# INFO-H-515:
# PART BIG DATA ANALYTICS

## Map-reduce analytics

Gianluca Bontempi

Machine Learning Group
Boulevard de Triomphe - CP 212
http://mlg.ulb.ac.be

# COARSE-GRAINED PARALLELISM

We will discuss how to distribute machine learning computing in the following setting

- Training dataset that does not fit in the main memory of a single machine
- Computing cluster infrastructure, i.e. a large collection of commodity hardware connected by Ethernet or inexpensive switches
- Distributed file system (e.g. HDFS), providing replication and redundancy
- Map Reduce programming model partitioning the computation in tasks (that can be restarted without affecting the others) allowing then scalability and robustness.
- Map = Scatter pieces of a problem across hosts
- Reduce= Gather/Aggregate partial solution into a final result
- Increasing performance is not necessarily the main goal. Fault tolerance and scalability (avoid bottlenecks) are more important ones!

# FROM THE CREATOR OF MAP-REDUCE

Part of the reason we didn't develop MapReduce earlier was probably because when we were operating at a smaller scale, then our computations were using fewer machines, and therefore robustness wasn't quite such a big deal: it was fine to periodically checkpoint some computations and just restart the whole computation from a checkpoint if a machine died. Once you reach a certain scale, though, that becomes fairly untenable since you'd always be restarting things and never make any forward progress. (Jeff Dean)

# MAP-REDUCE AND SPARK

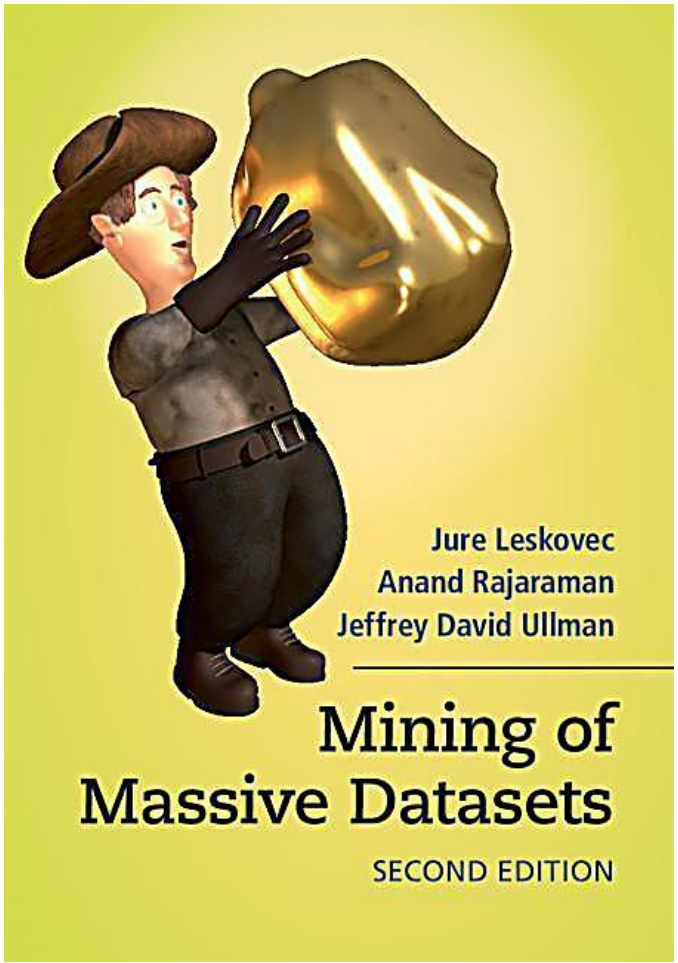
Spark extends the original Hadoop's MR in several ways

1. From a rigid map-the-reduce format to a more general directed acyclic graph of operators (DAGs won't be executed until an action is triggered). No need to write intermediate results on the distributed filesystem
2. Richer set of transformations (numerical, string, time)
3. Extension with in-memory processing
4. Large ecosystem (Cassandra, Kafka) and many APIs (Pyspark)
5. Good tradeoff between exploratory analytics (machine learning) and operational analytics (deployment, production)

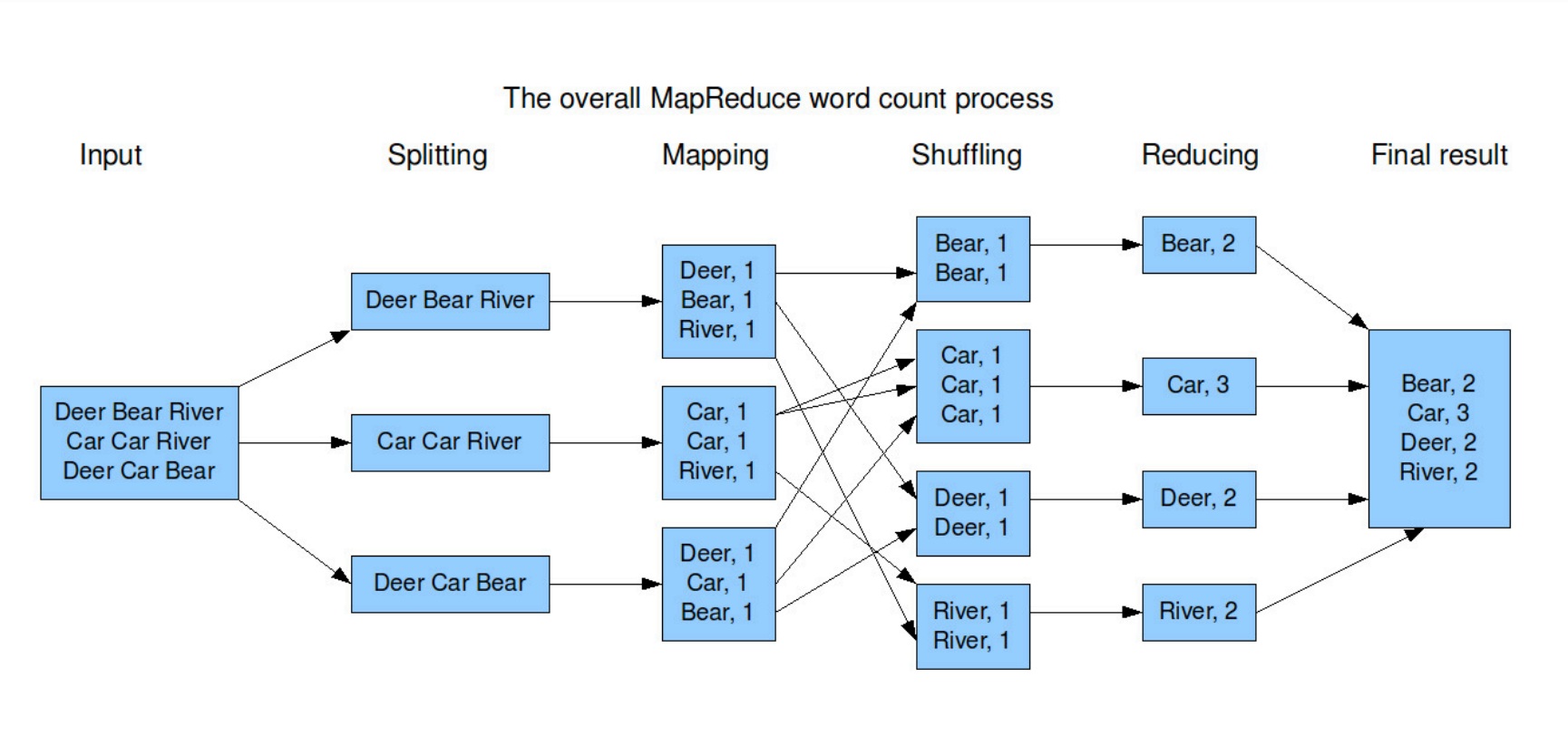
Spark code of the MR examples in
`https://github.com/gbonte/gbcodepyspark/`

# DISCLAIMER

Much of the following material is taken from the book [6]

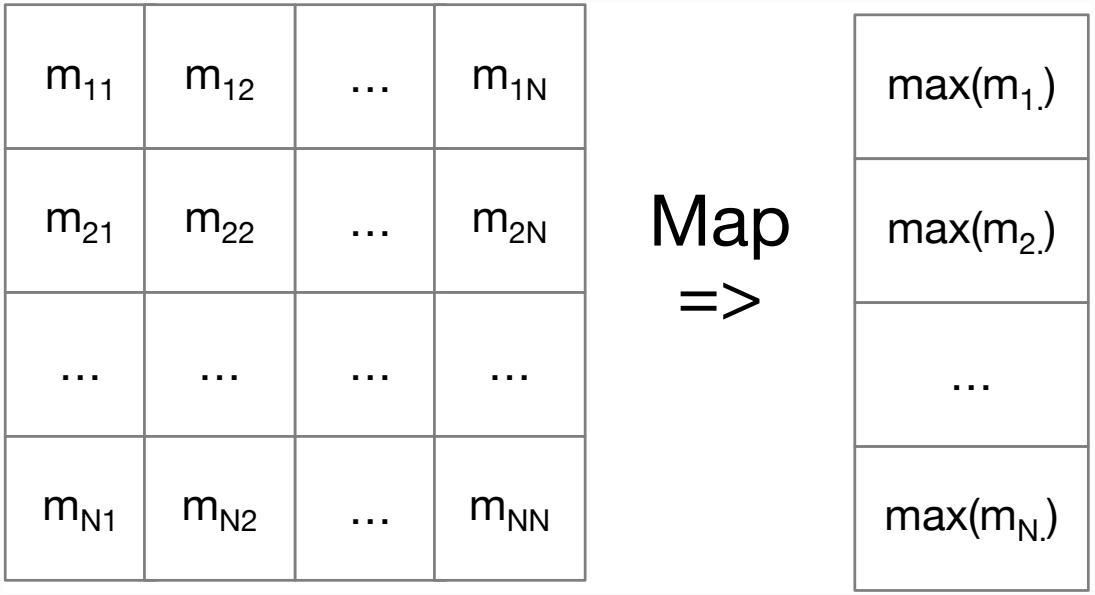


# MAP-REDUCE EXAMPLES




- M: operates on a list of values in order to produce a new list of values, by applying the same computation to each value.
- R: operates on a list of values to collapse or combine those values into a single value (or some number of values), again by applying the same computation to each value.

# ROW-WISE MAXIMUM




Very simple example (embarassingly parallel). No Reduce step necessary. All the processing is done in a row-wise manner by the map function. Note that the first row of the output is dependent only on the first row of the input.

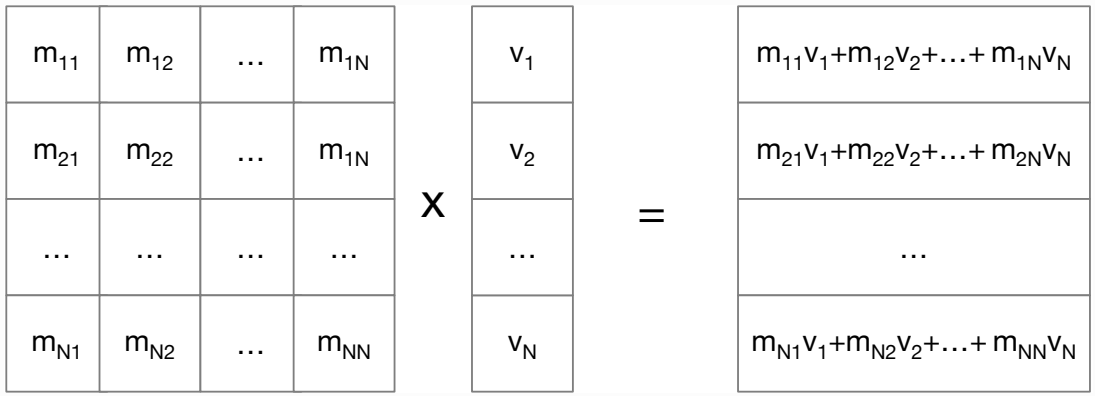
# MATRIX-VECTOR MULTIPLICATION BY MR

Let us consider a square $N \times N$ matrix $M = [m_{ij}]$ and a $N \times 1$ vector v. The result is a $N \times 1$ vector whose ith element is

$$x_i = \sum_{j=1}^{N} m_{ij} v_j, \qquad i = 1, \ldots, N$$

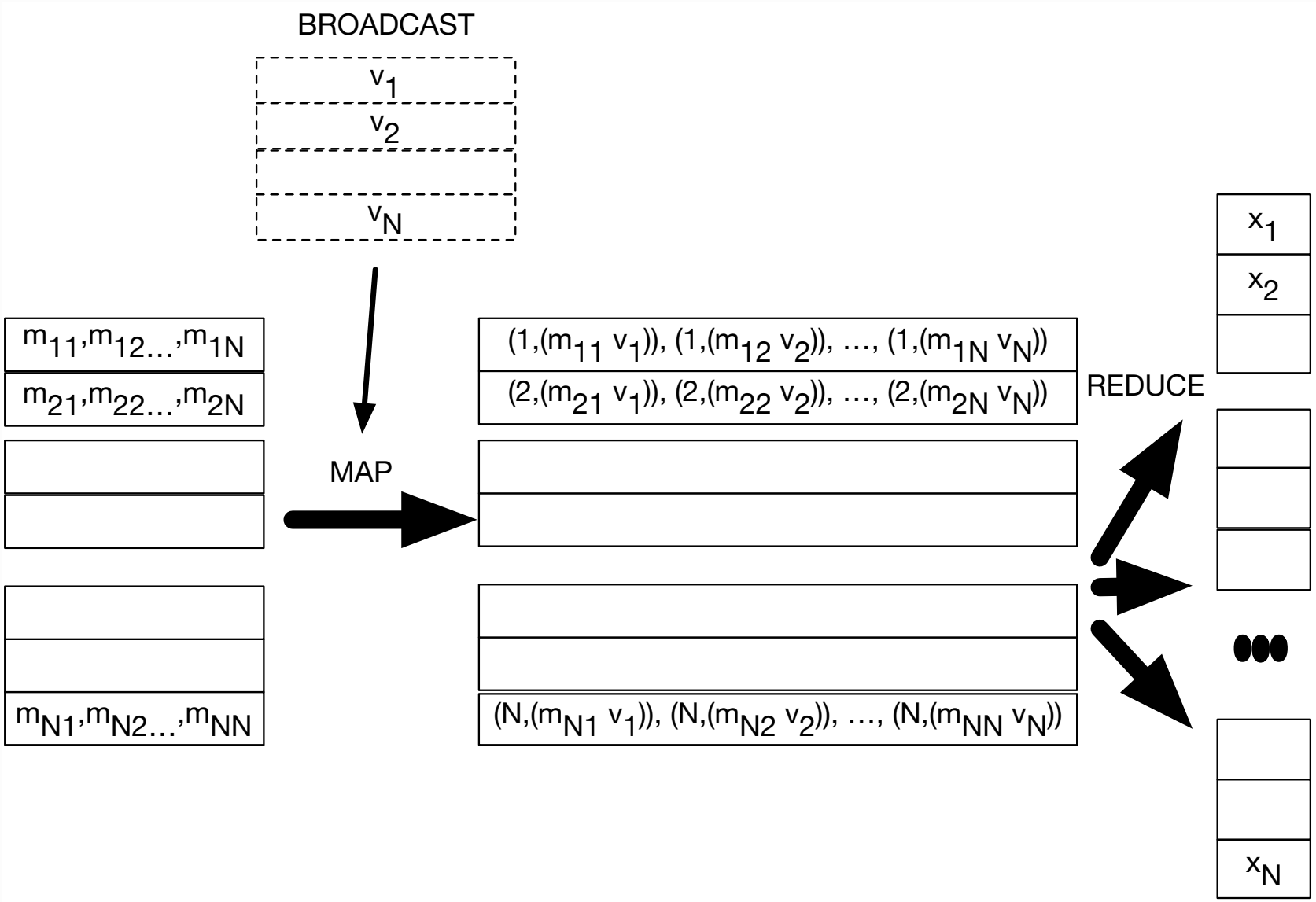
If the vector can fit in memory it can be broadcasted to each chunk and made available to all applications of the Map function

# MATRIX-VECTOR MULTIPLICATION




Is Map alone sufficient? Is the first row of the output dependent only on the first row of the inputs?

# MATRIX-VECTOR MULTIPLICATION BY MR



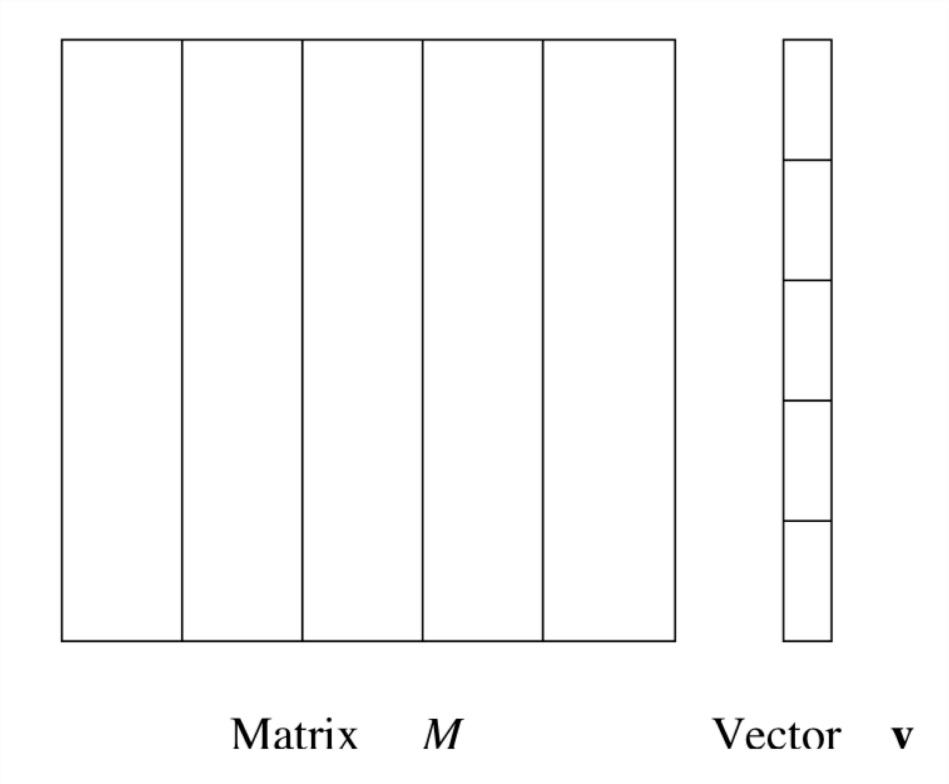
# FROM 1ST PART OF THE COURSE...

## BROADCAST VARIABLES

- Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.
- They can be used, for example, to give every node, a copy of a large input dataset, in an efficient manner.
- All broadcast variables will be kept at all the worker nodes for use in one or more Spark operations.

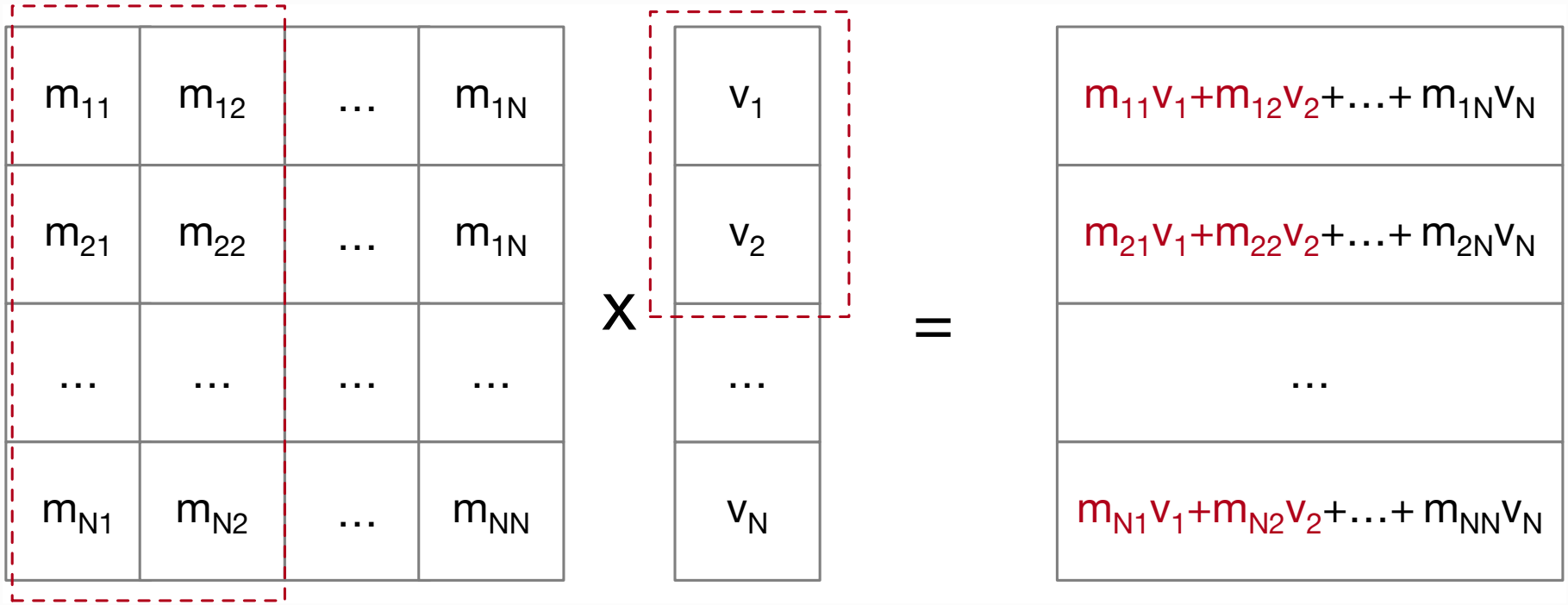
# WHAT IF N IS TOO LARGE?

Decomposition into S stripes to have the vector fitting into memory.




$$x_i = \sum_{j=1}^{N} m_{ij} v_j = \sum_{s=1}^{S} \sum_{j=s[1]}^{s[N_s]} m_{ij} v_j$$

where $s[1], \ldots, s[N_s]$ are the indices of the sth stripe.


$m_{11}$ $m_{12}$ … $m_{1N}$
$m_{21}$ $m_{22}$ … $m_{1N}$
… … … …
$m_{N1}$ $m_{N2}$ … $m_{NN}$
x
$v_1$
$v_2$
…
$v_N$
=
$m_{11}v_1+m_{12}v_2+\ldots+m_{1N}v_N$
$m_{21}v_1+m_{22}v_2+\ldots+m_{2N}v_N$
…
$m_{N1}v_1+m_{N2}v_2+\ldots+m_{NN}v_N$

# RELATIONAL DATABASES

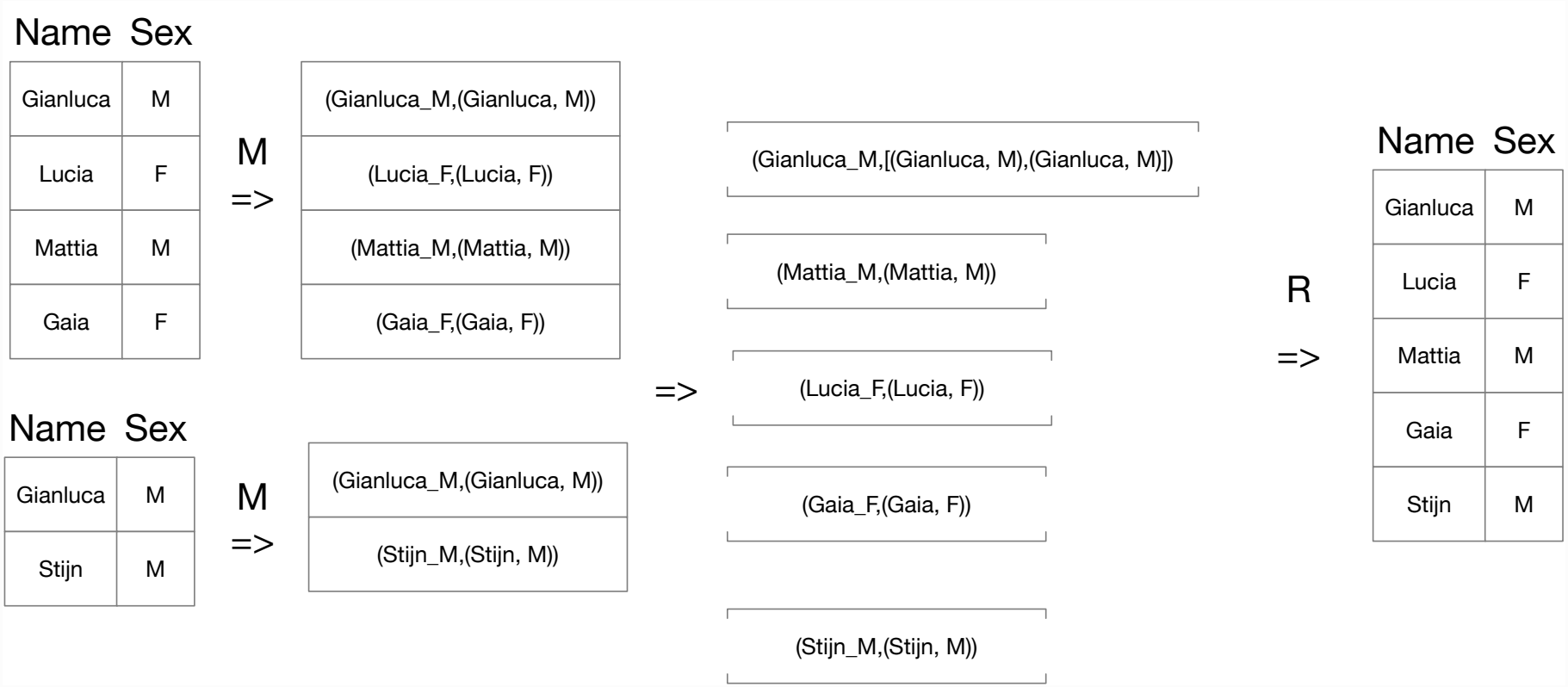
- A relation is a table whose **rows** are called **tuples** and **column** headers **attributes**. The set of attributes is called schema.
- Relational algebra operations: selection, projection, union/intersection/difference, join.
- Natural **join**: given two relations, it is the set of joint tuples such that the original ones agree on all the common attributes
- **Grouping and aggregation**: it partition tuples according to the values of the grouping attribute (e.g. sex) and computes an aggregated value (MAX, MIN, COUNT, SUM, AVG) related to an attribute (e.g. age) which is not in the grouping set. The output is a tuple per group having as grouping attribute the common value (e.g. male) and a component for each aggregation (e.g. average male ages).

# MR IMPLEMENTATION OF RELATIONAL OPERATIONS

[6] presents MR implementations of relational operations:

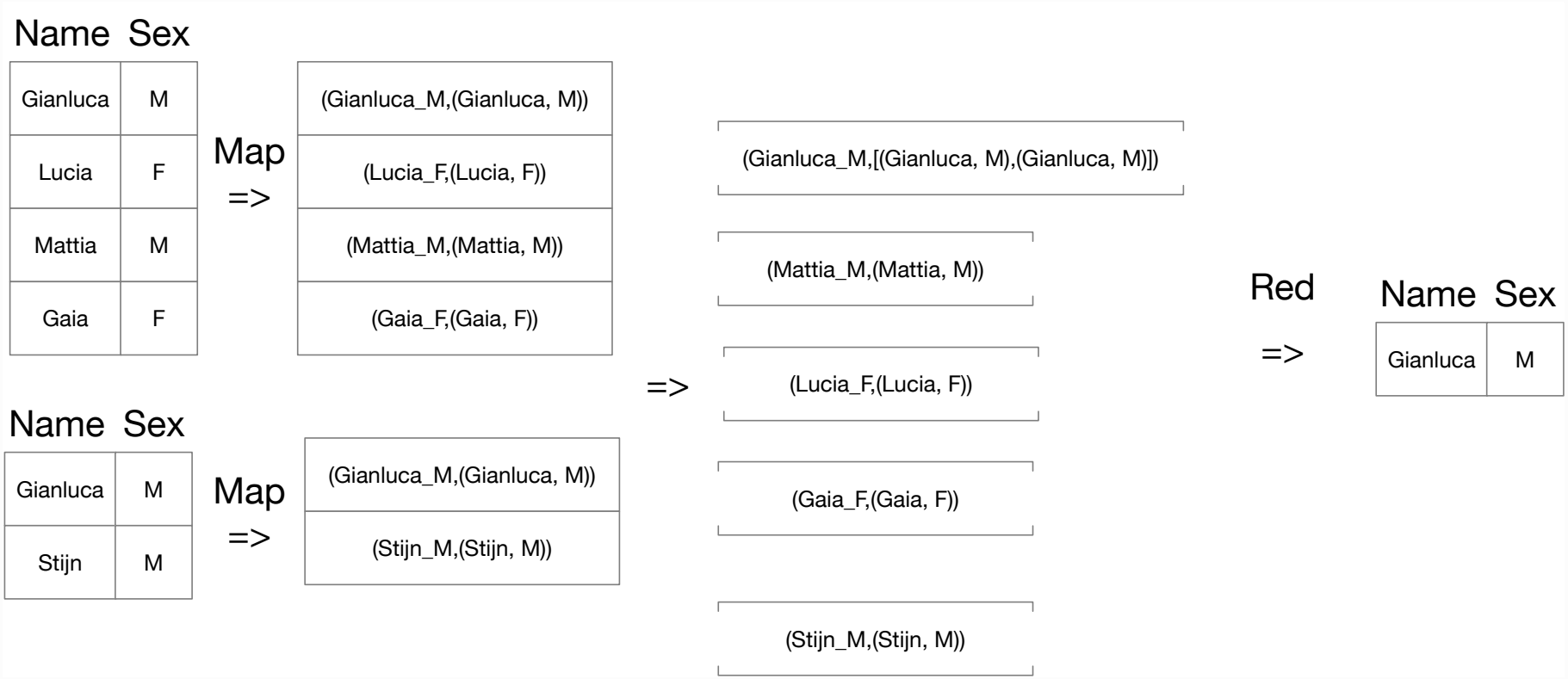
- Selection:
  - Map: returns (t, t) if the selection condition is satisfied.
  - Reduce: returns the value.
- Projection:
  - Map: returns $(t', t')$ where $t'$ is the projected subset of the tuple t.
  - Reduce: it eliminates duplicates (e.g. two different original tuples which became the same after projection).
- Union of two relations having the same schema:
  - Map: return (t, t) for each tuple t.
  - Reduce: If key t has value t or [t, t], return t.
- Intersection of two relations having the same schema:
  - Map: return (t, t) for each tuple t.
  - Reduce: If key t has value [t, t], then return t. Otherwise, return nothing.

# MR EXAMPLE: UNION OF TWO TABLES




Union R(Name,Sex) and S(Name,Sex)

# MR EXAMPLE: INTERSECTION OF TWO TABLES

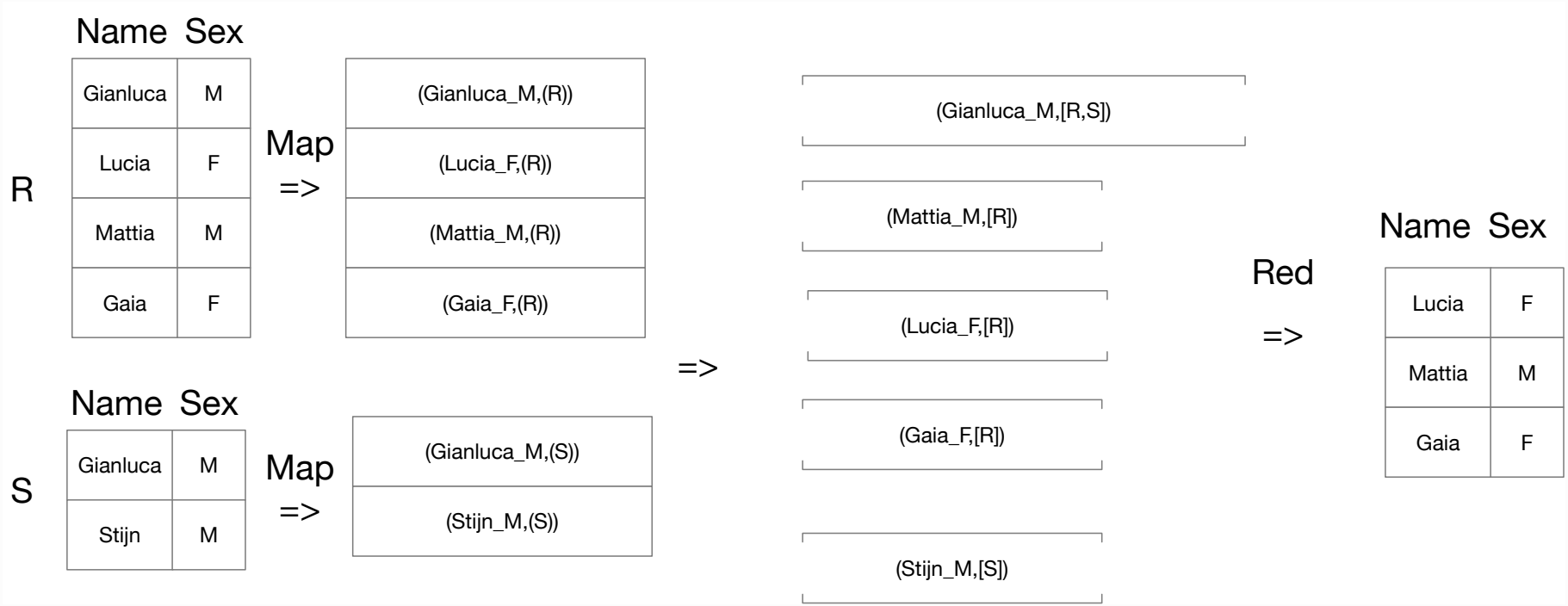



Intersection of R(Name,Sex) and S(Name,Sex)

# MR IMPLEMENTATION OF RELATIONAL OPERATIONS

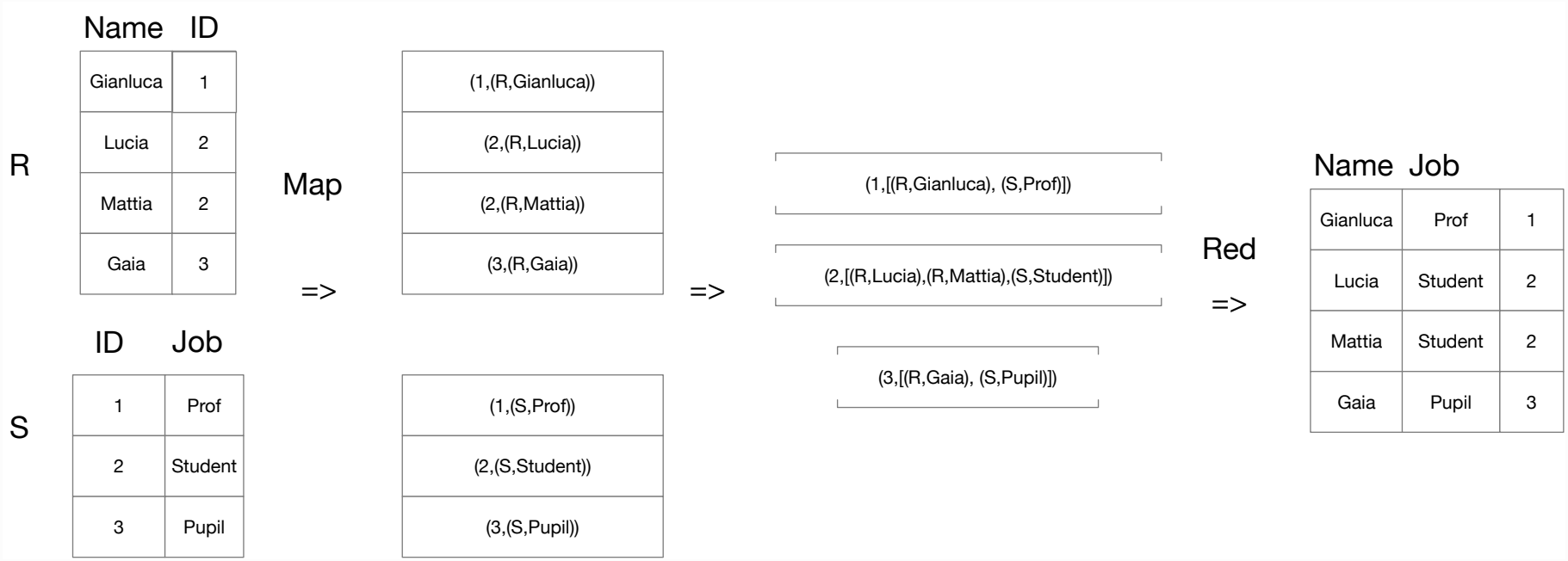
- Difference $R - S$:
  - Map: For a tuple t in R, produce key-value pair $(t, R)$, and for a tuple t in S, produce key-value pair $(t, S)$.
  - Reduce: For each key t, if the associated value list is [R], then return t. Otherwise, return nothing.
- Join $R(A, B) \bowtie_B S(B, C)$:
  - Map: For each tuple $(a, b)$ of R, produce the key-value pair $(b, (R, a))$ . For each tuple $(b, c)$ of S, produce the key-value pair $(b, (S, c))$.
  - Reduce: Each key value b will be associated with a list of pairs of the form $(R, a)$ or $(S, c)$. Construct from those pairs all possible triples of the form $(a, b, c)$.

# MR EXAMPLE: DIFFERENCE OF TWO TABLES




Difference R(Name, Sex) − S(Name, Sex):

# MR EXAMPLE JOIN OF TWO TABLES




Join $R(\text{Name}, \text{ID}) \bowtie_{\text{ID}} S(\text{ID}, \text{Job})$:

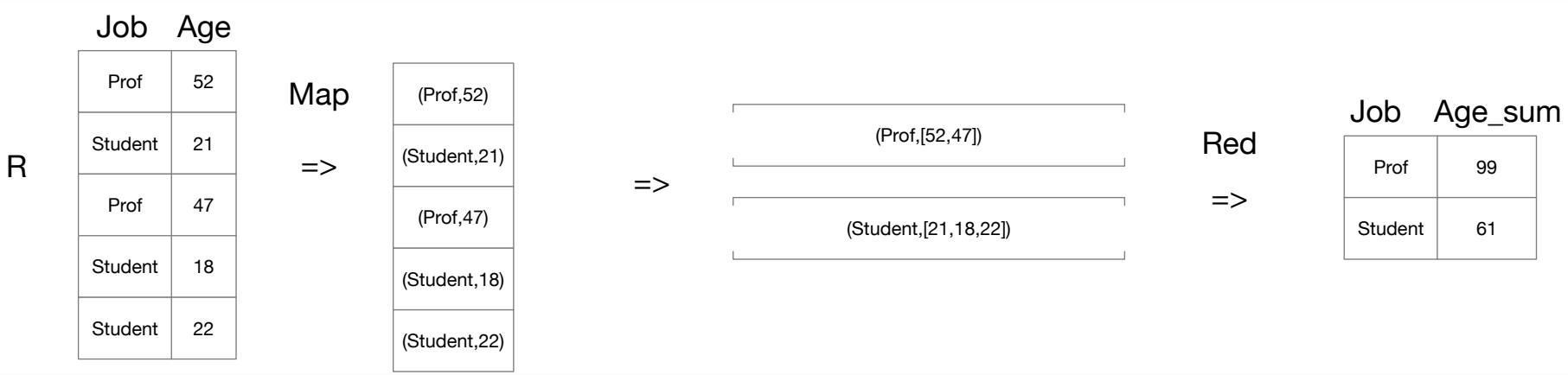
# MR IMPLEMENTATION OF RELATIONAL OPERATIONS

Consider the relation $R(A, B, C)$

- Grouping on A and Aggregation on B: $\gamma_{A,\theta(B)}(R(A, B, C))$
  - Map: For each tuple $(a, b, c)$ produce the key-value pair $(a, b)$.
  - Reduce: Each key a represents a group. Apply the aggregation operator to the list of B-values associated with key a. The output is the pair $(a, x)$ where x is the result of aggregation.

  In case of several grouping attributes, the key is a list of values of a tuple for those attributes. In case of several aggregations, the result of aggregation is a list of values too.

# MR EXAMPLE GROUPING AND AGGREGATION




$\gamma_{\text{Job},\sum(\text{Age})}(\text{R}(\text{Job},\text{Age}))$: Grouping on Job and Aggregation (Sum) on Age

# MATRIX MULTIPLICATION IN RELATIONAL FORM

Let us consider the product $P = MN$ where M is $[M_r, M_c]$, N is $[N_r, N_c]$ and $M_c = N_r$.

- The generic element of P is

$$p_{ik} = \sum_{j=1}^{M_c} m_{ij} n_{jk}$$

- This multiplication can be interpreted in a relational form (particularly useful in the case of sparse matrices) where
  - matrix M is a relation $M(I, J, V)$ with tuples $(i, j, m_{ij})$ and
  - matrix N is a relation $N(J, K, V)$ with tuples $(j, k, n_{jk})$.
- Multiplication as the sequence of
  1. natural join on attribute J
  2. grouping on I and K and
  3. aggregation (sum).

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \end{bmatrix} = \begin{bmatrix} m_{11}n_{11} + m_{12}n_{21} & m_{11}n_{12} + m_{12}n_{22} & m_{11}n_{13} + m_{12}n_{23} \\ m_{21}n_{11} + m_{22}n_{12} & m_{21}n_{12} + m_{22}n_{22} & m_{21}n_{13} + m_{22}n_{23} \end{bmatrix}$$

$R_1 =$

| I | J | M |
|---|---|---|
| 1 | 1 | $m_{11}$ |
| 1 | 2 | $m_{12}$ |
| 2 | 1 | $m_{21}$ |
| 2 | 2 | $m_{22}$ |

$R_2 =$

| J | K | N |
|---|---|---|
| 1 | 1 | $n_{11}$ |
| 1 | 2 | $n_{12}$ |
| 1 | 3 | $n_{13}$ |
| ... | ... | ... |
| 3 | 1 | $n_{31}$ |
| 3 | 2 | $n_{32}$ |
| 3 | 3 | $n_{33}$ |

$R_1 \bowtie_J R_2 =$

| I | J | K | M | N |
|---|---|---|---|---|
| 1 | 1 | 1 | $m_{11}$ | $n_{11}$ |
| 1 | 1 | 2 | $m_{11}$ | $n_{12}$ |
| 1 | 1 | 3 | $m_{11}$ | $n_{13}$ |
| 2 | 1 | 1 | $m_{21}$ | $n_{11}$ |
| 2 | 1 | 2 | $m_{21}$ | $n_{12}$ |
| 2 | 1 | 3 | $m_{21}$ | $n_{13}$ |
| 1 | 2 | 1 | $m_{12}$ | $n_{21}$ |
| 1 | 2 | 2 | $m_{12}$ | $n_{22}$ |
| 1 | 2 | 3 | $m_{12}$ | $n_{23}$ |
| 2 | 2 | 1 | $m_{22}$ | $n_{21}$ |
| 2 | 2 | 2 | $m_{22}$ | $n_{22}$ |
| 2 | 2 | 3 | $m_{22}$ | $n_{23}$ |

$\gamma_{I,K,SUM(M*N)} =$

| I | K | P |
|---|---|---|
| 1 | 1 | $m_{11}n_{11} + m_{12}n_{21}$ |
| 1 | 2 | $m_{11}n_{12} + m_{12}n_{22}$ |
| 1 | 3 | $m_{11}n_{13} + m_{12}n_{23}$ |
| 2 | 1 | $m_{21}n_{11} + m_{22}n_{21}$ |
| 2 | 2 | $m_{21}n_{12} + m_{22}n_{22}$ |
| 2 | 3 | $m_{21}n_{13} + m_{22}n_{23}$ |

# MATRIX MULTIPLICATION IN MR (TWO STEPS)

Step 1:

- Map: for each element of M produce $(j, (M, i, m_{ij}))$ and for each element of N produce $(j, (N, k, n_{jk}))$
- Reduce: for each key j consider all the pairs of elements coming from M and N and compute $m_{ij} n_{jk}$ and return $((i, k), m_{ij} n_{jk})$

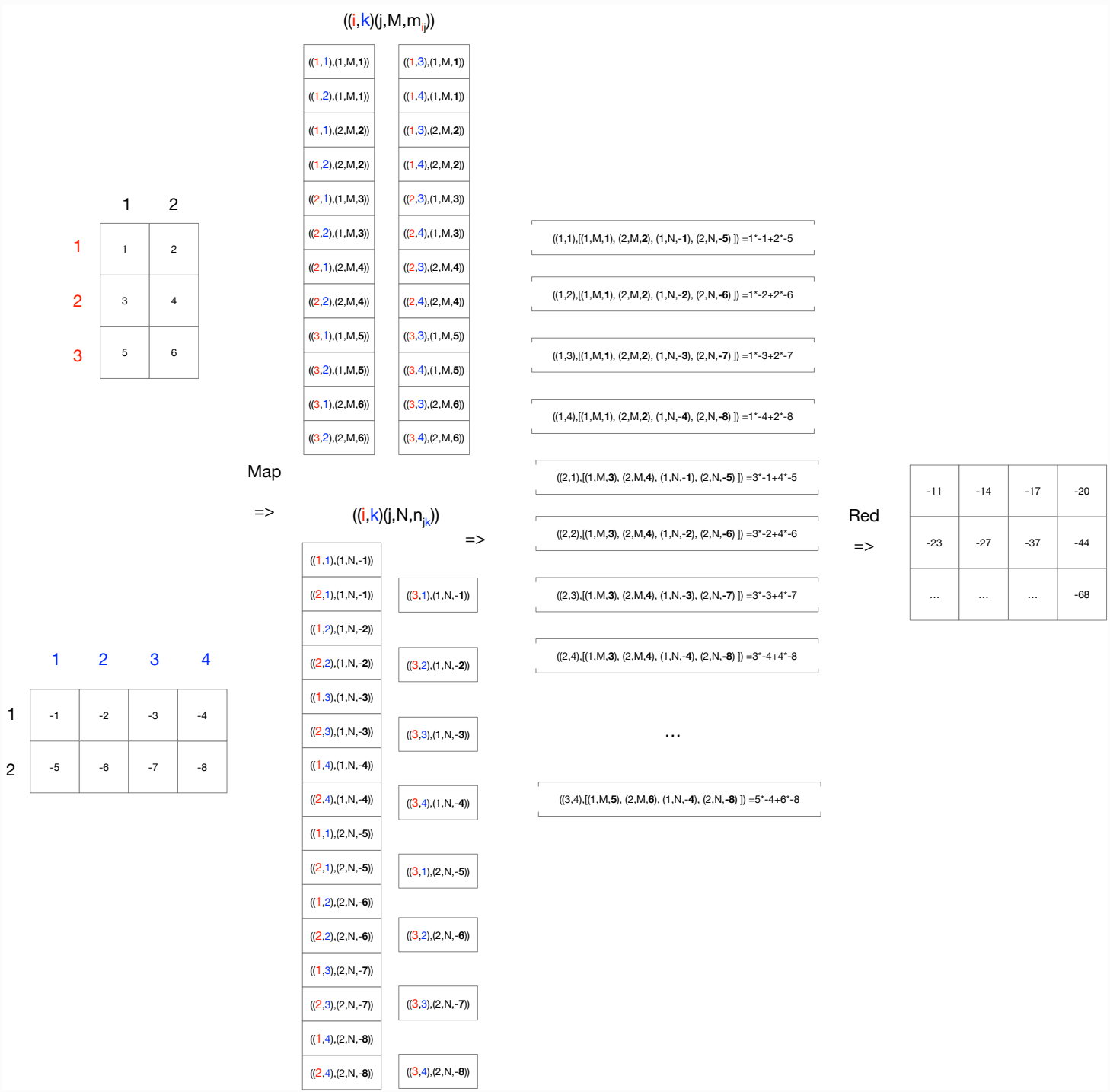
Step 2:

- Map: identity function
- Reduce: for each key $(i, k)$ return the sum of the list of values associated with the key.

# MATRIX MULTIPLICATION IN MR (SINGLE STEP)

Matrix $M = [m_{ik}]$ of size $M_r \times M_c$
Matrix $N = [n_{jk}]$ of size $N_r \times N_c$ and $M_c = N_r$

- Map:
  1. for each element of $M$ produce $((i, k), (j, M, m_{ij}))$ for $i = 1, 2, \ldots, M_r$, $k = 1, 2, \ldots, N_c$, $j = 1, \ldots, M_c$
  2. for each element of $N$ produce $((i, k), (j, N, n_{jk}))$ for $i = 1, 2, \ldots, M_r$, $k = 1, 2, \ldots, N_c$, $j = 1, \ldots, M_c$,
- Reduce: for each key $(i, k)$ consider all the pairs $(j, M, m_{ij})$ and $(j, N, n_{jk})$. Sort them according to the $j$ , multiply the values corresponding to the same $j$ and sum them.


((i,k)(j,M,m_ij))
1 2
1 2
3 4
5 6
((1,1),(1,M,1))
((1,2),(1,M,1))
((1,1),(2,M,2))
((1,2),(2,M,2))
((2,1),(1,M,3))
((2,2),(1,M,3))
((2,1),(2,M,4))
((2,2),(2,M,4))
((3,1),(1,M,5))
((3,2),(1,M,5))
((3,1),(2,M,6))
((3,2),(2,M,6))
((1,3),(1,M,1))
((1,4),(1,M,1))
((1,3),(2,M,2))
((1,4),(2,M,2))
((2,3),(1,M,3))
((2,4),(1,M,3))
((2,3),(2,M,4))
((2,4),(2,M,4))
((3,3),(1,M,5))
((3,4),(1,M,5))
((3,3),(2,M,6))
((3,4),(2,M,6))
Map
=>
((i,k)(j,N,n_jk))
=>
1 2 3 4
1 -1 -2 -3 -4
2 -5 -6 -7 -8
((1,1),(1,N,-1))
((2,1),(1,N,-1))
((1,2),(1,N,-2))
((2,2),(1,N,-2))
((1,3),(1,N,-3))
((2,3),(1,N,-3))
((1,4),(1,N,-4))
((2,4),(1,N,-4))
((1,1),(2,N,-5))
((2,1),(2,N,-5))
((1,2),(2,N,-6))
((2,2),(2,N,-6))
((1,3),(2,N,-7))
((2,3),(2,N,-7))
((1,4),(2,N,-8))
((2,4),(2,N,-8))
((3,1),(1,N,-1))
((3,2),(1,N,-2))
((3,3),(1,N,-3))
((3,4),(1,N,-4))
((3,1),(2,N,-5))
((3,2),(2,N,-6))
((3,3),(2,N,-7))
((3,4),(2,N,-8))
((1,1),[(1,M,1), (2,M,2), (1,N,-1), (2,N,-5) ]) =1*-1+2*-5
((1,2),[(1,M,1), (2,M,2), (1,N,-2), (2,N,-6) ]) =1*-2+2*-6
((1,3),[(1,M,1), (2,M,2), (1,N,-3), (2,N,-7) ]) =1*-3+2*-7
((1,4),[(1,M,1), (2,M,2), (1,N,-4), (2,N,-8) ]) =1*-4+2*-8
((2,1),[(1,M,3), (2,M,4), (1,N,-1), (2,N,-5) ]) =3*-1+4*-5
((2,2),[(1,M,3), (2,M,4), (1,N,-2), (2,N,-6) ]) =3*-2+4*-6
((2,3),[(1,M,3), (2,M,4), (1,N,-3), (2,N,-7) ]) =3*-3+4*-7
((2,4),[(1,M,3), (2,M,4), (1,N,-4), (2,N,-8) ]) =3*-4+4*-8
...
((3,4),[(1,M,5), (2,M,6), (1,N,-4), (2,N,-8) ]) =5*-4+6*-8
Red
=>
-11 -14 -17 -20
-23 -27 -37 -44
... ... ... -68

# ROADMAP TO SCALABILITY

# BIG DATA ANALYTICS IN PRACTICE

Two possible solutions

1. Use some recent open source machine learning platforms for big data
   - Mahout on top of Hadoop. It implements classification (perceptron, logistic regression, Naive Bayes, random forest, SVM), clustering, locally weighted regression, SVD (singular value decomposition), PCA, ICA
   - Trident ML on top of Storm. It implements linear classification (Perceptron), linear regression, clustering, feature scaling (standardization, normalization)
   - MLLib on top of Spark. It implements in iterative manner binary classification (SVM and Logistic Regression), linear regression, clustering (k-mean), collaborative filtering for recommender system.
2. Redesign and/or adapt algorithms in a scalable manner according to the MR paradigm.

# DISTRIBUTION OF LEARNING ALGORITHMS

We will consider a number of algorithms

- Supervised learning
  - Regression: least squares
  - Classification: Naive Bayes classification
  - Regression/Classification: KNN
  - Averaging algorithms
- Unsupervised learning: Kmeans
- Feature selection:
  - Ranking/correlation
  - mRMR feature selection

# NOTA BENE

- MR is **not** a solution to every problem, not even every problem that profitably can use many compute nodes operating in parallel
- The use of a distributed-file-system makes sense only when files are very large and are **rarely updated** in place.
- The **communication cost** is the dominant cost, since each single task is typically very simple and the interconnect speed of a computing cluster is much lower than the processor execution speed.
- Different MR implementations of the same algorithm may induce different communication costs

# ROUTES TO SCALABILITY

Make (not all) ML algorithms scalable by using one of these principles

1. Summation principle
   - expectation
   - cost function
   - gradient
   - log-likelihood
   - frequency
   - mean, variance
2. Independent tasks
   - feature ranking
   - cross-validation, bagging
   - testing
   - model selection (racing)
3. Statistical sampling properties (estimation)
4. Divide-and-conquer strategy: nonlinear dependence decomposed into local simpler dependencies.

# SUMMATION FORM PRINCIPLE

- ML and **sums** are strongly related.
- Accuracy of a learner (e.g. Mean-squared-error) expressed as a function (typically expectation) of the data distribution.
- Estimation of the accuracy on the basis of data (e.g. by cross-validation) requires the computation of sampled versions of the expectation, i.e. means or sums over the data.
- Averaging approaches (e.g. bagging) rely on averages.
- Probability estimations via frequency require computations of sums.
- Training cost functions can be written as sums: for instance sum of squared errors.
- If the error cost function is a sum, the gradient of the error is a sum too.

# SUMMATION FORM PRINCIPLE

- Once an algorithm does sums over the data we can easily distribute the calculations over multiple workers (e.g. cores)

$$a + b + c + d + e + f + ... = (a + b) + (c + d) + (e + f) + ....$$

- We first divide the dataset into as many pieces as workers (partitioning)
- Then we give each worker a block of data (scatter)
- Eventually we aggregate the results (gather).
- Well-known example of algorithm in summation form is least-squares.
- Sum is an example of **monoid** operator that can be easily made parallel in the reduce step.

# MONOIDS AND MR

- A monoid is composed by a set S, an associative binary mapping $\cdot : S \times S \to S$ and an identity element.
- An associative operation $\cdot$ satisfies the following equality for all a, b, c

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- In MR, a mapper is not constrained while the reducer may be distributed by implementing (the iterated application of) an associative operation
- Commutative monoids: max, addition, multiplication, union , intersection

$$\mathbf{max}(a, \mathbf{max}(b, c)) = \mathbf{max}(\mathbf{max}(a, b), c)$$

- Non monoids: mean, median,subtraction

$$\text{avg}(1, 2, 3, 4, 5) \neq \text{avg}(\text{avg}(1, 2, 3), \text{avg}(4, 5))$$

- Since average is not associative but sum and count are, the reduce implementation of average relies on sum and counting.

(key, value)

R

| Job | Age |
|---|---|
| Prof | 52 |
| Student | 21 |
| Prof | 47 |
| Student | 18 |
| Student | 22 |

Map

=>

| |
|---|
| (Prof,52) |
| (Student,21) |
| (Prof,47) |
| (Student,18) |
| (Student,22) |

=>

| |
|---|
| (Prof,[52,47]) |
| (Student,[21,18,22]) |

Red
(No monoid)

=>

| Job | Age_mean |
|---|---|
| Prof | 49.5 |
| Student | 20.33 |

(key, (value, count) )

R

| Job | Age |
|---|---|
| Prof | 52 |
| Student | 21 |
| Prof | 47 |
| Student | 18 |
| Student | 22 |

Map

=>

| |
|---|
| (Prof,(52,1)) |
| (Student, (21,1)) |
| (Prof,(47,1)) |
| (Student, (18,1)) |
| (Student, (22,1)) |

=>

| |
|---|
| (Prof,[(52,1),(47,1)]) |
| (Student,[(21,1),(18,1)]) |
| (Student,[ (22,1)]) |

=>

| |
|---|
| (Prof,[(99,2)]) |
| (Student,[(39,2),(22,1)]) |

Red
(Monoid)

=>

| Job | Age_mean |
|---|---|
| Prof | 49.5 |
| Student | 20.33 |

# INDEPENDENCE

Several phases of ML pipeline may be carried out independently, where independence can be interpreted both in **computational** (no communication) and **statistical** (no information) sense

- Prediction for different test values; note that prediction is the most expensive step in some algorithms (lazy).
- Cross-validation
- Averaging of estimators (bagging)
- Univariate ranking of features (assumption of independence)
- Generation of i.i.d. subsamples.
- Conditional independence (Naive Bayes)

# STATISTICAL SAMPLING

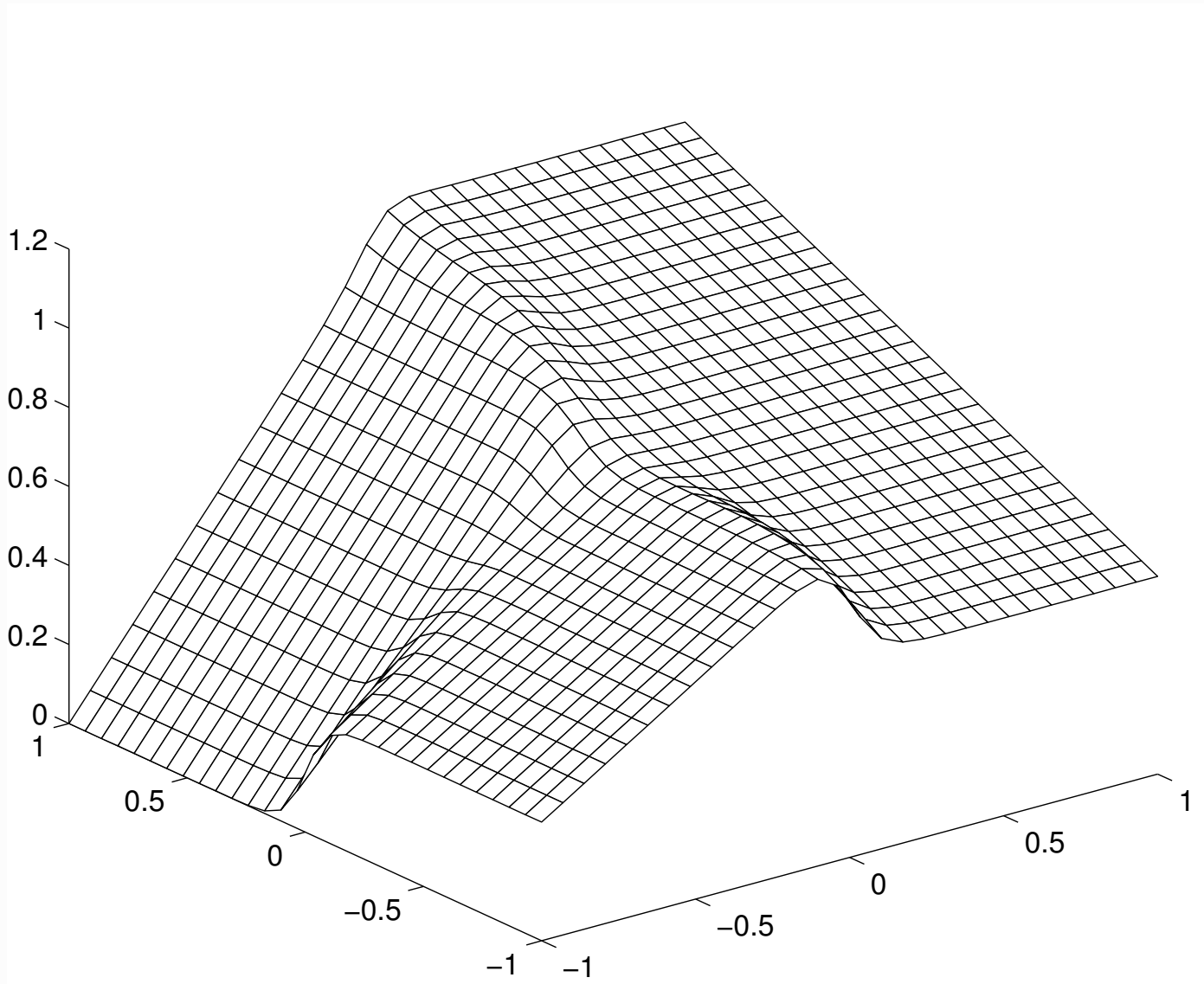
- Statistical sampling: it allows inferences about a population to be made from observations made on a relatively small number of individuals in the population (the sample).
- The idea is to use the original big dataset as the population and portions of it as samples to infer properties of the original dataset
- Map can create several subsamples that can be parallelly analysed at the Reduce level.

# DIVIDE-AND-CONQUER

Learning strategy that attacks a complex problem by dividing it into simpler problems whose solutions can be combined to yield a solution to the original problem. This principle presents two main advantages.

1. simpler problems can be solved with simpler estimation techniques; in statistics this means to adopt linear techniques, well studied and developed over the years.
2. the learning method can better adjust to the properties of the available dataset.

Examples: Radial Basis Function, KNN, local learning


1.2
1
0.8
0.6
0.4
0.2
0
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

# SOME MR ML ALGORITHMS

# MULTIPLE LINEAR DEPENDENCY

- Consider a linear relation between an independent variable $x \in \mathcal{X} \subset \mathbb{R}^n$ and a dependent random variable $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}$

$$\mathbf{y} = \beta_0 + \beta_1 x_{\cdot 1} + \beta_2 x_{\cdot 2} + \cdots + \beta_n x_{\cdot n} + \mathbf{w}$$

  where $\mathbf{w}$ represents a random variable with mean zero and constant variance $\sigma^2_{\mathbf{w}}$.
- In matrix notation the equation can be written as:

$$\mathbf{y} = x^T \beta + \mathbf{w}$$

  where x stands for the $[p \times 1]$ vector $x = [1, x_{\cdot 1}, x_{\cdot 2}, \ldots, x_{\cdot n}]^T$, $\beta = [\beta_0, \ldots, \beta_n]^T$ is the vector of parameters and $p = n + 1$ is the total number of model parameters.
- NB: in the following $x_{\cdot i}$ will denote the ith variable of the vector x, while $x_i$ will denote the ith observation of the vector x.

# THE MULTIPLE LINEAR REGRESSION MODEL

Consider N observations $D_N = \{\langle x_i, y_i\rangle : i = 1, \ldots, N\}$, where $x_i = (1, x_{i1}, \ldots, x_{in})$, generated according to the previous model. We suppose that the following multiple linear relation holds

$$Y = X\beta + W$$

where Y is the $[N \times 1]$ response vector, X is the $[N \times p]$ data matrix, whose $j^{th}$ column of X contains readings on the $j^{th}$ regressor, $\beta$ is the $[p \times 1]$ vector of parameters

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ . \\ . \\ . \\ x_N^T \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ . \\ \beta_n \end{bmatrix}$$

# NORMAL EQUATIONS

The **least-squares estimator** $\hat{\boldsymbol{\beta}}$ minimizes the cost function

$$\hat{\beta} = \arg\min_{b} \sum_{i=1}^{N} (y_i - x_i^T b)^2 = \arg\min_{b} \left( (Y - Xb)^T (Y - Xb) \right)$$

It can be shown that it satisfies the least-squares normal equations

$$(X^T X)\hat{\beta} = X^T Y$$

Assuming $X$ is of full column rank, we obtain

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

where the $X^T X$ matrix is a symmetric $[p \times p]$ matrix.

# MR DISTRIBUTION OF THE LEAST-SQUARES SOLUTION

Two settings

- very large N $>>$ n with n small: distribution of the computations of $X^TX$ and $X^TY$
- very large N and n: iterative solution by stochastic gradient descent (see class on sequential algorithms)

# LEAST-SQUARES SUMMATION FORM

Suppose $N = 3$, $p = 2$

$$X^T X = \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} =$$

$$= \begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 & x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} \\ x_{12}x_{11} + x_{22}x_{21} + x_{32}x_{31} & x_{12}^2 + x_{22}^2 + x_{32}^2 \end{bmatrix}$$

$$x_1^T x_1 = \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \end{bmatrix} = \begin{bmatrix} x_{11}^2 & x_{11}x_{12} \\ x_{11}x_{12} & x_{12}^2 \end{bmatrix}$$

then

$$X^T X = \sum_{i=1}^{N} x_i^T x_i$$

where $x_i$ is the $[1, p]$ vector denoting the ith row of the matrix X.

# LEAST-SQUARES SUMMATION FORM

Analogously it can be shown that

$$X^T Y = \sum_{i=1}^{N} x_i^T y_i$$

where $y_i$ is the ith value of the vector Y and $X^T Y$ has a size $[p, 1]$.

# DISTRIBUTED COMPUTATION OF $X^tX$ AND $X^ty$

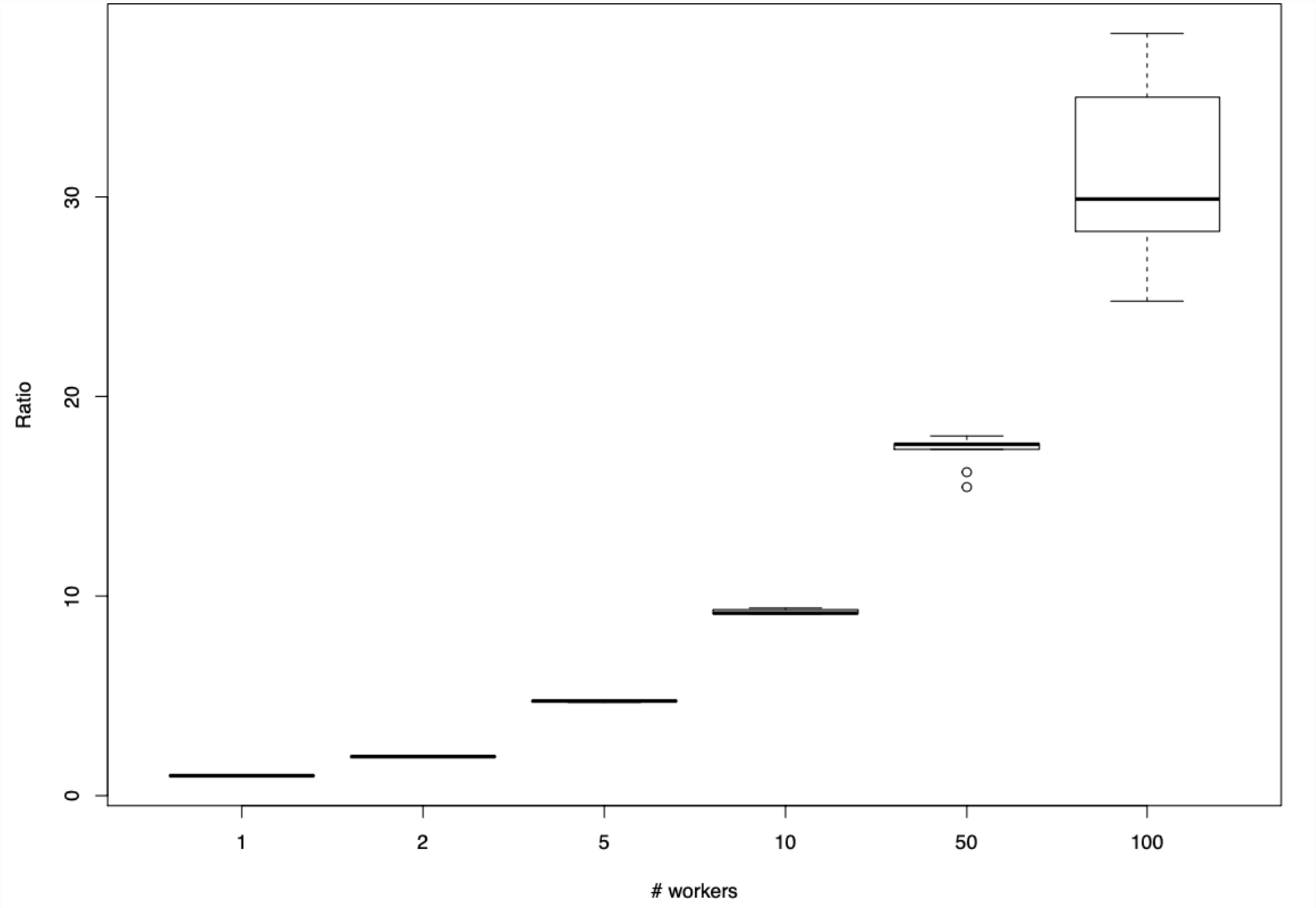
Create a matrix Z of size $[N, p+1]$ by concatenating X and Y

1. Map: computation of partial $X^TX$. For each ith row of X compute $x_i^T x_i$ of size $[p, p]$
2. Reduce: sum all $x_i^T x_i$ .
3. Map: computation of partial $X^TY$. For each ith row of X ($i = 1, \ldots, N$) compute $x_i^T y_i$ of size $[p, 1]$
4. Reduce: sum all $x_i^T y_i$.
5. Collect: collect the reduced matrices $X^TX$ and $X^TY$, and compute $\hat{\beta}$ in a conventional manner.

# SOME CONSIDERATIONS

- Instead of a row-wise computation it is possible to implement a **partition-wise** computation
- Summation form allows a linear speed up with the number of workers.
- Intermediate results of least-squares are useful in other tasks, too.
- $X^TX$ is an estimation of the **covariance matrix** (if all the columns of X are normalized, i.e. zero mean and unit variance): frequently used in linear statistics (e.g. Principal Component Analysis, Linear Discriminant Analysis)
- $X^TY$ is an estimation of the **correlation** input/output vector (again in normalized settings): frequently used in linear statistics, e.g. ranking of features
- Note that above computations are scalable with N but not n.

# SPEED-UP ASSESSMENT



# NAIVE BAYES CLASSIFIER

- NB classifier has shown in some domains an accuracy comparable to that of neural networks and decision tree learning.
- Consider a binary classification problem with n inputs and a random output variable $\mathbf{y}$ that takes values in the set $\{c_1, \ldots, c_K\}$.
- The Bayes optimal classifier should return

$$c^*(x) = \arg \max_{k=1,\ldots,K} \text{Prob}\{\mathbf{y} = c_k | x\}$$

- We can use Bayes' theorem to rewrite this expression as

$$c^*(x) = \arg \max_{k=1,\ldots,K} \frac{\text{Prob}\{x|\mathbf{y} = c_k\}\,\text{Prob}\{\mathbf{y} = c_k\}}{\text{Prob}\{x\}} =$$

$$= \arg \max_{k=1,\ldots,K} \text{Prob}\{x|\mathbf{y} = c_k\}\,\text{Prob}\{\mathbf{y} = c_k\}$$

# NAIVE BAYES CLASSIFIER

- How to estimate these two terms with finite dataset?
- Prob $\{\mathbf{y} = c_k\}$: frequency with which each target class occurs in the training set.
- Estimation of Prob $\{x|\mathbf{y} = c_k\}$ much harder.
- NB is based on the simplifying assumption that inputs are conditionally independent given the target value:

$$\text{Prob}\{x|\mathbf{y} = c_k\} = \text{Prob}\{x_1, \ldots, x_n|\mathbf{y} = c_k\} = \prod_{j=1}^{n} \text{Prob}\{x_j|\mathbf{y} = c_k\}$$

- The NB classification is then

$$c_{NB}(x) = \arg \max_{k=1,\ldots,K} \text{Prob}\{\mathbf{y} = c_k\} \prod_{j=1}^{n} \text{Prob}\{x_j|\mathbf{y} = c_k\}$$

# DISTRIBUTED COMPUTATION OF NAIVE BAYES CLASSIFIER

Let us suppose that all data are categorical

- If inputs $x_j$ are binaries the estimation of Prob $\{x_j | \mathbf{y} = c_k\}$ boils down to the counting of the frequencies of the occurrences of the different values of $x_j$ for a class $c_k$.
- We need to sum over $x_j = 0$ and $x_j = 1$ for each label $c_k$
- Once data are partitioned, it is enough to compute a sum for each block and then aggregate the results

# EXAMPLE

Let us consider the following dataset (example from [5])

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis (classificati |
|---|---|---|---|---|---|
| $D_1$ | Sunny | Hot | High | Weak | No |
| $D_2$ | Sunny | Hot | High | Strong | No |
| $D_3$ | Overcast | Hot | High | Weak | Yes |
| $D_4$ | Rain | Mild | High | Weak | Yes |
| $D_5$ | Rain | Cool | Normal | Weak | Yes |
| $D_6$ | Rain | Cool | Normal | Strong | No |
| $D_7$ | Overcast | Cool | Normal | Strong | Yes |
| $D_8$ | Sunny | Mild | High | Weak | No |
| $D_9$ | Sunny | Cool | Normal | Weak | Yes |
| $D_{l0}$ | Rain | Mild | Normal | Weak | Yes |
| $D_{11}$ | Sunny | Mild | Normal | Strong | Yes |
| $D_{12}$ | Overcast | Mild | High | Strong | Yes |
| $D_{13}$ | Overcast | Hot | Normal | Weak | Yes |
| $D_{14}$ | Rain | Mild | High | Strong | No |

# EXAMPLE: MAP

The map() function counts the attributes and their associations with the classification classes. For instance the map of the first row will return:

```
(<Sunny,No>, <1>)
(<Hot,No>, <1>)
(<High,No>, <1>)
(<Weak,No>, <1>)
(<CLASS,No>, <1>)
```

# EXAMPLE: REDUCER INPUT

| Key | Value |
|---|---|
| <CLASS,No> | [<1>,<1>,<1>,<1>,<1>] |
| <CLASS,Yes> | [<1>,<1>,<1>,<1>,<1>,<1>,<1>,<1>,<1>] |
| <Cool,No> | [<1>] |
| <Cool,Yes> | [<1>,<1>,<1>] |
| <High,No> | [<1>,<1>,<1>,<1>] |
| <High,Yes> | [<1>,<1>,<1>] |
| <Hot,No> | [<1>,<1>] |
| <Hot,Yes> | [<1>,<1>] |
| <Mild,No> | [<1>,<1>] |
| <Mild,Yes> | [<1>,<1>,<1>,<1>] |
| <Normal,No> | [<1>] |
| <Normal,Yes> | [<1>,<1>,<1>,<1>,<1>,<1>] |
| <Overcast,Yes> | [<1>,<1>,<1>,<1>] |
| <Rain,No> | [<1>,<1>] |
| <Rain,Yes> | [<1>,<1>,<1>] |
| <Strong,No> | [<1>,<1>,<1>] |
| <Strong,Yes> | [<1>,<1>,<1>] |
| <Sunny,No> | [<1>,<1>,<1>] |
| <Sunny,Yes> | [<1>,<1>] |
| <Weak,No> | [<1>,<1>] |
| <Weak,Yes> | [<1>,<1>,<1>,<1>,<1>,<1>] |

# EXAMPLE: REDUCER OUTPUT

| Key | Value |
|---|---|
| <CLASS,No> | 5 |
| <CLASS,Yes> | 9 |
| <Cool,No> | 1 |
| <Cool,Yes> | 3 |
| <High,No> | 4 |
| <High,Yes> | 3 |
| <Hot,No> | 2 |
| <Hot,Yes> | 2 |
| <Mild,No> | 2 |
| <Mild,Yes> | 4 |
| <Normal,No> | 1 |
| <Normal,Yes> | 6 |
| <Overcast,Yes> | 4 |
| <Rain,No> | 2 |
| <Rain,Yes> | 3 |
| <Strong,No> | 3 |
| <Strong,Yes> | 3 |
| <Sunny,No> | 3 |
| <Sunny,Yes> | 2 |
| <Weak,No> | 2 |
| <Weak,Yes> | 6 |

# EXAMPLE: CONDITIONAL PROBABILITY

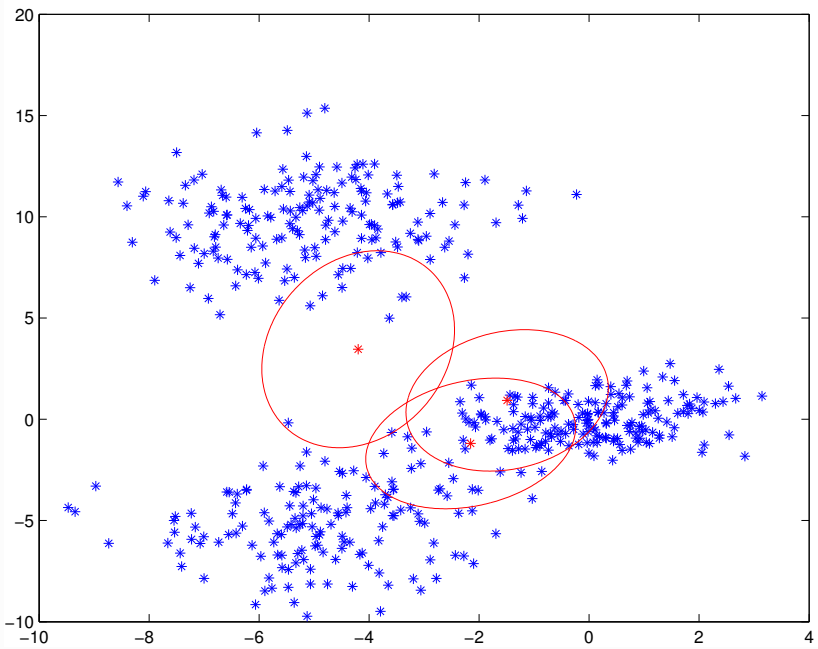
From the output of the reducer it is easy to compute the conditional probability.
For instance for the variable $\mathbf{x}_1$=Outlook that can take as values {Sunny, Overcast, Rain} we obtain

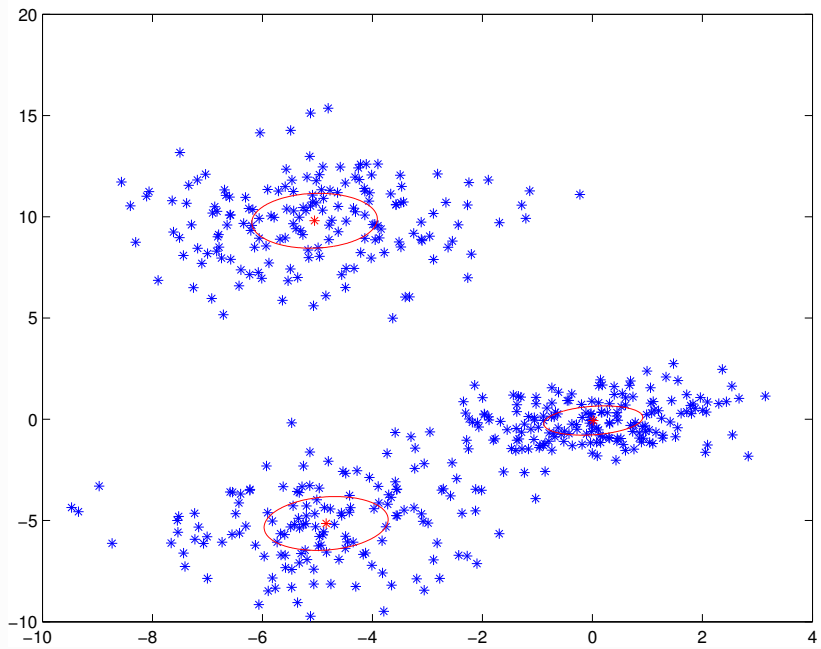
$$\text{Prob}\left\{\mathbf{x}_1 = \text{Sunny}|\mathbf{y} = \text{No}\right\} = \frac{\text{Prob}\left\{\mathbf{x}_1 = \text{Sunny}, \mathbf{y} = \text{No}\right\}}{\text{Prob}\left\{\mathbf{y} = \text{No}\right\}} = \frac{3}{5}$$

where 3 is the value of <Sunny,No> and 5 is the value of <Class,No>. Once the probability table is available it is possible to use it to perform Naive Bayes classification on on new input vectors.

# K-MEANS

- Given $K > 0$ (where K is the number of clusters) and a set of N n-dimensional objects, clustering is the process of grouping a set of N n-dimensional into K clusters of similar objects.
- Objects should be similar to one another within the same cluster and dissimilar to those in other clusters.
- K-Means is a distance-based unsupervised clustering algorithm.
- K-Means clustering has many useful applications. For example, it can be used to find a group of consumers with common behaviors, or to cluster documents based on the similarity of their contents.
- Selection of K is specific to the application or problem domain. There is no magic formula to find it.


20
15
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4


20
15
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4

# K-MEANS

- Initially, K points $C = \{c_1, \ldots, c_K\}$ are chosen as cluster centers; these are called cluster centroids.
- There are many ways to initialize the cluster centroids, one of which is to choose the K points randomly from the sample of n points.
- Once the K initial cluster centroids are chosen, we calculate the distance from every point in the input set to each of the K centers, and then assign each point to the specific cluster center whose distance is closest.
- When all objects have been assigned, then we again recalculate the positions of the K centroids.
- These two steps are repeated until the cluster centroids no longer change (or change very little).

# DISTRIBUTED COMPUTATION OF K-MEANS

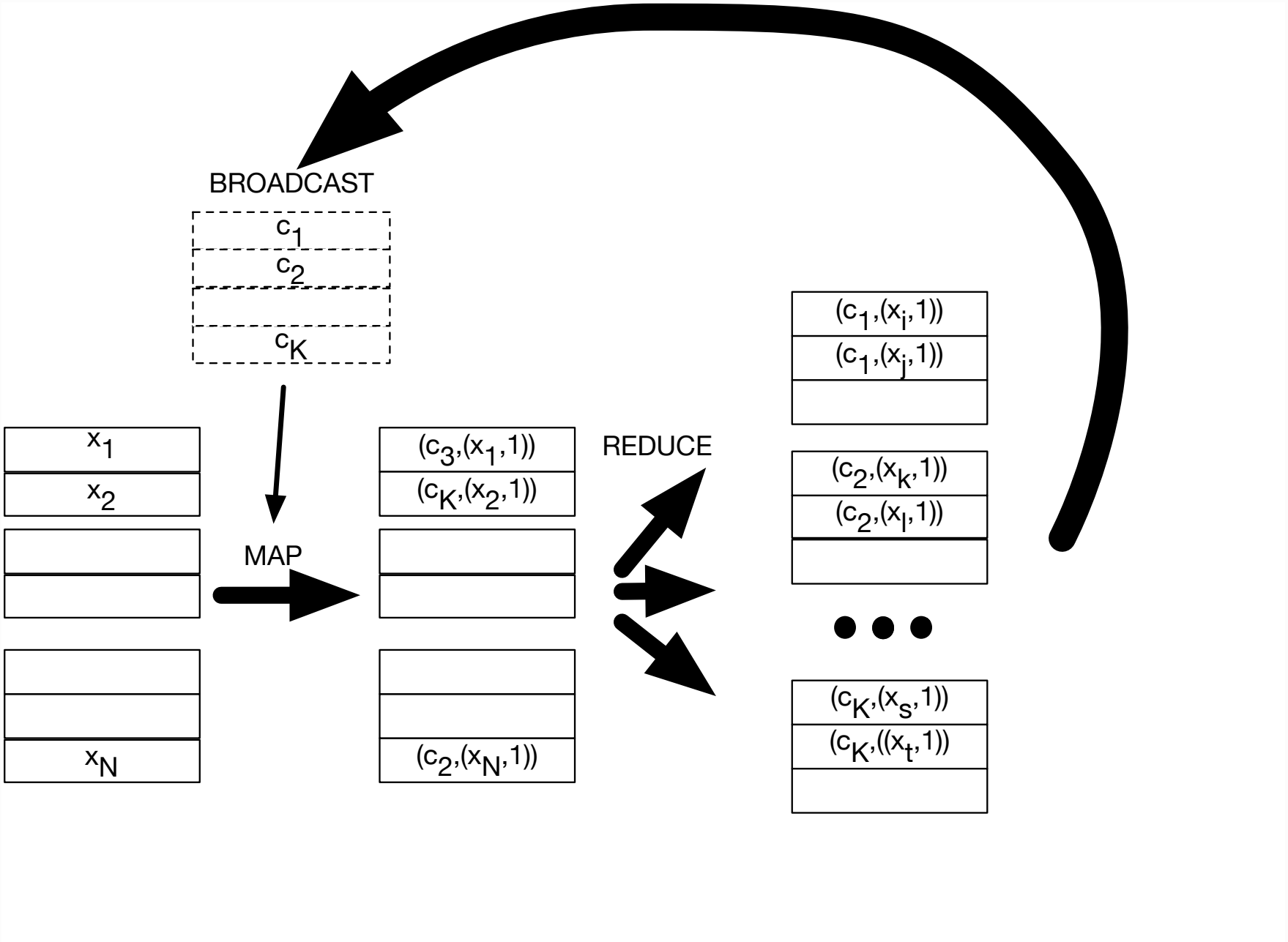
- Cost function is again a sum

$$\arg \min_{c_1,\ldots,c_K} \sum_{k=1}^{K} \sum_{x_i \in \mathcal{C}_k} \|x_i - c_k\|$$

  where $\mathcal{C}_k$ is the set of points of size $[1, n]$ which belong to the kth cluster of size $[1, n]$.
- For a very large N, the bottleneck is the computation of the distances of the N points to the centroids

# MR COMPUTATION OF K-MEANS

1. We partition the dataset X of size [N, n], and we broadcast the set of centroid coordinates of size [K, n].
2. We distribute the computation of the distances to a number of parallel workers
3. A map function returns for each $x_i$ a (key,value) pair where the key is the index $k_i$ of the closest centroid, and the value is a pair $(x_i, 1)$ that will allow to compute the sum and number of observations in each cluster.
4. A reduce step returns the updated coordinates of each cluster centroid by averaging.
5. By iterating the steps above, we converge to the solution.


BROADCAST
$c_1$
$c_2$
$c_K$
$x_1$
$x_2$
$x_N$
MAP
$(c_3,(x_1,1))$
$(c_K,(x_2,1))$
$(c_2,(x_N,1))$
REDUCE
$(c_1,(x_i,1))$
$(c_1,(x_j,1))$
$(c_2,(x_k,1))$
$(c_2,(x_l,1))$
$(c_K,(x_s,1))$
$(c_K,((x_t,1))$

# A KNN CLASSIFIER

Suppose a training set is available and the classification is required for a [1, n] query vector. The classification procedure of a kNN classifier can be summarised in these steps:

1. Compute the distance between the query and the N training samples according to a predefined metric.
2. Rank the neighbours on the basis of their distance to the query.
3. Select a subset of the k nearest neighbors. Each of these neighbours has an associated class.
4. Return the class which characterises the majority of the k nearest neighbors.

# DISTRIBUTED KNN

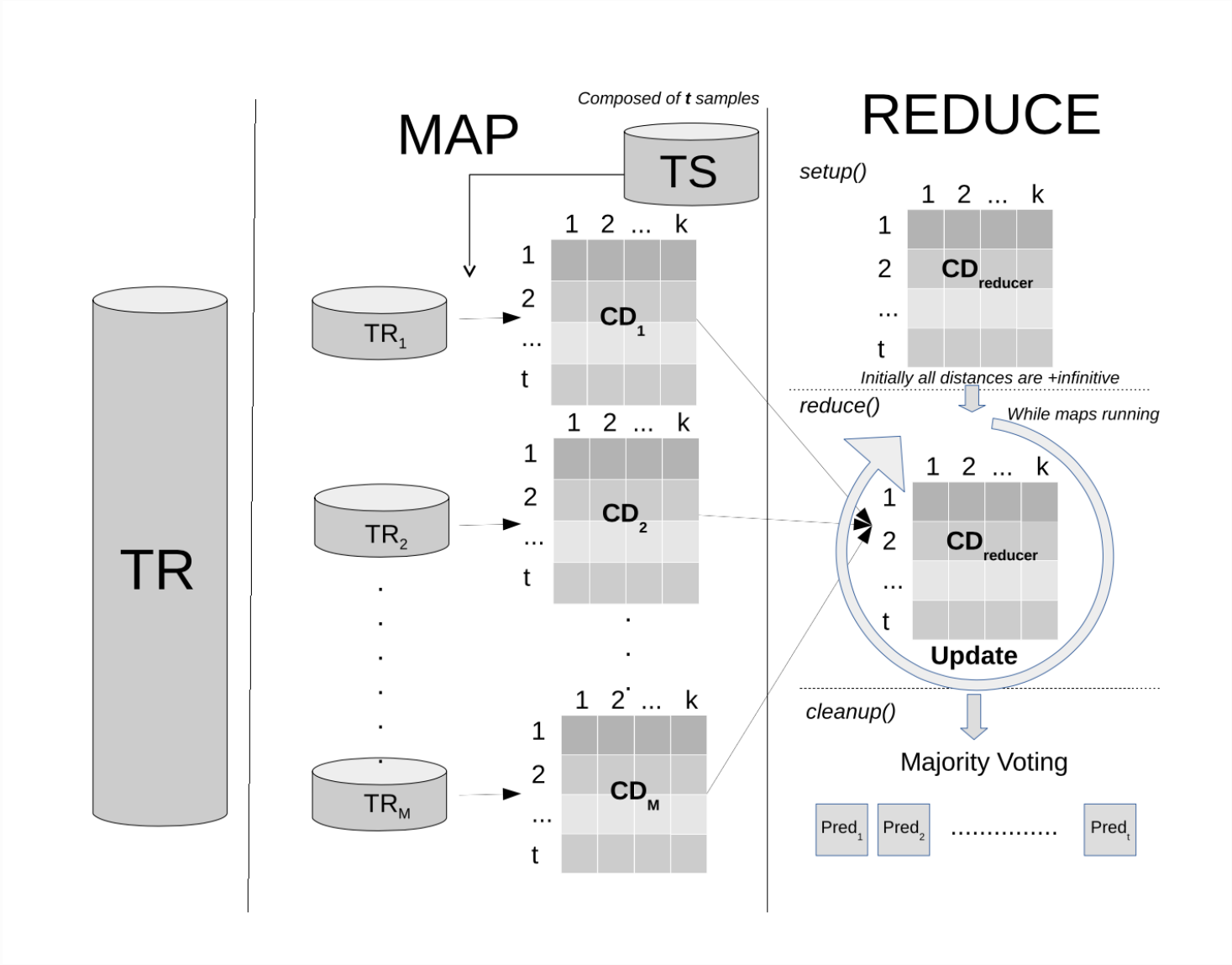
- Lazy algorithm: nothing is done until a query is done.
- No training computational cost.
- All computational effort concerns the prediction step: for each query point, neighbours have to be identified and classification returned
- Problem: comparing a query point with each sample in a huge database is infeasible because of the linear complexity O(N).
- Curse of dimensionality makes the problem still worse.
- However, retrieving a set of approximate nearest neighbours (ANN) is often sufficient.
- Distribution of the prediction computational effort.

# MR-KNN

In [4], the authors propose a MapReduce-based approach for k-Nearest neighbour classification.
In this approach, the **test set of size** [t, n] **is broadcast** to all workers.

- Map: for each chunk of data, it computes the similarity between all the test examples and a portion of the training set: the k nearest neighbours and their computed distance values will be emitted to the reduce stage.
- Reduce: it determines which are the final k nearest neighbours from the list provided by the maps. It is implemented by merging two sorted lists of size k.

# MR-KNN


MAP
Composed of t samples
TS
TR
TR1
TR2
TRM
CD1
CD2
CDM
1 2 ... k
1
2
...
t
REDUCE
setup()
CDreducer
Initially all distances are +infinitive
reduce()
While maps running
Update
cleanup()
Majority Voting
Pred1
Pred2
Predt

From [4]

# PASTING AND RANDOM SUBSPACES

Supervised task for which the dataset cannot reside in memory

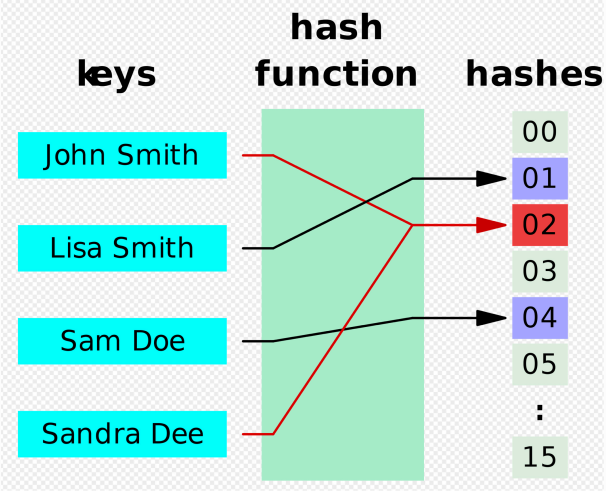
- Breiman (the genial inventor of regression tree, random forest, bagging,...) proposed the Pasting method [1]
- Pasting: learning an ensemble of models individually built on random subsets of the training examples, hence alleviating the memory requirements since the base models would be built on only small parts of dataset.
- Ho [2] proposed to learn an ensemble of models individually built on random on random subsets of the input variables (or features).

# RANDOM PATCHES

- The two ideas has been merged by [3] in the Random Patch algorithm, which consists in creating a number of datasets (covering a portion of the features and a portion of the observations), fit them with a model and combine the predictions.
- The portion of the feature and sample space is controlled by two hyper-parameters.
- Highly parallel: each patch can be managed independently by a processor.
- If the size of the patch is compatible with the central memory, conventional memory architectures may be used.
- Open challenges: subsets of data may present different statistical properties than overall dataset. For example, confidence intervals based on subsets of data will generally be wider than confidence intervals based on the original data (papers of M. Jordan on statistics, computation and scalability).

# HASHES

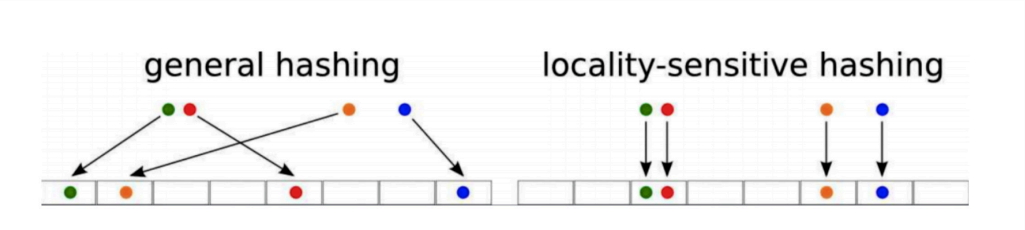
- An hash function takes a **key** value (datum of any type) as input and returns a bucket integer number in a range $[0, B - 1]$ as result.
- Aim of hash function is to randomize keys, i.e. to send approximately an equal number of keys to each bucket number



# LOCALITY SENSITIVITY HASHING

Suppose we have a very large set of elements and we wish to compute the similarity of every pair

- LSH idea: hash each element several times in such a way that similar elements are more likely to be hashed to the same bucket than dissimilar ones.
- Any pair hashed to the same bucket is a candidate pair for similarity
- False positives: dissimilar elements in the same bucket
- False negatives: truly similar elements not hashing to the same bucket.
- LSH also known as near-neighbour search



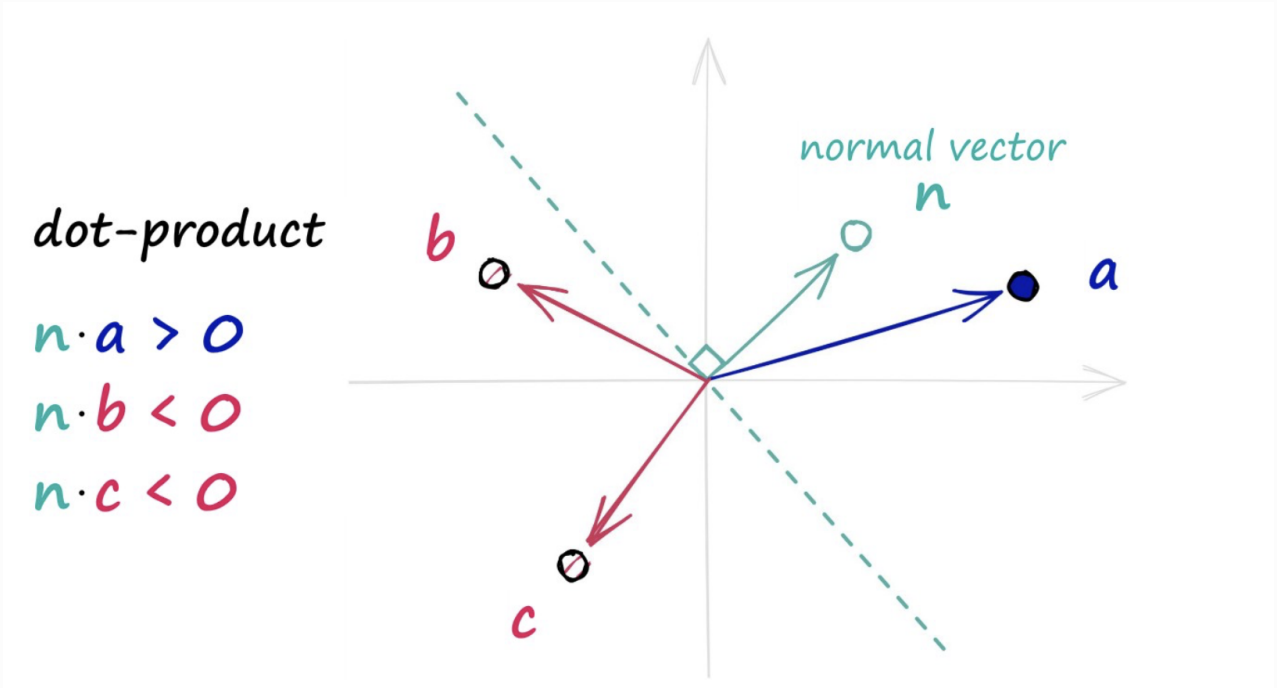
# LSH FUNCTIONS FOR COSINE DISTANCE

Let us consider two vectors $x_1 \in \mathbb{R}^n, x_2 \in \mathbb{R}^n$ in a n-dimensional space and the cosine distance.

- The locality-sensitive family for the cosine distance is a set of B binary codes functions, each built from a randomly chosen vector $h_b \in \mathbb{R}^n, b = 1, \ldots, B$
- $f_b(x) = \text{sign}(h_b \cdot x)$, i.e. $f_b(x_1) = f_b(x_2)$ if and only if the dot products $h_b \cdot x_1$ and $h_b \cdot x_2$ have the same sign
- each n-dimensional vector is transformed in a B-dimensional binary code. Since $n >> B$ coding reduces storage. and Hamming distance can be calculated efficiently in a bitwise manner.
- a modified version is the **sketch** which consists in restricting to vectors $h_b$ whose components are $+1$ and $-1$

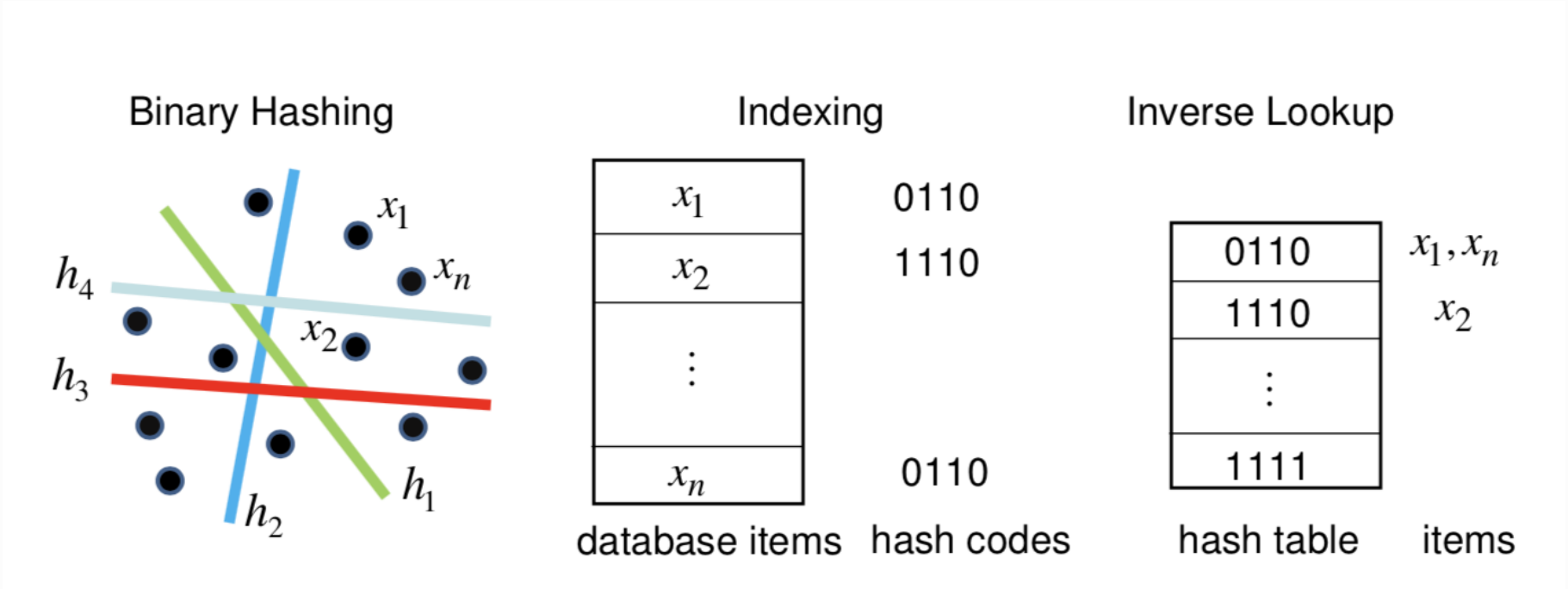
# HYPERPLANCE AND COSINE

Cosine similarity is the cosine of the angle between two vectors.




From `https://www.pinecone.io/learn/locality-sensitive-hashing-random-projection/`

n = 2 input dimensions, B = 4 bits binary code (each corresponding to a hyperplane):




From [8].

# MR DIVIDE AND CONQUER

In case of massive training sets of dimensionality n it could be effective to partition it according to a locality principle.
By defining a number B of sketch vectors $h_b \in \{-1, 1\}^n$, it is is possible to project each training point in a B dimensional space and use the hashing for defining sets of neighbouring points.
Training:

1. Broadcast the S matrix of dimensionality [n, B] where each column corresponds to a sketch vector
2. Map: for each training input vector $x_i$ compute the matrix product $x_i \cdot S$ and store the sign vector as hashing key

For each test point q:

1. compute the matrix product $q \cdot S$ and store the sign vector
2. Filter the training set by selecting only the subset with the same hashing
3. Make a conventional learning procedure by using a subset of the training set

# LEARNING TO HASH

- LSH are data independent hash functions.
- In spite of their elegant theoretical properties their performance has been shown insufficient in many real-world settings.
- To achieve high precision long hash codes are required and semantic proximity is not always respected (semantic gap).
- ”Learning to hash” [8] are recent approaches which use machine learning to exploit data distribution or class labels to optimize the hash function.
- Particularly useful in images where the similarity (or distance) between image pairs is usually not defined via a simple metric.

# RANKING FOR FEATURE SELECTION

- It assesses the importance (or relevance) of each variable with respect to the output by using a univariate measure. They are supervised techniques of complexity O(n).
- Measures of relevance which are commonly used are:
  - Pearson correlation (the greater the more relevant) which assumes linearity;
  - mutual information (the greater the more relevant).
- After the univariate assessment the method ranks the variables in a decreasing order of relevance.
- These methods are fast (complexity O(n)) and their output is intuitive and easy to understand. At the same time they disregard redundancies and higher order interactions between variables (e.g. genes).
- The best k features taken individually do not necessarily constitute the best k variate vector.

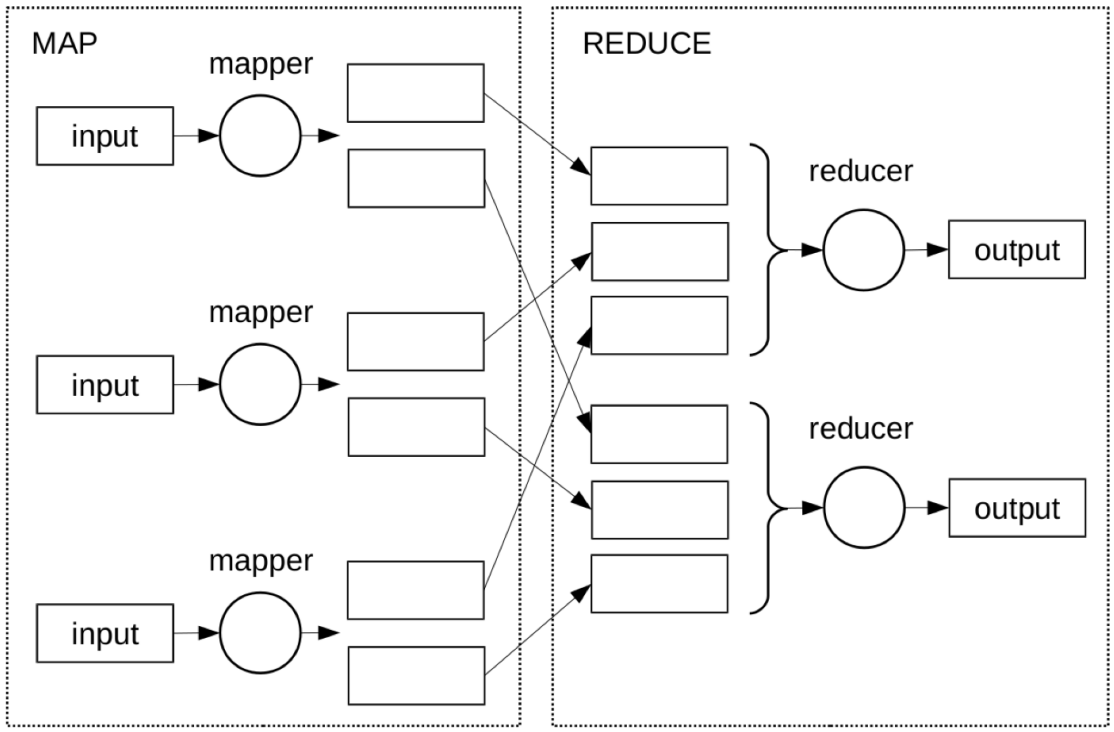
# MR FOR FILTER SELECTION

Supervised task with n input features and one target.
Training input/output data in a rectangular format: each row contains both the input values and the associated output.

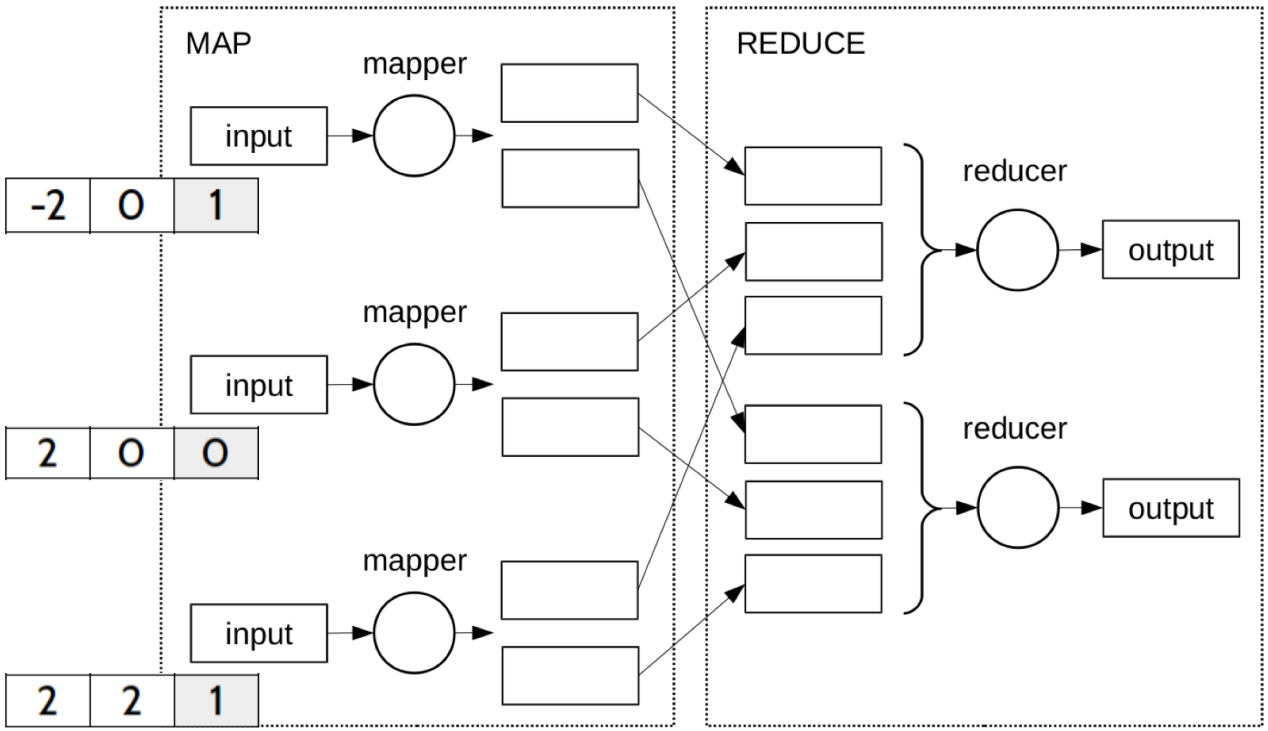
- Map:
  - key: feature index
  - value: pair composed of the feature value and the corresponding target value
- Reduce: the tuples with the same key (referring to the same feature) are grouped and the bivariate score (e.g. correlation or mutual information) computed.

The output is a vector of size n containing the association score of each input features.
For more advanced strategies look at [7].

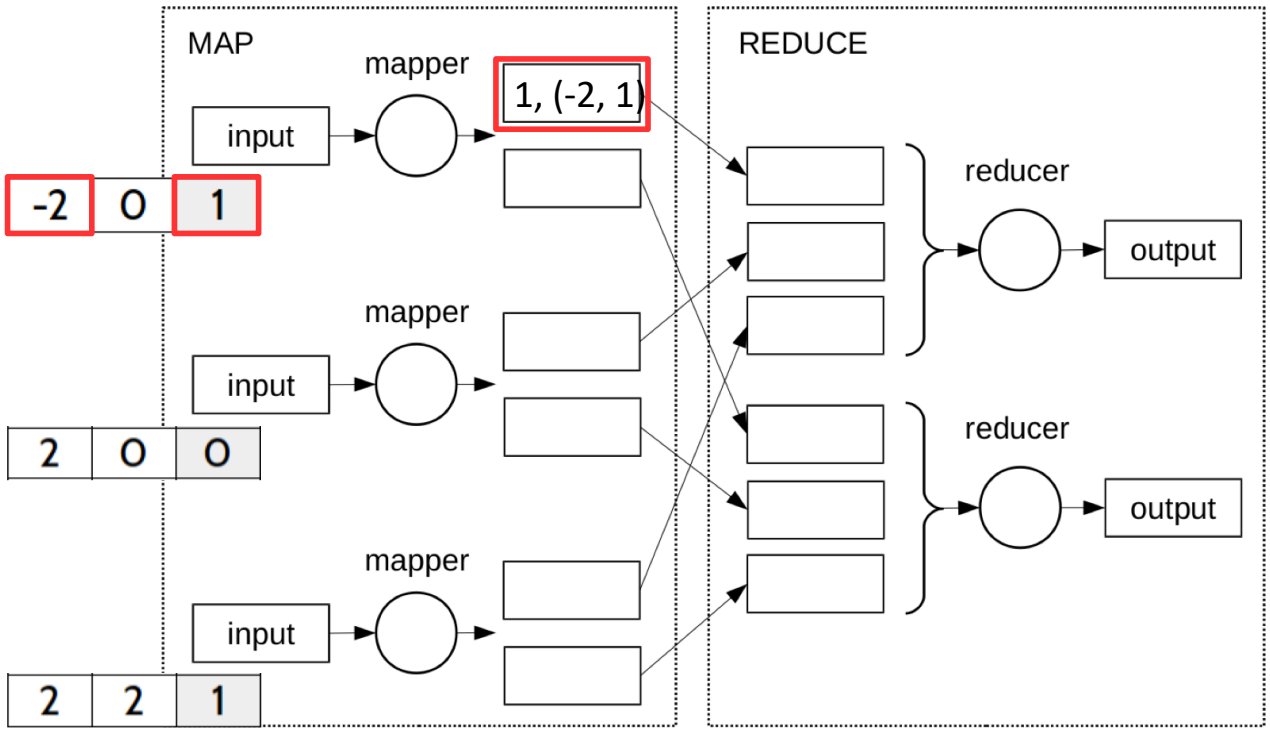
| *1* | *2* | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |



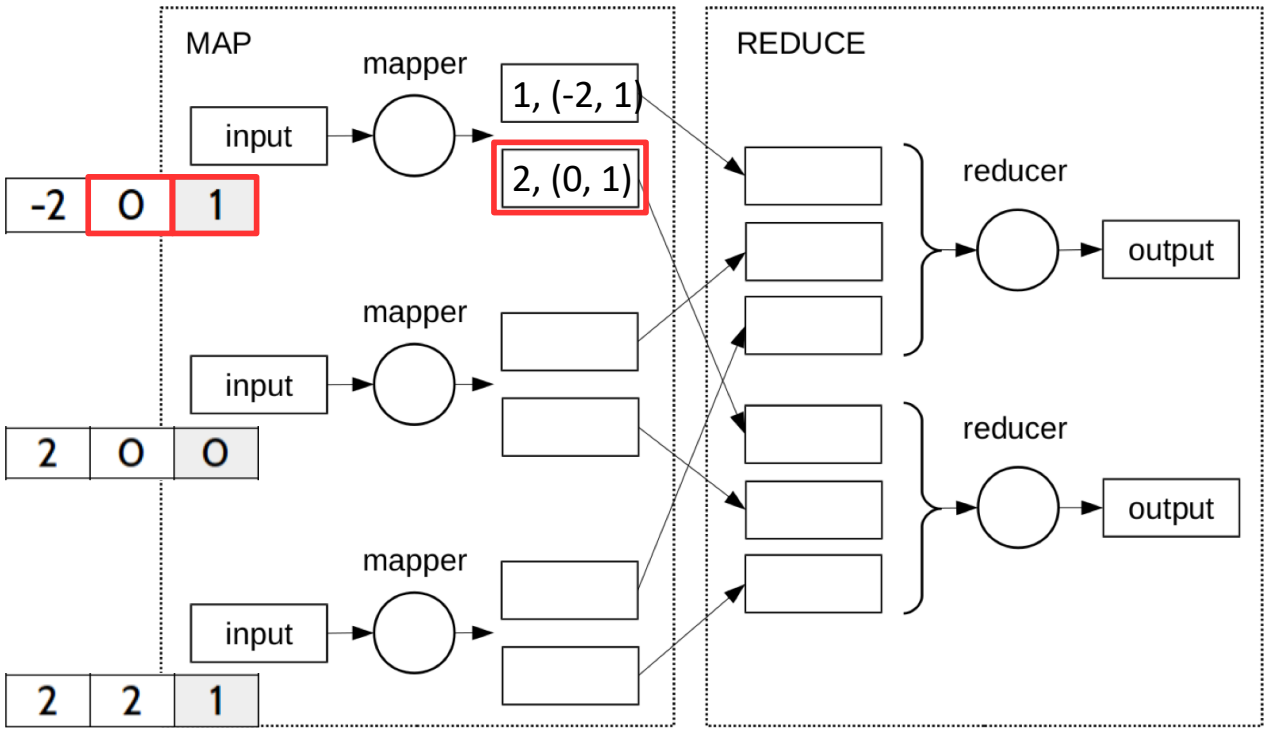
| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |

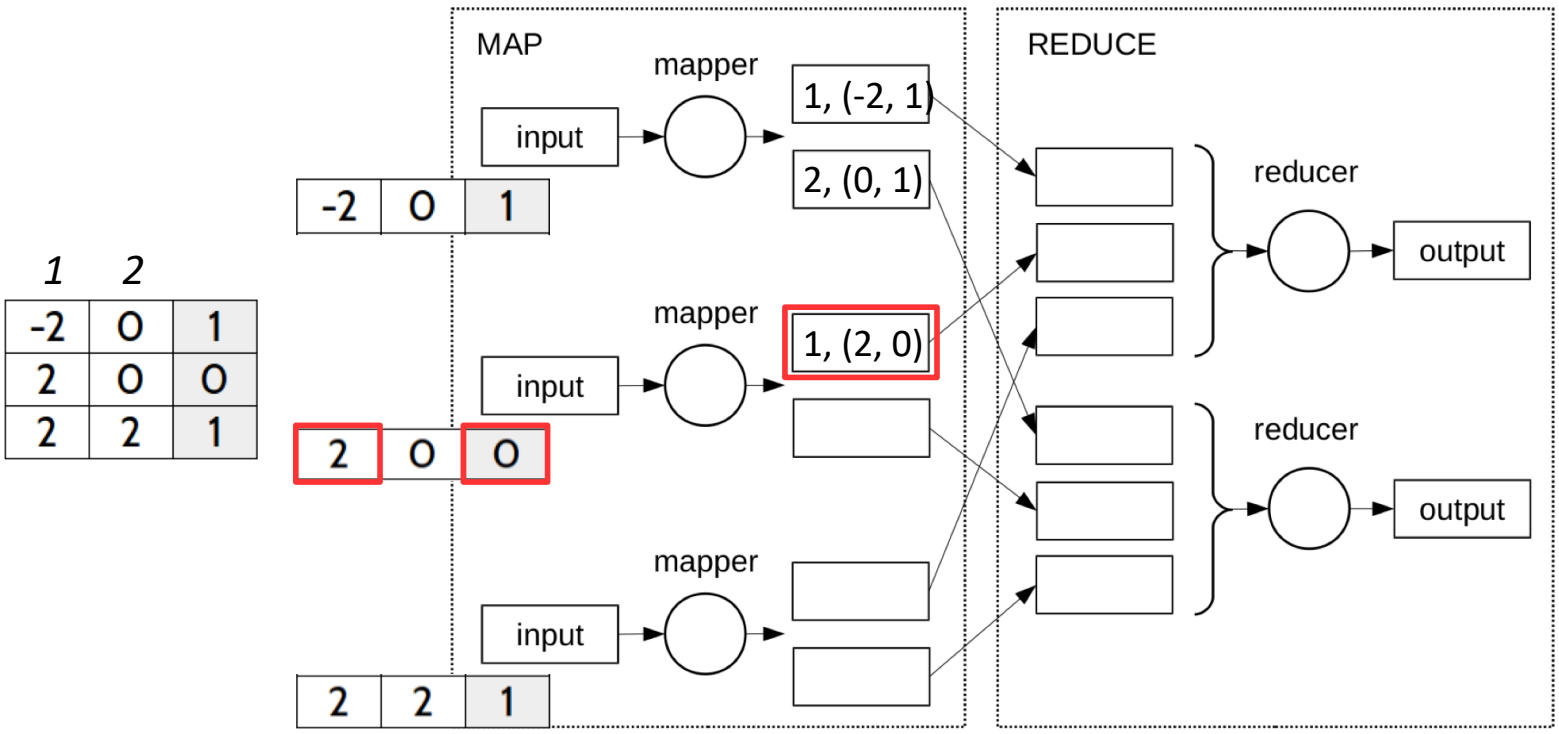


| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |

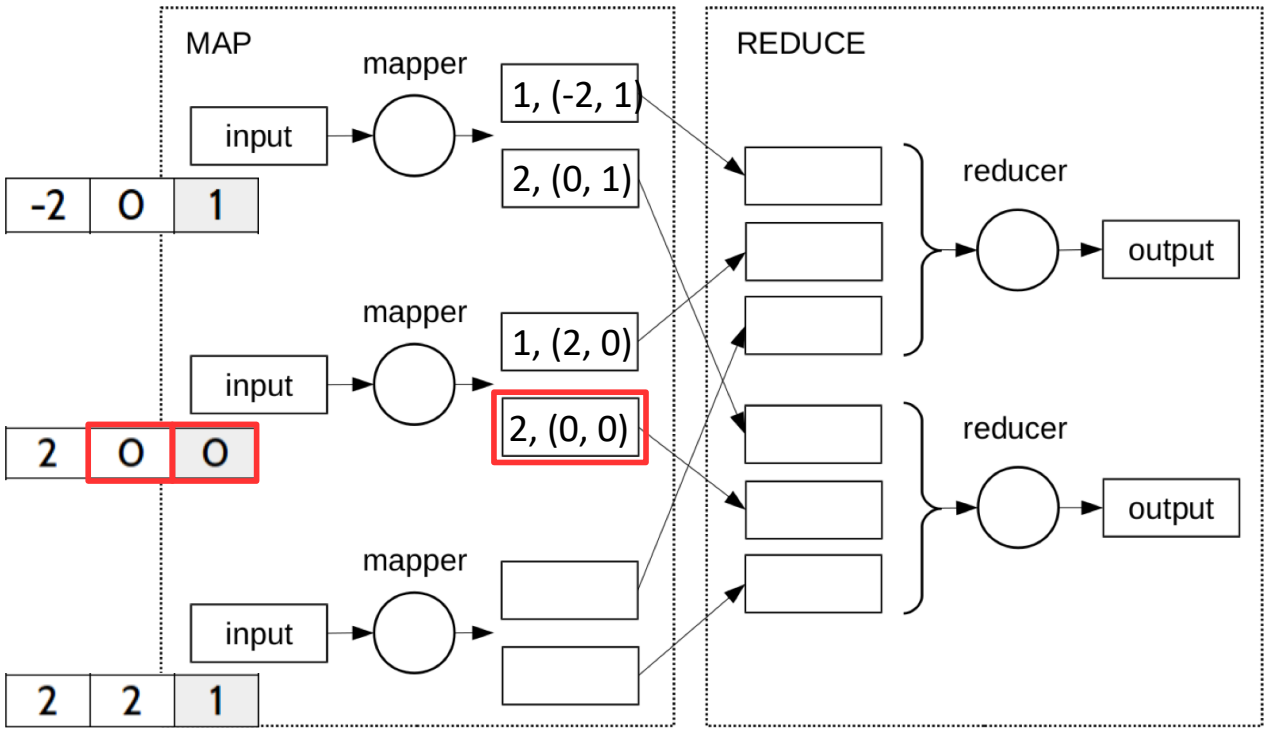


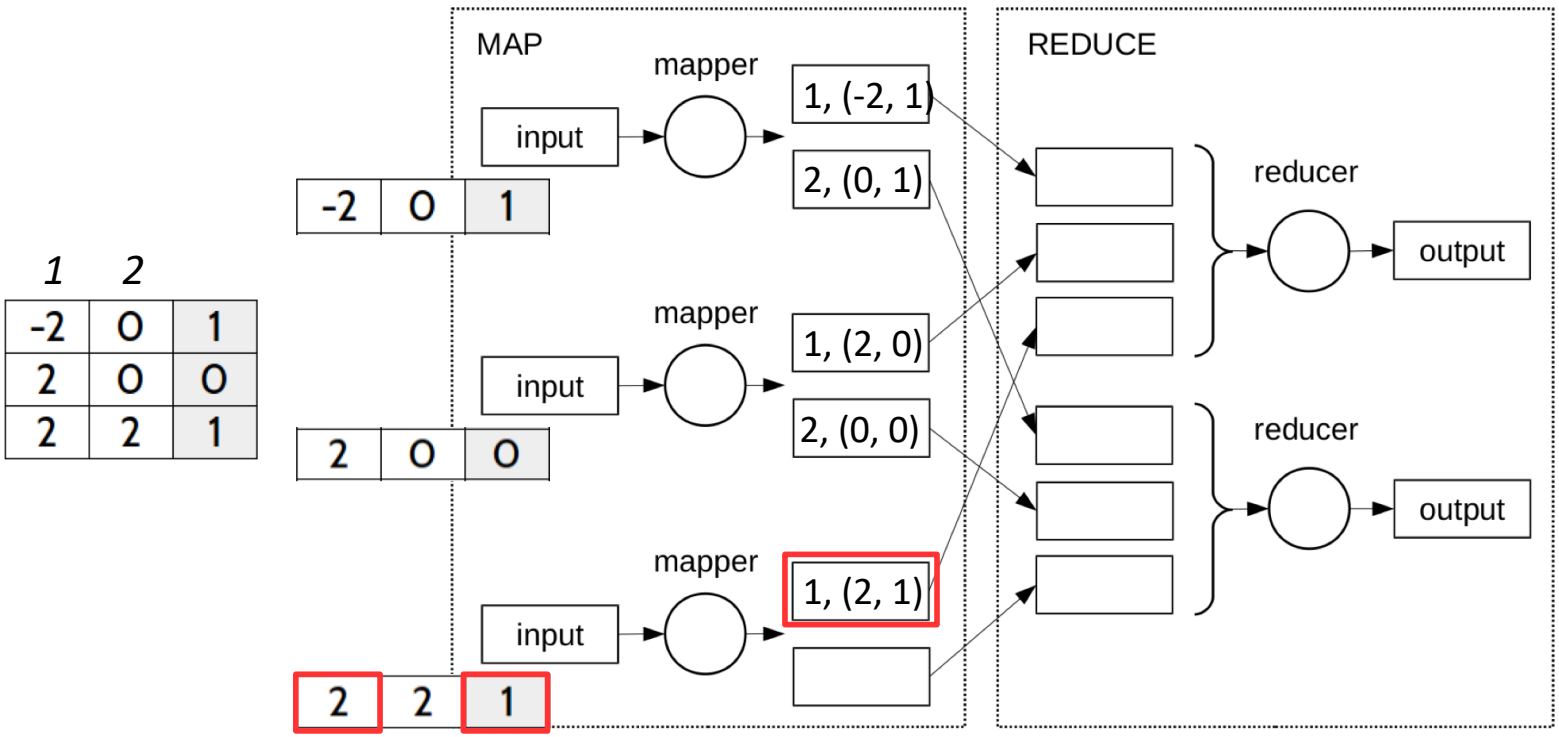
| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |



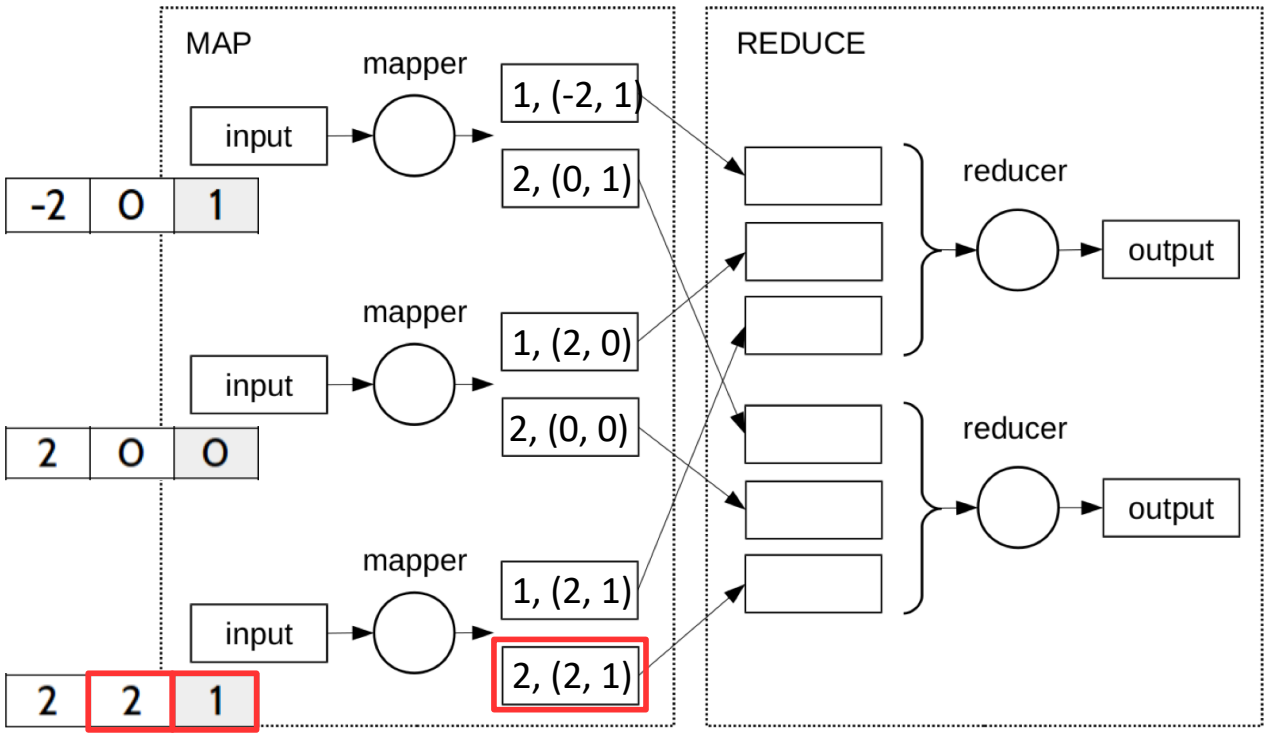

1 2
-2 0 1
2 0 0
2 2 1
MAP
mapper
input
-2 0 1
1, (-2, 1)
2, (0, 1)
mapper
input
2 0 0
1, (2, 0)
mapper
input
2 2 1
REDUCE
reducer
output
reducer
output

| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |

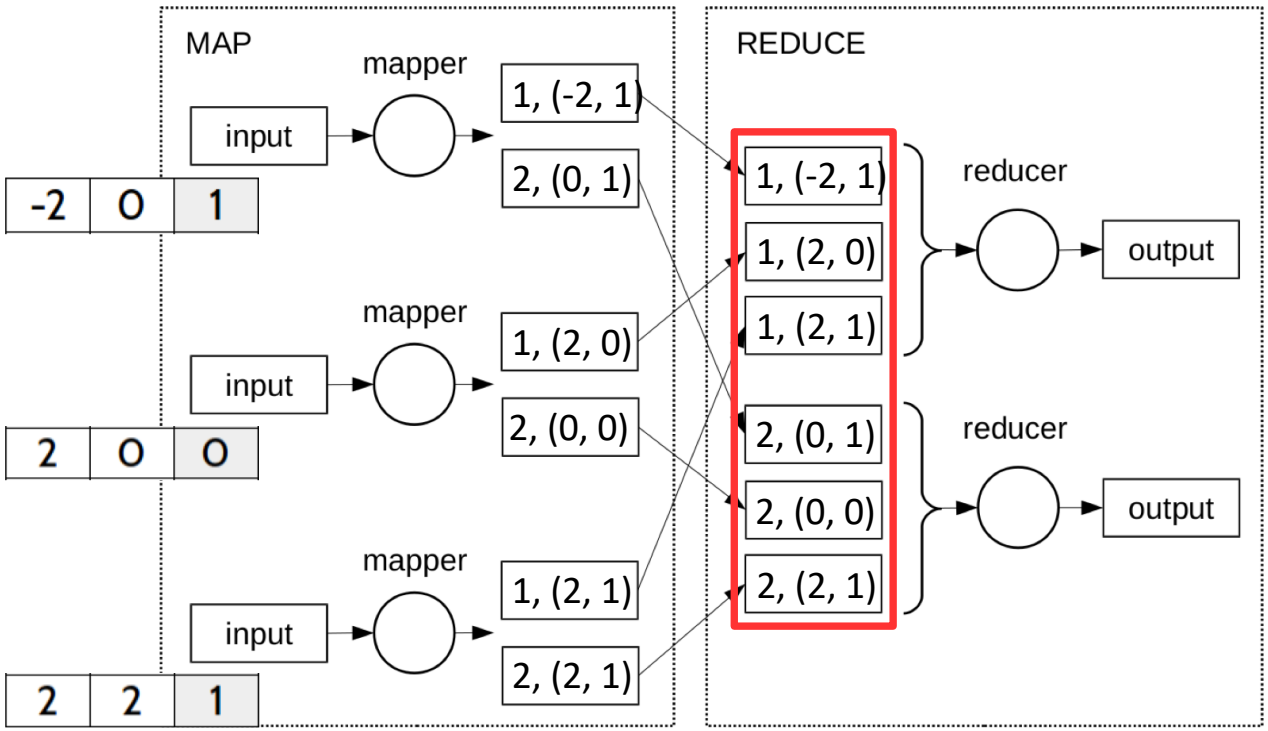


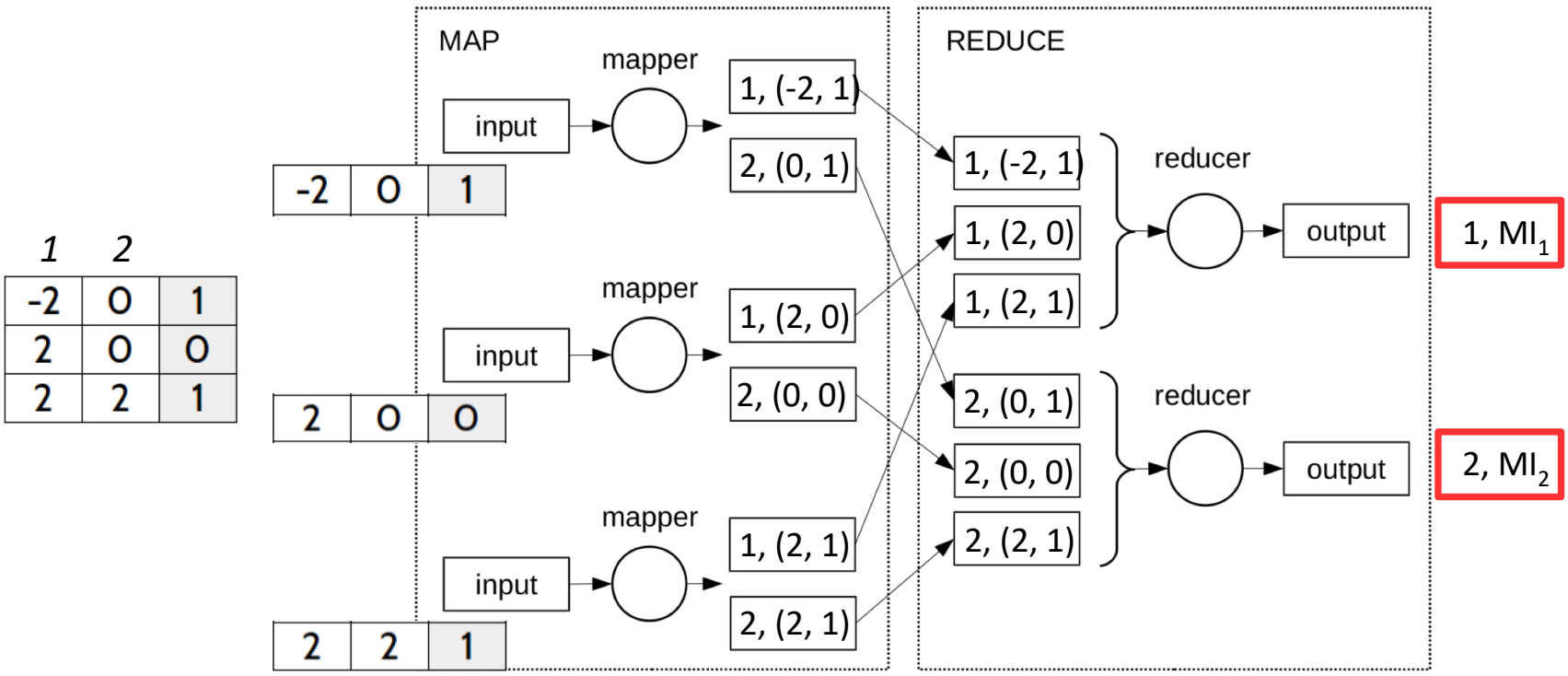

1 2
-2 0 1
2 0 0
2 2 1
MAP
mapper
input
-2 0 1
1, (-2, 1)
2, (0, 1)
mapper
input
2 0 0
1, (2, 0)
2, (0, 0)
mapper
input
2 2 1
1, (2, 1)
REDUCE
reducer
output
reducer
output

| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |



| 1 | 2 | |
|---|---|---|
| -2 | 0 | 1 |
| 2 | 0 | 0 |
| 2 | 2 | 1 |




1
2
-2
0
1
2
0
0
2
2
1
MAP
mapper
input
-2
0
1
1, (-2, 1)
2, (0, 1)
mapper
input
2
0
0
1, (2, 0)
2, (0, 0)
mapper
input
2
2
1
1, (2, 1)
2, (2, 1)
REDUCE
1, (-2, 1)
1, (2, 0)
1, (2, 1)
reducer
output
1, MI1
2, (0, 1)
2, (0, 0)
2, (2, 1)
reducer
output
2, MI2

# HOMEWORK

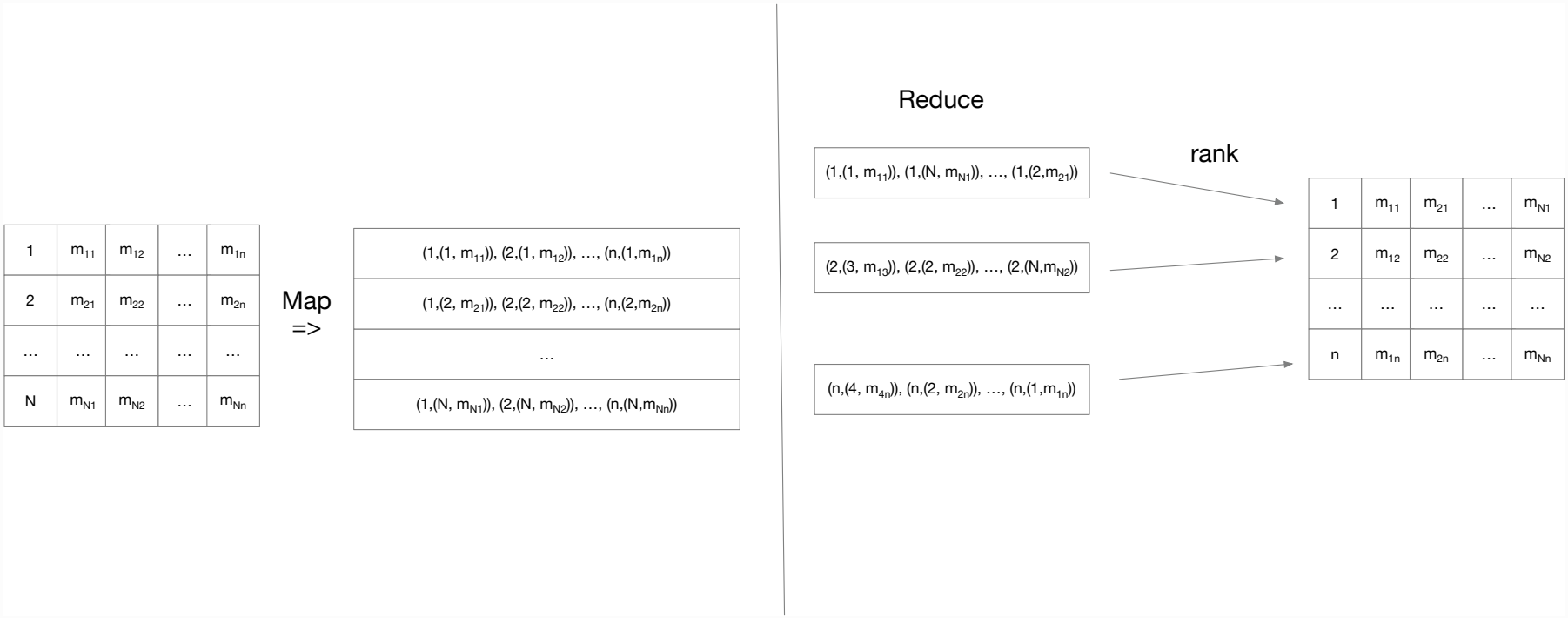
Consider a big rectangular matrix

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1,n} \\ m_{21} & m_{22} & \dots & m_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ m_{N1} & m_{N2} & \dots & m_{N,n} \end{bmatrix}$$

Write a map-reduce pseudo-code to transpose the matrix. Note that

- a pseudo-code description of the map and reduce functions are required
- you can use drawings and text to illustrate and document your code

# SOLUTION


1 m11 m12 … m1n
2 m21 m22 … m2n
… … … … …
N mN1 mN2 … mNn
Map
=>
(1,(1, m11)), (2,(1, m12)), …, (n,(1,m1n))
(1,(2, m21)), (2,(2, m22)), …, (n,(2,m2n))
…
(1,(N, mN1)), (2,(N, mN2)), …, (n,(N,mNn))
Reduce
rank
(1,(1, m11)), (1,(N, mN1)), …, (1,(2,m21))
(2,(3, m13)), (2,(2, m22)), …, (2,(N,mN2))
(n,(4, m4n)), (n,(2, m2n)), …, (n,(1,m1n))
1 m11 m21 … mN1
2 m12 m22 … mN2
… … … … …
n m1n m2n … mNn

# EXAM QUESTION

Let us consider a feature selection problem with a single output target $Y$ and $n$ input variables $X_1,...,X_n$. Suppose that the dataset (made of $N$ observations) is stored in a vertical format, i.e. each line corresponds to a sample

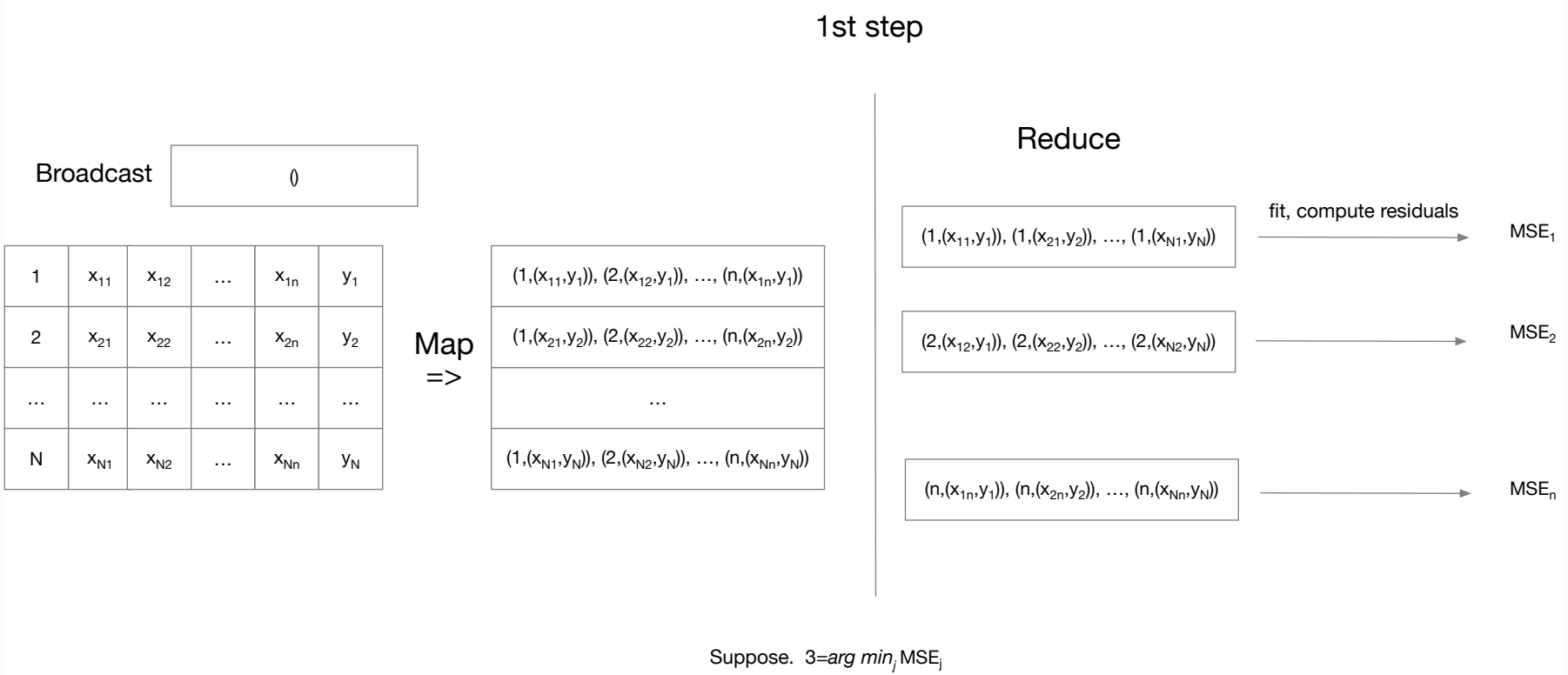
1, $x_{11}$, ..., $x_{1n}$, $y_1$
2, $x_{21}$, ..., $x_{2n}$, $y_2$
....
N, $x_{N1}$, ..., $x_{Nn,}y_N$

Write a map-reduce pseudo-code to perform forward selection where at each step the selection aims to minimize the linear least-squares residual error.

Note that

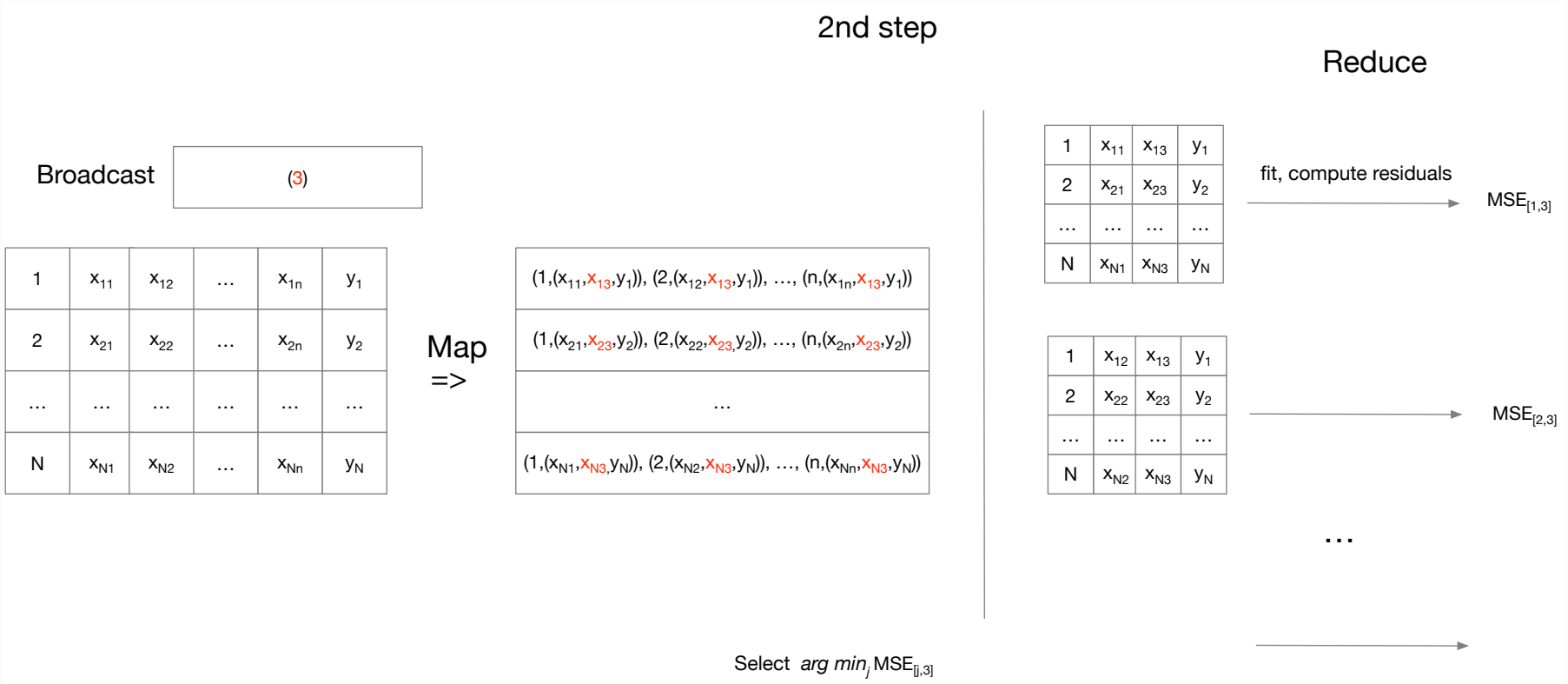
- we make the hypothesis that the number $n$ is huge and that the number of features to select is $k << n$
- we make the hypothesis that each least-squares computation can be computed in a memory-resident manner (i.e. no distributed computation is required for such step)
- a pseudo-code description of the map and reduce functions (notably in terms of input-output behaviour) is required
- you can use drawings and/or text to illustrate and document your code

# SOLUTION: 1ST STEP




Suppose. $3 = arg\ min_j MSE_j$

# SOLUTION: 2ND STEP



# COMPLEXITY ISSUES

# RATIONALE FOR MR PARADIGM

Main motivations

- Decompose a complex task in simpler ones (reducers) that can be executed in main memory
- Reduce latency
- Increase throughput

Strategy:

- 3 steps: Map (data preparation) + communication + Reduce (output production)

Trade-off:

- find a balance between amount of communication and size of single reducer.
- the less communication is used, the higher is the complexity (e.g. size, memory occupation and computing time ) of each single task

No single MR solutions but a spectrum of alternative solutions.

# INPUT-OUTPUT DEPENDENCY AND MR

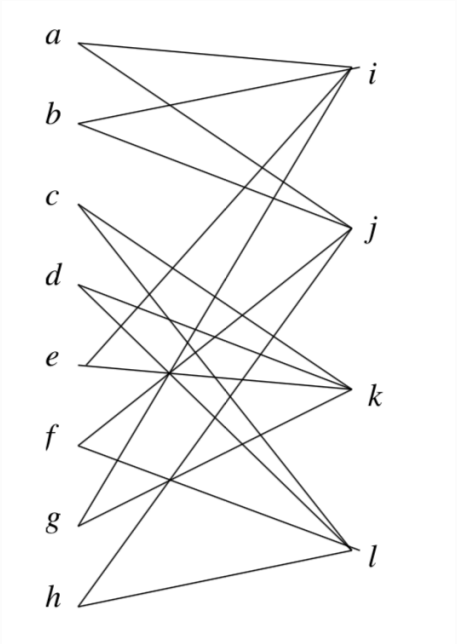
Let us consider a computing problem characterized by

1. a set of p inputs
2. a set of outputs
3. a many-many relationship linking outputs to inputs

**When is a problem solvable by MR**? A problem can be solved by a single MR job if for every output of the problem there is at least a reducer that is assigned all the inputs that are related to that output.

Problems may greatly differ in terms of their input output relationships (e.g. one to one, many to many) and may be described by a graph.

# GRAPH REPRESENTATION




- Number of inputs: p
- Reducer input size: **fan-in** q.
- Mapper replication rate : **fan-out** r.

In this example $p = 8$, $q = 4$, $r = 2$

# REDUCER SIZE AND REPLICATION RATE

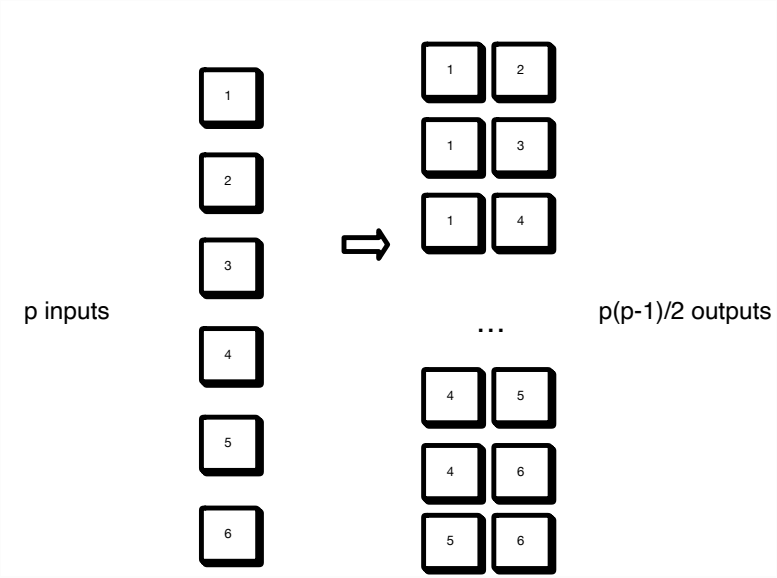
The complexity of a MR algorithm depends on the tradeoff of two quantities [6]:

1. **Mapper replication rate** r: number of key-value pairs produced by Map tasks per input. This quantity has an impact on the **communication cost**.
2. **Reducer input size** q: the number of inputs of reducer (fan-in). This quantity is related to the **complexity** of the reducer and the number of reducers. The larger q, the smaller the number of reducers (and the parallelism) and the higher their complexity (e.g. memory size).

Quantities are typically **inversely proportional** (i.e. qr =constant), i.e. the higher the complexity of the reducer the lower the communication cost and viceversa. For instance we can decrease the communication cost by grouping inputs.

# PAIRED ASSESSMENT

- Very large number p of objects (e.g. images, vectors, variables)
- Compute some statistics related to all pairs (e.g. most similar or most correlated).
- Input matrix of size $D = [p, n]$ and we want to have $DD^T$ of size $[p, p]$. For instance each row correspond to an image of n pixels
- Given p inputs we have $\binom{p}{2} = p(p-1)/2$ outputs (i.e. complexity $O(p^2)$).

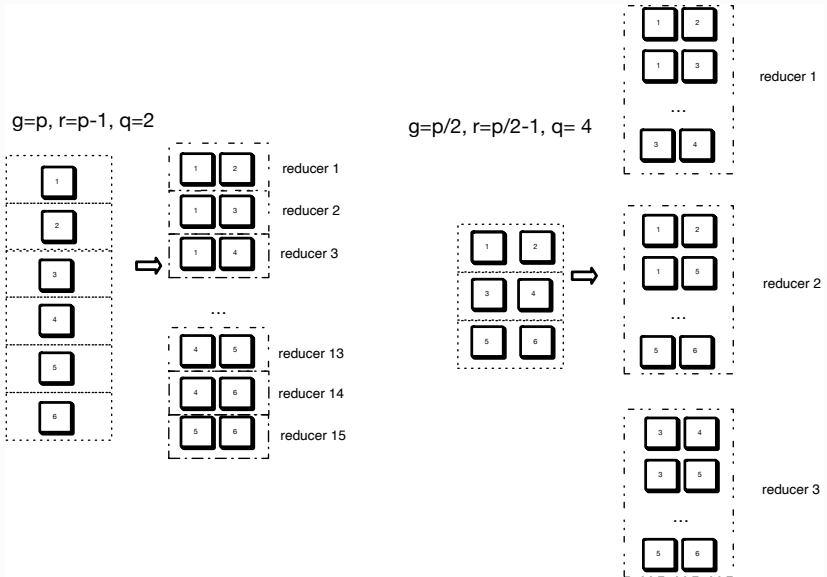


# PAIRED ASSESSMENT BY MR

- We partition the p inputs into $2 \leq g \leq p$ equal-sized groups.
- We have $\binom{g}{2}$ reducers (one for each pair of groups)
- Each reducer gets inputs from two groups, so it will have a number of inputs $q = 2p/g$.
- Each object (belonging to a group) is replicated $r = g - 1$ times.
- Since

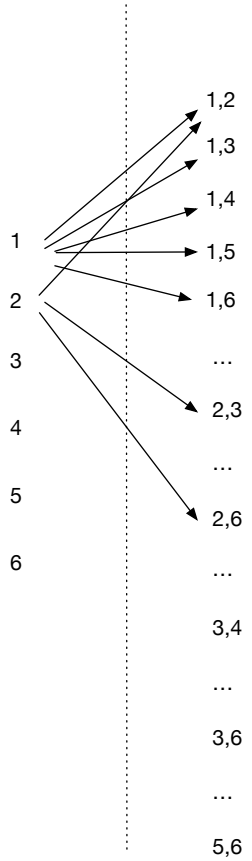
$$q(r+1) = qg = 2p$$

is constant, the reducer size is inversely proportional to the replication size.

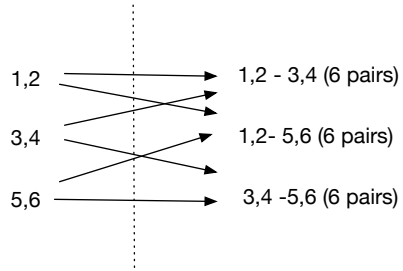


# TRADE-OFF COMMUNICATION VS REDUCER COMPLEXITY

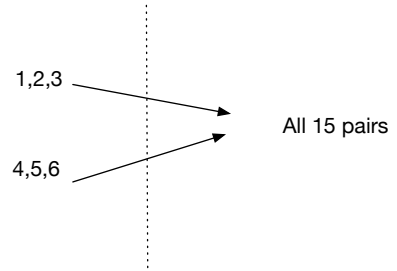
**p=6 inputs**




**g=6** groups (one per element)
replication rate **r=g-1**=5,
**g(g-1)/2**=6*5/2=15 reducers
with **q=2p/g**=2 inputs




**g=3** groups,
replication rate r=2,
3*2/2=3 reducers with
q=2*6/3=4 inputs




**g=2** groups,
replication rate r=1,
2*1/2=1 reducers with
q=2*6/2=6 inputs

**q(r+1)=2p**


Leo Breiman.
Pasting small votes for classification in large databases and on-line.
Machine Learning, 36(1):85–103, Jul 1999.


Tin Kam Ho.
The random subspace method for constructing decision forests.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, Aug 1998.


Gilles Louppe and Pierre Geurts.
Ensembles on random patches.
In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, Machine Learning and Knowledge Discovery in Databases, pages 346–361, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.


J. Maillo, I. Triguero, and F. Herrera.
A mapreduce-based k-nearest neighbor approach for big data classification.
In 2015 IEEE Trustcom/BigDataSE/ISPA, volume 2, pages 167–172, Aug 2015.

M. Parsian.
Data Algorithms: Recipes for Scaling Up with Hadoop and Spark.
O'Reilly Media, 2015.

Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman.
Mining Massive Datasets.
2014.

Claudio Reggiani, Yann-Aël Le Borgne, and Gianluca Bontempi.
Feature selection in high-dimensional dataset using mapreduce.
In Bart Verheij and Marco Wiering, editors, Artificial Intelligence, pages 101–115, Cham, 2018. Springer International Publishing.

J. Wang, W. Liu, S. Kumar, and S. F. Chang.
Learning to hash for indexing big data: A survey.
Proceedings of the IEEE, 104(1):34–57, Jan 2016.