2.1 Producer API

The Producer API allows applications to send streams of data to topics in the Kafka cluster.

Examples showing how to use the producer are given in the javadocs.

To use the producer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.7.0</version>
</dependency>
```

2.2 Consumer API

The Consumer API allows applications to read streams of data from topics in the Kafka cluster.

Examples showing how to use the consumer are given in the javadocs.

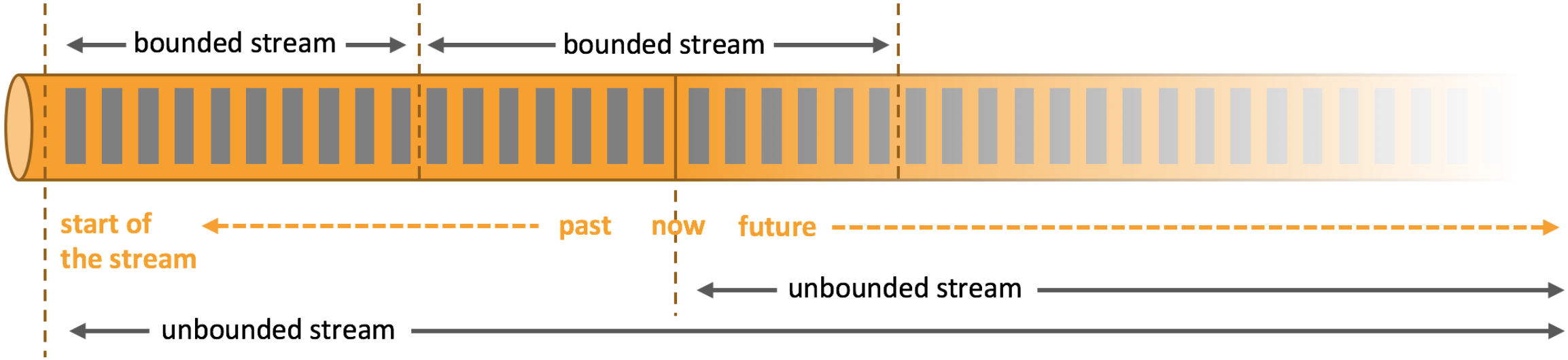
To use the consumer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.7.0</version>
```

We have an smart assistant

What is **Stream Processing?**

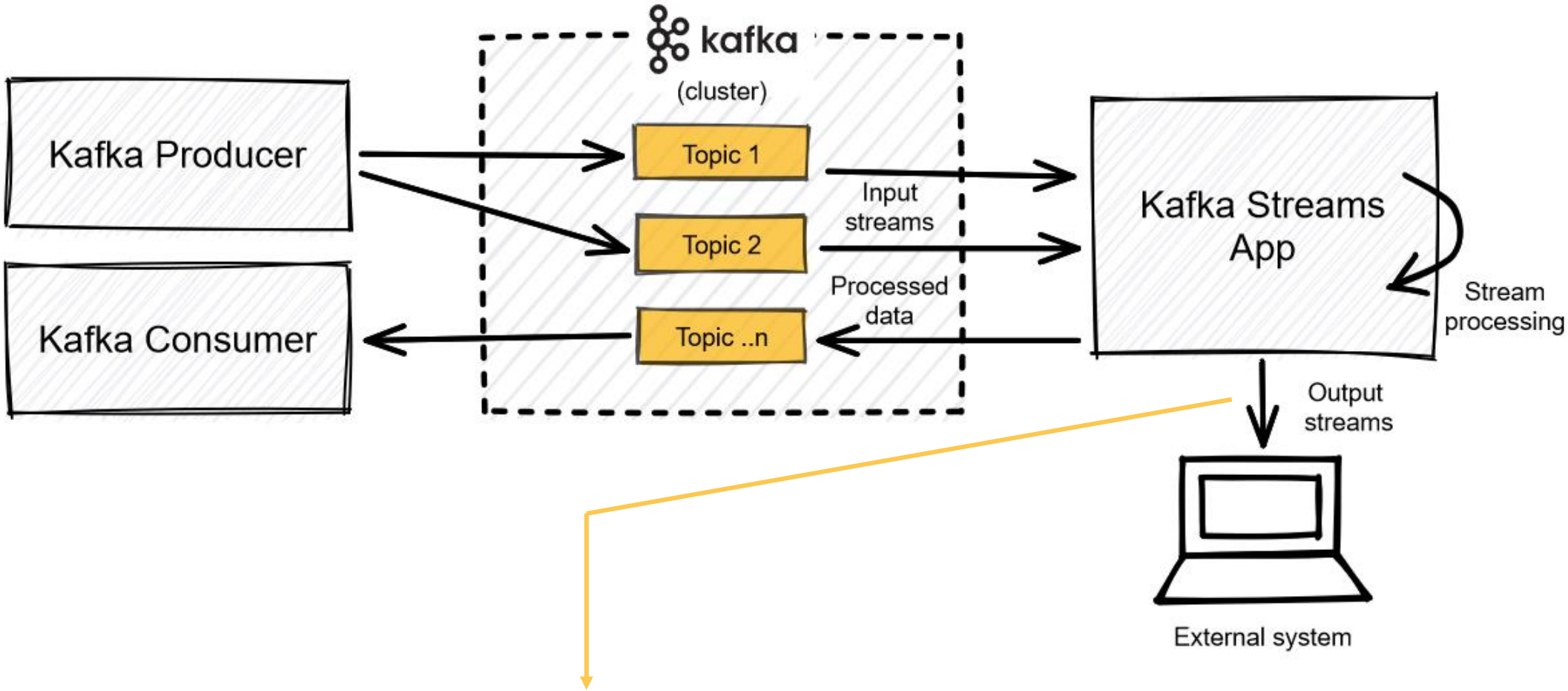
Stream processing (or streaming) is a process or application you implement that deals with an uninterrupted flow of data and performs work as soon as that data arrives.



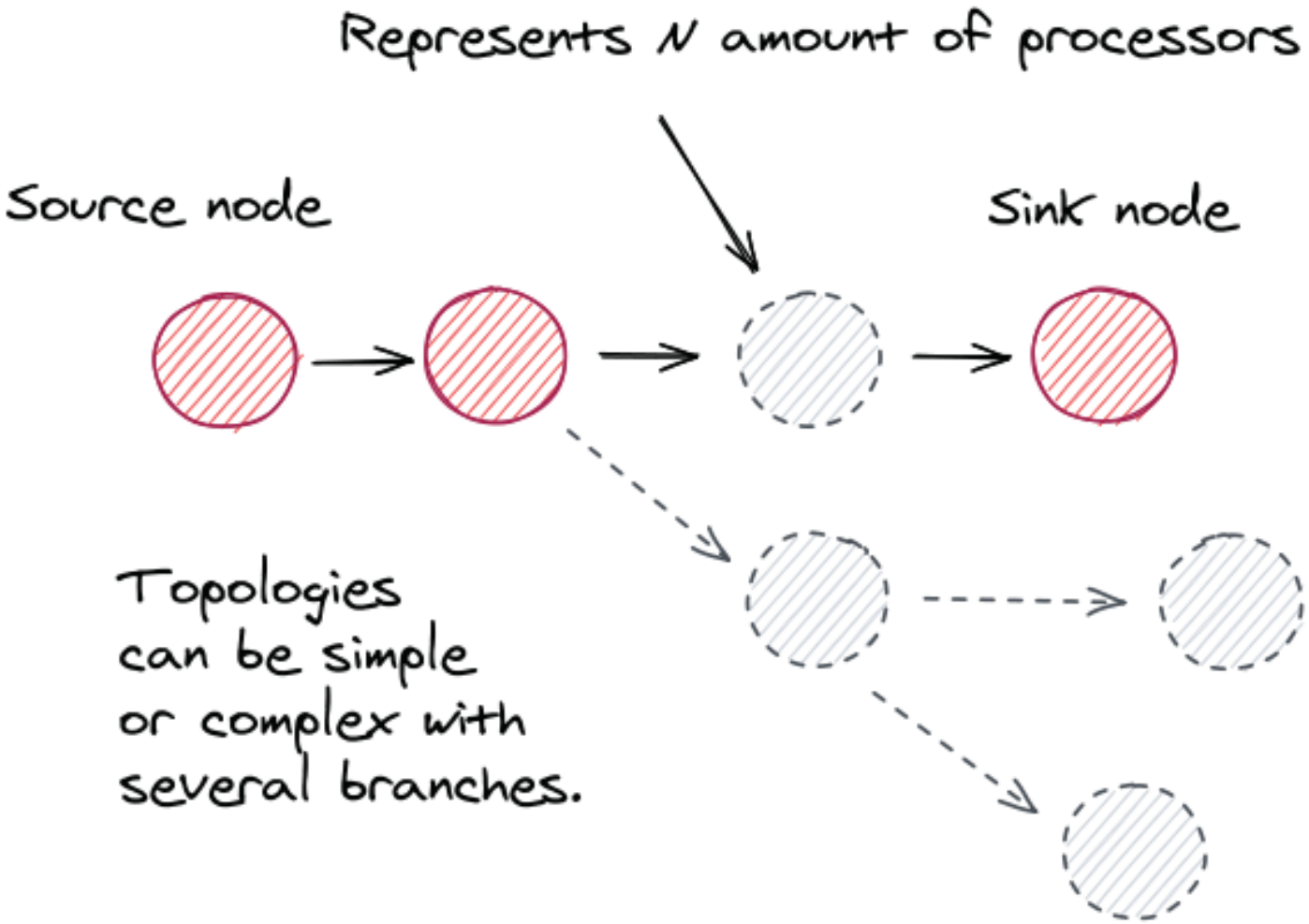
Stream processing (or streaming) is a process or application you implement that deals with an uninterrupted flow of data and performs work as soon as that data arrives.

Kafka Streams = { **Streams DSL API (A high–level abstraction)**
Processor API (A low–Level API)

DSL comes from Domain-Specific Language

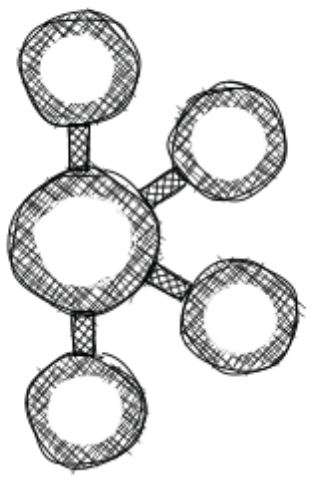


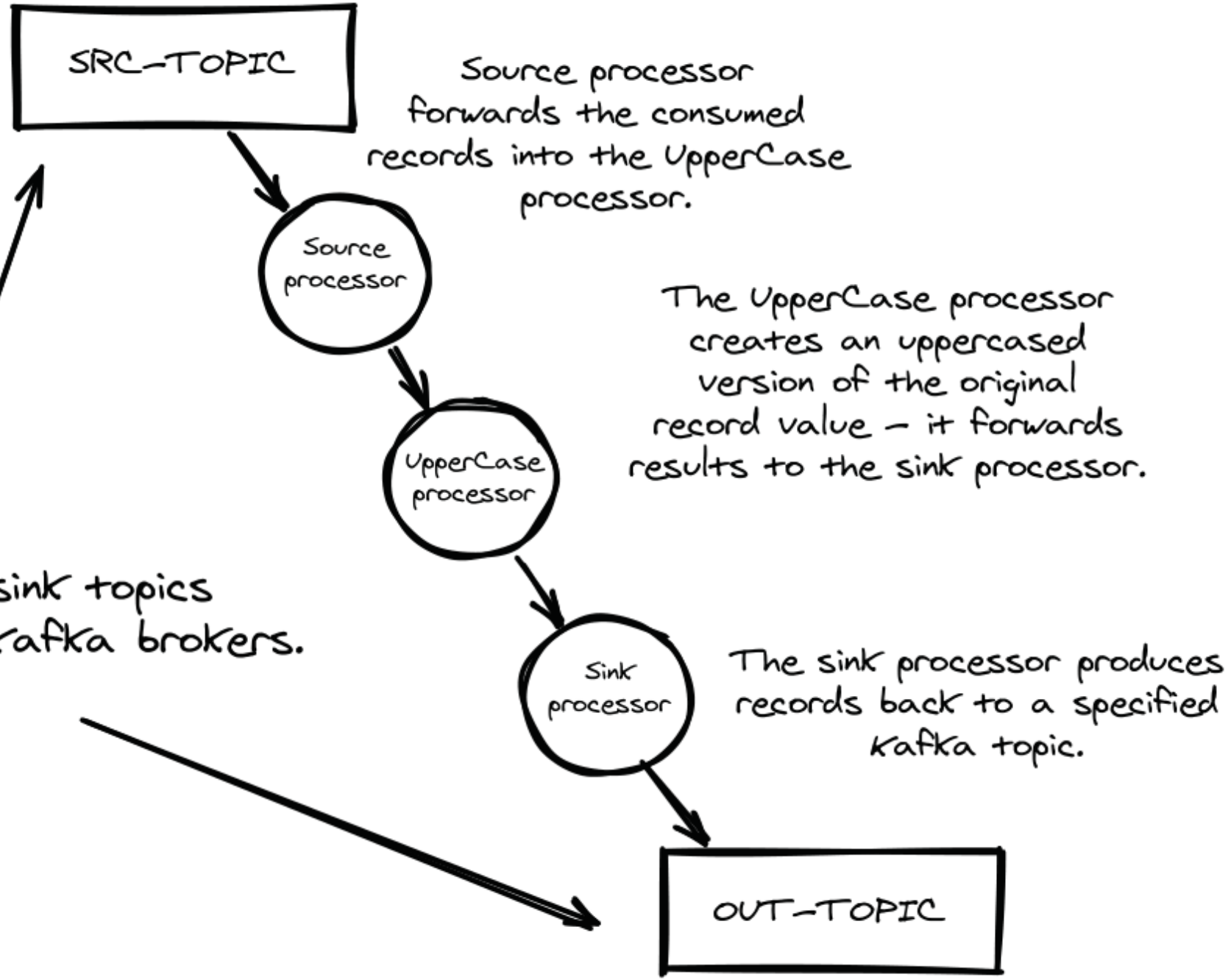
We can do this by using **Connect Sink Connectors**



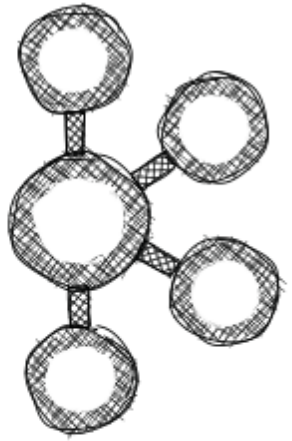
Kafka Streams is a **graph** with a source node, any number of processing nodes, and a sink node.

Kafka Streams application is a separate entity that interacts with Kafka clusters to process data but does not run directly on the Kafka broker servers.



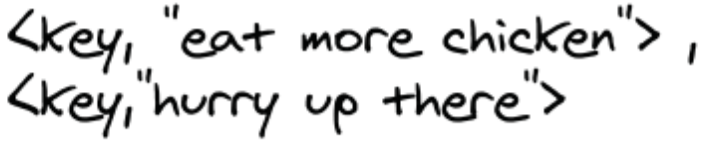


Source and sink topics are on the Kafka brokers.



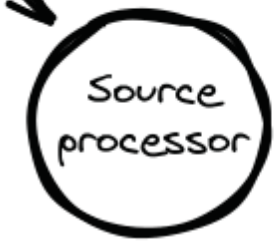
A **KStream** is an object and represents a continuous and potentially infinite stream of records flowing through a Kafka topic. When you create a **KStream** object in your Kafka Streams application, you are essentially creating a representation of an unbounded stream of data that you can process and manipulate using various operations provided by the Kafka Streams API.

```
KStream<String, String> simpleFirstStream =
        builder.stream("src-topic",
Consumed.with(Serdes.String(), Serdes.String()));
```

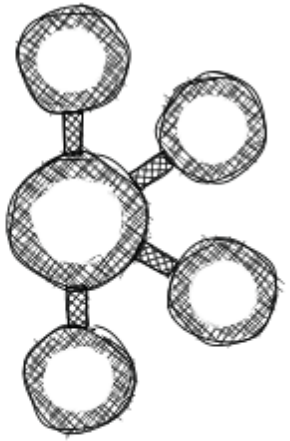


The **.stream()** method in the StreamsBuilder class is used to create a KStream or KTable from a Kafka topic

key-value records consumed from the topic(s) named when creating the source node



A **KTable** is an immutable data structure that is continuously updated based on the events in the input Kafka topic. It is designed to handle updates through a mechanism known as **changelog streams**.



SRC-TOPIC

<key, "eat more chicken">,
<key, "hurry up there">

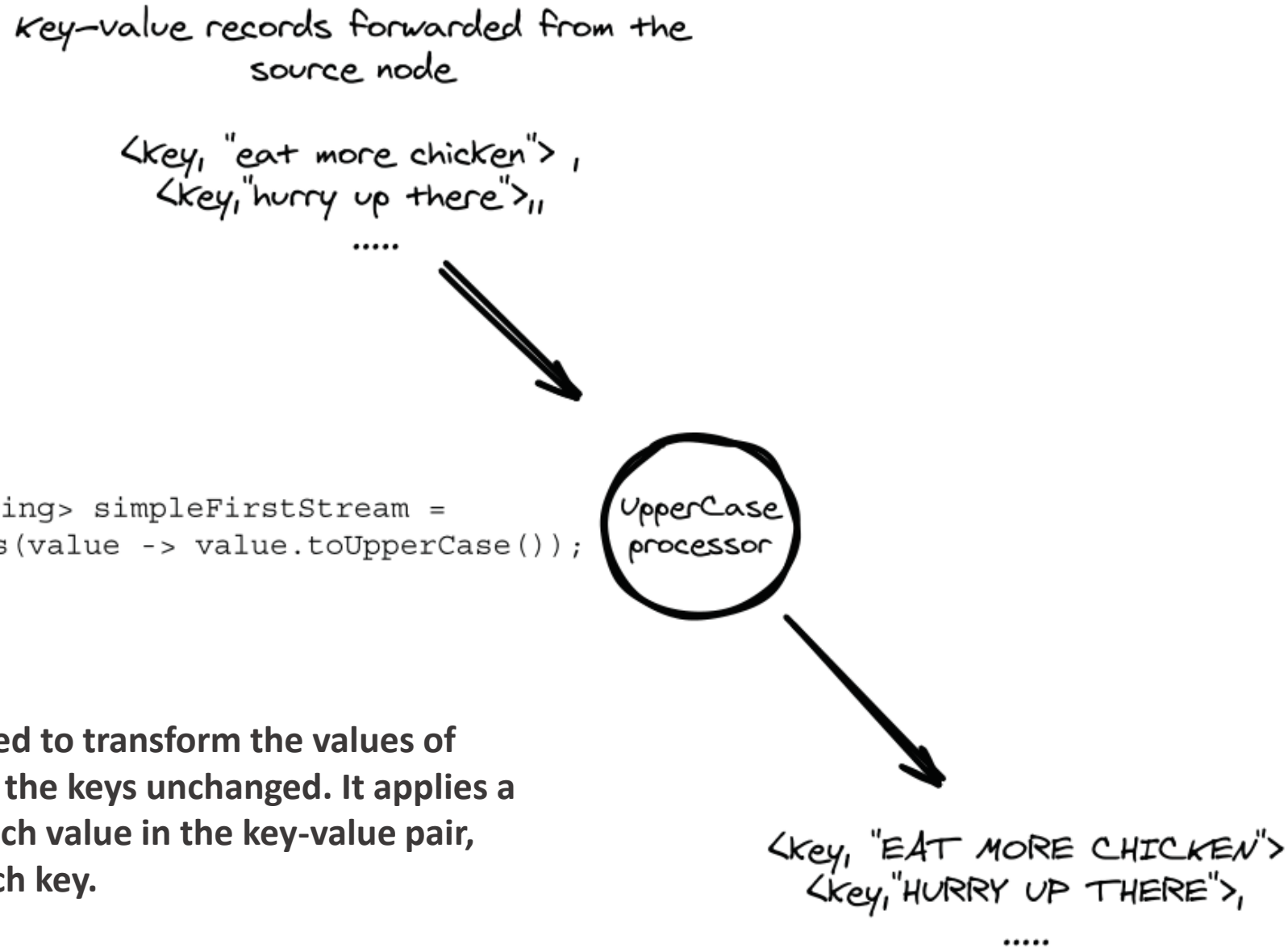
key-value records consumed from the topic(s) named when creating the source node

```
KStream<String, String> simpleFirstStream =
        builder.stream("src-topic",
Consumed.with(Serdes.String(), Serdes.String()));
```

The Serdes class (short for Serializer/Deserializer) is a fundamental component used for serializing (converting objects to bytes) and deserializing (converting bytes to objects) data when reading from or writing to Kafka topics.

Since in Kafka Streams we have produce and consume operations, so we should use of Serdes class.

Source processor



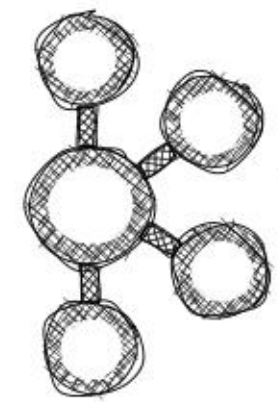
The mapValues function is used to transform the values of key-value pairs while keeping the keys unchanged. It applies a transformation function to each value in the key-value pair, producing a new value for each key.

Key-value records forwarded from the UpperCase processor

<Key, "EAT MORE CHICKEN">
<Key,"HURRY UP THERE">,
.....

Sink processor

Out-Topic



```
upperCasedStream.to("out-topic",
        Produced.with(Serdes.String(),
       Serdes.String()));
```

Coding Time. Fasten Your Seat Belt!

In the upcoming videos, we are going to investigate three different awesome examples.