Kafka
Distributed Streaming Platform

Vijay Kumar NM

Session 1

- ❑ History of messaging
- ❑ Introduction to Kafka
- ❑ Introduction to ZooKeeper
- ❑ Kafka and ZK concepts
- ❑ Use Cases

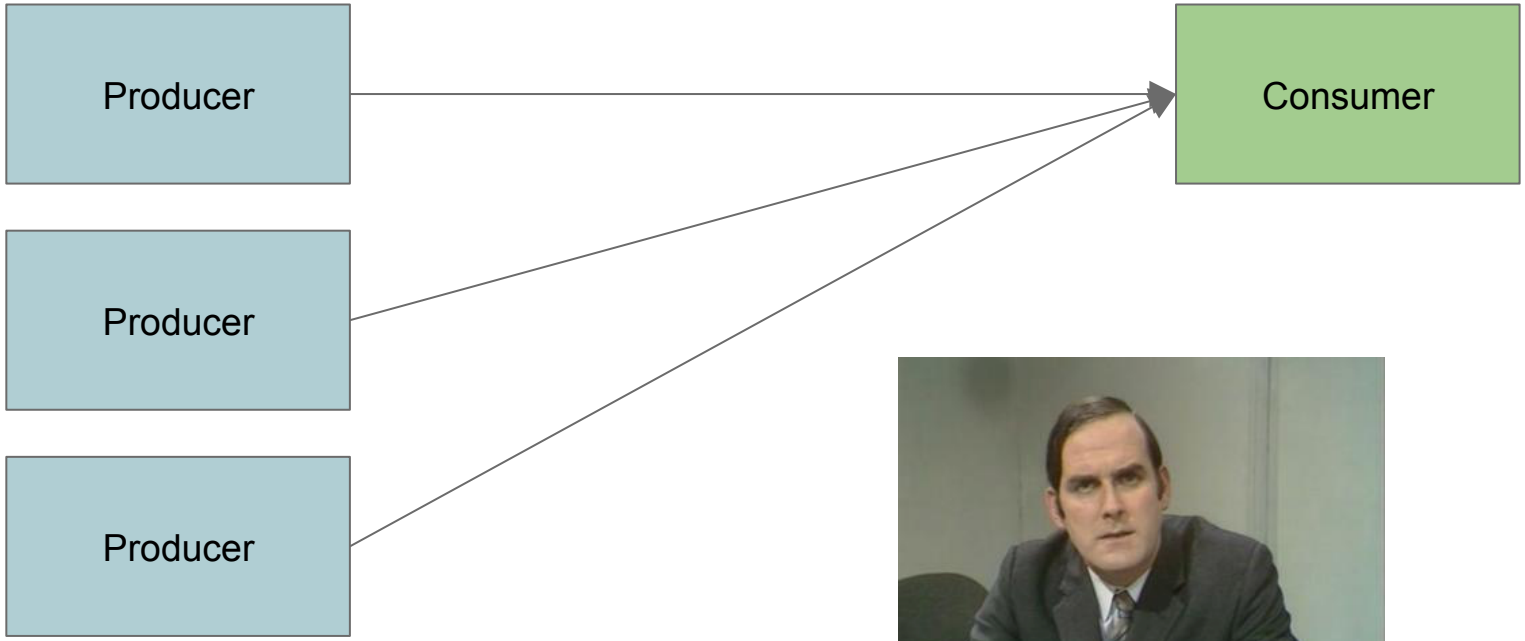
Session 2

- ❑ Installing Zookeeper
- ❑ Installing Kafka
- ❑ Kafka Cluster Configuration
- ❑ Console Consumer and Producer
- ❑ Check fault-tolerence and Leader Selection
- ❑ Java Consumer and Producer (avro)
- ❑ Producing and Consuming from specific Partition
- ❑ Custom serializer
- ❑ Kafka Internals

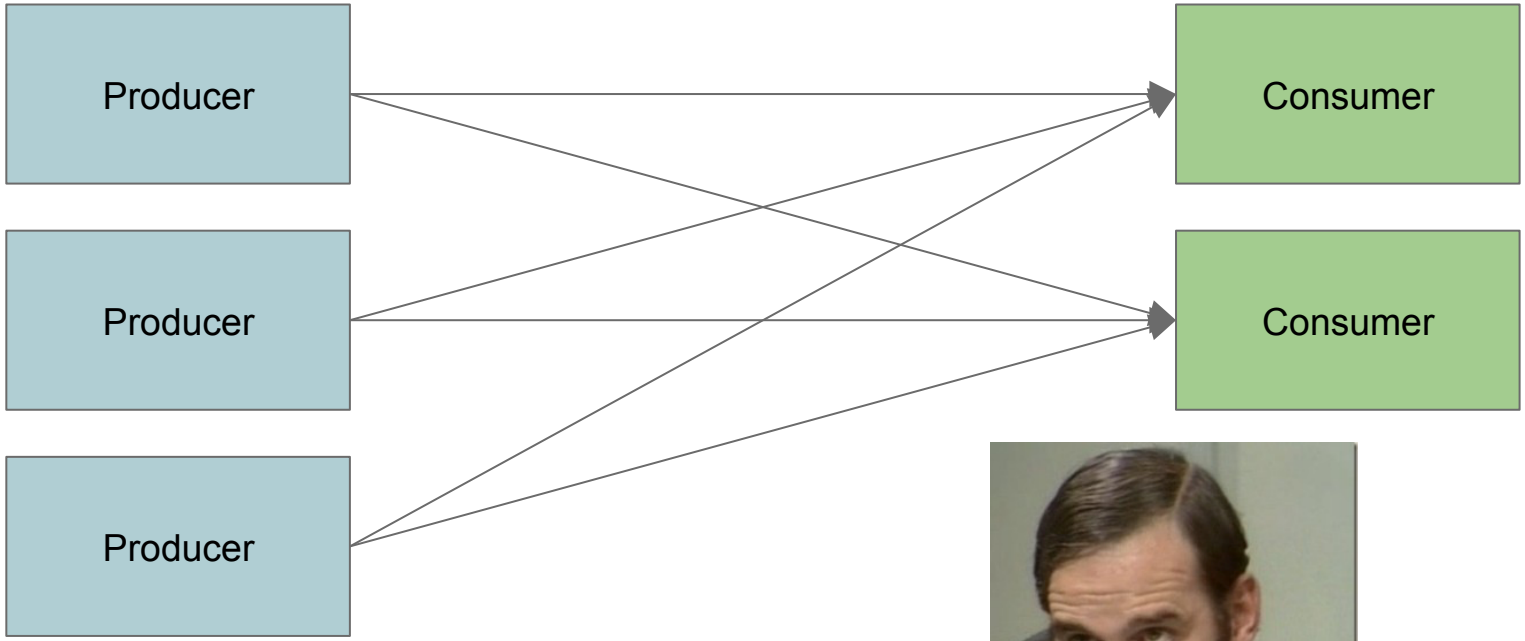
Session 3

- ❑ Kafka Connectors
- ❑ Kafka Streaming
- ❑ Common Issues and Solutions
- ❑ Debugging Issues

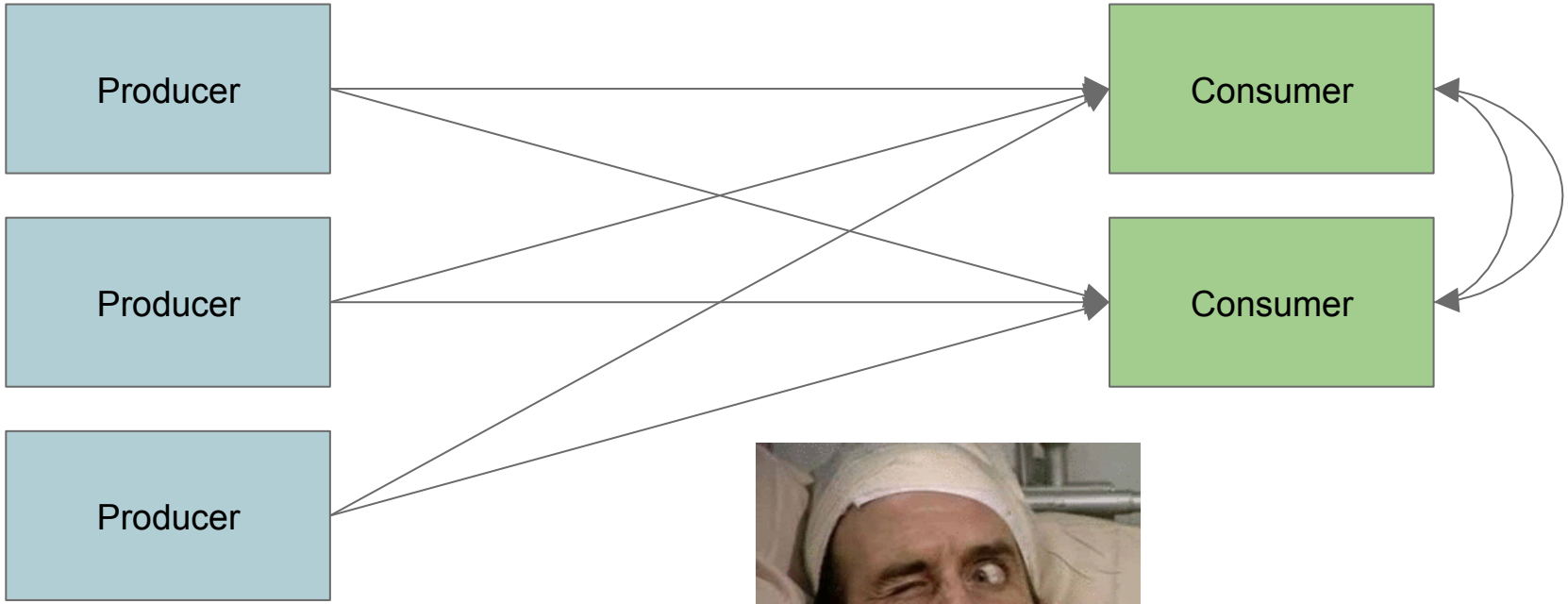
Data Pipelines (Phase 1)



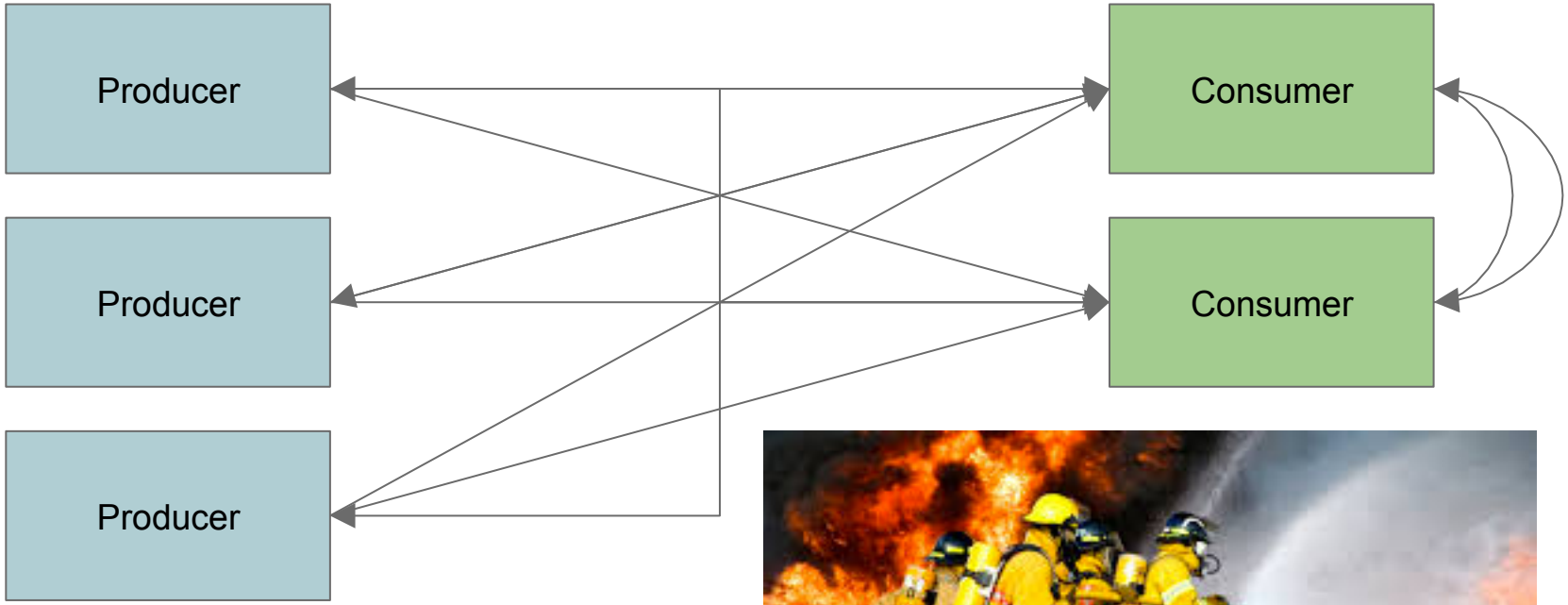
Data Pipelines (Phase 2)



Data Pipelines (Phase 3)



Data Pipelines (Phase 4)



Data Pipelines (Phase 5)

What is Kafka?

It looks like a messaging system

It has Messaging features

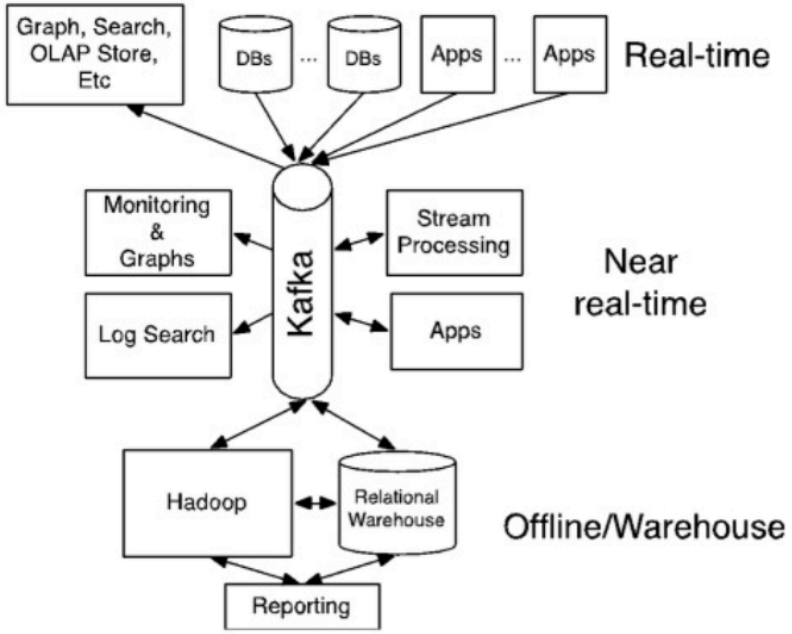
But in reality, it’s a distributed, replicated log

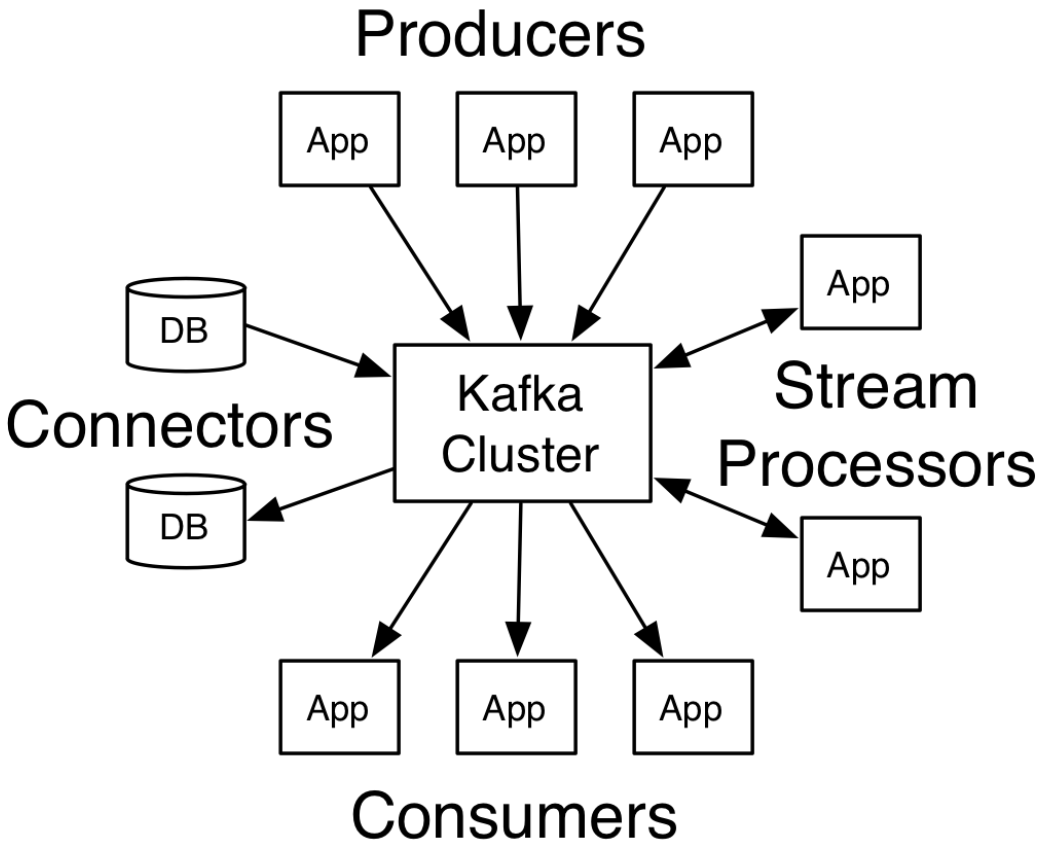
Read and write

In real time (Streaming)

As much as you want

As fast as your network can go




Producers
App
App
App
DB
DB
Connectors
Kafka
Cluster
App
Stream
Processors
App
App
App
App
Consumers

Kafka

Some basic info

Created at LinkedIn (still in production, 2011)

Was meant to move data around in massive scale and massive velocity

Partitioned for horizontal scalability

Follows the Pub-Sub architecture

Concepts

Kafka is run as a **cluster** on one or more servers

The Kafka cluster stores streams of **records** in categories called **topics**

Each **record** consists of a **key**, a **value** and a **timestamp**

Processes that published messages to a Kafka topic are called **producers**

Processes that subscribe to topics and process the feed of published messages are called **consumers**

Each of the nodes (servers) that are part of the Kafka Cluster are called **brokers**.

Topics and Logs

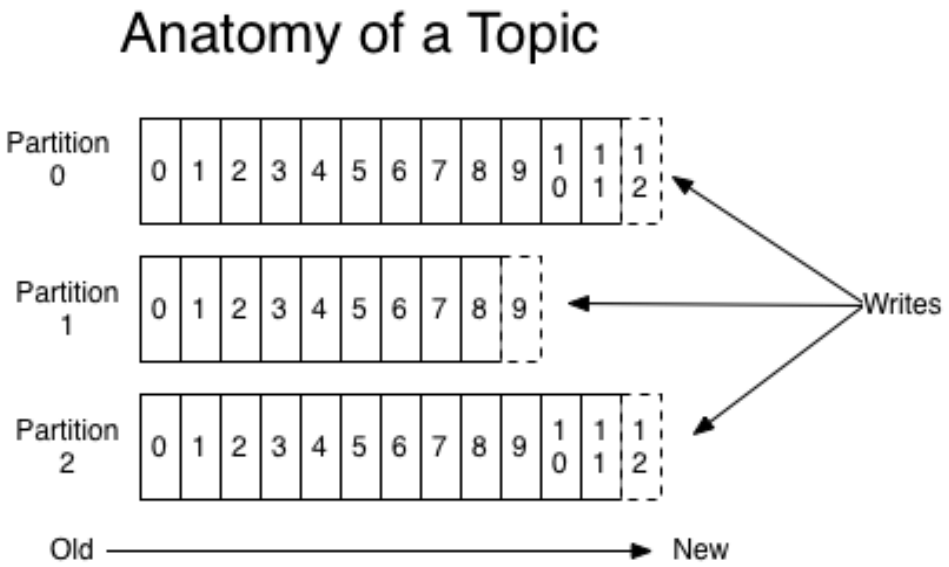
Topic is a Category

Topic is multi-subscriber

Each Topic is Partitioned

Partition is Ordered, Immutable and Sequence of records

Record has Sequential Id number (Offset)



Producer and Consumer

Topic is a Category

Topic is **multi-subscribe**r

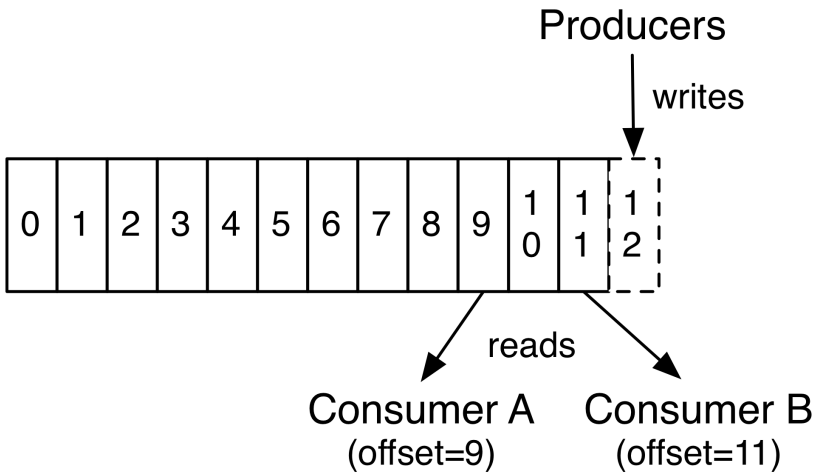
Each Topic is **Partitioned**

Partition is **Ordered**, **Immutable** and Sequence of records

Partitions have **segments**

Record has Sequential Id number (**Offset**)

Records have **retention period**



What’s different from Message Bus?

Topics are partitioned - Order is guaranteed per partition!

Partition is replicated for fault-tolerance

Producers push data to Kafka

Consumers PULL data from Kafka

Tunable ACK from the partition leader for producers

This approach allows for high scalability

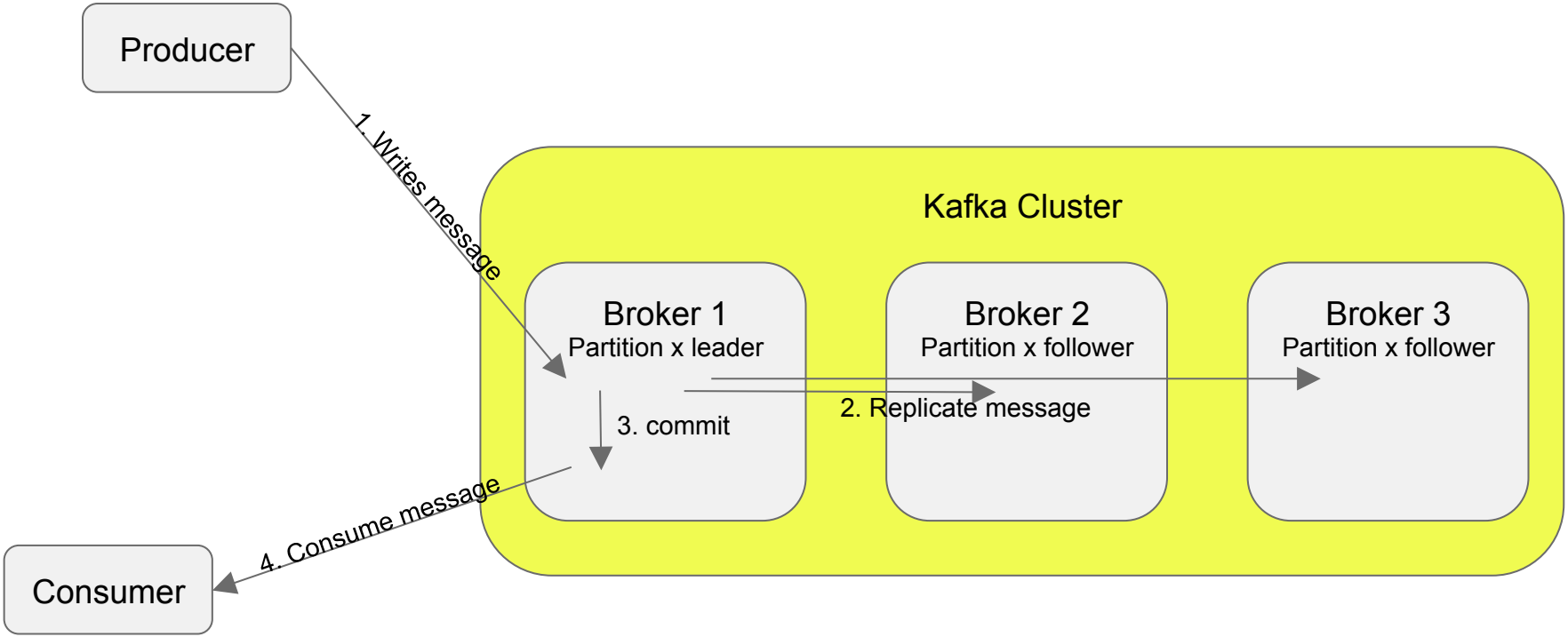
Multiple Consumers can do different things at the same time

How data is transmitted?

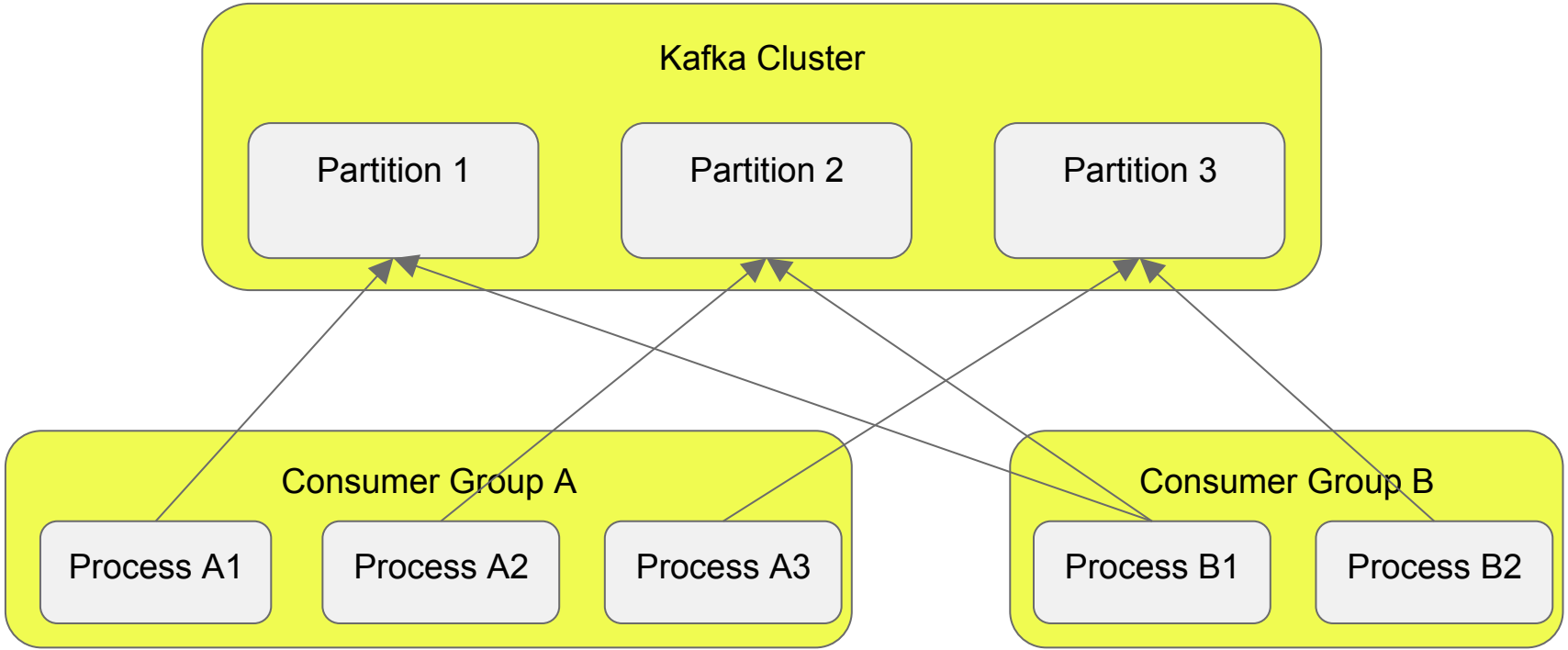
Communication between all components is done via a

high performance simple **binary API** over **TCP** protocol.

Pushing a message



Reading a message

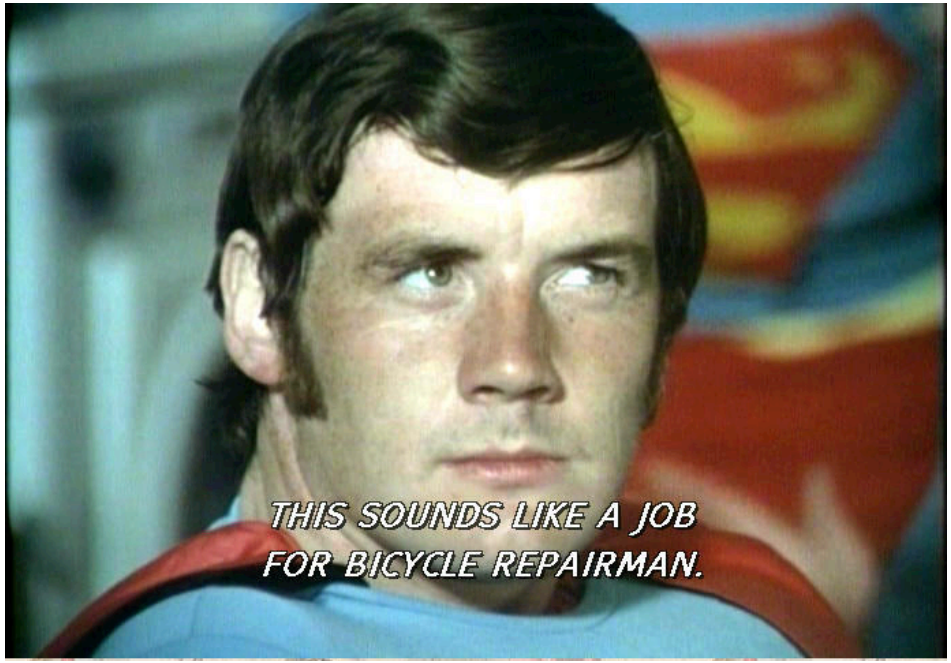


So what can we do with this?

Kafka can be the super-connector to everything.

Can be used for:

1. Stream Processing
2. Data pipelines
3. Micro-Services



What is Streaming?

It lets you publish and subscribe to streams of records (similar to a message queue or EMS)

It lets you store streams of records in a fault-tolerant way.

It lets you process streams of records as they occur.

Stream Processing, Use Cases

Life doesn’t happen in batches. The goal is to streamline.

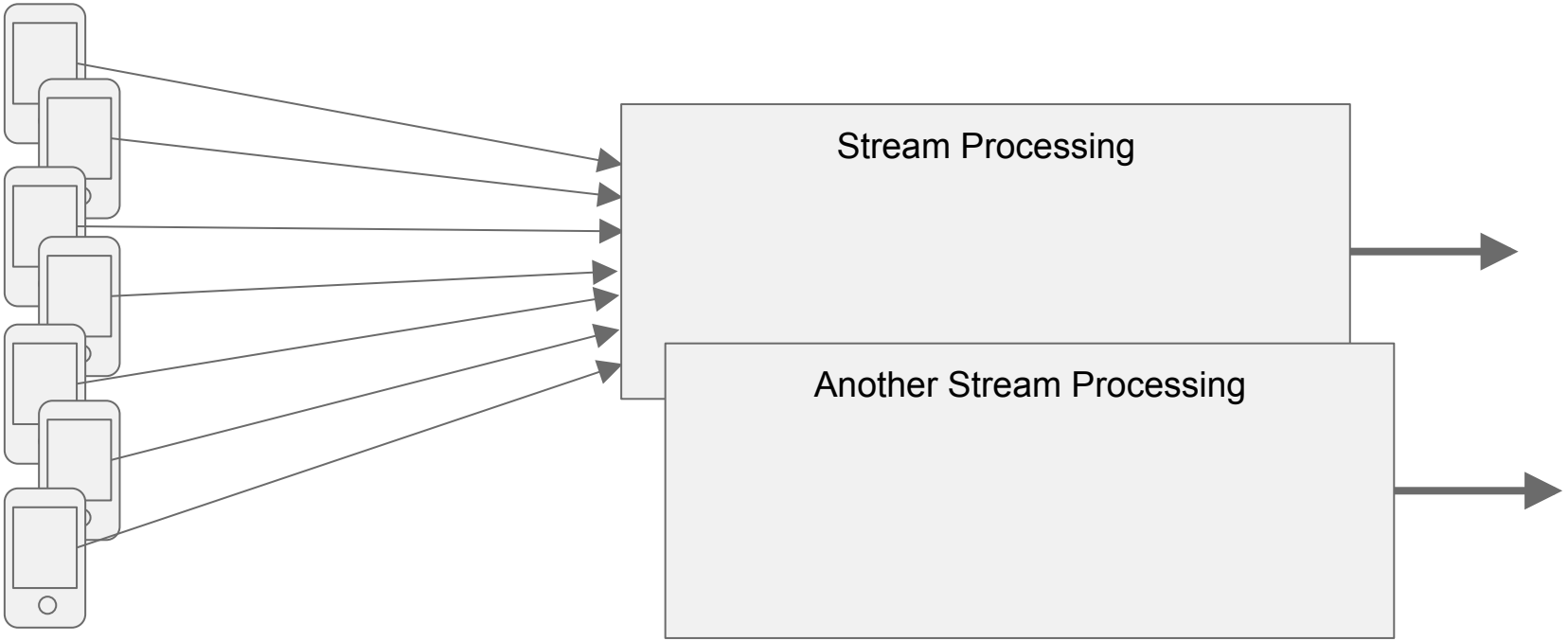
Many of the systems we want to monitor and to understand happen as a continuous stream of events—heartbeats, ocean currents, machine metrics, GPS signals.

Planes, Trains, and Automobiles: Connected Vehicles and the IoT

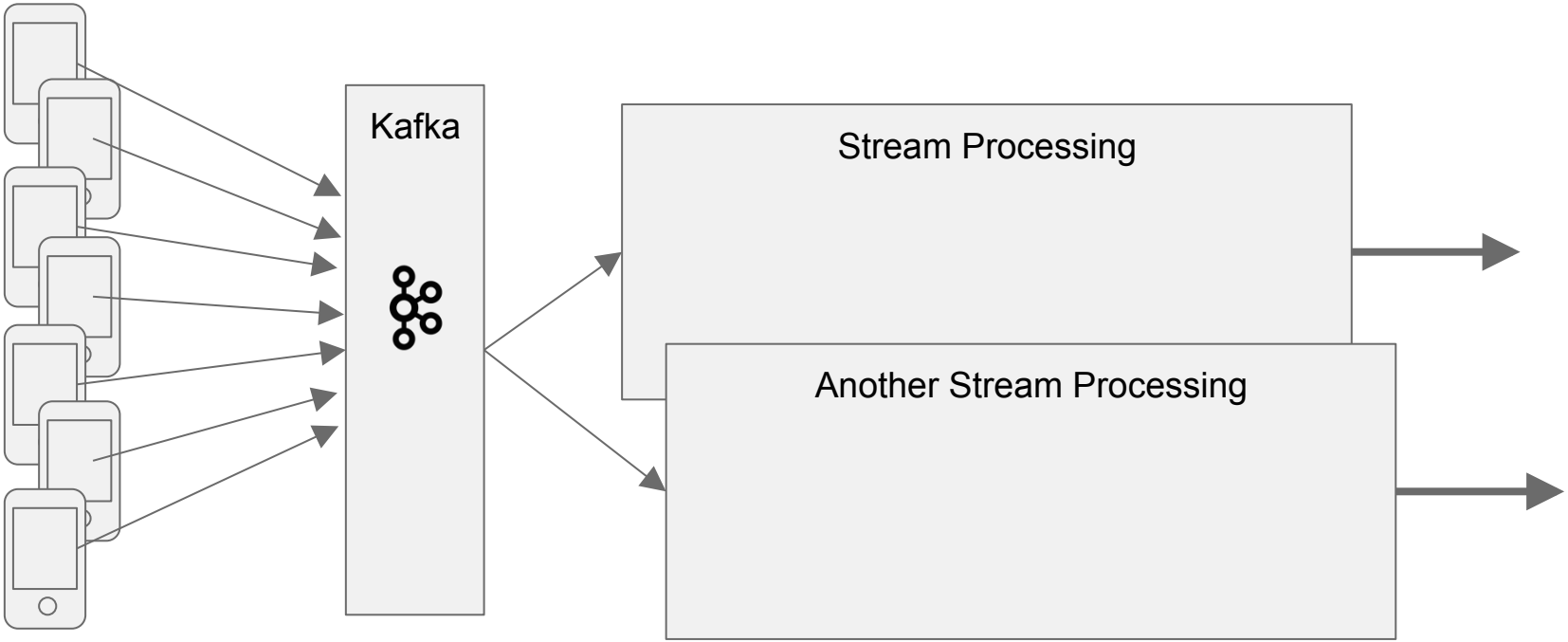
Collect and analyze information from these events as a stream of data. **time-value of information**

Even analysis of sporadic events such as website traffic can benefit from a streaming data approach.

Stream Processing



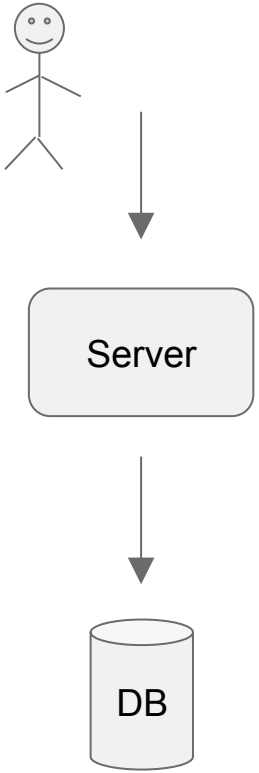
Stream Processing

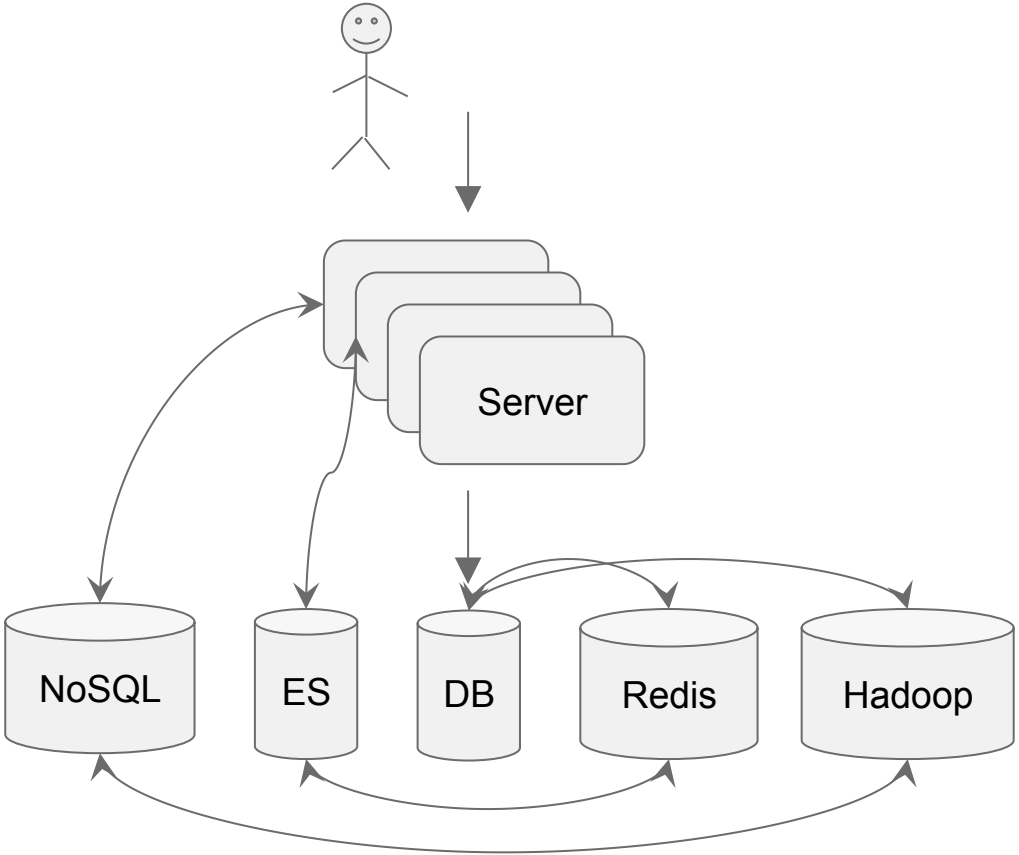


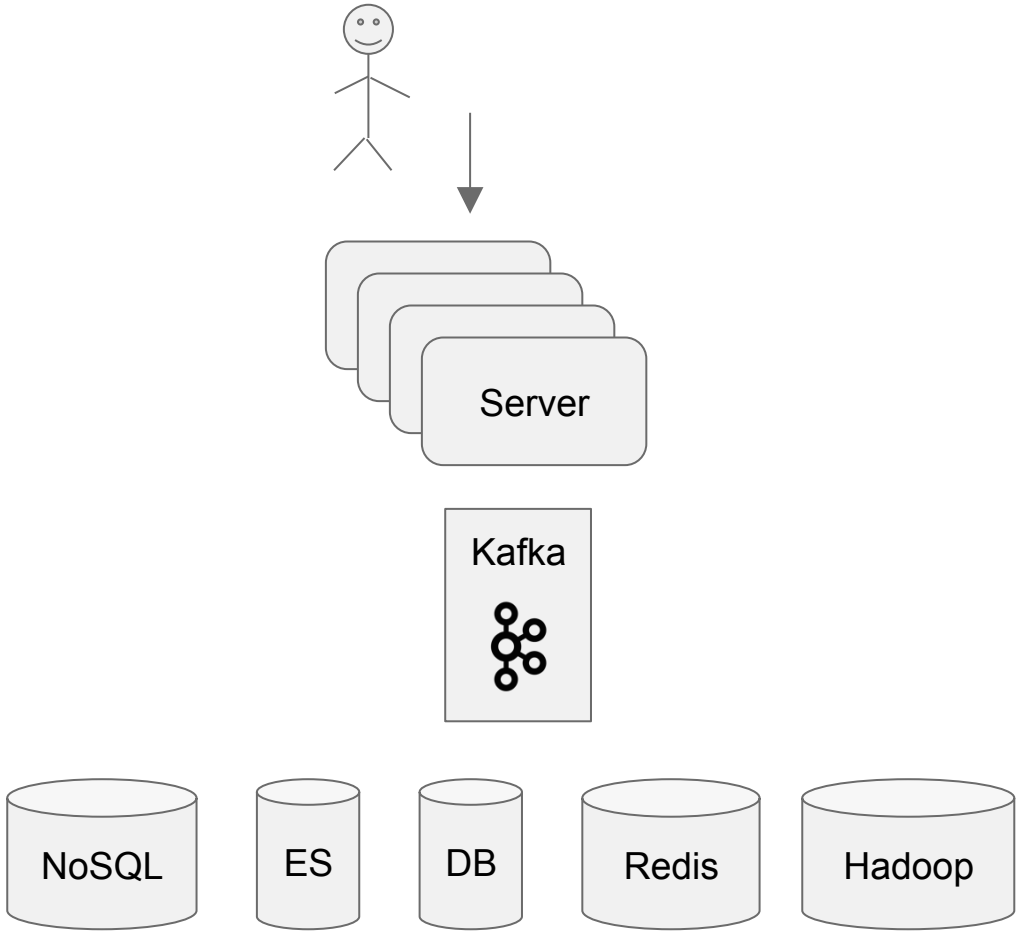
Data Pipelines

In the age of Data, We tend to save data in multiple places.

But more and more of those places require the data ASAP


Server
DB

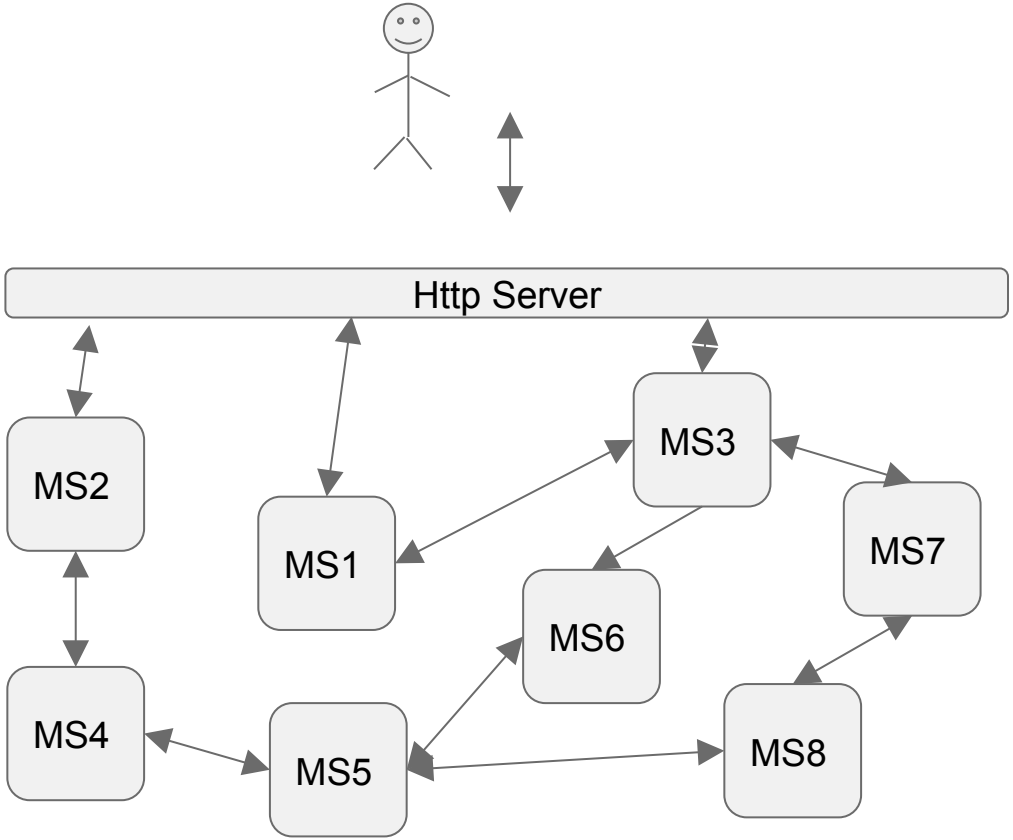

Server
NoSQL
ES
DB
Redis
Hadoop


Server
Kafka
NoSQL
ES
DB
Redis
Hadoop

Micro-Services

The loch-ness monster of the Software architecture world.

Specialized, small (micro?) services that each handles a very specific task or set of tasks that are closely related.


Http Server
MS2
MS1
MS3
MS7
MS6
MS4
MS5
MS8

The problem?

Complexity

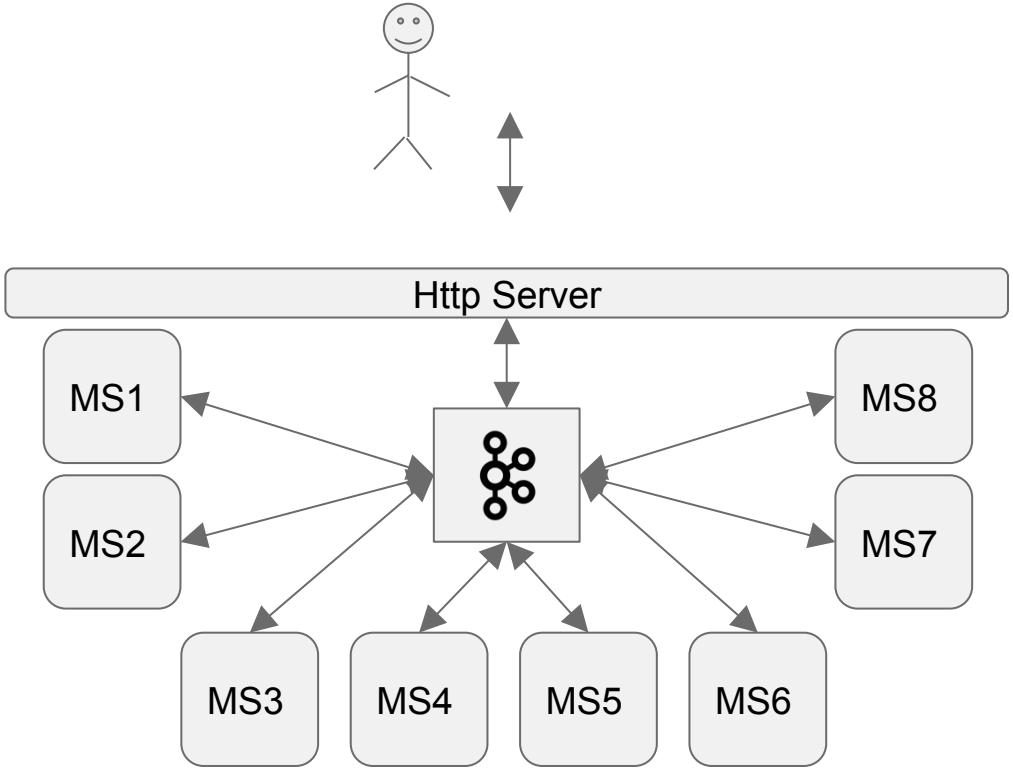
Reliability

(A)Synchronous

Complex interactions between 2 or more micro services require them to know each other very (perhaps too) well.

If 4 micro-services are connected in 1 pipeline, each with 99.9% availability, the pipeline as a whole has 99.5% percent.

Some pipelines are synchronous and some are not. Managing this can be hard, and sometimes we will block operations, only because it is easier.


Http Server
MS1
MS2
MS3
MS4
MS5
MS6
MS7
MS8

Performance?

100 ,000 msg/sec

On a barely tweaked, 3 node cluster

Performance?

2 ,000, 000 msg/sec

On a heavily tweaked cluster

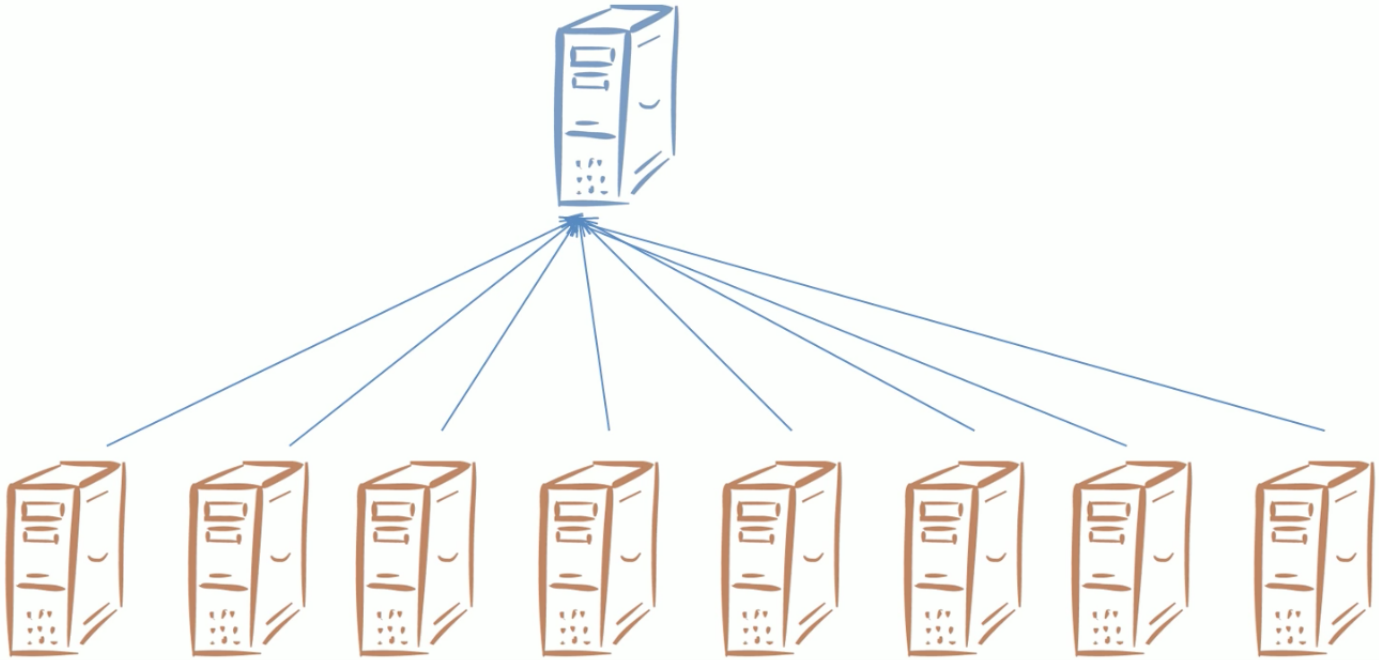
A word on Apache ZooKeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Apache Kafka (as well as many other Apache Big Data platforms) use Apache ZooKeeper to manage shared state among different nodes.

Why do we need Zookeeper?

Why do we need Zookeeper?



A word about Confluent

Founded by the creators of Apache Kafka (LinkedIn)

Provide professional services for Kafka

And Kafka related Products:

- Kafka Connect
- Kafka Streams
- Control Center

Recap (1 of 2)

Kafka is a distributed, replicated log

Kafka is run as a **cluster** on one or more servers

Each of the nodes (servers) that are part of the Kafka Cluster are called **brokers**.

The Kafka cluster stores streams of **records** in categories called **topics**

Each Topic is **Partitioned** for parallel consumption to increase throughput

Partition is Ordered, Immutable and Sequence of records

Each **record** (message) consists of a **key**, a **value** and a **timestamp**

Records have retention period

Processes that published messages to a Kafka topic are called **producers**

Processes that subscribe to topics and process the feed of published messages are called **consumer**

Recap (1 of 2)

Guarantees:

- ✓ Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
- ✓ That is, if a record M1 is sent by the same producer as a record M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.
- ✓ A consumer instance sees records in the order they are stored in the log.
- ✓ For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

Install Zookeeper

https://zookeeper.apache.org

```
> bin/zkServer.sh start
```

```
> bin/zkCli.sh -server 127.0.0.1:2181
```

Install Kafka

http://kafka.apache.org/

```
> tar -xzf kafka_2.11-0.10.2.0.tgz

> cd kafka_2.11-0.10.2.0

> bin/kafka-server-start.sh config/server.properties
```

Topic

> bin/kafka-topics.sh --create --zookeeper 10.2.3.163:2181 --replication-factor 2 --partitions 1 --topic test

> bin/kafka-topics.sh --list --zookeeper 10.2.3.163:2181

> bin/kafka-topics.sh --describe --zookeeper 10.2.3.163:2181 --topic test

Messages

> bin/kafka-console-producer.sh --broker-list **10.2.3.168**:9092 --topic test

> bin/kafka-console-consumer.sh --bootstrap-server **10.2.3.169**:9092 --from-beginning --topic test

> bin/kafka-console-producer.sh --broker-list **10.2.3.169**:9092 --topic test

Kafka Broker Configuration (1 of 3)

broker.id

port

log.dirs

zookeeper.connect

Kafka Broker Configuration (2 of 3)

auto.create.topics.enable	true
delete.topic.enable	false
num.partitions	1
default.replication.factor	1

Kafka Broker Configuration (3 of 3)

log.retention.bytes	-1
log.retention.ms	
log.retention.minutes	
log.retention.hours	7days
log.retention.check.interval.ms	300000

Kafka Producer Configuration (1 of 2)

bootstrap.servers		
key.serializer		
value.serializer		
acks	1	0,1, all
retries	0	
compression.type	none,	gzip, snappy, lz4
buffer.memory	33554432 bytes	

Kafka Producer Configuration (2 of 2)

batch.size	16384 bytes
linger.ms	0 ms
request.timeout.ms	30000
timeout.ms	30000
client.id	
max.request.size	1048576 bytes

Kafka Replication

As with most distributed systems automatically handling failures requires having a precise definition of what it means for a node to be "alive". For Kafka node liveness has two conditions

- **A node must be able to maintain its session with ZooKeeper (via ZooKeeper's heartbeat mechanism)**
- **If it is a follower it must replicate the writes happening on the leader and not fall "too far" behind**

We refer to nodes satisfying these two conditions as being "in sync" to avoid the vagueness of "alive" or "failed". The leader keeps track of the set of "in sync" nodes. If a follower dies, gets stuck, or falls behind, the leader will remove it from the list of in sync replicas. The determination of stuck and lagging replicas is controlled by the replica.lag.time.max.ms configuration.

Kafka Consumer Configuration

Consumer Rebalancing

If there are more consumers than partitions on the topic, some consumers will never see a message

If there are more partitions than consumers, some consumers will receive data from multiple partitions

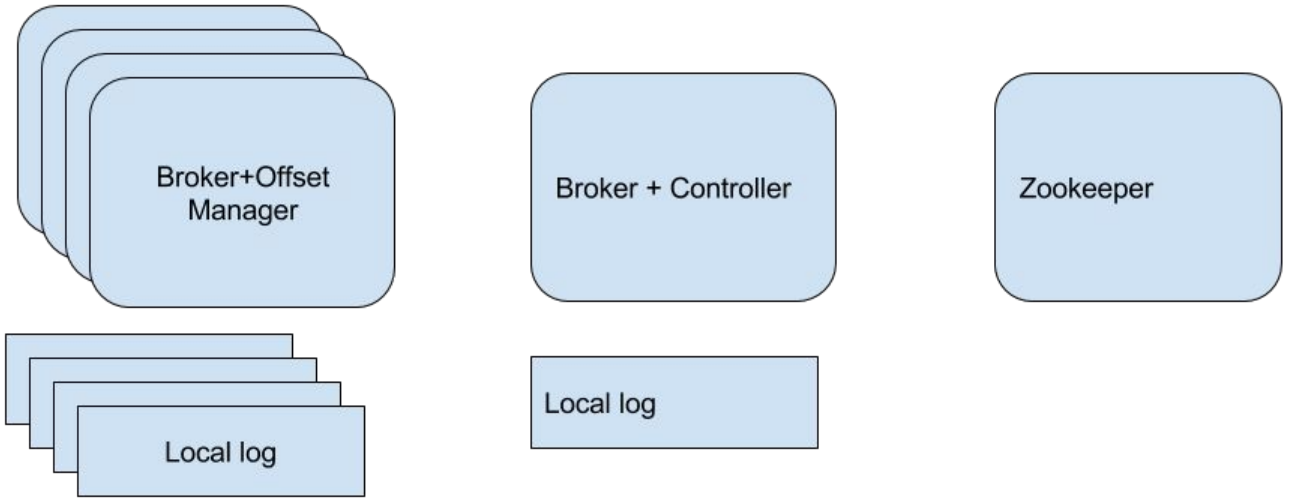
if there are multiple partitions per consumers there is NO guarantee about the order you receive messages, other than that within the partition the offsets will be sequential. For example, you may receive 5 messages from partition 10 and 6 from partition 11, then 5 more from partition 10 followed by 5 more from partition 10 even if partition 11 has data available.

Adding more consumers/threads will cause Kafka to re-balance, possibly changing the assignment of a Partition to a Thread.

Kafka Internals

Vijay Kumar NM

High level architecture



Role of the broker

Broker is handling read/writes

It forward messages for replication

It does compactions on its own logs replica (without influence to any other copies)

Offset management

Prior to 0.81 it was pure Zookeeper responsibility to hold offsets metadata. Starting from 0.81 - there is special offset manager service.

It runs with Broker, use special topic to store offsets and also do in-memory caching as optimization.

Role of the controller

Handling “cluster wide events” sent by Zookeeper (all in sync with Zookeeper registries)

- Brokers list change (registration, failure)
- Leaders election
- Change in topics (deleted, added, num of partitions changed)
- Track partitions replica

Zookeeper role

- Kafka controller registration
- List of topics and partitions
- Partition states
- Brokers registration (id, host, port)

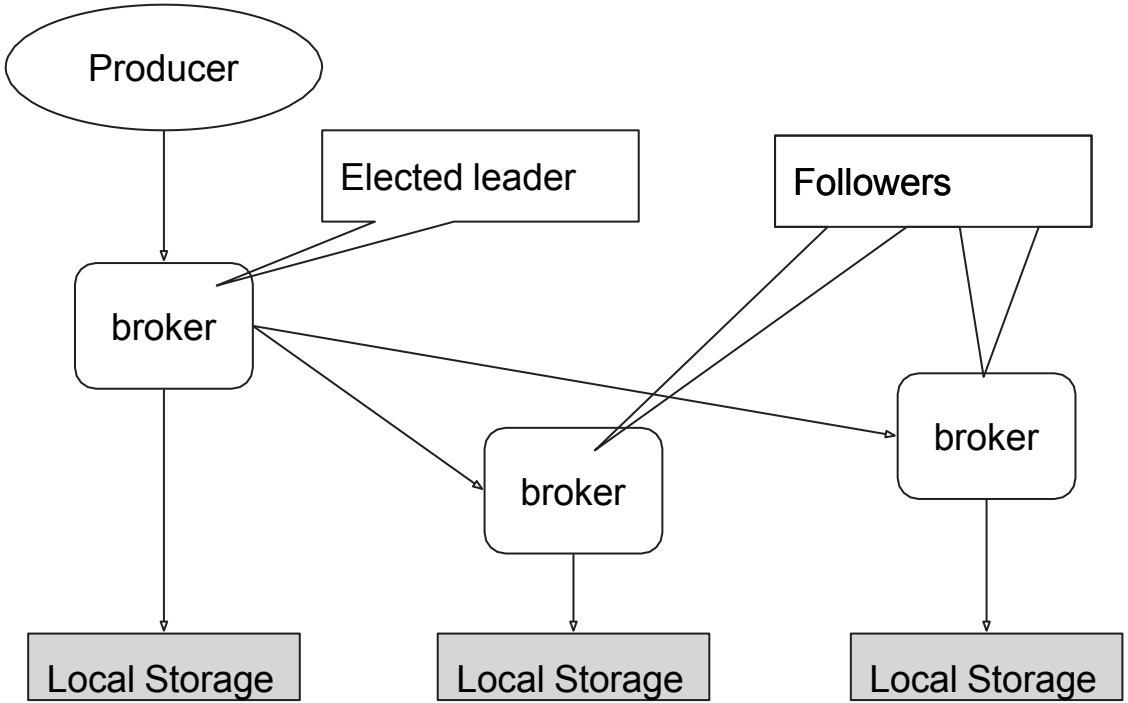
Partitions

- Each partition has its leader broker and N followers.
- Consumers and producers works with leaders only.
- Partition is the main mechanism of a scale, within the topic.
- Producer specify the target partition via Partitioner implementation (balancing within available topic partitions)

Access Pattern

- Writers write data in massive streams. Data is already "ordered". (This ordering is re-used)
- Readers consume data, each one from some position, sequentially.

Write path


Producer
Elected leader
Followers
broker
broker
broker
Local Storage
Local Storage
Local Storage

Read path

Read always happens via partition leader.

Kafka helps to balance consumers within the group. Each topic’s partition could be read by single consumer at a time to avoid simultaneous read of the same message by several consumers within the same group.

Why is it so fast?

1. Sequential disk access - optimal disk utilization
2. Zero copy - save CPU cycles.
3. Compression - save network bandwidth.
4. Network and Disk formats of messages are the same.
5. Local storage is used.
6. No attempts to cache / optimize.

Compression

Up to 0.7 Kafka - special kind of compressed messages was handled by clients (producer and consumer parts), the Broker was not involved.

Starting 8.1, Kafka broker repackages messages in order to support logical offsets.

Indexing

- When data flows into the broker, it is being “indexed”.
- Index files are stored alongside “segments”.
- Segments are files with the data.

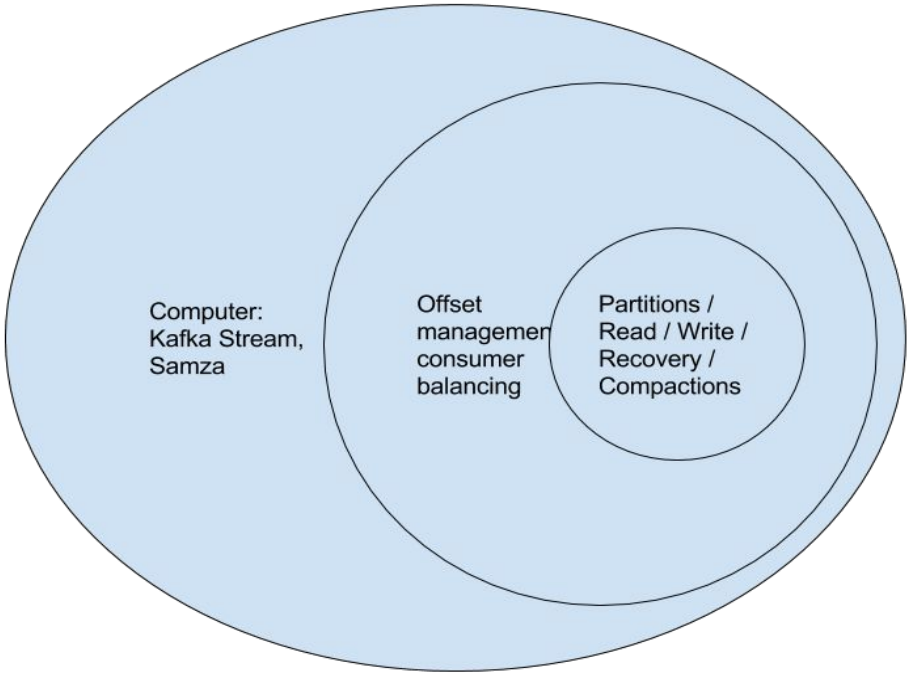
Consumer API Levels

- Low level API : work with partitions and offsets
- High level API : Work with topics, automatic offset management, load balancing.

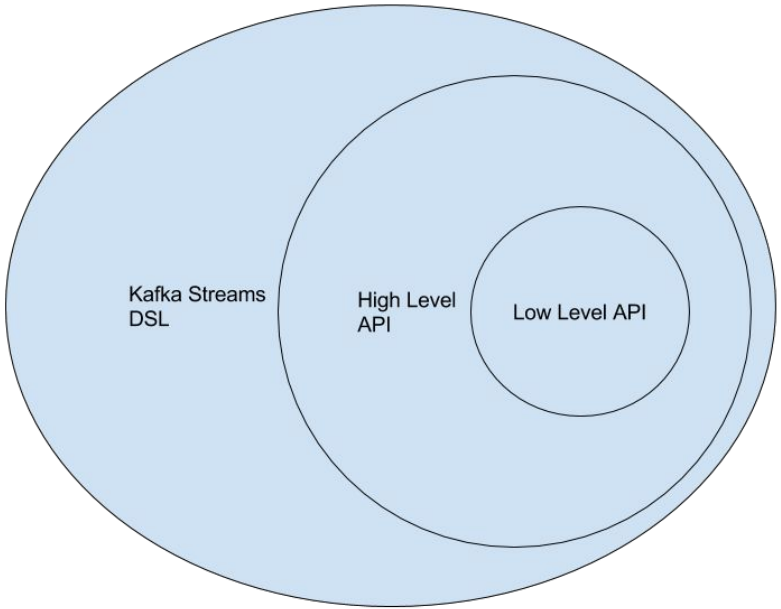
Can be rephrased as

- Low level API : Database
- High level : Queue

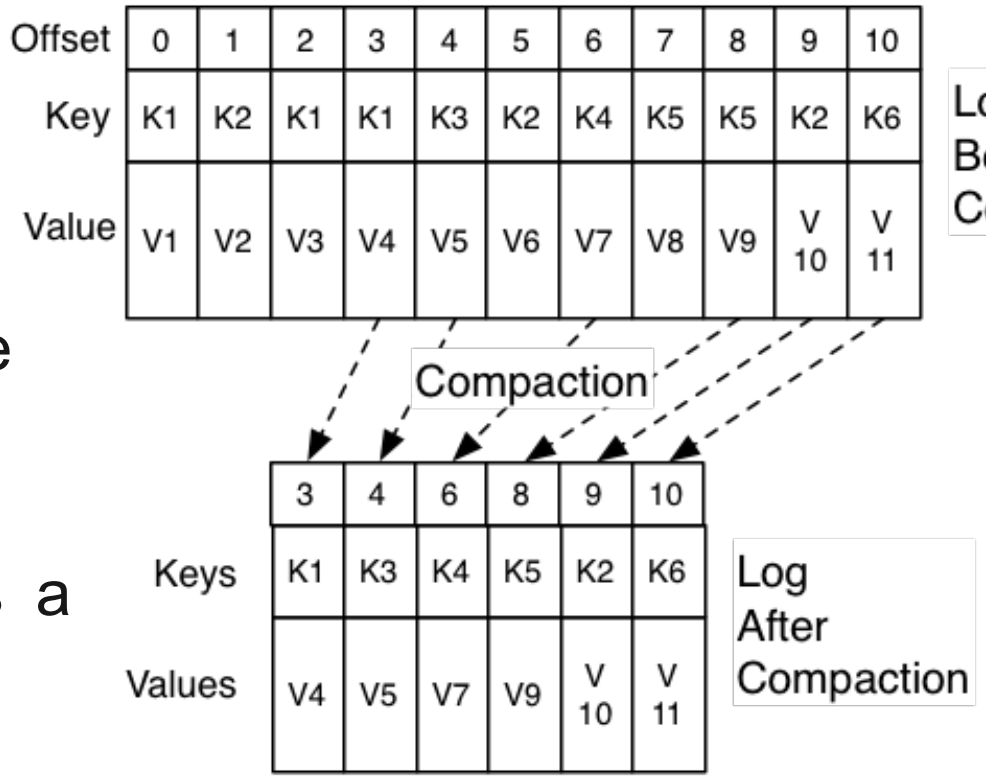
Layers of functionality



Levels of abstraction



Key Compaction



Kafka is capable to store only latest value per key.

It is not a Queue. It is a table.

Compute

Samza, Kafka Streams relation to Kafka is like MapReduce, Spark relation to HDFS

Kafka became media on top of which we build computational layers.

Kafka Security

Vijay Kumar NM

Security

IoT

Massive inputs of data

Whole lot of aggregations, transformations & unification

Security

Apache Kafka is frequently used to store critical data making it one of the most important components of a company’s data infrastructure

Multi-tenancy is an essential requirement

security features are crucial for multi-tenancy

Previous to 0.9, Kafka had no built-in security features

cyber-attacks are a common occurrence and the threat of data breaches is a reality

Security

Four key security features were added in Apache Kafka 0.9, which is included in the Confluent Platform 2.0:

Administrators can require client authentication using either Kerberos or Transport Layer Security (TLS) client certificates, so that Kafka brokers know who is making each request

A Unix-like permissions system can be used to control which users can access which data.

Network communication can be encrypted, allowing messages to be securely sent across untrusted networks.

Administrators can require authentication for communication between Kafka brokers and ZooKeeper.

Kafka
Streams and Connectors

Vijay Kumar NM

Kafka Platform

Streams and Connect apps are just (java) apps

Streams and Connect are libraries

Can be deployed like any other (java) app

Multiple instances of the same app can be launched

Use tools like Mesos, kubernetes, Docker Swarm, ...

Transformation of
a stream of data fragments
into a continuous flow of
information

Stream Processing

Get real-time insights

Lower processing latency

Easier to test

Easier to maintain

Easier to scale

Different way of thinking

At-least-once vs exactly-once

Time

Every company is already doing stream processing

(more or less ...)

A Stream

```
Key 1 -> value 1
Key 2 -> value 2
Key 1 -> value 3
    ...
```

A Table

Key 1	value 3
Key 2	value 2

A Table through time

Timestamp 1

| Key 1 | value 1 |

Timestamp 2

| Key 1 | value 1 |
| Key 2 | value 2 |

Timestamp 3

| Key 1 | value 3 |
| Key 2 | value 2 |

Timestamp ...

Let’s remove the redundancy

A Table through time as SETs

Timestamp 1

```
SET(key1 -> value1)
```

Timestamp 2

```
SET(key2 -> value2)
```

Timestamp 3

```
SET(key 1 -> value3)
```

Timestamp ...

Changelog

```
SET(key1 -> value1)
SET(key2 -> value2)
SET(key1 -> value3)
```

Stream

```
key1 -> value1
key2 -> value2
key1 -> value3
```

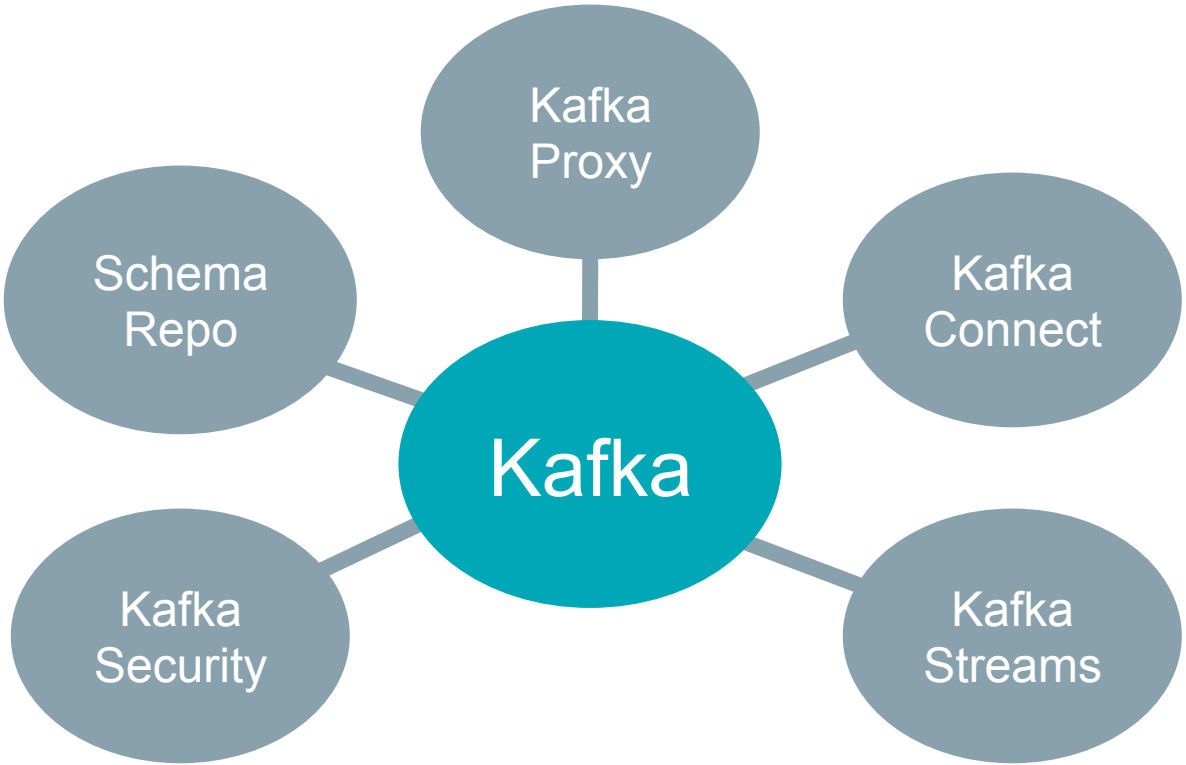
Tables are materialized views of streams

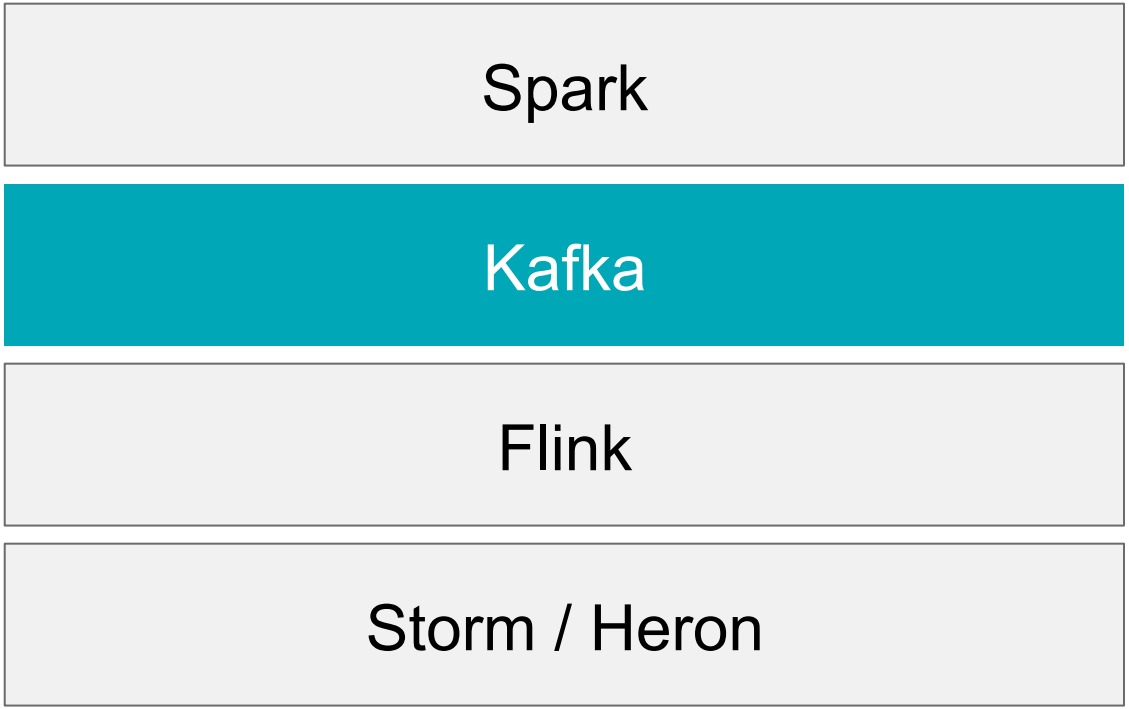
Events used to manipulate core data.
Today events are our core data

Daan Gerits, 2012

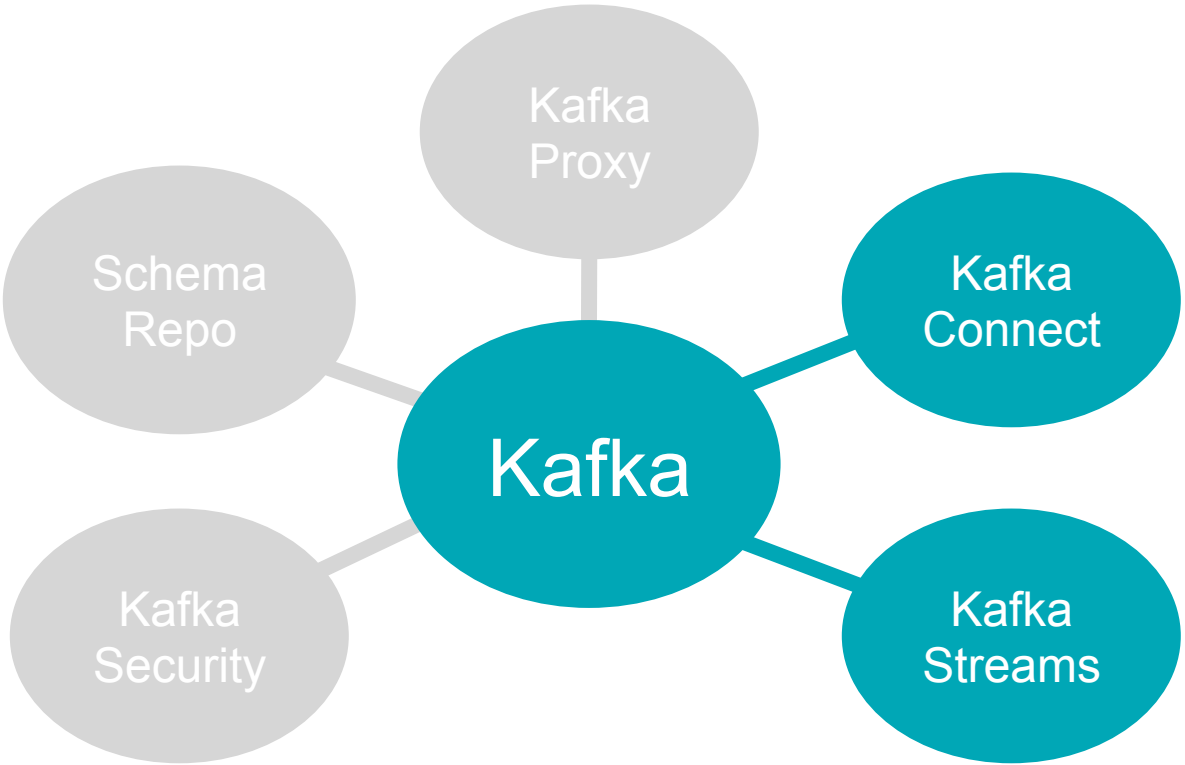
Every stream process app is a combination of state and streams

Streaming vs batch
is like
agile vs waterfall
but then for data.


Kafka
Proxy
Schema
Repo
Kafka
Connect
Kafka
Kafka
Security
Kafka
Streams


Spark
Kafka
Flink
Storm / Heron

Build apps, not Jobs

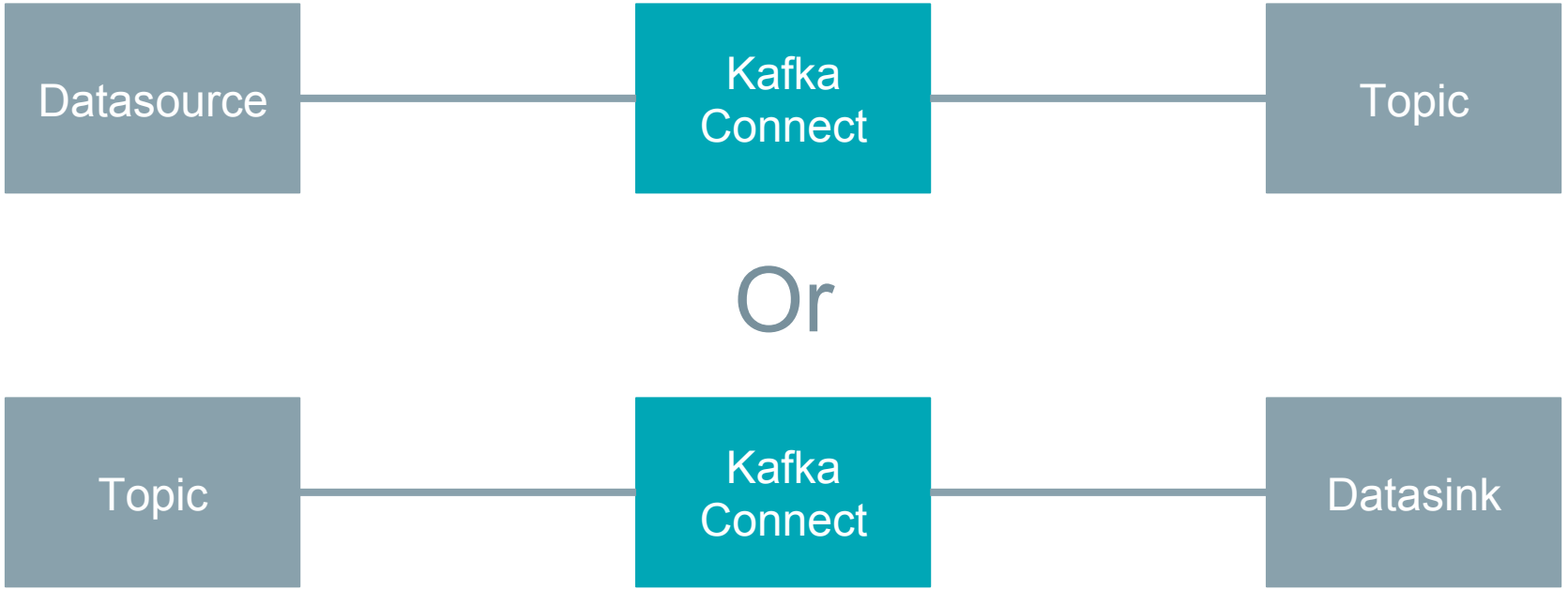

Kafka
Proxy
Schema
Repo
Kafka
Connect
Kafka
Kafka
Security
Kafka
Streams

Kafka Connect

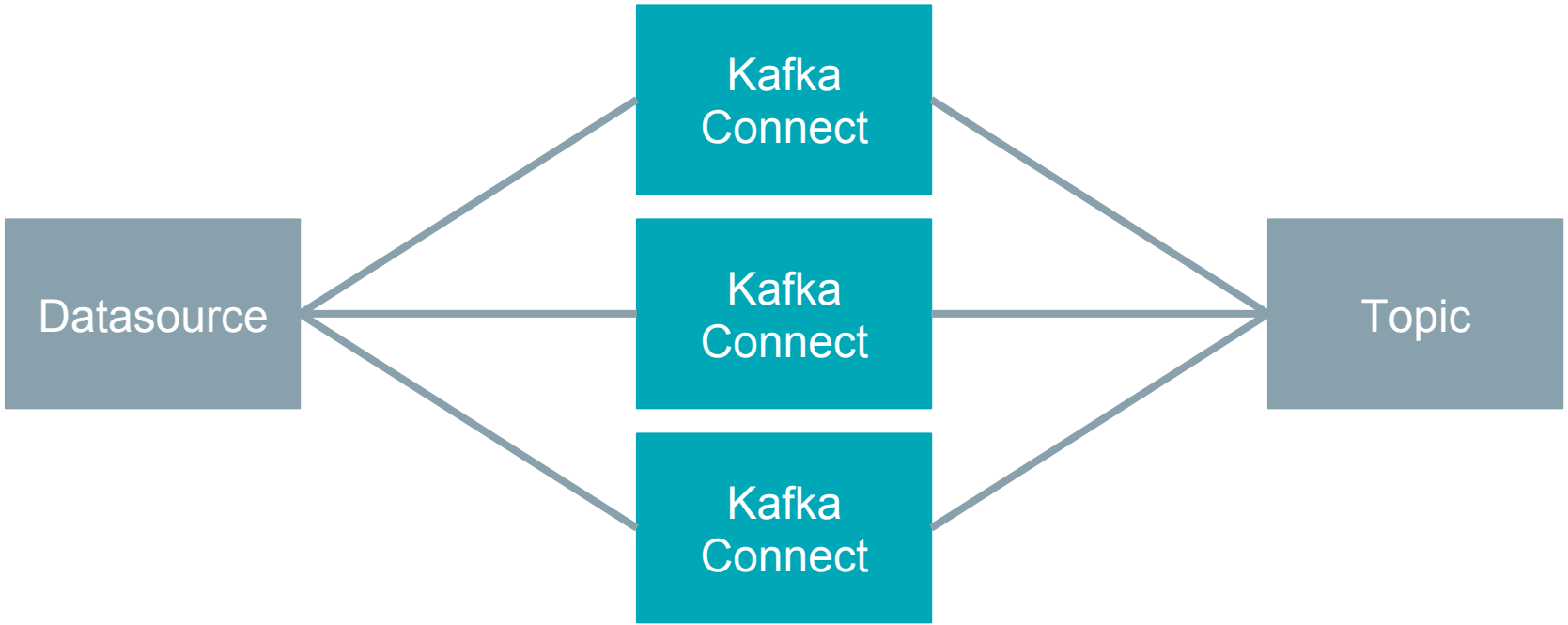
Getting data in and out

A
Simple and scalable
way to get
data in and out
of topics

Kafka Connect



Kafka Connect



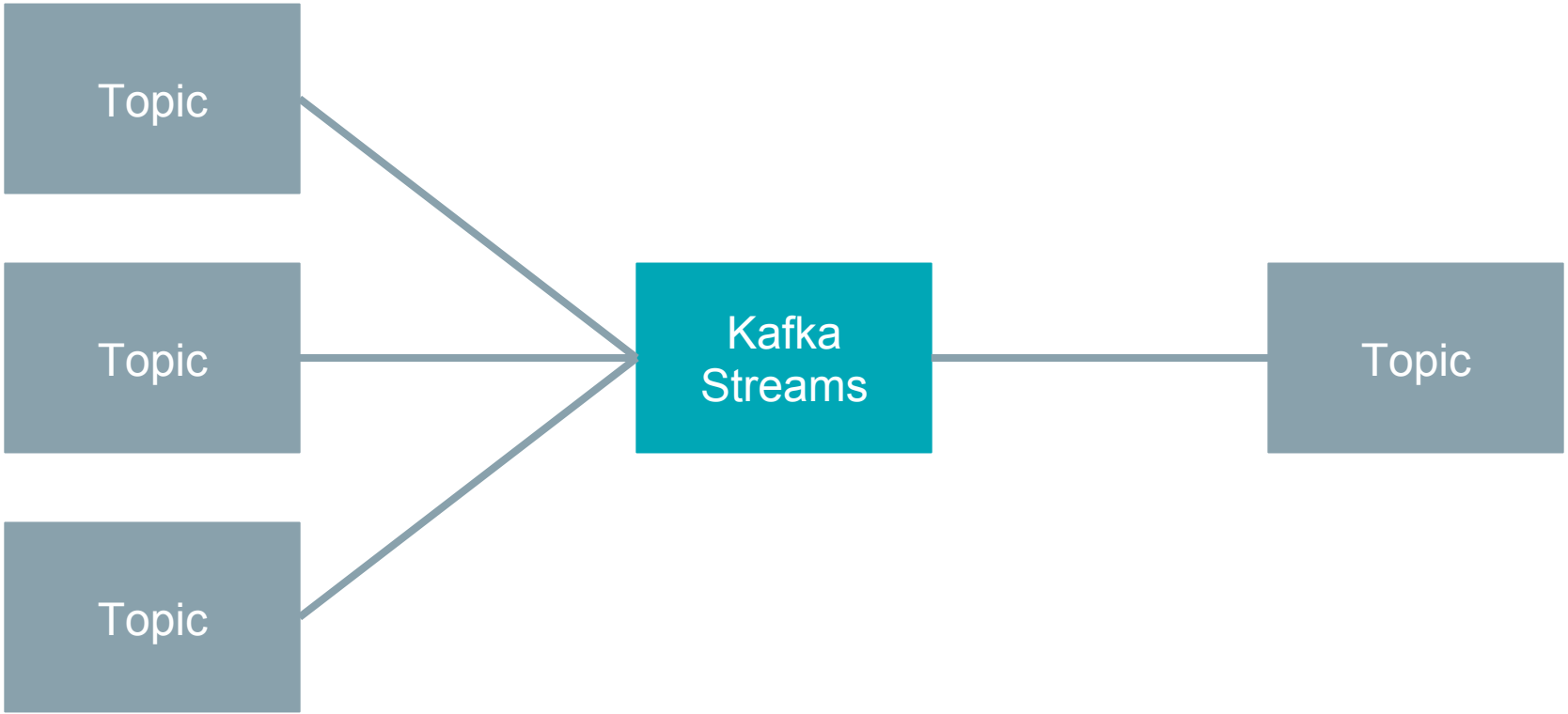
Kafka Connect

MySQL ⬢ Salesforce ⬢ Redis ⬢ MQTT ⬢ InfluxDB ⬢ RethinkDB ⬢ HBase ⬢ Solr ⬢ Couchbase ⬢ Elasticsearch ⬢ Hazelcast ⬢ Google PubSub ⬢ HDFS ⬢ S3 ⬢ Splunk ⬢ Spooldir ⬢ JDBC ⬢ Syslog ⬢ Cassandra ⬢ Vertica ⬢ DB2 ⬢ Goldengate ⬢ Jenkins ⬢ PredictionIO ⬢ JMS ⬢ Twitter ⬢ Attunity ⬢ MSSQL ⬢ Postgres ⬢ DynamoDB ⬢ IRC ⬢ Kudu ⬢ Ignite ⬢ MongoDB ⬢ Bloomberg Ticker ⬢ FTP

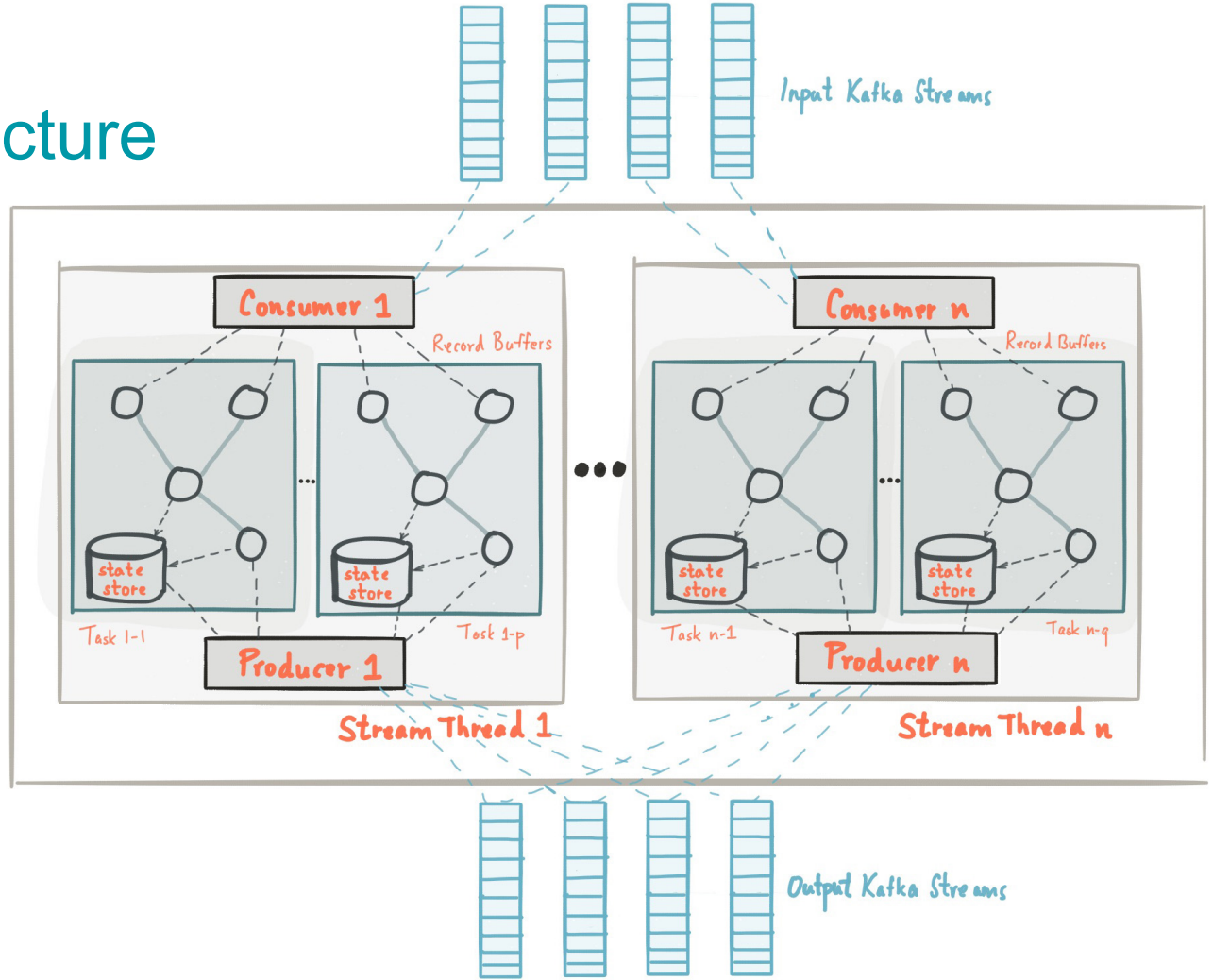
Kafka Streams

Processing streaming data

Kafka Streams



Architecture

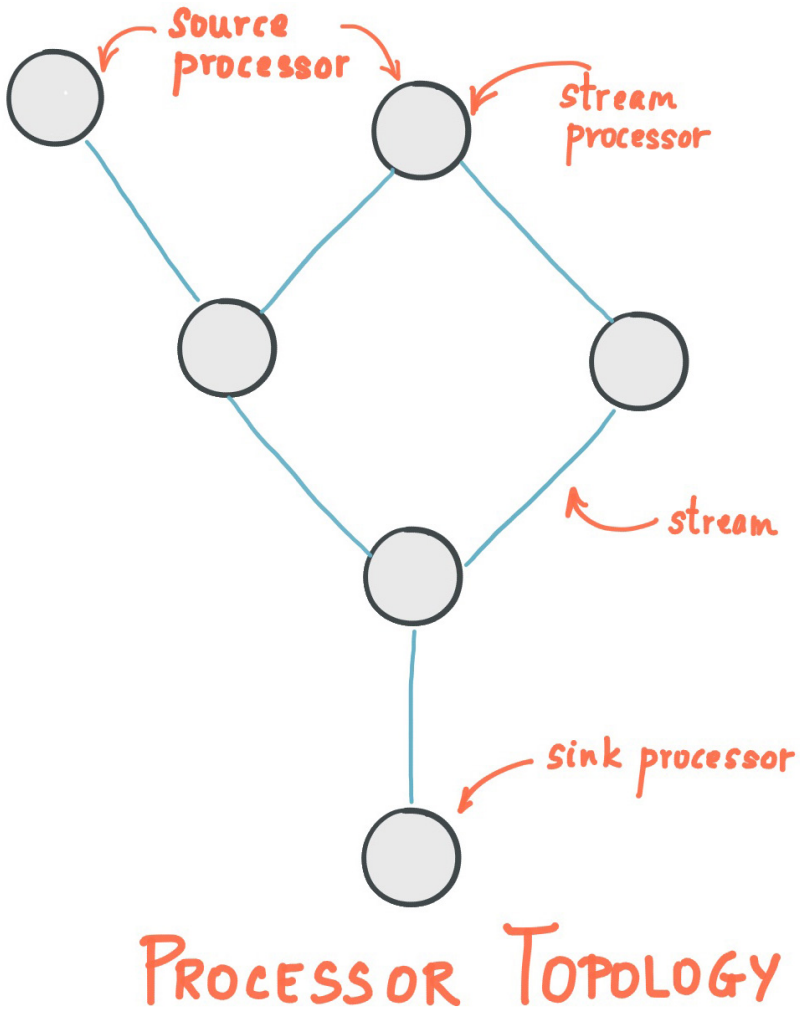


Highlights of Kafka Streams:

- Simple and lightweight client library
- Has no external dependencies on systems other than Apache Kafka itself.
- Supports fault-tolerant local state, which enables very fast and efficient stateful operations like windowed joins and aggregations.
- Employs one-record-at-a-time processing to achieve millisecond processing latency, and supports event-time based windowing operations with late arrival of records.

Stream Processing Topology

- ❑ Stream Processing Application
- ❑ Stream
- ❑ Processor
  - ✓ Source Processor
  - ✓ Sink Processor



Time

A critical aspect in stream processing is the notion of time, and how it is modeled and integrated. For example, some operations such as windowing are defined based on time boundaries.

Common notions of time in streams are:

- ❑ **Event time**
- ❑ **Ingestion time**
- ❑ **Processing time**

Kafka Streams

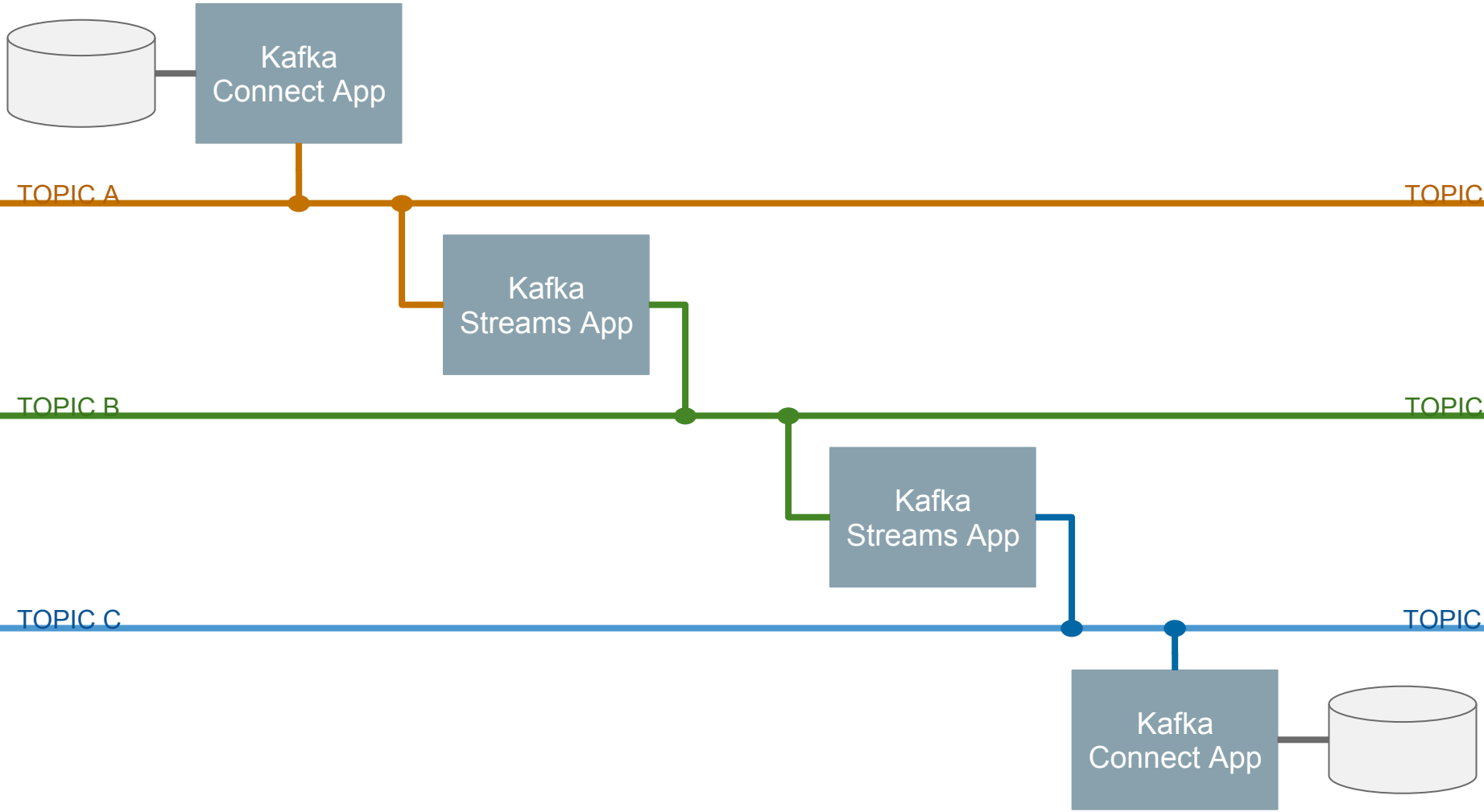
KStream for a stream of data

KTable to keep the latest value for each key

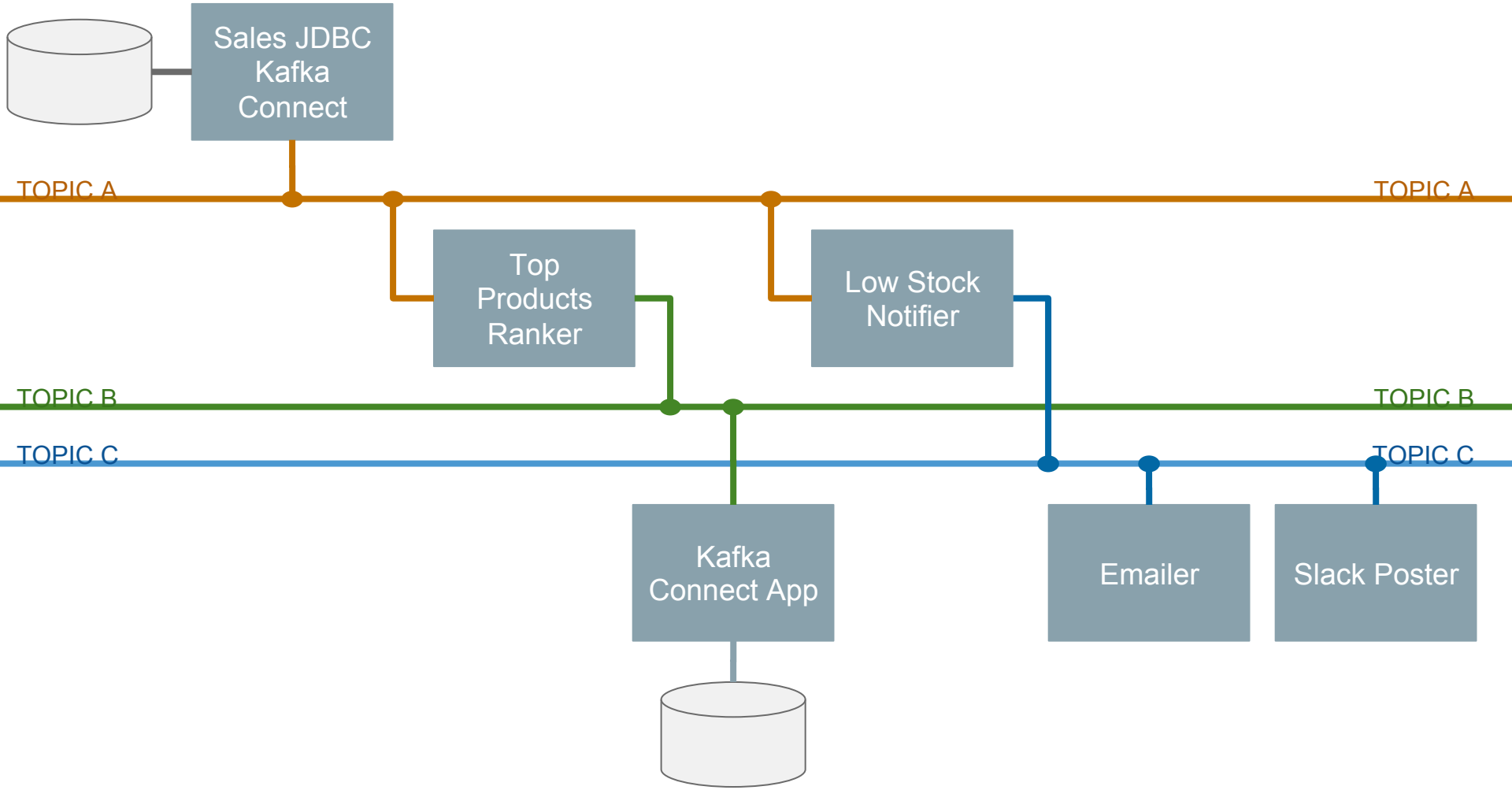
KTable state is distributed across app instances

Transform from streams to tables and tables to streams

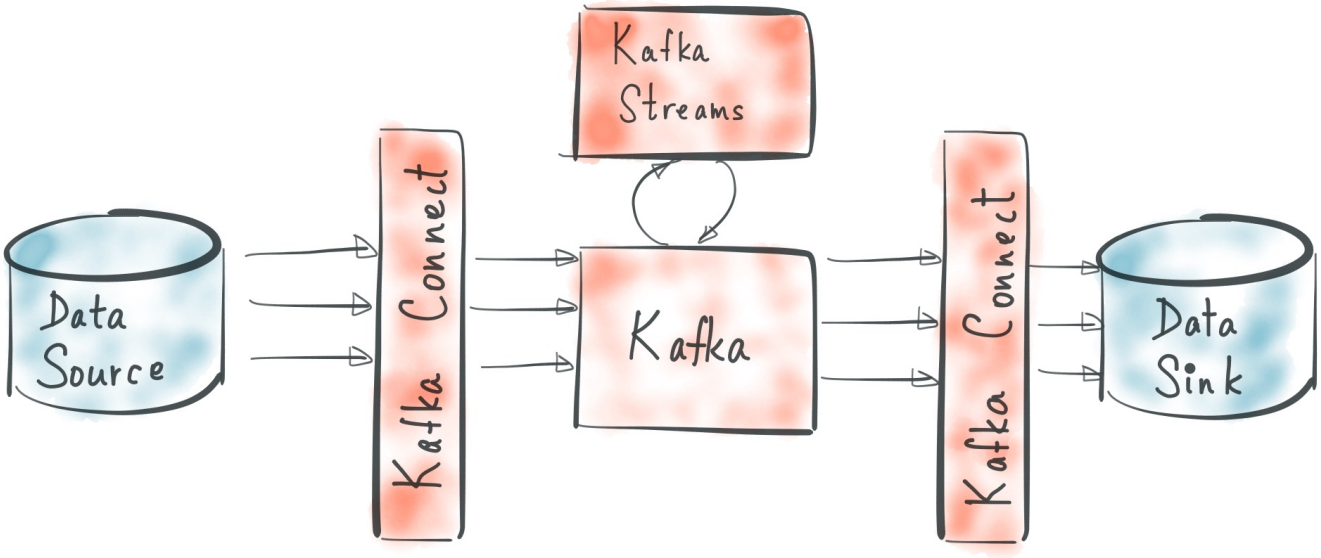
Choose which field to use as “timestamp”


Kafka
Connect App
TOPIC A
TOPIC A
Kafka
Streams App
TOPIC B
TOPIC B
Kafka
Streams App
TOPIC C
TOPIC C
Kafka
Connect App


Kafka
Kafka
Kafka
Kafka Connect
Kafka Streams
Kafka Connect


Sales JDBC Kafka Connect
TOPIC A
TOPIC A
Top Products Ranker
Low Stock Notifier
TOPIC B
TOPIC B
TOPIC C
TOPIC C
Kafka Connect App
Emailer
Slack Poster

Kafka Connect + Streams



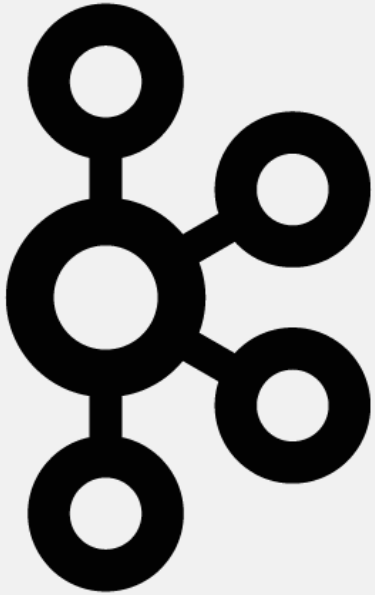
Thanks for your attention

References

Kafka Best Practices - https://community.hortonworks.com/articles/80813/kafka-best-practices-1.html

Performance - https://www.cloudera.com/documentation/kafka/latest/topics/kafka_performance.html

Questions?



https://github.com/nmvijay/Kafka-Training