Building event-driven (Micro)Services with Apache Kafka

Guido Schmutz
Voxxed Days Banff – 27.10.2018

@gschmutz

guidoschmutz.wordpress.com

Guido Schmutz

Working at Trivadis for more than 21 years

Oracle ACE Director for Fusion Middleware and SOA

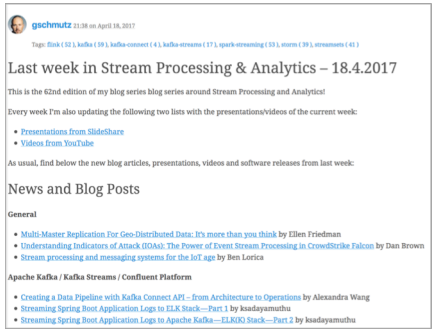
Oracle Groundbreaker Ambassador

Consultant, Trainer Software Architect for Java, Oracle, SOA and Big Data / Fast Data

Head of Trivadis Architecture Board

Technology Manager @ Trivadis

More than 30 years of software development experience



Contact: guido.schmutz@trivadis.com

Blog: http://guidoschmutz.wordpress.com

Slideshare: http://www.slideshare.net/gschmutz

Twitter: gschmutz

trivadis makes IT easier.

With over 600 specialists and IT experts in your region.

- 14 Trivadis branches and more than 600 employees
- 200 Service Level Agreements
- Over 4,000 training participants
- Research and development budget: CHF 5.0 million
- Financially self-supporting and sustainably profitable
- Experience from more than 1,900 projects per year at over 800 customers

trivadis
makes IT easier.

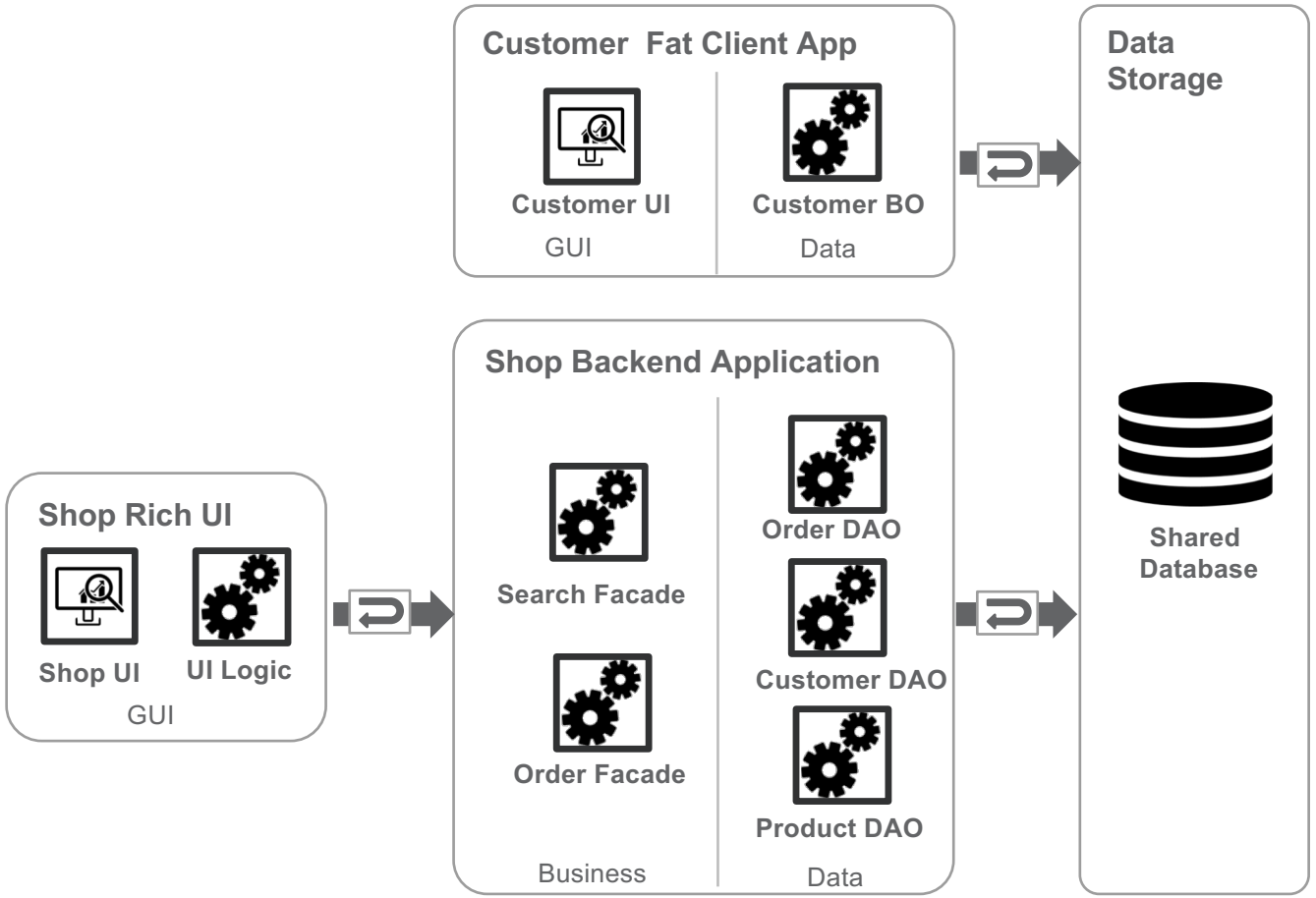
Agenda

1. Where do we come from?
2. Using Microservices
3. Can we do (even) better?
4. Kafka for Event-Driven (Micro)Services
5. What about streaming sources?
6. What about integrating legacy applications?
7. What about (historical) data analytics?
8. Summary

Where do we come from?

trivadis
makes IT easier.

Traditional Approach



sync request/response

trivadis
makes IT easier.

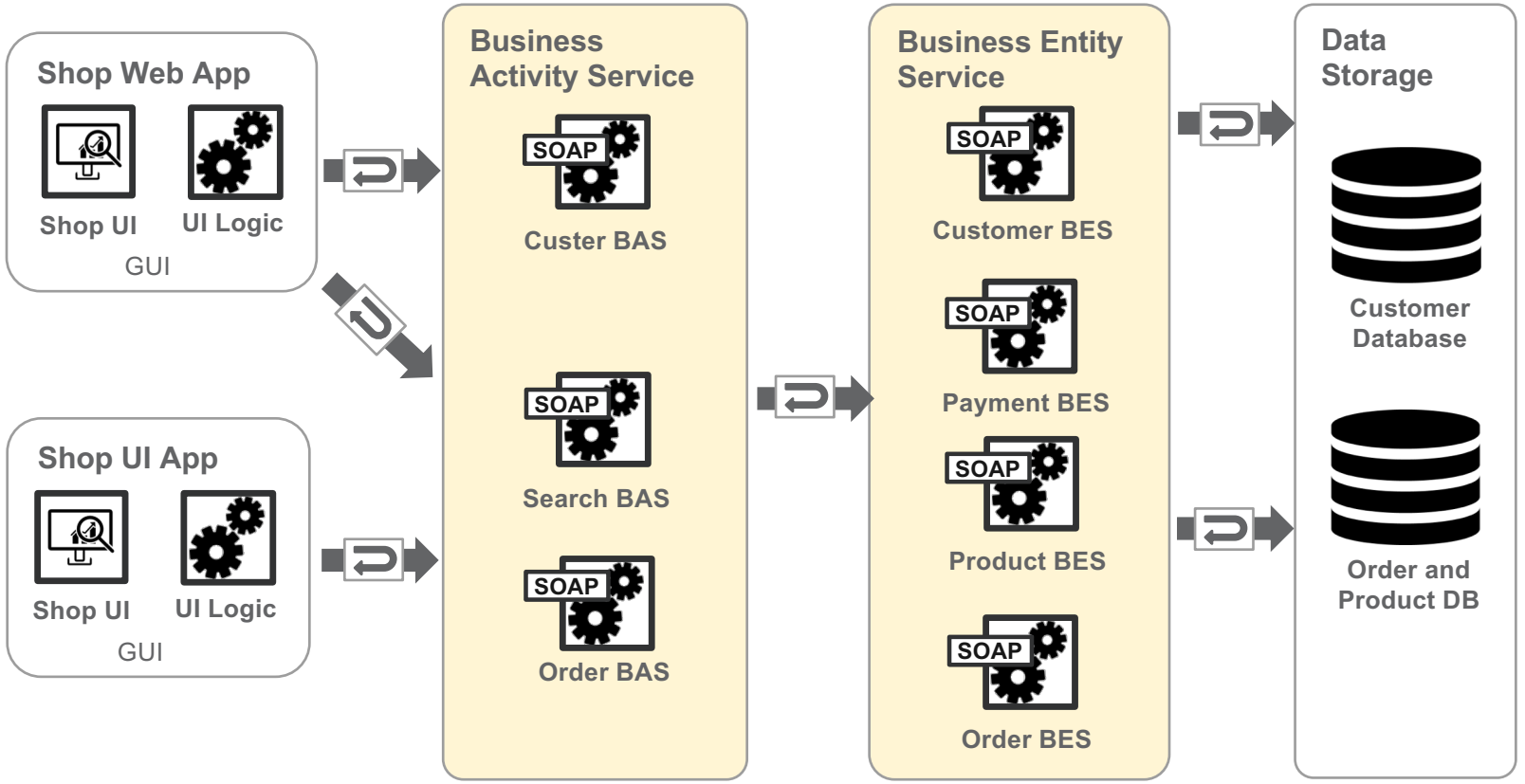
SOA Approach

Contract-first Web Services

Technical layers offer their own interfaces

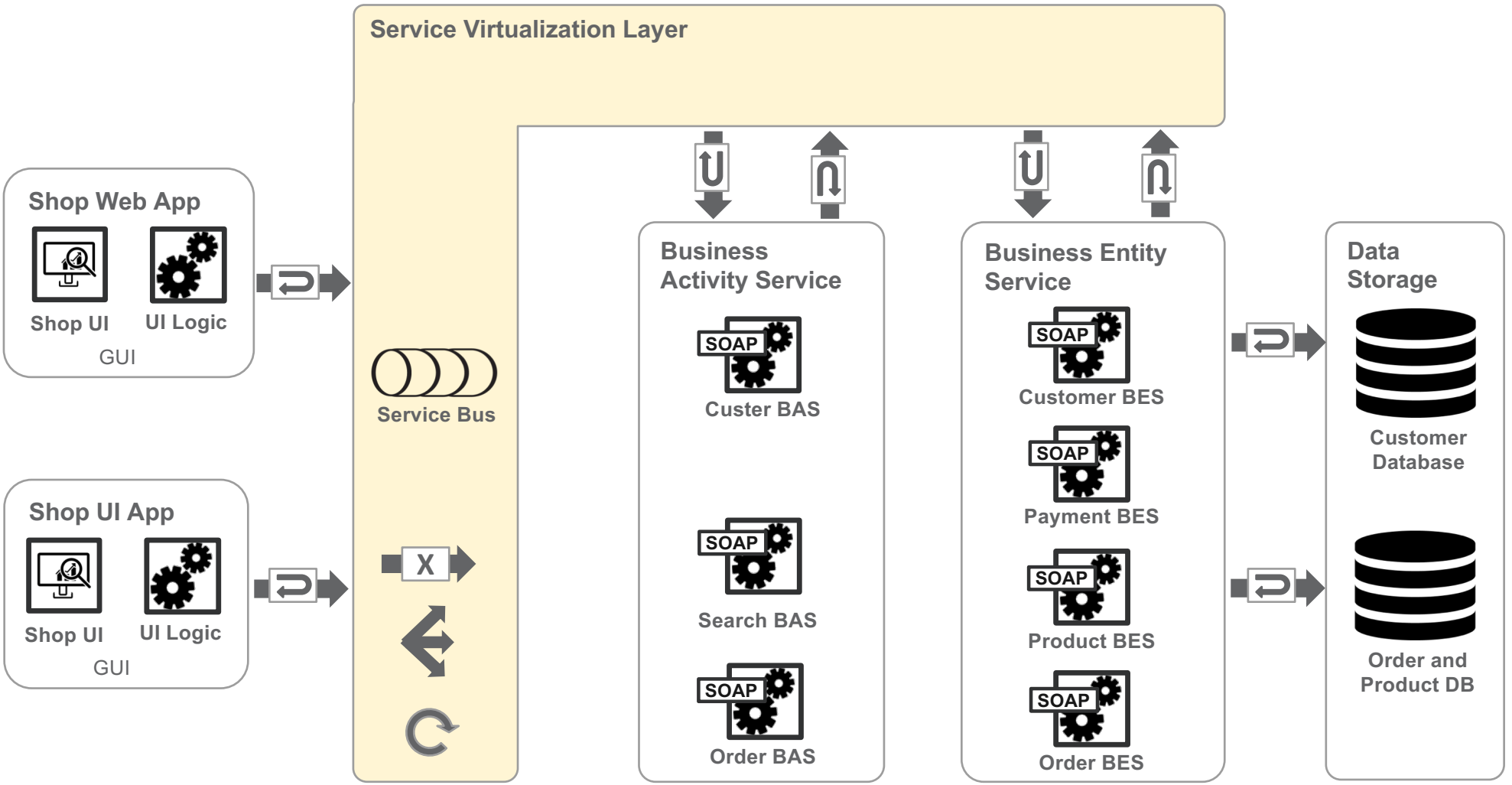
Reuse on each level

Lower layer often wraps legacy code



trivadis
makes IT easier.

Virtualized SOA Approach



Using Microservices

trivadis
makes IT easier.

Microservice Approach

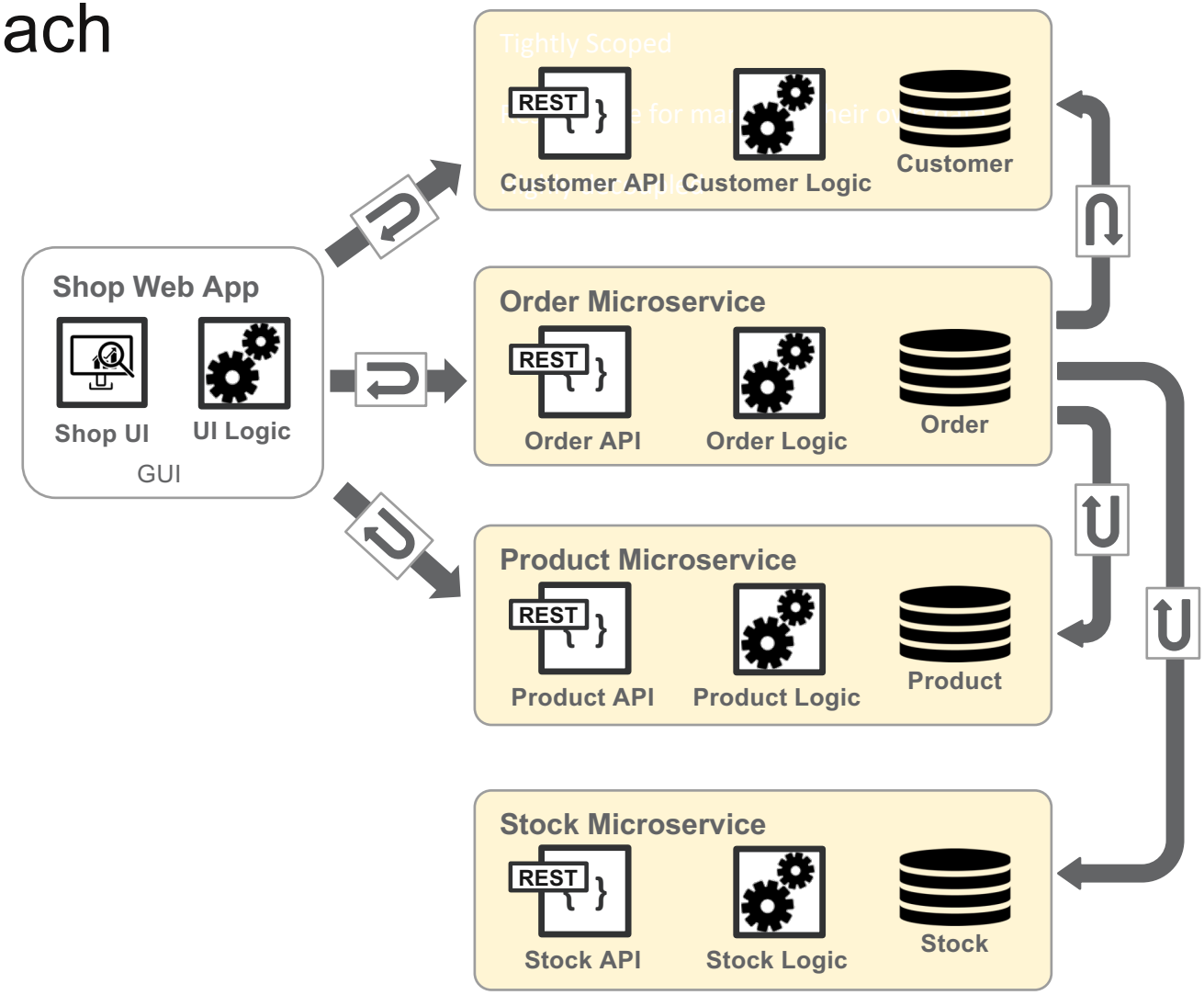
Tightly Scoped behind clear interfaces

Responsible for managing their own data (not necessarily the infrastructure)

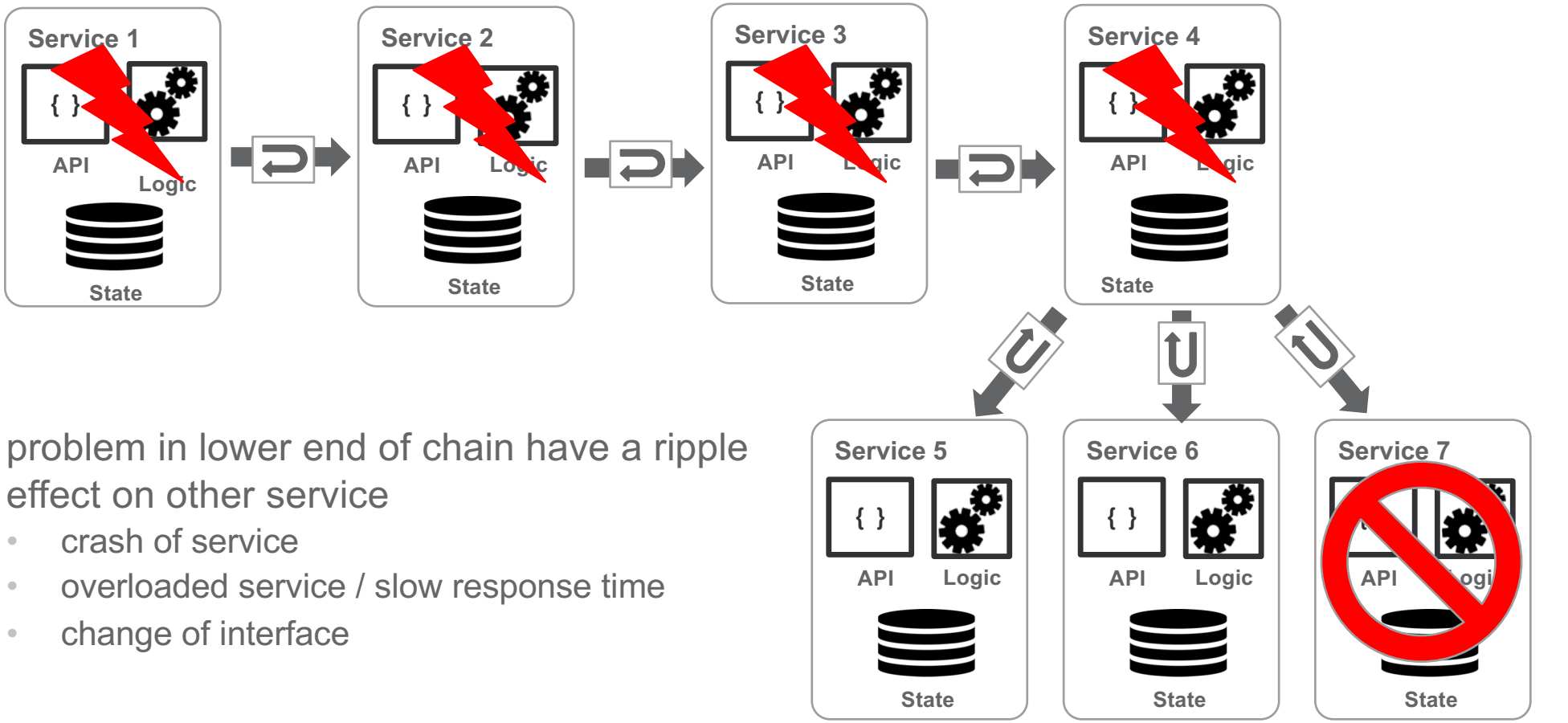
Should be highly decoupled

Independently deployable, self-contained and autonomous

SOA done right ?!



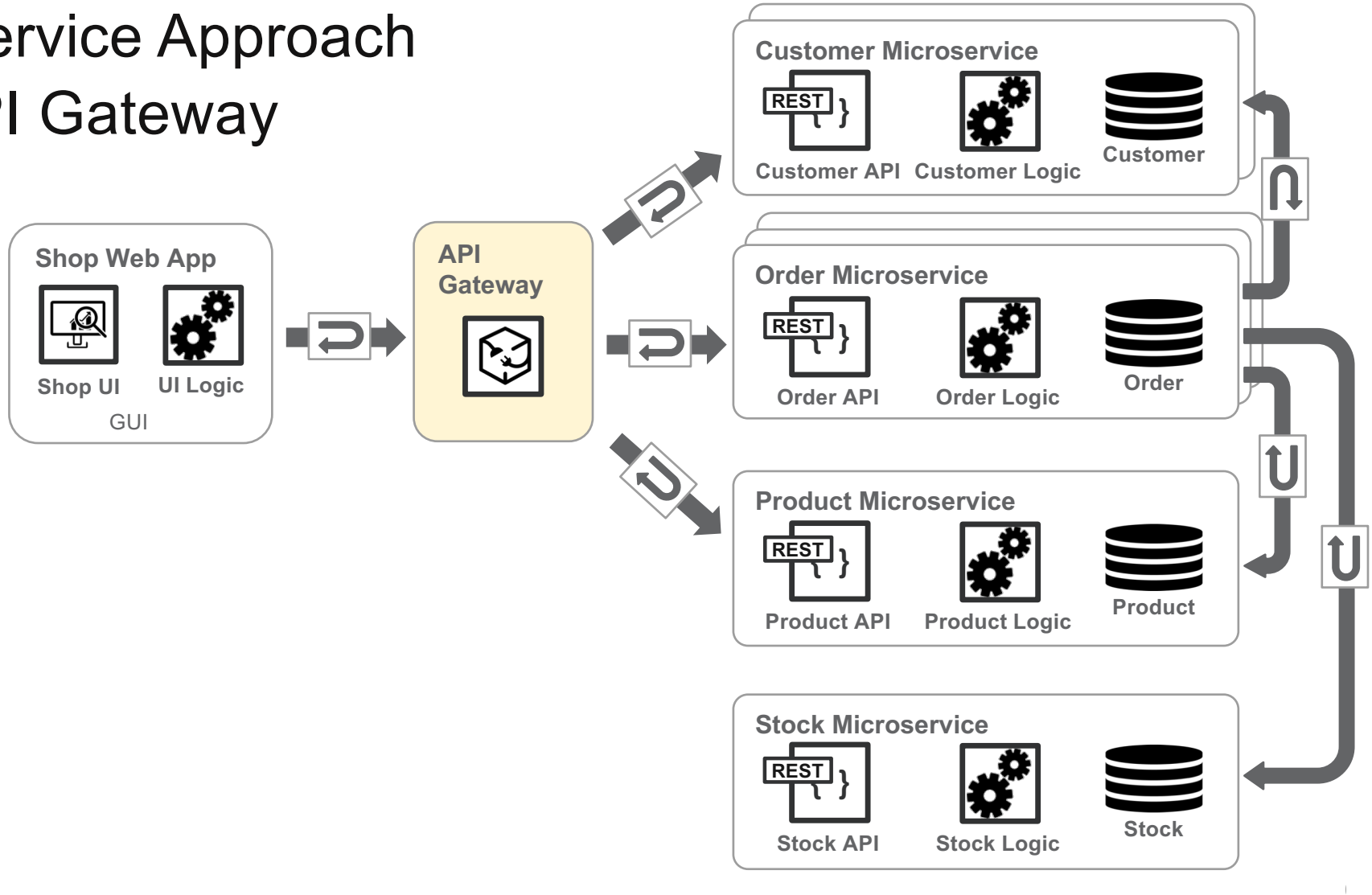
Synchronous World of Request-Response leads to tight, point-to-point couplings



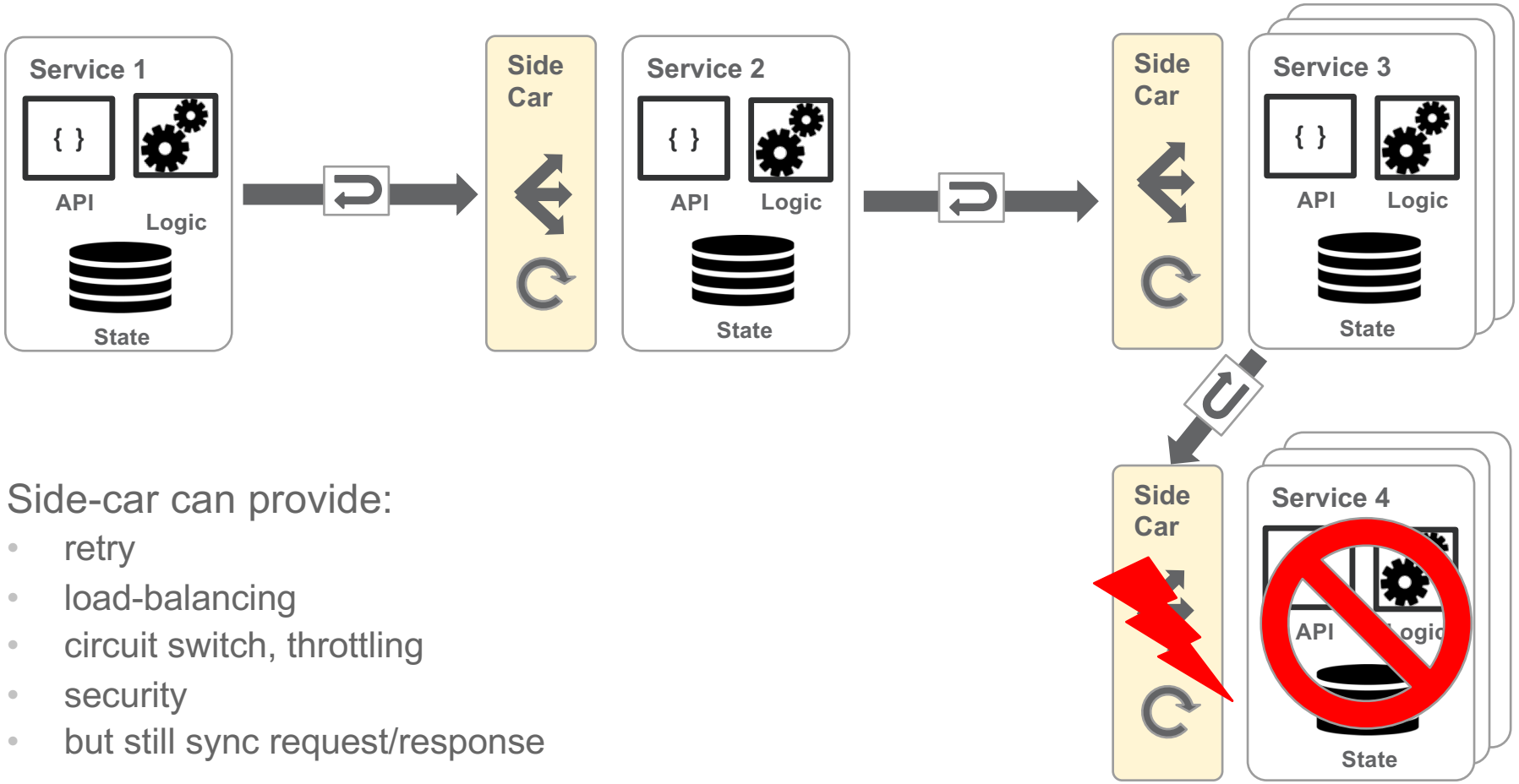
problem in lower end of chain have a ripple effect on other service

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

Microservice Approach with API Gateway



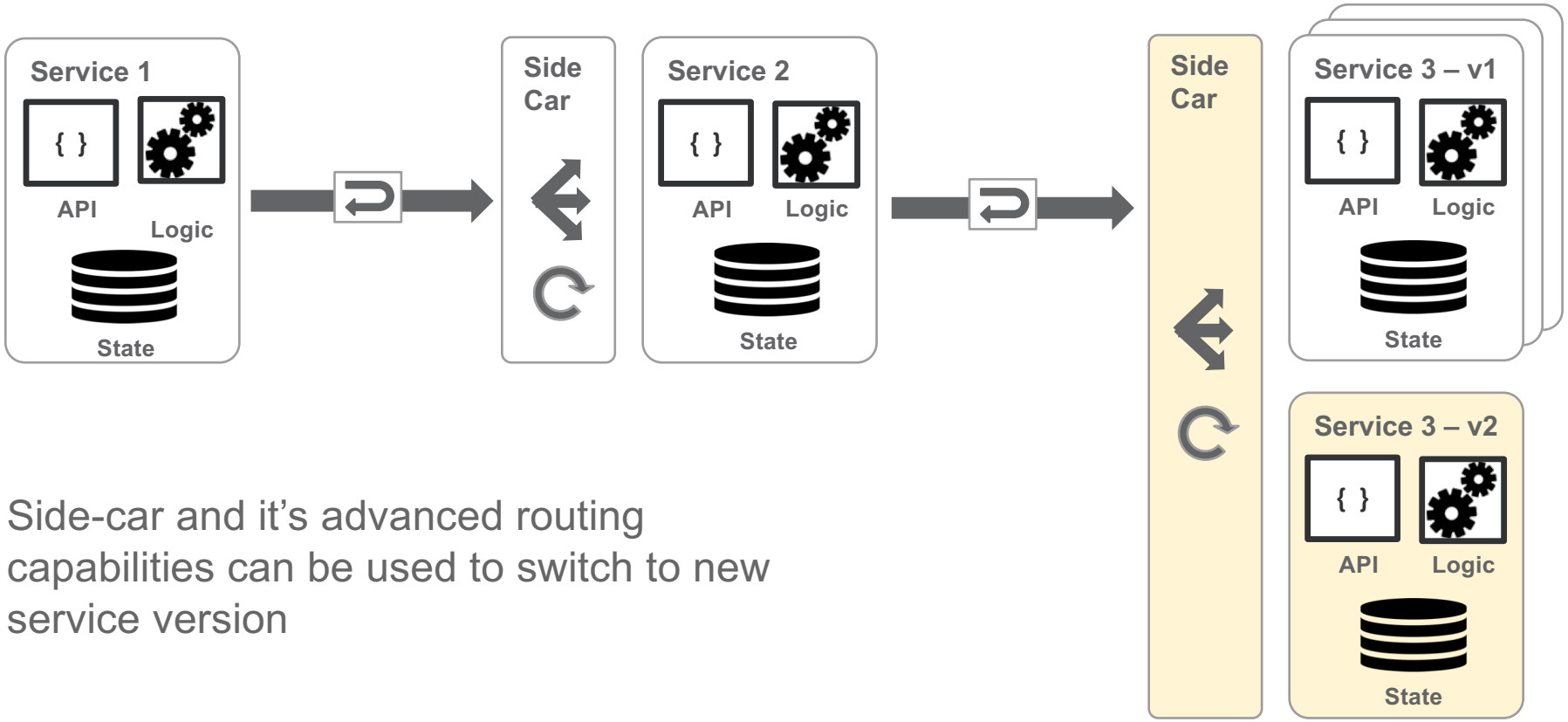
Microservice Approach with Side Car (i.e. K8s & Istio)



Side-car can provide:

- retry
- load-balancing
- circuit switch, throttling
- security
- but still sync request/response

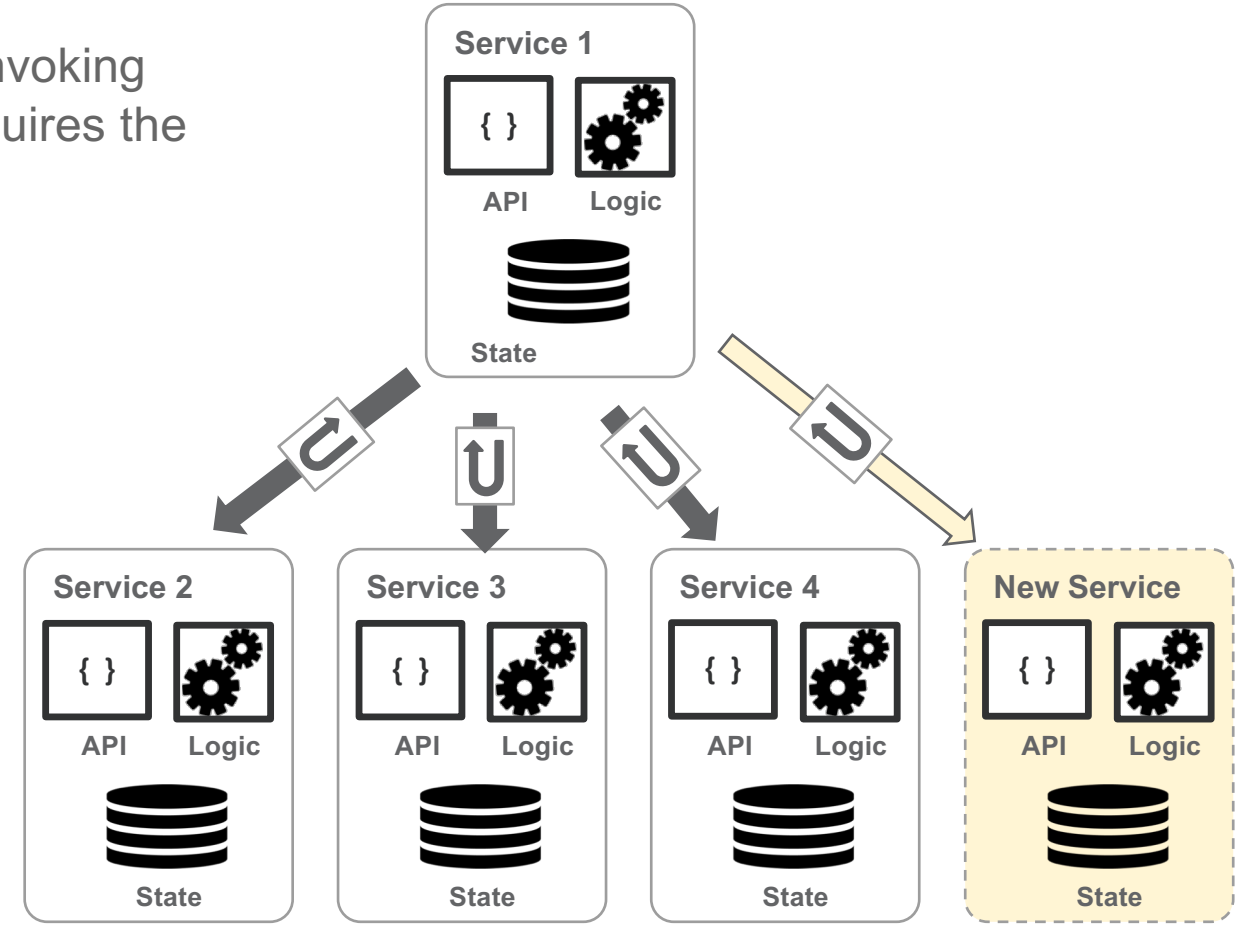
Microservice Approach with Side Car (i.e. K8s & Istio)



Side-car and it's advanced routing capabilities can be used to switch to new service version

Side Car provides abstraction ….

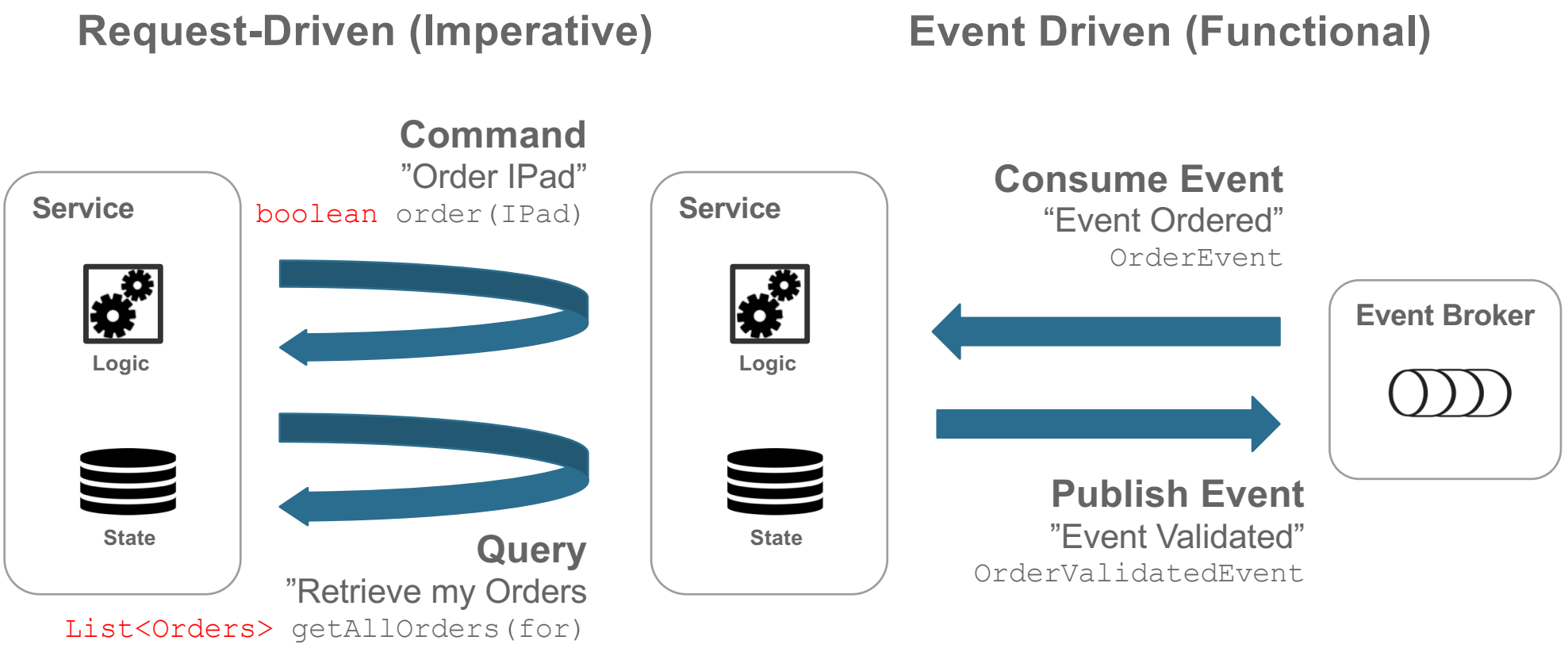
but we still have to change the invoking service …. If another service requires the same information



Can we do (even) better?

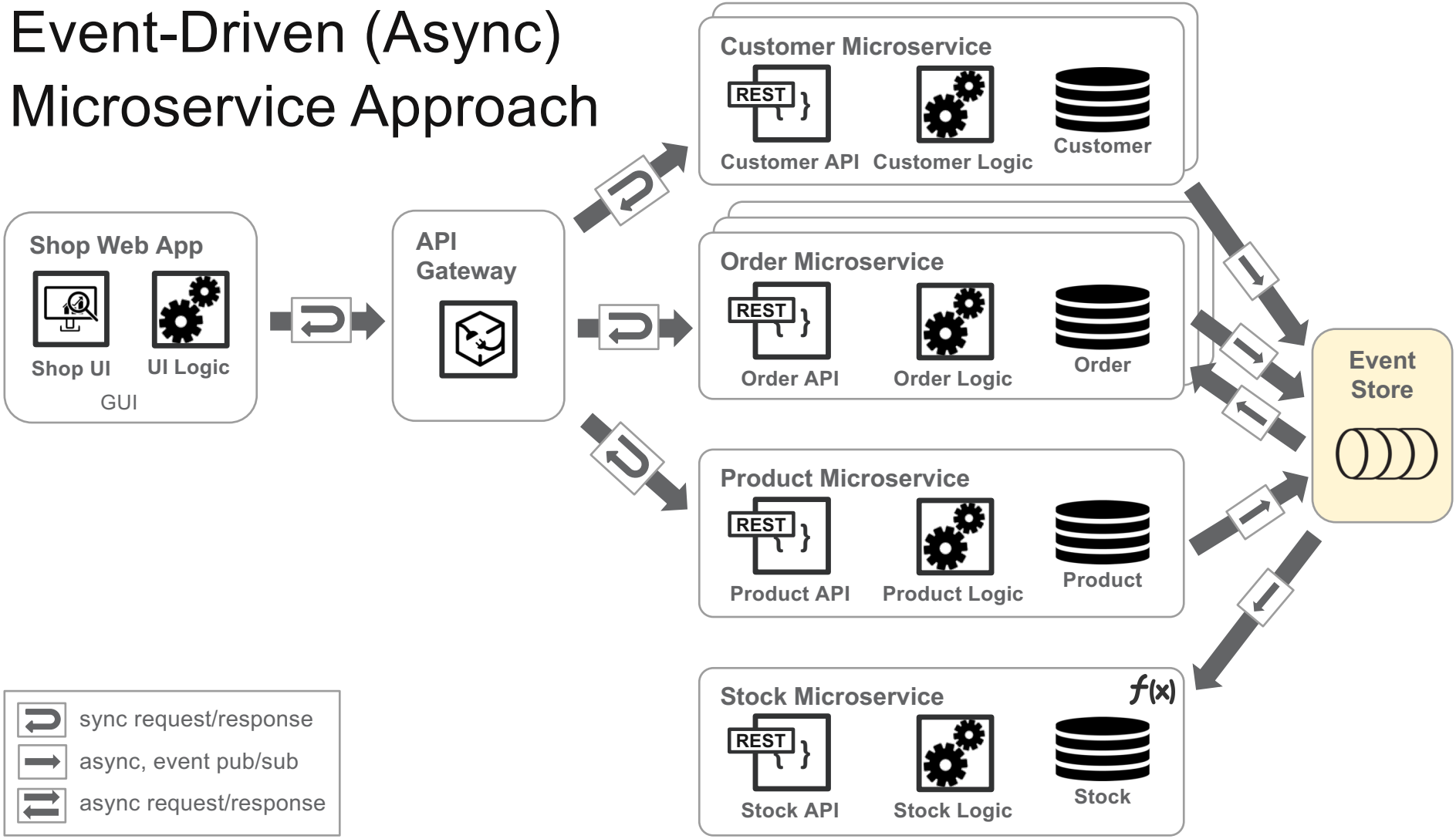
trivadis
makes IT easier.

Three mechanisms through which services can interact



trivadis
makes IT easier.

Event-Driven (Async) Microservice Approach

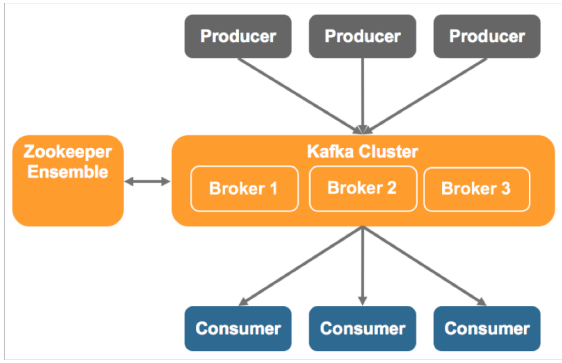


Kafka for Event-Driven (Micro)Services

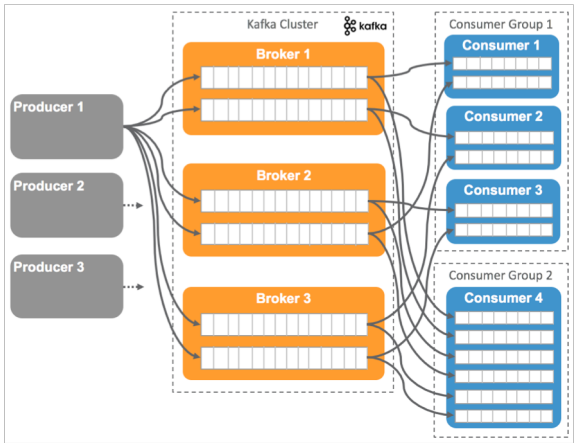
trivadis
makes IT easier.

Apache Kafka – A Streaming Platform

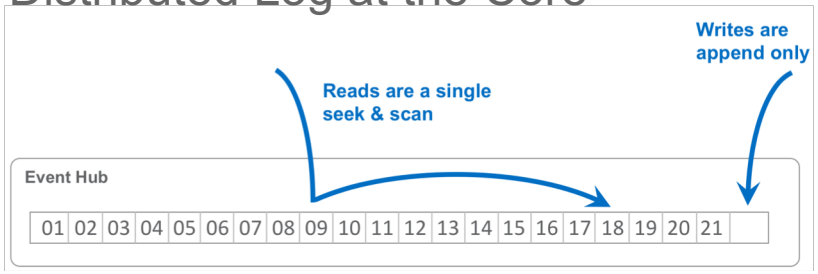
High-Level Architecture



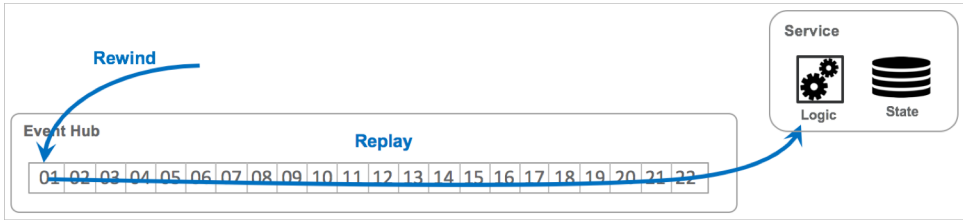
Scale-Out Architecture



Distributed Log at the Core



Logs do not (necessarily) forget

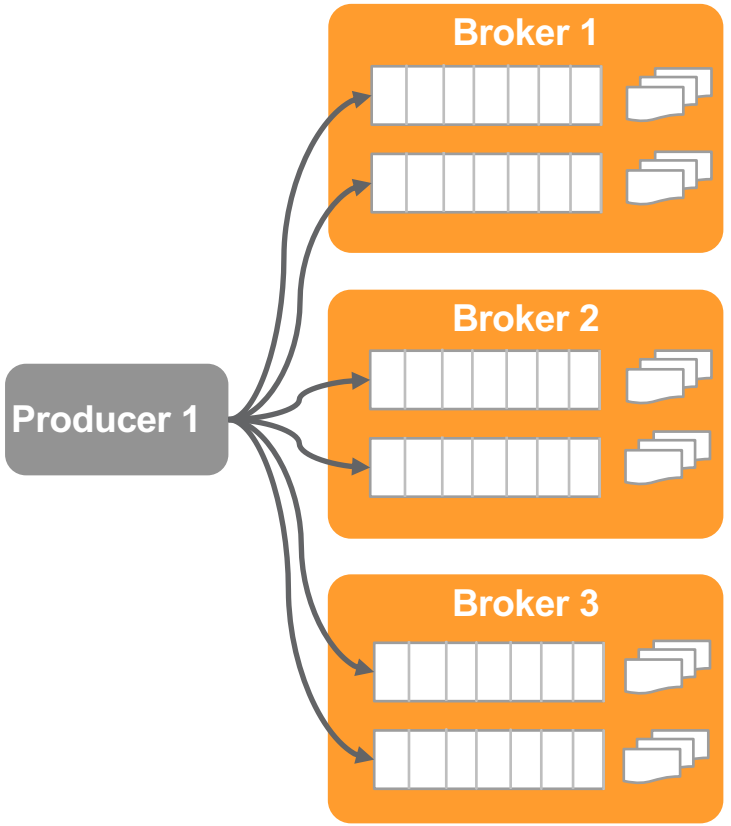


trivadis
makes IT easier.

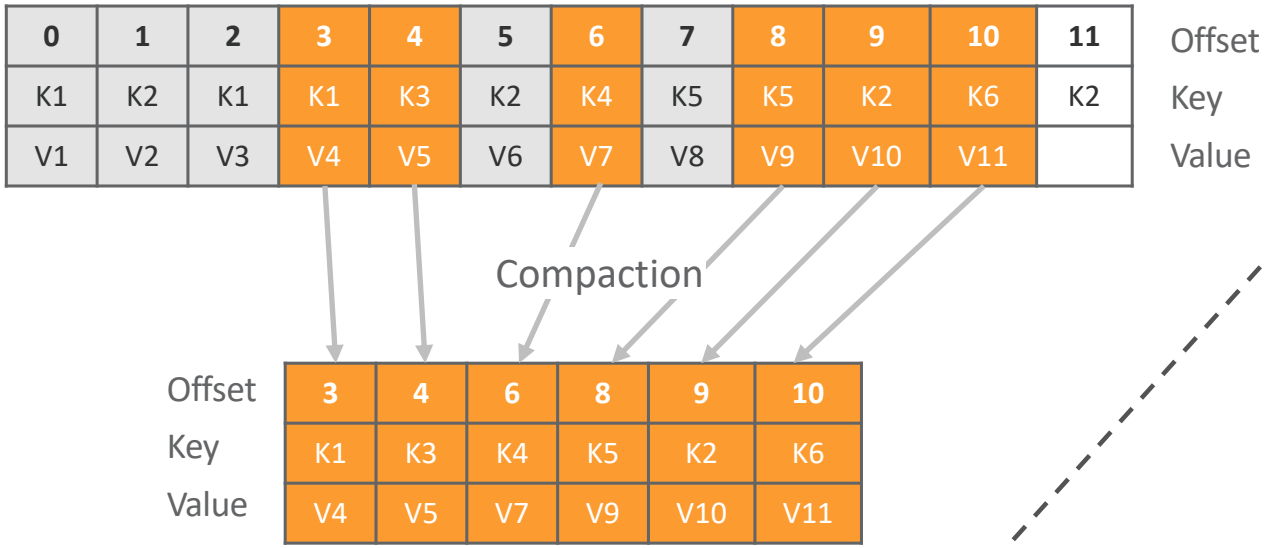
Hold Data for Long-Term – Data Retention

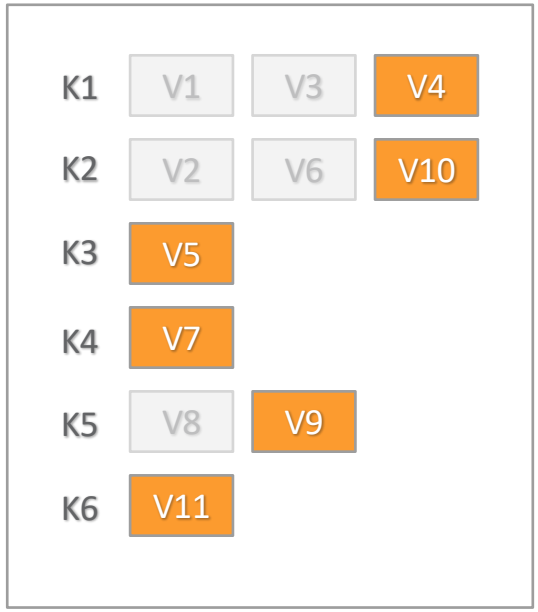
1. Never
2. Time based (TTL)
   `log.retention.{ms | minutes | hours}`
3. Size based
   `log.retention.bytes`
4. Log compaction based
   (entries with same key are removed):

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



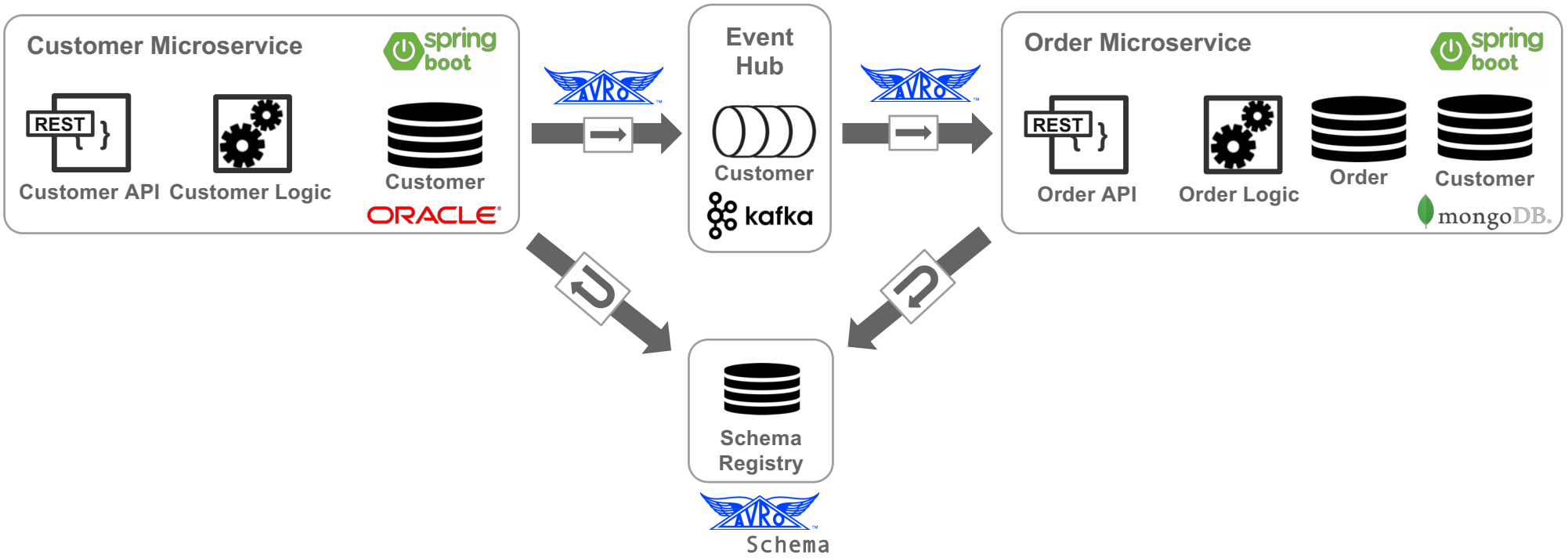
Keep Topics in Compacted Form





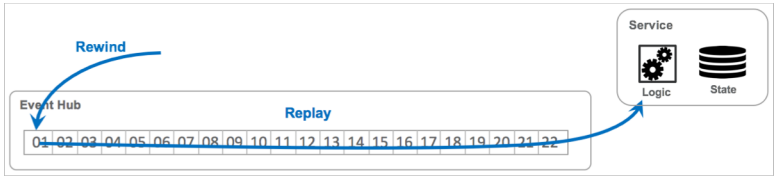
trivadis
makes IT easier.

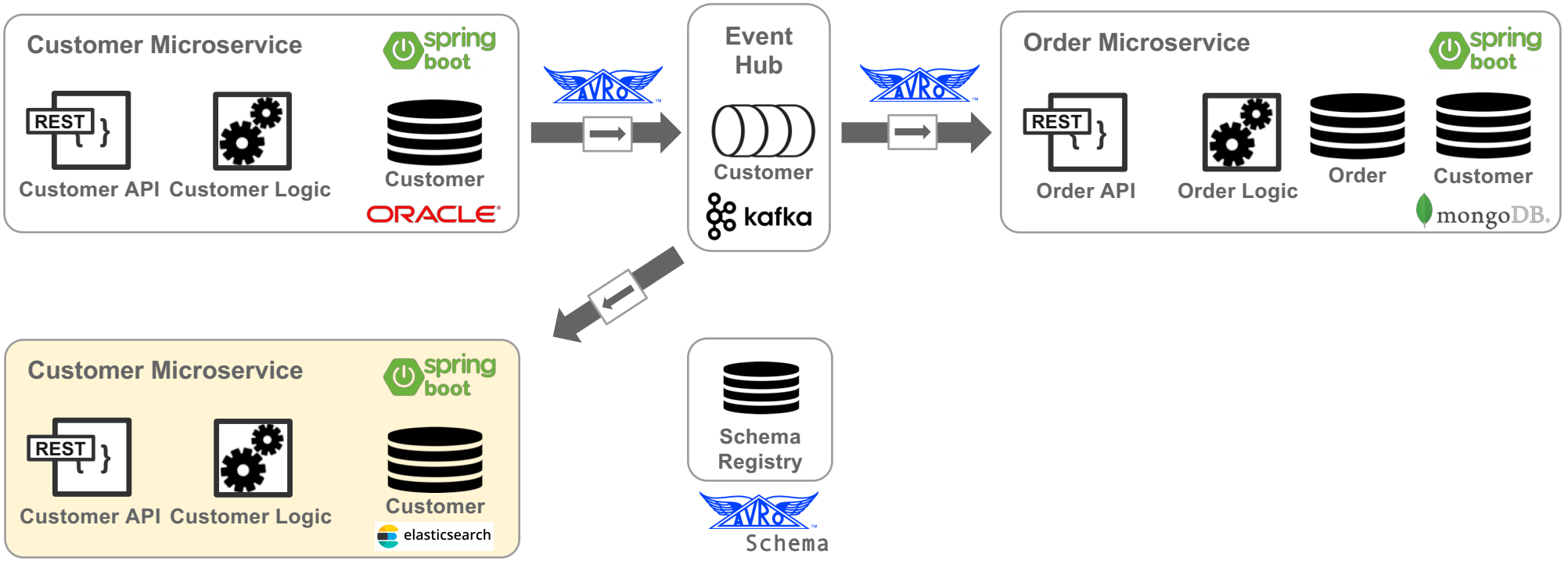
Demo



https://github.com/gschmutz/various-demos/tree/master/event-driven-microservices

Adding a new service by bootstrapping from Event Hub

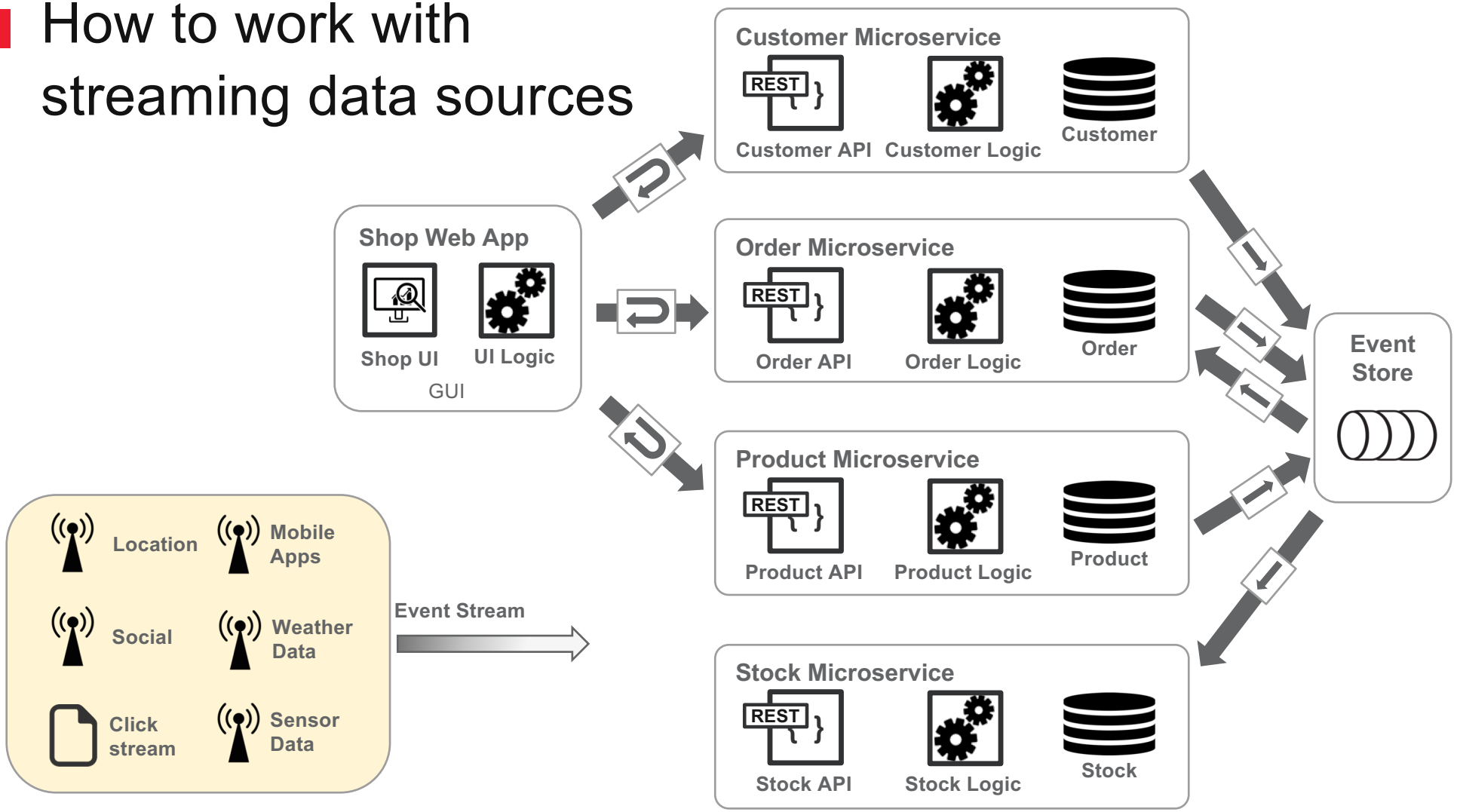




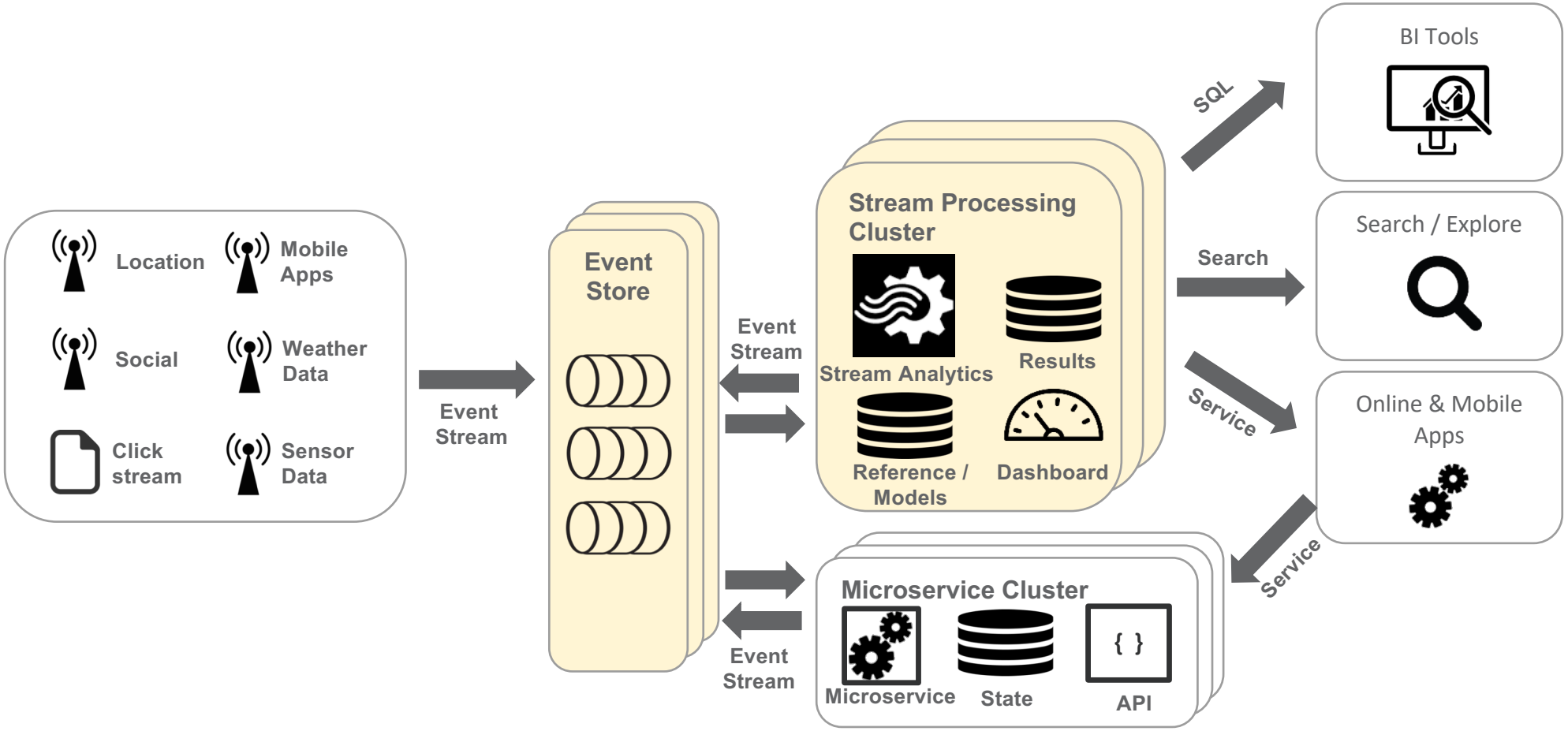
trivadis
makes IT easier.

What about streaming sources?

How to work with streaming data sources

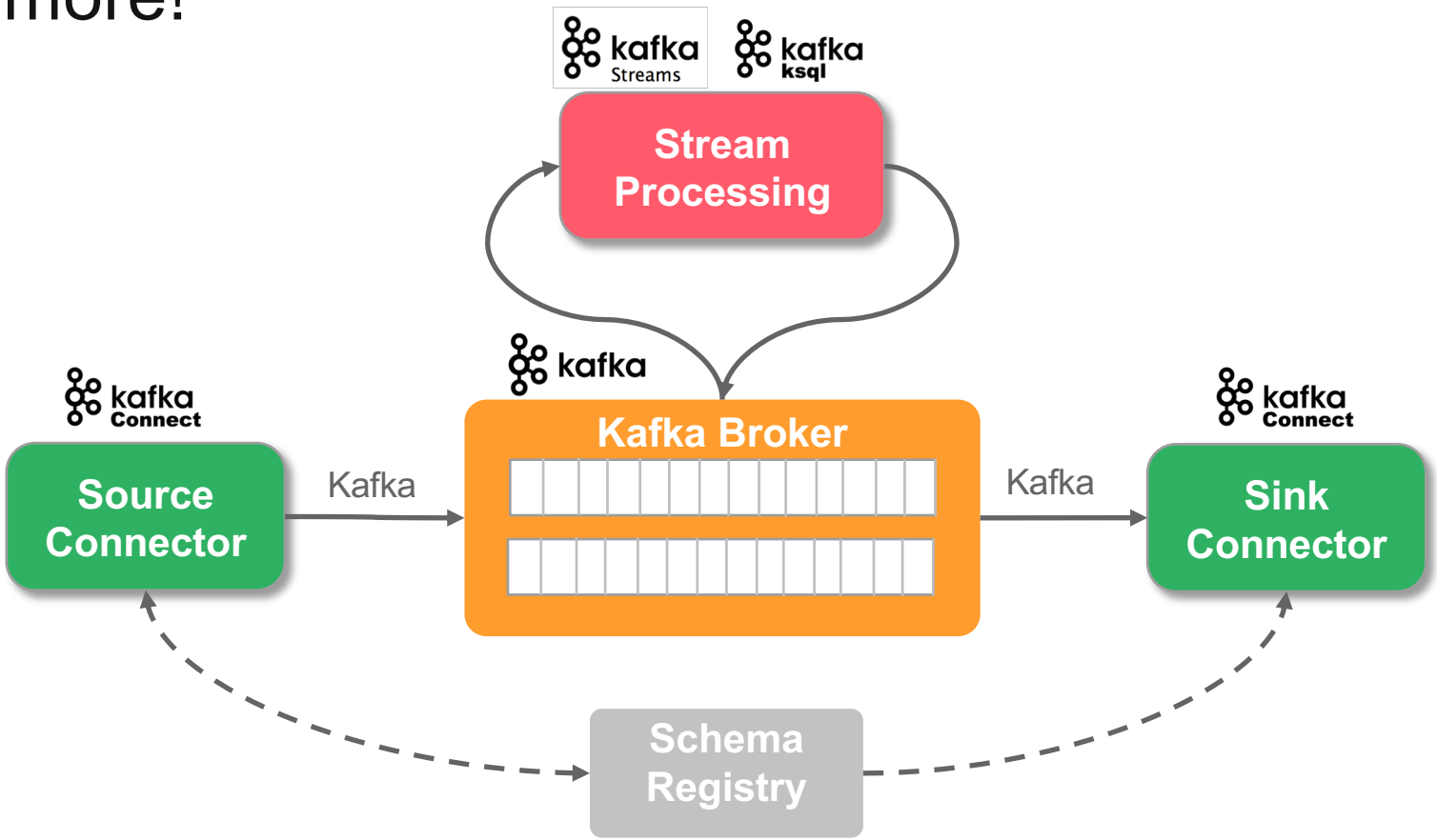


Streaming Processing & Microservices Architecture

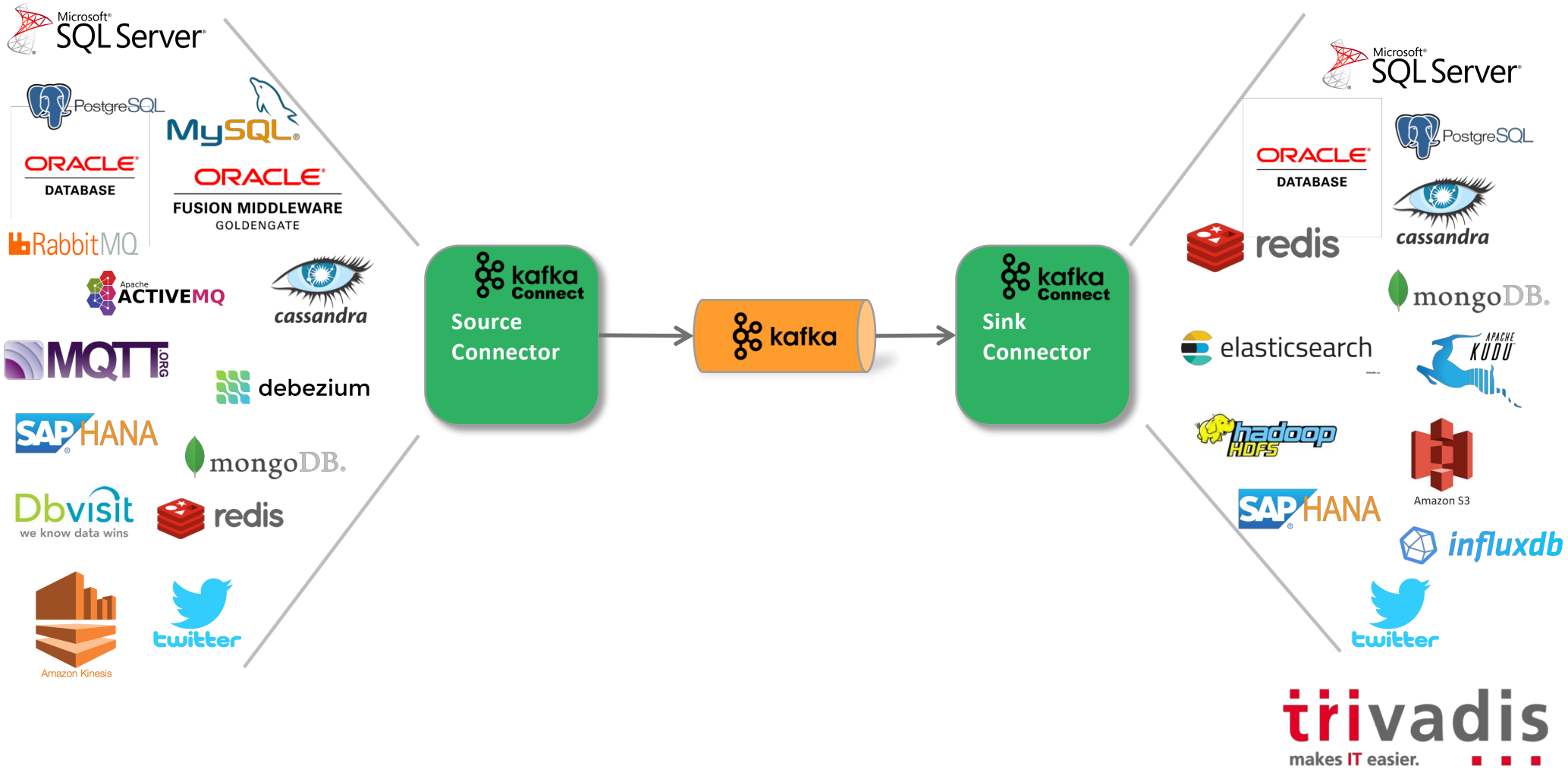


trivadis
makes IT easier.

- Apache Kafka – scalable message processing and more!



Kafka Connect - Overview



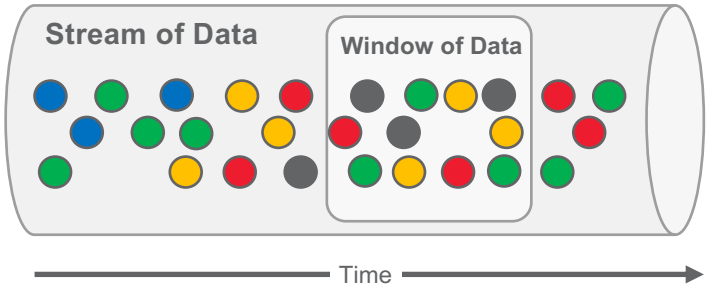
Stream Processing Capability - Windowing

Computations over events done using **windows of data**

Due to size and never-ending nature of it, it's not feasible to keep entire stream of data in memory

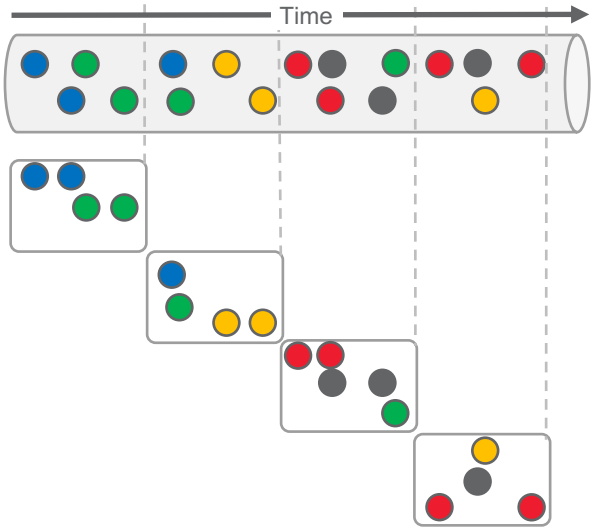
A window of data represents a certain amount of data where we can perform computations on

Windows give the power to keep a working memory and look back at recent data efficiently

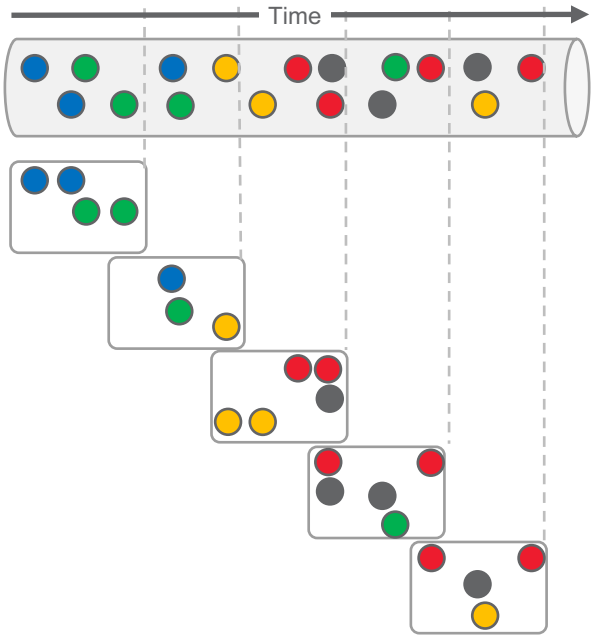


Stream Processing Capability - Windowing

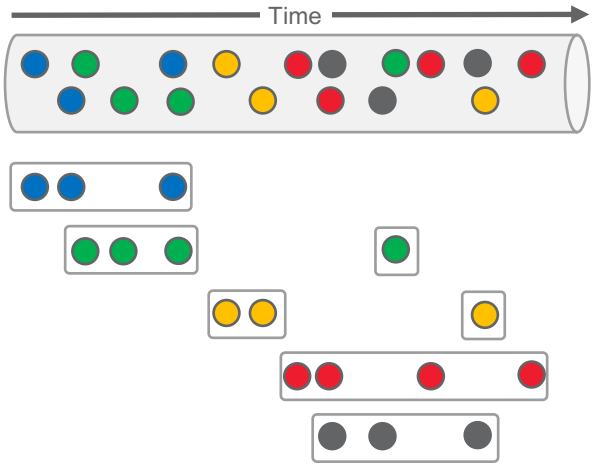
Fixed Window (aka Tumbling Window) - eviction policy always based on the window being full and trigger policy based on either the count of items in the window or time



Sliding Window (aka Hopping Window) - uses eviction and trigger policies that are based on time: *window length* and *sliding interval length*



Session Window – composed of sequences of temporarily related events terminated by a gap of inactivity greater than some timeout



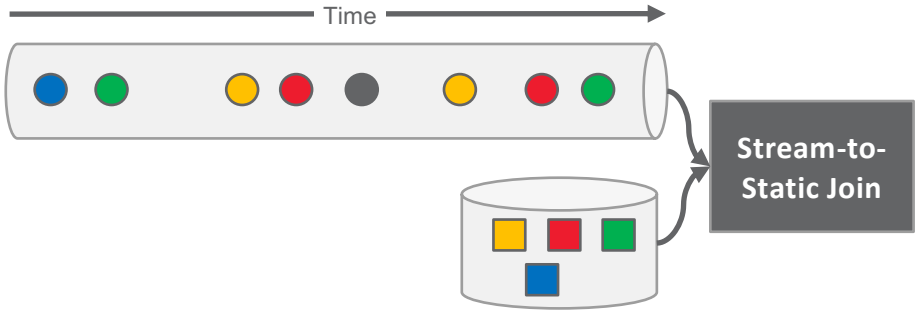
trivadis
makes IT easier.

Stream Processing Capability - Joining

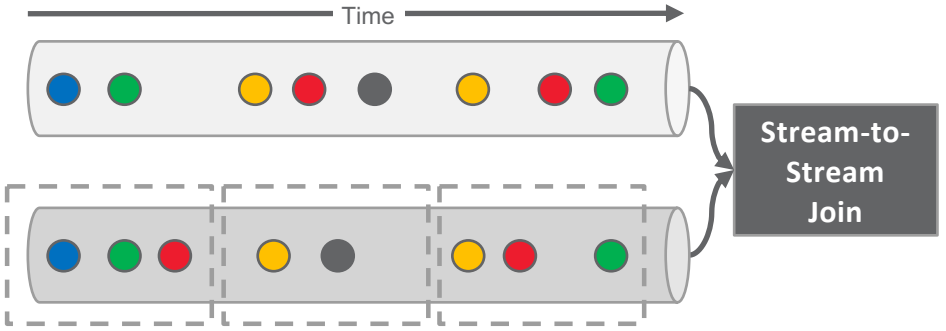
Challenges of joining streams

1. Data streams need to be aligned as they come because they have different timestamps
2. since streams are never-ending, the joins must be limited; otherwise join will never end
3. join needs to produce results continuously as there is no end to the data

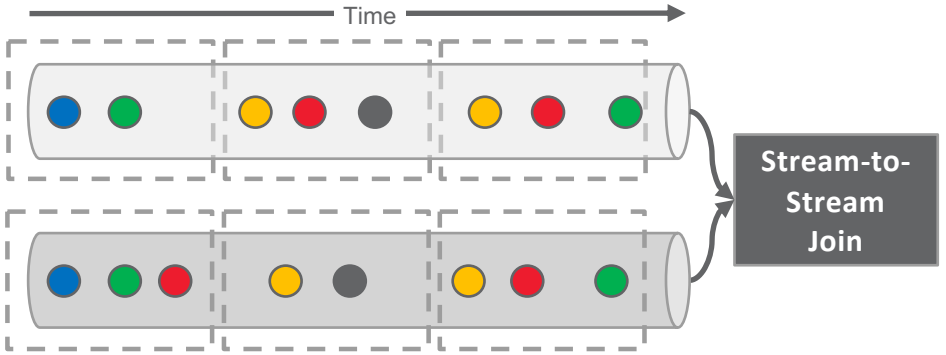
Stream to Static (Table) Join



Stream to Stream Join (one window join)



Stream to Stream Join (two window join)



trivadis
makes IT easier.

Stream Processing Capability - State Management

Necessary if stream processing use case is dependent on previously seen data or external data

Windowing, **Joining** and **Pattern Detection** use State Management behind the scenes

State Management services can be made available for custom state handling logic

State needs to be managed as close to the stream processor as possible

Options for State Management

How does it handle failures? If a machine crashes and the/some state is lost?

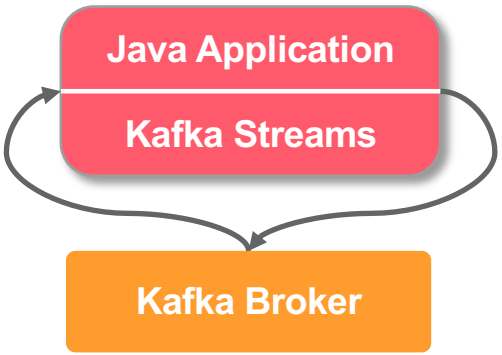
Kafka Streams - Overview

Designed as a **simple and lightweight library in Apache Kafka**

no other dependencies than Kafka

Supports **fault-tolerant local state**

Supports **Windowing** (Fixed, Sliding and Session) and **Stream-Stream / Stream-Table Joins**



Millisecond processing latency, no micro-batching

At-least-once **and exactly-once** processing guarantees

```
KTable<Integer, Customer> customers =
   builder.stream("customer");

KStream<Integer, Order> orders =
   builder.stream("order");

KStream<Integer, String> enriched =
   orders.leftJoin(customers, …);

joined.to("orderEnriched");
```

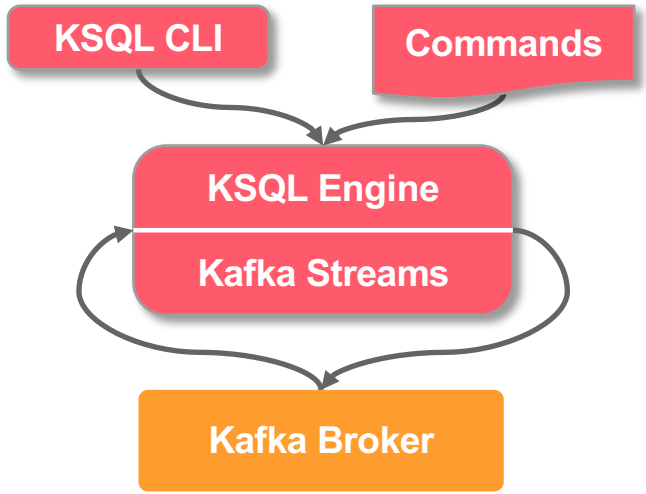
trivadis
makes IT easier.

KSQL - Overview

Stream Processing with zero coding using SQL-like language

Built on top of Kafka Streams

Interactive (CLI) and headless (command file)



STREAM and TABLE as first-class citizens

- STREAM = data in motion
- TABLE = collected state of a stream

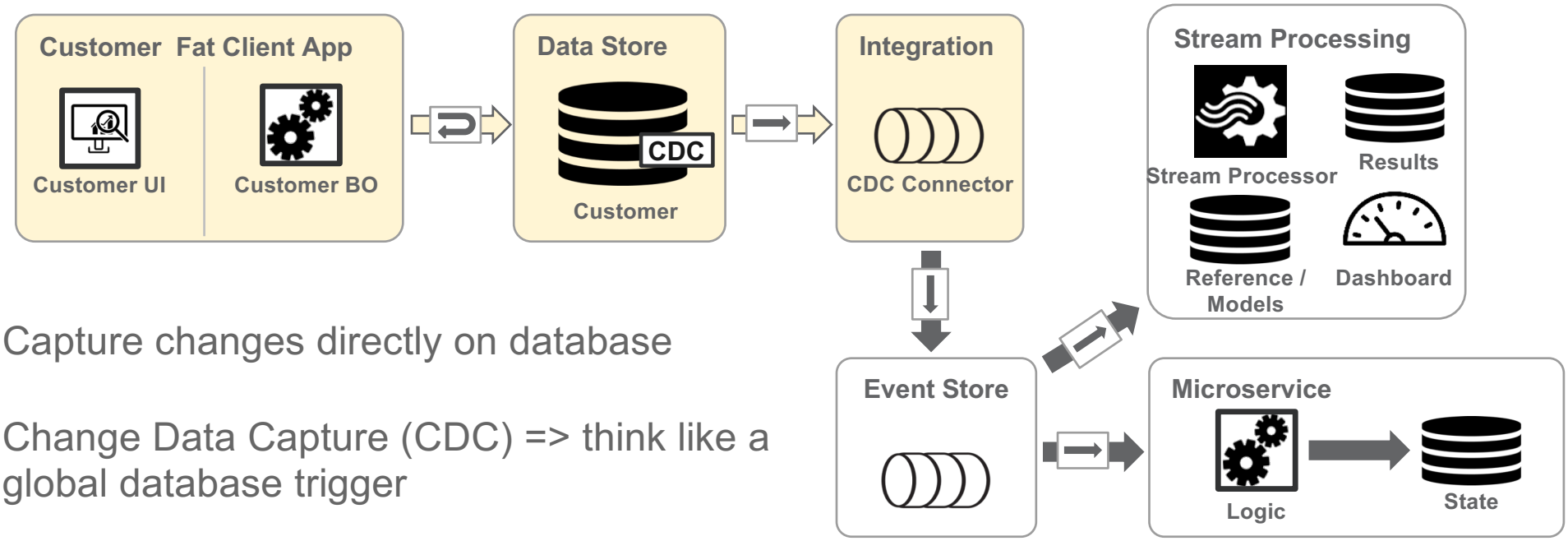
```
ksql> CREATE STREAM customer_s \
      WITH (kafka_topic='customer', \
        value_format='AVRO');

 Message
----------------
 Stream created

ksql> SELECT * FROM customer_s \
WHERE address->country = 'Switzerland';
...
```

trivadis
makes IT easier.

What about integrating legacy applications?

trivadis
makes IT easier.

Integrate existing systems through CDC



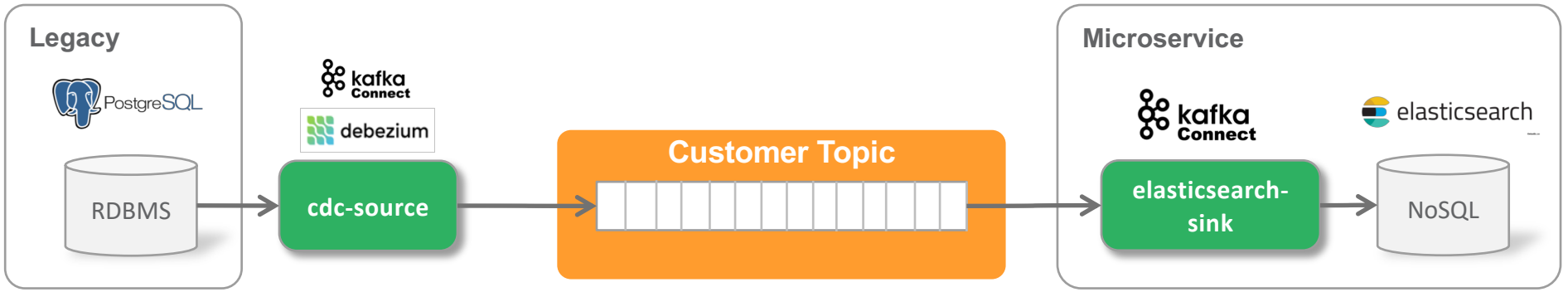
Capture changes directly on database

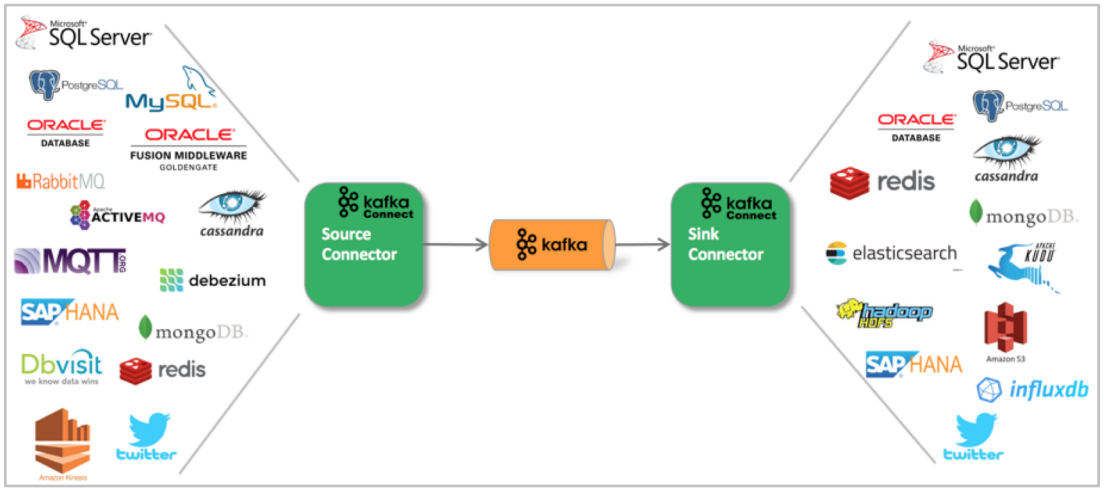
Change Data Capture (CDC) => think like a global database trigger

Transform existing systems to event producer

trivadis
makes IT easier.

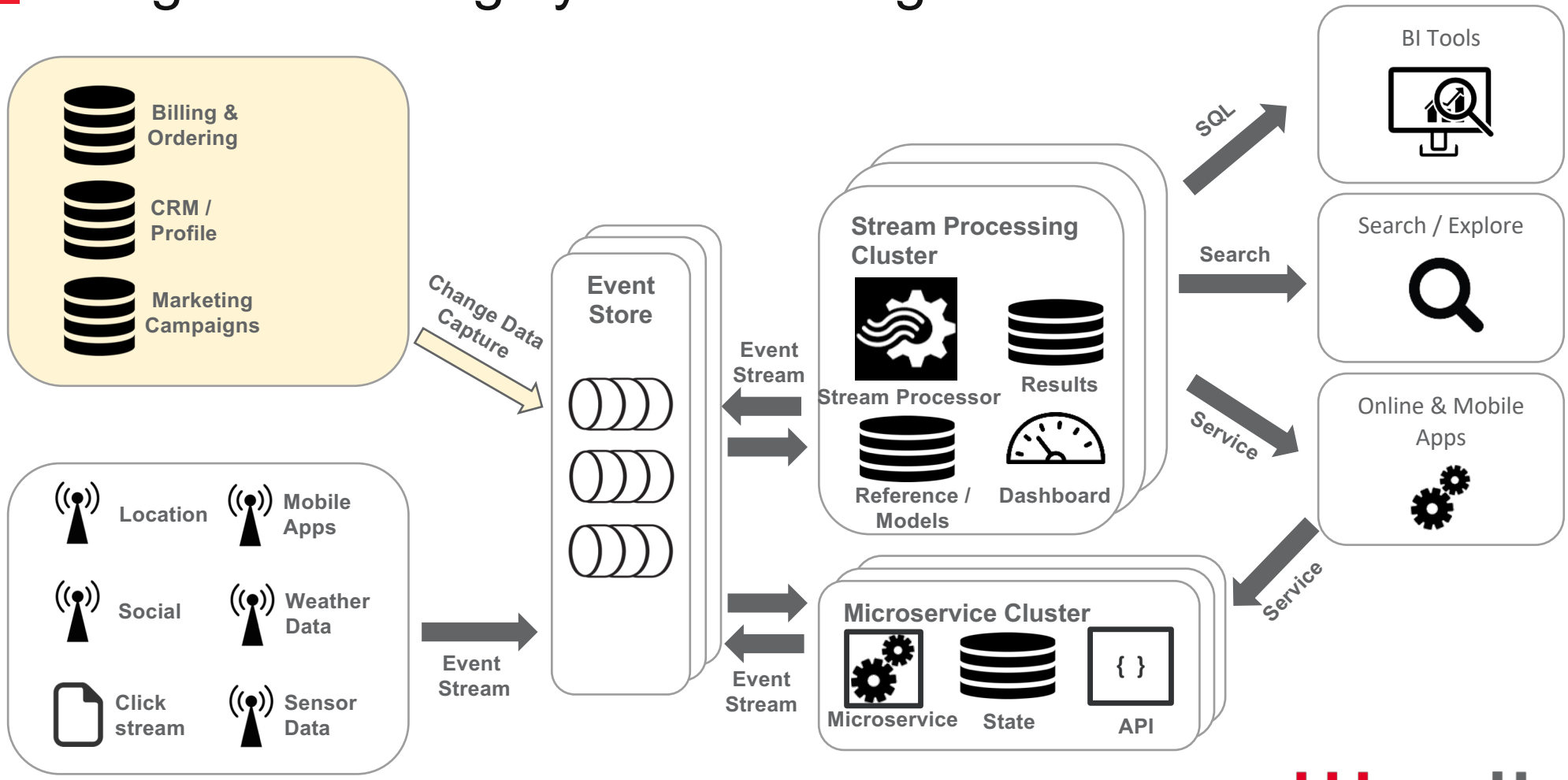
Change Data Capture (CDC) with Kafka





trivadis
makes IT easier.

Integrate existing systems through CDC

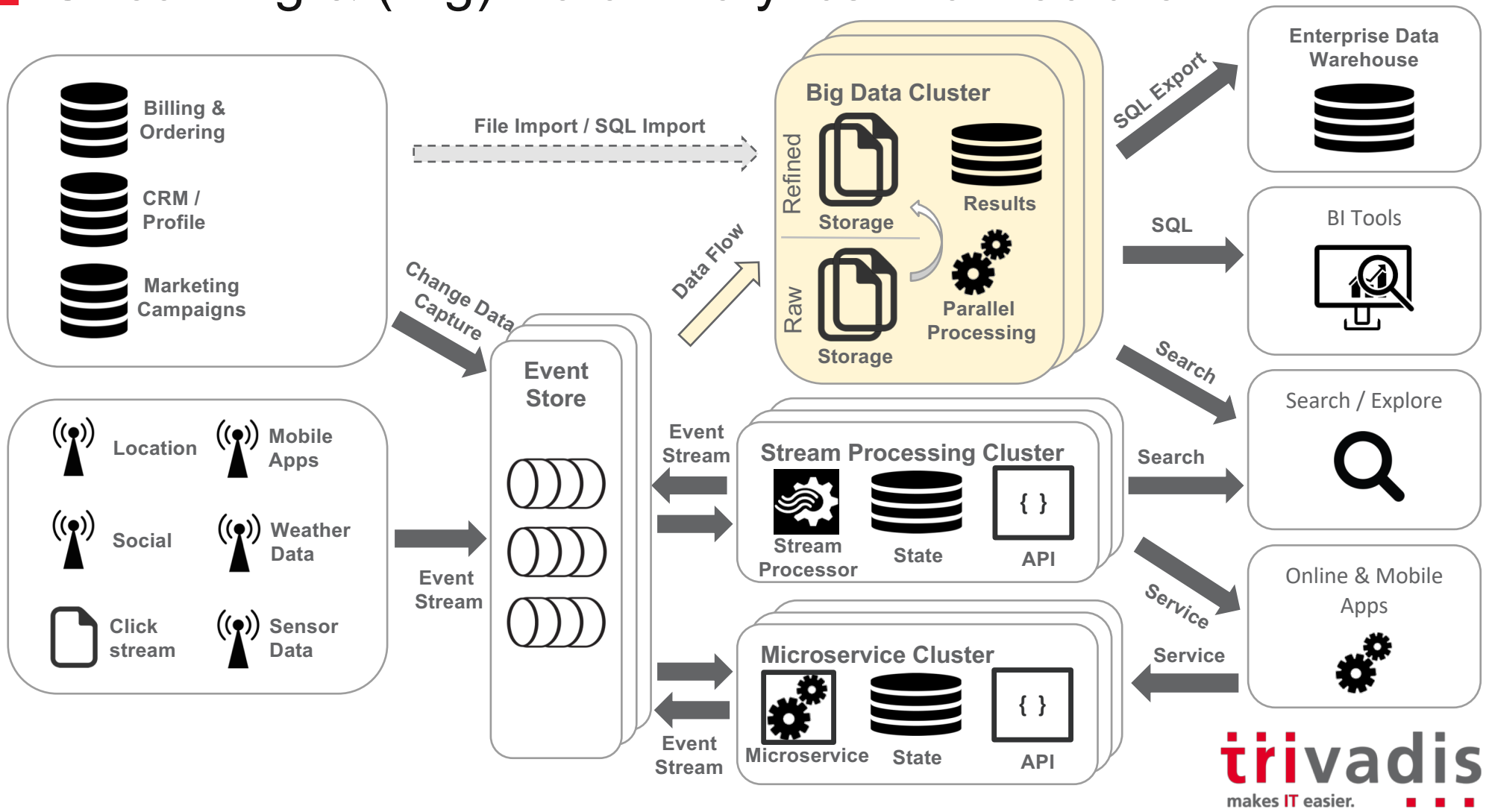


trivadis
makes IT easier.

What about (historical) data analytics?

trivadis
makes IT easier.

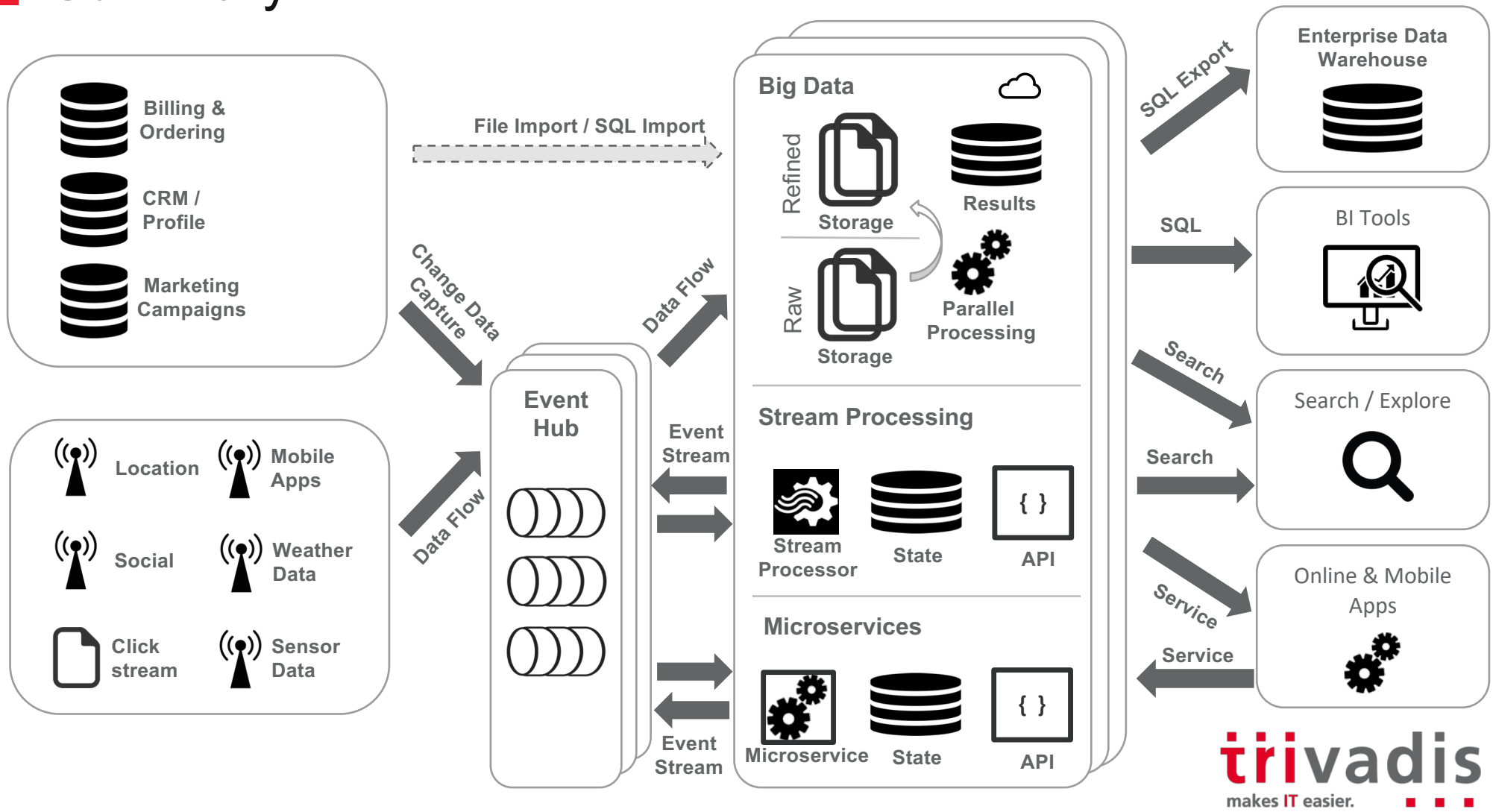
Streaming & (Big) Data Analytics Architecture



Summary

trivadis
makes IT easier.

Summary


Billing & Ordering
CRM / Profile
Marketing Campaigns
Location
Mobile Apps
Social
Weather Data
Click stream
Sensor Data
File Import / SQL Import
Change Data Capture
Data Flow
Data Flow
Event Hub
Event Stream
Event Stream
Big Data
Refined
Storage
Results
Raw
Storage
Parallel Processing
Stream Processing
Stream Processor
State
API
Microservices
Microservice
State
API
SQL Export
SQL
Search
Search
Service
Service
Enterprise Data Warehouse
BI Tools
Search / Explore
Online & Mobile Apps
trivadis
makes IT easier.

Summary

- service autonomy is key in a Microservices Architecture!
- not all communication need to be synchronous => separate into
  - commands
  - events
  - queries
- Kafka is well suited as an event broker / event store
  - brings many more interesting features beyond just “message passing”

References

Microservices Blog Series, Ben Stopford, Confluent:

- https://www.confluent.io/blog/tag/microservices

Apache Kafka for Microservices: A Confluent Online Talk Series:

- https://www.confluent.io/landing-page/microservices-online-talk-series/

Turning the database inside-out with Apache Samza, Martin Kleppmann, Con

- https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Event sourcing, CQRS, stream processing and Apache Kafka: What's the connection?, Neha Narkhede, Confluent:

- https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

Immutability Changes Everything, Pat Helland, Salesforce:

- http://cidrdb.org/cidr2015/Papers/CIDR15_Paper16.pdf

Commander: Better Distributed Applications through CQRS and Event Sourcing, Bobby Calderwood:

- https://www.youtube.com/watch?v=B1-gS0oEtYc

trivadis
makes IT easier.