Big Data Learning with Evolutionary Algorithms

Isaac Triguero
School of Computer Science
University of Nottingham
United Kingdom
Isaac.Triguero@nottingham.ac.uk

Mikel Galar
Dept. Automatic and Computation
Public University of Navarre
Spain
mikel.galar@unavarra.es

http://www.cs.nott.ac.uk/~pszit/BigDataCEC2017.html

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

ASAP research group

30 1987-2017 upna

upna Research Institutes

5th June 2017

Outline

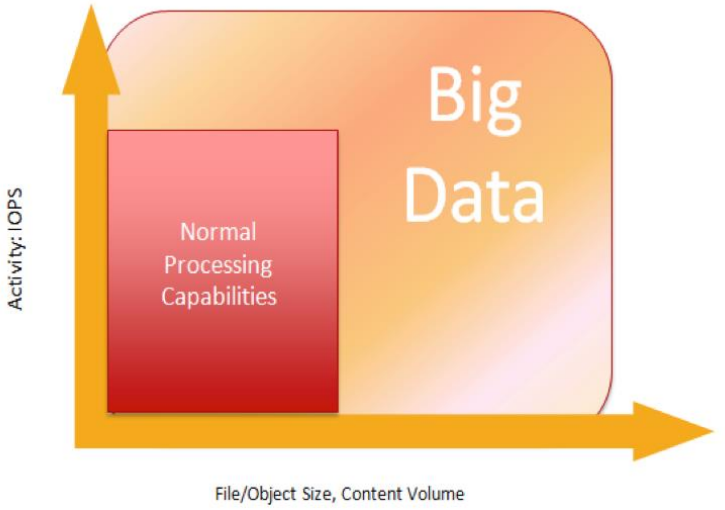
- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions

What is Big Data?

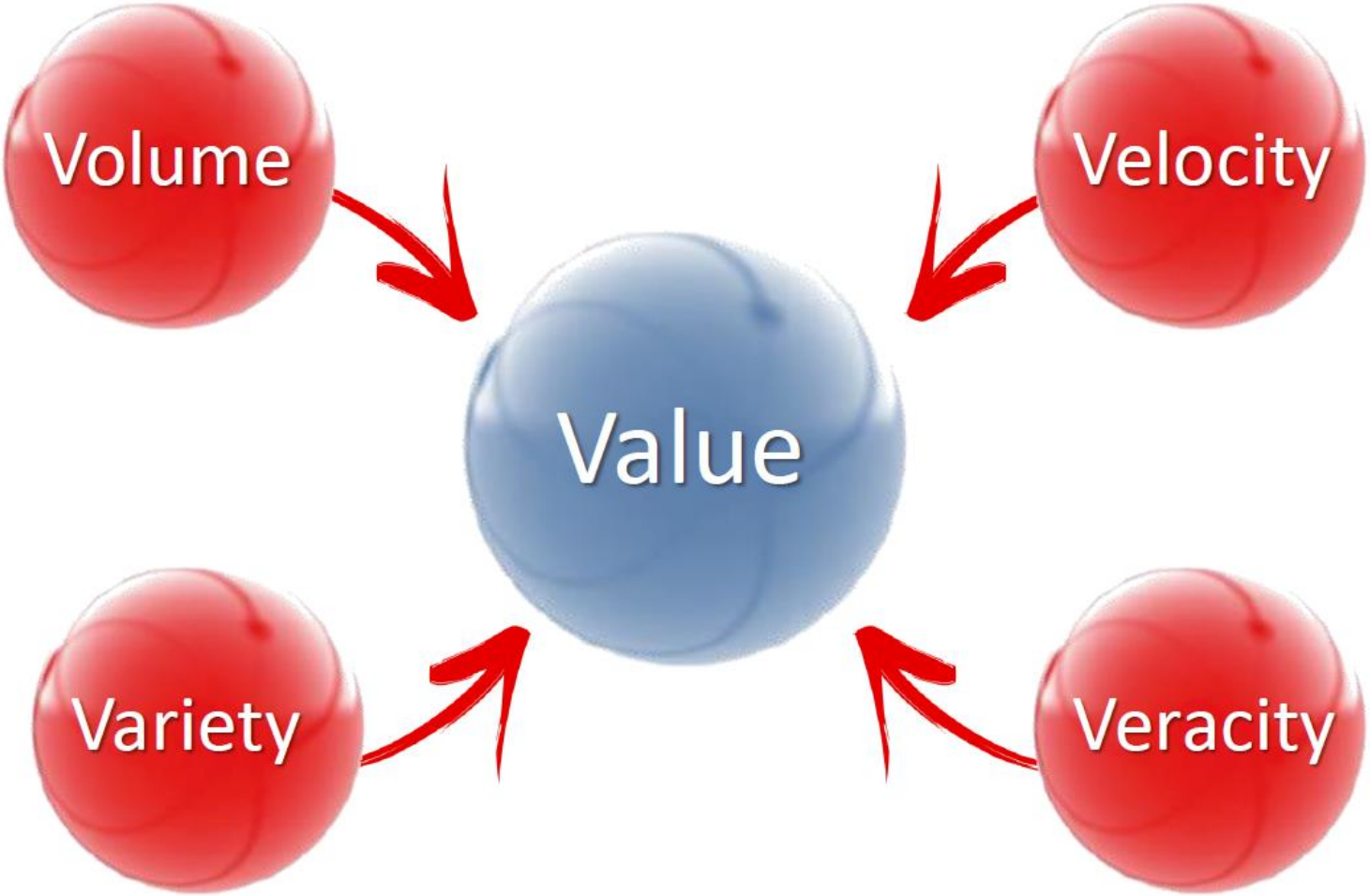
There is no a standard definition!

"*Big Data*" involves data whose volume, diversity and complexity requires new techniques, algorithms and analyses to extract valuable knowledge (hidden) .

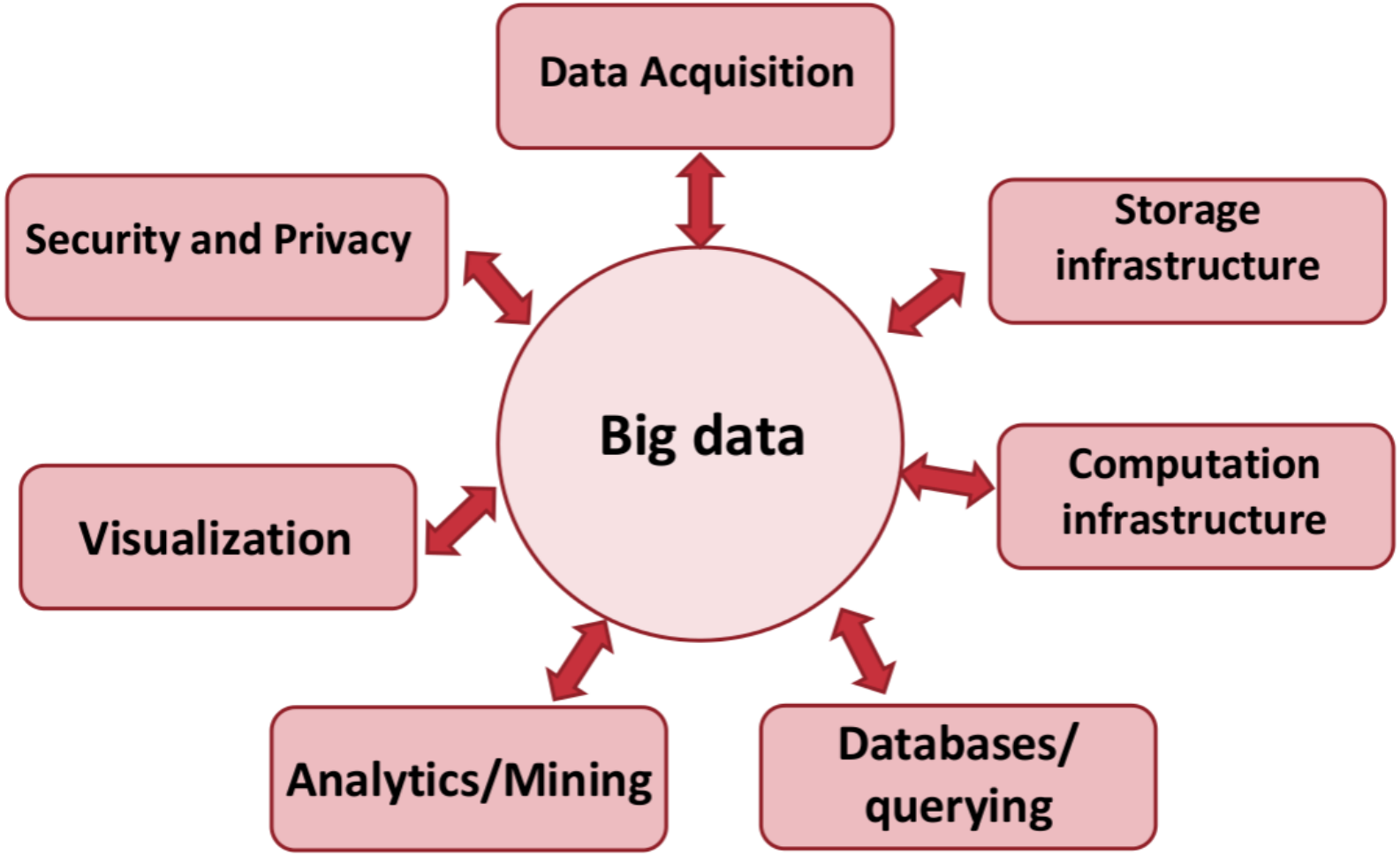
Data Intensive applications



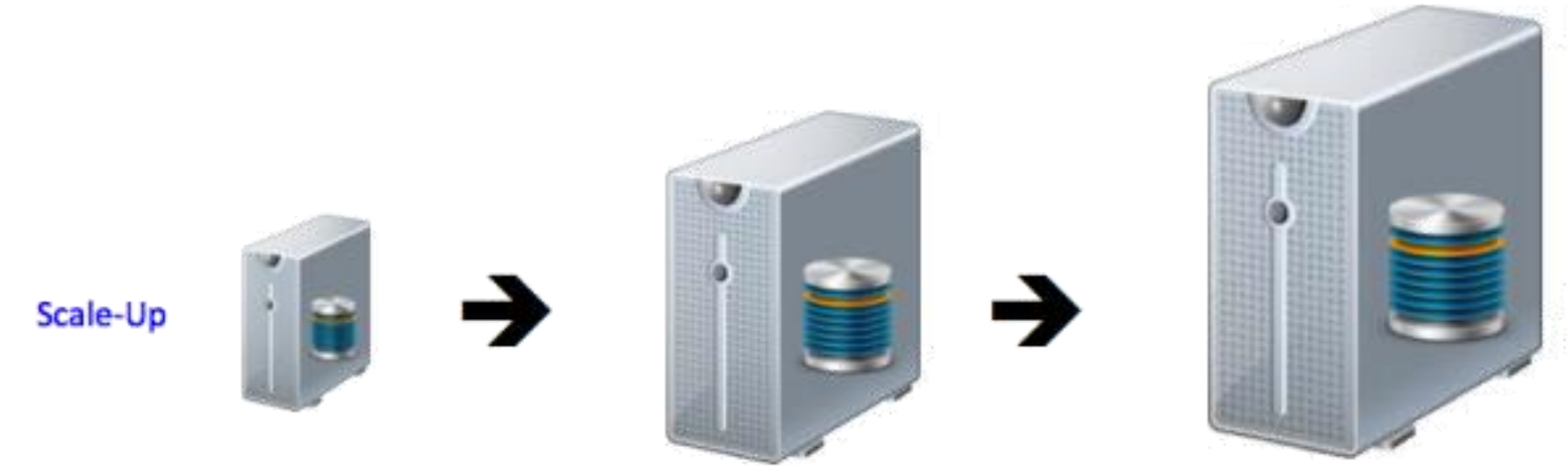
What is Big Data? The 5V's definition



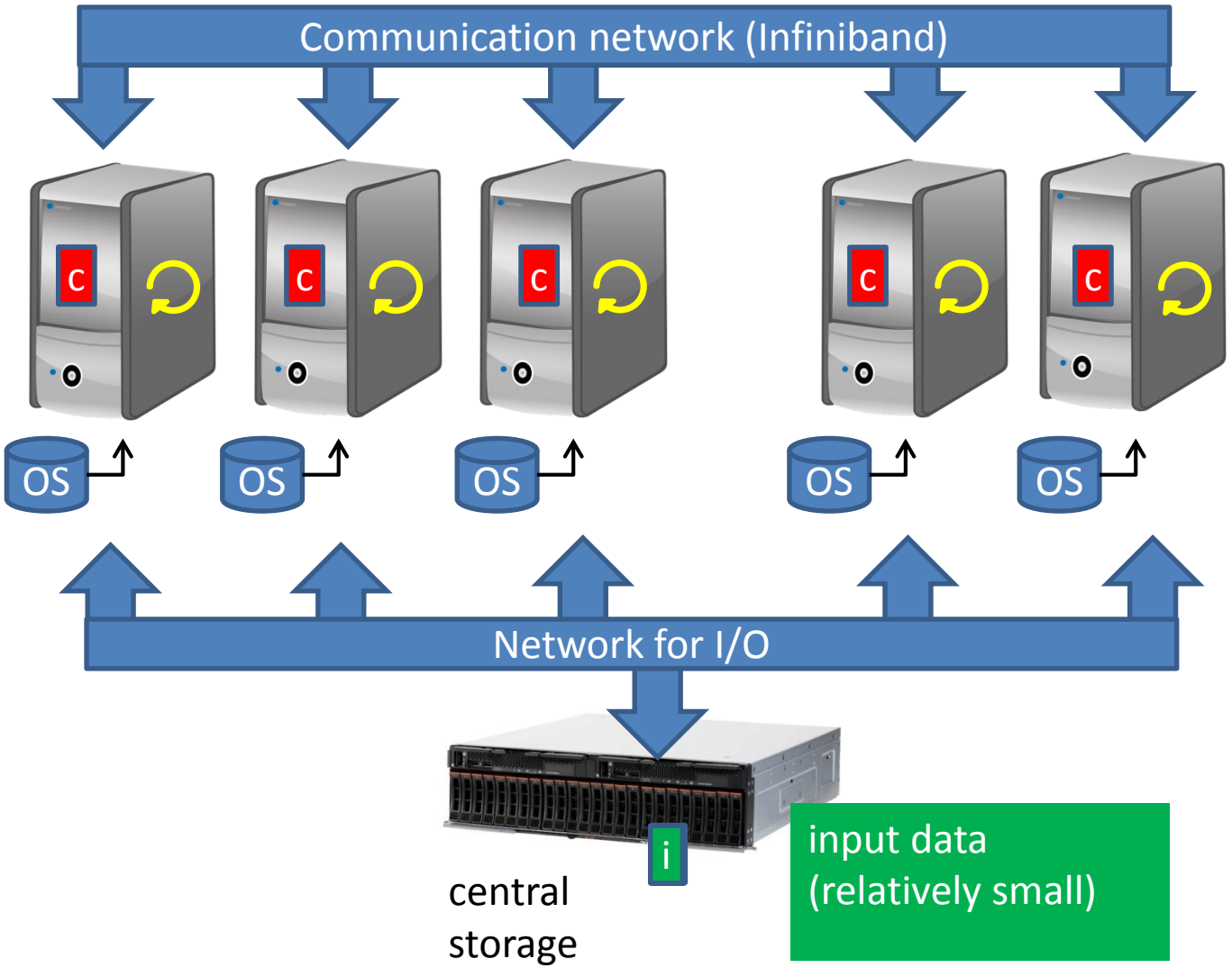
Big data has many faces



How to deal with data intensive applications? **Scale-up vs. Scale-out**



Traditional HPC way of doing things



Lots of communication

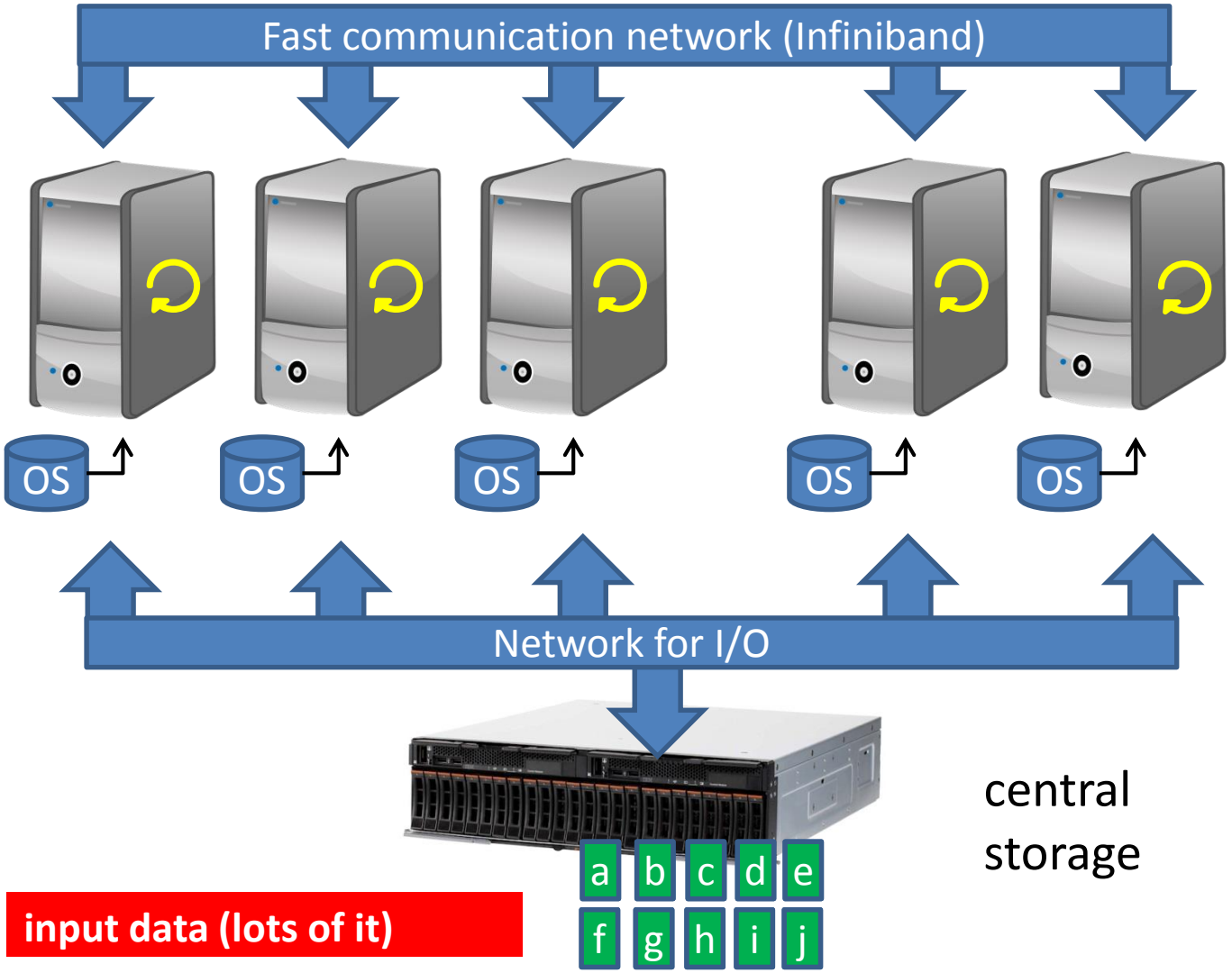
worker nodes (lots of them)

Lots of computations

Limited I/O

Source: Jan Fostier. Introduction to MapReduce and its Application to Post-Sequencing Analysis

Data-intensive jobs



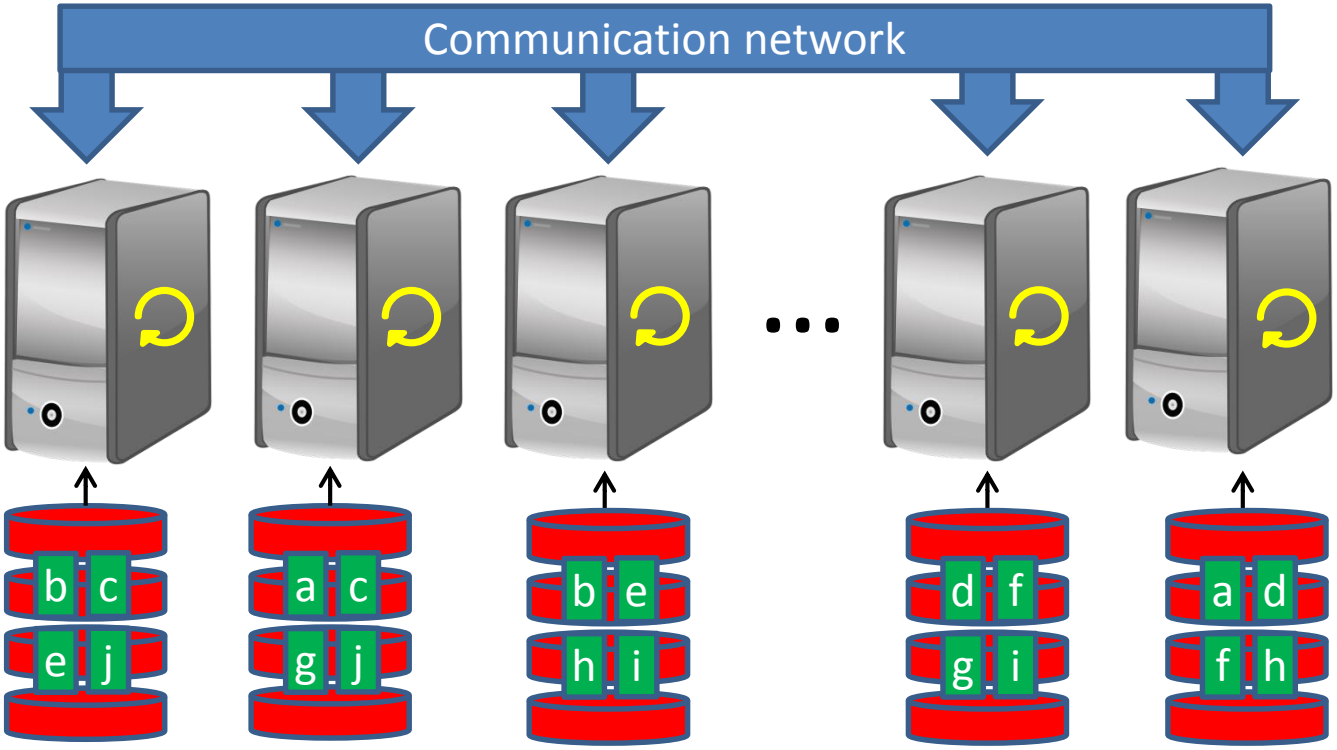
Limited communication

Low compute intensity

doesn't scale

Lots of I/O

Data-intensive jobs



Limited communication

Low compute intensity

input data (lots of it)

Solution: store data on local disks of the nodes that perform computations on that data ("**data locality**")

Distributed systems in Big Data

- **Objective: To apply an operation to all data**
  - One machine cannot process or store all data
    - Data is distributed in a cluster of computing nodes
    - It does not matter which machine executes the operation
    - It does not matter if it is run twice in different nodes (due to failures or stalled nodes)
    - We look for an abstraction of the complexity behind distributed systems
  - **DATA LOCALITY is crucial**
    - Avoid data transfers between machines as much as possible

Distributed systems in Big Data

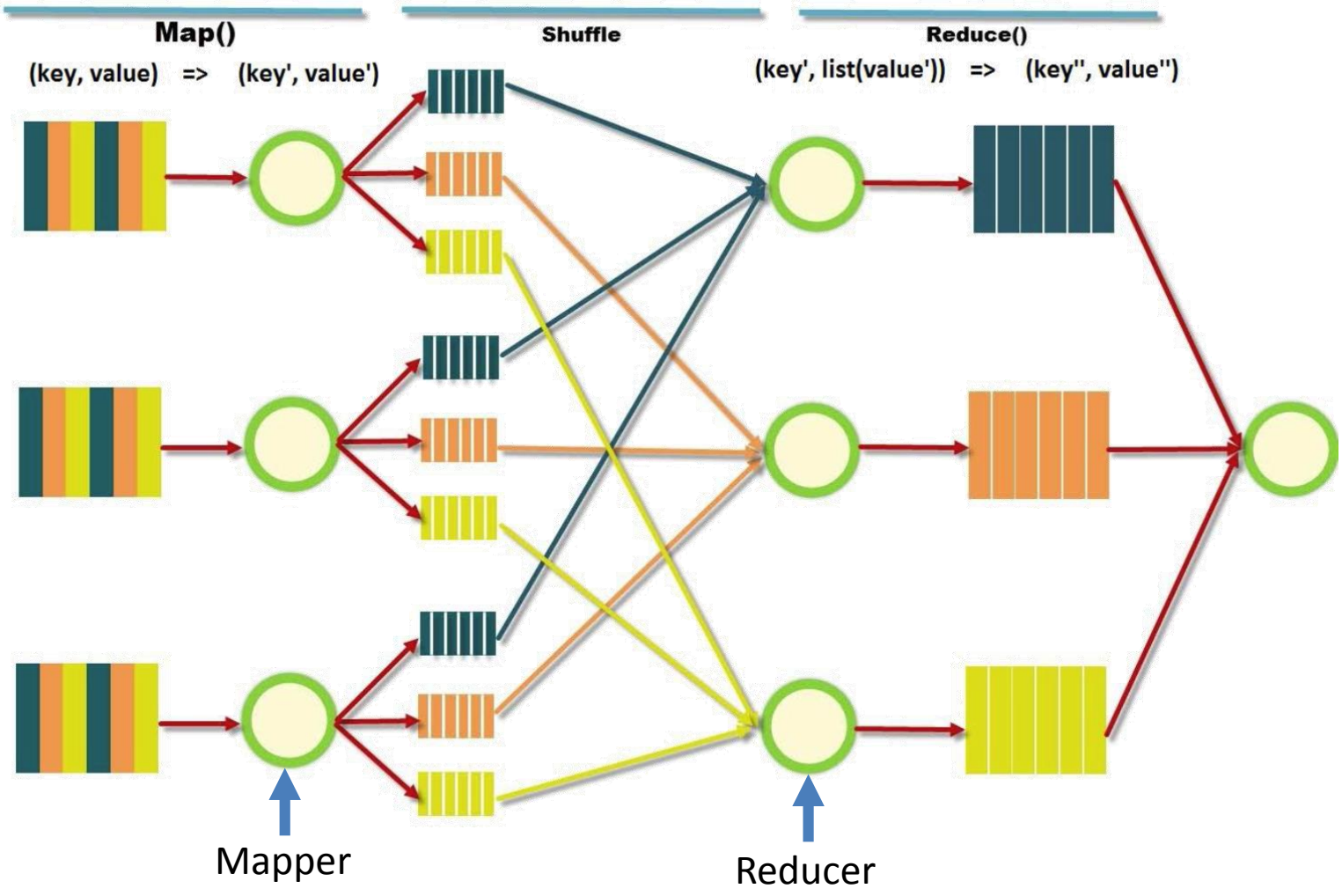
New programming model: **MapReduce**

- *"Moving computation is cheaper than moving computation and data at the same time"*
- **Idea**
  - Data is distributed among nodes (distributed file system)
  - Functions/operations to process data are distributed to all the computing nodes
  - Each computing node works with the data stored in it
  - Only the necessary data is moved across the network

MapReduce

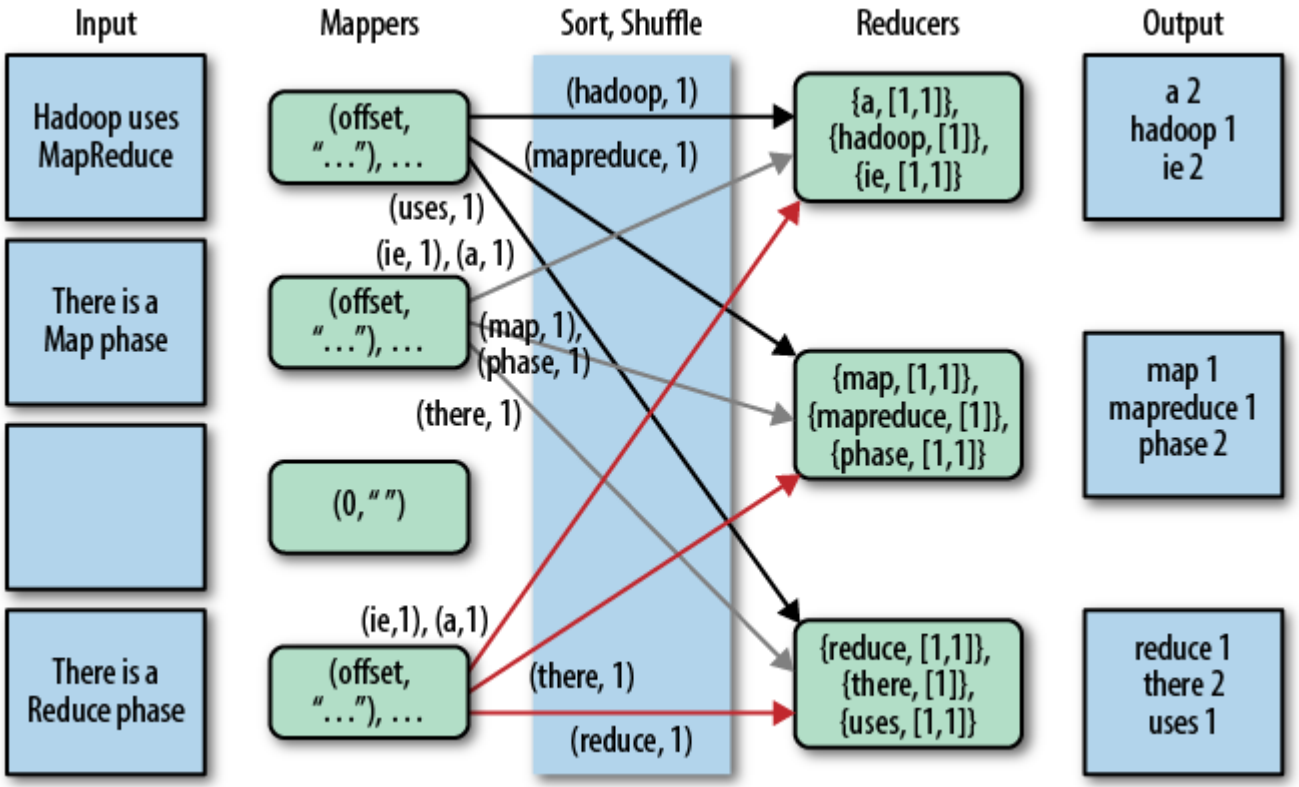
- Parallel Programming model
- **Divide & conquer strategy**
  - **divide**: partition dataset into smaller, independent chunks to be processed in parallel (*map*)
  - **conquer**: combine, merge or otherwise aggregate the results from the previous step (*reduce*)
- Based on **simplicity** and **transparency** to the programmers, and assumes **data locality**
- Becomes popular thanks to the open-source project Hadoop! (Used by Google, Facebook, Amazon, …)

MapReduce: How it works



MapReduce: Example

- **WordCount**



Hadoop

- ***Open source framework for Big Data processing***
  - **Based on** two Works published by Google
    - **Google File System** (GFS)[Ghe03]
    - **MapReduce** algorithm[Dea04]
  - **Composed of**
    - **Hadoop Distributed File System** (HDFS) ➔ Storage
    - **Implementation of the MapReduce algorithm** ➔ Processing

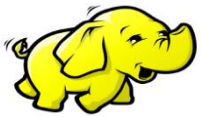
[Ghe03] S. Ghemawat, H. Gobioff, S.-T. Leung. The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). ACM, New York, NY, USA, 29-43. 2003
[Dea04] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10. 2004.

Hadoop

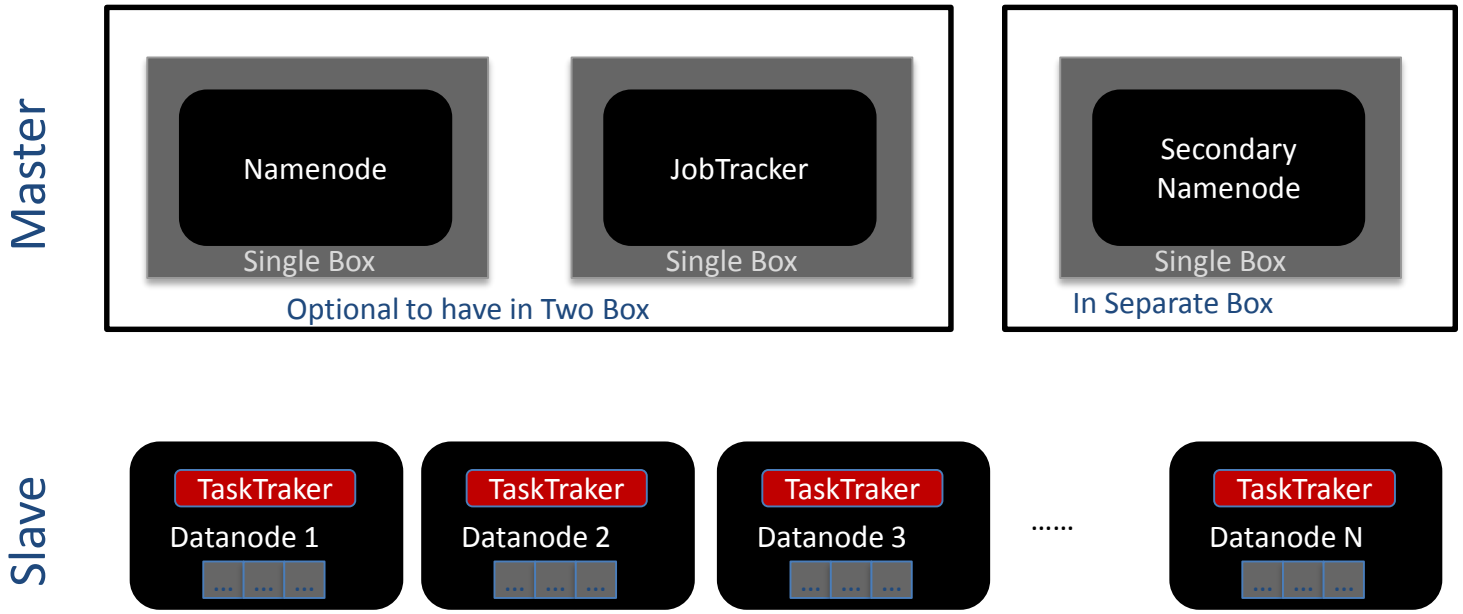
- Hadoop is:
  - An **open-source** framework written in Java
  - Distributed storage of very large data sets (Big Data)
  - Distributed processing of very large data sets

- This framework consists of a **number of modules**
  - *Hadoop Common*
  - ***Hadoop Distributed File System (HDFS)***
  - *Hadoop YARN* – resource manager
  - ***Hadoop MapReduce*** – programming model

Hadoop: A master/slave architecture



- **Master**: NameNode, JobTracker
- **Slave**: {DataNode, TaskTraker}, ..., {DataNode, TaskTraker}



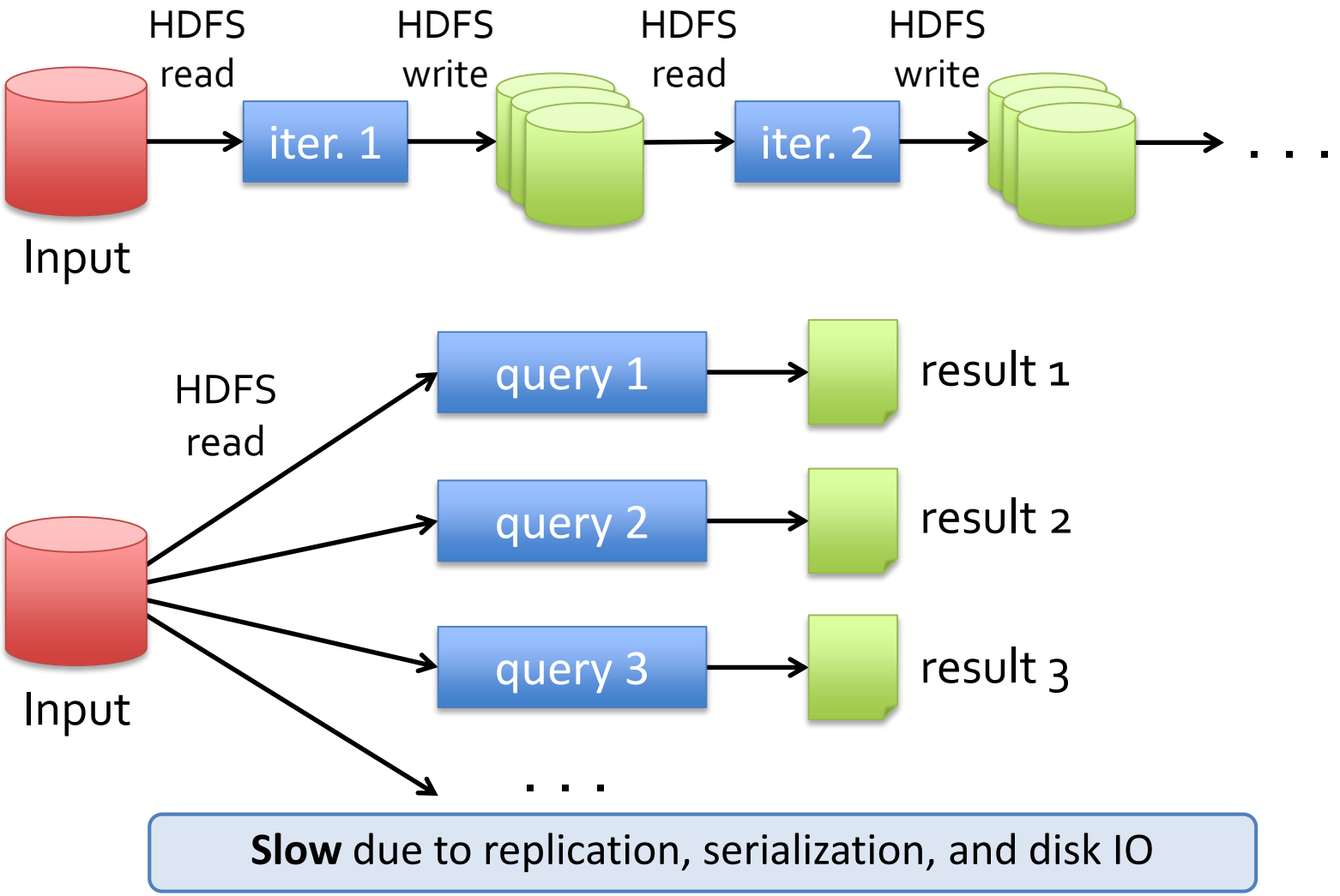
Distributed File System: HDFS

- **HDFS – Hadoop Distributed File System**
  - Distributed File System written in Java
  - Scales to clusters with thousands of computing nodes
    - Each node stores part of the data in the system
  - Fault tolerant due to data replication
  - Designed for big files and low-cost hardware
    - GBs, TBs, PBs
  - Efficient for read and append operations (random updates are rare)
  - High throughput (for bulk data) more important than low latency

Hadoop MapReduce: Main Characteristics

- **Automatic parallelization:**
  - Depending on the size of the input data ➔ there will be multiple MAP tasks!
  - Depending on the number of Keys <k, value> ➔ there will be multiple REDUCE tasks!
- **Scalability:**
  - It may work over every data center or cluster of computers.
- **Transparent for the programmer**
  - Fault-tolerant mechanism.
  - Automatic communications among computers

Data Sharing in Hadoop MapReduce

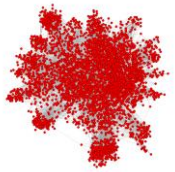


Paradigms that do not fit with Hadoop MapReduce

- **Directed Acyclic Graph (DAG) model:**
  - The DAG defines the dataflow of the application, and the vertices of the graph defines the operations on the data
- **Graph model:**
  - More complex graph models that better represent the dataflow of the application
  - Cyclic models -> Iterativity.
- **Iterative MapReduce model:**
  - An extented programming model that supports iterative MapReduce computations efficiently

New platforms to overcome Hadoop's limitations

GIRAPH (APACHE Project)
(http://giraph.apache.org/)
Iterative graph processing



GPS - A Graph Processing System, (Stanford)
http://infolab.stanford.edu/gps/
Amazon's EC2

Distributed GraphLab (Carnegie Mellon Univ.)
https://github.com/graphlab-code/graphlab
Amazon's EC2

Spark (UC Berkeley)http://spark.incubator.apache.org/research.html

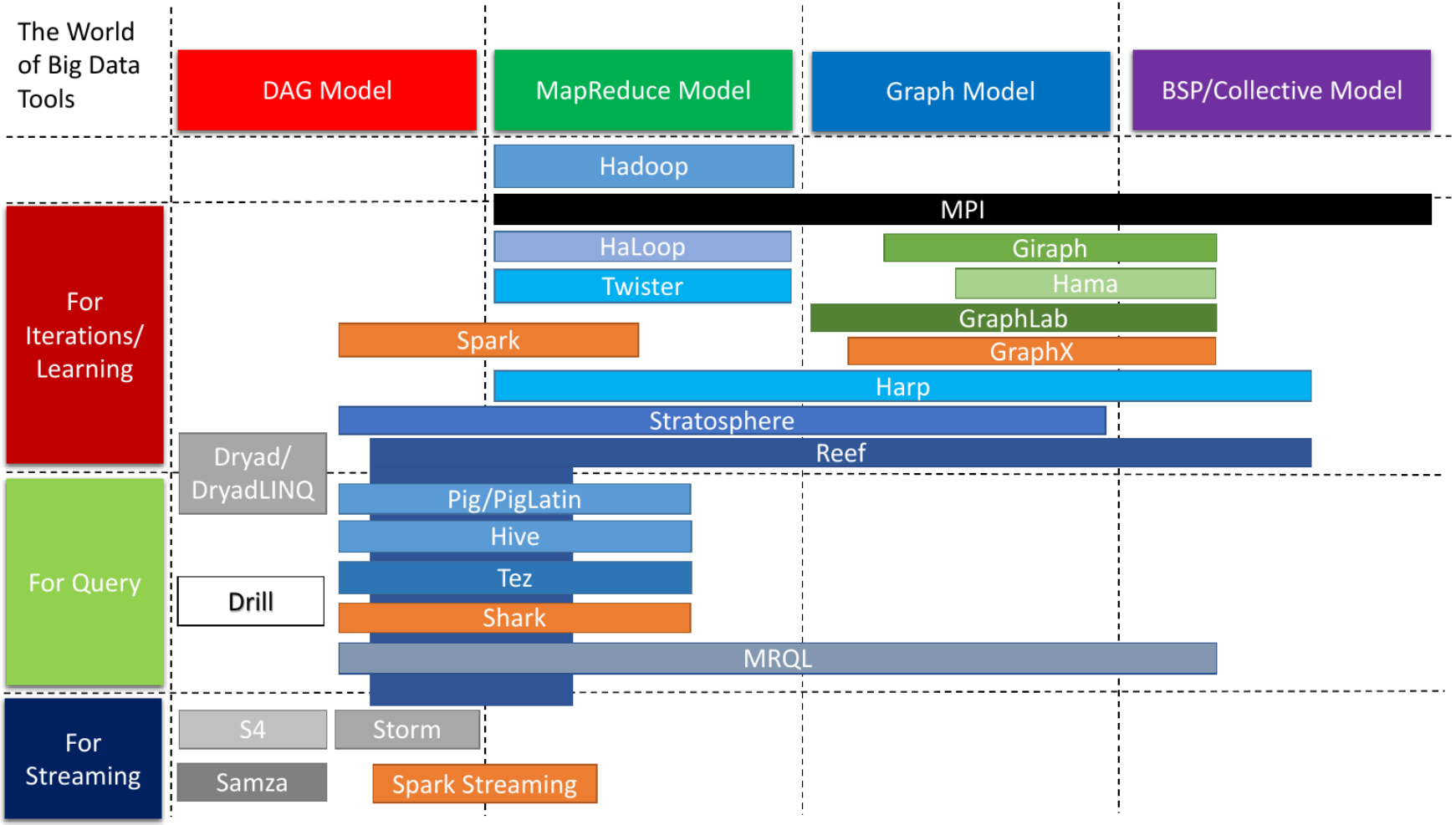
Twister (Indiana University)
http://www.iterativemapreduce.org/
Private Clusters

PrIter (University of Massachusetts Amherst, Northeastern University-China)
http://code.google.com/p/priter/
Private cluster and Amazon EC2 cloud

HaLoop
(University of Washington)
http://clue.cs.washington.edu/node/14
http://code.google.com/p/haloop/
Amazon's EC2

GPU based platforms
Mars
Grex

Big data technologies



What is Spark?

Fast and **Expressive** Cluster Computing Engine Compatible with Apache Hadoop

Up to **10×** faster on disk,
100× in memory

2-5× less code

Efficient

- General execution graphs
- In-memory storage

Usable

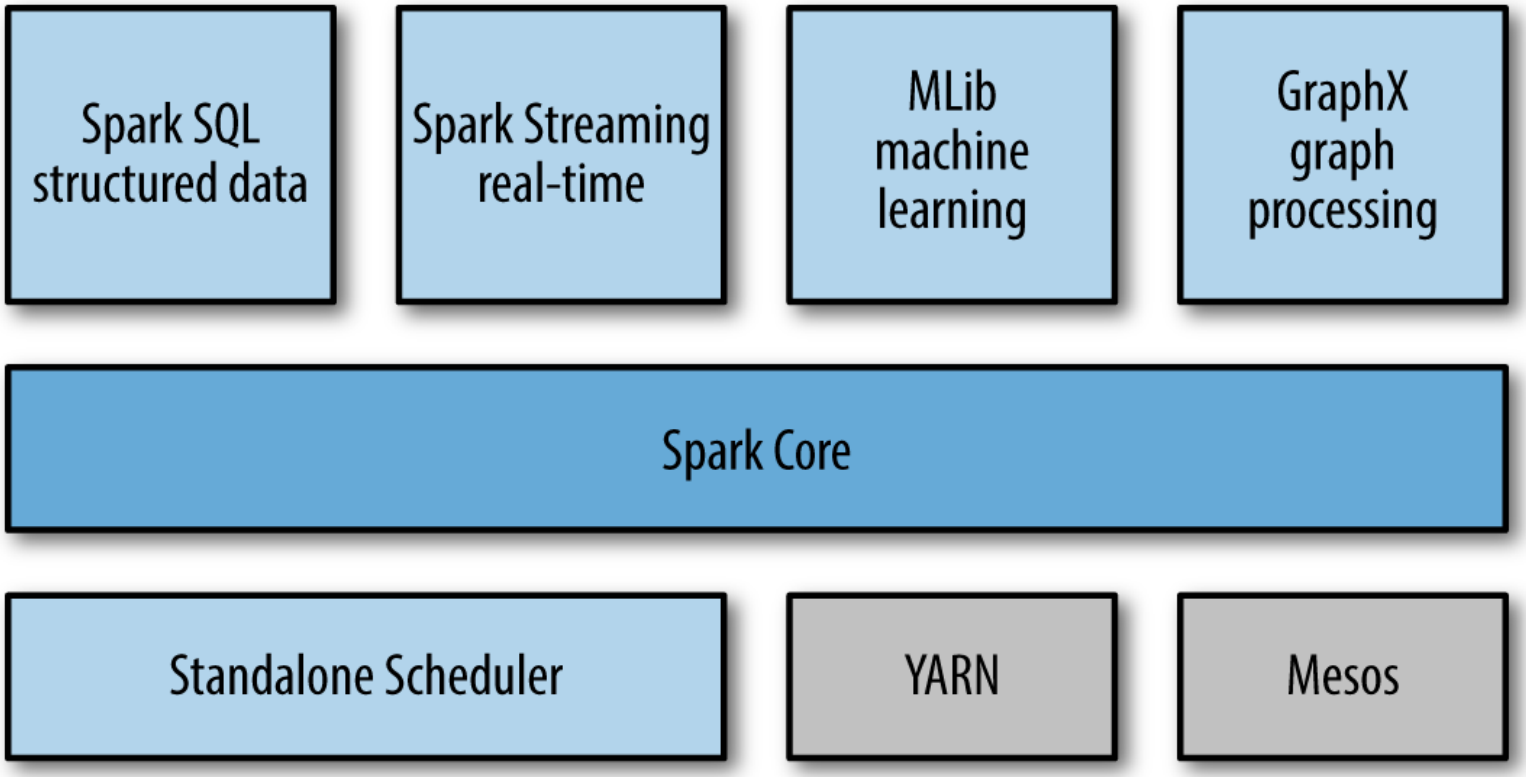
- Rich APIs in Java, Scala, Python
- Interactive shell

What is Spark?

- Data processing engine (only)
- Without a distributed file system
  - Uses other existing DFS
    - HDFS, NoSQL…
    - Hadoop is not a prerequisite

- Works with different cluster management tools
  - Hadoop (YARN)
  - Mesos
  - Standalone mode (included in Spark)

What is Spark?



Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- **Retain the attractive properties of MapReduce**:
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

Initial Solution: augment data flow model with "resilient distributed datasets" (RDDs)

RDDs in Detail

- An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs:
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, Hbase.
- *Can be cached for future reuse*

Operations with RDDs

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage

Transformations (define a new RDD)
map filter sample union groupByKey reduceByKey join cache …

Parallel operations (return a result to driver)
reduce collect count save lookupKey …

Operations with RDDs

- **RDDs – Simple example**

```
>>> lines = sc.textFile("README.md") # Creates an RDD
>>> lines.count() # Counts the number of elements in the RDD
127
>>> lines.first() # First element of the RDD -> 1st line of
README.md
u'# Apache Spark'
```

- Simple wordCount in Spark

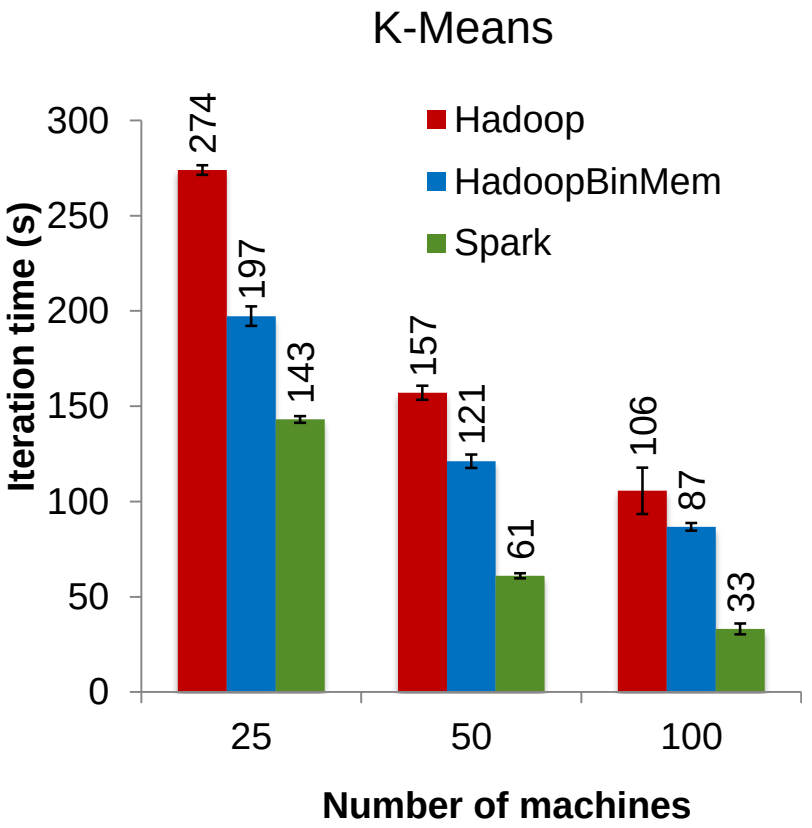
```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
\
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Spark vs. hadoop

Lines of code for K-Means

Spark ~ 90 lines –

Hadoop ~ 4 files, > 300 lines

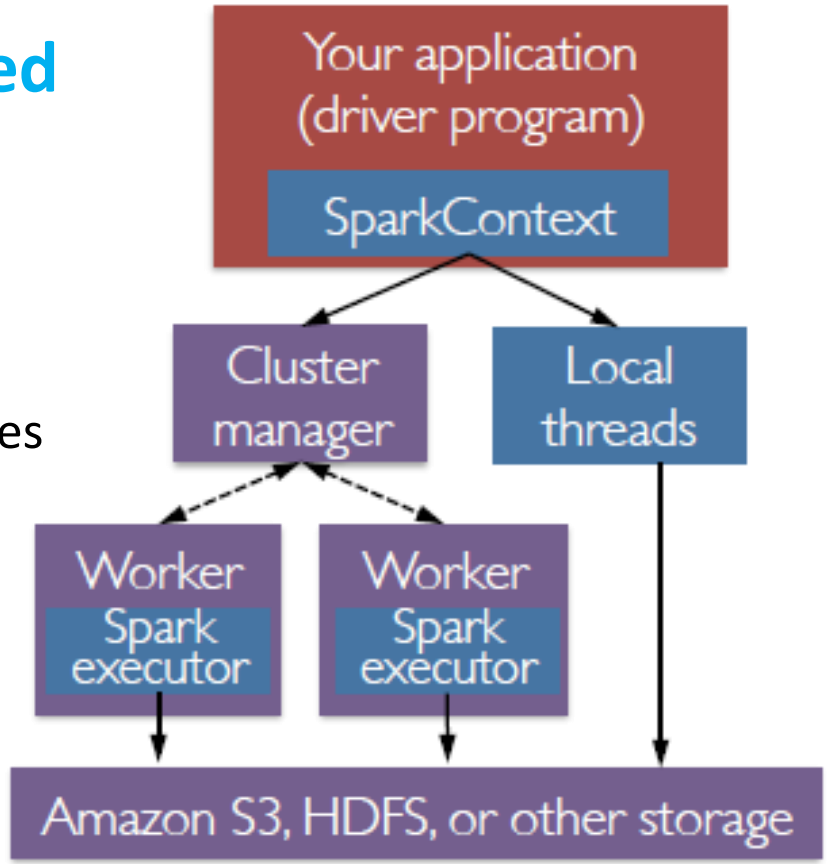


[Zaharia et. al, NSDI'12]

M. Zaharia et al. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**. NSDI 2012.

Spark

- **Driver** and **Workers**
  - A **Spark program is composed of two programs**
    - **Driver program**
    - **Workers program**
      - Executed in the computing nodes
      - Or in local threads
  - RDDs are distributed across the whole cluster



SparkSQL: Datasets y DataFrames

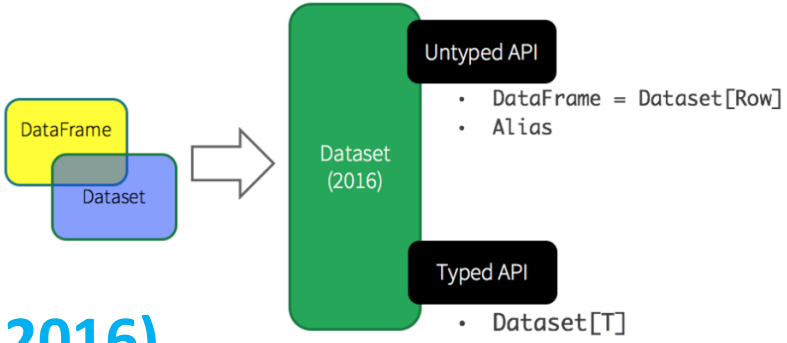
- **Structured APIs**
  - **DataFrames (>= Spark 1.3)**
    - Idea: **RDDs of rows with columns that can be accessed by their names**
    - **Similar to Pandas in Python (dataframes in R)**
    - Avoid Java serialization performed by RDDs
    - API natural for developers familiar with building query plans (SQL)
    - Introduced as a part of Tungsten project
      - Efficient memory management
    - Concept of schema to describe data

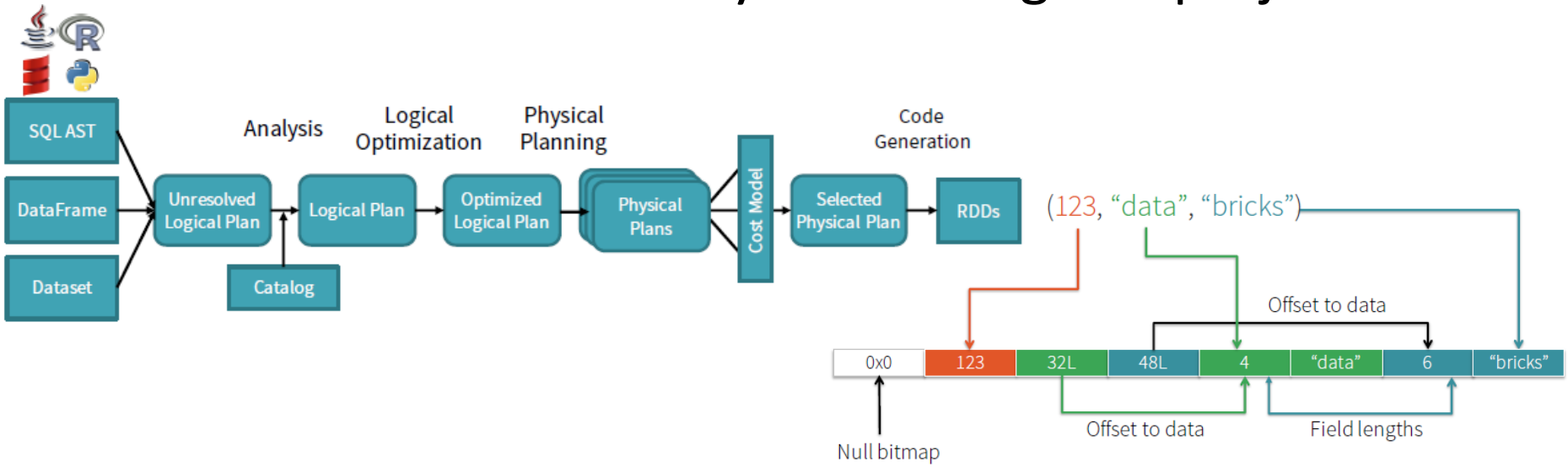
SparkSQL: Datasets y DataFrames

- **Structured APIs**
  - **Datasets (>= Spark 1.6)**
    - Idea: Strongly typed RDDs
    - **Functional transformations (map, flatMap, filter)**
    - **Best of both RDDs and DataFrames**
      - Object-oriented programming
      - Compile-time type safety
      - Catalyst optimization
      - Tungsten off-heap memory optimization
    - Only for Scala and Java

SparkSQL: Datasets y DataFrames

- **Structured APIs**
  - **DataFrames and Datasets**
  - **Fused in Spark 2.0 (November 2016)**
    - A DataFrame is just a Dataset of Rows: Dataset[Row]
  - Both make us of Catalyst and Tungsten projects





SparkSQL: Datasets y DataFrames

- **Structured APIs in Spark**
  - Analysis of the **reported errors** before a job is executed

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Flink

https://flink.apache.org/

Flink | Overview | Features | Downloads | FAQ | Quickstart | Documentation

Apache Flink is an open source platform for distributed stream and batch data processing.

Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams.

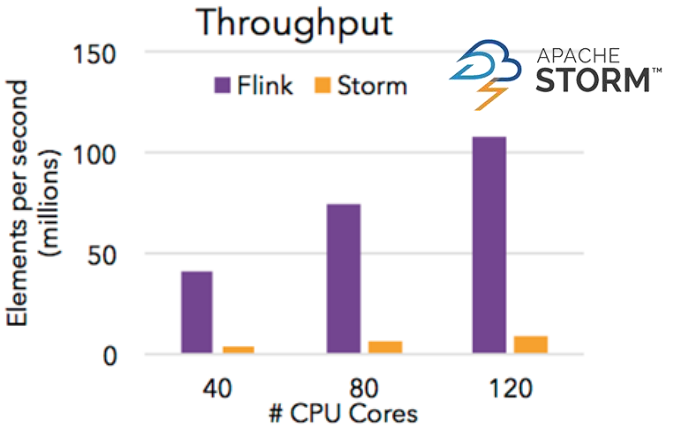
Flink includes **several APIs** for creating applications that use the Flink engine:

1. DataStream API for unbounded streams embedded in Java and Scala, and
2. DataSet API for static data embedded in Java, Scala, and Python,
3. Table API with a SQL-like expression language embedded in Java and Scala.

Flink also bundles **libraries for domain-specific use cases**:

1. CEP, a complex event processing library,
2. Machine Learning library, and
3. Gelly, a graph processing API and library.

You can **integrate** Flink easily with other well-known open source systems both for data input and output as well as deployment.



Batch on Streaming

Batch processing applications run efficiently as special cases of stream processing applications.

Streaming First

High throughput and low latency stream processing with exactly-once guarantees.

APIs, Libraries, and Ecosystem

DataSet, DataStream, and more. Integrated with the Apache Big Data stack.

2001-2010

2010-2017

Big Data: Technology and Chronology

The Apache Software Foundation

2001
3V's Gartner
Doug Laney

Google
MapReduce

Hadoop Ecosystem

2009-2013 Flink
TU Berlin
Flink Apache (Dec. 2014) Volker Markl

2004
MapReduce
Google
Jeffrey Dean

Big Data

Flink

hadoop

YAHOO!

2010 Spark
U Berckeley
Apache Spark
Feb. 2014
Matei Zaharia

Spark

2008
Hadoop
Yahoo!
Doug Cutting

2010-2017:

Big Data Analytics: Mahout, MLLib, …

Hadoop Ecosystem

Applications New Technology

Big Data Ecosystem

Apache MESOS™

Cluster management

kubernetes

Cluster management

Pig

Scripting para MapReduce

Sqoop

Hadoop to RDBMS

Data serialization

SQL

NoSQL Database

Streaming data

DAG execution

ZooKeeper

Coordinador

Streaming data

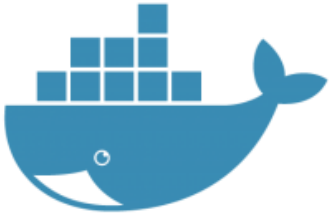
NoSQL on HDFS

Kafka

Gestión de fuentes de datos

Zepelin

Web-based notebook



Docker

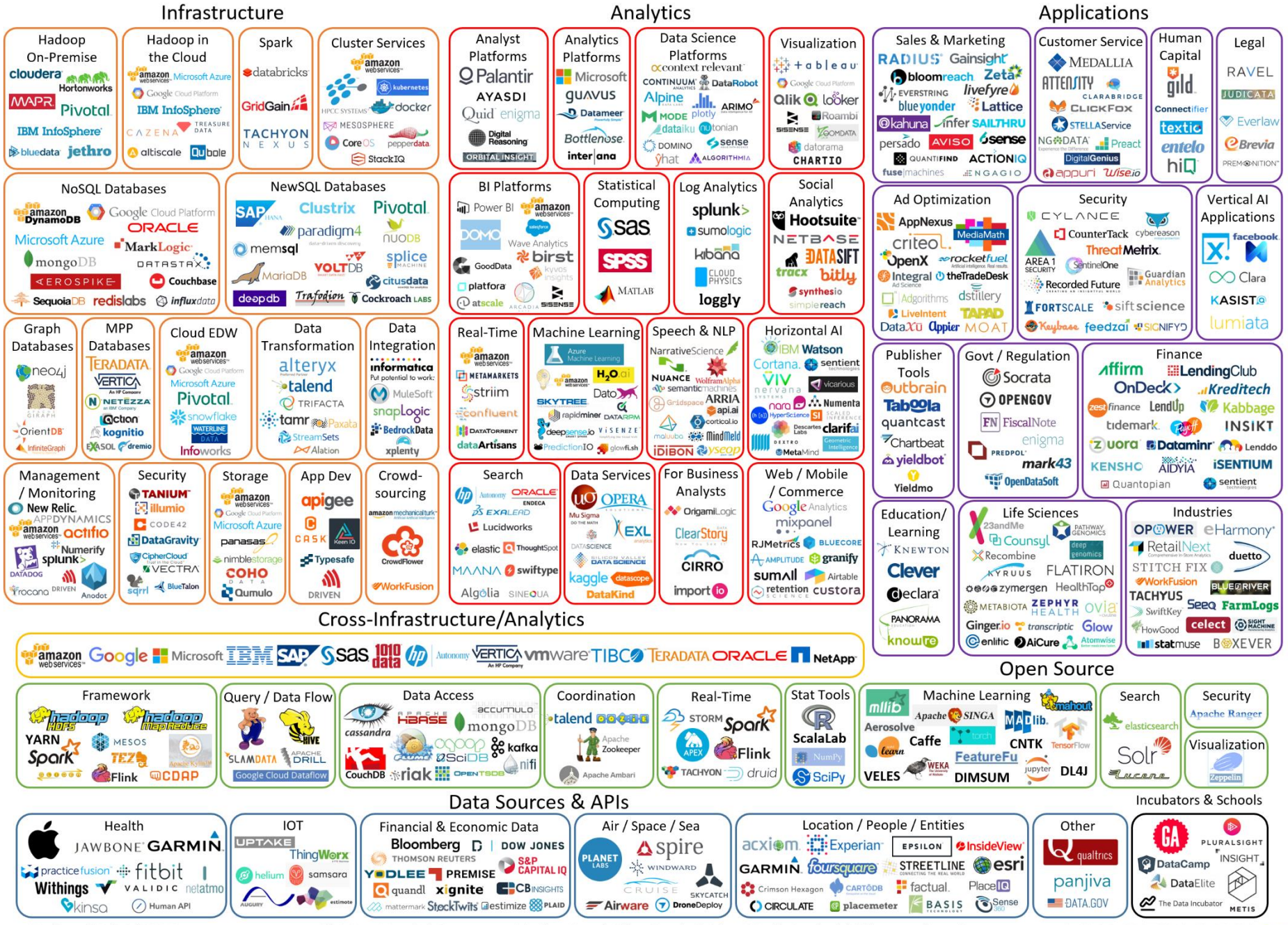
Containers

Log data

Impala

Interactive SQL

Big Data Landscape 2016 (Version 3.0)



Last Updated 3/23/2016

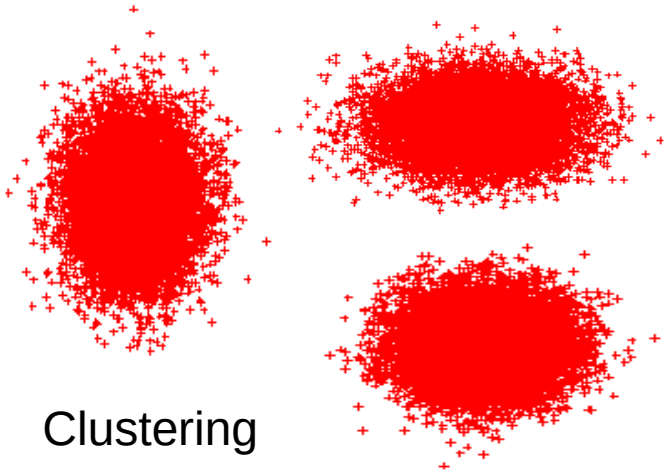
FIRSTMARK

Outline

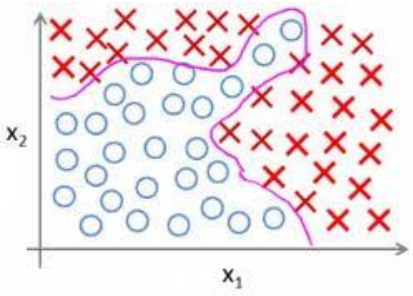
- ❑ Introduction to Big Data
- ❑ Big Data Analytics
- ❑ Evolutionary algorithms in the big data context
- ❑ A demo with MLlib
- ❑ Conclusions

Big Data Analytics

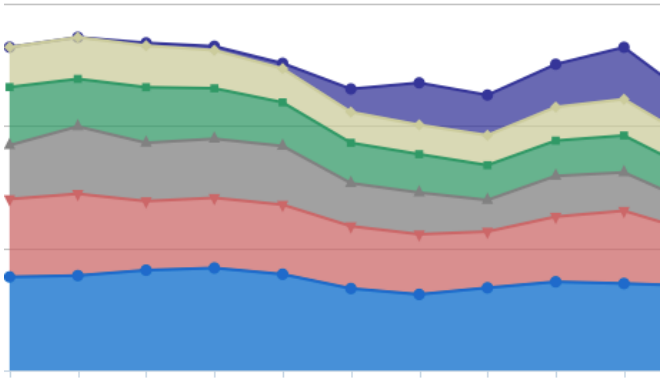
Potential scenarios



Clustering

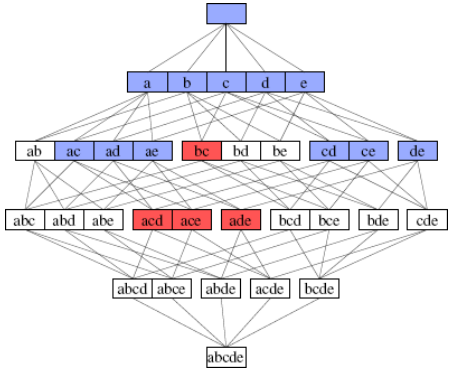


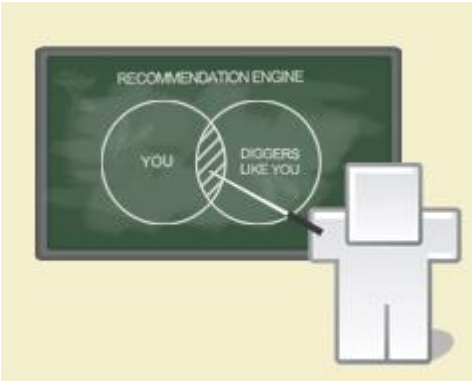
Classification



Real Time Analytics/
Big Data Streams

Association





Recommendation
Systems

Social Media Mining

Social Big Data

Big Data Analytics: A 3 generational view

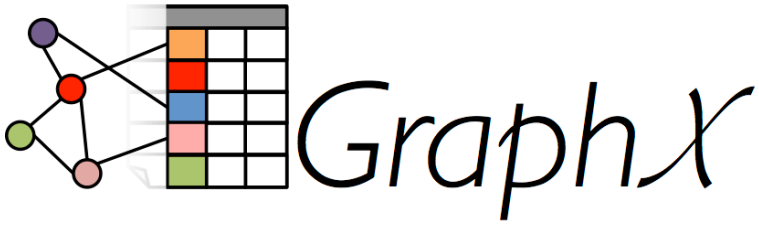
Generation	1st Generation	2nd Generation	3rd Generation
Examples	SAS, R, Weka, SPSS, KEEL	Mahout, Pentaho, Cascading	Spark, Haloop, GraphLab, Pregel, Giraph, ML over Storm
Scalability	Vertical	Horizontal (over Hadoop)	Horizontal (Beyond Hadoop)
Algorithms Available	Huge collection of algorithms	Small subset: sequential logistic regression, linear SVMs, Stochastic Gradient Descendent, k-means clustering, Random forest, etc.	Much wider: CGD, ALS, collaborative filtering, kernel SVM, matrix factorization, Gibbs sampling, etc.
Algorithms Not Available	Practically nothing	Vast no.: Kernel SVMs, Multivariate Logistic Regression, Conjugate Gradient Descendent, ALS, etc.	Multivariate logistic regression in general form, k-means clustering, etc. – Work in progress to expand the set of available algorithms
Fault-Tolerance	Single point of failure	Most tools are FT, as they are built on top of Hadoop	FT: HaLoop, Spark Not FT: Pregel, GraphLab, Giraph

Mahout (Samsara)

- First ML library initially based on Hadoop MapReduce.
- Abandoned MapReduce implementations from version 0.9.
- Nowadays it is focused on a new **math environment** called **Samsara.**
- It is integrated with **Spark, Flink** and **H2O**
- Main algorithms:
  - Stochastic Singular Value Decomposition (ssvd, dssvd)
  - Stochastic Principal Component Analysis (spca, dspca)
  - Distributed Cholesky QR (thinQR)
  - Distributed regularized Alternating Least Squares (dals)
  - Collaborative Filtering: Item and Row Similarity
  - Naive Bayes Classification

http://mahout.apache.org/

Spark Libraries



Spark MLlib

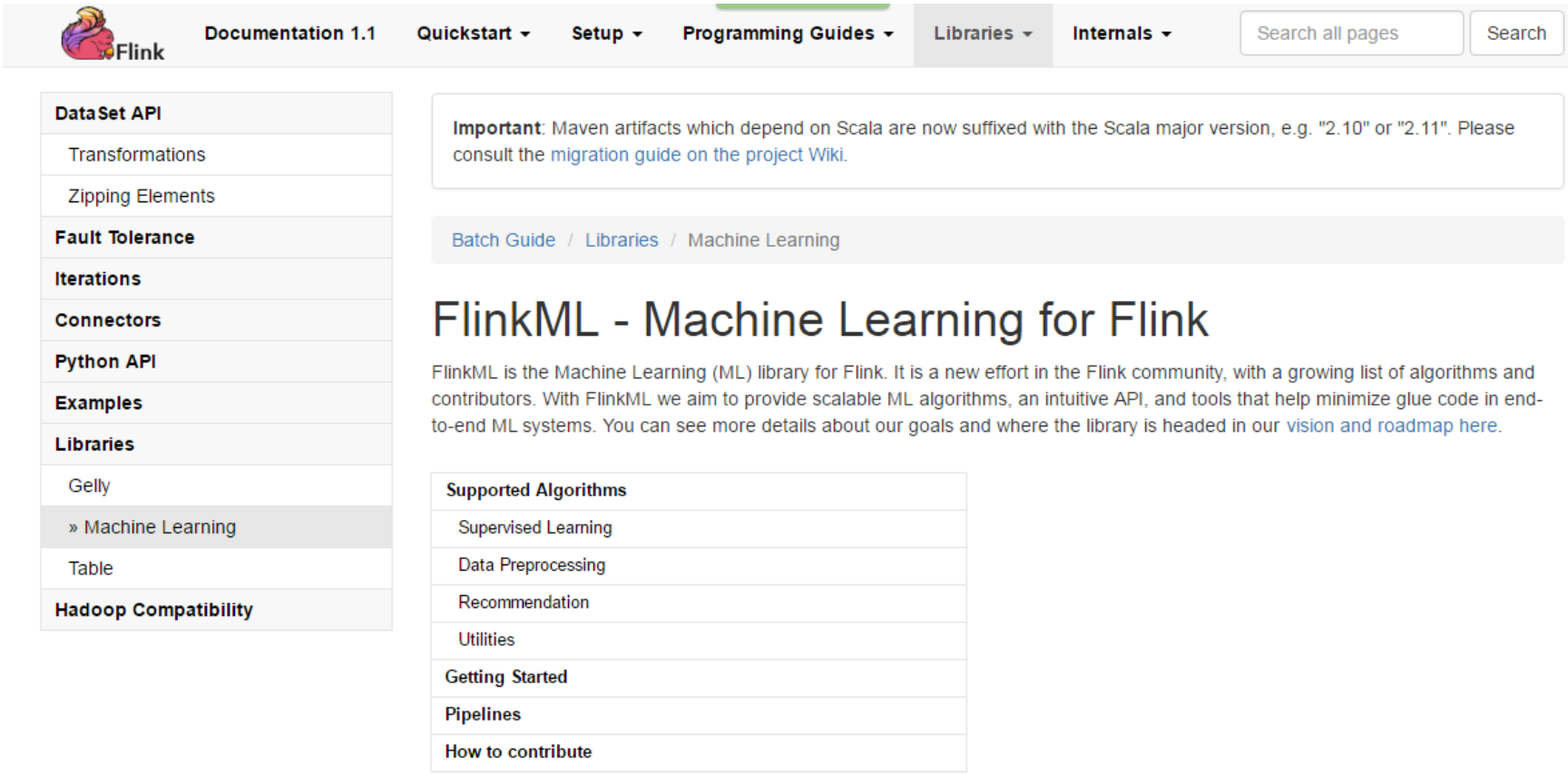
MLlib types, algorithms and utilities

This lists functionality included in `spark.mllib`, the main MLlib API.

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)
- PMML model export

https://spark.apache.org/mllib/

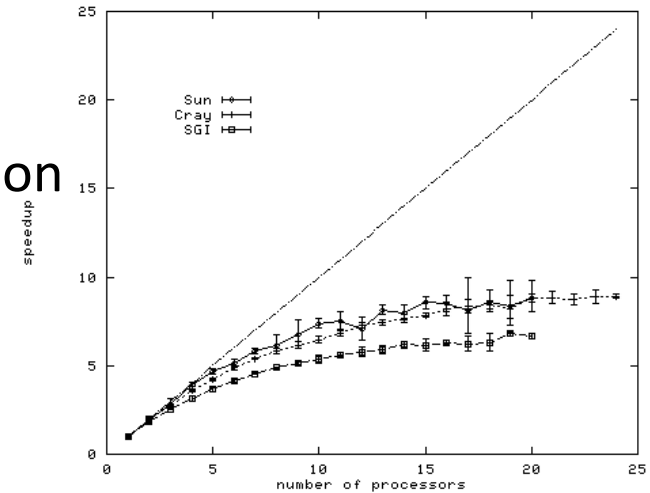
FlinkML



https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/ml/

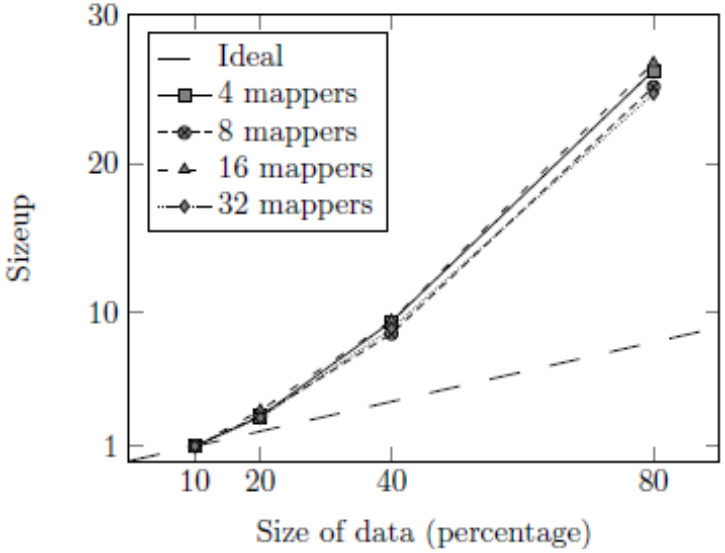
Scalability

- **Speed-up (m cores)**
  - How much faster can the same data be processed with m cores instead of 1 core
    - $Speedup(m) = \frac{runtime\ on\ 1\ core}{runtime\ on\ m\ cores}$
  - The data size is kept constant and the number of cores is increased
  - Ideal speed-up is linear
    - Speedup(m) = m
  - In practice
    - Difficult to obtain due to communication and synchronization overhead



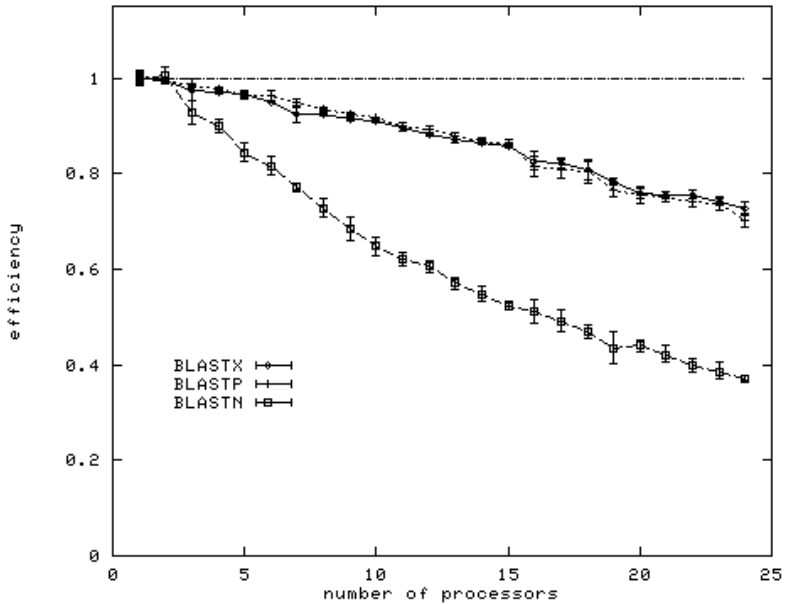
Scalability

- **Size-up (data, m)**
  - How much time does it take to execute m times larger data
    - $Sizeup(data, m) = \frac{runtime\ for\ processing\ m{\cdot}data}{runtime\ for\ processing\ data}$
  - The number of cores is kept constant and the data size is increased
  - Ideal size-up is linear
    - Sizeup(data, m) = m
  - In practice
    - Few algorithms are linear with respect to the data



Scalability

- **Scale-up (data, m)**
  - Measures the ability of the system to run a m-times greater job with a m-times larger system
    - $Sizeup(data, m) = \frac{runtime\ for\ processing\ on\ 1\ core}{runtime\ for\ processing\ m{\cdot}data\ on\ m\ cores}$
  - Both the number of cores and the data are increased
  - Ideal scale-up is 1
    - Scale-up(data, m) = 1
  - In practice
    - Few algorithms achieve a scale-up of 1



Machine Learning in Big Data: Global vs. Local

Two main ways for learning a model in Big Data:

- **Locally**
  - **A model is created for each partition** of the data (only using the data of that partition)
  - **All the models are combined** when predicting the class of a new example → Ensemble
- **Globally**
  - A **single model** is created using **all the available data**
  - They try to obtain the same model as the one that would be obtained if the method could be executed in a single node

Machine Learning in Big Data: Global vs. Local

Local model

- **Advantages**
  - Usually faster
  - Gets faster as the number of partitions is increased
  - Any existing model can be applied
  - Only the aggregation phase has to be designed
- **Disadvantages**
  - **Slow in test phase**, too many models have to be executed
  - **Loss of accuracy** as the number of partitions increases
    - With few partitions, accuracy can improve due to the ensemble effect
    - With too many partitions, the accuracy tends to drop, since there are not enough examples in each partition
  - **They do not take advantage of the data as a whole**

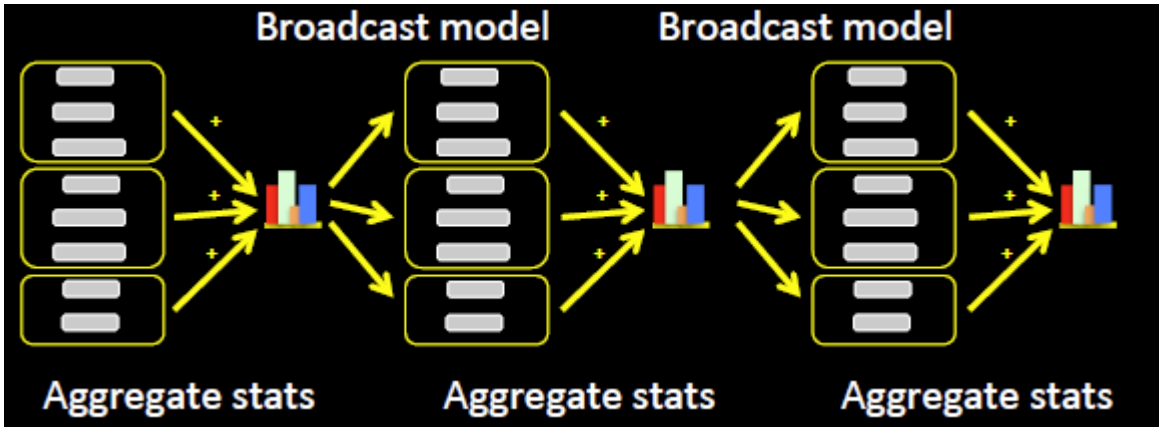
Machine Learning in Big Data: Global vs. Local

Global model

- **Advantages**
  - Greater **accuracy** is expected (not proved)
  - All the examples are used to learn a single model
  - Anyway, a global ensemble can also be built
  - The model is independent of the number of partitions
  - Faster in test phase
- **Disadvantages**
  - **More complex design and implementation**
  - Distributed nature of Big Data processing has to be taken into account (computation/communication)

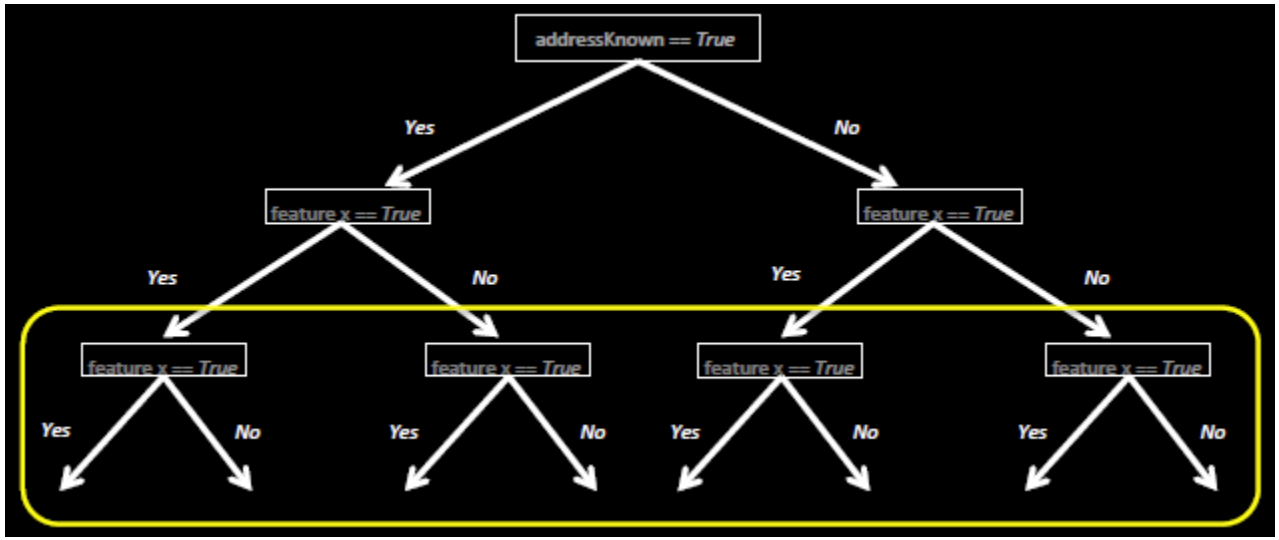
Decision Trees for Big Data

- **Decision Trees in Spark**
  - **Differences** with respect to classical models
    - All the nodes in a level are learned with a single pass through the whole dataset
    - Numeric attributes are discretized into bins in order to reduce the computational cost



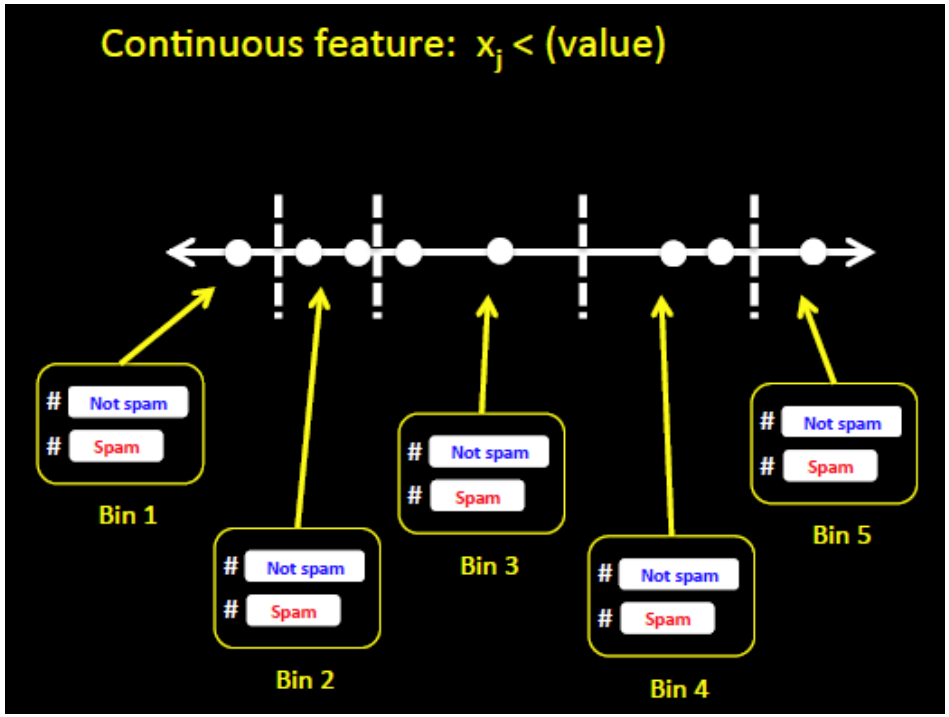
Decision Trees for Big Data

- **Decision Trees in Spark**
  - **Differences** with respect to classical models
    - All the nodes in a level are learned with a single pass through the whole dataset



Decision Trees for Big Data

- **Decision Trees in Spark**
  - **Differences** with respect to classical models
    - Numeric attributes are discretized into bins in order to reduce the computational cost

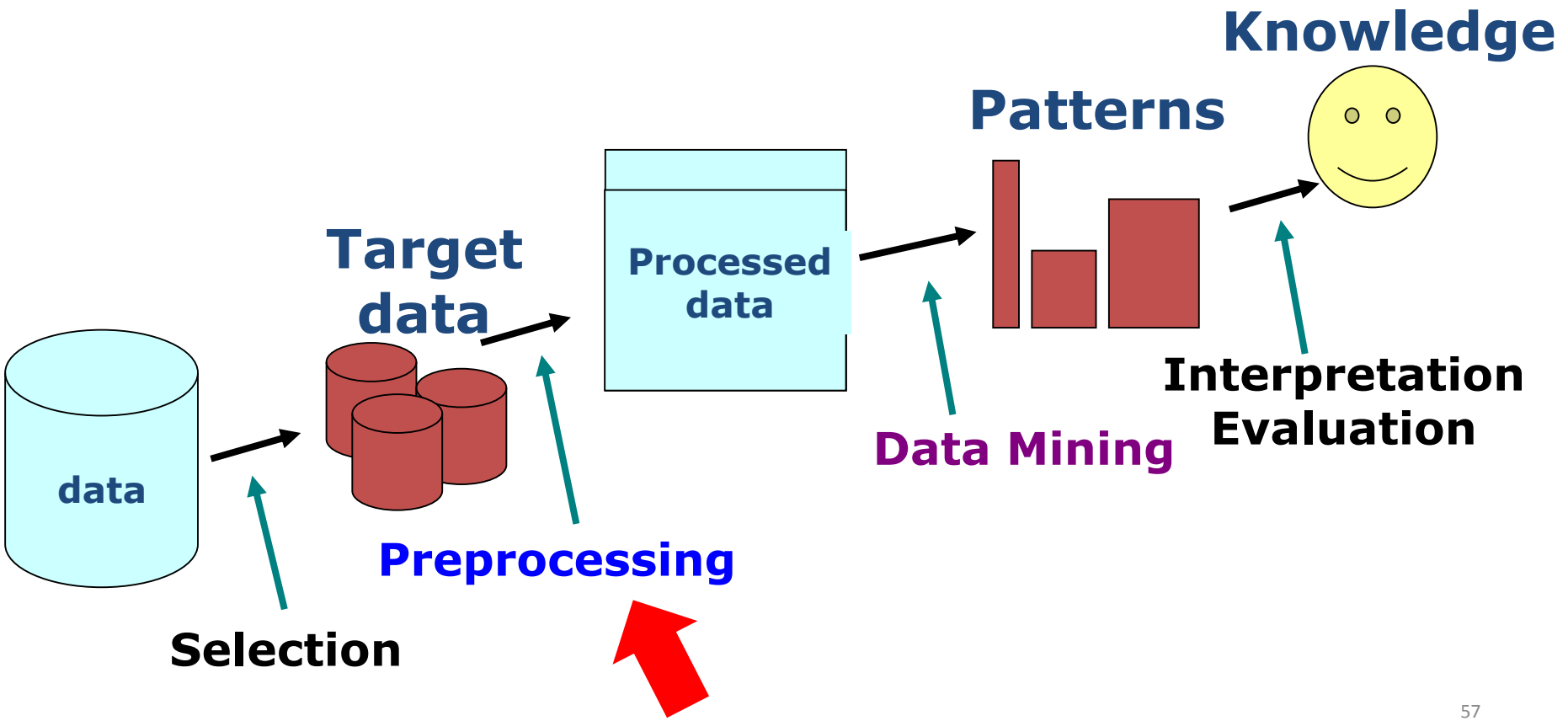


Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions

Data Preprocessing for Big Data

Data Preprocessing: Tasks to discover quality data prior to use knowledge extraction algorithms.



Evolutionary algorithms for data preprocessing

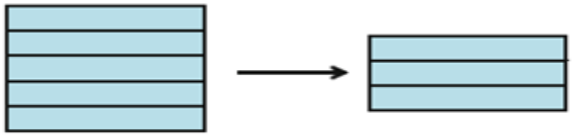
- Many preprocessing stages can be modelled as **optimisation** processes. For example:
  - Feature selection/weighting
  - Instance selection/Generation
- Evolutionary algorithms have excelled in this task in data with a moderate size.
- However, their practical application is **limited** to problems with no more than **tens of thousands of instances** because of:
  - Excessive chromosome size
  - Runtime requirements

Evolutionary algorithms for instance reduction in Big data

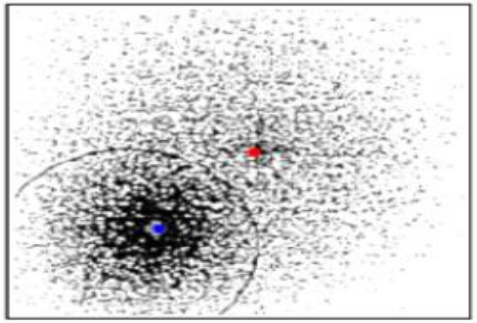
I. Triguero, D. Peralta, J. Bacardit, S.García, F. Herrera**. MRPR: A MapReduce solution for prototype reduction in big data classification**. Neurocomputing 150 (2015) 331–345

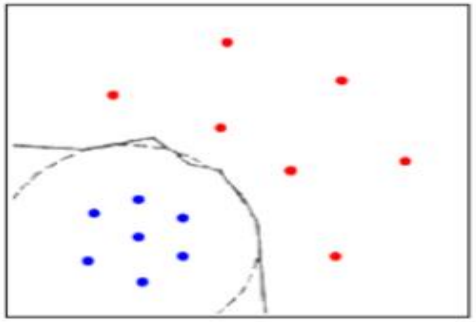
I. Triguero, D. Peralta, J. Bacardit, S.García, F. Herrera**. A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**. Evolutionary Computation (CEC), 2014 IEEE Congress on, 3036-3043

Instance Reduction



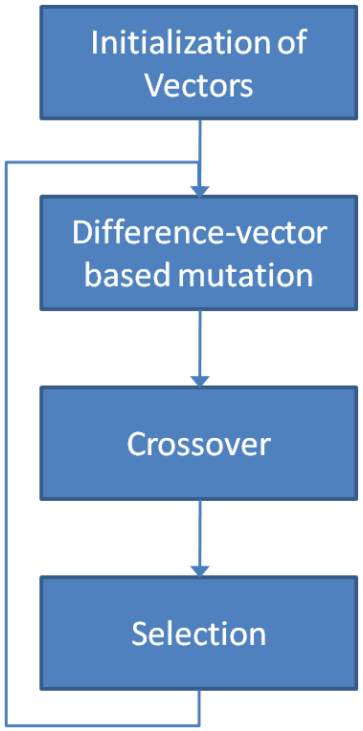
- **Objective**: reduce the number of samples to find better decision boundaries between classes, by **selecting** relevant samples or artificially **generating** new ones.
- We focused on **Prototype Generation (PG)** models, which are based on the **Nearest Neighbour (NN)** classifier.
- Advantages:
  - ✓ *Reduce Storage Requirements*
  - ✓ *Remove noisy samples*
  - ✓ *Speed up learning process*





Evolutionary Prototype Generation (EPG)

- EPG algorithms adjust the positioning of the prototypes.
- Each individual encodes a single prototype or a complete generated set with **real codification**.
- The fitness function is computed as the classification performance in the training set using the Generated Set.
- Currently, best performing approaches use ***Differential Evolution***.
- *Known issues:*
  - Dealing with big data becomes impractical



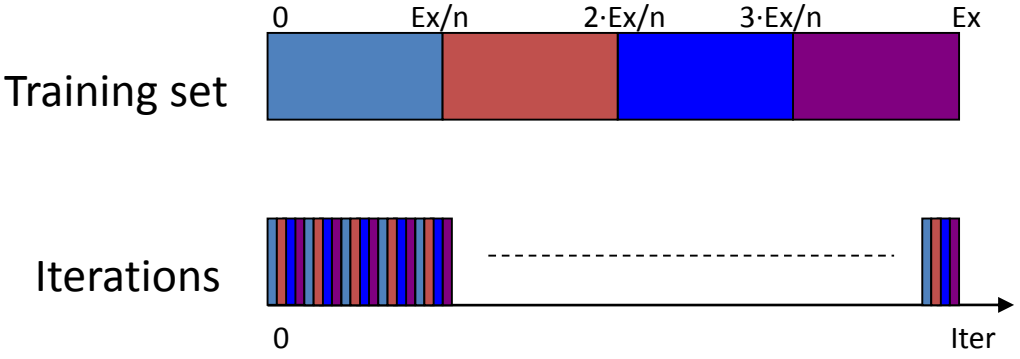
I. Triguero, S. García, F. Herrera, **IPADE: Iterative Prototype Adjustment for Nearest Neighbor Classification**. *IEEE Transactions on Neural Networks 21 (12) (2010) 1984-1990*

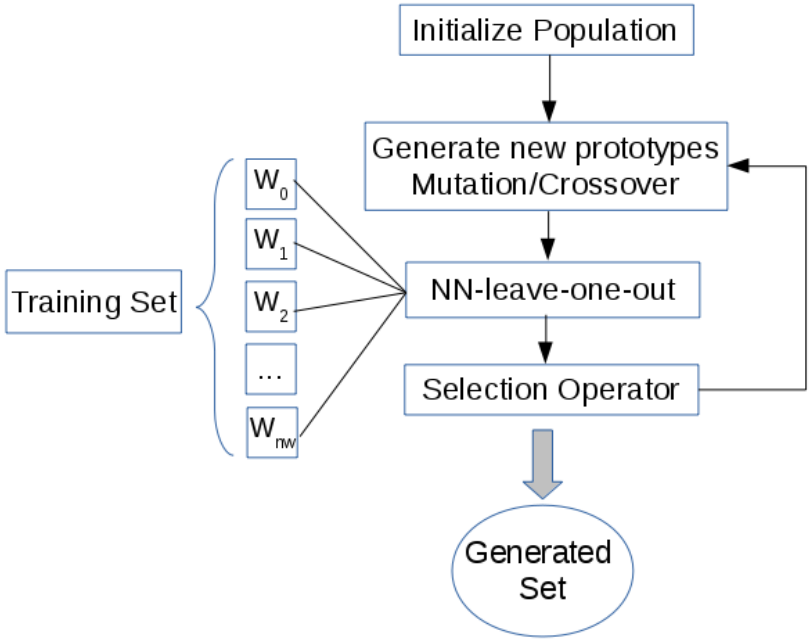
Evolutionary Prototype Generation for Big Data sets

Objectives

- The design of a scalable EPG approach that embraces the huge storage and processing capacity of cloud platforms.
- To do so, we rely on the success of **Hadoop MapReduce** in combination with a **windowing** scheme for evolutionary models.

Parallelising EPG with windowing





Main properties:

✓Avoids a (potentially biased) static prototype selection/generation

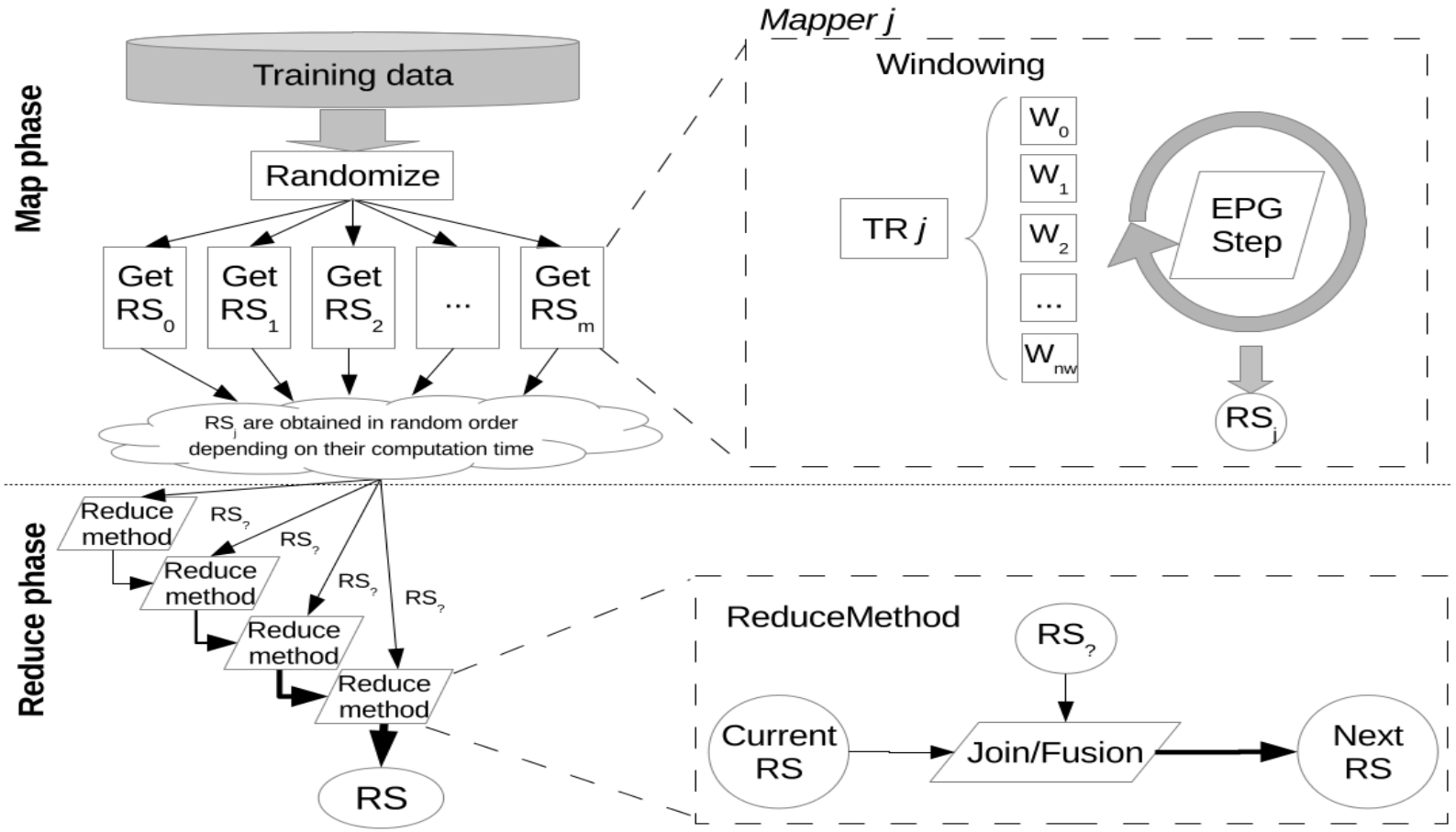
✓This mechanism also introduces some generalization pressure

Parallelising EPG with windowing

Properties:

- Within this scheme, the algorithm **disposes of the whole information** although it is accessed in successive iterations.
- This model itself aims **to improve the runtime requirements** of EPG models. But it does not deal with the memory consumption problem.
- This is why we use this strategy as a **second level parallelization** scheme after a previous distribution of the processing in a cluster of computing elements.

MRPR: Evolutionary Prototype Generation for Big Data sets



Evolutionary Prototype Generation for Big Data sets

Experimental Study

- 4 big data sets: Poker (1M), KddCup (4.8M), Susy (5M), RLCP(5.7M).
- Performance measures: Accuracy, reduction rate, runtime, test classification time and speed up.
- 3x5 fold-cross validation
- Number of mappers = 64/128/256/512/1024.
- Number of reducers=1
- PG techniques tested: SSMA-SFLSDE, LVQ3, RSP3
- PS techniques tested: DROP3, FCNN

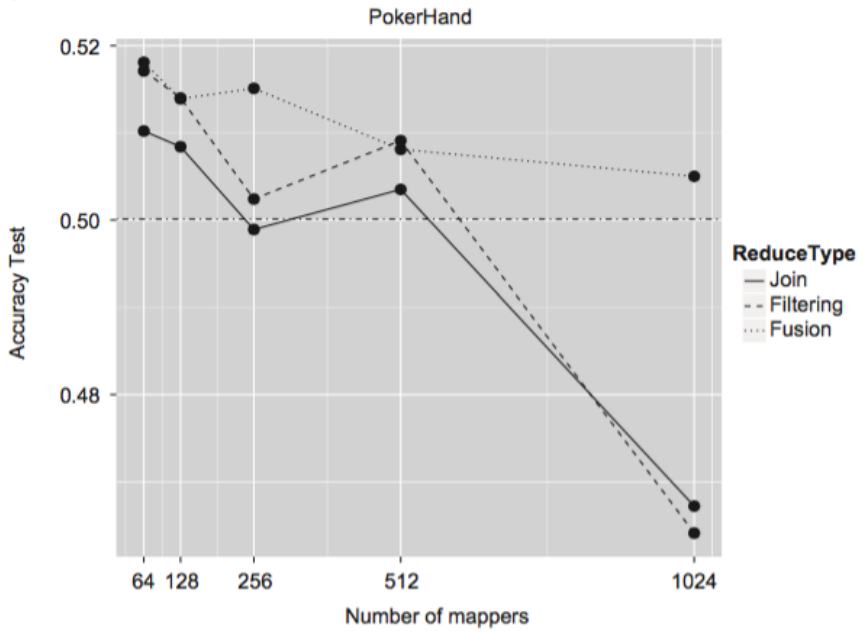
Evolutionary Prototype Generation for Big Data sets

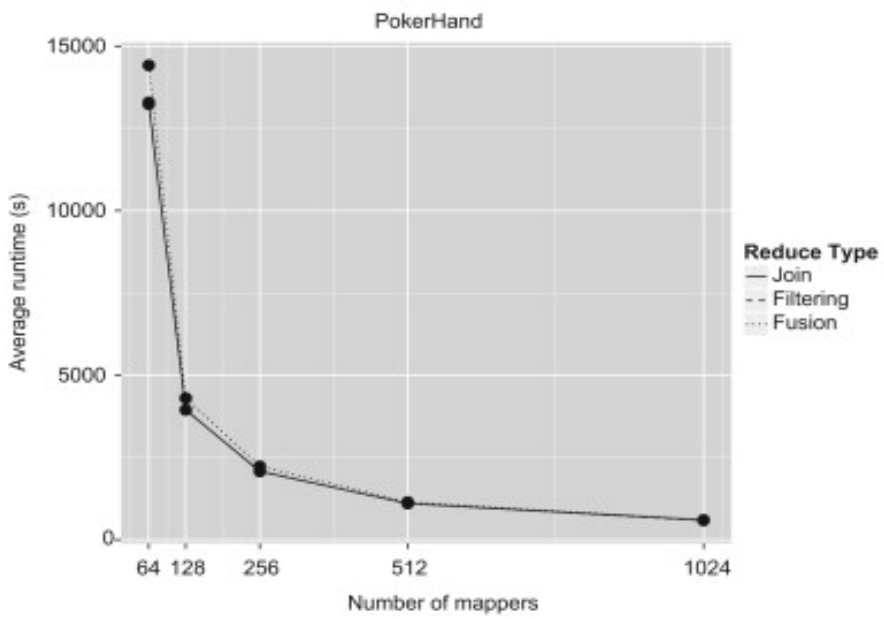
Table : Results obtained for the PokerHand problem.

Reduce type	#Mappers	Training		Test		Runtime		Reduction rate		Classification time (*TS*)
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	
Join	64	**0.5158**	0.0007	**0.5102**	0.0008	13236.6012	147.8684	97.5585	0.0496	1065.1558
Filtering	64	***0.5212***	0.0008	**0.5171**	0.0014	13292.8996	222.3406	98.0714	0.0386	848.0034
Fusion	64	**0.5201**	0.0011	***0.5181***	0.0015	14419.3926	209.9481	99.1413	0.0217	374.8814
Join	128	**0.5111**	0.0005	**0.5084**	0.0011	3943.3628	161.4213	97.2044	0.0234	1183.6378
Filtering	128	**0.5165**	0.0007	**0.5140**	0.0007	3949.2838	135.4213	97.7955	0.0254	920.8190
Fusion	128	**0.5157**	0.0012	**0.5139**	0.0006	4301.2796	180.5472	99.0250	0.0119	419.6914
Join	256	**0.5012**	0.0010	0.4989	0.0010	2081.0662	23.6610	96.5655	0.0283	1451.1200
Filtering	256	**0.5045**	0.0010	**0.5024**	0.0006	2074.0048	25.4510	97.2681	0.0155	1135.2452
Fusion	256	**0.5161**	0.0004	**0.5151**	0.0007	2231.4050	14.3391	98.8963	0.0045	478.8326
Join	512	**0.5066**	0.0007	**0.5035**	0.0009	1101.8868	16.6405	96.2849	0.0487	1545.4300
Filtering	512	**0.5114**	0.0010	**0.5091**	0.0005	1101.2614	13.0263	97.1122	0.0370	1472.6066
Fusion	512	**0.5088**	0.0008	**0.5081**	0.0009	1144.8080	18.3065	98.7355	0.0158	925.1834
Join	1024	0.4685	0.0008	0.4672	0.0008	598.2918	11.6175	95.2033	0.0202	2132.7362
Filtering	1024	0.4649	0.0009	0.4641	0.0010	585.4320	8.4529	96.2073	0.0113	1662.5460
Fusion	1024	**0.5052**	0.0003	**0.5050**	0.0009	601.0838	7.4914	98.6249	0.0157	1345.6998
NN	–	0.5003	0.0007	0.5001	0.0011	–	–	–	–	48760.8242

Evolutionary Prototype Generation for Big Data sets

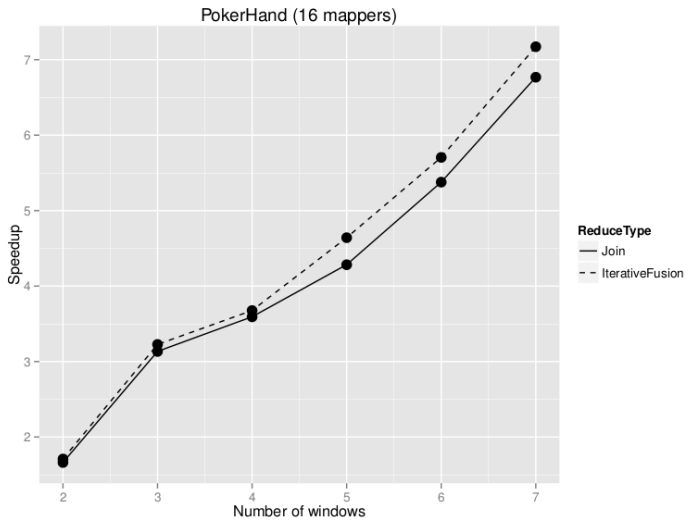
a





Evolutionary Prototype Generation for Big Data sets

#Windows nw	#Mappers	Training		Test		Runtime	
		Avg.	Std.	Avg.	Std.	Avg.	Std.
1	16	0.5121	0.0028	0.5120	0.0031	15058.4740	1824.6586
2	16	0.5115	0.0035	0.5113	0.0036	8813.7134	678.1335
3	16	0.5038	0.0032	0.5039	0.0033	4666.5424	412.5351
4	16	0.5052	0.0060	0.5055	0.0057	4095.8610	941.5737
5	16	0.5041	0.0024	0.5034	0.0022	3244.0716	534.8720
6	16	0.5031	0.0042	0.5028	0.0041	2639.4266	360.3121
7	16	0.5000	0.0067	0.4998	0.0069	2099.5182	339.7356
1	32	0.5089	0.0031	0.5086	0.0029	6963.5734	294.3580
2	32	0.5084	0.0045	0.5080	0.0041	4092.5484	855.7351
3	32	0.5067	0.0025	0.5065	0.0024	2343.1542	104.7222
4	32	0.5012	0.0045	0.5012	0.0039	1639.0032	335.6036
5	32	0.5012	0.0045	0.5012	0.0039	1639.0032	335.6036
6	32	0.4824	0.0104	0.4820	0.0101	1083.1116	143.9288
7	32	0.4838	0.0072	0.4835	0.0065	1129.8838	173.9482



Evolutionary Prototype Generation for Big Data sets

- Evolutionary algorithms continue to be the best performing models for Instance reduction in the big data context.
- Great synergy between the windowing and MapReduce approaches. They complement themselves in the proposed two-level scheme.
- Without windowing, EPG could not be applied to datasets larger than approximately ten thousands instances.
- The application of this model has resulted in a very big reduction of storage requirements and classification time for the NN rule.

https://github.com/triguero/MRPR

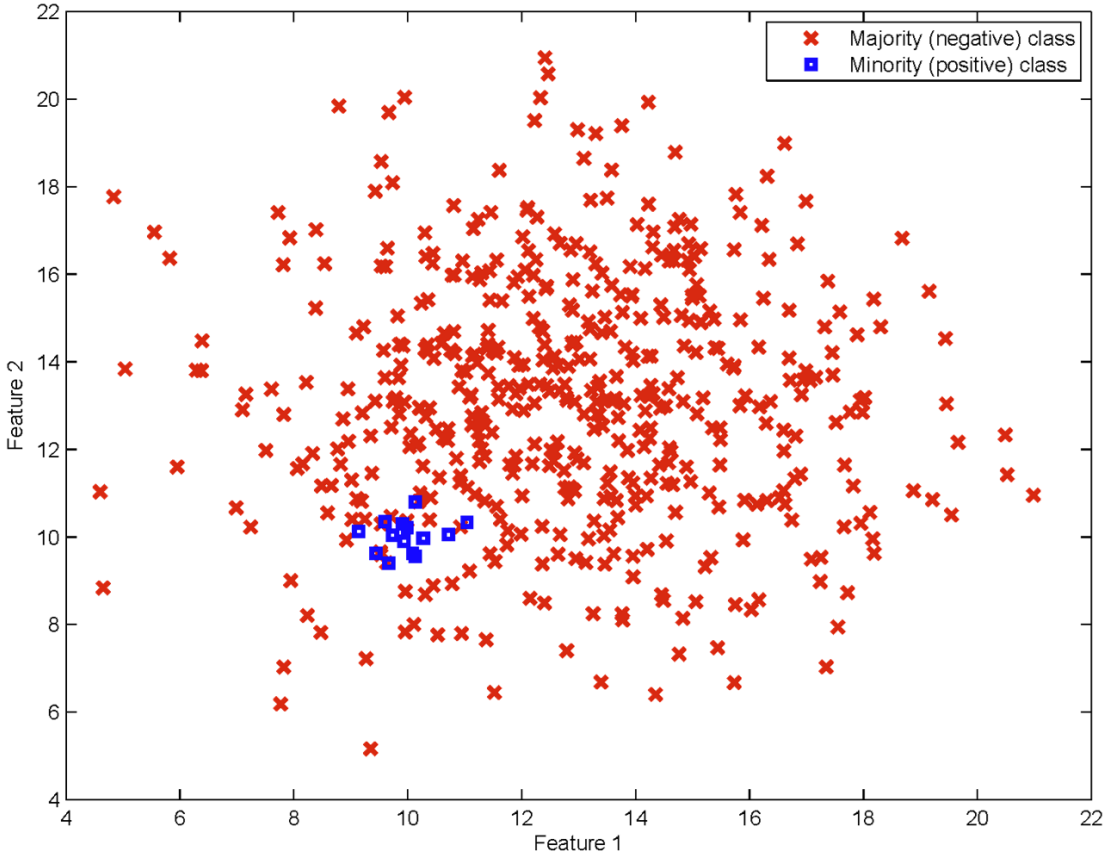
Evolutionary algorithms for imbalanced Big data sets

I Triguero, M Galar et. al. **Evolutionary undersampling for imbalanced big data classification**. IEEE Congress on Evolutionary Computation (CEC), 2015.

I Triguero, M Galar et. al. **Evolutionary Undersampling for Extremely Imbalanced Big Data Classification under Apache Spark**. IEEE Congress on Evolutionary Computation (CEC), 2016.

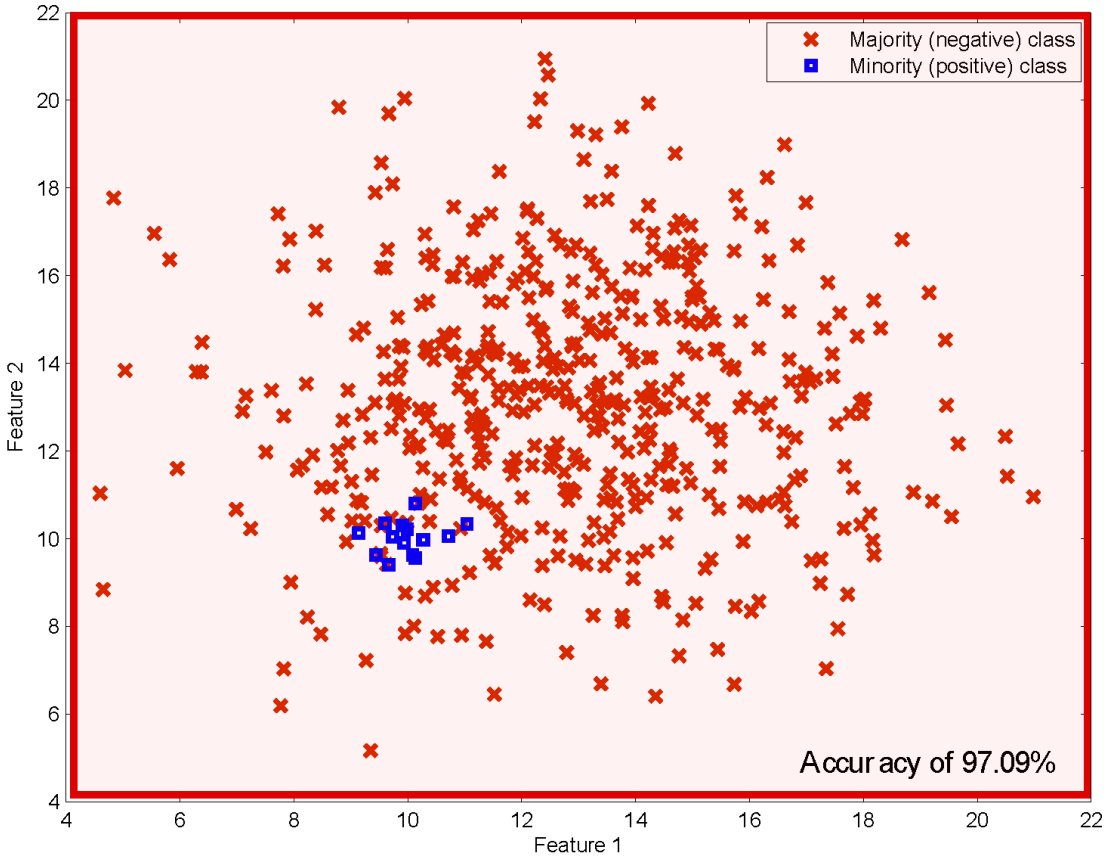
I Triguero, M Galar et. al. **A First Attempt on Global Evolutionary Undersampling for Imbalanced Big Data**. IEEE Congress on Evolutionary Computation (CEC), 2017.

Class imbalance problem



An example of an imbalanced data-set

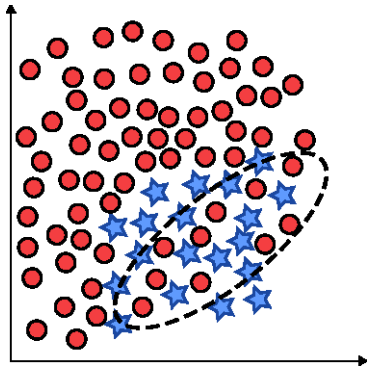
Class imbalance problem



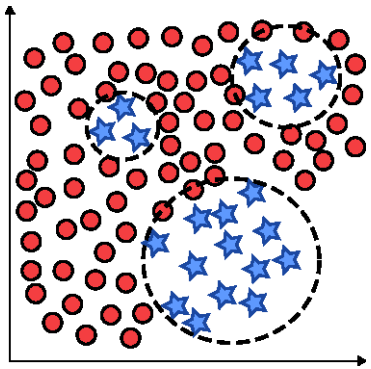
Standard classifiers → models biased in favour of the majority class

Class imbalance problem

- Skewed data distribution by itself is not harmful
- But… a series of **difficulties** usually turn up
  - *Small sample size*
  - *Overlapping or class separability*
  - *Small disjuncts*



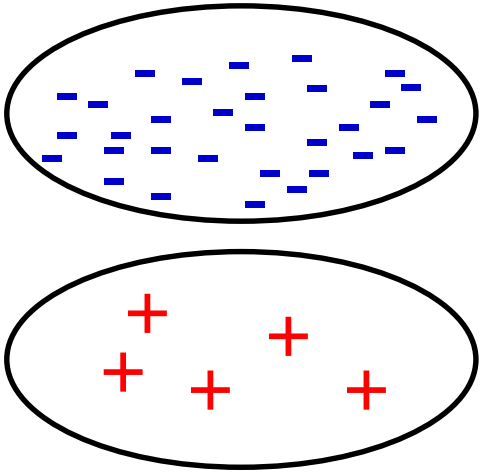
(a) Class overlapping



(b) Small disjuncts

Class imbalance problem

- Two main approaches to tackle this problem:
  - **Data sampling**
    - **Undersampling**
    - Oversampling
    - Hybrid approaches
  - Algorithmic modifications
  - Cost-sensitive approaches
  - Ensemble models



V. López et al, **An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics**, Information Sciences 250 (2013)
M. Galar et al, **A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches**, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 42 (4) (2012)

Evolutionary Undersampling

- Evolutionary undersampling (EUS) aims to select the **best subset of negative instances** from the original training set.
- EUS not only intends to balance the training set, but also to increase the overall performance on both classes of the problem.
- To do so, a genetic algorithm is used to search for an optimal subset of instances.
- This resulting set can be used by any standard classification model.

S. Garcia and F. Herrera, "**Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy**", Evolutionary Computation, 17 (3) (2009). 275–306

Evolutionary Undersampling in the big data context

- EUS **does not generate** more data, as opposed to oversampling methods.
- However, the increasing number of instances would lead to obtain an excessive chromosome size that can limit their application.
- The required runtime increases not only with the **number of examples** but also with **the imbalance ratio** (IR).

Evolutionary undersampling

- Representation of the solution

$$V = (v_{x_1}, v_{x_2}, v_{x_3}, v_{x_4}, \ldots, v_{x_{n^-}}), \;\; v_{x_i} \in \{0, 1\} \;\; \text{for all} \;\; i = 1, \ldots, n^-$$

- Performance: g-mean, 1NN hold-one-out

$$\text{g-mean} = \sqrt{\text{TP}_{rate} \cdot \text{TN}_{rate}}$$

- Fitness Function

$$\text{fitness}_{\text{EUS}} = \begin{cases} \text{g-mean} - \left|1 - \frac{n^+}{N^-}\right| \cdot P & \text{if } N^- > 0 \\ \text{g-mean} - P & \text{if } N^- = 0, \end{cases}$$

- We use the CHC algorithm and GM as performance measure.

L. J. Eshelman, The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in Foundations of Genetic Algorithms, G. J. E. Rawlins, Ed. San Francisco, CA: Morgan Kaufmann, 265-283, 1991.

EUS-BD: A two-level parallelisation model for EUS

- A two-level parallelisation scheme:
  - The MapReduce phase will allow us to divide the computational effort over different machines.

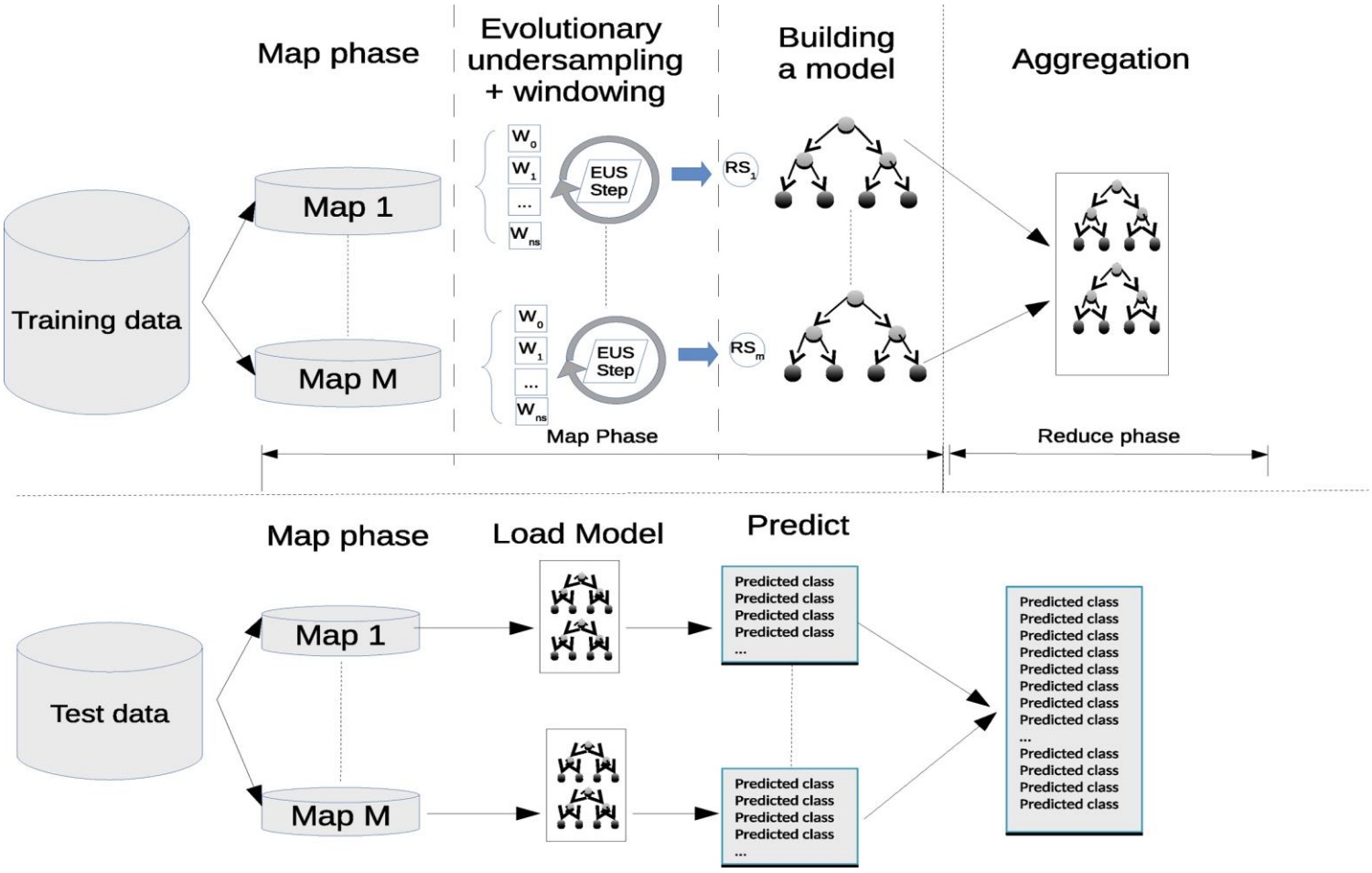
    Goal: Memory limitations and runtime.
  - The windowing scheme will be applied to reduce the computational time required by EUS.

    Goal: Runtime.

Windowing for Class Imbalance

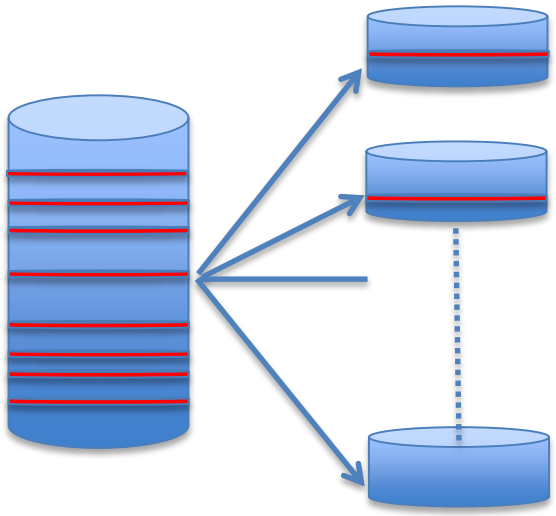
- Disjoint windows with equal class distribution may lead to information loss of the positive class.
- The minority class set will be always used to evaluate a chromosome.
- The majority class set is divided into several disjoint strata. The size of each subset will correspond to the number of minority class instances.
  - It means: Fixed number of strata.

The EUS-BD scheme



EUS-BD known issues

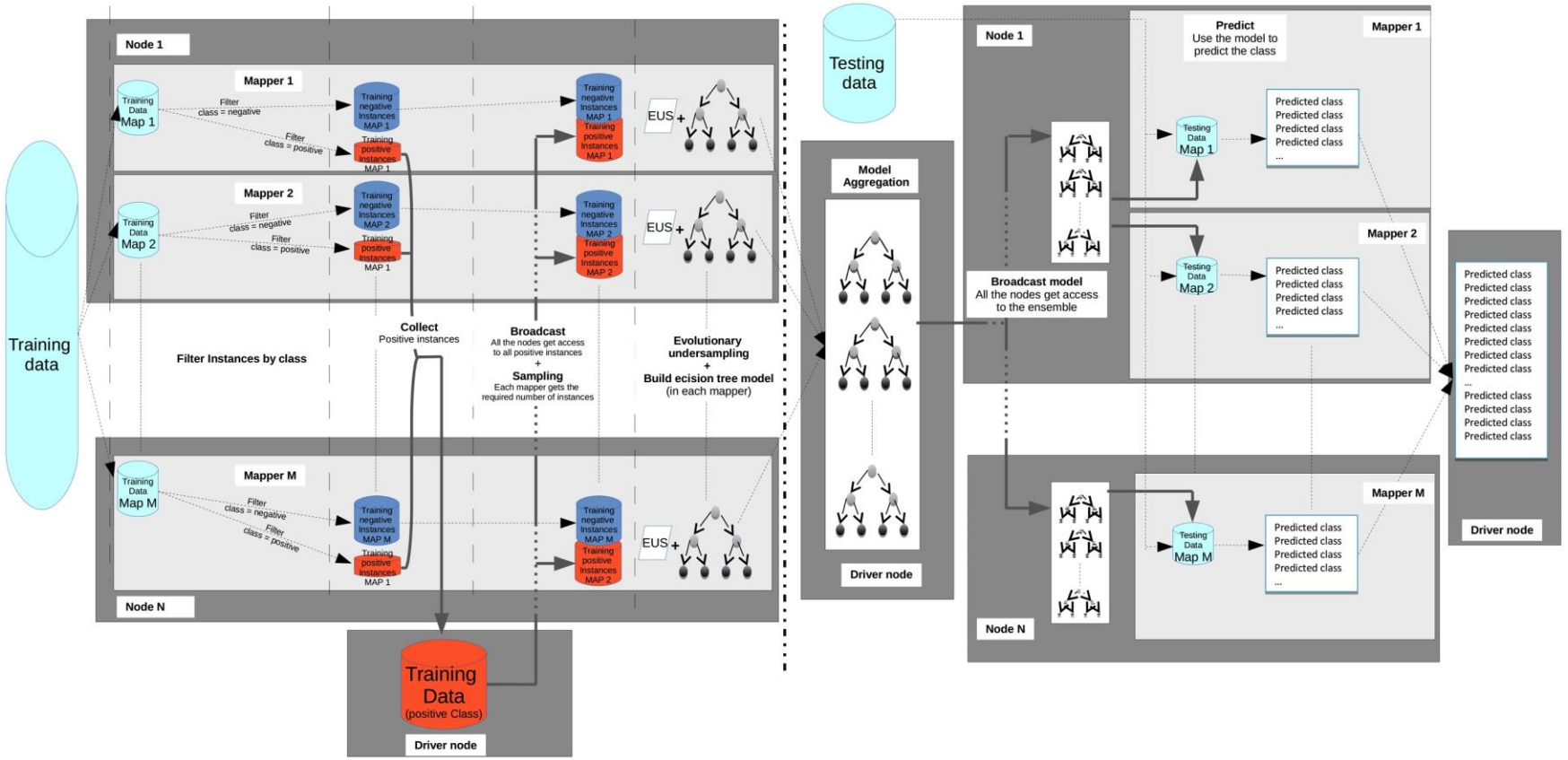
- This is a local model!
- Extremely imbalanced cases: **lack of density** from the minority class.
- We used Spark to alleviate that issue splitting training data into positive and negative sets. The positive set was accessible in all the maps (via broadcast)



Too small amount of positive data.

Or even no minority-class elements!

EUS-BD for Extremely Imbalanced datasets



EUS-BD for Extremely Imbalanced datasets

Data set	#features	#negative	#positive	IR
ECBDL'14 (50%)	631	17101086	344333	49
ECBDL'14 (25%)	631	8550324	172386	49
Kddcup DOS vs. PRB	41	3883370	41102	94.48
Kddcup DOS vs. R2L	41	3883370	1126	3448.82
Kddcup DOS vs. U2R	41	3883370	52	74680.25

TABLE III: Running times obtained by the Hadoop and Spark implementations of EUS-ImbBD and EUS-ExtImbBD.

		Hadoop-based		**Spark-based**		**Spark improvement**	
Dataset	No. of maps	Build time (s)	Classif. time (s)	Build time(s)	Classif. time (s)	Build time	Classif. time
Kddcup DOS vs. PRB	128	422.4786	34.264	297.5048	0.2942	29.58%	99.14%
	256	240.4662	36.7934	143.3428	0.3566	40.39%	99.03%
	512	156.4354	48.424	87.0195	0.2739	44.37%	99.43%
Kddcup DOS vs. R2L	128	444.7252	31.7255	320.8192	0.0876	27.86%	99.72%
	256	266.2424	36.1147	187.4562	0.1024	29.59%	99.72%
	512	178.8536	42.0057	148.319	0.1371	17.07%	99.67%
Kddcup DOS vs. U2R	128	459.6002	31.8436	340.2297	0.0986	25.97%	99.69%
	256	248.1038	35.5862	193.0784	0.1081	22.18%	99.70%
	512	152.3752	46.6194	101.683	0.1275	33.27%	99.73%

EUS-BD for Extremely Imbalanced datasets

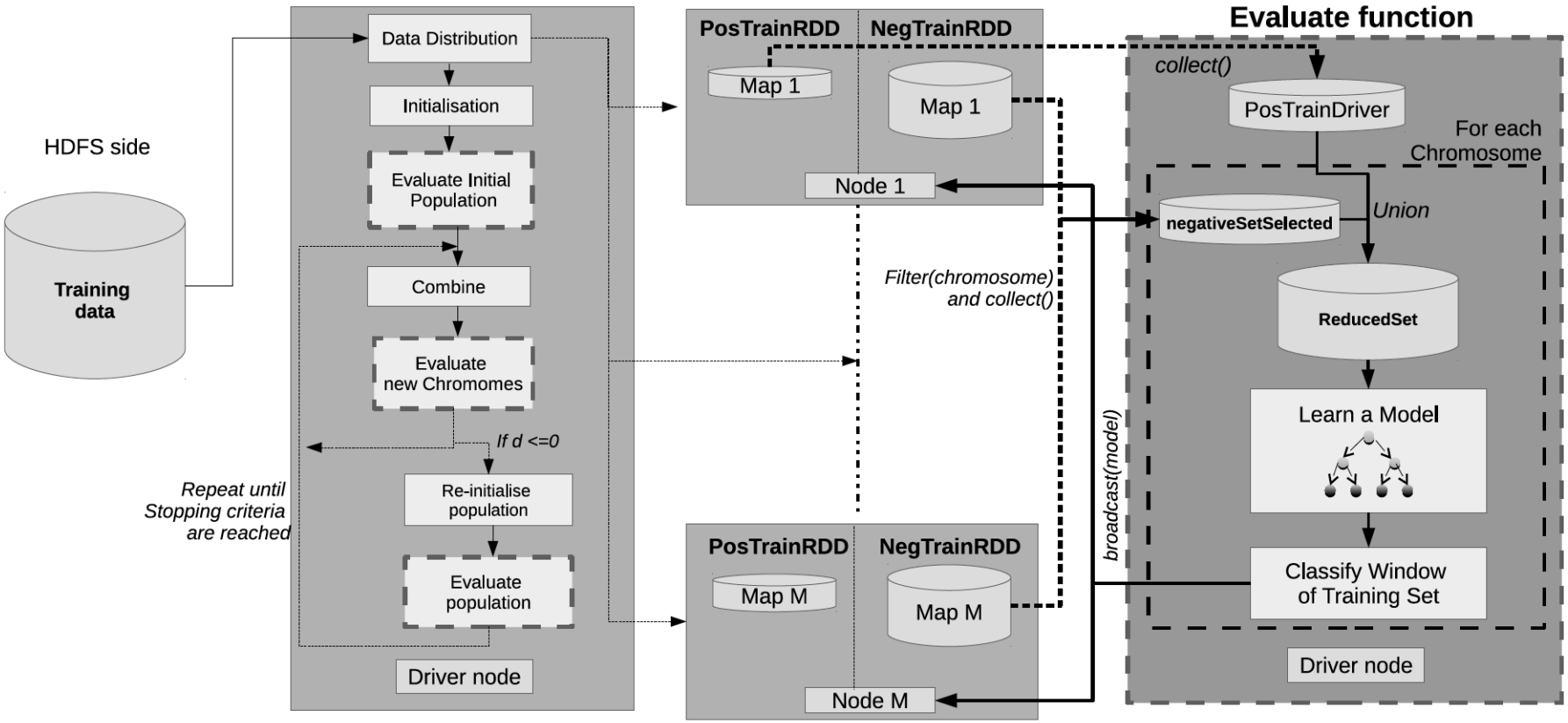
TABLE V: Results obtained in ECBLD'14 (50%)

Method	#Maps	Build time(s)	Classif. time(s)	AUC	GM
EUS-S-ImbBD	8192	497.0958	0.4999	0.5556	0.3572
	4096	775.2016	0.5531	0.4821	0.4276
	2048	1476.3512	0.7878	0.6674	0.6645
EUS-S-ExtImbBD	8192	2181.5089	3.5404	0.6641	0.6640
	4096	3456.5938	6.0428	0.6662	0.6657
	2048	6433.8072	9.4064	0.6731	0.6704
RUS-S-ImbBD	8192	557.4869	0.5540	0.6319	0.6066
	4096	518.1471	1.0960	0.4659	0.4339
	2048	483.7361	1.0960	0.6651	0.6622

Can we develop a Global Evolutionary model?

- Ideally, EUS should have access to all the data as a whole. **Can we do that with the current technology?**
  - A binary representation of the selected instances implies a chromosome size equal to the number of negative instances.
  - However, the resulting selected dataset tends to be fairly small, and balanced.
  - *Assumption*: The number of positive instance is so reduced that it perfectly fits in main memory of a single computer.

Global Evolutionary Undersampling



Outline

- ❑ What is Big data?
- ❑ How to deal with Data Intensive applications?
- ❑ Big Data Analytics
- ❑ A demo with MLlib
- ❑ Conclusions

Demo

- In this demo we will show two ways of working with Apache Spark:
  - Interactive mode with Spark Notebook.
  - Standalone mode with IntelliJ.

- All the code used in this presentation is available at:

http://www.cs.nott.ac.uk/~pszit/BigDataCEC2017.html

DEMO with Spark Notebook in local

SPARK NOTEBOOK

http://spark-notebook.io/

DEMO with Spark Notebook in local

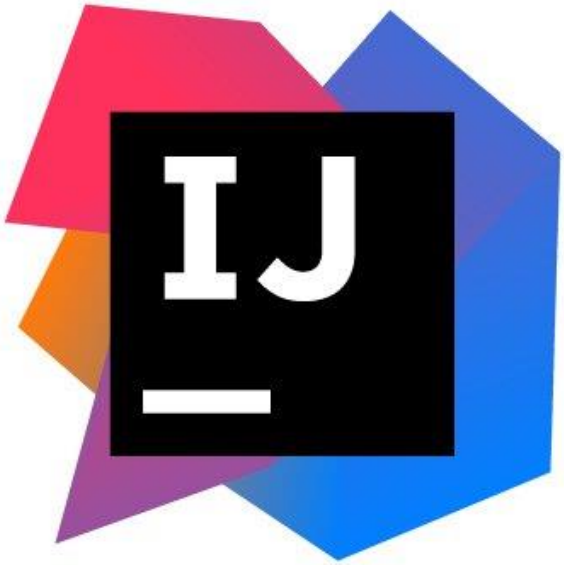
Advantages:

- ✓ Interactive.
- ✓ Automatic plots.
- ✓ It allows connection with a cluster.
- ✓ Tab completion

Disadvantages:

- ❑ Built-in for specific spark versions.
- ❑ Difficult to integrate your own code.

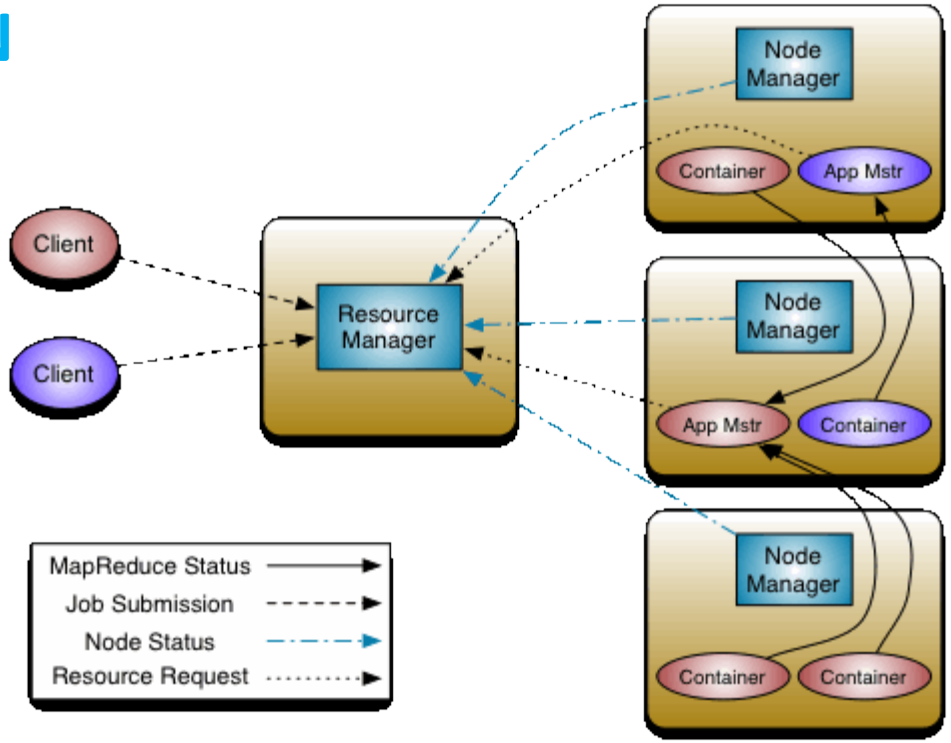
DEMO with IntelliJ IDE



https://www.jetbrains.com/idea/

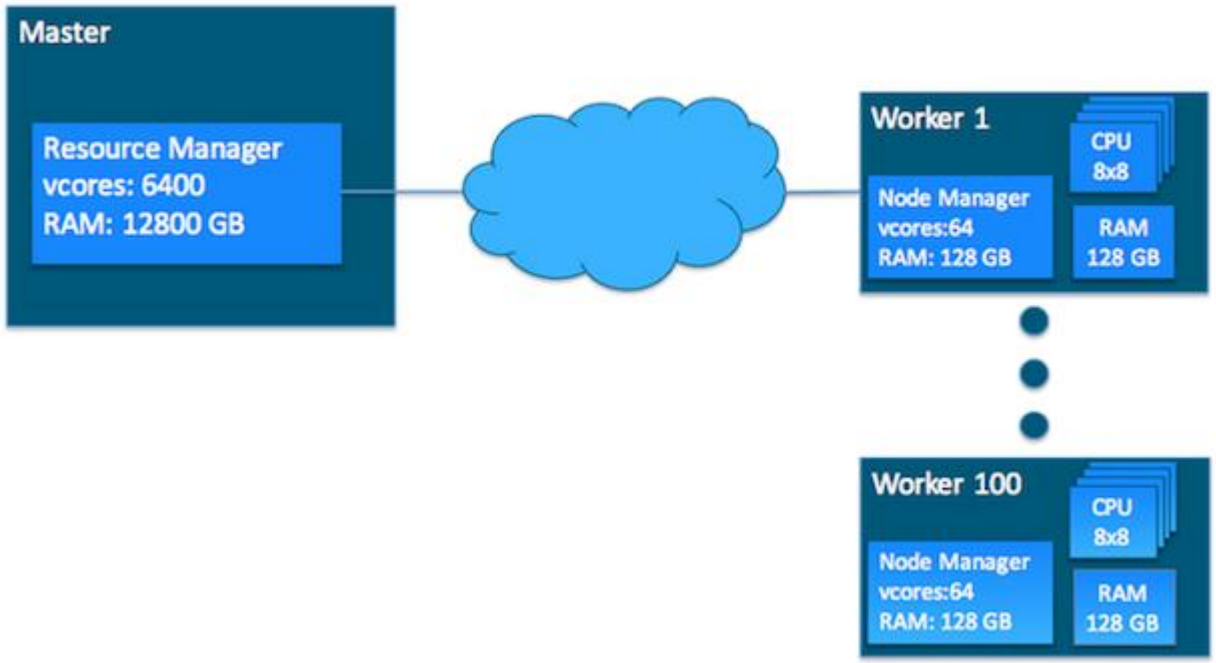
MapReduce in a cluster

- **Hadoop V2**
  - **YARN**



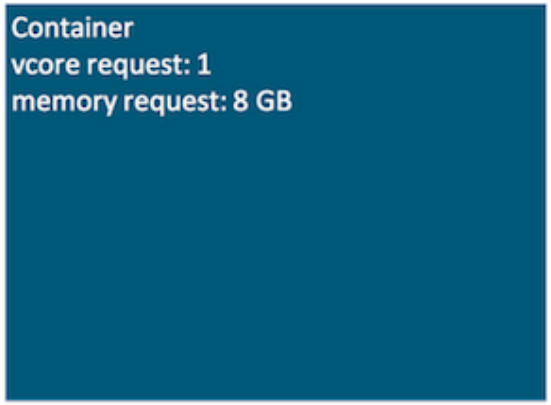
MapReduce in a cluster

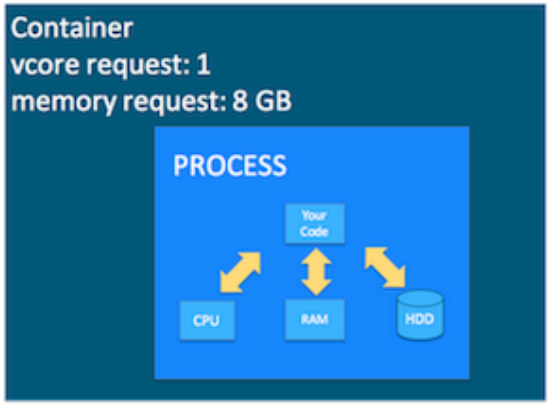
- **Hadoop V2**



MapReduce in a cluster

- **Hadoop V2**
  - **Container**
    - A subset of the resources of the cluster (part of a node)
      - Number of **cores**
      - Quantity of **RAM memory**
    - A hold request is made
    - Once granted, a process (task) can be run in the container



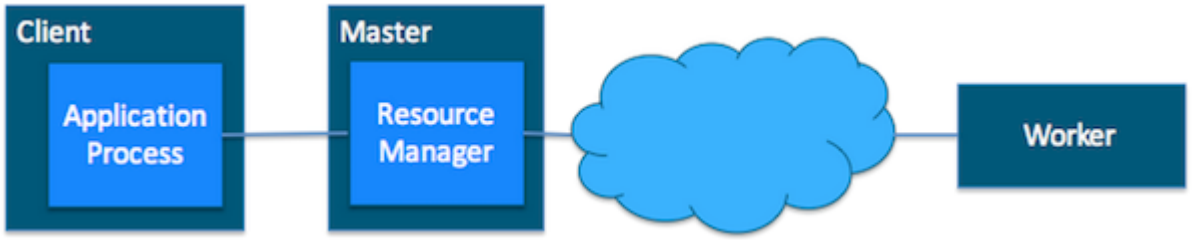


MapReduce in a cluster

- **Hadoop V2**
  - **ApplicationMaster**
    - New concept
    - **Responsible for the processing**
    - Responsible for negotiating with the ResourceManager and working with the NodeManagers
    - **In charge of the fault tolerance**
      - ResourceManager is no longer used for this task

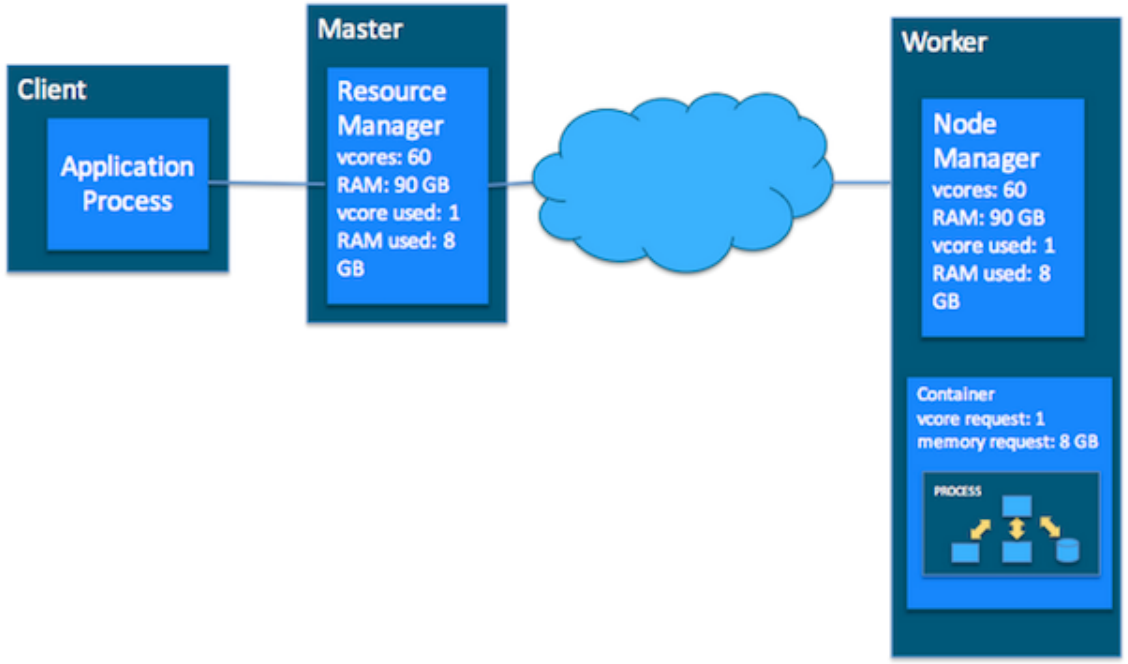
MapReduce in a cluster

- **Hadoop V2**
  - **Execution** of a MapReduce process
    1. The client launches the process (connection with the ResourceManager)



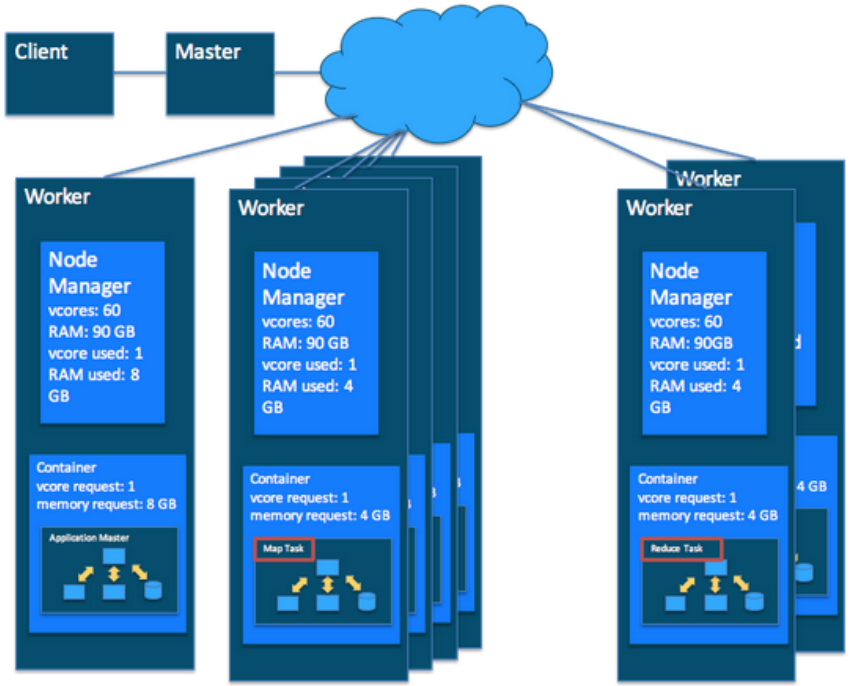
MapReduce in a cluster

- **Hadoop V2**
  - **Execution** of a MapReduce process
    2. The ResourceManager requests a container where the ApplicationMaster is executed



MapReduce in a cluster

- **Hadoop V2**
  - **Execution** of a MapReduce process
    3. The ApplicationMaster request the containers to execute all the tasks (in different nodes)

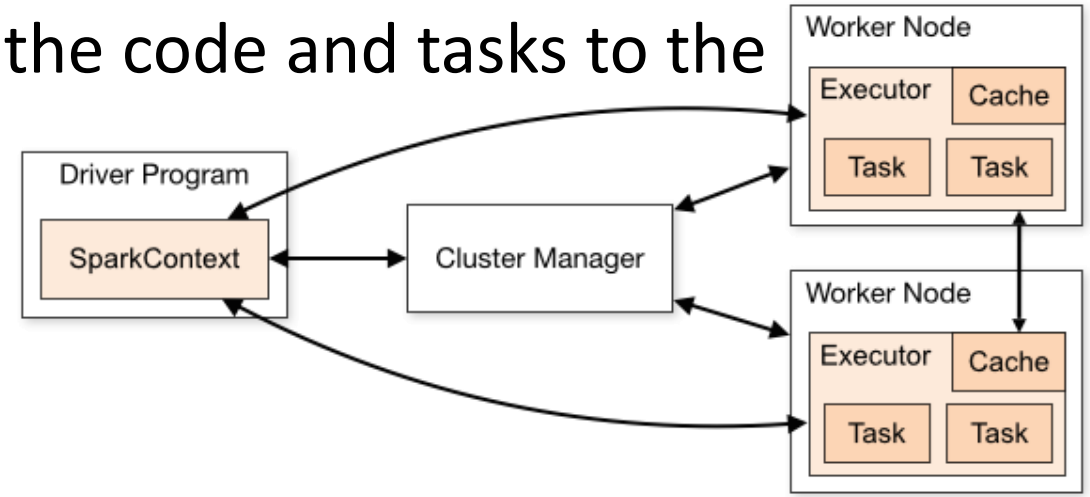


MapReduce in a cluster

- **Hadoop V2**
  - **Execution** of a MapReduce process
    4. All the tasks are executed in the containers
       - Containers are released once its tasks are finished
    5. The ApplicationMaster ends when all the tasks have been executed. Then, its container is released

Spark: Execution in a cluster

- **SparkContext** (sc) is created in the driver
  - Using the **sc a connection with the cluster manager is established**
  - Once connected, executors are requested
    - The processes that perform the computation and store the data
  - The driver sends the code and tasks to the executors



Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions

Conclusions

- We need new strategies to deal with big datasets
  - Choosing the right technology is like choosing the right data structure in a program.
- The world of big data is rapidly changing. Being up-to-date is difficult but necessary.
- Evolutionary models are powerful tools in data mining. They need to be adapted and redesigned to take the maximum advantages of their promising properties.

Big Data Learning with Evolutionary Algorithms

Isaac Triguero
School of Computer Science
University of Nottingham
United Kingdom
Isaac.Triguero@nottingham.ac.uk

Mikel Galar
Dept. Automatic and Computation
Public University of Navarre
Spain
mikel.galar@unavarra.es

http://www.cs.nott.ac.uk/~pszit/BigDataCEC2017.html

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

ASAP research group

30 1987-2017 upna

upna Research Institutes

5th June 2017

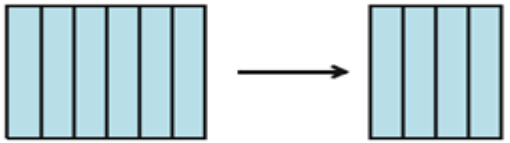
Extra slides

Evolutionary algorithms for Feature Selection/Weighting in Big Data

D. Peralta,et al. **Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**. Mathematical Problems in Engineering, 2015

I. Triguero, et al. **ROSEFW-RF: The winner algorithm for the ECBDL'14 Big Data Competition: An extremely imbalanced big data bioinformatics problem**. Knowledge-Based Systems (2015)

Feature Selection



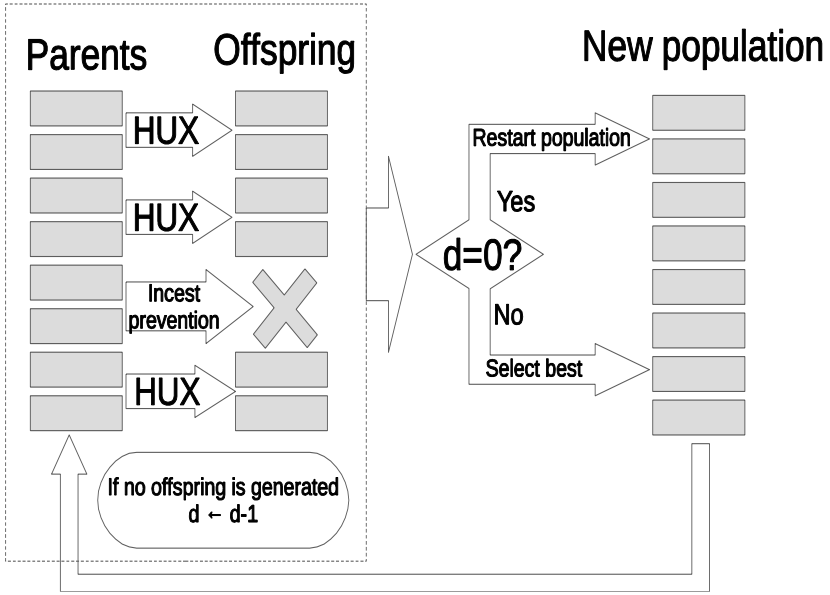
Var. 1. Var. 5 Var. 13

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- The outcome of FS would be:
  - Less data → algorithms could learn quicker
  - Higher accuracy → the algorithm generalizes better
  - Simpler results → easier to understand them

Evolutionary Feature Selection

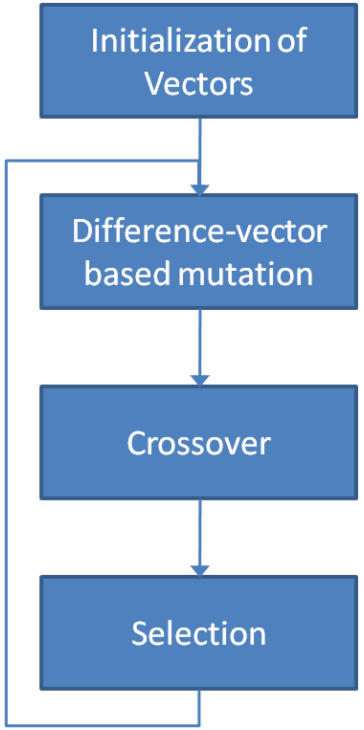
- Each individual represents a set of selected features (**binary vector**).
- The individuals are crossed and mutated to generate new candidate sets of features.
- Fitness function:
  - Classification performance in the training dataset using only the features in the corresponding set.



L. J. Eshelman, **The CHC adaptative search algorithm: How to have safe search when engaging in nontraditional genetic recombination**, in: G. J. E. Rawlins (Ed.), Foundations of Genetic Algorithms, 1991, pp. 265--283.

Evolutionary Feature Weighting

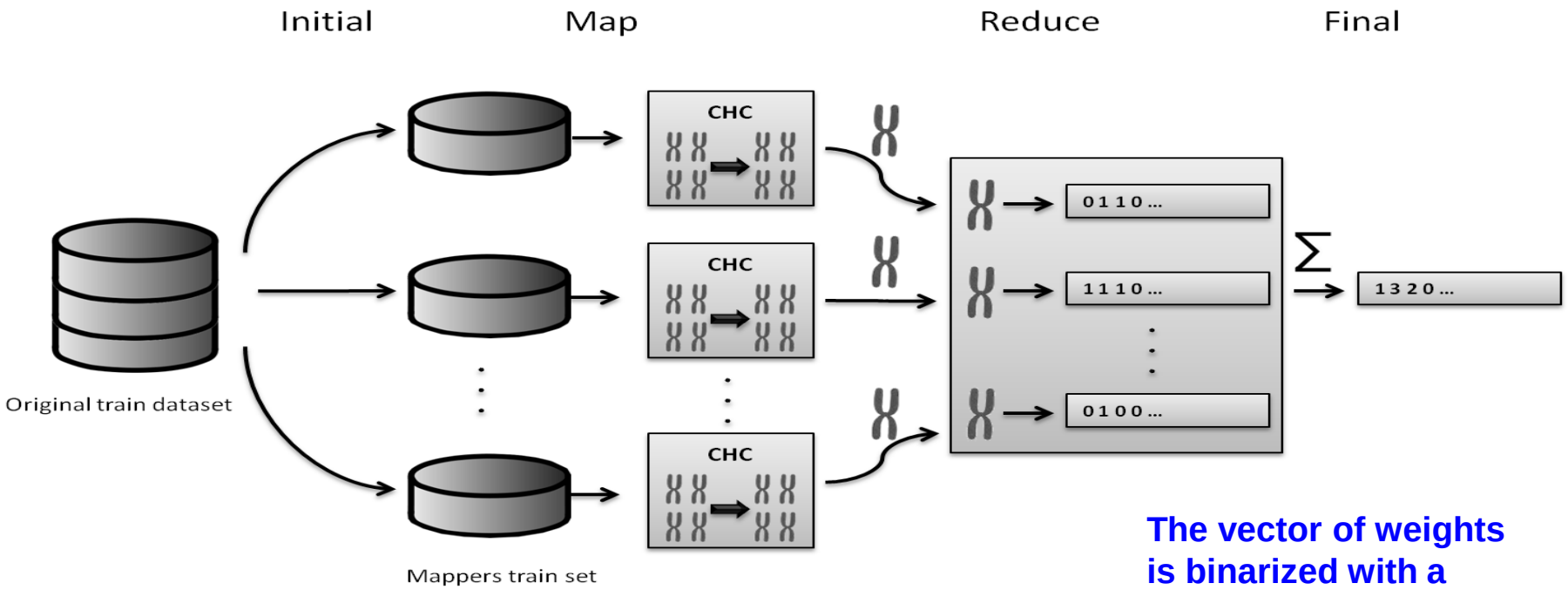
- Each individual represents the importance of the features in the range 0,1 (**real vector**).
- The individuals are crossed and mutated to generate new candidate importances.
- Fitness function:
  - Classification performance in the training dataset using taking into consideration the **weights of the features**.



Neri, F., Tirronen, V., **Scale factor local search in differential evolution**. Memetic Computing 1:2 (2009) 153-171

Evolutionary Feature Selection/Weighting for Big Data

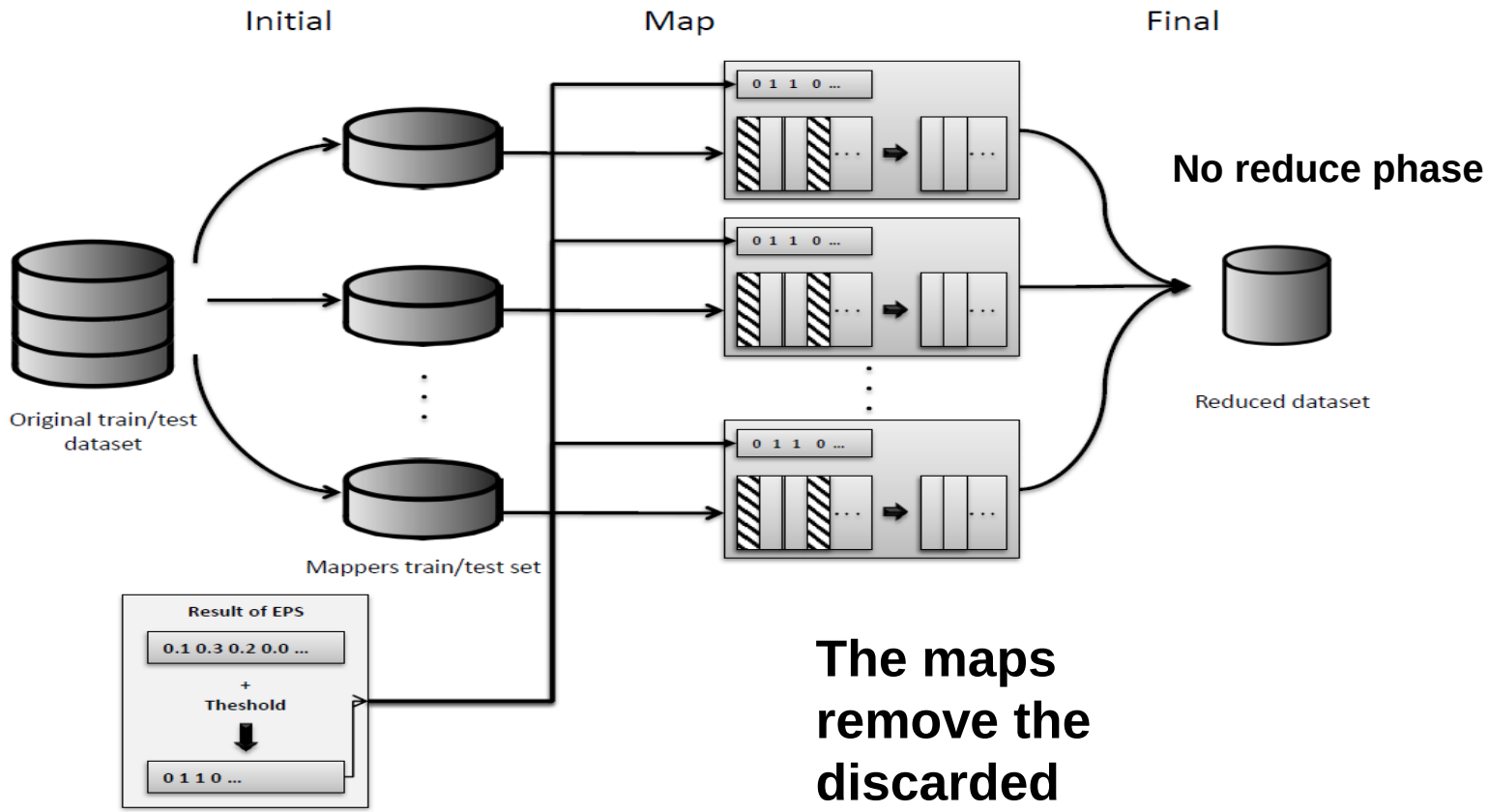
MapReduce EFS process



The vector of weights is binarized with a threshold

Evolutionary Feature Selection/Weighting for Big Data

Dataset reduction

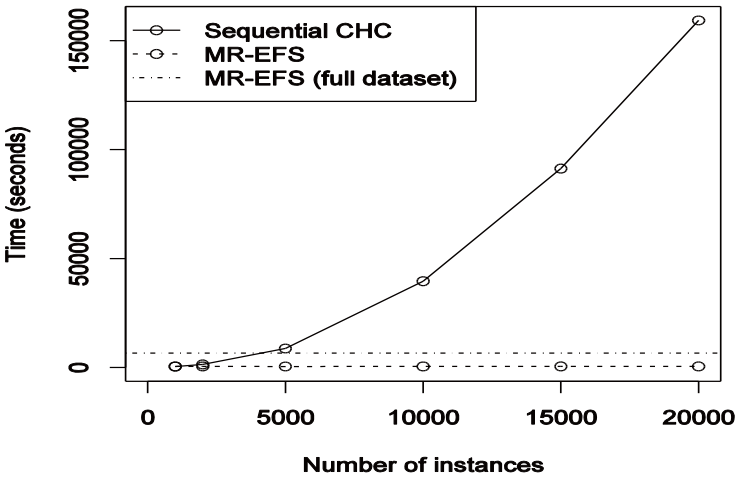


Evolutionary Feature Selection

Experimental Study: EFS scalability in MapReduce

Table 3: Execution times (in seconds) over the epsilon subsets

Instances	Sequential CHC	MR-EFS	Splits
1000	391	419	1
2000	1352	409	2
5000	8667	413	5
10 000	39 576	431	10
15 000	91 272	445	15
20 000	159 315	455	20
400 000	–	6531	512



- CHC is quadratic w.r.t. the number of instances
- Splitting the dataset yields nearly quadratic acceleration

Evolutionary Feature Selection

Experimental Study: Classification

- Two datasets
  - epsilon
  - ECBDL14, after applying Random Oversampling
- The reduction rate is controlled with the weight threshold
- Three classifiers in Spark
  - SVM
  - Logistic Regression
  - Naïve Bayes
- Performance measures
  - $\text{AUC} = \frac{\text{TPR} + \text{TNR}}{2}$
  - Training runtime

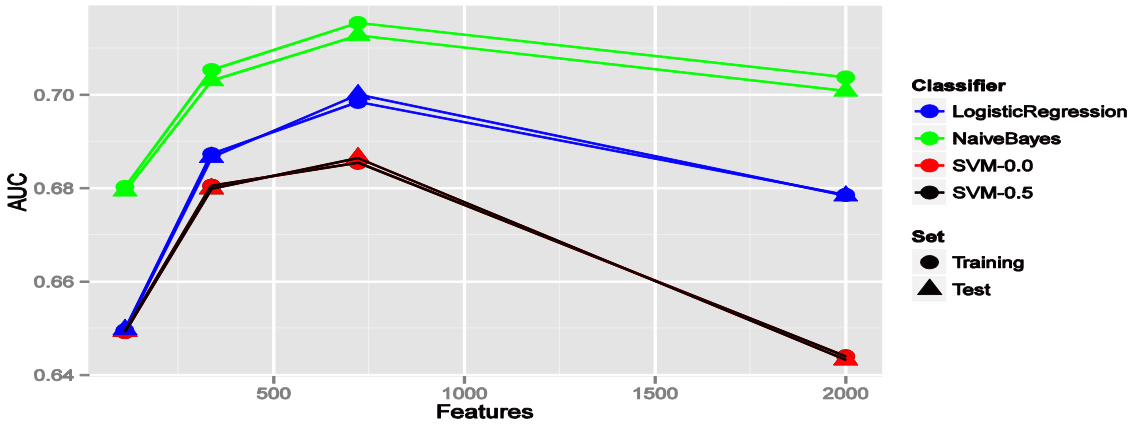
Dataset	Training instances	Test instances	Features	Splits	Instances per split
epsilon	400 000	100 000	2000	512	~780
ECBDL14	31 992 921	2 897 917	631	–	–
ECBDL14-ROS	65 003 913	2 897 917	631	32 768	~1984

Evolutionary Feature Selection

Experimental Study: results

Table 4: AUC results for the Spark classifiers using epsilon

		Logistic Regression		Naive Bayes		SVM ($\lambda = 0.0$)		SVM ($\lambda = 0.5$)	
Threshold	**Features**	Training	Test	Training	Test	Training	Test	Training	Test
0.00	2000	0.6786	0.6784	0.7038	0.7008	0.6440	0.6433	0.6440	0.6433
0.55	721	**0.6985**	**0.7000**	**0.7154**	**0.7127**	**0.6855**	**0.6865**	**0.6855**	**0.6865**
0.60	337	0.6873	0.6867	0.7054	0.7030	0.6805	0.6799	0.6805	0.6799
0.65	110	0.6496	0.6497	0.6803	0.6794	0.6492	0.6493	0.6492	0.6493



Evolutionary Feature Weighting at GECCO-2014

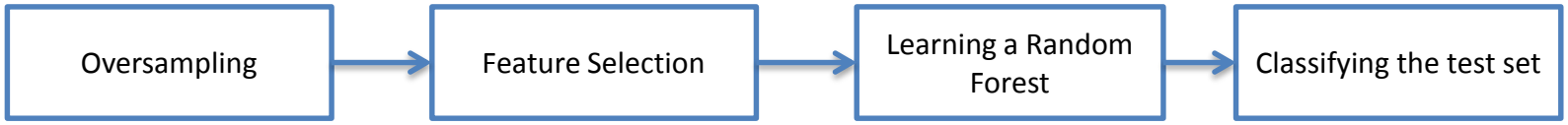
- **Details of the training data**:

❑ 32 million training samples (56.7GB of disk space)

❑ 2.9 million of test samples (5.1GB of disk space)

❑ 631 features (539 real & 92 nominal values)

❑ 2 labels; 98% non-contact samples



I. Triguero, et al. **ROSEFW-RF: The winner algorithm for the ECBDL'14 Big Data Competition: An extremely imbalanced big data bioinformatics problem**. Knowledge-Based Systems (2015)

Results

Team	TPR	TNR	TPR * TNR
Efdamis	0.73043	0.73018	0.53335
ICOS	0.70321	0.73016	0.51345
UNSW	0.69916	0.72763	0.50873
Efdamis-Without FS	0.7041	0.7103	0.500175
HyperEns	0.64003	0.76338	0.48858
PUC-Rio_ICA	0.65709	0.71460	0.46956

I. Triguero, et al. **ROSEFW-RF: The winner algorithm for the ECBDL'14 Big Data Competition: An extremely imbalanced big data bioinformatics problem**. Knowledge-Based Systems (2015)

Evolutionary Feature Selection/Weighting

- The proposed MapReduce processes provide several advantages:
  - It enables tackling Big Data problems
  - The feature weight vector is more flexible than a binary vector
- The data reduction process in MapReduce provides a scalable and flexible way to apply the feature selection/weighting,
- Both the accuracy and the runtime of the classification were improved after the preprocessing.

https://github.com/triguero/MR-EFS