Key-Value RDD

Transformations on Pair RDDs

keys()

Returns an RDD with the keys of each tuple.

```
>>> var m = sc.parallelize(List((1, 2), (3, 4))).keys
>>> m.collect()
Array[Int] = Array(1, 3)
```

Transformations on Pair RDDs

values()

Return an RDD with the values of each tuple.

```
>>> var m = sc.parallelize(List((1, 2), (3, 4))).values
>>> m.collect()
Array(2, 4)
```

Transformations on Pair RDDs

groupByKey()

Group values with the same key.

```
var rdd = sc.parallelize(List((1, 2), (3, 4), (3, 6)));
var rdd1 = rdd.groupByKey()
var vals = rdd1.collect()
for( i <- vals){
    for (k <- i.productIterator) {
        println("\t" + k);
    }
}
```

Questions - Set Operations

What will be the result of the following?

```
var rdd = sc.parallelize(Array(("a", 1), ("b", 1), ("a", 1)));
rdd.groupByKey().mapValues(_.size).collect()
```

Transformations on Pair RDDs

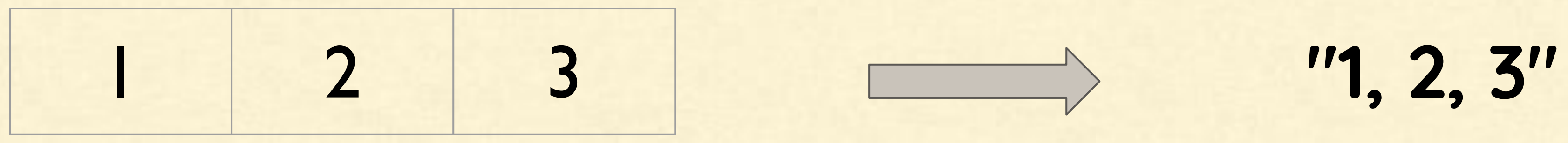
combineByKey(createCombiner, mergeValue, mergeCombiners, numPartitions=None)

Combine values with the same key using a different result type.
Turns RDD[(K, V)] into a result of type RDD[(K, C)]

createCombiner, which turns a V into a C (e.g., creates a one-element list)
mergeValue, to merge a V into a C (e.g., adds it to the end of a list)
mergeCombiners, to combine two C's into a single one.

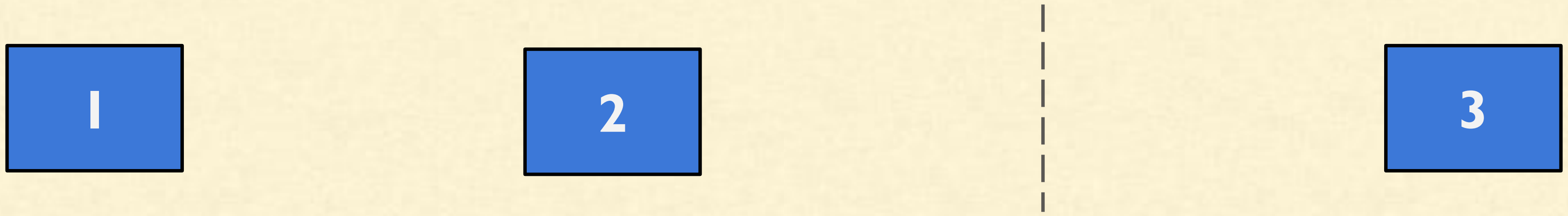
```
var myrdd = sc.parallelize(List(1,2,3,4,5)).map(("x", _))
def cc(x:Int):String = x.toString
def mv(x:String, y:Int):String = {x + "," + y}
def mc(x:String, y:String):String = {x + ", " + y}
myrdd1.combineByKey(cc, mv, mc).collect()

Array((x,1, 2, 3, 4,5))
```

Example: combineByKey



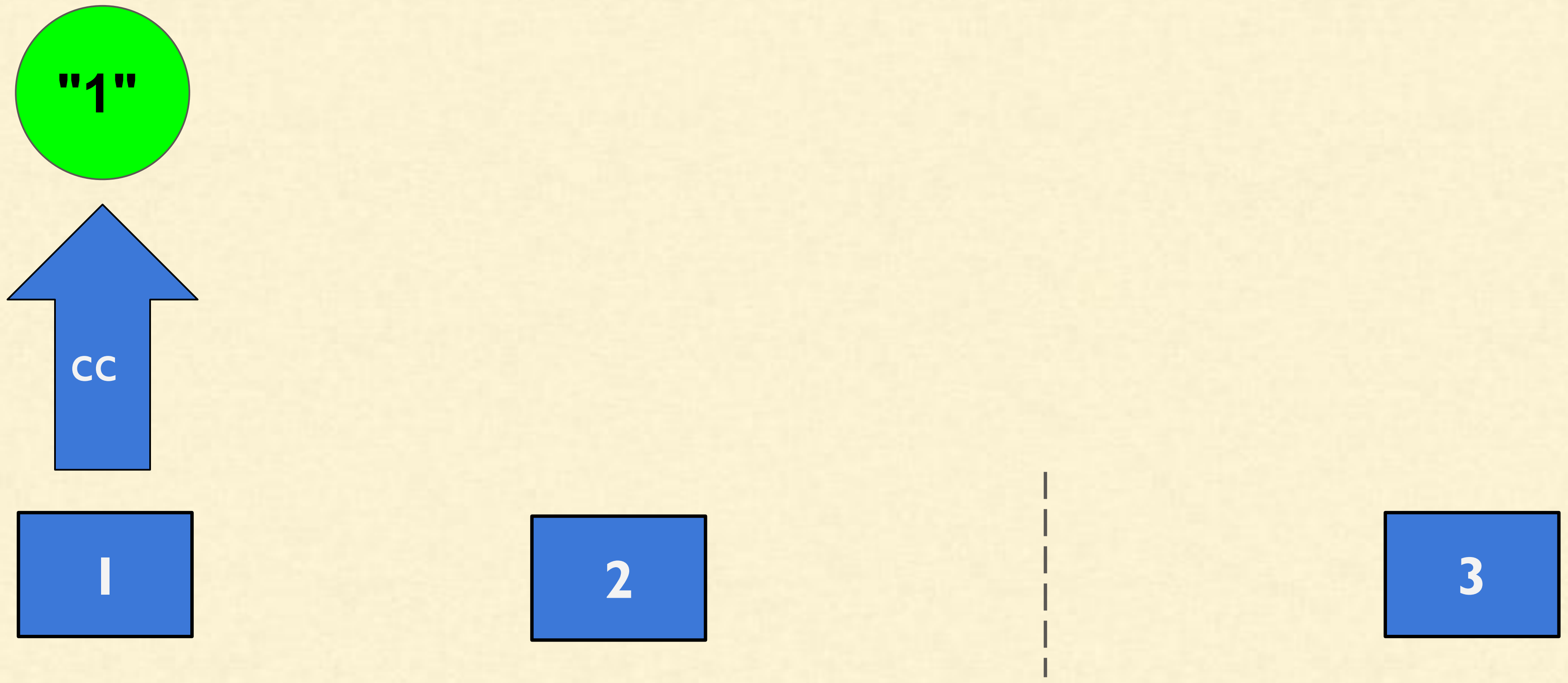
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
```

Example: combineByKey



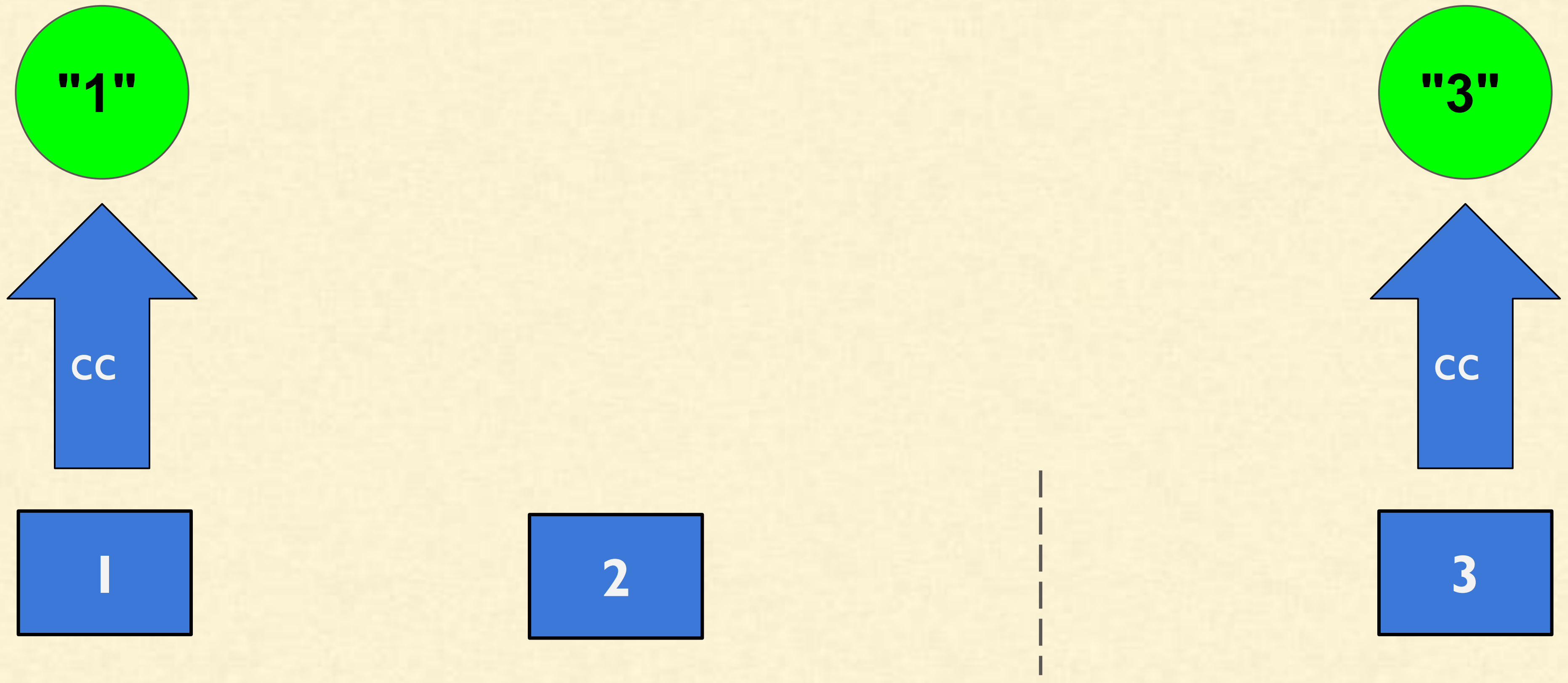
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
```

Example: combineByKey



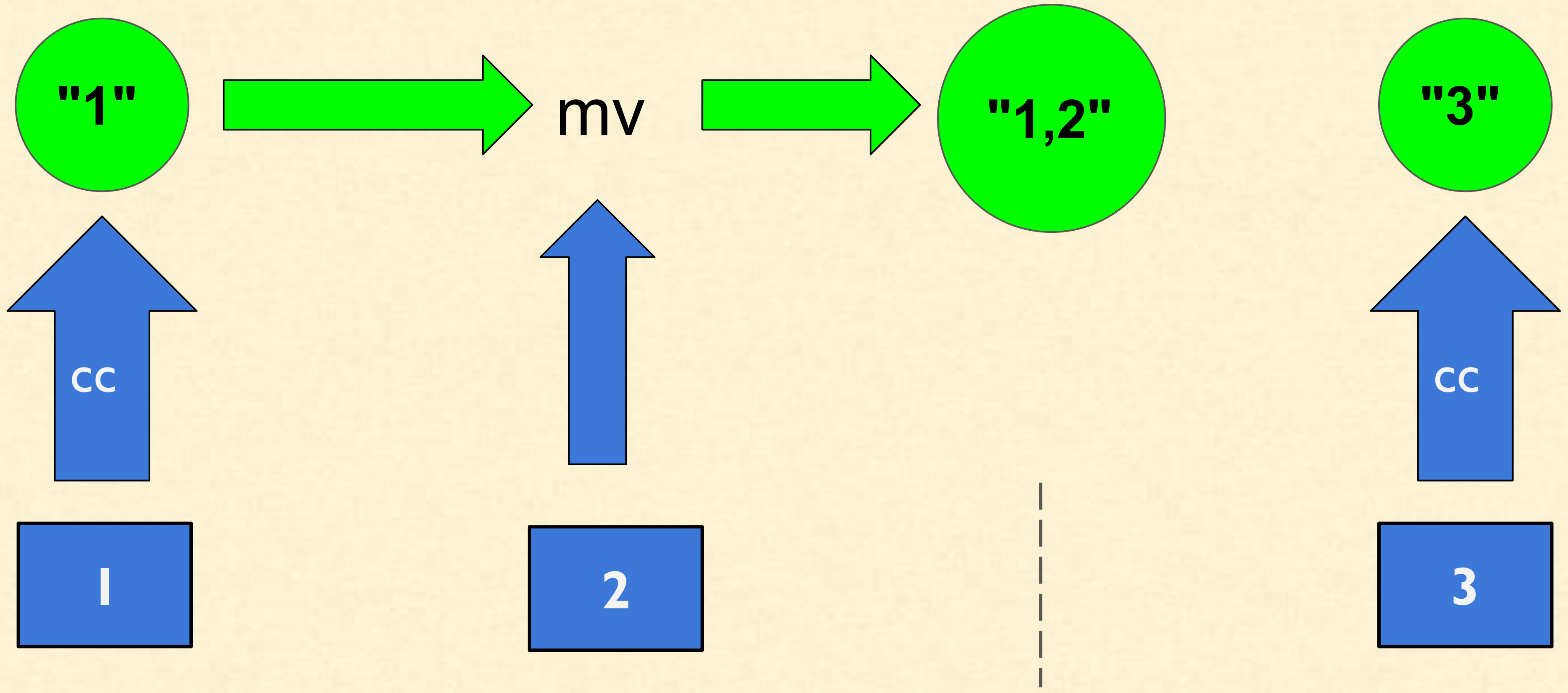
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
```

Example: combineByKey



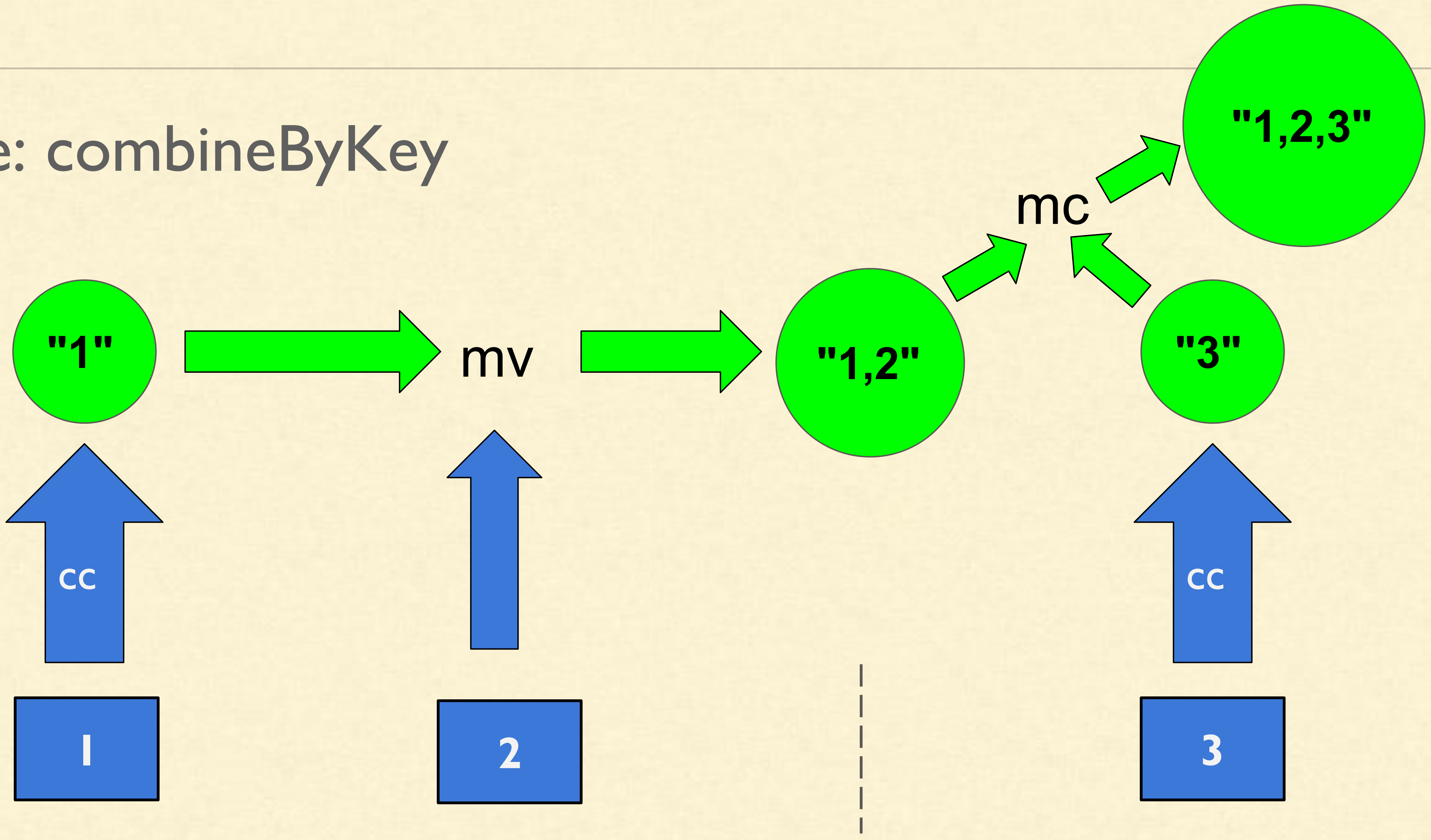
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
```

Example: combineByKey



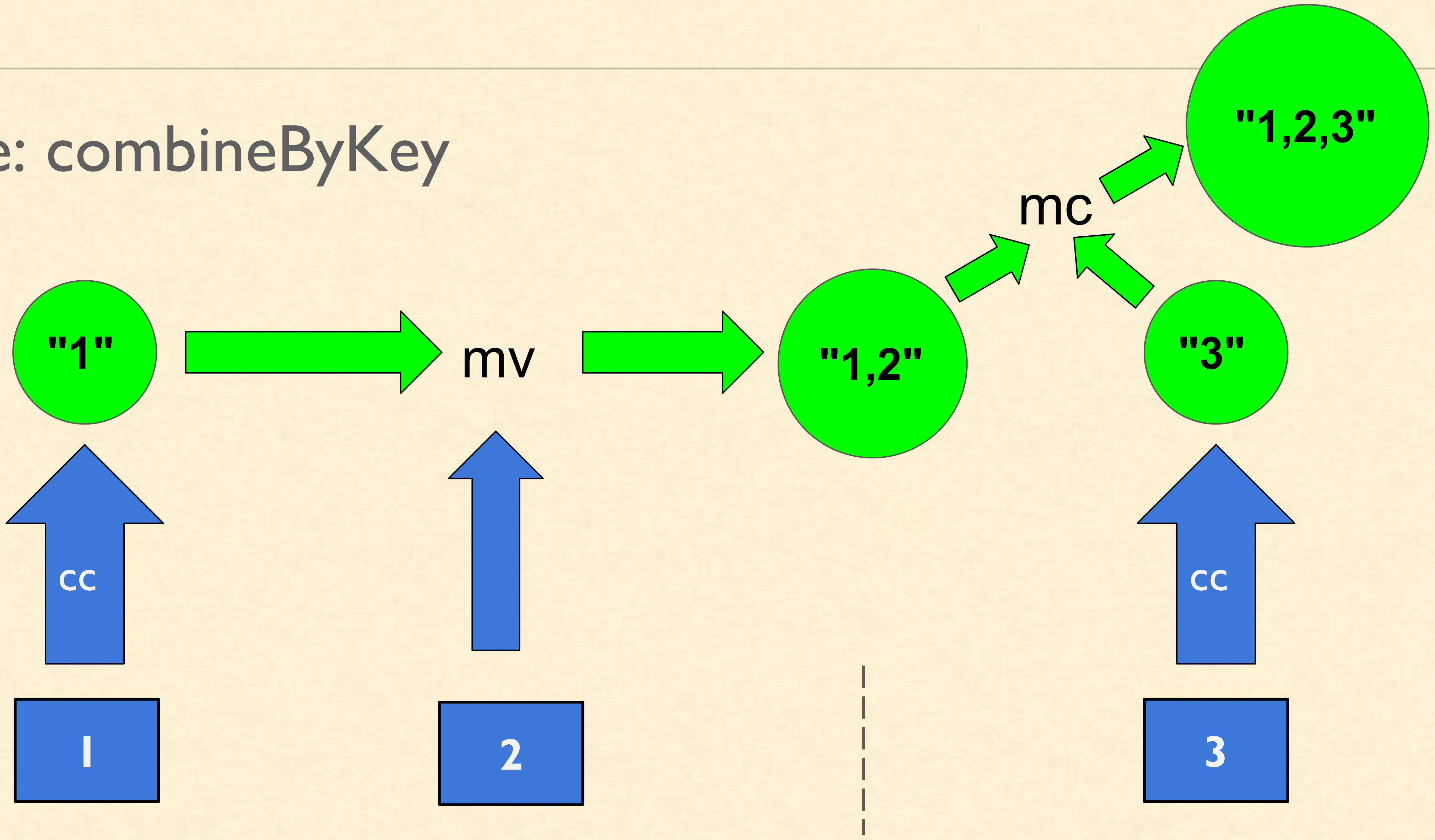
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
def mv(x:String, y:Int):String = {x + "," + y.toString}
```

Example: combineByKey



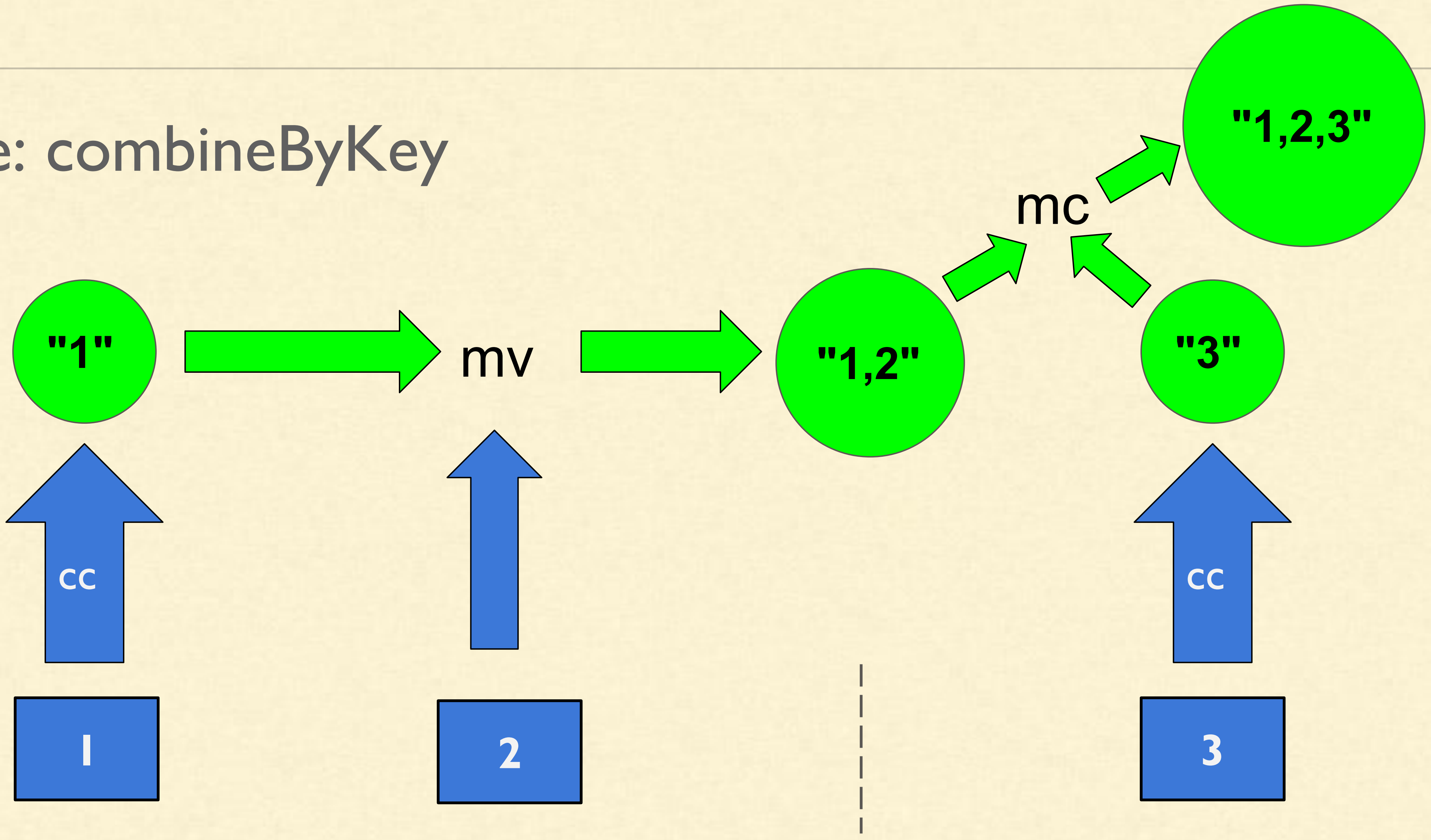
```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
def mv(x:String, y:Int):String = {x + "," + y.toString}
def mc(x:String, y:String):String = {x + ", " + y}
```

Example: combineByKey



```
var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
def mv(x:String, y:Int):String = {x + ":" + y.toString}
def mc(x:String, y:String):String = {x + "," + y}
myrdd.combineByKey(cc, mv, mc)
```

Example: combineByKey



var myrdd = sc.parallelize(List(1,2,3), 2).map(("x", _))
def cc(x:Int):String = x.toString
def mv(x:String, y:Int):String = {x + "," + y.toString}
def mc(x:String, y:String):String = {x + ", " + y}
myrdd.combineByKey(cc, mv, mc).collect()(0)._2

String = 1, 2, 3, 4,5

Questions - Set Operations

What will be the result of the following?

```
def cc (v): return ("[" , v , "]");

def mv (c, v): return c[0:-1] + (v, "]")

def mc(c1,c2):    return c1[0:-1] + c2[1:]

mc(mv(cc(1), 2), cc(3))
```

Questions - Set Operations

What will be the result of the following?

```
def cc (v): return ("[" , v , "]");

def mv (c, v): return c[0:-1] + (v, "]")

def mc(c1,c2):    return c1[0:-1] + c2[1:]

mc(mv(cc(1), 2), cc(3))
```

```
('[', 1, 2, 3, ']')
```

Questions - Set Operations

What will be the result of the following?

```
def cc (v): return ("[" , v , "]");

def mv (c, v): return c[0:-1] + (v, "]")

def mc(c1, c2): return c1[0:-1] + c2[1:]

rdd = sc.parallelize([("a", 1), ("b", 2), ("a", 3)])
rdd.combineByKey(cc,mv, mc).collect()
```

What will be the result of the following?

```
def cc (v): return ("[" , v , "]");

def mv (c, v): return c[0:-1] + (v, "]")

def mc(c1, c2):    return c1[0:-1] + c2[1:]

rdd = sc.parallelize([("a", 1), ("b", 2), ("a", 3)])
rdd.combineByKey(cc,mv, mc).collect()
```

```
[('a', ('[', 1, 3, ']')), ('b', ('[', 2, ']'))]
```

Transformations on Pair RDDs

sortByKey(ascending=true, numPartitions=current partitions)

Sorts this RDD, which is assumed to consist of (key, value) pairs.

Transformations on Pair RDDs

sortByKey(ascending=true, numPartitions=current partitions)

Sorts this RDD, which is assumed to consist of (key, value) pairs.

```
>>> var tmp = List(('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5))
>>> sc.parallelize(tmp).sortByKey().collect()

Array((1,3), (2,5), (a,1), (b,2), (d,4))
```

Transformations on Pair RDDs

sortByKey(ascending=true, numPartitions=current partitions)

Sorts this RDD, which is assumed to consist of (key, value) pairs.

```
>>> var tmp = List(('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5))

>>> sc.parallelize(tmp).sortByKey(true, 1).collect()
Array((1,3), (2,5), (a,1), (b,2), (d,4))
```

Transformations on Pair RDDs

sortByKey(ascending=true, numPartitions=current partitions)

Sorts this RDD, which is assumed to consist of (key, value) pairs.

```
>>> var tmp = List(('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5))

>>> sc.parallelize(tmp).sortByKey(ascending=false,
numPartitions=2).collect()
Array((d,4), (b,2), (a,1), (2,5), (1,3))
```

Transformations on Pair RDDs

subtractByKey(other, numPartitions=None)

Return each (key, value) pair in self that has no pair with matching key in other.

```
>>> var x = sc.parallelize(List(("a", 1), ("b", 4), ("b", 5), ("a", 2)))
>>> var y = sc.parallelize(List(("a", 3), ("c", None)))
>>> x.subtractByKey(y).collect()
[('b', 4), ('b', 5)]
```

Transformations on Pair RDDs

join(other, numPartitions=None)

Return an RDD containing all pairs of elements with matching keys in self and other.
Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in self and (k, v2) is in other.

Transformations on Pair RDDs

join(other, numPartitions=None)

Return an RDD containing all pairs of elements with matching keys in self and other.
Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in self and (k, v2) is in other.

```
>>> var x = sc.parallelize(List(("a", 1), ("b", 4), ("c", 5)))
>>> var y = sc.parallelize(List(("a", 2), ("a", 3), ("d", 7)))
>>> x.join(y).collect()
Array((a,(1,2)), (a,(1,3)))
```

Transformations on Pair RDDs

leftOuterJoin(other, numPartitions=None)

Perform a left outer join of self and other.

For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other have key k.

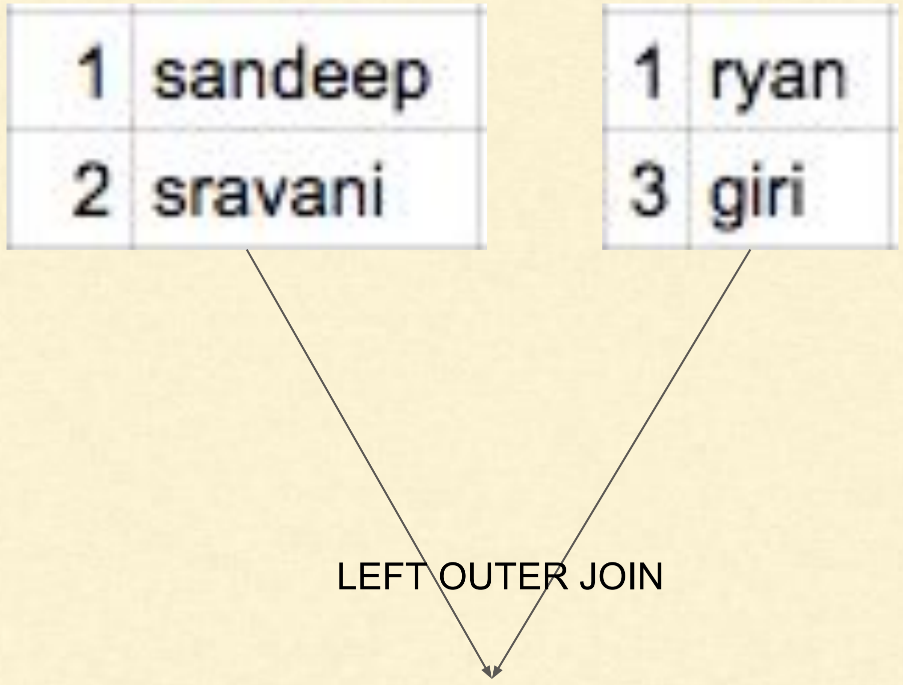
Hash-partitions the resulting RDD into the given number of partitions.

```
>>> var x = sc.parallelize(List(("a", 1), ("b", 4)))
>>> var  y = sc.parallelize(List(("a", 2)))
>>> x.leftOuterJoin(y).collect()
Array((a,(1,Some(2))), (b,(4,None)))
```

Questions - Set Operations

What will be the result of the following?

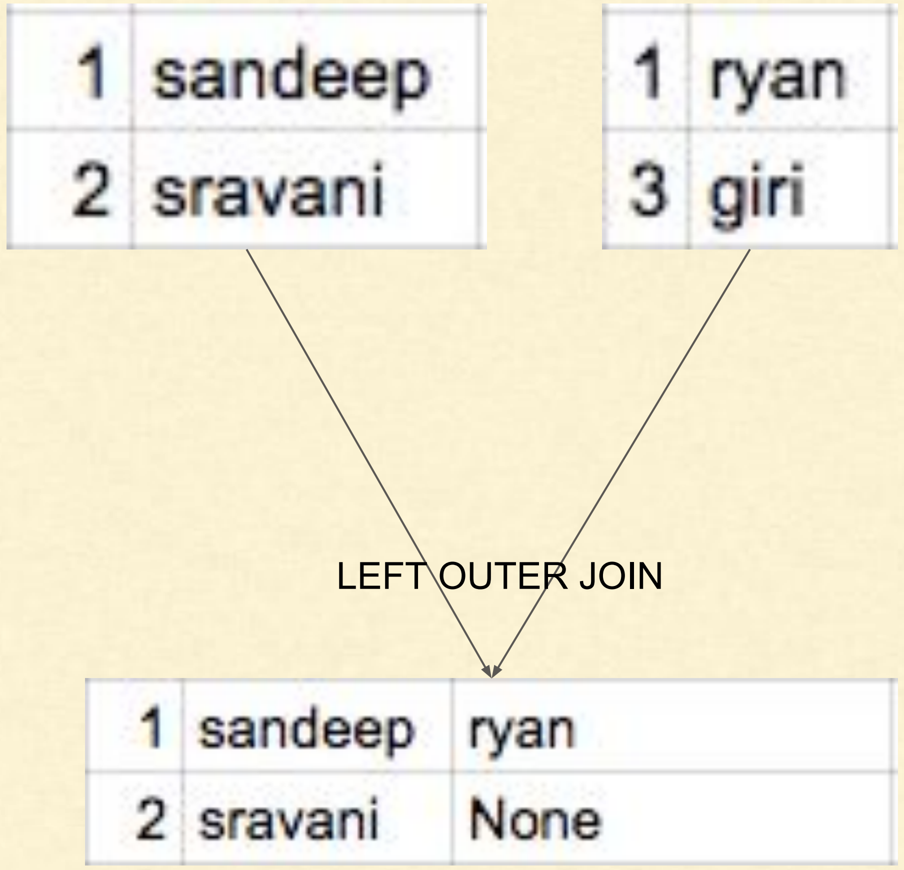
```
x = sc.parallelize(
    [(1, "sandeep"), ("2", "sravani")])
y = sc.parallelize(
    [(1, "ryan"), (3, "giri")])

x.leftOuterJoin(y).collect()
```



Questions - Set Operations

What will be the result of the following?

```
x = sc.parallelize(
    [(1, "sandeep"), ("2", "sravani")])
y = sc.parallelize(
    [(1, "ryan"), (3, "giri")])

x.leftOuterJoin(y).collect()
```



[(1, ('sandeep', 'ryan')), ('2', ('sravani', None))]

Transformations on Pair RDDs

rightOuterJoin(other, numPartitions=None)

Perform a right outer join of ***self*** and ***other***.

For each element (k, w) in ***other***, the resulting RDD will either contain all pairs (k, (v, w)) for v in this, or the pair (k, (None, w)) if no elements in ***self*** have key k.

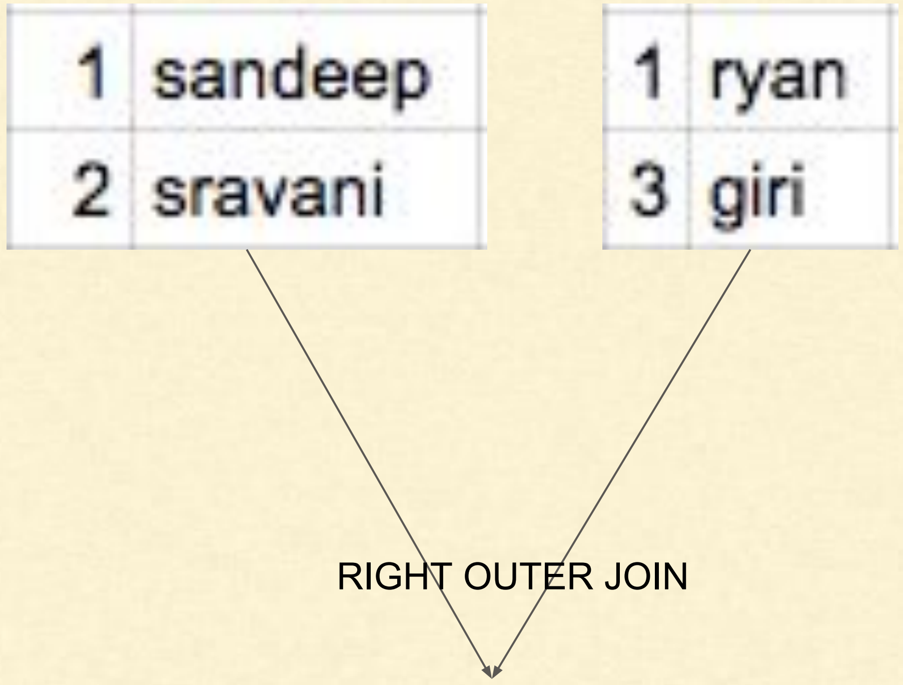
Hash-partitions the resulting RDD into the given number of partitions.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> y.rightOuterJoin(x).collect()
[('a', (2, 1)), ('b', (None, 4))]
```

Questions - Set Operations

What will be the result of the following?

```
x = sc.parallelize(
    [(1, "sandeep"), ("2", "sravani")])
y = sc.parallelize(
    [(1, "ryan"), (3, "giri")])

x.rightOuterJoin(y).collect()
```



1	sandeep
2	sravani

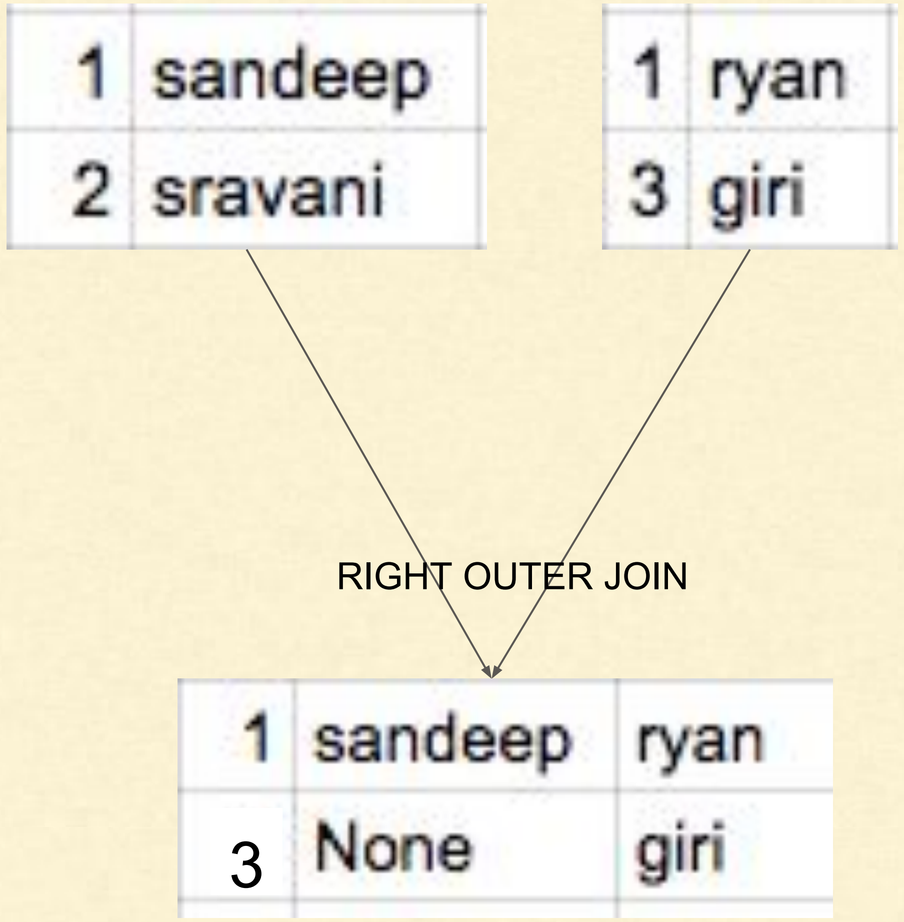
1	ryan
3	giri

Questions - Set Operations

What will be the result of the following?

```
x = sc.parallelize(
    [(1, "sandeep"), ("2", "sravani")])
y = sc.parallelize(
    [(1, "ryan"), (3, "giri")])

x.rightOuterJoin(y).collect()
```

```
[(1, ('sandeep', 'ryan')), (3, (None, 'giri'))]
```



1	sandeep
2	sravani

1	ryan
3	giri

1	sandeep	ryan
3	None	giri

Transformations on Pair RDDs

cogroup(other, numPartitions=None)

For each key k in self or other, return a resulting RDD that contains a tuple with the list of values for that key in self as well as other.

```
>>> var x = sc.parallelize(List(("a", 1), ("b", 4)))
>>> var y = sc.parallelize(List(("a", 2), ("a", 3)))
>>> var cg = x.cogroup(y)
>>> cgl = cg.collect()

Array((a,(CompactBuffer(1),CompactBuffer(2, 3))),
(b,(CompactBuffer(4),CompactBuffer())))
```

This is basically same as:

((a, ([1], [2,3])), (b, ([4], []))))

Actions Available on Pair RDDs

countByKey()

Count the number of elements for each key, and return the result to the master as a dictionary.

```
>>> var rdd = sc.parallelize(List(("a", 1), ("b", 1), ("a", 1), ('a', 10)))
>>> rdd.countByKey()

Map(a -> 2, a -> 1, b -> 1)
```

Actions Available on Pair RDDs

lookup(key)

Return the list of values in the RDD for key. This operation is done efficiently if the RDD has a known partitioner by only searching the partition that the key maps to.

```
var lr = sc.parallelize(1 to 1000).map(x => (x, x) )
lr.lookup(42)
Job 24 finished: lookup at <console>:28, took 0.037469 s
WrappedArray(42)

var sorted = lr.sortByKey()
sorted.lookup(42)  # fast

Job 21 finished: lookup at <console>:28, took 0.008917 s
ArrayBuffer(42)
```

Basics of RDD

Thank you!