Procesamiento de datos masivos

Jesús Morán

viu | Universidad Internacional de Valencia

17.01.22

- No es streaming puro -> Microbatches
  - Streaming sobre batch
- No soporta nativamente iteraciones
- Fallos de memoria -> mejoró en las últimas versiones

- Es streaming puro
- Batch sobre streaming
- Soporte nativo para iteraciones
- Gestión de memoria
- Simplifica la configuración y optimización

Rendimiento: Flink vs Spark

- No hay bala de plata
- Los dos son rápidos

Flink

The VLDB Journal
DOI 10.1007/s00778-014-0357-y

REGULAR PAPER

The Stratosphere platform for big data analytics

Alexander Alexandrov · Rico Bergmann · Stephan Ewen · Johann-Christoph Freytag · Fabian Hueske · Arvid Heise · Odej Kao · Marcus Leich · Ulf Leser · Volker Markl · Felix Naumann · Mathias Peters · Astrid Rheinländer · Matthias J. Sax · Sebastian Schelter · Mareike Höger · Kostas Tzoumas · Daniel Warneke

Received: 10 July 2013 / Revised: 18 March 2014 / Accepted: 1 April 2014

Abstract We present Stratosphere, an open-source software stack for parallel data analysis. Stratosphere brings together a unique set of features that allow the expressive, easy, and efficient programming of analytical applications at very large scale. Stratosphere's features include "in situ" data processing, a declarative query language, treatment of user-defined functions as first-class citizens, automatic program parallelization and optimization, support programs, and a scalable and efficient execu Stratosphere covers a variety of "Big Data" us as data warehousing, information extraction an data cleansing, graph analysis, and statistical an cations. In this paper, we present the overall syst ture design decisions, introduce Stratosphere th ple queries, and then dive into the internal wo system's components that relate to extensibili ming model, optimization, and query execution mentally compare Stratosphere against popular

Stratosphere is funded by the German Research Foundation (DFG) under grant FOR 1306.

Apache Flink™: Stream and Batch Processing in a Single Engine

Paris Carbone[†]
Asterios Katsifodimos[*]

[†]KTH & SICS Sweden
parisc,haridi@kth.se

Stephan Ewen[‡]
Volker Markl[*]

[‡]data Artisans
first@data-artisans.com

Seif Haridi[†]
Kostas Tzoumas[‡]

[*]TU Berlin & DFKI
first.last@tu-berlin.de

Abstract

Apache Flink[1] is an open-source system for processing streaming and batch data. Flink is built on the philosophy that many classes of data processing applications, including real-time analytics, continuous data pipelines, historic data processing (batch), and iterative algorithms (machine learning, graph analysis) can be expressed and executed as pipelined fault-tolerant dataflows. In this paper, we present Flink's architecture and expand on how a (seemingly diverse) set of use cases can be unified under a single execution model.

1 Introduction

Data-stream processing (e.g., as exemplified by complex event processing systems) and static (batch) data processing (e.g., as exemplified by MPP databases and Hadoop) were traditionally considered as two very different

- Tiene un optimizador
  - Las transformaciones no se ejecutan secuencialmente
- Evaluación perezosa
- Entorno de ejecución
- Registros de un dataset: sin clave, con clave, con clave compuesta
- Sistema de ejecución Streaming: DataStream y ~~DataSet~~

- Archivos:
  - readFile(inputFormat, path)
  - readTextFile(path): Utiliza TextInputFormat (cada registro es una línea del archivo)
  - readCsvFile(path): lee un CSV y nos lo guarda como una tupla
- Desde colección:
  - fromCollection(Collection)
  - fromElements(T ...)
  - generateSequence(from, to): desde un número a otro
- Otros:
  - createInput(inputFormat): cualquier fuente de datos con inputFormat Ejemplo: mongoInputFormat

- Conectores streaming:
  - Kafka
  - RabbitMQ
  - Google Pub/Sub
  - Ni-Fi
  - Kinesis
  - Cassandra
  - Redis
  - Elasticsearch
  - Sistema de archivos de Hadoop
  - Twitter

- Arhivo:
  - writeAsText(): utiliza TextOutputFormat (cada registro se guarda en una línea)
  - writeAsCsv(...): utiliza CsvOutputFormat (guarda como csv indicando delimitador, etc.)
  - write(): utiliza FileOutputFormat y nos sirve para crear nuestros otros métodos customizados de almacenamiento en archivo
- Otro:
  - print(),printToErr(), print(String msg) y printToErr(String msg): muestra los datos por consola
  - output(): exporta a cualquier fuente de datos con OutputFormat, por ejemplo MongoOutputFormat.

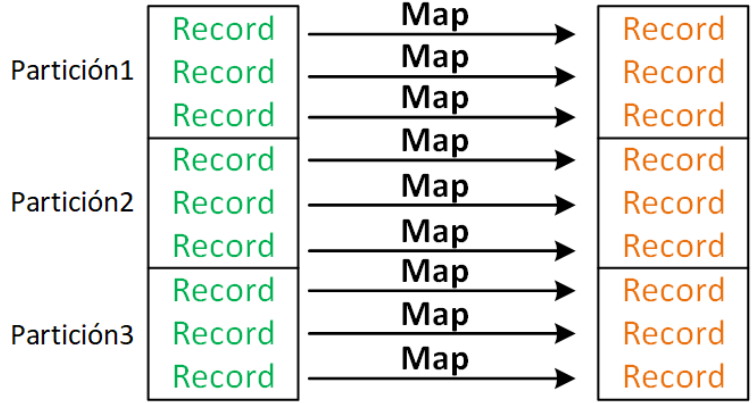
- map
  - Por cada registro siempre emite un registro

```
DataSource<String> data = env.fromCollection(Arrays.asList("uno", "dos", "tres",
                                                           "cuatro", "cinco", "seis",
                                                           "siete", "ocho", "nueve"));
DataSet<Integer> lettersOfRegisters = data.map( value -> value.length() );
```

```
-------------------------
Data before map:
Partition 0:  uno, dos, tres
Partition 1:  cuatro, cinco, seis
Partition 2:  siete, ocho, nueve
-------------------------
```

```
-------------------------
Data after map:
Partition 0:  3, 3, 4
Partition 1:  6, 5, 4
Partition 2:  5, 4, 5
-------------------------
```

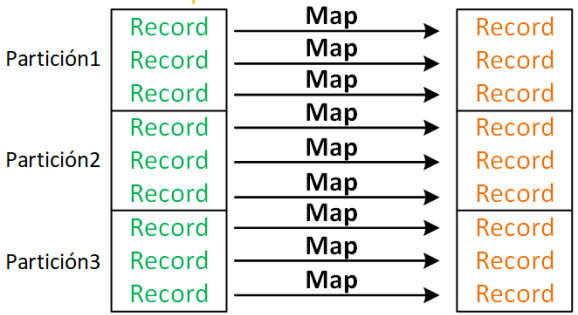
```
Task 0: map(uno) -> 3
        map(dos) -> 3
        map(tres) -> 4
Task 1: map(cuatro) -> 6
        map(cinco) -> 5
        map(seis) -> 4
Task 2: map(siete) -> 5
        map(ocho) -> 4
        map(nueve) -> 5
```



- map
  - Por cada registro siempre emite un registro

```
DataSource<String> data = env.fromCollection(Arrays.asList("uno", "dos", "tres",
                                                           "cuatro", "cinco", "seis",
                                                           "siete", "ocho", "nueve"));
DataSet<Integer> lettersOfRegisters = data.map(new sumLetters());
```

```
public static final class sumLetters implements MapFunction<String, Integer> {
    private static final long serialVersionUID = -6900227250953473523L;

    @Override
    public Integer map(String value) throws Exception {
        return value.length();
    }
}
```




```
------------------------
Data before map:
Partition 0:  uno, dos, tres
Partition 1:  cuatro, cinco, seis
Partition 2:  siete, ocho, nueve
------------------------
```

```
Task 0: map(uno) -> 3
        map(dos) -> 3
        map(tres) -> 4
Task 1: map(cuatro) -> 6
        map(cinco) -> 5
        map(seis) -> 4
Task 2: map(siete) -> 5
        map(ocho) -> 4
        map(nueve) -> 5
```

```
------------------------
Data after map:
Partition 0:  3, 3, 4
Partition 1:  6, 5, 4
Partition 2:  5, 4, 5
------------------------
```

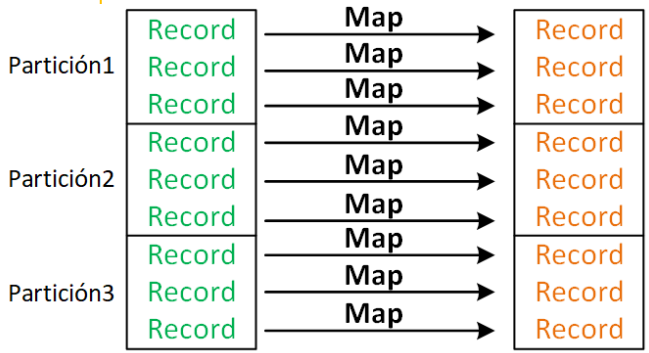
- map
  - Por cada registro siempre emite un registro

```
DataSource<String> data = env.fromCollection(Arrays.asList("uno", "dos", "tres",
                                                           "cuatro", "cinco", "seis",
                                                           "siete", "ocho", "nueve"));
DataSet<Integer> lettersOfRegisters = data.map(new MapFunction<String, Integer>() {
    private static final long serialVersionUID = 1L;

    @Override
    public Integer map(String value) throws Exception {
        return value.length();

    }
});
```

```
------------------------
Data before map:
Partition 0:  uno, dos, tres
Partition 1:  cuatro, cinco, seis
Partition 2:  siete, ocho, nueve
------------------------
```

```
Task 0: map(uno) -> 3
        map(dos) -> 3
        map(tres) -> 4
Task 1: map(cuatro) -> 6
        map(cinco) -> 5
        map(seis) -> 4
Task 2: map(siete) -> 5
        map(ocho) -> 4
        map(nueve) -> 5
```

```
------------------------
Data after map:
Partition 0:  3, 3, 4
Partition 1:  6, 5, 4
Partition 2:  5, 4, 5
------------------------
```

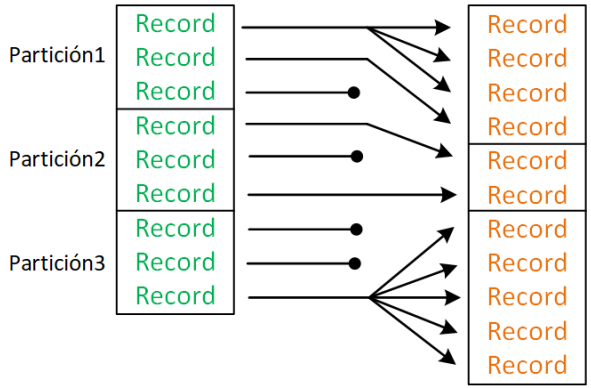


flatMap

- Por cada registro puede emitir 0, 1 o varios registros

```
DataSource<String> data = env.fromCollection(Arrays.asList(
                                "son tres palabras", "cuatro", "-1",
                                "uno", "-1", "otra",
                                "-1", "-1", "tercer particion con cinco palabras"));
DataSet<String> newData = data.flatMap(new FlatMapFunction<String, String>() {
    private static final long serialVersionUID = -6900227250953473523L;

    @Override
    public void flatMap(String value, Collector<String> out) throws Exception {
        if (! value.equals("-1"))
            for (String word : value.split(" "))
                out.collect(word);
    }
});
```

```
------------------------
Data before flatMap:
Partition 0:  son tres palabras, cuatro, -1
Partition 1:  uno, -1, otra
Partition 2:  -1, -1, tercer particion con cinco palabras
------------------------
```

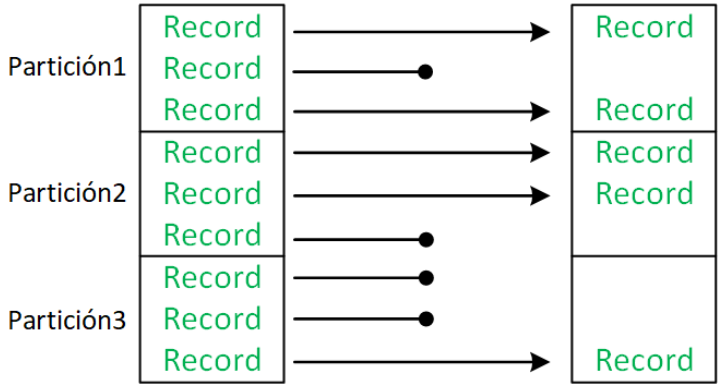



```
Task 0: flatMap(son tres palabras) -> son, tres, palabras
        flatMap(cuatro) -> cuatro
        flatMap(-1) ->
Task 1: flatMap(uno) -> uno
        flatMap(-1) ->
        flatMap(otra) -> otra
Task 2: flatMap(-1) ->
        flatMap(-1) ->
        flatMap(tercer particion con cinco palabras) -> tercer, particion, con, cinco, palabras
```

```
------------------------
Data after flatMap:
Partition 0:  son, tres, palabras, cuatro
Partition 1:  uno, otra
Partition 2:  tercer, particion, con, cinco, palabras
------------------------
```

- filter
  - Por cada registro o lo emite o no

```
DataSource<Integer> data = env.fromCollection(Arrays.asList(1, -1, 3,
                                                            4, 5, -1,
                                                            -1, -1, 9));
DataSet<Integer> filteredData = data.filter(new FilterFunction<Integer>() {
    private static final long serialVersionUID = 4934848865379455313L;

    @Override
    public boolean filter(Integer value) throws Exception {
        if (value != -1)
            return true;
        else
            return false;
    }
});
```




```
-------------------------
Data before filter:
Partition 0:  1, -1, 3
Partition 1:  4, 5, -1
Partition 2:  -1, -1, 9
-------------------------
```

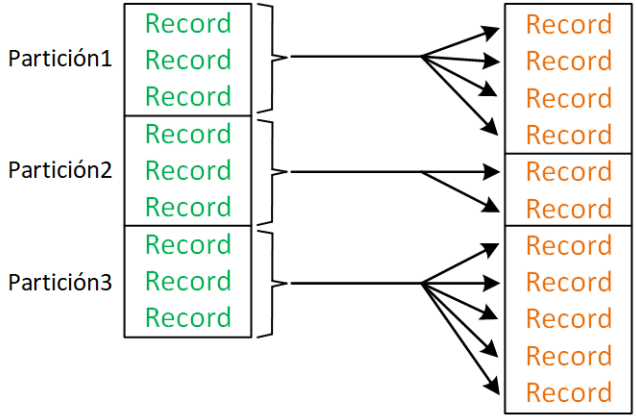
```
Task 0: filter(1) -> true
        filter(-1) -> false
        filter(3) -> true
Task 1: filter(4) -> true
        filter(5) -> true
        filter(-1) -> false
Task 2: filter(-1) -> false
        filter(-1) -> false
        filter(9) -> true
```

```
--------------------
Data after filter:
Partition 0:  1, 3
Partition 1:  4, 5
Partition 2:  9
--------------------
```

- mapPartition
  - Por cada partición puede emitir 0, 1 o n registros

```
DataSource<String> data = env.fromCollection(Arrays.asList(
                        "las cuatro palabras emitidas", "-1", "esto no se emite",
                        "se", "emite", "-1",
                        "lo último que se emite", "-1", "esto tampoco se emite"));
DataSet<String> newData = data.mapPartition(
                                    new MapPartitionFunction<String, String> () {
    private static final long serialVersionUID = -1760362441299143862L;

    @Override
    public void mapPartition(Iterable<String> values, Collector<String> out)
                                                                    throws Exception {
        for (String value : values) {
            if (value.equals("-1"))
                break;
            else
                out.collect(value);
        }
    }
});
```

```
-------------------------
Data before mapPartition:
Partition 0:  las cuatro palabras emitidas, -1, esto no se emite
Partition 1:  se, emite, -1
Partition 2:  lo último que se emite, -1, esto tampoco se emite
-------------------------
```

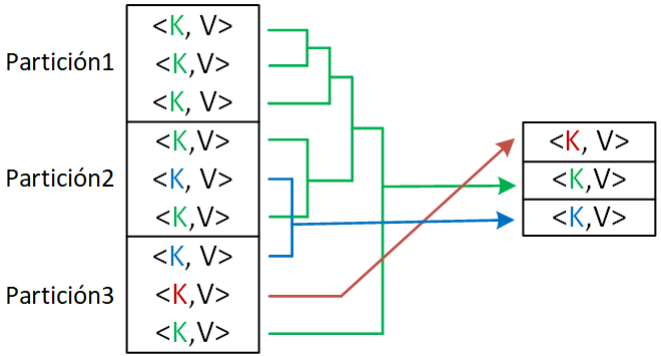



```
-------------------------
Data after filter:
Partition 0:  las cuatro palabras emitidas
Partition 1:  se, emite
Partition 2:  lo último que se emite
-------------------------
```

```
Task 0: mapPartition(las cuatro palabras emitidas, -1, esto no se emite) -> las cuatro palabras emitidas
Task 1: mapPartition(se, emite, -1) -> se, emite
Task 2: mapPartition(lo último que se emite, -1, esto tampoco se emite) -> lo último que se emite
```

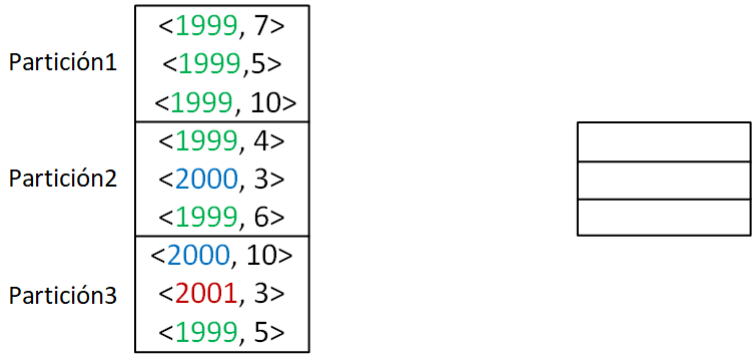
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



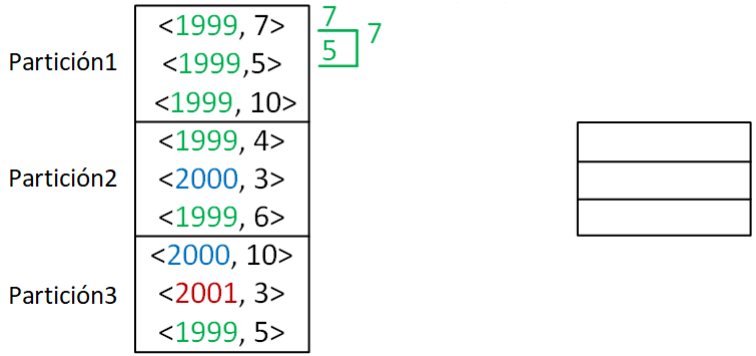
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



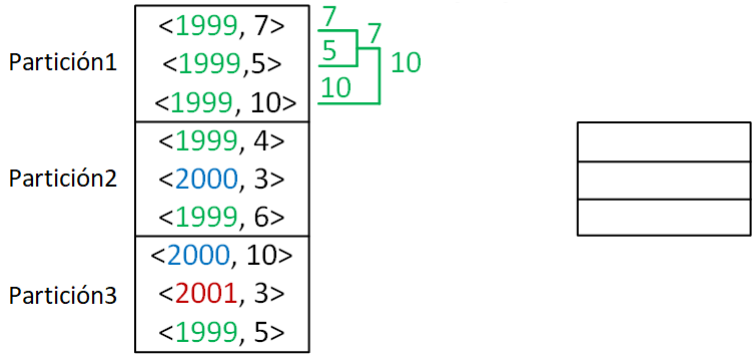
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



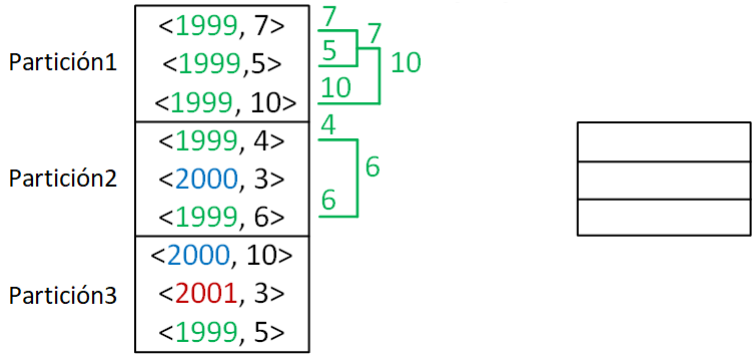
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



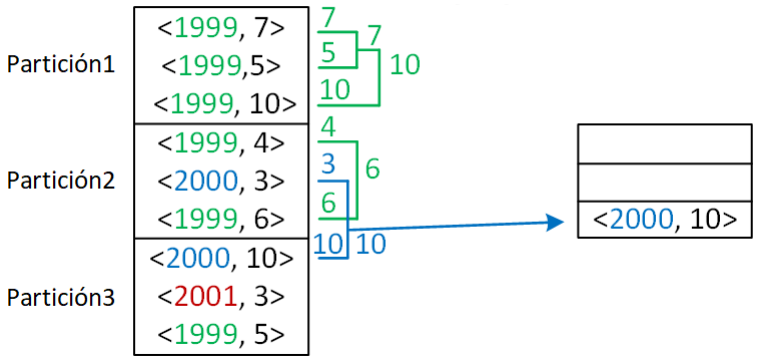
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



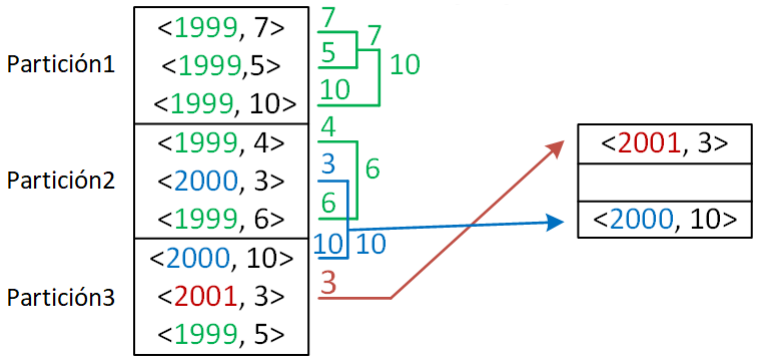
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



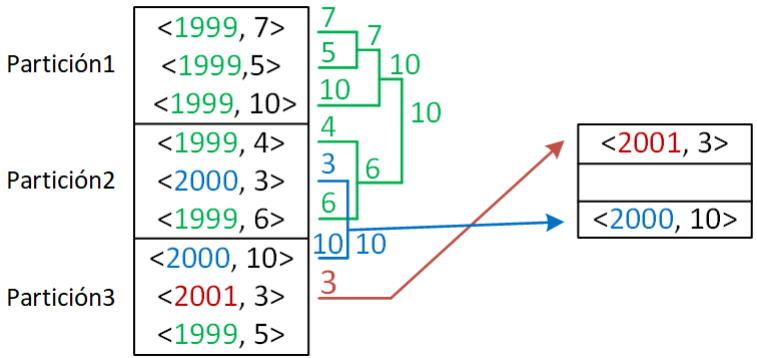
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



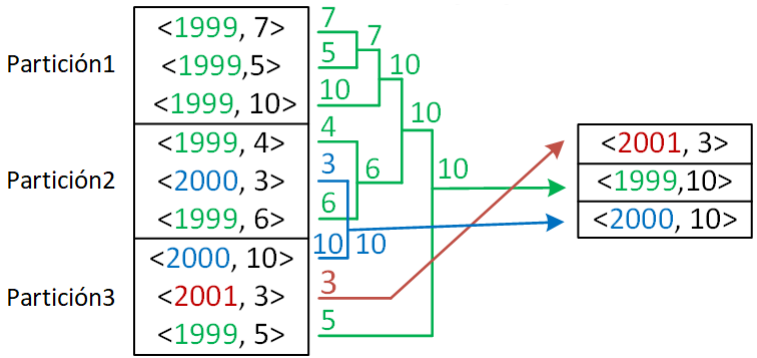
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



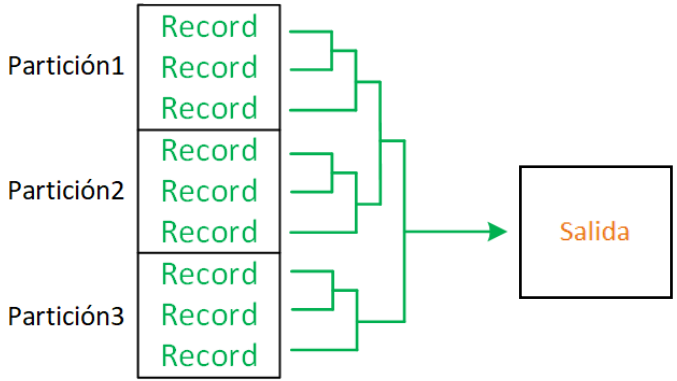
- reduce
  - Emite 1 registro (por DataSet o **por clave**)

```
DataSet<Tuple2<Integer, Integer>> data = env.fromCollection(Arrays.asList(
                                new Tuple2<Integer, Integer>(1999, 7),
                                new Tuple2<Integer, Integer>(1999, 5),
                                new Tuple2<Integer, Integer>(1999, 10),

                                new Tuple2<Integer, Integer>(1999, 4),
                                new Tuple2<Integer, Integer>(2000, 3),
                                new Tuple2<Integer, Integer>(2000, 7),

                                new Tuple2<Integer, Integer>(2000, 10),
                                new Tuple2<Integer, Integer>(2001, 3),
                                new Tuple2<Integer, Integer>(1999, 5)));

DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduce(
                            new ReduceFunction<Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -7573583914328613340L;

    @Override
    public Tuple2<Integer, Integer> reduce(Tuple2<Integer, Integer> accum,
                            Tuple2<Integer, Integer> value) throws Exception {
        if (value.f1 > accum.f1) //value.temperature > accum.temperature
            return value;
        else
            return accum;
    }
});
```



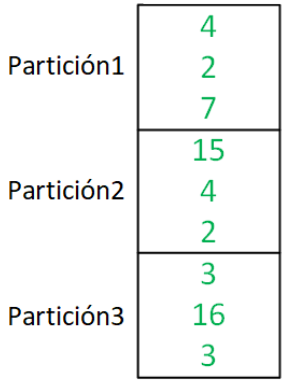
- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```



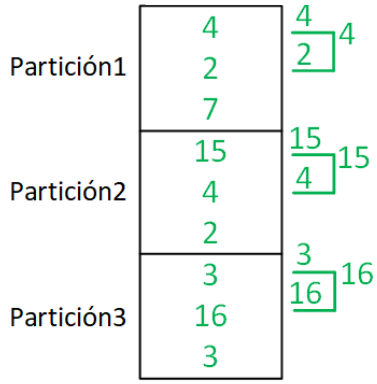
- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```



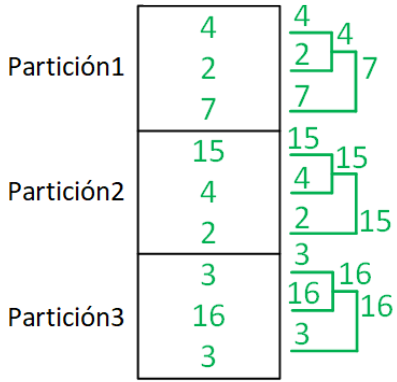
- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```



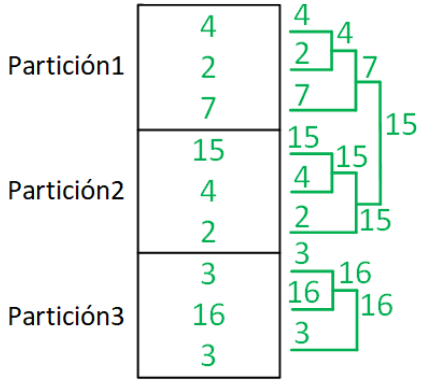
- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```



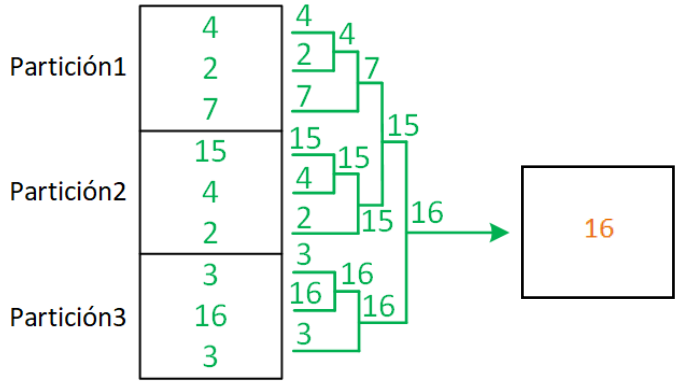
- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```



- reduce
  - Emite 1 registro (**por DataSet** o por clave)

```
DataSet<Integer> data = env.fromCollection(Arrays.asList(4,2,7,
                                                         15, 4, 2,
                                                         3, 16, 3));

DataSet<Integer> newData = data.reduce(new ReduceFunction<Integer>() {
    private static final long serialVersionUID = -7719634106939607881L;

    @Override
    public Integer reduce(Integer accum, Integer value) throws Exception {
        if (value > accum)
            return value;
        else
            return accum;
    }
});
```

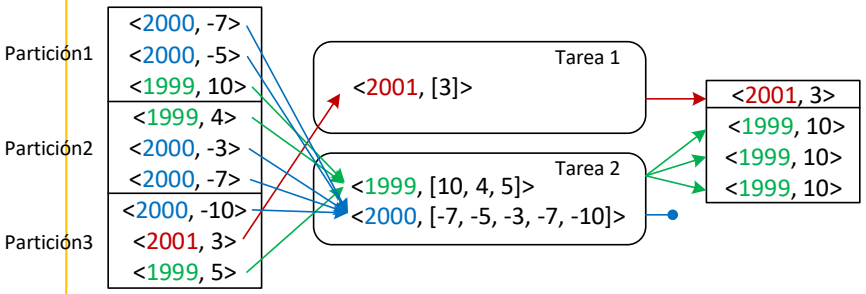


- reduceGroup
  - Emite 0, 1 o N registros (por DataSet o **por clave**)
  - Puede ejecutarse con o **sin Combine**

```
DataSet<Tuple2<Integer, Integer>> newData = data.groupBy(0).reduceGroup(
        new GroupReduceFunction<Tuple2<Integer, Integer>,
                                Tuple2<Integer, Integer>>() {
    private static final long serialVersionUID = -4668003431890746821;

    @Override
    public void reduce(Iterable<Tuple2<Integer, Integer>> values,
                       Collector<Tuple2<Integer, Integer>> out) throws Exception {
        int maxTemp = Integer.MIN_VALUE;
        int year = -1;
        int count = 0;
        for (Tuple2<Integer, Integer> value : values) {
            year = value.f0;
            if (value.f1 > maxTemp) //temperature > maxTemp
                maxTemp = value.f1;
            count++;
        }

        if (maxTemp > 0) {
            for (int i = 0; i < count; i++)
                out.collect(new Tuple2<Integer, Integer>(year, maxTemp));
        }
    }
});
```

Si la temperatura máxima de un año < 0 -> no emite

Si la temperatura máxima de un año > 0 -> emite <año, temperatura máxima> tantas veces como pares <clave, valor> tenga

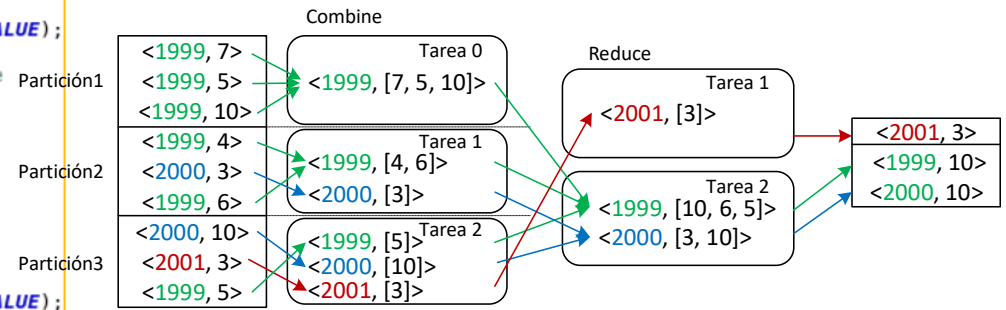


Tareas

- reduceGroup
  - Emite 0, 1 o N registros (por DataSet o **por clave**)
  - Puede ejecutarse **con** o sin **Combine**

```
public class miReduceCombine implements
    GroupReduceFunction<Tuple2<Integer, Integer>, Tuple2<Integer, Integer>>,
    GroupCombineFunction<Tuple2<Integer, Integer>, Tuple2<Integer, Integer>>{

    private static final long serialVersionUID = 7760143877649257585L;
    @Override
    public void reduce(Iterable<Tuple2<Integer, Integer>> values,
                       Collector<Tuple2<Integer, Integer>> out) {
        Tuple2<Integer, Integer> yearMaxTemp =
            new Tuple2<Integer, Integer>(Integer.MIN_VALUE, Integer.MIN_VALUE);
        for (Tuple2<Integer, Integer> value : values)
            if (value.f1 > yearMaxTemp.f1) //temperature > max temperature
                yearMaxTemp = value;

        out.collect(yearMaxTemp);
    }
    @Override
    public void combine(Iterable<Tuple2<Integer, Integer>> values,
                       Collector<Tuple2<Integer, Integer>> out)
            throws Exception {
        Tuple2<Integer, Integer> yearMaxTemp =
            new Tuple2<Integer, Integer>(Integer.MIN_VALUE, Integer.MIN_VALUE);
        for (Tuple2<Integer, Integer> value : values)
            if (value.f1 > yearMaxTemp.f1) //temperature > max temperature
                yearMaxTemp = value;

        out.collect(yearMaxTemp);
    }
}
```

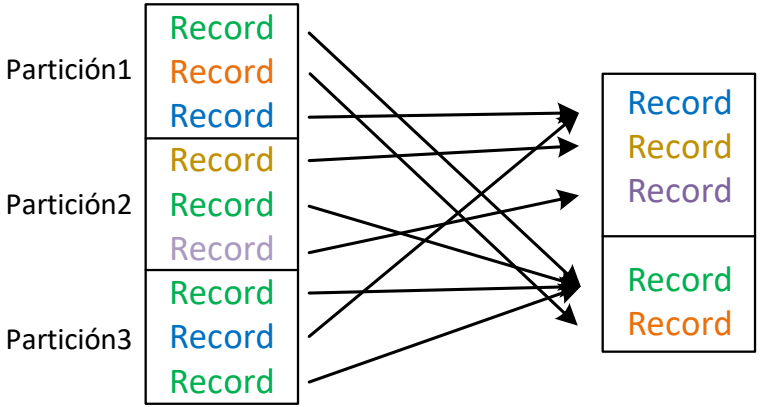
Combine pude no ejecutarse



- Distinct
  - Elimina los duplicados

```
DataSource<Integer> data = env.fromCollection(Arrays.asList(1, 2, 3,
                                                            4, 1, 6,
                                                            1, 3, 1));
DataSet<Integer> lettersOfRegisters = data.distinct();
```

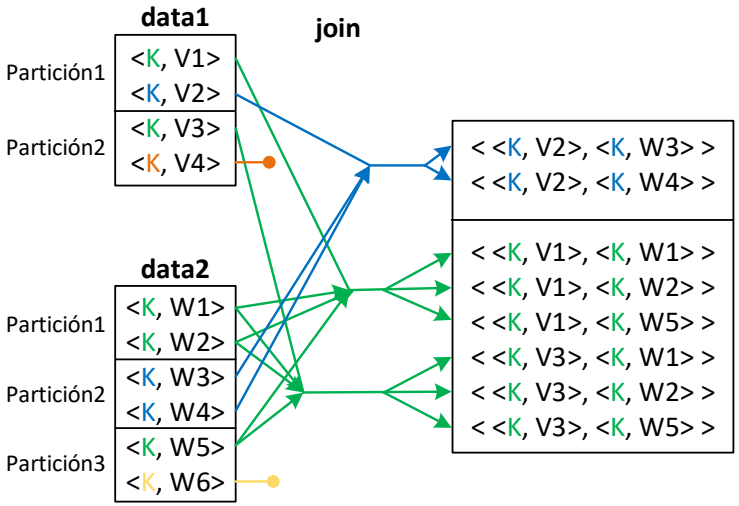
```
-------------------------
Data before distinct:
Partition 0:  1, 2, 3
Partition 1:  4, 1, 6
Partition 2:  1, 3, 1
-------------------------
Data after distinct:
Partition 0:  3, 4, 6
Partition 1:  1, 2
-------------------------
```



- join
  - Ejecuta un inner join
  - Oros joins soportados por Flink: leftOuterJoin y rightOuterJoin

```
DefaultJoin<Tuple2<String, String>, Tuple2<String, String>> newData =
                                        data1.join(data2).where(0).equalTo(0);
```

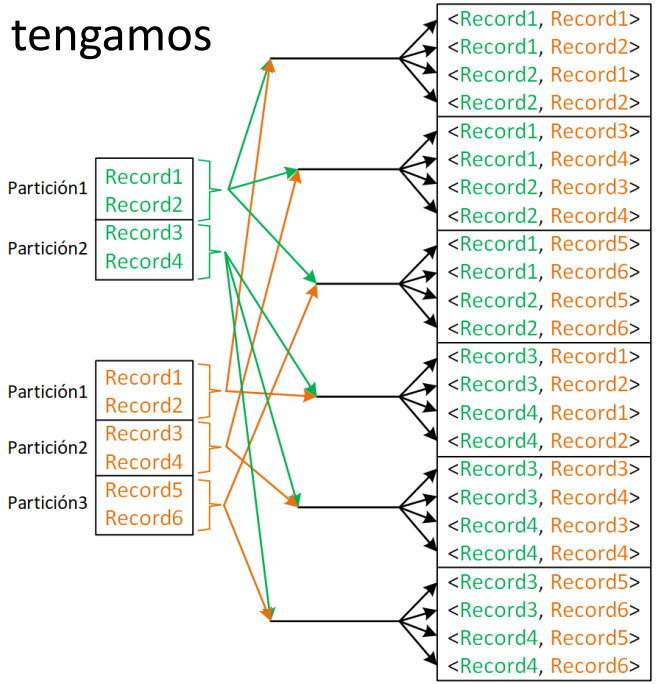
```
-------------------------
Data 1 before operation:        [(compra1,tarjeta1), (compra2,tarjeta1),
(compra1,tarjeta2), (compra3,tarjeta1)]
Data 2 before operation:        [(compra1,Ordenador), (compra1,altavoces),
(compra2,coche), (compra2,ruedas), (compra1,teclado), (compra4,otro)]
Data after operation: [
        ((compra1,tarjeta1),(compra1,Ordenador))
        ((compra1,tarjeta2),(compra1,Ordenador))
        ((compra1,tarjeta1),(compra1,altavoces))
        ((compra1,tarjeta2),(compra1,altavoces))
        ((compra2,tarjeta1),(compra2,coche))
        ((compra2,tarjeta1),(compra2,ruedas))
        ((compra1,tarjeta1),(compra1,teclado))
        ((compra1,tarjeta2),(compra1,teclado))
]
```



- Cross (producto cartesiano
  - Crea todos los pares de todos los datos que tengamos

```
DataSet<String> data1 = env.fromCollection(Arrays.asList(
                                            "tarjeta1", "tarjeta2",
                                            "tarjeta3", "tarjeta4"));

DataSet<String> data2 = env.fromCollection(Arrays.asList(
                                            "Ordenador", "altavoces",
                                            "coche", "ruedas",
                                            "teclado", "otro"));

DataSet<Tuple2<String, String>> newData = data1.cross(data2);
```

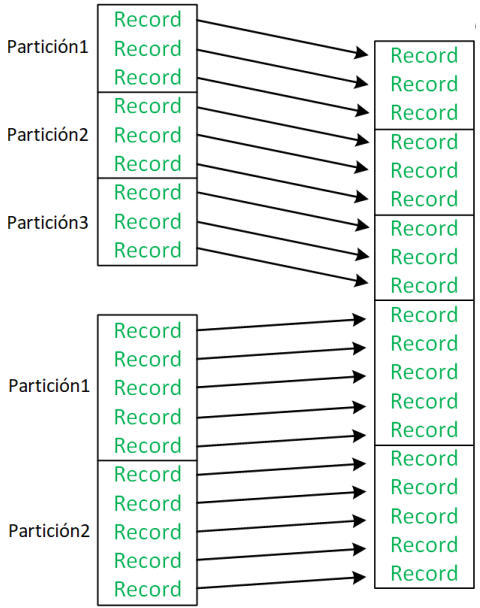
```
Data 1 before operation:        [tarjeta1, tarjeta2, tarjeta3, tarjeta4]
Data 2 before operation:        [Ordenador, altavoces, coche, ruedas,
teclado, otro]
Data after operation:   [(tarjeta1,Ordenador), (tarjeta2,Ordenador),
(tarjeta3,Ordenador), (tarjeta4,Ordenador), (tarjeta1,altavoces),
(tarjeta2,altavoces), (tarjeta3,altavoces), (tarjeta4,altavoces),
(tarjeta1,coche), (tarjeta2,coche), (tarjeta3,coche), (tarjeta4,coche),
(tarjeta1,ruedas), (tarjeta2,ruedas), (tarjeta3,ruedas), (tarjeta4,ruedas),
(tarjeta1,teclado), (tarjeta2,teclado), (tarjeta3,teclado),
(tarjeta4,teclado), (tarjeta1,otro), (tarjeta2,otro), (tarjeta3,otro),
(tarjeta4,otro)]
```



- union
  - Une dos DataSets

```
DataSet<String> data1 = env.fromCollection(Arrays.asList(
                                        "1", "2", "3",
                                        "4", "5", "6",
                                        "7", "8", "9"));

DataSet<String> data2 = env.fromCollection(Arrays.asList(
                                "10", "11", "12", "13", "14",
                                "15", "16", "17", "18", "19"));

DataSet<String> newData = data1.union(data2);
```

```
Data 1 before operation:        [1, 2, 3, 4, 5, 6, 7, 8, 9]
Data 2 before operation:        [10, 11, 12, 13, 14, 15, 16, 17,
18, 19]
Data after operation:   [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19]
```



- Otras:
  - Rebalance: re-particionar los datos
  - Particionado personalizado
  - First: obtener n elementos
  - Collect: obtener una lista con el DataSet
  - RichFunction: funciones que proporcionan información de contexto
  - …

- DataSet: cada vez en más desuso
- Table: para datos estructurados
- DataStream: para datos (des)estructurados

- PyFlink: en implementación

Gracias

viu | Universidad Internacional de Valencia