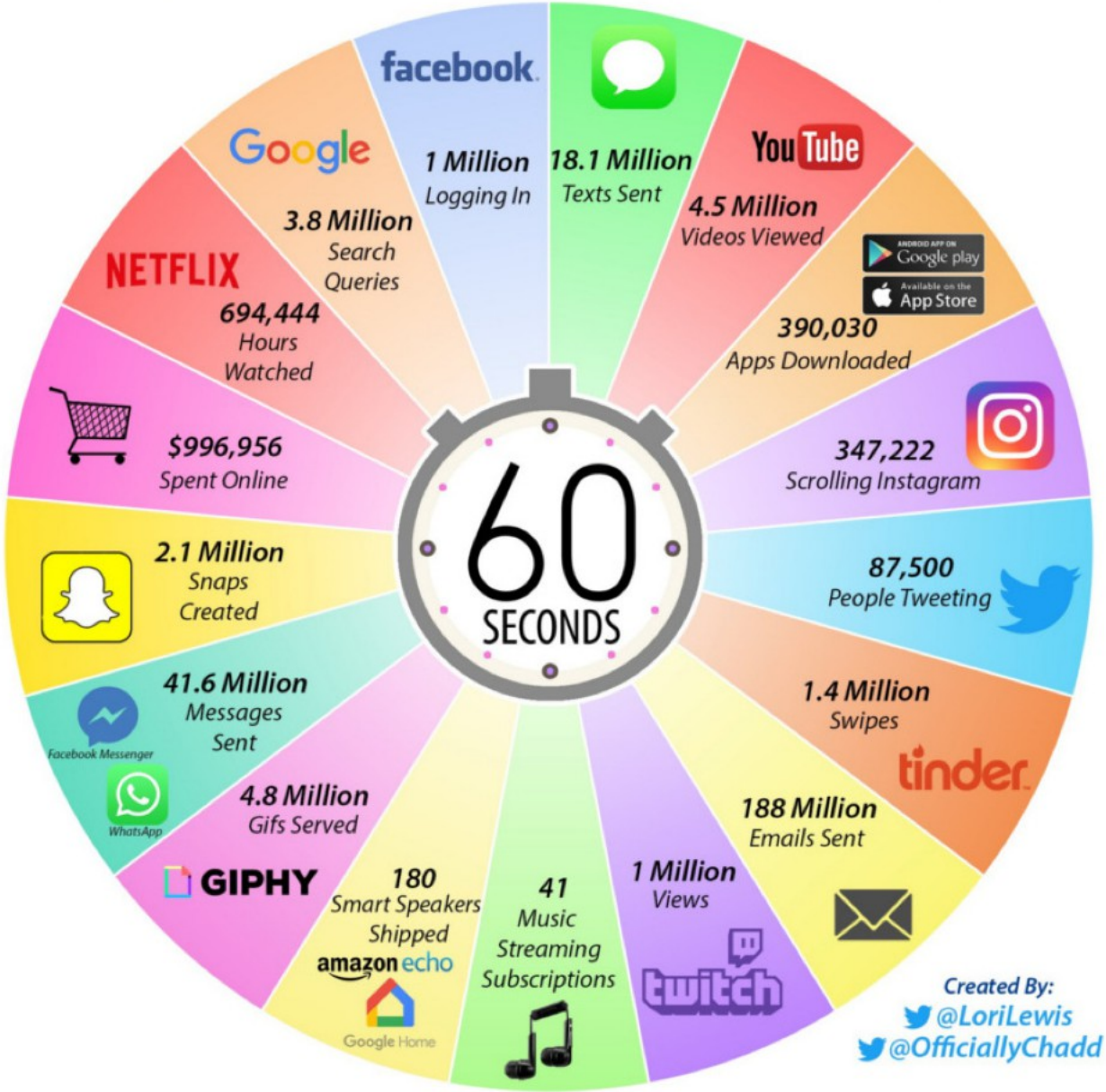
# Big Data y Ecosistema Hadoop


Fuente: http://www.usine-digitale.fr/

# 2019 This Is What Happens In An Internet Minute




Created By:
@LoriLewis
@OfficiallyChadd


## Data Management

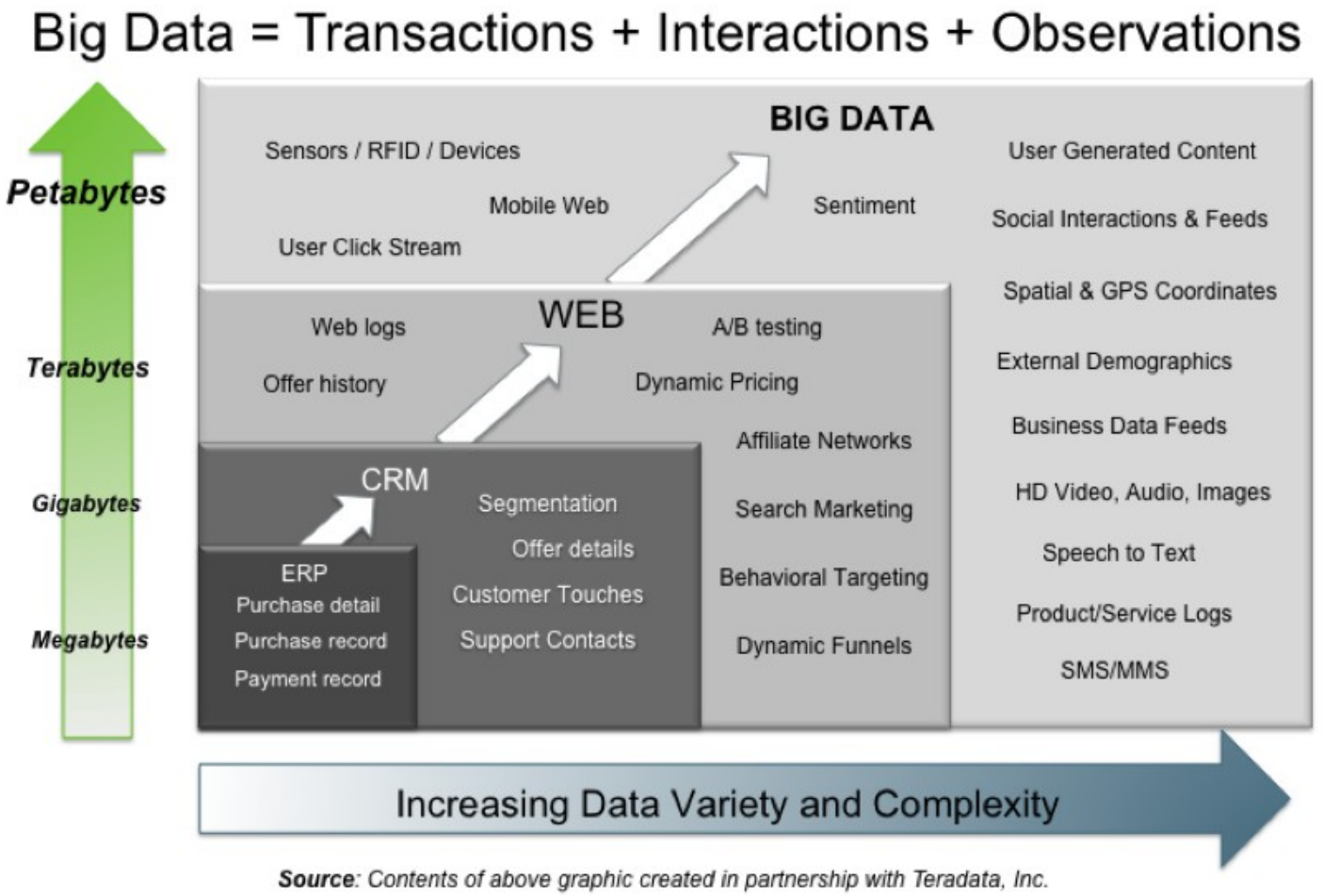
- Aquisition & recording
- Extraction, cleaning & annotation
- Integration, aggregation & representation

## Analytics

- Modelling & analysis
- Interpretation

Beyond the hype: Big data concepts, methods, and analytics. A. Gandomi | M.Haider , IJIM 2015

# Sobre la naturaleza del Big Data

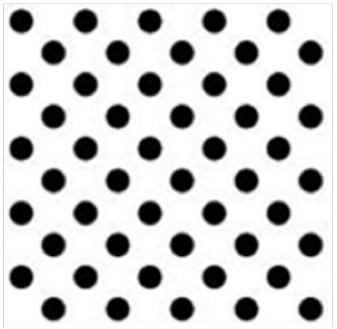
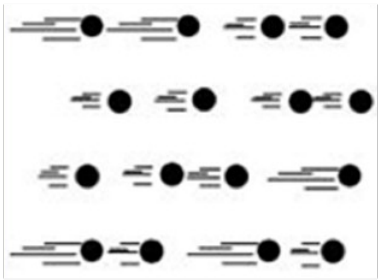
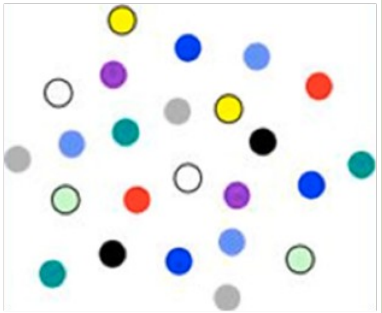
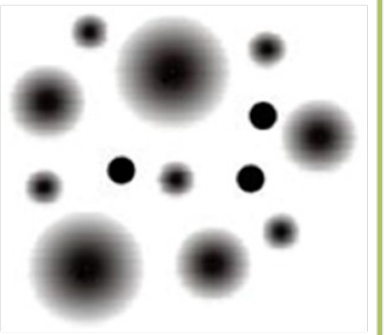



Fuente: http://hortonworks.com/blog/7-key-drivers-for-the-big-data-market/

# Big Data y la frescura del dato

- El "mundo viejo" trabaja sobre transacciones, datos históricos
- Este nuevo mundo en algunos casos trabaja sobre datos casi en *real-time*
  - *Ej:*
    - *stream analytics*
    - *sentiment analysis*

# Hay una 5ta “V”

| Volume | Velocity | Variety | Veracity | Value |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **Data at Rest** | **Data in Motion** | **Data in Many Forms** | **Data in Doubt** | **Data into Money** |
| Terabytes to Exabytes of existing data to process | Streaming data, requiring milliseconds to seconds to respond | Structured, unstructured, text, multimedia,… | Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations | Business models can be associated to the data |

# La 5ta V

**Mantenimiento Predictivo**

**300 sensores**

**200 GB de datos por día**

**Se transmiten y analizan en Dinamarca**

# La 5ta V

## Aplicación en Agro: John Deere

**El desafío de IA es el agro**

**Uno de los principales clientes cloud en el mundo**

**De 5MM a 15MM de mediciones por segundo**

**130K máquinas conectadas**

**Más de 100 empresas agregan datos adicionales: clima, imágenes áreas, análisis de suelos**

IEEE: spectrum_na_february_2020 - John Deere's Quest to Solve Agriculture's Deep-Learning Problems: https://read.nxtbook.com/ieee/spectrum/spectrum_na_february_2020/john_deere_s_quest_to_solve_a.html

# Big Data para la academia

Michael Stonebraker considera que *Big Data* quiere decir al menos tres cosas:

- gran **volumen**
  - análisis de datos simple (SQL)
  - análisis de datos complejo (no SQL)
- gran **velocidad**
  - "drink from a fire hose"
- gran **variedad**
  - intregrar una gran cantidad de fuentes diversas

M. Stonebraker Big Data Means at Least Three Different Things...., http://www.nist.gov/itl/ssd/is/upload/NIST-stonebraker.pdf
http://www.bizjournals.com/boston/blog/startups/2013/03/michael-stonebraker-what-is-big-data.html

# V de “Volumen”

# Almacenamiento tradicional

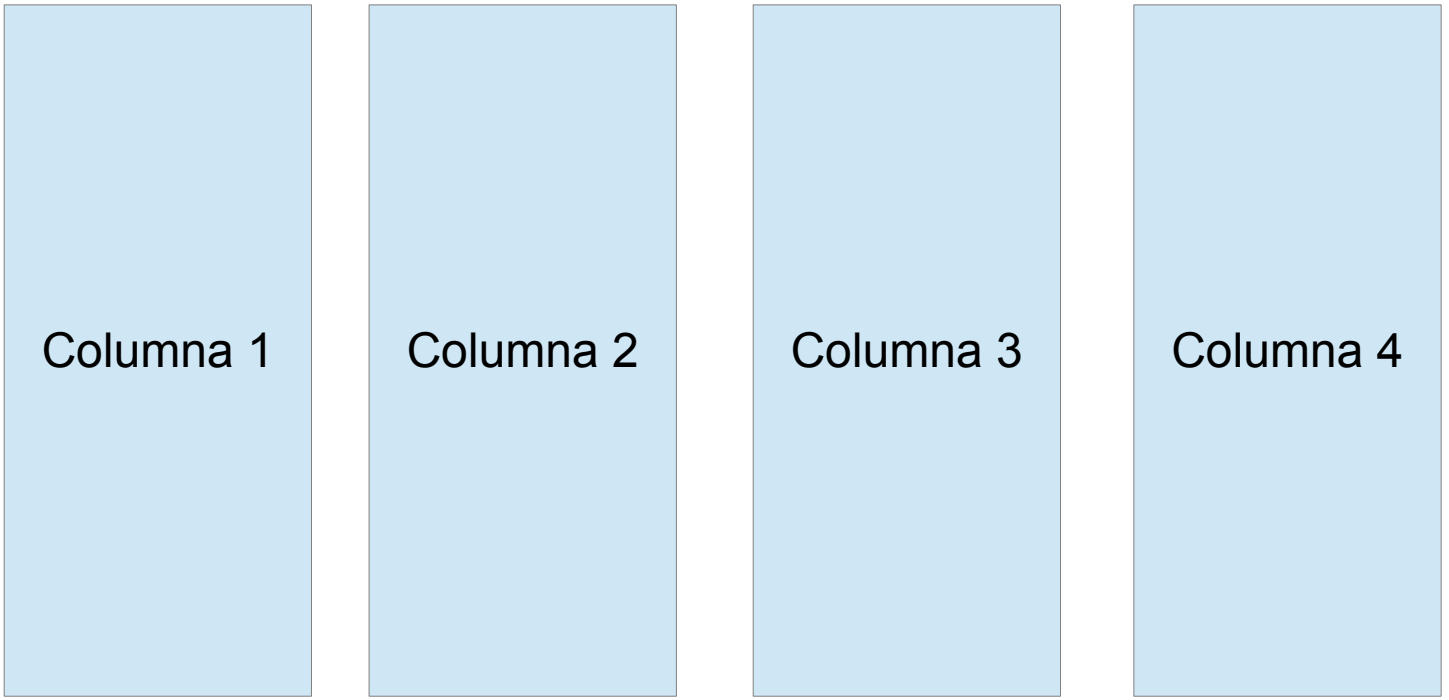

**Almacenamiento por filas**

- Los registros (filas) se almacenan de manera secuencial en el disco
- Los sistemas de Data Warehouse tradicionales están orientados a almacenamiento por filas

# Almacenamiento tradicional

| Número | Apellido | Nombre | Clave |
|---|---|---|---|
| 1. | Skywalker | Luke | 3FN-Z768 |
| 2 | Kenobi | Obi-wan | 7TR-K345 |
| 3 | Organa | Leia | 8NN-R266 |

# Almacenamiento columnar




**Almacenamiento por filas**

- Los registros (filas) se almacenan de manera columnar
- El siguiente bloque en disco contiene el mismo atributo, pero del siguiente registro

# Almacenamiento columnar

| Número | 1. | 2. | 3. |
|---|---|---|---|
| Apellido | Skywalker | Kenobi | Organa |
| Nombre | Luke | Obi-wan | Leia |
| Clave | 3FN-Z768 | 7TR-K345 | 8NN-R266 |

```
1, Skywalker, Luke, 3FN-Z768; 2, Kenobi, Obi-wan, 7TR-K345; 3, Organa, Leia, 8NN-R266
```

```
1, 2, 3; Skywalker, Kenobi, Organa; Luke, Obi-wan, Leia; 3FN-Z768, 7TR-K345, 8NN-R266
```

# Almacenamiento columnar

| Número | 1. | 2. | 3. |
|---|---|---|---|
| Apellido | Skywalker | Kenobi | Organa |
| Nombre | Luke | Obi-wan | Leia |
| Clave | 3FN-Z768 | 7TR-K345 | 8NN-R266 |

```
1, Skywalker, Luke, 3FN-Z768; 2, Kenobi, Obi-wan, 7TR-K345; 3, Organa, Leia, 8NN-R266
```

```
1, 2, 3; Skywalker, Kenobi, Organa; Luke, Obi-wan, Leia; 3FN-Z768, 7TR-K345, 8NN-R266
```

# El próximo paso el Data Warehouse

- Operaciones matemáticas complejas
  - Machine learning
  - Clustering
  - Detección de tendencias

- Detrás de esto existen
  - Multiplicaciones de matrices
  - Descomposición QR
  - Regresiones lineales

# Big Data – Analytics
# Ejemplo

- Considere los precios de cierre de todos los días de trading de los últimos 5 años para los stocks de acciones A y B

- ¿Cuál es la covarianza entre las dos series temporales?

$$s_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \frac{1}{n}\sum_{i=1}^{n}x_i y_i - \bar{x}\cdot\bar{y}$$

# Big Data – Analytics
# Ejemplo (II)

Ahora para 4000 stocks

Los datos están representados en la siguiente matriz de 4000 x 1000

| Stock | t1 | t2 | t3 | t4 | t5 | ... | t1000 |
|---|---|---|---|---|---|---|---|
| S1 | | | | | | | |
| S2 | | | | | | | |
| ... | | | | | | | |
| S4000 | | | | | | | |

# Big Data – Analytics
# Ejemplo (II)

Stock * $Stock^T$

Intentemos hacer esto en SQL ¿Alguna idea?

# Big Data – Analytics

## Descripción del trabajo de un DS

- Buscar y limpiar los datos (la mayor parte del tiempo)

```
Until (cansado o aburrido) {
    Data managment operations(s)
    Complex analytics operations(s)
}
```

# Big Data – Analytics Soluciones

- R, SAS, SPSS, Matlab
  - Gestión de datos básico o inexistente
  - Generalmente se trabaja sobre archivos planos, directamente sobre el file system
- Base de datos relacional
  - Álgebra línea muy pobre o inexistente
- Base de datos relacional con motor de álgrebra relacional
  - “pesadilla de dos sistemas”

# Big Data – Analytics Soluciones

- Bases de datos de Arrays
  - SciDB, PostGIS, Oracle GeoRaster, MonetDB
- Operaciones SQL sobre arrays
- Incluye librerías analíticas

# Big Data – Analytics Soluciones

```
CREATE ARRAY products <name:string,price:float,sold:datetime>
[i=0:*,1,0];

CREATE ARRAY A [i=0:99,10,0, j=0:99,10,0];

SELECT j FROM A WHERE j > 3 and j < 7;
```

# Las soluciones Big Data

# MapReduce

# Los aportes de Google

- Stack de software de Google
  - Google File System (GFS)
    - Sistema de archivos para cluster que permite acceder a todos los discos del datacenter de Google como un sistema de archivos masivo, distribuido y redundante
  - Map Reduce
    - Framework de procesamiento distribuido para paralelizar algoritmos en una gran cantidad de servidores, potencialmente no confiables, y ser capaz de manejar datasets masivos

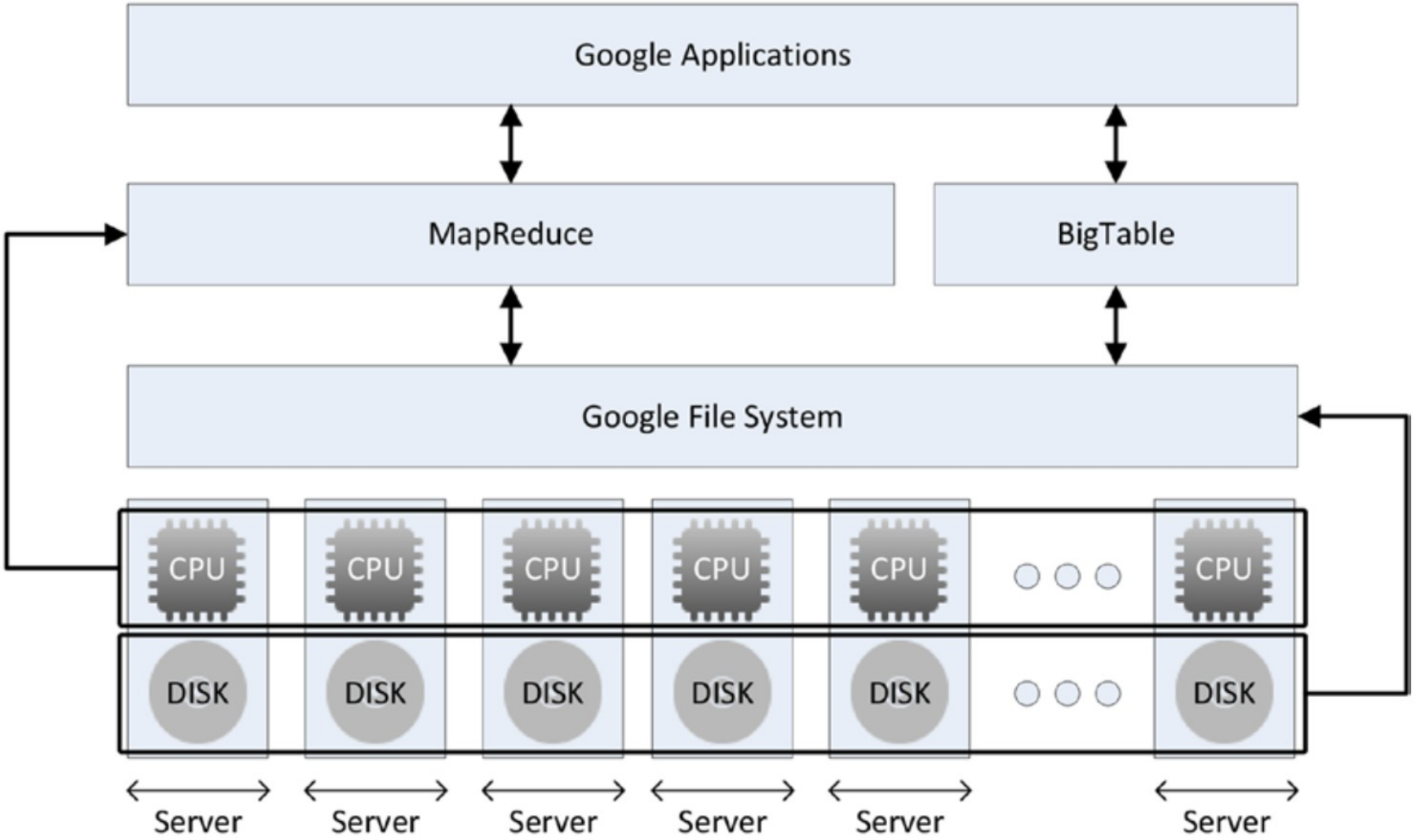
# Los aportes de Google (II)

- Stack de software de Google
  - BigTable
    - Un sistema de base de datos no relacional que utiliza el sistema de archivos de Google (GFS) para el almacenamiento
    - Wide-column store

# Los aportes de Google (II)

- Stack de software de Google
  - BigTable
    - Un sistema de base de datos no relacional que utiliza el sistema de archivos de Google (GFS) para el almacenamiento
    - Wide-column store

# Los aportes de Google (III)



# MapReduce

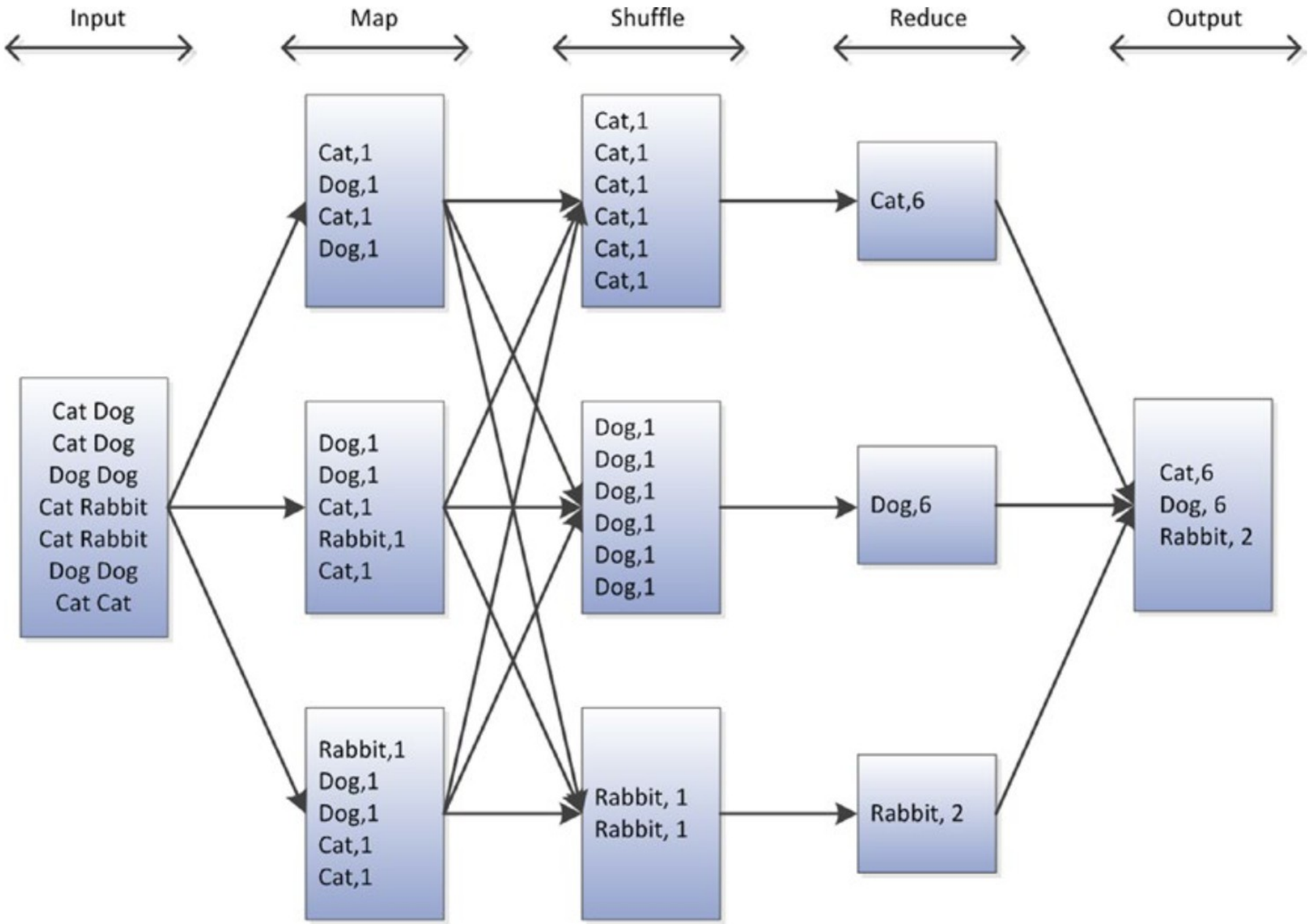
- Paradigma de programación paralela
- Entorno de ejecución distribuido.
- El modelo básico tiene dos fases:
  - Fase **map**: generar parejas (clave,valor) a partir de la entrada
  - Fase **reduce**: agrupar las parejas a partir del valor de la clave y producir una salida

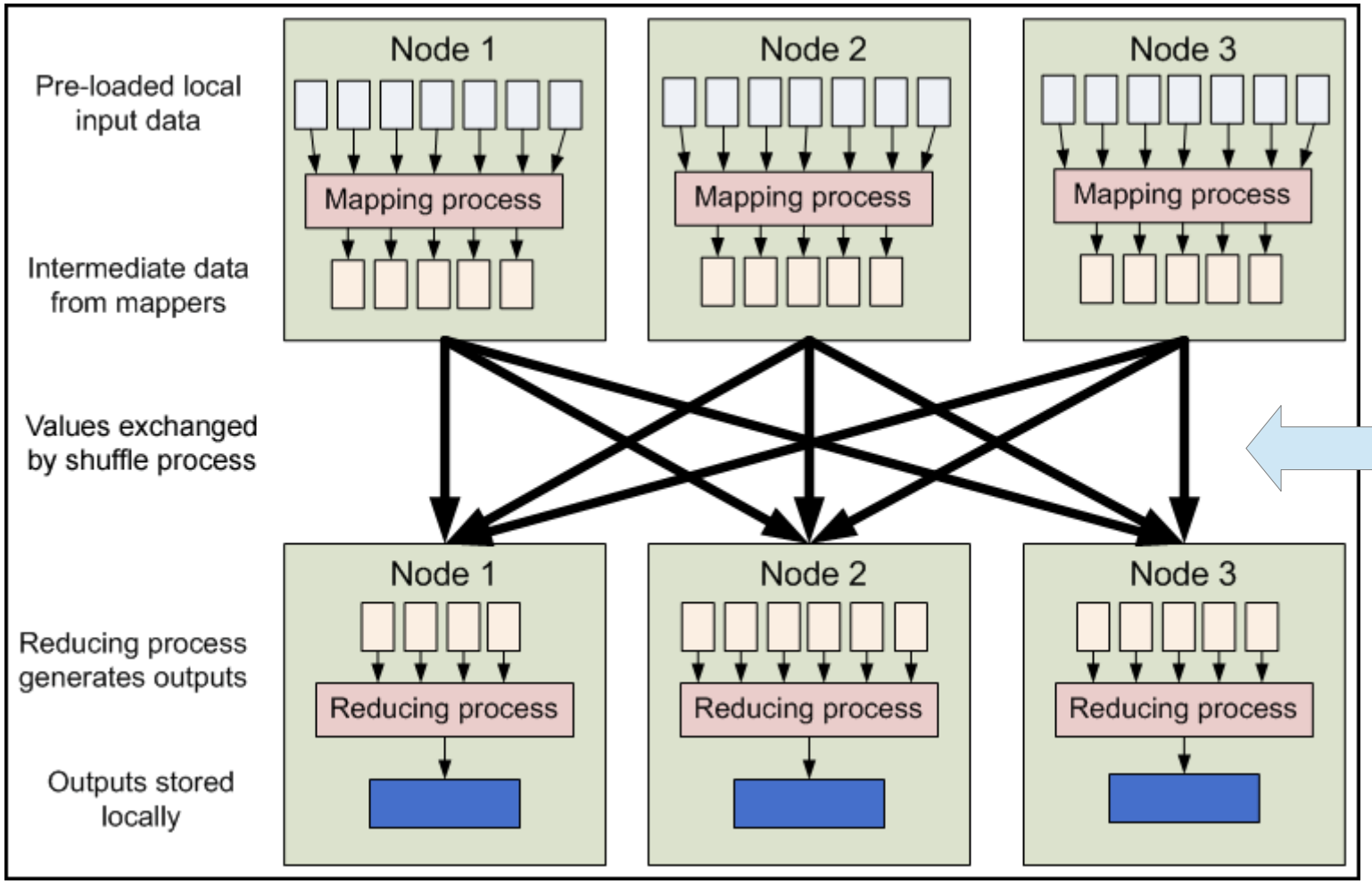
Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." OSDI (2004)

# MapReduce (ii)

- En el modelo básico se deben programar dos funciones:
  - map: $(k_1,v_1) \rightarrow [(k_2,v_2)]$
  - reduce: $(k_2,[v_2]) \rightarrow [(k_3,v_3)]$

- Vamos a aplicarlo a un problema simple: conteo de palabras


Input
Map
Shuffle
Reduce
Output
Cat Dog
Cat Dog
Dog Dog
Cat Rabbit
Cat Rabbit
Dog Dog
Cat Cat
Cat,1
Dog,1
Cat,1
Dog,1
Dog,1
Dog,1
Cat,1
Rabbit,1
Cat,1
Rabbit,1
Dog,1
Dog,1
Cat,1
Cat,1
Cat,1
Cat,1
Cat,1
Cat,1
Cat,1
Cat,1
Dog,1
Dog,1
Dog,1
Dog,1
Dog,1
Dog,1
Rabbit, 1
Rabbit, 1
Cat,6
Dog,6
Rabbit, 2
Cat,6
Dog, 6
Rabbit, 2

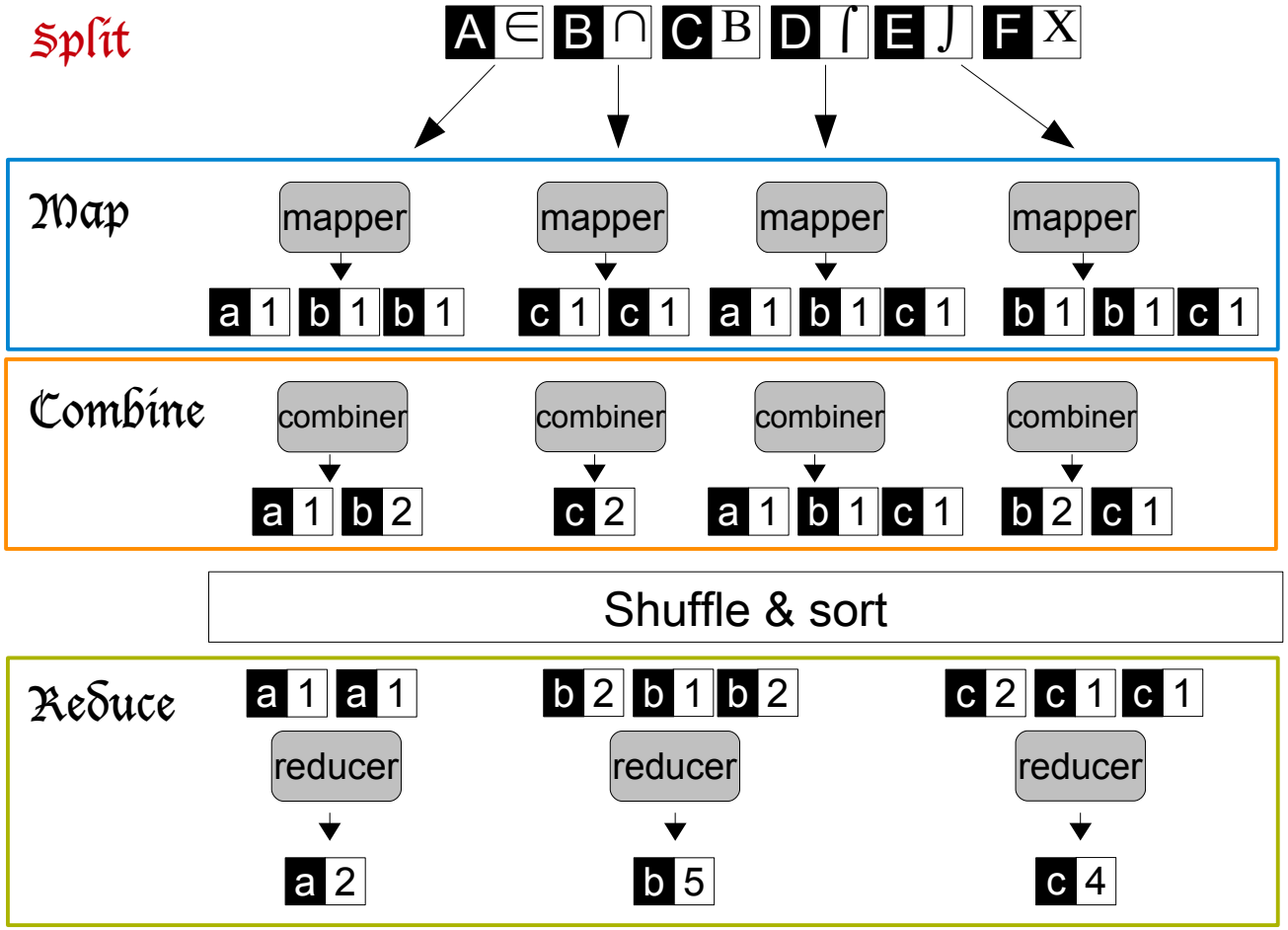



Reducir todo lo posible la cantidad de datos que pasan a la fase de *reduce*

"Hadoop Tutorial from Yahoo!" by Yahoo! Inc.

# Conteo de palabras : *combiners*

- Permiten computar resultados parciales a la salida de cada *mapper* (mini-reducers)



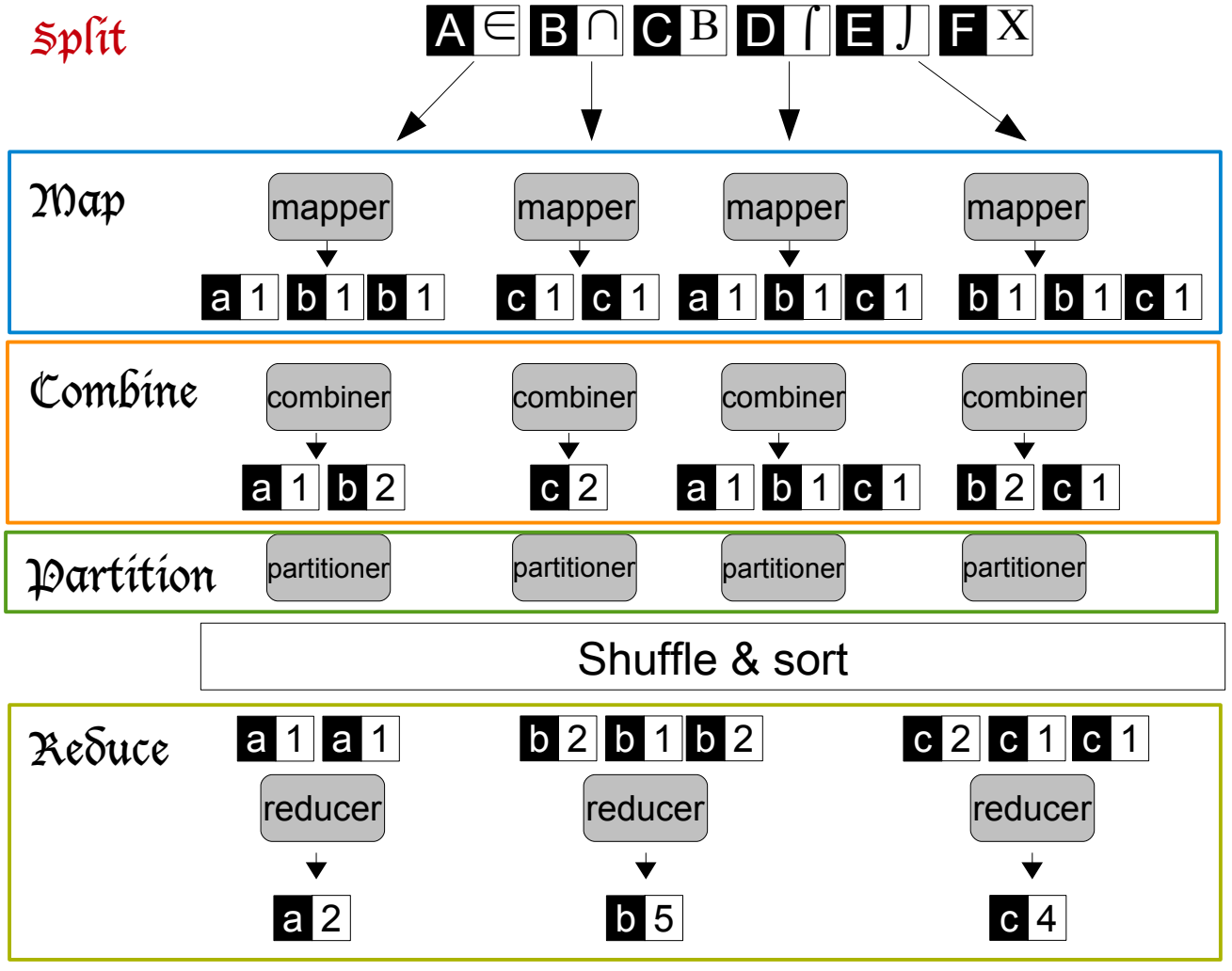

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

# Conteo de palabras : *partitioners*

- Permiten determinar cómo se distribuyen las parejas en los diferentes *reducers*



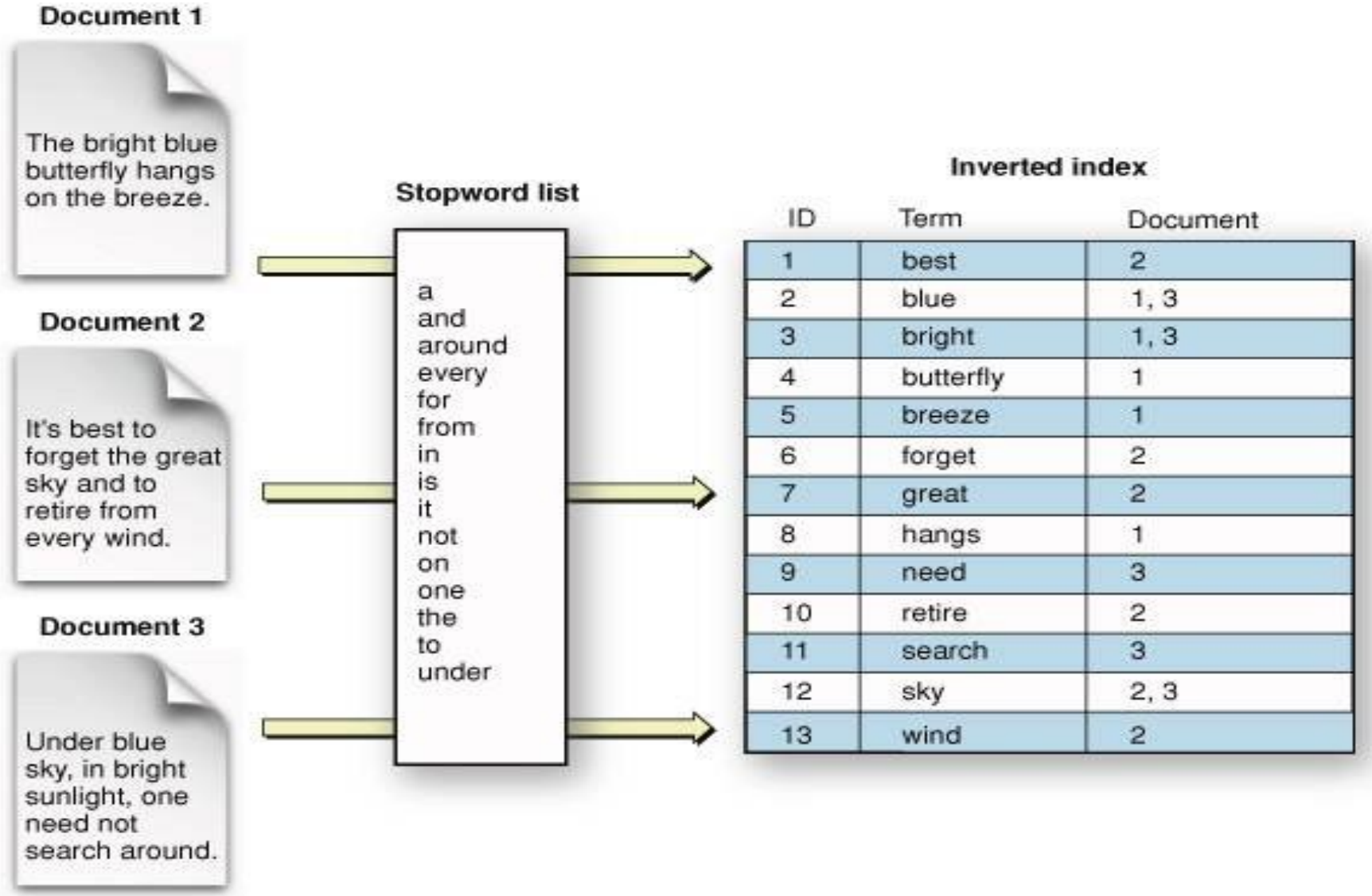

Un método de particionado sencillo consiste en computar hash(k) MOD #reducers.

# Patrones en MapReduce

- No puedo resolver cualquier problema con esta técnica
- Para algunos problemas hay patrones de diseño definidos:
  - Sumarización e indexado
  - Filtrado
  - Organización de datos (particionado, transformaciones)
  - Join

Donald Miner and Adam Shook. MapReduce Design Patterns Building Effective Algorithms and Analytics for Hadoop and Other Systems. O'Reilly Media,Inc., 2012.

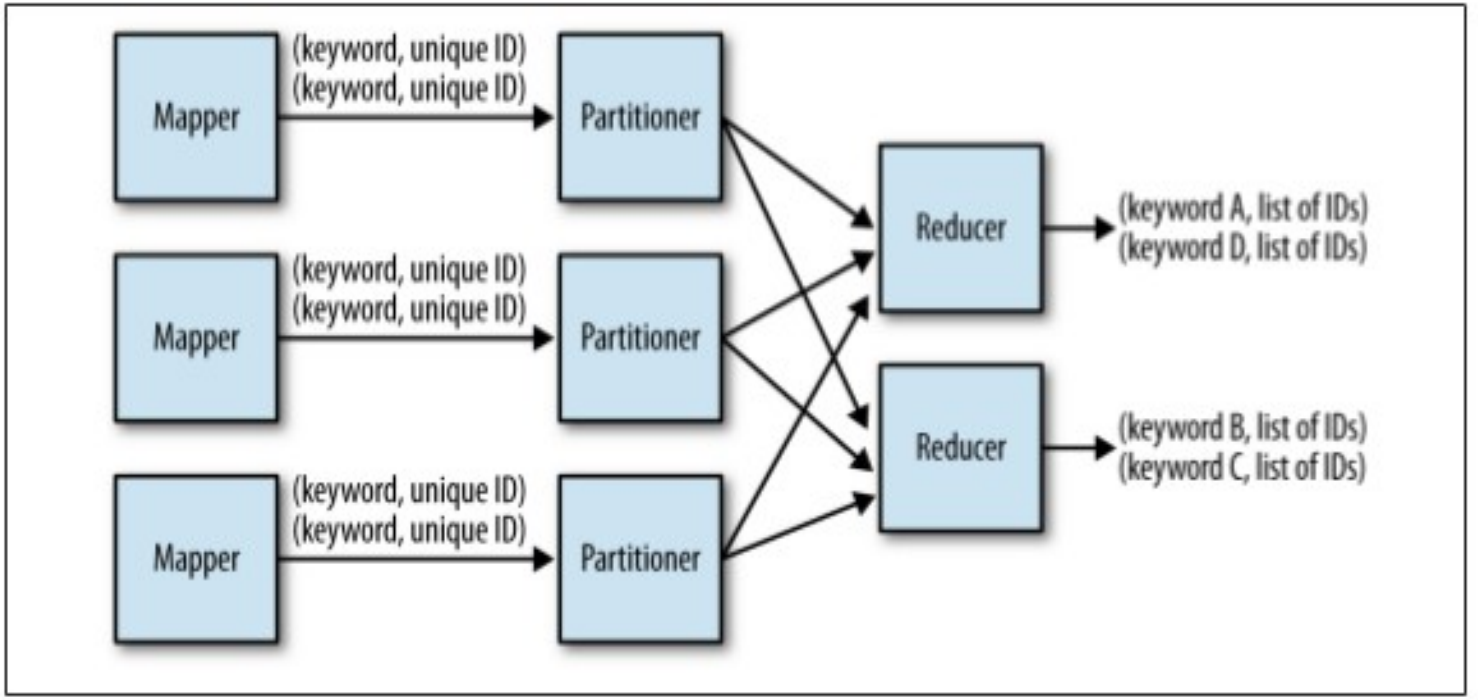
# Indices inversos




| ID | Term | Document |
|---|---|---|
| 1 | best | 2 |
| 2 | blue | 1, 3 |
| 3 | bright | 1, 3 |
| 4 | butterfly | 1 |
| 5 | breeze | 1 |
| 6 | forget | 2 |
| 7 | great | 2 |
| 8 | hangs | 1 |
| 9 | need | 3 |
| 10 | retire | 2 |
| 11 | search | 3 |
| 12 | sky | 2, 3 |
| 13 | wind | 2 |

# Indices inversos

- Fue el caso de uso que dio origen a MapReduce en Google
- Construir una lista de URLs que contienen cierta palabra



# Indices inversos

```
map(key, container) {
  for each element in container {
      element_meta =
           extract_metadata(element, container)
      emit(element, [container_id, element_meta])
  }
}

reduce(element, container_ids) {
  element_stat =
       compute_stat(container_ids)
  emit(element, [element_stat, container_ids])
}
```

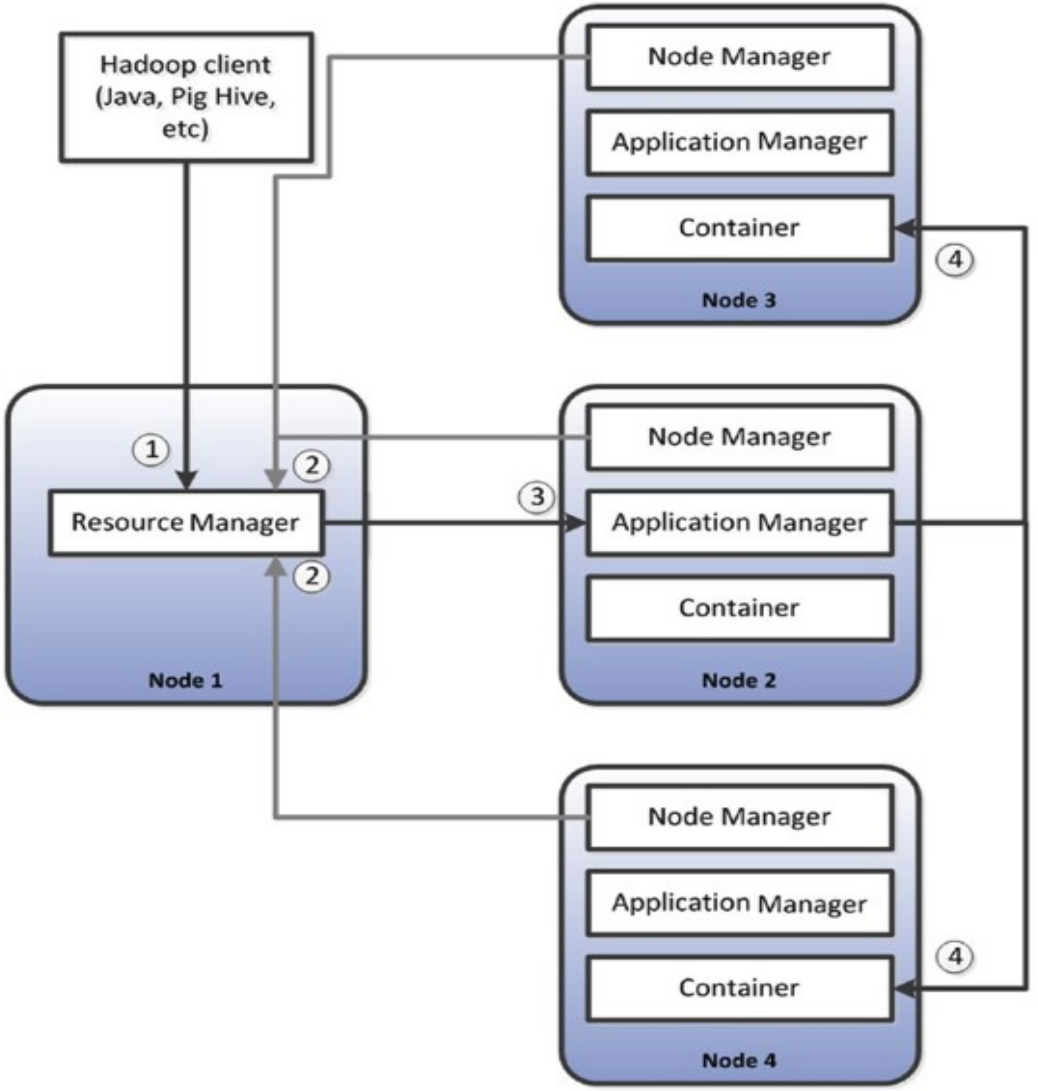
# Hadoop: Open-Source Google Stack

# Hadoop: Open-Source Google Stack

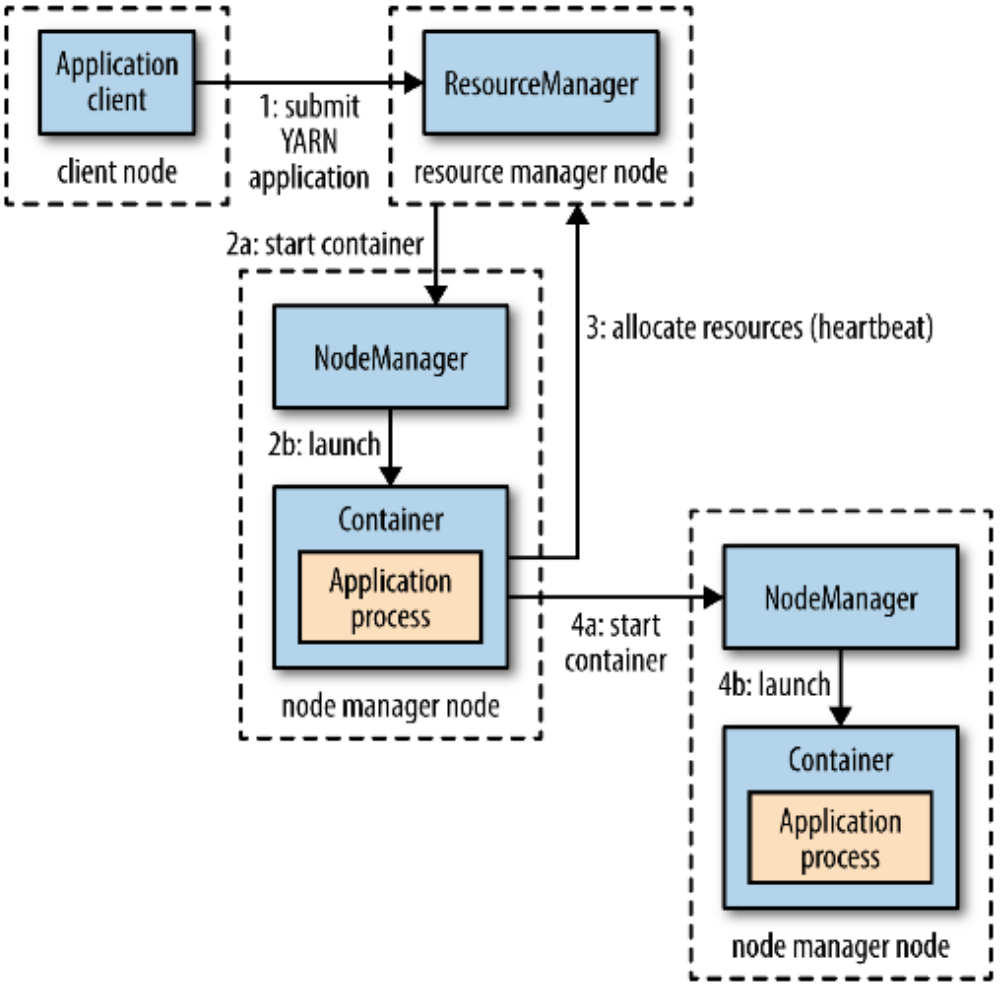
# Hadoop

- Implementa Google Stack con licenciamiento Open Source
  - Comenzó como un proyecto de un buscador abierto: Nutch
  - Creció rápidamente, primeras implementaciones
    - Yahoo! (2006)
      - 5 petabytes de storage, 10K cores
    - Facebook (2007)
      - 100 petabytes

# Arquitectura Hadoop (2.0)



# Ejecutando un programa en Hadoop 2.0




Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 4th Edition 2015.

# HBase

- En 2006 Google publica paper describiendo BigTable, sirviendo como los fundamentos de desarrollo de una base noSQL: Hbase
- Usa Hadoop HDFS como filesystem
- Tiene la capacidad de crear tablas de gran tamaño, superando los tamaños máximos de RBDMS tradicionales (MySQL, Oracle)
- HDFS provee de redundancia automática para HBase

# HBase

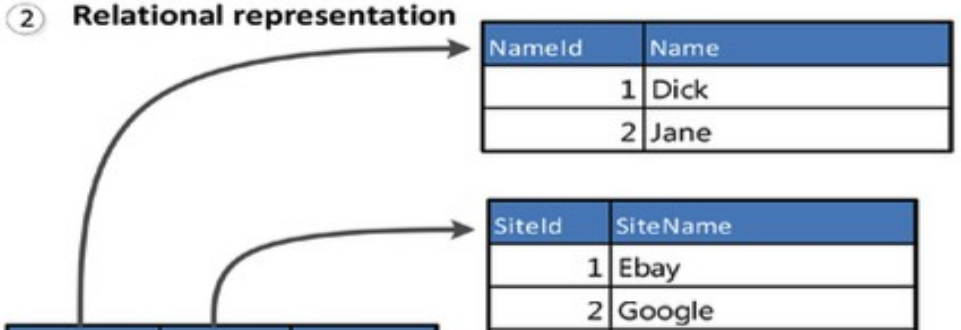
- HDFS permite almacenar archivos con cualquier estructura en Hadoop
  - Hbase obliga estructura
    - Columnas, filas, tablas, claves
    - Muy diferente al modelo relacional
- En cada celda, pueden haber varias versiones de un valor. Cada versión del valor se identifica con un timestamp. Esto le permite tener una “tercera dimensión”

## ① Raw Data

| Name | Site | Visits |
|---|---|---|
| Dick | Ebay | 507,018 |
| Dick | Google | 690,414 |
| Jane | Google | 716,426 |
| Dick | Facebook | 723,649 |
| Jane | Facebook | 643,261 |
| Jane | ILoveLarry.com | 856,767 |
| Dick | MadBillFans.com | 675,230 |

---

## ② Relational representation



| NameId | Name |
|---|---|
| 1 | Dick |
| 2 | Jane |

| SiteId | SiteName |
|---|---|
| 1 | Ebay |
| 2 | Google |
| 3 | Facebook |
| 4 | ILoveLarry.com |
| 5 | MadBillFans.com |

| NameId | SiteId | Visits |
|---|---|---|
| 1 | 1 | 507,018 |
| 1 | 3 | 690,414 |
| 2 | 3 | 716,426 |
| 1 | 3 | 723,649 |
| 2 | 3 | 643,261 |
| 2 | 4 | 856,767 |
| 1 | 5 | 675,230 |

---

## ③ HBase version

| Id | Name | Ebay | Google | Facebook | (other columns) | MadBillFans.com |
|---|---|---|---|---|---|---|
| 1 | Dick | 507,018 | 690,414 | 723,649 | ..... | 675,230 |

| Id | Name | Google | Facebook | (other columns) | ILoveLarry.com |
|---|---|---|---|---|---|
| 2 | Jane | 716,426 | 643,261 | ..... | 856,767 |

# HBase – Fundamentos de diseño

- ¿Cómo será la estructura de filas y que datos contendrán?
- ¿Cuántas familias de columnas tendremos?
- ¿Qué datos iran en cada familia de columnas?
- ¿Cuántas columnas hay en cada familia de columnas?
- ¿Cuáles serán los nombres de las columnas? No es necesario definirlo en la creación de la tabla, se utililizará en la lectura / escritura

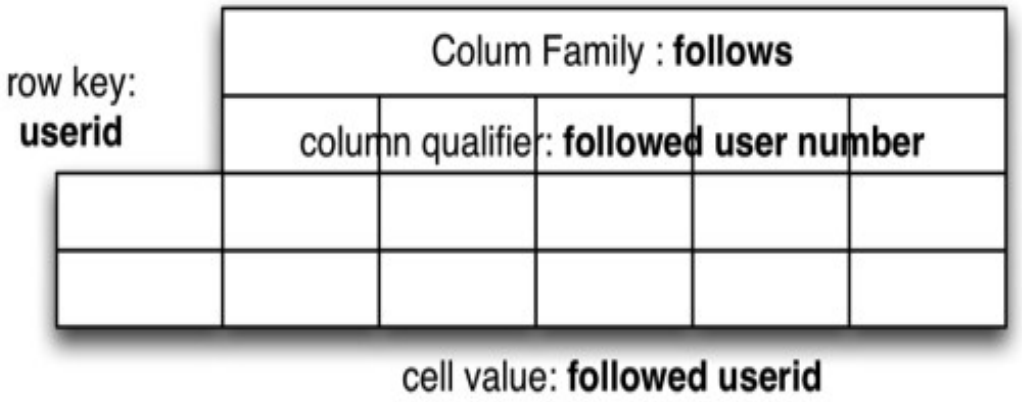
# HBase – Fundamentos de diseño

- ¿Cuáles son los datos que almacenaremos a nivel de celda?
- ¿Cuántas versiones del dato tendremos en cada celda?

# HBase – Fundamentos de diseño

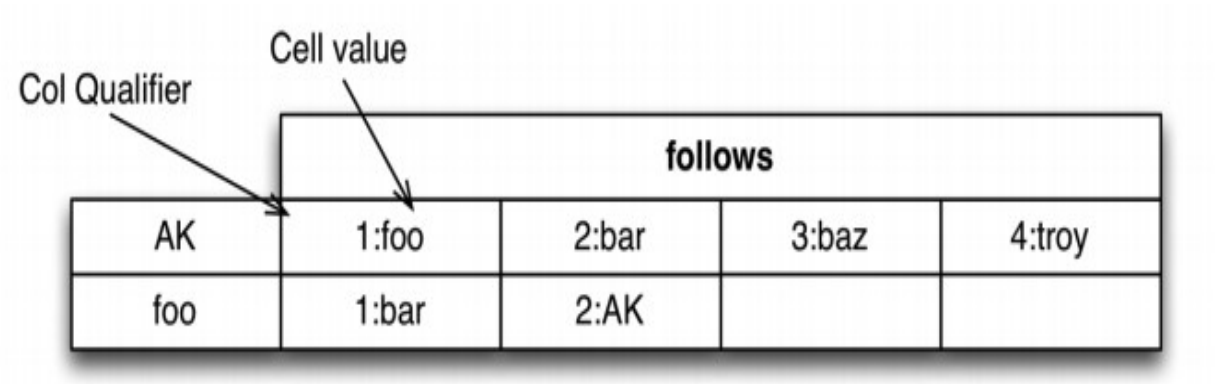
- Ejemplo
  - Relaciones en Twitter
    - Ejemplo caso para bases de Grafos...
  - Definimos los patrones de acceso de la aplicación
    - Patrón de lectura
      - ¿A quienes sigue un usuario?
      - ¿El usuario particular A sigue al usuario B?
      - ¿Quién sigue a un usuario particular A?
    - Patrón de escritura
      - Un usuario sigue a un nuevo usuario
      - Un usuario deja de seguir a un usuario

# Hbase – Fundamentos de diseño




- Esta tabla almacena, en una sola fila, una lista de usuarios seguidos por un usuario en particular.
- La clave de la fila es el ID del usuario seguidor
- Cada columna contiene el ID del usuario que se está siguiendo.

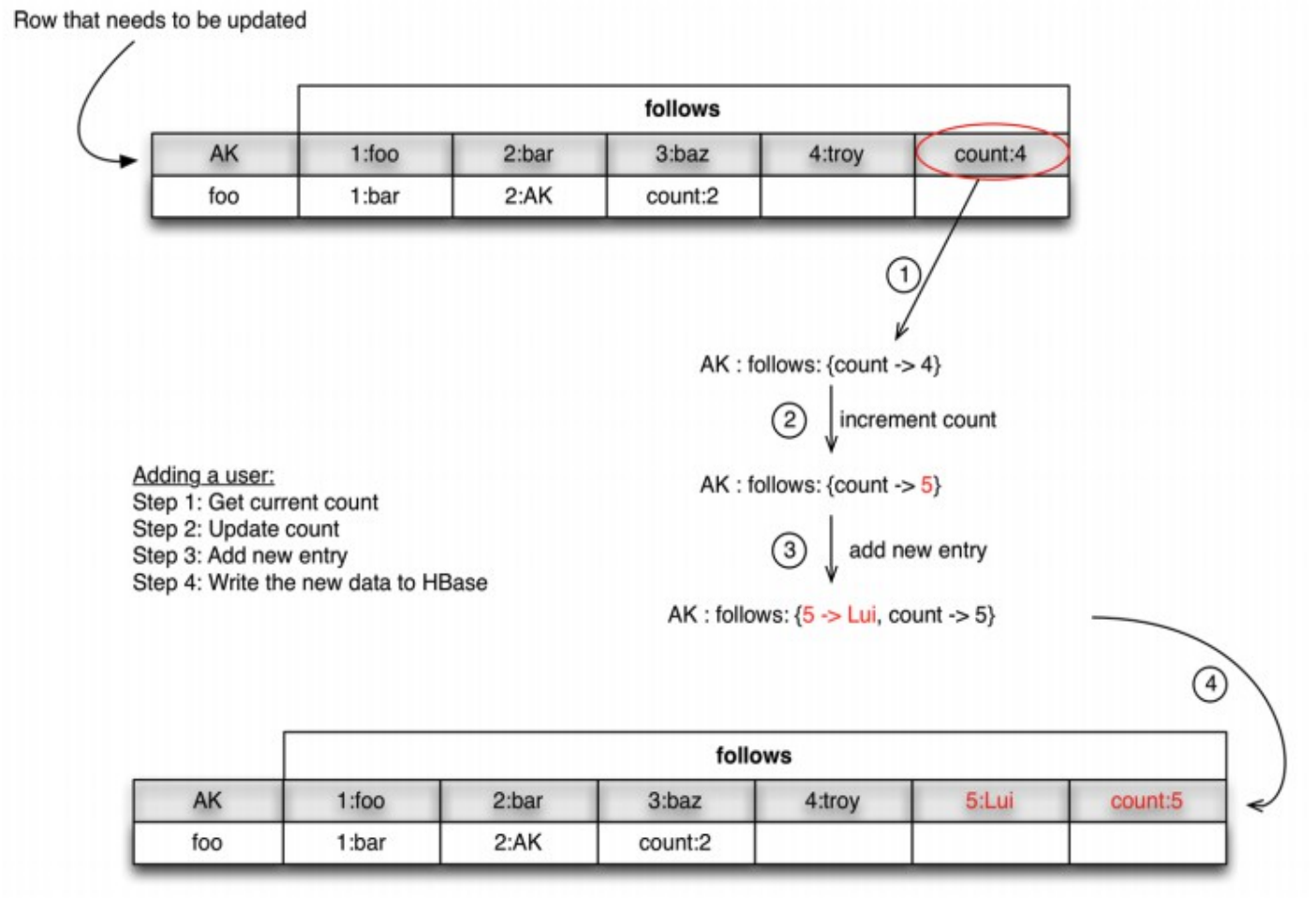
# Hbase – Fundamentos de diseño




¿Quienes siguen un usuario en particular?

- Es costoso, requiere iterar toda la tabla
- Además, agregar un nuevo usuario también es costoso. Requiere leer toda la fila, para determinar cual es el último, de forma de darle nombre al nuevo.

# Hbase – Fundamentos de diseño

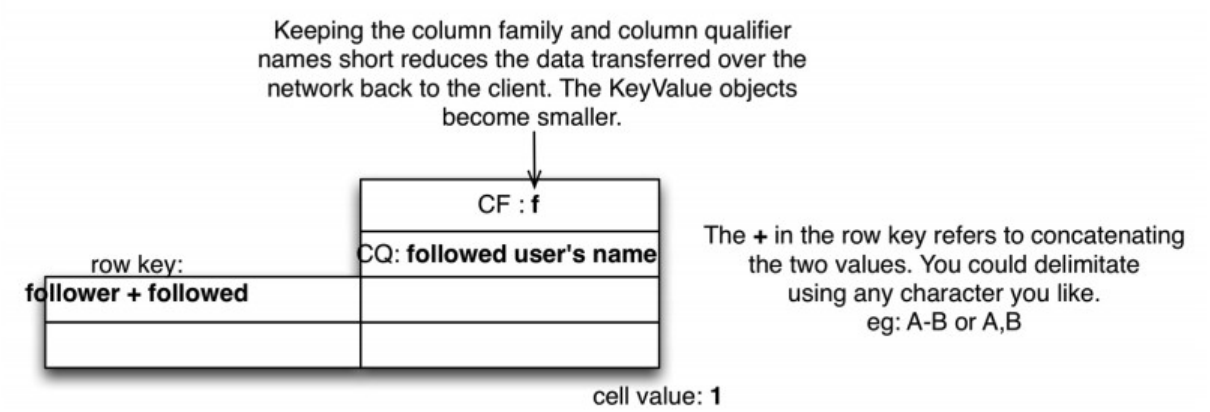


# Hbase – Fundamentos de diseño

| | follows | | | |
|---|---|---|---|---|
| AK | foo:1 | bar:1 | baz:1 | troy:1 |
| foo | bar:1 | AK:1 | | |

- Los calificadores de las columnas son dinámicos y se almacenan como byte[]
- Puedo invertirlos y ya no requiero tener un contador
- Esto simplifica agregar y eliminar usuarios
- Seguimos con el problema de determinar quienes siguen a un usuario en particular
  - Dos posibles soluciones
    - Otra tabla con datos invertidos
    - Usar otro indice
      - Todos los datos de las celdas son arrays de bytes, no importa lo que se almacene

# Hbase – Fundamentos de diseño

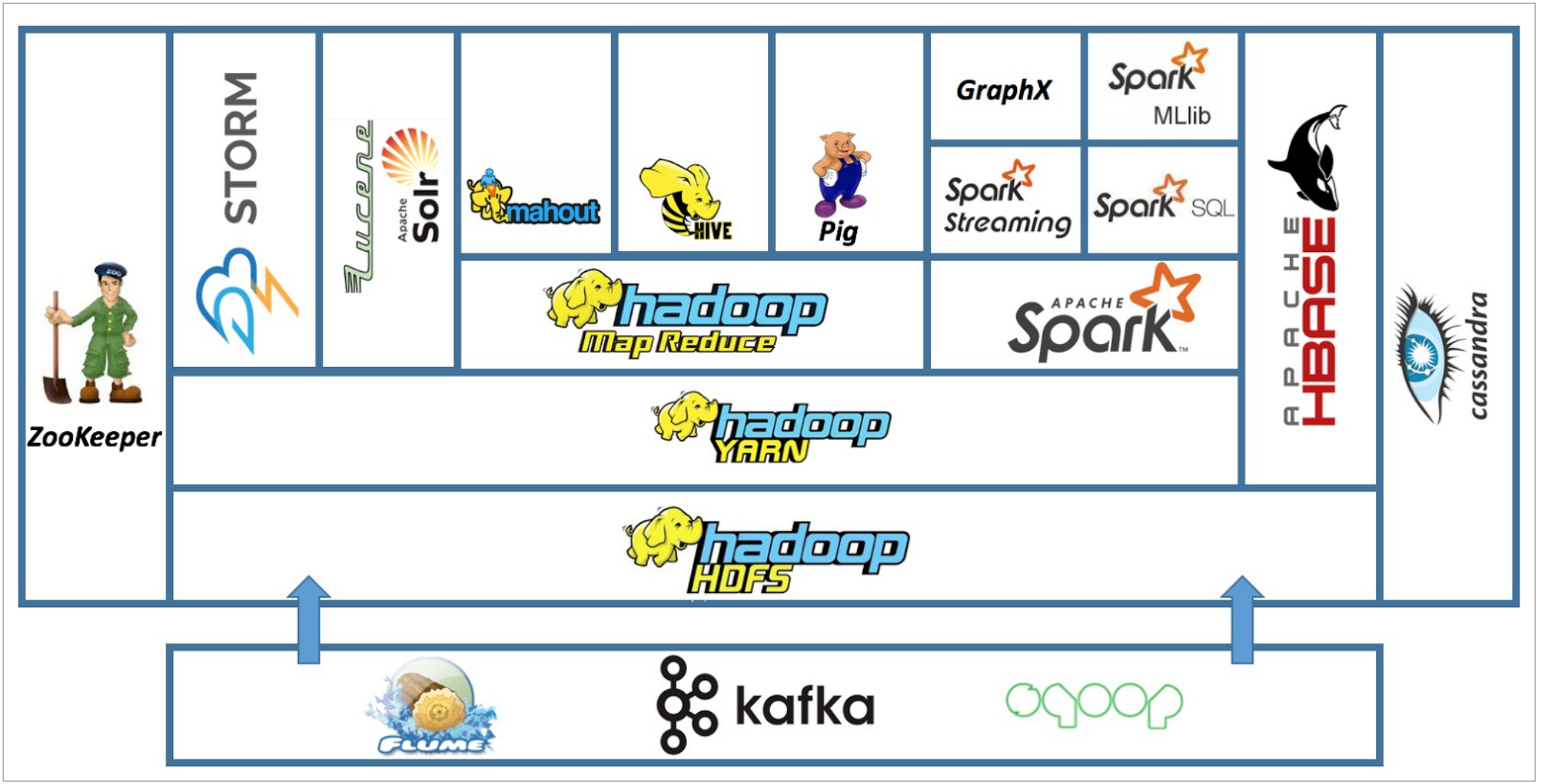



| | f |
|---|---|
| AK+foo | James Foo:1 |
| AK+bar | Jimmy Bar:1 |
| AK+baz | Ricky Baz:1 |
| AK+troy | Troy:1 |
| foo+bar | Jimmy Bar:1 |
| foo+AK | AK:1 |

- La clave ahora tiene seguidor y usuario seguido
- El nombre de la familia de la columna se redujo a f (simplemente reduce carga de E/S, tanto en disco como en red)

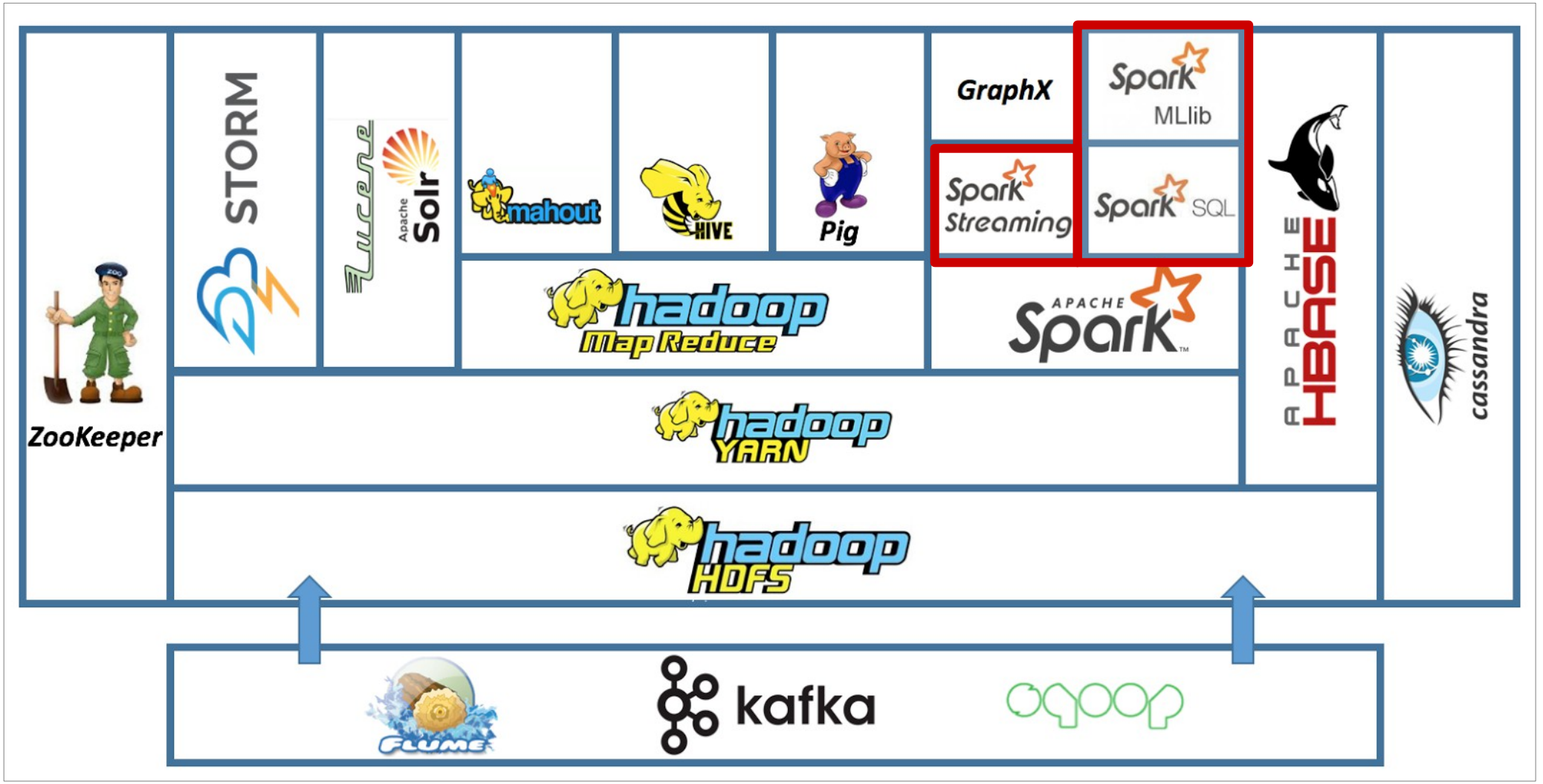
# HBase

- Diseñar y programar sobre Hadoop es complejo
- Aparecieron abstracciones
- **Pig** es un lenguaje de alto nivel que permite realizar consultas
  - B = ORDER A BY col4 DESC;
  - C = LIMIT B 10;
- **Hive** provee una abstracción sobre Hadoop.
  - Modelo de datos: tablas, vistas, etc
  - HiveQL como lenguaje de consultas


ZooKeeper
STORM
Lucene
Apache Solr
mahout
HIVE
Pig
GraphX
Spark MLlib
Spark Streaming
Spark SQL
hadoop Map Reduce
APACHE Spark
APACHE HBASE
cassandra
hadoop YARN
hadoop HDFS
FLUME
kafka
sqoop

# Hadoop: resumen

- HDFS como base
- SQL como punto de acceso
- ¡Mercados de Data warehouse y Hadoop ahora son la misma cosa!
- Marketing asociado a Hadoop: Data Lake


ZooKeeper
STORM
Lucene
Apache Solr
mahout
HIVE
Pig
GraphX
Spark MLlib
Spark Streaming
Spark SQL
hadoop Map Reduce
APACHE Spark
APACHE HBASE
cassandra
hadoop YARN
hadoop HDFS
FLUME
kafka
oozie

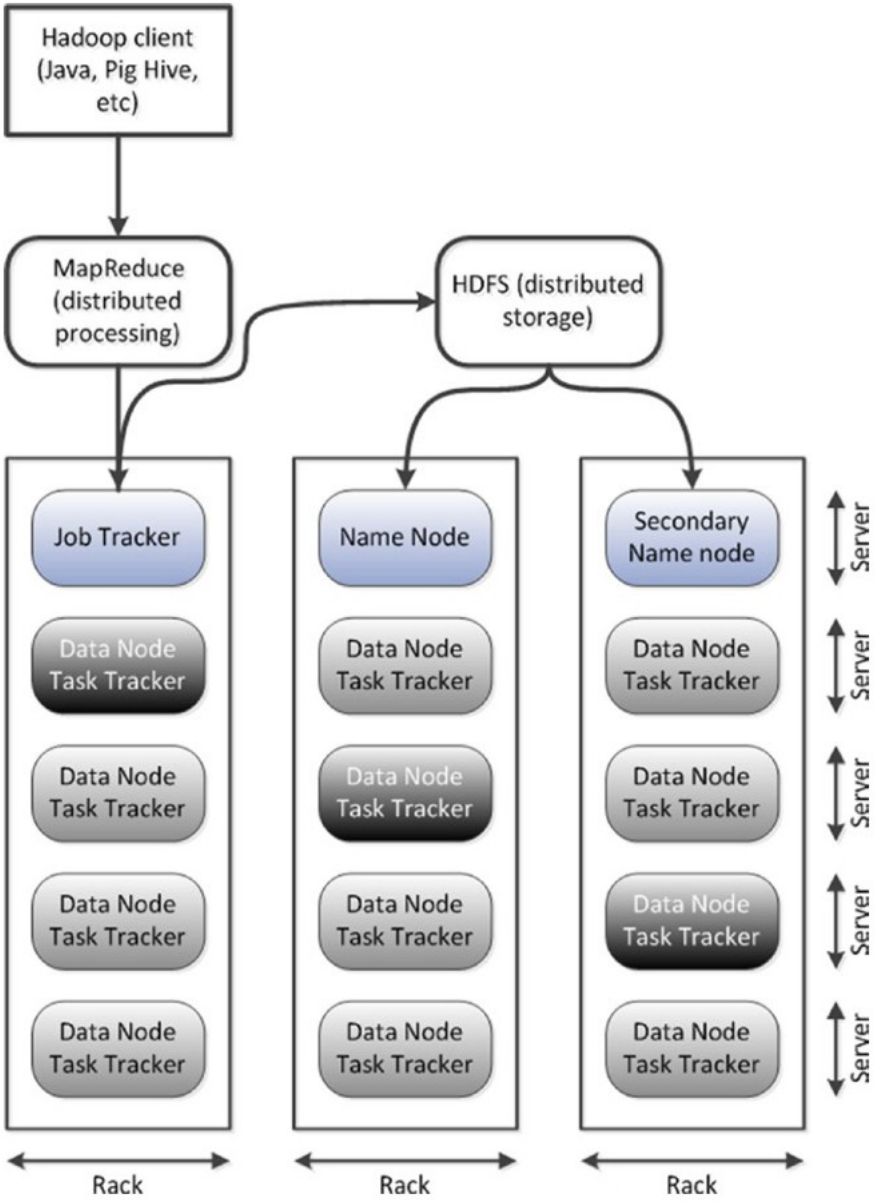
# Spark

- Originalmente concevido como un Map-Reduce más funcional y rápido
- Base de datos en memoria, distribuida, tolera fallas
- De acuerdo a Matei Zaharia

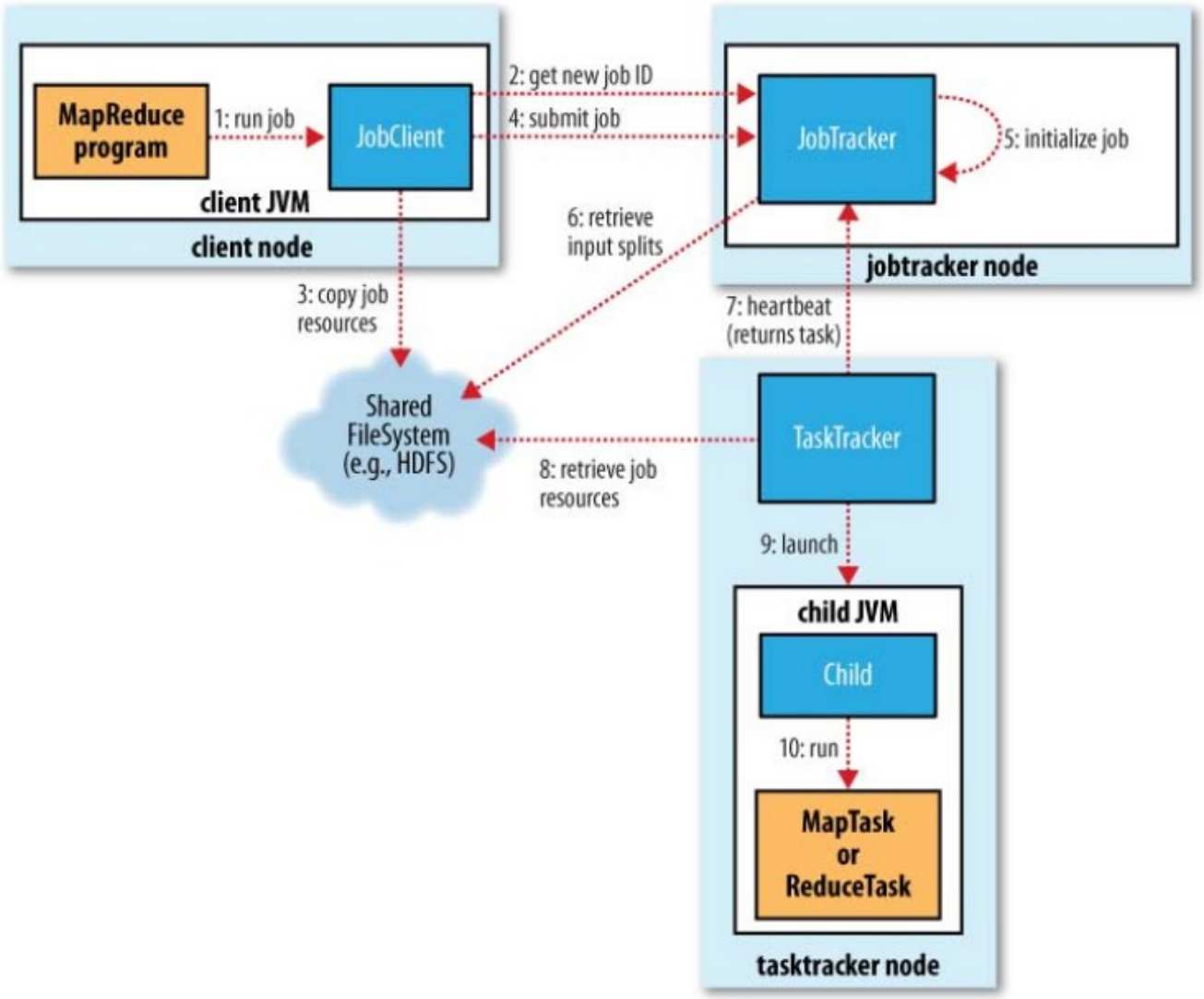
# Hadoop como entorno resuelve:

- *Scheduling* de tareas
- "Mueve" el código a los datos (comenzar la tarea en el nodo que tiene los datos necesarios)
- Sincronización entre procesos
- Errores y manejo de fallas

# Arquitectura: Hadoop 1.0

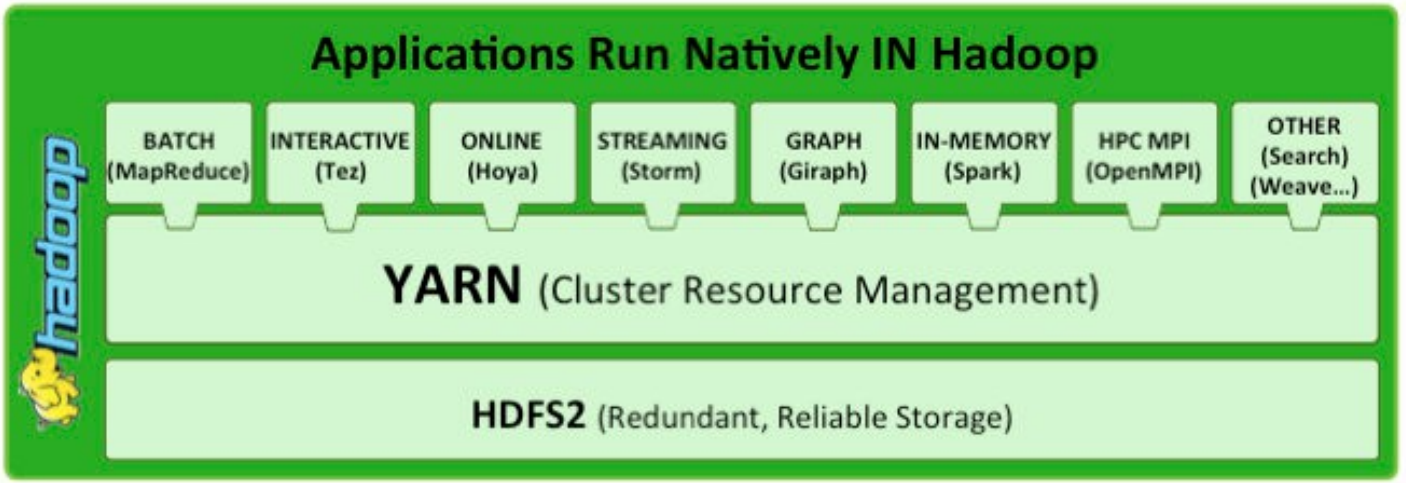


# Ejecutando un programa MapReduce en Hadoop 1.0




Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012.

# Arquitectura: Hadoop 2.0




- Aparece YARN (Yet Another Resource Negotiator)
- El JobTracker y TaskTracker son reemplazados por 3 componentes:
  - **ResourceManager:** controla el acceso a todos los recursos
  - **NodeManager:** corre en cada nodo y maneja sus recursos. Responde al RM
  - **ApplicationManager:** controla la ejecución de la tarea.

# Ejecutando un programa en Hadoop 2.0

Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 4th Edition 2015.

# Conteo de palabras en Java

- En el método principal se establece, entre otros:
  - Quién implementa map, reduce y combine
  - Formatos de entrada y salida

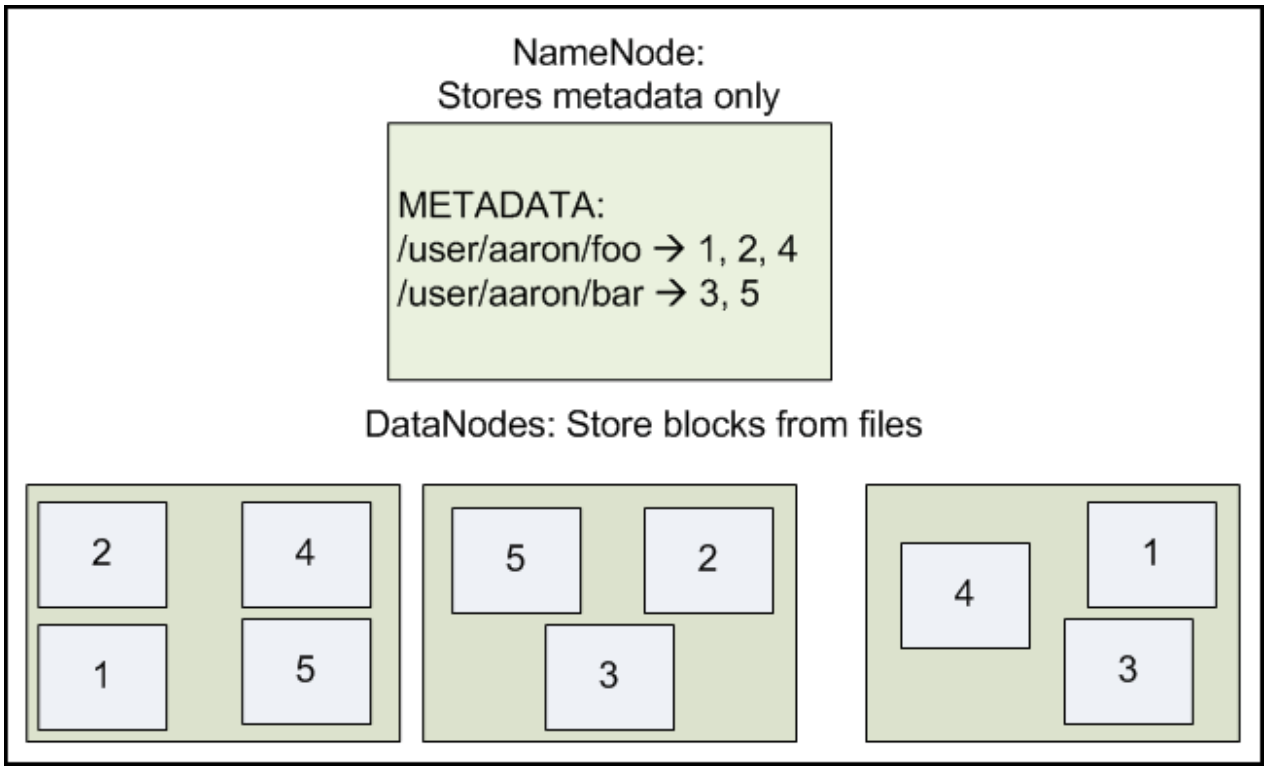
```
public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    ...
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
{
...
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

     Job job = new Job(conf, "wordcount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }
}
```

# Conteo de palabras en Java (ii)

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
     StringTokenizer tokenizer = new StringTokenizer(line);
     while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         context.write(word, one);
     }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
     for (IntWritable val : values) {
         sum += val.get();
     }
     context.write(key, new IntWritable(sum));
    }
 }
```

# Hadoop Distributed Filesystem (HDFS)

- Sistema de archivos distribuído
- Organiza los datos en *bloques* (64 MB)
- Dos tipos de nodos:
  - *namenode*
  - *datanodes*




"Hadoop Tutorial from Yahoo!" by Yahoo! Inc.

# Pero programar sobre Hadoop no es sencillo :(

- Aparecen abstracciones
- Pig es un lenguaje de alto nivel que permite realizar consultas
  - B = ORDER A BY col4 DESC;
  - C = LIMIT B 10;
- Hive provee una abstracción sobre Hadoop.
  - Modelo de datos: tablas, vistas, etc
  - HiveQL como lenguaje de consultas