Apache Spark

Session 5 - Key-Value RDD

Note: The Session will be recorded.

Start the recording..

WELCOME - KNOWBIGDATA

- ❏ Expert Instructors
- ❏ CloudLabs
- ❏ Lifetime access to LMS
- ❏ Presentations
- ❏ Class Recording
- ❏ Assignments + Quizzes
- ❏ Project Work
- ❏ Real Life Project
- ❏ Course Completion Certificate
- ❏ 24x7 support
- ❏ KnowBigData - Alumni
  - ❏ Jobs
  - ❏ Stay Abreast (Updated Content, Complimentary Sessions)
  - ❏ Stay Connected

COURSE CONTENT

I	Introduction to Big Data with Apache Spark
II	Downloading Spark and Getting Started
III	Programming with RDDs
IV	Working with Key/Value Pairs
V	Loading and Saving Your Data
VI	Advanced Spark Programming
VII	Running on a Cluster
VIII	Tuning and Debugging Spark
IX	Spark SQL, SparkR
X	Spark Streaming
XI	Machine Learning with MLlib, GraphX

About Instructor?

2014	**KnowBigData**	Founded
2014 2012	**Amazon**	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012	**InMobi**	Built Recommender that churns 200 TB
2011 2006 2002	**tBits Global**	Founded tBits Global Built an enterprise grade Document Management System
	D.E.Shaw	Built the big data systems before the term was coined
2002	**IIT Roorkee**	Finished B.Tech.

Key - Value Pair RDDs

1. Special operations on RDDs containing key/value pairs
2. These RDDs are called pair RDDs.
3. Pair RDDs are a useful building block in many programs
4. For example, pair RDDs have a reduceByKey()

1. A Pair in Python is defined as (x, y)
2. Also known as tuple
3. A tuple is an immutable sequence of Python objects.
4. You can convert a list into tuple

Creating Key-Value Pair RDDs

By Directly Loading as Key-Value Pairs

We will discuss it later

By using a map function:

- words = sc.parallelize("this is a bird".split())
- def toKWWords(w):
  return (w, 1);
- words.map(toKWWords).collect()

Transformations on Pair RDDs

keys()

Return an RDD with the keys of each tuple.

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

Transformations

glom()

Return an RDD created by coalescing all elements within each partition into a list.

```
>>> rdd = sc.parallelize([1, 2, 3, 4], 2)
>>> rdd.glom().collect()
[[1, 2], [3, 4]]
```

Transformations on Pair RDDs

values()

Return an RDD with the values of each tuple.

```
>>> m = sc.parallelize([(1, 2), (3, 4)]).values()
>>> m.collect()
[2, 4]
```

Transformations on Pair RDDs

mapValues(func)

Apply a function to each value of a pair RDD without changing the key.

```
>>> rdd = sc.parallelize([(1, 2), (3, 4)])
>>> m = rdd.mapValues(lambda x:x+1)
>>> m.collect()
[(1, 3), (3, 5)]
```

Transformations on Pair RDDs

flatMapValues(f)

Pass each value in the key-value pair RDD through a flatMap function without changing the keys; this also retains the original RDD's partitioning.

```
>>> x = sc.parallelize([("a", "x y z"), ("b", "p r")])
>>> def f(v):return v.split();
>>> x.flatMapValues(f).collect()
[('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'p'), ('b', 'r')]
```

Transformations on Pair RDDs

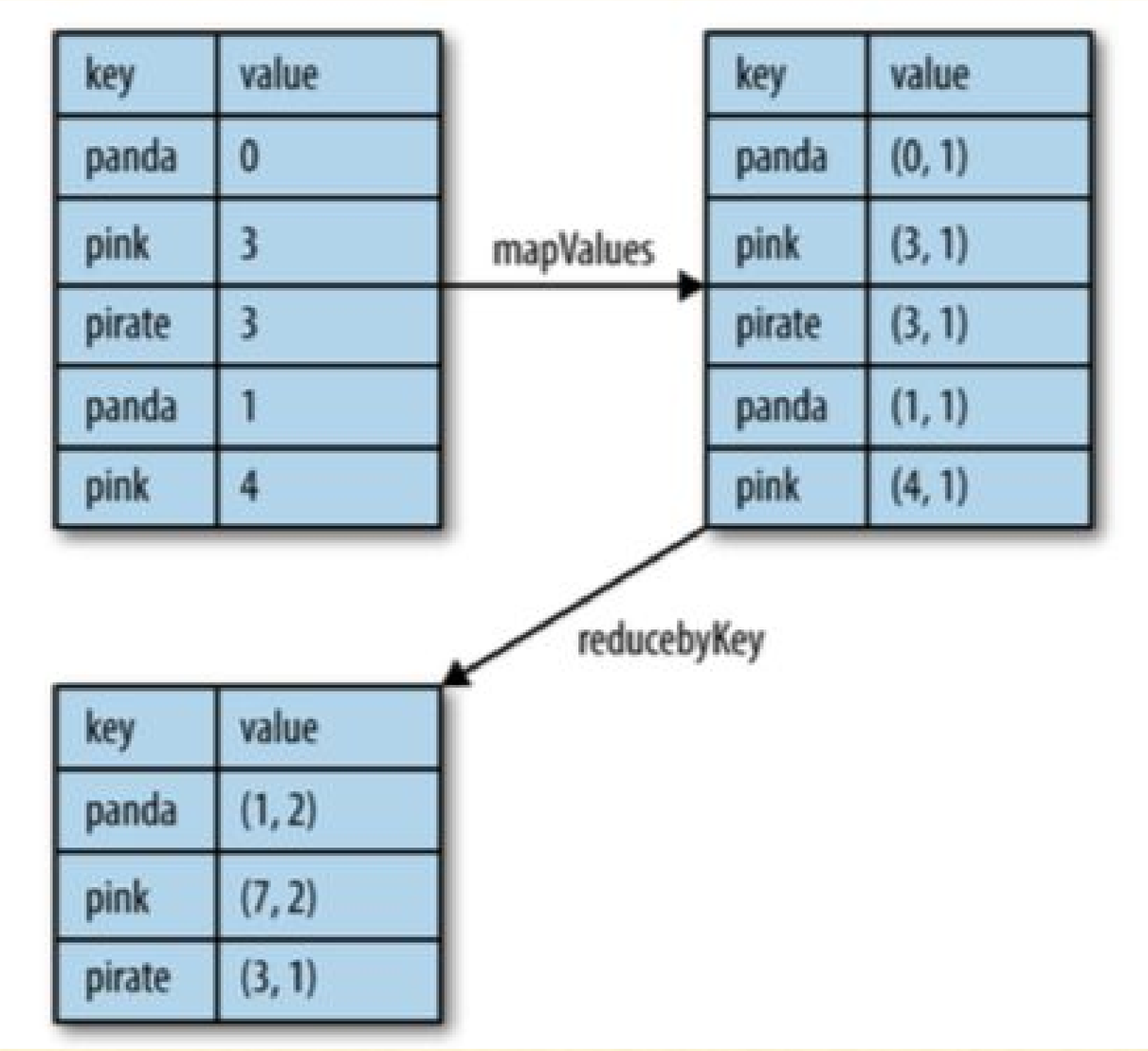
reduceByKey(func)

Combine values with the same key.

```
rdd = sc.parallelize([(1, 2), (3, 4), (3, 6)]);
def sm(x, y):
    return x+y;

rdd1 = rdd.reduceByKey(sm)
rdd1.collect()
[(1, 2), (3, 10)]
```

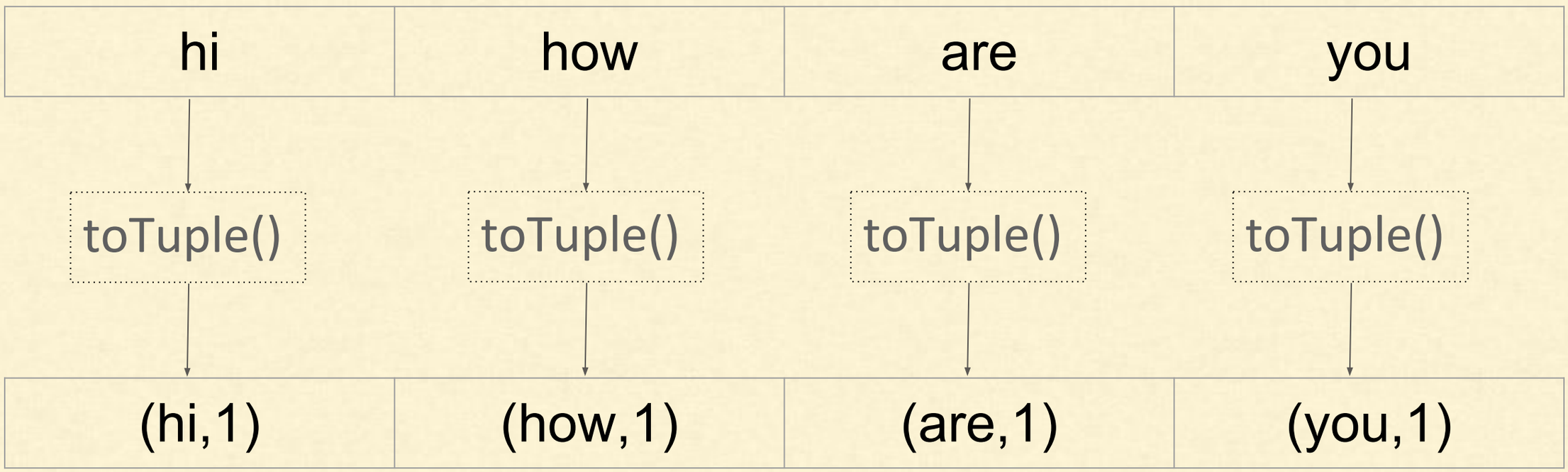
Transformations on Pair RDDs

reduceByKey(func)



Transformation Example (2)

- words = sc.parallelize(['hi', 'how', 'are', 'you'])
- def toTuple(word):
  return (word, 1);
- tuples = words.map(toTuple)
  [('hi', 1), ('how', 1), ('are', 1), ('you', 1)]



Questions - Set Operations

What will be the result of the following?

```
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 10)])
>>> def f(x, y):
    if x > y: return x;
    else: return y;

>>> rdd.reduceByKey(f).collect()
```

What will be the result of the following?

```
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 10)])
>>> def f(x, y):
    if x > y: return x;
    else: return y;

>>> rdd.reduceByKey(f).collect()
```

```
[('a', 10), ('b', 1)]
```

Word Count example

```
#STEP 1: Load the data
lines = sc.textFile('/data/mr/wordcount/input/big.txt')

#STEP 2: Prepare Words
def lineToWords(line):
    return line.split(" ");

words = lines.flatMap(lineToWords)

#STEP 3: convert to key-value pairs
def toKV(word):
    return (word, 1);

wordsKV = words.map(toKV);
```

Word Count example

```
#Step 4: Reduce
def sm(v1, v2):
    return v1+v2;

wc = wordsKV.reduceByKey(sm)
wc.take(10);

#Step 5: Save the data
wc.saveAsTextFile('hdfs://hadoop1.knowbigdata.com/user/spark1/wordcount-output')
```

Transformations on Pair RDDs

groupByKey()

Group values with the same key.

```
rdd = sc.parallelize([(1, 2), (3, 4), (3, 6)]);
rdd1 = rdd.groupByKey()
vals = rdd1.collect()
for i in vals:
    print str(i[0]);
    for j in i[1]:
        print "\t :"   + str(j);
----
1
     :2
3
     :4
     :6
```

Questions - Set Operations

What will be the result of the following?

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)]);
rdd.groupByKey().mapValues(len).collect()
```

Questions - Set Operations

What will be the result of the following?

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)]);
rdd.groupByKey().mapValues(len).collect()
```

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

[('a', 1), ('b', 1), ('a', 1)]
=> groupBY => [('a', [1, 1]), ('b', [1])]
=> mapValues(len) => [('a', 2), ('b', 1)]

Transformations on Pair RDDs

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

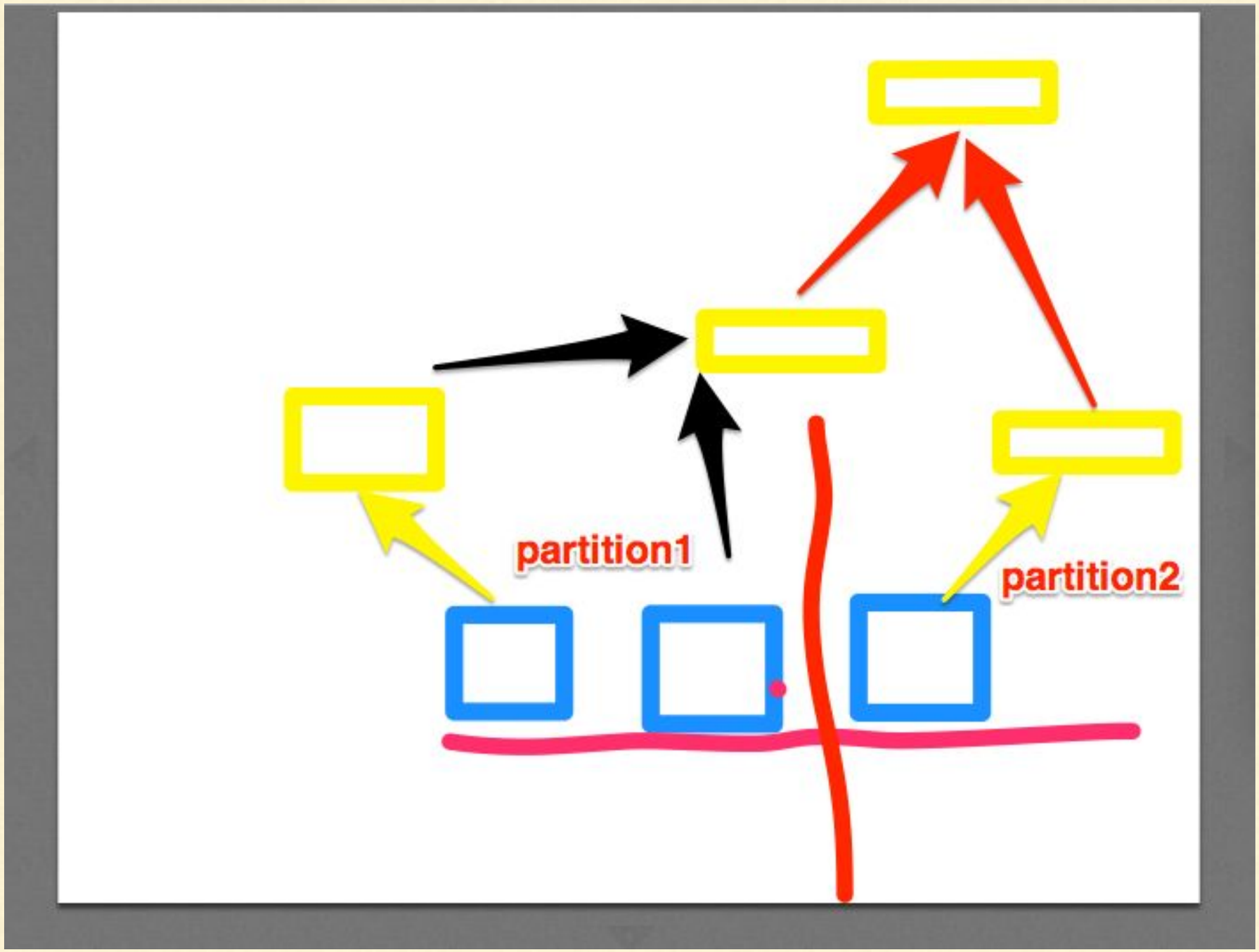
Combine values with the same key using a different result type.
Turns RDD[(K, V)] into a result of type RDD[(K, C)]

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.

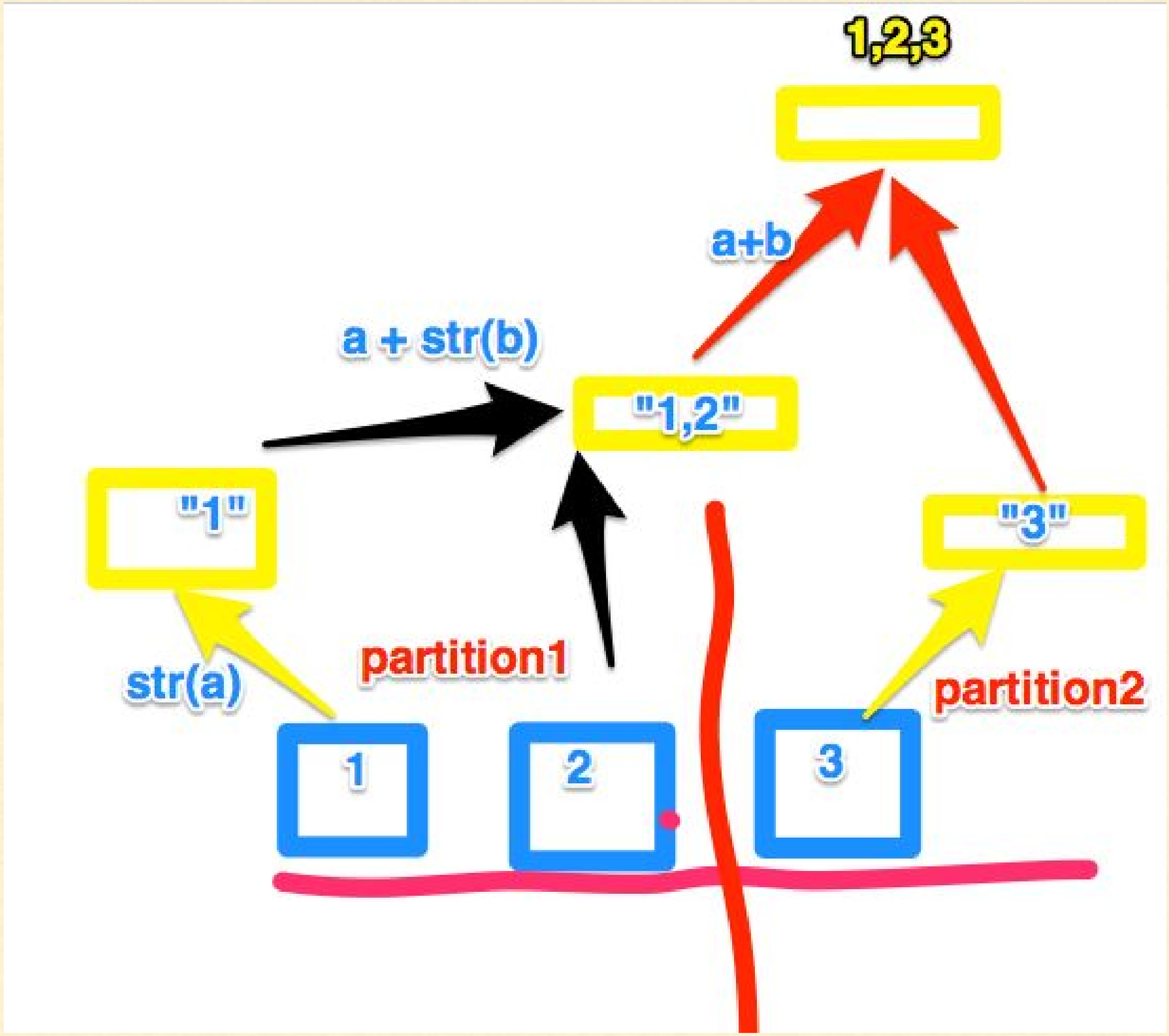
```
rdd1 = sc.parallelize([("a", 1), ("a", 2), ("a", 3), ("b", 3)])
def f(x): return str(x)
def g(x, y): return str(x)+ "," +y;
def h(x, y): return x+"," + y;
mergedrdd = rdd1.combineByKey(f, g, h, numPartitions=None)
```

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

Transformations on Pair RDDs



Transformations on Pair RDDs



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()
```

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()
```

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

Transformations on Pair RDDs

sortByKey(ascending=True, numPartitions=None, keyfunc=<function <lambda>>)

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

```
>>> tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
>>> sc.parallelize(tmp).sortByKey().first()
('1', 3)

>>> sc.parallelize(tmp).sortByKey(True, 1).collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]

>>> sc.parallelize(tmp).sortByKey(ascending=True, numPartitions=2).
collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
```

Transformations on Pair RDDs

sortByKey(ascending=True, numPartitions=None, keyfunc=<function <lambda>>)

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

```
>>> tmp = [('Mary', 1), ('had', 2), ('a', 3), ('little', 4), ('lamb', 5), ('whose', 6), ('fleece', 7), ('was', 8), ('white', 9)]
>>> sc.parallelize(tmp).sortByKey(True, 3, keyfunc=lambda k: k.lower()).collect()
[('a', 3), ('fleece', 7), ('had', 2), ('lamb', 5),...('white', 9), ('whose', 6)]
```

Transformations on Pair RDDs

subtractByKey(other, numPartitions=None)

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

```
>>> x = sc.parallelize([("a", 1), ("b", 4), ("b", 5), ("a", 2)])
>>> y = sc.parallelize([("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.
Performs a hash join across the cluster.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>>x.join(y).collect()
[('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.

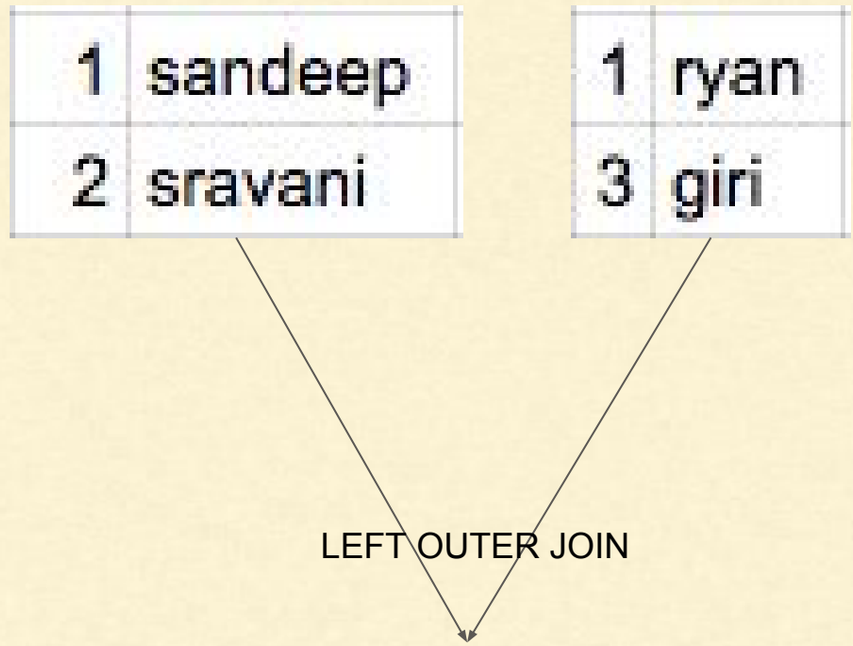
Hash-partitions the resulting RDD into the given number of partitions.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>>x.leftOuterJoin(y).collect()
[('a', (1, 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
2	sravani

1	ryan
3	giri

LEFT OUTER JOIN

1	sandeep	ryan
2	sravani	None

[(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.

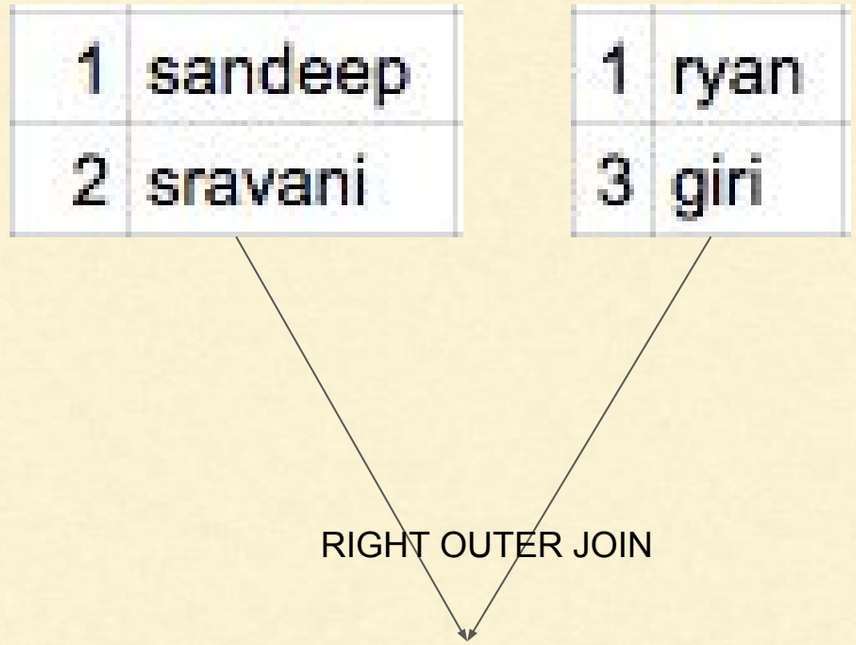
Hash-partitions the resulting RDD into the given number of partitions.

```
>>> 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?

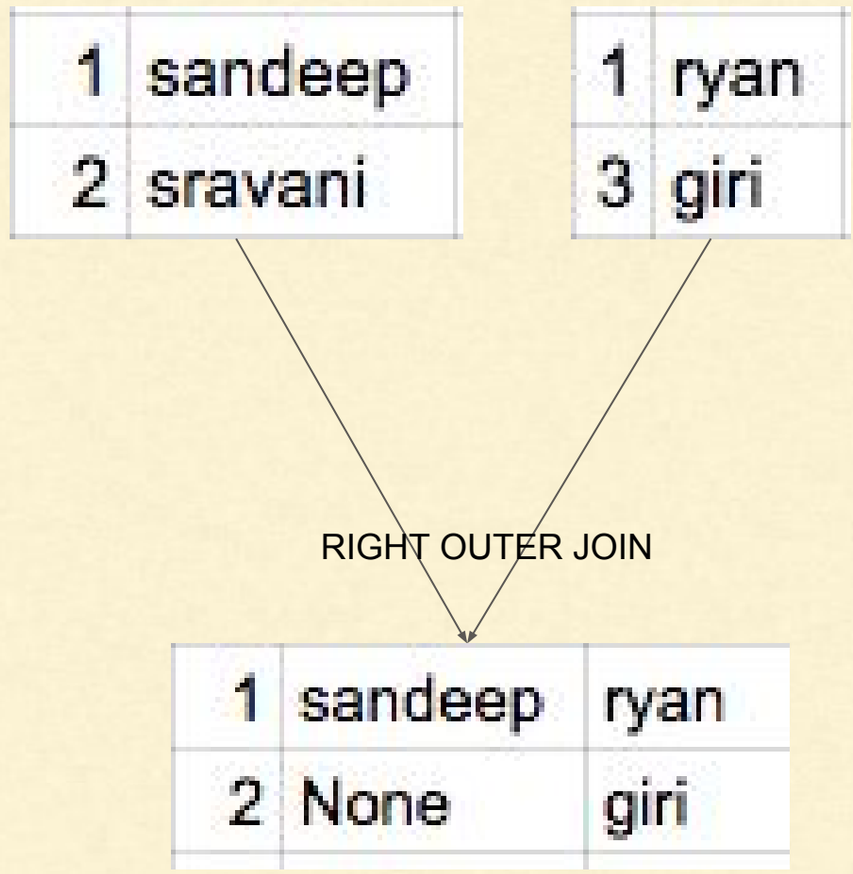
```
x = sc.parallelize(
    [(1, "sandeep"), ("2", "sravani")])
y = sc.parallelize(
    [(1, "ryan"), (3, "giri")])

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



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'))]

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.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> cg = x.cogroup(y)
>>> cgl = cg.collect()
>>> [(x, tuple(map(list, y))) for x, y in sorted(list(cgl))]
[('a', ([1], [2, 3])), ('b', ([4], []))]
---
for x, y in list(cg.collect()):
    for z in y:
        for z1 in z:
            print str(x) + ":" + str(z1)
```

Questions - Set Operations

What will be the result of the following?

```
>>> x = sc.parallelize([("a", 1),("a", 3), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> [(x, tuple(map(list, y))) for x, y in sorted(list(x.
cogroup(y).collect()))]
```

Questions - Set Operations

What will be the result of the following?

```
>>> x = sc.parallelize([("a", 1),("a", 3), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> [(x, tuple(map(list, y))) for x, y in sorted(list(x.
cogroup(y).collect()))]
```

[('a', ([1, 3], [2])), ('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.

```
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1), ('a', 10)])
>>> rdd.countByKey().items()
[('a', 3), ('b', 1)]
```

Actions Available on Pair RDDs

collectAsMap()

Return the key-value pairs in this RDD to the master as a dictionary.

```
>>> m = sc.parallelize([("1", 2), ("3", 4)]).collectAsMap()
>>> m['1']
2
>>> m['3']
4
```

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.

```
>>> l = range(1000)
>>> rdd = sc.parallelize(zip(l, l), 10)
>>> rdd.lookup(42)  # slow
[42]
>>> sorted = rdd.sortByKey()
>>> sorted.lookup(42)  # fast
[42]
>>> sorted.lookup(1024)
[]
```

Data Partitioning

- Layout data to minimize transfer
- Not helpful if you need to scan the dataset only once
- Helpful when dataset is reused multiple times in key-oriented operations
- Available for k-v rdds
- Causes system to group elements based on key
- Spark does not give explicit control which worker node has which key
- It lets program control/ensure which set of key will appear together
  - - based on some hash value for e.g.
  - - or you could range-sort

Data Partitioning - Example

Application Scenerio

- Keeps a large table **UserData**
  - which is an RDD of **(UserID, UserSubscriptionTopicsInfo)** pairs
- Periodically combines UserData with last five mins **events** (smaller)
  - say, a table of (UserID, LinkInfo) pairs for users who have clicked a link on a website in those five minutes.

Objective:

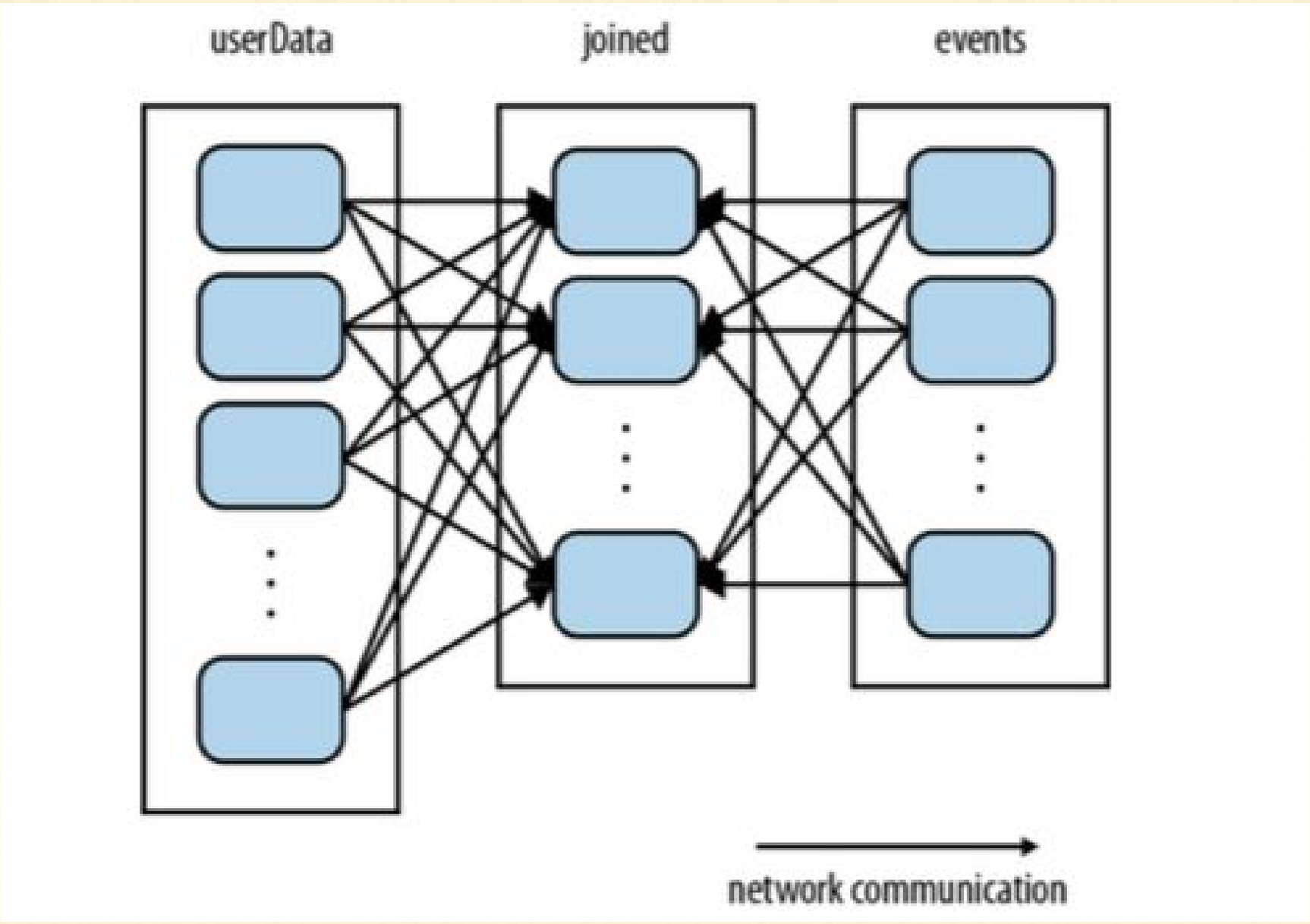
Count how many users visited a link that was not one of their subscribed topics.

Data Partitioning - Example

```
# Initialization code; we load the user info from a Hadoop SequenceFile on HDFS.
# This distributes elements of userData by the HDFS block where they are found,
# and doesn't provide Spark with any way of knowing in which partition a
# particular UserID is located.

sc = SparkContext(...)
userData = sc.sequenceFile("hdfs://...").persist()

# Function called periodically to process a logfile of events in the past 5 minutes
# we assume that this is a SequenceFile containing (UserID, LinkInfo) pairs.

def processNewLogs(logFileName):
    events = sc.sequenceFile(logFileName)
    joined = userData.join(events)
    # RDD of (UserID, (UserInfo, LinkInfo)) pairs

    offTopicVisits = joined.filter(
        lambda (userId, (userInfo, linkInfo)): not userInfo.topics.contains(linkInfo.topic)
    ).count()
  println("Number of visits to non-subscribed topics: " + offTopicVisits)
}
```

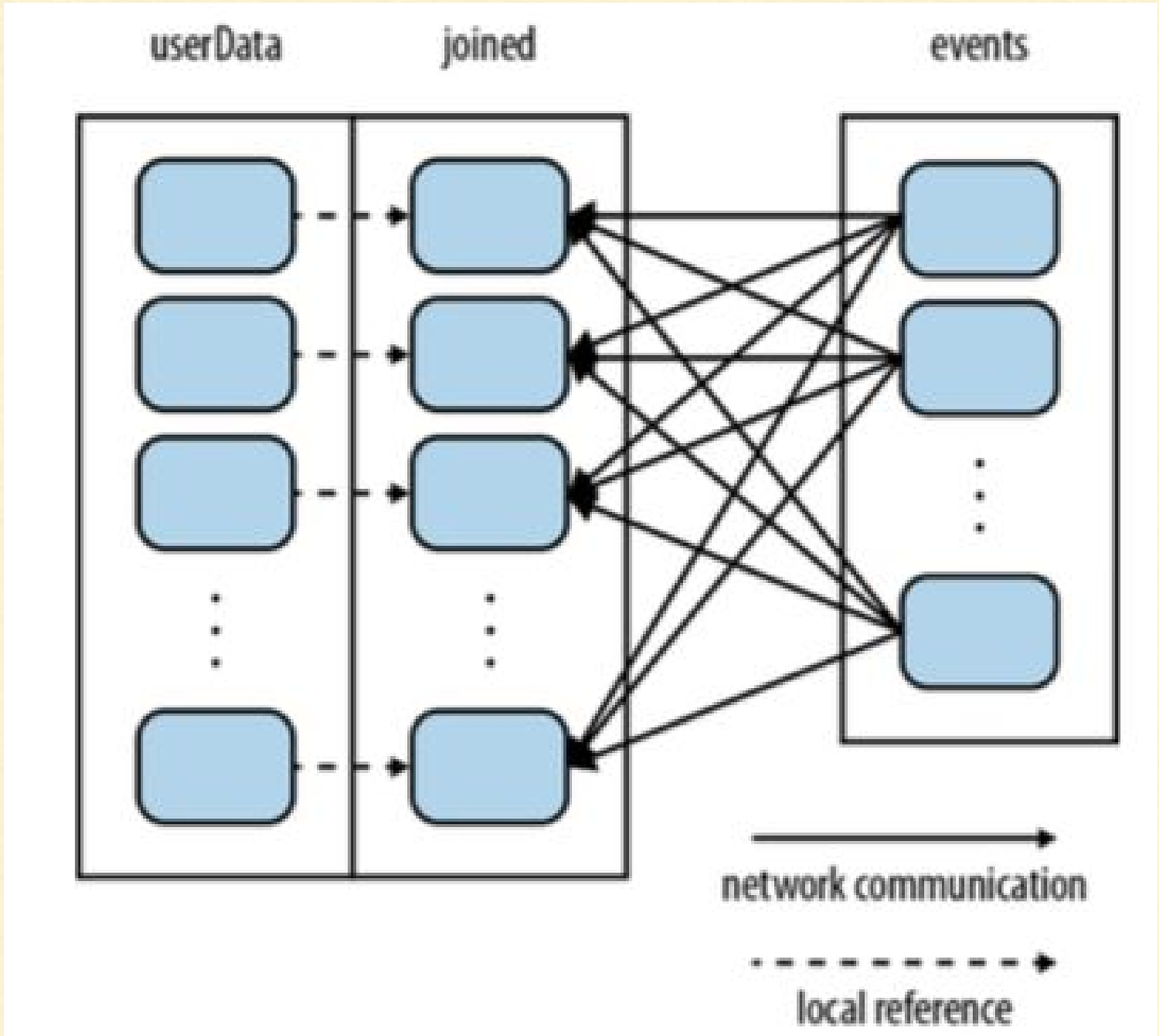
Data Partitioning - Example

Code will run fine as is, but it will be inefficient. Lot of network transfer



Data Partitioning - Example

Fix: userData = sc.sequenceFile("hdfs://...").**partitionBy(10)**.persist()
Note: partitionBy() is a transformation, so it always returns a new RDD



Data Partitioning - Example

Determining an RDD's Partitioner

rdd.partitioner

Operations That Benefit from Partitioning

- cogroup()
- groupWith(), groupByKey(), reduceByKey(),
- join(), leftOuterJoin(), rightOuter Join()
- combineByKey(), and lookup().

Data Partitioning - Customize

```
import urlparse
def hash_domain(url):
    return hash(urlparse.urlparse(url).netloc)

rdd.partitionBy(20, hash_domain) # Create 20 partitions
```

Data Partitioning - Example Problem - PageRank

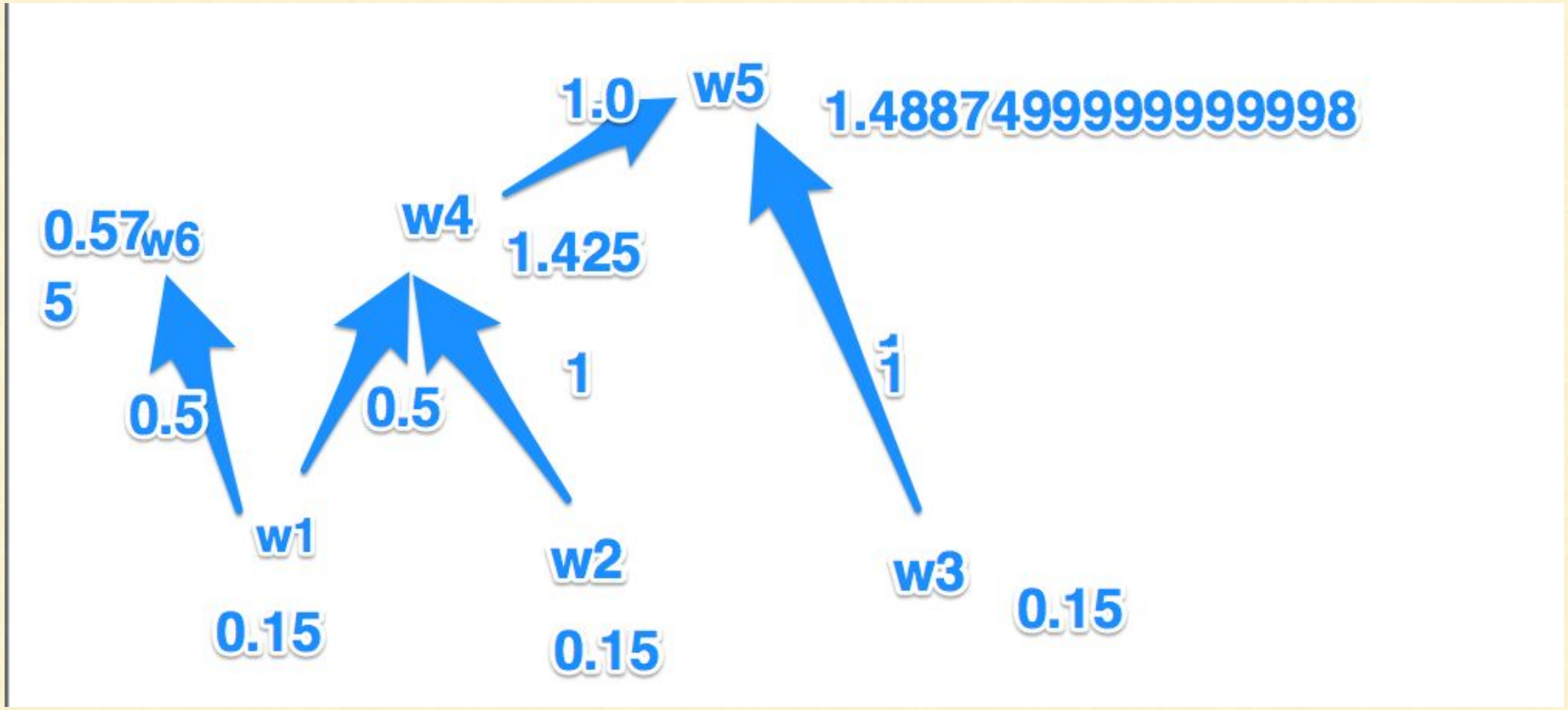
The PageRank algorithm aims to assign a measure of importance (a “rank”) to each document in a set based on how many documents have links to it. PageRank is an iterative algorithm that performs many joins, so it is a good use case for RDD partitioning.
The algorithm maintains two datasets: one of (pageID, link List) elements containing the list of neighbors of each page, and one of (pageID, rank) elements containing the current rank for each page. It proceeds as follows:

1. Initialize each page’s rank to 1.0.
2. On each iteration, have page p send a contribution of rank(p) /numNeighbors(p) to its neighbors (the pages it has links to).
3. Set each page’s rank to 0.15 + 0.85 * contributionsReceived.

The last two steps repeat for several iterations, during which the algorithm will converge to the correct PageRank value for each page. In practice, it’s typical to run about 10 iterations.

[(w1, w6, w4), (w2, w4), (w4, w5), (w3, w5)]

Data Partitioning - Example Problem - PageRank



Data Partitioning - Example Problem - PageRank

```
# Assume that our neighbor list was saved as a Spark objectFile
links = sc.objectFile("links").partitionBy(100).persist()

# Initialize each page's rank to 1.0; since we use mapValues, the resulting RDD
# will have the same partitioner as links
ranks = links.mapValues(lambda v: 1.0)
// Run 10 iterations of PageRank
for (i in range(0:10):
    contributions = links.join(ranks).flatMap(
        lambda (pageId, (links, rank)):
            links.map(dest => (dest, rank / links.size
        )
    ranks = contributions.reduceByKey(
lambda (x, y): x + y).mapValues(v => 0.15 + 0.85*v)

// Write out the final ranks
ranks.saveAsTextFile("ranks")
```

Apache Spark

Thank you.

+1 419 665 3276 (US)
+91 803 959 1464 (IN)

reachus@knowbigdata.com

Subscribe to our Youtube channel for latest videos - https://www.youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA