Conceptos y Aplicaciones de Big Data

MapReduce

- Combiners
- Múltiples jobs
- Parametrización de task

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

Temario

- Función *combiner*
- Problemas de múltiples *jobs*
- Pasaje de información a *mappers* y *reducers*

Repaso – Etapa map

- Trabaja sobre un split de los datos.
- Es fácilmente paralelizable, se ejecutan tantos *mappers* como splits existan en el cluster.
- Por cada par clave-valor procesado, puede "escribir" como salida de 0 a *M* pares resultado.

<k1, v1> → list(<k2,v2>)

Repaso – Etapa map

```
def fmap(key, value, context):
    words = value.split()
    for w in words:
        context.write(w, 1)
```

Los datos se procesan "como vienen". No existe ningún orden ni por clave, ni por valor

Repaso – Etapa reduce

- Trabaja sobre todas la tuplas que tienen una misma clave.
- Se paraleliza por clave, se ejecutan tantos *reducers* como claves se generaron en la etapa *map*.
- Por cada par clave-lista(valor) procesado, puede "escribir" como salida de 0 a *N* pares resultado.

<k2, list(v2)> → list(<k3, v3>)

Repaso – Etapa reduce

```
def fred(key, values, context):
    c=0
    for v in values:
        c=c+1
    context.write(key, c)
```

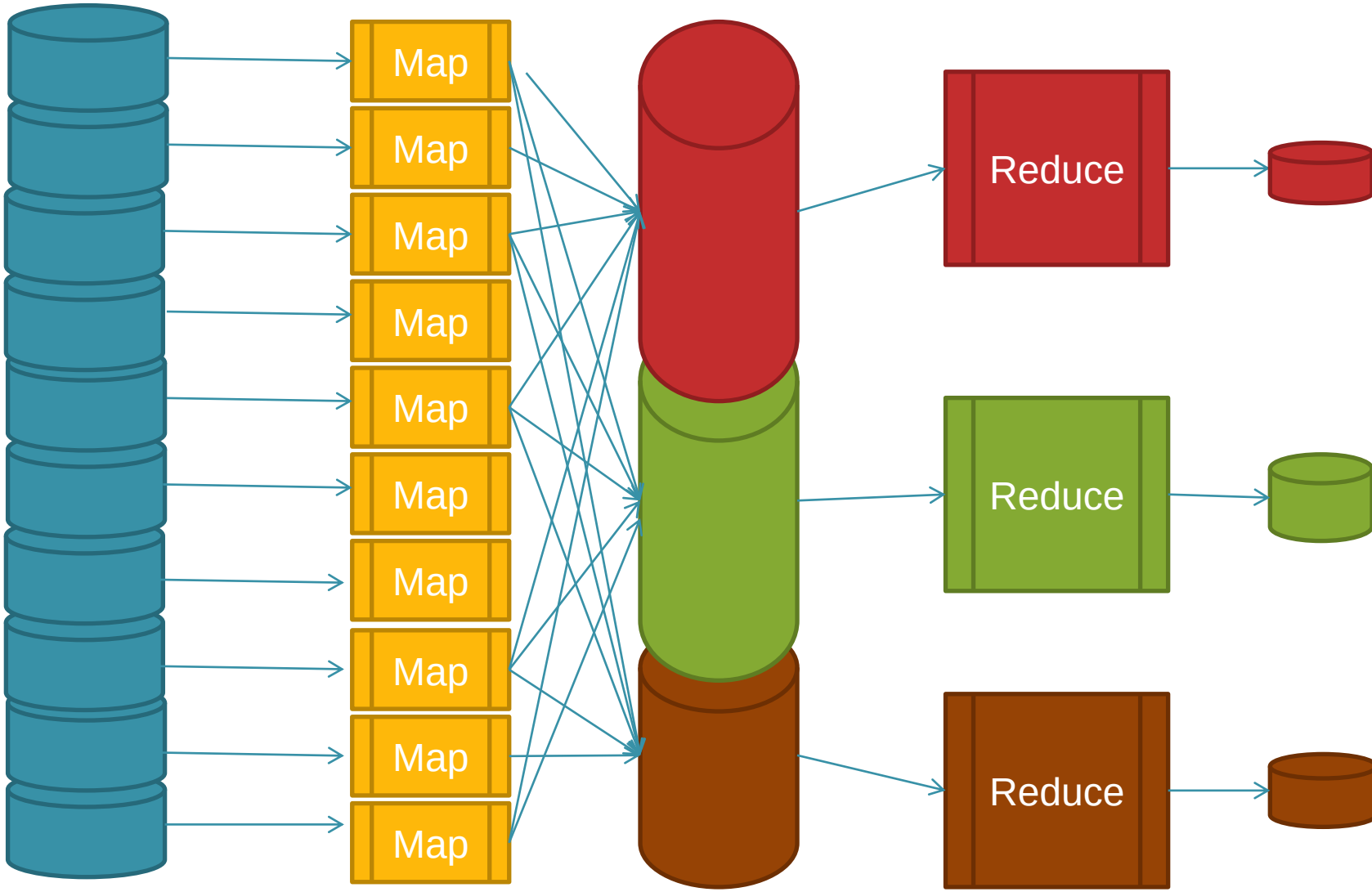
Se invoca a la función tantas veces como claves se tengan que procesar.
No hay forma de saber cuando es la última llamada a esta función.
La única garantía (si fuera llamada más de una vez) es que, en sucesivas invocaciones las claves vienen ordenadas.

Repaso – Etapa reduce

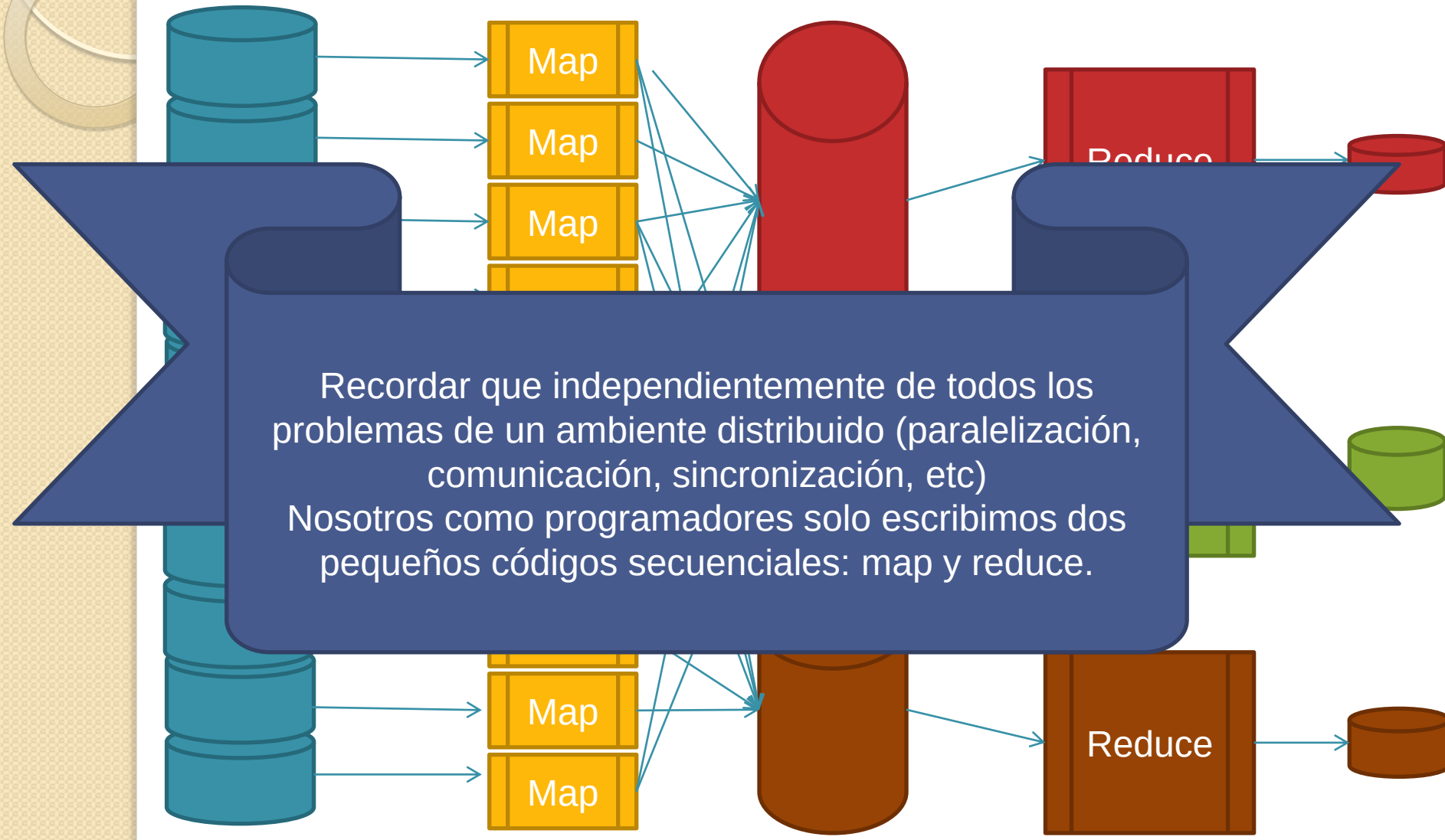
```
def fred(key, values, context):
    c=0
    for v in values:
        c=c+1
    context.write(key, c)
```

Por cada clave se recibe la lista de valores asociados.
Esta lista solo puede ser recorrida una única vez.
NO se puede acceder a los elementos por índice.
NO se puede adelantar ni rebobinar.

Repaso – Map y Reduce



Repaso – Map y Reduce



Ejemplo

- Un hipermercado tiene almacenada todas las compras del último bimestre en todas sus sucursales.

  <id_sucursal, id_compra, monto>
- Se desea saber cuál es la compra de mayor dinero.
  - ¿Qué hace el *map*?
  - ¿Qué hace el *reduce*?
  - ¿Con cuanta información trabaja el *reduce*?

Ejemplo – ¿Qué hace el map?

```
def fmap(key, value, context):
    context.write(key, value)
```

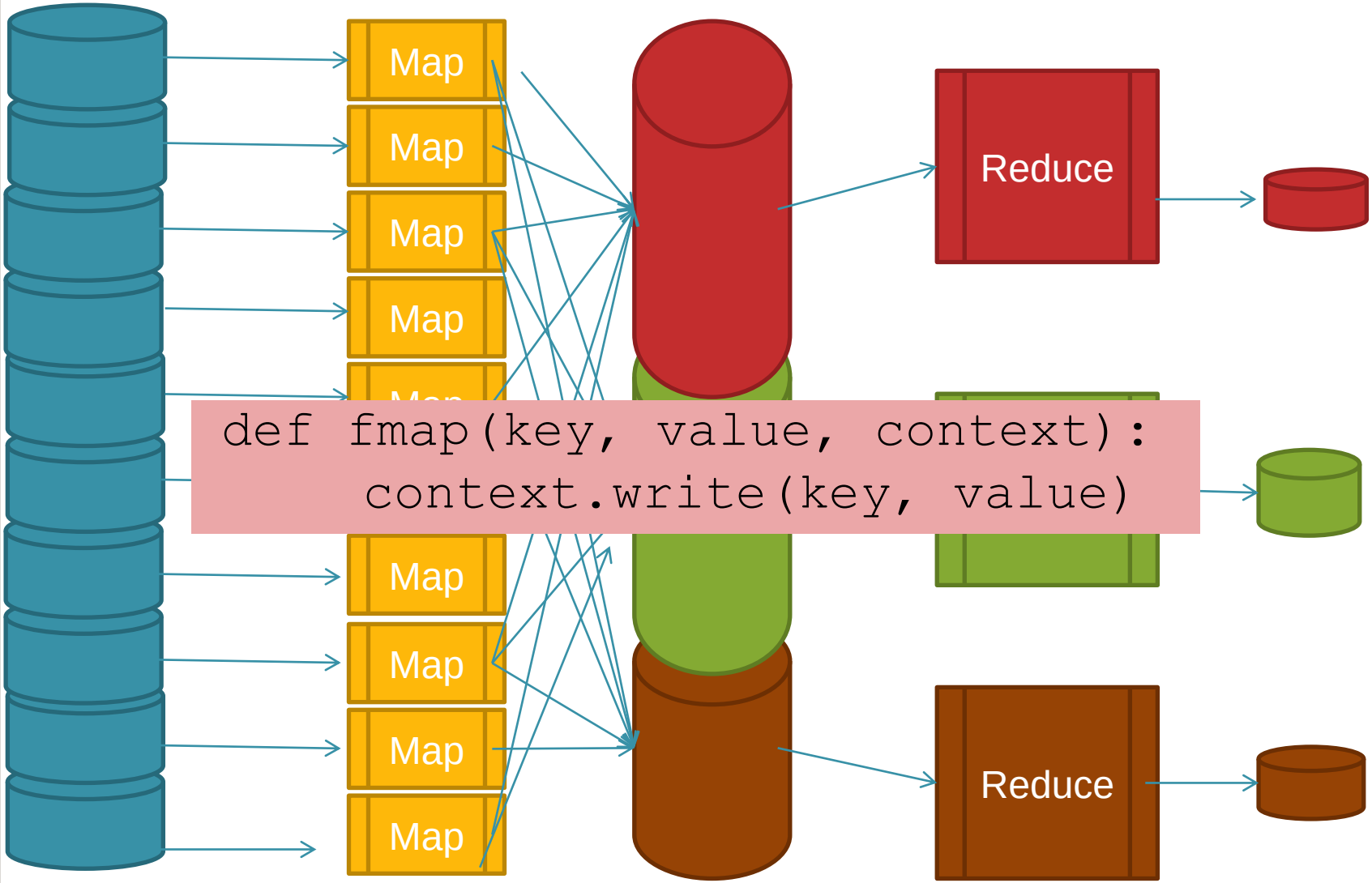
Si el id de sucursal es la *key* y el id de compra y monto vienen en *value*, entonces el map lo único que puede hacer es pasar esa información al reduce y no mucho más.

Ejemplo – ¿Qué hace el reduce?

```
def fred(key, values, context):
    max = -1
    for v in values:
        if v[1] > max:
            max = v[1]
            id_c = v[0]
    context.write(key, (id_c, max))
```

El *reducer* busca el máximo monto asociado a un id_sucursal

Ejemplo – ¿Con cuanta información trabaja el reduce?



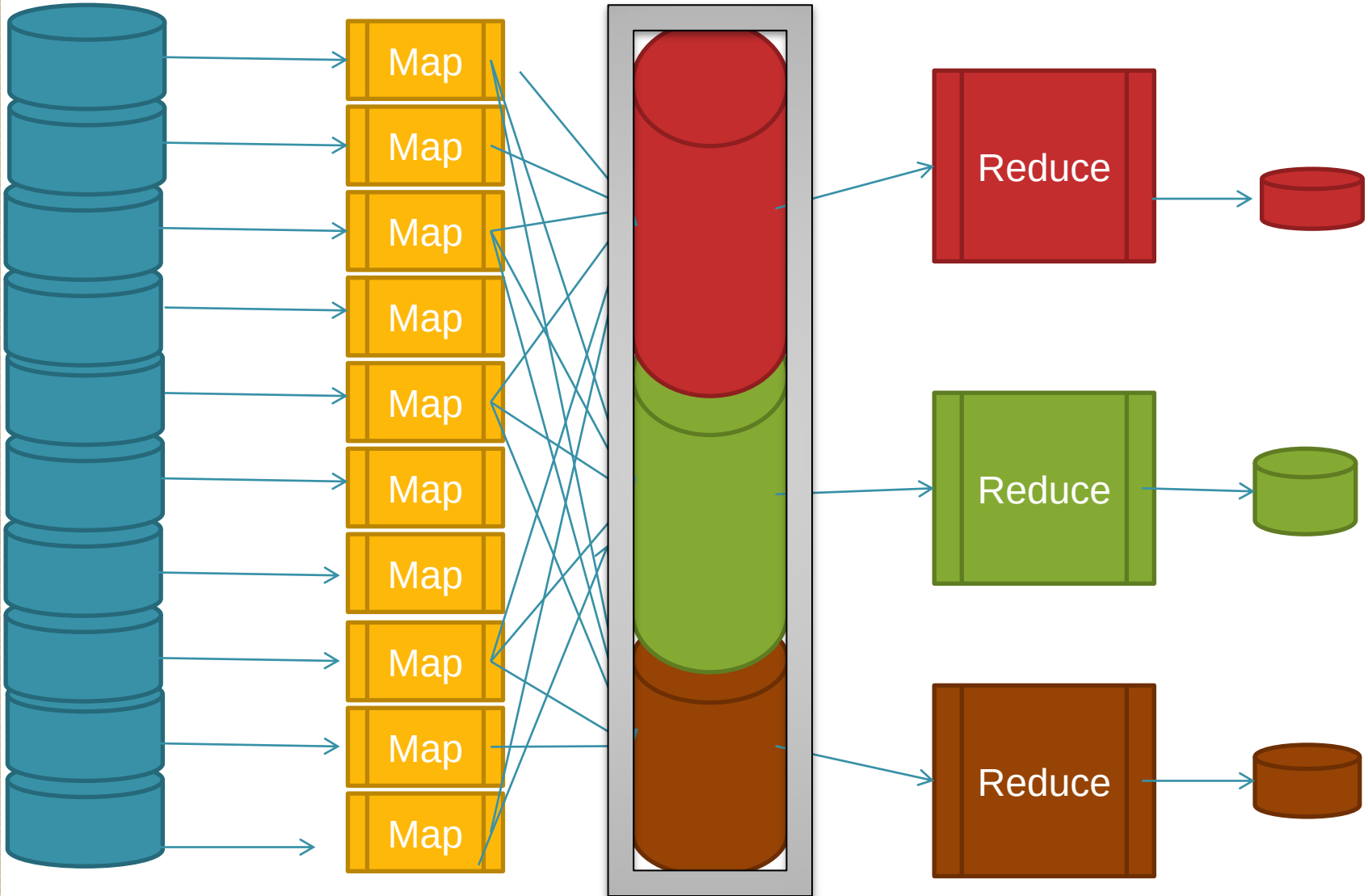
Problema

- Una solución "rebuscada" sería que cada *mapper* devuelva su máximo…
- …pero si queremos el máximo por sucursal… ¿tiene sentido complicar aún más la función *map*?
- La solución más "elegante" y respetar la filosofía del paradigma es usar una función *combiner*.

Función combiner

- La gran desventaja que tiene el proceso de ejecución de un Job MapReduce es la enorme cantidad de datos que se deben mover en el cluster al finalizar la etapa map.
- En la mayoría de los problemas, los reducers trabajan con todos (o gran parte de ellos) los datos del dataset.
- La función *combiner* es una función que permite minimizar la cantidad de datos a "mover" por el cluster.

Lo que se busca es que el volumen de datos a guardar en disco sea lo más chico posible.



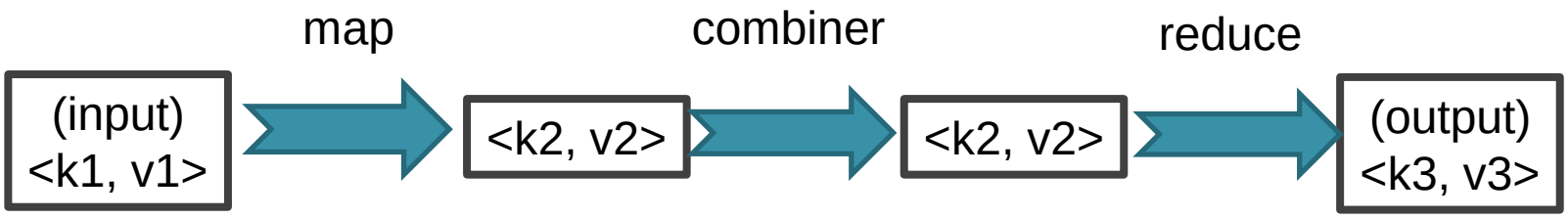
Función combiner

- Esta función es utilizada para "disminuir" la carga de datos en la tarea reduce.
- La función *combiner* se ejecuta con la salida de los *mappers*.
- Se ejecuta en el mismo nodo donde se ejecutó el *map*.
- Solo trabaja con la salida producida por la tarea *map* (antes de escribir los datos a disco).
- Hadoop no garantiza cuantas veces es invocada esta función, ya que internamente es solo una "tarea de optimización".

Función combiner

- La función *combiner* se ejecuta con la salida del *map* y su resultado es la entrada de la tarea *reduce*. Por lo tanto la interface de esta función es

<k2,list(v2)> ➔ list(<k2,v2>)

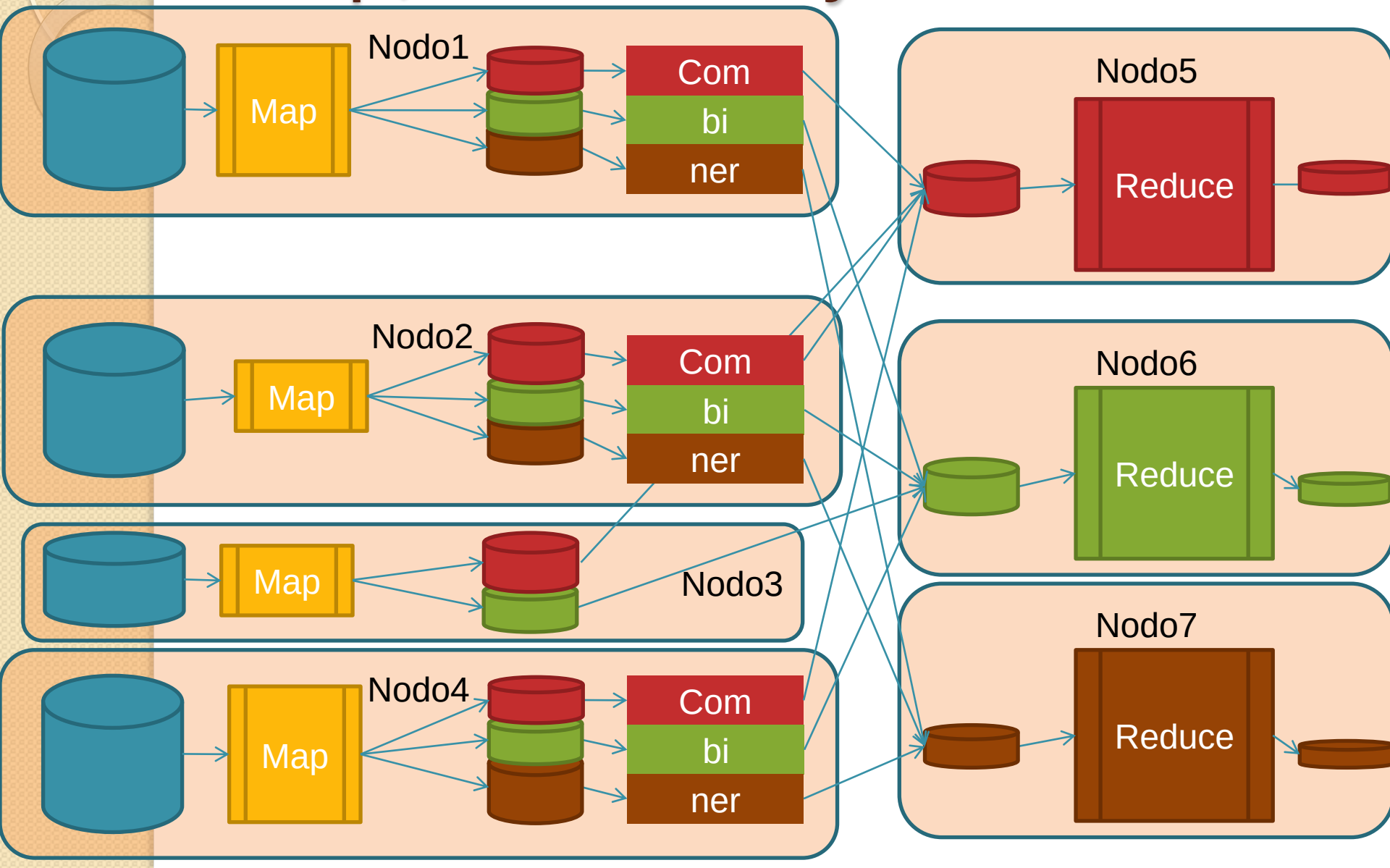


Función combiner

```
def fcom(key, values, context):
    c=0
    for v in values:
        c=c+1
    context.write(key, c)
```

La función combiner trabaja con la misma filosofía que la función reduce

Map, Combiner y Reduce



Ejemplo

- ¿Qué cambios hay que hacer al problema del hipermercado si además se desea buscar el mínimo?
- ¿Y si además queremos el promedio?
- ¿Y si además queremos el desvío estándar

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}$$