# Modèle de Biclustering dans un paradigme "Mapreduce"

Tugdual Sarazin, Mustapha Lebbah, Hanene Azzag

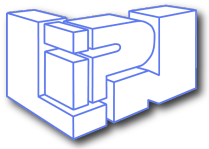
**EGC2015, Atelier CluCo**
**Luxembourg le 27 janvier 2015**

tugdual@smoke-watchers.com
@TugdualSarazin

SmokeWatchers

# Outline

- Context
- MapReduce and Spark
- SOM clustering with MapReduce
- Topological Biclustering with MapReduce
- Conclusion

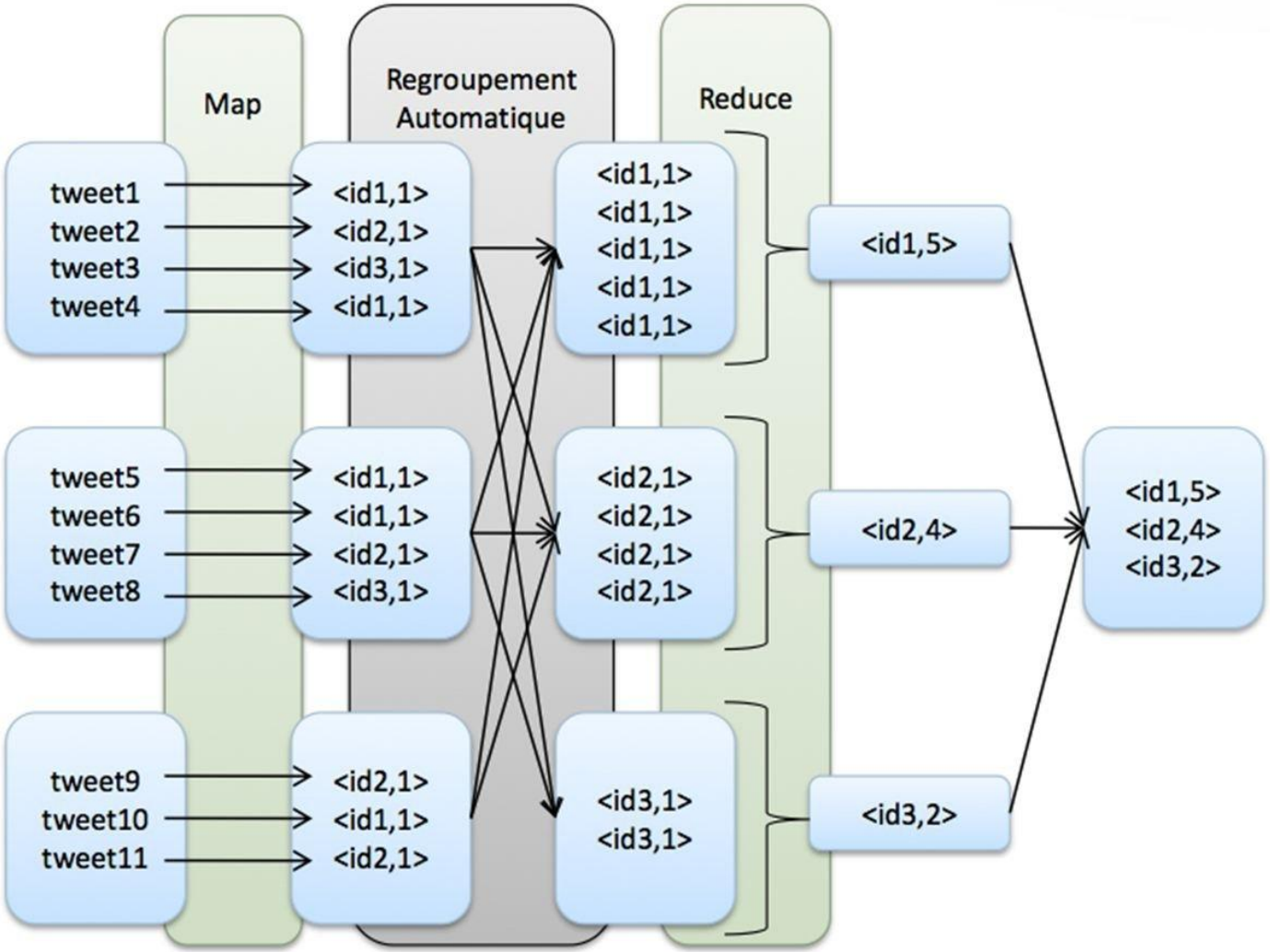
# SmokeWatchers

## SmokeWatchers Data driven


## SmokeWatchers Social driven

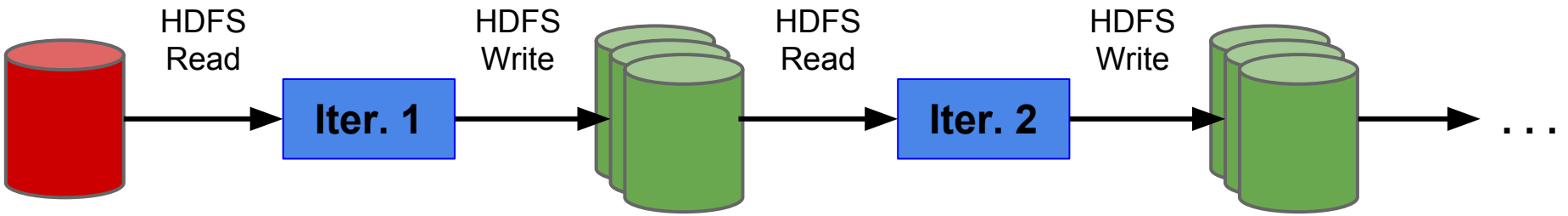
SmokeWatchers

# MapReduce

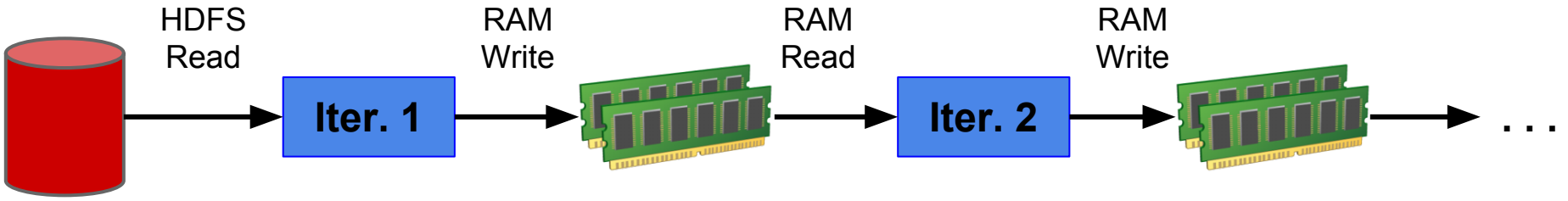


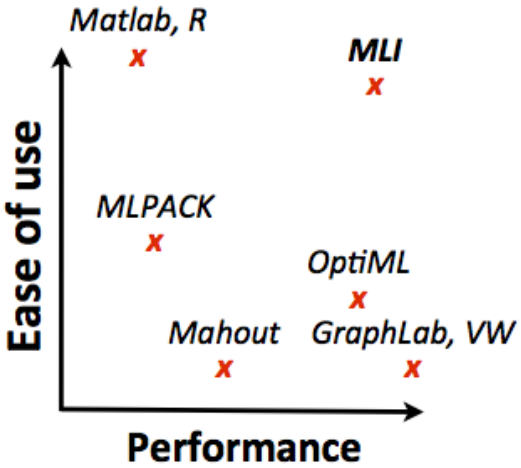

SmokeWatchers

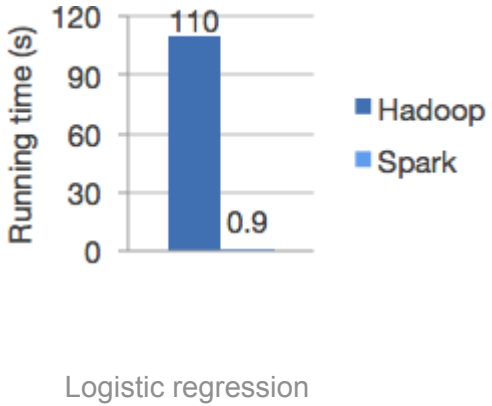
# Machine learning with Hadoop MapReduce



# Machine learning with Spark

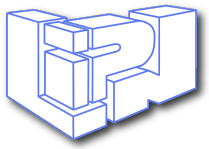


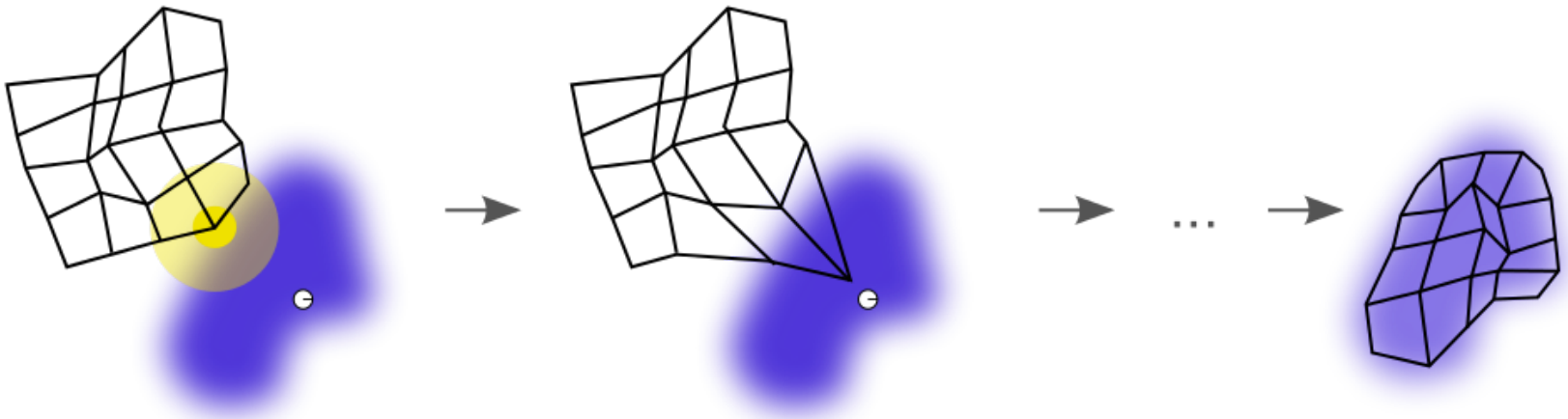




*[Sparks et al ICDM 2013]*

# SOM -MapReduce

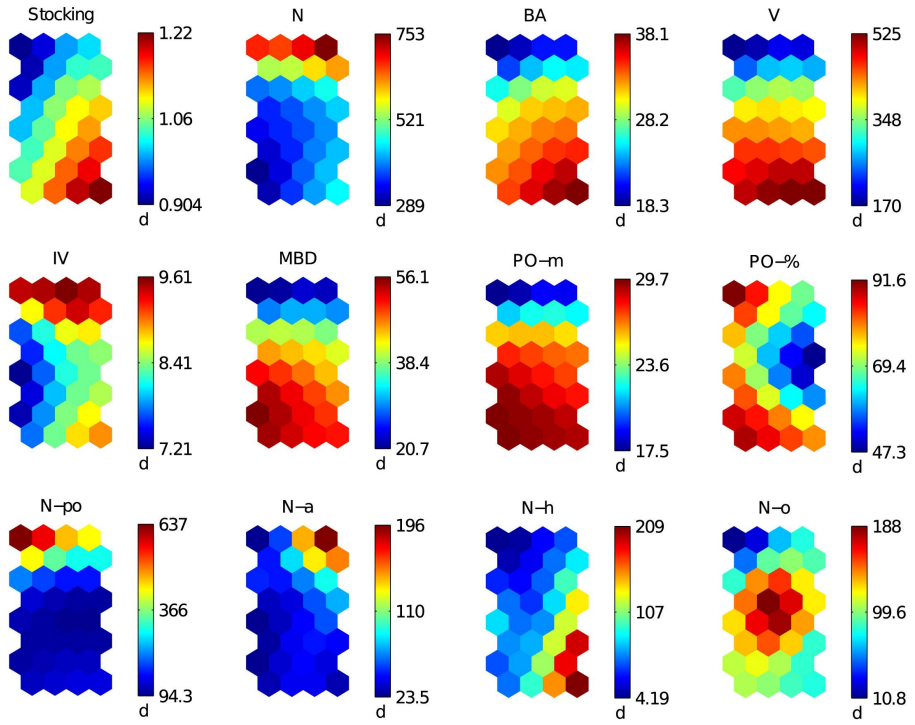



# Self-Organizing Map



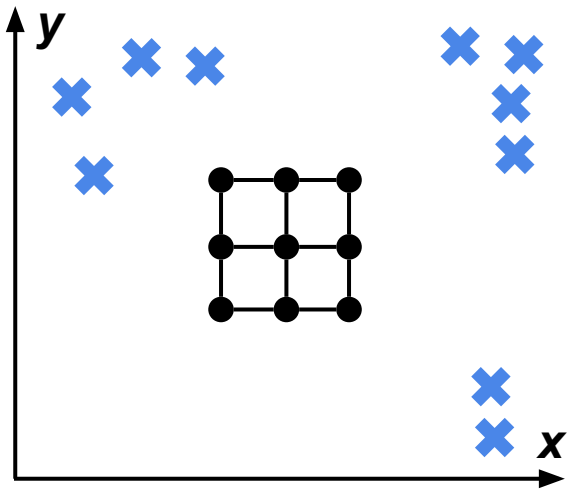
# SOM benefits

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




SmokeWatchers

# SOM MR
## Initialization

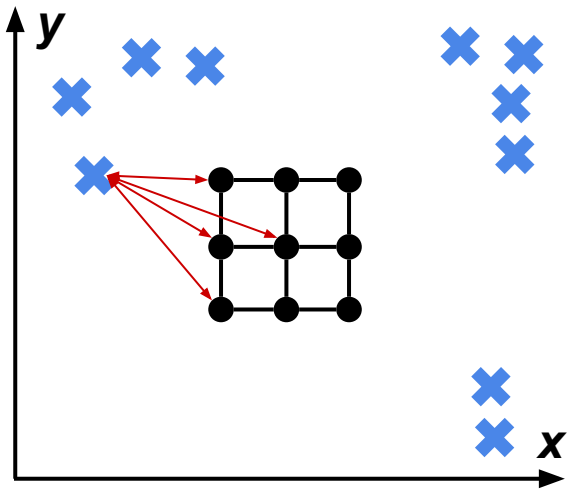



```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)
```

SmokeWatchers

# SOM MR

## Map: distance data 1

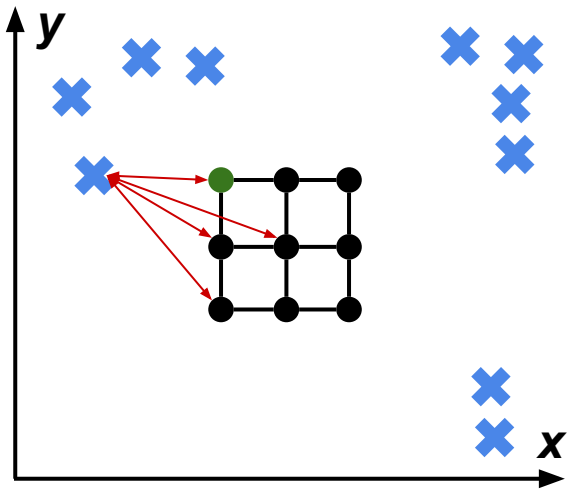



```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)
```

SmokeWatchers

# SOM MR

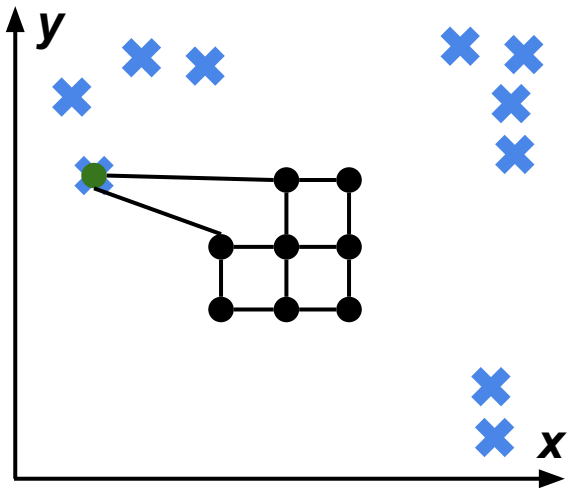
## Map: select closest prototype




```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

    val closestP = closestPrototype(prototypes, d)
```

# SOM MR

## Map: process new local prototypes

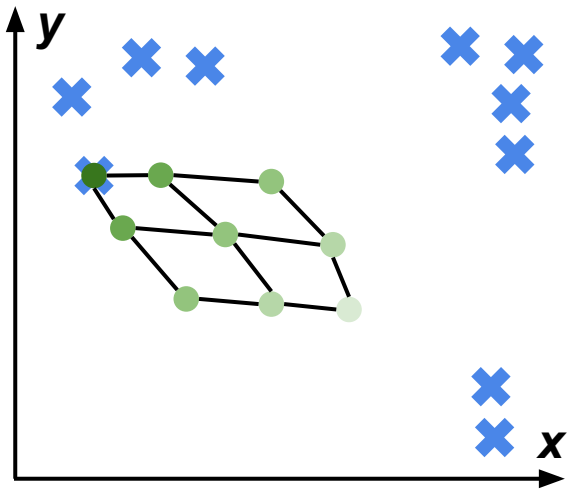



```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

    val closestP = closestPrototype(prototypes, d)
```

SmokeWatchers

# SOM MR

## Map : process new local prototypes

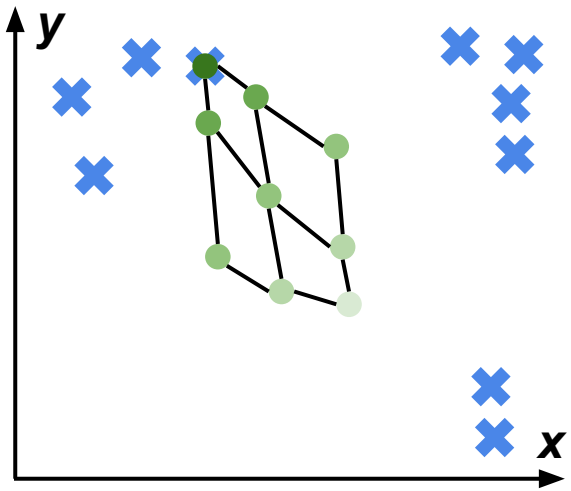



```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
```

SmokeWatchers

# SOM MR

## Map : process new local prototypes

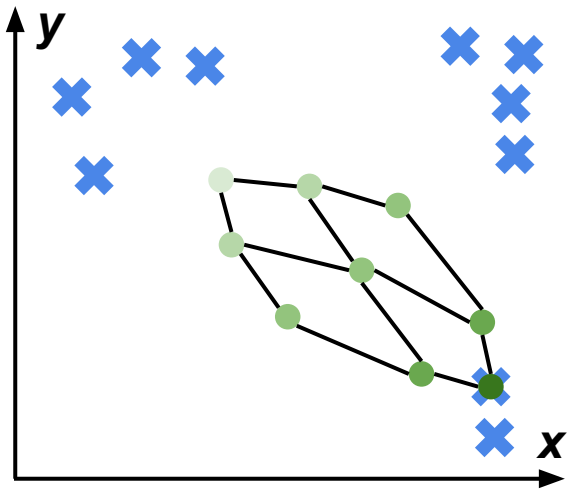



```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

  prototypes = data.map{d =>
    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
  }
```

SmokeWatchers

# SOM MR

## Map : process new local prototypes



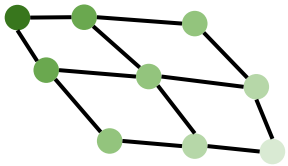

```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

  prototypes = data.map{d =>
    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
  }
```

SmokeWatchers

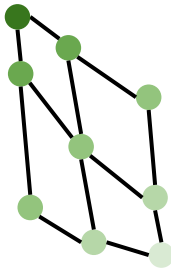
# SOM MR

## Maps outputs

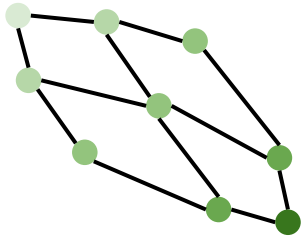
**Map 1**



**Map 2**



**Map 3**



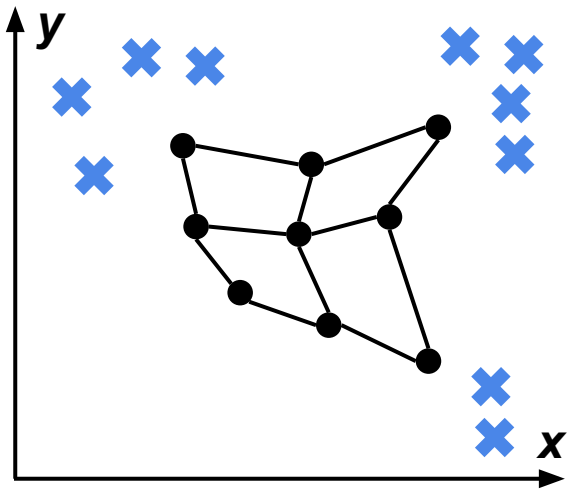
.....

```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

  prototypes = data.map{d =>
    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
  }
```

SmokeWatchers

# SOM MR

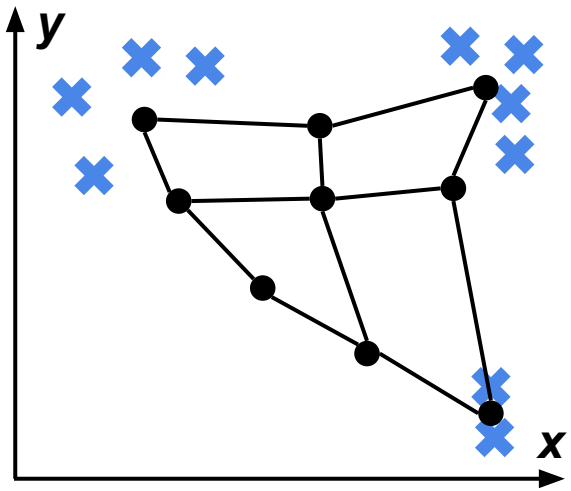
## Reduce: Merge locals models




```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

  prototypes = data.map{d =>
    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
  }.reduce{ (a, b) =>
    a.zip(b).map{case ((aV, aT),(bV, bT)) =>
      (aV+bV, aT+bT)
    }
  }.map(r => r._1 / r._2)
```

SmokeWatchers

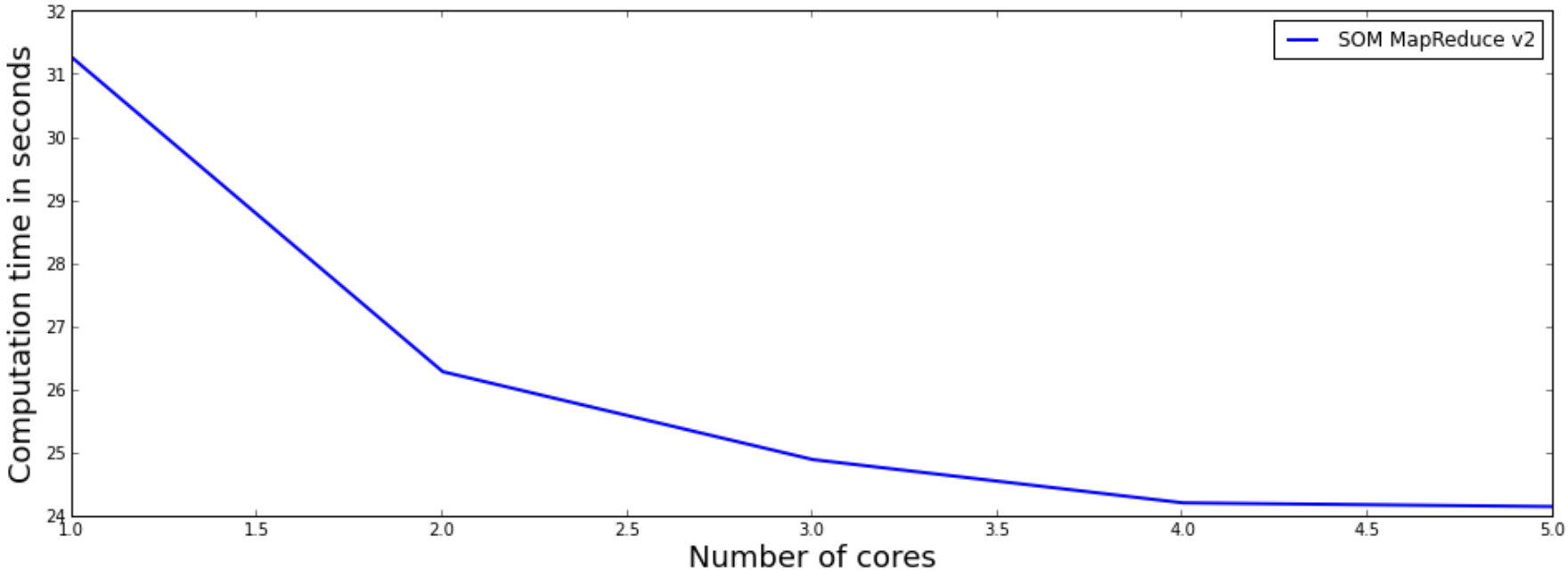
# SOM MR
## 10 iterations




```
var prototypes = Array.fill(9)(
  Vector(Array.fill(2)(Random.nextDouble))
)

for (i <- 1 until 10) {
  prototypes = data.map{d =>
    val closestP = closestPrototype(prototypes, d)
    prototypes.map{ p =>
      val t = T(p, closestP)
      (p*t, t)
    }
  }.reduce{ (a, b) =>
    a.zip(b).map{case ((aV, aT),(bV, bT)) =>
      (aV+bV, aT+bT)
    }
  }.map(r => r._1 / r._2)
}
```

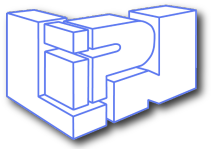
# Speed up

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



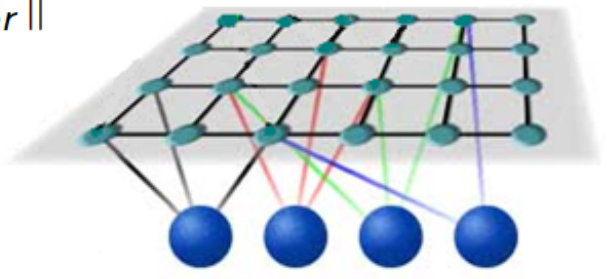

SmokeWatchers

# MapReduce Topological BiClustering

# Topological Bi-clustering

$$\tilde{R}(\phi_w, \phi_z, G) = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{\mathbf{x}_i \in P_k} \sum_{\mathbf{x}^j \in Q_l} \sum_{r=1}^{K} \mathcal{K}^T(\delta(r,k)) \parallel x_i^j - g_r^l \parallel^2$$

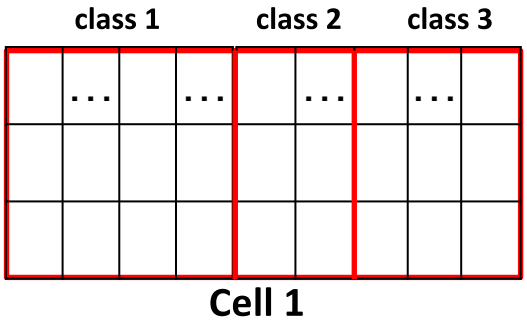


$\mathbf{g}_k = (g_k^1, g_k^2 ..., g_k^l, ..., g_k^L)$ de dimension $L < d$ où $g_k^l$ est le prototype du bloc $B_k^l$.

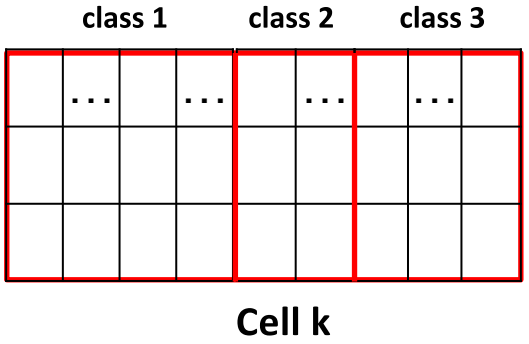
$g_1 = (g^1_{1,} g^2_{1,} g^3_{1})$

.....

$g_k = (g^1_{k,} g^2_{k,} g^3_{k})$




.....




Same cluster of variables on all cell of the map

## 1) Observation assignment

$$x_i^j \ (x^1,\ x^2,\ x^3,\ x^4,\ x^5,\ x^6,\ x^7,\ x^8,\ x^9)$$

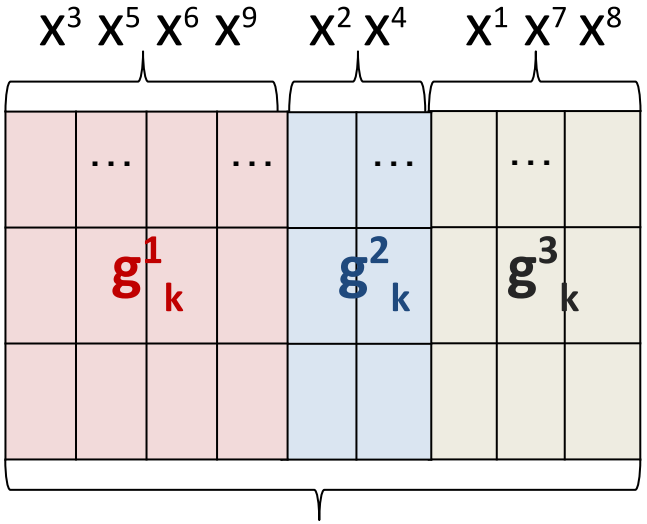
.....

$$g_k \ (g^3{}_k, g^2{}_k, g^1{}_k, g^3{}_k, g^1{}_k, g^1{}_k, g^2{}_k, g^3{}_k, g^1{}_k)$$

## 2) variable assignment

$$\begin{pmatrix} x^j{}_1 \\ x^j{}_2 \\ \dots \\ x^j{}_i \\ \dots \\ x^j{}_n \end{pmatrix} \quad \begin{pmatrix} g^1{}_{\varphi(x1)} \\ g^1{}_{\varphi(x2)} \\ \dots \\ g^1{}_{\varphi(xi)} \\ \dots \\ g^1{}_{\varphi(xn)} \end{pmatrix} \quad \begin{pmatrix} g^2{}_{\varphi(x1)} \\ g^2{}_{\varphi(x2)} \\ \dots \\ g^2{}_{\varphi(xi)} \\ \dots \\ g^2{}_{\varphi(xn)} \end{pmatrix} \quad \begin{pmatrix} g^3{}_{\varphi(x1)} \\ g^3{}_{\varphi(x2)} \\ \dots \\ g^3{}_{\varphi(xi)} \\ \dots \\ g^3{}_{\varphi(xn)} \end{pmatrix}$$

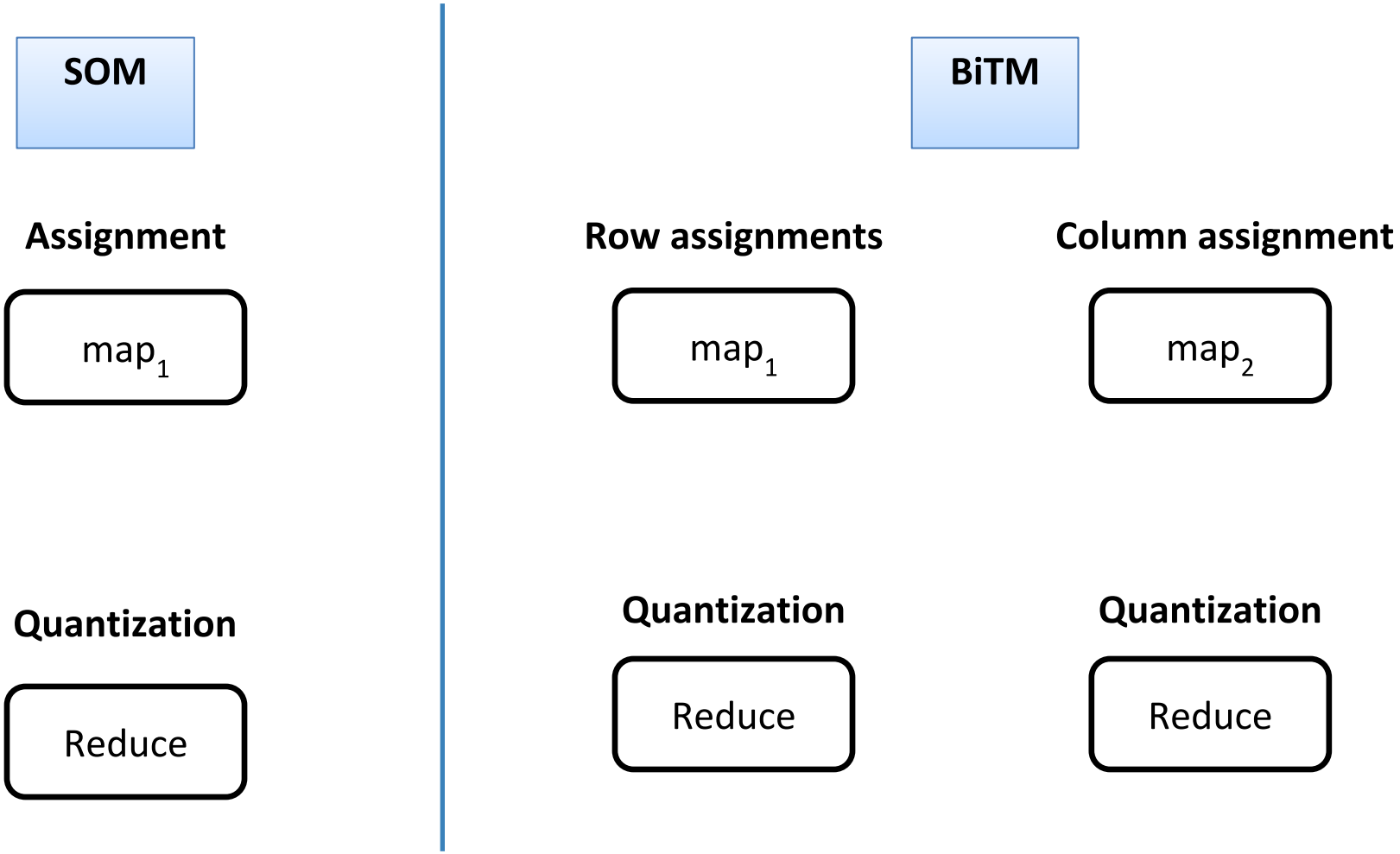
**A cell *k of the map***




$$g_k = (g^1{}_k, g^2{}_k, g^3{}_k)$$

## 3) Quantization

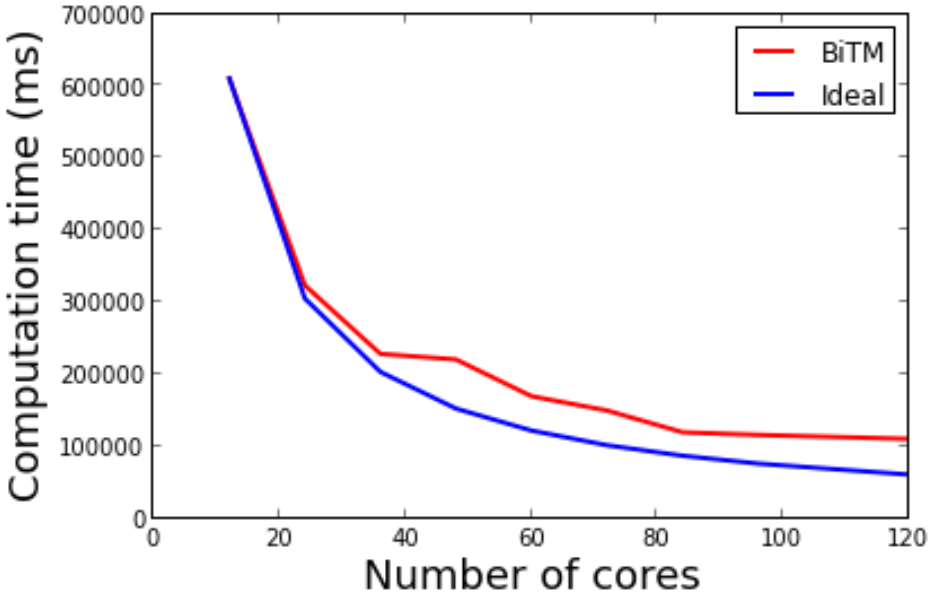
# MapReduce / Spark




*https://github.com/TugdualSarazin/spark-clustering*

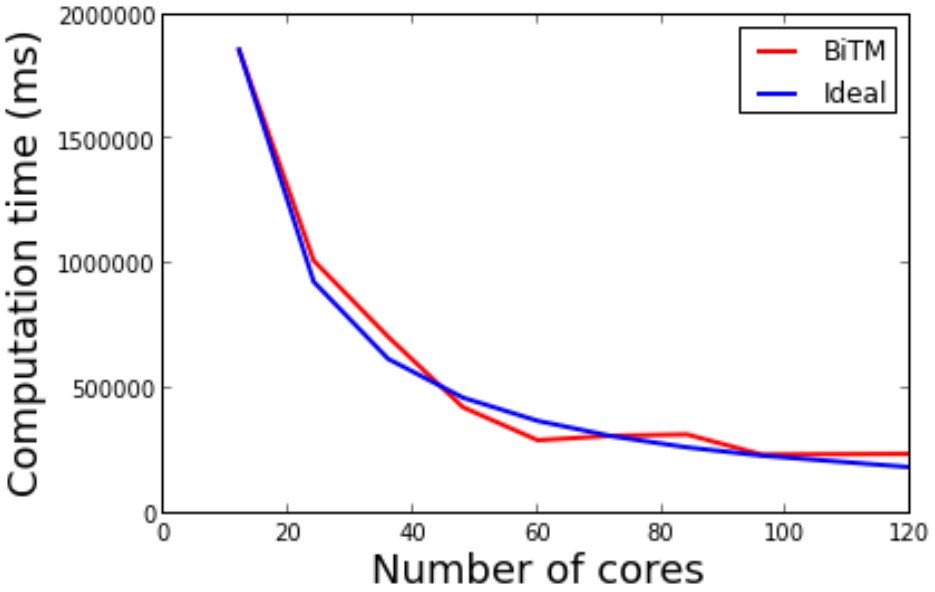
SmokeWatchers

# BiTM-MapReduce-SPARK




**2 millions, 20 variables**

**2 millions, 40 variables**



# Conclusions & perspectives

- Data set size has increased significantly
  - MapReduce is crucial for some algorithms
  - Deal with large data

- New approach
  - Deal with mixed data
  - Clustering data stream using batch paradigm

SmokeWatchers

# Thank you!

Questions?