Tecnologías de datos masivos

Doble Grado en Ingeniería en Tecnologías de Telecomunicación y Business Analytics

Shuffle

Spark - Shuffle

- Spark mantiene el máximo tiempo posible la mayor cantidad posible de datos en memoria
- Hay determinadas operaciones que requieren datos que datos de distitnas paritciones se junte en una sola para poder operarar
- Estas operaciones llaman al proceso de **Shuffle**
- Cada vez que se llame a una operación que requiera **shuffle** se cambiará de stage
- Este proceso hace que lso datos de una partición dejen de estar en memoria y se escriban a disco

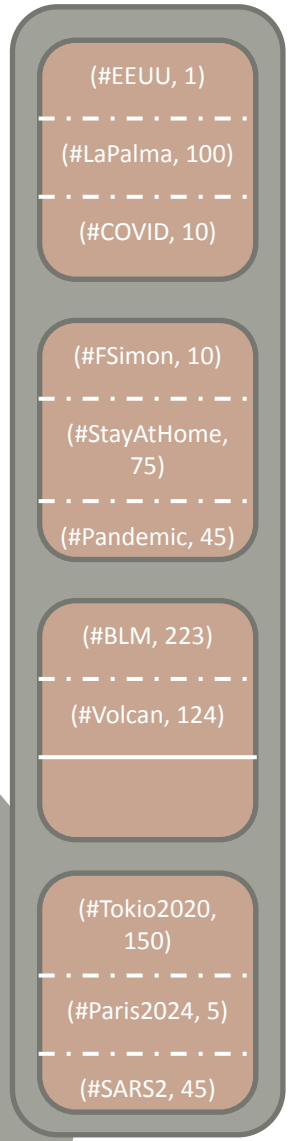
Spark - Shuffle

- Las transformaciones de **shuffle** son las más costosas a nivel computacional que se pueden hacer en Spark
- Se recomienda minimizar tanto el número de transformaciones de **shuffle** como los registros involucrados en la propia transformación
- Toda operación de **shuffle** en Spark requiere un **RDD** con clave valor aunque algunas, como distinct, generan el **RDD** de clave valor en la misma operación

Spark - sortByKey

- Ordena las distintas claves de un **RDD** del tipo clave valor
- A partir de esta orden los resultados están ordenados hasta que se modifique la clave
- Es una función del tipo **wide** por lo que llame a **shuffle** y termina un **stage**
- El **RDD** resultado tendrá el mismo número de registros que el **RDD** entrante
- sortByKey() : RDD[K, V]

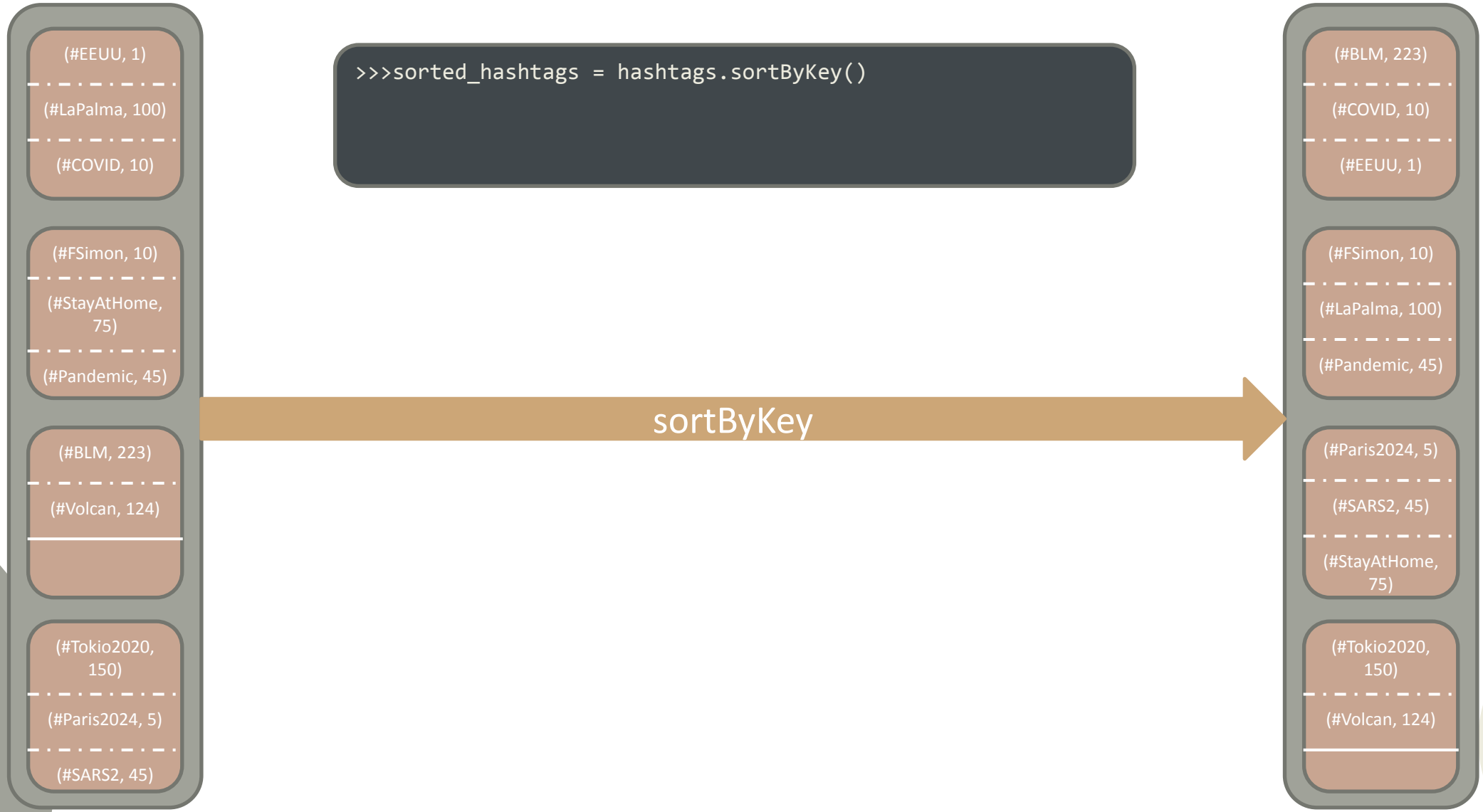
Spark - sortByKey




```
>>>sorted_hashtags = hashtags.sortByKey()
```

Spark - sortByKey

```
>>>sorted_hashtags = hashtags.sortByKey()
```



Spark - sortByKey

```
>>>trendy_topic = hashtags.map(lambda hastag_hits:
(hastag_hits[1],
hastag_hits[0])).sortByKey(ascending=False).map(lambda
hastag_hits: (hastag_hits[1], hastag_hits[0]))
```

(#EEUU, 1)

(#LaPalma, 100)

(#COVID, 10)

(#FSimon, 10)

(#StayAtHome, 75)

(#Pandemic, 45)

(#BLM, 223)

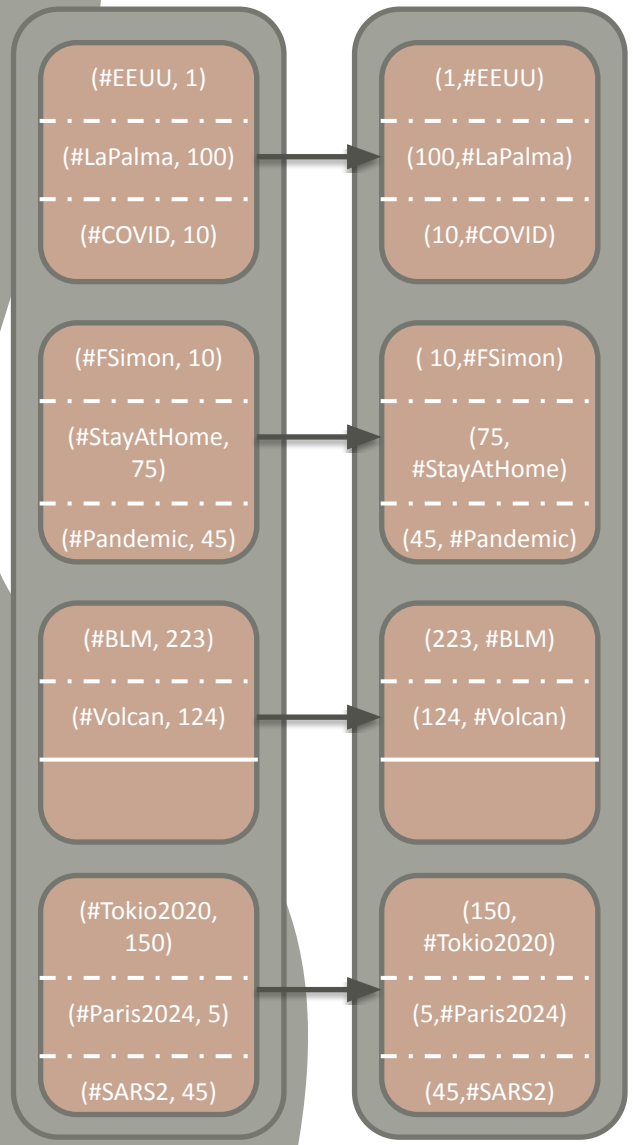
(#Volcan, 124)

(#Tokio2020, 150)

(#Paris2024, 5)

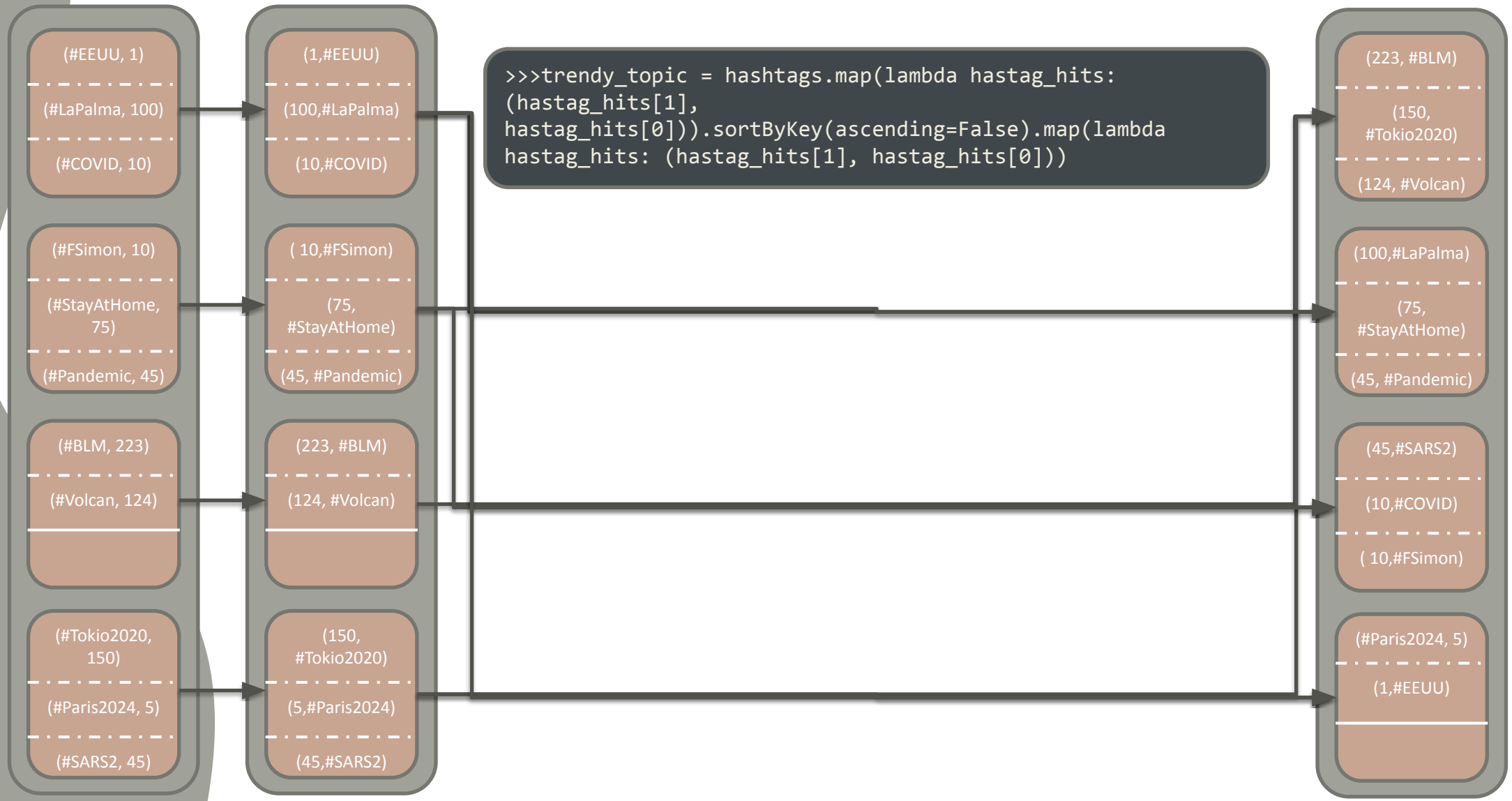
(#SARS2, 45)

Spark - sortByKey



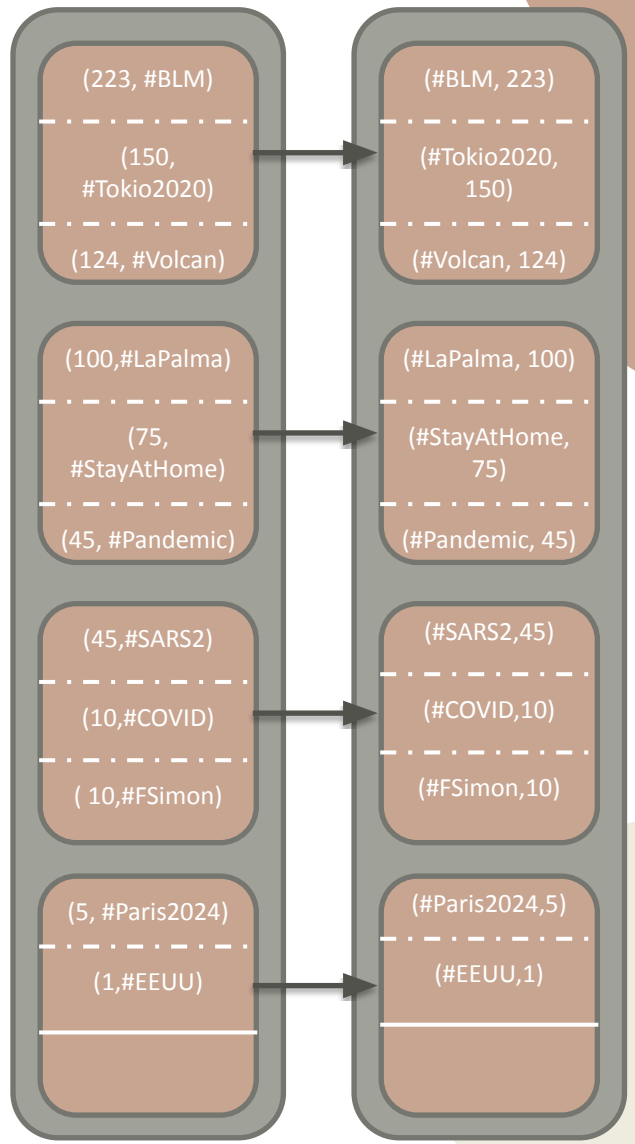

```
>>>trendy_topic = hashtags.map(lambda hastag_hits:
(hastag_hits[1],
hastag_hits[0])).sortByKey(ascending=False).map(lambda
hastag_hits: (hastag_hits[1], hastag_hits[0]))
```

Spark - sortByKey



Spark - sortByKey

```
>>>trendy_topic = hashtags.map(lambda hastag_hits:
(hastag_hits[1],
hastag_hits[0])).sortByKey(ascending=False).map(lambda
hastag_hits: (hastag_hits[1], hastag_hits[0]))
```



Spark - groupByKey

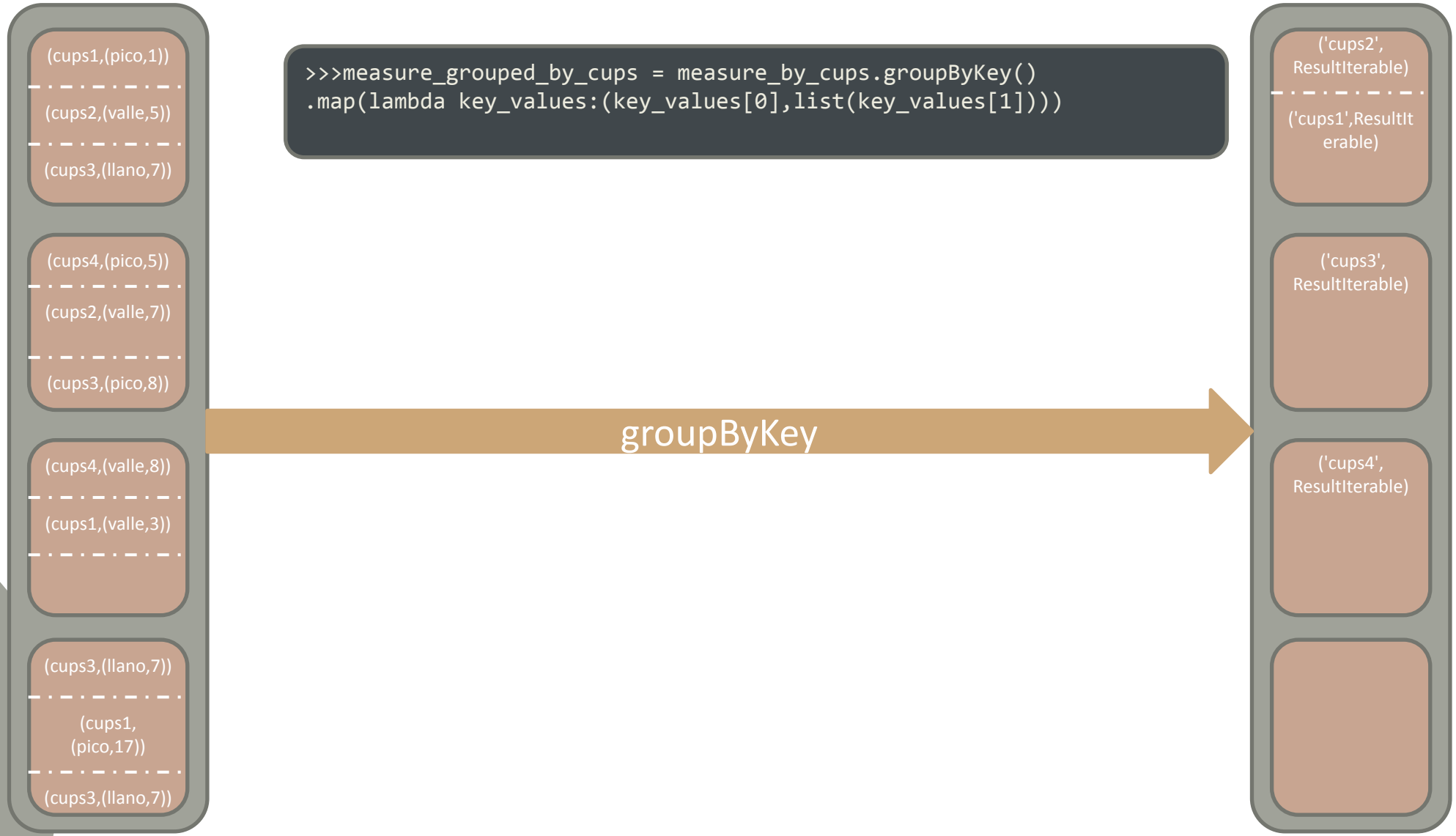
- La función groupByKey agurpa todos los valores de una misma clave en un mismo registro
- Es del tipo **Wide** por lo que acaba un stage y llama a **Shuffle**
- En el **RDD** resultante todos los valores de una clave tienen que caber en una misma particion.
- Es la operación de combinación más ineficiente de Spark, sólo se usa cuando no hay otra opción
- El resultado tendrá tantos registros como claves tiene el **RDD** de entrada
- groupByKey() : RDD[K, List[V]]

Spark - groupByKey

```
>>>measure_grouped_by_cups = measure_by_cups.groupByKey()
.map(lambda key_values:(key_values[0],list(key_values[1])))
```

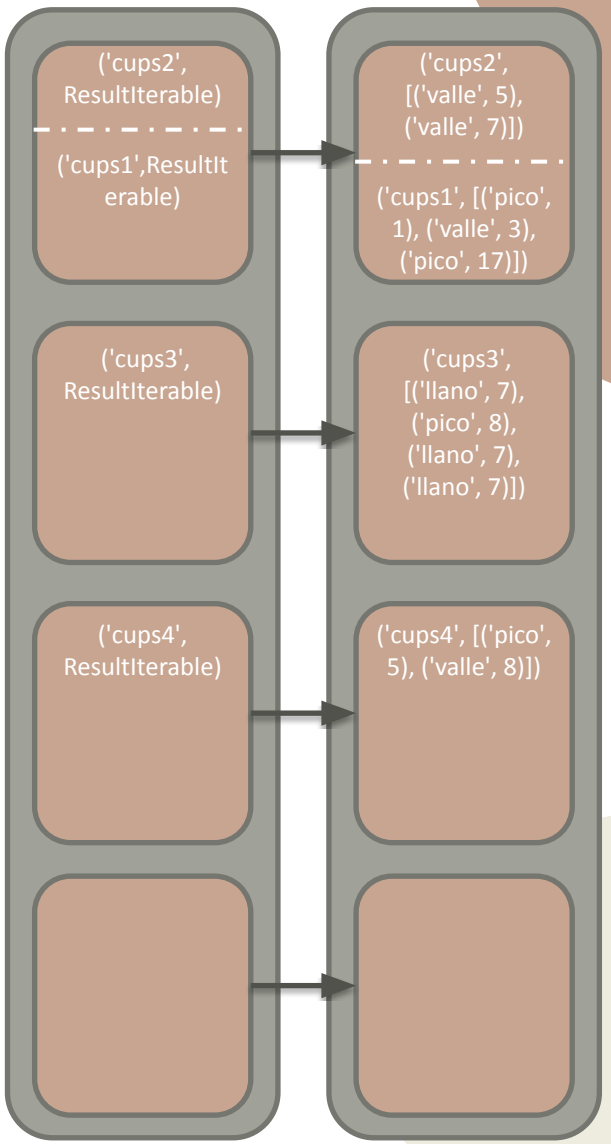
Spark - groupByKey

```
>>>measure_grouped_by_cups = measure_by_cups.groupByKey()
.map(lambda key_values:(key_values[0],list(key_values[1])))
```



Spark - groupByKey

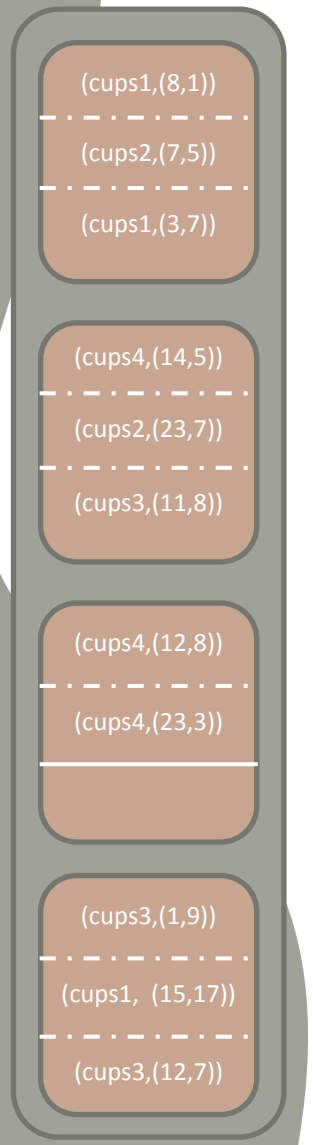
```
>>>measure_grouped_by_cups = measure_by_cups.groupByKey()
.map(lambda key_values:(key_values[0],list(key_values[1])))
```



Spark - reduceByKey

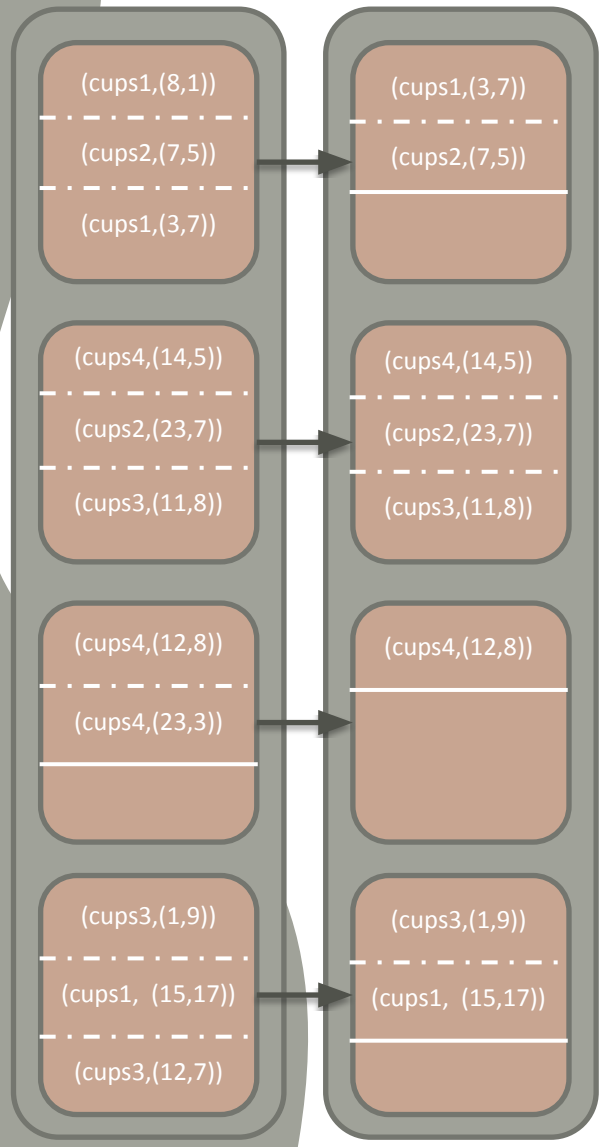
- La función **reduceByKey** agrega los valores de una misma clave.
- Es similar a la operación Reduce de Map&Reduce pero a diferencia del reduce, reduceByKey sólo puede utilizarse en operaciones que sean conmutativas y asociativas
- Es del tipo **Wide** por lo que acaba un stage y llama a Shuffle
- El resultado tendrá tantos registros como claves tiene el **RDD** de entrada
- reduceByKey([U, U] => U) : RDD[K,U]

Spark - reduceByKey




```
>>>def get_max_hour_meassure(measure_hour1, measure_hour2):
    if measure_hour1[1] >= measure_hour2[1]:
      return measure_hour1
    else:
      return measure_hour2

>>>max_measure_and_hour_by_cups = measure_and_hour_by_cups
.reduceByKey(lambda: measure_hour1, measure_hour2:
get_max_hour_meassure(measure_hour1, measure_hour2))
```

Spark - reduceByKey



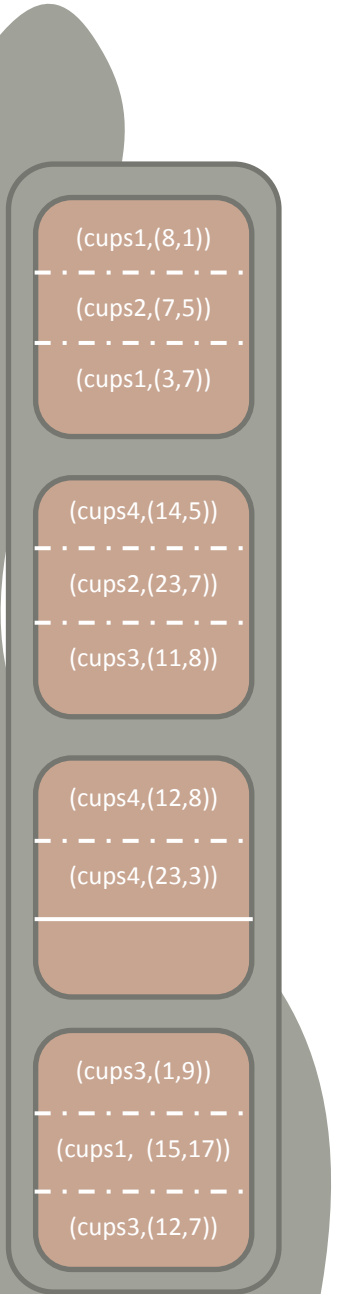

```
>>>def get_max_hour_meassure(measure_hour1, measure_hour2):
    if measure_hour1[1] >= measure_hour2[1]:
      return measure_hour1
    else:
      return measure_hour2

>>>max_measure_and_hour_by_cups = measure_and_hour_by_cups
.reduceByKey(lambda: measure_hour1, measure_hour2:
get_max_hour_meassure(measure_hour1, measure_hour2))
```

Spark - reduceByKey

Spark - reduceByKey

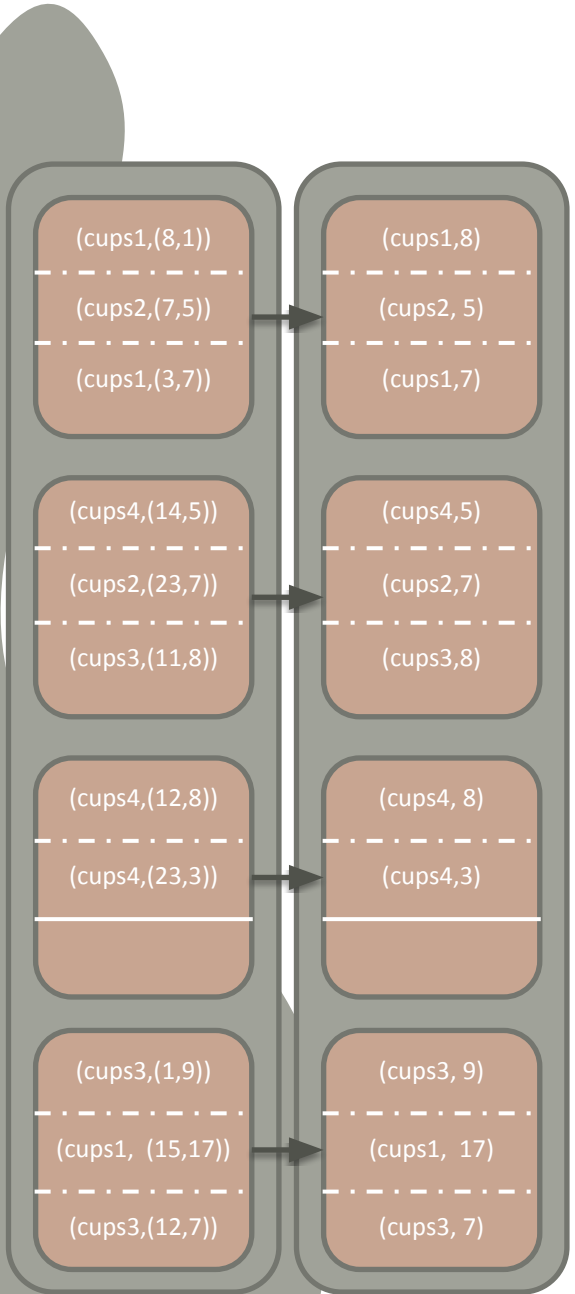
- Si el tipo de datos de la clave no es el que queremos para resolver nuestro problema podemos hacer un map o un flatMap previo
- Aumentan el tiempo de cálculo
- Reducen el número de elementos con los que hacer shuffle
- Siempre es preferible al groupByKey, pero hay casos en los que sólo se puede hacer un groupByKey(operaciones no conmutativas ni asociativas)

Spark - reduceByKey



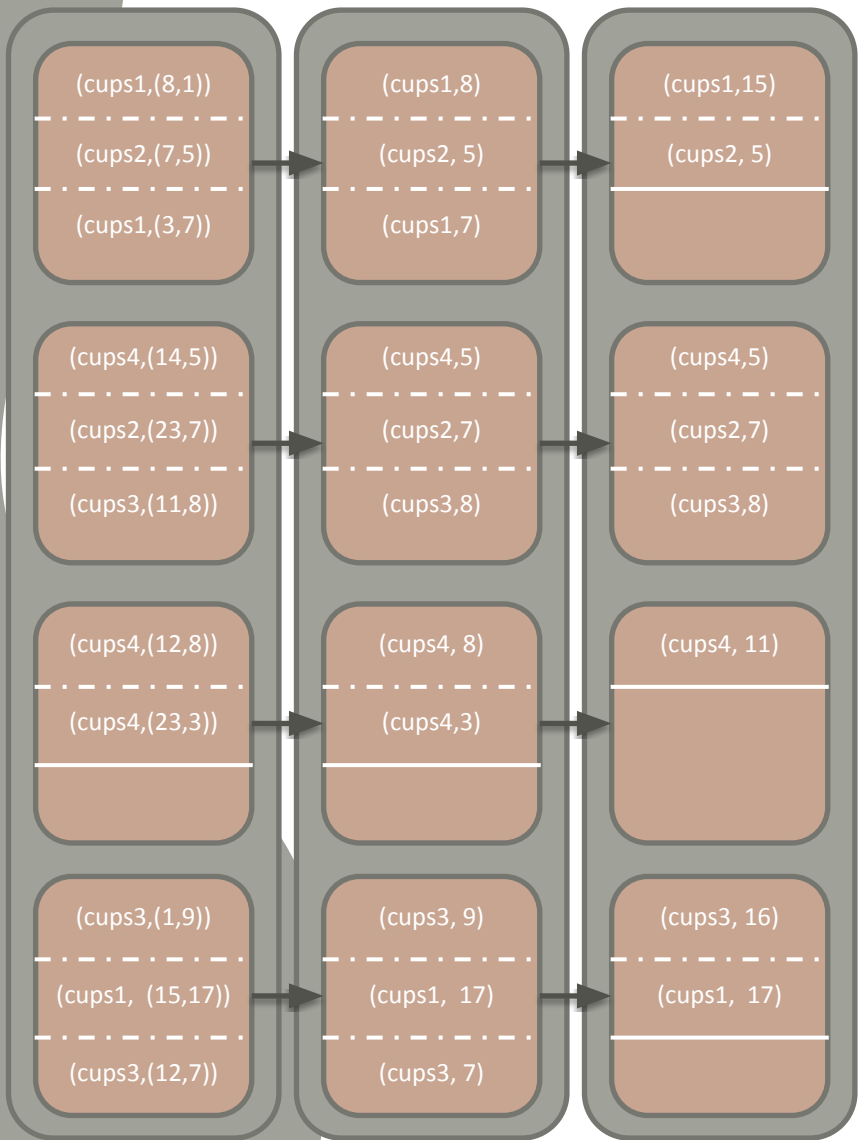

```
>>>amount_of_messurement_by_cups = measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
  (measure_and_hour_by_cups[0], measure_and_hour_by_cups[1][1]))
.reduceByKey(lambda: measure1, measure2: measure1 + measure2)
```

Spark - reduceByKey



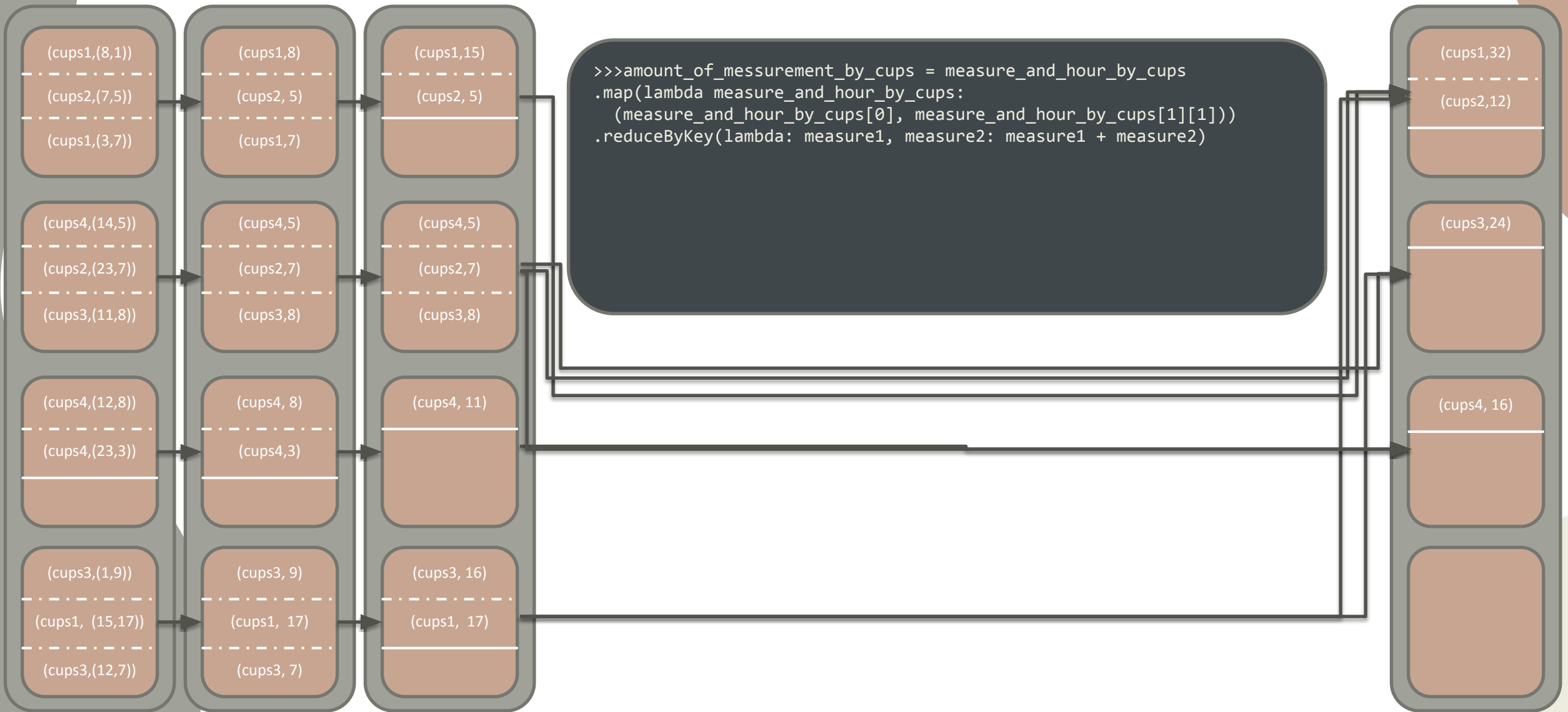

```
>>>amount_of_messurement_by_cups = measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
  (measure_and_hour_by_cups[0], measure_and_hour_by_cups[1][1]))
.reduceByKey(lambda: measure1, measure2: measure1 + measure2)
```

Spark - reduceByKey




```
>>>amount_of_messurement_by_cups = measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
  (measure_and_hour_by_cups[0], measure_and_hour_by_cups[1][1]))
.reduceByKey(lambda: measure1, measure2: measure1 + measure2)
```

Spark - reduceByKey

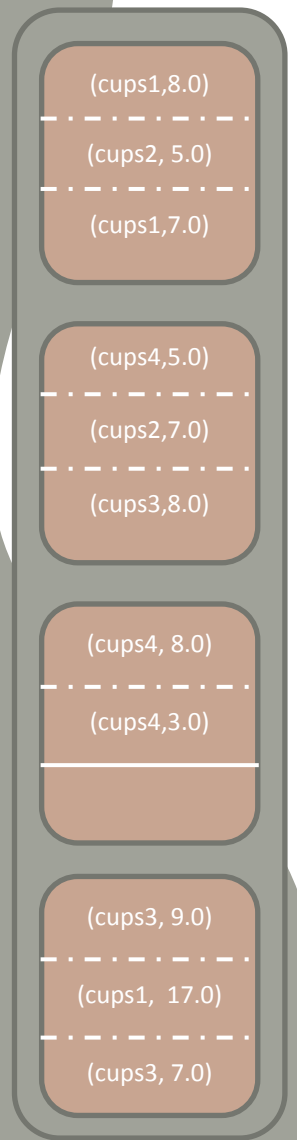


Spark - reduceByKey

- Algunas operaciones no pueden resolverse directamente de manera conmutativa y asociativa

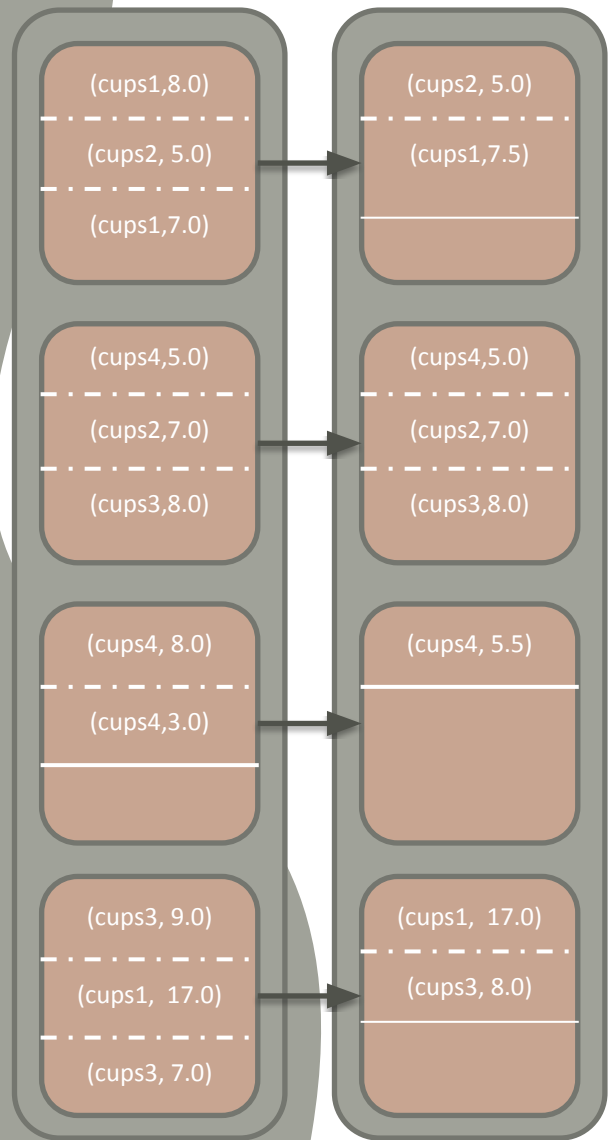
- Cambiando la forma en la que se procesan los datos podemos convertirlo en un problema que se puede resolver de manera conmutativa y asociativa.

Spark - reduceByKey



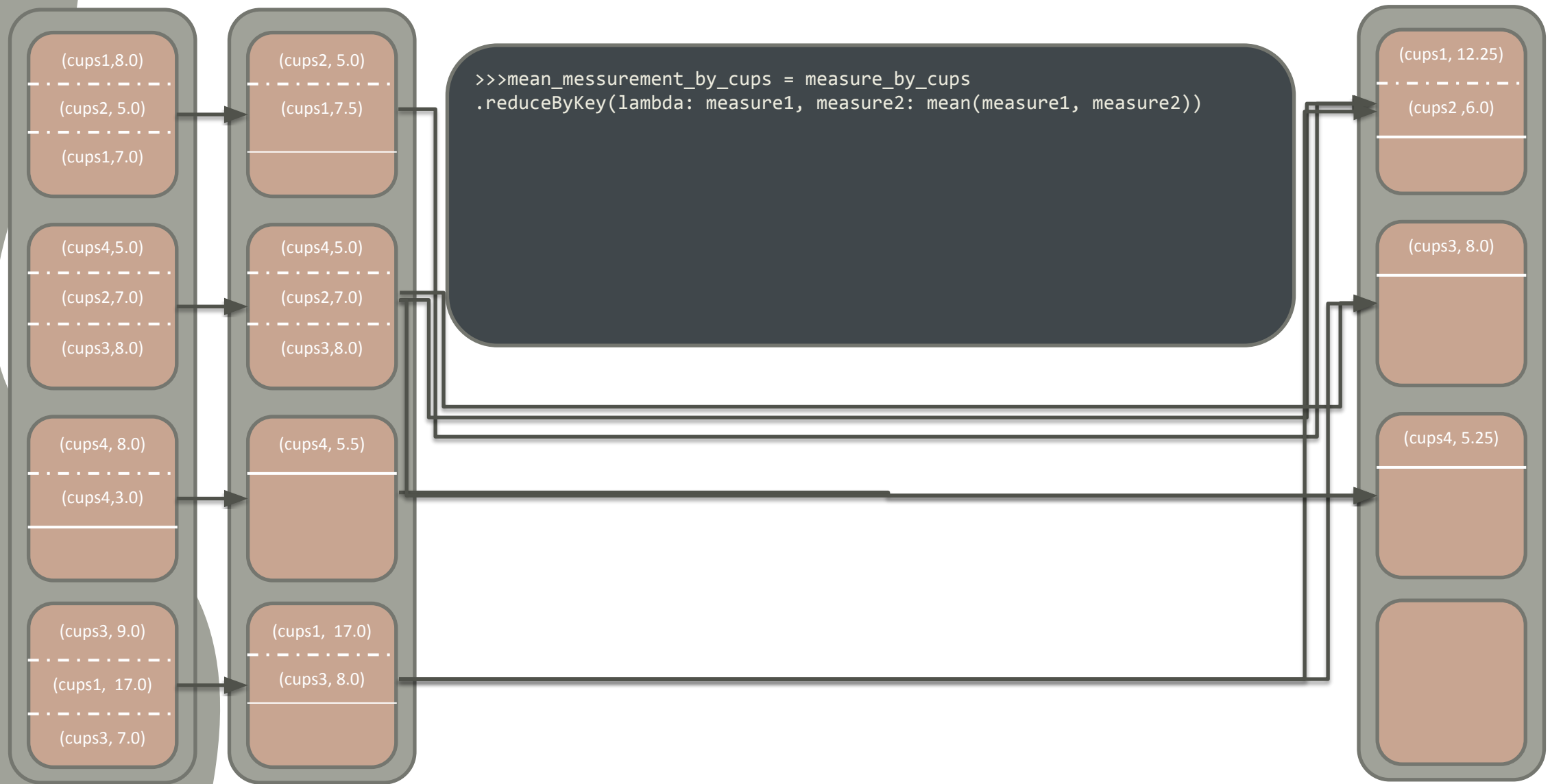

```
>>>mean_messurement_by_cups = measure_by_cups
.reduceByKey(lambda: measure1, measure2: mean(measure1, measure2))
```

Spark - reduceByKey

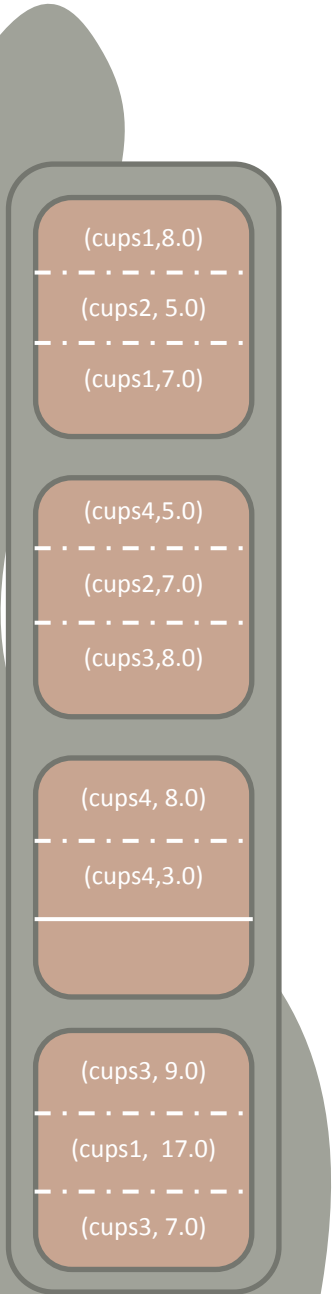



```
>>>mean_messurement_by_cups = measure_by_cups
.reduceByKey(lambda: measure1, measure2: mean(measure1, measure2))
```

Spark - reduceByKey

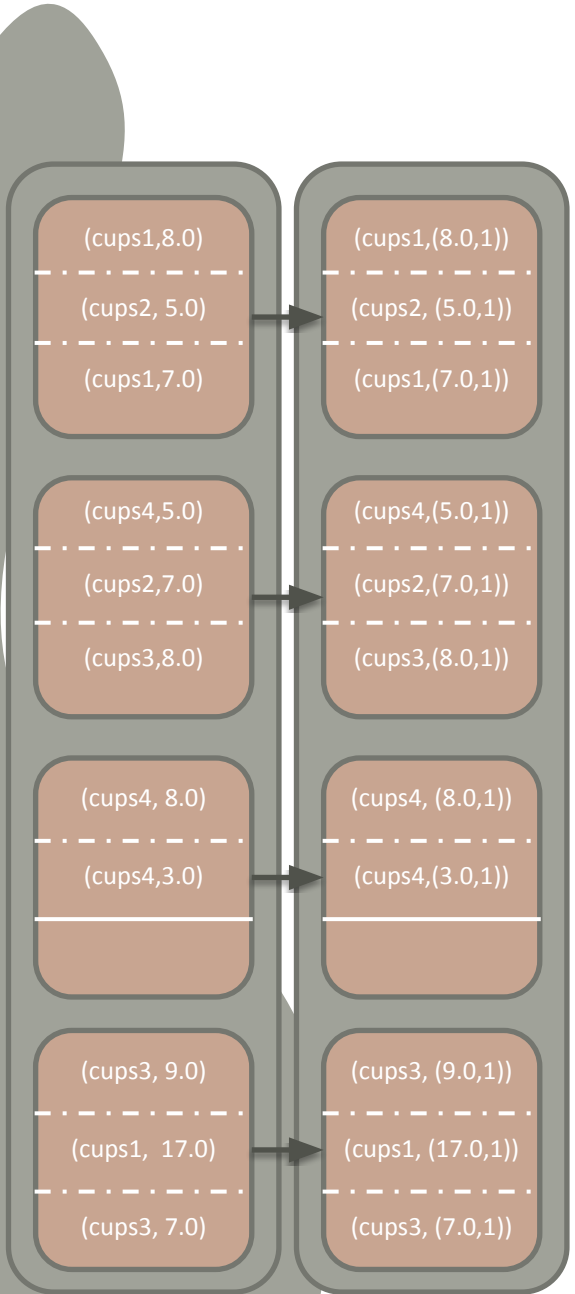


Spark - reduceByKey



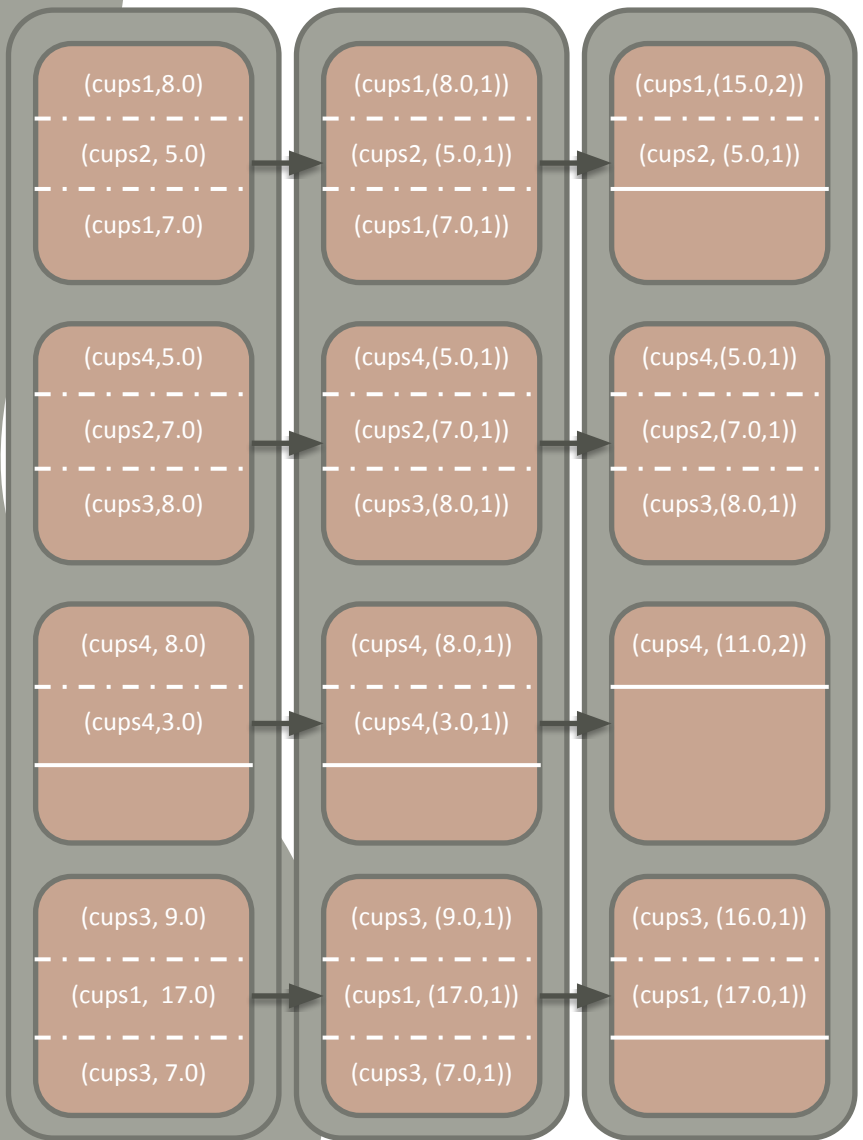

```
>>>amount_of_messurement_by_cups =
measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
   (measure_and_hour_by_cups[0],
    (measure_and_hour_by_cups[1],1)))
.reduceByKey(lambda measure1, measure2:
   (measure1[0]+measure2[0], measure1[1]+measure2[1])
.map(lambda cups_sum_counter:
   (cups_sum_counter[0],
    cups_sum_counter[1][0]/cups_sum_counter[1][1]))
```

Spark - reduceByKey



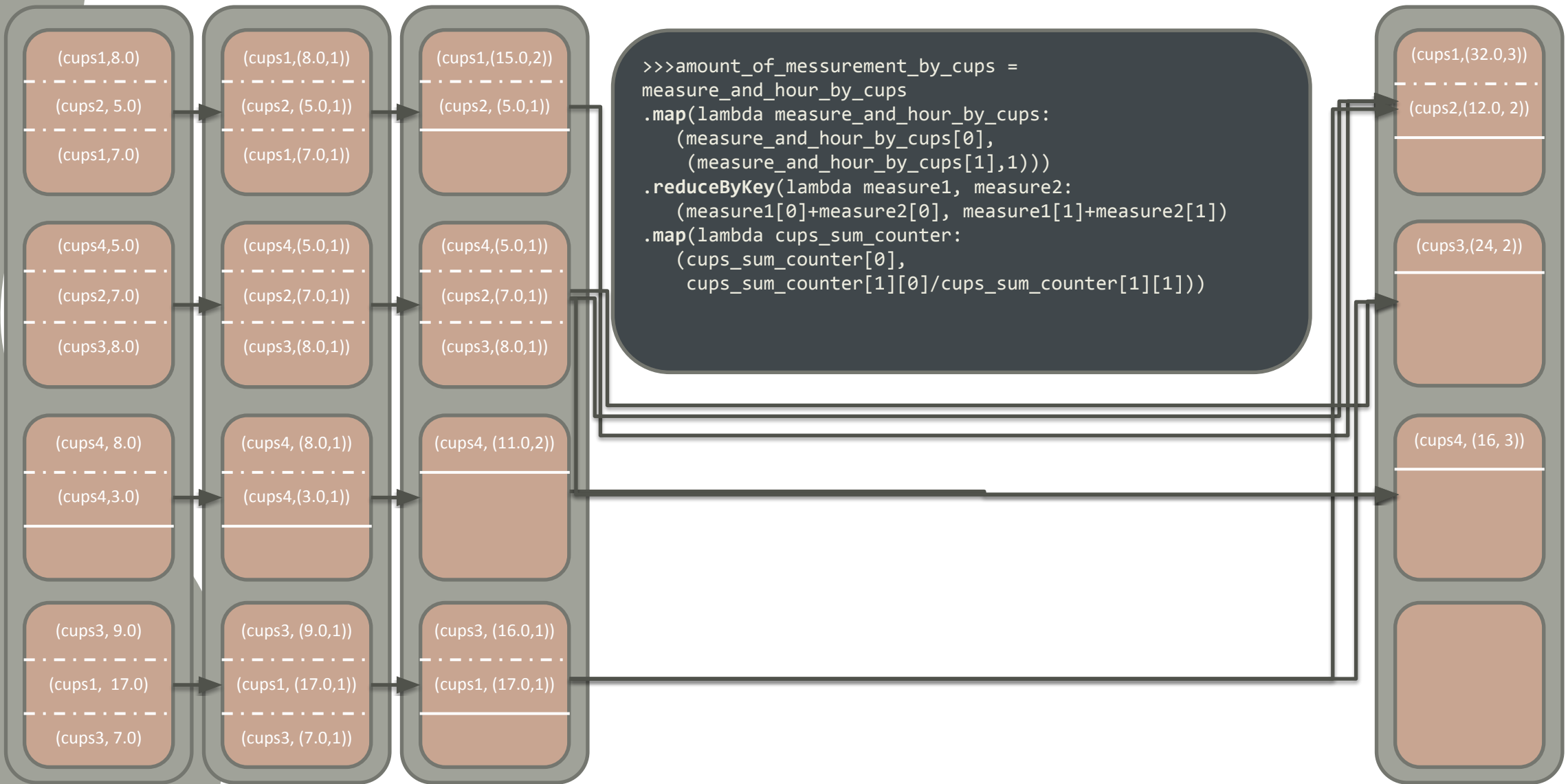

```
>>>amount_of_messurement_by_cups =
measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
   (measure_and_hour_by_cups[0],
    (measure_and_hour_by_cups[1],1)))
.reduceByKey(lambda measure1, measure2:
   (measure1[0]+measure2[0], measure1[1]+measure2[1])
.map(lambda cups_sum_counter:
   (cups_sum_counter[0],
    cups_sum_counter[1][0]/cups_sum_counter[1][1]))
```

Spark - reduceByKey



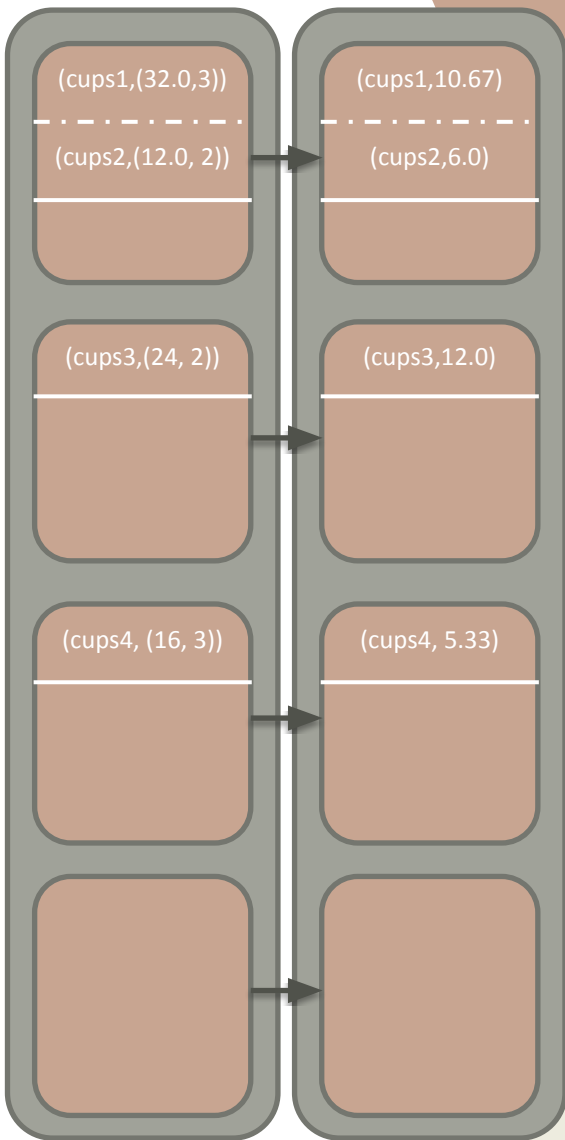

```
>>>amount_of_messurement_by_cups =
measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
   (measure_and_hour_by_cups[0],
    (measure_and_hour_by_cups[1],1)))
.reduceByKey(lambda measure1, measure2:
   (measure1[0]+measure2[0], measure1[1]+measure2[1])
.map(lambda cups_sum_counter:
   (cups_sum_counter[0],
    cups_sum_counter[1][0]/cups_sum_counter[1][1]))
```

Spark - reduceByKey



Spark - reduceByKey

```
>>>amount_of_messurement_by_cups =
measure_and_hour_by_cups
.map(lambda measure_and_hour_by_cups:
   (measure_and_hour_by_cups[0],
    (measure_and_hour_by_cups[1],1)))
.reduceByKey(lambda measure1, measure2:
   (measure1[0]+measure2[0], measure1[1]+measure2[1])
.map(lambda cups_sum_counter:
   (cups_sum_counter[0],
    cups_sum_counter[1][0]/cups_sum_counter[1][1]))
```

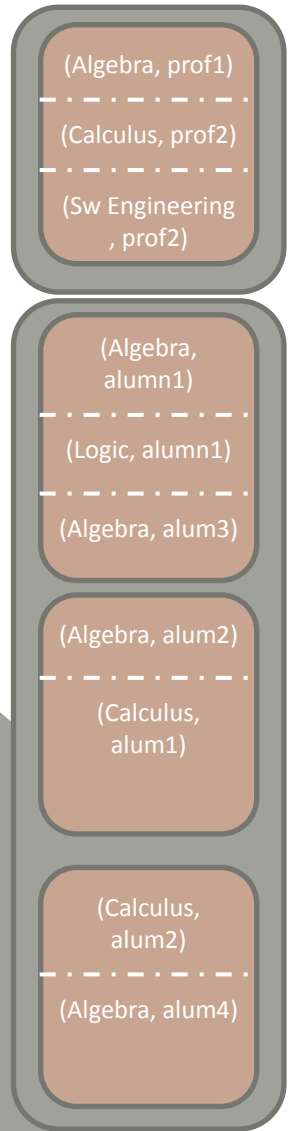


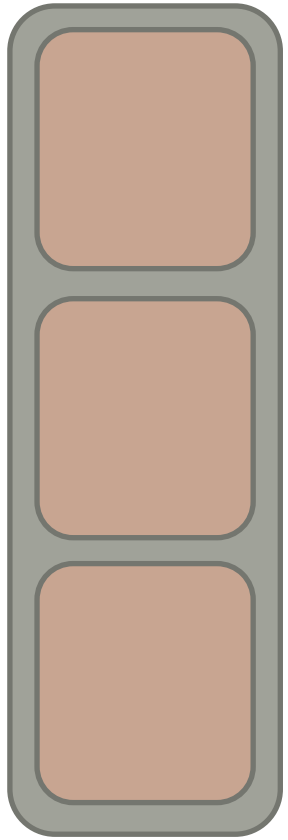
Spark - join

- La función **join** junta los valores de dos **RDDs** con la misma clave.
- El resultado será una tupla en la cual la clave tendrá que ser la misma en ambos **RDD** y el valor una tupla con los valores de ambos **RDD**
- Es del tipo **Wide** por lo que acaba un stage y llama a Shuffle
- El resultado tendrá tantos registros como claves tiene en común ambos **RDDs** de entrada
- join(rdd[K, W]) : RDD[K,[V, W]]

Spark - join

```
>>>teachers_student_by_course =
    teachers_by_course.join(students_by_course)
```





Spark - join

```
>>>teachers_student_by_course =
    teachers_by_course.join(students_by_course)
```

Spark - tipos de joins

- Spark nos deja hacer los distintos tipos de joins de sql

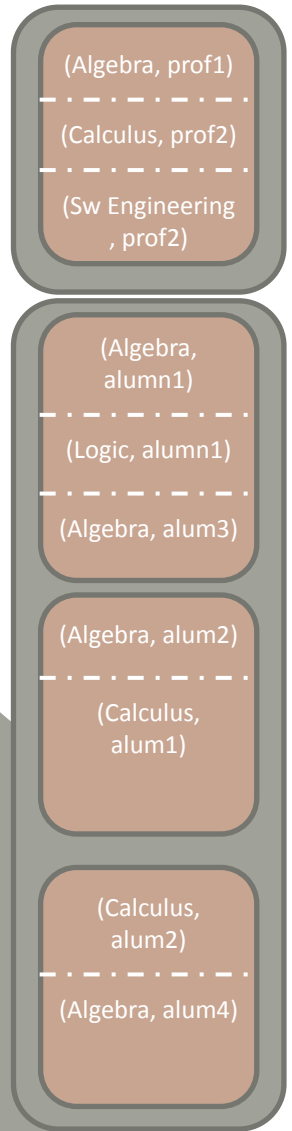
Left Join

Right Join

Inner Join

Full outer Join

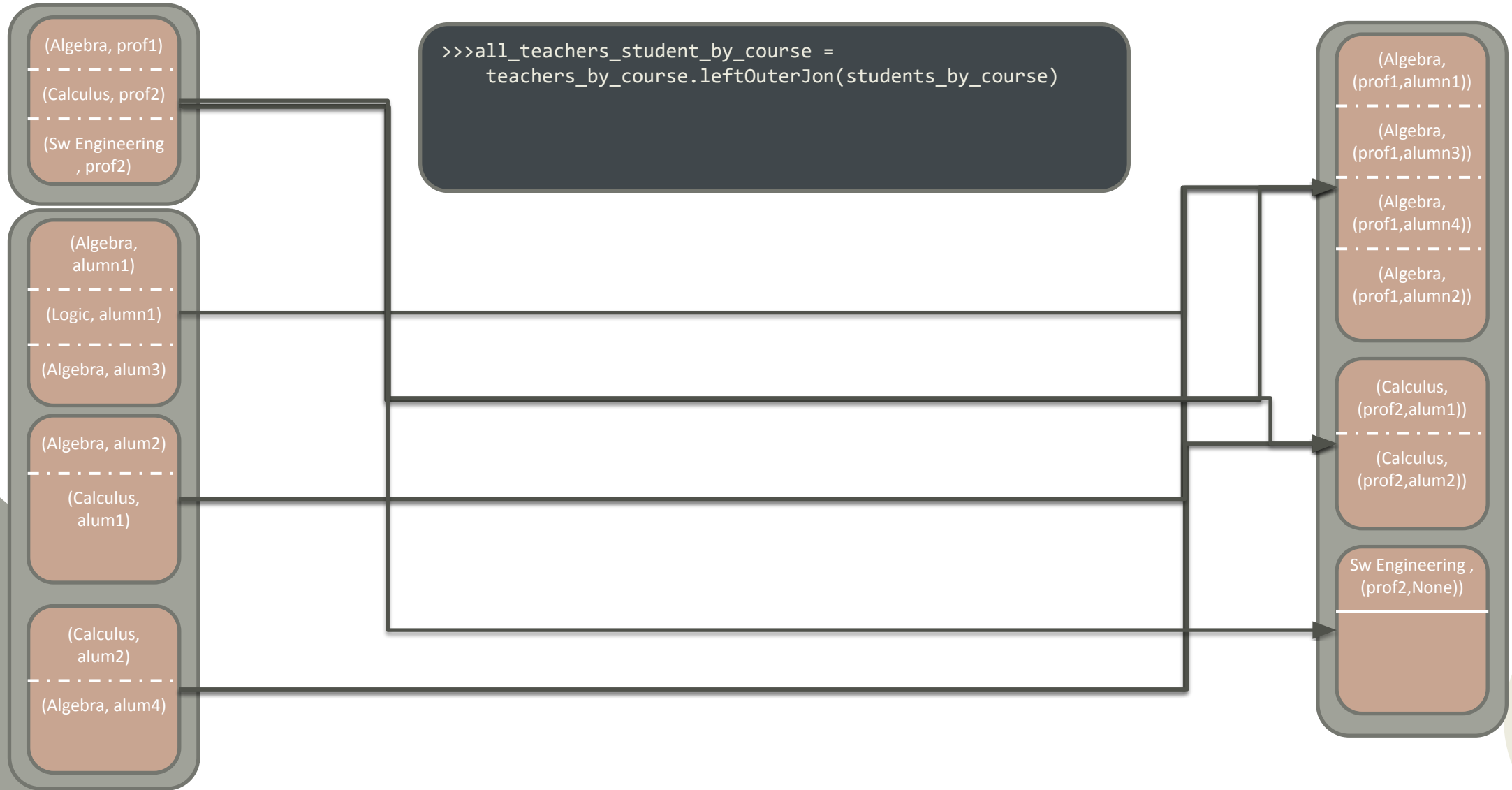
Spark - leftOuterJoin



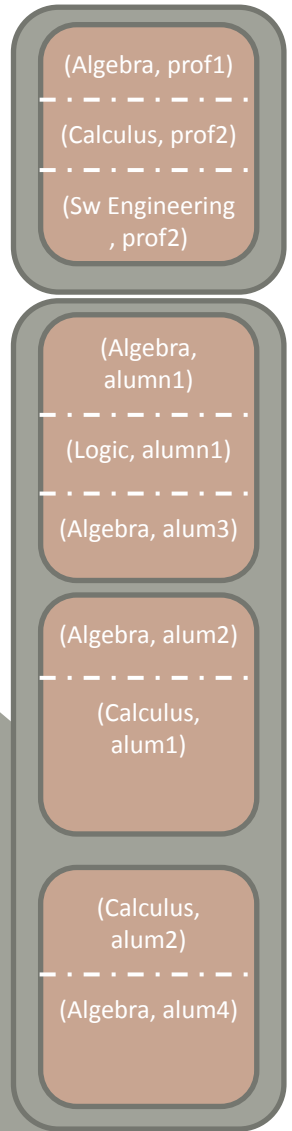

```
>>>all_teachers_student_by_course =
    teachers_by_course.leftOuterJon(students_by_course)
```

Spark - leftOuterJoin

```
>>>all_teachers_student_by_course =
    teachers_by_course.leftOuterJon(students_by_course)
```



Spark - rigthOuterJoin

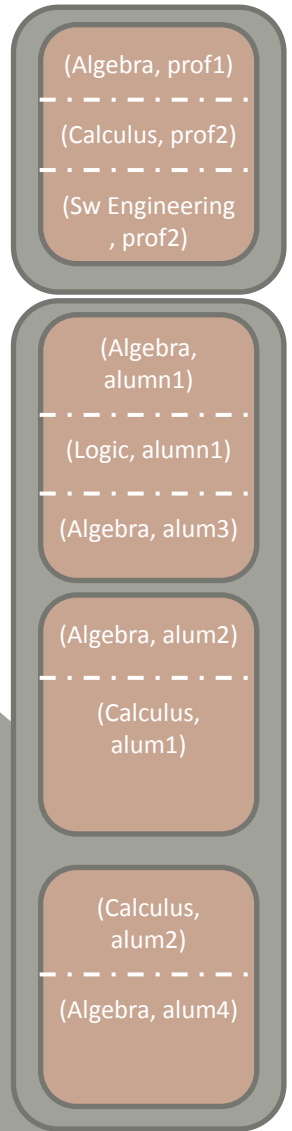



```
>>>teachers_all_student_by_course =
    teachers_by_course.rigthOuterJoin(students_by_course)
```

Spark - rigthOuterJoin

```
>>>teachers_all_student_by_course =
    teachers_by_course.rigthOuterJoin(students_by_course)
```

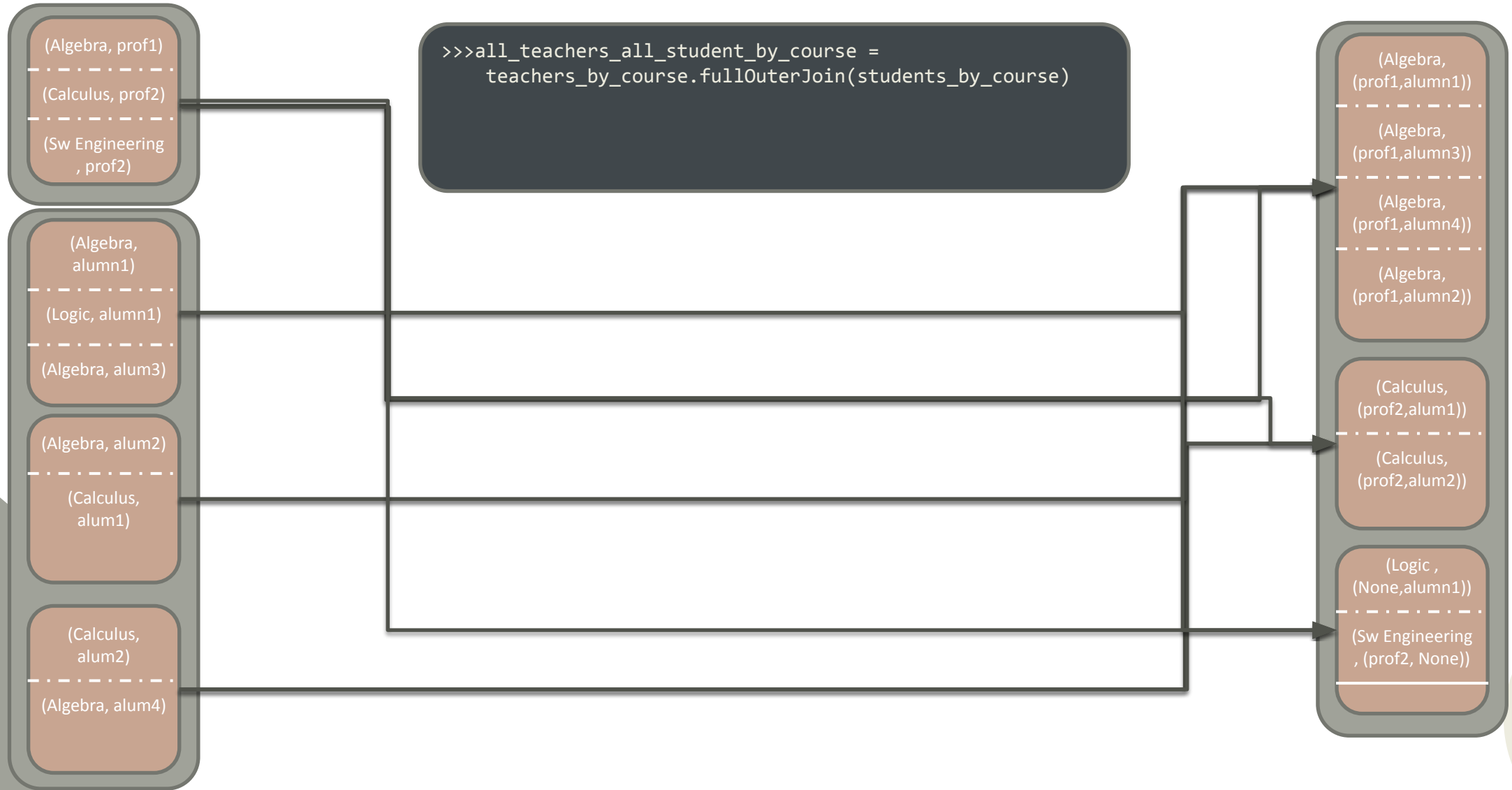
Spark - fullOuterJoin




```
>>>all_teachers_all_student_by_course =
    teachers_by_course.fullOuterJoin(students_by_course)
```

Spark - fullOuterJoin

```
>>>all_teachers_all_student_by_course =
    teachers_by_course.fullOuterJoin(students_by_course)
```



Apache Spark