



EVENT STREAM PROCESSING USING KAFKA STREAMS



Fredrik Vraalsen, JFokus 2018, 05.02.2018



WIFI

- Jfokus2018 / Jfokus2018

GETTING STARTED

```
git clone https://github.com/fredriv/kafka-streams-workshop.git
```

```
cd kafka-streams-workshop
```

```
./gradlew build
```

AGENDA

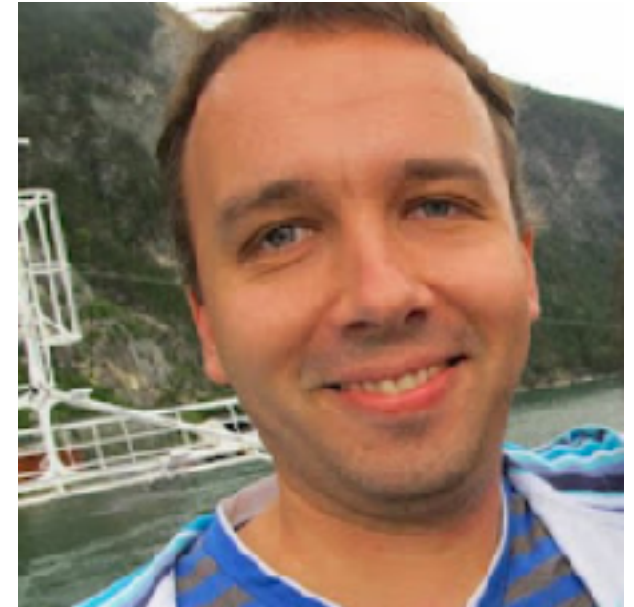
- Intro
- Why streaming?
- Kafka & Kafka Streams
- The basics: Filter, transform and routing
- Next level: Aggregations and windowing
- Joins, enrichments

WHO ARE WE?

- Data Engineers
- Schibsted Data Platform



FREDRIK VRAALSEN

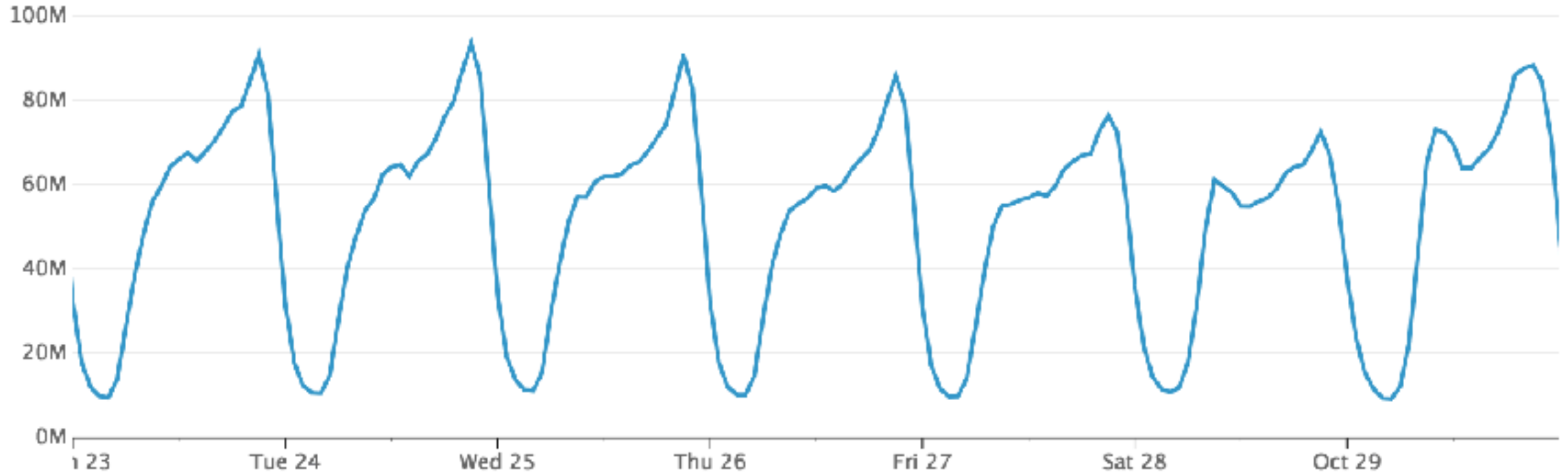


ØYVIND LØKLING

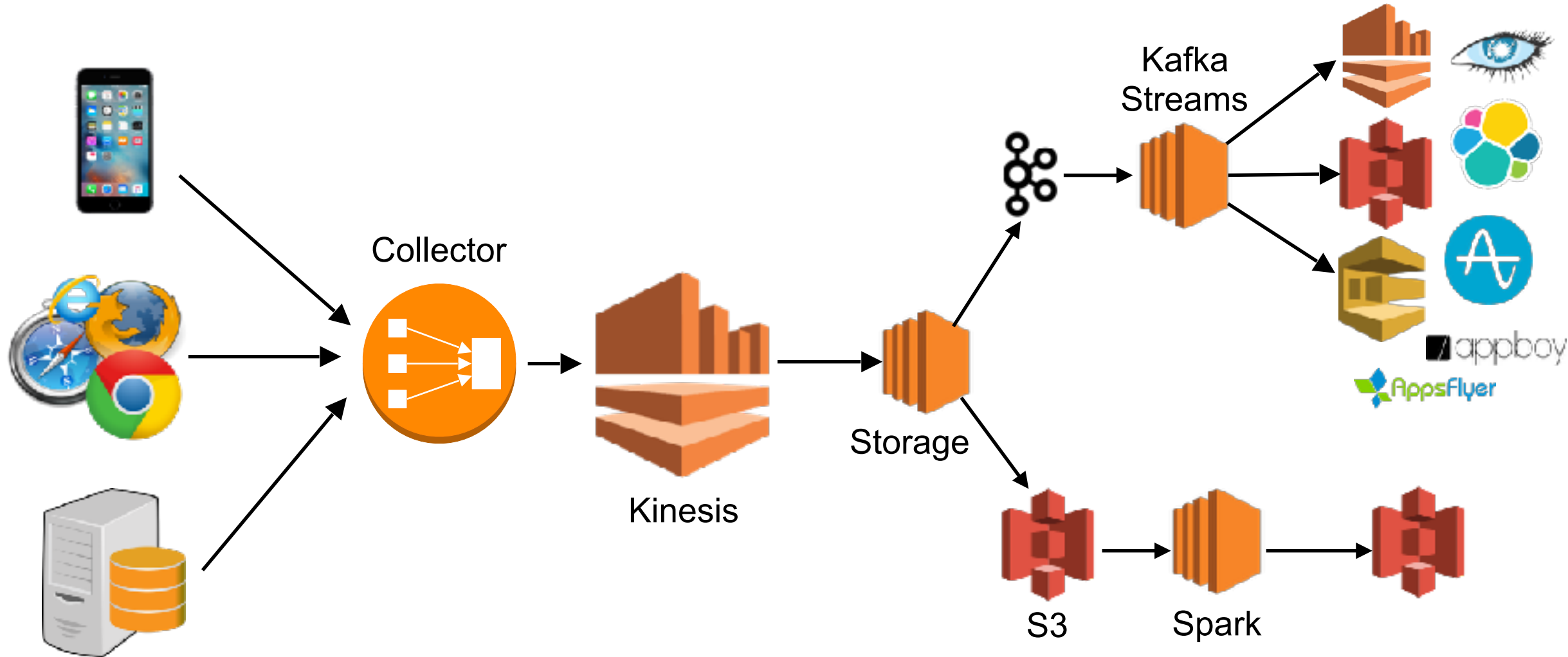
SCHIBSTED



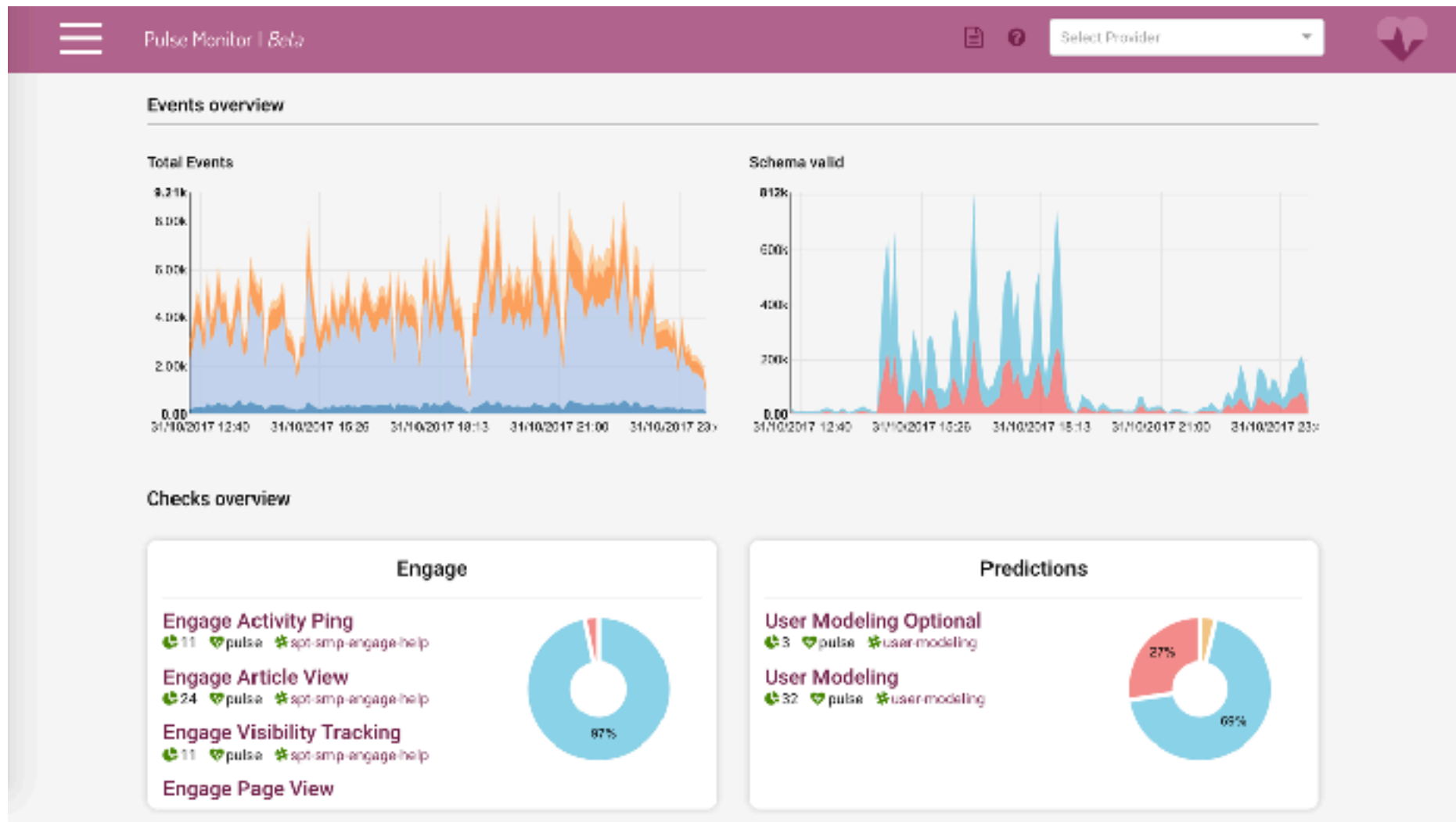
INCOMING EVENTS



DATA PIPELINE



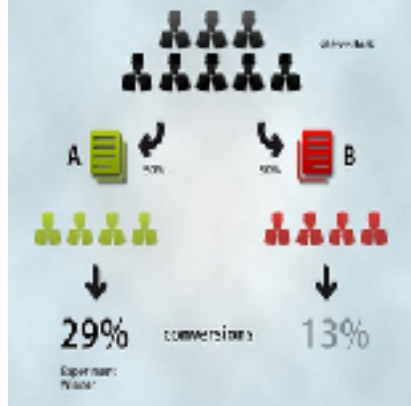
DATA QUALITY



3RD PARTY ANALYTICS



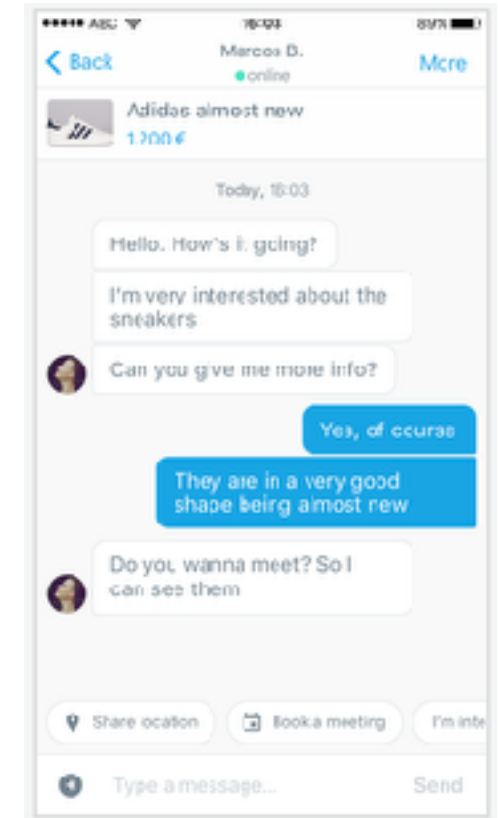
DATA-DRIVEN APPLICATIONS



<https://www.flickr.com/photos/raahulrodriguez/14683524180>



<https://pixabay.com/en/map-photoshop-geolocation-journey-947471/>



<https://www.slideshare.net/DataStax/c-for-deep-learning-andrew-jefferson-tracktable-cassandra-summit-2016>



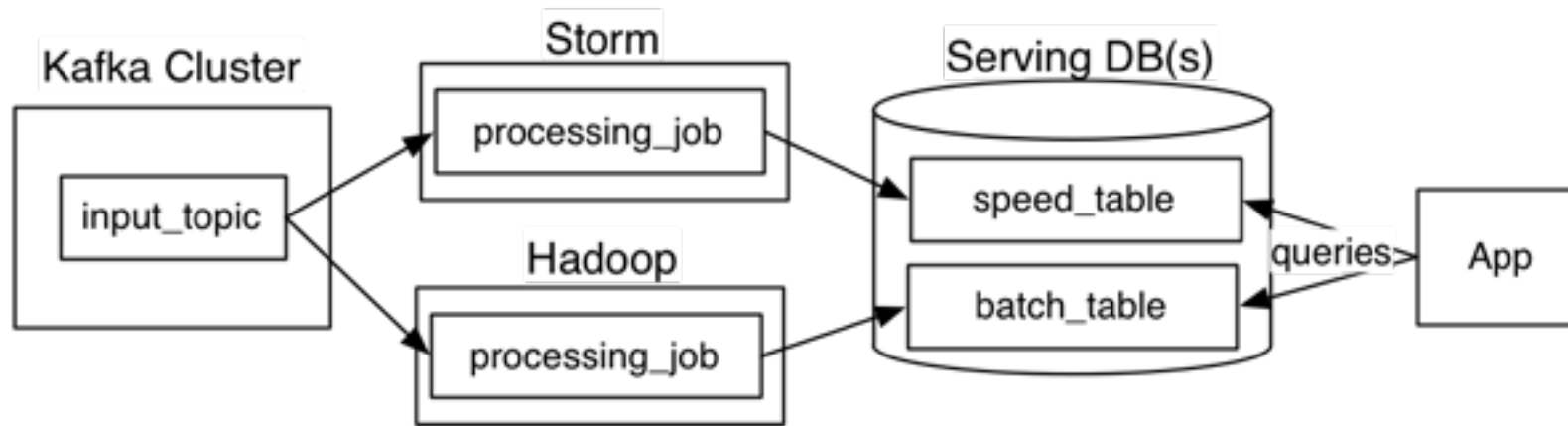
700,000,000 events/day

Lars Marius Garshol, lars.marius.garshol@schibsted.com
2016-09-07, JavaZone 2016
<http://twitter.com/larsga>



WHY STREAMING?

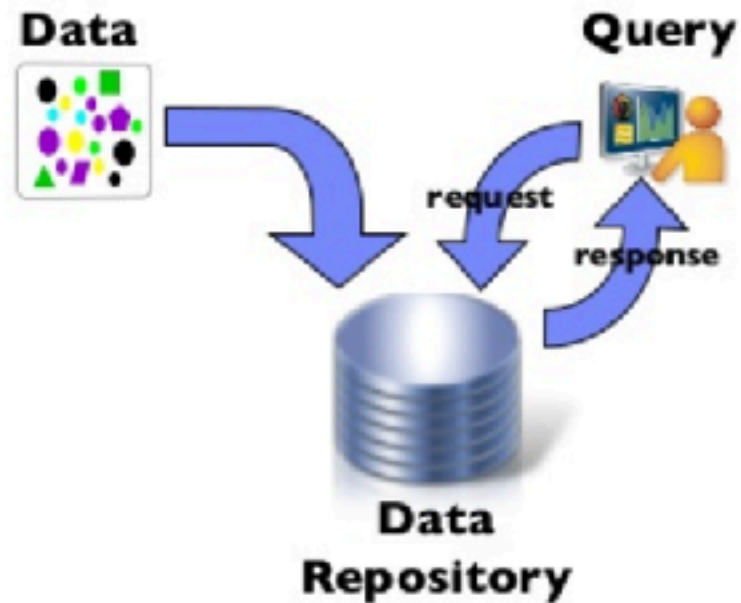
- Real-time, low latency



<https://www.oreilly.com/ideas/questioning-the-lambda-architecture>

STREAM PROCESSING

Traditional Processing



Stream Processing



<https://www.slideshare.net/ConfluentInc/kafka-and-stream-processing-taking-analytics-realtime-mike-spicer>

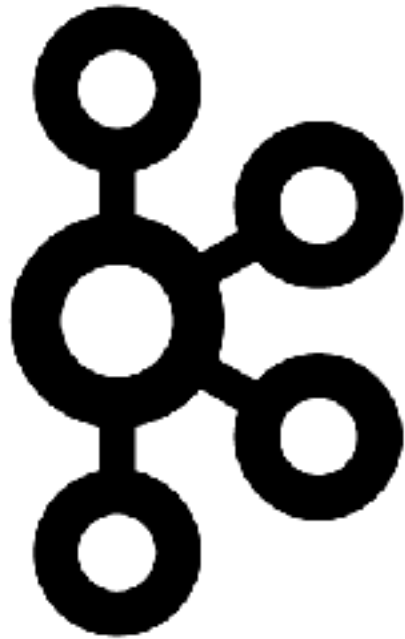
STREAM PROCESSING

- Unbounded datasets
- Unordered events
- Correctness
- Time

STREAM PROCESSING



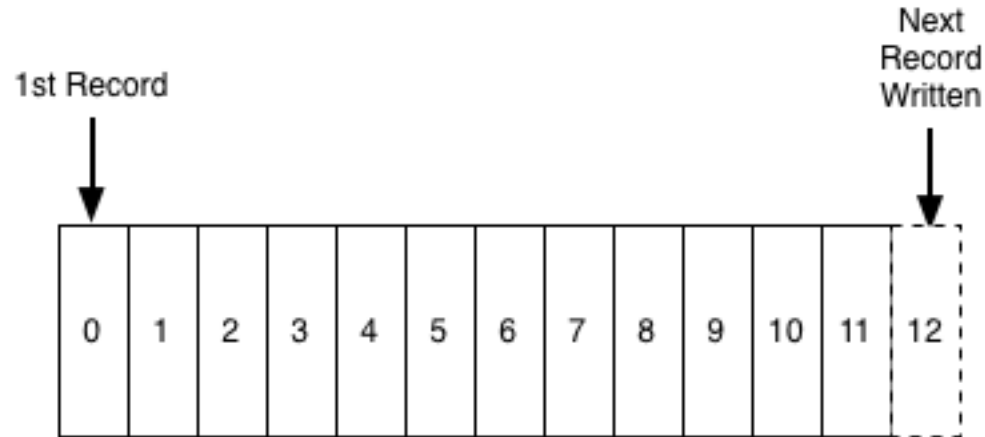
WHY kafka?







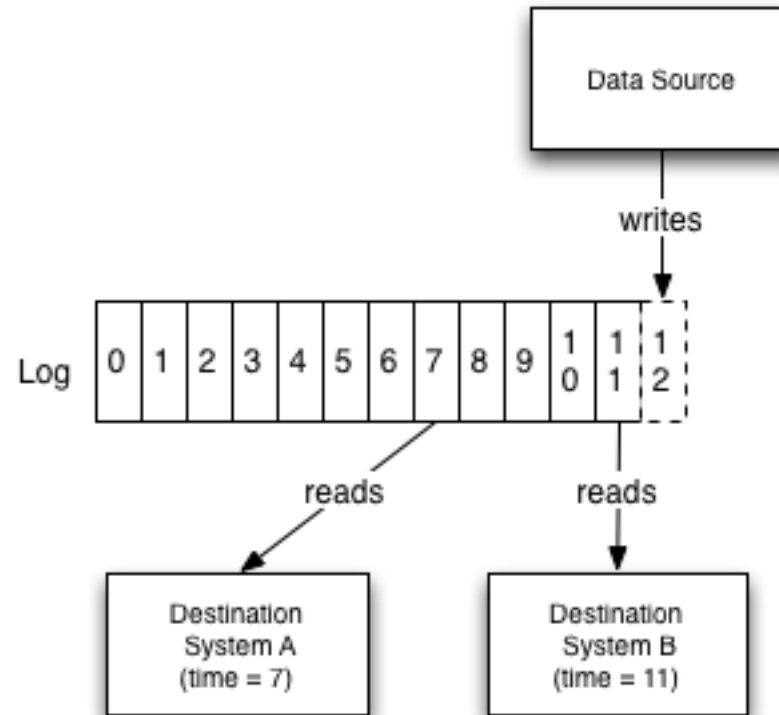
kafka AND THE DISTRIBUTED LOG



<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>



kafka AND THE DISTRIBUTED LOG

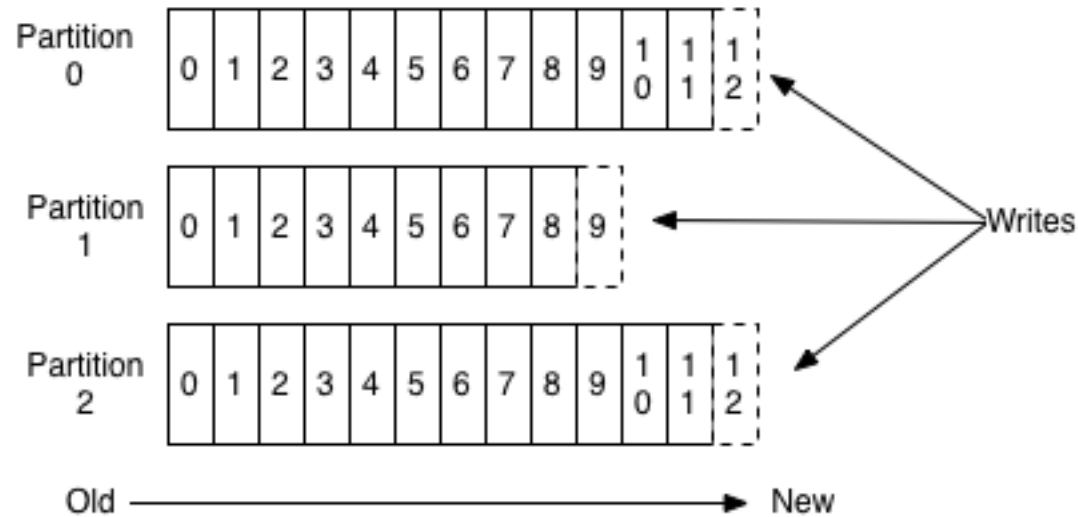


<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>



kafka AND THE DISTRIBUTED LOG

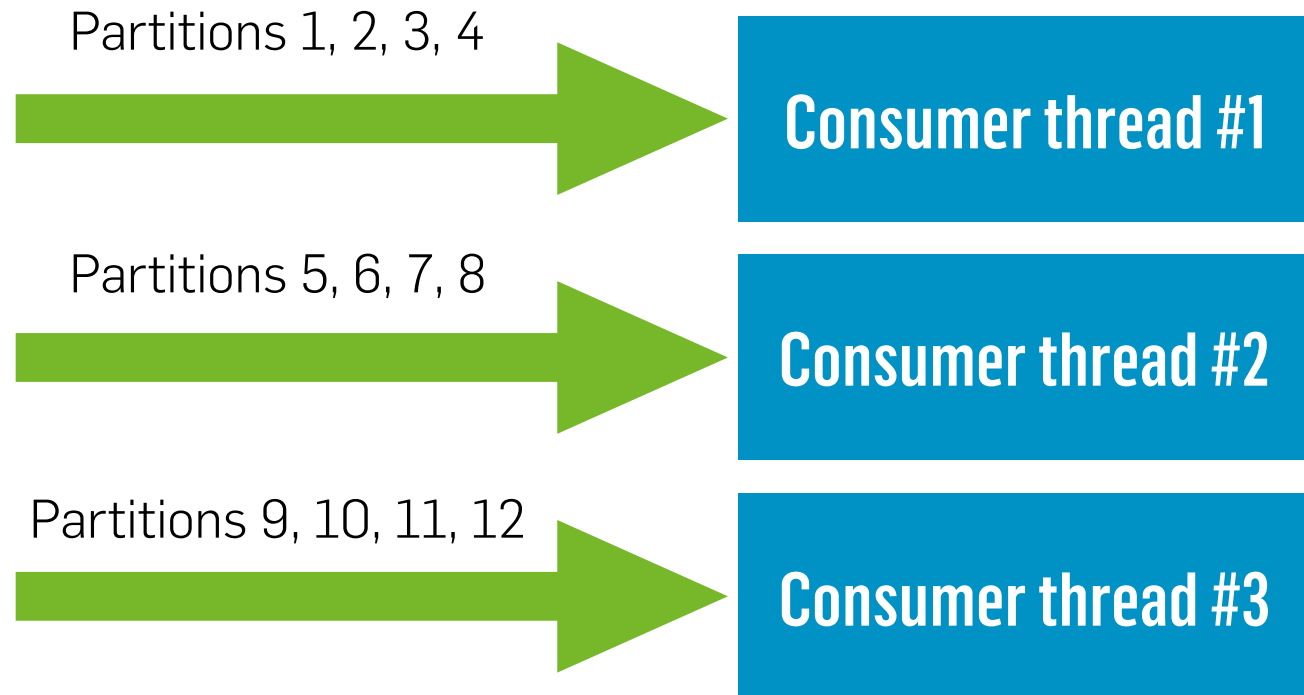
Anatomy of a Topic



<http://kafka.apache.org/documentation.html>

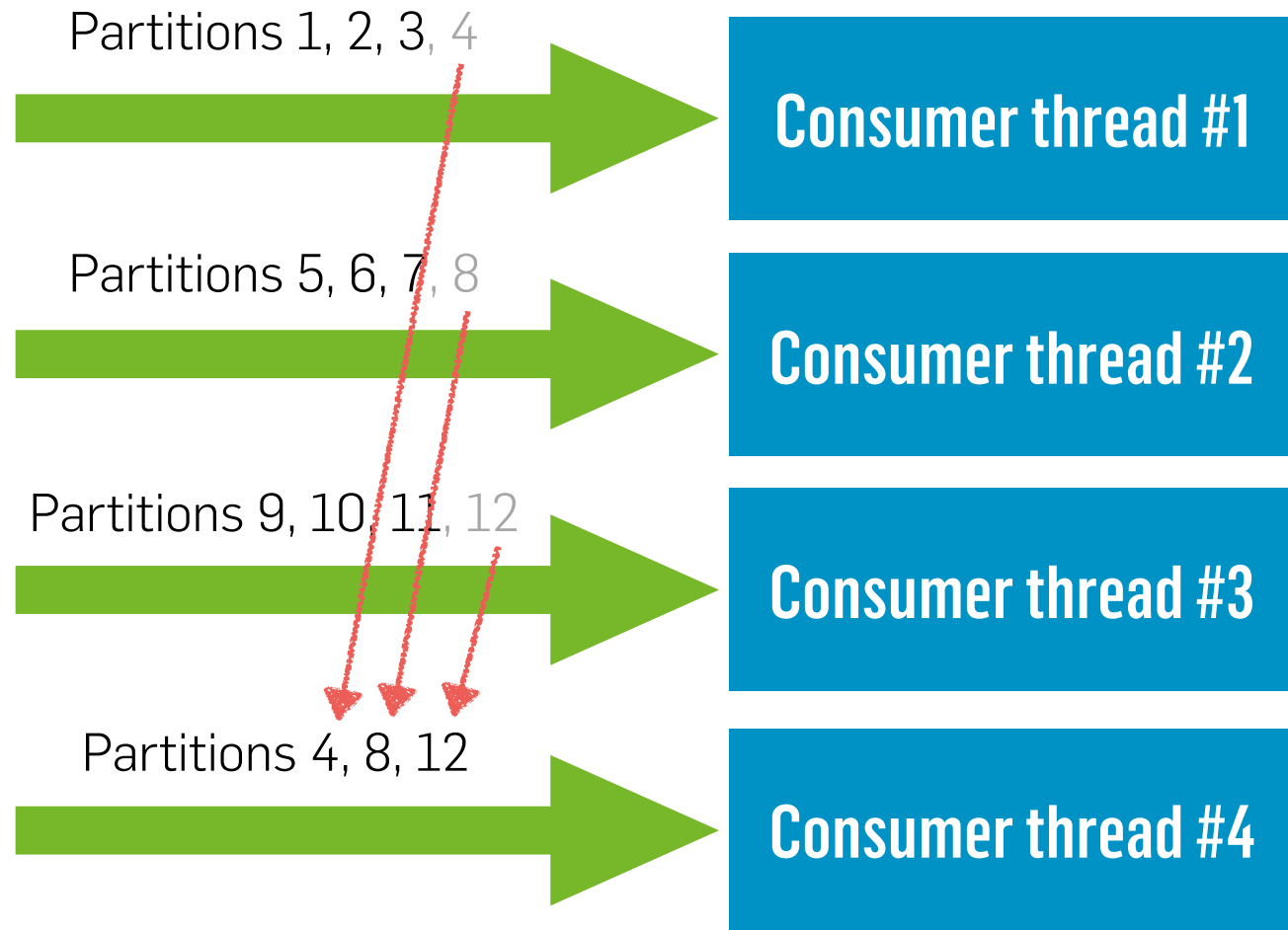


kafka CONSUMER GROUPS





kafka CONSUMER GROUPS





kafka LOG EVENTS

(Key, Value)



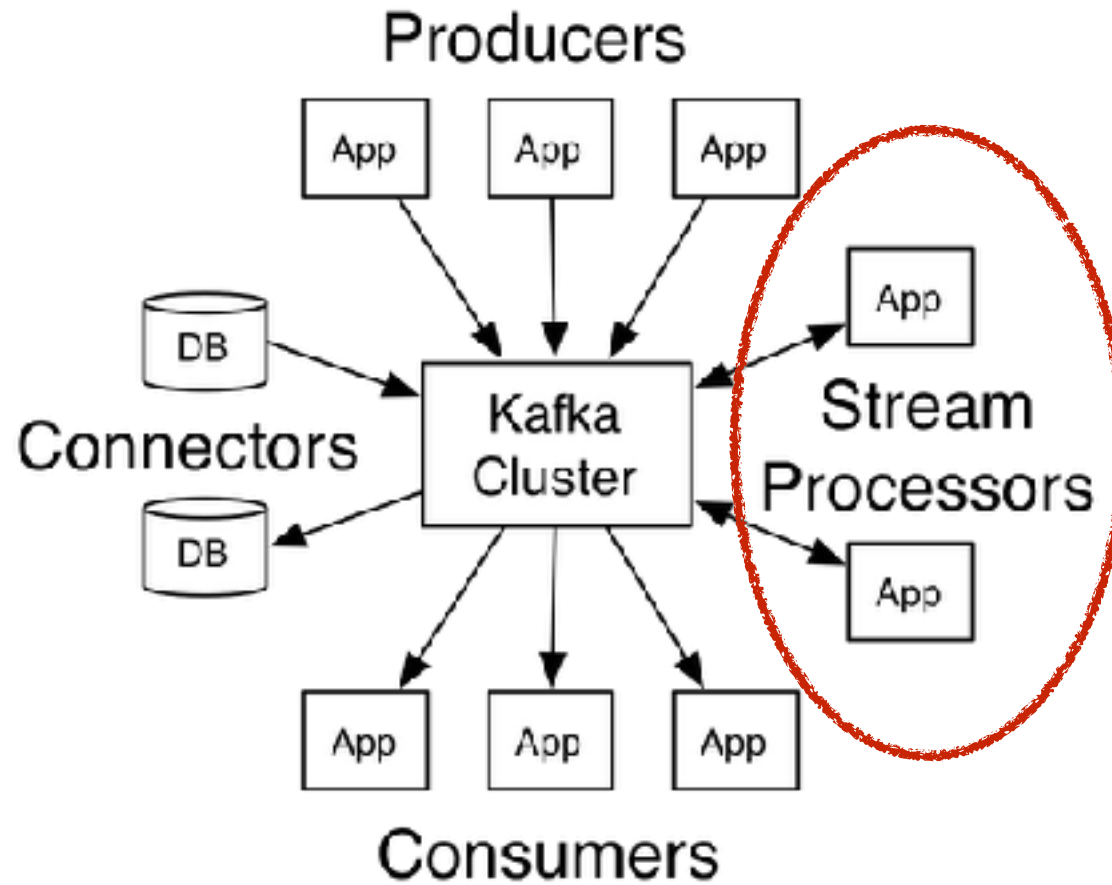
kafka LOG EVENTS

(Key, Value)

+ Timestamp, Metadata



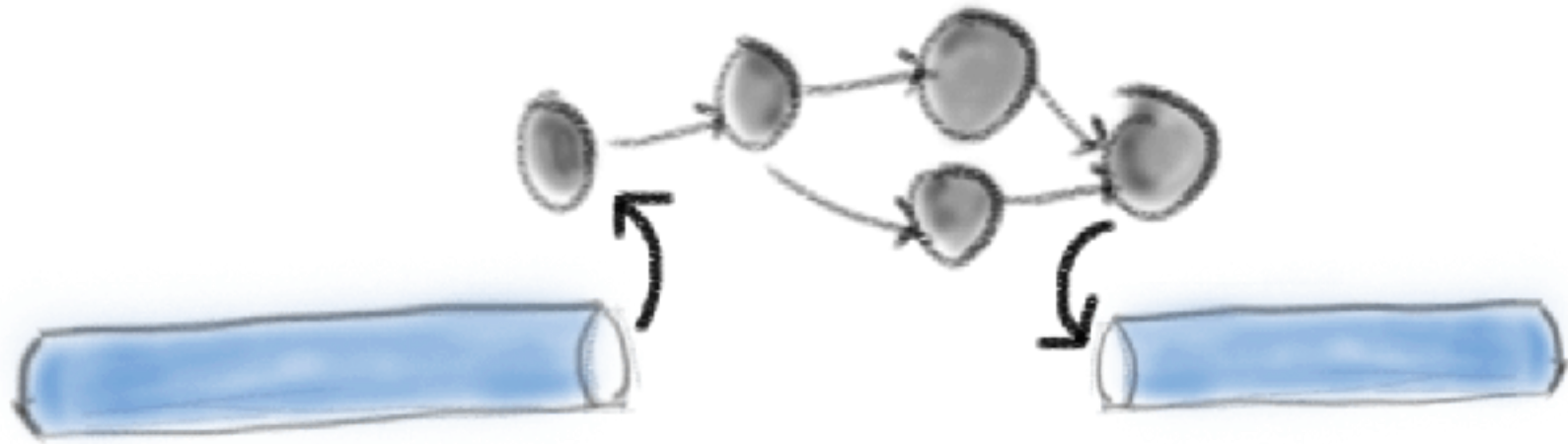
kafka ECOSYSTEM



<http://kafka.apache.org/documentation.html>

Hello

Kafka Streams

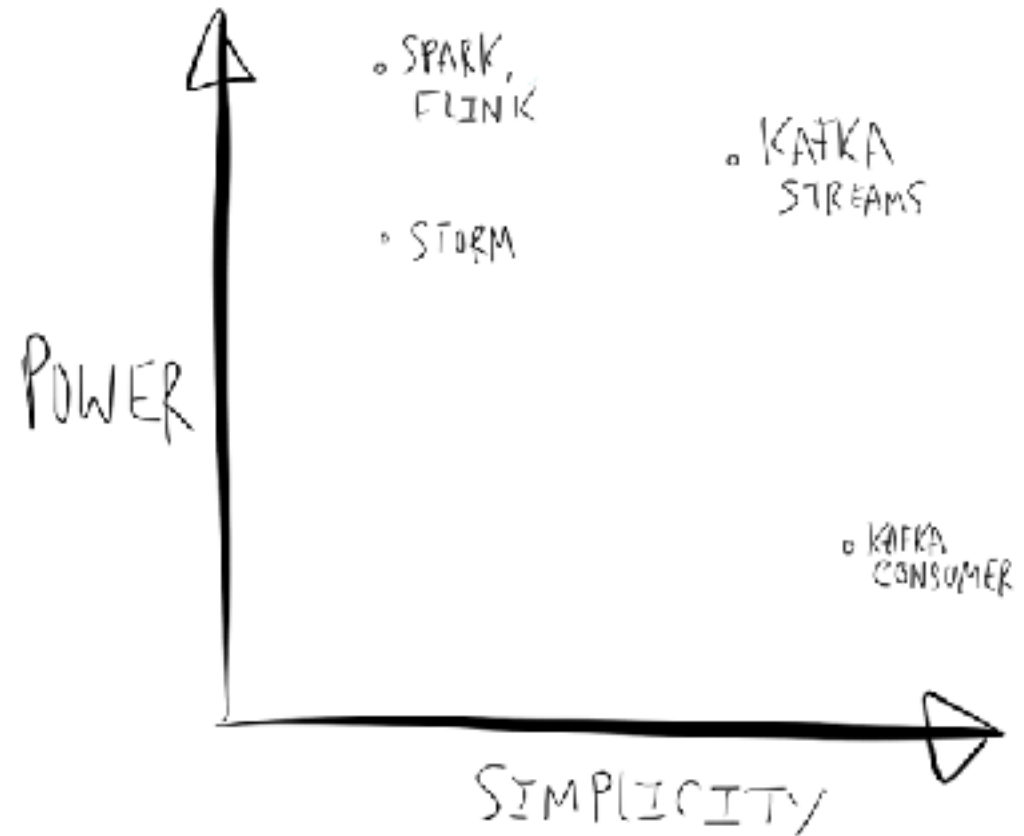


<http://vishnuviswanath.com/hello-kafka-streams.html>

vishnuviswanath.com

KAFKA STREAMS

- Lightweight library
- Streams and tables
- High-level DSL
- Low level API



STREAM PROCESSING COMPONENTS

- Filter
 - Slim down stream, privacy / security
- Transform
 - Bring data into "form", change schema
- Aggregate
 - Compute aggregate values and state (count, sum, etc.)
- Time windows
- Join
 - Enrich by joining datasets



ANATOMY OF A KAFKA STREAMS APP

ANATOMY OF A KAFKA STREAMS APP

```
Properties config = new Properties();  
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");  
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
```

ANATOMY OF A KAFKA STREAMS APP

```
Properties config = new Properties();  
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");  
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
```

```
StreamsBuilder builder = new StreamsBuilder();  
// TODO Build topology
```

ANATOMY OF A KAFKA STREAMS APP

```
Properties config = new Properties();  
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");  
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
```

```
StreamsBuilder builder = new StreamsBuilder();  
// TODO Build topology
```

```
KafkaStreams streams = new KafkaStreams(topology, config);  
streams.start();
```

```
Runtime.getRuntime().addShutdownHook(new Thread(() ->  
    streams.close(10, TimeUnit.SECONDS)  
));
```

ANATOMY OF A KAFKA STREAMS APP

```
Properties config = new Properties();  
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "hello-world-app");  
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
```

```
StreamsBuilder builder = new StreamsBuilder();  
// TODO Build topology
```

```
KafkaStreams streams = new KafkaStreams(topology, config),  
streams.start();
```

```
Runtime.getRuntime().addShutdownHook(new Thread(() ->  
    streams.close(10, TimeUnit.SECONDS)  
));
```


HELLO KAFKA STREAMS

```
StreamBuilder builder = new StreamBuilder();
```

```
Serde<String> strings = Serdes.String();
```

HELLO KAFKA STREAMS

```
StreamBuilder builder = new StreamBuilder();
```

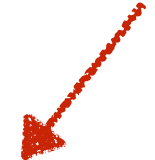
```
Serde<String> strings = Serdes.String();
```

```
KStream<String, String> articles = builder.stream("Articles",  
Consumed.with(strings, strings));
```

Topic

Key

Value



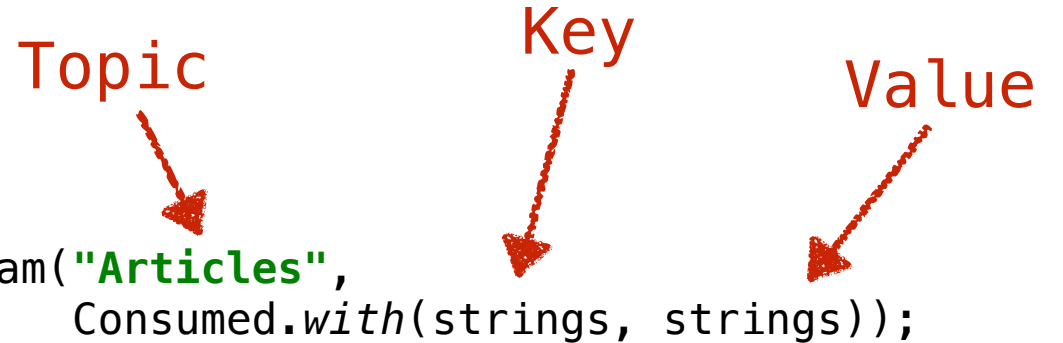
HELLO KAFKA STREAMS

```
StreamBuilder builder = new StreamBuilder();
```

```
Serde<String> strings = Serdes.String();
```

```
KStream<String, String> articles = builder.stream("Articles",  
Consumed.with(strings, strings));
```

```
articles.print(Printed.toSysOut());
```



HELLO KAFKA STREAMS

```
StreamBuilder builder = new StreamBuilder();  
Serde<String> strings = Serdes.String();  
KStream<String, String> articles = builder.stream("Articles",  
Consumed.with(strings, strings));  
  
articles.print(Printed.toSysOut());
```

2, {"site": "foxnews", "title": "Russia internet wars continue, to Trump's ..."}
3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}
4, {"site": "cnn", "title": "Italian town sells homes for \$1"}
6, {"site": "cnn", "title": "US: More Russia sanctions in 'near future'"}
1, {"site": "cnn", "title": "Ex-Googlers create a self driving car ..."}
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}
7, {"site": "foxnews", "title": "FBI officials review damning surveillance memo ..."}
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}

Topic Key Value

JSON

```
StreamBuilder builder = new StreamBuilder();
```

```
Serde<String> strings = Serdes.String();  
Serde<JsonNode> json = new JsonNodeSerde();
```

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
Consumed.with(strings, json));
```

```
articles.print(Printed.toSysOut());
```

```
2, {"site": "foxnews", "title": "Russia internet wars continue, to Trump's ..."}  
3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}  
4, {"site": "cnn", "title": "Italian town sells homes for $1"}  
6, {"site": "cnn", "title": "US: More Russia sanctions in 'near future'"}  
1, {"site": "cnn", "title": "Ex-Googlers create a self driving car ..."}  
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}  
7, {"site": "foxnews", "title": "FBI officials review damning surveillance memo ..."}  
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}
```

Topic

Key

Value

Consumed.with(strings, json);

SERDES

```
class JsonNodeSerde extends Serde[JsonNode] {
```

```
  override val serializer = new Serializer[JsonNode] {  
    override def serialize(topic: String, data: JsonNode): Array[Byte] = ???  
    override def configure ...  
    override def close ...  
  }
```

```
  override val deserializer = new Deserializer[JsonNode] {  
    override def deserialize(topic: String, data: Array[Byte]): JsonNode = ???  
    override def configure ...  
    override def close ...  
  }
```

```
  override def configure ...  
  override def close ...  
}
```

SERDES

```
class JsonNodeSerde extends Serde[JsonNode] {  
  
  private val mapper = new ObjectMapper() // Jackson JSON API  
  
  override val serializer = new Serializer[JsonNode] {  
    override def serialize(topic: String, data: JsonNode): Array[Byte] = mapper.writeValueAsBytes(data)  
    override def configure ...  
    override def close ...  
  }  
  
  override val deserializer = new Deserializer[JsonNode] {  
    override def deserialize(topic: String, data: Array[Byte]): JsonNode = mapper.readTree(data)  
    override def configure ...  
    override def close ...  
  }  
  
  override def configure ...  
  override def close ...  
}
```

DEFAULT SERDES

```
config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());  
config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, JsonNodeSerde.class);
```

```
StreamBuilder builder = new StreamBuilder();
```

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```




THE BASICS

FILTER

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

FILTER

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KStream<String, JsonNode> bbcArticles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()));
```

FILTER

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

```
KStream<String, JsonNode> bbcArticles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()));
```

```
bbcArticles.print(Printed.toSysOut());
```

```
3, {"site": "bbc", "title": "Employees urged to let staff 'rest'"}  
5, {"site": "bbc", "title": "What to watch for in Trump's SOTU speech"}  
8, {"site": "bbc", "title": "The truth about the origin of macaroni cheese"}
```

TRANSFORM

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

```
KStream<String, JsonNode> bbcArticles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()));
```

```
KStream<String, String> bbcTitles = bbcArticles  
    .mapValues(article -> article.path("title").asText());
```

```
bbcTitles.print(Printed.toSysOut());
```

```
3, Employees urged to let staff 'rest'  
5, What to watch for in Trump's SOTU speech  
8, The truth about the origin of macaroni cheese
```

FILTER AND TRANSFORM

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

```
KStream<String, String> bbcTitles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()))  
    .mapValues(article -> article.path("title").asText());
```

```
bbcTitles.print(Printed.toSysOut());
```

```
3, Employees urged to let staff 'rest'  
5, What to watch for in Trump's SOTU speech  
8, The truth about the origin of macaroni cheese
```

WRITING BACK TO KAFKA

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KStream<String, String> bbcTitles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()))  
    .mapValues(article -> article.path("title").asText());  
  
bbcTitles.to("BBC-Titles", Produced.with(strings, strings));
```

```
fredriv@fredriv-mac 20:30:33  
~ master $ kafka-console-consumer.sh --bootstrap-server localhost:29092 --topic BBC-Titles --from-beginning  
What to watch for in Trump's SOTU speech  
Employees urged to let staff 'rest'  
The truth about the origin of macaroni cheese  
^CProcessed a total of 3 messages
```

SCALA GOTCHAS

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KStream<String, String> bbcTitles = articles  
    .filter((key, article) -> "bbc".equals(article.path("site").asText()))  
    .mapValues(article -> article.path("title").asText());  
  
bbcTitles.to("BBC-Titles", Produced.with(strings, strings));
```


SCALA GOTCHAS

```
val articles = builder.stream("Articles",  
                             Consumed.`with`(strings, json))  
  
val bbcTitles = articles  
  .filter((key, article) => article.path("site").asText == "bbc")  
  .mapValues[String](article => article.path("title").asText)  
  
bbcTitles.to("BBC-Titles", Produced.`with`(strings, strings))
```

SCALA DSL

```
import kafkastreams.scalautils.JacksonDSL._
import kafkastreams.scalautils.KafkaStreamsDSL._

val articles = builder.streamS[String, JsonNode]("Articles")

val bbcTitles = articles
  .filter((key, article) => article("site").asText == "bbc")
  .mapValuesS(article => article("title").asText)

bbcTitles.toS("BBC-Titles")
```

SCALA DSL

```
import kafkastreams.scalautils.JacksonDSL._  
import kafkastreams.scalautils.KafkaStreamsDSL._
```

```
val articles = builder.streamS[String, JsonNode]("Articles")
```

```
val bbcTitles = articles \  
  (_("site").asText == "bbc") ~>  
  (_("title").asText)
```

```
bbcTitles ~> "BBC-Titles"
```

TRANSFORM

- Convert 1 input event to sequence of output events
 - Zero, one or many
- flatMap / flatMapValues
 - value -> Iterable
 - e.g. Collection, Arrays.asList, Iterator(*ish*)

FLATMAP

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, String> authors = articles  
    .flatMapValues(json -> json.path("authors").elements()) // fails :-(  
    .mapValues(author -> author.asText());
```

FLATMAP

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, String> authors = articles  
    .flatMapValues(json -> new Iterable<JsonNode>() {  
        @Override  
        public Iterator<JsonNode> iterator() {  
            return json.path("authors").elements();  
        }  
    })  
    .mapValues(author -> author.asText());
```

FLATMAP

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, String> authors = articles  
    .flatMapValues(json -> () -> json.path("authors").elements())  
    .mapValues(author -> author.asText());
```

BRANCHING

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, JsonNode>[] articlesBySite = articles.branch(
```

```
);
```


BRANCHING

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, JsonNode>[] articlesBySite = articles.branch(  
    (key, value) -> "bbc".equals(value.path("site").asText()),  
    (key, value) -> "cnn".equals(value.path("site").asText()),  
    (key, value) -> "foxnews".equals(value.path("site").asText()),  
    (key, value) -> true  
);
```

BRANCHING

```
KStream<String, JsonNode> articles = builder.stream("Articles");
```

```
KStream<String, JsonNode>[] articlesBySite = articles.branch(  
    (key, value) -> "bbc".equals(value.path("site").asText()),  
    (key, value) -> "cnn".equals(value.path("site").asText()),  
    (key, value) -> "foxnews".equals(value.path("site").asText()),  
    (key, value) -> true  
);
```

```
articlesBySite[0].to("BBC-Articles");  
articlesBySite[1].to("CNN-Articles");  
articlesBySite[2].to("FoxNews-Articles");  
articlesBySite[3].to("Other-Articles");
```

EXERCISE 1

- Open `Exercise_1_FilterAndTransform`
- Implement methods to make tests pass
 - Run in IDE or with: `./gradlew test`
 - Tests in `Exercise_1_FilterAndTransformTest`
 - Test data in `ClickEvents`
- Should be sufficient with: `filter`, `mapValues`, `flatMapValues`, `branch`
- For more:
 - <https://kafka.apache.org/10/javadoc/index.html>
 - <https://docs.confluent.io/4.0.0/streams/developer-guide/dsl-api.html>



NEXT LEVEL

AGGREGATIONS

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

Number of articles per site?

AGGREGATIONS

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KGroupedStream<String, JsonNode> grouped = articles  
    .groupBy((key, value) -> value.path("site").asText(),  
           Serialized.with(strings, json));
```

AGGREGATIONS

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KGroupedStream<String, JsonNode> grouped = articles  
    .groupBy((key, value) -> value.path("site").asText(),  
            Serialized.with(strings, json));  
  
KTable<String, Long> counts = grouped.count();
```

AGGREGATIONS

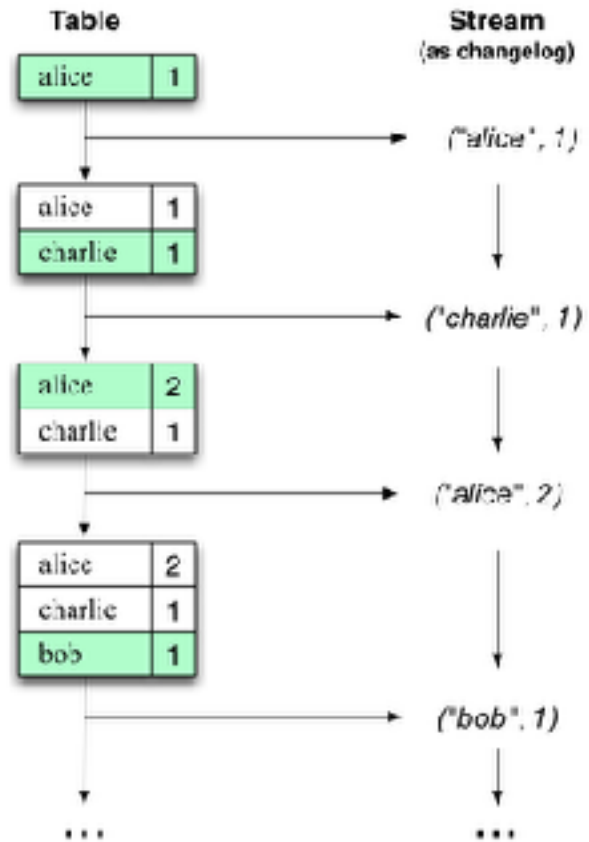
```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));
```

```
KTable<String, Long> articlesPerSite = articles  
    .groupBy((key, value) -> value.path("site").asText(),  
            Serialized.with(strings, json))  
    .count();
```

```
articlesBySite.toStream().print(Printed.toSysOut());
```

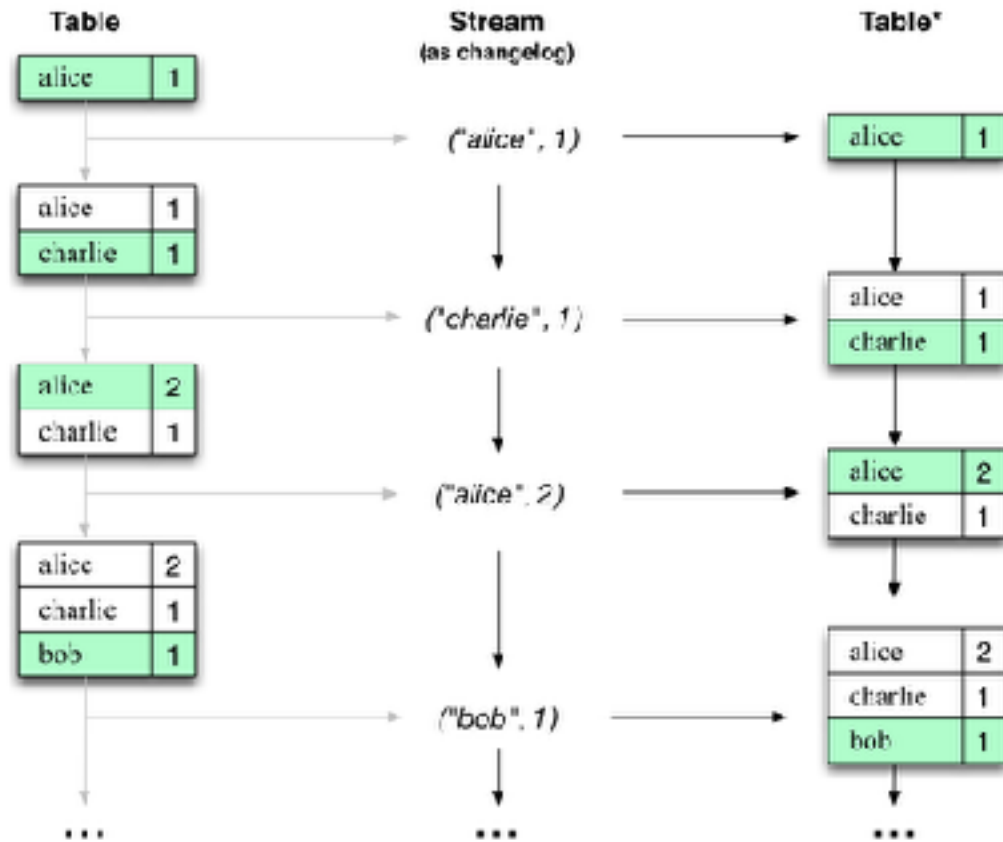
```
foxnews, 2  
cnn, 3  
bbc, 3
```


TABLES VS STREAMS



http://kafka.apache.org/documentation/streams/developer-guide#streams_duality

TABLES VS STREAMS

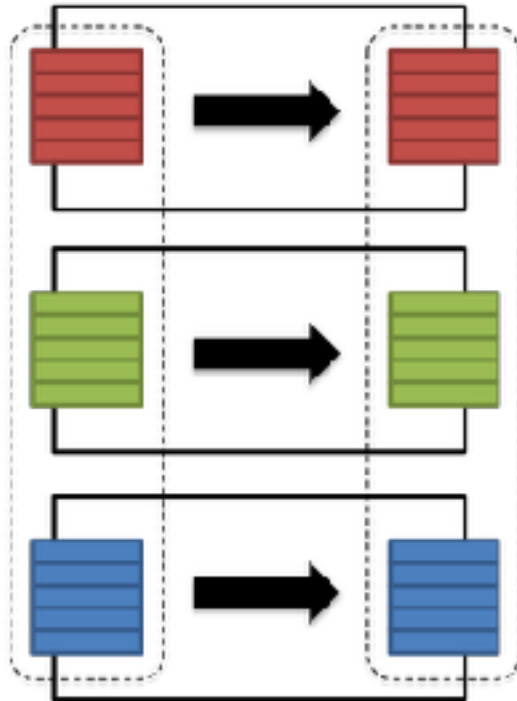


http://kafka.apache.org/documentation/streams/developer-guide#streams_duality

REPARTITIONING

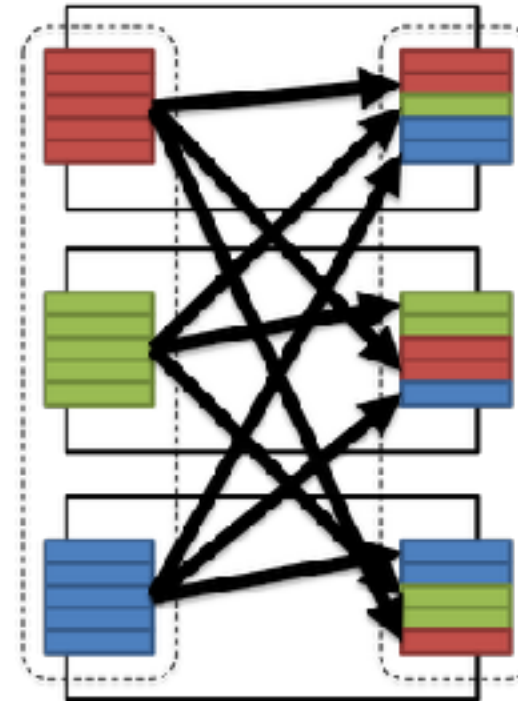
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing



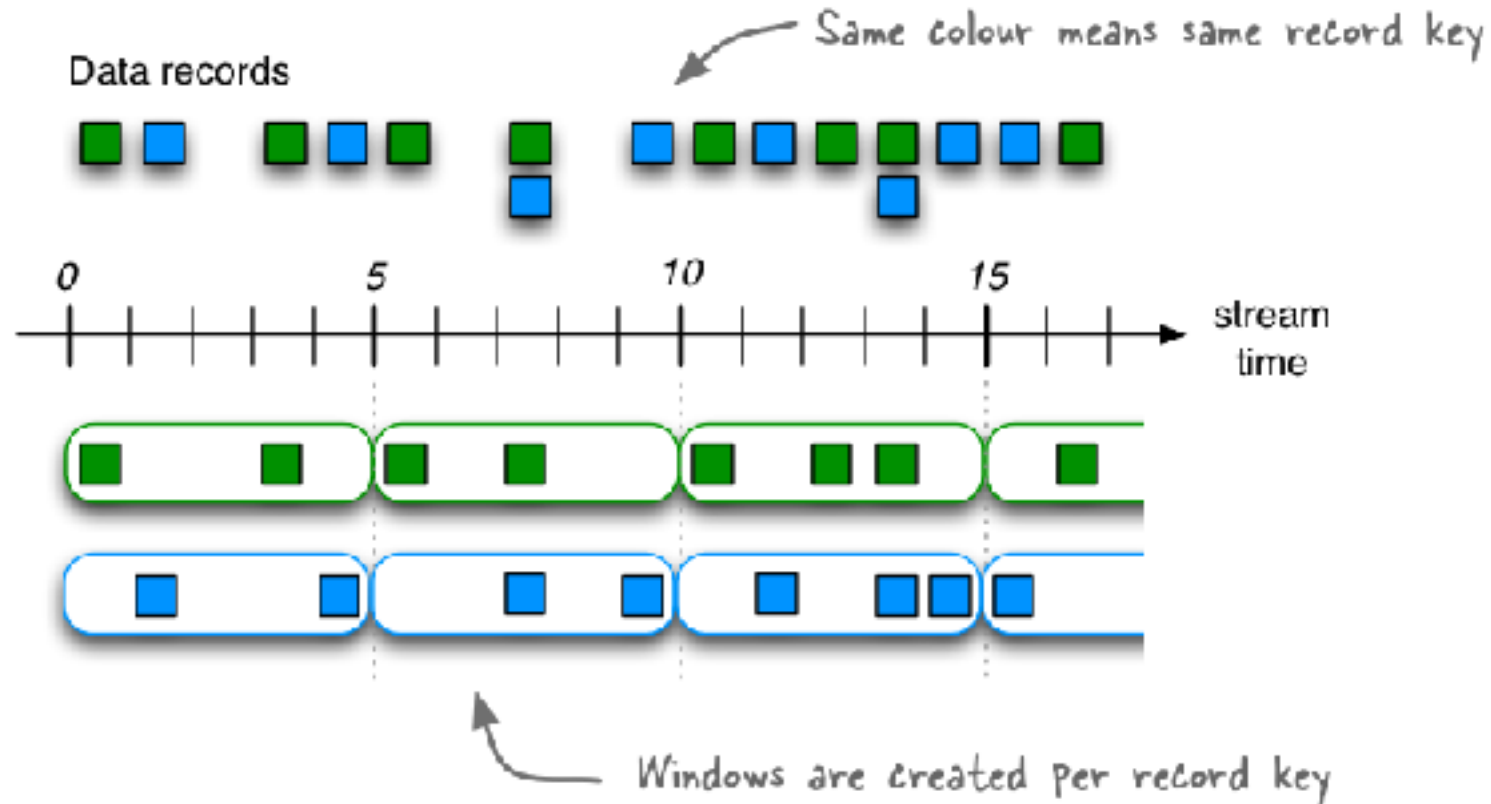
<http://horicky.blogspot.com.es/2013/12/spark-low-latency-massively-parallel.html>

REPARTITIONING

- map, flatMap, selectKey
- groupByKey
- groupBy

WINDOWED AGGREGATIONS

A 5-min Tumbling Window



<https://docs.confluent.io/current/streams/developer-guide.html#windowing>

WINDOWING

```
KStream<String, JsonNode> articles = builder.stream("Articles",  
                                                Consumed.with(strings, json));  
  
KGroupedStream<String, JsonNode> grouped = articles  
    .groupBy((key, value) -> value.path("site").asText(),  
            Serialized.with(strings, json));  
  
KTable<Windowed<String>, Long> articlesPerHour = grouped  
    .windowedBy(TimeWindows.of(TimeUnit.HOURS.toMillis(1)))  
    .count(Materialized.as("articles-per-hour"));
```

EXERCISE 2

- Open `Exercise_2_Aggregations`
- Implement methods to make tests pass
 - Run in IDE or with: `./gradlew test`
 - Tests in `Exercise_2_AggregationTest`
 - Test data in `ClickEvents`
- Should be sufficient with:
 - `map`, `selectKey`, `groupBy`, `groupByKey`, `count`, `reduce`



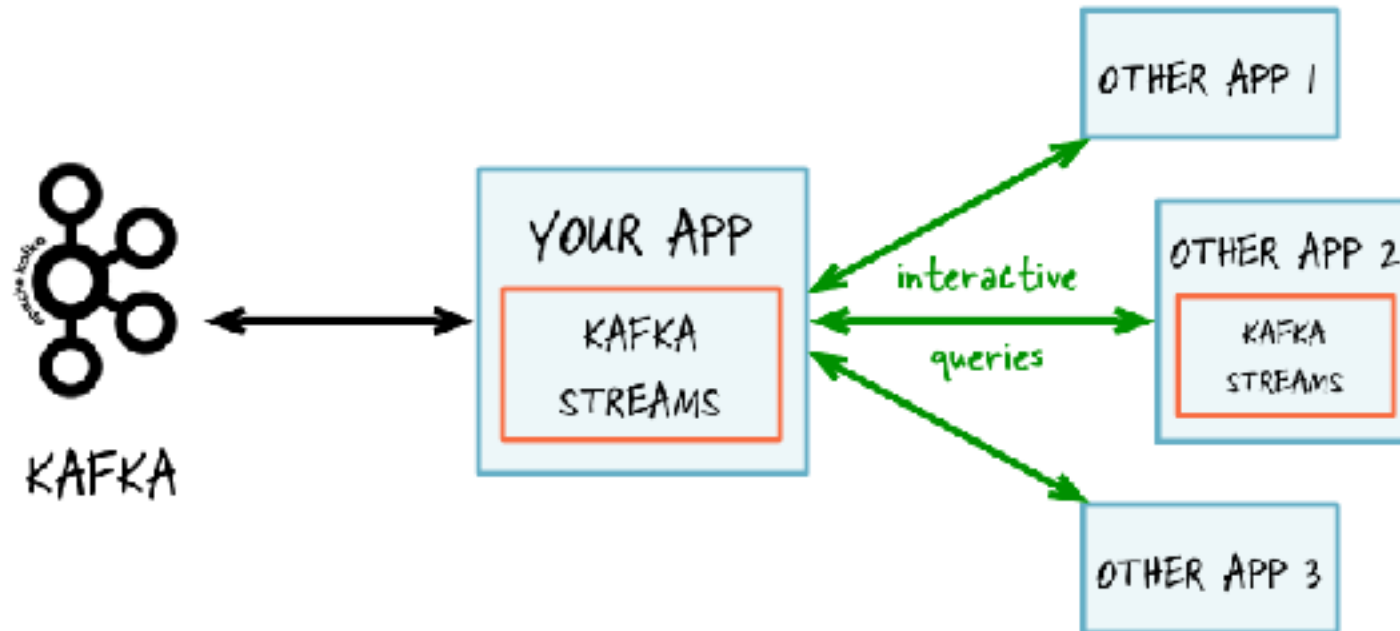
CONNECTING TO THE WORLD

QUERYABLE STATE STORES

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



<https://docs.confluent.io/current/streams/developer-guide.html#streams-developer-guide-interactive-queries>

QUERYING STATE STORES

```
ReadOnlyWindowStore<String, Long> articlesPerHour =  
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());
```

QUERYING STATE STORES

```
ReadOnlyWindowStore<String, Long> articlesPerHour =  
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());  
  
long from = 0L; // Jan 1st 1970  
long to = System.currentTimeMillis();  
WindowStoreIterator<Long> articles = articlesPerHour.fetch("bbc", from, to);
```

QUERYING STATE STORES

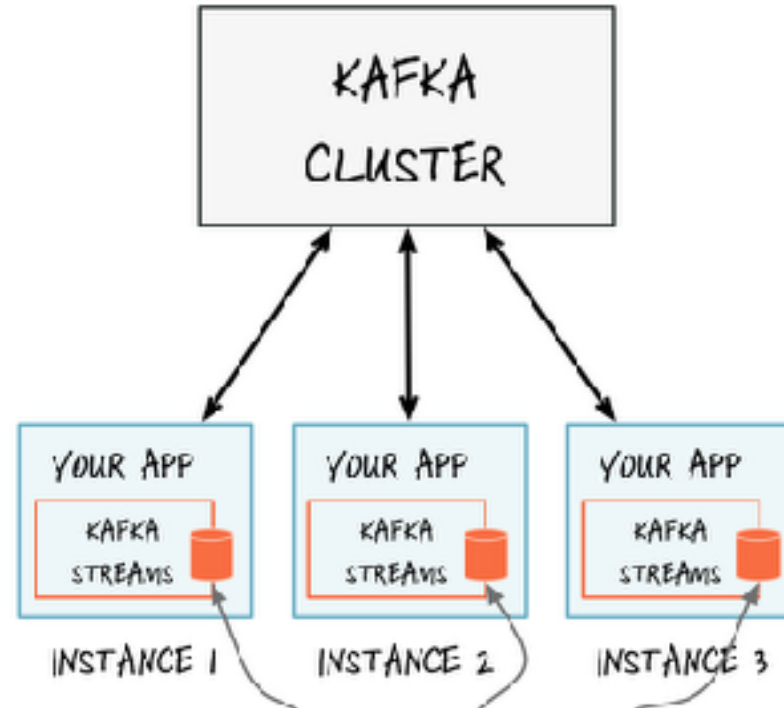
```
ReadOnlyWindowStore<String, Long> articlesPerHour =
    streams.store("articles-per-hour", QueryableStoreTypes.windowStore());

long from = 0L; // Jan 1st 1970
long to = System.currentTimeMillis();
WindowStoreIterator<Long> articles = articlesPerHour.fetch("bbc", from, to);

articles.forEachRemaining(kv -> {
    Instant timestamp = Instant.ofEpochMilli(kv.key);
    System.out.println("BBC published " + kv.value +
        " article(s) in hour " + timestamp);
});
```

BBC published 1 article(s) in hour 2018-02-04T22:00:00Z

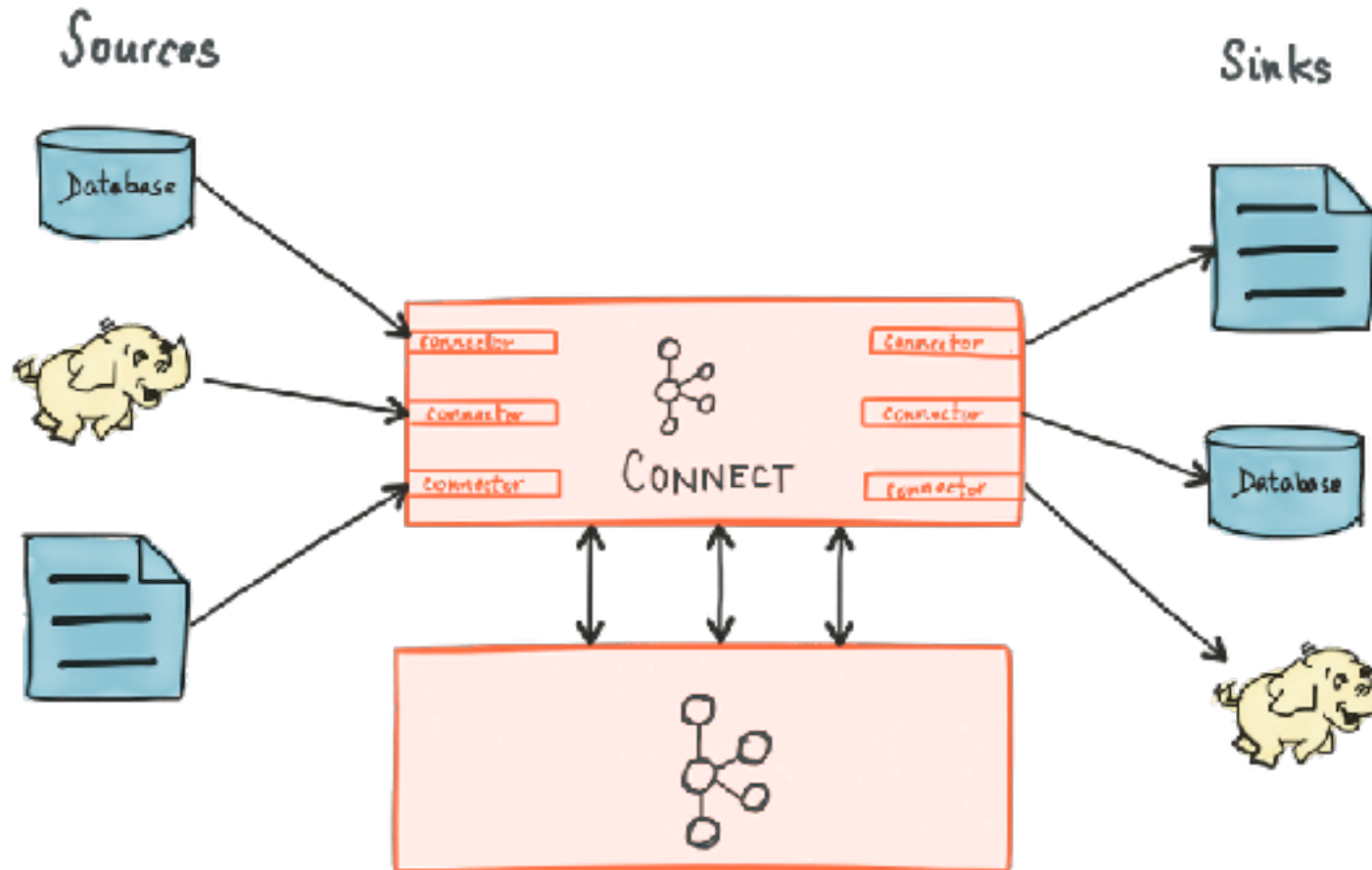
QUERYABLE STATE STORES



Data of the state store "word-count" is split across many local store instances, each of which manages only a partition of the entire state store.

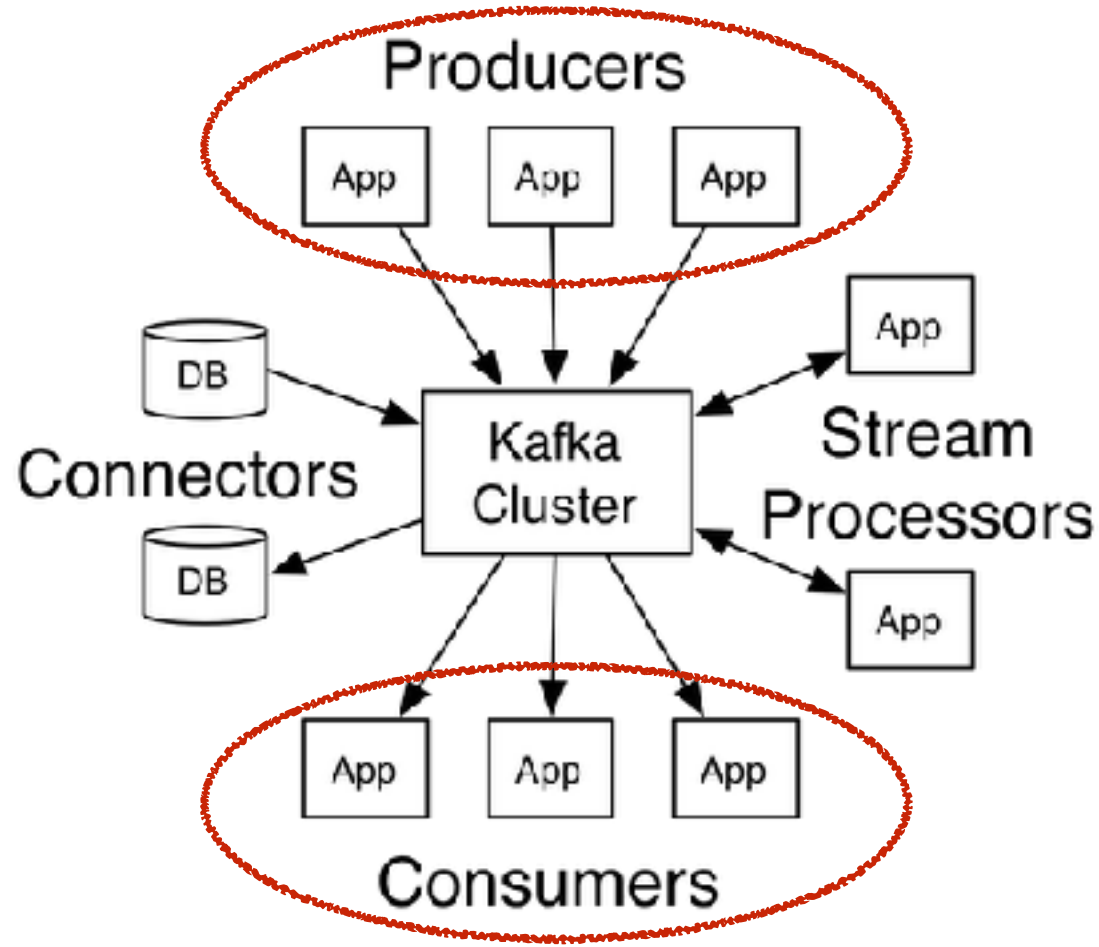
<https://docs.confluent.io/current/streams/developer-guide.html#streams-developer-guide-interactive-queries>

GETTING DATA IN AND OUT



<https://www.confluent.io/blog/announcing-kafka-connect-building-large-scale-low-latency-data-pipelines/>

GETTING DATA IN AND OUT



<http://kafka.apache.org/documentation.html>



JOINS AND ENRICHMENTS

JOINING STREAMS



<https://churchtecharts.org/home/2014/10/5/important-safety-tip-with-dante-dont-cross-the-streams>

JOINS

Join operands	Type	(INNER) JOIN	LEFT JOIN	OUTER JOIN
KStream-to-KStream	Windowed	Supported	Supported	Supported
KTable-to-KTable	Non-windowed	Supported	Supported	Supported
KStream-to-KTable	Non-windowed	Supported	Supported	Not Supported
KStream-to-GlobalKTable	Non-windowed	Supported	Supported	Not Supported
KTable-to-GlobalKTable	N/A	Not Supported	Not Supported	Not Supported

JOINS

```
KStream<String, String> reads = builder.stream("ArticleReads",  
                                             Consumed.with(strings, strings));  
  
KTable<String, JsonNode> users = builder.table("Users",  
                                              Consumed.with(strings, json));
```

JOINS

```
KStream<String, String> reads = builder.stream("ArticleReads",  
                                             Consumed.with(strings, strings));  
  
KTable<String, JsonNode> users = builder.table("Users",  
                                             Consumed.with(strings, json));  
  
KStream<String, JsonNode> readsByCountry = reads  
    .join(users, (article, user) -> user.path("country"));
```

JOINS

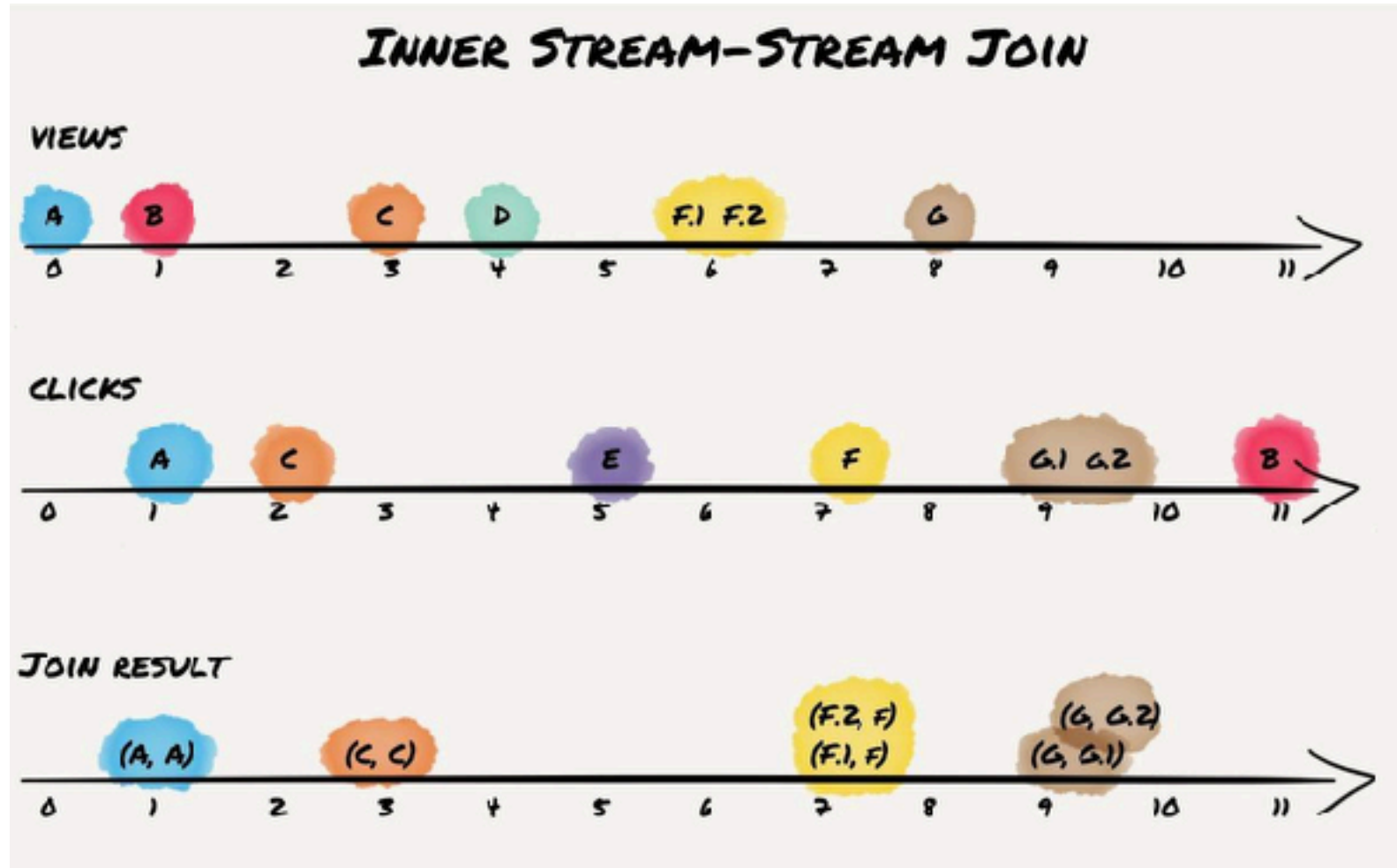
```
KStream<String, String> reads = builder.stream("ArticleReads",  
                                             Consumed.with(strings, strings));
```

```
KTable<String, JsonNode> users = builder.table("Users",  
                                             Consumed.with(strings, json));
```

```
KStream<String, JsonNode> readsByCountry = reads  
    .join(users, (article, user) -> user.path("country"));
```

```
KTable<String, Long> readsPerCountry = readsByCountry  
    .groupBy((userId, country) -> country.asText(),  
           Serialized.with(strings, json))  
    .count();
```

JOINS



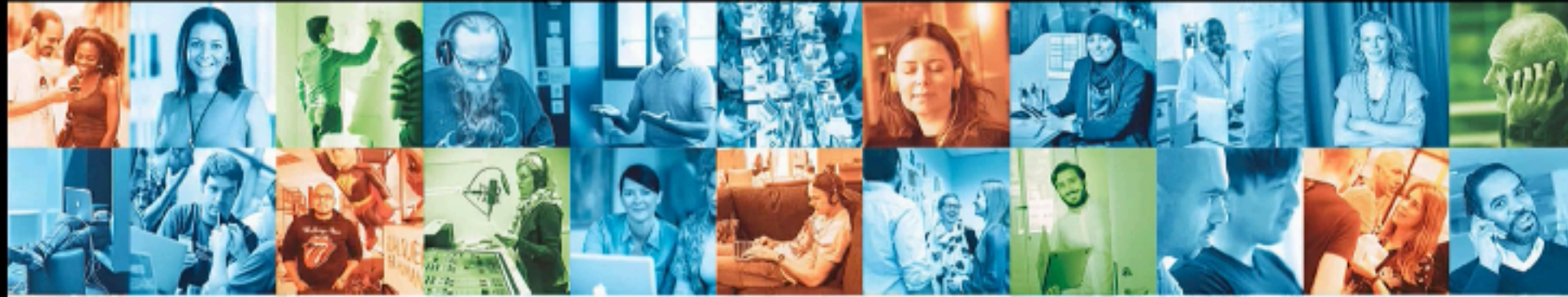
<https://www.confluent.io/blog/crossing-streams-joins-apache-kafka/>

ENRICHMENTS

- Enrich events with information from external sources
- Call 3rd party service in mapValues
- Use Processor API for batching

JavaZone

4



TRANSLATING 700 MILLION EVENTS FROM IP TO COORDINATES EACH DAY USING KAFKA STREAMS

JavaZone 2017

Håkon Åmdal, 14.09.2017





QUESTIONS?

FURTHER READING

- <https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101>
- <https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102>
- <https://www.confluent.io/blog/>
- <https://www.confluent.io/blog/crossing-streams-joins-apache-kafka/>

- <https://docs.confluent.io/current/streams/>
- <http://kafka.apache.org/documentation/streams/>



THANK YOU!

@FREDRIV

FREDRIK.VRAALSEN@SCHIBSTED.COM