Stream the flow

A presentation about Kafka, Akka and Spark

Sébastien DIAZ

Principles of the demonstration

- Never stop the flow
- Accept to be late
- Accept to lose your data
- Support VVV(VV)
  - Volume
  - Velocity
  - Variety
  - (Veracity)
  - (Value)

Flow Description

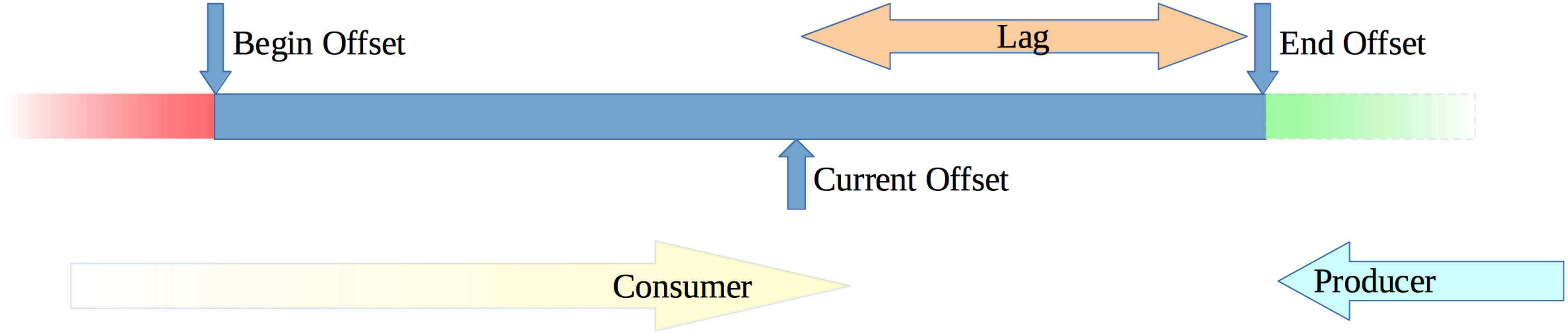
- Combination of more than 45 RSS sources
  - Include: Wall Street Journal, The Economist, Reuters, Bloomberg,….

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>Page</title>
  <link>https://XXX</link>
  <description>Free</description>
  <item>
    <title>Tuto</title>
    <link>https://XXXX</link>
    <description>New</description>
  </item>
</channel>
</rss>
```

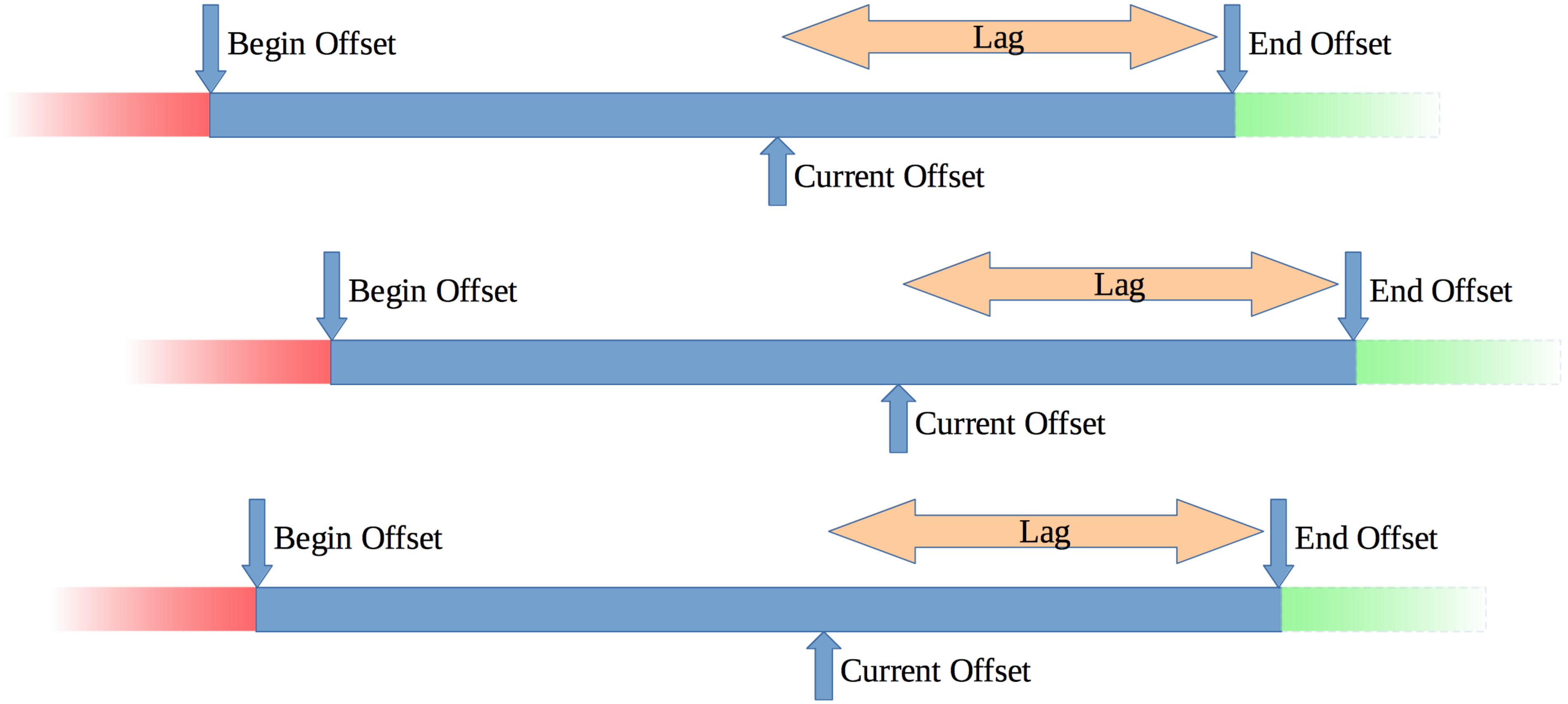
Demonstrated Technologies

- One Broker :
  - Kafka
- Three Streaming Solutions
  - Java 8
  - Akka Stream
  - Spark Structured Streaming
- Languages
  - Java
  - Python
  - Scala

Kafka and Topics



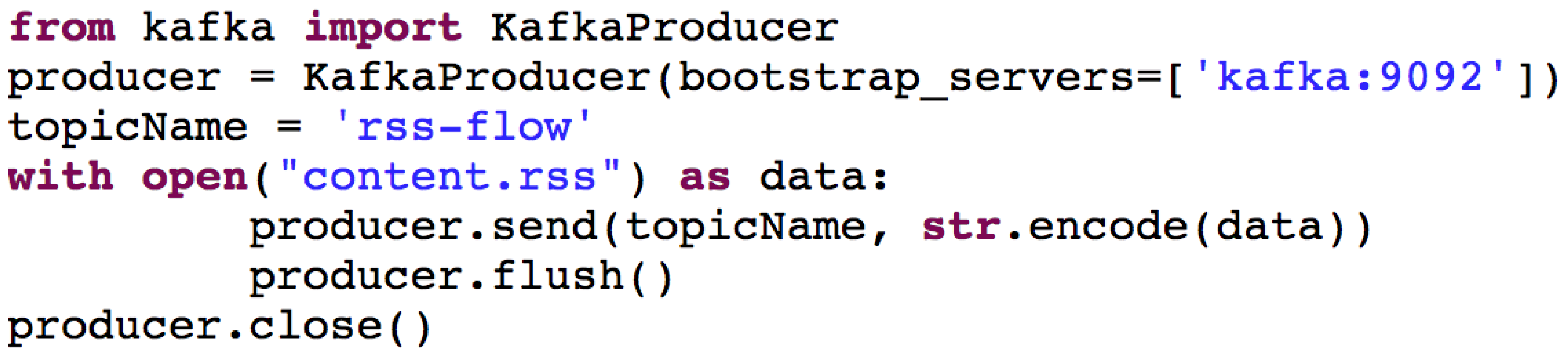
Kafka and Partitions



Kafka Features

- Replication
- Store
- Distributed
- Fault tolerence cluster
- Consumer Group
- ACL
- SSL 1/2 ways
- JAAS Support

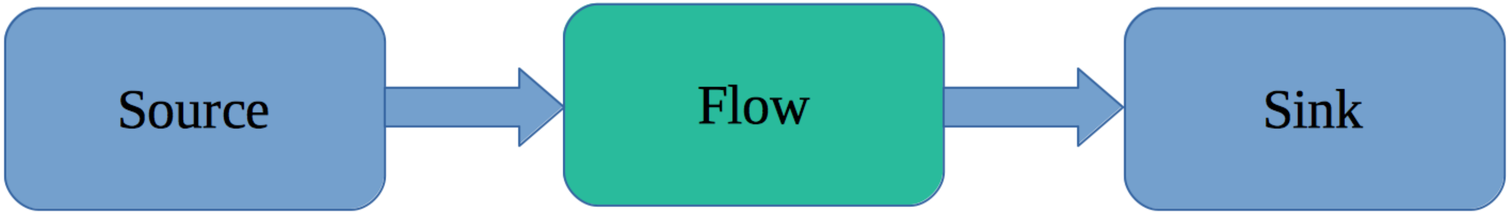
Push the Flow



```
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers=['kafka:9092'])
topicName = 'rss-flow'
with open("content.rss") as data:
         producer.send(topicName, str.encode(data))
         producer.flush()
producer.close()
```

Akka Stream

- Reactive
- Non Blocking
- Asynchronous
- Distributable
- Cluster



MapReduce

- Invented By Google

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

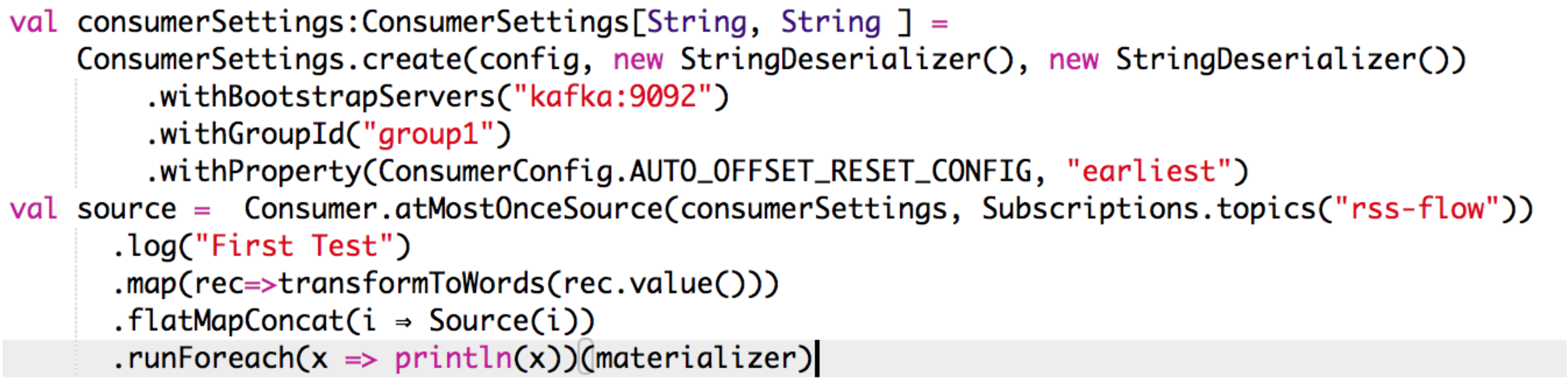
- 2 operations:
  - Map
    - Transformation
  - Reduce
    - Associate map and reduce recursively

Operations

- Basic
  - Map
  - Flat Map
  - Reduce
  - Filter
  - Foreach
  - Collect

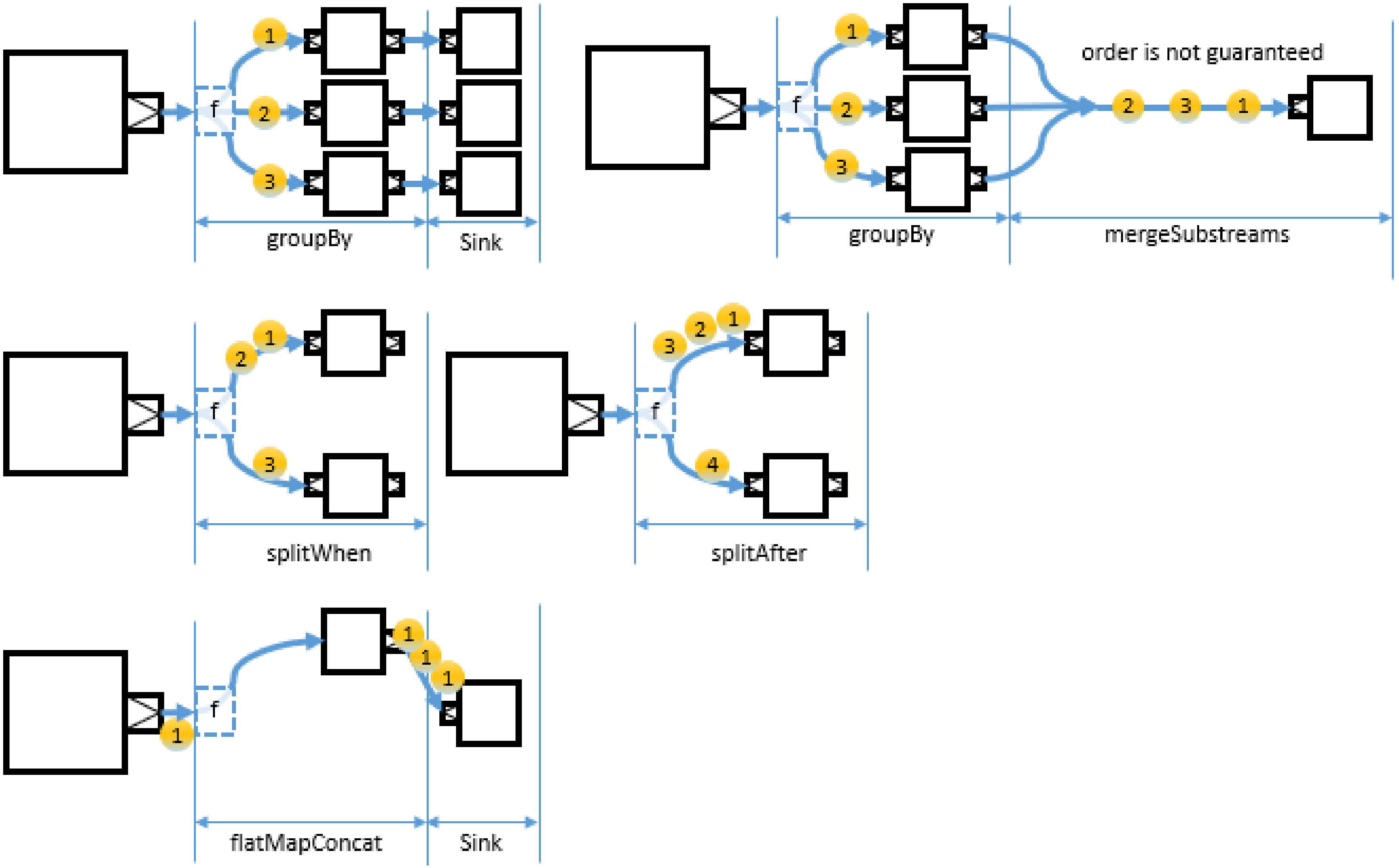
- Advanced
  - Detach
  - Drop
  - Fold
  - Grouped
  - Limit
  - Scan
  - Take
  - Watch
  - Lazy

Open the stream

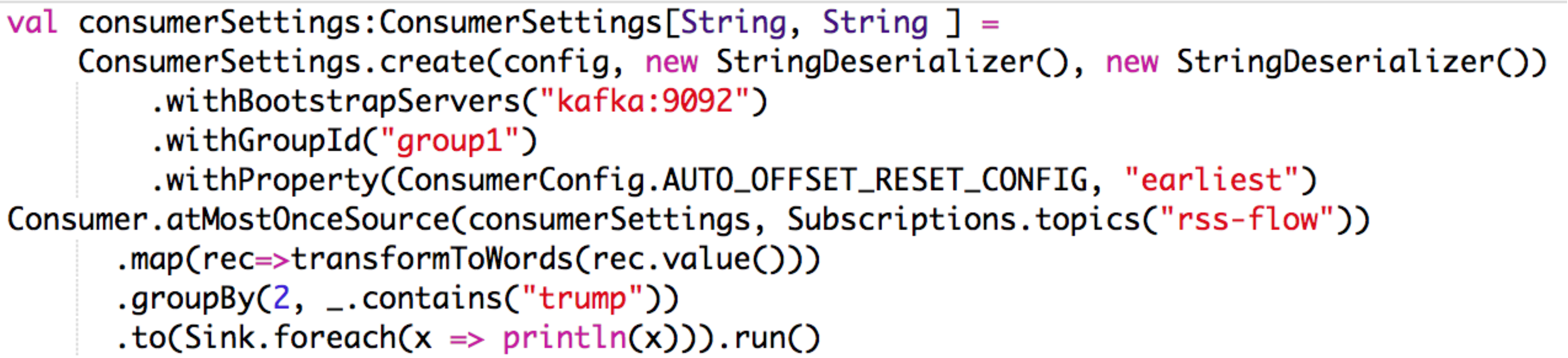


```
val consumerSettings:ConsumerSettings[String, String ] =
    ConsumerSettings.create(config, new StringDeserializer(), new StringDeserializer())
        .withBootstrapServers("kafka:9092")
        .withGroupId("group1")
        .withProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest")
val source =  Consumer.atMostOnceSource(consumerSettings, Subscriptions.topics("rss-flow"))
      .log("First Test")
      .map(rec=>transformToWords(rec.value()))
      .flatMapConcat(i ⇒ Source(i))
      .runForeach(x => println(x))(materializer)
```

Substream

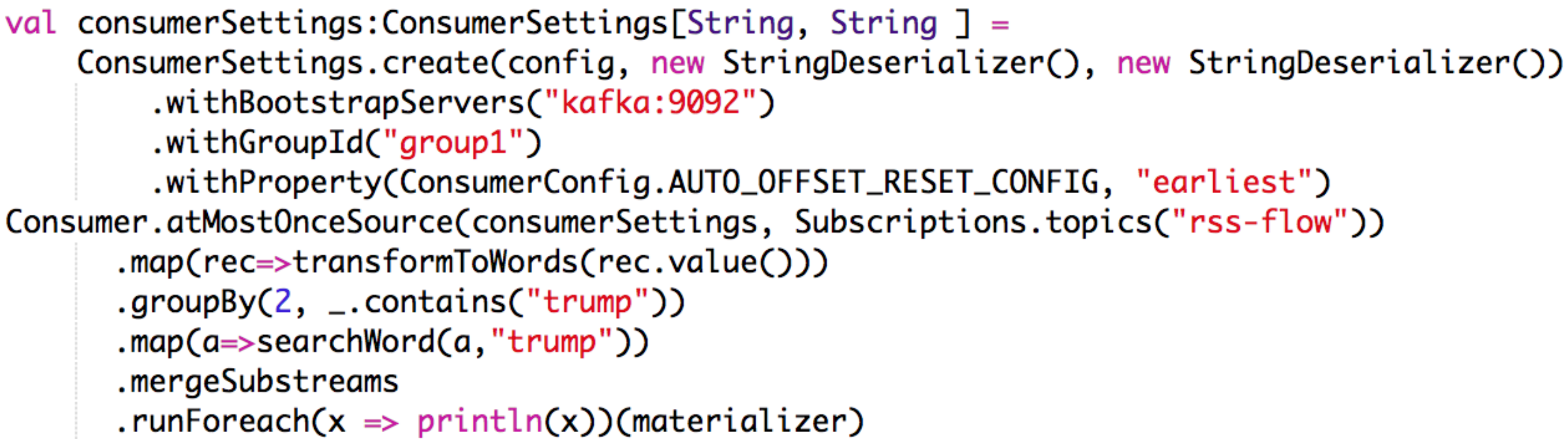

f
1
2
3
groupBy
Sink
f
1
2
3
order is not guaranteed
2
3
1
groupBy
mergeSubstreams
f
1
2
3
splitWhen
3
2
1
f
4
splitAfter
1
f
1
1
1
flatMapConcat
Sink

Check GroupBy



```
val consumerSettings:ConsumerSettings[String, String ] =
    ConsumerSettings.create(config, new StringDeserializer(), new StringDeserializer())
        .withBootstrapServers("kafka:9092")
        .withGroupId("group1")
        .withProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest")
Consumer.atMostOnceSource(consumerSettings, Subscriptions.topics("rss-flow"))
      .map(rec=>transformToWords(rec.value()))
      .groupBy(2, _.contains("trump"))
      .to(Sink.foreach(x => println(x))).run()
```

Check GroupBy and merge

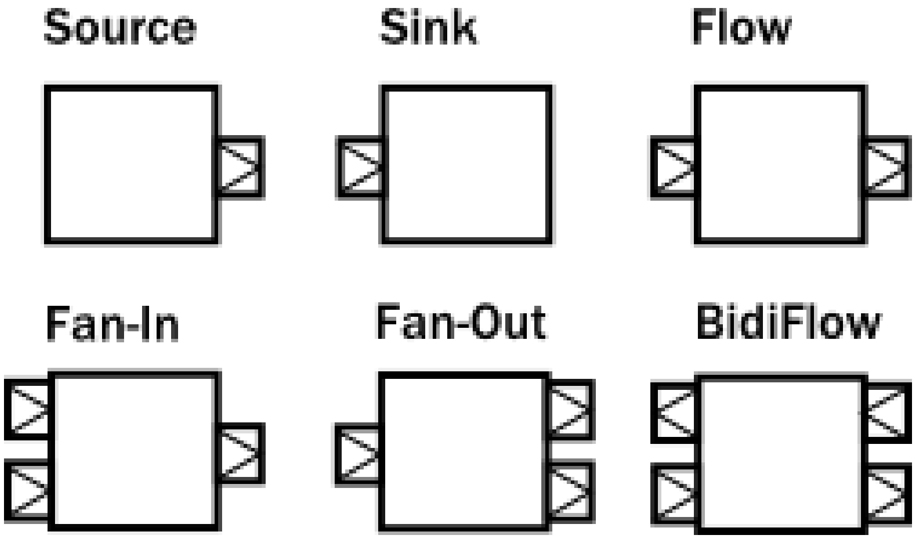


```
val consumerSettings:ConsumerSettings[String, String ] =
    ConsumerSettings.create(config, new StringDeserializer(), new StringDeserializer())
        .withBootstrapServers("kafka:9092")
        .withGroupId("group1")
        .withProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest")
Consumer.atMostOnceSource(consumerSettings, Subscriptions.topics("rss-flow"))
      .map(rec=>transformToWords(rec.value()))
      .groupBy(2, _.contains("trump"))
      .map(a=>searchWord(a,"trump"))
      .mergeSubstreams
      .runForeach(x => println(x))(materializer)
```

Graph on Flow

- Fan In
  - Merge
  - Zip
  - Concat

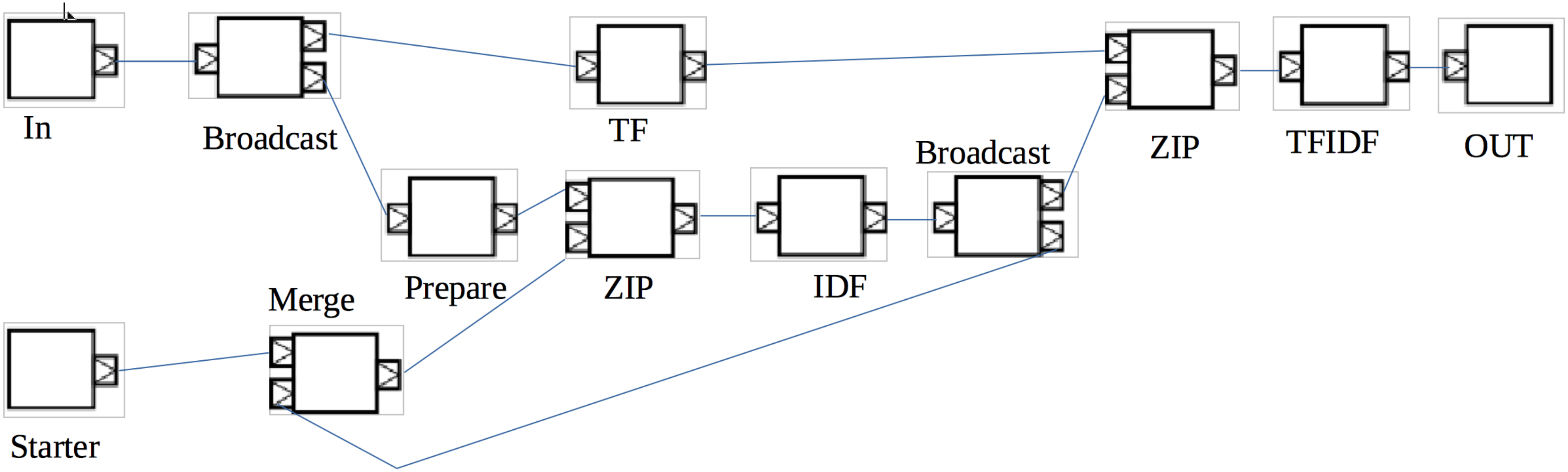
- Fan Out
  - Broadcast
  - Balance
  - Unzip



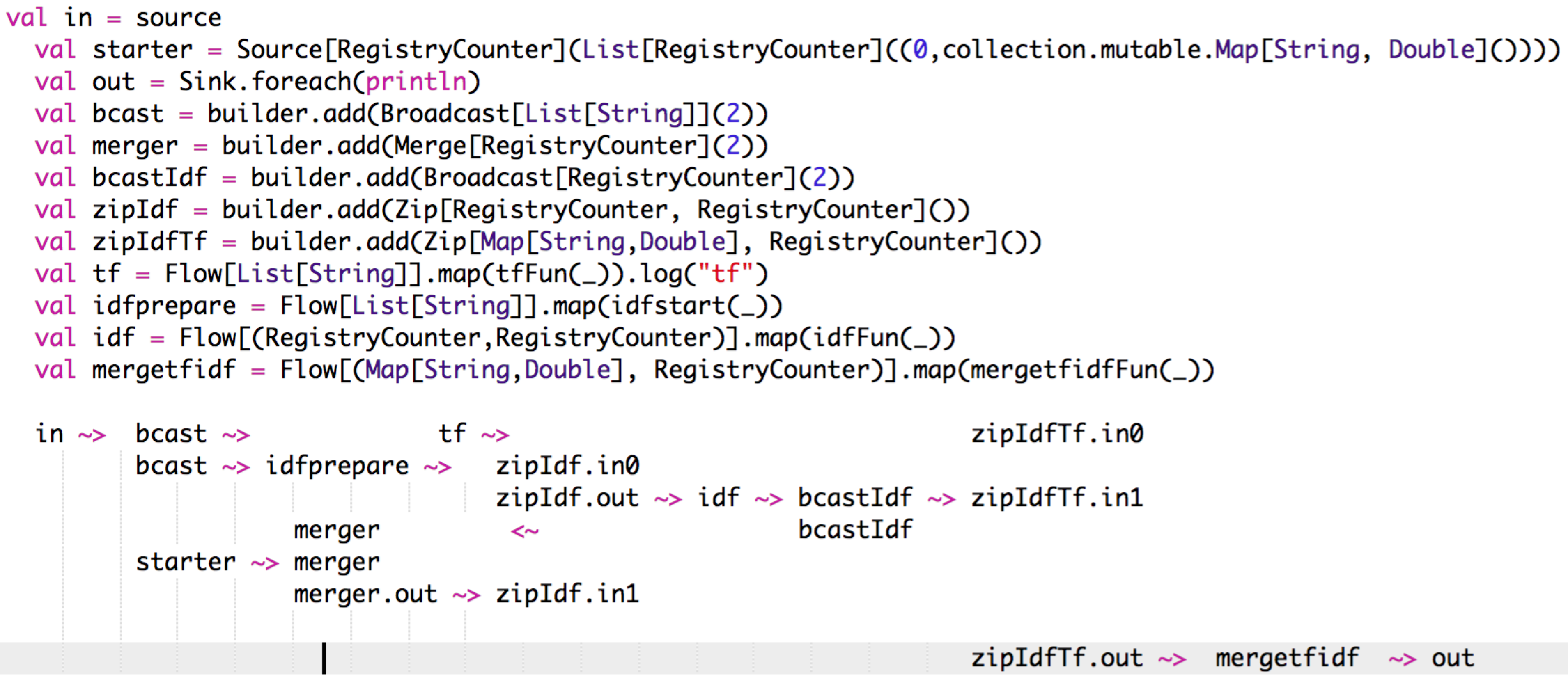
TF IDF as Transformation

- TF : Term Frequency
  - The raw count of a term in a document
- IDF : Inverse Document Frequency
  - Common or rare across all documents

Graph of TF IDF’



Graph Code



```
val in = source
  val starter = Source[RegistryCounter](List[RegistryCounter]((0,collection.mutable.Map[String, Double]())))
  val out = Sink.foreach(println)
  val bcast = builder.add(Broadcast[List[String]](2))
  val merger = builder.add(Merge[RegistryCounter](2))
  val bcastIdf = builder.add(Broadcast[RegistryCounter](2))
  val zipIdf = builder.add(Zip[RegistryCounter, RegistryCounter]())
  val zipIdfTf = builder.add(Zip[Map[String,Double], RegistryCounter]())
  val tf = Flow[List[String]].map(tfFun(_)).log("tf")
  val idfprepare = Flow[List[String]].map(idfstart(_))
  val idf = Flow[(RegistryCounter,RegistryCounter)].map(idfFun(_))
  val mergetfidf = Flow[(Map[String,Double], RegistryCounter)].map(mergetfidfFun(_))

  in ~>  bcast ~>             tf ~>                                zipIdfTf.in0
         bcast ~> idfprepare ~>   zipIdf.in0
                                  zipIdf.out ~> idf ~> bcastIdf ~> zipIdfTf.in1
                    merger         <~                  bcastIdf
         starter ~> merger
                    merger.out ~> zipIdf.in1

                                                                   zipIdfTf.out ~>  mergetfidf  ~> out
```

Errors and Recovery

- mapError : Transform the error
- recover : Transform and log the error
- revoverWith : Switch to another Source
- recoverWithRetries : Switch to another Source with retry
- Delayed restart

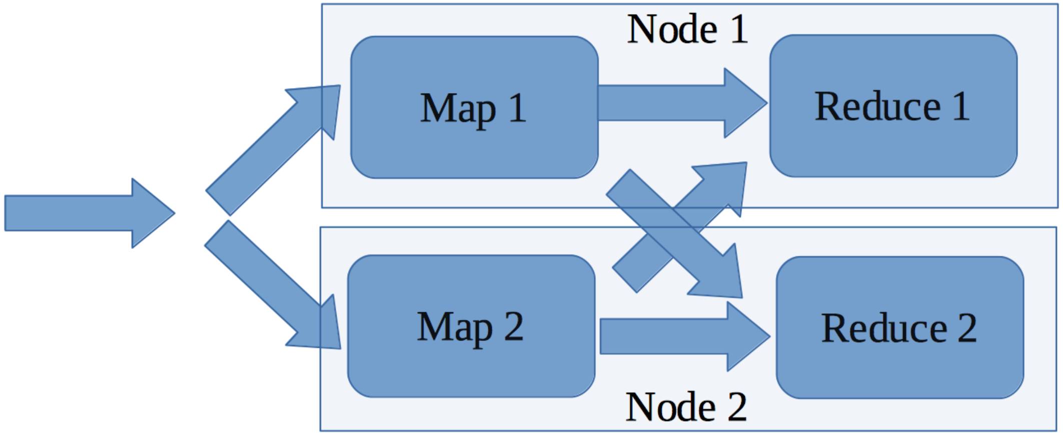
When It’s huge or complex !

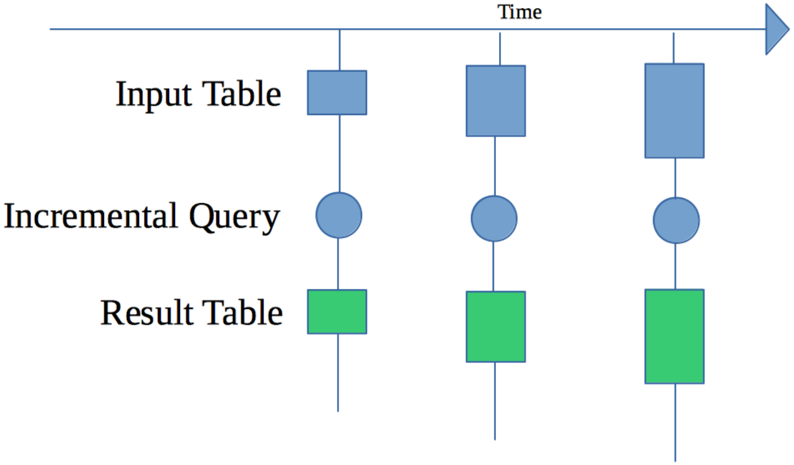
- New strategies
  - Windows based
  - Remove (out of the time,…)
  - Add Batch
- New platform
  - Spark
  - Hadoop

Meteosat Second Generation :
> 100 TB / Day

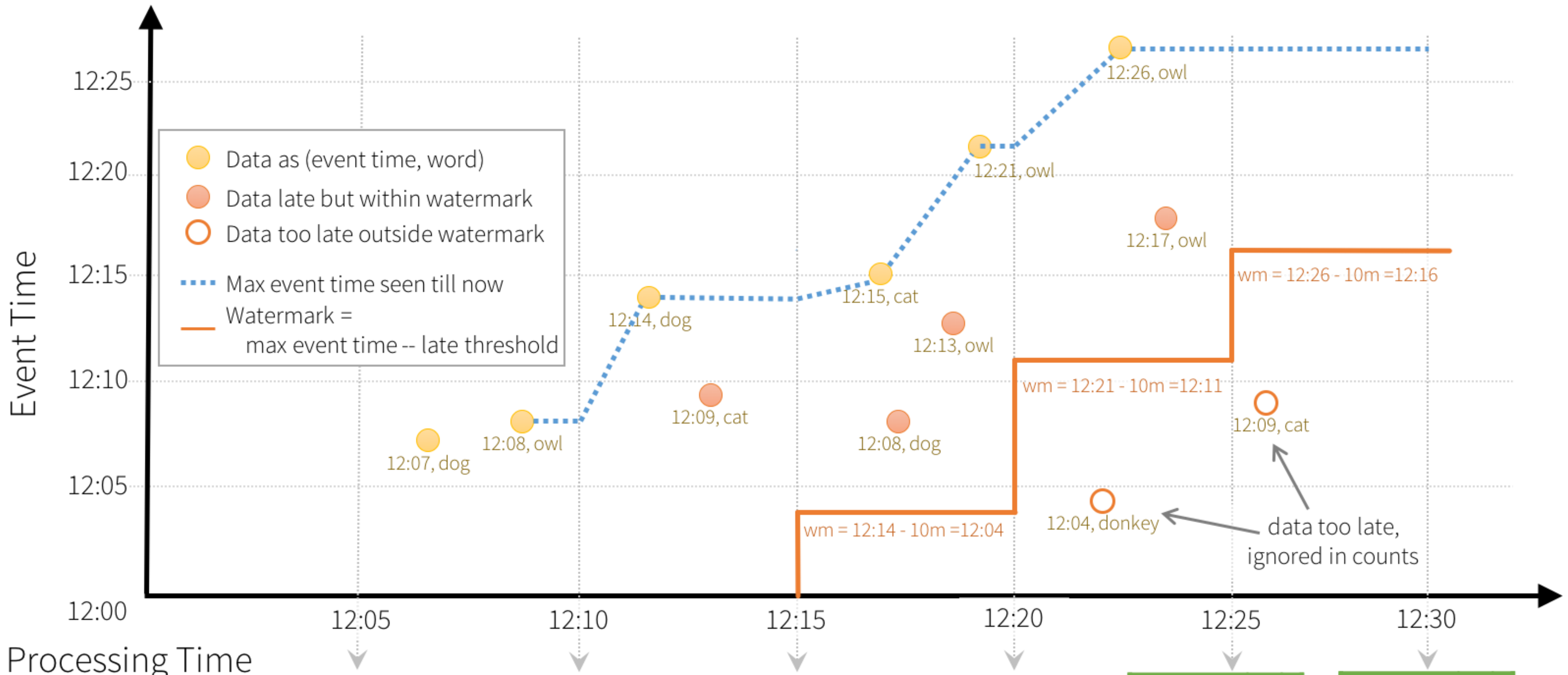
Spark Structered Streaming

- Fast
- Scalable
- Fault – Tolerence
- SQL Engine
- Incremental or Continous





Windows and Watermark



Processing Time
with 5 min triggers

partial counts for window 12:00 - 12:10 maintained as internal state while waiting for late data, so not yet added to result table

12:00 - 12:10	owl	1
12:00 - 12:10	cat	1
12:00 - 12:10	dog	2

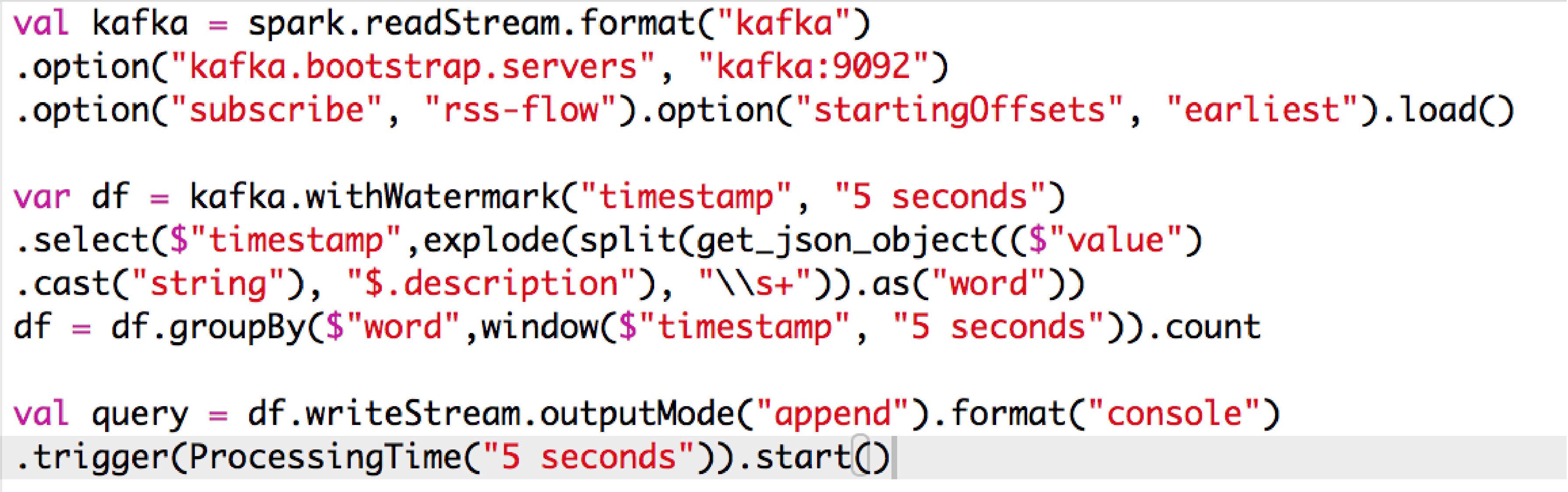
12:00 - 12:10	owl	1
12:00 - 12:10	cat	1
12:00 - 12:10	dog	2
12:05 - 12:15	owl	2
12:05 - 12:15	cat	2
12:05 - 12:15	dog	3

final counts for 12:00 - 12:10 added to table when watermark > 12:10, late data counted, and intermediate state for window dropped

Watermarking in **Windowed Grouped Aggregation** with **Append Mode**

Result Tables after each trigger

Simple Incremental Streaming

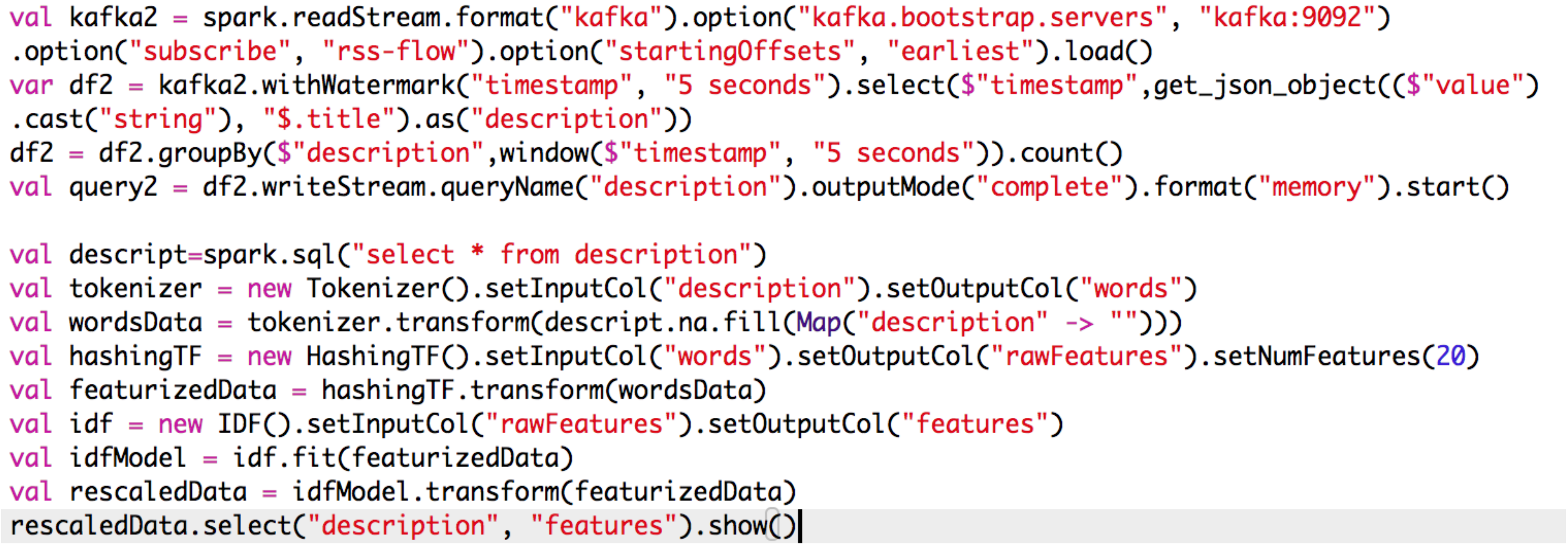


```
val kafka = spark.readStream.format("kafka")
.option("kafka.bootstrap.servers", "kafka:9092")
.option("subscribe", "rss-flow").option("startingOffsets", "earliest").load()

var df = kafka.withWatermark("timestamp", "5 seconds")
.select($"timestamp",explode(split(get_json_object(($"value")
.cast("string"), "$.description"), "\\s+")).as("word"))
df = df.groupBy($"word",window($"timestamp", "5 seconds")).count

val query = df.writeStream.outputMode("append").format("console")
.trigger(ProcessingTime("5 seconds")).start()
```

And TF IDF



```
val kafka2 = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "kafka:9092")
.option("subscribe", "rss-flow").option("startingOffsets", "earliest").load()
var df2 = kafka2.withWatermark("timestamp", "5 seconds").select($"timestamp",get_json_object(($"value")
.cast("string"), "$.title").as("description"))
df2 = df2.groupBy($"description",window($"timestamp", "5 seconds")).count()
val query2 = df2.writeStream.queryName("description").outputMode("complete").format("memory").start()

val descript=spark.sql("select * from description")
val tokenizer = new Tokenizer().setInputCol("description").setOutputCol("words")
val wordsData = tokenizer.transform(descript.na.fill(Map("description" -> "")))
val hashingTF = new HashingTF().setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(20)
val featurizedData = hashingTF.transform(wordsData)
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)
val rescaledData = idfModel.transform(featurizedData)
rescaledData.select("description", "features").show()
```

THANK YOU

SMACK Architecture

- Spark : Processing
- Mesos : Cluster Management
- Akka : Actor model
- Cassandra : Nosql and Big Table
- Kafka : Stream