



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática

Procesamiento Sistemas Big Data

PH.D. Erika Rosas Olivos

erosas@inf.utfsm.cl

Septiembre 2023



Main references

- Designing Data-Intensive Applications, Martin Kleppmann, O'Reilly, 2017
- MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI 2004.
- Big Data: principles and best practices of scalable real-time data systems, Nathan Marz, Manning, 2015
- Kafka The Definitive Guide, Narkhede, Shapira & Palino, O'Reilly, 2017
- High Performance Spark, Karay & Warren, O'Reilly, 2017
- Getting stated with Storm, Leibiusky, Eisbruch & Simonassi, O'Reilly, 2012
- Big Data Science and Analytics, A hands-on approach, Arshdeep Bahia and Vijay Madisetti, 2016.
- Mastering Large Datasets with Python, John T. Wolohan, Manning, 2019.



Objetivo

- Estudiar modelos distribuidos y tecnologías que soporten el procesamiento de big data.
- Enfoque: Estudiar teoría basado en sistemas reales y aplicaciones.
- Énfasis en identificar y justificar las desiciones de diseño y tradeoff asociados.



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

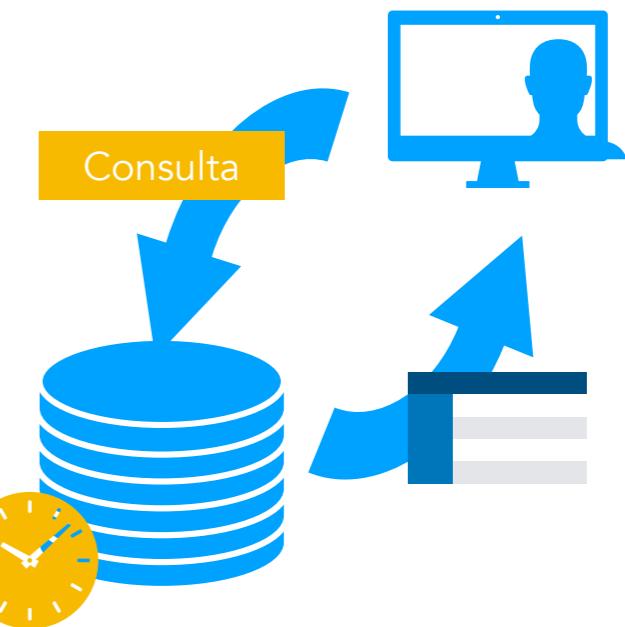
Departamento de Informática

Tipos de procesamiento

Procesamiento de consultas interactivas

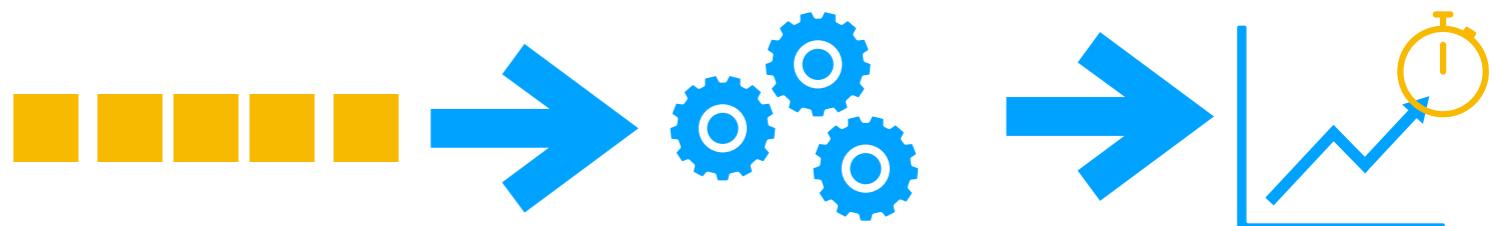
La principal métrica es el tiempo de respuesta

Ejemplos APIs basadas en HTTP/REST



Procesamiento en batch (offline):
La entrada son grandes cantidades de datos, trabajos los procesan y se produce una salida.
Métrica principal es throughput: cantidad de datos procesados por período de tiempo.

Procesamiento stream (casi tiempo real):
Un trabajo opera sobre eventos poco después de que ocurren.





Motivación

- Mayor avance: Google creó en el 2004 el algoritmo Map-Reduce.
- Este ha sido implementado en sistemas de código abierto como Hadoop, CouchDB, MongoDB.
- Tan nuevo no era! El procesamiento en batch era usado de manera similar en 1940s y 1950s (Máquinas clasificadores de tarjetas de IBM)

A dos programadores jóvenes se les ocurrió como ranquear las páginas de Internet. Querían hacerlo en base a la importancia de los sitios que las linkeaban en Internet. Pensaban que Internet tenía que ser como en el colegio: eres más importante si los chicos importantes hablan de ti. Les encantó la idea, pero ¿cómo podrían analizar todo Internet?



Procesamiento en batch usando herramientas Unix

- Archivo de log de un servidor Web, se agrega una línea luego de cada petición.

```
216.58.210.78 - - [27/Feb/2015:17:55:11 +0000] "GET /css/typography.css HTTP/1.1"  
200 3377 "http://martin.kleppmann.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X  
10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115  
Safari/537.36"
```

- Formato del log:
- Análisis del log:

```
$remote_addr $remote_user [$time_local] "$request"  
$status $body_bytes_sent "$http_referer" "$http_user_agent"
```

```
cat /var/log/nginx/access.log | ①  
awk '{print $7}' | ②  
sort | ③  
uniq -c | ④  
sort -r -n | ⑤  
head -n ⑥ 5
```

- 1) Leer el archivo de log
- 2) Dividir la linea, salida es el argumento 7
- 3) Ordenar alfabéticamente la lista de URLs
- 4) Filtrar las líneas repetidas, -c devuelve un contador por cada URL diferente.
- 5) Ordena por el número al comienzo de cada línea.
- 6) La salida es solo las primeras 5 líneas.

Procesa GBs en segundos!



Procesamiento en batch usando Ruby

- Este es un programa simple que hace lo mismo que el anterior.

`counts = Hash.new(0)` ① Counts mantiene un contador por cada URL

```
File.open('/var/log/nginx/access.log') do |file|
  file.each do |line|
    url = line.split[6] ② Toma el argumento 7vo
    counts[url] += 1 ③ Incrementa el contador
  end
end
```

`top5 = counts.map{|url, count| [count, url] }.sort.reverse[0...5]` ④ Ordena la tabla hash por el contador
`top5.each{|count, url| puts "#{count} #{url}" }` ⑤ Imprime las 5 primeras

Hash Table en memoria



Comparación ¿Cuál es mejor?

- Depende del número de URL diferentes.
- Si todas caen en memoria es mejor la versión de Ruby.
- Sino, el enfoque de ordenar hace más eficiente el uso de discos y paralelización multi-core.

- Ideas importantes Unix
 - Cada programa hace una cosa bien
 - Una salida de un programa puede ser la entrada de otro.
 - Unix pipes.
 - Interfaz uniforme -> archivo
 - Solo corre en una máquina.



Procesamiento Batch: Map Reduce

- Modelo de programación para procesar muchos terabytes en miles de máquinas.
- Dos operaciones:
 - MAP: Procesa un par clave/valor y genera un conjunto intermedio de clave/valor.
 - REDUCE: Integra todos los valores intermedios asociados a la misma clase intermedia.
 - Implementaciones: Automáticamente paralleliza y ejecuta.

El sistema se encarga de

- Partitionar los datos de entrada.
- Planificar la ejecución
- Manejo de fallas
- Manejo de comunicación inter-máquinas



Modelo de programación

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI 2004.

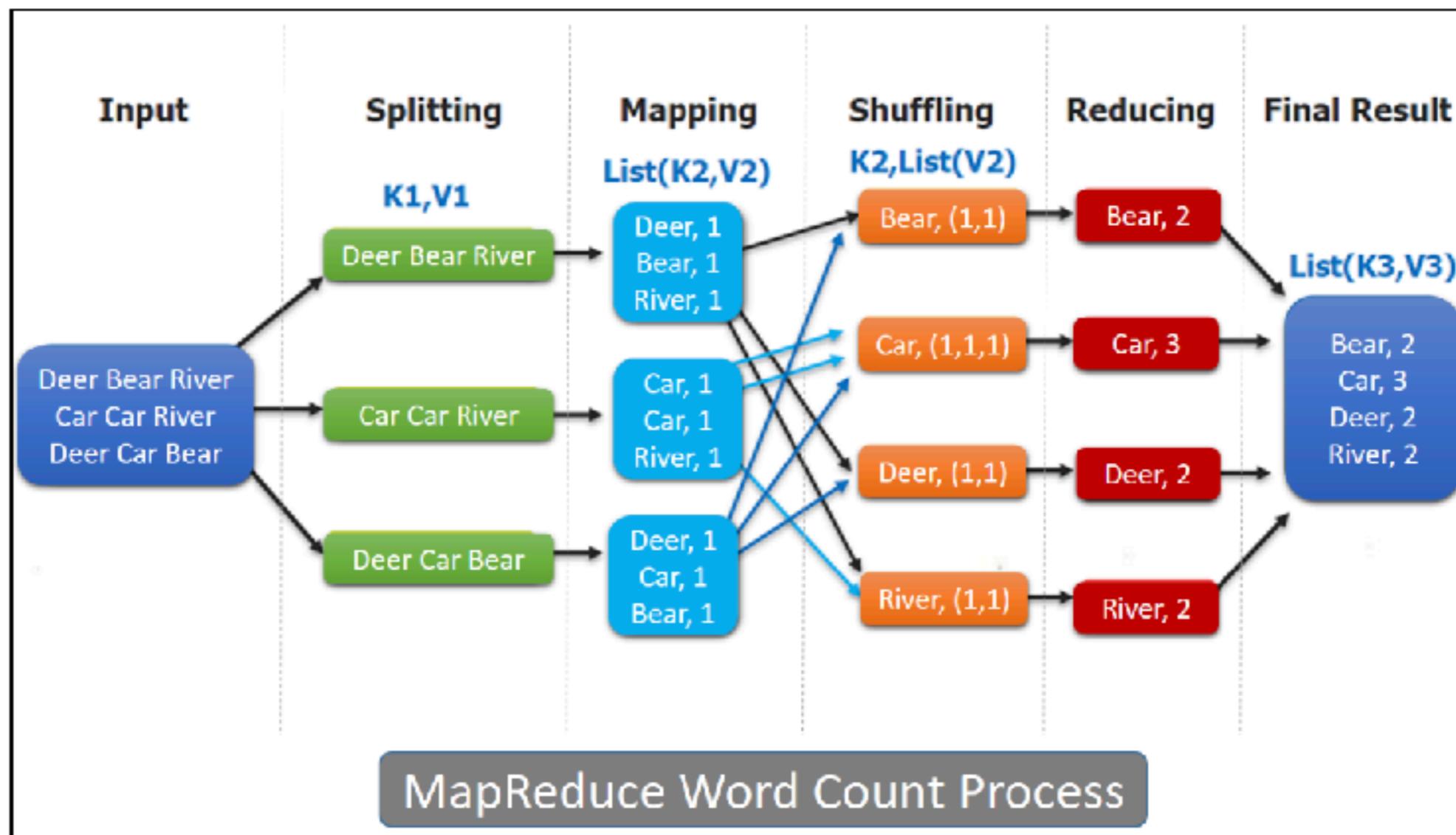
- Entrada: Pares Clave/valor
- Salida: Pares Clave/valor
- Funciones:
 - Map (Escrita por el usuario)
 - Toma los pares de entrada y produce pares intermedios.
 - Map-reduce automáticamente agrupa los valores intermedios asociados a la misma clave y se los pasa a la función Reduce.
 - Reduce (Escrita por el usuario):
 - Acepta la clave con todos los valores asociados a esa clave para agruparlos.

```
map(String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:  
        EmitIntermediate(w, "1");  
  
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

En Hadoop MapReduce el mapper y el reducer son clases Java. En MongoDB y CouchDB son funciones Javascript.



Figura conteo de palabras



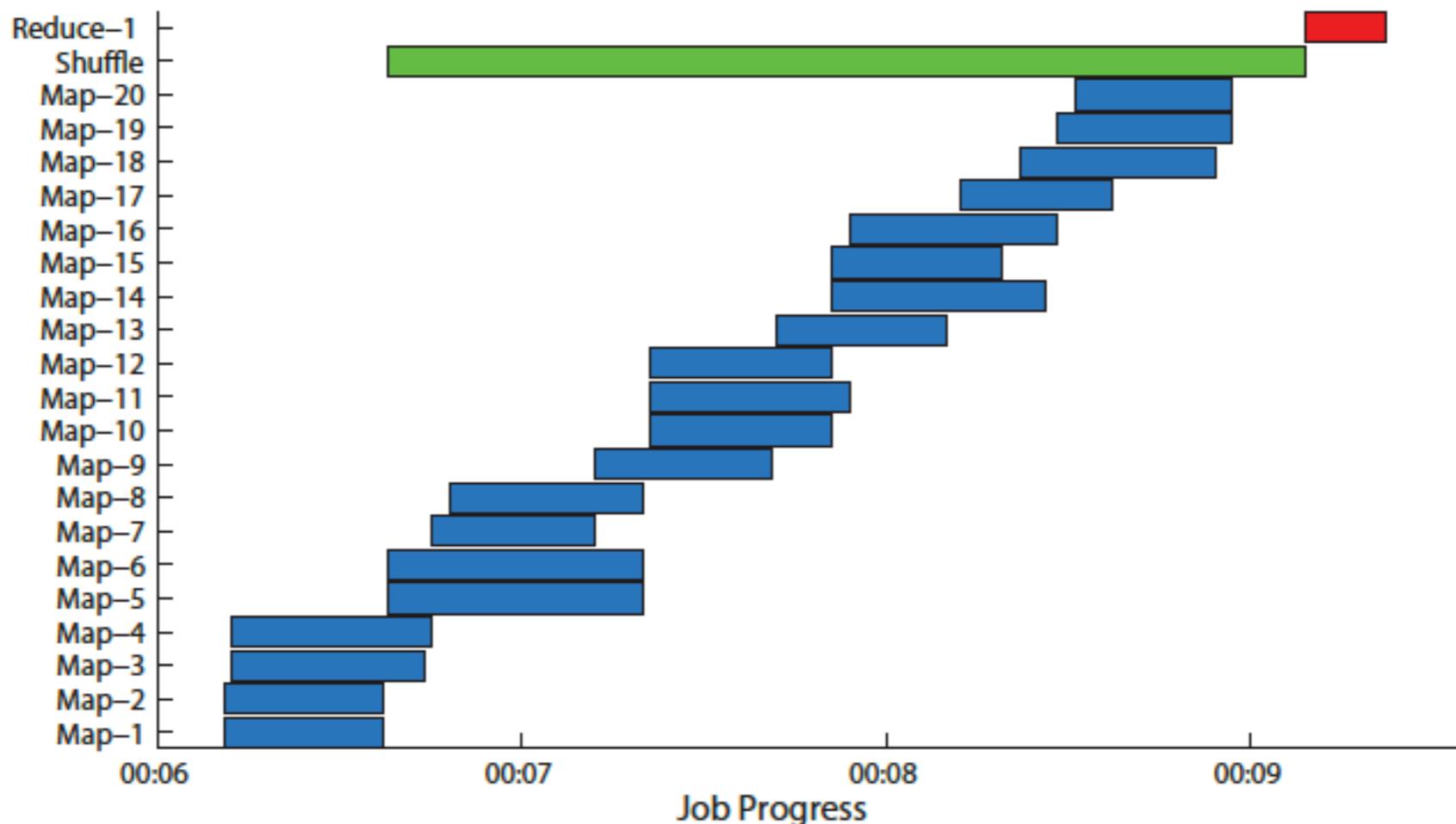
Source: <https://www.edupristine.com/blog/hadoop-mapreduce-framework>



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática

Conteo de palabras asignación tiempo

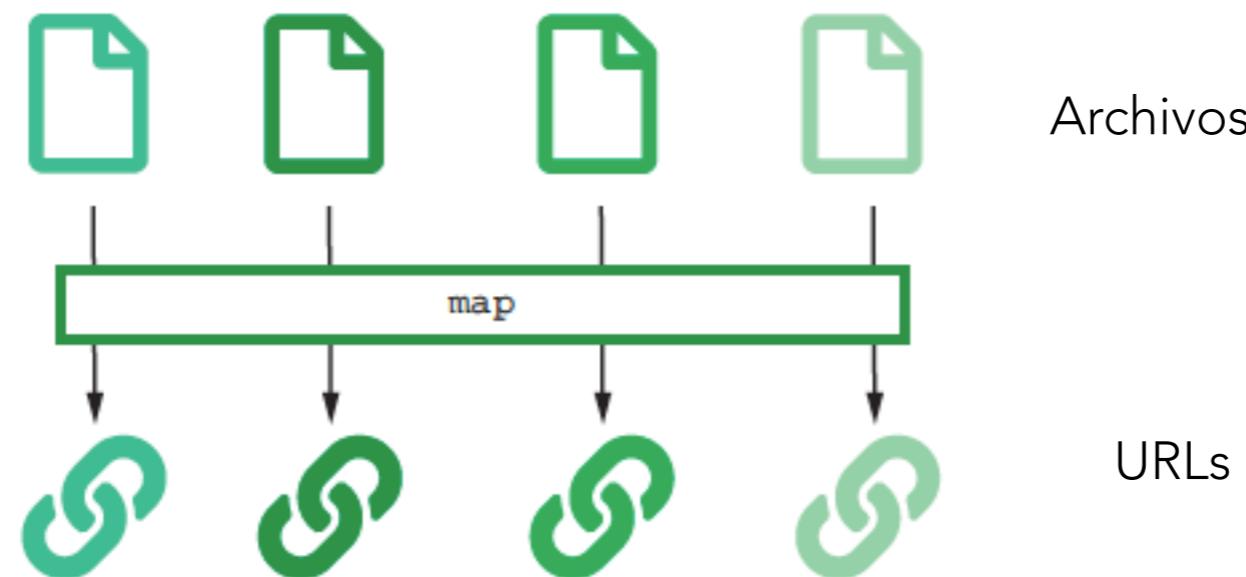


pproach, Arshdeep Bahia and Vijay Madisetti, 2016.



Función MAP para transformar datos

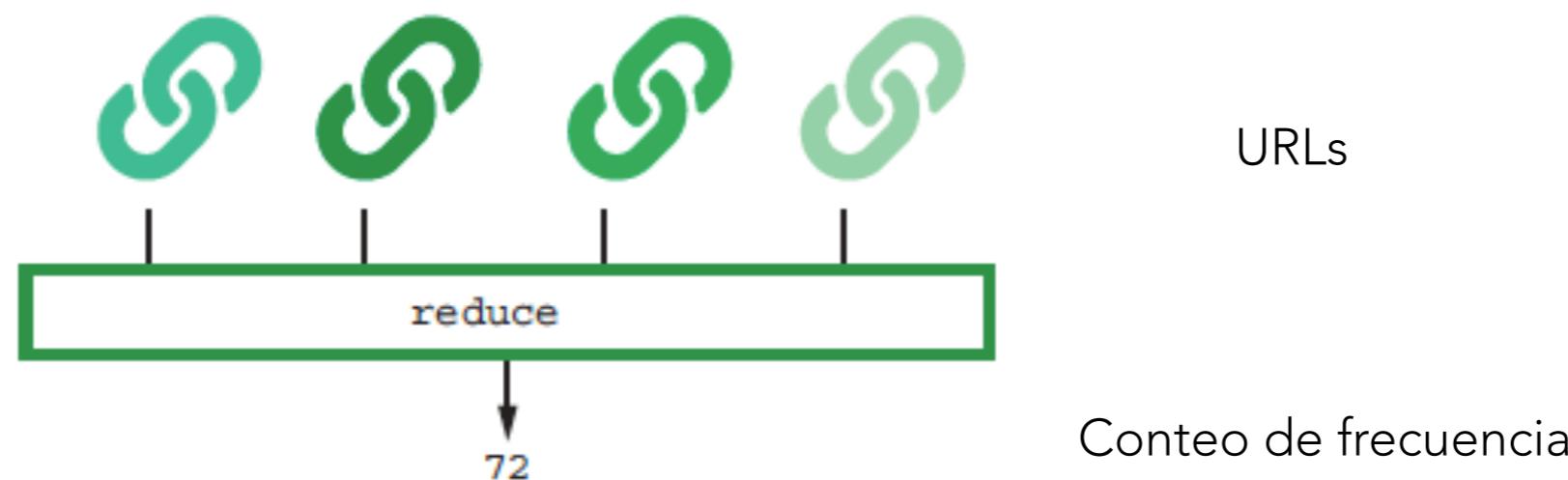
- Típicamente se usa la función MAP para transformar secuencias de datos desde un tipo a otro.
- Retiene o aumenta el número de objetos que entraron.
- Ejemplos: entra un número n , map le suma siete 7, el valor de resultado es la suma.
 - Se crea una función llamada `add_seven`, que recibe n y devuelve $n+7$.
- Se puede usar para reemplazar loops FOR.





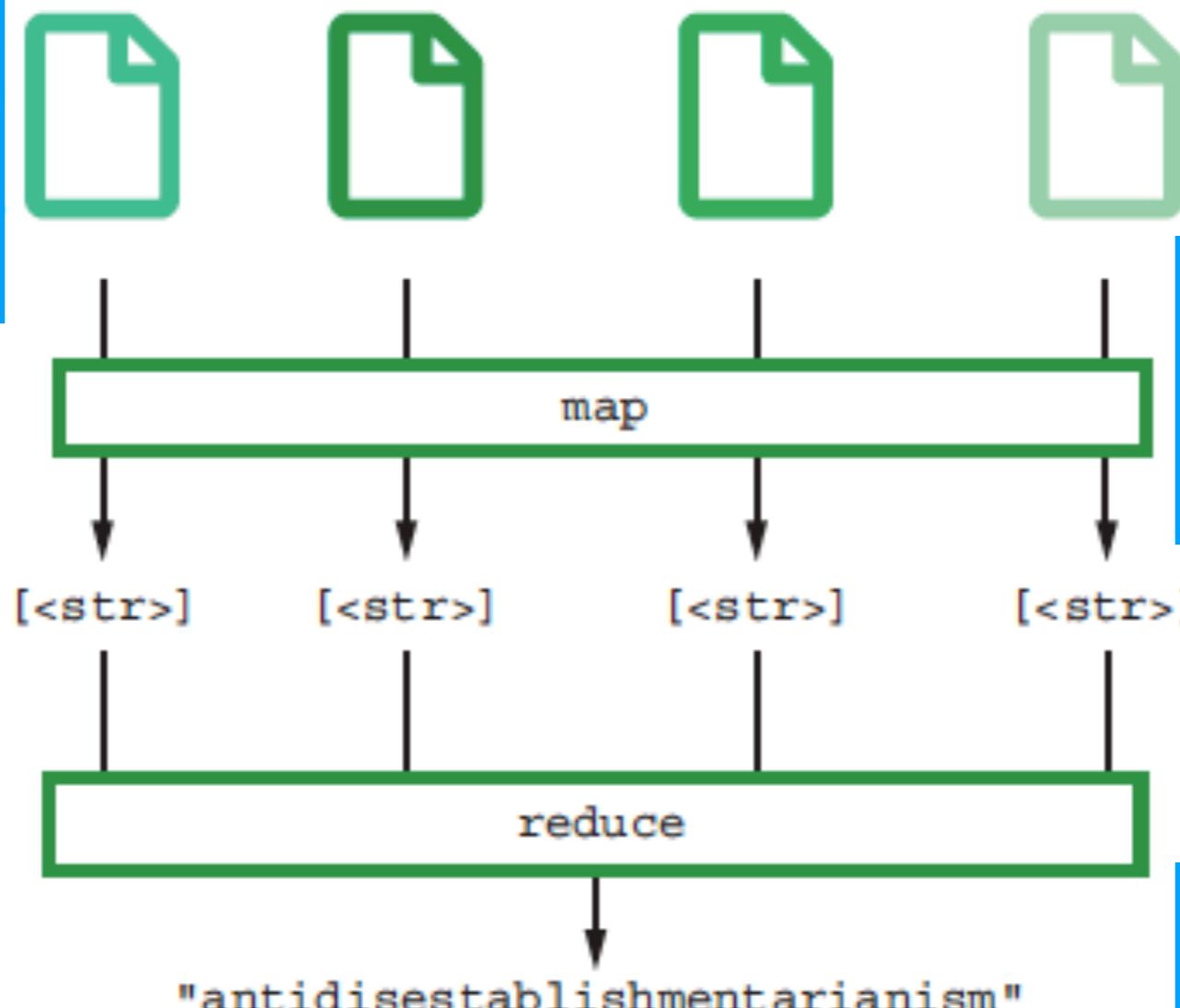
Función REDUCE para transformaciones avanzadas

- Por ejemplo, tomar una secuencia y convertirla en algo de diferente largo.
- Nos permite transformar una secuencia de datos en una estructura de datos de cualquier tipo o tamaño.
- Más flexible que el MAP.





Map y reduce pueden combinarse para ejecutar flujos de trabajo del estilo "transformar y consolidar"



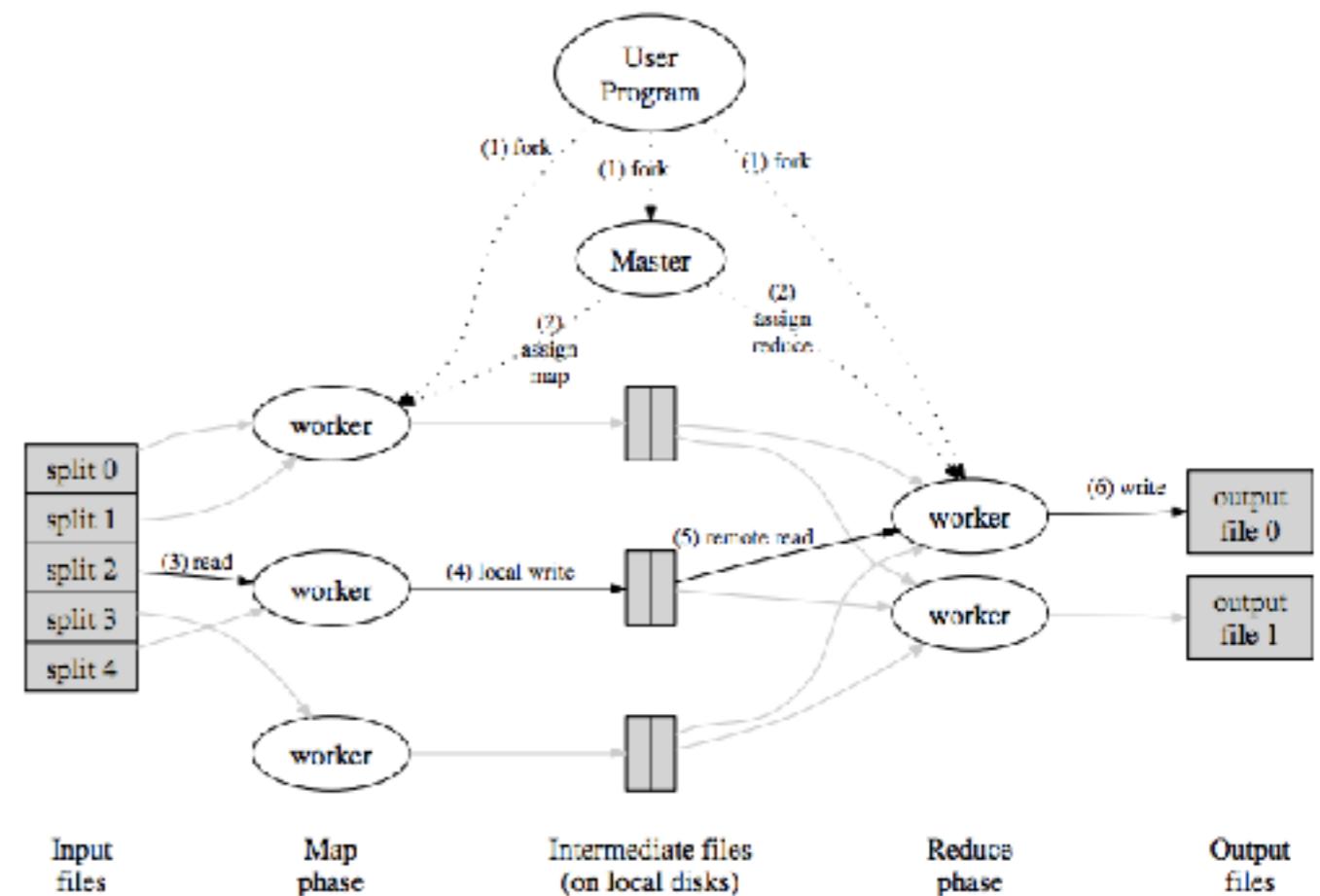
La función map puede ser usada para transformar las páginas Web en listas de palabras como strings

Podemos usar la función reduce para encontrar el string más largo de todas las listas de strings.



¿Cómo funciona Map-Reduce?

- Las invocaciones del Map son distribuidas en múltiples máquinas, participando automáticamente los datos de entrada en M partes.
- Las invocaciones del Reduce son distribuidas al partitionar el espacio de la clave intermedia en R partes.



En HDFS

Shuffle
El ordenamiento es local para el trabajador que realiza el map. Luego, el trabajador que hará el reducer busca el archivo intermedio.

En HDFS



Ejemplo: Número de visitas a página en el tiempo

```
function pageviewsOverTime(masterDataset, url, startHour, endHour) {  
    pageviews = 0  
    for(record in masterDataset) {  
        if(record.url == url &&  
            record.time >= startHour &&  
            record.time <= endHour) {  
            pageviews += 1  
        }  
    }  
    return pageviews  
}
```

Código similar al del conteo de palabras, pero la clave que emite el mapper es una estructura que contiene la URL y la hora en que se vio la página.

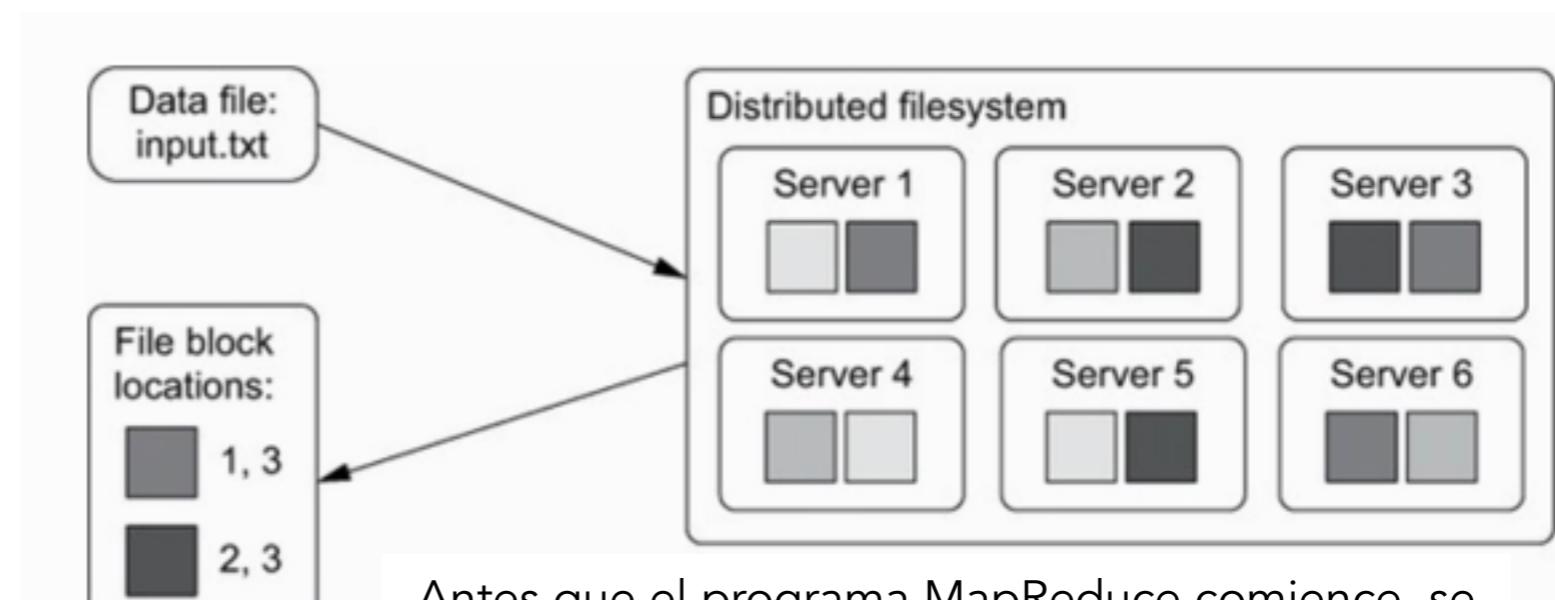
Cada record de pageview contiene URL y timestamp.

Objetivo: Determinar el número total de visitas a una página con una URL particular para un rango de tiempo dado en horas.

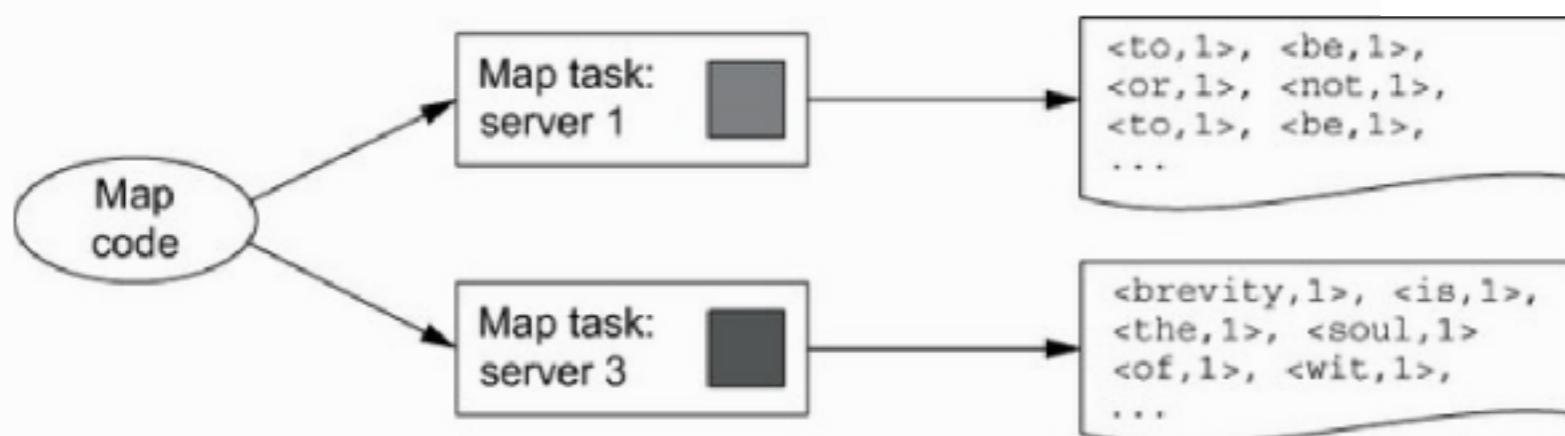
```
function map(record) {  
    key = [record.url, toHour(record.timestamp)]  
    emit(key, 1)  
}  
  
function reduce(key, vals) {  
    emit(new HourPageviews(key[0], key[1], sum(vals)))  
}
```



Ejemplo de conteo de palabras



Antes que el programa MapReduce comience, se determinan las localizaciones de los bloques en el sistema de archivos distribuido.



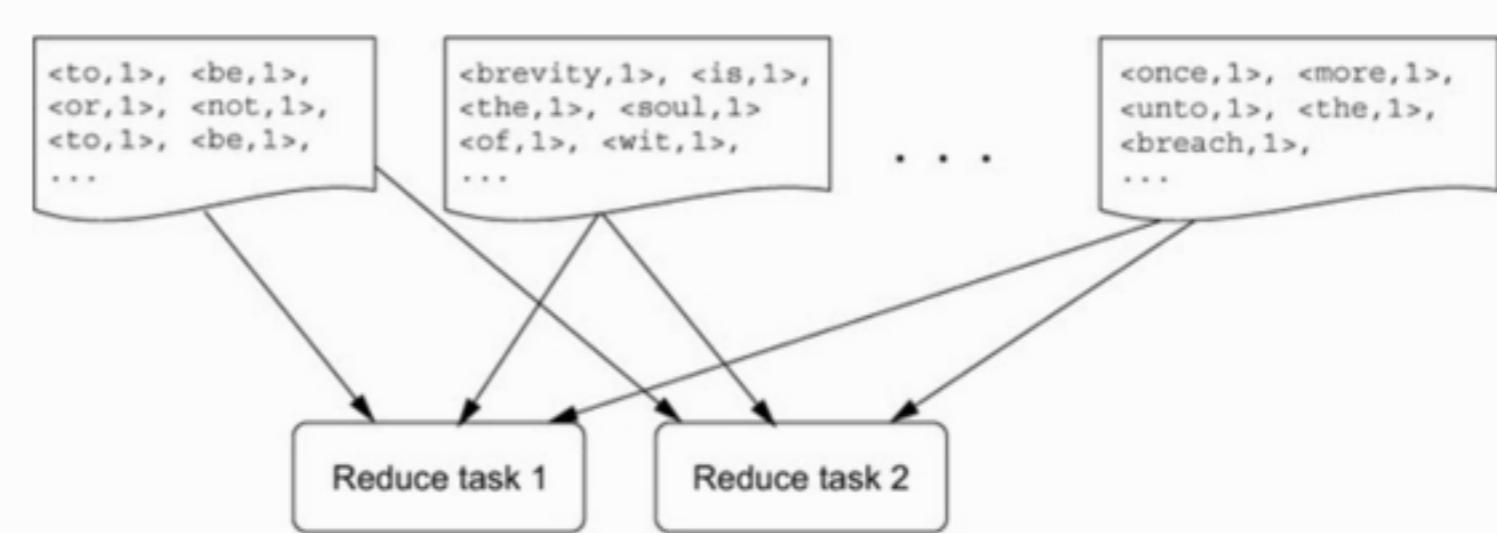
1. El código se envía a los servidores que alojan a los archivos de entrada para limitar el tráfico de red en el cluster.

2. Las tareas MAP generan pares clave/valor intermedios que son redirigidos a las tareas del reduce.

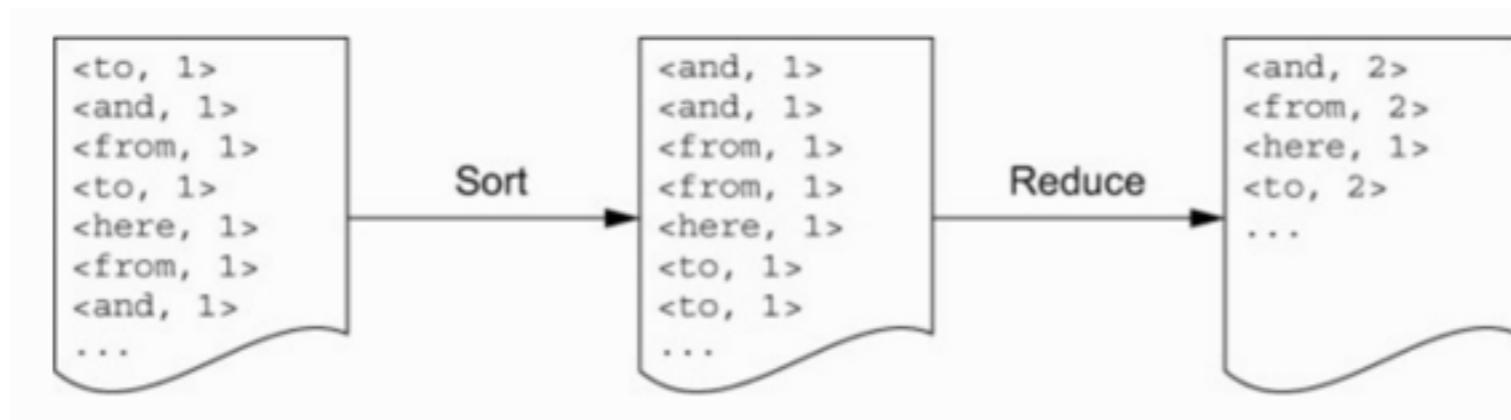


Las tareas de REDUCE no pueden comenzar hasta que todas las tareas MAP hayan terminado.

Ejemplo de conteo de palabras



Durante la etapa de shuffle, todos los pares clave/valor generados por las tareas MAP son distribuidas en las tareas reduce. En este proceso, todos los pares con la misma clave son enviados al mismo reducer.



Las tareas reduce ordena los datos de entrada por clave y luego realiza sus funciones en un el grupo de valores resultante.



- En nodo maestro hace ping a los trabajadores periódicamente.
- Las tareas se reinician al estado inicial ocioso.
- Las tareas son re-ejecutadas.
- El maestro escribe checkpoint de manera periódica para lidiar con fallas.
- Se planifica la ejecución de los trabajos de respaldo (stragglers)

Un straggler (rezagado) es una máquina que toma un tiempo inusual en completar sus tareas.

Caso de fallas

PARTICIÓN

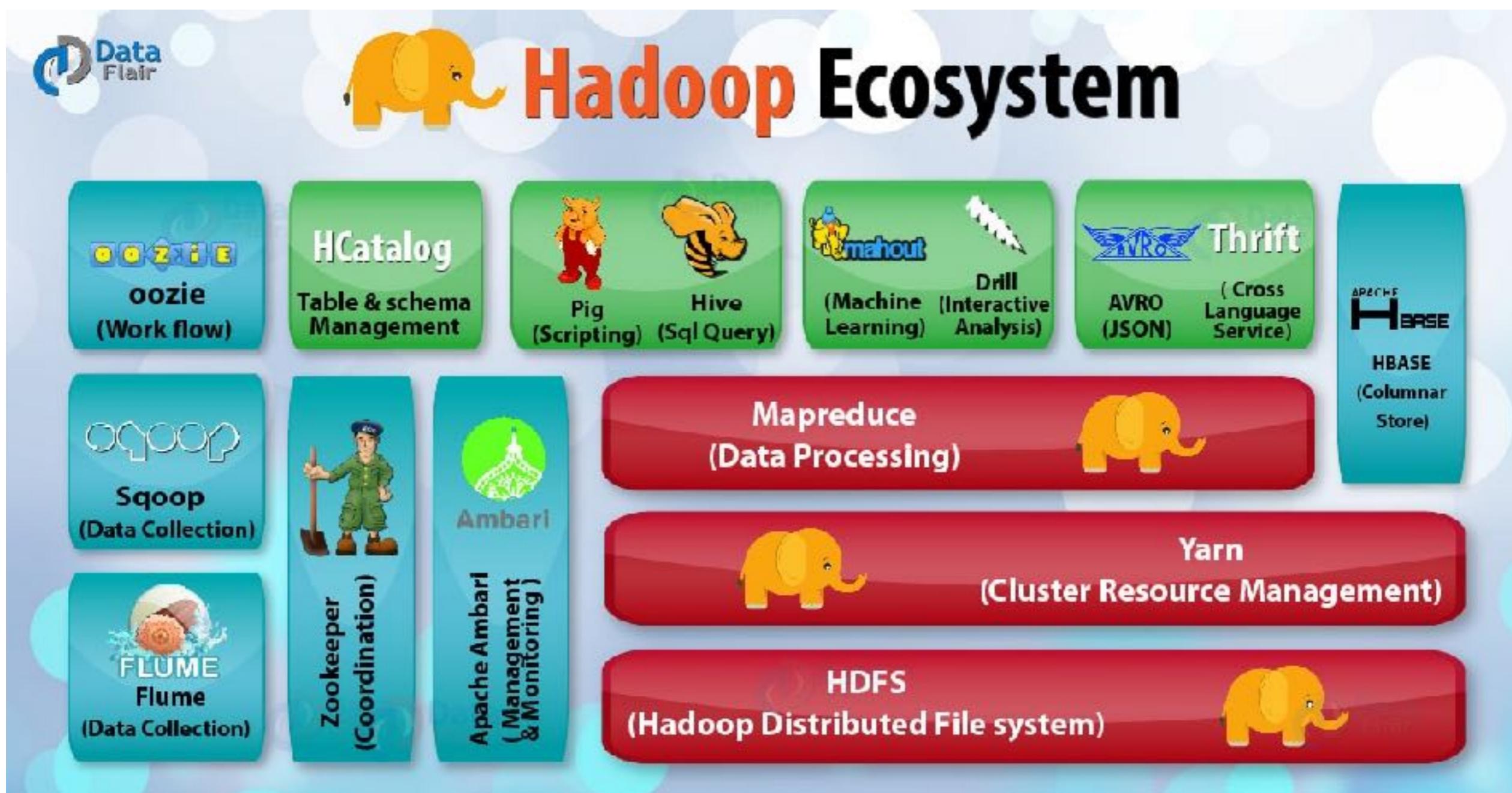
M map y R reducers. Más grande que el número de trabajadores.

Ejemplo: $M=200.000$ y $R=5.000$ usando 2.000 trabajadores.

Ejemplo partición: $\text{Hash(key)} \bmod R$

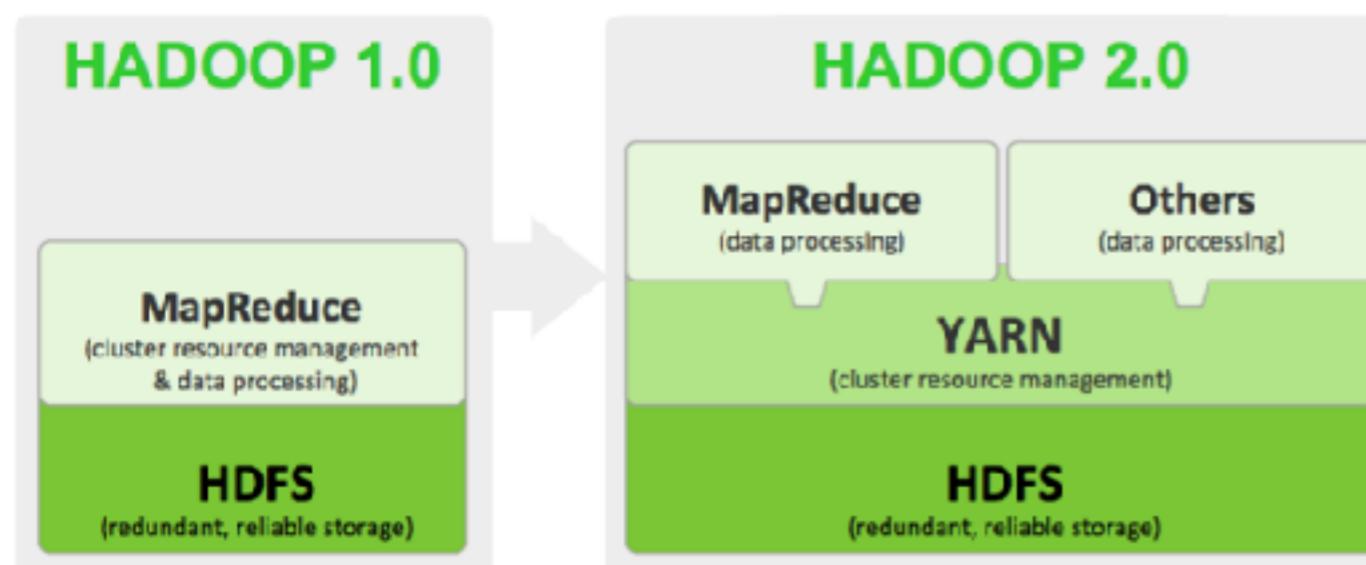


Ecosistema Hadoop





- Librería de software que permite procesar grandes conjuntos de datos a través de clusters de computadores usando modelos de programación simples.
- Diseñado para escalar desde un solo servidor hasta miles de máquinas, cada una con almacenamiento y computación
- Es escalable: Y! Tiene un cluster de 4000-nodos
- Gran poder de cómputo: ordenamiento de un TB de enteros aleatorios en 62 segundos.



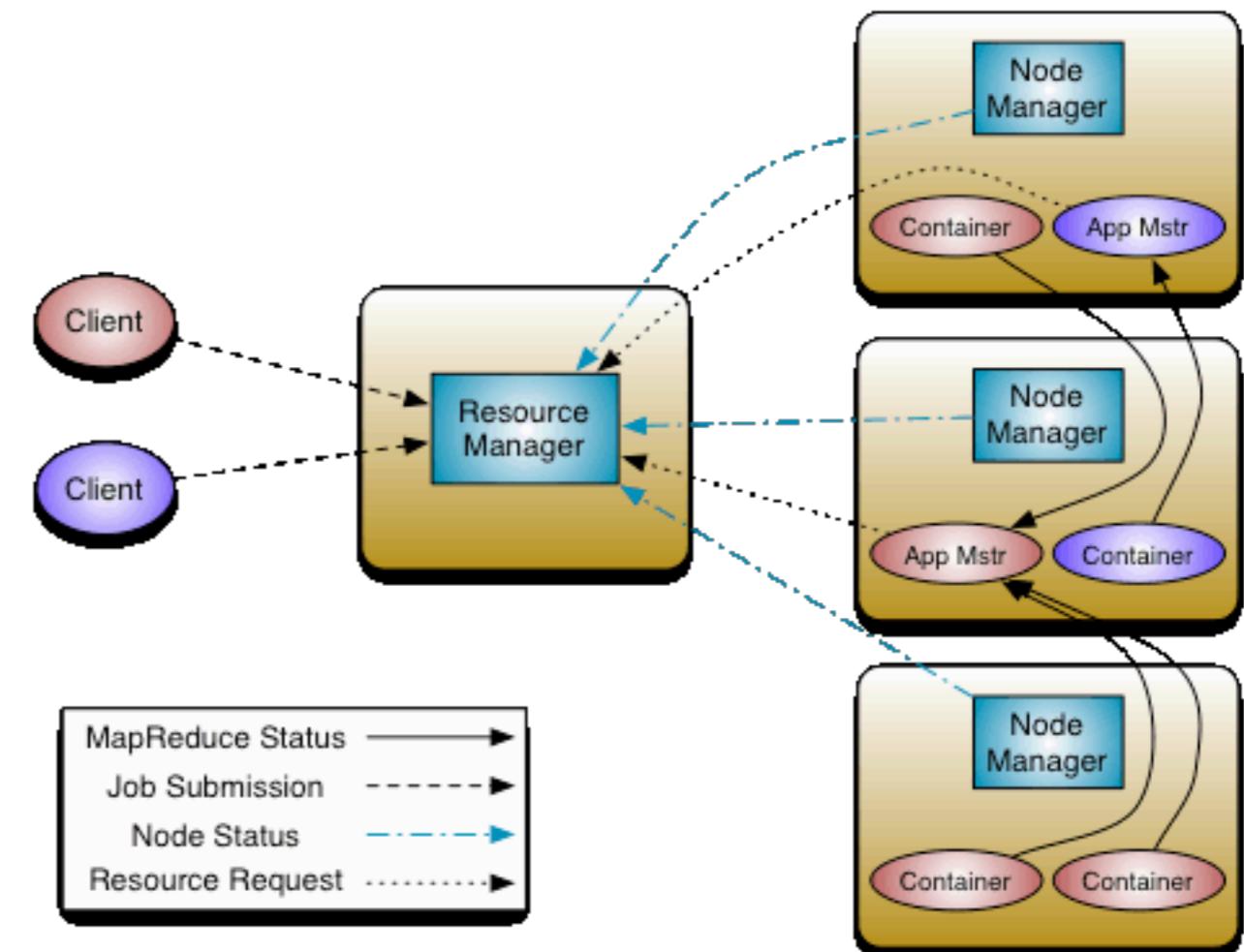
Hadoop 3.0 Introducer Erasure Coding sobre HDFS para evitar 200% adicional de uso de almacenamiento y pasar a 50% adicional.

Yarn: Manejador de recursos, similar a Mesos. Es el coordinador central de la planificación del cluster, cuidando que los trabajos sean planificados en el cluster de manera ordenada considerando:

- Restricción de recursos
- Estrategias de planificación
- Prioridades
- Justicia, etc.



- YARN se hace cargo del manejo del cluster.
- **Resource Manager:** Planificador y asignación global.
- Crea un **Application Master** por cada aplicación.
- En cada nodo hay un **Node Manager** para administrar los procesos en esa máquina.
- RM crea **contenedores** en cada nodo para ejecutar tareas de tipo map y reduce.





Casos de uso



<https://cwiki.apache.org/confluence/display/HADOOP2/PoweredBy>

- **Amazon:** Clusters vary from 1 to 100 nodes. They build Amazon's product search indices with Hadoop.
- **Adobe:** We use Apache Hadoop and Apache HBase in several areas from social services to structured data storage and processing for internal use. Clusters from 5 to 15 nodes + 30 with HDFS.
- **Alibaba:** A 15-node cluster dedicated to processing sorts of business data dumped out of database and joining them together. These data will then be fed into iSearch, our vertical search engine.
- **ATXcursions:** Sentiment analysis of review websites and social media (tourism industry). Marketing tool that analyzes the most valuable/useful reviewers from sites like Tripadvisor and Yelp as well as social media. Helps marketers and business owners find community members most relevant to their business.
- **Cloudspace:** Log reporting and parsing systems to scale. Log aggregation and



Casos de uso de **hadoop**

- **EBay:** Using it for Search optimization and Research.
- **ETH Zurich Systems Group:** Use Hadoop for analysis of biological data.
- **Eyealike:** Facial similarity and recognition across large datasets. Image content based advertisement and auto-tagging for social media. Image based video copyright protection.
- **Facebook:** We use Apache Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning. One of the clusters is a 1100 machine cluster with 8800 cores and 12 PB of raw storage.
- **FollowNews:** For storing logs, news analysis and tag analysis.



Casos de uso de **hadoop**

- **Last.fm**: 100 nodes clusters, used for charts calculation royalty reporting, log analysis, A/B testing, dataset merging. Also used for large scale audio feature analysis over millions of tracks.
- **Resu.me**: Use Hadoop to process user resume data and run algorithms for our recommendation engine.
- **Spotify**: We use Apache Hadoop for content generation, data aggregation, reporting, analysis
- **Twitter**: We use Apache Hadoop to store and process tweets, log files, and many other types of data generated across Twitter. We store all data as compressed LZO files.
- **Yahoo!**: More than 100,000 CPUs running Hadoop. Used to support research for Ad Systems and Web Search.



Ejemplo: Puntuación de Influencia

Dataset como Twitter que contiene registros de reacciones, que contienen los campos sourceID y el responderID. ResponderID hizo retweet o respondió a un post de sourceID.

Objetivo: Determinar el puntaje de influencer de cada persona en la red social

```
function influence_score(masterDataset, personId) {  
    influence = new Map()  
    for(record in masterDataset) {  
        curr = influence.get(record.responderId) || new Map(default=0)  
        curr[record.sourceId] += 1  
        influence.set(record.sourceId, curr)  
    }  
  
    score = 0  
    for(entry in influence) {  
        if(topKey(entry.value) == personId) {  
            score += 1  
        }  
    }  
    return score  
}
```

Calcula la cantidad de influencia entre todos los pares de personas

Cuenta el número de personas para las que personId es el top Influencer



Ejemplo: Puntuación de Influencia

```
function map1(record) {  
    emit(record.responderId, record.sourceId)  
}  
  
function reduc1(userid, sourceIds) {  
    influence = new Map(default=0)  
    for(sourceId in sourceIds) {  
        influence[sourceId] += 1  
    }  
    emit(topKey(influence))  
}  
  
function map2(record) {  
    emit(record, 1)  
}  
  
function reduce2(influencer, vals) {  
    emit(new InfluenceScore(influencer, sum(vals)))  
}
```

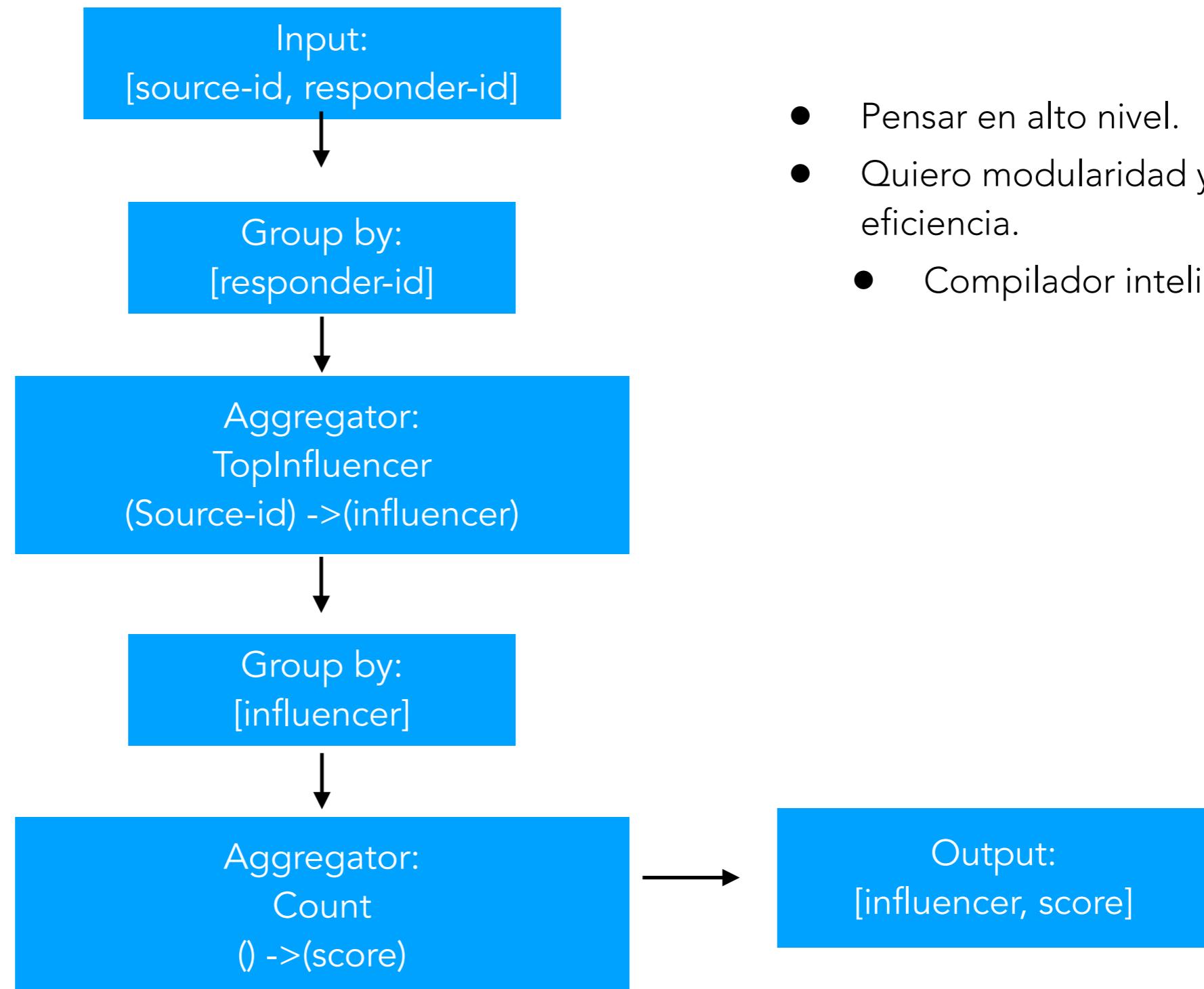
El primer trabajo determina el top influencer para cada usuario

Los datos de los top influencer para cada usuarios son usados para determinar el número de personas que cada usuario influencia.

Se requieren dos trabajos de MapReduce encadenados.



Diagrama Pipe del ejemplo



- Pensar en alto nivel.
- Quiero modularidad y eficiencia.
- Compilador inteligente



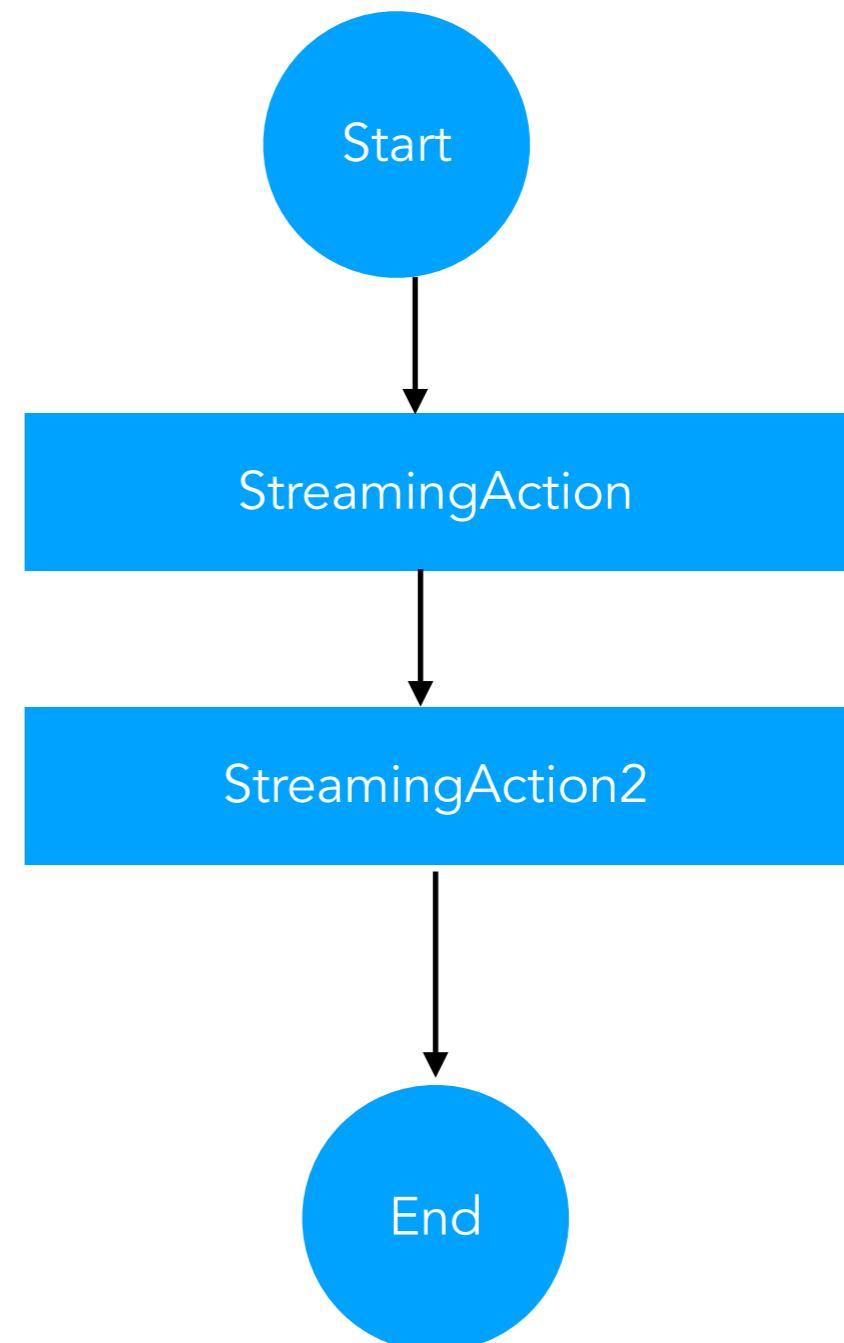
Workflows de MapReduce

- Los trabajos de MapReduce son encadenados en workflows.
- La salida de un trabajo es la entrada del siguiente trabajo.
- Cuando se construyen sistemas de recomendación, son comunes los Workflows de 50 a 100 trabajos MapReduce.
- Algunas herramientas de alto nivel para Hadoop, como Pig, Hive, Cascading, Crunch y FlumeJava, también configuran workflows de múltiples etapas de MapReduce que son automáticamente conectadas juntas.



Apache Oozie

- Encadenar trabajos de MapReduce para hacer análisis de datos.
- Planificador de flujos de trabajo que permite administrar los trabajos MapReduce.
- Se organizan como DAG (Grafos Dirigidos Acíclicos).
- Usa lenguaje basado en XML.
- Soporta Hadoop, HDFS, Pig, Java, Email, Shell, Hive, Sqoop, SSH, etc.
- Otros planificadores de workflows: Azkaban, Luigi, Airflow, Pinball, etc.





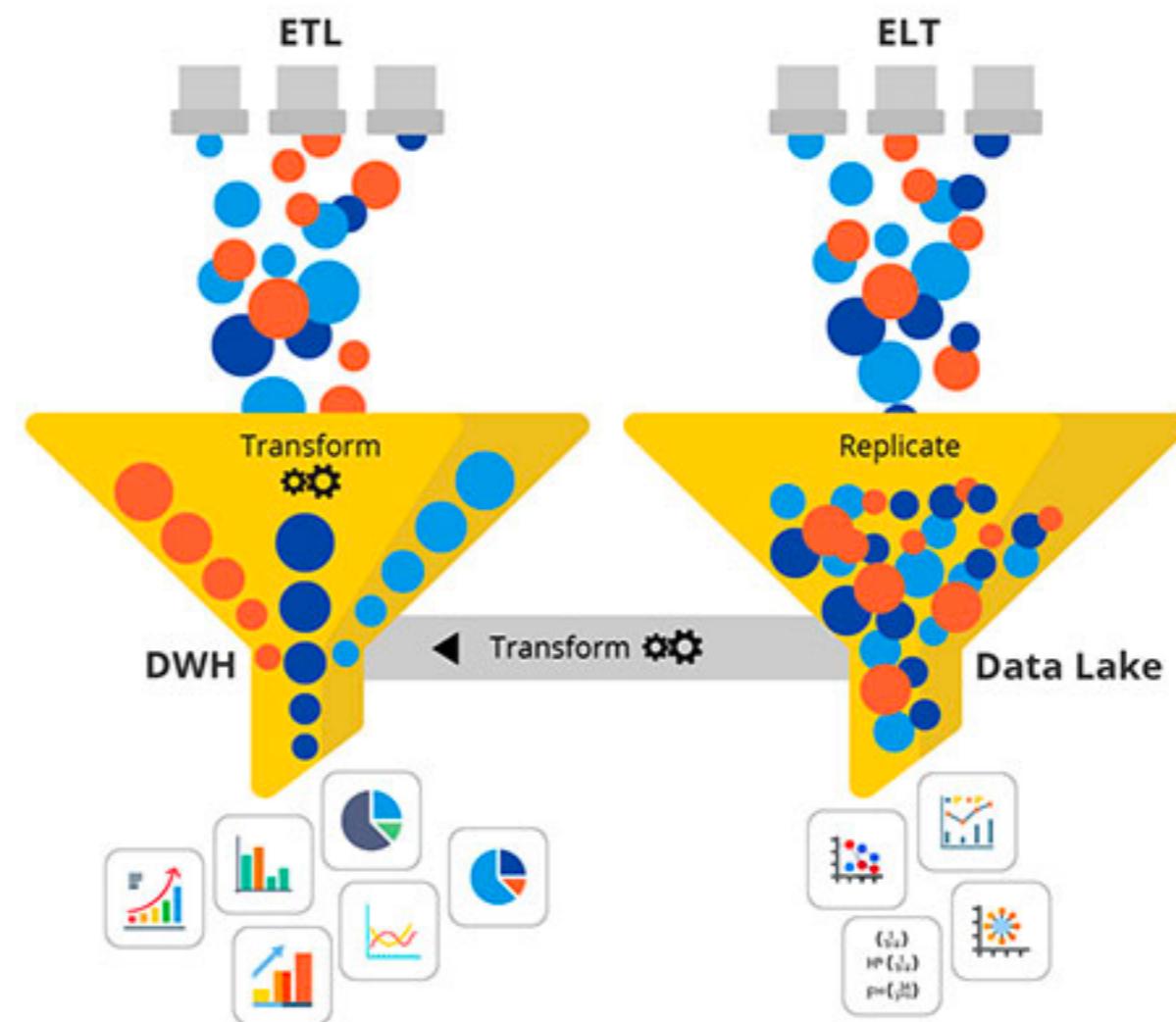
Conclusiones MapReduce

- Los programas MapReduce se ejecutan de forma completamente distribuida sin punto central de contención.
- MapReduce es escalable: las funciones map y reduce que se proveen son ejecutadas en paralelo en un cluster.
- El usuario no debe lidiar con los desafíos de concurrencia y asignar tareas a máquinas.
- Las computaciones MapReduce son tolerantes a fallas.
 - Las tareas deben ser deterministas.
- Compiladores y lenguajes de alto nivel facilitan el trabajo con MapReduce.
- Computación en batch es de alta latencia por naturaleza.
 - Se puede aprovechar de hacer análisis profundos y cómputos caros.
- Casos de uso: ETL (leer en bruto, transformar en forma relacional), search indexes, sistemas de recomendación, construir sistemas de máquinas de aprendizaje como clasificadores, etc.



Batch pipelines

- **ETL Extract - Transform - Load:**
Proceso de ingestión de datos de distintas fuentes, transformarlos y consolidarlos en un sistema lógicamente centralizado como un DWH.
- **ELT Extract - Load - Transform:** Los datos son almacenados directamente en destino (DataLake) en bruto y luego es transformada cuando se necesite.
- **EL Extract - Load :** Se refiere a importar datos tal cual están al sistema.

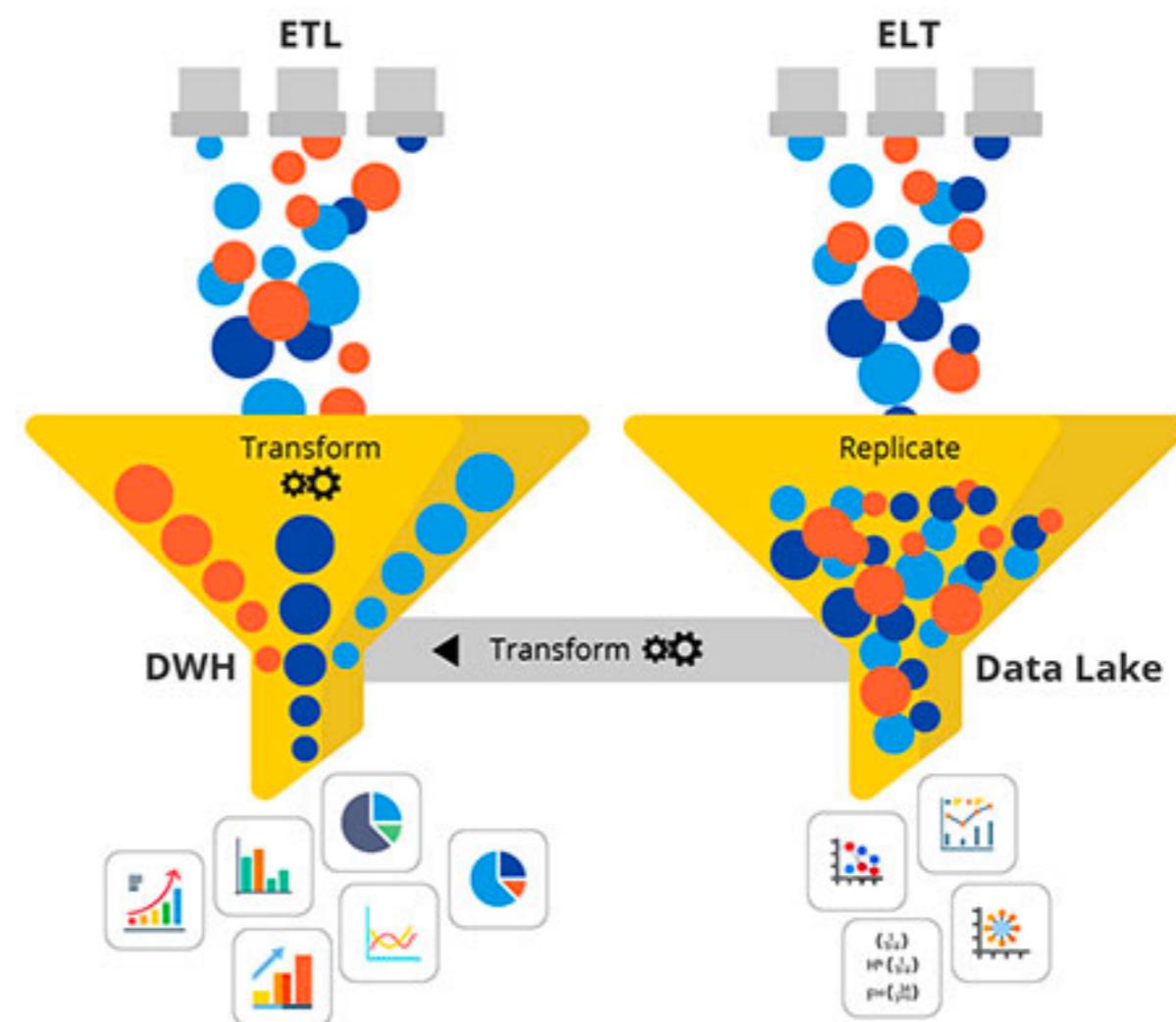


<https://www.heavy.ai/technical-glossary/etl-vs-elt>



Batch pipelines

- **ETL Extract - Transform - Load:**
Cuando no voy a ocupar los datos brutos para análisis posterior.
- **ELT Extract - Load - Transform:** Se usa si no estás seguro que transformaciones serán necesarias. Más usado para dar posibilidades a Data Science y ML.
- **EL Extract - Load :** Se usa cuando los datos ya están limpios y correctos.



<https://www.heavy.ai/technical-glossary/etl-vs-elt>



- ¿ETL, ELT?

Ejemplo ETL: Extraer datos de una fuente que requiere validación o anonimización (quitar nombre, edad, sexo, localización), se almacenan datos sin esto.

Ejemplo: Tweets -> Anonimizar con Amazon Kinesis -> Almacenar en BD

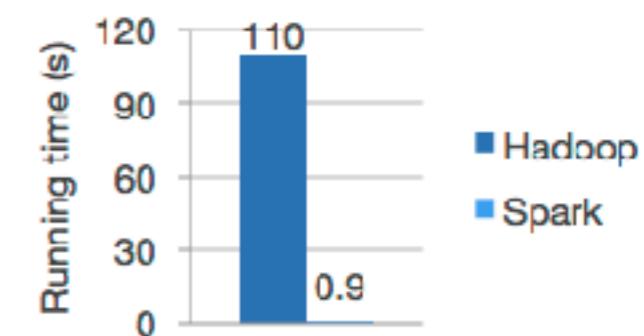
Ejemplo ELT: Extraer datos de varias fuentes y almacenar en Data Lake como Amazon S3. Transformación puede ser hecha después y almacenada en data warehouse para reporte.

EJEMPLO: LOGS -> BIG QUERY -> Transformar con BQ y crear una nueva tabla.



Framework de código abierto que provee métodos de análisis para el procesamiento de análisis a gran escala.

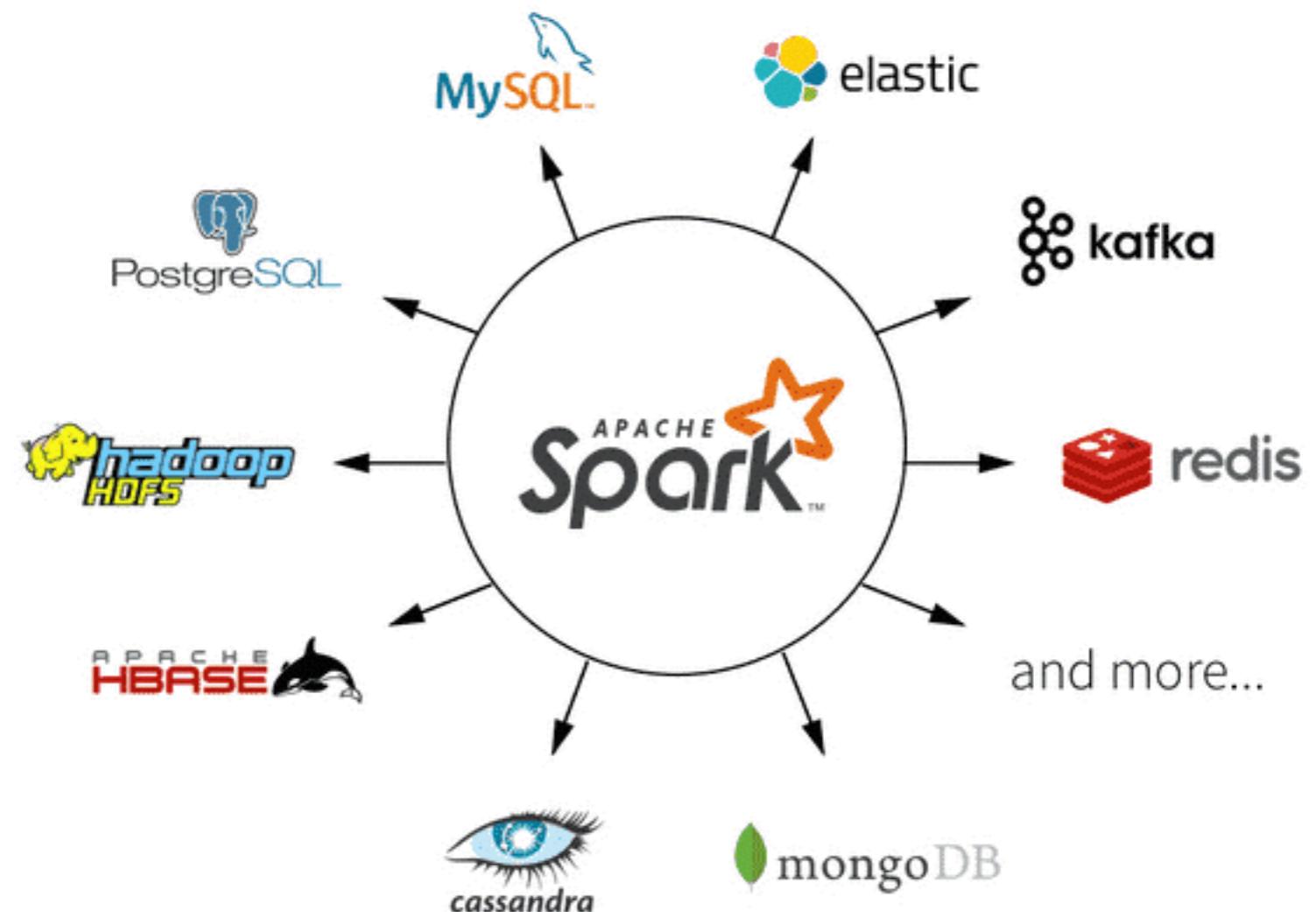
- Alto rendimiento, dice ser 100 veces más rápido de Hadoop en memoria o 10 veces más rápido en disco.
- Soporta lenguajes como Java, Scala, Python, R, y SQL.
- Apache Spark es 100% open source.
- Idea clave: Procesamiento en memoria y optimización de ejecución de trabajos.
- APIs fáciles de usar, más de 100 operadores para transformar datos.



Logistic regression in Hadoop and Spark



- Viene empaquetado con librerías de alto nivel e integración con diferentes servicios usados en Big Data.

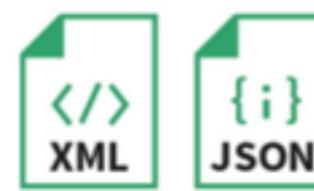


UNSTRUCTURED



More flexible

SEMI-STRUCTURED



More efficient storage and performance

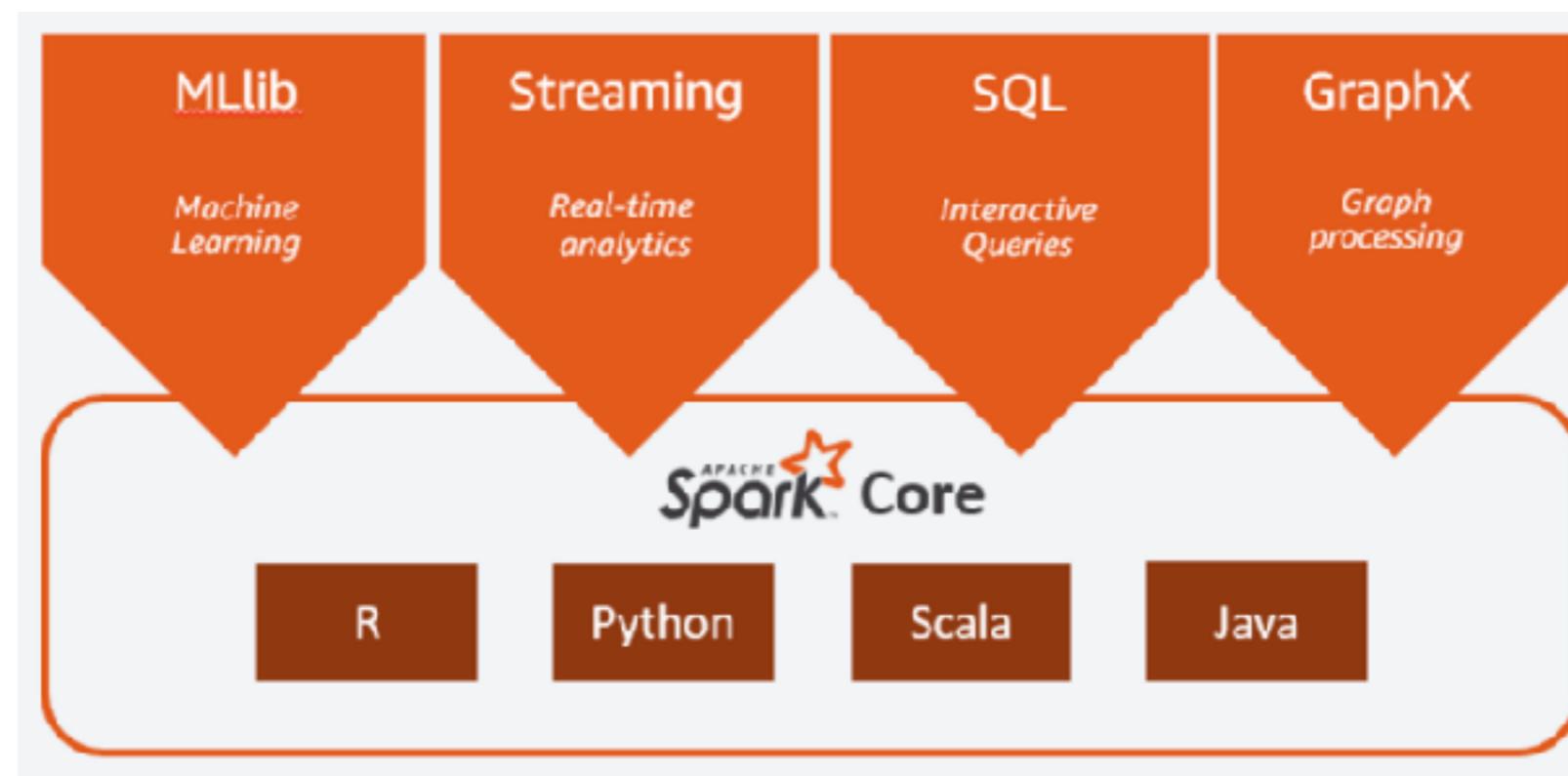
STRUCTURED





Herramientas Spark

- Soporte de varias herramientas de alto nivel: Combina SQL, streaming y analítica compleja.
- **Spark Streaming:** componente para el análisis de stream de datos.
- **Spark SQL:** Permite consultas interactivas de datos usando consultas SQL.
- **Spark MLib:** Provee implementaciones de algoritmos de machine learning comúnmente usados como clustering, clasificación, regresión, filtrado colaborativo, reducción de dimensionalidad, etc.
- **Spark GraphX:** Provee implementaciones de algoritmos para grafos como PageRank.



<https://aws.amazon.com/big-data/what-is-spark/>



Spark Powered By

[Databricks](#): Formed by the original creators of Apache Spark, Databricks is working to expand the open source project and simplify big data and machine learning. We're deeply committed to keeping all our work on Spark open source.

- We provided a cloud-optimized platform to run Spark and ML applications on Amazon Web Services and Azure, as well as a comprehensive training program.

[Data Mechanics](#): Data Mechanics is a cloud-native Spark platform that can be deployed on a Kubernetes cluster inside its customers AWS, GCP, or Azure cloud environments.

- Our focus is to make Spark easy-to-use and cost-effective for data engineering workloads. We also develop the free, cross-platform, and partially open-source Spark monitoring tool Data Mechanics Delight.

[eBay Inc.](#): Using Spark core for log transaction aggregation and analytics

[Faimdata](#): Ecommerce and data intelligence solutions to retail industry on top of Spark and Spark Streaming.

[Flytxt](#): Big Data analytics for subscriber profiling and personalization in telecommunications domain.

[MyFitnessPal](#): use to clean-up user entered food data using explicit and implicit user signals to identify high-quality food items.

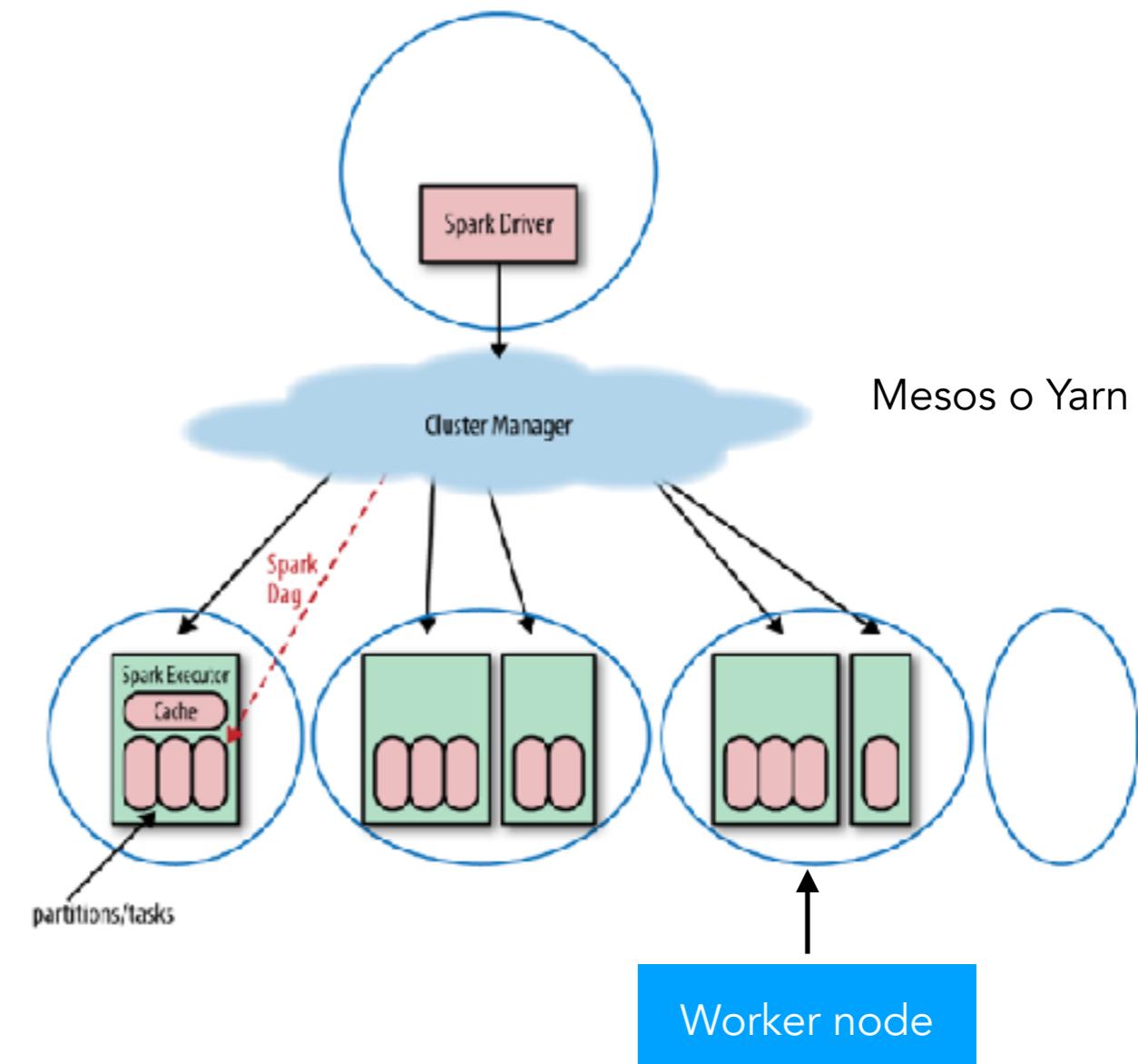
[Netflix, Amazon, Groupon, TripAdvisor, Yahoo!, IBM, Intel, Cisco, Oracle, Spotify, etc.](#)

<https://spark.apache.org/powerd-by.html>



Spark Application

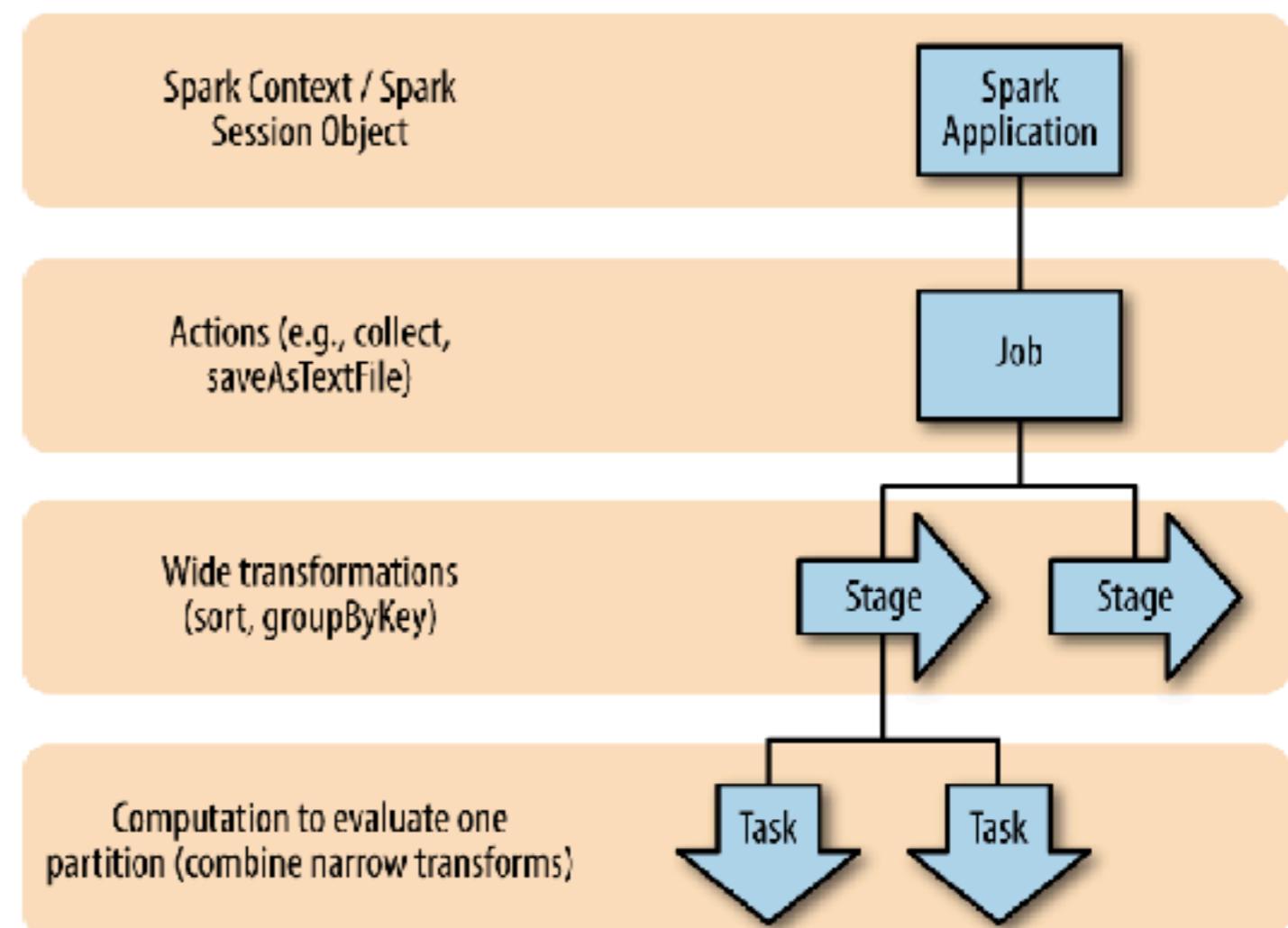
- La app de spark corresponde a un conjunto de trabajos spark definidos en el programa Driver.
- El *SparkContext* es un conjunto de parámetros de configuración para correr los trabajos spark.
- Cuando se inicia Spark Context, un driver y un conjunto de “executors” comienzan en los nodos del cluster.
- Cada executor tiene su propia JVM.
- El *SparkContext* determine cuántos recursos son asignados a cada executor.





Anatomía de un trabajo Spark

- Con cada acción el Spark Scheduler construye un grafo de ejecución y lanza un trabajo de Spark.
- Cada trabajo consiste de Stages, las que son pasos en las transformaciones de datos que se requieren para materializar el RDD final.
- Cada Stage consiste en un conjunto de tareas que representa cada computación paralela y que es realizada por los ejecutores.





Versiones de Spark

- Antes de Spark 2.0: Interfaz de programación era **RDD**: Resilient Distributed Datasets
- Spark 2.0: RDDs API son reemplazados por **Dataframes** y **Dataset**, mejor rendimiento que RDD.
- Antes de Spark 2.0: Streaming se hacía en **Spark Streaming**.
- Spark 2.0: Hay dos tipos de streaming **Spark Structured Streaming** y **Continuous Streaming (experimental)**.



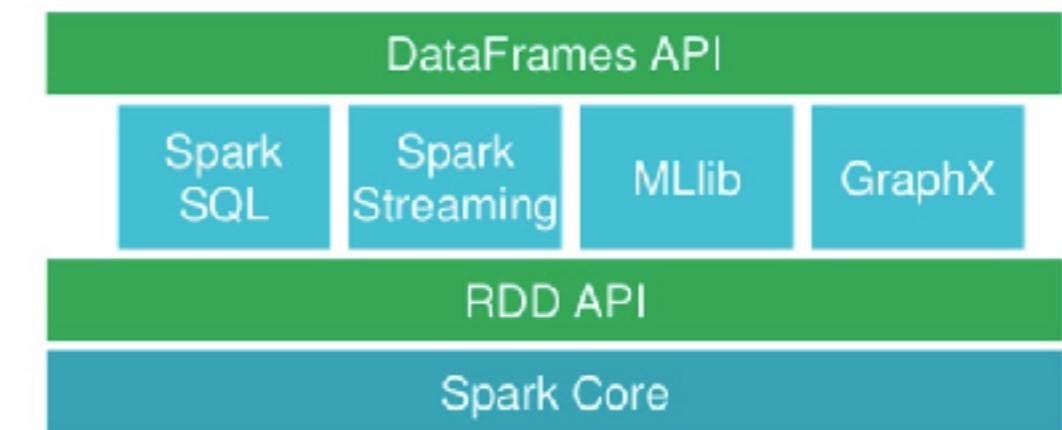
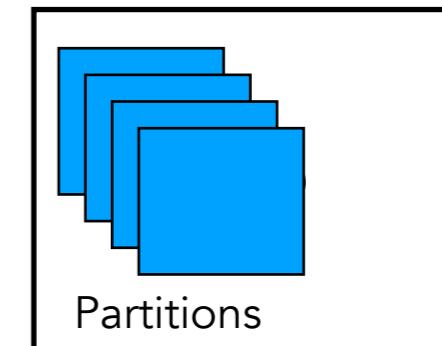


RDD: Resilient Distributed Dataset

- RDDs son conjunto de registros distribuidos de solo lectura.
- Pueden crearse a partir de colecciones existentes o cargando un dataset externo:

```
#crear RDD desde archivo local
lines= sc.textFile("file:///home/spark/archivo.md")
#Crear RDD paralelizando colecciones existentes
data = sc.parallelize([1,2,2,3,3,4,5])
```

- Son inmutables -> se pueden crear nuevos a partir de otros.
- Características:
 - Distribuidos: Los datos de un RDD puede ser localizada en múltiples nodos.
 - Resiliente: RDD hace seguimiento al linaje del RDD para recuperar en caso de fallas.
 - Evaluaciones "Lazy": Solo cuando se requiera materializar los datos, se realiza la computación.





Sobre RDDs y computación

“Lazy”

RDDs soportan dos tipos de operaciones Transformaciones y Acciones:

- **Transformaciones** crean nuevos RDD desde los existentes. Ejemplos:

- Errors = rdd.filter(lambda line: "ERROR" in line)
- lineLengths = line.map(lambda s: len(s))
lineLengths.take(5)
[14, 0, 78, 72, 73]
- count = words.reduceByKey(lambda a,b:a+b)
counts.take(3)
[(u'all',1), (u'when',1), (u'including',3)]

- Las transformaciones son “lazy”, no se procesan hasta que una acción requiere que el resultado sea retornado al programa Driver.
- La API de Spark encadena transformaciones y acciones.



Sobre RDDs y computación

“Lazy”

RDDs soportan dos tipos de operaciones Transformaciones y Acciones:

- **Acciones** materializan un valor en un programa del usuario. Ejemplos:
 - `lines.count()`
 - `data.saveAsTextFile('/path/file')`
 - `lineLengths = line.map(lambda s: len(s))`
`totalLength = lineLengths.reduce(lambda a,b: a +b)`



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

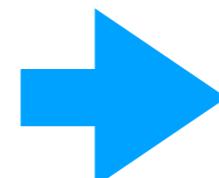
Departamento de Informática

```
// Read input file  
val input = sc.textFile("input.txt")
```

```
val tokenized = input  
.map(line => line.split(" "))  
.filter(words => words.size > 0) // remove empty lines
```

```
val counts = tokenized // frequency of log levels  
.map(words => (words(0), 1)).  
.reduceByKey{ (a, b) => a + b, 2 }
```

input.txt
INFO Server started
INFO Bound to port 8080
WARN Cannot find srv.conf



```
// Read input file  
val input = sc.textFile( )
```

```
val tokenized = input  
.map( )  
.filter( )
```

```
val counts = tokenized  
.map( )  
.reduceByKey{ }.
```

sc.textFile().map().filter().map().reduceByKey()

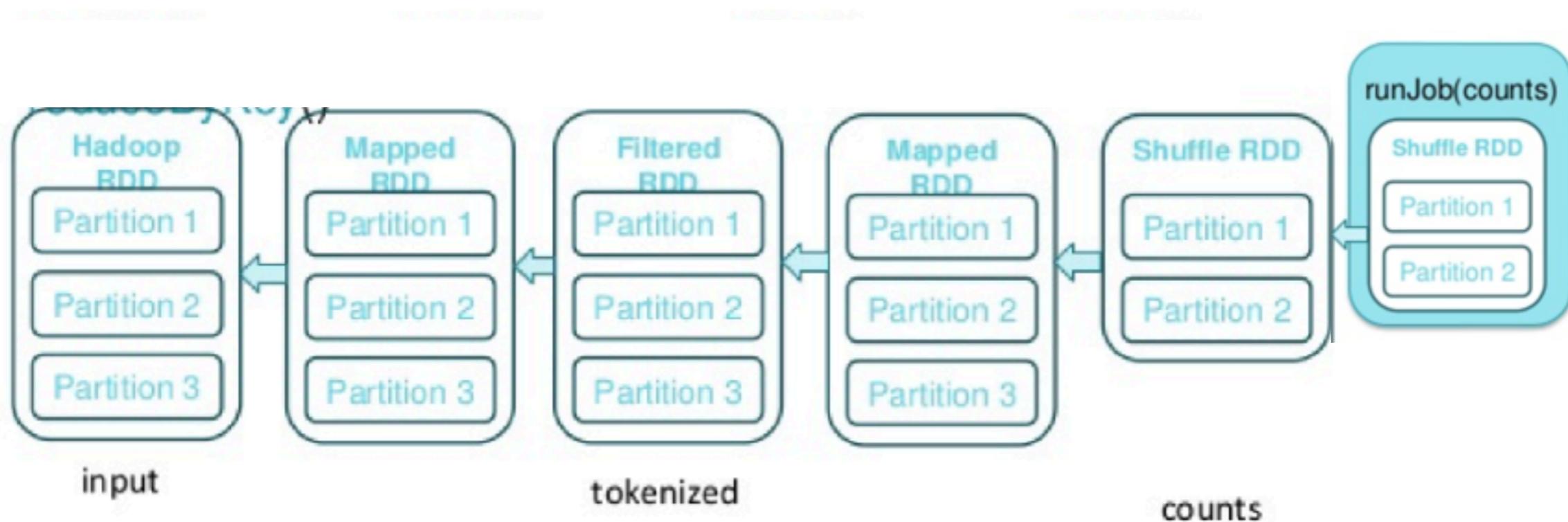
Transformaciones



Sobre RDDs y computación “Lazy”

sc.textFile().map().filter().map().reduceByKey()

Transformaciones



DAG's are materialized though a method sc.runJob

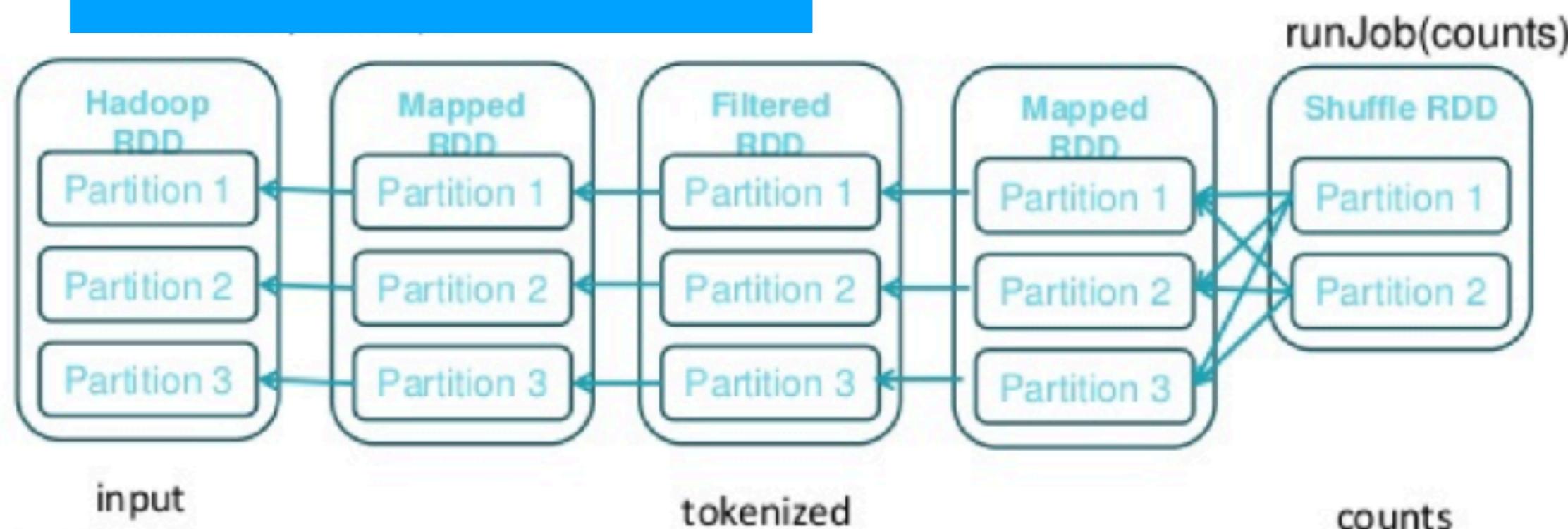


Sobre RDDs y computación

“Lazy”

Narrow transformations can be pipelined
into a single Stage

Wide transformations are the boundaries
between stages, require data across
partitions





Ejemplo conteo de palabras

■ Box 7.19: Apache Spark Python program for computing word count

```
from operator import add
from pyspark import SparkContext

sc = SparkContext(appName="WordCountApp")
lines = sc.textFile("file.txt")
counts = lines.flatMap(lambda x:
x.split(' ')).map(lambda x: (x, 1)).reduceByKey(add)

output = counts.collect()
for (word, count) in output:
    print "%s: %i" % (word, count)
```

Transformaciones

Acción

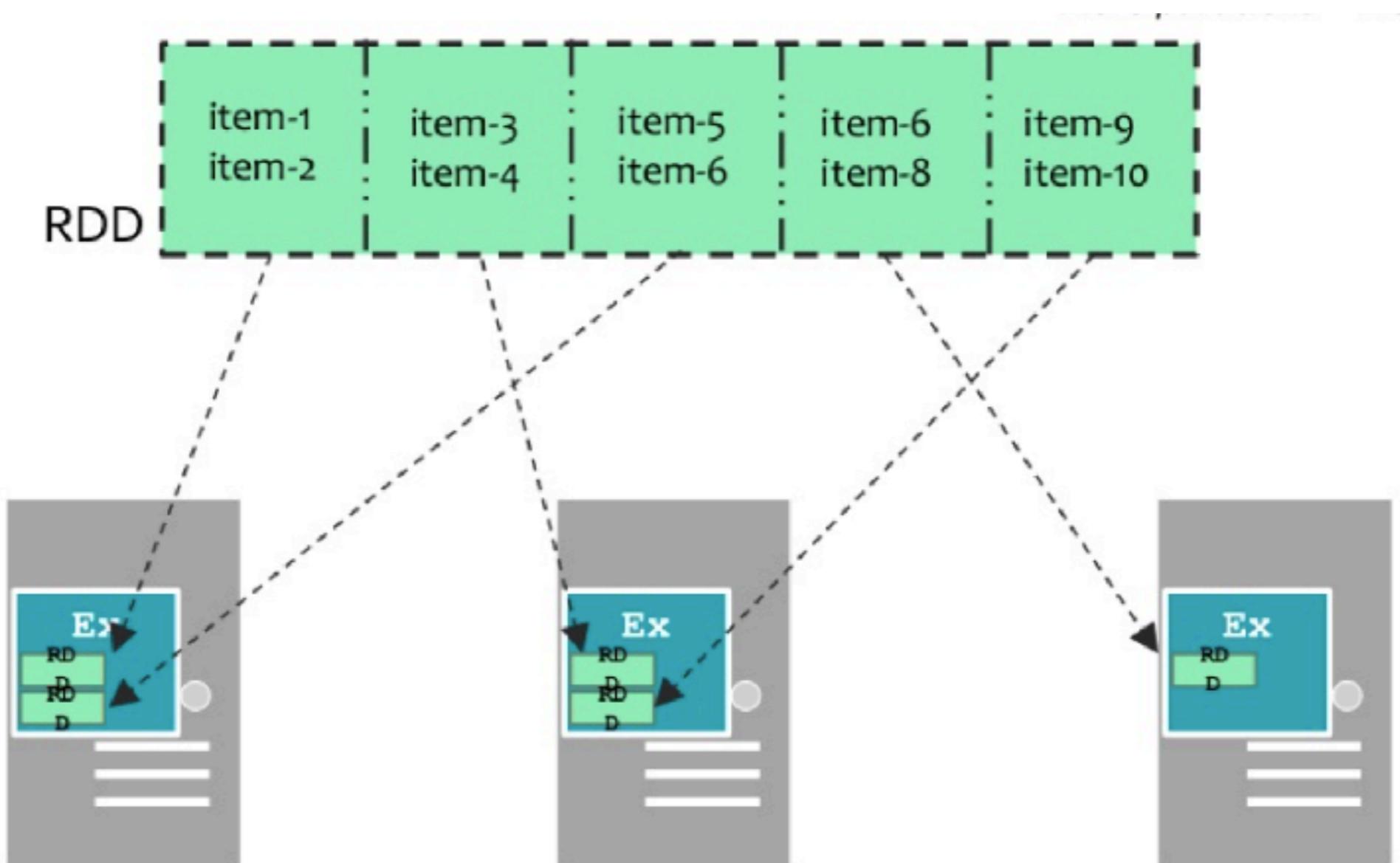
Fuente: Big Data Science and Analytics Book



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática

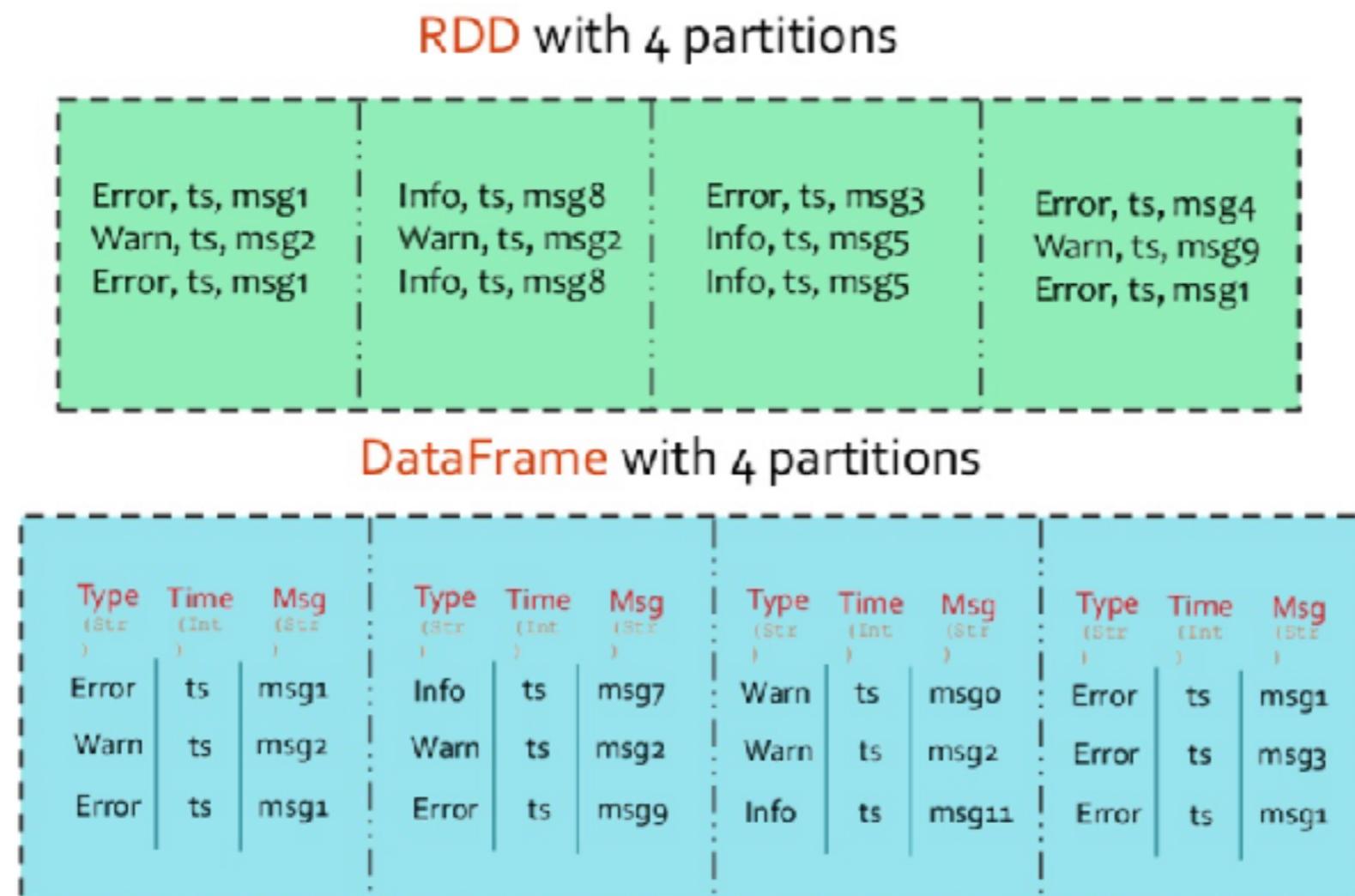
Modelo de datos de Spark





Dataset y Dataframe

- Dataframe: Colección de datos, datos son **organizados en columnas con nombre**, como una tabla en una base de datos relacional.





Ejemplo manejo Dataframe

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# spark is an existing SparkSession
df = spark.read.json("examples/src/main/resources/people.json")
# Displays the content of the DataFrame to stdout
df.show()
# +---+---+
# | age| name|
# +---+---+
# | null|Michael|
# | 30| Andy|
# | 19| Justin|
# +---+---+
```

```
# spark, df are from the previous example
# Print the schema in a tree format
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
# +---+
# | name|
# +---+
# | Michael|
# | Andy|
# | Justin|
# +---+

# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +---+---+
# | name|(age + 1)|
# +---+---+
# | Michael| null|
# | Andy| 31|
# | Justin| 20|
# +---+---+
```

```
# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+
# | age|name|
# +---+
# | 30|Andy|
# +---+

# Count people by age
df.groupBy("age").count().show()
# +---+---+
# | age|count|
# +---+---+
# | 19| 1|
# | null| 1|
# | 30| 1|
# +---+---+
```



Procesamiento de Stream

Streaming: tipo de motor de procesamiento de datos que está diseñado considerando conjuntos de datos infinitos.

- En el **procesamiento de batch** la entrada es acotada, tiene un tamaño finito conocido.
- **Datos no acotados/ilimitados:** los datos llegan gradualmente en el tiempo, el dataset nunca está completo.
- En **procesamiento de stream:** el enfoque es procesar cada evento a medida que sucede.
- Porqué usarlo en Big Data?
 - Alcanza latencias bajas.
 - Los datasets no acotados y masivos son cada vez más comunes.
 - Procesar los datos cuando se general reparte la carga de forma más equitativa en el tiempo, usando recursos de manera más predecible y consistente.



Procesamiento de Stream

Stream de datos

Es una secuencia ordenada y continua de items en tiempo real.

- Se procesan datos “on-the-fly”
- Altas tasas de entrada de datos
- La latencia importa (segundos a milisegundos)
- Un registro es conocido como un evento: objeto pequeño, auto-contenido e inmutable sobre algo que pasó.
- Usualmente contiene un timestamp para indicar cuándo sucedió.

Evento

Evento

Evento

Evento

Evento

Evento

Evento

Ejemplo: texto, JSON, forma binaria, etc.



Procesando streams

- Un trozo de código que procesa streams: Operador o trabajo (job)
- Un operador consume streams de entrada en una forma de solo lectura y escribe su salida en una localización diferente de una forma de solo append.
- Ejemplos de uso: Detección de fraudes, analizar cambio de precios en mercados financieros, monitorear sistemas de fabricación, seguimiento de actividades militares y sistemas inteligentes, etc.





Procesando streams

Un evento a la vez
Caso de estudio: Storm

- Menor latencia
- Semántica at-least-once
- Semántica exactly-once en algunos casos
- Modelo de programación simple

VS

Micro-batch
Caso de estudio: Spark Streaming

- Mayor rendimiento (número de eventos procesados por período de tiempo)
- Semántica At-least-once y exactly-once

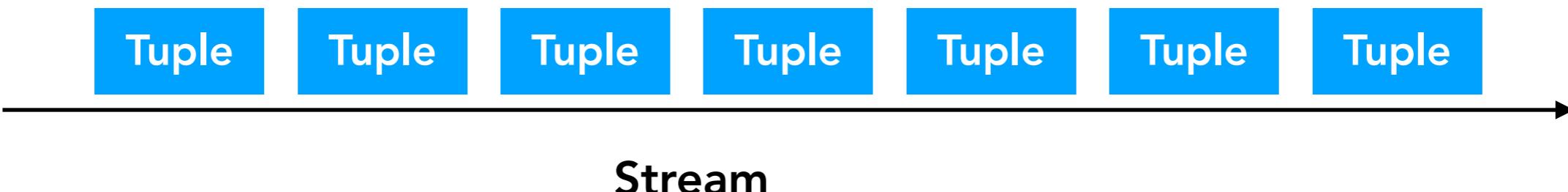
Otros ejemplos: Flink, Kafka Streams, Samza

Otros ejemplos: Storm-Trident



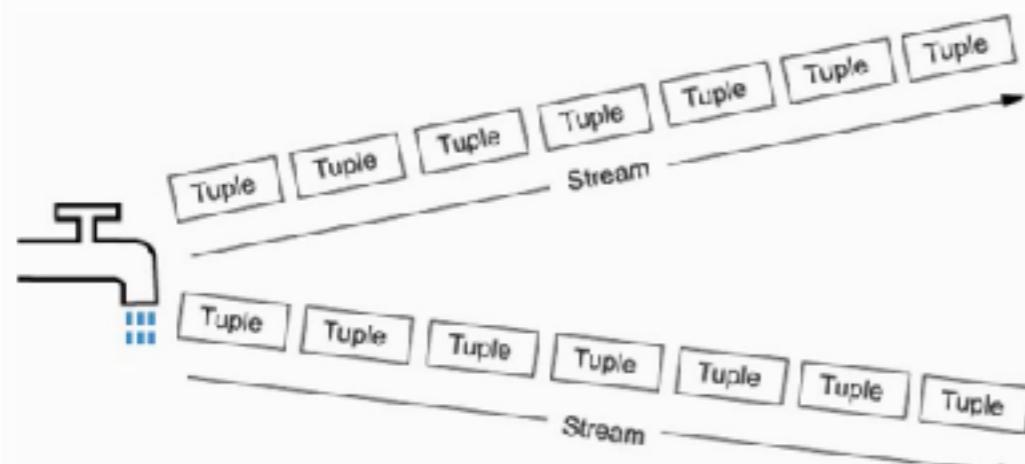
Storm es un sistema de procesamiento de stream de datos distribuido, confiable y tolerante a fallas.

- Topologías: La lógica de una aplicación es un pipeline de procesamiento que se representa con un grafo.
- Un programa se despliega en un clusters.
- Streams: Es una abstracción importante de Storm. Secuencia de tuplas ilimitadas que es procesada en paralelo de una manera distribuida.



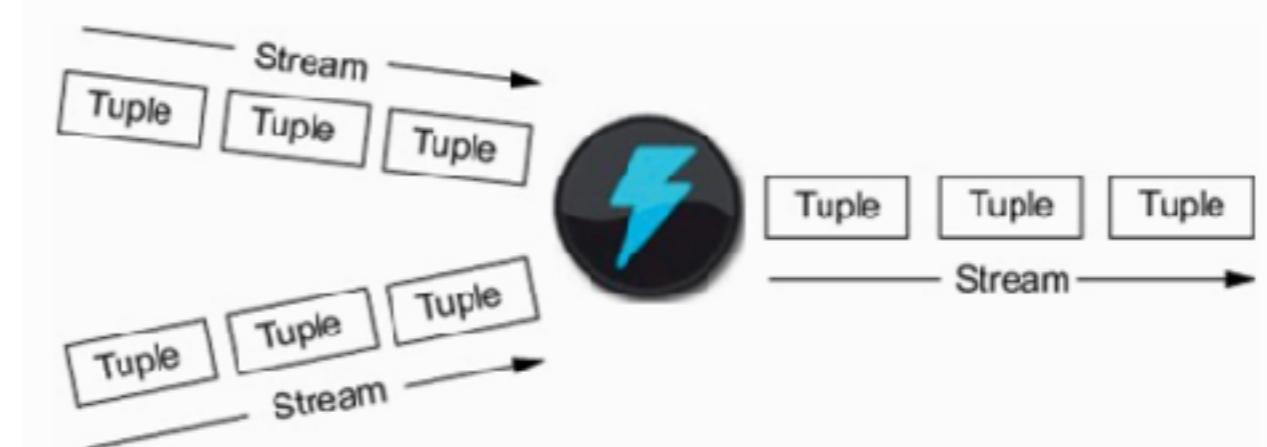


Spouts: Una fuente de streams en una topología.



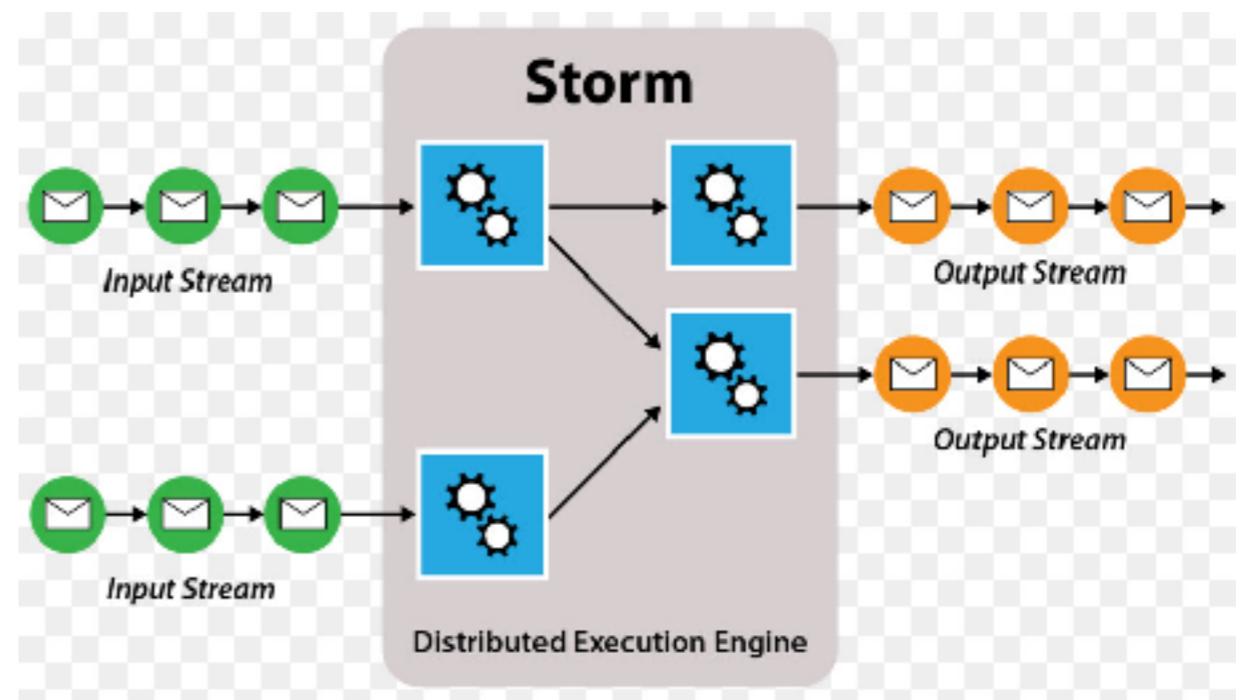
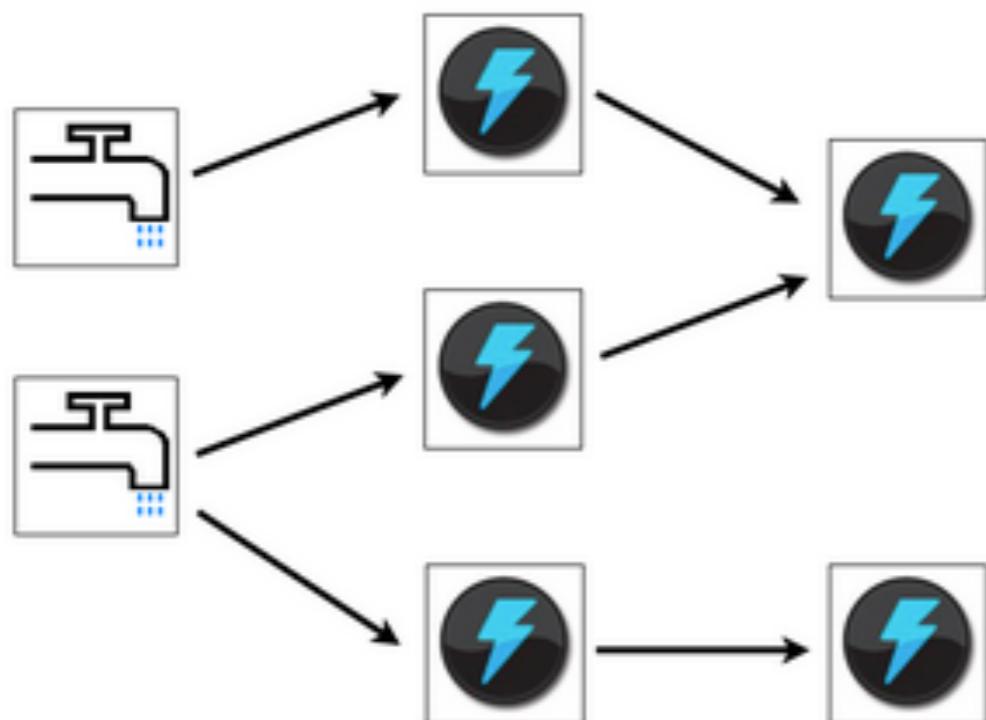
Bolts: Todo el procesamiento en la topología se hace en bolts.

- Ejecuta funciones, filtra datos, calcula agregaciones, joins, actualiza bases de datos, etc.
- Los bolts procesan la entrada de una o más fuentes y producen uno o más streams de salida.

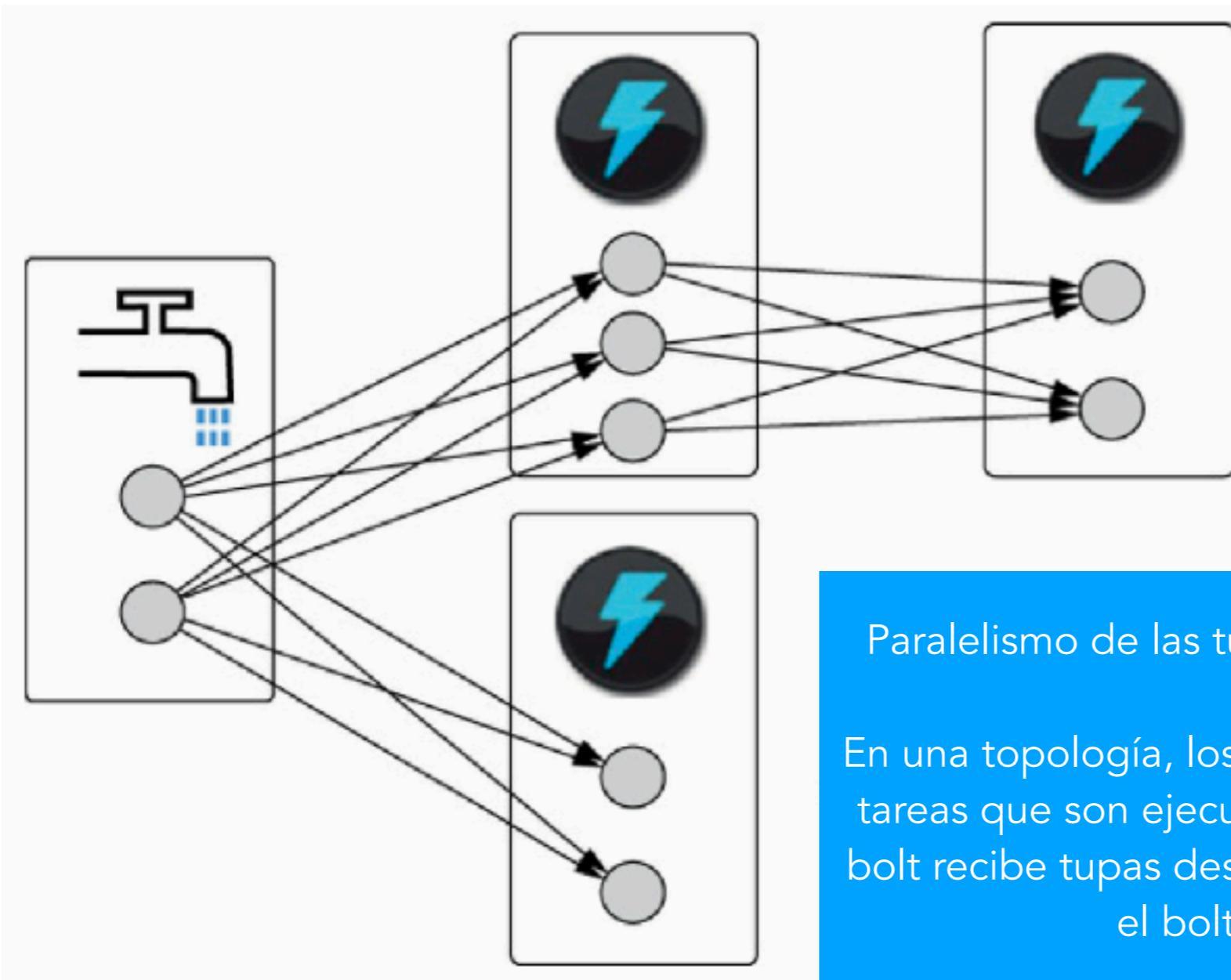




- Una **topología** conecta spouts y bolts y define como las tuplas fluyen a través de la aplicación de Storm.



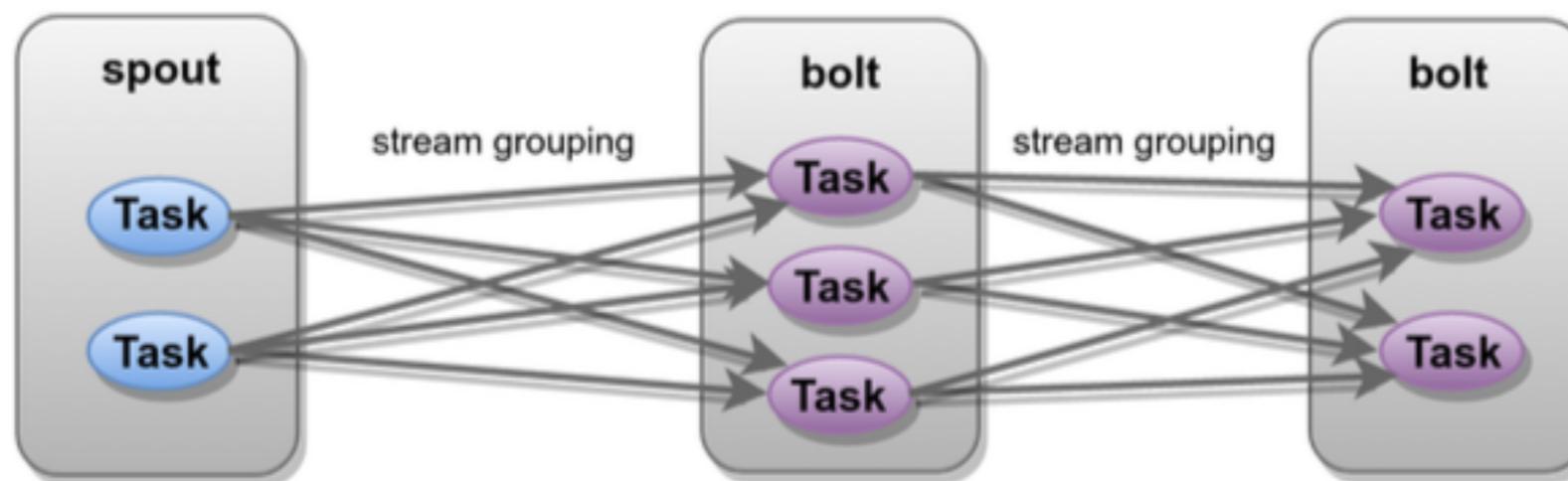
- Cada instancia de un spout o bolt se llama tarea.





Cuando una tarea emite una tupla, ¿cuál de las tareas del consumidor la recibirá?

- **Stream grouping:** Especifica como las tuplas deben ser particionadas entre las tareas consumidoras. Ejemplos:
 - Shuffle grouping: Random round robin
 - Fields grouping : hashing de un subconjunto de campos de la tuple y saca el módulo del resultado con el número de las tareas consumidoras.
 - All grouping: Los stream son replicados en todas las tareas del bolt.
 - Global grouping: Todo el stream va a un sola tarea de bolt.

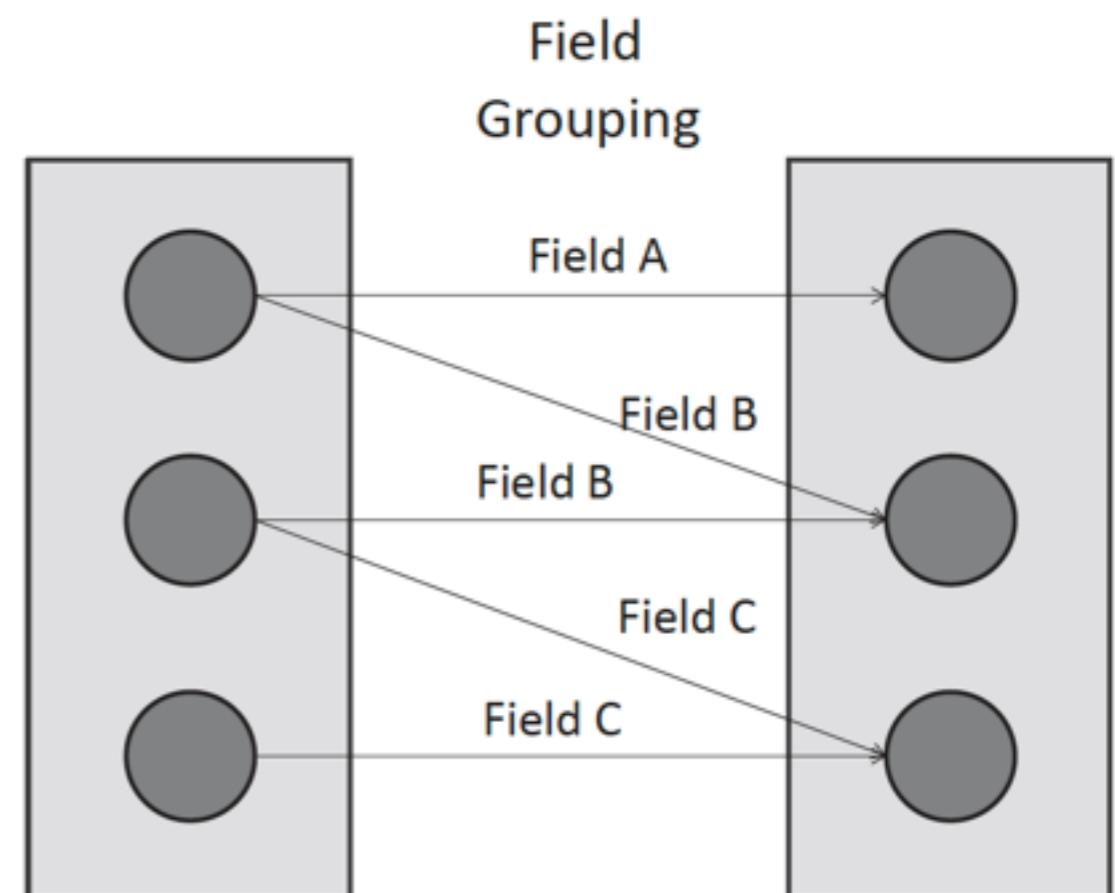




Stream grouping

Field grouping

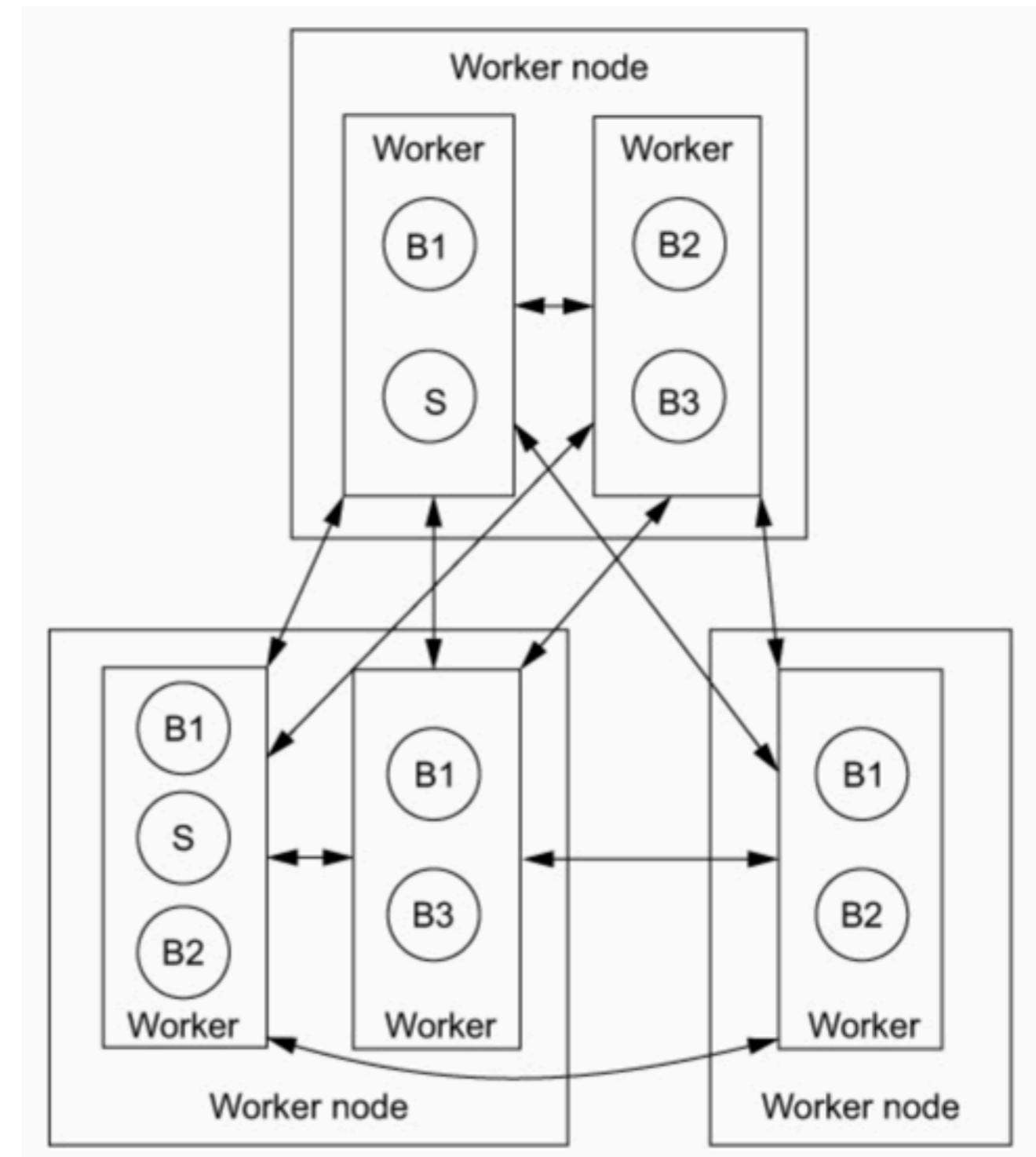
- Hashing de un subconjunto de campos de la tupla y saca el módulo del resultado con el número de las tareas consumidoras.
- El hashing entrega la clave, y todos los datos que tienen la misma clave asociada van al mismo nodo.
- Este es muy usado para agregar grandes conjuntos de datos.





Arquitectura Storm

- Las tareas son diseminadas entre diferentes trabajadores del cluster.
- Cada trabajador es una máquina virtual.
- Una vista física de como la topología podría ser distribuida en tres servidores.

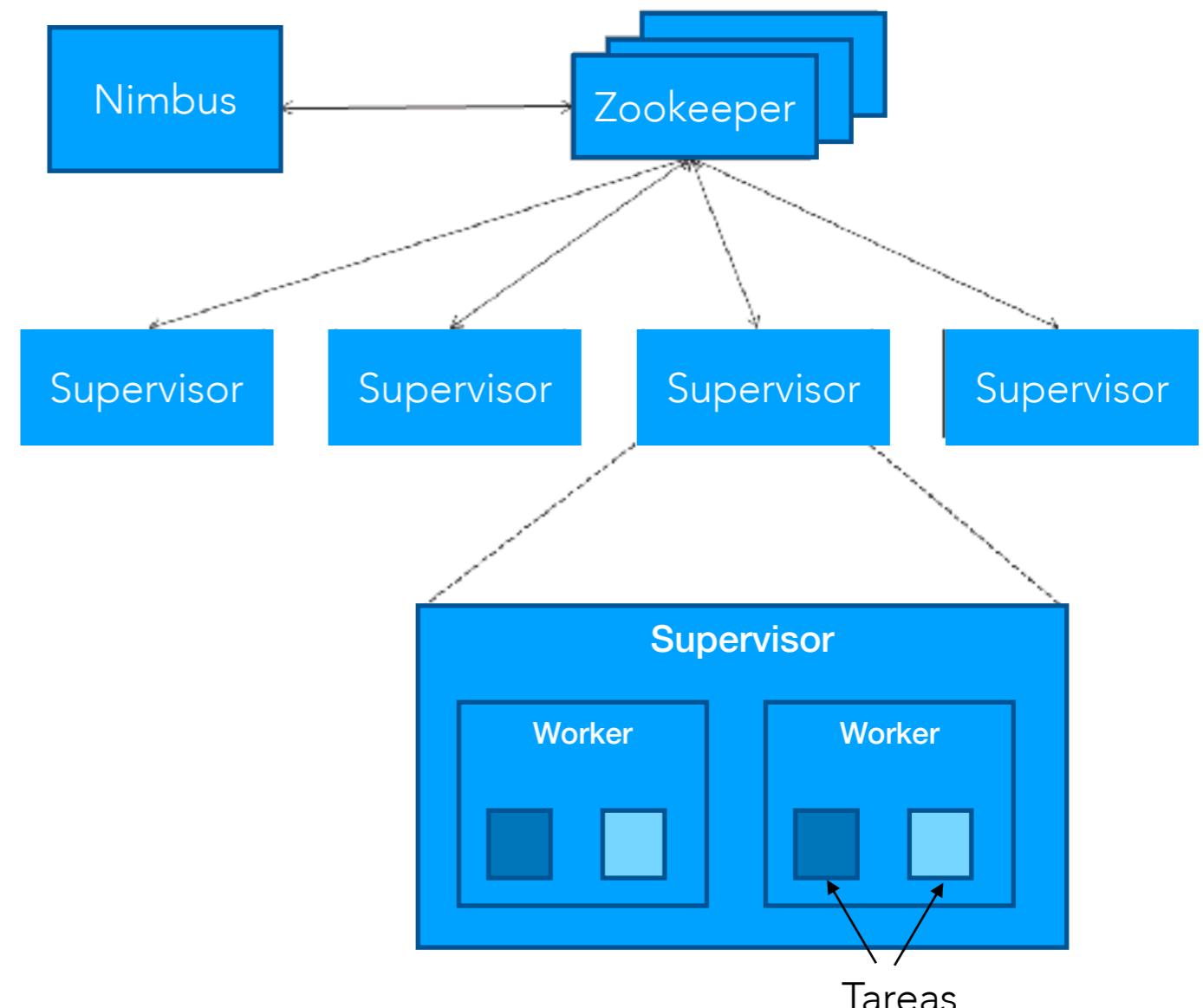




- **Nimbus:** responsable de distribuir el código de la topología y tareas en el cluster, lanzar los workers en el clusters y monitorear ejecución.
- **Supervisor:** Ellos se comunican con Nimbus a través de Zookeeper.
- **Zookeeper:** Servicio de coordinación distribuida con alto rendimiento. Mantiene la información de la configuración, provee sincronización y servicios de grupo. Mantienen el estado del cluster de Storm.
- Las topologías se empaquetan como Jar y se envían a Nimbus para su ejecución.

Arquitectura Storm

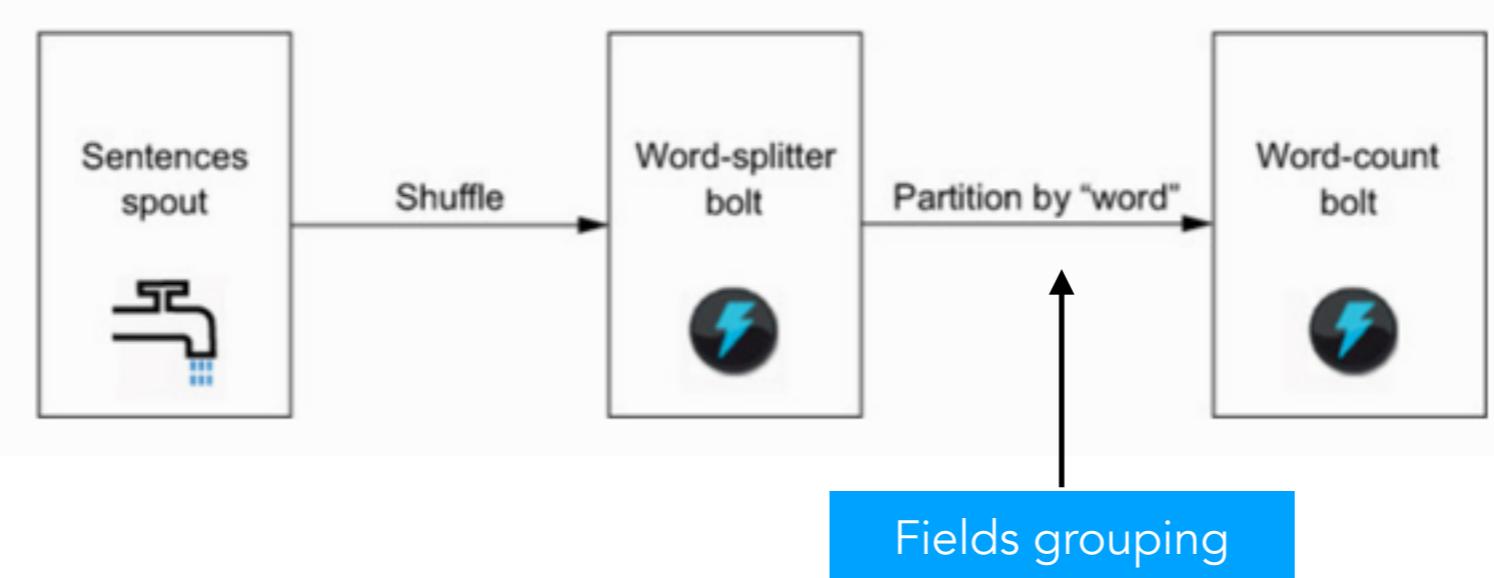
Componentes de un cluster de Storm





Topología ejemplo: conteo de palabras

Topología conteo de palabras



Los bolts son definidos como objetos porque pueden mantener estado interno.

```
class SplitterBolt{  
    function execute(sentence){  
        for(word in sentence.split(" ")){  
            emit(word)  
        }  
    }  
}
```

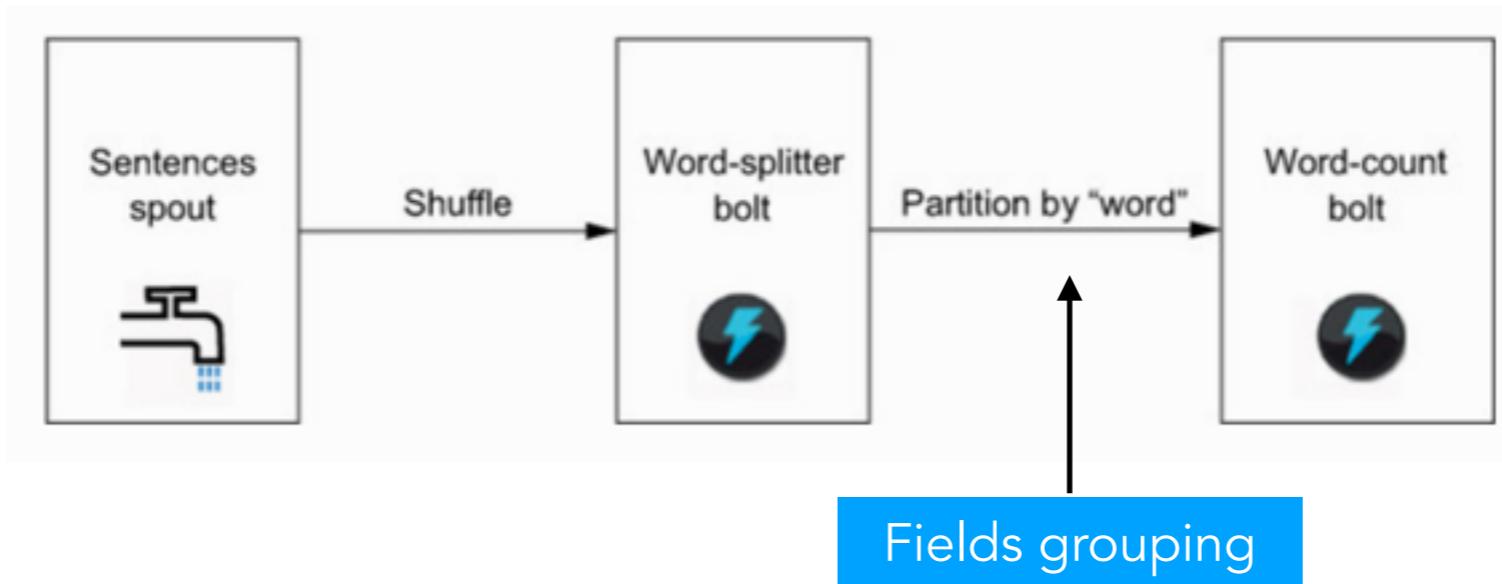
Los bolts reciben tuplas. En este caso, este bolt recibe una tupla con un campo.

Emite una palabra al stream de salida. Cualquier subscriptor a este bolt recibirá la palabra.



Topología ejemplo: conteo de palabras

Topología conteo de palabras



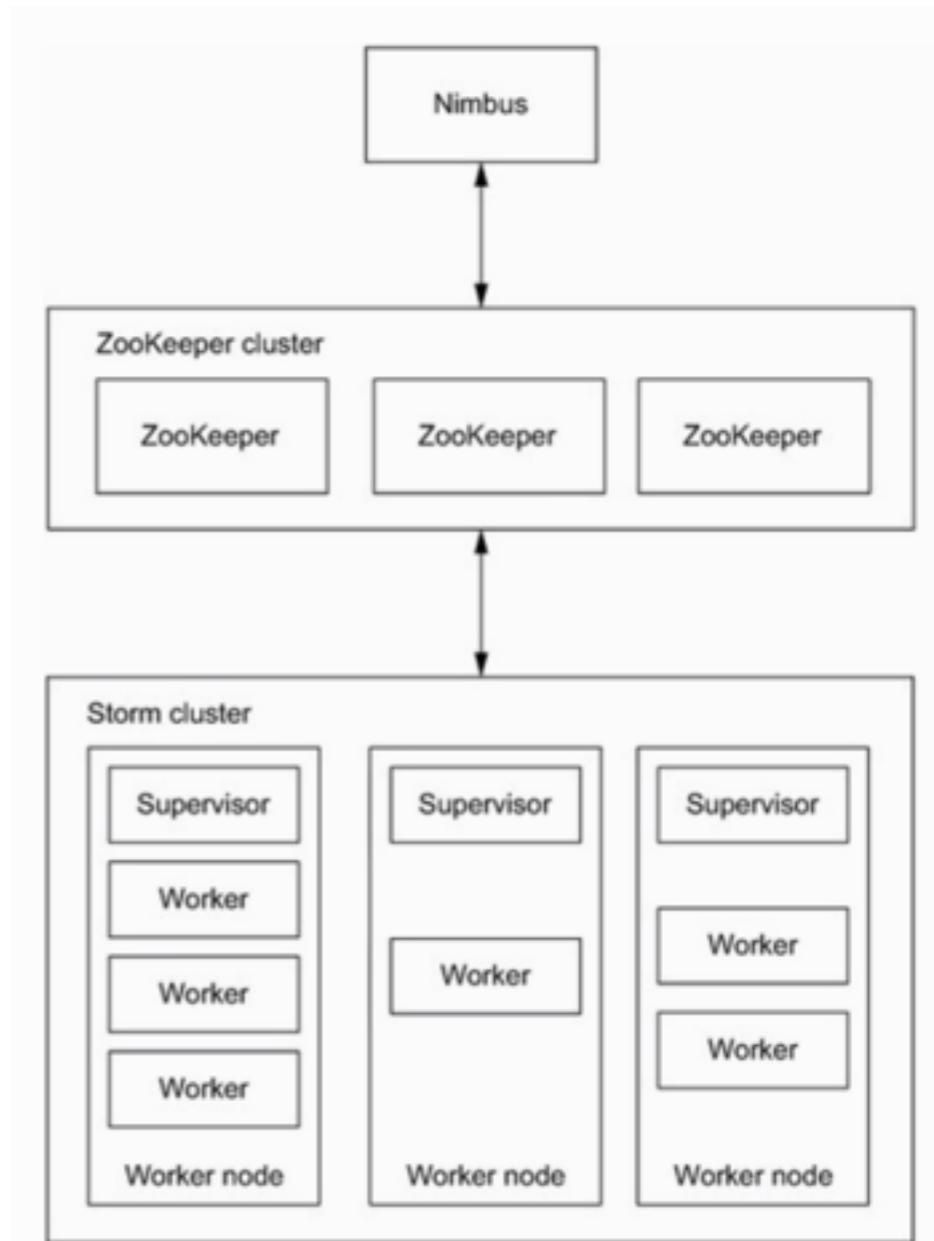
```
class WordCountBolt{  
    counts = Map(default=0)  
  
    function execute(word){  
        counts[word]++  
        emit(word, counts[word])  
    }  
}
```

Los contadores de las palabras se mantienen en memoria.



Clusters y despliegue

- Hay dos tipos de nodos en un **cluster de Storm**.
- **Nodo Master** -> corre **Nimbus**, responsable de distribuir código, asignar tareas a workers, monitorear fallas.
- **Nodos Workers** -> corre un daemon **Supervisor**, ejecuta porción de topología.
- Storm mantiene todos los estados en **Zookeeper**, para que los daemons (Supervisor y Nimbus) sean sin estado. Provee coordinación distribuida altamente confiable.
- Storm usa **ZeroMQ**: librería de socket, framework concurrente, más rápido que TCP, multicast, paso de mensajes escalable multicore, asíncrono, pub/sub, pipeline, request-reply.





Cluster y despliegue

```
Public static void main (String[ ] args) throws Exception {  
    TopologyBuilder builder = new TopologyBuilder();  
    builder.setSpout("sentence-spout", new RandomSentenceSpout(), 8);  
    builder.setBolt("splitter", new SplitSentence(),12)  
        .shuffleGrouping("sentence-spout");  
    builder.setBolt("count", new WordCount(), 12)  
        .fieldsGrouping("splitter", new Fields("word"));  
    Config conf = new Config();  
    conf.setNumWorkers(4);  
    StormSubmitter.submitTopology( "word-count-topology", conf,  
        builder.createTopology());  
    conf.setMaxSpoutPending(1000);
```

Spawns four worker nodes
among the Storm servers

Provides a name when
submitting the topology

Caps the number of
unasked tuples a spout can
emit



Garantías de procesamiento de mensajes

- **Exactly-once:** Cada elemento es procesado exactamente una vez.
 - **At-least-once:** Cada elemento es procesado al menos una vez. Puede ser procesado más veces (duplicados).
 - **At-most-once:** Cada elemento es procesado a lo más una vez y es posible que no sea procesado.
-
- Storm mantiene garantías de at-least-once sin garantías en las colas intermedias.
 - Storm usa un algoritmo para hacer seguimiento de tuplas que son emitidas y puede volver a emitirlas desde el spout si hay una falla en el flujo.
 - Mantiene timeout que asegura que las fallas son detectadas.

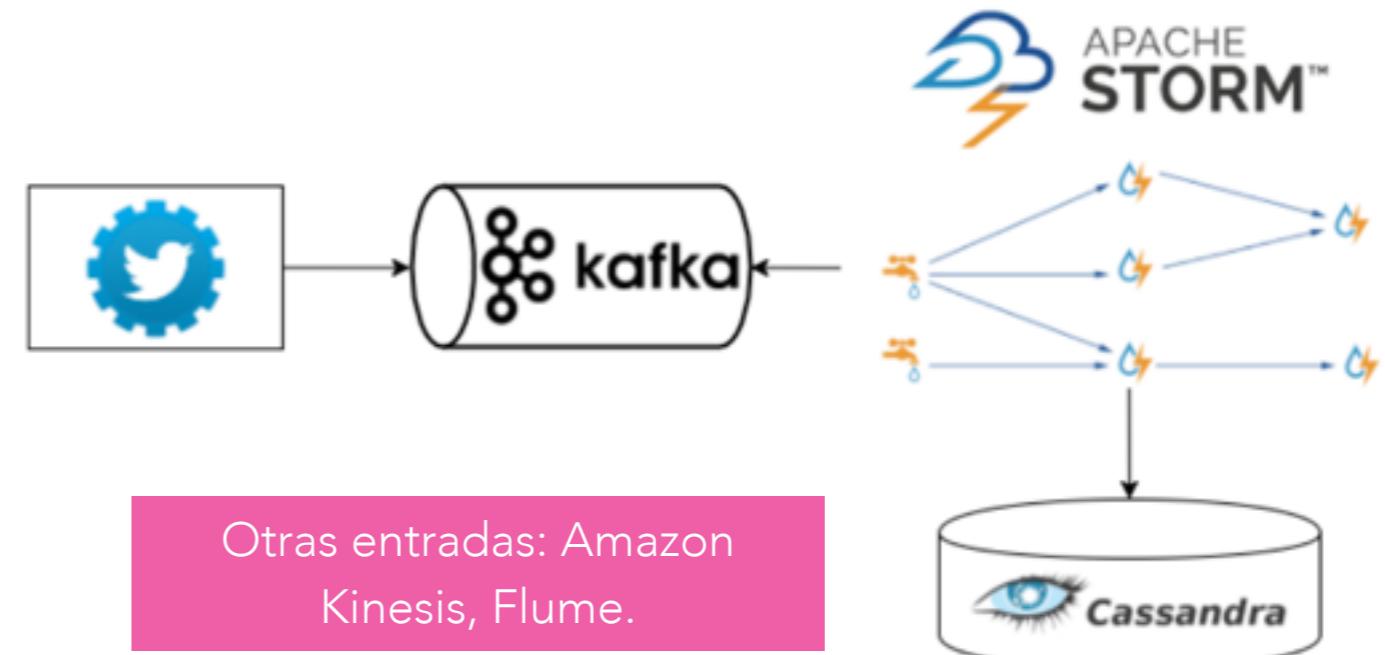
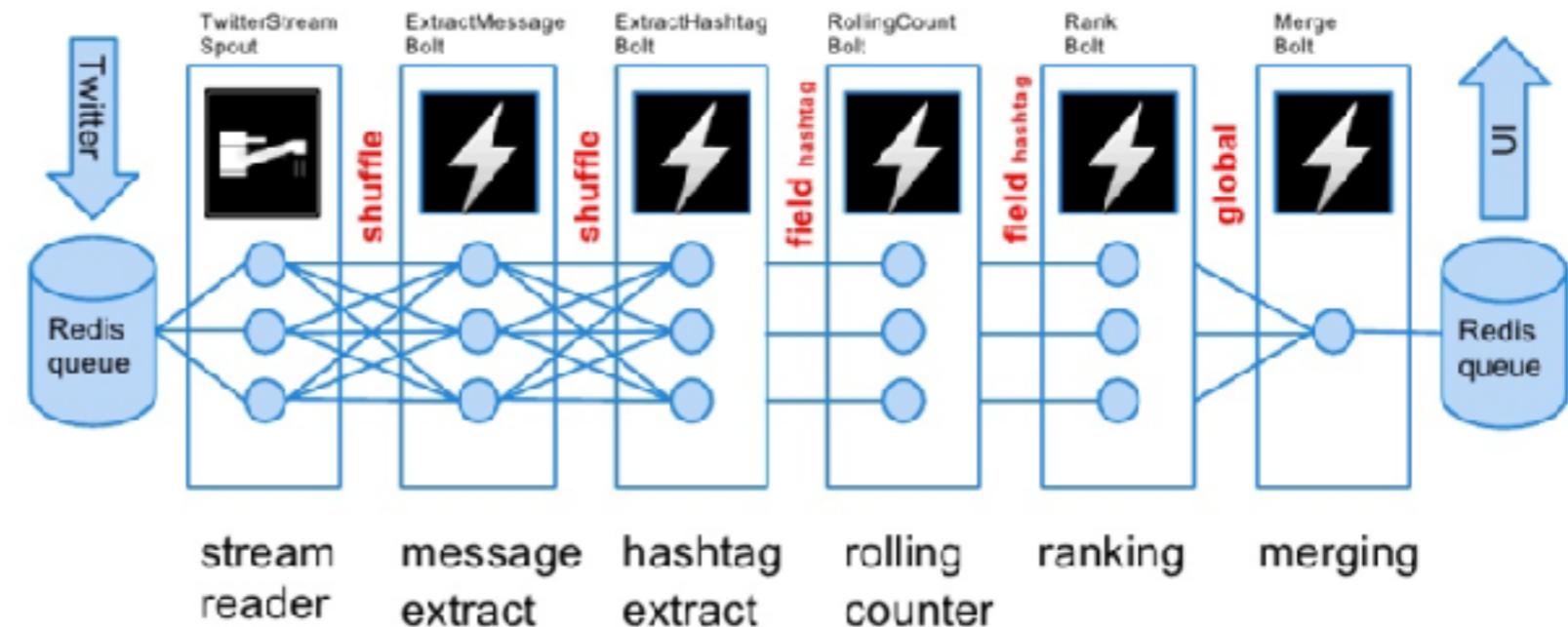


Redis: Almacenamiento en memoria no relacional con soporte de persistencia.

Kafka: sistema pub/sub. Los datos son almacenados de forma durable, en orden y pueden ser leídos de manera determinista. Puede distribuir los datos para proveer protección contra fallas

Cassandra: Base de datos de código abierto, distribuida, descentralizada, escalable, altamente disponible, tolerante a fallas:
Basada en Dynamo y Bigtable fue creada en Facebook.

Ecosistema de Storm





Casos de Uso de Storm

- **Twitter:** discovery, realtime analytics, personalization, search, revenue optimization, and many more. Storm integrates with the rest of Twitter's infrastructure, including database systems (Cassandra, Memcached, etc), the messaging infrastructure, Mesos, and the monitoring/alerting systems. Storm's isolation scheduler makes it easy to use the same cluster both for production applications and in-development applications, and it provides a sane way to do capacity planning.
- **Spotify:** Spotify serves streaming music to over 10 million subscribers and 40 million active users. Storm powers a wide range of real-time features at Spotify, including music recommendation, monitoring, analytics, and ads targeting. Together with Kafka, memcached, Cassandra, and netty-zmtp based messaging, Storm enables us to build low-latency fault-tolerant distributed systems with ease.
- **Groupon:** use Storm to build real-time data integration systems, analyze, clean, normalize, and resolve large amounts of non-unique data points with low latency and high throughput.
- **The Weather channel:** Each topology is responsible for fetching one dataset from an internal or external network, reshaping the records for use by our company, and persisting the records to relational databases. It is particularly useful to have an automatic mechanism for repeating attempts to download and manipulate the data when there is a hiccup.



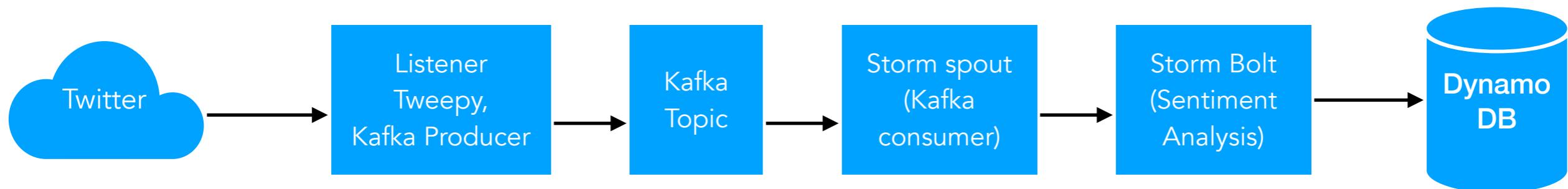
Casos de Uso de Storm

- **Verisign:** Global leader in domain names and Internet Security. Example: use Storm in security monitoring to analyze the network telemetry data of our globally distributed infrastructure in order to mitigate cyber attacks.
- **Cerner:** Leader in health care information technology. Use Storm to process massive amounts of clinical data in real-time.
- **Flipboard:** social magazine. Use in content search, realtime analytics and to generate custom magazine feeds.
- **Taobao:** statistics of logs and extract useful information from statistics in almost real-time.
- **Alipay:** China's leading third-party online payment platform. Use Storm to calculate realtime trade quantity, trade amount, top N seller trading information, user registers count (more than 100 million messages per day). Log processing more than 6T data per day.



Análisis de sentimientos en Twitter en tiempo real

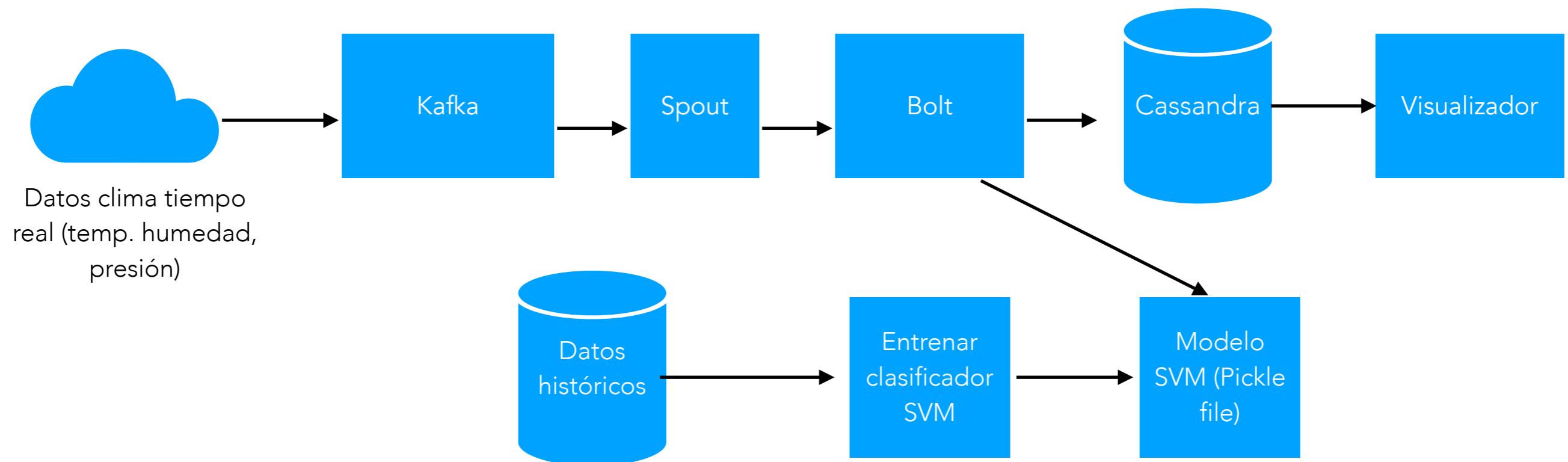
- Se usan frameworks de Apache Storm y Apache Kafka.
- El listener conecta a Twitter con la API de streaming (no todos!) y alimenta a un tópico de Kafka.
- Este a su vez alimenta al spout de Storm.
- Los bolts son stateless (sin estado) y realizan análisis de sentimientos usando lexicon.
- Dynamo DB almacena el resultado.
- Otros sistemas pueden leer los tweets de la BD con su sentimiento y los muestra en una página Web por ejemplo.





Análisis de datos climáticos en tiempo real

- El sistema determina si se espera niebla en una localización particular.





Procesando streams

Un evento a la vez (Native Streaming)

Caso de estudio: Storm

- Menor latencia
- Semántica at-least-once.
- Modelo de programación simple

Otros ejemplos: Flink, Kafka Streams,
Samza

VS

Micro-batch

Caso de estudio: Spark Structured
Streaming

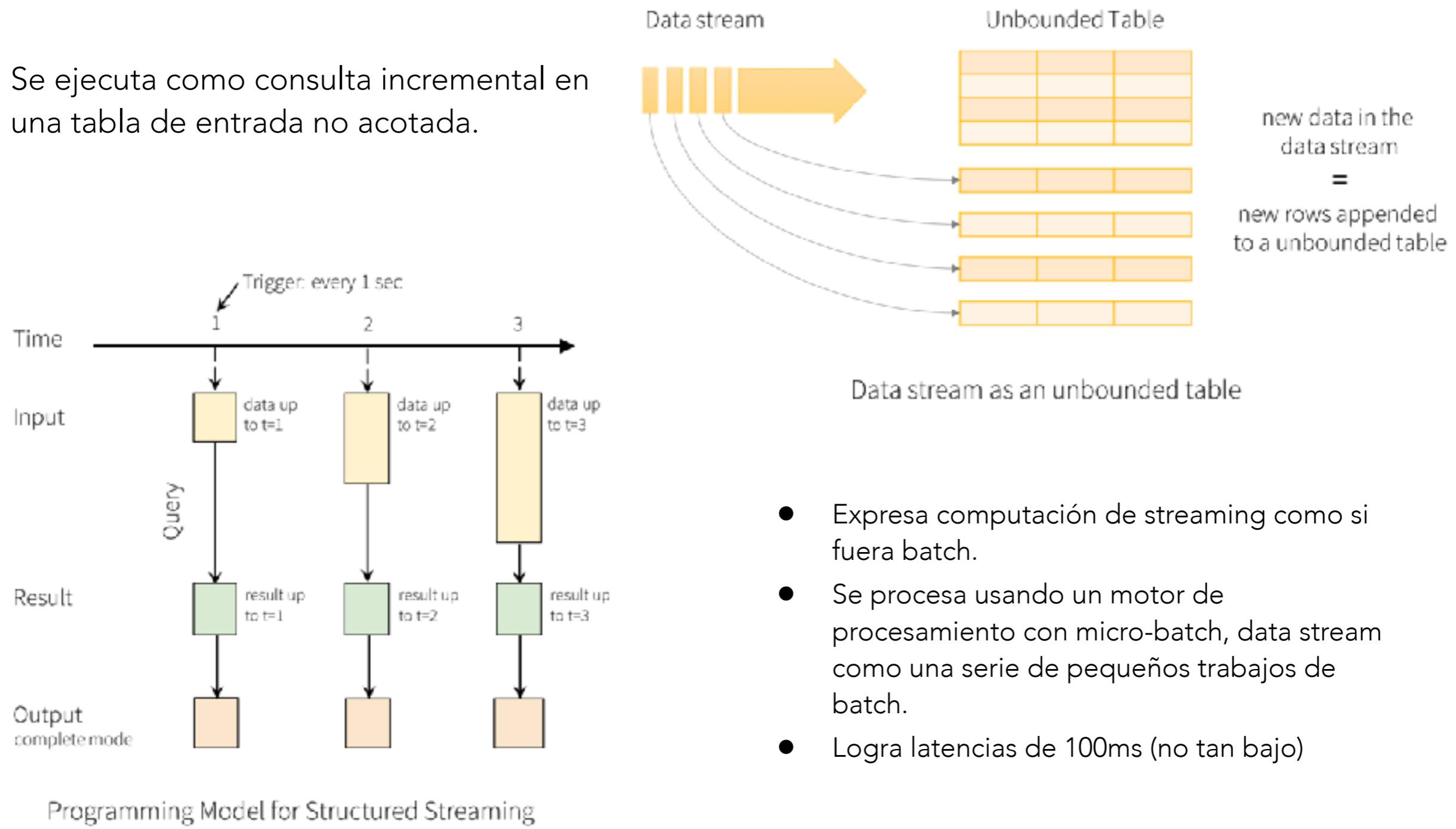
- Mayor rendimiento (número de eventos procesados por período de tiempo)
- Semántica exactly-once

Otros ejemplos: Storm-Trident



Spark Structured Streaming

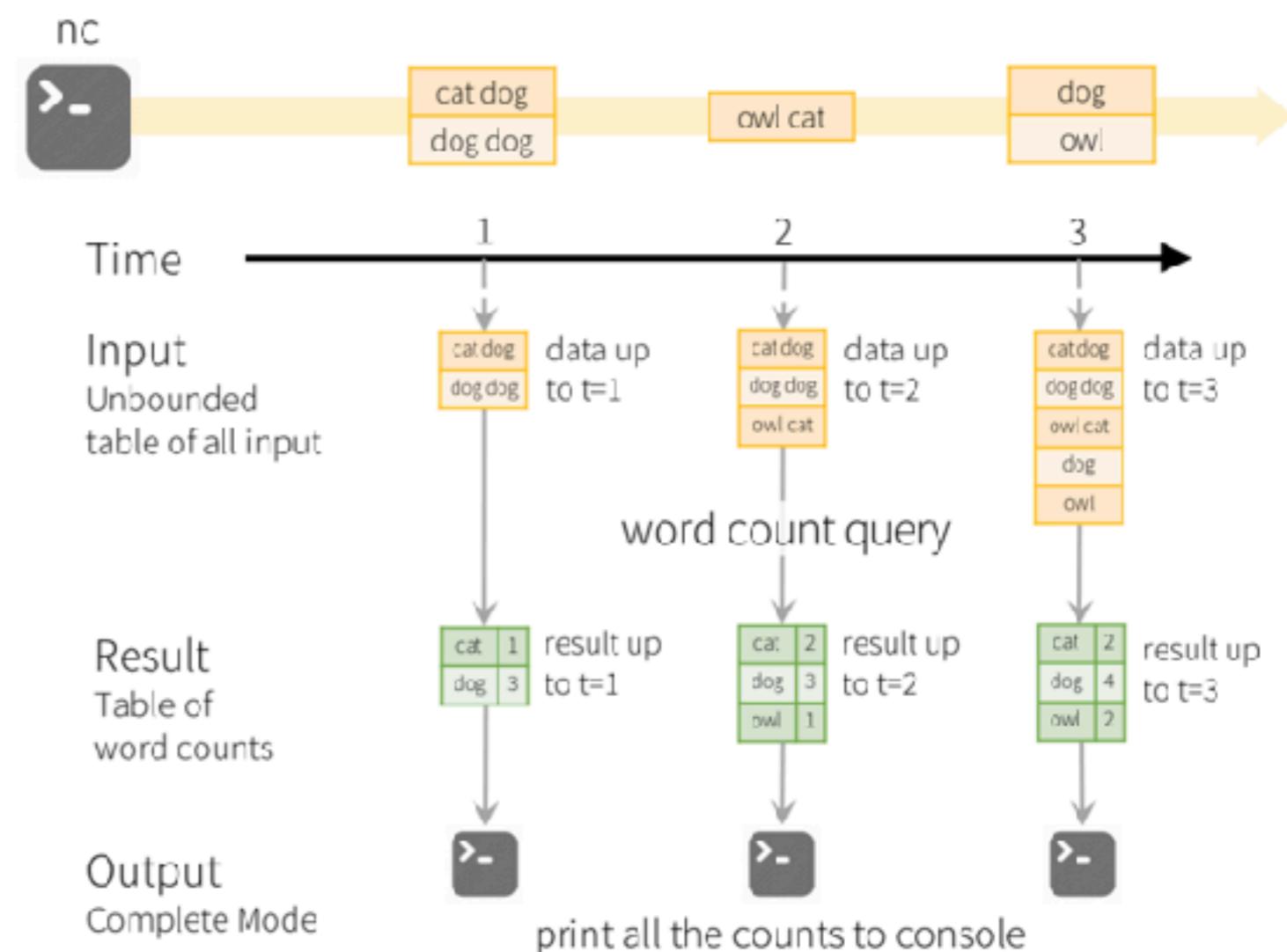
- Se ejecuta como consulta incremental en una tabla de entrada no acotada.





Ejemplos salida Structured Streaming

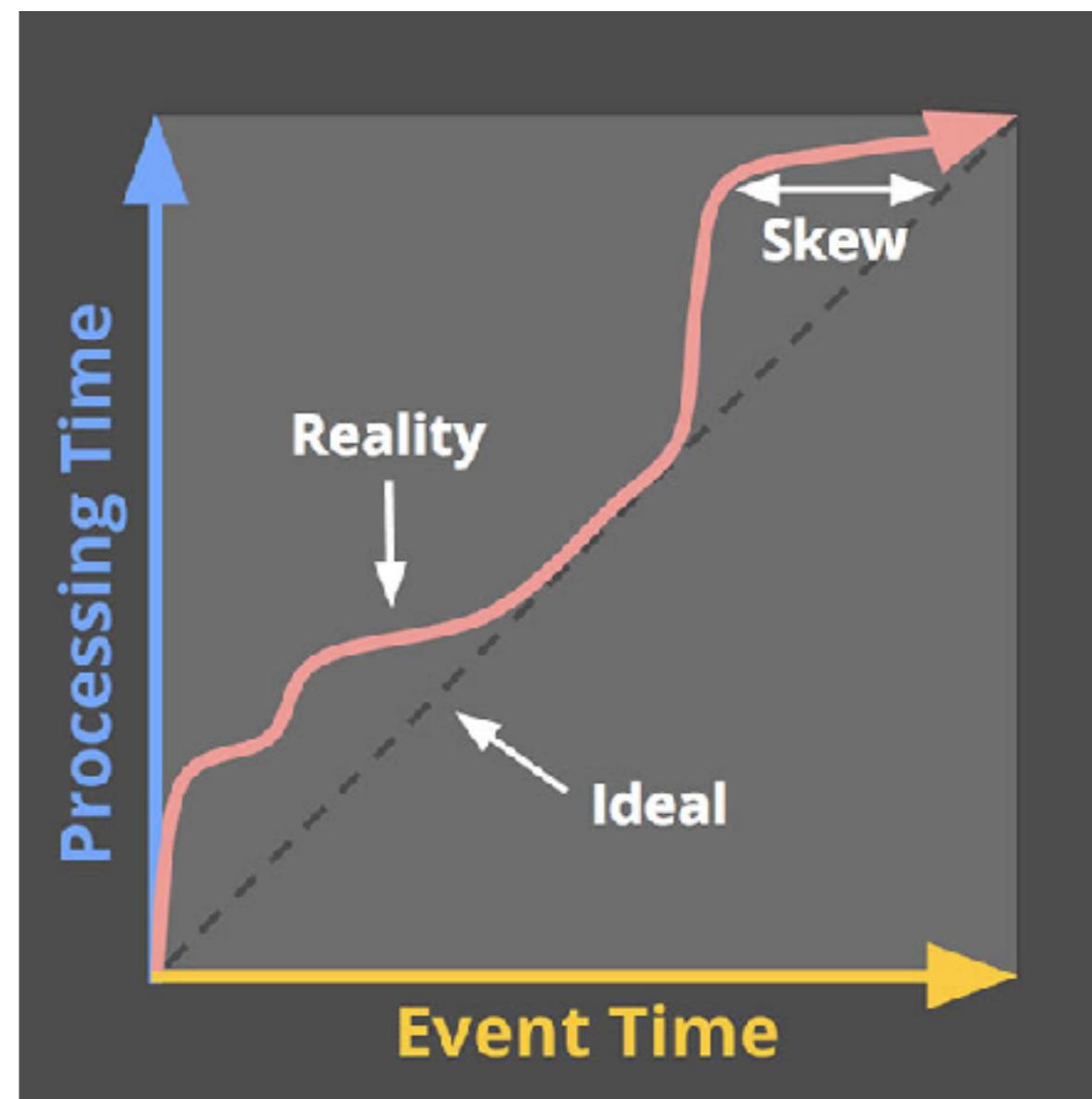
- **Complete Mode:** Escribir la tabla completa al almacenamiento externo.
- **Append Mode:** Solo escribe las filas nuevas.
- **Update Mode:** Solo escribe las filas que fueron actualizadas.





Datos con tiempo de eventos

- Tiempo en que los datos son generados no es el mismo en que son recibidos por el sistema.
- Ejemplo: IoT
- Event-time es un valor en una columna de cada fila.
- Las agregaciones por ventanas tiempo se pueden hacer de manera consistente.
- Se soporta Watermarking: especificar un threshold para que tan tarde puede llegar un dato, permitiendo limpiar estados antiguos.





Operaciones sobre los DataFrames/Datasets

- Soporta la mayoría de las operaciones de Batch
- No soporta algunas operaciones que naturalmente no se pueden realizar sobre streaming data, por ejemplo:
 - No soporta sorting, a menos que sea sobre la agregación del output.
 - No soporta “take the first N rows”.

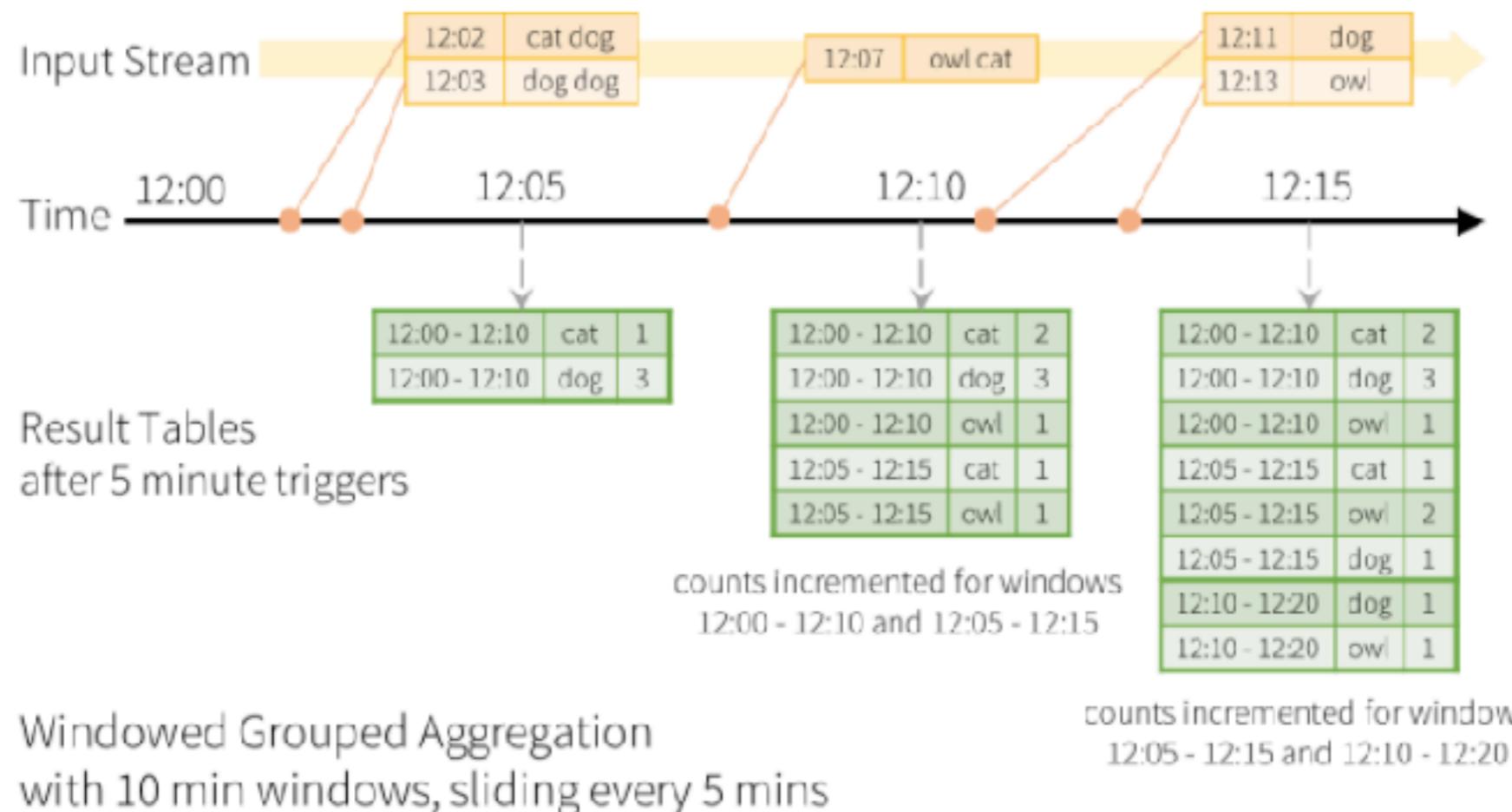
```
df = ... # streaming DataFrame with IOT device data with schema { device: string, deviceType: string, signal: double, time: DateType }

# Select the devices which have signal more than 10
df.select("device").where("signal > 10")

# Running count of the number of updates for each device type
df.groupBy("deviceType").count()
```



Agregación sobre Ventanas de tiempo por tiempo de Evento



```
words = ... # streaming DataFrame of schema { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of each group
windowedCounts = words.groupBy(
    window(words.timestamp, "10 minutes", "5 minutes"),
    words.word
).count()
```

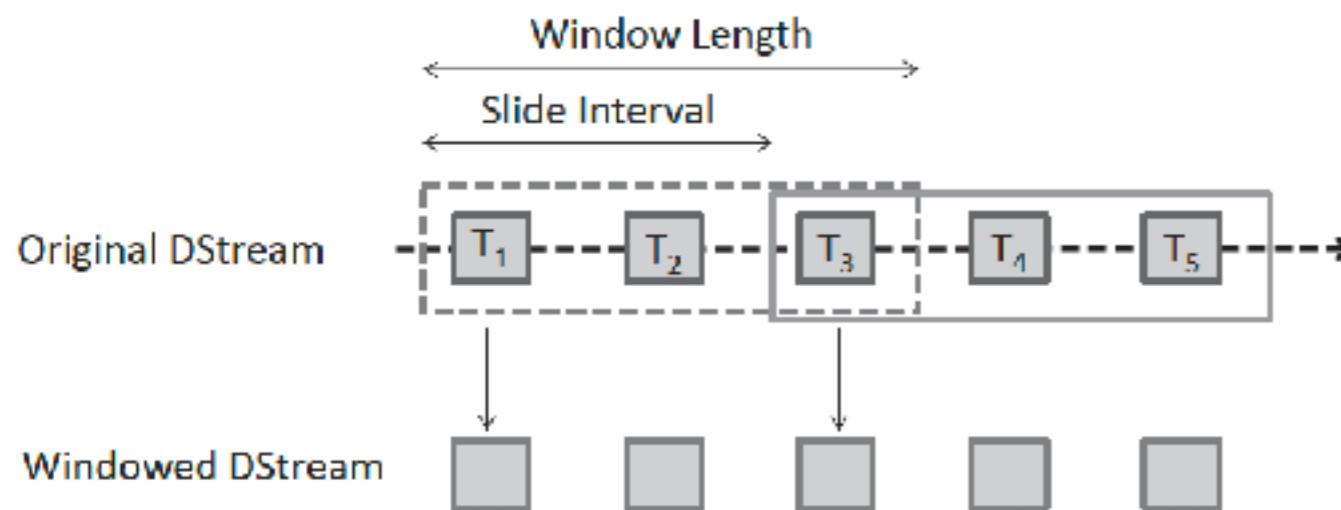
Window length

Slide window



Streaming con Spark

- Las operaciones con estado son soportadas con ventanas y requieren checkpointing.
- Las transformaciones se hacen sobre una ventana de datos.
- Los RDD en la ventana se combinan.
- 2 parámetros, el tamaño de la ventana y cada cuánto se mueve la ventana o cada cuánto se realiza la operación.



Fuente: Big Data Science and Analytics Book

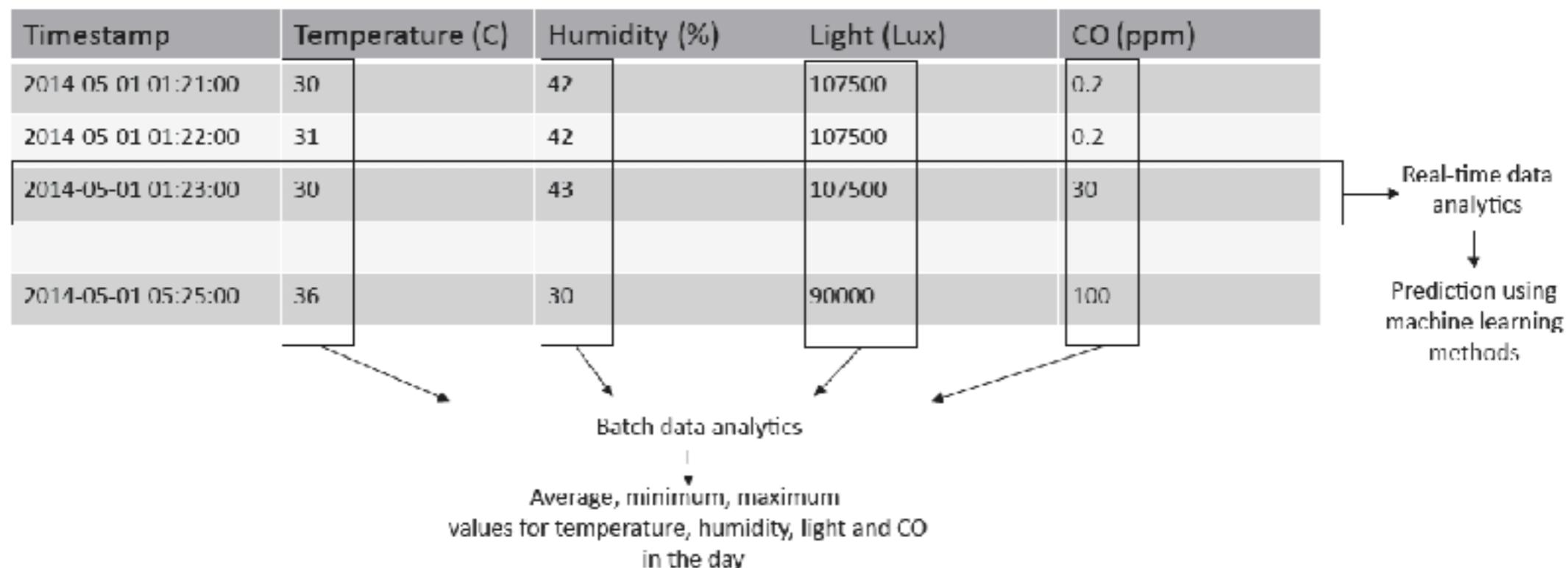
Operaciones comunes:

- Window: retorna un RDD con los datos de la ventana.
- countByWindow: Realiza la operación count sobre los elementos de la ventana.
- reduceByWindow: Agrega los elementos de la ventana.
- reduceByKeyAndWindow: Agrega los con la misma clave en la ventana.



Ejemplo Análisis Datos de Sensores en tiempo real

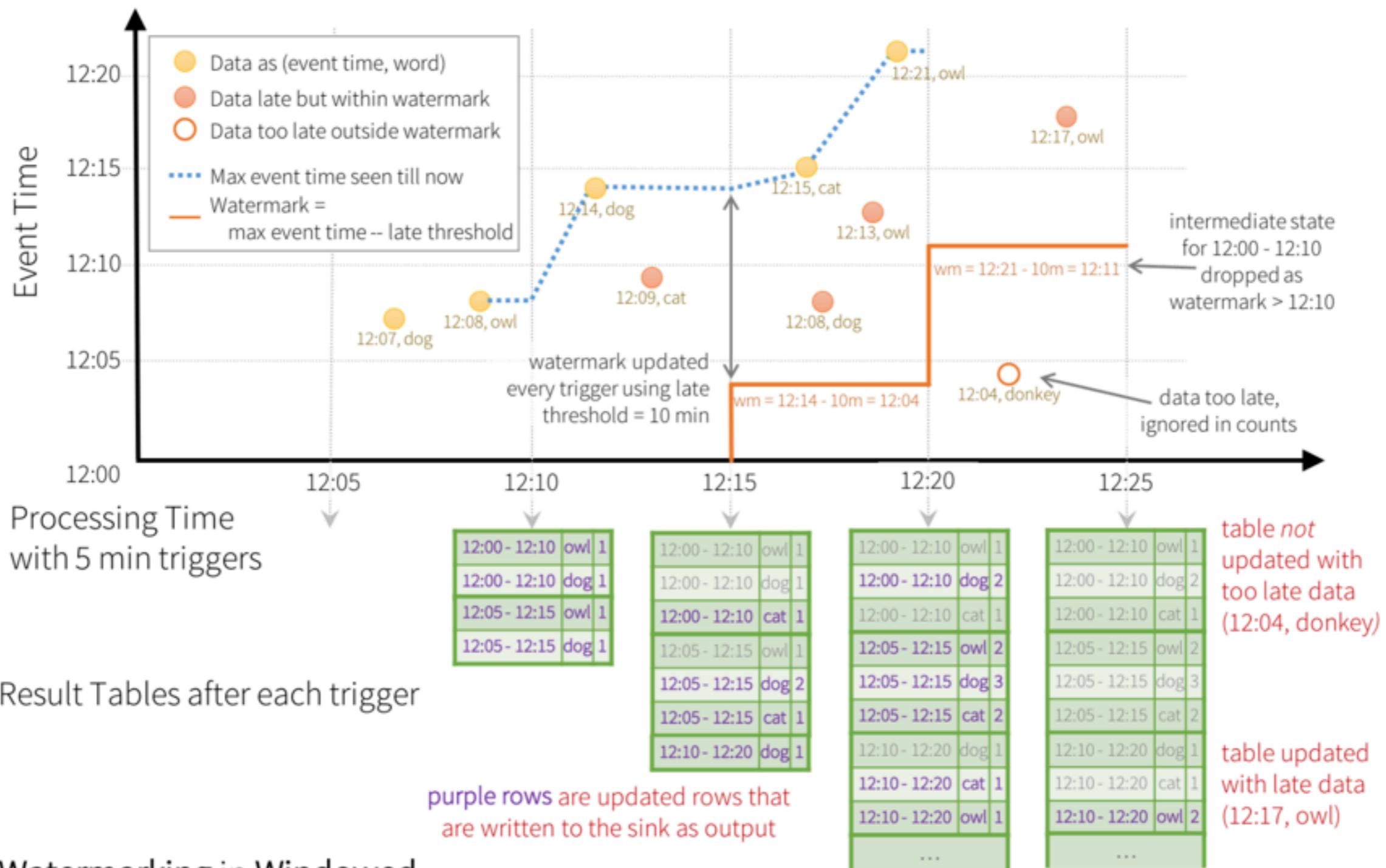
- Contexto sistema de detección de incendios forestales.
- Nodos desplegados por un bosque, con sensor de humedad, temperatura, luz, monóxido de carbono.
- Los datos son enviados a un Kafka topic.



Fuente: Big Data Science and Analytics Book



Watermarking

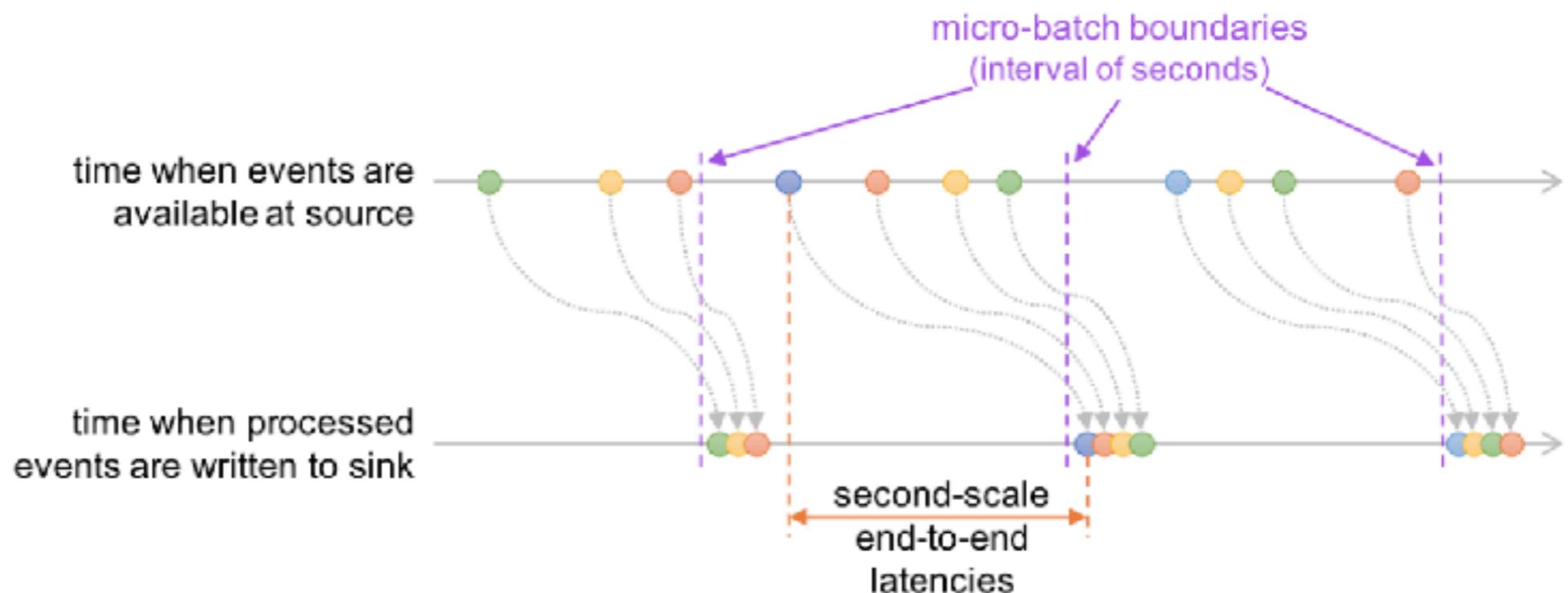


Watermarking in Windowed
Grouped Aggregation with Update Mode

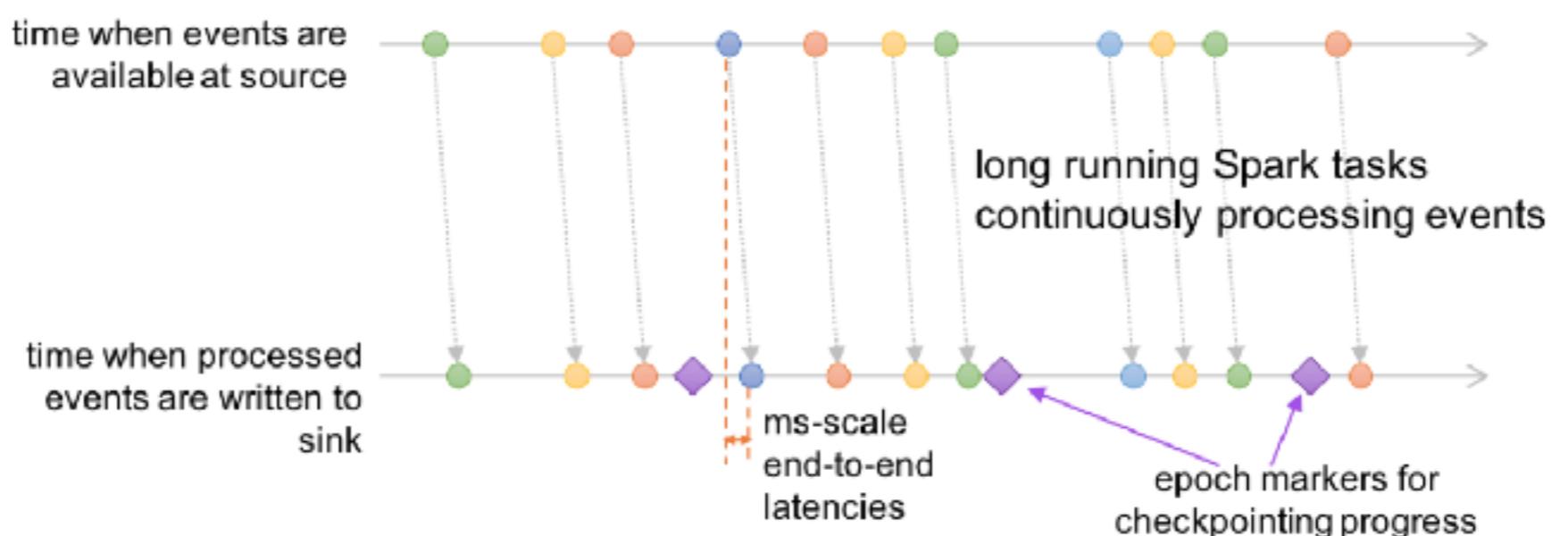


Análisis de latencia

Microbatching



Continuous processing





Cómo comparar Sistemas de procesamiento de Stream

- Muchos sistemas: Storm, Spark Streaming, Kafka Streaming, Flink, etc.
- Garantías de procesamiento: at-least-once, exactly-once, at-most-once
- Tolerancia a fallas: checkpointing
- Manejo de estado: Procesamiento con estado
- Rendimiento: Eventos por segundo, latencia, escalabilidad.
- Otras: soporte de event time processing, windowing, etc.
- Madurez: ha sido usado a gran escala por las grandes compañías? Soporte de comunidad?



Comparaciones

Storm

- Uno de los sistemas de procesamiento de stream más antiguos de código abierto.
- Ventajas: Baja latencia, streaming nativo, buena comunidad y alto rendimiento.
- Desventajas: No maneja estado, posee semántica de at-least-once, sin características avanzadas.

Spark Structured Streaming

- Da soporte completo a la arquitectura Lambda (implementación batch y streaming).
- Ventajas: alto throughput, tolerante a fallas, semántica de exactly-once, gran comunidad.
- Desventajas: No es streaming nativo para requerimientos de muy baja latencia.

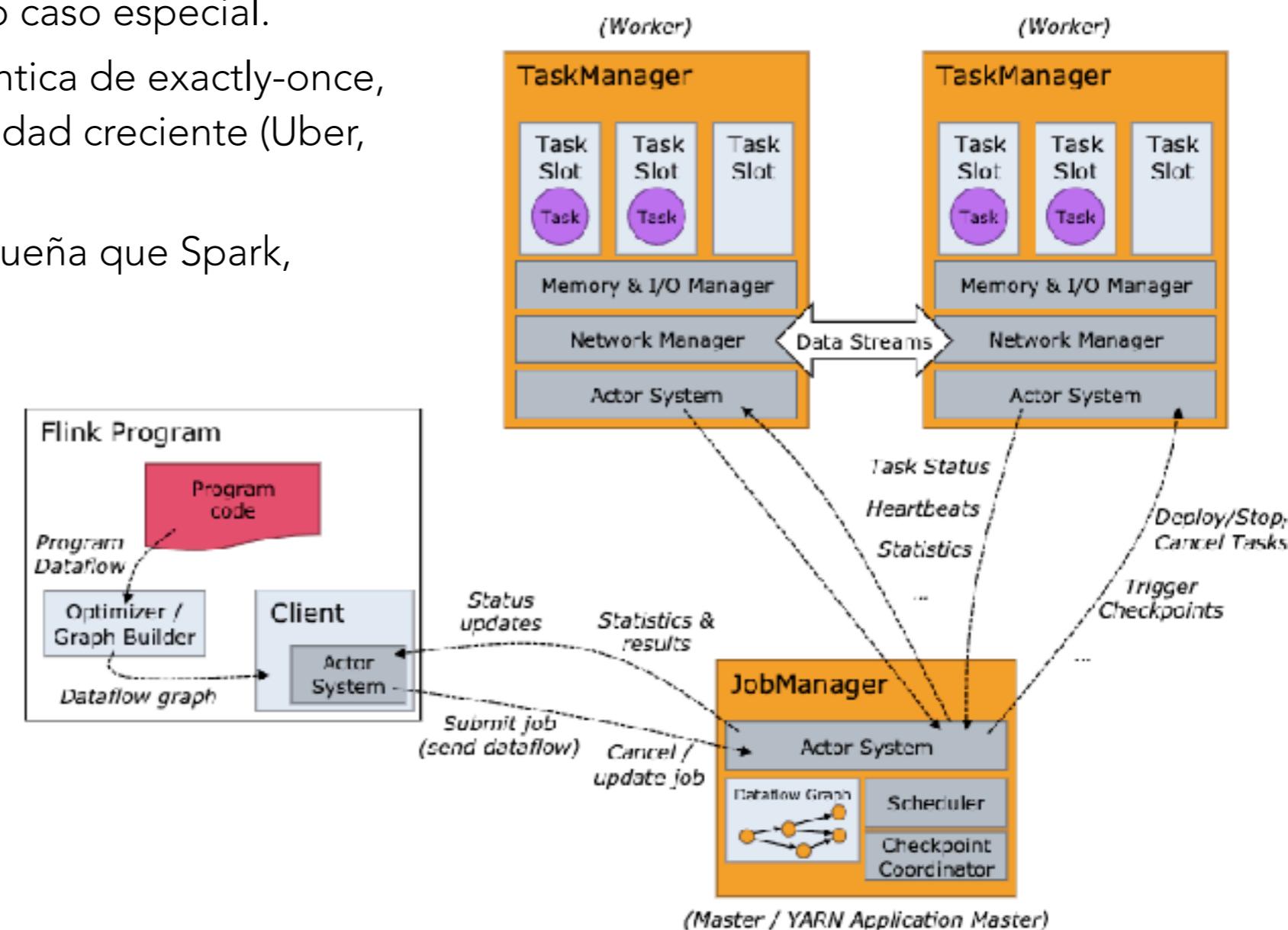


Comparaciones

Flink

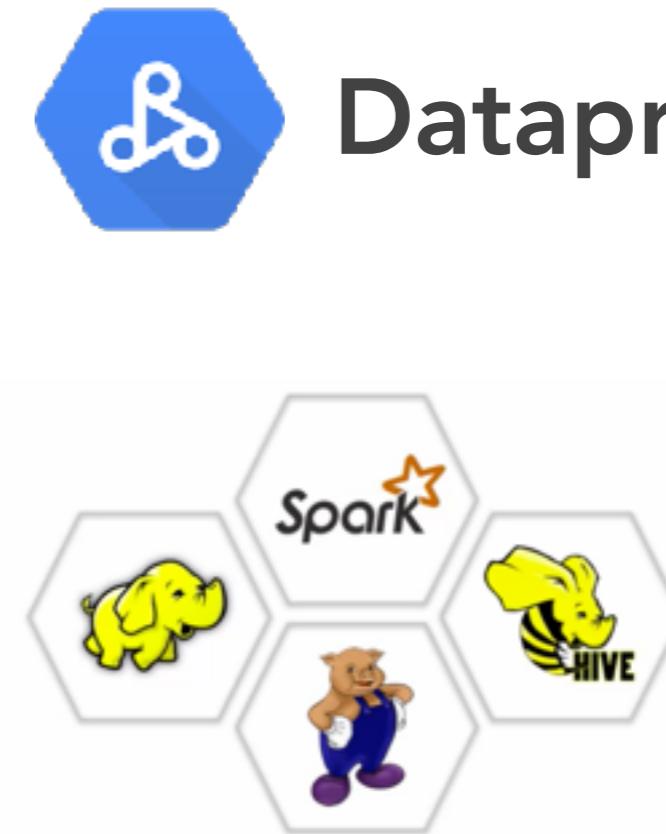
- Sucesor de Storm
- También procesa batch pero como caso especial.
- Ventajas: auto-configurable, semántica de exactly-once, mejor latencias que Spark, comunidad creciente (Uber, Alibaba)
- Desventajas: Comunidad más pequeña que Spark, entra tarde al mercado.

Ver el procesamiento en batch como un caso particular de streaming

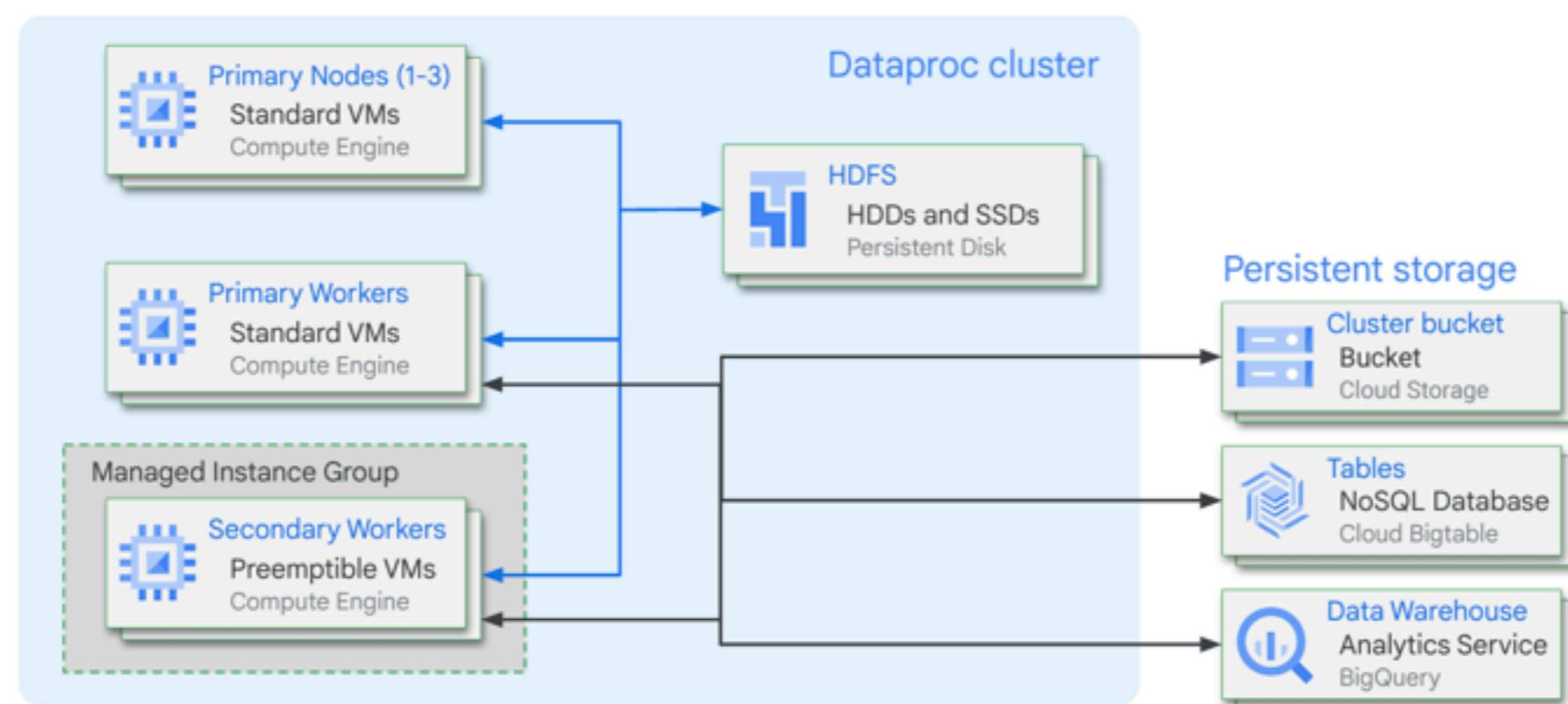




Dataproc

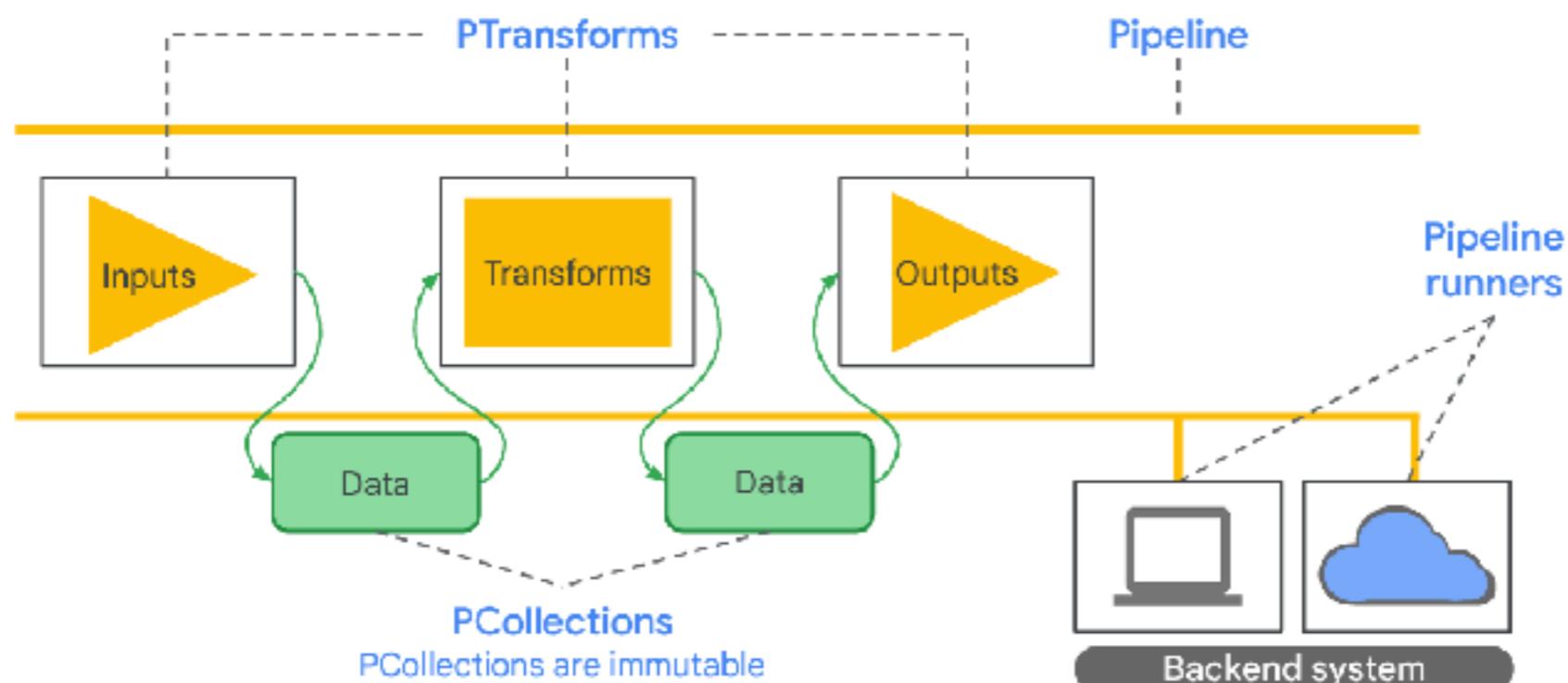


- Los clusters que se ocupan “on-premise” no son elásticos.
- Dataproc es un servicio para correr el Ecosistema de Hadoop y Spark en clusters elásticos.
- Ideal para migrar servicios ya implementados con herramientas open source.



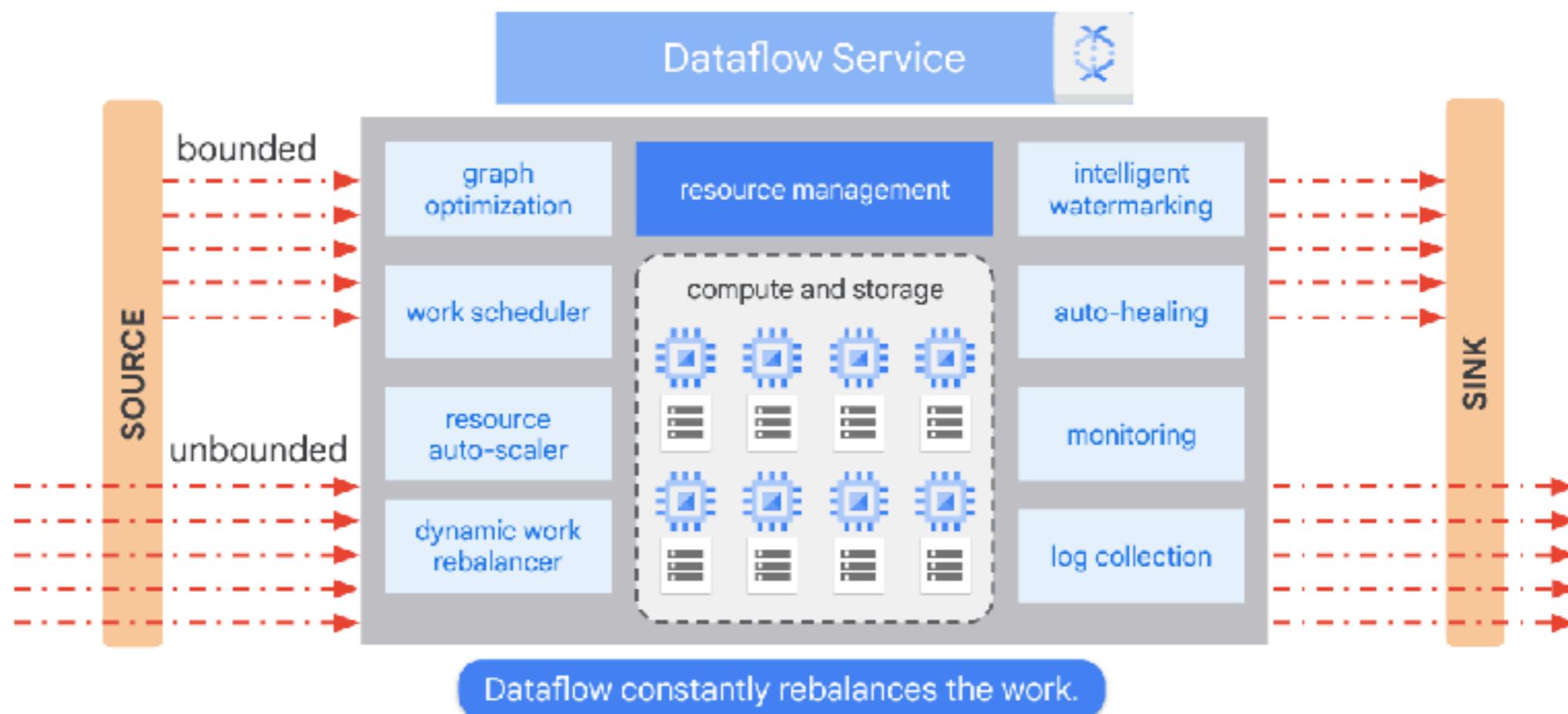


- Provee procesamiento batch y streaming
- Utiliza el modelo de Apache Beam
- 4 conceptos: PCollections, PTransforms, Pipeline y Pipeline runners.



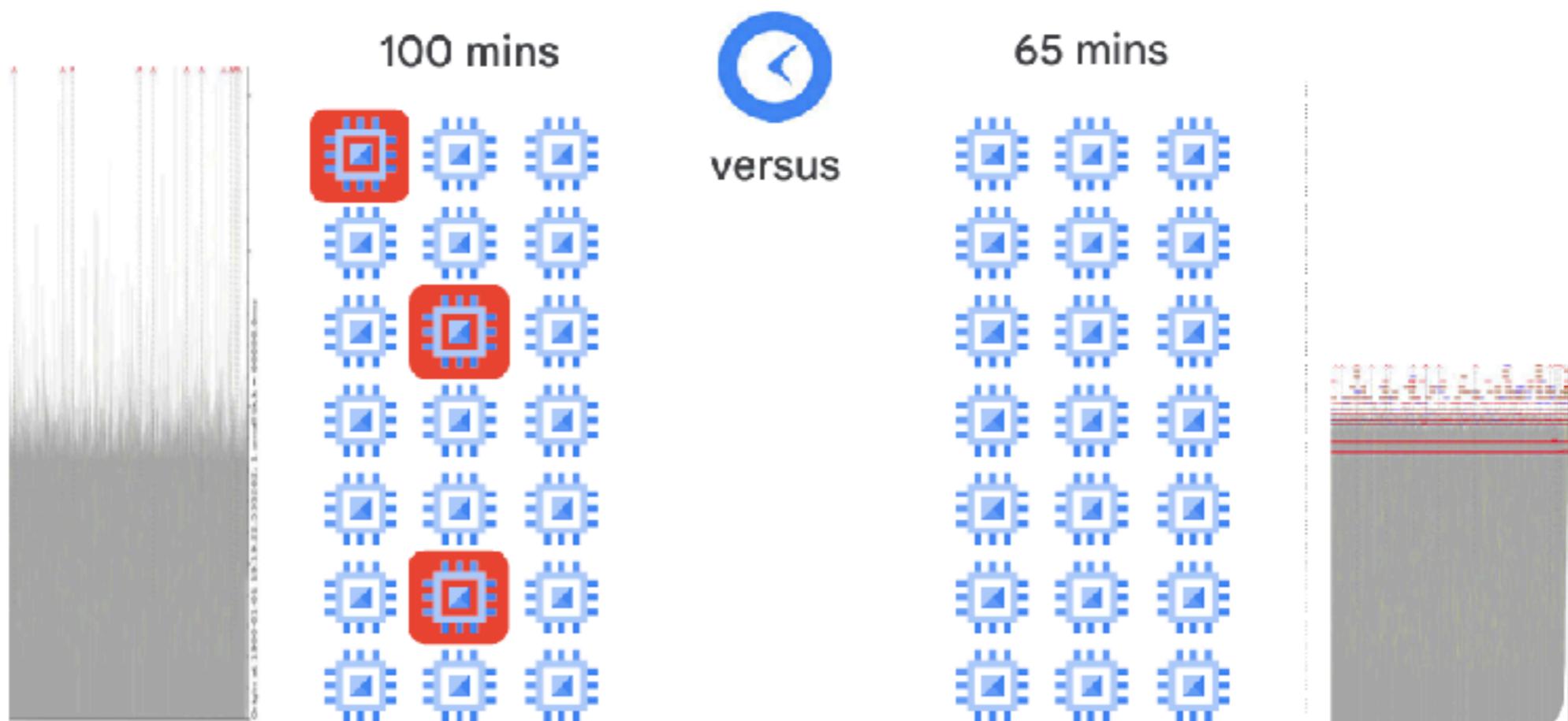


- Provee ejecución eficiente de Apache Beam.
- Dataflow elige como ejecutar el pipeline. Es un pipeline runner.





- Rebalanceo dinámico en tiempo de ejecución.





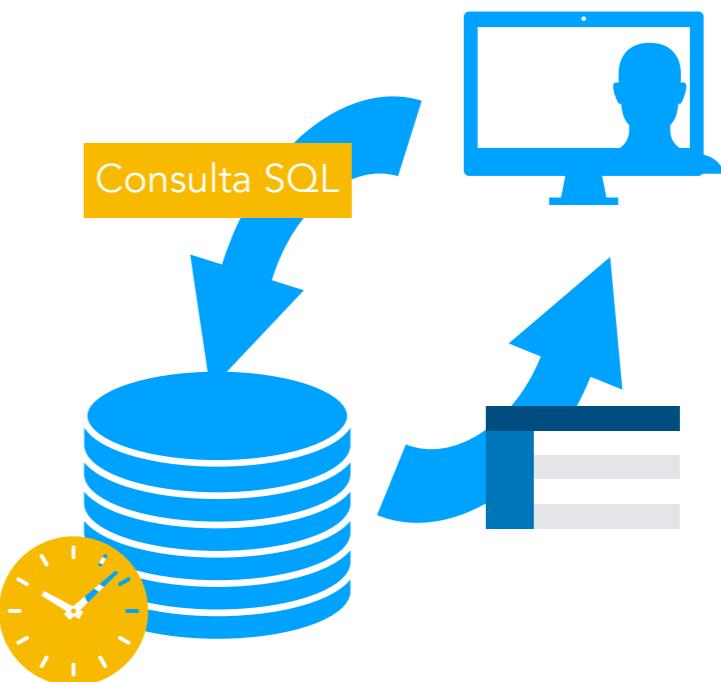
Comparativa Dataflow y Dataproc

	Dataflow	Dataproc
Recomendado para	Nuevos pipelines de procesamiento de datos, unificando batch con streaming.	Trabajos existentes de aplicaciones de Hadoop/Spark, machine learning/data science ecosystem, large-batch jobs.
Administrado completamente	Si	No
Auto-escalable	Si, adaptivo	Si, basado en la utilización del cluster.
Tecnología opensource	Apache Beam	Hadoop, Hive, Pig, Apache Big Data ecosystem, Spark, Flink, Presto, Druid.



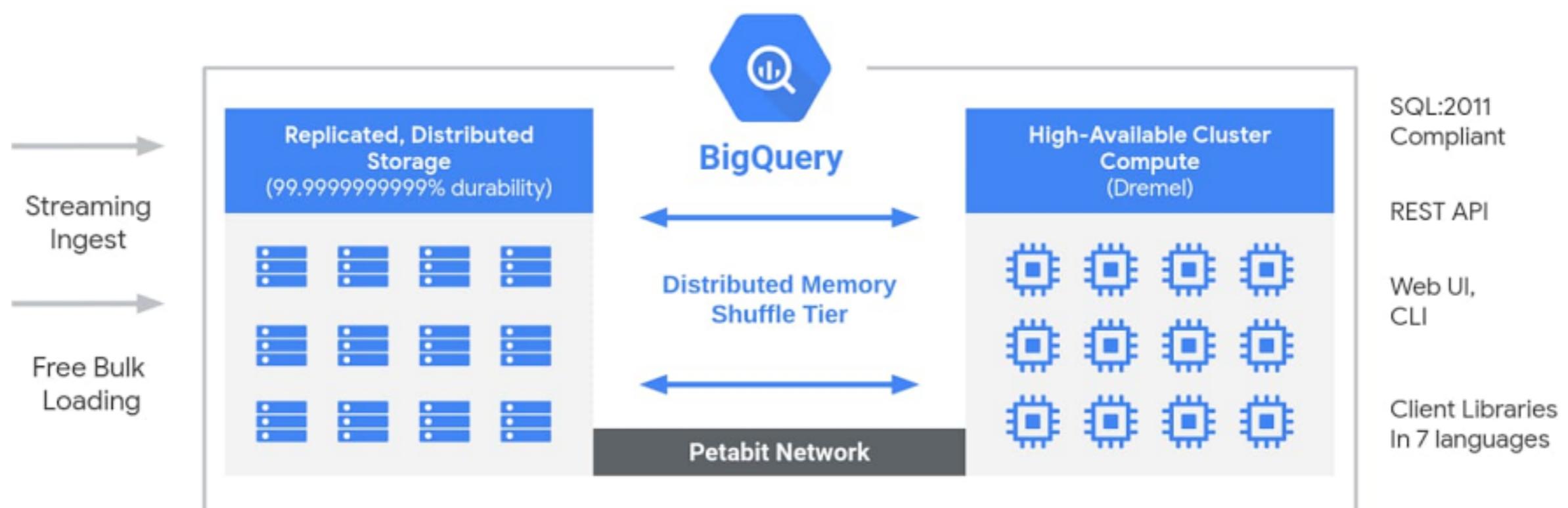
Consultas interactivas

- Permiten a los usuarios escribir consultas para consultar datos usando lenguajes de tipo SQL.
- Algunas de las tecnologías más usadas son:
 - **SparkSQL** -> Usa DataFrames
 - **Hive** es un Datawarehouse construido sobre Hadoop, organizando los datos en tablas, como una base de datos relacional.
 - En **AWS Redshift** Datawarehouse usa una arquitectura MPP (Massive Parallel Processing) y **Athena** que tiene enfoque serverless y sin almacenamiento que puede consultar datos en S3.
 - **BigQuery**: Serverless, escala a petabytes.



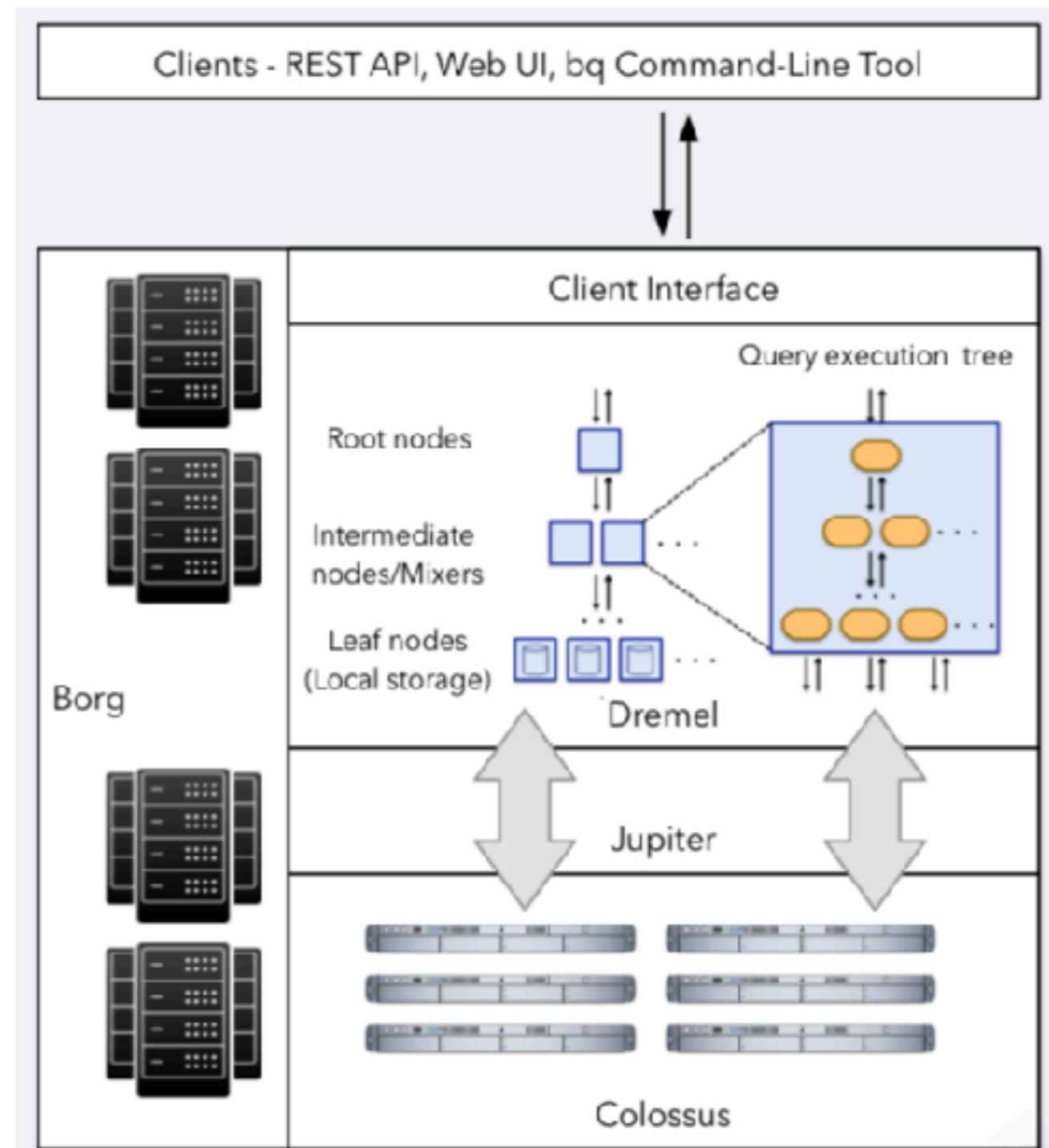


- Se separa computación y almacenamiento
- Almacenamiento columnar
- Saca provecho de la red Petabit de Google



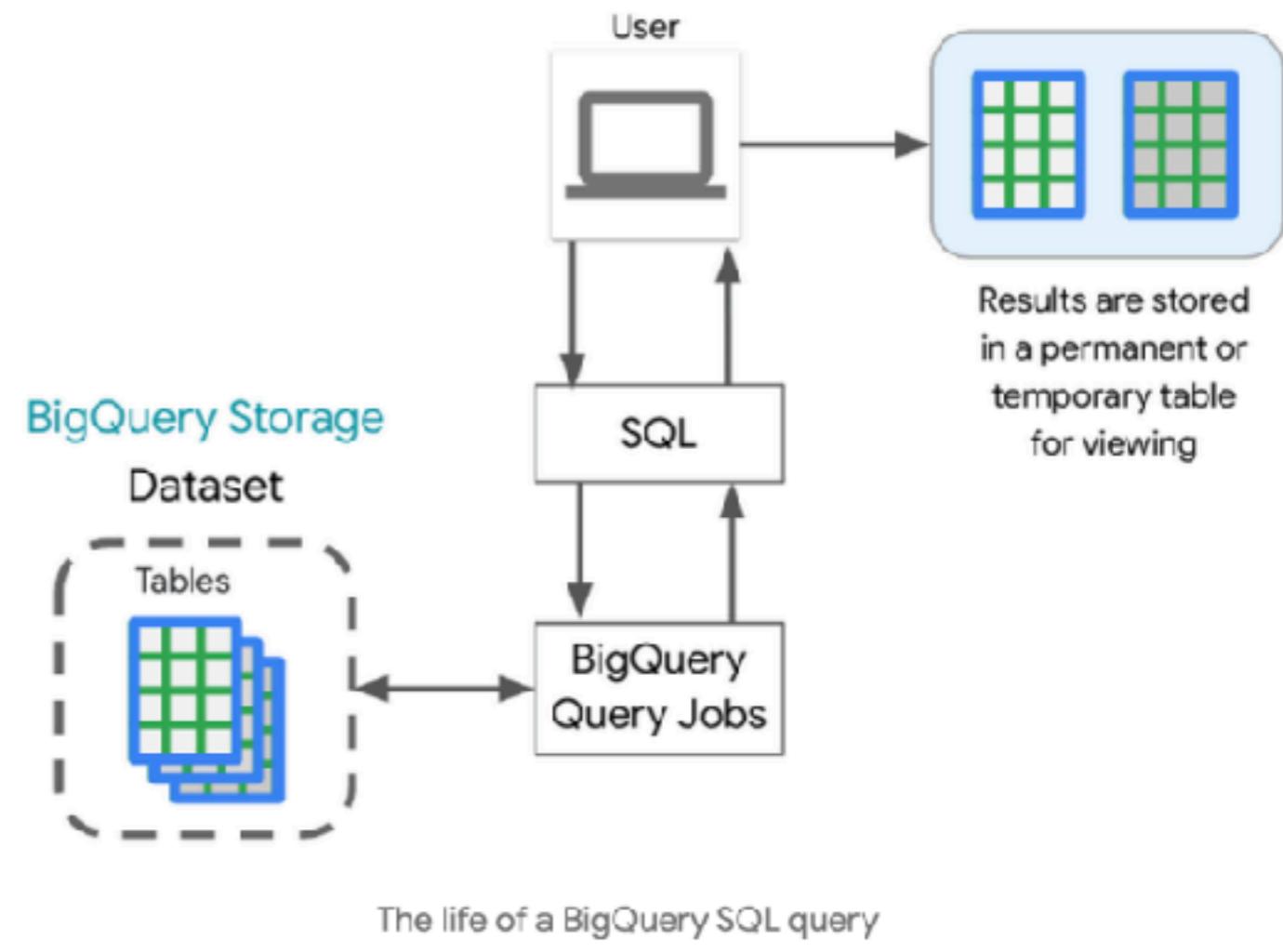


- **Dremel:** Convierte una consulta en un árbol de ejecución.
- **Colossus:** Almacenamiento!
- **Jupiter:** Red de Google.
- **Borg:** Asignación de recursos.





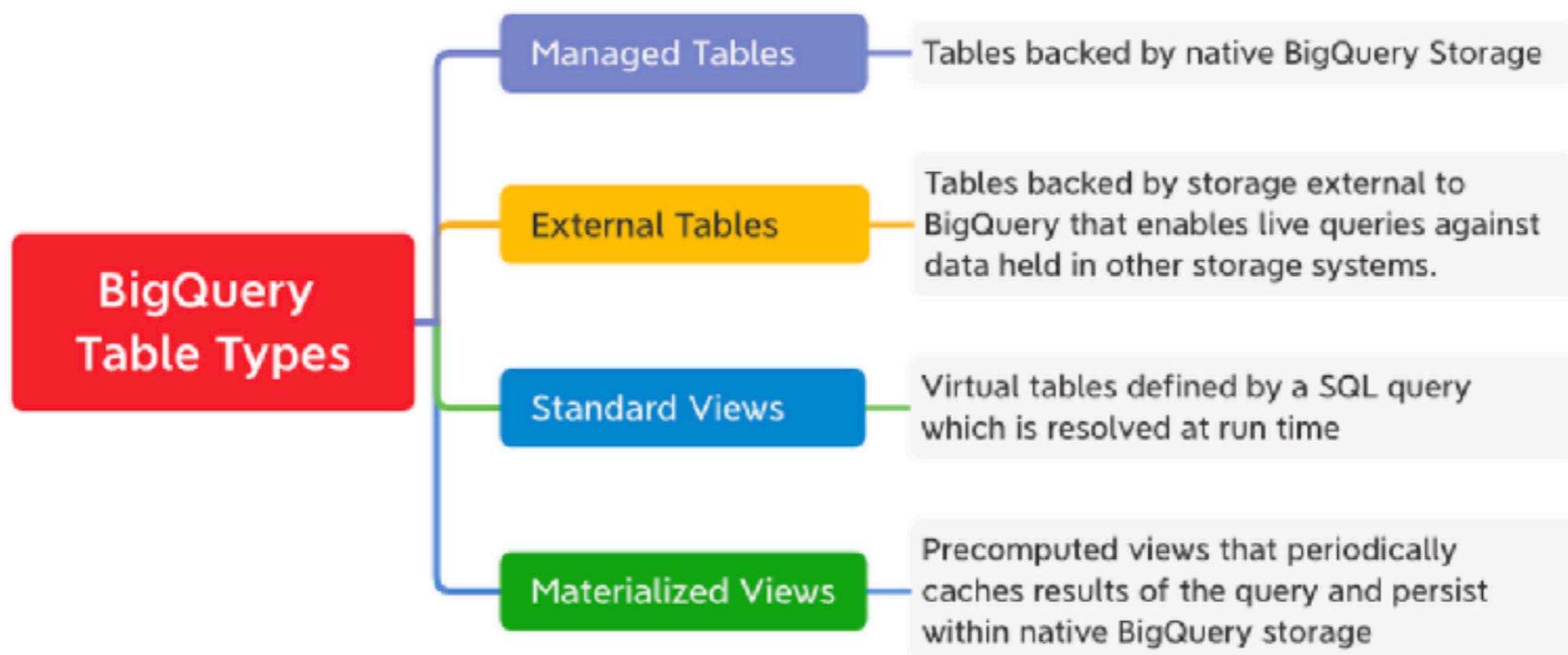
- Un usuario envía una consulta SQL a Big Query.
- BigQuery consulta las Tablas en el Dataset.
- Los resultados se almacenan en una tabla para ver los resultados.
- El usuario define que tipo de tabla almacena el resultado.
- Las consultas SQL quedan en historial, se pueden guardar y también se pueden compartir.



Source: Documentación GCP



Tipos de Tablas en BigQuery



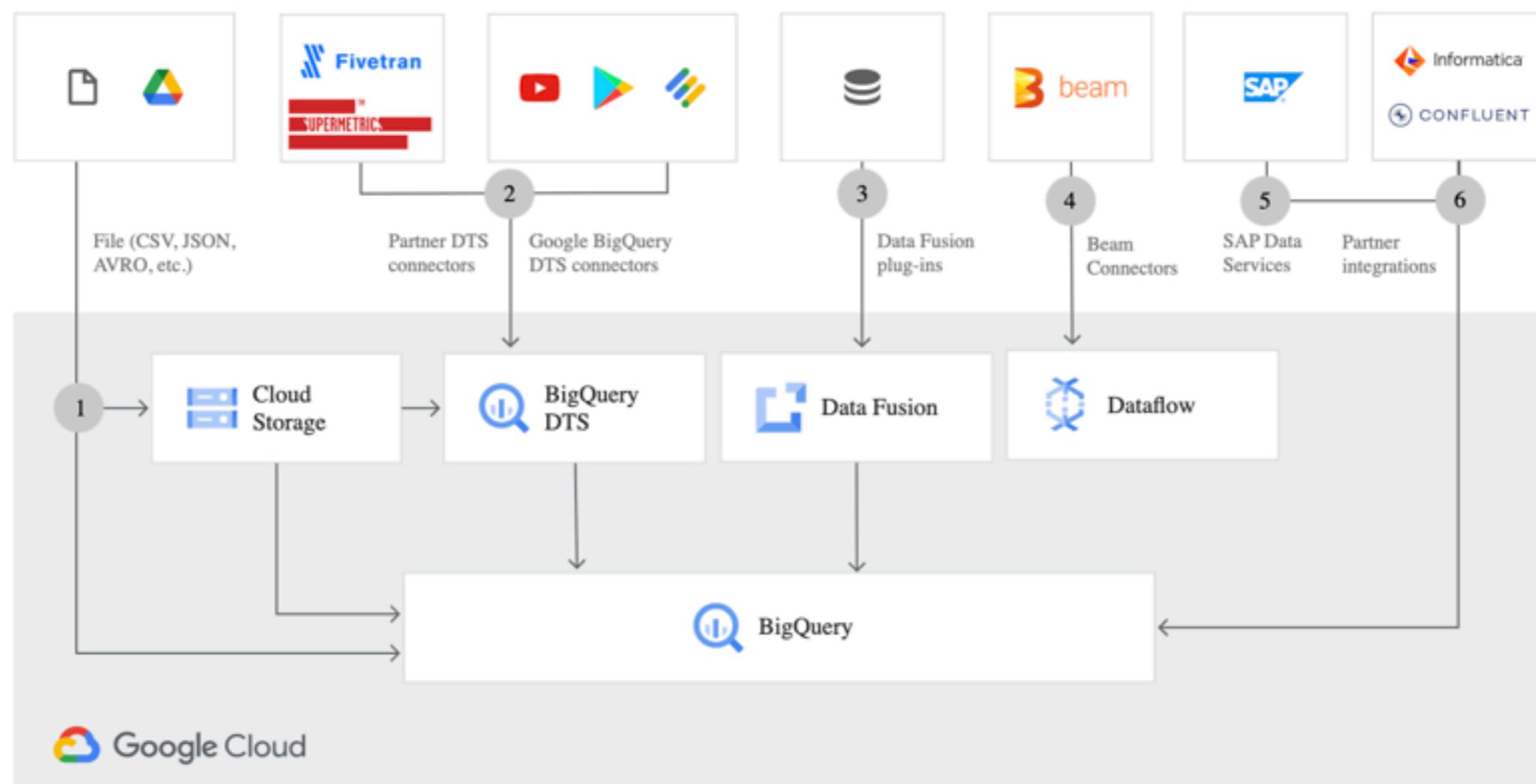
BigQuery Table Types



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática

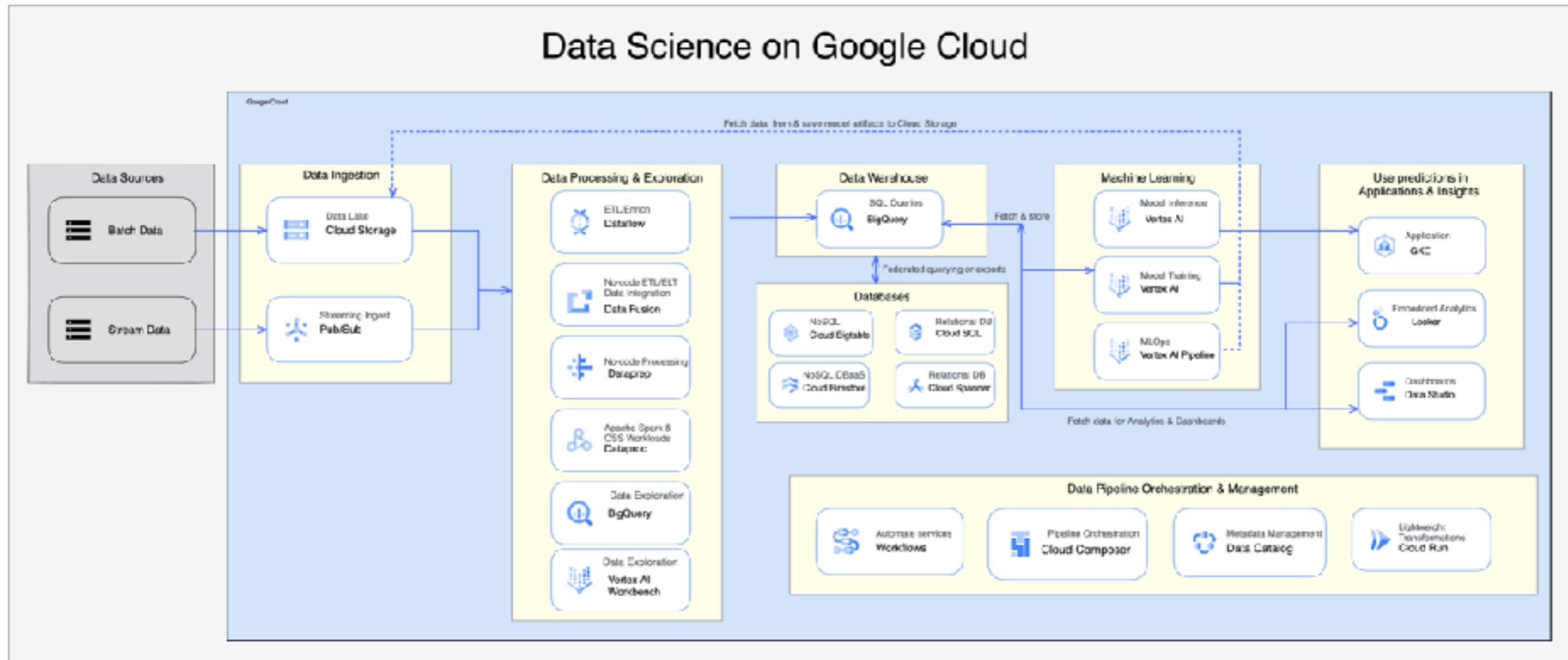
Como llevar datos a BigQuery





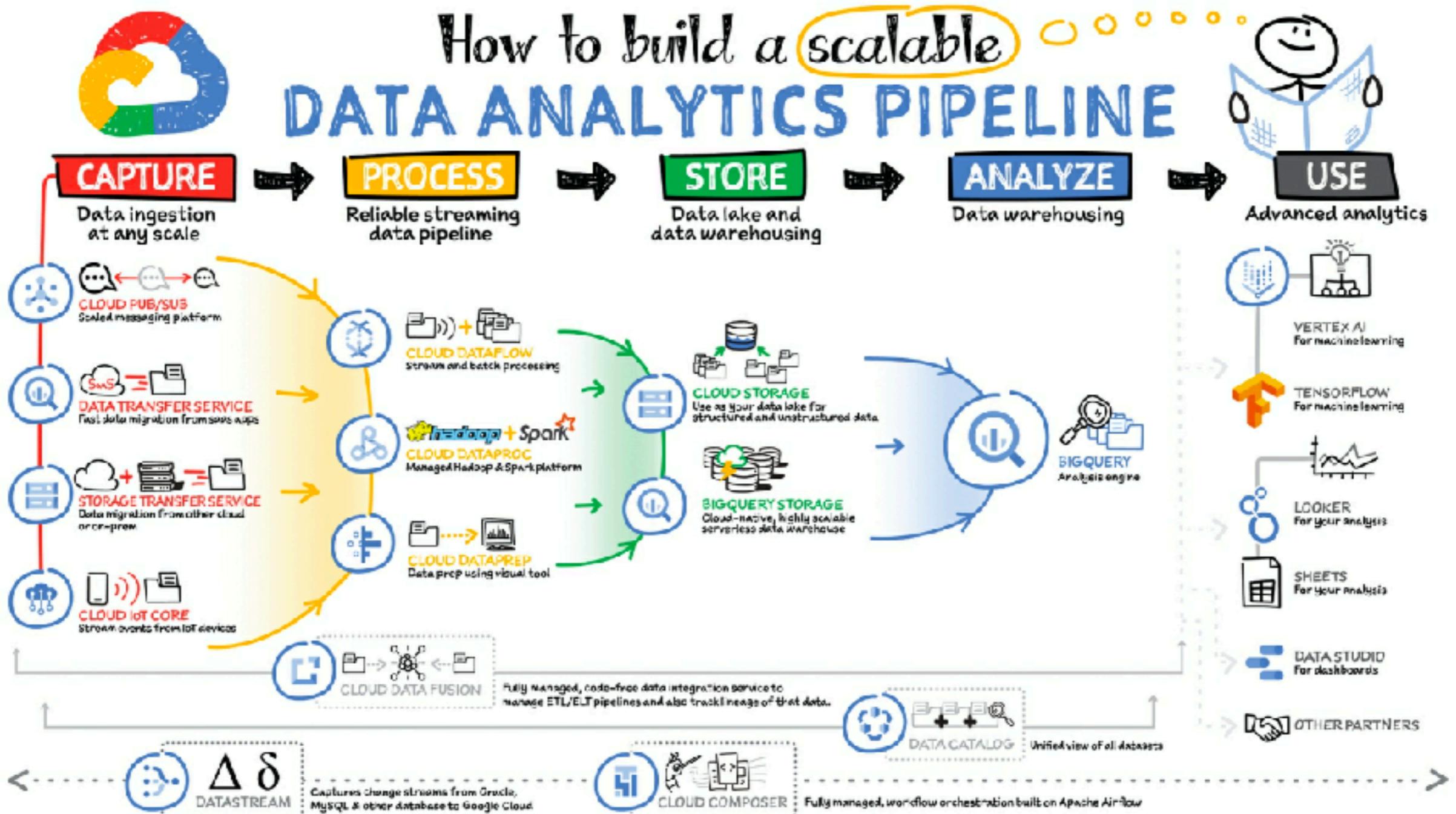
Modelos de Referencia

Data Science on Google Cloud





Modelos de Referencia



Source: Libro Visualizing Google Cloud Platform, Priyanka Vergadia, Wiley, 2022

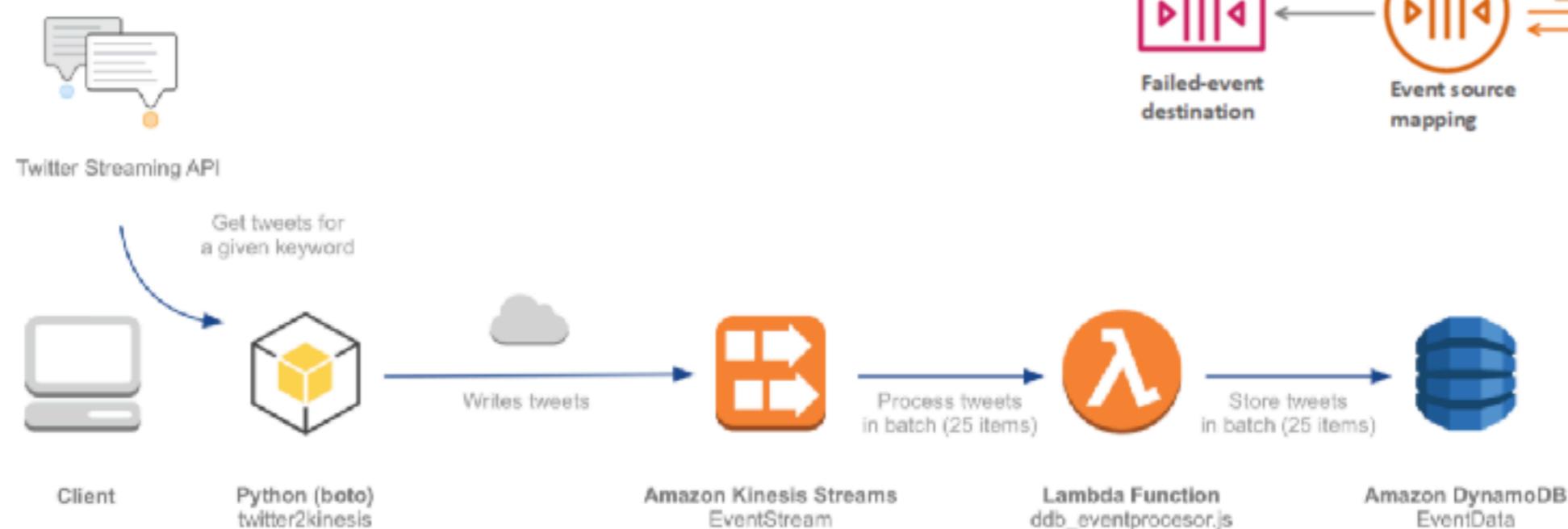


UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática

Más sobre Procesamiento

- AWS Lambda: Procesamiento sin servidores de eventos.
 - Máximo 15 min de procesamiento de 1 evento.
 - Máximo de ejecuciones concurrentes: 1,000 (puede incrementarse a 10x)



Event Source Mapping with Kinesis Stream

