confluent

Cloud Native Data Pipelines with Apache Kafka

Gwen Shapira, Software Engineer @gwenshap

What is a Cloud Native Application?

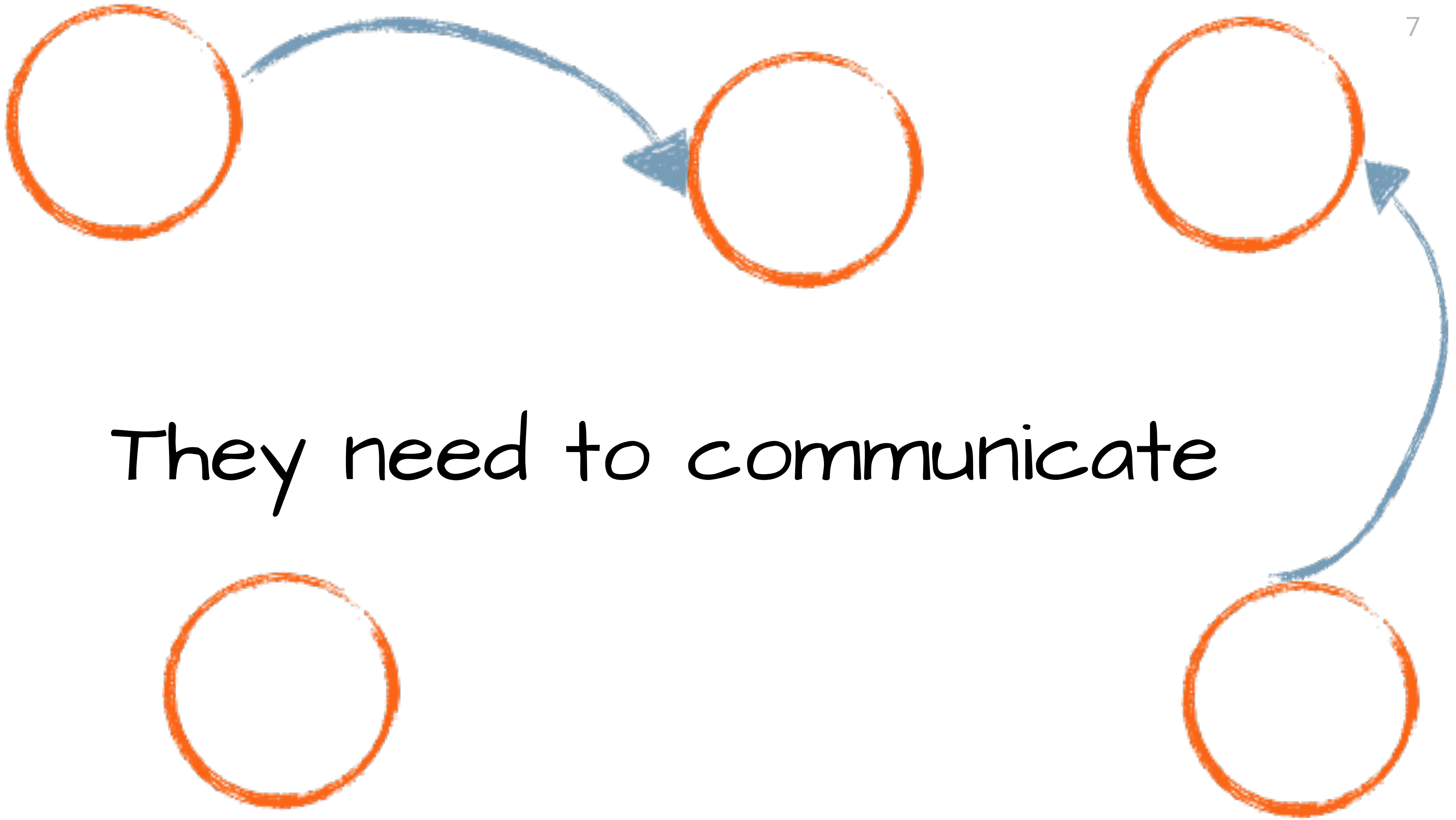
Common ideas

- Resilience
- Elasticity
- Agility
- DevOps

You will build
Cloud Native Applications
from
Non Cloud Native components

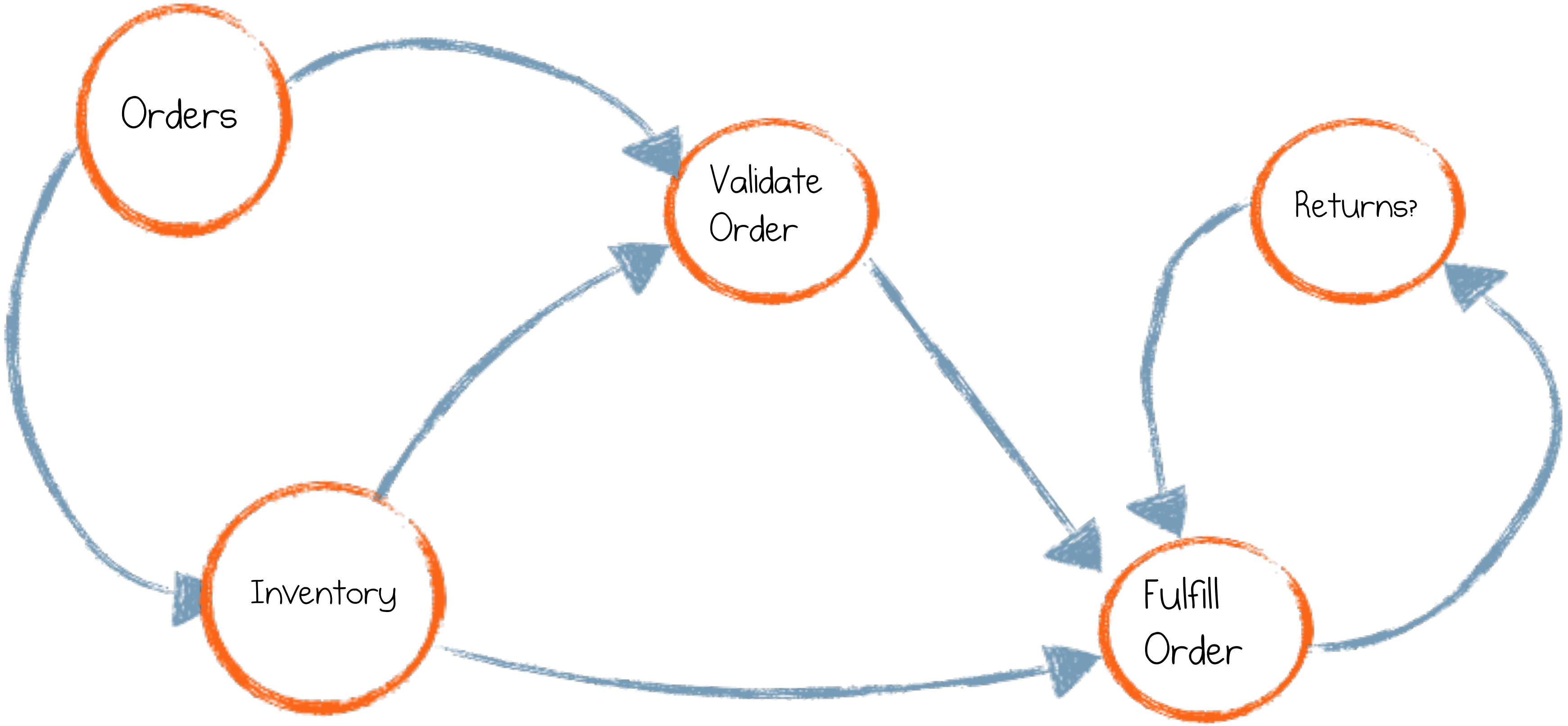
What do
Cloud Native architectures
look like?

You Have Microservices



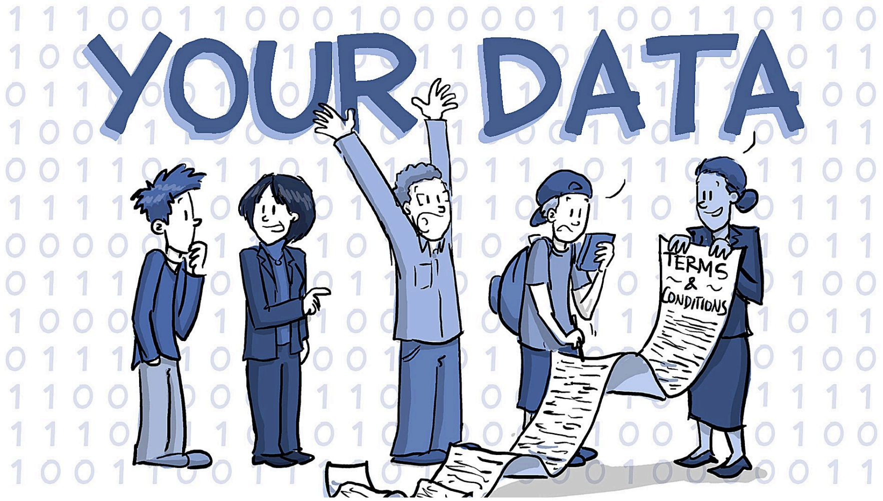
They need to communicate

I know! I'll use REST APIs



But, we forgot something…

The Problem is DATA



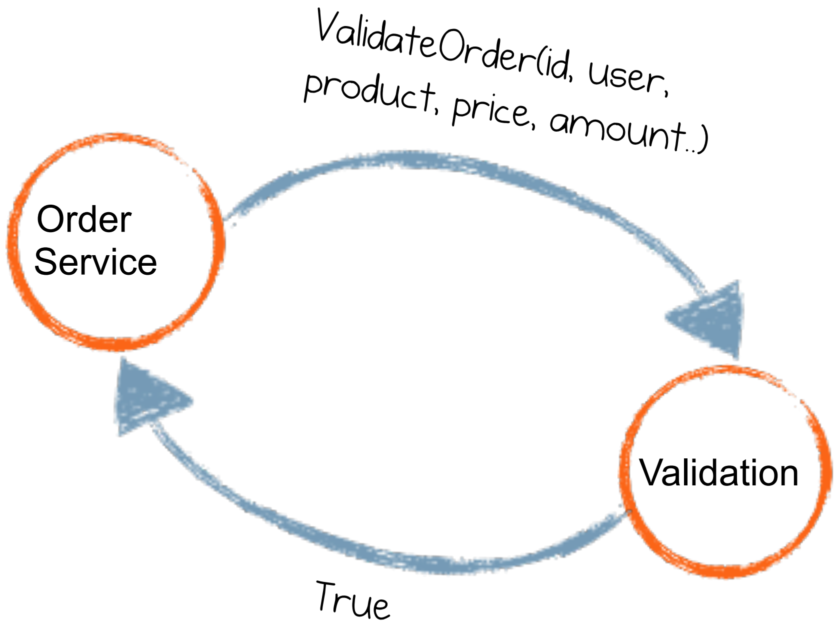
confluent

Cloud Native Architectures are Different.

We need data architectures for cloud.

And Data is about context and sharing

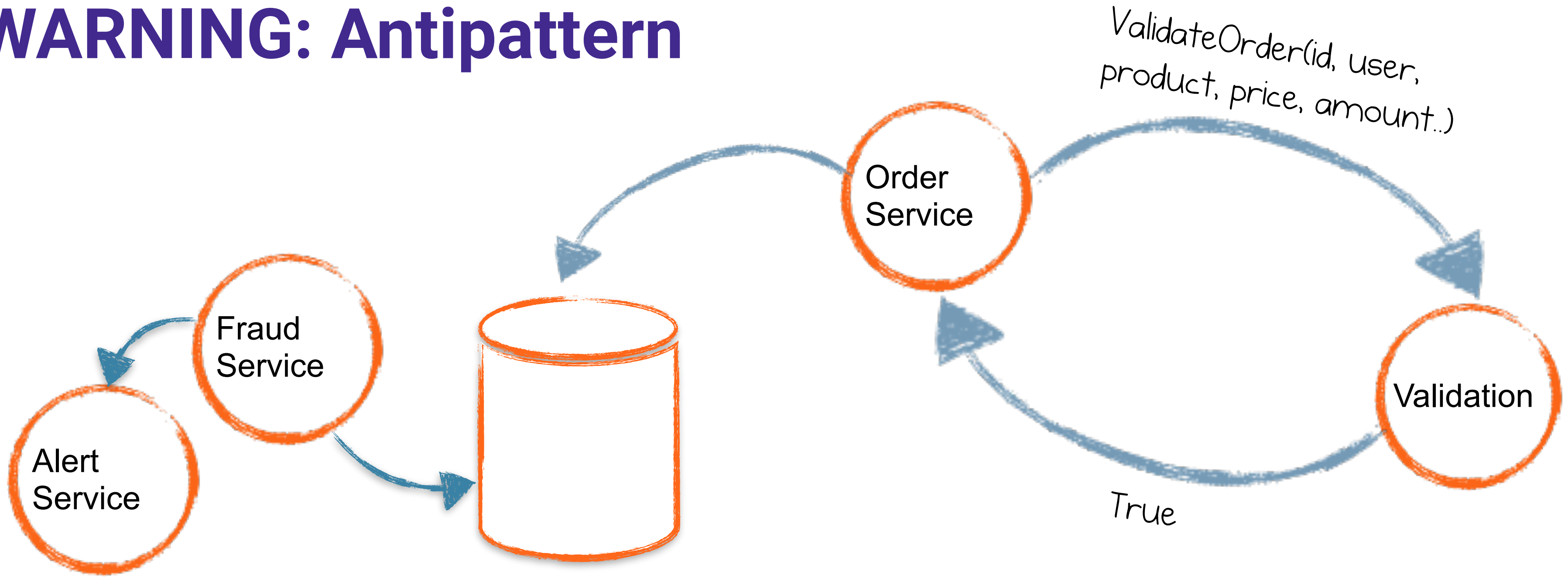
Lets say I have this:



We need Fraud Detection

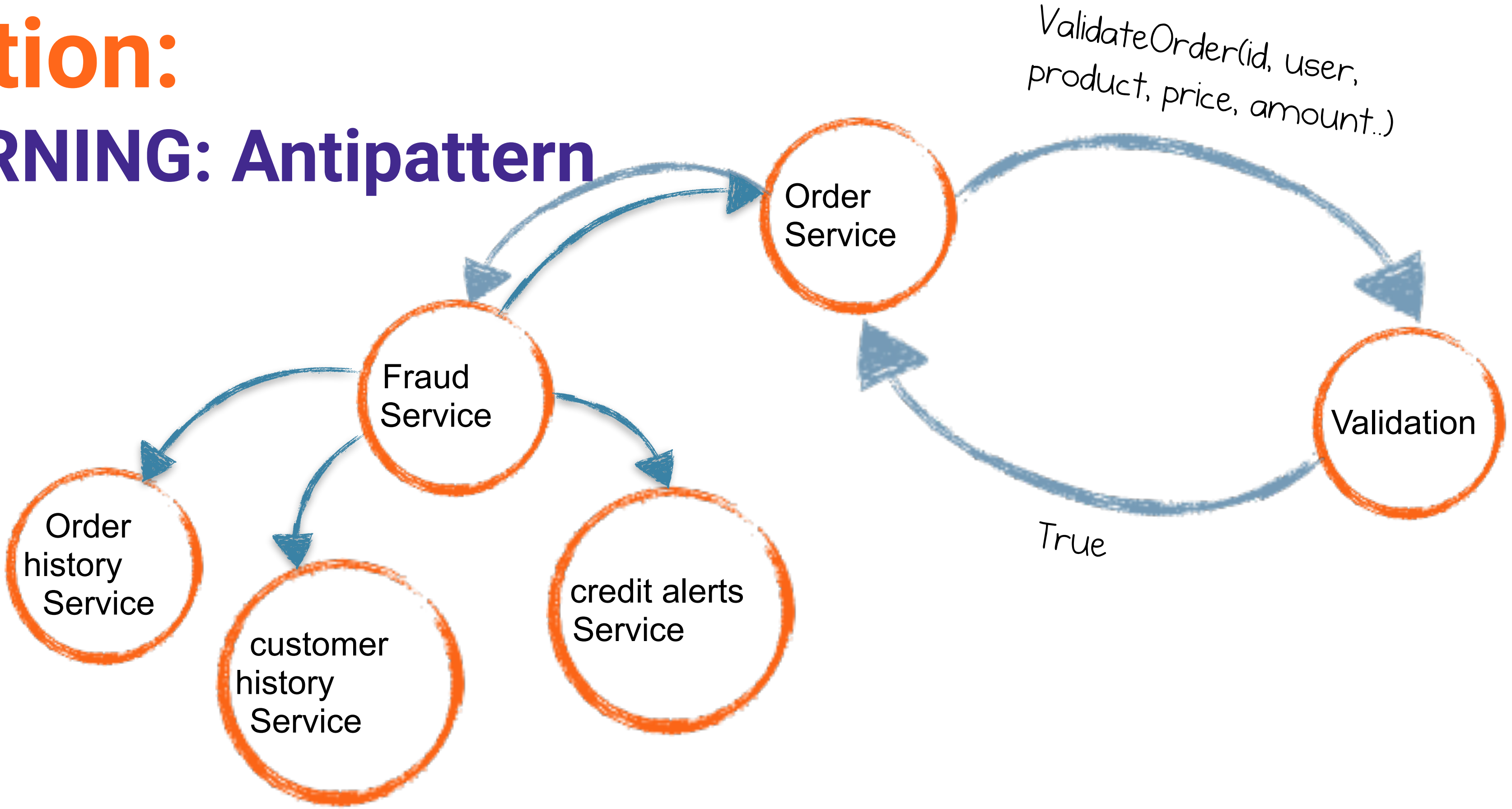
Option:

WARNING: Antipattern

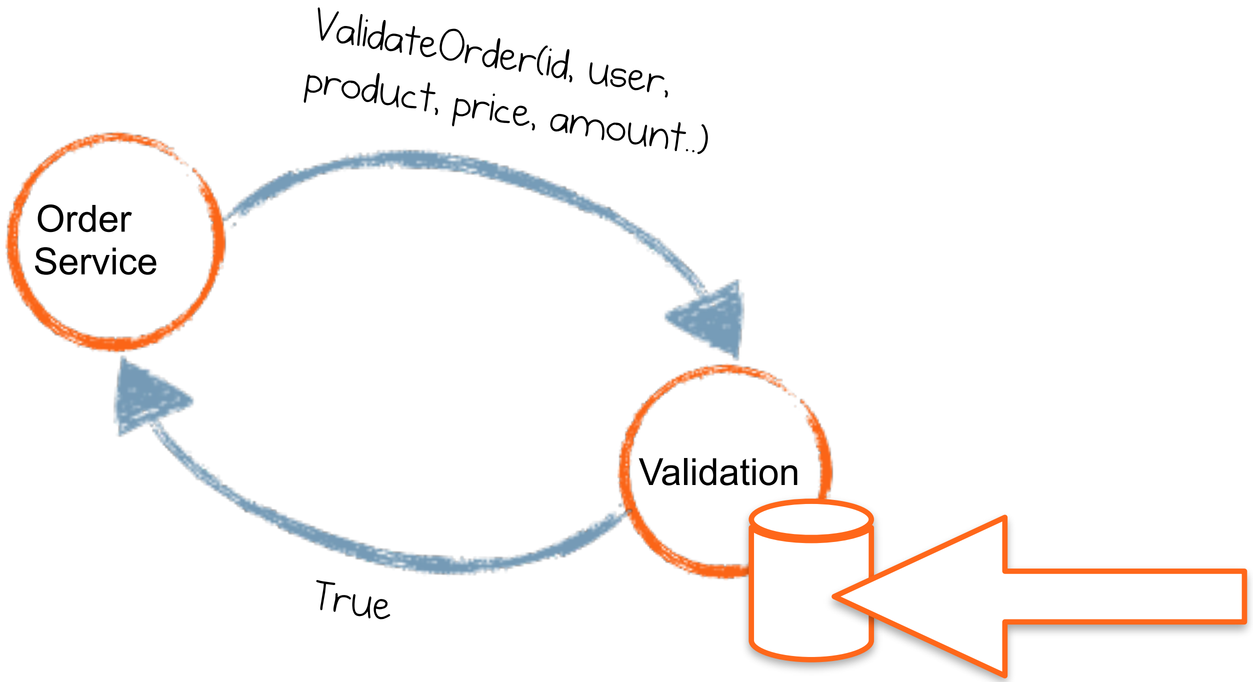


Option:

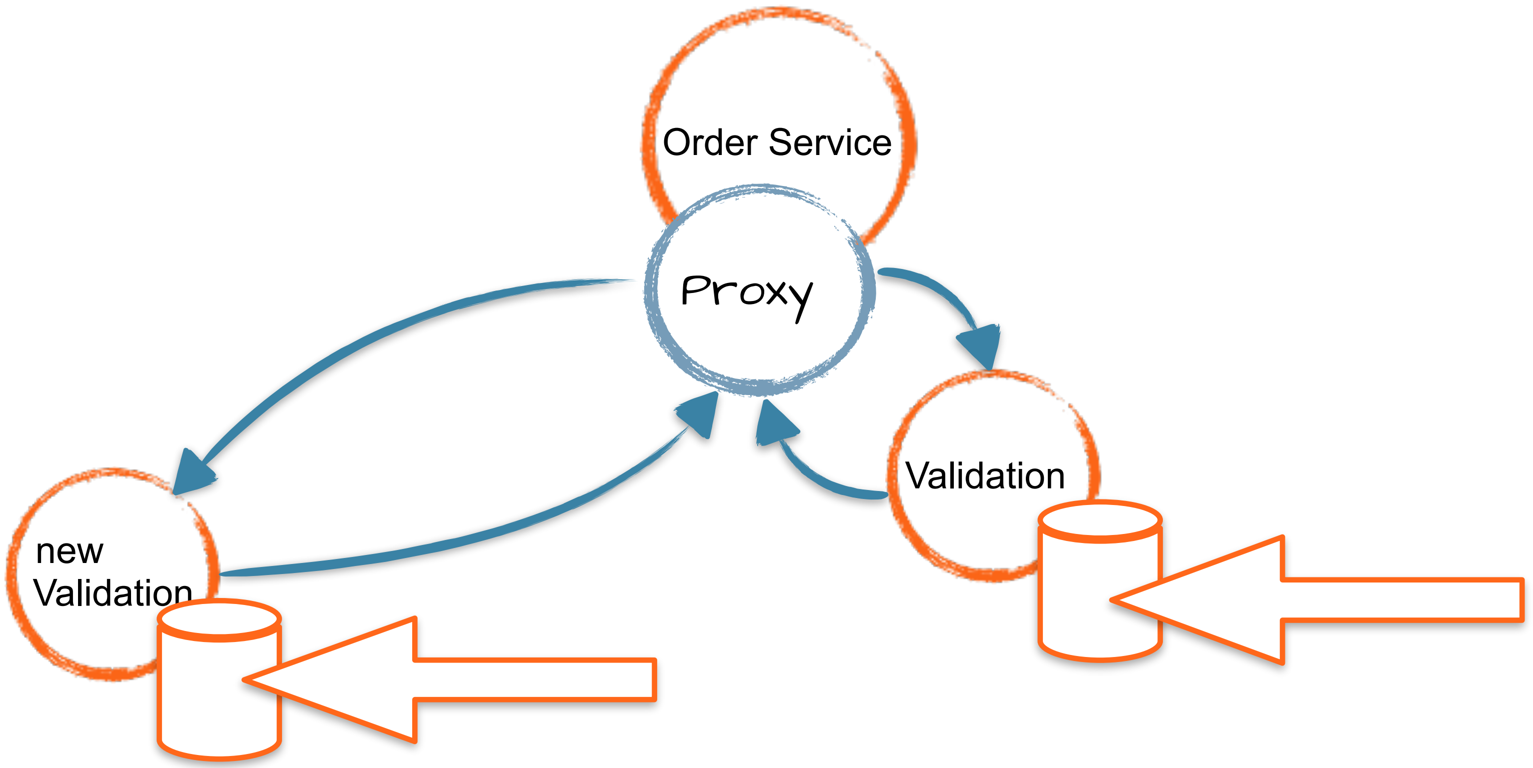
WARNING: Antipattern



What I want is really smart validator



Maybe even more than one



The challenges

- Services are really Stateful
- Data has history
- Data is shared

Lets Look at Patterns

Publish Events

Events are not:

- Commands
- Queries
- Requests

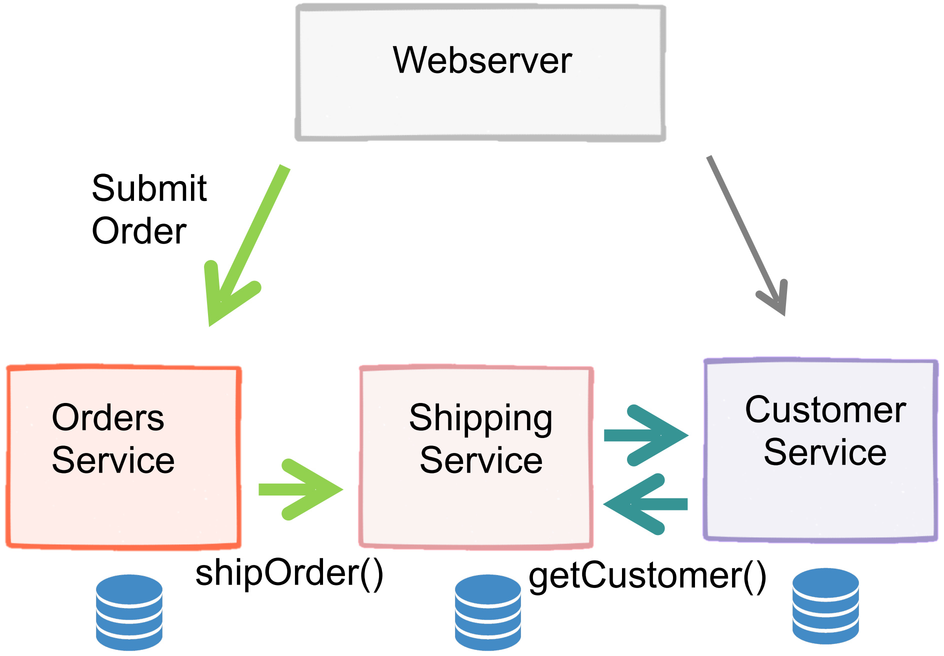
Events are:

- Things that happened
- Notification
- Data

confluent

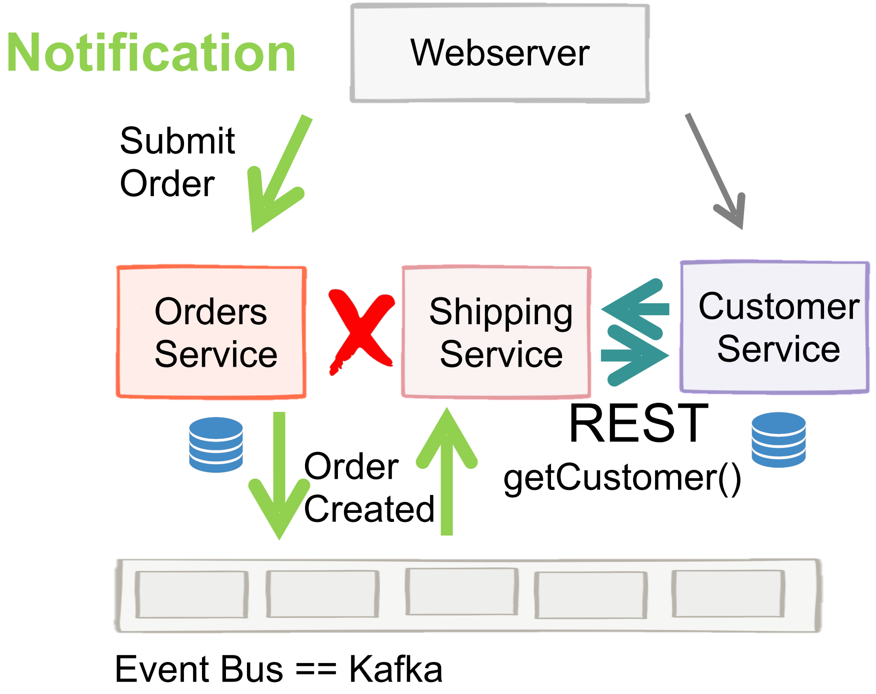
Buying an iPad (with REST)

- Orders Service calls Shipping Service to tell it to ship item.
- Shipping service looks up address to ship to (from Customer Service)



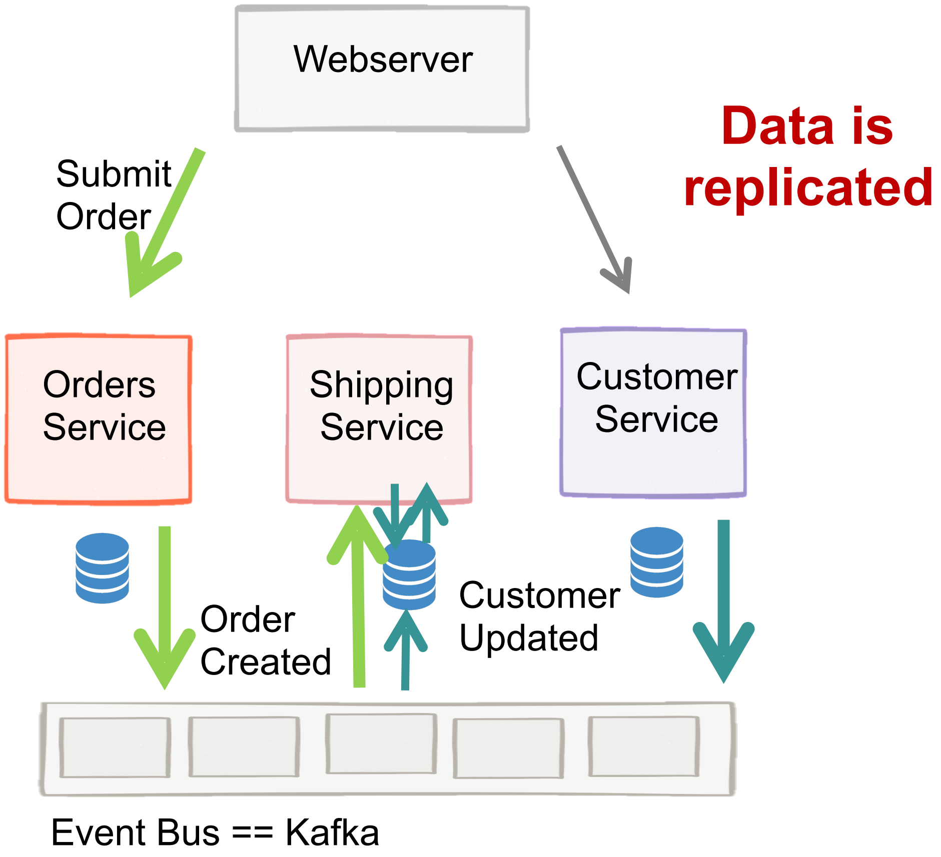
Using events for Notification

- Orders Service no longer knows about the Shipping service (or any other service). Events are fire and forget.



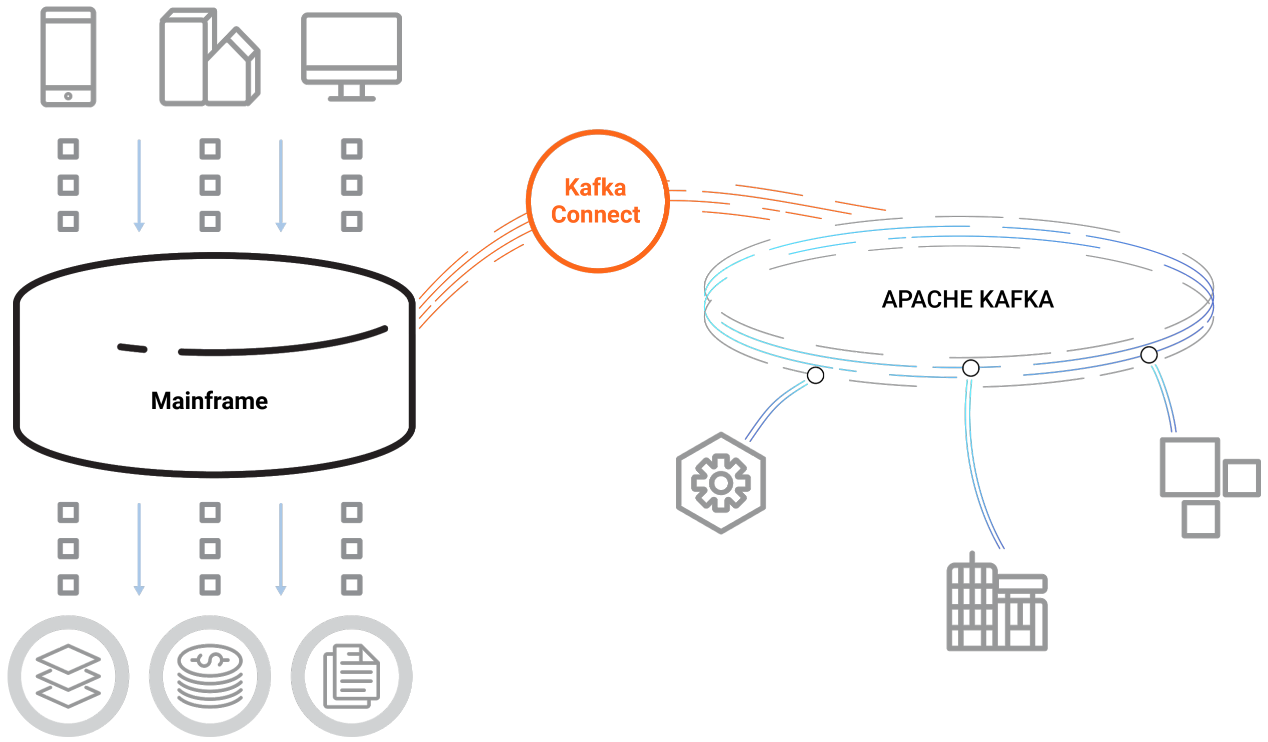
Using events to share facts

- Call to Customer service is gone.
- Instead data in replicated, as events, into the shipping service, where it is queried locally. .



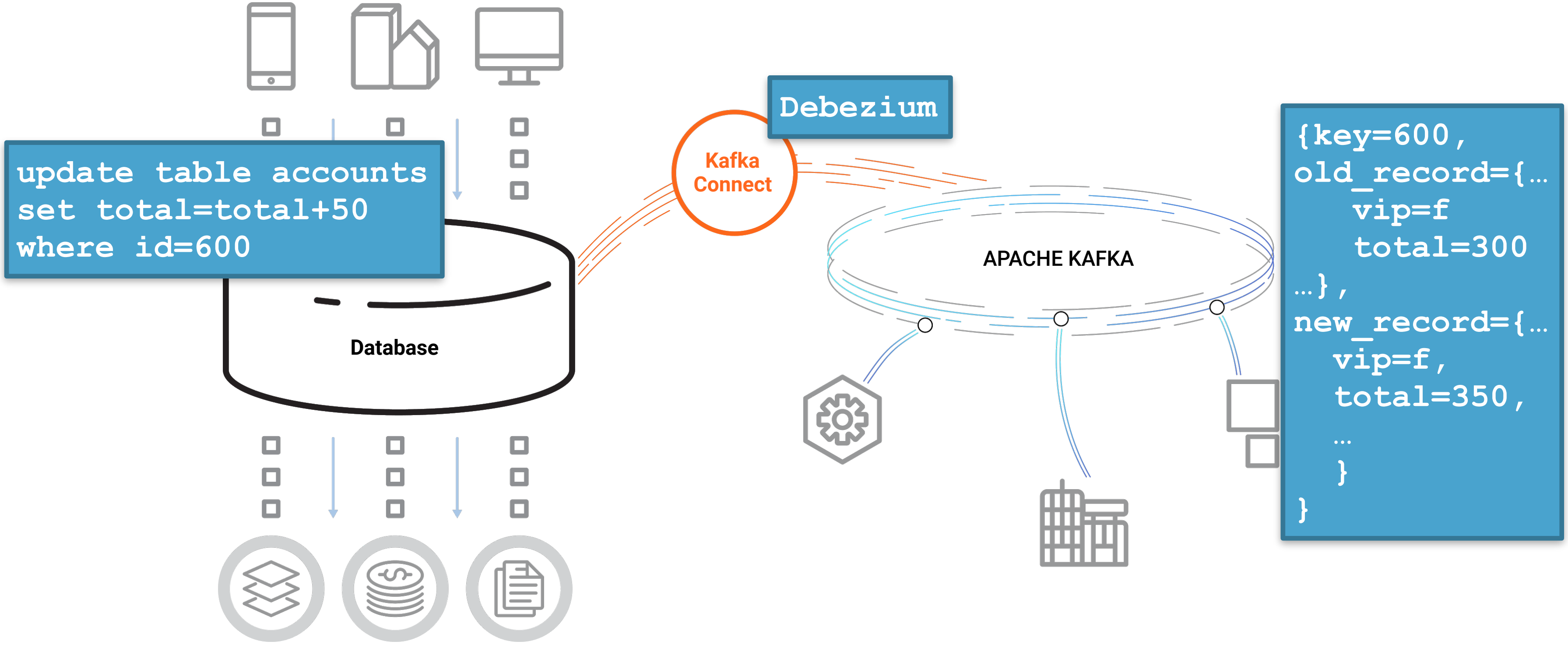
Need someone else's events?

Change Data Capture



Need someone else's events?

Change Data Capture



```
update table accounts
set total=total+50
where id=600
```

```
{key=600,
old_record={…
   vip=f
   total=300
…},
new_record={…
  vip=f,
  total=350,
  …
  }
}
```

Local state
for Microservices

We have a stream of events:

```
{order:1,
 product: iphone,
 status: created
}
```

event 1

```
{order:1,
 product: iphone,
 status: valid
}
```

event 2

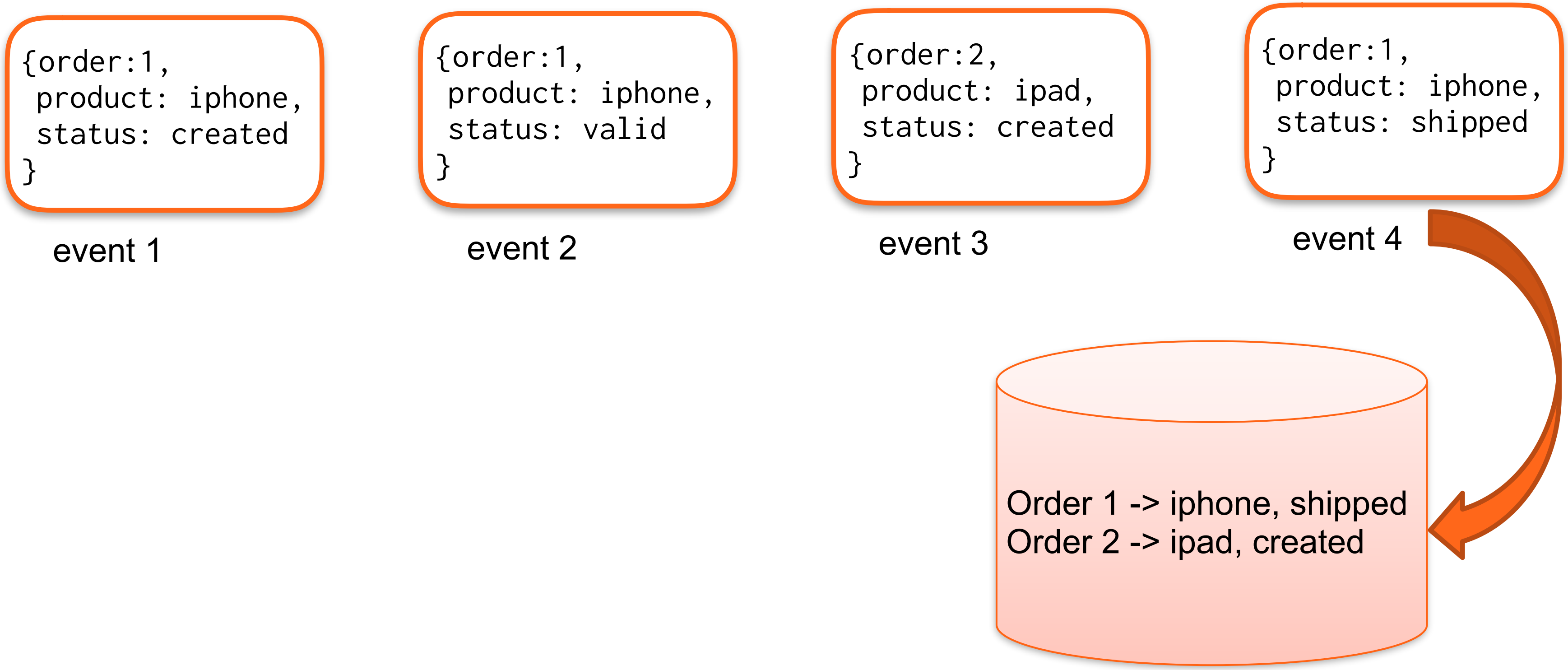
```
{order:2,
 product: ipad,
 status: created
}
```

event 3

```
{order:1,
 product: iphone,
 status: shipped
}
```

event 4

Store current state:



```
{order:1,
 product: iphone,
 status: created
}
```

```
{order:1,
 product: iphone,
 status: valid
}
```

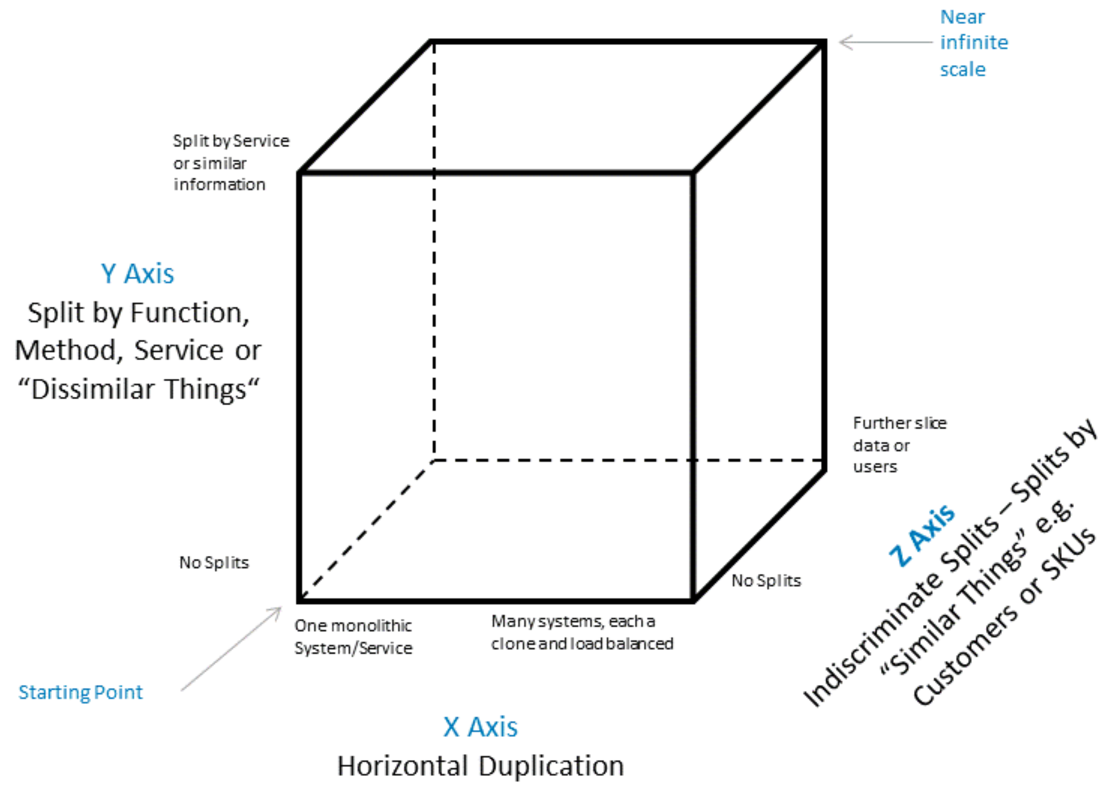
```
{order:2,
 product: ipad,
 status: created
}
```

```
{order:1,
 product: iphone,
 status: shipped
}
```

Duplicate data?

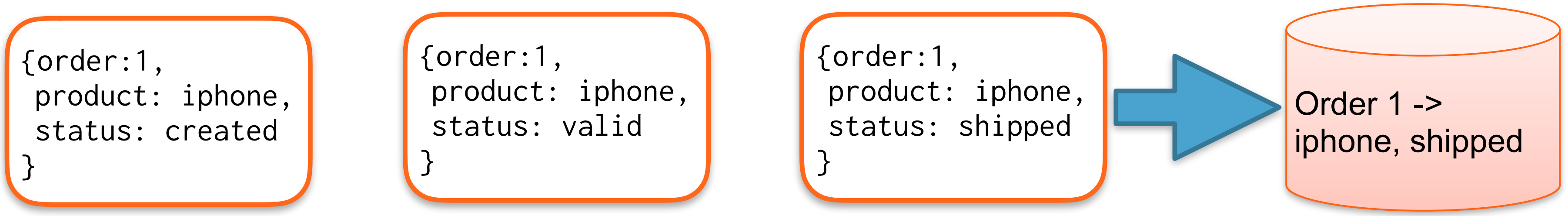
- Low risk due to shared event stream
- Just the data you need
- Sharded with the application
-

AKF Scale Cube


Near infinite scale
Split by Service or similar information
Y Axis
Split by Function, Method, Service or "Dissimilar Things"
No Splits
Starting Point
One monolithic System/Service
Many systems, each a clone and load balanced
X Axis
Horizontal Duplication
No Splits
Further slice data or users
Z Axis
Indiscriminate Splits – Splits by "Similar Things" e.g. Customers or SKUs

Sharded View

Odd orders:

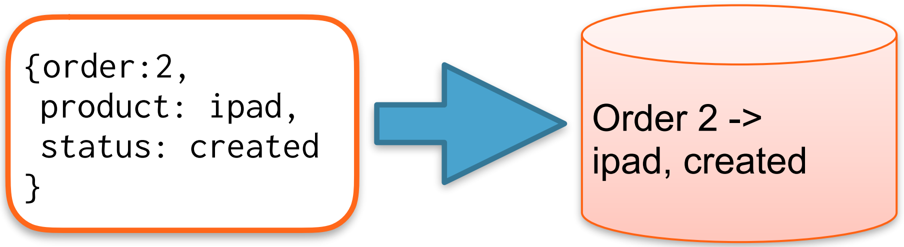


```
{order:1,
 product: iphone,
 status: created
}
```

```
{order:1,
 product: iphone,
 status: valid
}
```

```
{order:1,
 product: iphone,
 status: shipped
}
```

Even orders:

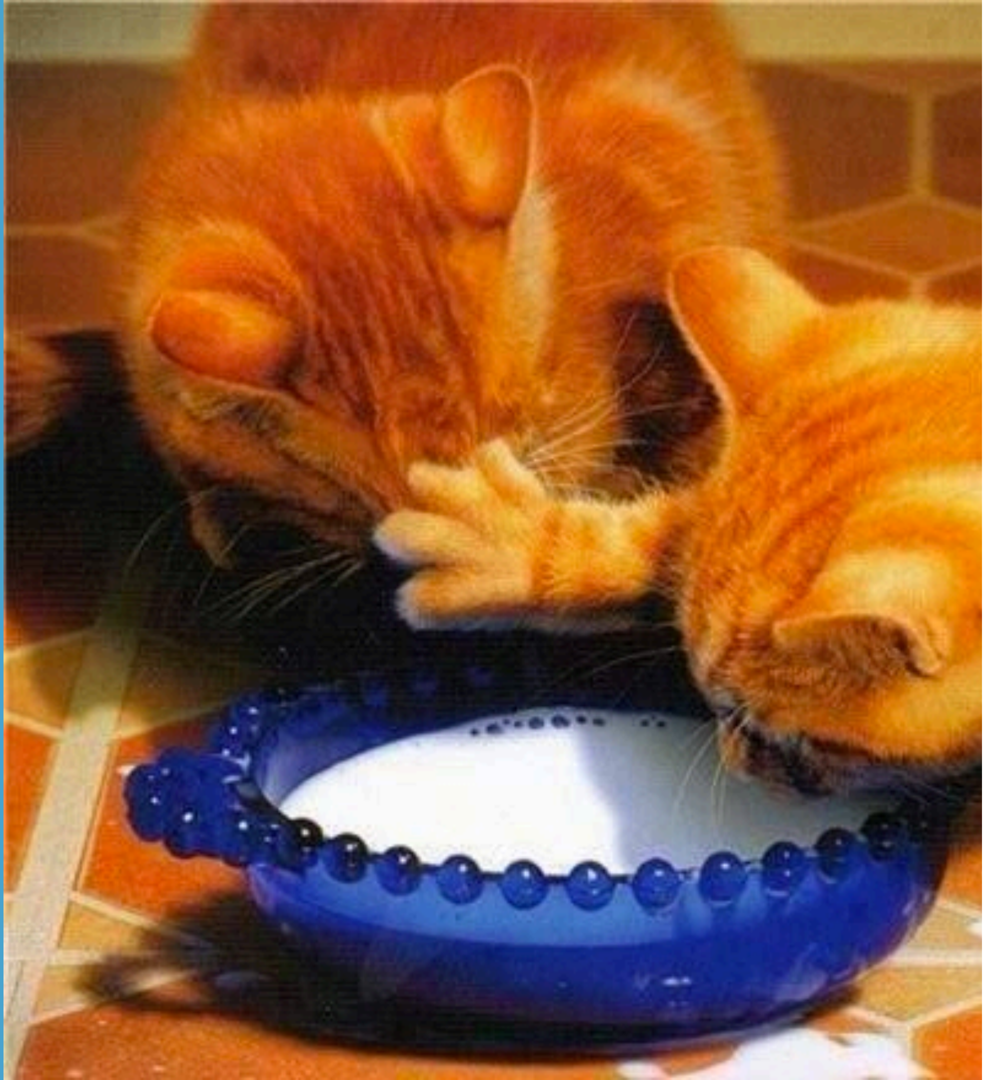


```
{order:2,
 product: ipad,
 status: created
}
```

confluent

Better than shared DB

- The data I need, the way I need it
- Reduced dependencies
- Low latency
- Events are also triggers

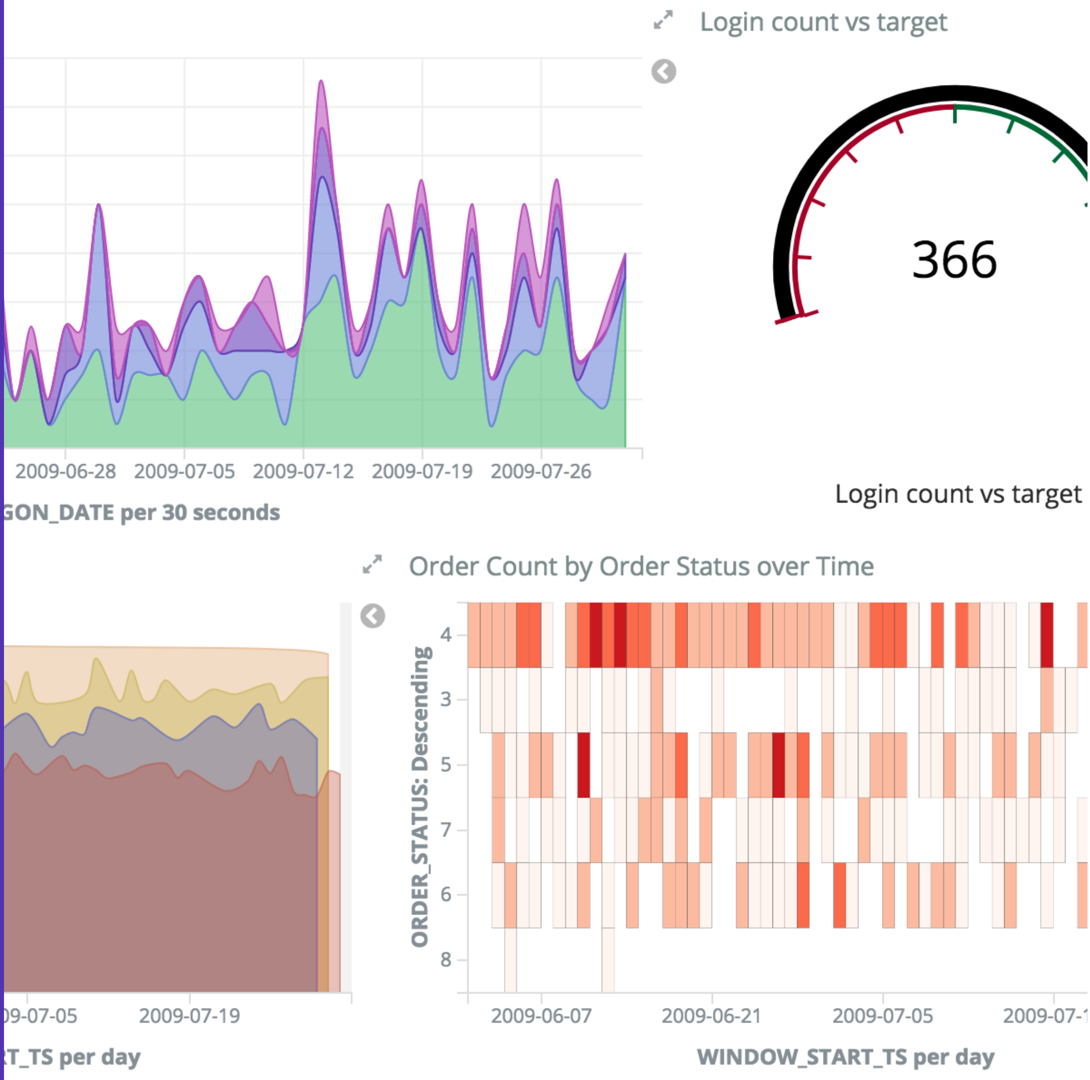


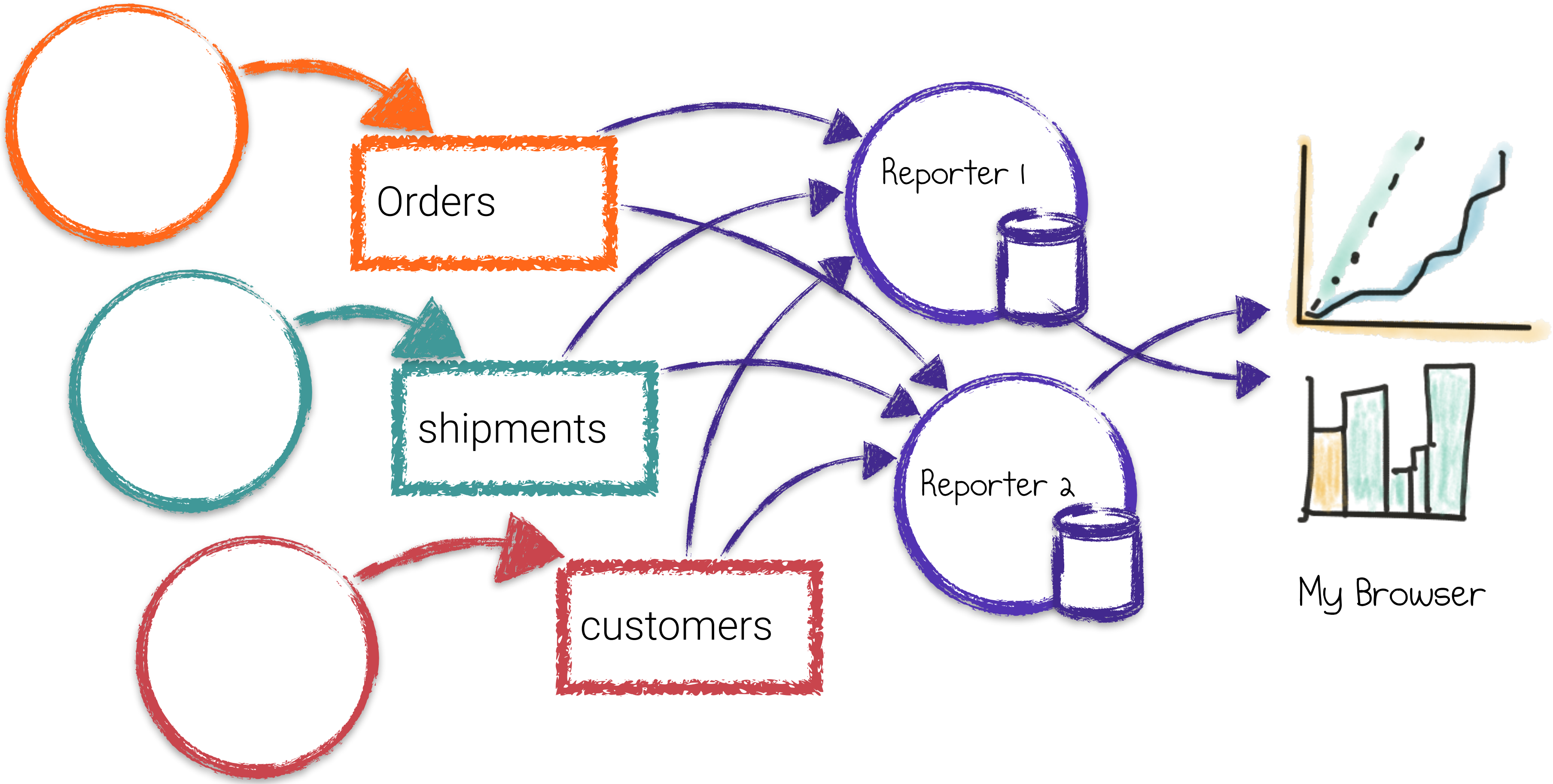
```
select order_id,
customer_id, product where
total_value>10000
…
And also, if you get one
like that in the future,
execute callback()
```

Reporting Live from Streams of Events

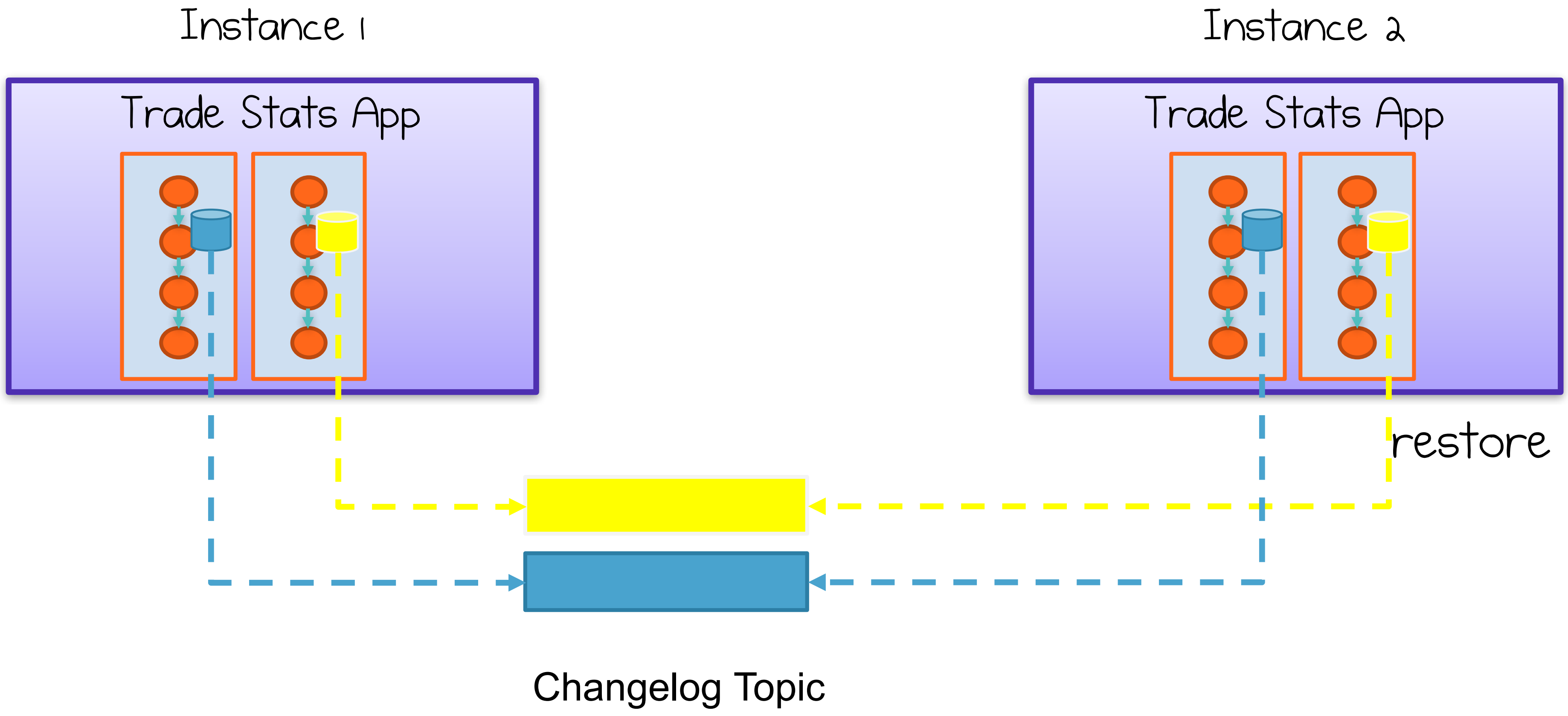
Requirements

- Aggregated reports
- Combining data from many services
- Updated in real time
- Scalable and resilient




Orders
shipments
customers
Reporter 1
Reporter 2
My Browser

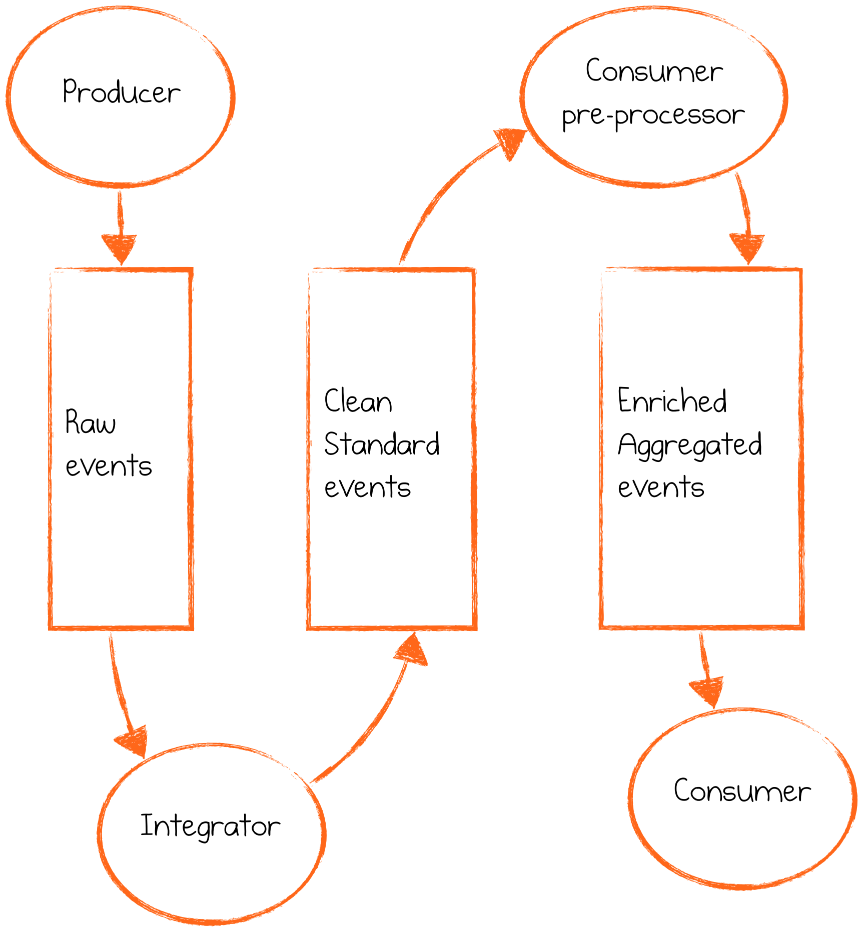
State Recovery



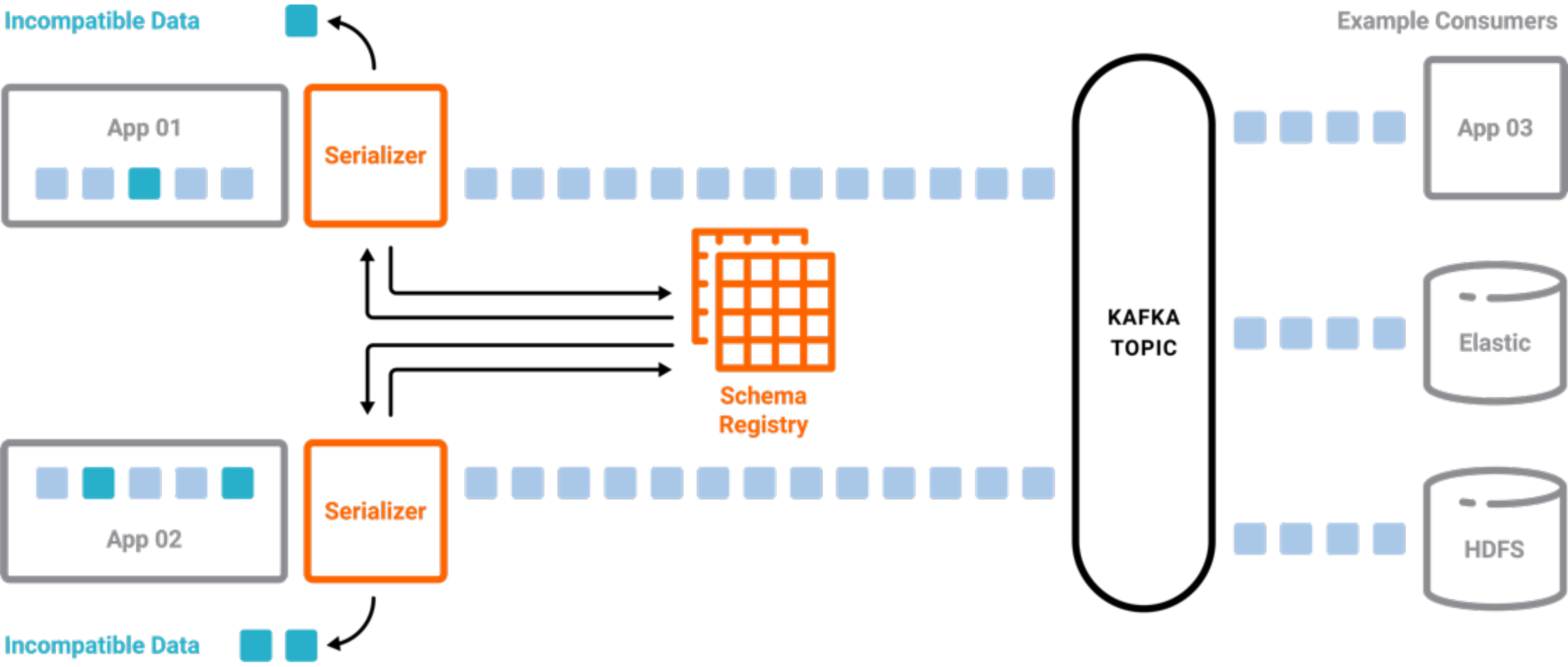
3-layer data model

Who controls the data format?

- Publishers?
- Consumers?
- How do we share events?



In **Event Streaming** World
Event Schemas ARE the API


Incompatible Data
Example Consumers
App 01
Serializer
App 03
Schema
Registry
KAFKA
TOPIC
Elastic
App 02
Serializer
HDFS
Incompatible Data

confluent

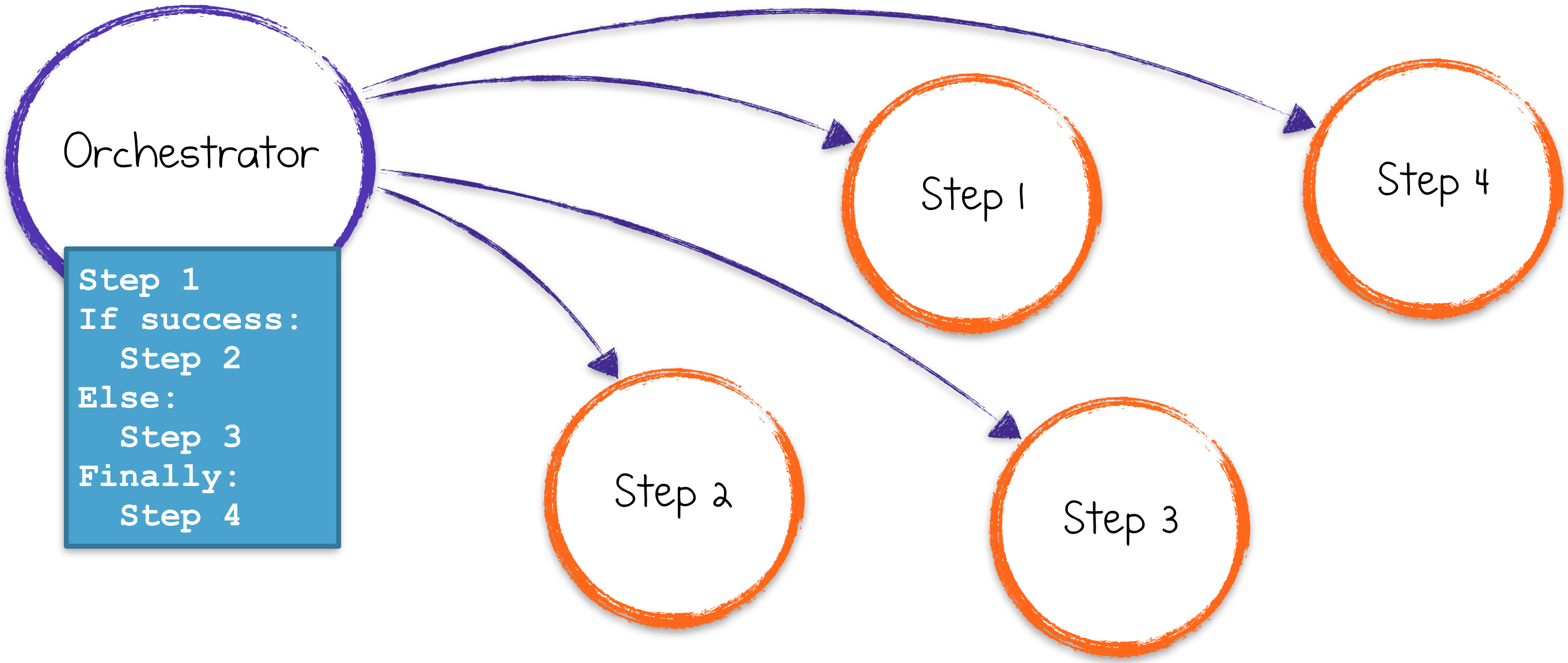
Take Away Points!

Remember This

- As you design cloud-native architectures
  - don't forget the data
- Publish events
- Build views and reports from events
- Be nice to each other

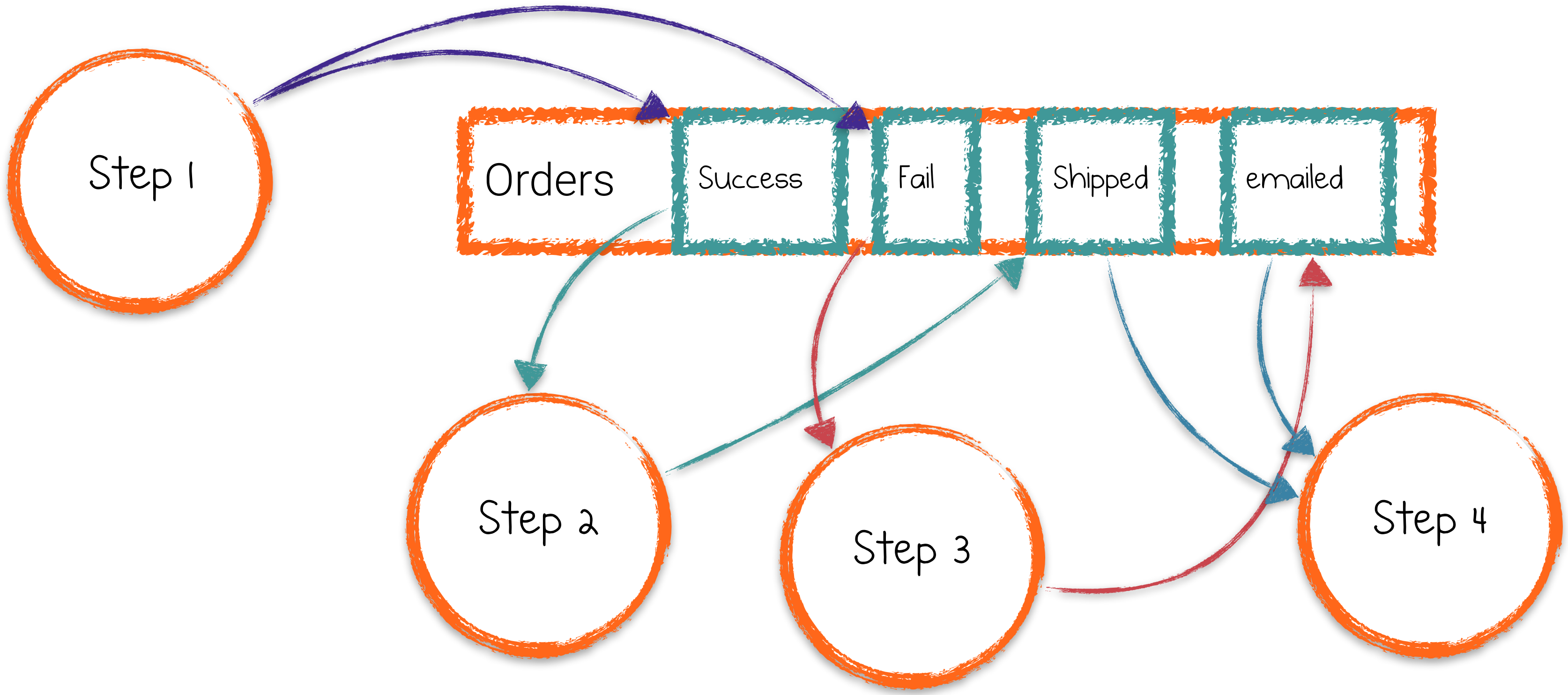
Orchestration vs Choreography

Orchestration: One Service to Rule them all



```
Step 1
If success:
   Step 2
Else:
   Step 3
Finally:
   Step 4
```

Choreography: We react to each other


Step 1
Orders
Success
Fail
Shipped
emailed
Step 2
Step 3
Step 4