Paradigmas de Procesamiento de Datos a Gran Escala

Un Recorrido por Enfoques, Ideas y Arquitecturas

Deborah Famadas Rodríguez

Universidad de la Habana

September 22, 2025

El Universo NoSQL: Más Allá de las Tablas

Tipos, Usos y Ejemplos (Parte 1)

Las bases de datos **NoSQL** ("Not Only SQL") ofrecen modelos de datos flexibles y escalan horizontalmente, siendo ideales para las necesidades del Big Data y las aplicaciones web modernas.

1. Bases de Datos Clave-Valor

Modelo: El más simple. Almacena datos en un diccionario (llave única, valor opaco).
Uso Ideal: Cachés de alta velocidad, almacenamiento de sesiones de usuario, carritos de compra.
Ejemplos: Redis, Riak, Amazon DynamoDB.

2. Bases de Datos de Documentosb

Modelo: Almacena datos en documentos semi-estructurados como JSON o BSON.
Uso Ideal: Sistemas de gestión de contenido (CMS), catálogos de productos, perfiles de usuario.
Ejemplos: MongoDB, CouchDB, Elasticsearch.

3. Bases de Datos Columnares

Modelo: Almacena datos en columnas en lugar de filas. Optimizado para agregaciones rápidas.
Uso Ideal: Data Warehousing, Business Intelligence (BI), análisis de grandes volúmenes de datos.
Ejemplos: Apache Cassandra, HBase, Google Bigtable.

4. Bases de Datos de Grafos

Modelo: Usa nodos y aristas para representar y almacenar relaciones complejas entre datos.
Uso Ideal: Redes sociales, sistemas de recomendación, detección de fraude.
Ejemplos: Neo4j, JanusGraph, Amazon Neptune.

El Desafío de la Escala Web

Una Solución Integral de Google

A principios de los 2000, para indexar la web, Google no creó una única herramienta, sino un ecosistema de tres tecnologías diseñadas para trabajar en sinergia sobre hardware económico.

Almacenamiento Distribuido + Cómputo Distribuido + Base de Datos Distribuida

Juntos, estos componentes permitieron procesar petabytes de datos de forma tolerante a fallos, sentando las bases de la ingeniería de datos moderna.

Los Tres Pilares Fundacionales

Google File System (GFS) - 2003

El Almacenamiento.

- Diseñado para archivos gigantes.
- Distribuido y tolerante a fallos.

MapReduce - 2004

El Cómputo. ⚙

- Un modelo simple para computación paralela.
- Abstrae la complejidad de la distribución.

Bigtable - 2006

La Base de Datos. 🗄

- Almacén NoSQL para datos estructurados.
- Escalabilidad masiva para acceso rápido.

La Democratización: El Ecosistema Hadoop

De la Inspiración a la Realidad Open-Source

Inspirado por los *papers* de Google, el proyecto Nutch implementó versiones de código abierto:

- GFS → **HDFS** (Hadoop Distributed File System).
- MapReduce → **Hadoop MapReduce**.
- Bigtable → **HBase**.

Nació así **Apache Hadoop**, la plataforma que democratizó el procesamiento de Big Data, permitiendo a miles de empresas analizar grandes volúmenes de datos.

MapReduce: La Filosofía de la Abstracción

El Contrato del Desarrollador

El framework le promete al programador: "Tú me das tu lógica de negocio en dos funciones simples (`map` y `reduce`). Yo me encargo del resto".

El framework gestiona automáticamente:

- Paralelización y distribución de tareas.
- Balanceo de carga.
- Tolerancia a fallos (re-ejecución de tareas).
- Localidad del dato (mover el cómputo al dato).

El Flujo de MapReduce

Función `map`

Aplica una función a cada elemento de entrada de forma independiente y en paralelo.

$$\texttt{map(k1, v1)} \rightarrow \texttt{list(k2, v2)}$$

Función `reduce`

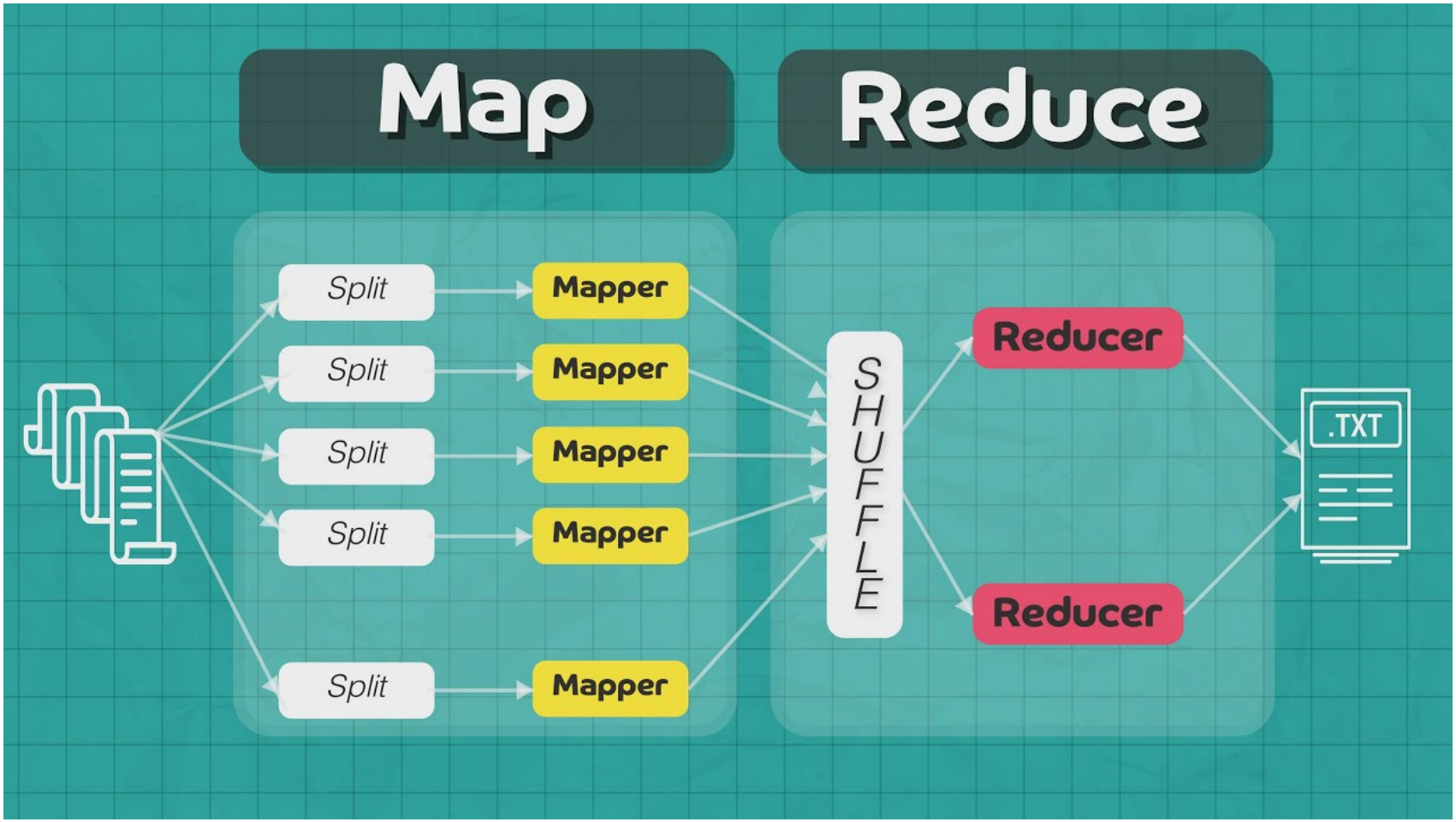
Agrega todos los valores asociados a una misma clave para producir un resultado final.

$$\texttt{reduce(k2, list(v2))} \rightarrow \texttt{list(k3, v3)}$$

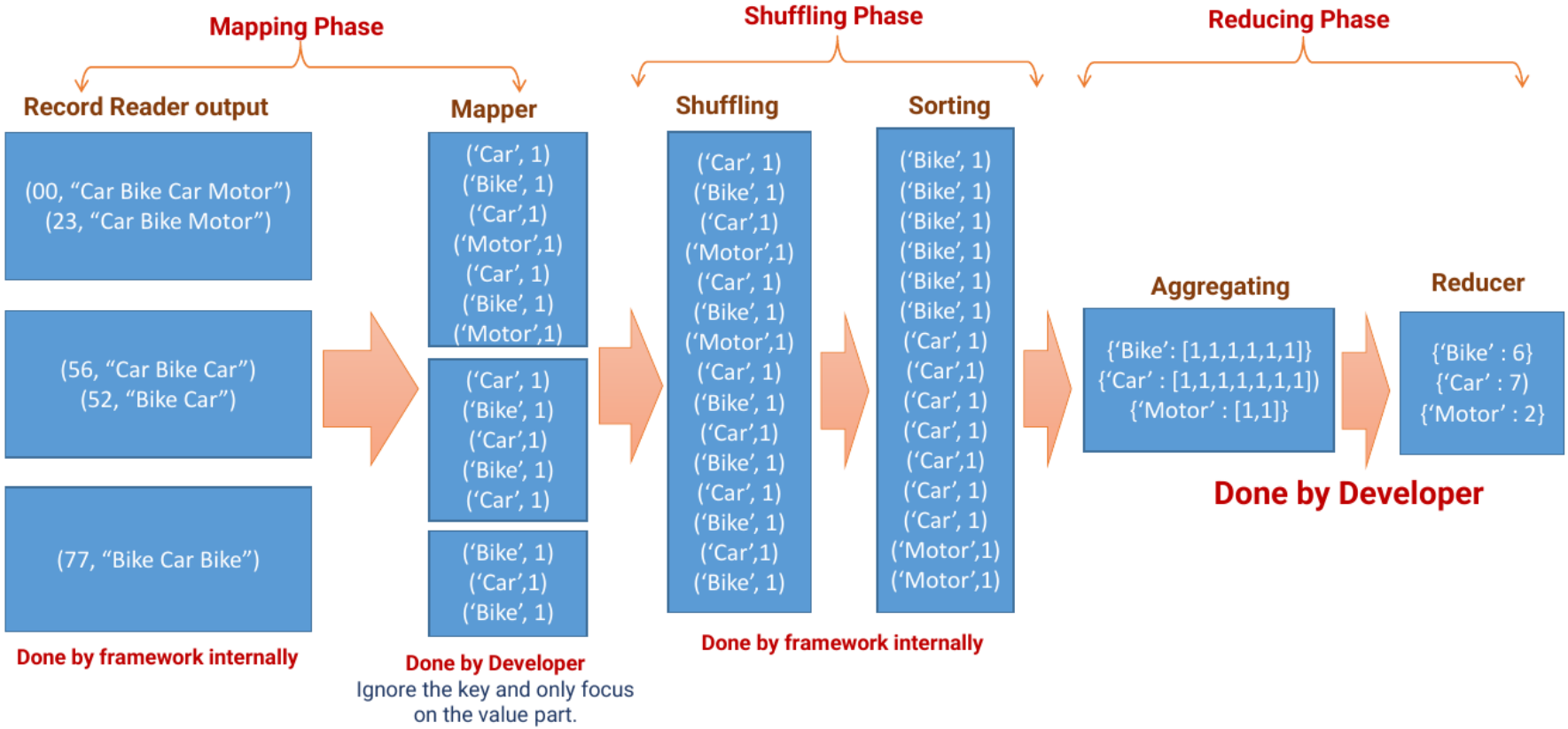
Ejemplo Clásico: Conteo de Palabras

Input: `"El perro come perro"`
Map Output: `[("El", 1), ("perro", 1), ("come", 1), ("perro", 1)]`
Shuffle & Sort: Agrupa por clave → `("perro", [1, 1])`, `("El", [1])`, ...
Reduce Output: `("perro", 2)`, `("El", 1)`, ...

El Flujo de MapReduce



El Flujo de MapReduce



Optimizando MapReduce: El Rol del *Combiner*

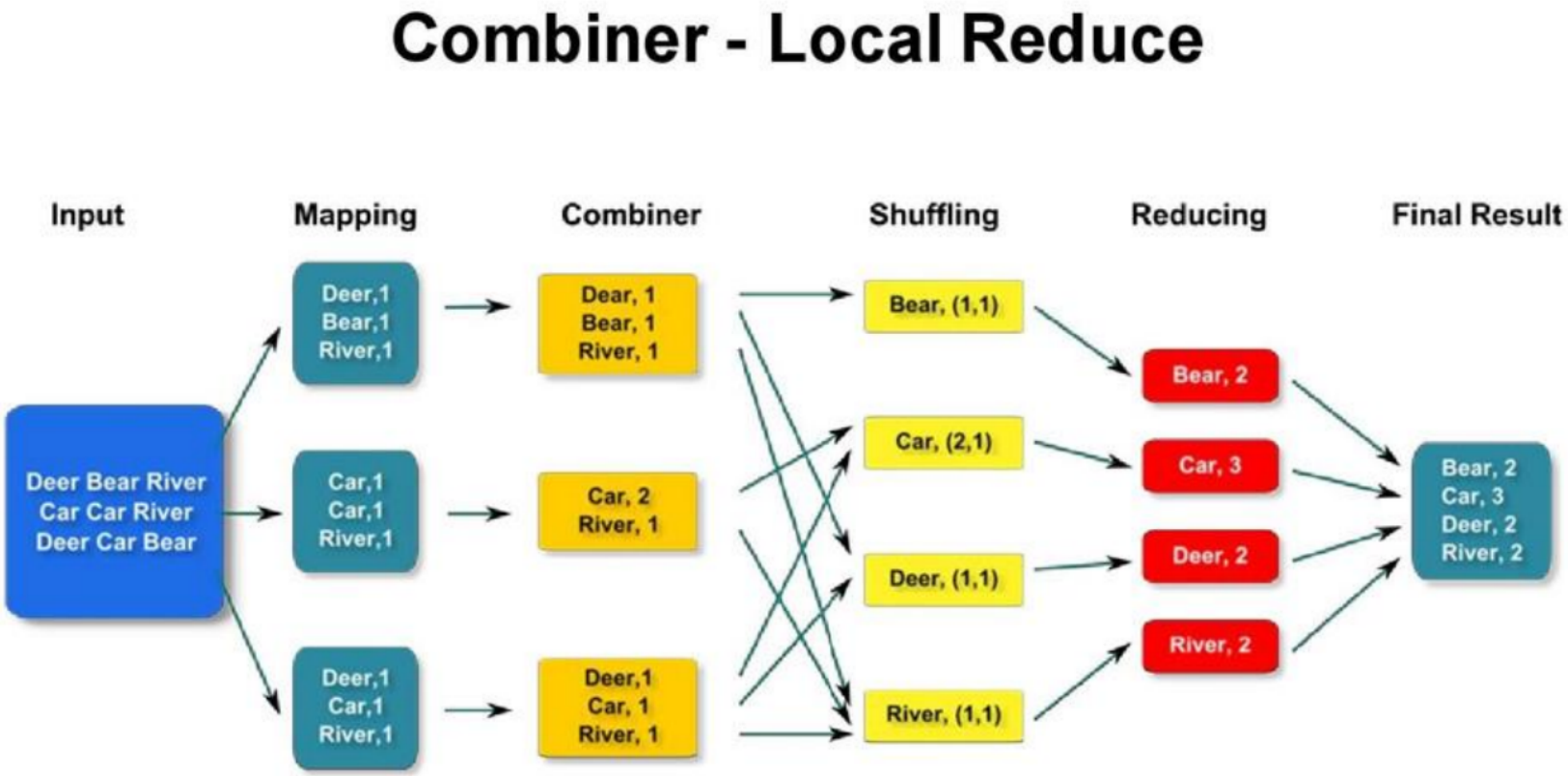
El **Combiner**, también conocido como *"mini-reducer"*, es una optimización clave para reducir la congestión de red en trabajos de MapReduce.

- **Problema:** Los Mappers pueden generar una gran cantidad de datos intermedios, saturando la red al enviarlos a los Reducers.
- **Solución:** El Combiner resume la salida del Mapper **localmente** en cada nodo, agrupando registros con la misma clave *antes* de que sean enviados por la red.

Flujo de Trabajo Simplificado

Mapper $\rightarrow$ **Combiner (Local)** $\rightarrow$ *Shuffle&Sort* $\rightarrow$ *Reducer*

El Flujo de MapReduce



Análisis del Paradigma MapReduce

El Cuello de Botella del Disco

La fase intermedia de "Shuffle & Sort" es el corazón del framework, pero también su principal característica definitoria en cuanto a rendimiento.

- **Intensivo en Red y Disco:** Los resultados intermedios de los mappers se escriben en disco local, se envían por la red, y los reducers los leen de vuelta del disco.
- **Escalabilidad Extrema:** Puede escalar a miles de nodos.
- **Robustez:** Su dependencia del disco lo hace muy tolerante a fallos.
- **Ineficiente para Algoritmos Iterativos:** Cada iteración requiere un nuevo job MapReduce con lecturas/escrituras a disco.

Ideal para: ETL masivo y cargas de trabajo de alto rendimiento (throughput) no urgentes.

Respondiendo a Nuevas Necesidades

Las Limitaciones de MapReduce Inspiraron un Nuevo Enfoque

MapReduce era excelente para ETL a gran escala, pero menos eficiente para:

- **Análisis Interactivo:** Los analistas no podían esperar minutos por cada consulta.
- **Machine Learning Iterativo:** Algoritmos como PageRank o K-Means requerían múltiples pasadas, con un coste de disco prohibitivo en cada iteración.

La Idea Clave de Spark

Un nuevo motor, diseñado para el **procesamiento en memoria**, que evita escribir en disco en los pasos intermedios, reutilizando los datos entre operaciones.

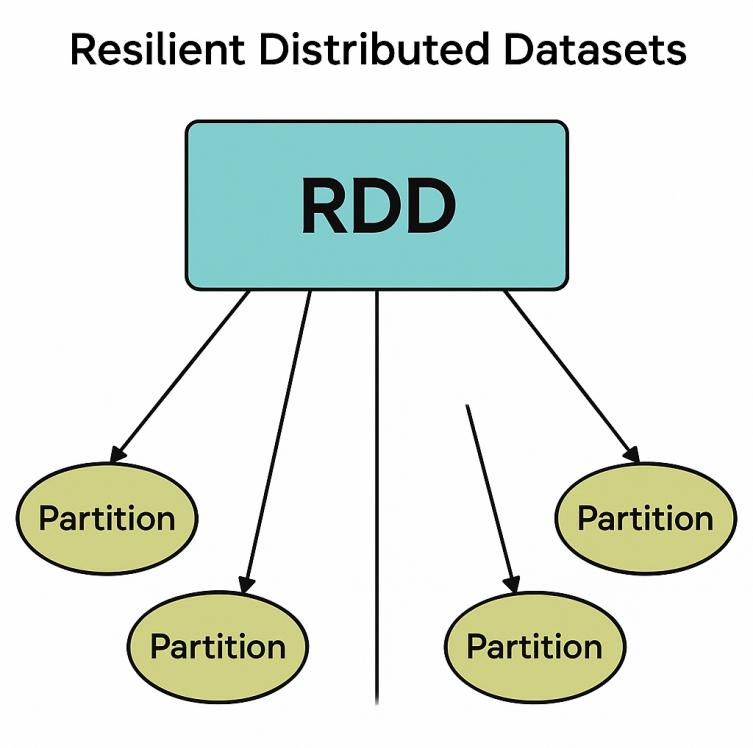
La Abstracción Clave: RDDs y DataFrames

Resilient Distributed Datasets (RDDs)

Una colección de objetos **distribuida**, **inmutable** y **tolerante a fallos** que puede ser procesada en paralelo.

- **Tolerancia a Fallos por Linaje:** En lugar de replicar datos, Spark recuerda la secuencia de transformaciones (el "linaje") que creó un RDD. Si un nodo falla, puede reconstruir las particiones perdidas re-ejecutando esas transformaciones.
- **Evaluación Perezosa (Lazy Evaluation):** Las transformaciones (`map`, `filter`) construyen un plan (DAG). La computación solo se dispara cuando se invoca una acción (`count`, `save`). Esto permite al motor optimizar el plan completo.

La Abstracción Clave: RDDs y DataFrames



DataFrames y Datasets

Abstracciones de nivel superior construidas sobre RDDs que añaden esquemas y un potente optimizador (Catalyst), facilitando el trabajo con datos estructurados.

Procesamiento de Flujos (Streaming)

- **¿Qué es?:** Procesa flujos de datos **infinitos y continuos** a medida que llegan, evento por evento.
- **Objetivo Principal: Baja latencia**. Diseñado para dar respuestas en milisegundos o segundos.
- **Casos de Uso Típicos:**
  - Detección de fraude en transacciones financieras.
  - Monitorización de sistemas en tiempo real.
  - Análisis de datos de sensores IoT.
  - Personalización de contenido web en vivo.
- **Herramientas Clave:** Apache Flink, Apache Kafka Streams, Spark Streaming.

Conceptos Fundamentales en Streaming

Ventanas (Windows)

Como los flujos son infinitos, las agregaciones (sumas, conteos) deben realizarse sobre subconjuntos acotados de datos.

- **Tumbling Window (Fija):** Ventanas de tamaño fijo que no se solapan (ej: cada 5 minutos).
- **Sliding Window (Deslizante):** Ventanas de tamaño fijo que se solapan (ej: la actividad de los últimos 5 minutos, calculada cada minuto).
- **Session Window (Por Sesión):** Agrupa eventos por periodos de actividad, separados por inactividad.

Conceptos Fundamentales en Streaming

Gestión del Tiempo y Estado

- **Marcas de Agua (Watermarks):** Mecanismo para manejar el retraso en la llegada de eventos y saber cuándo una ventana está "completa" y puede ser procesada.
- **Procesamiento con Estado (Stateful):** Capacidad del sistema de recordar información de eventos pasados para usarla en el procesamiento de eventos futuros (ej: un contador).

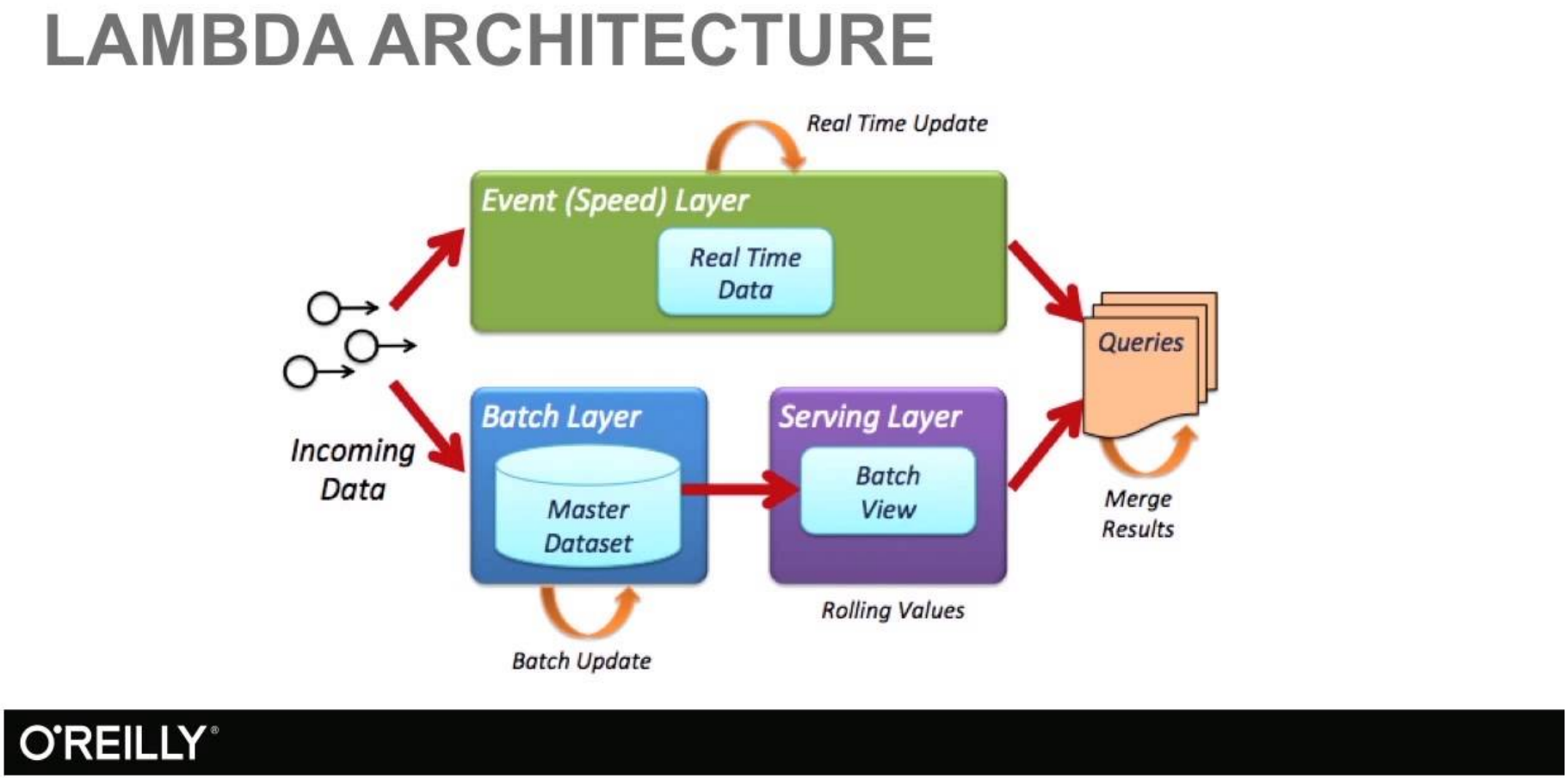
Arquitecturas Híbridas: Lambda y Kappa

Arquitectura Lambda

Combina una capa Batch y una capa de Velocidad (Streaming).

- **Capa Batch:** Reprocesa todos los datos para obtener vistas 100% precisas.
- **Capa de Velocidad:** Procesa datos en tiempo real para vistas de baja latencia.
- **Ventaja:** Muy robusta.
- **Desventaja:** Compleja, requiere duplicar la lógica.

Lambda


LAMBDA ARCHITECTURE
Real Time Update
Event (Speed) Layer
Real Time Data
Queries
Incoming Data
Batch Layer
Master Dataset
Serving Layer
Batch View
Merge Results
Rolling Values
Batch Update
O'REILLY

Arquitecturas Híbridas: Lambda y Kappa

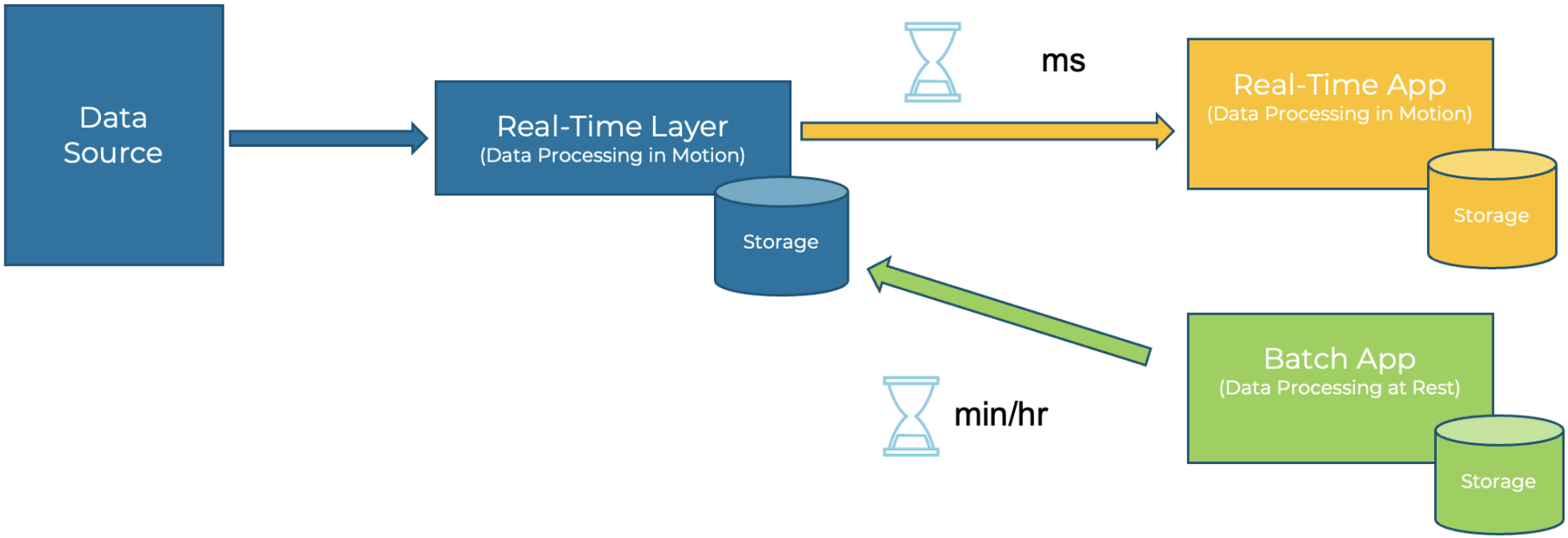
Arquitectura Kappa

Simplifica la arquitectura tratando todo como un flujo.

- Utiliza un único motor de streaming.
- Los recálculos (el rol de la capa batch) se hacen reprocesando el flujo de eventos desde el principio.
- **Ventaja:** Mucho más simple, sin código duplicado.
- **Requisito:** Un motor de streaming muy rápido y un sistema de almacenamiento de eventos (como Apache Kafka) que permita la relectura.

Kappa Architecture

One pipeline for real-time and batch consumers



Procesamiento de Grafos

Cuando las relaciones importan más que los datos

Paradigma especializado para datos interconectados

Modelos de computación

- Modelo Pregel (think like a vertex)
- Traversales de grafos
- Pattern matching

Algoritmos típicos

- PageRank
- Detección de comunidades
- Camino más corto
- Recomendación

Optimizaciones especializadas

- Particionamiento de grafos
- Estrategias de traversación
- Compresión de grafos

Casos de uso

- Redes sociales
- Sistemas de recomendación
- Detección de fraudes
- Bioinformática

Herramientas representativas

Modelo de Computación Pregel

Think like a vertex

Paradigma de computación inspirado en MapReduce pero optimizado para grafos

Fases de ejecución

1. Inicialización
2. Supersteps iterativos
3. Votación para terminar
4. Finalización

Operaciones por vértice

- Recibir mensajes
- Procesar mensajes
- Enviar mensajes a vecinos
- Modificar estado local

Ventajas

- Escalabilidad para grafos grandes
- Expresividad para algoritmos iterativos
- Tolerancia a fallos integrada

Comparación de Paradigmas

Conclusión

No existe "la mejor" herramienta, sino el enfoque más apropiado para el problema. Los ecosistemas modernos combinan varios de estos paradigmas.

Preguntas y Discusión

Gracias por su atención.