
DigitalHouse > Coding School

# DATA SCIENCE

Procesamiento Distribuído con Apache Spark

## OBJETIVOS DE LA CLASE

1. Introduction to stream processing
2. Spark Structured Streaming
3. Spark MLlib
4. Práctica

# Stream Processing

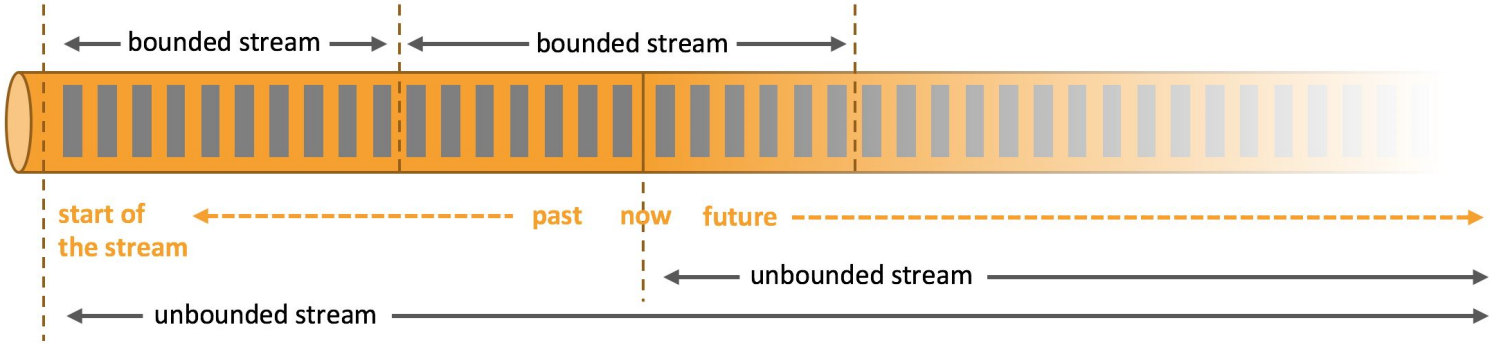
# Stream Processing

**Los datos se origin como un flujo constante**

Procesamiento **batch** => *datasets finitos*

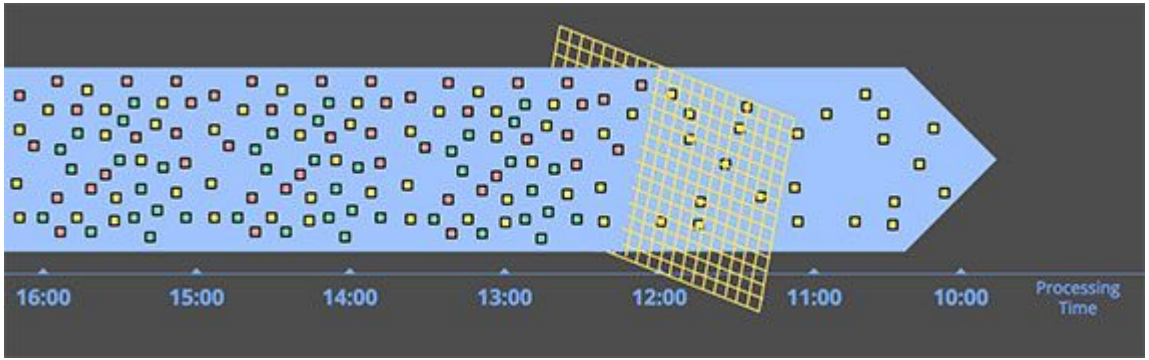
Procesamiento de **streams** => *datasets infinitos*

Stream processing se refiere a procesar el dato en el instante que se recibe de manera contínua.

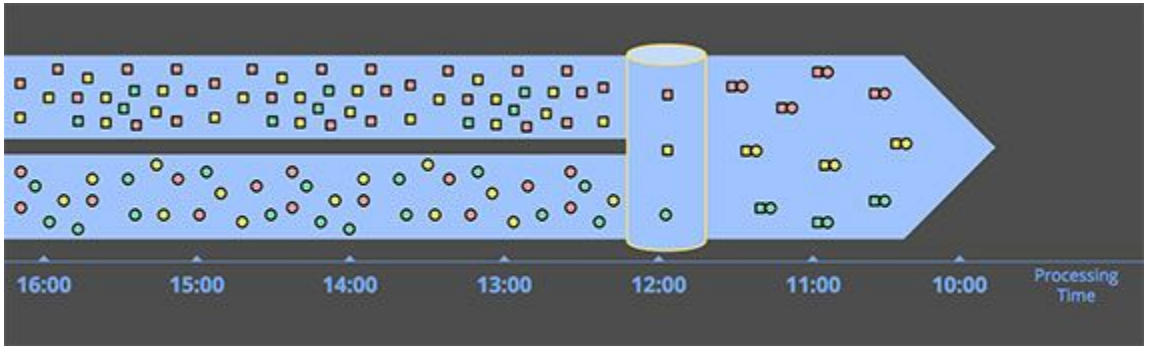



# Operaciones sobre streams

FILTER

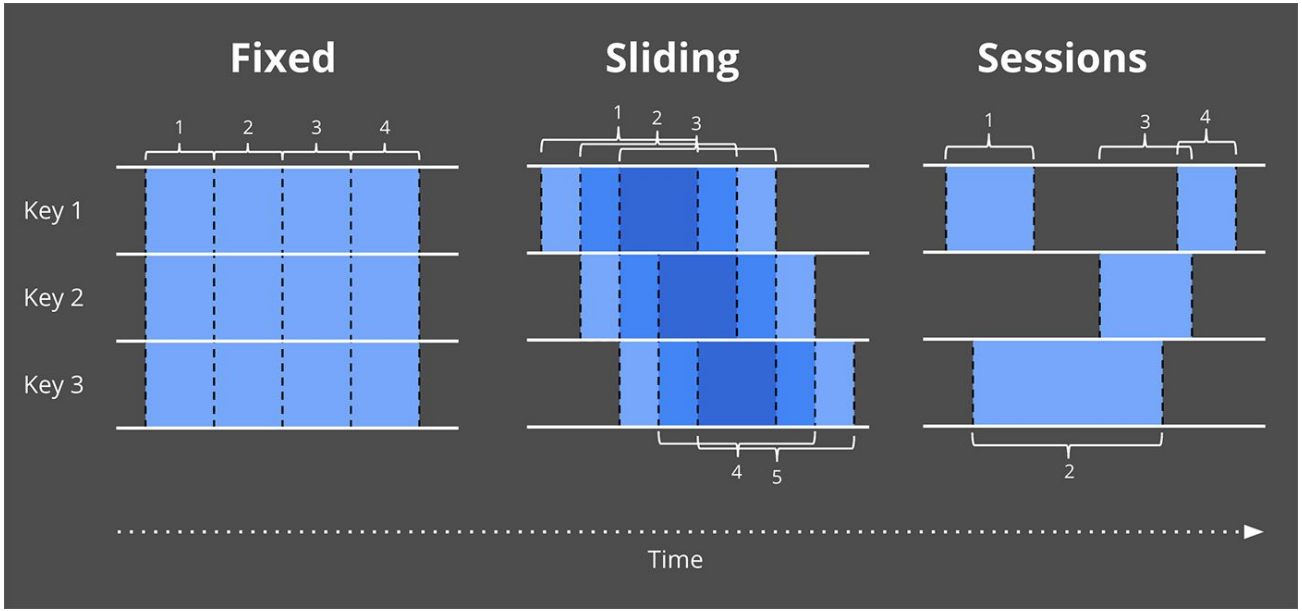



INNER JOIN



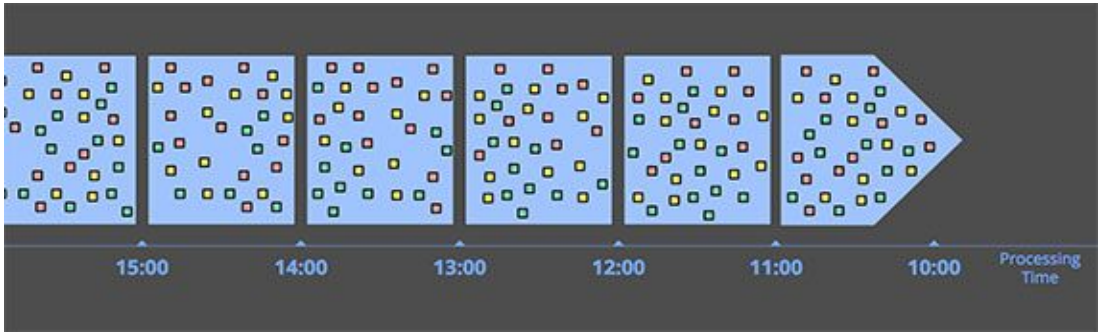
# Windowing de un stream

Consiste en dividir un stream en partes discretas para computar métricas con contexto (count, min, max, median, etc.)

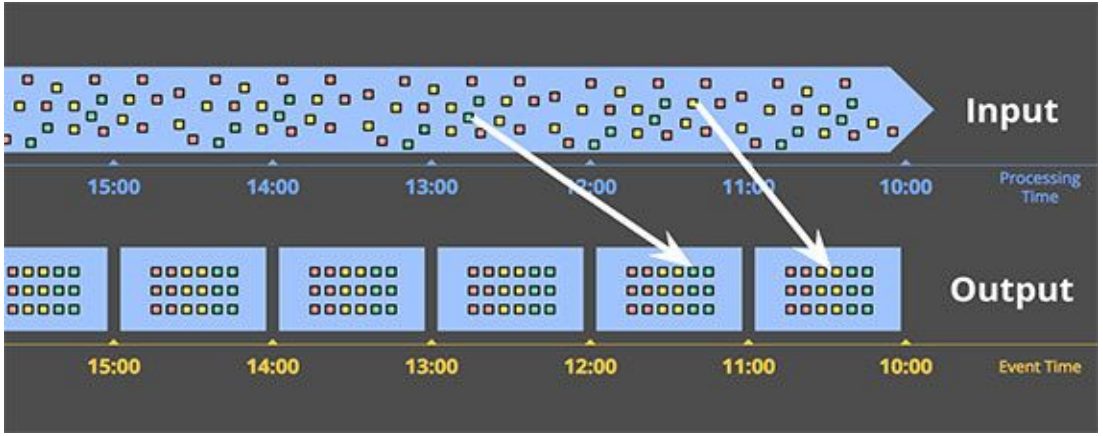


# Windowing

PROCESSING TIME WINDOW

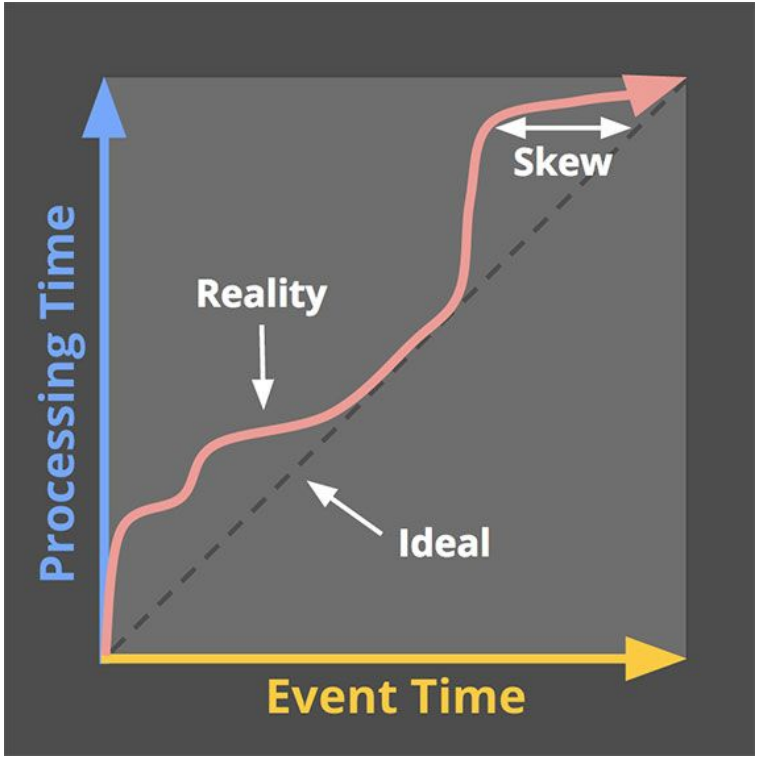



EVENT TIME WINDOW

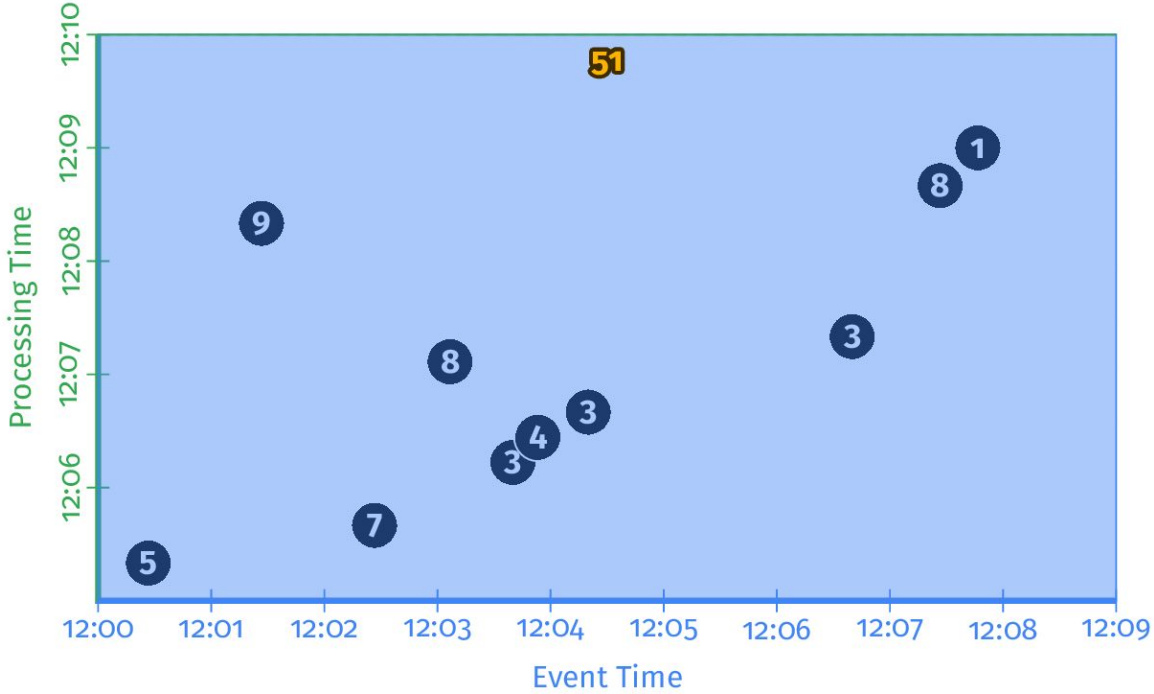


# Tiempo del eventos vs Tiempo de Procesamiento

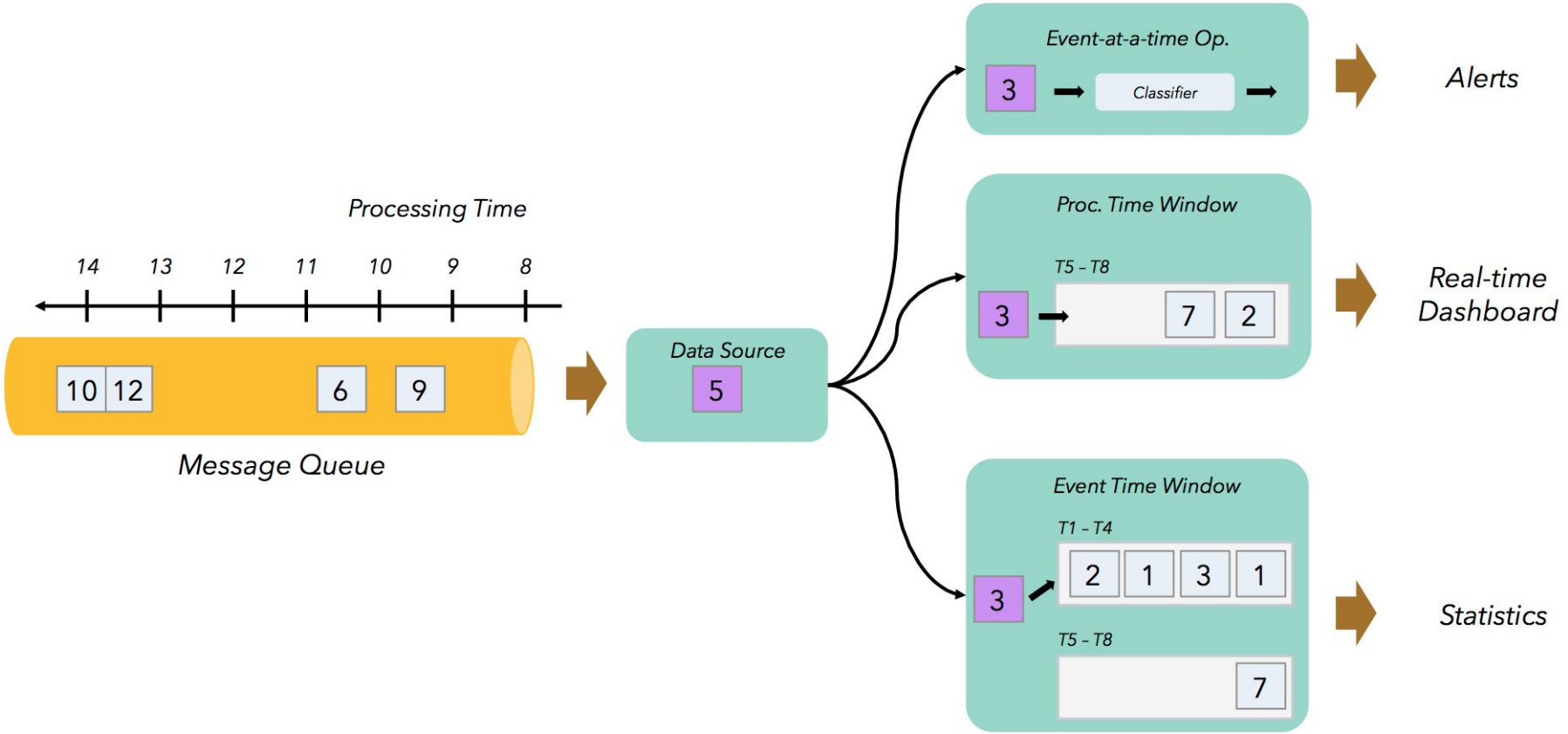
- **Tiempo del evento:** momento en el tiempo donde ocurrió el evento
- **Tiempo de procesamiento:** momento en el cual el sistema recibe el dato del evento y lo procesa.
- **Idealmente iguales** pero por lo general tienen diferencias.



# Tiempo del eventos vs Tiempo de Procesamiento

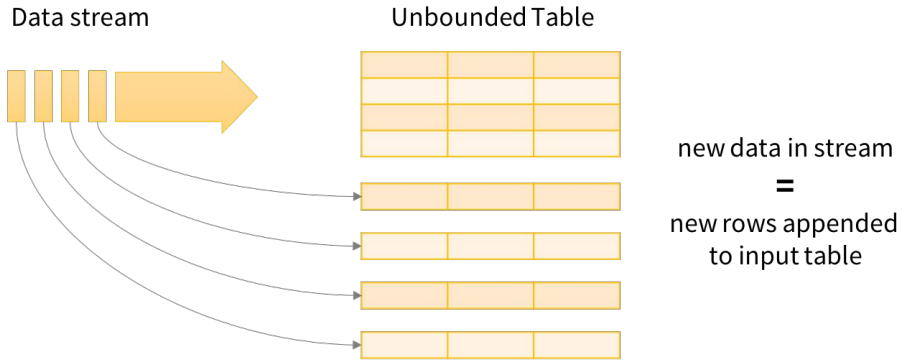


# Tipos de Procesamiento



# Spark Structured Streaming

# Utilizando Spark Streaming




Data stream as an unbounded Input Table

```
// Read data once from an S3 location
val inputDF = spark.read.json("s3://logs")

// Do operations using the standard DataFrame API and write to MySQL
inputDF.groupBy($"action", window($"time", "1 hour")).count()
       .writeStream.format("jdbc")
       .save("jdbc:mysql//…")
```

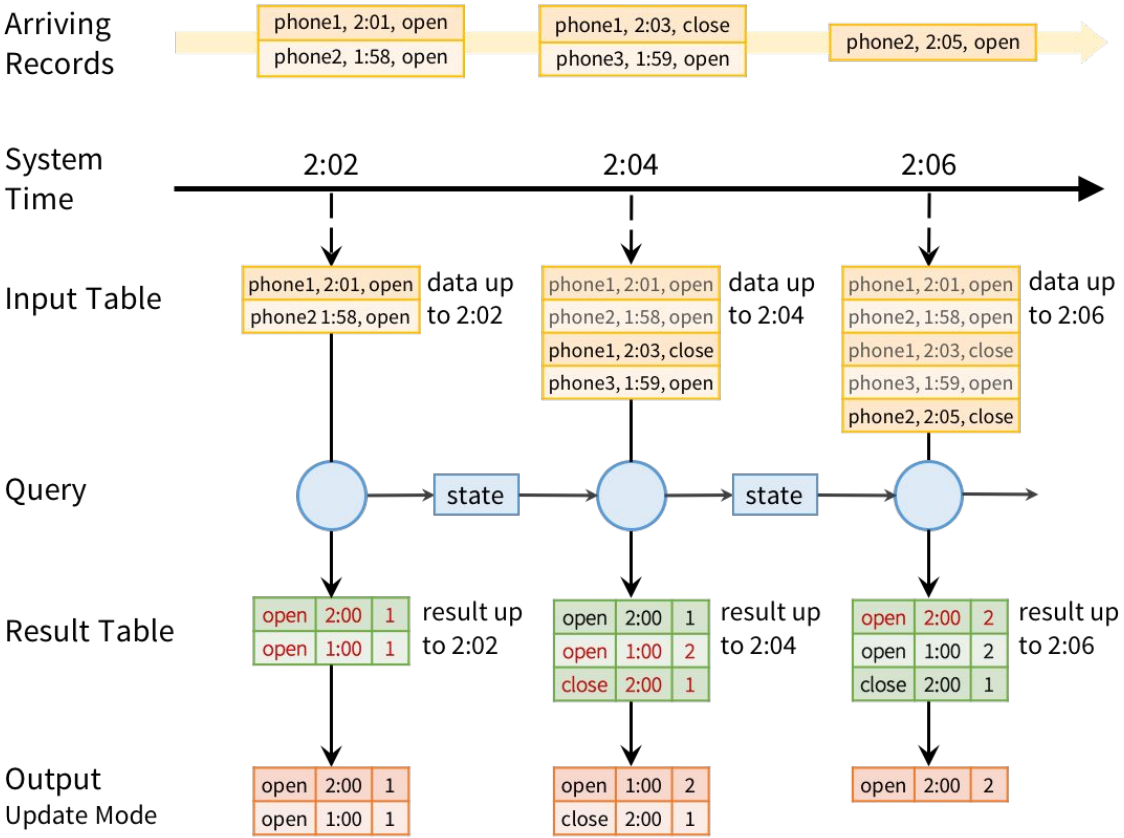
```
// Read data continuously from an S3 location
val inputDF = spark.readStream.json("s3://logs")

// Do operations using the standard DataFrame API and write to MySQL
inputDF.groupBy($"action", window($"time", "1 hour")).count()
       .writeStream.format("jdbc")
       .start("jdbc:mysql//…")
```

# Structured Streaming

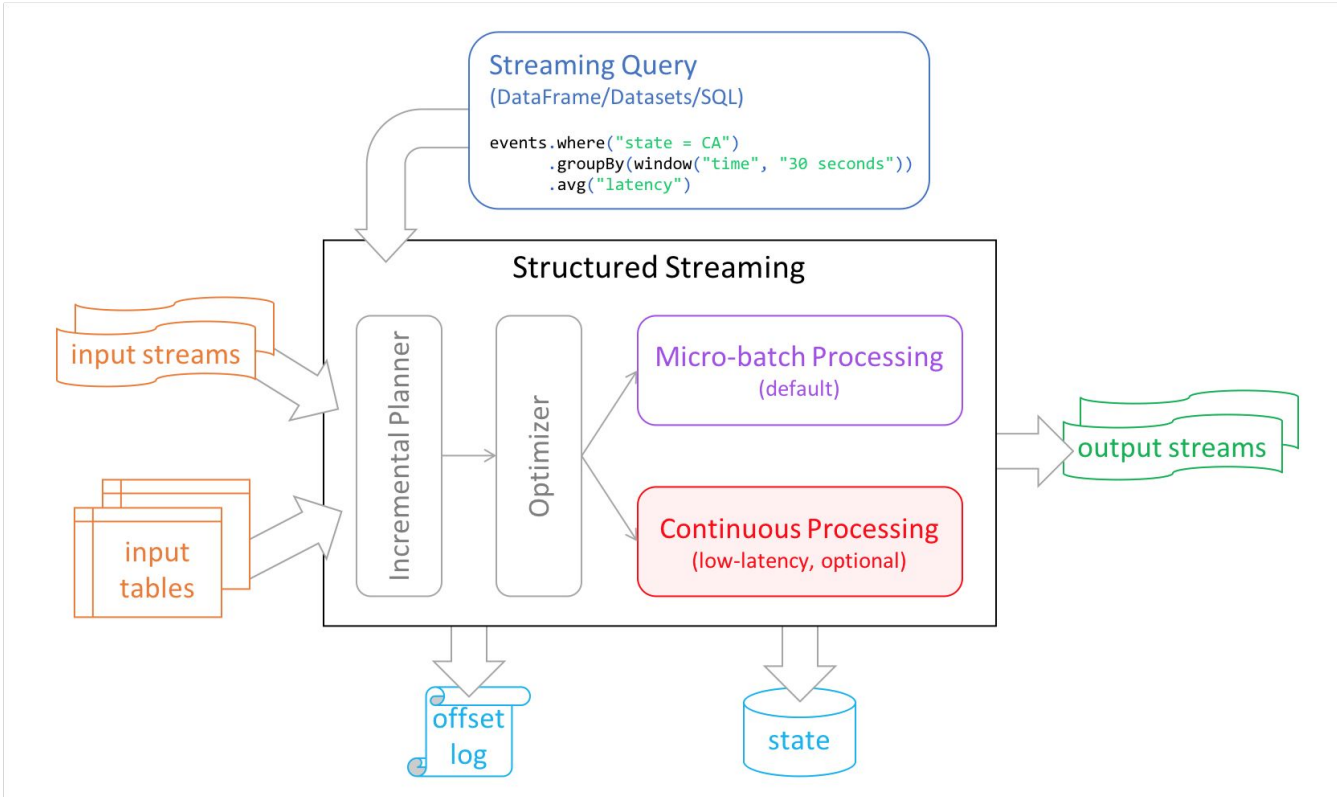


- Spark **se ocupa de mantener la consistencia** entre el ingreso de datos, el procesamiento y la escritura al destino.
- Spark necesita **fuentes de datos donde los mismos pueden volver a ser leídos** una vez consumidos (Kafka, Kinesis, HDFS).
- Se describe la query a correr **similar a una operación sobre Dataframes** y Spark se ocupa de correrla constantemente.

# Ejemplo


Arriving Records
phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
phone2, 2:05, open
System Time
2:02
2:04
2:06
Input Table
phone1, 2:01, open
phone2 1:58, open
data up to 2:02
phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
data up to 2:04
phone1, 2:01, open
phone2, 1:58, open
phone1, 2:03, close
phone3, 1:59, open
phone2, 2:05, close
data up to 2:06
Query
state
state
Result Table
open 2:00 1
open 1:00 1
result up to 2:02
open 2:00 1
open 1:00 2
close 2:00 1
result up to 2:04
open 2:00 2
open 1:00 2
close 2:00 1
result up to 2:06
Output
Update Mode
open 2:00 1
open 1:00 1
open 1:00 2
close 2:00 1
open 2:00 2

# Modelo de ejecución



# Modos de Output

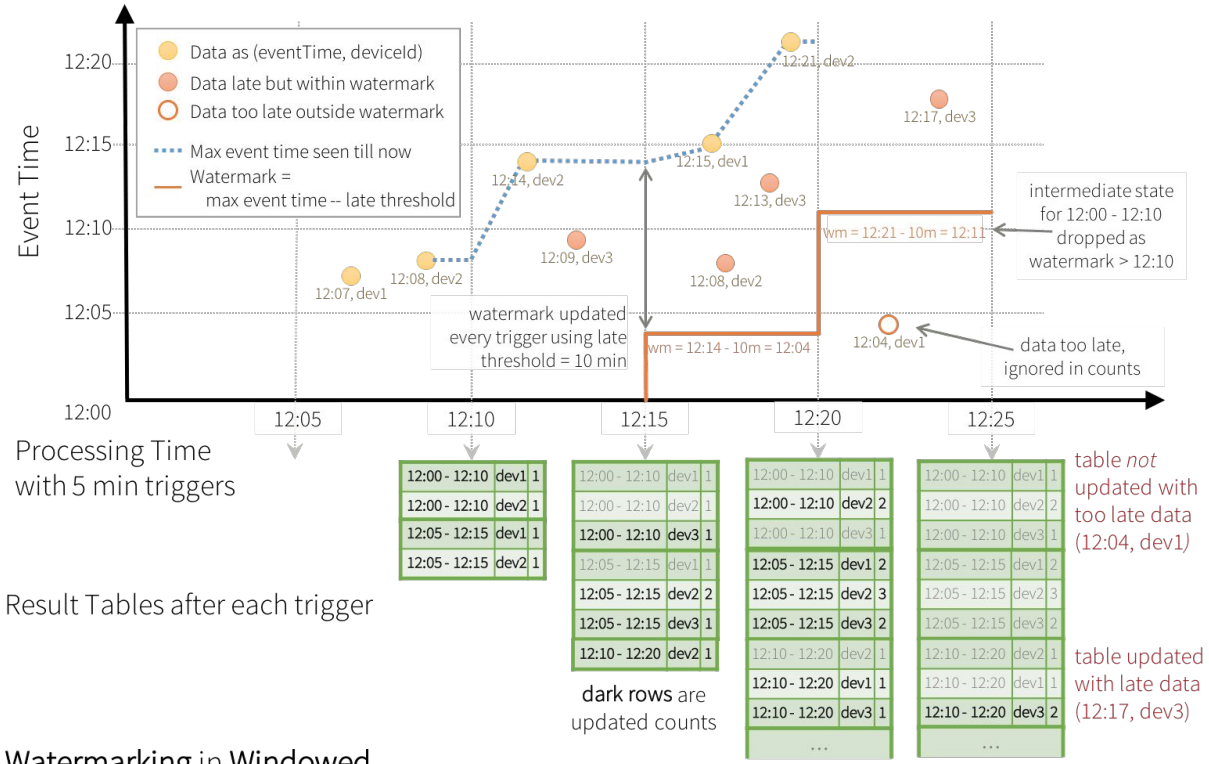
- **Append:** solamente las filas nuevas agregadas a la tabla de procesamiento serán escritas al storage externo.
- **Completo:** toda la tabla de resultados en memoria será enviada al storage externo en cada trigger.
- **Update:** solamente las filas que tengan cambios serán actualizadas en el sistema externo. Este modo solamente funciona con destinos que permitan updates como MySQL.

# Watermarking

Watermarking es una forma en la cual herramientas como Spark simbolizan que no van a procesar mensajes más viejos que cierto tiempo

```
windowedCountsDF = \
  eventsDF \
    .withWatermark("eventTime", "10 minutes") \
    .groupBy(
      "deviceId",
      window("eventTime", "10 minutes", "5 minutes")) \
    .count()
```

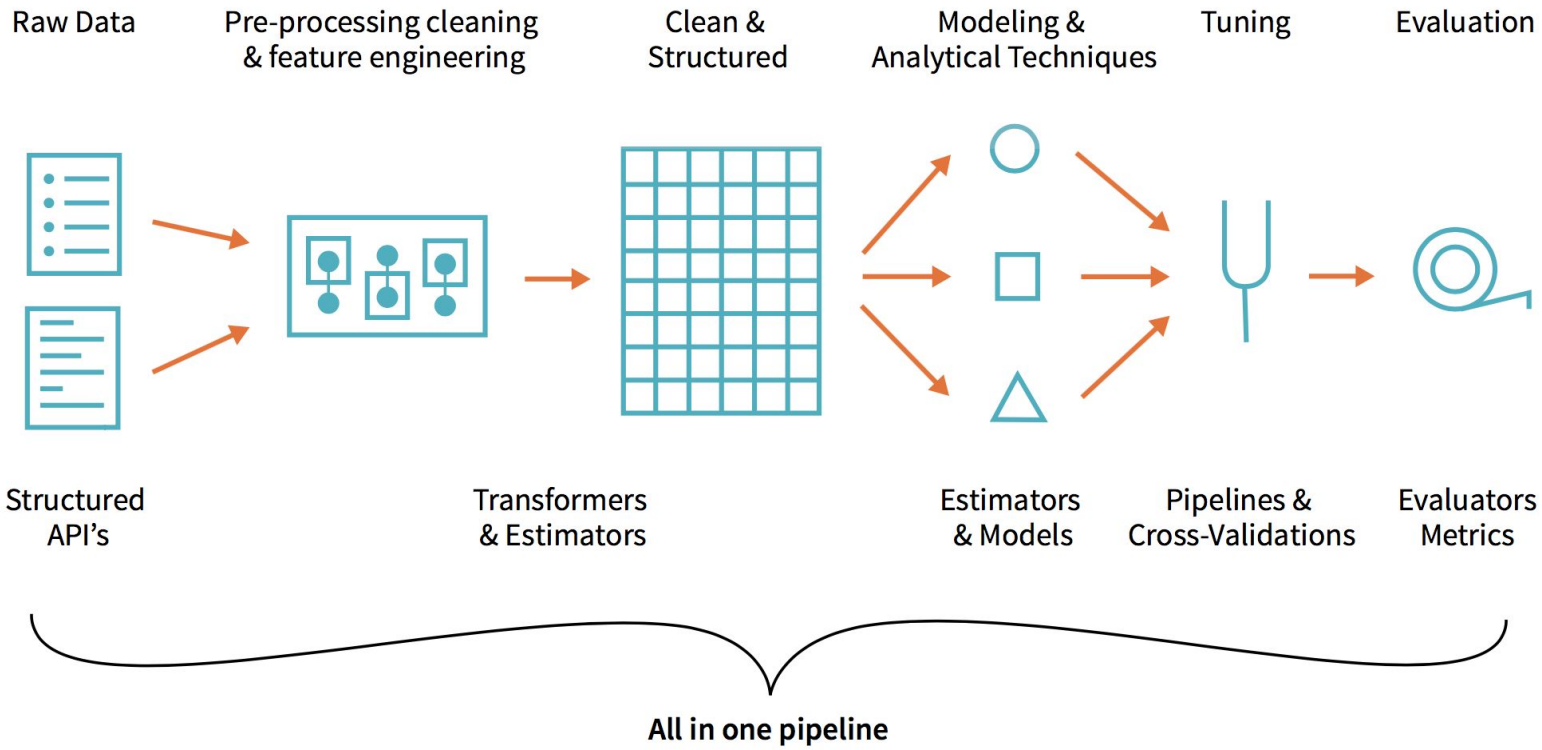
# Ejemplo con Watermarking




**Watermarking** in **Windowed Grouped Aggregation**

# Spark MLlib

# Spark MLlib


Raw Data
Pre-processing cleaning & feature engineering
Clean & Structured
Modeling & Analytical Techniques
Tuning
Evaluation
Structured API's
Transformers & Estimators
Estimators & Models
Pipelines & Cross-Validations
Evaluators Metrics
All in one pipeline

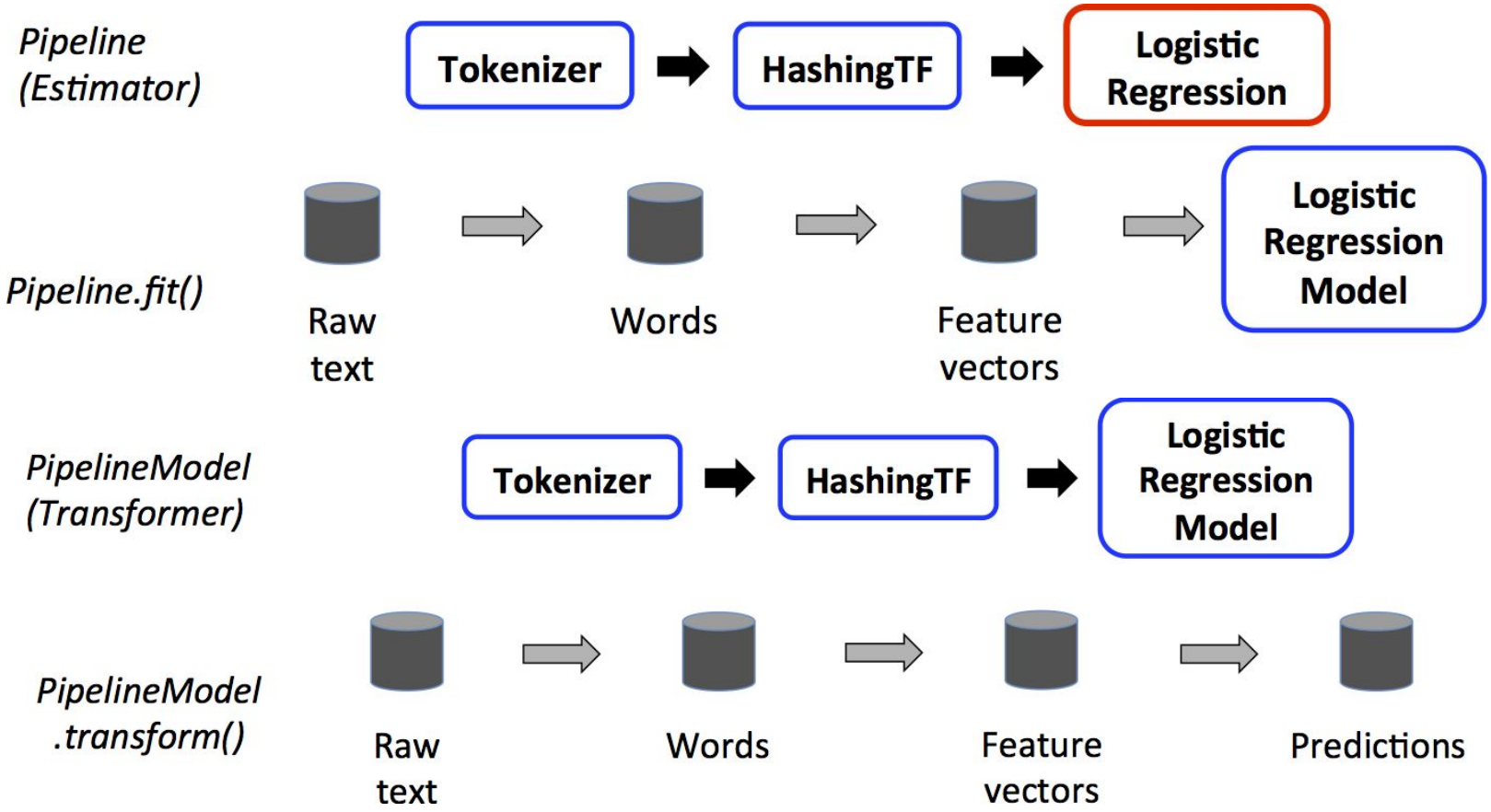
# Spark MLlib

- MLlib estandariza muchos algoritmos y técnicas de feature engineering para combinarlos en un mismo workflow denominado **Pipeline**.
- Cuando procesamos texto para predicción tenemos que:
  - Dividir en palabras el texto de cada documento.
  - Convertir estas palabras en un vector de features.
  - Entrenar un modelo con el vector y los labels asociados.
  - Realizar predicciones sobre nuevos documentos.

# Spark MLlib

- En MLlib un workflow se representa como un objeto Pipeline con muchos PipelineStages (Transformers y Estimators).
- **Transformer:** es un algoritmo que transforma un DataFrame en otro con el método transform().
- **Estimator:** algoritmo que puede ser entrenado con los datos con el método fit() y da como resultado un Transformer.
- **Parameter:** tanto los Estimators como los Transformers tienen parámetros que permiten configurarlos.

# Spark MLlib


Pipeline
(Estimator)
Tokenizer
HashingTF
Logistic Regression
Pipeline.fit()
Raw text
Words
Feature vectors
Logistic Regression Model
PipelineModel
(Transformer)
Tokenizer
HashingTF
Logistic Regression Model
PipelineModel
.transform()
Raw text
Words
Feature vectors
Predictions

# Práctica