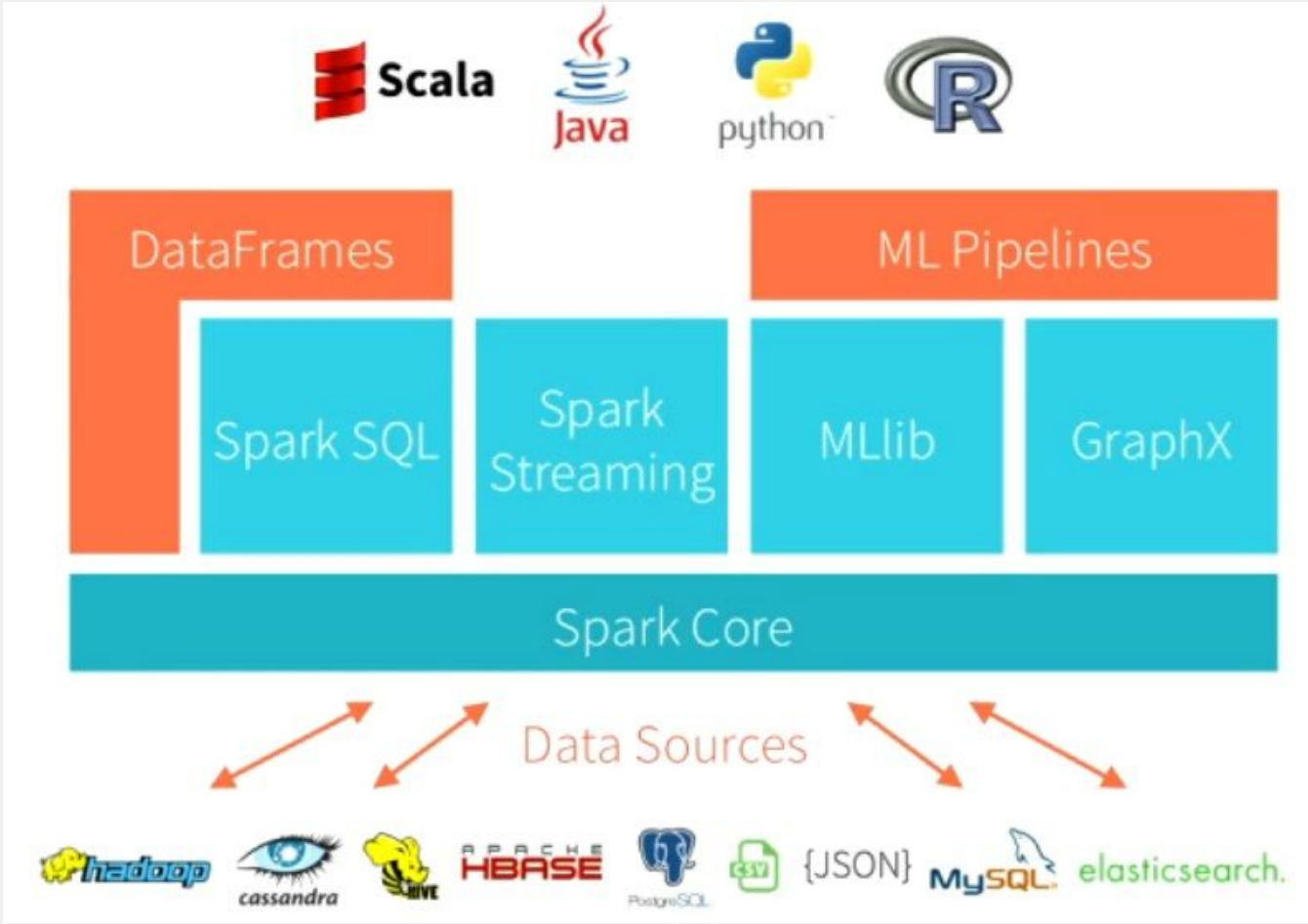
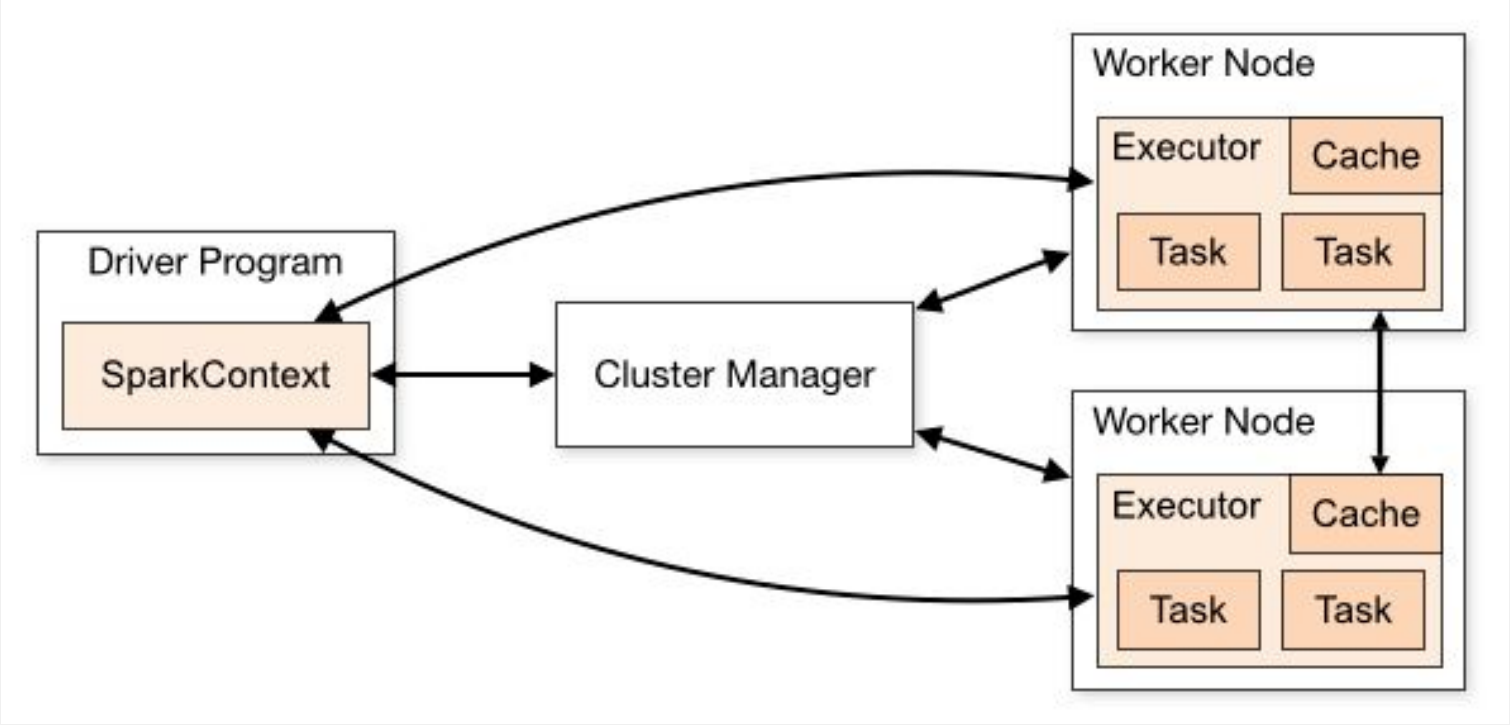
Spark RDD

Apache Spark

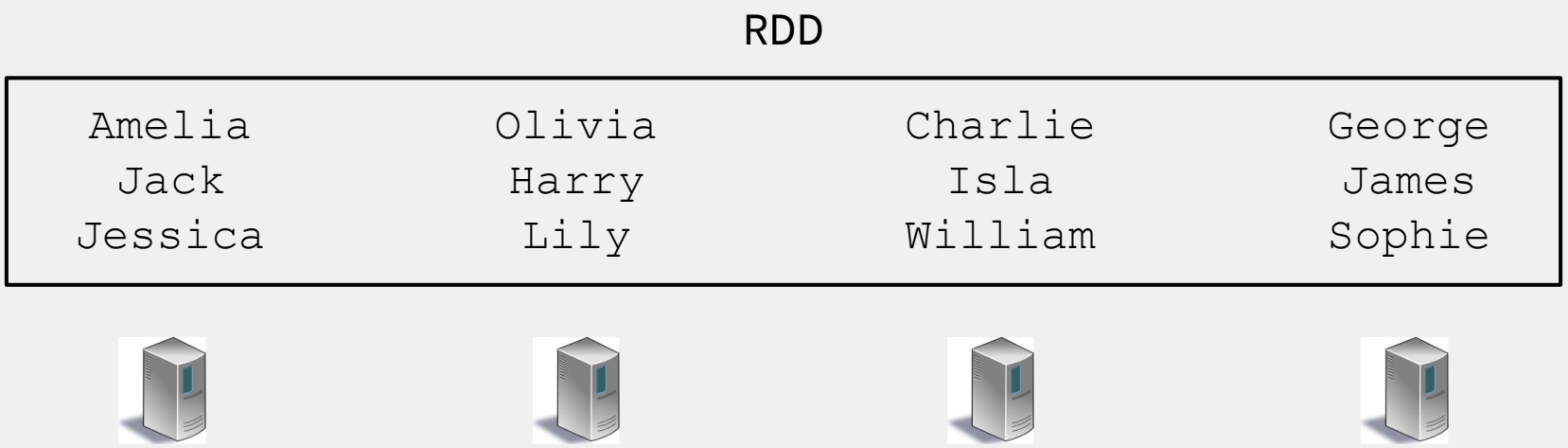


Apache Spark Architecture

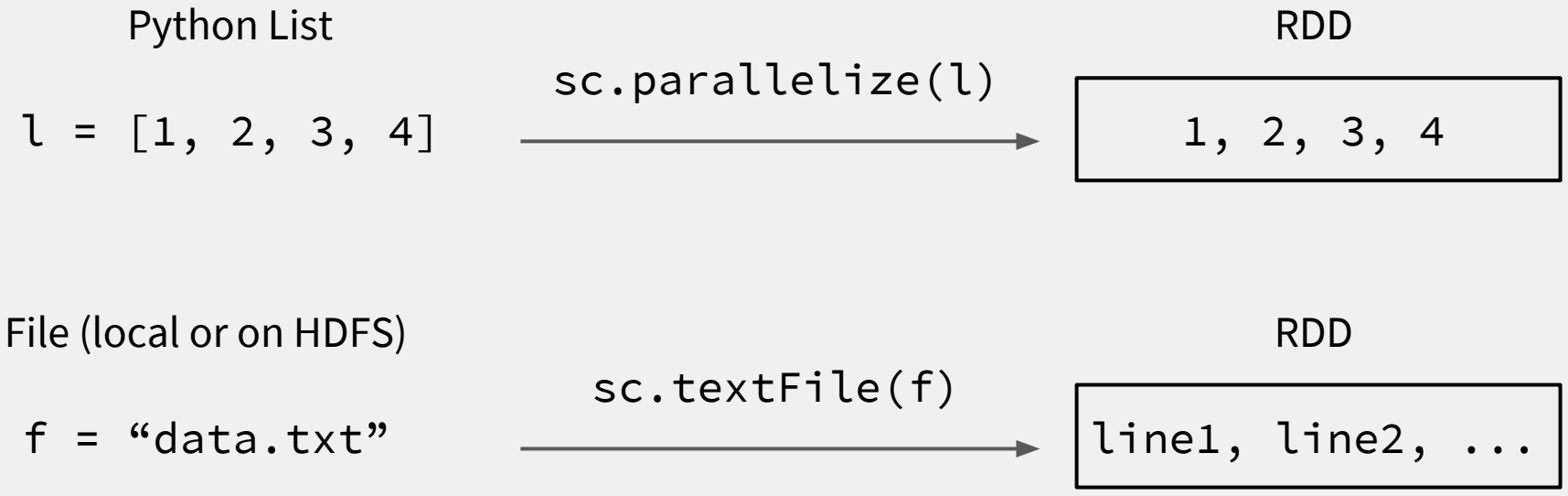


Apache Spark: Resilient Distributed Datasets (RDDs)

- Most basic abstraction
- Collection of elements (e.g. data points)
- Divided across the cluster



Creating an RDD



Basic RDD manipulation

- `rdd.count()`: returns the number of elements in your RDD
- `rdd.take(n)`: returns the first n elements in your RDD

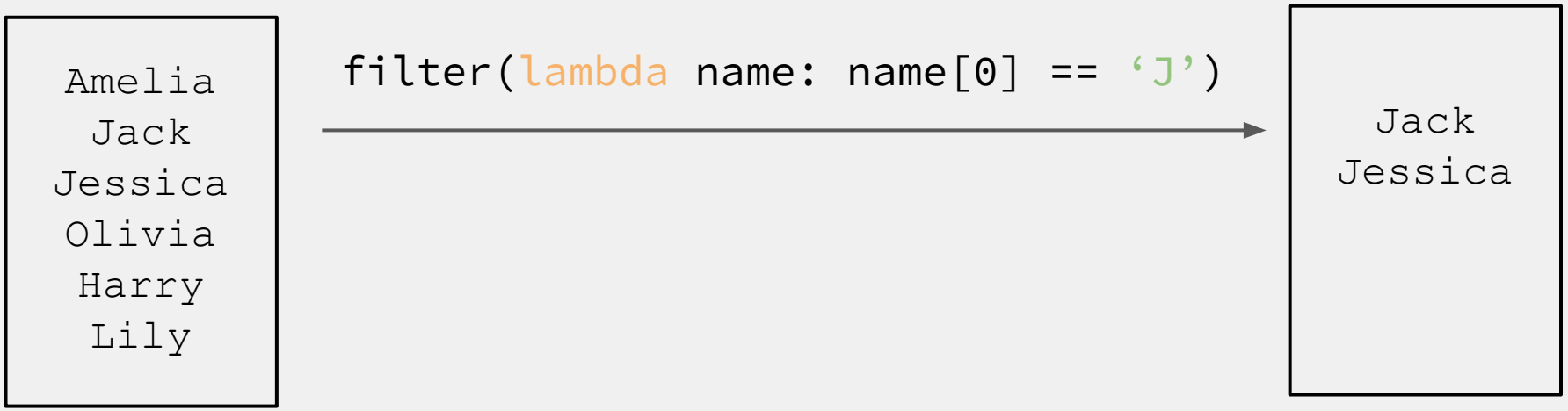
Transformations and Actions

- Transformations: operations that return a new RDD
- Actions: operations that return a value to your Python programs

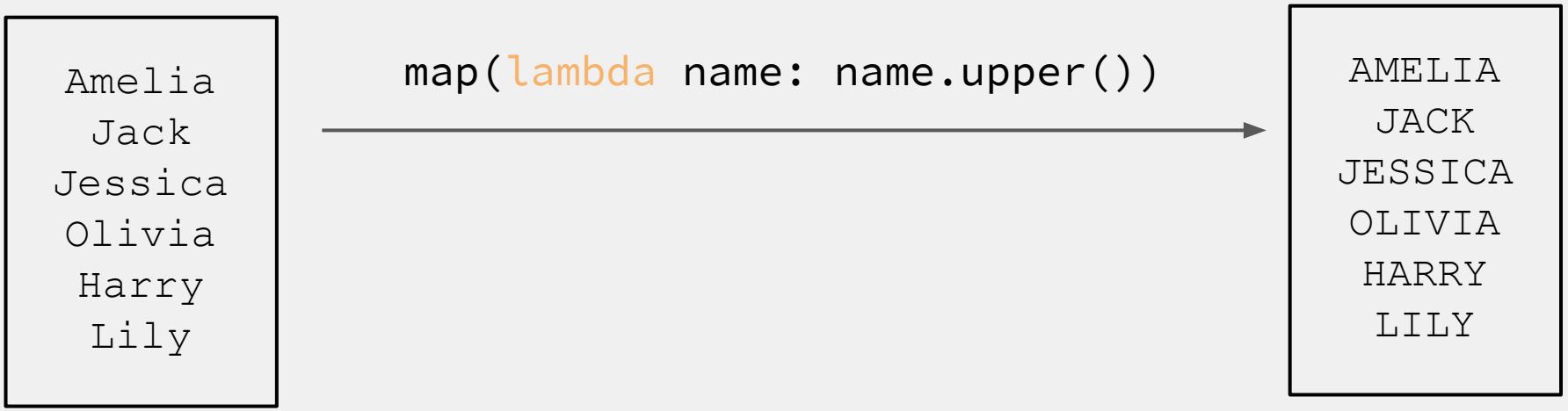
Quiz:

- Is `count()` an action or a transformation?
- Is `take()` an action or a transformation?

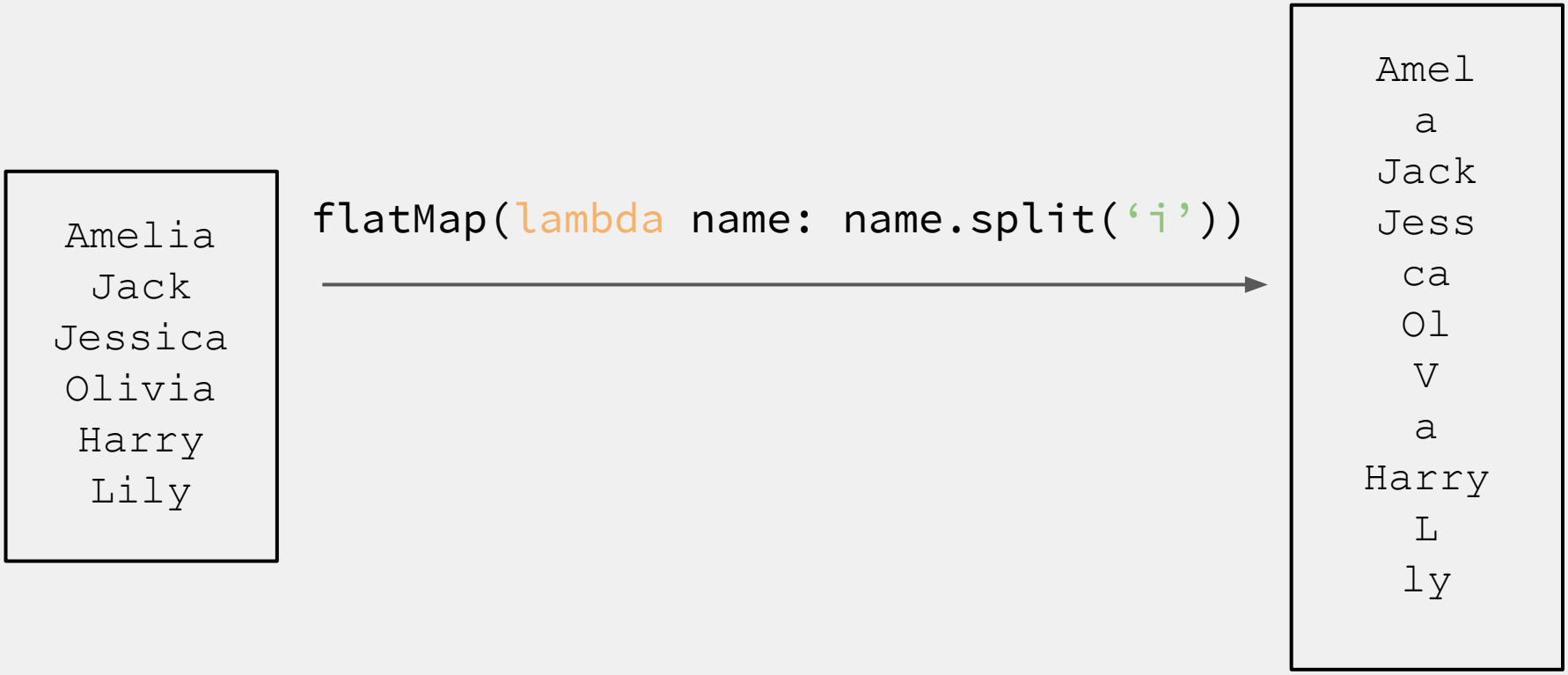
Common transformations: `filter()`



Common transformations: map()

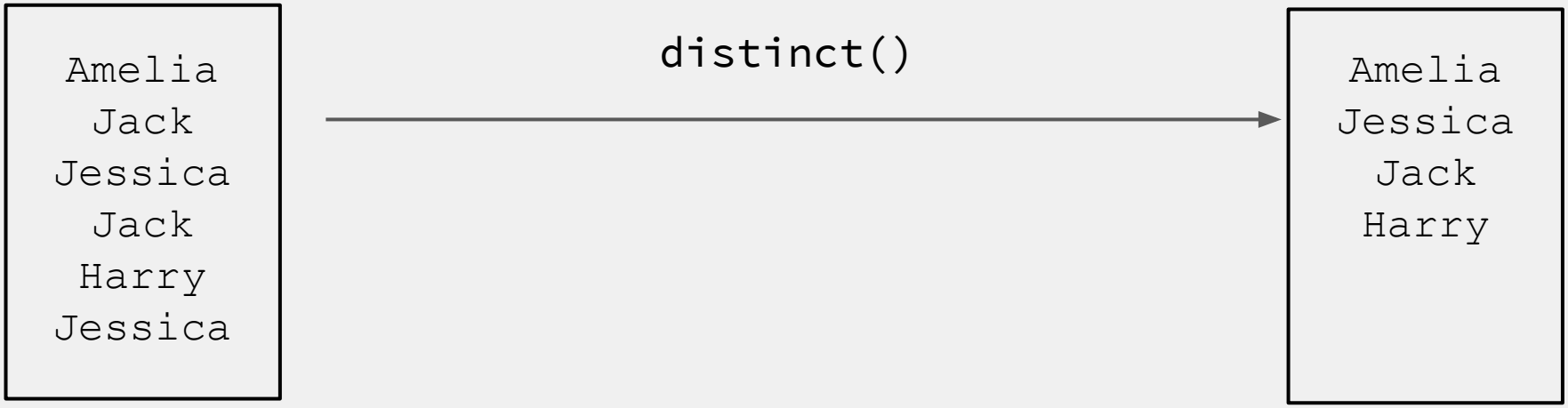


Common transformations: `flatMap()`



Note: the function **split** returns a list

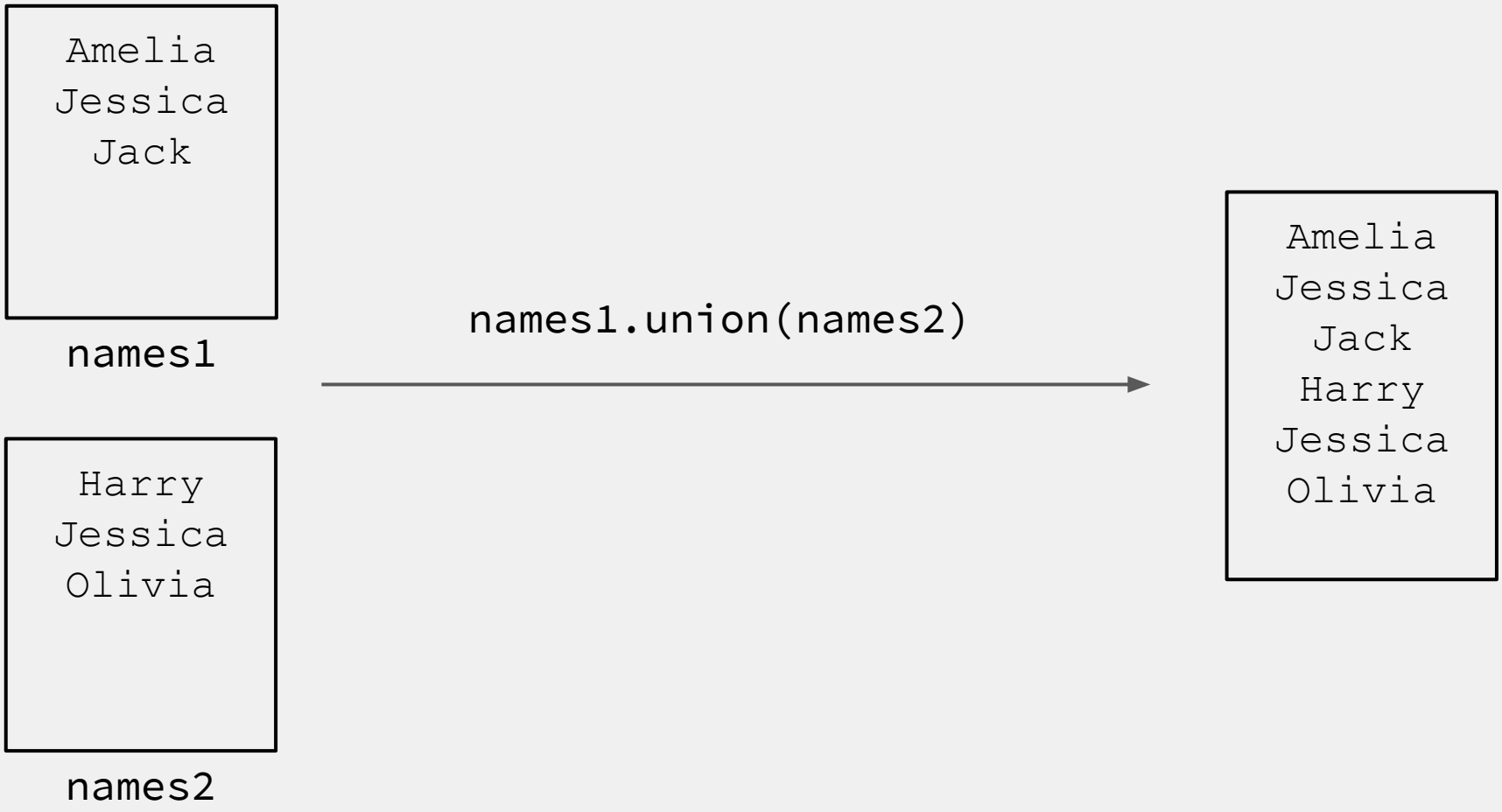
Common transformations: `distinct()`



Hands-on session

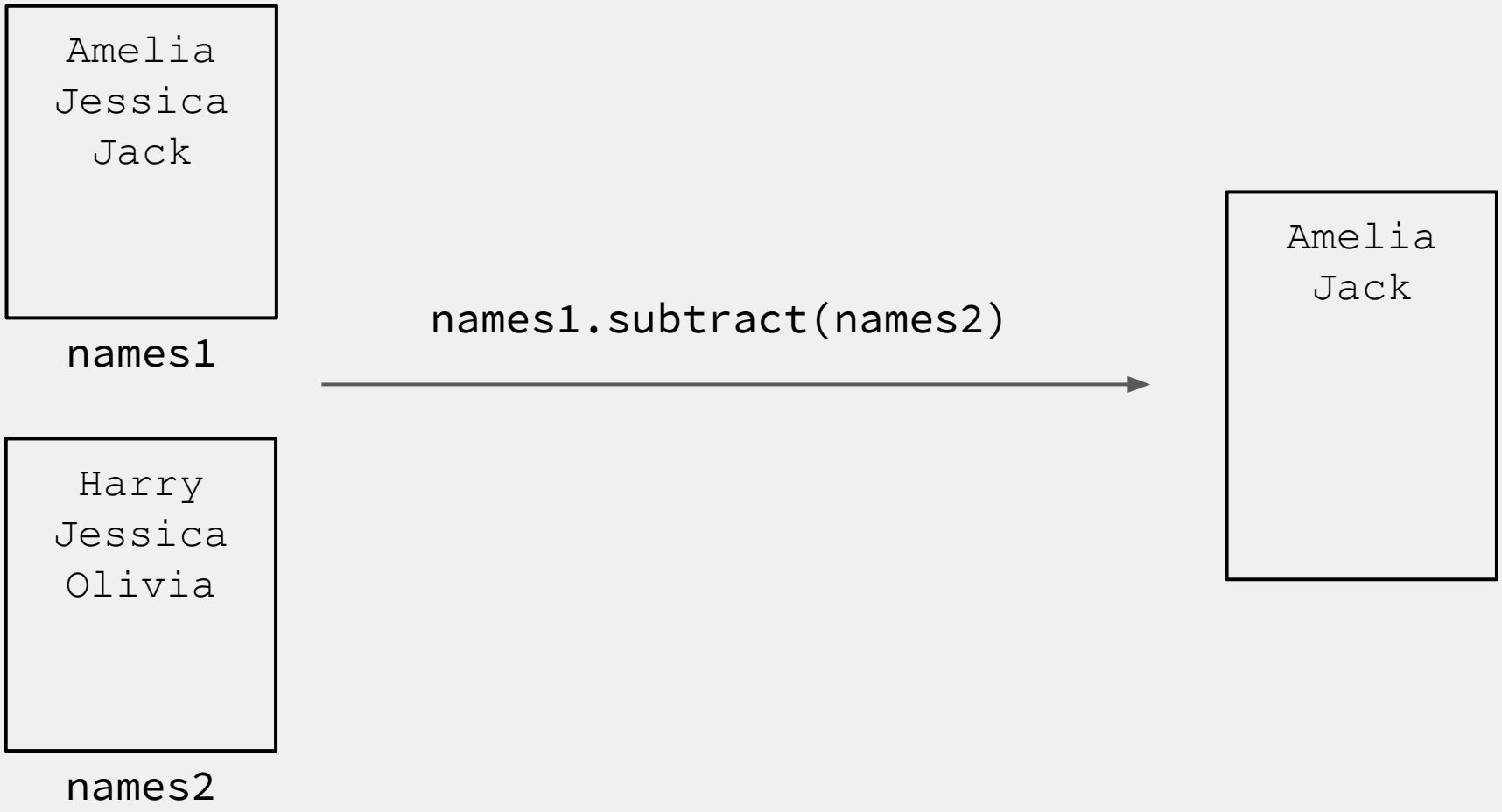
>>> Create RDDs, `filter()`, `map()` and `distinct()`

Set-like transformations: `union()`



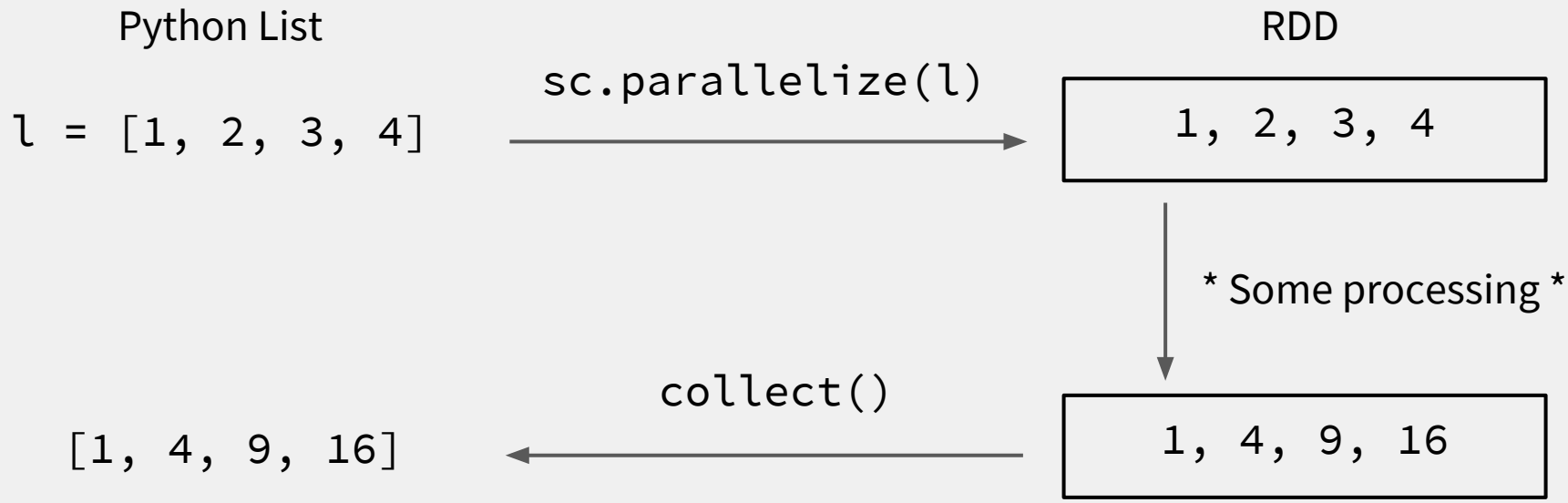
Set-like transformations: `intersection()`

Set-like transformations: `subtract()`



Common actions

- `count()`
- `take()`
- `collect()`: opposite of `parallelize()`, turns an RDD into a Python list



Common actions: `reduce()`

Takes as input a function that processes two elements and returns one

Hands-on session

>>> Set like transformations, `reduce()` & bonus

RDD Basics: recap

- RDDs are Sparks main abstraction to hold collections
- They have a rich API to process them
  - Transformations (`filter()`, `map()`, `union()`...) return processed RDDs
  - Actions (`count()`, `collect()`, `reduce()`...) return values to the Python program

Lazy evaluation

All transformations are *lazy*, they are only executed once an action gets called.

Similar in concept to a recipe:

- Transformations (e.g. `filter()`, `map()`) add instructions onto the recipe
- Actions (e.g. `count()`, `collect()`) process the entire recipe to return a result

Lazy evaluation

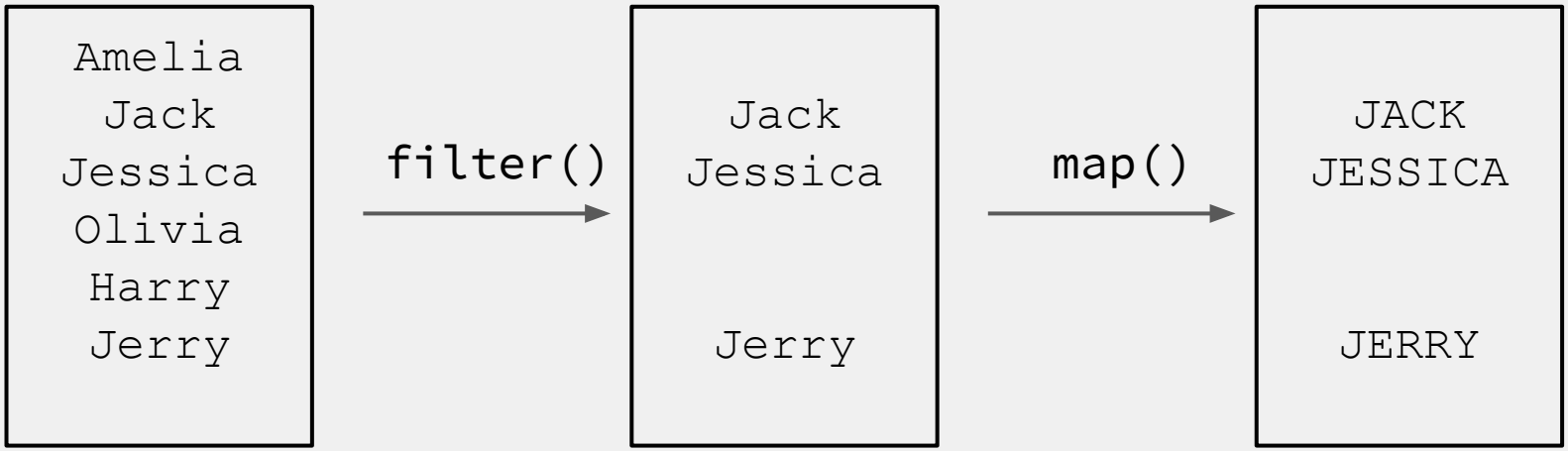
Can lead to inefficiencies:

```
lines = sc.textFile("war-and-peace.txt")
lines.count()
lines.count()
```

Cache an RDD in memory using `persist()`

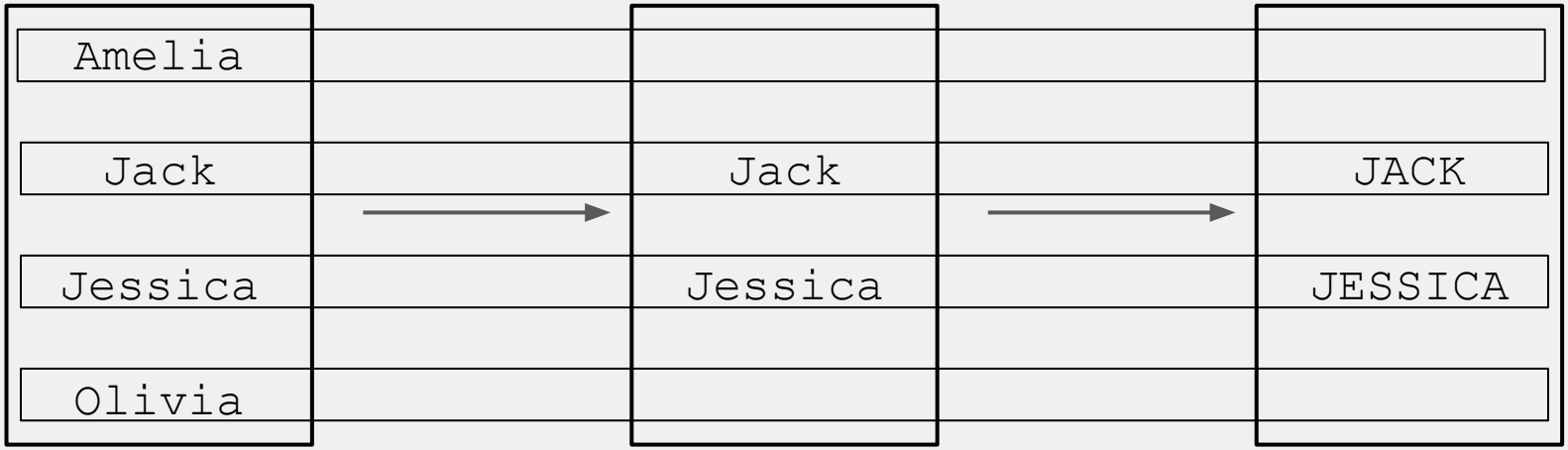
Lazy evaluation

Allows Spark to batch together transformations:



Lazy evaluation

Allows Spark to batch together transformations:



Pair RDDs

Spark's way of storing key/value pairs

Normal RDD where elements are tuples

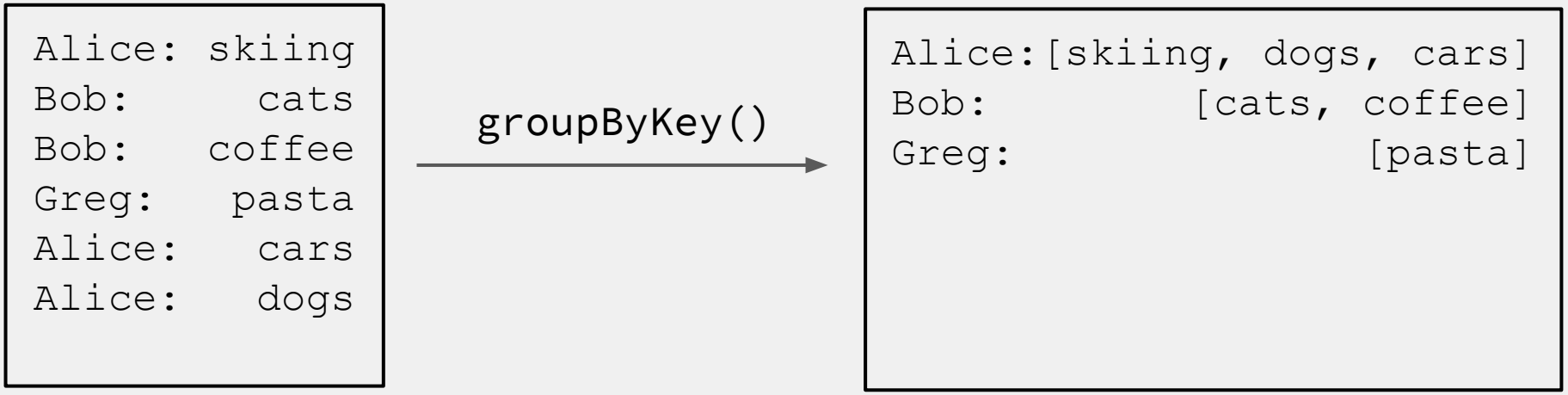
```
(key, value)
```

Note that in Python, the key and value of a tuple can be accessed as:

```
key = key_value[0]

value = key_value[1]
```

Transformations on pair RDDs: `groupByKey()`



Transformations on pair RDDs: `reduceByKey()`

Hands-on session

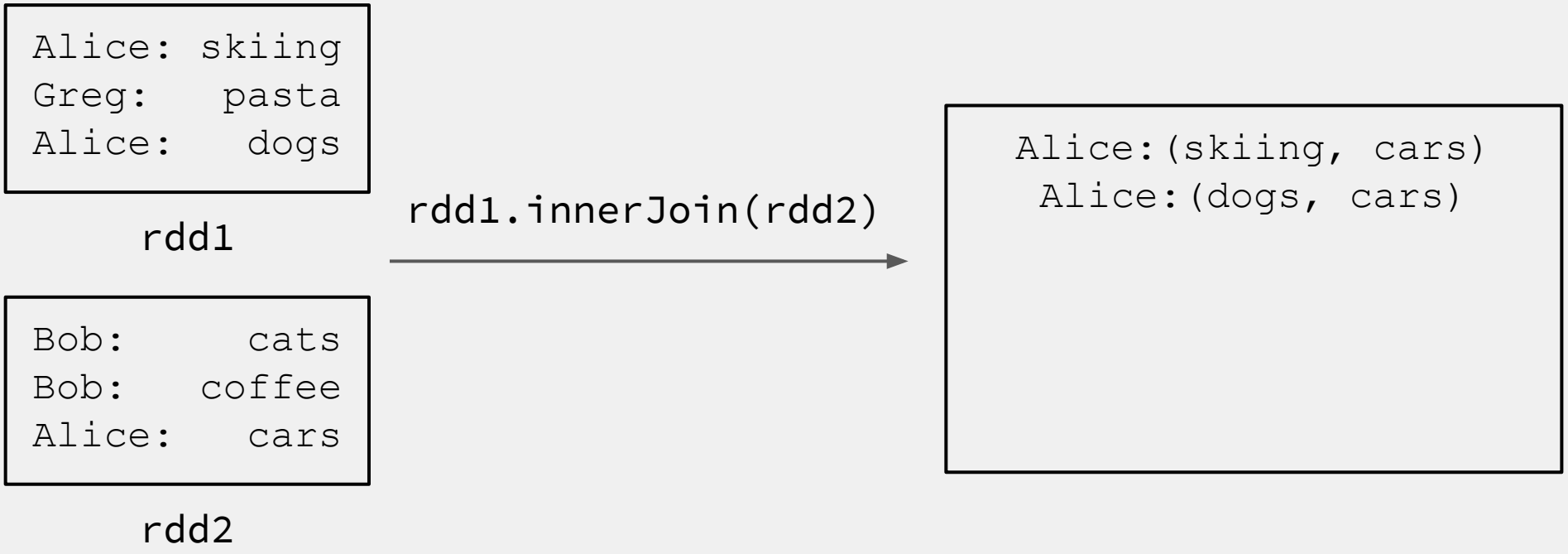
>>> Word count in Spark, average of each key

Some useful pair RDD transformations

- `keys()`: an RDD of the keys
- `values()`: an RDD of the values
- `mapValues(func)` and `flatMapValues(func)`: Apply `func` onto the values

Joins: `innerJoin()`

Keeps all keys that are in both RDD:



Joins: `leftOuterJoin()`

Keeps all keys in the first RDD:
e.g. A list of all customers being linked with their purchase history

```
Alice: skiing
Greg:   pasta
Alice:   dogs
```

rdd1

```
Bob:     cats
Bob:   coffee
Alice:   cars
```

rdd2


```
Alice: (skiing, cars)
Alice:   (dogs, cars)
Greg:   (pasta, None)
```

Actions of pair RDDs

- `lookup(key)`: returns the values associated with a key
- `collectAsMap()`: returns a Python dictionary

Quiz >>> Implement lookup as a transformation

- `lookup(key)`: returns the values associated with a key

RDD Further: recap

- Transformations are lazy
- Pair RDDs store keys and values
- They have a few specific transformations
  - `groupByKey(), reduceByKey()...`
- Joins are useful to merge pair RDDs together