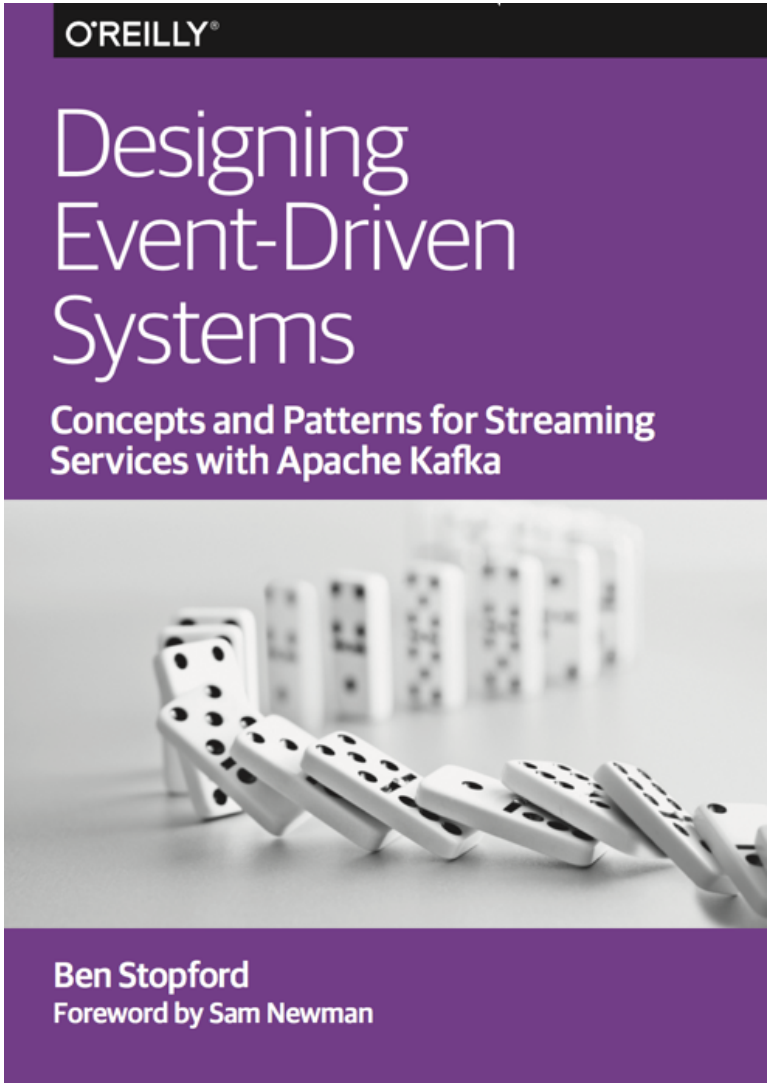
Building Event Driven Services with Apache Kafka and Kafka Streams

Ben Stopford

@benstopford

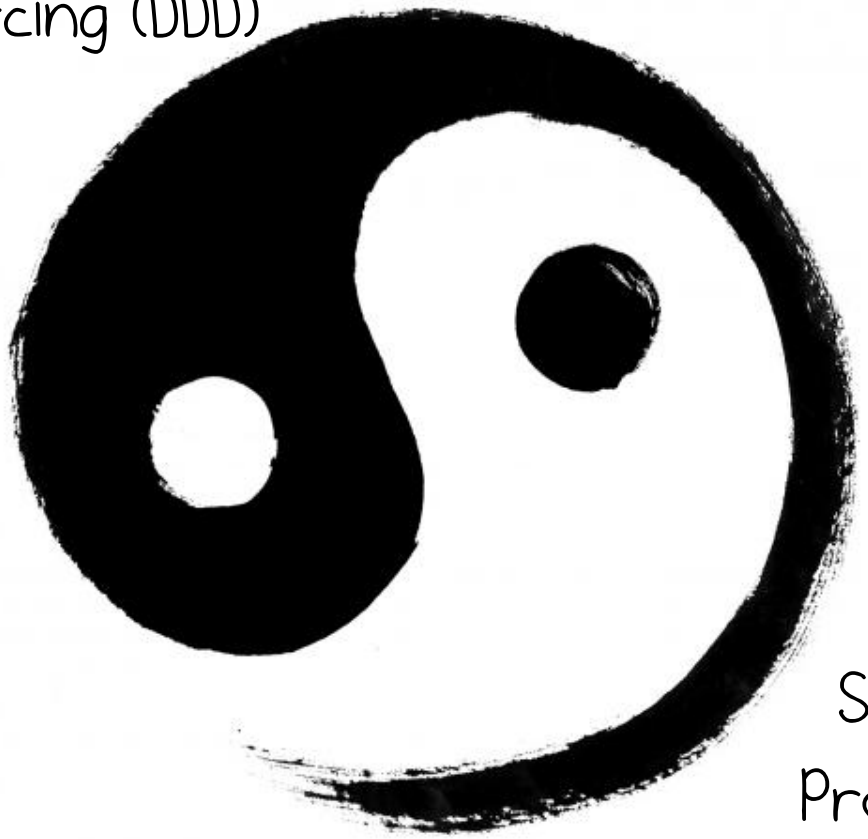
There is a book!

http://bit.ly/designing-event-driven-systems



Event Driven Architectures

Event Sourcing (DDD)



Stream Processing

Today's ecosystems get pretty big

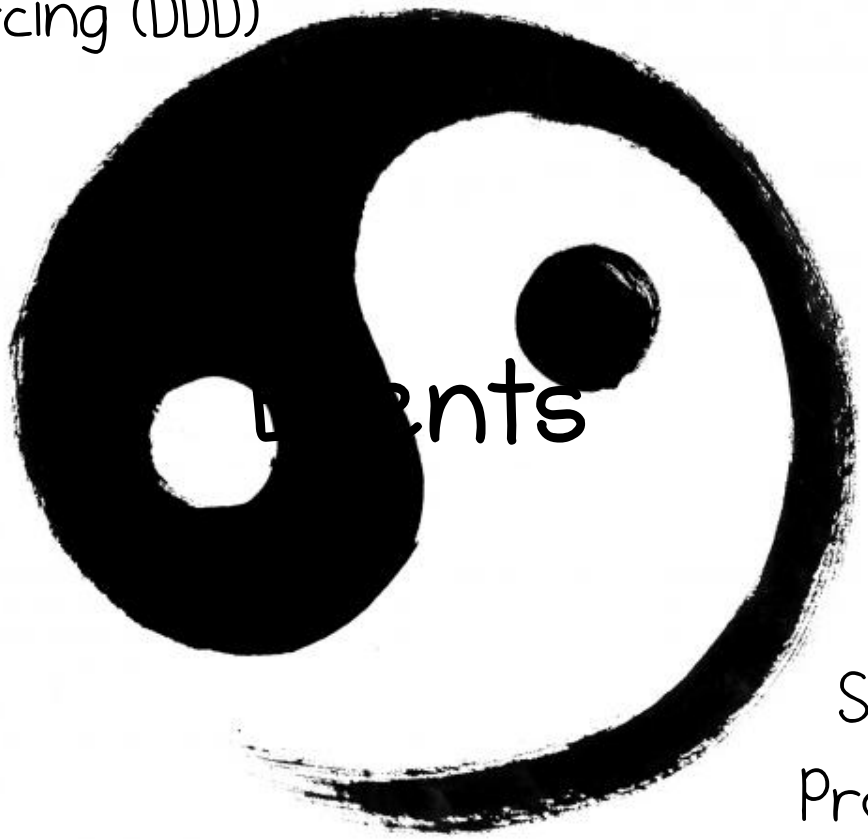
- 2.2 trillion messages per day (6 Petabytes)
- Up to 400 Microservices pre cluster.
- 20-200 Brokers per cluster

Today's ecosystems get pretty big

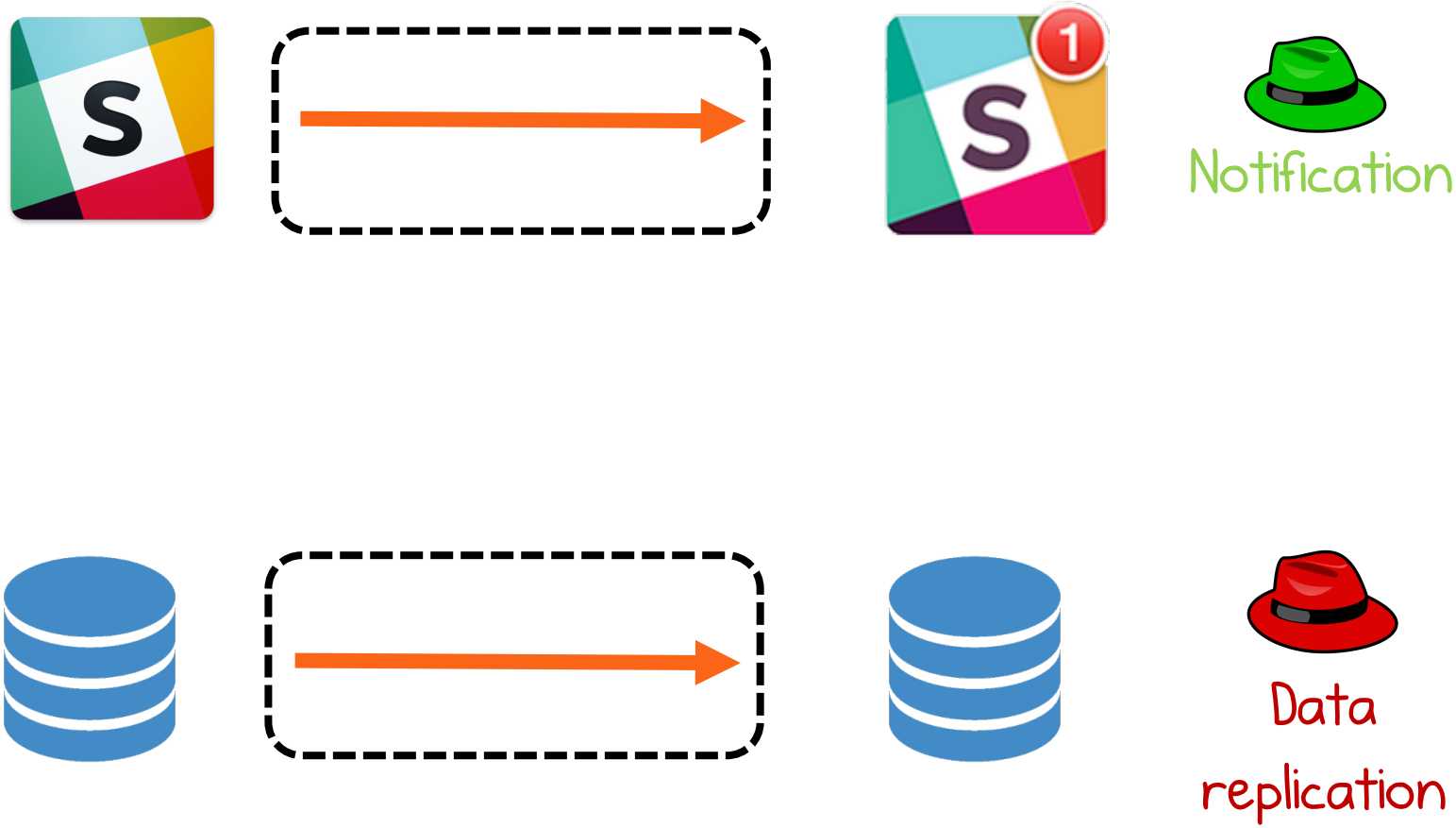
- 1 billion messages per day
- 20,000 messages per second
- 100 teams

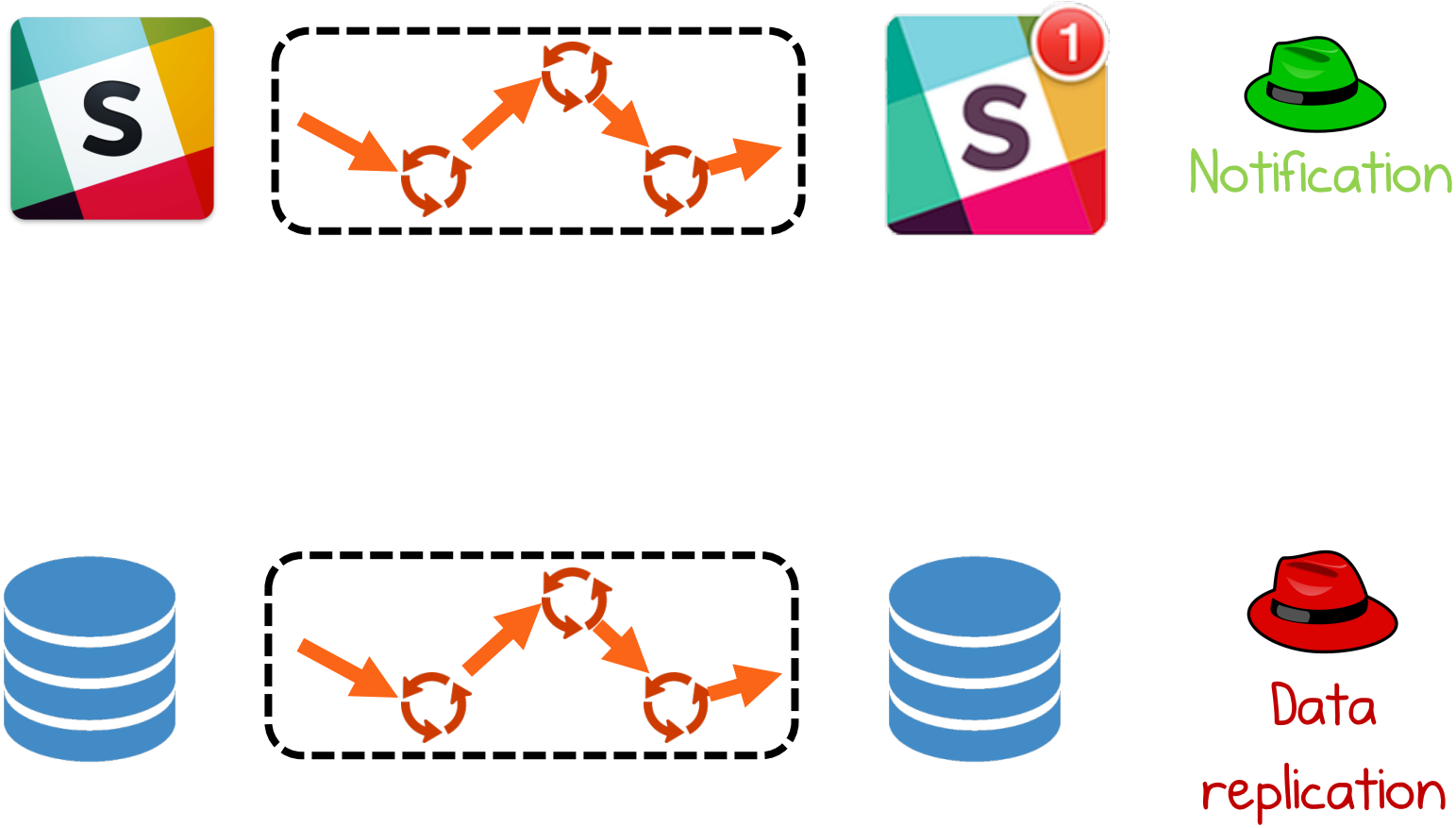
Event Driven Architectures

Event Sourcing (DDD)

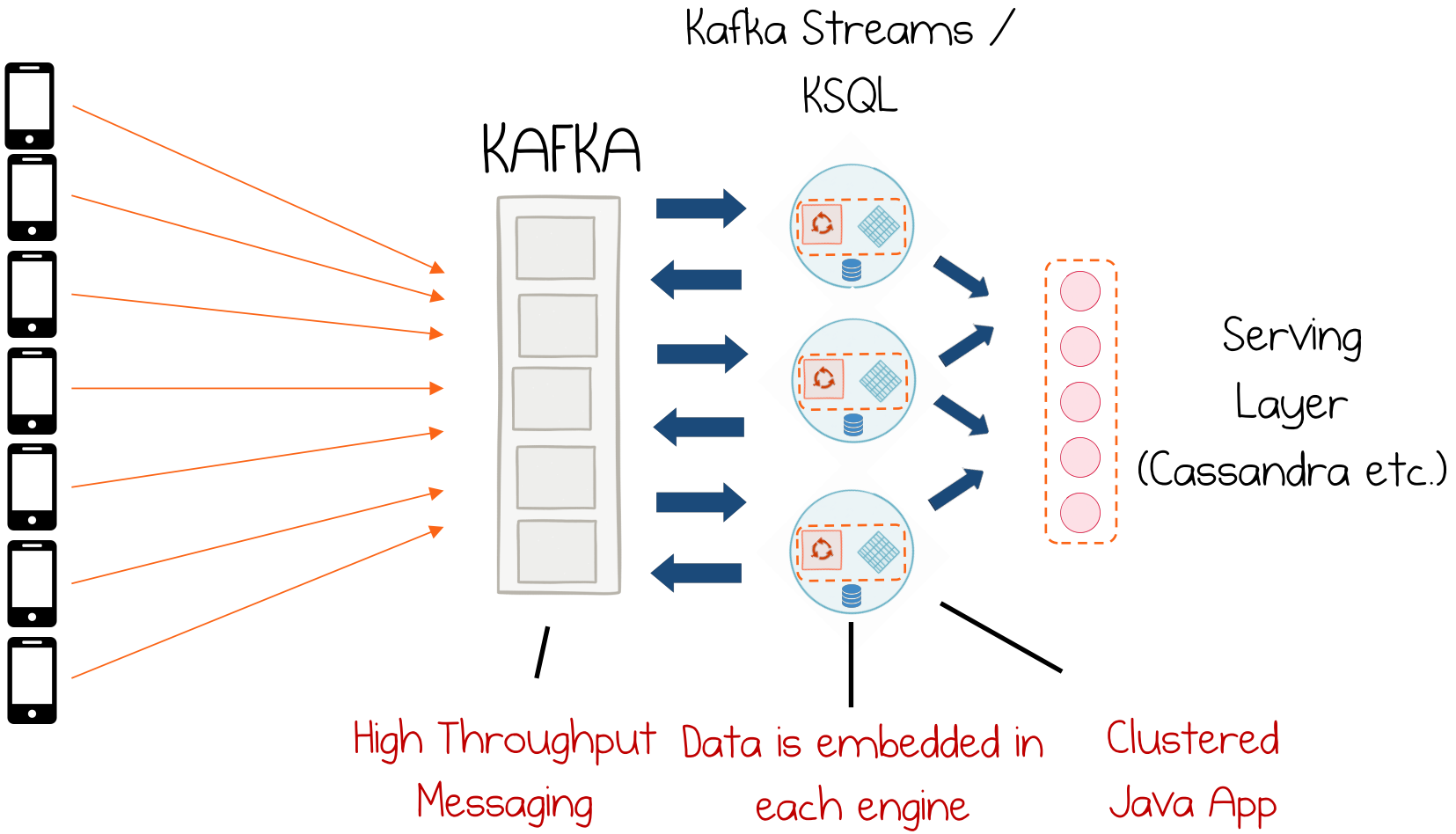


Stream Processing

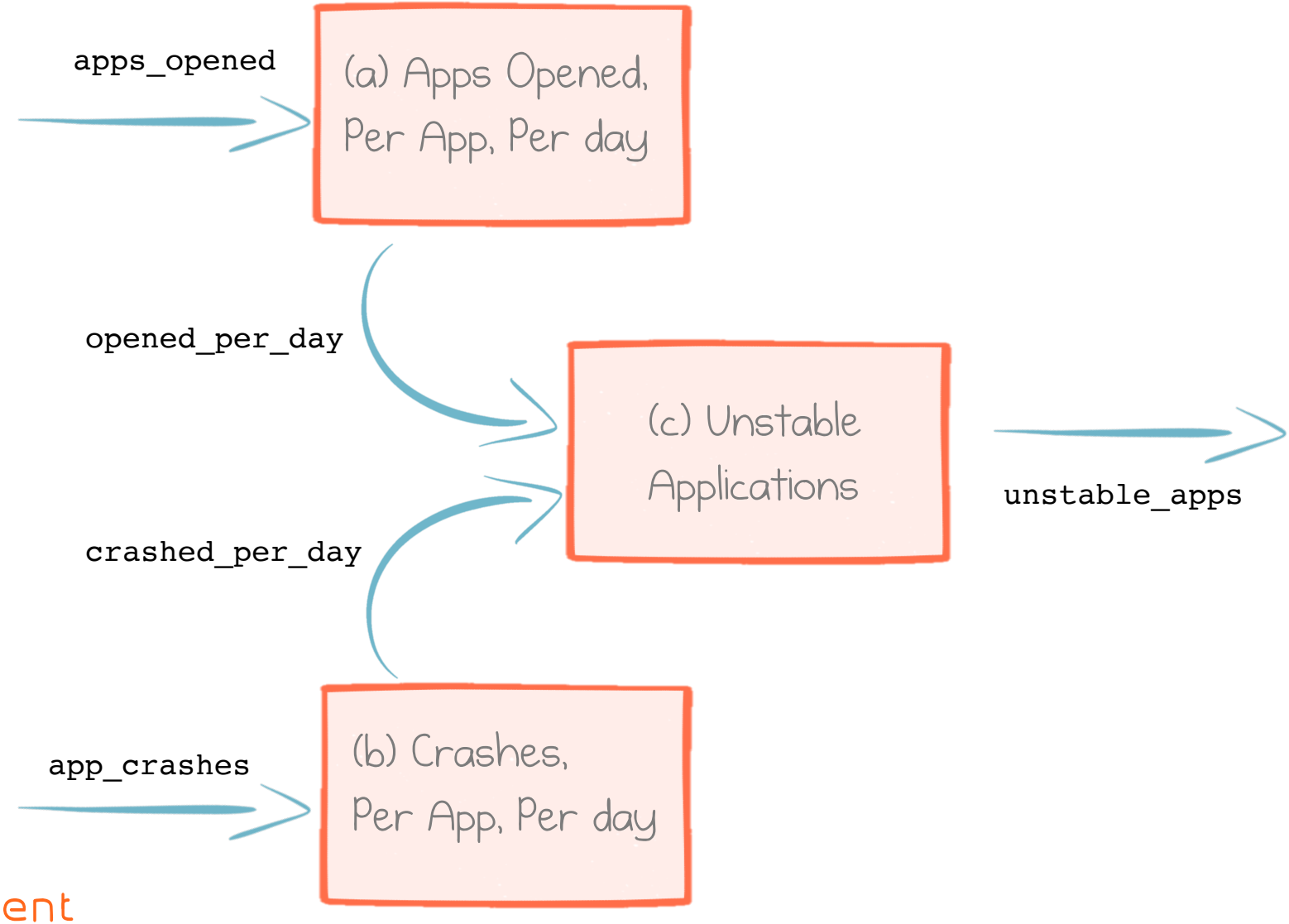

Notification
Data
replication


Notification
Data
replication

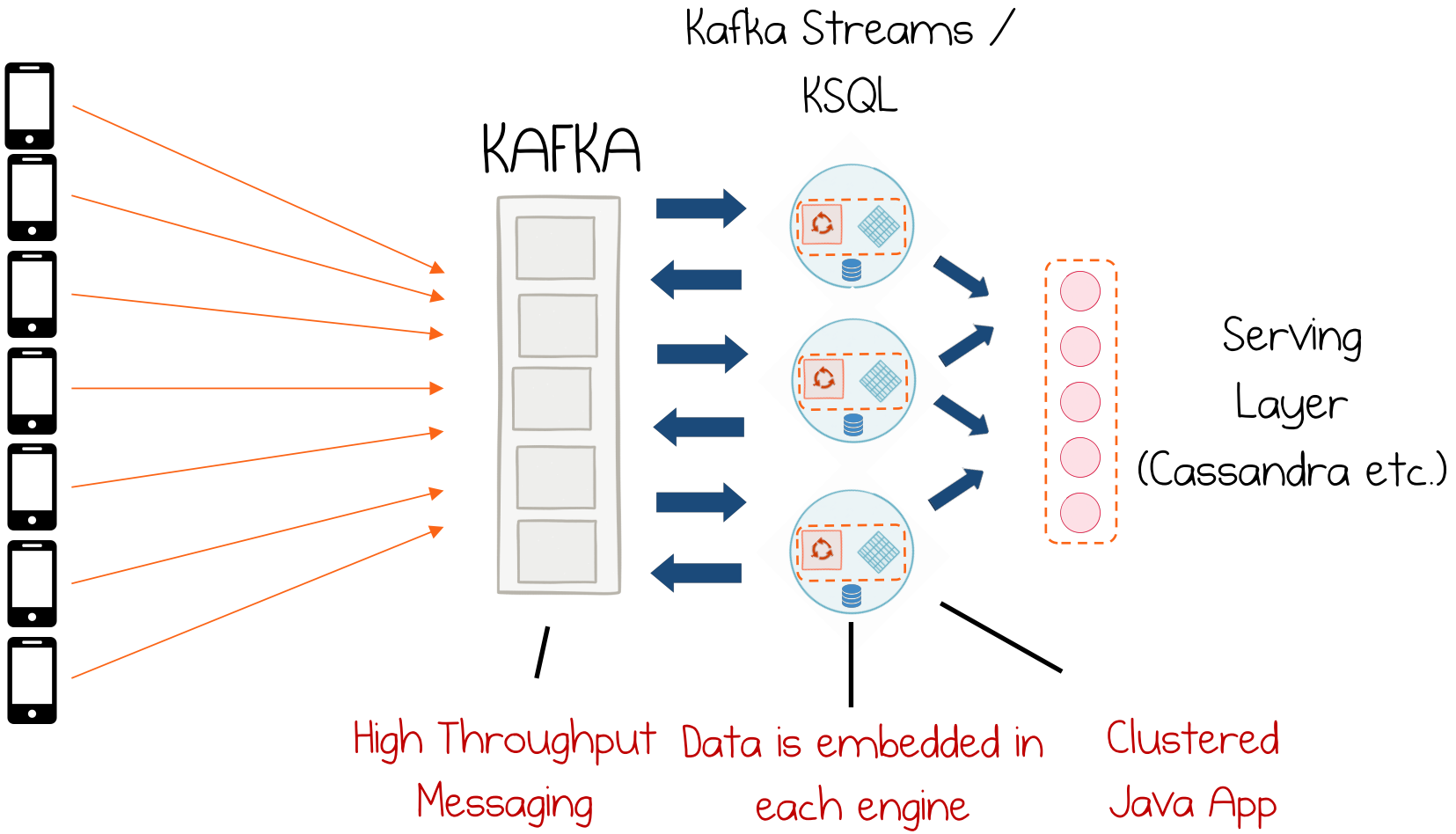
Streaming Platforms



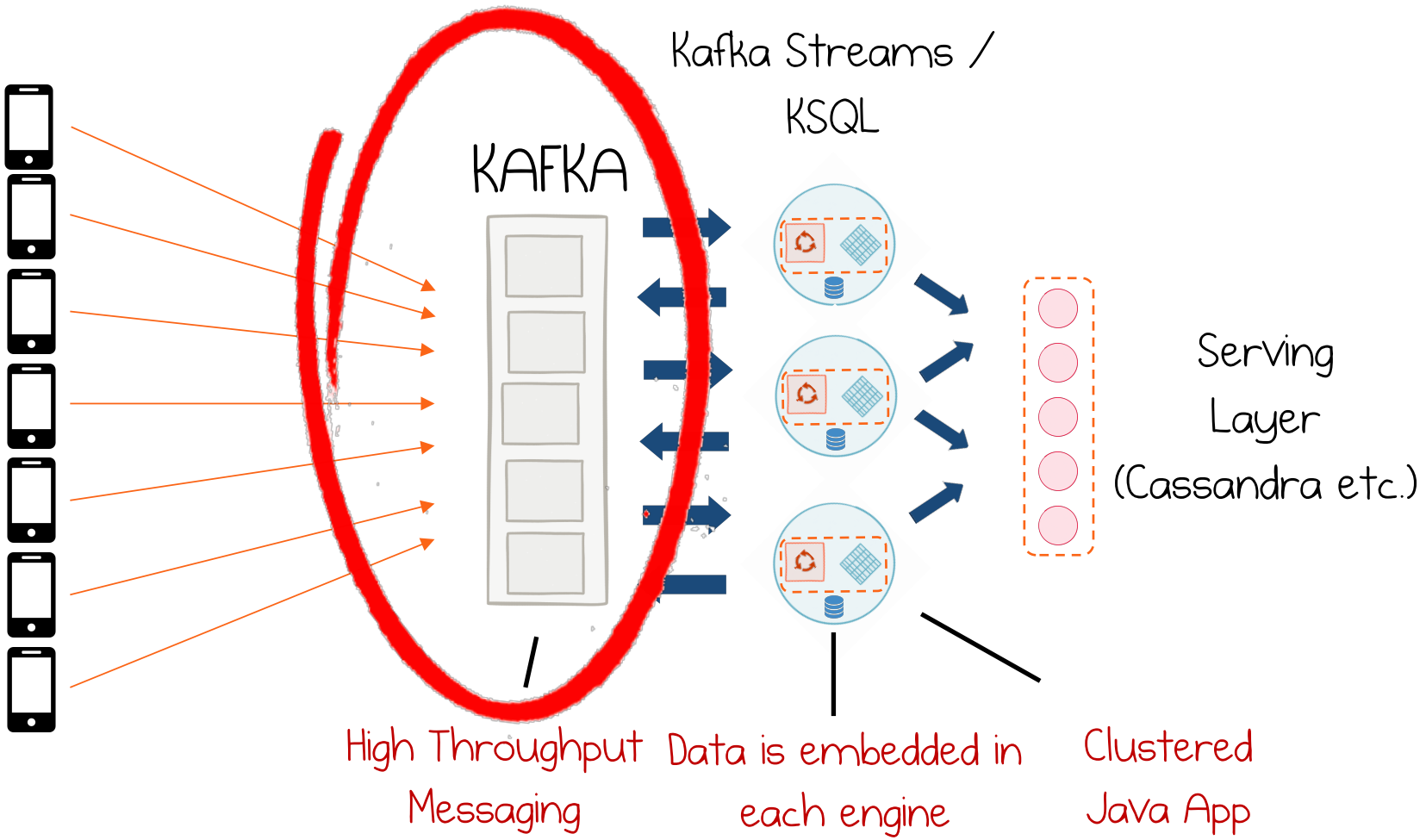
Streaming Pipeline



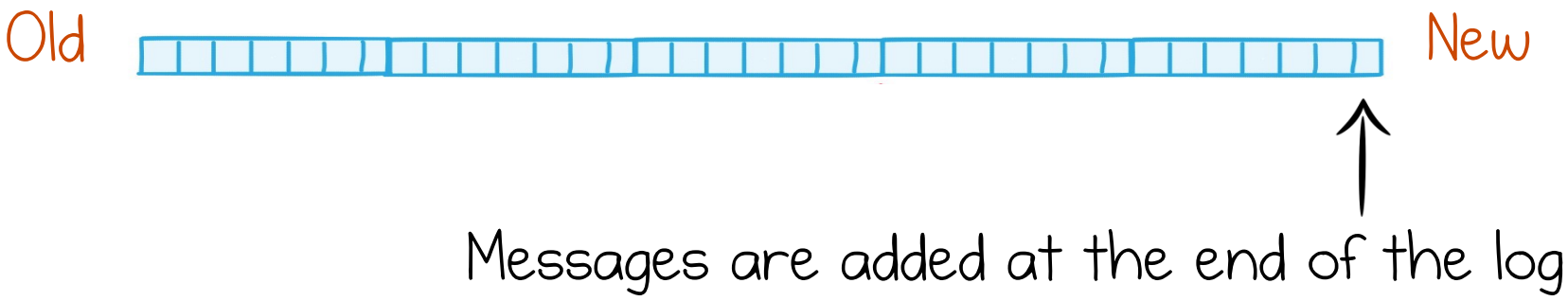
Streaming Platforms



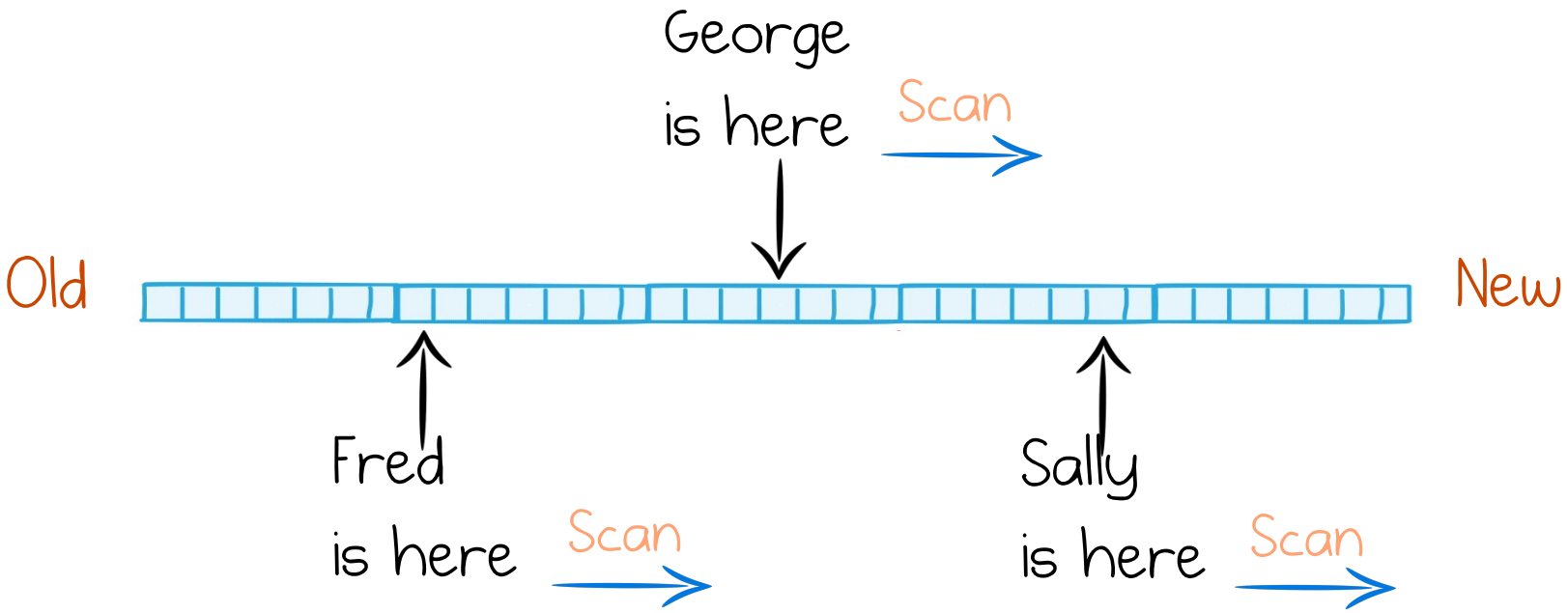
Streaming Platforms



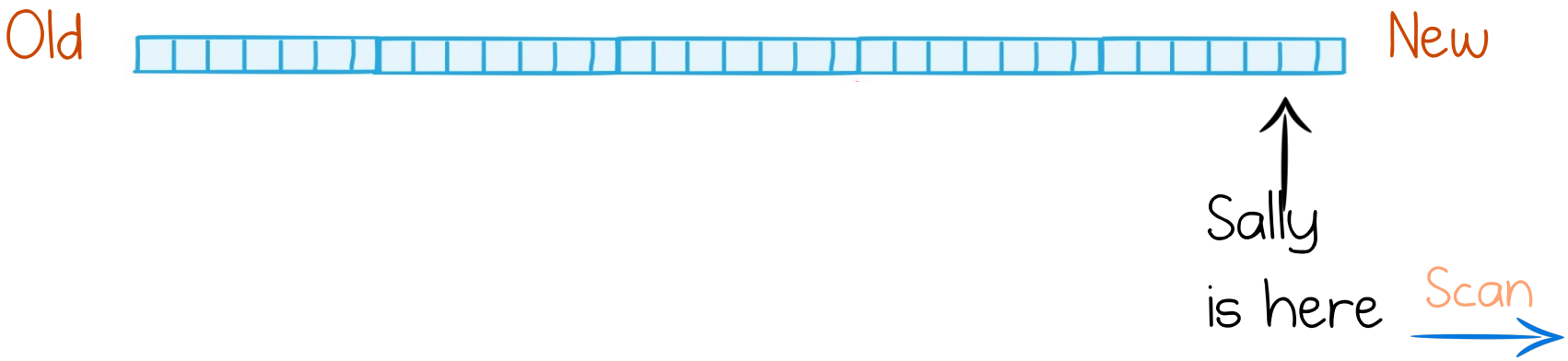
An event log is a simple idea



Readers have a position all of their own

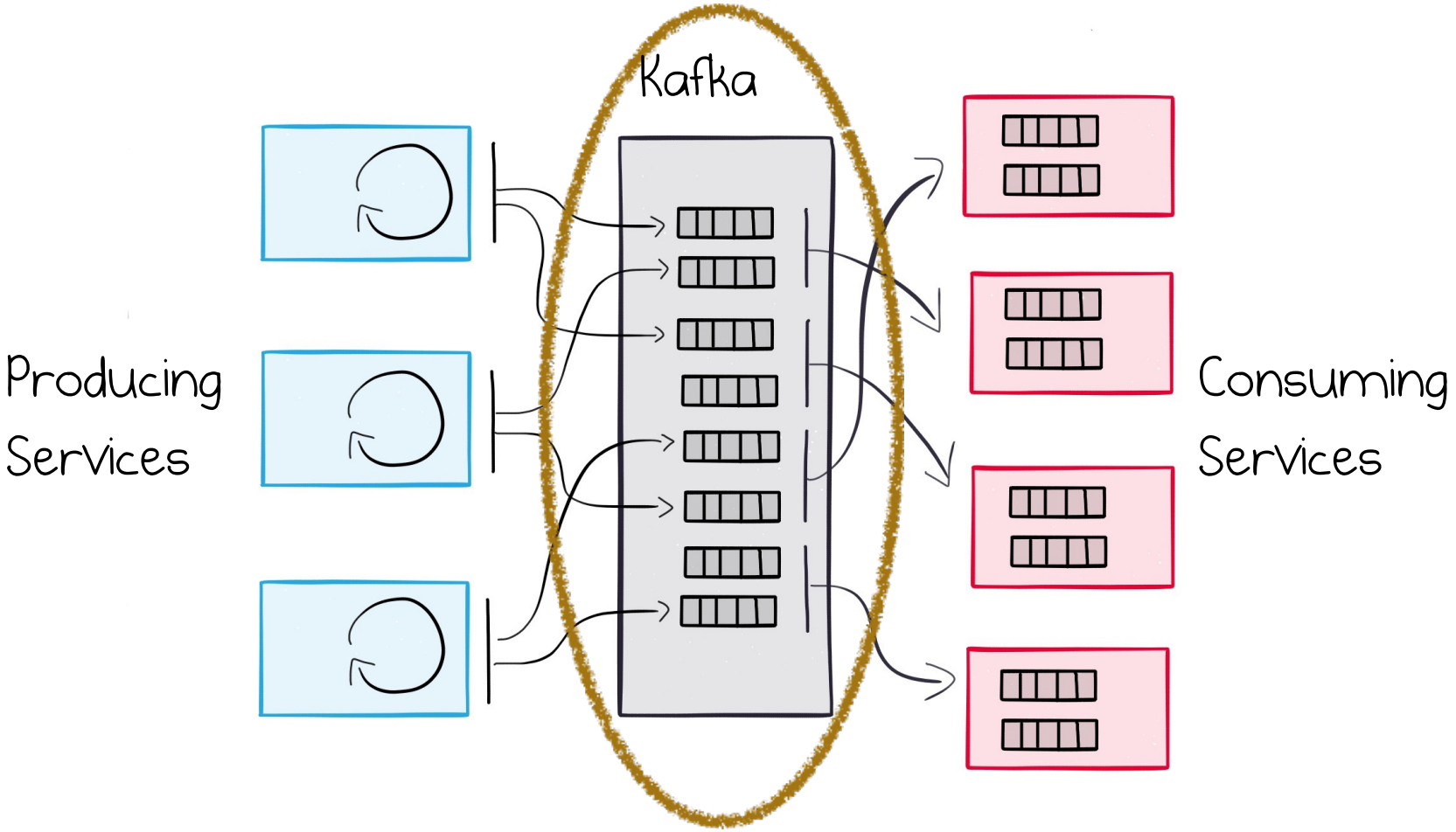


You can rewind and replay, just like Tivo!

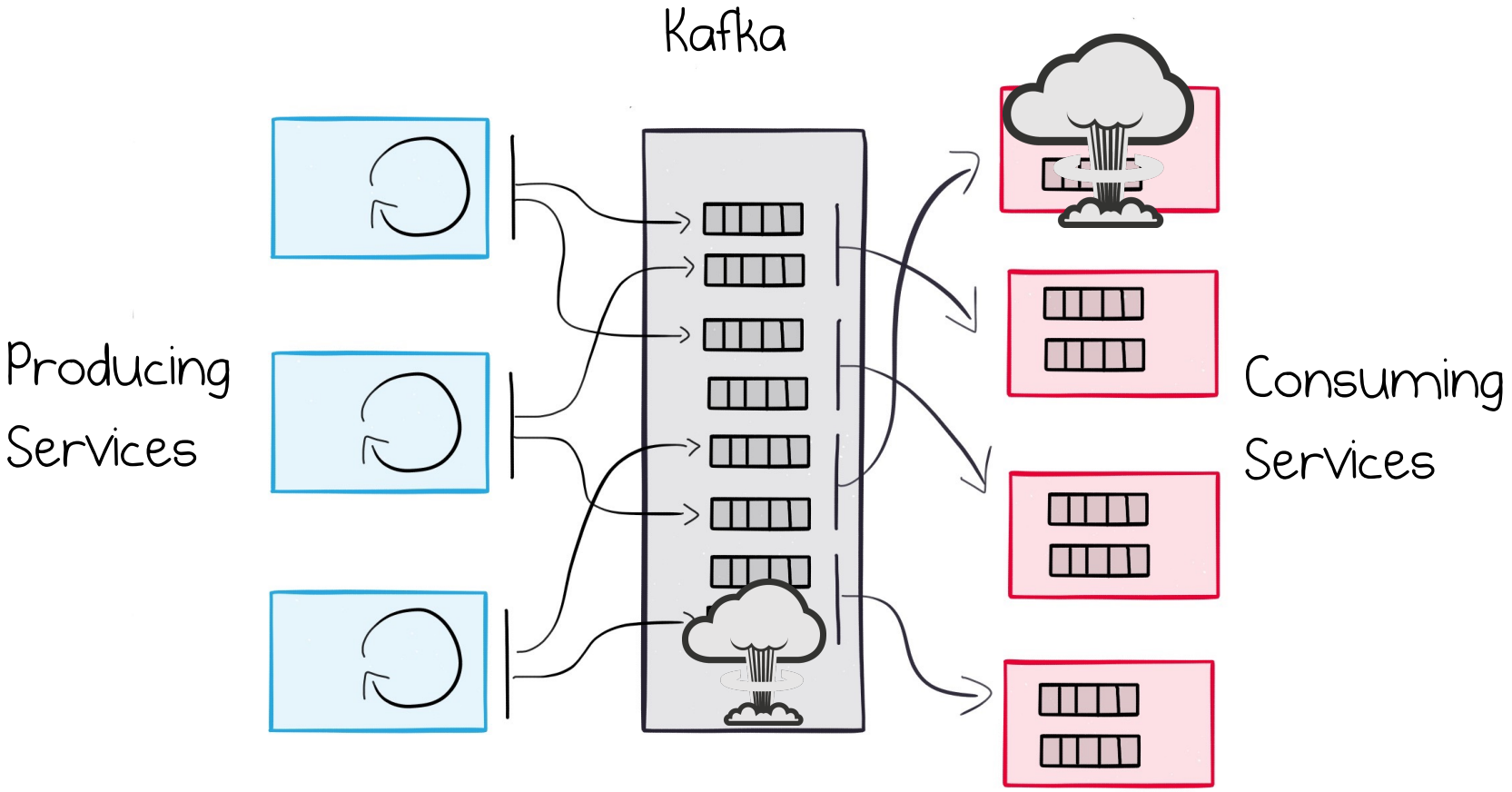


The hard part: Tying it all together!

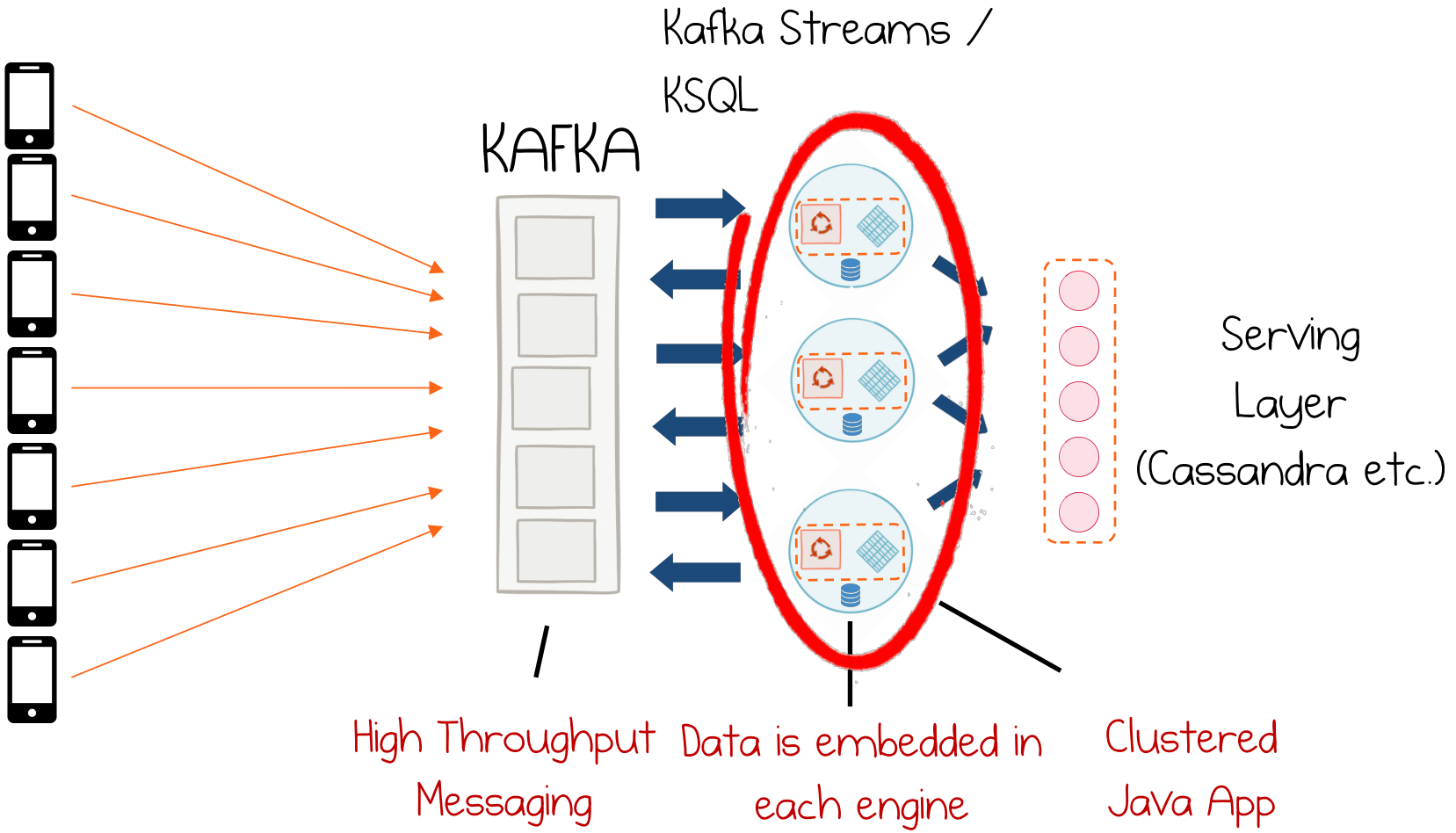
Many "logs" over many machines



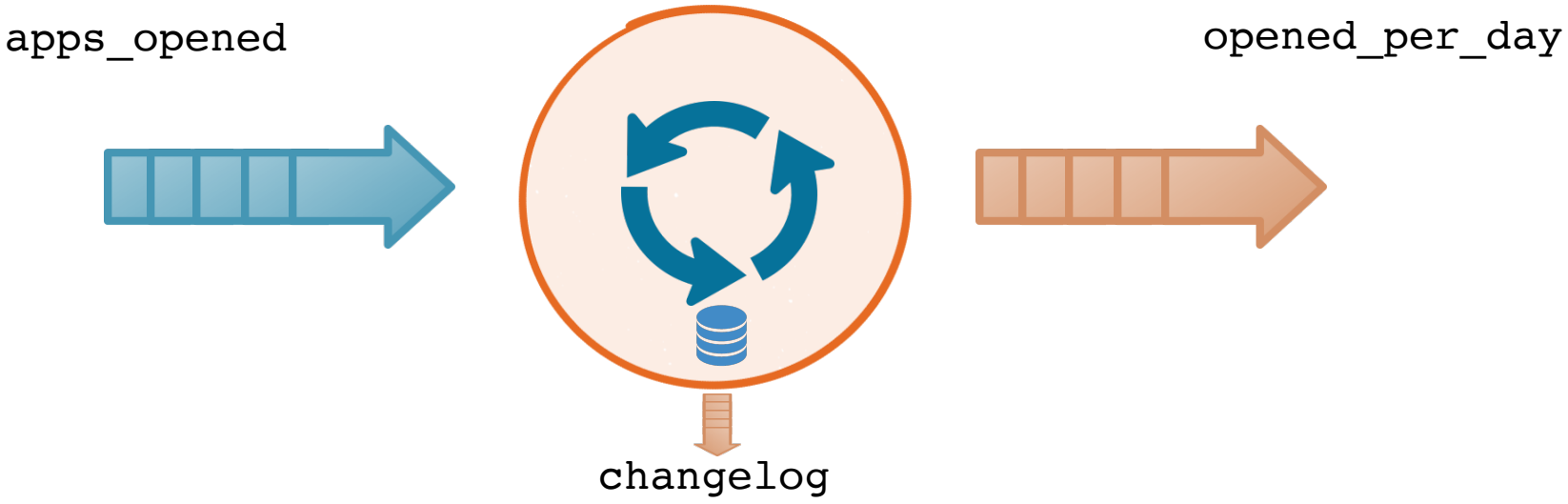
Resistant to Failure

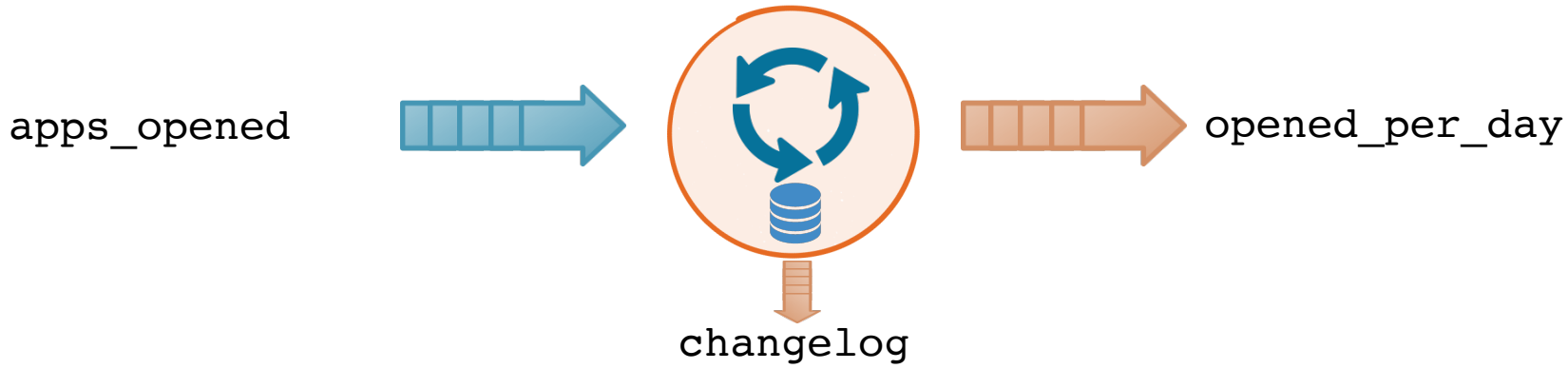


Streaming Platforms

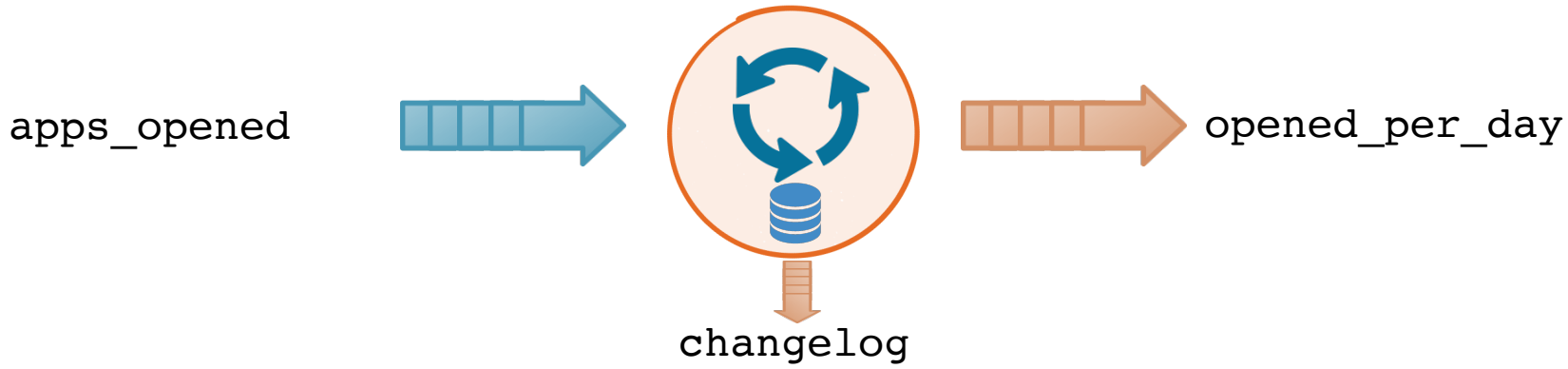


Streaming Example

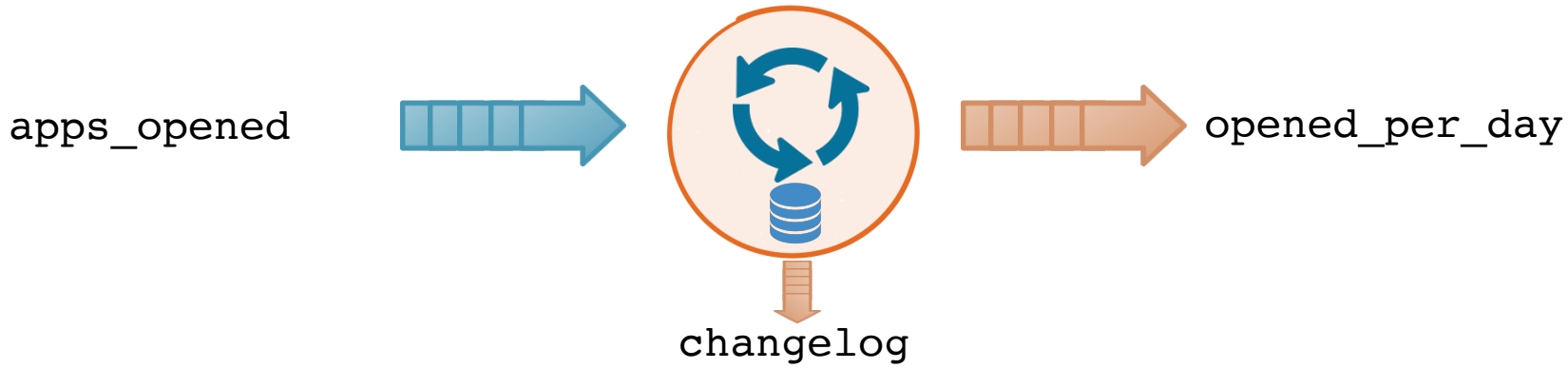




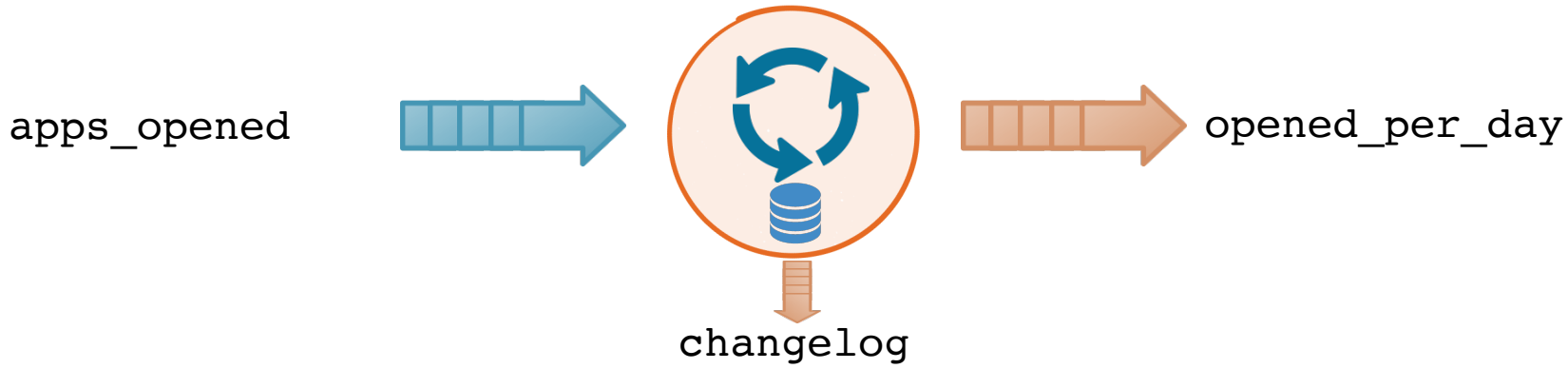

```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



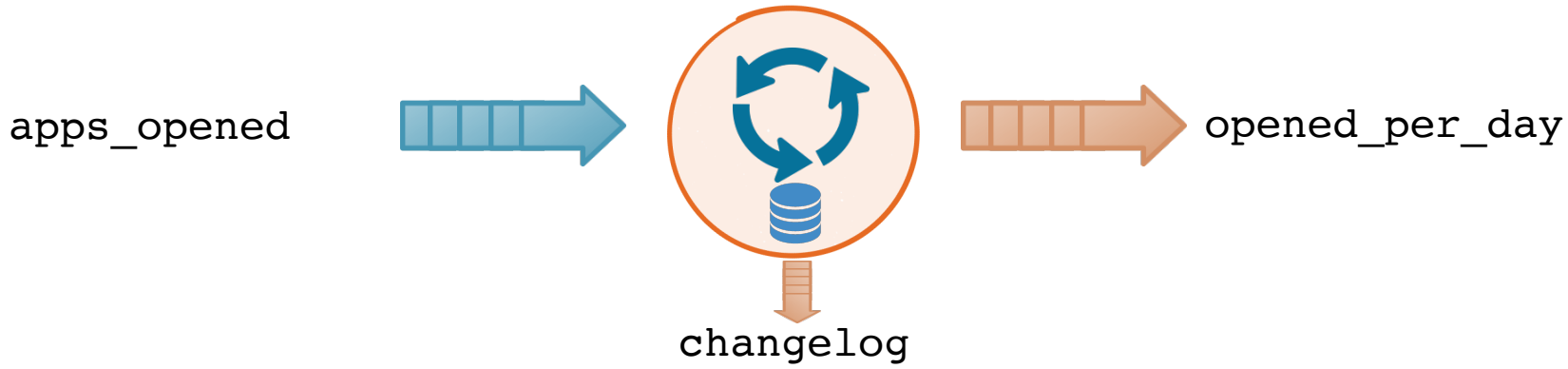

```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



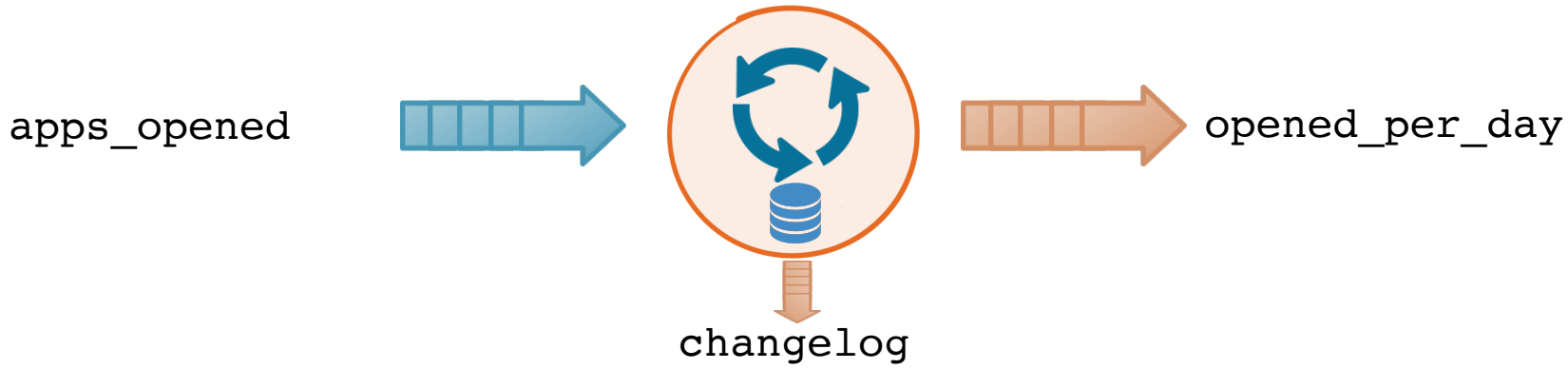

```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```




```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



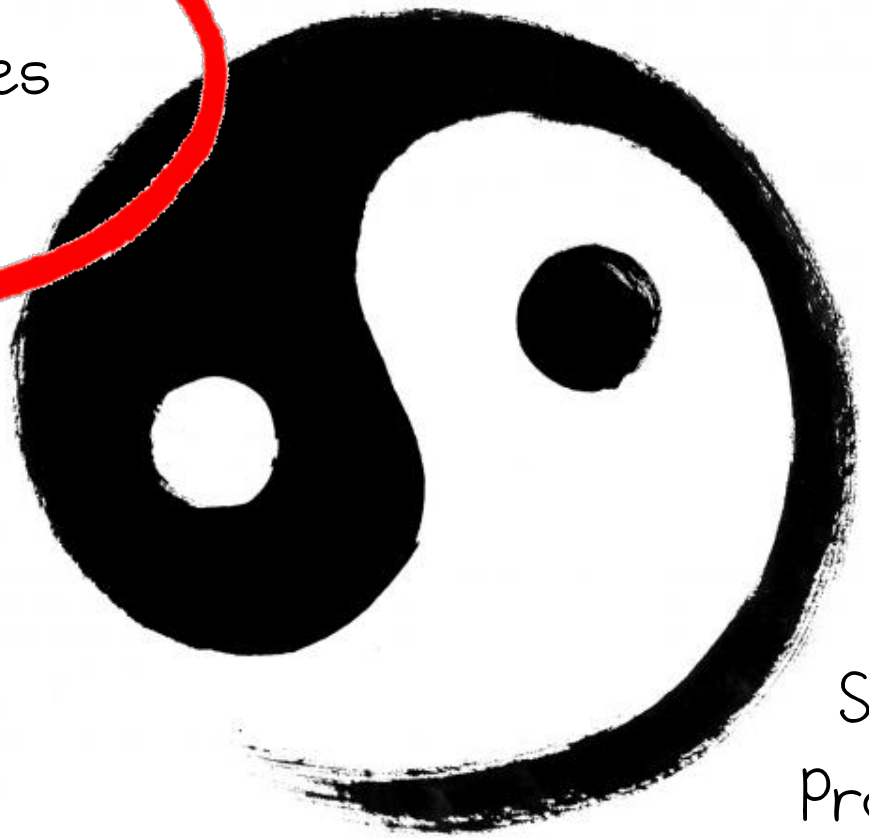

```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```




```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```

Streaming is manipulating events in flight, at scale.

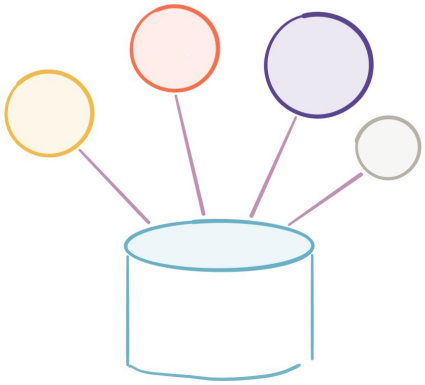
Event Driven Architectures



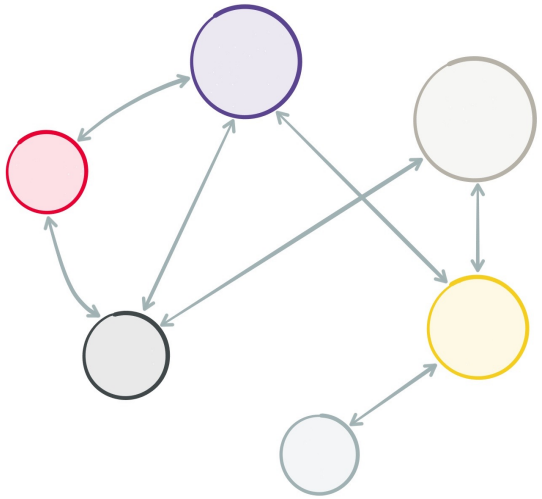
Stream Processing

Increasingly we build ecosystems

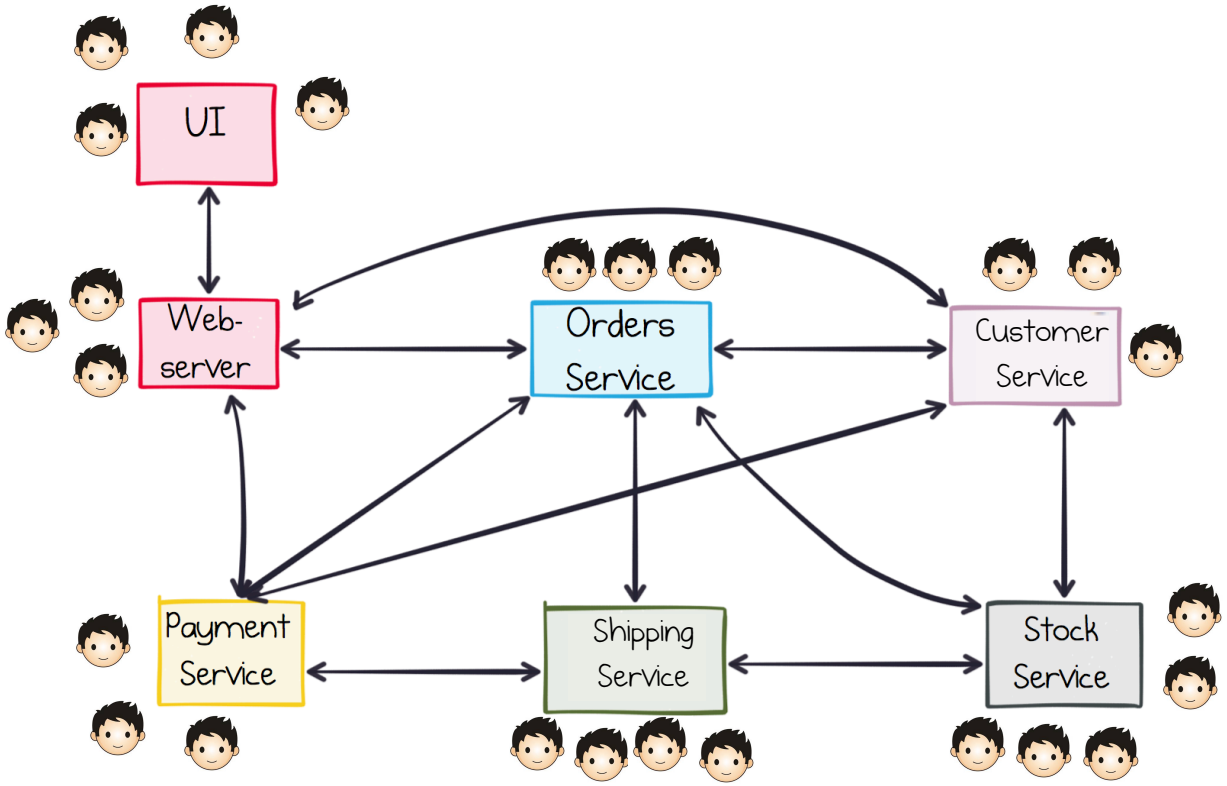
App



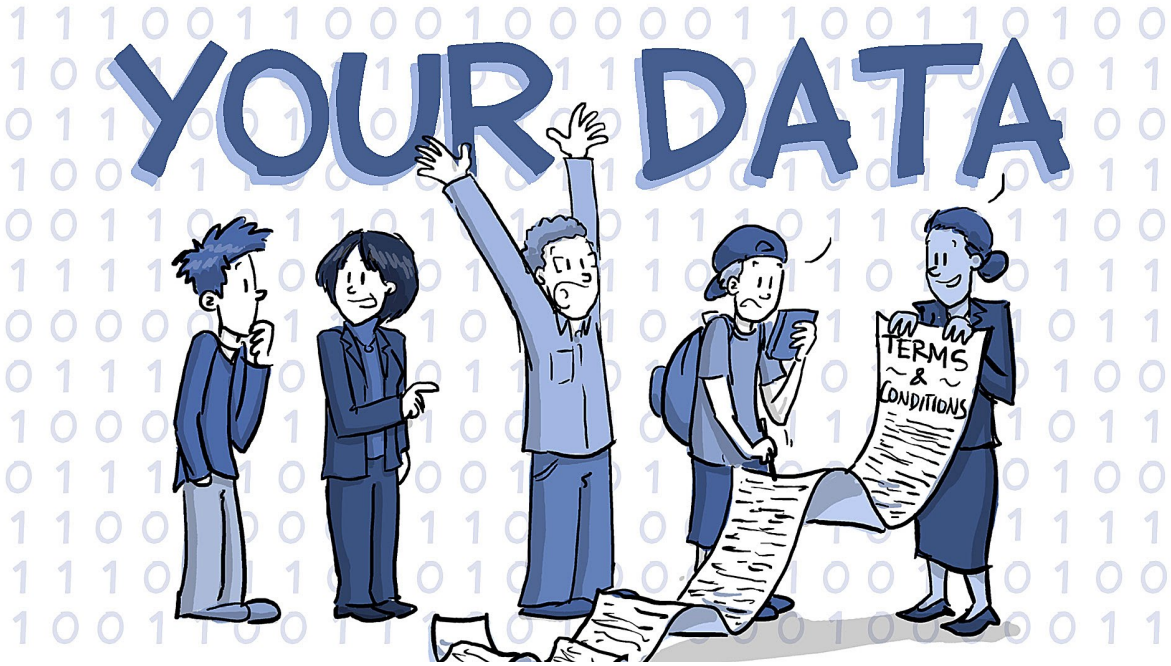
Ecosystems



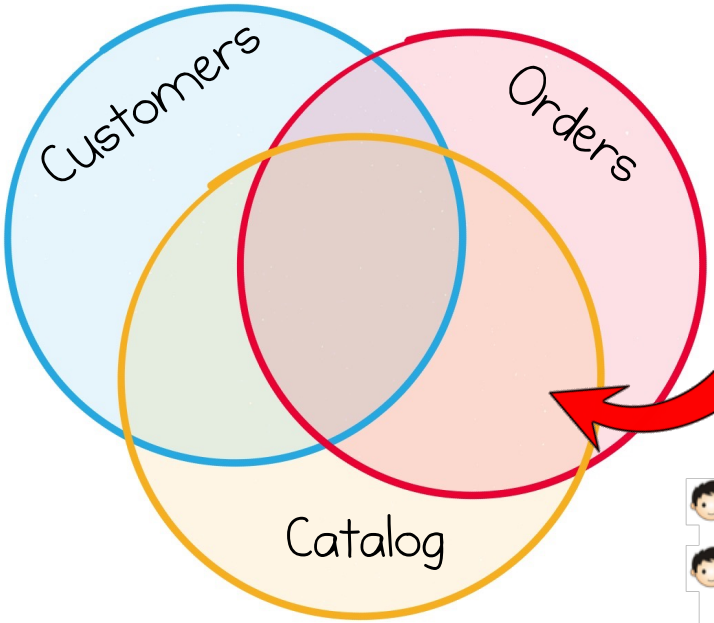
SOA / Microservices / EDA



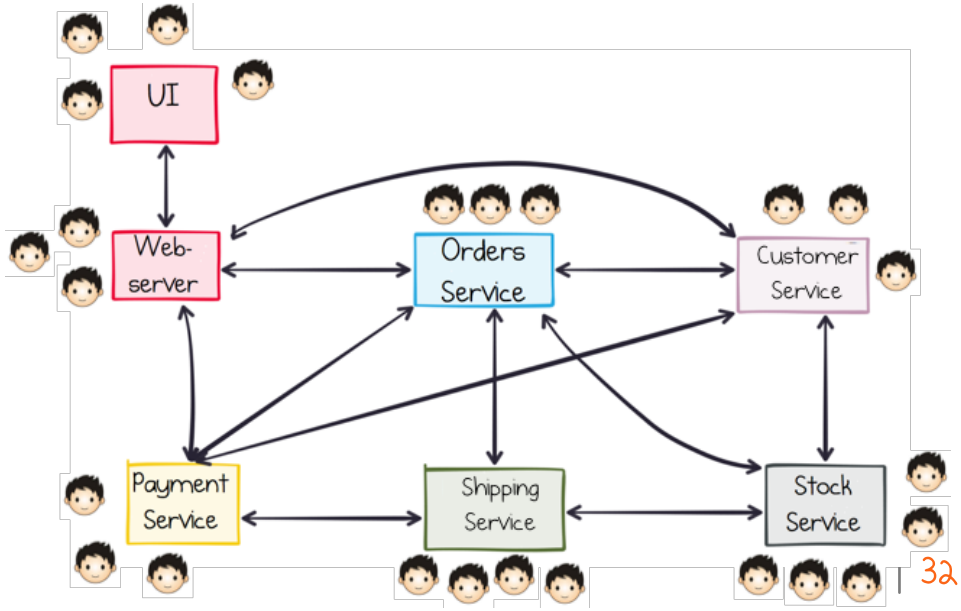
The Problem is DATA



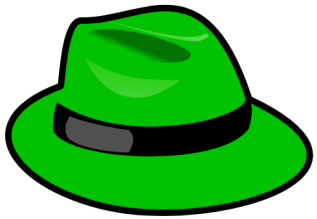
Most services share the same core facts.



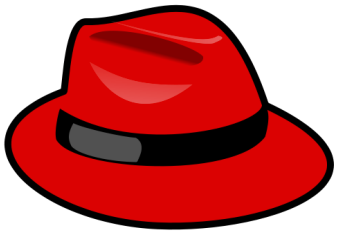
Most services live in here



Events have two hats

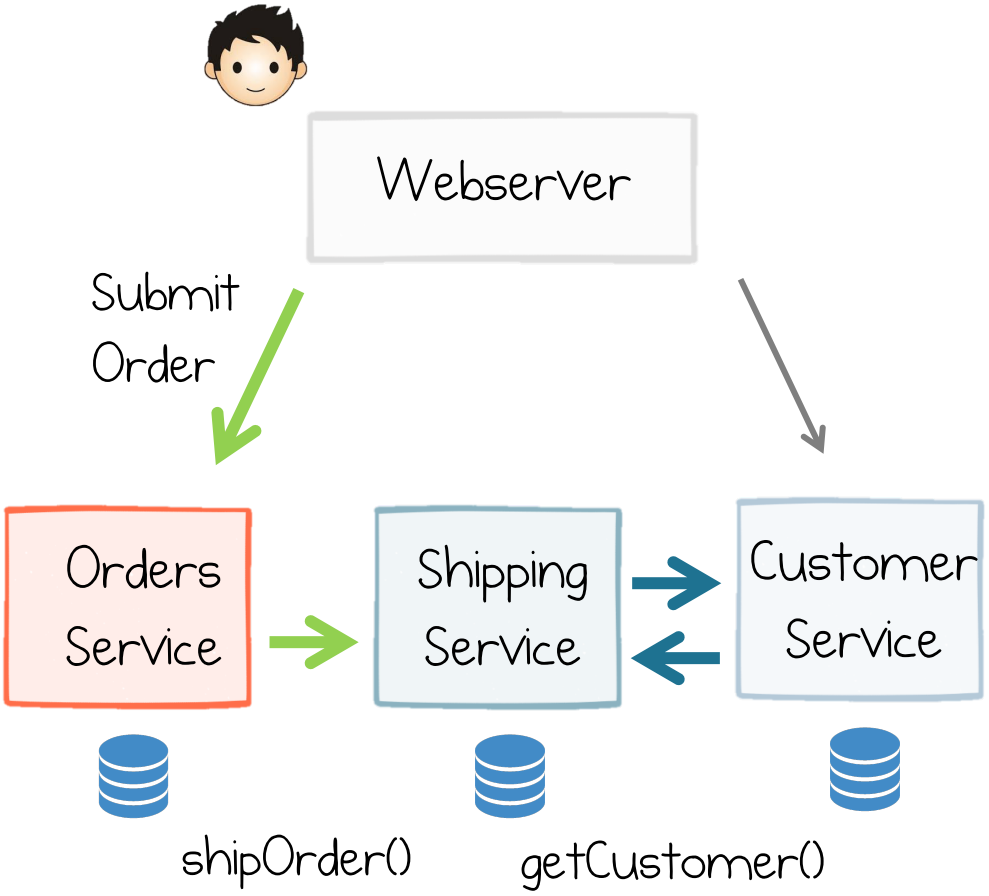


Notification

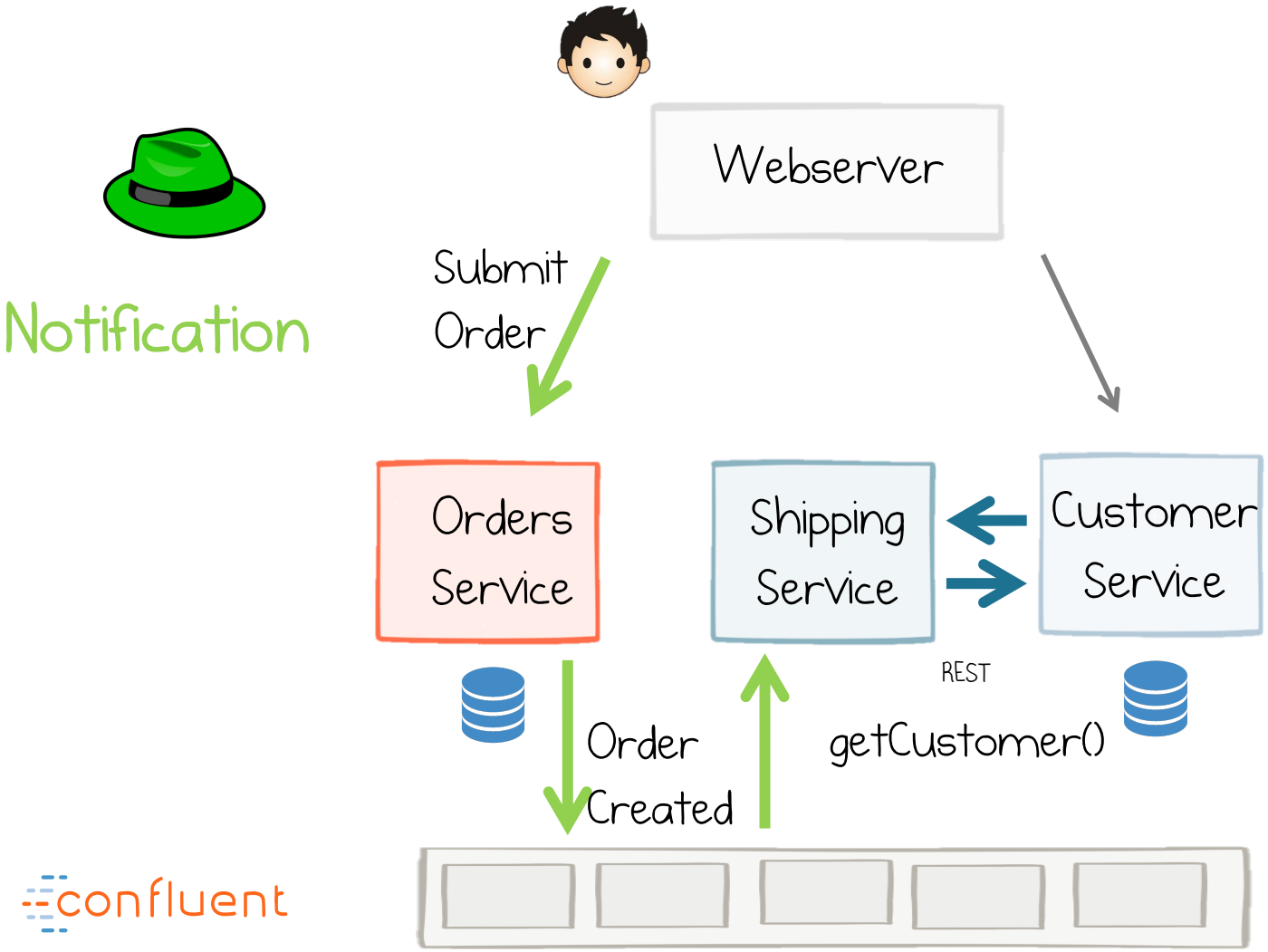


Data replication

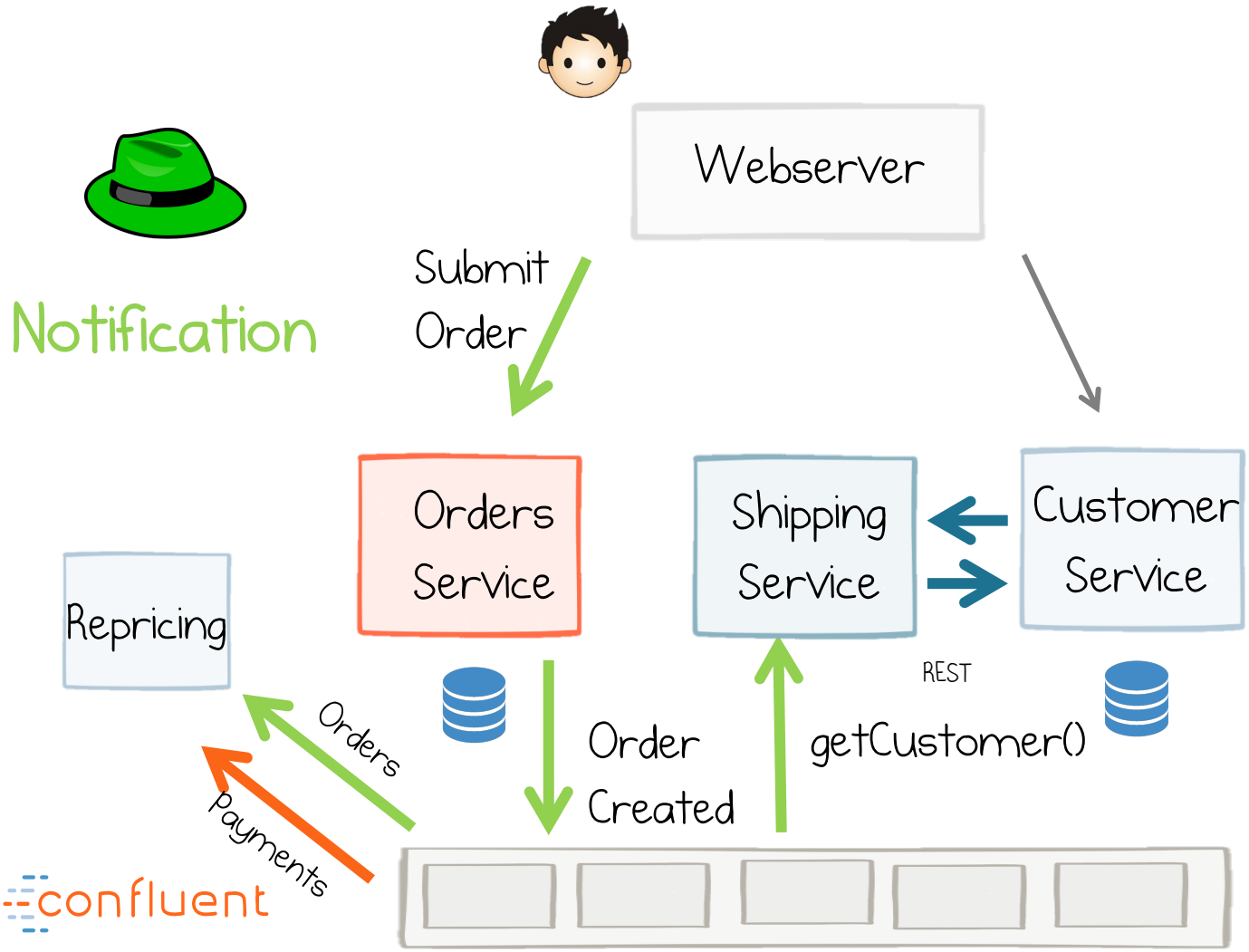
Buying an iPad (with REST/RPC)



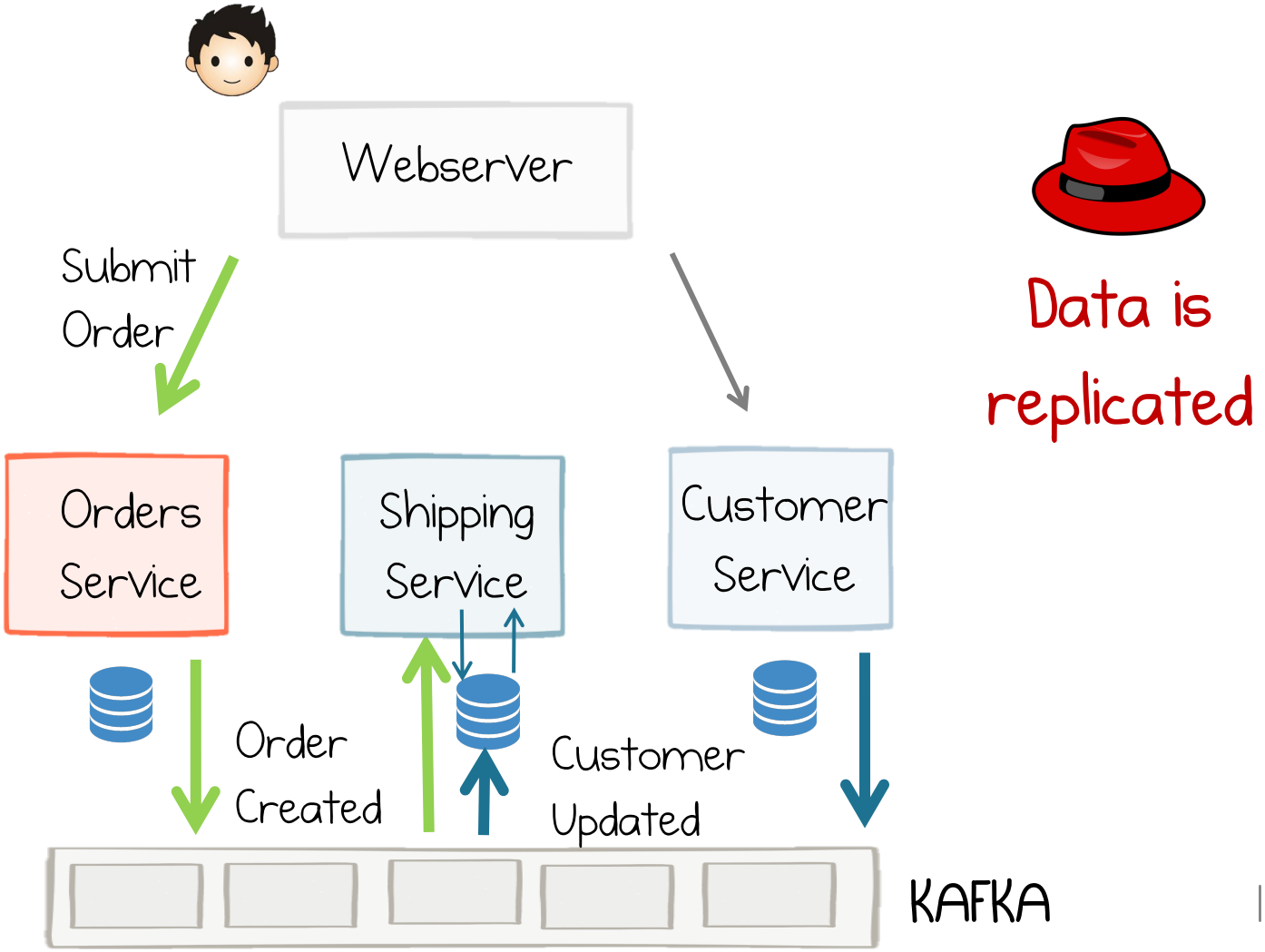
Events for Notification Only



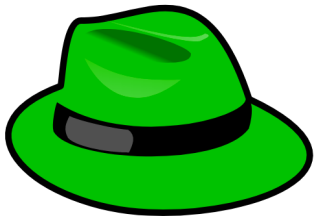
Pluggability



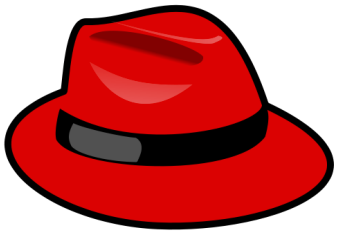
Events for Data Locality



Events have two hats



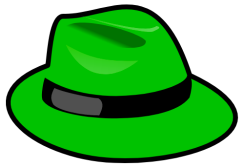
Notification



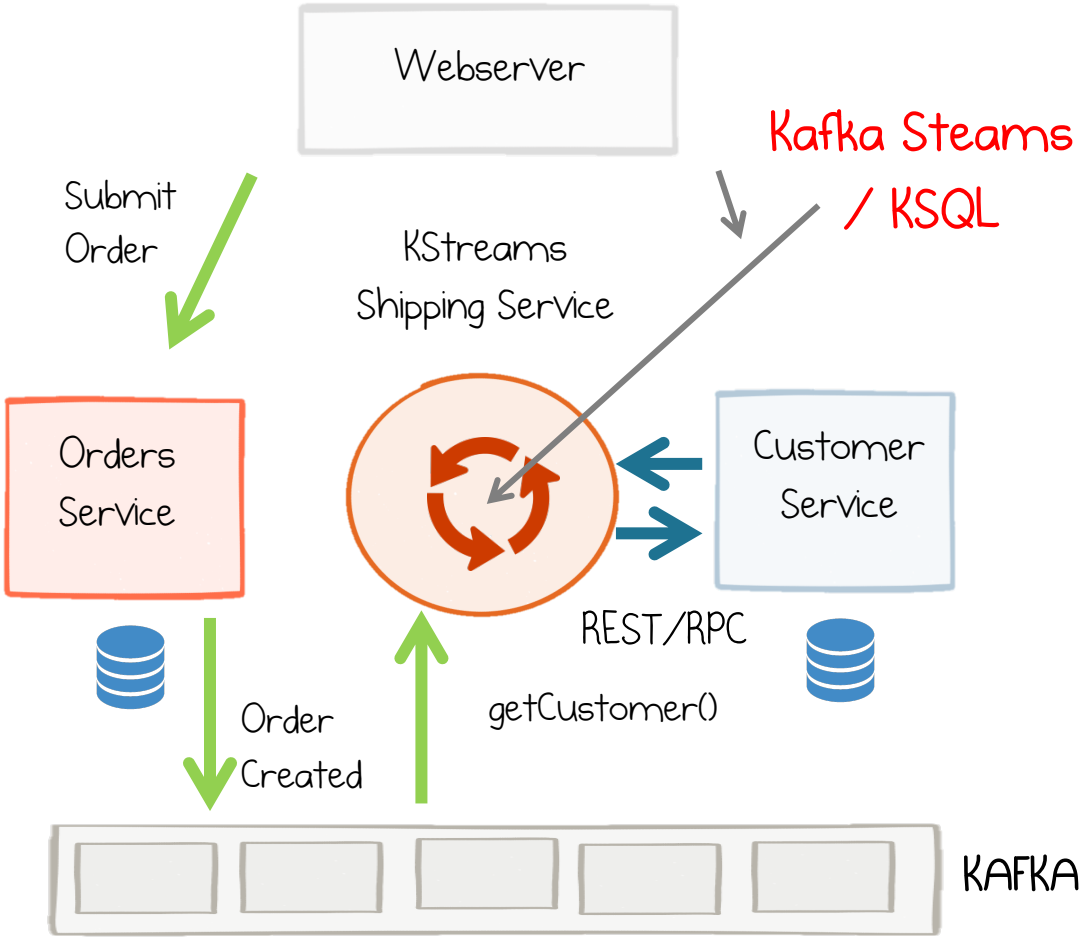
Data replication

Stateless / Stateful Stream Processing Relates to these hats

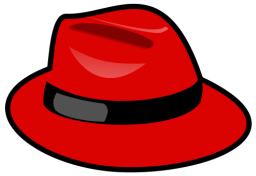
Stateless Stream Processing



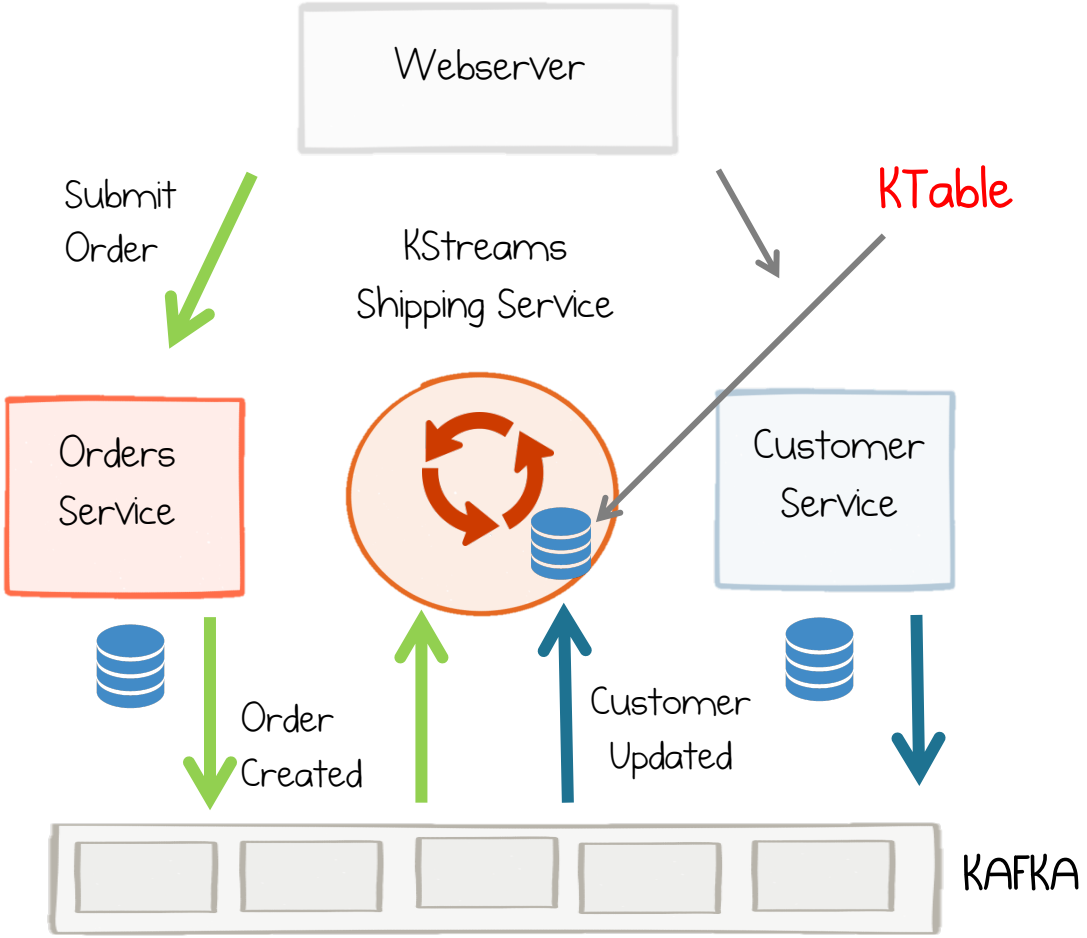
Notification



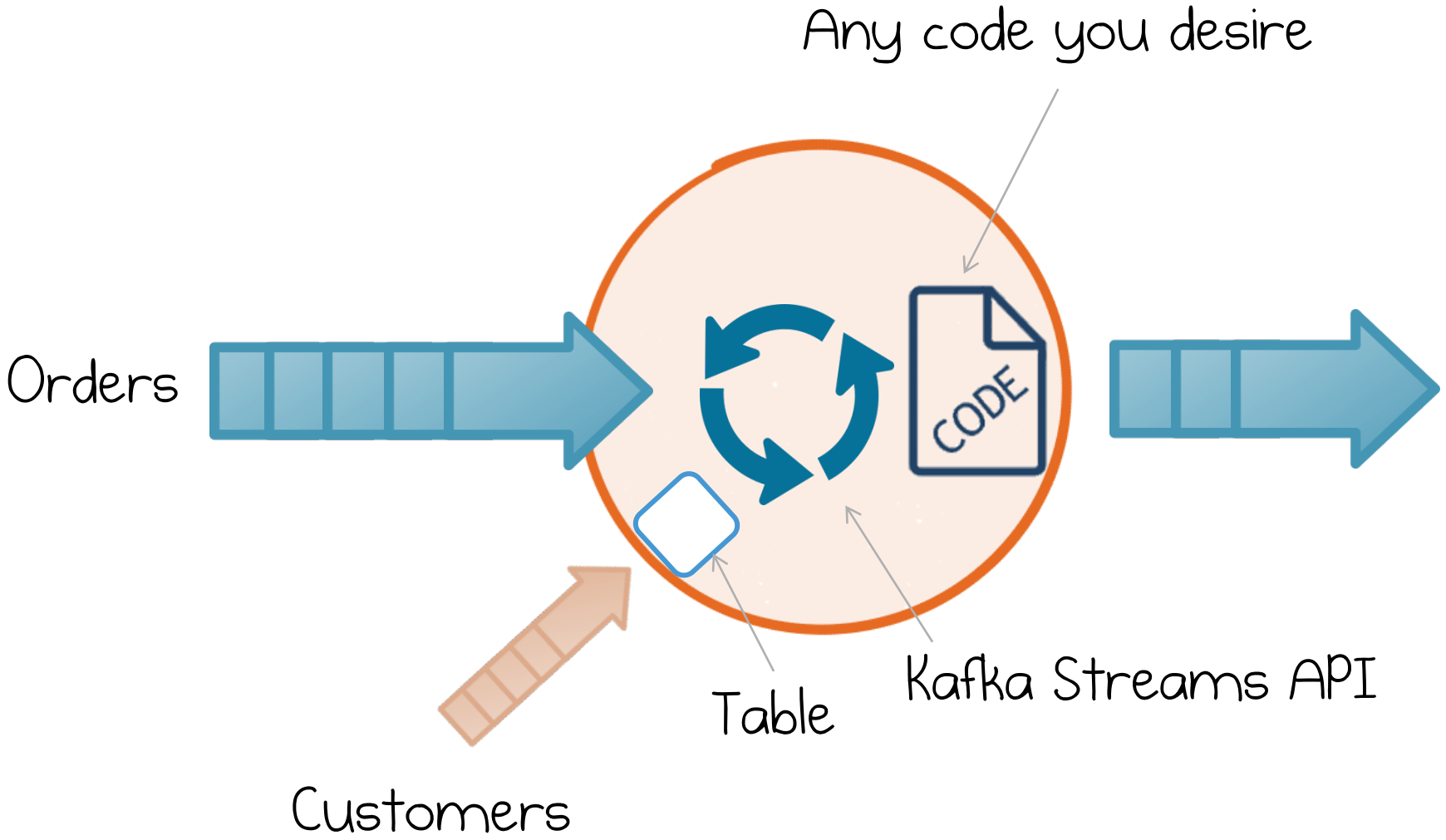
Stateful Stream Processing



Data replication



KSQL ~ KStreams

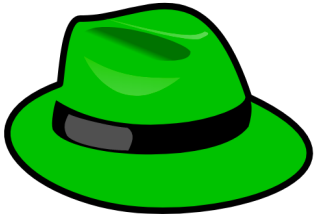

Any code you desire
Orders
CODE
Customers
Table
Kafka Streams API

It's just java

```
builder.stream("Orders")
.join("Customers",…)
.transform((key, value) ->
   {//Any code your heart desires!}
)
.to("Shipments");
```

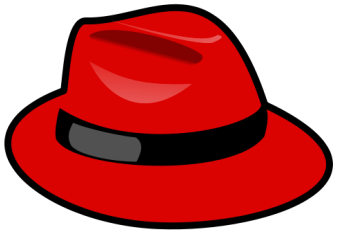
Streaming can be stateful or stateless

1. Joining & Operating on Streams



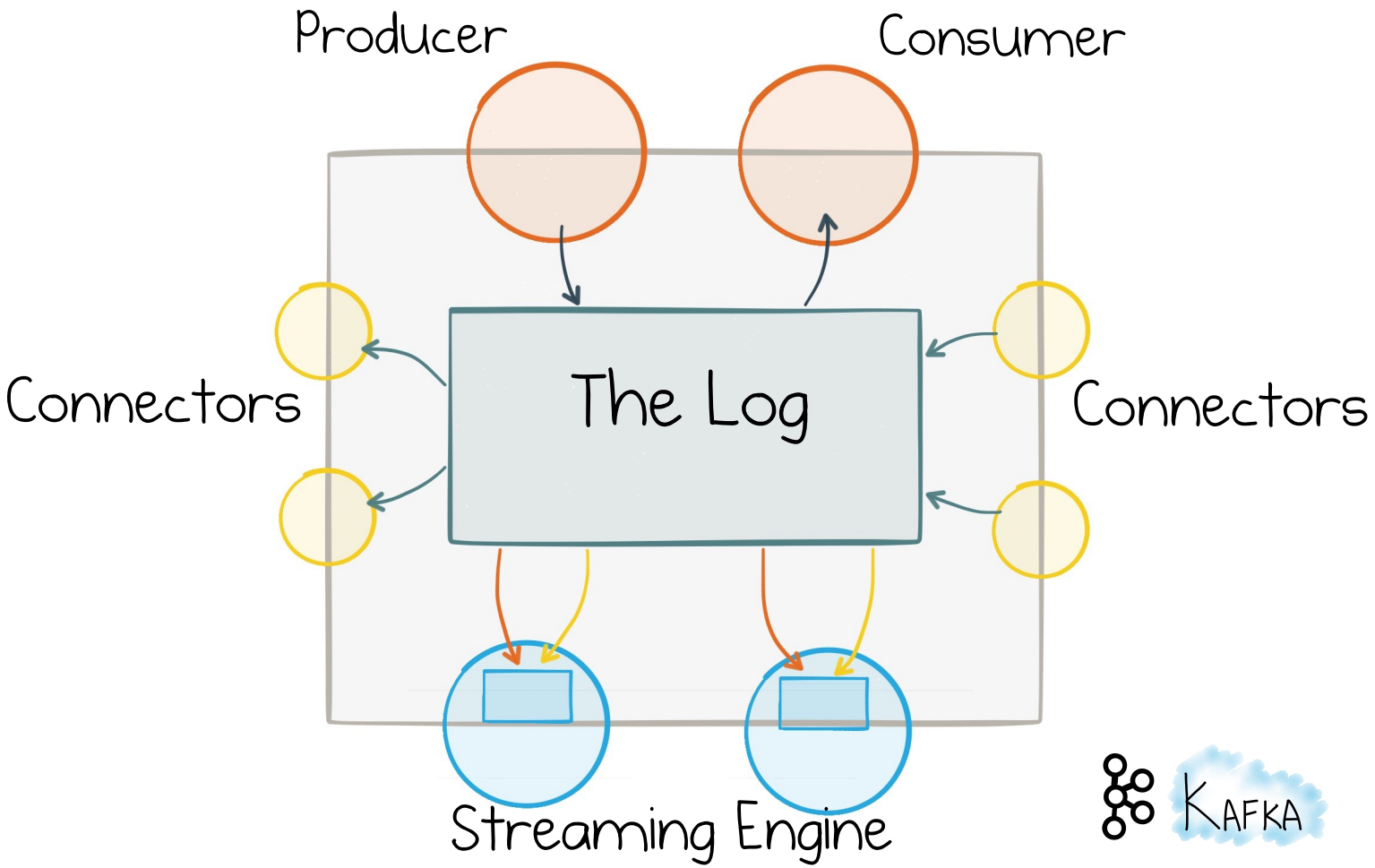
On Notification

2. Joining & Operating on Materialized Tables



Data Replication

Streaming Platform

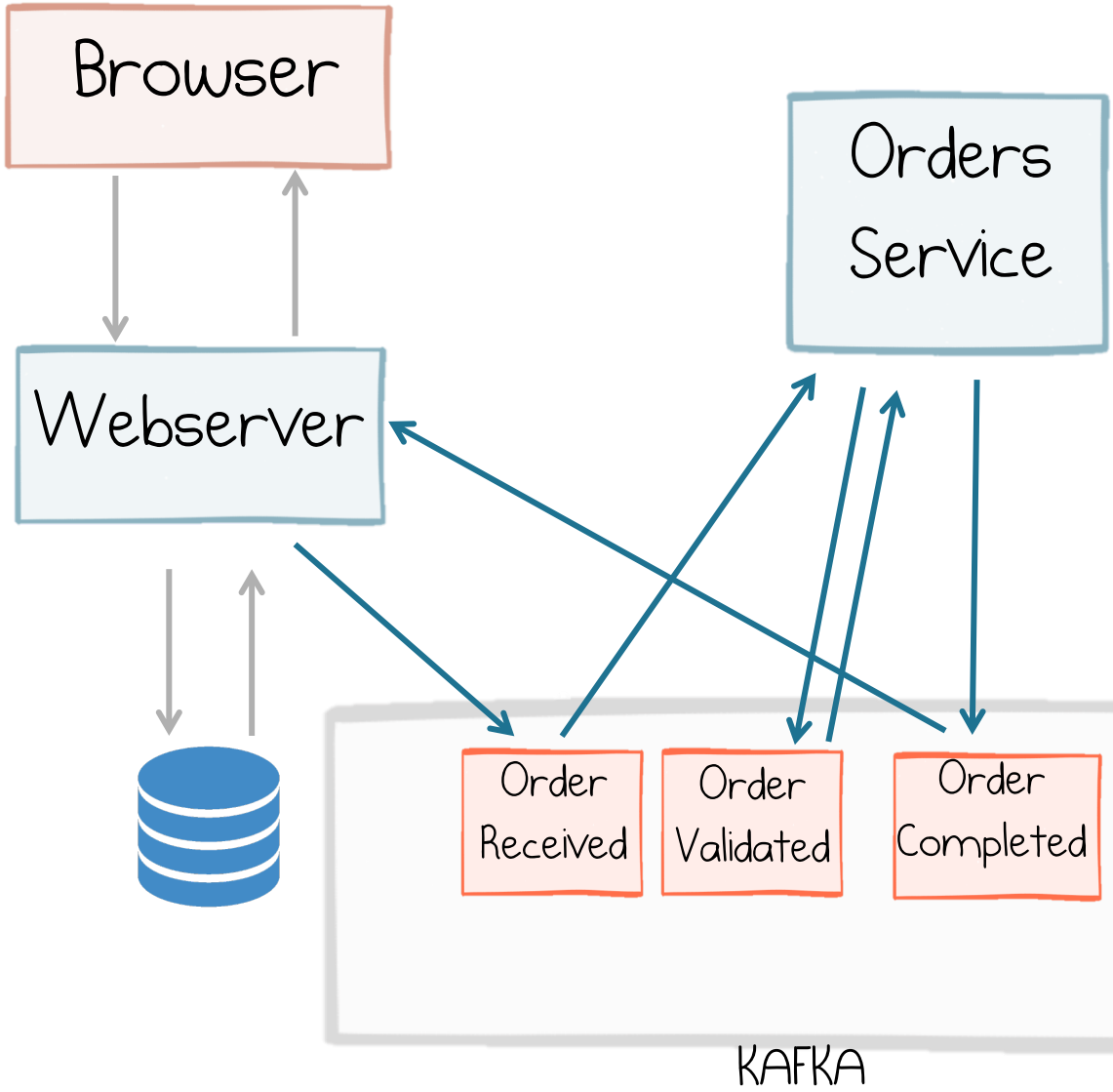


Event Driven Example

1. Use events to decouple and to collaborate

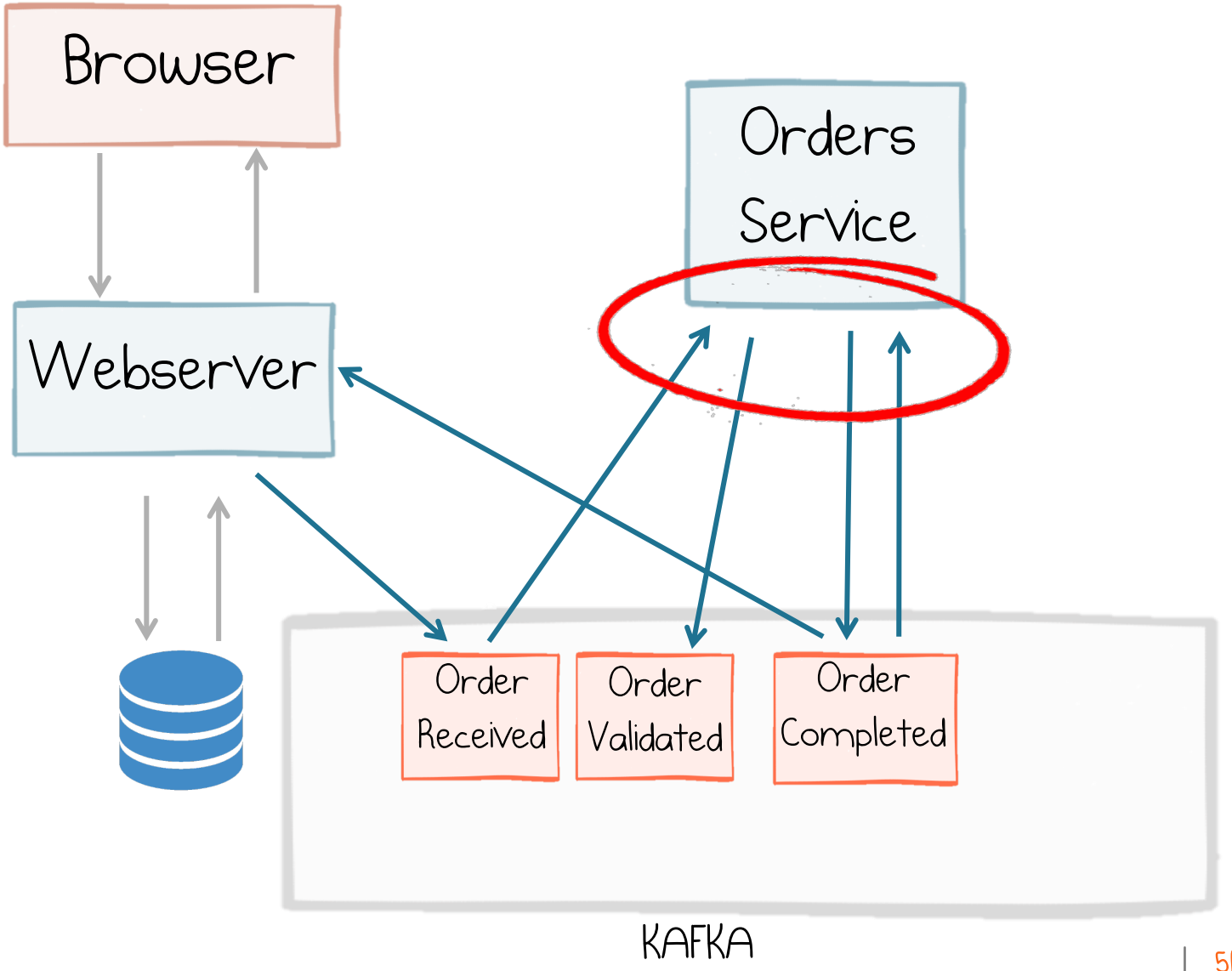
Event Collaboration

Notification

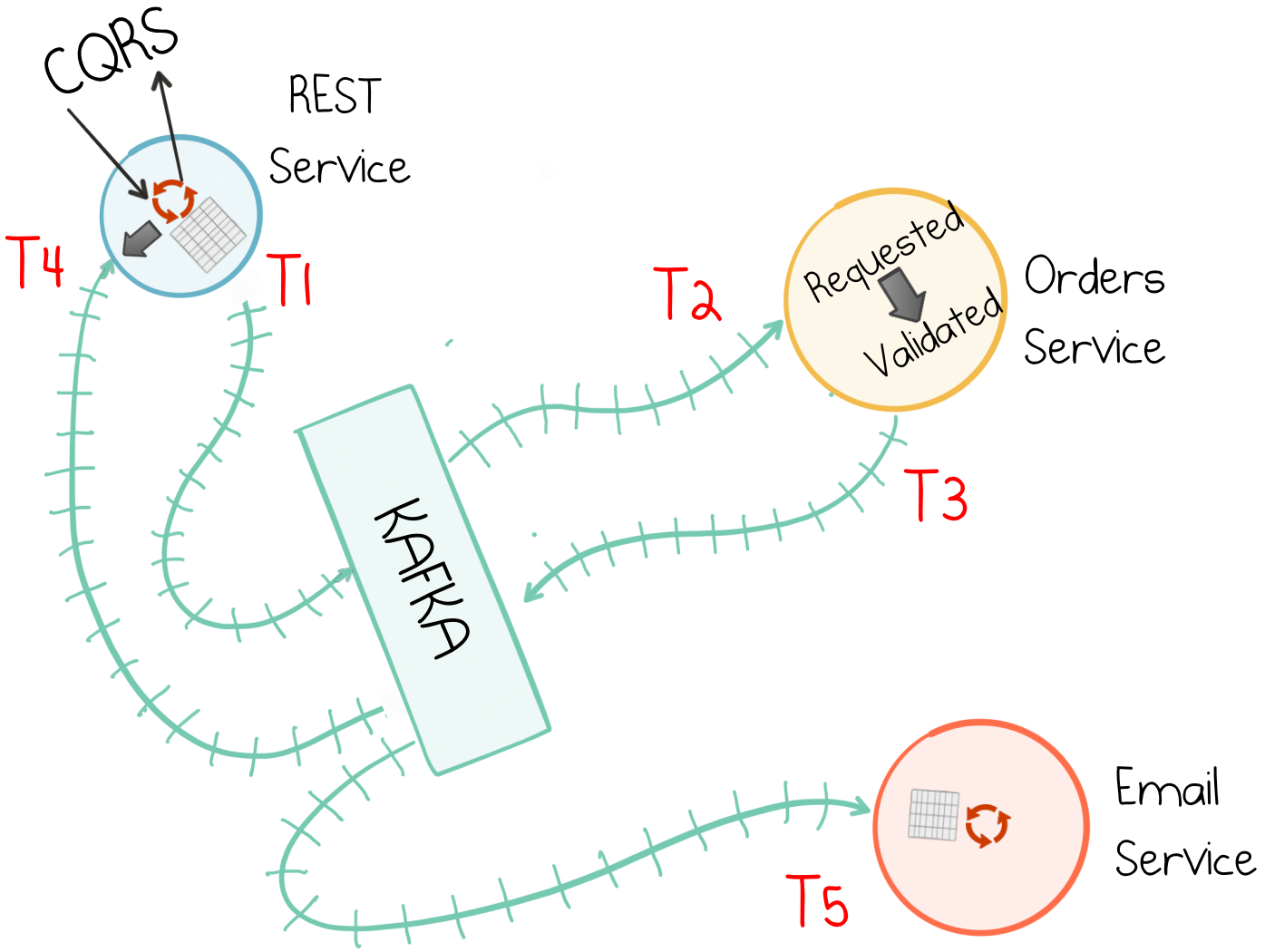


2. Use the Single Writer Principal

State changes to a topic owned by one service

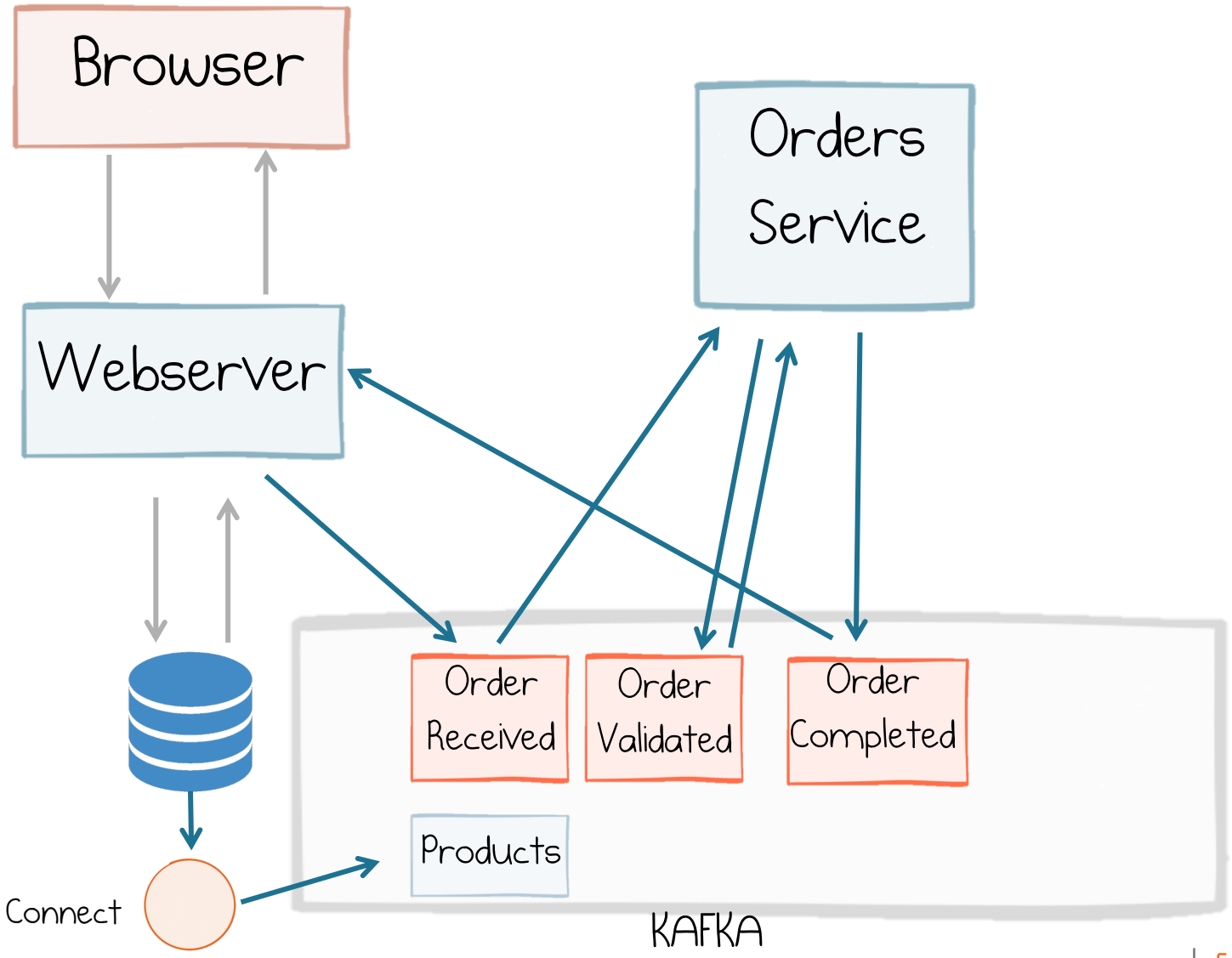


Local consistency points in the absence of Global Consistency



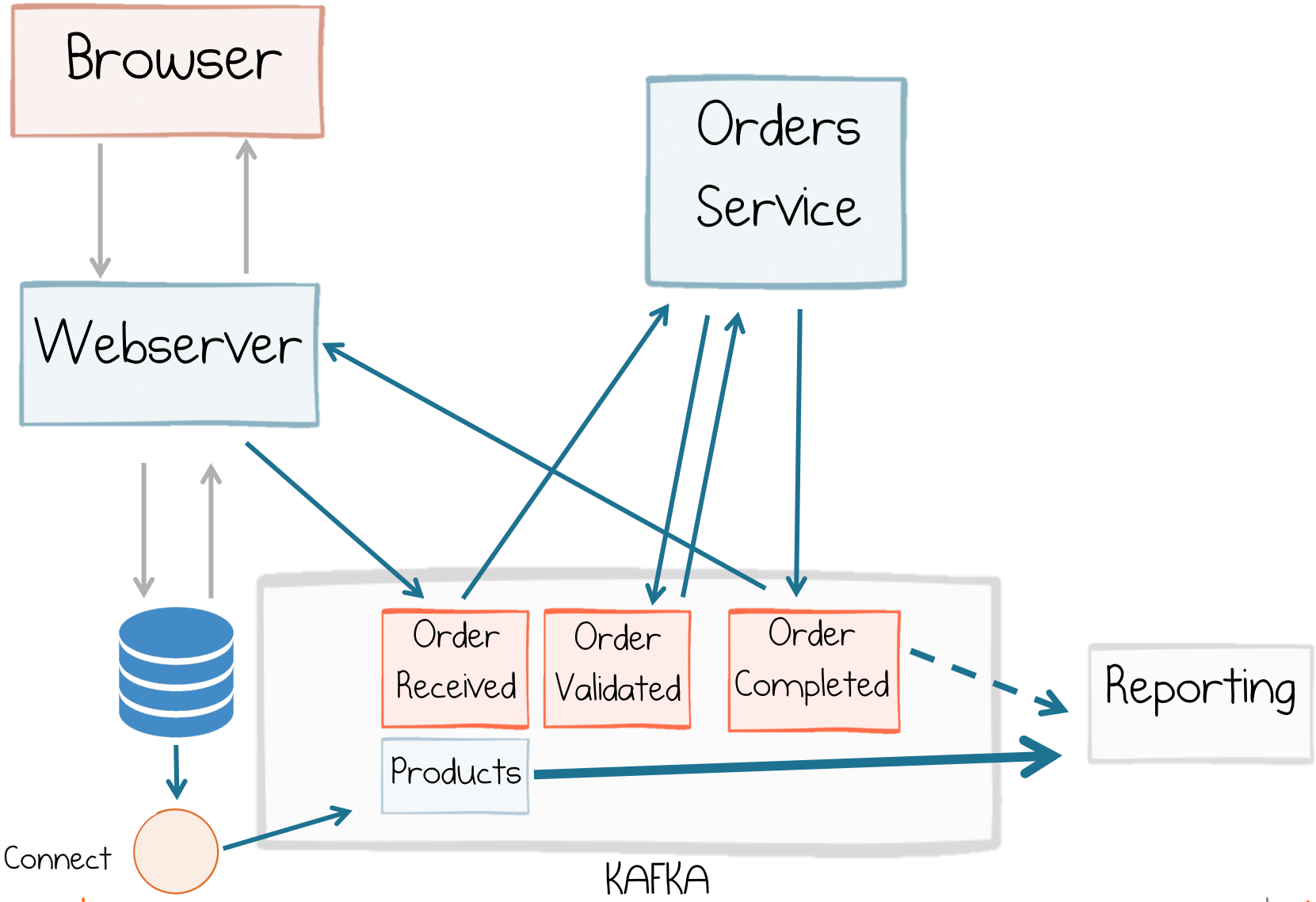
3. Convert legacy databases to Events (with CDC)

Make Legacy Datasets Available via the Log

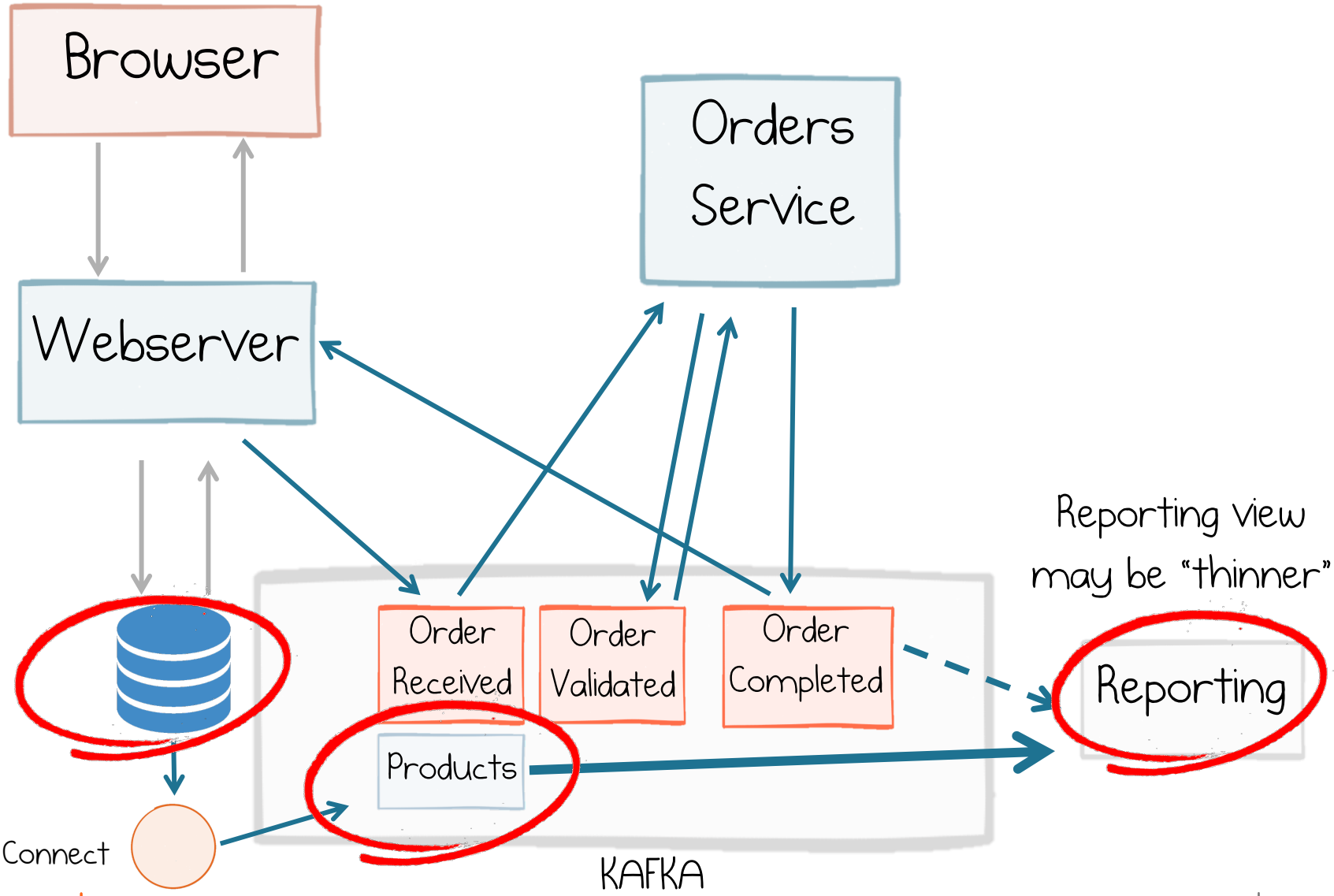


4. Use Kafka as an Event Store

Shared Source of Truth

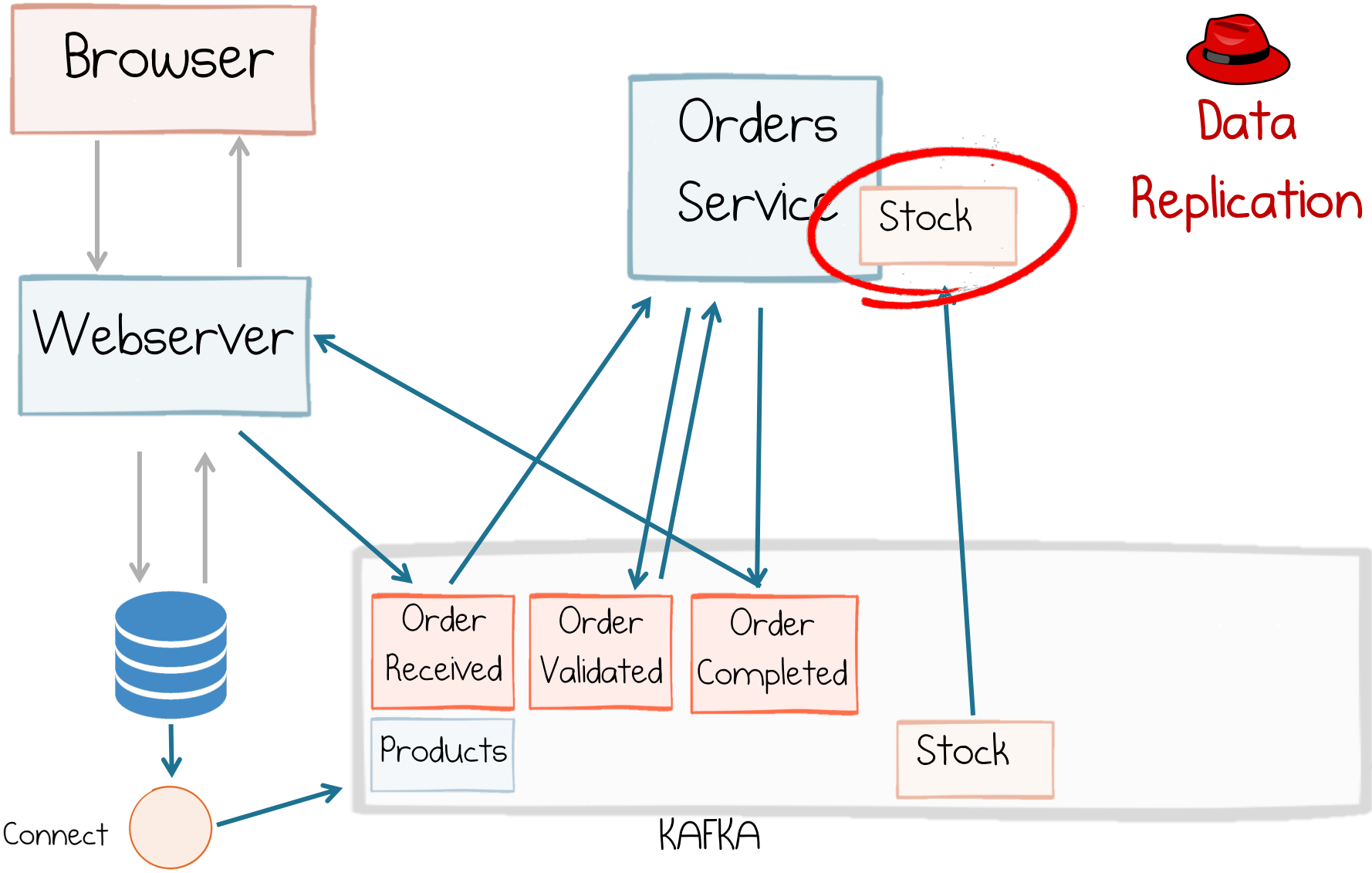


Product Catalogue stored in 3 places



5. Derive "Materialized Views" instead of caching

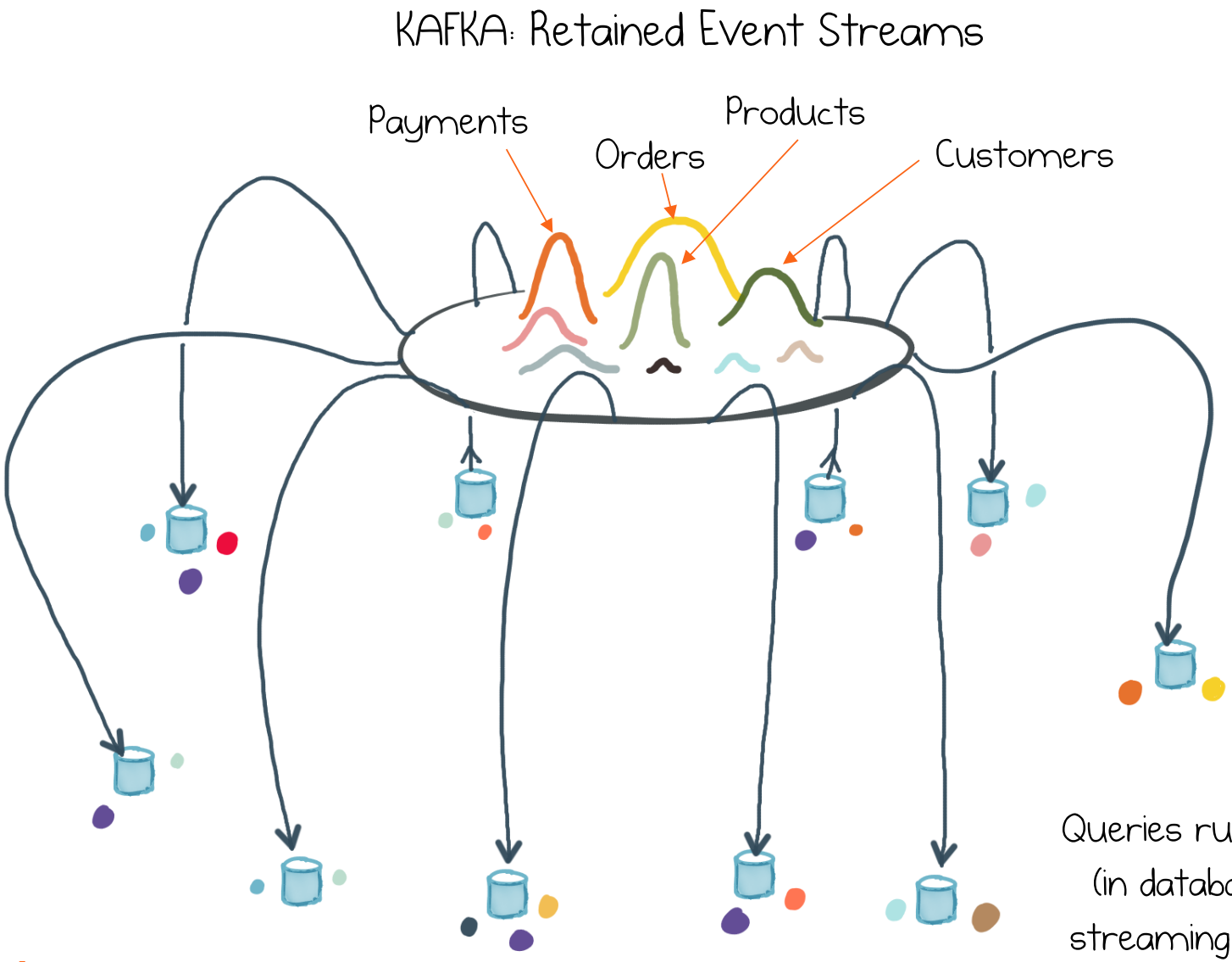
Materialize Stock 'View' Inside Service



Is stateful a good idea?

- Standby Replicas
- Disk Checkpoints
- Compacted topics
- Or just use a database!

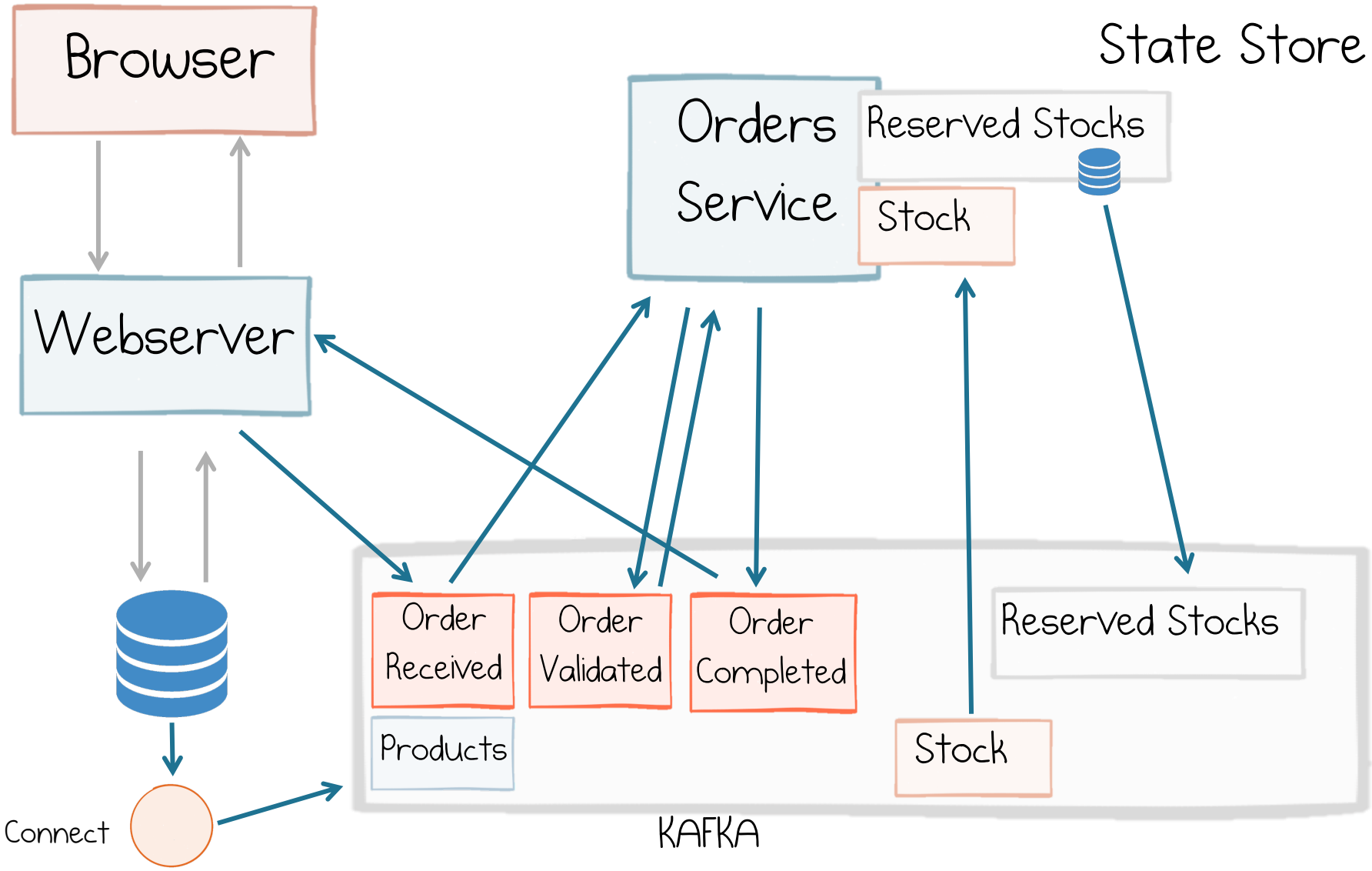
Database Inside Out Pattern



Queries run locally (in database or streaming engine)

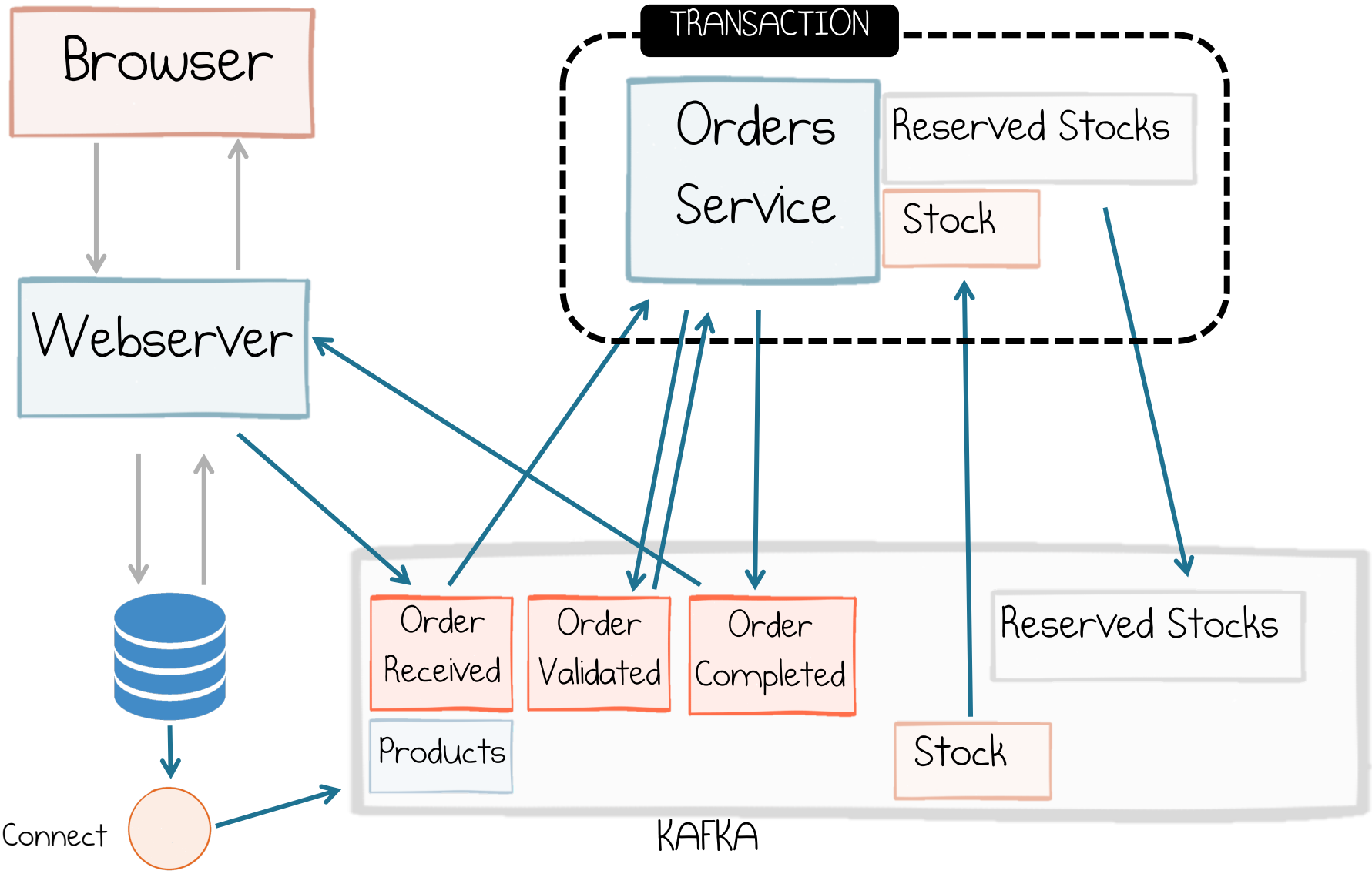
6. Write to State Stores, just like a local 'database', backed up in Kafka

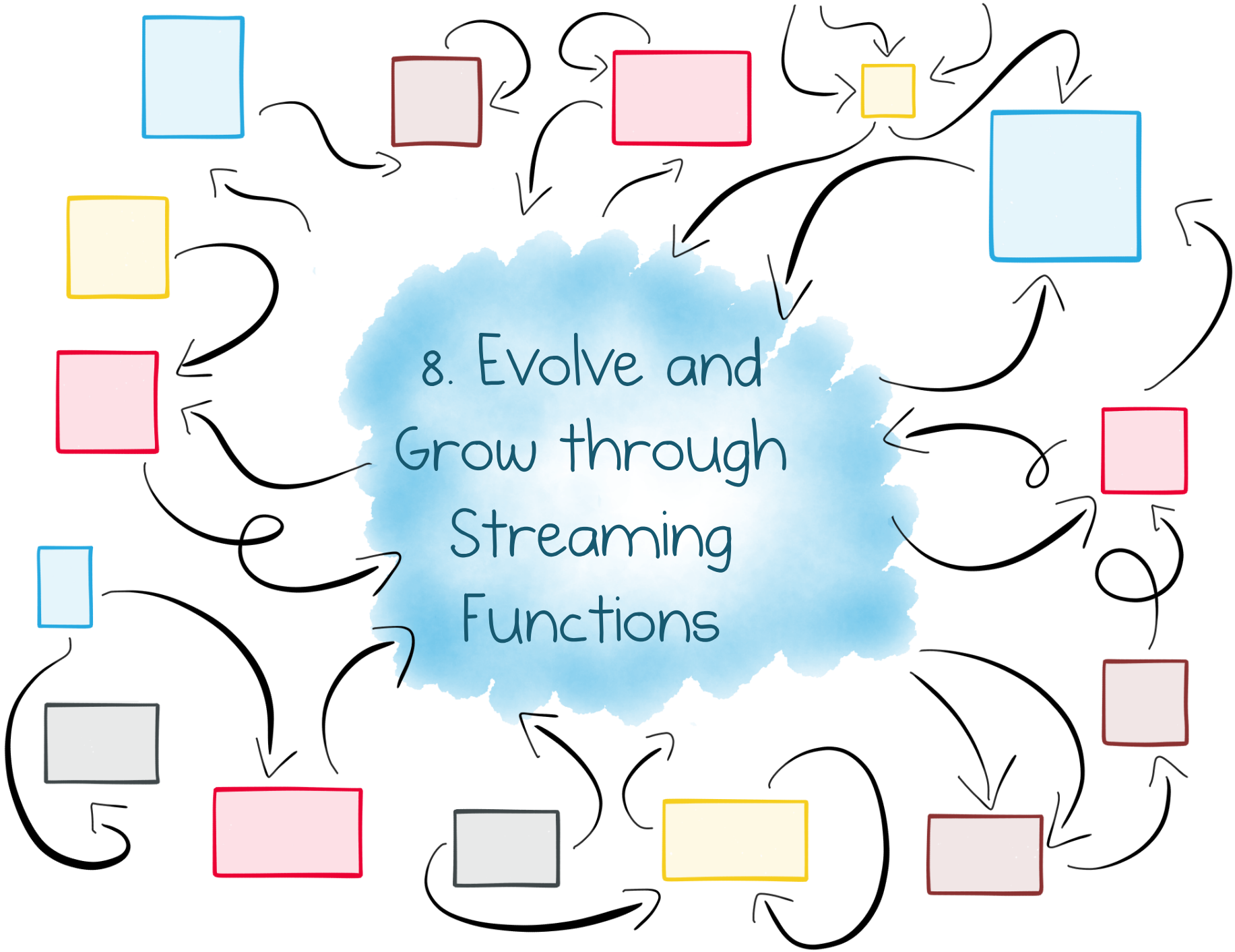
State stores behave like local databases



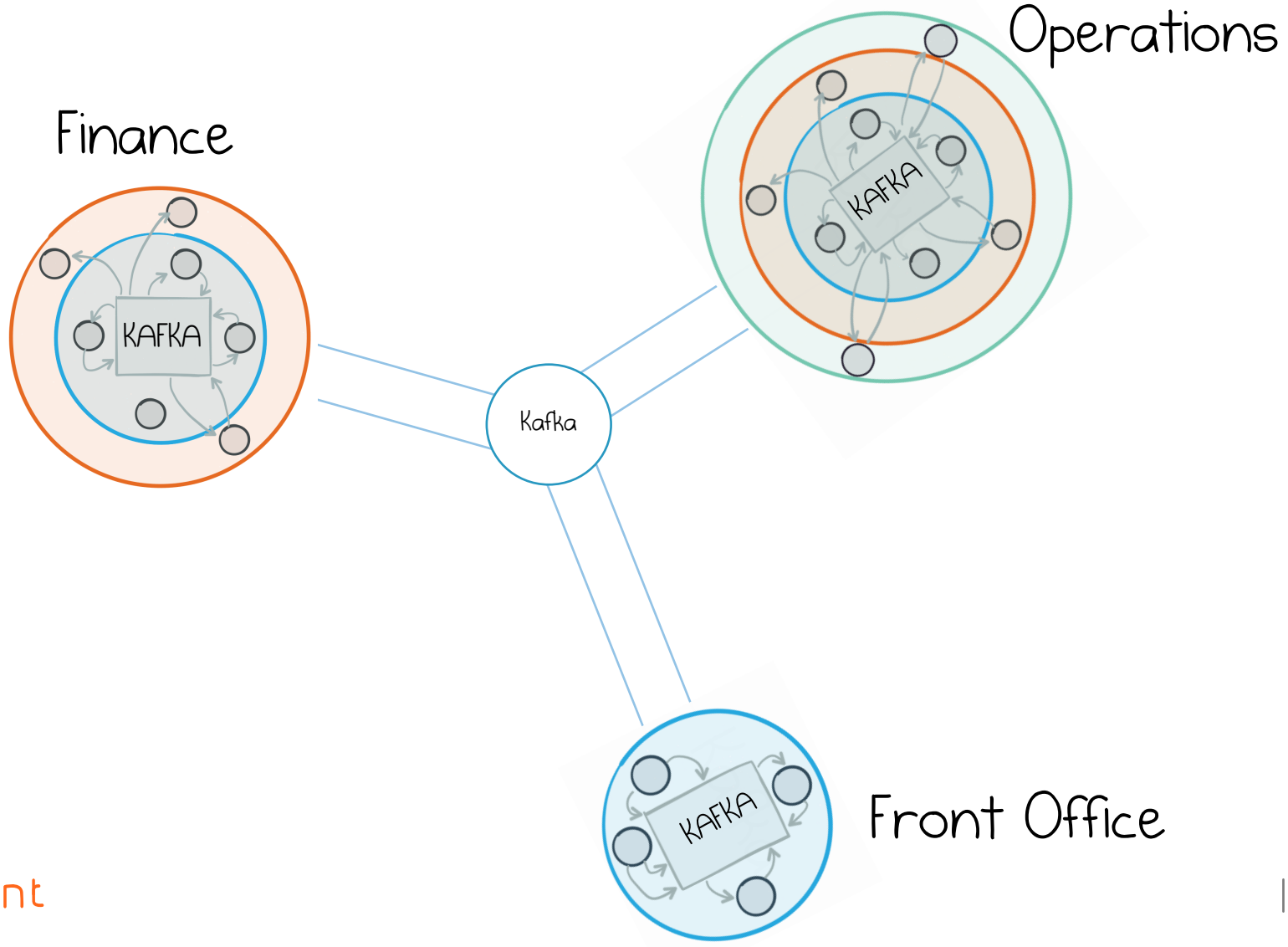
7. Use Transactions to tie All Interactions Together

Transactions

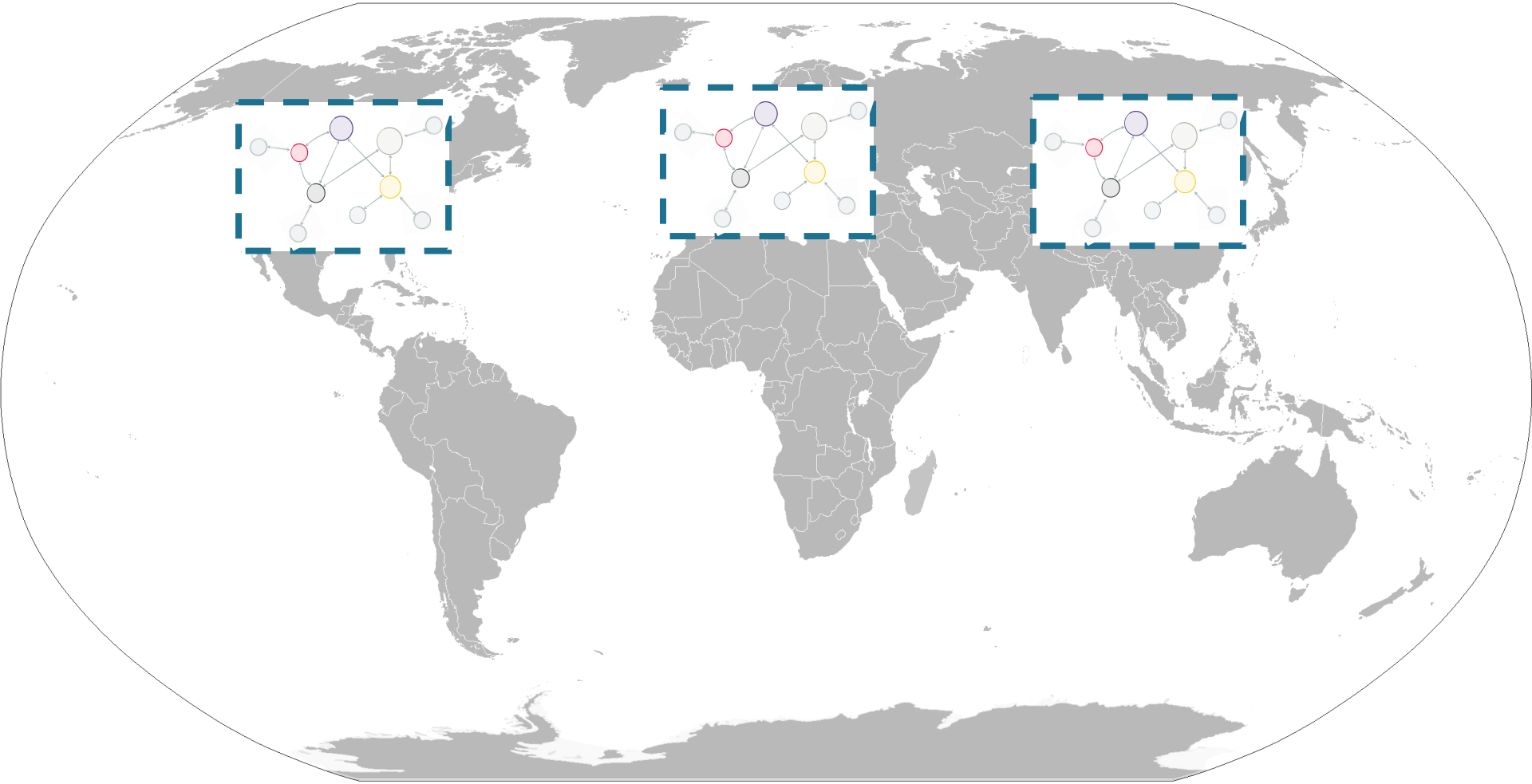



8. Evolve and
Grow through
Streaming
Functions

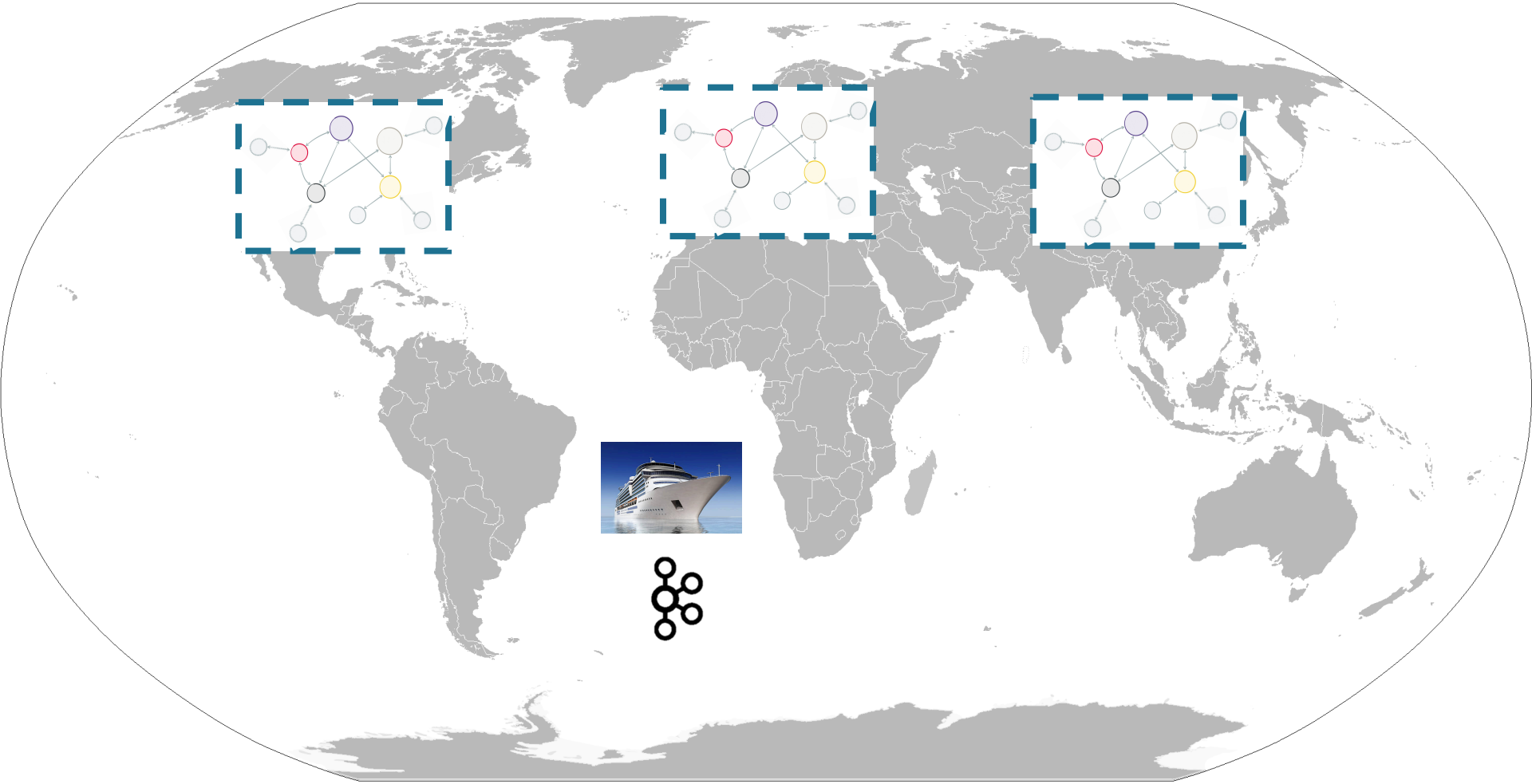
Tiered Contexts



Span regions or clouds



Handle Disconnectedness



So...

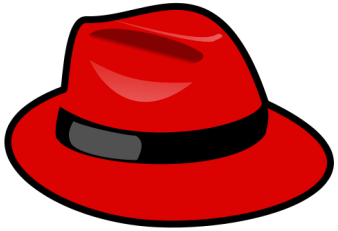
Optimize for complexity vs optimize for scale

Event Driven Architectures

Stream Processing

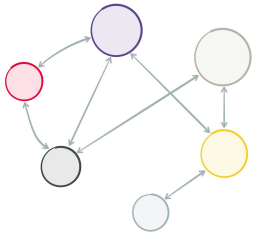
Events provide the key to evolutionary architectures

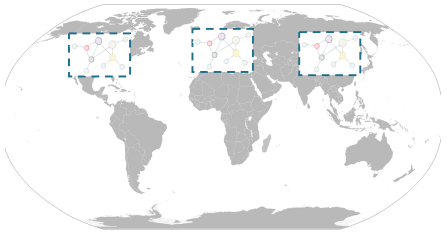
Notification



Data replication

Spectrum of use cases





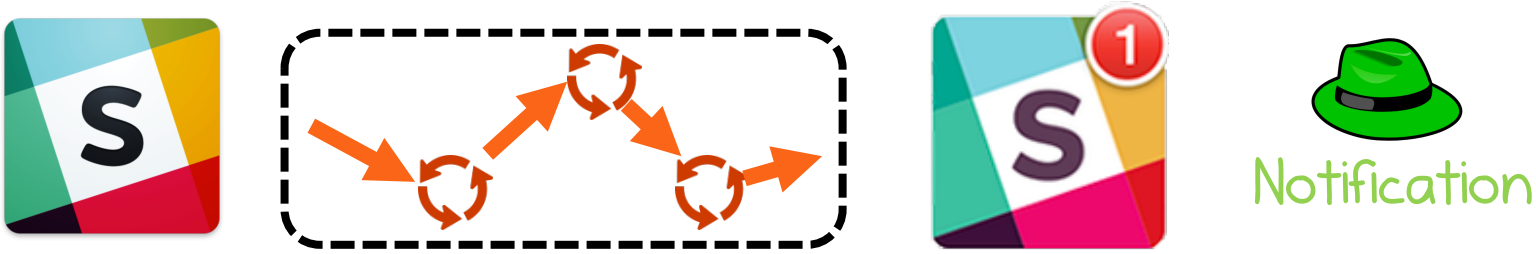
Finer Grained,
Collaborative,
Connected

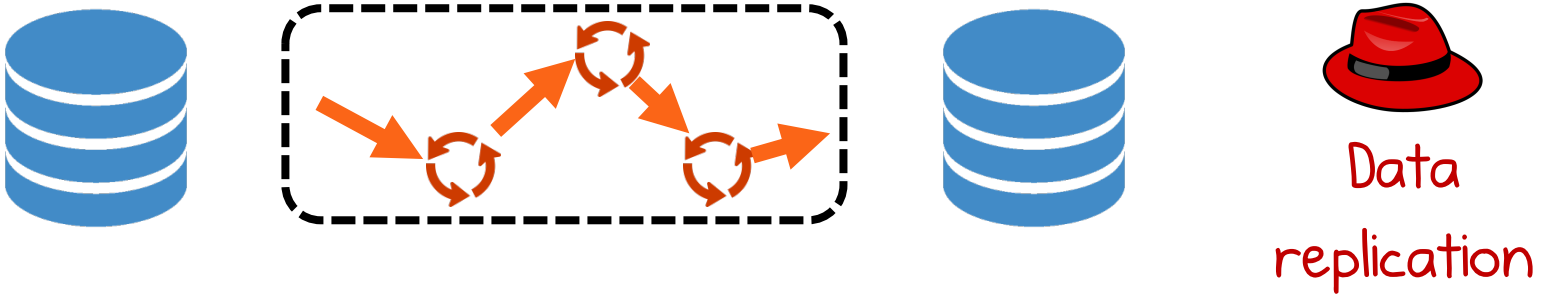
Courser Grained,
Non-collaborative,
Disconnected

Notification

Data Replication

Events to transcend individual services



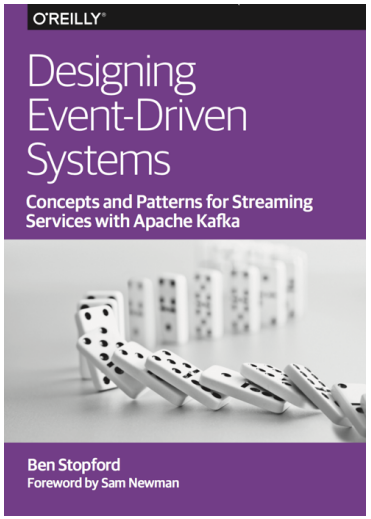


Start Simple and Evolve

1. Broadcast events
2. Retain them in the log
3. Evolve the event-stream with streaming functions
4. Recasting the event stream into views when you need to query.

Find out more

Book: http://bit.ly/designing-event-driven-systems



Software: https://confluent.io/download/

Cloud: https://www.confluent.io/confluent-cloud/

Kubernetes Operator: https://www.confluent.io/confluent-operator/

Example code: http://bit.ly/kafka-microservice-examples

Twitter: @benstopford