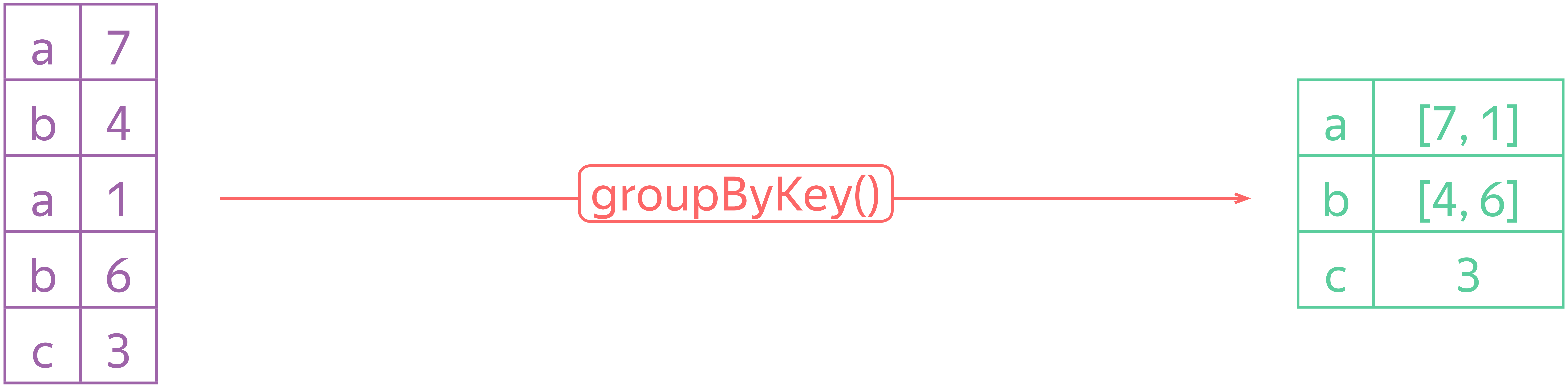
Yandex

Transformations 2

Keyed transformations

- <u>Def</u>: groupByKey*(): RDD[(K, V)]* ➞ *RDD[(K, Array[V])]*
  - groups all values with the same key into the array
  - returns a set of the arrays with corresponding keys



a	7
b	4
a	1
b	6
c	3

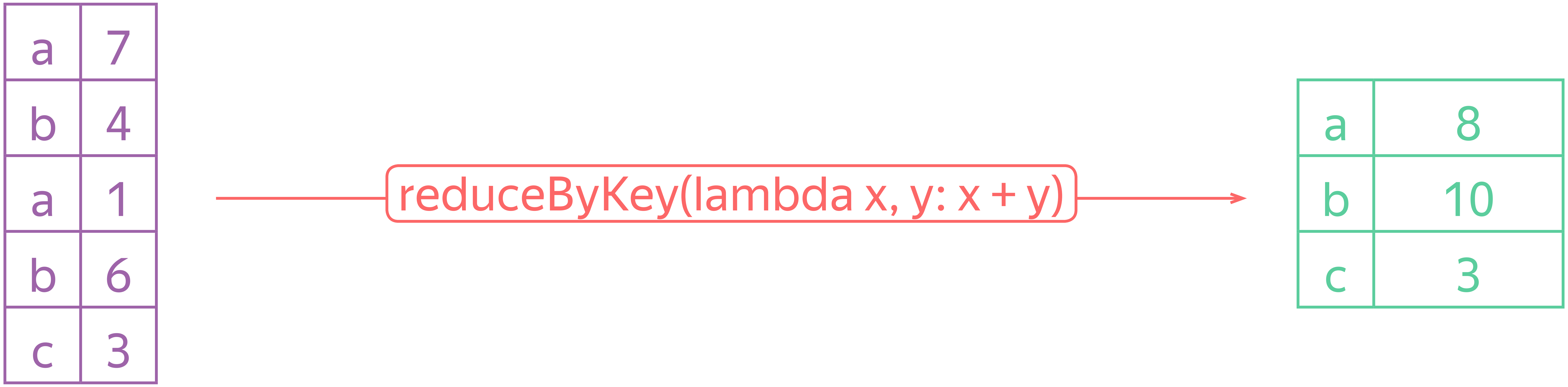
groupByKey()

a	[7, 1]
b	[4, 6]
c	3

Keyed transformations

- Def: groupByKey*(): RDD[(K, V)] → RDD[(K, Array[V])]*
  - groups all values with the same key into the array
  - returns a set of the arrays with corresponding keys

- Def: reduceByKey(f*: (V, V) → V): RDD[(K, V)] → RDD[(K, V)]*
  - folds all values with the same key using the given function f
  - returns a set of the folded values with corresponding keys



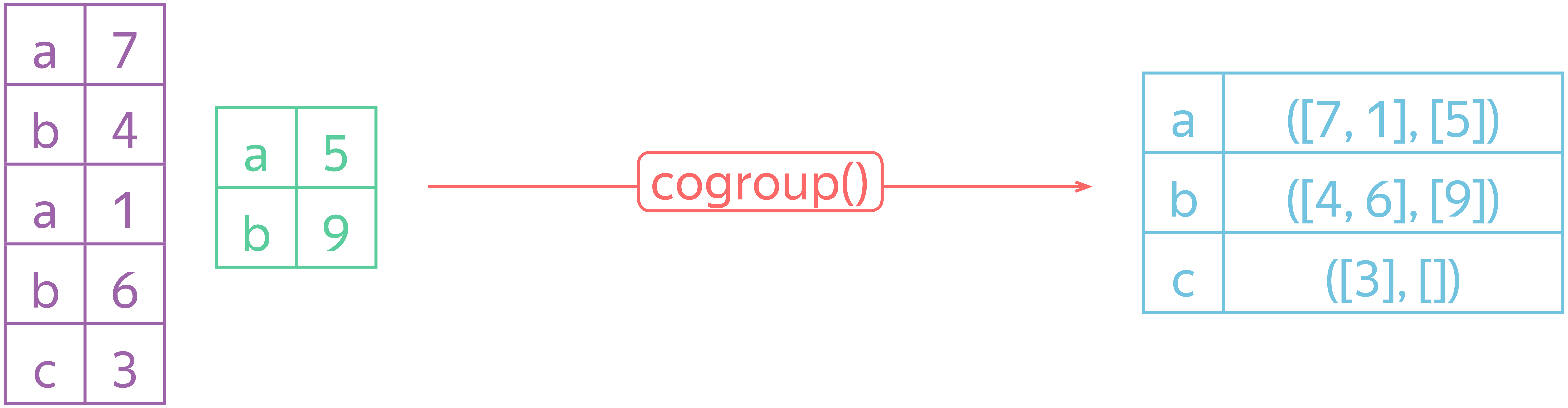
a	7
b	4
a	1
b	6
c	3

reduceByKey(lambda x, y: x + y)

a	8
b	10
c	3

Cogroup transformation

- Def: X.cogroup(*Y: RDD[(K, W)]): RDD[(K, V)] ➞ RDD[(K, (Array[V], Array[W]))]*
  - given two keyed RDDs, groups all values with the same key
  - returns a triple (k, X-values, Y-value) for every key where X-values are all values found under the key k in X and Y-values are similar



a	7
b	4
a	1
b	6
c	3

a	5
b	9

a	([7, 1], [5])
b	([4, 6], [9])
c	([3], [])

Cogroup transformation

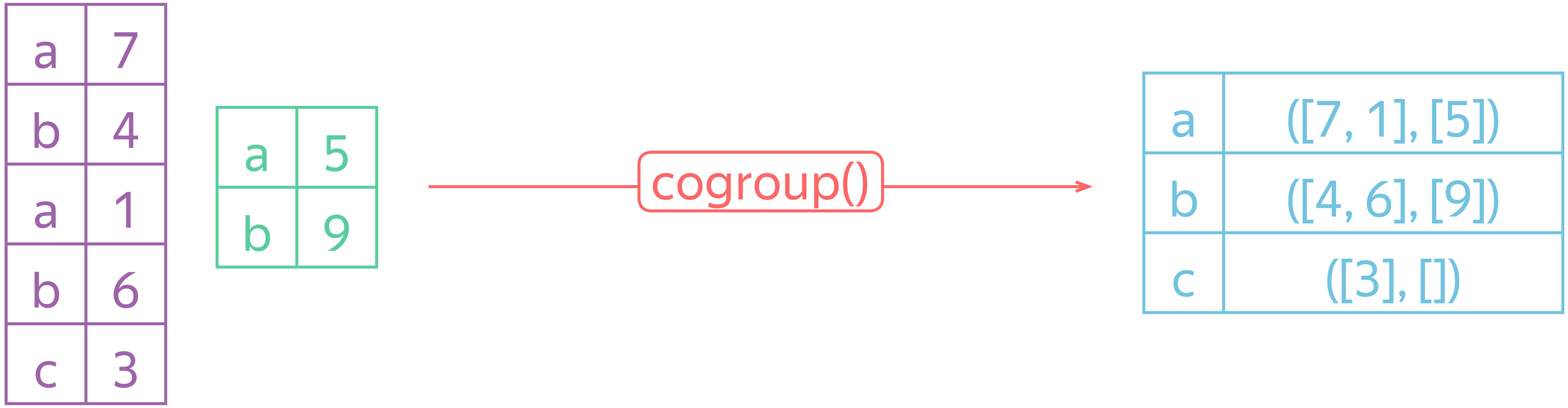
› Def: X.cogroup(Y: … ay[W]))]

› given two keyed … me key

› returns a triple (k … ere X-values are all
values found und … milar

How to compute an inner join
from the result of cogroup?

That is, all triples (k, x, y) where
(k, x) is in X and (k, y) is in Y.



a	7
b	4
a	1
b	6
c	3

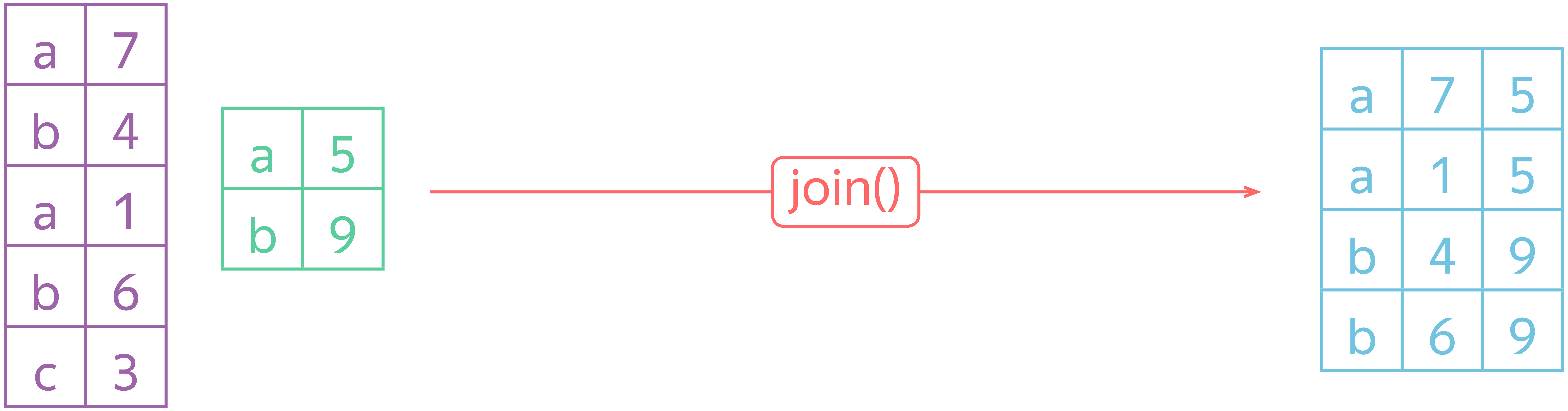
a	5
b	9

a	([7, 1], [5])
b	([4, 6], [9])
c	([3], [])

Joins

- Def: X.join(*Y: RDD[(K, W)]): RDD[(K, V)]* ➞ *RDD[(K, V, W)]*
  - given two keyed RDDs, returns all matching items in two datasets
  - that are triples (k, x, y) where (k, x) is in X and (k, y) is in Y

- Also: X.leftOuterJoin, X.rightOuterJoin, X.fullOuterJoin



a	7
b	4
a	1
b	6
c	3

a	5
b	9

join()

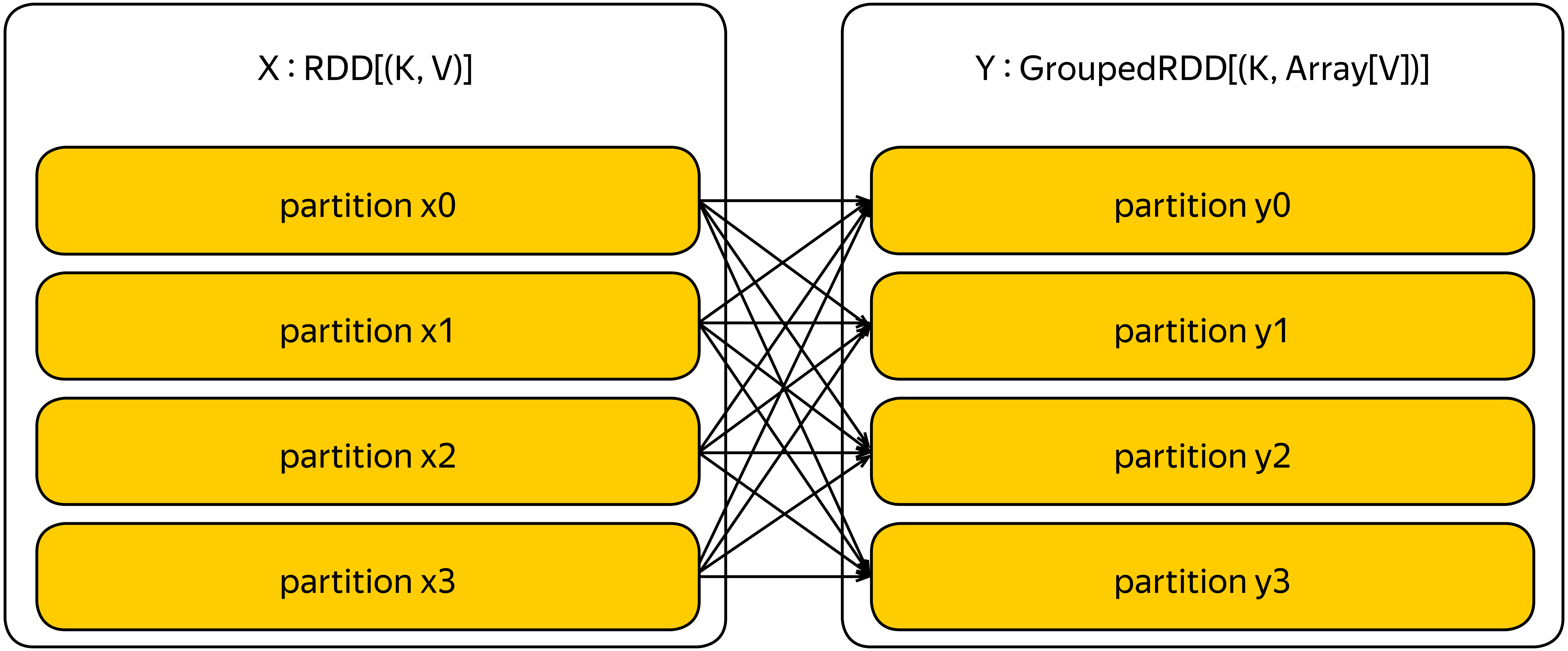
a	7	5
a	1	5
b	4	9
b	6	9

Grouped RDD

- Y = X.groupByKey*(): RDD[(K, V)]* ➞ *RDD[(K, Array[V])]*
  - Y.partitions*()* ➞ *Array[Partition]*
    - returns a set of partitions of the key space
  - Y.iterator*(*p*: Partition*, parents*: Array[Iterator[(K,V)]])*
    ➞ *Iterator[(K, Array[V])]*
    - iterate over every parent partition to select pairs with the key in the partition range, group the pairs by the key – a shuffle operation!
    - return an iterator over the result
  - Y.dependencies*()* ➞ *Array[Dependency]*
    - k-th output partition depends on all input partitions

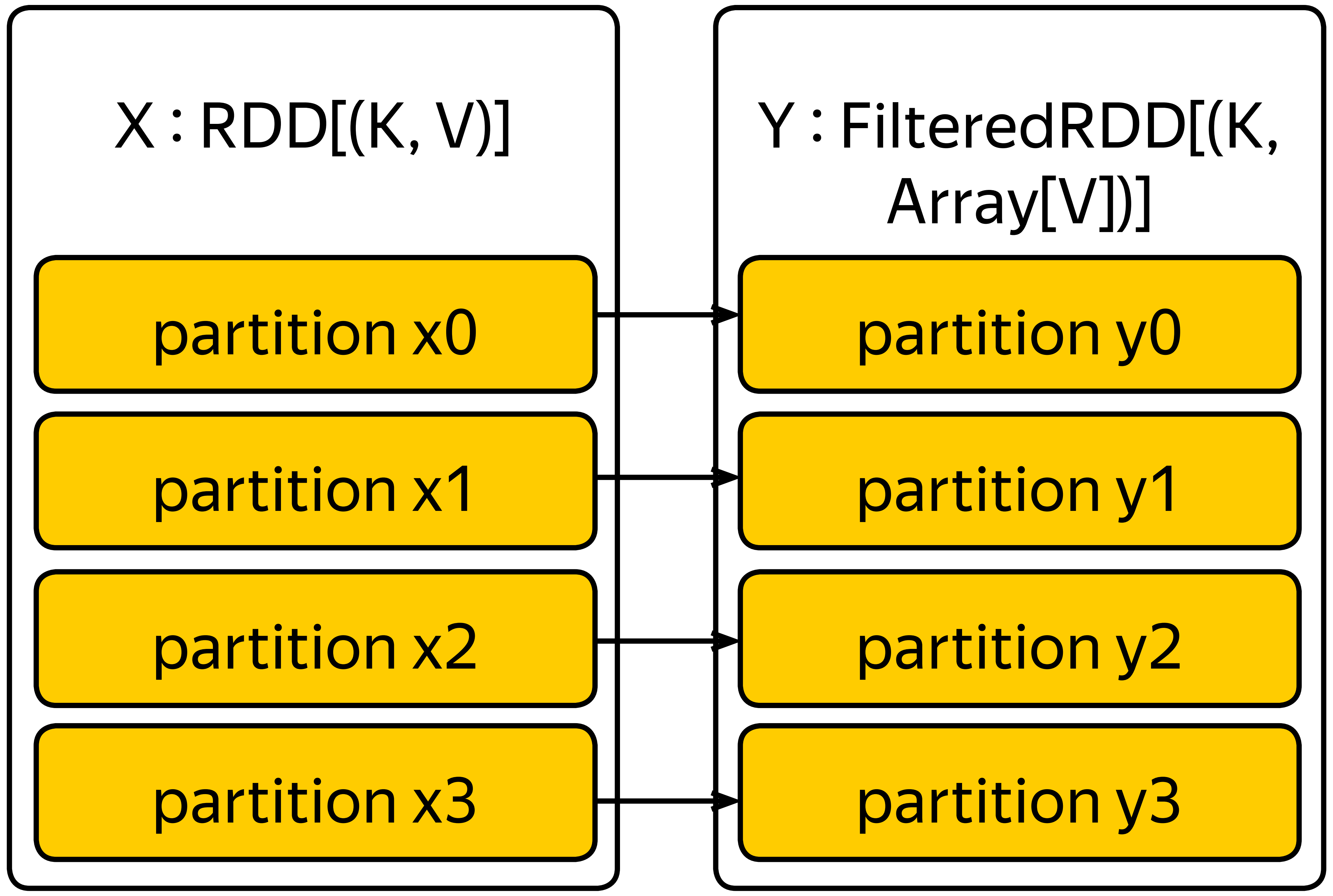
Grouped RDD – Shuffle

- Y = X.groupByKey*(): RDD[(K, V)]* ➡ *RDD[(K, Array[V])]*



Narrow & Wide dependencies

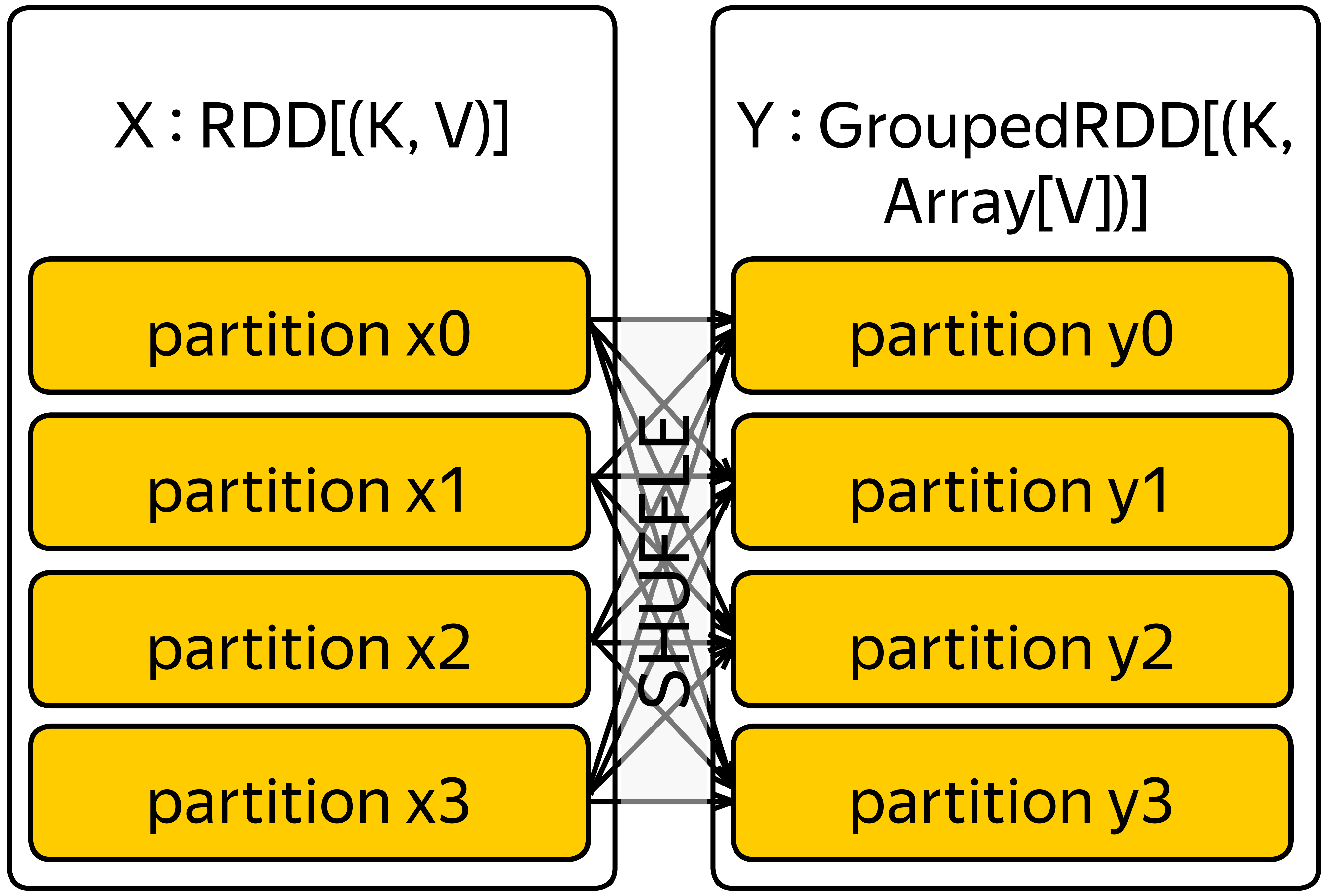
Y = X.filter(p)



Narrow dependencies

at most one child partition for every parent partition

Y = X.groupByKey()



Wide dependencies

more than one child partition for every parent partition

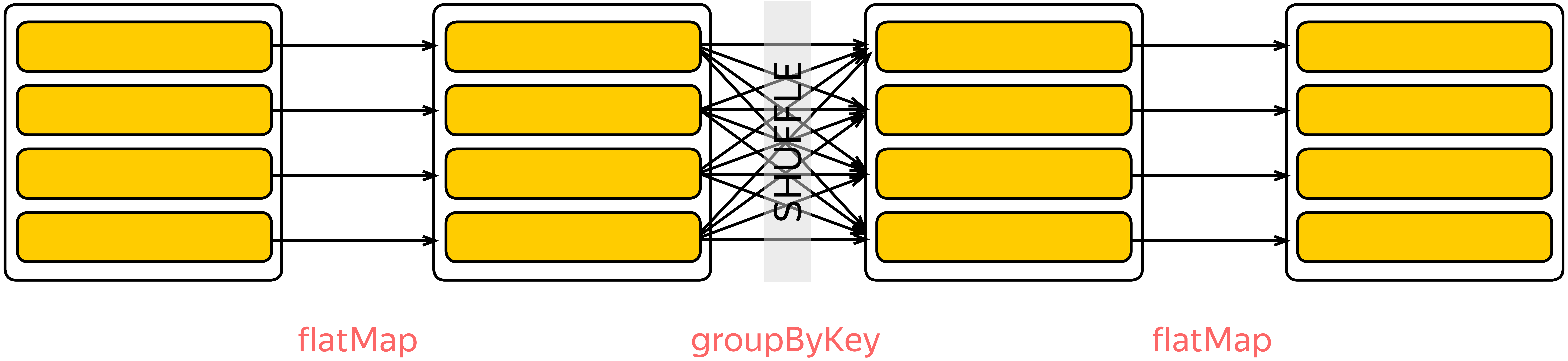
Plenty of transformations!

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- mapValues
- sample
- distinct
- union
- intersection
- groupByKey
- reduceByKey
- aggregateByKey
- sortByKey
- join
- cogroup
- cartesian
- coalesce
- repartition
- ... and others!

MapReduce in Spark

- Example: Y = X.flatMap(*m*).groupByKey().flatMap(r)

X	: *RDD[T]*	
.flatMap(m)	: *RDD[(K, V)]*,	m: *T* ➡ *Array[(K, V)]*
.groupByKey()	: *RDD[(K, Array[V])]*	
.flatMap(r)	: *RDD[U]*,	r: *(K, Array[V])* ➡ *Array[U]*



Quiz

Summary

- Transformation
  - is a description of how to obtain a new RDD from existing RDDs
  - is the primary way to “modify” data (given that RDDs are immutable)

- Transformations are lazy, i.e. no work is done until data is explicitly requested (next video!)

- There are transformations with narrow and wide dependencies

- MapReduce can be expressed with a couple of transformations

- Complex transformations (like joins, cogroup) are available

BigDATAteam