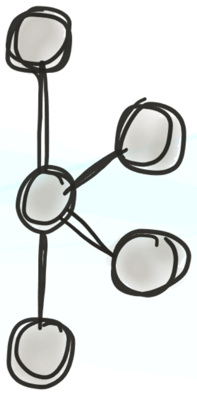
Demystifying Stream Processing with Apache Kafka



Neha Narkhede, Confluent

confluent

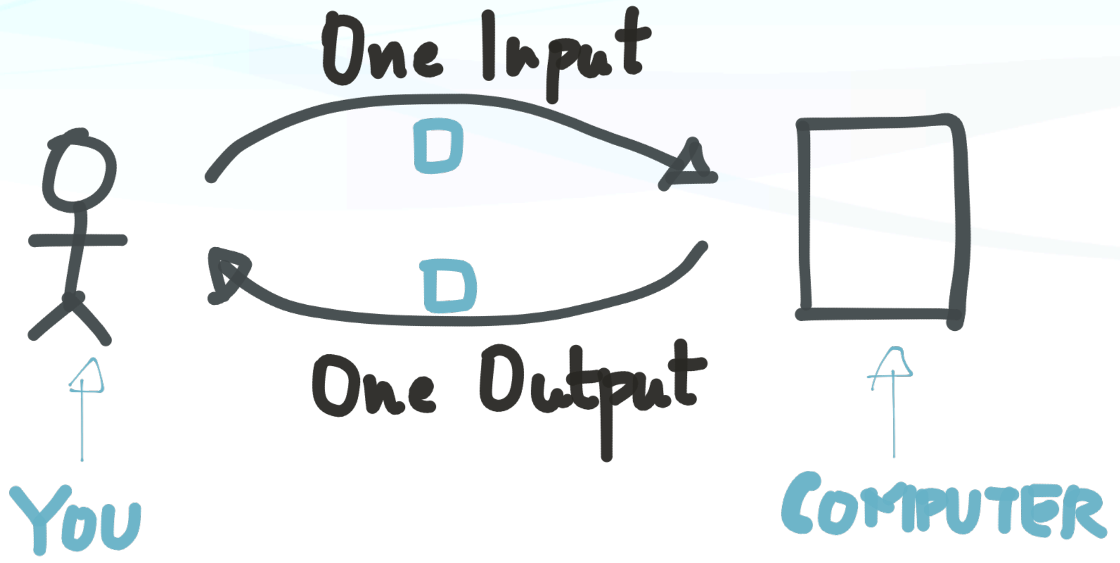
What is STREAM PROCESSING

confluent

3 Paradigms for Programming

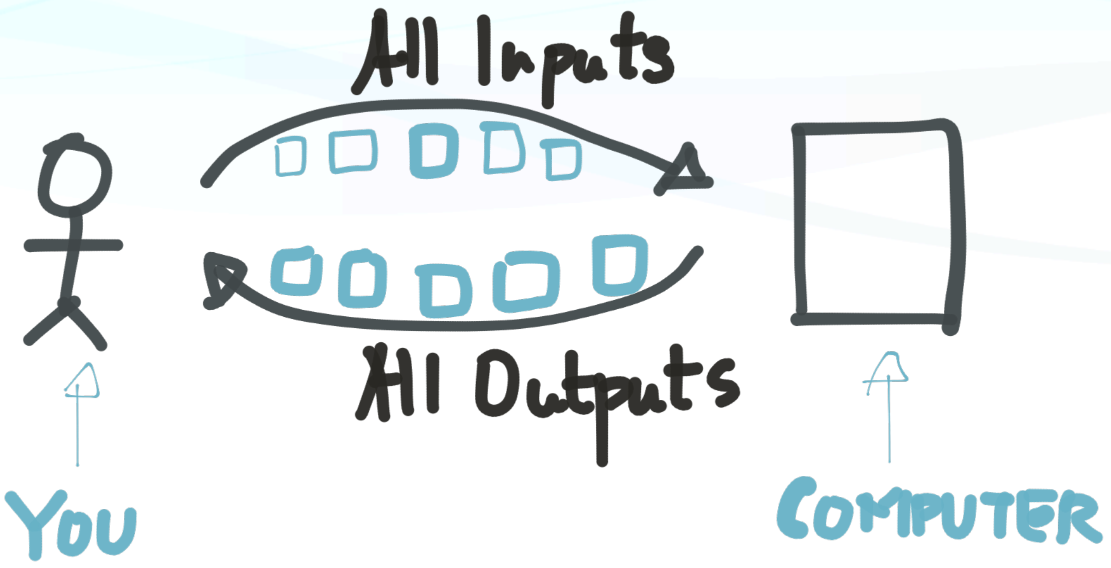
1. request/response
2. batch
3. stream processing

Request/Response



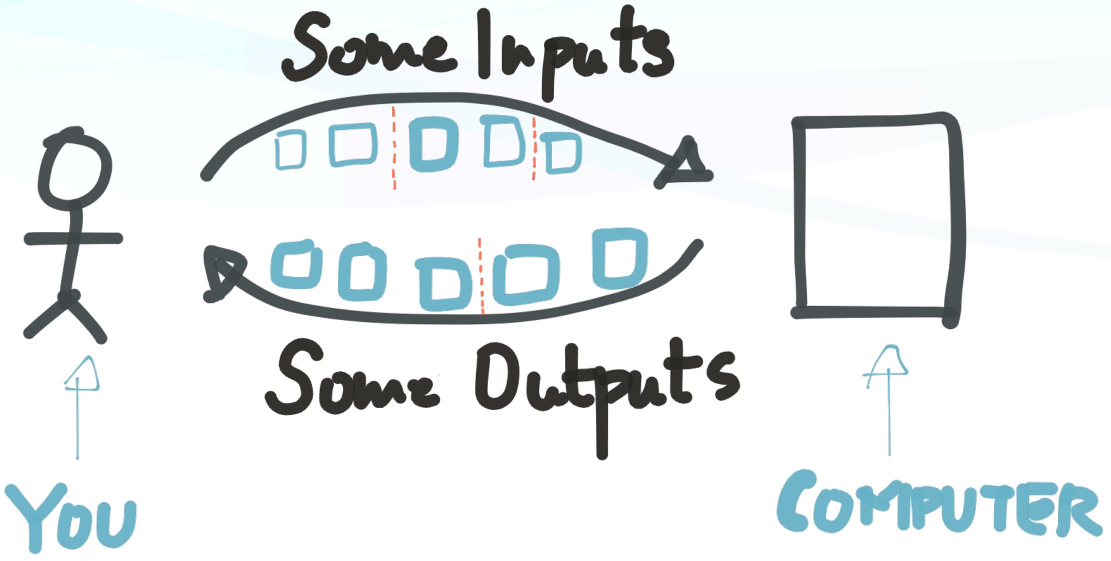
confluent

BATCH



confluent

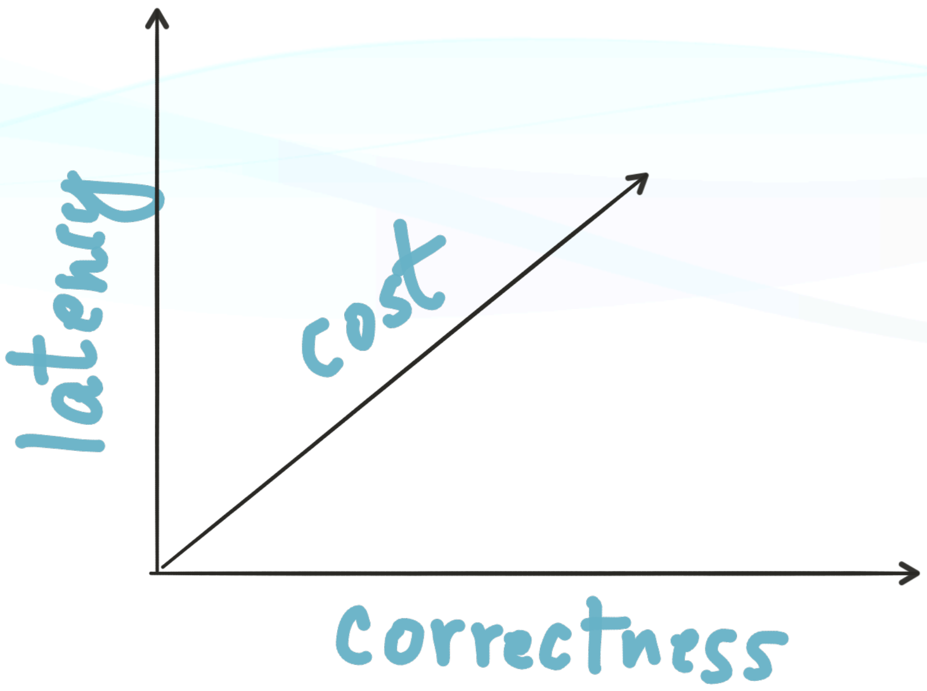
STREAM PROCESSING



STREAM PROCESSING isn't (NECESSARILY)

- Transient
- Approximate
- Lossy

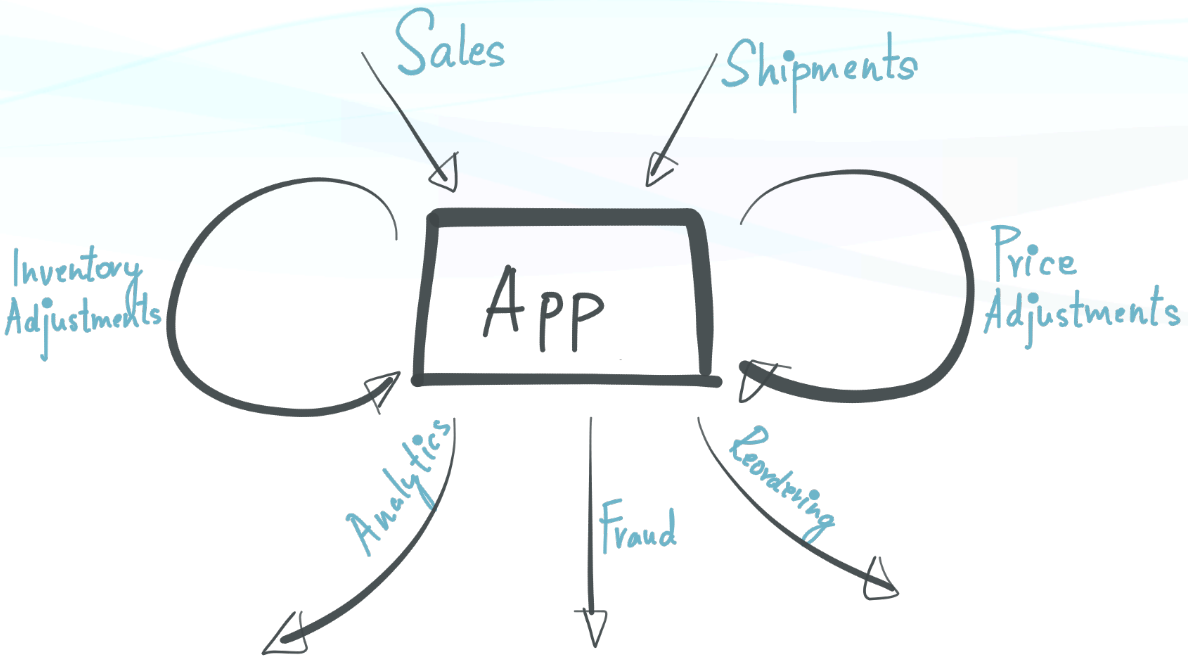
TRADE OFFS



confluent

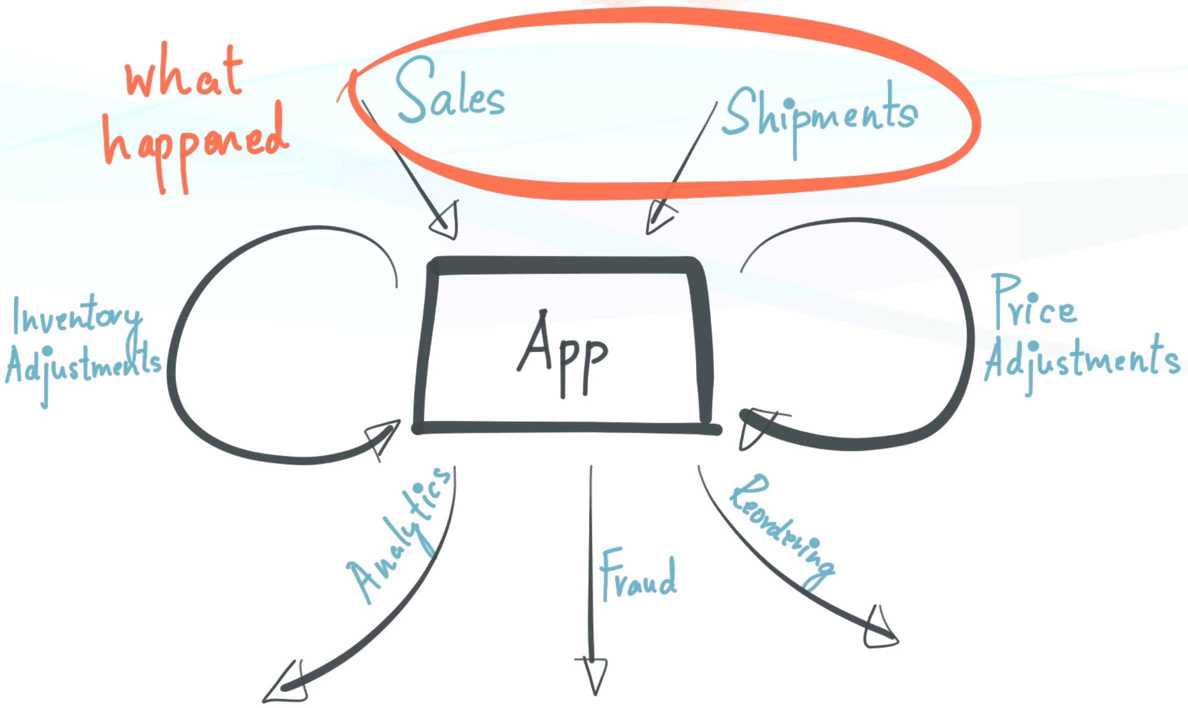
Stream Processing in the Large

Retail



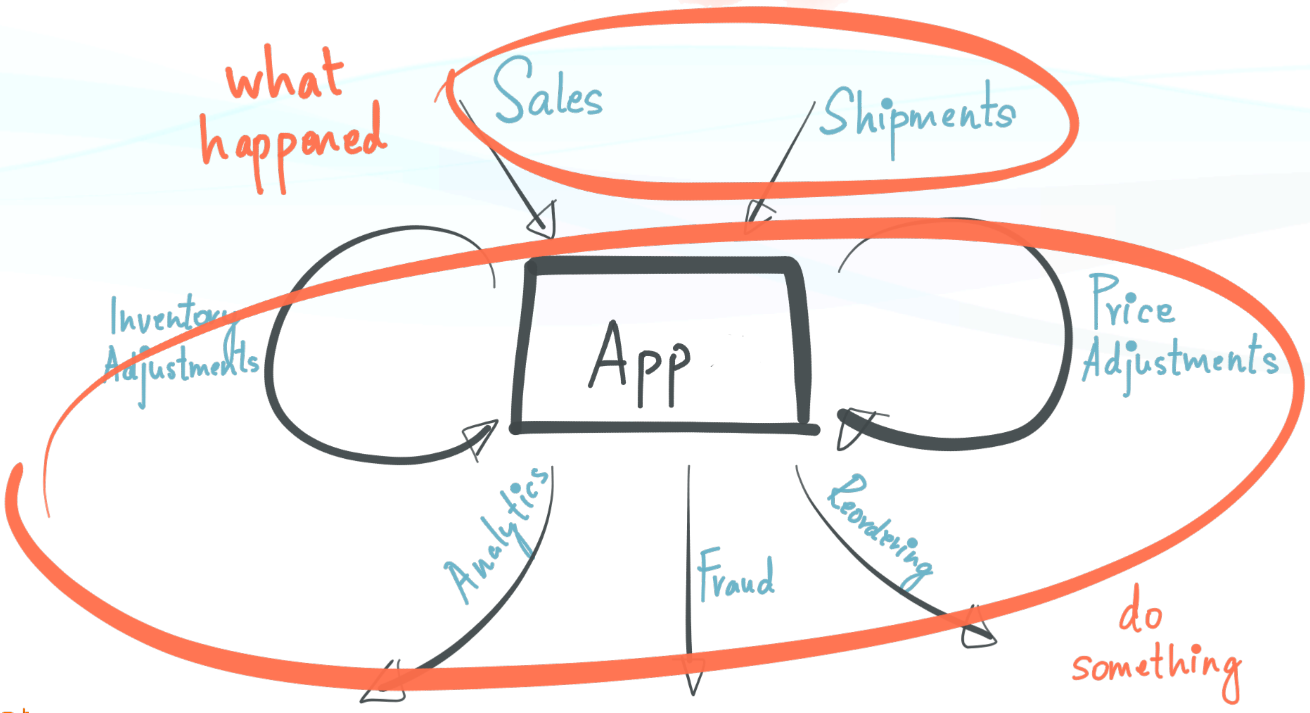
confluent

RETAIL

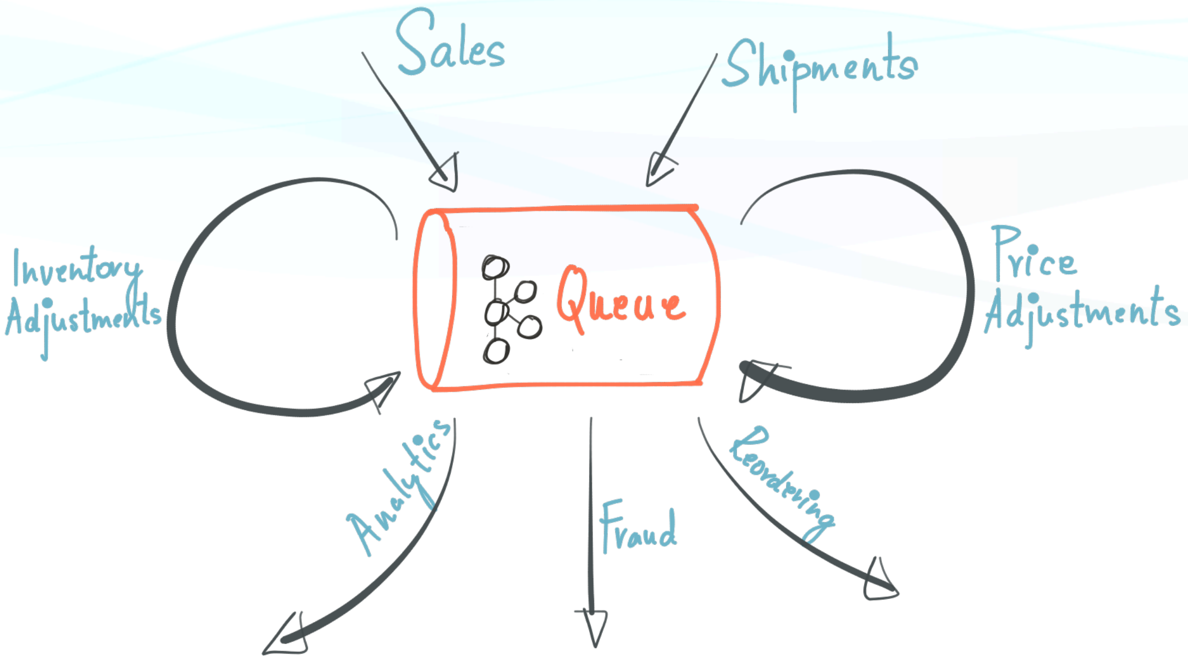


confluent

RETAIL


what happened
Sales
Shipments
Inventory Adjustments
App
Price Adjustments
Analytics
Fraud
Reordering
do something

RETAIL



confluent

stream processing

$$= f(\text{“what happened”})$$

↑

events

confluent

stream processing

=

f ("what happened")

↑

events

confluent

Stream Processing with Kafka

2 approaches

confluent

Approach #1

DIY!

stream processing

PROBLEM: STREAM PROCESSING HAS SOME hard parts

- partitioning & scalability
- semantics & fault tolerance
- state
- windowing & time
- reprocessing

confluent

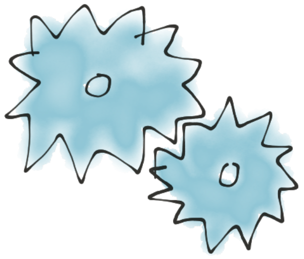
APPROACH #2

Use a stream processing framework

- SPARK
- STORM
- SAMZA
- FLINK

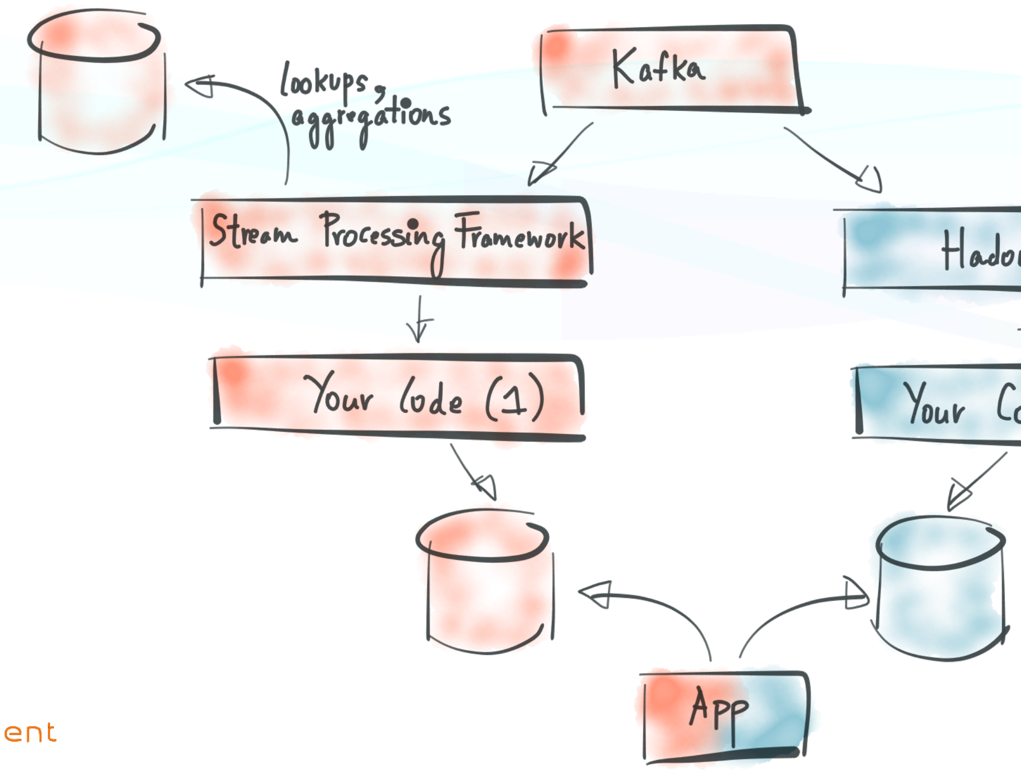
ET AL

PROBLEM : LOTS OF moving parts!



confluent

Example Architecture

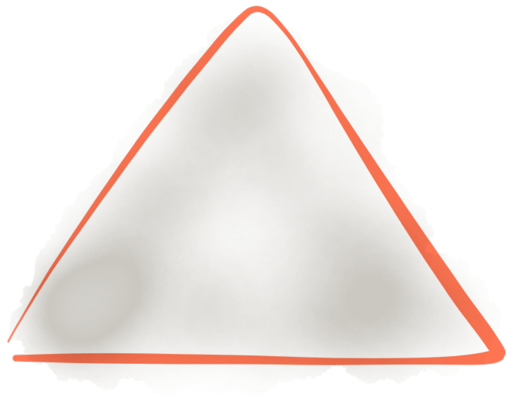


The Hard Part Of Distributed Systems

Coding < Debugging < Operations

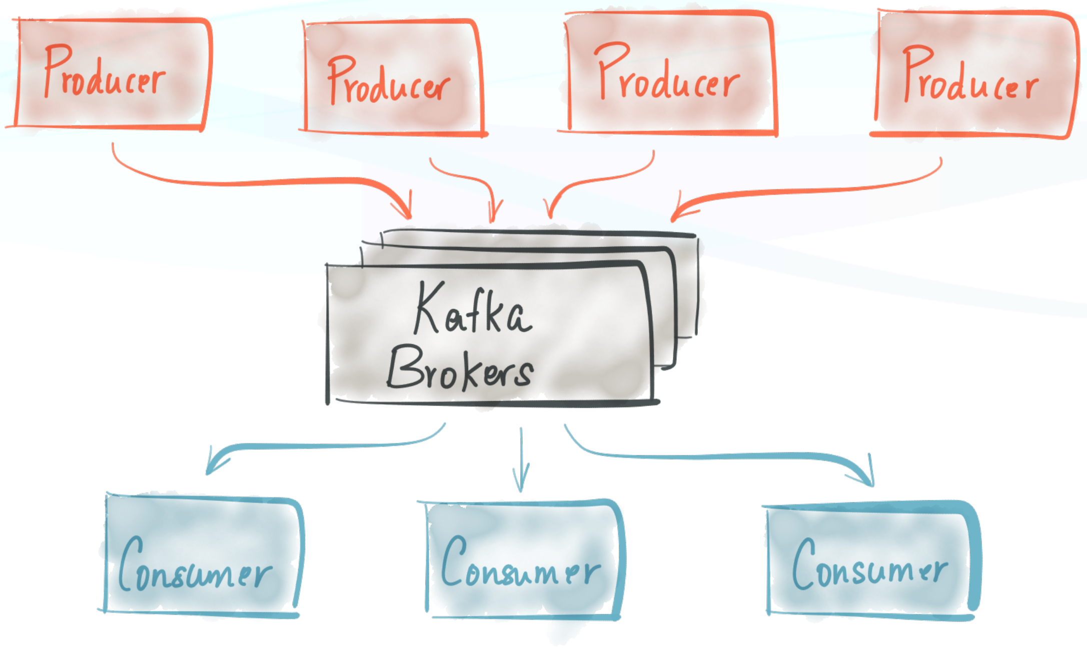
Is There A better WAY?

Kafka Provides primitives



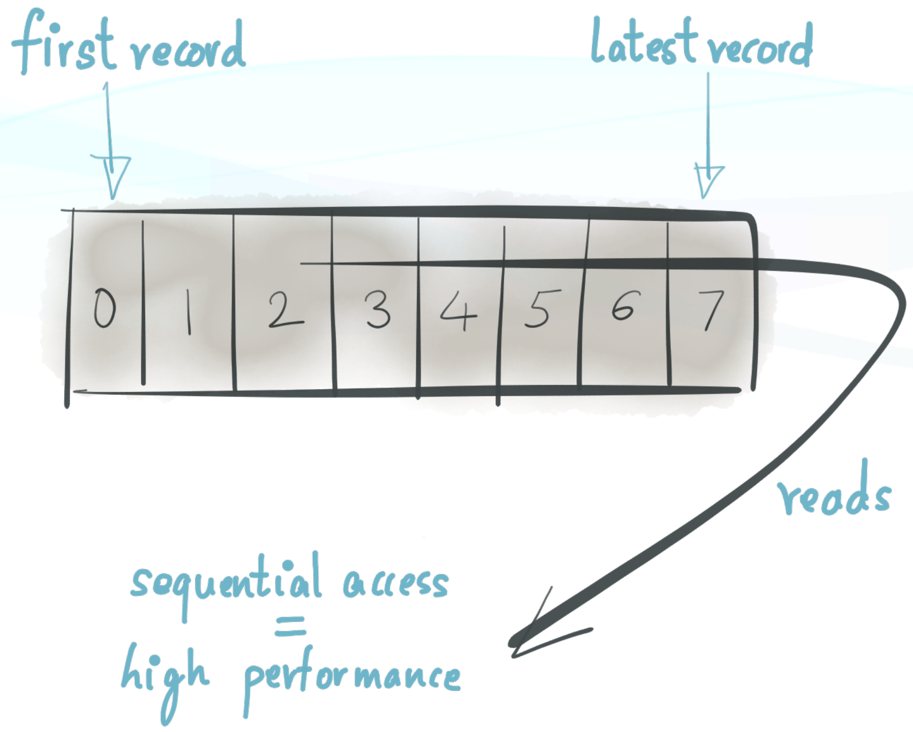
confluent

Apache Kafka : 10,000 FT View



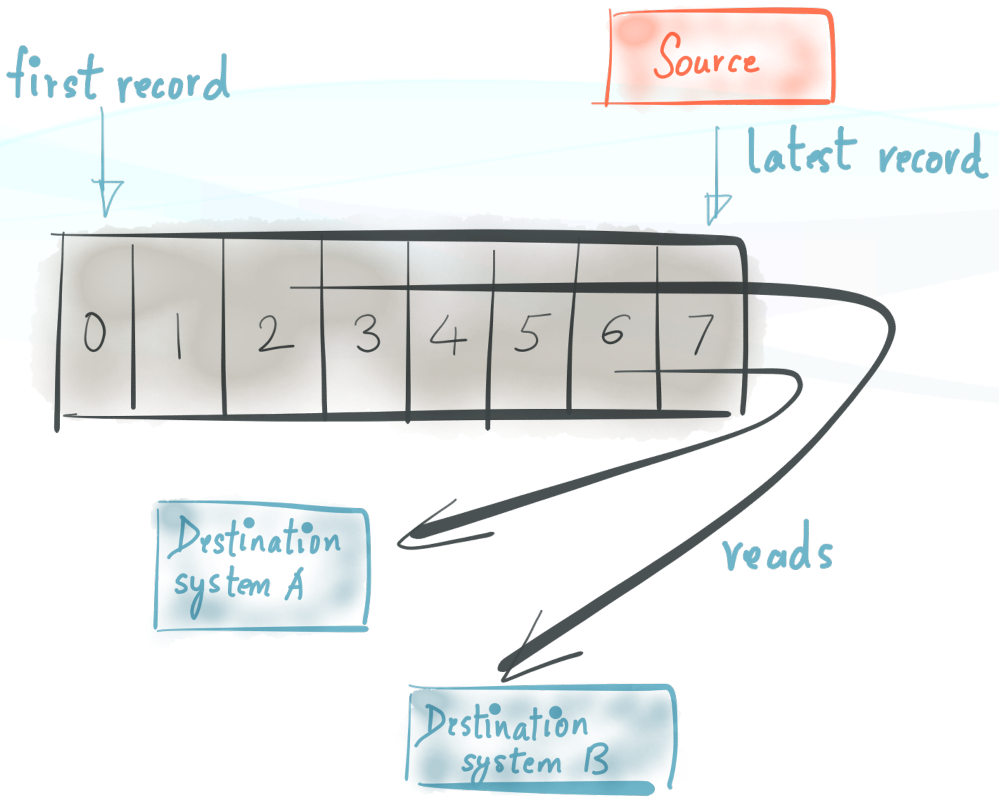
confluent

Kafka Key Idea : Log



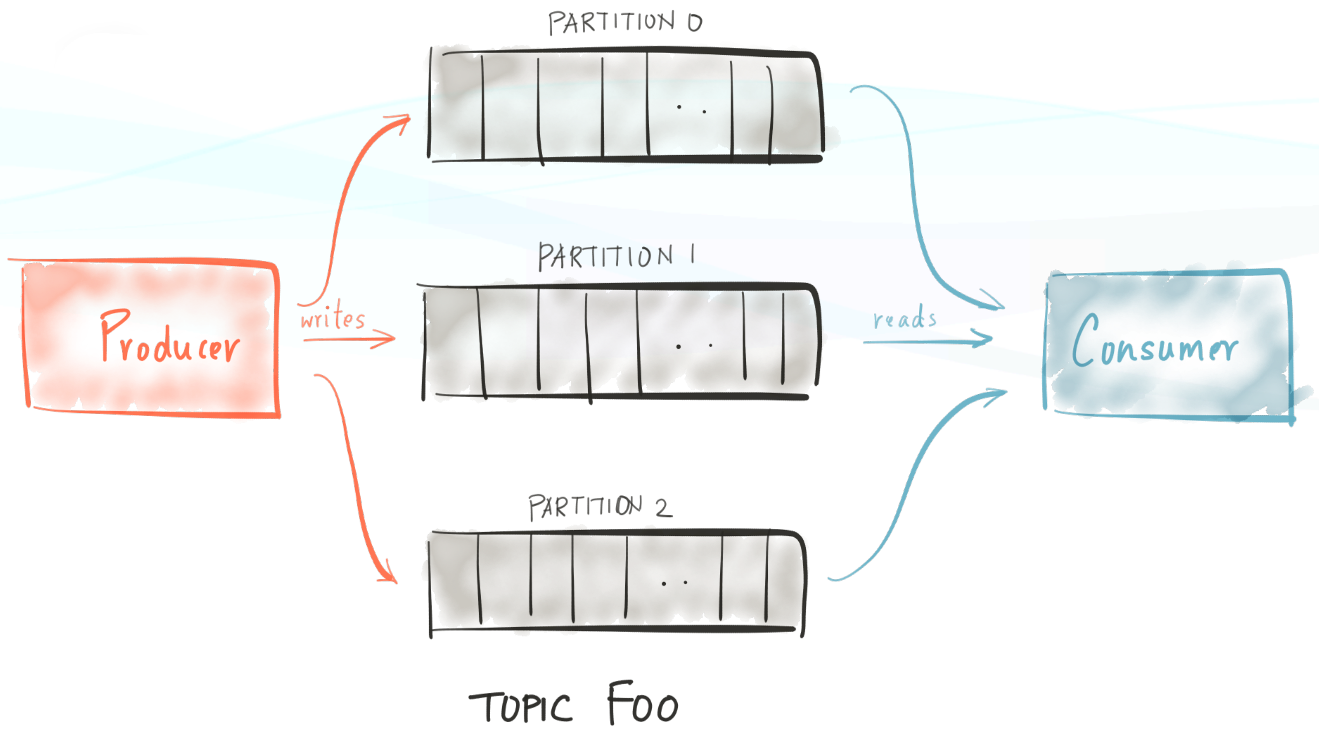
confluent

Logs & Pub-Sub



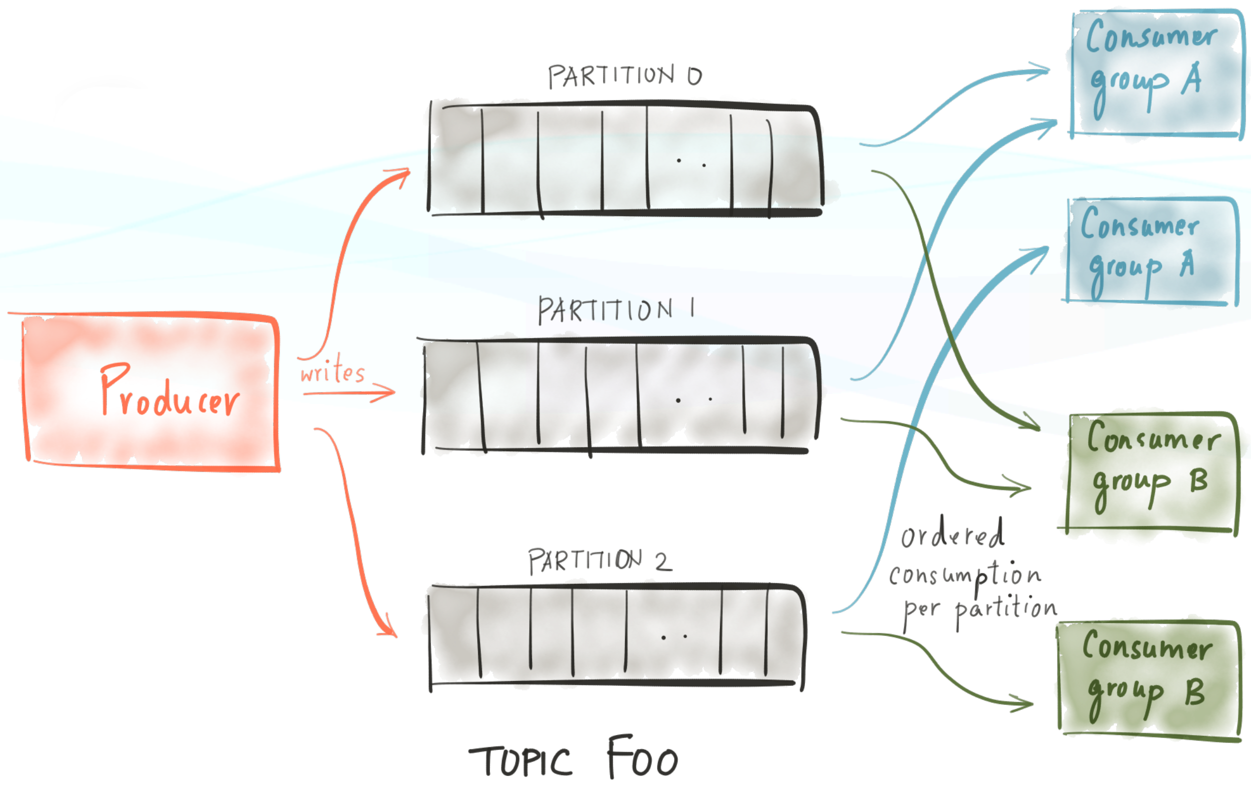
first record
Source
latest record
0
1
2
3
4
5
6
7
reads
Destination system A
Destination system B

Topic ≈ Partitioned Log



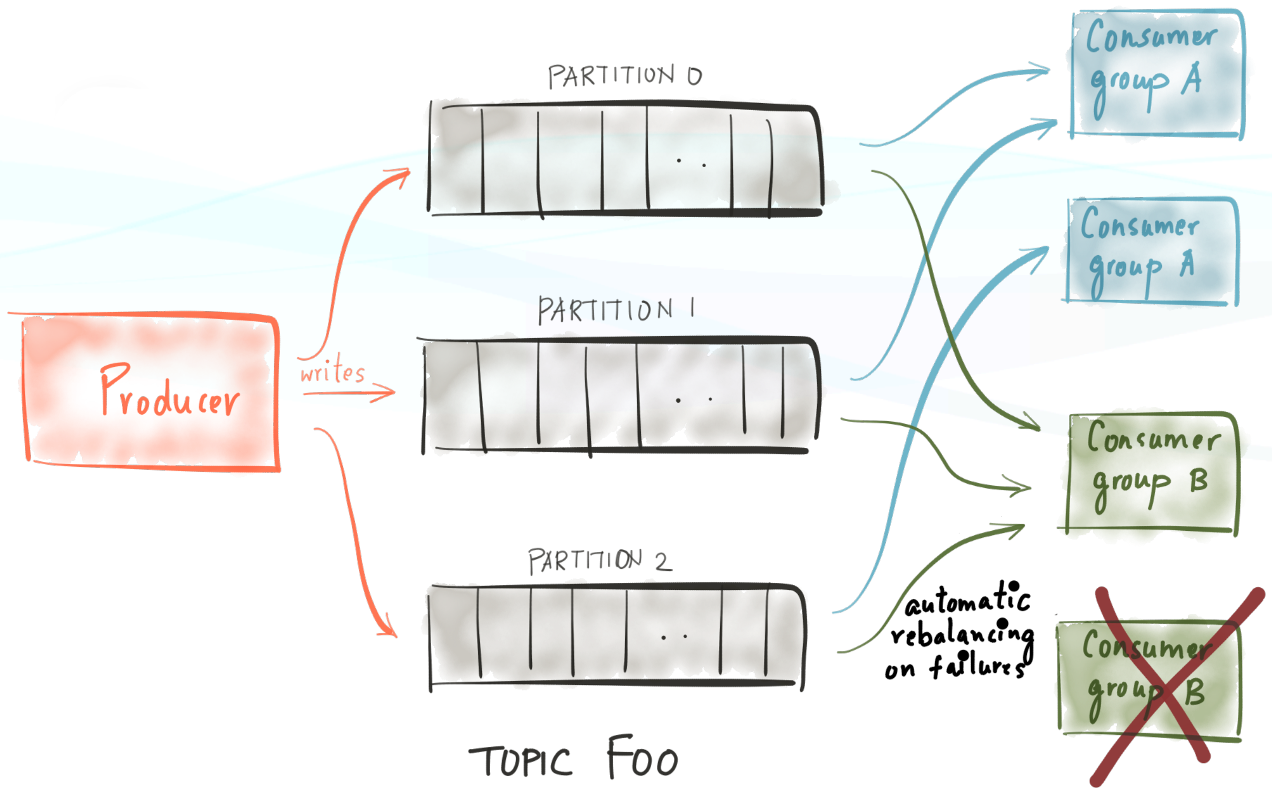
confluent

SCALABLE CONSUMPTION



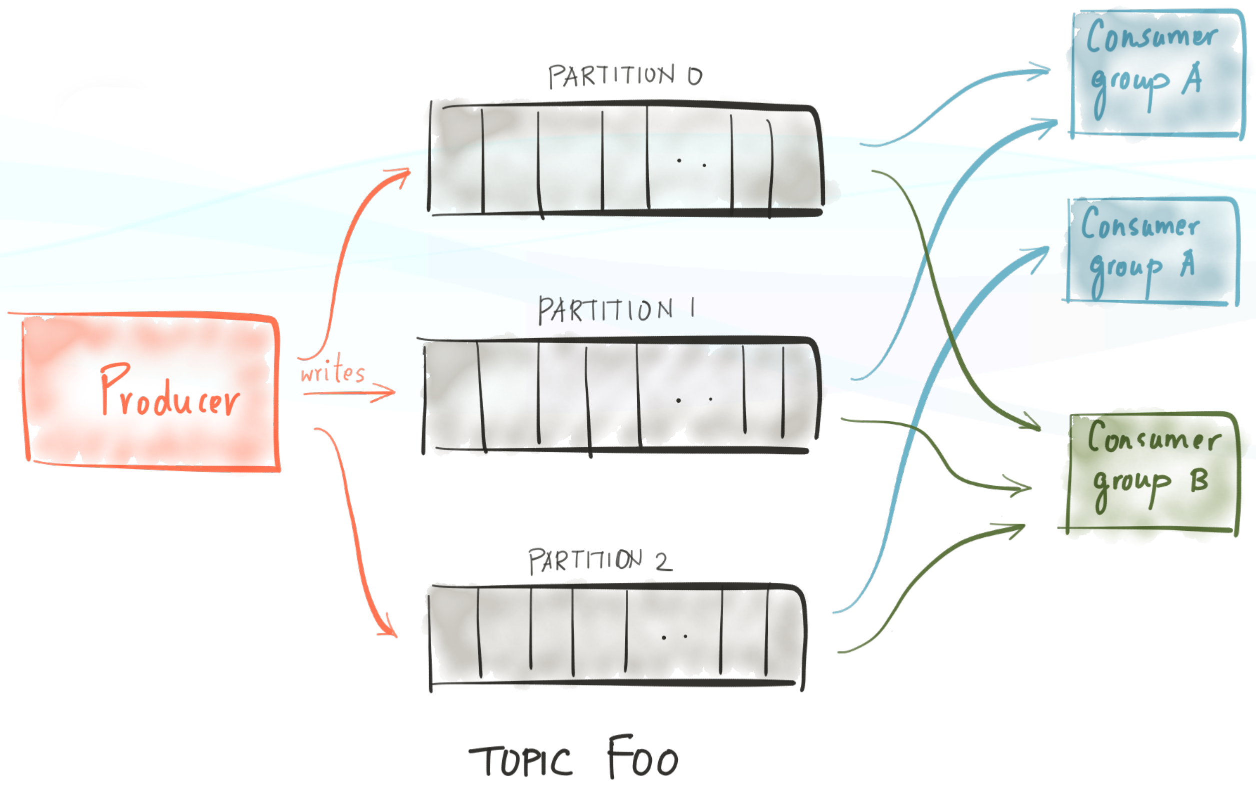
confluent

Fault Tolerant Consumption



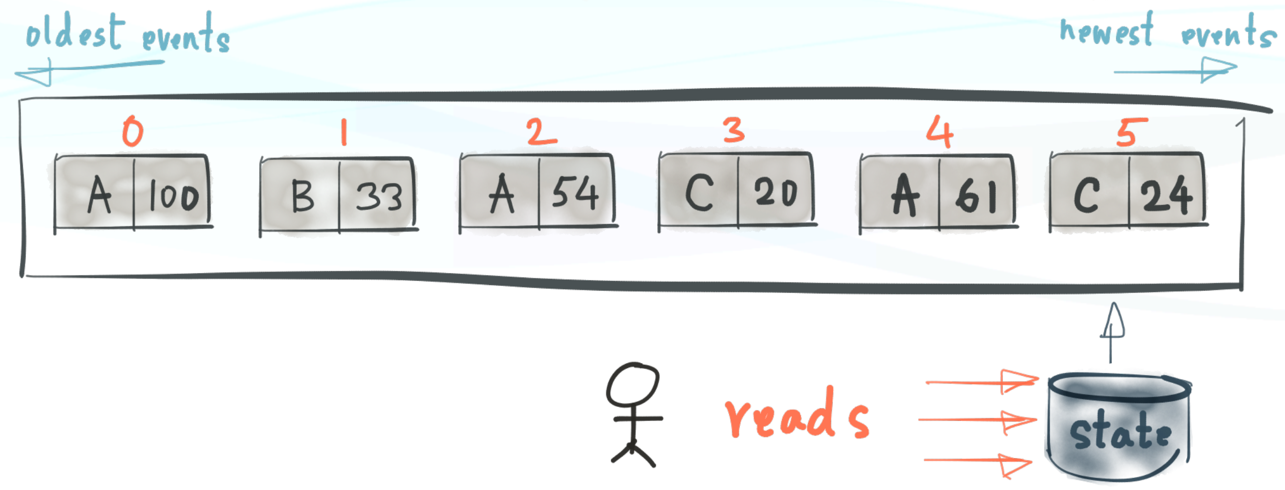
confluent

FAULT TOLERANT CONSUMPTION

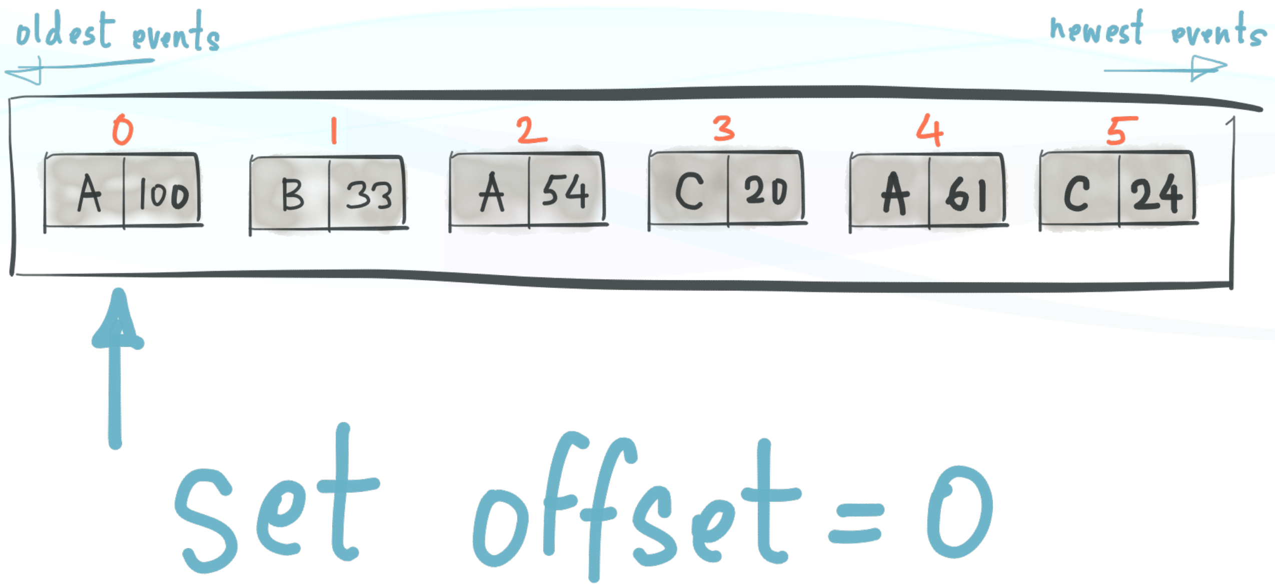


confluent

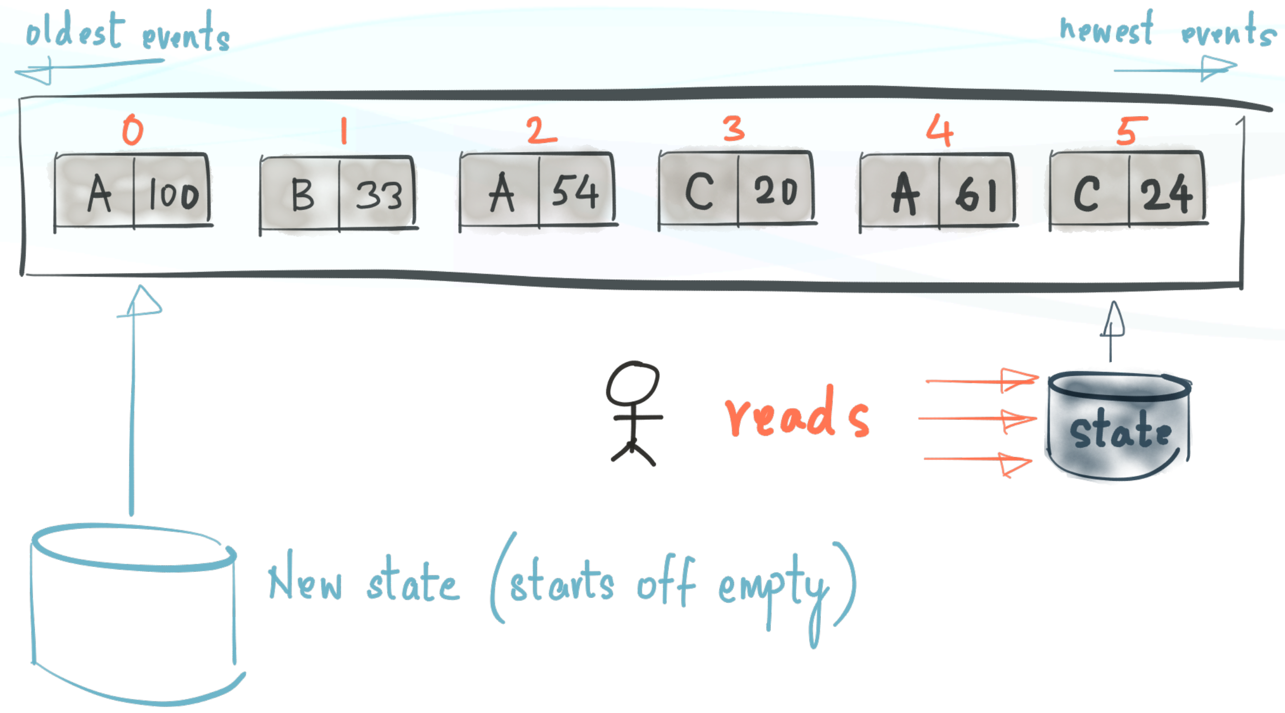
REPROCESSING



REPROCESSING

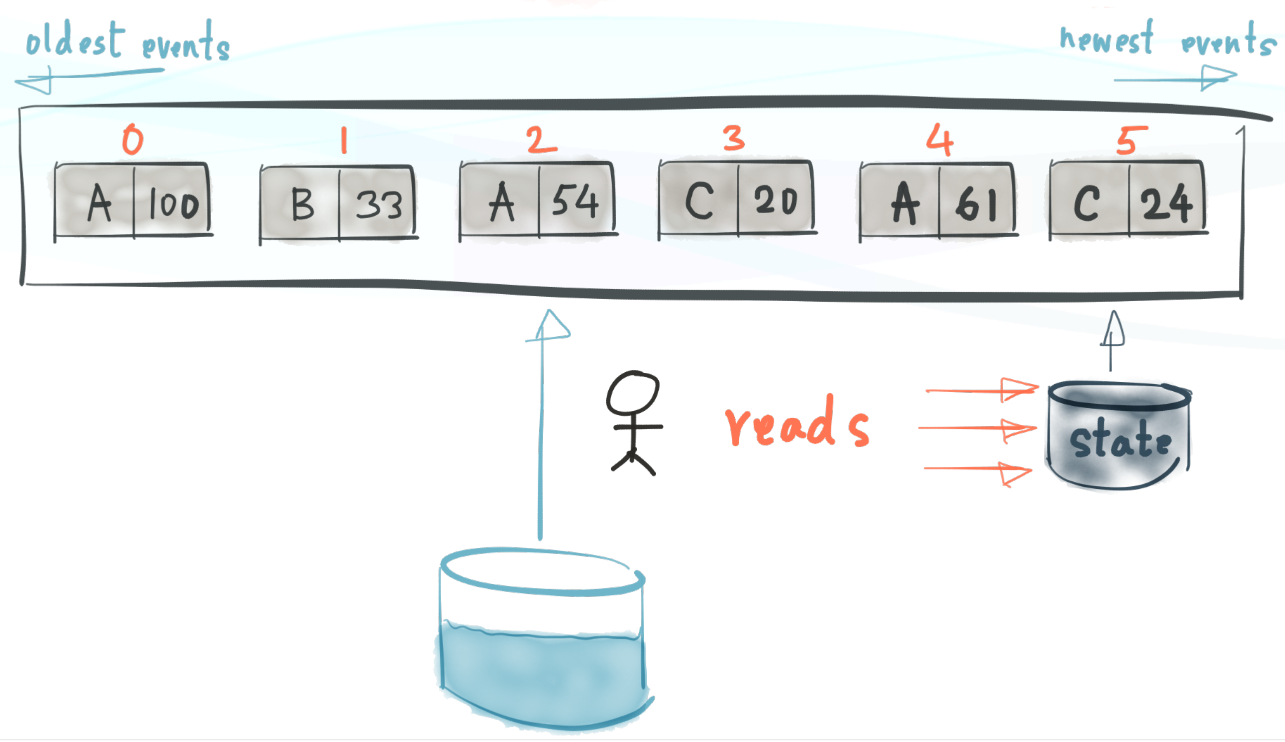


REPROCESSING



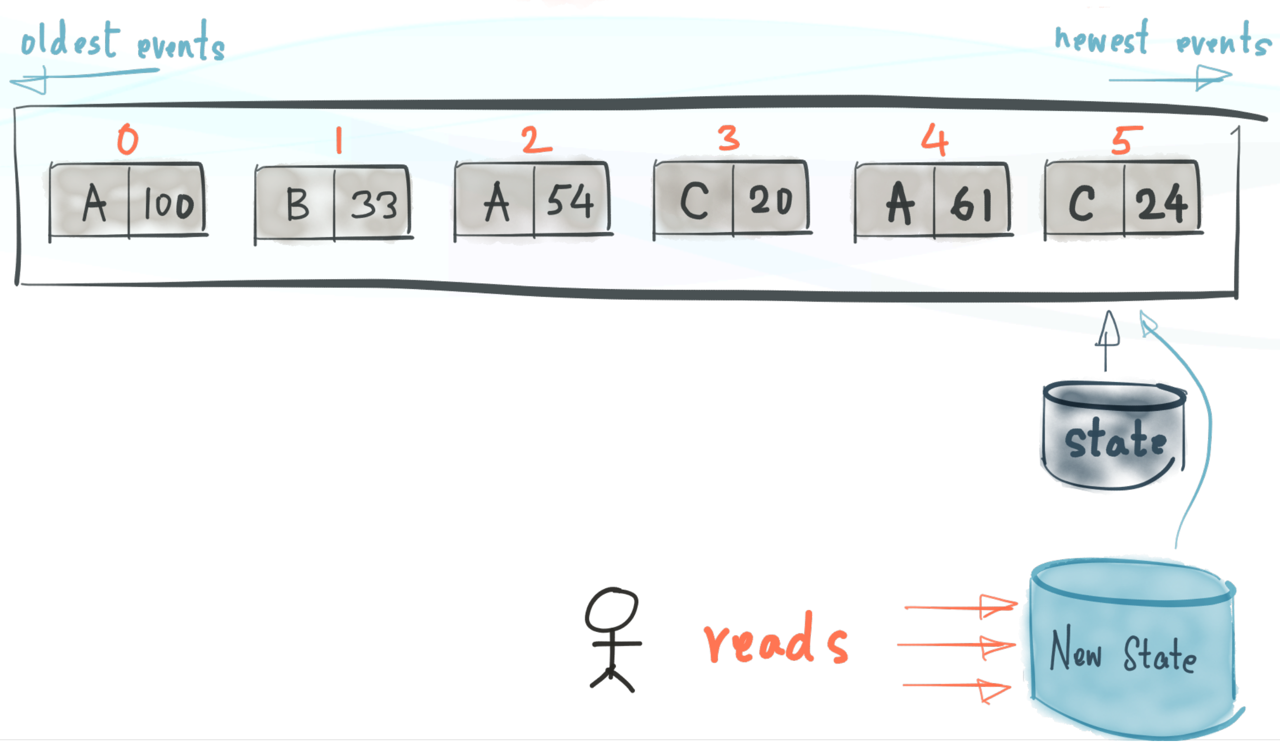
confluent

REPROCESSING



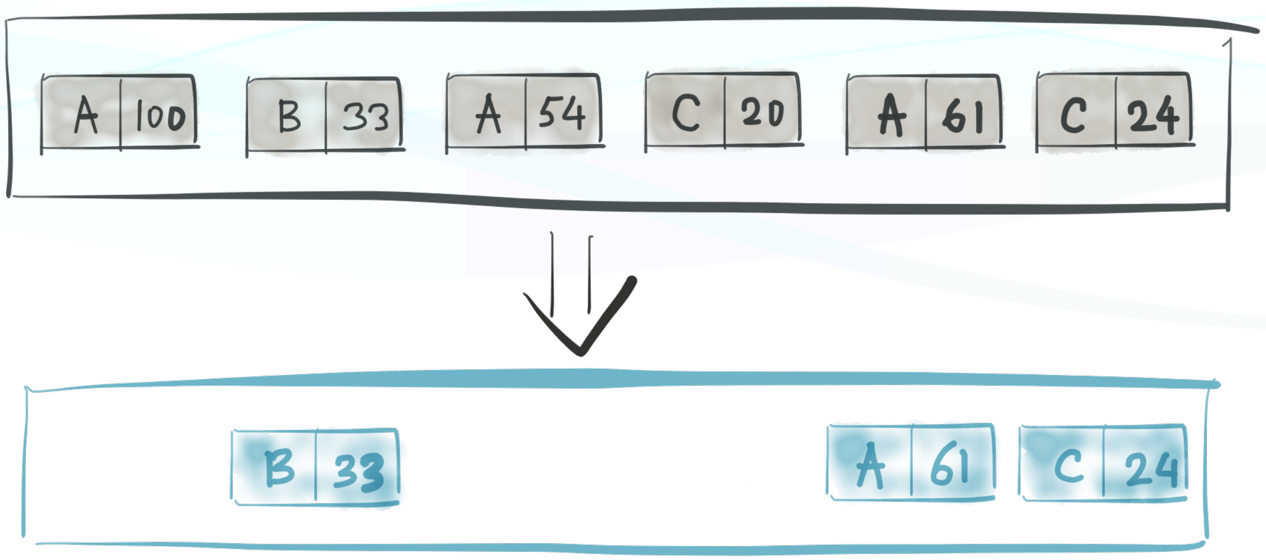
confluent

Reprocessing



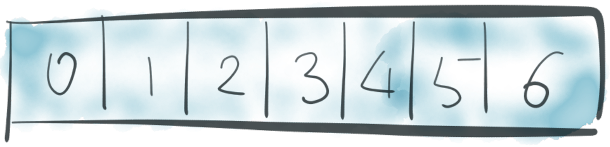
confluent

Log Compaction



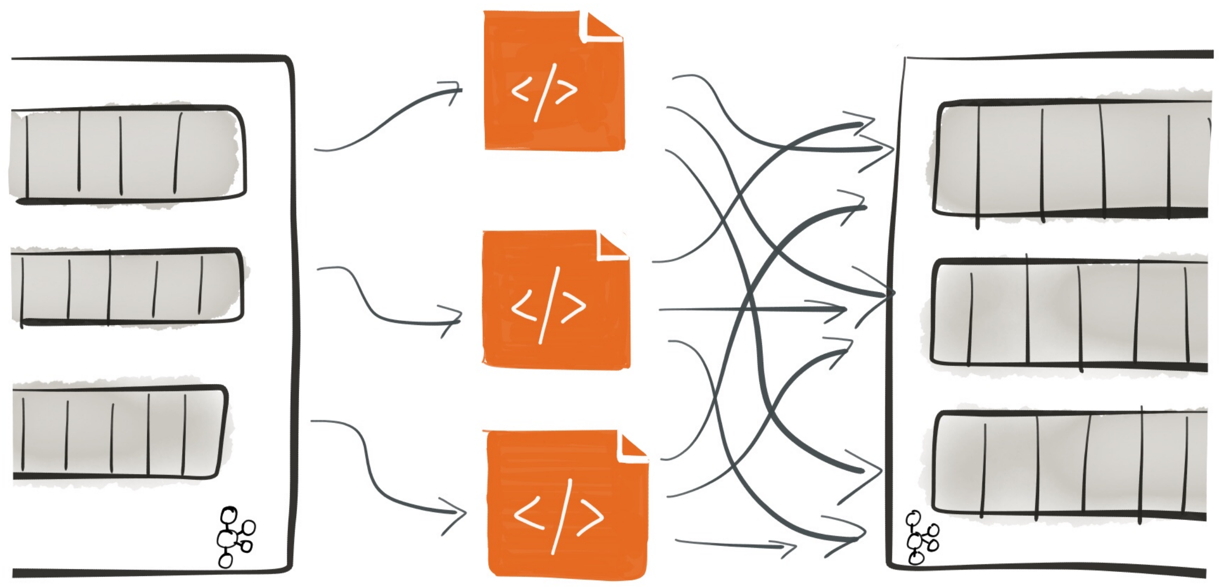
confluent

Logs unify Batch & Stream Processing



confluent

STREAM PROCESSING



CONSUMER + CUSTOM CODE + PRODUCER

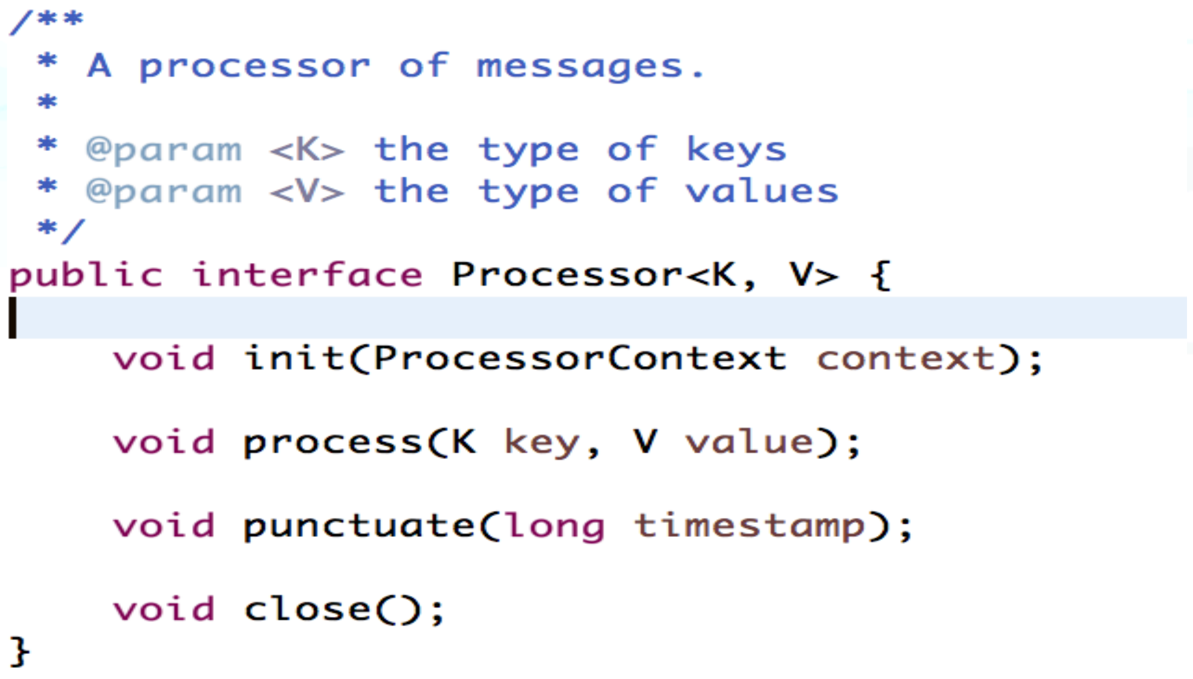
Kafka Streams

- library
- 2 interfaces
  1. processor API
  2. kstream DSL

confluent

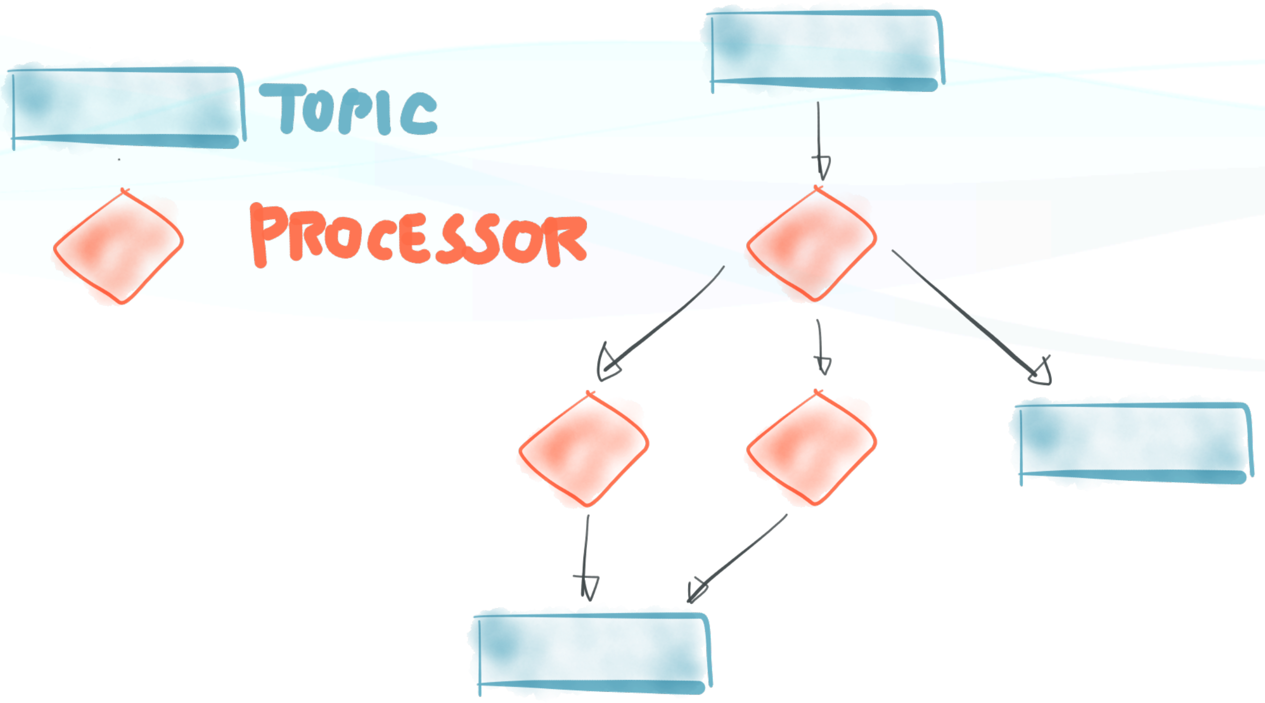
LIBRARIES VS FRAMEWORKS

Processor API



```java
/**
 * A processor of messages.
 *
 * @param <K> the type of keys
 * @param <V> the type of values
 */
public interface Processor<K, V> {

    void init(ProcessorContext context);

    void process(K key, V value);

    void punctuate(long timestamp);

    void close();
}
```

DAGS



confluent

Processor Api

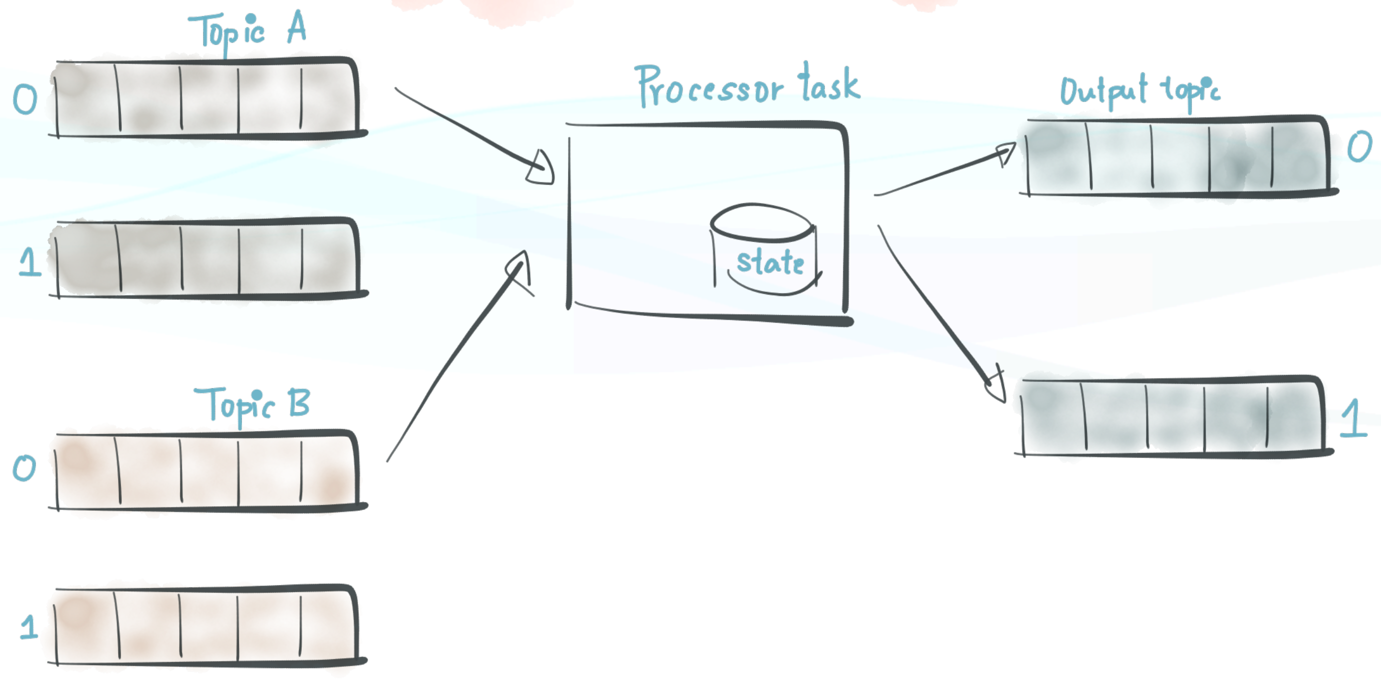
```java
public static void main(String[] args) throws Exception {
    Properties props = new Properties();
    props.put(StreamingConfig.CLIENT_ID_CONFIG, "Example-Processor-Job");
    props.put(StreamingConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamingConfig.TIMESTAMP_EXTRACTOR_CLASS_CONFIG, WallclockTimestampExtractor.class);
    StreamingConfig config = new StreamingConfig(props);

    TopologyBuilder builder = new TopologyBuilder();
    builder.addSource("SOURCE", new StringDeserializer(), new StringDeserializer(), "topic-source");
    builder.addProcessor("PROCESS", new MyProcessorDef(), "SOURCE");
    builder.addSink("SINK", "topic-sink", new StringSerializer(), new IntegerSerializer(), "PROCESS");
    KafkaStreaming streaming = new KafkaStreaming(builder, config);
    streaming.start();
}
```

KSTREAMS

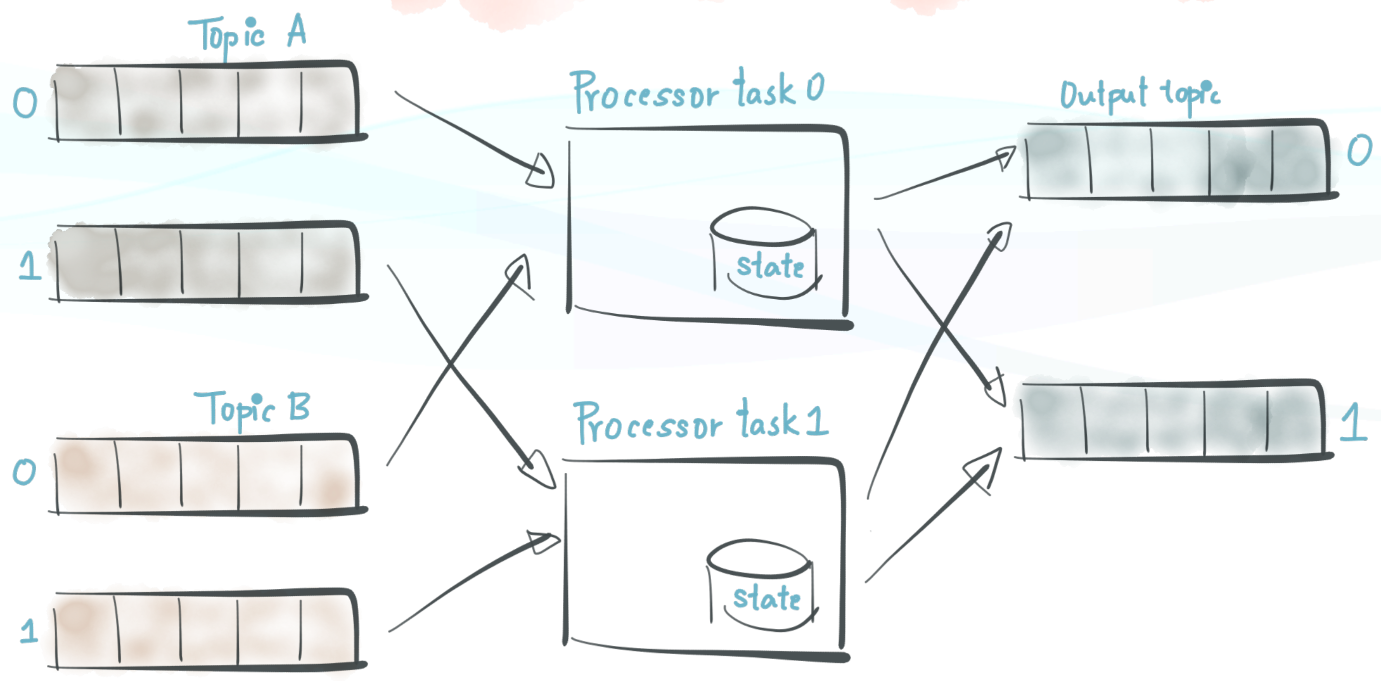
```
StreamingConfig config = new StreamingConfig(props);

// build the topology
KStreamBuilder builder = new KStreamBuilder();
KStream<String, String> stream1 = builder.from("topic1");

KStream<String, Integer> stream2 =
    stream1.map((key, value) -> new KeyValue<>(key, value.cost()))
           .aggregate(0, (oldval, newval) -> oldval + newval);

KStream<String, Integer>[] streams = stream2
    .filter((key, value) -> value > 10);

streams[0].sendTo("topic2");
streams[1].sendTo("topic3");

// start the process
KafkaStreaming kstream = new KafkaStreaming(builder, config);
kstream.start();
```

confluent

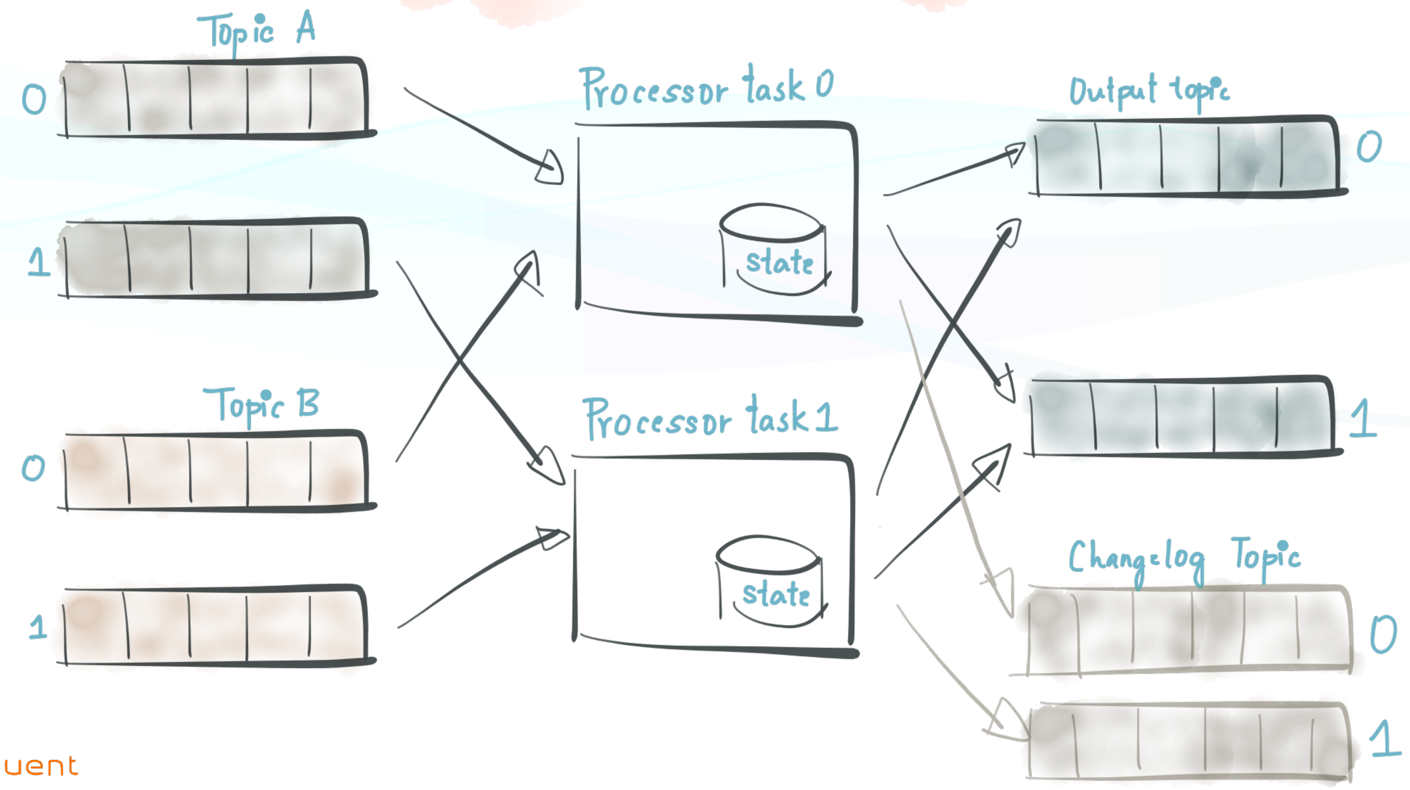
Execution Model



Execution Model

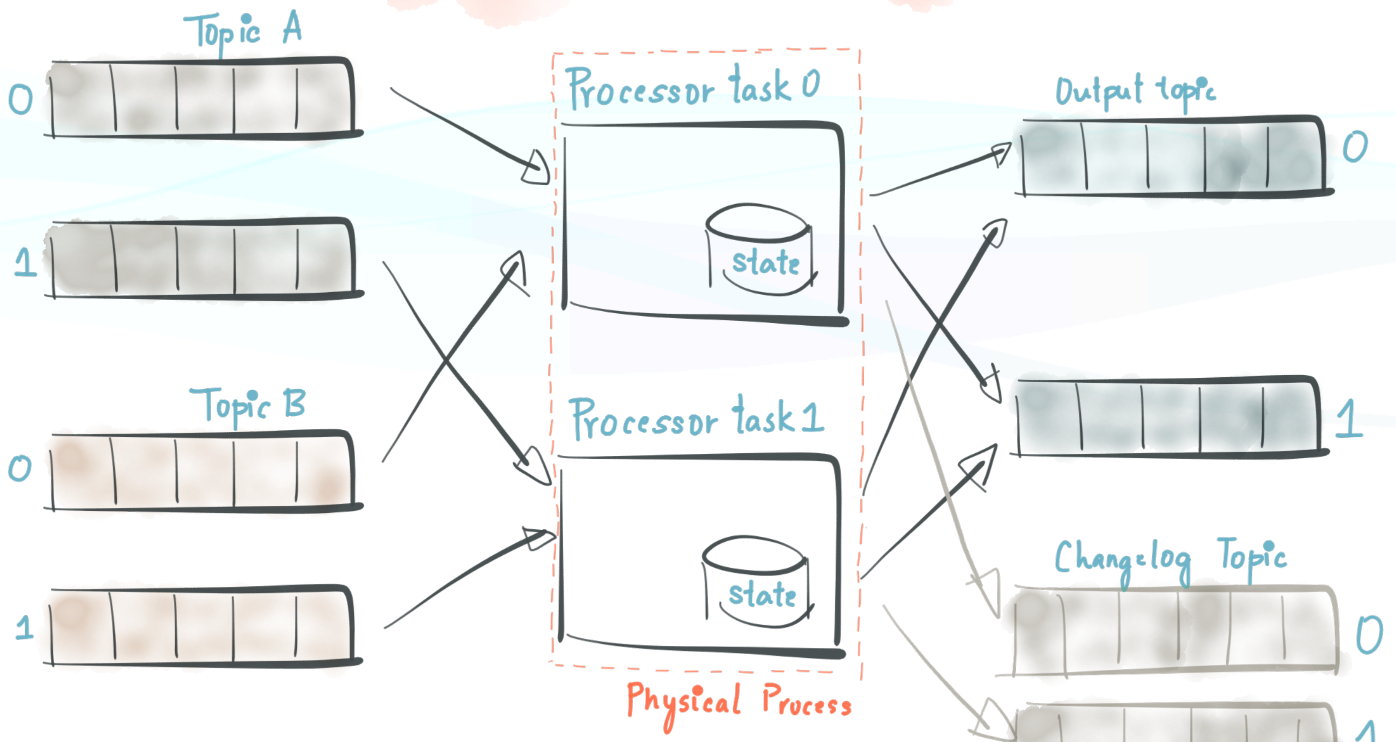


Execution Model



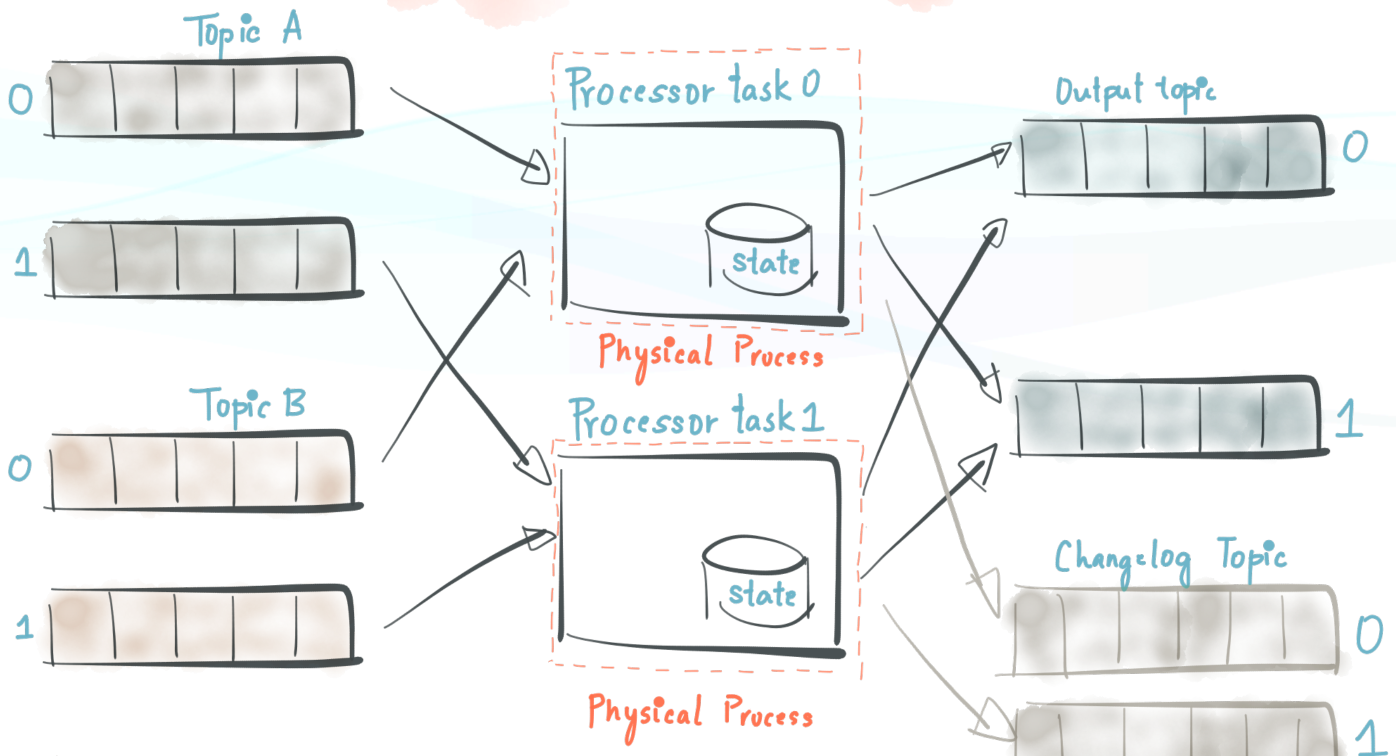
confluent

Execution Model



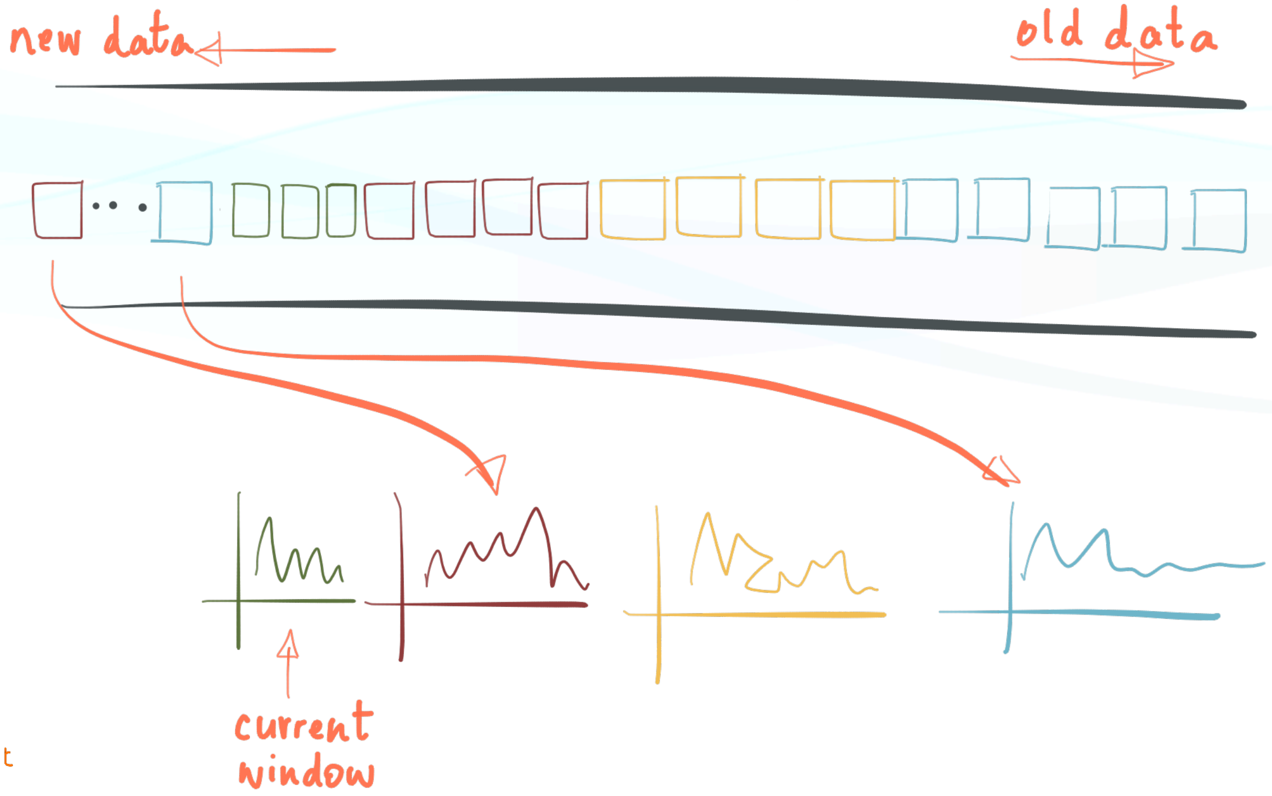
confluent

Execution Model



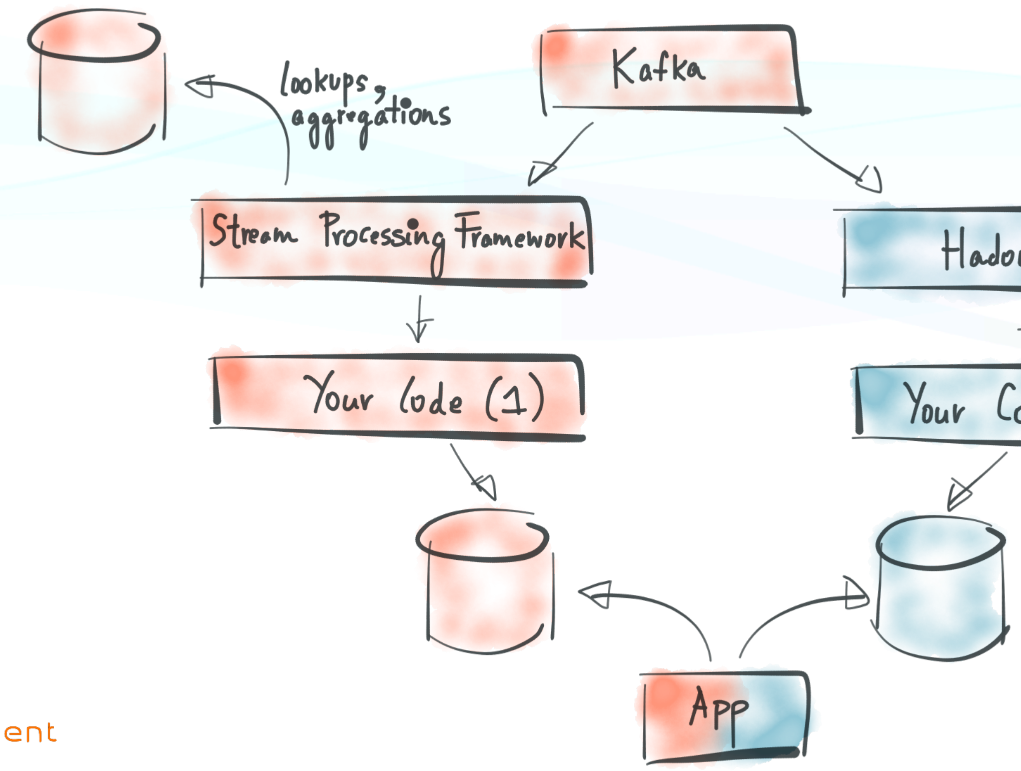
TIME

WINDOWING



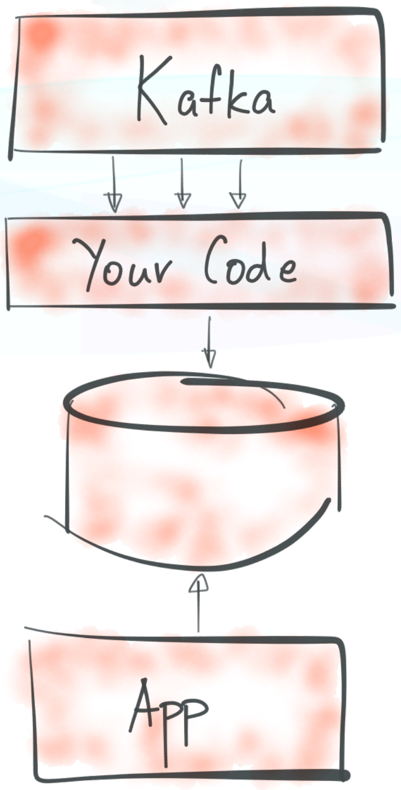
confluent

EXAMPLE ARCHITECTURE



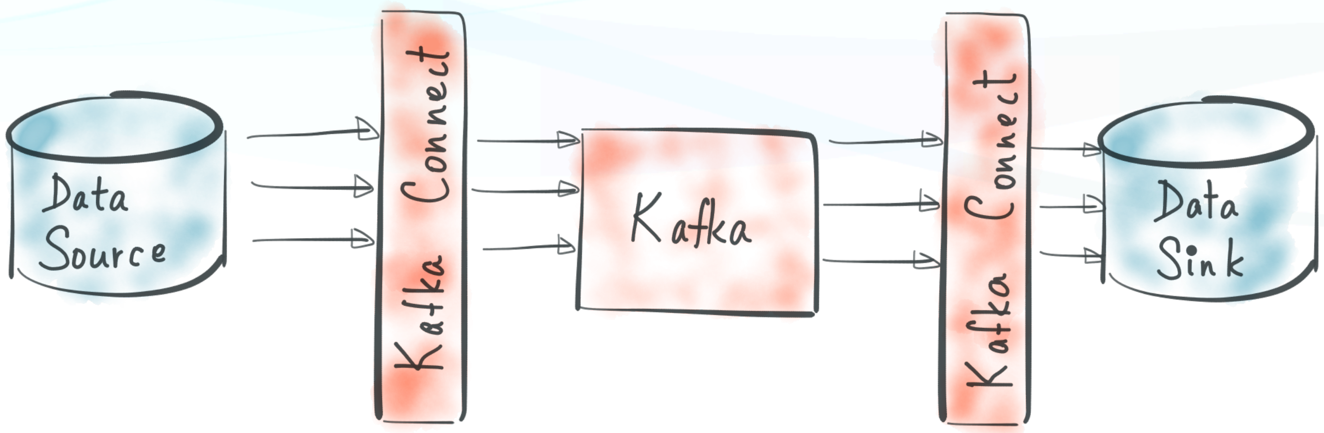
confluent

Kafka Streams



confluent

Kafka Connect



confluent

Kafka Connect Does The Hard Work

1. Scale out
2. Fault tolerance
3. Central management

confluent

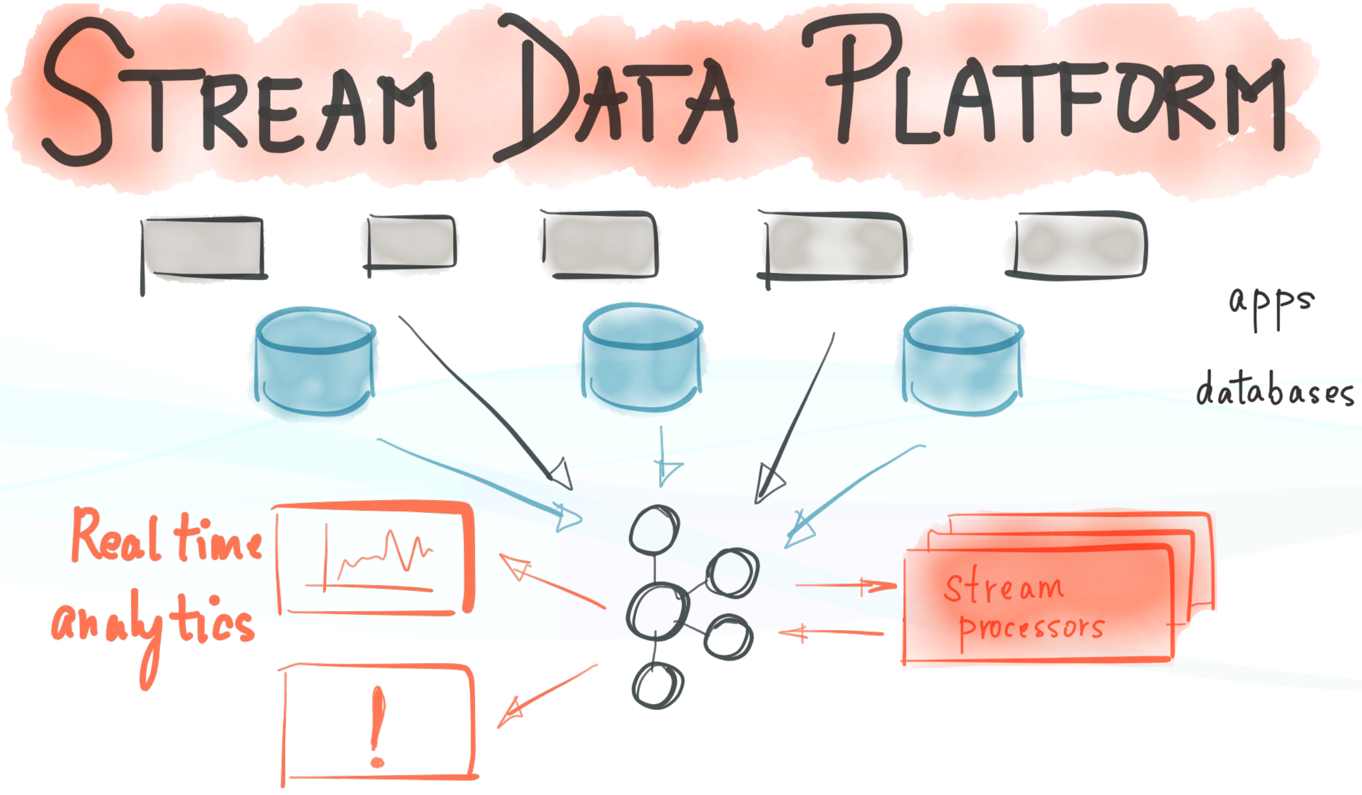
Timeline

- Kafka Connect
- Kafka Streams

Next Steps

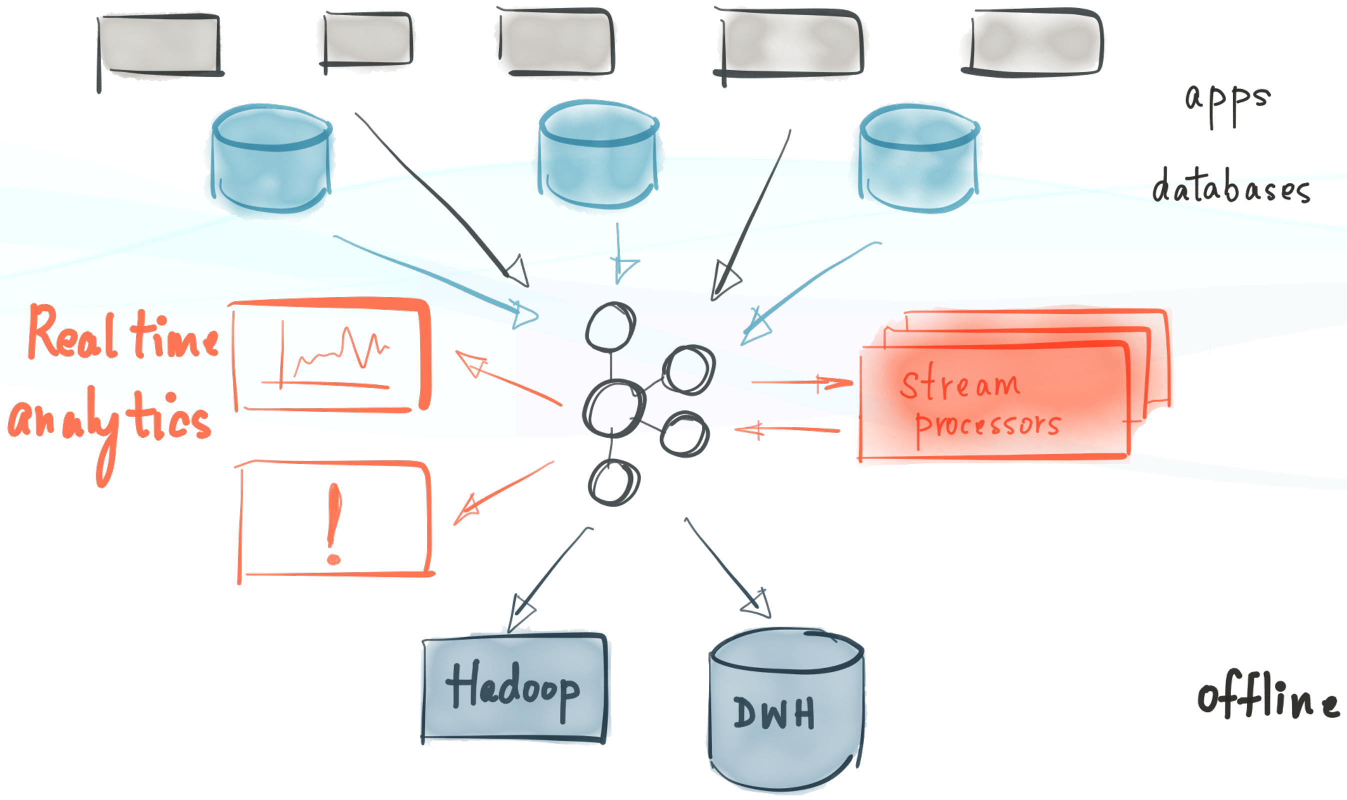
- Expanded DSL
- Exactly-once

confluent


Stream Data Platform
apps
databases
Real time analytics
!
Stream processors

confluent

Stream Data Platform



confluent

CONFLUENT PLATFORM = KAFKA++

Confluent Platform

- Schemas
- Clients
- REST
- Connectors
- Management tools

. . .

kafkasummit

April 26, 2016
San Francisco, CA

www.kafka-summit.org

Organized by confluent

THE END

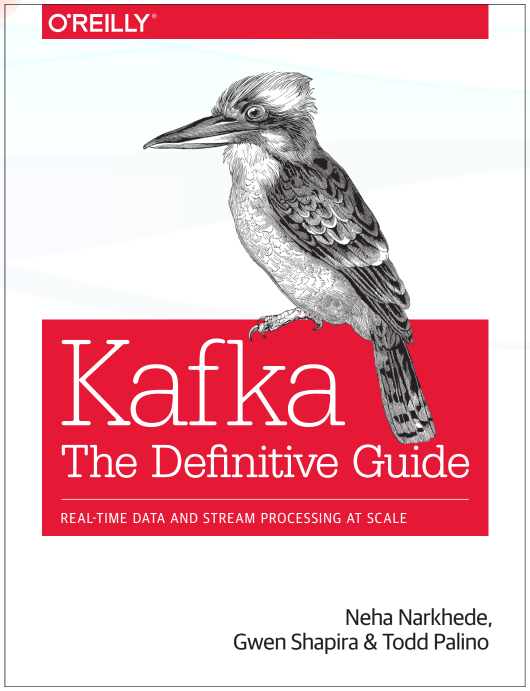
@nehanarkhede | confluent.io/blog

Download Apache Kafka
and Confluent Platform

apache kafka

www.confluent.io/download

confluent


O'REILLY
Kafka
The Definitive Guide
REAL-TIME DATA AND STREAM PROCESSING AT SCALE
Neha Narkhede,
Gwen Shapira & Todd Palino