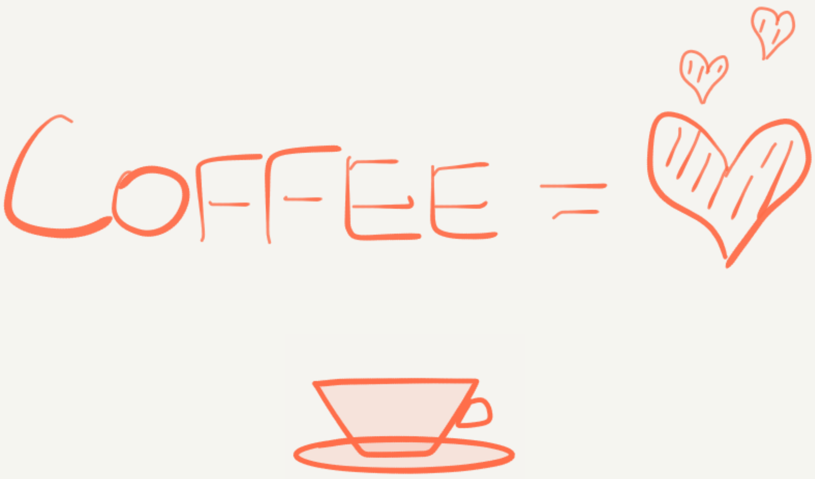
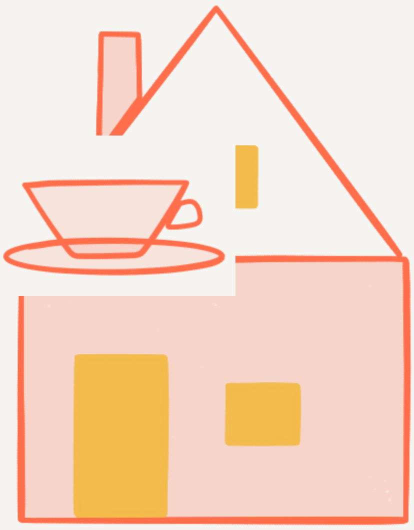
Introducing Kafka Streams

Michael Noll
michael@confluent.io

miguno

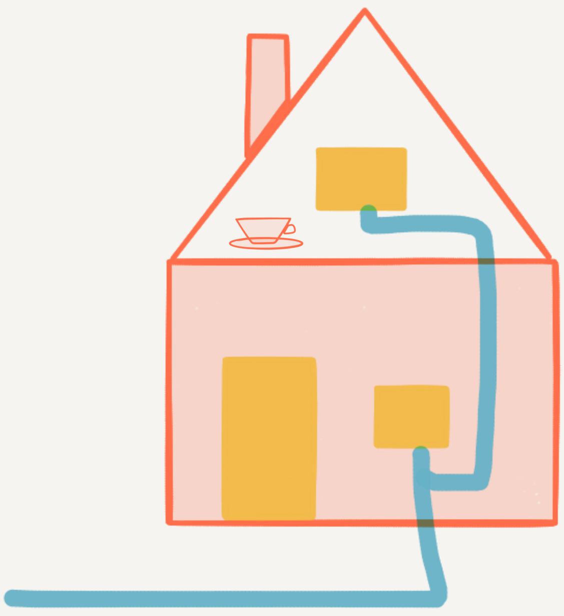
Berlin Buzzwords, June 06, 2016

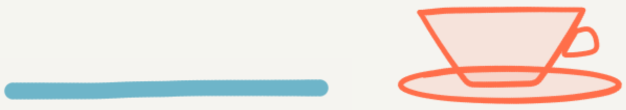

COFFEE =



DATA = WATER

KAFKA = PIPES





KAFKA STREAMS

= COFFEE MACHINE

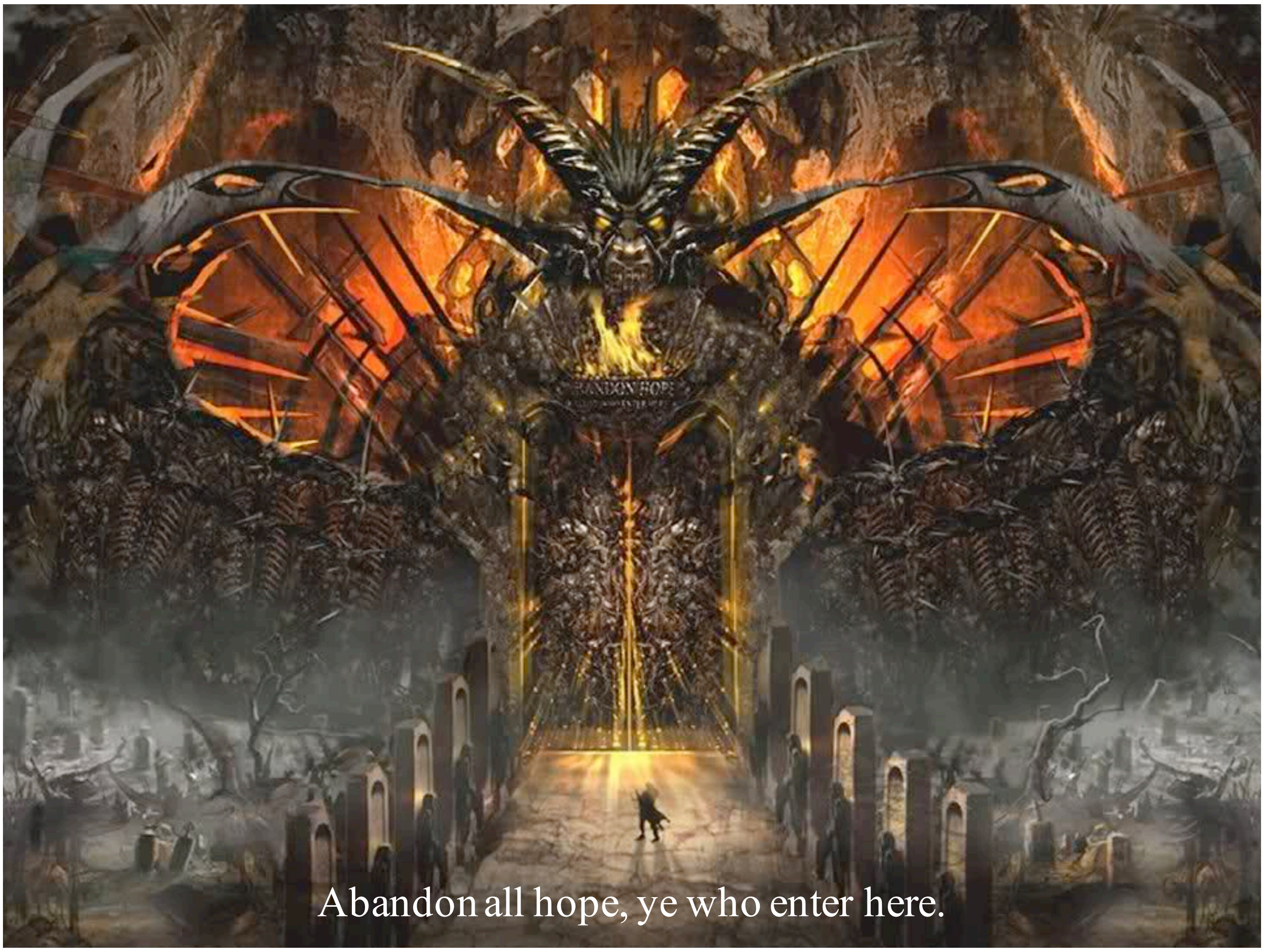
HAPPY USER

Stream processing in Real Life™

…before Kafka Streams

…somewhat exaggerated

…but perhaps not that much

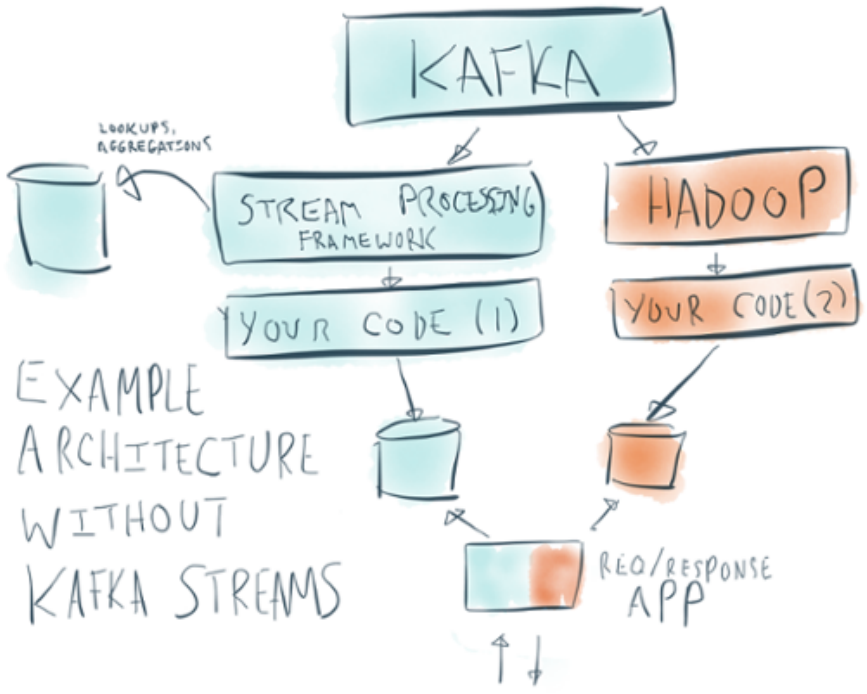


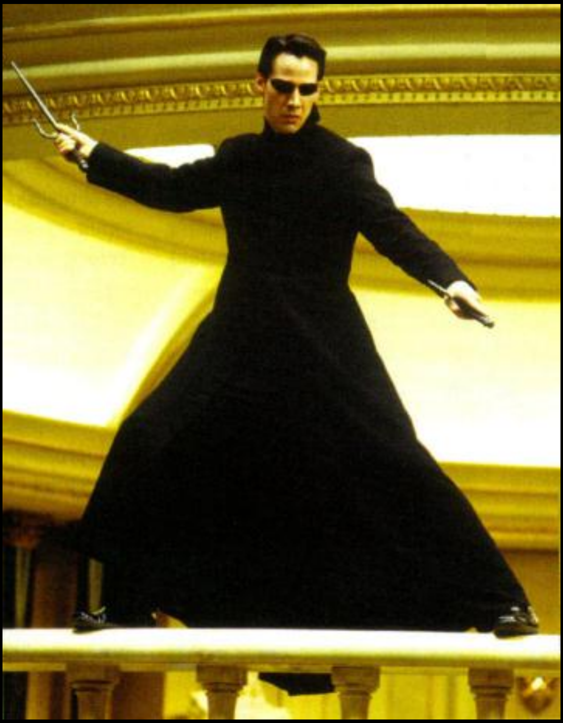
Abandon all hope, ye who enter here.

How did this… (#machines == 1)

```
scala> val input = (1 to 6).toSeq

// Stateless computation
scala> val doubled = input.map(_ * 2)
Vector(2, 4, 6, 8, 10, 12)

// Stateful computation
scala> val sumOfOdds = input.filter(_ % 2 != 0).reduceLeft(_ + _)
res2: Int = 9
```

…turn into stuff like this (#machines > 1)





MAX(VALUE) && MIN(DISTRACTION)

MAKE COMPLEX THINGS

SIMPLE EASY FUN

"DEVELOPER EFFICIENCY"

SCALING HUMANS

(OUR BRAIN CAPACITY DOES NOT DOUBLE EVERY 18 MONTHS ☹)

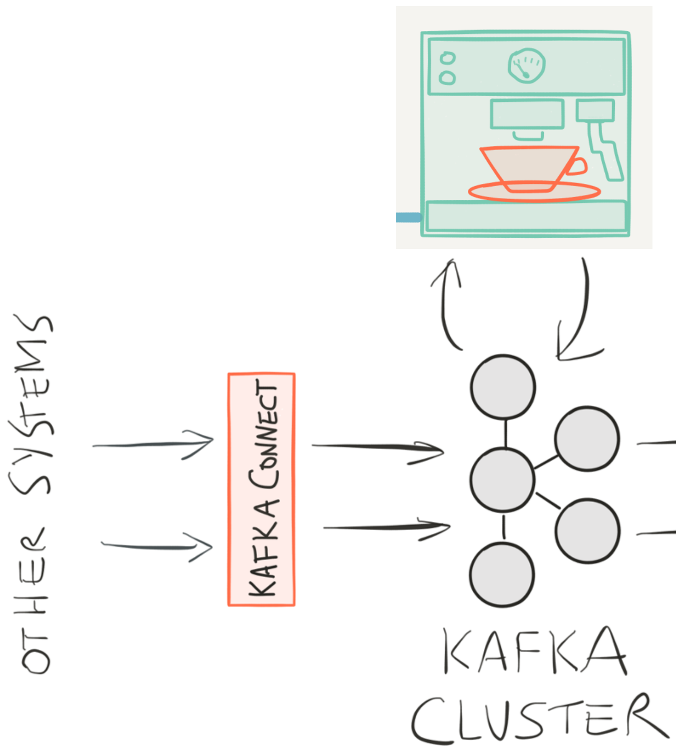
Taken at a session at ApacheCon: Big Data, Hungary, September 2015

Kafka Streams
stream processing made simple

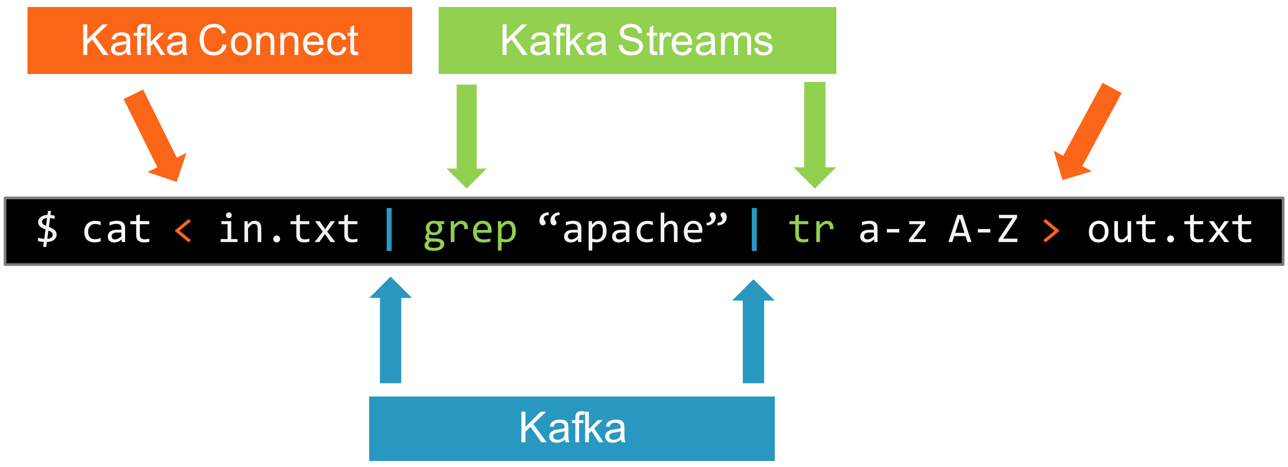
Kafka Streams

- **Powerful yet easy-to use Java library**
- Part of open source Apache Kafka, introduced in v0.10, May 2016
- Source code: https://github.com/apache/kafka/tree/trunk/streams
- Build your own stream processing applications that are
  - highly scalable
  - fault-tolerant
  - distributed
  - stateful
  - able to handle late-arriving, out-of-order data
  - <more on this later>

Kafka Streams



What is Kafka Streams: Unix analogy



```
$ cat < in.txt | grep "apache" | tr a-z A-Z > out.txt
```

What is Kafka Streams: Java analogy

1996	1 core	java.lang
2004	multi-core	java.util.concurrent
2016	multi-machine	java.distributed
		org.apache.kafka.streams

When to use Kafka Streams (as of Kafka 0.10)

Recommended use cases

- **Application Development**
- “Fast Data” apps (small or big data)
- Reactive and stateful applications
- Linear streams
- Event-driven systems
- Continuous transformations
- Continuous queries
- Microservices

Questionable use cases

- **Data Science / Data Engineering**
- “Heavy lifting”
- Data mining
- Non-linear, branching streams (graphs)
- Machine learning, number crunching
- What you’d do in a data warehouse

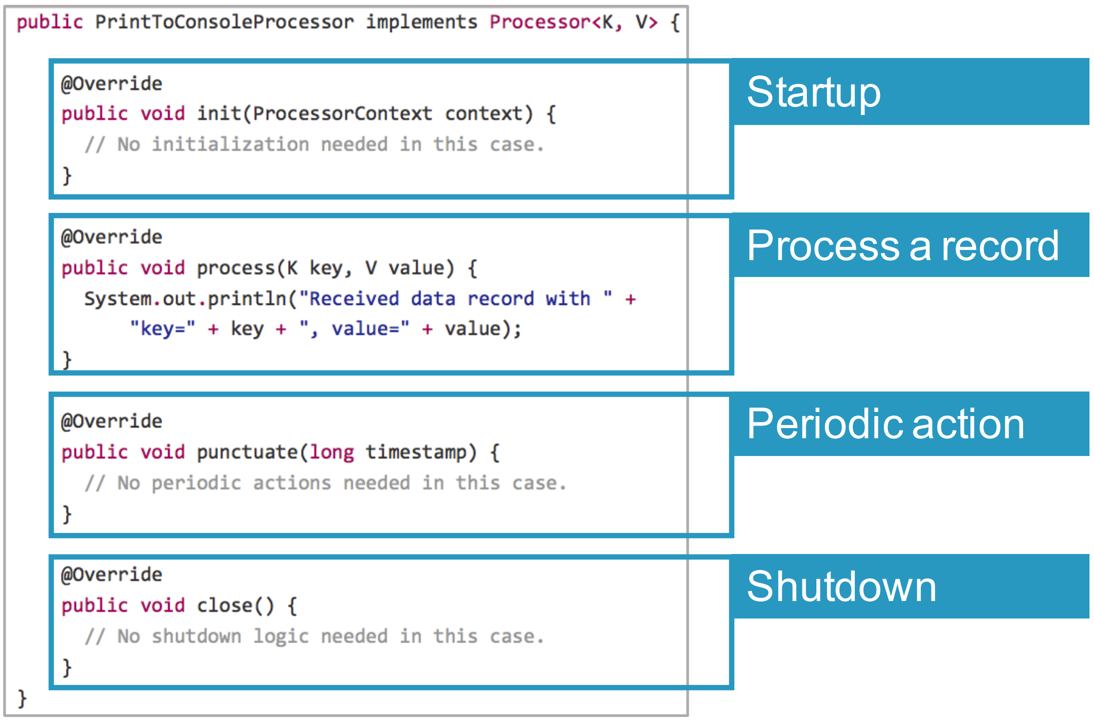
Alright, can you show me some code now? ☺

- API option 1: Kafka Streams DSL (declarative)

```
KStream<Integer, Integer> input = builder.stream(“numbers-topic”);

// Stateless computation
KStream<Integer, Integer> doubled = input.mapValues(v -> v * 2);

// Stateful computation
KTable<Integer, Integer> sumOfOdds = input
    .filter((k,v) -> v % 2 != 0)
    .selectKey((k, v) -> 1)
    .reduceByKey((v1, v2) -> v1 + v2, ”sum-of-odds");
```

Alright, can you show me some code now? ☺

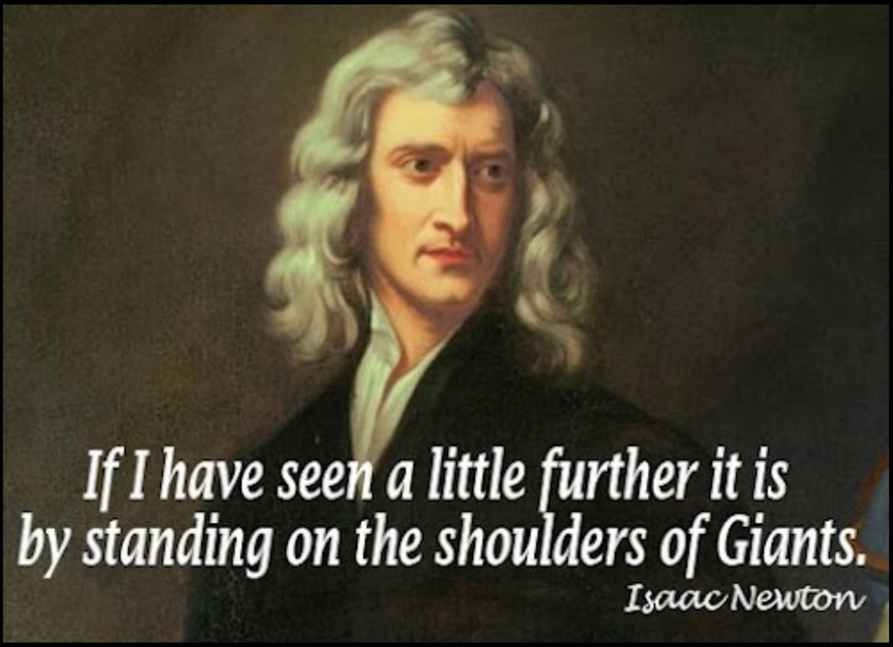
- API option 2: low-level Processor API (imperative)



```
public PrintToConsoleProcessor implements Processor<K, V> {

    @Override
    public void init(ProcessorContext context) {
      // No initialization needed in this case.
    }

    @Override
    public void process(K key, V value) {
      System.out.println("Received data record with " +
          "key=" + key + ", value=" + value);
    }

    @Override
    public void punctuate(long timestamp) {
      // No periodic actions needed in this case.
    }

    @Override
    public void close() {
      // No shutdown logic needed in this case.
    }
}
```

API, coding
MAKE COMPLEX THINGS
SIMPLE EASY FUN
Operations, debugging, …

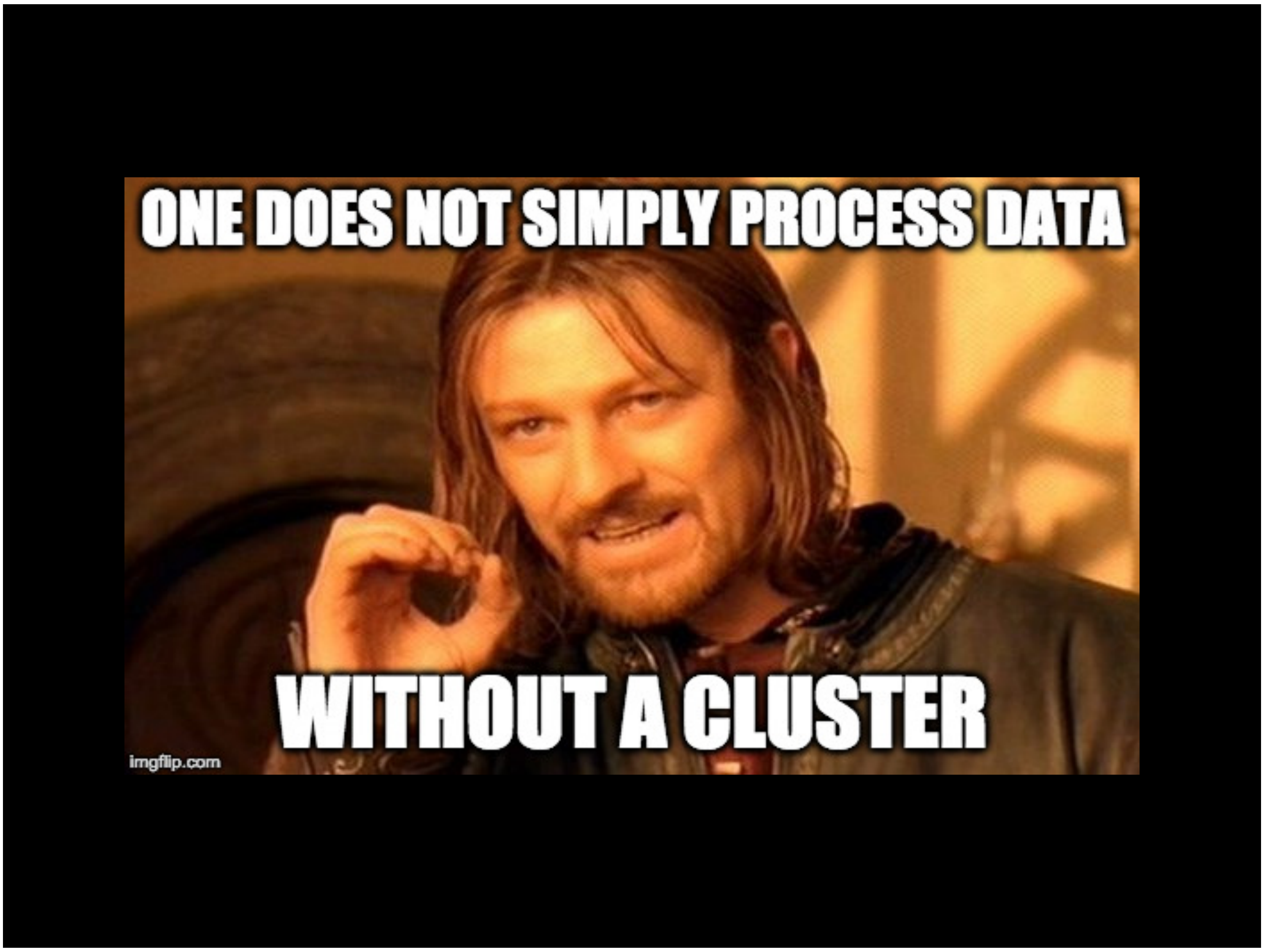
Kafka Streams outsources hard problems to Kafka


If I have seen a little further it is
by standing on the shoulders of Giants.
Isaac Newton

How do I install Kafka Streams?

- There is and there should be no “install”.
- It’s a library. Add it to your app like any other library.

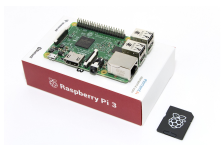
```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-streams</artifactId>
  <version>0.10.0.0</version>
</dependency>
```


ONE DOES NOT SIMPLY PROCESS DATA
WITHOUT A CLUSTER
imgflip.com

Do I need to install a CLUSTER to run my apps?

- No, you don't. Kafka Streams allows you to stay lean and lightweight.
- Unlearn bad habits: "do cool stuff with data != must have cluster"

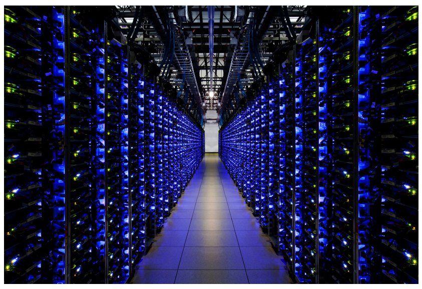
Ok.



Ok.

Ok.

Ok.

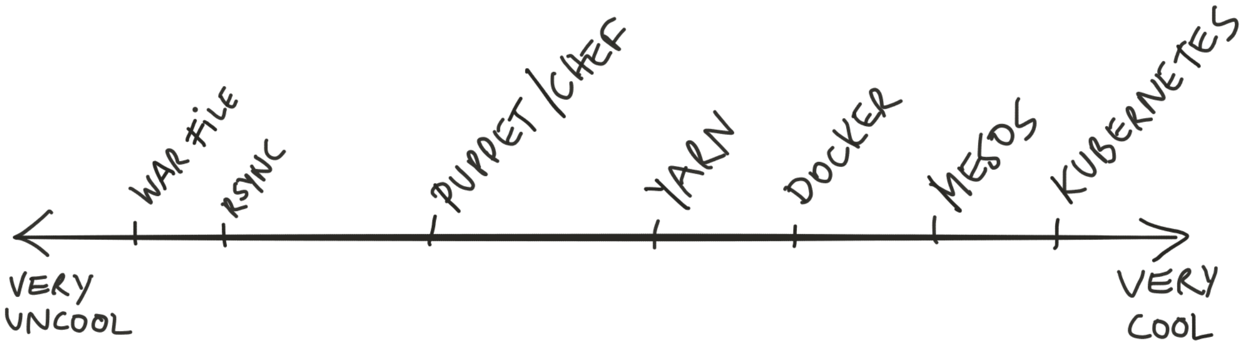


How do I package and deploy my apps? How do I …?

WORKS ON DATA
imgflip.com

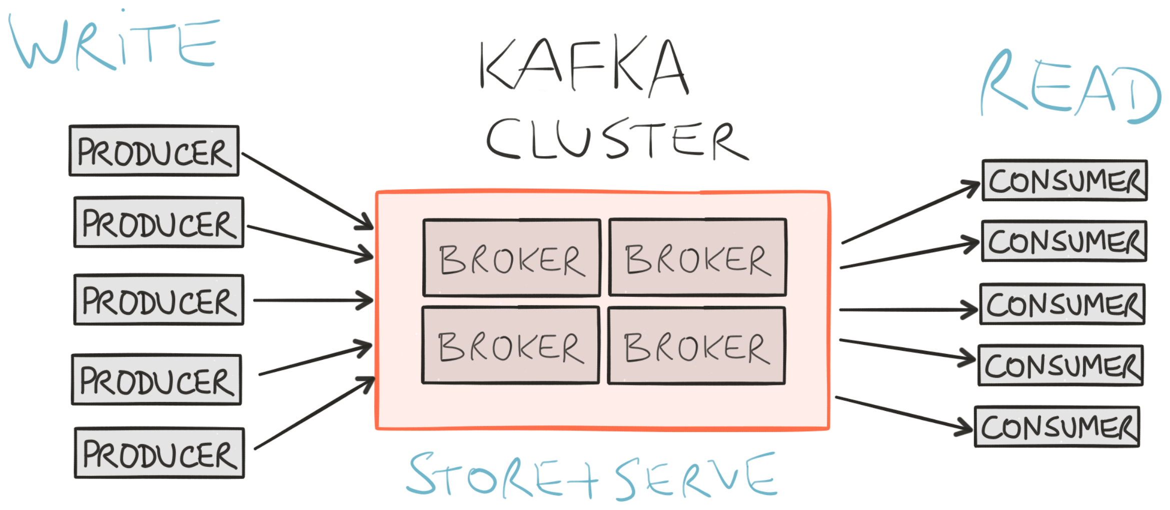
How do I package and deploy my apps? How do I …?

- Whatever works for you. Stick to what you/your company think is the best way.
  - Why? Because an app that uses Kafka Streams is…a normal Java app.
- Your Ops/SRE/InfoSec teams may finally start to ~~love~~ not hate you.

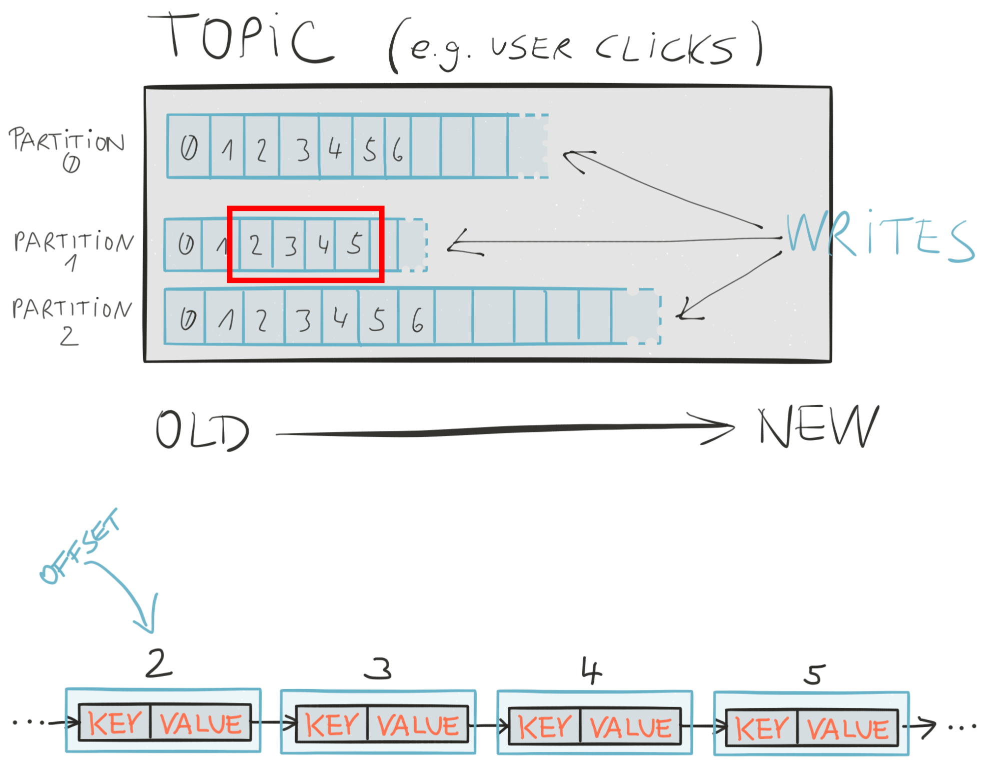


Kafka concepts

Kafka concepts

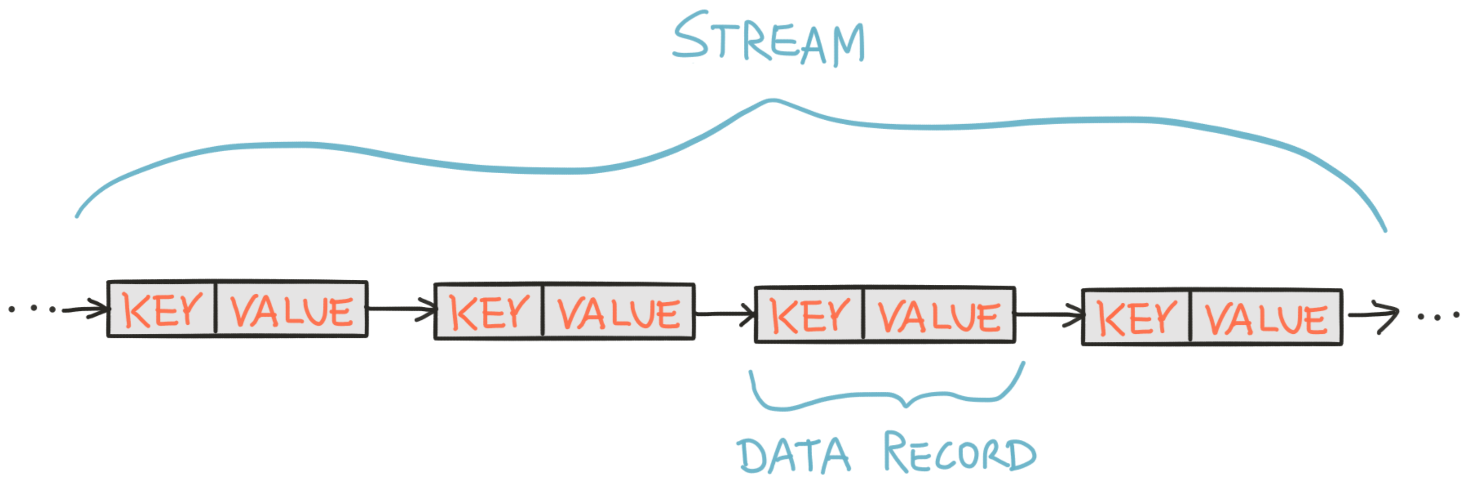


Kafka concepts

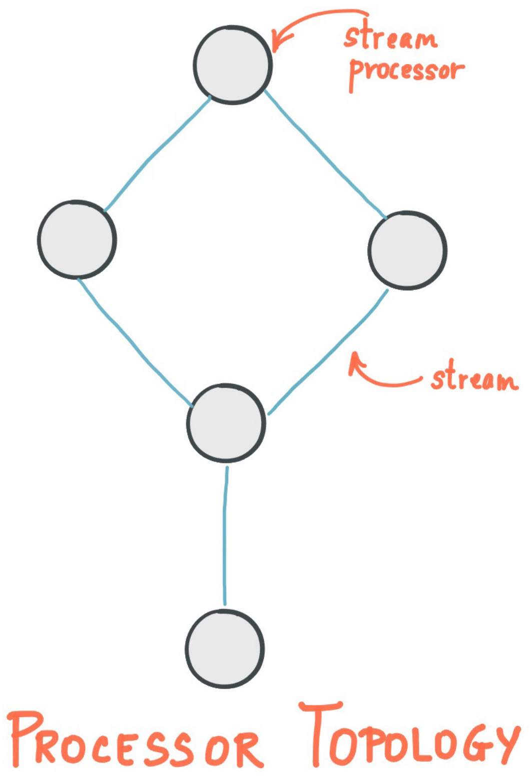


Kafka Streams concepts

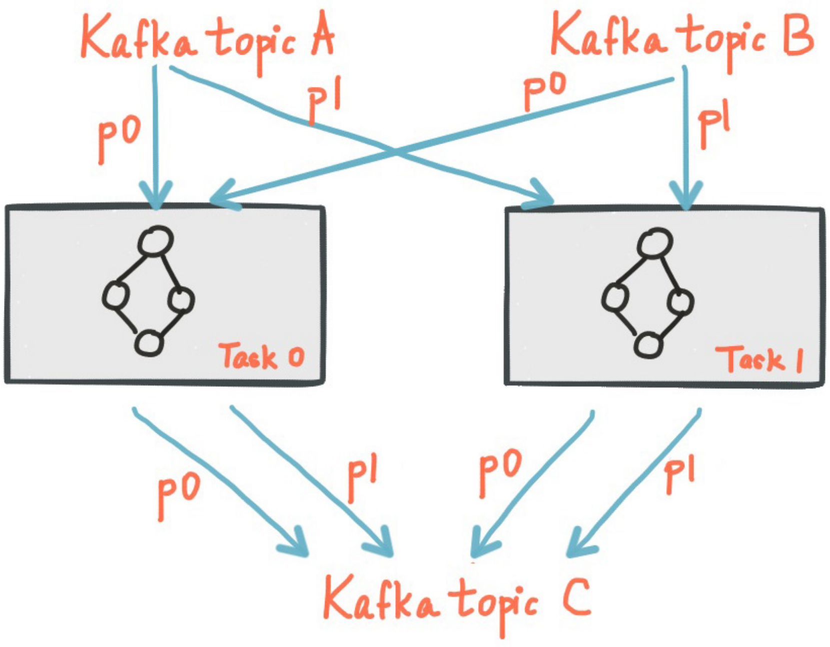
Stream



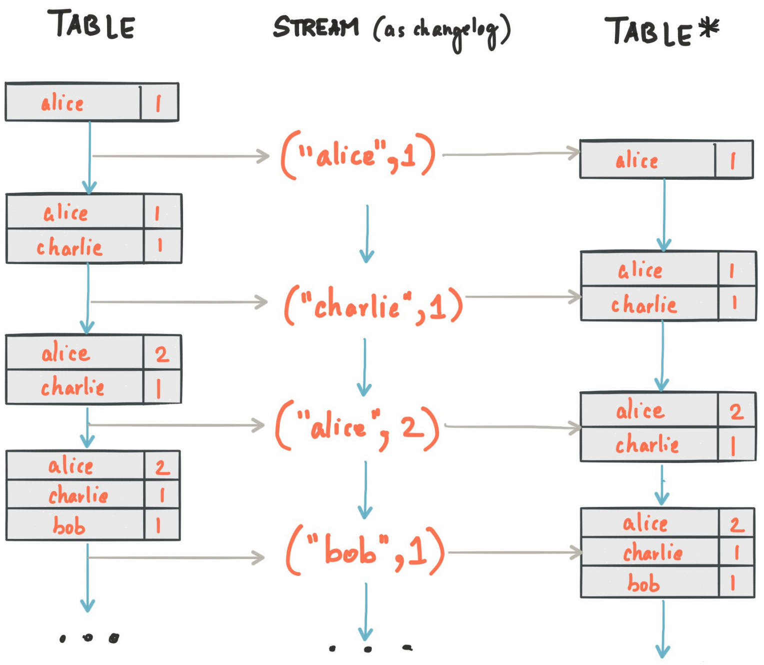
Processor topology



Stream partitions and stream tasks



Streams meet Tables



http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple
http://docs.confluent.io/3.0.0/streams/concepts.html#duality-of-streams-and-tables

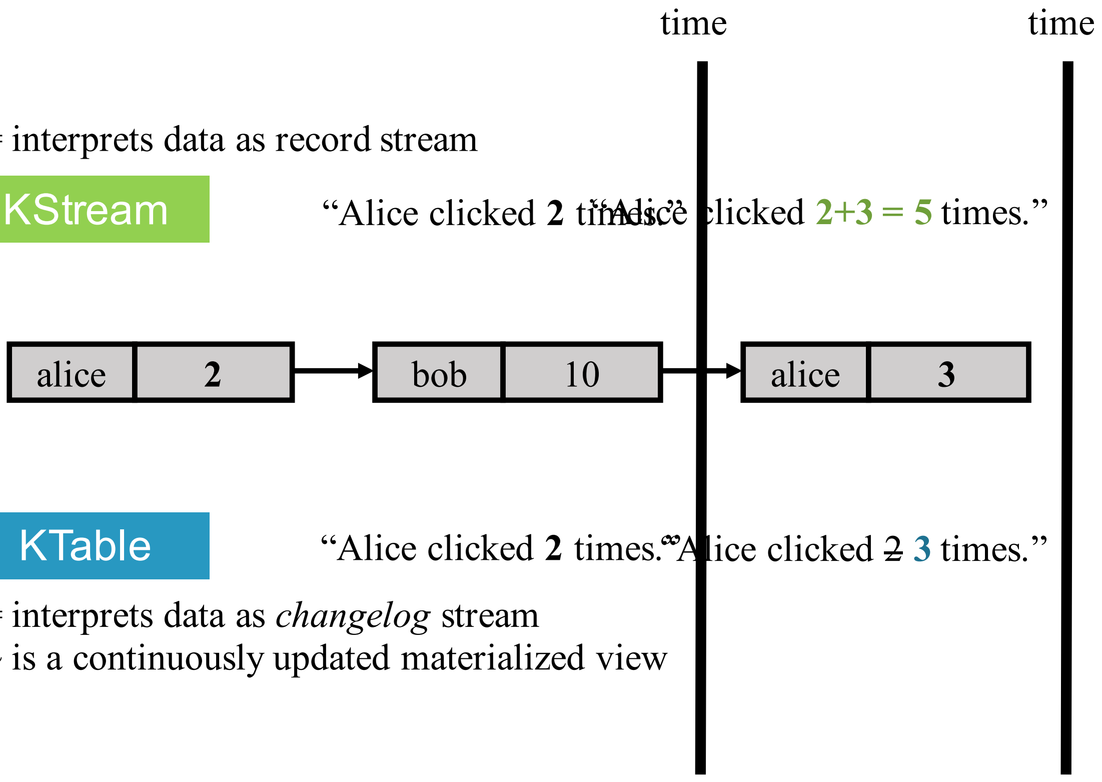
Streams meet Tables – in the Kafka Streams DSL

= interprets data as record stream

KStream

"Alice clicked **2** times."

"Alice clicked **2+3 = 5** times."



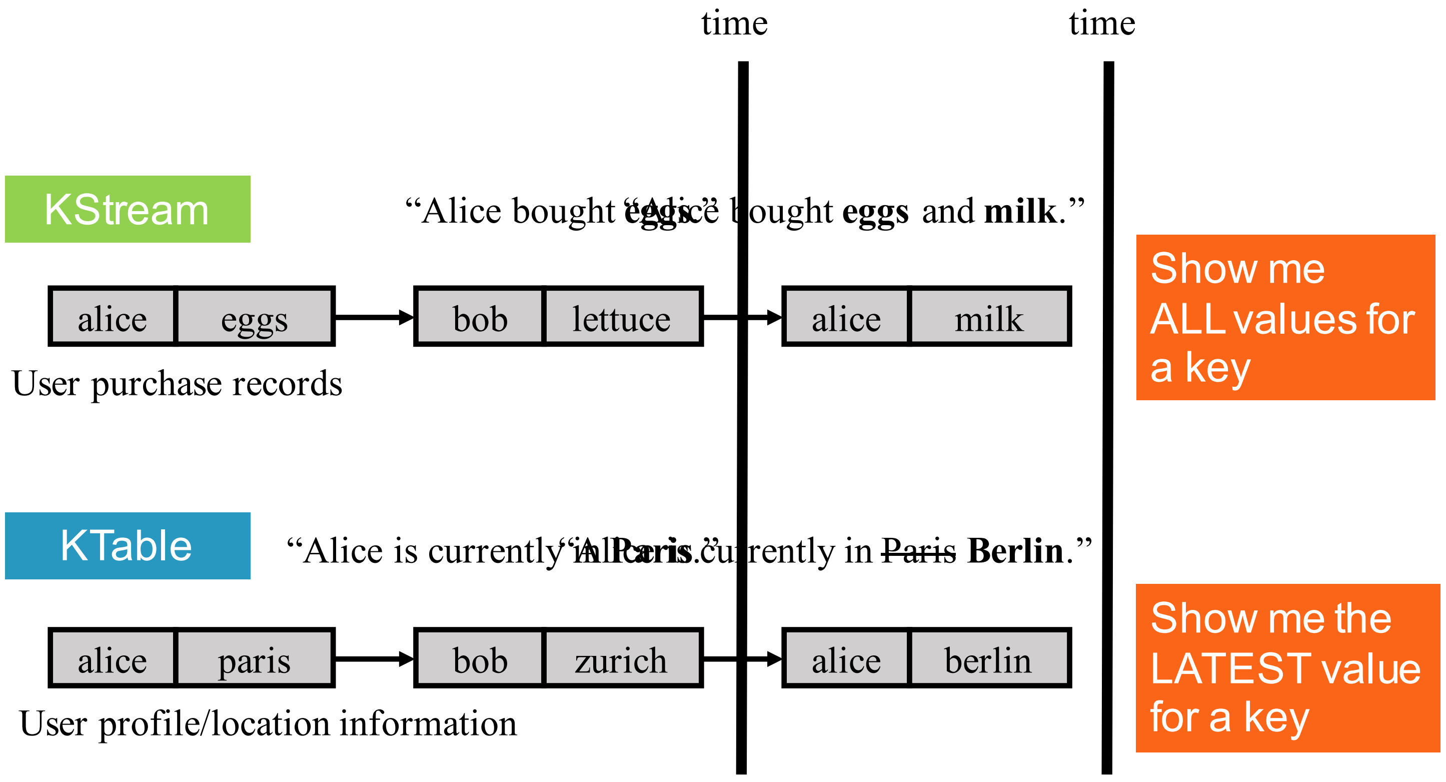
KTable

"Alice clicked **2** times."

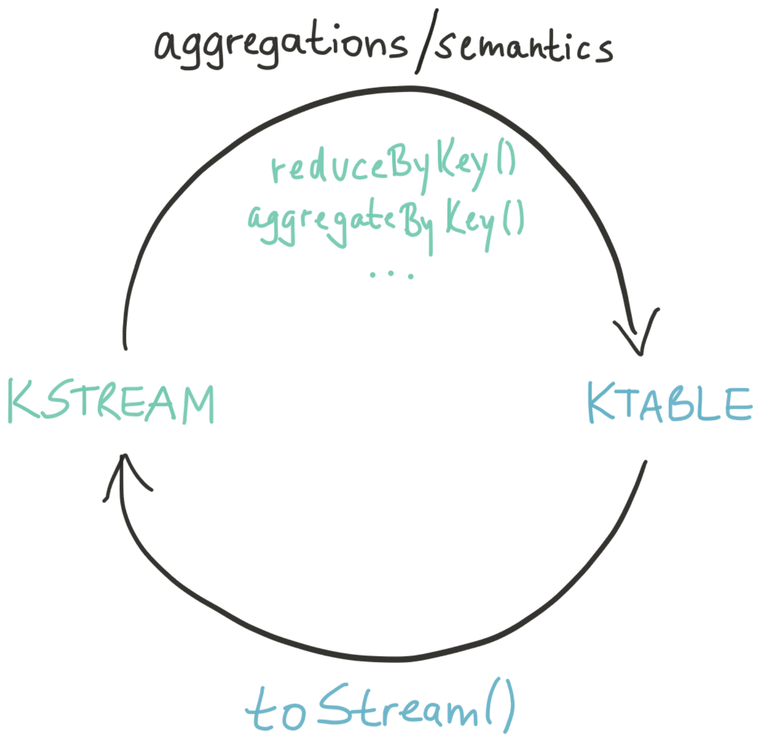
"Alice clicked ~~2~~ **3** times."

= interprets data as *changelog* stream
~ is a continuously updated materialized view

Streams meet Tables – in the Kafka Streams DSL


time
time
KStream
“Alice bought **eggs**.”
“Alice bought **eggs** and **milk**.”
alice eggs
bob lettuce
alice milk
User purchase records
Show me ALL values for a key
KTable
“Alice is currently in **Paris**.”
“Alice is currently in ~~Paris~~ **Berlin**.”
alice paris
bob zurich
alice berlin
User profile/location information
Show me the LATEST value for a key

Streams meet Tables – in the Kafka Streams DSL



Streams meet Tables – in the Kafka Streams DSL

- JOIN example: compute user clicks by region via KStream.leftJoin(KTable)

```
// e.g. "alice" -> 13L
KStream<String, Long> userClicksStream = ...;

// e.g. "alice" -> "europe"
KTable<String, String> userRegionsTable = ...;
```

Streams meet Tables – in the Kafka Streams DSL

- JOIN example: compute user clicks by region via KStream.leftJoin(KTable)

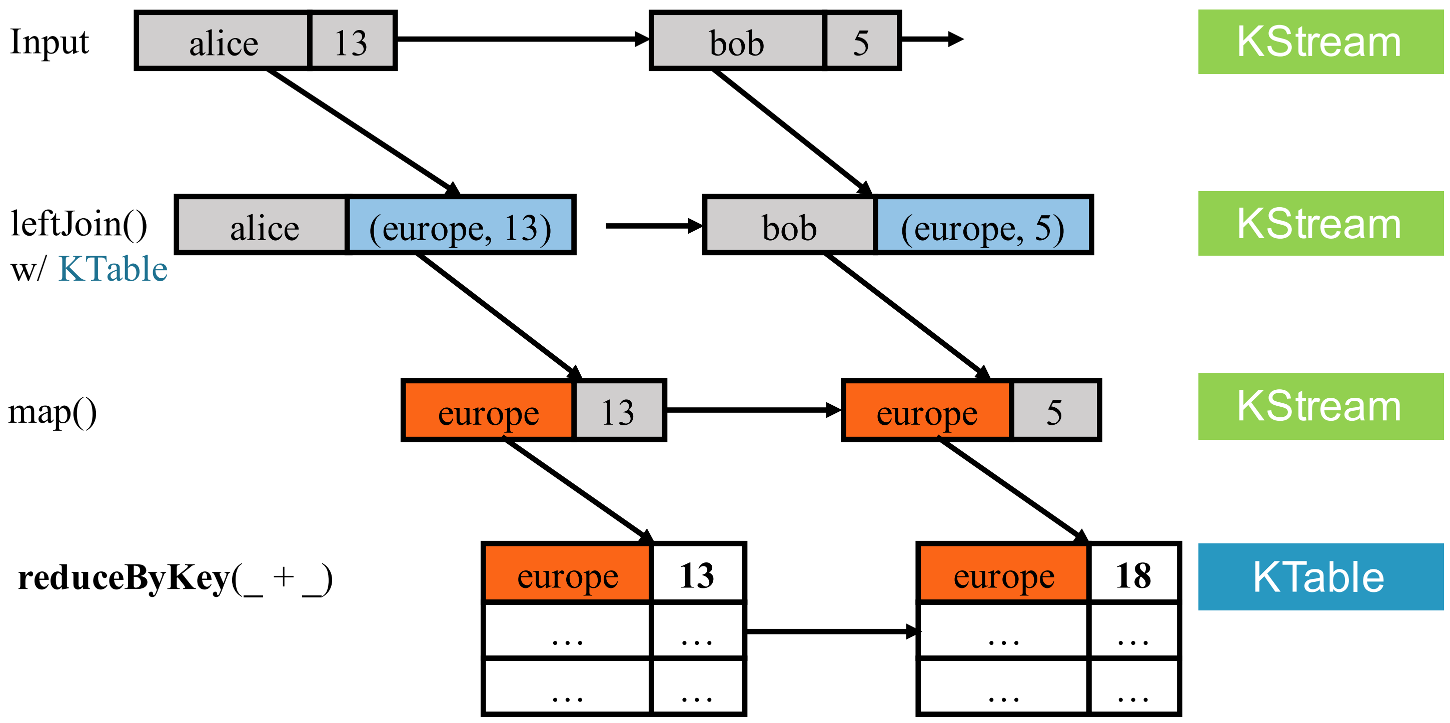
```
// e.g. "alice" -> 13L
KStream<String, Long> userClicksStream = ...;

// e.g. "alice" -> "europe"
KTable<String, String> userRegionsTable = ...;

// Compute the number of user clicks per region, e.g. "europe" -> 13L
KTable<String, Long> clicksPerRegion = userClicksStream
    .leftJoin(userRegionsTable, (clicks, region) -> new RegionWithClicks(region == null ? "UNKNOWN" : region, clicks))
    .map((user, regionWithClicks) -> new KeyValue<>(regionWithClicks.region(), regionWithClicks.clicks()))
    .reduceByKey(
        (firstClicks, secondClicks) -> firstClicks + secondClicks,
        stringSerde, longSerde, "ClicksPerRegion");
```

Even simpler in Scala because, unlike Java, it natively supports tuples:

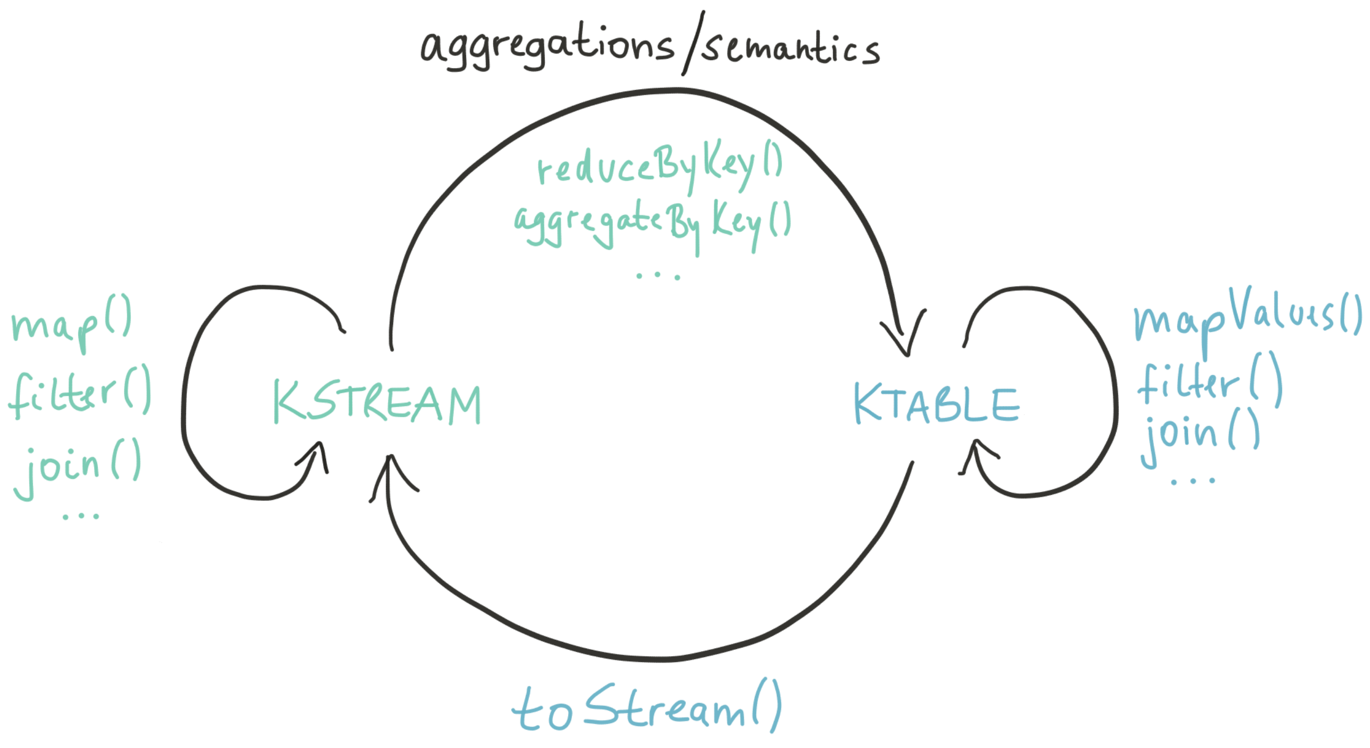
```
.leftJoin(userRegionsTable, (clicks: Long, region: String) => (if (region == null) "UNKNOWN" else region, clicks))
.map((user: String, regionWithClicks: (String, Long)) => new KeyValue(regionWithClicks._1, regionWithClicks._2))
```

Streams meet Tables – in the Kafka Streams DSL

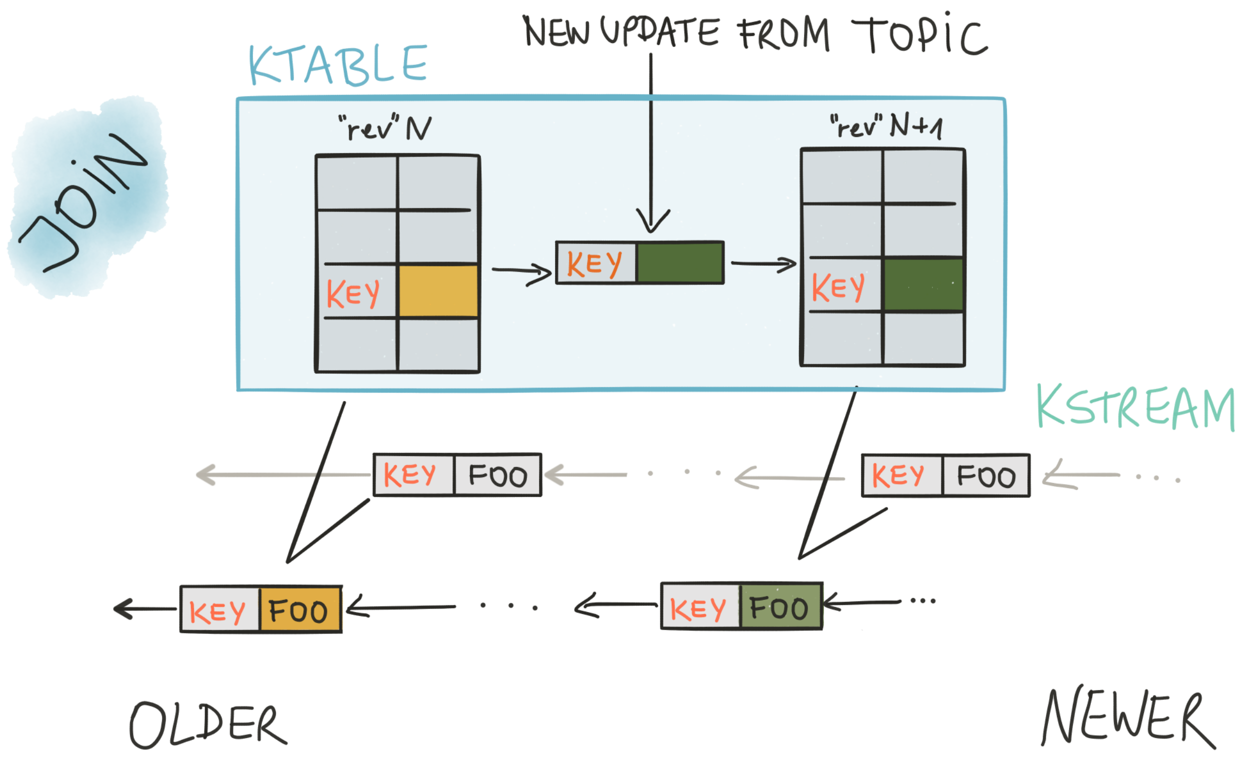
- JOIN example: compute user clicks by region via KStream.leftJoin(KTable)



Streams meet Tables – in the Kafka Streams DSL



Streams meet Tables – in the Kafka Streams DSL



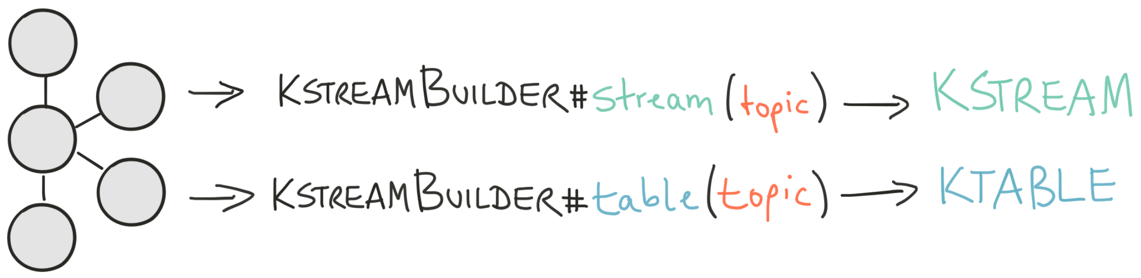
Kafka Streams key features

Key features in 0.10

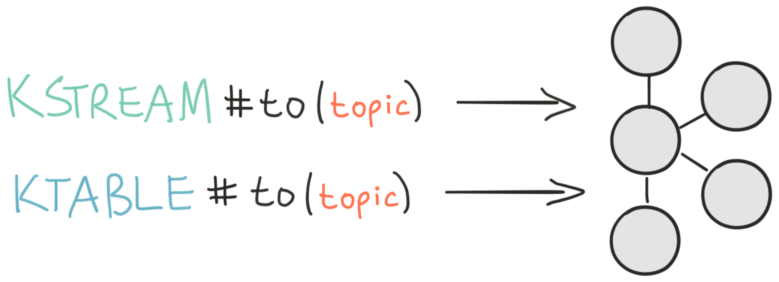
- **Native, 100%-compatible Kafka integration**
  - **Also inherits Kafka's security model, e.g. to encrypt data-in-transit**
  - **Uses Kafka as its internal messaging layer, too**

Native Kafka integration

- Reading data from Kafka

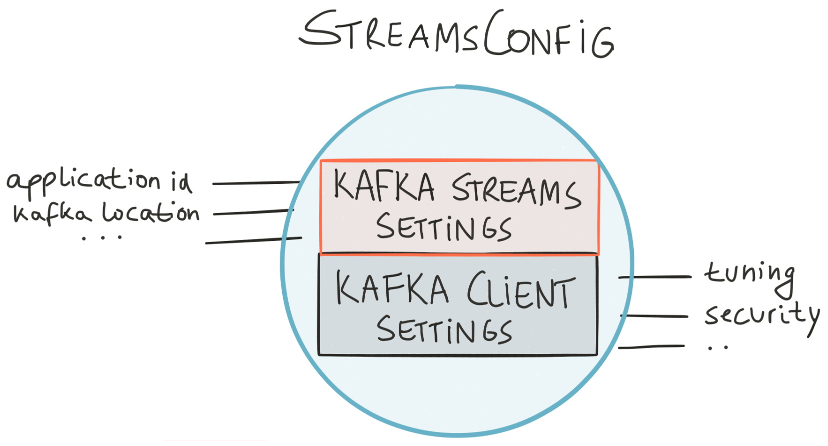

KSTREAMBUILDER#stream(topic) → KSTREAM
KSTREAMBUILDER#table(topic) → KTABLE

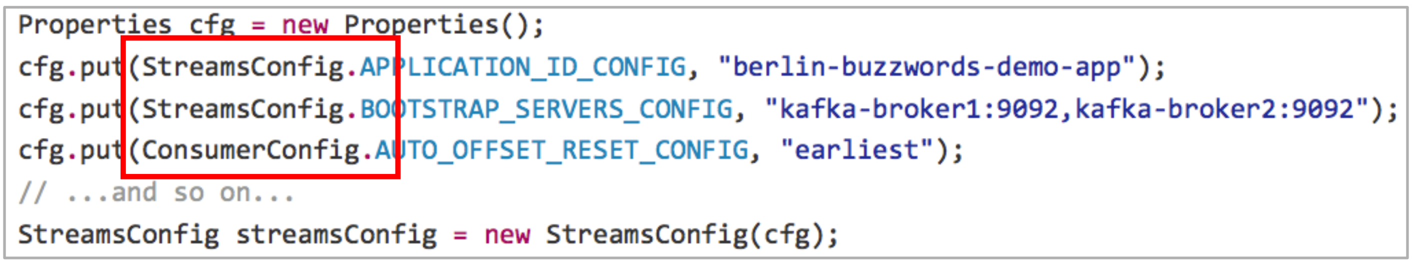
- Writing data to Kafka



Native Kafka integration

- You can configure both Kafka Streams plus the underlying Kafka clients



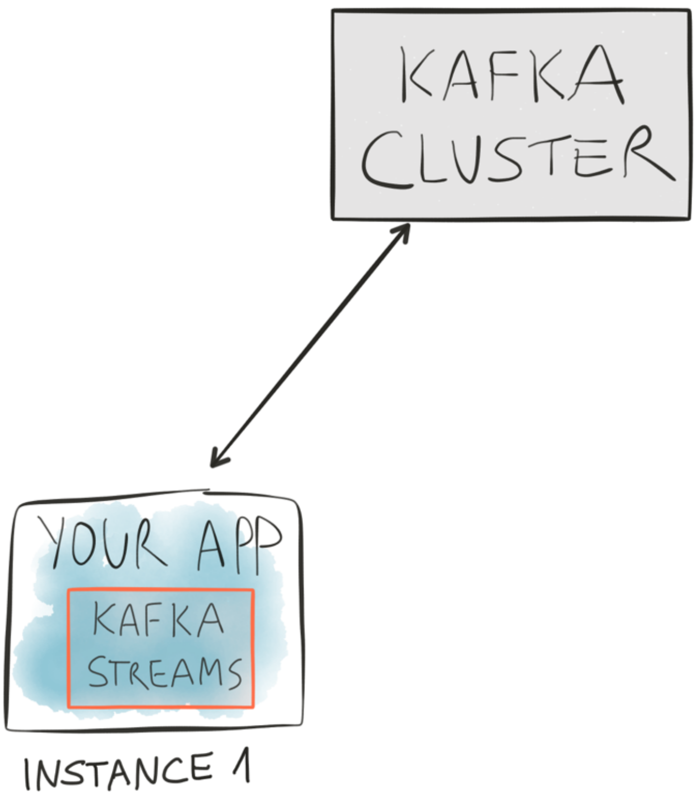


```
Properties cfg = new Properties();
cfg.put(StreamsConfig.APPLICATION_ID_CONFIG, "berlin-buzzwords-demo-app");
cfg.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092,kafka-broker2:9092");
cfg.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
// ...and so on...
StreamsConfig streamsConfig = new StreamsConfig(cfg);
```

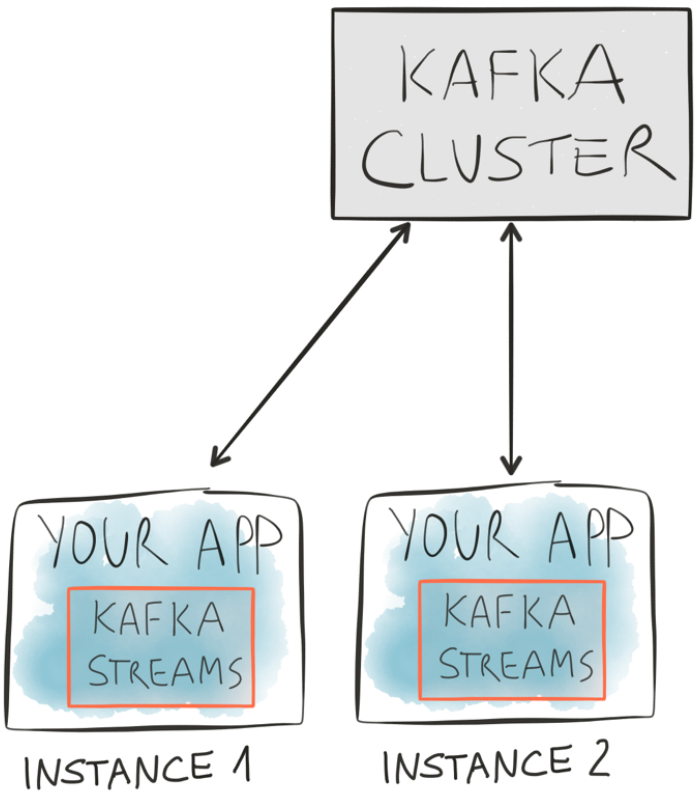
Key features in 0.10

- Native, 100%-compatible Kafka integration
  - Also inherits Kafka's security model, e.g. to encrypt data-in-transit
  - Uses Kafka as its internal messaging layer, too
- **Highly scalable**
- **Fault-tolerant**
- **Elastic**

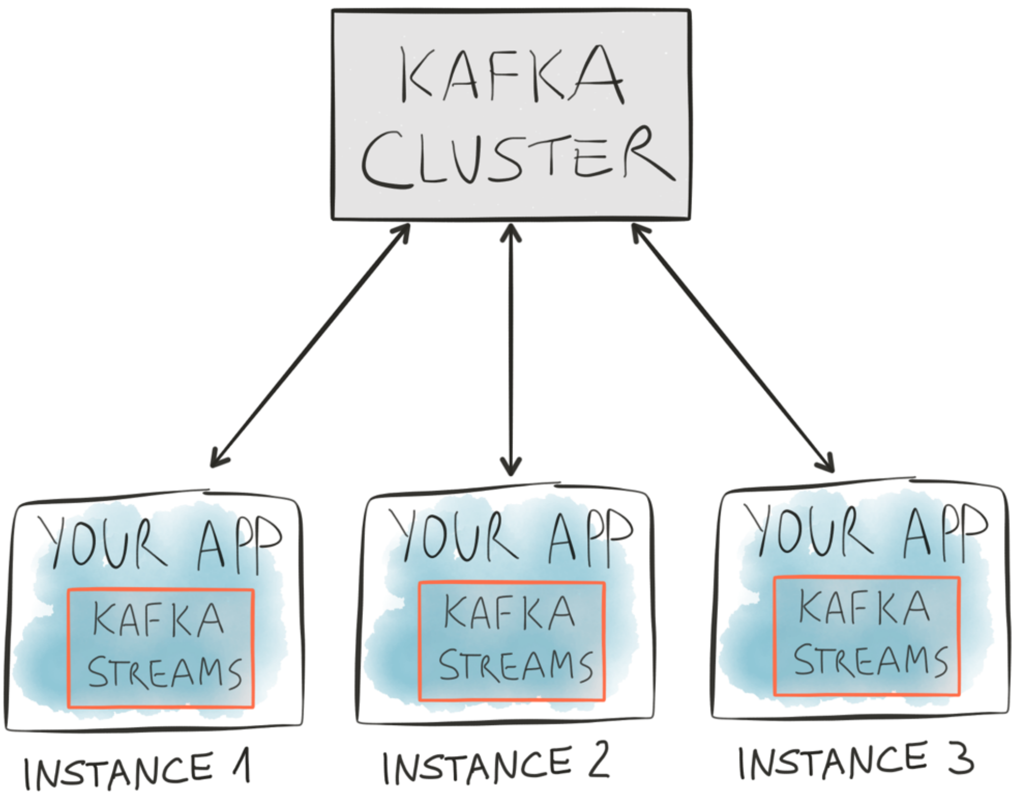
Execution model



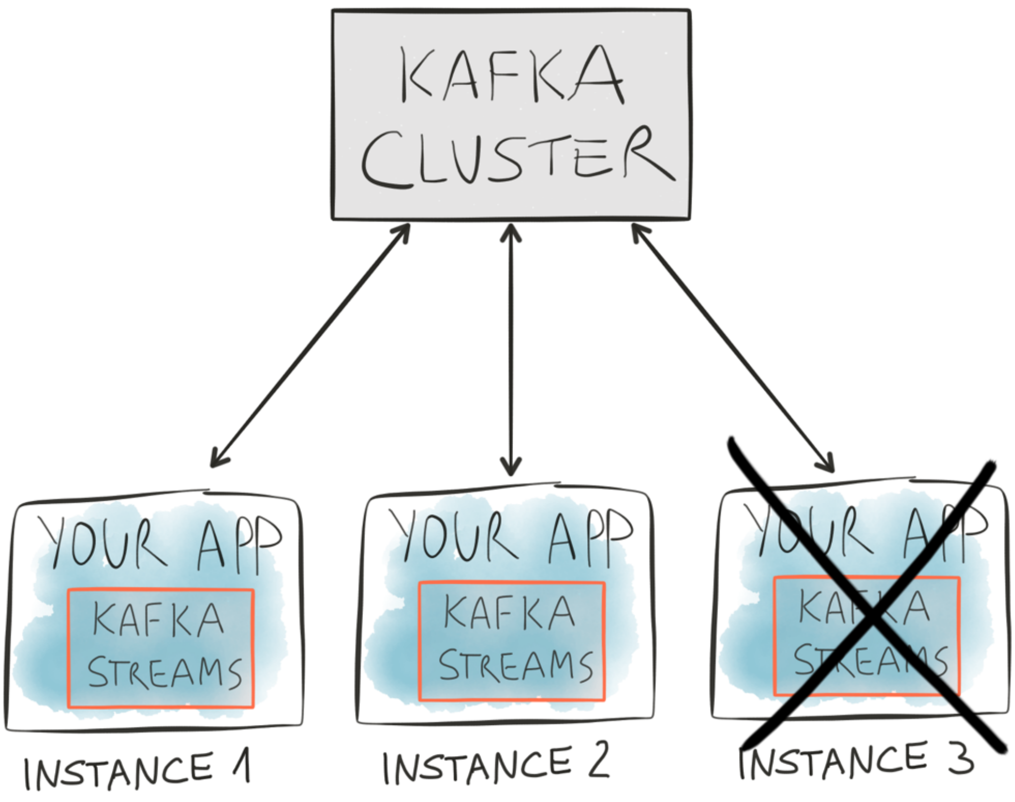
Execution model



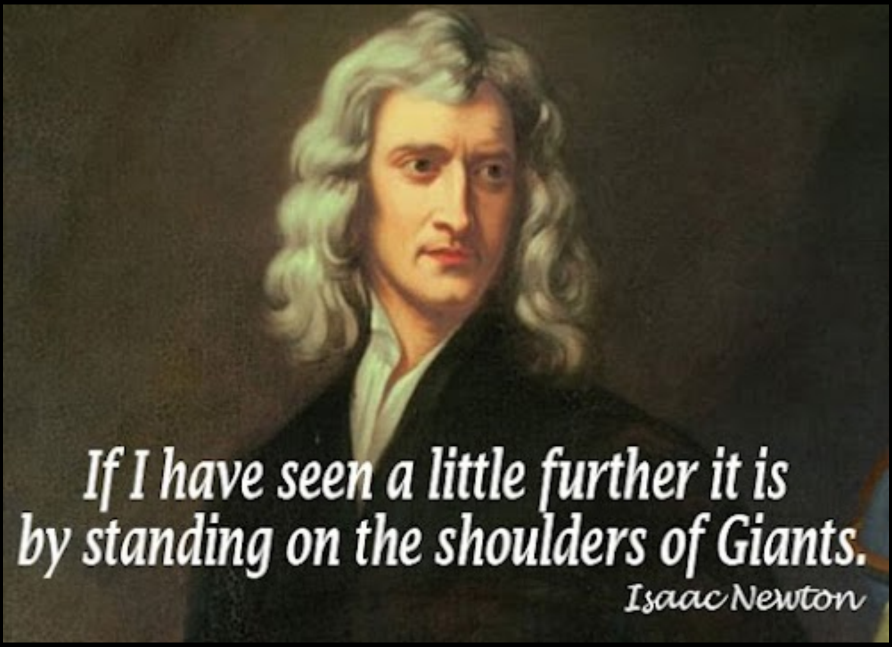
Execution model



Execution model



Kafka Streams outsources hard problems to Kafka


If I have seen a little further it is
by standing on the shoulders of Giants.
Isaac Newton

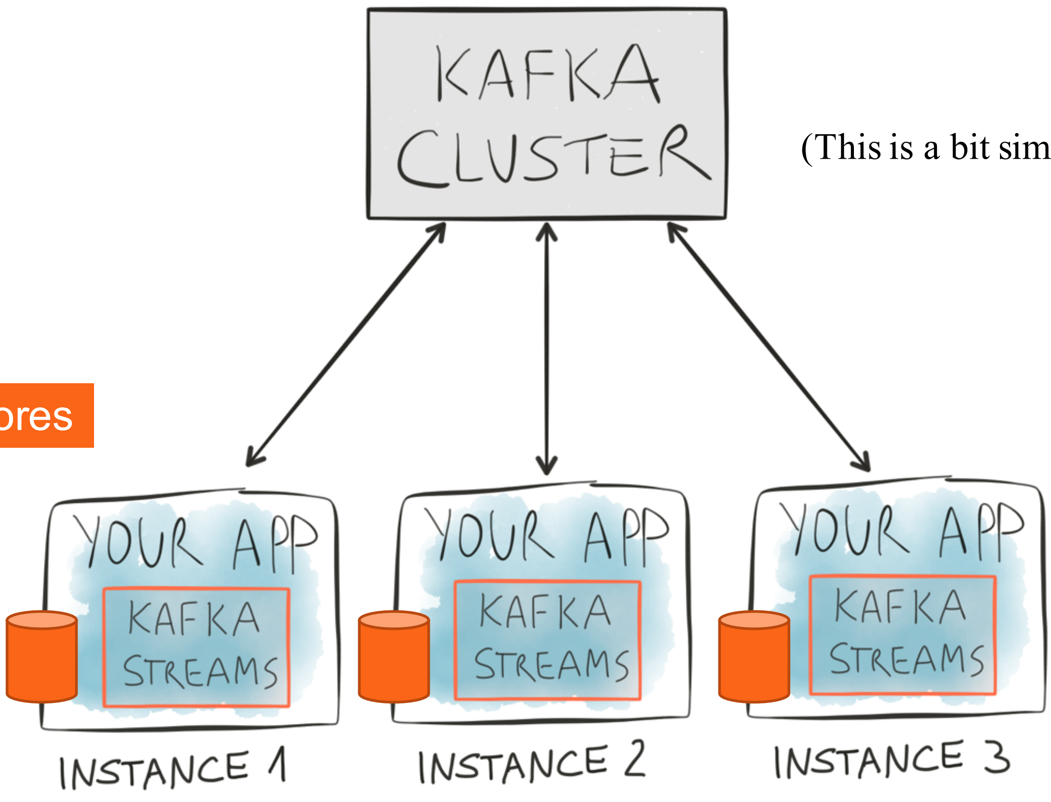
Key features in 0.10

- Native, 100%-compatible Kafka integration
  - Also inherits Kafka's security model, e.g. to encrypt data-in-transit
  - Uses Kafka as its internal messaging layer, too
- Highly scalable
- Fault-tolerant
- Elastic
- **Stateful and stateless computations (e.g. joins, aggregations)**

Stateful computations

- Stateful computations like **aggregations** or **joins** require state
  - We already showed a join example in the previous slides.
  - Windowing a stream is stateful, too, but let's ignore this for now.
- **State stores** in Kafka Streams
  - Typically: key-value stores
  - Pluggable implementation: RocksDB (default), in-memory, your own …
- State stores are **per stream task** for isolation (think: share-nothing)
- State stores are **local** for best performance
- State stores are **replicated to Kafka** for elasticity and for fault-tolerance

Execution model



(This is a bit simplified.)

State stores

Remember?

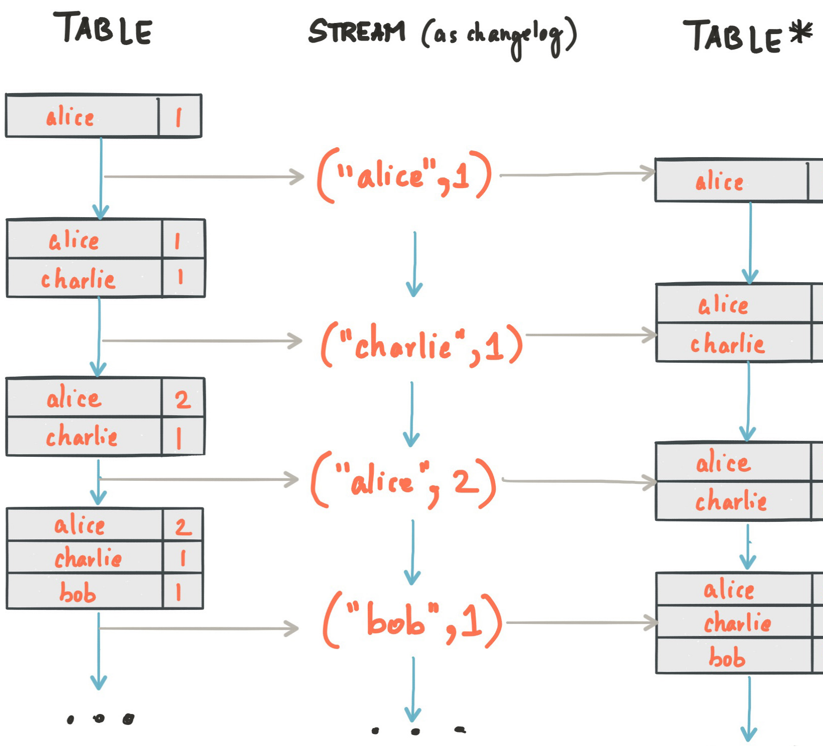
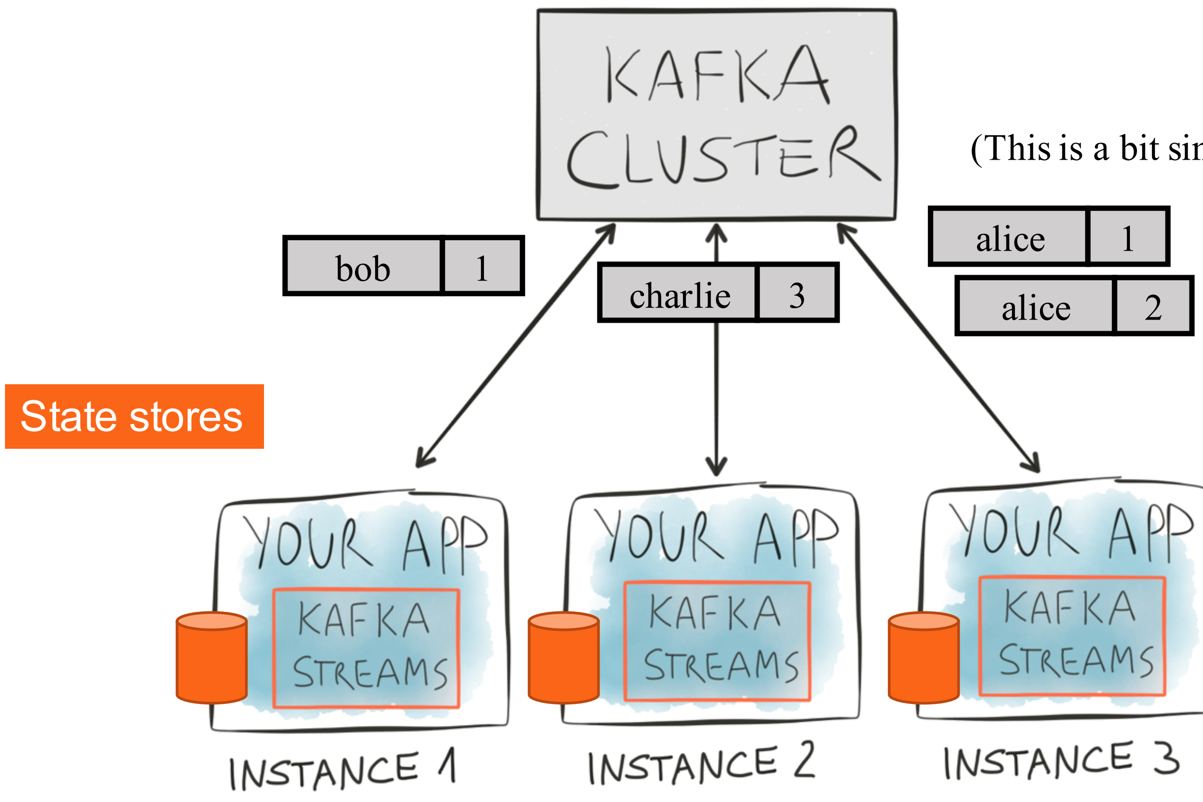


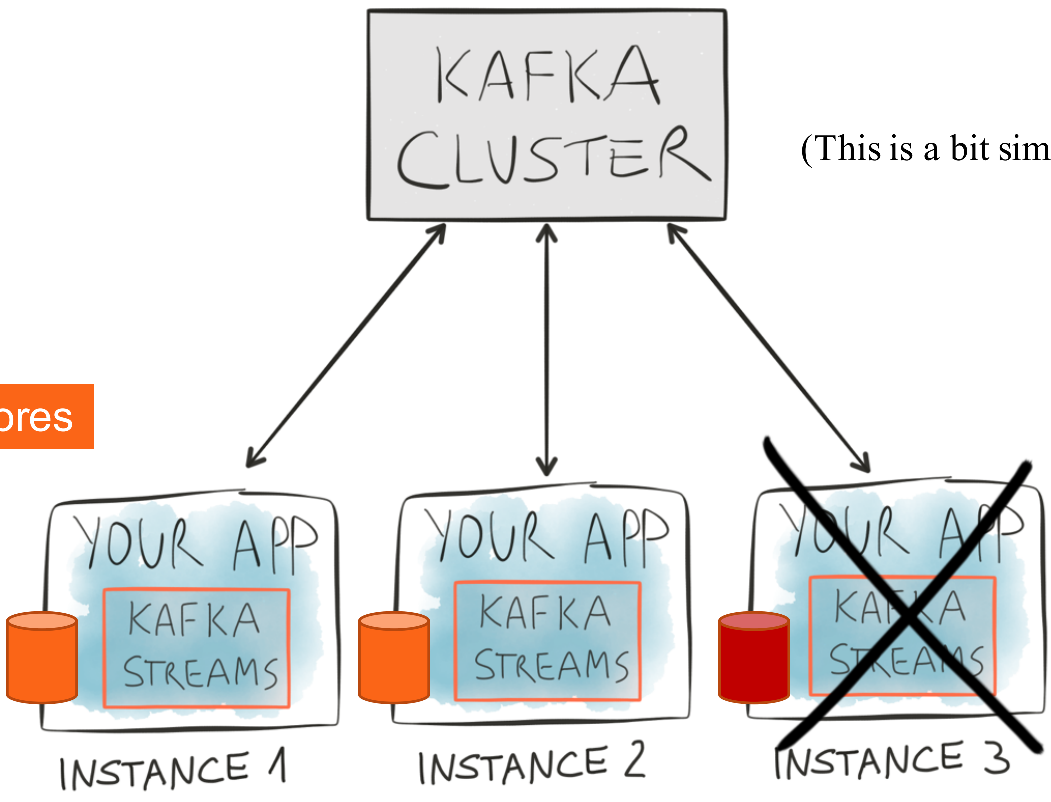
TABLE
STREAM (as changelog)
TABLE*
alice 1
alice 1
charlie 1
alice 2
charlie 1
alice 2
charlie 1
bob 1
("alice",1)
("charlie",1)
("alice", 2)
("bob",1)
. . .

Execution model

(This is a bit simplified.)



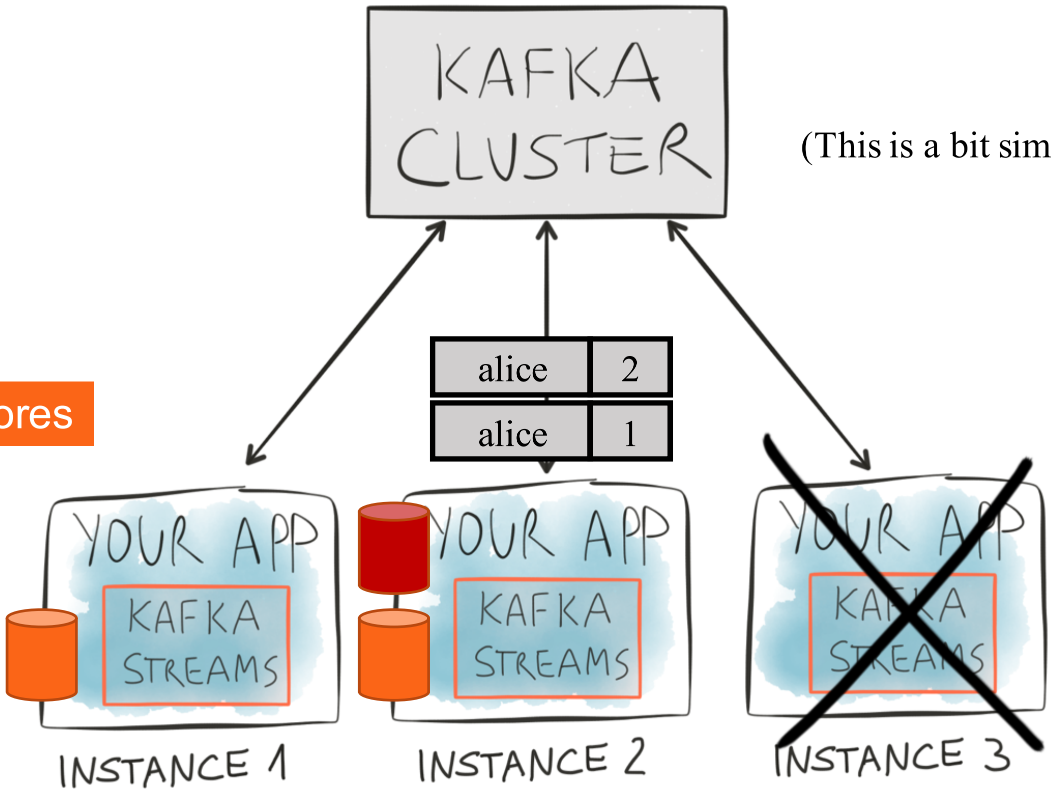
Execution model



(This is a bit simplified.)

State stores

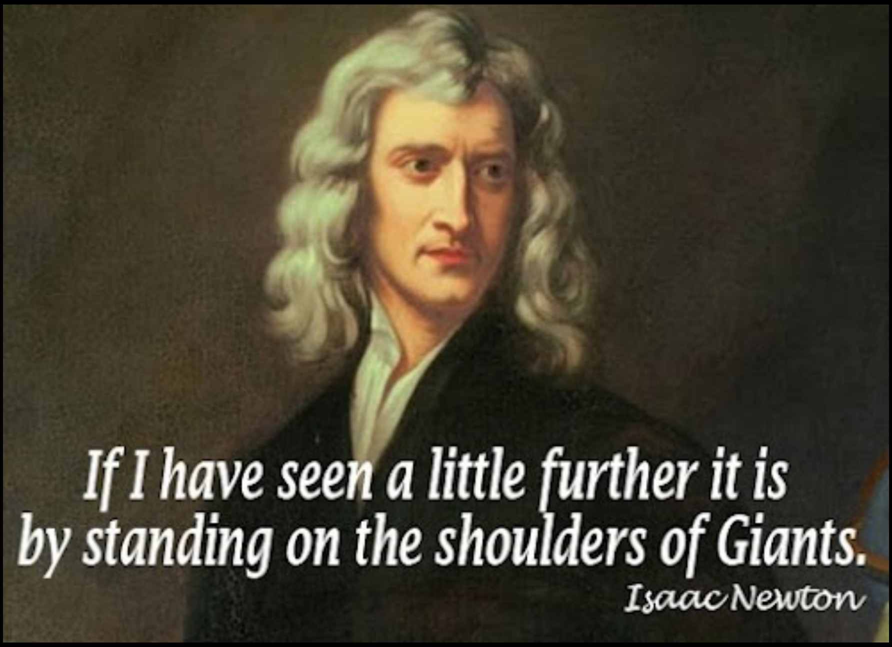
Execution model



(This is a bit simplified.)

State stores

Kafka Streams outsources hard problems to Kafka

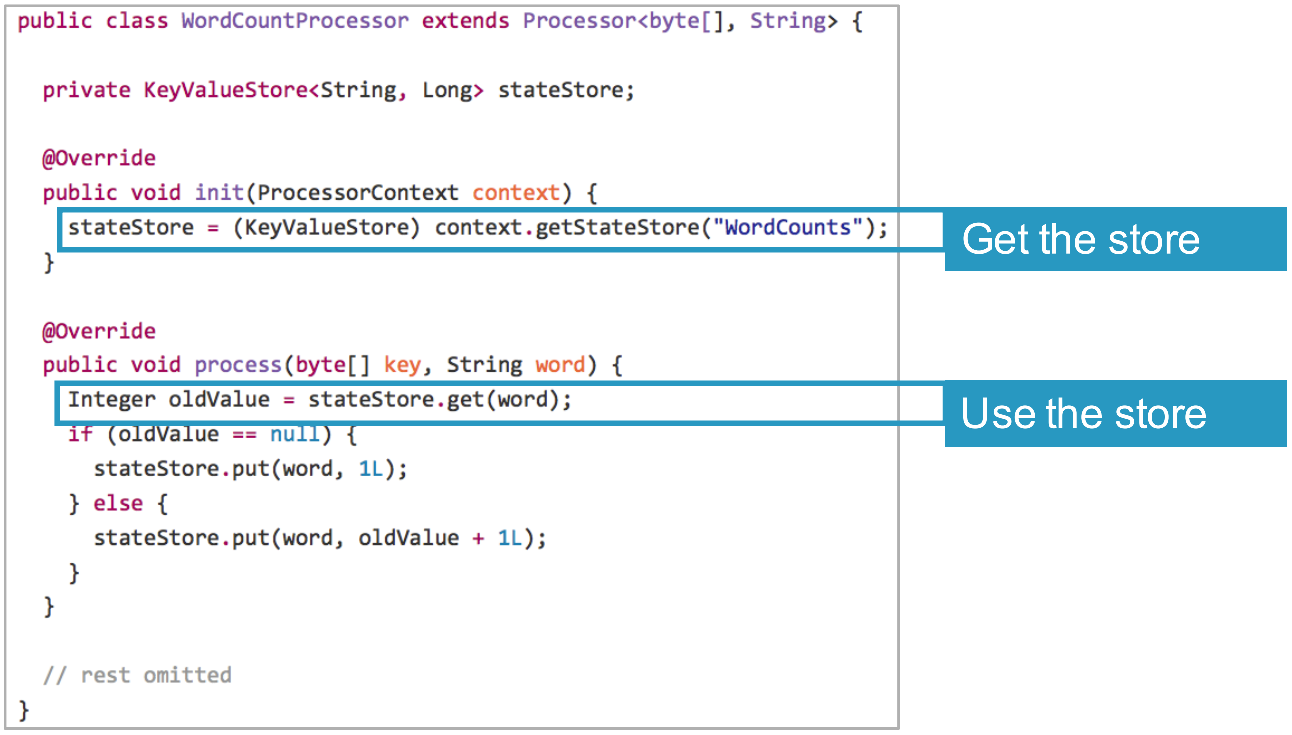

If I have seen a little further it is
by standing on the shoulders of Giants.
Isaac Newton

Stateful computations

- **Kafka Streams DSL**: abstracts state stores away from you
  - Stateful operations include
    - count(), reduceByKey(), aggregate(), …
- **Low-level Processor API**: direct access to state stores
  - Very flexible but more manual work for you

Stateful computations

- Use the low-level Processor API to interact directly with state stores

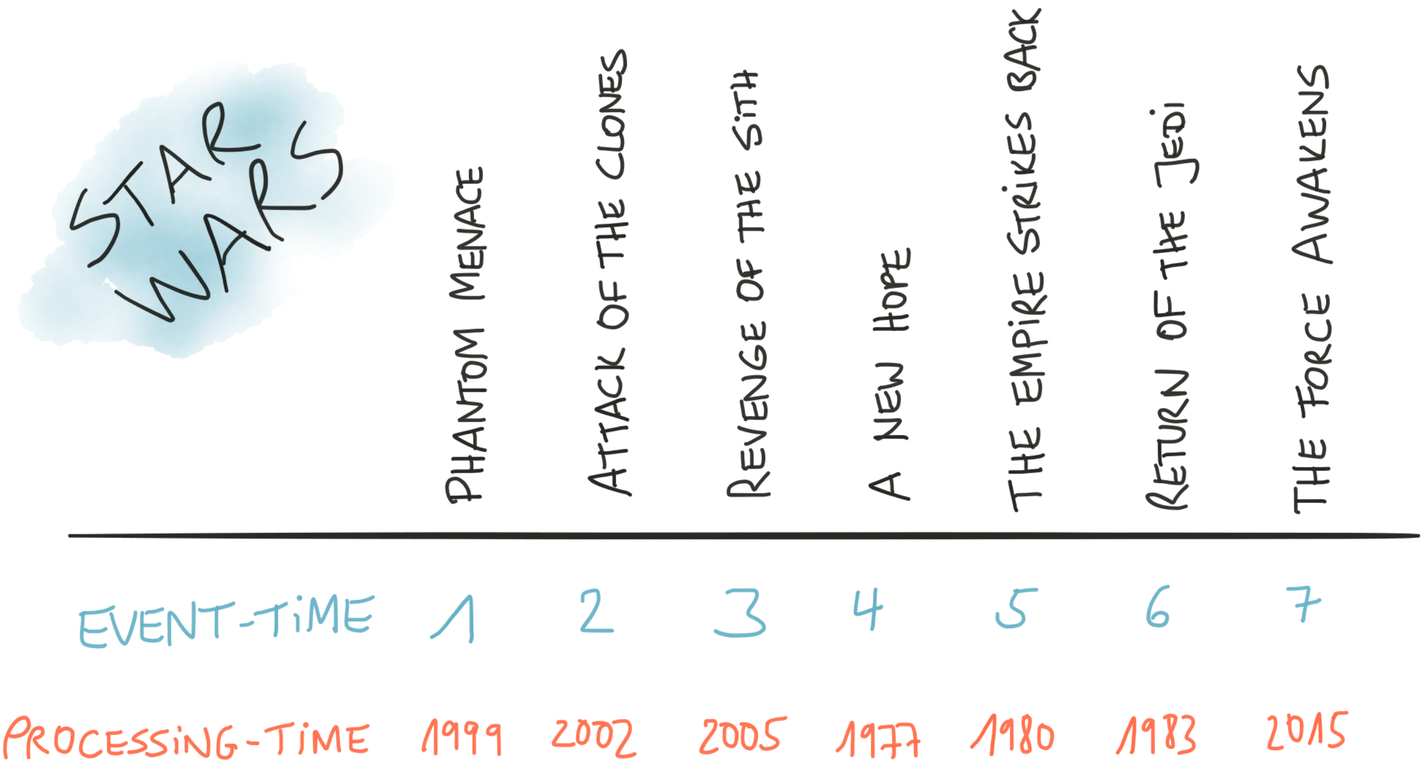


```
public class WordCountProcessor extends Processor<byte[], String> {

  private KeyValueStore<String, Long> stateStore;

  @Override
  public void init(ProcessorContext context) {
    stateStore = (KeyValueStore) context.getStateStore("WordCounts");
  }

  @Override
  public void process(byte[] key, String word) {
    Integer oldValue = stateStore.get(word);
    if (oldValue == null) {
      stateStore.put(word, 1L);
    } else {
      stateStore.put(word, oldValue + 1L);
    }
  }

  // rest omitted
}
```

Key features in 0.10

- Native, 100%-compatible Kafka integration
  - Also inherits Kafka's security model, e.g. to encrypt data-in-transit
  - Uses Kafka as its internal messaging layer, too
- Highly scalable
- Fault-tolerant
- Elastic
- Stateful and stateless computations
- **Time model**

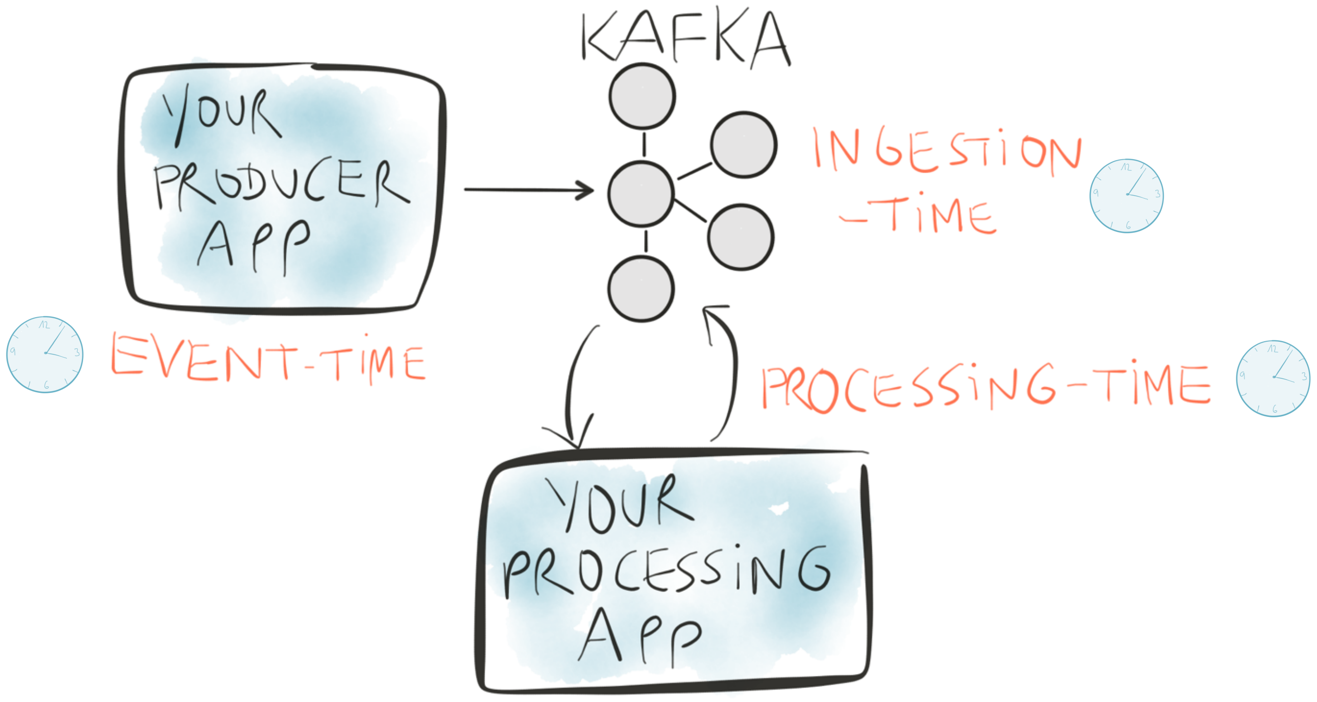
Time



STAR WARS

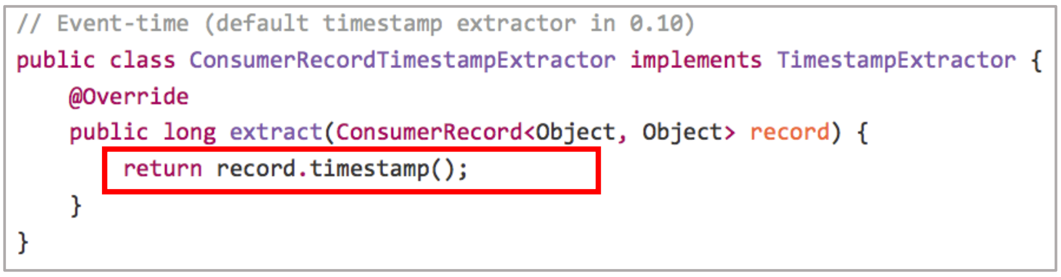
	PHANTOM MENACE	ATTACK OF THE CLONES	REVENGE OF THE SITH	A NEW HOPE	THE EMPIRE STRIKES BACK	RETURN OF THE JEDI	THE FORCE AWAKENS
EVENT-TIME	1	2	3	4	5	6	7
PROCESSING-TIME	1999	2002	2005	1977	1980	1983	2015

Time

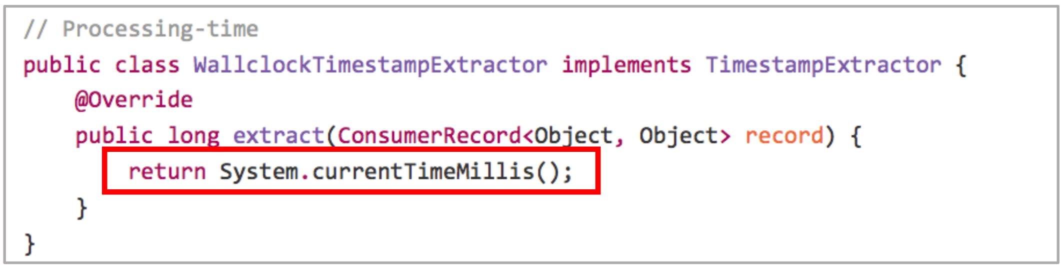

KAFKA
YOUR PRODUCER APP
INGESTION-TIME
EVENT-TIME
PROCESSING-TIME
YOUR PROCESSING APP

Time

- You configure the desired time semantics through **timestamp extractors**
- Default extractor yields **event-time** semantics
  - Extracts embedded timestamps of Kafka messages (introduced in v0.10)



```
// Event-time (default timestamp extractor in 0.10)
public class ConsumerRecordTimestampExtractor implements TimestampExtractor {
    @Override
    public long extract(ConsumerRecord<Object, Object> record) {
        return record.timestamp();
    }
}
```



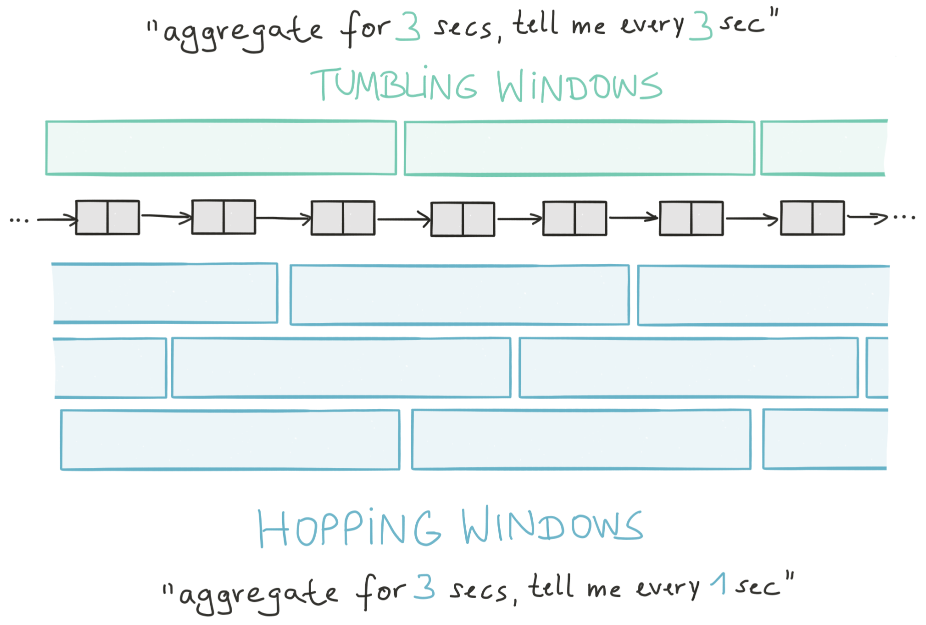
```
// Processing-time
public class WallclockTimestampExtractor implements TimestampExtractor {
    @Override
    public long extract(ConsumerRecord<Object, Object> record) {
        return System.currentTimeMillis();
    }
}
```

Key features in 0.10

- Native, 100%-compatible Kafka integration
  - Also inherits Kafka’s security model, e.g. to encrypt data-in-transit
  - Uses Kafka as its internal messaging layer, too
- Highly scalable
- Fault-tolerant
- Elastic
- Stateful and stateless computations
- Time model
- **Windowing**

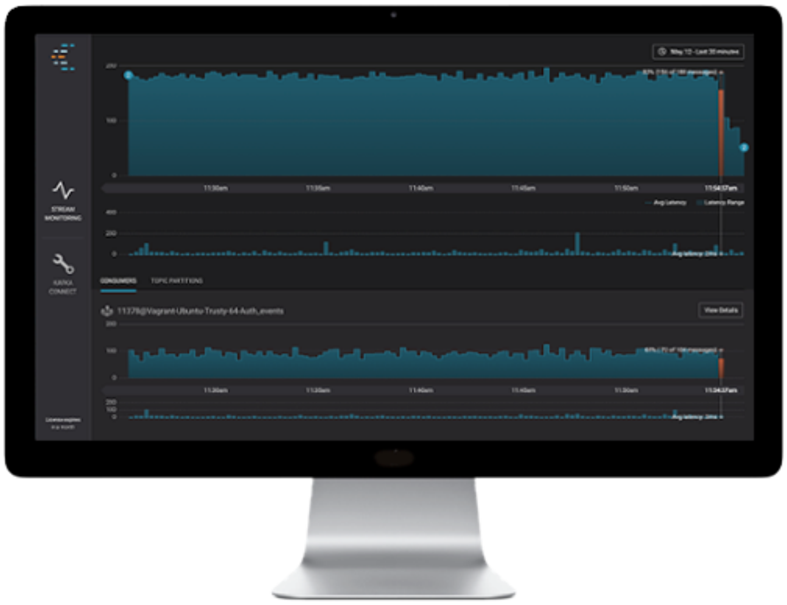
Windowing

```
TimeWindows.of(3000)
```


"aggregate for 3 secs, tell me every 3 sec"
TUMBLING WINDOWS
HOPPING WINDOWS
"aggregate for 3 secs, tell me every 1 sec"

```
TimeWindows.of(3000).advanceBy(1000)
```

Windowing use case: monitoring (1m/5m/15m averages)



Confluent Control Center for Kafka

Key features in 0.10

- Native, 100%-compatible Kafka integration
  - Also inherits Kafka’s security model, e.g. to encrypt data-in-transit
  - Uses Kafka as its internal messaging layer, too
- Highly scalable
- Fault-tolerant
- Elastic
- Stateful and stateless computations
- Time model
- Windowing
- **Supports late-arriving and out-of-order data**
- **Millisecond processing latency, no micro-batching**
- **At-least-once processing guarantees (exactly-once is in the works)**

Wrapping up

Where to go from here?

- Kafka Streams is available in Apache Kafka 0.10 and Confluent Platform 3.0
  - http://kafka.apache.org/
  - http://www.confluent.io/download (free + enterprise versions, tar/zip/deb/rpm)

- Kafka Streams demos at https://github.com/confluentinc/examples
  - Java 7, Java 8+ with lambdas, and Scala
  - WordCount, Joins, Avro integration, Top-N computation, Windowing, …

- Apache Kafka documentation: http://kafka.apache.org/documentation.html
- Confluent documentation: http://docs.confluent.io/3.0.0/streams/
  - Quickstart, Concepts, Architecture, Developer Guide, FAQ

- Join our bi-weekly *Ask Me Anything* sessions on Kafka Streams
  - Contact me at michael@confluent.io for details

Some of the things to come

- Exactly-once semantics
- Queriable state – tap into the state of your applications
- SQL interface
- Listen to and collaborate with the developer community
    - Your feedback counts a lot! Share it via users@kafka.apache.org

Tomorrow's keynote (09:30 AM) by Neha Narkhede, co-founder and CTO of Confluent

"Application development and data in the emerging world of stream processing"

Want to contribute to Kafka and open source?

Join the Kafka community
http://kafka.apache.org/

…in a great team with the creators of Kafka?

Confluent is hiring ☺
http://confluent.io/

Questions, comments? Tweet with #bbuzz and /cc to @ConfluentInc