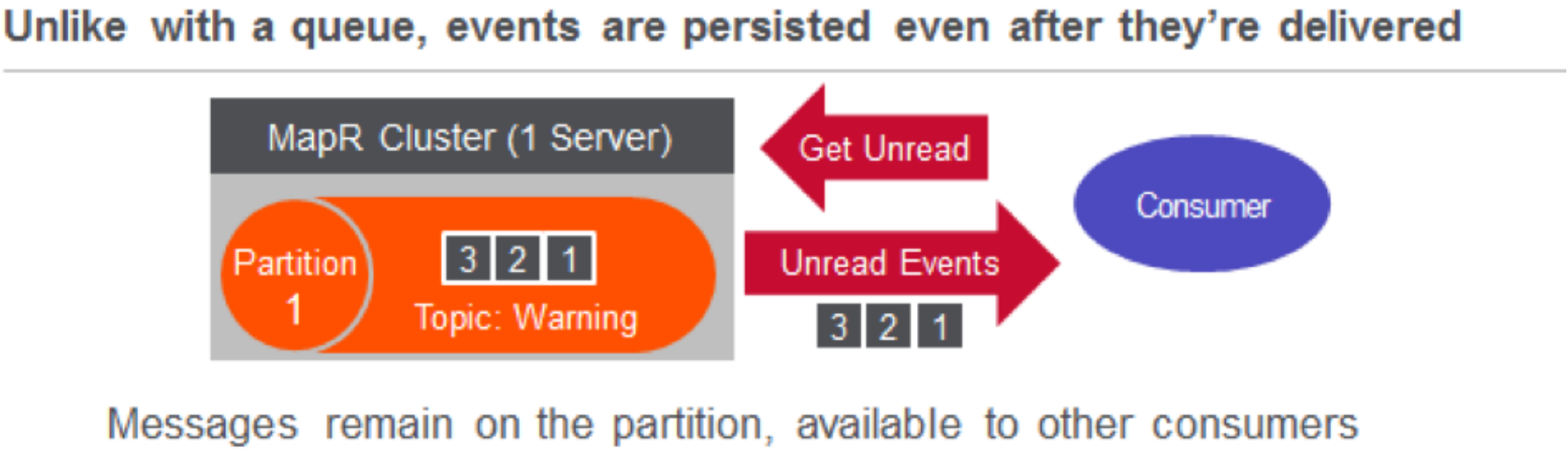
Event driven architecture & Streams

Streams

- Data is not retrieved from a fixed storage device, but rather, handled in near-real-time, with users or other systems rapidly injecting information into our system.
- data injection is asynchronous, where **we do not know ahead of time** when the data will be present.
- In order to facilitate this asynchronous style of data handling, **we have to rethink older polling-based models and instead, use a lighter, more streamlined method.**

Events (message) Stream

- Events are grouped into logical collections of events (often as Topics). Topics can be partitioned for parallel processing. You can think of a partitioned Topic like a queue, **events are delivered in the order they are received.**



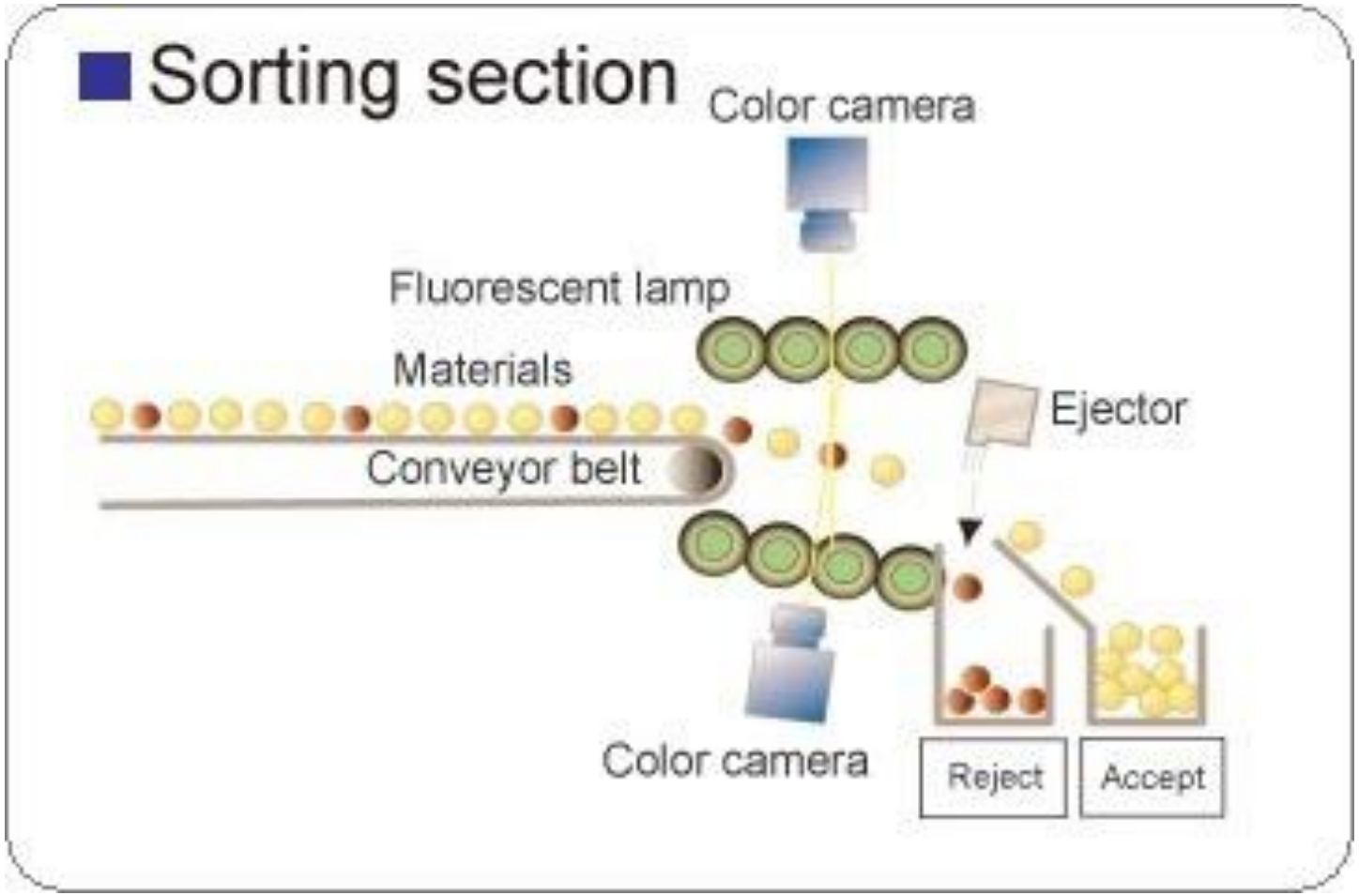
Event Driven Microservices Patterns
https://dzone.com/articles/event-driven-microservices-patterns

Events (message) Stream

- Support multiple consumers
  - topics can have multiple different consumers,
  - remain on the partition, **available to other consumers**.
  - processing of the same messages by different consumers for different purposes.
- Persist events
  - Events are **persisted**, even after they are delivered they
  - Events are **not deleted from Topics when read**
- Can be used to **cache datasets**
  - Older messages are automatically deleted based on the Stream's **time-to-live setting**.
- **Pipelining** is also possible where a consumer enriches an event and publishes it to another topic.

Event Driven Microservices Patterns
https://dzone.com/articles/event-driven-microservices-patterns

Stream processing (intuition)



https://www.quora.com/What-is-streaming-data

Event driven architecture models

Pub/sub

- The messaging infrastructure keeps track of subscriptions. When an event is published, it sends the event to each subscriber. After an event is received, it cannot be replayed, and new subscribers do not see the event.

Event streaming

- Events are written to a log. Events are strictly ordered (within a partition) and durable. Clients don't subscribe to the stream, instead a client can read from any part of the stream. The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

Event-driven architecture style
https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven

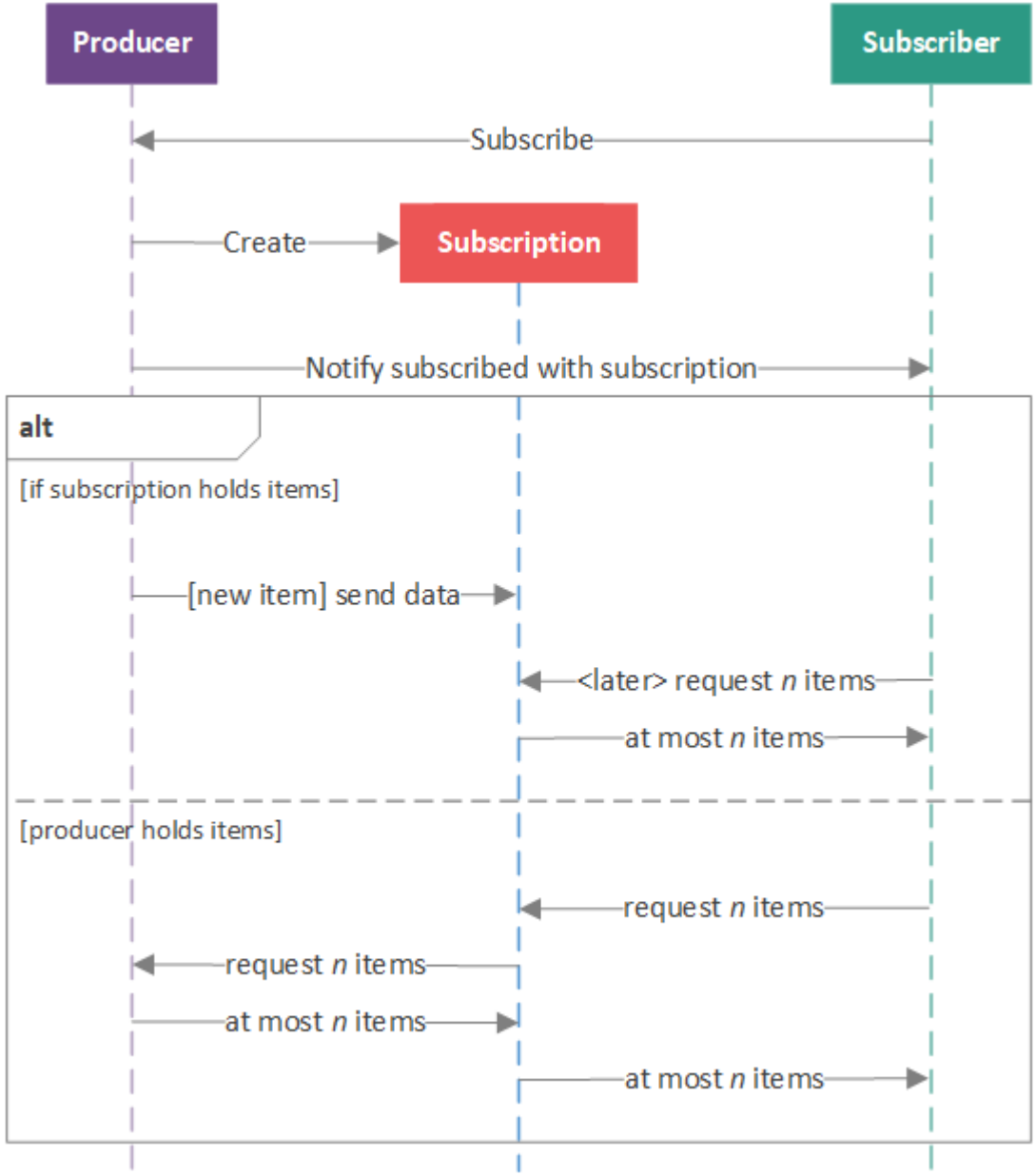
Publishers, Subscribers, and Subscriptions

publisher

- if **items are available**, the publisher pushes the maximum receivable number of items to the subscriber.
- the publisher **pushes** that number of items to the subscriber.

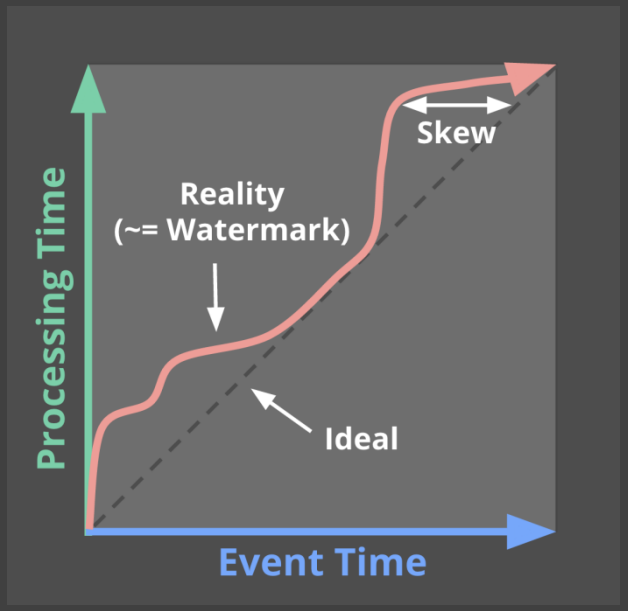
subscriber

- A **subscriber informs** a **publisher** that it is willing to accept a given number of **items** (**requests** a given number of items)
- the subscriber informs the publisher how many items it is willing to handle



Reactive Streams in Java 9
https://dzone.com/articles/reactive-streams-in-java-9

Jumping over “foundations”, hands on



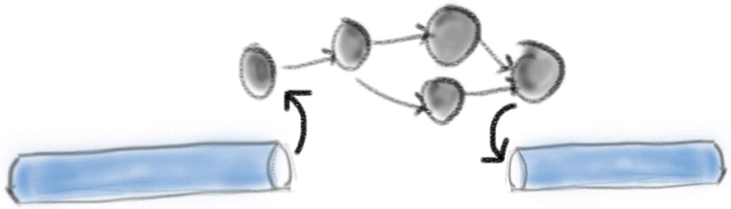
What?	Sums of integers, keyed by team.
Where?	Within fixed event-time windows of one hour.
When?	• Early: Every 5 minutes of processing time. • On-time: When the watermark passes the end of the window. • Late: Every 10 minutes of processing time. • Final: When the watermark passes the end of the window + two hours.
How?	Panes accumulate new values into prior results.

```
gameEvents
  [... input ...]
  .apply("LeaderboardTeamFixedWindows", Window
    .<GameActionInfo>into(FixedWindows.of(
      Duration.standardMinutes(Durations.minutes(60))))
    .triggering(AfterWatermark.pastEndOfWindow()
      .withEarlyFirings(AfterProcessingTime.pastFirstElementInPane()
        .plusDelayOf(Durations.minutes(5)))
      .withLateFirings(AfterProcessingTime.pastFirstElementInPane()
        .plusDelayOf(Durations.minutes(10))))
    .withAllowedLateness(Duration.standardMinutes(120)
    .accumulatingFiredPanes())
  .apply("ExtractTeamScore", new ExtractAndSumScore("team"))
  [... output ...]
```

The world beyond batch: Streaming 101
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
The world beyond batch: Streaming 102
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

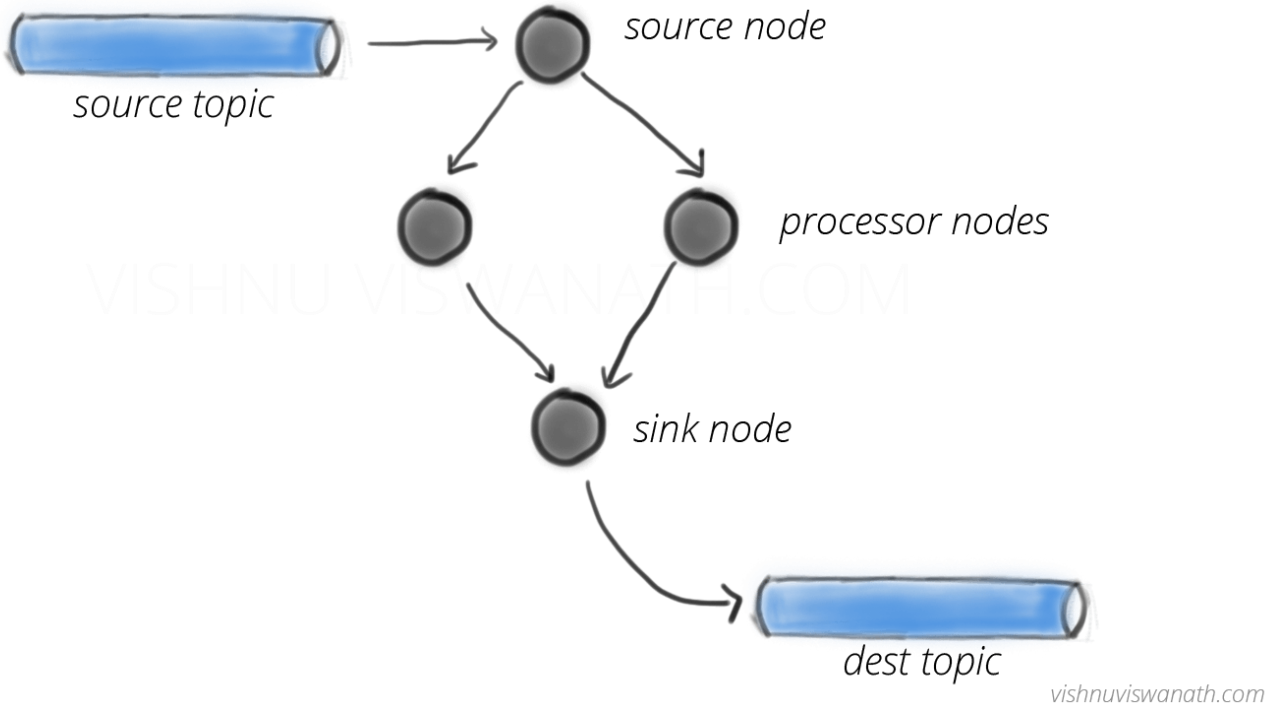
Kafka Streams: an event streaming solution

Kafka Streams

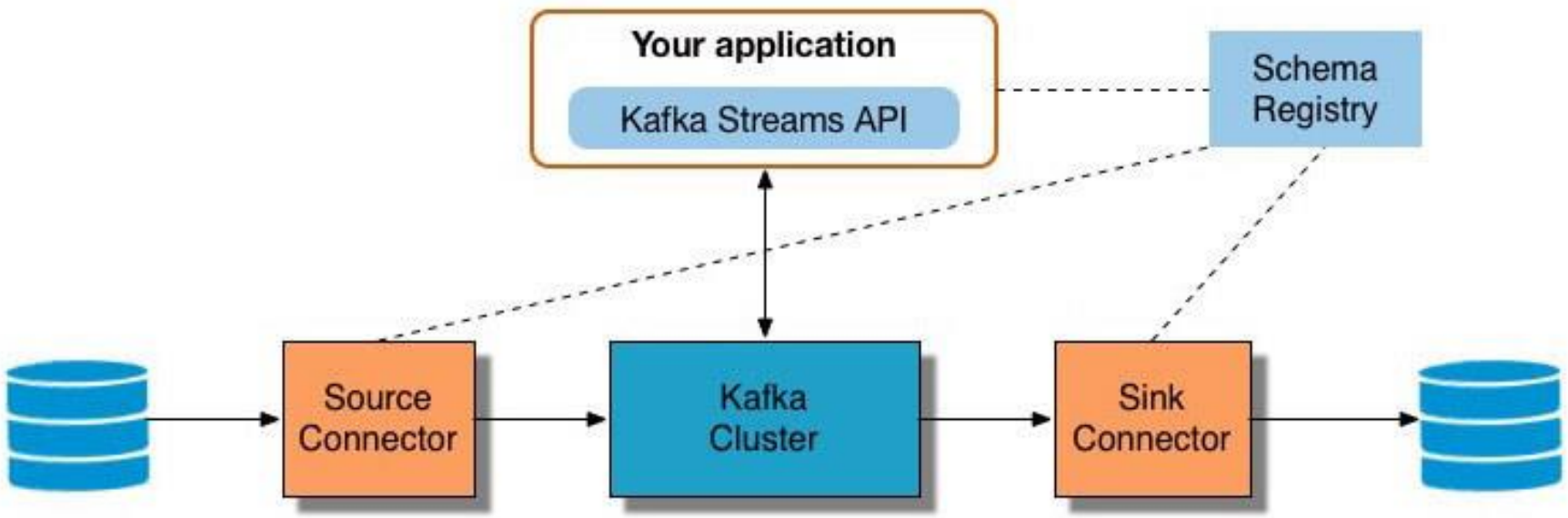


vishnuviswanath.com

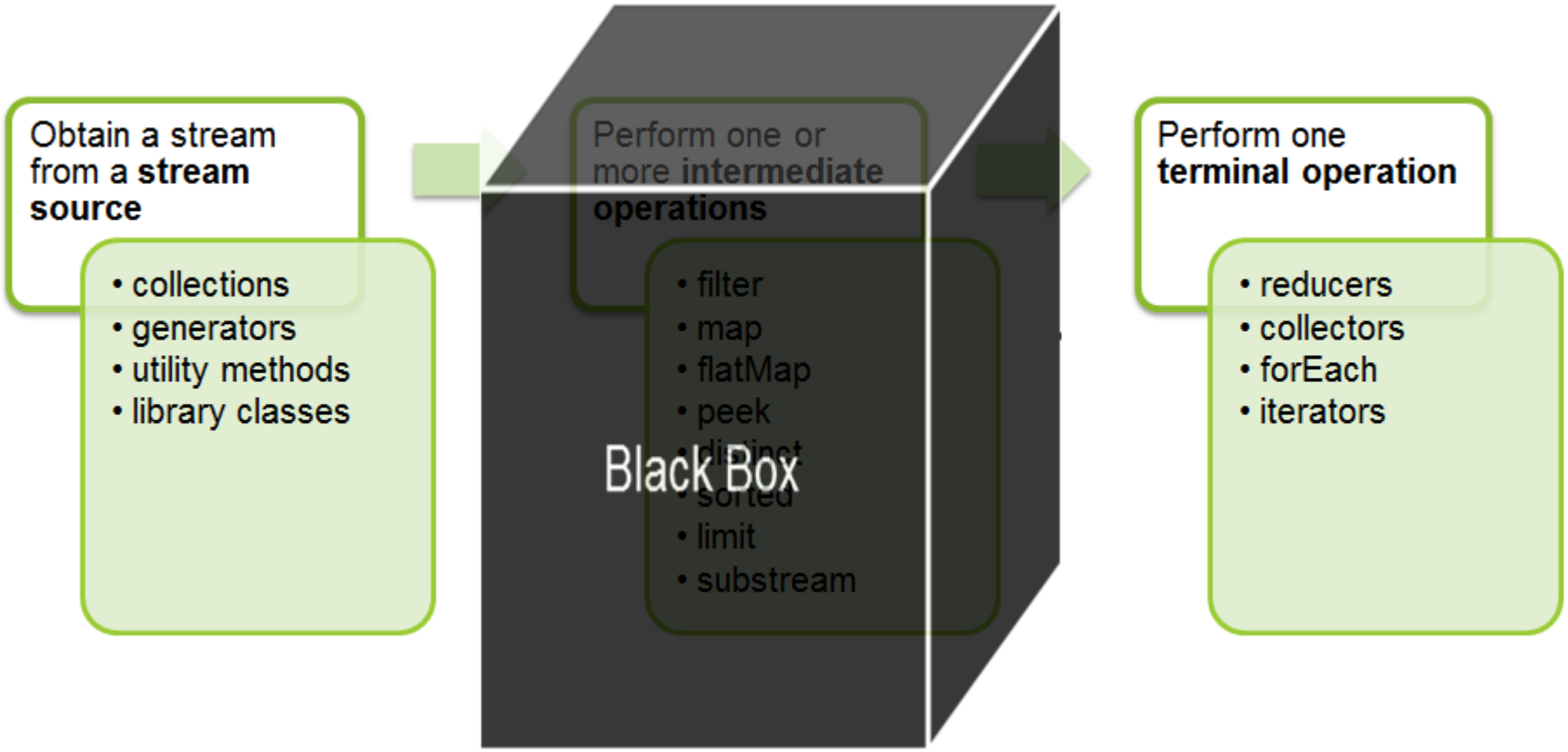
Kafka Streams: A newcomer



http://vishnuviswanath.com/hello-kafka-streams.html



Building a Real-Time Streaming ETL Pipeline in 20 Minutes
https://www.confluent.io/blog/building-real-time-streaming-etl-pipeline-20-minutes/



https://stackoverflow.com/questions/22942223/collecting-stream-back-into-the-same-collection-type

Executive Summary

Since co-authoring the Reactive Manifesto in 2013, we've seen the topic of Reactive go from being a virtually unacknowledged technique for constructing applications—used by only

Reactive Programming versus Reactive Systems
https://www.lightbend.com/reactive-programming-versus-reactive-systems

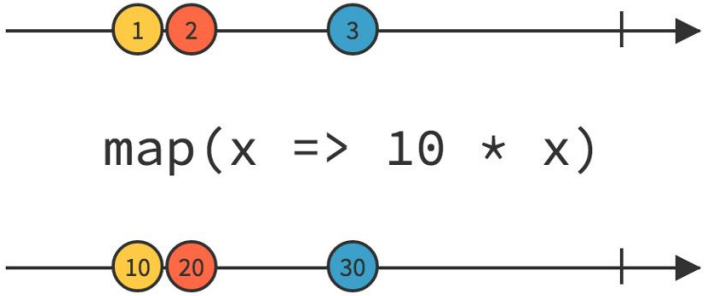
Reactive Programming versus Reactive Systems
https://www.lightbend.com/reactive-programming-versus-reactive-systems

Reactive programming

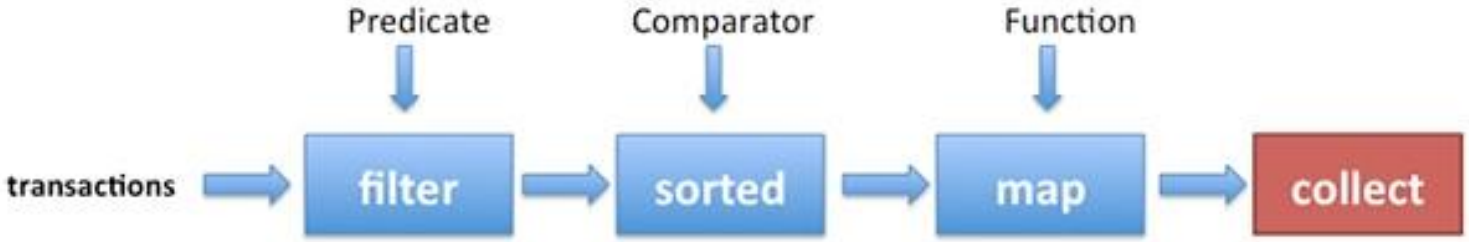
- is not functional or streams
  - Finite observable vs Unbounded stream
  - Can do streams without reactive ...
  - Application scope vs system
- But
  - nice explanation in reactive programming can help
  - Functional paradigm is natural
- “Equivalent” Concepts
  - Pipelines , flow of of data
  - Function / processor
  - Source , sink or producer / consumer or observable / consumer

Functional to the rescue

- Loose coupling
  - Functions are “context free”
  - No state
  - Ideally no side effects
  - Can be moved
- Predictable & Scalable
  - load is not dependent on function
  - Runtime, Data
  - Ideally just increase resources
- asynchronous
  - Event driven
  - Stream abstraction
  - Data flow



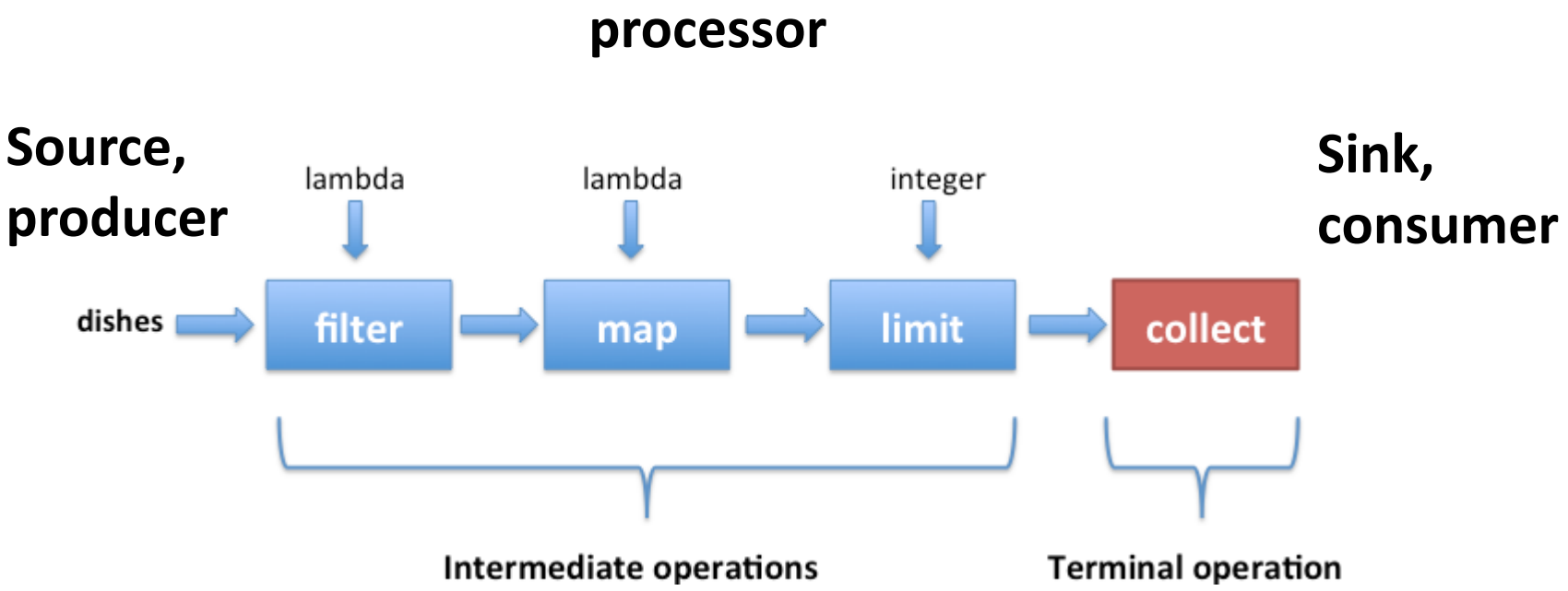
java streams are monads




```
List<Integer> transactionsIds =
    transactions.stream()
                .filter(t -> t.getType() == Transaction.GROCERY)
                .sorted(comparing(Transaction::getValue).reversed())
                .map(Transaction::getId)
                .collect(toList());
```

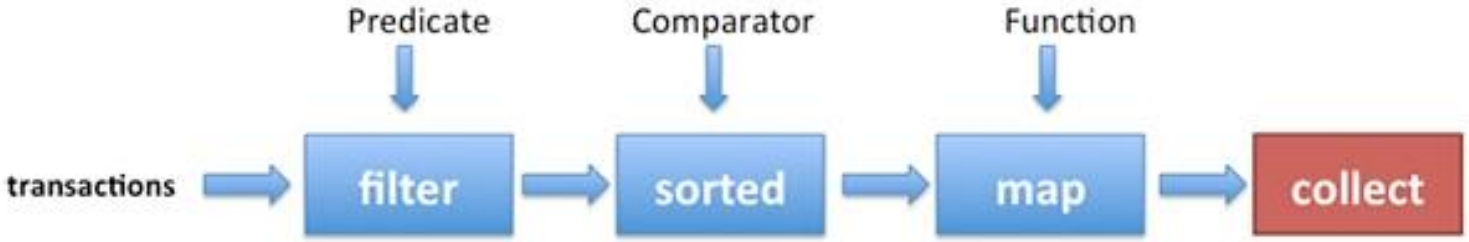
In functional programming, a monad is a structure that represents computations defined as sequences of steps. A type with a monad structure defines what it means to chain operations, or nest functions of that type together.

Processing Data with Java SE 8 Streams, Part 1
http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html



Processing Data with Java SE 8 Streams, Part 1
http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

java streams are monads




```
List<Integer> transactionsIds =
    transactions.stream()
                .filter(t -> t.getType() == Transaction.GROCERY)
                .sorted(comparing(Transaction::getValue).reversed())
                .map(Transaction::getId)
                .collect(toList());
```

In functional programming, a monad is a stru
computations defined as sequences
structure defines what it means to ch
functions of that type together.

You will see chaining in "messaging" streams examples in java

Processing Data with Java SE 8 Streams, Part 1
http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

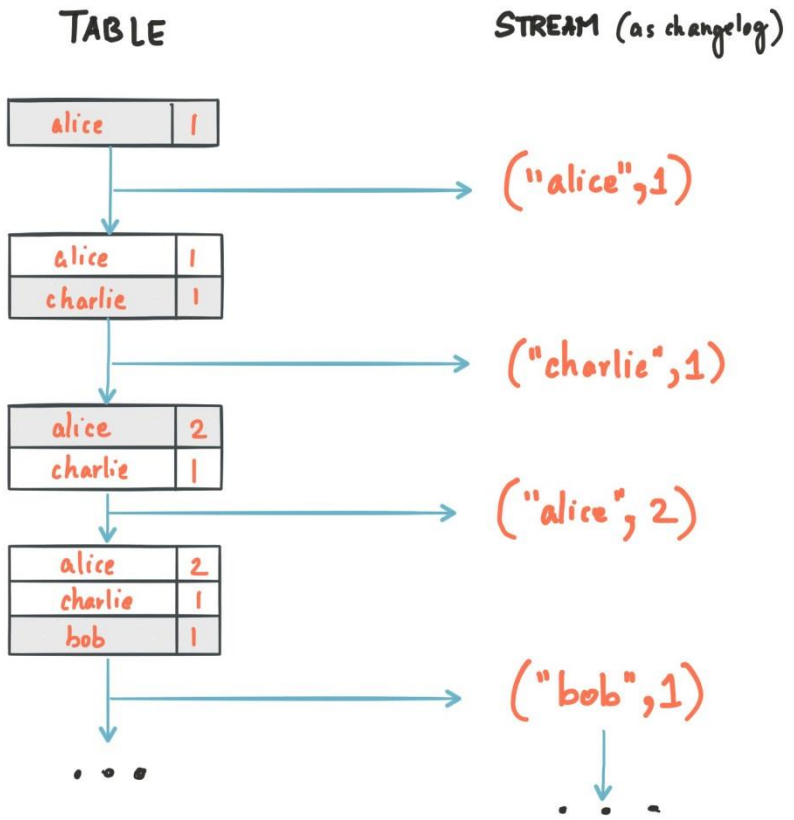
What is a stream?

- is a sequence of records.
- Kafka models a stream as a log, that is, a never-ending sequence of key/value pairs
- often called a changelog

Introducing Kafka Streams: Stream Processing Made Simple
https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/
https://docs.confluent.io/current/streams/concepts.html#kafka-101

Tables and streams

- a pure stream is one where the changes are interpreted as just INSERT statements (since no record replaces any existing record), whereas a table is a stream where the changes are interpreted as UPDATEs (since any existing row with the same key is overwritten).



Creating source streams from Kafka

To stream **To ktable**

```
StreamsBuilder builder = new StreamsBuilder();

KStream<String, Long> wordCounts = builder.stream(
    "word-counts-input-topic", /* input topic */
    Consumed.with(
      Serdes.String(), /* key serde */
      Serdes.Long()   /* value serde */
    );
```

https://docs.confluent.io/current/streams/developer-guide/dsl-api.html

How does this relate to streams & tables?

Tables capture a point-in-time snapshot of a time-varying relation.

Streams capture the evolution of a time-varying relation over time.

Foundations of Streaming SQL or: How I learned to love stream & table theory [Strata NYC 2017]
https://www.youtube.com/watch?v=YO95iAkwapQ
https://www.infoq.com/presentations/beam-model-stream-table=theory

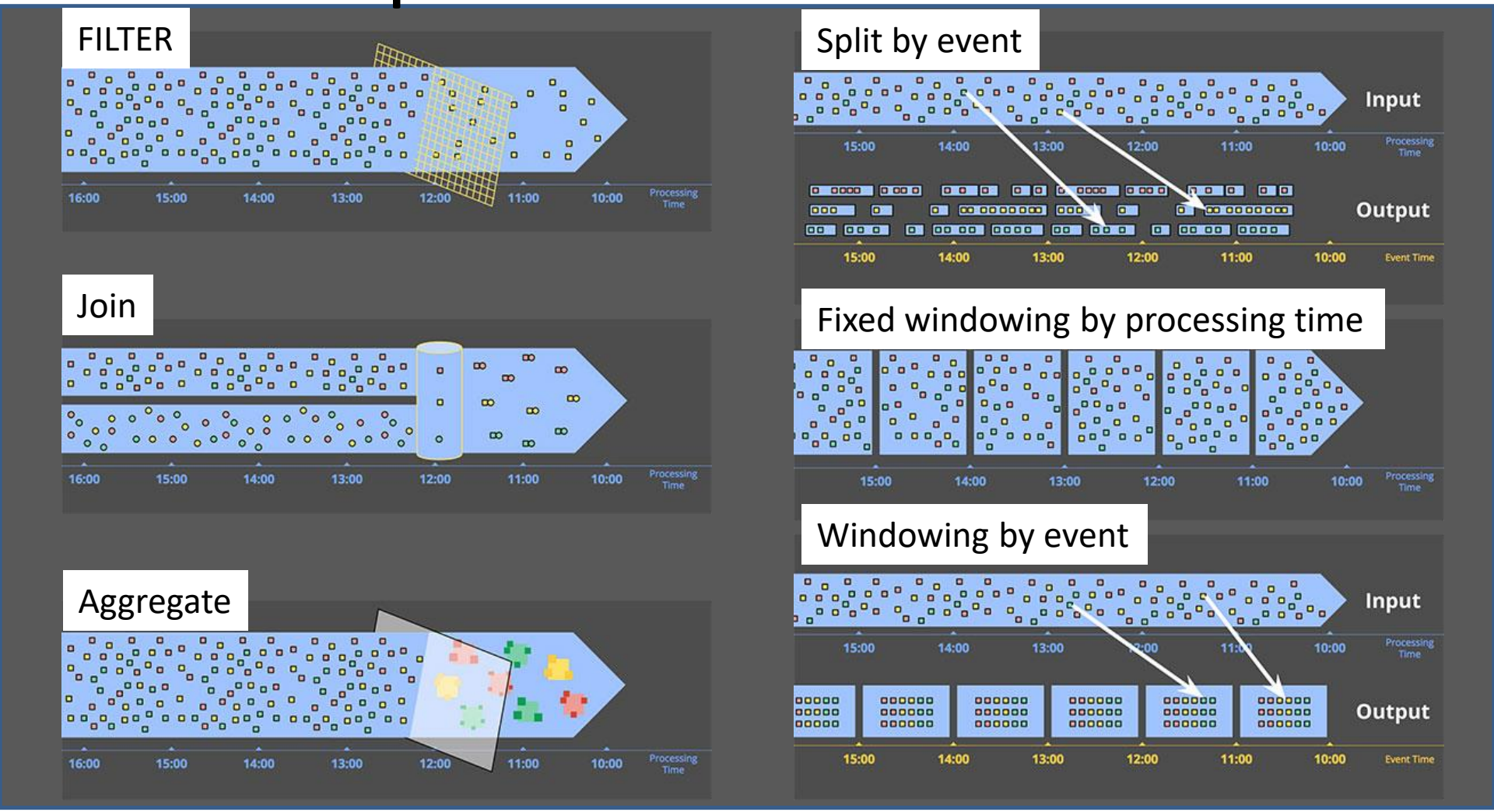
General theory of stream & table relativity

Pipelines : tables + streams + **operations**
Tables : data at rest
Streams : data in motion
Operations : (stream | table) → (stream | table) transformations

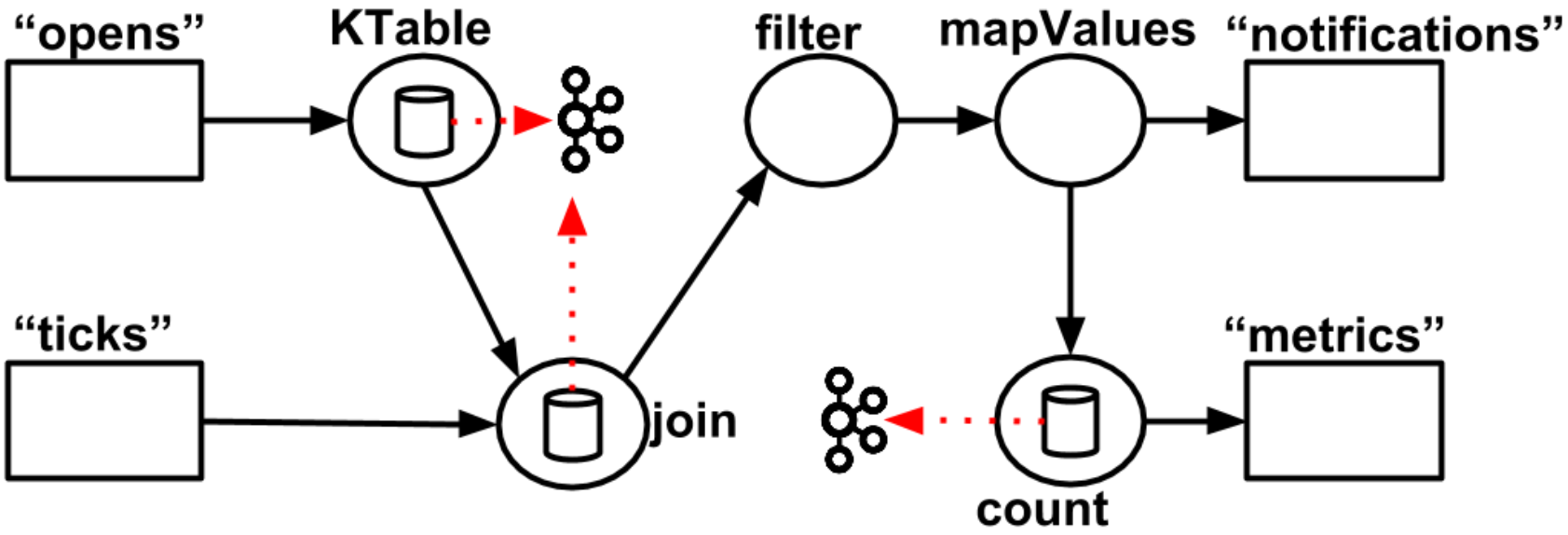
- stream → stream: Non-grouping (element-wise) operations
  Leaves stream data in motion, yielding another stream.
- stream → table: Grouping operations
  Brings stream data to rest, yielding a table.
  Windowing adds the dimension of time to grouping.
- table → stream: Ungrouping (triggering) operations
  Puts table data into motion, yielding a stream.
  Accumulation dictates the nature of the stream (deltas, values, retractions).
- table → table: (none)
  Impossible to go from rest and back to rest without being put into motion.

Foundations of Streaming SQL or: How I learned to love stream & table theory [Strata NYC 2017]
https://www.youtube.com/watch?v=YO95iAkwapQ
https://www.infoq.com/presentations/beam-model-stream-table=theory

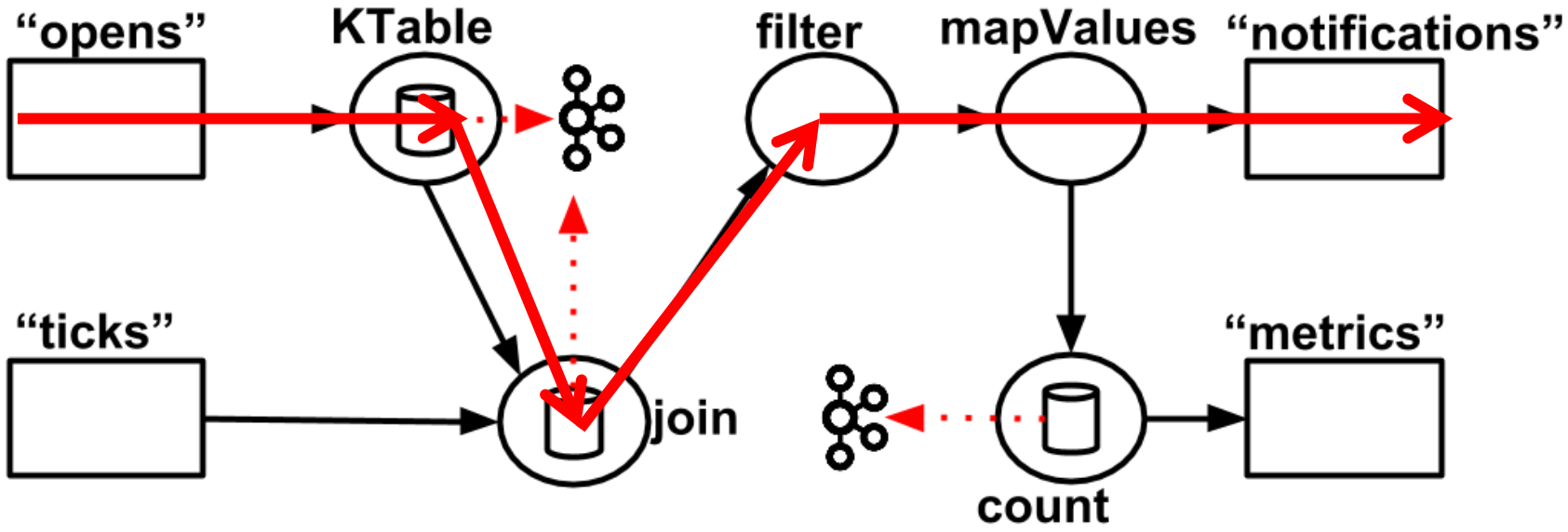
Operations on streams



The world beyond batch: Streaming 101
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
The world beyond batch: Streaming 102
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

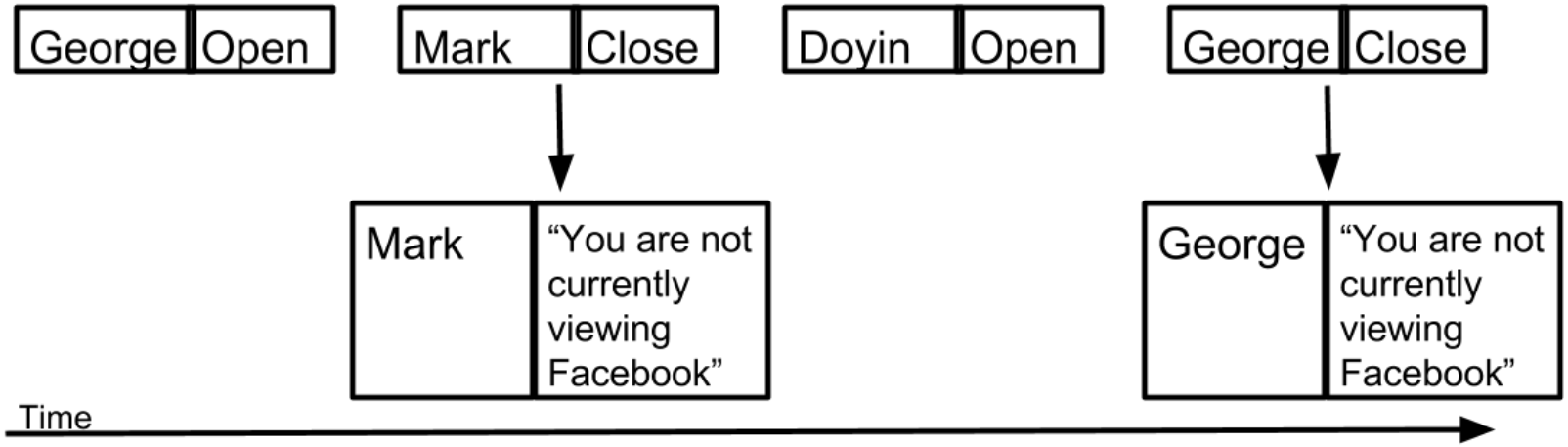


Kafka Streams - Not Looking at Facebook
https://timothyrenner.github.io/engineering/2016/08/11/kafka-streams-not-looking-at-facebook.html




```
// Stream from a Kafka topic.
builder.stream("open")

        // Remove "Open" events.
        .filter((k,v) -> v == "Close")

        // Add the helpful notification message.
        .mapValues(v -> "You are not currently viewing Facebook.")

        // Sink to a Kafka topic.
        .to("notifications");
```

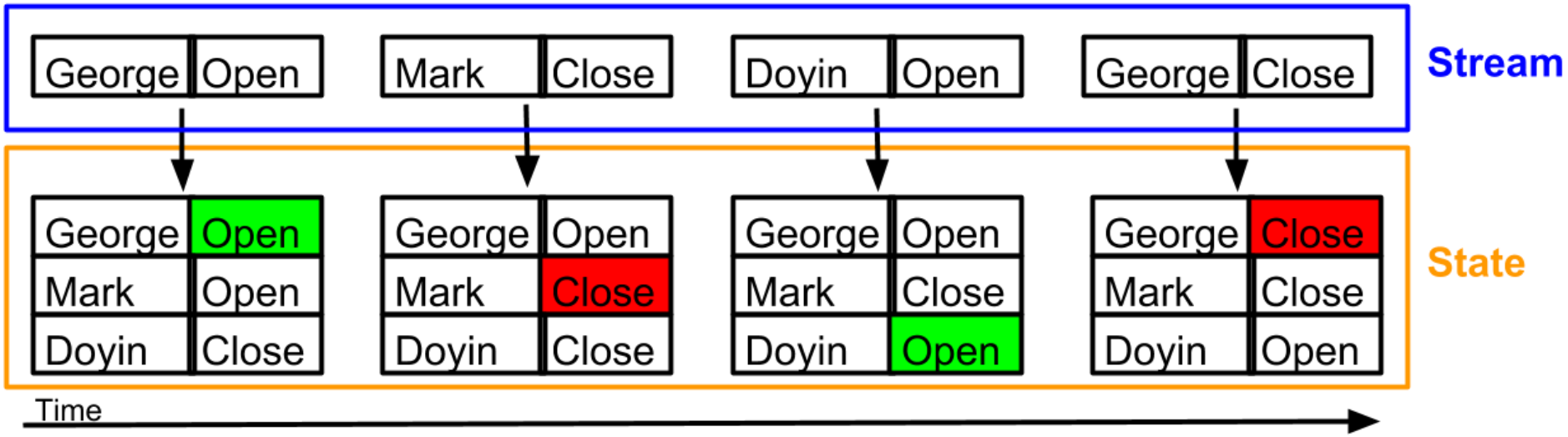
KStream




```
// Stream from a Kafka topic.
builder.stream("open")

       // Remove "Open" events.
       .filter((k,v) -> v == "Close")

       // Add the helpful notification message.
       .mapValues(v -> "You are not currently viewing Facebook.")

       // Sink to a Kafka topic.
       .to("notifications");
```

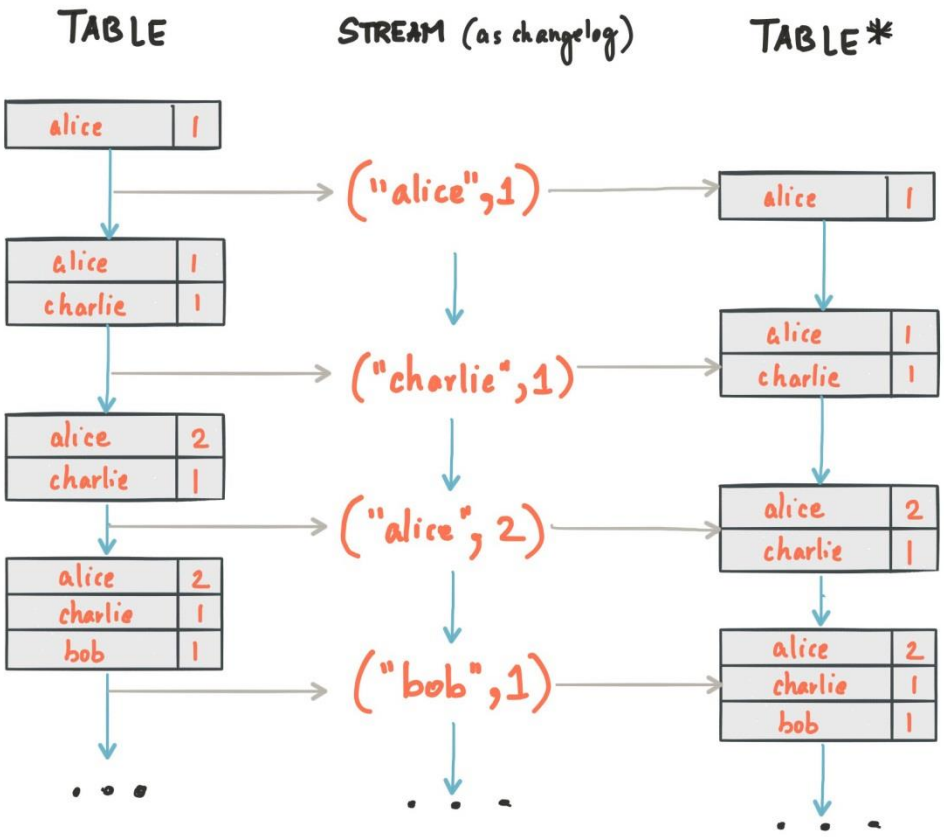
Kafka Streams - Not Looking at Facebook
https://timothyrenner.github.io/engineering/2016/08/11/kafka-streams-not-looking-at-facebook.html

ktable



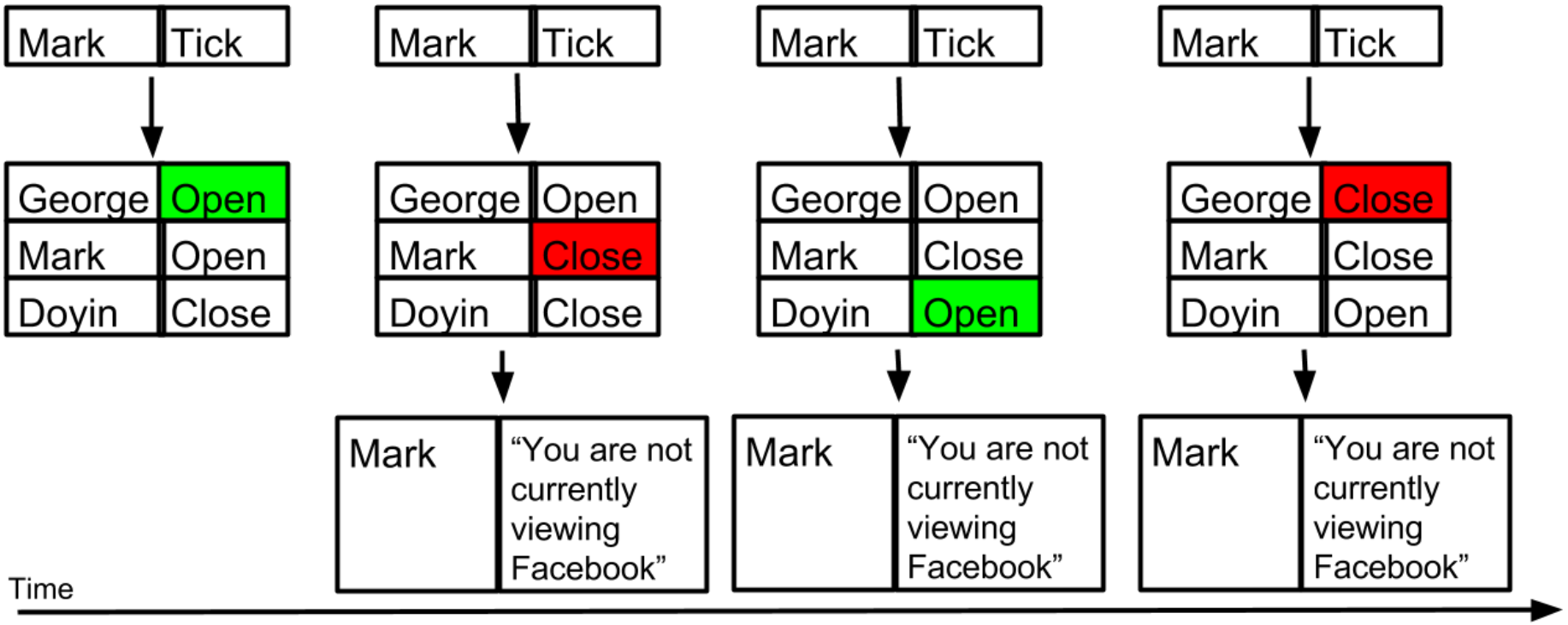
Kafka Streams - Not Looking at Facebook
https://timothyrenner.github.io/engineering/2016/08/11/kafka-streams-not-looking-at-facebook.html

Tables and Streams are Dual

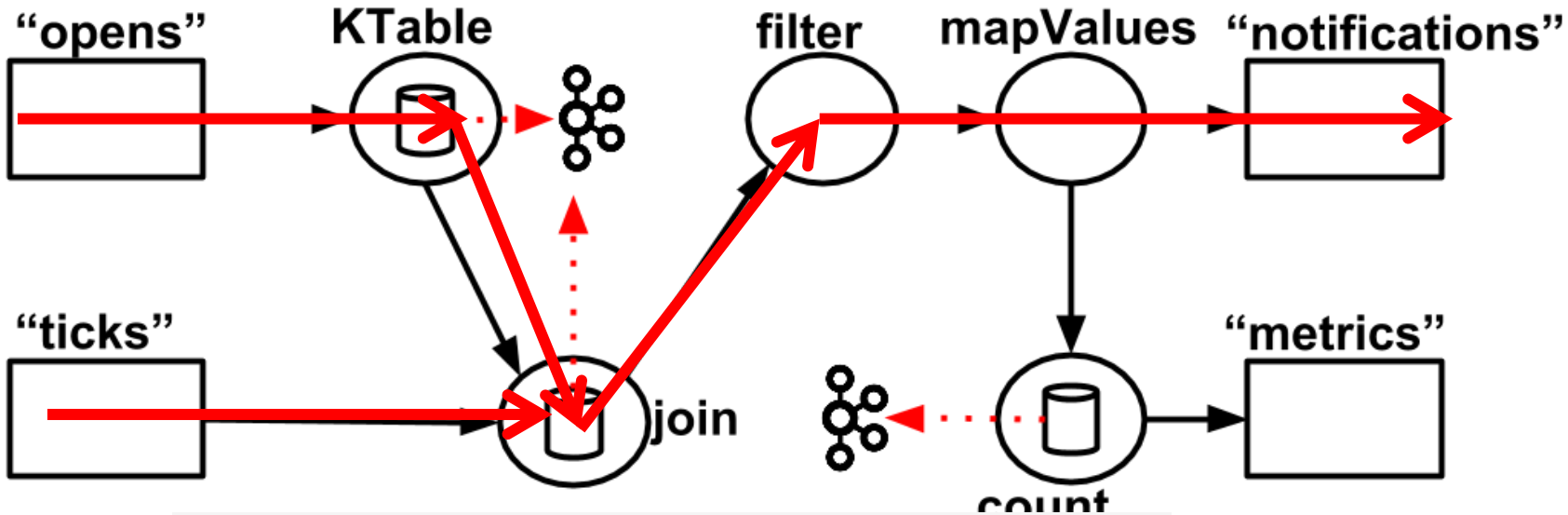


Introducing Kafka Streams: Stream Processing Made Simple
https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/

Ktable in the middle of streams

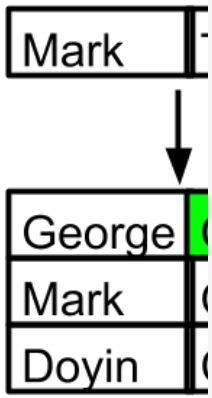


Kafka Streams - Not Looking at Facebook
https://timothyrenner.github.io/engineering/2016/08/11/kafka-streams-not-looking-at-facebook.html



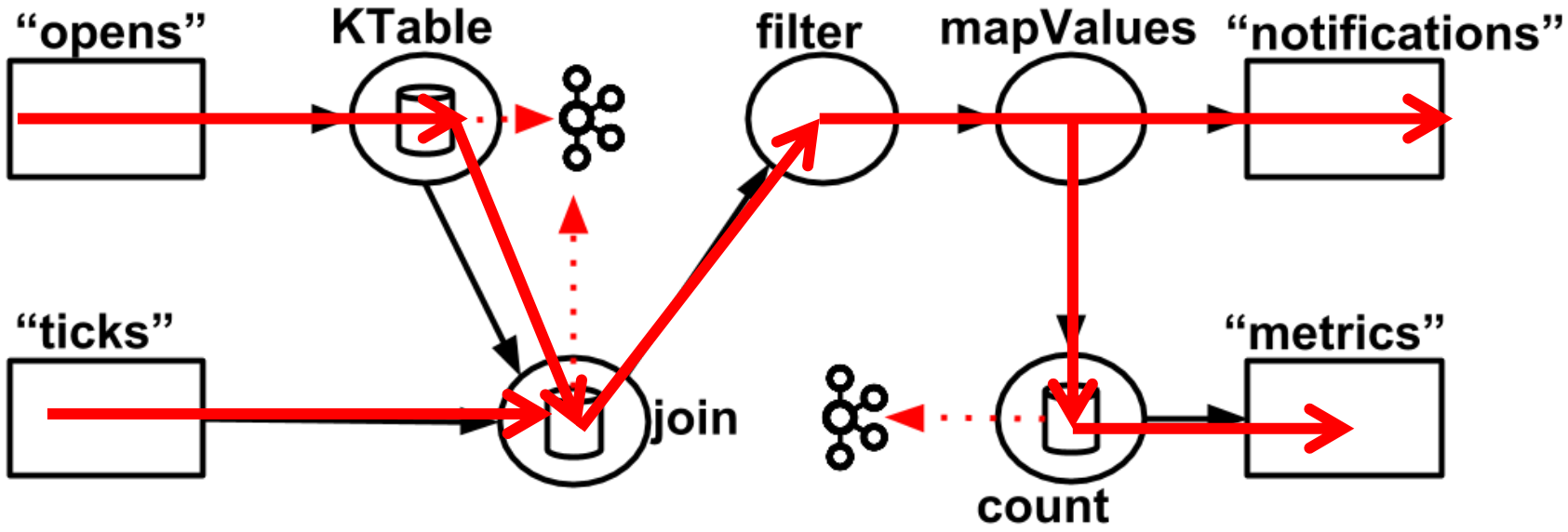

```
KStream<String, String> ticks = builder.stream("tick");

// Now we stream "open" as a table.
KTable<String, String>  open = builder.table("open");

// Join ticks to the open table.
KStream<String, String> closedTicks =
    ticks.leftJoin(open, (vtick, vopen) -> vopen)

         // Filter the "closed" entries in the new stream.
         .filter((k,v) -> v == "Closed");

        // Now fill in the message.
        .mapValues(v -> "You are not currently viewing Facebook.");

// Sink the messages to the "notifications" topic.
closedTicks.to("notifications");
```

Ktable in the middle of streams




```
KStream<String, String> ticks = builder.stream("tick");

// Now we stream "open" as a table.
KTable<String, String>  open = builder.table("open");

// Join ticks to the open table.
KStream<String, String> closedTicks =
    ticks.leftJoin(open, (vtick, vopen) -> vopen)

         // Filter the "closed" entries in the new stream.
         .filter((k,v) -> v == "Closed");

        // Now fill in the message.
        .mapValues(v -> "You are not currently viewing Facebook.");

// Sink the messages to the "notifications" topic.
closedTicks.to("notifications");
```

Kafka Streams - Not Looking at Facebook
https://timothyrenner.github.io/engineering/2016/08/11/kafka-streams-not-looking-at-facebook.html




```
KTable<Windowed<String>, Long> notificationCounts =
    closedTicks.countByKey(
        TimeWindows.of("notificationCounts", 60000L * 10)
                   // "Hop" the windows every minute.
                   .advanceBy(60000L)
                   // Ignore late values.
                   .until(60000L * 10));

// Convert the table updates to a stream.
// Note we need to extract the key from the window.
notificationCounts.toStream((k,v) -> k.key())
                  .to("metrics");
```

Windows

```
KTable<Windowed<String>, Long> notificationCounts =
    closedTicks.countByKey(
        TimeWindows.of("notificationCounts", 60000L * 10)
                   // "Hop" the windows every minute.
                   .advanceBy(60000L)
                   // Ignore late values.
                   .until(60000L * 10));

// Convert the table updates to a stream.
// Note we need to extract the key from the window.
notificationCounts.toStream((k,v) -> k.key())
                  .to("metrics");
```

Basic example

```
(...)
public class Stream {
 public static void main(final String[] args) throws Exception {
  final Properties streamsConfiguration = new Properties();
  streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "messageStream");
  streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
  streamsConfiguration.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());

  final Serde<String> stringSerde = Serdes.String();
  final KStreamBuilder builder = new KStreamBuilder();
  final KStream<String, String> incomingMessageStream = builder.stream(stringSerde, stringSerde, "messages");
  final KStream<String, String> processedStream = incomingMessageStream.mapValues(message -> message + " -
Processed");

  processedStream.to(stringSerde, stringSerde, "messagesProcessed");

  final KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);
  streams.start();
  Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
 }
}
```

How to process streams with Kafka Streams?
https://medium.com/@yagizdemirsoy/how-to-process-streams-with-kafka-streams-e1dfefe20466

(des)serialization

```
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();

KStreamBuilder builder = new KStreamBuilder();
KStream<String, String> textLines = builder.stream(stringSerde, stringSerde, "TextLinesTopic");
KStream<String, Long> wordCounts = textLines
      .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
      .map((key, word) -> new KeyValue<>(word, word))
      // Required in Kafka 0.10.0 to re-partition the data because we re-keyed the stream in the `map` step.
      // Upcoming Kafka 0.10.1 does this automatically for you (no need for `through`).
      .through("RekeyedIntermediateTopic")
      .countByKey("Counts")
      .toStream();
wordCounts.to(stringSerde, longSerde, "WordsWithCountsTopic");

KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
```

- Kafka is content agnostic so it needs:
- deserialization to extract context from message
- serialization to sent to stream

https://gist.github.com/miguno/a9b44c1efc9b579565cfad75e543c808

Streams from/to topics

```
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();

KStreamBuilder builder = new KStreamBuilder();
KStream<String, String> textLines = builder.stream(stringSerde, stringSerde, "TextLinesTopic");
KStream<String, Long> wordCounts = textLines
      .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
      .map((key, word) -> new KeyValue<>(word, word))
      // Required in Kafka 0.10.0 to re-partition the data because we re-keyed the stream in the `map` step.
      // Upcoming Kafka 0.10.1 does this automatically for you (no need for `through`).
      .through("RekeyedIntermediateTopic")
      .countByKey("Counts")
      .toStream();
wordCounts.to(stringSerde, longSerde, "WordsWithCountsTopic");

KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
```

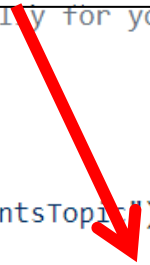
Stream origin are streams and generate streams – in kafka these are placed in topics

https://gist.github.com/miguno/a9b44c1efc9b579565cfad75e543c808

Configurations , start and stop

```
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();

KStreamBuilder b
KStream<String,
KStream<String,
      .flatMapVa
      .map((key,
      // Require                                                          p.
      // Upcoming Kafka 0.10.1 does this automatically for you (no need for  through ).
      .through("RekeyedIntermediateTopic")
      .countByKey("Counts")
      .toStream();
wordCounts.to(stringSerde, longSerde, "WordsWithCountsTopic");

KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
```

```
Properties streamsConfiguration = new Properties();
streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-lambda-example");
streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
streamsConfiguration.put(StreamsConfig.ZOOKEEPER_CONNECT_CONFIG, "localhost:2181");
streamsConfiguration.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
streamsConfiguration.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
```



https://gist.github.com/miguno/a9b44c1efc9b579565cfad75e543c808

processor

```java
public class WordCountProcessor implements Processor<String, String> {

  private ProcessorContext context;
  private KeyValueStore<String, Long> kvStore;

  @Override
  @SuppressWarnings("unchecked")
  public void init(ProcessorContext context) {
      // keep the processor context locally because we need it in punctuate() and commit()
      this.context = context;

      // call this processor's punctuate() method every 1000 time units.
      this.context.schedule(1000);

      // retrieve the key-value store named "Counts"
      kvStore = (KeyValueStore) context.getStateStore("Counts");
  }

  @Override
  public void process(String dummy, String line) {
      String[] words = line.toLowerCase().split(" ");

      for (String word : words) {
          Long oldValue = kvStore.get(word);
          if (oldValue == null) {
              kvStore.put(word, 1L);
          } else {
              kvStore.put(word, oldValue + 1L);
          }
      }
  }

  @Override
  public void punctuate(long timestamp) {
      KeyValueIterator<String, Long> iter = this.kvStore.all();
      while (iter.hasNext()) {
          KeyValue<String, Long> entry = iter.next();
          context.forward(entry.key, entry.value.toString());
      }
      iter.close();
      // commit the current processing progress
      context.commit();
  }

  @Override
  public void close() {
      // close the key-value store
      kvStore.close();
  }

}
```

```
@Override
init(ProcessorContext context)
@Override
process(String key, String value)
@Override
punctuate(long timestamp)
@Override
close()
```

punctuate method will be executed periodically - with the frequency we set in **init()** call to context.schedule(1000)
We send the data to the next Processor
with **context.forward(entry.key, entry.value.toString())**.

Kafka Streams for Stream processing
https://balamaci.ro/kafka-streams-for-stream-processing/
https://docs.confluent.io/3.2.0/streams/developer-guide.html#processor-api

State stores

RockDB

```
// Creating a persistent key-value store:
// here, we create a `KeyValueStore<String, Long>` r
import org.apache.kafka.streams.processor.StateStore
import org.apache.kafka.streams.state.Stores;

// Note: The `Stores` factory returns a supplier for
// because that's what you typically need to pass as
StateStoreSupplier countStoreSupplier =
  Stores.create("persistent-counts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .persistent()
    .build();
```

In memory

```
// Creating an in-memory key-value store:
// here, we create a `KeyValueStore<String, Long>` r
import org.apache.kafka.streams.processor.StateStor
import org.apache.kafka.streams.state.Stores;

// Note: The `Stores` factory returns a supplier fo
// because that's what you typically need to pass a
StateStoreSupplier countStoreSupplier =
  Stores.create("inmemory-counts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .inMemory()
    .build();
```

```
import org.apache.kafka.streams.processor.StateStoreSupplier;
import org.apache.kafka.streams.state.Stores;

Map<String, String> changelogConfig = new HashMap();
// override min.insync.replicas
changelogConfig.put("min.insyc.replicas", "1")

StateStoreSupplier countStoreSupplier = Stores.create("Counts")
            .withKeys(Serdes.String())
            .withValues(Serdes.Long())
            .persistent()
            .enableLogging(changelogConfig) // enable changelogging, with custom changelog settings
            .build();
```

Connecting Processors and State Stores

```
TopologyBuilder builder = new TopologyBuilder();

// add the source processor node that takes Kafka topic "source-topic" as input
builder.addSource("Source", "source-topic")

    // add the WordCountProcessor node which takes the source processor as its upstream processor
    .addProcessor("Process", () -> new WordCountProcessor(), "Source")

    // add the count store associated with the WordCountProcessor processor
    .addStateStore(countStoreSupplier, "Process")

    // add the sink processor node that takes Kafka topic "sink-topic" as output
    // and the WordCountProcessor node as its upstream processor
    .addSink("Sink", "sink-topic", "Process");
```

Kafka Streams for Stream processing
https://balamaci.ro/kafka-streams-for-stream-processing/

State store

- DSL API restricts access to a read-only view of the state store (via ReadOnlyKeyValueStore) as opposed to the Processor API using which you can get access to a writable view which allows you to mutate the state store

```
//DSL API

KafkaStreams ks = ...;
ReadOnlyKeyValueStore<String, Double> myStore = ks.store(storeName,
                                                         QueryableStoreTypes.<String, Double>keyValueStore());

//Processor API
ProcessorContext pc = ... ;
KeyValueStore<String, Double> kvStateStore = (KeyValueStore<String, Double>) pc.getStateStore("my-state-store");
kvStateStore.put("my-key",2); //mutate operation
```

State store

```
public class MyProductCountProcessor extends AbstractProcessor<Long, String> {
    private ProcessorContext context;
    private KeyValueStore<Long, Long> kvStore;

    public void init(ProcessorContext context) {
         (…)
        this.kvStore = (KeyValueStore<Long, Long>) context.getStateStore("Counts");
    }

    public void process(Long productId, String browser) {
        Long oldValue = this.kvStore.get(word);
        if (oldValue == null) {                    this.kvStore.put(word, 1L);
        } else {                                   this.kvStore.put(word, oldValue + 1L);        }
    }

    //This method will be called as a way to do something periodically
    public void punctuate(long timestamp) {
        KeyValueIterator<Long, Long> iter = this.kvStore.all();

        while (iter.hasNext()) {
            KeyValue<String, Long> entry = iter.next();
            context.forward(entry.key, entry.value.toString());                    }
        iter.close();
        context.commit();
    }
}
```

Kafka Streams for Stream processing
https://balamaci.ro/kafka-streams-for-stream-processing/

GlobalKTable

A GlobalKTable is created via a KStreamBuilder. For example:

```
builder.globalTable("topic-name", "queryable-store-name");
```

all GlobalKTables are backed by a ReadOnlyKeyValueStore and are therefore queryable via the interactive queries API. For example:

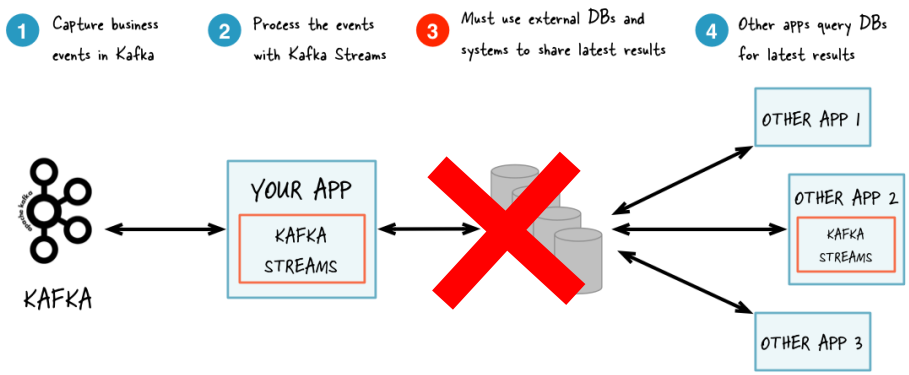
```
final GlobalKTable globalOne = builder.globalTable("g1", "g1-store");
final GlobalKTable globalTwo = builder.globalTable("g2", "g2-store");
…
final KafkaStreams streams = ...;
streams.start()
…
ReadOnlyKeyValueStore view = streams.store("g1-store",
QueryableStoreTypes.keyValueStore());
 view.get(key); // can be done on any key, as all keys are present
```

Kafka Streams for Stream processing
https://balamaci.ro/kafka-streams-for-stream-processing/
https://github.com/confluentinc/kafka-streams-examples/blob/4.0.0-post/src/main/java/io/confluent/examples/streams/GlobalKTablesExample.java

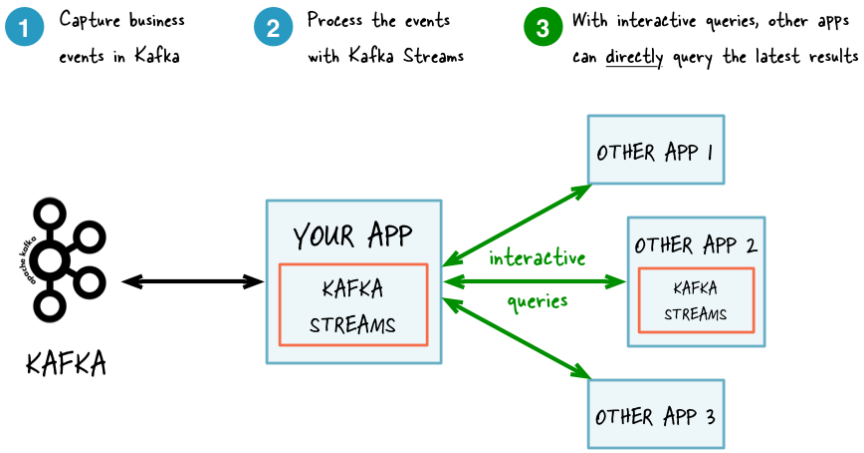
https://kafka.apache.org/0102/javadoc/org/apache/kafka/streams/kstream/GlobalKTable.html

Interactive Queries

no



with



Interactive Queries

```
StreamsConfig config = ...;
KStreamBuilder builder = ...;
KStream<String, String> textLines = ...;

// Define the processing topology (here: WordCount)
KGroupedStream<String, String> groupedByWord = textLines
  .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
  .groupBy((key, word) -> word, stringSerde, stringSerde);

// Create a window state store named "CountsWindowStore" that contains the word counts for every minute
groupedByWord.count(TimeWindows.of(60000), "CountsWindowStore");
```

```
// Get the window store named "CountsWindowStore"
ReadOnlyWindowStore<String, Long> windowStore =
    streams.store("CountsWindowStore", QueryableStoreTypes.windowStore());

// Fetch values for the key "world" for all of the windows available in this application instance
// To get *all* available windows we fetch windows from the beginning of time until now.
long timeFrom = 0; // beginning of time = oldest available
long timeTo = System.currentTimeMillis(); // now (in processing-time)
WindowStoreIterator<Long> iterator = windowStore.fetch("world", timeFrom, timeTo);
while (iterator.hasNext()) {
  KeyValue<Long, Long> next = iterator.next();
  long windowTimestamp = next.key;
  System.out.println("Count of 'world' @ time " + windowTimestamp + " is " + next.value);
}
```

https://kafka.apache.org/10/documentation/streams/developer-guide/interactive-queries.html

REST interface

```
static WordCountInteractiveQueriesRestService startRestProxy(final KafkaStreams streams, final int port)
    throws Exception {
  final HostInfo hostInfo = new HostInfo(DEFAULT_HOST, port);
  final WordCountInteractiveQueriesRestService
      wordCountInteractiveQueriesRestService = new WordCountInteractiveQueriesRestService(streams, hostInfo);
  wordCountInteractiveQueriesRestService.start(port);
  return wordCountInteractiveQueriesRestService;
}

static KafkaStreams createStreams(final Properties streamsConfiguration) {
  final Serde<String> stringSerde = Serdes.String();
  StreamsBuilder builder = new StreamsBuilder();
  KStream<String, String>
      textLines = builder.stream(TEXT_LINES_TOPIC, Consumed.with(Serdes.String(), Serdes.String()));

  final KGroupedStream<String, String> groupedByWord = textLines
      .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
      .groupBy((key, word) -> word, Serialized.with(stringSerde, stringSerde));

  // Create a State Store for with the all time word count
  groupedByWord.count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("word-count")
      .withValueSerde(Serdes.Long()));

  // Create a Windowed State Store that contains the word count for every
  // 1 minute
  groupedByWord.windowedBy(TimeWindows.of(60000))
      .count(Materialized.<String, Long, WindowStore<Bytes, byte[]>>as("windowed-word-count")
          .withValueSerde(Serdes.Long()));

  return new KafkaStreams(builder.build(), streamsConfiguration);
}
```

https://github.com/confluentinc/kafka-streams-examples/tree/4.0.0-post/src/main/java/io/confluent/examples/streams/interactivequeries

REST interface

The example is using JAX-RS over jetty

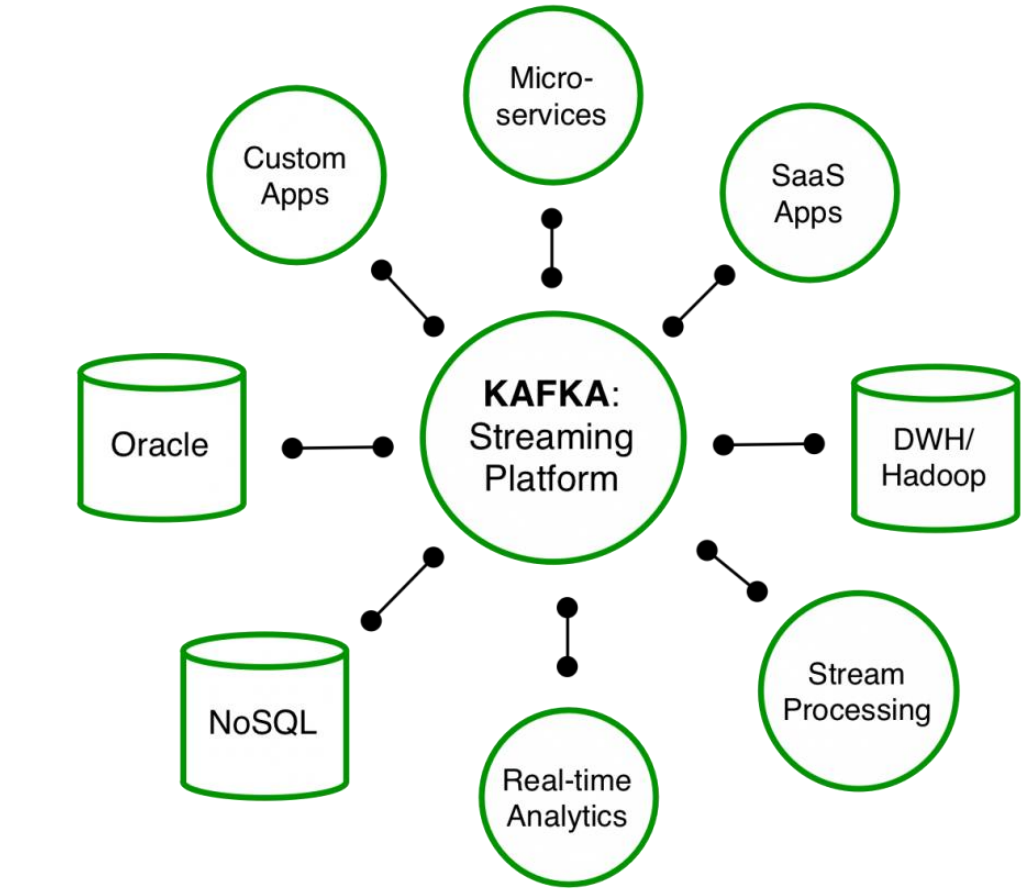
Get store view (queriable interface)

Expose data for REST

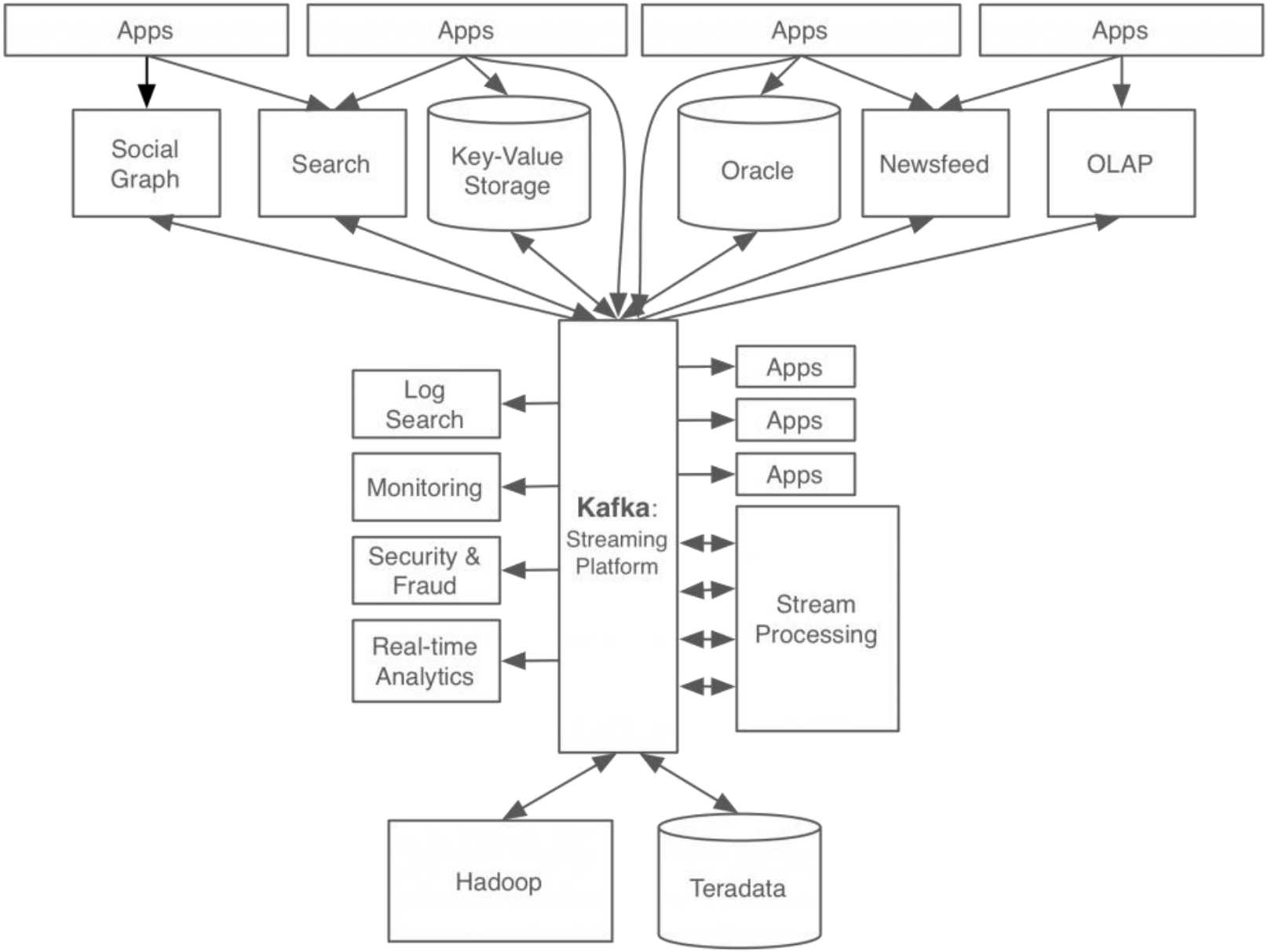
```
static WordCountInteractiveQueriesRestService startRestProxy(final KafkaStreams streams, final int port)
    throws Exception {
  final HostInfo hostInfo = new HostInfo(DEFAULT_HOST, port);
  final WordCountInteractiveQueriesRestService
      wordCountInteractiveQueriesRestService = new WordCountInteractiveQueriesRestService(streams, hostInfo);
  wordCountInteractiveQueries
  return wordCountInteractive
}

static KafkaStreams createStr
  final Serde<String> stringS
  StreamsBuilder builder = ne
  KStream<String, String>
      textLines = builder.str

  final KGroupedStream<String
      .flatMapValues(value ->
      .groupBy((key, word) ->

  // Create a State Store for
  groupedByWord.count(Materia
      .withValueSerde(Serdes.

  // Create a Windowed State
  // 1 minute
  groupedByWord.windowedBy(Ti
      .count(Materialized.<St
          .withValueSerde(Ser

  return new KafkaStreams(builder.build(), streamsConfiguration);
}
```

```
@GET()
@Path("/windowed/{storeName}/{key}/{from}/{to}")
@Produces(MediaType.APPLICATION_JSON)
public List<KeyValueBean> windowedByKey(@PathParam("storeName") final String storeName,
                                        @PathParam("key") final String key,
                                                          Long from,
                                                          ong to) {

  // Lookup the WindowStore with the provided storeName
  final ReadOnlyWindowStore<String, Long> store = streams.store(storeName,
                                                                QueryableStoreTypes.<String, Long>windowStore());
  if (store == null) {
    throw new NotFoundException();
  }

  // fetch the window results for the given key and time range
  final WindowStoreIterator<Long> results = store.fetch(key, from, to);

  final List<KeyValueBean> windowResults = new ArrayList<>();
  while (results.hasNext()) {
    final KeyValue<Long, Long> next = results.next();
    // convert the result to have the window time and the key (for display purposes)
    windowResults.add(new KeyValueBean(key + "@" + next.key, next.value));
  }
  return windowResults;
}
```

https://github.com/confluentinc/kafka-streams-examples/tree/4.0.0-post/src/main/java/io/confluent/examples/streams/interactivequeries



https://www.confluent.io/blog/stream-data-platform-1/



https://www.confluent.io/blog/stream-data-platform-1/

Spring cloud

- “Spring Cloud Stream is a framework built on top of Spring Boot and Spring Integration that helps in creating event-driven or message-driven microservices.”

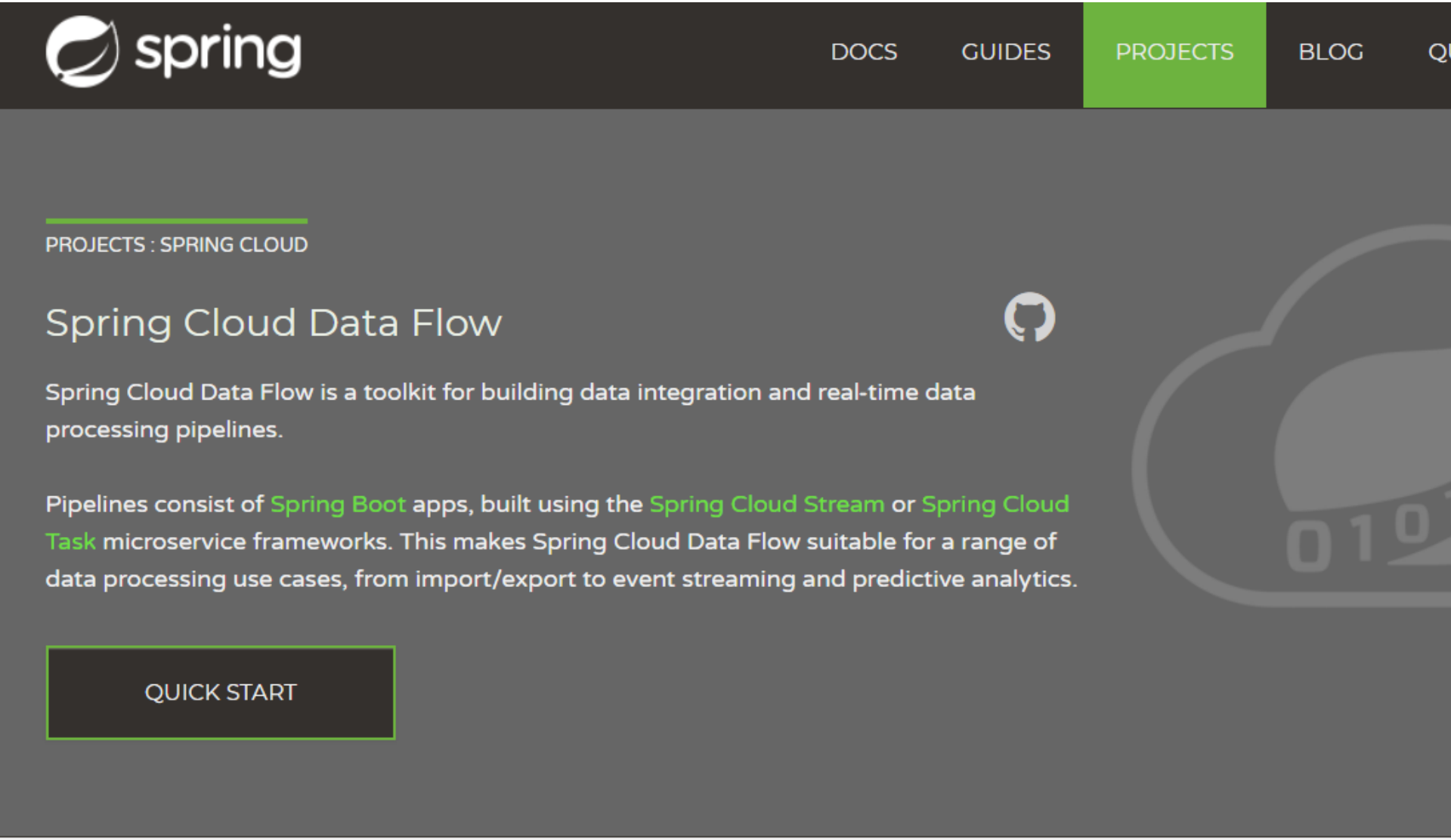
```
@SpringBootApplication
@EnableBinding(Processor.class)
public class MyLoggerServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyLoggerServiceApplication.class, args);
    }

    @StreamListener(Processor.INPUT)
    @SendTo(Processor.OUTPUT)
    public LogMessage enrichLogMessage(LogMessage log) {
        return new LogMessage(String.format("[1]: %s", log.getMessage()));
    }
}
```

Introduction to Spring Cloud Stream
http://www.baeldung.com/spring-cloud-stream

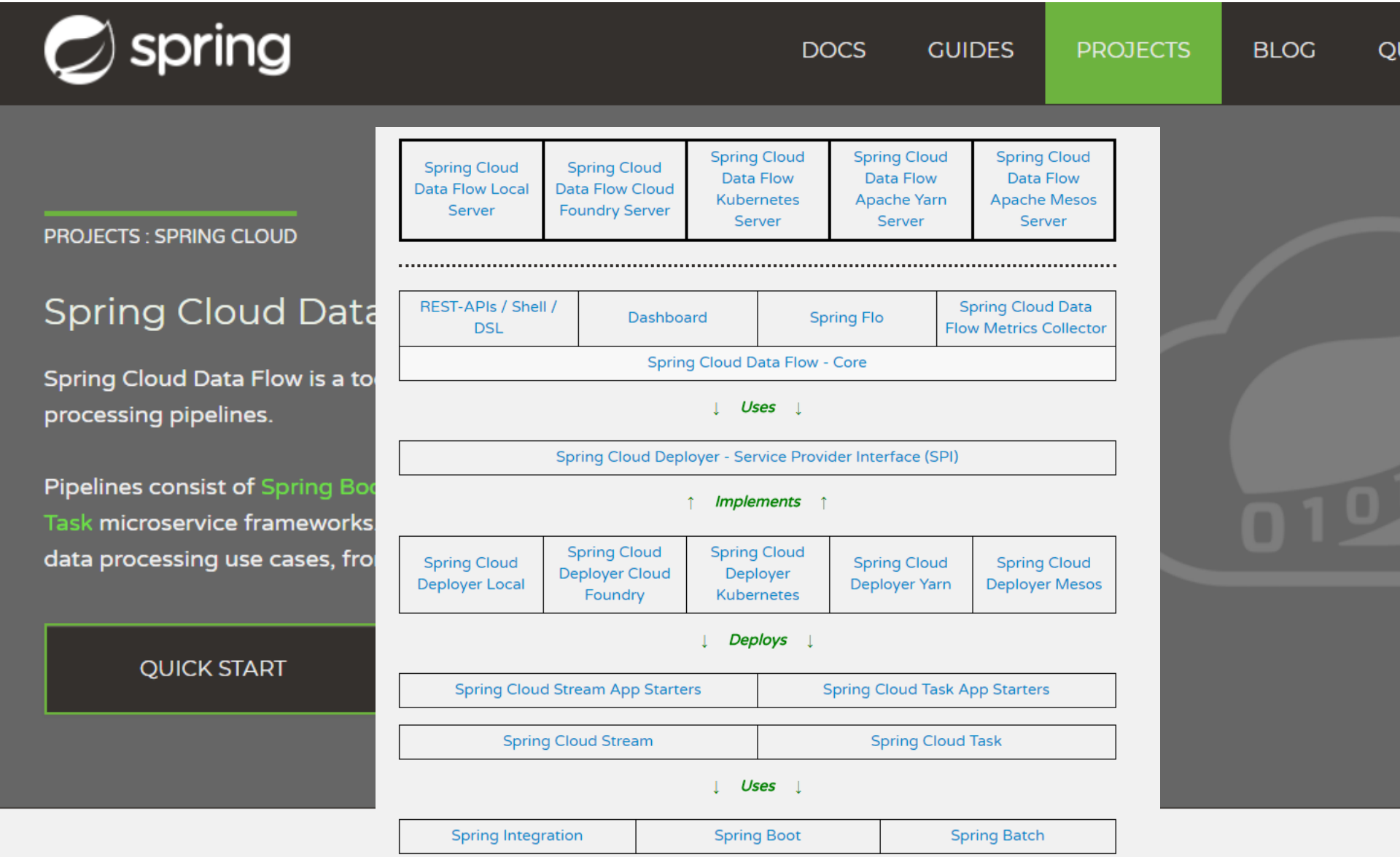
Spring cloud kafka streaming binding

```
@SpringBootApplication
@EnableBinding(KStreamProcessor.class)
public class WordCountProcessorApplication {

    @StreamListener("input")
    @SendTo("output")
    public KStream<?, String> process(KStream<?, String> input) {
        return input
                .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
                .map((key, word) -> new KeyValue<>(word, word))
                .groupByKey(Serdes.String(), Serdes.String())
                .count(TimeWindows.of(5000), "store-name")
                .toStream()
                .map((w, c) -> new KeyValue<>(null, "Count for " + w.key() + ": " + c));
    }

    public static void main(String[] args) {
        SpringApplication.run(WordCountProcessorApplication.class, args);
    }
```

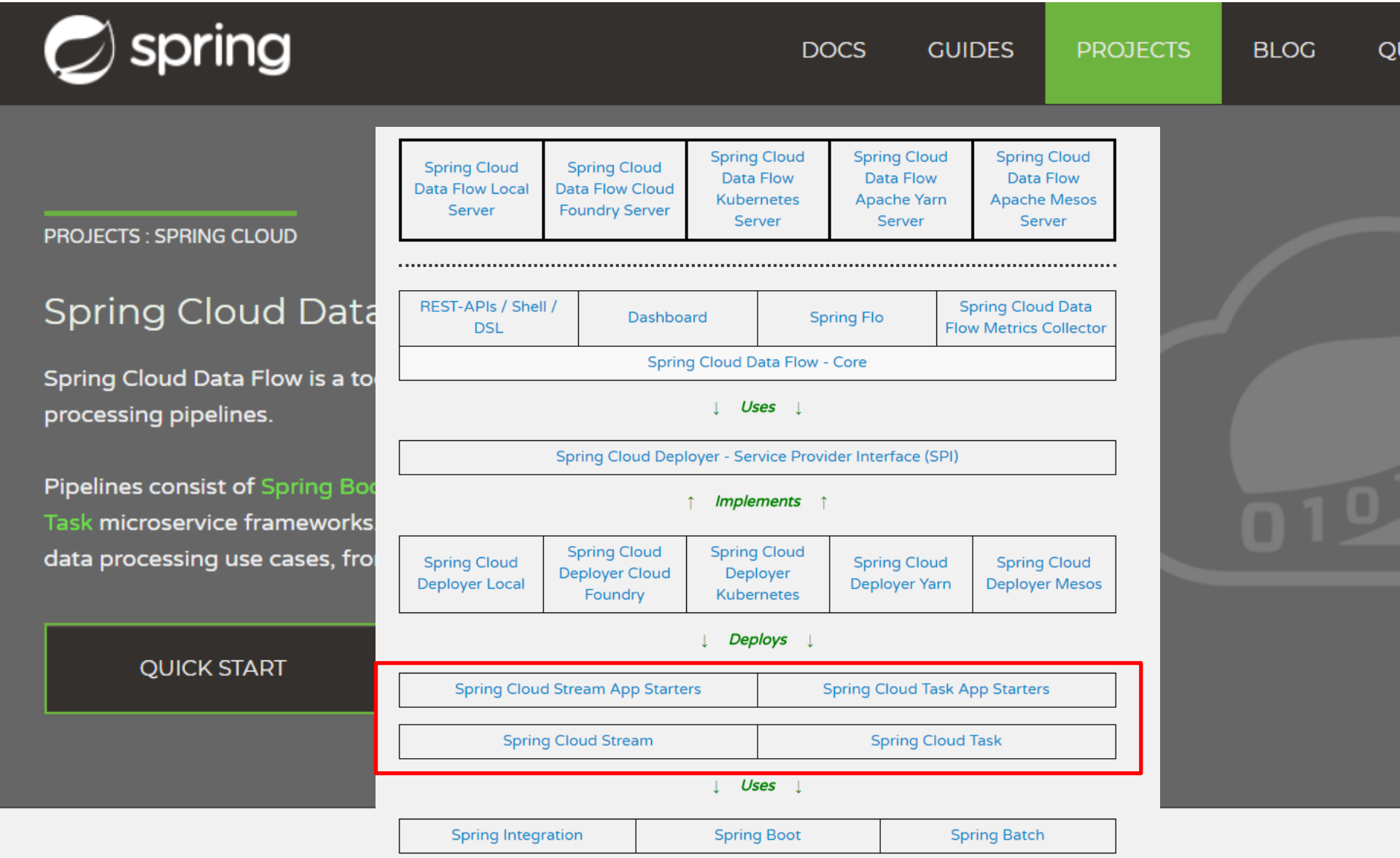
Spring Cloud Stream Reference Guide
https://docs.spring.io/spring-cloud-stream/docs/current/reference/htmlsingle/



https://cloud.spring.io/spring-cloud-dataflow/#quick-start



https://cloud.spring.io/spring-cloud-dataflow/#quick-start



https://cloud.spring.io/spring-cloud-dataflow/#quick-start
https://cloud.spring.io/spring-cloud-stream-app-starters/

PROJECTS : SPRING CLOUD

Spring Cloud Dat

Spring Cloud Data Flow is a to
processing pipelines.

Pipelines consist of Spring Bo
Task microservice framework
data processing use cases, fro

QUICK START

Source	Processor	Sink
file	aggregator	aggregate-counter
ftp	bridge	cassandra
gemfire	filter	counter
gemfire-cq	groovy-filter	field-value-counter
http	groovy-transform	file
jdbc	header-enricher	ftp
jms	httpclient	gemfire
load-generator	pmml	gpfdist
loggregator	python-http	hdfs
mail	python-jython	hdfs-dataset
mongodb	scriptable-transform	jdbc
mqtt	splitter	log
rabbit	tasklaunchrequest-transform	mongodb
s3	tcp-client	mqtt
sftp	tensorflow	pgcopy
syslog	transform	rabbit
tcp	twitter-sentiment	redis-pubsub
tcp-client		router
time		s3
trigger		sftp
triggertask		task-launcher-cloudfoundry
twitterstream		task-launcher-local
		task-launcher-yarn
		tcp
		throughput
		websocket

https://cloud.spring.io/spring-cloud
https://cloud.spring.io/spring-cloud

confluent

Product Solutions Developers Blog Docs Download

KSQL

Streaming SQL for Apache Kafka

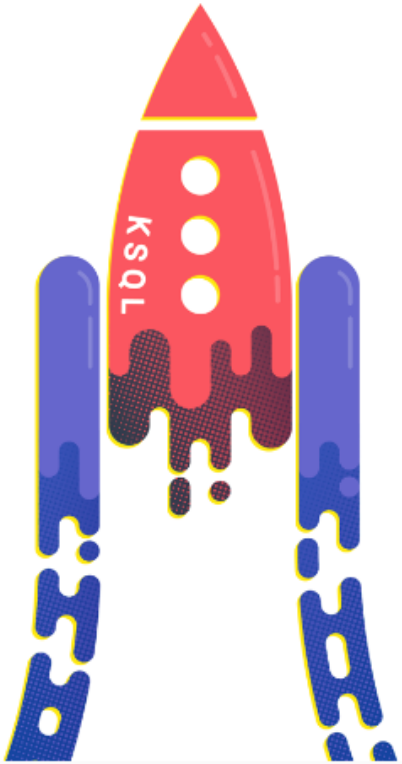
KSQL is an open source, Apache 2.0 licensed streaming SQL engine that enables stream processing against Apache Kafka®.

KSQL makes it easy to read, write, and process streaming data in real-time, at scale, using SQL-like semantics. It offers an easy way to express stream processing transformations as an alternative to writing an application in a programming language such as Java or Python.

KSQL provides powerful stream processing capabilities such as joins, aggregations, event-time windowing, and more!

Try KSQL

Notify me when KSQL is generally available



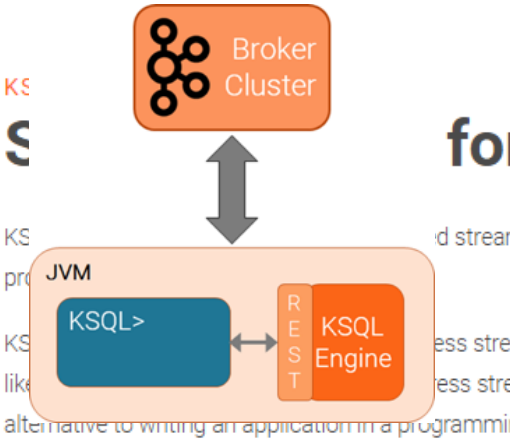
KSQL: Query your streams without writing code

ssing against Kafka today.

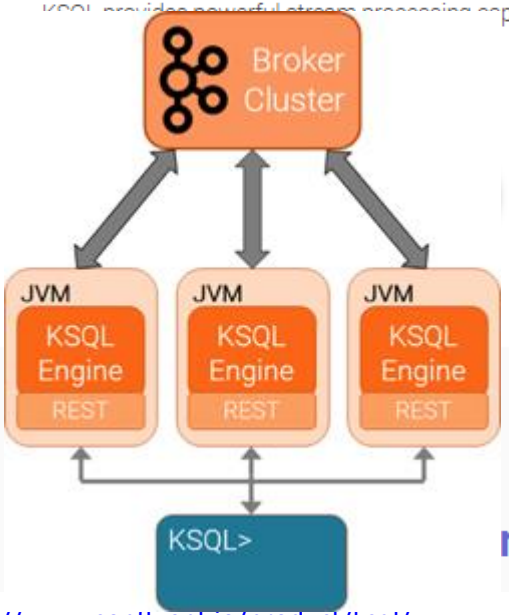
https://www.confluent.io/product/ksql/
https://github.com/confluentinc/ksql/blob/v0.4/docs/concepts.md#concepts

confluent

Product Solutions Developers Blog Docs Download




```
CREATE STREAM vip_actions AS
SELECT userid, page, action
FROM clickstream c
LEFT JOIN users u ON c.userid = u.user_id
WHERE u.level = 'Platinum';
```




```
CREATE TABLE error_counts AS
SELECT error_code, count(*)
FROM monitoring_stream
WINDOW TUMBLING (SIZE 1 MINUTE)
WHERE  type = 'ERROR'
GROUP BY error_code;
```

ry your streams without writing code

essing against Kafka today.

https://www.confluent.io/product/ksql/
https://github.com/confluentinc/ksql/blob/v0.4/docs/concepts.md#concepts

LANDOOP

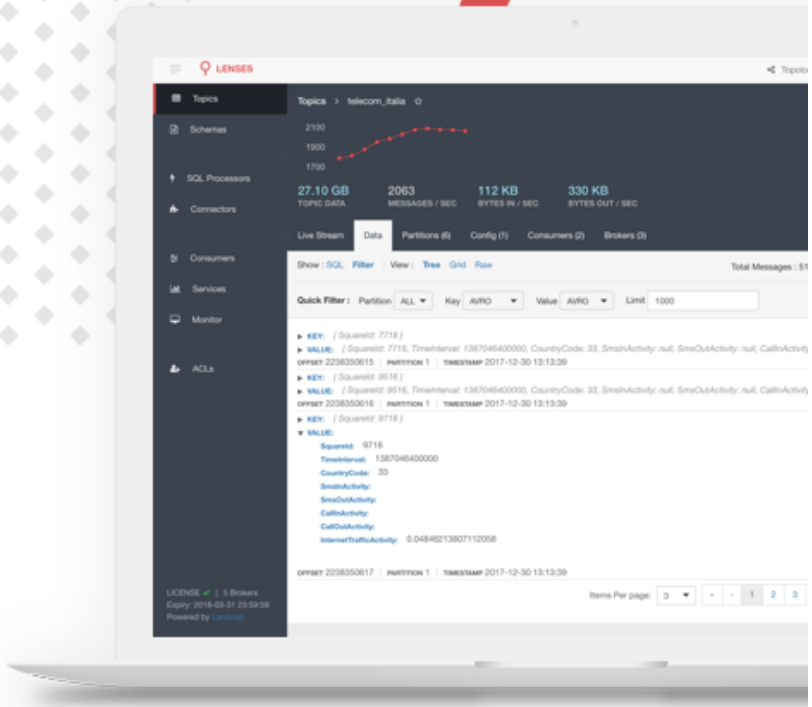
PRODUCTS

DOWNLOAD

LENSES

Successful streaming over Apache Kafka ®

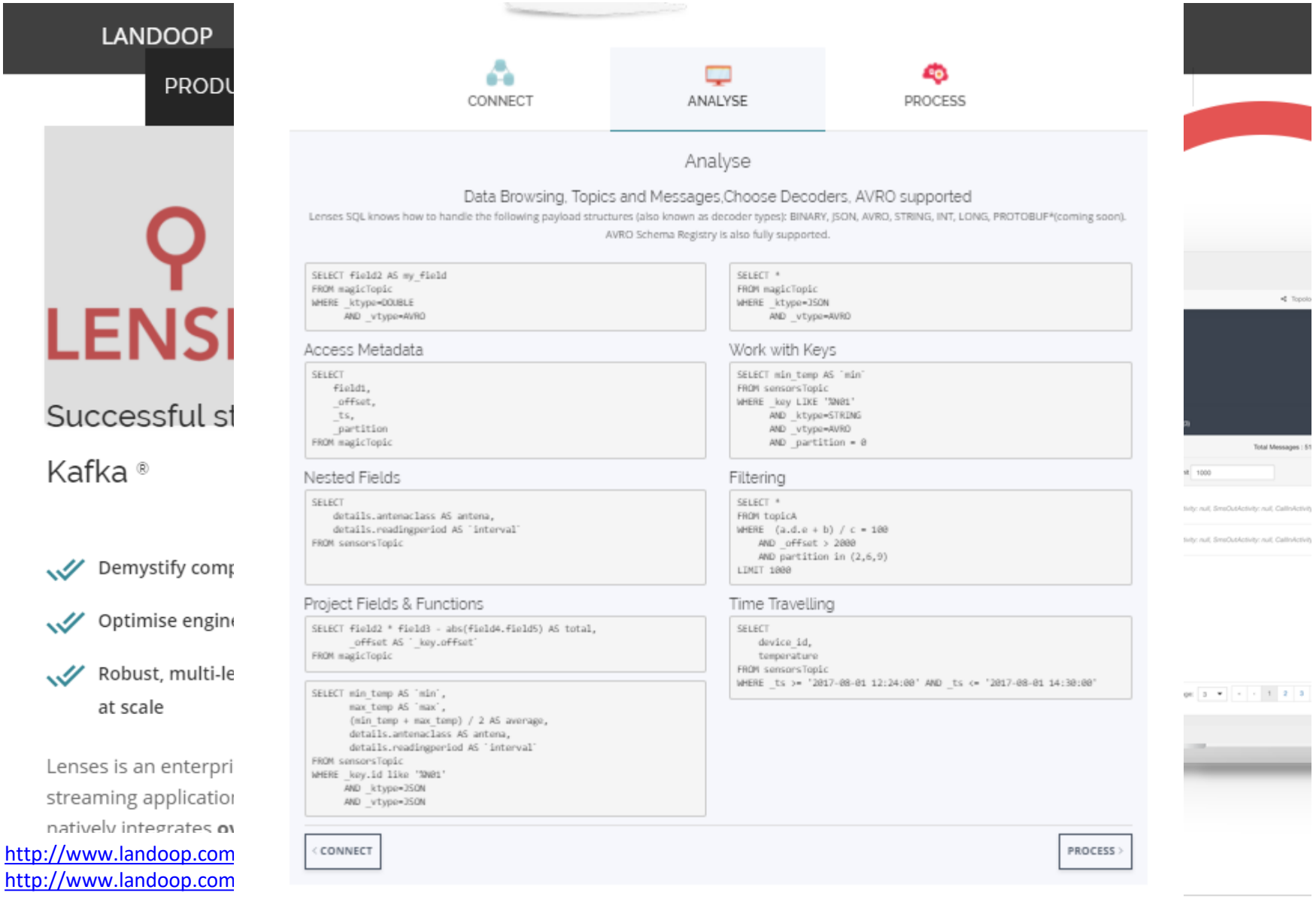
- Demystify complexity, Focus on what's important.
- Optimise engineering resources
- Robust, multi-level production tooling for extreme resilience, at scale



Lenses is an enterprise grade product that provides faster streaming application deliveries and data flow management that natively integrates **over Apache Kafka**.

http://www.landoop.com/kafka-lenses/
http://www.landoop.com/kafka/kafka-sql/

fka with a rich user



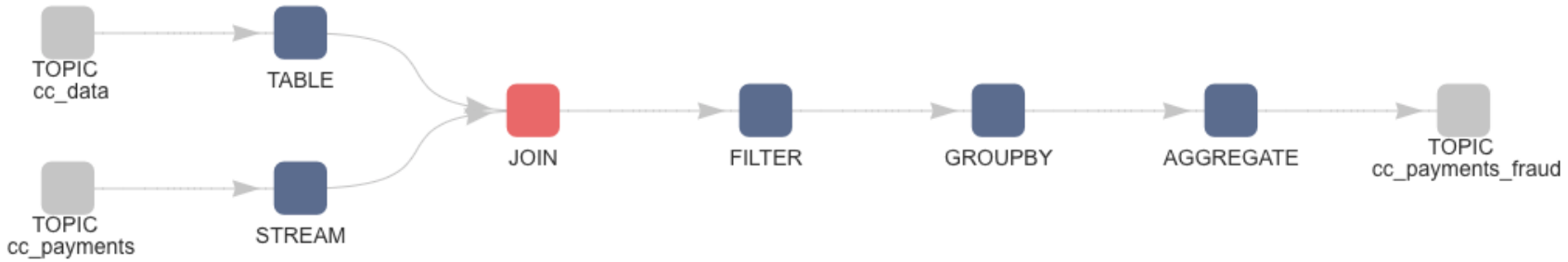
http://www.landoop.com
http://www.landoop.com

```
SET `autocreate`=true;
SET `auto.offset.reset`='earliest';
SET `commit.interval.ms`='120000';
SET `compression.type`='snappy';

INSERT INTO `cc_payments_fraud`
WITH tableCards AS (
  SELECT *
  FROM `cc_data`
  WHERE _ktype='STRING' AND _vtype='AVRO' )

SELECT STREAM
  p.currency,
  sum(p.amount) as total,
  count(*) usage
FROM `cc_payments` AS p LEFT JOIN tableCards AS c ON p._key = c._key
WHERE p._ktype='STRING' AND p._vtype='AVRO' and c.blocked is true
GROUP BY tumble(1,m), p.currency
```

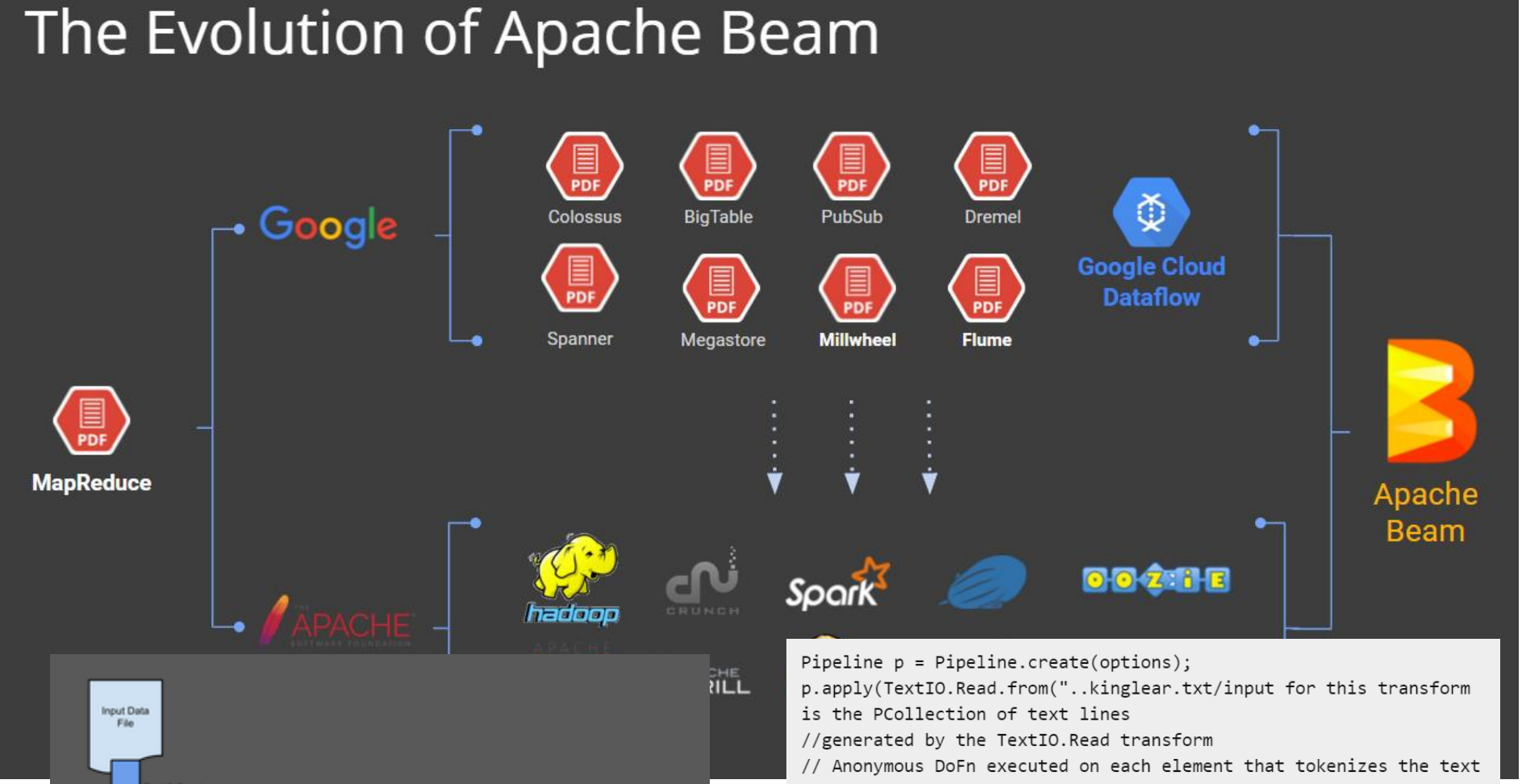
Demystify complexity, Focus on what's important.

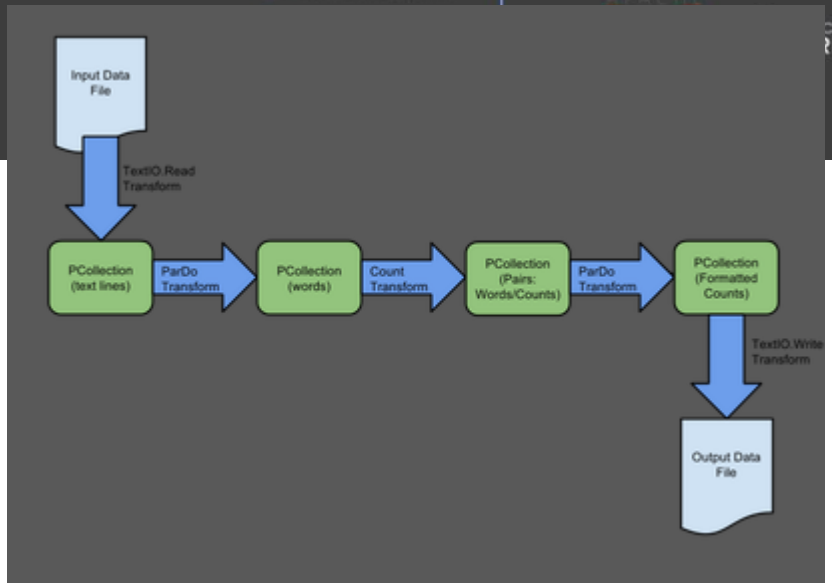
Stream Topology

click on the nodes for details



The Evolution of Apache Beam






```
Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("..kinglear.txt/input for this transform
is the PCollection of text lines
//generated by the TextIO.Read transform
// Anonymous DoFn executed on each element that tokenizes the text
lines into individual words.
.apply(ParDo.named("ExtractWords").of(newng, String>() {
     @Override
     public void processElement(ProcessContext c) {
       for (String word : c.element().split("[^a-zA-Z']+")) {
         if (!word.isEmpty()) {
           c.output(word);
         }
       }
     }
  }))
//  each element in output PCollection represents an individual
word in the text.
```

Beam

What?	Sums of integers, keyed by team.
Where?	Within fixed event-time windows of one hour.
When?	• Early: Every 5 minutes of processing time. • On-time: When the watermark passes the end of the window. • Late: Every 10 minutes of processing time. • Final: When the watermark passes the end of the window + two hours.
How?	Panes accumulate new values into prior results.

```
gameEvents
  [... input ...]
  .apply("LeaderboardTeamFixedWindows", Window
    .<GameActionInfo>into(FixedWindows.of(
      Duration.standardMinutes(Durations.minutes(60))))
    .triggering(AfterWatermark.pastEndOfWindow()
      .withEarlyFirings(AfterProcessingTime.pastFirstElementInPane()
        .plusDelayOf(Durations.minutes(5)))
      .withLateFirings(AfterProcessingTime.pastFirstElementInPane()
        .plusDelayOf(Durations.minutes(10))))
    .withAllowedLateness(Duration.standardMinutes(120)
    .accumulatingFiredPanes())
  .apply("ExtractTeamScore", new ExtractAndSumScore("team"))
  [... output ...]
```

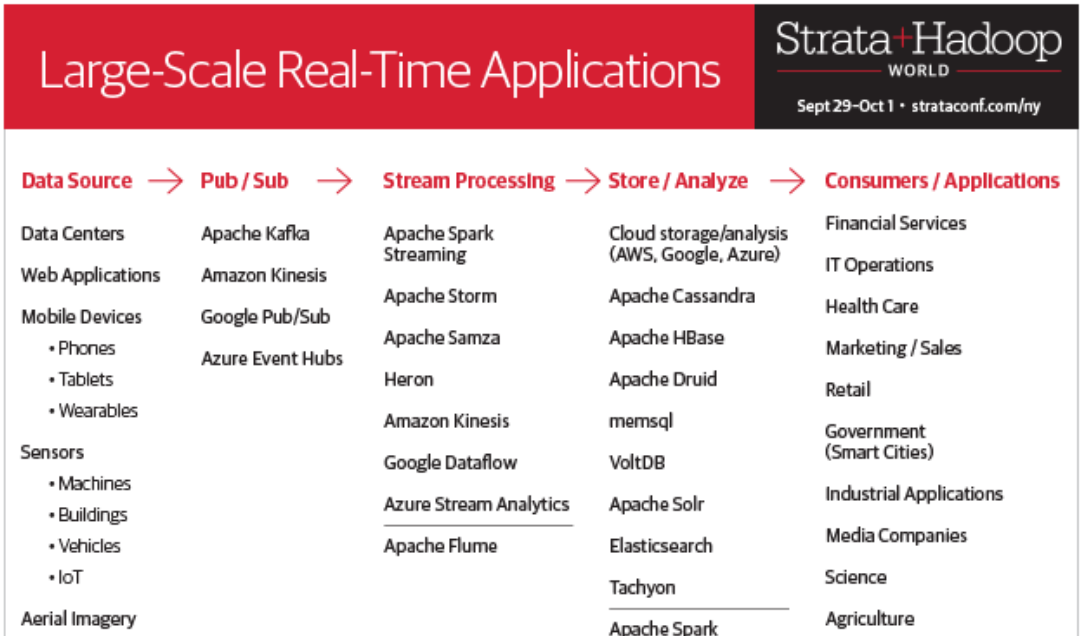
The world beyond batch: Streaming 101
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
The world beyond batch: Streaming 102
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

TECHNOLOGY MUSINGS

Just another WordPress.com site

ANALYTICS

Real-time Streaming ETL and Real-time BI – Part 2



Real-time Streaming ETL and Real-time BI – Part 2
https://phisymmetry.wordpress.com/2015/09/26/real-time-streaming-etl-and-real-time-bi-part-2/

TECHNOLOGY MUSINGS

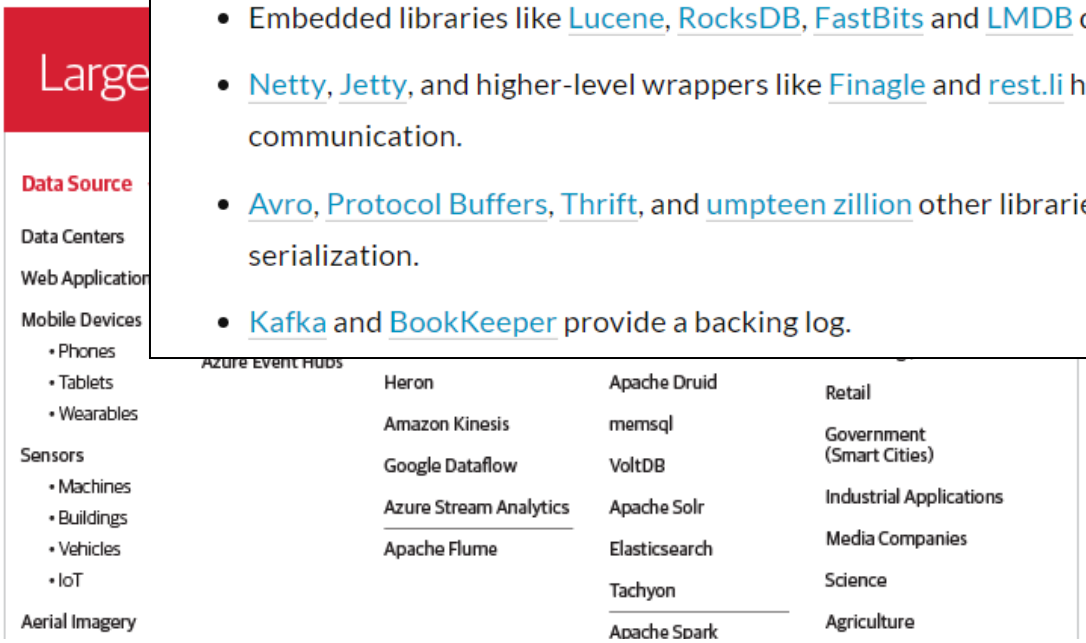
Just another WordPress.com site

ANALYTICS

Real-ti
Real-ti

Now , how does the overall stack look like !

- Zookeeper handles much of the system coordination (via Service Managers like Helix or Curator).
- Mesos and YARN process virtualization and resource management.
- Embedded libraries like Lucene, RocksDB, FastBits and LMDB do indexing.
- Netty, Jetty, and higher-level wrappers like Finagle and rest.li handle remote communication.
- Avro, Protocol Buffers, Thrift, and umpteen zillion other libraries handle serialization.
- Kafka and BookKeeper provide a backing log.



Real-time Streaming ETL and Real-time BI – Part 2
https://phisymmetry.wordpress.com/2015/09/26/real-time-streaming-etl-and-real-time-bi-part-2/

The END