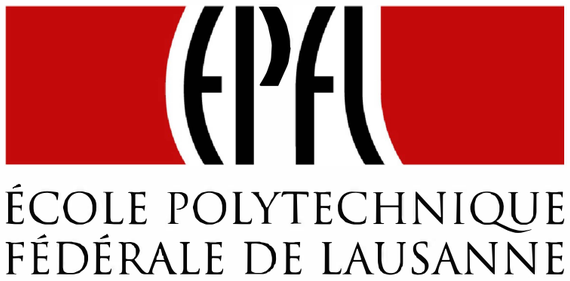


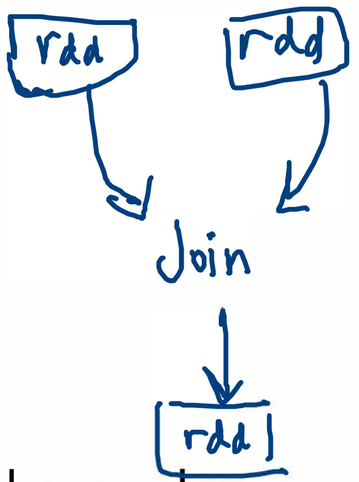

# Joins

Big Data Analysis with Scala and Spark

Heather Miller

# Joins

Joins are another sort of transformation on Pair RDDs. They're used to combine multiple datasets They are one of the most commonly-used operations on Pair RDDs!




There are two kinds of joins:

- Inner joins (`join`)
- Outer joins (`leftOuterJoin`/`rightOuterJoin`)

The key difference between the two is what happens to the keys when both RDDs don't contain the same key.

For example, if I were to join two RDDs containing different `customerIDs` (the key), the difference between inner/outer joins is what happens to customers whose IDs don't exist in both RDDs.

# Example Dataset...

**Example:** Let's pretend the Swiss Rail company, CFF, has two datasets. One **RDD representing customers and their subscriptions (`abos`), and another** representing customers and cities they frequently travel to (`locations`). **(*E.g.*, gathered from CFF smartphone app.)

Let's assume the following concrete data:

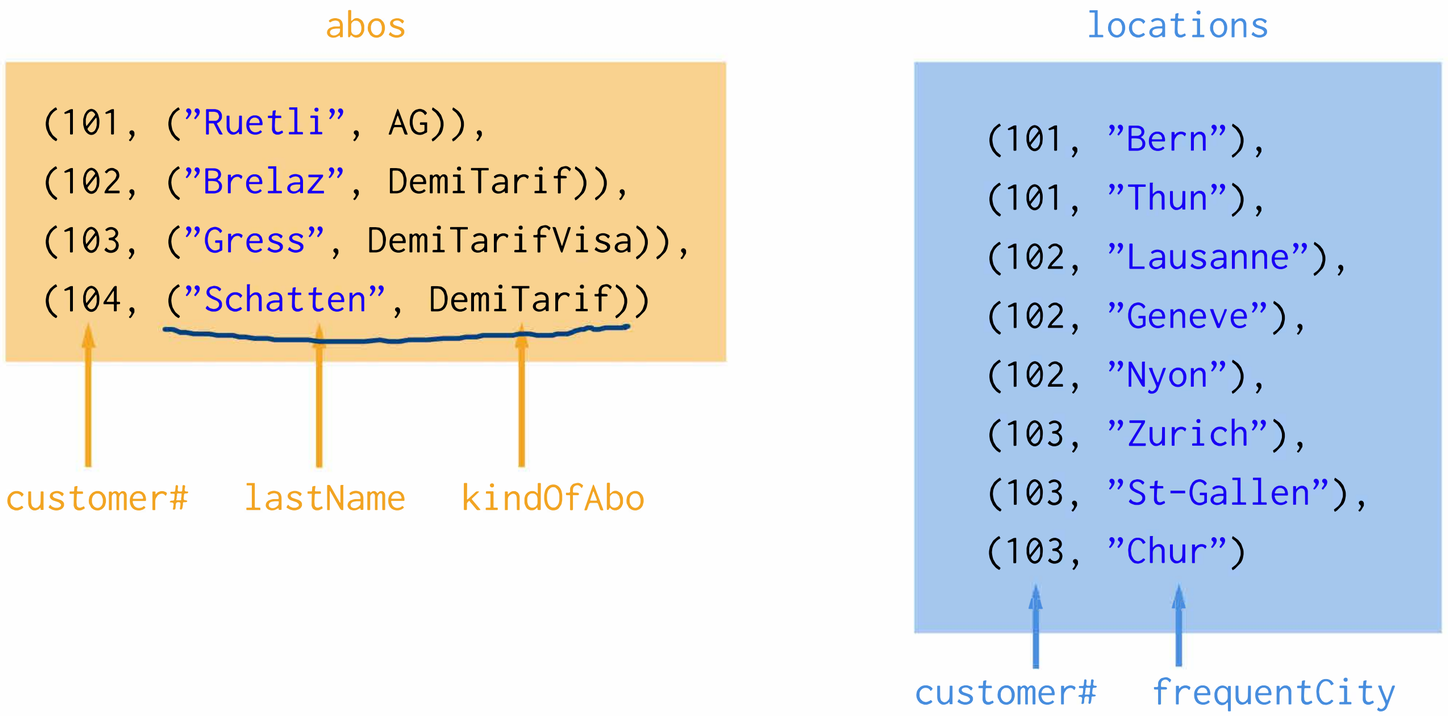
```
val as = List((101, ("Ruetli", AG)), (102, ("Brelaz", DemiTarif)),
              (103, ("Gress", DemiTarifVisa)), (104, ("Schatten", DemiTarif)))
val abos = sc.parallelize(as)

val ls = List((101, "Bern"), (101, "Thun"), (102, "Lausanne"), (102, "Geneve"),
              (102, "Nyon"), (103, "Zurich"), (103, "St-Gallen"), (103, "Chur"))
vals locations = sc.parallelize(ls)
```

# Example Dataset... (2)

**Example:** Let's pretend the CFF has two datasets. One RDD representing customers and their subscriptions (`abos`), and another representing customers and cities they frequently travel to (`locations`). (*E.g.*, gathered from CFF smartphone app.)

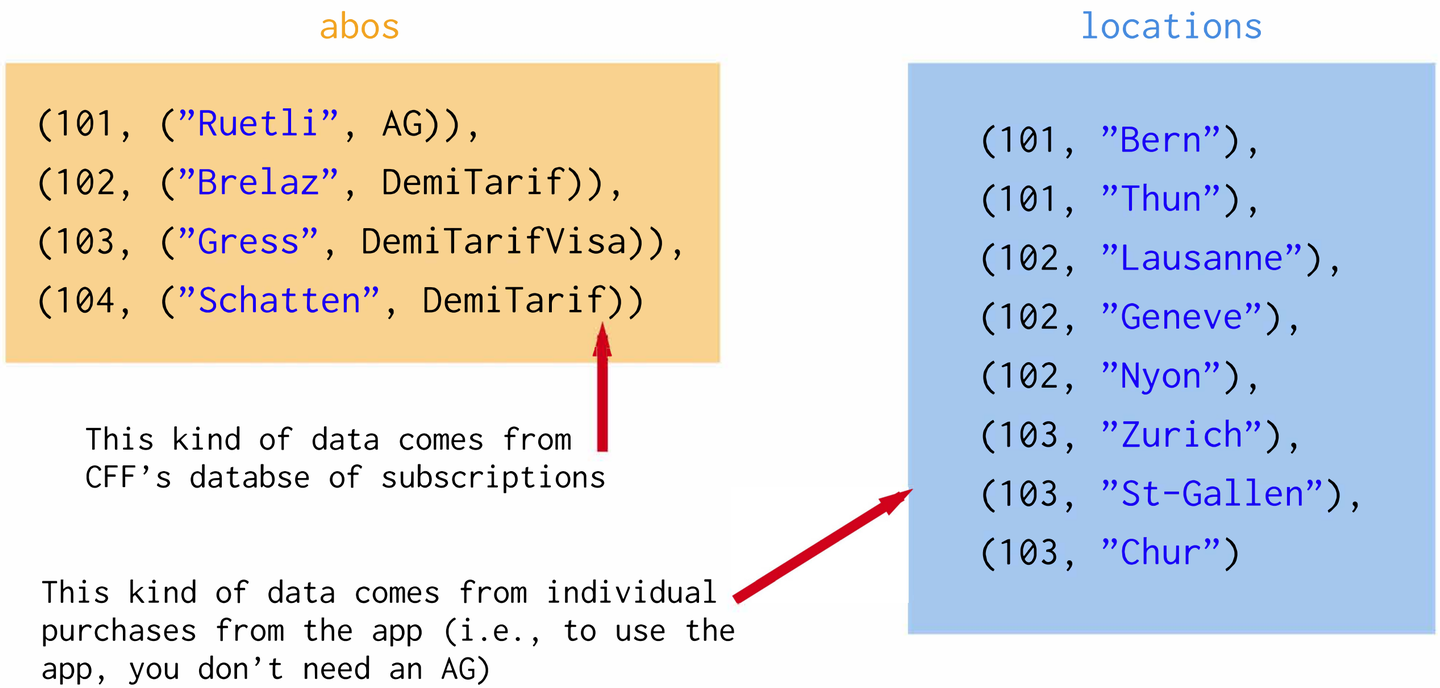
Let's assume the following concrete data: **(visualized)**



# Example Dataset... (3)

**Example:** Let's pretend the CFF has two datasets. One RDD representing customers and their subscriptions (`abos`), and another representing customers and cities they frequently travel to (`locations`). (*E.g.*, gathered from CFF smartphone app.)

Let's assume the following concrete data: **(visualized)**



# Inner Joins (`join`)

Inner joins return a new RDD containing combined pairs whose **keys are present in both input RDDs**.

```
def join[W](other: RDD[(K, W)]): RDD[(K, (V, W))]
```

**Example:** Let's pretend the CFF has two datasets. One RDD representing customers and their subscriptions (`abos`), and another representing customers and cities they frequently travel to (`locations`). (*E.g.*, gathered from CFF smartphone app.)

How do we combine only customers that have a subscription and where there is location info?

```
val abos = ... // RDD[(Int, (String, Abonnement))]
val locations = ... // RDD[(Int, String)]

val trackedCustomers = ???
```

# Inner Joins (`join`)

**Example:** Let's pretend the CFF has two datasets. One RDD representing customers and their subscriptions (`abos`), and another representing customers and cities they frequently travel to (`locations`). (*E.g.*, gathered from CFF smartphone app.)

How do we combine only customers that have a subscription and where there is location info?

```
val abos = ... // RDD[(Int, (String, Abonnement))]
val locations = ... // RDD[(Int, String)]
```

# Inner Joins (join)

**Example continued with concrete data:**

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

```
val trackedCustomers = abos.join(locations)
// trackedCustomers: RDD[(Int, ((String, Abonnement), String))]
```

# Inner Joins (join)

## Example continued with concrete data:

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to combine both RDDs into one:**

**How do we combine only customers that have a subscription and where there is location info?**

# Inner Joins (join)

**Example continued with concrete data:**

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

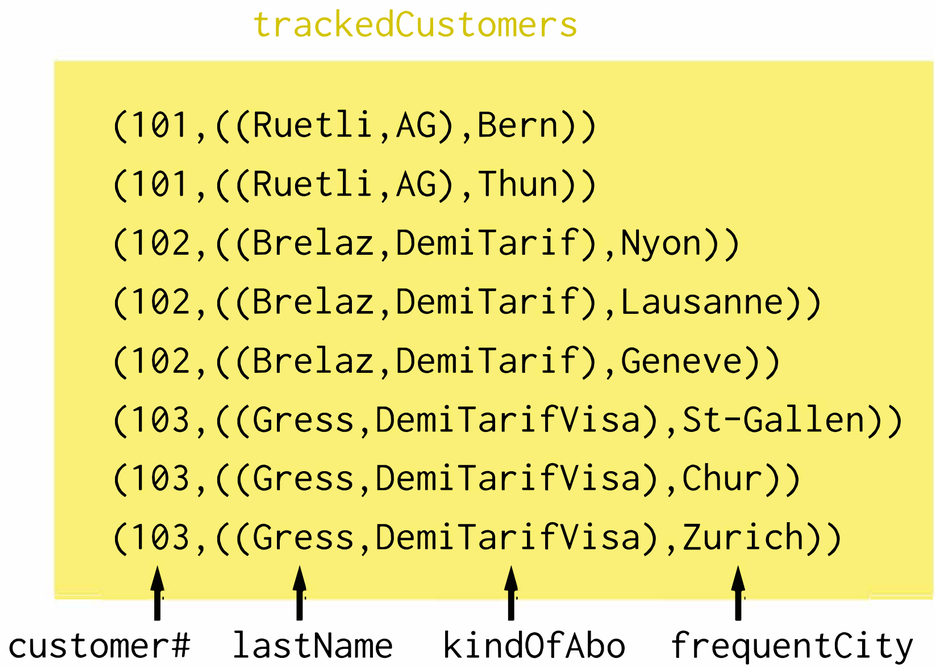
locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to make a new RDD with only these!**

# Inner Joins (join)

**Example continued with concrete data:**




```
val trackedCustomers = abos.join(locations)
// trackedCustomers: RDD[(Int, ((String, Abonnement), String))]
```

# Inner Joins (`join`)

**Example continued with concrete data:**

```
trackedCustomers.collect().foreach(println)
// (101,((Ruetli,AG),Bern))
// (101,((Ruetli,AG),Thun))
// (102,((Brelaz,DemiTarif),Nyon))
// (102,((Brelaz,DemiTarif),Lausanne))
// (102,((Brelaz,DemiTarif),Geneve))
// (103,((Gress,DemiTarifVisa),St-Gallen))
// (103,((Gress,DemiTarifVisa),Chur))
// (103,((Gress,DemiTarifVisa),Zurich))
```

What happened to customer 104?

# Inner Joins (`join`)

**Example continued with concrete data:**

```
trackedCustomers.collect().foreach(println)
// (101,((Ruetli,AG),Bern))
// (101,((Ruetli,AG),Thun))
// (102,((Brelaz,DemiTarif),Nyon))
// (102,((Brelaz,DemiTarif),Lausanne))
// (102,((Brelaz,DemiTarif),Geneve))
// (103,((Gress,DemiTarifVisa),St-Gallen))
// (103,((Gress,DemiTarifVisa),Chur))
// (103,((Gress,DemiTarifVisa),Zurich))
```

What happened to customer 104?

Customer 104 does *not* occur in the result, because there is no location data for this customer Remember, inner joins require keys to occur in *both* source RDDs (i.e., we must have location info).

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

Outer joins return a new RDD containing combined pairs whose **keys don't have to be present in both input RDDs**.

Outer joins are particularly useful for customizing how the resulting joined RDD deals with missing keys. With outer joins, we can decide which RDD's keys are most essential to keep–the left, or the right RDD in the join expression.

```
def leftOuterJoin[W](other: RDD[(K, W)]): RDD[(K, (V, Option[W]))]
def rightOuterJoin[W](other: RDD[(K, W)]): RDD[(K, (Option[V], W))]
```

(Notice the insertion and position of the `Option`!)

**Example:** Let's assume the CFF wants to know for which subscribers the CFF has managed to collect location information. E.g., it's possible that someone has a demi-tarif, but doesn't use the CFF app and only pays cash for tickets.

Which join do we use?

# Outer Joins (leftOuterJoin, rightOuterJoin)

**Example continued with concrete data:**

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to combine both RDDs into one:**

**The CFF wants to know for which subscribers the CFF has managed to collect location information. E.g., it's possible that someone has a demi-tarif, but doesn't use the CFF app and only pays cash for tickets.**

```
val abosWithOptionalLocations = ???
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

**Example:** Let's assume the CFF wants to know for which subscribers the CFF has managed to collect location information. E.g., it's possible that someone has a demi-tarif, but doesn't use the CFF app and only pays cash for tickets.

Which join do we use?

```
val abosWithOptionalLocations = ???
```

# Outer Joins (leftOuterJoin, rightOuterJoin)

**Example continued with concrete data:**

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to make a new RDD with these!**

```
val abosWithOptionalLocations = ???
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

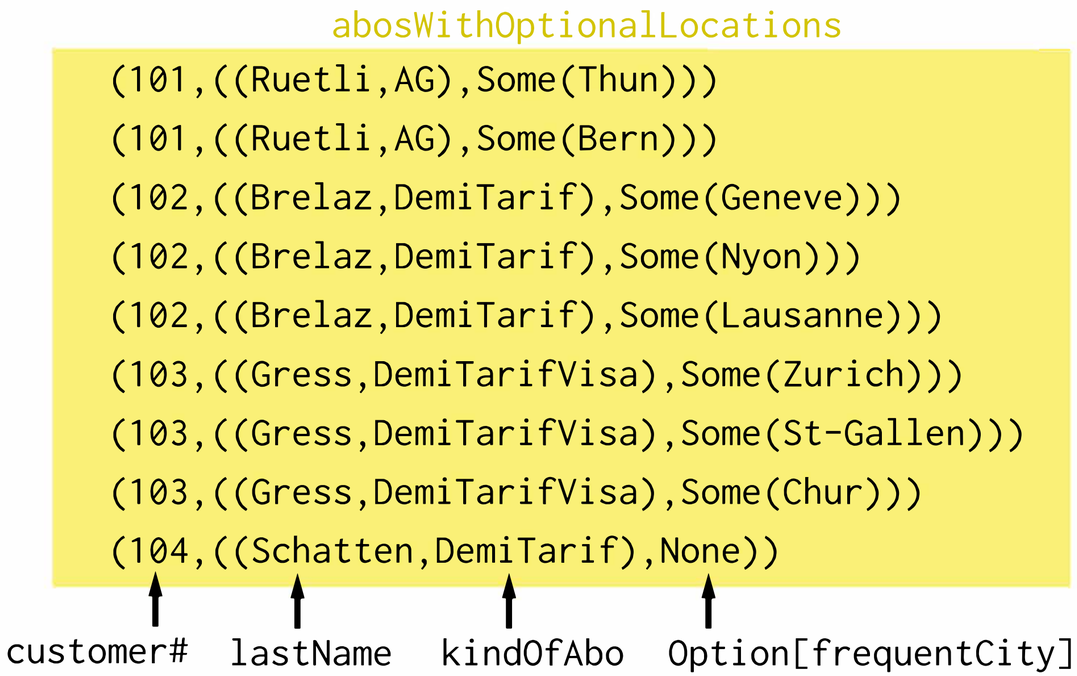
**Example:** Let's assume the CFF wants to know for which subscribers the CFF has managed to collect location information. E.g., it's possible that someone has a demi-tarif, but doesn't use the CFF app and only pays cash for tickets.

Which join do we use?

```
val abosWithOptionalLocations = abos.leftOuterJoin(locations)
// abosWithOptionalLocations: RDD[(Int, ((String, Abonnement), Option[String]))]
```

# Outer Joins (leftOuterJoin, rightOuterJoin)

**Example continued with concrete data:**




```
val abosWithOptionalLocations = abos.leftOuterJoin(locations)
// abosWithOptionalLocations: RDD[(Int, ((String, Abonnement), Option[String]))]
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

**Example continued with concrete data:**

```
val abosWithOptionalLocations = abos.leftOuterJoin(locations)
abosWithOptionalLocations.collect().foreach(println)
// (101,((Ruetli,AG),Some(Thun)))
// (101,((Ruetli,AG),Some(Bern)))
// (102,((Brelaz,DemiTarif),Some(Geneve)))
// (102,((Brelaz,DemiTarif),Some(Nyon)))
// (102,((Brelaz,DemiTarif),Some(Lausanne)))
// (103,((Gress,DemiTarifVisa),Some(Zurich)))
// (103,((Gress,DemiTarifVisa),Some(St-Gallen)))
// (103,((Gress,DemiTarifVisa),Some(Chur)))
// (104,((Schatten,DemiTarif),None))
```

Since we use a `leftOuterJoin`, keys are guaranteed to occur in the left source RDD. Therefore, in this case, we see customer 104 because that customer has a demi-tarif (the left RDD in the join).

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

We can do the converse using a `rightOuterJoin`.

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to combine both RDDs into one:**

**The CFF wants to know for which customers (smartphone app users) it has subscriptions for. E.g., it's possible that someone uses the mobile app, but has no demi-tarif.**

```
val customersWithLocationDataAndOptionalAbos = ???
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

We can do the converse using a `rightOuterJoin`.

abos

```
(101, ("Ruetli", AG)),
(102, ("Brelaz", DemiTarif)),
(103, ("Gress", DemiTarifVisa)),
(104, ("Schatten", DemiTarif))
```

locations

```
(101, "Bern"),
(101, "Thun"),
(102, "Lausanne"),
(102, "Geneve"),
(102, "Nyon"),
(103, "Zurich"),
(103, "St-Gallen"),
(103, "Chur")
```

**We want to make a new RDD with only these!**

```
val customersWithLocationDataAndOptionalAbos = ???
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

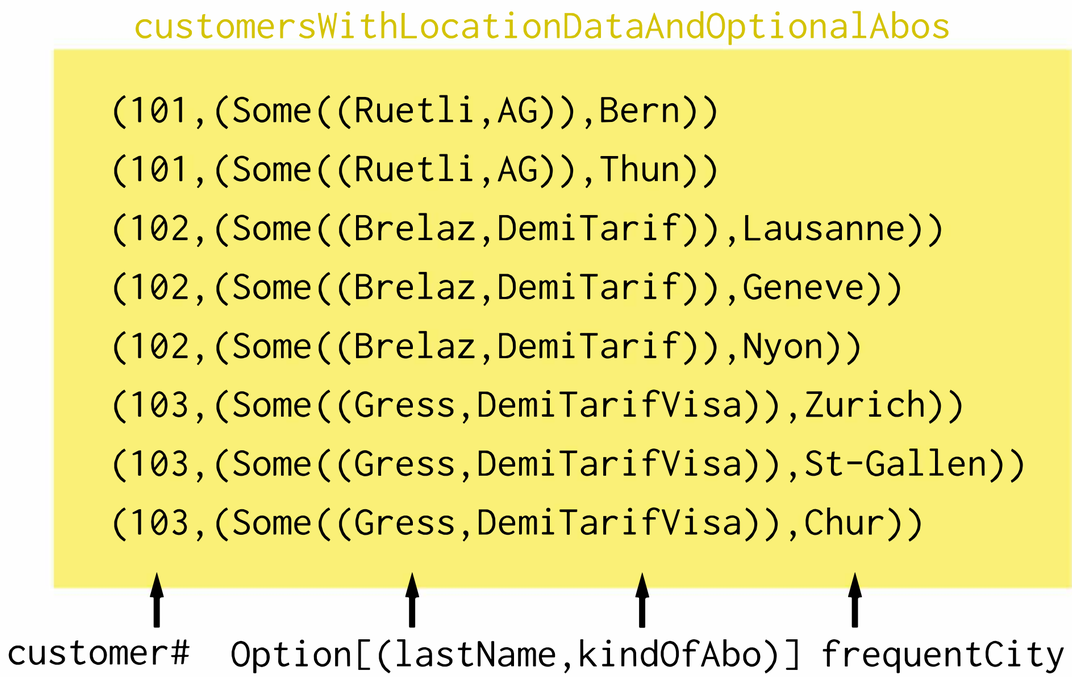
We can do the converse using a `rightOuterJoin`.

**Example:** Let's assume in this case, the CFF wants to know for which customers (smartphone app users) it has subscriptions for. E.g., it's possible that someone uses the mobile app, but has no demi-tarif.

```
val customersWithLocationDataAndOptionalAbos =
  abos.rightOuterJoin(locations)
// RDD[(Int, (Option[(String, Abonnement)], String))]
```

# Outer Joins (leftOuterJoin, rightOuterJoin)

**Example continued with concrete data:**




```
val customersWithLocationDataAndOptionalAbos =
  abos.rightOuterJoin(locations)
// RDD[(Int, (Option[(String, Abonnement)], String))]
```

# Outer Joins (`leftOuterJoin`, `rightOuterJoin`)

**Example continued with concrete data:**

```
val customersWithLocationDataAndOptionalAbos =
  abos.rightOuterJoin(locations)
// RDD[(Int, (Option[(String, Abonnement)], String))]

customersWithLocationDataAndOptionalAbos.collect().foreach(println)
// (101,(Some((Ruetli,AG)),Bern))
// (101,(Some((Ruetli,AG)),Thun))
// (102,(Some((Brelaz,DemiTarif)),Lausanne))
// (102,(Some((Brelaz,DemiTarif)),Geneve))
// (102,(Some((Brelaz,DemiTarif)),Nyon))
// (103,(Some((Gress,DemiTarifVisa)),Zurich))
// (103,(Some((Gress,DemiTarifVisa)),St-Gallen))
// (103,(Some((Gress,DemiTarifVisa)),Chur))
```

Note that, here, customer 104 disappears again because that customer doesn't have location info stored with the CFF (the right RDD in the join).

## ?? org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[366] ??

Think again what happens when you have to do a `groupBy` or a `groupByKey`. Remember our data is distributed!

## ?? org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[366] ??

Think again what happens when you have to do a `groupBy` or a `groupByKey`. Remember our data is distributed!

We typically have to move data from one node to another to be "grouped with" its key. Doing this is called "shuffling".

## ?? org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[366] ??

Think again what happens when you have to do a `groupBy` or a `groupByKey`. Remember our data is distributed!

We typically have to move data from one node to another to be "grouped with" its key. Doing this is called "shuffling".

**Shuffles Happen**

Shuffles can be an enormous hit to because it means that Spark must send data from one node to another. Why? **Latency!**

We'll talk more about these in the next lecture.