UNIVERSITAT DE BARCELONA

# Bases de dades avançades

## curs 2017/18

Enric Biosca Trias ebiosca@ub.edu

Dept. Matemàtica Aplicada i Anàlisi.

**Universitat de Barcelona**

UNIVERSITAT DE BARCELONA

# Temari

1. Sistemes analítics i Datawarehouse
2. Big Data & Storage
3. NoSQL

# Big Data i Storage

## Bases de dades avançades curs 17/18

Enric Biosca Trias ebiosca@ub.edu

Dept. Matemàtica Aplicada i Anàlisi.

**Universitat de Barcelona**

# Introducció a Spark

# RDDs

- **RDDs can hold any type of element**
  - Primitive types: integers, characters, booleans, etc.
  - Sequence types: strings, lists, arrays, tuples, dicts, etc. (Including nested data types)
  - Scala/Java Objects (if serializable)
  - Mixed types
- **Some types of RDDs have additional functionality**
  - Pair RDDs
    - RDDs consisting of Key8Value pairs
  - Double RDDs
    - RDDs consisting of numeric data

# Creating RDDs From Collections

- **You can create RDDs from collections instead of files**
  - **`sc.parallelize(`*`collection`*`)`**

```
> randomnumlist = \
    [random.uniform(0,10) for _ in xrange(10000)]
> randomrdd = sc.parallelize(randomnumlist)
> print "Mean: %f" % randomrdd.mean()
```

- **Useful when**
  - Testing
  - Generating data programmatically
  - Integrating

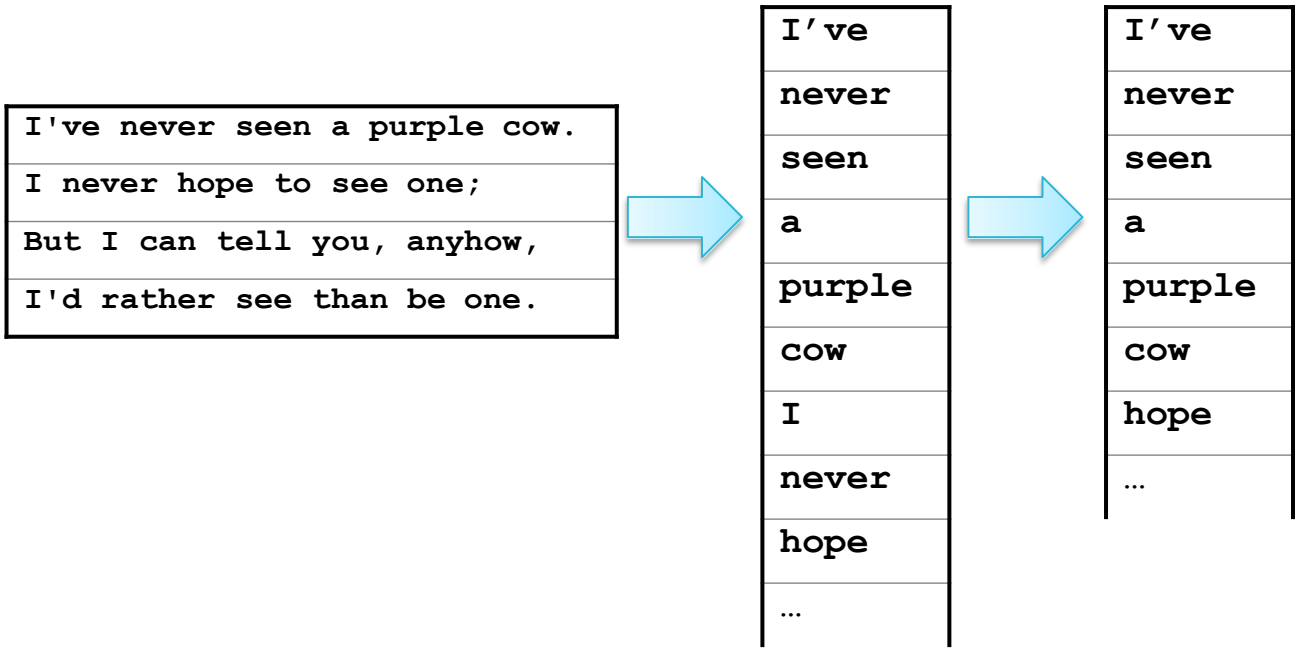
# Some Other General RDD Operations

- **Transformations**
  - **`flatMap`** – maps one element in the base RDD to multiple elements
  - **`distinct`** – filter out duplicates
  - **`union`** – add all elements of two RDDs into a single new RDD
- **Other RDD operations**
  - **`first`** – return the first element of the RDD
  - **`foreach`** – apply a function to each element in an RDD
  - **`top(`*`n`*`)`** – return the largest *n* elements using natural ordering
- **Sampling operations**
  - **`takeSample(`*`withReplacement, num`*`)`** – return an array of `num` sampled elements
- **Double RDD operations**
  - Statistical functions, e.g., **`mean`**, **`sum`**, **`variance`**, **`stdev`**

# Example: `flatMap` and `distinct`

```
> sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .distinct()
```

```
> sc.textFile(file).
   flatMap(line => line.split("\\W")).
   distinct()
```



# Pair RDDs

- **Pair RDDs are a special form of RDD**
  - Each element must be a key-value pair (a two-element tuple)
  - Keys and values can be any type
- **Why?**
  - Use with MapReduce algorithms
  - Many additional functions are available for common data processing needs
    - e.g., sorting, joining, grouping, counting, etc.

Pair RDD

| |
|---|
| (key1,value1) |
| (key2,value2) |
| (key3,value3) |
| … |

# Creating Pair RDDs

- **The first step in most workflows is to get the data into key/value form**
  - What should the RDD be keyed on?
  - What is the value?
- **Commonly used functions to create Pair RDDs**
  - **`map`**
  - **`flatMap` / `flatMapValues`**
  - **`keyBy`**

# Example: A Simple Pair RDD

! **Example: Create a Pair RDD from a tab-separated file**

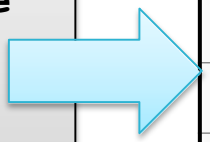
Python

```
> users = sc.textFile(file) \
    .map(lambda line: line.split('\t')) \
    .map(lambda fields: (fields[0],fields[1]))
```

Scala

```
> val users = sc.textFile(file).
    map(line => line.split('\t')).
    map(fields => (fields(0),fields(1)))
```

```
user001   Fred Flintstone
user090   Bugs Bunny
user111   Harry Potter
…
```



| |
|---|
| (user001,Fred Flintstone) |
| (user090,Bugs Bunny) |
| (user111,Harry Potter) |
| … |

# Example: Keying Web Logs by User ID

```
> sc.textFile(logfile) \
    .keyBy(lambda line: line.split(' ')[2])
```

```
> sc.textFile(logfile).
    keyBy(line => line.split(' ')(2))
```

User ID

```
56.38.234.188 - 99788 "GET /KBDOC-00157.html HTTP/1.0" …
56.38.234.188 - 99788 "GET /theme.css HTTP/1.0" …
203.146.17.59 - 25254 "GET /KBDOC-00230.html HTTP/1.0" …
…
```

```
(99788,56.38.234.188 - 99788 "GET /KBDOC-00157.html…)
(99788,56.38.234.188 - 99788 "GET /theme.css…)
(25254,203.146.17.59 - 25254 "GET /KBDOC-00230.html…)
…
```

# Question 1: Pairs With Complex Values

- **How would you do this?**
  - Input: a list of postal codes with latitude and longitude
  - Output: postal code (key) and lat/long pair (value)

```
00210   43.005895   -71.013202
00211   43.005895   -71.013202
00212   43.005895   -71.013202
00213   43.005895   -71.013202
00214   43.005895   -71.013202
…
```

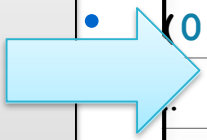
?

```
00210  (43.005895,-71.013202)
00211  (43.005895,-71.013202)
00212  (43.005895,-71.013202)
00213  (43.005895,-71.013202)
```

# Answer 1: Pairs With Complex Values

```
> sc.textFile(file) \
  .map(lambda line: line.split()) \
  .map(lambda fields: (fields[0],(fields[1],fields[2])))
```

```
00210   43.005895   -71.013202
00211   43.005895   -71.013202
00212   43.005895   -71.013202
00213   43.005895   -71.013202
00214   43.005895   -71.013202
…
```



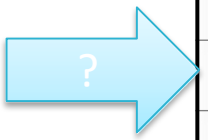
- (00210,(43.005895,-71.013202))
- (00211,(43.005895,-71.013202))
- (00212,(43.005895,-71.013202))
- (00213,(43.005895,-71.013202))
- …

# Question 2: Mapping Single Rows to Multiple Pairs (1)

- **How would you do this?**
  - Input: order numbers with a list of SKUs in the order
  - Output: order (key) and sku (value)

Input Data

```
00001 sku010:sku933:sku022
00002 sku912:sku331
00003 sku888:sku022:sku010:sku594
00004 sku411
```




```
(00001,sku010)
(00001,sku933)
(00001,sku022)
(00002,sku912)
(00002,sku331)
(00003,sku888)
...
```

# Question 2: Mapping Single Rows to Multiple Pairs (2)

! **Hint: `map` alone won't work**

```
00001    sku010:sku933:sku022
00002    sku912:sku331
00003    sku888:sku022:sku010:sku594
00004    sku411
```

```
(00001,(sku010,sku933,sku022))
(00002,(sku912,sku331))
(00003,(sku888,sku022,sku010,sku594))
(00004,(sku411))
```

# Answer 2: Mapping Single Rows to Multiple Pairs (1)

```
> sc.textFile(file)
```

| | |
|---|---|
| 00001 | sku010:sku933:sku022 |
| 00002 | sku912:sku331 |
| 00003 | sku888:sku022:sku010:sku594 |
| 00004 | sku411 |

# Answer 2: Mapping Single Rows to Multiple Pairs (2)

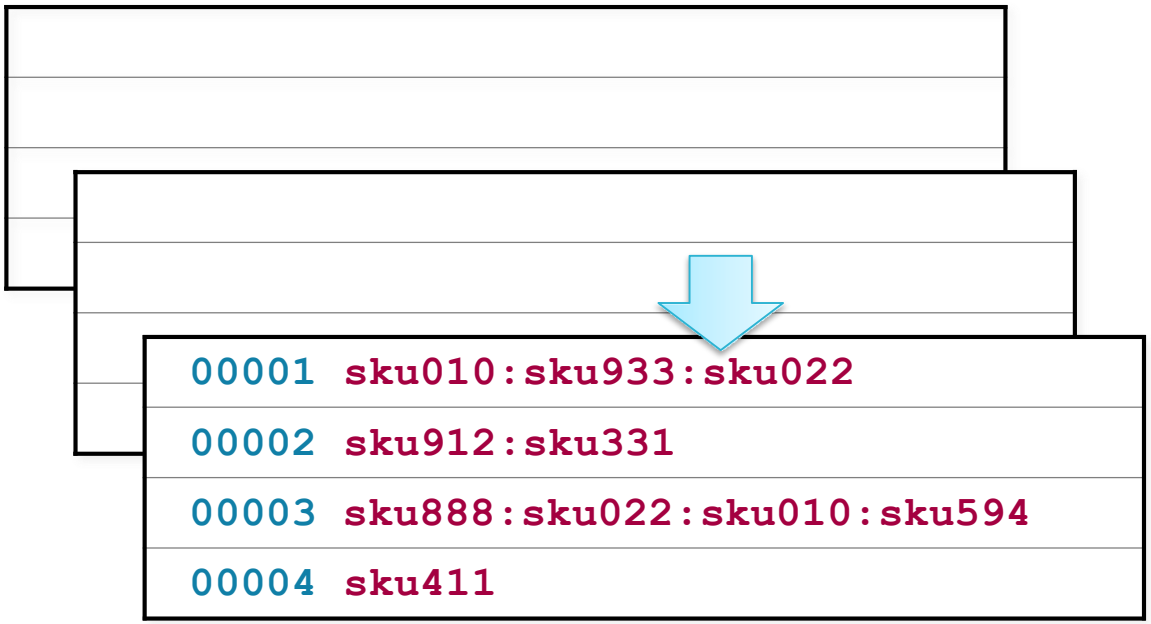
```
> sc.textFile(file) \
  .map(lambda line: line.split('\t'))
```

| | |
|---|---|
| 00001 | sku010:sku933:sku022 |
| 00002 | sku912:sku331 |
| 00 | |
| 00 | |

| |
|---|
| [00001,sku010:sku933:sku022] |
| [00002,sku912:sku331] |
| [00003,sku888:sku022:sku010:sku594] |
| [00004,sku411] |

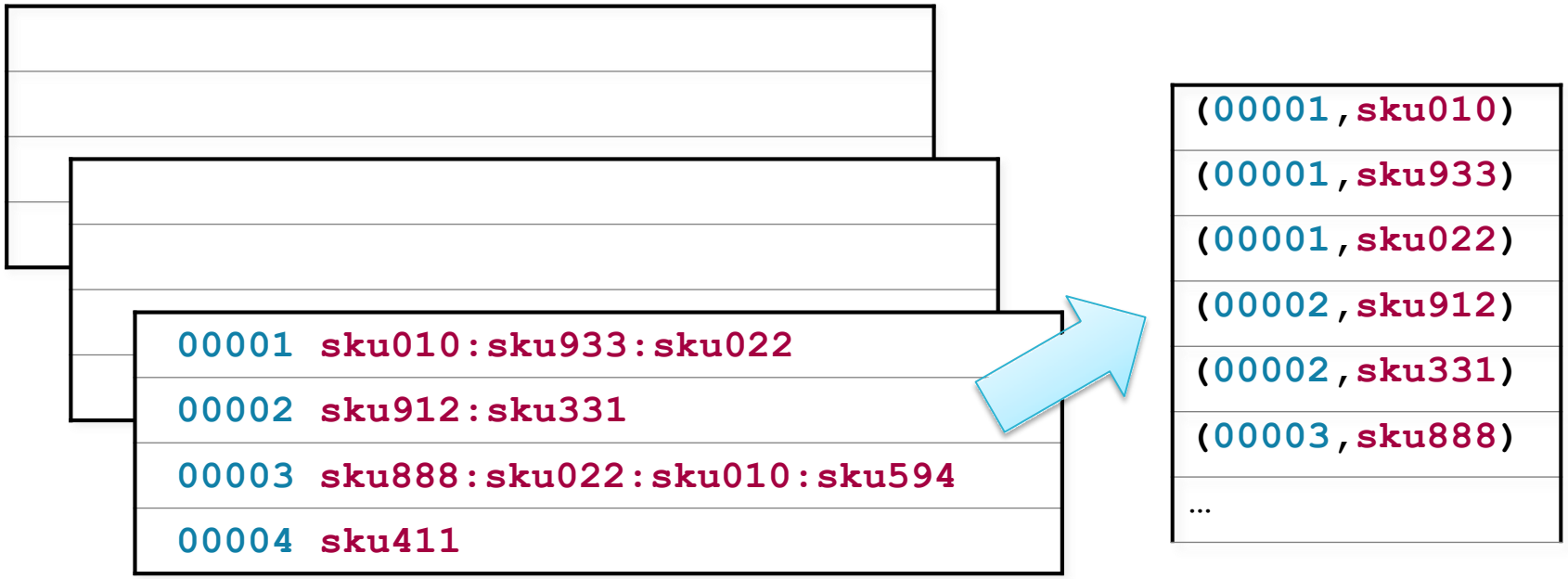
# Answer 2: Mapping Single Rows to Multiple Pairs (3)

```
> sc.textFile(file) \
  .map(lambda line: line.split('\t')) \
  .map(lambda fields: (fields[0],fields[1]))
```



# Answer 2: Mapping Single Rows to Multiple Pairs (4)

```
> sc.textFile(file) \
  .map(lambda line: line.split('\t')) \
  .map(lambda fields: (fields[0],fields[1])) \
  .flatMapValues(lambda skus: skus.split(':'))
```



# MapReduce

- **MapReduce is a common programming model**
  - Easily applicable to distributed processing of large data sets
- **Hadoop MapReduce is the best-known implementation**
  - Somewhat limited
    - Each job has one Map phase, one Reduce phase
    - Job output is saved to files
- **Spark implements MapReduce with much greater flexibility**
  - Map and Reduce functions can be interspersed
  - Results are stored in memory
    - Operations can easily be chained

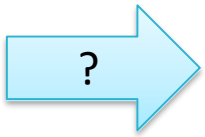
# MapReduce in Spark

- **MapReduce in Spark works on Pair RDDs**
- **Map phase**
  - Operates on one record at a time
  - "Maps" each record to one or more new records
  - **`map`** and **`flatMap`**
- **Reduce phase**
  - Works on Map output
  - Consolidates multiple records
  - **`reduceByKey`**

# MapReduce Example: Word Count

## Result

```
the cat sat on the mat
the aardvark sat on the sofa
```




| | |
|---|---|
| aardvark | 1 |
| cat | 1 |
| mat | 1 |
| on | 2 |
| sat | 2 |
| sofa | 1 |
| the | 4 |

UNIVERSITAT DE BARCELONA

# Example: Word Count (1)

```
> counts = sc.textFile(file)
```

| |
|---|
| the cat sat on the mat |
| the aardvark sat on the sofa |

UNIVERSITAT DE BARCELONA

# Example: Word Count (2)

```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split())
```

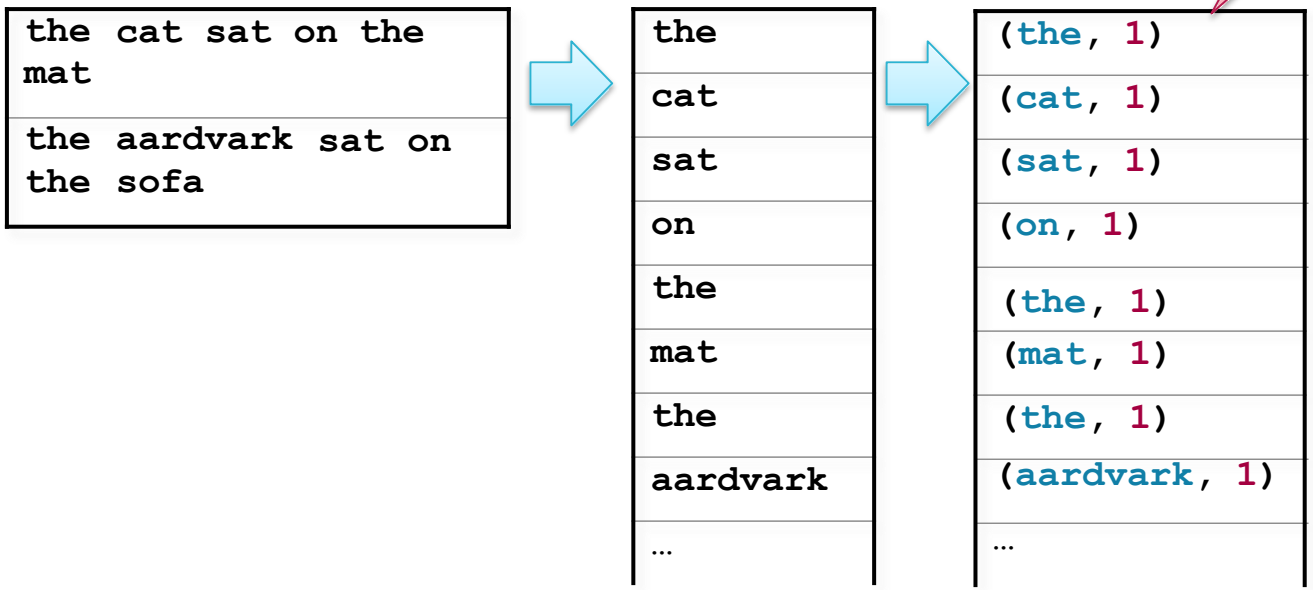
| the cat sat on the mat |
| --- |
| the aardvark sat on the sofa |

| the |
| --- |
| cat |
| sat |
| on |
| the |
| mat |
| the |
| aardvark |
| … |

# Example: Word Count (3)

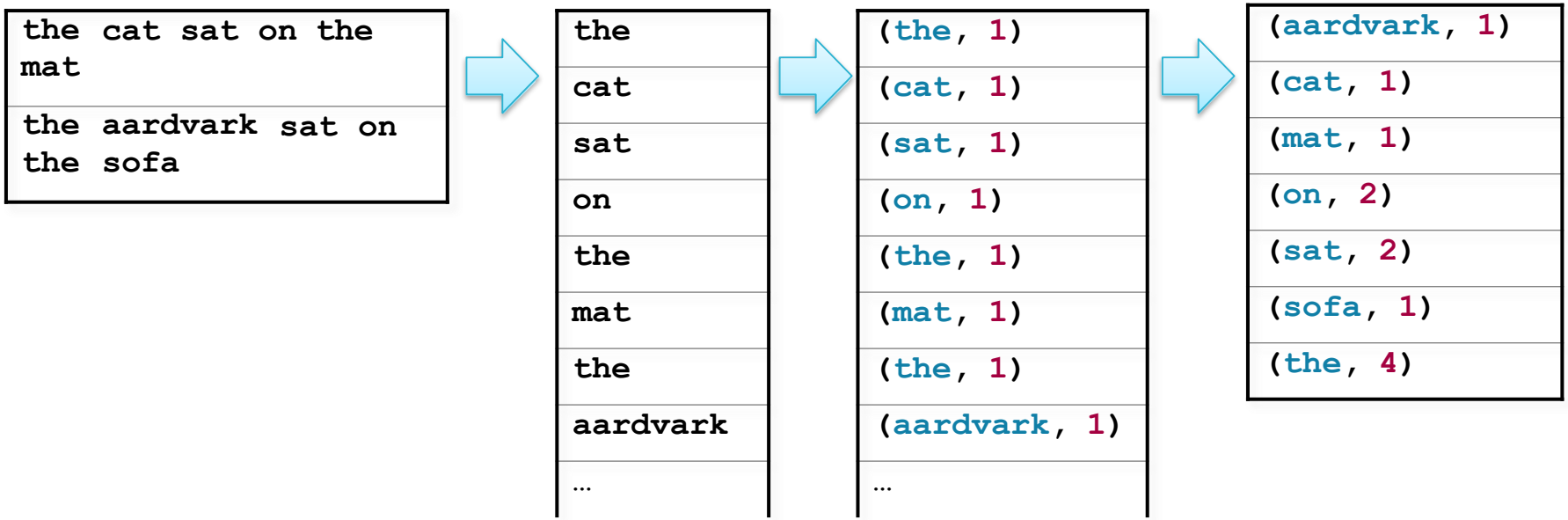
```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1))
```

Key8 Value Pairs



# Example: Word Count (4)

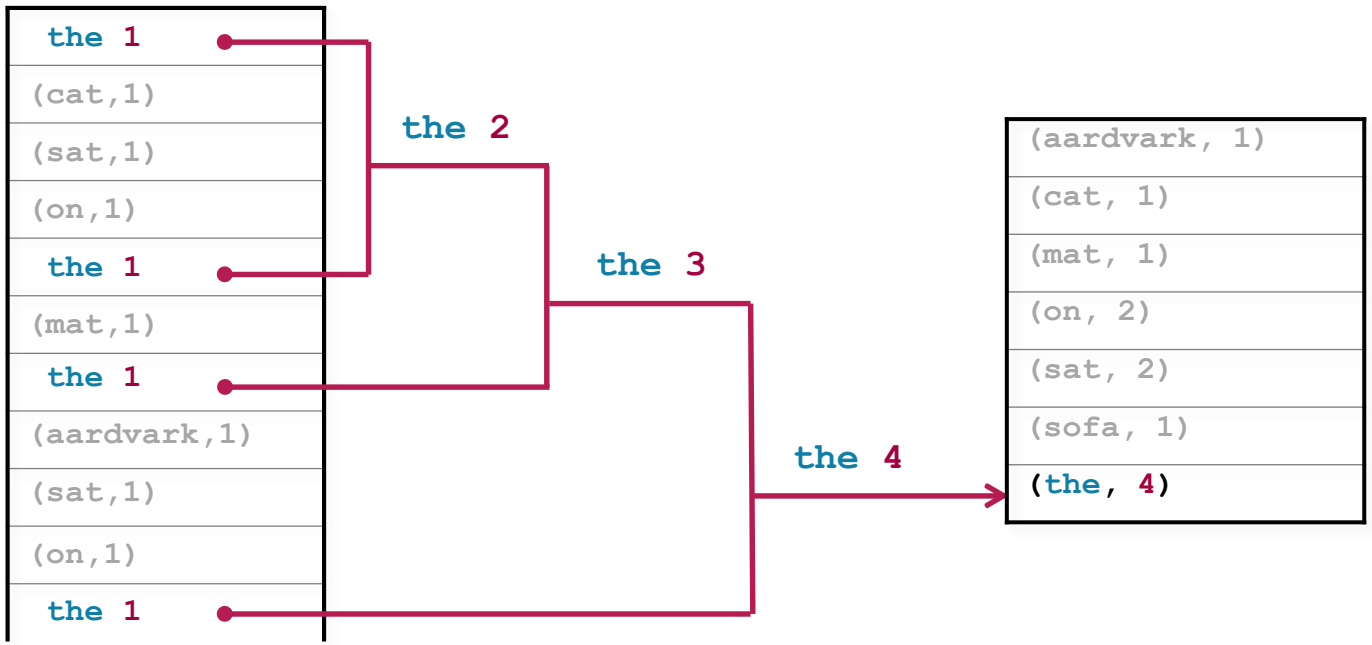
```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1)) \
    .reduceByKey(lambda v1,v2: v1+v2)
```



# ReduceByKey

- **ReduceByKey functions must be**
  - Binary – combines values from two keys
  - Commutative – x+y = y+x
  - Associative – (x+y)+z = x+(y+z)

```
> counts = sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .map(lambda word: (word,1)) \
  .reduceByKey(lambda v1,v2: v1+v2)
```



# Word Count Recap (the Scala Version)

```
> val counts = sc.textFile(file).
    flatMap(line => line.split("\\W")).
    map(word => (word,1)).
    reduceByKey((v1,v2) => v1+v2)
```

```
> val counts = sc.textFile(file).
    flatMap(_.split("\\W")).
    map((_,1)).
    reduceByKey(_+_)
```

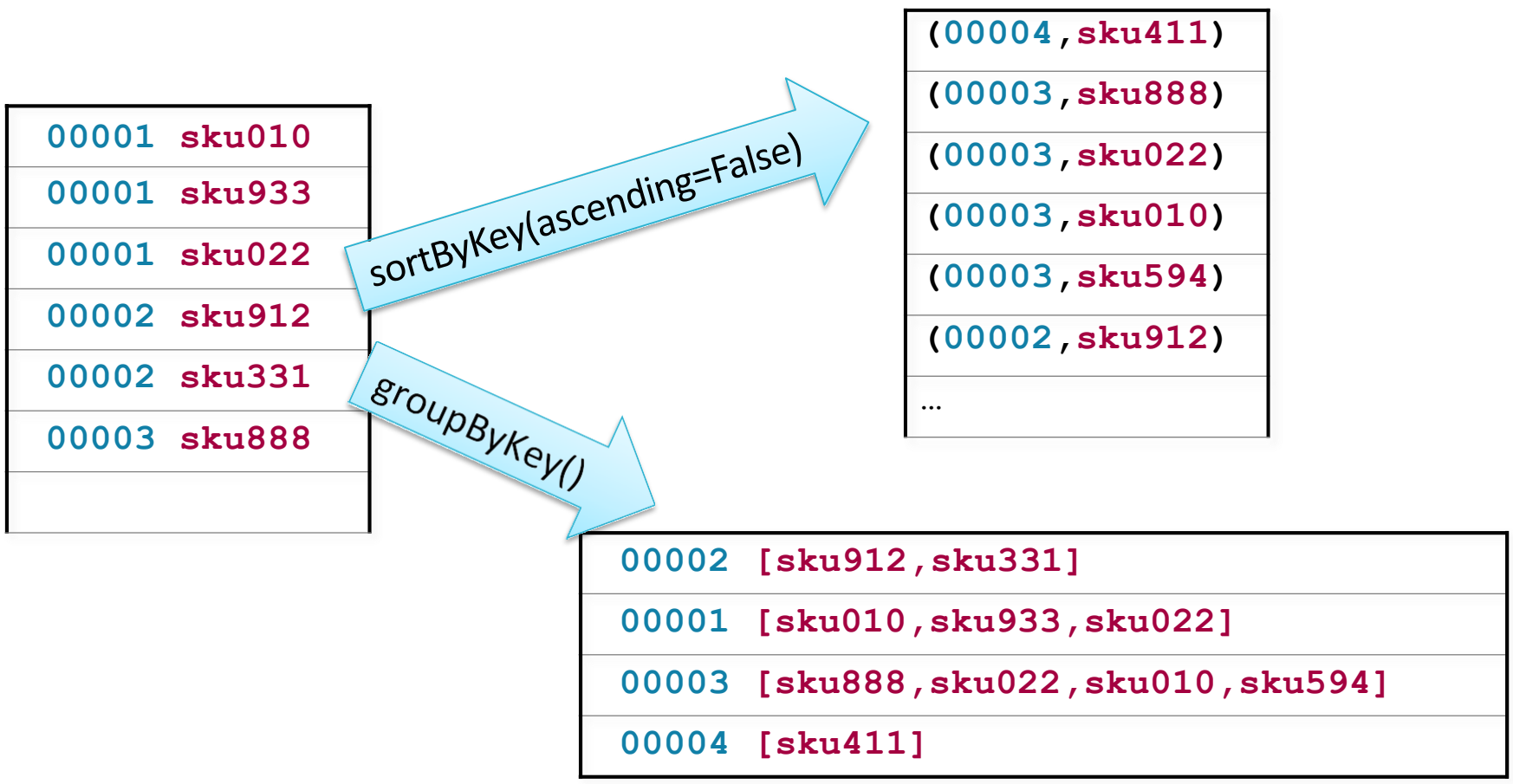
# Why Do We Care About Counting Words?

- **Word count is challenging over massive amounts of data**
  - Using a single compute node would be too time-consuming
  - Number of unique words could exceed available memory
- **Statistics are o_en simple aggregate functions**
  - Distributive in nature
  - e.g., max, min, sum, count
- **MapReduce breaks complex tasks down into smaller elements which can be executed in parallel**
- **Many common tasks are very similar to word count**
  - e.g., log file analysis

# Pair RDD Operations

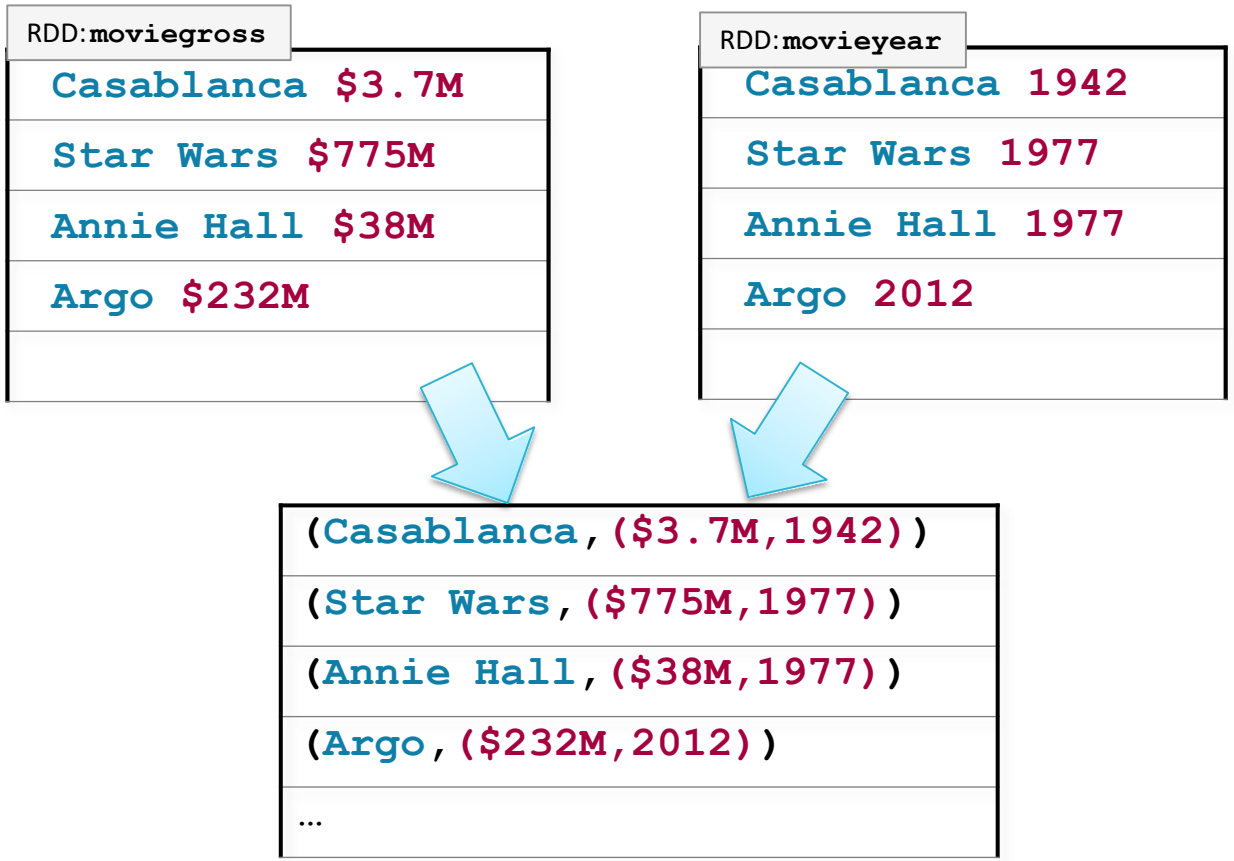
- **In addition to `map` and `reduce` functions, Spark has several operations specific to Pair RDDs**
- **Examples**
  - **`countByKey`** – return a map with the count of occurrences of each key
  - **`groupByKey`** – group all the values for each key in an RDD
  - **`sortByKey`** – sort in ascending or descending order
  - **`join`** – return an RDD containing all pairs with matching keys from two RDDs

# Example: Pair RDD Operations



# Example: Joining by Key

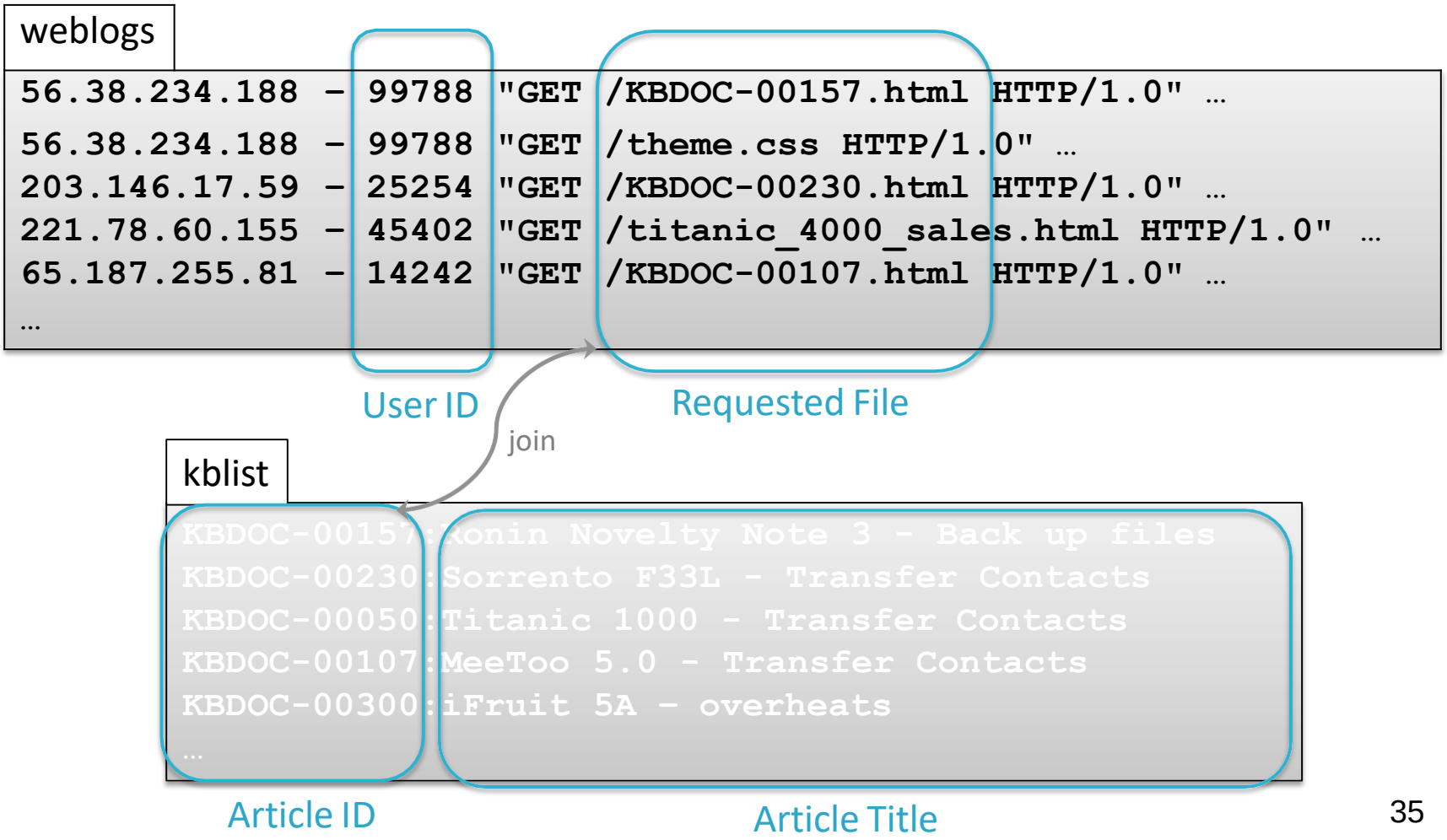
```
> movies = moviegross.join(movieyear)
```



# Using Join

- **A common programming paaern**
  1. Map separate datasets into key8value Pair RDDs
  2. Join by key
  3. Map joined data into the desired format
  4. Save, display, or continue processing…

# Example: Join Web Log With Knowledge Base Articles (1)



# Example: Join Web Log With Knowledge Base Articles (2)

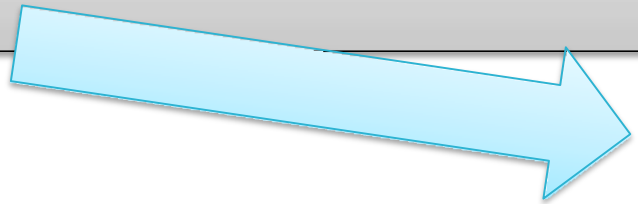
- **Steps**
  1. 
     a. Map web log requests to **`(docid,userid)`**
     b. Map KB Doc index to **`(docid,title)`**
  1. Join by key: **`docid`**
  2. Map joined data into the desired format: **`(userid,title)`**
  3. Further processing: group titles by User ID

# Step 1a: Map Web Log Requests to (docid,userid)

```
> import re
> def getRequestDoc(s):
    return re.search(r'KBDOC-[0-9]*',s).group()

> kbreqs = sc.textFile(logfile) \
    .filter(lambda line: 'KBDOC-' in line) \
    .map(lambda line: (getRequestDoc(line), line.split(' ')[2])) \
    .distinct()
```

```
56.38.234.188 - 99788 "GET /KBDOC-00157.html HTTP/1.0" …
56.38.234.188 - 99788 "GET /theme.css HTTP/1.0" …
203.146.17.59 - 25254 "GET /KBDOC-00230.html HTTP/1.0" …
221.78.60.155 - 45402 "GET /titanic_4000_sales.html …
65.187.255.81 - 14242 "GET /KBDOC-00107.html HTTP/1.…
…
```



kbreqs

| | |
|---|---|
| KBDOC-00157 | 99788 |
| KBDOC-00203 | 25254 |
| KBDOC-00107 | 14242 |
| | |

# Step 1b: Map KB Index to (docid,title)

```
> kblist = sc.textFile(kblistfile) \
  .map(lambda line: line.split(':')) \
  .map(lambda fields: (fields[0],fields[1]))
```

```
KBDOC-00157:Ronin Novelty Note 3 - Back up files
KBDOC-00230:Sorrento F33L - Transfer Contacts
KBDOC-00050:Titanic 1000 - Transfer Contacts
KBDOC-00107:MeeToo 5.0 - Transfer Contacts
KBDOC-00206:iFruit 5A – overheats
…
```

kblist

| |
|---|
| KBDOC-00157 Ronin Novelty Note 3 - Back up files |
| KBDOC-00230 Sorrento F33L - Transfer Contacts |
| KBDOC-00050 Titanic 1000 - Transfer Contacts |
| KBDOC-00107 MeeToo 5.0 - Transfer Contacts |
| |

# Step 2: Join By Key `docid`

```
> titlereqs = kbreqs.join(kblist)
```

kbreqs

| | |
|---|---|
| KBDOC-00157 | 99788 |
| KBDOC-00230 | 25254 |
| KBDOC-00107 | 14242 |

kblist

| | |
|---|---|
| KBDOC-00157 | Ronin Novelty Note 3 - Back up files |
| KBDOC-00230 | Sorrento F33L - Transfer Contacts |
| KBDOC-00050 | Titanic 1000 - Transfer Contacts |
| KBDOC-00107 | MeeToo 5.0 - Transfer Contacts |

| | |
|---|---|
| KBDOC-00157 | (99788,Ronin Novelty Note 3 - Back up files) |
| KBDOC-00230 | (25254,Sorrento F33L - Transfer Contacts) |
| KBDOC-00107 | (14242,MeeToo 5.0 - Transfer Contacts) |

# Step 3: Map Result to Desired Format (userid,title)

```
> titlereqs = kbreqs.join(kblist) \
    .map(lambda (docid,(userid,title)): (userid,title))
```

| |
|---|
| (KBDOC-00157,(99788,Ronin Novelty Note 3 - Back up files)) |
| (KBDOC-00230,(25254,Sorrento F33L - Transfer Contacts)) |
| (KBDOC-00107,(14242,MeeToo 5.0 - Transfer Contacts)) |
| … |

| |
|---|
| 99788,Ronin Novelty Note 3 - Back up files |
| 25254,Sorrento F33L - Transfer Contacts |
| 14242,MeeToo 5.0 - Transfer Contacts |
| |

# Step 4: Continue Processing – Group Titles by User ID

```
> titlereqs = kbreqs.join(kblist) \
    .map(lambda (docid,(userid,title)): (userid,title)) \
    .groupByKey()
```

| |
|---|
| 99788 Ronin Novelty Note 3 - Back up files |
| 25254 Sorrento F33L - Transfer Contacts |
| 14242 MeeToo 5.0 - Transfer Contacts |
| |

| |
|---|
| (99788,[Ronin Novelty Note 3 - Back up files,<br>Ronin S3 - overheating]) |
| (25254,[Sorrento F33L - Transfer Contacts]) |
| (14242,[MeeToo 5.0 - Transfer Contacts,<br>MeeToo 5.1 - Back up files,<br>iFruit 1 - Back up files,<br>MeeToo 3.1 - Transfer Contacts]) |
| … |

UNIVERSITAT DE BARCELONA

# Example Output

```
> for (userid,titles) in titlereqs.take(10):
    print 'user id: ',userid
    for title in titles: print '\t',title

user id: 99788
  Ronin Novelty Note 3 - Back up files
  Ronin S3 - overheating
user id: 25254
  Sorrento F33L - Transfer Cont
user id: 14242
  MeeToo 1.- Transfer Contac  t
  MeeToo 2.- Back up files
  iFruit 1 - Back up files
  MeeToo 3.1 - Transfer Contacts
...
```

```
99788 [Ronin Novelty Note 3 - Back up files,
       Ronin S3 - overheating]

25254,[Sorrento F33L - Transfer Contacts]

14242,[MeeToo 5.0 - Transfer Contacts,
       MeeToo 5.1 - Back up files,
       iFruit 1 - Back up files,
       MeeToo 3.1 - Transfer Contacts]
```

# Aside: Anonymous Function Parameters

! **Python and Scala paaern matching can help improve code readability**

Python

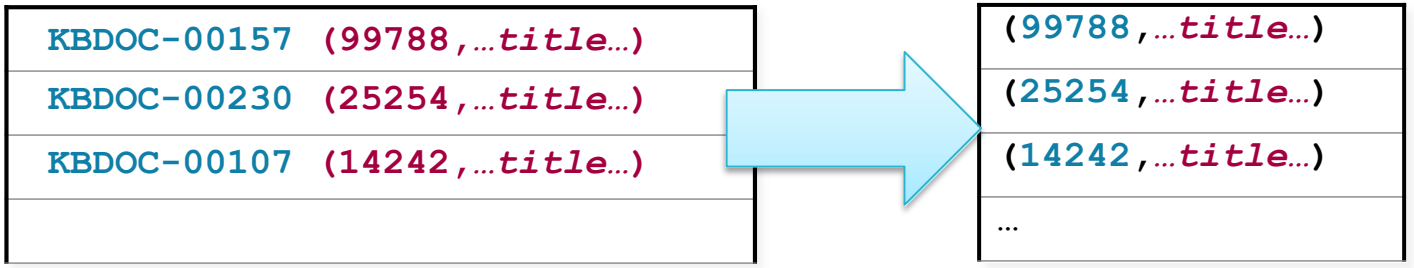
```
> map(lambda (docid,(userid,title)): (userid,title))
```

Scala

```
> map(pair => (pair._2._1,pair._2._2))
```

OR

```
> map{case (docid,(userid,title)) => (userid,title)}
```



# Other Pair Operations

- **Some other pair operations**
  - **`keys`** – return an RDD of just the keys, without the values
  - **`values`** – return an RDD of just the values, without keys
  - **`lookup(`*`key`*`)`** – return the value(s) for a key
  - **`leftOuterJoin`**, **`rightOuterJoin`** – join, including keys defined only in the left or right RDDs respectively
  - **`mapValues`**, **`flatMapValues`** – execute a function on just the values, keeping the key the same
- **See the `PairRDDFunctions` class Scaladoc for a full list**

# Key Points

- **Pair RDDs are a special form of RDD consisting of Key#Value pairs (tuples)**
- **Spark provides several operations for working with Pair RDDs**
- **MapReduce is a generic programming model for distributed processing**
  - Spark implements MapReduce with Pair RDDs
  - Hadoop MapReduce and other implementations are limited to a single Map and Reduce phase per job
  - Spark allows flexible chaining of map and reduce operations
  - Spark provides operations to easily perform common MapReduce algorithms like joining, sorting, and grouping

UNIVERSITAT DE BARCELONA

# Bases de dades avançades

## curs 2017/18

Enric Biosca Trias ebiosca@ub.edu

Dept. Matemàtica Aplicada i Anàlisi.

**Universitat de Barcelona**