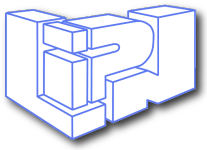
Clustering with Spark-MapReduce **Vichy'14**

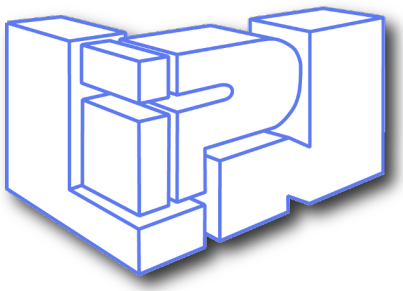
Tugdual Sarazin, Mustapha Lebbah, Hanene Azzag

tugdual.sarazin@altic.org
@TugdualSarazin

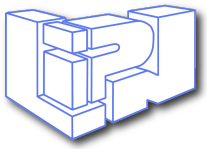


CIFRE

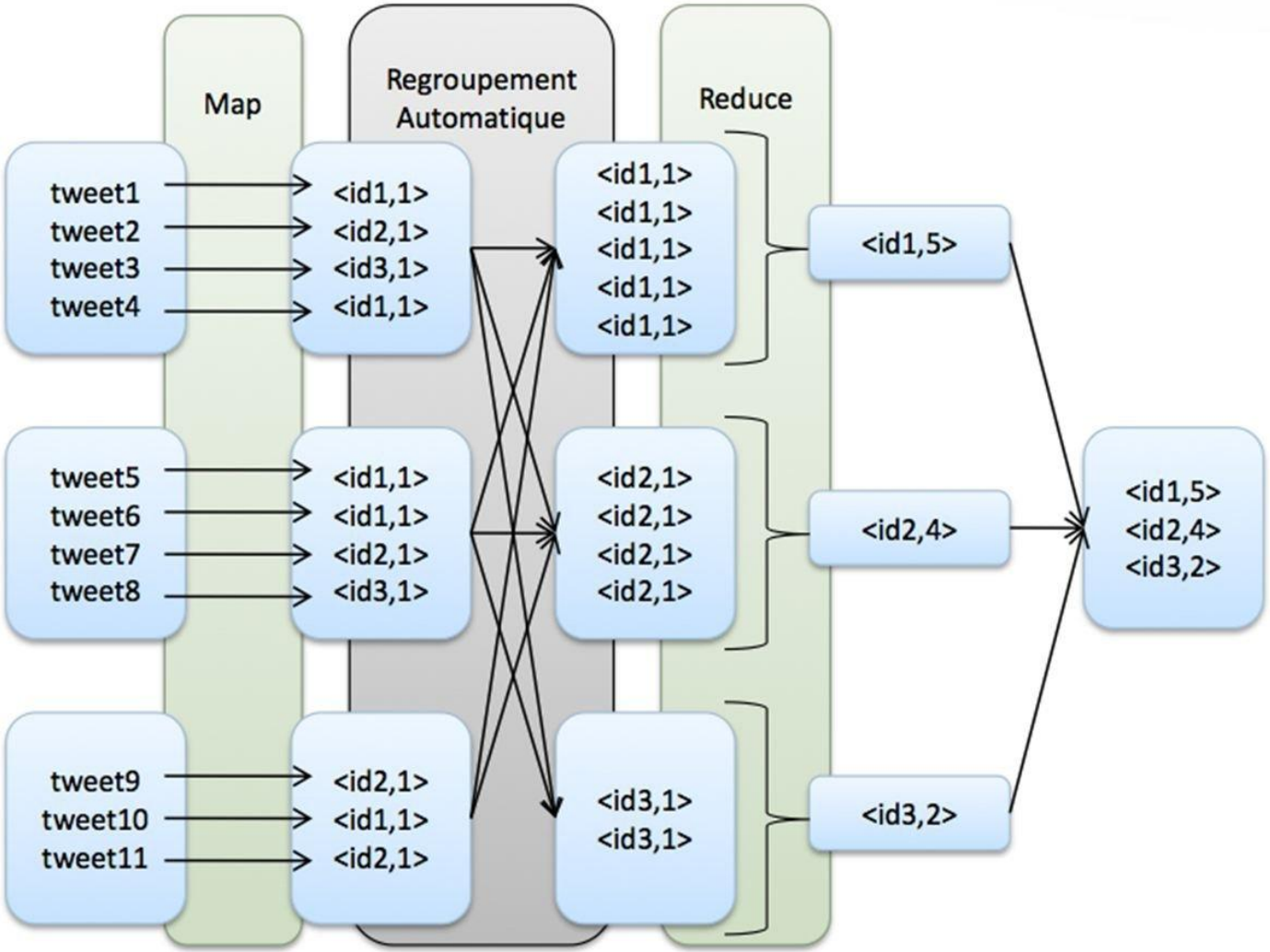
- Business Intelligence
- Data integration
- Open Source
- BigData and machine learning



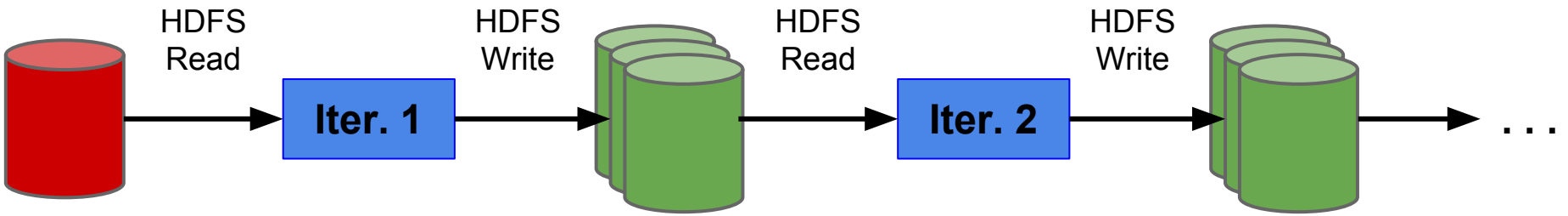
- Computer science laboratory of Paris-Nord University
- A3 team : Machine learning and applications
- Encadrants: M. Lebbah, H. Azzag



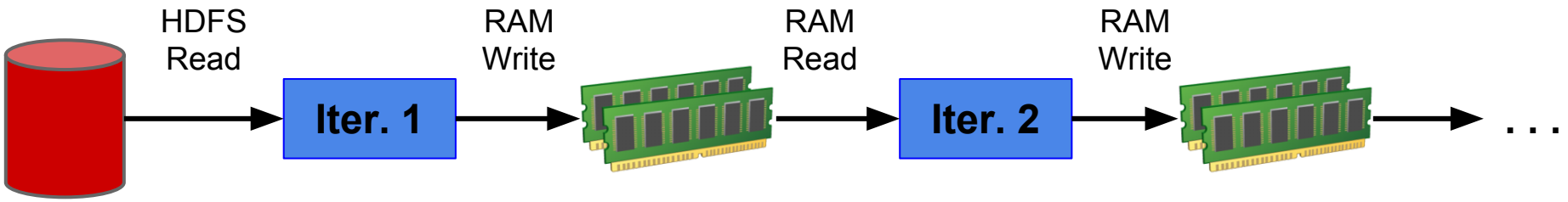
MapReduce

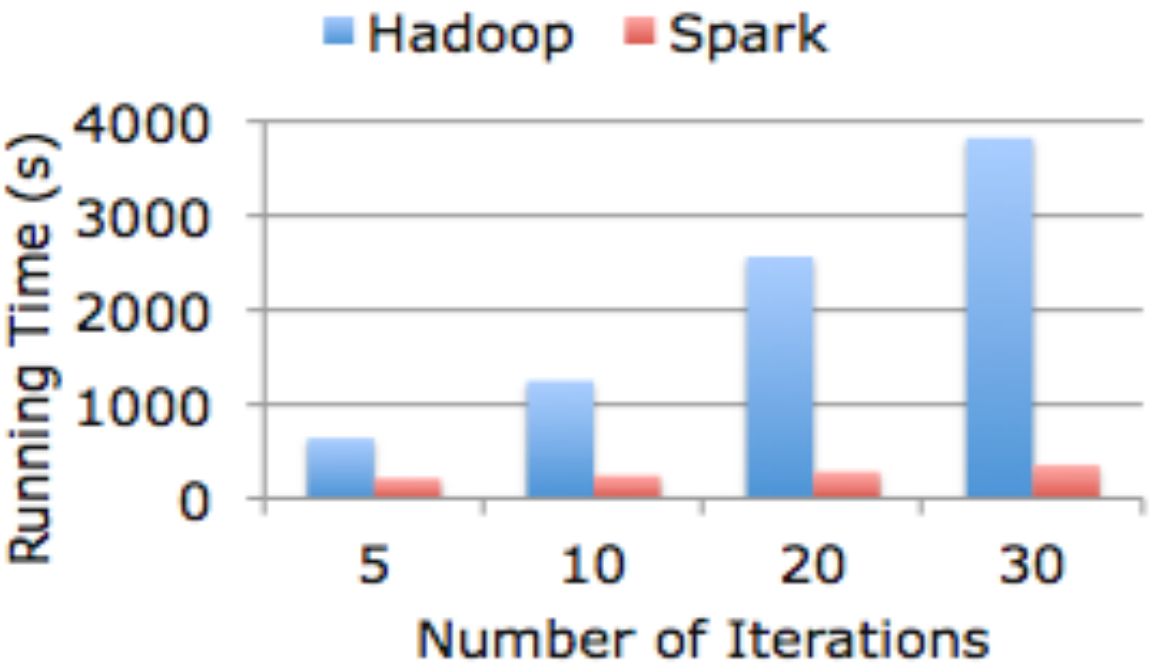

Map
Regroupement Automatique
Reduce
tweet1
tweet2
tweet3
tweet4
tweet5
tweet6
tweet7
tweet8
tweet9
tweet10
tweet11
<id1,1>
<id2,1>
<id3,1>
<id1,1>
<id1,1>
<id1,1>
<id2,1>
<id3,1>
<id2,1>
<id1,1>
<id2,1>
<id1,1>
<id1,1>
<id1,1>
<id1,1>
<id1,1>
<id2,1>
<id2,1>
<id2,1>
<id2,1>
<id3,1>
<id3,1>
<id1,5>
<id2,4>
<id3,2>
<id1,5>
<id2,4>
<id3,2>

Machine learning with Hadoop MapReduce

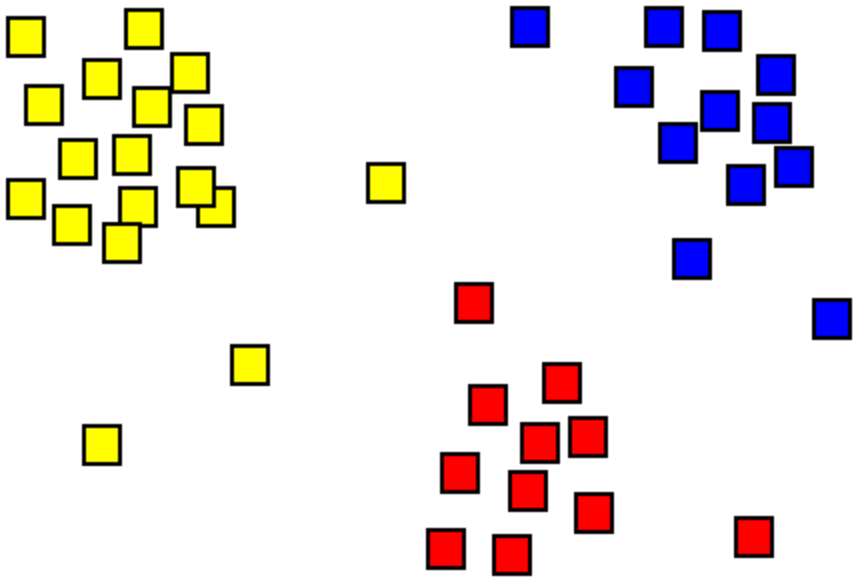


Machine learning with Spark





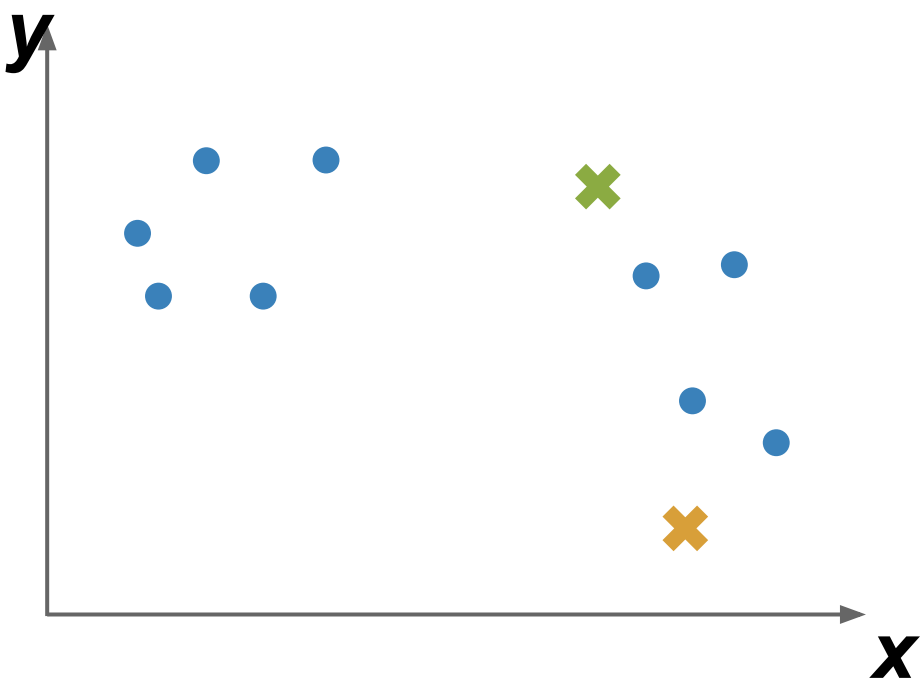
Clustering (machine learning)



Types of clustering

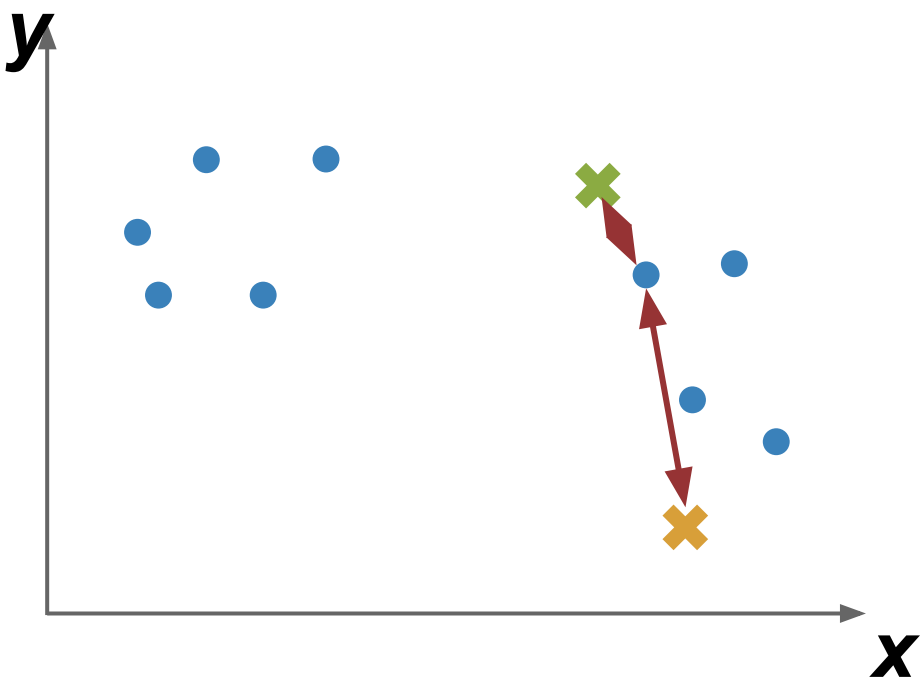
- Hierarchical clustering
- Density clustering (e.g. DBSCAN)
- Centroid clustering (e.g. k-means)

K-means



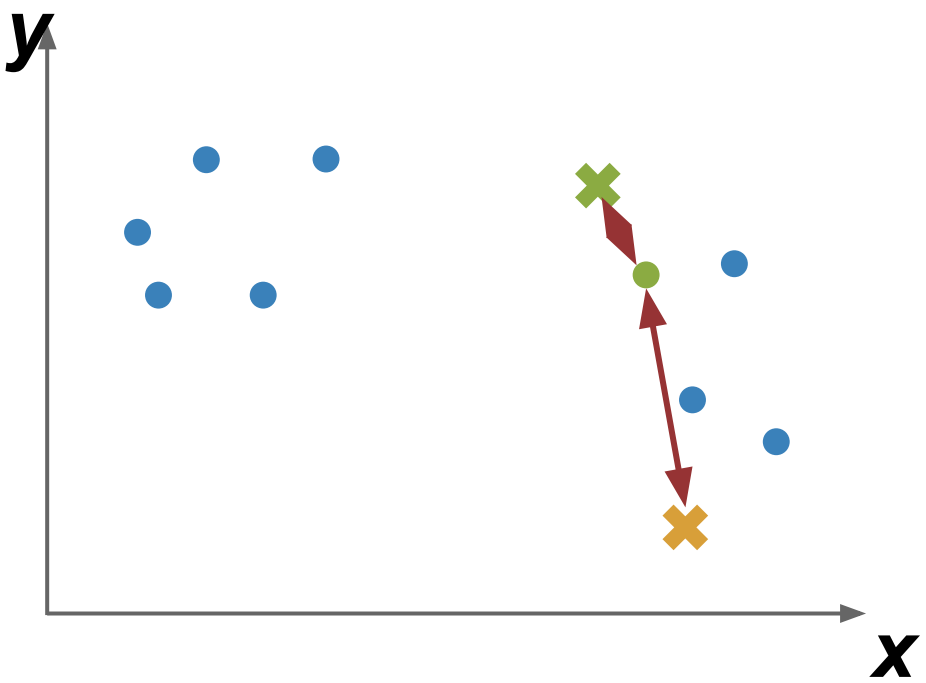

```
data = spark.textFile("hdfs://...")
            .map(parsePoint)
centroids = Array(
             Point(randX(), randY()),
             Point(randX(), randY()))
```

K-means - process the distances for each prototypes



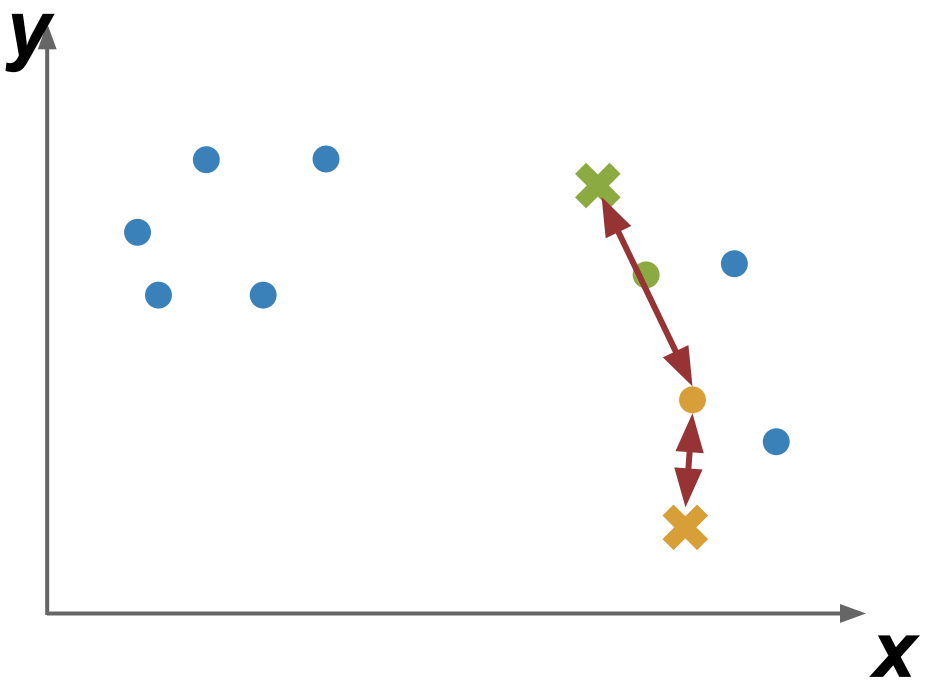

```
closestCentroid(p, centroids)
```

K-means - affectations



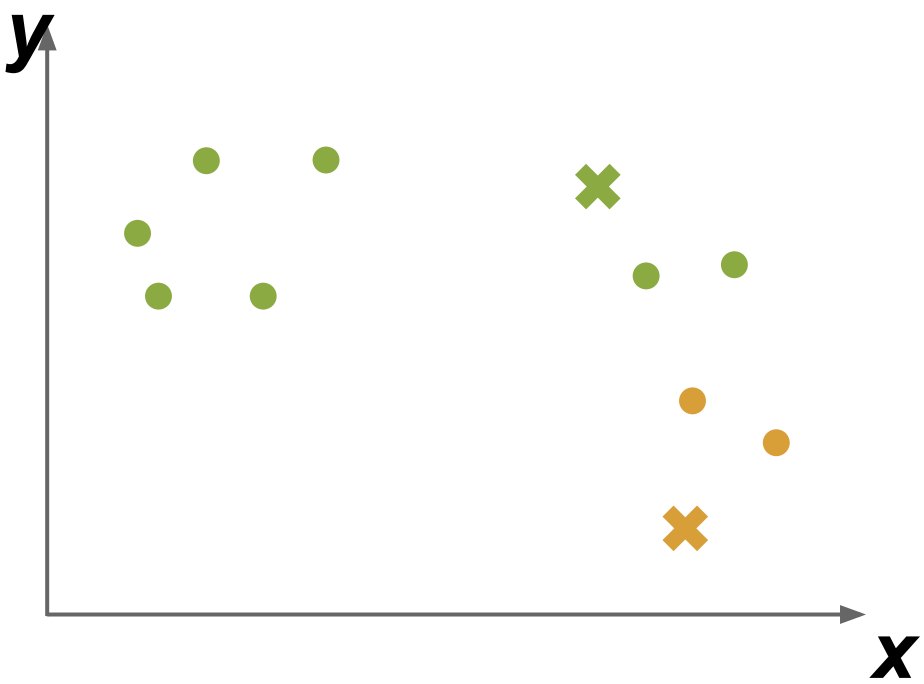

```
closestCentroid(p, centroids)
```

K-means - affectations



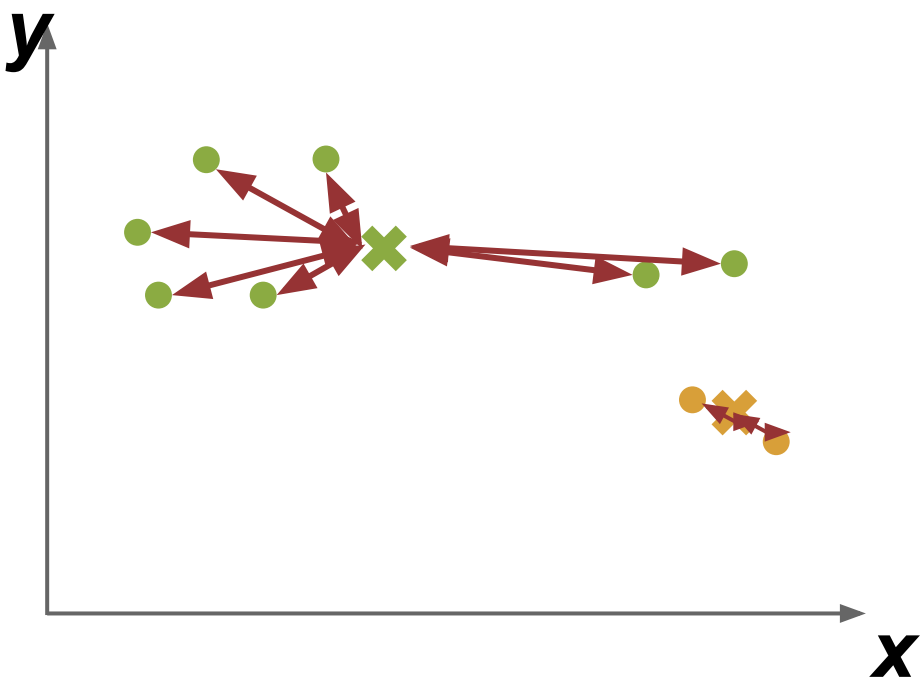

```
closestCentroid(p, centroids)
```

K-means - Map(affecations)



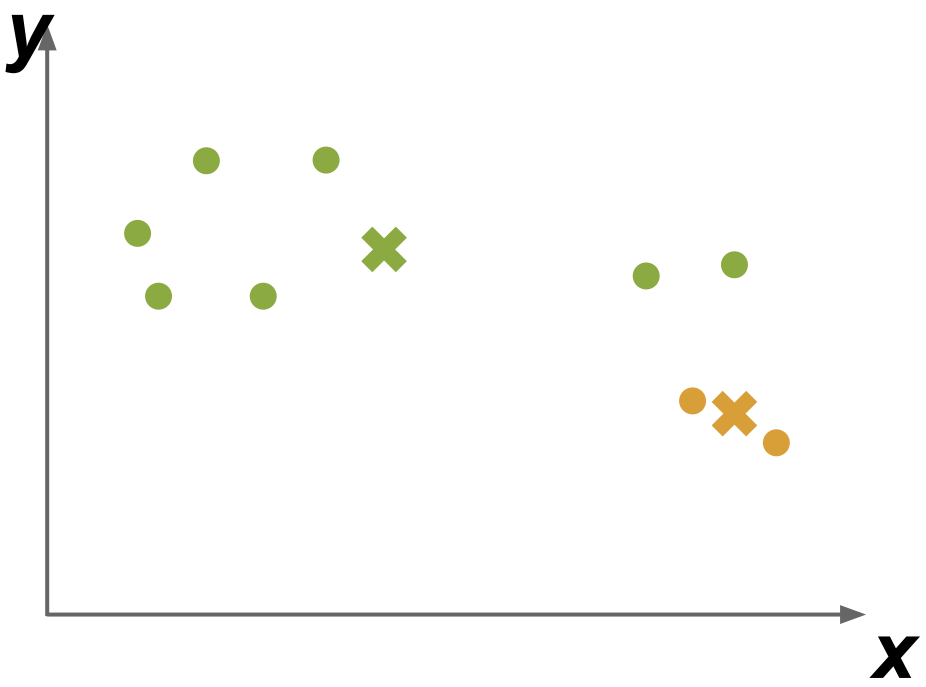

```
  val closest = data.map(p =>
    (closestCentroid(p, centroids),
(p, 1))
  )
```

K-means - Reduce(update prototypes)



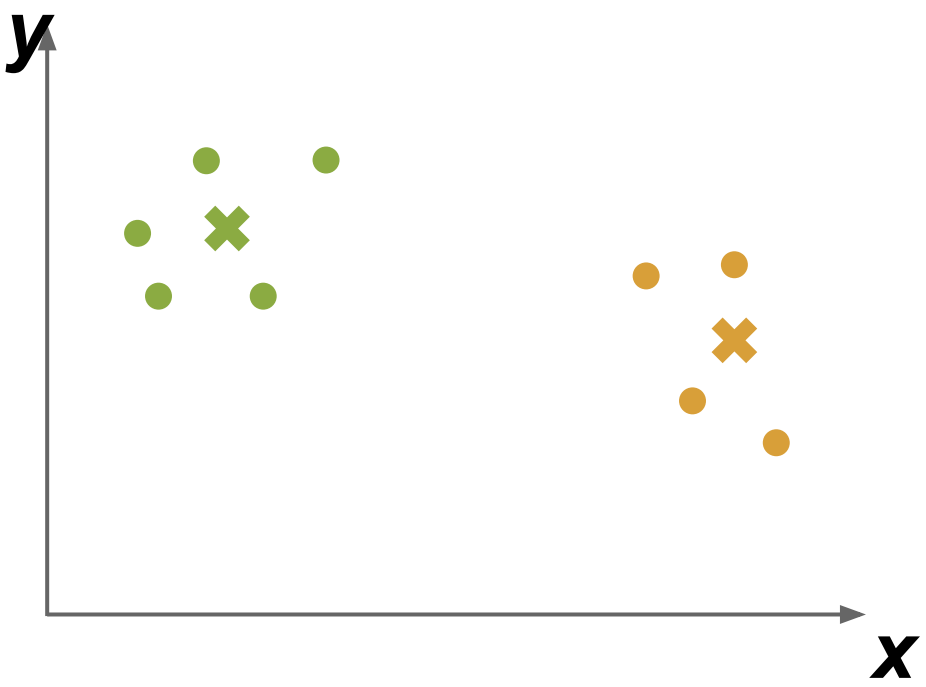

```
  val pointStats=closest.reduceByKey{
    case ((p1, sum1), (p2, sum2)) =>
(p1 + p2, sum1 + sum2)
  }
```

K-means - Iteration 1



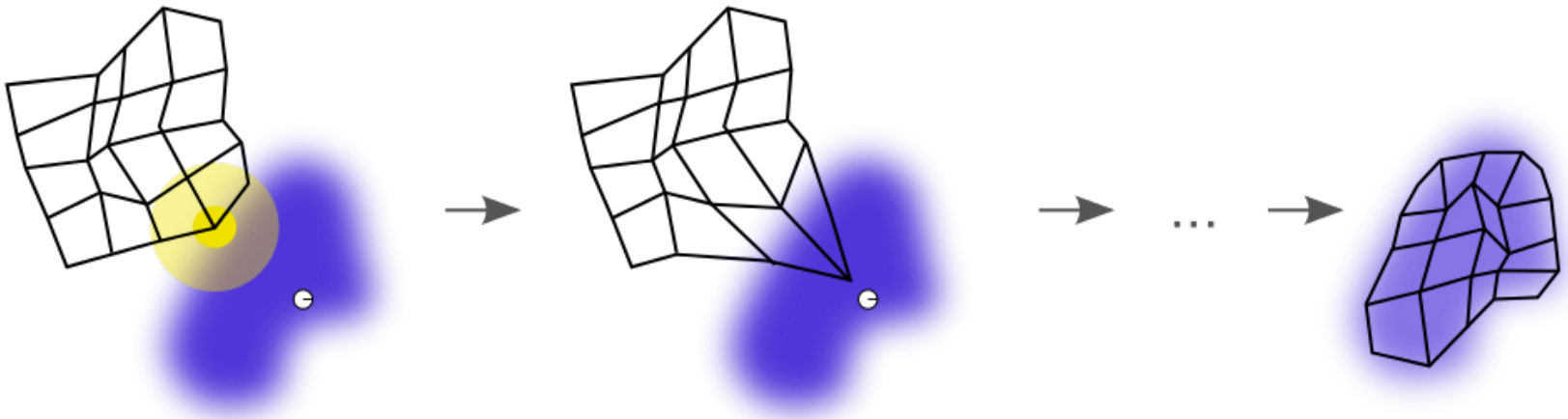

```
  val pointStats=closest.reduceByKey{
    case ((p1, sum1), (p2, sum2)) =>
(p1 + p2, sum1 + sum2)
  }
  pointStats.foreach{case(id, value)
=>
  centroids(id) = value._1 / value._2
  }
```

K-means - Iteration 2



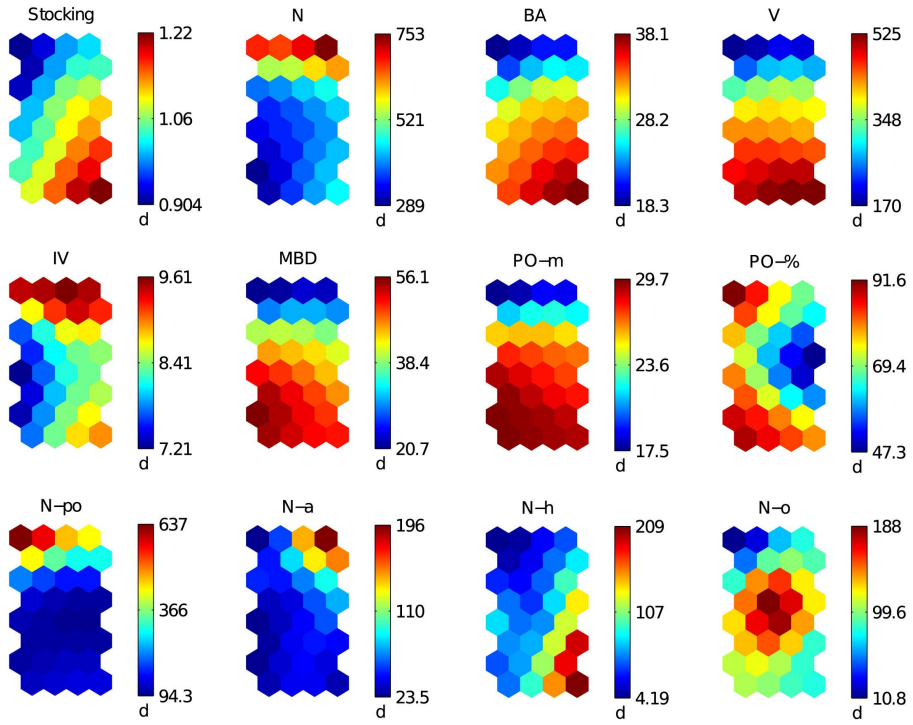

```
for (i <- 1 until 10) {
  val closest = data.map(p =>
    (closestCentroid(p, centroids),
(p, 1))
  )
  val pointStats=closest.reduceByKey{
    case ((p1, sum1), (p2, sum2)) =>
(p1 + p2, sum1 + sum2)
  }
  pointStats.foreach{case(id, value)
=>
  centroids(id) = value._1 / value._2
  }
}
```

Self-Organizing Map

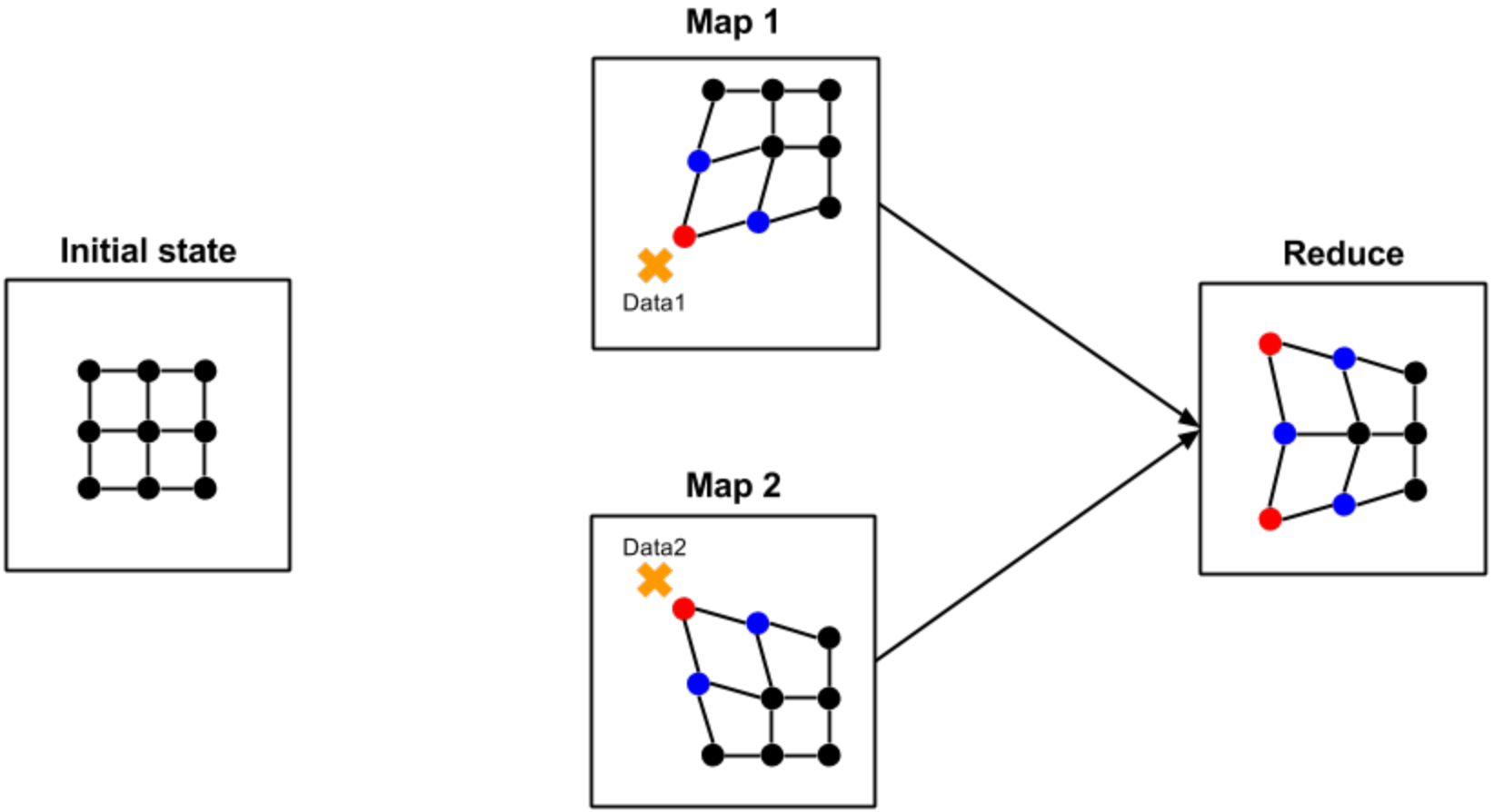


SOM benefits

- Better clustering performances than k-means
- Linear complexity
- Topological visualization

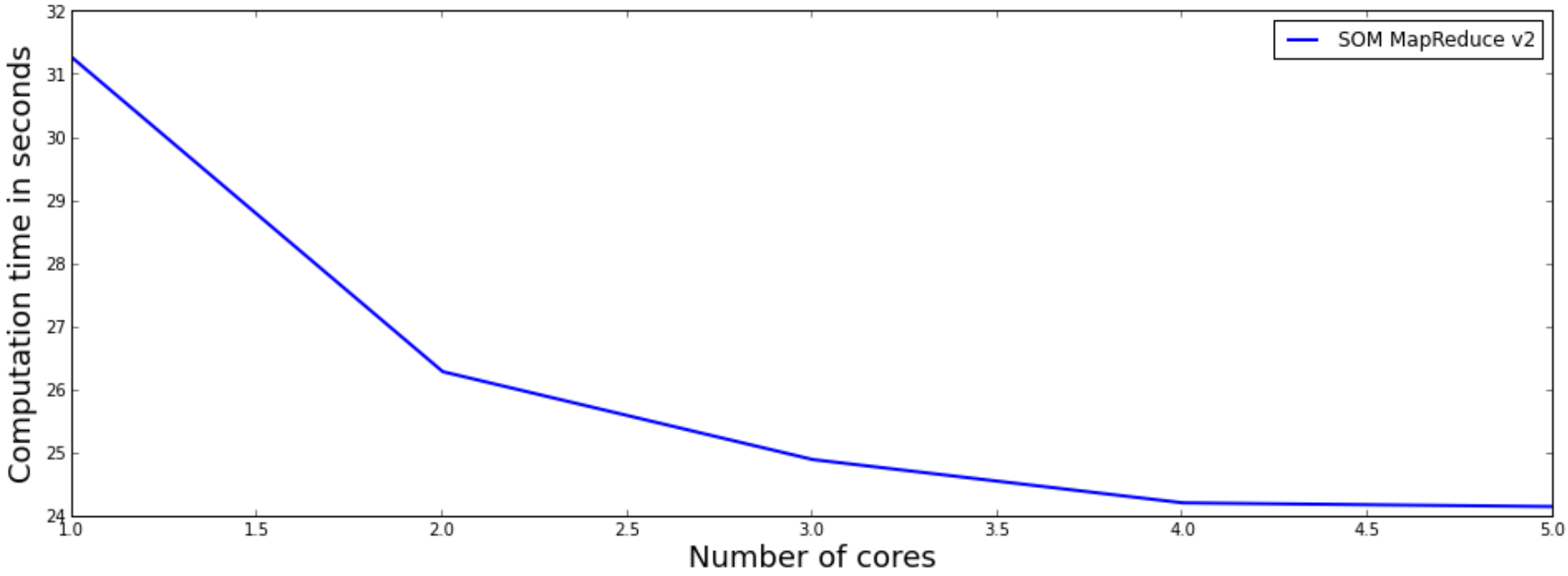


SOM MapReduce - iteration



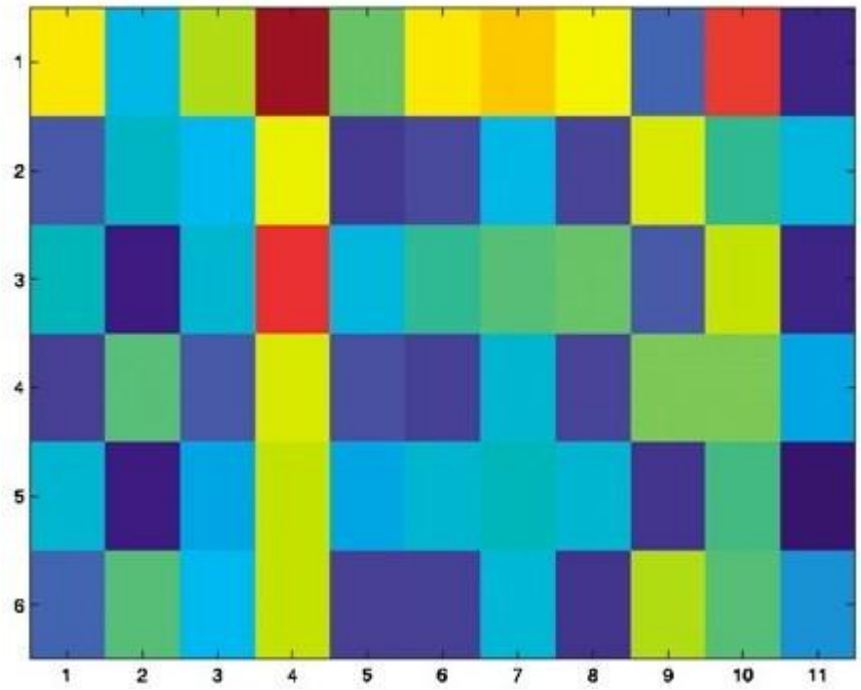
Speed up

- 100 millions observations
- SOM - Map : 10 x 10

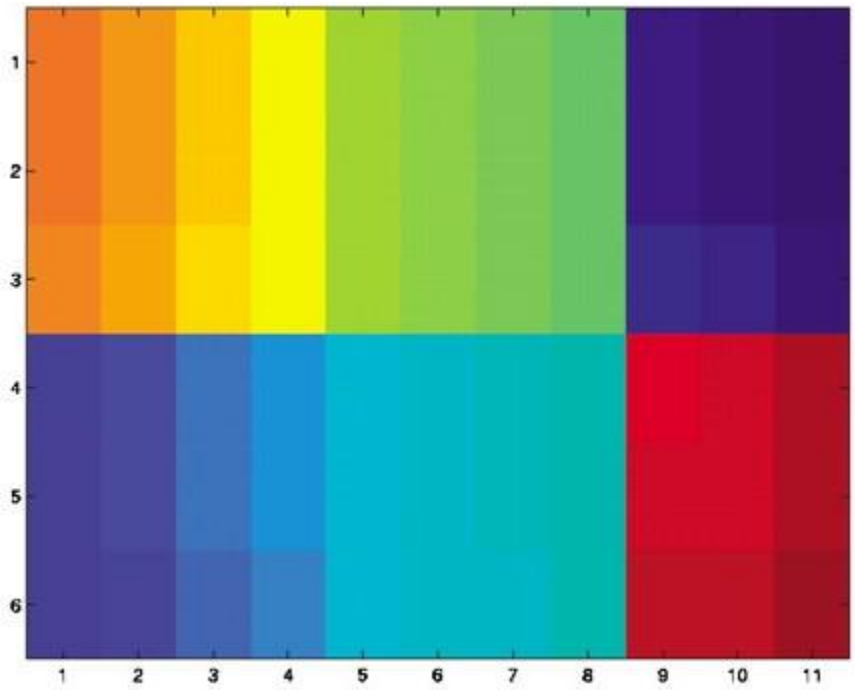


Bitm : Biclustering Topological Map

Raw Data

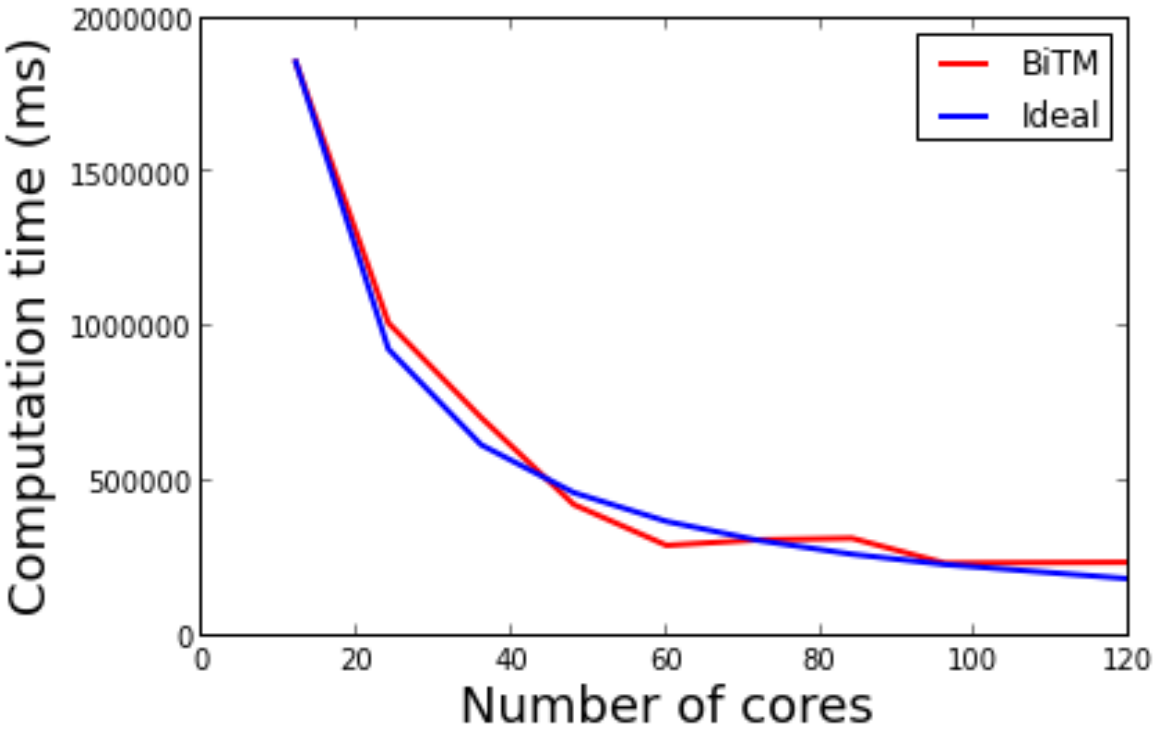


BiClustering



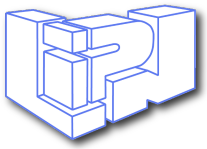
Bitm : speedup

- 2 millions observations
- SOM - Map : 5 x 5



Evolutions

- Improve performances
- Feature selection
- Increase the number of cores
  - Grid 5000
  - Google Compute engine



Thank you!

Questions?