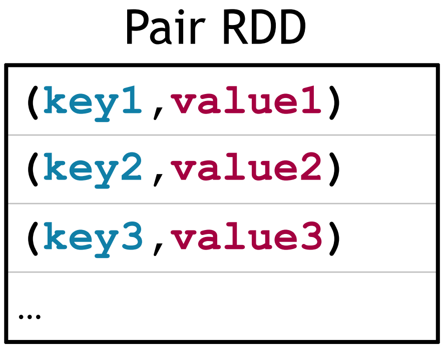
Class 6: Pair RDDs

New York University

Summer 2017

- **Key-Value Pair RDDs**
- Map-Reduce
- Other Pair RDD Operations

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



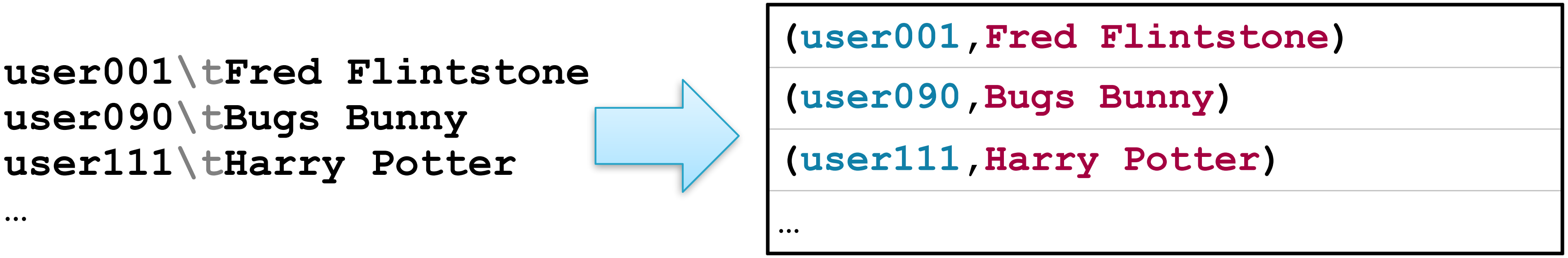
Pair RDD

```
(key1,value1)
(key2,value2)
(key3,value3)
…
```

- **Common functions for creating Pair RDDs**
  - **`map`**
  - **`flatMap`** / **`flatMapValues`**
  - **`keyBy`**

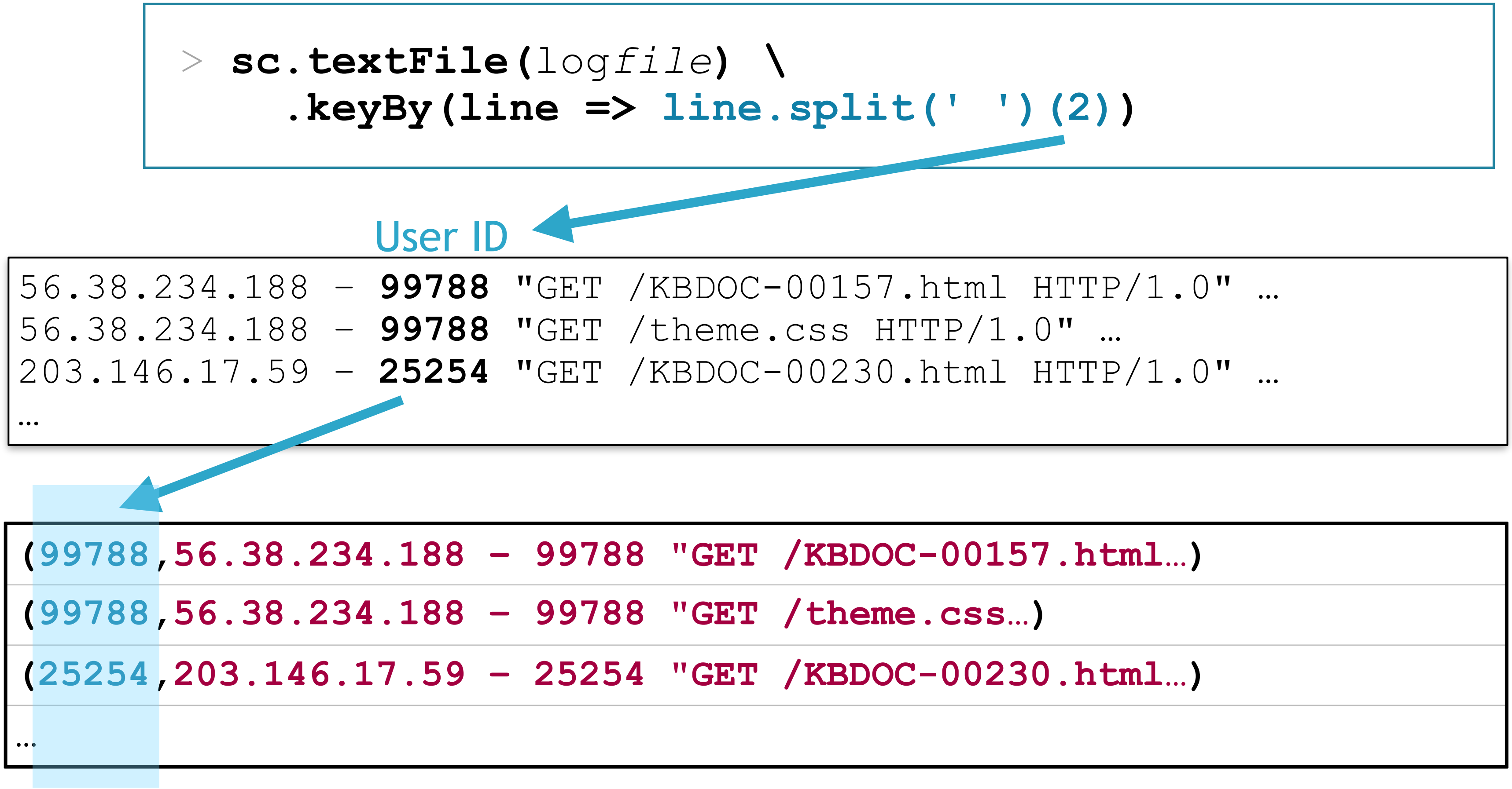
- **Example: Create a Pair RDD from a tab-separated file**

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



```
user001\tFred Flintstone
user090\tBugs Bunny
user111\tHarry Potter
…
```

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



```
> 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…)
…
```

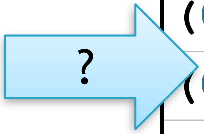
- **How would you do this?**
  - Input: a list of postal codes with latitude and longitude
  - Output: postal code (key) and lat/long pair (value)

```
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))
…
```

```
> sc.textFile("file").
   map(line => line.split('\t')).
   map(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))
…
```

- Key-Value Pair RDDs
- **Map-Reduce**
- Other Pair RDD Operations

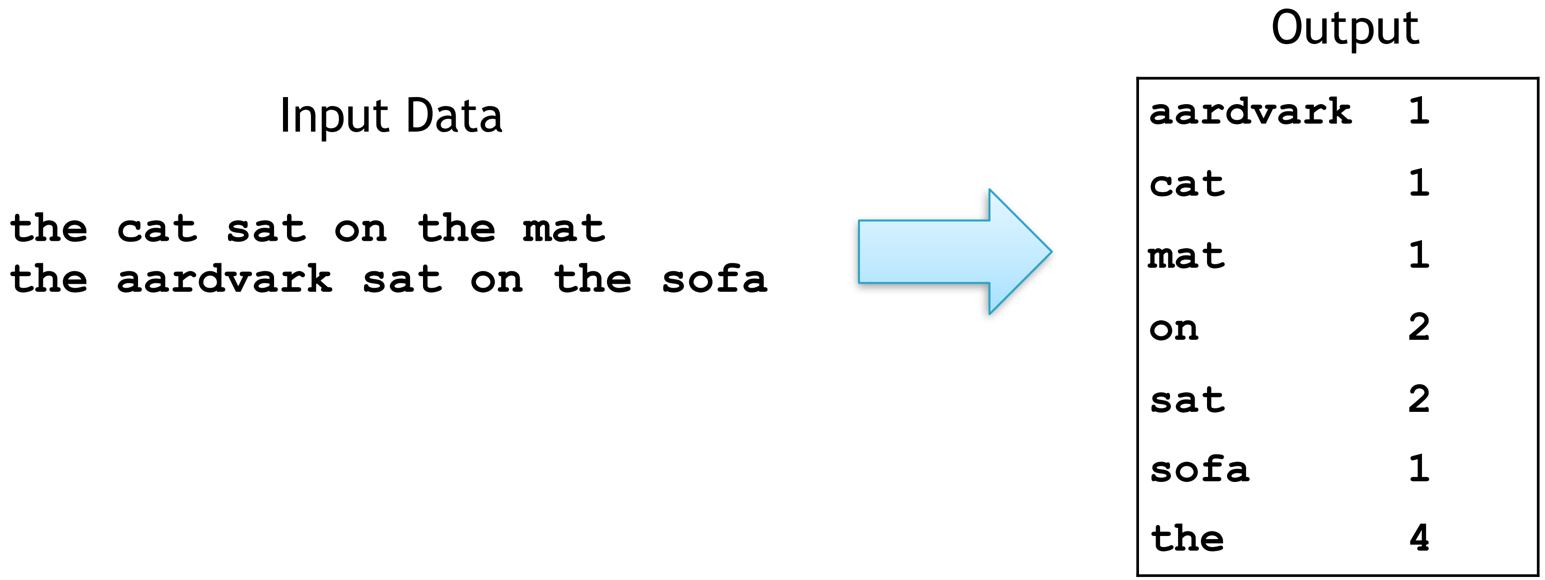
- **Map-reduce is a common programming model**
  - Easily applicable to distributed processing of large data sets
- **Hadoop MapReduce is a well known implementation**
  - Somewhat limited
    - Each job has one Map phase, one Reduce phase
    - Job output is saved to files
- **Spark implements map-reduce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

- **Map-reduce in Spark works on Pair RDDs**

- **Map phase**
  - Operates on one record at a time
  - Maps each record to one or more new records
  - e.g. **`map`**, **`flatMap`**, **`filter`**, **`keyBy`**

- **Reduce phase**
  - Works on map output, or output from other tools in same key-value format
  - Consolidates multiple records
  - e.g. **`reduceByKey`**, **`sortByKey`**, **`mean`**

- **How would you do this?**
  - Input: Text lines
  - Output: Word count



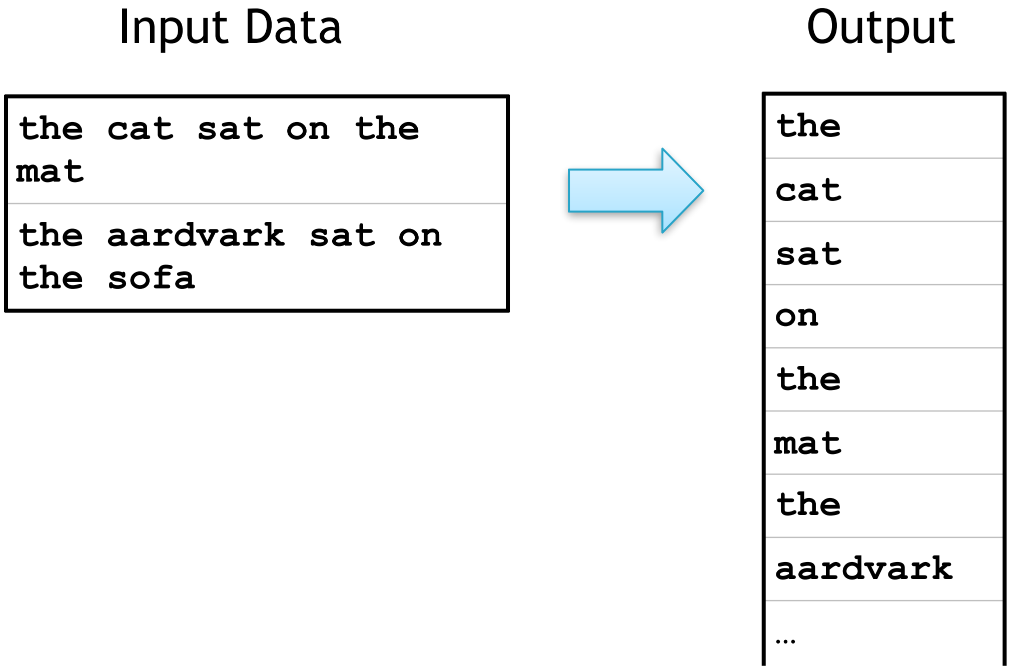
Input Data

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

Output

```
aardvark   1
cat        1
mat        1
on         2
sat        2
sofa       1
the        4
```

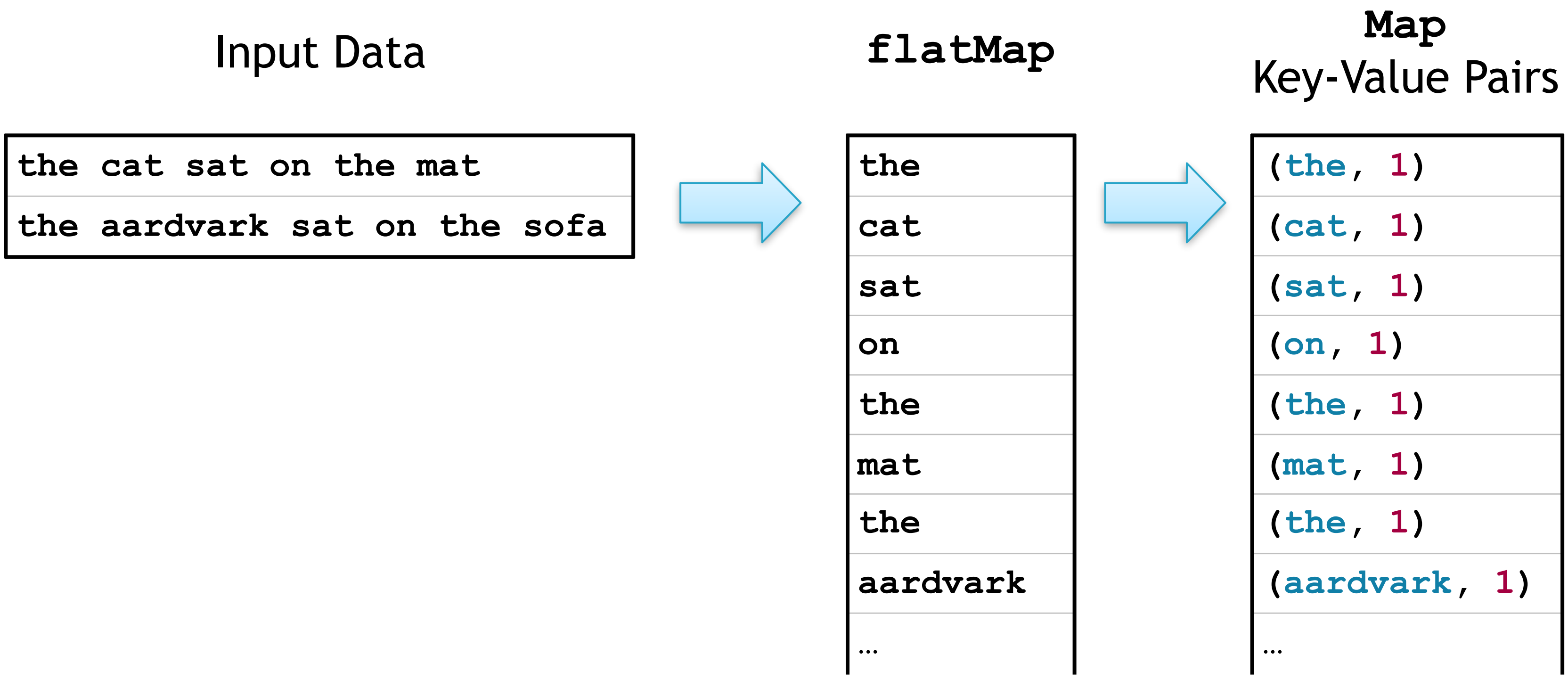
- **Use `flatMap`**



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

Output
`the`
`cat`
`sat`
`on`
`the`
`mat`
`the`
`aardvark`
…

- **`flatMap`, followed by `map` to map the word as key and output with a '1'**
  - **This is the same technique you use with Hadoop MapReduce when implementing WordCount**



Input Data

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

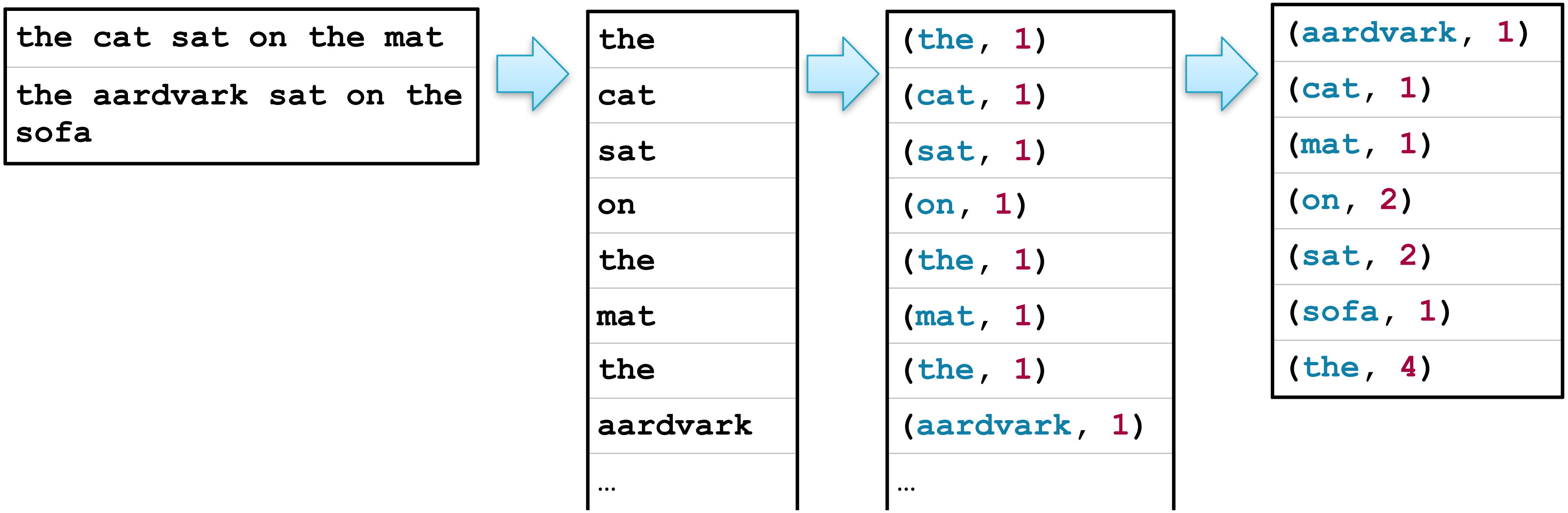
`flatMap`

```
the
cat
sat
on
the
mat
the
aardvark
…
```

`Map`
Key-Value Pairs

```
(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
…
```

- After `flatMap` and `map` we have key-value pairs that can be the input to a reduce
- followed by reduceByKey specifying addition of two values, and that sum gets added to another value, and so on:



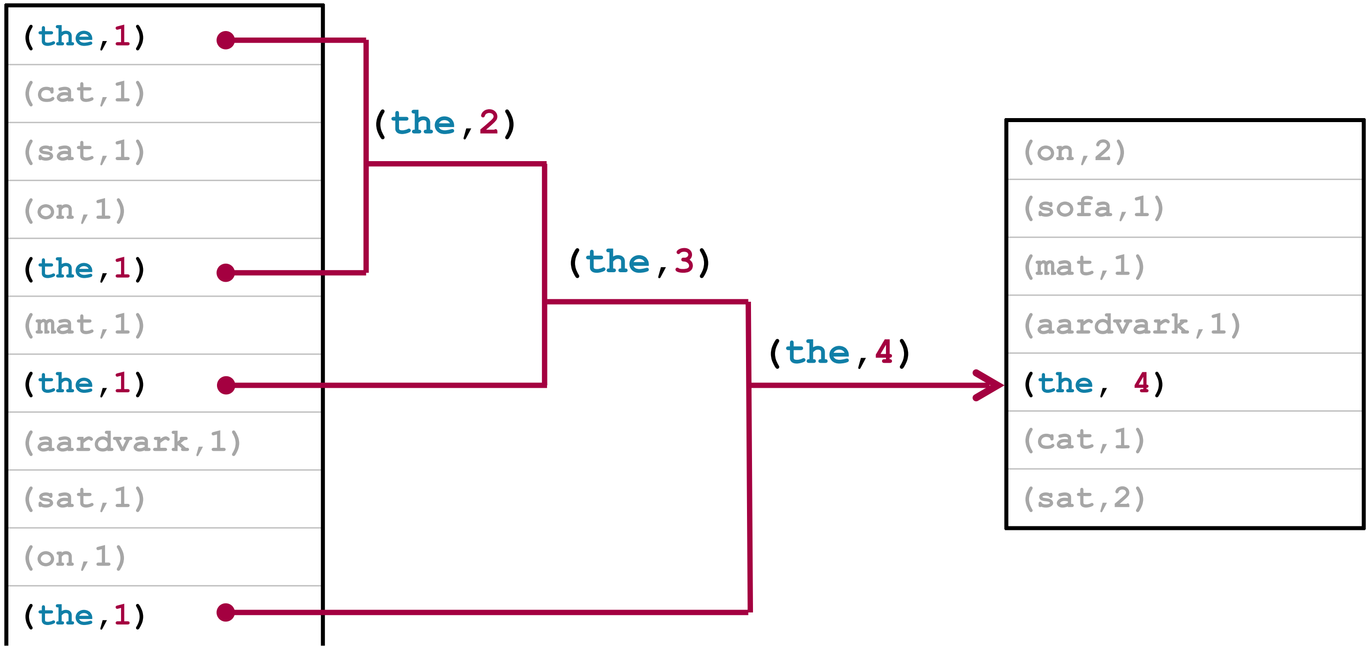
```
the cat sat on the mat
the aardvark sat on the sofa
```

```
the
cat
sat
on
the
mat
the
aardvark
…
```

```
(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
…
```

```
(aardvark, 1)
(cat, 1)
(mat, 1)
(on, 2)
(sat, 2)
(sofa, 1)
(the, 4)
```

- **The function passed to `reduceByKey` combines values from two keys**
  - Function must be binary



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

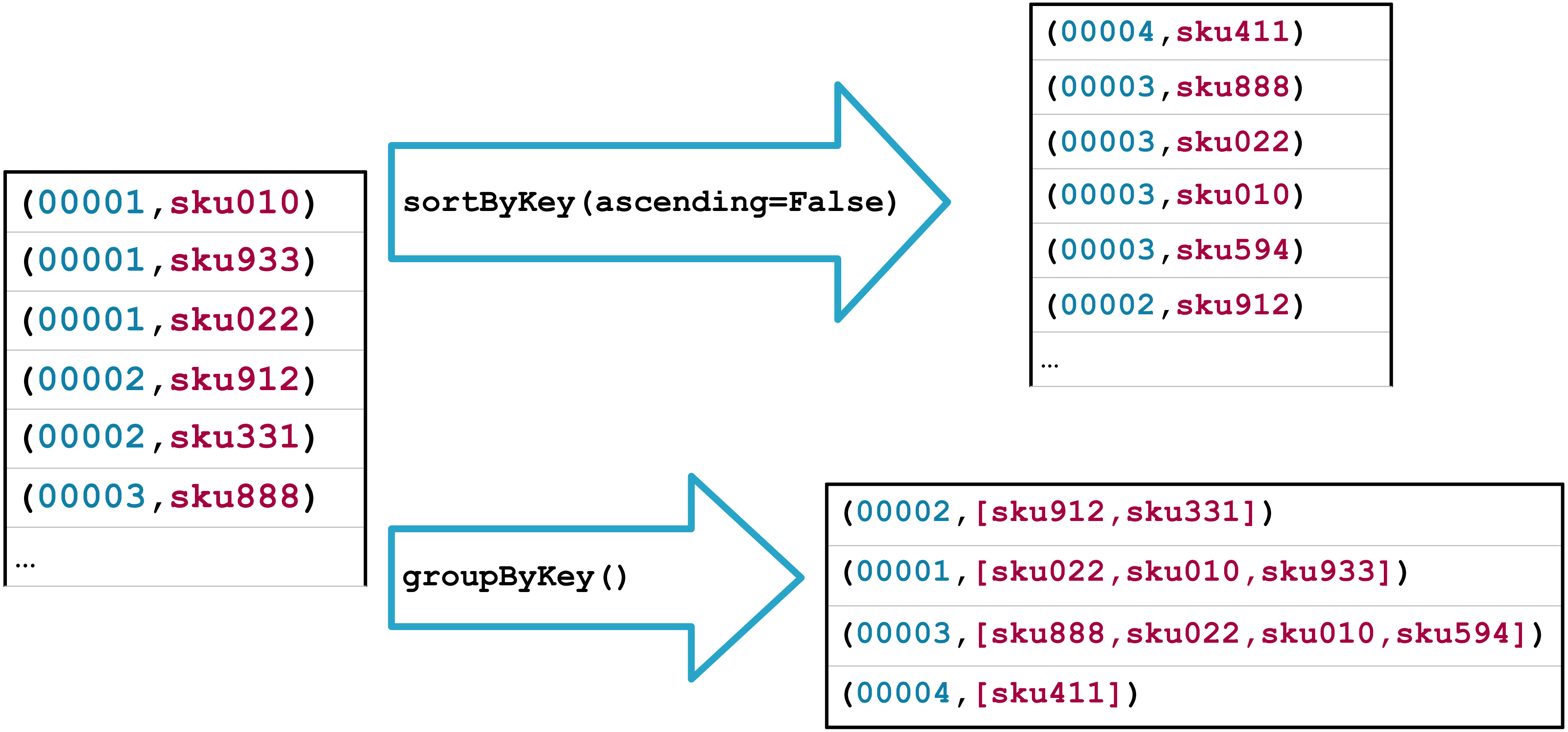
OR

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

- **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 often simple aggregate functions**
  - Distributive in nature
  - e.g., max, min, sum, count
- **Map-reduce 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

- Key-Value Pair RDDs
- Map-Reduce
- **Other Pair RDD Operations**

- **In addition to `map` and `reduce` functions, Spark has several operations specific to Pair RDDs**
  - **`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



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

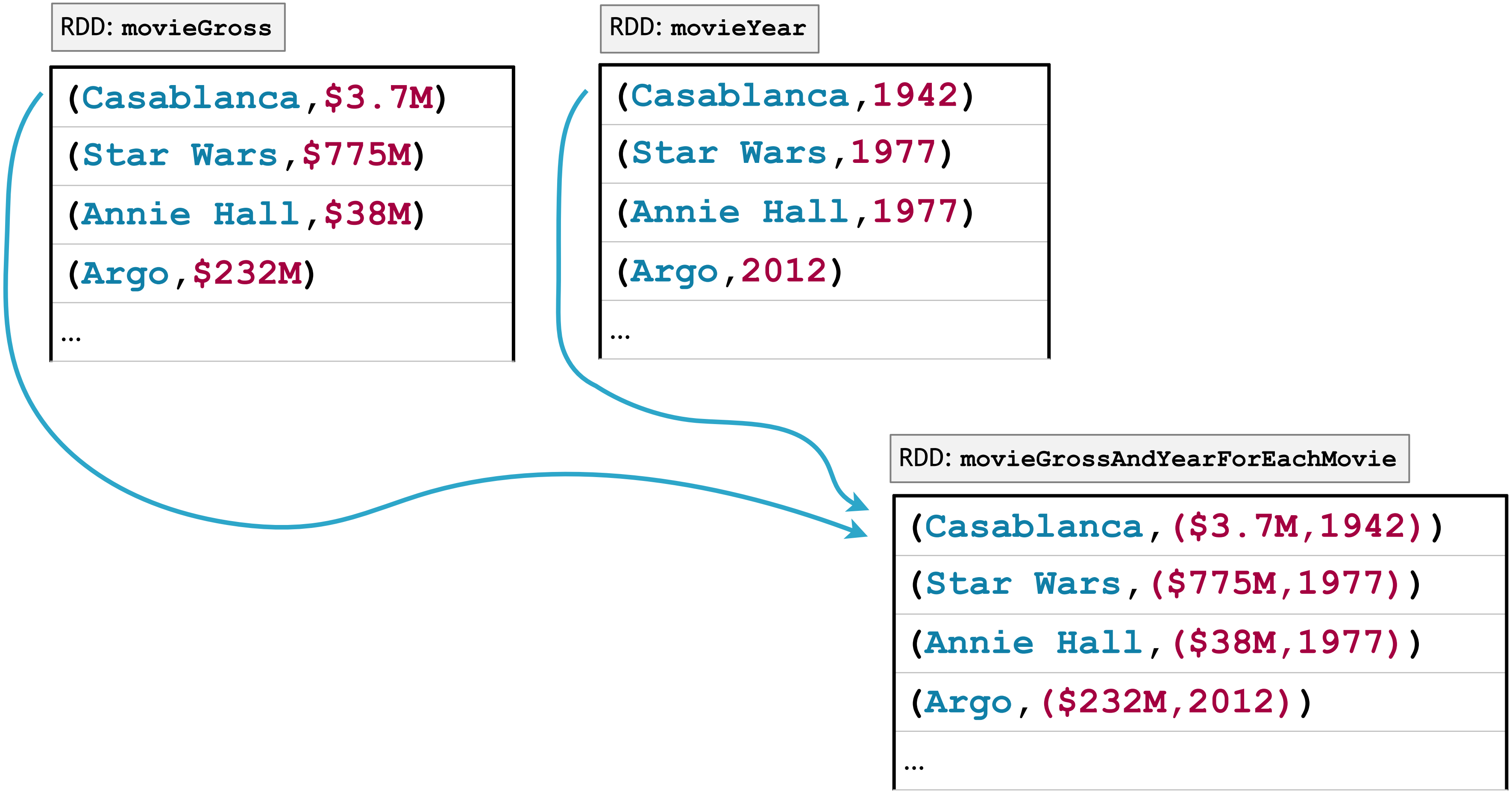
sortByKey(ascending=False)

```
(00004,sku411)
(00003,sku888)
(00003,sku022)
(00003,sku010)
(00003,sku594)
(00002,sku912)
…
```

groupByKey()

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

Using `join`



RDD: `movieGross`

```
(Casablanca,$3.7M)
(Star Wars,$775M)
(Annie Hall,$38M)
(Argo,$232M)
…
```

RDD: `movieYear`

```
(Casablanca,1942)
(Star Wars,1977)
(Annie Hall,1977)
(Argo,2012)
…
```

RDD: `movieGrossAndYearForEachMovie`

```
(Casablanca,($3.7M,1942))
(Star Wars,($775M,1977))
(Annie Hall,($38M,1977))
(Argo,($232M,2012))
…
```

- **Some other pair RDD 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`** , **`fullOuterJoin`** – join, including keys defined in the left, right or either RDD respectively
  - **`mapValues`**, **`flatMapValues`** – execute a function on just the values, keeping the key the same
- **See the `PairRDDFunctions` class Scaladoc for a full list**

Homework

Study for midterm exam.