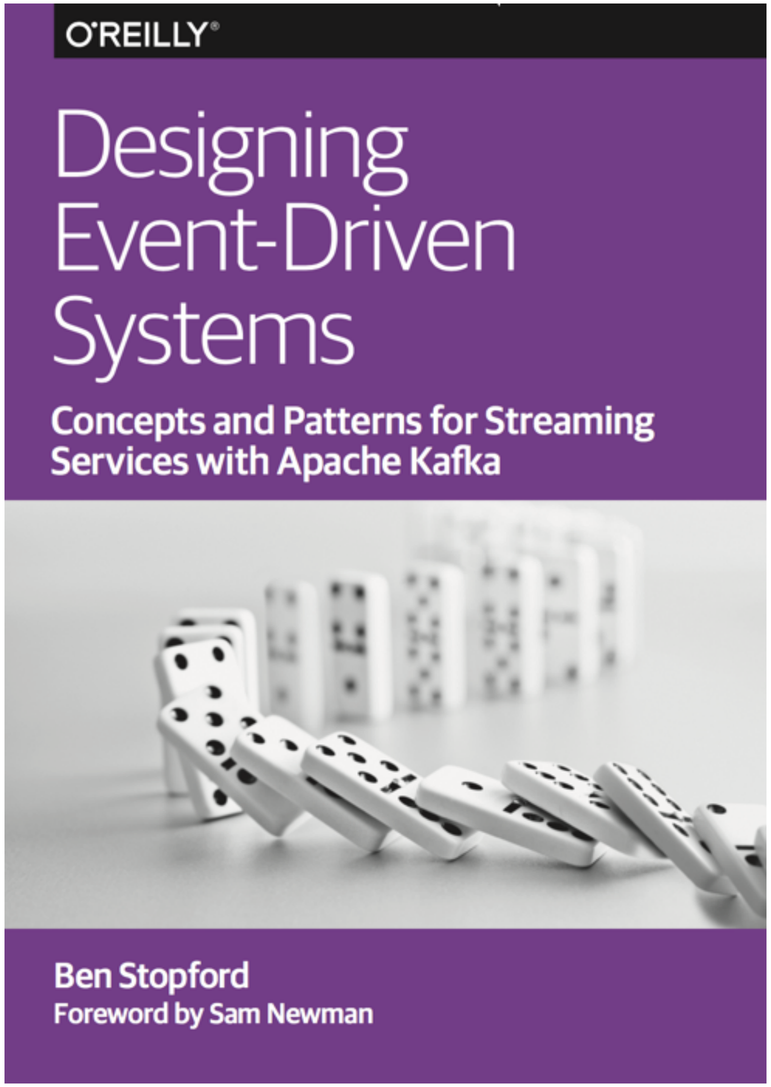
Building Event Driven Services with Apache Kafka, Kafka Streams & KSQL

Ben Stopford

@benstopford

There is a book!


O'REILLY
Designing Event-Driven Systems
Concepts and Patterns for Streaming Services with Apache Kafka
Ben Stopford
Foreword by Sam Newman

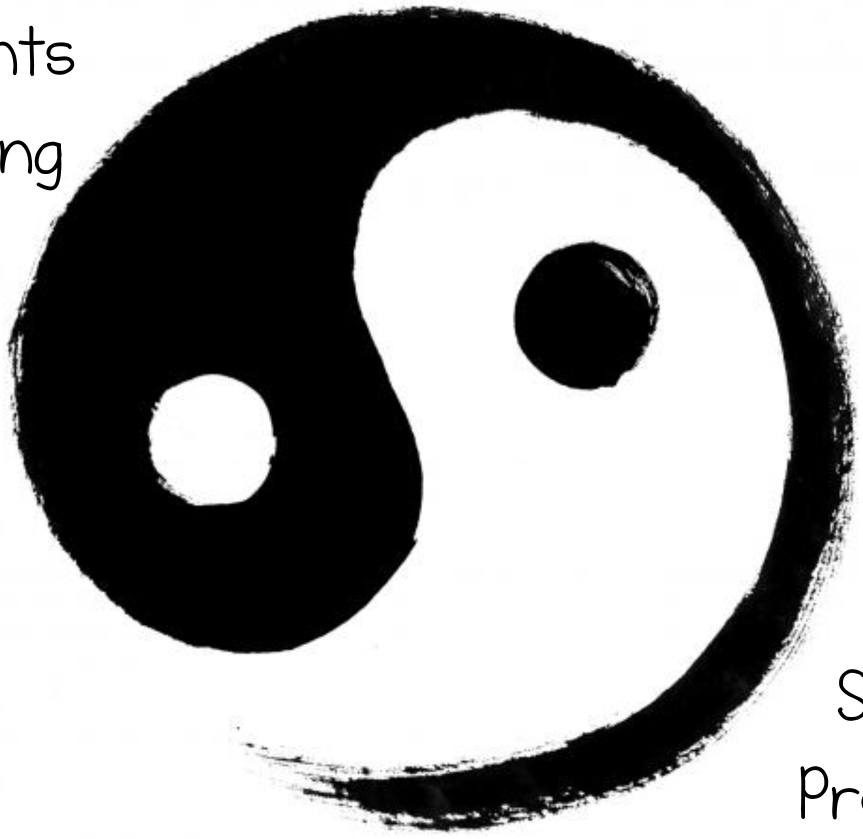
http://bit.ly/designing-event-driven-systems

Event Driven Architectures

Business Events

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

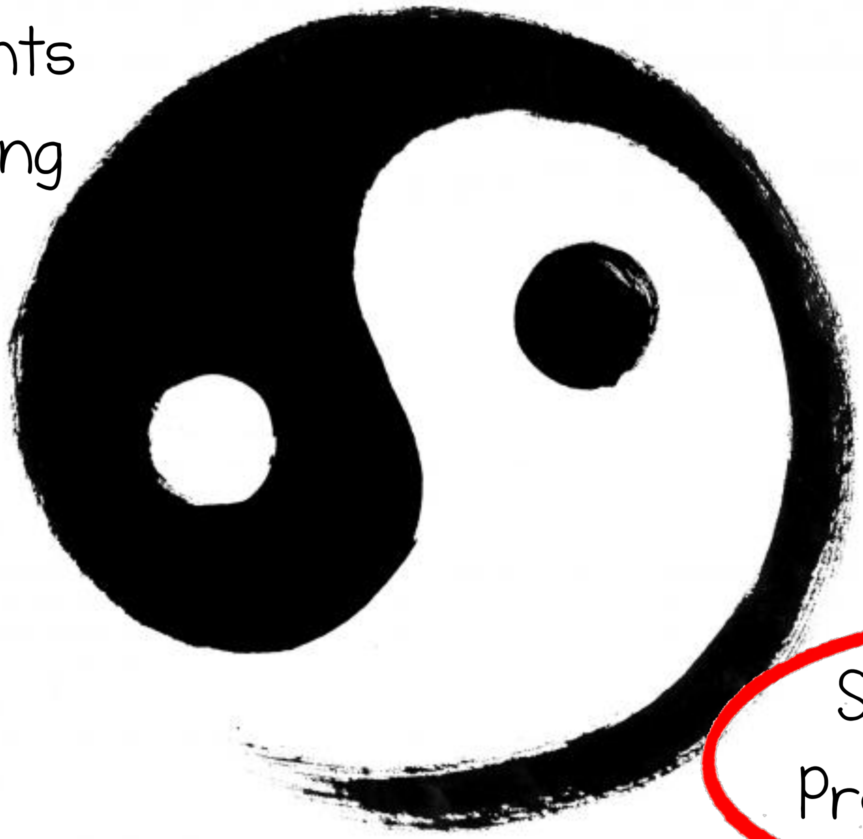
- 1 billion messages per day
- 20,000 messages per second
- 100 teams

Event Driven Architectures

Business Events

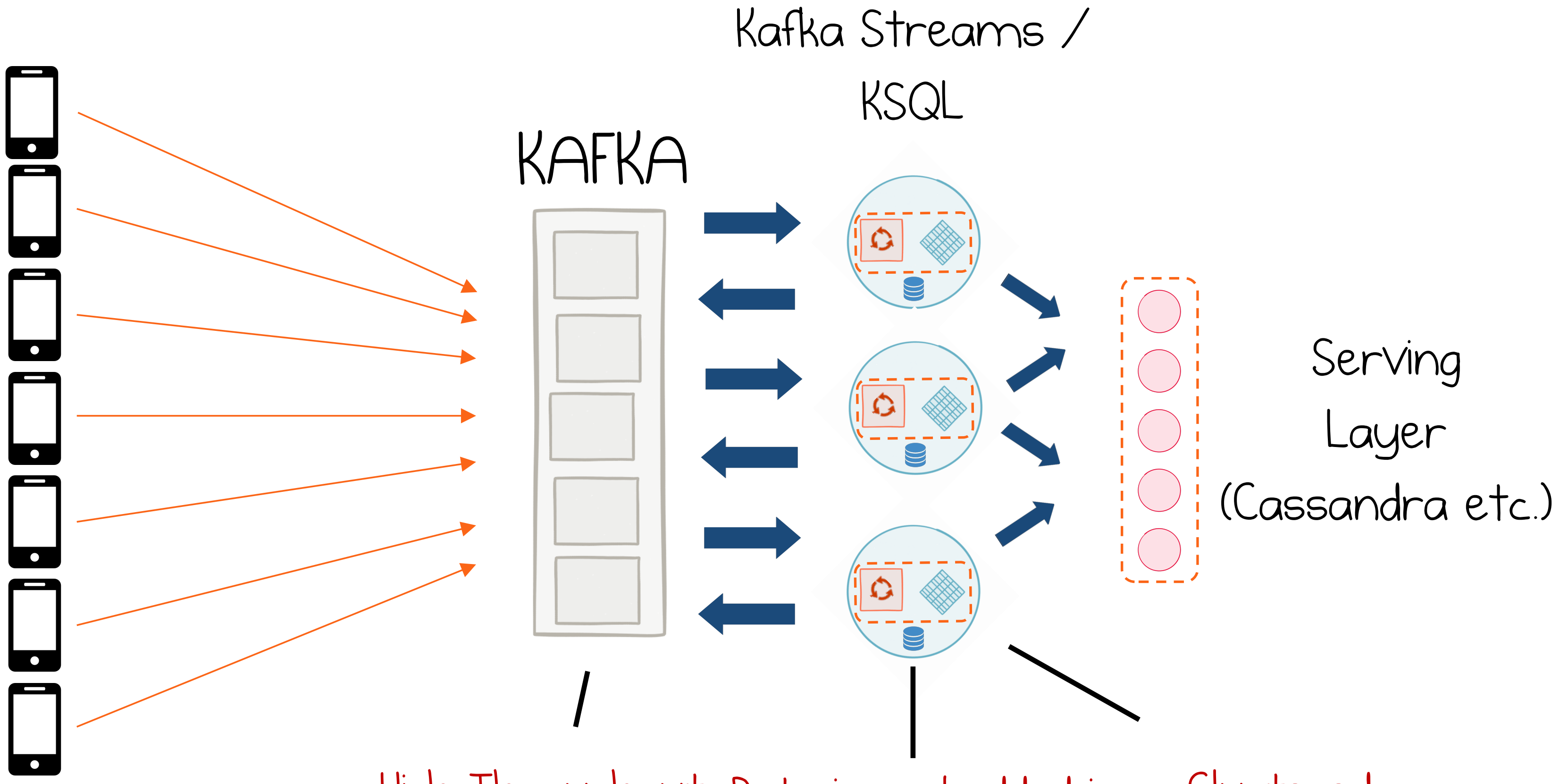
Event Sourcing

DDD



Stream Processing

Streaming Platforms



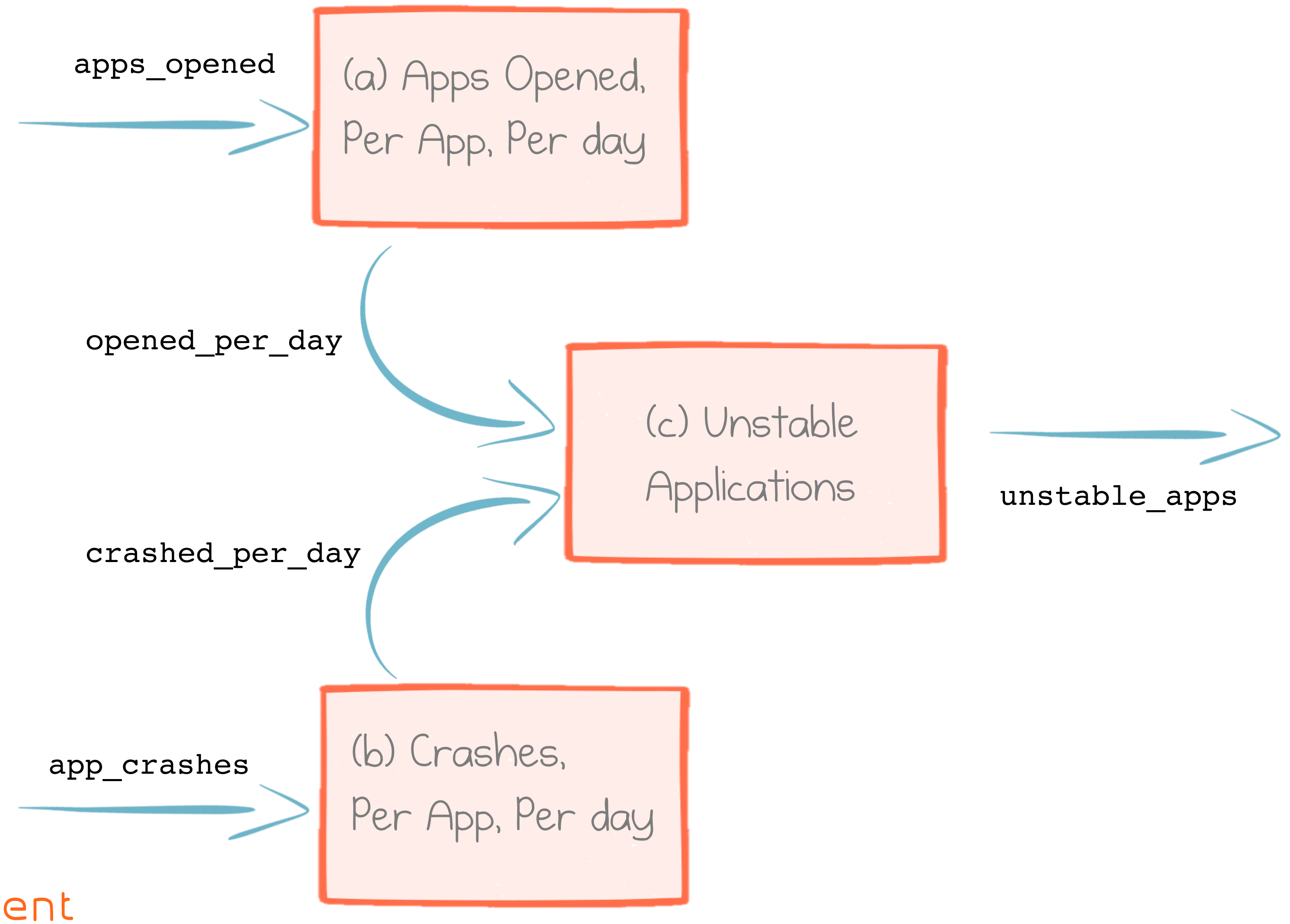
High Throughput Messaging

Data is embedded in each engine

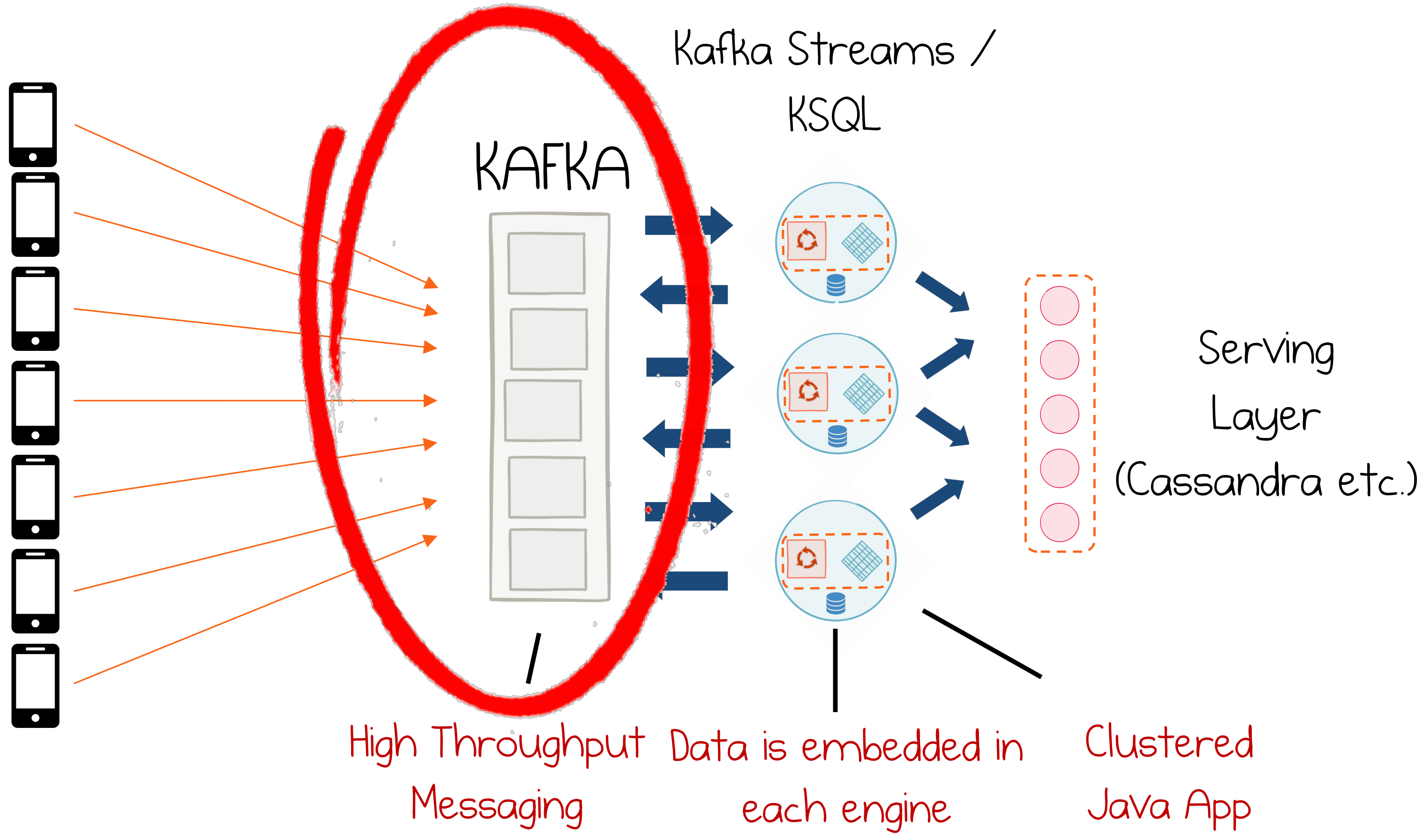
Clustered Java App

confluent

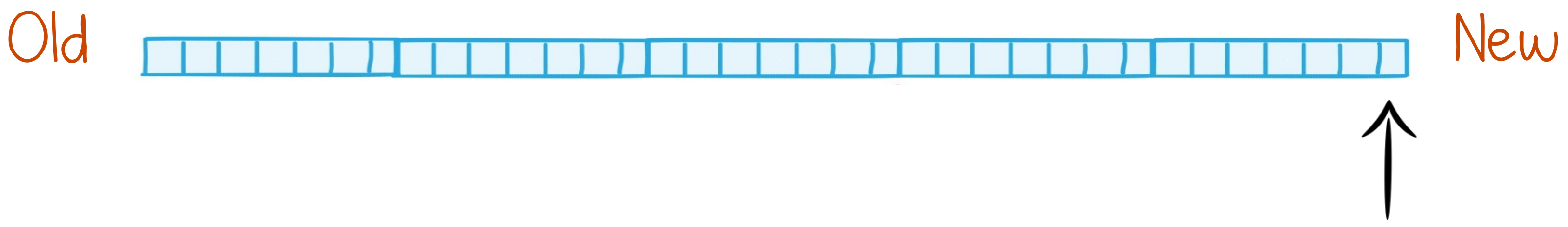
Streaming Pipeline


apps_opened
(a) Apps Opened, Per App, Per day
opened_per_day
(c) Unstable Applications
unstable_apps
crashed_per_day
app_crashes
(b) Crashes, Per App, Per day

Streaming Platforms

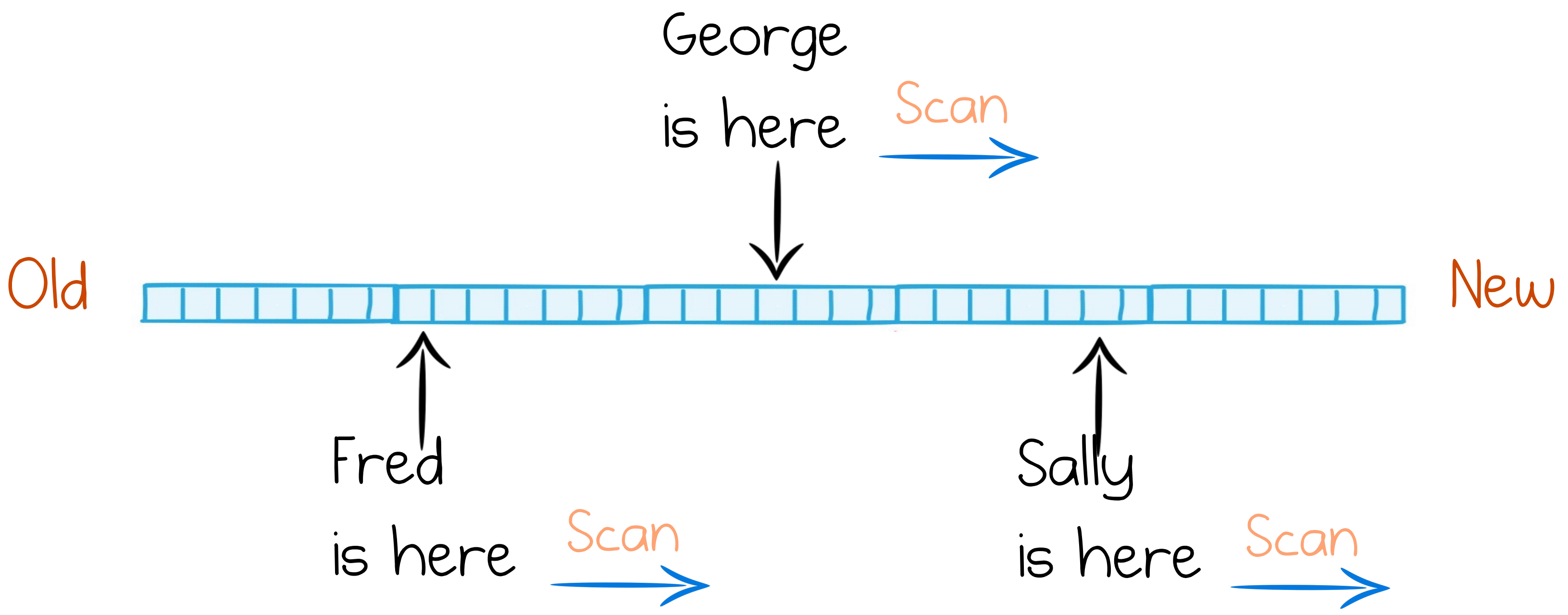

Kafka Streams / KSQL
KAFKA
Serving Layer (Cassandra etc.)
High Throughput Messaging
Data is embedded in each engine
Clustered Java App

An event log is a simple idea

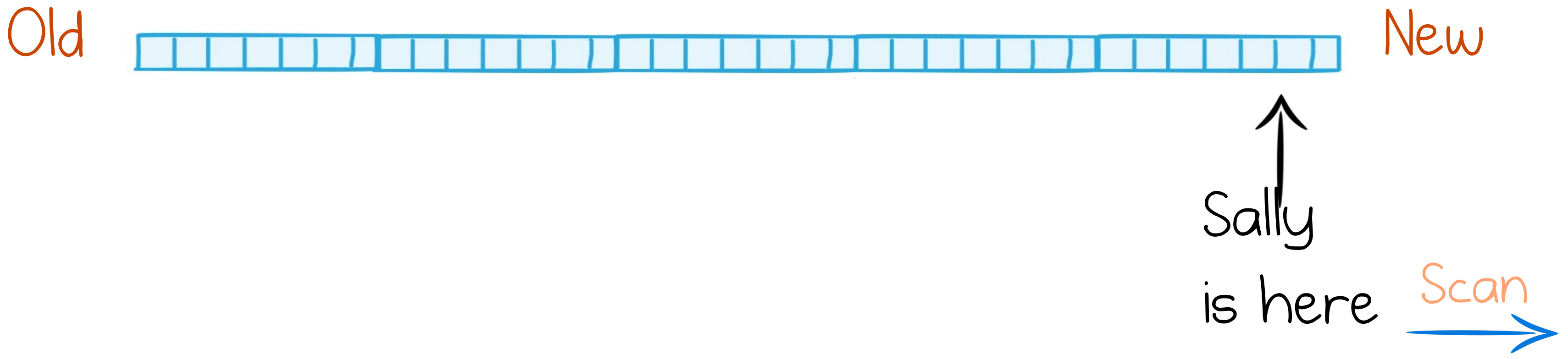


Messages are added at the end of the log

Readers have a position all of their own

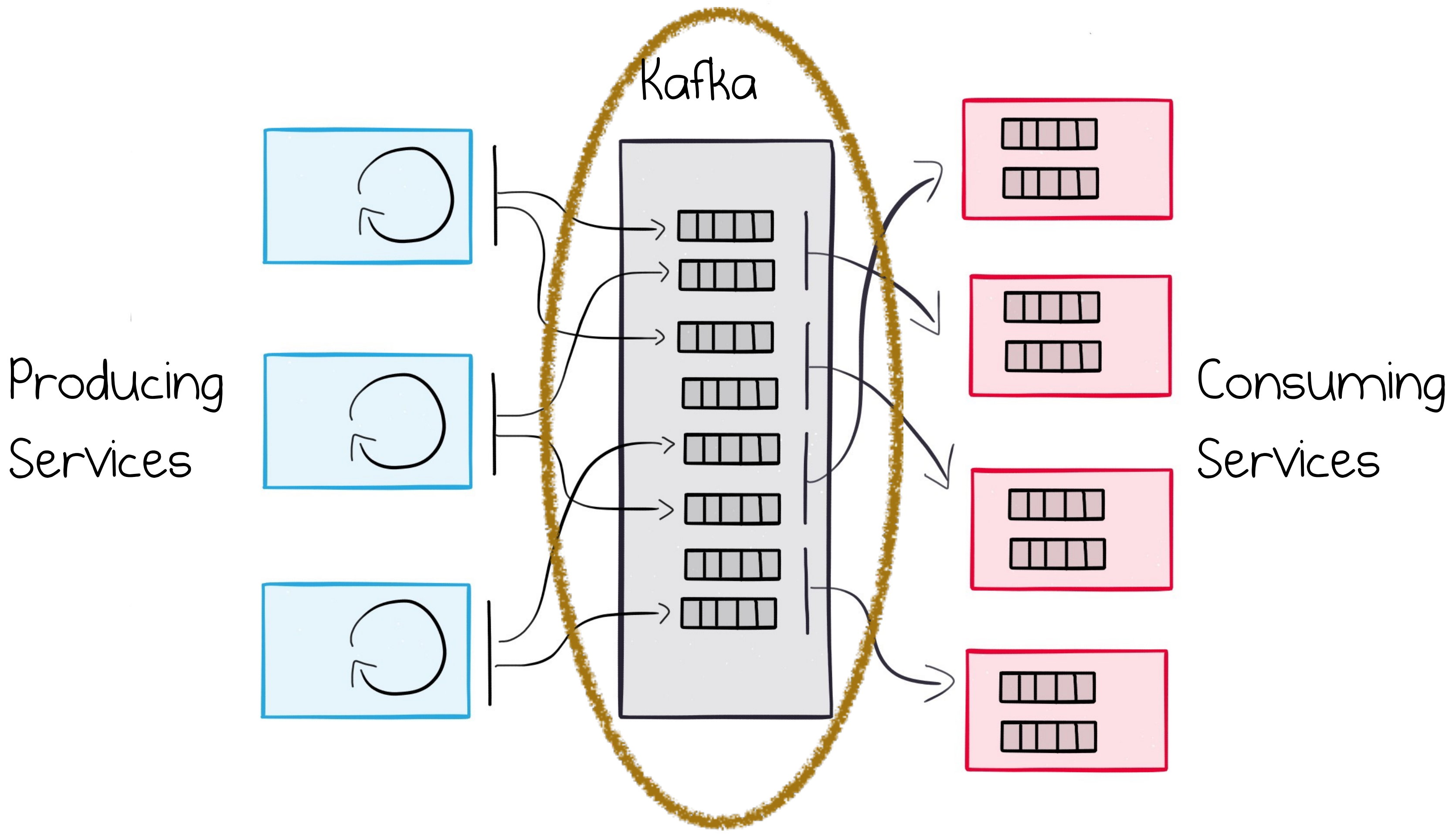


You can rewind and replay, just like Tivo!

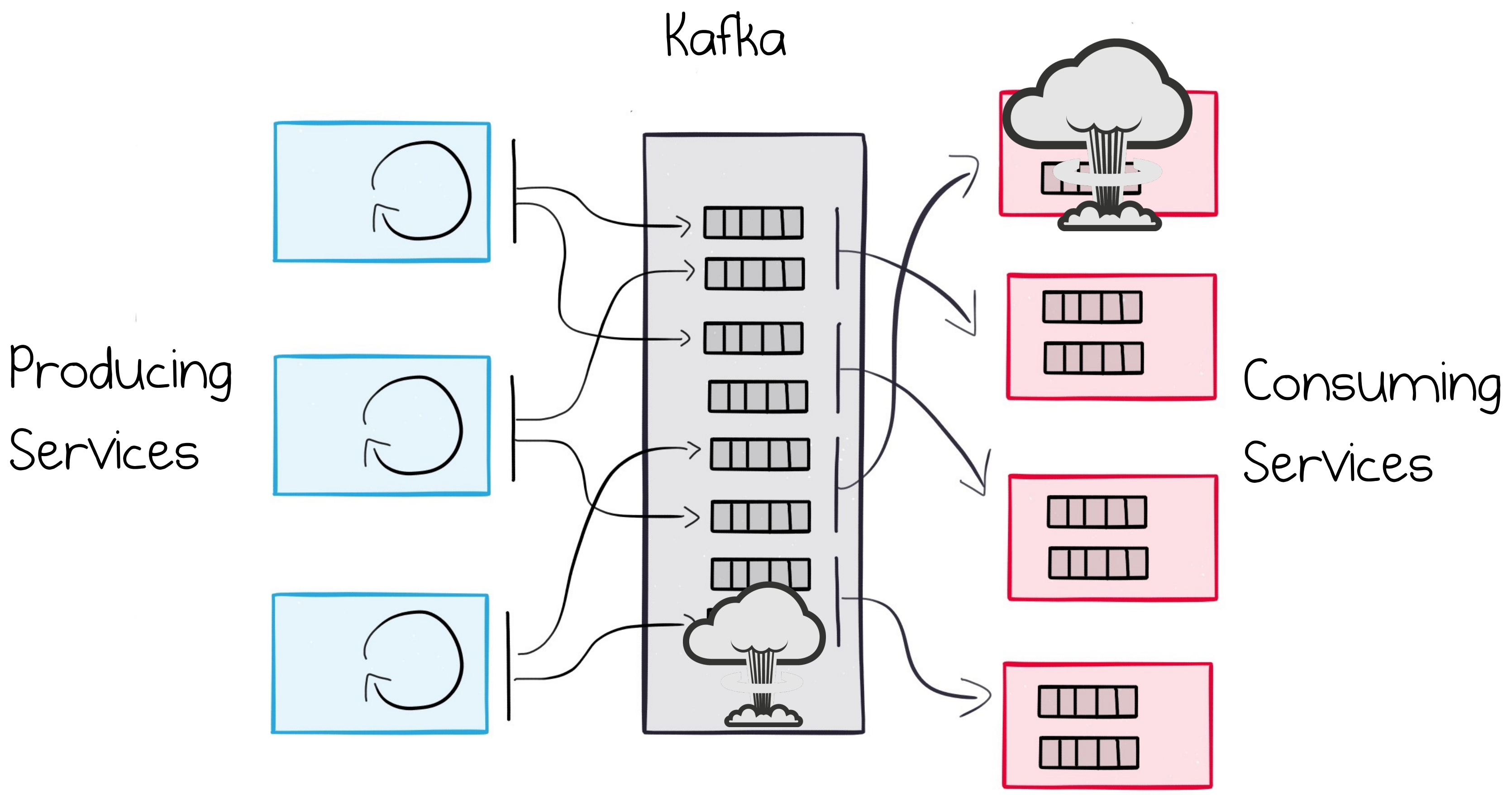


The hard part: Tying it all together!

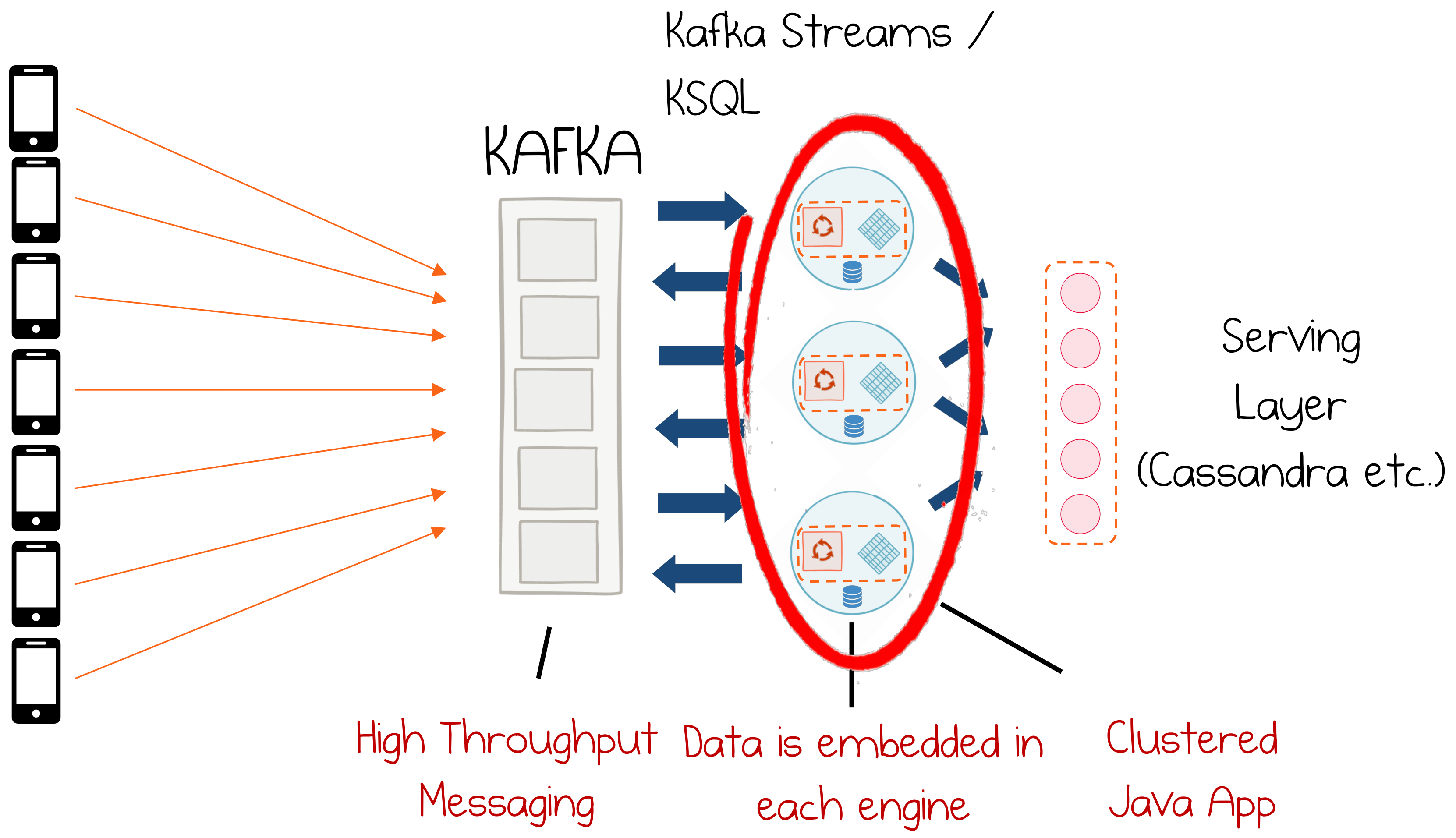
Many "logs" over many machines



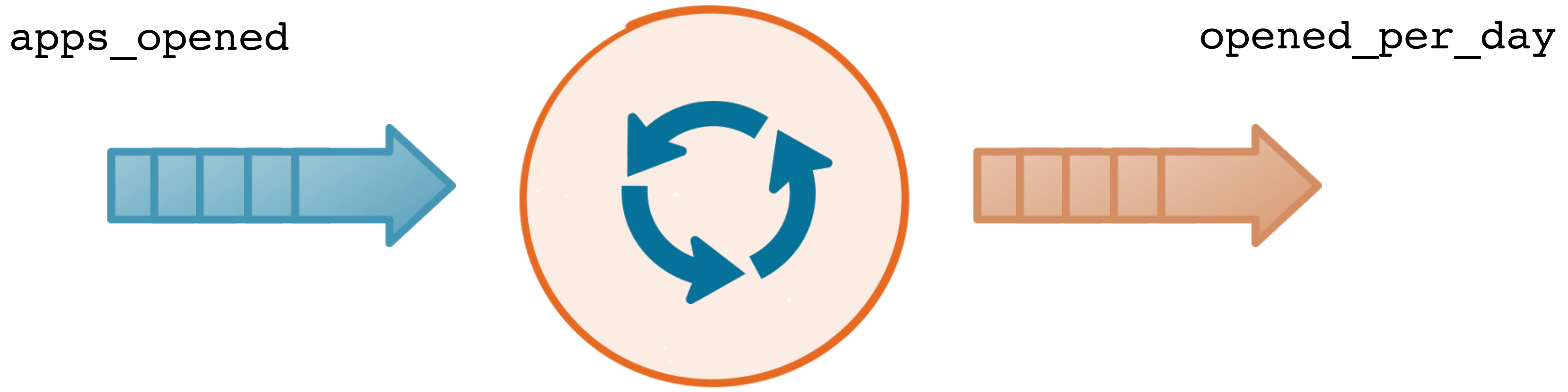
Resistant to Failure

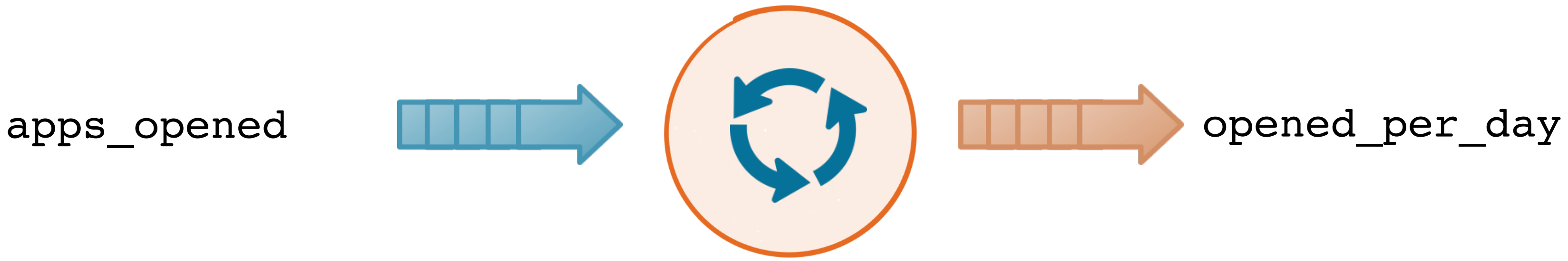


Streaming Platforms

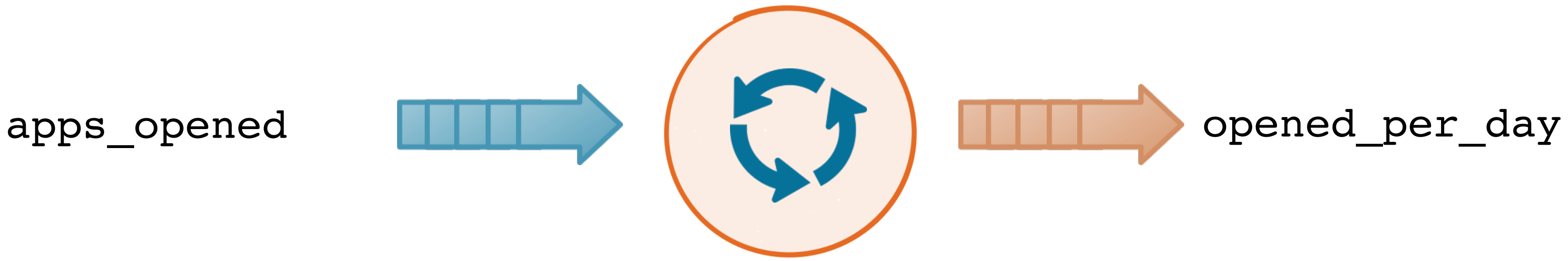


Streaming Example

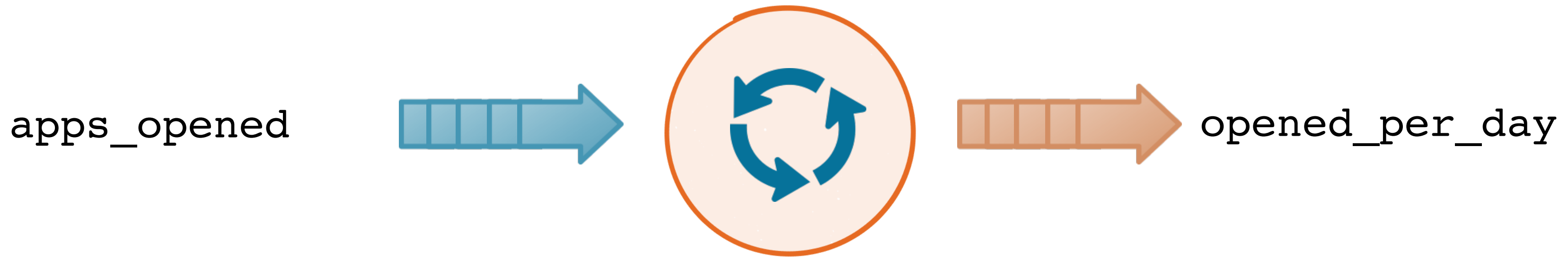




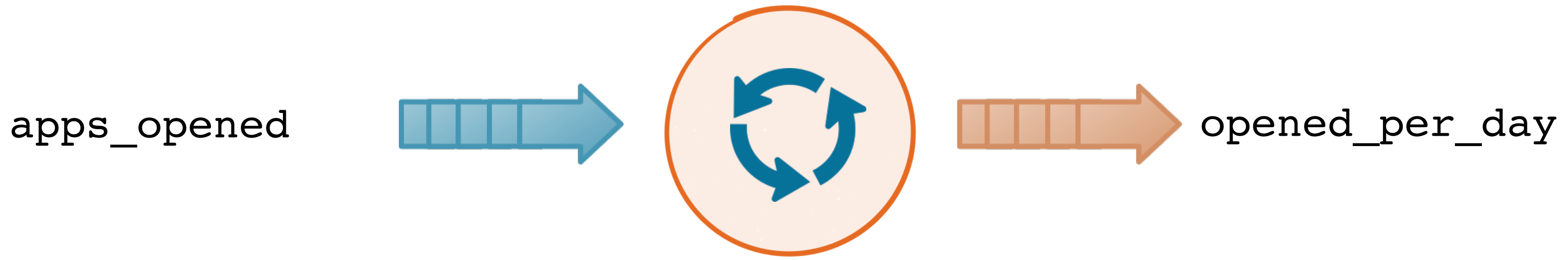
```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



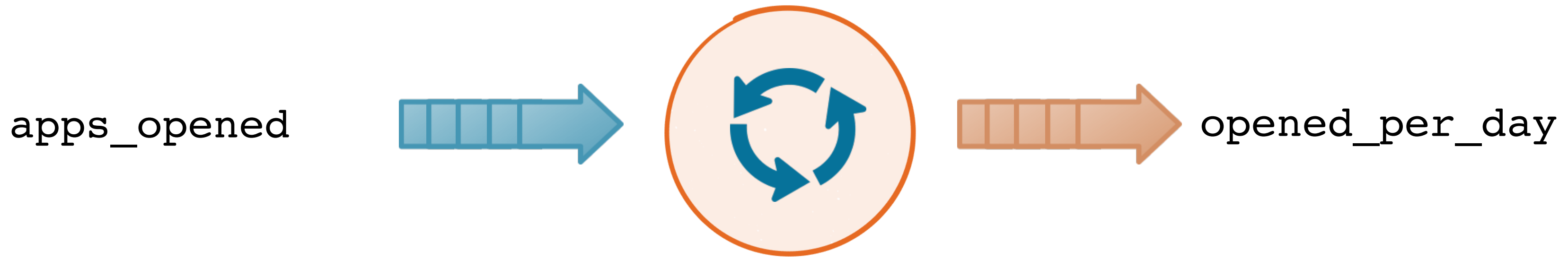
```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



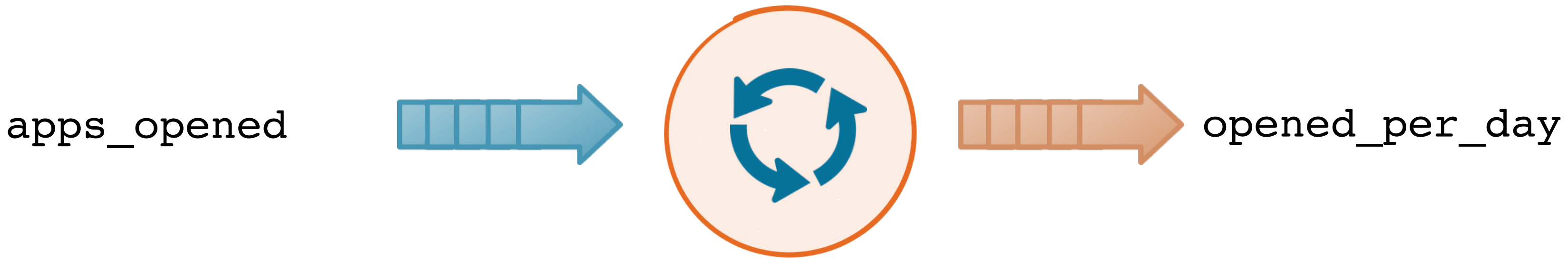
```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```



```
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

Business Events

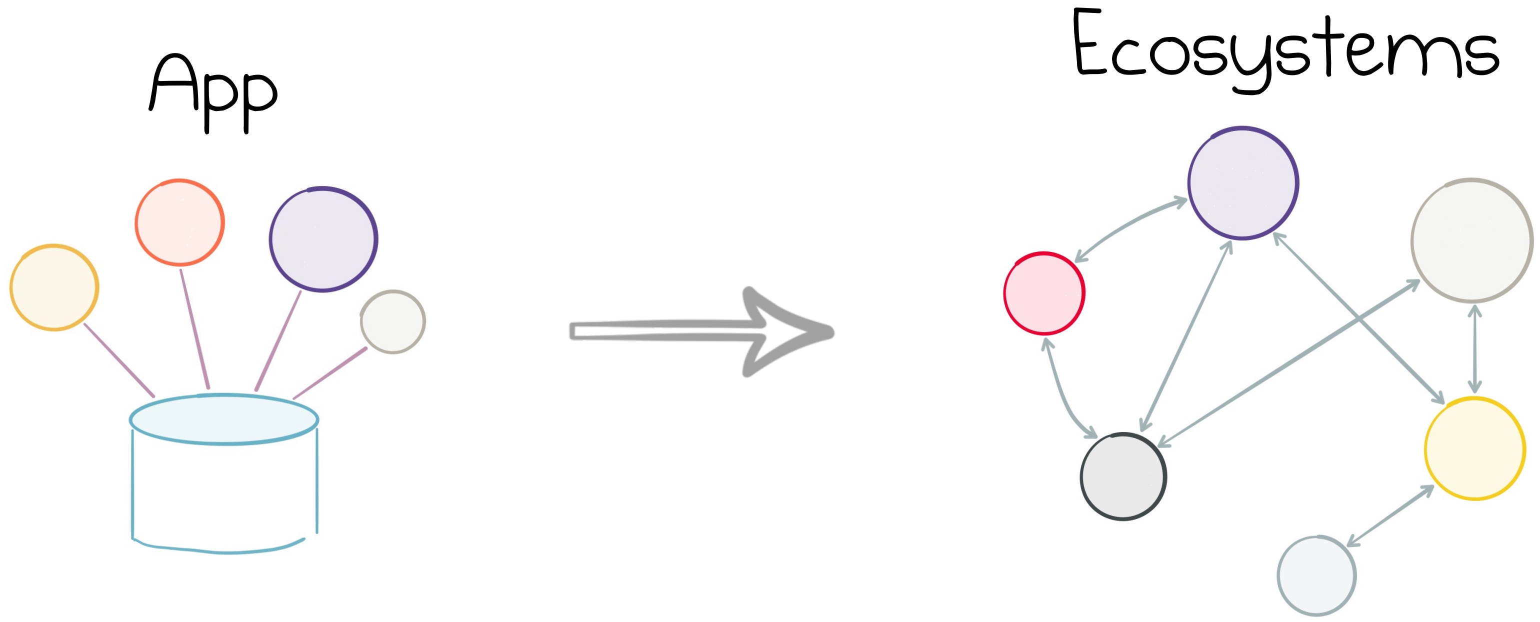
Event Sourcing

DDD

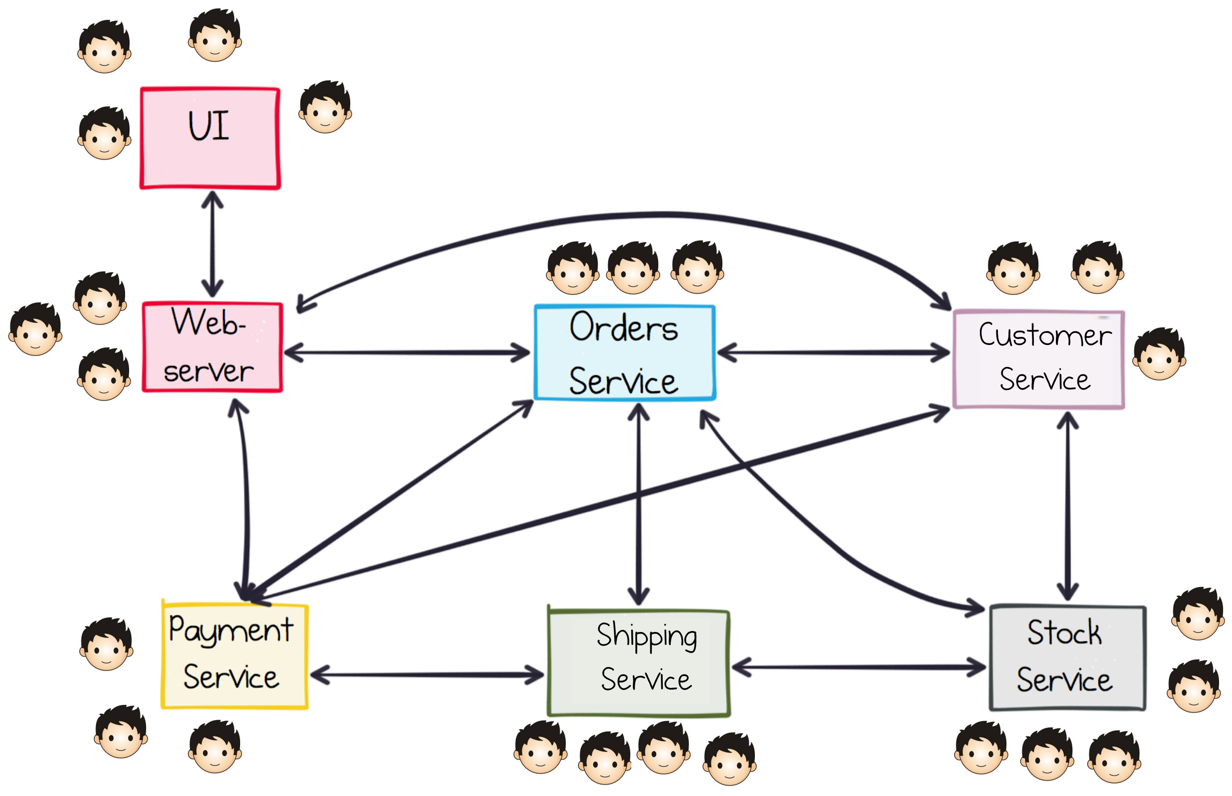
Stream

Processing

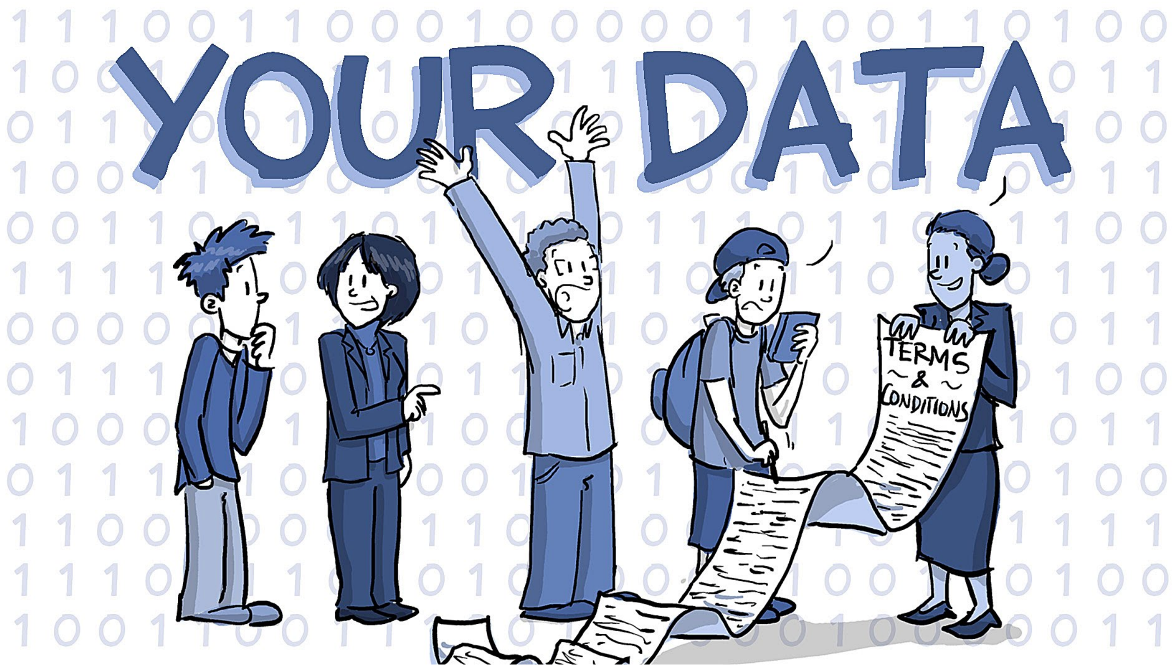
Increasingly we build ecosystems



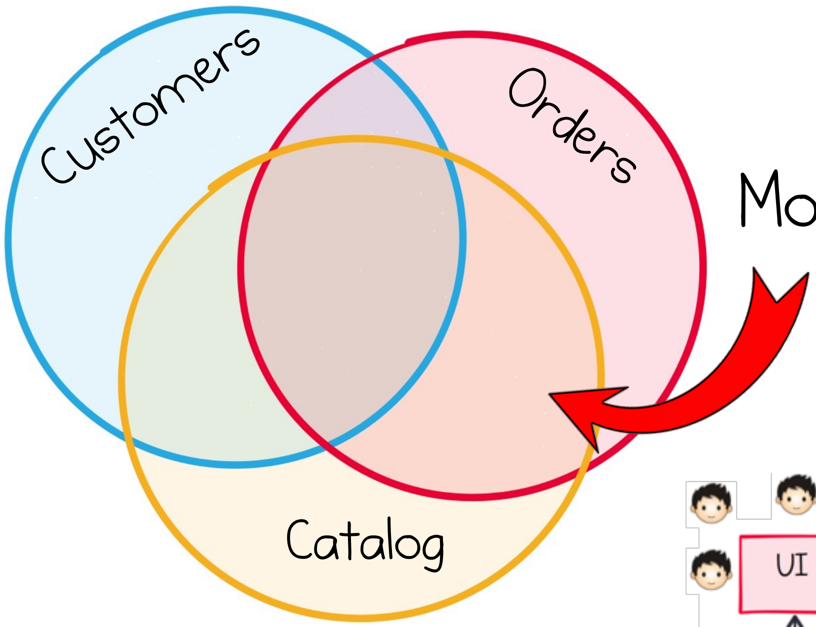
SOA / Microservices / EDA



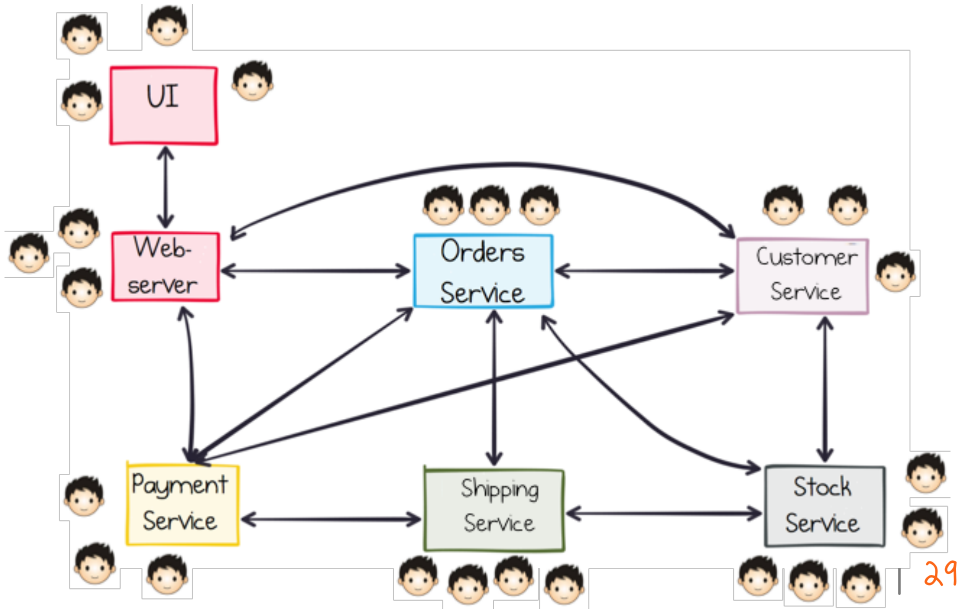
The Problem is DATA



Most services share the same core facts.

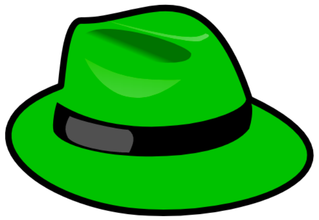


Most services live in here

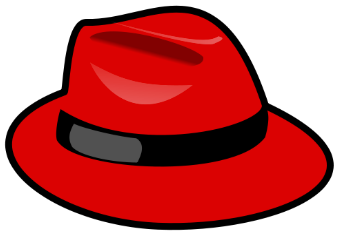


confluent

Events have two hats

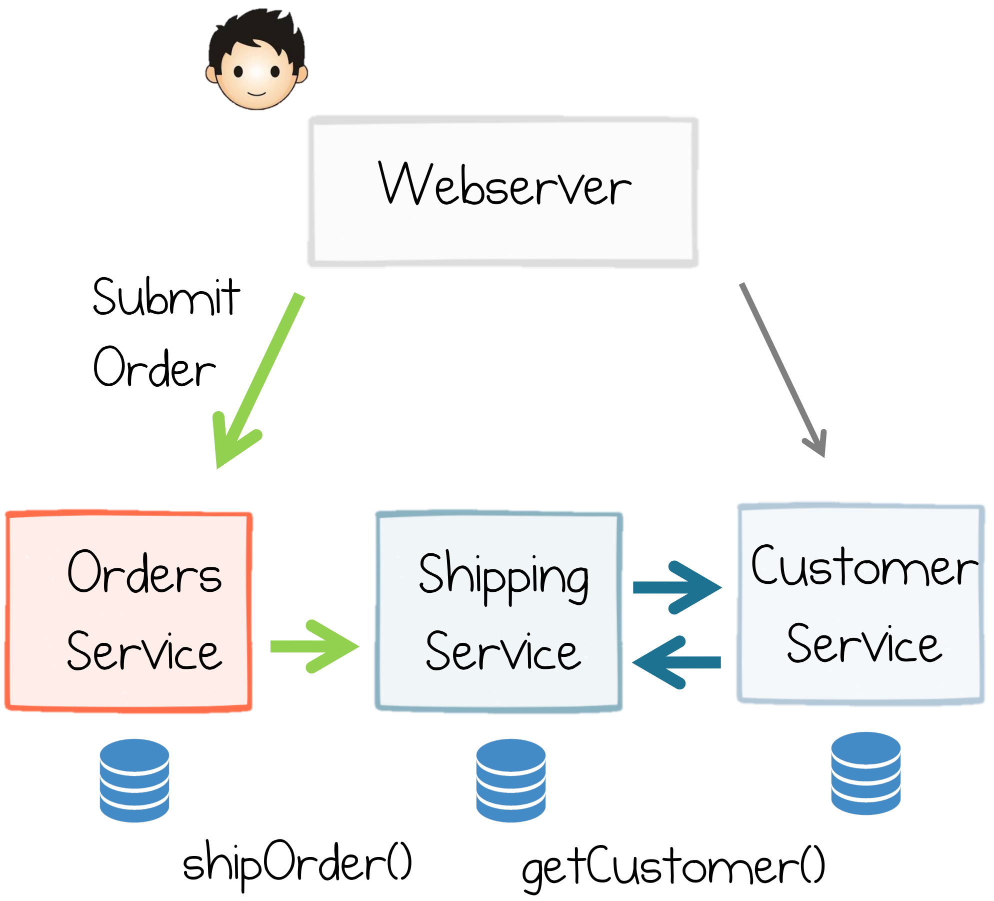


Notification

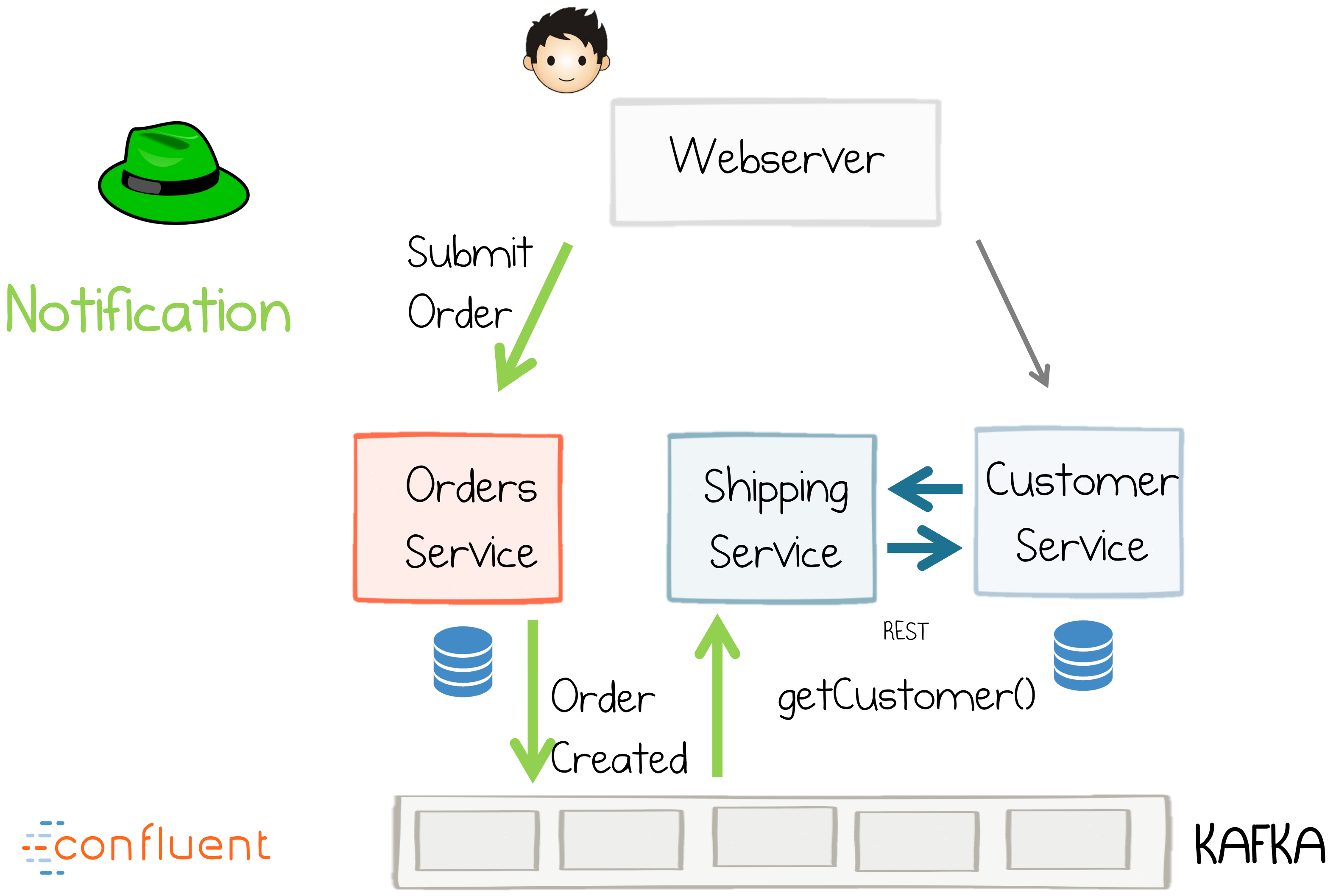


Data replication

Buying an iPad (with REST/RPC)

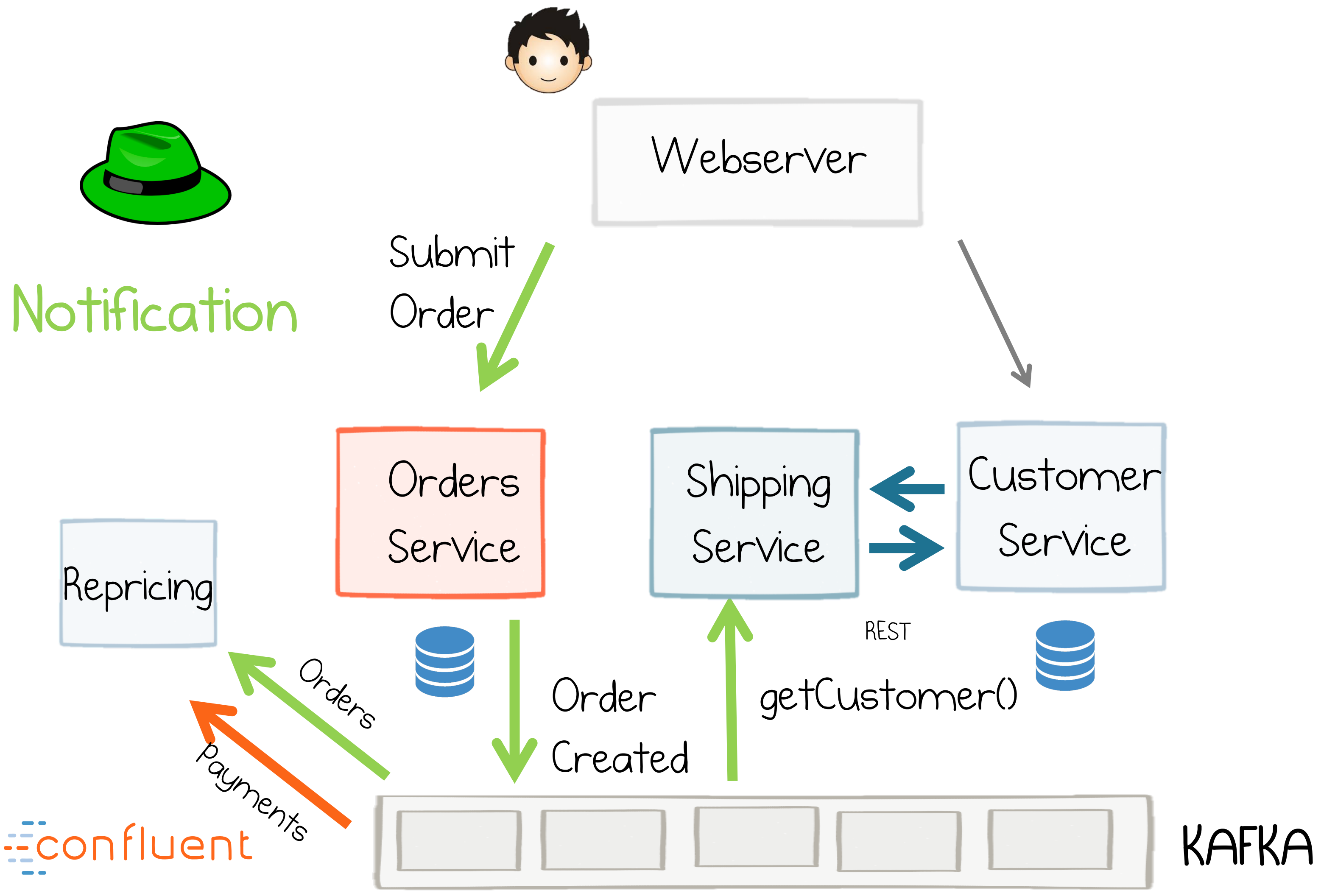


Events for Notification Only

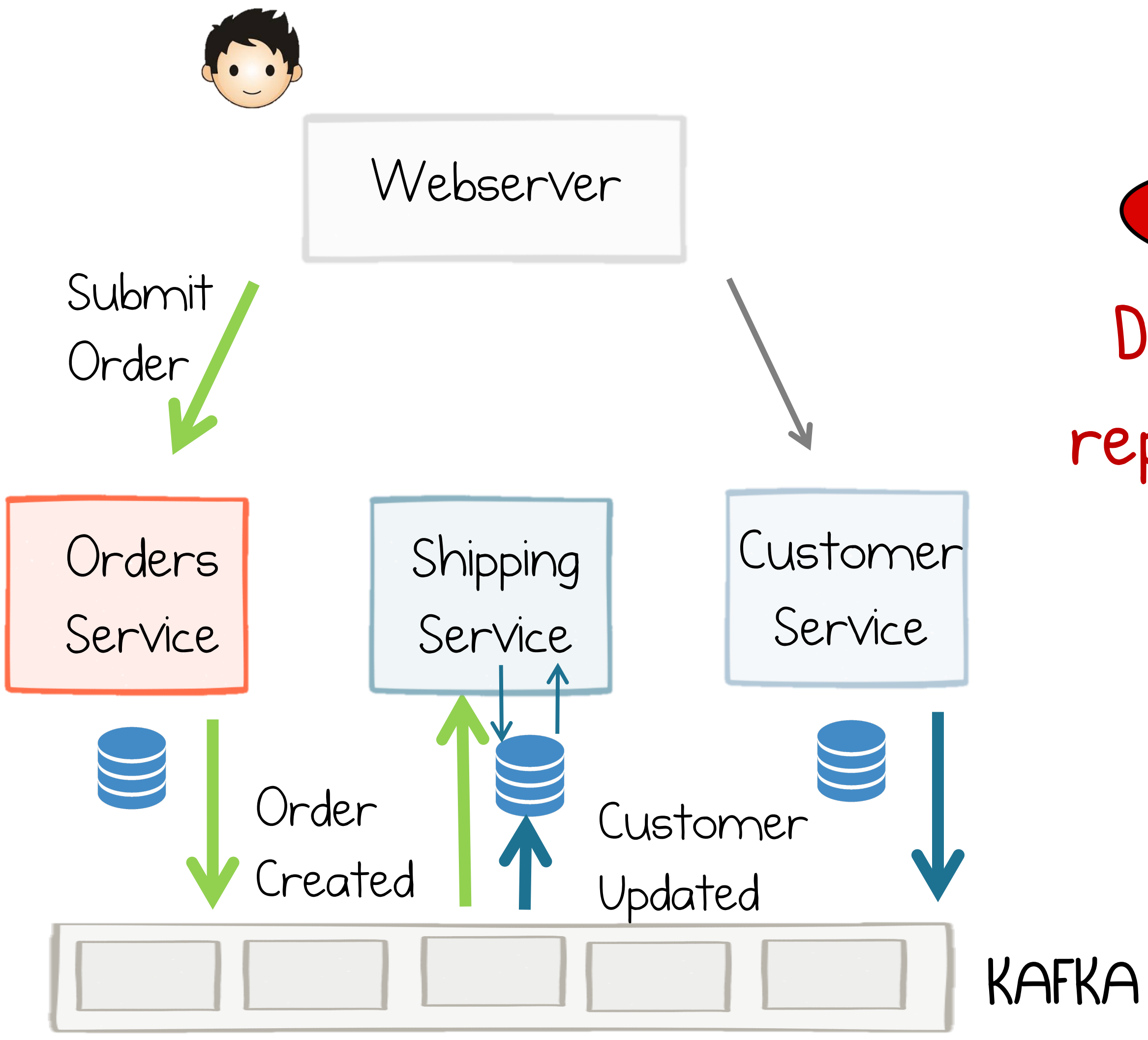


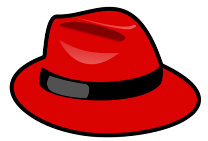
confluent

Pluggability



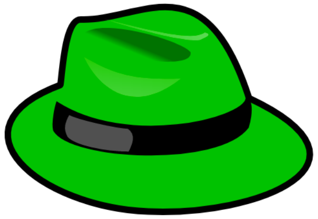
Events for Data Locality



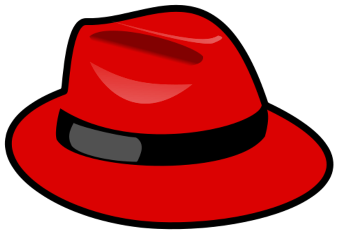


confluent

Events have two hats



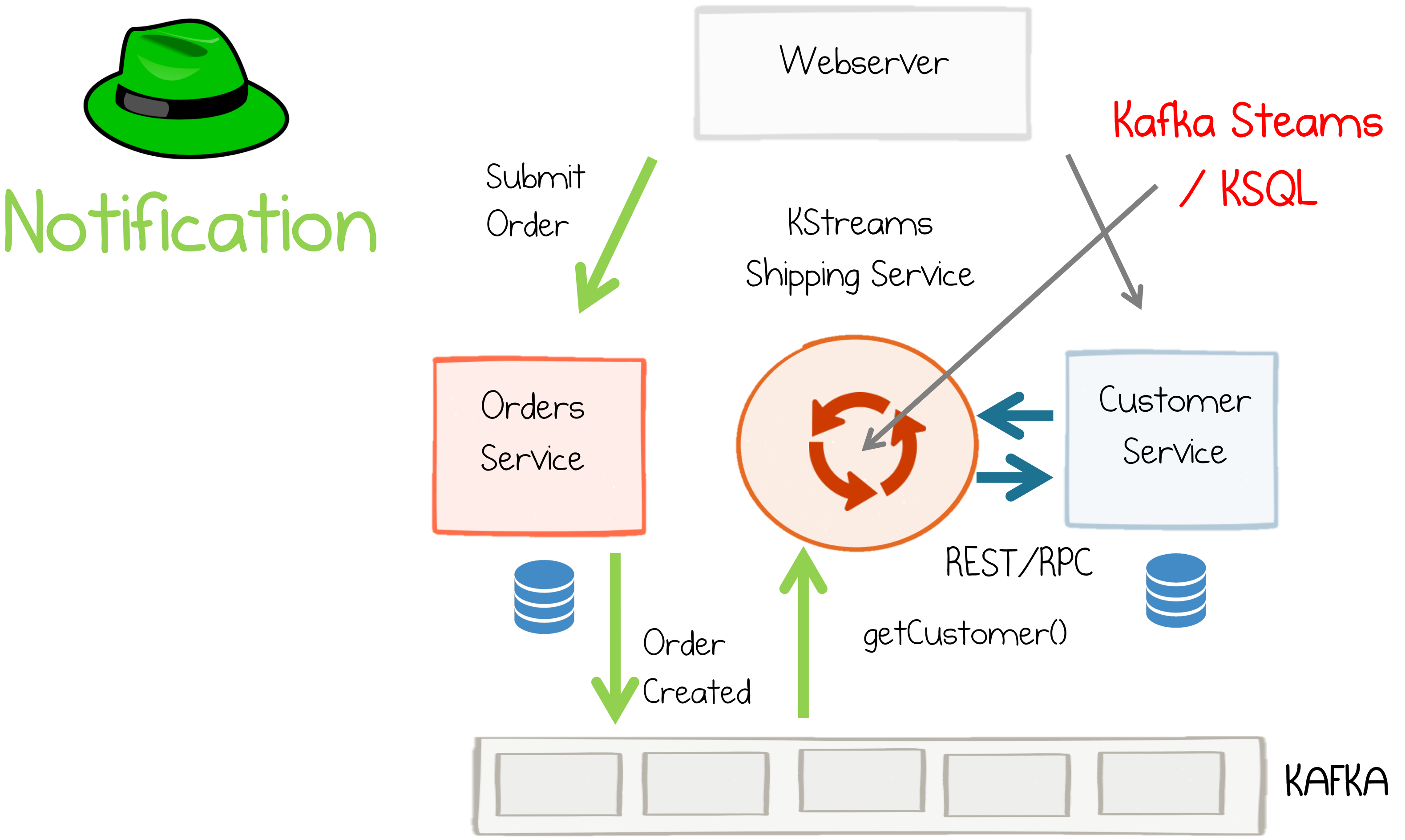
Notification



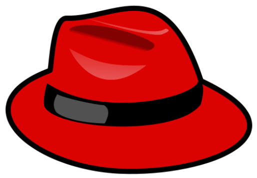
Data replication

Stateless / Stateful Stream Processing Relates to these hats

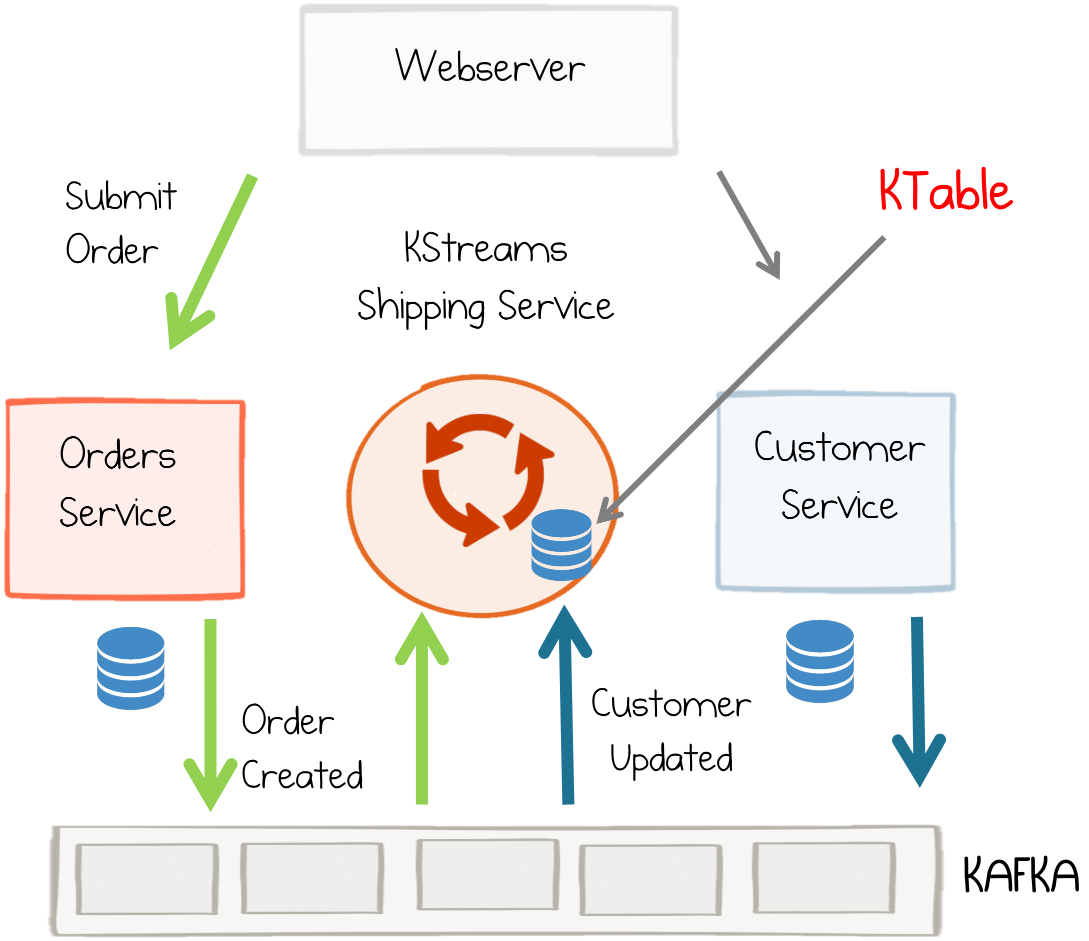
Stateless Stream Processing



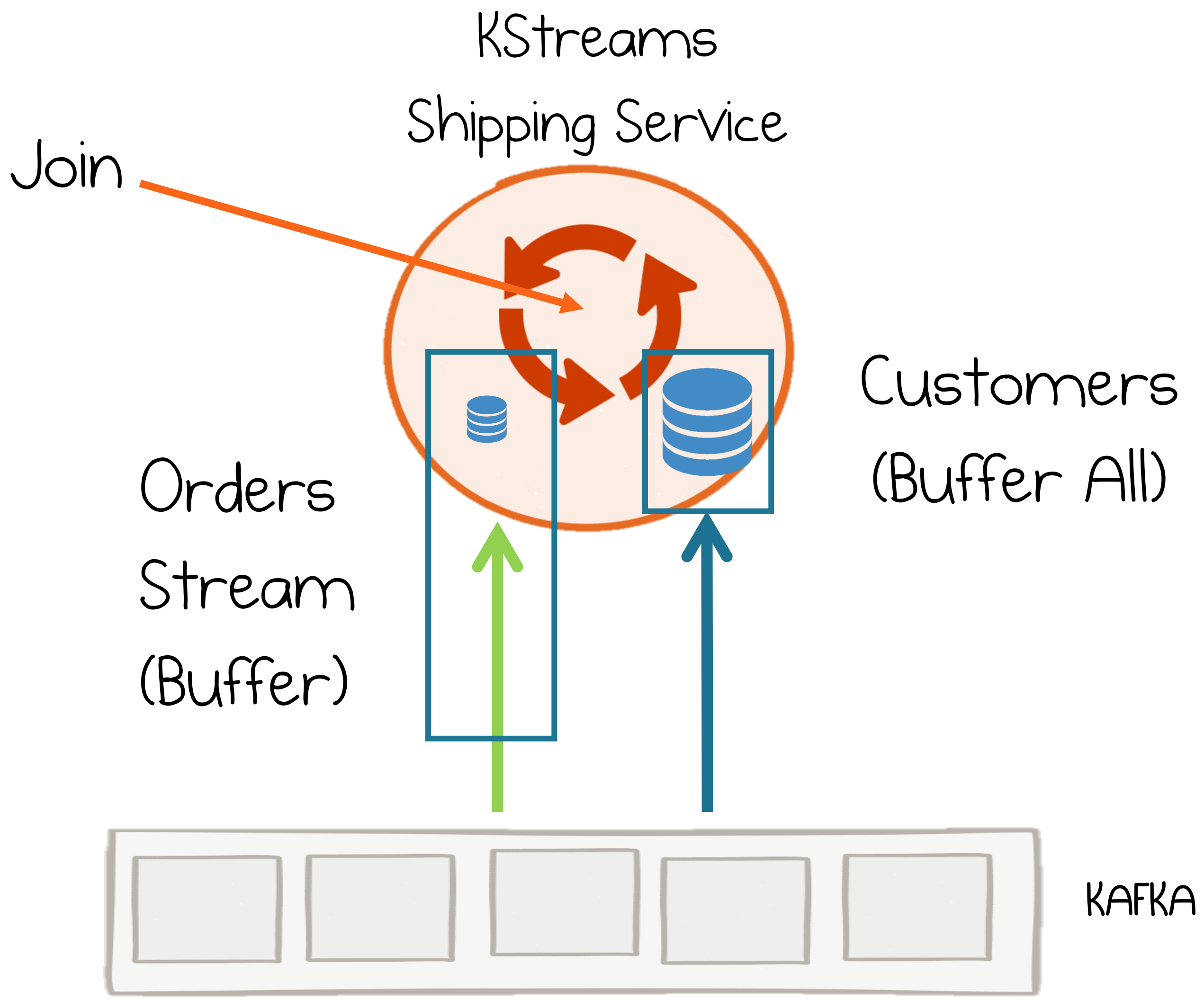
Stateful Stream Processing



Data replication



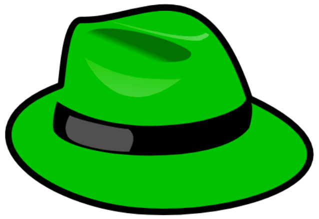
Streams & Tables



KSQL ~ KStreams

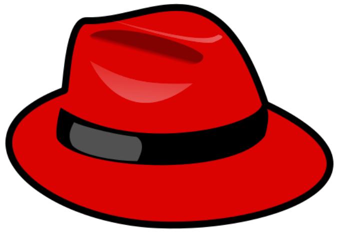
Streaming is about

1. Joining & Operating on Streams



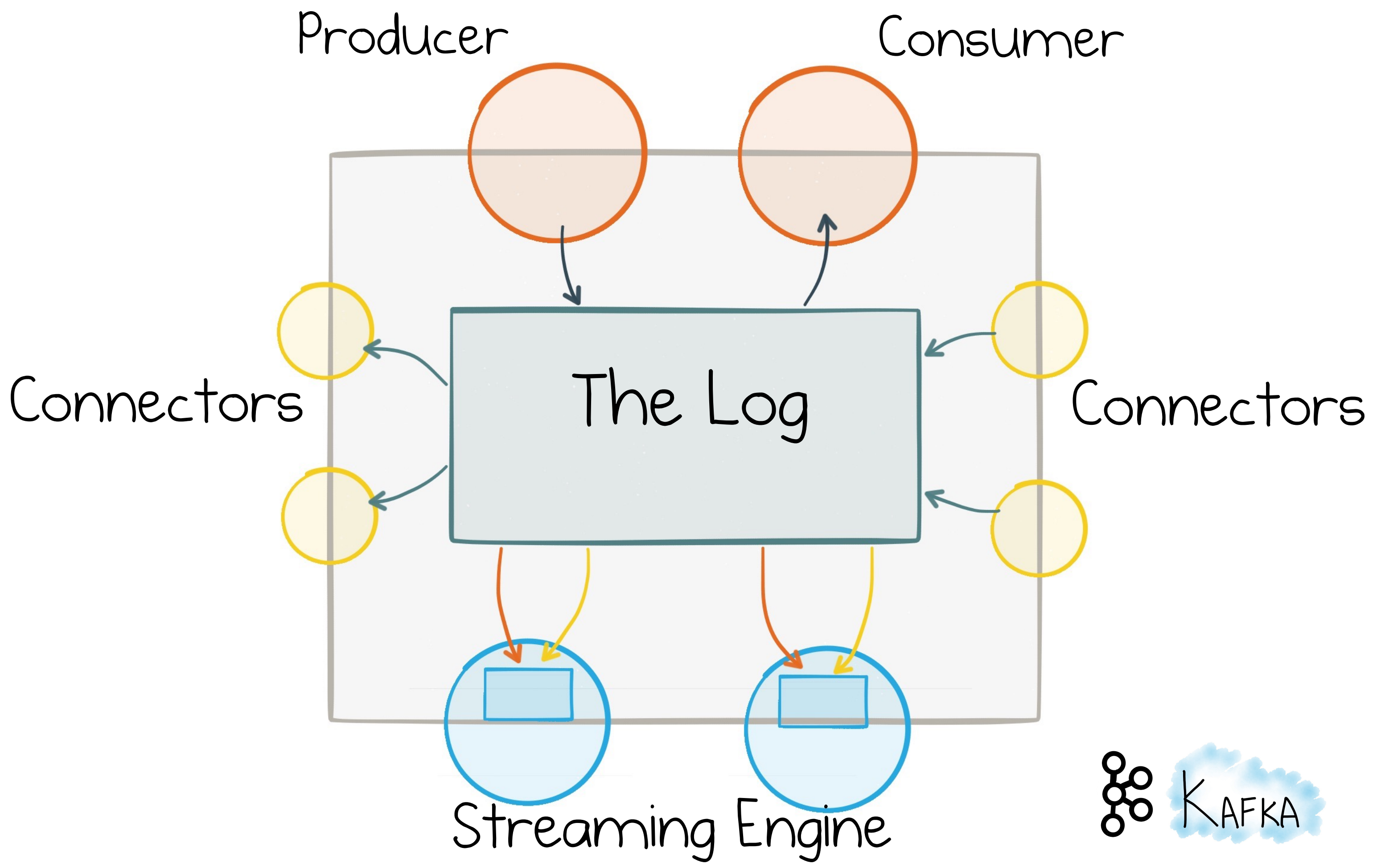
On Notification

2. Joining & Operating on Materialized Tables



Data Replication

Kafka: a Streaming Platform

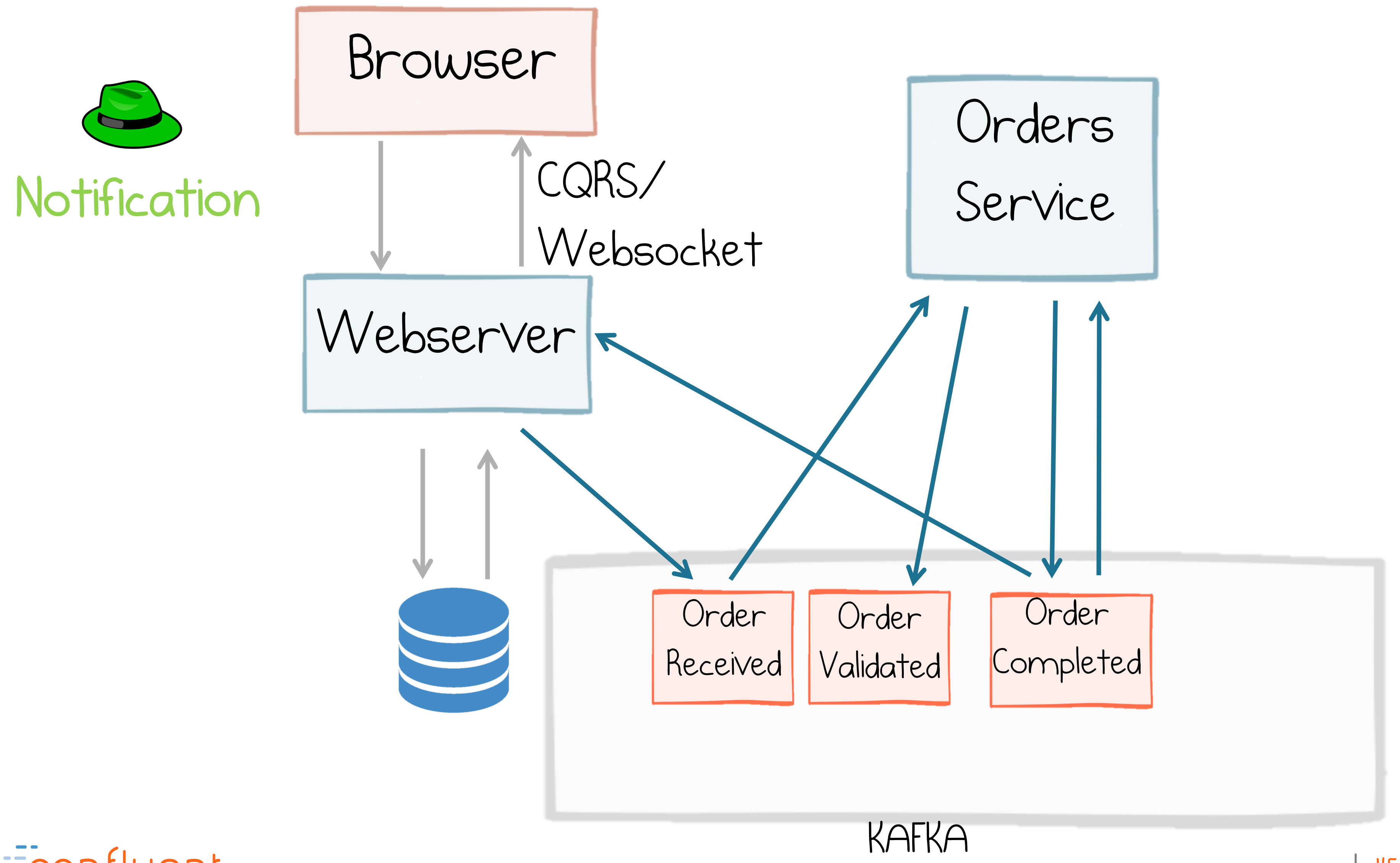


Kafka

8 Steps to Streaming Services

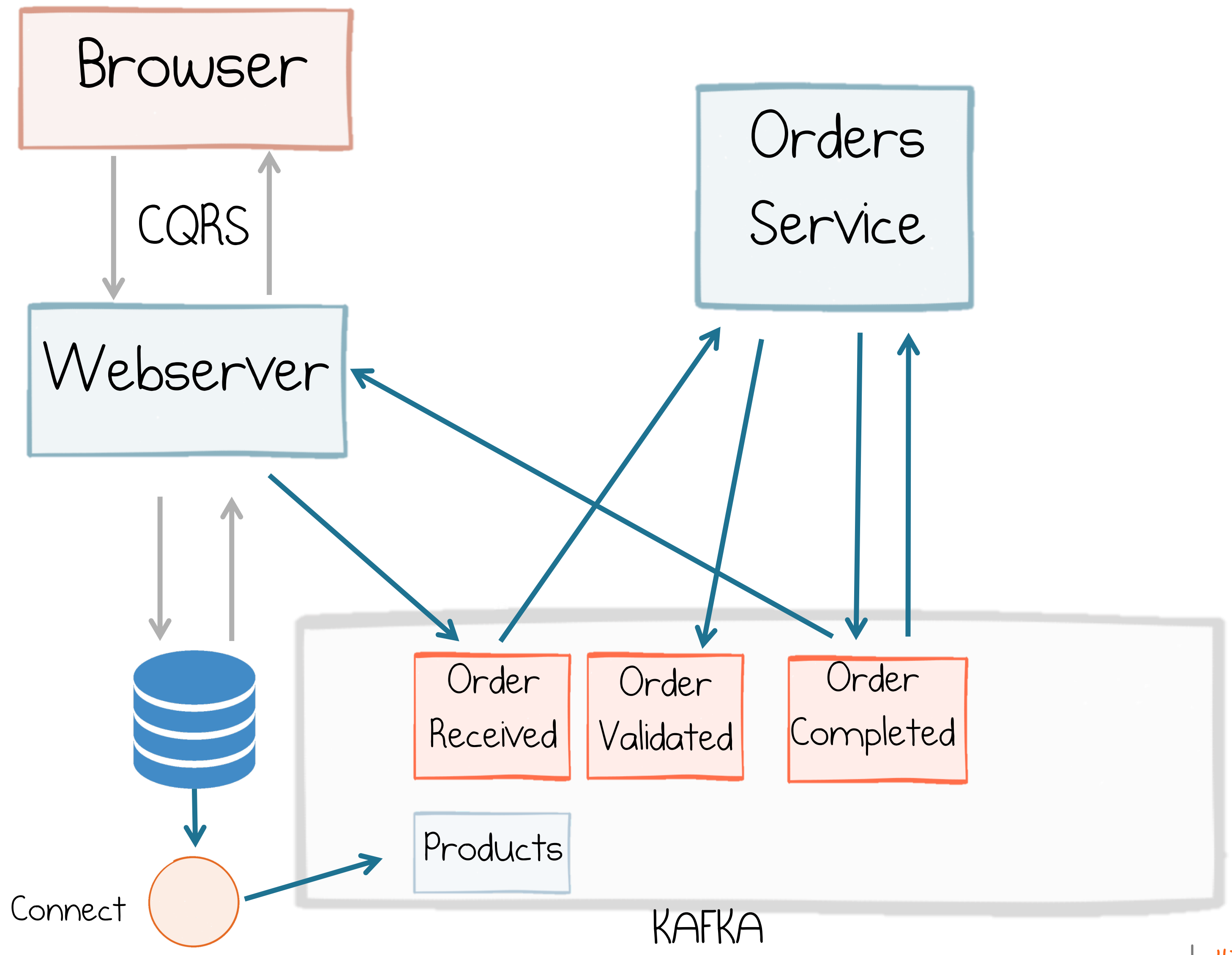
1. Use events to decouple and collaborate

Event Collaboration



2. Use Connect (& CDC) to evolve away from legacy

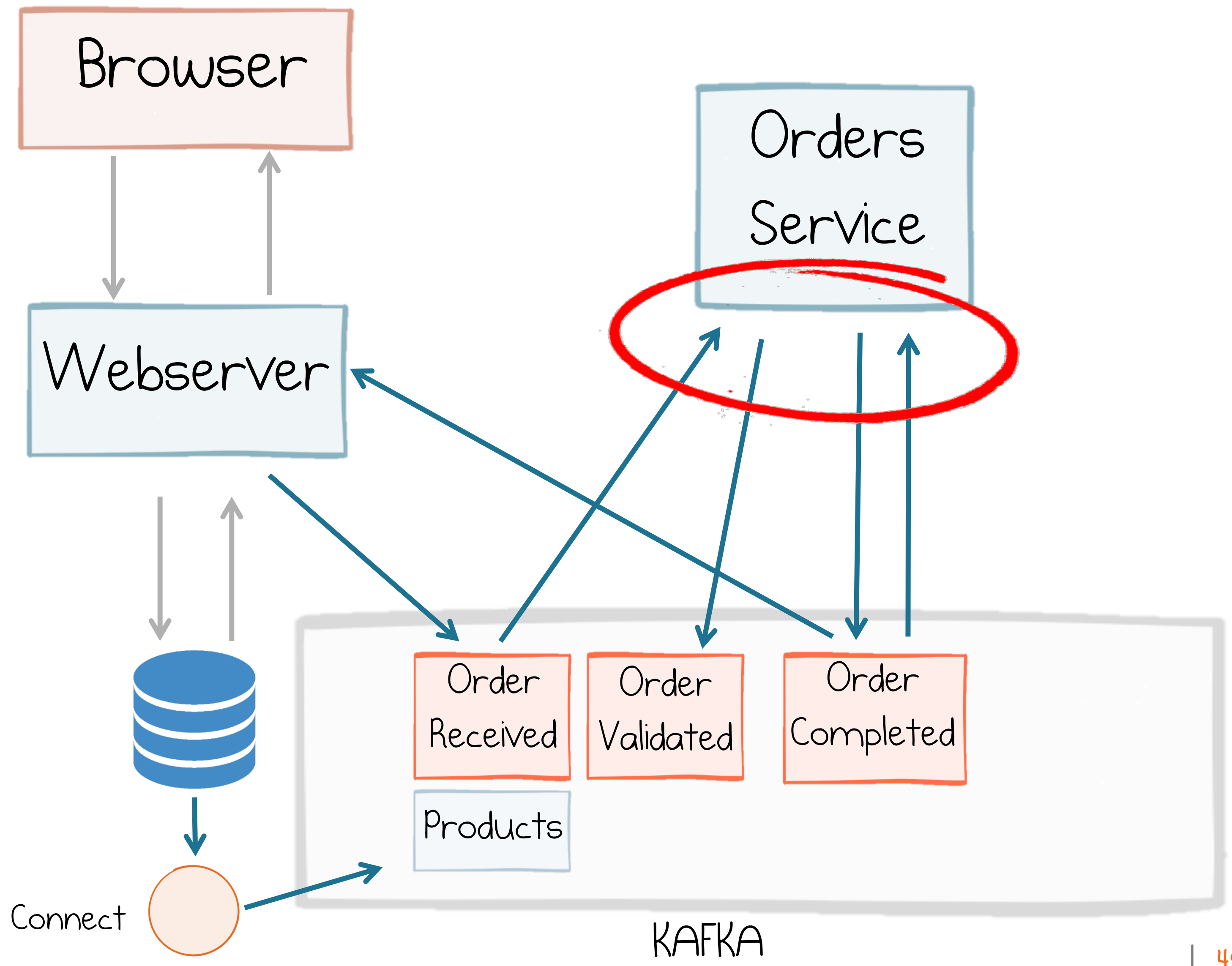
Make Legacy Datasets Available via the Log



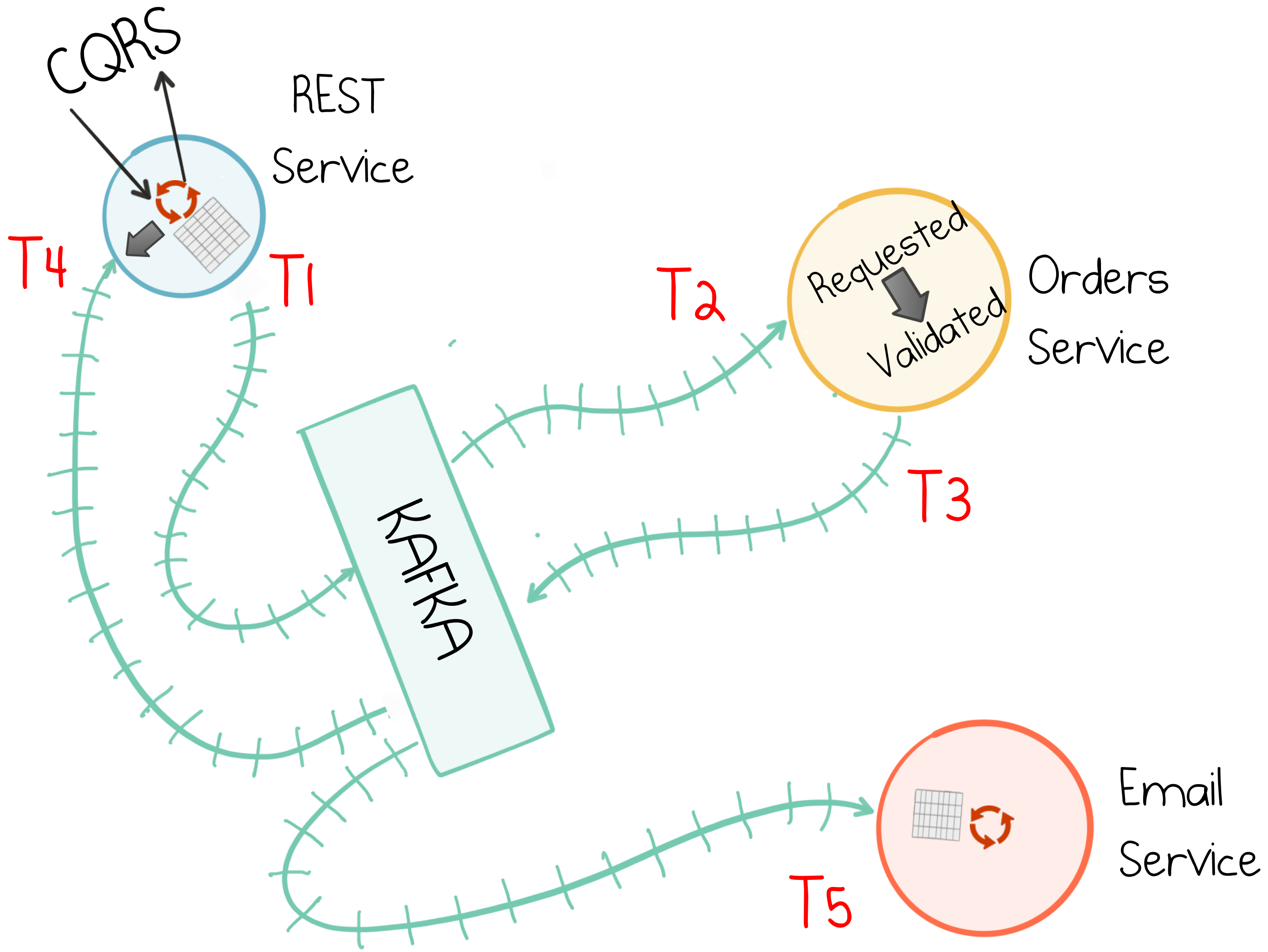
confluent

3. Use the Single Writer Principal

State changes to a topic owned by one service

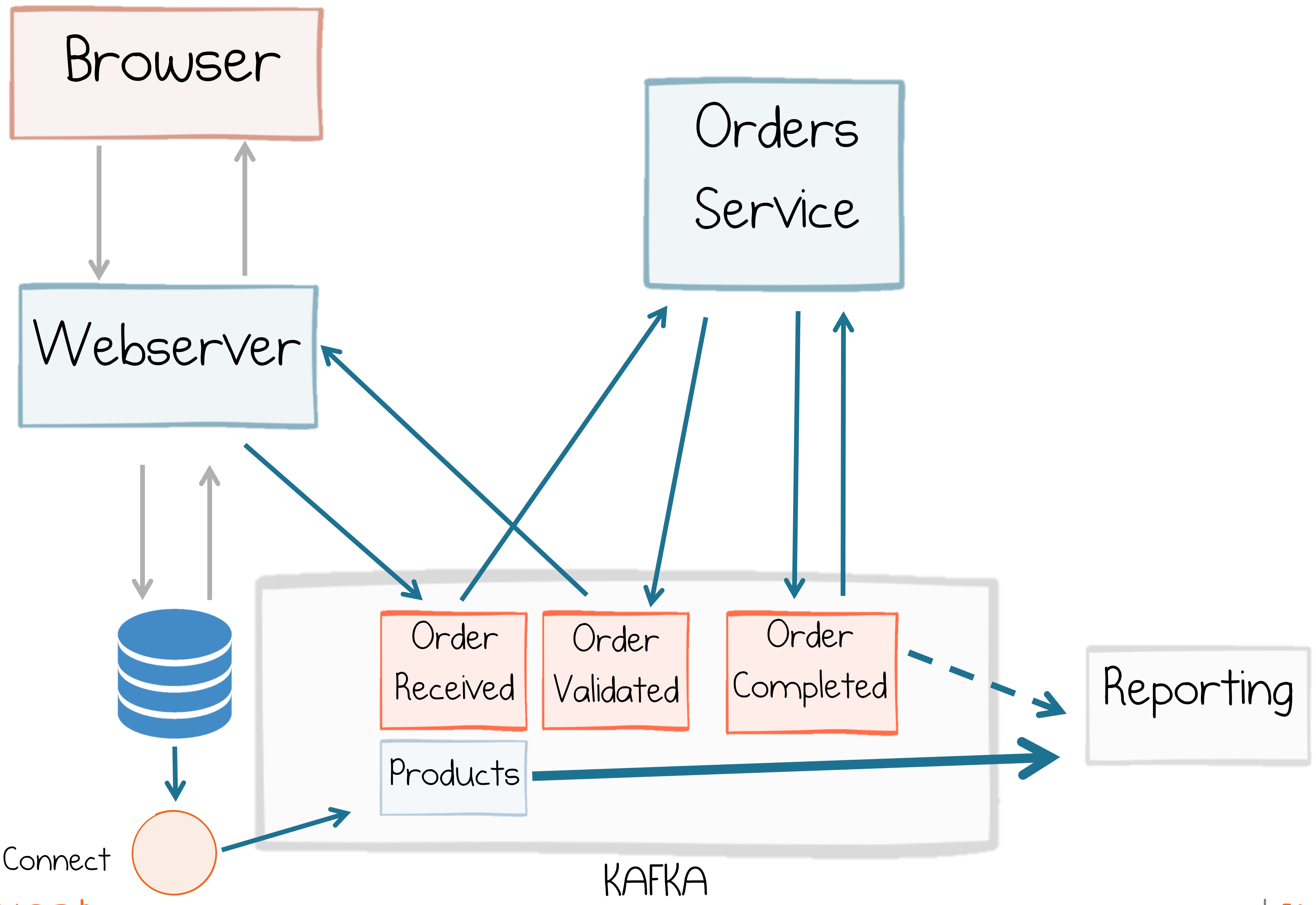


Local consistency points in the absence of Global Consistency

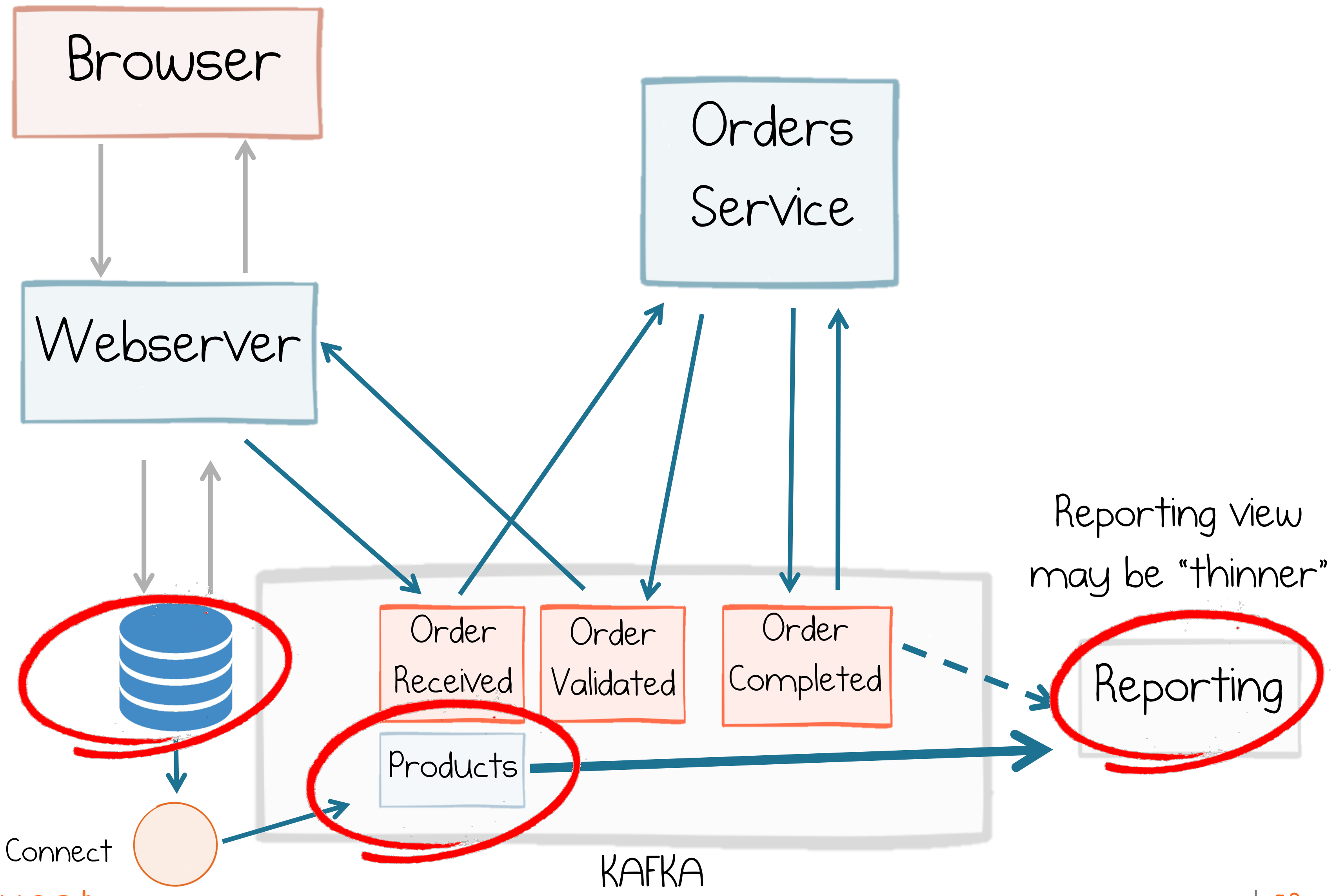


4. Use Kafka as a Shared Source of Truth (Messaging that Remembers)

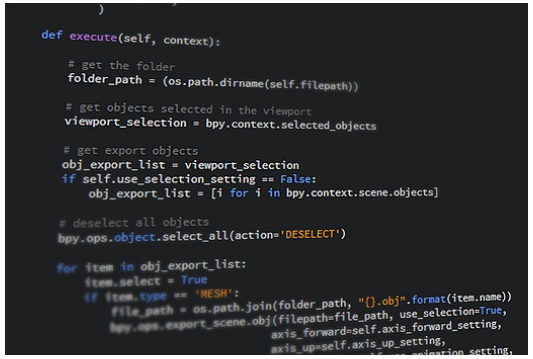
Shared Source of Truth



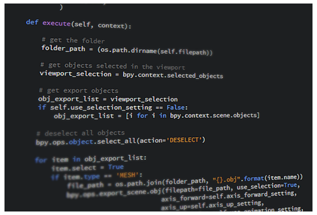
Product Catalogue stored in 3 places



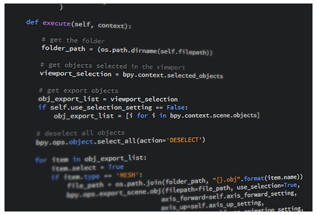
5. Move Data to Code



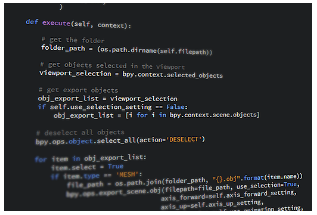
```
        )

    def execute(self, context):

        # get the folder
        folder_path = (os.path.dirname(self.filepath))

        # get objects selected in the viewport
        viewport_selection = bpy.context.selected_objects

        # get export objects
        obj_export_list = viewport_selection
        if self.use_selection_setting == False:
            obj_export_list = [i for i in bpy.context.scene.objects]

        # deselect all objects
        bpy.ops.object.select_all(action='DESELECT')

        for item in obj_export_list:
            item.select = True
            if item.type == 'MESH':
                file_path = os.path.join(folder_path, "{}.obj".format(item.name))
                bpy.ops.export_scene.obj(filepath=file_path, use_selection=True,
                                         axis_forward=self.axis_forward_setting,
                                         axis_up=self.axis_up_setting,
```



```
        )

    def execute(self, context):

        # get the folder
        folder_path = (os.path.dirname(self.filepath))

        # get objects selected in the viewport
        viewport_selection = bpy.context.selected_objects

        # get export objects
        obj_export_list = viewport_selection
        if self.use_selection_setting == False:
            obj_export_list = [i for i in bpy.context.scene.objects]

        # deselect all objects
        bpy.ops.object.select_all(action='DESELECT')

        for item in obj_export_list:
            item.select = True
            if item.type == 'MESH':
                file_path = os.path.join(folder_path, "{}.obj".format(item.name))
                bpy.ops.export_scene.obj(filepath=file_path, use_selection=True,
                                         axis_forward=self.axis_forward_setting,
                                         axis_up=self.axis_up_setting,
```

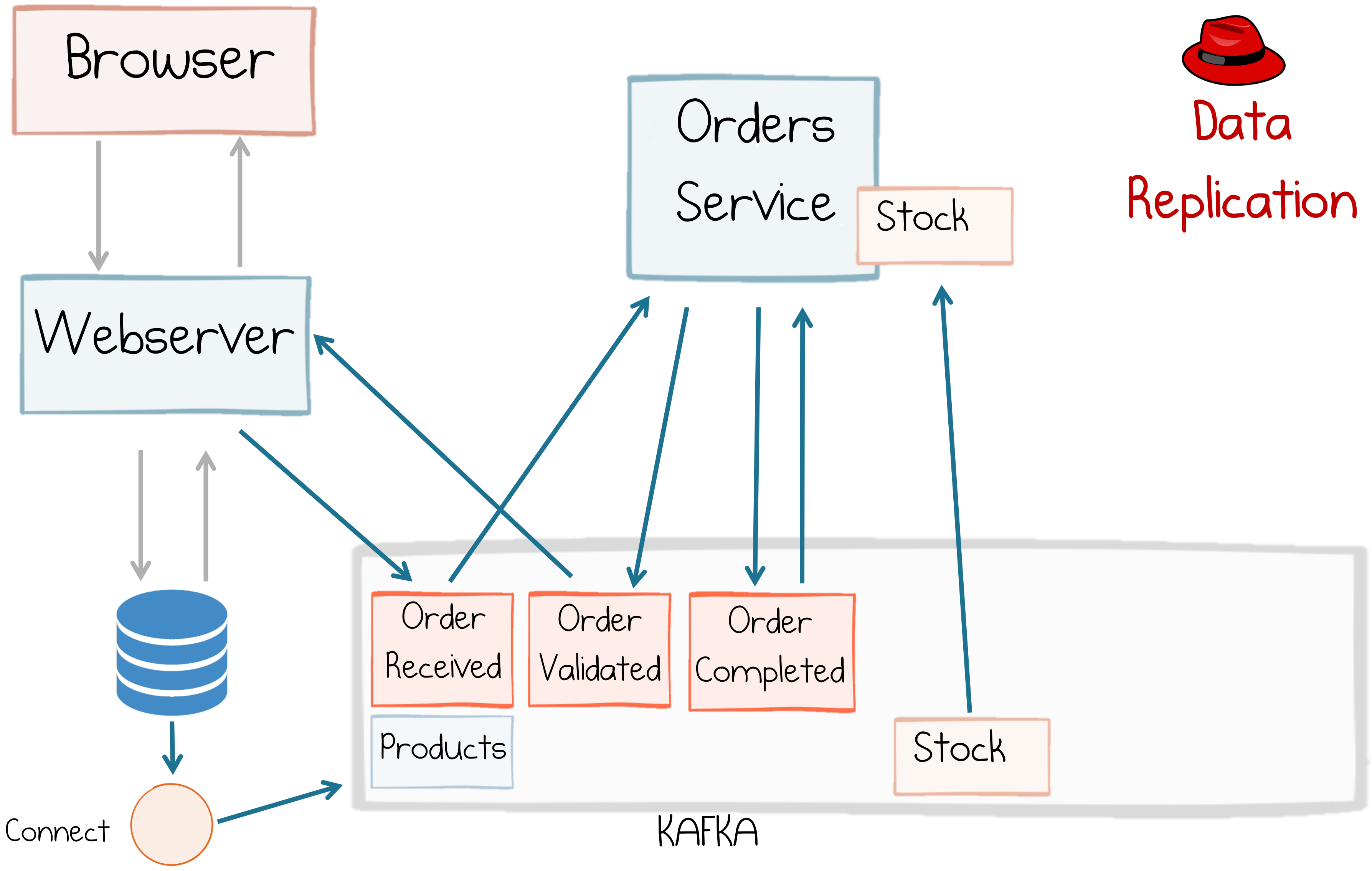


```
        )

    def execute(self, context):

        # get the folder
        folder_path = (os.path.dirname(self.filepath))

        # get objects selected in the viewport
        viewport_selection = bpy.context.selected_objects

        # get export objects
        obj_export_list = viewport_selection
        if self.use_selection_setting == False:
            obj_export_list = [i for i in bpy.context.scene.objects]

        # deselect all objects
        bpy.ops.object.select_all(action='DESELECT')

        for item in obj_export_list:
            item.select = True
            if item.type == 'MESH':
                file_path = os.path.join(folder_path, "{}.obj".format(item.name))
                bpy.ops.export_scene.obj(filepath=file_path, use_selection=True,
                                         axis_forward=self.axis_forward_setting,
                                         axis_up=self.axis_up_setting,
```



```
        )

    def execute(self, context):

        # get the folder
        folder_path = (os.path.dirname(self.filepath))

        # get objects selected in the viewport
        viewport_selection = bpy.context.selected_objects

        # get export objects
        obj_export_list = viewport_selection
        if self.use_selection_setting == False:
            obj_export_list = [i for i in bpy.context.scene.objects]

        # deselect all objects
        bpy.ops.object.select_all(action='DESELECT')

        for item in obj_export_list:
            item.select = True
            if item.type == 'MESH':
                file_path = os.path.join(folder_path, "{}.obj".format(item.name))
                bpy.ops.export_scene.obj(filepath=file_path, use_selection=True,
                                         axis_forward=self.axis_forward_setting,
                                         axis_up=self.axis_up_setting,
```

Materialize Stock 'View' Inside Service

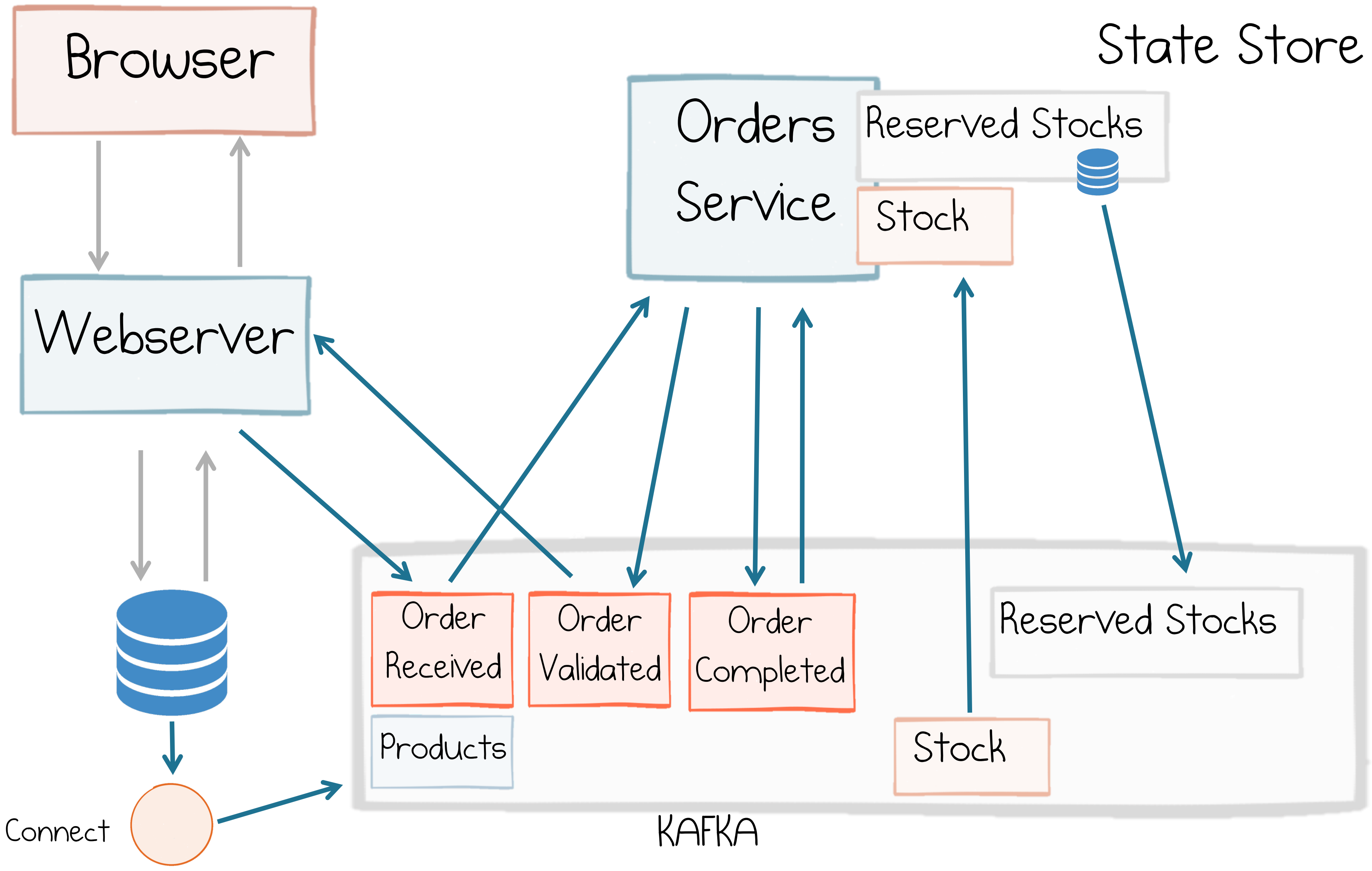


Kafka has several features for reducing the need to move data on startup

- Standby Replicas
- Disk Checkpoints
- Compacted topics

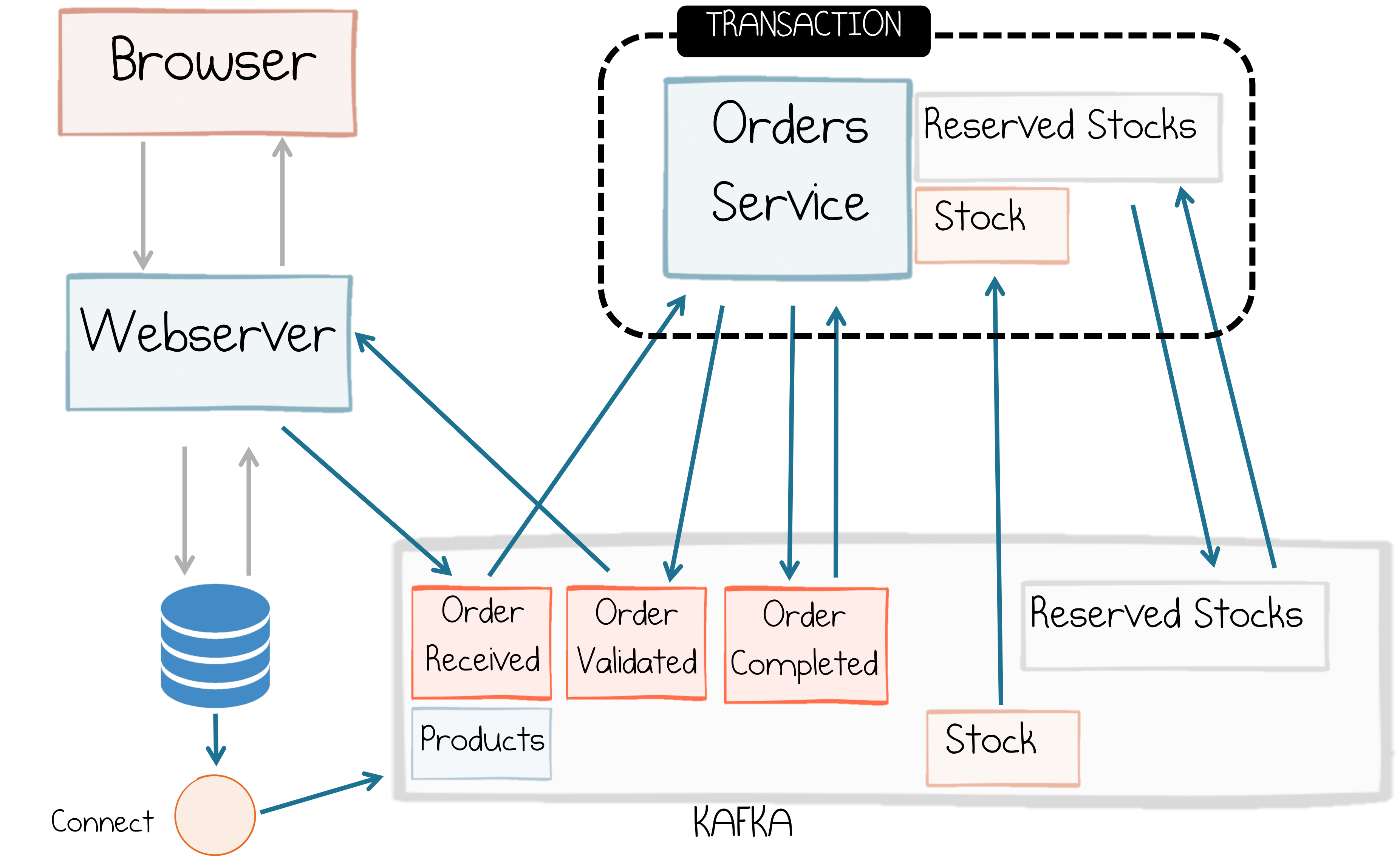
6. Write to State Stores, just like a local 'database', backed up in Kafka

State stores behave like local databases



7. Use Transactions to tie All Interactions Together

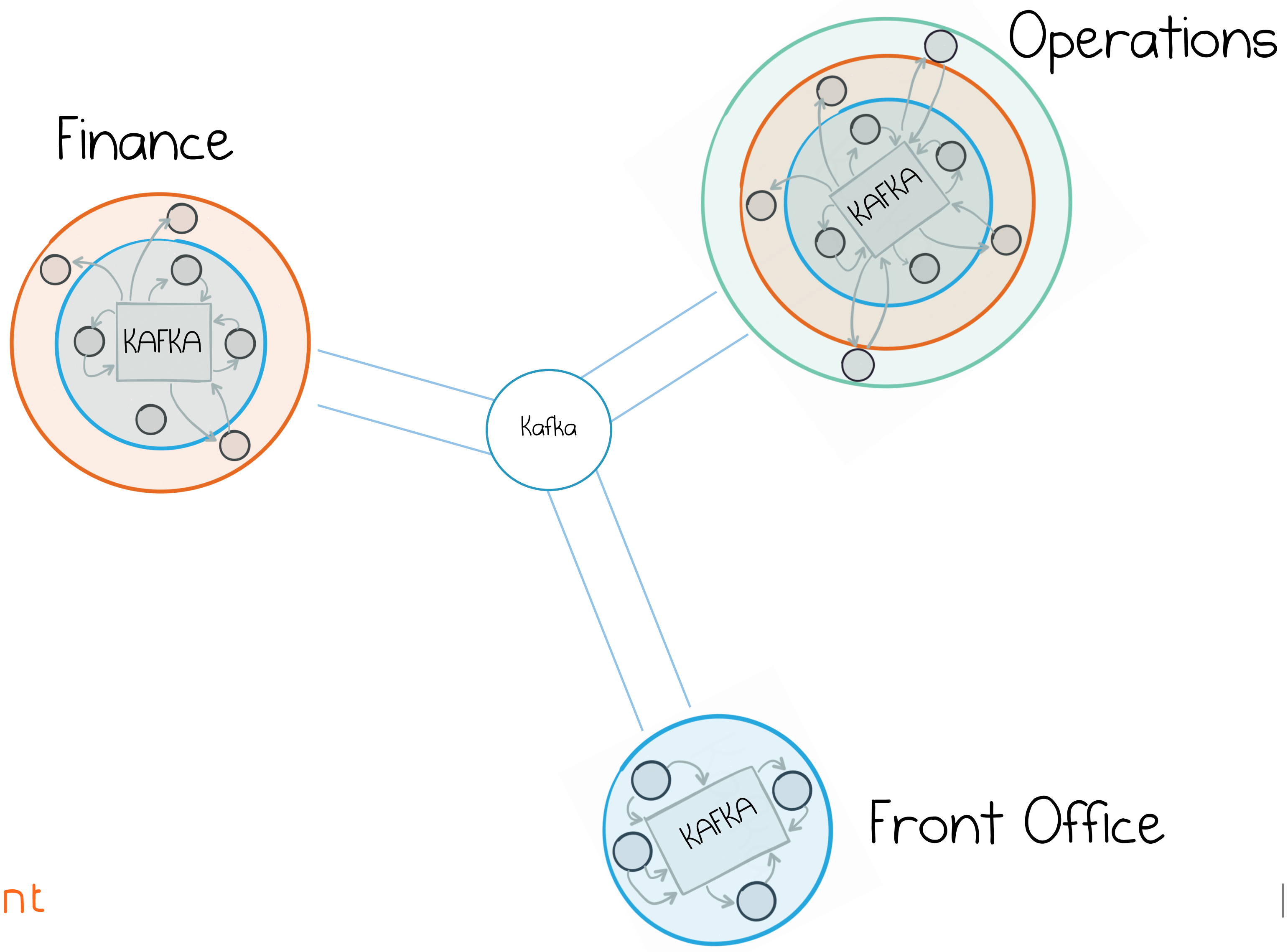
Transactions



confluent

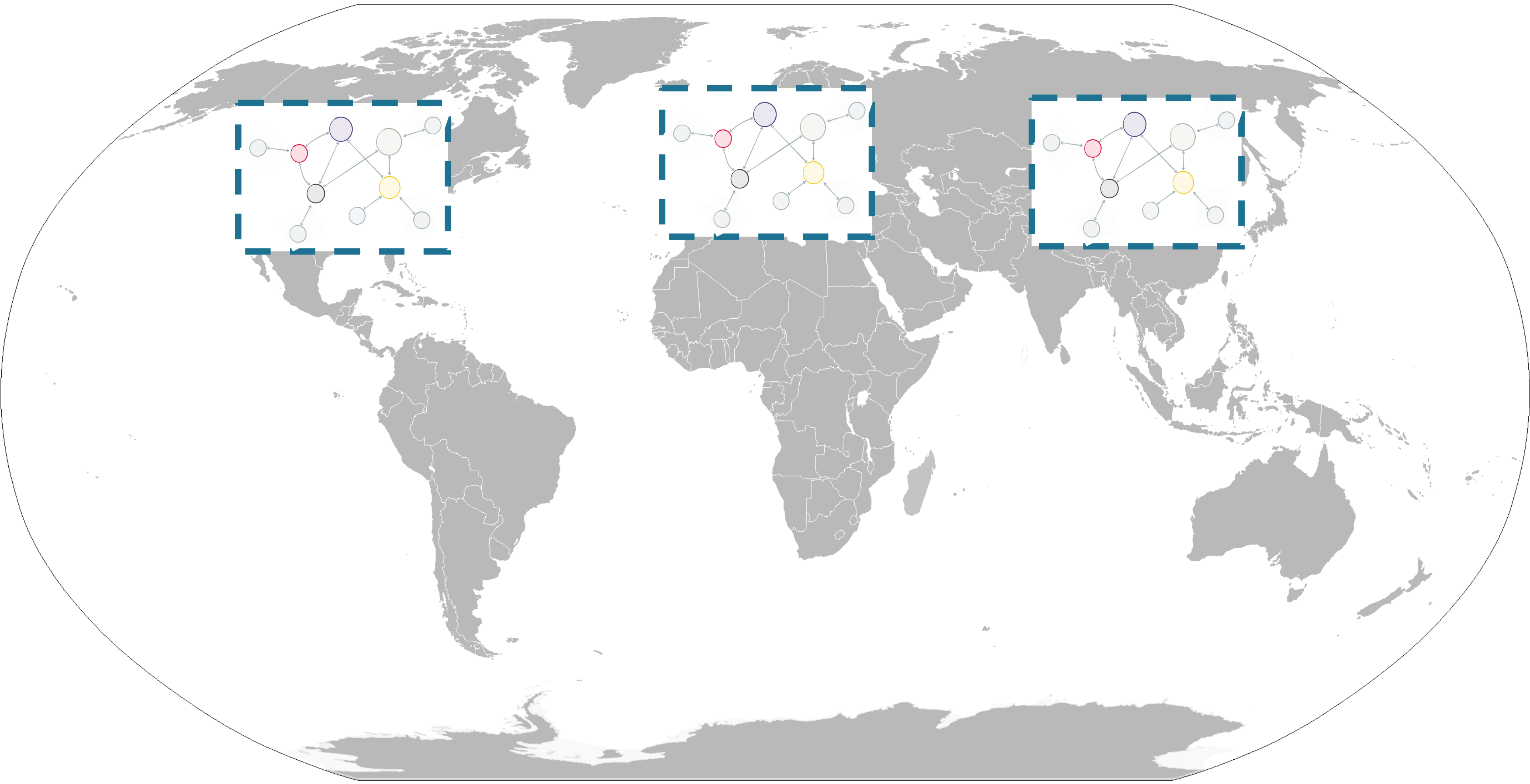
8. Evolve and Grow

Tiered Contexts



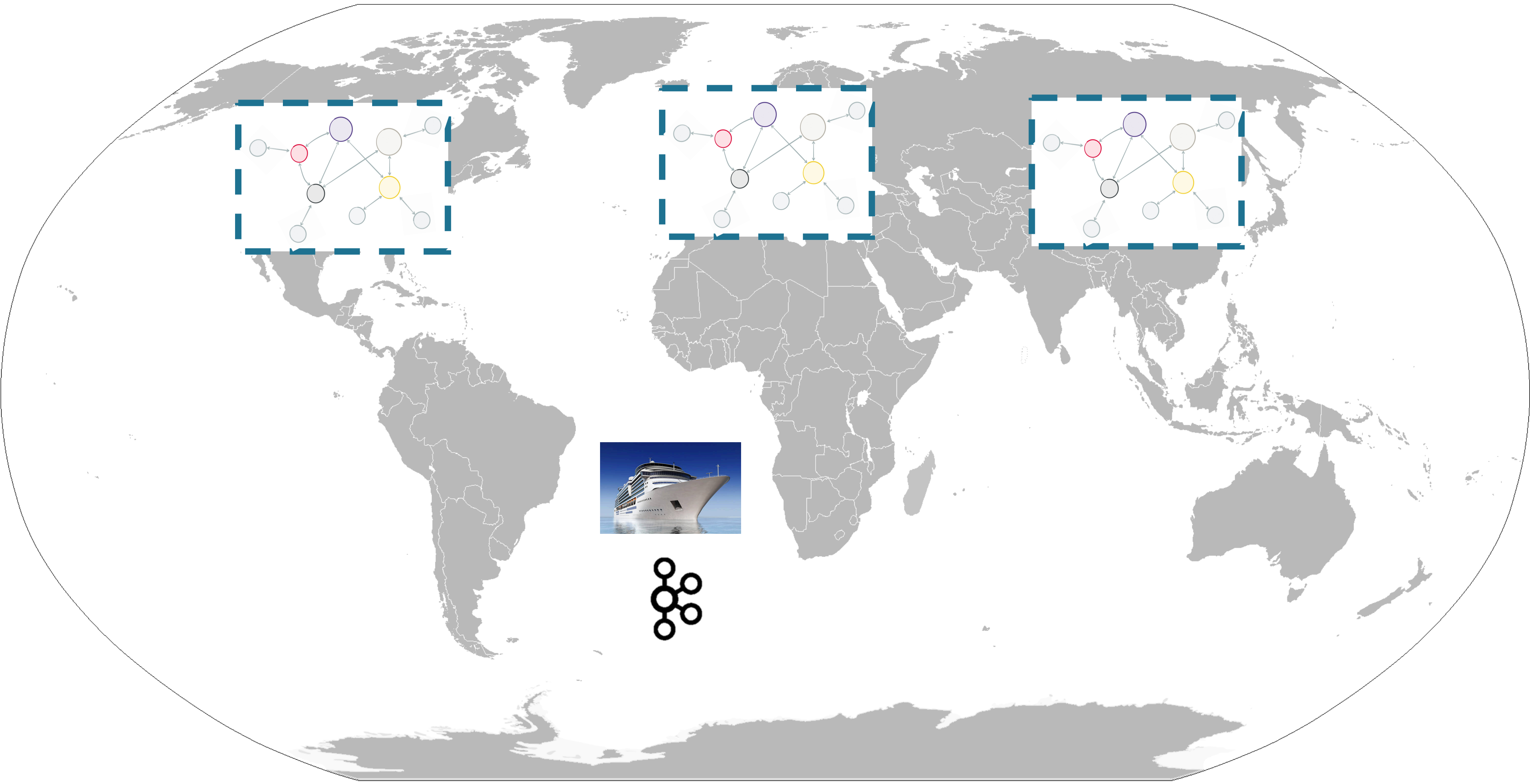
confluent

Span regions or clouds



confluent

Handle Disconnectedness



confluent

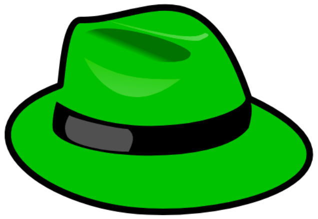
So…

Optimize for complexity vs optimize for scale

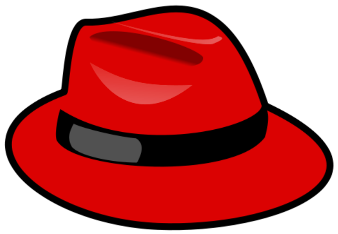
Event Driven Architectures

Stream Processing

Events provide the key to evolutionary architectures

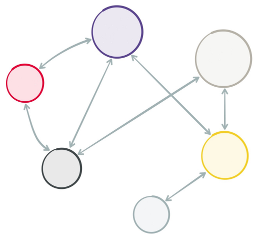


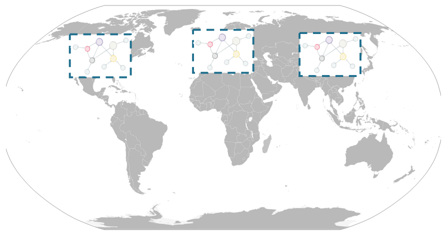
Notification



Data replication

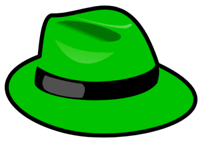
Spectrum of use cases

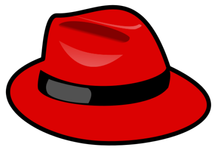




Finer Grained,
Collaborative,
Connected

Courser Grained,
Non-collaborative,
Disconnected





Notification

Data Replication

confluent

Streaming is the toolset for dealing with events at scale

Event Driven Services

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

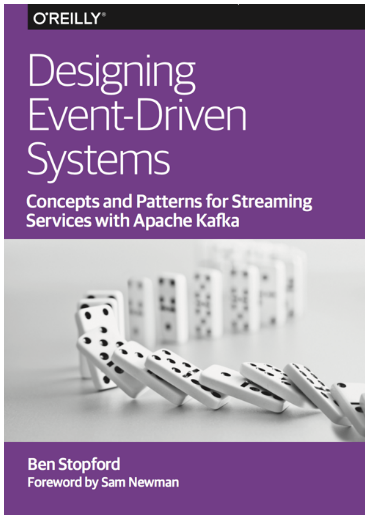
Find out more

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

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

Code: http://bit.ly/kafka-microservice-examples

Twitter: @benstopford


O'REILLY
Designing Event-Driven Systems
Concepts and Patterns for Streaming Services with Apache Kafka
Ben Stopford
Foreword by Sam Newman