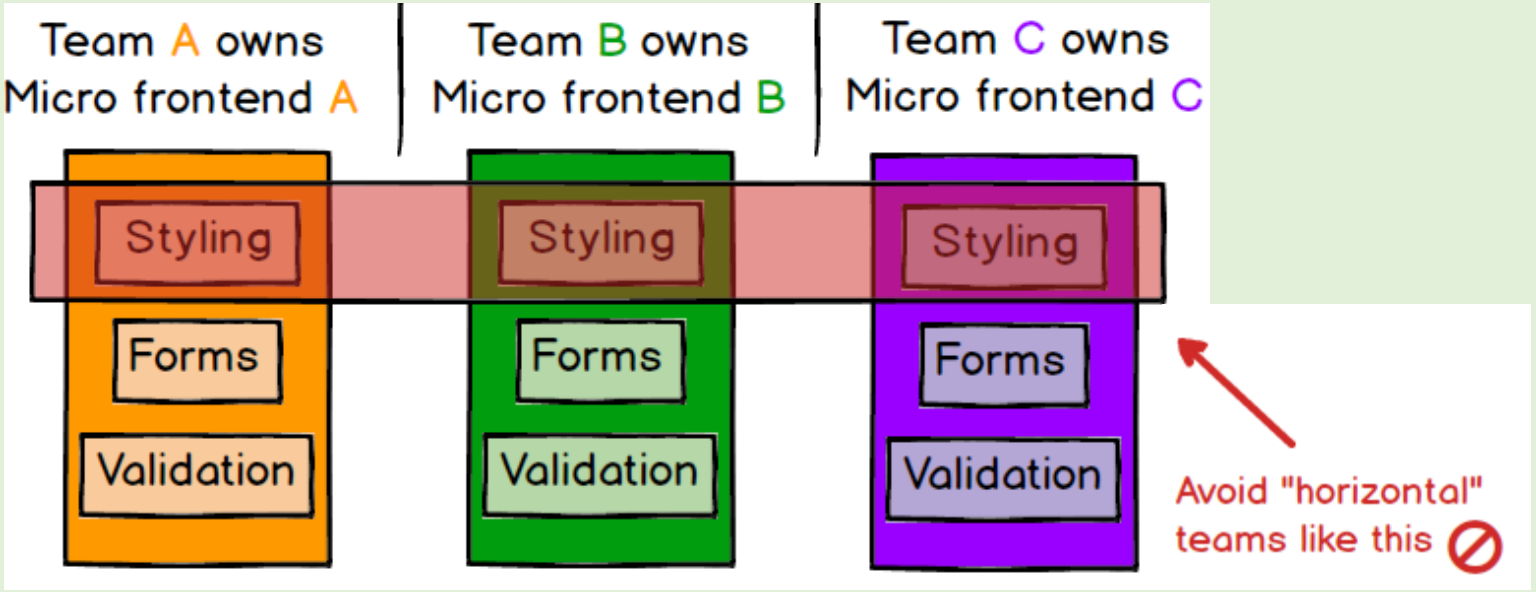
Evolutionary Architecture



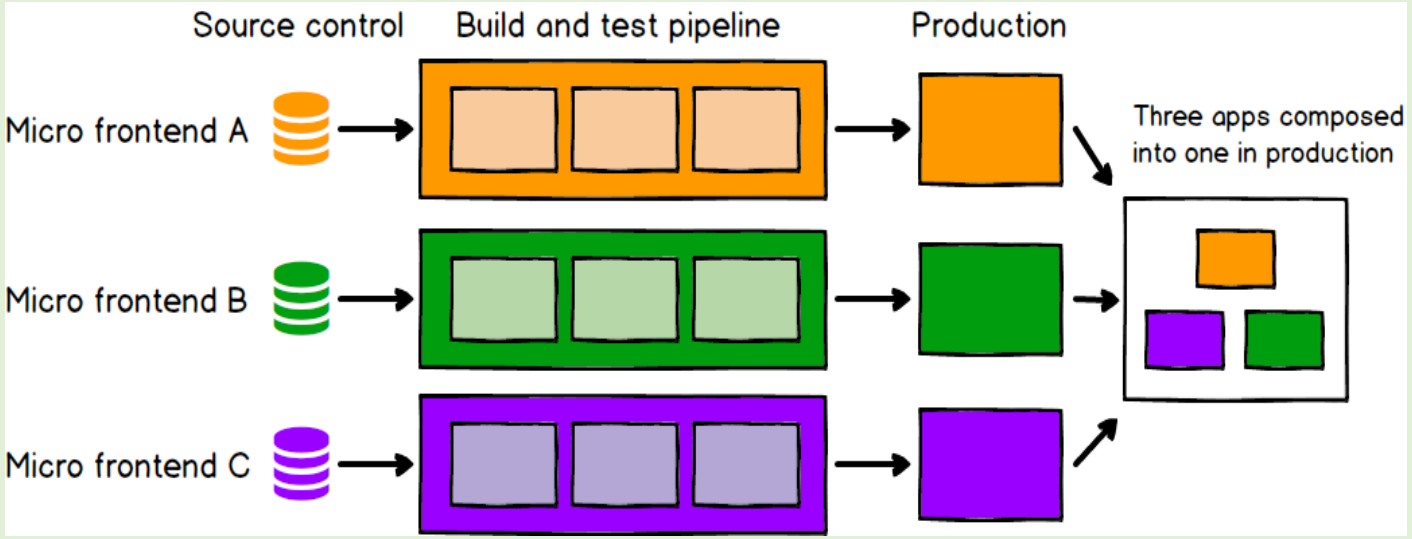
Microfrontend / Microapp



Microapp Ownership



3 product-oriented, "vertical" teams ✔



Thank You

www.MyPassionFor.Net | @dotNetAuthor

https://www.linkedin.com/in/vidyavrat/

https://github.com/vidyavrat