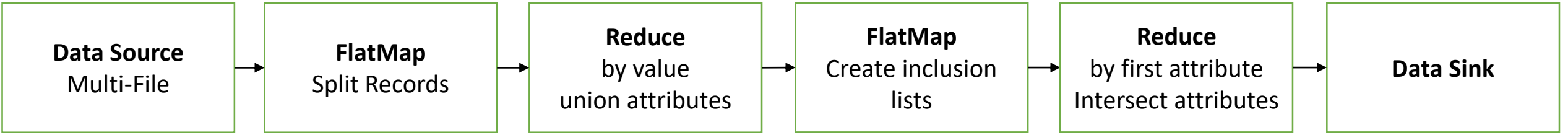

From Paper: Scaling Out the Discovery of Inclusion Dependencies

Load data → Flatmap headers to splitted rows → Reduce union → GroupByKey → Collect set attributes → Explode tupels → Create inclusion list → Remove self from inclusion list → GroupBy attribute → Collect_set target attributes → Map set of sets of attributes to intersections → Filter out empty items → Print

```
val data = inputs.map(
  spark.read
      .option("inferSchema", "false")
      .option("header", "true")
      .option("delimiter", ";")
      .csv(_)
) // as data frames
```

```
val tupels = data.map(df => {
  val headers = df.columns.toSeq
  df.flatMap(row => row.toSeq.asInstanceOf[Seq[String]]
     .zip(headers).map( x=>(x._1,x._2) ) )
})
```

```
.reduce(_.union(_))
//.repartition(32)

val cells = tupels
    .groupBy( cols = $"_1")
    .agg(collect_set($"_2").alias( alias = "tupels"))
```

```
val inclList =cells
  .select(explode($"tupels"), $"tupels")
  .as[(String,Seq[String])]
  .map( x => (x._1,x._2.filter(_!=x._1)))
  .groupBy( cols = $"_1")
  .agg(collect_set($"_2"))
  .as[(String,Seq[Seq[String]])]
  .map(x => (x._1,x._2.reduce(_.intersect(_))))
```

```
.filter(x=> x._2.length > 0)
.foreach(r => println(r._1+
  " < "+r._2.reduce(_+", "+_)) )
```