Procesamiento de grandes volúmenes de datos utilizando Map-Reduce

Actualmente Map-Reduce está relacionado con los sistemas de bases de datos No-SQL, sistemas DBMS No-SQL tales como Mongo DB incluyen comandos para ejecutar operaciones Map-Reduce sobre los objetos almacenados en la BD.

Map-Reduce está siendo muy utilizado por empresas que requieren procesar muchos datos de manera off-line. Por ejemplo operaciones relacionadas con la aplicación de técnicas de minería de datos sobre colecciones de datos obtenidos desde distintos sistemas operando en producción en la empresa (e.g., clicks de usuarios de sitios Web de tiendas de retail).

Map-Reduce

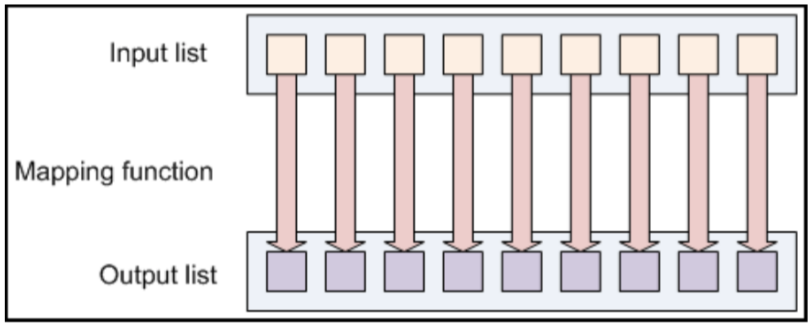
- Modelo de programación para procesar grandes cantidades de datos en paralelo distribuyendo la carga de trabajo sobre muchas máquinas

- Inspirado en el uso de dos funciones: **map** y **reduce**, usadas en la programación funcional

- El procesamiento utiliza como entrada un conjunto pares **key/value** que representa cada subconjunto de los datos y produce como salida otro conjunto **key/value**

Map

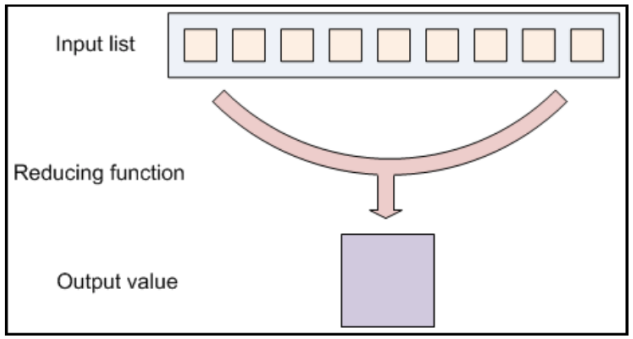
- Esta función toma los pares de entrada key/value y genera un par key/value intermedio.
- Luego, todos los pares intermedios son agrupados mediante la key intermedia y son enviados a la función **reduce**



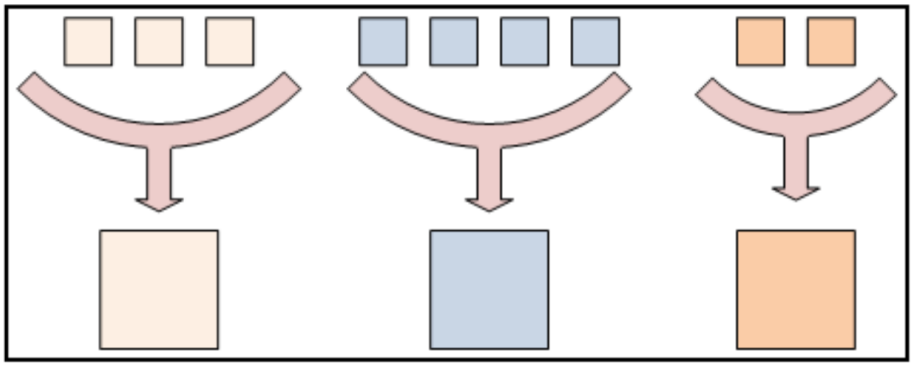
developer.yahoo.com/hadoop/tutorial/

Reduce

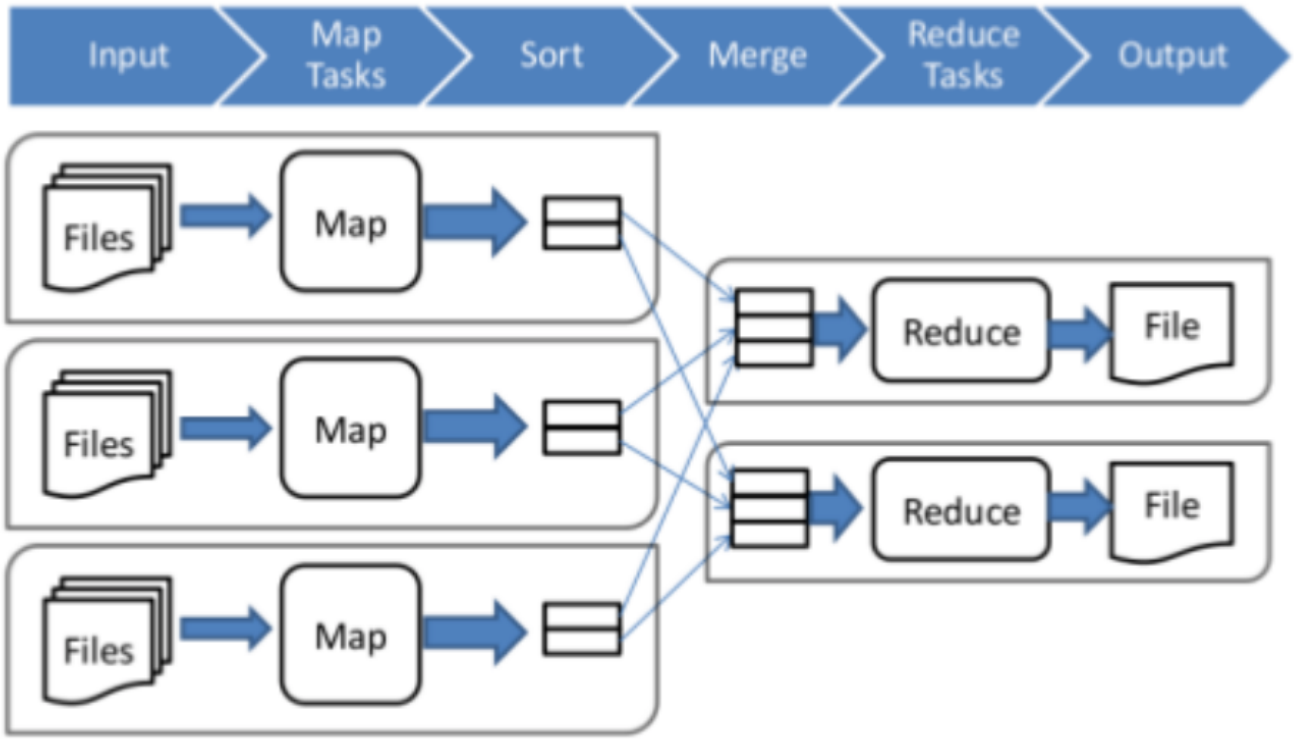
- Esta función acepta una **key** intermedia y un conjunto de valores para esa key.
- Esta mezcla dichos valores para formar valores más pequeños.
- Esta función generalmente genera una salida por cada invocación



oper.yahoo.com/hadoop/tutorial/



Different colors represent different keys. All values with the same key are presented to a single reduce task.


Input
Map Tasks
Sort
Merge
Reduce Tasks
Output
Files
Map
Files
Map
Files
Map
Reduce
File
Reduce
File

AN EXAMPLE APPLICATION: WORD COUNT

A simple MapReduce program can be written to determine how many times different words appear in a set of files. For example, if we had the files:

foo.txt: Sweet, this is the foo file

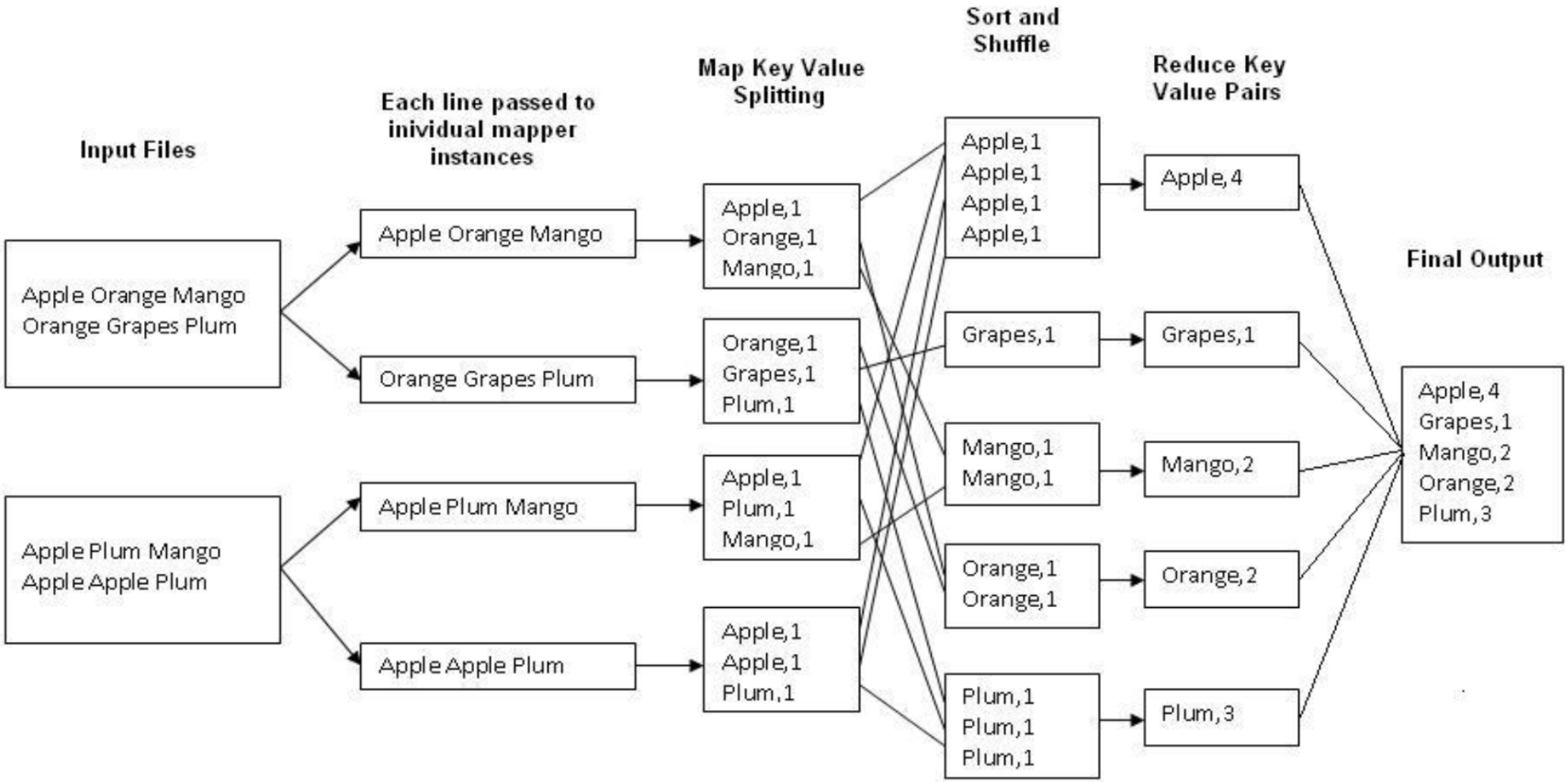
bar.txt: This is the bar file

We would expect the output to be:

```
sweet 1
this  2
is    2
the   2
foo   1
bar   1
file  2
```

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```


Input Files
Each line passed to inividual mapper instances
Map Key Value Splitting
Sort and Shuffle
Reduce Key Value Pairs
Final Output
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple,1
Orange,1
Mango,1
Orange,1
Grapes,1
Plum,1
Apple,1
Plum,1
Mango,1
Apple,1
Apple,1
Plum,1
Apple,1
Apple,1
Apple,1
Apple,1
Grapes,1
Mango,1
Mango,1
Orange,1
Orange,1
Plum,1
Plum,1
Plum,1
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3

MapReduce Tutorial
By: Steve Krenzel
http://stevekrenzel.com/finding-friends-with-mapreduce

Assume the friends are stored as Person->[List of Friends], our friends list is then:

A -> B C D
B -> A C D E
C -> A B D E
D -> A B C E
E -> B C D

(A B) ->
(A C) ->
(A D) ->
(B C) ->
(B D) ->
(B E) ->
(C D) ->
(C E) ->
(D E) ->

Se requiere conocer los amigos en común entre todos los pares de usuarios A, B, C y D.

For map(A -> B C D) :

(A B) -> B C D
(A C) -> B C D
(A D) -> B C D

For map(B -> A C D E) :

(A B) -> A C D E
(B C) -> A C D E
(B D) -> A C D E
(B E) -> A C D E

(Note that A comes before B in the key)

For map(C -> A B D E) :

(A C) -> A B D E
(B C) -> A B D E
(C D) -> A B D E
(C E) -> A B D E

For map(D -> A B C E) :

(A D) -> A B C E
(B D) -> A B C E
(C D) -> A B C E
(D E) -> A B C E

And finally for map(E -> B C D):

(B E) -> B C D
(C E) -> B C D
(D E) -> B C D

Before we send these key-value pairs to the reducers, we group them by their keys and get:

(A B) -> (A C D E) (B C D)
(A C) -> (A B D E) (B C D)
(A D) -> (A B C E) (B C D)
(B C) -> (A B D E) (A C D E)
(B D) -> (A B C E) (A C D E)
(B E) -> (A C D E) (B C D)
(C D) -> (A B C E) (A B D E)
(C E) -> (A B D E) (B C D)
(D E) -> (A B C E) (B C D)

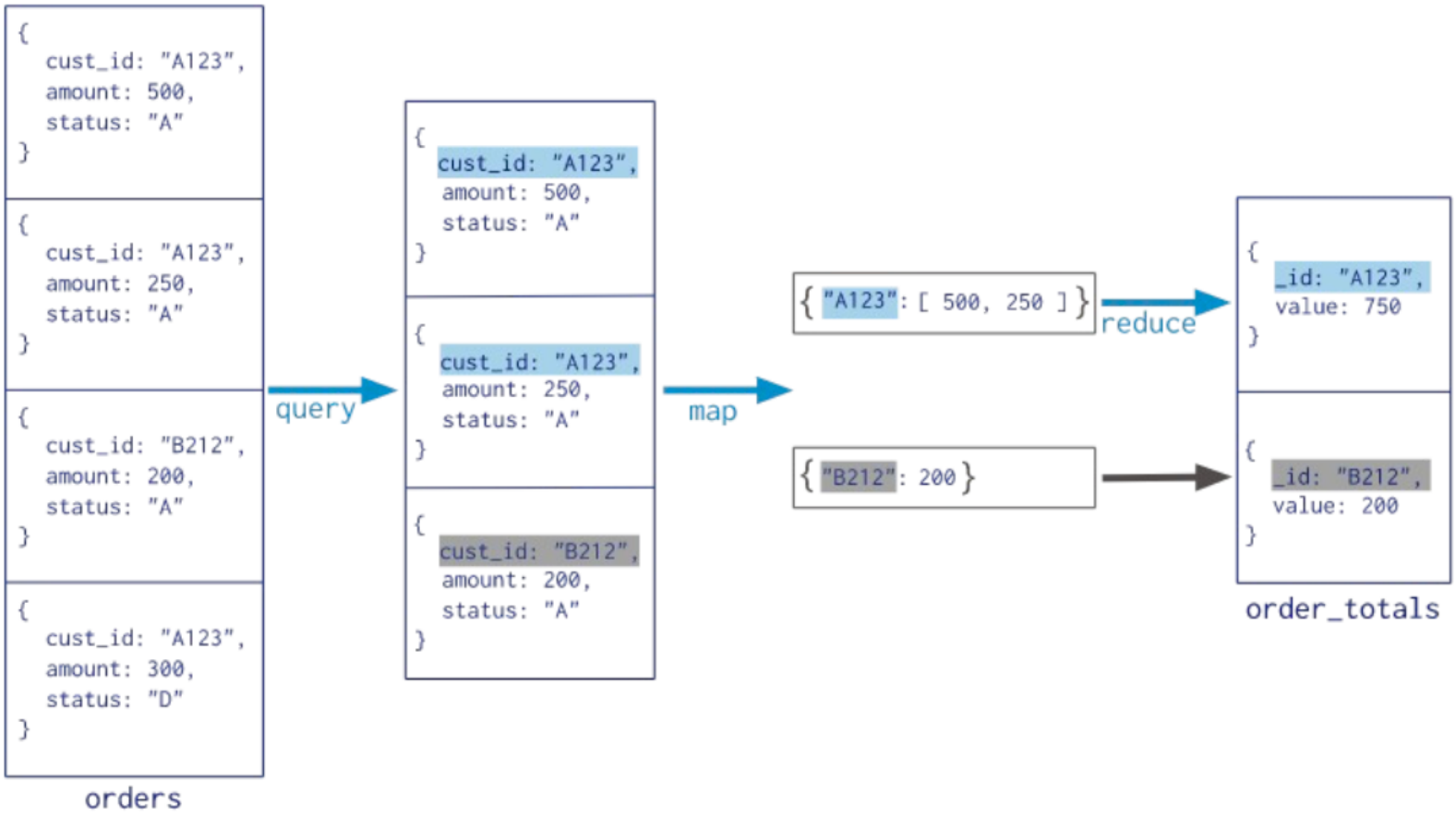
The result after reduction is:

(A B) -> (C D)
(A C) -> (B D)
(A D) -> (B C)
(B C) -> (A D E)
(B D) -> (A C E)
(B E) -> (C D)
(C D) -> (A B E)
(C E) -> (B D)
(D E) -> (B C)

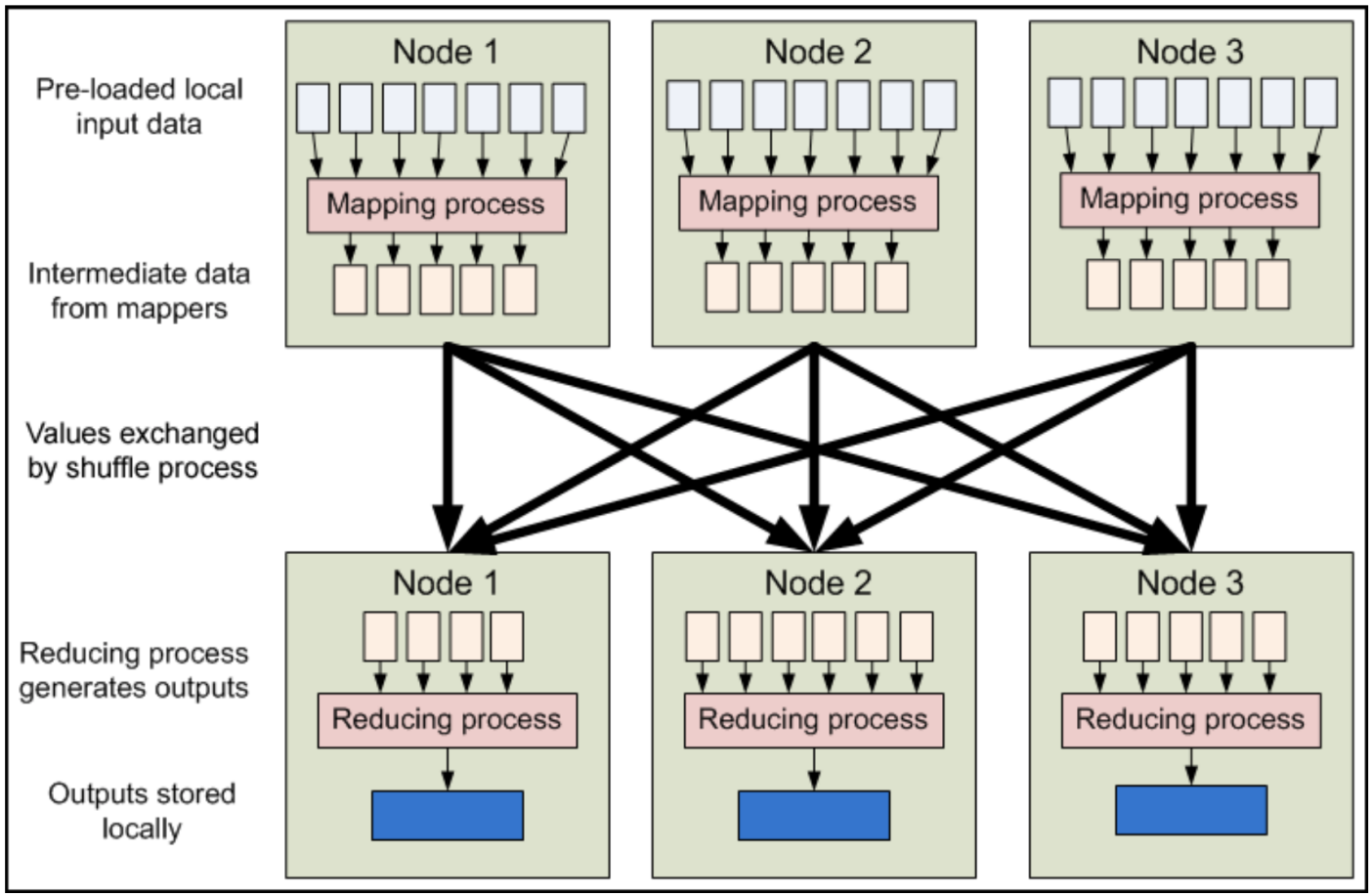
AGGREGATION MAP-REDUCE

Collection

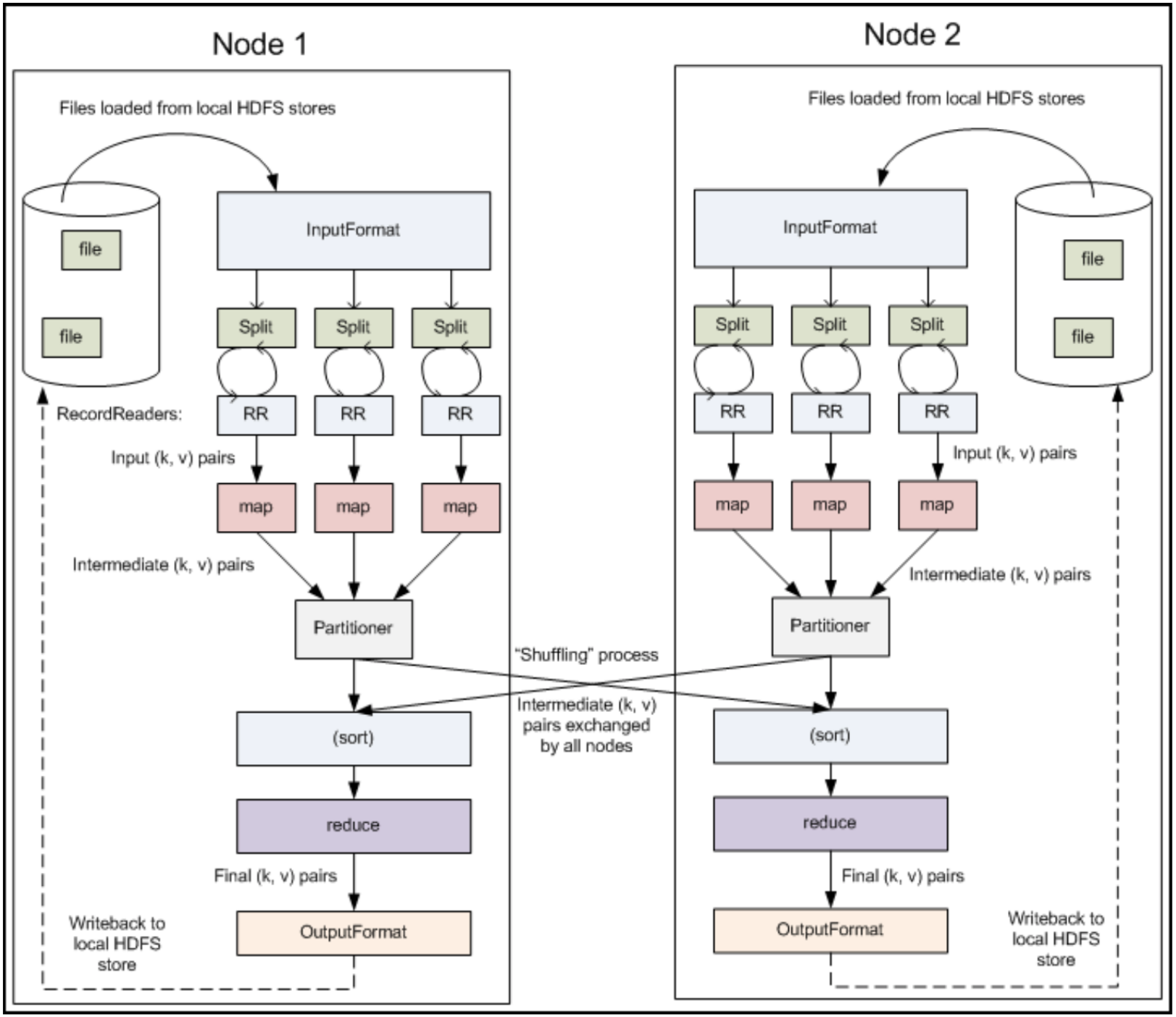
```
db.orders.mapReduce(
          map      →   function() { emit( this.cust_id, this.amount ); },
          reduce   →   function(key, values) { return Array.sum( values ) },
                       {
          query    →     query: { status: "A" },
          output   →     out: "order_totals"
                       }
                     )
```



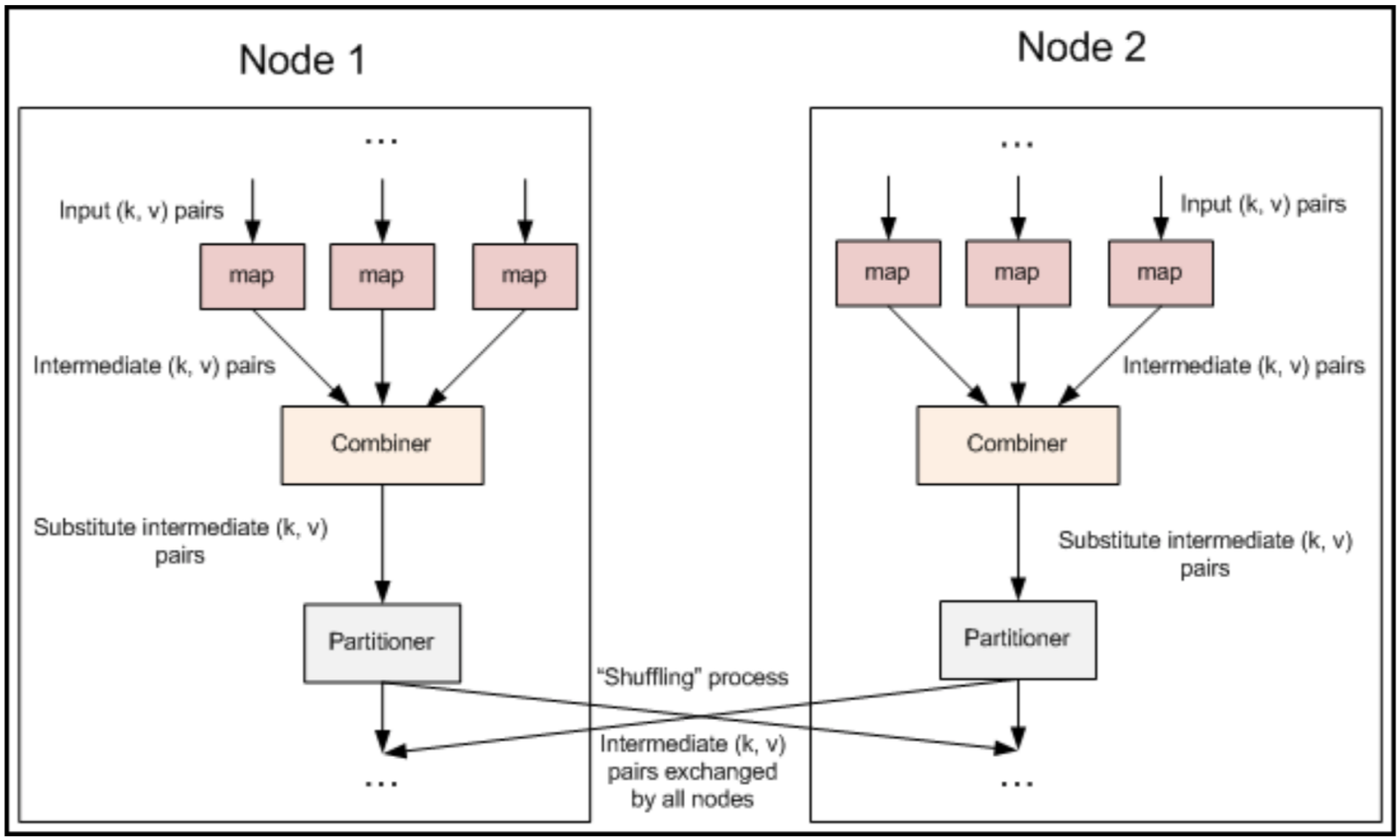
MapReduce: Flujo de datos



developer.yahoo.com/hadoop/tutorial/



developer.yahoo.com/hadoop/tutorial/



developer.yahoo.com/hadoop/tutorial/

Les dejo planteado un problema: Obtener una solución Map-Reduce para la necesidad de información representada por la siguiente consulta SQL

```
SELECT nombre_ciudad, COUNT(*)
FROM Habitantes, Ciudades
WHERE Habitantes.codc = Ciudades.codc
GROUP BY nombre_ciudad;
```

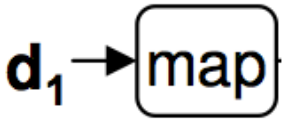
donde (1) los habitantes y ciudades de Chile están almacenados en un mismo archivo de texto sobre un sistema de archivos distribuido; (2) existen P mappers y P reducers; y (3) las ciudades y personas están almacenadas al azar en el archivo de texto con la restricción de que cada persona y ciudad se almacenan en una línea de texto de manera individual con campos separados por el carácter “,”.

Inverted Index Construction

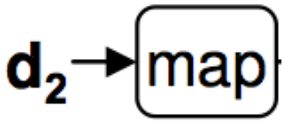
- **Map** function parses each document
  - Emit *term* as **key** and (*doc_id* , *tf*) as **value**
  - Emit other information as necessary (e.g. term position)
- **Reduce**
  - Each value represents a posting
  - Might want to sort the postings (e.g., by *doc_id* or *tf*)

- The set of all output pairs forms a simple inverted index

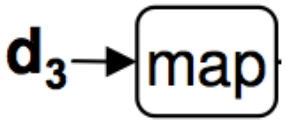
Inverted Index Construction



"A A B C"

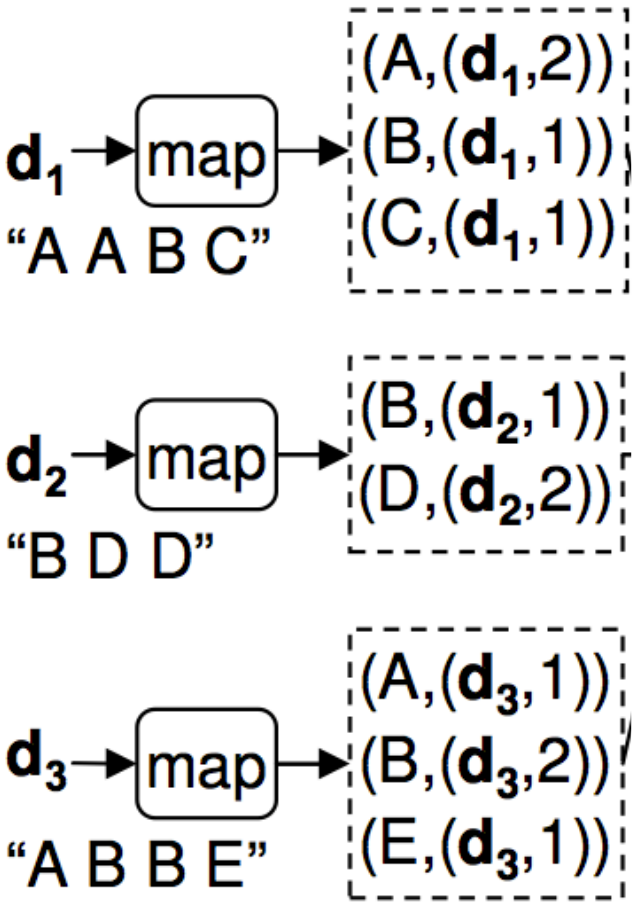


"B D D"

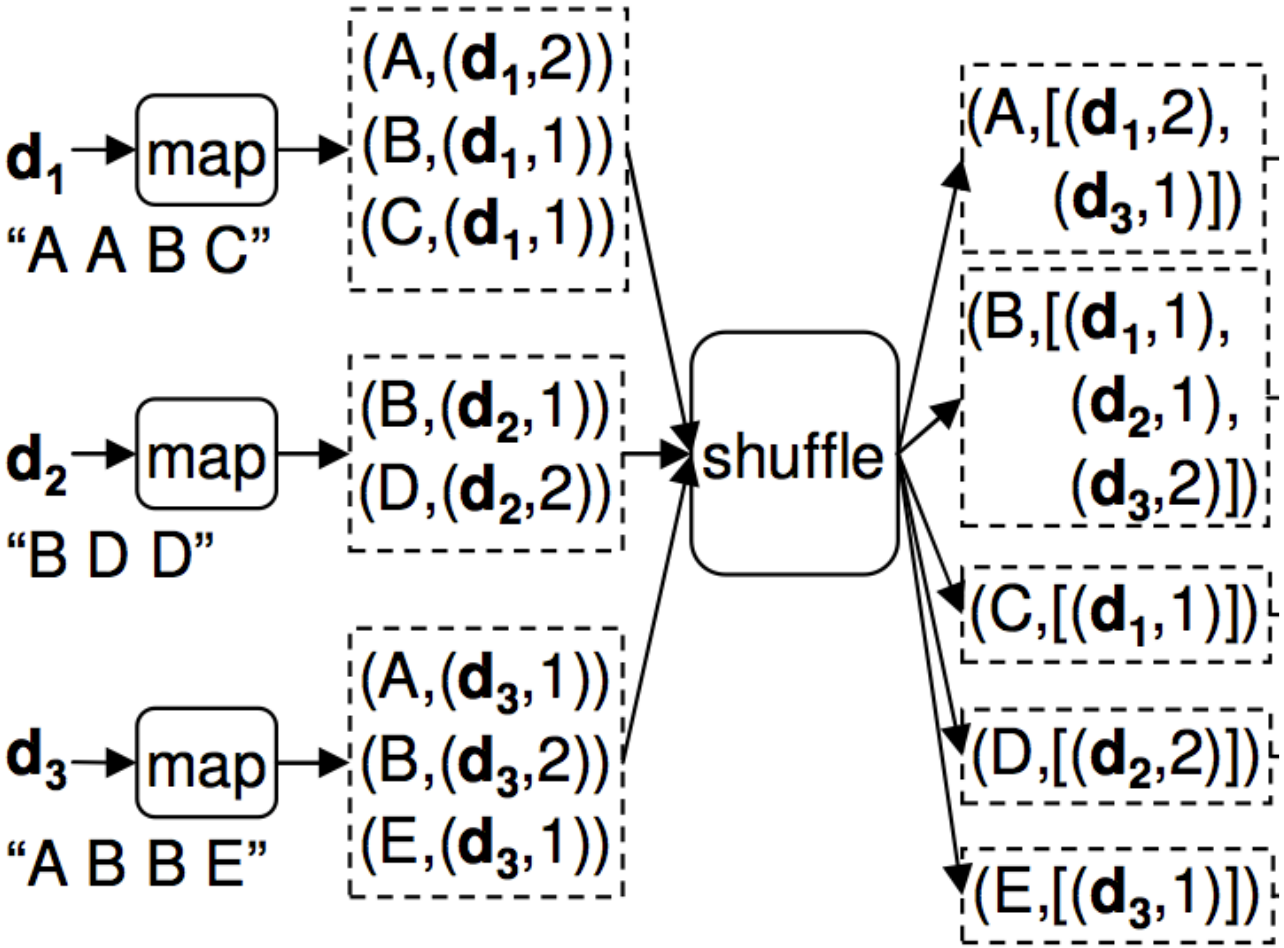


"A B B E"

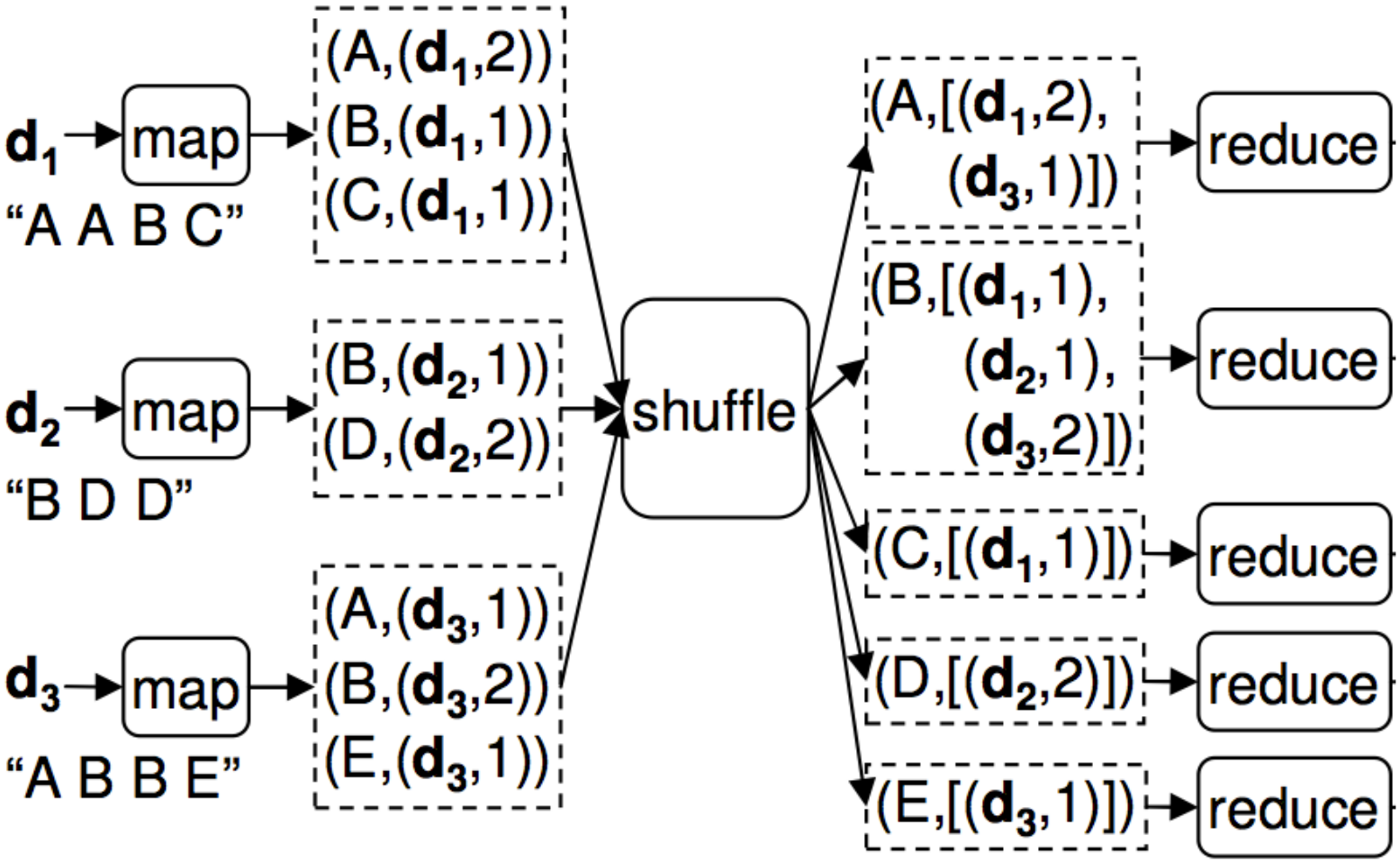
Inverted Index Construction



Inverted Index Construction

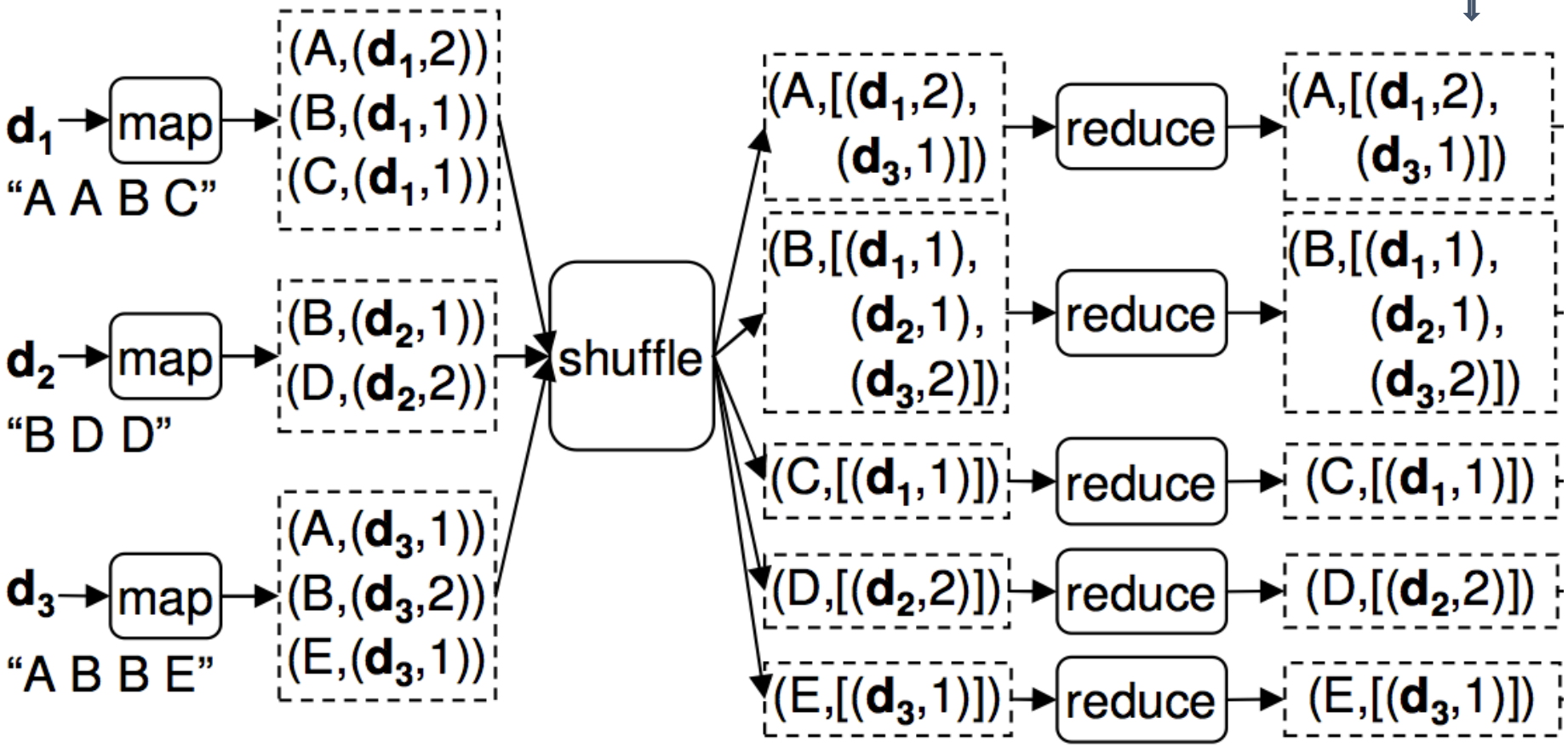


Inverted Index Construction

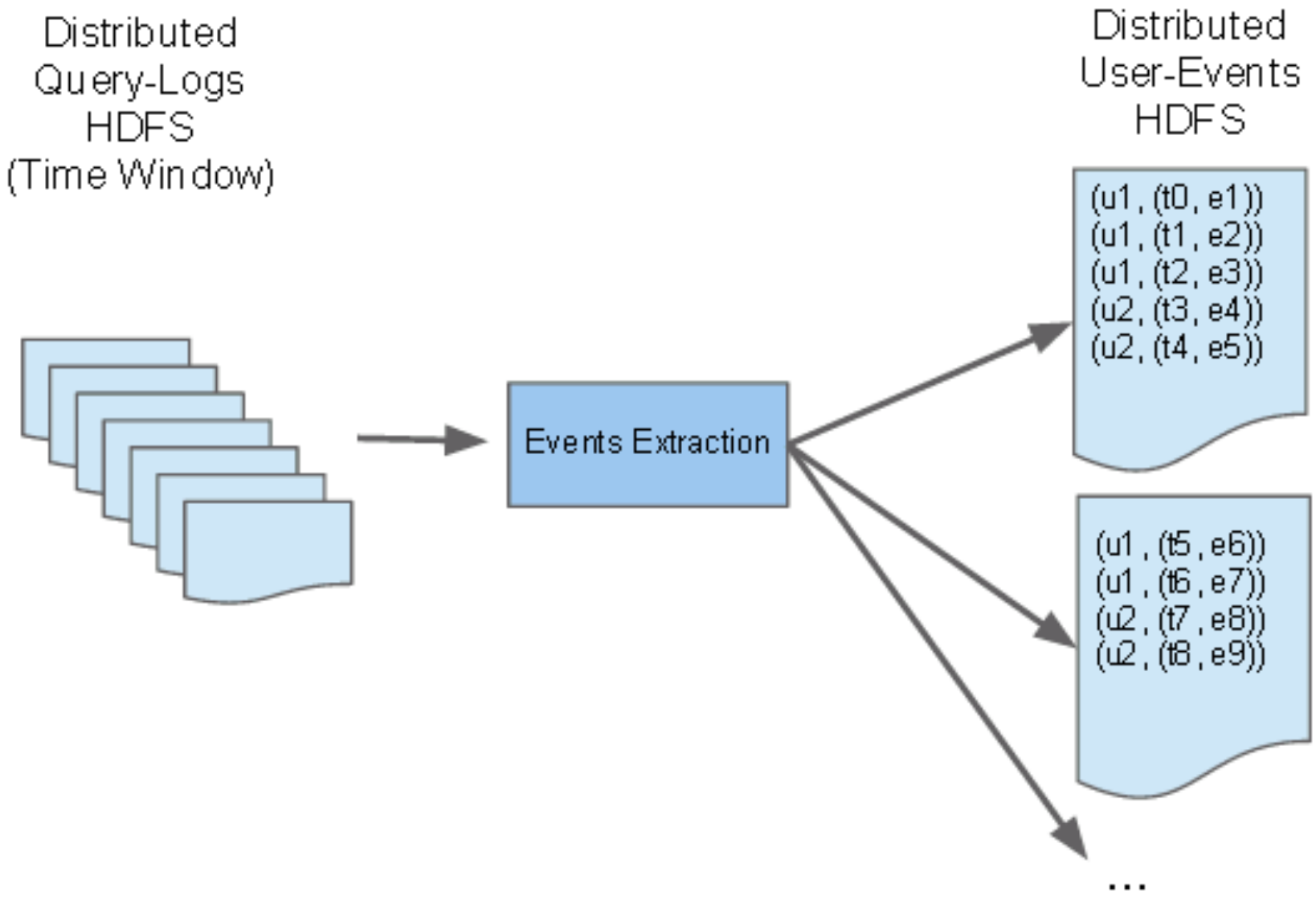


Inverted Index Construction

Indice comprimido generado por los reducers

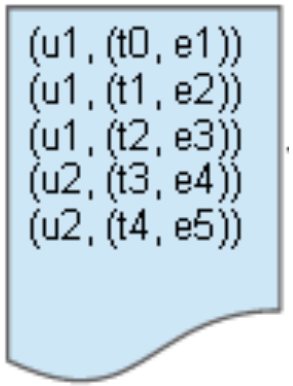


Session detection

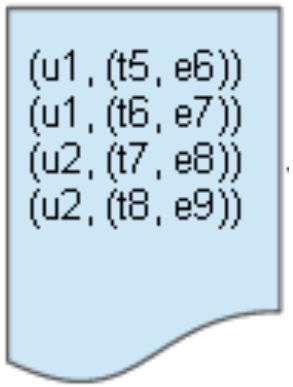


Session detection

mappers: cat user events documents (HDFS)
reducers: session and metrics extraction (python script)

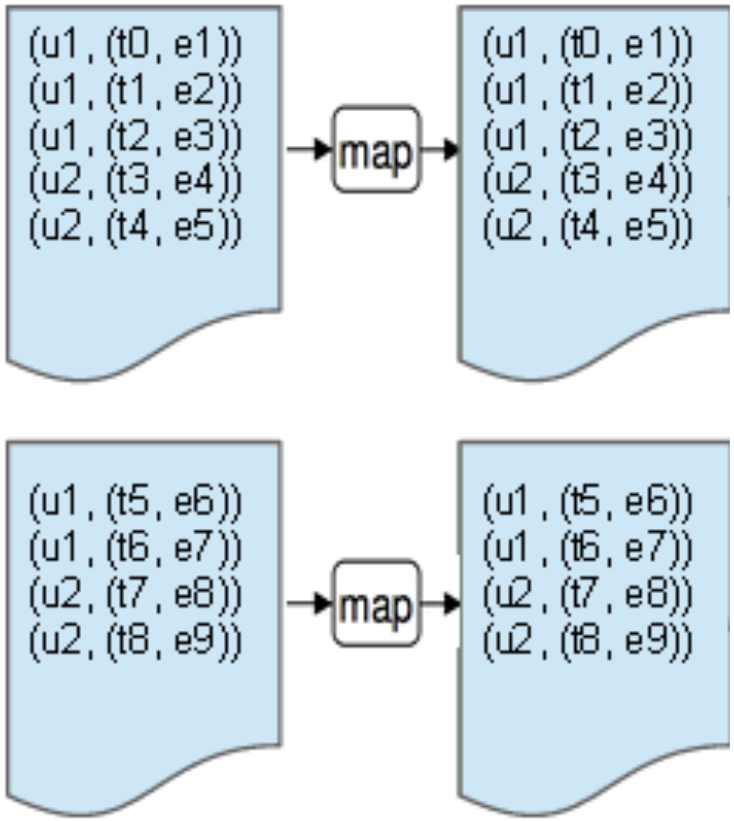


Distributed file containing a huge
query log in compressed text format



Session detection

mappers: cat user events documents (HDFS)
reducers: session and metrics extraction (python script)

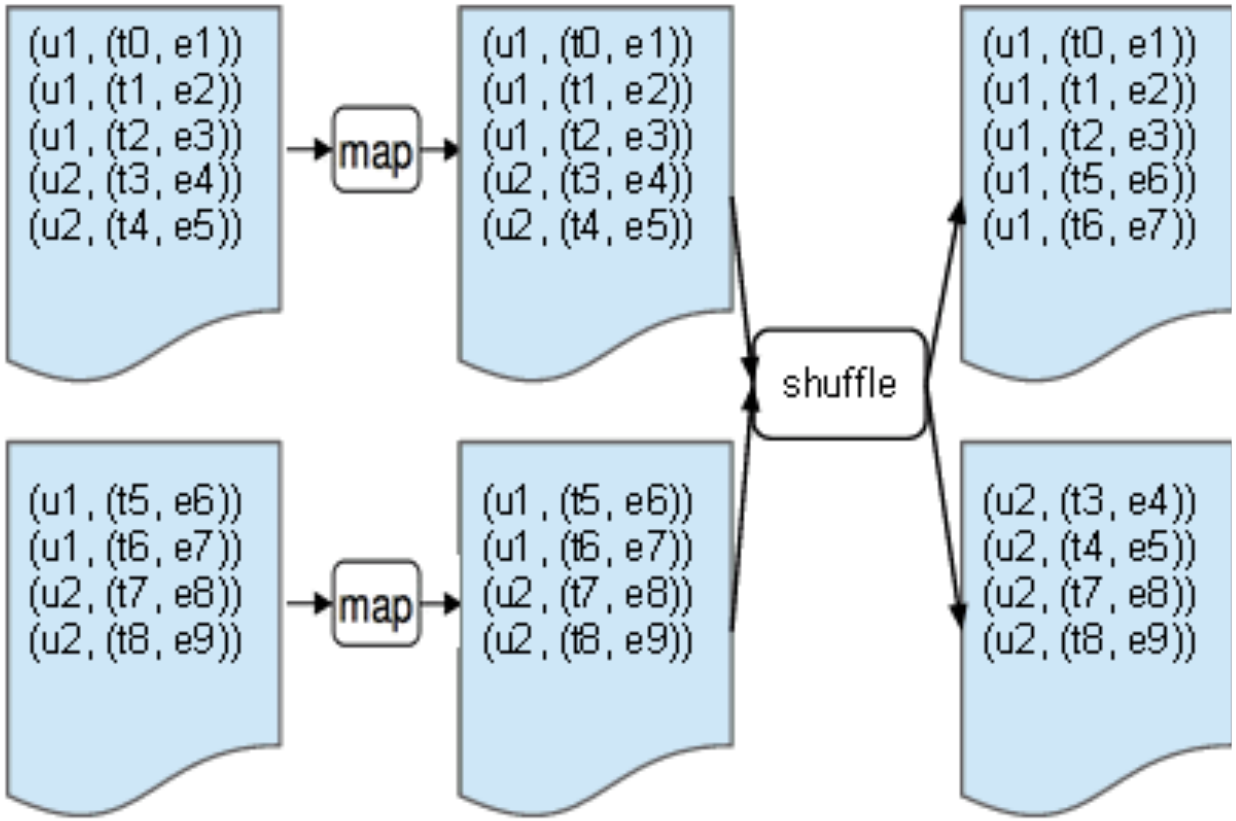


Hasta aquí los mappers han ingresado dentro de Hadoop tuplas del tipo

(*llave*= **UserID**, *valor*={timestamp,clickedURL, queryID})

Session detection

mappers: cat user events documents (HDFS)
reducers: session and metrics extraction (python script)



Session detection

mappers: cat user events documents (HDFS)
reducers: session and metrics extraction (python script)