zalando

A MATERIALIZATION ENGINE FOR DATA INTEGRATION WITH FLINK

MIHAIL VIERU

13-09-2017

AGENDA

- Microservices Architecture
- Data Integration Challenge
- Materialization Engine
- Flink Backend
- Stream Compaction
- Advantages over Legacy Approach

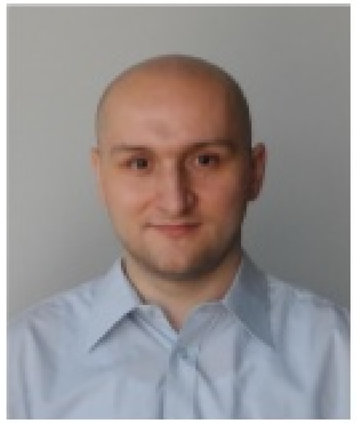
ABOUT ME

Mihail Vieru

Big Data Engineer

Team "Flux" Stream Processing

Data Engineering Dept.



Europe's leading online fashion platform

15 countries

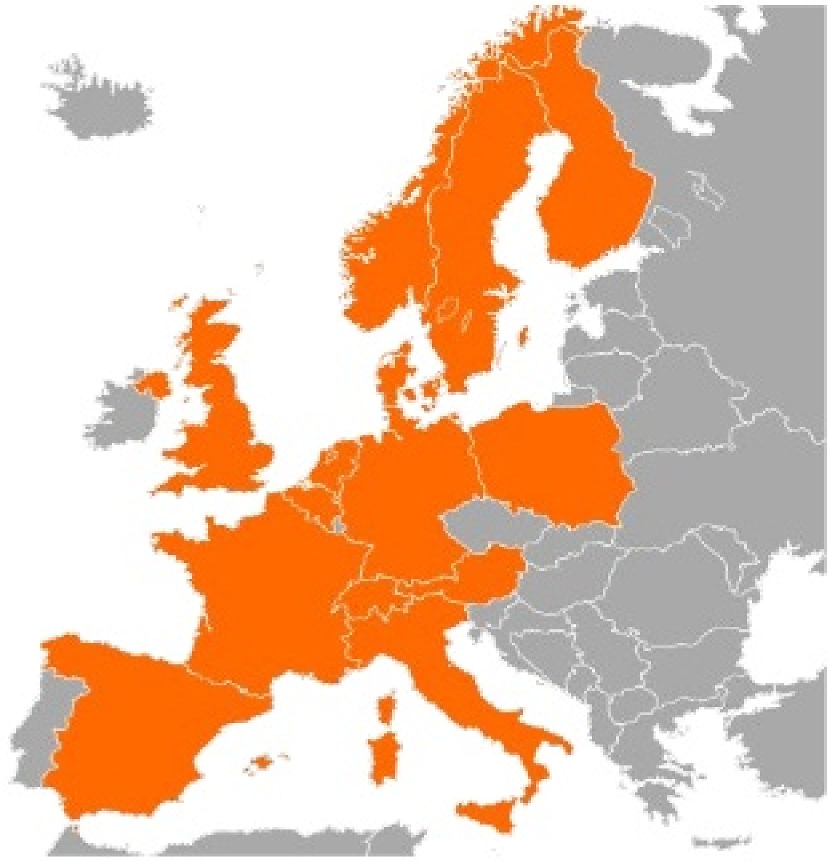
~21 million active customers

~3.6 billion € revenue 2016

250,000+ products

2,000 brands

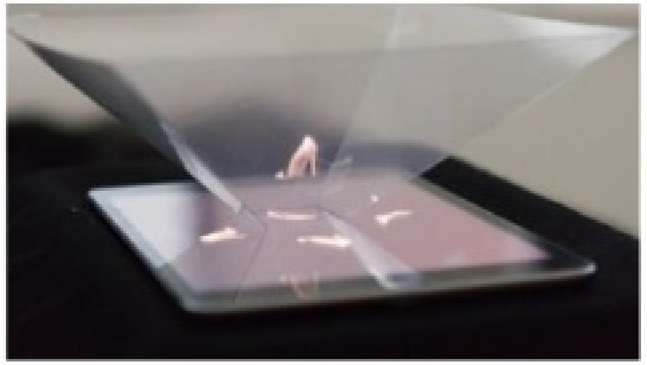
13,000+ employees in Europe



zalando

WE ARE CONSTANTLY INNOVATING TECHNOLOGY

HOME-BREWED, CUTTING-EDGE & SCALABLE technology solutions



help our brand to **WIN ONLINE**

~ 1,800 employees from

77 nations

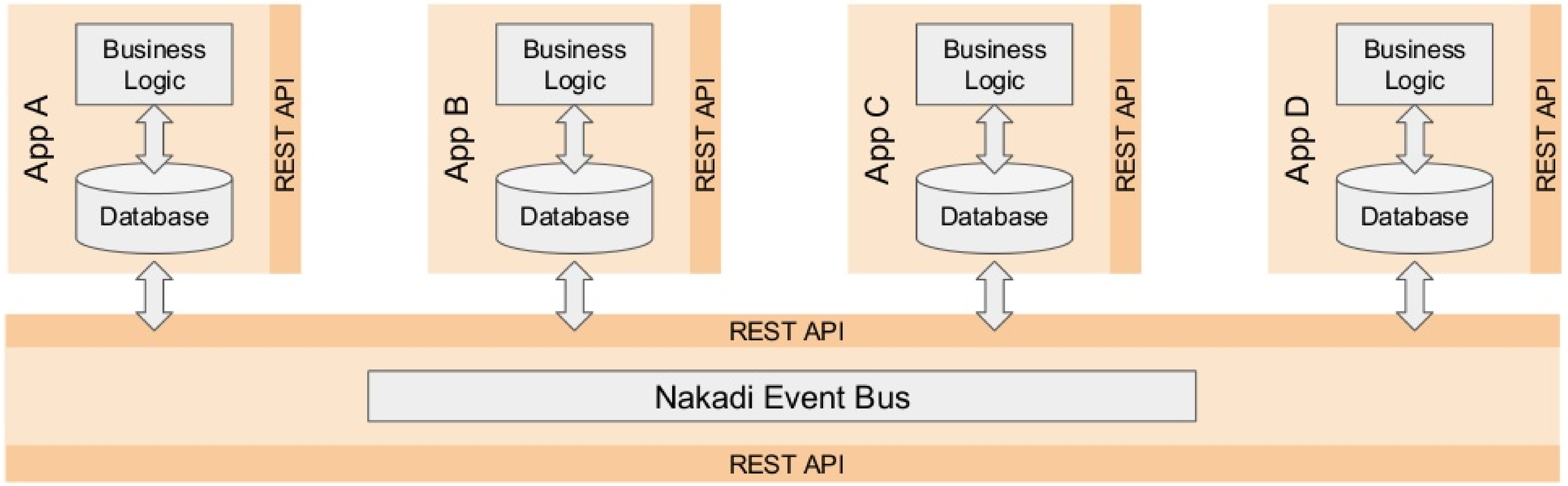
6 tech locations + HQs in Berlin

tech.zalando.com

zalando

MICROSERVICES ARCHITECTURE

MICROSERVICES ARCHITECTURE



Everything runs on Amazon Web Services

zalando

NAKADI - CENTRAL EVENT BUS

A distributed event bus that implements a RESTful API abstraction over Kafka-like queues.

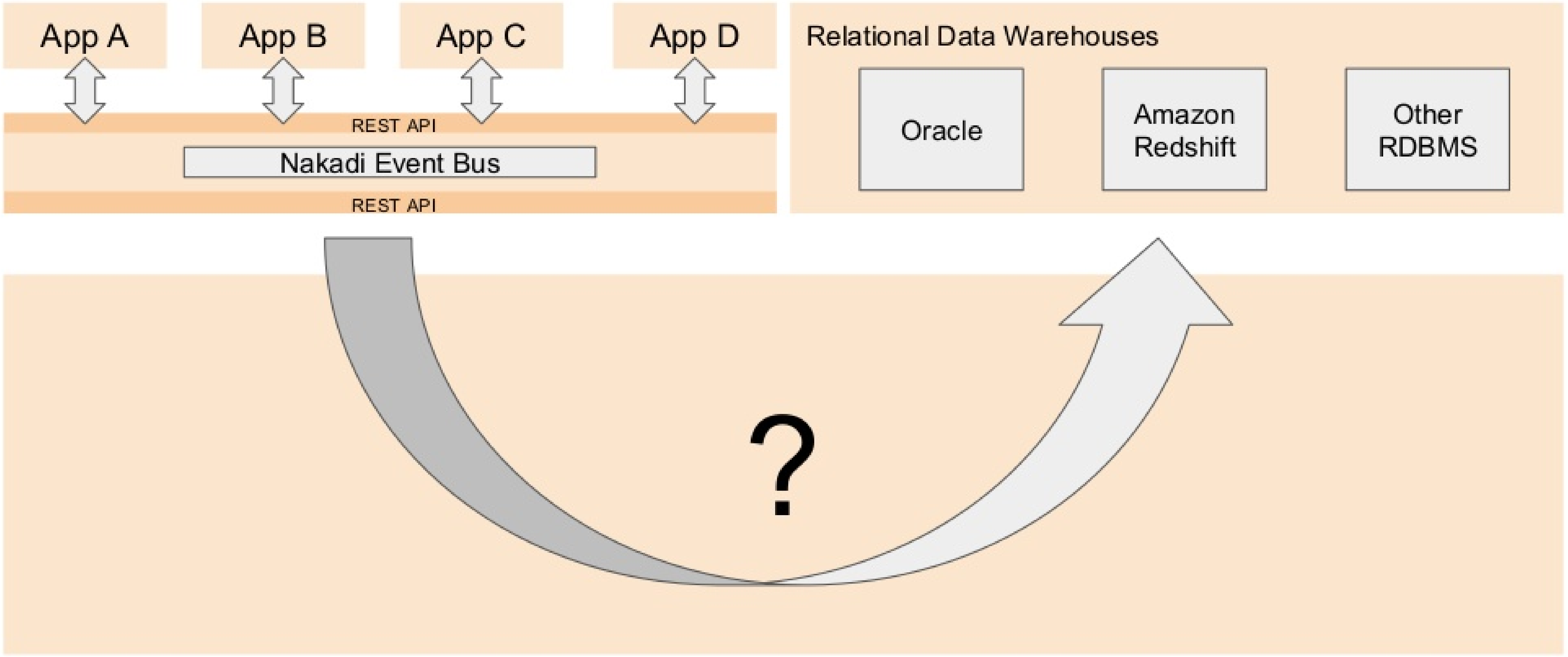
- 1000s of Kafka topics
- high variance in event structure, size & throughput
- consumer can specify partition offsets and batch size in the GET request

https://github.com/zalando/nakadi

zalando

DATA INTEGRATION CHALLENGE

DATA INTEGRATION CHALLENGE



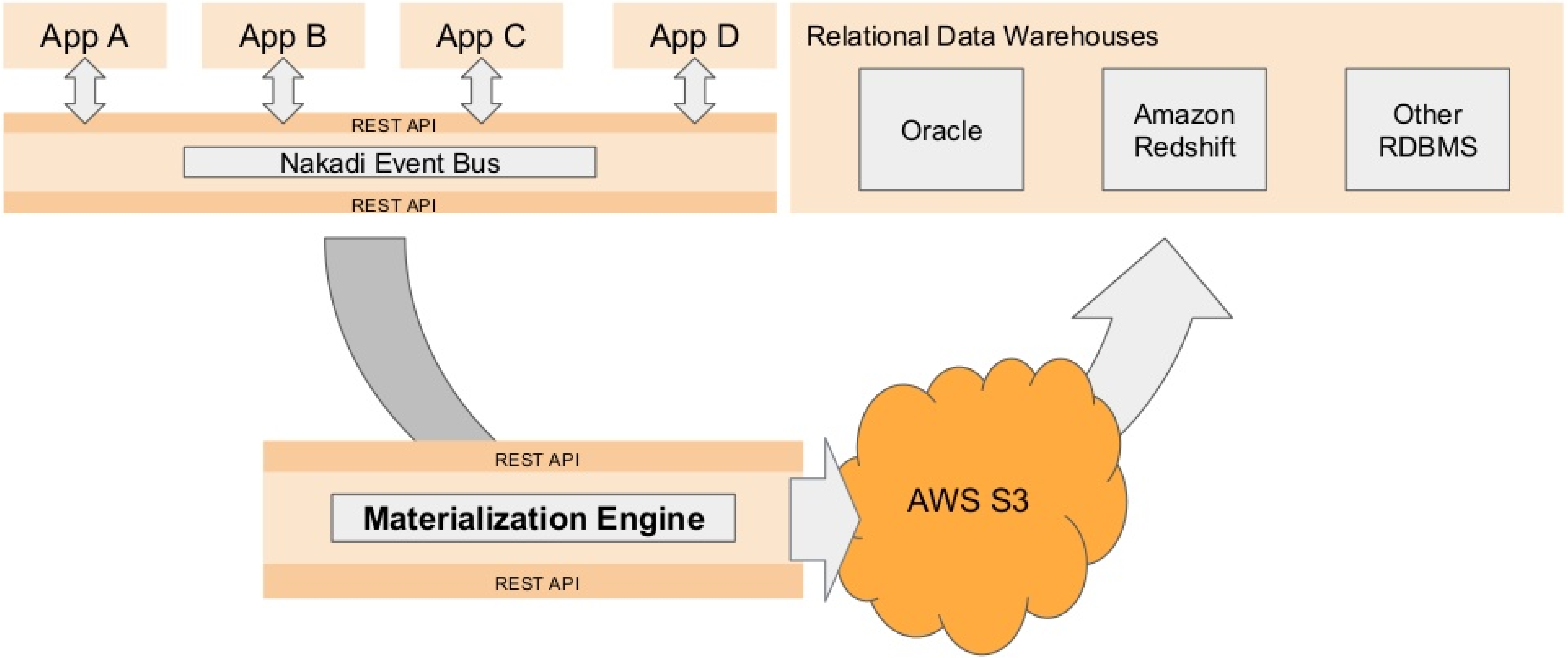
REQUIREMENTS

Goal: Consume data streams in a relational database friendly way

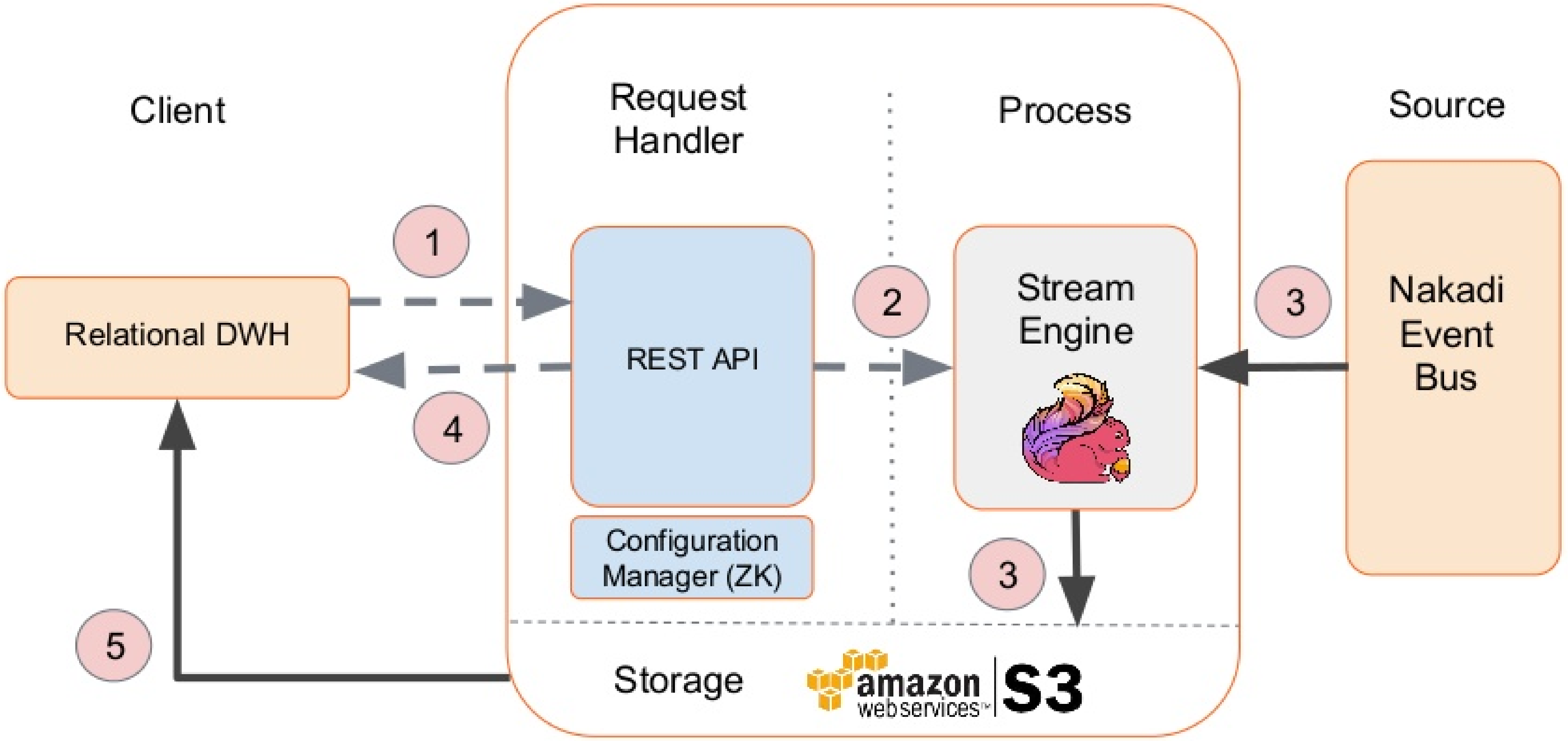
- Materialize data from Nakadi event bus into cloud storage
- Transform complex JSON events into easily ingestible CSV flat files, incl. the flattening of arrays
- Relieve load on the (monolithic) data warehouse by compacting event streams according to event properties

MATERIALIZATION ENGINE

DATA INTEGRATION CHALLENGE



MATERIALIZATION ENGINE



zalando

MATERIALIZATION ENGINE

- Materialization API as another abstraction layer over Flink’s REST API: deploy, monitor, control jobs
- Configuration Manager (ZooKeeper) stores per Nakadi topic:
    - JSON-to-CSV mapping
    - partitioning key
    - ordering key (for compaction)

FLINK BACKEND

FLINK BACKEND: STREAM TO BATCH

- Short-lived stream-to-batch jobs: 1 Job / Call
- Stoppable Nakadi Stream Source
- Modified BucketingSink for S3 Frankfurt region
  - Writes files to TaskManagers' attached EBS storage
  - Moves them to persistent S3 storage
  - Circumvents HADOOP-13324 of Flink's S3 Connector
- Batch processing using a Streaming API

FLINK BACKEND: ACCUMULATORS

- Flink jobs expose progress through accumulators
    - # Nakadi consumers finished
    - # events read
    - # files delivered to S3
- Materialization API periodically queries Flink's REST API to measure progress.
  Issues job stop request upon completion

FLINK BACKEND: CLUSTER

- Flink in standalone mode inside Docker containers on AWS EC2 t2.large instances
- Cluster specifics:
  - Overprovisioning of TaskManagers/ TaskSlots
  - UpScaling on TaskSlots via CloudWatch

STREAM COMPACTION

STREAM COMPACTION: MOTIVATION

- Relinquish DWH resources by reducing the size of the data to ingest
- Can be applied to events which represent changes to the same resource, i.e. having the same partitioning key, but different ordering keys:

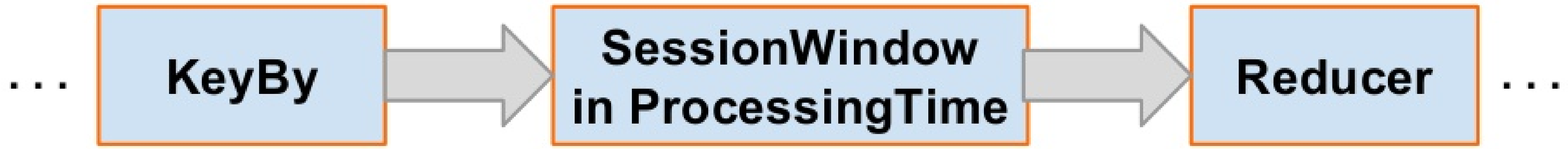
```
{                                           {
   "article_id":     123,                      "article_id":     123,
   "brand":          "Nike",                   "brand":          "Nike",
   "model":          "Air Max",                "model":          "Air Max",
   "version":        1,                        "version":        2,
   "available_qty":  50,                       "available_qty":  30,
}                                           }
```

zalando

STREAM COMPACTION: IMPLEMENTATION

Deploy Flink jobs with config as parameter from Request Handler.

Compact the stream according to the partitioning and ordering keys.



Compaction rates up to 70%

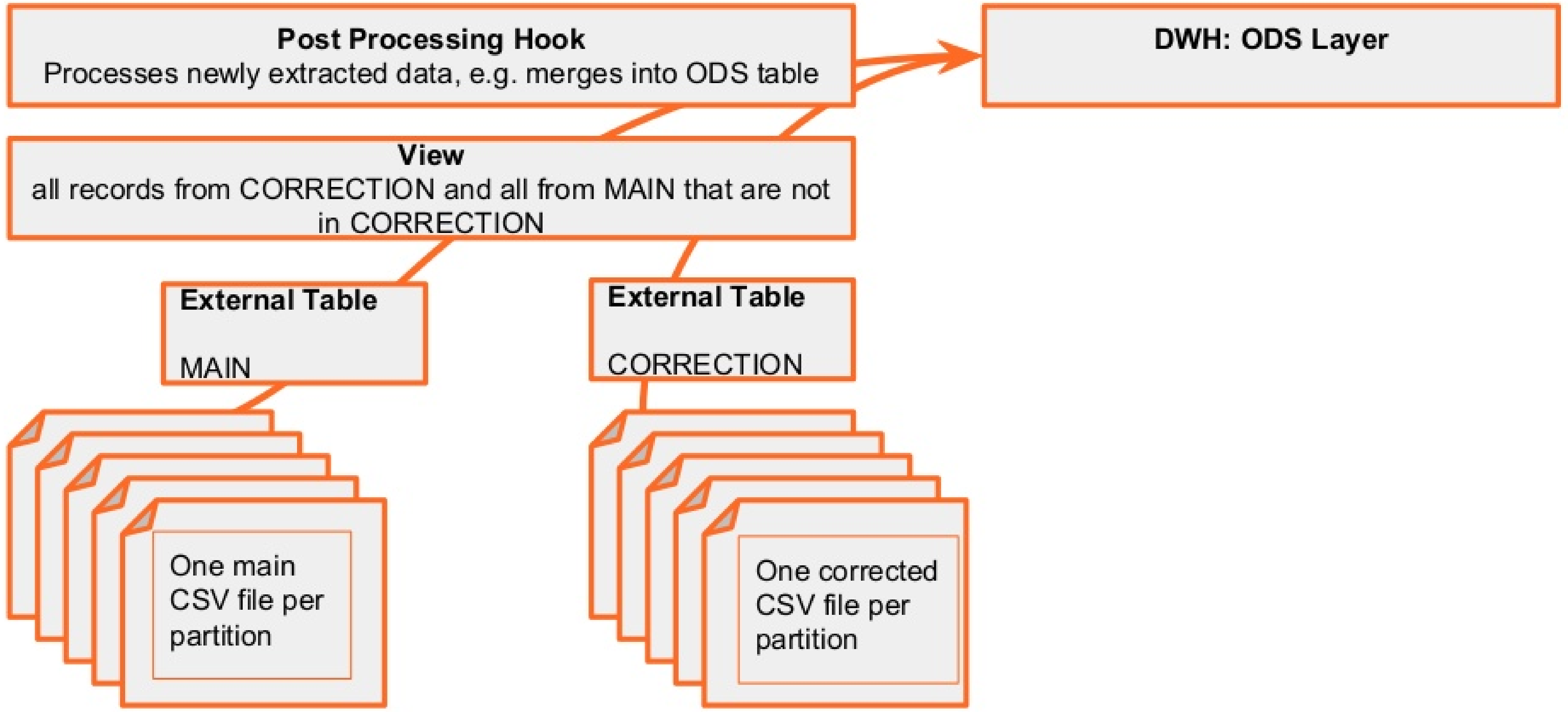
zalando

STREAM COMPACTION: OUT-OF-ORDER EVENTS

Out-of-order events = Events which were received not in the order we expected, e.g. higher `version` value first

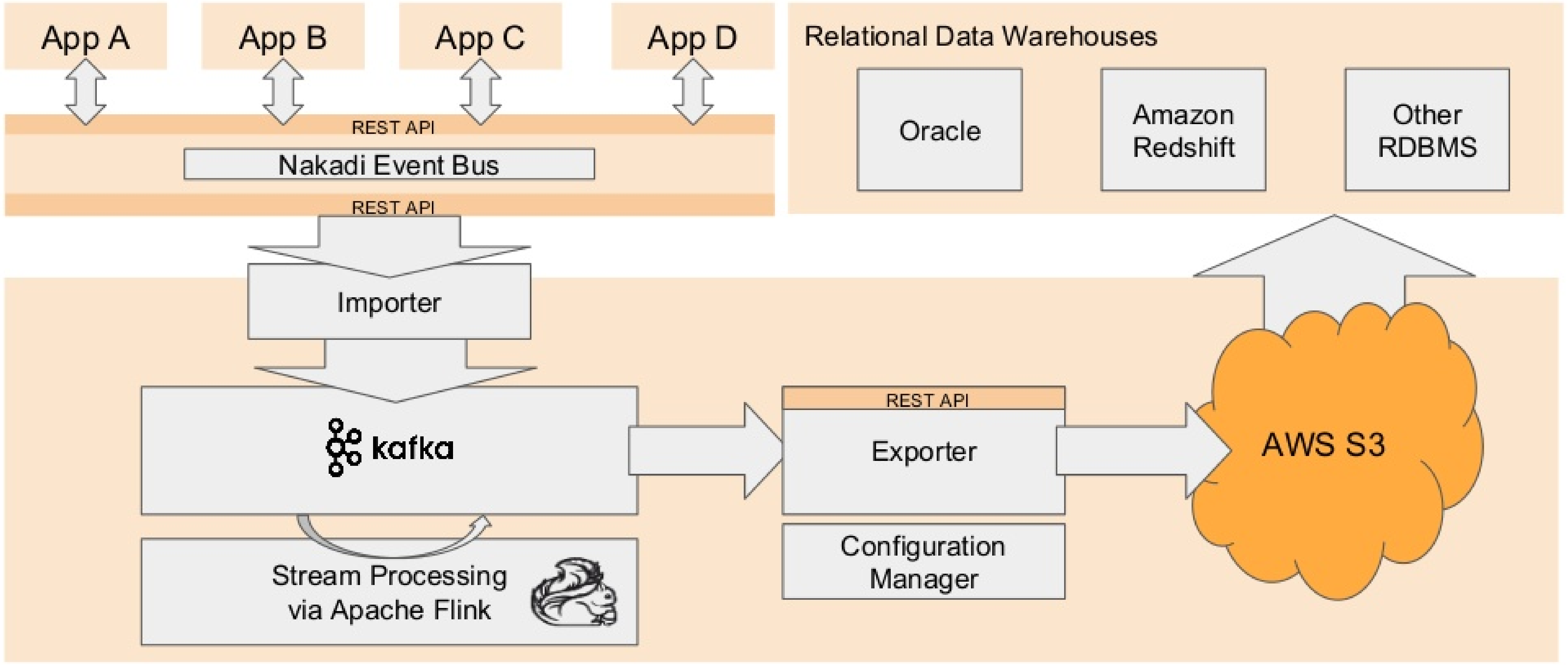
- Materialize out-of-order events in a different CSV file (correction file)
  - Generated as a side output from main stream
- Rel. DWH creates a view from main and corrected files before merging into Operational Data Store (ODS) table

DWH CLIENT'S IMPORT PROCESS


Post Processing Hook
Processes newly extracted data, e.g. merges into ODS table
DWH: ODS Layer
View
all records from CORRECTION and all from MAIN that are not in CORRECTION
External Table
MAIN
External Table
CORRECTION
One main CSV file per partition
One corrected CSV file per partition

ADVANTAGES OVER LEGACY APPROACH

LEGACY DATA INTEGRATION ARCHITECTURE



zalando

ADVANTAGES OVER LEGACY APPROACH

Fewer stacks: Flink + Materialization API + ZooKeeper
instead of Importer + Kafka + Flink + Exporter API + ZooKeeper

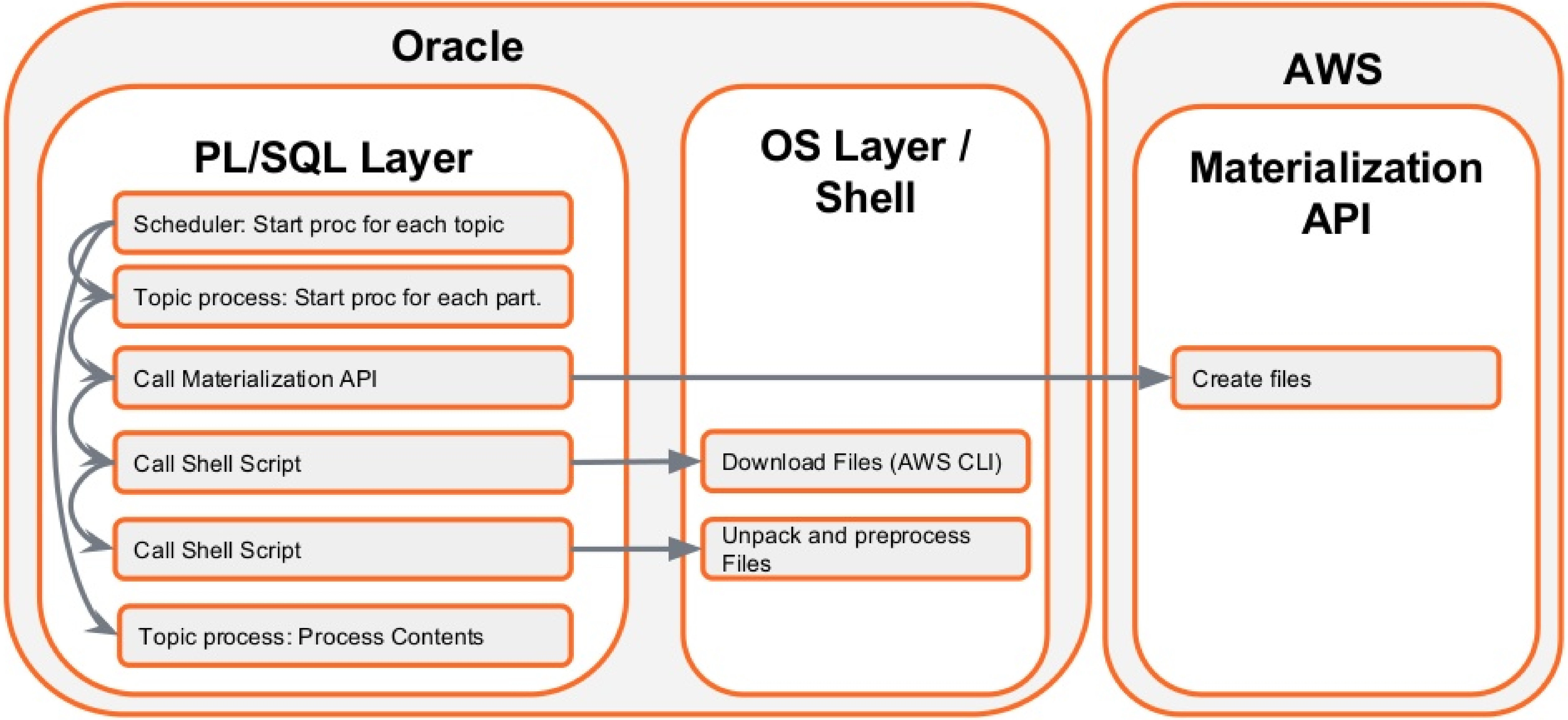
- Reduced AWS costs
- Decreased operational overhead
  - No data redundancy (Nakadi + Kafka), no Importer setup
  - Far less maintenance, e.g. no streaming array flattening jobs
- Easier reasoning and implementation through Flink's API
  - Compaction
  - Extensible feature set (more ETL in the future)

THANK YOU

mihail.vieru@zalando.de

BACKUP SLIDES

ORACLE CLIENT'S IMPORT PROCESS


Oracle
PL/SQL Layer
Scheduler: Start proc for each topic
Topic process: Start proc for each part.
Call Materialization API
Call Shell Script
Call Shell Script
Topic process: Process Contents
OS Layer / Shell
Download Files (AWS CLI)
Unpack and preprocess Files
AWS
Materialization API
Create files