Event-Driven Architectures

Sébastien Mosser, 08.10.2018

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

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

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

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.*

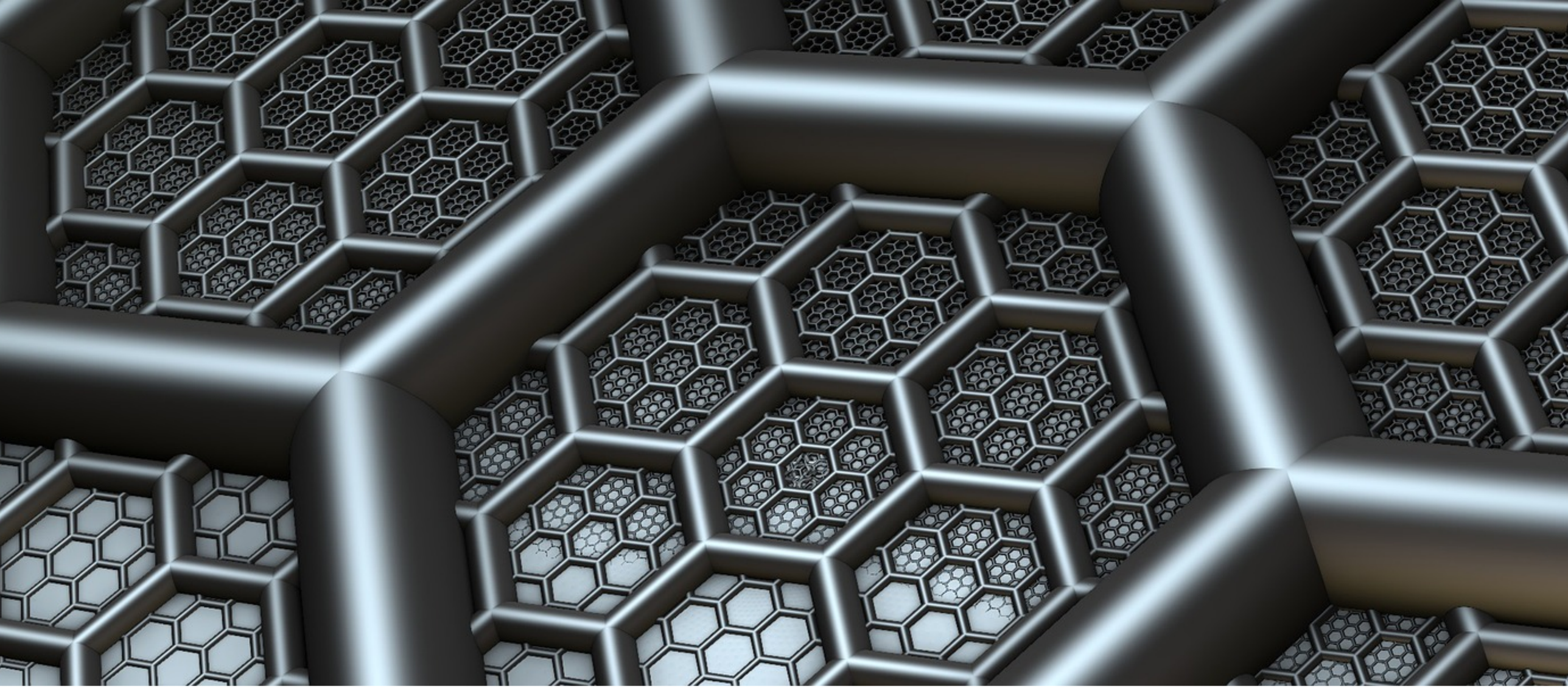
Domain Driven Design

Building event-driven Microservices with Kafka Ecosystem

Guido Schmutz
Zurich, 8.3.2018

@gschmutz

guidoschmutz.wordpress.com



Microservice Architecture (and hexagonal ones)

Microservices Architecture

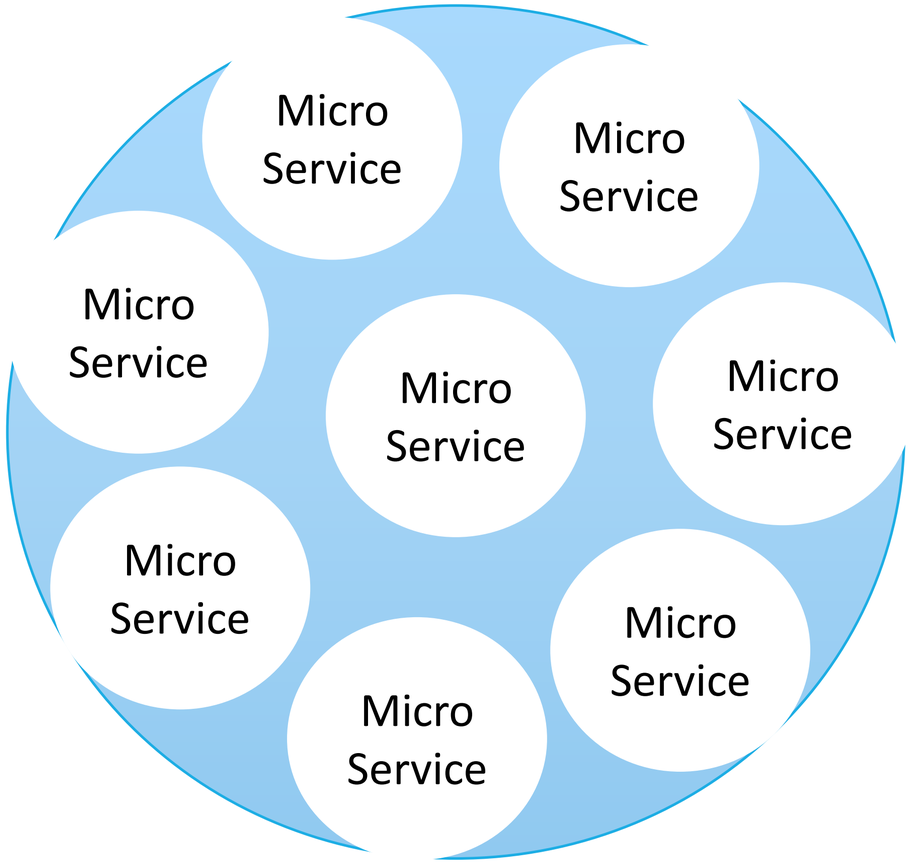
Infrastructure Architecture

- API Gateway, Service Discovery
- Event Bus / Streams
- Service Mesh

Software Design

- Domain Driven Design
- Event Sourcing & CQRS
- Functional Reactive Programming

Summary – Micro Services Intro



Key Features

1. Small in size
2. Messaging–enabled
3. Bounded by contexts
4. Autonomously developed
5. Independently deployable
6. Decentralized
7. Language–agnostic
8. Built and released with automated processes

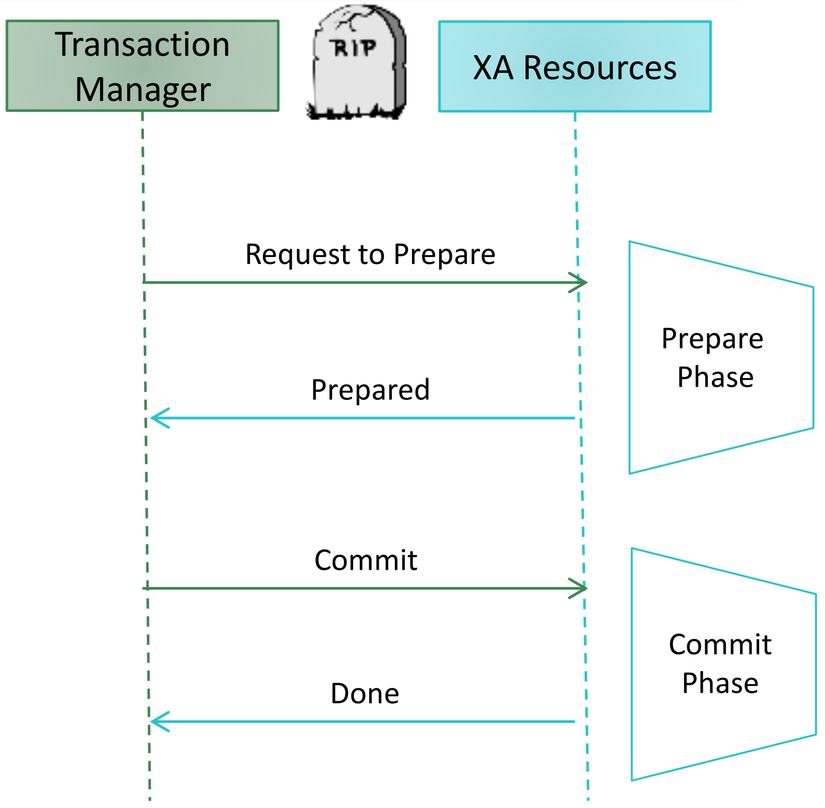
Benefits

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

Martin Fowler – Micro Services Architecture
https://martinfowler.com/articles/microservices.html
Dzone – SOA vs Micro Services : https://dzone.com/articles/microservices-vs-soa-2

Distributed Transactions : 2 Phase Commit

2 PC or not 2 PC, Wherefore Art Thou XA?



How does 2PC impact scalability?

- Transactions are committed in two phases.
- This involves communicating with every database (XA Resources) involved to determine if the transaction will commit in the first phase.
- During the second phase each database is asked to complete the commit.
- While all of this coordination is going on, locks in all of the data sources are being held.
- ***The longer duration locks create the risk of higher contention.***
- *Additionally,* ***the two phases require more database processing time than a single phase commit.***
- **The result is lower overall TPS in the system.**

Source : Pat Helland (Amazon) : Life Beyond Distributed Transactions Distributed Computing : http://dancres.github.io/Pages/

Solution : Resilient System

- Event Based
- Design for failure
- Asynchronous Recovery
- Make all operations idempotent.
- Each DB operation is a 1 PC

Handling Invariants – Monolithic to Micro Services

In a typical Monolithic App Customer Credit Limit info and the order processing is part of the same App. Following is a typical pseudo code.

In Micro Services world with Event Sourcing, it's a distributed environment. The order is cancelled if the Credit is NOT available. If the Payment Processing is failed then the Credit Reserved is cancelled.

Monolithic 2 Phase Commit

Begin Transaction

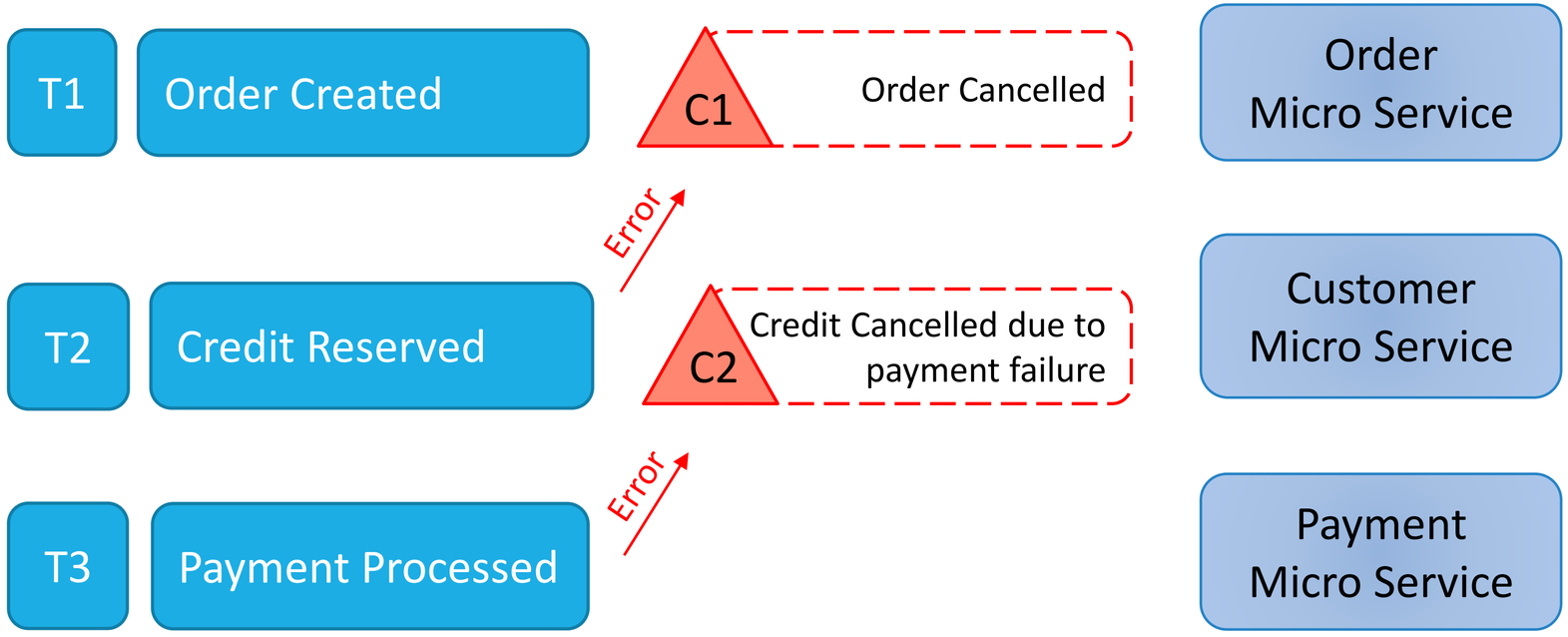
If Order Value <= Available Credit

Process Order

Process Payments

End Transaction

https://en.wikipedia.org/wiki/Invariant_(computer_science)



Scalability Best Practices : Lessons from ebay

	Best Practices	Highlights
#1	Partition By Function	• Decouple the Unrelated Functionalities. • Selling functionality is served by one set of applications, bidding by another, search by yet another. • 16,000 App Servers in 220 different pools • 1000 logical databases, 400 physical hosts
#2	Split Horizontally	• Break the workload into manageable units. • eBay's interactions are stateless by design • All App Servers are treated equal and none retains any transactional state • Data Partitioning based on specific requirements
#3	Avoid Distributed Transactions	• 2 Phase Commit is a pessimistic approach comes with a big COST • **CAP Theorem** (Consistency, Availability, Partition Tolerance). Apply any two at any point in time. • @ eBay No Distributed Transactions of any kind and NO 2 Phase Commit.
#4	Decouple Functions Asynchronously	• If Component A calls component B synchronously, then they are tightly coupled. For such systems to scale A you need to scale B also. • If Asynchronous A can move forward irrespective of the state of B • SEDA (Staged Event Driven Architecture)
#5	Move Processing to Asynchronous Flow	• Move as much processing towards Asynchronous side • Anything that can wait should wait
#6	Virtualize at All Levels	• Virtualize everything. eBay created their on O/R layer for abstraction
#7	Cache Appropriately	• Cache Slow changing, read-mostly data, meta data, configuration and static data.

Source: http://www.infoq.com/articles/ebay-scalability-best-practices

24 June 2018

Summary

1. Highly Scalable & Resilient Architecture
2. Technology Agnostic
3. Easy to Deploy
4. SAGA for Distributed Transaction
5. Faster Go To Market

In a Micro Service Architecture,

The **services tend to get simpler**, but the **architecture tends to get more complex**.

That **complexity** is often managed with **Tooling, Automation, and Process.**

Microservices ***solve*** organizational problems

~

Microservices ***cause*** technical problems

Microservices Testing Strategy

Unit Testing

A unit test exercises the smallest piece of testable software in the application to determine whether it behaves as expected.

Integration Testing

An integration test verifies the communication paths and interactions between components to detect interface defects

Component Testing

A component test limits the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract Testing

An integration contract test is a test at the boundary of an external service verifying that it meets the contract expected by a consuming service.

End 2 End Testing

An end-to-end test verifies that a system meets external requirements and achieves its goals, testing the entire system, from end to end

Source: https://martinfowler.com/articles/microservice-testing/#agenda

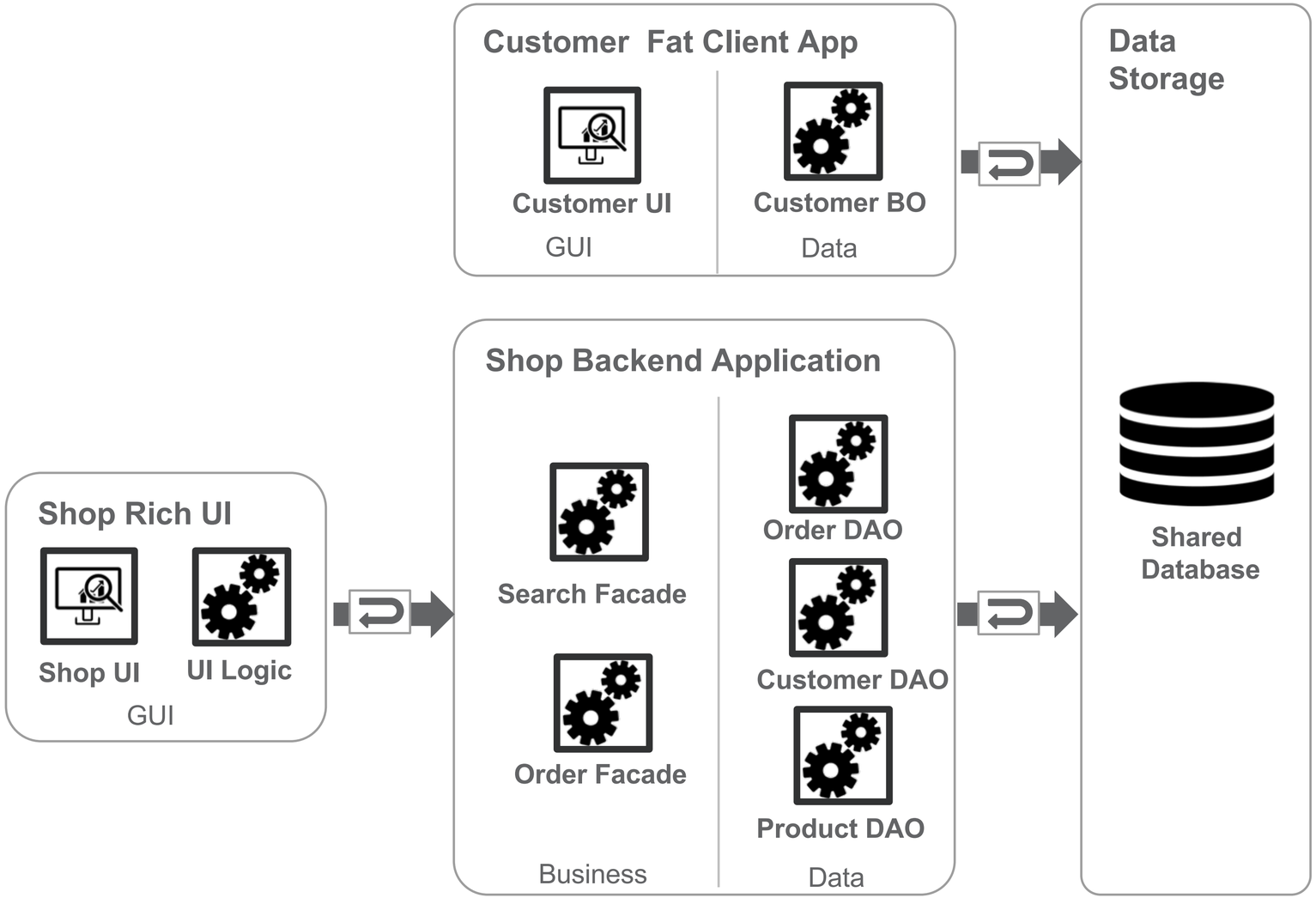
Building event-driven Microservices with Kafka Ecosystem

Guido Schmutz
Zurich, 8.3.2018

@gschmutz guidoschmutz.wordpress.com

Event-Driven Architectures & Kafka

Traditional Approach



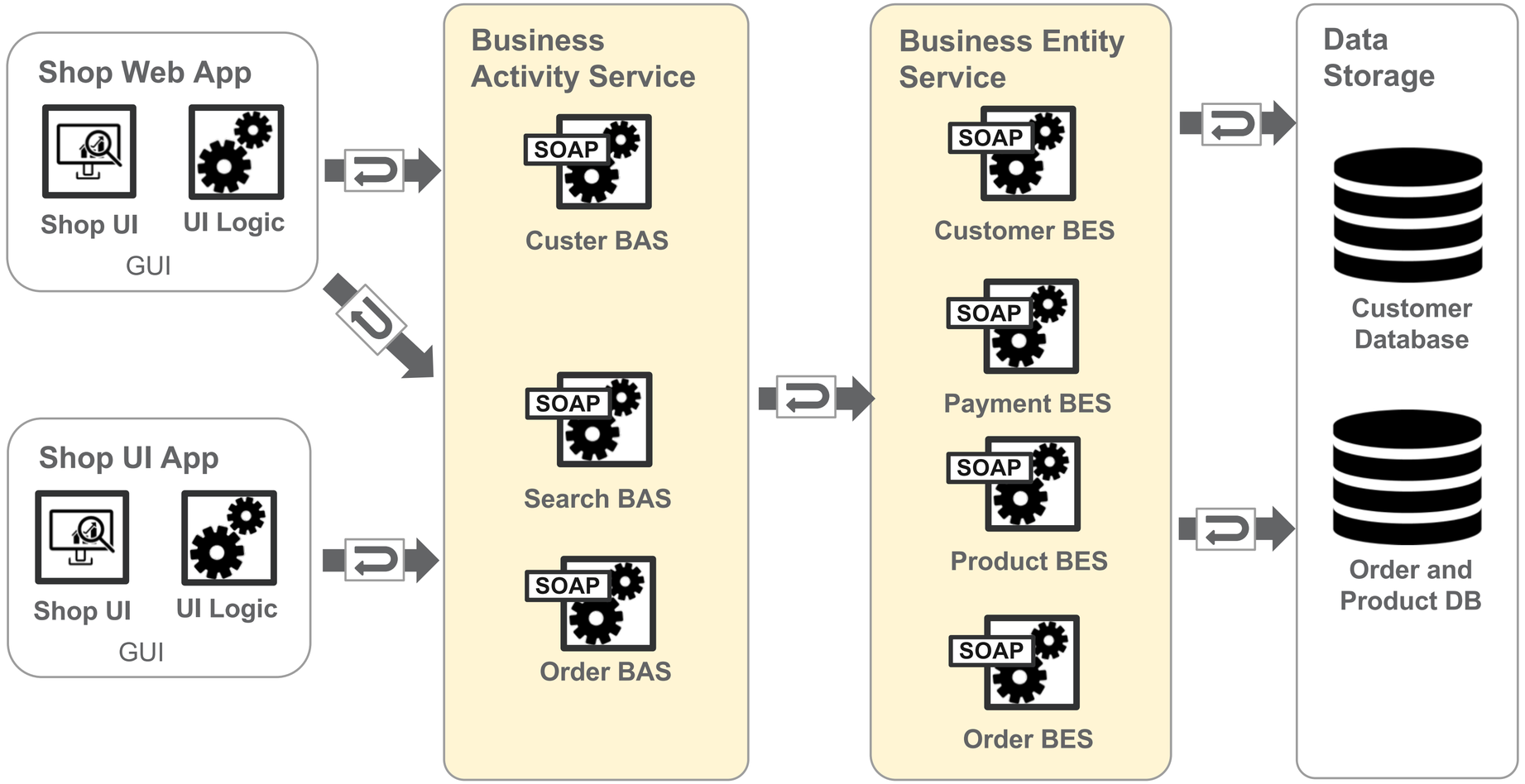
SOA Approach

Contract-first Web Services

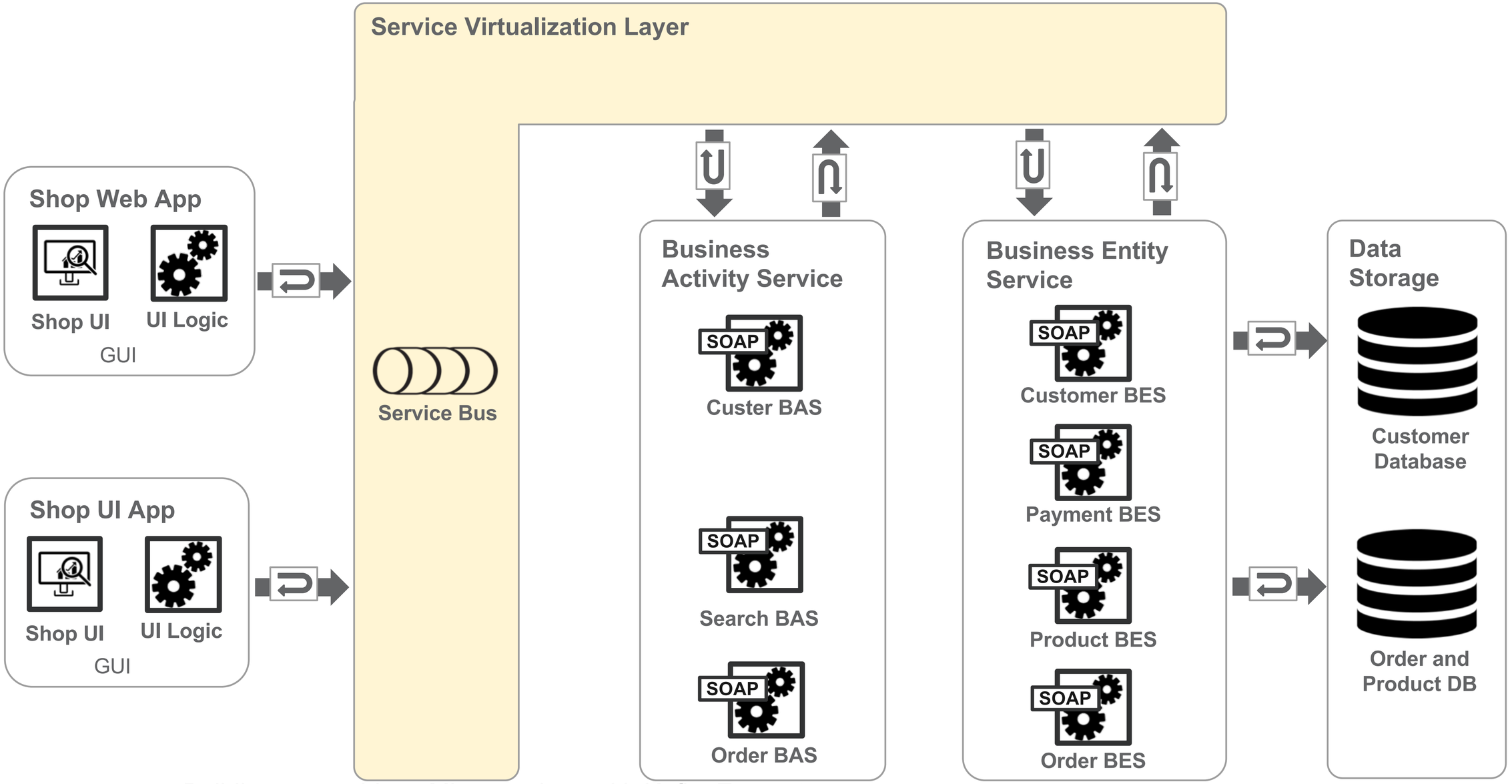
Technical layers offer their own interfaces

Reuse on each level

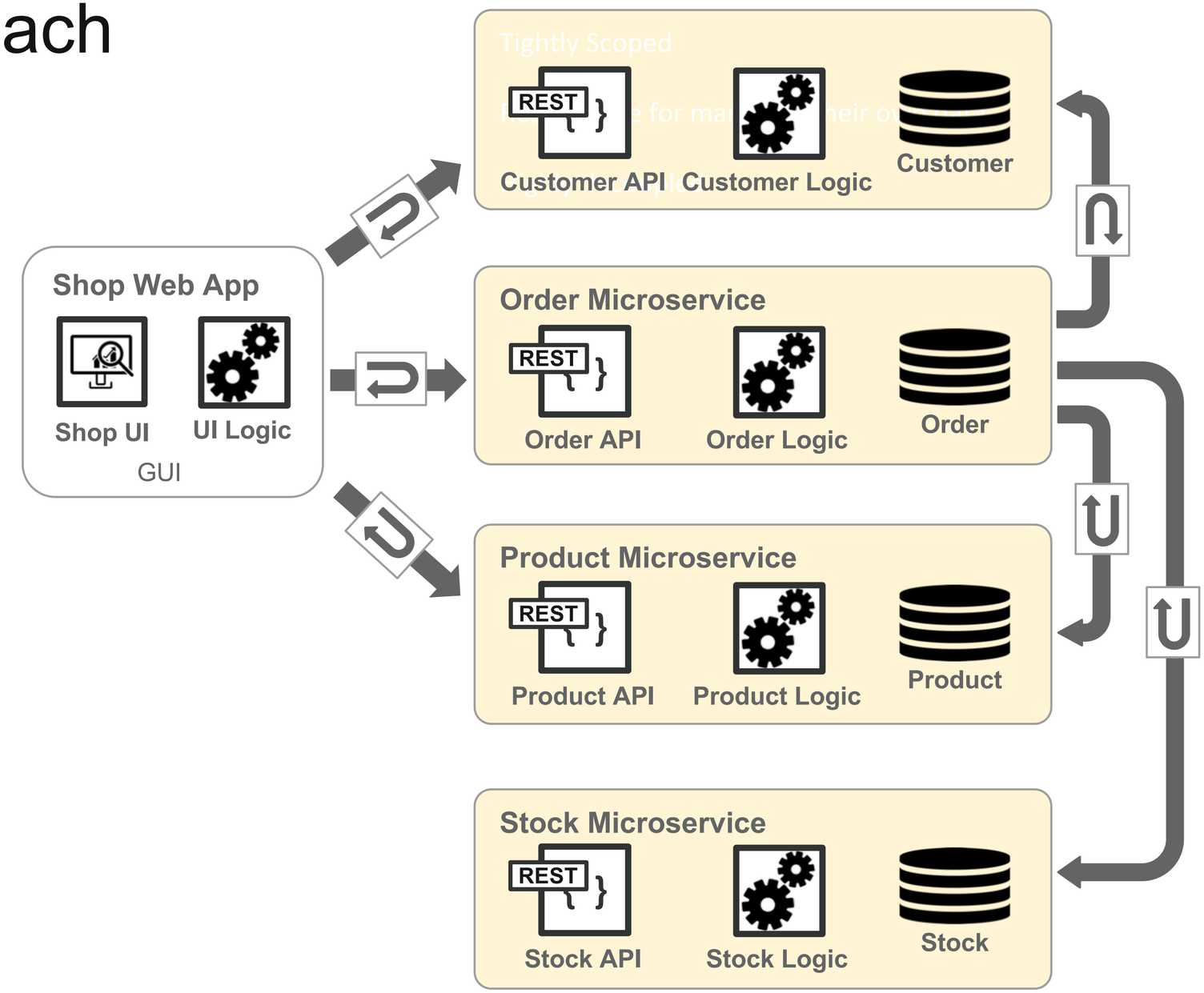
Lower layer often wraps legacy code



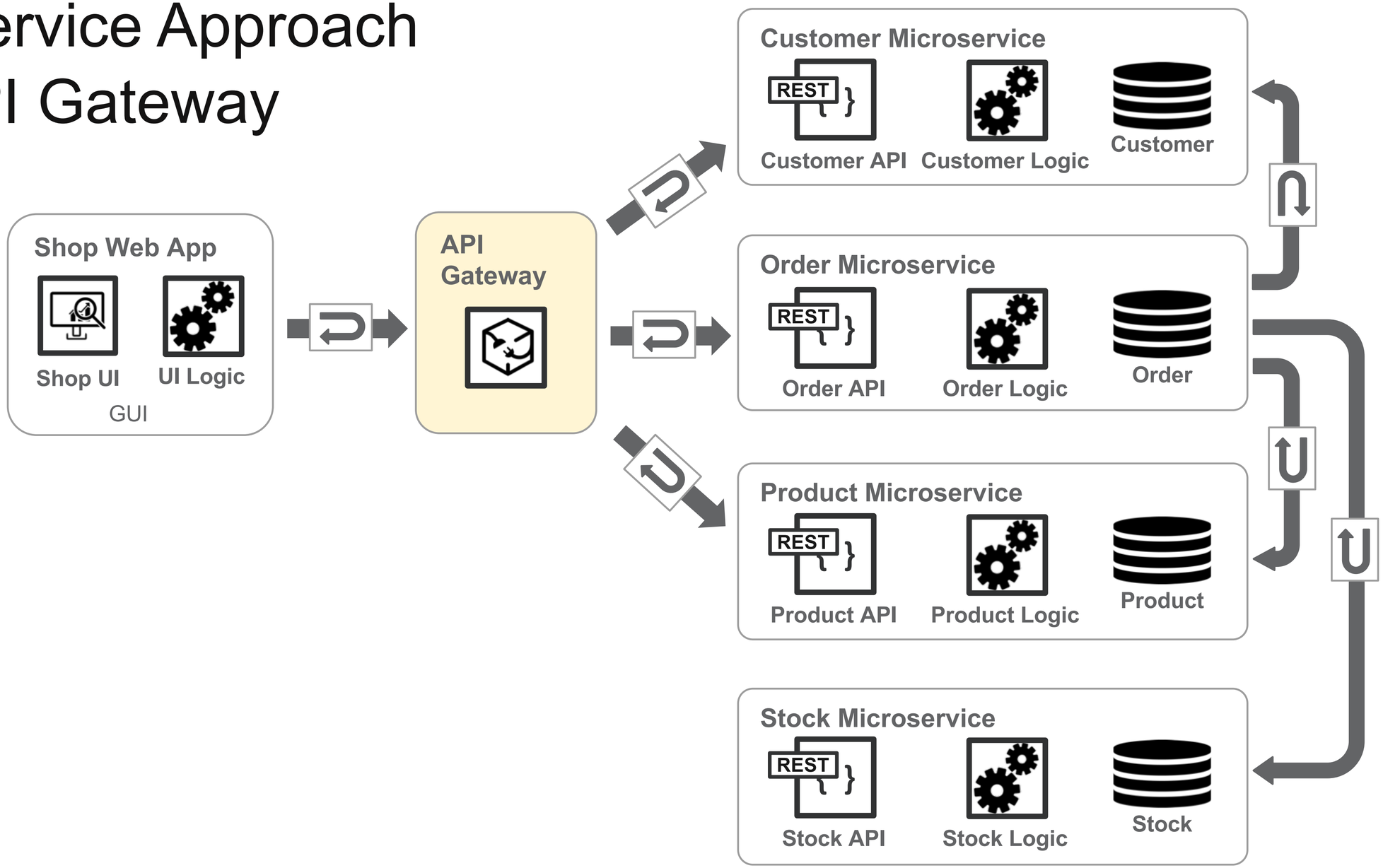
Virtualized SOA Approach



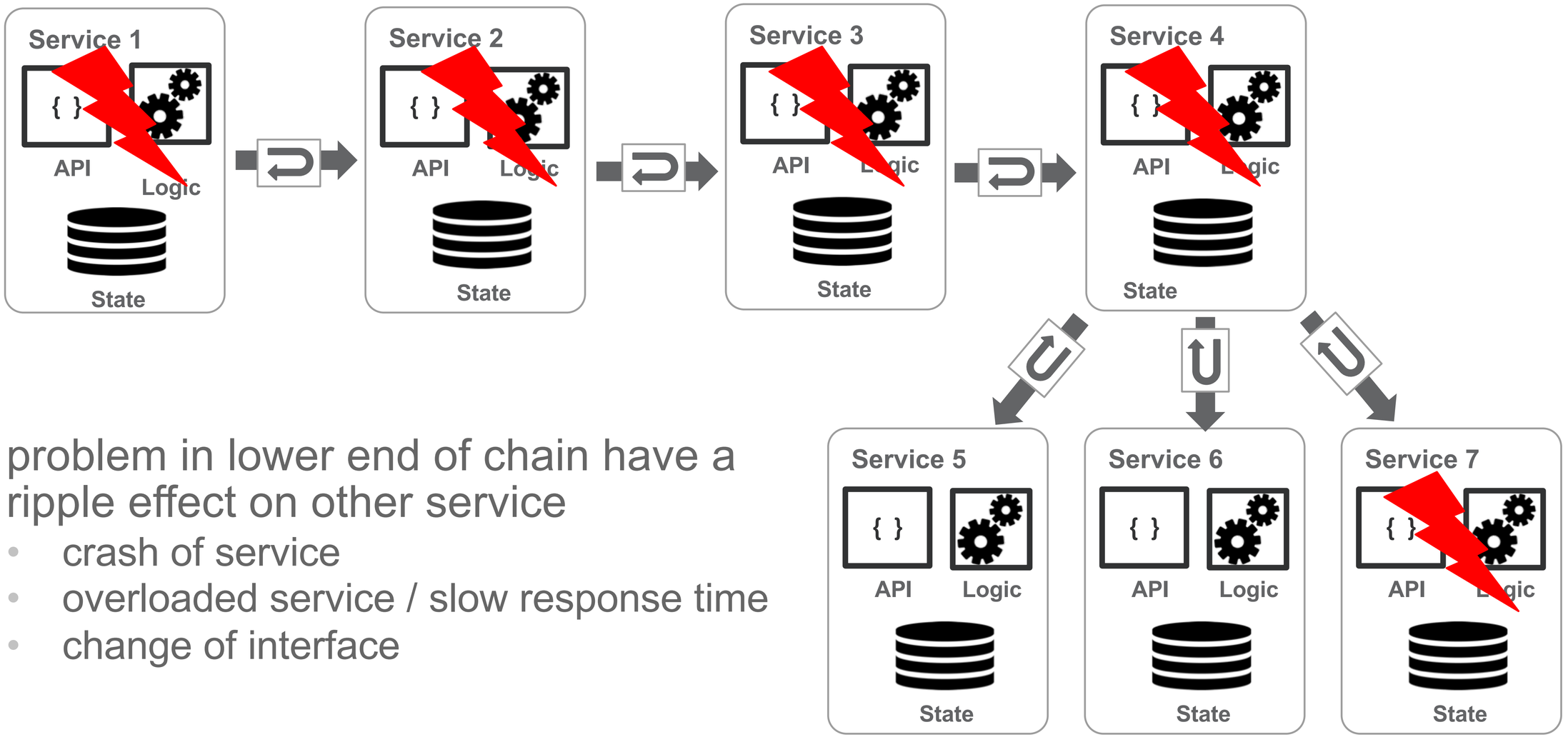
Microservice Approach



Microservice Approach with API Gateway



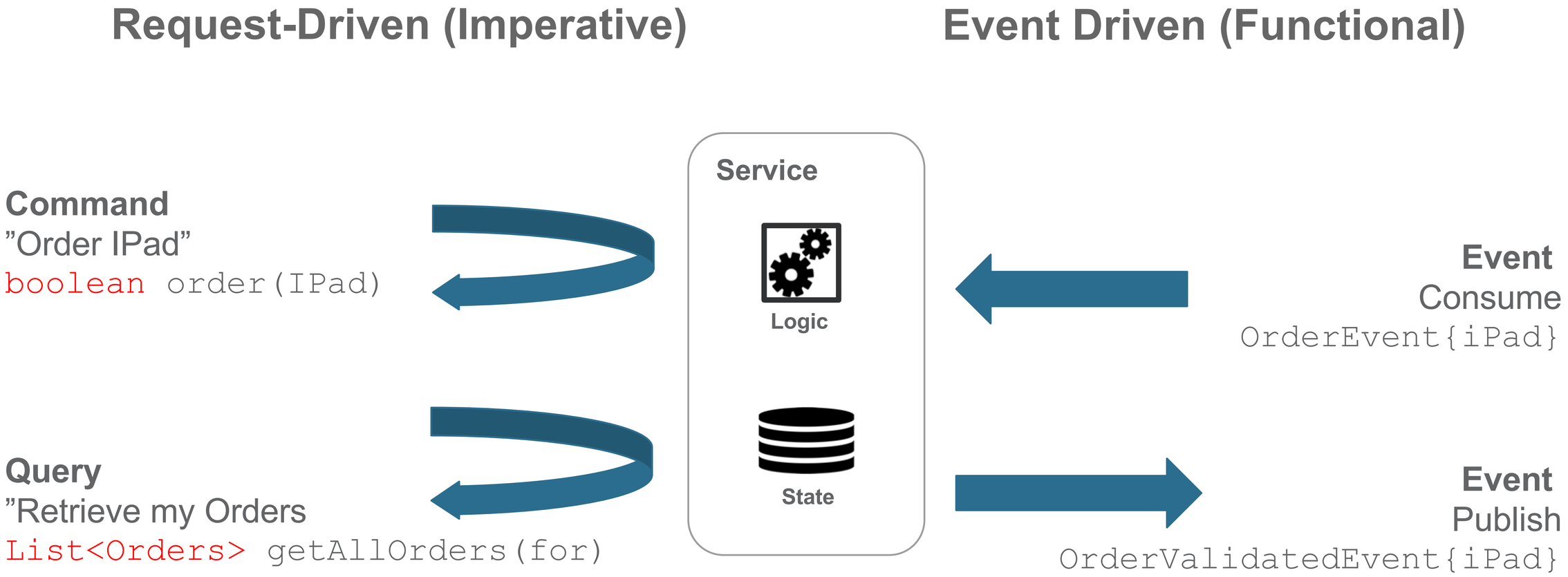
Synchronous World of Request-Response leads to tight, point-to-point couplings



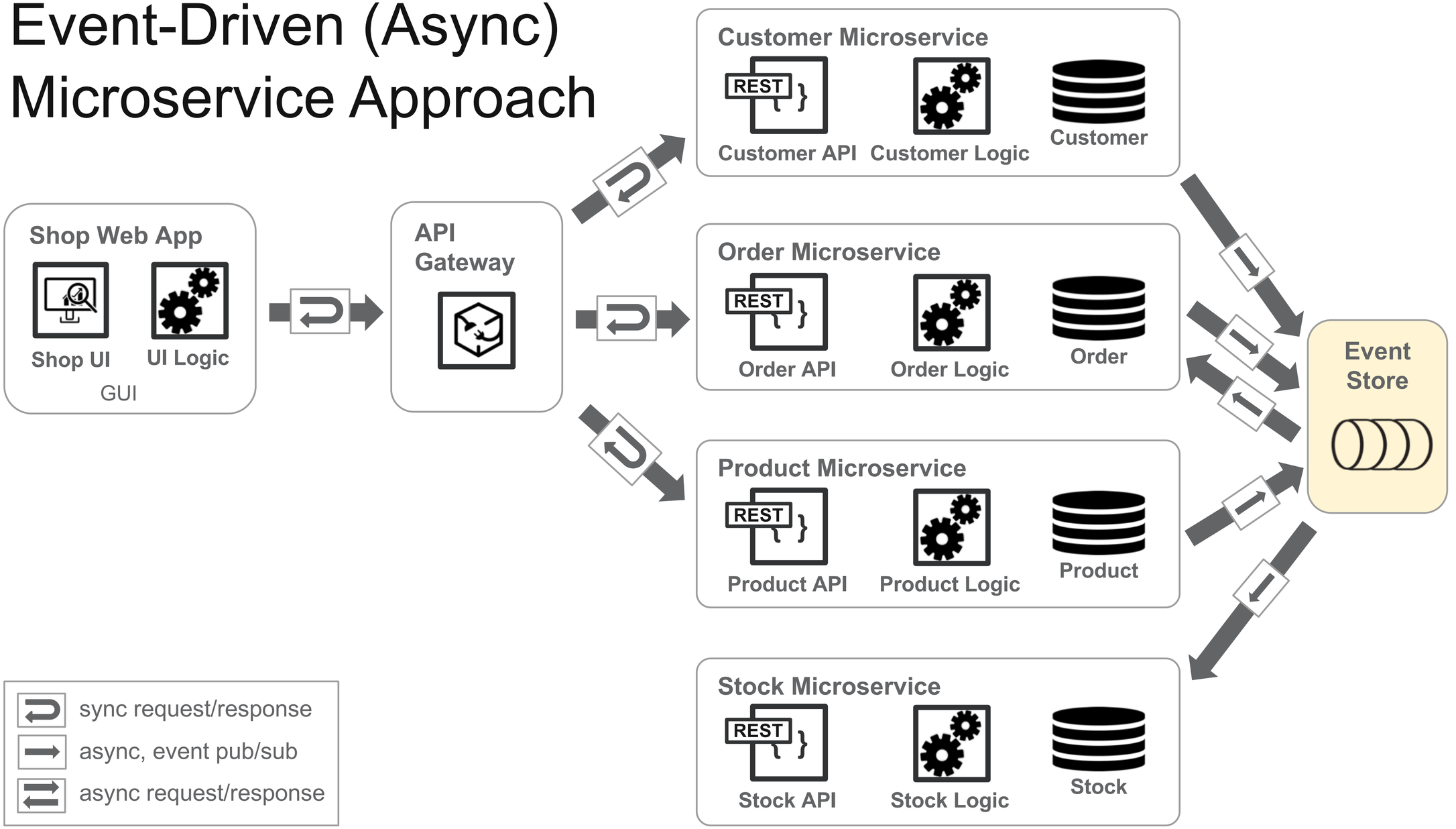
problem in lower end of chain have a ripple effect on other service

- crash of service
- overloaded service / slow response time
- change of interface

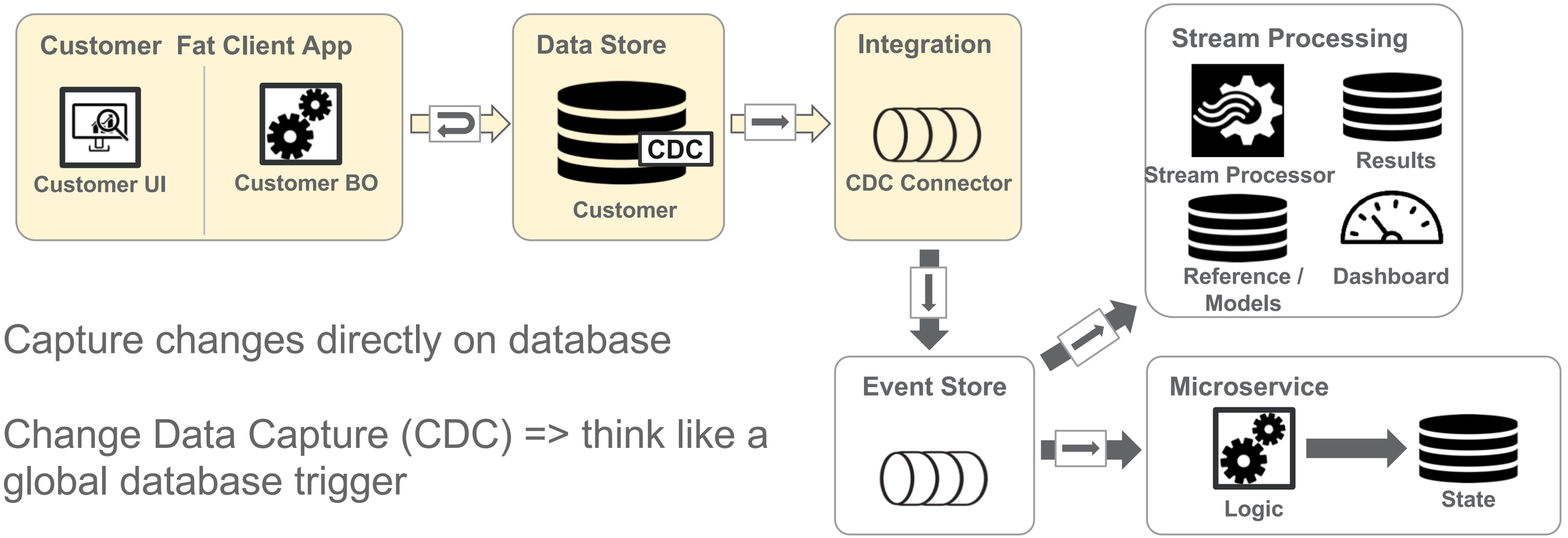
3 mechanisms through which services interact



Event-Driven (Async) Microservice Approach



Integrate existing systems through CDC



Capture changes directly on database

Change Data Capture (CDC) => think like a global database trigger

Transform existing systems to event producer

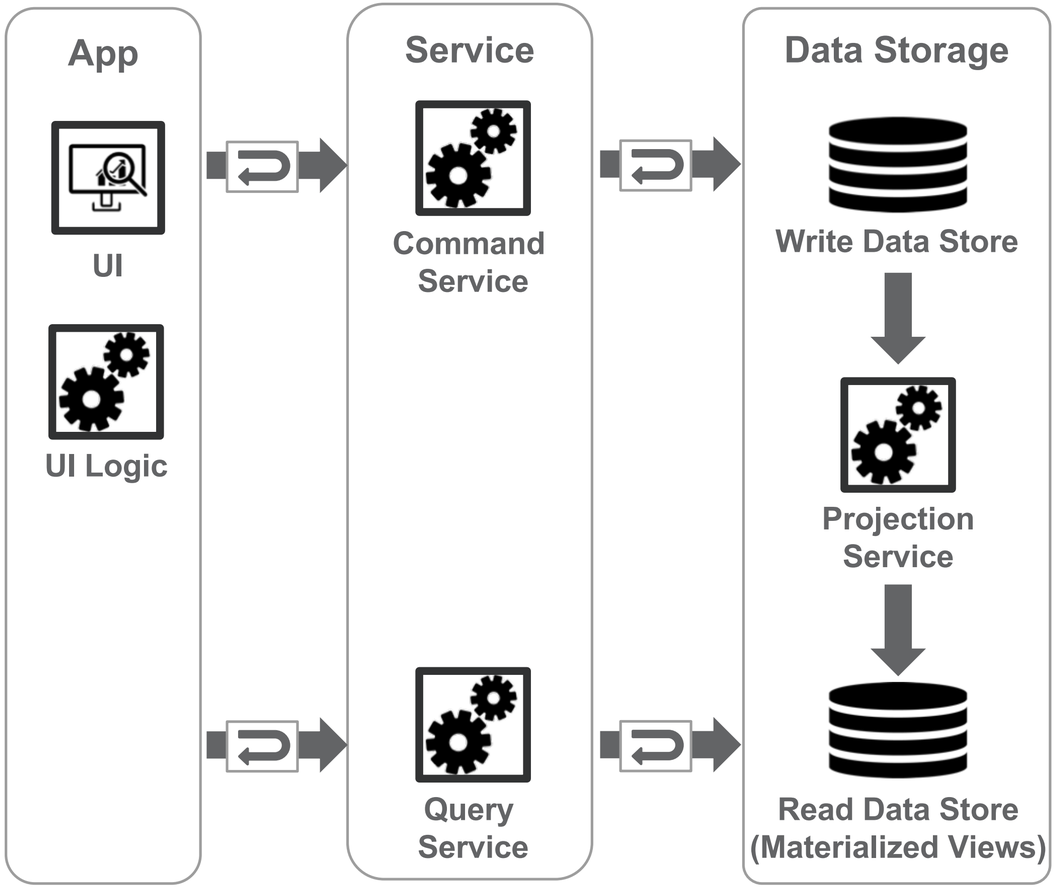
Command Query Responsibility Segregation (CQRS)

Optimize different nonfunctional requirements for **read** and **write** behavior

split between

- **commands** that trigger changes in state
- **queries** that provide read access to the state of resources

support services with **higher performance and capacity requirements** for reading data than for writing data



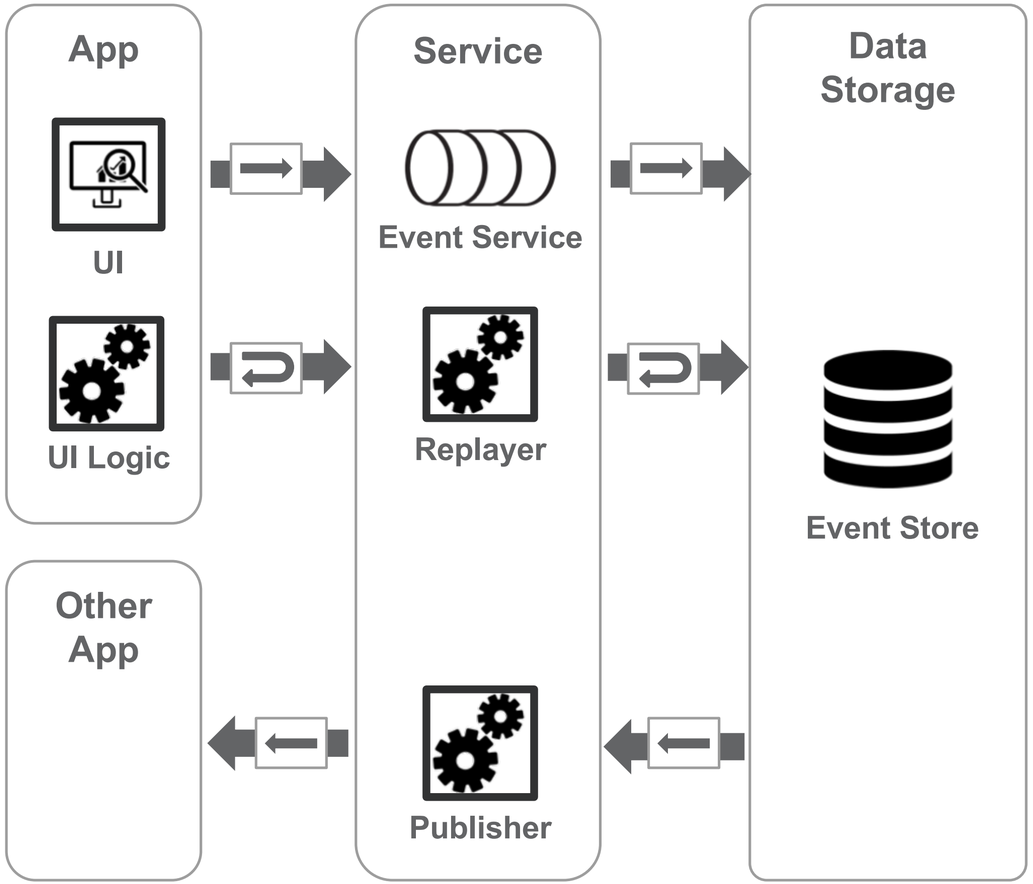
Event Sourcing

persists the state of a business entity as a **sequence of state-changing events**

Whenever state of business entity changes, a new event is appended to the list of events

Saving an event is a single operation and is inherently atomic

The application reconstructs an entity's current state **by replaying the events**

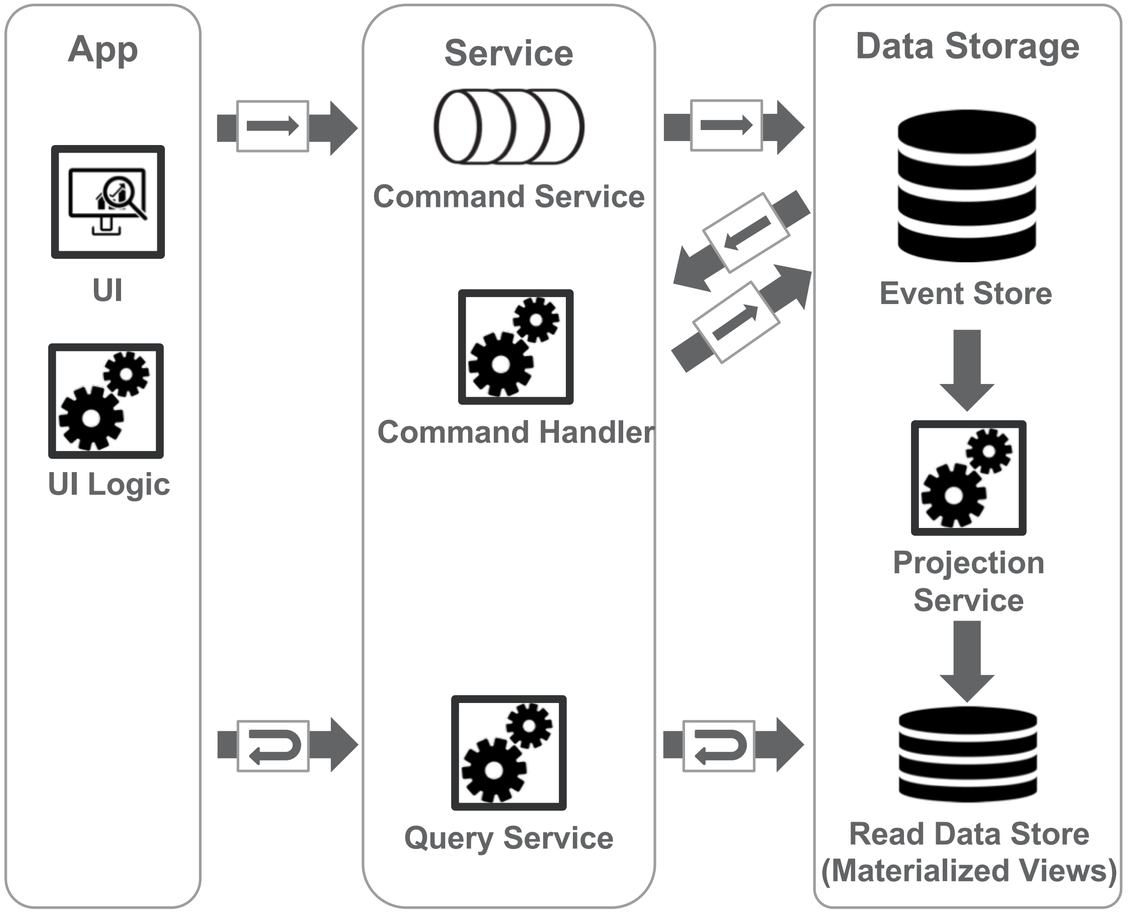


Event Sourcing & CQRS

Event sourcing is commonly combined with the CQRS pattern

materializing views from the stored events

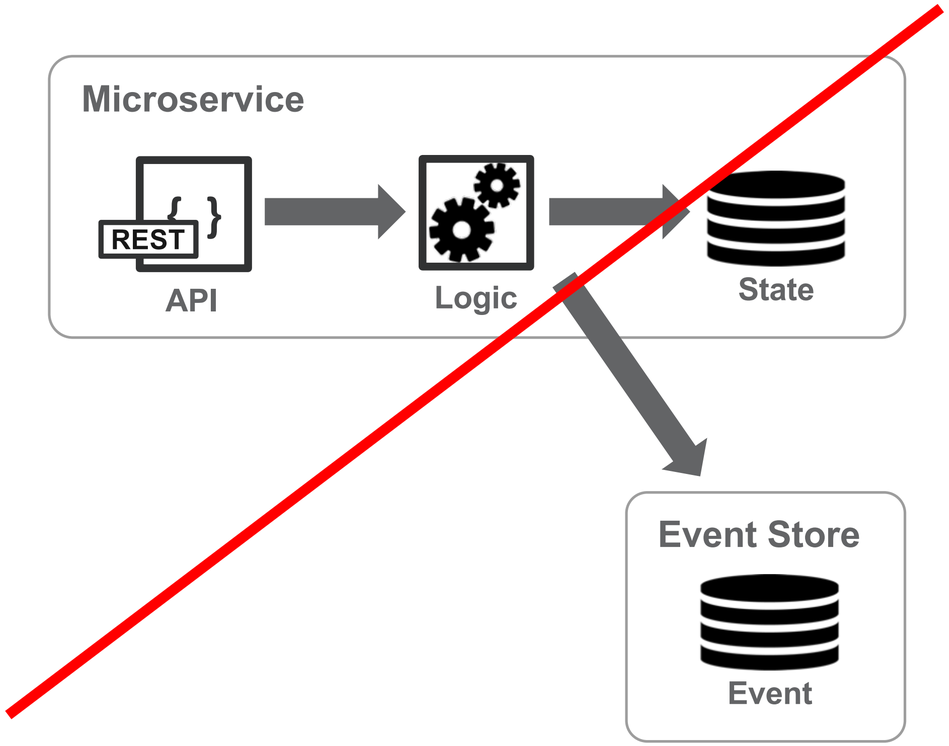
Optionally Commands can be stored in event store and transformed into events by the command handler



Have only one „source of truth“

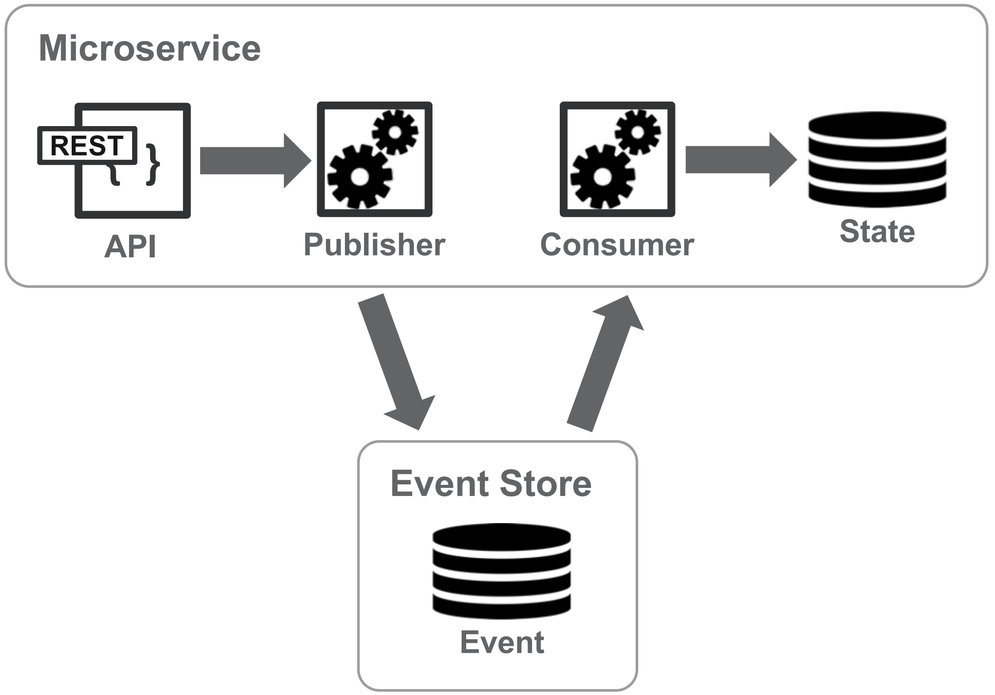
Avoid double write!

- Would need distributed transactions



Write Event first then consume it from same micro service

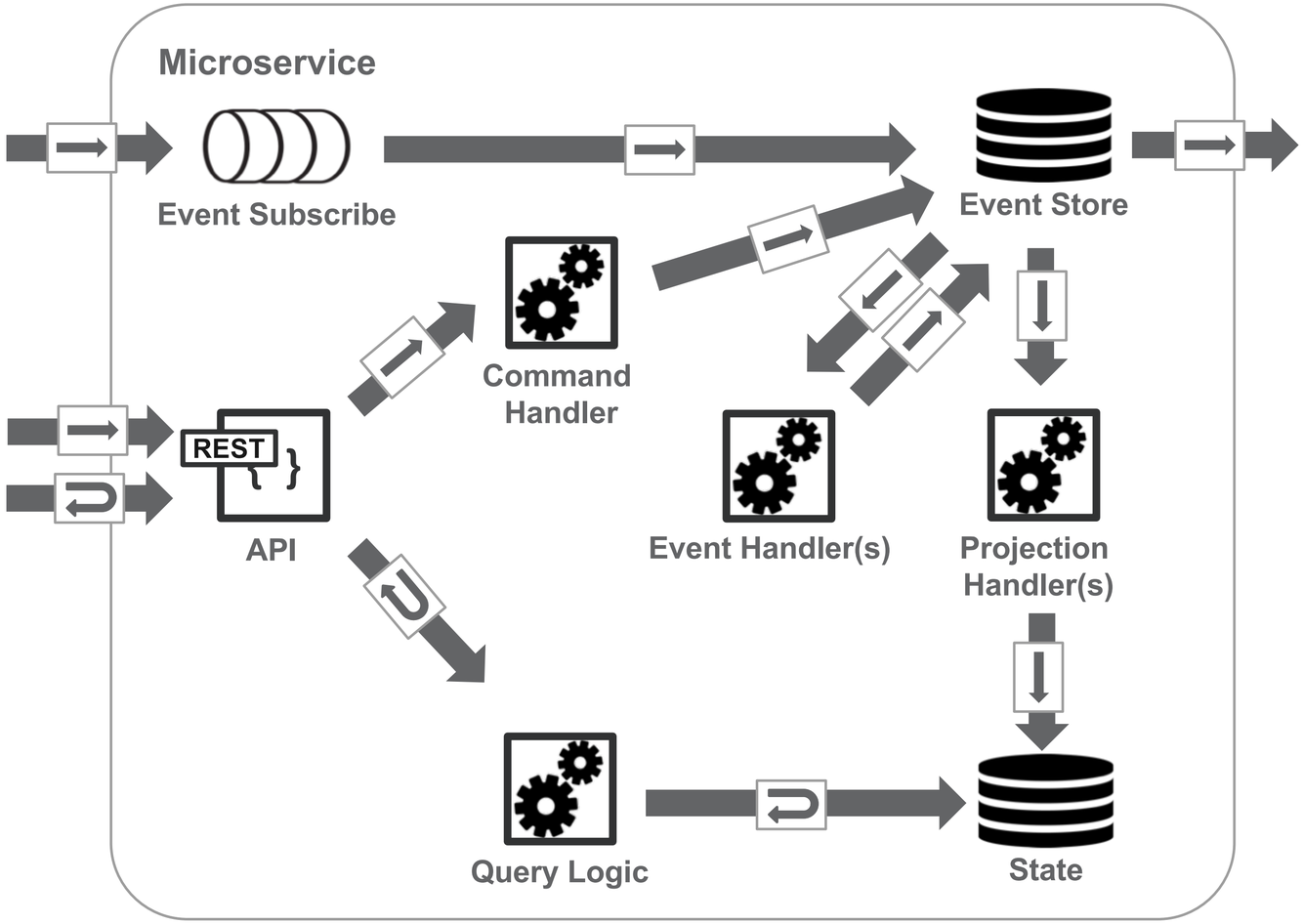
- “eat your own dog food”



Using Event Sourcing with Microservices

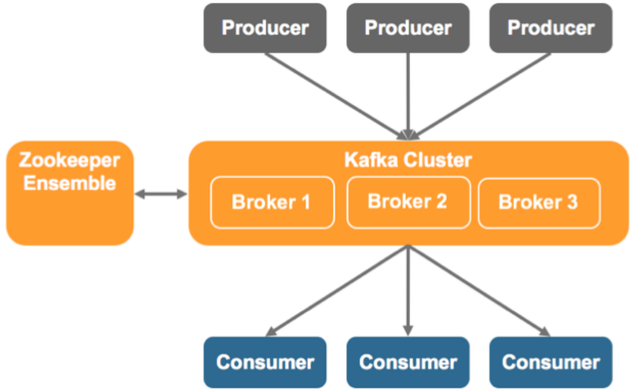
"Event sourcing enables building a forward-compatible application architecture—the ability to add more applications in the future that need to process the same event but create a different materialized view."

Neha Narkhede, Confluent Blog

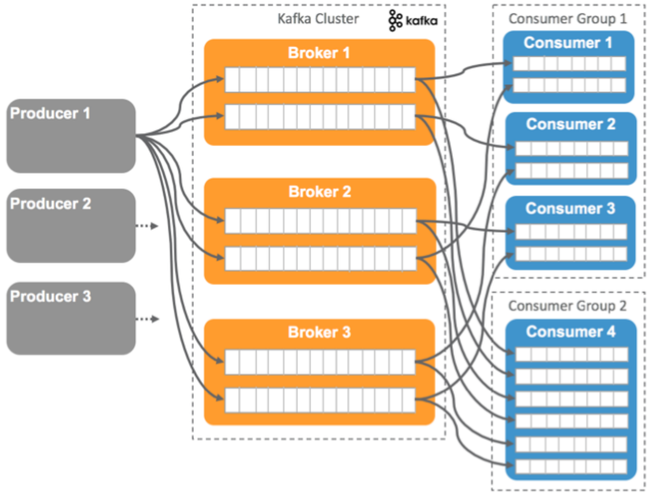


Apache Kafka – A Streaming Platform

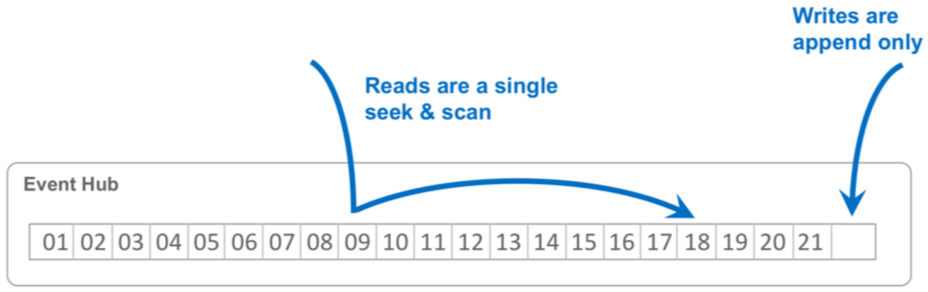
High-Level Architecture



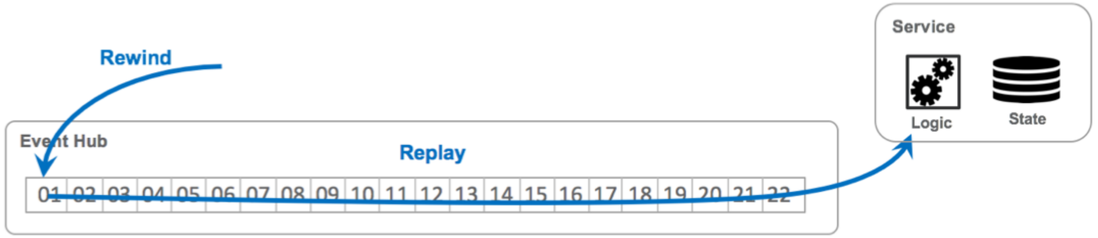
Scale-Out Architecture



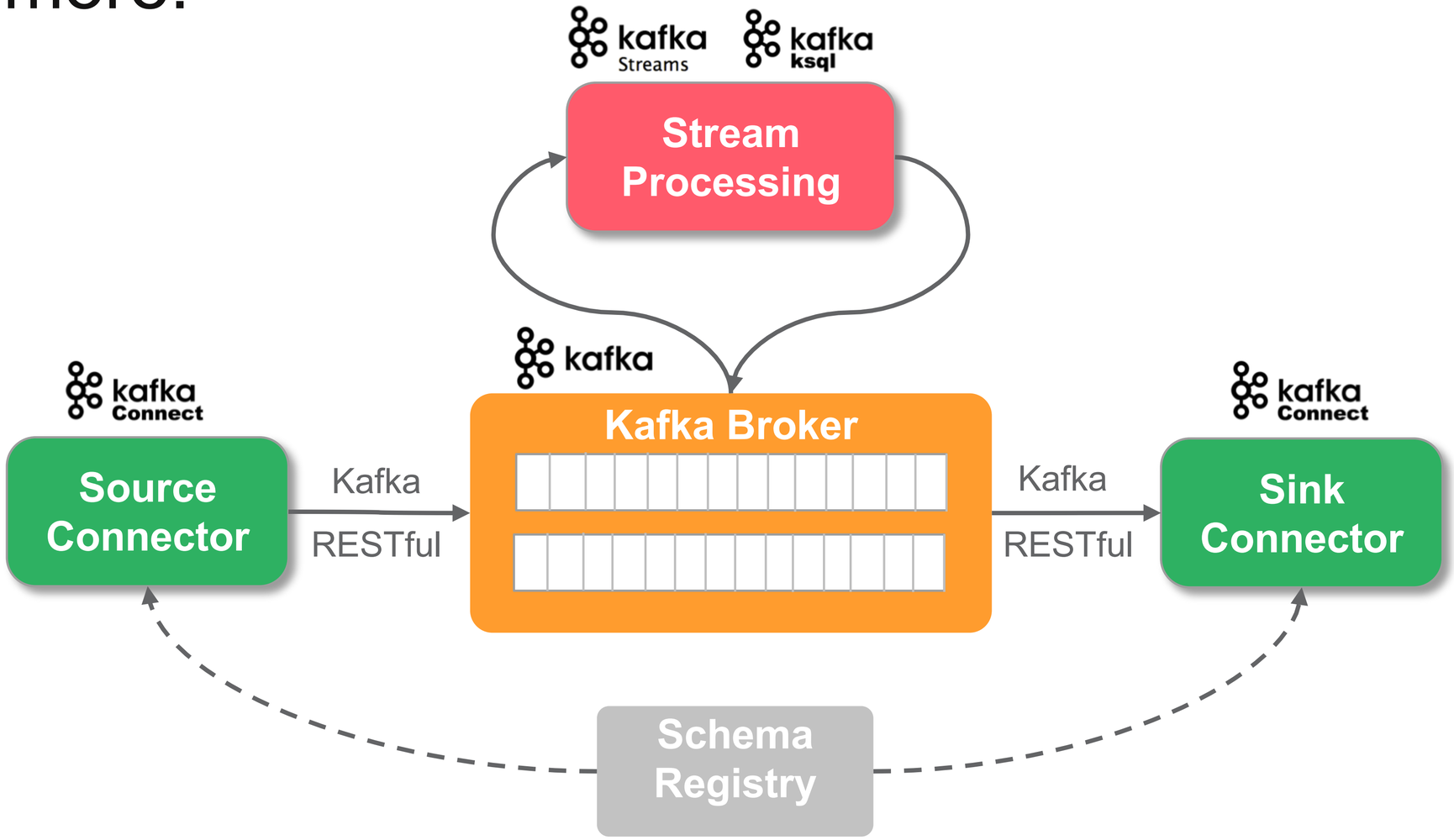
Distributed Log at the Core



Logs do not (necessarily) forget



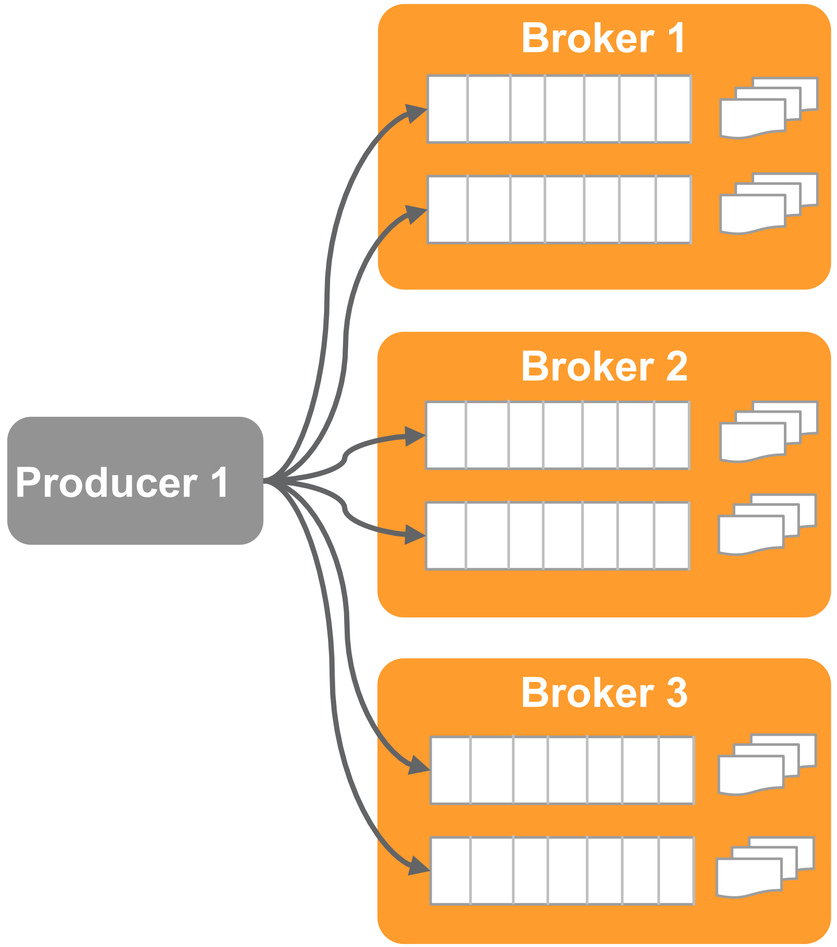
Apache Kafka – scalable message processing and more!



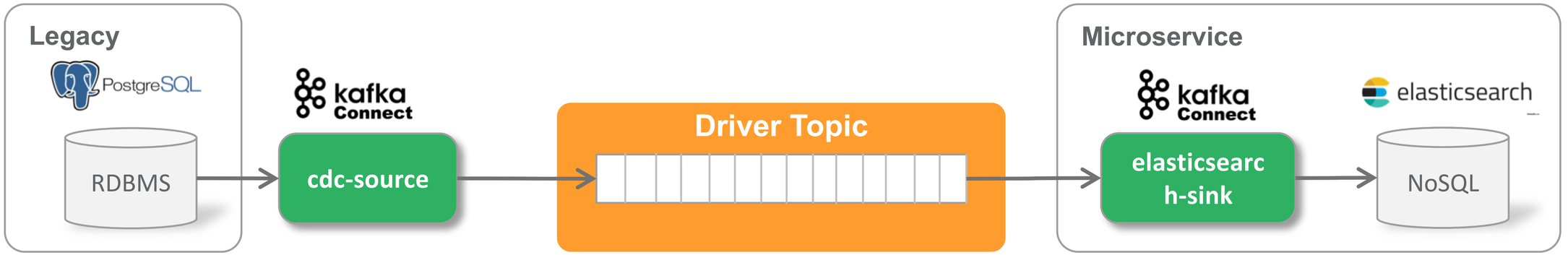
Hold Data for Long-Term – Data Retention

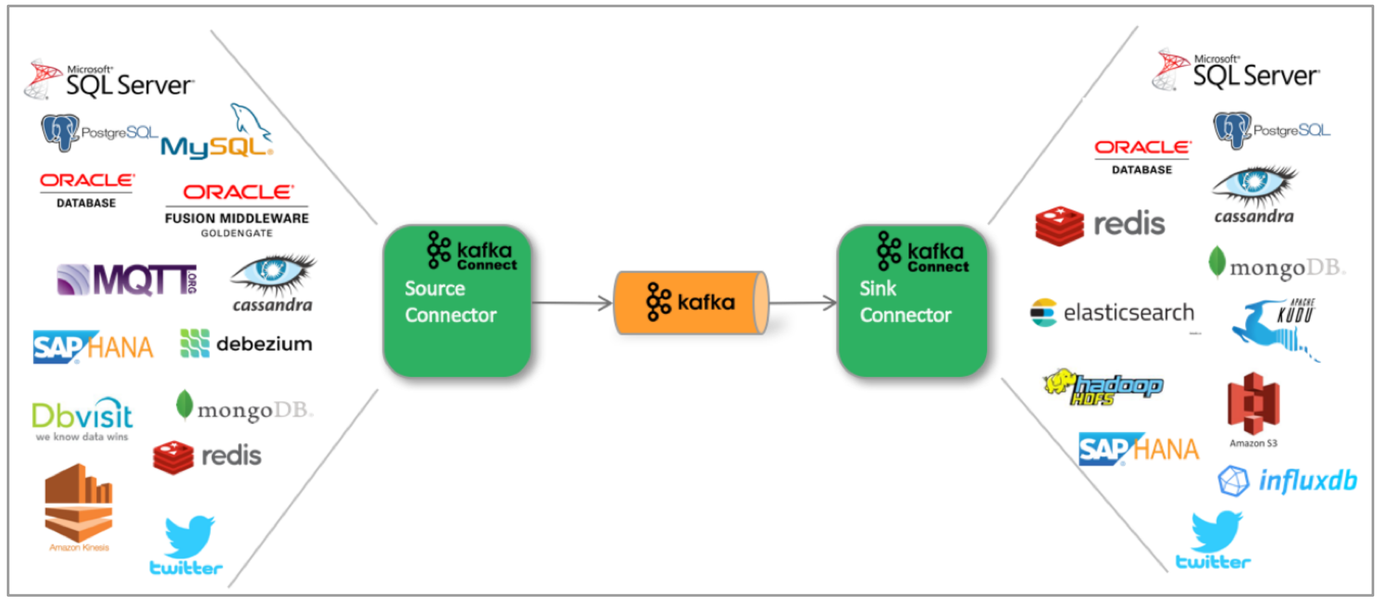
1. Never
2. Time based (TTL)
   `log.retention.{ms | minutes | hours}`
3. Size based
   `log.retention.bytes`
4. Log compaction based
   (entries with same key are removed):

```
kafka-topics.sh --zookeeper zk:2181 \
                --create --topic customers \
                --replication-factor 1 \
                --partitions 1 \
                --config cleanup.policy=compact
```



Change Data Capture (CDC)





trivadis makes IT easier.