Resilient Distributed Datasets

Resilient Distributed Dataset

Dataset

Data storage created from:
HDFS, S3, HBase, JSON, text,
Local hierarchy of folders

Or created transforming
another RDD

Resilient Distributed Dataset

Distributed

Distributed across the cluster of machines

Divided in partitions, atomic chunks of data

Resilient Distributed Dataset

Resilient

Recover from errors, e.g. node failure, slow processes

Track history of each partition, re-run

RDD in PySpark

From the PySpark console:

```
integer_RDD = sc.parallelize(range(10), 3)
```

Check partitions

Gather all data on the driver:

```
integer_RDD.collect()
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Check partitions

Maintain splitting in partitions:

```
integer_RDD.glom().collect()
```

```
Out: [[0, 1, 2], [3, 4, 5], [6, 7, 8, 9]]
```

Read text into Spark

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

from HDFS:

```
text_RDD =
sc.textFile("/user/cloudera/input/testfile1")
text_RDD.take(1) #outputs the first line
```

Wordcount in Spark: map

```
def split_words(line):
    return line.split()

def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
```

```
pairs_RDD.collect()
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

Wordcount in Spark: reduce

```
def sum_counts(a, b):
    return a + b

wordcounts_RDD = pairs_RDD.reduceByKey(sum_counts)

wordcounts_RDD.collect()
```

```
Out[]:
[(u'A', 1),
 (u'ago', 1),
 (u'far', 2),
 (u'away', 1),
 (u'in', 1),
 (u'long', 1),
 (u'a', 1),
 (u'time', 1),
 (u'galaxy', 1)]
```

Transformations

Transformations

- RDD are immutable
- Never modify RDD in place
- Transform RDD to another RDD
- Lazy

Create RDD

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

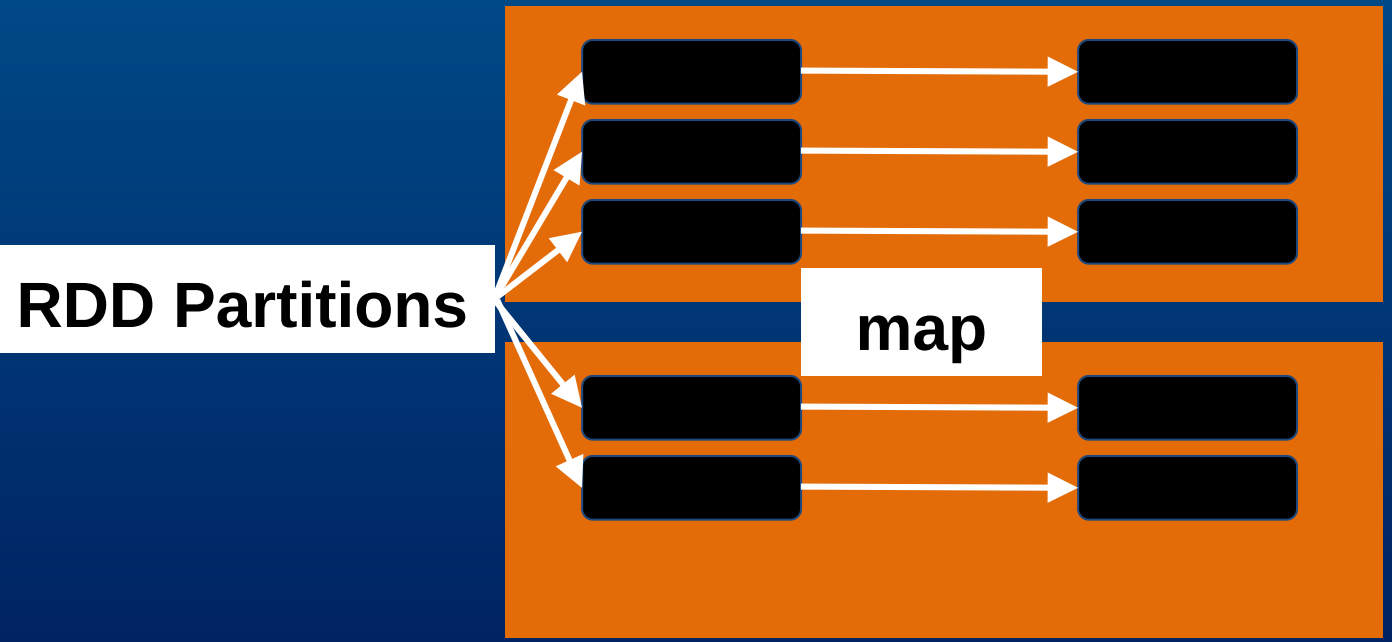
Apply a transformation: map

`map` : apply function to each element of RDD

```
def lower(line):
    return line.lower()
lower_text_RDD = text_RDD.map(lower)
```

map

`map` : apply function to each element of RDD



Other transformations

- flatMap(func) - map then flatten output
- filter(func) - keep only elements where func is true
- sample(withReplacement, fraction, seed) - get a random data fraction
- coalesce(numPartitions) - merge partitions to reduce them to numPartitions

flatMap

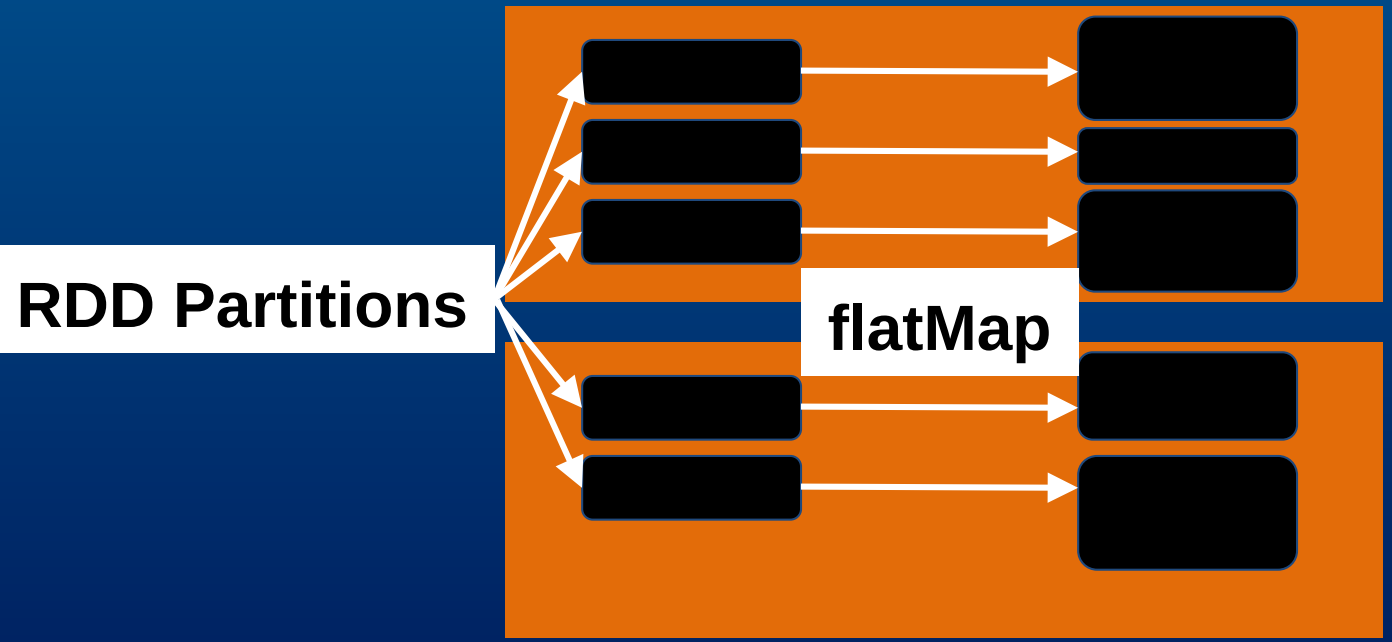
```
def split_words(line):
    return line.split()

words_RDD =
text_RDD.flatMap(split_words)
words_RDD.collect()
```

```
Out[]:
[u'A',
 u'long',
 u'time',
 u'ago',
 u'in',
 u'a',
 u'galaxy',
 u'far',
 u'far',
 u'away']
```

flatMap

flatMap : map then flatten output



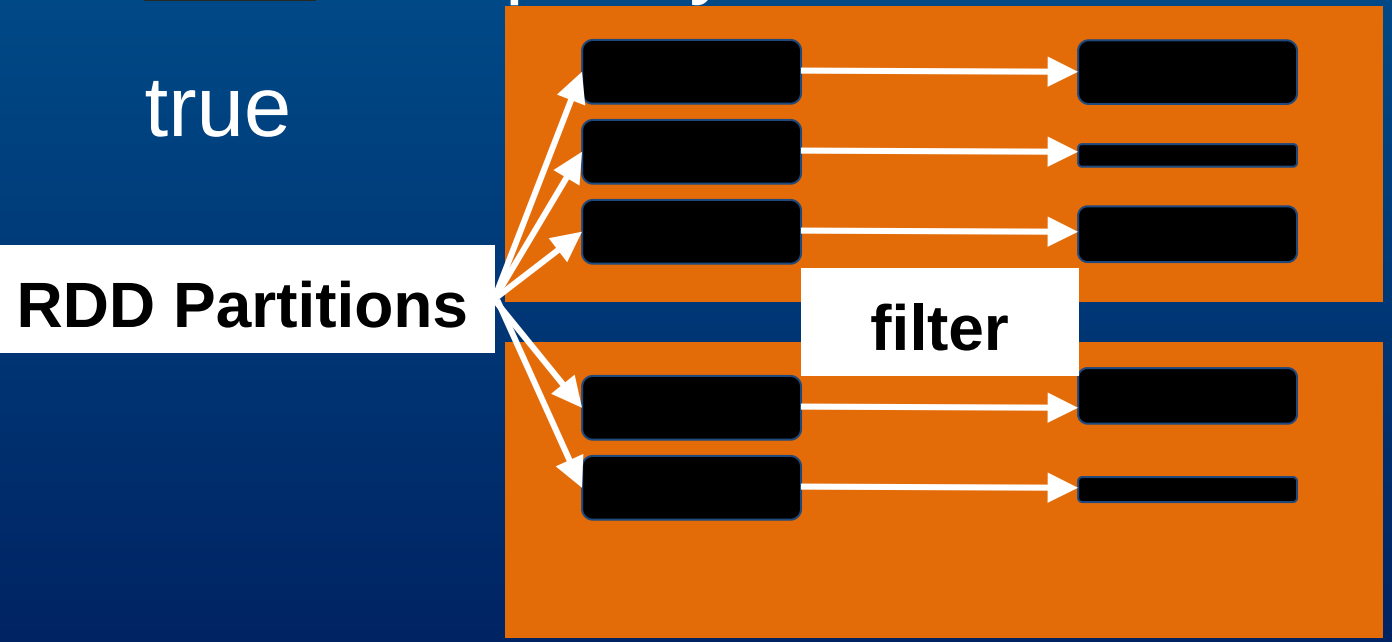
filter

```
def starts_with_a(word):
    return word.lower().startswith("a")
words_RDD.filter(starts_with_a).collect()
```

```
Out[]: [u'A', u'ago', u'a', u'away']
```

filter

`filter` : keep only elements where func is true



coalesce

```
sc.parallelize(range(10), 4).glom().collect()
```

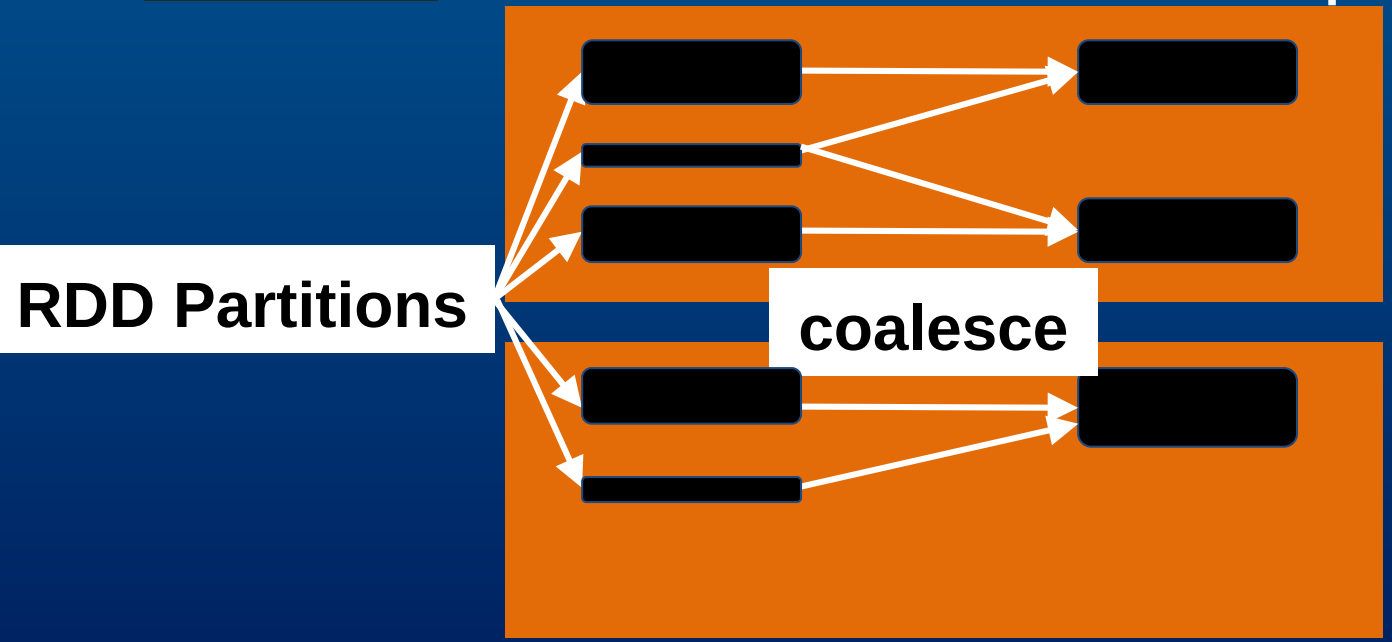
```
Out[]: [[0, 1], [2, 3], [4, 5], [6, 7, 8, 9]]
```

```
sc.parallelize(range(10), 4).coalesce(2).glom().collect()
```

```
Out[]: [[0, 1, 2, 3], [4, 5, 6, 7, 8, 9]]
```

coalesce

`coalesce` : reduce the number of partitions



Wide Transformations

Transformations of (K,V) pairs

```
def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
```

```
pairs_RDD.collect()
```

```
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

groupByKey

`groupByKey` : (K, V) pairs => (K, iterable of all V)

(A, 1)

(B, 8)

(A, [1, 2, 5])

(B, [8])

(A, 2)

(A, 5)

```
pairs_RDD.groupByKey().collect()
```

```
Out[]: [(u'A', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'ago', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'far', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'away', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'in', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'long', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'a', <pyspark.resultiterable.ResultIterable at XXX>),<
```

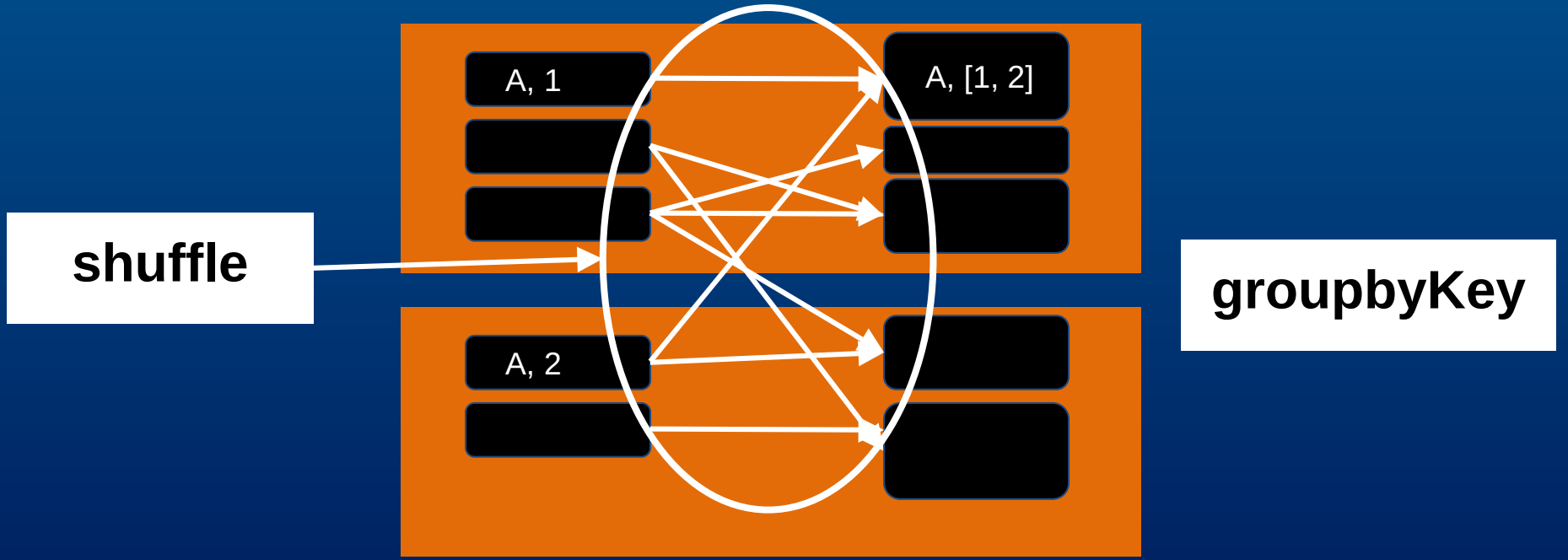
<MORE output>

```
for k,v in pairs_RDD.groupByKey().collect():
    print "Key:", k, ",Values:", list(v)
```

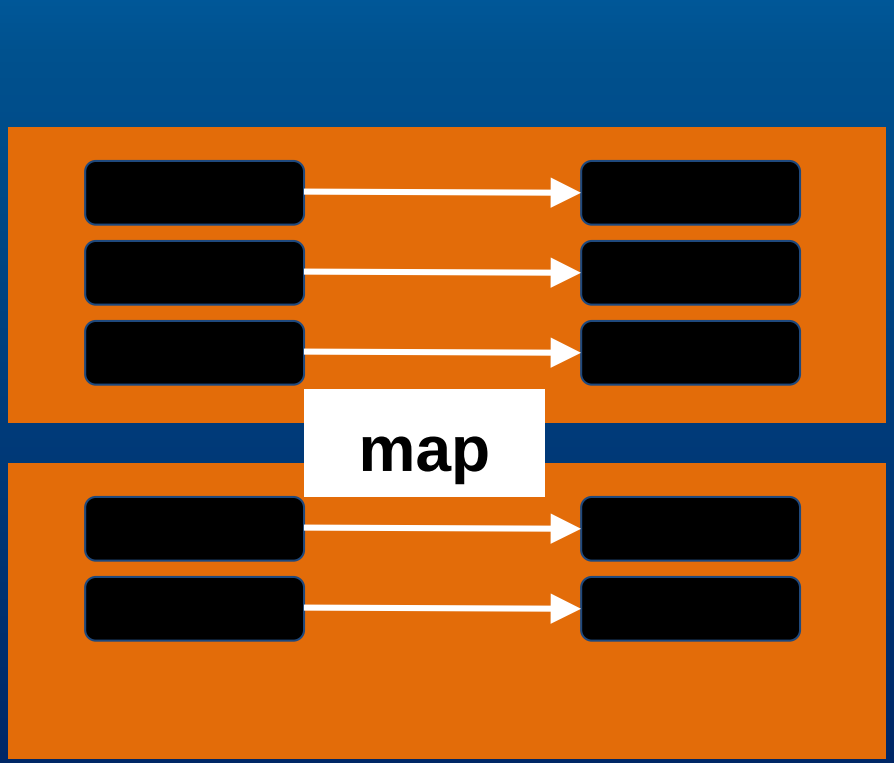
```
Out[]: Key: A , Values: [1]
Key: ago , Values: [1]
Key: far , Values: [1, 1]
Key: away , Values: [1]
Key: in , Values: [1]
Key: long , Values: [1]
Key: a , Values: [1]
<MORE output>
```

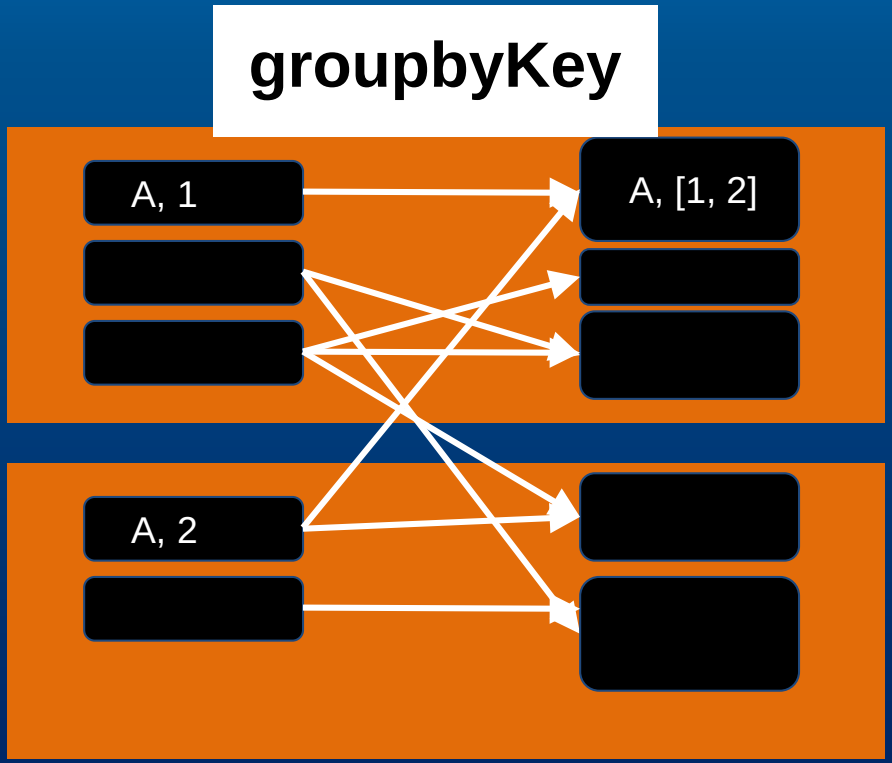
groupByKey

`groupByKey` : (K, V) pairs => (K, iterable of all V)



Narrow vs Wide





Wide transformations

- `groupByKey` : (K, V) pairs => (K, iterable of all V)
- `reduceByKey(func)` : (K, V) pairs => (K, result of reduction by func on all V)
- `repartition(numPartitions)` : similar to coalesce, shuffles all data to increase or decrease number of partitions to numPartitions

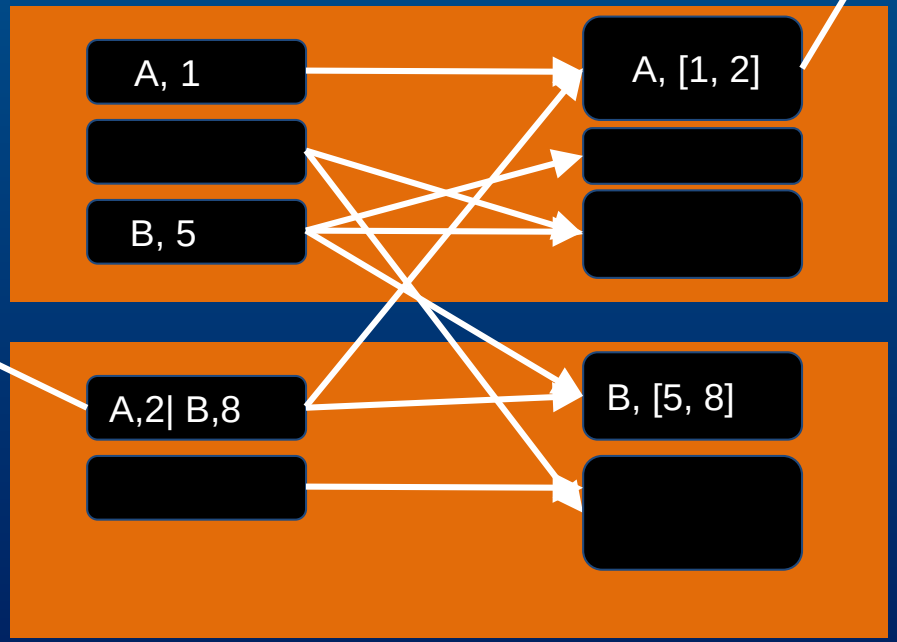
Shuffle

Shuffle

- Global redistribution of data
- High impact on performance

Shuffle

requests data over the network



writes to disk

Know shuffle, avoid it

- Which operations cause it?
- Is it necessary?

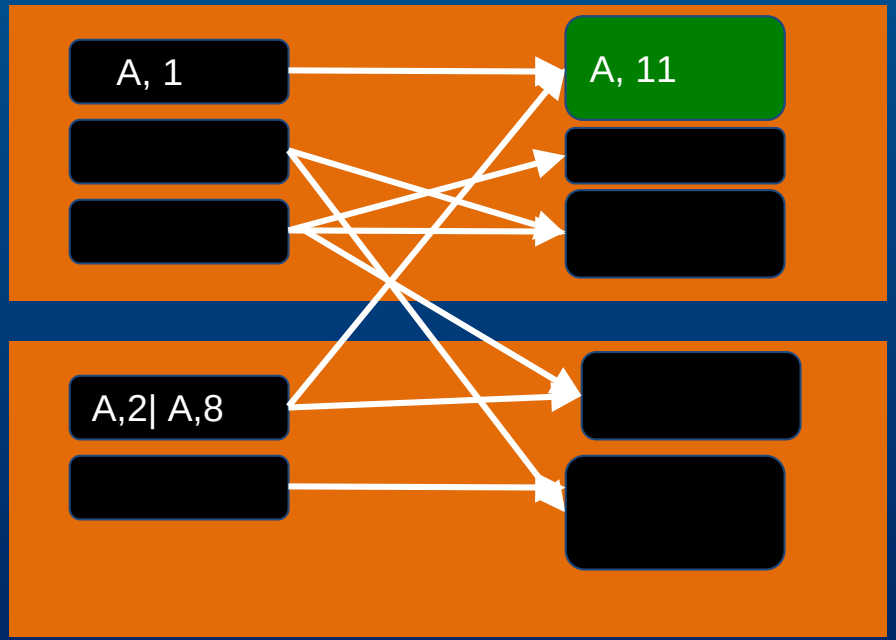
Really need groupByKey?

`groupByKey`: (K, V) pairs => (K, iterable of all V)

if you plan to call `reduce` later in the pipeline,

use `reduceByKey` instead.

groupByKey + reduce



reduceByKey