



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Almacenamiento y bases de datos para sistemas Big Data

PH.D. Erika Rosas Olivos

[erosas@inf.utfsm.cl](mailto:erosas@inf.utfsm.cl)

Septiembre 2023



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Almacenamiento

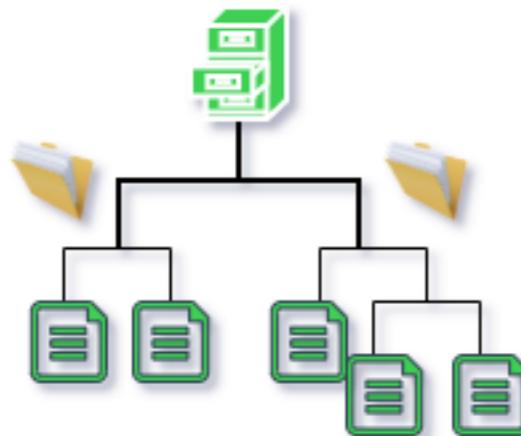
- ¿Porqué importa?
- ¿Cómo manejan ustedes el almacenamiento?



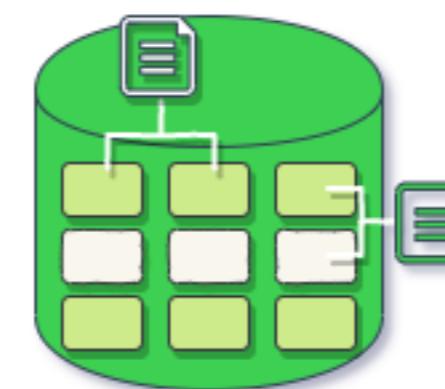
# Tipos de Almacenamiento

- Podemos encontrar 3 tipos de almacenamientos, los almacenamientos de archivos, los de bloques, y los de objetos.

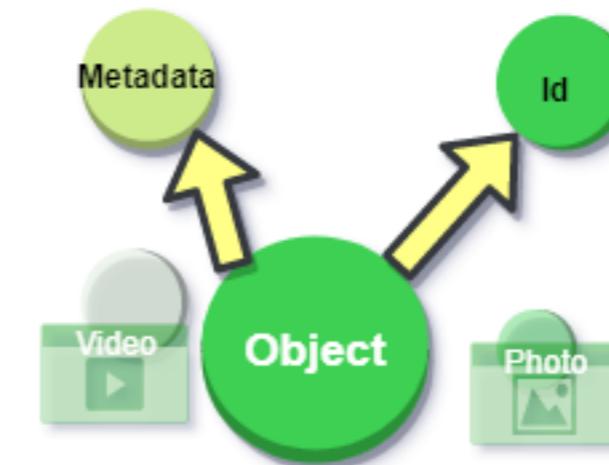
## File Store



## Block Store



## Object Store

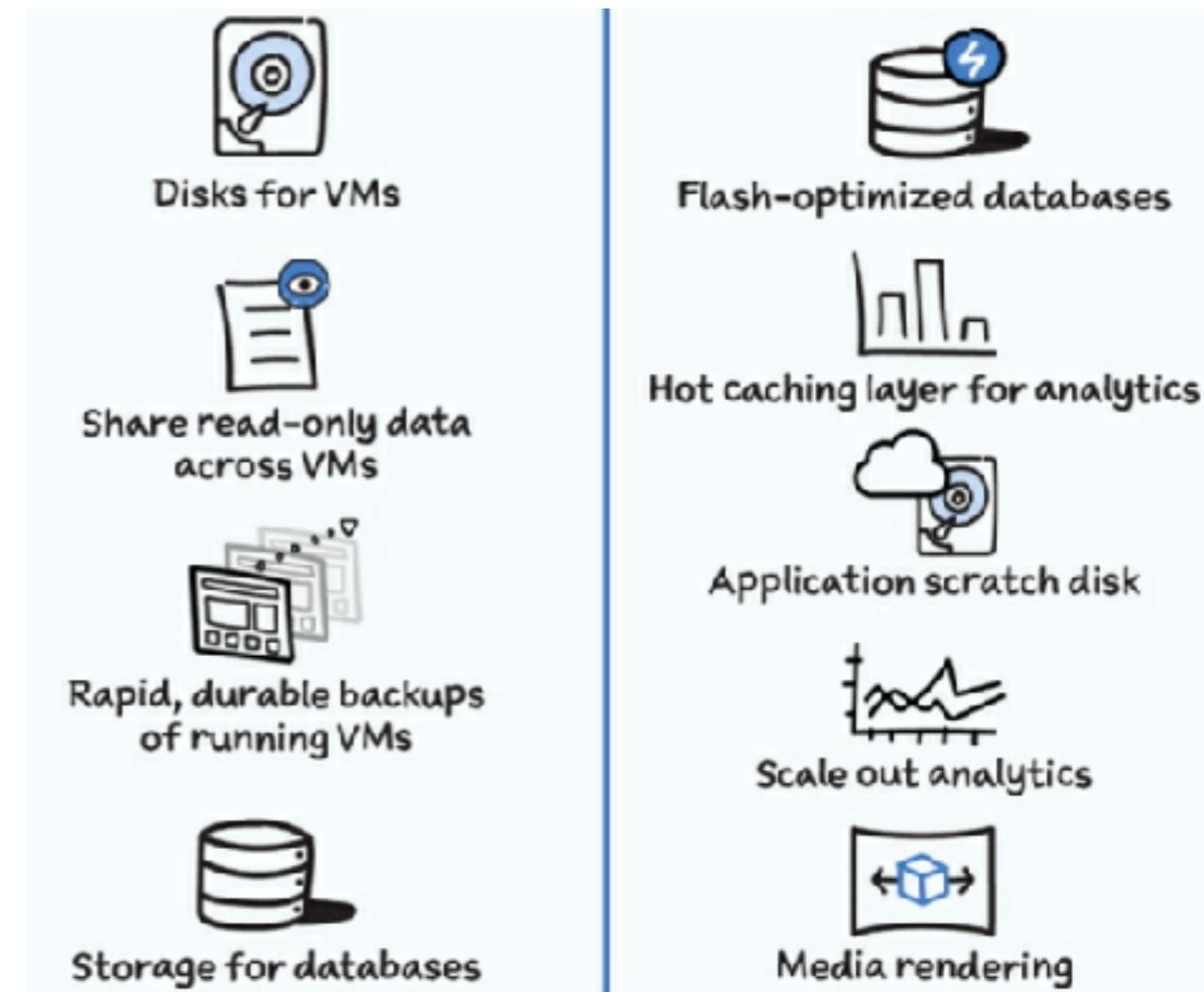


<https://rajkumaraug20.medium.com/file-storage-vs-block-storage-vs-object-storage-2519031a2646>

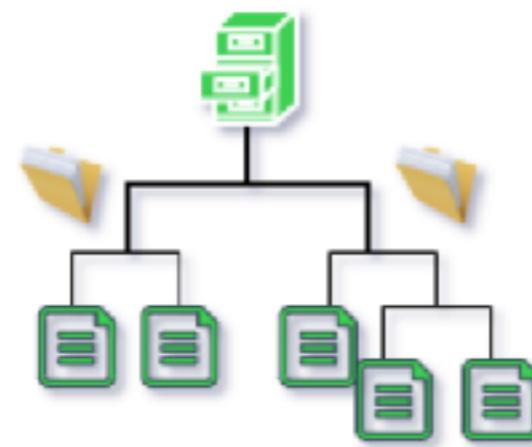


# Almacenamiento de Bloques

- Los archivos son particionados en bloques de igual tamaño, cada uno con su propia dirección.
- Sin información adicional como metadatos ni estructura de archivos.
- Se accede con llamadas del sistema operativo a los bloques.
- Usado cuando se requiere alto rendimiento y bajas latencias.
- Sirve para:
  - Bases de datos
  - Caché
  - Discos de máquinas virtuales

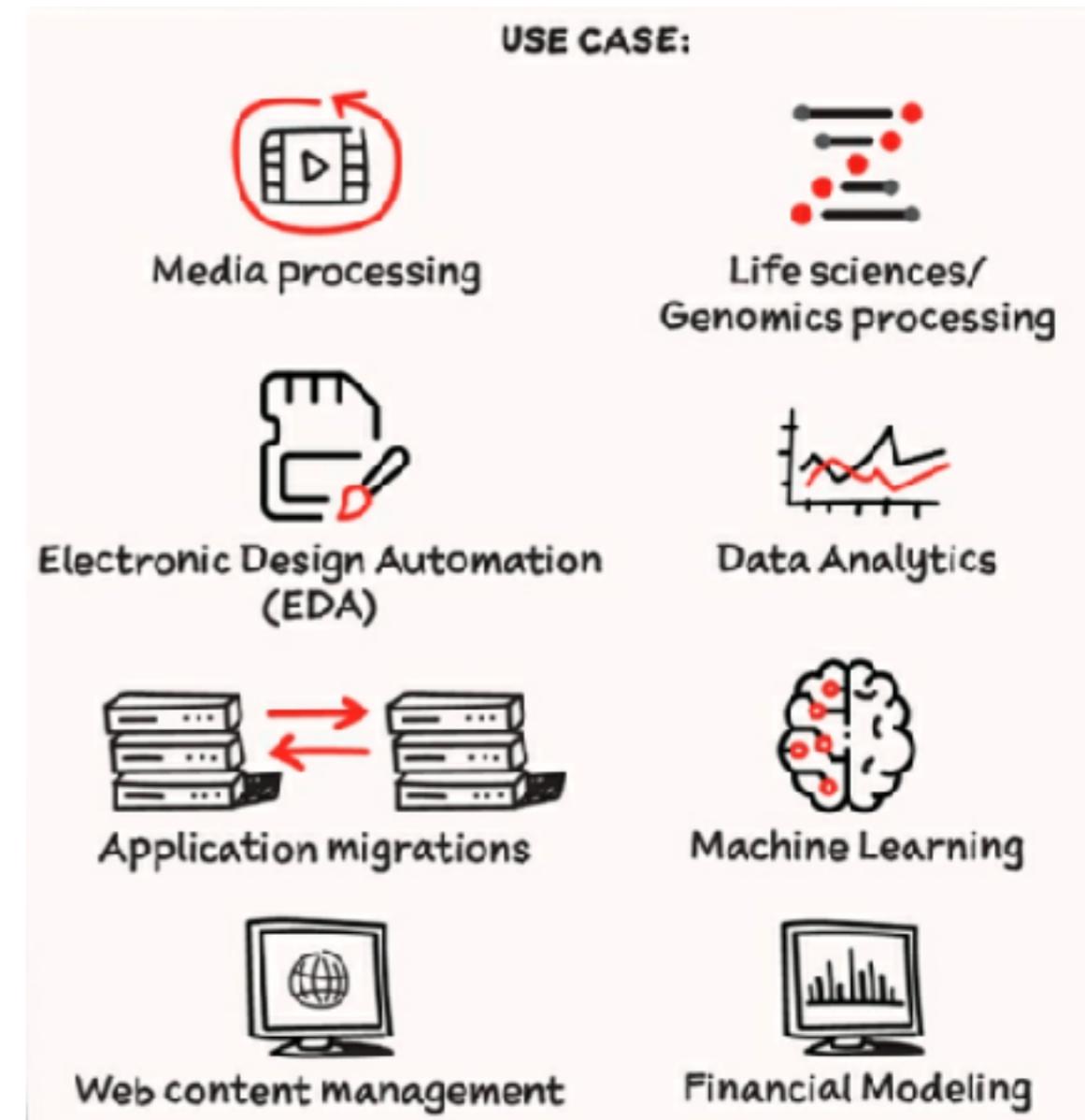


Source: Libro Visualizing Google Cloud Platform,  
Priyanka Vergadia, Wiley, 2022

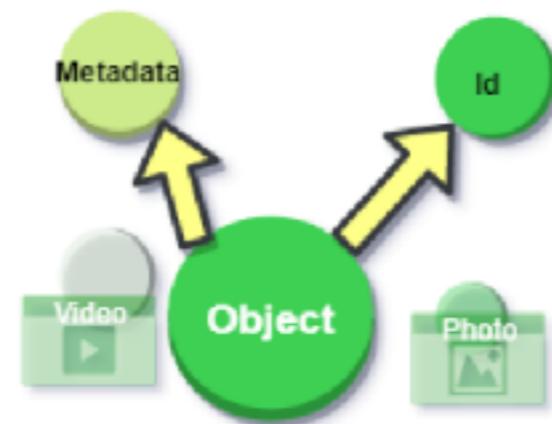


# Almacenamiento de Archivos

- Almacenamiento de archivos compartido (Network Attached Storage).
- Los archivos son almacenados y organizados en carpetas.
- Las carpetas están organizadas en una jerarquía de directorios y subdirectorios.
- Para datos no estructurados.
- Se accede con un PATH.
- Sirve para:
  - Compartir datos entre usuarios
  - Administración de contenido Web



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



# Almacenamiento de Objetos

- Se almacenan objetos binarios, blobs y datos no estructurados:
  - Tiene ID, metadatos, atributos y los datos.
- Se acceden sobre HTTP/s usando API & URI.
- Casos de uso:
  - Data lake
  - Almacenamiento de Videos e imágenes
  - Páginas Web
  - Logs, respaldo
  - Alta disponibilidad y durabilidad



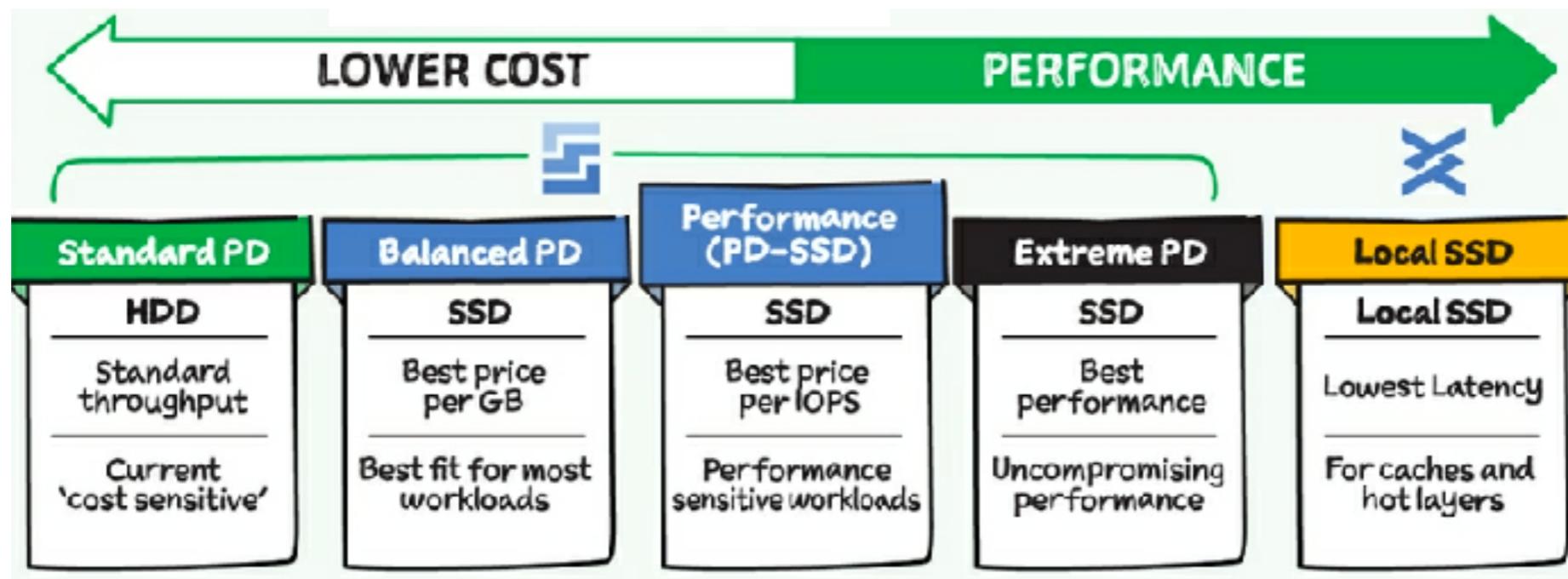
Source: Libro Visualizing Google Cloud Platform, Priyanka Vergadia, Wiley, 2022



# Ejemplo Almacenamiento en Bloque



- Persistent Disk:
  - Solución para altas tasas de IOPS.
  - Solución para baja latencia.
  - Almacenamiento de bajo costo.
  - Los discos son independientes de las instancias a las que están unidas.
  - Se puede cambiar su tamaño de manera transparente.

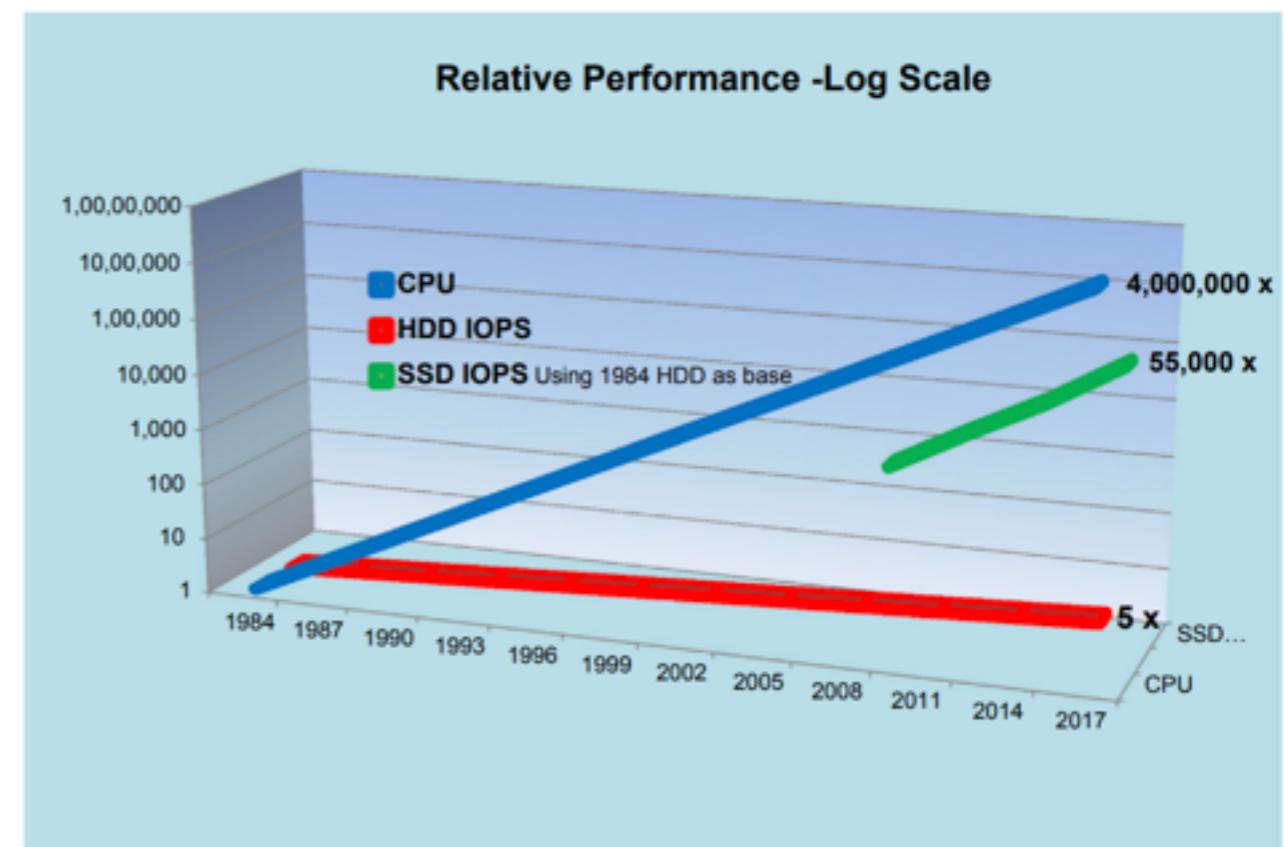
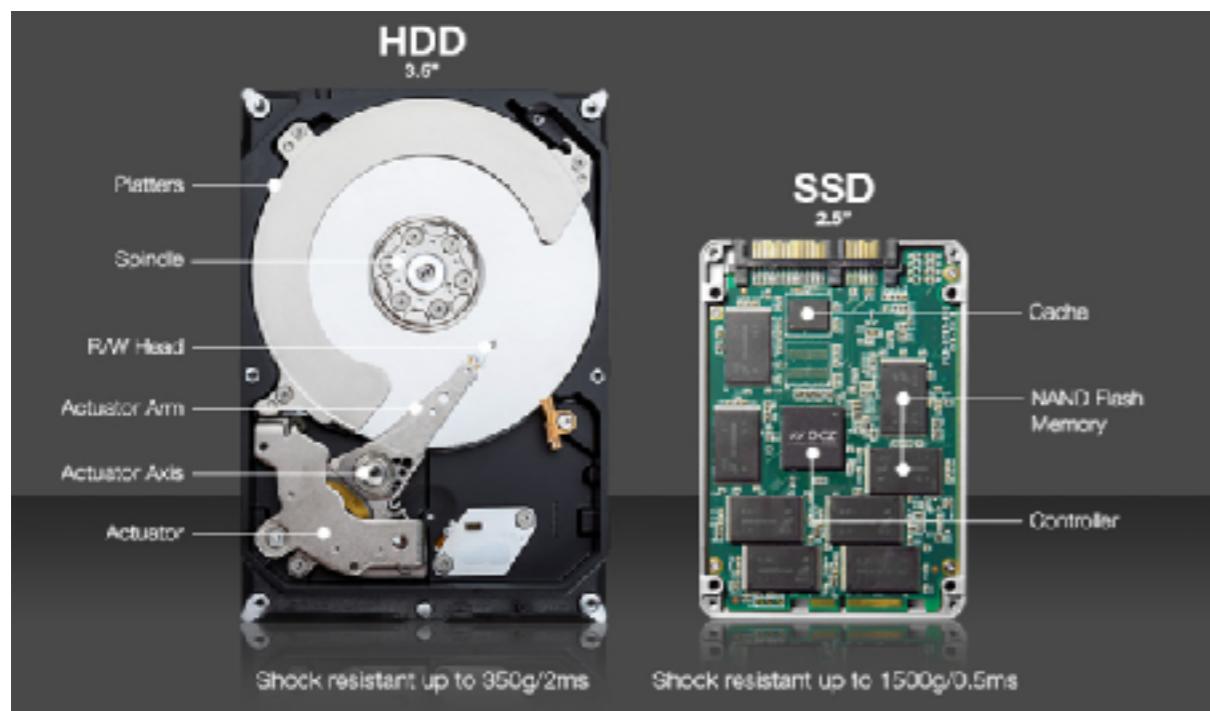




UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# HDD versus SSD



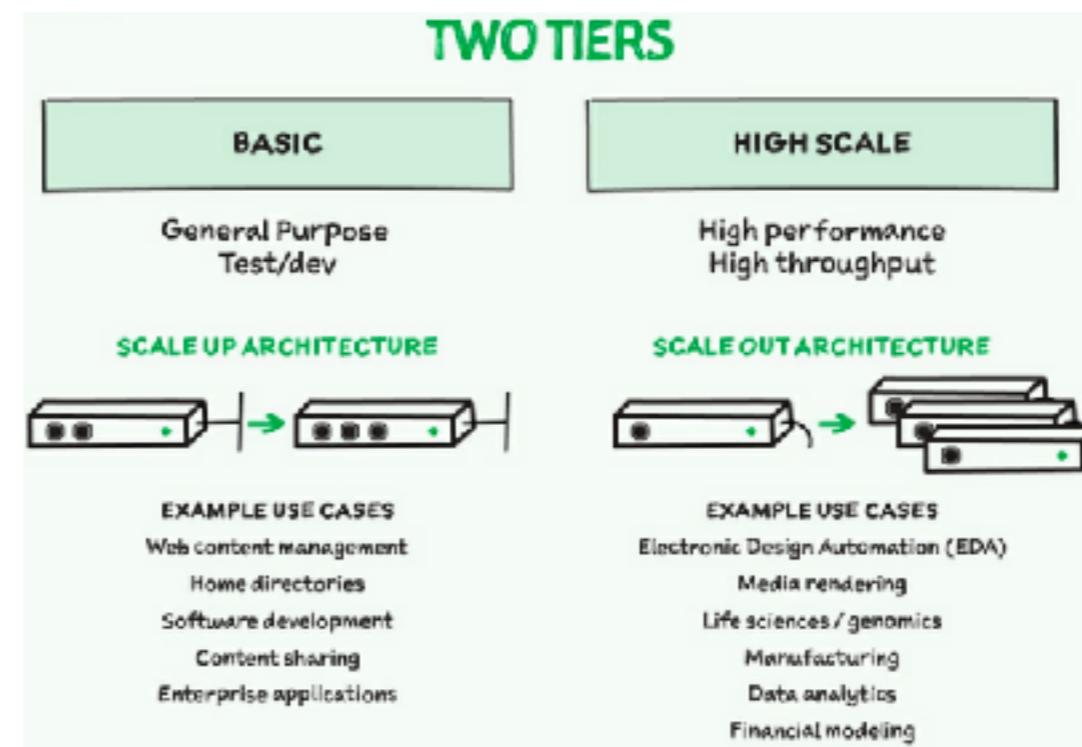
¿Cómo impacta y dónde?



# Ejemplo Almacenamiento de Archivos



- Filestore
  - Almacenamiento administrado en la Nube.
  - NAS: Network attached storage
  - Para datos no estructurados
  - Se puede montar en cualquier cosa que hable NFS.





# Sistemas de Archivos Distribuidos

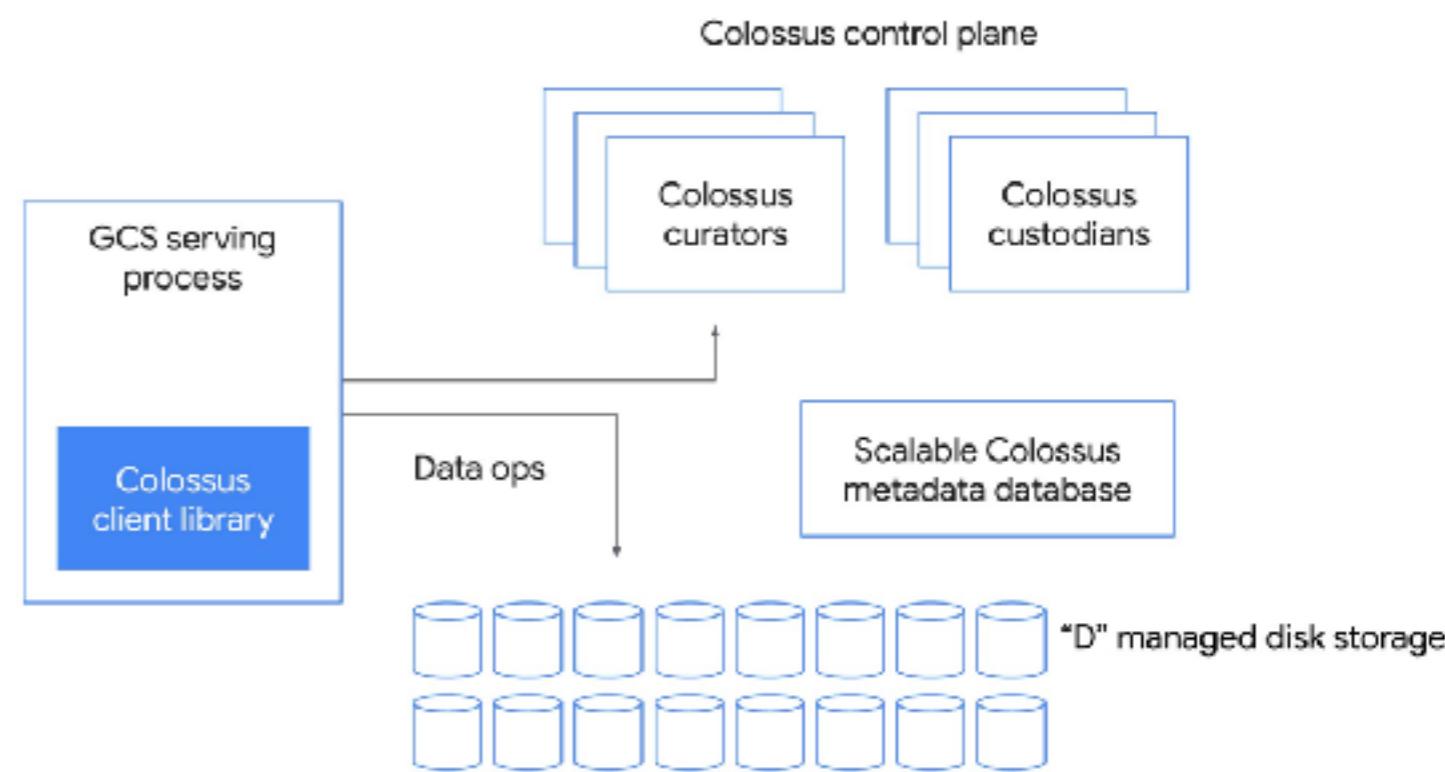
**Objetivo:** Tienen el propósito de emular la funcionalidad de un sistema de archivos no distribuido.

- **Sistemas de archivos** son responsables de la organización, almacenamiento, recuperación, nombramiento, compartir y proteger archivos.
- Los archivos contienen datos y atributos.
  - **Data:** secuencia de items de datos, accesibles por operaciones de lectura y escritura.
  - **Atributos:** Un solo registro que contiene información como largo del archivo, timestamps, tipo, dueño, etc.
- **Metadata:** término usando generalmente para referirse a toda la información entra almacenada por el sistema de archivos que es necesaria para su administración como los directorios.



# Colossus para escalabilidad masiva

- Sucesor de GFS con diseño que mejora escalabilidad y disponibilidad.
- Modelo de almacenamiento de metadatos distribuidos.
- **Librería cliente:** Compleja, la aplicación configura el tradeoff rendimiento - costo para diferentes tipos de cargas.
- **Plano de control:**
  - Curators: servicio de metadatos escalable consiste en muchos curadores.
  - Custodians: Balanceadores de espacio de disco y reconstrucción RAID. Corren en background.
- **Base de datos para metadatos:** Se almacenan en BigTable.
- **Servidores de Archivos D:** Los datos fluyen directamente entre clientes y servidores de archivos.



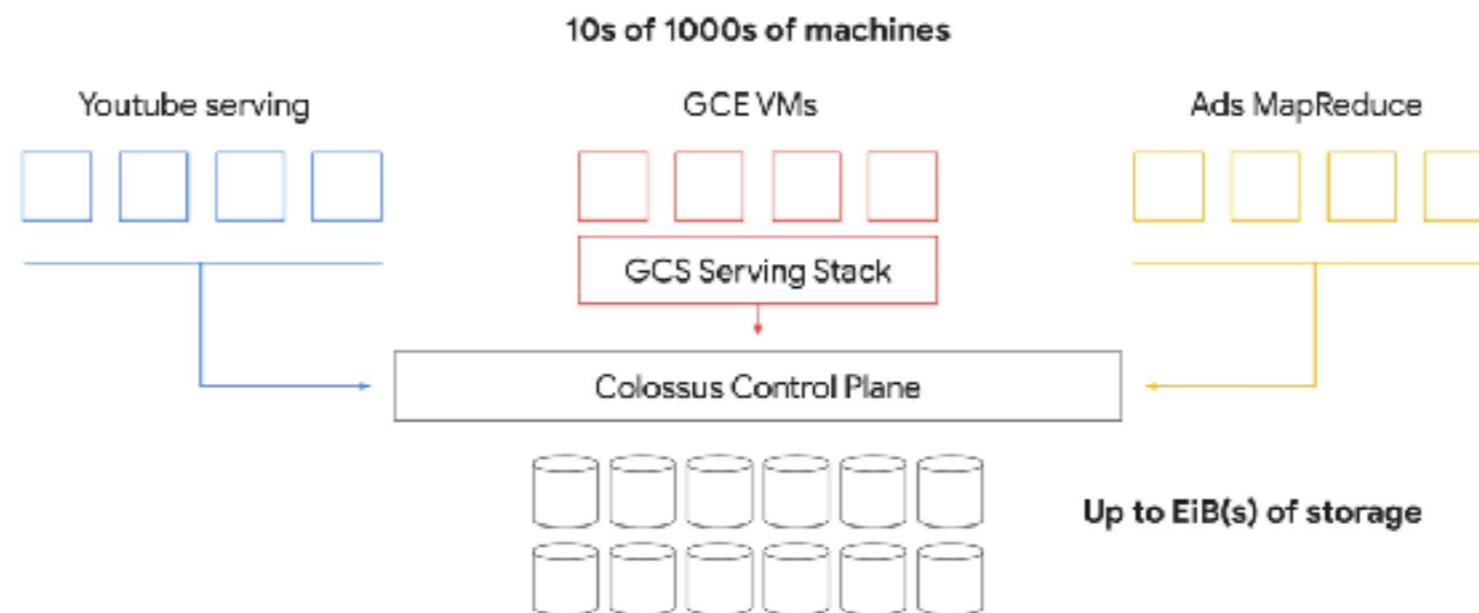
<https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>



# Colossus para escalabilidad masiva

- Infraestructura para múltiples servicios.
- Para todos los servicios de almacenamiento de Google se usa **Spanner** (almacenamiento metadata de localización y permisos), **Borg** (aprovisionamiento de recursos), y **Colossus**.
- Mezcla de discos y almacenamiento flash en varios tamaños y tipos. Sistema ajusta a IO requerido, disponibilidad y durabilidad.
- Esto soporta una variedad de patrones y frecuencias de acceso.
- Ejemplo: hot data la ponemos en flash y es distribuida en el cluster para acceso paralelo.

## Typical cluster



<https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>



# Sistemas de archivo distribuidos para Big Data



“You’re probably not going to implement your own storage engine from scratch, but you do need to select a storage engine that is appropriate for your application, from the many that are available.”

In order to tune a storage engine to perform well on your kind of workload, you need to have a rough idea of what the storage engine is doing under the hood.

Martin Kleppmann, Designing Data-Intensive Applications, O'Reilly.

lustre®





# ¿Qué requerimientos priorizar?

- Métricas: Número de operaciones de entrada salida por segundo (Throughput IOPS), MB/s en lecturas, en escrituras, ya sea con acceso secuencial o aleatorio, latencia, etc.
- Tamaño del metadata? Tamaño de los archivos?
- Confidencialidad?
- Soporte de operaciones adicionales? Bloqueo de archivos, extensión de atributos, cuotas, permisos, links simbólicos, etc.



Jakob Blomer: Experiences on File Systems, 2015



- Métricas: Número de operaciones de entrada salida por segundo (Throughput IOPS), MB/s en lecturas, en escrituras, ya sea con acceso secuencial o aleatorio, latencia, etc.
- Tamaño del metadata? Tamaño de los archivos?
- Confidencialidad?
- Soporte de operaciones adicionales? Bloqueo de archivos, extensión de atributos, cuotas, permisos, links simbólicos, etc.



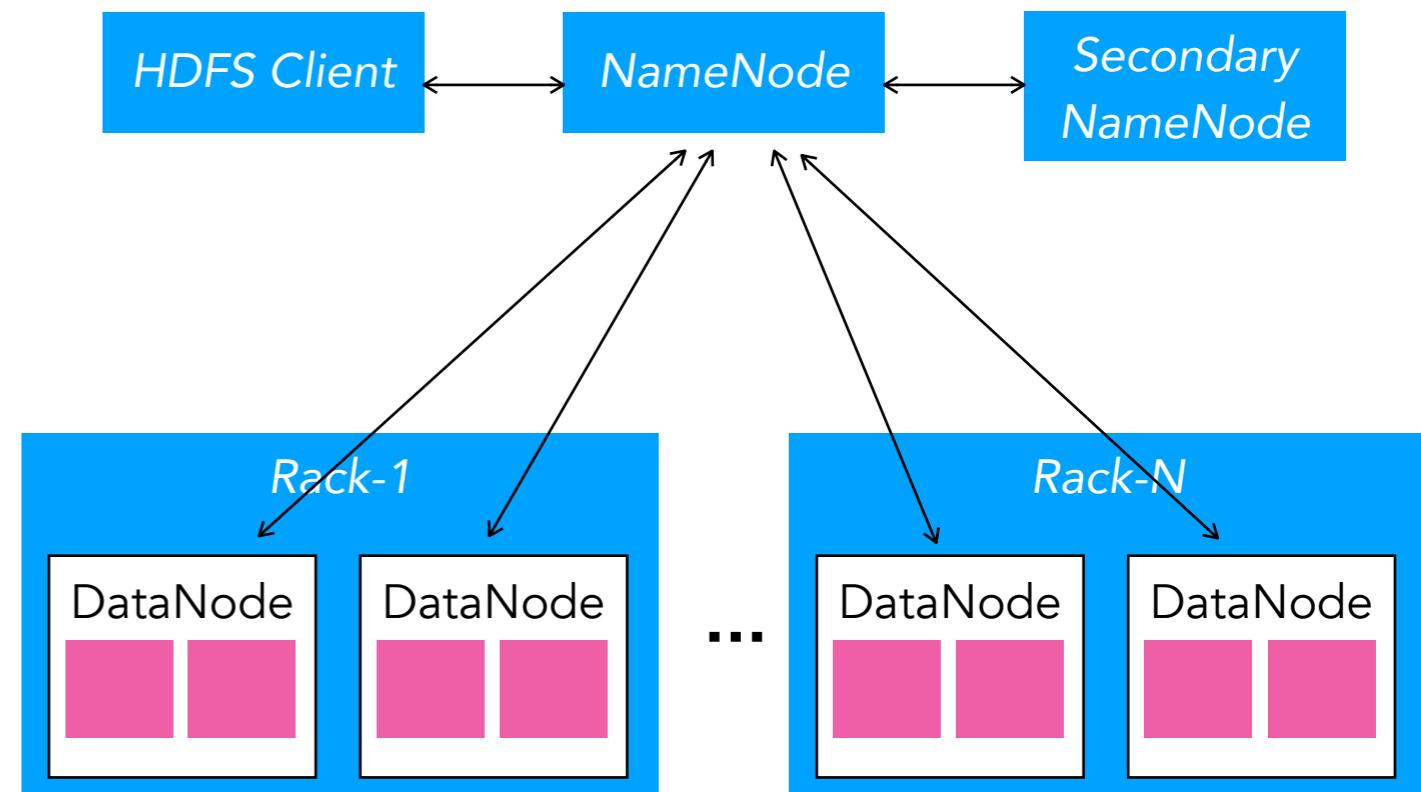
Jakob Blomer: Experiences on File Systems, 2015



# HDFS: Hadoop Distributed File System

- Puede ser ejecutado en grandes clusters.
- Diseñado para trabajar con *commodity hardware*.
- **NameNode:**
  - Administra el espacio de nombres.
  - Almacena los metadatos.
  - Administra el flujo de control (operaciones de lectura y escritura) pero no el de datos.
- **Secondary NameNode:**
  - Se le delega checkpointing si el Namenode no tiene recursos disponibles.
- **DataNode:**
  - Almacenan los bloques de datos.
  - Sirven las lecturas y escrituras.
  - Se reportan periódicamente con el Namenode.

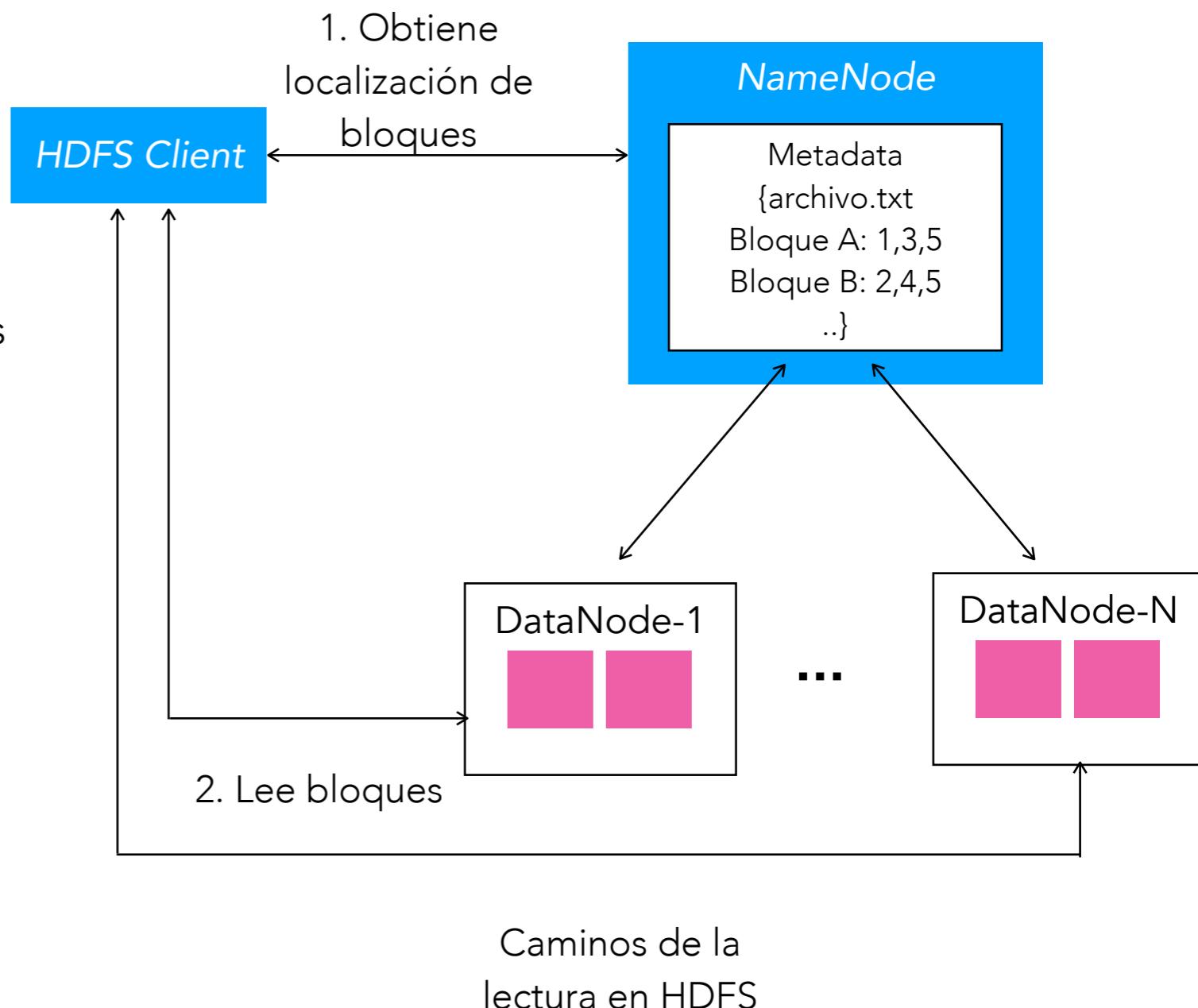
Arquitectura





# HDFS: Hadoop Distributed File System

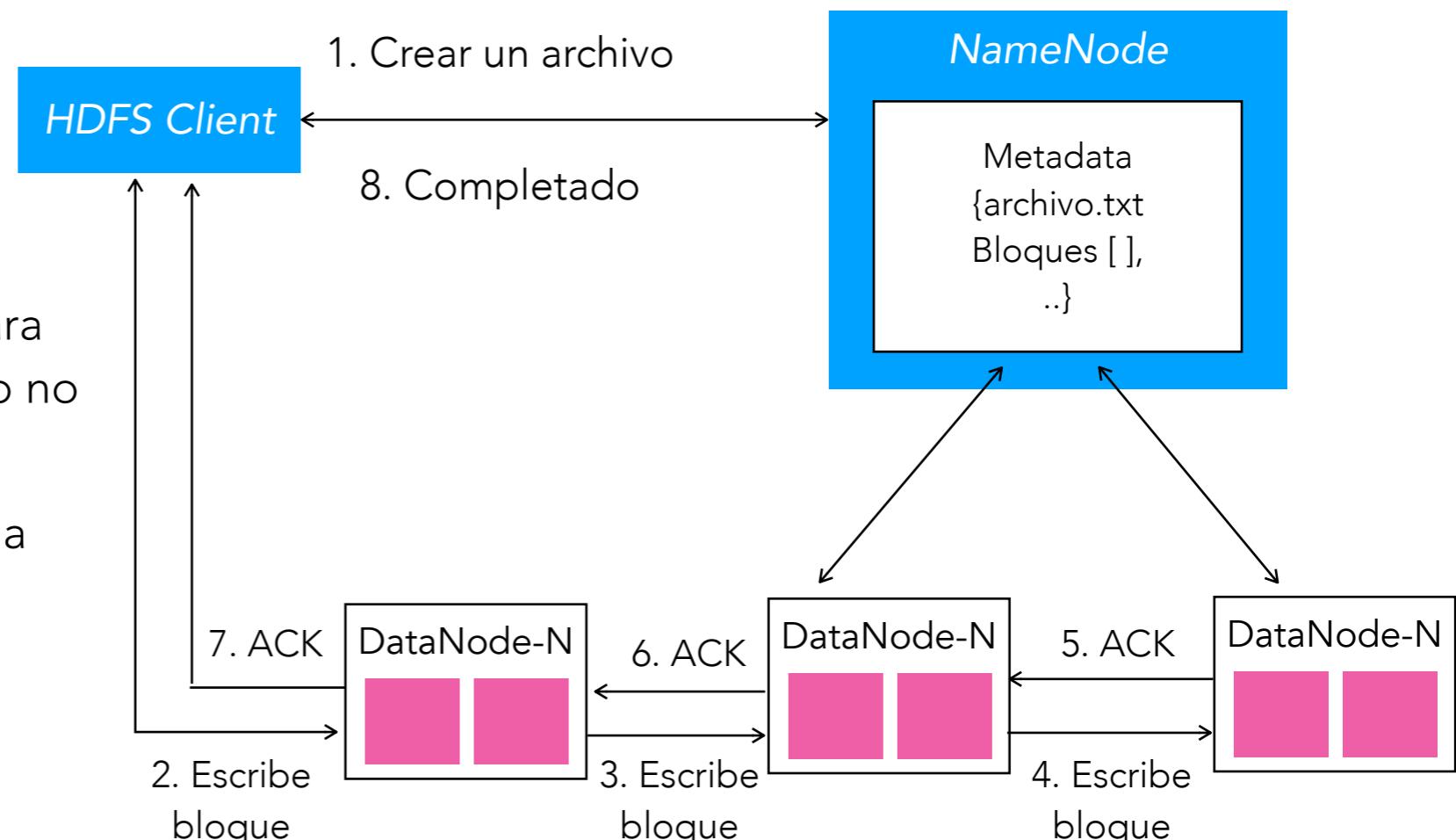
- Los bloques son replicados en los Datanodes (3 veces por defecto)
- La localización considera no localizar las réplicas en un mismo rack.
- Camino de la lectura:
  - La localización de los bloques son entregados en orden de cercanía al cliente.





# HDFS: Hadoop Distributed File System

- Camino de la escritura:
  - Namenode chequea que el cliente tenga los permisos para crear archivo, o que el archivo no exista en el sistema.
  - El Namenode responde con la localización de los objetos al cliente.
  - El proceso se hace para cada bloque.



## Replication Pipelining

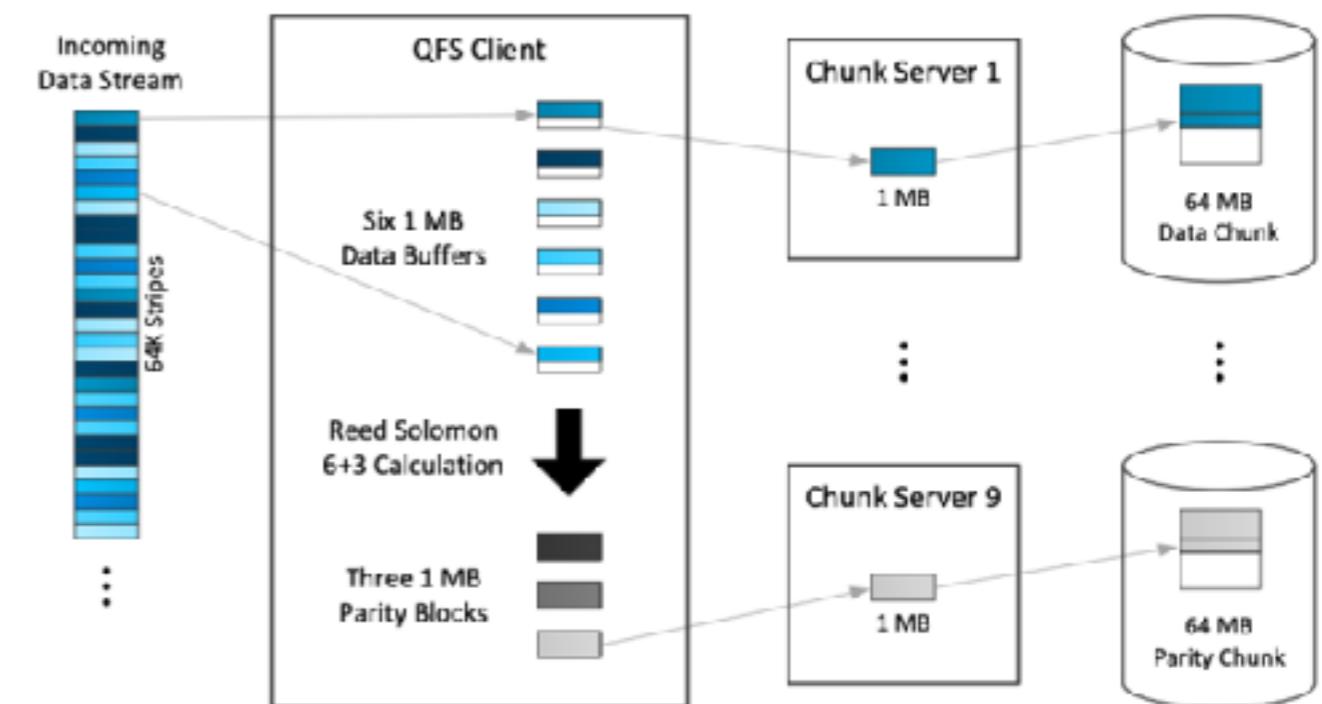
Caminos de la  
escritura en HDFS

Arshdeep Bahga and Vijay Madisetti, Big Data Science & Analytics, 2016.



- Alternativa a HDFS, escrito en C++ (manejo directo de memoria).
- Compatible con MapReduce.
- Características
  - Corrección de errores Reed-Solomon: HDFS expande x3 los datos. QFS usa la mitad del espacio del disco usando las técnicas de corrección de errores de RS.
  - 2x mejor rendimiento en escrituras (dado que escribe menos que HDFS).
  - Lecturas de disco en paralelo para aumentar rendimiento de las lecturas.

# QFS: Quantcast File System



Ovsianikov et al, The Quantcast File System, In Proceedings of the International Conference on Very Large Data Bases, Vol 6, N° 11.

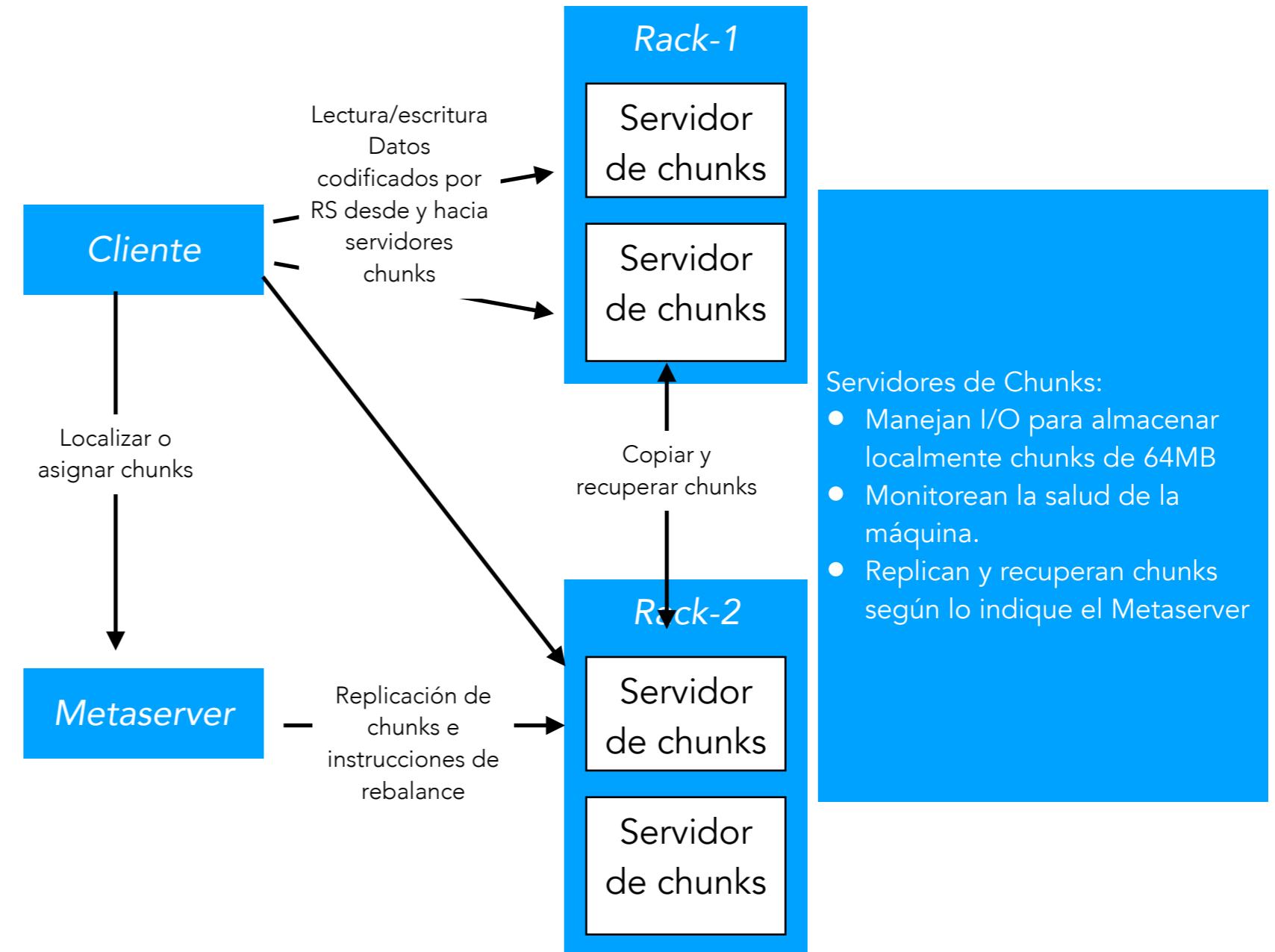


Cliente:

- Implementa interfaz de archivos de alto nivel.
- En escritura, RS codifica chunks y distribuye particiones a 9 servidores de chunks.
- En lectura, RS recolecta particiones de 6 chunks servers y recompose el chunk.
- Lee chunks de paridad según lo necesita para recrear datos faltantes,.

Metaserver:

- Mapa /archivo/ruta a IDs de chunks.
- Administra la localización de los chunks.
- Direcciona a los clientes a los servidores de chunks
- Administra la salud del sistema de archivos

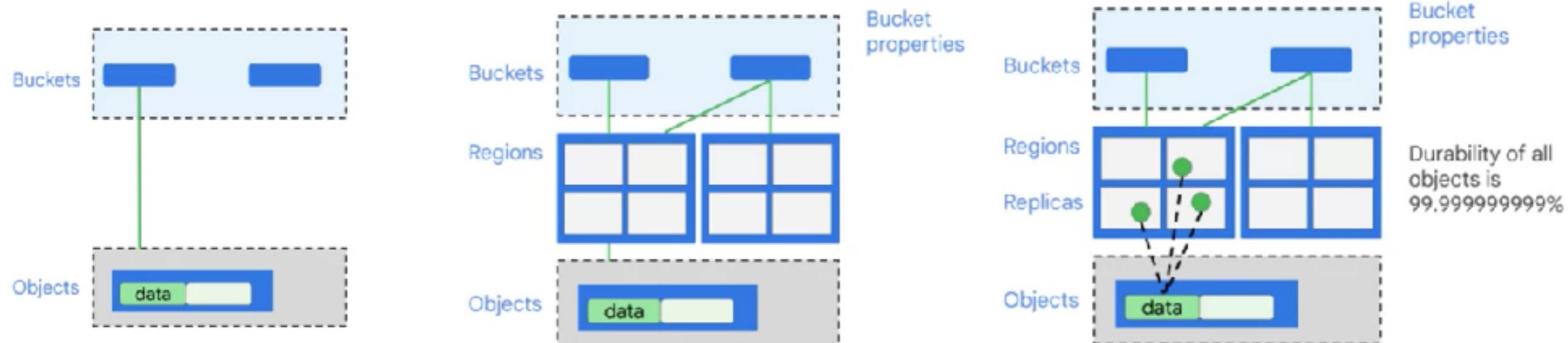


Ovsianikov et al, The Quantcast File System, In Proceedings of the International Conference on Very Large Data Bases, Vol 6, N° 11.



- Características:
  - Entrega 4 tipos de almacenamiento
  - Escalable hasta los exabytes
  - El tiempo de acceso al primer bytes es en milisegundos
  - Entrega alta disponibilidad
  - Posee una sola API

Los objetos  
almacenados son  
inmutables





# Cloud Storage

- Consola y gsutil dan la ilusión de tener un sistema de archivos jerárquico, a pesar de no existir.



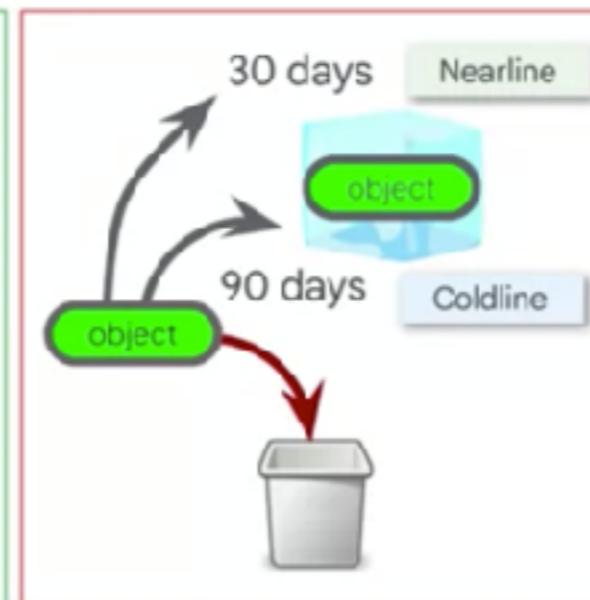
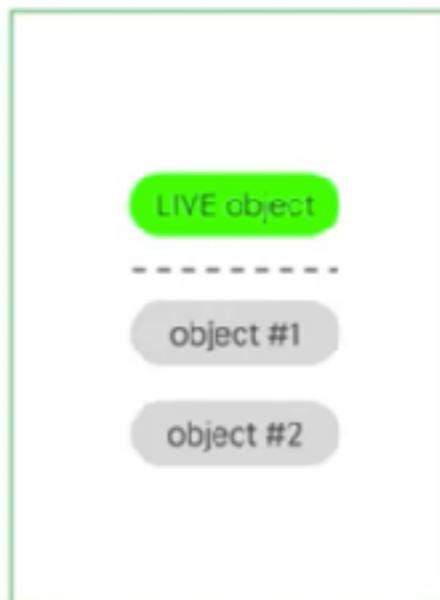


# Cloud Storage

	Standard	Nearline	Coldline	Archive
Caso de uso	Hot data	Datos accesados una vez al mes	Datos accesados una vez al año	Datos archivados, respaldos, recuperación desastres
Tiempo mínimo de almacenamiento	Ninguno	30 días	90 días	365 días
Precio almacenamiento ejemplo por GB por mes	0,020 CLP	0,010 CLP	0,004 CLP	0,0012 CLP
Precio retrieval ejemplo por GB	Ninguno	0,010 CLP	0,02 CLP	0,05 CLP
Disponibilidad	99,95% (multii/dual) 99,90% (region)	99,90% (multii/dual) 99,00% (region)		Ninguna
Durabilidad			99,99999999%	



# Cloud Storage



Más detalles en lab





# Cloud Storage

- Subir datos a Cloud Storage
- Consola y comando gsutil
  - -m para subir en paralelo
  - rsync para sincronizar carpetas

Más detalles en lab

Data Transfer

- Transfer Jobs
- Agent Pools
- Transfer Appliance | request
- Transfer Appliance | monitor

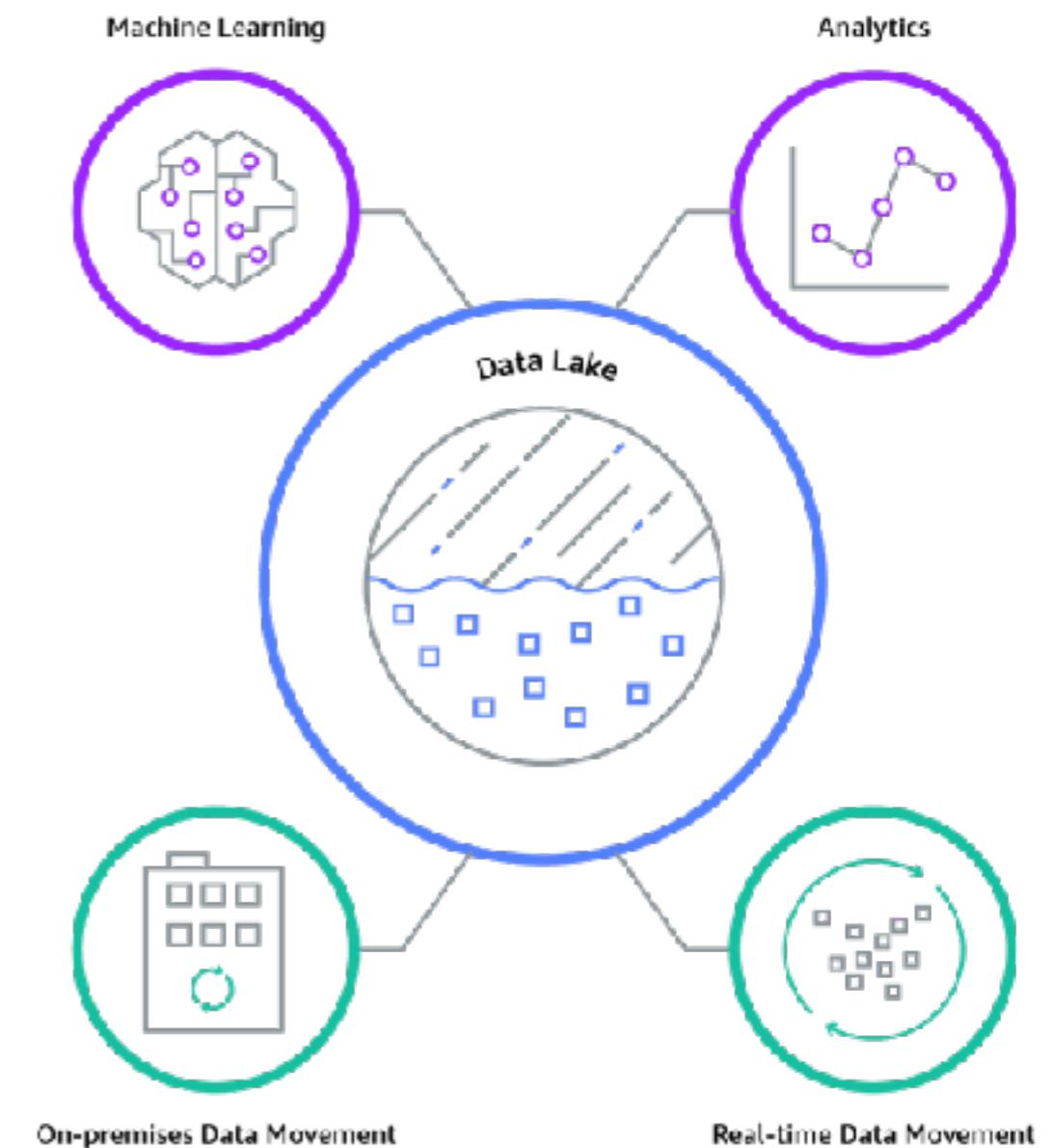
Source Type

- Amazon S3
- Azure Blob Storage or Data Lake Storage Gen2
- Google Cloud Storage
- POSIX filesystem
- URL list



- Un lago de datos es un repositorio centralizado que permite almacenar **datos estructurados y no estructurados** a cualquier escala.
- Objetivo es general valor de los datos.
- Evita silos para hacer consultas de analítica cruzada.
- Es diferente a un datawarehouse (base de datos optimizada para análisis de datos relacional)
- Generalmente usa almacenamiento de objetos como S3 en AWS y Cloud Storage en GCP.

# Data Lakes



<https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/>



# Componentes Data Lake

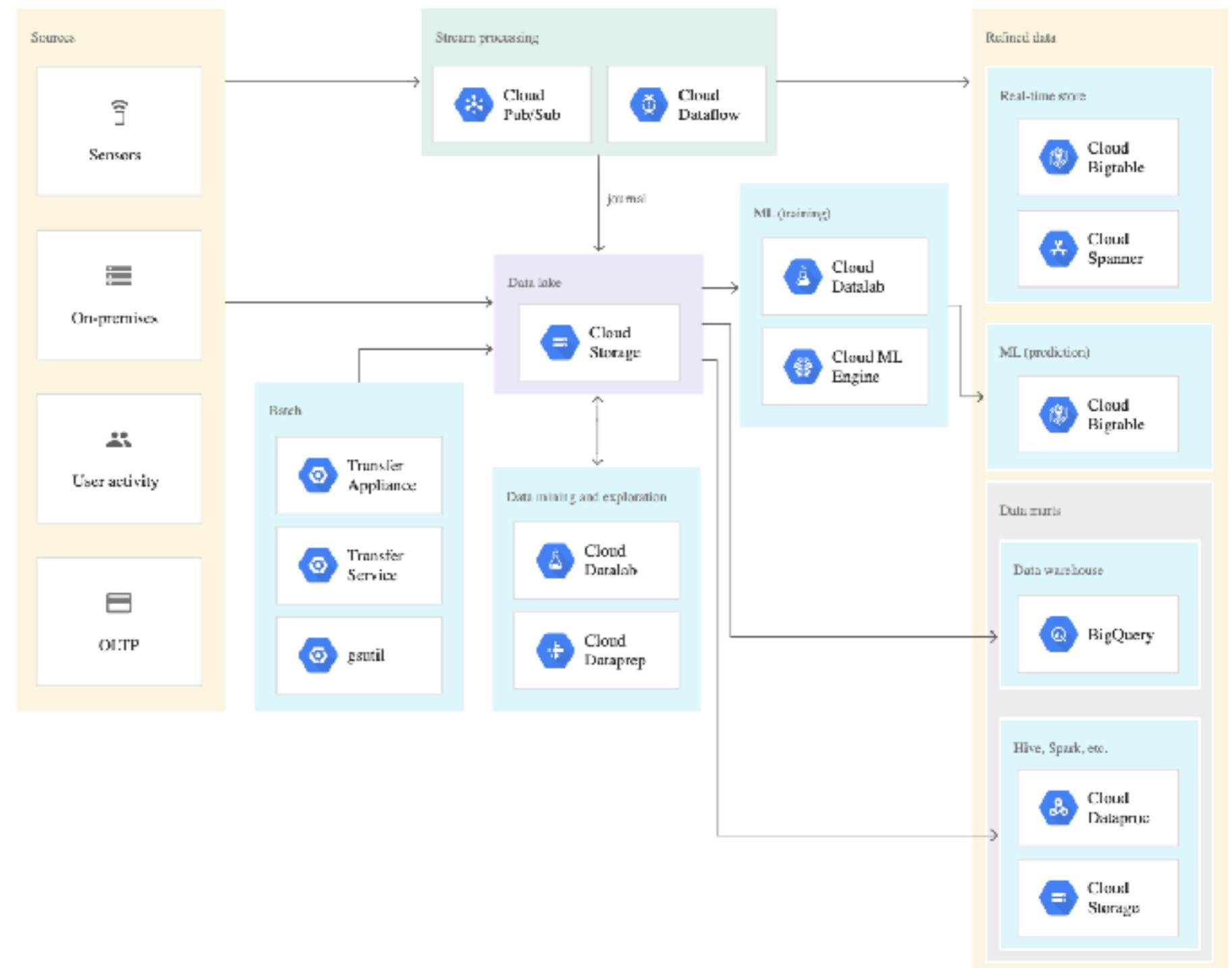
- Almacenamiento
  - De bajo costo (lifecycles?) y alta durabilidad
  - Soporta cualquier tipo de datos
- Catálogo y búsqueda:
  - Metadatos
  - Usados para estadísticas, clasificación y descubrimiento de datos.
- Sistemas de encriptación, autenticación, autorización, etc.
- APIs:
  - Exponer data lake a clientes
  - Catálogo de consultas



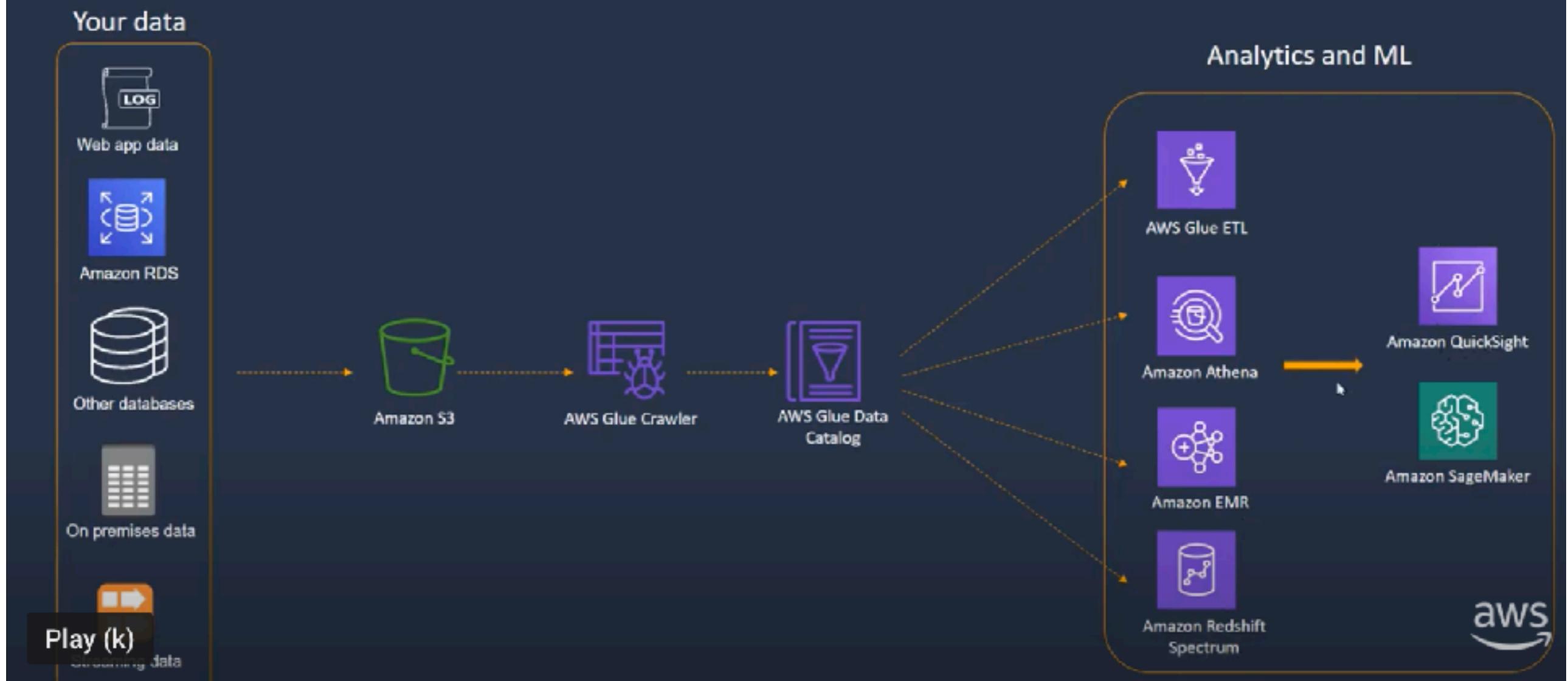
UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Arquitectura típica Data Lake en GCP



# Typical Data Lake Architecture





# Data Lake versus Data Warehouse

Characteristics		Data Warehouse	Data Lake
<b>Data</b>	Relational data	Relational data	Non-relational data
<b>Schema</b>	Defined schema or schema-on-read	OLD Data Warehouse No soportan el crecimiento de datos no estructurados y semi estructurados.	Written schema or schema-on-write
<b>Price/Performance</b>	Fast processing	Altos costos en almacenamiento de datasets grandes.	Query response times
<b>Data Quality</b>	High quality of data	No soporta bien Machine Learning y Data Science.	Any data quality
<b>Users</b>	Business users		Data scientists (using data)
<b>Analytics</b>	Batch processing		Machine learning and profiling

Nueva generación de DW: Apache Hive, Spark SQL, Cloudera Impala, Facebook Presto, Apache Tajo, Apache Drill. Algunos basados en Google's Dremel como BigQuery de GCP.



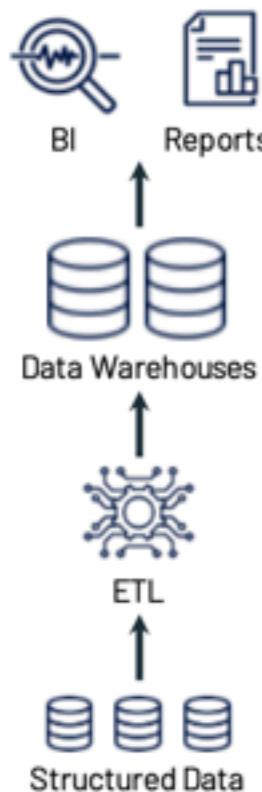
**Tendencia reciente:** Plataforma a la que abordaremos la próxima clase

Múltiples almacenamientos con diferente semántica.  
Muchos trabajos ETL que puede producir que los datos no sean consistentes ni "Frescos".  
Alto costo por trabajos duplicados y datos duplicados.

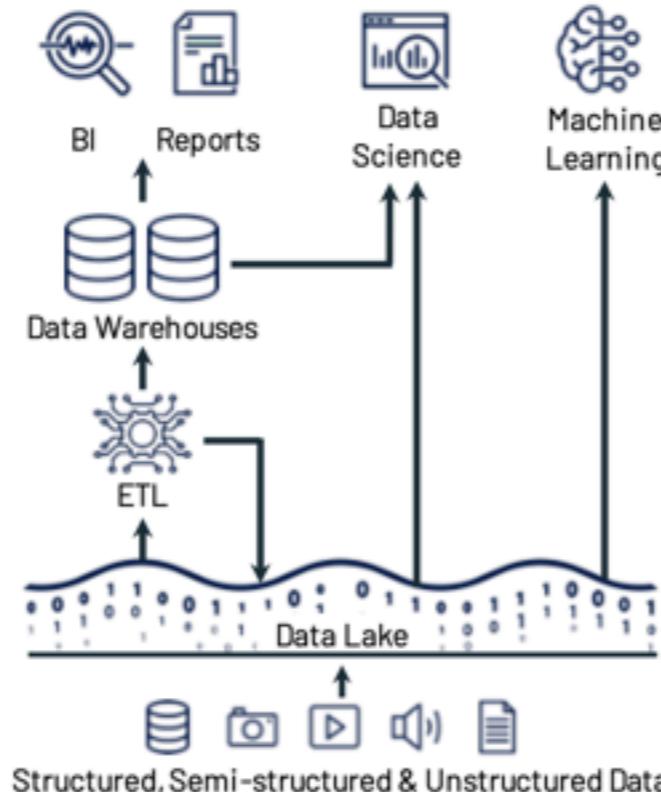
# Data Lake house

Este es un modelo que combina data warehouse, data lake y analítica avanzada. Lo

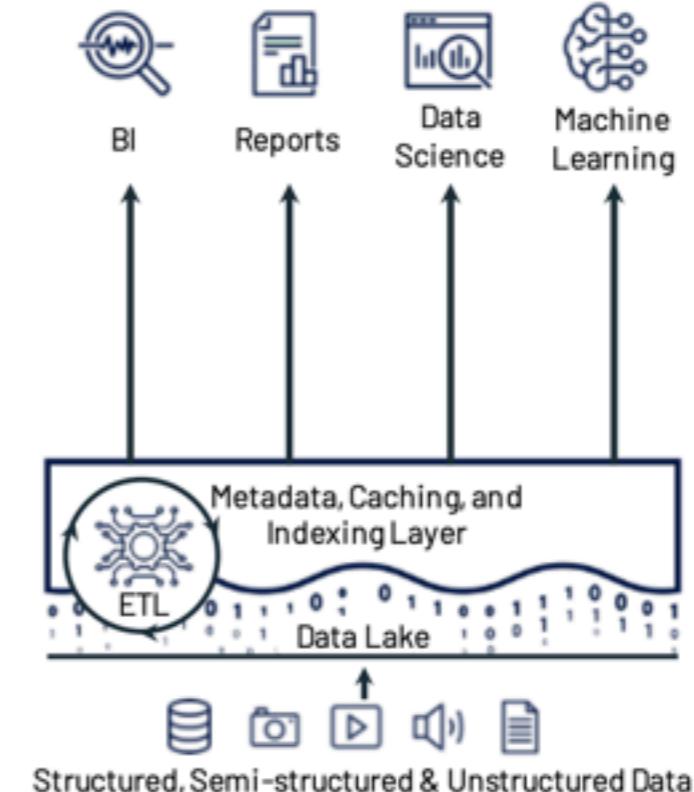
## Datawarehouse



## Data lake



## Lakehouse



[https://www.cidrdb.org/cidr2021/papers/cidr2021\\_paper17.pdf](https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf)



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Bases de datos para sistemas Big Data

PH.D. Erika Rosas Olivos

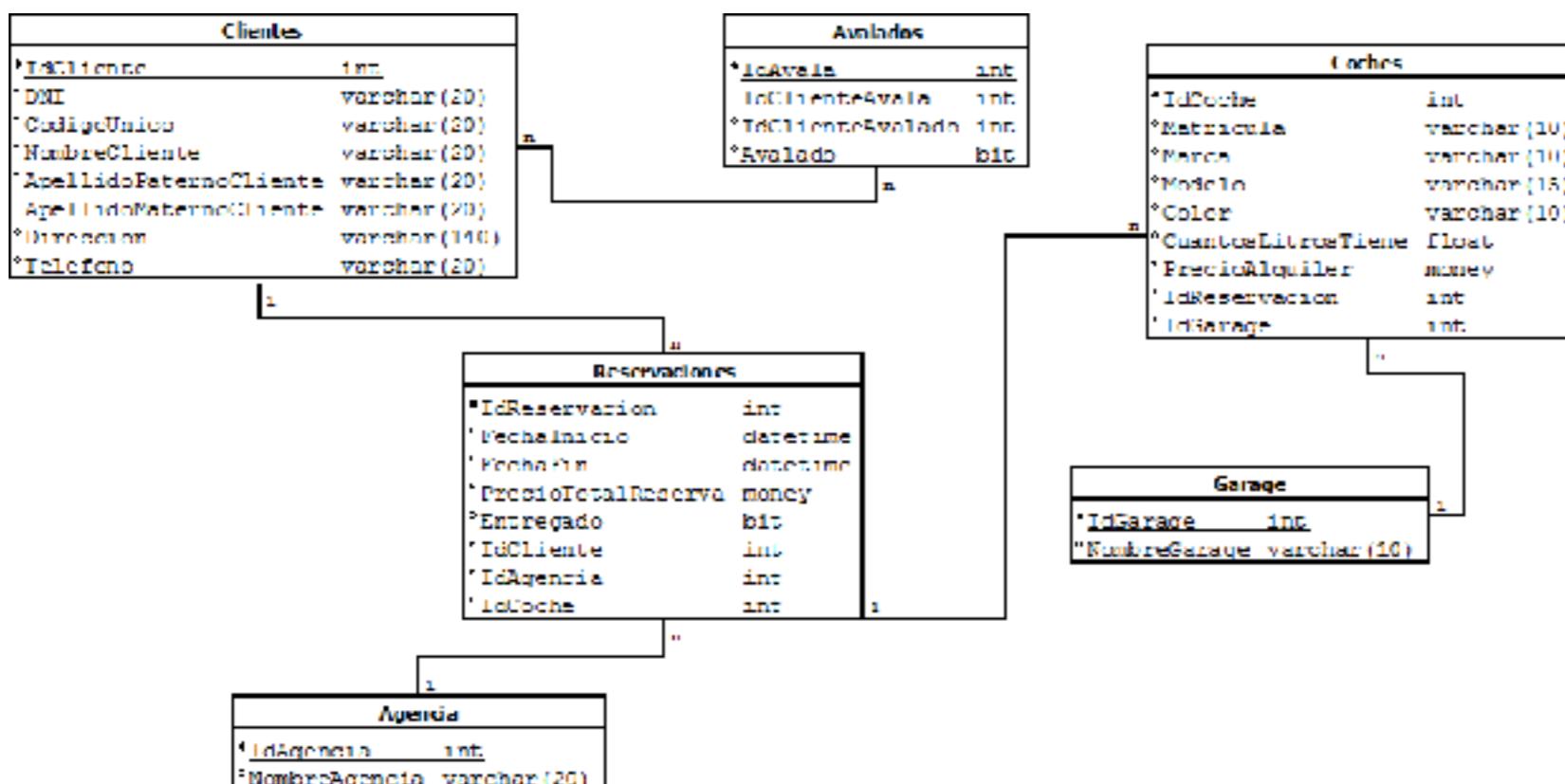
[erosas@inf.utfsm.cl](mailto:erosas@inf.utfsm.cl)

Septiembre 2023



# Algo de historia

- (1980) Invención de bases de datos relacionales para disminuir la presión en el almacenamiento de los sistemas.
    - Almacenamiento costoso.
    - Modelos normalizados son eficientes en el uso de recursos de almacenamiento.



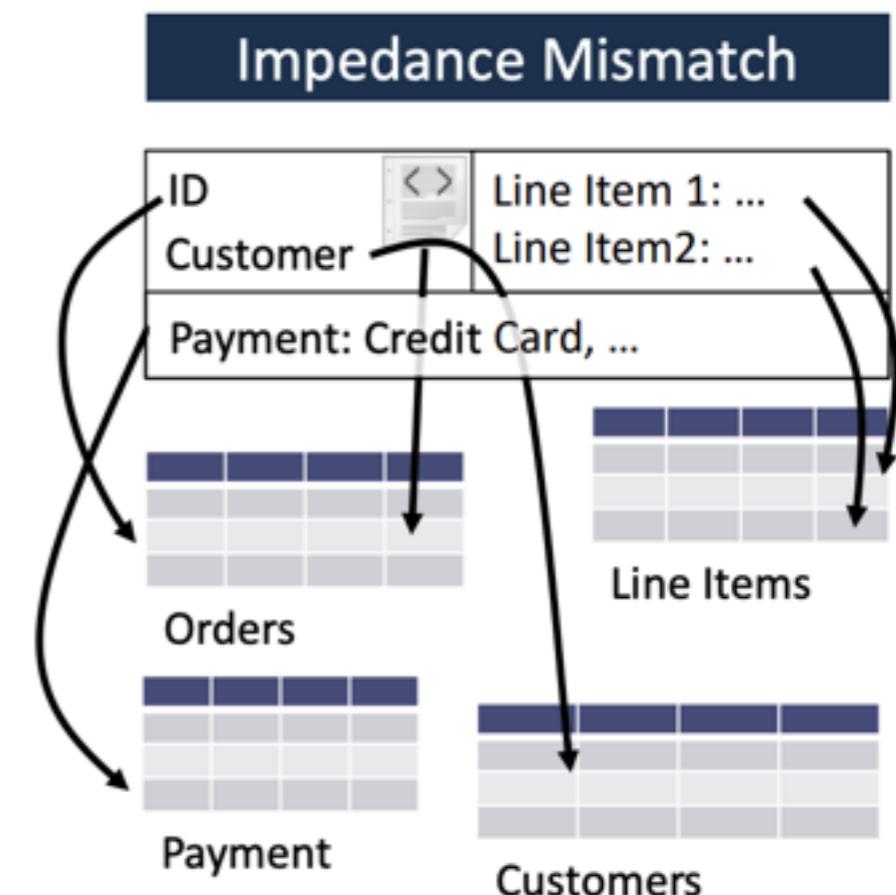
# Herramientas: RBDBMS SQL

## Ejemplo de un modelo relacional



# Bases de datos NoSQL

- (2010s) NoSQL interpretado como “Not Only SQL”
- Surgen por:
  - Necesidad de mayor escalabilidad, ej. datasets muy grandes o un gran throughput en escrituras.
  - Preferencia por software libre y de código abierto.
  - Operaciones de consultas especializadas no soportadas bien por el modelo relacional.
  - Deseo de un modelo de datos más expresivo y dinámico.



Ref. Designing Data-intensive applications, Martin Kleppmann, O'Reilly

Ref. Scalable Data Management, F. Gessler, W. Wingerath and N. Ritter @baqendcom



# Ejemplo Perfil de LinkedIn

<http://www.linkedin.com/in/williamhgates>

  
**Bill Gates**  
Greater Seattle Area | Philanthropy

#### Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

#### Experience

Co-chair - Bill & Melinda Gates Foundation  
2000 - Present

Co-founder, Chairman - Microsoft  
1975 - Present

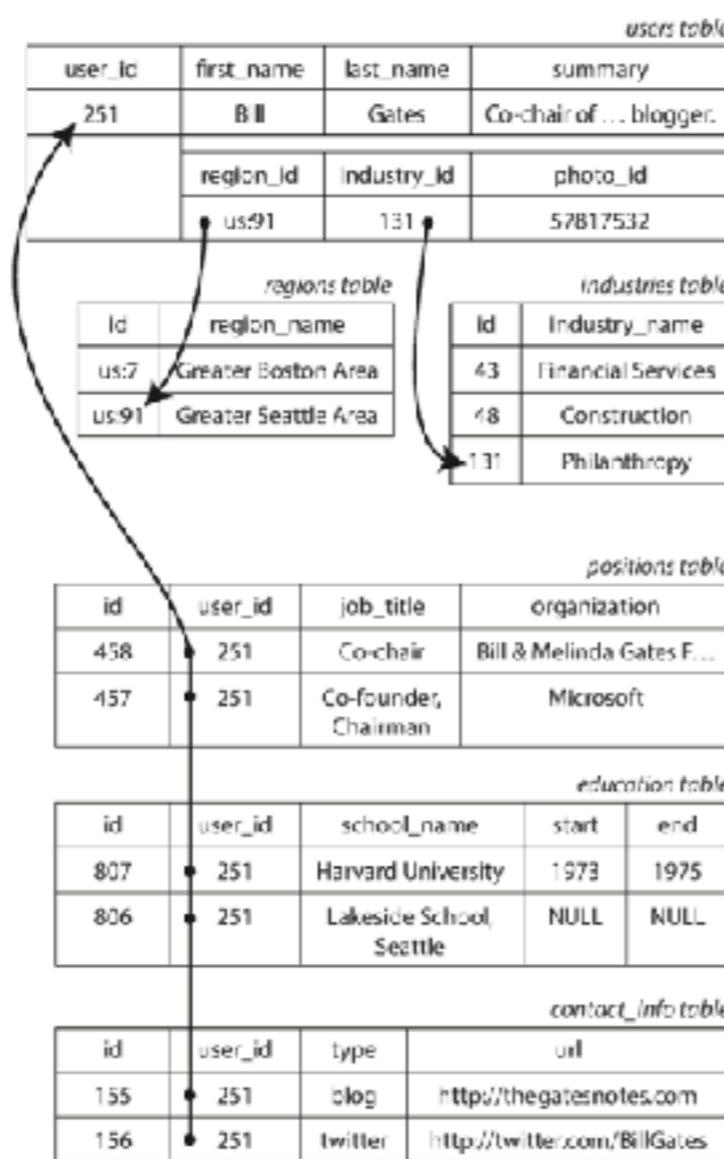
#### Education

Harvard University  
1973 - 1975

Lakeside School, Seattle

#### Contact Info

Blog: [thegatesnotes.com](http://thegatesnotes.com)  
Twitter: @BillGates



```
{
  "user_id": 251,
  "first_name": "Bill",
  "last_name": "Gates",
  "summary": "Co-chair of the Bill & Melinda Gates Foundation... Active blogger.",
  "region_id": "us:91",
  "industry_id": 131,
  "photo_url": "/p/7/669/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University", "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog": "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```



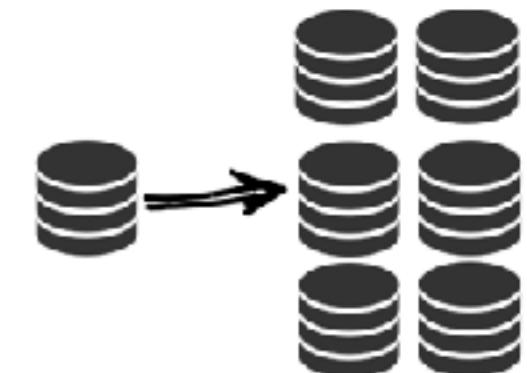
# Cambio de paradigma NoSQL

- Desde SW comercial a SW libre.
- Desde HW especializado a “Commodity” HW.
- Desde escalabilidad vertical a escalabilidad horizontal.
- Desde Red altamente disponible (Infiniband, Fabric Path) a red de tipo “commodity” (Ethernet)
- Desde Almacenamiento altamente disponible (RAID) a discos “Commodity” (HDD)
- Arquitectura Shared memory hacia Shared nothing

**Scale-Up (vertical scaling):**



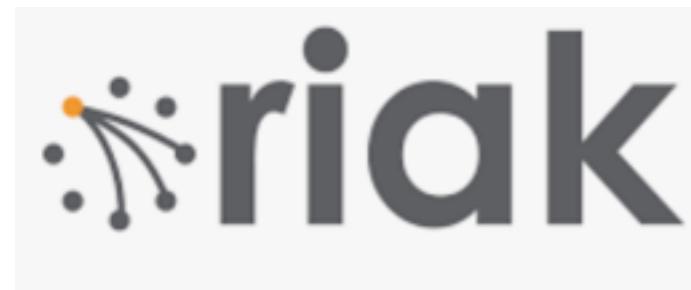
**Scale-Out (horizontal scaling):**





UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática



Muchas opciones ¿Cuál elegir?

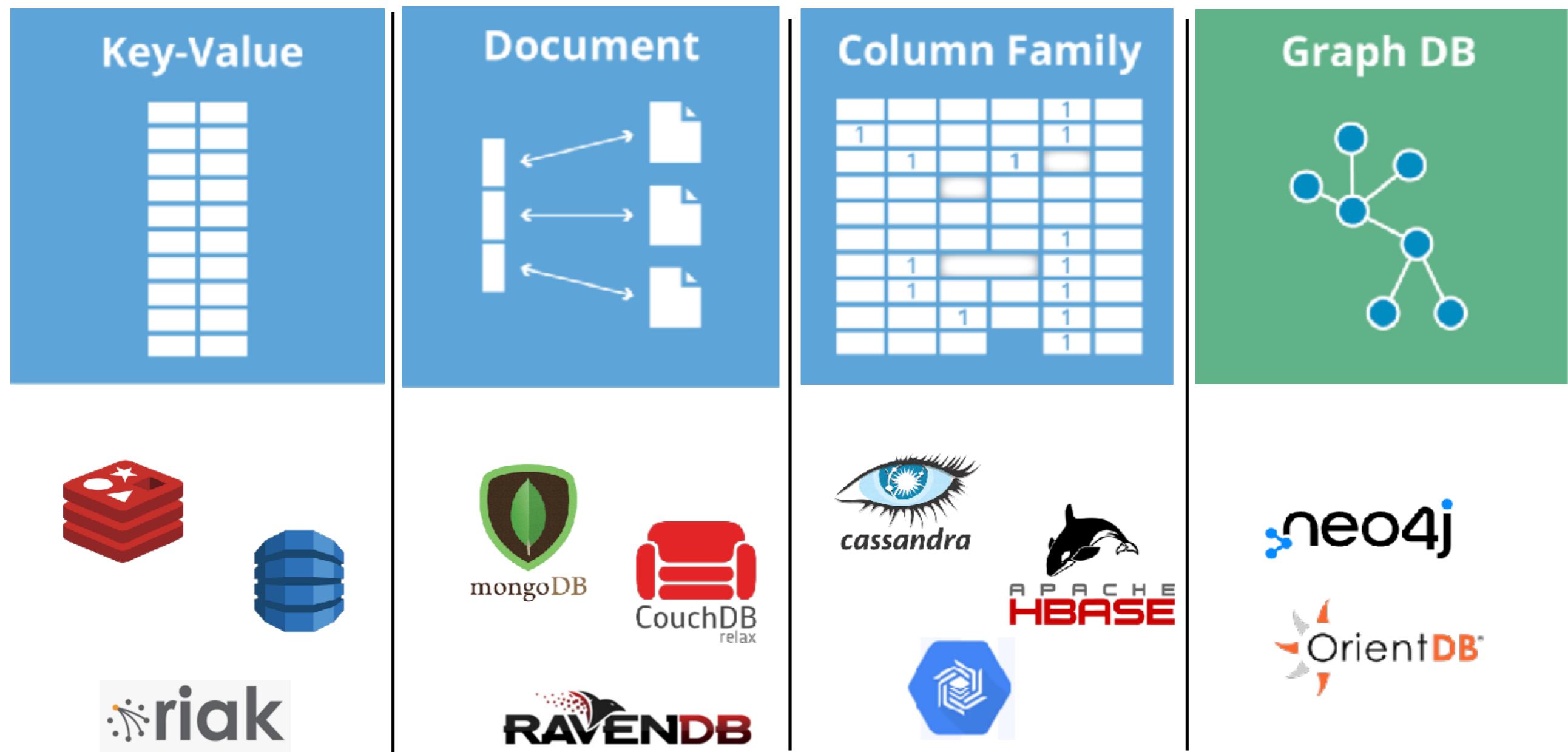
Pensar en el modelo de datos, integración con otros sistemas, escalabilidad, número de transacciones, soporte.



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Bases de datos NoSQL



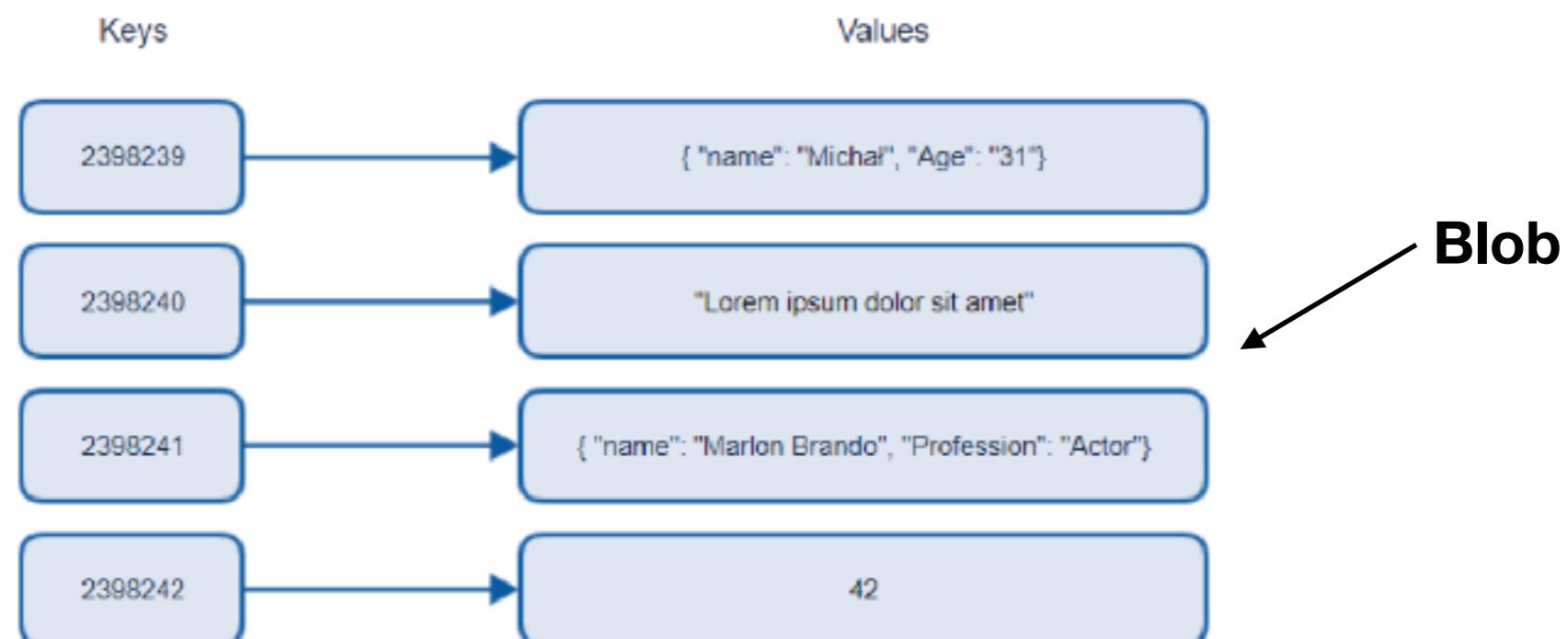
Logos: <https://neo4j.com/blog/aggregate-stores-tour/>



# Modelo Key-Value

- Pares clave valor identificados únicamente por la Key.
- Se consultan items por Key.
- Generalmente para aplicaciones con baja cantidad de lecturas pequeñas con modelos de datos simples
- Ejemplos: DynamoDB, Redis, Riak.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

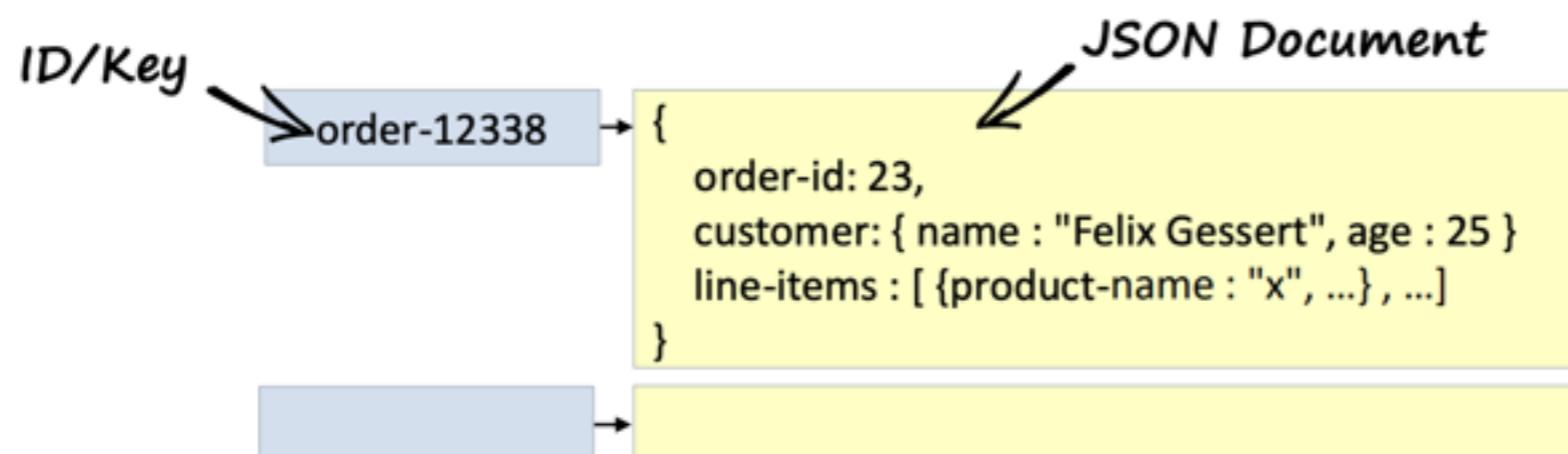


Ref. Big Data Analytics: A Hands-on Approach by Arshdeep Bahga Vijay Madisett  
Ref. Scalable Data Management, F. Gessler, W. Wingerath and N. Ritter @baqendcom



# Modelo Documentos

- Modelo de datos de Documentos únicamente identificados por ID de documento.
- Consulta de documentos por ID.
- Aplicaciones que tengan datos codificados como documentos en formatos JSON o XML, y que tengan un número variable de atributos.
- Ejemplos: MongoDB, CouchDB, RavenDB



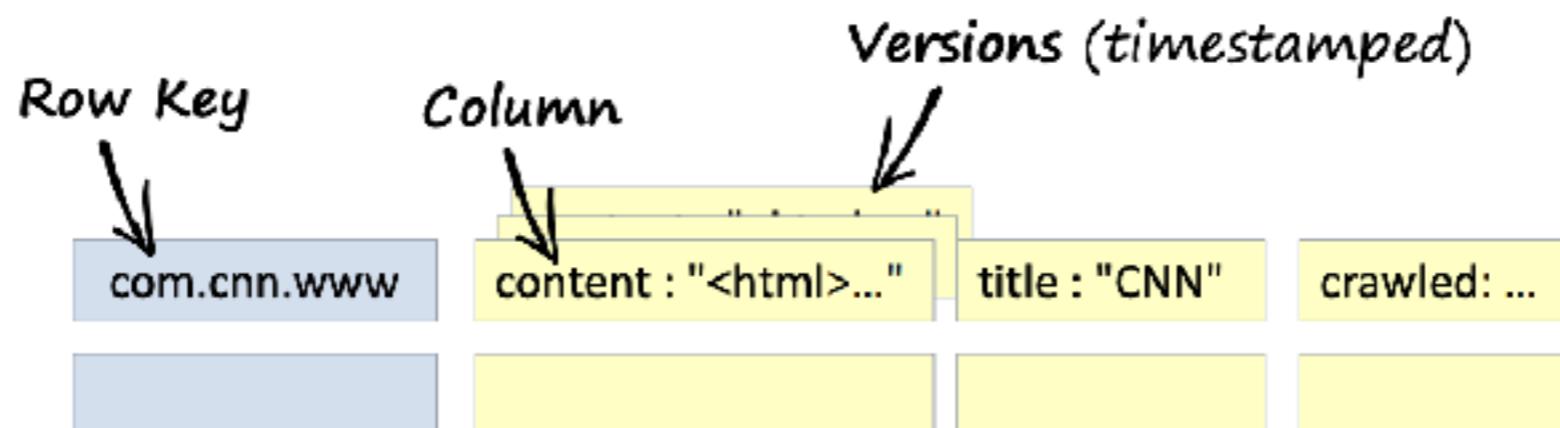
Modelos de datos  
desnormalizados permiten  
recuperar y manipular datos  
relacionadas con una sola  
operación en la base de datos.

Ref. Big Data Analytics: A Hands-on Approach by Arshdeep Bahga Vijay Madisett  
Ref. Scalable Data Management, F. Gessert, W. Wingerath and N. Ritter @baqendcom



# Modelo Wide-Column

- Modelo de datos que tiene columnas con nombres y valores agrupadas en familias.
- Se consulta por filas por clave.
- Aplicaciones que involucren grandes volúmenes de datos, con alto volumen e lecturas y escrituras, con requerimientos de alta disponibilidad.
- Ejemplo Google BigTable, HBase, Cassandra.

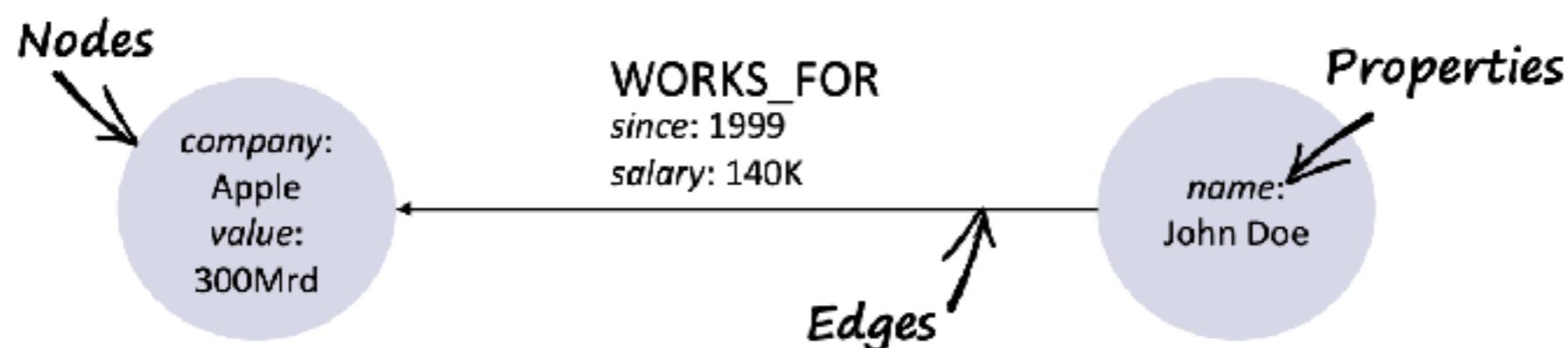


Ref. Big Data Analytics: A Hands-on Approach by Arshdeep Bahga Vijay Madisett  
Ref. Scalable Data Management, F. Gessert, W. Wingerath and N. Ritter @baqendcom



# Modelo Grafos

- Modelo de datos de grafos con nodos y sus relaciones.
- Consulta con lenguaje de grafos como Cypher.
- Aplicaciones que involucran datos en entidades y relaciones entre las entidades, datos espaciales.
- Ejemplo: Neo4j, OrientDB.





# Técnicas comunes en NoSQL BD

## Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach

Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,ker,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

### Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Fi-

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is in-

- SSTable
- Memtable - B-Trees
- Bloom filters
- WAL (Write Ahead Log)

- Consistency Hashing
- Sharding
- Replication
- Relojes de Vectores
- Merkle Tress
- Bloom filters

## Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,  
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall  
and Werner Vogels

Amazon.com

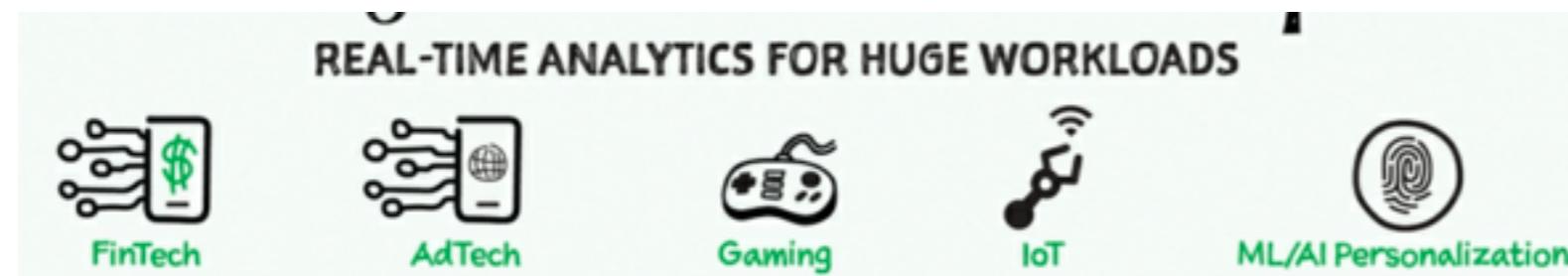
### ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and availability of the

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and



- Escala a petabytes
- Latencias de menos de 10 milisegundos.
- Motor de almacenamiento para aplicaciones de ML
- Fácil integración con herramientas open source de Big Data como Hadoop y Hbase.
- Es la base de datos que ocupa Google para sus servicios de Gmail, Maps, Analytics y Búsqueda.
- Casos de ejemplo:





# Bigtable: Wide-column DB

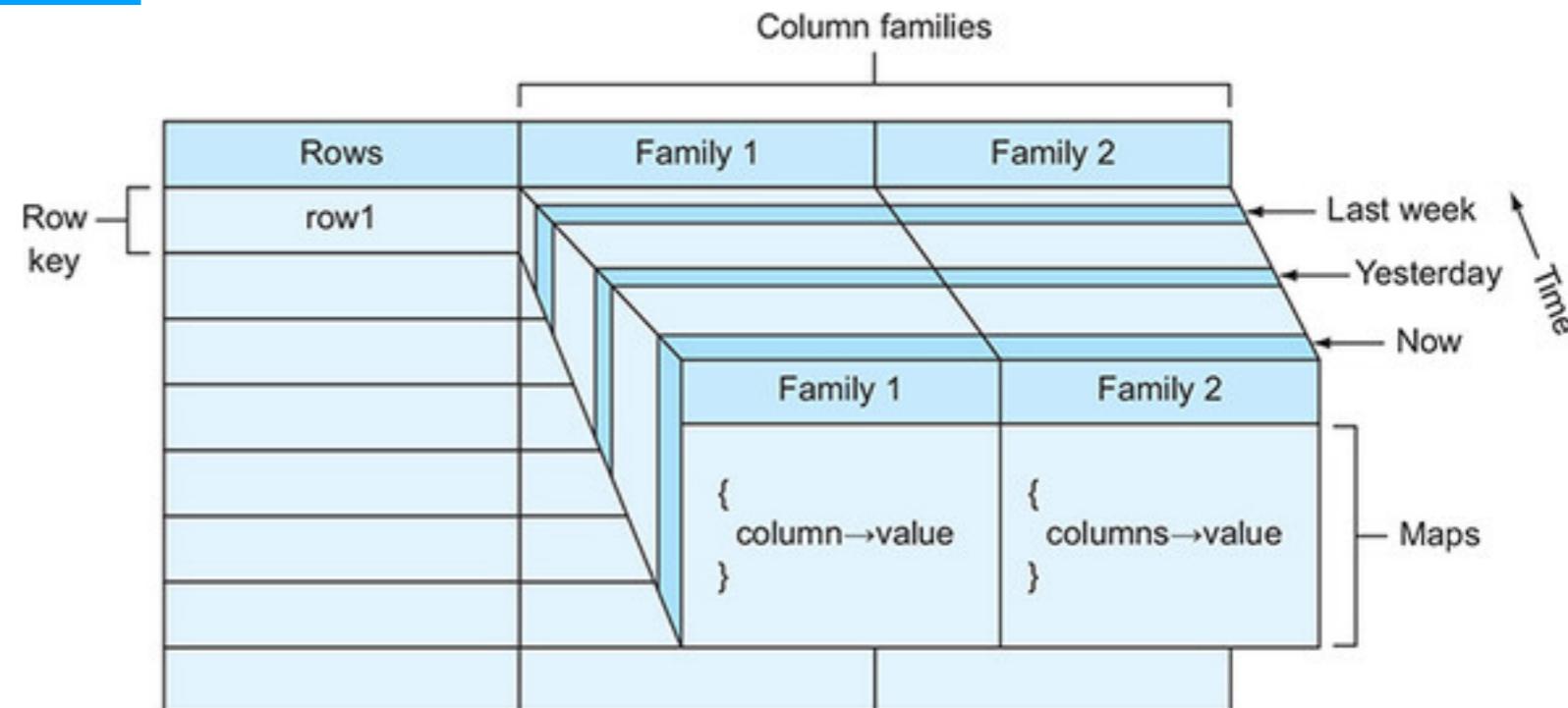
- Se requería un sistema de almacenamiento distribuido **tolerante a fallas** que provea acceso a datos que son **indexados** de una manera más sofisticada que GFS en cuanto a contenido y estructura.
- ¿Usar una base de datos relacional? Son reconocidos los problemas de rendimiento y escalabilidad en estos sistemas.
- BigTable no soporta un modelo relacional de datos.
  - No hay restricción que cubran el ancho de la tabla.
  - No hay transacciones multi fila.
- Bigtable tiene un modelo de **tabla** con una interfaz simplificada adecuada para el estilo de aplicación de Google: almacenamiento eficiente y recuperación de datasets masivos de datos estructurados.
- Bigtable soporta dinamismo en el formato de la tabla.

Una tabla es una estructura tridimensional que contiene celdas indexadas por clave de la fila, clave de columna y timestamp.

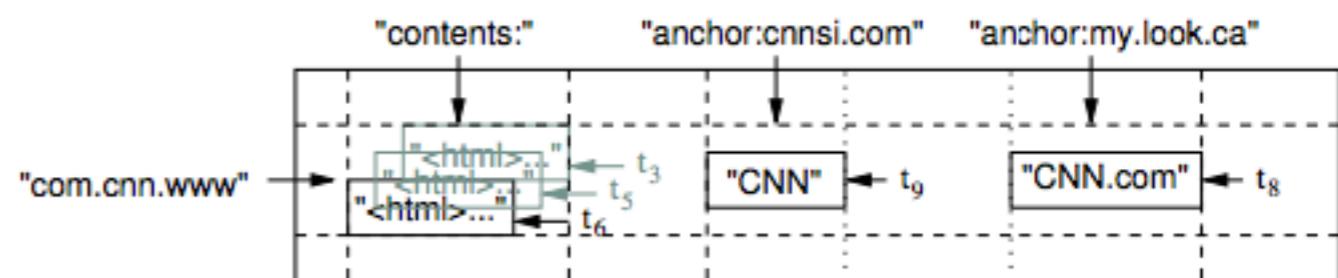
- Es la unidad de distribución y balance de carga.



## Bigtable interface



- **Filas:** Tiene una clave de fila (hasta 64kB).
  - Datos son mantenidos en **orden lexicográfico** por clave de fila.
  - Acceso a filas es atómico.
- **Columnas:** organizadas en **familias**, que es un agrupamiento lógico donde los datos de las columnas tienden a ser del mismo tipo.
  - Sintaxis. *family:qualifier*
  - La familia es la unidad básica del control de acceso.
- **Timestamps:** Cualquier celda puede tener múltiples versiones indexadas por timestamp.

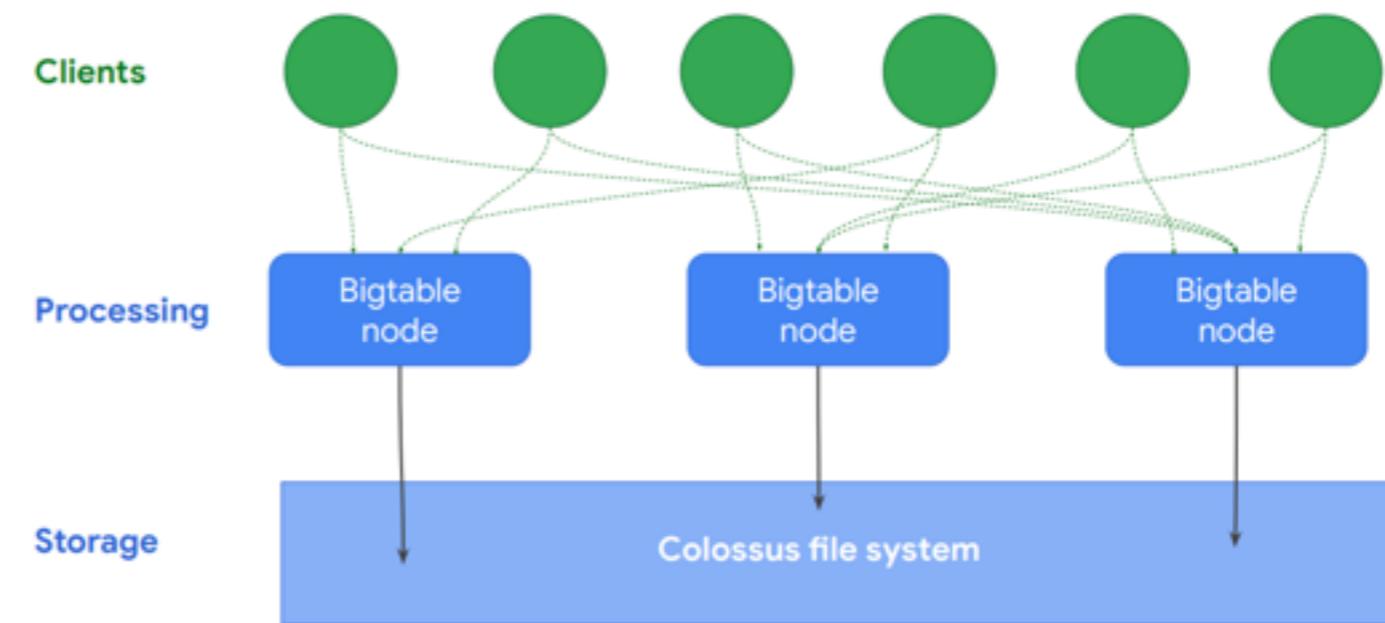




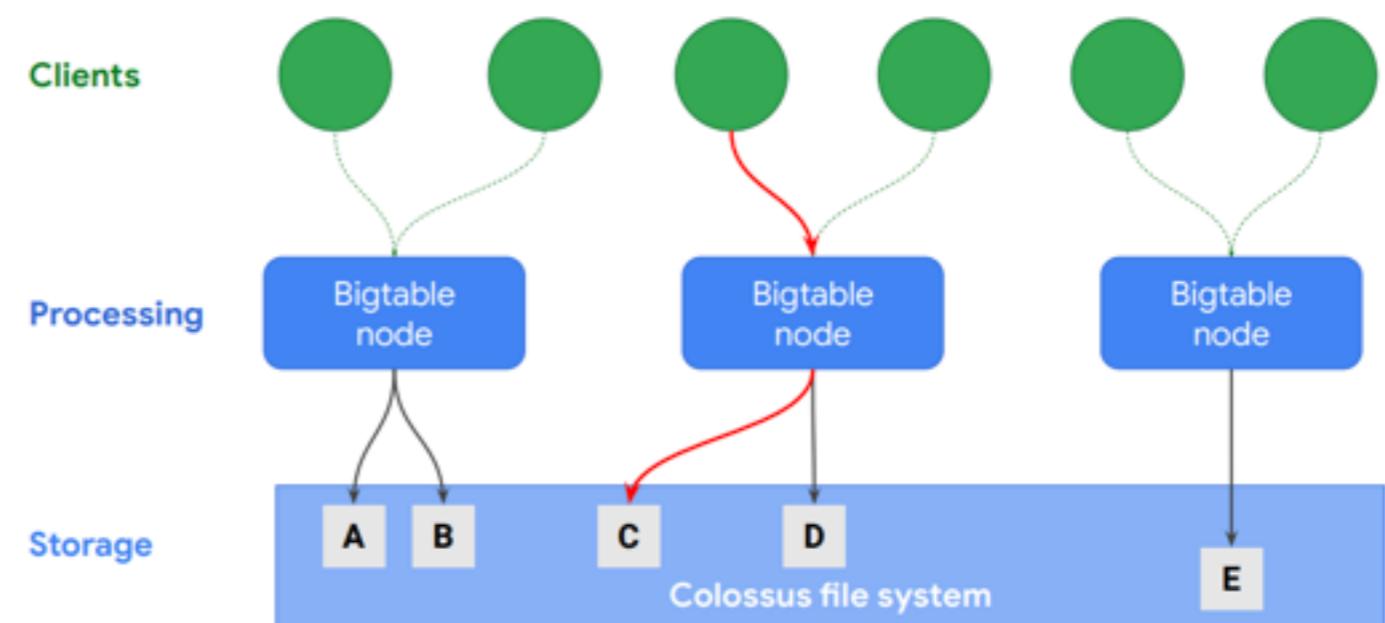
# Arquitectura de BigTable

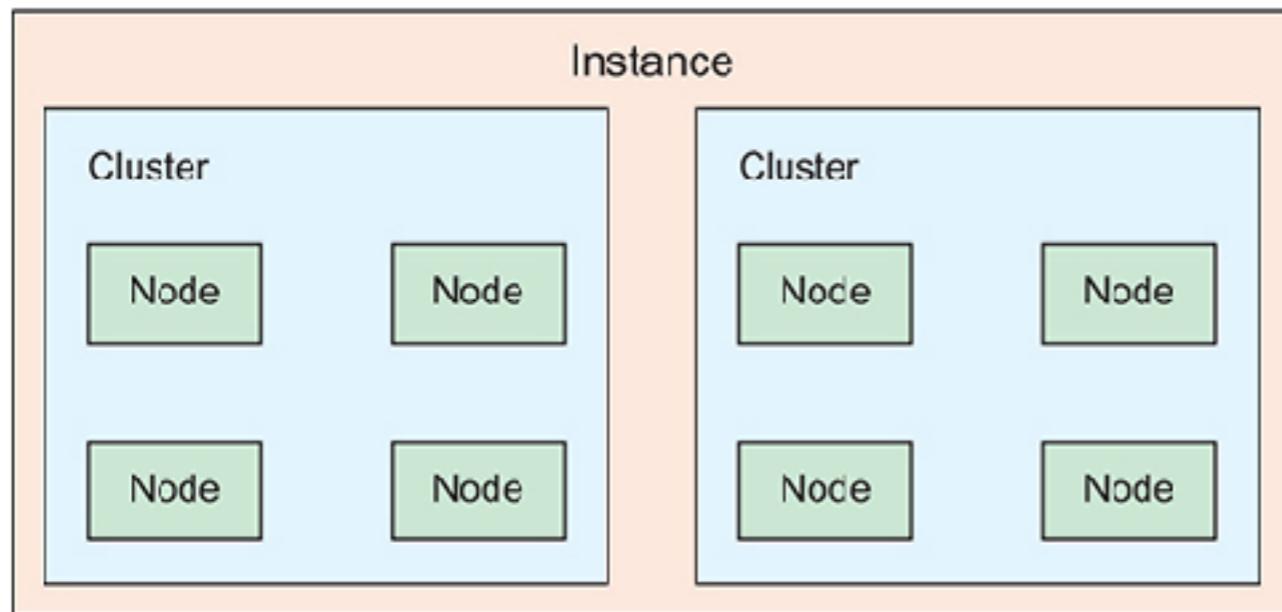
- Tablets se almacenan en Colossus en formato SSTable

Processing is separated from storage



Rebalances without moving data



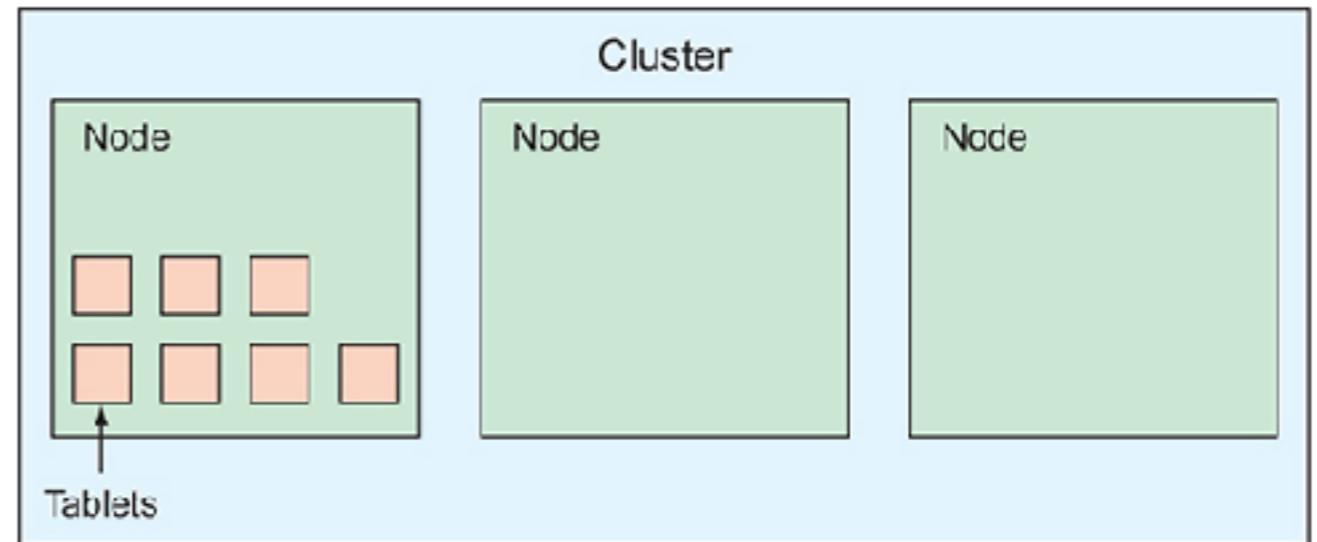


Una instancia en BigTable es un contenedor para datos.

Una instancia tiene uno o más clusters.

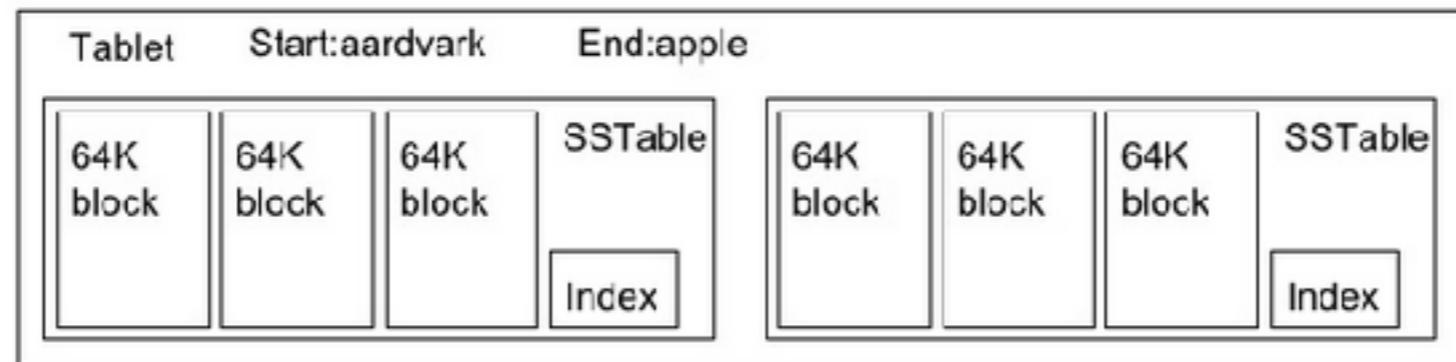
Un cluster tiene al menos un nodo.

Una tabla pertenece a una instancia, no a un cluster ni a un nodo.





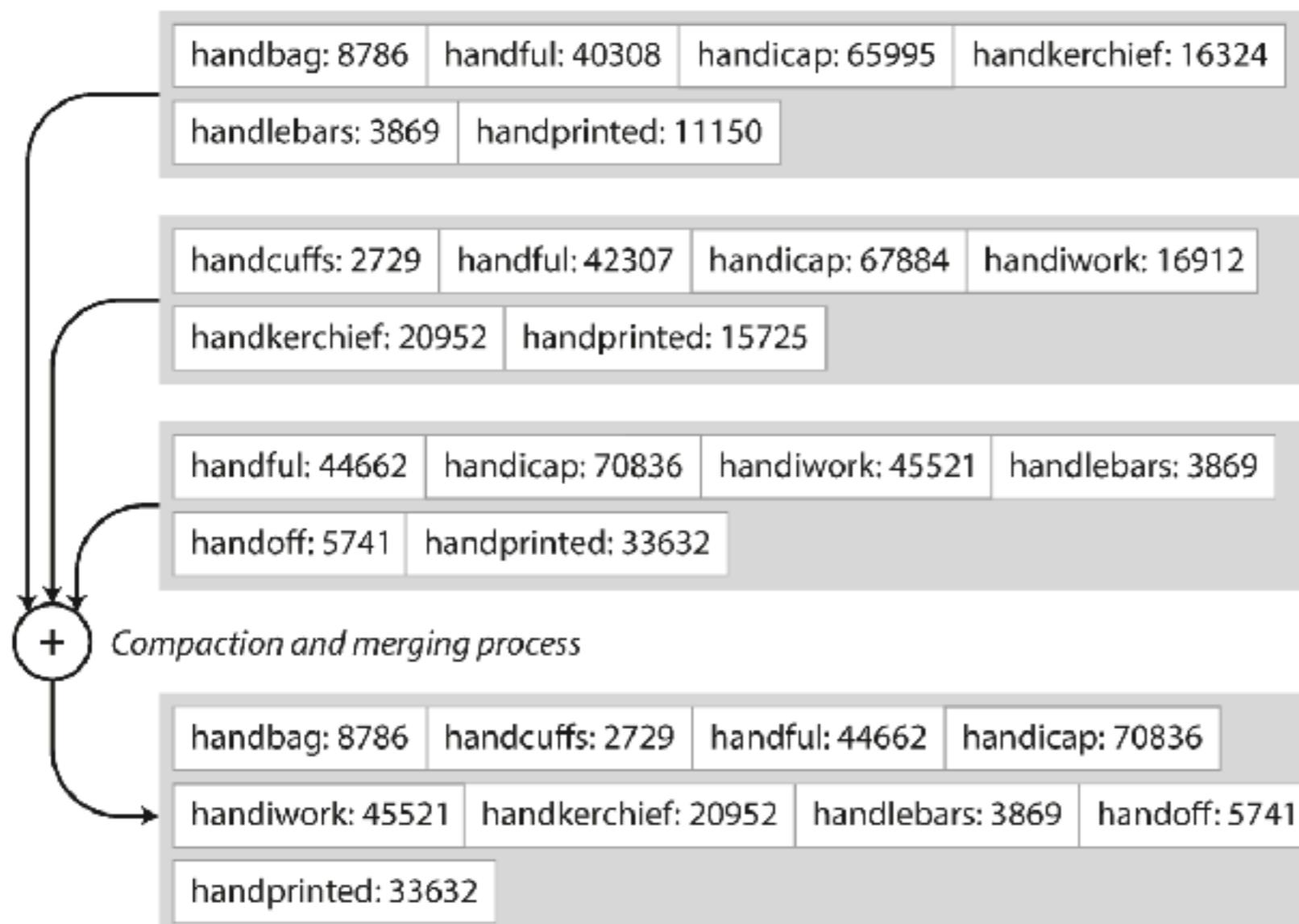
# SSTable



Mapping de tablas de Bigtable a GFS:

- Una tabla se divide en múltiples tablets dividiendo por fila, tomando el rango de hasta 100-200MB y mapeándolo en una tablet.
- Cada tablet se representa por una estructura de almacenamiento que consiste en un conjunto de archivos que almacenan datos: **SSTable** (Sorted Strings Table).
- El mapeo de las tablets a SSTables se hace con un índice jerárquico inspirado en los B+-trees.

**SSTable:** formato de archivo; unidad principal de almacenamiento; organizado como un map ordenado e inmutable de clave a valor.



# SSTable: Merge

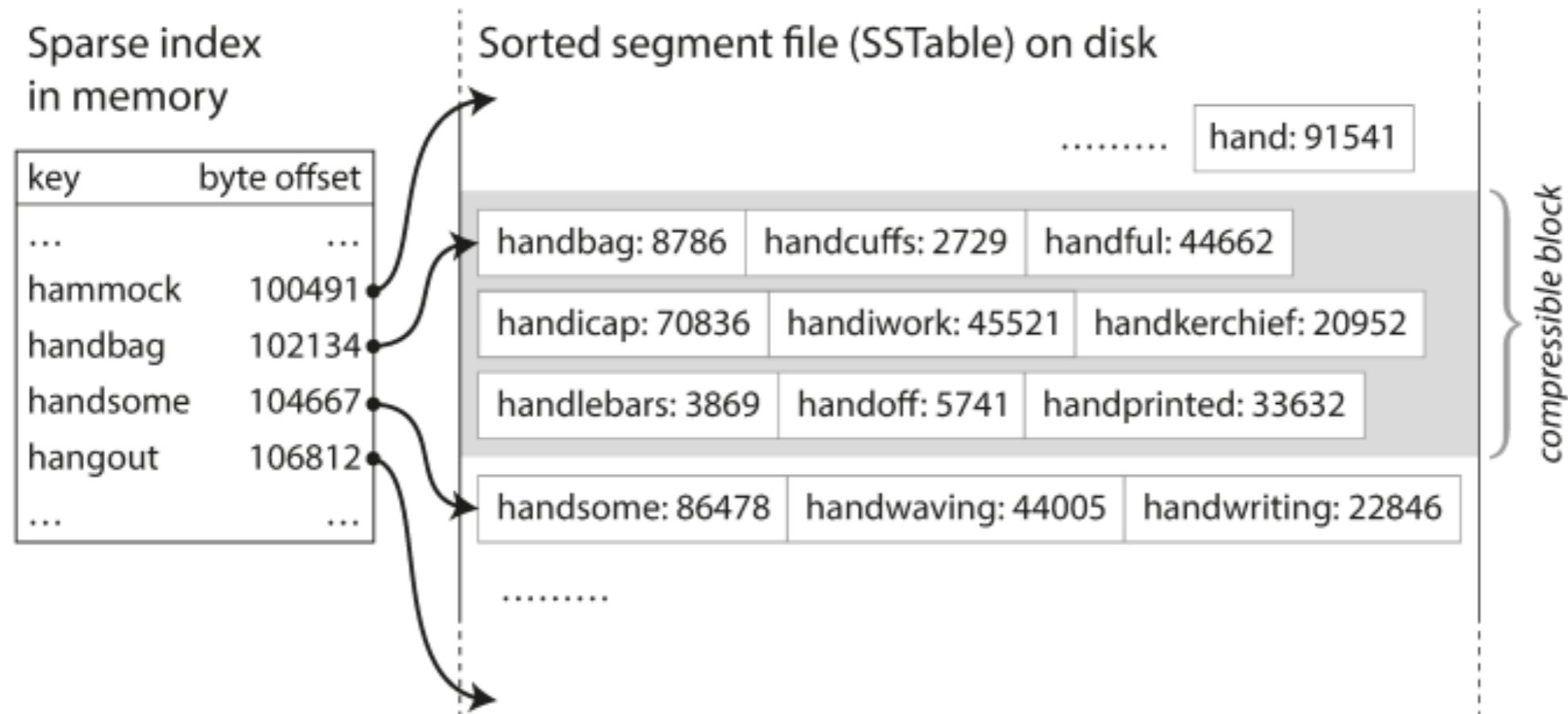
\* La secuencia de pares clave, valor está ordenada por key.

\* Cada key solo aparece una vez en el segmento.

\*Fácil de compactar, entregando una nueva secuencia ordenada, con una key una sola vez.

# SSTable: Índice en memoria

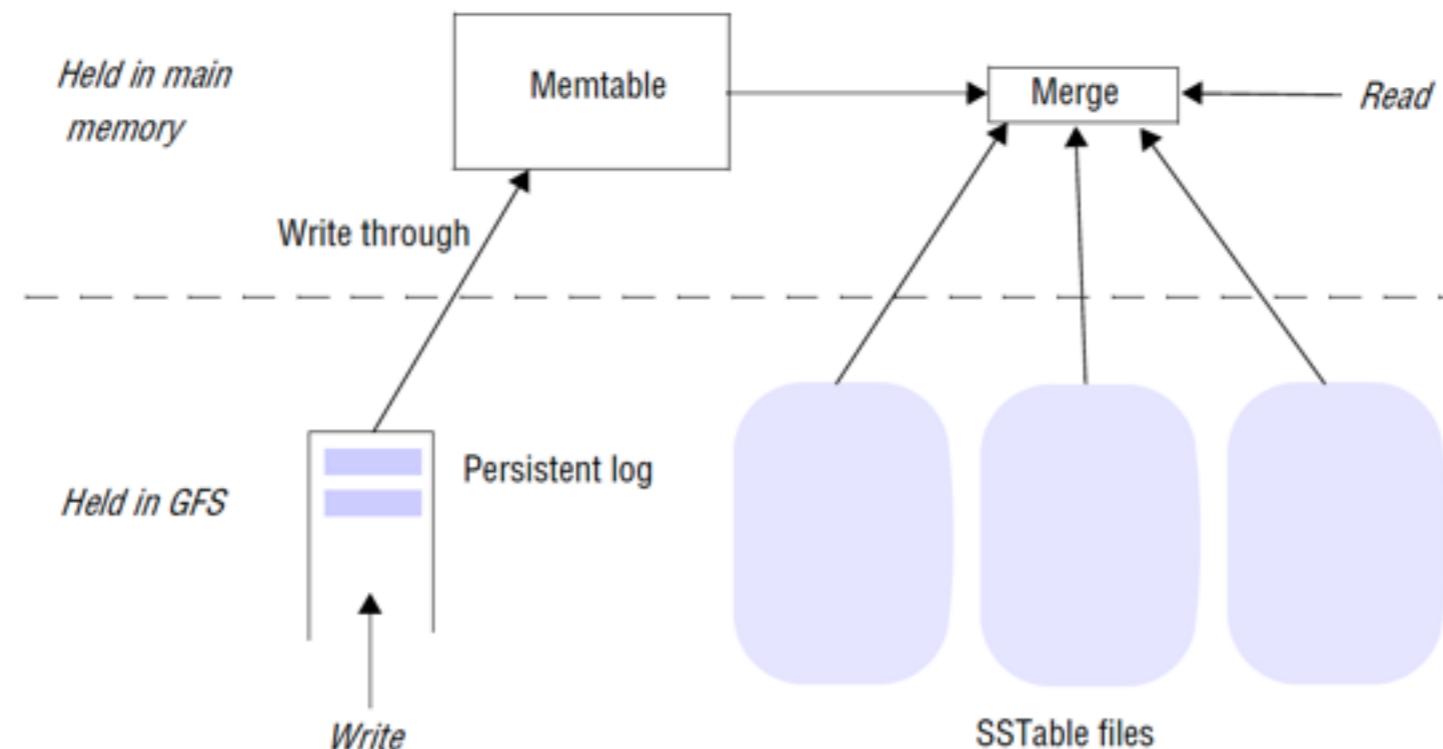
- \* Para encontrar una key no se necesita mantener un índice de todos las keys en memoria. Se utiliza un índice en memoria para indicar los offsets de solo algunas keys. Unos pocos por cada KB.



Designing data intensive applications



# Arquitectura de almacenamiento en Bigtable



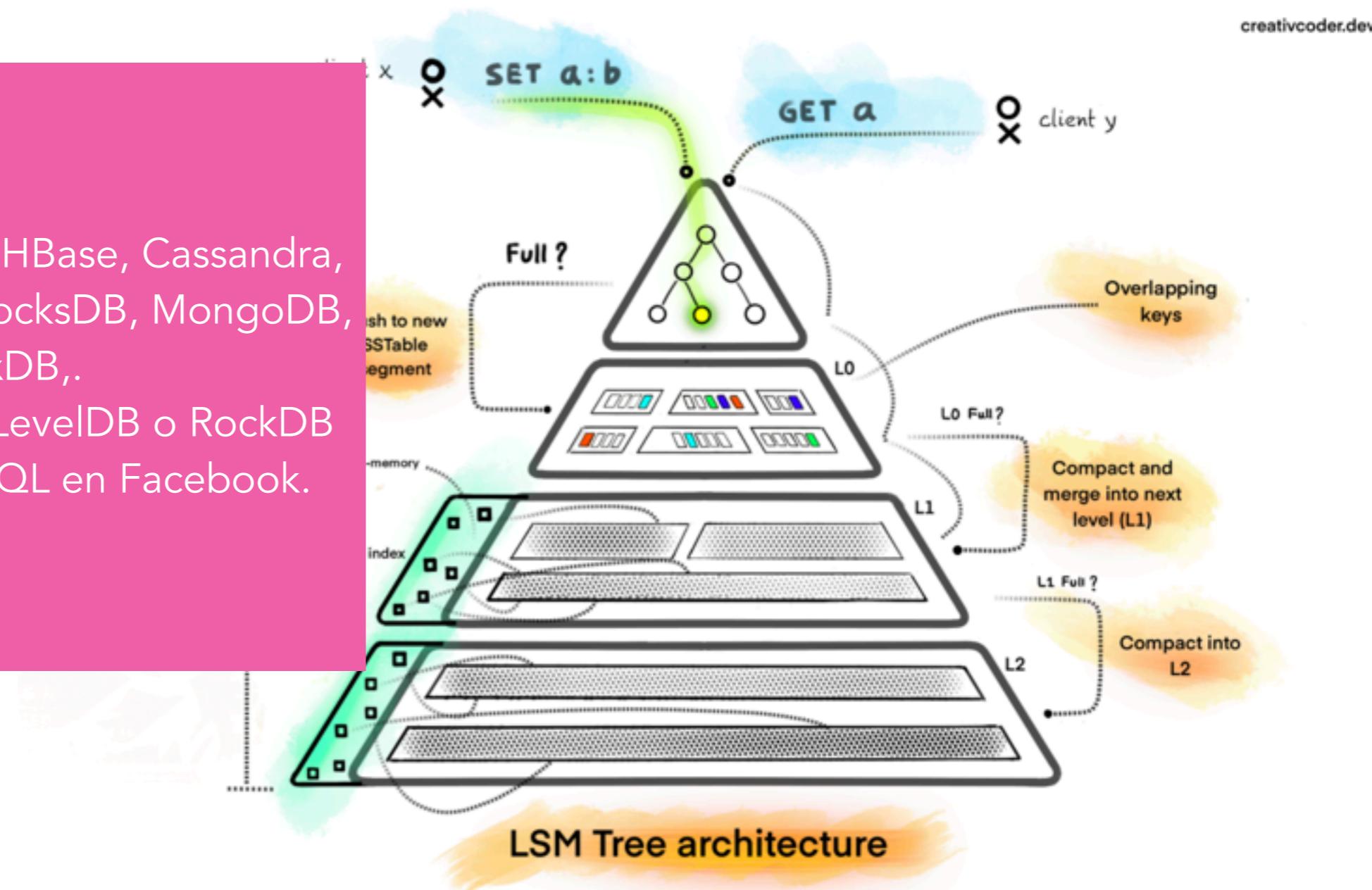
- Las **escrituras** hacen commit primero a un log para soportar recuperación ante fallas.
- Las entradas del log son escritas a través de la **memtable** que se almacena en memoria principal.
- SSTables actúan como un snapshot del estado de las tablet. La recuperación es implementada usando haciendo uso de los logs desde el último snapshot.
- Las **lecturas** son servidas al proveer una vista agregada de los datos de la SSTable combinada con la Memtable.



# Arquitectura LSM-tree

- \* ● Implementado por HBase, Cassandra, PNUTS, LevelDB, RocksDB, MongoDB, InfluxDB,.
- \* ● Y BD que ocupan LevelDB o RockDB como Riak o MySQL en Facebook.

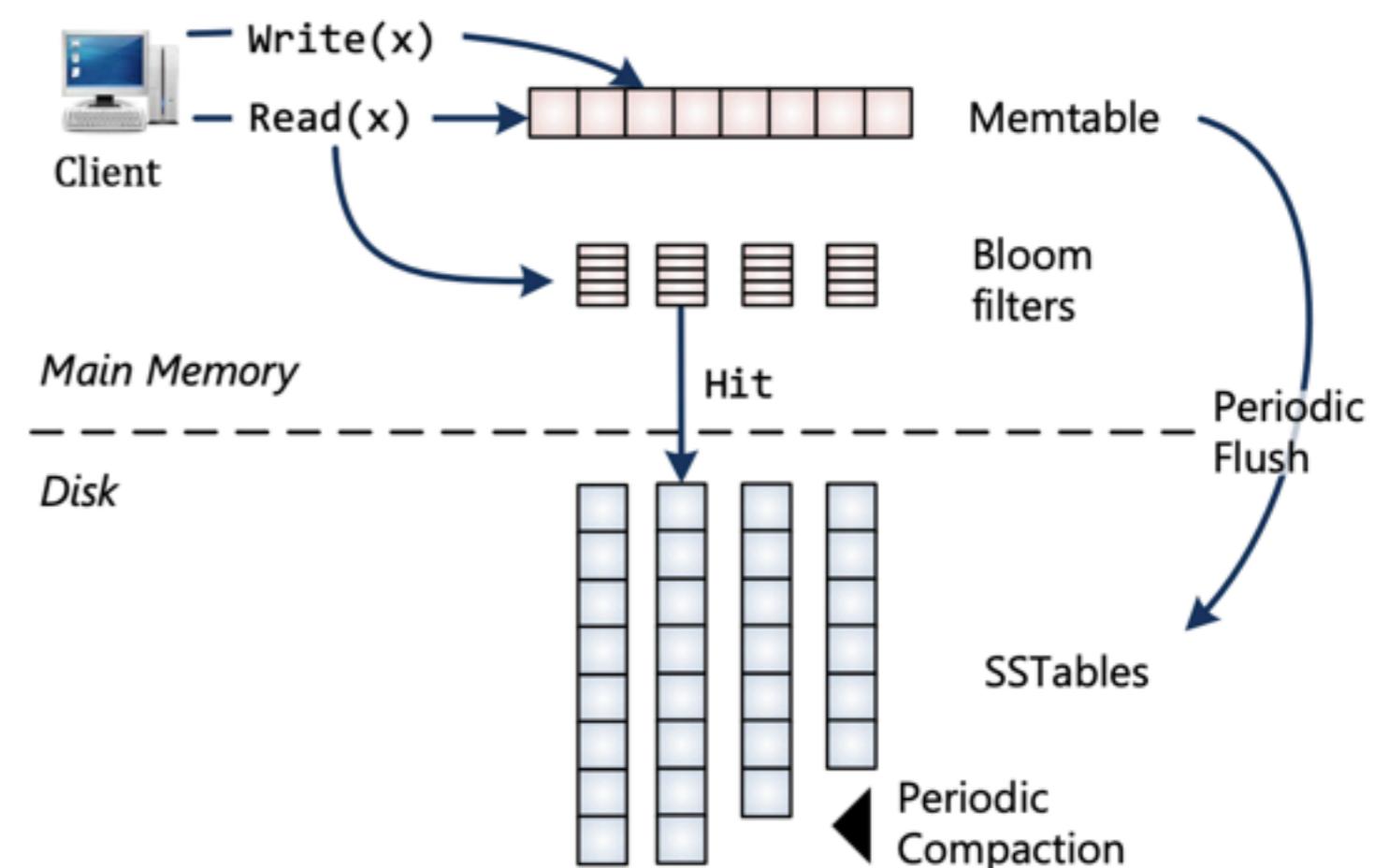
principio de fusionar y compactar se llaman LSM (log-structured merge-tree).





- \* Estructura de datos de memoria eficiente para aproximar el contenido de un conjunto.
- \* Te dice si una key no aparece en una base de datos, ahorrando las búsquedas innecesarias a disco para keys que no existen.

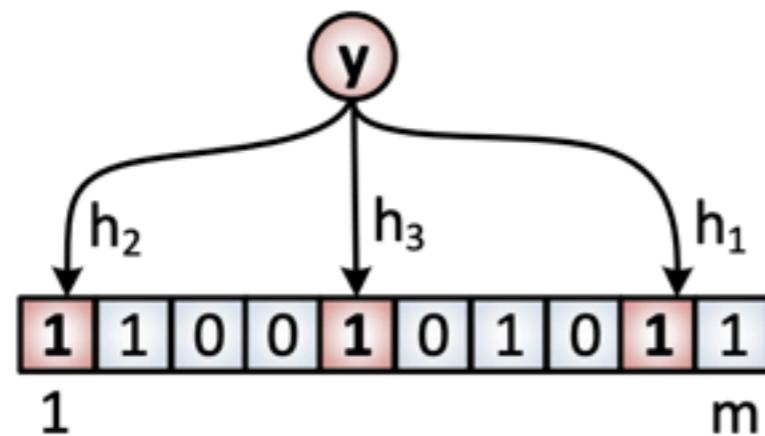
# Optimización Bloom Filters



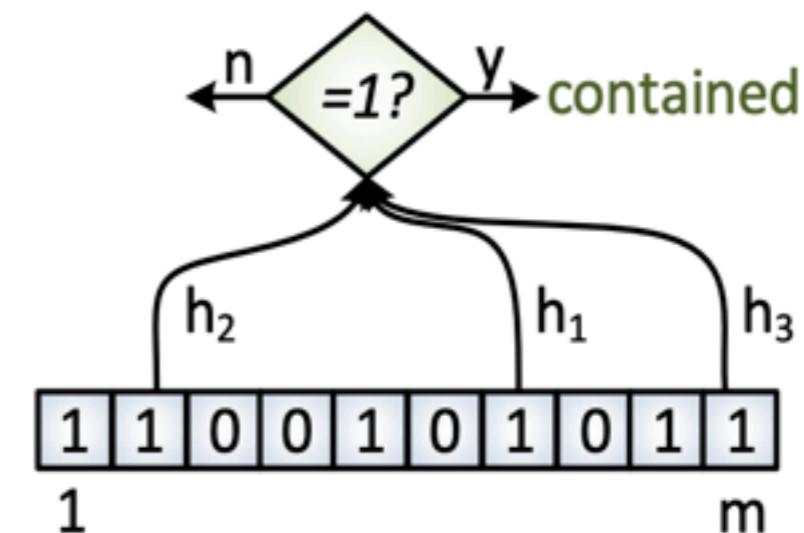


# Optimización Bloom Filters

- Arreglo de largo  $m$  y  $k$  funciones de hash independientes.
- Insert (obj) : agregar al conjunto
- Contiene (obj): puede dar falso positivo
- Trade-off: tamaño  $m$ .



**Insert y**

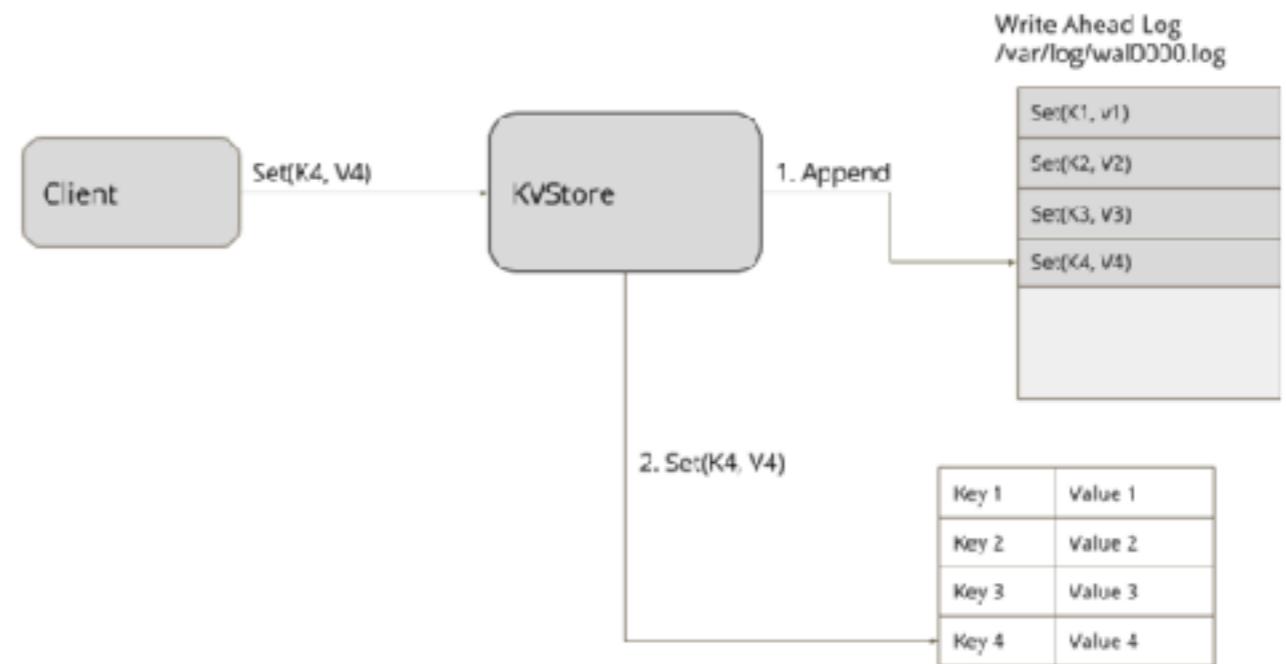


**Query x**



# Write-ahead log (WAL)

- Confiabilidad, ¿qué pasa cuando hay fallas en memtable que está en memoria?
- Debe escribir todos los registros de log antes de una escritura en memtable.
- WAL debe estar en almacenamiento persistente, como disco.
- Se puede recrear todo lo perdido a través de re-hacer los registros del WAL.



© 2019 Thoughtworks



# Amazon y sus requerimientos



- Aplicación es una plataforma mundial de e-commerce que sirve millones de clientes en momentos de alto tráfico.
- Se requiere un sistema de almacenamiento altamente disponible de tipo key-value.
- Se sacrifica consistencia en escenarios de falla: Las fallas son el caso normal.
- En el caso de Amazon el ambiente de ejecución es confiable.
- No requieren espacio de nombres jerárquico o esquemas relacionales complejos.
- Amazon tiene aplicaciones sensibles a la latencia (99.9% de las lecturas y escrituras desean que sean en unos pocos cientos de milisegundos)
- Posee varios datacenters en el mundo.
- Requiere que almacenamiento altamente escalable para soportar crecimiento continuo.



# AWS DynamoDB

- \* Desarrollado por Amazon en el 2008
- \* Ha evolucionado a DynamoDB desde la presentación de este artículo. Número actuales que soporta:
  - 7 millones de transacciones por segundo
  - Hasta 54.4 millones de transacciones por segundo en eventos como CyberMonday/ BlackFriday



## Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jempani, Gunevardhan Kakulapati,  
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall  
and Werner Vogels

[Amazon.com](http://Amazon.com)

### ABSTRACT

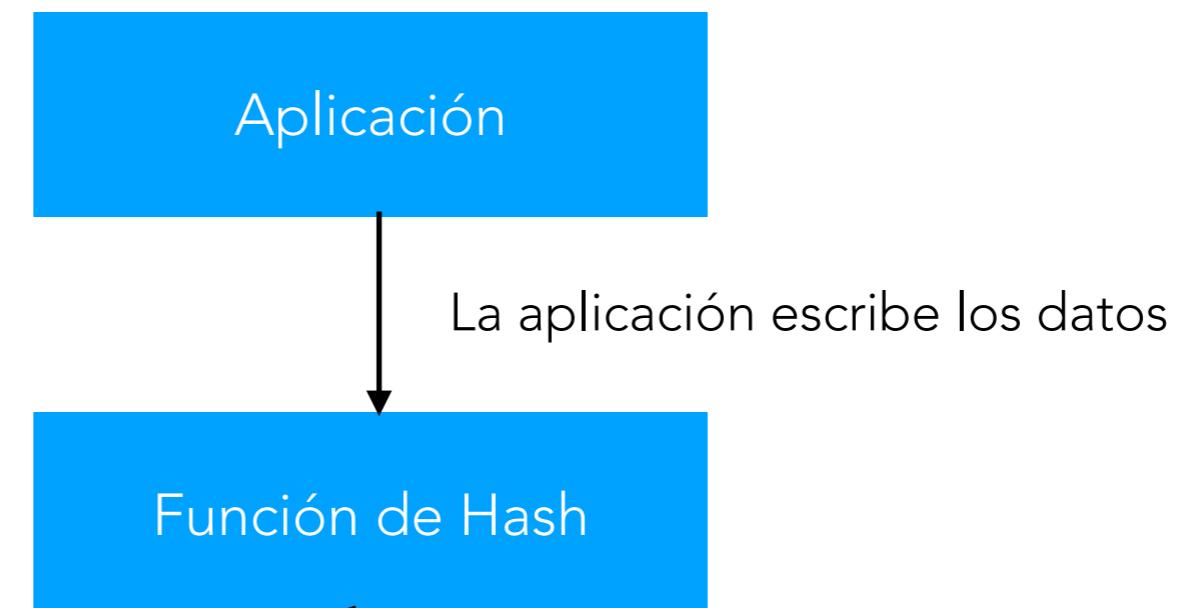
Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service-oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.



La partición para los datos es determinada al hacer hash de la clave primaria de los datos.



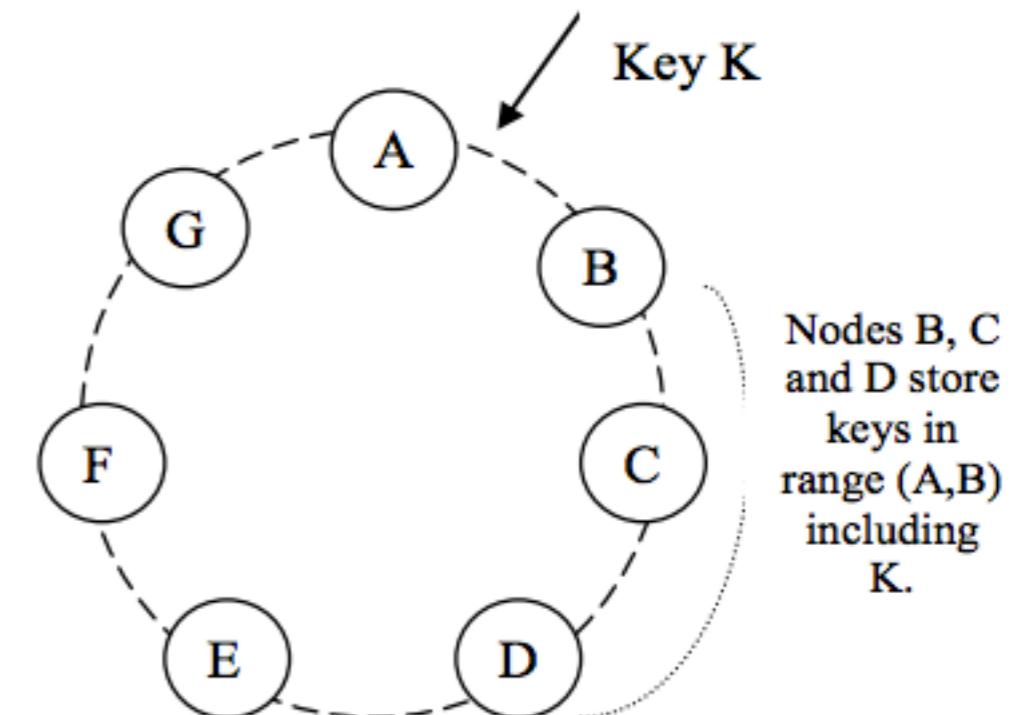
.....

Key	Value
161305173	{name: 'Ivor Merritt', address: 'Ap #527-9960 Vel St.', city: 'Lauw', zip: '5624', country: 'Peru'}
162307206	{name: 'Cade Nguyen', address: '486, 6221 Et St.', city: 'Barnstaple', zip: '10903', country: 'Ukraine'}



# Particionamiento en Dynamo

- Se partitionan los datos dinámicamente sobre un conjunto de nodos.
- **Consistent hashing:**
  - El rango de salida de una función de hash MD5 es tratada como un espacio circular fijo (anillo).
  - A cada nodo se le asigna un valor aleatorio dentro de este espacio, que representa su posición en el anillo.
  - Cada item de datos es identificado con una clave y es asignado a un nodo haciendo hash del objeto mismo: se asigna al primer nodo con identificador mayor al hash en el sentido del reloj.
  - Cada nodo es responsable por la región entre él y su predecesor en el anillo.



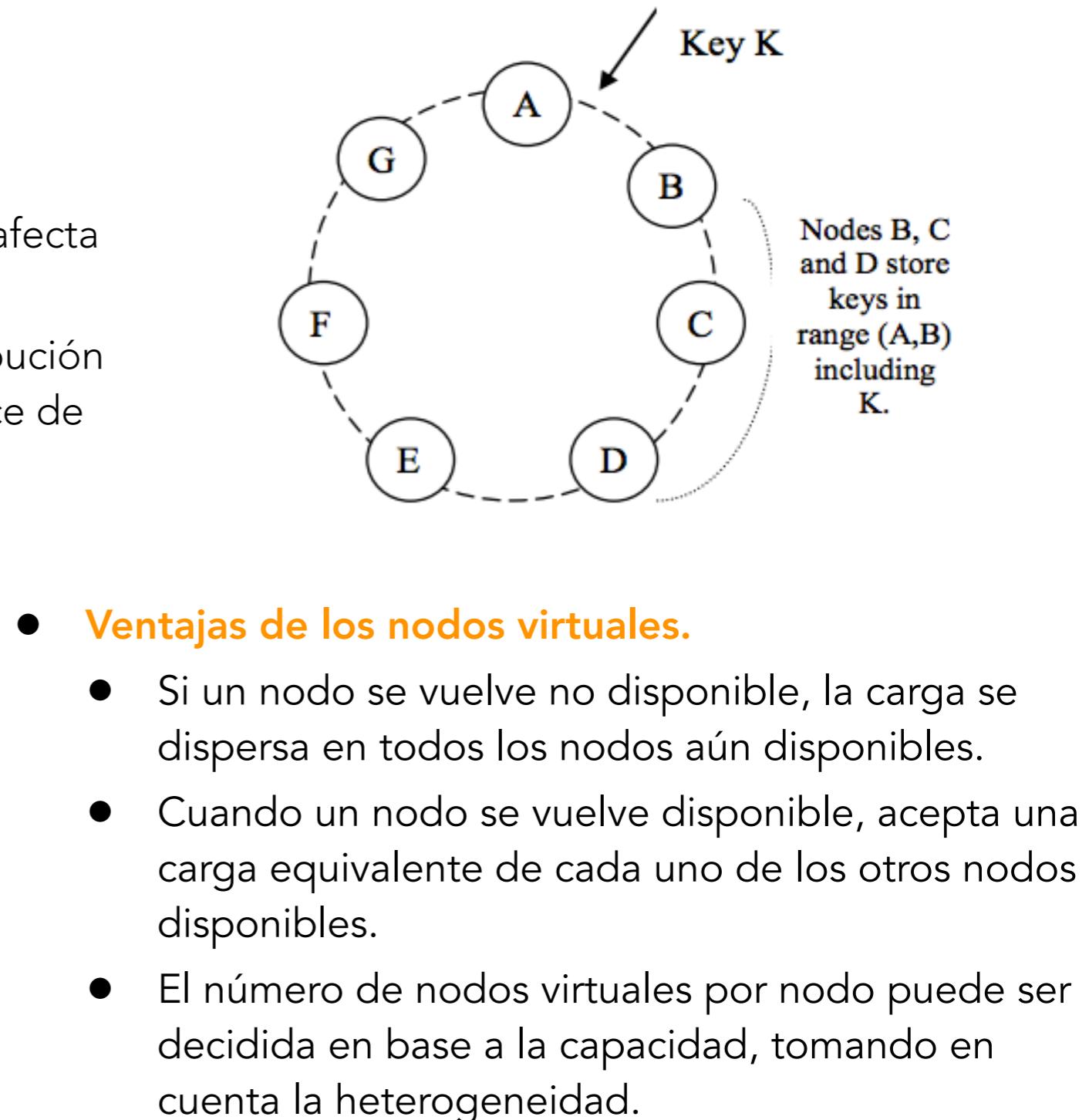


# Algoritmo de Particionamiento

- **Escalado incremental:**

- Ventajas: La salida o llegada de nodos afecta a sus vecinos inmediatos solamente.
- El posicionamiento aleatorio produce distribución de datos no uniforme, y por ende desbalance de carga.
- No considera la heterogeneidad en el rendimiento de los nodos.

Variante del hash consistente: A cada nodo se le asignan múltiples puntos en el anillo:  
Nodos virtuales.



- **Ventajas de los nodos virtuales.**

- Si un nodo se vuelve no disponible, la carga se dispersa en todos los nodos aún disponibles.
- Cuando un nodo se vuelve disponible, acepta una carga equivalente de cada uno de los otros nodos disponibles.
- El número de nodos virtuales por nodo puede ser decidida en base a la capacidad, tomando en cuenta la heterogeneidad.



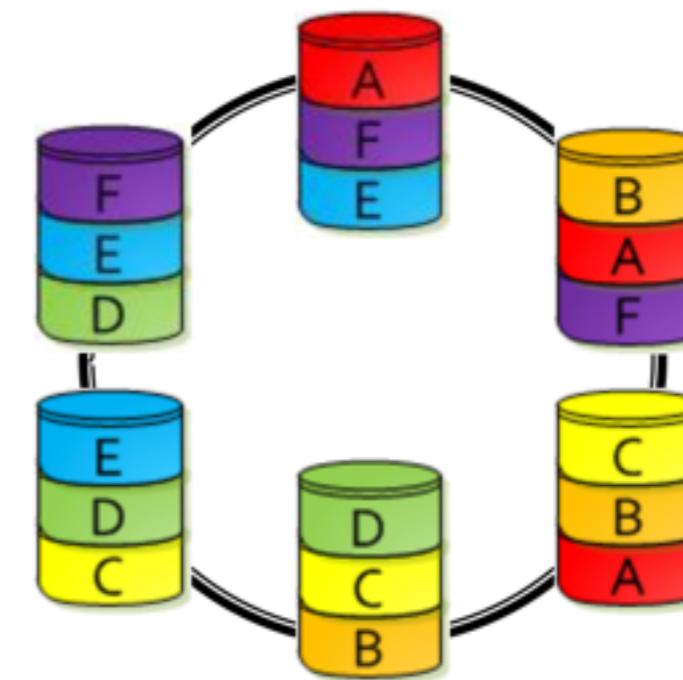
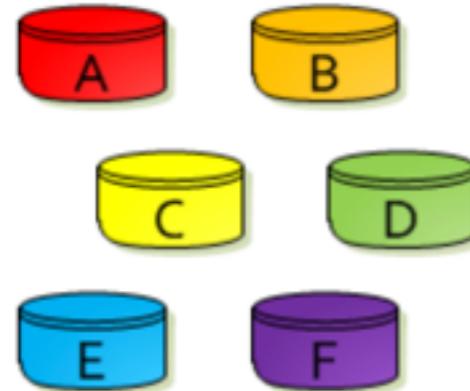
# Sharding

- **Basado en Hash:** El hash determina la partición
  - Pro: Distribución uniforme
  - Contra: No hay localidad de los datos.
  - Implementado en Dynamo, Redis, Cassandra, Azure Table, MongoDB, Riak
- **Basado en Rango:** Asigna rango sobre los campos para particionar
  - Pro: Permite Range Scans y ordenamiento
  - Contra: Se requiere reparticionamiento y balance
  - Implementado en BigTable, HBase, DocumentDB, Hypertable, MongoDB, RethinkDB, Espresso



# Particiones y replicación

\* Cada partición es replicada N veces.





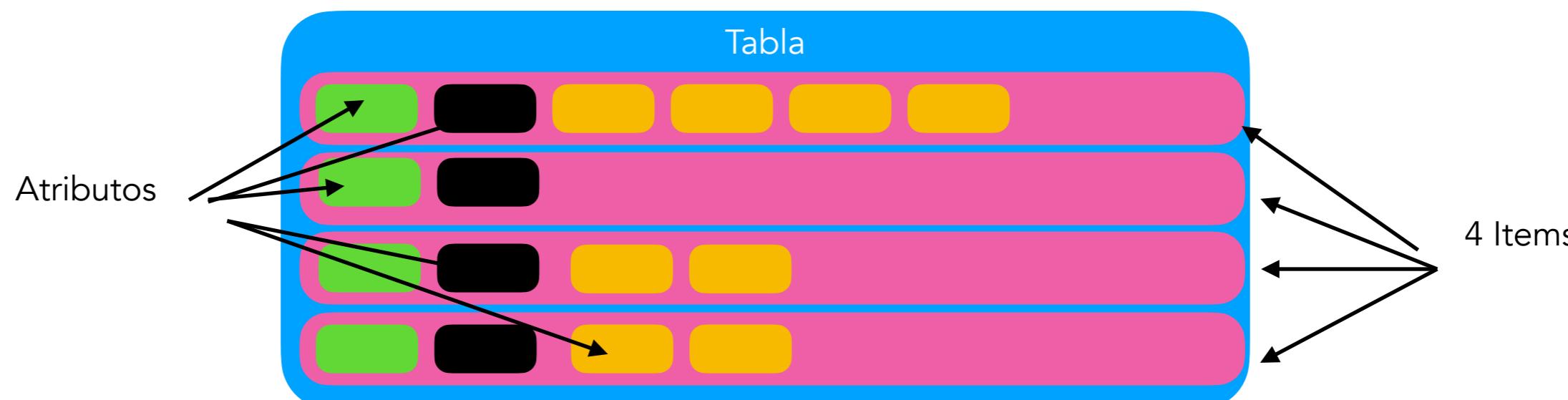
# Replicación: ¿Cuándo?

- Asíncrona (lazy)
  - Escrituras se aceptan inmediatamente
  - Pro: escrituras rápidas, sin coordinación requerida
  - Contra: Inconsistencia
  - Implementado en DynamoDB, Riad, CouchDB, Redis, Cassandra, Voldemort, MongoDB, RethinkDB
- Síncrona (eager)
  - El nodo que acepta escrituras propaga antes de hacer ACK.
  - Pro: Consistencia
  - Contra: Necesita protocolo de commit, no tiene disponibilidad bajo particiones de red.
  - Implementado en BigTable, HBase, Accumulo, CouchBase, MongoDB, RethinkDB



# DynamoDB: Modelo de datos

- Modelo incluye Tablas, Items y Atributos.
  - **Tabla:** colección de Items. Solo se requiere primary key.
  - **Item:** colección de atributos.
  - **Atributos:** pueden ser agregados en tiempo de ejecución.
    - Los items de una tabla pueden tener diferentes atributos.
    - Cada atributo es un par clave-valor.

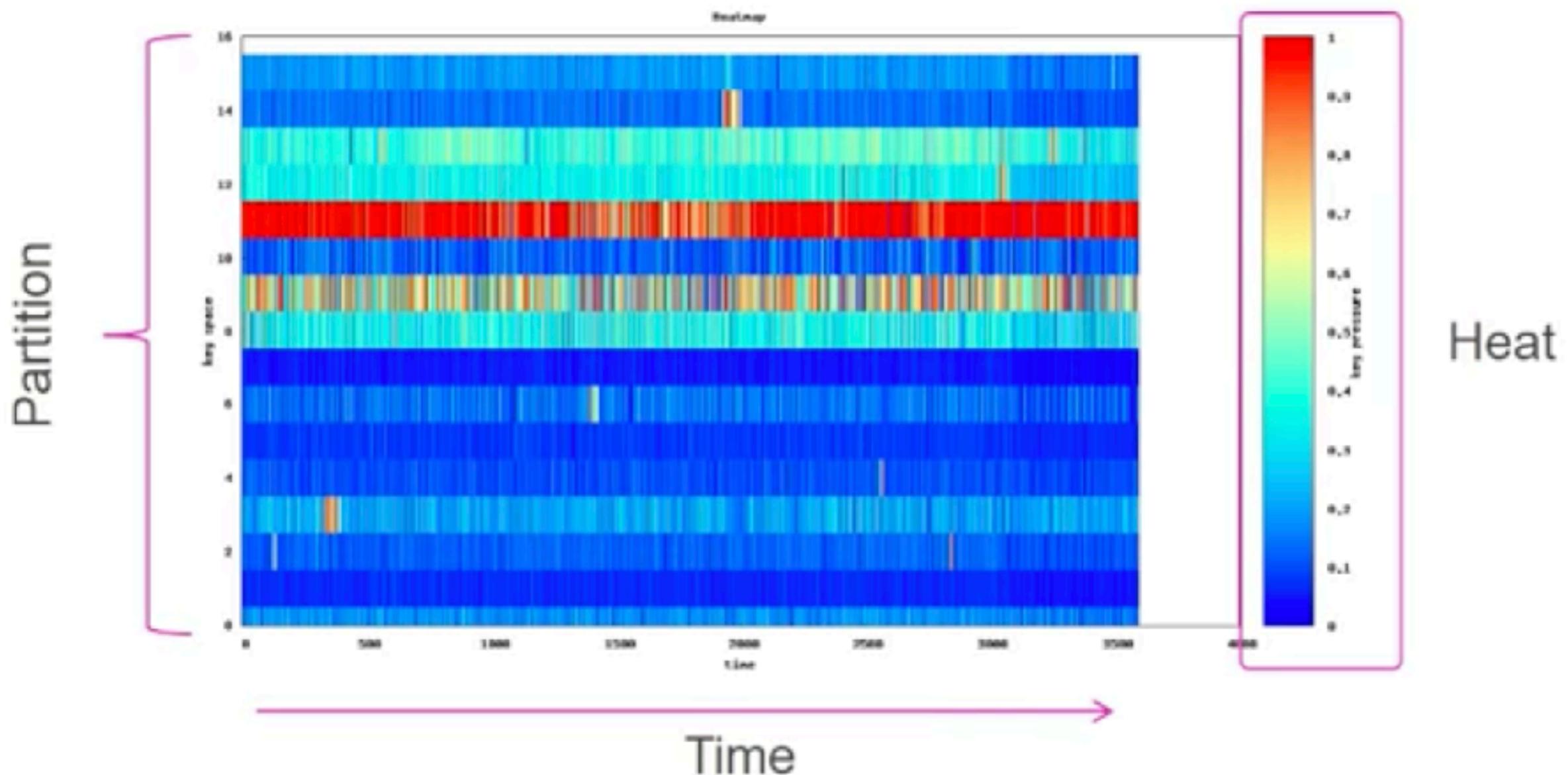




UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Particiones desbalanceadas





Joins en Modelos relacionales ->  
Demasiado lento, muchas lecturas en  
diferentes partes.

Joins en NoSQL -> No hay necesidad, la  
tabla guarda la vista completa que se  
necesita.

# Importante

## SQL vs. NoSQL design pattern

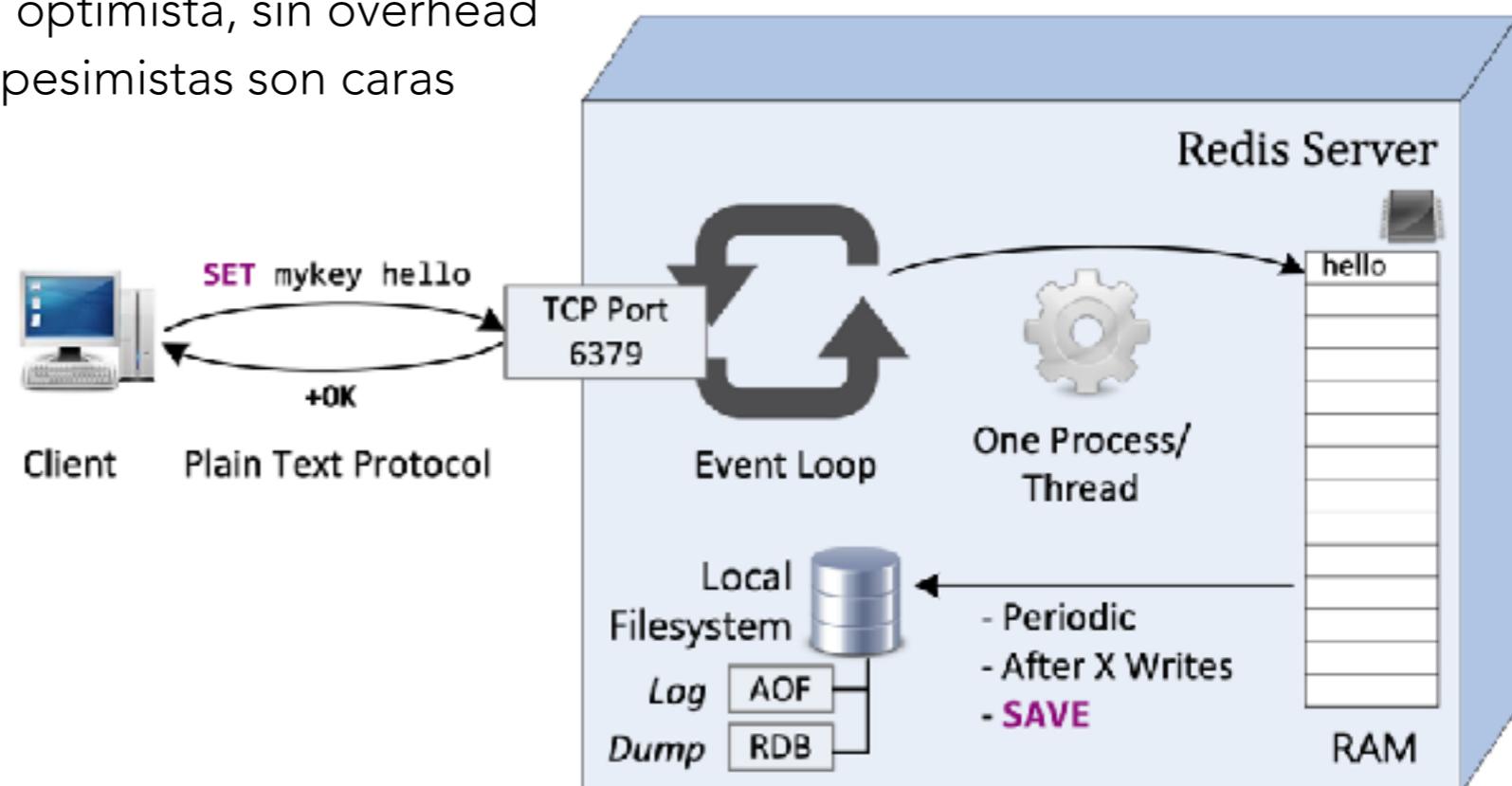


Desnormalizar  
Índices secundarios  
Claves compuestas



- Remote Dictionary Server
- Almacén en memoria de key-value
- Replicación Maestro-esclavo asíncrono.
- Persistencia configurable a través de logs (Append Only File) y snapshots (Redis Database Snapshot).
- Un solo hilo (single-threaded)
- Alto rendimiento > 100K ops/seg: optimista, sin overhead para confiabilidad. Transacciones pesimistas son caras (overhead).

# Redis: key-value





# Bases de datos de documentos

- Almacenan datos semi-estructurados en forma de documentos.
- Documentos son codificados con diferentes estándares como JSON, XML, BSON, YAML, etc.
- No hay requerimiento estricto de esquema, pero los documentos que se almacenan son similares entre sí.
- Los documentos tienen campos y valores.
- Cada documento se identifica con una clave única o ID.
- Útil cuando toda la información de un objeto necesita ser recuperado en su conjunto.
- Ejemplo: todos los datos de un producto.



- Base de datos no relacional orientada a documentos.
- Unidad básica de datos en MongoDB es el documento.
- Los documentos se agrupan formando colecciones.
- Las colecciones no tienen un esquema fijo, pueden tener diferentes conjunto de clave, valor.

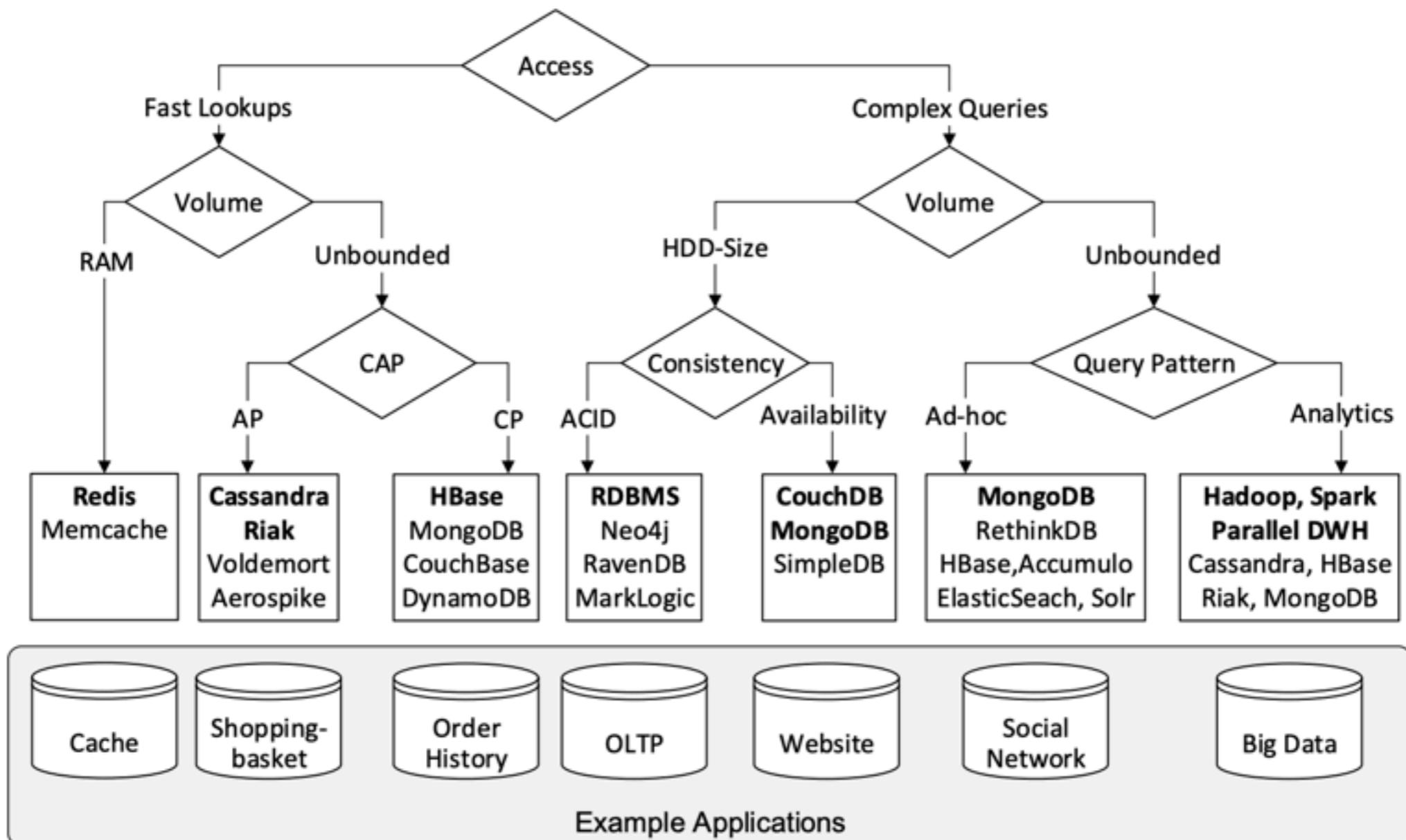
ID	Document
56fd4f59849f6367af489537	<pre>{   "title" : "Motorola Moto G (3rd Generation)",   "features" : [     "Advanced water resistance",     "13 MP camera",     "5in HD display",     "Quad core processing power",     "5MP rear camera",     "Great 24hr battery",     "4G LTE Speed"   ],   "specifications" : {     "Color" : "Black",     "Size" : "16 GB",     "Dimensions" : "0.2 x 2.9 x 5.6 inches",     "Weight" : "5.4 ounces"   },   "price" : 219.99 }</pre>





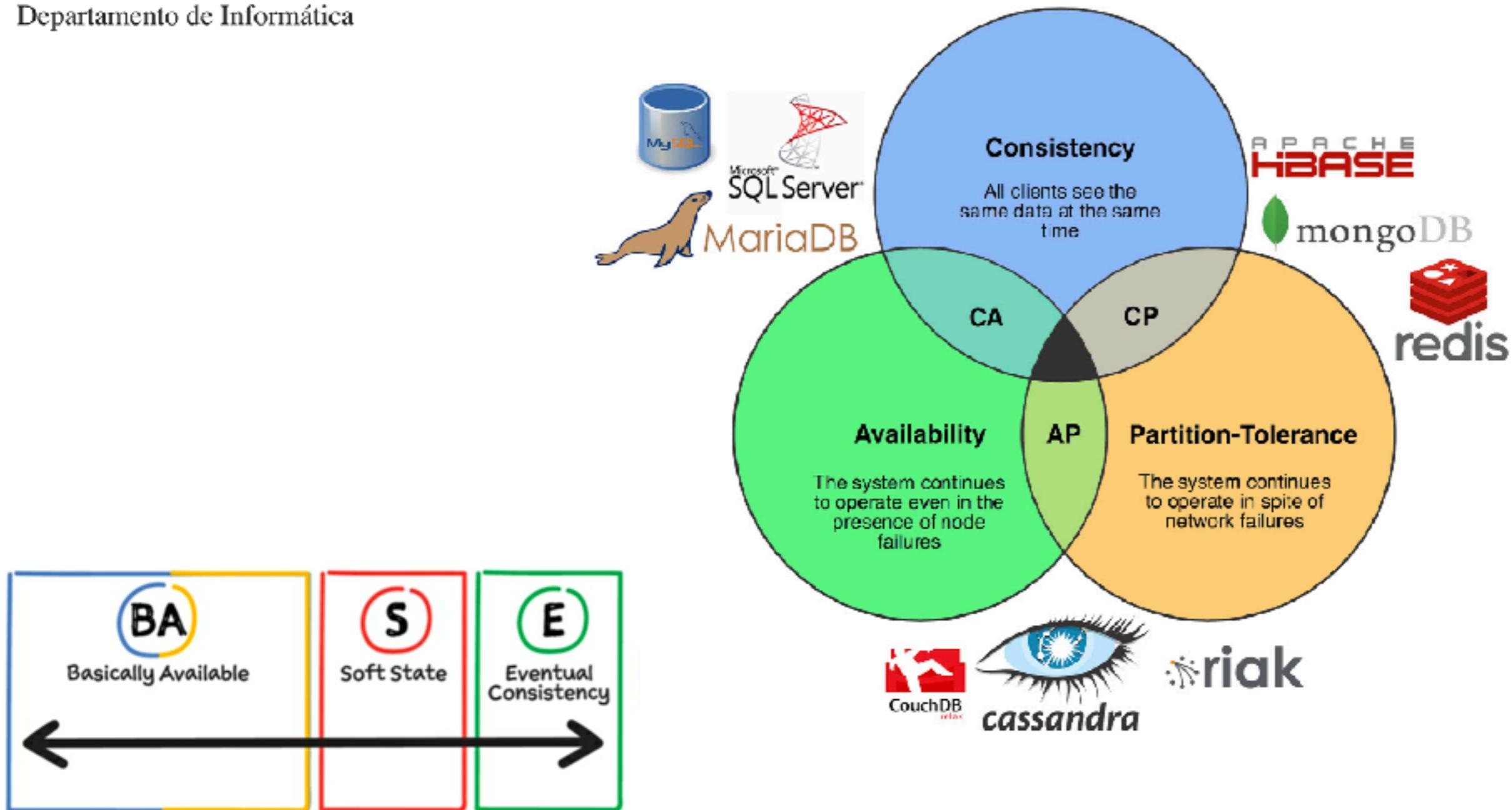
# Árbol de Decisión NoSQL

## Gessert et al.





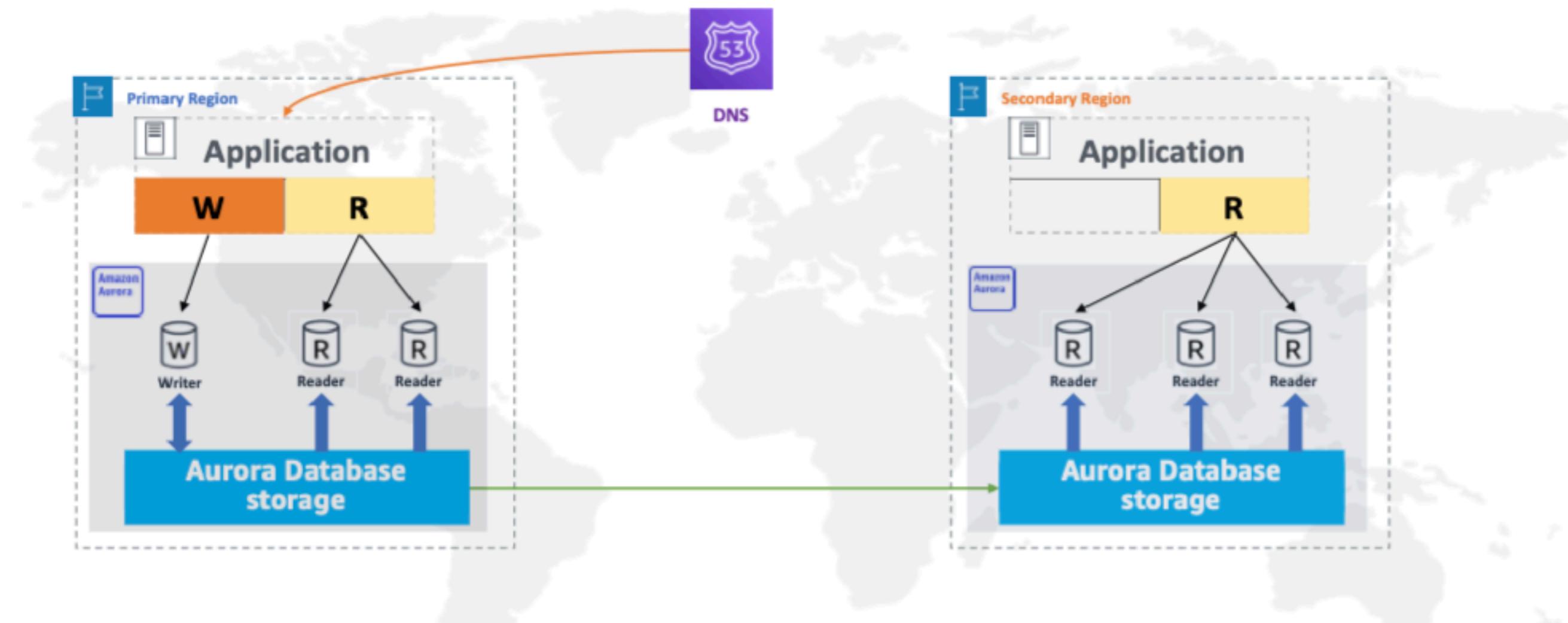
# Elegir en base a teorema CAP



Choosing the right NoSQL database for the job: a quality attribute evaluation, Lourenco, Cabral, Carreiro, Viera, BeranrdinoJournal of Big Data, 2015.



# Global Databases



Aurora de AWS es una DB Global, pero no escala en escrituras, sino que solo en lecturas.



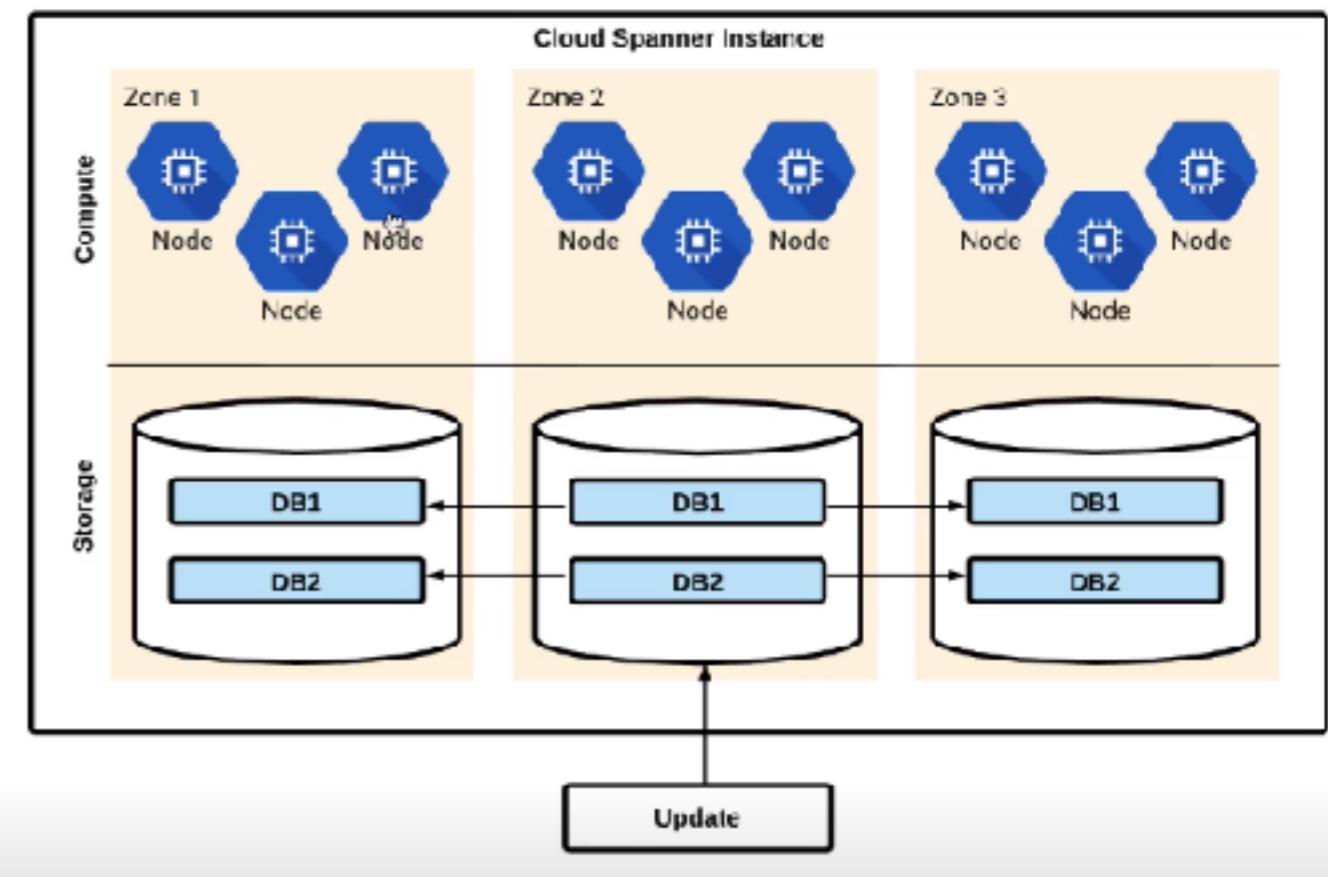
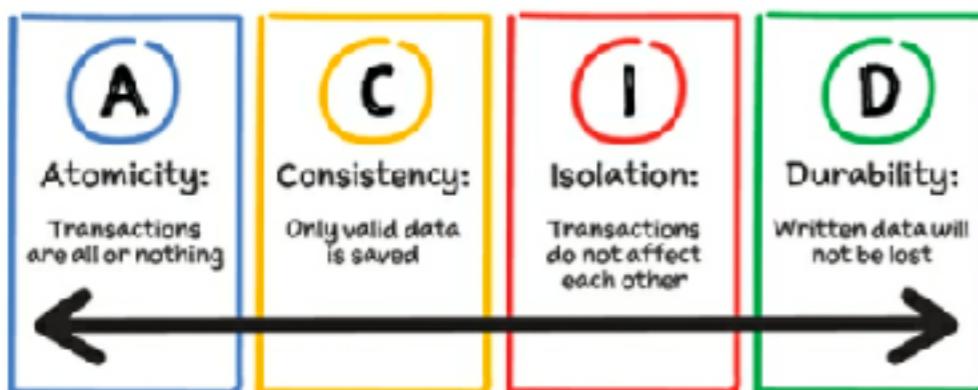
# Aparición New SQL

	RDBMS	NoSQL	NewSQL
Esquema	Esquema relacional / Tabla	Sin esquema estricto	Ambos
Escalabilidad	Lecturas escalables	Escalable en lecturas y escrituras, horizontalmente escalable	Escalable en lecturas y escrituras, horizontalmente escalable
Alta Disponibilidad	Configurable	Automática	Incluida
ACID-CAP	ACID	CAP	ACID
Ejemplos	Oracle, MySQL, PostgreSQL	MongoDB, Cassandra, Redis, HBase	Google Spanner, VoltDB
Casos de uso	Financiero	Big Data, IoT, Redes sociales	Juegos en línea, comercio electrónico, industrias telecom

<https://medium.com/rabiprasadpadhy/google-spanner-a-newsql-journey-or-beginning-of-the-end-of-the-nosql-era-3785be8e5c38>



- Base de datos SQL altamente disponible.
- Escala a petabytes en volumen de datos.
- Asigna timestamp globales y consistentes en tiempo real.
- Dice ser altamente consistente y disponible. Pero en caso de particiones elige consistencia.
- Spanner corre en la red privada global de Google.





- Usa sharding automático llamado splits y replicación.

## Primary building blocks - Data splits

Key Space is partitioned into splits

Each split is replicated across zones

Each split has an elected leader

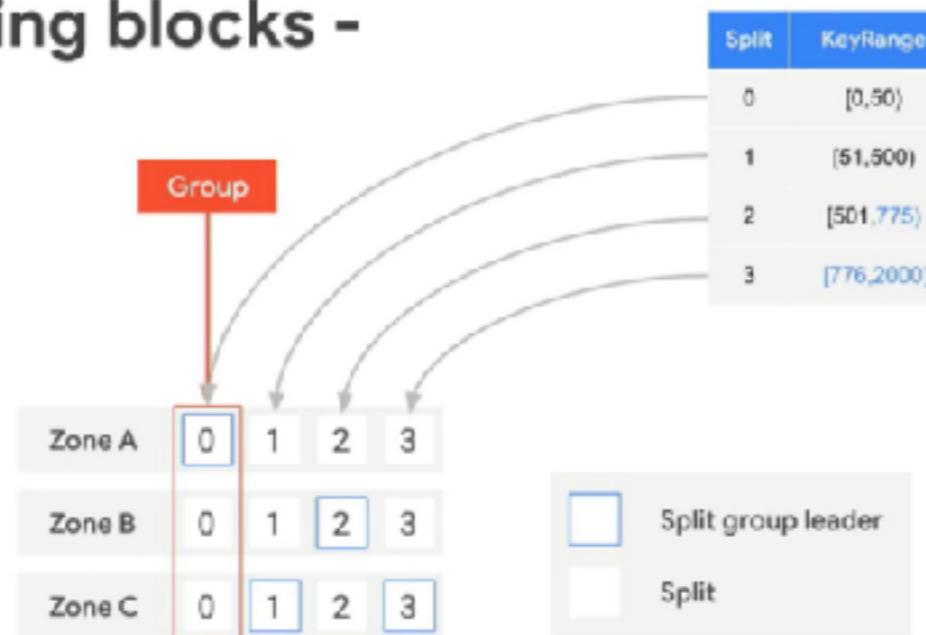
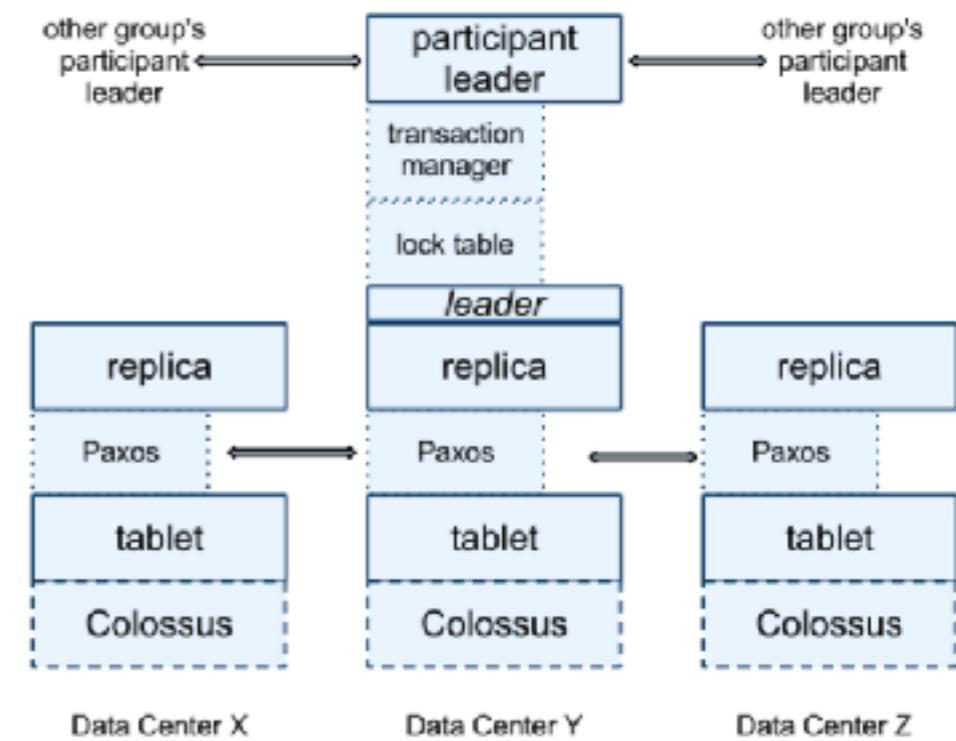


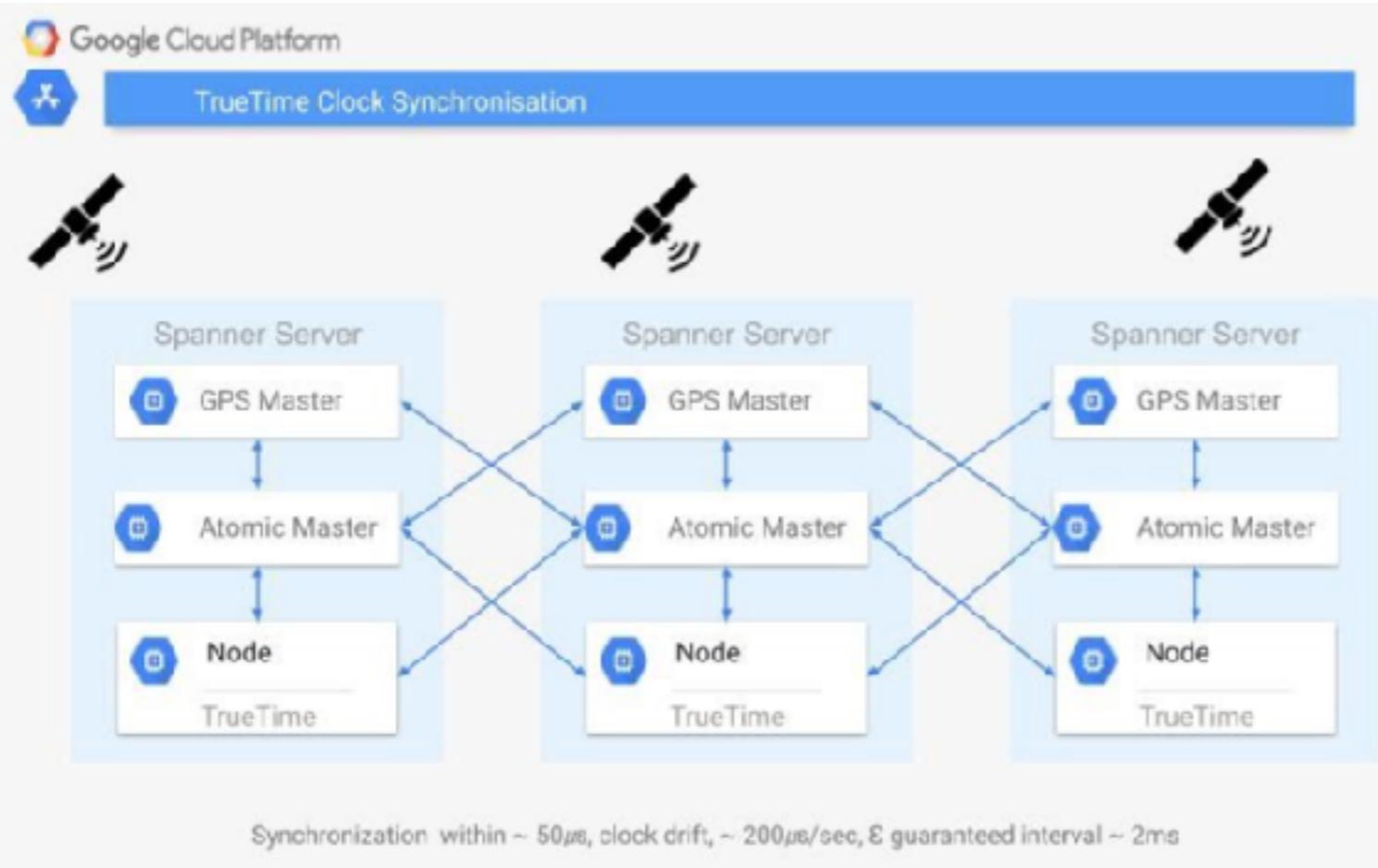
Image Credit: Google Cloud

- Usa Paxos para consenso y Colossus para almacenamiento de tablets.





- Problema clásico de sincronización de relojes en sistemas distribuidos.
- Mitigado con Sincronización relojes con TrueTime



# Spanner

