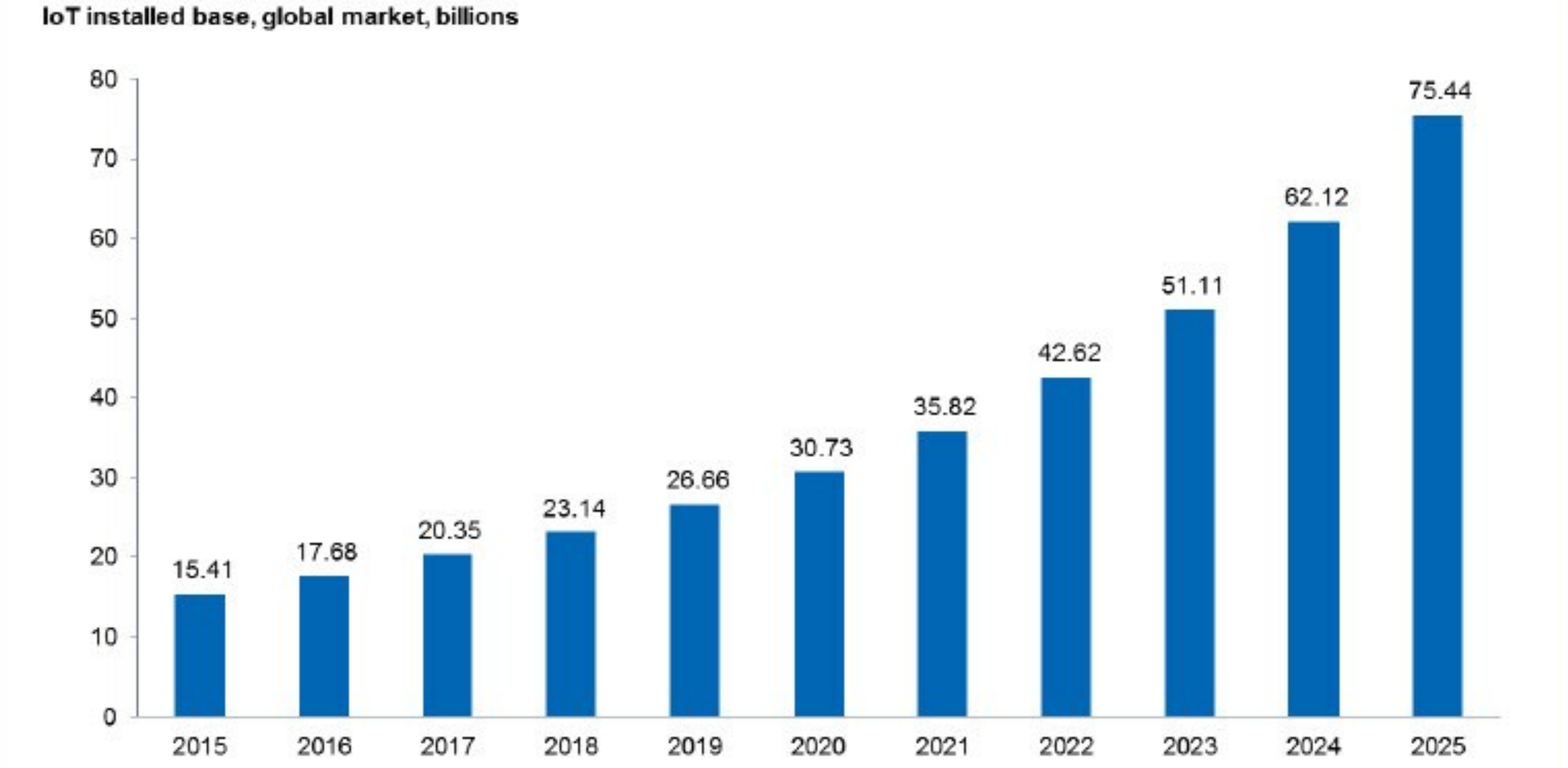
Data Streaming

Silviu Marcu
https://github.com/smarcu/datastreaming-presentation


BRACE YOURSELVES
MORE CHANGE IS COMING

Figure 1. The IoT market will be massive



Source: IHS

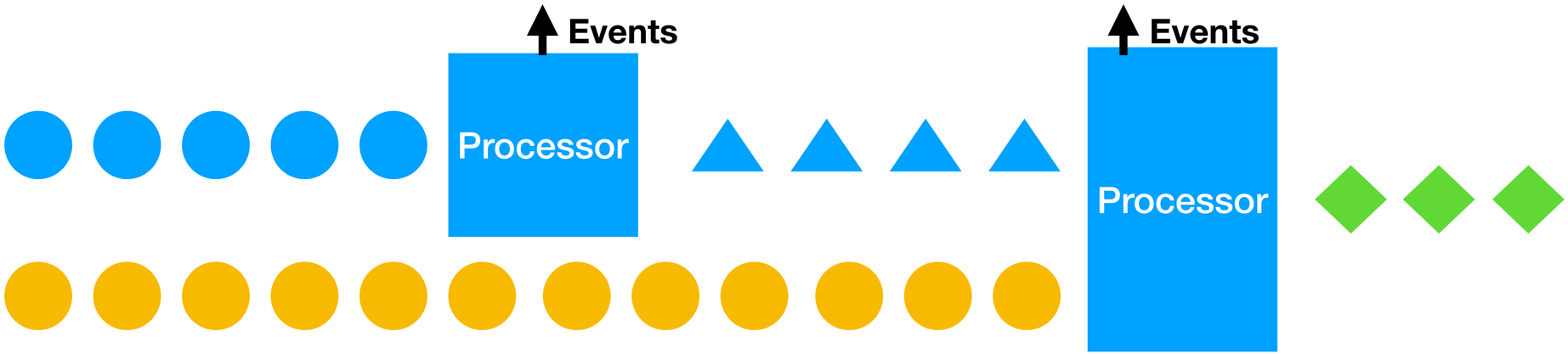
Data Streaming

Data continuously generated by a high number of data sources, which typically is sent simultaneously, in small sizes.

Log events generated by customers using your mobile or web applications
Ecommerce purchases
In-game player activity
Information from social networks
Financial trading floors
Geospatial services, and telemetry
Instrumentation in data centers

Stream Processing

Stream Processing defines the continuous processing applied on data streams, producing instant analytics, transformations or trigger events



Streaming Data Examples

Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance, detects any potential defects in advance, and places a spare part order automatically preventing equipment down time

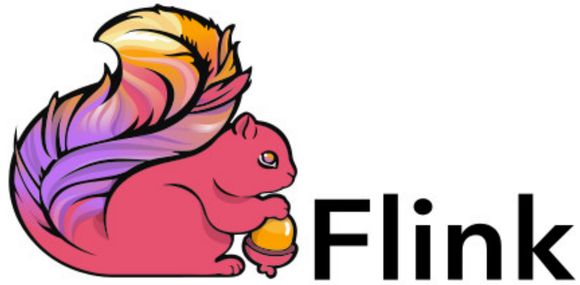
Streaming Data Examples

A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements

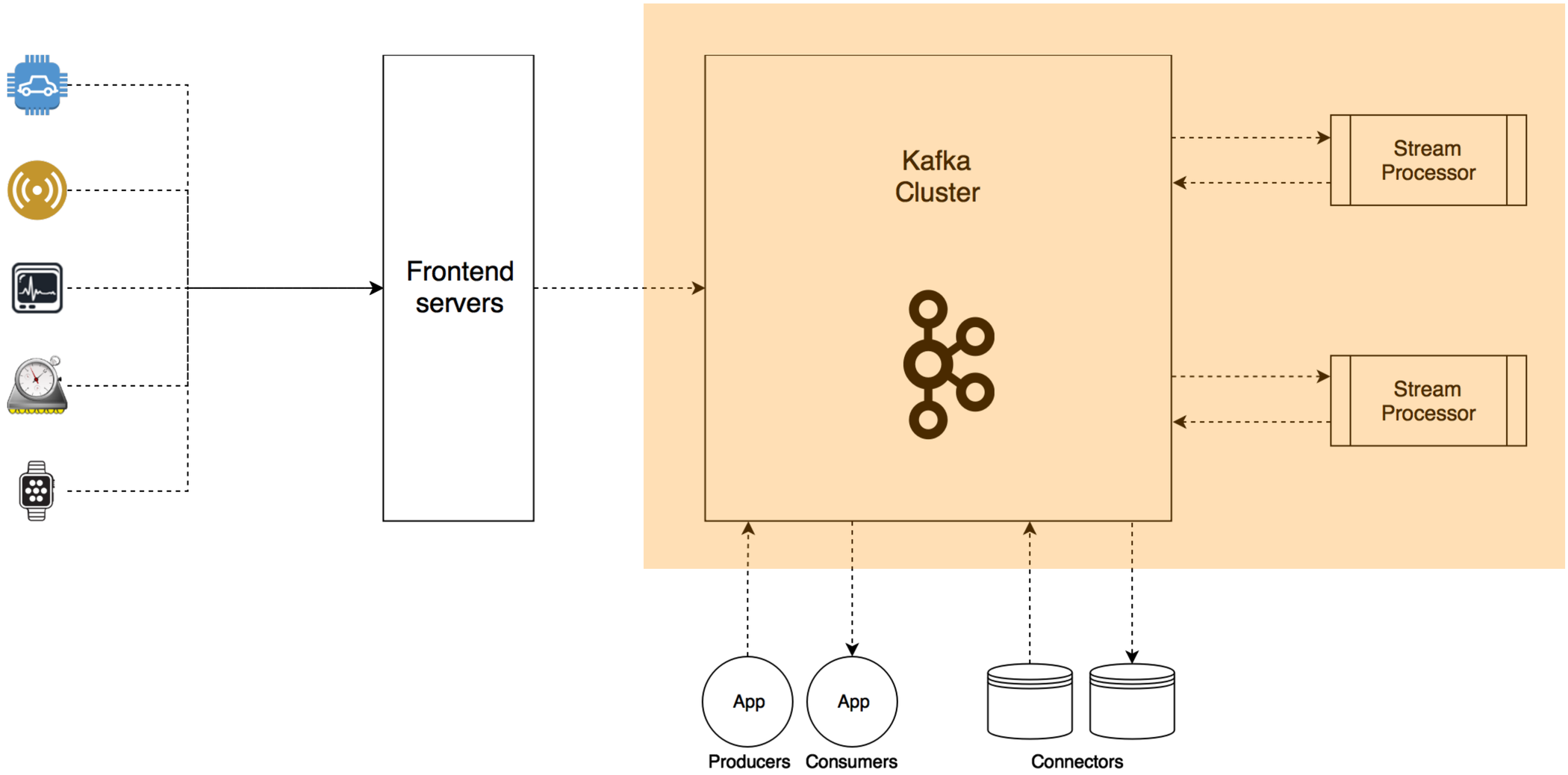
Streaming Data Examples

Realtime biosignals monitoring and detection

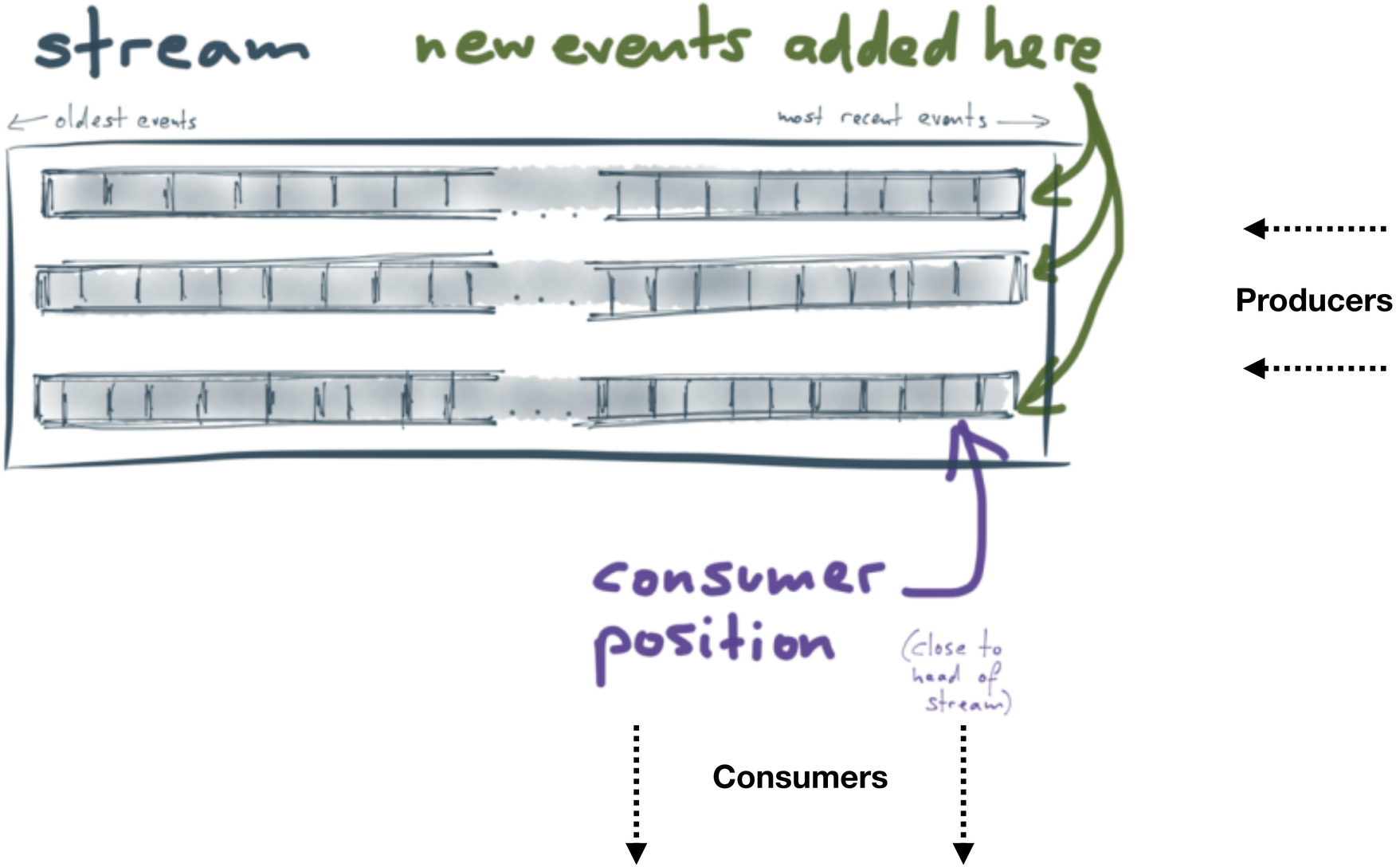
Streaming Frameworks



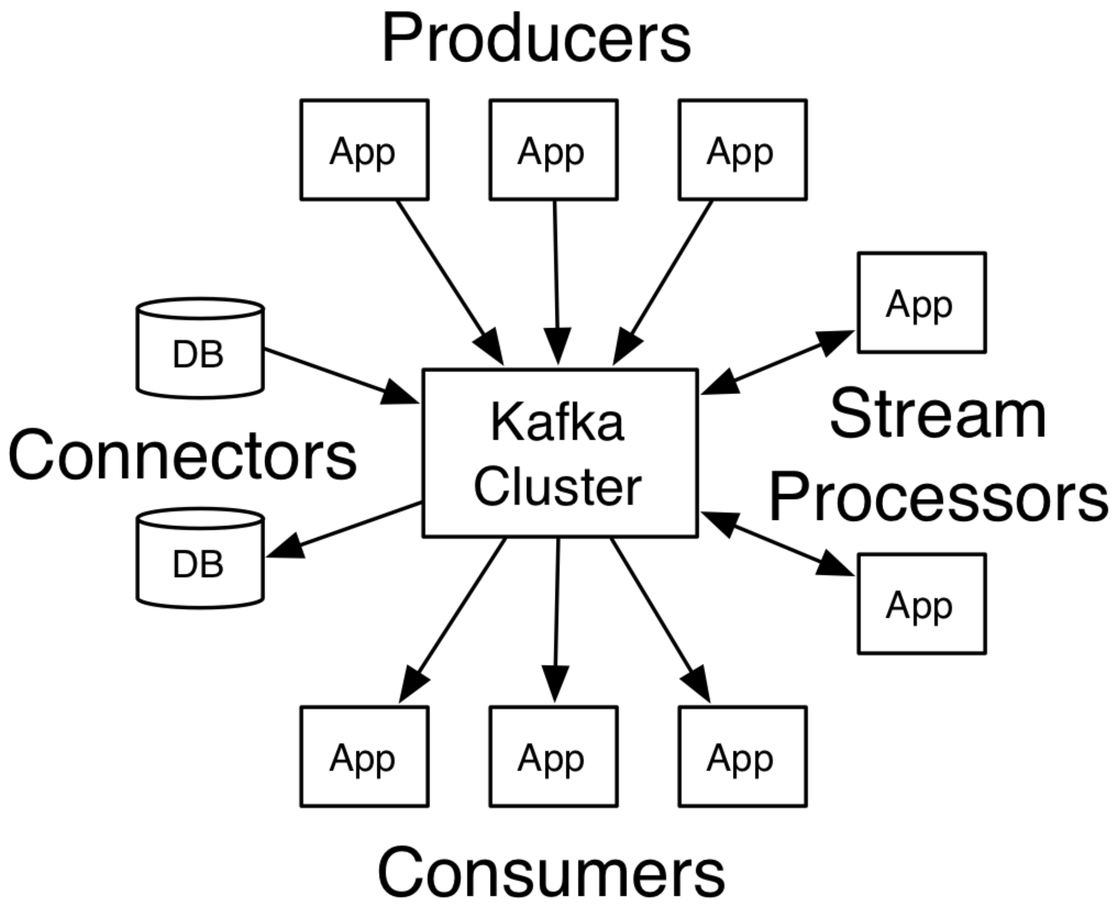
Stream Processing Architecture



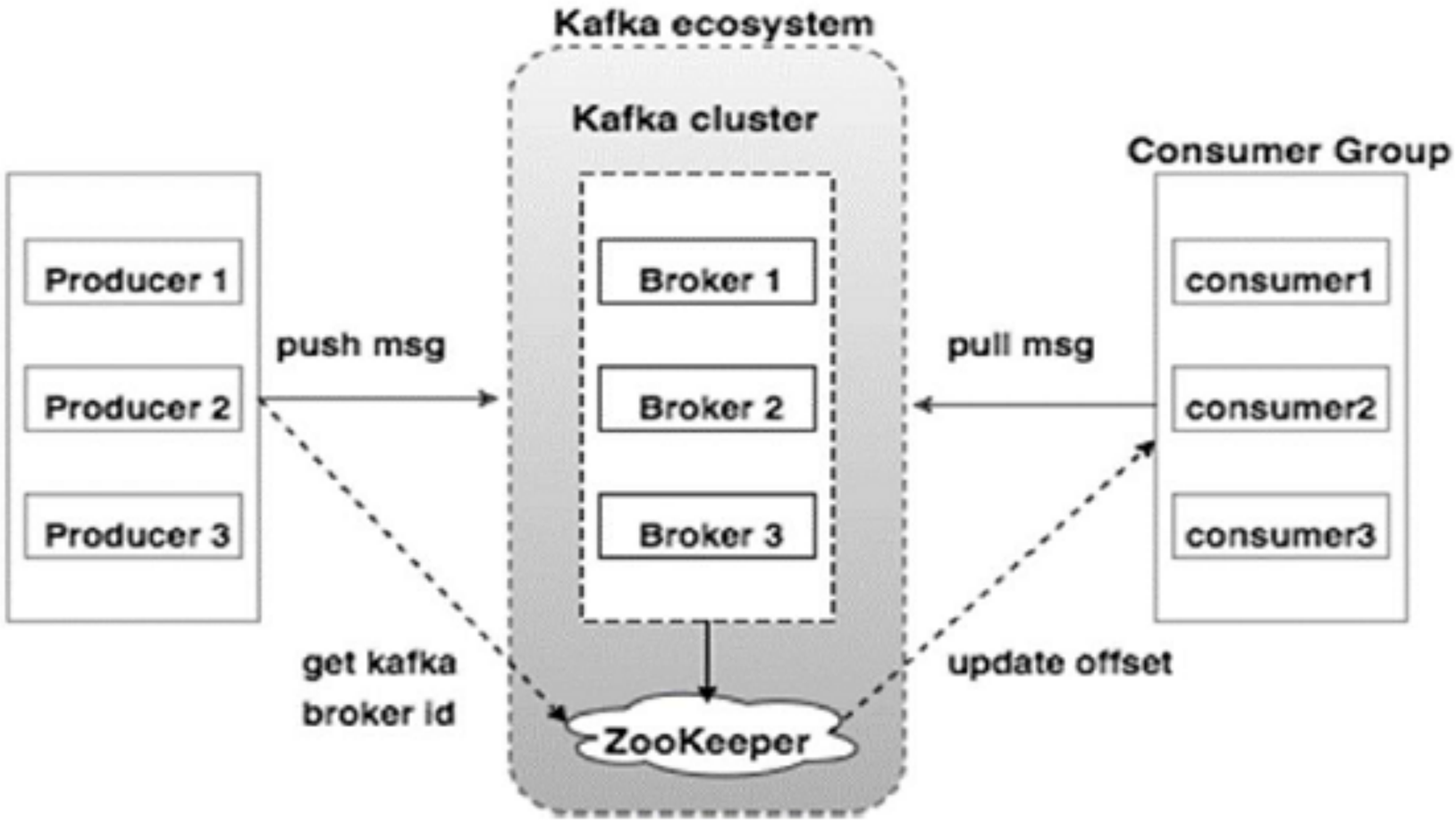
APACHE kafka
A distributed streaming platform



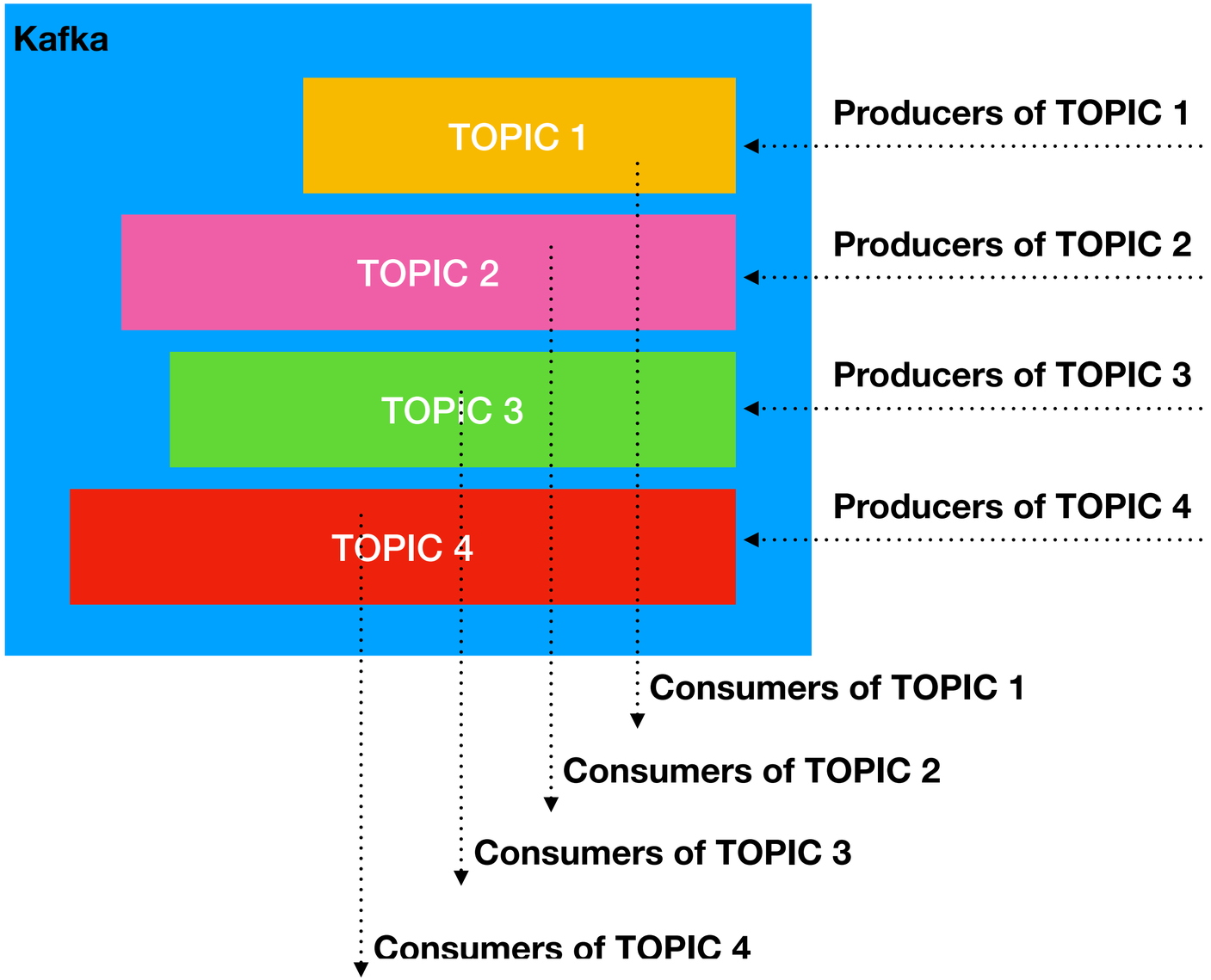
APACHE kafka®
A distributed streaming platform



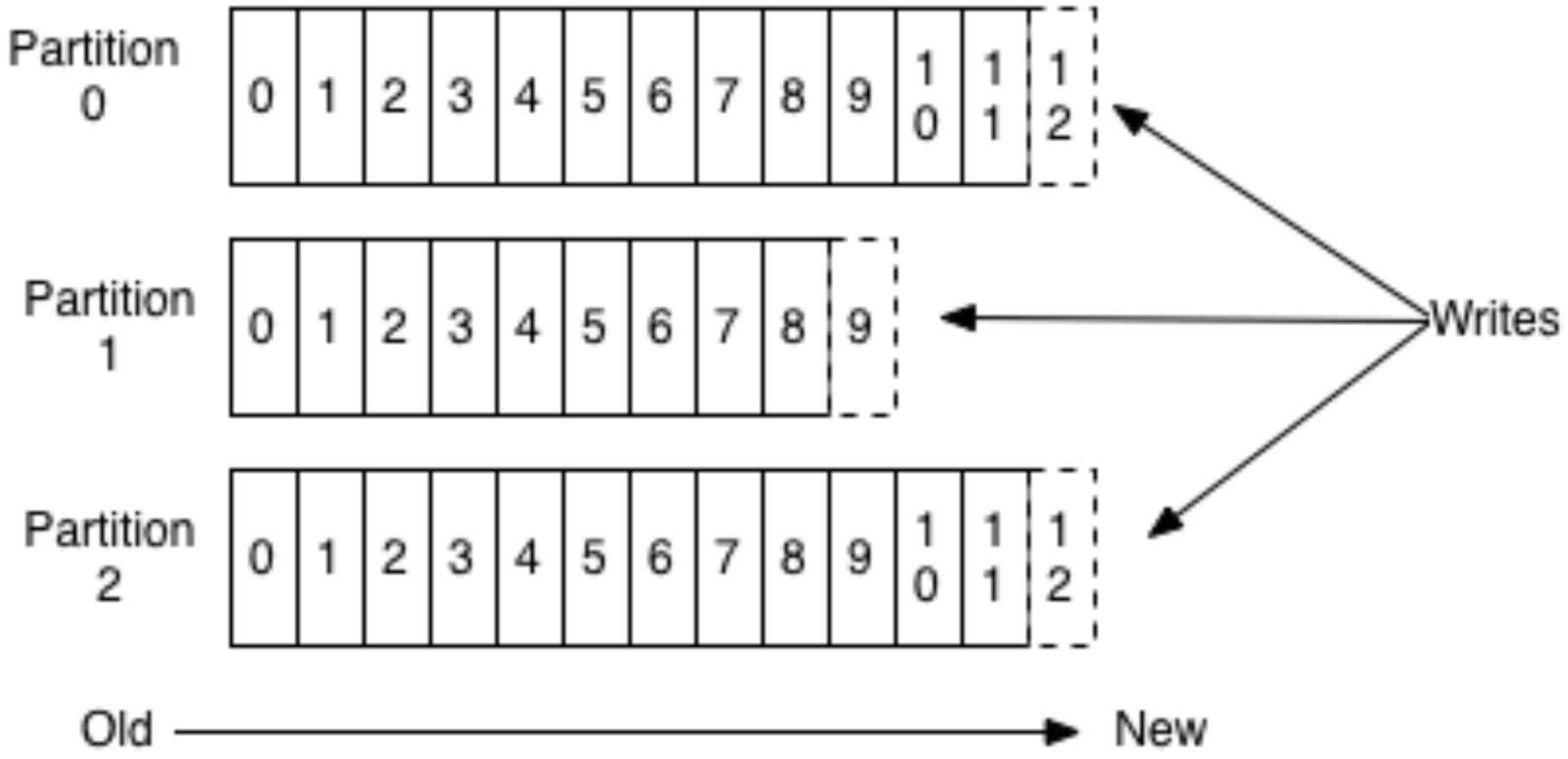
Kafka Cluster



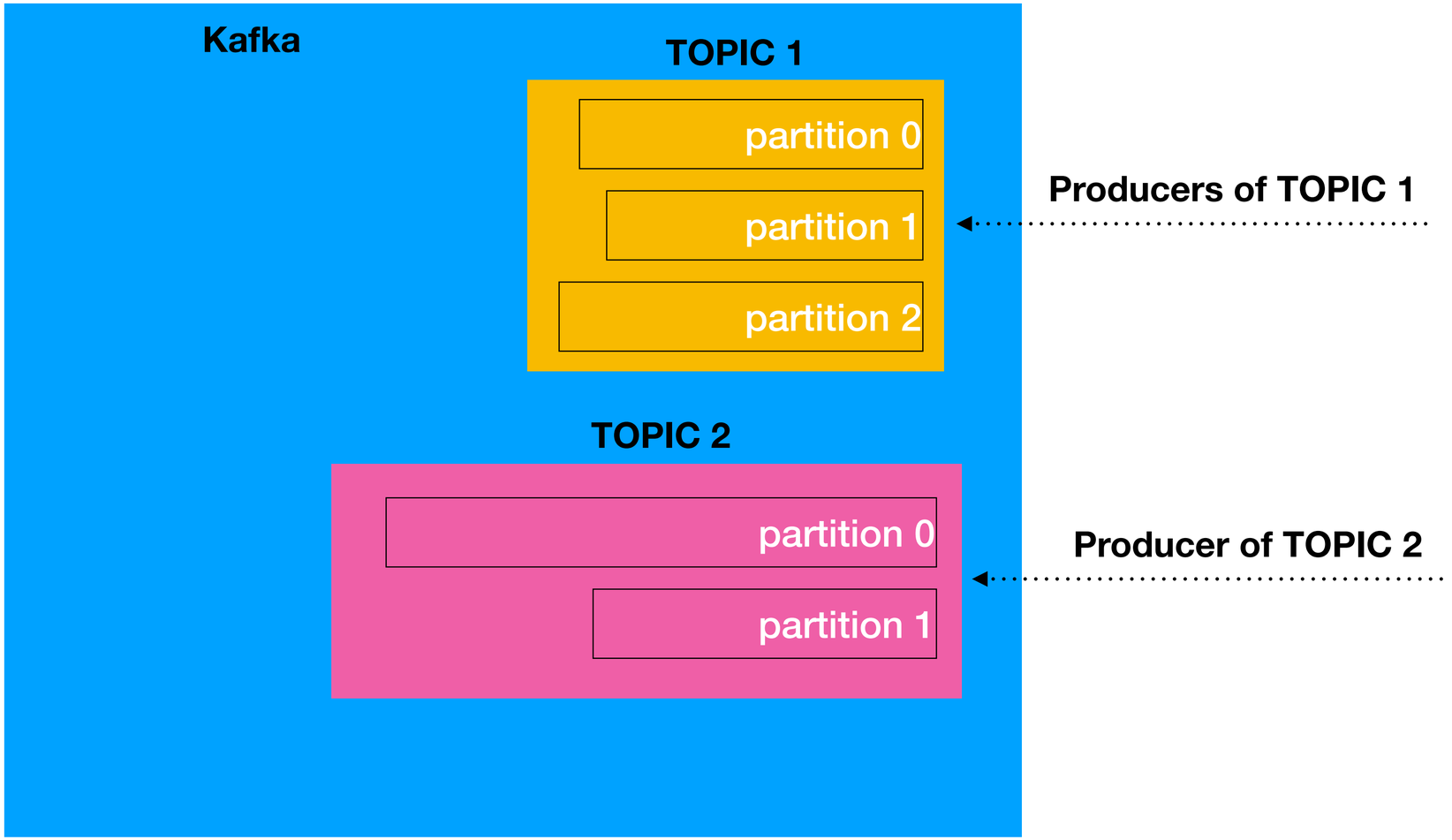
Topic


Kafka
TOPIC 1
TOPIC 2
TOPIC 3
TOPIC 4
Producers of TOPIC 1
Producers of TOPIC 2
Producers of TOPIC 3
Producers of TOPIC 4
Consumers of TOPIC 1
Consumers of TOPIC 2
Consumers of TOPIC 3
Consumers of TOPIC 4

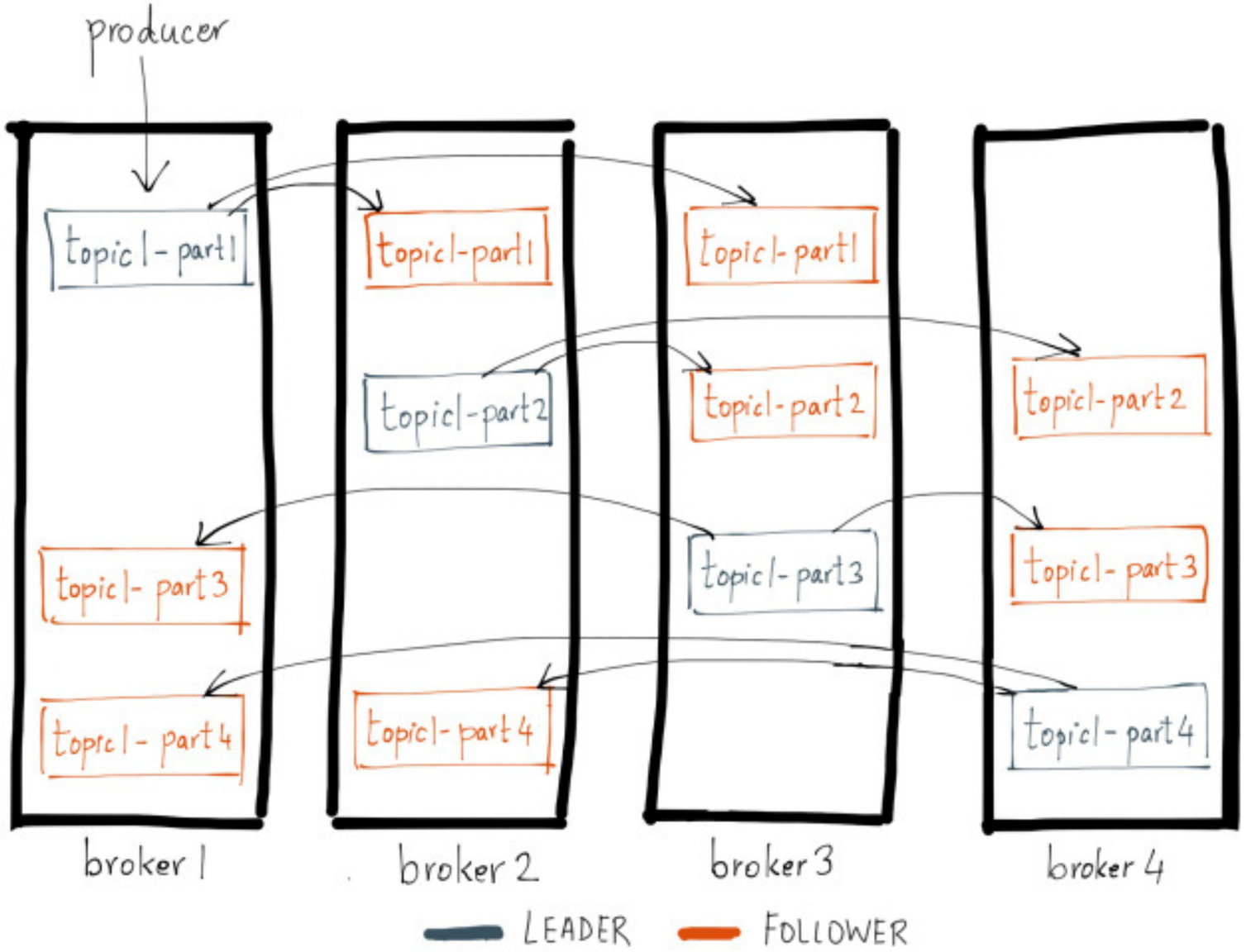
Anatomy of a Topic



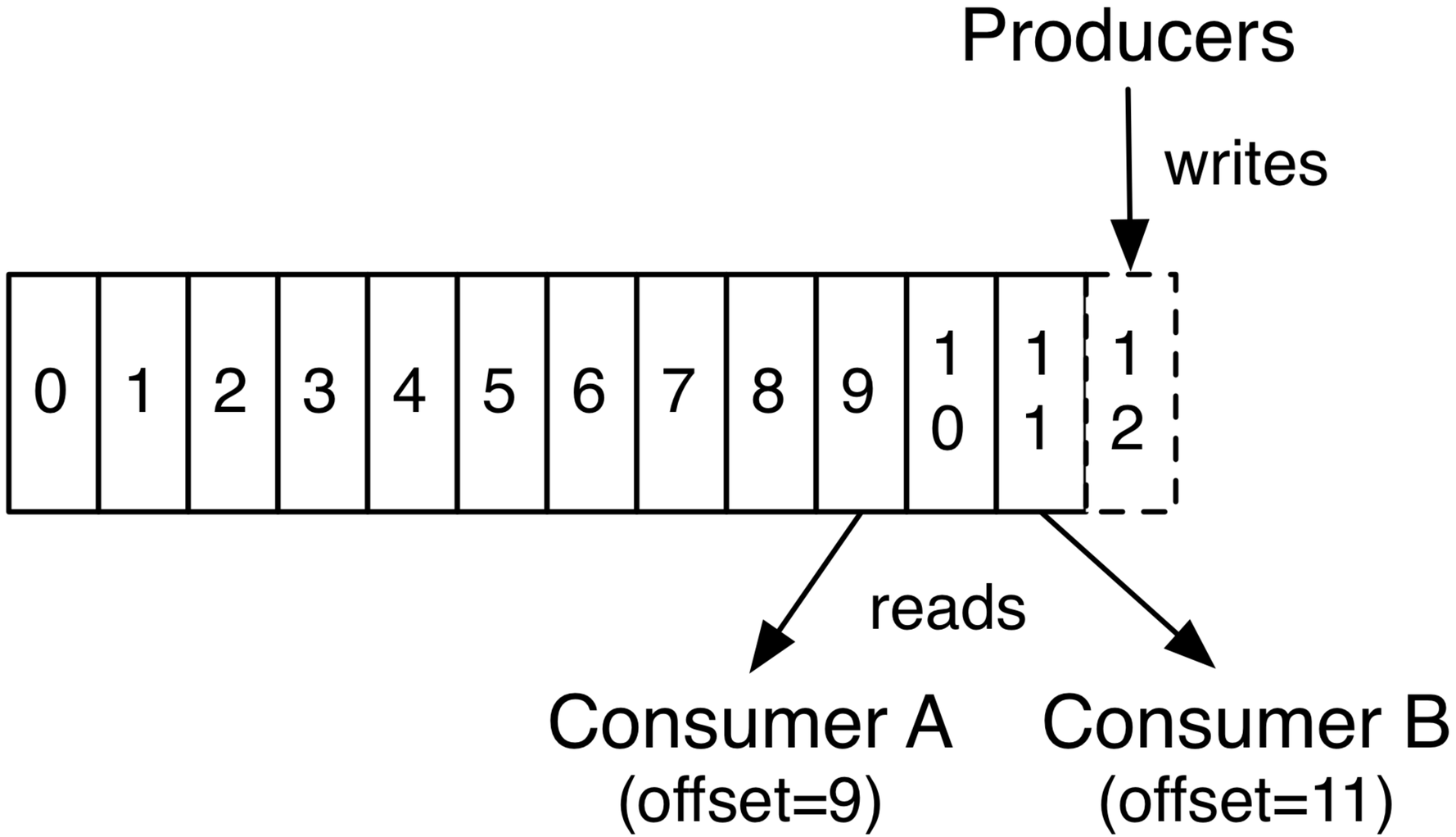
Topic / Partitions



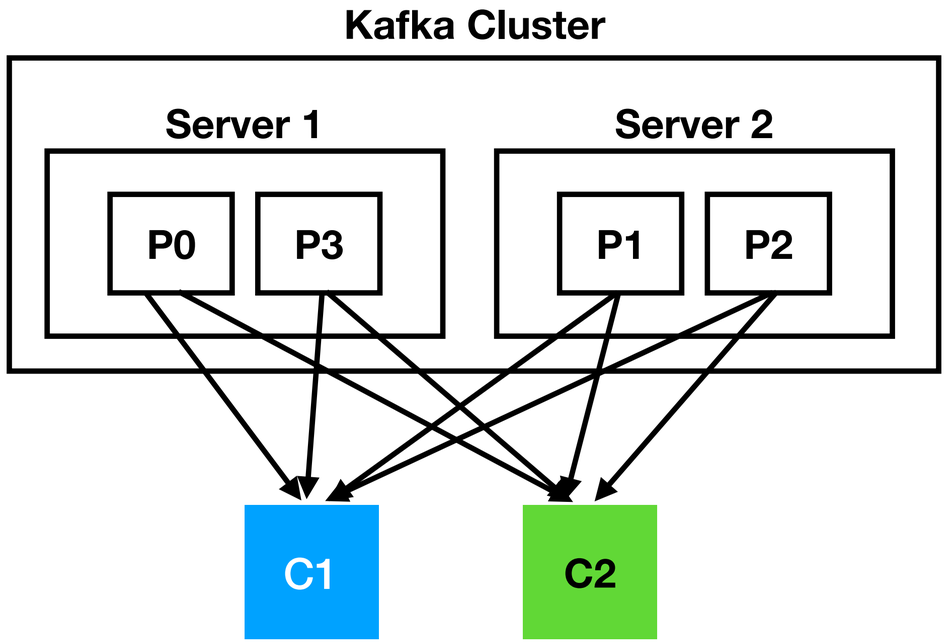
Topic Replication



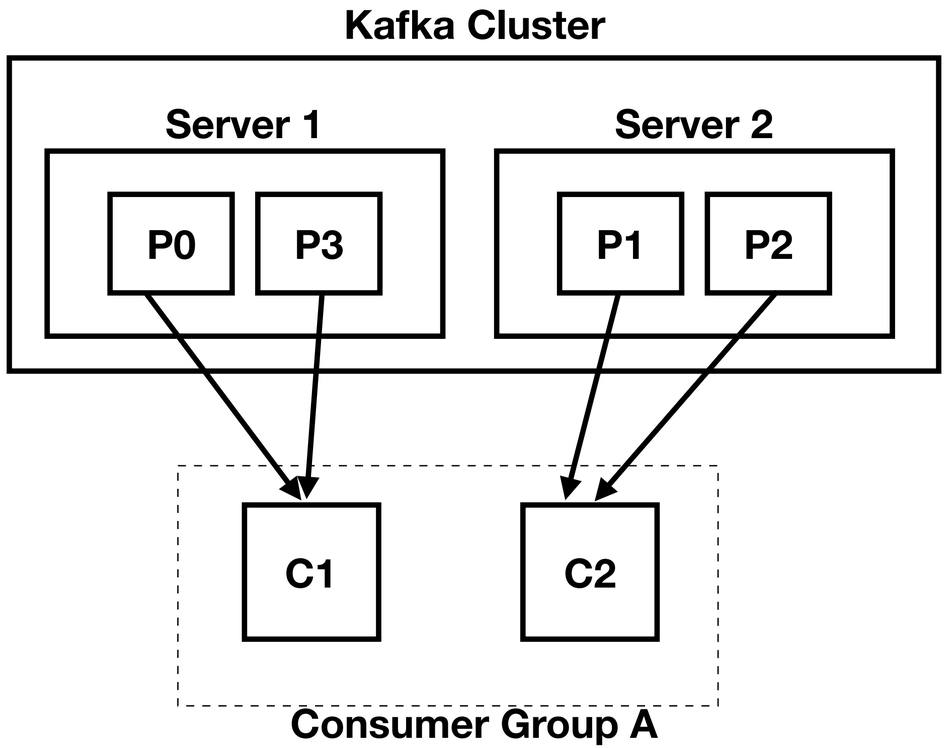
Topic Consumer



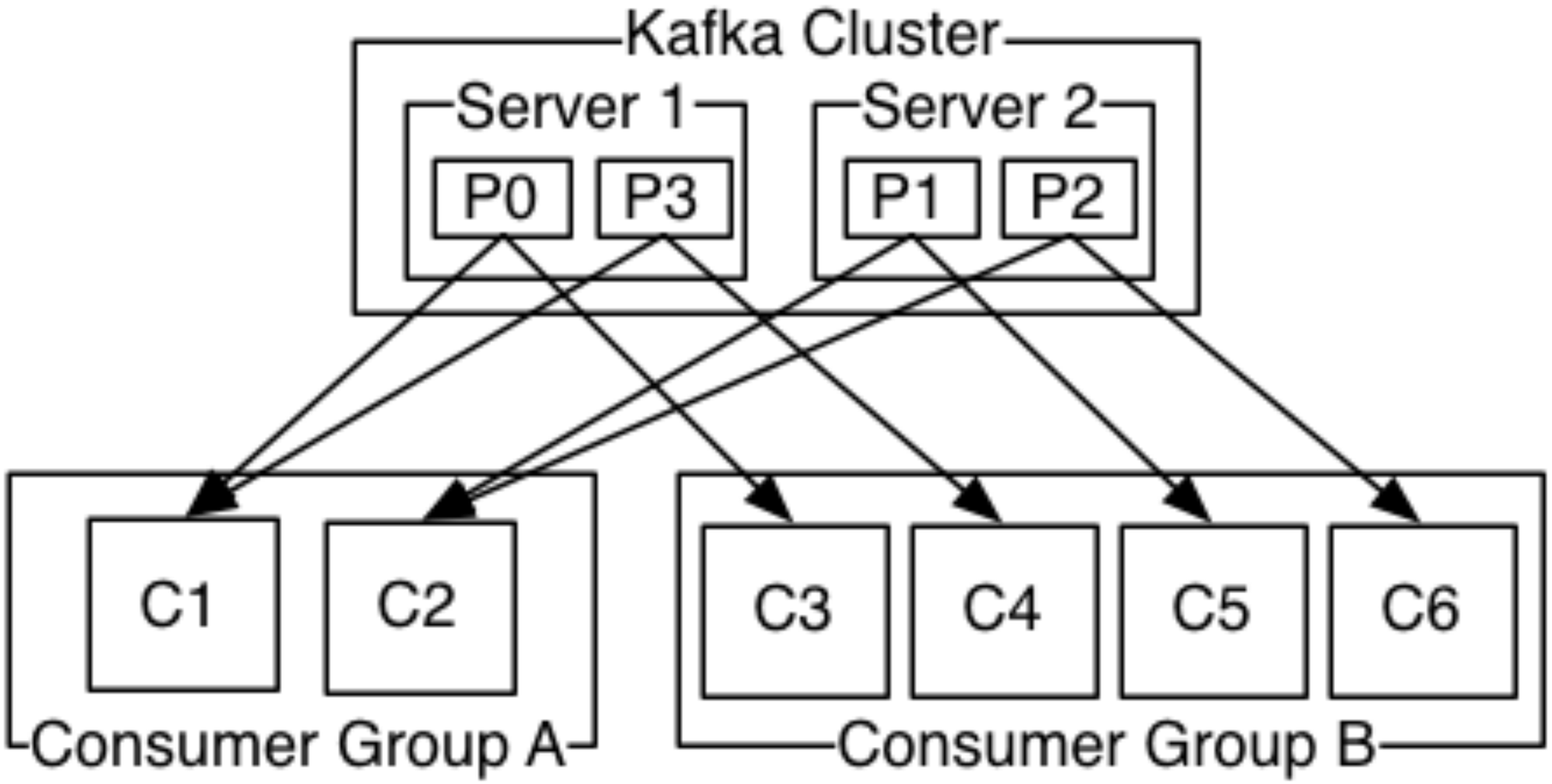
Distribution



Distribution



Distribution

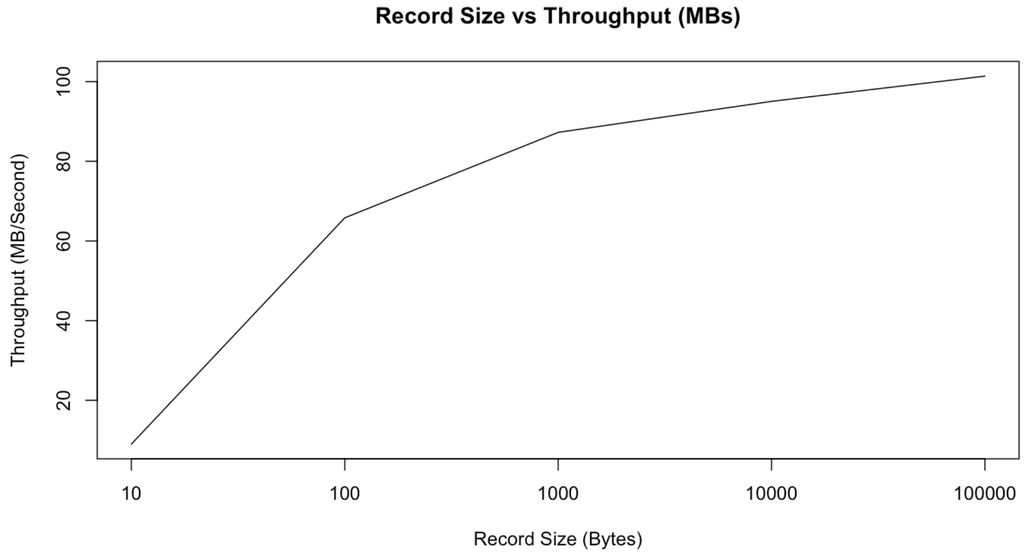


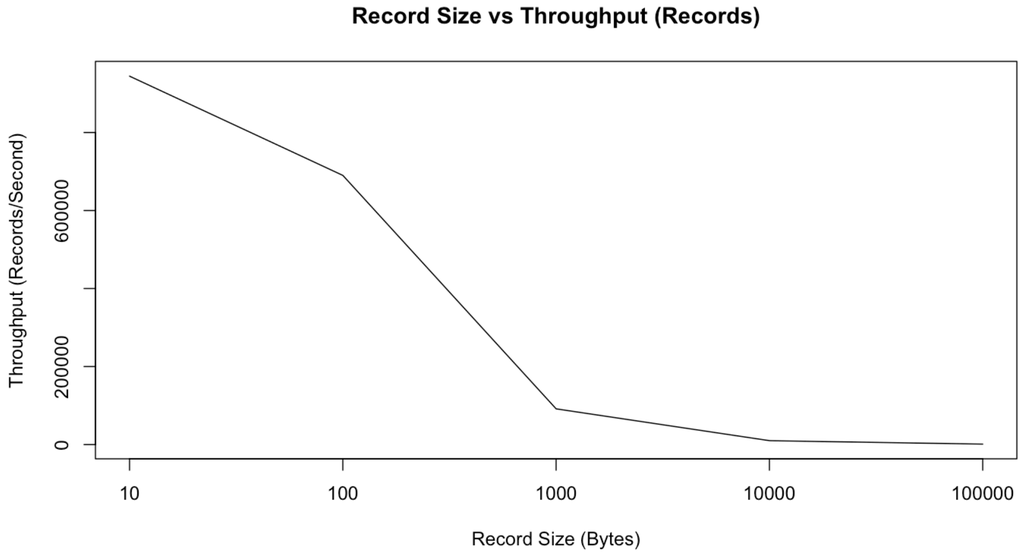
Kafka Performance

6 commodity machines (Xeon 2.5 GHz six cores, Six 7200 RPM SATA drives, 32GB of RAM, 1Gb Ethernet)

<u>3 x Nodes Kafka cluster</u>, 3 x zookeeper and consumers/producers

3 producers, 3x async replication	3 Consumers	Producer and Consumer	End-to-end Latency
2,024,032 records/sec	2,615,968 records/sec	795,064 records/sec	2 ms (median)
(193.0 MB/sec)	(249.5 MB/sec)	(75.8 MB/sec)	3 ms (99th percentile) 14 ms (99.9th percentile)



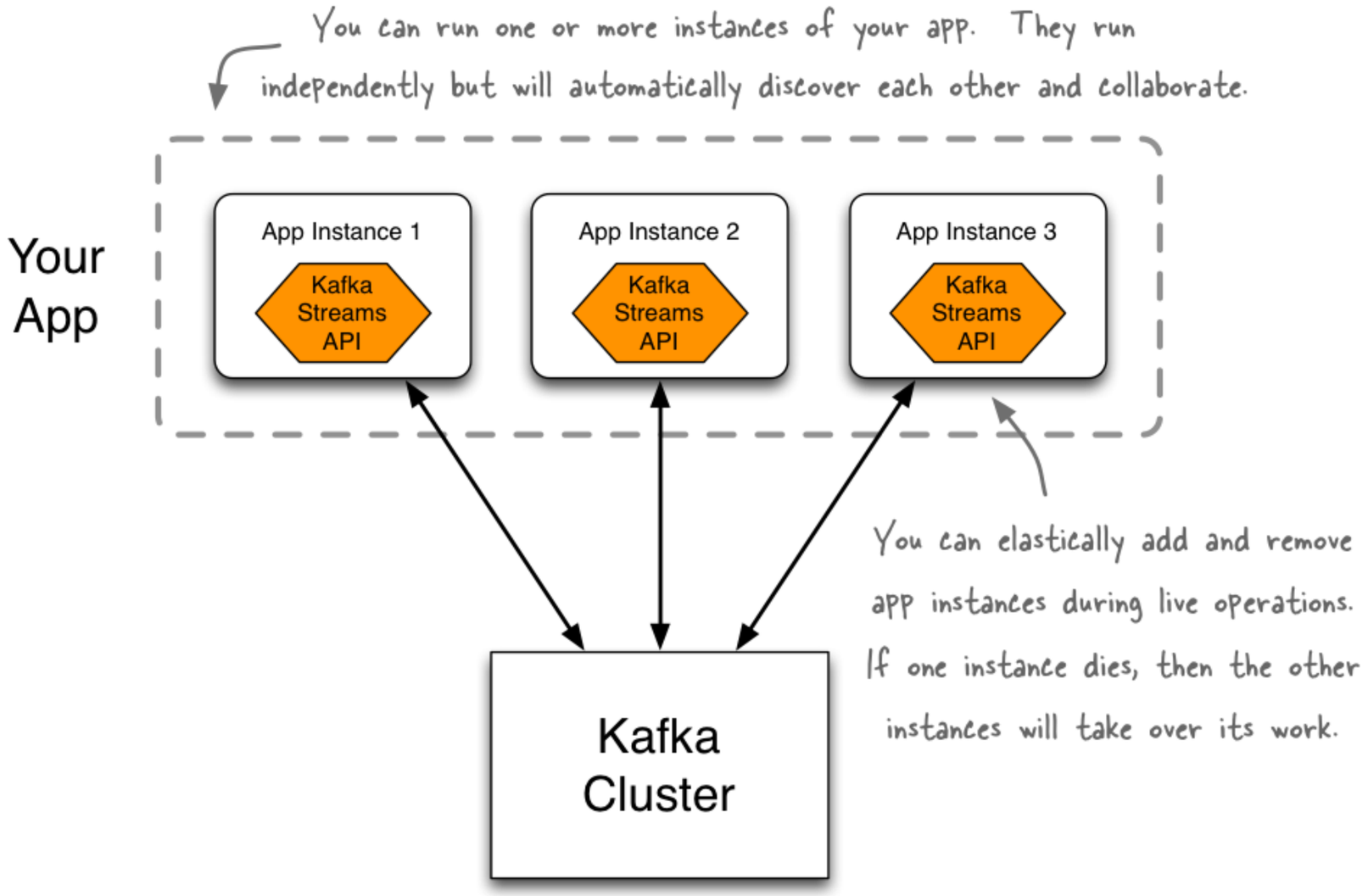


https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

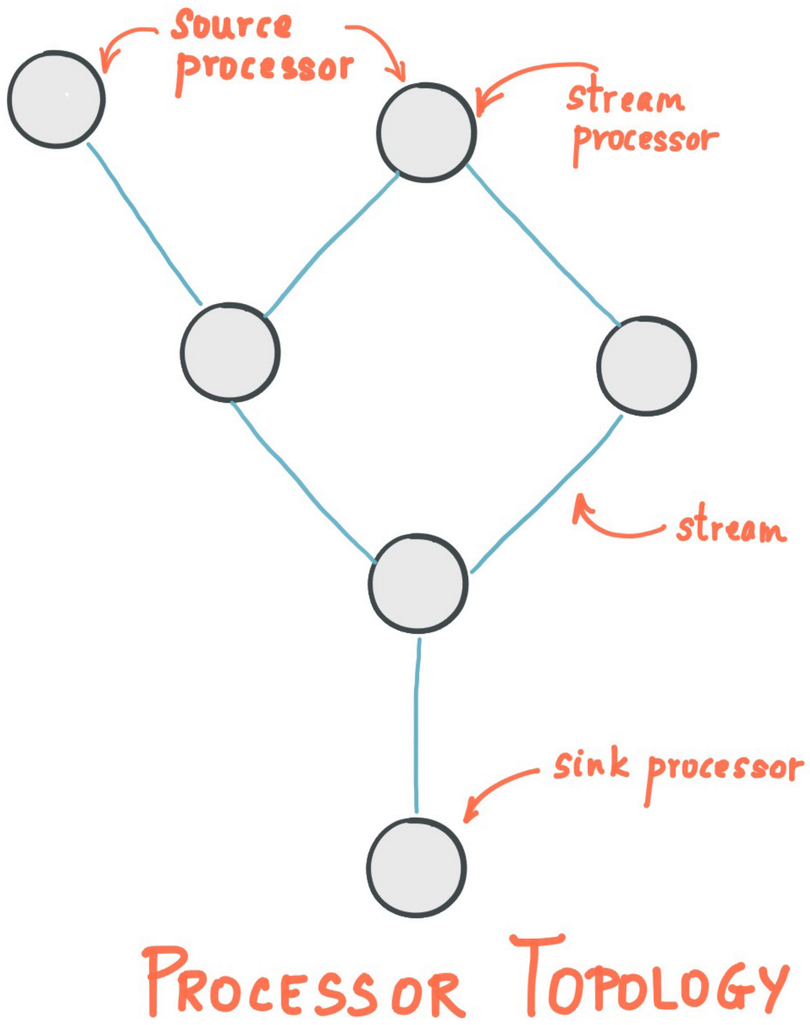
Kafka Stream Processing

- Stream Processing Code runs as regular code (compared to spark)
- A convenient DSL
- Stateless and Stateful processing including distributed joins and aggregations
- Windowing
- Supports exactly-once semantic
- Distributed processing and fault-tolerance with fast failover

Kafka Stream Processing - regular app



Stream Processor Topology



define topology using:

- KafkaStream DSL
- Processor API (low level)

Kafka Stream DSL

```
StreamsBuilder builder = new StreamsBuilder();

builder.<String, String>stream("streams-plaintext-input")
       .flatMapValues(value -> Arrays.asList(value.toLowerCase(Locale.getDefault()).split("\\W+")))
       .groupBy((key, value) -> value)
       .count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("counts-store"))
       .toStream()
       .to("streams-wordcount-output", Produced.with(Serdes.String(), Serdes.Long()));

Topology topology = builder.build();
KafkaStreams streams = new KafkaStreams(topology, props);

. . . . .

streams.start();
```

Kafka Stream DSL

Stateless Transformations

Branch: KStream → KStream
Filter: KStream → KStream or KTable → KTable
FlatMap: KStream → KStream
Foreach: KStream → void
GroupByKey: KStream → KGroupedStream
GroupBy: KStream → KGroupedStream or KTable → KGroupedTable
Map: KStream → KStream

Stateful Transformations

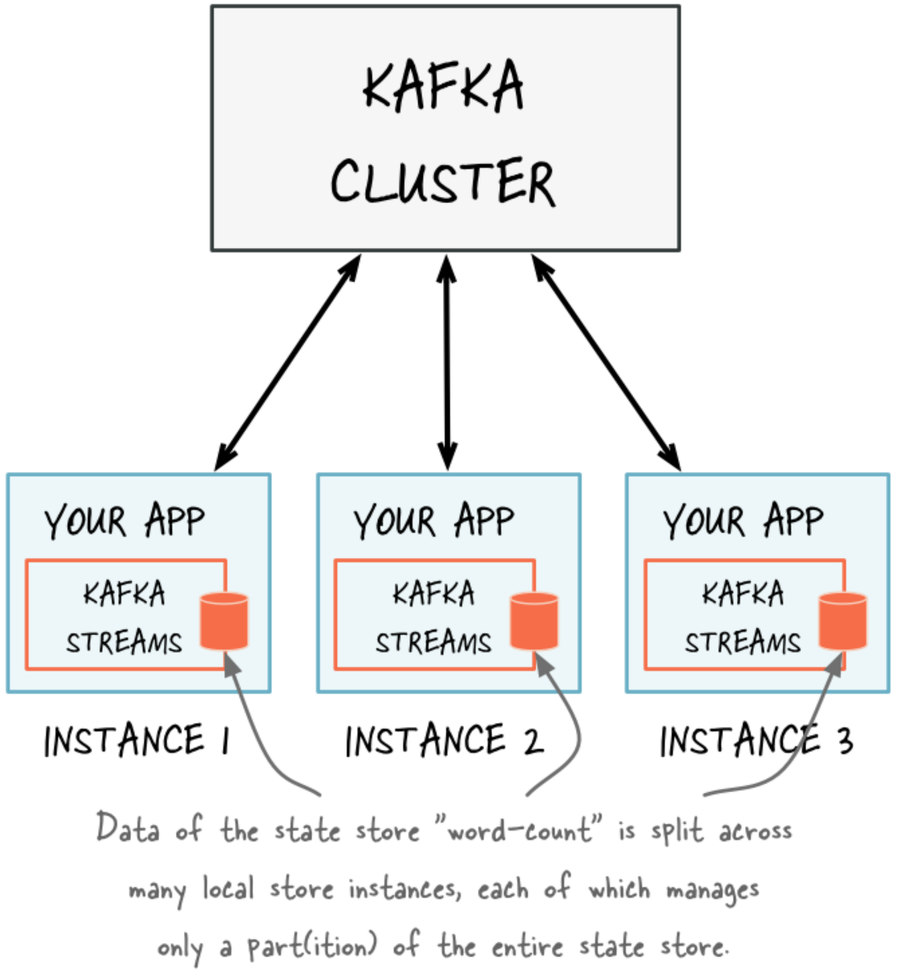
Aggregate: KGroupedStream → KTable or KGroupedTable → KTable
Aggregate (windowed): KGroupedStream → KTable
Count (Windowed): KGroupedStream → KTable
Reduce: KGroupedStream → KTable or KGroupedTable → KTable

Kafka Stream State

Some stream processing applications don't require state, which means the processing of a message is independent from the processing of all other messages.

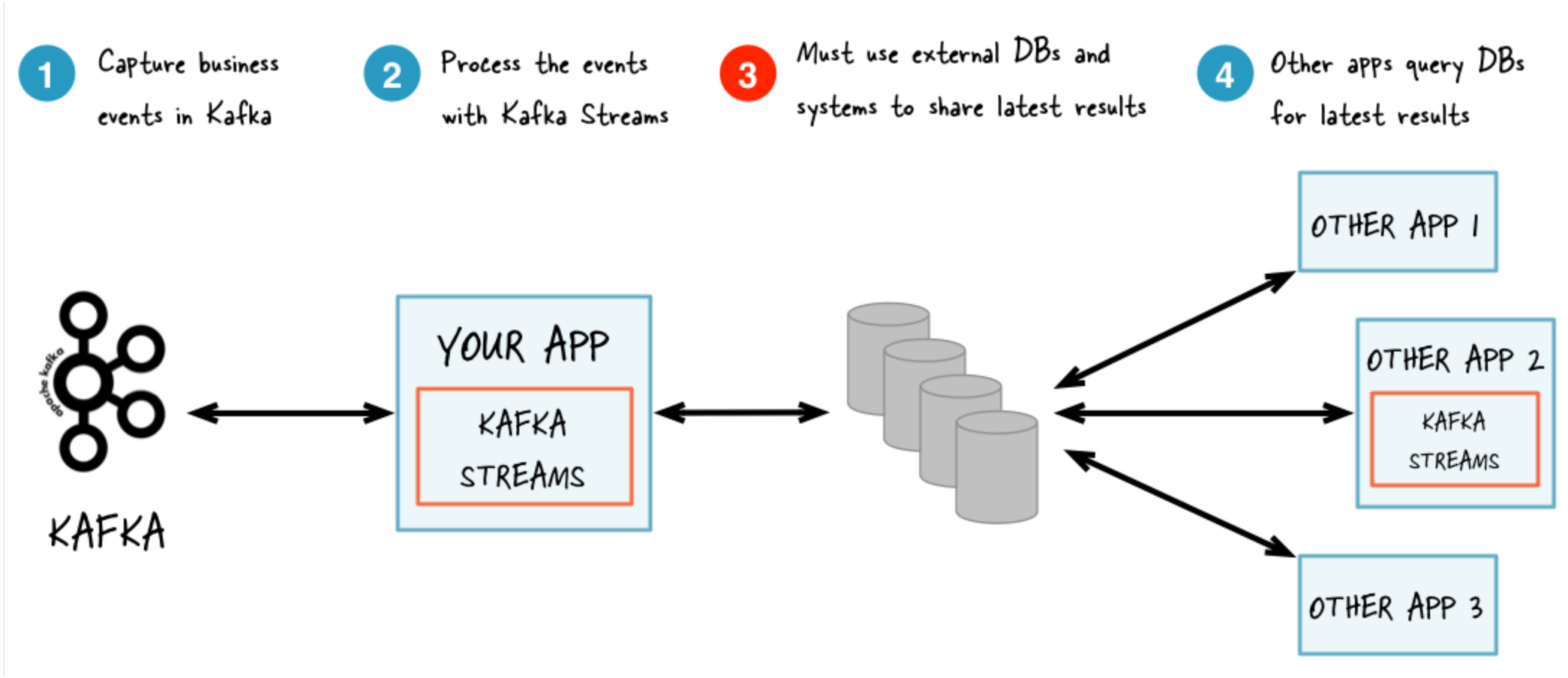
However, being able to maintain state opens up many possibilities for sophisticated stream processing applications: you can join input streams, or group and aggregate data records. Many such stateful operators are provided by the Kafka Streams DSL.

Kafka Stream (stateful)



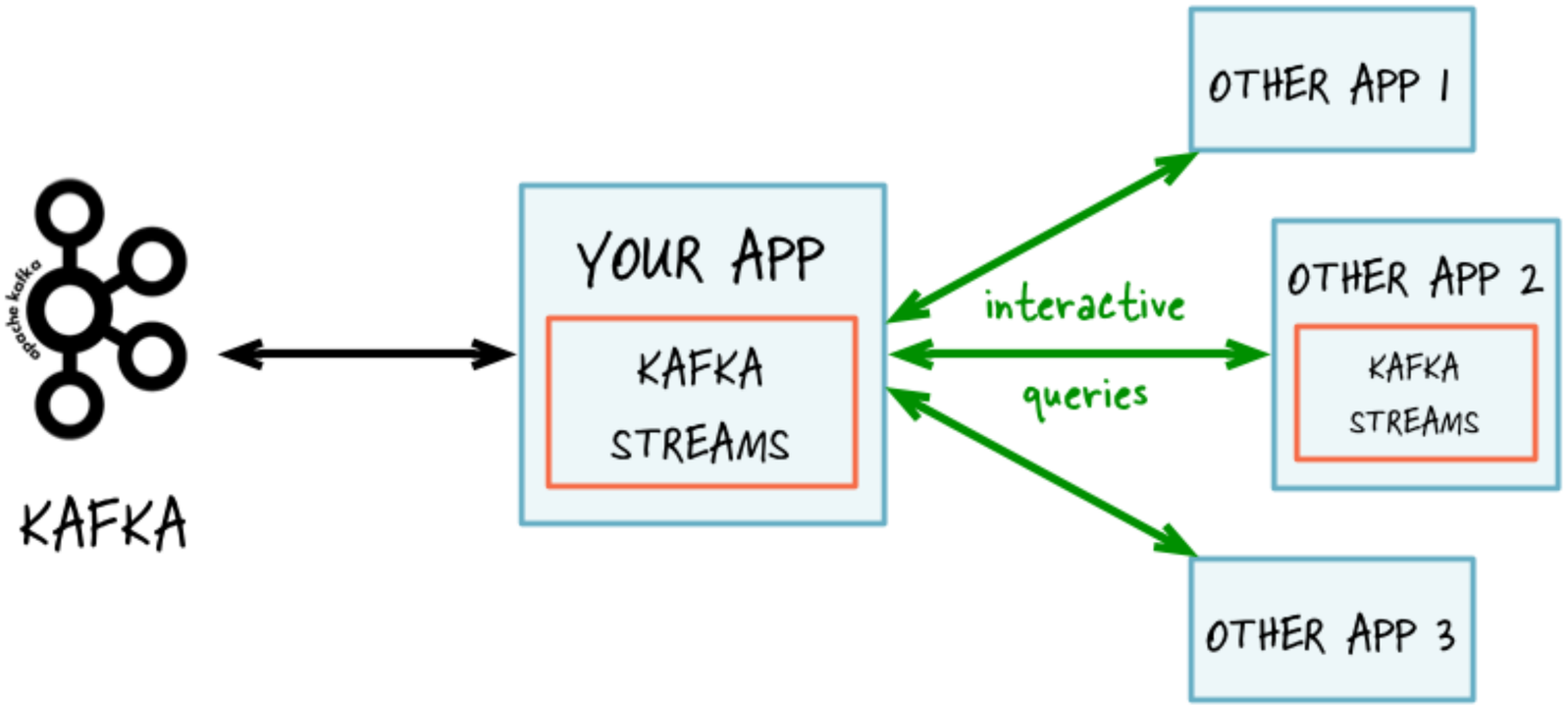
Kafka Stream
Stateful - share state (results)

Without interactive queries: increased complexity and heavier footprint of architecture

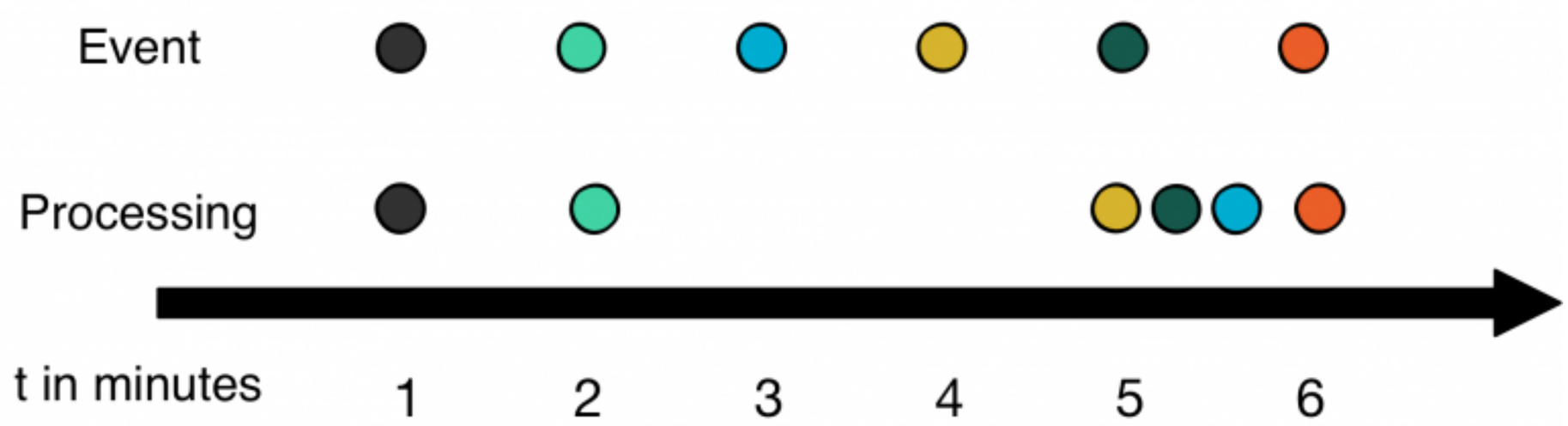


Kafka Stream
Stateful - Interactive Queries

1. Capture business events in Kafka
2. Process the events with Kafka Streams
3. With interactive queries, other apps can directly query the latest results



Timing



Event time: Time at which the event actually occurred

Ingestion time: Time at which the event was observed in the system

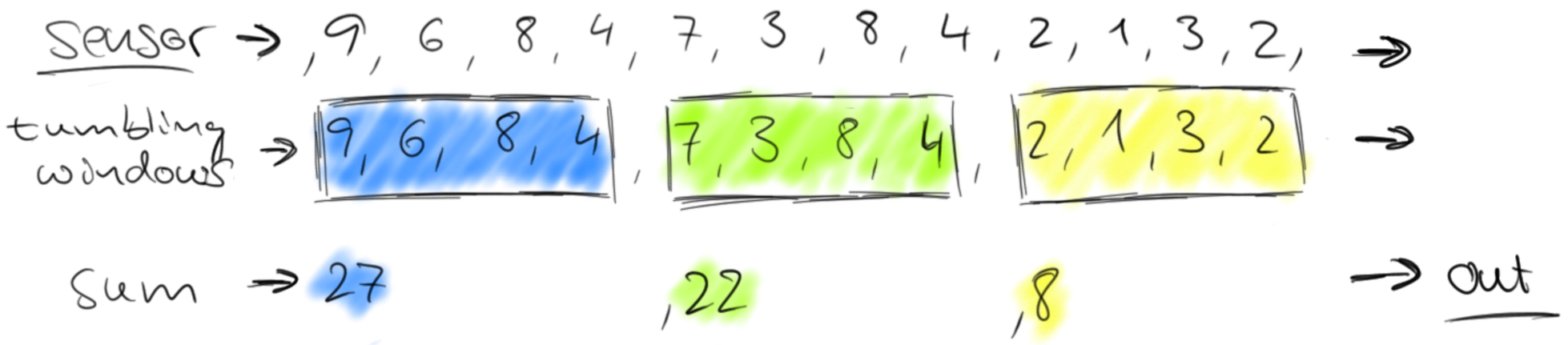
Processing time: Time at which the event was processed by the system

Timing - Processing

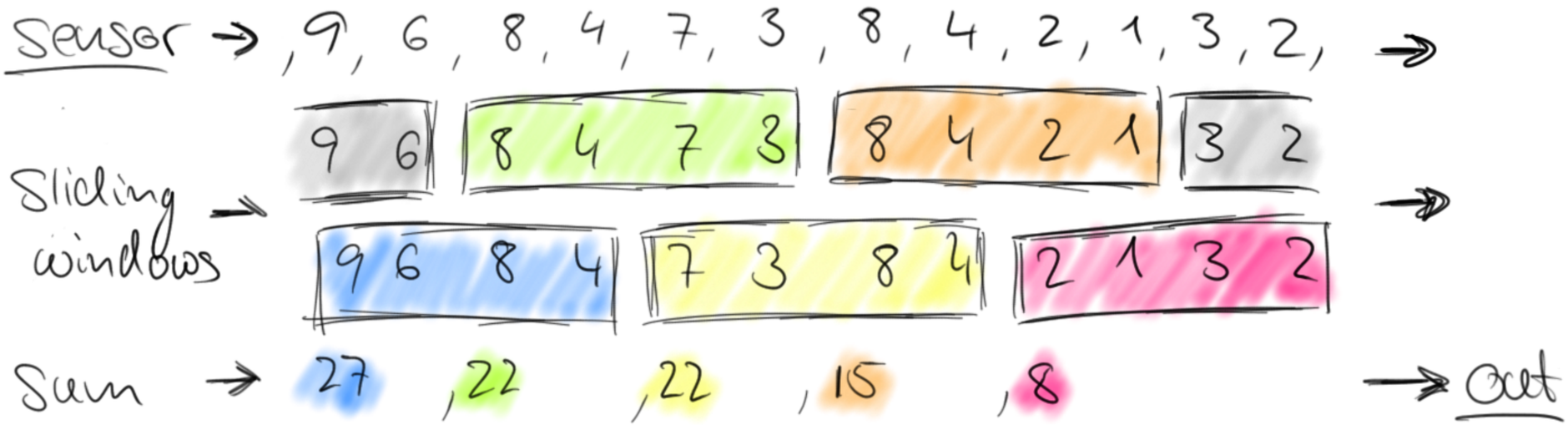
Event Time - vs - Ingestion Time

Kafka configuration at broker level or per topic

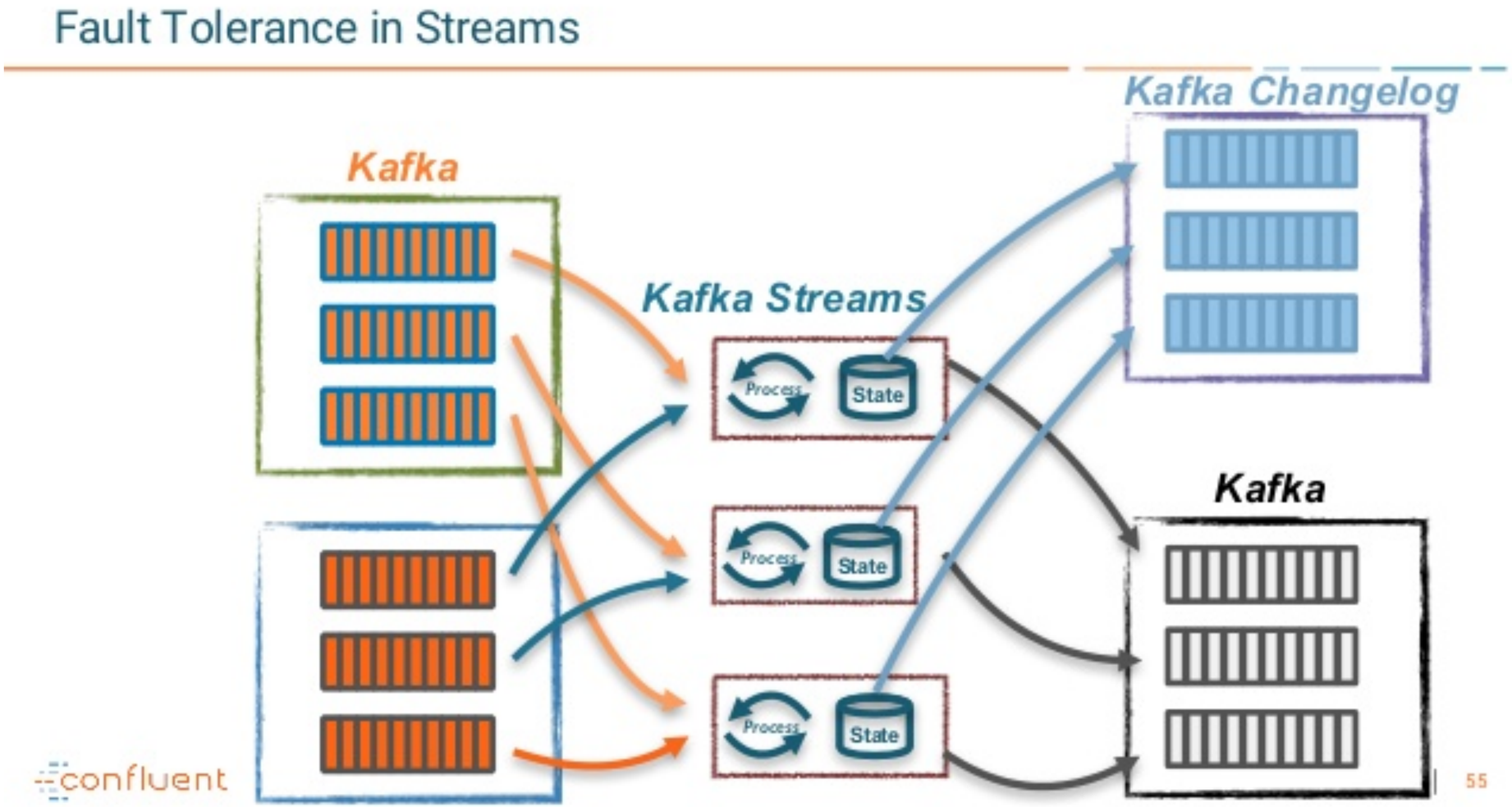
Timing - Tumbling Window



Timing - Sliding Window



Fault Tolerance



Processing Guarantees

At-least-once semantics

Records are never lost but may be redelivered. If your stream processing application fails, no data records are lost and fail to be processed, but some data records may be re-read and therefore re-processed. At-least-once semantics is enabled by default (processing.guarantee="at_least_once") in your Streams configuration.

Exactly-once semantics

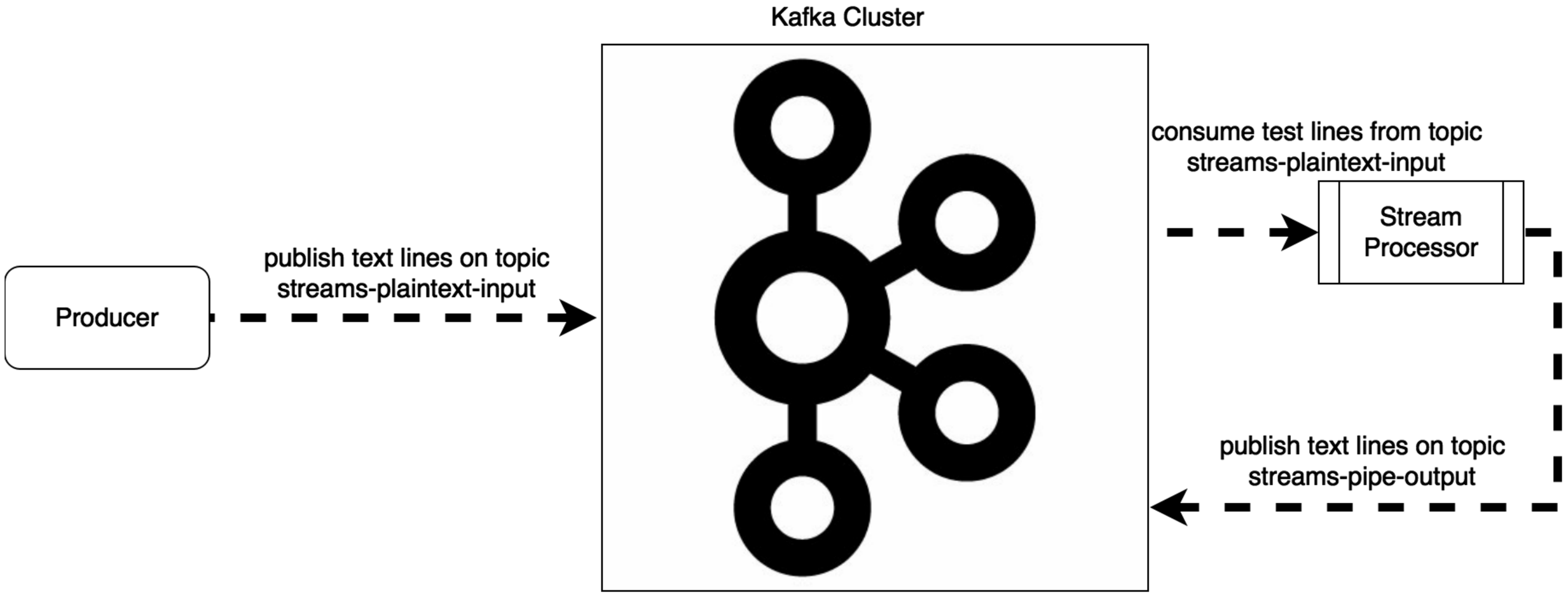
Records are processed once. Even if a producer sends a duplicate record, it is written to the broker exactly once. Exactly-once stream processing is the ability to execute a read-process-write operation exactly one time. All of the processing happens exactly once, including the processing and the materialized state created by the processing job that is written back to Kafka. To enable exactly-once semantics, set processing.guarantee="exactly_once" in your Streams configuration.

Reprocessing

Reprocessing capabilities so you can recalculate output when your code changes

Code examples

https://github.com/smarcu/datastreaming-presentation



Thank You