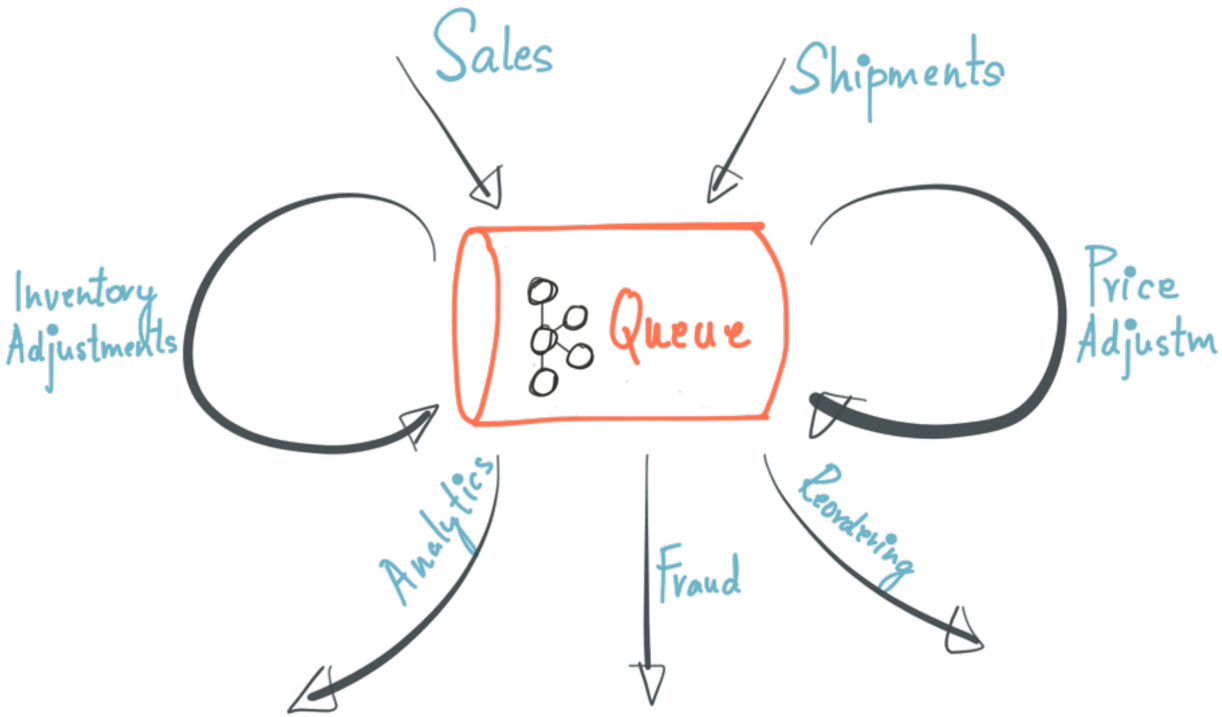
Kafka Streams

Stream processing Made Simple with Kafka

Kafka: Real-time Platforms

- *Persistent Buffering*
- *Logical Ordering*
- *Scalable "source-of-truth"*



Stream Processing with Kafka

Stream Processing with Kafka

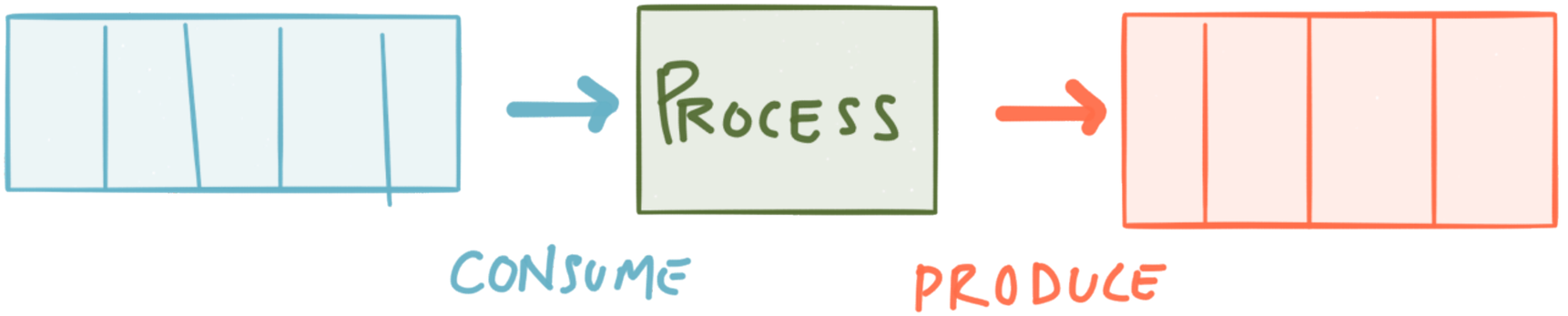
- *Option I: Do It Yourself !*

Stream Processing with Kafka

- *Option I: Do It Yourself !*

```
while (isRunning) {
    // read some messages from Kafka
    inputMessages = consumer.poll();

    // do some processing…

    // send output messages back to Kafka
    producer.send(outputMessages);
}
```

DIY Stream Processing



Stream Processing with Kafka

- *Option I: Do It Yourself !*
- *Option II: full-fledged stream processing system*
- *Option III: lightweight stream processing* ***library***

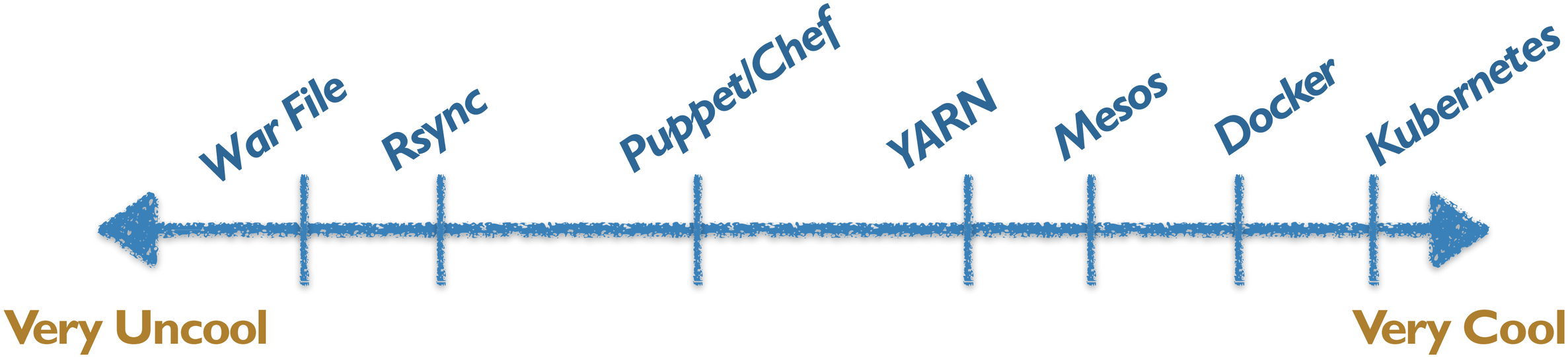
Kafka Streams

- *In Apache Kafka since v0.10, May 2016*
- *Powerful yet easy-to-use stream processing library*
  - *Event-at-a-time, Stateful*
  - *Windowing with out-of-order handling*
  - *Highly scalable, distributed,* fault tolerant
  - *and more..*

Anywhere, anytime

```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-streams</artifactId>
  <version>2.10.0.0</version>
</dependency>
```

Anywhere, anytime



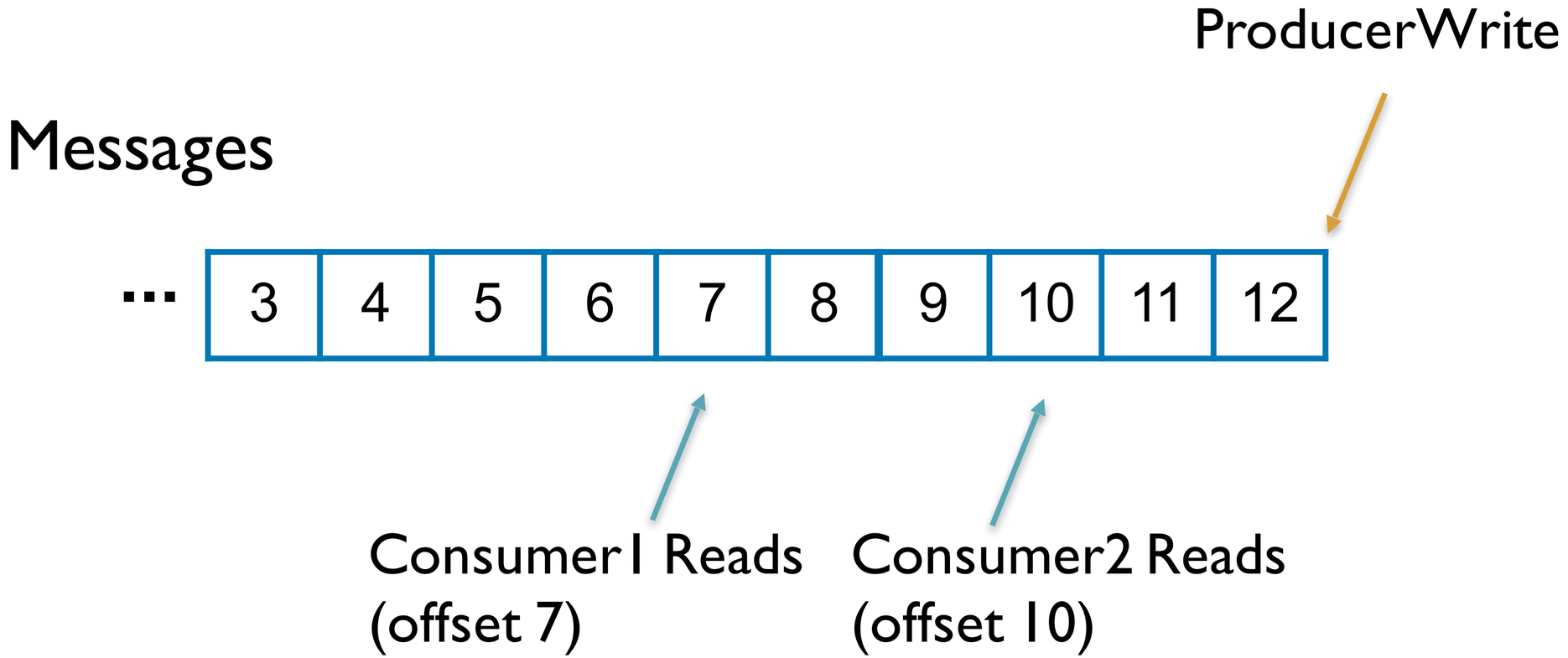
Kafka Streams DSL

```
public static void main(String[] args) {
    // specify the processing topology by first reading in a stream from a topic
    KStream<String, String> words = builder.stream("topic1");

    // count the words in this stream as an aggregated table
    KTable<String, Long> counts = words.countByKey("Counts");

    // write the result table to a new topic
    counts.to("topic2");

    // create a stream processing instance and start running it
    KafkaStreams streams = new KafkaStreams(builder, config);
    streams.start();
}
```

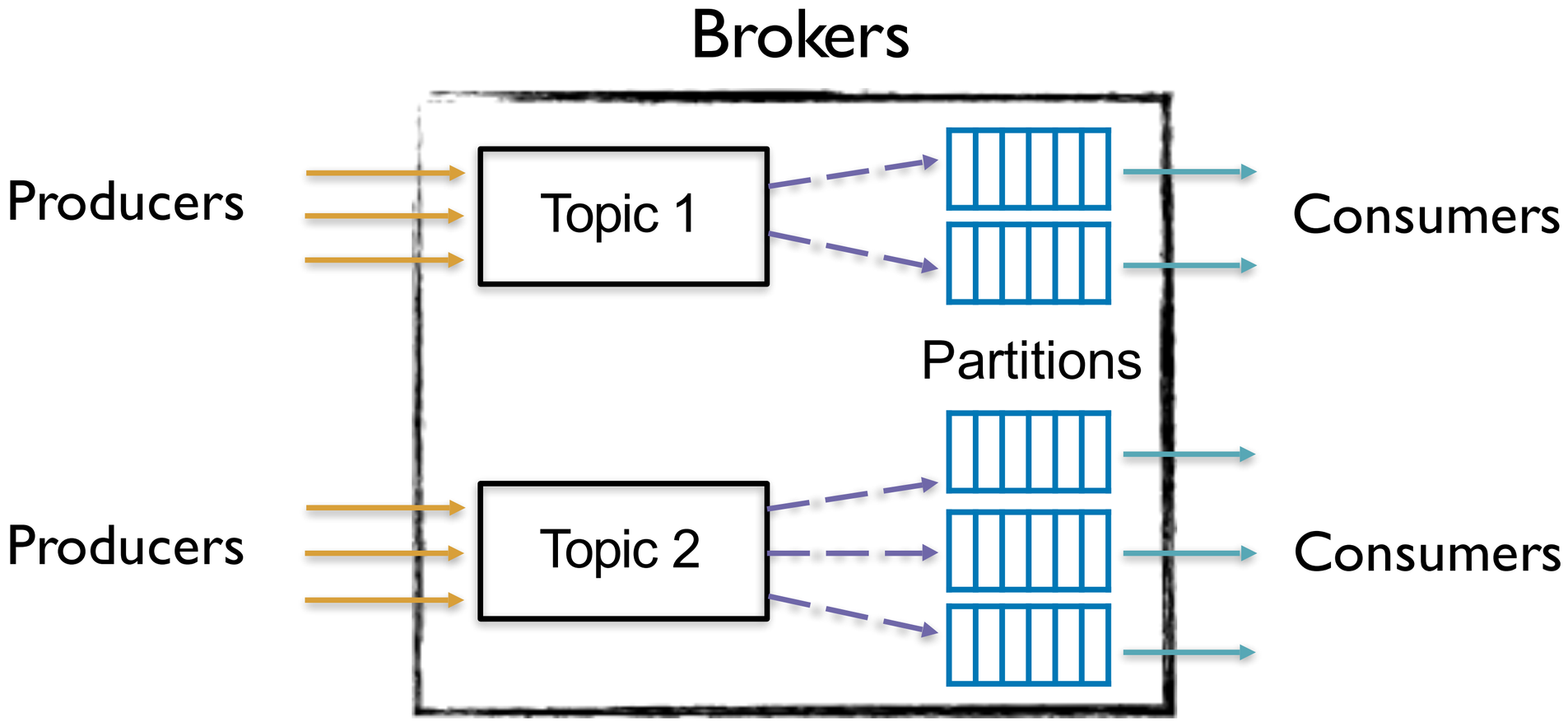
Native Kafka Integration

```
Property cfg = new Properties();

cfg.put(StreamsConfig.APPLICATION_ID_CONFIG, “my-streams-app”);
cfg.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, “broker1:9092”);
cfg.put(ConsumerConfig.AUTO_OFFSET_RESET_CONIFG, “earliest”);
cfg.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, “SASL_SSL”);
cfg.put(KafkaAvroSerDeConfig.SCHEMA_REGISTRY_URL_CONFIG, “registry:8081”);

StreamsConfig config = new StreamsConfig(cfg);

…
KafkaStreams streams = new KafkaStreams(builder, config);
```

Kafka Concepts: the *Log*

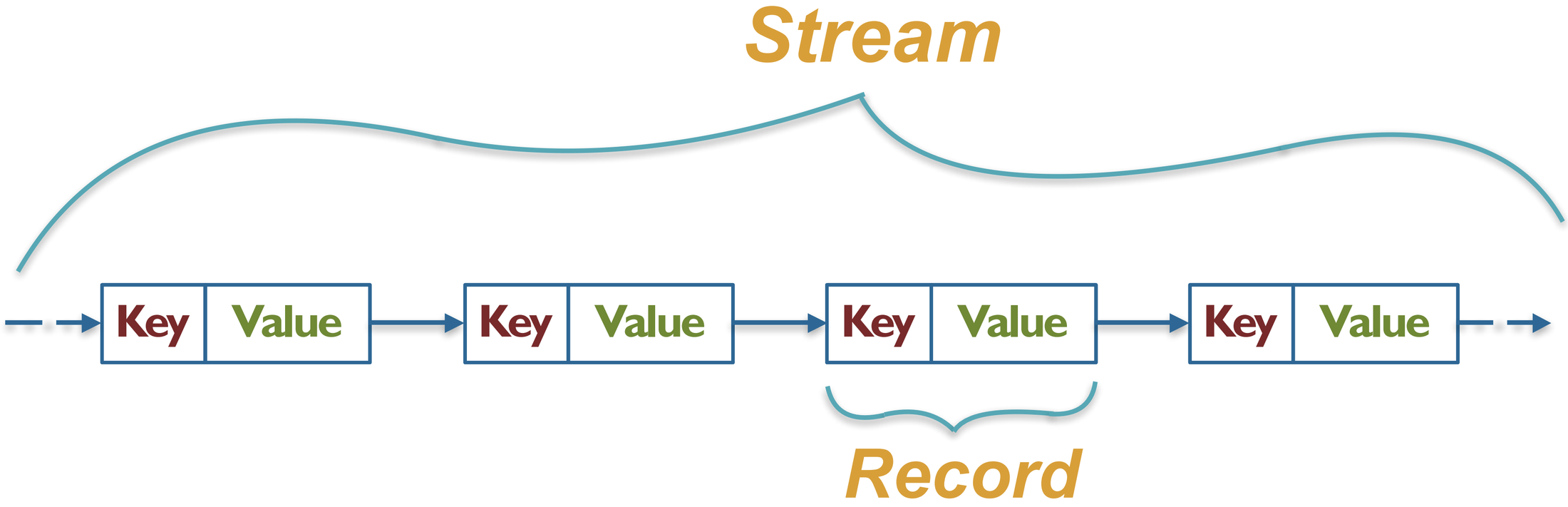


Kafka Concepts: the *Log*

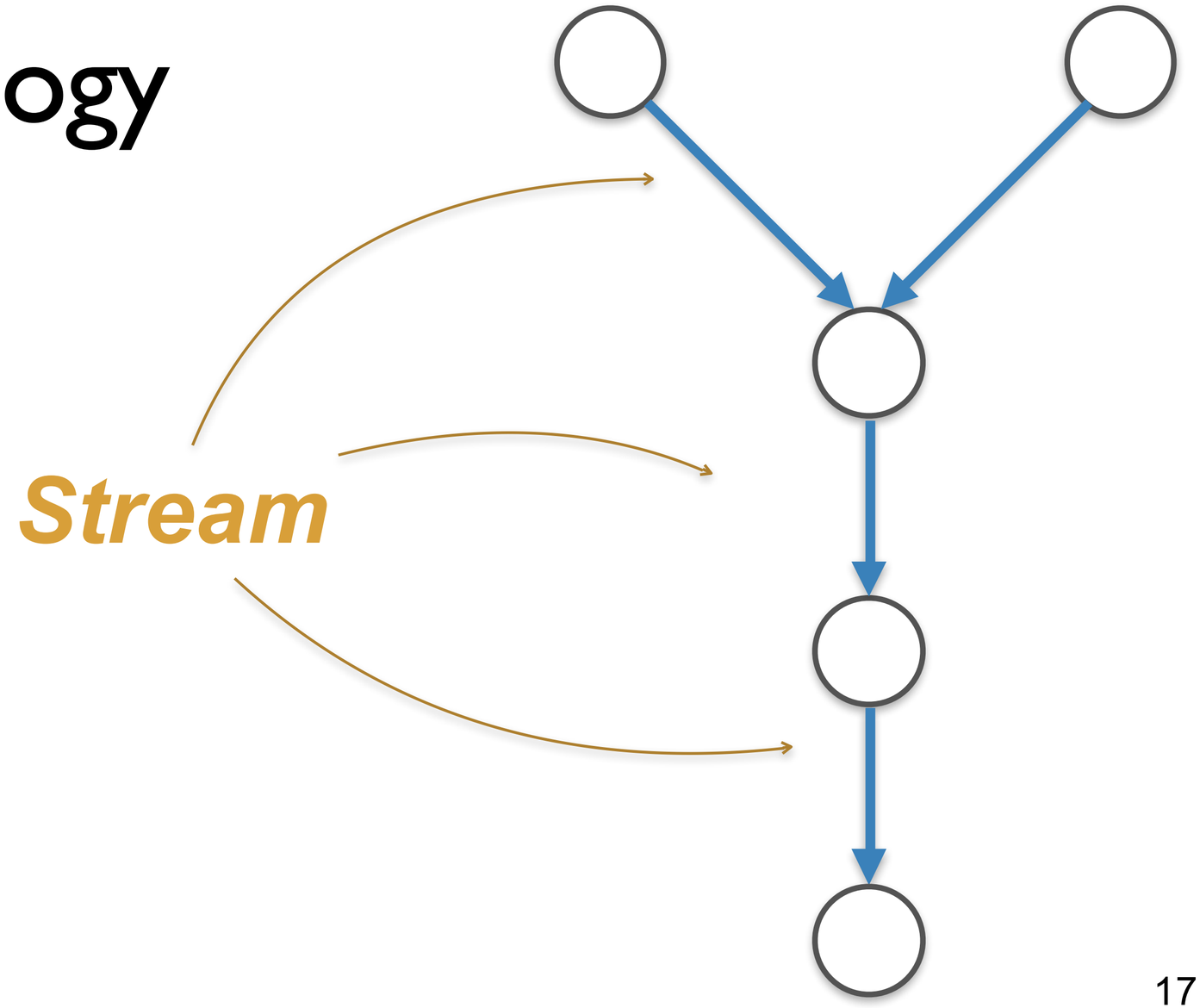


Kafka Streams: **Key Concepts**

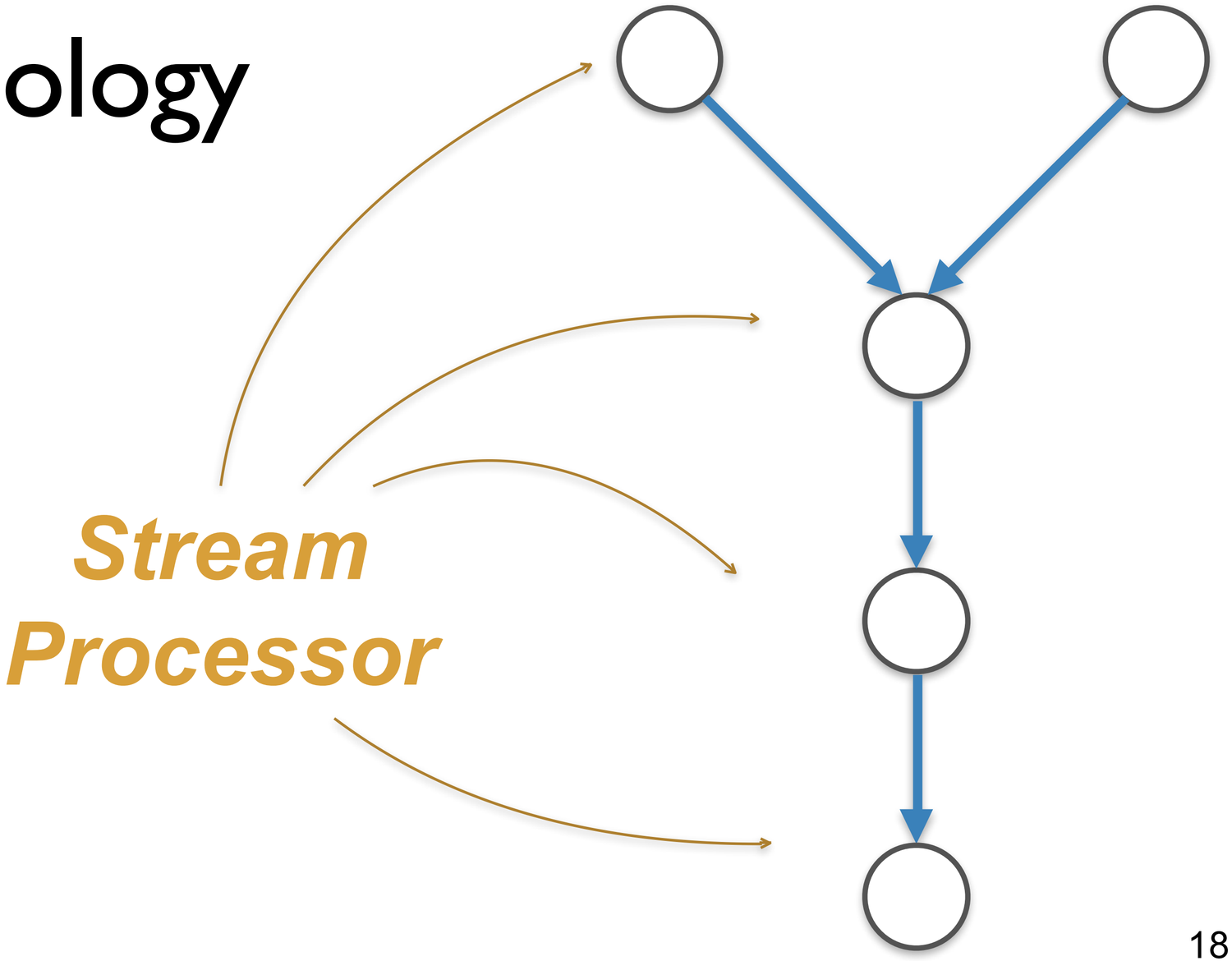
Stream and Records



Processor Topology

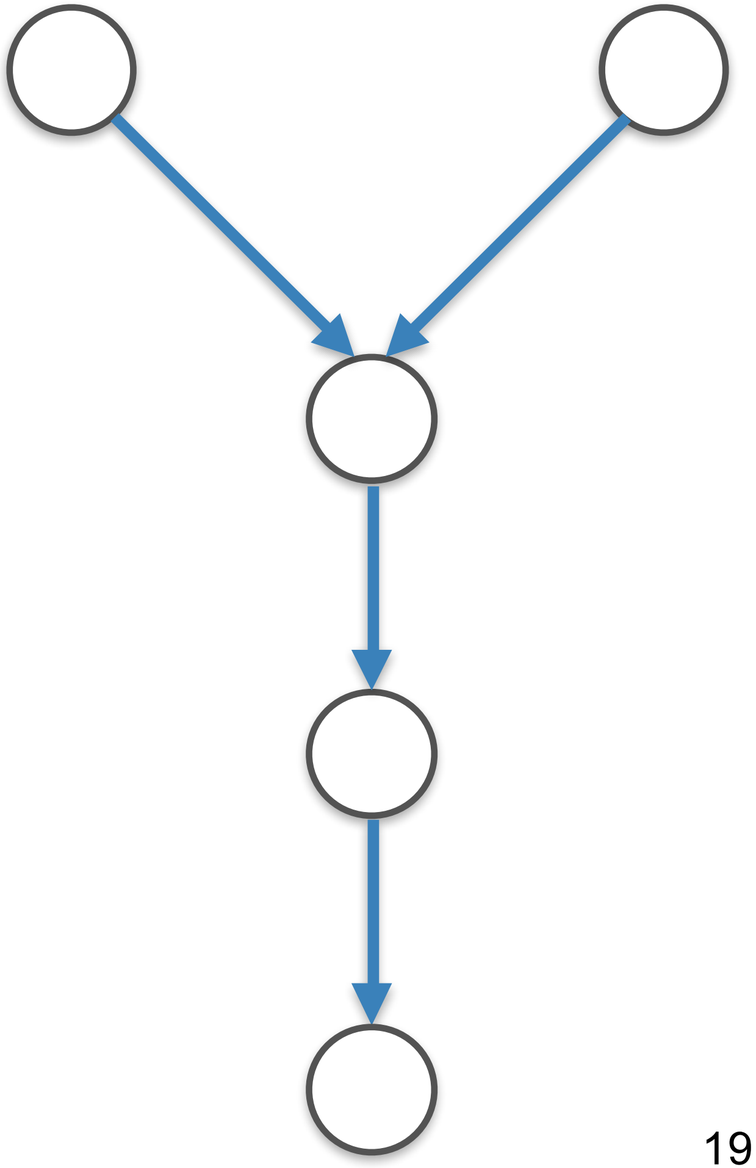


Processor Topology



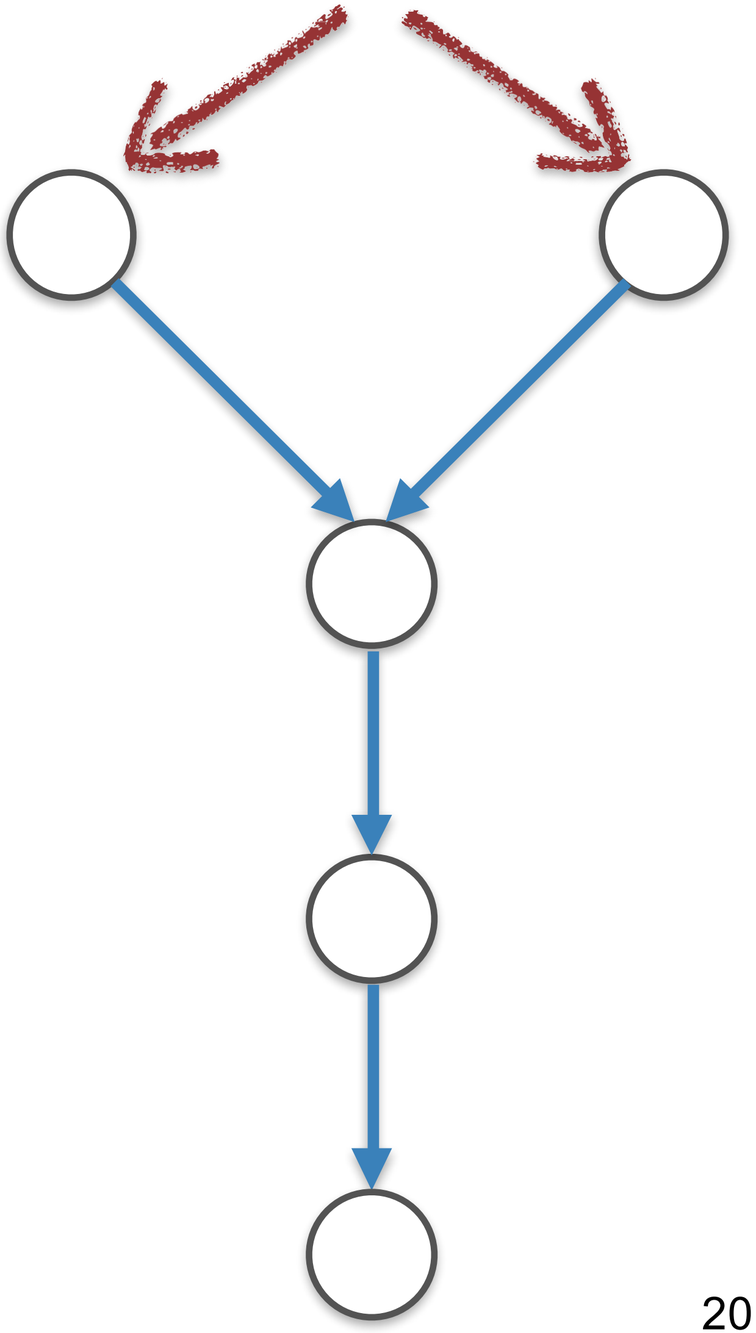
Processor Topology

```
KStream<..> stream1 = builder.stream(”topic1”);

KStream<..> stream2 = builder.stream(”topic2”);

KStream<..> joined = stream1.leftJoin(stream2, ...);

KTable<..> aggregated = joined.aggregateByKey(...);

aggregated.to(”topic3”);
```



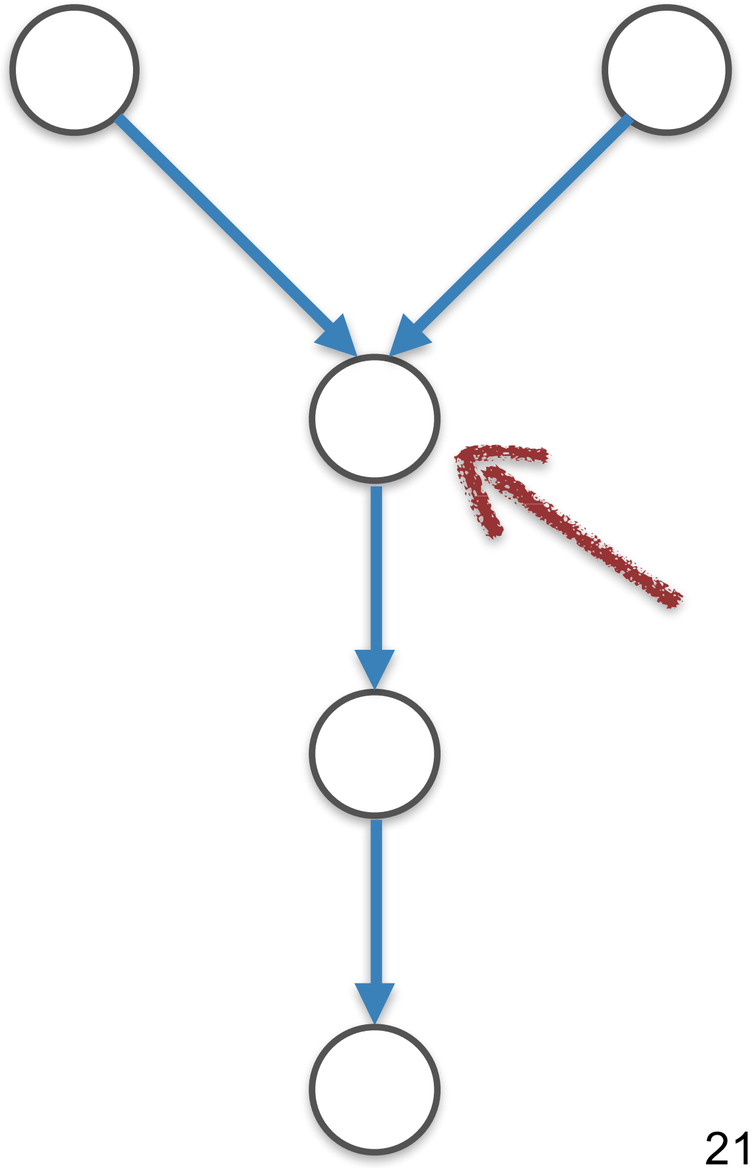
Processor Topology

```
KStream<..> stream1 = builder.stream(”topic1”);
KStream<..> stream2 = builder.stream(”topic2”);
KStream<..> joined = stream1.leftJoin(stream2, ...);
KTable<..> aggregated = joined.aggregateByKey(...);
aggregated.to(”topic3”);
```



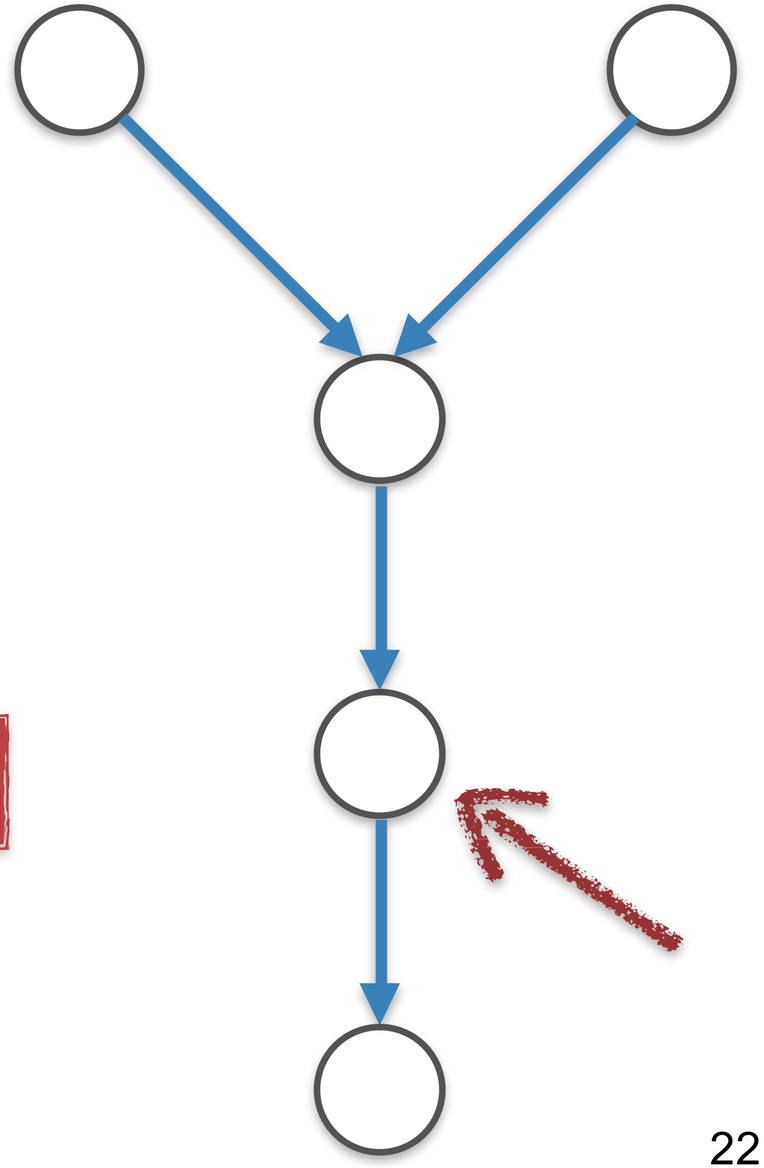
Processor Topology

```
KStream<..> stream1 = builder.stream(”topic1”);

KStream<..> stream2 = builder.stream(”topic2”);

KStream<..> joined = stream1.leftJoin(stream2, ...);

KTable<..> aggregated = joined.aggregateByKey(...);

aggregated.to(”topic3”);
```



Processor Topology

```
KStream<..> stream1 = builder.stream(”topic1”);

KStream<..> stream2 = builder.stream(”topic2”);

KStream<..> joined = stream1.leftJoin(stream2, ...);

KTable<..> aggregated = joined.aggregateByKey(...);

aggregated.to(”topic3”);
```



Processor Topology

```
KStream<..> stream1 = builder.stream(”topic1”);

KStream<..> stream2 = builder.stream(”topic2”);

KStream<..> joined = stream1.leftJoin(stream2, ...);

KTable<..> aggregated = joined.aggregateByKey(...);

aggregated.to(”topic3”);
```