Distributed Data Management

# Exercise Evaluation Assignment 4

Thorsten Papenbrock

F-2.04, Campus II

Hasso Plattner Institut


APACHE
Spark™

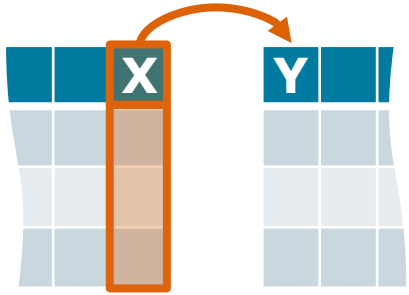
# Inclusion Dependency Discovery

**Definition:** Given two relational instances $r_i$ and $r_j$ for the schemata $R_i$, and $R_j$, respectively. The inclusion dependency $R_i[X] \subseteq R_j[Y]$ (short $X \subseteq Y$ ) with $X \subseteq R_i$, $Y \subseteq R_j$ and $|X| = |Y|$ is valid, iff $\forall\, t_i[X] \in r_i$ , $\exists t_j[Y] \in r_j : t_i[X] = t_j[Y]$.

"All values in X are also contained in Y"

We consider only unary INDs for this task, i.e., $|X| = |Y| = 1$

foreign-key candidates



Assignment 4: Spark

# Inclusion Dependency Discovery

| Name | Type | Equatorial diameter | Mass | Orbital radius | Orbital period | Rotation period | Confirmed moons | Rings | Atmosphere |
|---|---|---|---|---|---|---|---|---|---|
| Mercury | Terrestrial | 0.382 | 0.06 | 0.47 | 0.24 | 58.64 | 0 | no | minimal |
| Venus | Terrestrial | 0.949 | 0.82 | 0.72 | 0.62 | −243.02 | 0 | no | $CO_2$, $N_2$ |
| Earth | Terrestrial | 1.000 | 1.00 | 1.00 | 1.00 | 1.00 | 1 | no | $N_2$, $O_2$, Ar |
| Mars | Terrestrial | 0.532 | 0.11 | 1.52 | 1.88 | 1.03 | 2 | no | $CO_2$, $N_2$, Ar |
| Jupiter | Giant | 11.209 | 317.8 | 5.20 | 11.86 | 0.41 | 67 | yes | $H_2$, He |
| Saturn | Giant | 9.449 | 95.2 | 9.54 | 29.46 | 0.43 | 62 | yes | $H_2$, He |
| Uranus | Giant | 4.007 | 14.6 | 19.22 | 84.01 | −0.72 | 27 | yes | $H_2$, He |
| Neptune | Giant | 3.883 | 17.2 | 30.06 | 164.8 | 0.67 | 14 | yes | $H_2$, He |

| Planet | Rotation Period | Revolution Period |
|---|---|---|
| Mercury | 58.6 days | 87.97 days |
| Venus | 243 days | 224.7 days |
| Earth | 0.99 days | 365.26 days |
| Mars | 1.03 days | 1.88 years |
| Jupiter | 0.41 days | 11.86 years |
| Saturn | 0.45 days | 29.46 years |
| Uranus | 0.72 days | 84.01 years |
| Neptune | 0.67 days | 164.79 years |
| Pluto | 6.39 days | 248.59 years |

| Symbol | Unicode | Glyph |
|---|---|---|
| Sun | U+2609 | ☉ |
| Moon | U+263D | ☽ |
| Moon | U+263E | ☾ |
| Mercury | U+263F | ☿ |
| Venus | U+2640 | ♀ |
| Earth | U+1F728 | 🜨 |
| Mars | U+2642 | ♂ |
| Jupiter | U+2643 | ♃ |
| Saturn | U+2644 | ♄ |
| Uranus | U+2645 | ♅ |
| Uranus | U+26E2 | ⛢ |
| Neptune | U+2646 | ♆ |
| Eris | ≈ U+2641 | ♁ |
| Eris | ≈ U+29EC | ⧬ |
| Pluto | U+2647 | ♇ |
| Pluto | not present | -- |
| Aries | U+2648 | ♈ |
| Taurus | U+2649 | ♉ |
| Gemini | U+264A | ♊ |
| Cancer | U+264B | ♋ |
| Leo | U+264C | ♌ |
| Virgo | U+264D | ♍ |
| Libra | U+264E | ♎ |
| Scorpio | U+264F | ♏ |
| Sagittarius | U+2650 | ♐ |
| Capricorn | U+2651 | ♑ |
| Capricorn | U+2651 | ♑ |
| Aquarius | U+2652 | ♒ |
| Pisces | U+2653 | ♓ |
| Conjunction | U+260C | ☌ |
| ... | ... | ... |

| Planet | Synodic period | Synodic period (mean) | Days in retrograde |
|---|---|---|---|
| Mercury | 116 | 3.8 | ~21 |
| Venus | 584 | 19.2 | 41 |
| Mars | 780 | 25.6 | 72 |
| Jupiter | 399 | 13.1 | 121 |
| Saturn | 378 | 12.4 | 138 |
| Uranus | 370 | 12.15 | 151 |
| Neptune | 367 | 12.07 | 158 |

| Planet | Mean distance | Relative mean distance |
|---|---|---|
| Mercury | 57.91 | 1 |
| Venus | 108.21 | 1.86859 |
| Earth | 149.6 | 1.3825 |
| Mars | 227.92 | 1.52353 |
| Ceres | 413.79 | 1.81552 |
| Jupiter | 778.57 | 1.88154 |
| Saturn | 1,433.53 | 1.84123 |
| Uranus | 2,872.46 | 2.00377 |
| Neptune | 4,495.06 | 1.56488 |
| Pluto | 5,869.66 | 1.3058 |

| Sign | House | Domicile | Detriment | Exaltation | Fall | Planetary Joy |
|---|---|---|---|---|---|---|
| Aries | 1st House | Mars | Venus | Sun | Saturn | Mercury |
| Taurus | 2nd House | Venus | Pluto | Moon | Uranus | Jupiter |
| Gemini | 3rd House | Mercury | Jupiter | N/A | N/A | Saturn |
| Cancer | 4th House | Moon | Saturn | Jupiter | Mars | Venus |
| Leo | 5th House | Sun | Uranus | Neptune | Mercury | Mars |
| Virgo | 6th House | Mercury | Neptune | Pluto, Mercury | Venus | Saturn |
| Libra | 7th House | Venus | Mars | Saturn | Sun | Moon |
| Scorpio | 8th House | Pluto | Venus | Uranus | Moon | Saturn |
| Sagittarius | 9th House | Jupiter | Mercury | N/A | N/A | Sun |
| Capricorn | 10th House | Saturn | Moon | Mars | Jupiter | Mercury |
| Aquarius | 11th House | Uranus | Sun | Mercury | Neptune | Venus |
| Pisces | 12th House | Neptune | Mercury | Venus | Pluto, Mercury | Moon |

| Planet | Calculated (in AU) | Observed (in AU) | Perfect octaves | Actual distance |
|---|---|---|---|---|
| Mercury | 0.4 | 0.387 | 0 | 0 |
| Venus | 0.7 | 0.723 | 1 | 1.1 |
| Earth | 1 | 1 | 2 | 2 |
| Mars | 1.6 | 1.524 | 4 | 3.7 |
| Asteroid belt | 2.8 | 2.767 | 8 | 7.8 |
| Jupiter | 5.2 | 5.203 | 16 | 15.7 |
| Saturn | 10 | 9.539 | 32 | 29.9 |
| Uranus | 19.6 | 19.191 | 64 | 61.4 |
| Neptune | 38.8 | 30.061 | 96 | -96.8 |
| Pluto | 77.2 | 39.529 | 128 | 127.7 |



http://webdatacommons.org/webtables/index.html

# Inclusion Dependency Discovery - Rules

## Assignment

- Expected output
  - Write the discovered INDs lexicographically sorted to the console
  - Use the following style for your output:
    `<dependent> < <referenced1>, <referenced2>, …`
    - So the correct output should look as follows:

```
C_CUSTKEY < P_PARTKEY
C_NATIONKEY < S_NATIONKEY, N_NATIONKEY
L_COMMIT < L_SHIP, L_RECEIPT
L_LINENUMBER < C_NATIONKEY, S_NATIONKEY, O_ORDERKEY, L_SUPPKEY, N_NATIONKEY, S_SUPPKEY, P_PARTKEY, P_SIZE, C_CUSTKEY, L_PARTKEY
L_LINESTATUS < O_ORDERSTATUS
L_ORDERKEY < O_ORDERKEY
L_PARTKEY < P_PARTKEY
L_SUPPKEY < P_PARTKEY, S_SUPPKEY, C_CUSTKEY
L_TAX < L_DISCOUNT
N_NATIONKEY < C_NATIONKEY, S_NATIONKEY
N_REGIONKEY < C_NATIONKEY, S_NATIONKEY, N_NATIONKEY, R_REGIONKEY
O_CUSTKEY < P_PARTKEY, C_CUSTKEY
O_SHIPPRIORITY < C_NATIONKEY, S_NATIONKEY, N_REGIONKEY, N_NATIONKEY, R_REGIONKEY
P_SIZE < L_SUPPKEY, S_SUPPKEY, P_PARTKEY, C_CUSTKEY, L_PARTKEY
R_REGIONKEY < C_NATIONKEY, S_NATIONKEY, N_REGIONKEY, N_NATIONKEY
S_NATIONKEY < C_NATIONKEY, N_NATIONKEY
S_SUPPKEY < L_SUPPKEY, P_PARTKEY, C_CUSTKEY
```

Assignment 4: Spark

# Inclusion Dependency Discovery - Hint

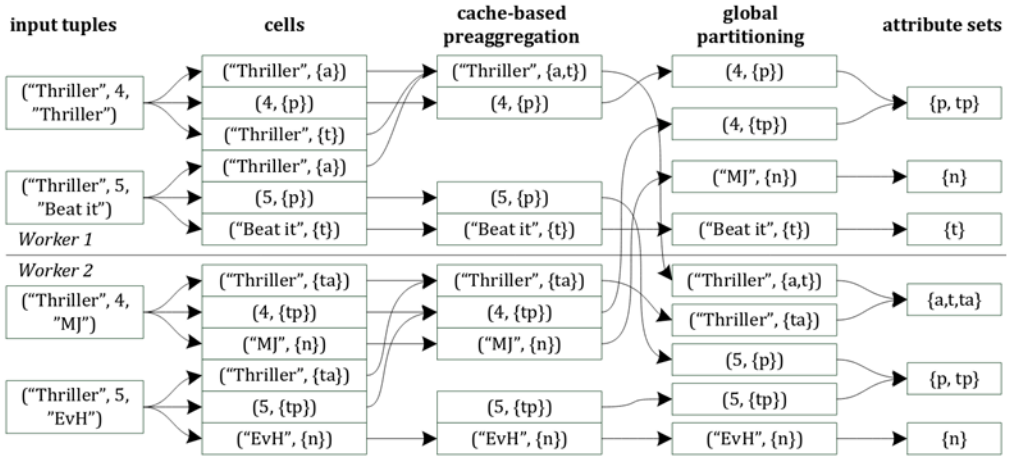



Figure 1: Dataflow for a distributed full outer join on all columns of a dataset. The attribute names and some values are abbreviated for the purpose of lucidity.

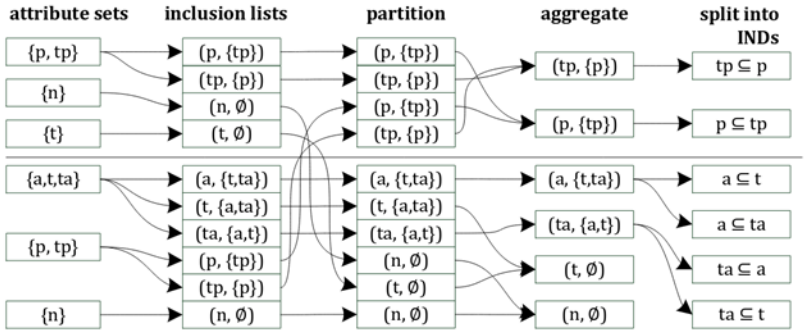



Figure 2: Dataflow for deriving INDs from the outer join product.

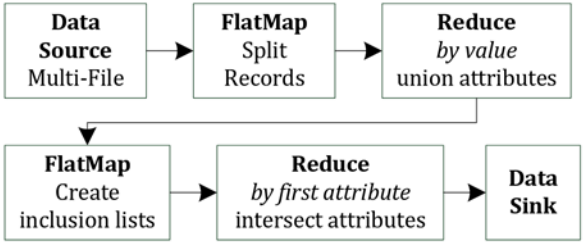



Figure 3: Stratosphere plan for unary IND detection.

**Scaling Out the Discovery of Inclusion Dependencies**
S. Kruse, T. Papenbrock, F. Naumann,
Proceedings of the conference on Database Systems for Business, Technology, and Web (BTW). pp. 445-454 (2015).

```scala
import org.apache.spark.sql.functions._
import org.apache.spark.sql._

val inputs = List("region", "nation", "supplier", "customer", "lineitem", "orders", "part")

def readData(fileName: String): Dataset[Row] = {
  spark
    .read
    .option("inferSchema", "false")
    .option("header", "true")
    .option("quote", "\"")
    .option("delimiter", ";")
    .csv(s"data/TPCH/tpch_$fileName.csv")
}

val data = inputs.map(readData)

def pairValuesToAttribute(dataframe: Dataset[Row]): Dataset[(String, String)] = {
  val columnNames = dataframe.columns
  dataframe
    .flatMap(r => columnNames.map(column => (r.getString(r.fieldIndex(column)), column)))
    .toDF("value", "attribute")
    .as[(String, String)]
}

val allValueAttributePairs = data
  .map(pairValuesToAttribute)
  .reduce(_.union(_))
```

```scala
import org.apache.spark.sql.functions._
import org.apache.spark.sql._

val inputs = List("region", "nation", "supplier", "customer", "lineitem", "orders", "part")

def readData(fileName: String): Dataset[Row] = {
  spark
    .read
    .option("inferSchema", "false")
    .option("header", "true")
    .option("quote", "\"")
    .option("delimiter", ";")
    .csv(s"data/TPCH/tpch_$fileName.csv")
}

val data = inputs.map(readData)

def pairValuesToAttribute(dataframe: Dataset[Row]): Dataset[(String, String)] = {
  val columnNames = dataframe.columns
  dataframe
    .flatMap(r => columnNames.map(column => (r.getString(r.fieldIndex(column)), column)))
    .toDF("value", "attribute")
    .as[(String, String)]
}

val allValueAttributePairs = data
  .map(pairValuesToAttribute)
  .reduce(_.union(_))
```

# Evaluation

| Team | executes? | correct? | terminates? |
|---|---|---|---|
| AlpAkka | | | |
| Alpha | | | |
| BlockchainOnAkka | | | |
| Chewbakka | | | |
| Code Monkeys | | | |
| Dally | | | |
| ddm_team_42 | | | |
| Distributed Wealth | | | |
| Duftes Daten Mischen (DDM) | | | |
| Euphorische Elefanten | | | |
| mAKKAronis | | | |
| MeMyselfAndSomeoneElse | | | |
| Multiprocessing Moguls | | | |
| So Called Engineers | | | |
| supreme-broccoli | | | |
| Taube_Nuesschen | | | |
| the_reapers | | | |
| Unknown Pleasures | | | |
| w00t? | | | |
| sindy | | | |

# Evaluation

| Team | executes? | correct? | terminates? |
|---|---|---|---|
| AlpAkka | no | | |
| Alpha | yes | | |
| BlockchainOnAkka | yes | | |
| Chewbakka | yes | | |
| Code Monkeys | no | | |
| Dally | yes | | |
| ddm_team_42 | yes | | |
| Distributed Wealth | yes | | |
| Duftes Daten Mischen (DDM) | yes | | |
| Euphorische Elefanten | yes | | |
| mAKKAronis | yes | | |
| MeMyselfAndSomeoneElse | yes | | |
| Multiprocessing Moguls | yes | | |
| So Called Engineers | yes | | |
| supreme-broccoli | yes | | |
| Taube_Nuesschen | yes | | |
| the_reapers | yes | | |
| Unknown Pleasures | yes | | |
| w00t? | yes | | |
| sindy | yes | | |

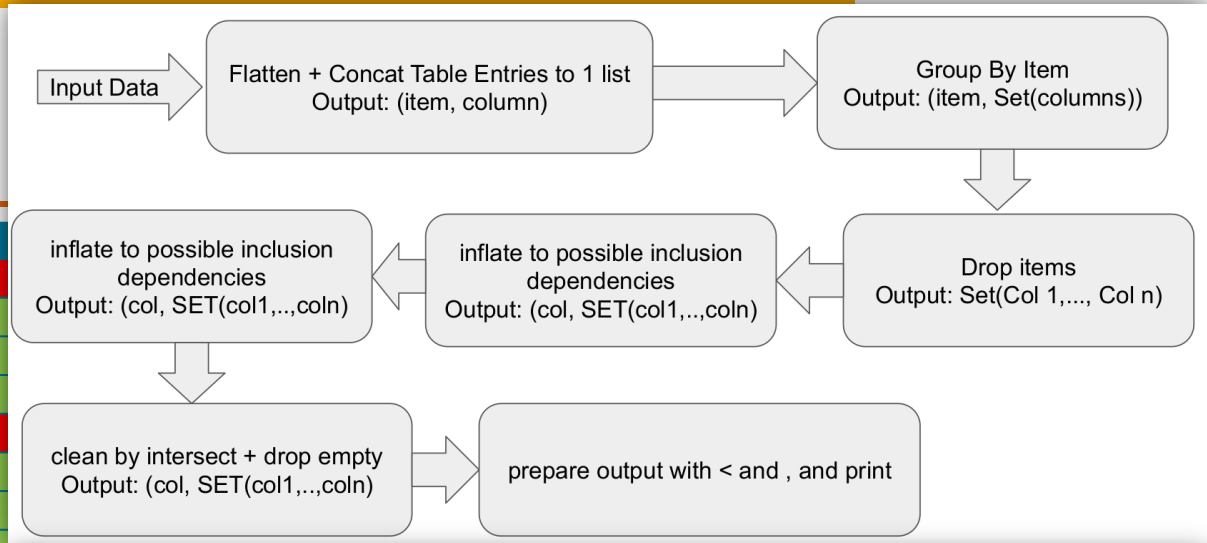
```
org.apache.spark.SparkException: Job aborted due to stage failure: Total size of serialized results of 10783 tasks (1024.0 MiB) is bigger than spark.driver.maxResultSize (1024.0 MiB)
```

Collect() action at the end of your pipelines collected more than 1GB result data even on a <1MB dataset …

There is `spark.driver.maxResultSize`, but how do the results become so huge?

# Assignment 4: Spark
# Evaluation

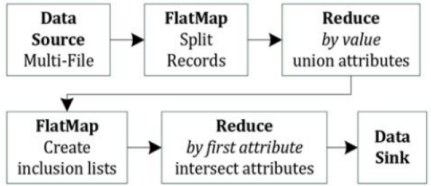
| Team | executes? |
|---|---|
| AlpAkka | no |
| Alpha | yes |
| BlockchainOnAkka | yes |
| Chewbakka | yes |
| Code Monkeys | no |
| Dally | yes |
| ddm_team_42 | yes |
| Distributed Wealth | yes |
| Duftes Daten Mischen (DDM) | yes |
| Euphorische Elefanten | yes |
| mAKKAronis | yes |
| MeMyselfAndSomeoneElse | yes |
| Multiprocessing Moguls | yes |
| So Called Engineers | yes |
| supreme-broccoli | yes |
| Taube_Nuesschen | yes |
| the_reapers | yes |
| Unknown Pleasures | yes |
| w00t? | yes |
| sindy | yes |




Für das Lösen der Aufgabe haben wir uns an das in der Vorlesung vorgestelltes Paper (*Scaling Out the Discovery of Inclusion Dependencies S. Kruse, T. Papenbrock, F. Naumann)* orientiert.

Unser Algorithmus besteht aus folgenden Schritten:

1. CSV Daten lesen
2. Daten in einzelne Spalten konvertieren und mappen
   a. Die geladenen Tabellen/Dataframes splitten
   b. Duplikate aus den einzelnen Spalten entfernen
   c. Mapping von Werten zu Attributen `{(v1 -> a1),(v1 -> a2)...}`
3. Attribute anhand der Werte gruppieren `(v1 -> {a1, a2})`
4. Listen für Inclusion Dependencies erzeugen `(a1, {a2})`
5. Inclusion Dependencies check:
   a. Gruppiere anhand des ersten Attributs `(a1 -> {{a2},{a3}})`
   b. Schnittmenge der Attribute bilden `{a2 ∩ a3}`
6. Filtern von leeren Listen
7. Daten sammeln (collect) und ausgeben



# Evaluation

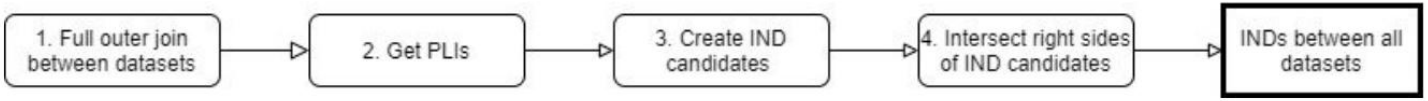
| Team | executes? | correct? | terminates? |
|---|---|---|---|
| AlpAkka | no | - | |
| Alpha | yes | yes | |
| BlockchainOnAkka | yes | mostly | |
| Chewbakka | yes | yes | |
| Code Monkeys | no | - | |
| Dally | yes | yes | |
| ddm_team_42 | yes | yes | |
| Distributed Wealth | yes | yes | |
| Duftes Daten Mischen (DDM) | yes | yes | |
| Euphorische Elefanten | yes | yes | |
| mAKKAronis | yes | yes | |
| MeMyselfAndSomeoneElse | yes | yes | |
| Multiprocessing Moguls | yes | yes | |
| So Called Engineers | yes | yes | |
| supreme-broccoli | yes | yes | |
| Taube_Nuesschen | yes | yes | |
| the_reapers | yes | mostly | |
| Unknown Pleasures | yes | no | |
| w00t? | yes | mostly | |
| sindy | yes | yes | |

No/wrong result sortation

Assignme
Evalua

| Team | | |
|---|---|---|
| AlpAkka | | |
| Alpha | | |
| BlockchainOnAkka | | |
| Chewbakka | | |
| Code Monkeys | | |
| Dally | | |
| ddm_team_42 | | |
| Distributed Wealth | | |
| Duftes Daten Mische | | |
| Euphorische Elefant | | |
| mAKKAronis | | |
| MeMyselfAndSomec | | |
| Multiprocessing Mog | | |
| So Called Engineers | | |
| supreme-broccoli | | |
| Taube_Nuesschen | | |
| the_reapers | yes | mostly |
| Unknown Pleasures | yes | no |
| w00t? | yes | mostly |
| sindy | yes | yes |

# SINDY




- Read files as Spark dataframes → Spark takes care of parallelization under the hood
- Relatively straightforward to implement

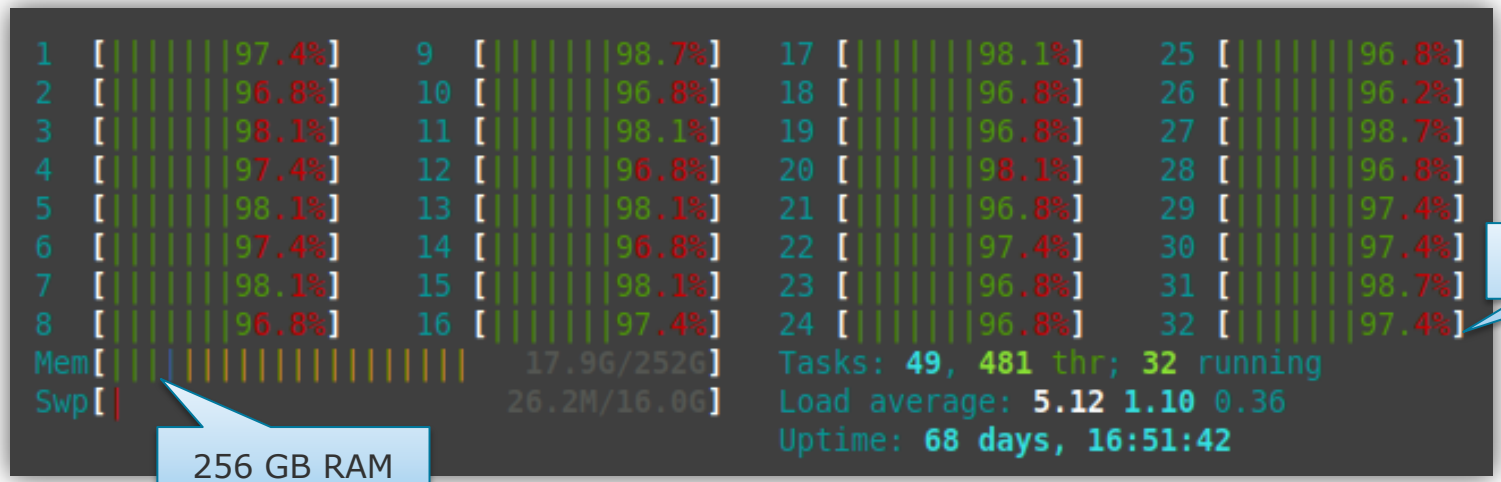


Result contains false positives

# Evaluation

| Team | executes? | correct? | terminates? |
|---|---|---|---|
| AlpAkka | no | - | - |
| Alpha | yes | yes | yes |
| BlockchainOnAkka | yes | mostly | yes |
| Chewbakka | yes | yes | yes |
| Code Monkeys | no | - | - |
| Dally | yes | yes | yes |
| ddm_team_42 | yes | yes | yes |
| Distributed Wealth | yes | yes | yes |
| Duftes Daten Mischen (DDM) | yes | yes | yes |
| Euphorische Elefanten | yes | yes | yes |
| mAKKAronis | yes | yes | yes |
| MeMyselfAndSomeoneElse | yes | yes | yes |
| Multiprocessing Moguls | yes | yes | yes |
| So Called Engineers | yes | yes | yes |
| supreme-broccoli | yes | yes | yes |
| Taube_Nuesschen | yes | yes | yes |
| the_reapers | yes | mostly | yes |
| Unknown Pleasures | yes | no | yes |
| w00t? | yes | mostly | yes |
| sindy | yes | yes | yes |

# Evaluation Server & Datasets




**TPC-H small**

10 kB

**TPC-H full**

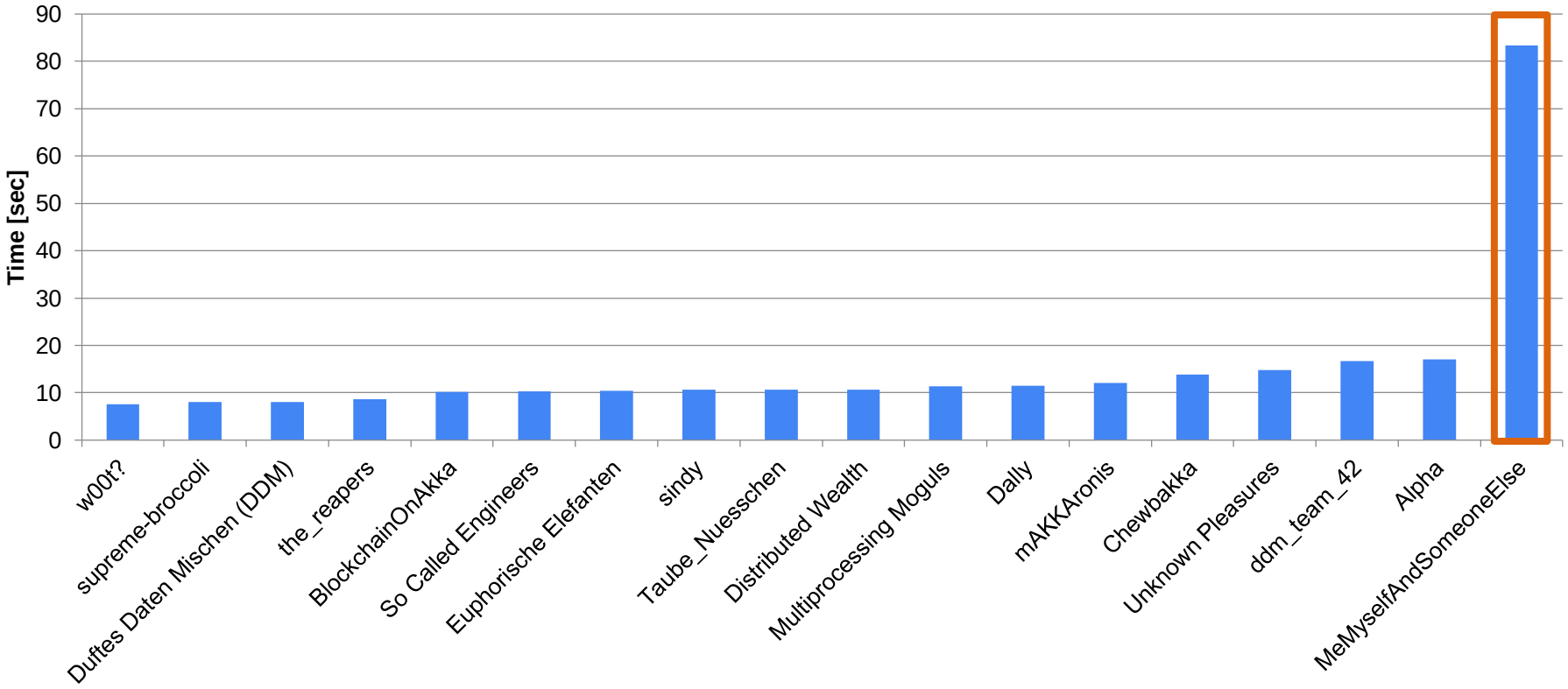
446 MB

**TPC-H large**

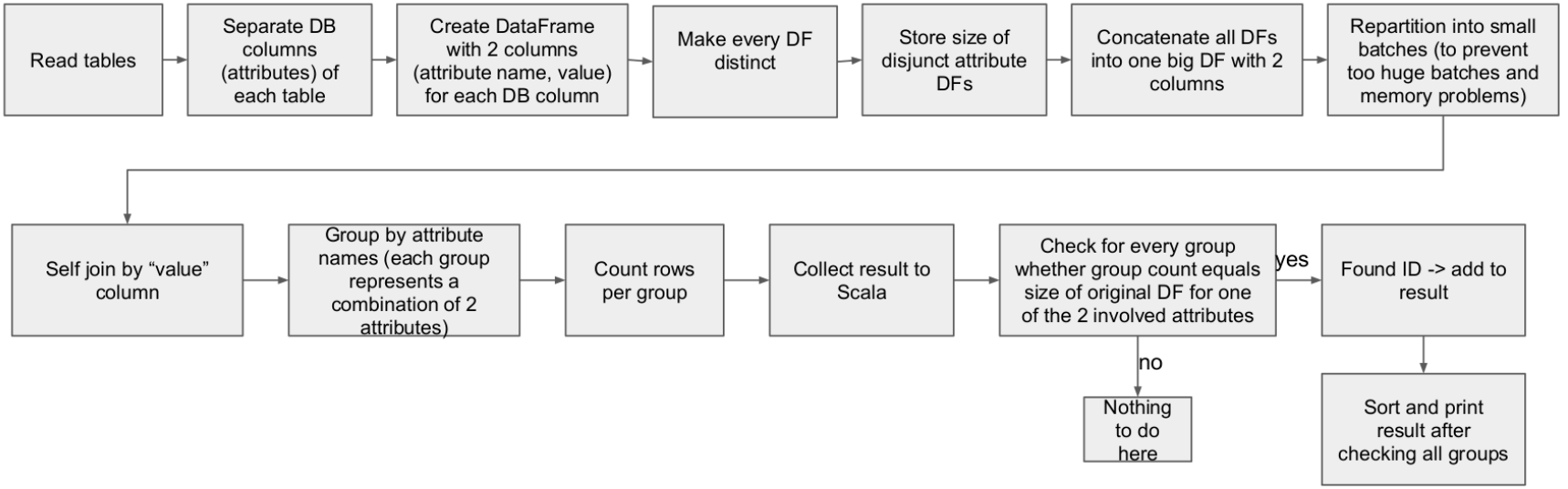
1.3 GB

Assignment 4: Spark

# Evaluation – TPC-H small (10kB)



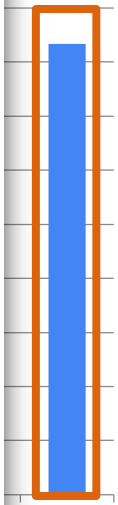
# General Workflow




Remarks:

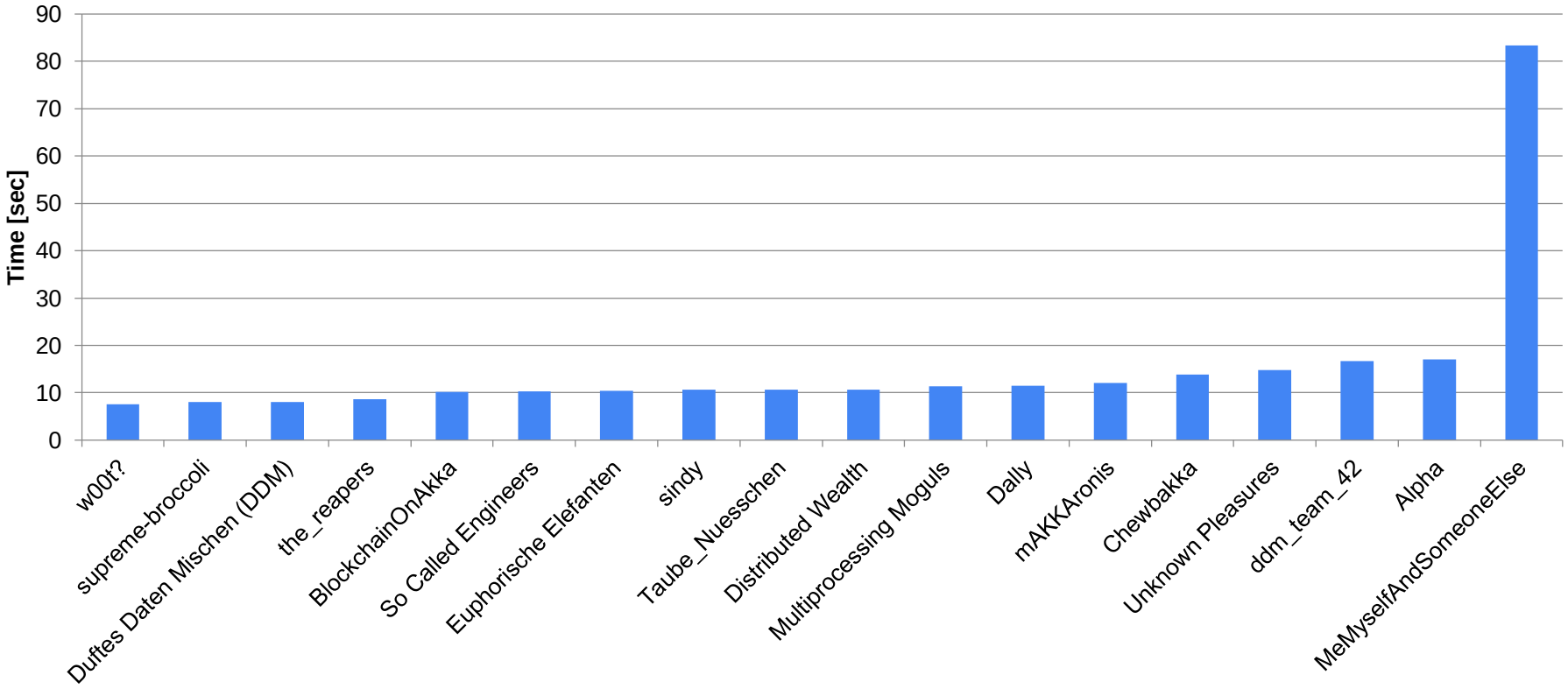
- Set spark configuration *spark.driver.maxResultSize* to "8g" to prevent memory problems with limited performance Laptop which we used for development

Hasso Plattner Institut



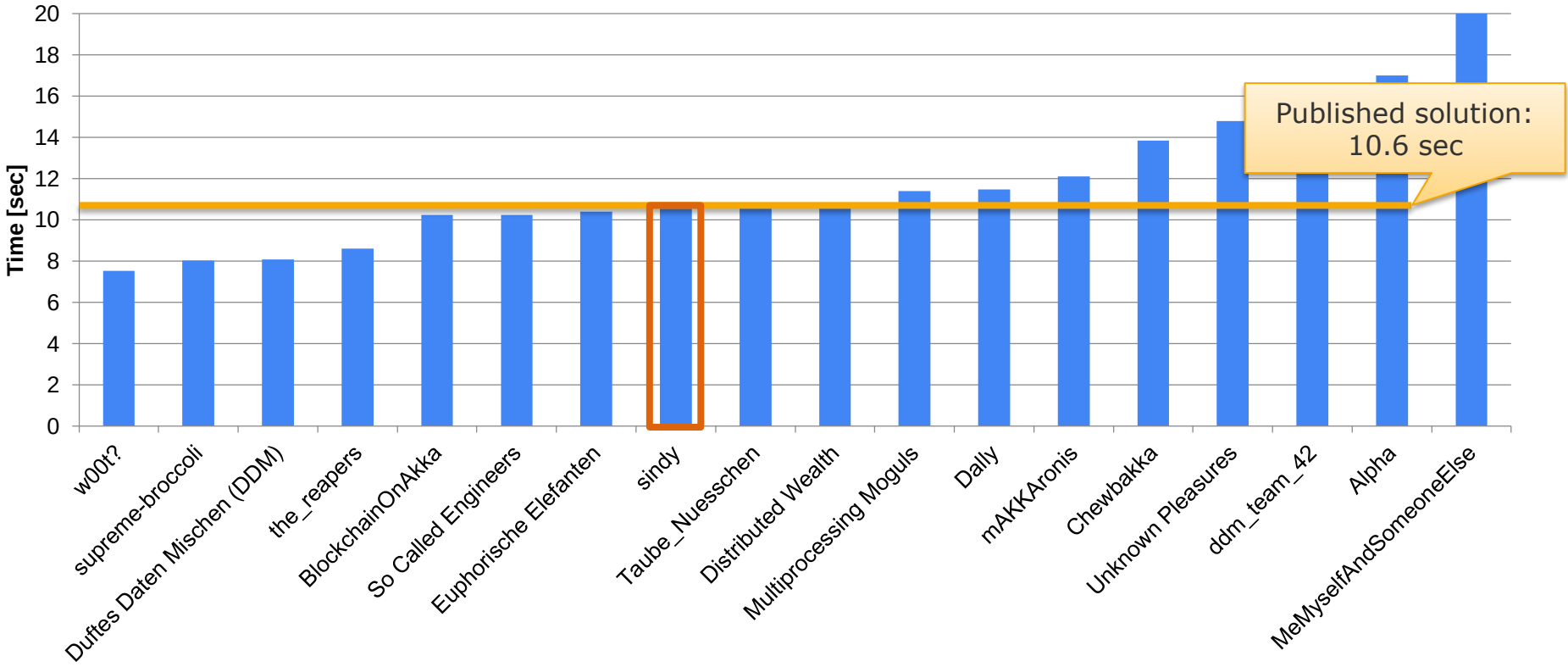
Assignment 4: Spark

# Evaluation – TPC-H small (10kB)

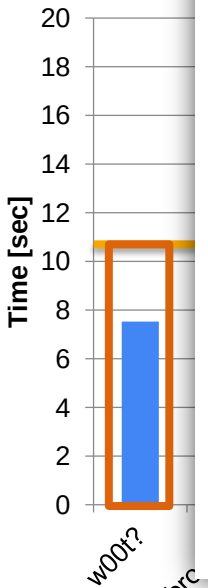


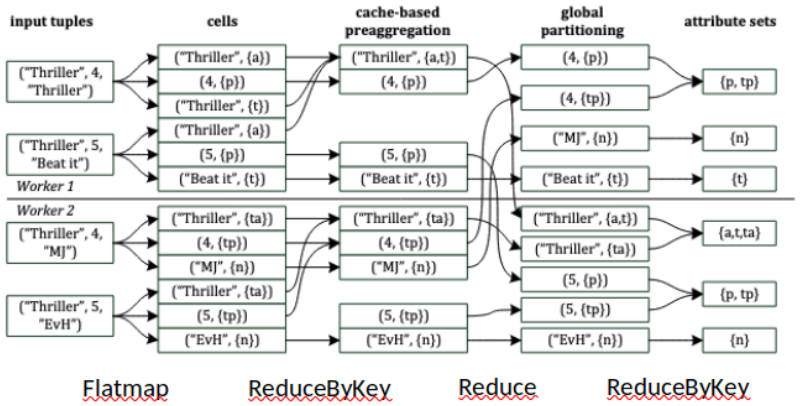
Assignment 4: Spark

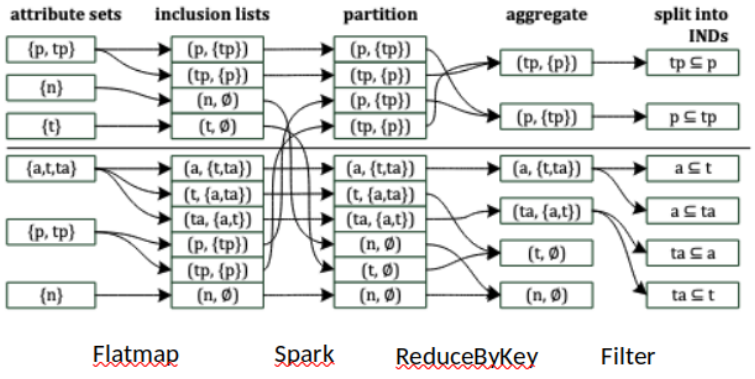
# Evaluation – TPC-H small (10kB)



# Evaluation – TPC-H small (10kB)



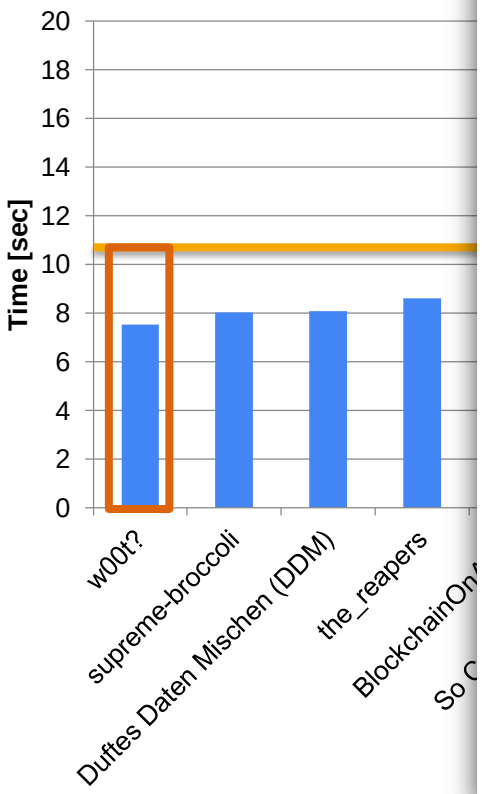




- We tried to replicate the paper's pipeline using Spark's RDDs (see above)

Assignment 4: Spark

# Evaluation – TPC-H small (10kB)

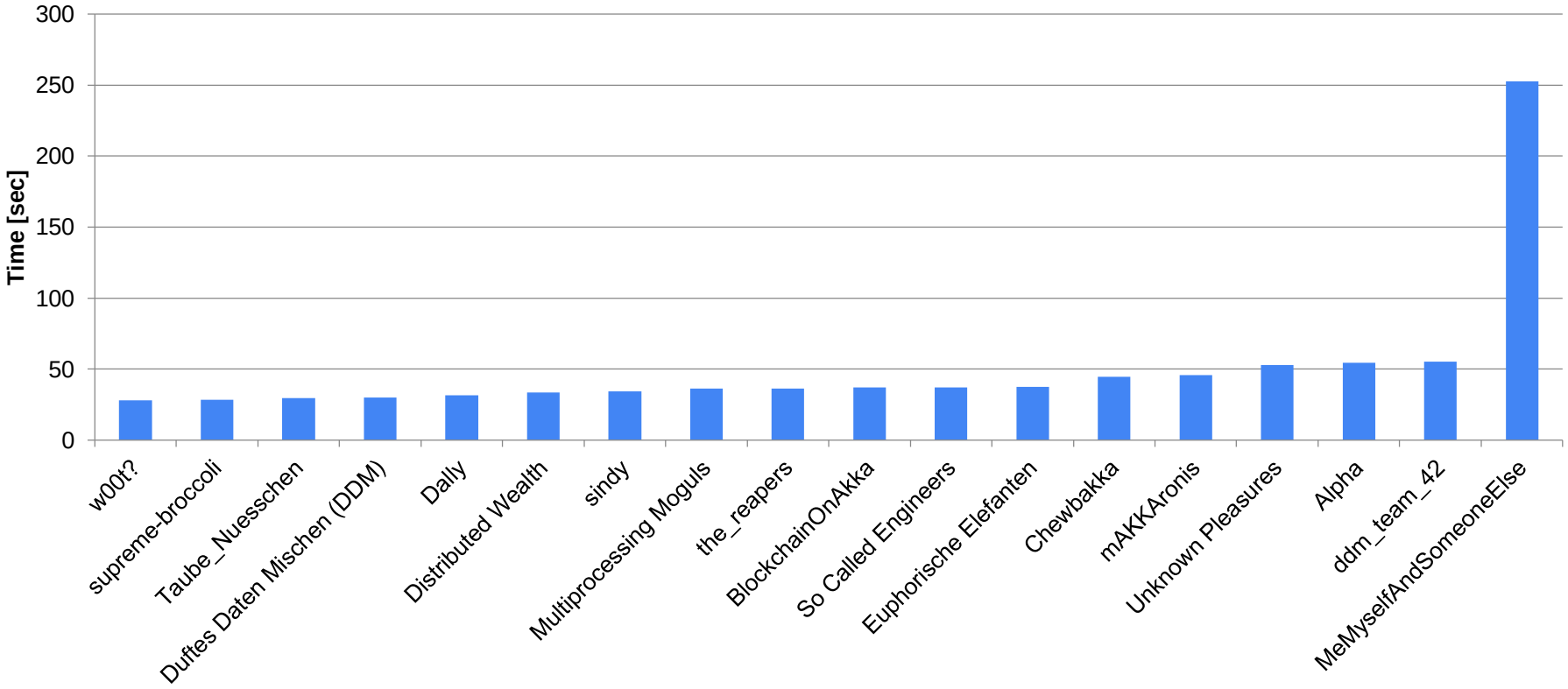



```scala
object Sindy {

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

    val data = inputs.map(input => {
      spark.read
        .option("header", "true")
        .option("delimiter", ";")
        .csv(input)
    })
    val cells = data.map(df => {
      val column_names = df.columns.map(List(_))
      df.flatMap(row => {
        row.toSeq.map(_.toString).zip(column_names)
      })
    })
    val cells_rdd = cells.map(df=>df.rdd)
    val cache_based_preaggregation = cells_rdd.map(rdd => rdd.reduceByKey((a, b) => (a ++ b).distinct))
    val global_partitioning = cache_based_preaggregation.reduce((a, b) => a.union(b))
    val attribute_sets = global_partitioning.reduceByKey((a, b) => (a ++ b).distinct).map(_._2)
    val inclusion_lists = attribute_sets.flatMap(attributeSet => {
      attributeSet.map(column => (column, attributeSet.filterNot(_ == column)))
    })
    //    val partition = inclusion_lists.reduce((a, b) => a.union(b)) // why no shuffle/reduce here?
    val aggregate = inclusion_lists.reduceByKey((a, b) => a.intersect(b))
    val splitted_inds = aggregate.filter(a => a._2.nonEmpty)
    val output = splitted_inds.map(a => s"${a._1} < ${a._2.mkString(", ")}")
    output.collect().foreach(println(_))
  }
}
```

Super parallel and minimized network traffic,
but become slow for larger dataset due to list copying.

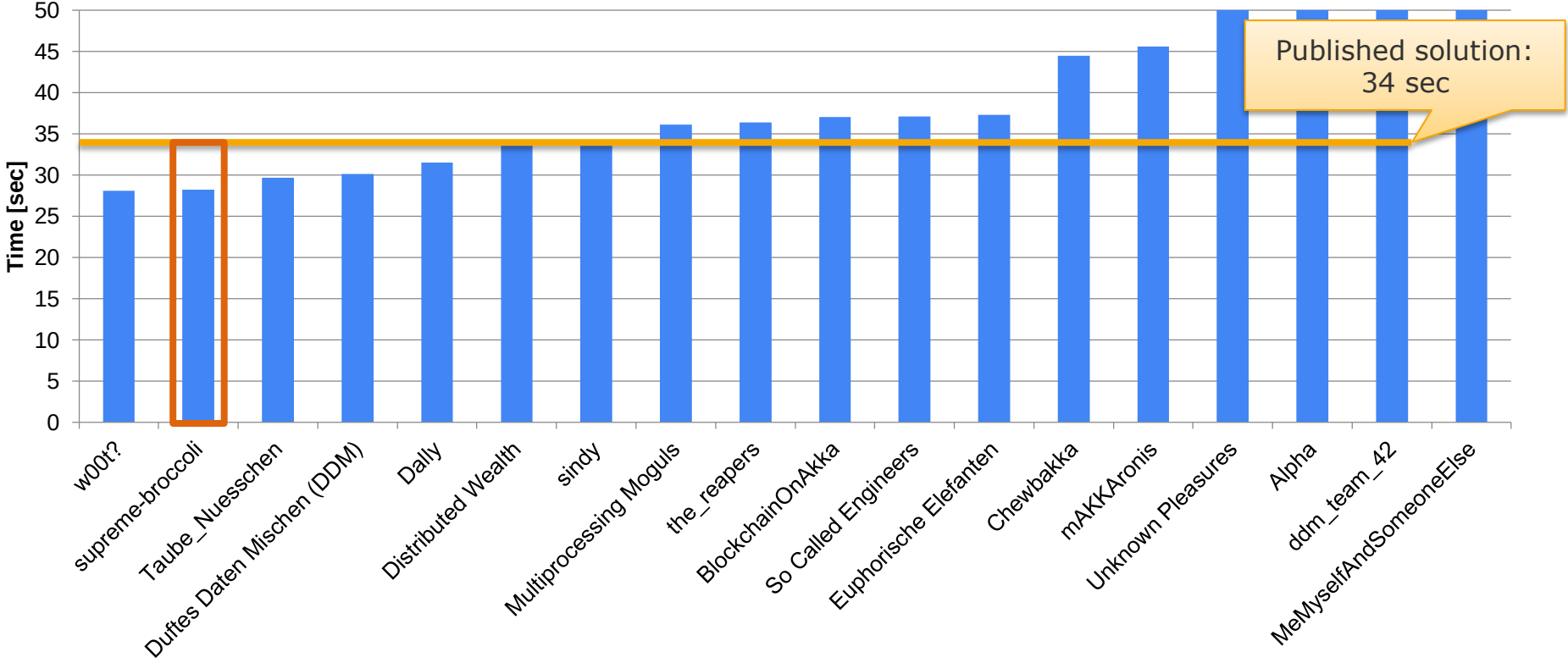
Assignment 4: Spark

# Evaluation – TPC-H full (446 MB)



Assignment 4: Spark

# Evaluation – TPC-H full (446 MB)



# Assignment 4
# Group: supreme-broccoli 👑🥦

Caution:
Code was simplified and shortened

## 1) Zip every cell with column name

```
map(df => {df.flatMap(row => row.toSeq.map(p => p.zip(columns))
```

## 2) Reduce by value union attributes

```
.reduce((a, b) => a.union(b)).reduceByKey((a, b) => (a ++ b).distinct).map(_._2)
```

## 3) Create inclusion lists

```
.flatMap(set => { set.map(column => (column , set - column)) })
```

## 4) Reduce by intersect and drop empty ones

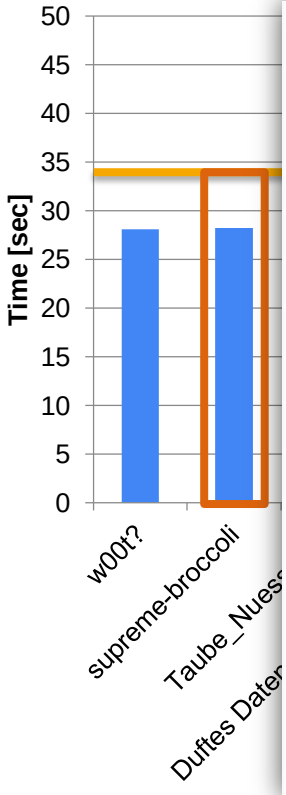
```
.reduceByKey(_ intersect _).filter(_._2.nonEmpty)
```

## 5) Collect and Print

```
.collect().sortBy(_._1).foreach(println())
```

Assignment 4: Spark

# Evaluation – TPC-H full (446 MB)

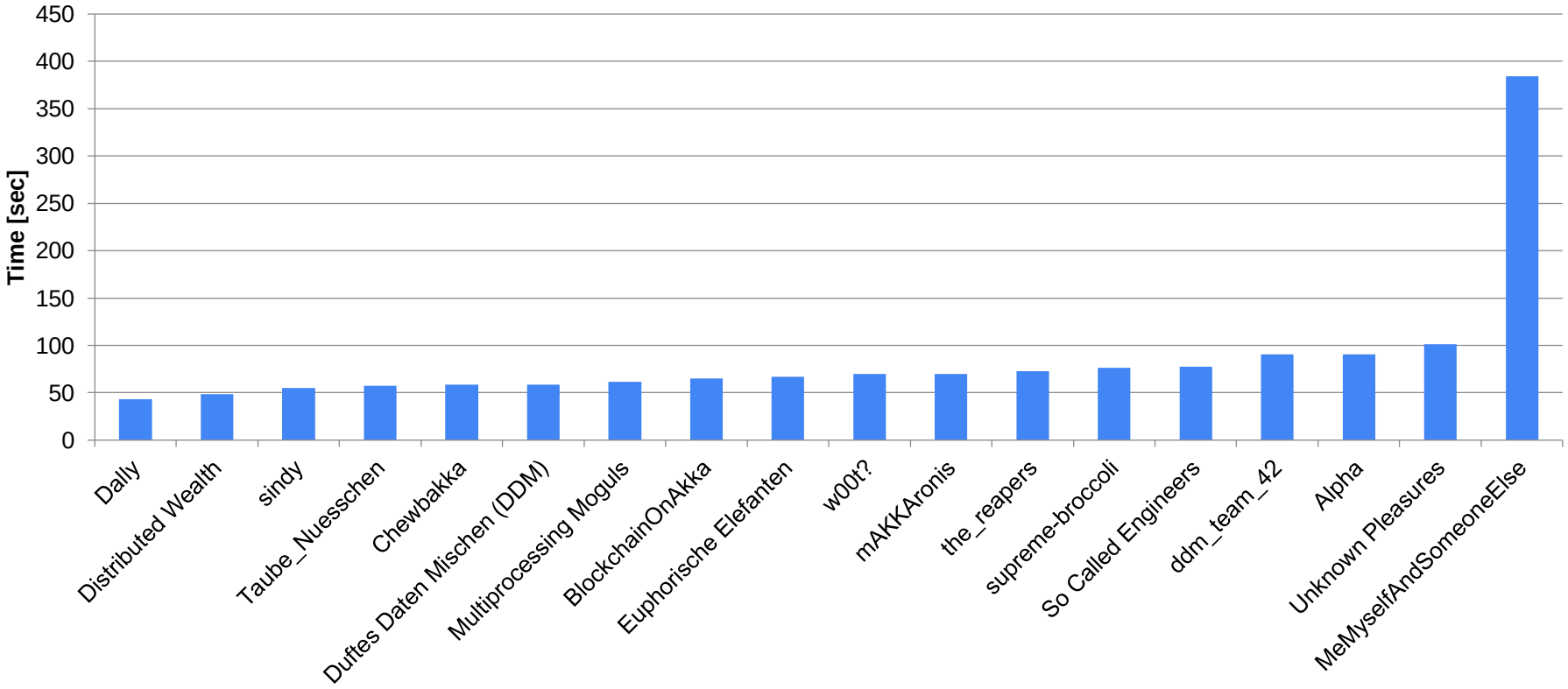



```scala
object Sindy {

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

    // stratosphere plan: data source multi-file
    val dfs = inputs.map(table => spark.read
      .option("sep", ";")
      .option("header", "true")
      .csv(table)
    )

    // step 1: -> "cells": zip every value of every cell with column name -> [("Thriller", (a)), ("Thriller", (t)), ("Thriller", (p))]
    // step 2: -> "cache-based preaggr.": pre-aggregate all values that occur multiple times -> (("Thriller", (a, t)) in worker 1, ("Thriller", (p)) in worker 2
    // stratosphere plan: flat-map split records; Step 1 + 2 can be combined
    val cells = dfs.map(df => {
      val columns = df.columns.map(name => List(name))
      val cells = df.flatMap(row => row.toSeq.map(p => String.valueOf(p)).zip(columns))
      cells.rdd.reduceByKey((a, b) => (a ++ b).distinct)
    })

    // stratosphere plan: reduce by value union attributes -> {a,t,p} {c}
    val attributeSets = cells.reduce((a, b) => a union b)
                             .reduceByKey((a, b) => (a ++ b).distinct)
                             .map(_._2)

    // stratosphere plan: flatmap create inclusion lists
    val inclusionLists = attributeSets
      .flatMap(attributeSet => {
        attributeSet.map(column => (column , attributeSet.toSet - column))
      })

    inclusionLists.reduceByKey(_ intersect _)
                  .filter(_._2.nonEmpty)
                  .collect()
                  .sortBy(_._1)
                  .foreach(n => println(s"${n._1} < ${n._2.mkString(", ")}")) // print all INDs
  }
}
```

Super parallel and minimized network traffic,
but become slow for larger dataset due to list copying.

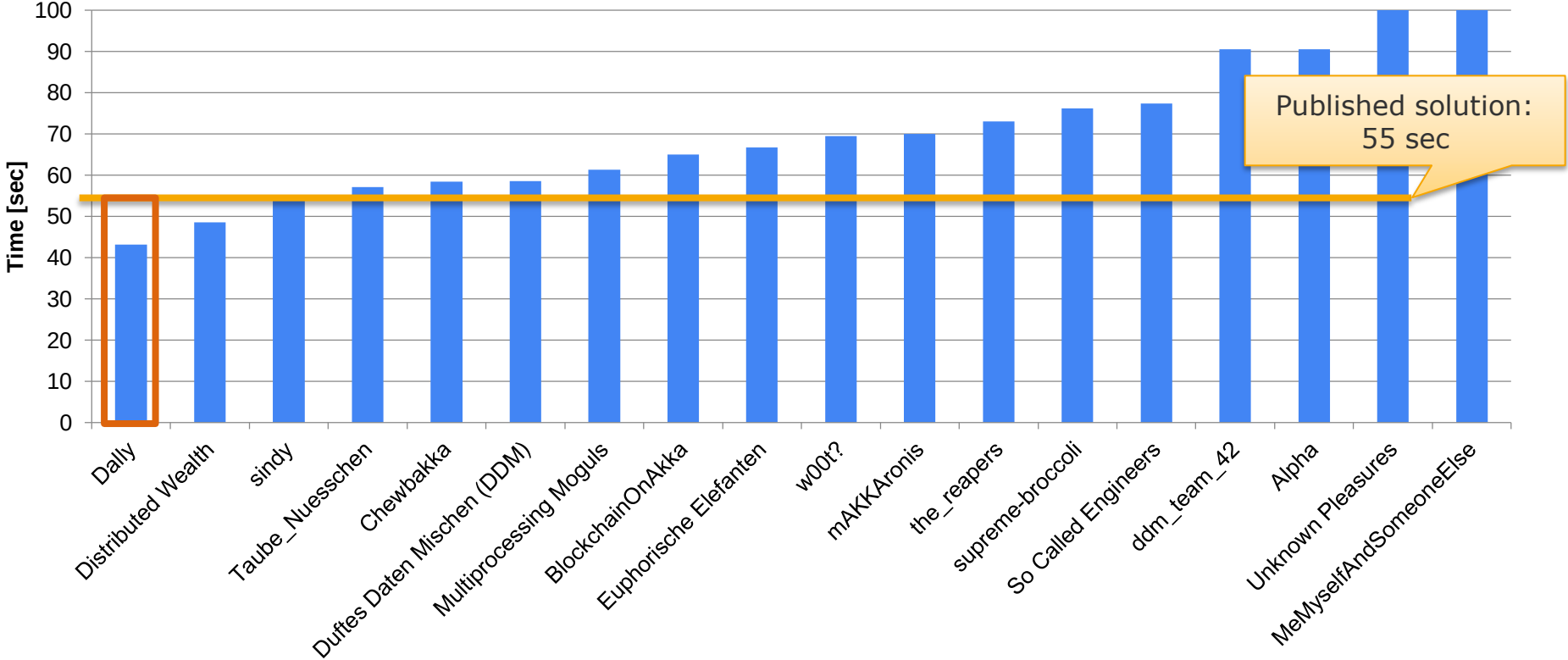
Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)

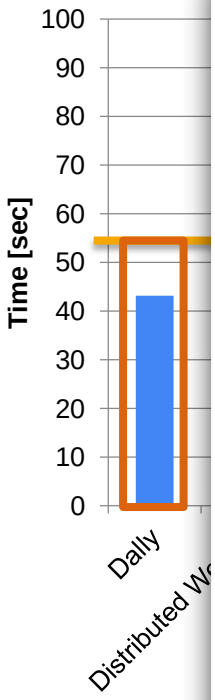


Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)



# Evaluation – TPC-H large (1.3 GB)

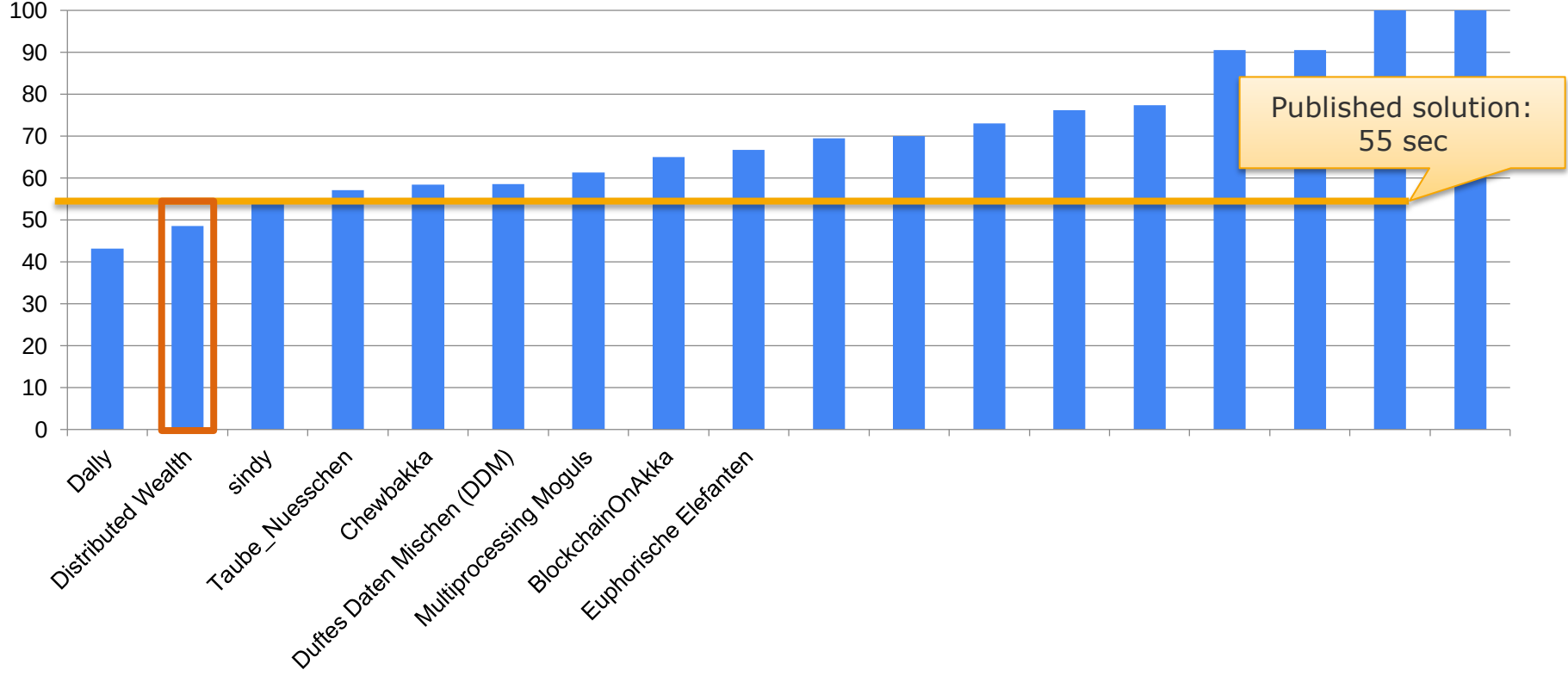



```scala
object Sindy {

  def discoverINDs(inputs: List[String], spark: SparkSession): Unit = {
    import org.apache.spark.sql.sources.DataSourceRegister
    import spark.implicits._

    val tableSetList = inputs.map(f => spark.read
      .option("inferSchema","true")
      .option("header", "true")
      .option("delimiter", ";")
      .csv(f)
      .flatMap(row => row.schema.fields.zipWithIndex.map(tuple => (row.get(tuple._2).toString,tuple._1.name))))

    val unionTables = tableSetList.reduce { (table1, table2) => table1.union(table2) }
      .dropDuplicates

    val groupedValues = unionTables.groupByKey(_._1)

    val keysToSet = groupedValues.mapGroups { case (_, rows) => rows.map(_._2).toSet }
      .dropDuplicates

    val inclusion = keysToSet.flatMap(set => set.map(key => (key, set - key)))

    val intersect = inclusion.groupByKey(_._1).reduceGroups((a, b) => (a._1, a._2.intersect(b._2))).map {case (a,(_,b)) => (a,b) }
      .filter(_._2.nonEmpty)
      .sort("_1")


      intersect
      .collect()
      .foreach { case (dependentKey, referencedKey) => println(dependentKey + " < " + referencedKey.toList.sorted.reduce(_ + "," + _)) }
  }
}
```

Super effective here,
because TPC-H is generated from a small seed,
but ineffective on non-generated datasets,
because deduplication costs are high.

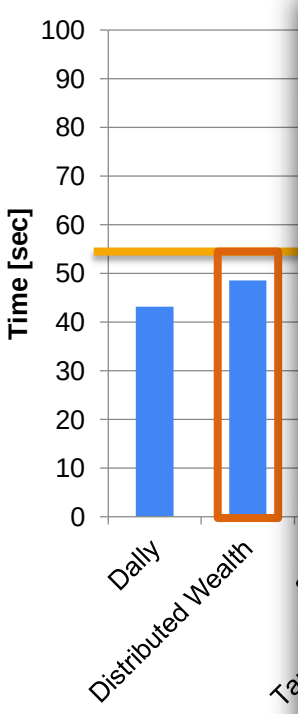
Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)

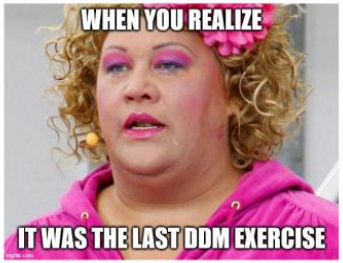


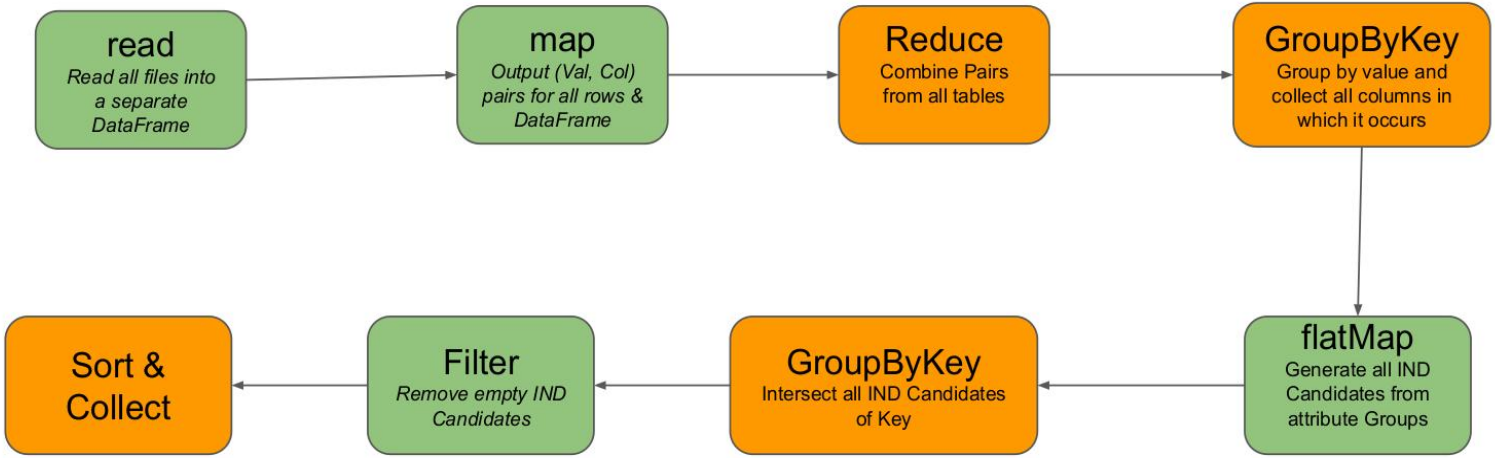
Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)



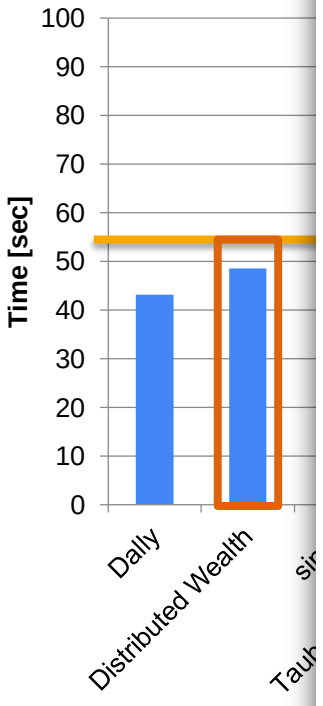

Cindy

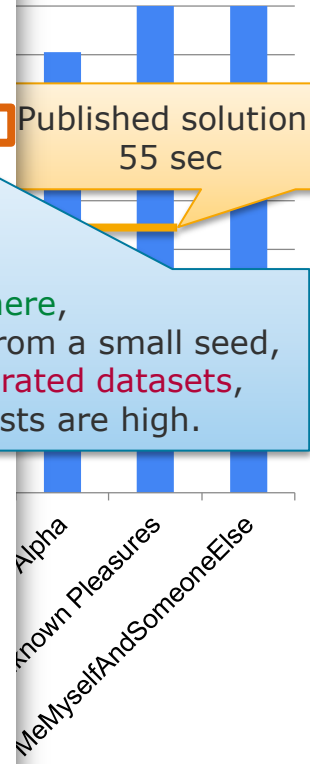


# Assignment 4: Evaluation

```scala
// Read files
val data = inputs.map(inputFile => spark.read
  .option("delimiter", ";")
  .option("header", "true")
  .csv(inputFile))

// Output (value, col) Tuples
val valueColumnPairs = data.map(df => {
  val cols = df.columns
  df.flatMap(row => {
    for (i <- cols.indices) yield {
      (row.getString(i), cols(i))
    }
  })// <- dataset
}) // <- list of datasets
//combine results from files into a single rdd
val flattenedPairs = valueColumnPairs.reduce((combined, newPair) => combined.union(newPair)).distinct()

// Create Attribute Groups
val attributeGroups = flattenedPairs
  .groupByKey(t => t._1)
  .mapGroups((_, iterator) => iterator.map(t => t._2).toSet)

// Generate IND Candidates
val INDCandidates = attributeGroups.flatMap(group => {
  group.map(col => (col, group - col))
})

// Generate all INDs
val allINDs = INDCandidates
  .groupByKey(t => t._1)
  .mapGroups((key, iterator) => (key, iterator
    .map(t => t._2)
    .reduce((intersected, newCandidate) => intersected.intersect(newCandidate))))
  .filter(t => t._2.nonEmpty)

// Output to Console
val sorted = allINDs.sort("_1")

def rowToString(row: (String, Set[String])): String = {
  row._1 + " < " + row._2.reduce((total, newString) => total + ", " + newString)
}

for (line <- sorted.collect) {
  println(rowToString(line))
}
```



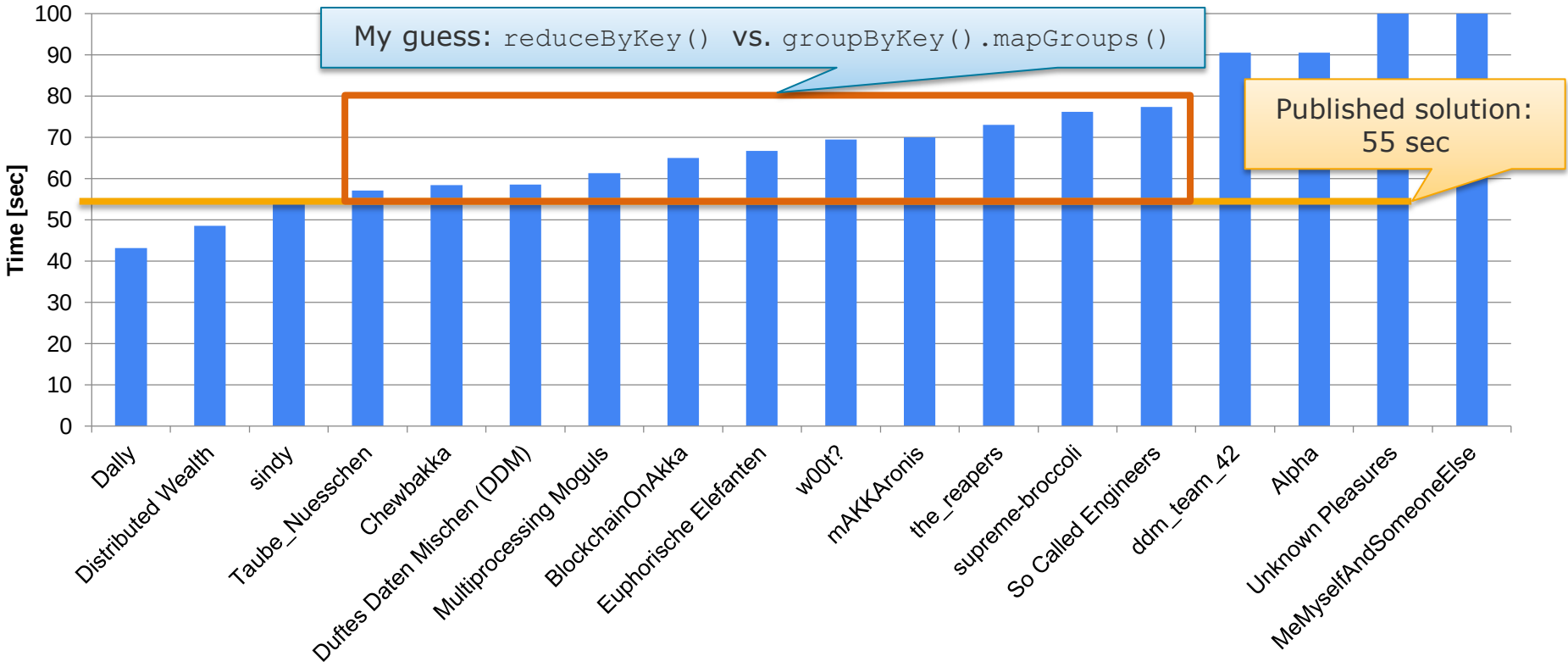

Published solution:
55 sec

Super effective here,
because TPC-H is generated from a small seed,
but ineffective on non-generated datasets,
because deduplication costs are high.

HPI Hasso Plattner Institut

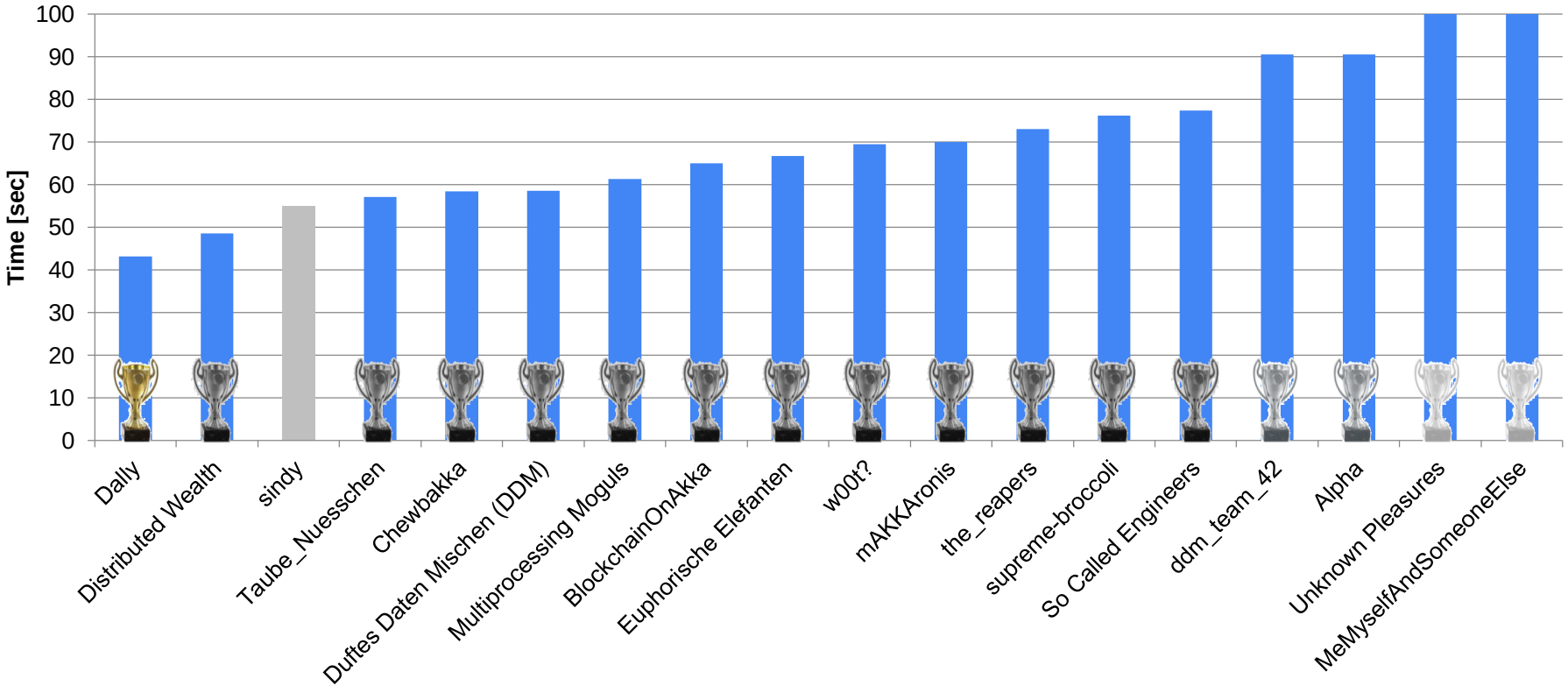
Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)



Assignment 4: Spark

# Evaluation – TPC-H large (1.3 GB)



Assignment 4: Spark

# 2018 Submissions

```scala
def discoverINDs(inputs: List[String], spark: SparkSession):
  import spark.implicits._
  val datasets = inputs.map(
    spark.read
      .option("inferSchema", "false")
      .option("header", "true")
      .option("sep", ";")
      .csv(_)
  )
  val cells = datasets.map(dataSet => {
    val datasetColumns = dataSet.columns
    dataSet.flatMap(row => row.toSeq.asInstanceOf[Seq[String]].zip(datasetColumns))
  }).reduce(_ union _).distinct()
  val attributeSets = cells.rdd.groupByKey().mapValues(_.toSet).values.distinct()
  val inclusionLists = attributeSets.flatMap(attributeSet =>
    attributeSet.map(x => (x , attributeSet - x))
  )
  val results = inclusionLists.reduceByKey(_ intersect _).filter(_._2.nonEmpty)
  results.foreach(resultPair => println(s"${resultPair._1} < ${resultPair._2.mkString(", ")}"))
}
```

Probably not worth the effort …

```scala
val allValueAttributePairs = data
  .map(pairValuesToAttribute)
  .reduce(_.union(_))

val inds = allValueAttributePairs
  .groupBy("value")
  .agg(collect_set("attribute") as "attributes")
  .select("attributes")
  .distinct()
  .withColumn("dependent", explode(col("attributes")))
  .as[(Seq[String], String)]
  .map(t => (t._2, t._1.filter(!_.equals(t._2))))
  .toDF("attribute", "attributes")
  .as[(String, Seq[String])]
  .groupByKey(t => t._1)
  .mapGroups{ case (key, iterator) => (key, iterator.map(t => t._2).reduce(_.intersect(_))) }
  .filter(_._2.size > 0)
  .select($"_1" as "dependent", $"_2" as "referenced")
  .as[(String, Seq[String])]
  .collect()

inds.sortBy(_._1).foreach(t => println(t._1 + " < " + t._2.mkString(", ")))
```

Assignment 4: Spark

# 2018 Submissions

HPI Hasso Plattner Institut

```
val allValueAttributePairs = data
  .map(pairValuesToAtt
  .reduce(_.union(_))

val inds = allValueAttributePairs
  .groupBy("value")
  .agg(collect_set("attribute") as "attributes")
  .select("attributes")
  .distinct()
  .withColumn("dependent", explode(col("attributes")))
  .as[(Seq[String], String)]
  .map(t => (t._2, t._1.filter(!_.equals(t._2))))
  .toDF("attribute", "attributes")
  .as[(String, Seq[String])]
  .groupByKey(t => t._1)
  .mapGroups{ case (key, iterator) => (key, iterator.m
  .filter(_._2.size > 0)
  .select($"_1" as "dependent", $"_2" as "referenced")
  .as[(String, Seq[String])]
  .collect()

inds.sortBy(_._1).foreach(t => println(t._1 + " < " + t._2.mkString(", ")))
```

Probably memory issues due to large intermediate state …

```
paths
  .map(loadFile)
  .map(toCells)
  .reduce(_.union(_))
  .groupByKey(_.value)
  .mapGroups((value, cells) => {
    val attributeUnion = cells.map(_.attributes).reduce(_.union(_))
    Cell(value, attributeUnion)
  })
  .flatMap(cell =>
    cell.attributes.map(attribute =>
      Cell(attribute, cell.attributes — attribute)
    )
  )
  .groupByKey(_.value)
  .mapGroups((value, cells) => {
    val attributeIntersection = cells.map(_.attributes).reduce(_.intersect(_))
    Cell(value, attributeIntersection)
  })
  .filter(_.attributes.nonEmpty)
  .collect
  .sortWith(_.value < _.value)
```