

Paradigma



MAPREDUCE

Prof. Waldo Hasperué
whasperue@lidi.info.unlp.edu.ar

MapReduce

- Es un framework para distribuir tareas en múltiples nodos
- El espíritu de MapReduce es "escriba una vez y lea muchas"
- Ventajas
 - Paralelización y distribución de tareas automática
 - Escalable
 - Tolerante a fallos
 - Monitoreo y capacidad de seguridad
 - Flexibilidad de programación (Java, Python, C#, Ruby, C++)
 - Abstracción al programador

MapReduce

- Es, a su vez, un paradigma de programación.
- Hay que pensar como resolver un problema sin tener acceso a todos los datos
- Ejemplo (cálculo del promedio):

```
acum = 0
```

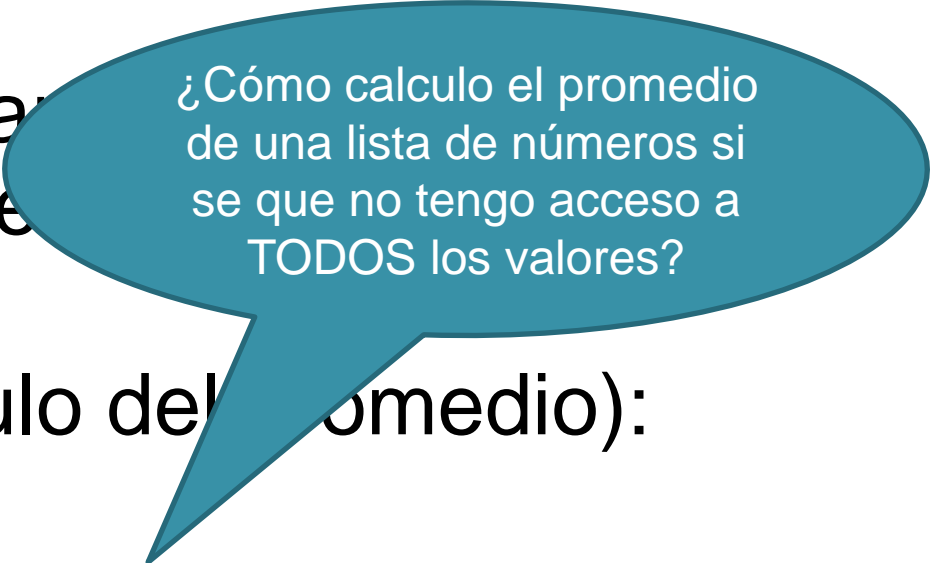
```
for d in datos:
```

```
    acum = acum + d
```

```
prom = suma / len(datos)
```

MapReduce

- Es, a su vez, un paradigma de programación.
- Hay que pensar el problema sin tener todos los datos



¿Cómo calculo el promedio de una lista de números si se que no tengo acceso a TODOS los valores?

- Ejemplo (cálculo del promedio):

```
acum = 0
```

```
for d in datos:
```

```
    acum = acum + d
```

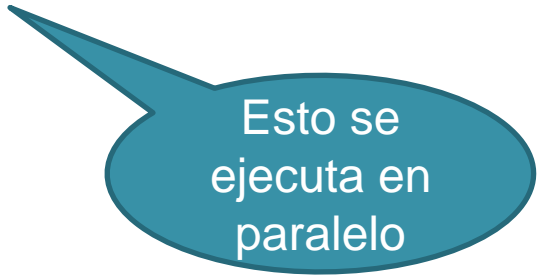
```
prom = suma / len(datos)
```

MapReduce

- El problema del cálculo del promedio se debe "repensar".

Pedirle a cada nodo que sume y cuente sus datos

```
acum = 0; n = 0
for nodo in cluster:
    acum = acum + nodo.acum
    n = n + nodo.n
promedio = acum / n
```



Esto se ejecuta en paralelo

MapReduce

- Se pensó en un proceso genérico que permita resolver cualquier problema
 - Paradigma MapReduce
- Toda tarea MapReduce se divide en dos fases:
 - Fase **map**: en la que los datos de entrada son procesados, uno a uno, y transformados en un conjunto intermedio de datos.
 - Fase **reduce**: se reúnen los resultados intermedios y se reducen a un conjunto de datos resumidos, que es el resultado final de la tarea.

MapReduce

En el ejemplo del promedio:

Map

Pedirle a cada nodo que sume y cuente sus datos

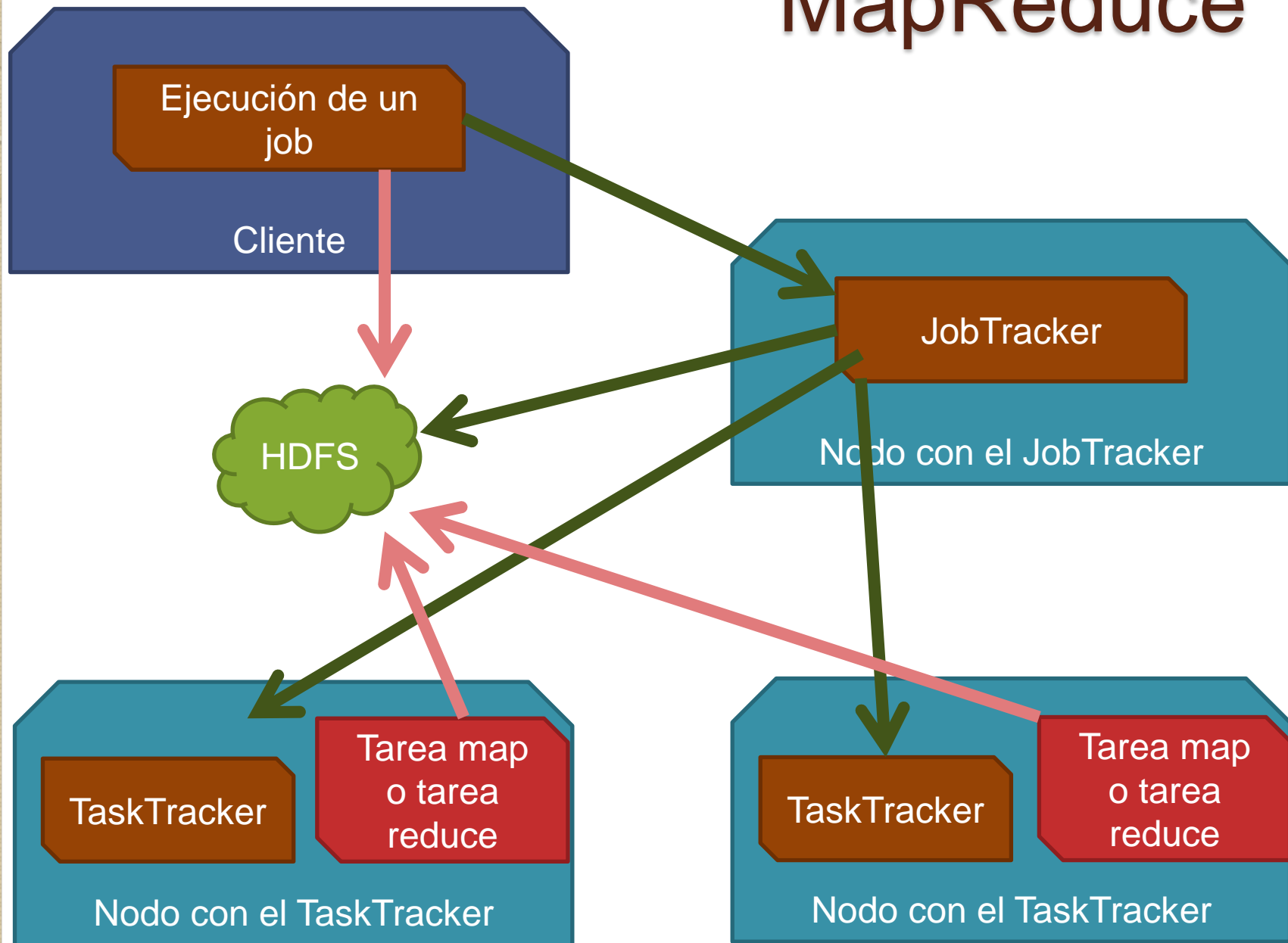
Reduce

```
acum = 0; n = 0
for nodo in cluster:
    acum = acum + nodo.acum
    n = n + nodo.n
promedio = acum / n
```

MapReduce

- La unidad de trabajo de MapReduce es un Job
- Un Job se divide en una tarea map y una tarea reduce.
- Los Jobs de MapReduce son controlados por un daemon conocido como JobTracker, el cual reside en el "nodo master"
- Los clientes envían Jobs MapReduce al JobTracker y este distribuye la tarea usando otros nodos del cluster
- Esos nodos se conocen como TaskTracker y son responsables de la ejecución de la tarea asignada y reportar el progreso al JobTracker

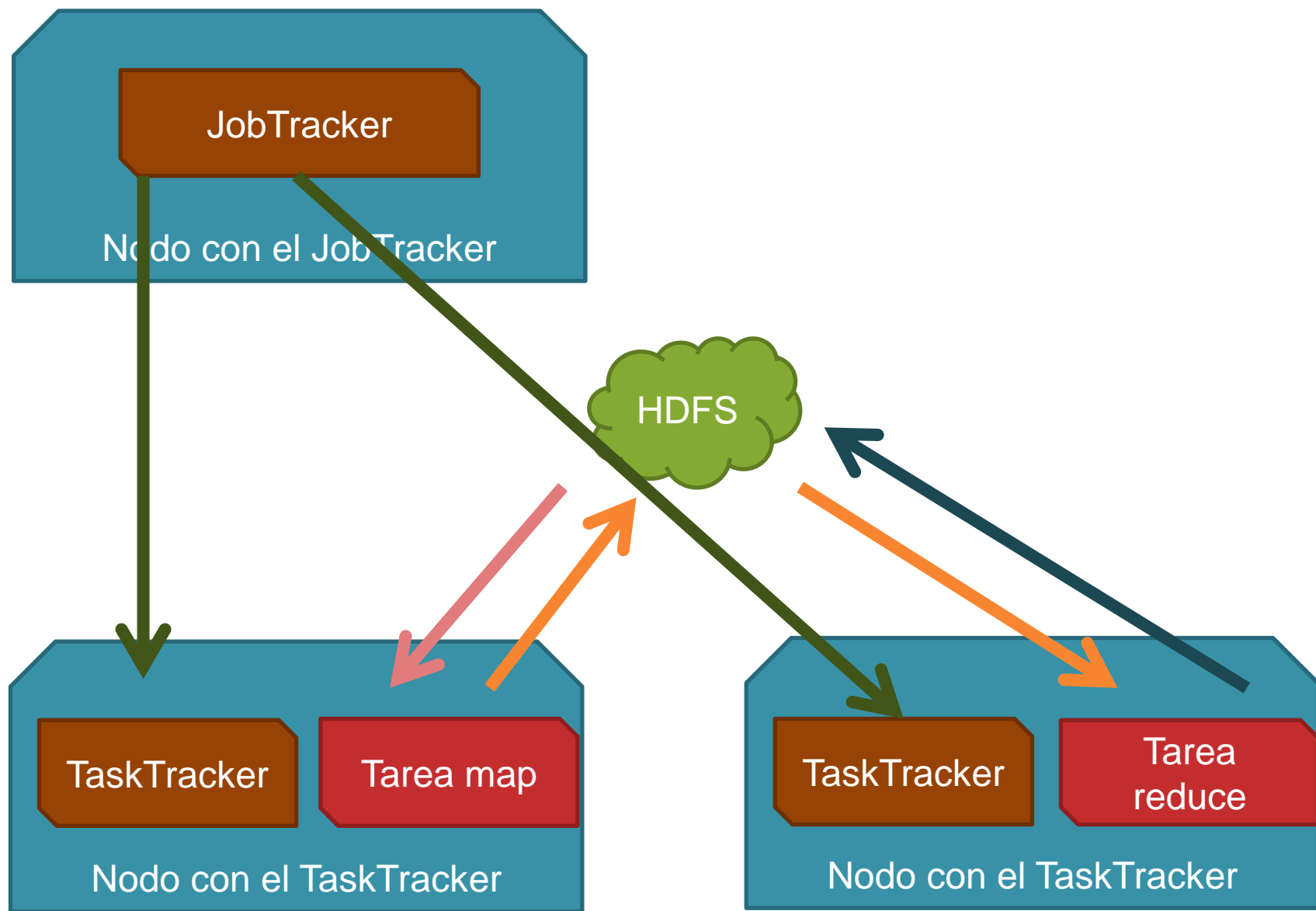
MapReduce



MapReduce

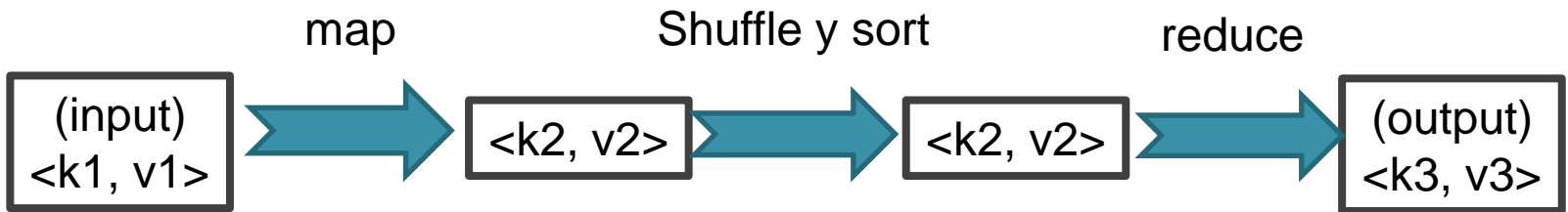
- Un job MapReduce es un proceso que se divide en cuatro fases:
 - Map → Shuffle → Sort → Reduce
- *Map* y *reduce* son las tareas que se deben programar para la aplicación.
- Cada TaskTracker ejecuta la tarea encomendada (*map* o *reduce*)
- *Shuffle* y *sort* son internas en la ejecución del job.

MapReduce



MapReduce

- Las tareas de map y reduce trabajan con el concepto de <clave, valor>



MapReduce

- Entrada de datos
 - MapReduce se "alimenta" de uno o mas archivos:
 - En el caso de archivos de texto plano, cada línea del archivo es un dato a procesar.
 - La clave de los archivos de texto plano es el offset de la línea dentro del archivo.
- El o los archivos de entrada son divididos en "splits" y cada TaskTracker trabaja sobre un "split".

Tarea map

- Se ejecutan múltiples instancias de la tarea map sobre diferentes porciones del dataset.
- Se intenta que cada map se ejecute sobre una copia local del dataset para minimizar el tráfico de datos. Una tarea map solo ve una porción del dataset de entrada.
- La tarea map lee los datos en forma de pares $\langle k1, v1 \rangle$ y produce una lista de cero, uno o más pares $\langle k2, v2 \rangle$.
 - $\langle k1, v1 \rangle \rightarrow \text{list}(\langle k2, v2 \rangle)$
donde **k2** es un identificador definido para el problema utilizado para agrupar los datos.
- Una tarea map debería analizar una tupla en forma independiente del resto de las tuplas.

Tarea map

Ejemplo del promedio:

Suponemos que los valores llegan todos juntos, en formato string, separados por algún carácter que permite hacer la operación de split.

```
def map(k1, v1):  
    values = v1.split()  
    acum = 0  
    for v in values:  
        acum = acum + float(v)  
    return ( 1 , (acum, len(values)) )
```

Tarea map

Ejemplo del promedio:

Por lo general la clave de entrada no se usa, aunque depende del problema. En este ejemplo no lo estamos usando

```
def map(k1, v1):  
    values = v1.split()  
    acum = 0  
    for v in values:  
        acum = acum + float(v)  
    return ( 1 , (acum, len(values)) )
```


Tarea map

Ejemplo del promedio:

```
def map(k1, v1):  
    values = v1.split()  
    acum = 0  
    for v in values:  
        acum = acum + float(v)  
    return ( 1 , (acum, len(values)))
```

Para este
ejemplo, la clave
de salida es un
valor arbitrario
(Ya veremos
porqué)

Tarea map

Ejemplo del promedio:

```
def map(k1, v1):  
    values = v1.split()  
    acum = 0  
    for v in values:  
        acum = acum + float(v)  
    return ( 1 , (acum, len(values)))
```

Como valor de
salida
devolvemos la
tupla (acum, n)

Tarea reduce

- Finalizadas las tareas intermedias *shuffle* y *sort* se ejecuta la tarea *reduce*.
- La tarea reduce lee los datos en forma de pares $\langle k2, \text{list}(v2) \rangle$ y produce una lista de cero o más pares $\langle k3, v3 \rangle$.
 - $\langle k2, \text{list}(v2) \rangle \rightarrow \text{list}(\langle k3, v3 \rangle)$
- La tarea reduce tiene todos los elementos para poder realizar cualquier operación de "resumen".

Tarea reduce

Ejemplo del promedio:

K2 es nuestro
valor arbitrario
"1", que no lo
usamos en este
ejemplo

```
def reduce(k2, v2):  
    acum = 0; n = 0  
    for v in v2:  
        acum = acum + v[0]  
        n = n + v[1]  
    return ( k2 , acum / n)
```

Tarea reduce

Ejemplo del promedio:

En v2 tenemos
TODAS las tuplas
(acum, n) devueltas
por cada
taskTracker que
ejecutó la tarea map

```
def reduce(k2, v2):  
    acum = 0; n = 0  
    for v in v2:  
        acum = acum + v[0]  
        n = n + v[1]  
    return ( k2 , acum / n)
```

Tarea reduce

Ejemplo del promedio:

```
def reduce(k2, v2):  
    acum = 0; n = 0  
    for v in v2:  
        acum = acum + v[0]  
        n = n + v[1]  
    return ( k2 , acum / n)
```

SIEMPRE
tendremos que
recorrer el
iterable v2
usando un for

Tarea reduce

Ejemplo del promedio:

```
def reduce(k2, v2):  
    acum = 0; n = 0  
    for v in v2:  
        acum = acum + v[0]  
        n = n + v[1]  
    return ( k2 , acum / n)
```

La salida del job será la tupla formada por una clave arbitraria (no importa en este ejemplo) y el cálculo del promedio

Ejemplo

- Se posee un dataset con resultados de eventos. De cada evento se conoce su resultado ("POSITIVO", "NEUTRO", "NEGATIVO").
- Se desea saber cuantos eventos positivos, neutrales y negativos hay en todo el dataset.

Ejemplo

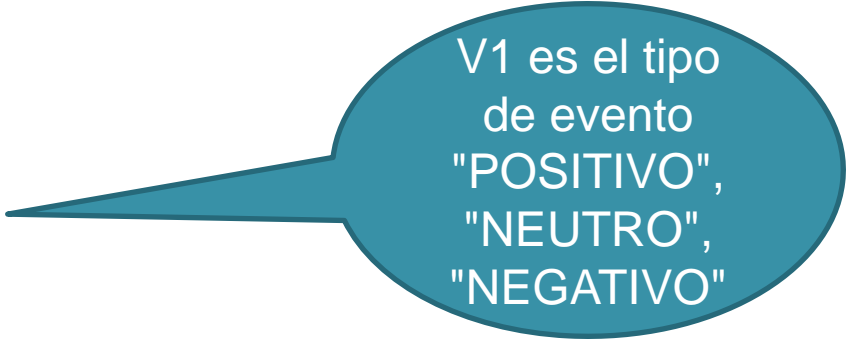
- El dataset es uno o más archivos de texto donde en cada línea está el resultado del evento

```
...  
POSITIVO  
POSITIVO  
NEGATIVO  
NEUTRO  
POSITIVO  
NEUTRO  
NEGATIVO  
NEGATIVO  
NEUTRO  
POSITIVO  
NEGATIVO  
...
```

Ejemplo

- Tarea map: nuestra intención es contar la ocurrencia de cada tipo de evento.

```
def map(k1, v1):
```

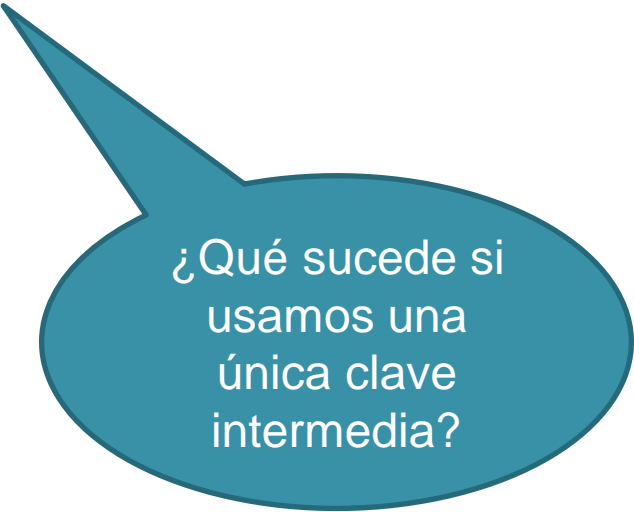


V1 es el tipo
de evento
"POSITIVO",
"NEUTRO",
"NEGATIVO"

Ejemplo

- Tarea map: nuestra intención es contar la ocurrencia de cada tipo de evento.

```
def map(k1, v1):  
    return ( 1 , v1)
```

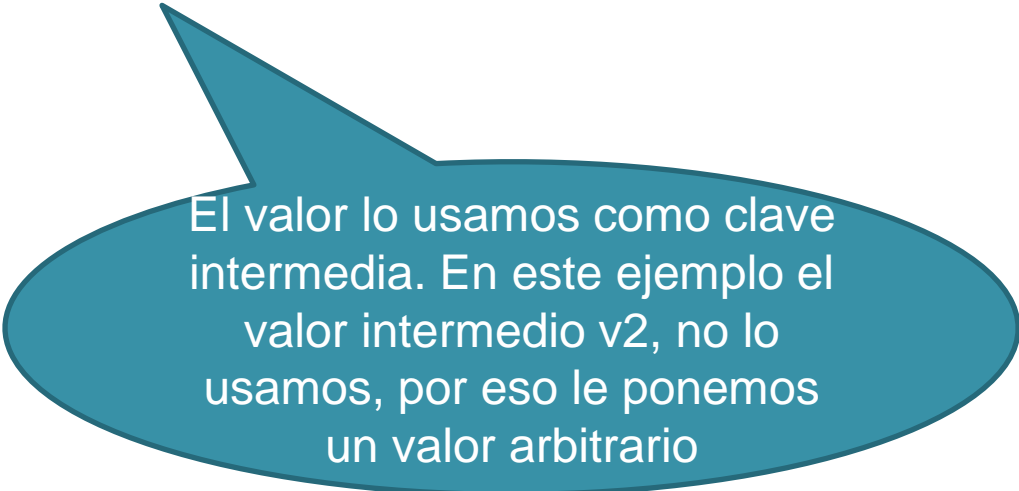


¿Qué sucede si
usamos una
única clave
intermedia?

Ejemplo

- Tarea map: nuestra intención es contar la ocurrencia de cada tipo de evento.

```
def map(k1, v1):  
    return ( v1 , 1)
```



El valor lo usamos como clave intermedia. En este ejemplo el valor intermedio v2, no lo usamos, por eso le ponemos un valor arbitrario

Tarea map - Ejemplo

Input

<1, POSITIVO>
<2, POSITIVO>
<3, NEGATIVO>

TT1

<4, NEUTRO>
<5, POSITIVO>
<6, NEUTRO>
<7, NEGATIVO>

TT2

<8, NEGATIVO>
<9, NEUTRO>
<10, POSITIVO>
<11, NEGATIVO>

TT3

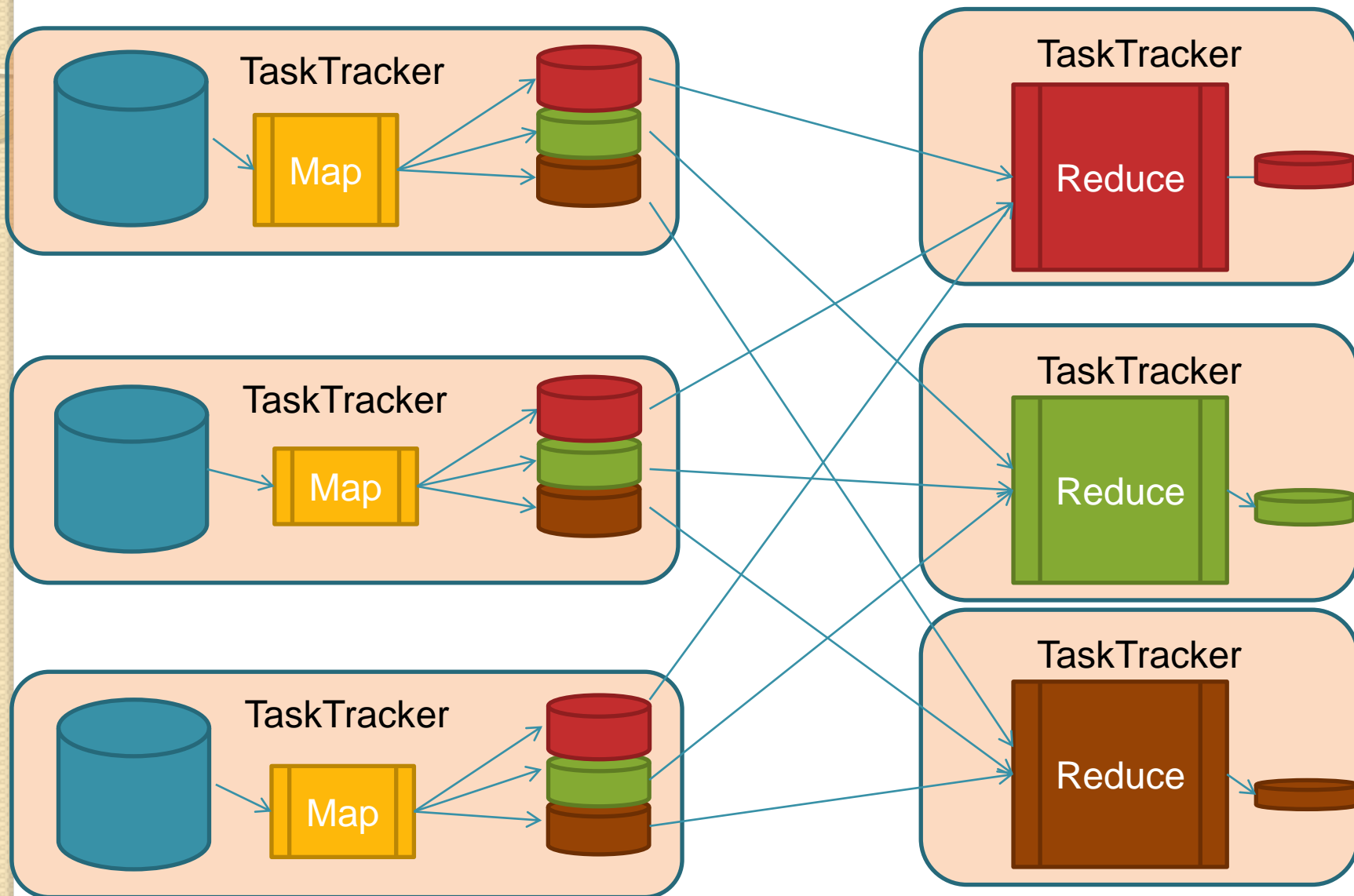
Output

<POSITIVO, 1>
<POSITIVO, 1>
<NEGATIVO, 1>

<NEUTRO, 1>
<POSITIVO, 1>
<NEUTRO, 1>
<NEGATIVO, 1>

<NEGATIVO, 1>
<NEUTRO, 1>
<POSITIVO, 1>
<NEGATIVO, 1>

Ejemplo



Ejemplo

- Cada TaskTracker que ejecuta la tarea de *reduce* recibe todas las tuplas del tipo "POSITIVO", "NEUTRO" o "NEGATIVO".
- Solo hay que contar cuantas ocurrencias existen.
- Tendremos una salida por cada tipo de evento

Ejemplo

- Solo hay que contar ocurrencias

```
def reduce(k2, v2):  
    n = 0  
    for v in v2:  
        n = n + 1  
    return ( k2 , n)
```

En este ejemplo se ejecutan tres *reducers*.

Cada uno hace la contabilidad de las tuplas que se generaron para cada una de las tres claves ("POSITIVO", "NEUTRO", "NEGATIVO")

Ejemplo "POSITIVO"

Input

<POSITIVO, [1

, 1

, 1

, 1

, 1

, 1

, 1

, 1

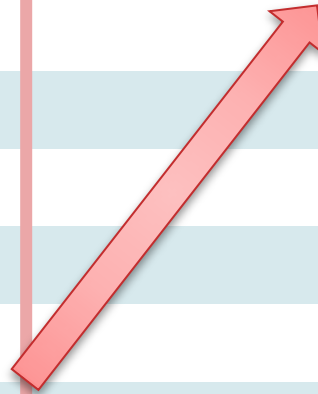
, 1

, 1

, 1]>

Output

<POSITIVO, 11>



TT4