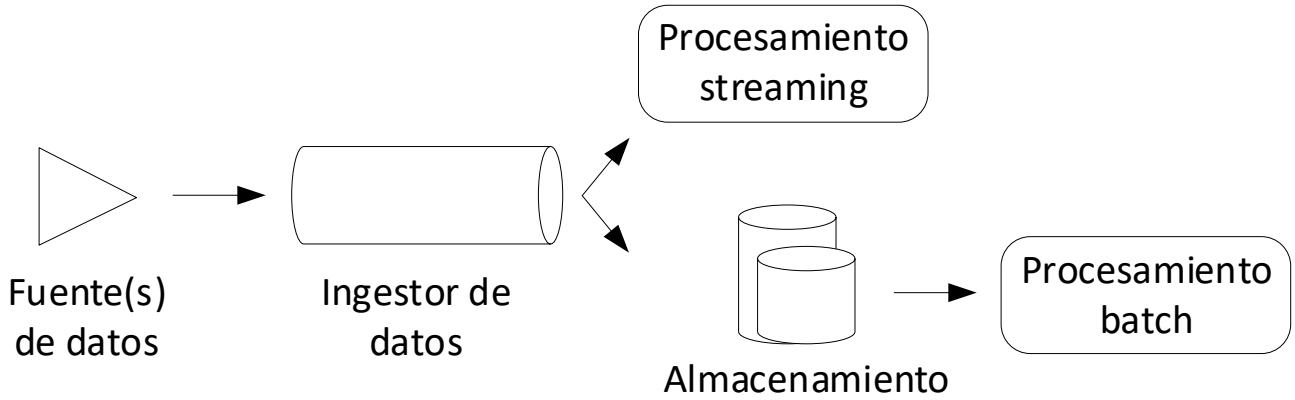
Procesamiento de datos masivos

Jesús Morán

viu | Universidad Internacional de Valencia

20.12.21

- Streaming: Datos que se generan continuamente a partir de una o varias fuentes de datos



Fuentes de datos

- Sensor de temperaturas
- Compras realizadas
- Comentarios en redes sociales
- Analítica wifi
- …

- Tecnologías:
  - Kafka
  - Kinesis
  - ØMQ
  - Flume
  - …

- Características:
  - Latencia:
    - Tiempo desde la fuente hasta que llega al sistema (o procesa)
    - Poca latencia: análisis que se tienen que obtener rápido
    - Mucha latencia: análisis que no son necesarios rápido (ej. análisis de históricos)
  - Rendimiento (Throughput):
    - Cantidad de datos que se envían en un tiempo

- Tipos de envío:
  - At-most-once delivery: Cada dato puede llegar 1 o 0 veces (pérdida)
    - Para programas que requieran baja latencia, alto rendimiento y pueden perder datos
  - At least-once: Cada dato puede llegar 1 o más veces (duplicidad)
    - Para programas que requieren sí o sí que se ejecuten datos no críticos, perdiendo rendimiento y latencia
  - Exactly-once: Cada dato llega 1 vez (sin pérdidas ni duplicidad)
    - Para programas que requieren sí o sí que se ejecuten datos críticos, perdiendo rendimiento y latencia

- **Streaming**: se procesan los datos uno a uno según se van generando
- **Micro-Batch**: se procesan subconjuntos de datos según se van generando
- **Batch**: se procesan todos los datos después que se acaben de generar
  - No siempre es posible ej. un streaming que no termine o termine en un tiempo largo

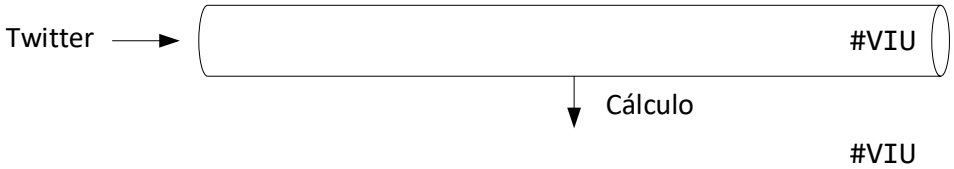
- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU

- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU

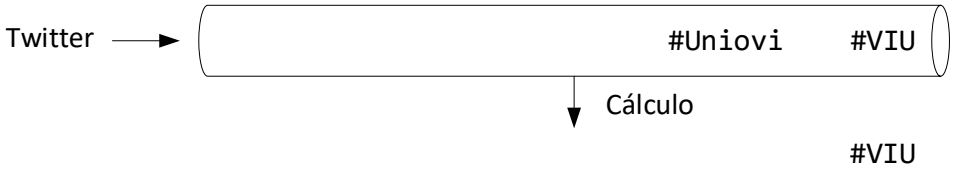
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



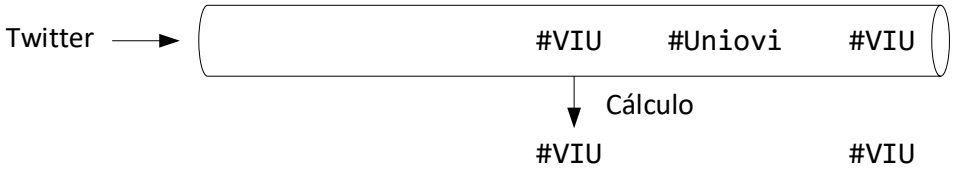
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



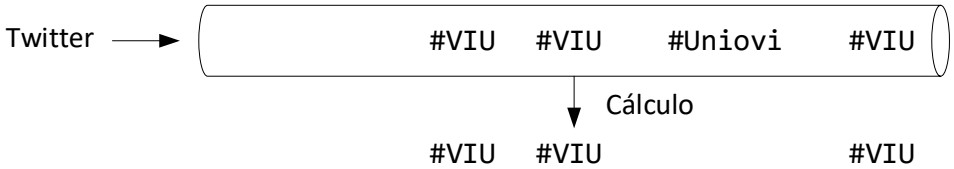
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



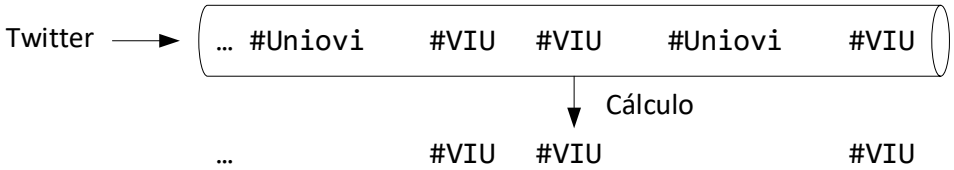
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



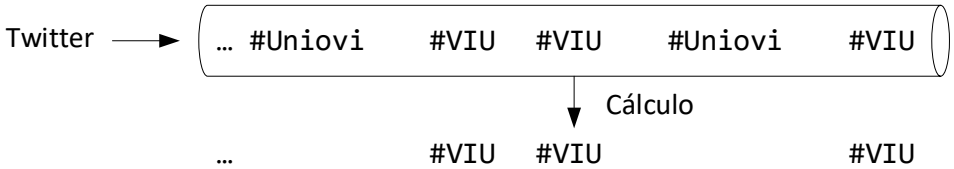
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



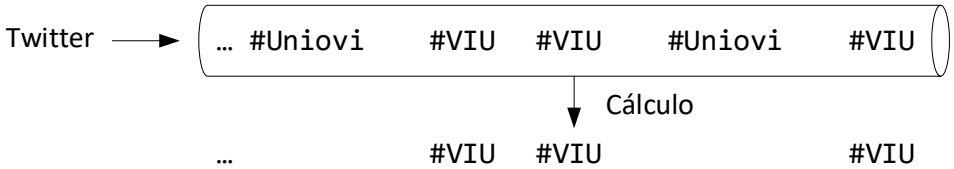
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU

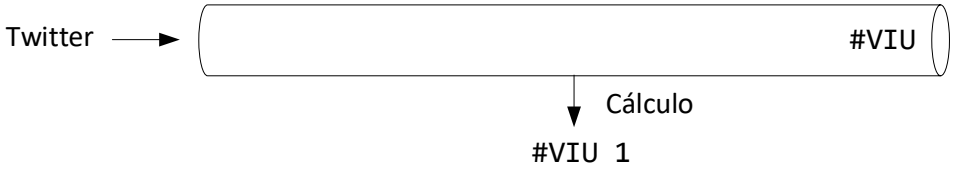


- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

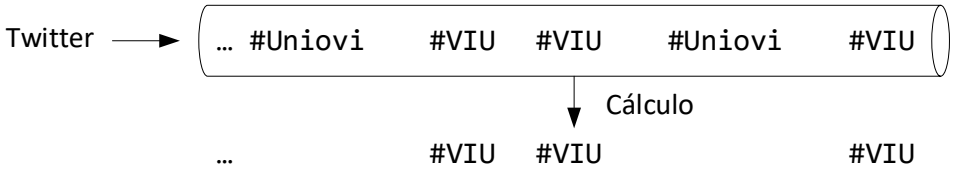
- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



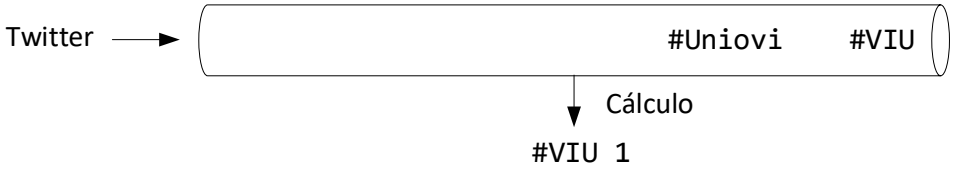
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU



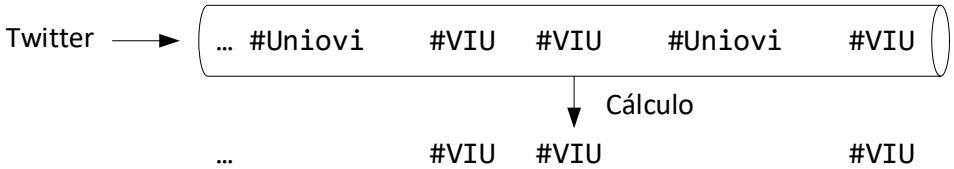
- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



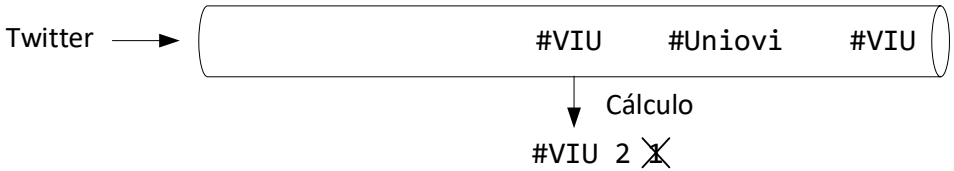
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU



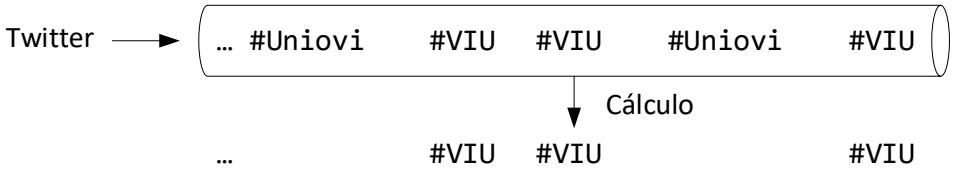
- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



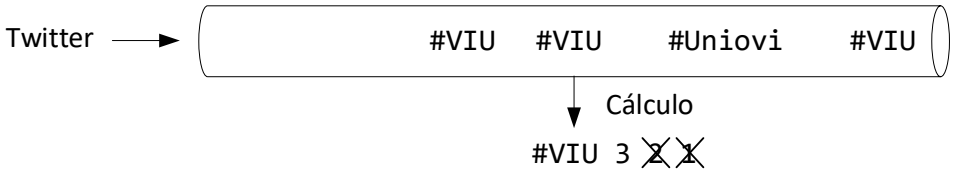
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU



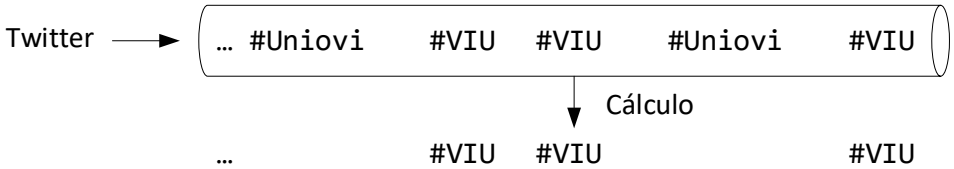
- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU



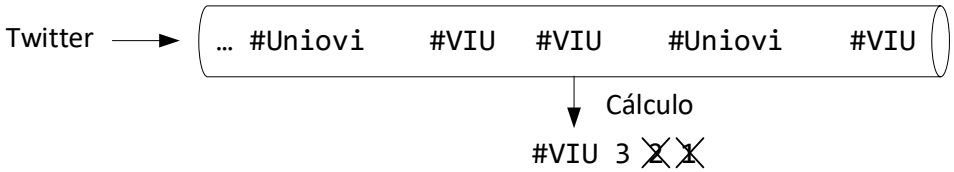
- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU



- **Stateless:**
  - El calculo no depende de ningún dato/evento anterior
  - Ej. Filtrar los tweets que sean de la VIU

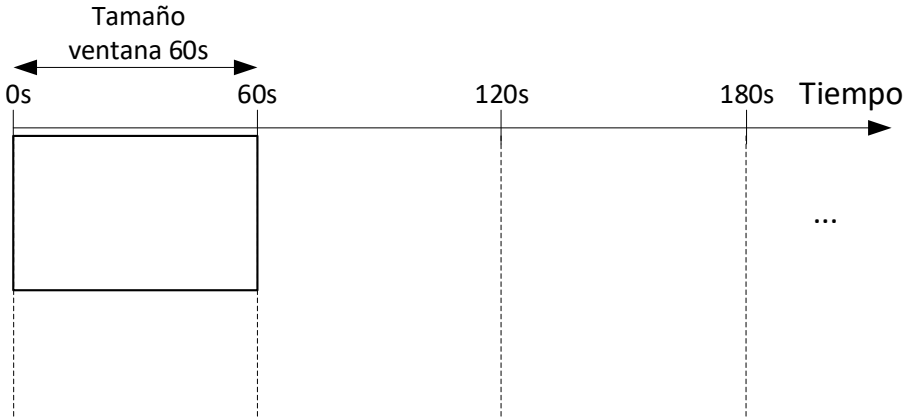


- **Stateful:**
  - El calculo depende de todos los anteriores datos/eventos
  - Ej. Contar los tweets de la VIU

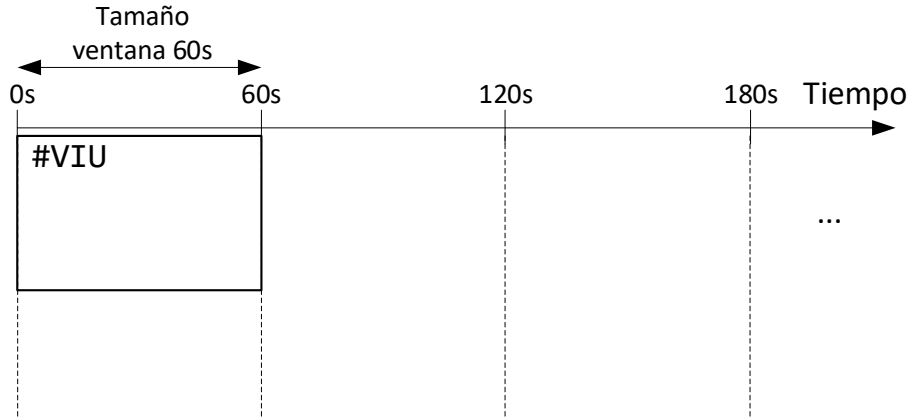


- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez

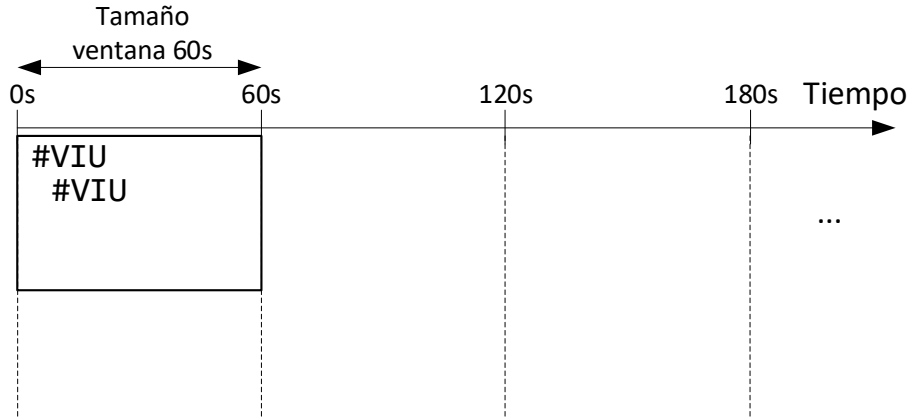
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



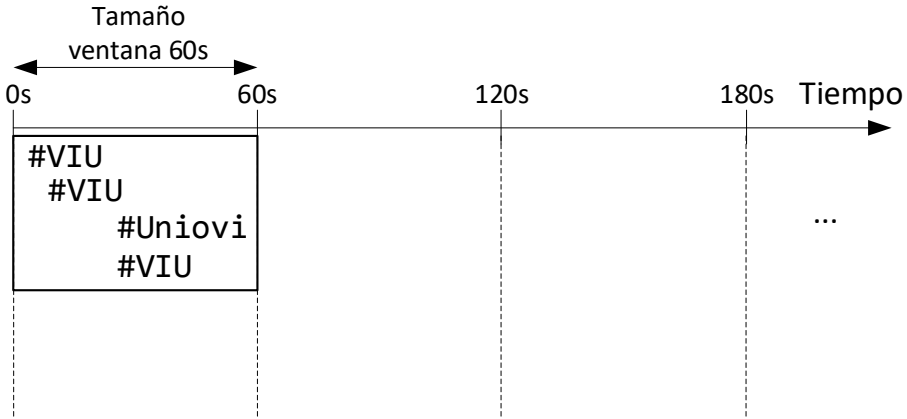
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



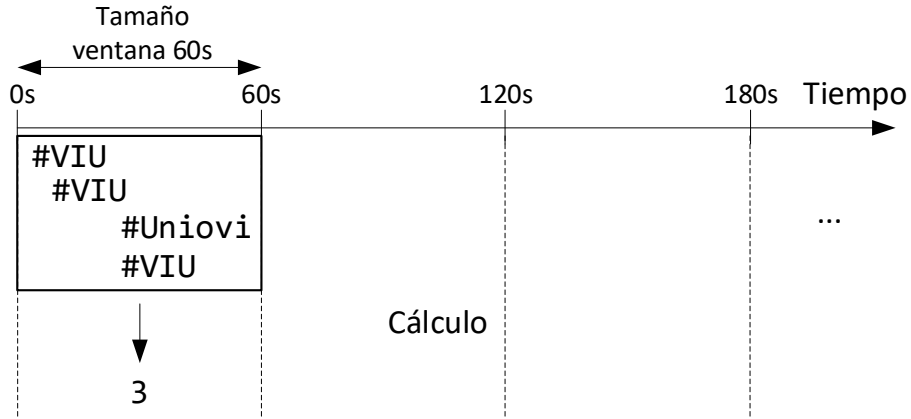
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



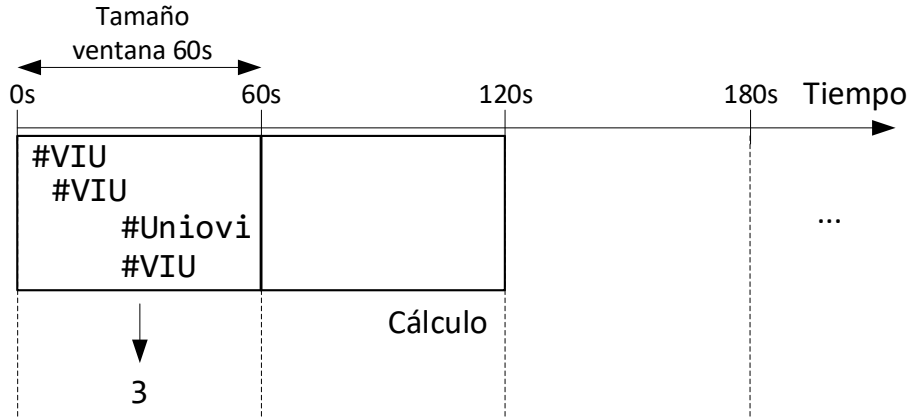
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



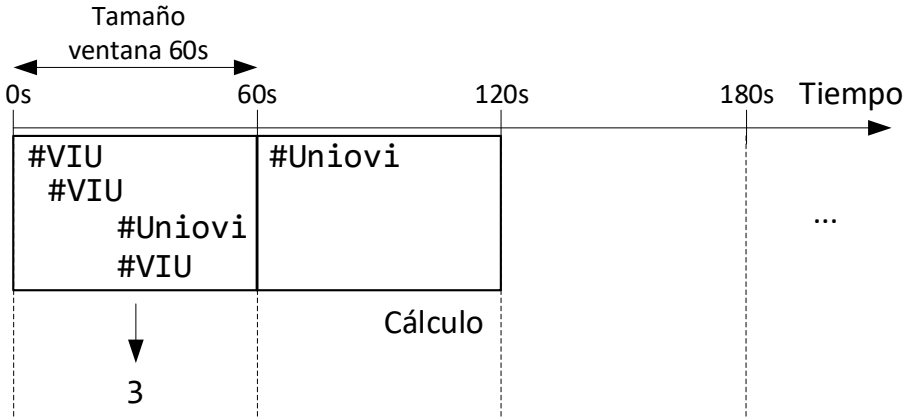
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



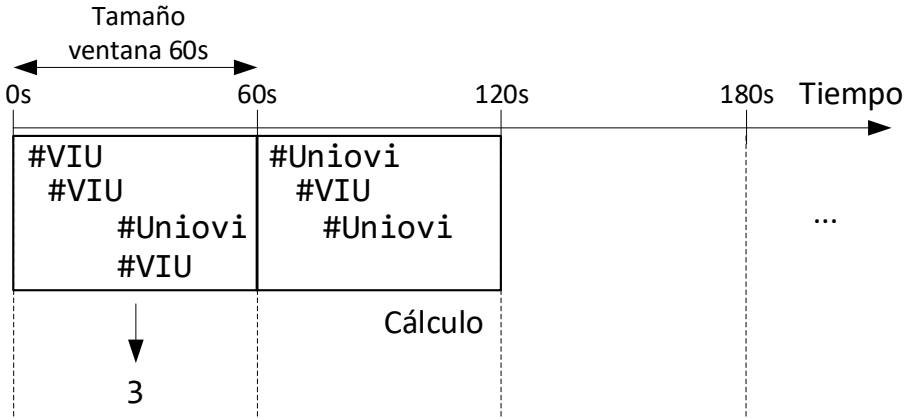
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



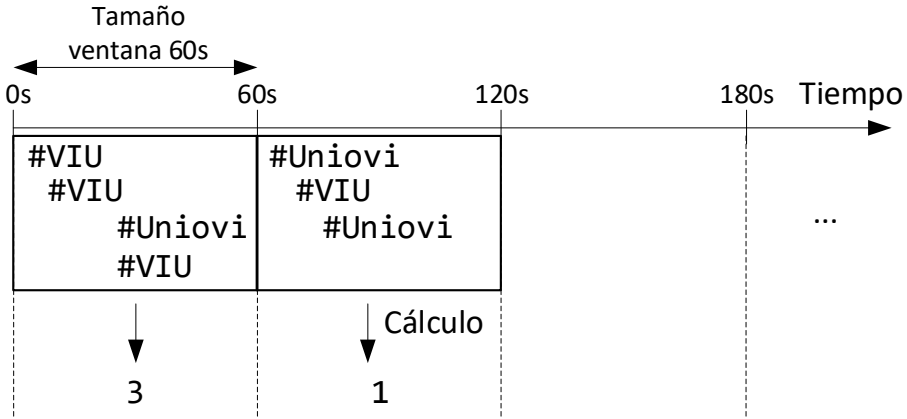
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



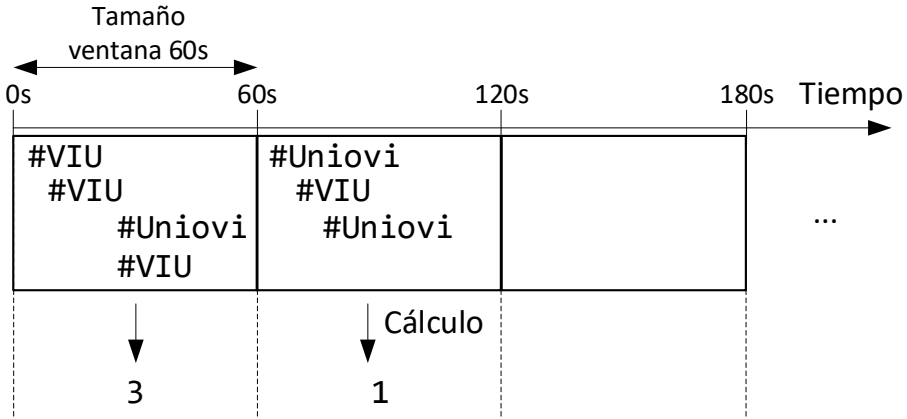
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



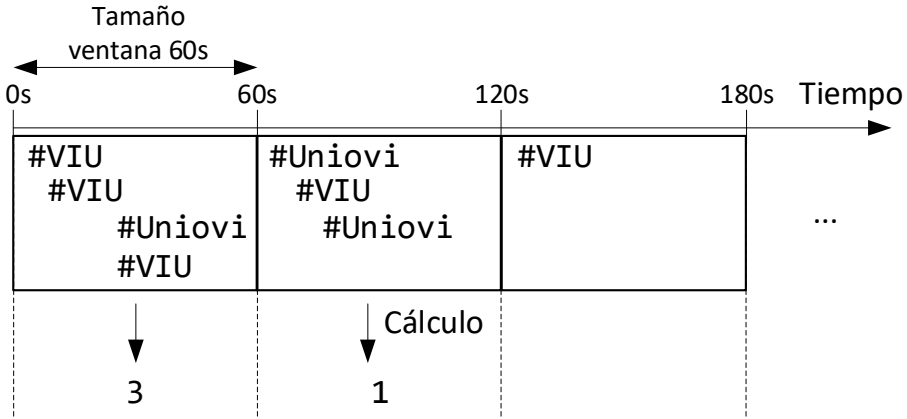
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



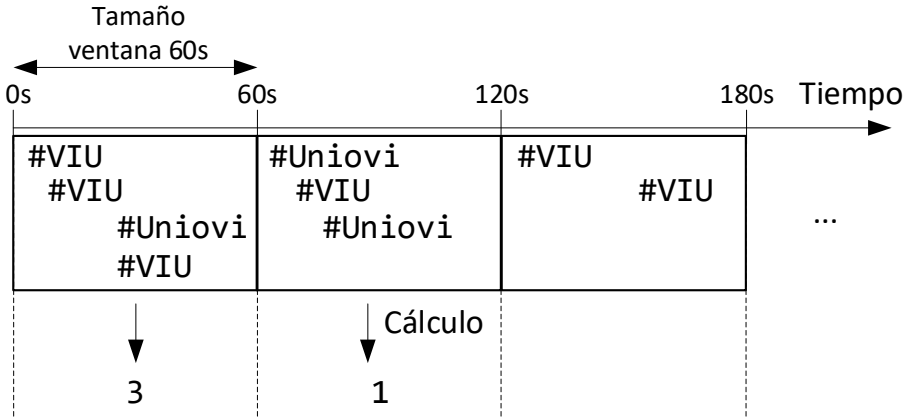
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



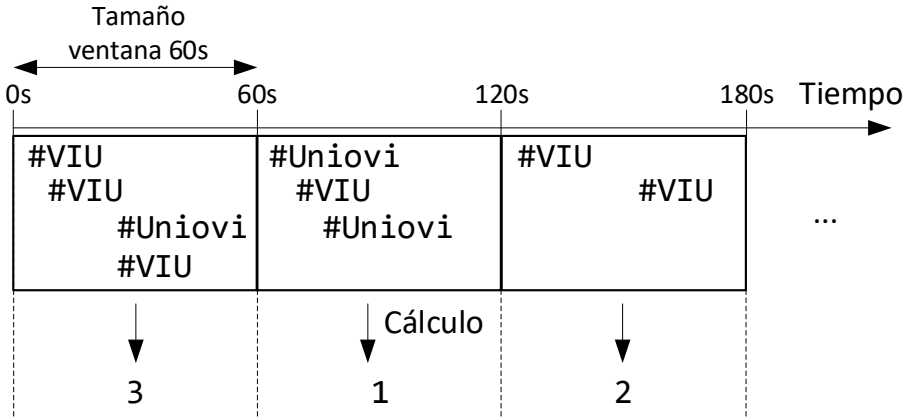
- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s



- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s

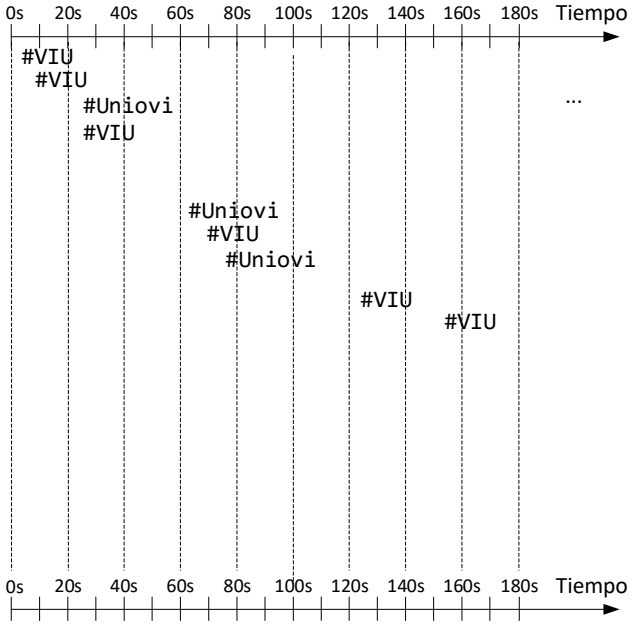


- **Fixed Window**
  - Se fija un tamaño de ventana
  - Todos los datos de esa ventana se procesan a la vez
  - Ejemplo: Obtener el número de tweets de la VIU en cada minuto -> Ventana = 60s

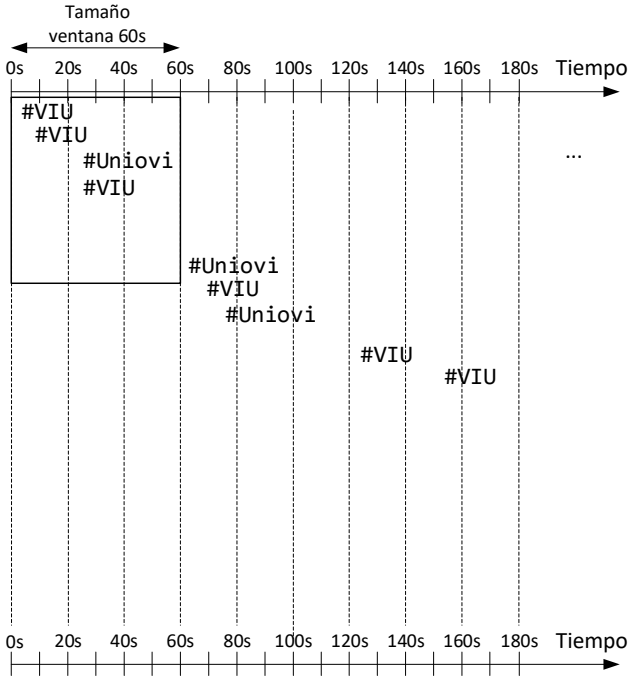


- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo

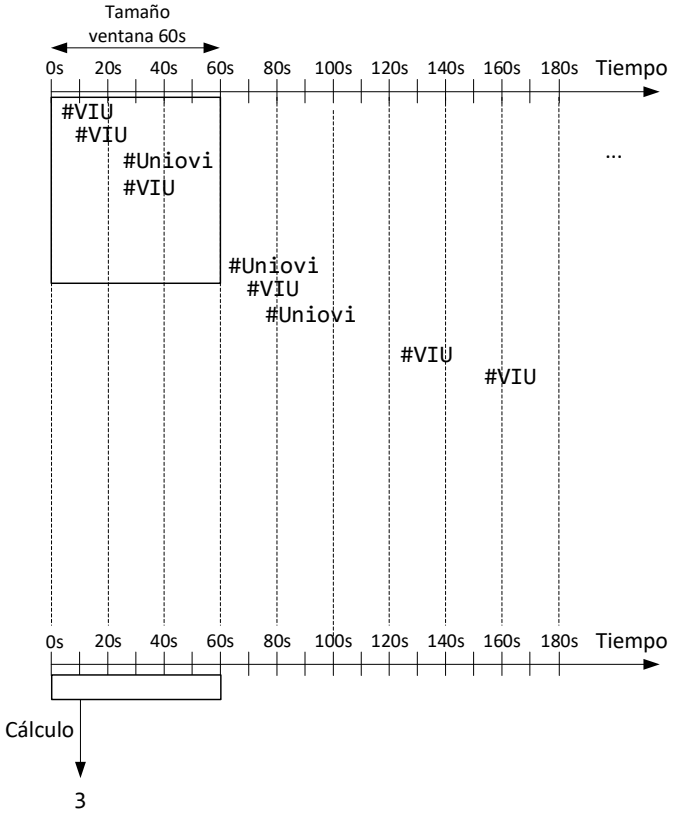
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



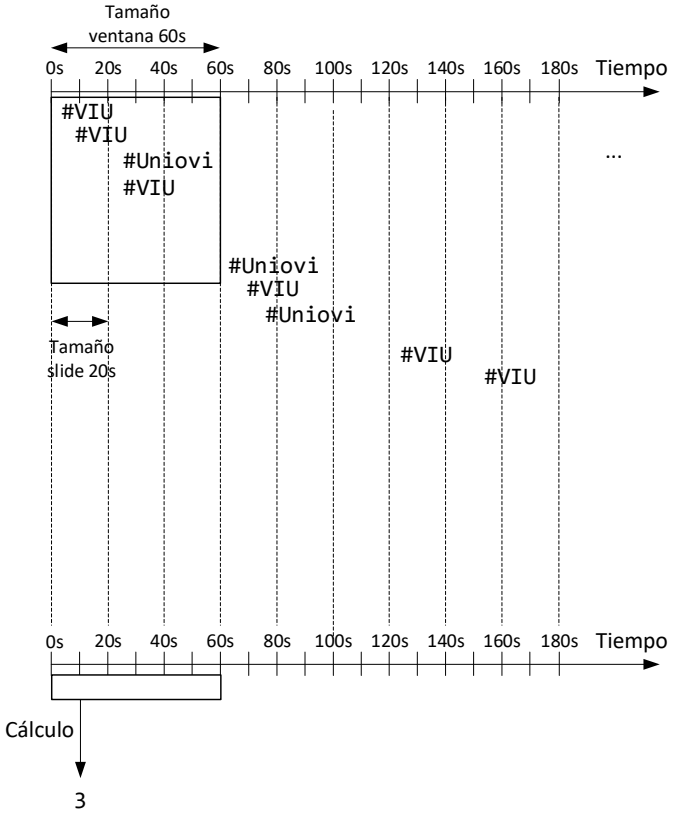
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



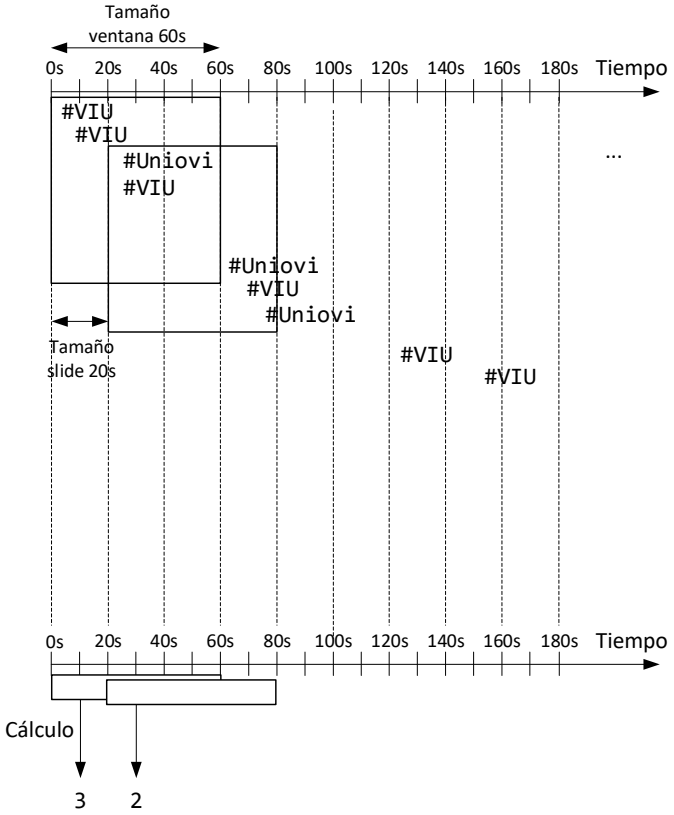
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



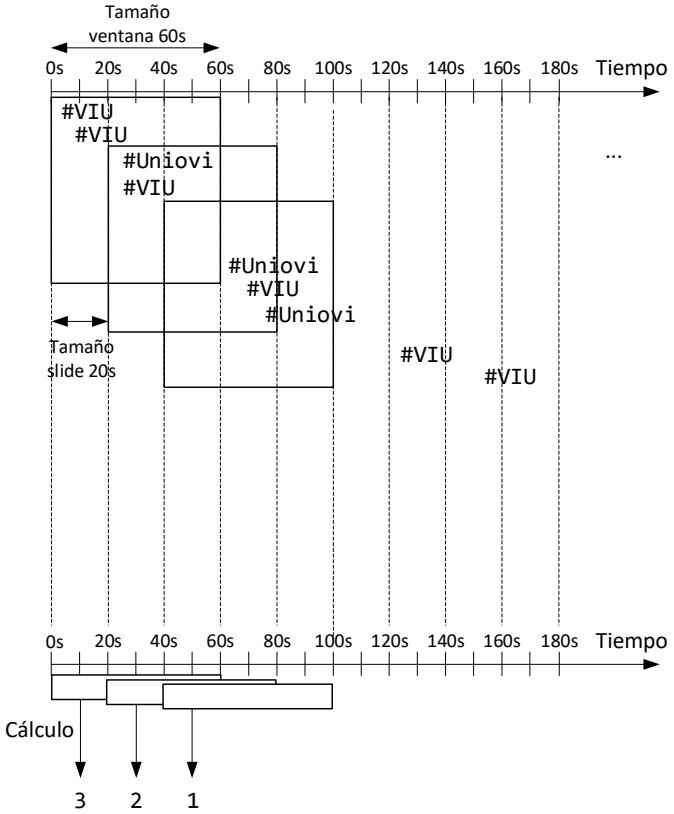
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



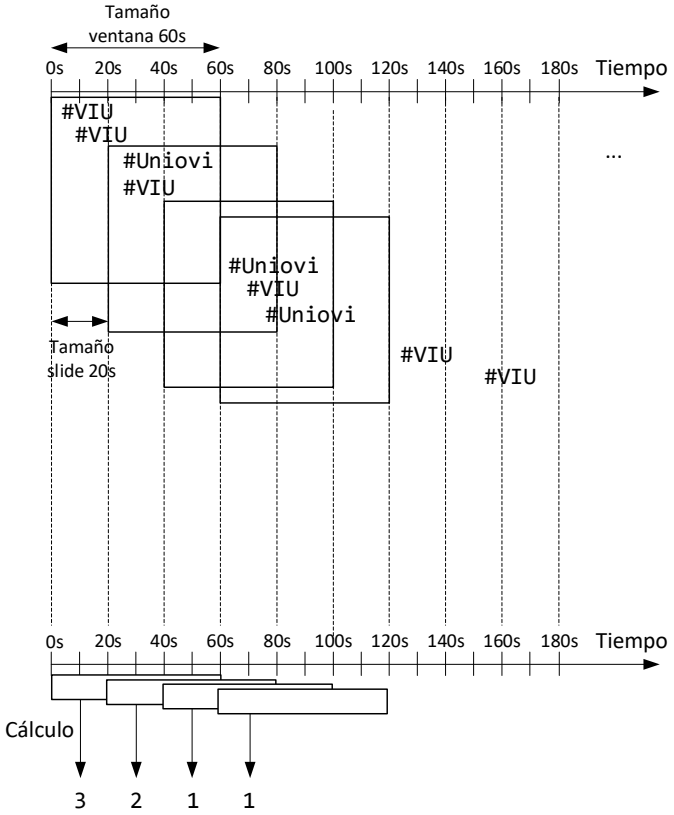
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



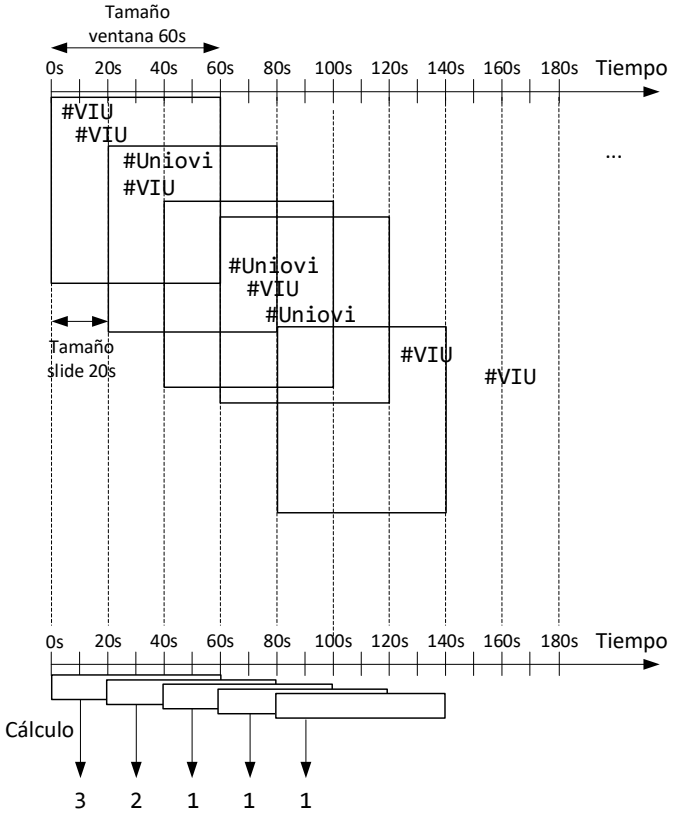
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



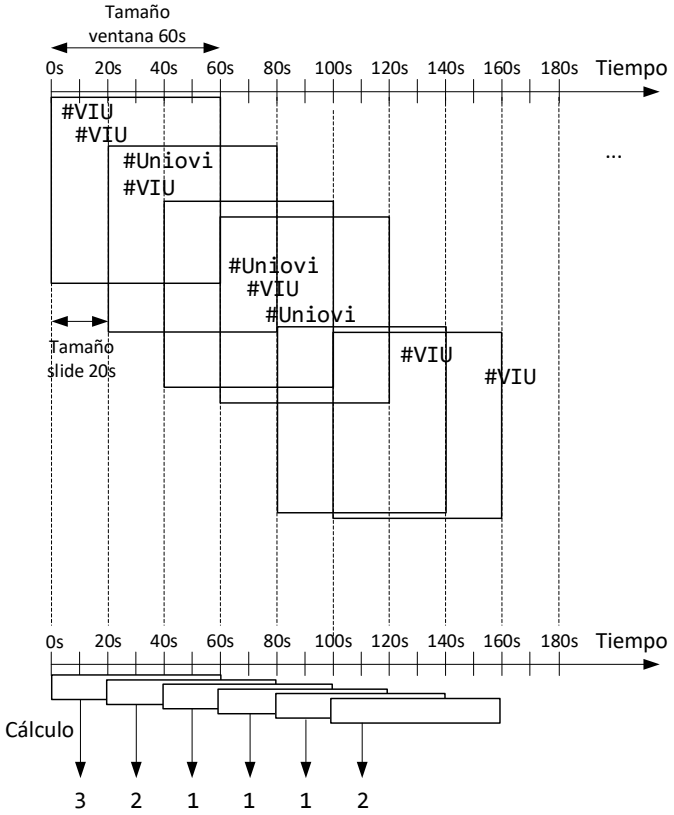
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



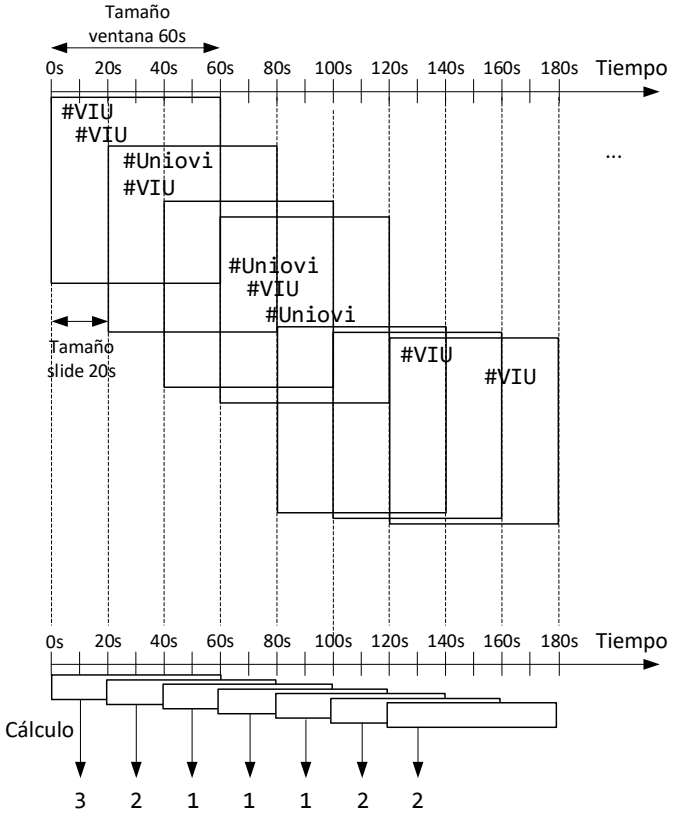
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



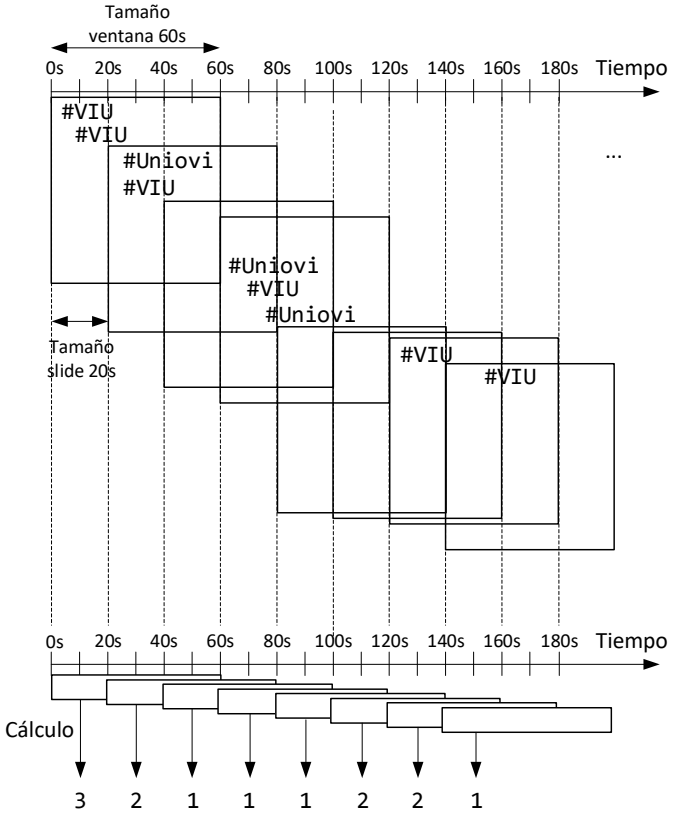
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



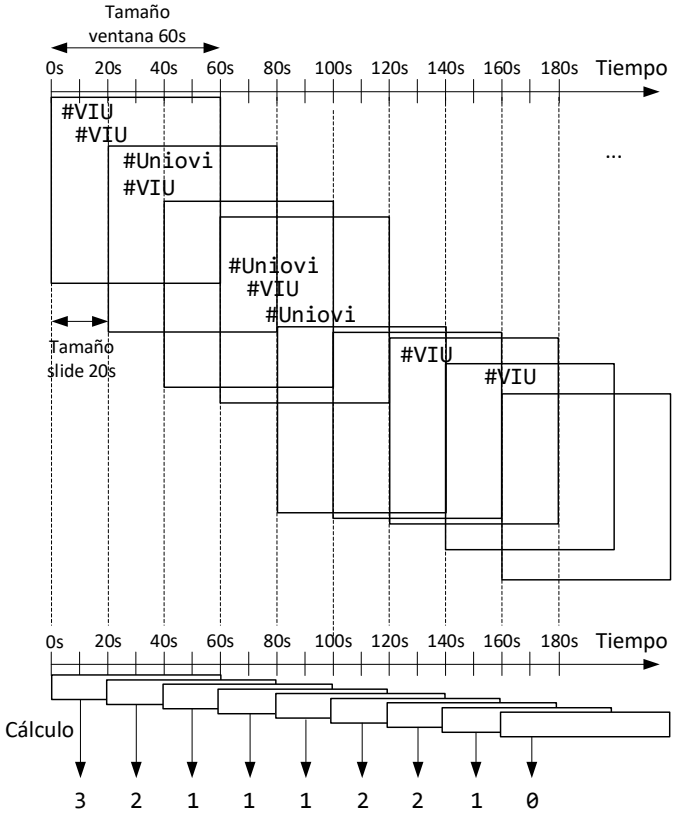
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



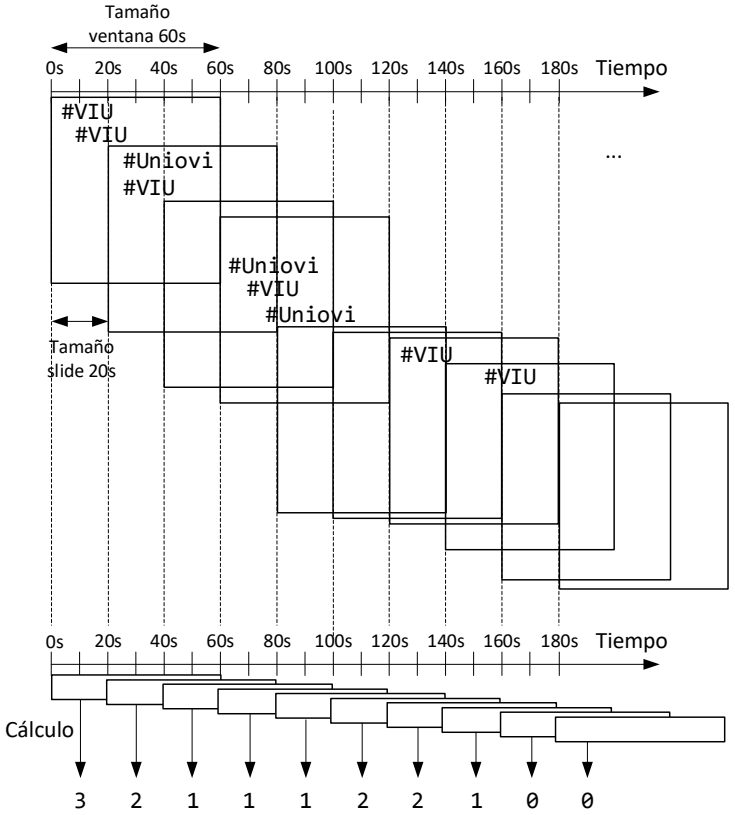
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



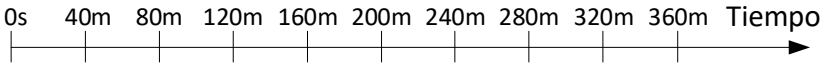
- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s



- **Fixed Window**
- **Slide Window**
  - Se fija un tamaño de ventana
  - La ventana se desplaza continuamente cada cierto tiempo
  - Ejemplo: Cada 20 segundos obtener el número de tweets de la VIU del último minuto -> Ventana = 60s, y Slide = 20s

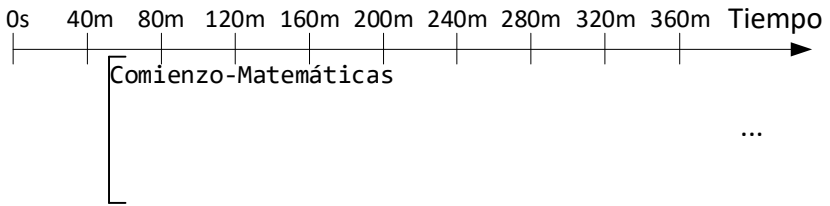


- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase

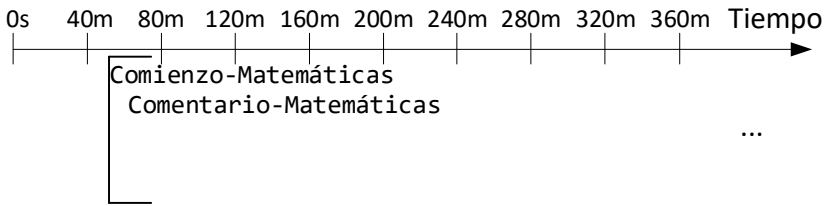


...

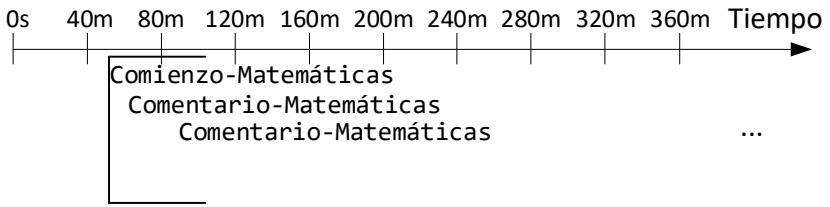
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



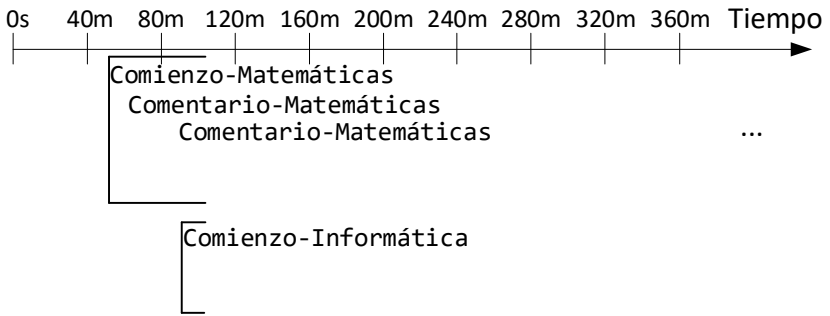
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



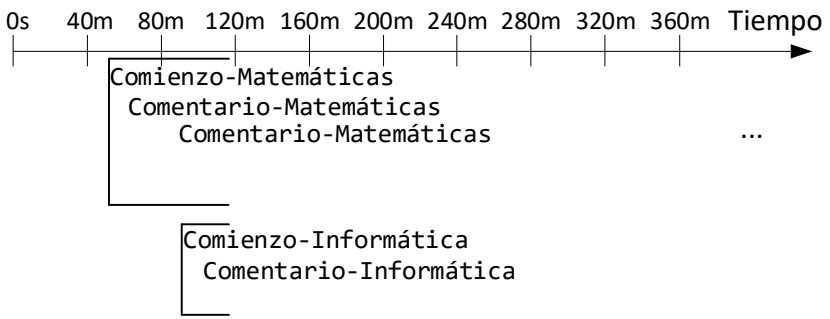
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



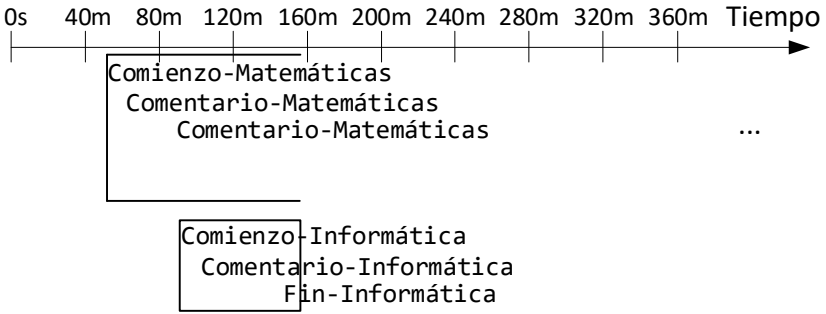
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



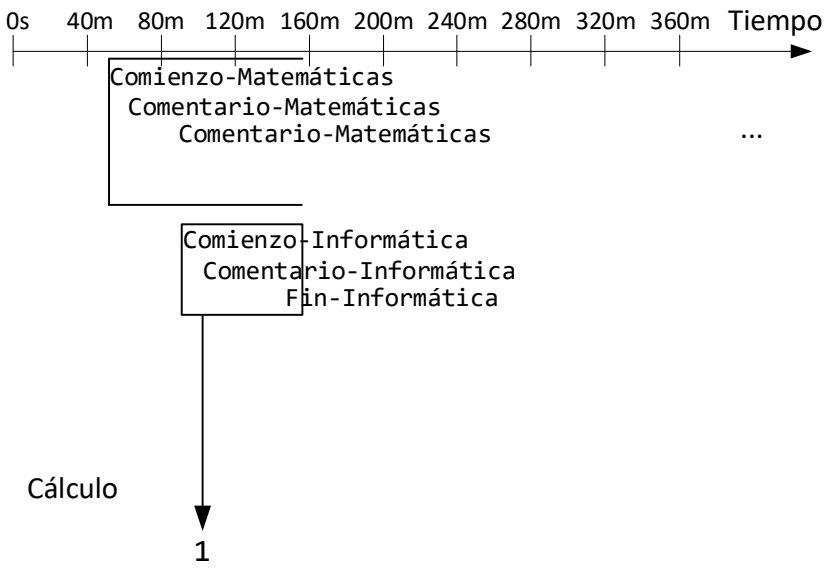
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



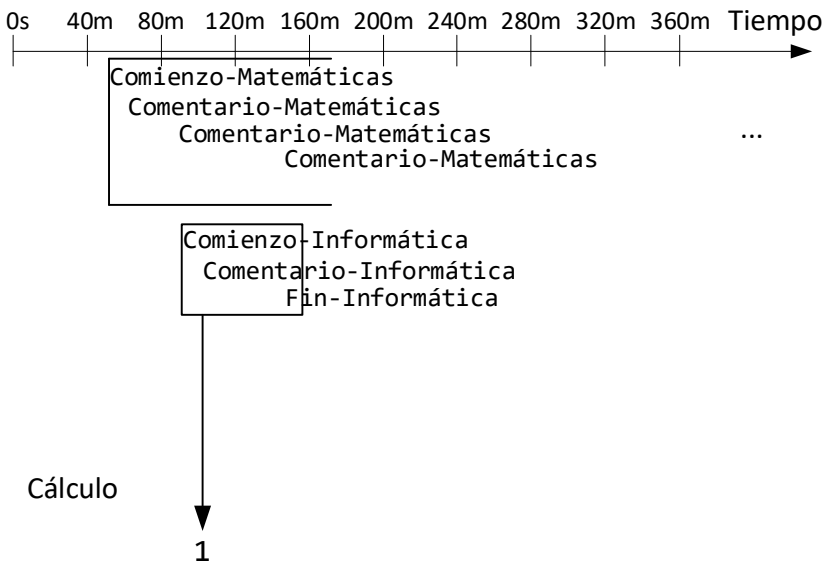
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



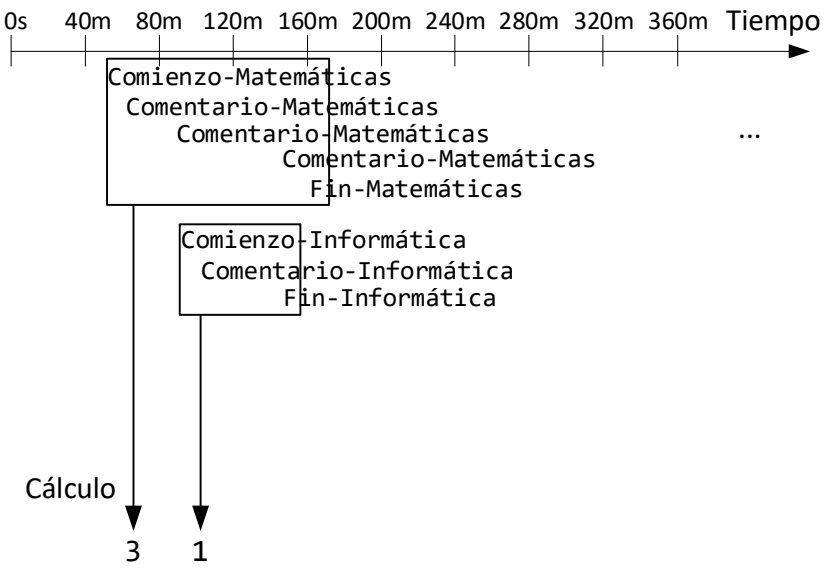
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



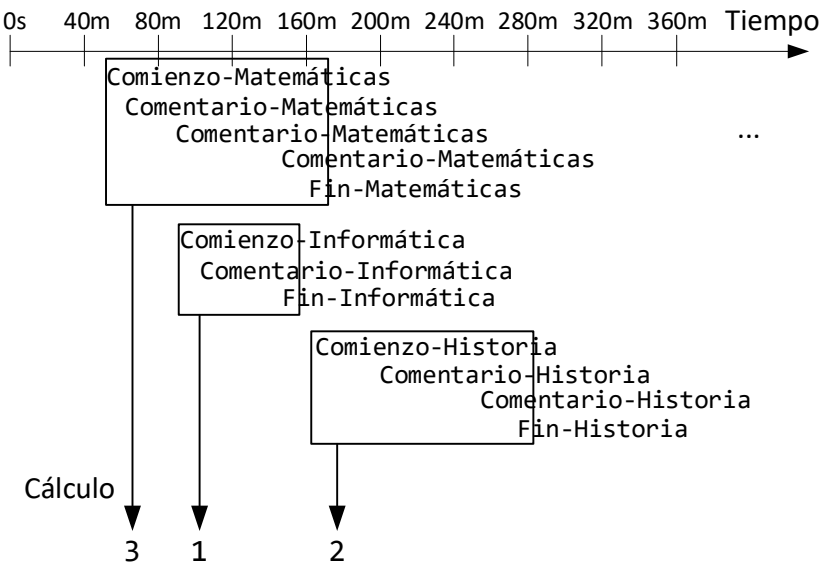
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



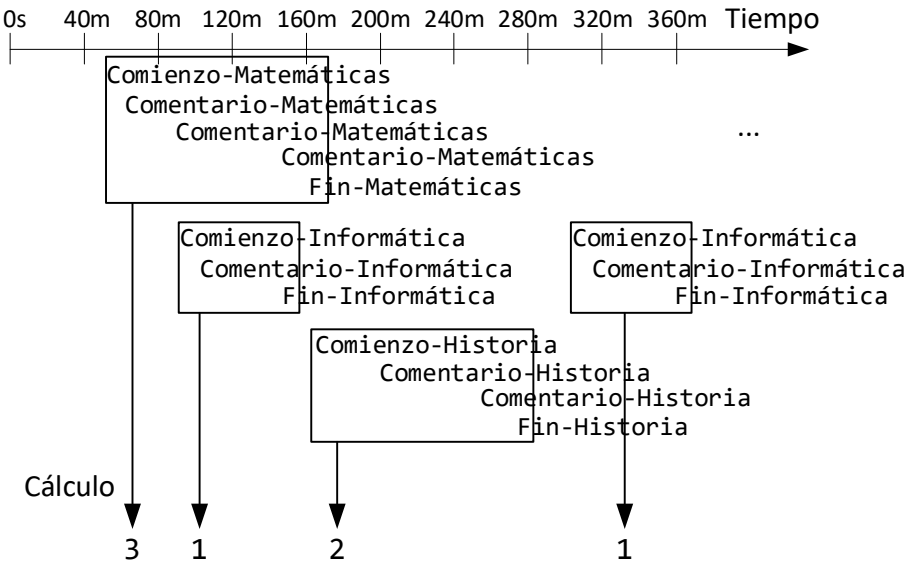
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



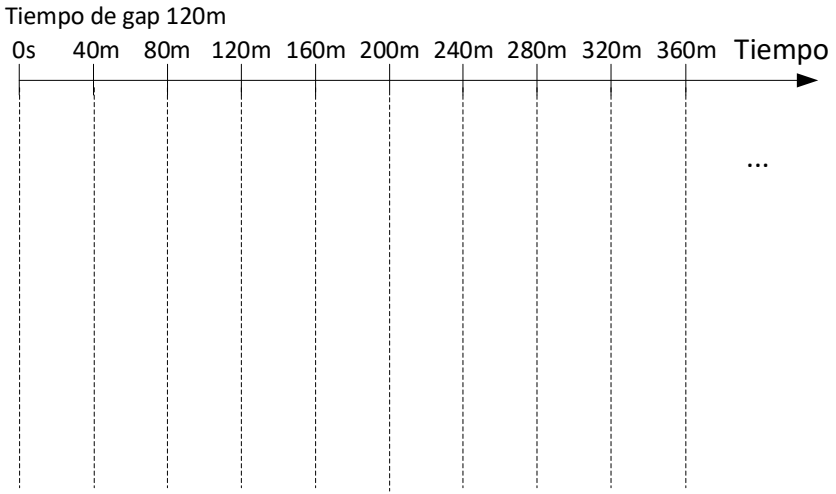
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



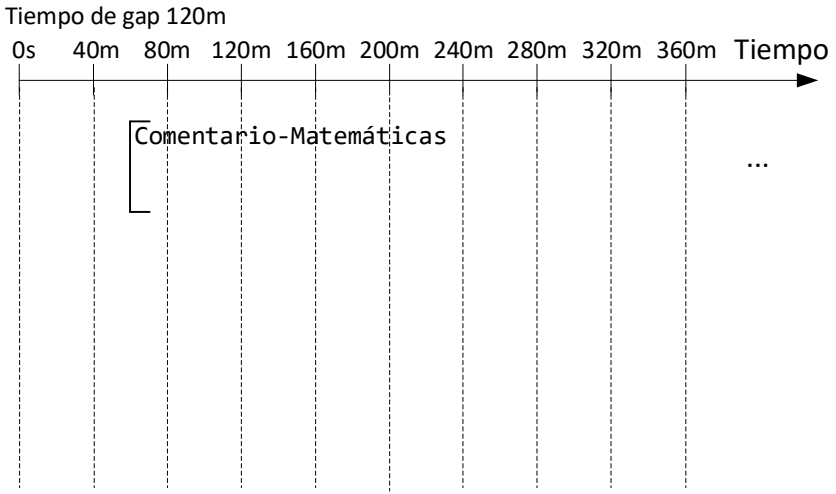
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
  - Ej. Obtener el número de comentarios por clase -> Un dato indica el inicio de clase y otro el fin de clase



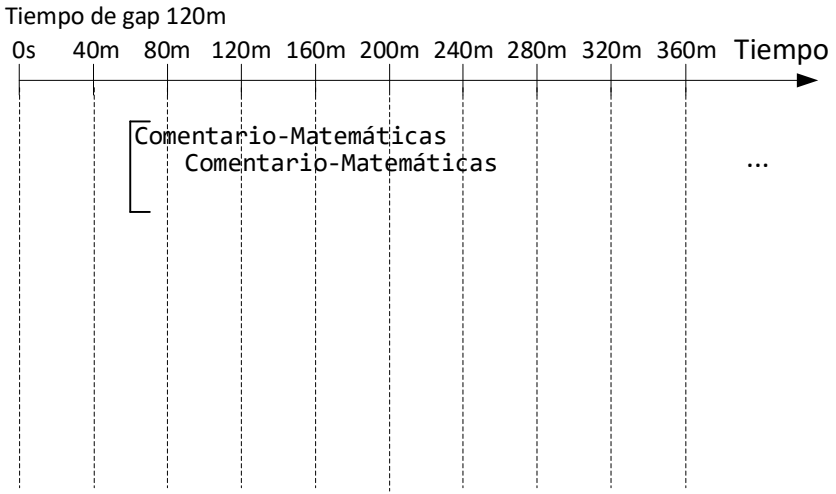
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



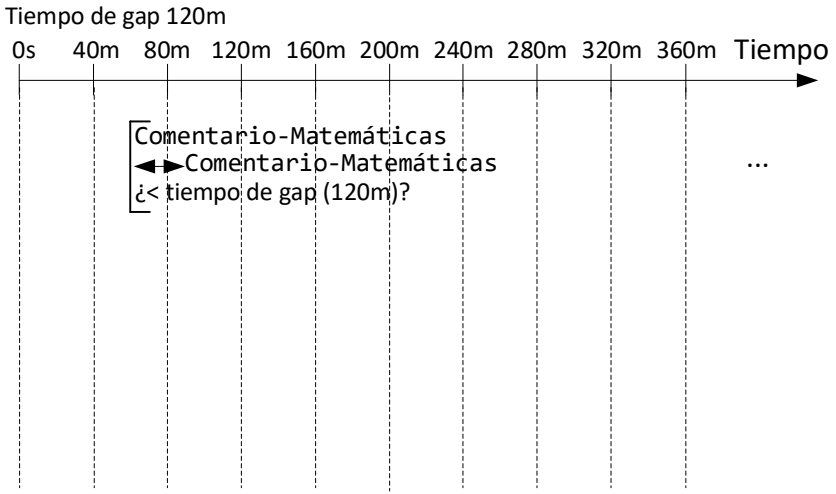
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



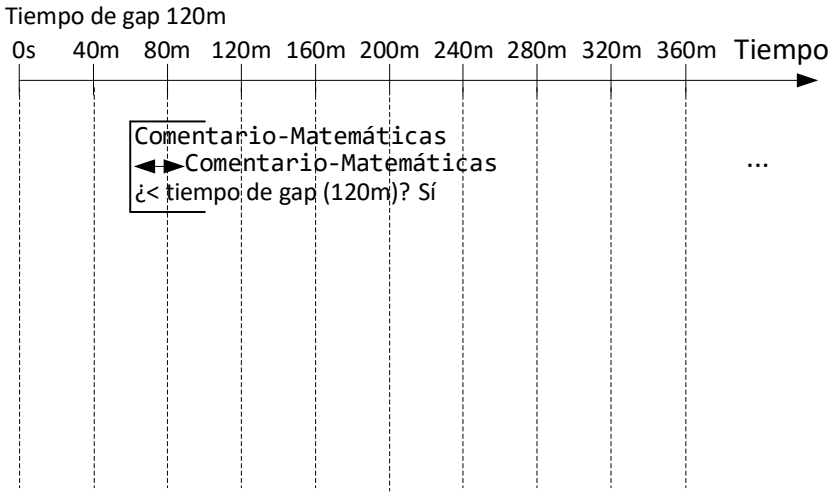
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



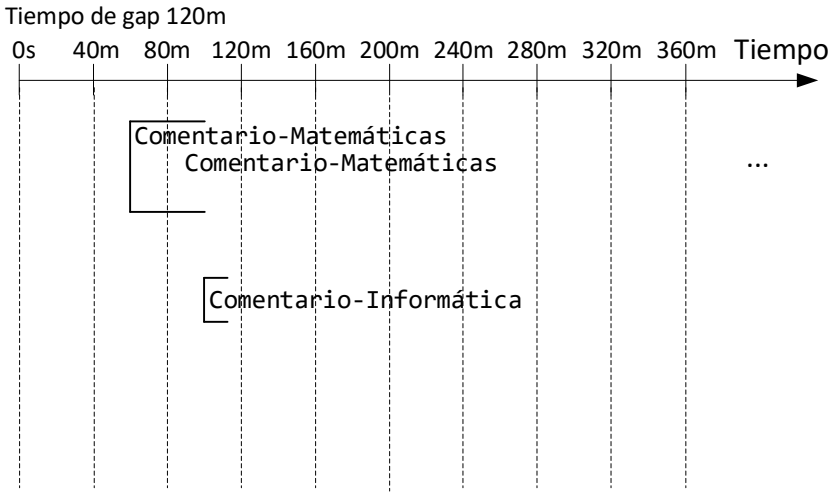
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



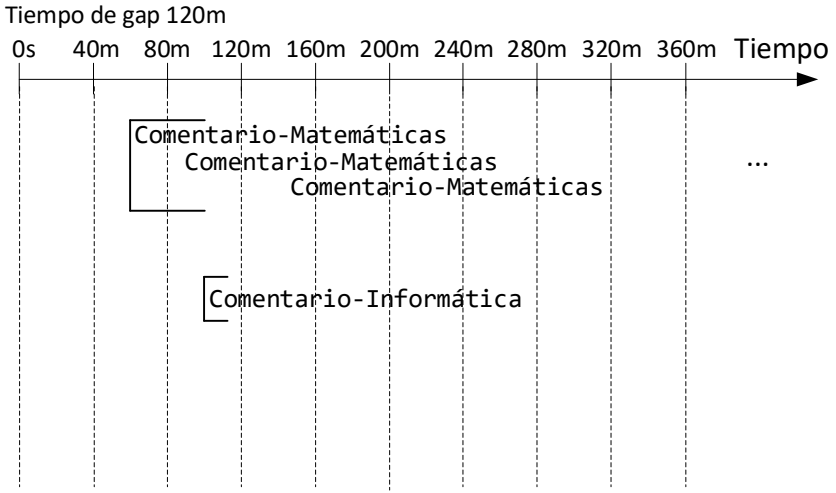
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



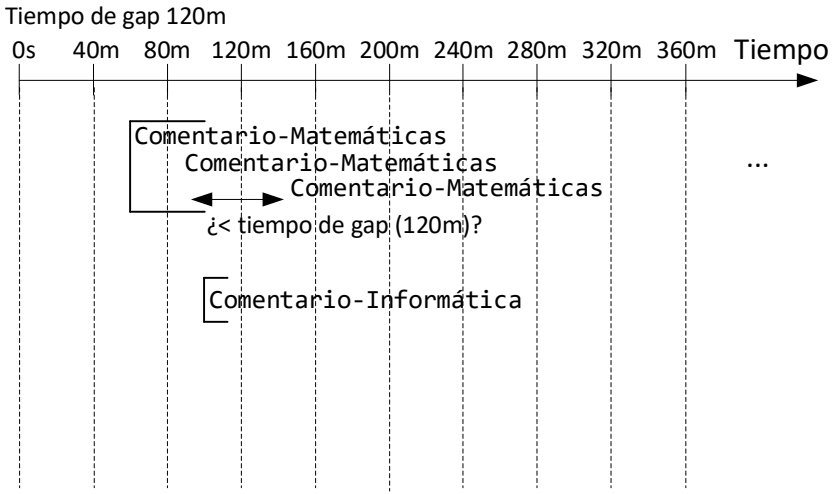
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



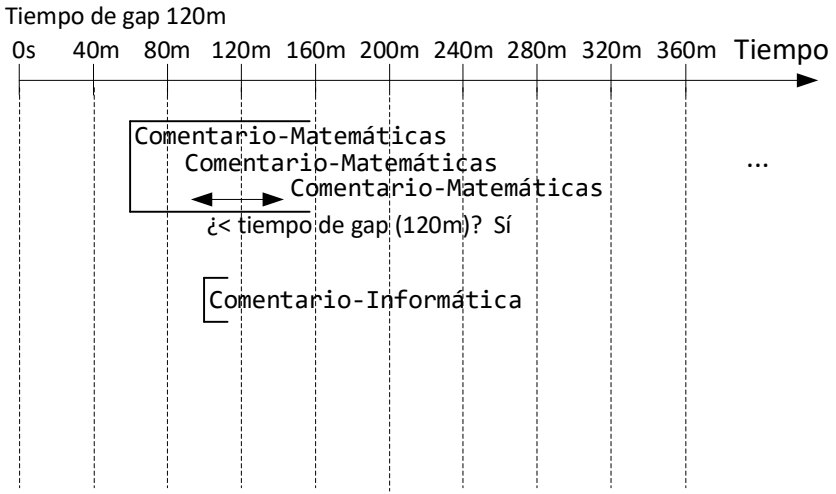
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



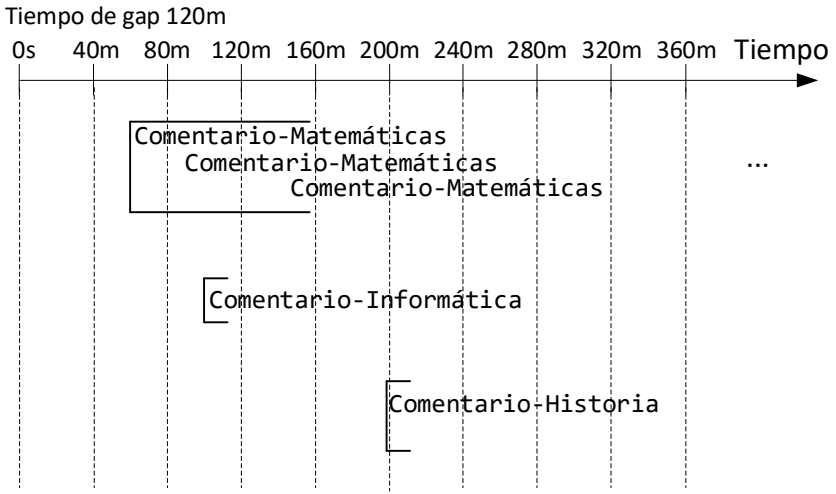
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



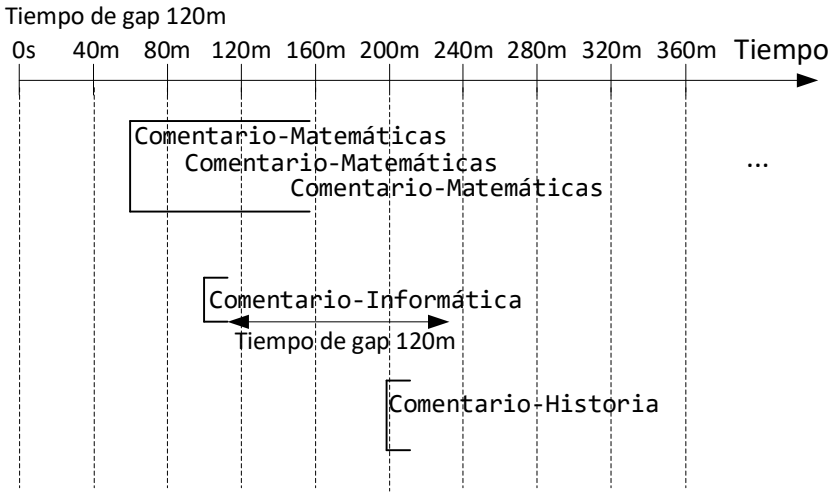
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



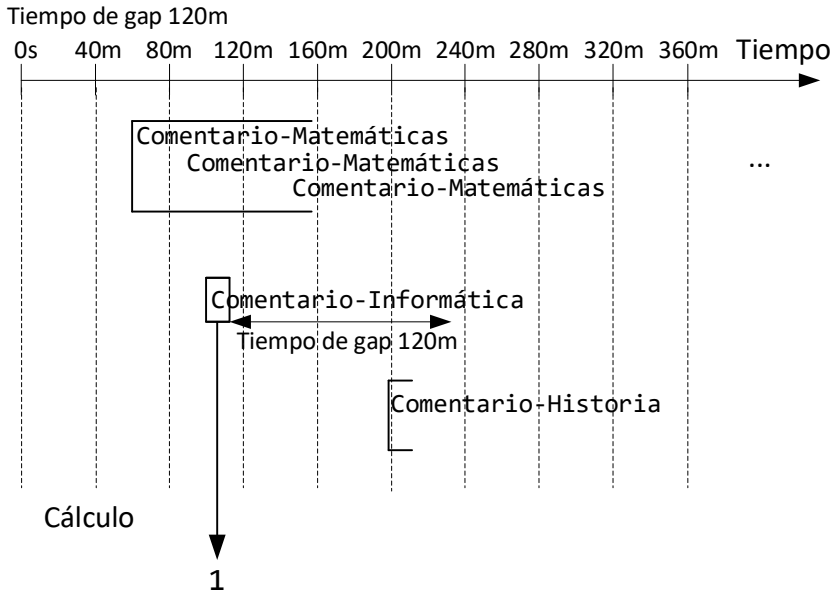
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



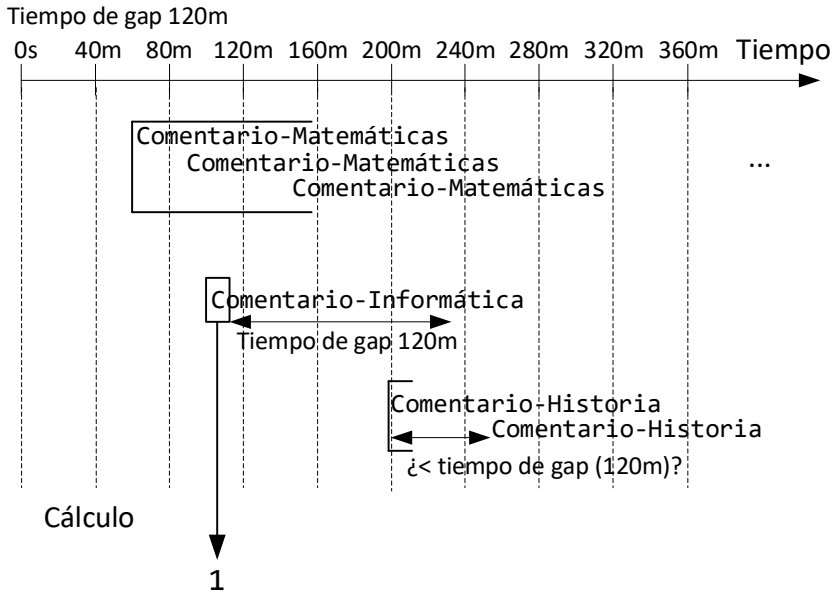
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



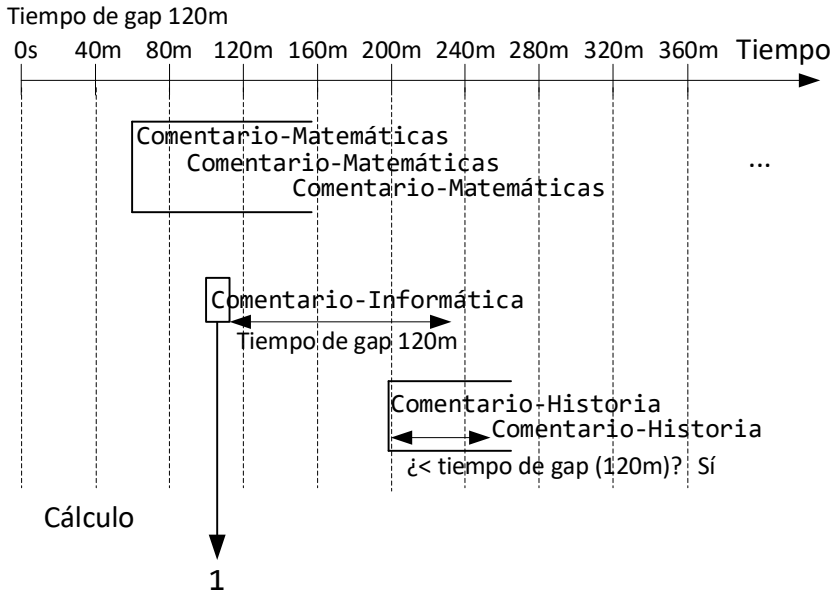
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



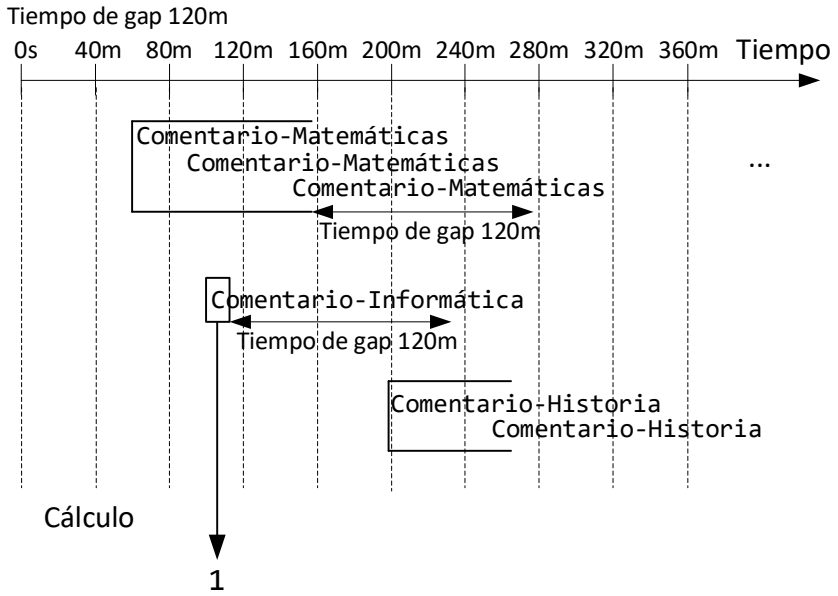
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



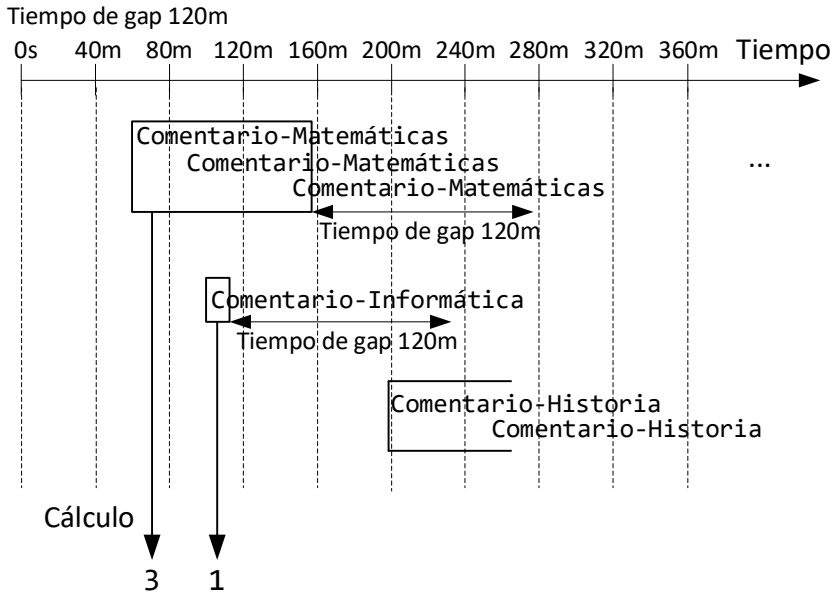
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



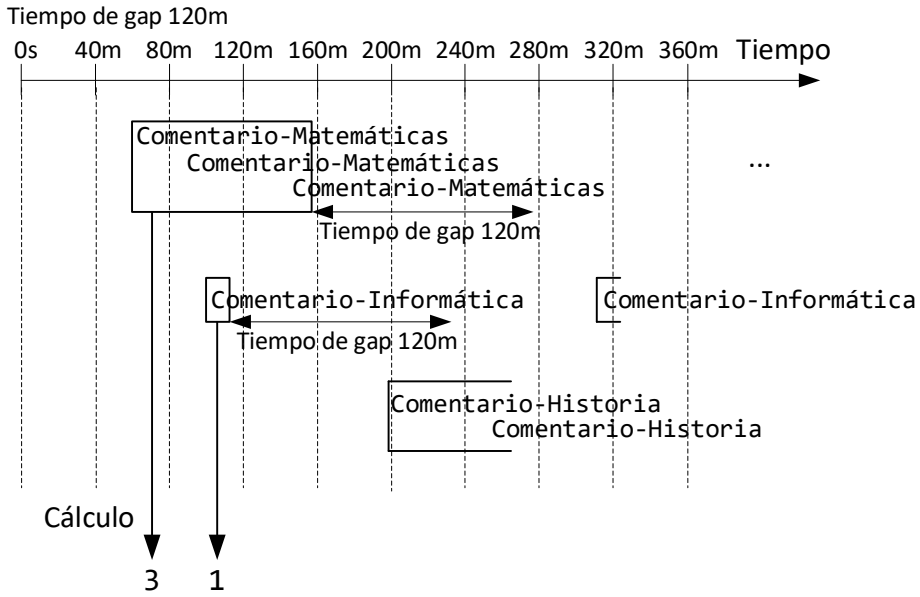
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



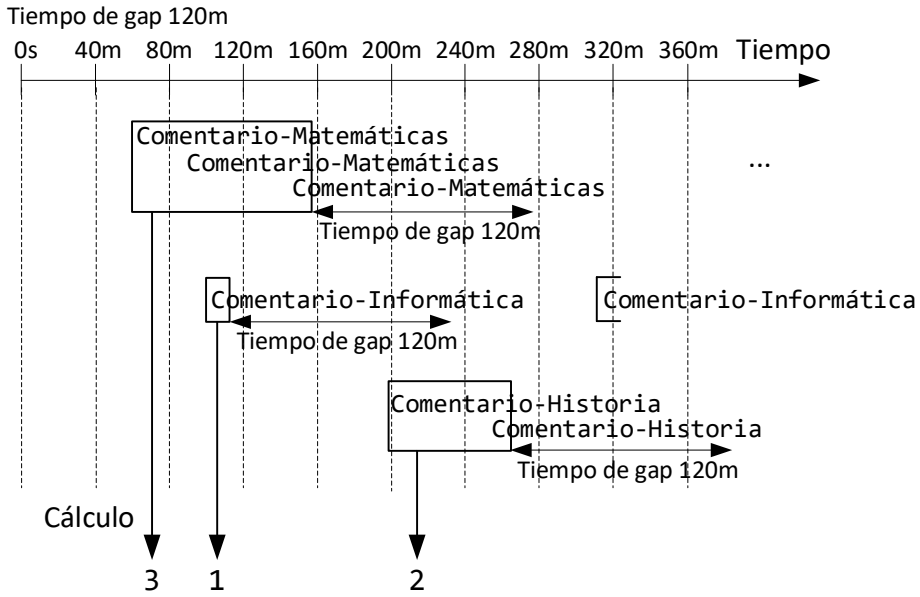
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



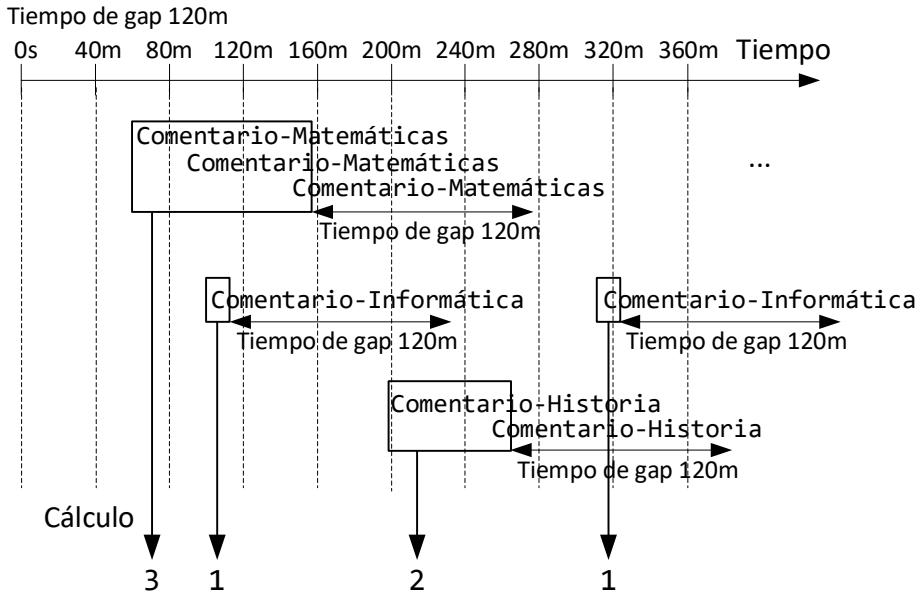
- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase

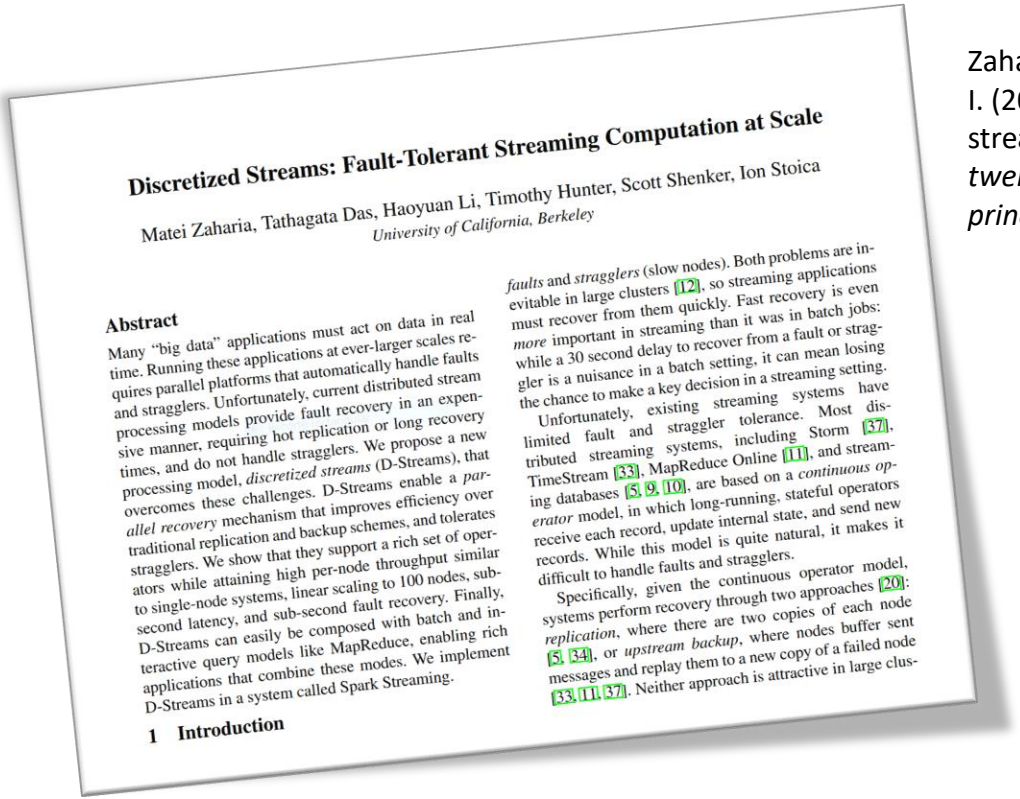


- **Fixed Window**
- **Slide Window**
- **Session widow**
  - Se pueden tener múltiples sesiones
  - Dirigidas por datos: las inicia un dato
  - Puede finalizarla:
    - Un dato
    - Un gap de tiempo sin recibir datos
  - Ej. Obtener el número de comentarios por clase -> Las clases duran menos de 120m, por lo que no deberían existir más de 120m entre comentarios de la misma clase



	Modelo	Latencia	Semántica	Windowing	Estado
Storm	Streaming	Muy baja	At-most-once At-least-once Exactly-once (con Trident)	Fixed Windows Sliding Windows	Stateless Stateful
Samza	Streaming	Muy baja	At-most-once At-least-once Exactly-once (en el futuro)	Fixed Windows Sliding Windows	Stateless Stateful
Spark Streaming	Micro-Batch	Baja	At-most-once At-least-once Exactly-once (con Structured Streaming)	Fixed Windows Sliding Windows Session Windows (con Structured Streaming)	Stateless Stateful
Flink	Streaming	Muy baja	At-most-once At-least-once Exactly-once	Fixed Windows Sliding Windows Session Windows	Stateless Stateful

Spark Streaming



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 processing model, *discretized streams* (D-Streams), that overcomes these challenges. D-Streams enable a *parallel recovery* mechanism that improves efficiency over traditional replication and backup schemes, and tolerates stragglers. We show that they support a rich set of operators while attaining high per-node throughput similar to single-node systems, linear scaling to 100 nodes, sub-second latency, and sub-second fault recovery. Finally, D-Streams can easily be composed with batch and interactive query models like MapReduce, enabling rich applications that combine these modes. We implement D-Streams in a system called Spark Streaming.

1 Introduction

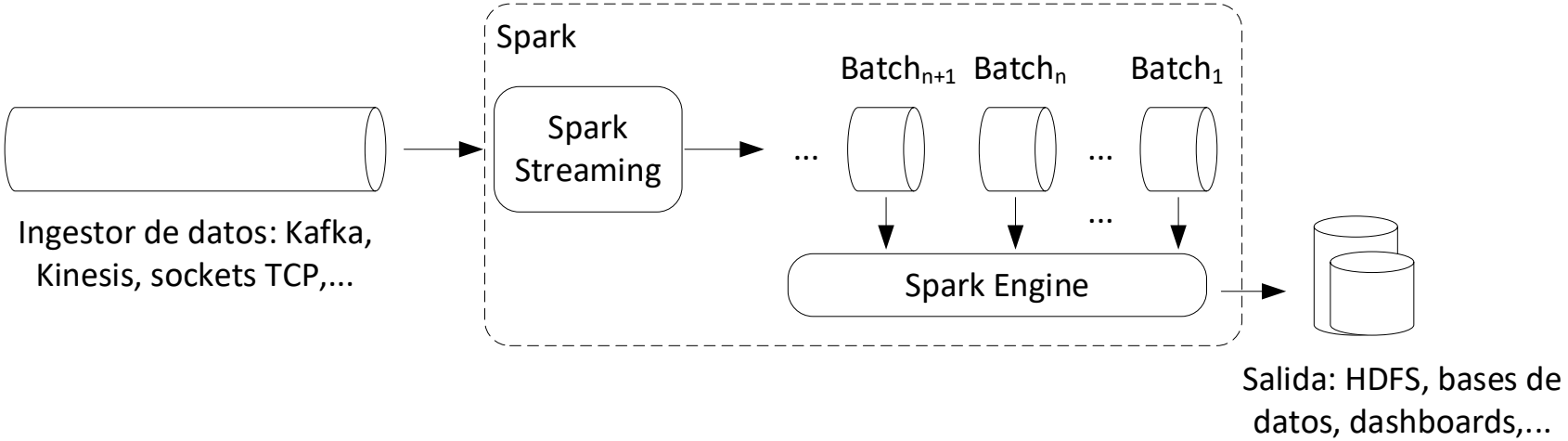
faults and *stragglers* (slow nodes). Both problems are inevitable in large clusters [12], so streaming applications must recover from them quickly. Fast recovery is even *more* important in streaming than it was in batch jobs: while a 30 second delay to recover from a fault or straggler is a nuisance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

Unfortunately, existing streaming systems have limited fault and straggler tolerance. Most distributed streaming systems, including Storm [37], TimeStream [33], MapReduce Online [11], and streaming databases [5, 9, 10], are based on a *continuous operator* model, in which long-running, stateful operators receive each record, update internal state, and send new records. While this model is quite natural, it makes it difficult to handle faults and stragglers.

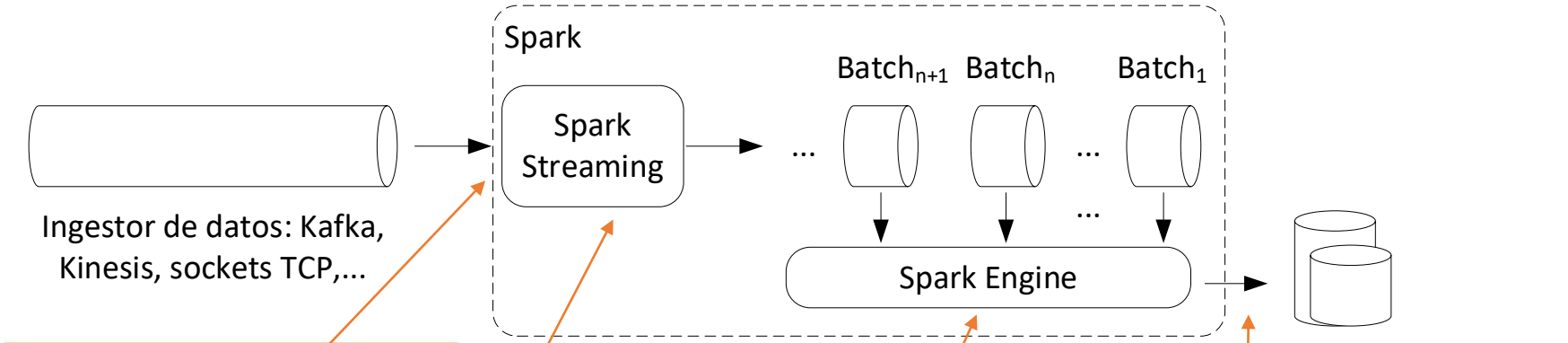
Specifically, given the continuous operator model, systems perform recovery through two approaches [20]: *replication*, where there are two copies of each node [5, 34], or *upstream backup*, where nodes buffer sent messages and replay them to a new copy of a failed node [33, 11, 37]. Neither approach is attractive in large clus-

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013, November). Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles* (pp. 423-438).

- Plataforma para procesamiento streaming
- Escalable
- Popular
- Tolerante a fallos
- Ecosistema Spark



- Spark Streaming: cada cierto tiempo se genera un Batch de datos
- Spark: Procesa ese Batch



Receiver: se encarga de la recepción de los datos

Spark Streaming: se encarga de generar y orquestar los Batchs

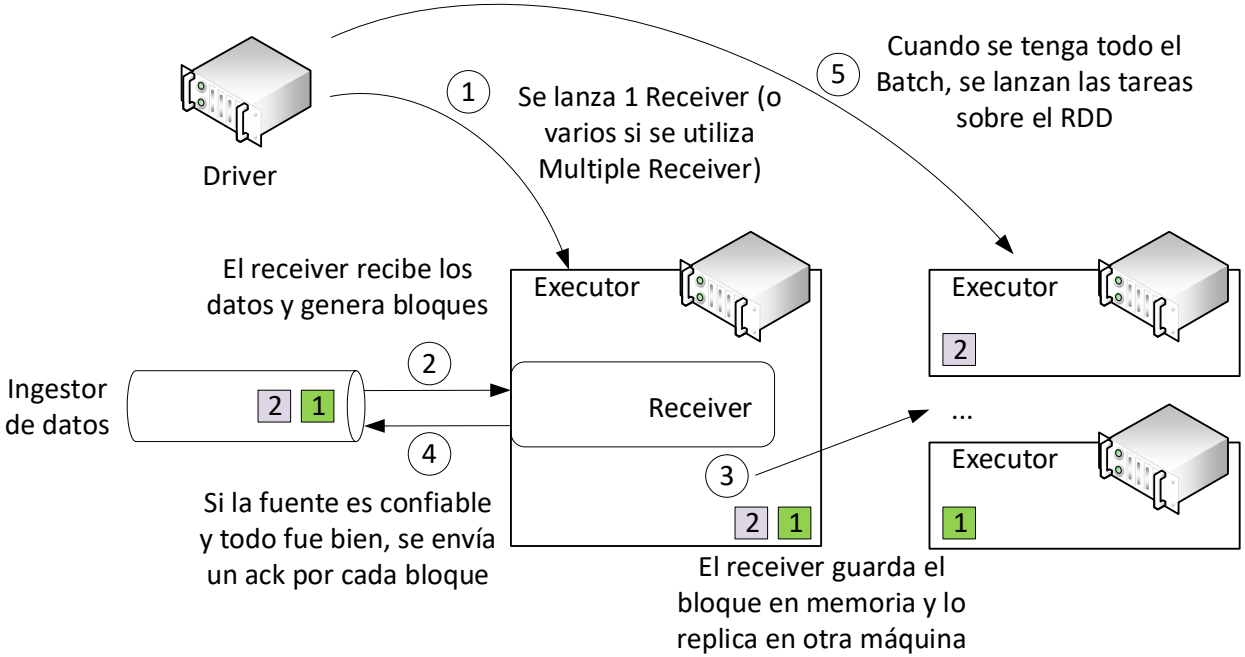
Spark Enginer: se encarga de ejecutar los Batchs

Salida: archivos, Kafka o Kinesis producer,...

Receivers

- Proceso que se ejecuta en un esclavo de forma indefinida (salvo que indiquemos que finalice)
- Permite consumir los datos que llegan desde un ingestor de datos
- Genera bloques con los datos que recibe (particionado)
- Guarda los bloques en memoria y los replica en otro ordenador (por defecto)
- Tipos de Receiver:
  - Confiables: Envían un ack al ingestor si fue capaz de replicar correctamente el dato
  - No confiables: No envían ack

Receivers



- Receivers implementados:
  - Kafka
  - Kinesis
  - Leer desde archivos
  - Sockets TCP
  - …
- Para implementar nuevos receivers: implementar clase abstracta `Receiver`

- Representa todo el flujo de datos
- Cada cierto tiempo:
  - Se genera un nuevo Batch
  - Se procesa el anterior Batch
- Cada Batch es un RDD
  - Puede tener varias particiones
  - Puede estar vacío `EmptyRDD`
- Un Dstream es una colección de RDDs con los datos que van llegando continuamente

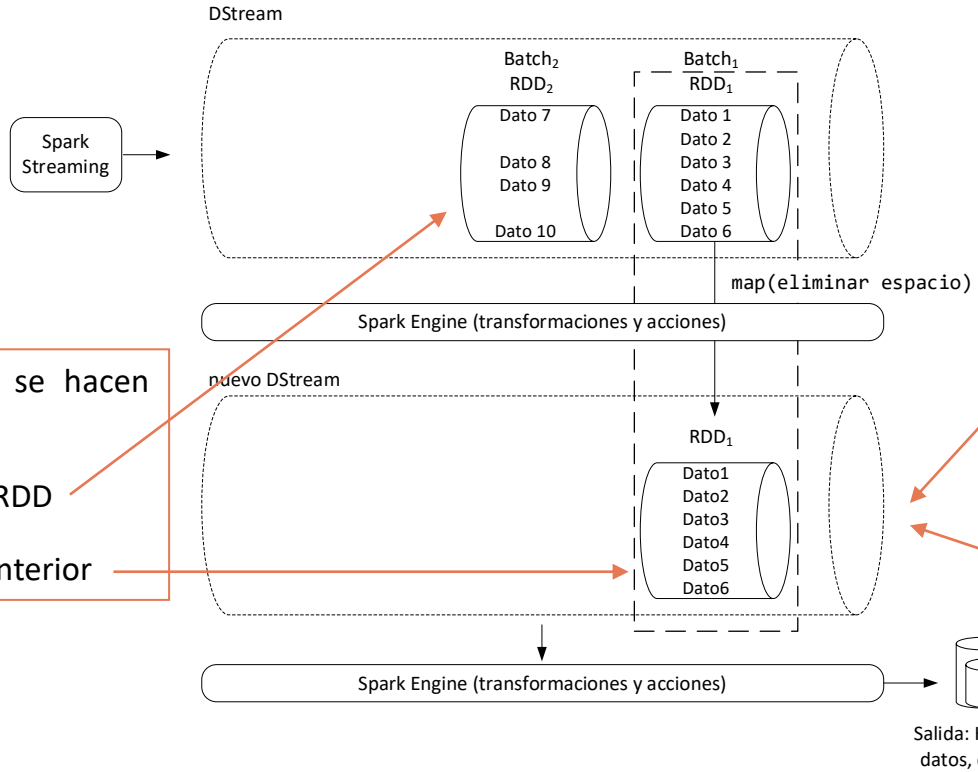
t = 1

Tras un tiempo se recibe un Batch de datos -> RDD

- El RDD está particionado (bloques)
- Cada partición está en memoria del Receiver y en otro ordenador (por defecto)

DStream

t = 2



En el segundo instante se hacen dos cosas:

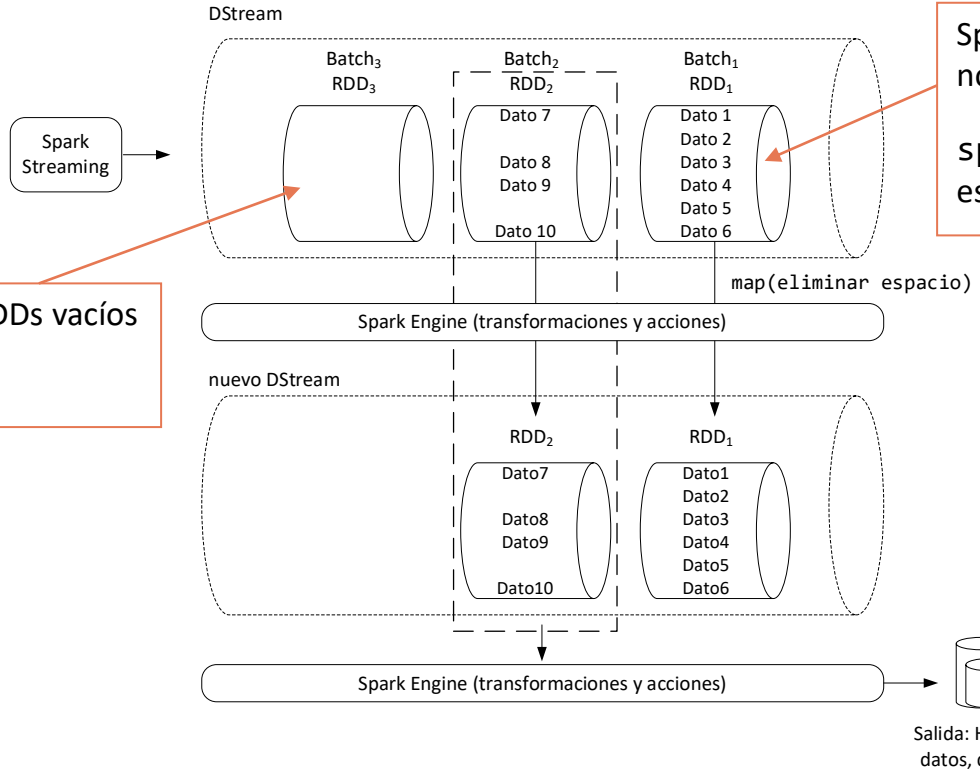
- Se genera el nuevo RDD
- Se procesa el RDD anterior

Se van generando nuevos Dstreams con las transformaciones

Tiene que finalizar el procesamiento del RDD1 antes de que llegue el siguiente instante de tiempo

DStream

t = 3



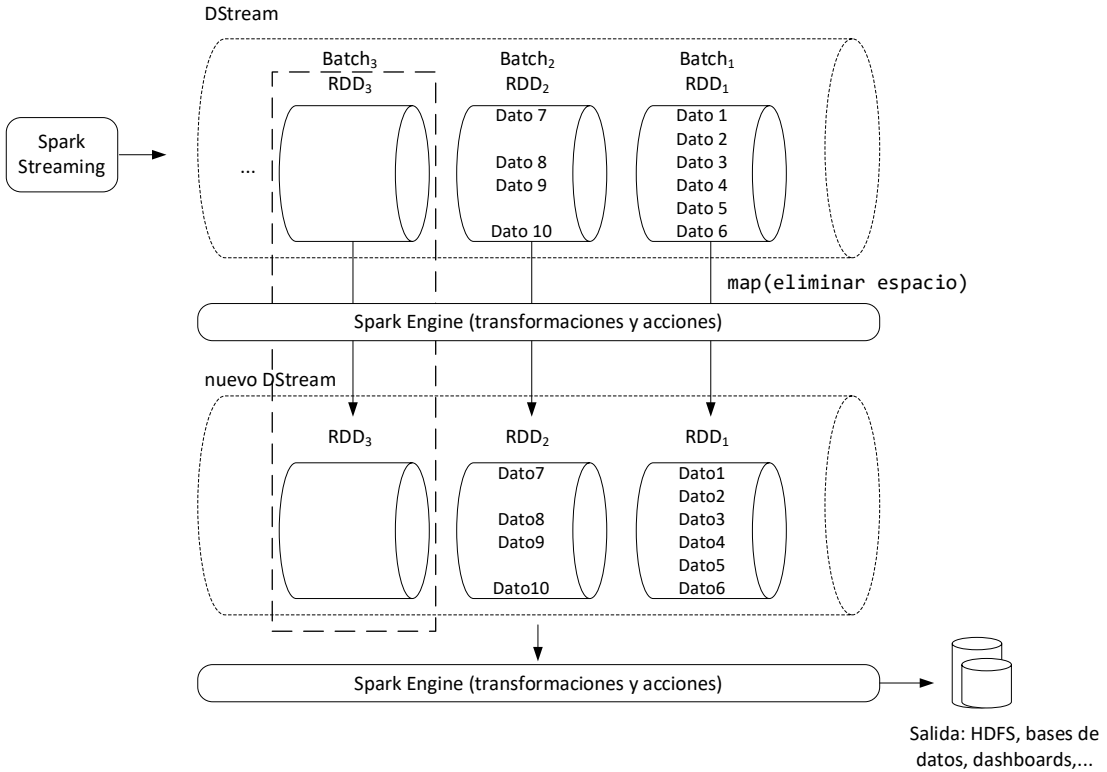
Spark despersiste los datos que no necesita debido a que:

`spark.streaming.unpersist` es `True`

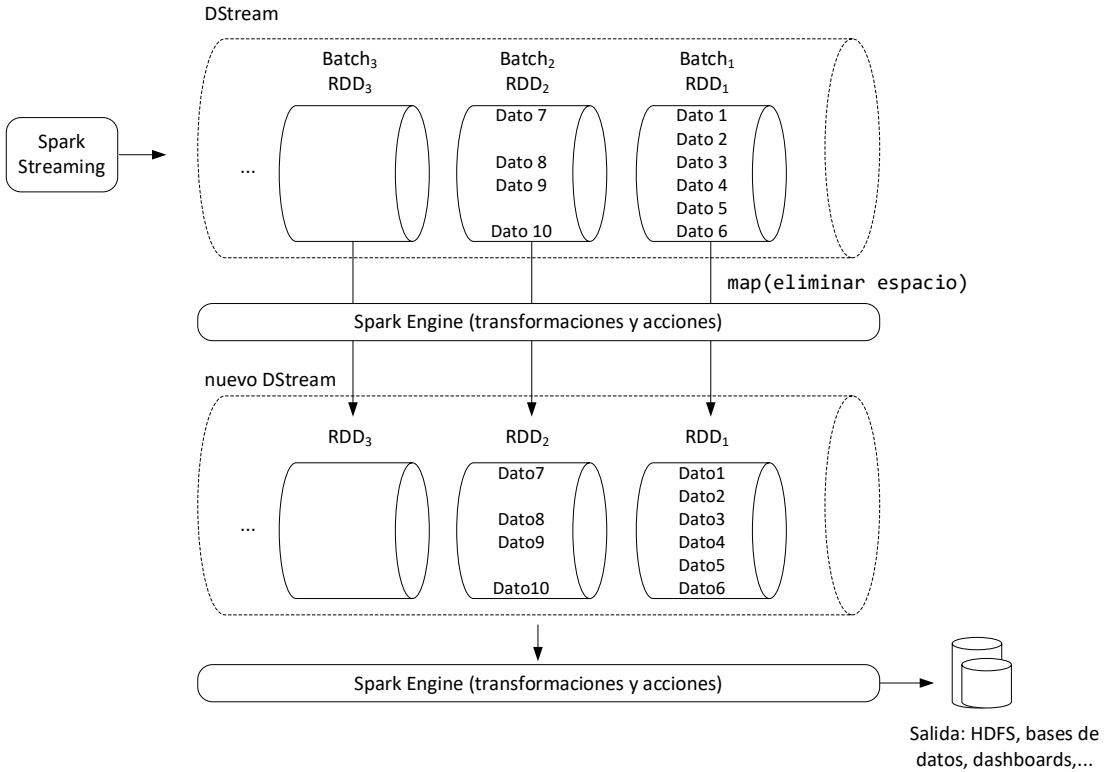
Pueden generarse RDDs vacíos

`EmtpyRDD`

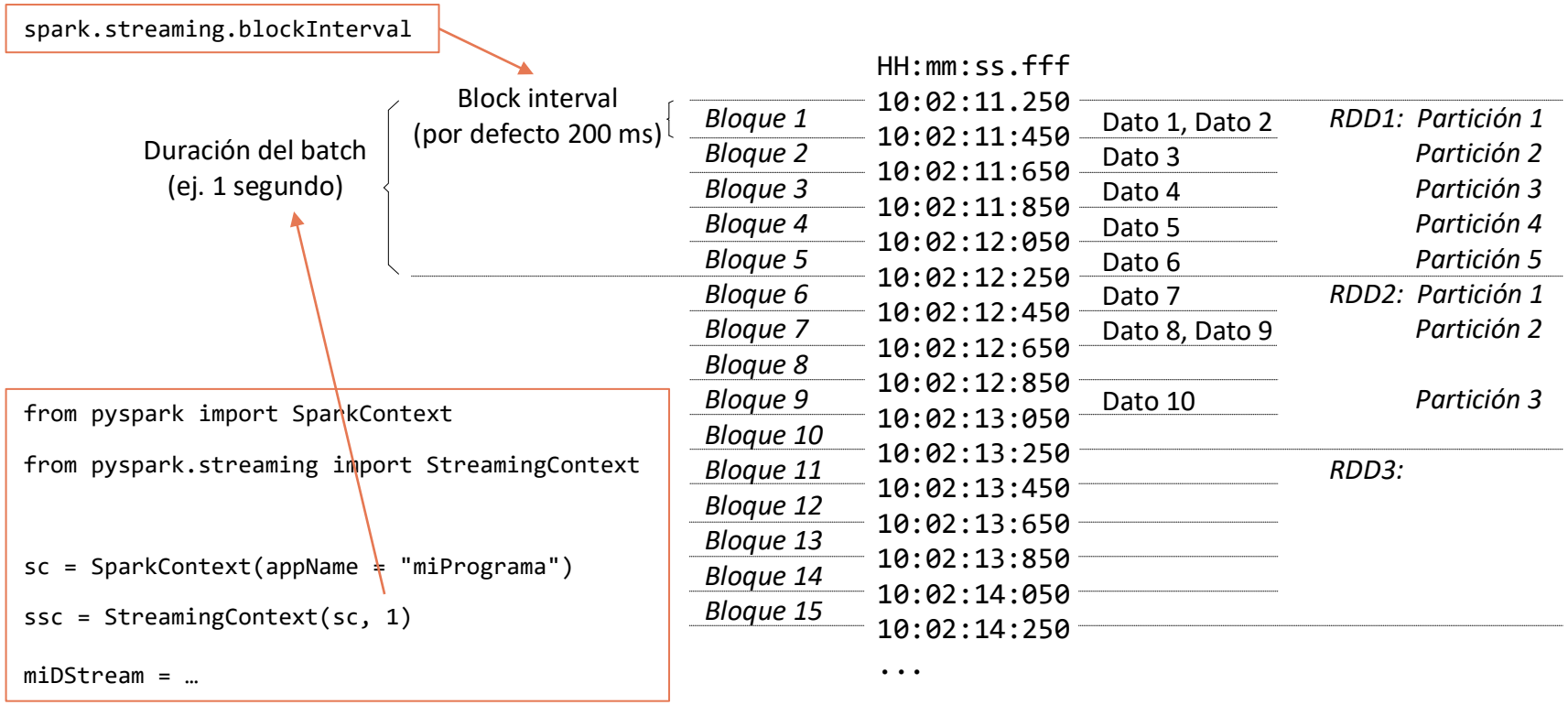
t = 4



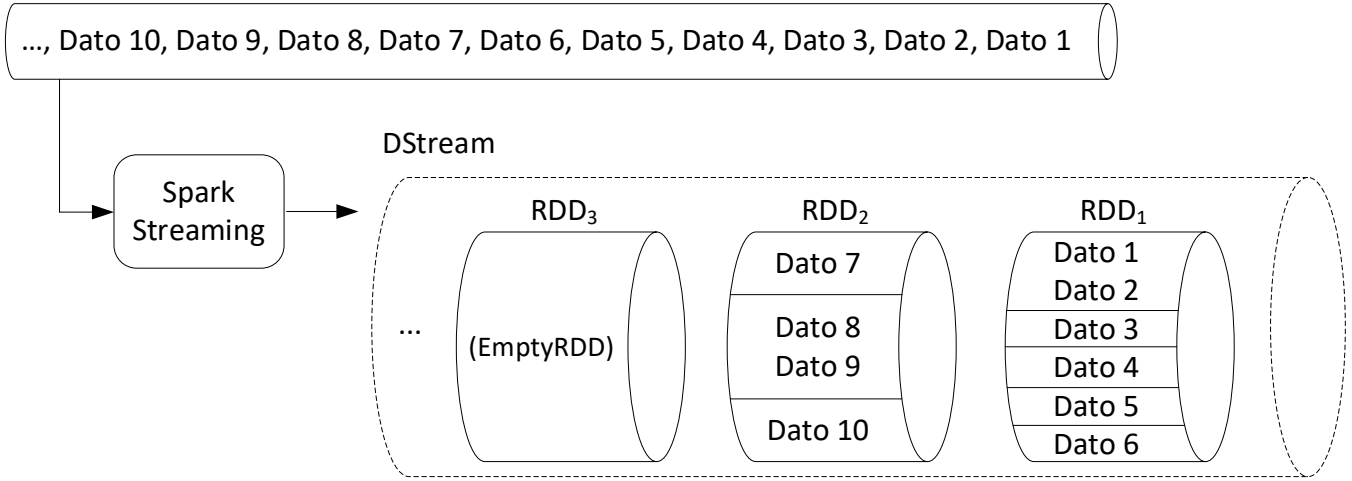
t = 5



Dstream: RDD y Particiones



Dstream: RDD y Particiones



- Bloques/Particiones:
  - Receiver-based approach:
    - Una partición cada `spark.streaming.blockInterval` segundos
  - Receiver-less approach:
    - Los ingestores de datos pueden gestionar diferentes particionados
    - Kafka direct: cada partición es una partición Kafka
- Multiple Receivers:
  - Generar varios receivers (varios Dstreams) y luego unirlos
  - Ej. un receiver por cada topic de Kafka

```
from pyspark import SparkContext

from pyspark.streaming import StreamingContext

sc = SparkContext(appName = "miPrograma")

ssc = StreamingContext(sc, 1)

…

ssc.start()
```

- Estamos indicando que se genere un Batch cada 1 segundo
- Otras opciones:
  - `ssc.awaitTermination()` Espera hasta que terminen las computaciones
  - `ssc.awaitTerminationOrTimeout(X)`: Espera a que se terminen las computaciones o X segundos
  - `ssc.stop(stopSparkContext=True, stopGraceFully=True)`

- Se pueden leer múltiples fuentes:
  - Kakfa:

    miStream = KafkaUtils.createStream(ssc, parámetros de kakfa, topic, etc)

  - Socket:

    ```
    miDStream = ssc.socketTextStream(ip, puerto)
    ```

  - QUEUERDD:
    - Crea un DStream a partir de una colección de datos
    - Ideal para hacer pruebas y depurar el código

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

miPrograma.py

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

if __name__ == "__main__":
  sc = SparkContext(appName="miPrograma")
  sc.setLogLevel("ERROR")
  ssc = StreamingContext(sc, 10)

  inputStream = ssc.socketTextStream("localhost", 9999)

  nuevoDStream = inputStream.map(lambda record: record.upper())
  nuevoDStream.pprint()

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

Cada 10 segundos crea un Batch

Recibe datos de localhost:9999

Transforma cada registro a mayúsculas

Imprime los datos

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`

miPrograma.py

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

if __name__ == "__main__":
  sc = SparkContext(appName="miPrograma")
  sc.setLogLevel("ERROR")
  ssc = StreamingContext(sc, 10)

  inputStream = ssc.socketTextStream("localhost", 9999)

  nuevoDStream = inputStream.map(lambda record: record.upper())
  nuevoDStream.pprint()

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

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`

2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`

miPrograma.py

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

if __name__ == "__main__":
  sc = SparkContext(appName="miPrograma")
  sc.setLogLevel("ERROR")
  ssc = StreamingContext(sc, 10)

  inputStream = ssc.socketTextStream("localhost", 9999)

  nuevoDStream = inputStream.map(lambda record: record.upper())
  nuevoDStream.pprint()

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

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
```

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`
2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`
3. Ejecutamos el programa. Desde otra terminal: `spark-submit miPrograma.py`

miPrograma.py

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

if __name__ == "__main__":
  sc = SparkContext(appName="miPrograma")
  sc.setLogLevel("ERROR")
  ssc = StreamingContext(sc, 10)

  inputStream = ssc.socketTextStream("localhost", 9999)

  nuevoDStream = inputStream.map(lambda record: record.upper())
  nuevoDStream.pprint()

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

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
```

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`
2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`
3. Ejecutamos el programa. Desde otra terminal: `spark-submit miPrograma.py`
4. Introducimos algún dato

miPrograma.py

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

if __name__ == "__main__":
  sc = SparkContext(appName="miPrograma")
  sc.setLogLevel("ERROR")
  ssc = StreamingContext(sc, 10)

  inputStream = ssc.socketTextStream("localhost", 9999)

  nuevoDStream = inputStream.map(lambda record: record.upper())
  nuevoDStream.pprint()

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

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
uno
dos
tres
```

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`
2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`
3. Ejecutamos el programa. Desde otra terminal: `spark-submit miPrograma.py`
4. Introducimos algún dato

Terminal 1

```
-------------------------------------------
Time: 2020-12-12 13:58:00
-------------------------------------------
UNO
DOS
TRES
```

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
uno
dos
tres
```

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`
2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`
3. Ejecutamos el programa. Desde otra terminal: `spark-submit miPrograma.py`

Terminal 1

```
-------------------------------------------
Time: 2020-12-12 13:58:00
-------------------------------------------
UNO
DOS
TRES
```

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
uno
dos
tres
cuatro
cinco
```

4. Introducimos algún dato
5. Después de 10 o más segundos, introducimos más datos

Ejemplo programa

- Se puede ejecutar en la máquina virtual de clase sin instalaciones extra

1. Le damos permisos de ejecución: `chmod u+x miPrograma.py`
2. Creamos un servidor que escuche en localhost:9999, para ello desde otra terminal: `netcat -lk -s localhost -p 9999`
3. Ejecutamos el programa. Desde otra terminal: `spark-submit miPrograma.py`

Terminal 1

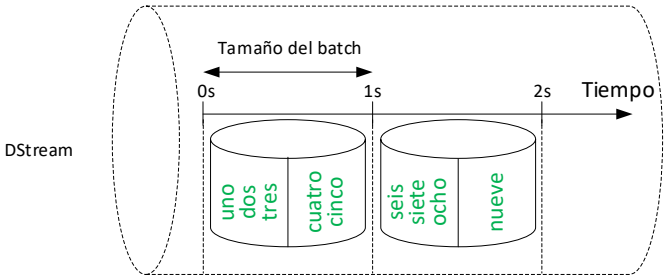
```
-------------------------------------------
Time: 2020-12-12 13:58:00
-------------------------------------------
UNO
DOS
TRES

-------------------------------------------
Time: 2020-12-12 13:58:10
-------------------------------------------
CUATRO
CINCO
```

Terminal 2

```
jesusmoran:.../jesus$ netcat -lk -s localhost -p 9999
uno
dos
tres
cuatro
cinco
```

4. Introducimos algún dato
5. Después de 10 o más segundos, introducimos más datos

Ejemplo QUEUERDD:

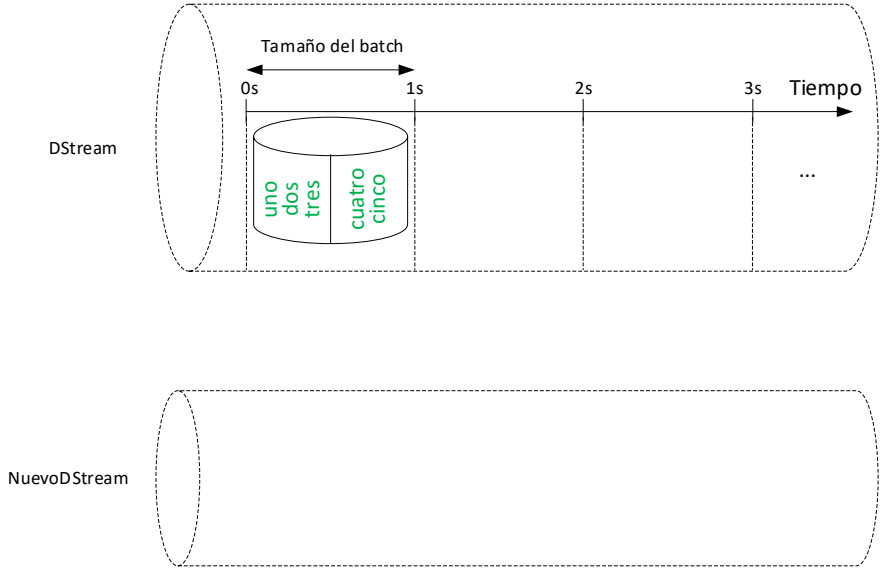



```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

def createQUEUERDD(data):
   rddQueue = []
   for datum in data:
       tmpRDD = ssc.sparkContext.emptyRDD()
       for partition in datum:
            partitionRDD = ssc.sparkContext.parallelize(partition, 1)
            tmpRDD = tmpRDD.union(partitionRDD)
        rddQueue += [tmpRDD]
   return rddQueue

sc = SparkContext(appName = "miPrograma")
ssc = StreamingContext(sc, 1)

data = [ [["uno", "dos", "tres"], ["cuatro", "cinco"]],  #RDD1 2 partitions
         [["seis", "siete", "ocho"], ["nueve"]]          #RDD2 2 partitions
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
…

ssc.start()
ssc.awaitTerminationOrTimeout(3)
```

Map

- Se aplica una función Map a cada registro

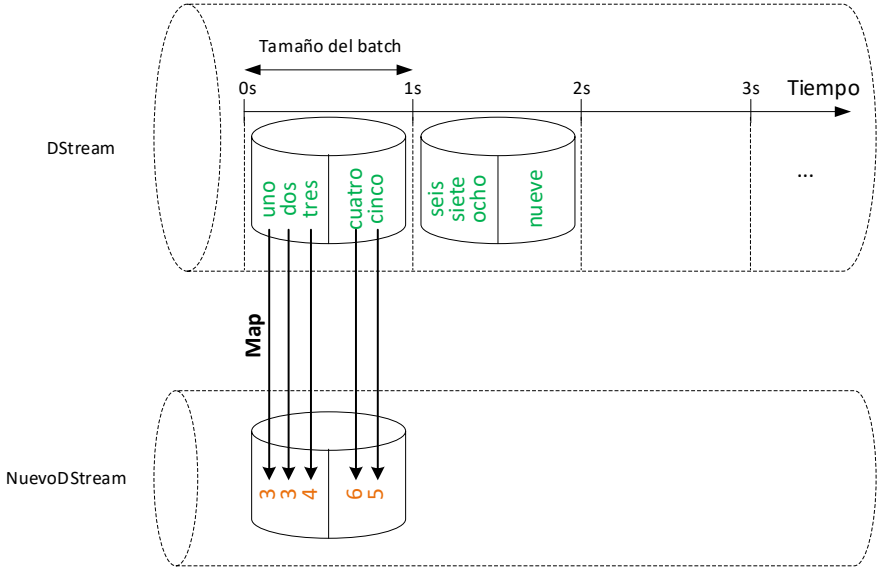
```
data = [[["uno", "dos", "tres"], ["cuatro", "cinco"]],
        [["seis", "siete", "ocho"], ["nueve"]]
       ]

rddQueueData = createQUEUERDD(data)

inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.map(lambda record: len(record))
```



Map

- Se aplica una función Map a cada registro

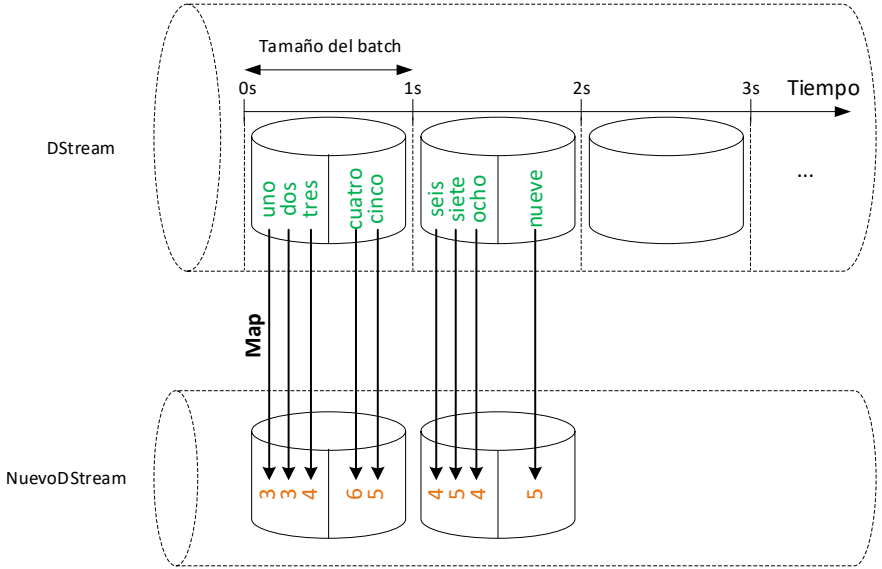
```
data = [[["uno", "dos", "tres"], ["cuatro", "cinco"]],
        [["seis", "siete", "ocho"], ["nueve"]]
       ]

rddQueueData = createQUEUERDD(data)

inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.map(lambda record: len(record))
```



Map

- Se aplica una función Map a cada registro

```
data = [[["uno", "dos", "tres"], ["cuatro", "cinco"]],
        [["seis", "siete", "ocho"], ["nueve"]]
       ]

rddQueueData = createQUEUERDD(data)

inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.map(lambda record: len(record))
```

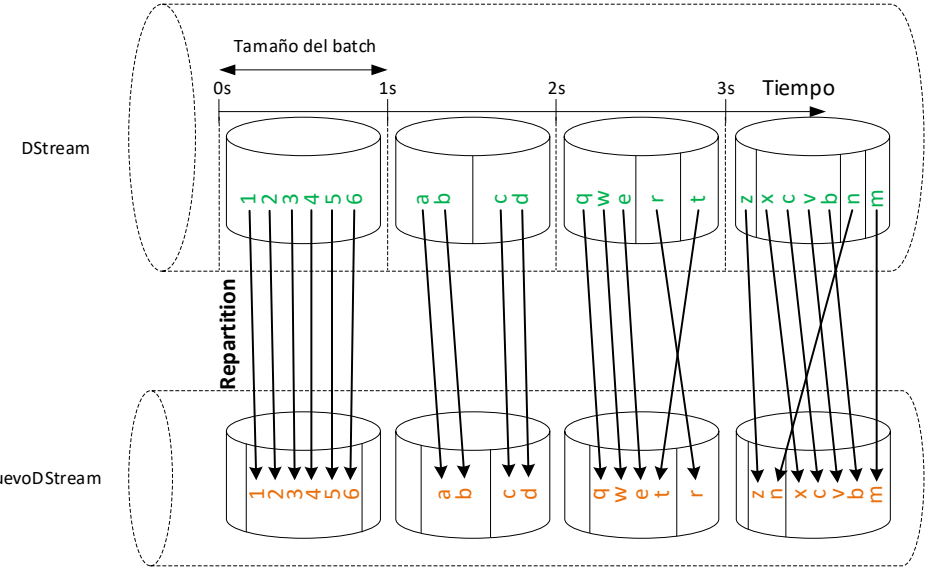


- Otros tipos de “Map”
  - FlatMap
  - Filter

Repartition

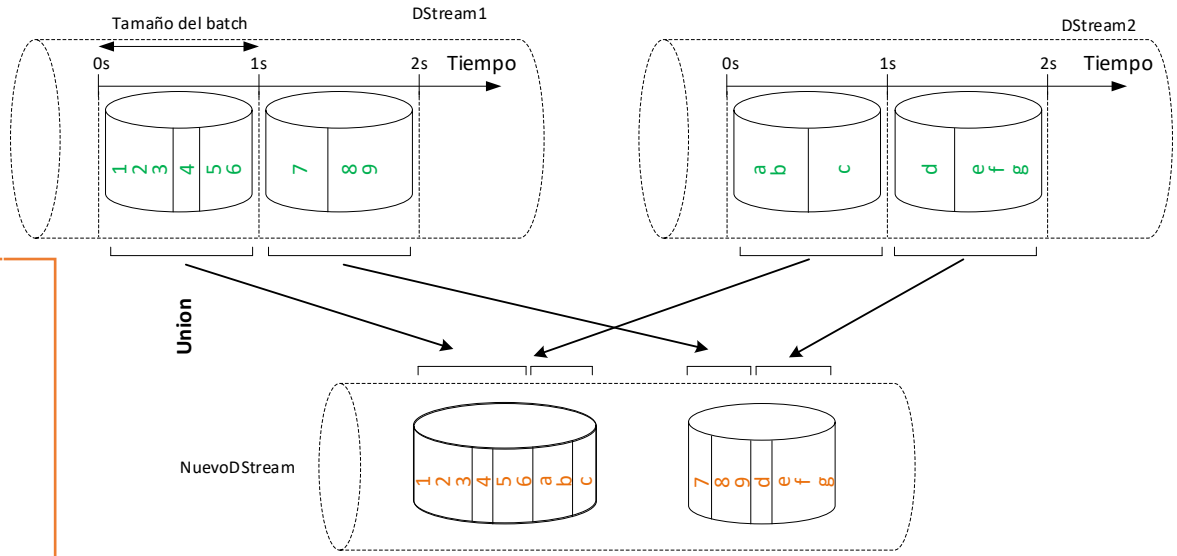
- Cambia el número de particiones de los RDD del DStream

```
data = [[[1, 2, 3, 4, 5, 6]],
        [["a", "b"], ["c", "d"]],
        [["q", "w", "e"], ["r"], ["t"]],
        [["z"], ["x", "c", "v", "b"],["n"],["m"]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.repartition(3)
```



Union

- Une dos Dstreams

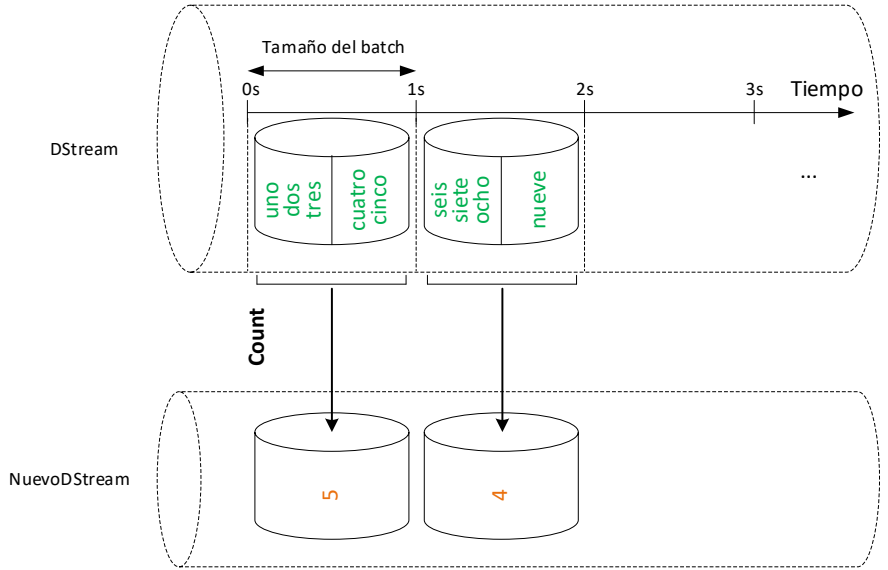



```
data1 = [[[1, 2, 3], [4], [5, 6]],
         [[7], [8, 9]]
        ]
data2 = [[["a", "b"], ["c"]],
         [["d"], ["e", "f", "g"]]
        ]
rddQueueData1 = createQUEUERDD(data1)
rddQueueData2 = createQUEUERDD(data2)

inputStream1 = ssc.queueStream(rddQueueData1)
inputStream2 = ssc.queueStream(rddQueueData2)

nuevoDStream = inputStream1.union(inputStream2)
```

- Otras “Union”
  - Joins:
    - Join
    - leftOuterJoin
    - rightOuterJoin
    - fullOuterJoin
  - Cogroup

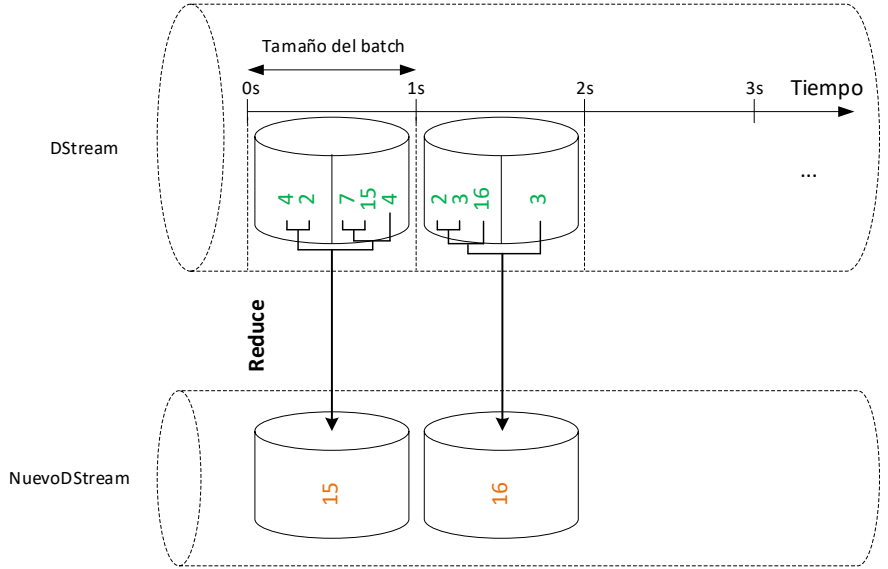
- Count
  - Cuenta el número de elementos de cada RDD
  - Devuelve RDDs con un registro

```
data = [[["uno", "dos", "tres"], ["cuatro", "cinco"]],
        [["seis", "siete", "ocho"], ["nueve"]]
       ]

rddQueueData = createQUEUERDD(data)

inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.count()
```



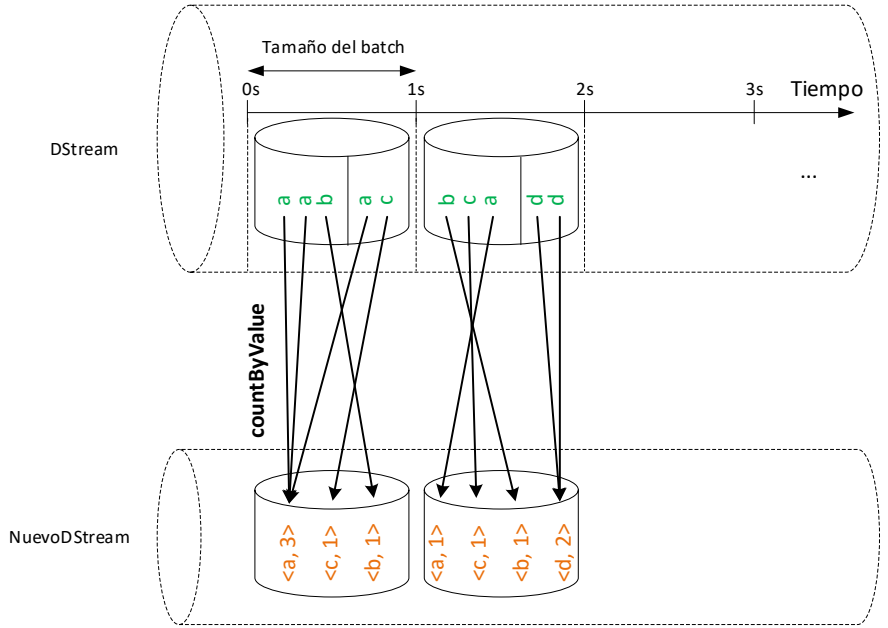
- Reduce
  - Aplica Reduce a cada RDD
  - Devuelve RDDs con un registro

```
def miFuncion(valor, acumulado):
  max = acumulado
  if valor > max:
     max = valor
  return max

data = [[[4, 2], [7, 15, 4]],
        [[2, 3, 16], [3]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.reduce(miFuncion)
```



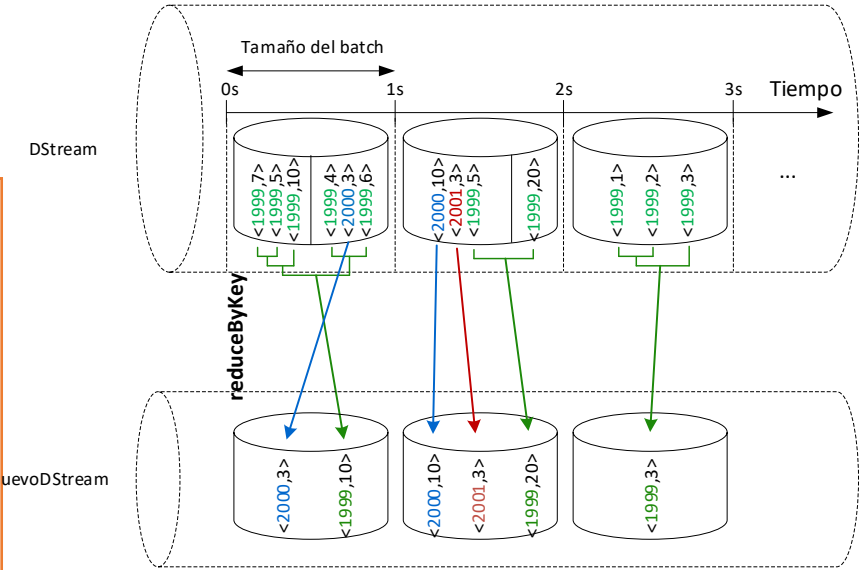
- CountByValue
  - Para cada RDD cuenta las veces que aparece cada registro

```
data = [[["a", "a", "b"], ["a", "c"]],
        [["b", "a", "c"], ["d", "d"]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.countByValue()
```



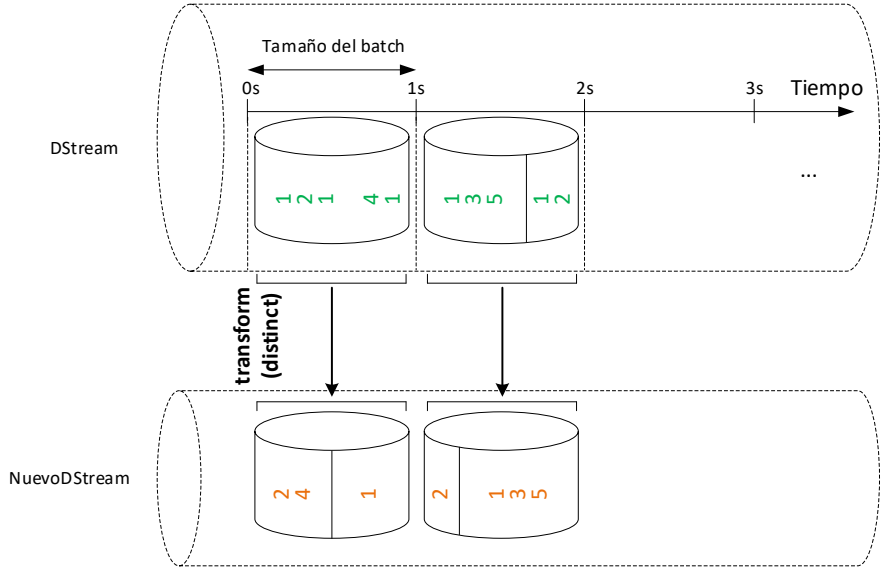
- reduceByKey
  - reduceByKey a cada RDD

```
def miFuncion(acumulado, valor):
  max = acumulado
  if valor > max:
     max = valor
  return max

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
      ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.reduceByKey(miFuncion)
```



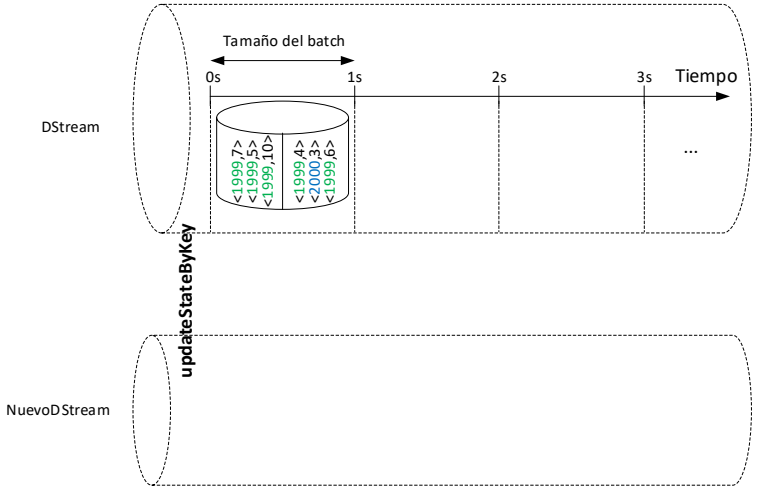
- transform
  - Aplica una función a cada RDD
  - Pueden utilizarse otras transformaciones de RDD que no estén implementadas en Spark Streaming

```
def miFuncion(rdd):
  return rdd.distinct()

data = [[[1, 2, 1], [4, 1]],
        [[1, 3, 5], [1, 2]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.transform(miFuncion)
```



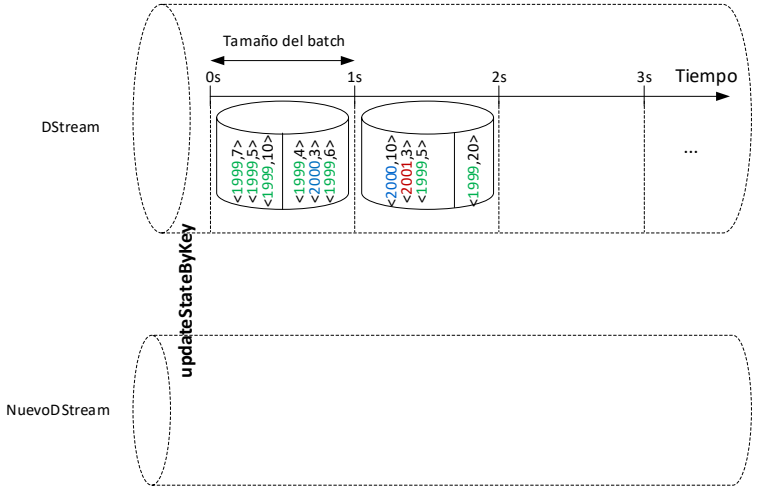
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



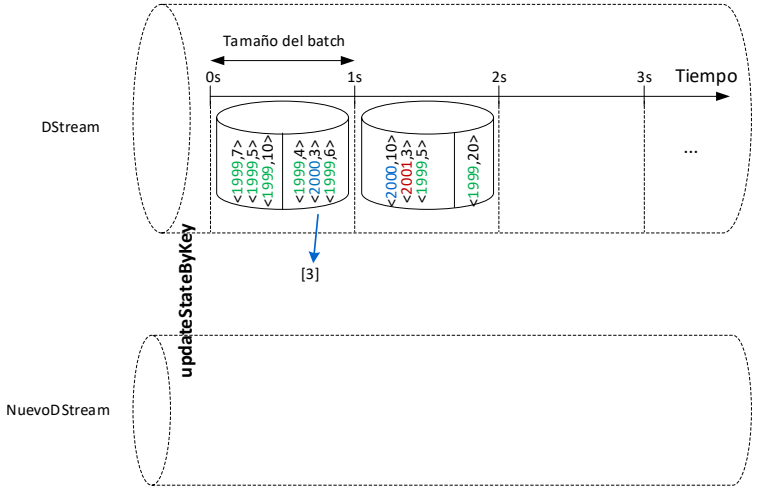
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



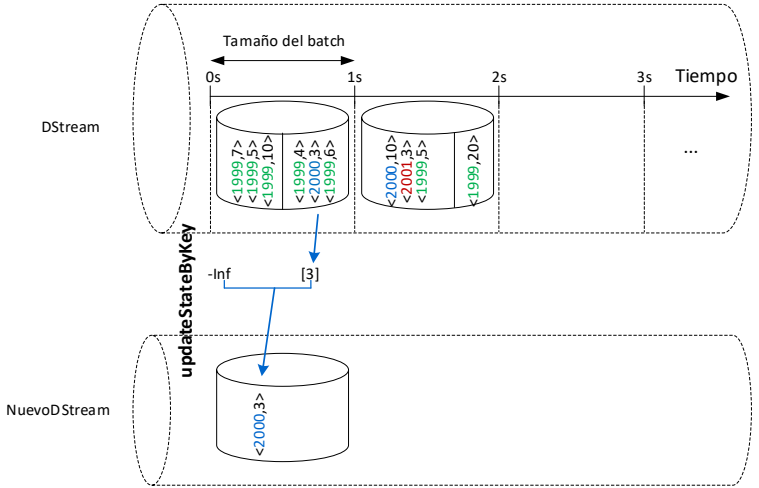
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



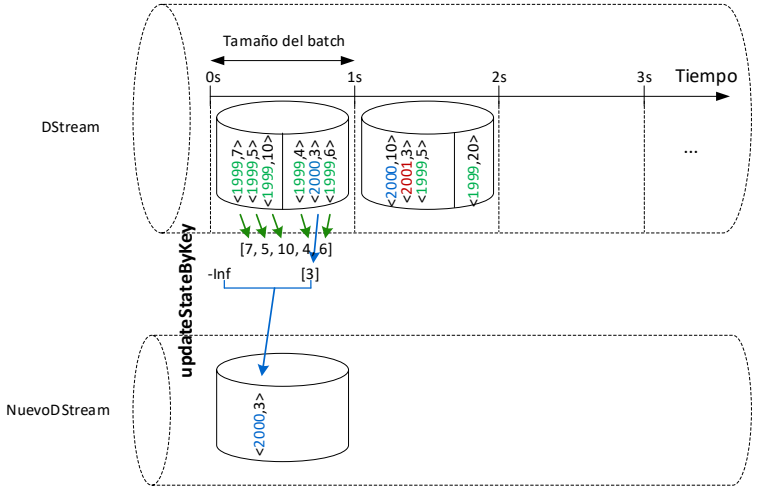
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



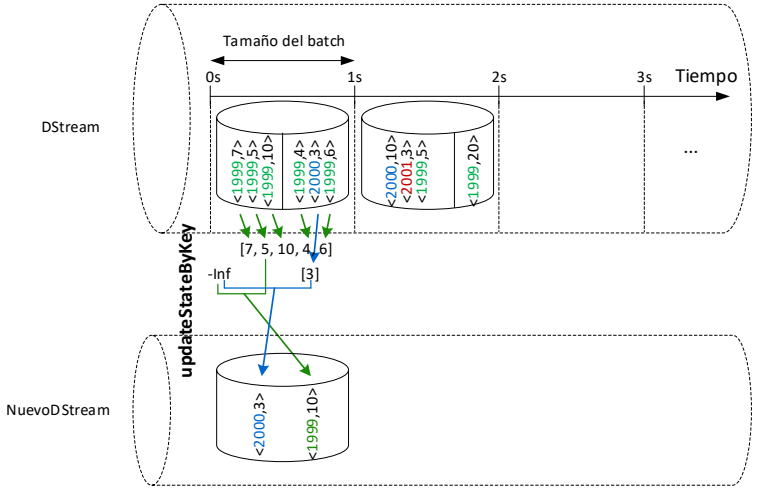
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



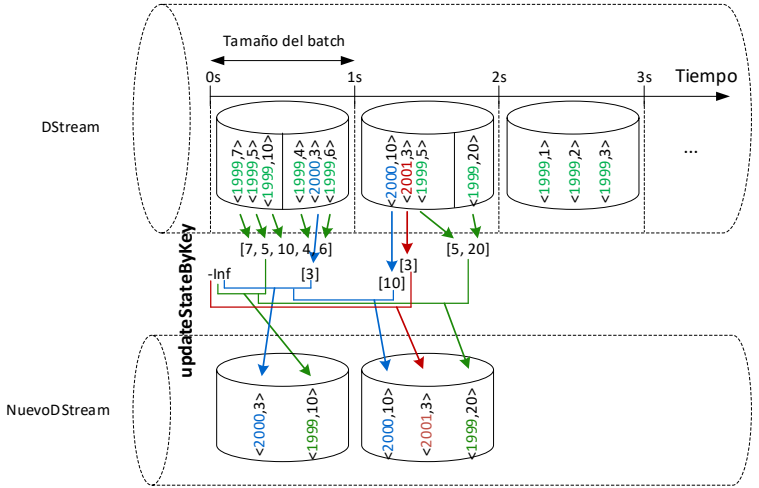
- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

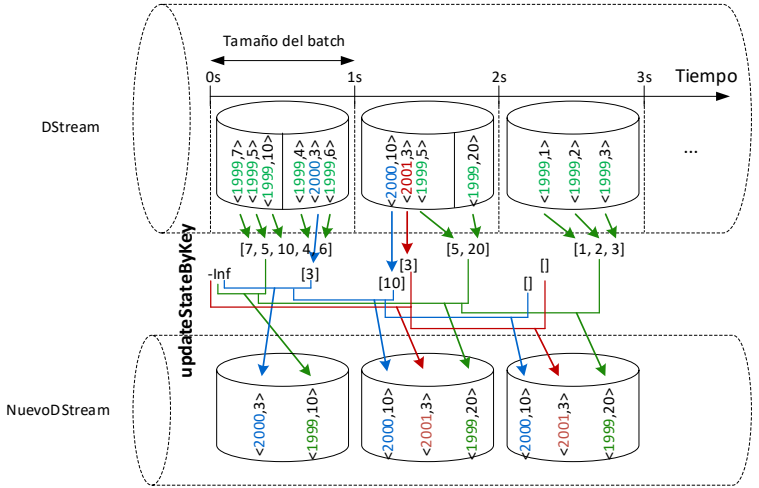
```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



- updateStateByKey
  - Guarda un estado por cada clave para todo el Dstream
  - Lo actualiza para todas las claves en cada batch

Notar que se actualizan todas las claves, aunque no estén en el último RDD

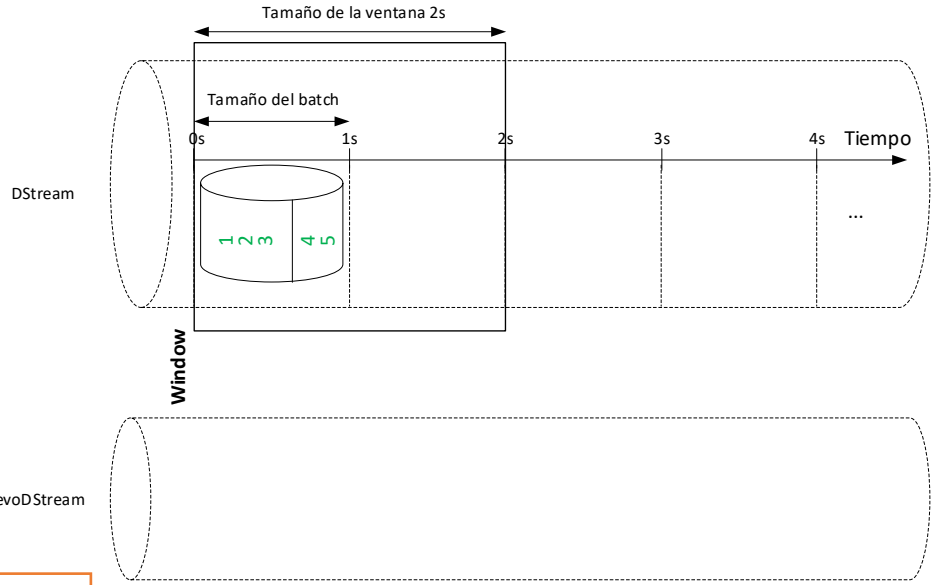
```
def miFuncion(todosLosValoresClave, acumulado):
  if acumulado is None :
     maximo = float("-Inf")
  else:
     maximo = acumulado
  if len(todosLosValoresClave) > 0:
     maximo = max(maximo, max(todosLosValoresClave))
  return maximo

data = [[[(1999, 7), (1999, 5), (1999, 10)], [(1999, 4), (2000, 3), (1999, 6)]],
        [[(2000, 10), (2001, 3), (1999, 5)], [(1999, 20)]],
        [[(1999, 1), (1999, 2), (1999, 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
nuevoDStream = inputStream.updateStateByKey(miFuncion)
```



window

- Permite generar una ventana **fija** o una slide

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 2)
```

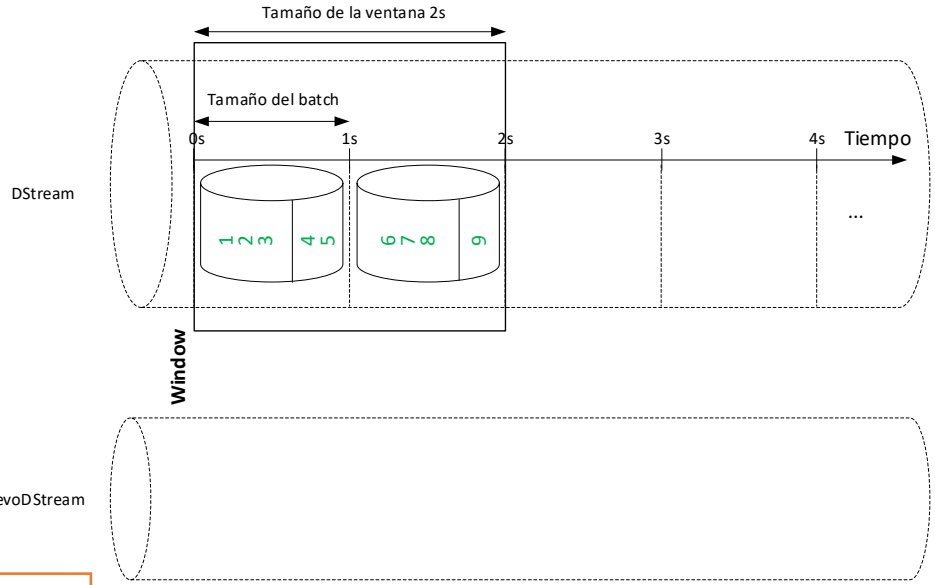


Ventana **fija**: Ventana de 2 segundos que desplaza 2 segundos (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana **fija** o una slide

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 2)
```



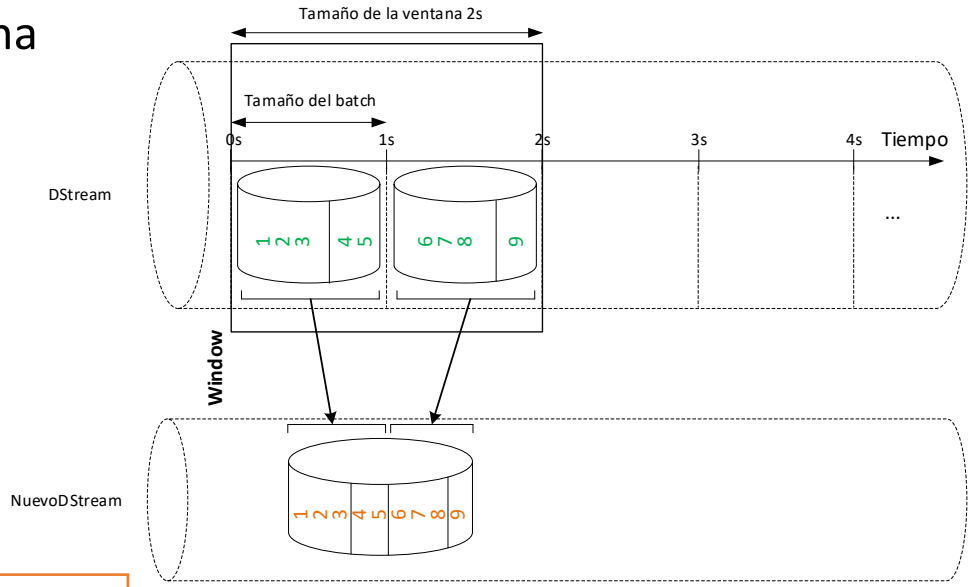
Ventana **fija**: Ventana de 2 segundos que desplaza 2 segundos (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana **fija** o una slide

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 2)
```

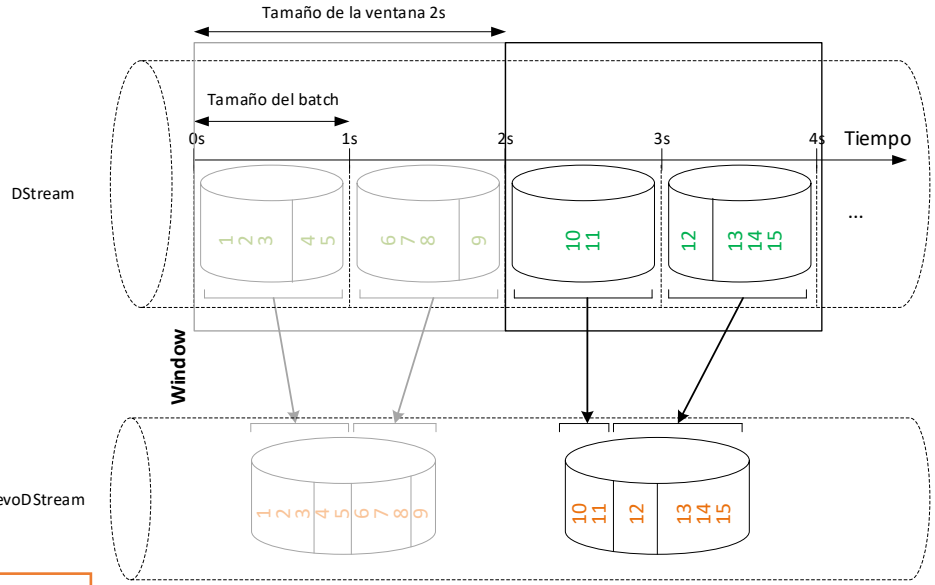
Ventana **fija**: Ventana de 2 segundos que desplaza 2 segundos (ambos tienen que ser múltiplos del tiempo de batch)



window

- Permite generar una ventana **fija** o una slide

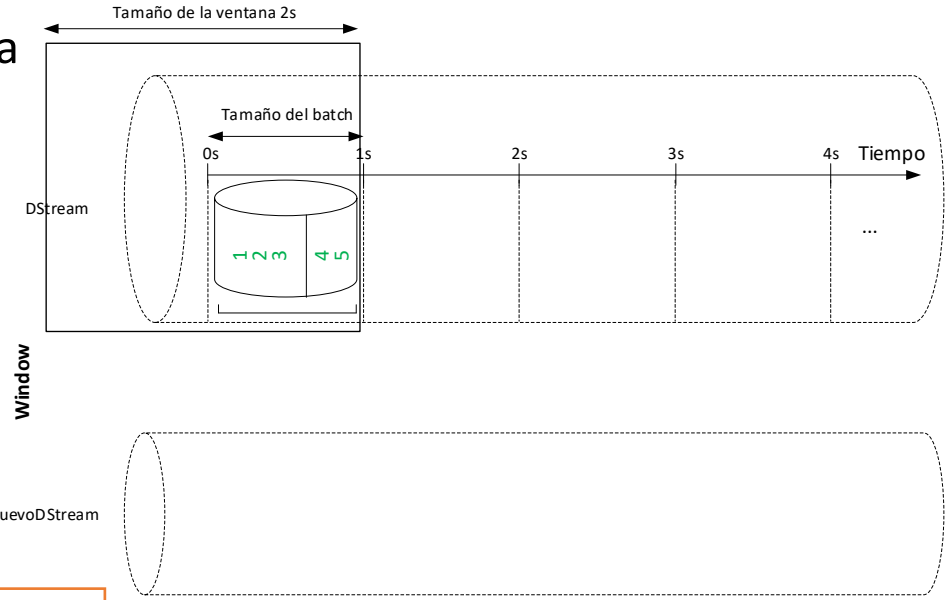
```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 2)
```



Ventana **fija**: Ventana de 2 segundos que desplaza 2 segundos (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana fija o una **slide**



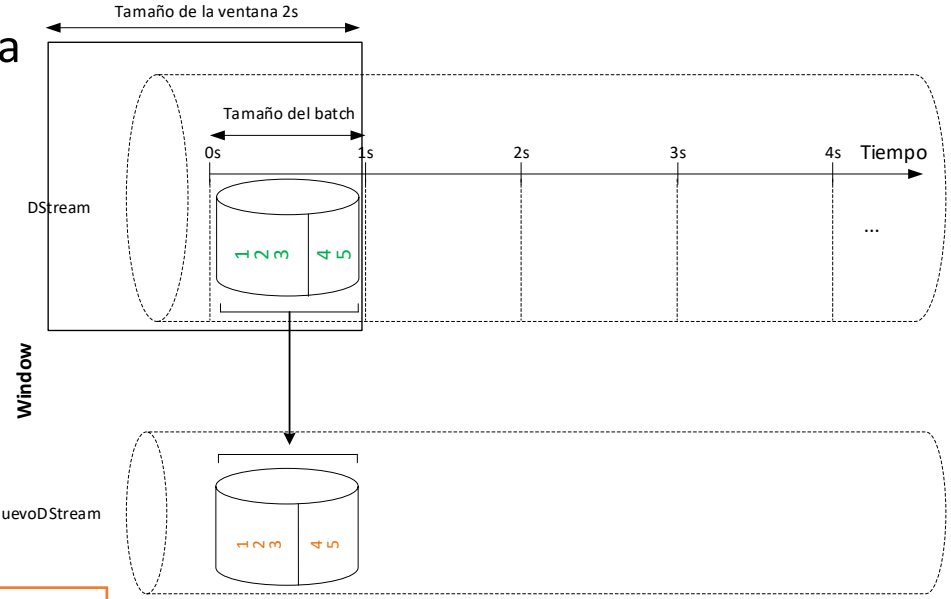

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 1)
```

Ventana **slide**: Ventana de 2 segundos que desplaza 1 segundo (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana fija o una **slide**

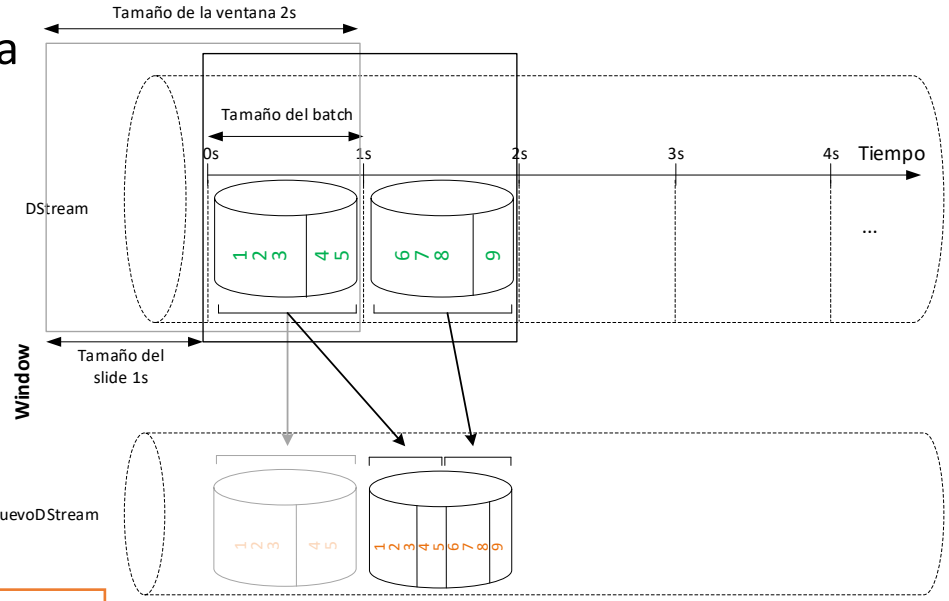
```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 1)
```



Ventana **slide**: Ventana de 2 segundos que desplaza 1 segundo (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana fija o una **slide**

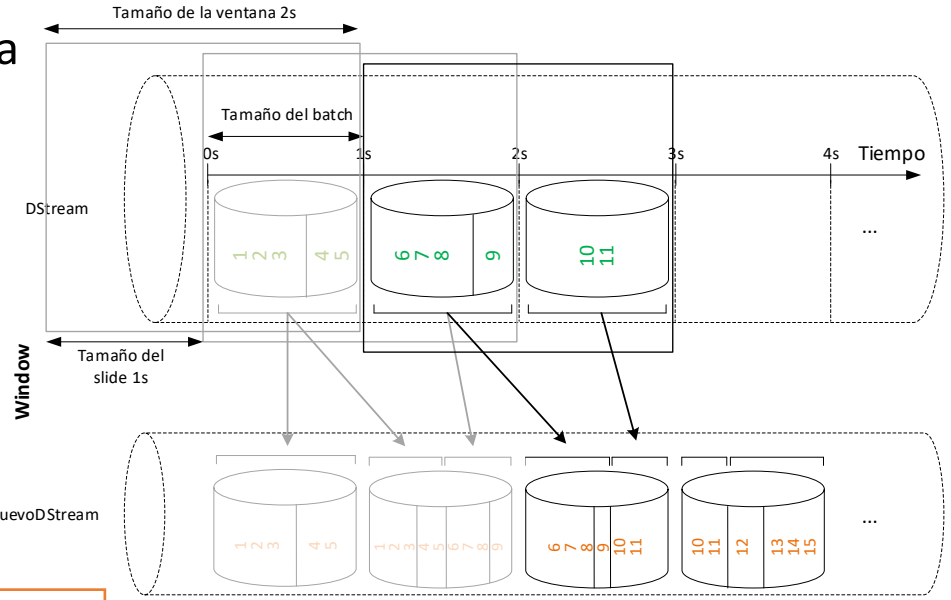



```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 1)
```

Ventana **slide**: Ventana de 2 segundos que desplaza 1 segundo (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana fija o una **slide**



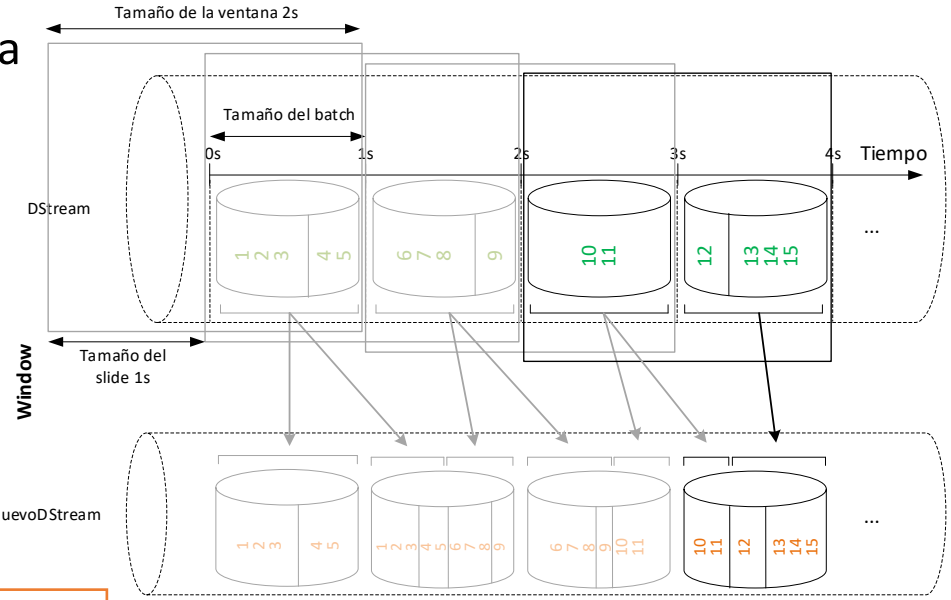

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 1)
```

Ventana **slide**: Ventana de 2 segundos que desplaza 1 segundo (ambos tienen que ser múltiplos del tiempo de batch)

window

- Permite generar una ventana fija o una **slide**

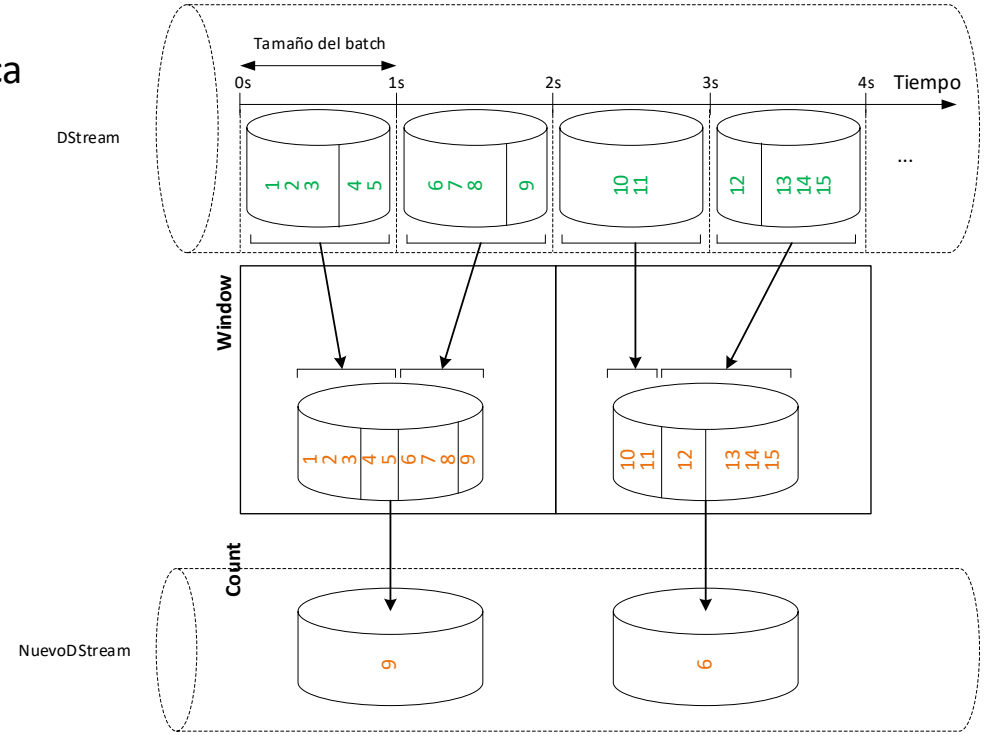
```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.window(2, 1)
```



Ventana **slide**: Ventana de 2 segundos que desplaza 1 segundo (ambos tienen que ser múltiplos del tiempo de batch)

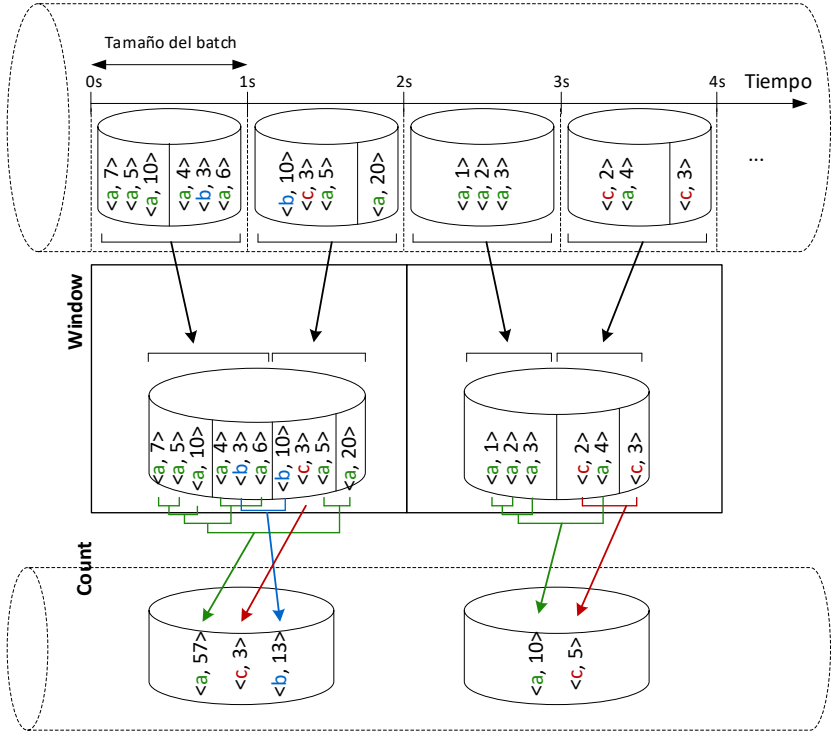
- {count, reduce}ByWindow
  - Hace una ventana y luego aplica la operación count o reduce

```
data = [[[1, 2, 3], [4, 5]],
        [[6, 7, 8], [9]],
        [[10, 11]],
        [[12], [13, 14, 15]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.countByWindow(2, 2)
```

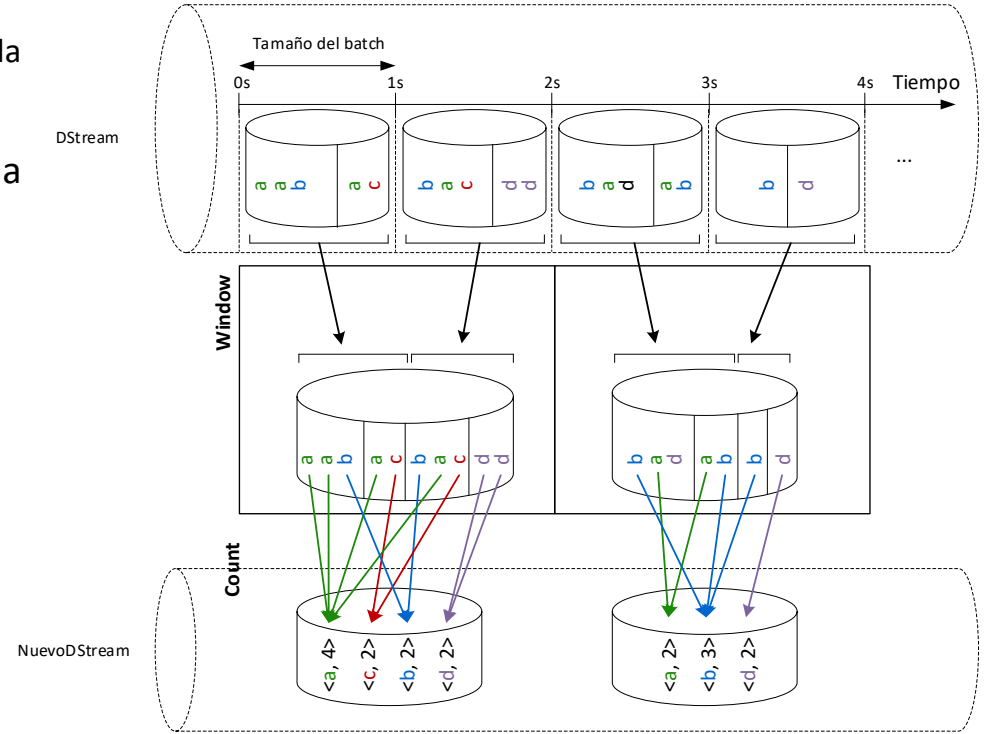


- reduceByKeyAndWindow
  - Hace una ventana y luego aplica la operación reduceByKey
  - Utiliza la función inversa de reduceByKey cuando hacemos slide para optimizar

```
def miFuncion(acumulado, valor):
  return acumulado + valor

def inversaMiFuncion(acumulado, valor):
  return acumulado - valor

data = [[[("a", 7), ("a", 5), ("a", 10)], [("a", 4), ("b", 3), ("a", 6)]],
        [[("b", 10), ("c", 3), ("a", 5)], [("a", 20)]],
        [[("a", 1), ("a", 2), ("a", 3)]],
        [[("c", 2), ("a", 4)], [("c", 3)]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.reduceByKeyAndWindow(miFuncion,
inversaMiFuncion, 2, 2)
```



- countByValueAndWindow
  - Hace una ventana y luego aplica la operación count
  - Aplica automáticamente una función inversa




```
data = [[["a", "a", "b"], ["a", "c"]],
        [["b", "a", "c"], ["d", "d"]],
        [["b", "a", "d"], ["a", "b"]],
        [["b"], ["d"]]
       ]

rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)

nuevoDStream = inputStream.countByValueAndWindow(2, 2)
```

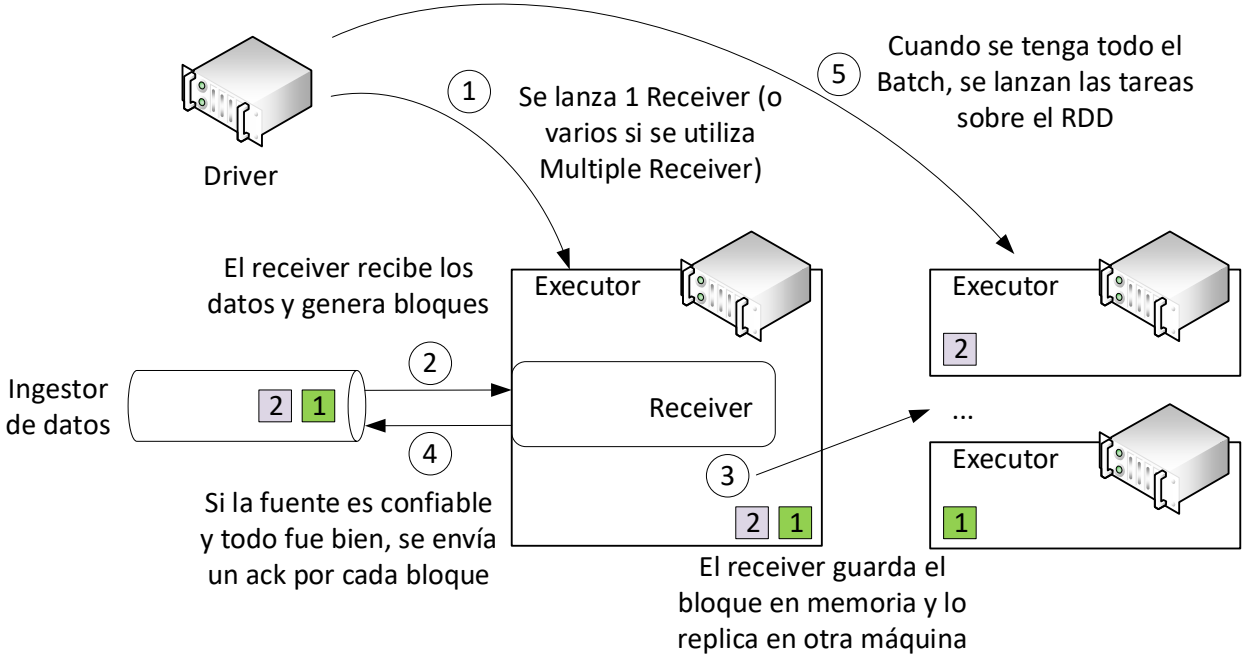
- Guardar el DStream:
  - `saveAsTextFiles(PREFIJO, [SUFIJO])`
    - Guarda cada Batch como PREFIJO-timestamp-SUFIJO
  - `saveAsObjectFiles(PREFIJO, [SUFIJO])`
    - Guarda cada Batch como sequence file
  - `saveAsHadoopFiles:`
    - Guarda cada Batch a partir de un OutputFormat de Hadoop

- forEachRDD:
  - Ejecuta una función por cada RDD
  - Se suele utilizar para guardar datos o enviarlos a otra fuente de datos
    - Ej. Crear un Kafka producer y enviarlos por Kafka
  - Sirve para depurar:

```
def printRDD(time, rdd):
  rdd_str = "RDD (" + str(time) + "):"
  partitions = rdd.glom().collect()
  numPartition = 0;
  for partition in partitions:
     rdd_str = rdd_str + "\n\tParticion: " + str(numPartition) + ": " + str(partition)
     numPartition = numPartition + 1
  print(rdd_str + "\n")

data = [[["uno", "dos", "tres"], ["cuatro", "cinco"]],
        [["seis", "siete", "ocho"], ["nueve"]]
       ]
rddQueueData = createQUEUERDD(data)
inputStream = ssc.queueStream(rddQueueData)
inputStream.foreachRDD(printRDD)
```

- print(NUM):
  - Imprime en el driver los NUM primeros datos
    - Por defecto imprime 10 datos
  - En Python es "`pprint`"

Pérdida de datos



- Problemas: ingestor de datos, máquina que ejecuta los datos, Receiver, Driver

- Problema en el ingestor de datos:
  - Suelen proporcionar mecanismos automáticamente
  - Si no reciben un ack, pueden renviar el dato
- Problemas en la máquina que ejecuta los datos:
  - Cada bloque está replicado, se puede recuperar sin pérdidas
  - Por defecto se utiliza nivel de persistencia `MEMORY_AND_DISK_SER_2`

- Problema en el Receiver:
  - El Driver lanza otro Receiver:
    - Los datos replicados se pueden ejecutar
    - Los datos no replicados pueden ser re-enviados si la fuente es confiable (no se envió ack)
  - Mecanismo de journaling WAL (Write Ahead Log):
    - Persiste los datos en un sistema tolerante a fallos (ej. HDFS)
    - Activación:
      - `spark.streaming.receiver.writeAheadLog.enable = True`
      - `ssc.checkpoint(checkpointDirectory)`
    - El Receiver enviará el ack después de persistir el dato
    - Es recommendable quitar la replicación (`StorageLevel.MEMORY_AND_DISK_SER`)

- Problema en el driver:
  - Se pierde toda la información del programa
  - Checkpoint:
    - Permite recuperarse de fallos en el driver
    - Permite guardar frecuentemente una parte del lineage (ej. HDFS)
    - Para activarlo:

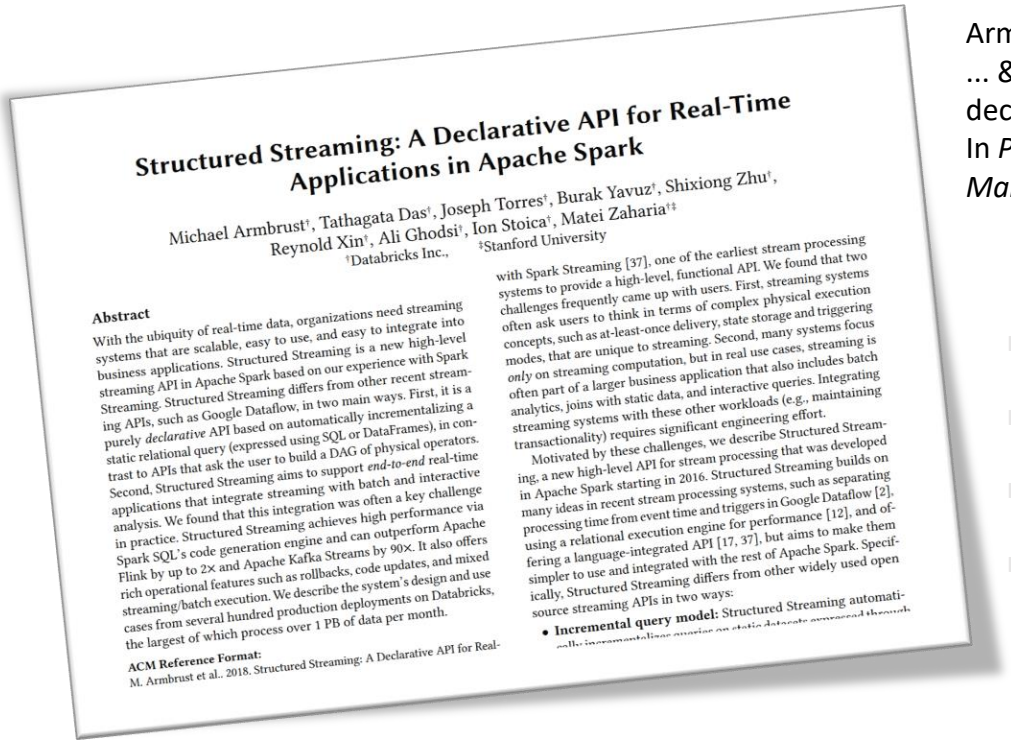
```
def functionToCreateContext():
    sc = SparkContext(...)
    ssc = StreamingContext(...)
    lines = ssc.socketTextStream(...
    ...
    ssc.checkpoint(checkpointDirectory)
    return ssc

# Get StreamingContext from checkpoint data or create a new one
context = StreamingContext.getOrCreate(checkpointDirectory, functionToCreateContext)
```

- Problema en el driver:
  - Se pierde toda la información del programa
  - Checkpoint:
    - Permite recuperarse de fallos en el driver
    - Permite guardar frecuentemente una parte del lineage (ej. HDFS)
    - Para activarlo:

```
def functionToCreateContext():
    sc = SparkContext(...)
    ssc = StreamingContext(...)
    lines = ssc.socketTextStream(...
    ...
    ssc.checkpoint(checkpointDirectory)
    return ssc
```

También se tiene que que utilizar para las funciones stateful

```
# Get StreamingContext from checkpoint data or create a new one
context = StreamingContext.getOrCreate(checkpointDirectory, functionToCreateContext)
```



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 applications that integrate streaming with batch and interactive analysis. We found that this integration was often a key challenge in practice. Structured Streaming achieves high performance via Spark SQL's code generation engine and can outperform Apache Flink by up to 2× and Apache Kafka Streams by 90×. It also offers rich operational features such as rollbacks, code updates, and mixed streaming/batch execution. We describe the system's design and use cases from several hundred production deployments on Databricks, the largest of which process over 1 PB of data per month.

ACM Reference Format:

M. Armbrust et al.. 2018. Structured Streaming: A Declarative API for Real-

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 Streaming, a new high-level API for stream processing that was developed in Apache Spark starting in 2016. Structured Streaming builds on many ideas in recent stream processing systems, such as separating processing time from event time and triggers in Google Dataflow [2], using a relational execution engine for performance [12], and offering a language-integrated API [17, 37], but aims to make them simpler to use and integrated with the rest of Apache Spark. Specifically, Structured Streaming differs from other widely used open source streaming APIs in two ways:

- **Incremental query model:** Structured Streaming automati-

Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., ... & Zaharia, M. (2018, May). Structured streaming: A declarative API for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 601-613)

- Para procesamiento estructurado
- Utiliza DataFrame en lugar de RDD
- Permite utilizar R (limitado)
- Permite modo Continuous processing
  - No hace micro-batch
  - No tiene todas las funcionalidades

Gracias

viu | Universidad Internacional de Valencia