

Demystifying STREAM PROCESSING

WITH
APACHE KAFKA

Neha Narkhede, Confluent

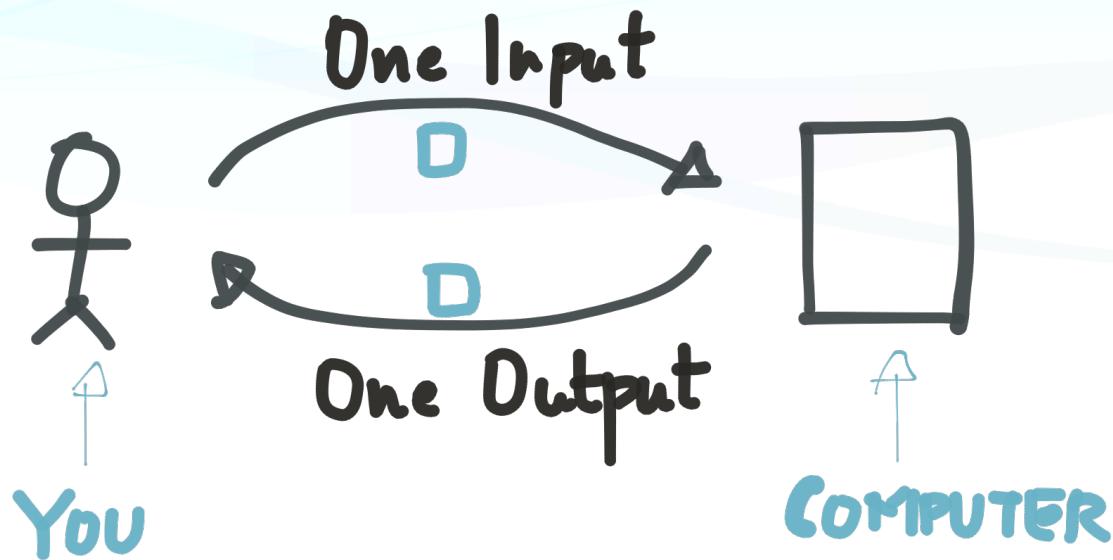
What is STREAM PROCESSING



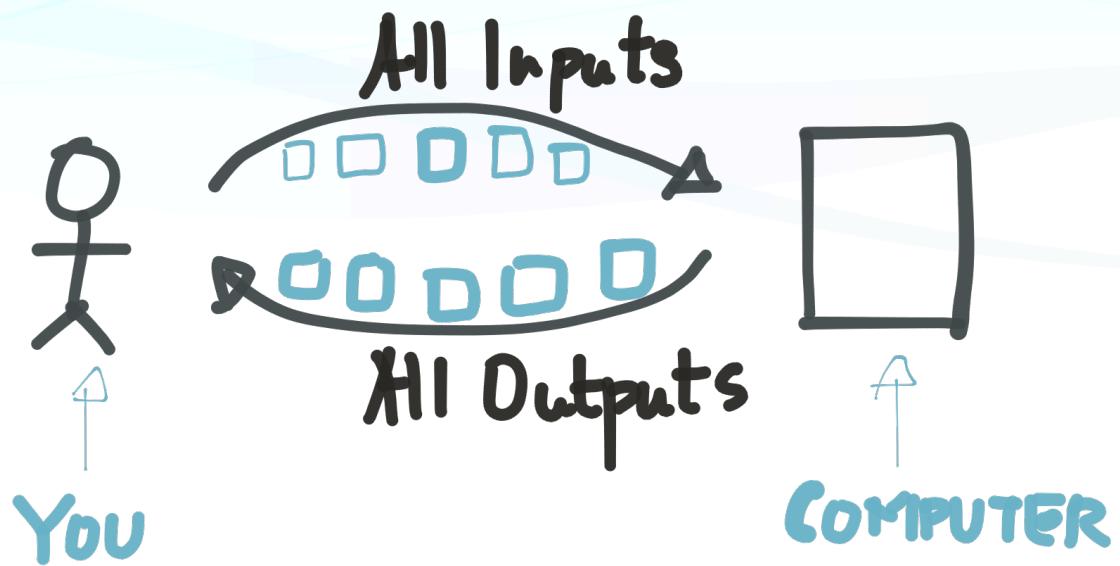
3 PARADIGMS FOR PROGRAMMING

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

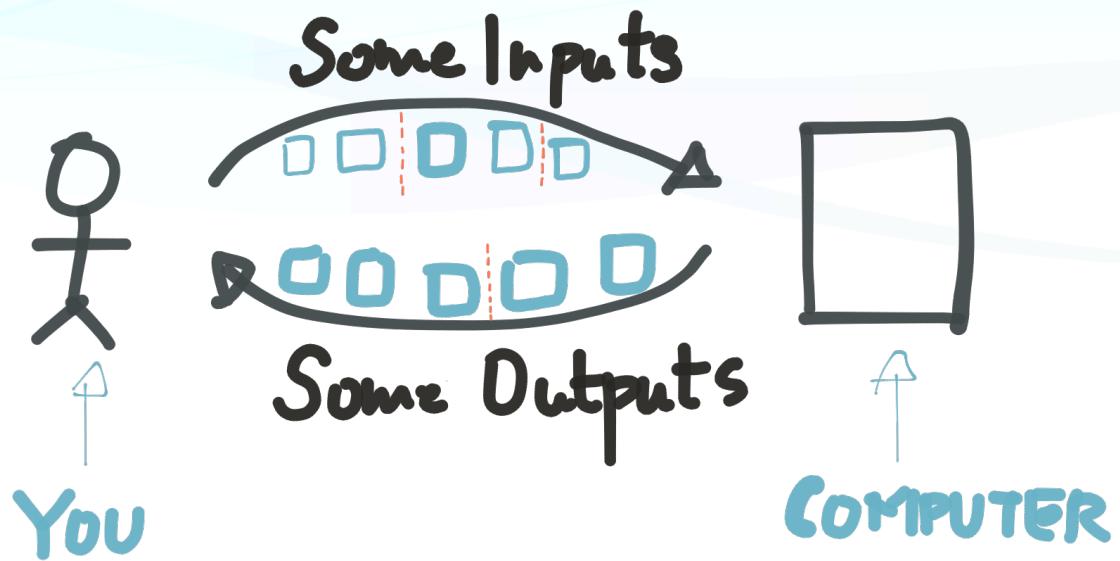
REQUEST / RESPONSE



BATCH



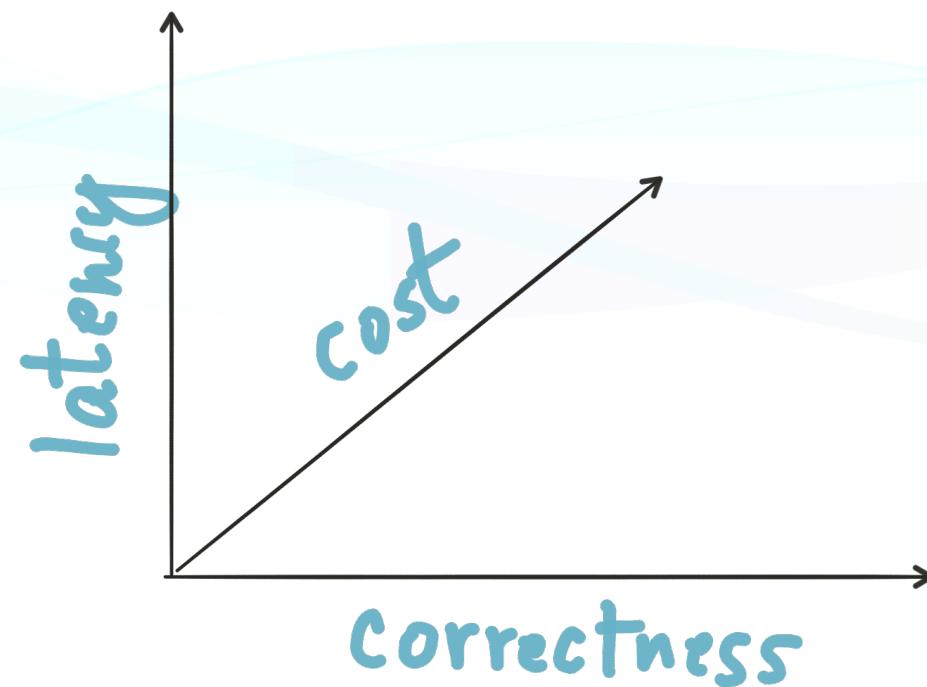
STREAM PROCESSING



STREAM PROCESSING isn't (NECESSARILY)

- Transient
- Approximate
- Lossy

TRADE OFFS

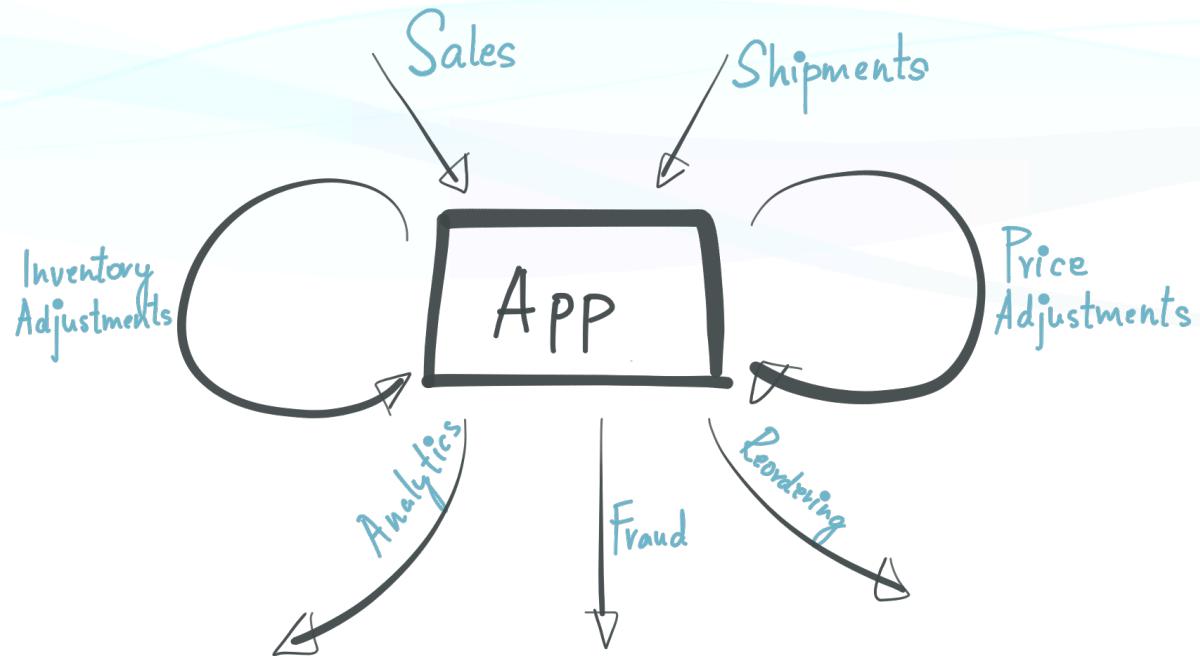


STREAM PROCESSING

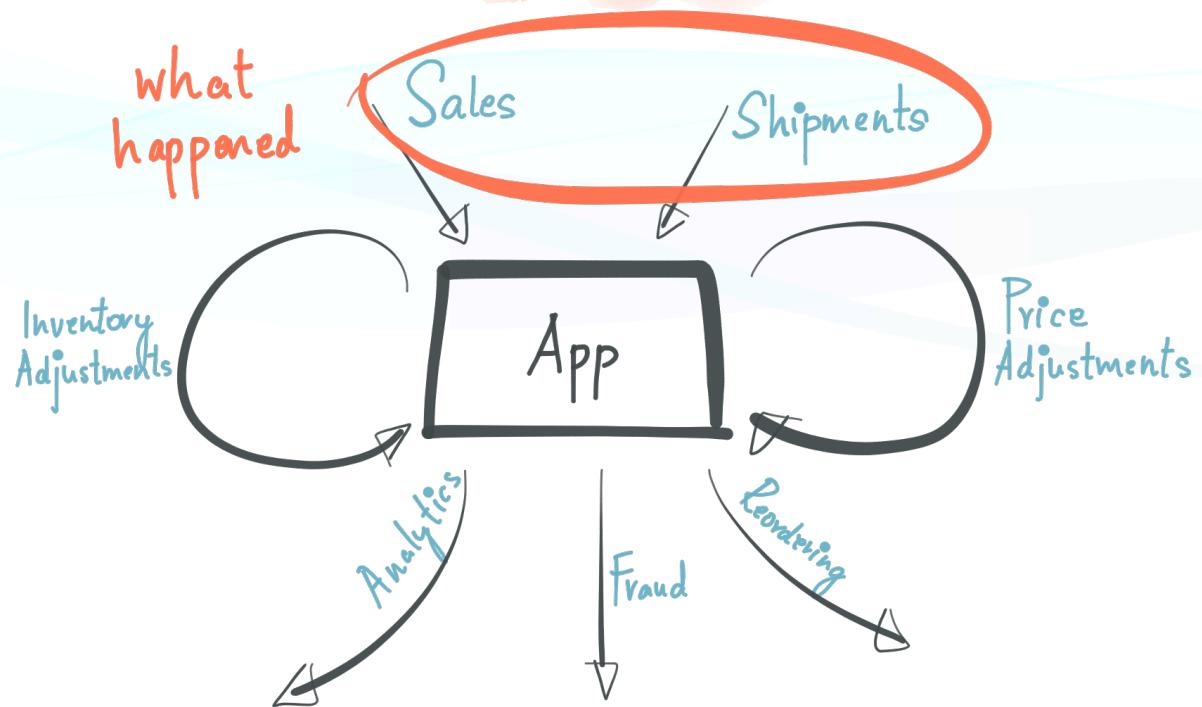
IN THE

LARGE

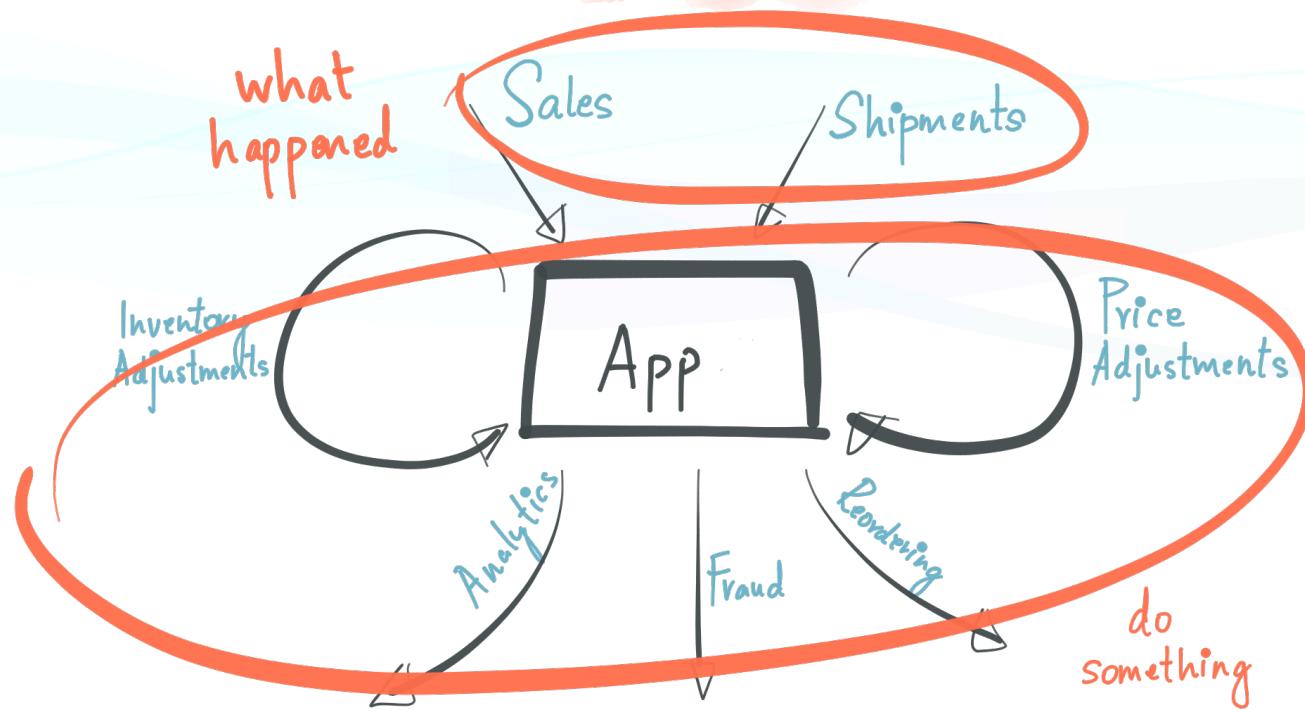
RETAIL



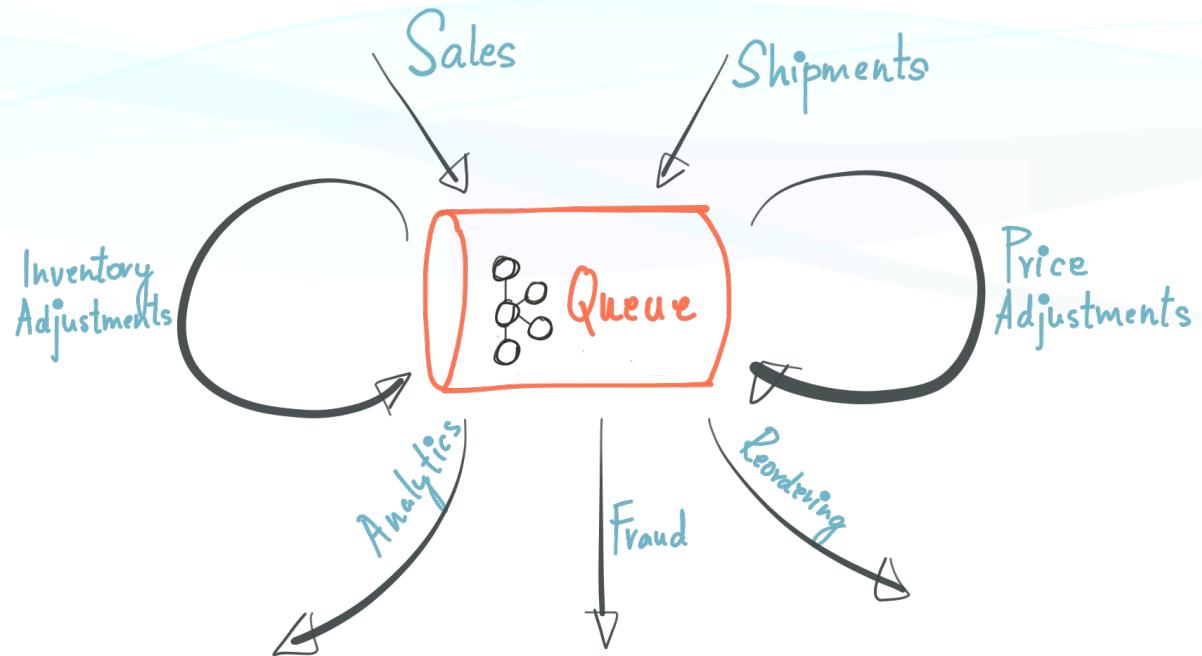
RETAIL



RETAIL



RETAIL



stream processing

=

f ("what happened")

events

stream processing

=

f ("what happened")



STREAM PROCESSING WITH KAFKA

2 approaches

APPROACH #1

DIY!

stream processing

PROBLEM: STREAM PROCESSING HAS SOME hard parts

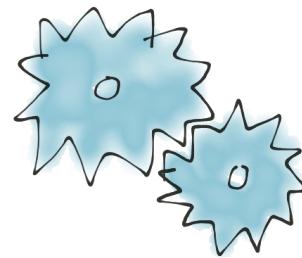
- partitioning & scalability
- Semantics & fault tolerance
- state
- windowing & time
- reprocessing

APPROACH #2

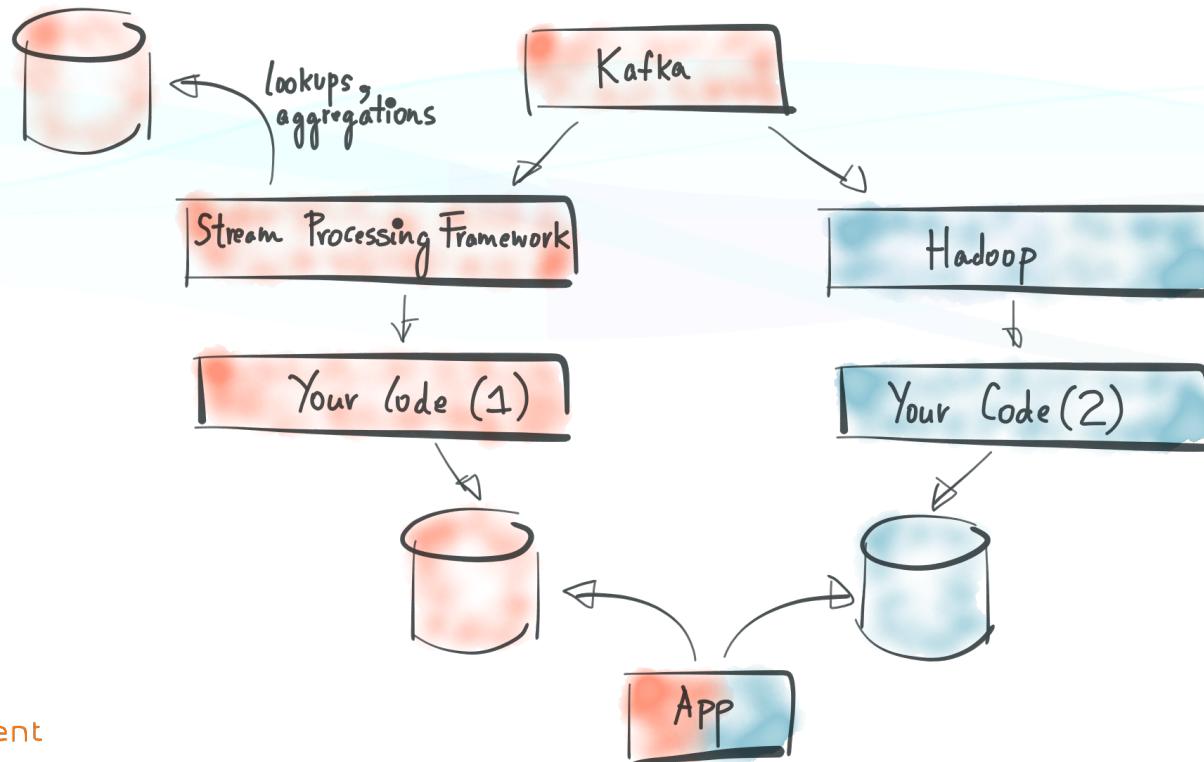
Use a
stream
processing
framework

- SPARK
- STORM
- SAMZA
- FLINK
- ET AL

PROBLEM : LOTS OF
moving parts !



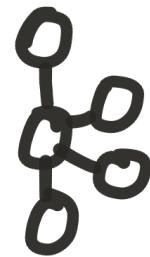
EXAMPLE ARCHITECTURE



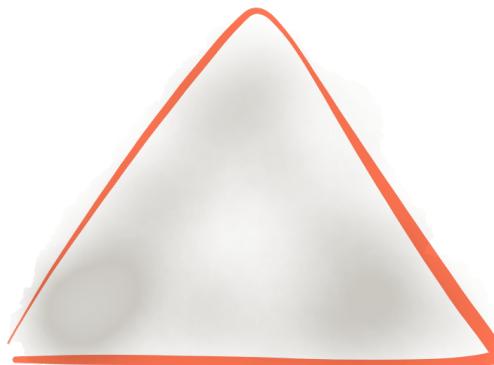
THE HARD PART OF DISTRIBUTED SYSTEMS

Coding < Debugging < Operations

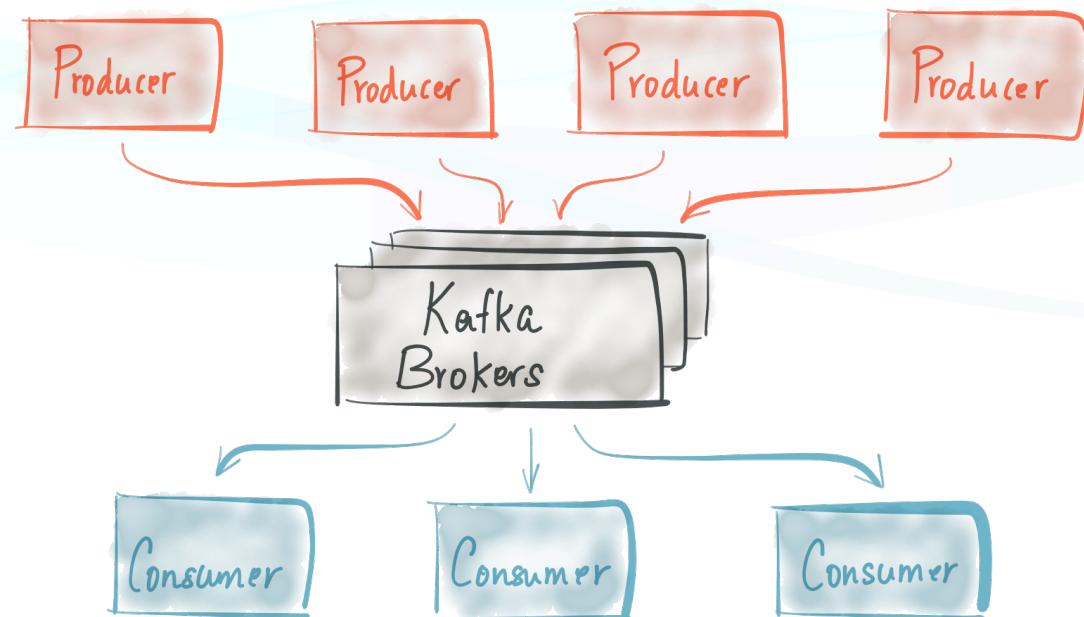
IS THERE A
better
WAY ?



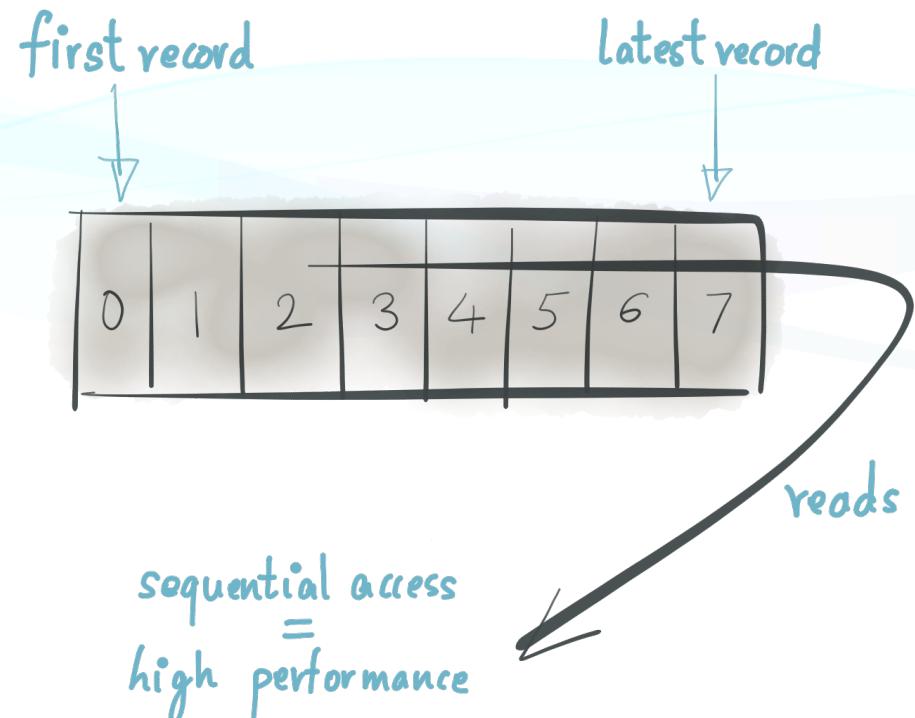
KAFKA
PROVIDES
Primitives



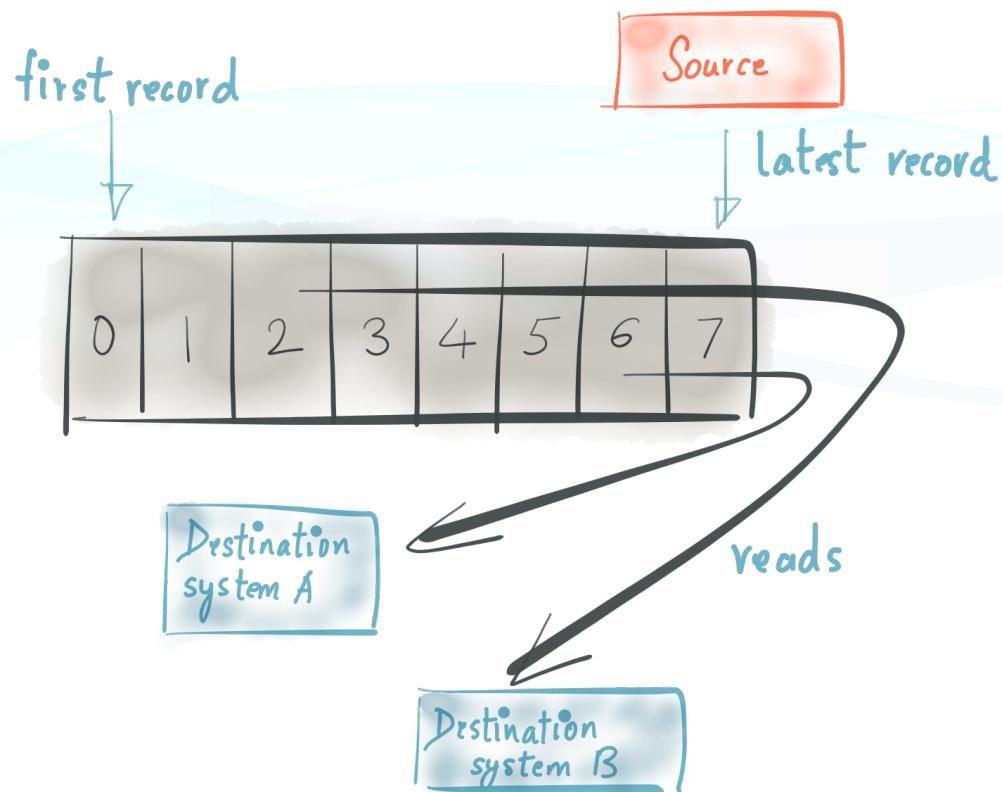
APACHE KAFKA : 10,000 FT VIEW

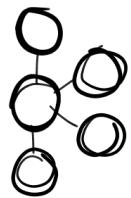


KAFKA KEY IDEA : LOG

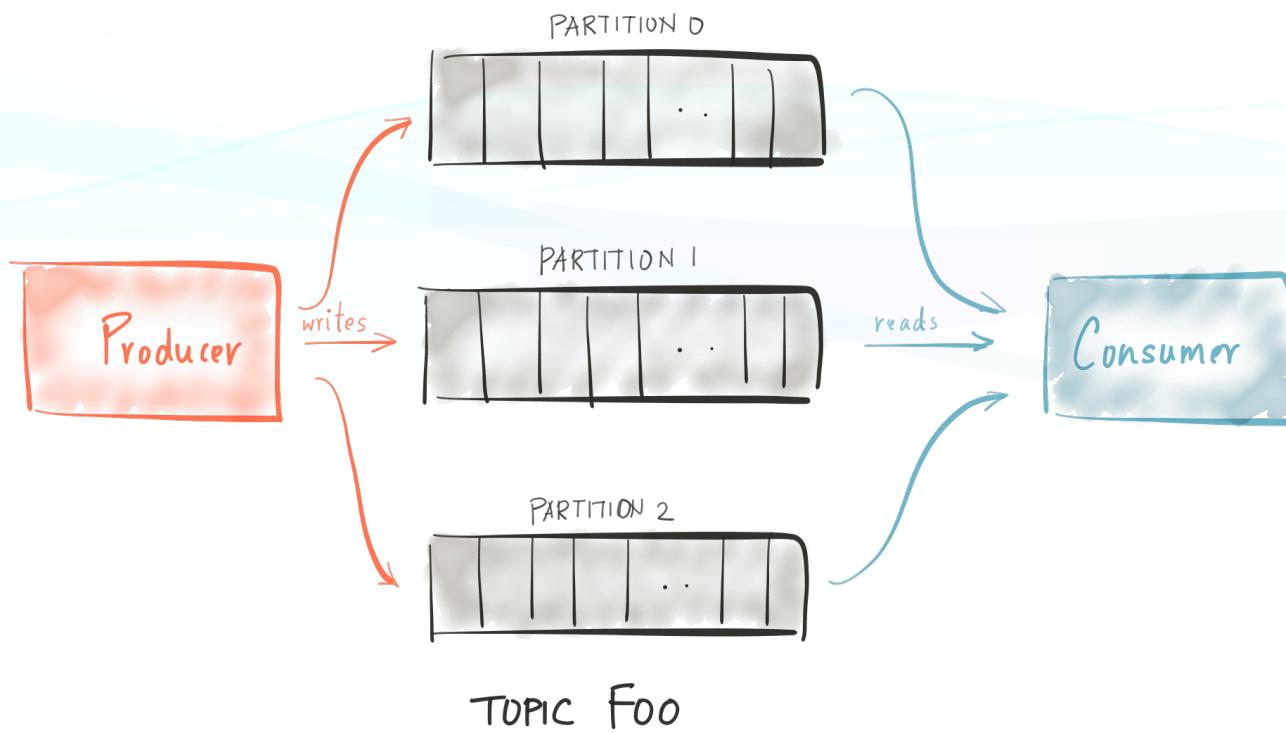


LOGS & PUB-SUB



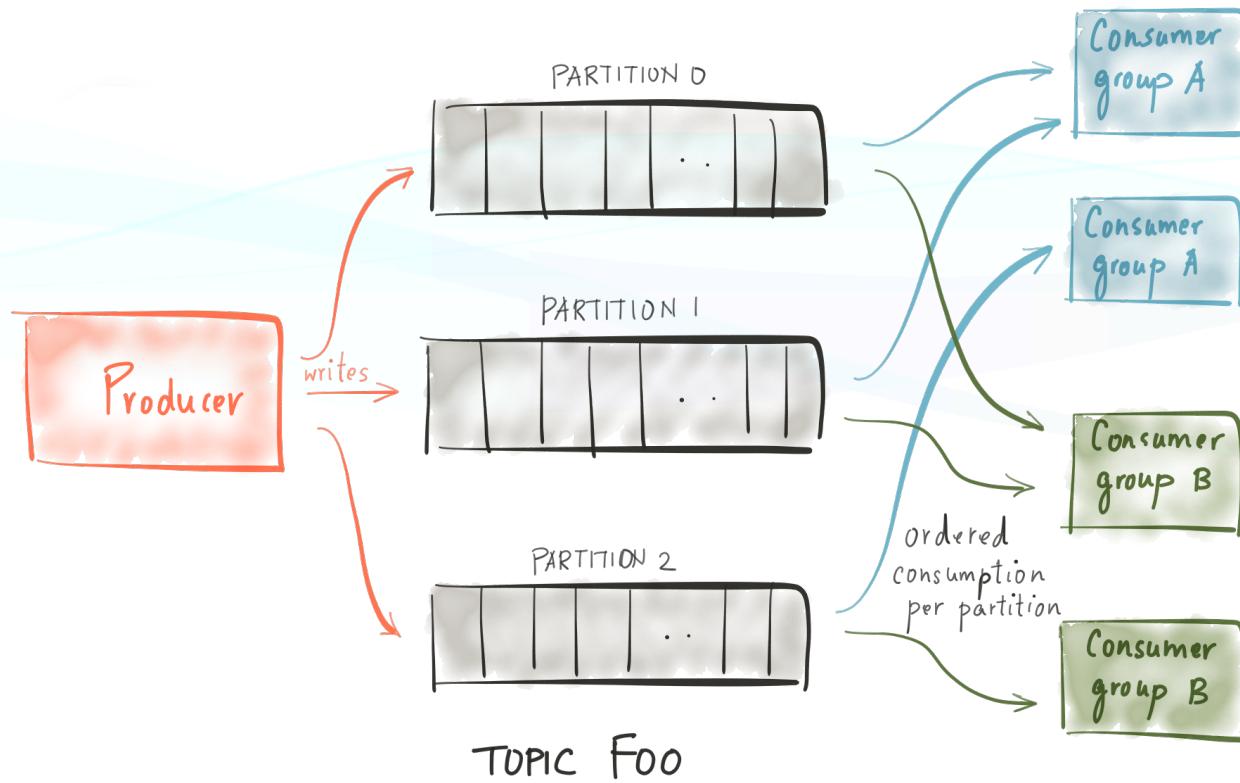


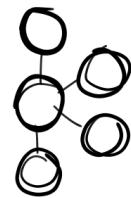
Topic \approx PARTITIONED LOG



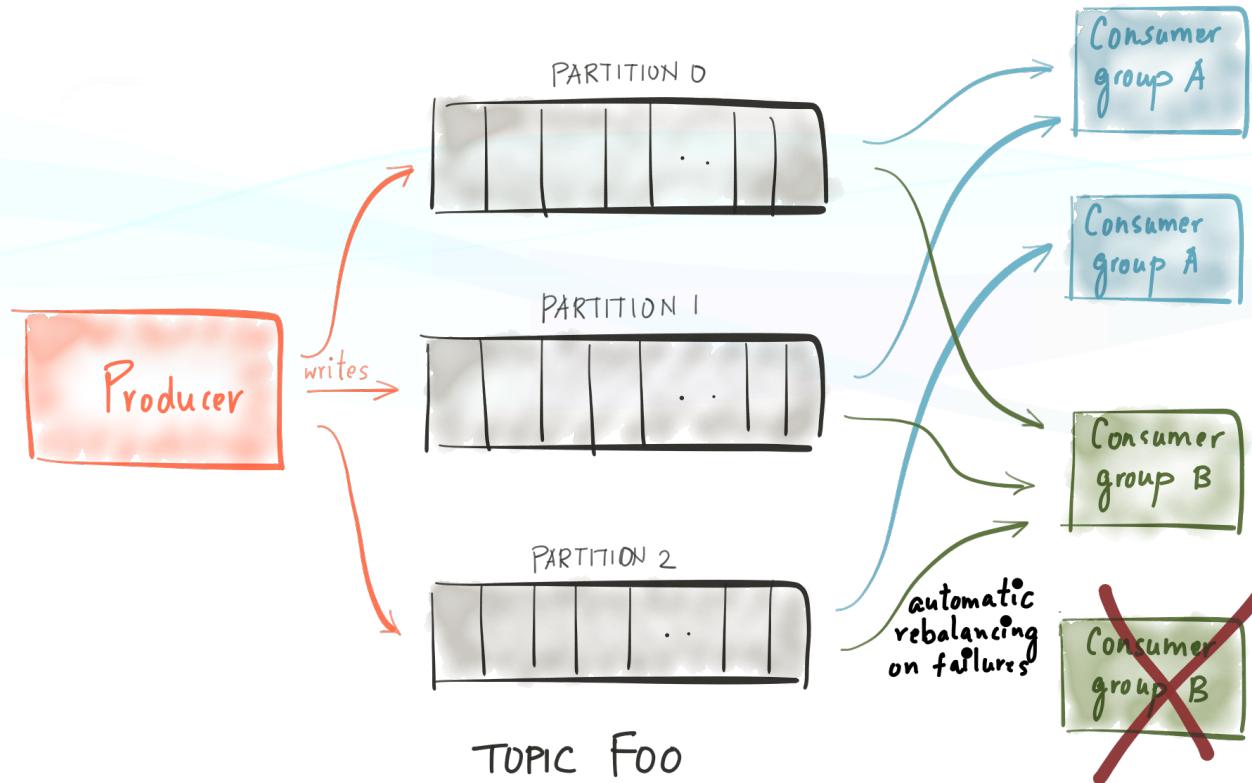


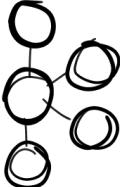
SCALABLE CONSUMPTION



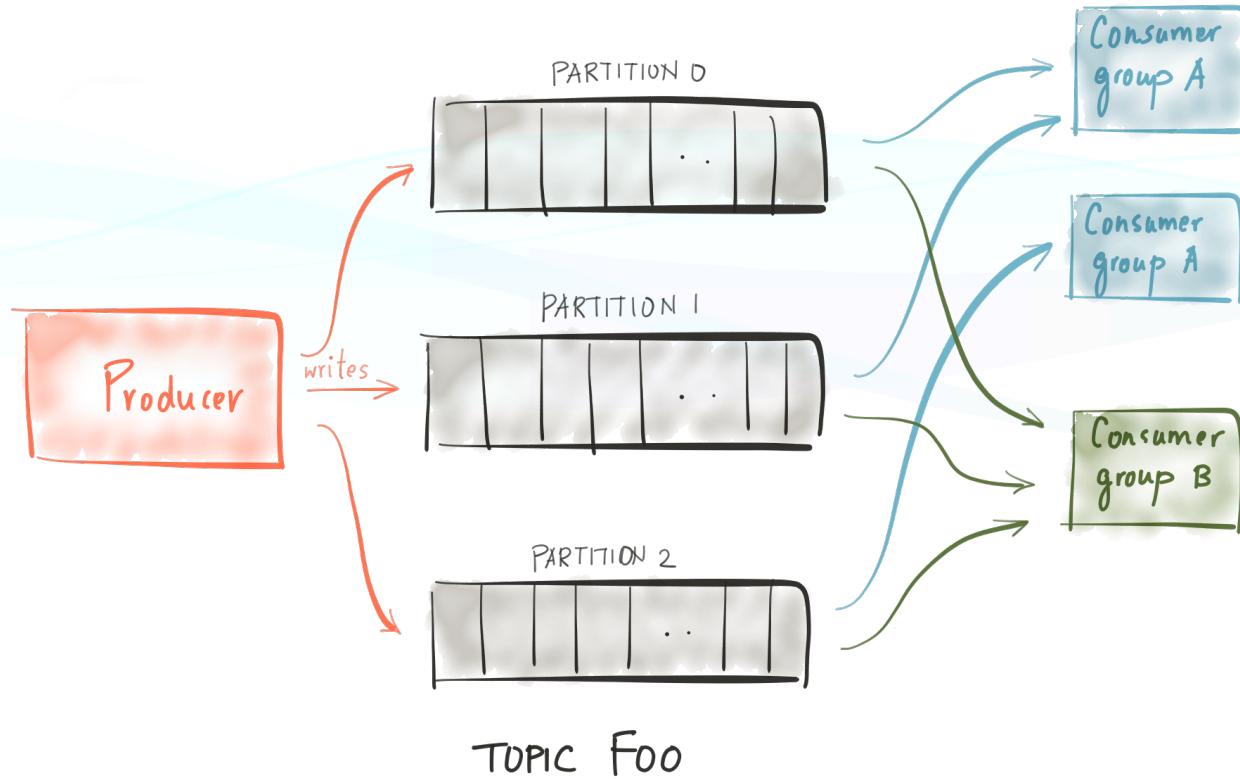


FAULT TOLERANT CONSUMPTION



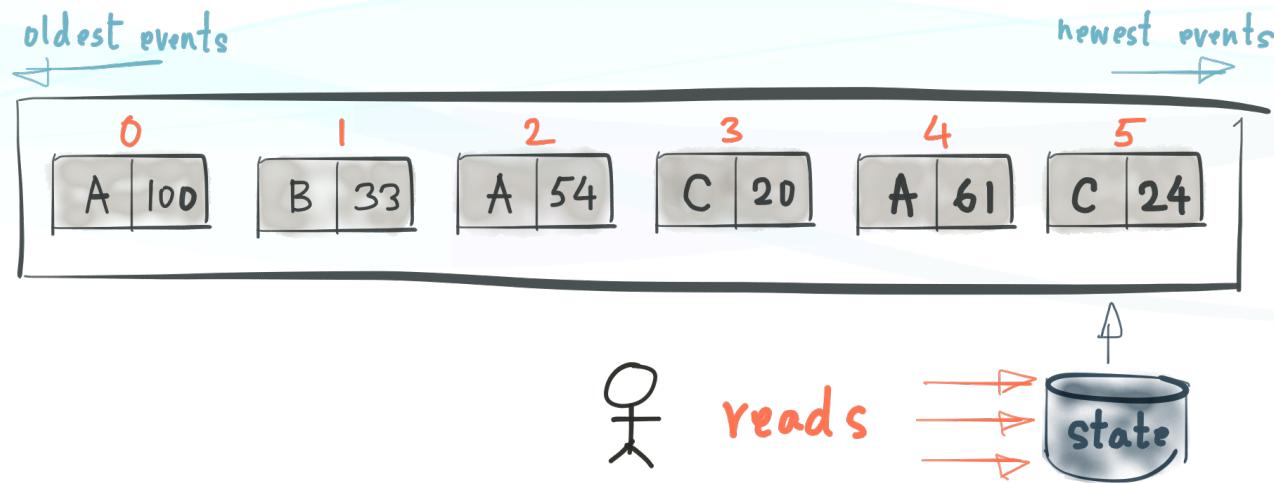


FAULT TOLERANT CONSUMPTION



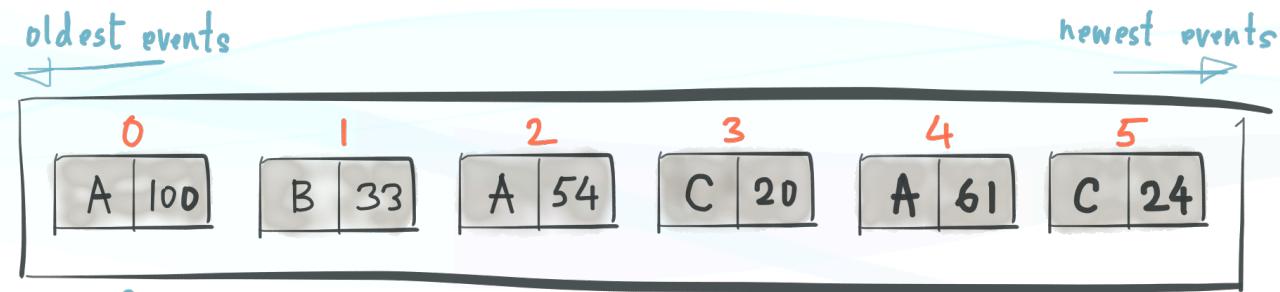


REPROCESSING

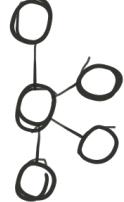




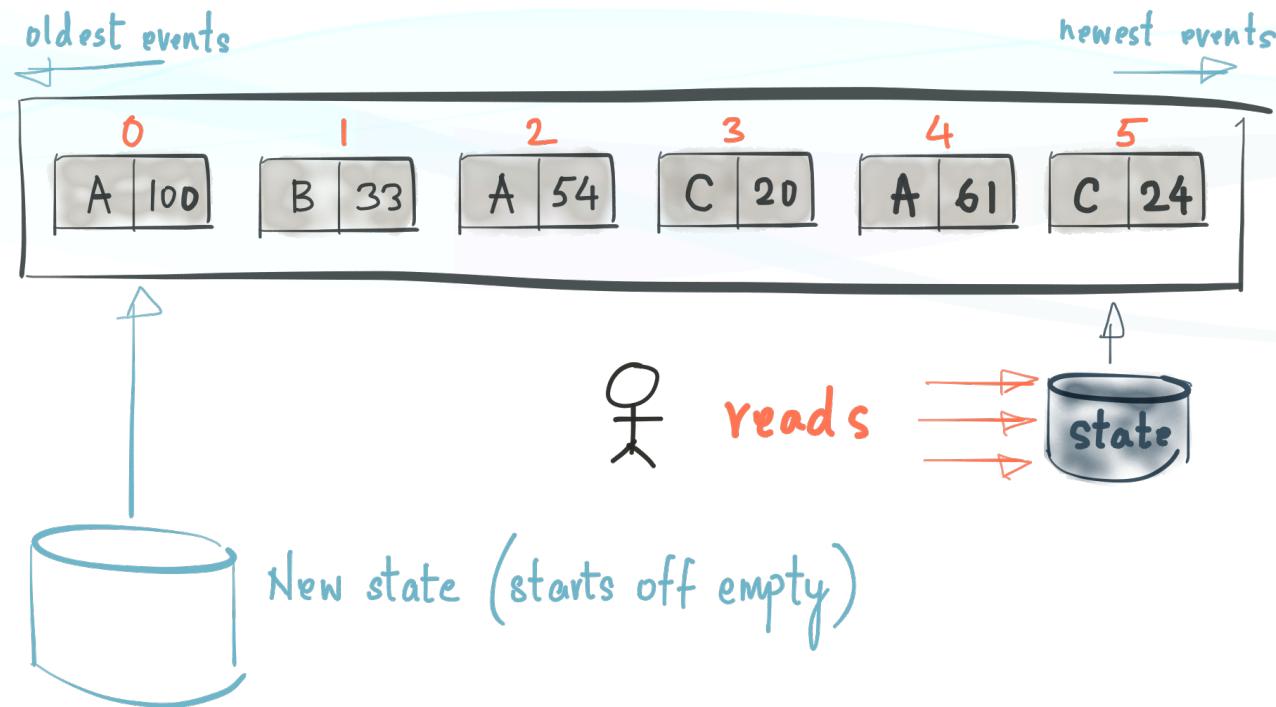
REPROCESSING

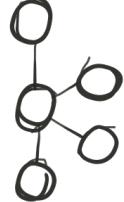


set offset = 0

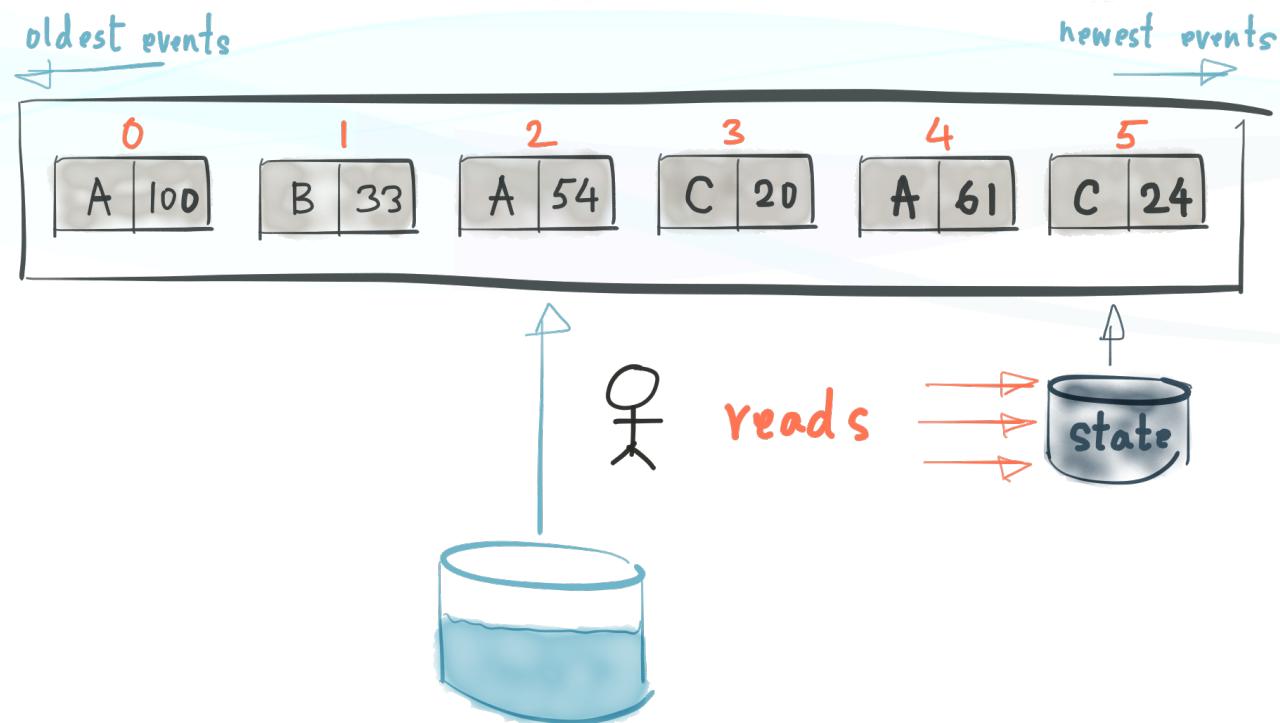


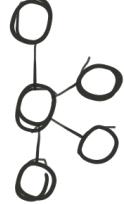
REPROCESSING



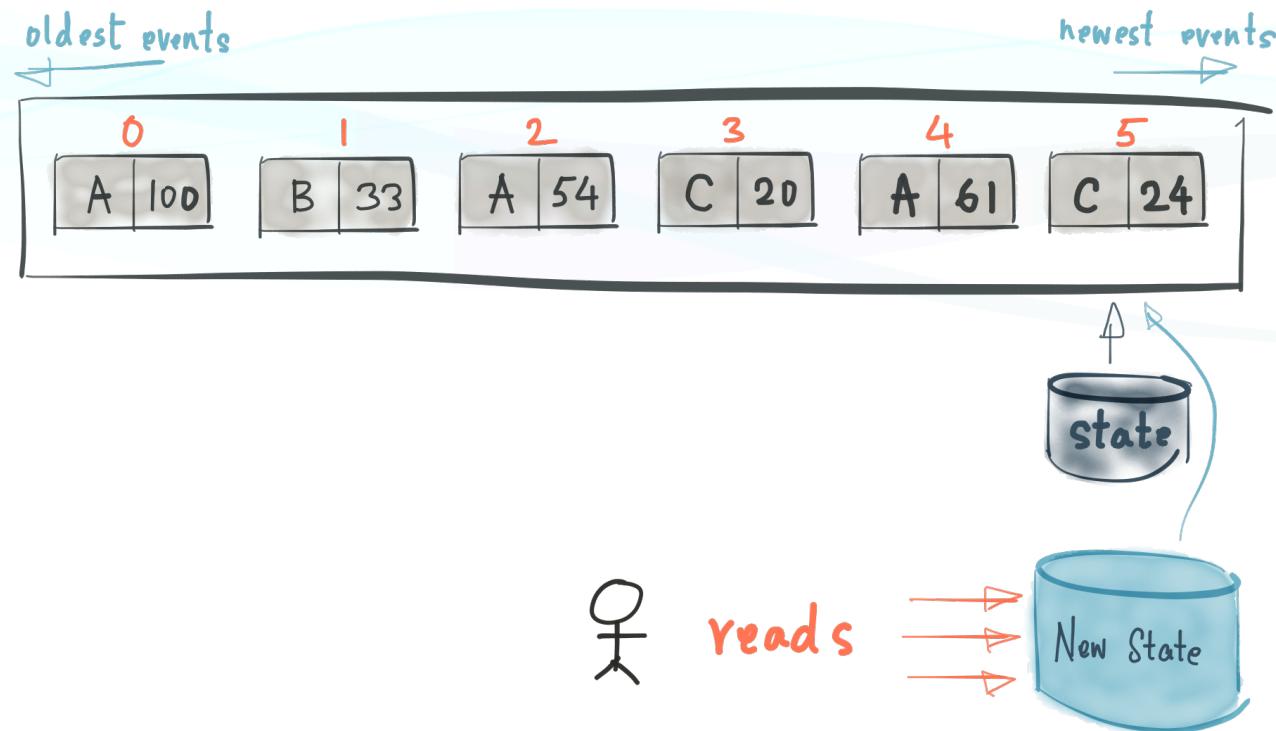


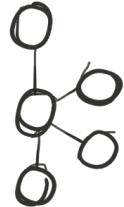
REPROCESSING



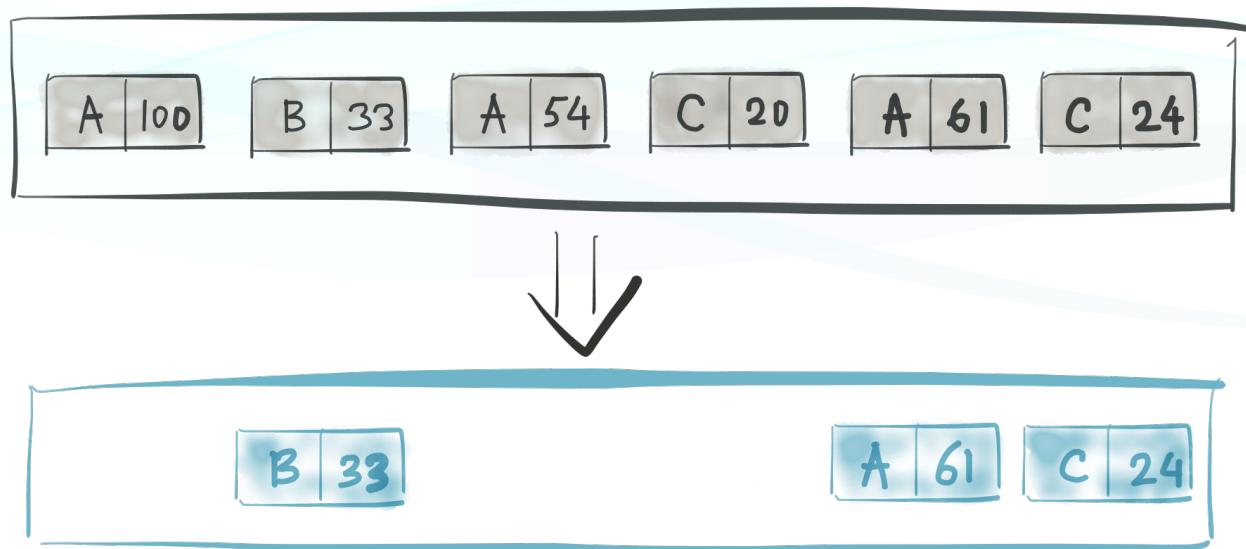


REPROCESSING





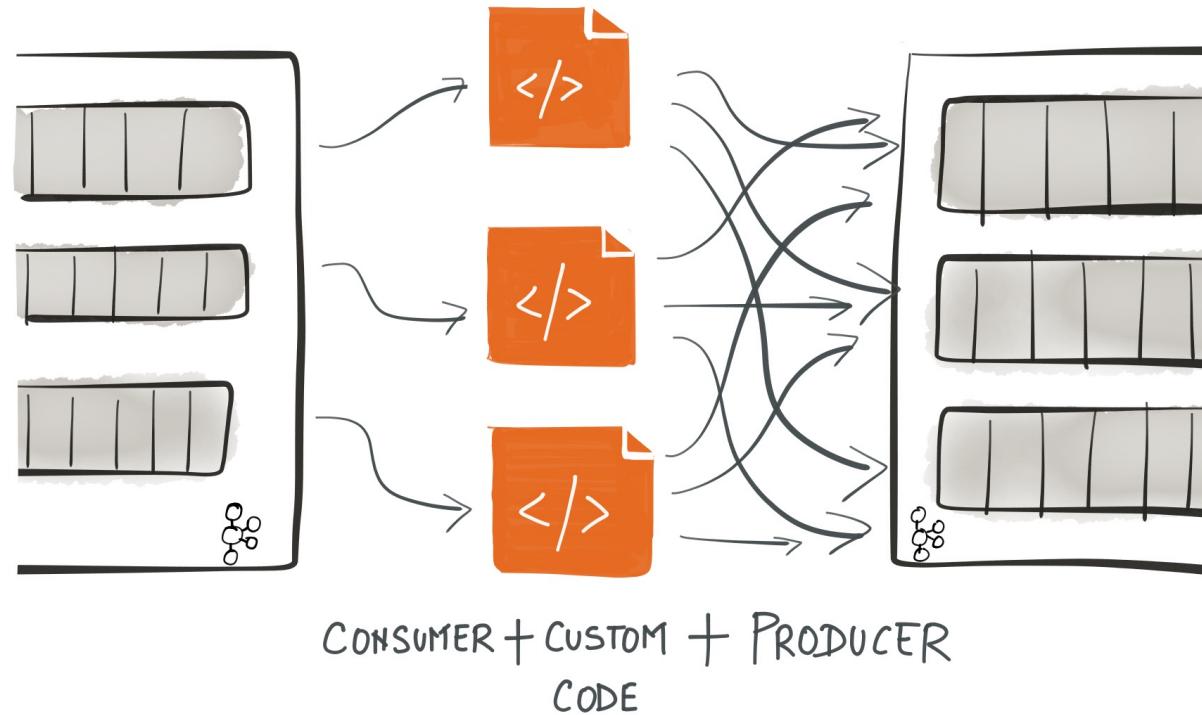
LOG COMPACTION



Logs unify BATCH & STREAM PROCESSING

0	1	2	3	4	5	6
---	---	---	---	---	---	---

STREAM PROCESSING





KAFKA STREAMS

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

LIBRARIES

VS

FRAMEWORKS

PROCESSOR API

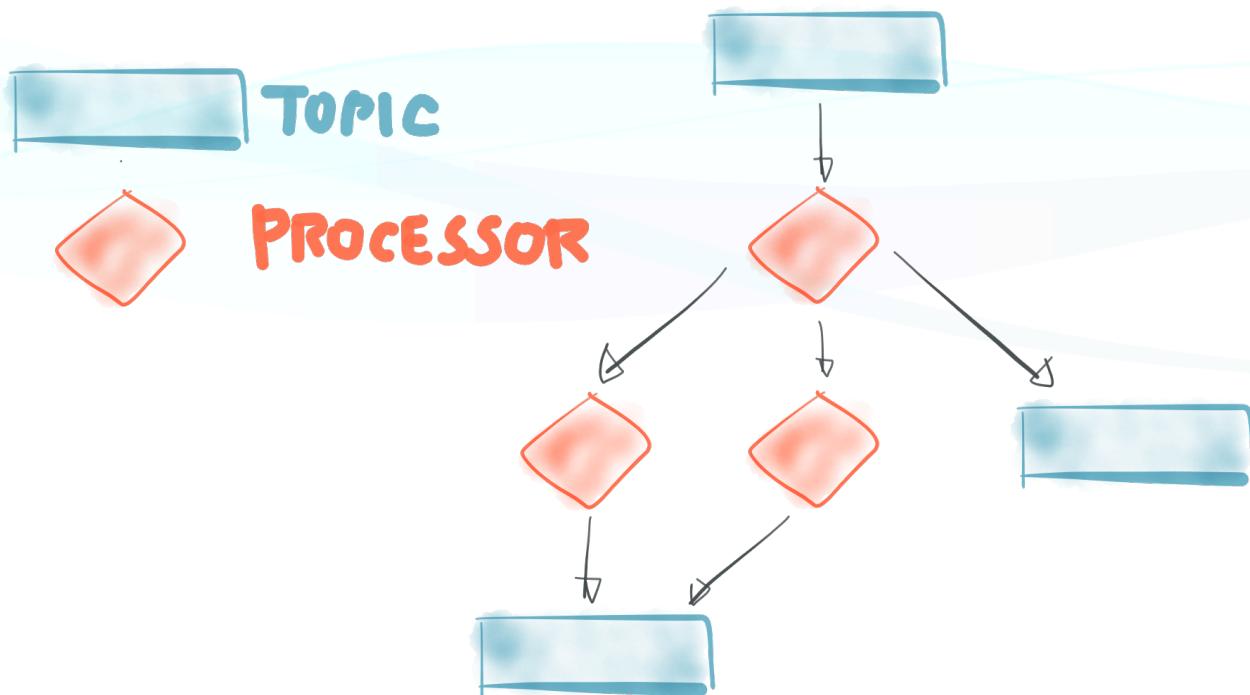
```
/**
 * A processor of messages.
 *
 * @param <K> the type of keys
 * @param <V> the type of values
 */
public interface Processor<K, V> {
    void init(ProcessorContext context);

    void process(K key, V value);

    void punctuate(long timestamp);

    void close();
}
```

DAGS



PROCESSOR API

```
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");

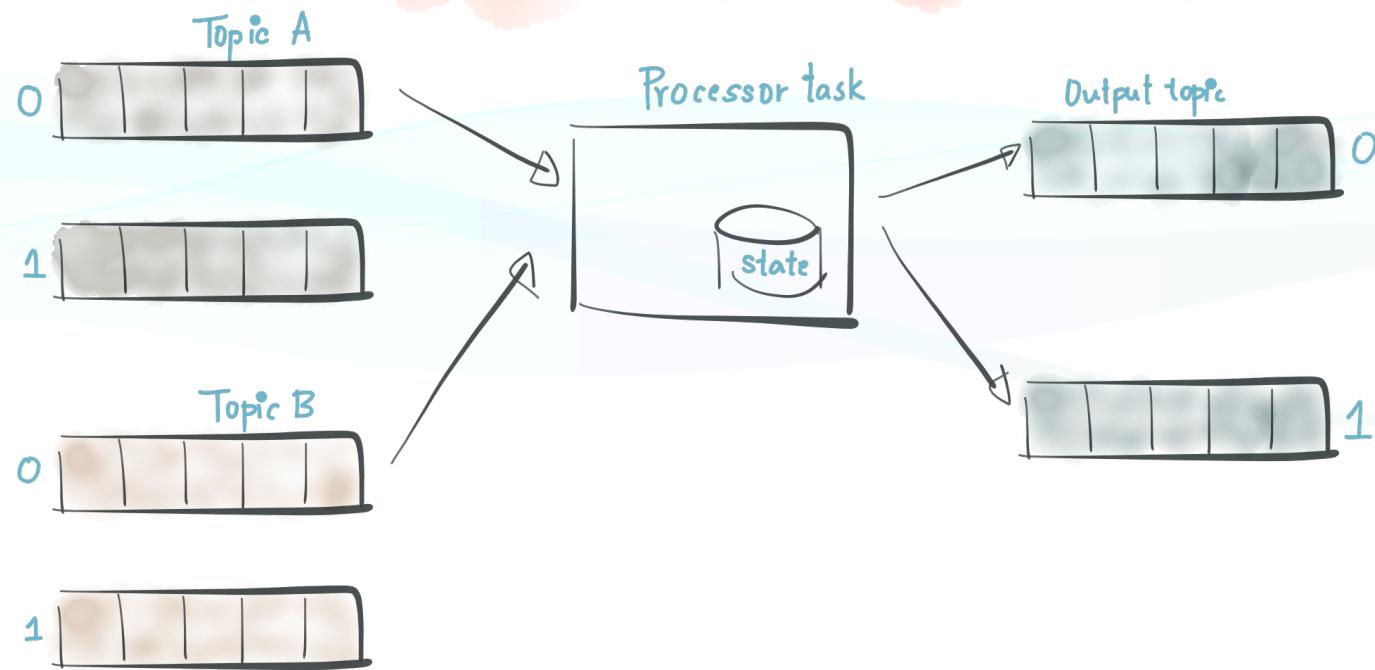
KStream<String, Integer> stream2 =
    stream1.map((key, value) -> new KeyValue<>(key, value.cost()))
        .aggregate(0, (oldval, newval) -> oldval + newval);

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

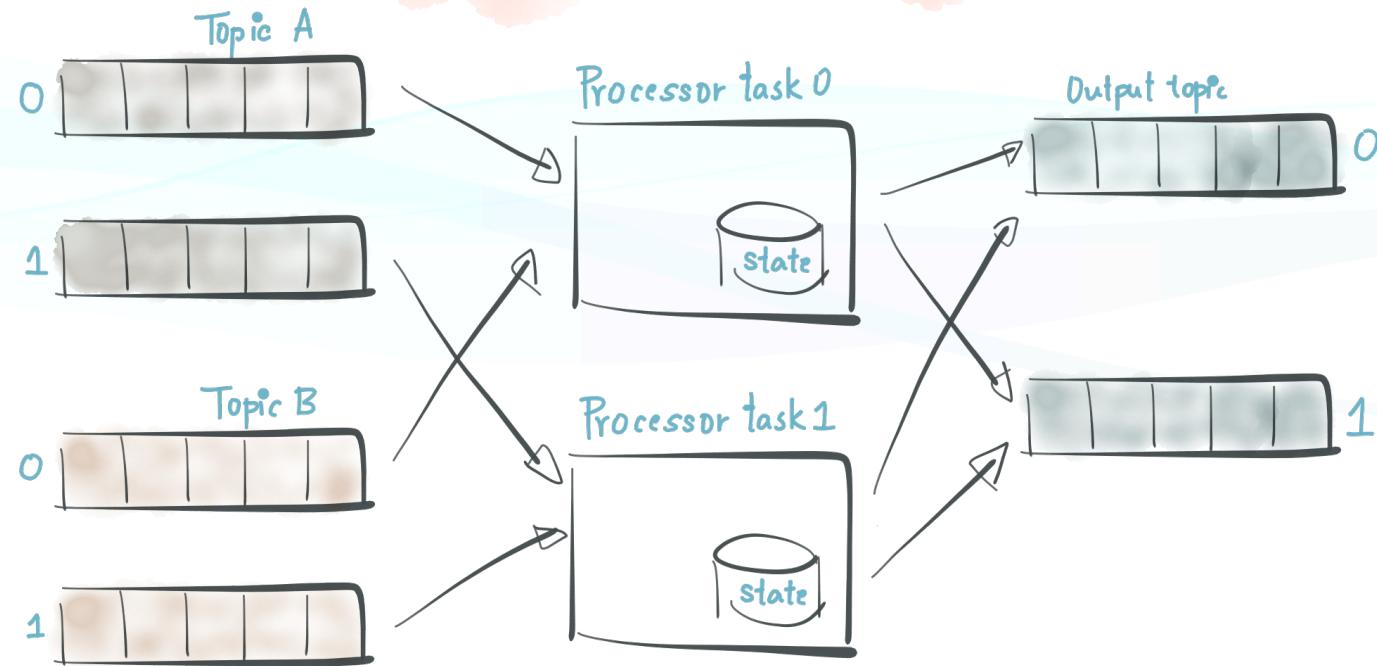
streams[0].sendTo("topic2");
streams[1].sendTo("topic3");

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

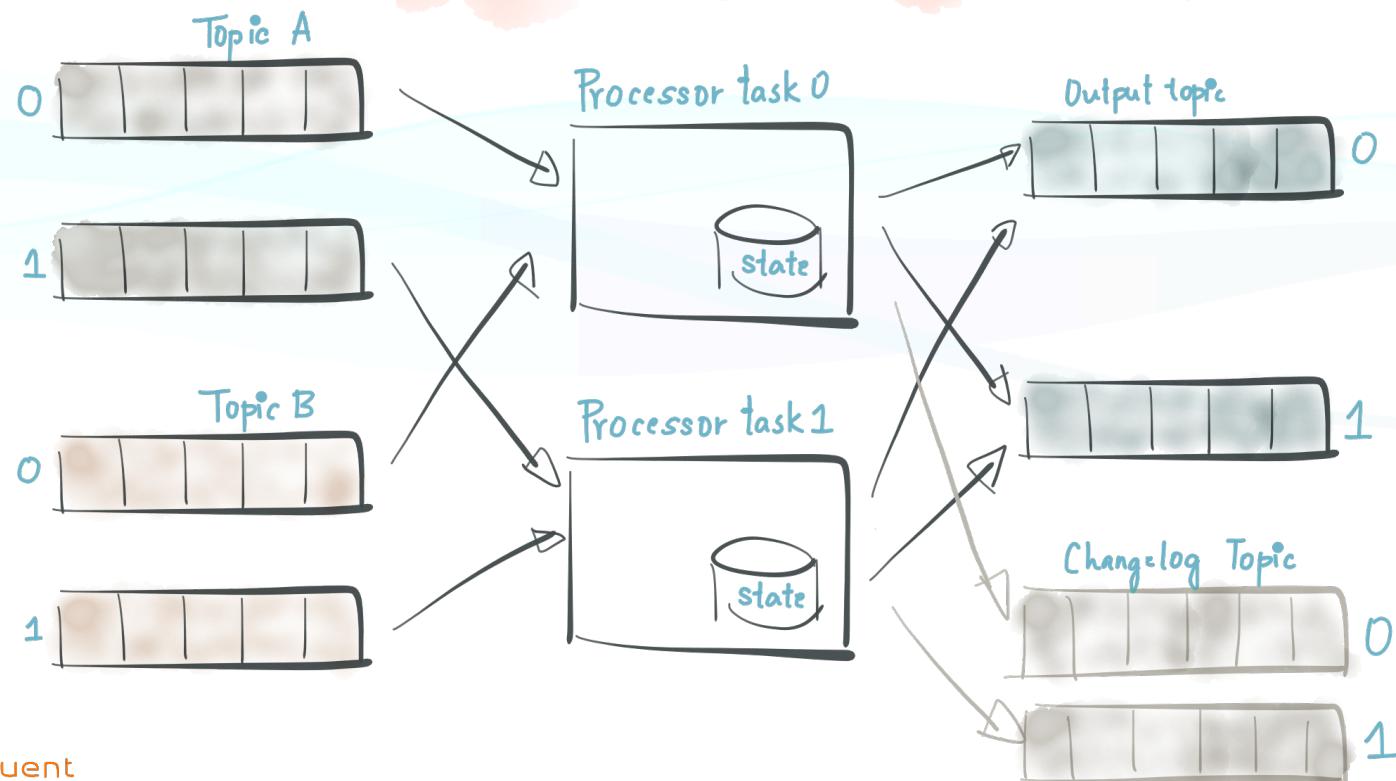
EXECUTION MODEL



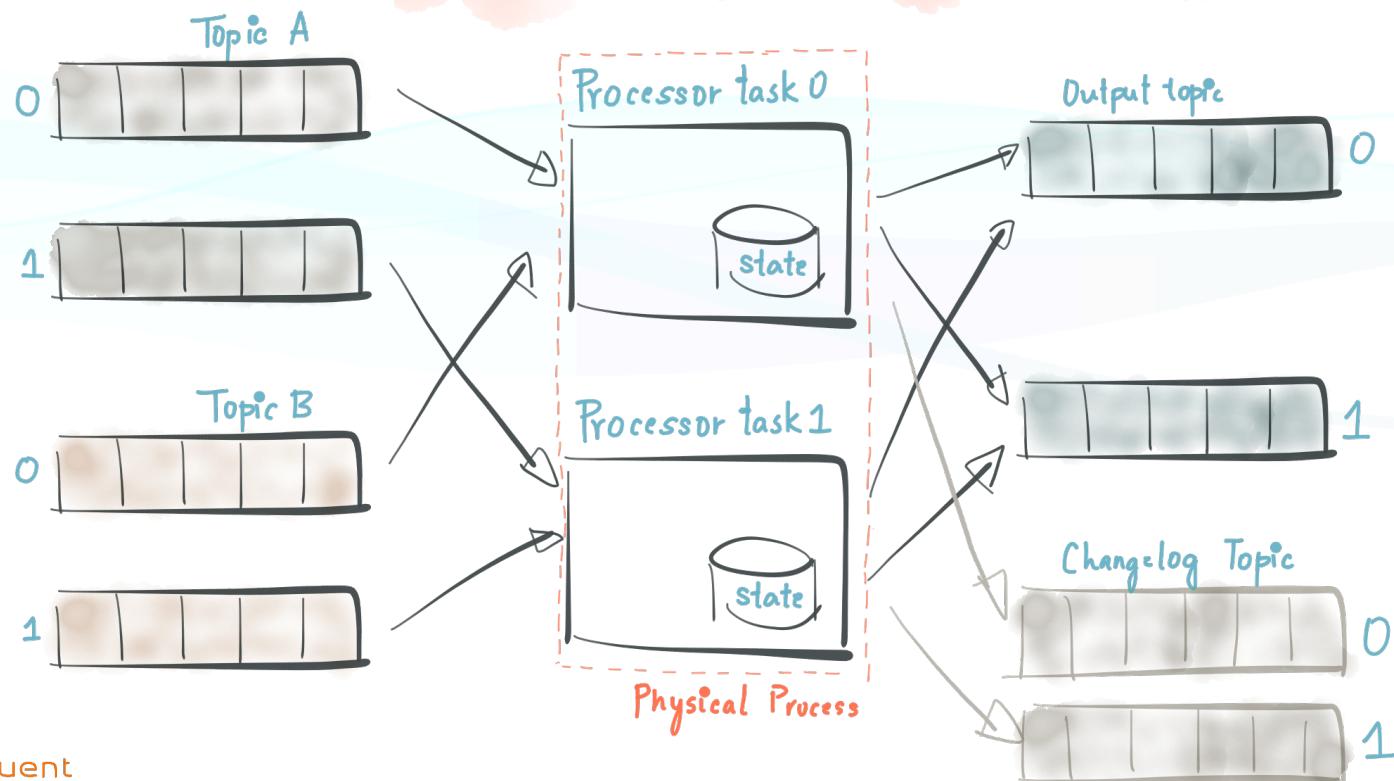
EXECUTION MODEL



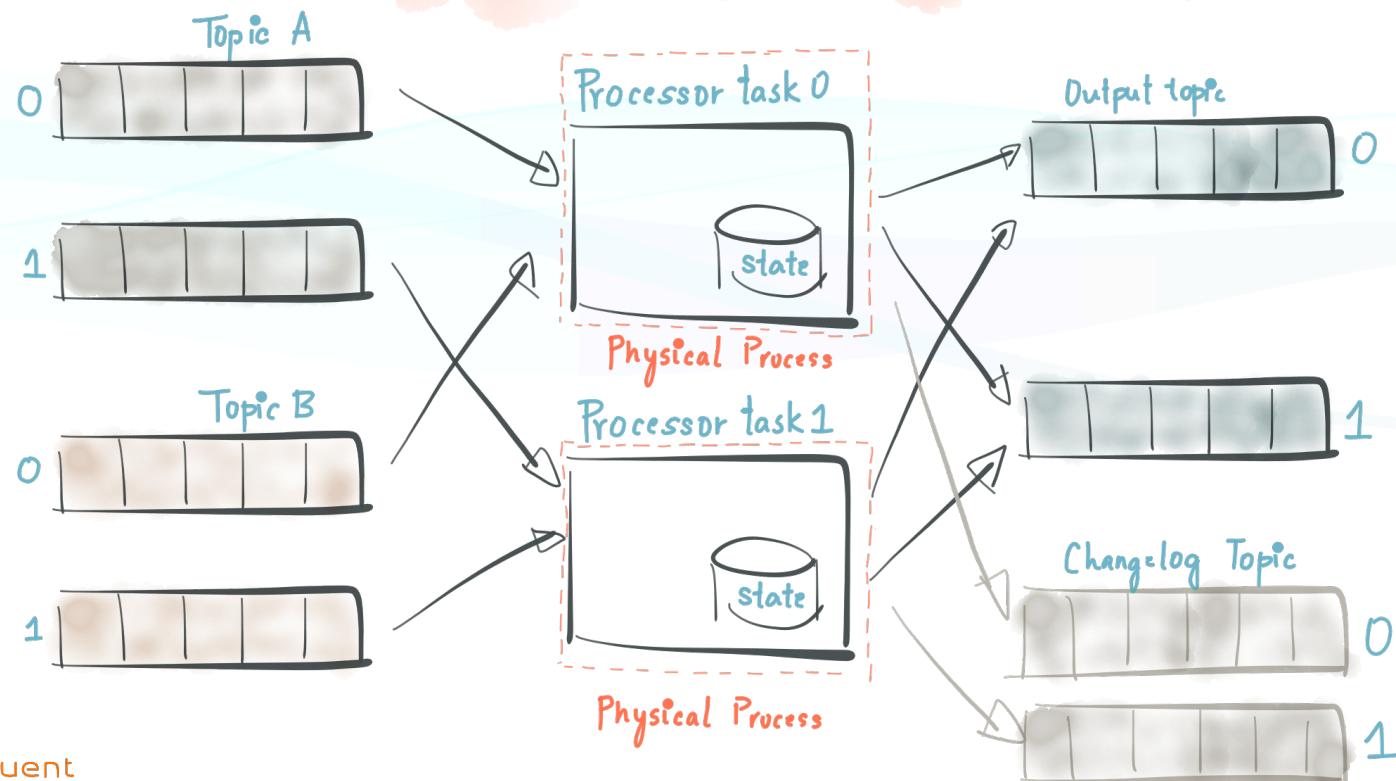
EXECUTION MODEL



EXECUTION MODEL



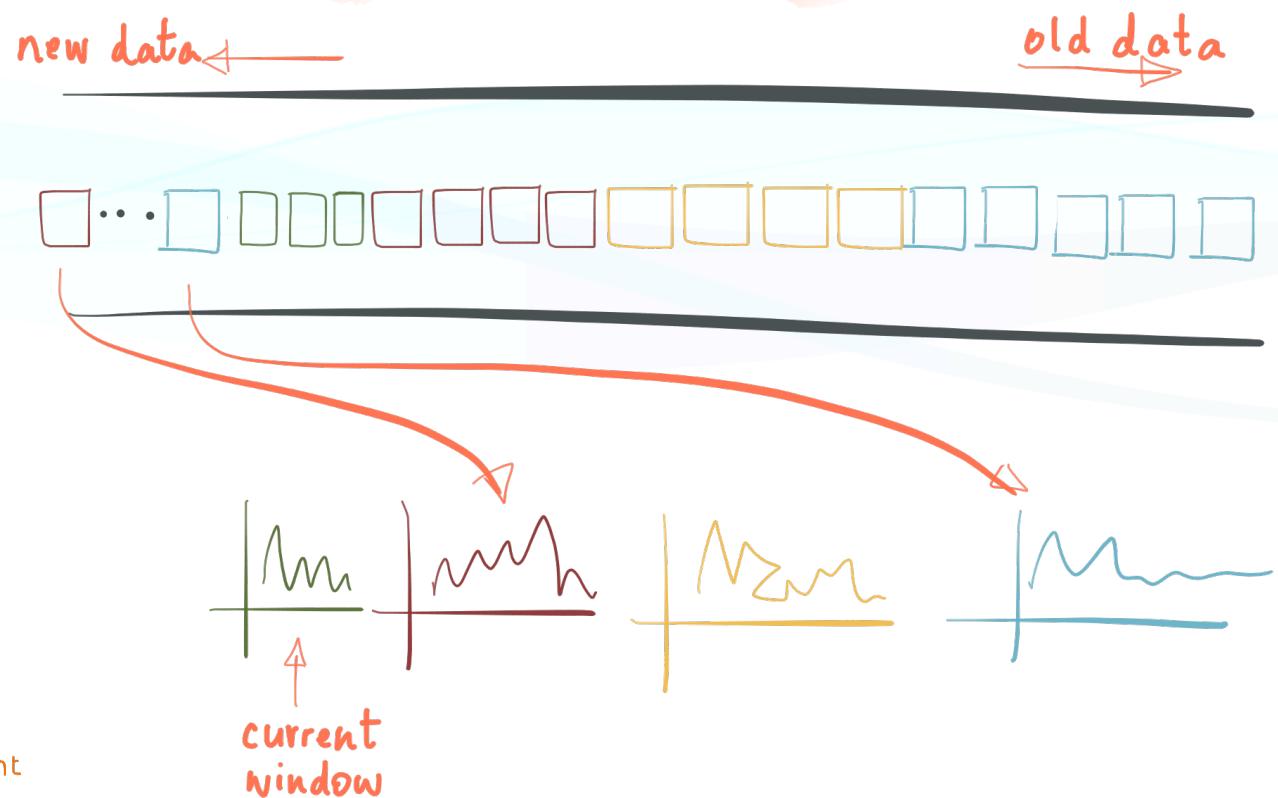
EXECUTION MODEL



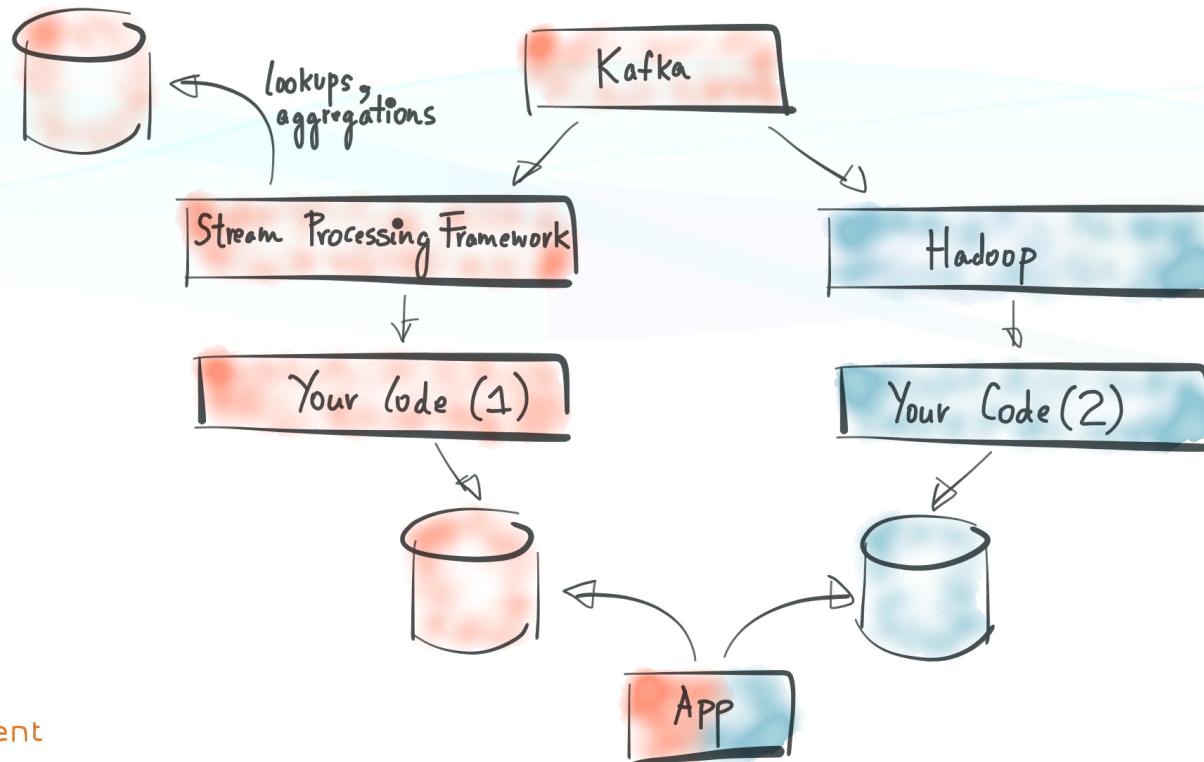
TIME



WINDOWING

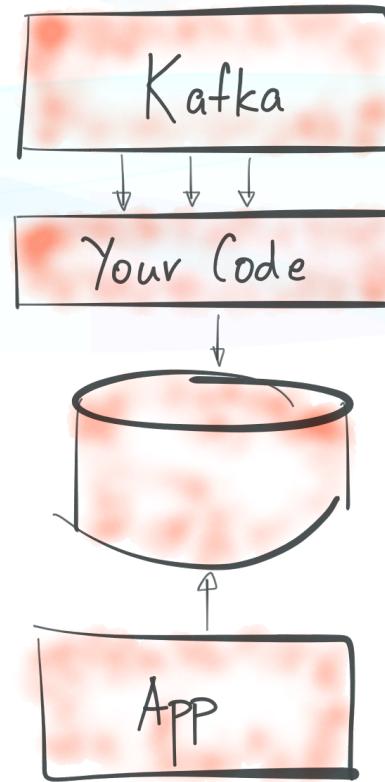


EXAMPLE ARCHITECTURE



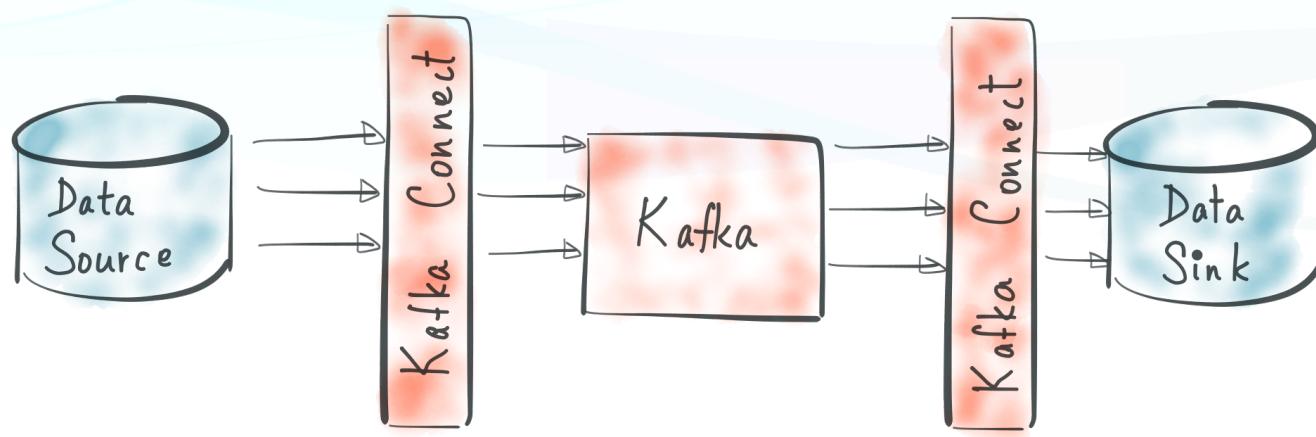


KAFKA STREAMS





KAFKA CONNECT



KAFKA CONNECT DOES THE HARD WORK

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

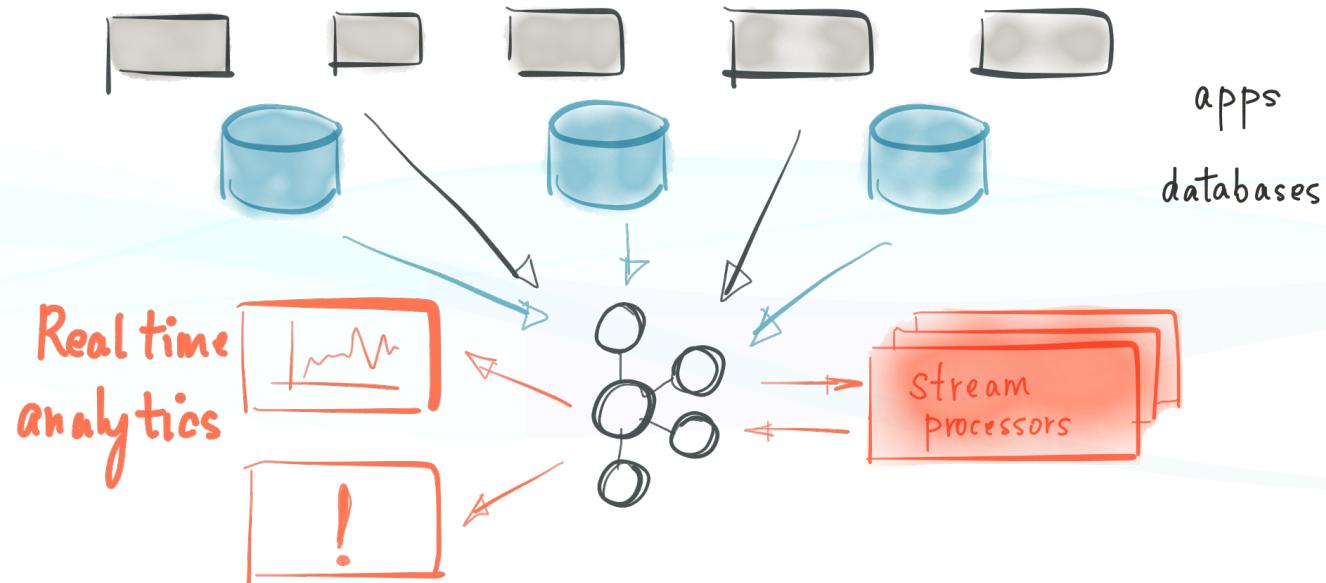
TIMELINE

- Kafka Connect
 - Kafka Streams

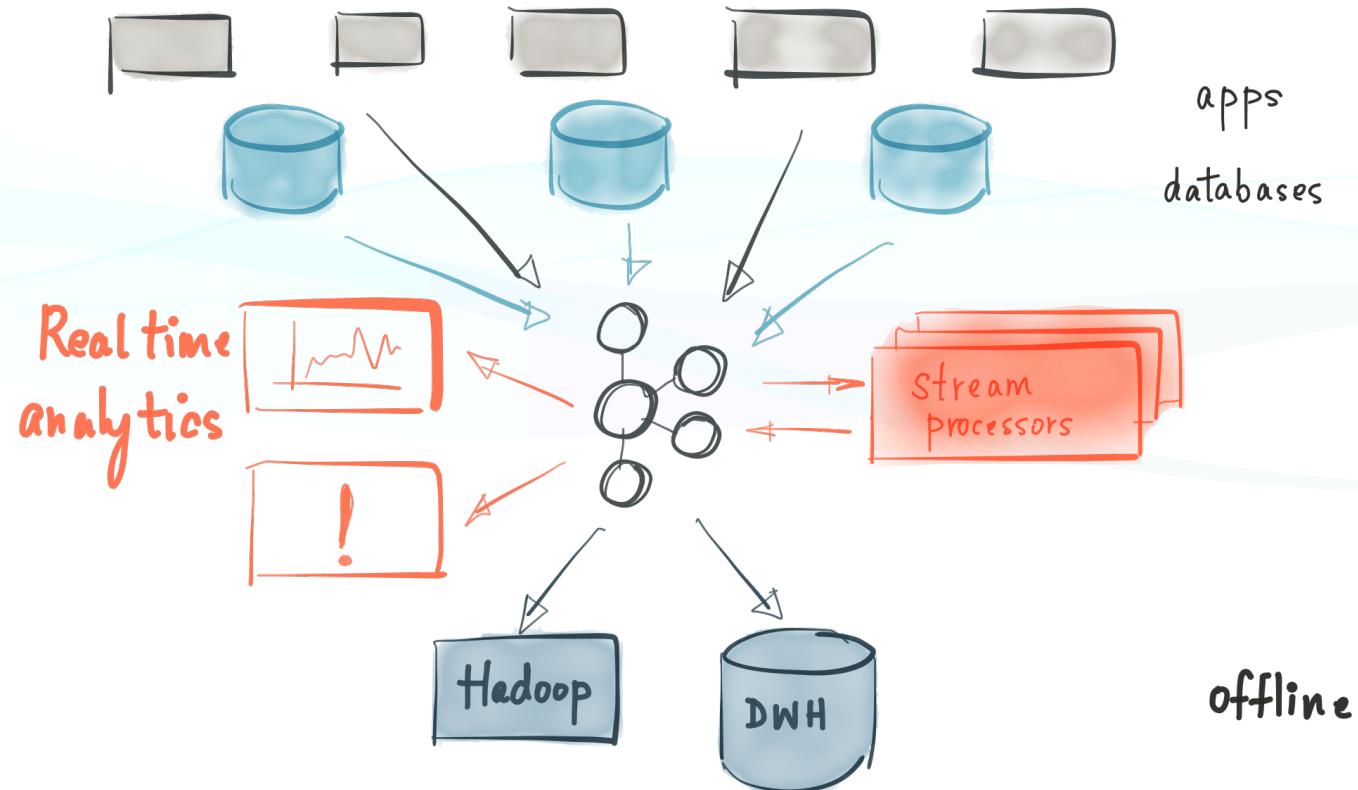
NEXT STEPS

- Expanded DSL
- Exactly-once

STREAM DATA PLATFORM



STREAM DATA PLATFORM



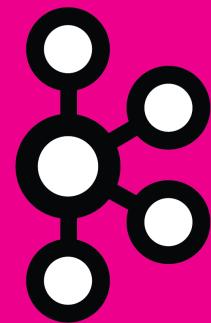
CONFLUENT
PLATFORM = KAFKA++



CONFLUENT PLATFORM

- Schemas
- Clients
- REST
- Connectors
- Management tools

...



kafka summit

**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



www.confluent.io/download

