



Spark Streaming

2021v3

Contenido

1. Conceptos en Streaming

2. Streaming clásico: DStreams

- Principios de funcionamiento
- Operativa básica
- Entrada/salida
- Transformaciones
- Robustez
- Ventanas

Contenido

3. Streaming estructurado

- Principios básicos
- Entrada/Salida
- Fuentes de datos externas: Kafka
- Procesado de ventanas
- Semántica de entregas
- Funcionalidad extra
- Procesado continuo

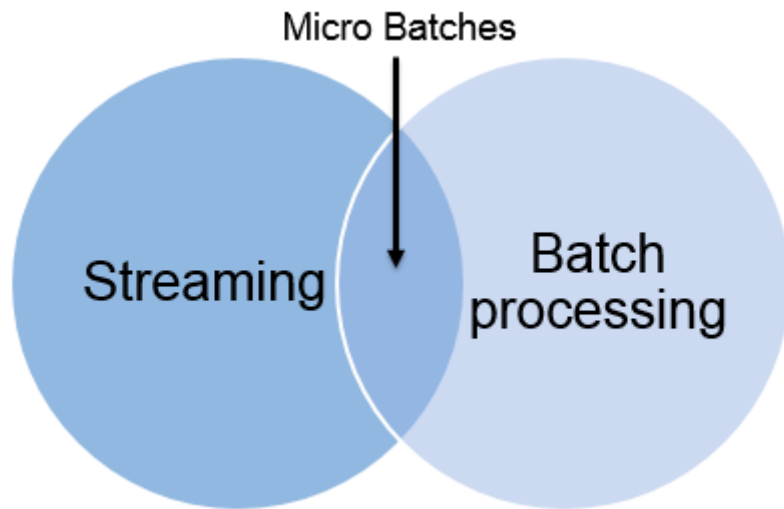
4. Machine Learning para Streaming

Conceptos básicos de streaming

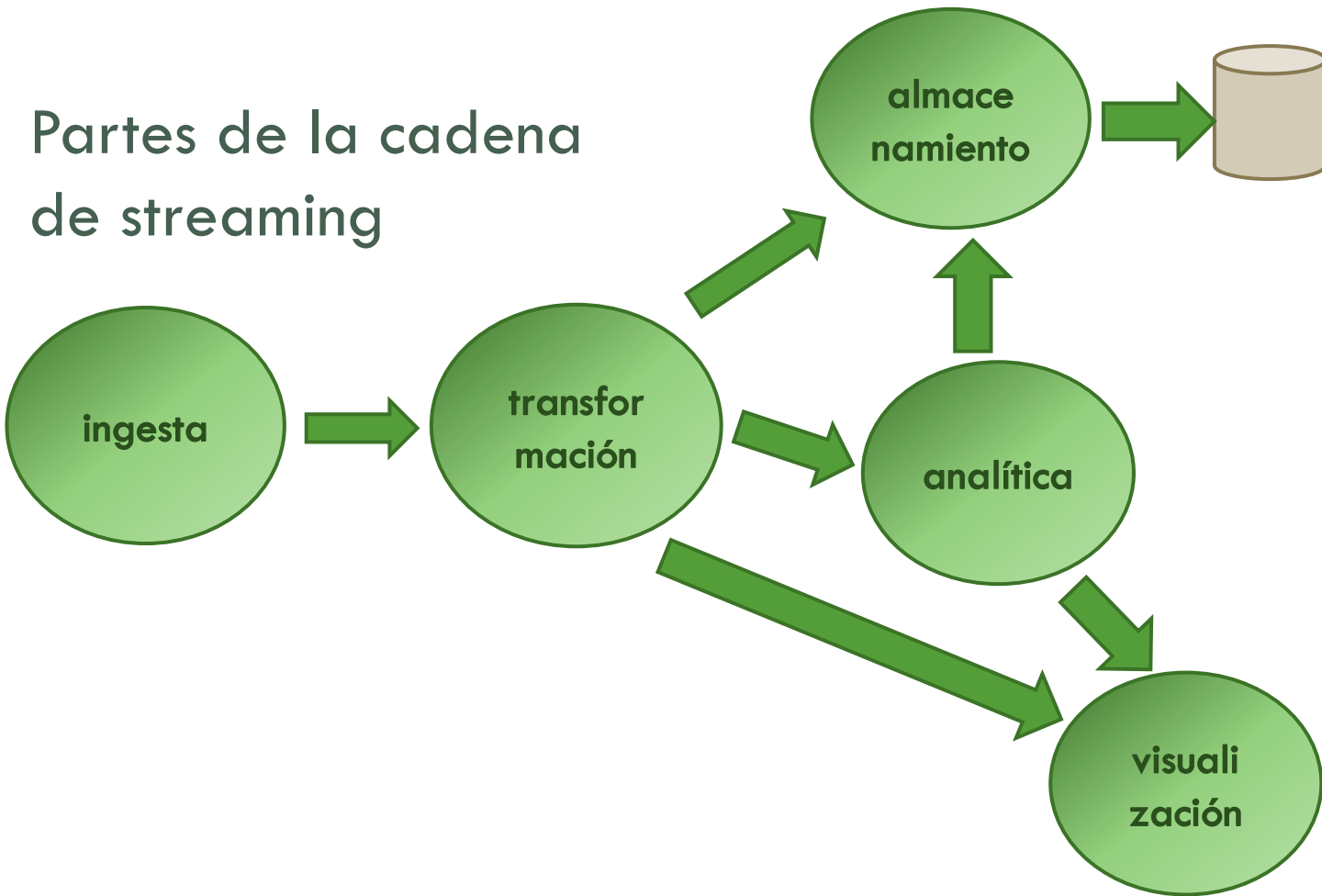
Modos de proceso datos: Batch vs. Streaming

- **Batch: procesado offline, por lotes**
 - datos almacenados en soporte persistente
 - acceso (típicamente) arbitrario
 - sin restricciones estrictas de proceso
- **Streaming: procesado online, en flujos**
 - datos provenientes de una fuente “viva”
 - acceso en secuencia
 - requiere procesado en un intervalo máximo de tiempo

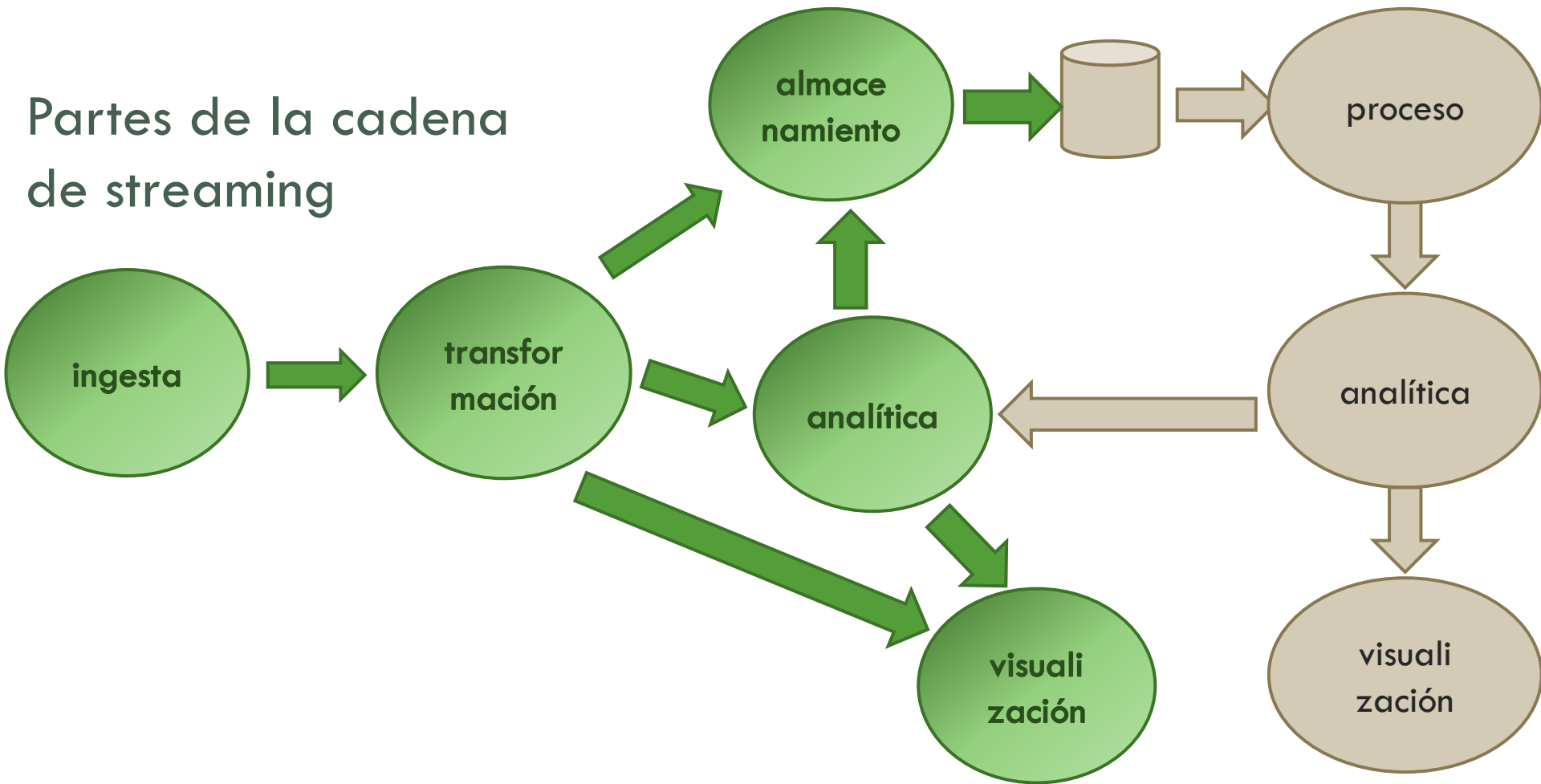
Modelos de streaming



Partes de la cadena de streaming




Partes de la cadena de streaming



Justificación

Procesar en
streaming
añade
complejidad



debe hacerse
cuando el
problema lo
justifica

Justificación

Procesar en
streaming
añade
complejidad



debe hacerse
cuando el
problema lo
justifica



Se exige tiempo de
reacción muy corto

La estructura de los
datos se presta a ello

Justificación

Procesar en
streaming
añade
complejidad



debe hacerse
cuando el
problema lo
justifica



Se exige tiempo de
reacción muy corto

detección de
fraude

La estructura de los
datos se presta a ello

redes
sociales

Justificación

Procesar en
streaming
añade
complejidad



debe hacerse
cuando el
problema lo
justifica



Se exige tiempo de
reacción muy corto

detección de
fraude

La estructura de los
datos se presta a ello

redes
sociales

Big Data streaming
es todavía más complejo



El volumen de datos
lo exige

Tecnologías software para streaming

Fuentes

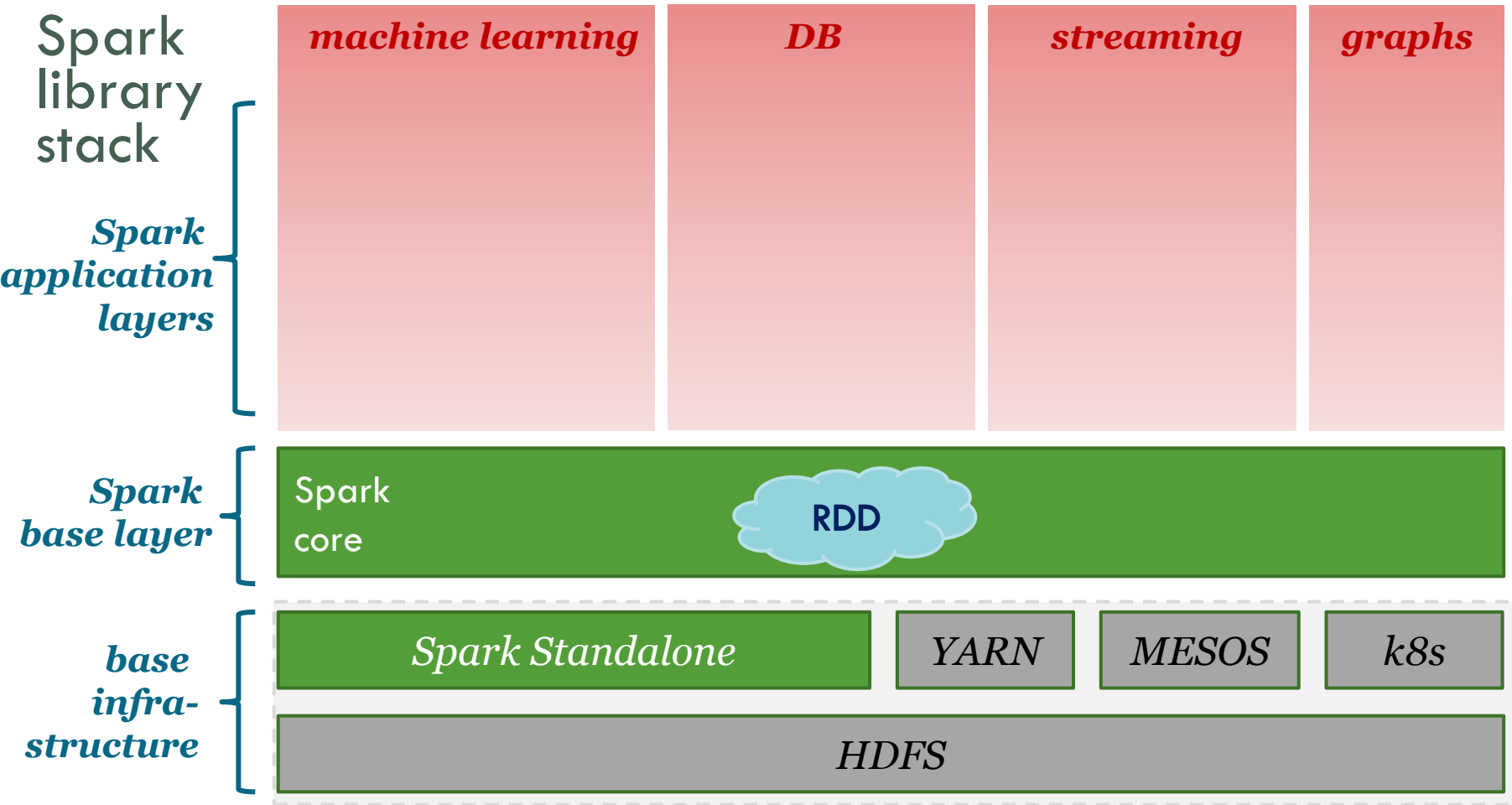
- Kafka
- Kinesis
- Flume

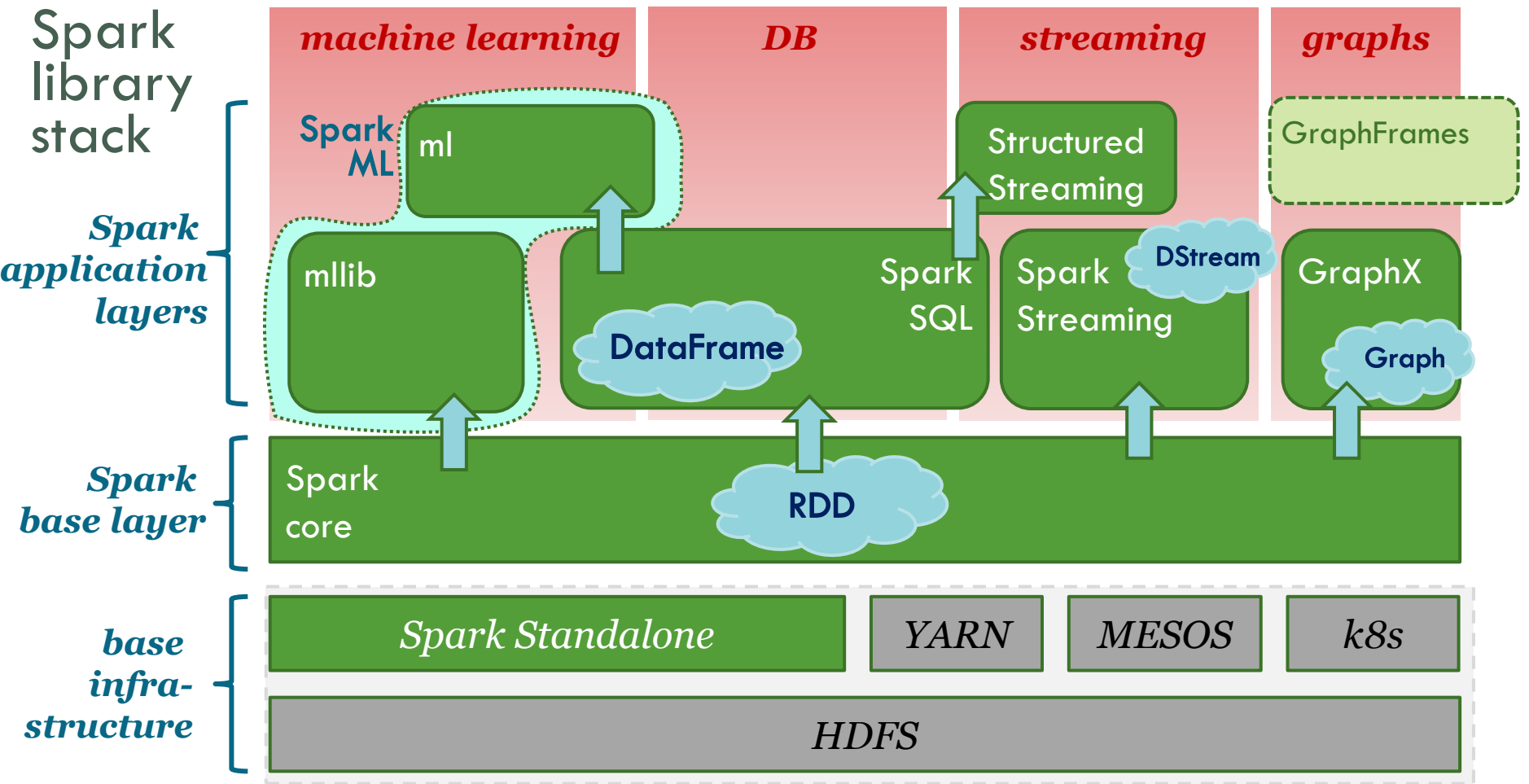
Procesado

- Spark Streaming
- Flink
- Kafka streams
- Storm
- NiFi
- Apex
- Samza
- Ignite

Spark streaming

- Arquitectura de micro-batches
- Dos interfaces disponibles:
 - Spark Streaming → Dstream → RDD
 - Structured Streaming → DataFrame





Streaming clásico (RDDs)

Principios de Spark Streaming

Matei Zaharia et al, [*Discretized Streams: Fault-Tolerant Streaming Computation at Scale*](#), Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

Discretized Streams: Fault-Tolerant Streaming Computation at Scale

Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

Many “big data” applications must act on data in real time. Running these applications at ever-larger scales requires parallel platforms that automatically handle faults and stragglers. Unfortunately, current distributed stream processing models provide fault recovery in an expensive manner, requiring hot replication or long recovery times, and do not handle stragglers. We propose a new

faults and stragglers (slow nodes). Both problems are inevitable in large clusters [12], so streaming applications must recover from them quickly. Fast recovery is even *more* important in streaming than it was in batch jobs: while a 30 second delay to recover from a fault or straggler is a nuisance in a batch setting, it can mean losing the chance to make a key decision in a streaming setting.

Unfortunately, existing streaming systems have

DStream

- Un **DStream** es el tipo básico de una aplicación Spark Streaming
- Se define como un conjunto sucesivo de mini-RDDs
 - Arquitectura de micro-batches
 - En cada instante de tiempo dado tenemos un RDD concreto
 - En el siguiente instante de tiempo, el RDD cambia para albergar los nuevos datos

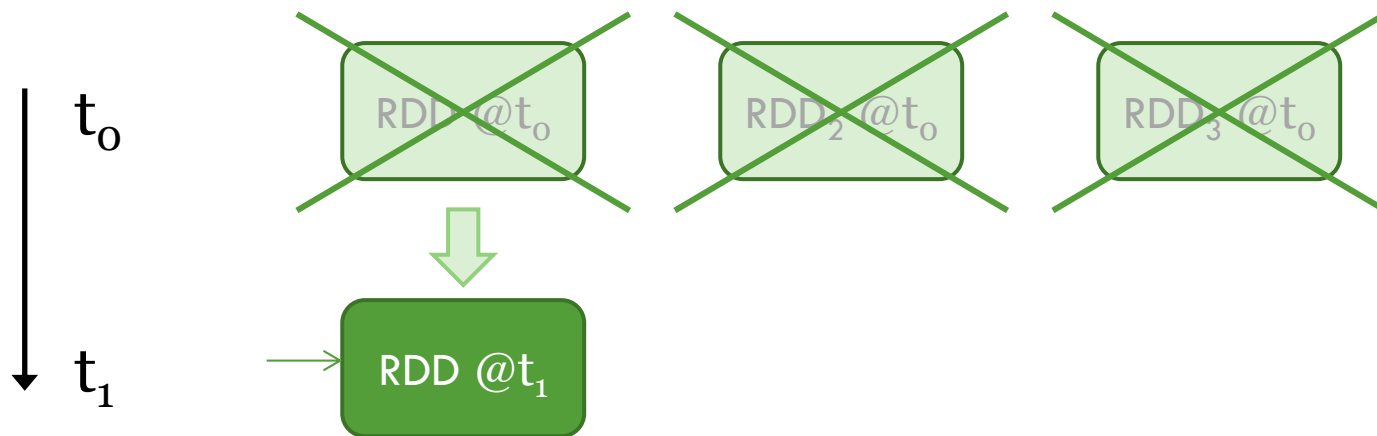
Flujo de proceso



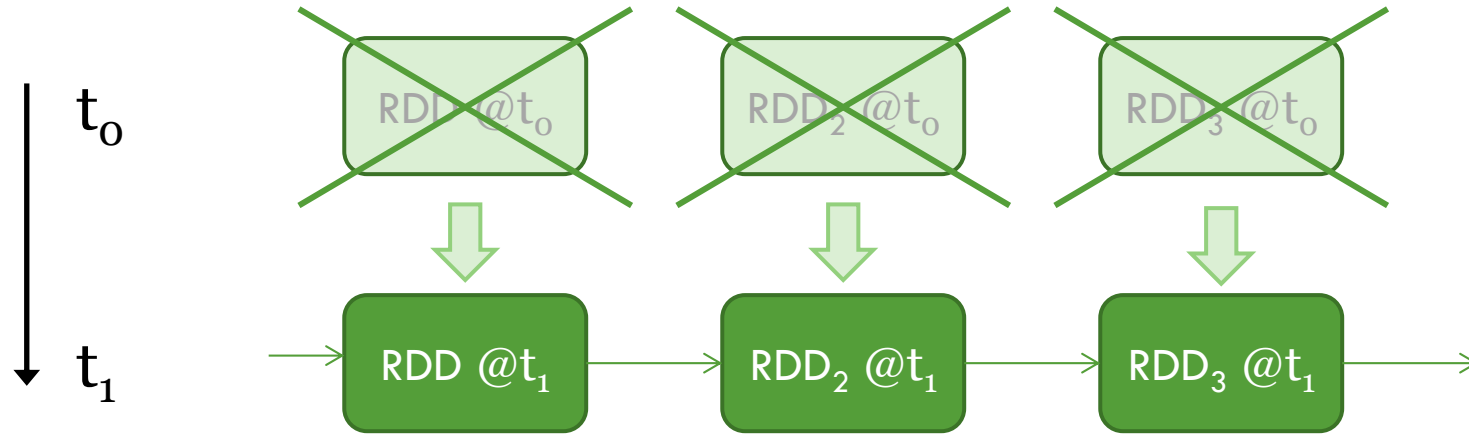
Flujo de proceso



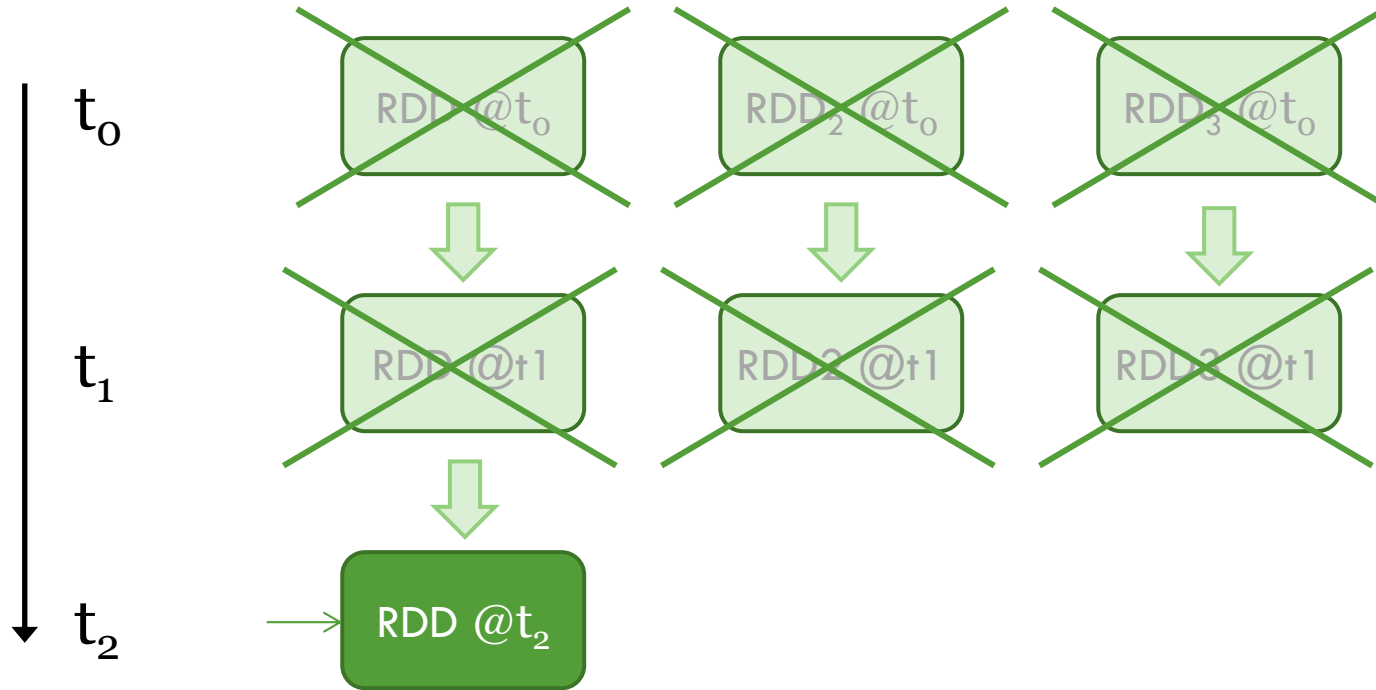
Flujo de proceso



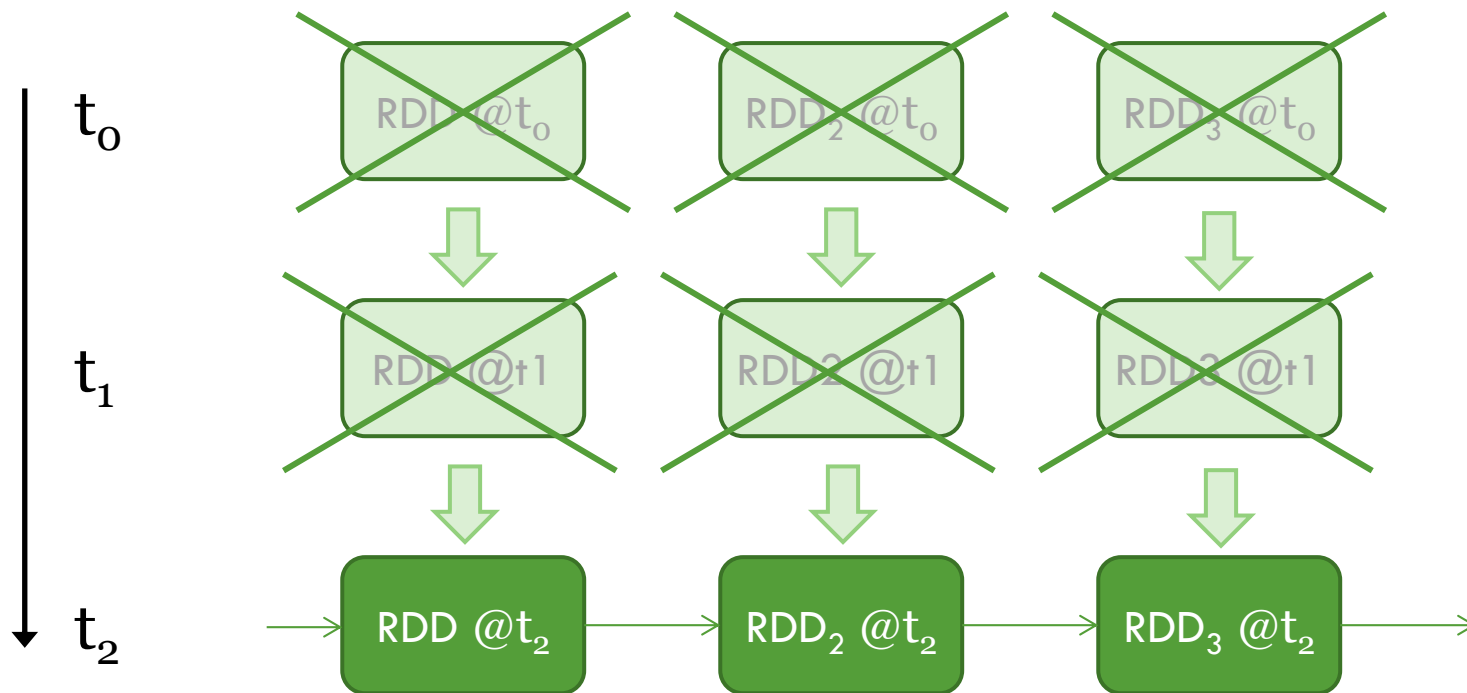
Flujo de proceso



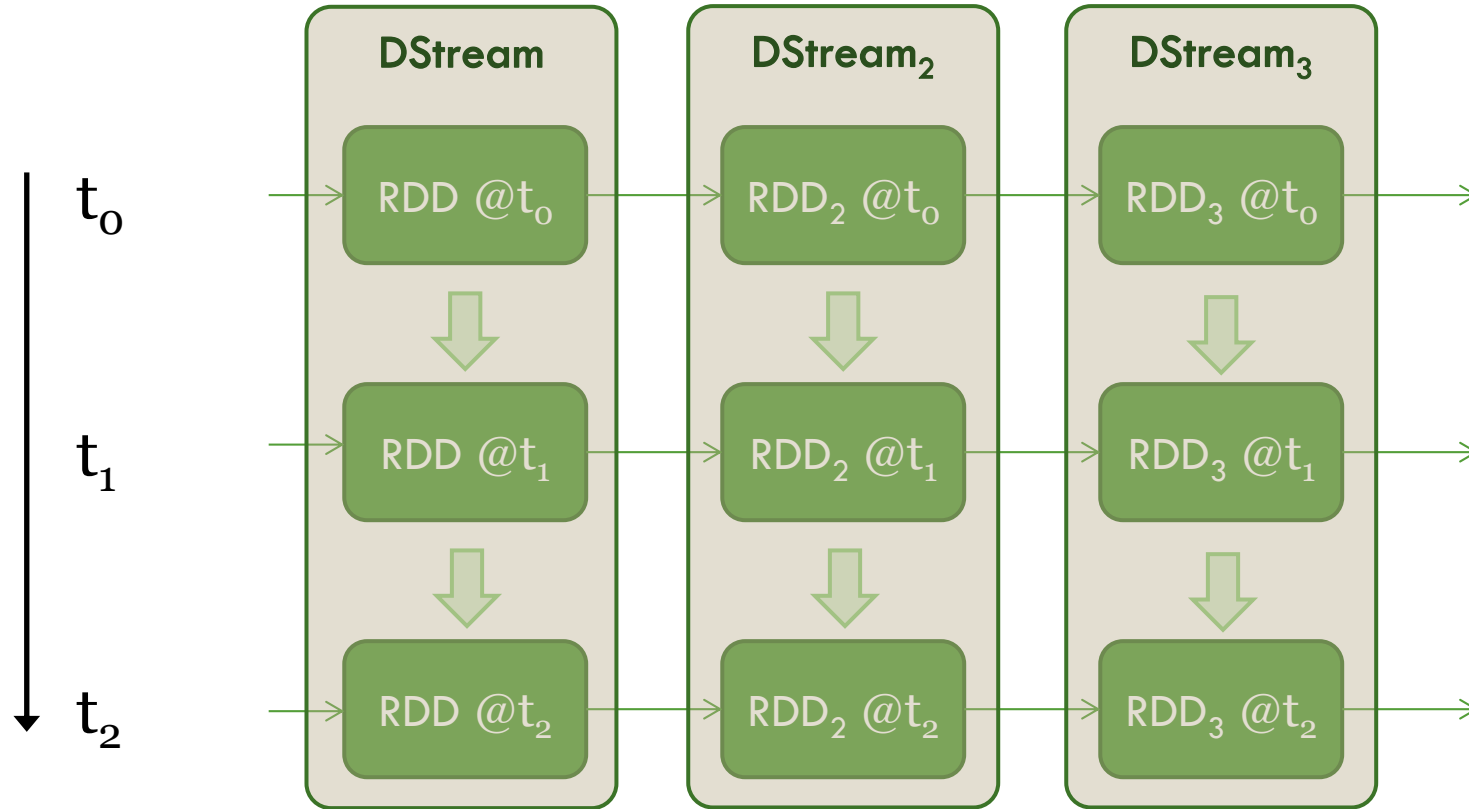
Flujo de proceso



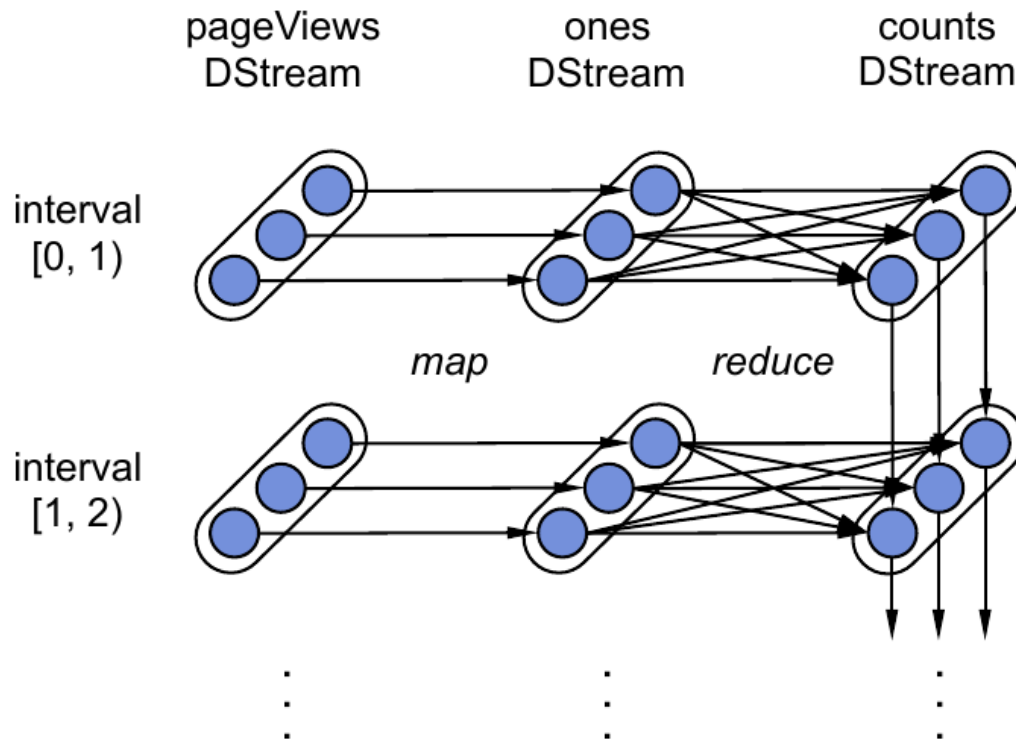
Flujo de proceso



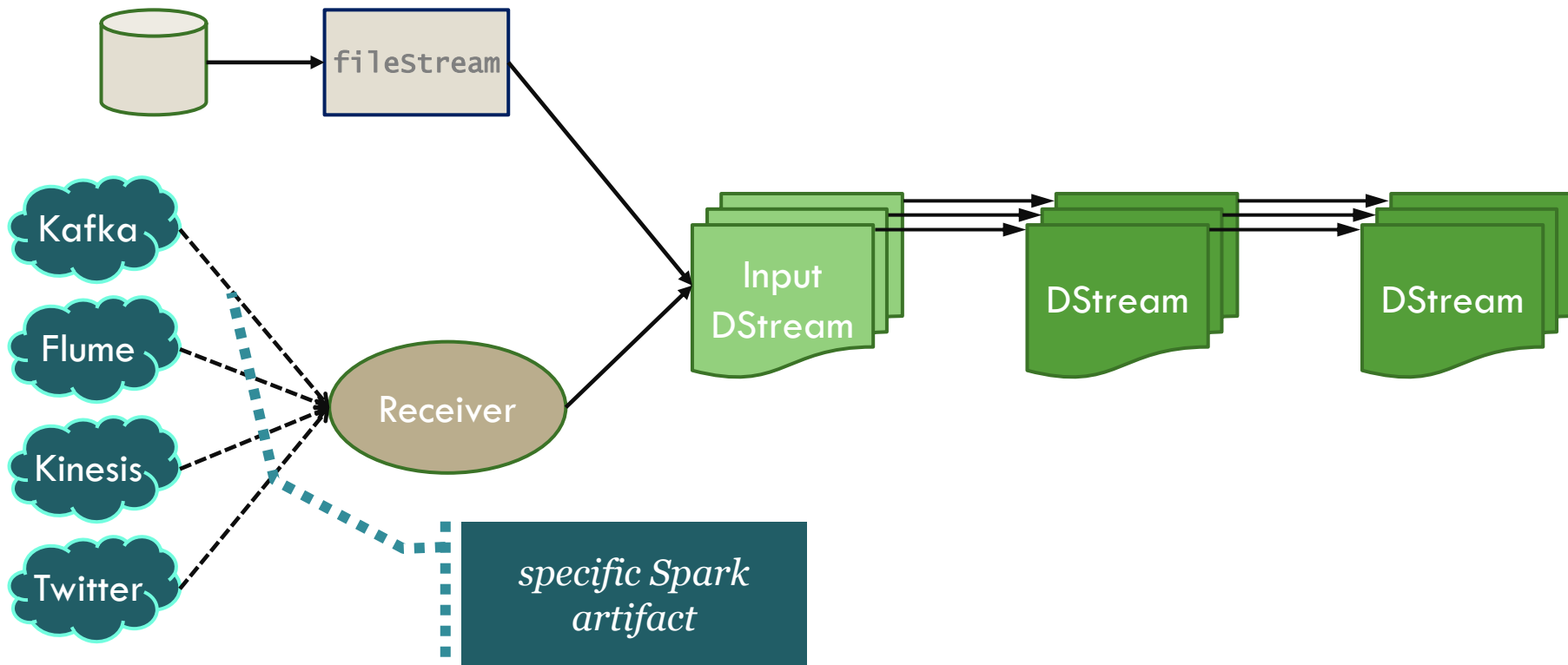
Flujo de proceso



DStreams están particionadas



Flujo de datos



Mecánica

Ejemplo básico

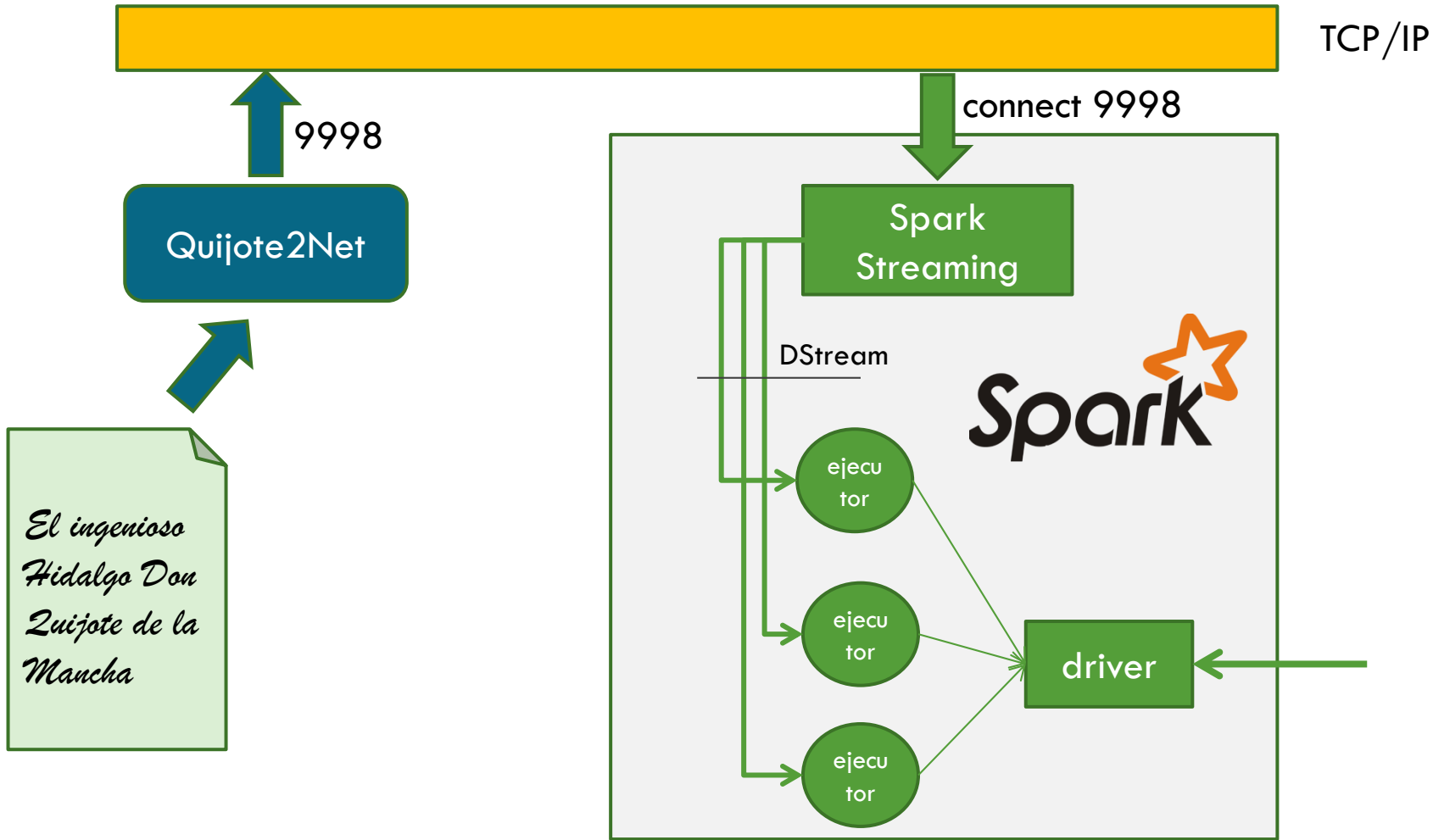
➤ Necesitamos **dos** consolas locales en la máquina virtual

1. Enviamos datos a un puerto local de la máquina

```
sh PublishData --port 9998
```

2. Arrancamos un proceso de Spark Streaming que consuma esos datos

```
spark-submit streaming0_simple.py
```



El contexto para streaming

- Las operaciones con RDDs tenían como contexto un **SparkContext**, las operaciones con DataFrames usan un **SQLContext/SparkSession** ... las operaciones de streaming usan un **StreamingContext**
- Un **StreamingContext** se crea a partir de un **SparkContext**

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
```

```
sc = SparkContext(master, appName)
ssc = StreamingContext(sc, 1)
```

... (definimos las operaciones en el ssc)

```
ssc.start()
ssc.awaitTermination()
```

El contexto para streaming

- Las operaciones con RDDs tenían como contexto un **SparkContext**, las operaciones con DataFrames usan un **SQLContext/SparkSession** ... las operaciones de streaming usan un **StreamingContext**
- Un **StreamingContext** se crea a partir de un **SparkContext**

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
```

lo creamos a partir del SparkContext

```
sc = SparkContext(master, appName)
ssc = StreamingContext(sc, 1)
```

le decimos el período de captura

... *(definimos la fuente desde el ssc)*

... *(definimos las operaciones sobre RDDs)*

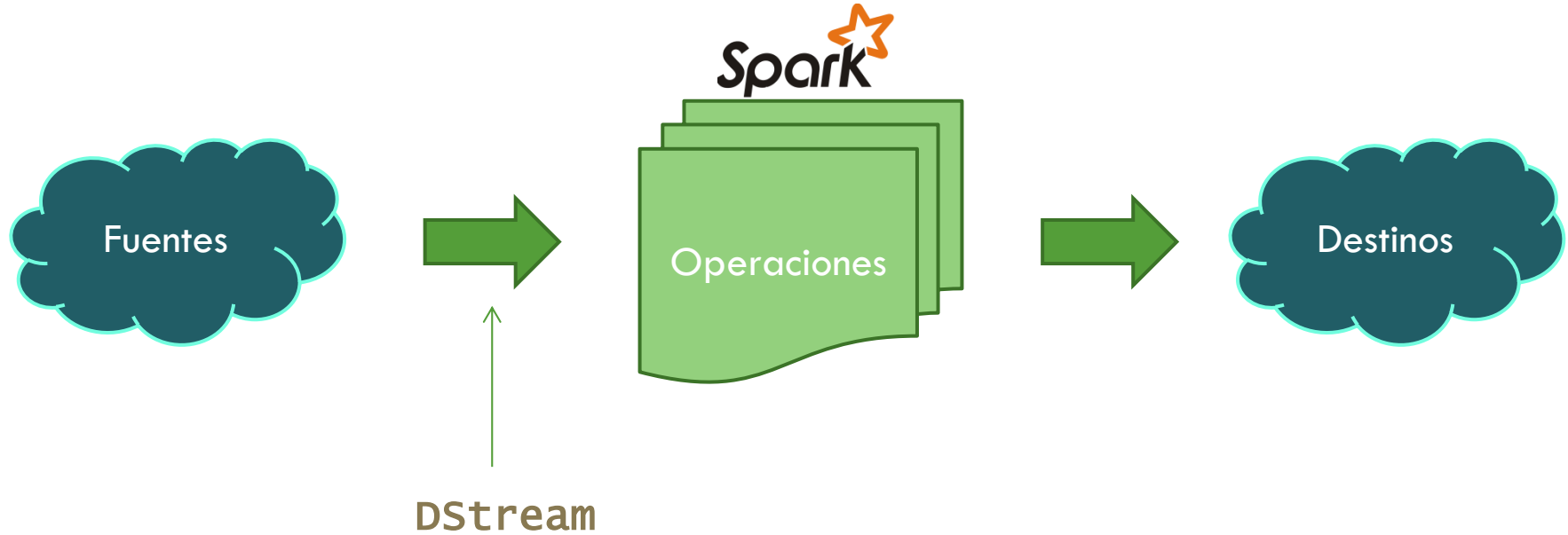
```
ssc.start()
ssc.awaitTermination()
```

un StreamingContext hay que arrancarlo

y pararlo o esperar a que termine

Entrada/salida

Flujo



Fuentes de datos

➤ Básicas

- Ficheros (e.g. HDFS)
- Sockets (host, port)

Monitoriza directorios, e ingesta los ficheros nuevos

➤ Avanzadas

- Kafka
- Flume
- Kinesis

Requieren paquetes adicionales

Salida

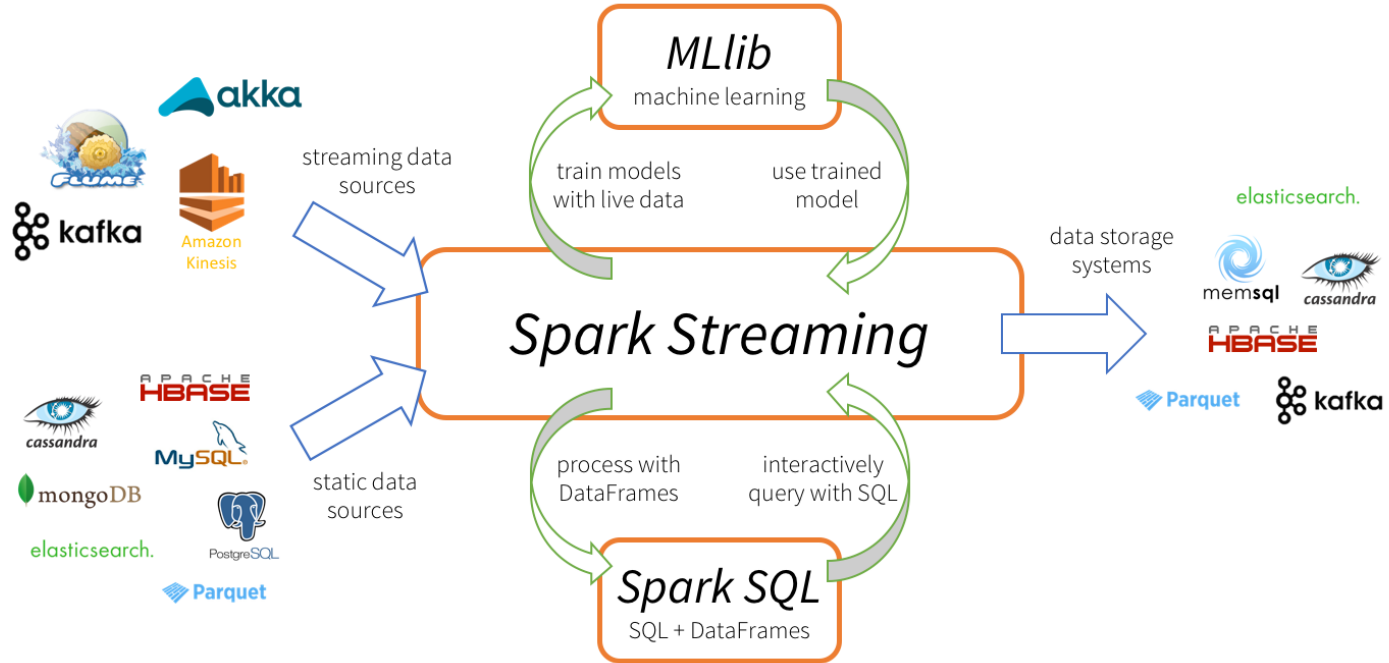
Método	Operación
<code>pprint(num=10)</code>	Escribe a consola los N primeros elementos del RDD <i>Se ejecuta en el driver</i>
<code>saveAsTextFiles</code>	Graba cada RDD como un fichero de texto independiente
<code>foreachRDD</code> (func)	Aplica una función al RDD generado en cada período de ejecución <i>La función se ejecuta en el driver</i>



El RDD recibido por la función está particionado en el cluster

Nota: los métodos `saveAsObjectFiles` & `saveAsHadoopFiles` no están disponibles en Python

Ecosistema de streaming



Spark Streaming en Notebooks

Notebooks con Spark Streaming

1. Asegurarse de que Spark tiene hebras de proceso suficientes

En la VM: comprobar la configuración de Spark en
`/opt/spark/current/conf/spark-default.conf`

y asegurarse de que arranca al menos 2 hebras:

```
spark.master=local[2]
```

2. Buscar un modo de ofrecer salida continua

Por ejemplo: usando widgets para Jupyter notebook ([ipywidgets](#))

- es necesario instalarlo (`pip install ipywidgets`) y activarlo
`jupyter nbextension enable --py widgetsnbextension --sys-prefix`
- en la VM de Spark ya está instalado y activado

3. Si es necesario, rearrancar el servicio *(la interfaz Web se interrumpirá)*

```
sudo systemctl restart notebook
```

Manipulación de streams

Operaciones

transformaciones	DStream → DStream
modificación de estado	updateStateByKey
	transform
ventanas	window
salida	saveAsTextFiles
	foreachRDD

Operaciones con DStreams

- Transformaciones
 - Similares a las transformaciones de RDDs (pero no todas)
- Operaciones de ventanas
 - Aplican a una ventana móvil de RDDs
- Operaciones de salida
 - Grabar a disco, escribir a consola, `foreachRDD()`
- Operaciones con DataFrames
- Aplicación de algoritmos de MLlib

Transformaciones

DStream → DStream	
map(func)	
flatMap(func)	
filter(func)	
union(otherStream)	
count()	
reduce(func)	
countByValue()	
reduceByKey(func, [numTasks])	
join(otherStream, [numTasks])	
cogroup(otherStream, [numTasks])	

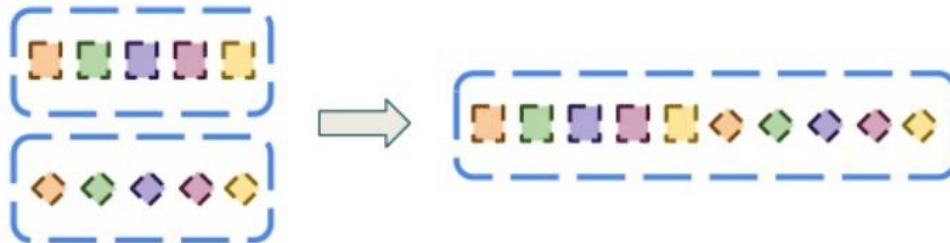
map,
flatmap,
filter



count,
reduce,
countByValue,
reduceByKey



union,
join
cogroup



Gerard Maas, Spark Summit Europe, 2017

Mantenimiento de estado

Manteniendo estado

- Por defecto, un RDD de un DStream va mutando a cada instante de tiempo. Los datos del instante anterior se pierden.
- Es posible también mantener estado
 - `updateStateByKey`
 - `mapWithState` (no disponible en Python)

Manteniendo estado: `updateStateByKey`

- **`updateStateByKey`** permite aplicar una función de actualización de estado a cada RDD de un DStream
- Se llama en los ejecutores de Spark, no en el driver (análogo a las transformaciones)
- Cada ejecutor guarda el valor retornado en cada llamada de la función de actualización y se lo pasa a la función en la siguiente llamada.
 - Esto permite llevar el estado de un micro-lote al siguiente
- Exige checkpointing

Robustez

Robustez

- Spark Streaming no posee de forma nativa la robustez estándar de Spark
- Los datos son “vivos”, luego no es posible regenerar el DAG
- Sin embargo tiene facilidades para resistencia frente a errores

Checkpointing

- grabación del estado a disco, para regeneración en caso de errores
- obligatorio en caso de guardar estado
- útil en el driver para recuperarse después de caídas

Otras medidas

- Write-Ahead-Logs

`spark.streaming.receiver.writeAheadLog.enable=true`

- Rearranque automático del driver

- (depende de configuración del cluster)

- Control de la tasa de lectura mediante *backpressure*

`spark.streaming.backpressure.enabled=true`

Procesado por ventanas

Funciones de ventana

DStream → windowed DStream

`window(windowLength, slideInterval)`

`countByWindow(windowLength, slideInterval)`

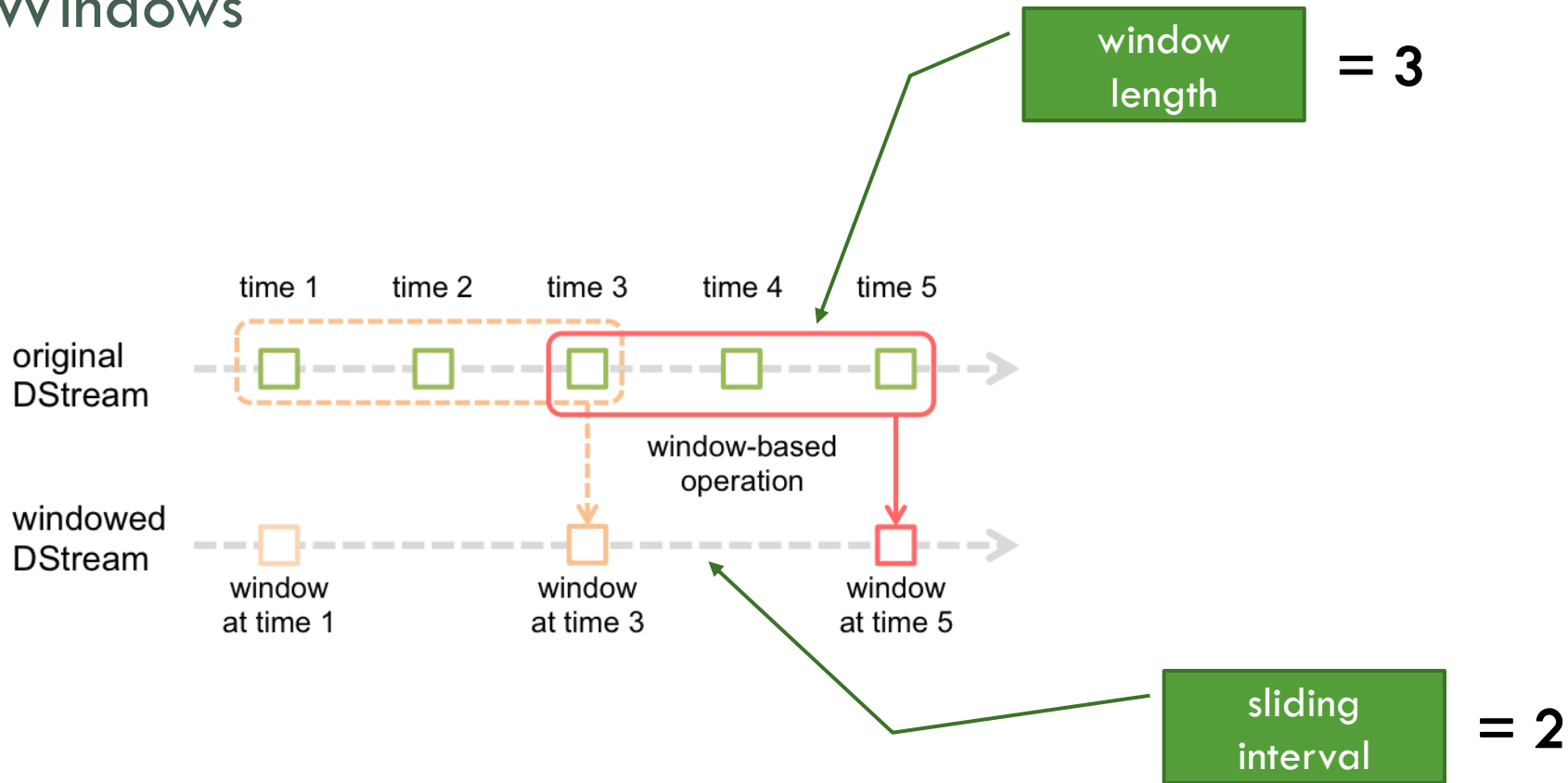
`reduceByWindow(func, windowLength, slideInterval)`

`reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])`

`reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])`

`countByValueAndWindow(windowLength, slideInterval, [numTasks])`

Windows



Criterios de diseño

Criterios de diseño

1. Ajustar el paralelismo de lectura mediante el dimensionamiento de input DStreams (por particionado o por multiplexación)
2. Revisar el número de cores disponible en cada ejecutor. Un receptor (input DStream) ocupa un core.
3. Ajustar la carga computacional mediante
 - Particionado de los DStreams
 - Intervalo de captura
4. Revisar la resistencia frente a errores (checkpointing, WAL, rearrancado del driver, backpressure)

Streaming estructurado

Streaming Estructurado

- Funcionalidad disponible en Spark 2.x
 - Disponible de forma experimental empezando en 2.1
 - Marcado como estable a partir de 2.2 (julio 2017)
- Disponible en Scala, Java, Python y R
- La parte más activa de Spark Streaming

Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark

Michael Armbrust[†], Tathagata Das[†], Joseph Torres[†], Burak Yavuz[†], Shixiong Zhu[†],
Reynold Xin[†], Ali Ghodsi[†], Ion Stoica[†], Matei Zaharia^{†‡}
[†]Databricks Inc., [‡]Stanford University

Abstract

With the ubiquity of real-time data, organizations need streaming systems that are scalable, easy to use, and easy to integrate into business applications. Structured Streaming is a new high-level streaming API in Apache Spark based on our experience with Spark Streaming. Structured Streaming differs from other recent streaming APIs, such as Google Dataflow, in two main ways. First, it is a purely *declarative* API based on automatically incrementalizing a static relational query (expressed using SQL or DataFrames), in contrast to APIs that ask the user to build a DAG of physical operators. Second, Structured Streaming aims to support *end-to-end* real-time

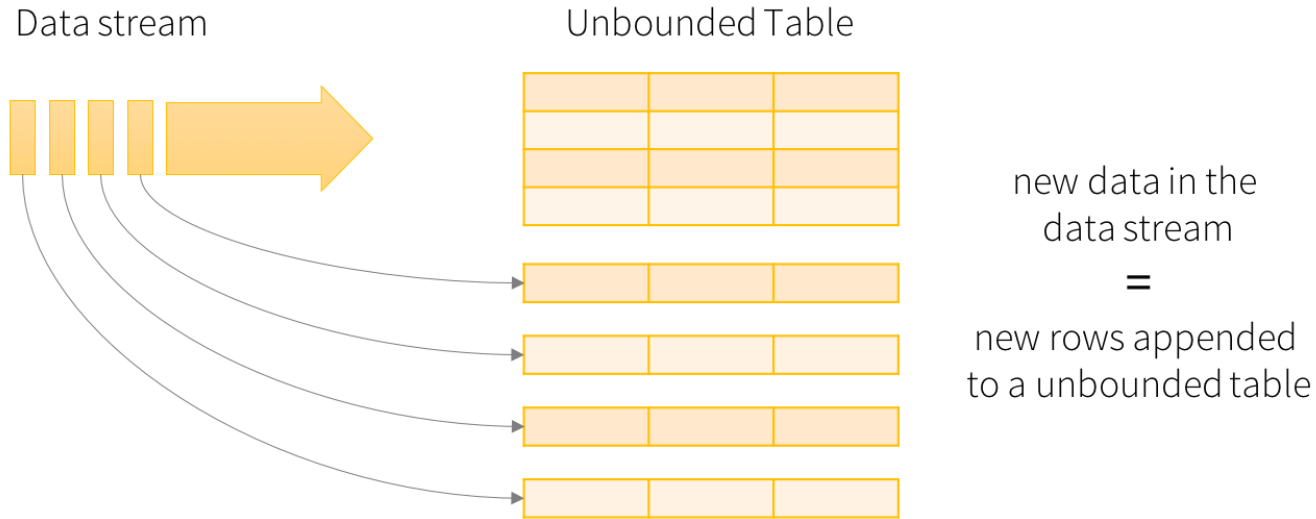
with Spark Streaming [37], one of the earliest stream processing systems to provide a high-level, functional API. We found that two challenges frequently came up with users. First, streaming systems often ask users to think in terms of complex physical execution concepts, such as at-least-once delivery, state storage and triggering modes, that are unique to streaming. Second, many systems focus *only* on streaming computation, but in real use cases, streaming is often part of a larger business application that also includes batch analytics, joins with static data, and interactive queries. Integrating streaming systems with these other workloads (e.g., maintaining transactionality) requires significant engineering effort.

Motivated by these challenges, we describe Structured Stream-

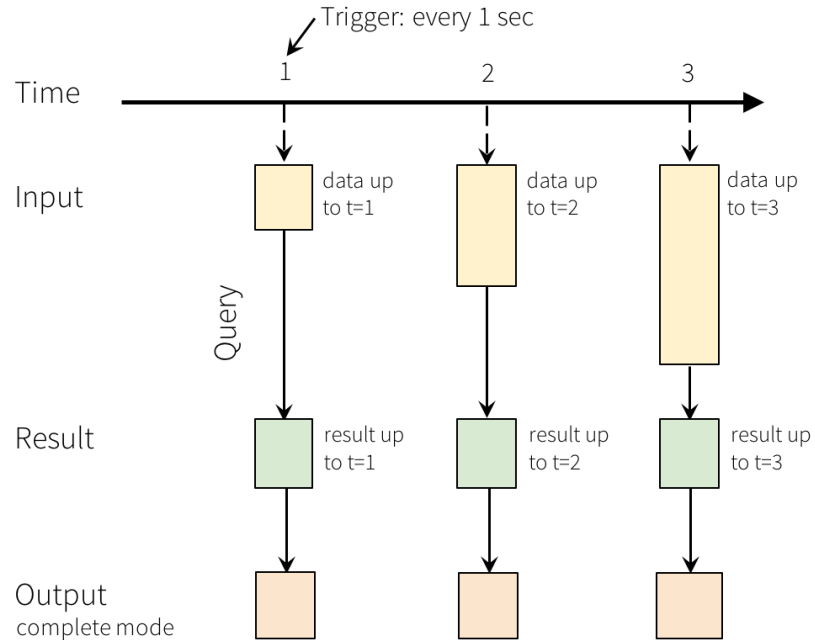
Principios básicos

¿Qué es streaming estructurado?

- Un API para trabajar con datos en tiempo real (streaming) usando directamente DataFrames
- Usa el motor de Spark SQL para ejecutarlo de forma incremental y continua, según llegan los datos
- Permite definir agregaciones, ventanas, y combinar streaming con batch.



Data stream as an unbounded table



Programming Model for Structured Streaming

Flujo de Streaming Estructurado

Punto de arranque
de la computación
en streaming
StreamingQuery



[readStream\(\)](#)

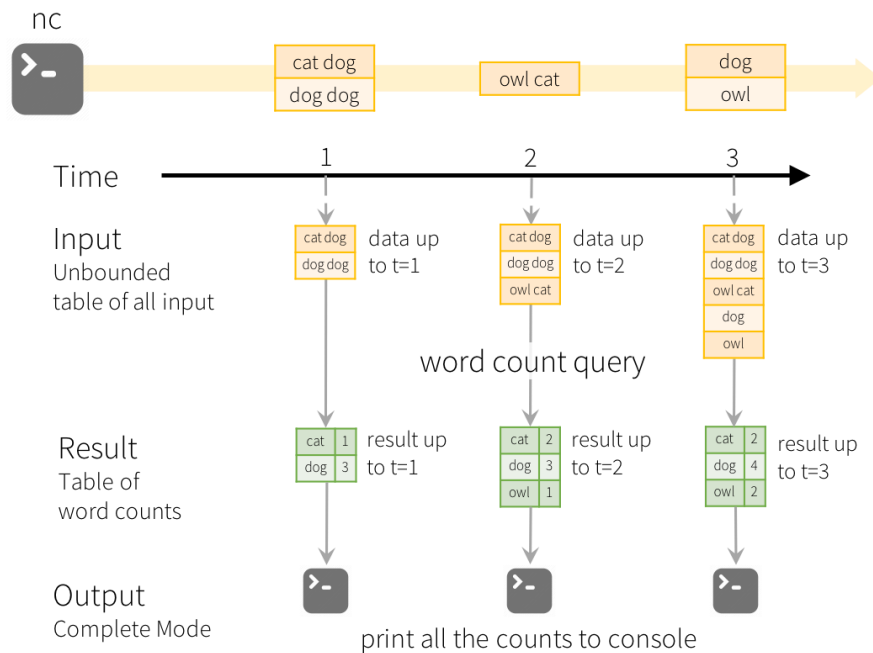


DataFrame methods

selection, projection, aggregation
window methods



[writeStream\(\)](#)



Model of the Quick Example

Entrada/Salida

Entrada: readStream ° ° °

Produce un
DataFrame

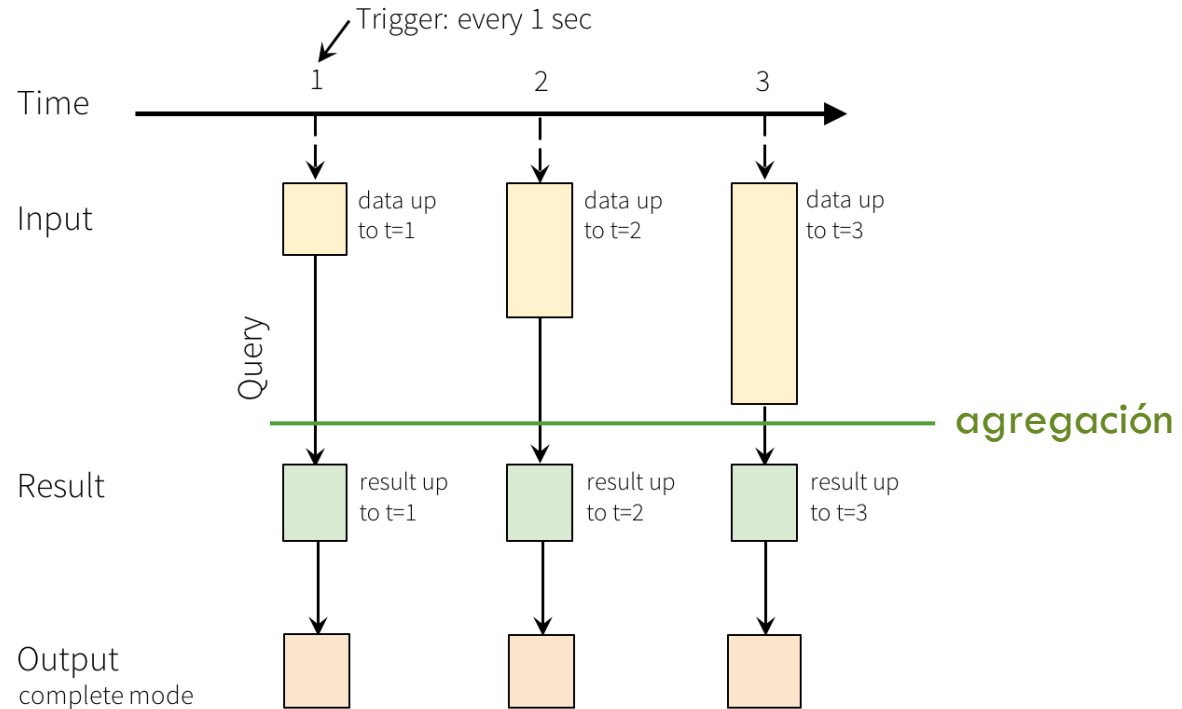
Tipo	Fuente
File source	Lee los ficheros existentes en un directorio
Socket source	Lee de un socket (<code>host:port</code>)
Rate source	Genera datos sintéticos (para pruebas)
Kafka source	Lee de un cluster de Kafka <i>(necesita el módulo de Kafka instalado)</i>

Modos de salida

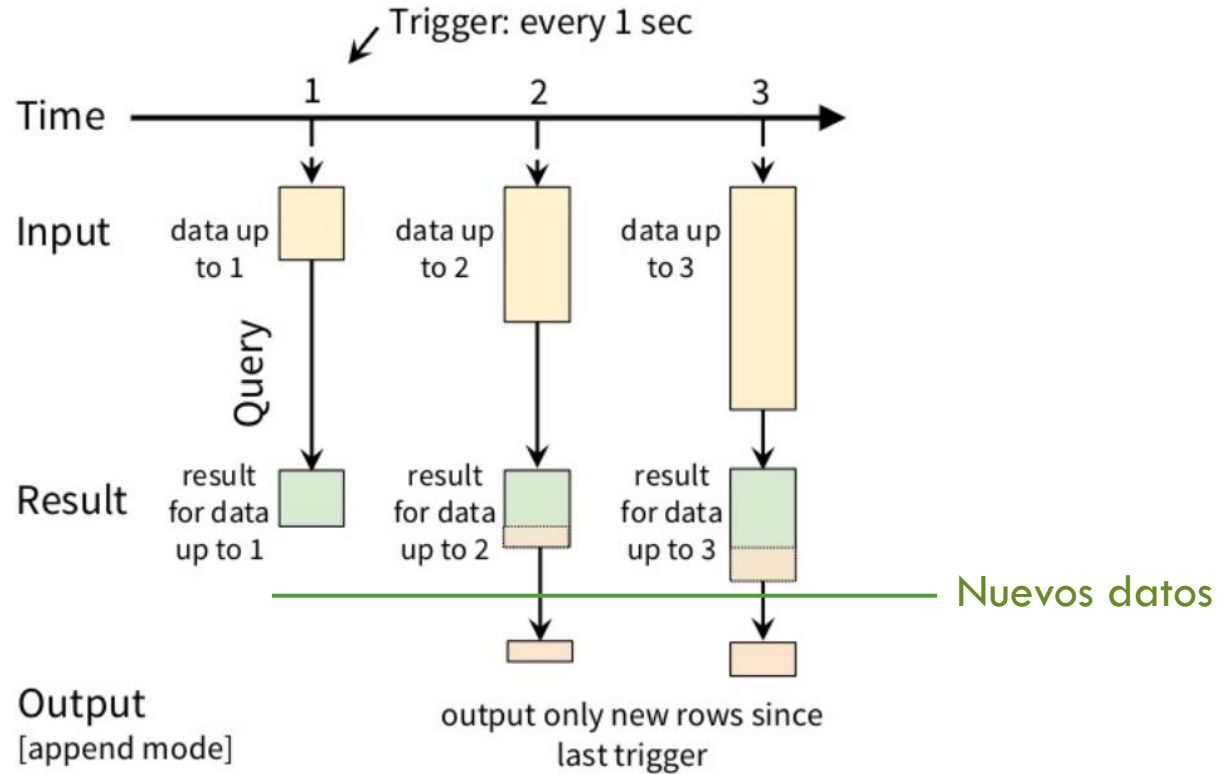
Modo	Funcionamiento	aplicaciones
complete	Escribe la tabla completa	agregaciones
append	Escribe las filas nuevas	select, where, map, flatMap, filter, join, etc
update	Escribe las filas modificadas	agregaciones

cada modo permite un subconjunto específico de operaciones

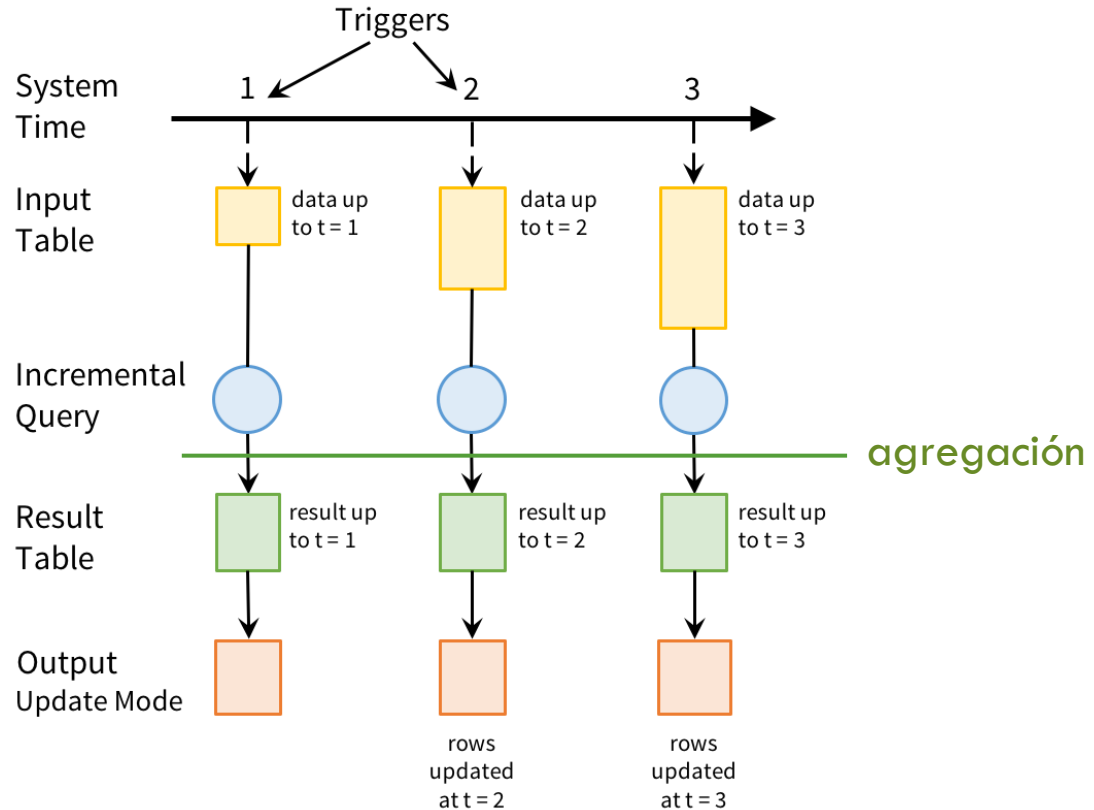
complete



append



update



Tipos de salida: `writeStream`

tipo	modos	sintaxis
File sink	append	<pre>writeStream .format("parquet") .option("path", "<path/to/destination/dir>") .start()</pre>
foreach sink	append, update, complete	<pre>writeStream .foreach(...) .start()</pre> <div>Scala & Java (Spark ≥ 2.1) Python (Spark ≥ 2.4)</div>
foreachBatch sink	append, update, complete	<pre>writeStream .foreachBatch(...) .start()</pre> <div>Scala & Java (Spark ≥ 2.1) Python (Spark ≥ 2.4)</div>
Console sink	append, update, complete	<pre>writeStream .format("console") .start()</pre>
Memory sink	append, complete	<pre>writeStream .format("memory") .queryName("tableName") .start()</pre>

foreach vs. foreachBatch

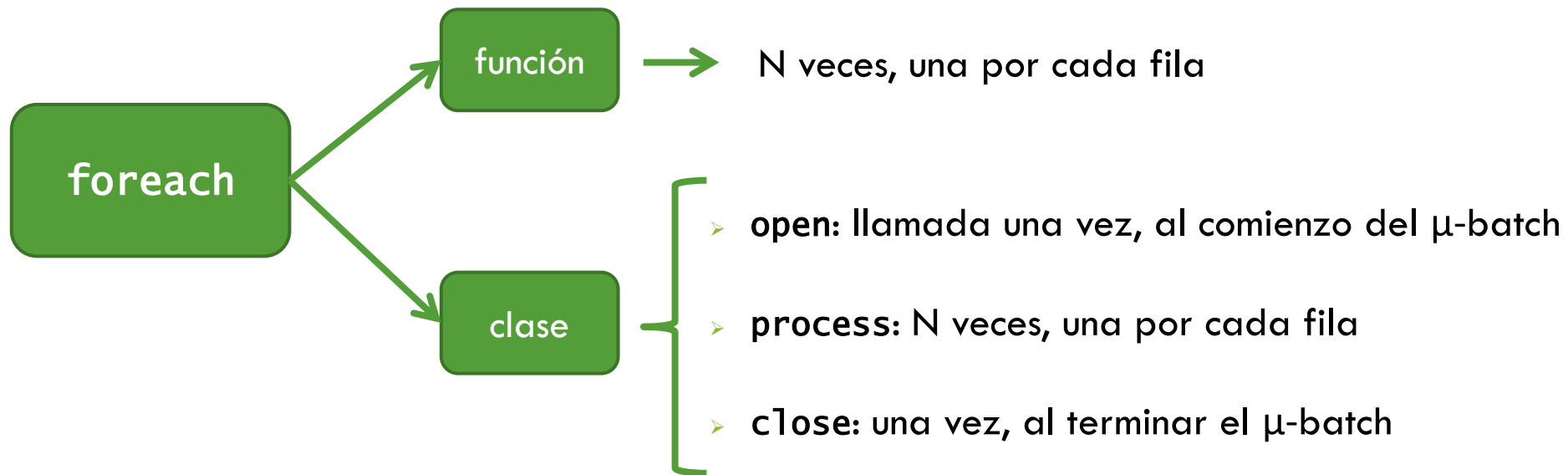
foreach

- Llamada en cada **fila** del DataFrame de salida
- N veces por cada micro-batch (tantas como filas)
- Especificada desde el punto de vista del **ejecutor**
- Recibe la fila (tipo Row)

foreachBatch

- Llamada en cada **micro-batch**
- 1 vez por cada micro-batch
- Especificada desde el punto de vista del **driver**
- Recibe un DataFrame

foreach variantes



Importante: hay que recordar que Spark es un sistema de procesamiento distribuido

El DataFrame producido por readStream en general estará **particionado**

Por tanto las filas están repartidas entre las particiones y **no hay ninguna garantía de contigüidad**

Opciones adicionales de salida

➤ Trigger interval

- Intervalo de ejecución de cada *query*
- Implícitamente define el tamaño del *batch*
`.trigger(processingTime='10 seconds')`

➤ Checkpointing

- Grabación a disco para recuperación en caso de error
`.option("checkpointLocation", "path/to/dir")`

➤ Query management

- API para gestión de objetos de *query*
`query.xxxx()`


API sobre objetos StreamingQuery

Grupo	API
Características	<code>query.id()</code> <code>query.runId()</code> <code>query.name()</code> <code>query.explain()</code>
Gestión	<code>query.stop()</code> <code>query.awaitTermination()</code> <code>query.exception()</code>
Monitorización (Spark \geq 2.1)	<code>query.recentProgress()</code> <code>query.lastProgress()</code> <code>query.status()</code>
Gestión global	<code>spark.streams()</code> <code>spark.streams().get(id)</code> <code>spark.streams().awaitAnyTermination()</code>

Extra: Kafka



Apache Kafka

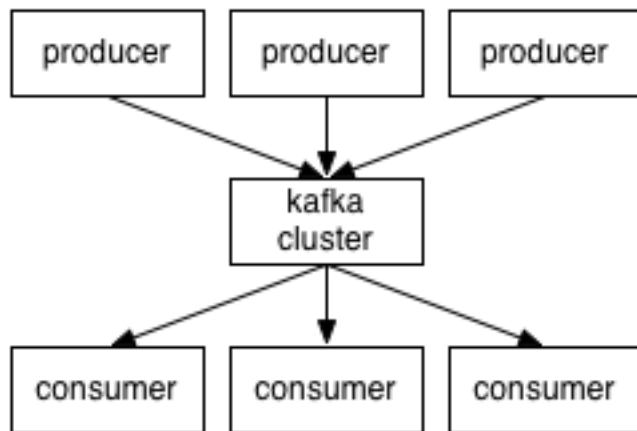
- Un sistema de captura, enrutado y envío de mensajes
- Originalmente creado dentro de LinkedIn
- Open Source
- Desarrollo impulsado por  CONFLUENT

*“a distributed, partitioned,
replicated commit log service”*

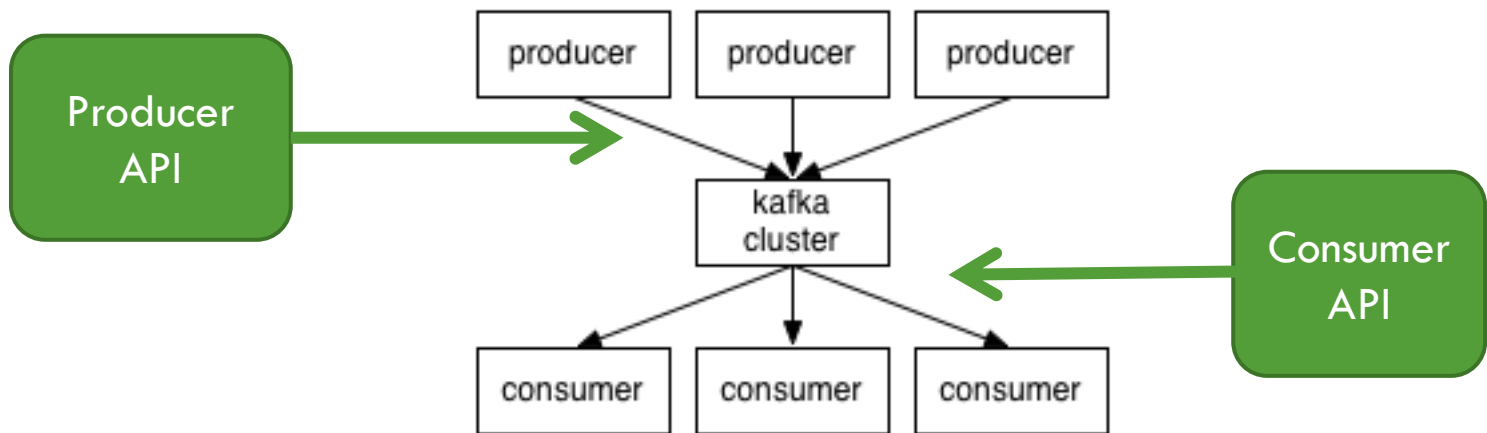
○ sea:

1. Kafka mantiene listas de mensajes en categorías llamadas **“topics”**
2. Los procesos que publican mensajes a un *topic* de Kafka se denominan **producers**
3. Los procesos que se suscriben a *topics* y recuperan mensajes de la lista y los procesan se denominan **consumers**
4. Kafka consiste en un cluster que comprende uno o varios servidores; cada uno de los cuales se denomina **broker**

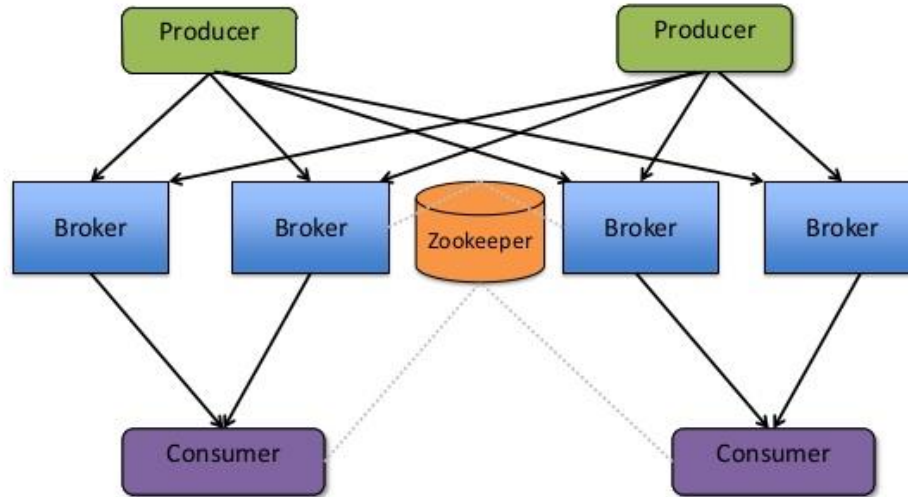
Diagrama de conexiones en Kafka



APIs proporcionadas

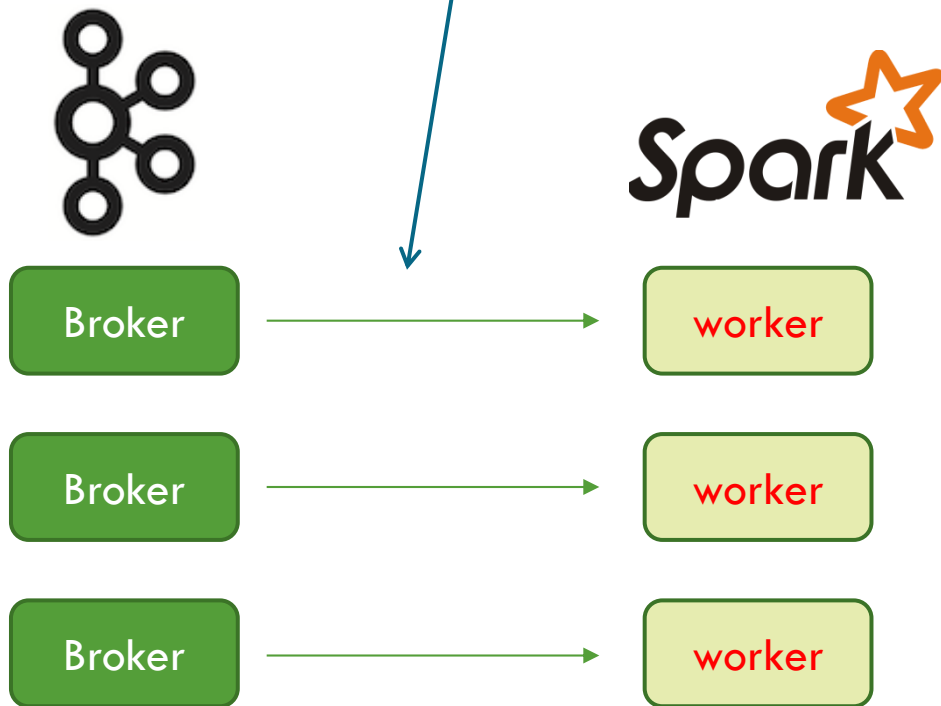


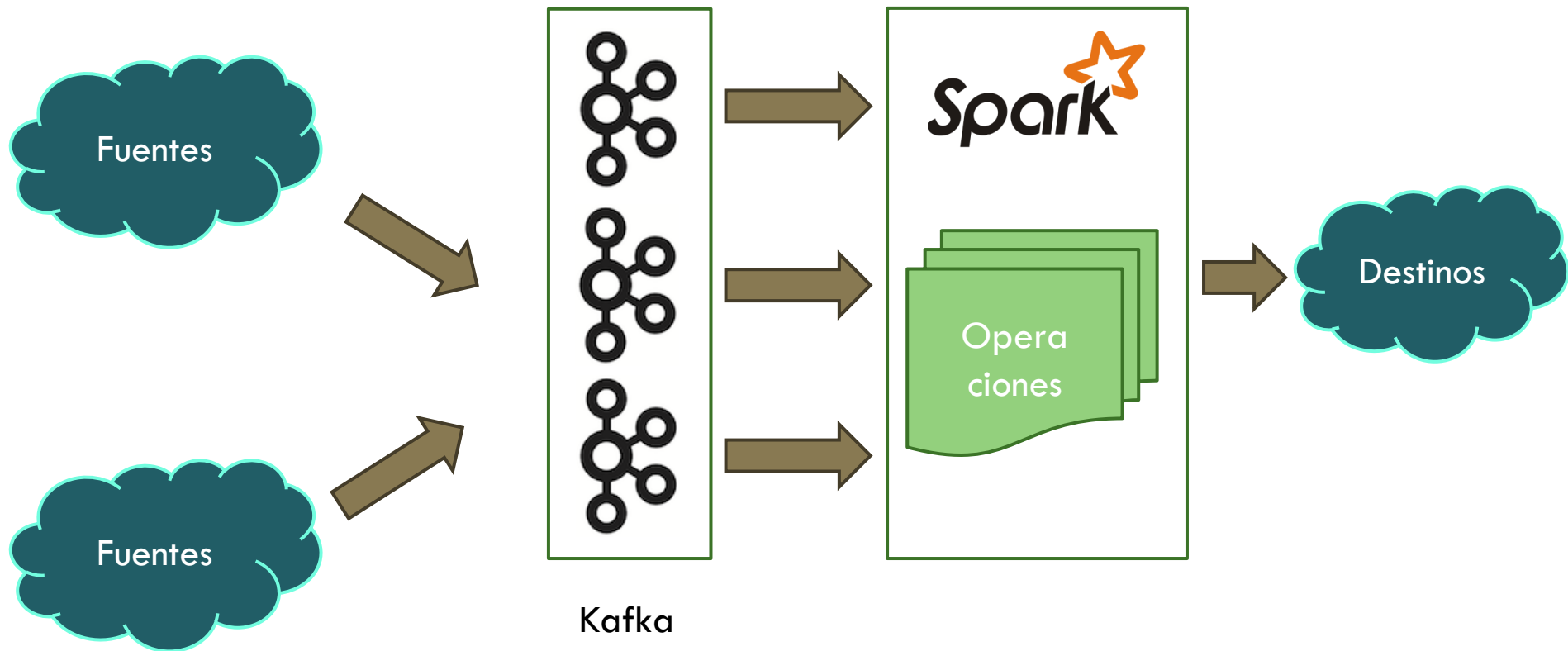
Kafka Architecture



Kafka & Spark

El paralelismo en Spark viene determinado por el particionado ofrecido por Kafka





Kafka & Spark

Kafka 0.8	RDD	<i>Receiver-based approach</i>	Scala, Java, Python	<i>deprecated in Spark 2.3 removed in Spark 3.0</i>
	RDD	<i>Direct approach</i>	Scala, Java, Python	
Kafka 0.10	RDD	<i>Direct stream</i>	Scala, Java	
	DataFrames	<u>Structured Streaming</u>	Scala, Java, Python, R	

Servidor Kafka de pruebas

Dirección	cluster1bigdata.ii.uam.es:9092
Topics	test
	books
	news
	tweet
	meteo

Uso de Kafka con Spark

Despliegue

- El uso del API de Kafka requiere de un módulo adicional, que no es parte estándar de la distribución de Spark
- Por tanto, para que funcione necesitamos cargar ese módulo para que esté disponible en el driver y en los ejecutores

Despliegue

Hay dos módulos disponibles:

- `spark-streaming-kafka-0-10_2.12:3.1.2`

Dstreams
(RDD)

- `spark-sql-kafka-0-10_2.12:3.1.2`

**Streaming
estructurado**
(DataFrames)

En Spark 3.x
no tiene API
de Python

Opciones de despliegue

	paquete	fichero
transitorio	como paquete lógico, en cada ejecución	como fichero(s) JAR, en cada ejecución
persistente	como paquete lógico, de forma permanente	como fichero(s) JAR, de forma permanente

Opciones de despliegue (paquete)

- 1. Indicar el paquete a cargar al lanzar la aplicación.

Spark lo descargará automáticamente de Internet (la primera vez) junto con sus dependencias

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 ...
```

- 2. Añadir la carga del paquete a la configuración de Spark en `spark-defaults.conf`

Spark lo descargará de Internet al arrancar

```
spark.jars.packages=org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2
```

Opciones de despliegue (ficheros)

- 3. Descargar manualmente el paquete (un `jar`) y añadirlo al lanzar la aplicación.

Spark lo mandará a todos los ejecutores

```
spark-submit --jars org.apache.spark_spark-sql-kafka-0-10_2.12-3.1.2.jar ...
```

- 4. Añadir la carga del fichero a la configuración de Spark en `spark-defaults.conf`

Spark lo leerá y mandará a los ejecutores

```
spark.jars=org.apache.spark_spark-sql-kafka-0-10_2.12-3.1.2.jar
```

Instalación en la máquina virtual

La configuración de la máquina virtual ya viene preparada para activar el paquete de Kafka, pero por defecto viene comentado

Para activarlo:

- Editar el fichero `/opt/spark/current/conf/spark-defaults.conf`
- Descomentar la línea (al final del fichero), cambiando:

```
# [2] structured streaming
```

```
#KFS#spark.jars.packages=org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2
```

a:

```
# [2] structured streaming
```

```
spark.jars.packages=org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2
```


Structured Streaming & Kafka

Kafka

- El despliegue de una aplicación Kafka con Structured Streaming es muy parecido al de Streaming clásico
- El paquete adicional necesario es **distinto**:
`org.apache.spark:spark-sql-kafka-0-10_2.13:3.1.2`
- El proceso de Spark se suscribe a un *topic* de Kafka (o a varios)
- Al igual que en Streaming clásico, en cada lote se pueden recibir 0, 1 o varios mensajes

Kafka

Una fuente Kafka aparece en Structured Streaming como un DataFrame

- Cada registro (mensaje) de Kafka es una fila
- Las columnas son los campos de un mensaje en Kafka

root

|-- key: binary (nullable = true)

|-- value: binary (nullable = true)

} mensaje

|-- topic: string (nullable = true)

|-- partition: integer (nullable = true)

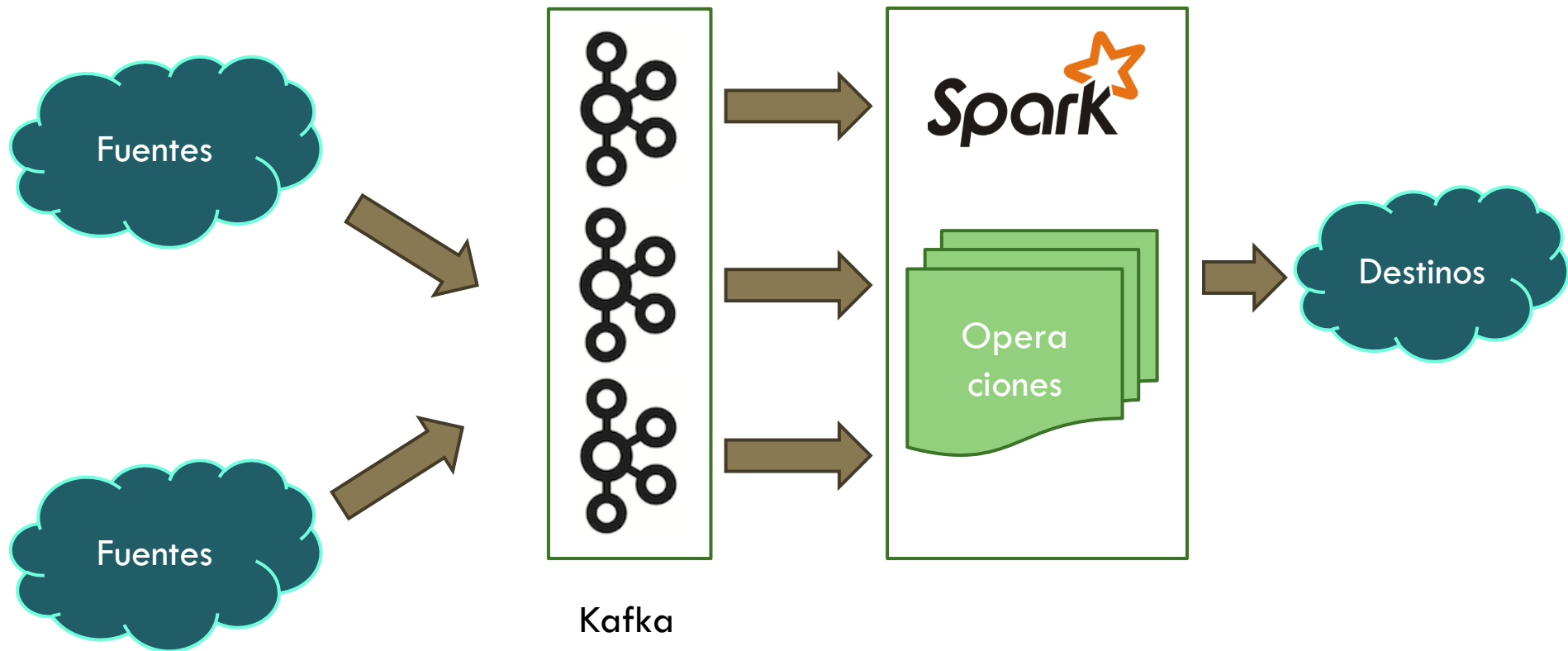
|-- offset: long (nullable = true)

} coordenadas del
mensaje en Kafka

|-- timestamp: timestamp (nullable = true)

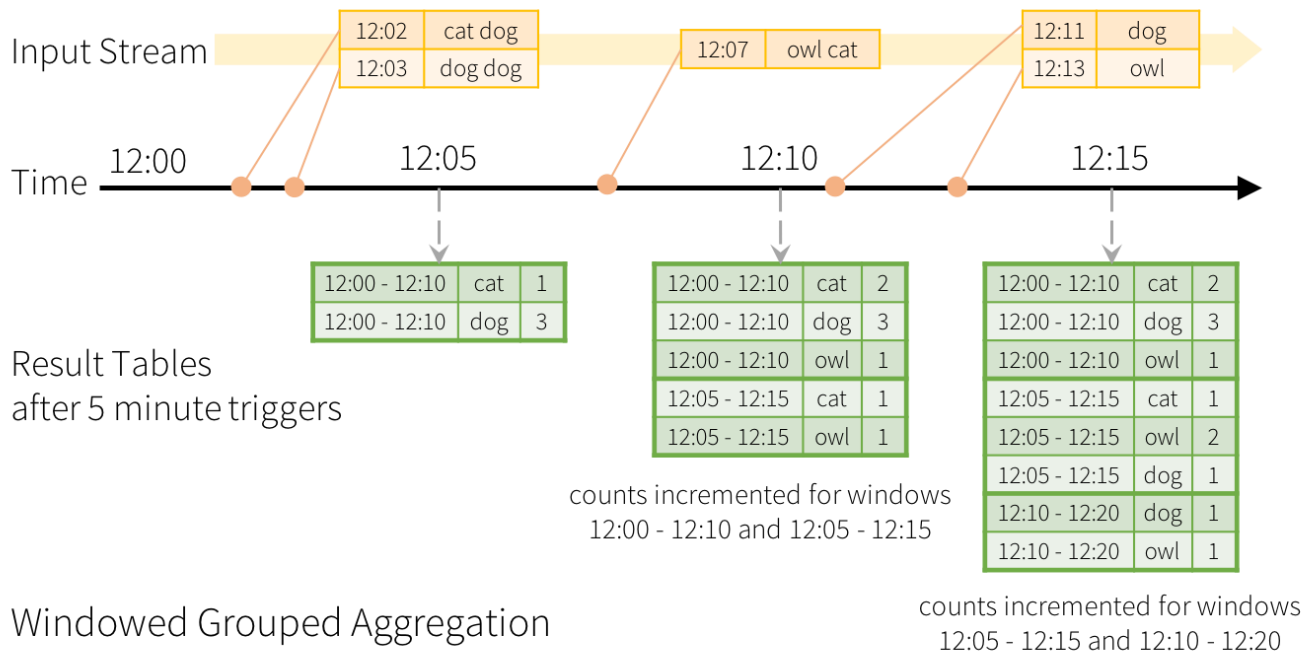
|-- timestampType: integer (nullable = true)

} marca de tiempo del
mensaje



Procesado de ventanas

Procesado de ventanas



Proceso de ventanas

- Se pueden usar ventanas temporales para agrupar datos

```
windowedCounts = words.groupBy(  
    window(words.timestamp, windowDuration, slideDuration),  
    words.word  
) .count().orderBy('window')
```

La definición de una ventana necesita:

1. Una **columna** con una marca de tiempo, sobre la que agrupar
2. La **duración** de la ventana (cómo son de anchas)
3. El **incremento temporal** de la ventana (cada cuánto las calculo)

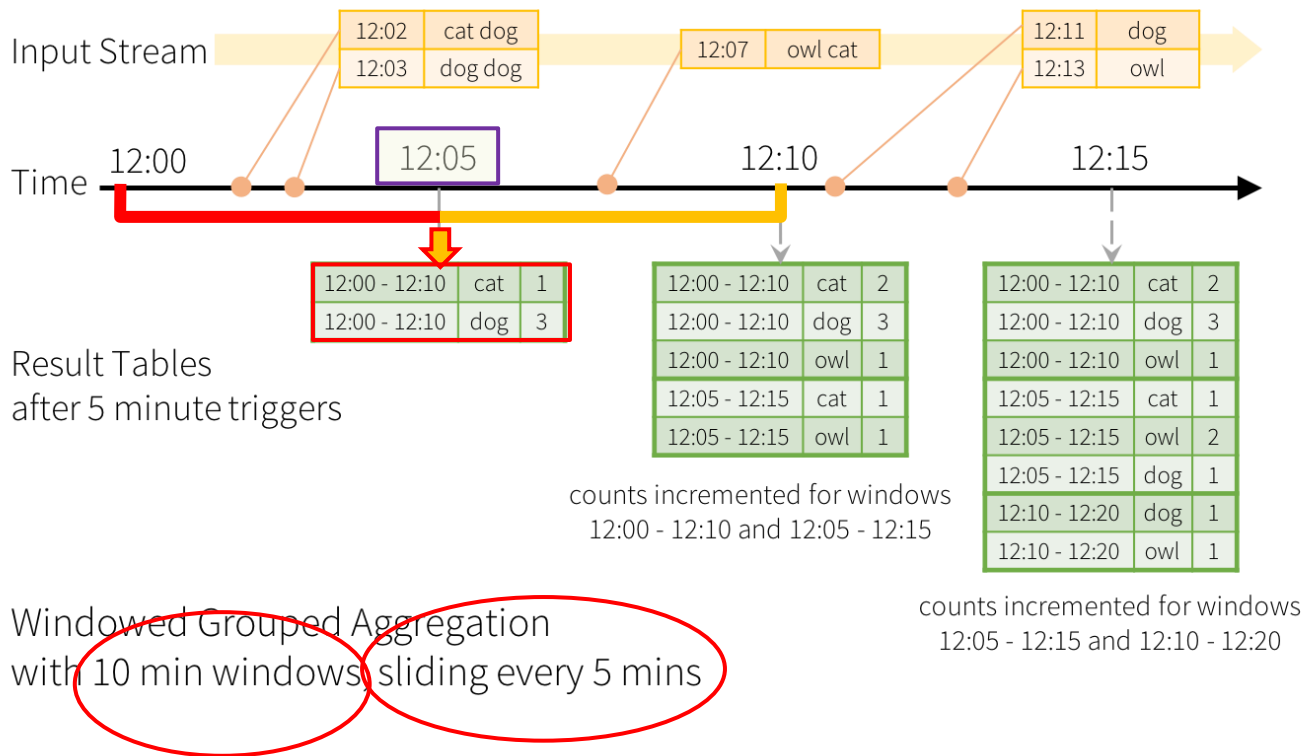
```
# streaming DataFrame with  
# schema: { timestamp: Timestamp, word: String }  
dfwords = ...
```

```
# Group the data by window and word and compute  
# the count of each group
```

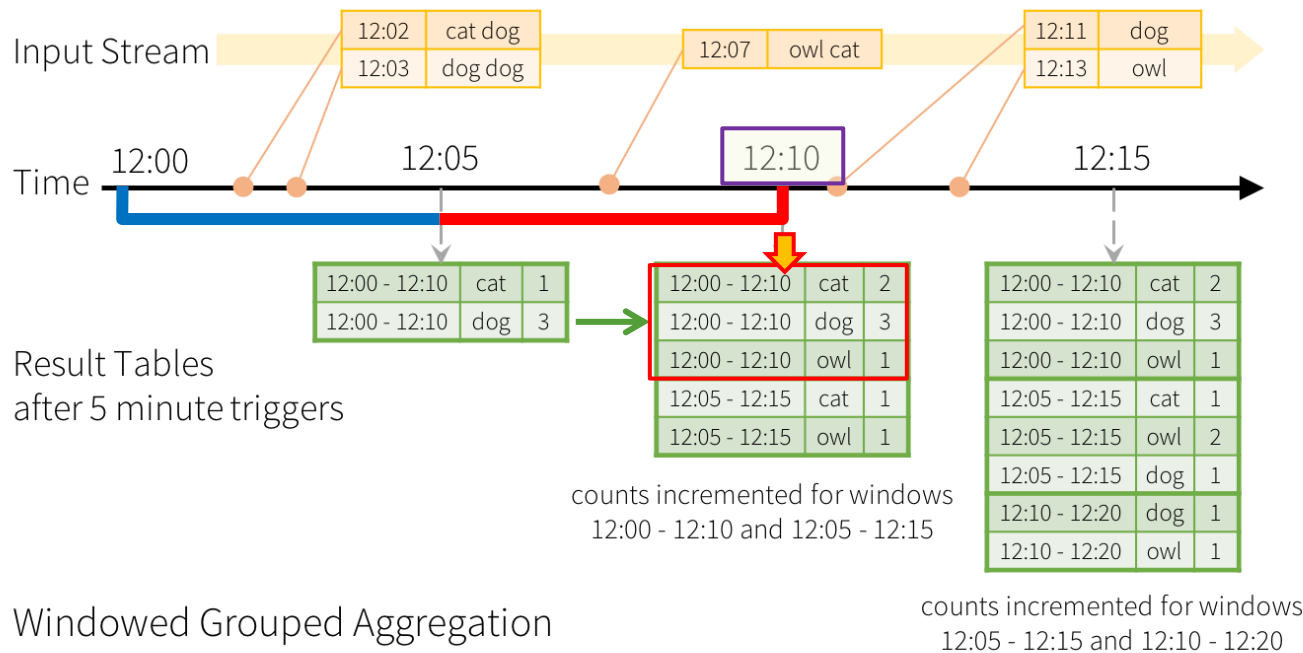
```
windowedCounts = dfwords.groupBy(  
    window(dfwords.timestamp, "10 minutes", "5 minutes"),  
    dfwords.word  
)
```



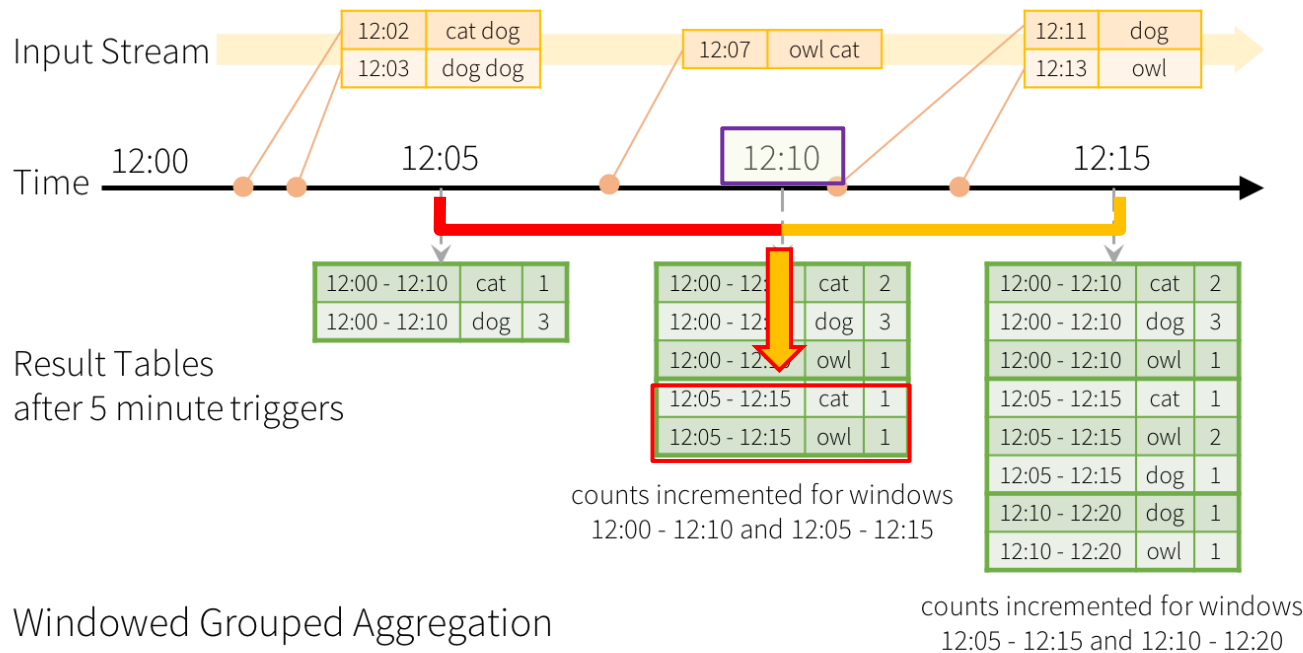
Procesado de ventanas



Procesado de ventanas

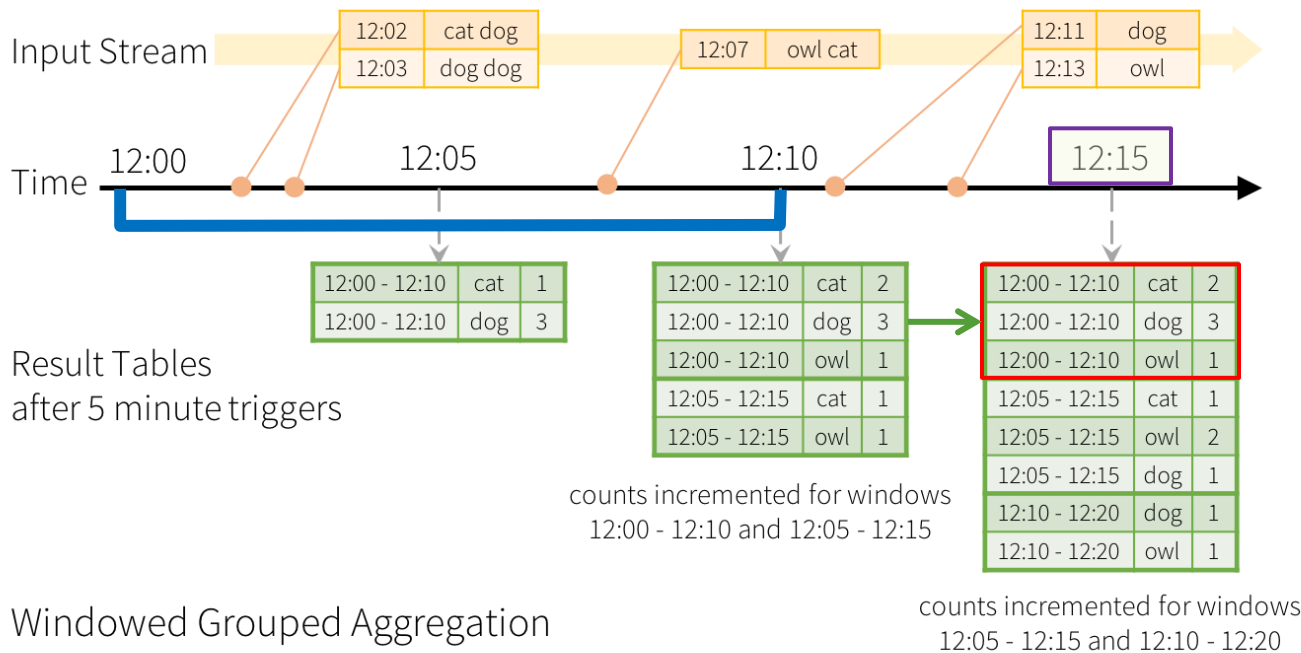


Procesado de ventanas



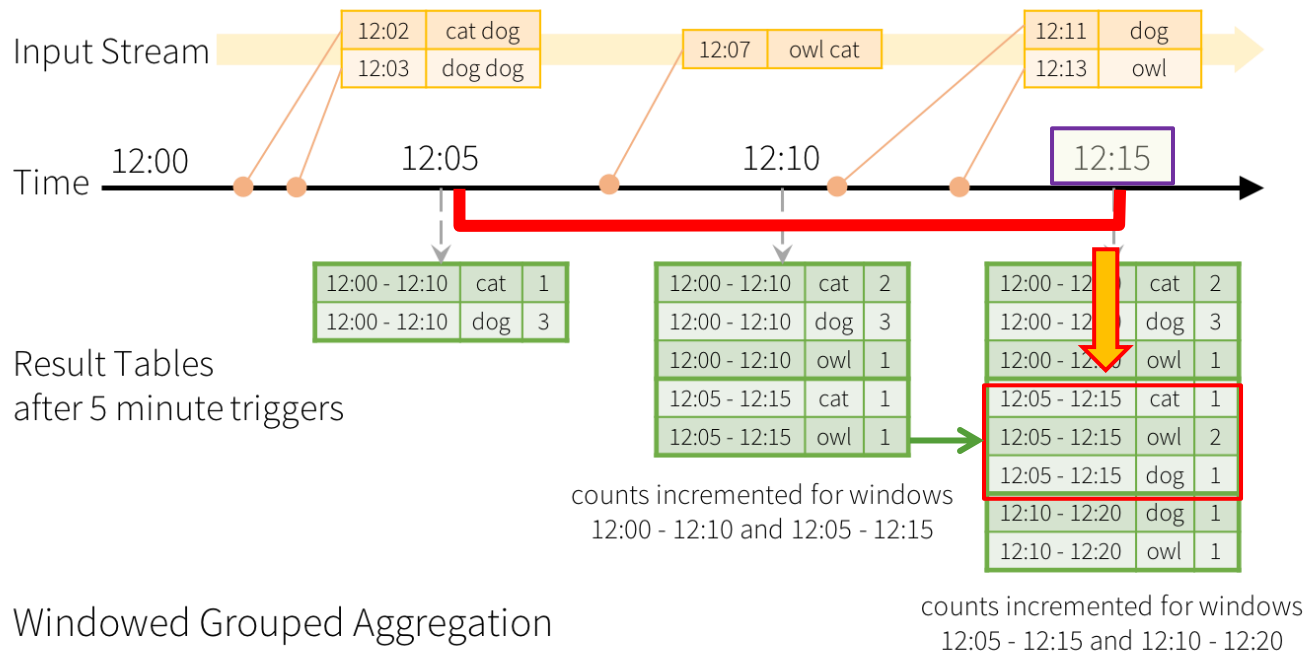
Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins

Procesado de ventanas

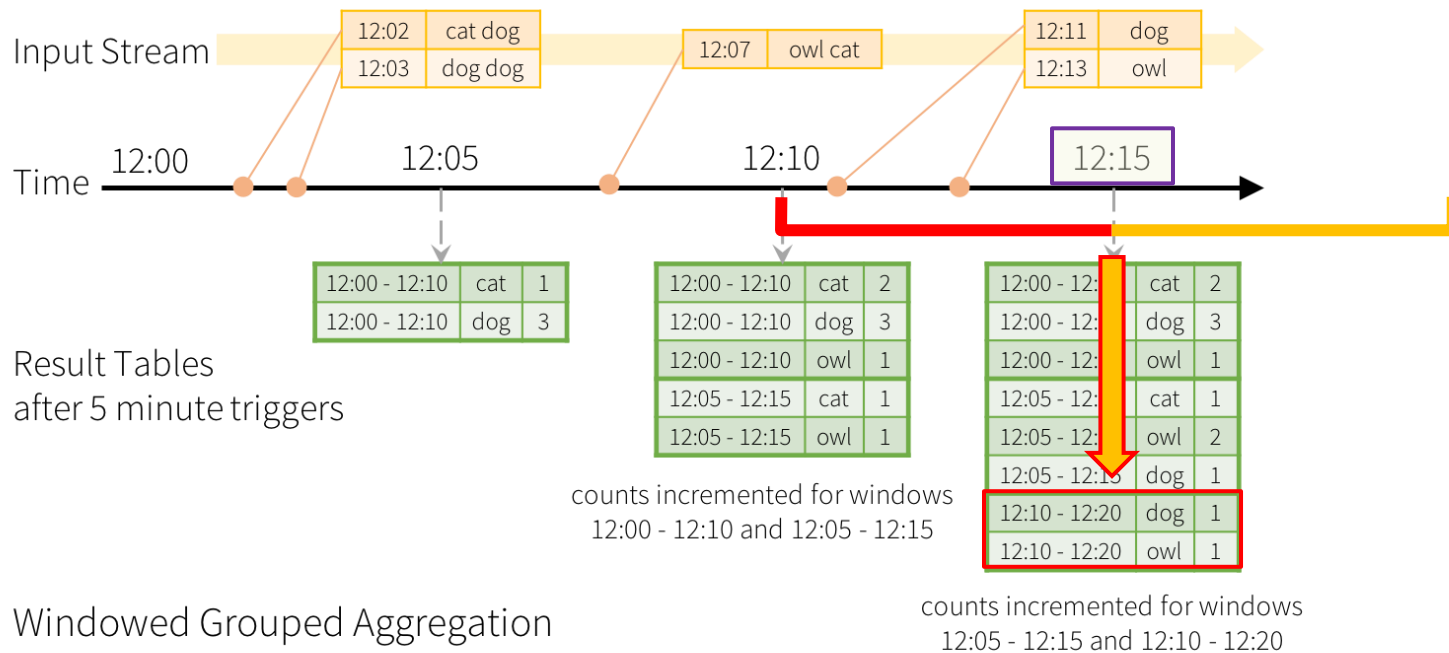


Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins

Procesado de ventanas



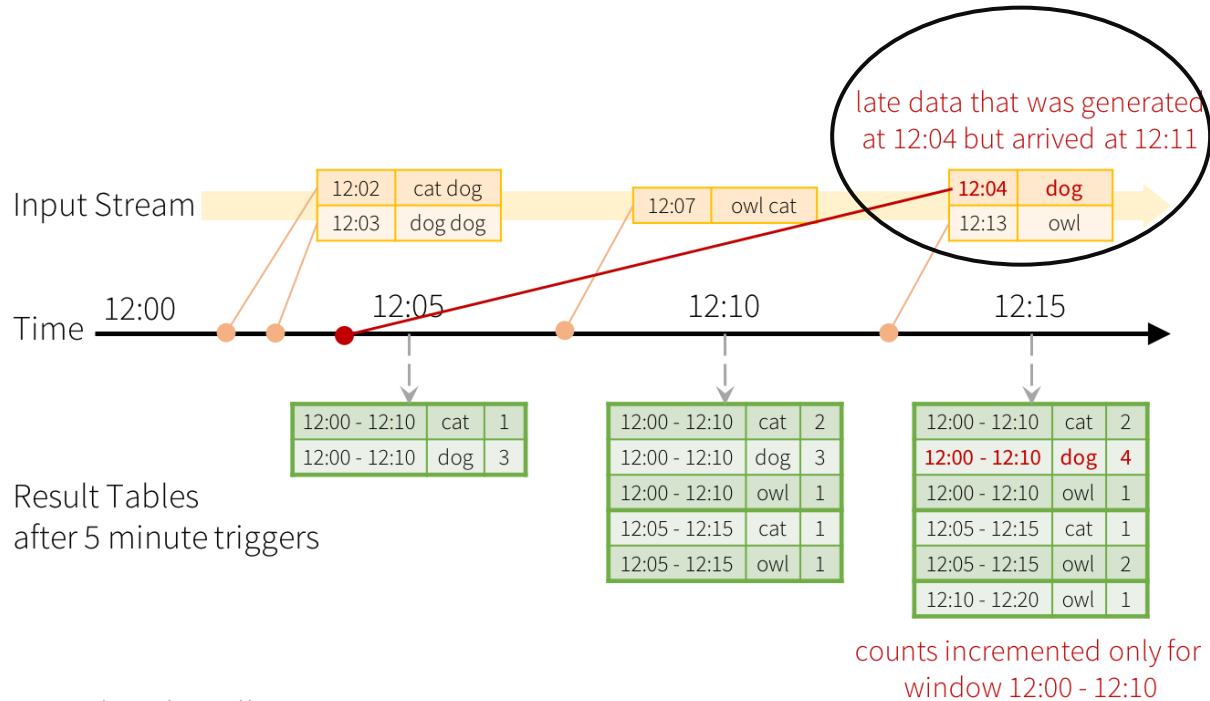
Procesado de ventanas



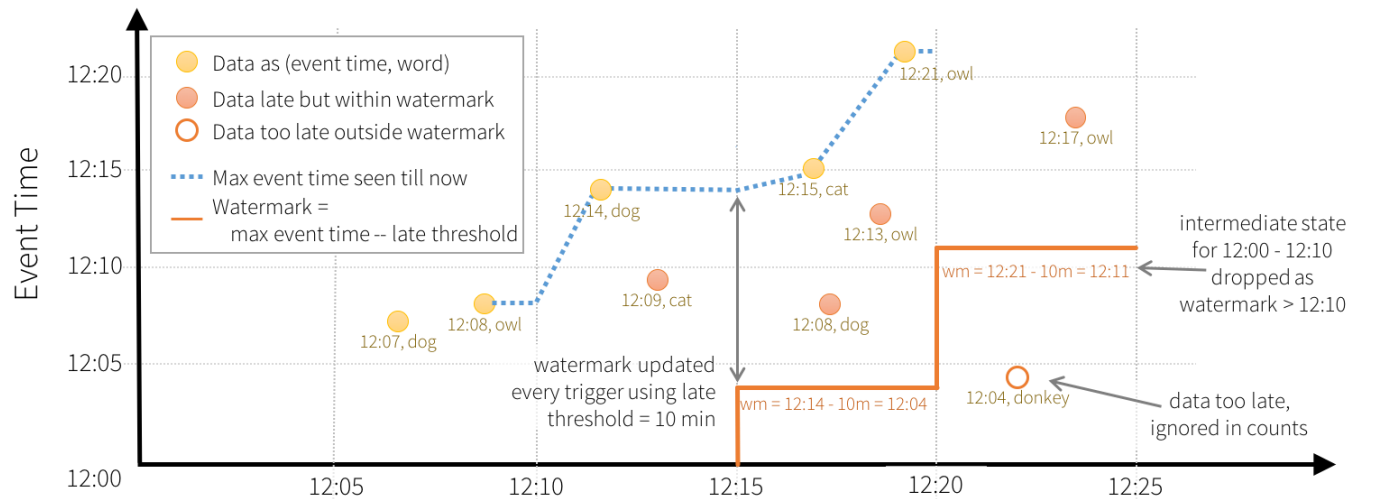
Late data & watermarking

- El procesado por ventanas acepta datos fuera de tiempo (datos con marca de tiempo anterior a su instante de llegada) y los asigna a las ventanas correctas
- Para evitar que el estado guardado sea excesivo hay que añadir una marca de tiempo límite (*watermarking*) que hace descartar los datos demasiado antiguos.
- Sólo válido en modos **update** or **append**

```
windowedCounts = words \
    .withwatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```



Late data handling in
Windowed Grouped Aggregation



Processing Time
with 5 min triggers

Result Tables after each trigger

12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	1

12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:00 - 12:10	cat	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	2
12:05 - 12:15	cat	1
12:10 - 12:20	dog	1

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	1
...		

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	2
...		

table *not*
updated with
too late data
(12:04, donkey)

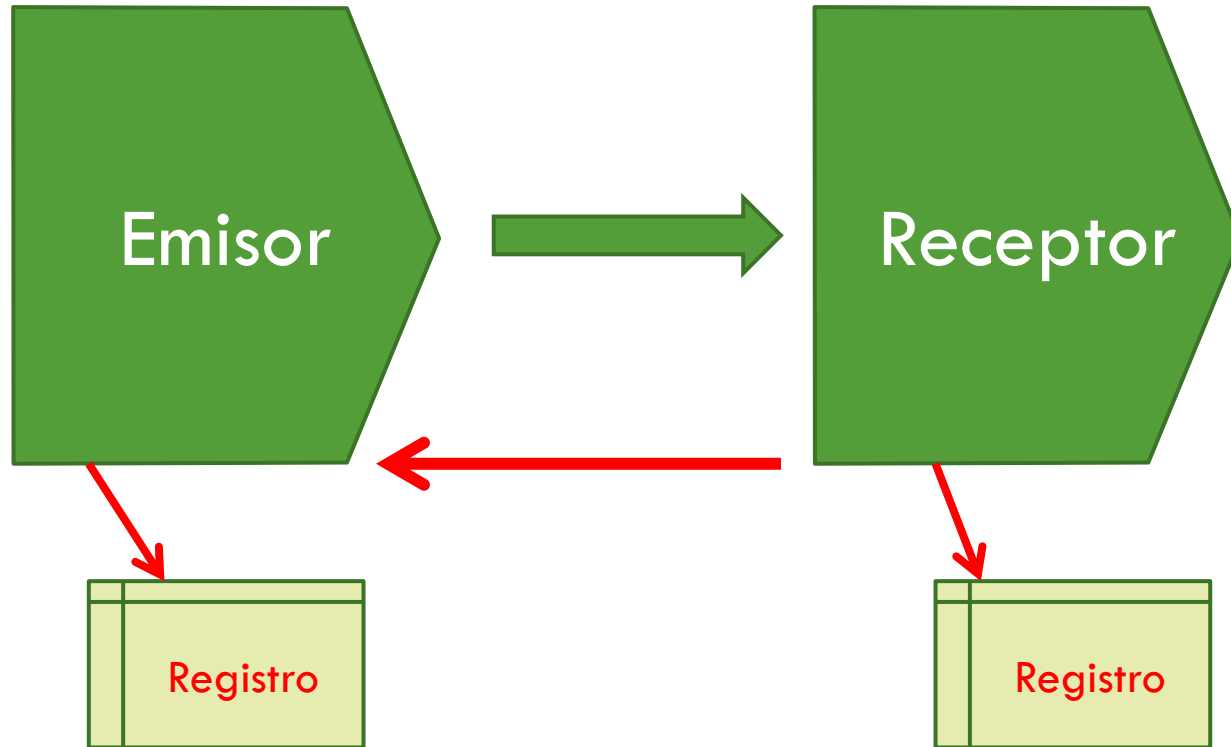
table updated
with late data
(12:17, owl)

purple rows are updated rows that
are written to the sink as output

Watermarking in Windowed
Grouped Aggregation with Update Mode

Semántica de entregas

Entregas y registro



Variante	Número de copias recibidas
at-most-once	0 o 1
at-least-once	1 o más
exactly-once	1

Funcionalidad extra

Desduplicado de datos

- Apropiado para streams con política *at-least-once*
- Con o sin watermarking
- Usa una de las columnas del DataFrame como identificador

```
streamingDf.dropDuplicates(column)
```

Joins

➤ Stream + Static

- Une un DataFrame proveniente de un `readStream` con un DataFrame estático
- Permite [inner joins y algunos outer joins](#)

➤ Stream + Stream (Spark \geq 2.3)

- Necesita activar watermarking en los dos streams
- Requiere (en función del modo) establecer condiciones del join que dejen fuera filas en función de su marca de tiempo (*event-time conditions*)

Continuous Processing

Procesado continuo

- Nuevo modo a partir de Spark 2.3
- Marcado como *experimental* (todavía en Spark 3.2)
- *Soporta solo parte de las operaciones*

```
spark \  
  .readStream \  
  .format(...) \  
  .load() \  
  .selectExpr(...) \  
  .writeStream \  
  .format(...) \  
  .trigger(continuous="1 second") \  
  .start()
```

Procesado continuo vs. streaming estructurado estándar

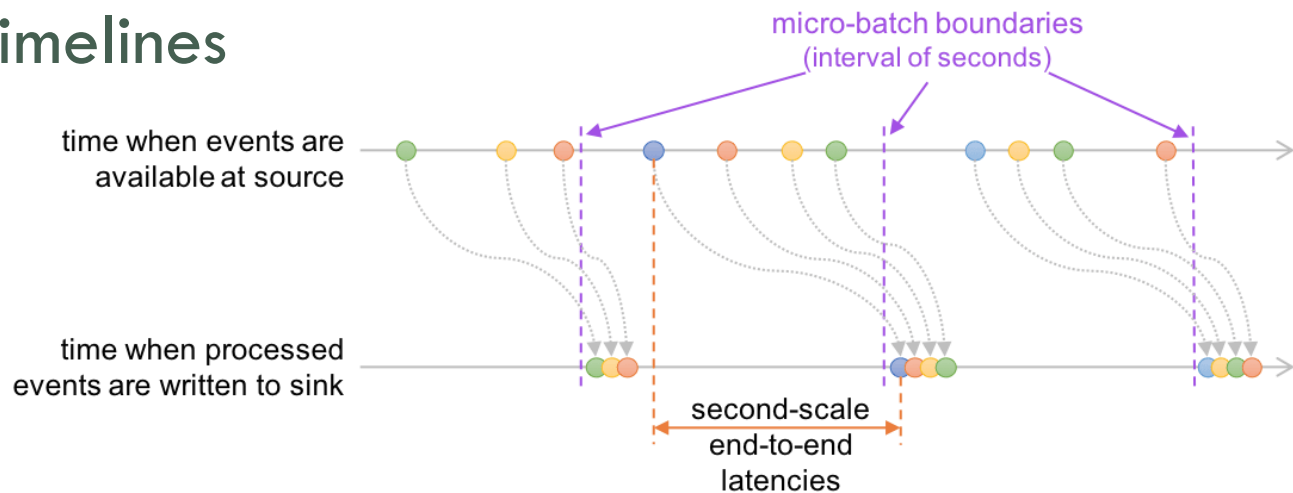
Micro-batches

- latencies of $\sim 100\text{ms}$ at best
- exactly-once guarantees

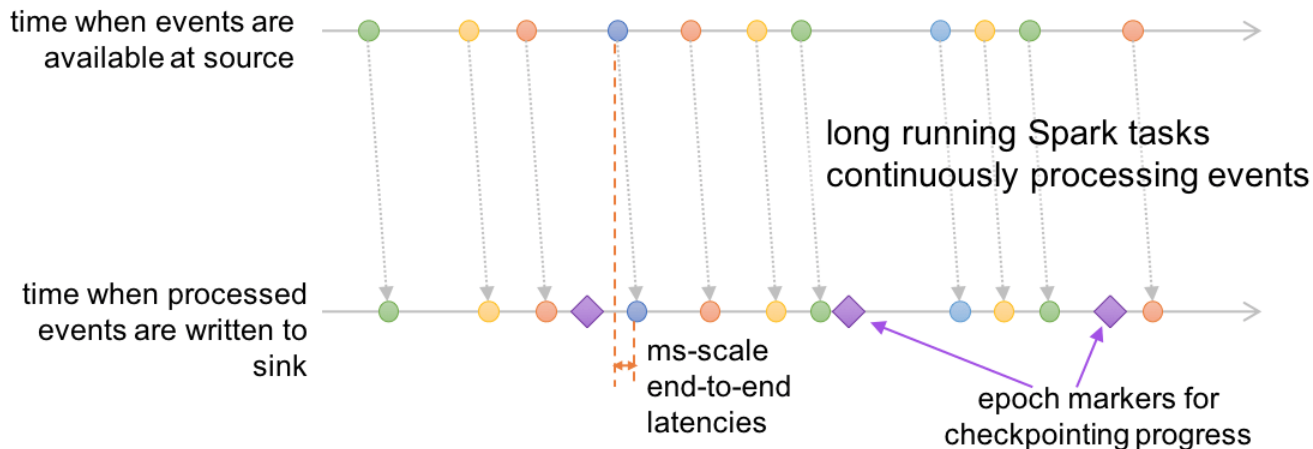
Continuous streaming

- low ($\sim 1\text{ ms}$) end-to-end latency
- at-least-once fault-tolerance guarantees

timelines



micro-batches



continuous processing

Machine Learning sobre Spark Streaming

Concepto

Machine Learning

Aprendizaje automático a partir de datos

Streaming

Los datos van llegando en tiempo real

entrenamiento

vs.

inferencia

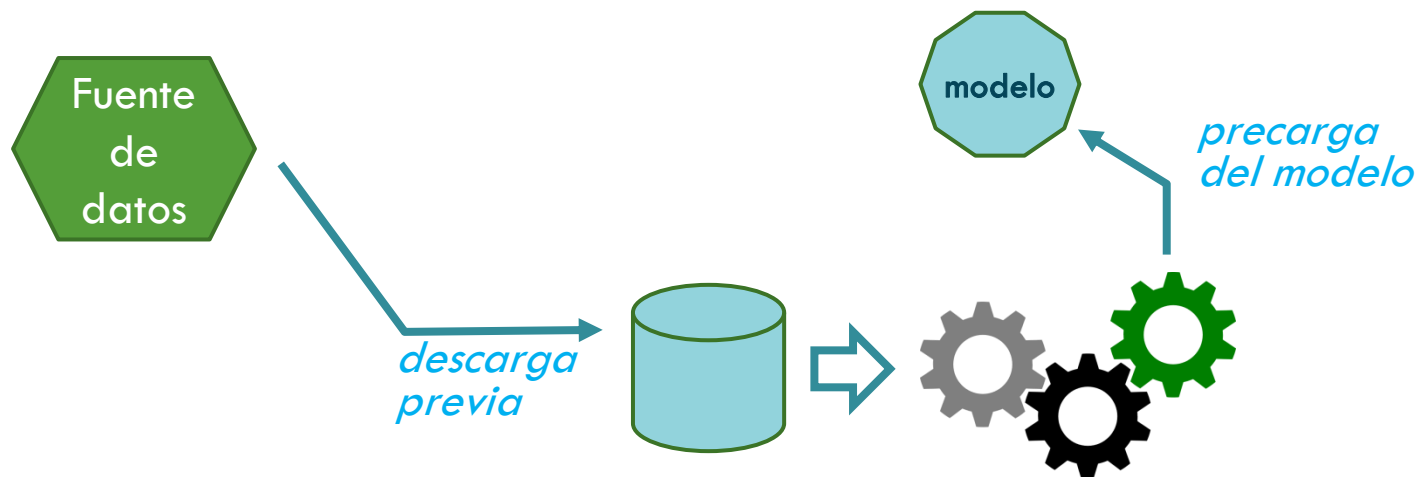
➤ Offline training

- El modelo estadístico se entrena con datos estáticos
- Luego se aplica a datos en tiempo real

➤ Online training

- El modelo estadístico se entrena en tiempo real
- El modelo se va adaptando a cambios en los datos

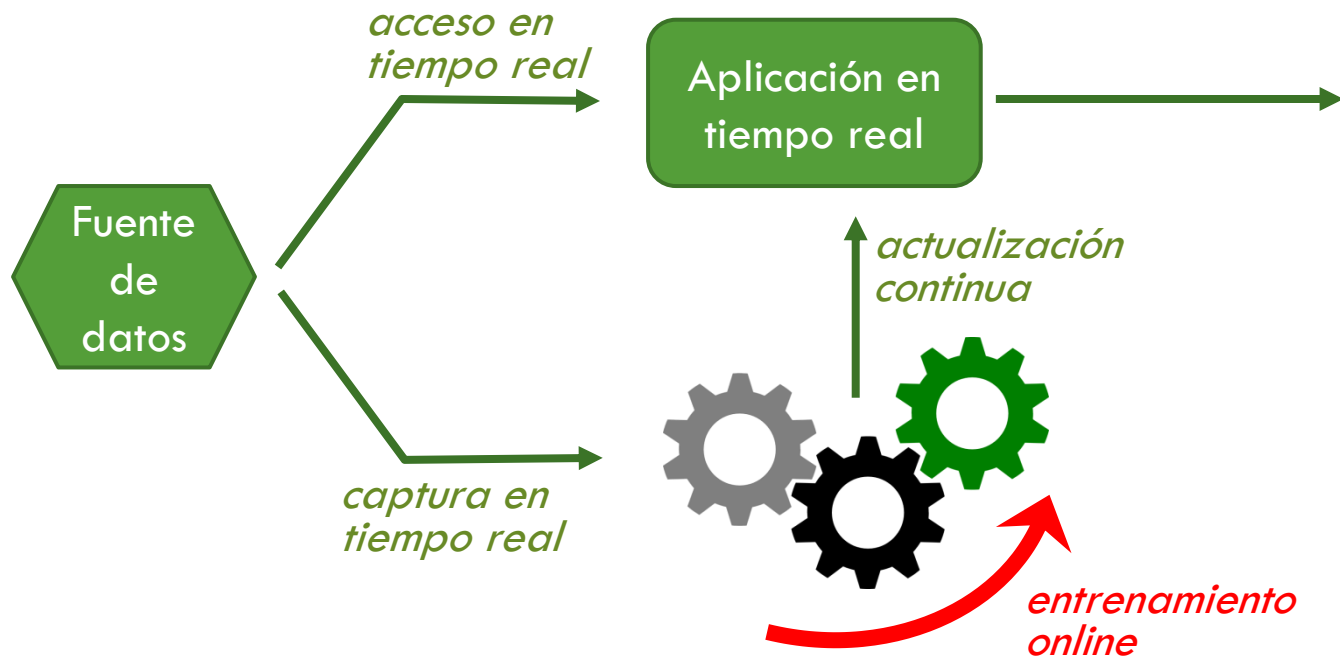
Offline training



Offline training



Online training



Modalidades (II) – Modelos mixtos

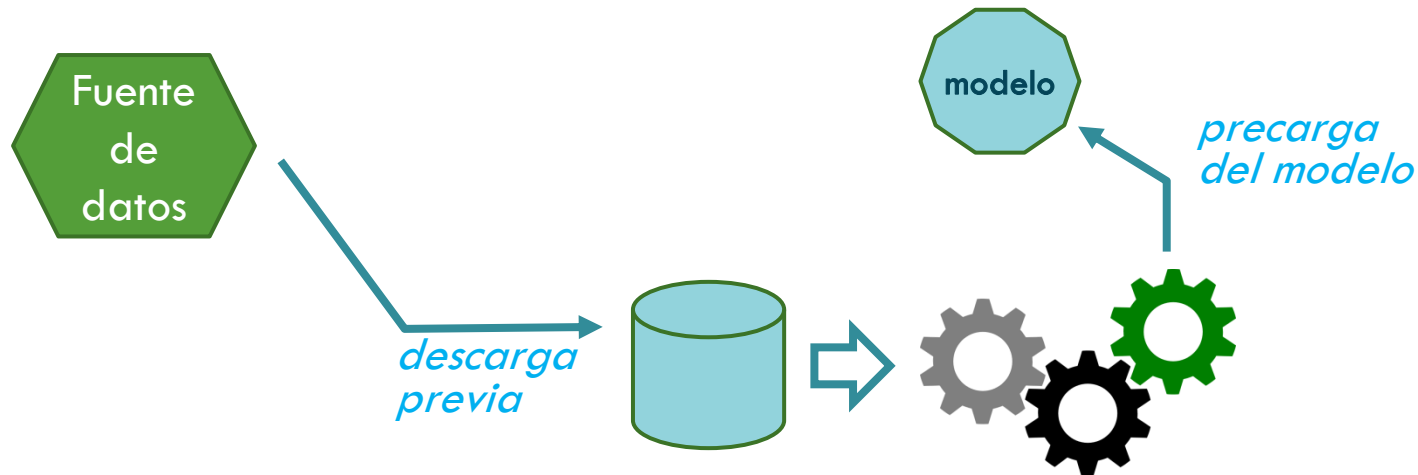
➤ **Offline Batches**

- Training offline
- Re-entrenamiento a intervalos regulares, usando lotes de datos nuevos

➤ **Online + Pre-training**

- Pre-entrenamiento de un modelo inicial offline
- Carga del modelo y actualización posterior online

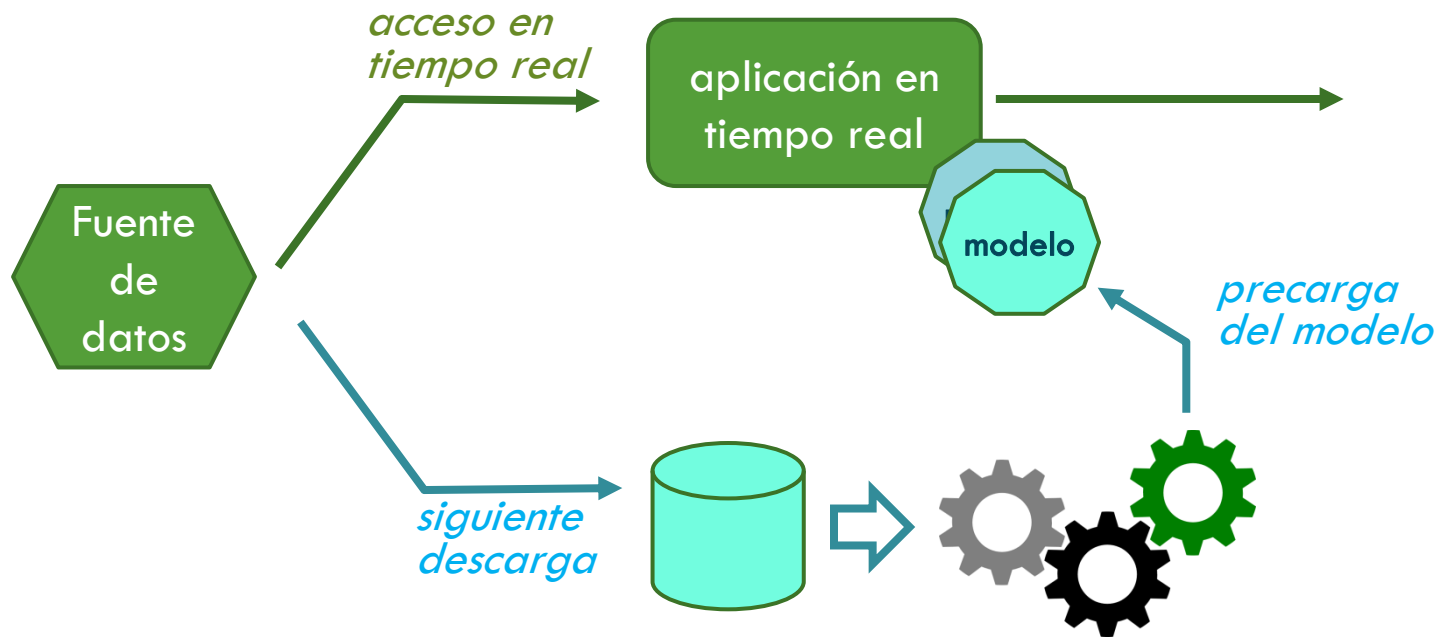
Offline batches



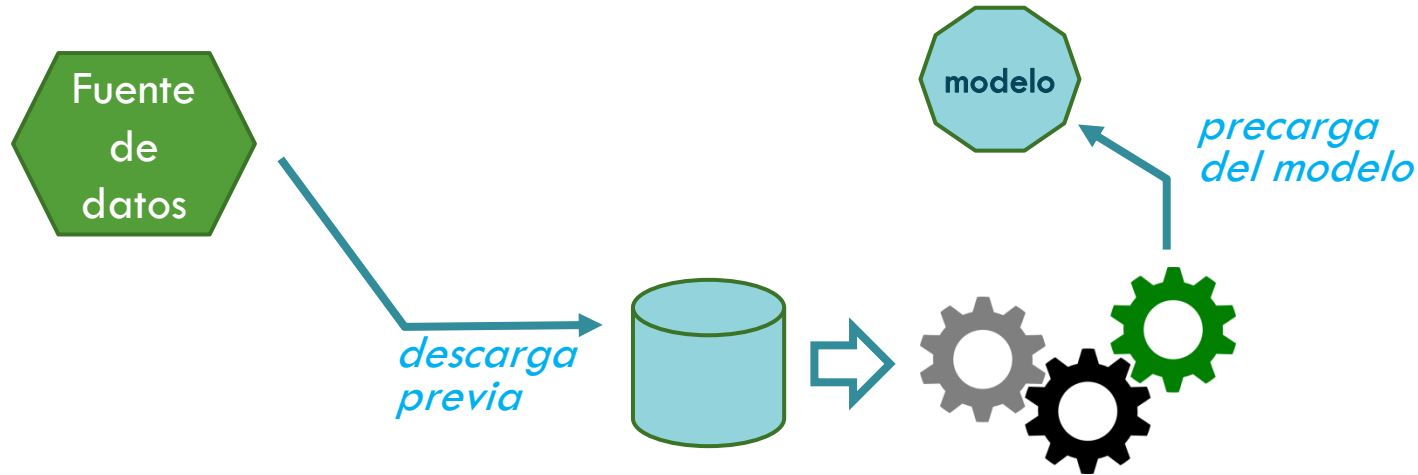
Offline batches



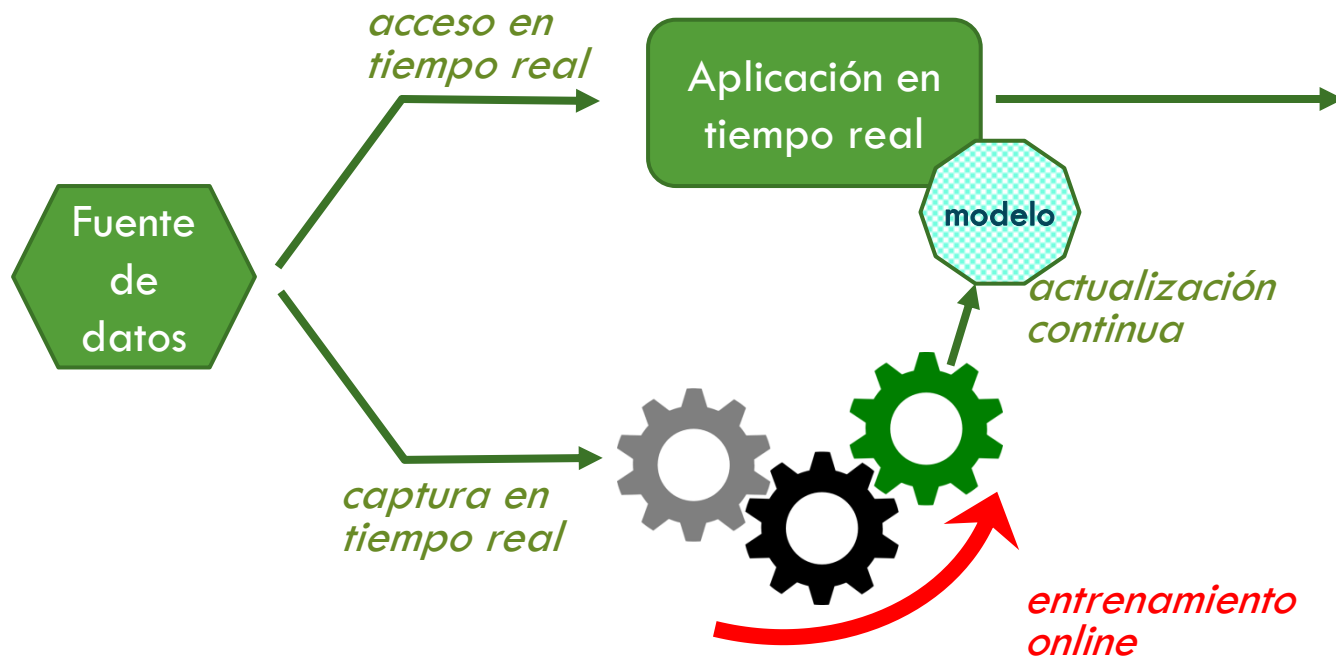
Offline batches



Online + pre-training



Online + pre-training



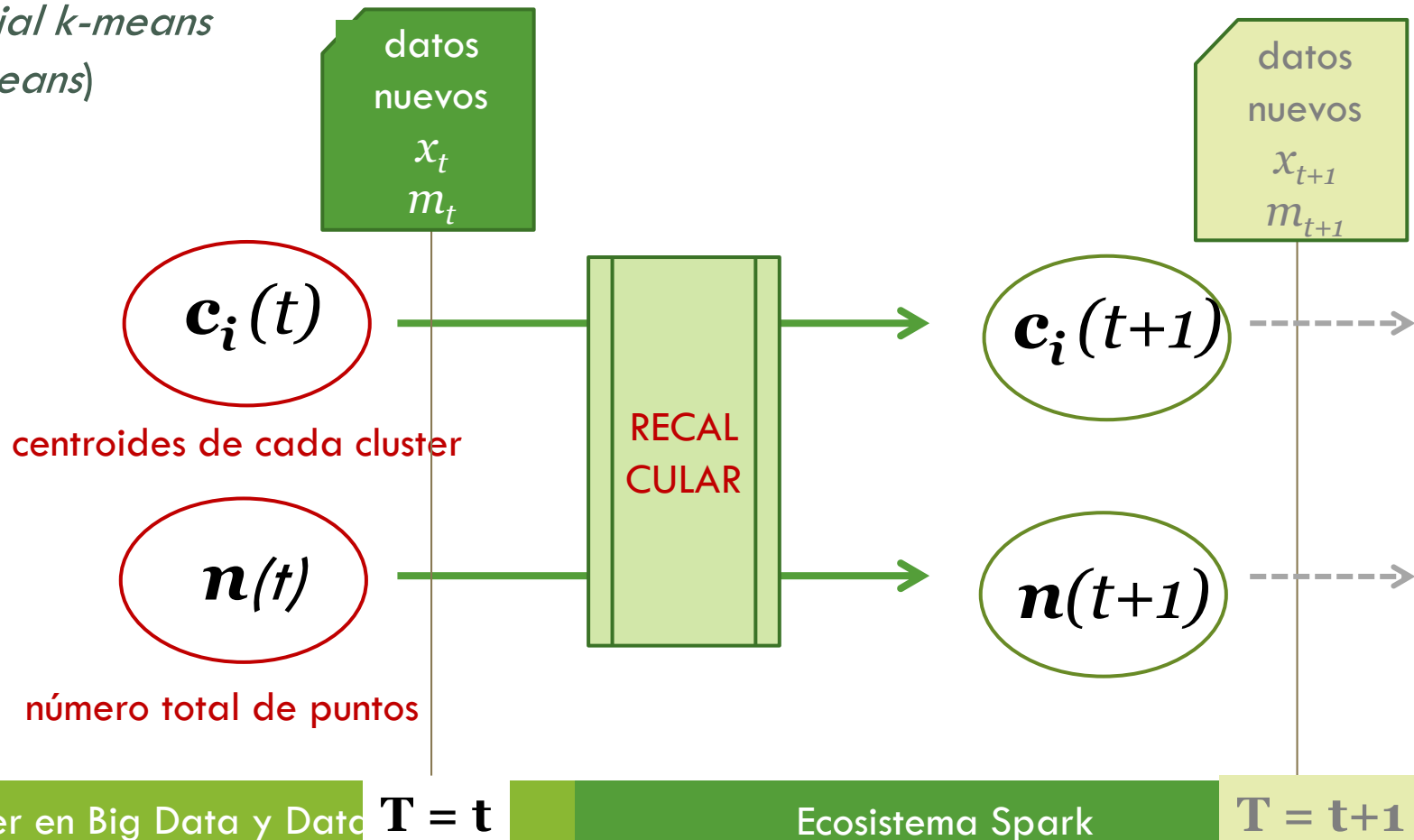
Online training

Online training en Spark Streaming

- Sólo unos pocos modelos
 - Aprendizaje supervisado: [regresión lineal](#)
 - Aprendizaje no supervisado: [k-means](#)
 - Procesado estadístico: [contrastes de hipótesis](#) (e.g. A/B Testing)
- Sólo disponible para Streaming Clásico (DStreams)

Streaming K-Means

(aka *sequential k-means*
or *online k-means*)



Streaming K-Means

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

Para cada
centroide:

$$n_{t+1} = n_t + m_t$$

Streaming K-Means

The diagram illustrates the Streaming K-Means algorithm with two main formulas. The first formula calculates the updated centroid c_{t+1} as a weighted average of the previous centroid c_t and the new data points x_t . The second formula calculates the updated number of points n_{t+1} as the sum of the previous number of points n_t and the number of new points m_t . Annotations in red and green explain the components of these formulas.

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

Annotations for the first formula:

- Valor ponderado del centroide anterior (points to $c_t n_t \alpha$)
- coeficiente de retención (points to α)
- centroide formado solo por los datos nuevos (points to $x_t m_t$)

Annotations for the second formula:

- nuevos valores (points to x_t)
- centroide del cluster en $t+1$ (points to c_{t+1})
- número de puntos en $t+1$ (points to n_{t+1})
- número de puntos anterior (points to n_t)
- número de nuevos puntos (points to m_t)

Coeficiente de retención

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

$\alpha = 0$



$\alpha = 1$

$$c_{t+1} = \frac{x_t m_t}{m_t}$$

$$c_{t+1} = \frac{c_t n_t + x_t m_t}{n_t m_t}$$

modelo sin memoria

(solo usa los datos nuevos)

- Aprende más rápido
- Oscila más

modelo uniforme

(los datos nuevos cuentan igual que los antiguos)

- Aprende más despacio
- Más estable

Streaming K-Means en Spark

- [Documentación](#)
- [API](#)
- [Ejemplo](#)

```
from pyspark.mllib.clustering import StreamingKMeans

# We create a model with random clusters and specify the number of clusters to find
model = StreamingKMeans(k=2, decayFactor=1.0).setRandomCenters(3, 1.0, 0)

# Now register the streams for training and testing and start the job,
# printing the predicted cluster assignments on new data points as they arrive.
model.trainOn(trainingStream)
```

Offline training

Offline training en Spark Streaming

- Disponible para
 - Streaming Clásico (**DStreams**) → basado en MLlib
 - Streaming Estructurado (**DataFrames**) → basado en ML

Proceso para Offline Learning

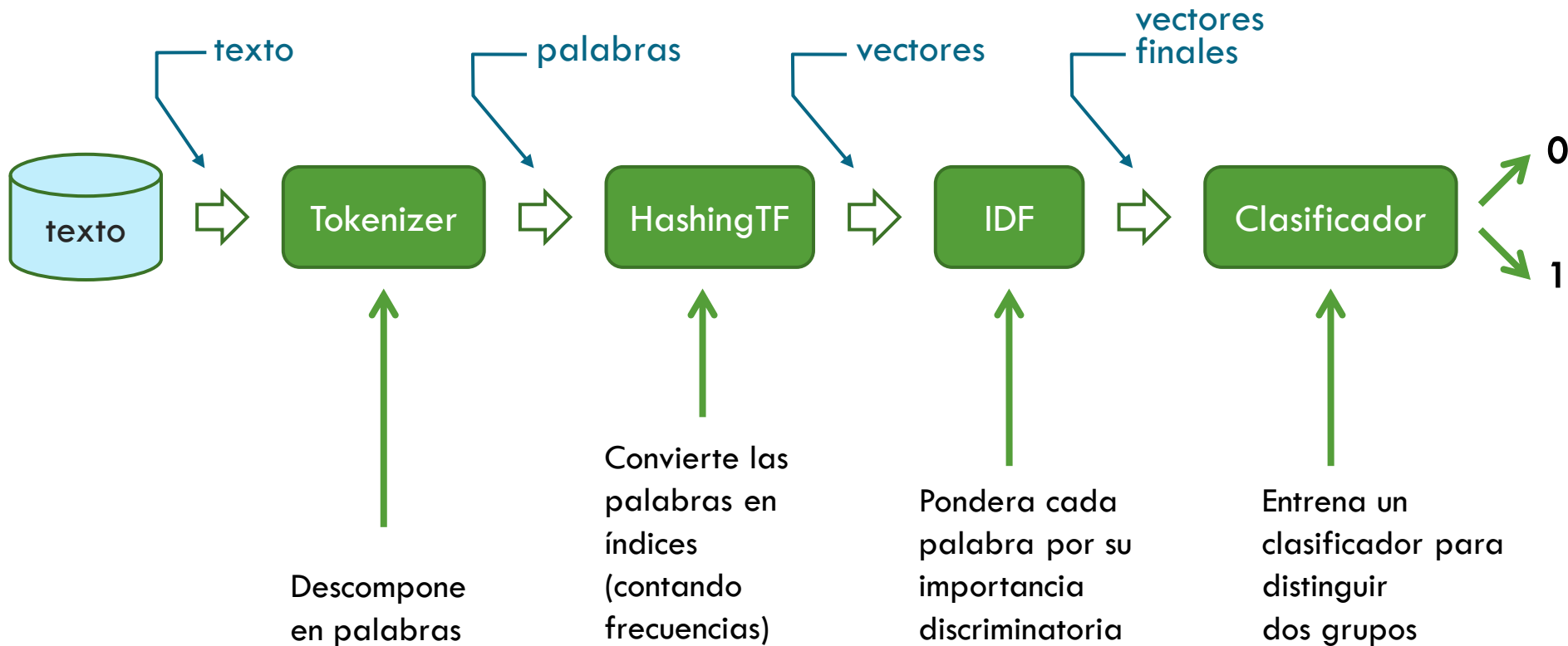
1. Entrenamiento

- Obtención de un conjunto de datos relevante
- Entrenamiento y depuración de un modelo
- Persistencia del modelo (*pipeline* de proceso)

2. Ejecución

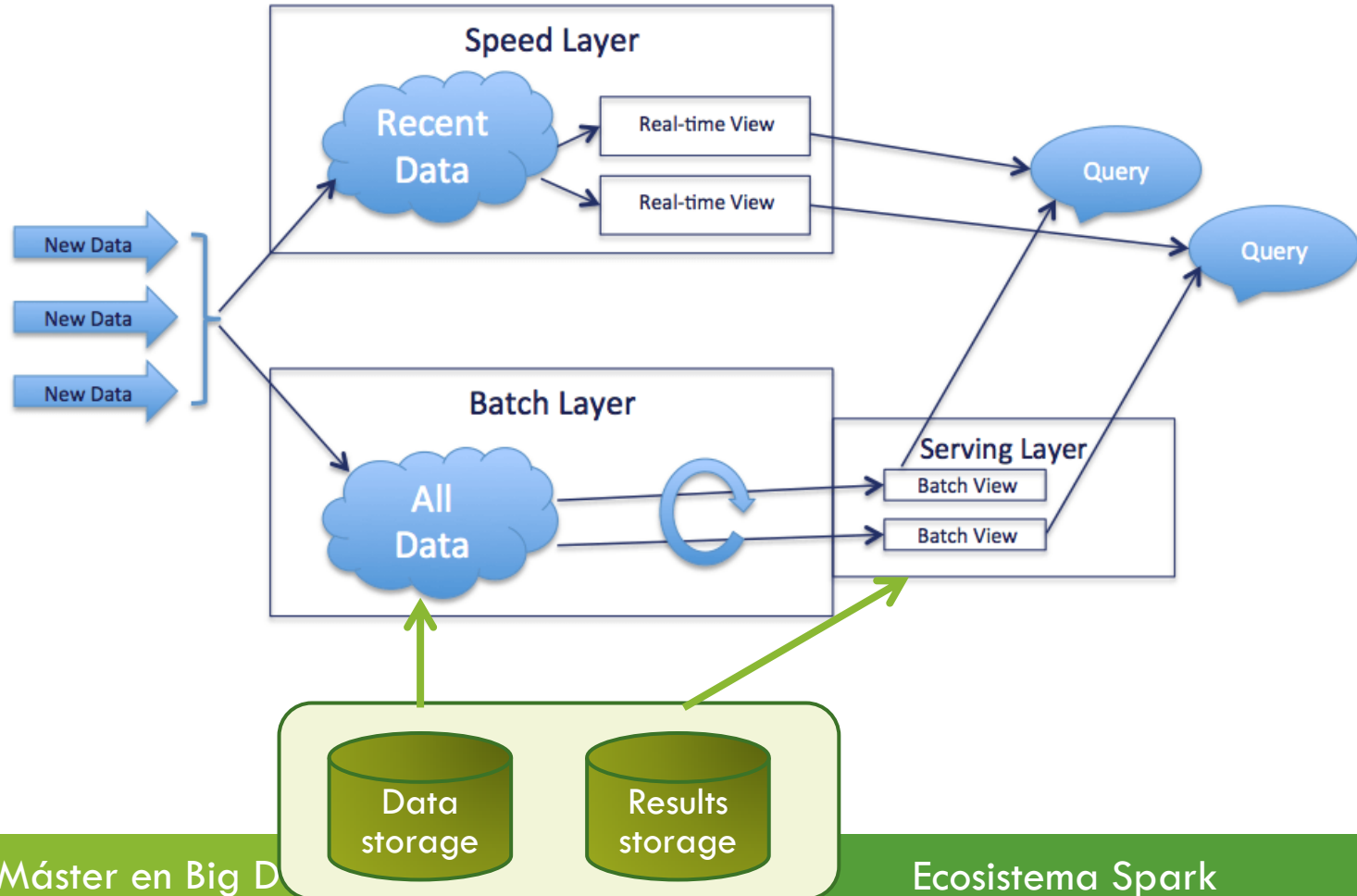
- Conexión a fuente de datos en Streaming
- Carga del modelo/pipeline
- Aplicación del modelo

Ejemplo de modelo



Apéndice: arquitecturas de tiempo real

The Lambda Architecture



The Kappa architecture

