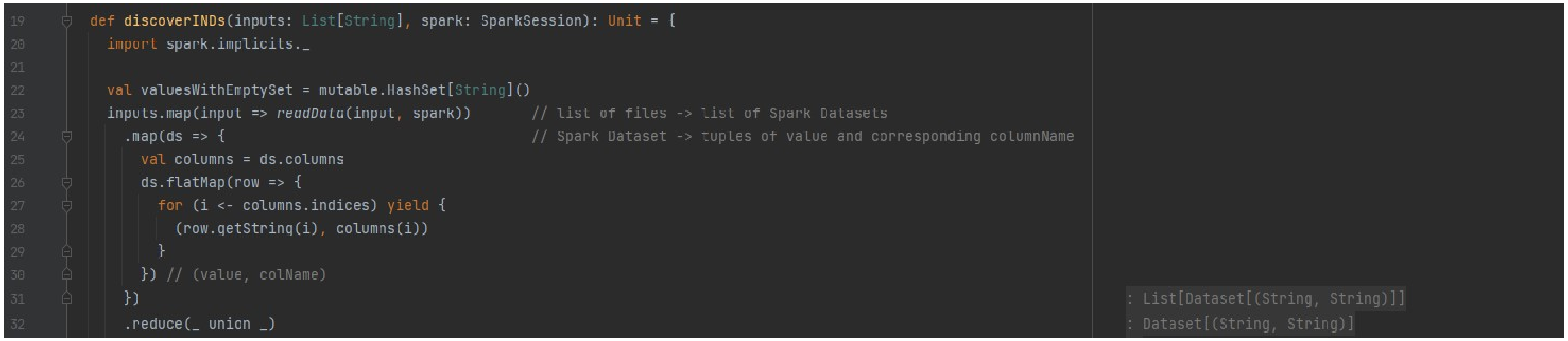
Distributed Data Management

spark-homework, Team null

Marcian Seeger & Justus Gillmann

Code 1

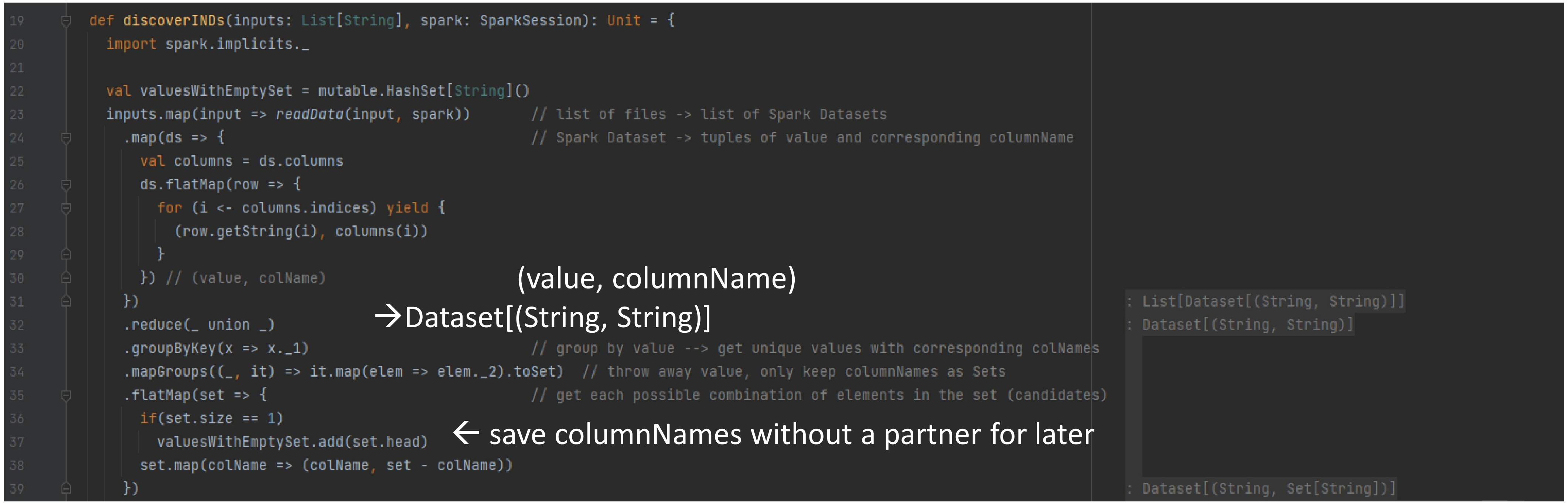


```
def discoverINDs(inputs: List[String], spark: SparkSession): Unit = {
  import spark.implicits._

  val valuesWithEmptySet = mutable.HashSet[String]()
  inputs.map(input => readData(input, spark))       // list of files -> list of Spark Datasets
    .map(ds => {                                    // Spark Dataset -> tuples of value and corresponding columnName
      val columns = ds.columns
      ds.flatMap(row => {
        for (i <- columns.indices) yield {
          (row.getString(i), columns(i))
        }
      }) // (value, colName)
    })                                                                                     : List[Dataset[(String, String)]]
    .reduce(_ union _)                                                                     : Dataset[(String, String)]
```

- Filepaths → Spark Datasets
- Dataset of rows → Dataset of (value, columnName)-pairs (for each row and value)

Code 2



```scala
def discoverINDs(inputs: List[String], spark: SparkSession): Unit = {
  import spark.implicits._

  val valuesWithEmptySet = mutable.HashSet[String]()
  inputs.map(input => readData(input, spark))       // list of files -> list of Spark Datasets
    .map(ds => {                                    // Spark Dataset -> tuples of value and corresponding columnName
      val columns = ds.columns
      ds.flatMap(row => {
        for (i <- columns.indices) yield {
          (row.getString(i), columns(i))
        }
      }) // (value, colName)
    })
    .reduce(_ union _)
    .groupByKey(x => x._1)                          // group by value --> get unique values with corresponding colNames
    .mapGroups((_, it) => it.map(elem => elem._2).toSet)  // throw away value, only keep columnNames as Sets
    .flatMap(set => {                               // get each possible combination of elements in the set (candidates)
      if(set.size == 1)
        valuesWithEmptySet.add(set.head)
      set.map(colName => (colName, set - colName))
    })
```

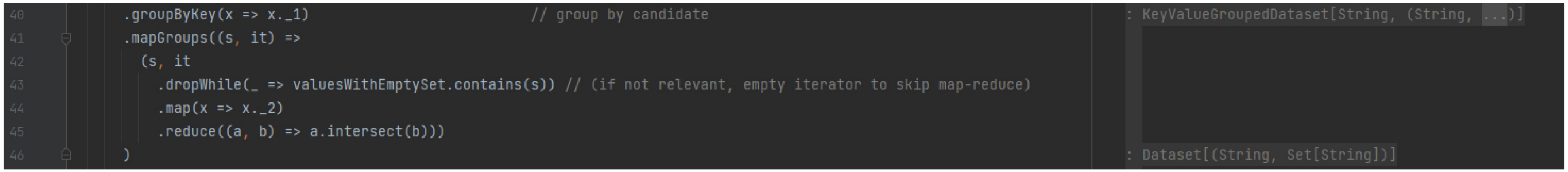
(value, columnName)
→Dataset[(String, String)]
← save columnNames without a partner for later
: List[Dataset[(String, String)]]
: Dataset[(String, String)]
: Dataset[(String, Set[String])]

- (value, columnName)-Tuples → groups of (value, all existing columnNames for this value)-Tuples
- → Sets of (all existing columnNames for „this“ value)
  („this value“ is thrown away)
- build all possible combinations of elements in the set (= IND candidates)

Code 3

(columnName, [more columnNames])

→ Dataset[(String, Set[String])]



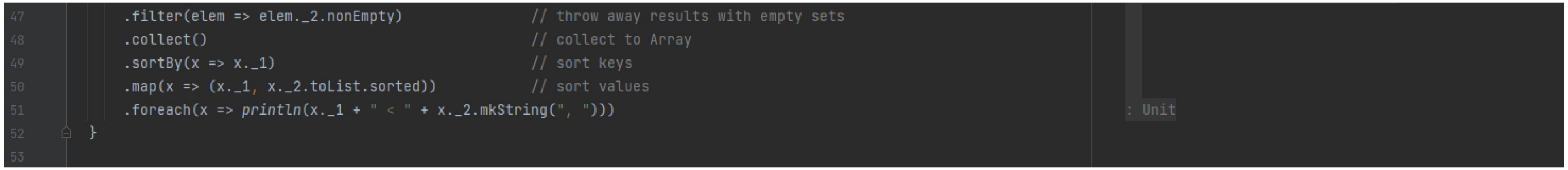
```scala
.groupByKey(x => x._1)                          // group by candidate                                        : KeyValueGroupedDataset[String, (String, ...)]
.mapGroups((s, it) =>
  (s, it
    .dropWhile(_ => valuesWithEmptySet.contains(s)) // (if not relevant, empty iterator to skip map-reduce)
    .map(x => x._2)
    .reduce((a, b) => a.intersect(b)))
)                                                                                                             : Dataset[(String, Set[String])]
```

- group by key (columnName / IND candidate) → get all Sets for each columnName
- clear iterator if the grouped by key was partnerless before (skip map-reduce)

→ only keep columnNames that are included in every Set for this key (columnName / …)

Code 4

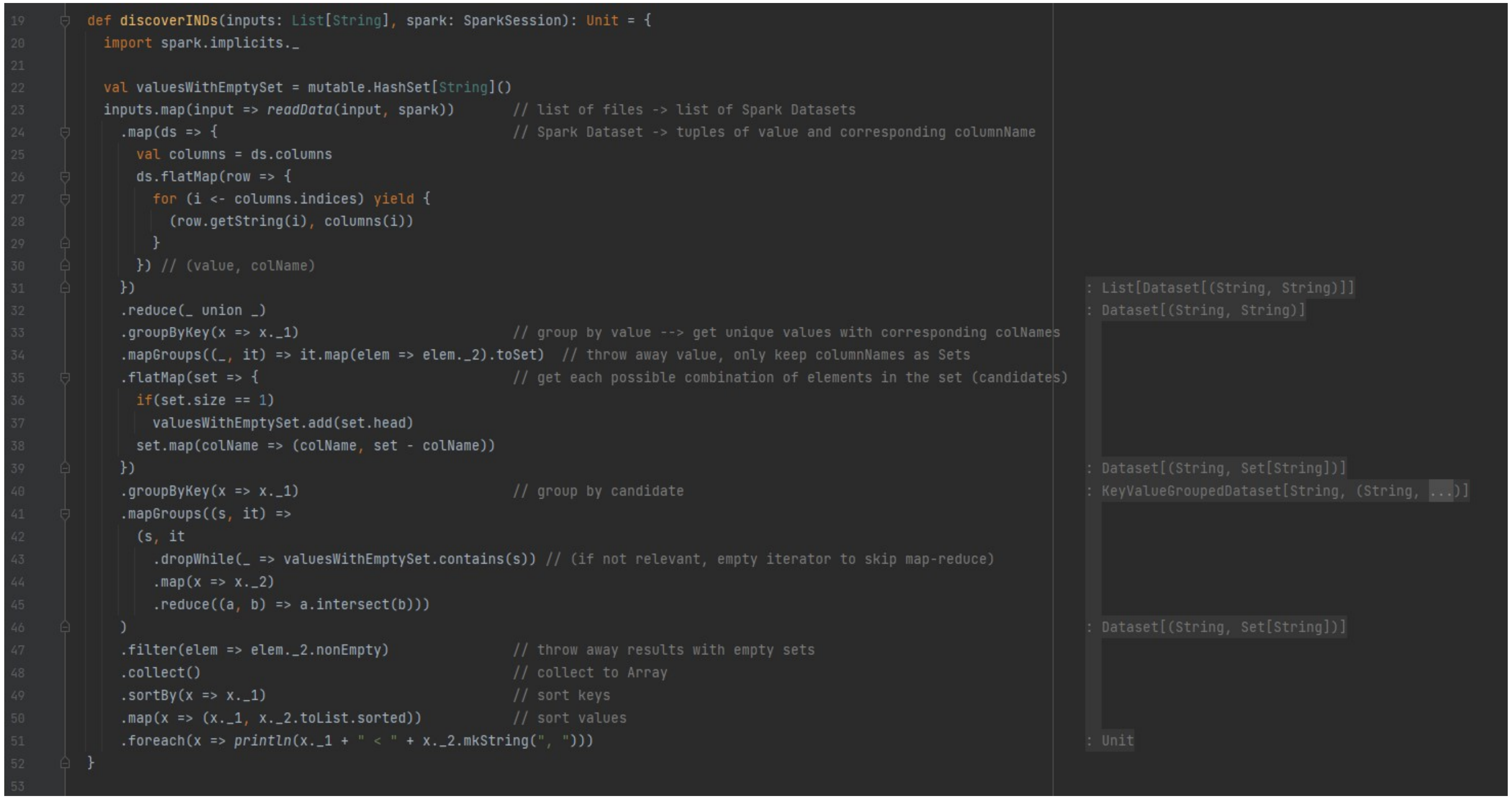
(columnName, [more columnNames])
→ Dataset[(String, Set[String])]



```
        .filter(elem => elem._2.nonEmpty)                 // throw away results with empty sets
        .collect()                                        // collect to Array
        .sortBy(x => x._1)                                // sort keys
        .map(x => (x._1, x._2.toList.sorted))             // sort values
        .foreach(x => println(x._1 + " < " + x._2.mkString(", ")))                                  : Unit
    }
```

- filter out results with empty sets
- sort results by key, then sort the sets
- print results

Code



```
def discoverINDs(inputs: List[String], spark: SparkSession): Unit = {
  import spark.implicits._

  val valuesWithEmptySet = mutable.HashSet[String]()
  inputs.map(input => readData(input, spark))       // list of files -> list of Spark Datasets
    .map(ds => {                                     // Spark Dataset -> tuples of value and corresponding columnName
      val columns = ds.columns
      ds.flatMap(row => {
        for (i <- columns.indices) yield {
          (row.getString(i), columns(i))
        }
      }) // (value, colName)
    })                                                                                                                    : List[Dataset[(String, String)]]
    .reduce(_ union _)                                                                                                    : Dataset[(String, String)]
    .groupByKey(x => x._1)                           // group by value --> get unique values with corresponding colNames
    .mapGroups((_, it) => it.map(elem => elem._2).toSet)  // throw away value, only keep columnNames as Sets
    .flatMap(set => {                                // get each possible combination of elements in the set (candidates)
      if(set.size == 1)
        valuesWithEmptySet.add(set.head)
      set.map(colName => (colName, set - colName))
    })                                                                                                                    : Dataset[(String, Set[String])]
    .groupByKey(x => x._1)                           // group by candidate                                                : KeyValueGroupedDataset[String, (String, ...)]
    .mapGroups((s, it) =>
      (s, it
        .dropWhile(_ => valuesWithEmptySet.contains(s)) // (if not relevant, empty iterator to skip map-reduce)
        .map(x => x._2)
        .reduce((a, b) => a.intersect(b)))
    )                                                                                                                     : Dataset[(String, Set[String])]
    .filter(elem => elem._2.nonEmpty)                // throw away results with empty sets
    .collect()                                       // collect to Array
    .sortBy(x => x._1)                               // sort keys
    .map(x => (x._1, x._2.toList.sorted))            // sort values
    .foreach(x => println(x._1 + " < " + x._2.mkString(", ")))                                                            : Unit
}
```