# Apache Spark at Viadeo

*Paris Hadoop User Group - 2014/04/09*
*Viadeo*

@EugenCepoi

*Open source author: Genson*
*Specialized in web scale systems and algorithmics*

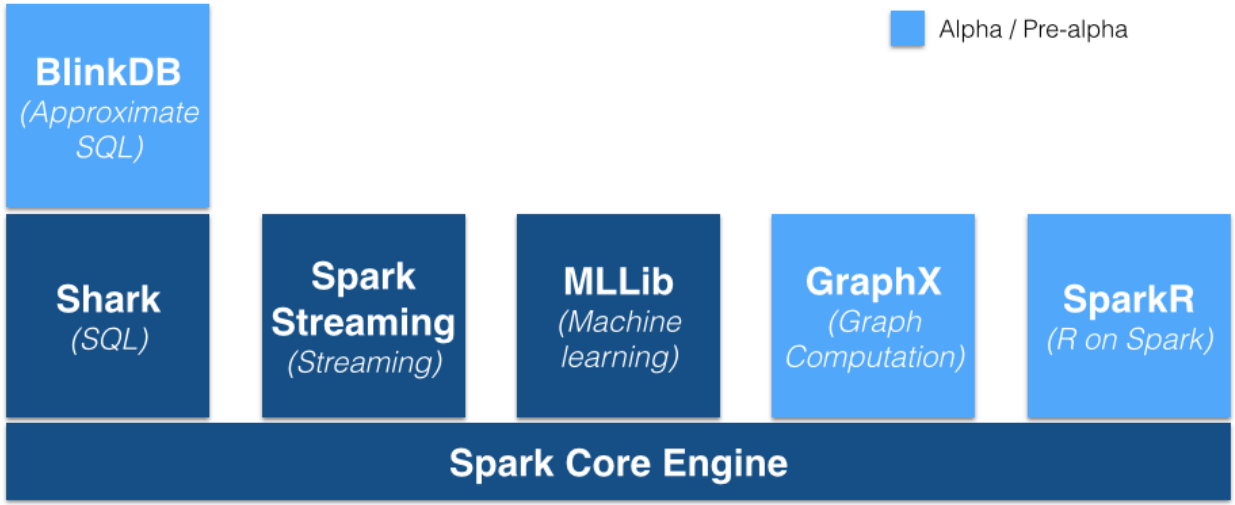
# Apache Spark

- http://spark.incubator.apache.org/

- http://vision.cloudera.com/mapreduce-spark/

- A fast & easy to use engine for distributed computing in Scala (Java & Python support)

# Spark ecosystem




Tachyon

# Why we like Spark

- Scala, functional languages are a great match for typical operations in distributed systems
- Easy to test, most of your code is standard Scala code
- Has already a stack for doing streaming, ML, graph computations
- Allows you to store data in RAM and even without it is fast!
- Provides a collect/broadcast mechanism allowing to fetch all the data on the “Driver” node or to send local data to every worker
- Has an interactive shell (a la Scala), useful to analyse your data and to debug

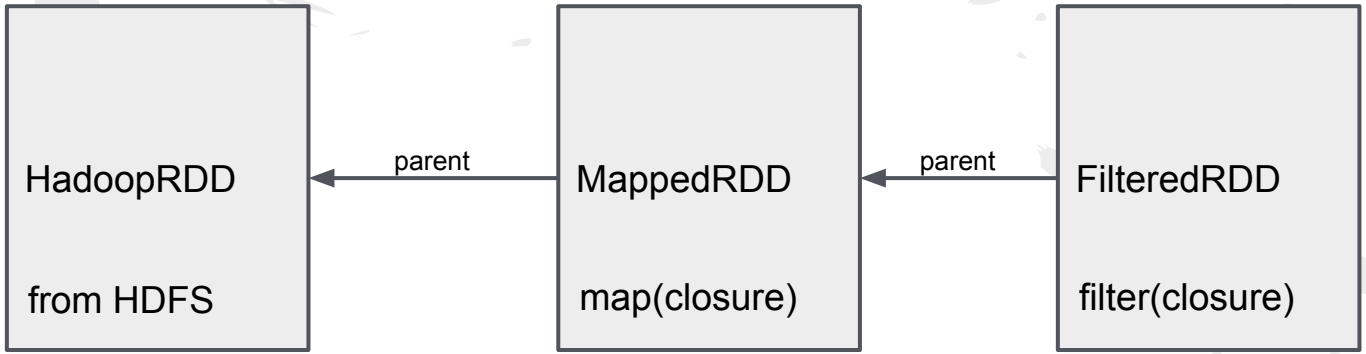
And much more...

# At the core of Spark...

## The Resilient Distributed Dataset - RDD

- Abstraction over partitioned data

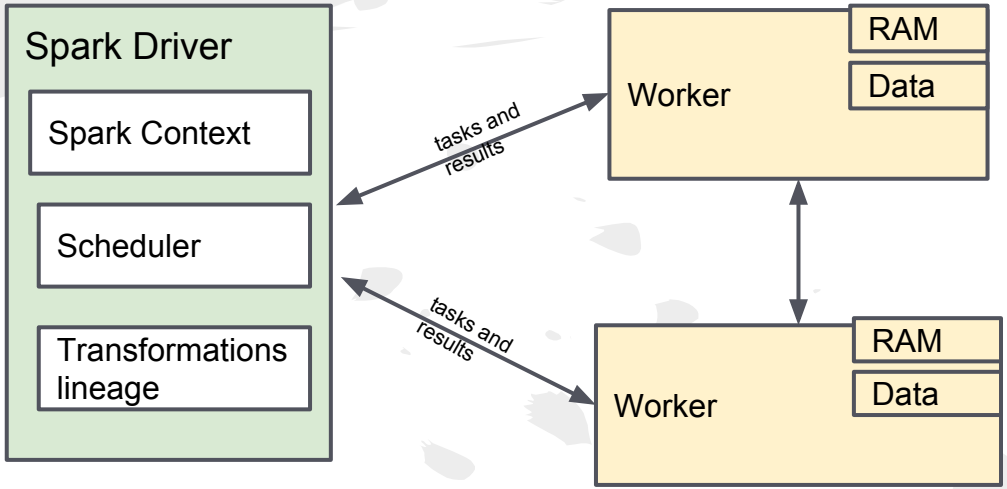
- RDDs are read only and can be built only from a stable storage or by transformations on other RDDs



# At the core of Spark...

**Spark Driver**
- Define & launch operations on RDDs
- Keeps track of RDD lineage allowing to recover lost partitions
- Use the RDD DAG in the Scheduler to define the stages and tasks to be executed

**Workers**
- Read/Transform/Write RDD partitions
- Need to communicate with other workers to share their cache and shuffle data



# Beginning with Spark

- Started to POC with Spark 0.8 summer 2013

- Standalone cluster on AWS in our VPC
  => hostname resolution problems
  => additionnal cost to get the data on S3 and push back
  => network performance issues

- Finally we chose to reuse our on-premise hadoop cluster

# Running Spark on Mesos

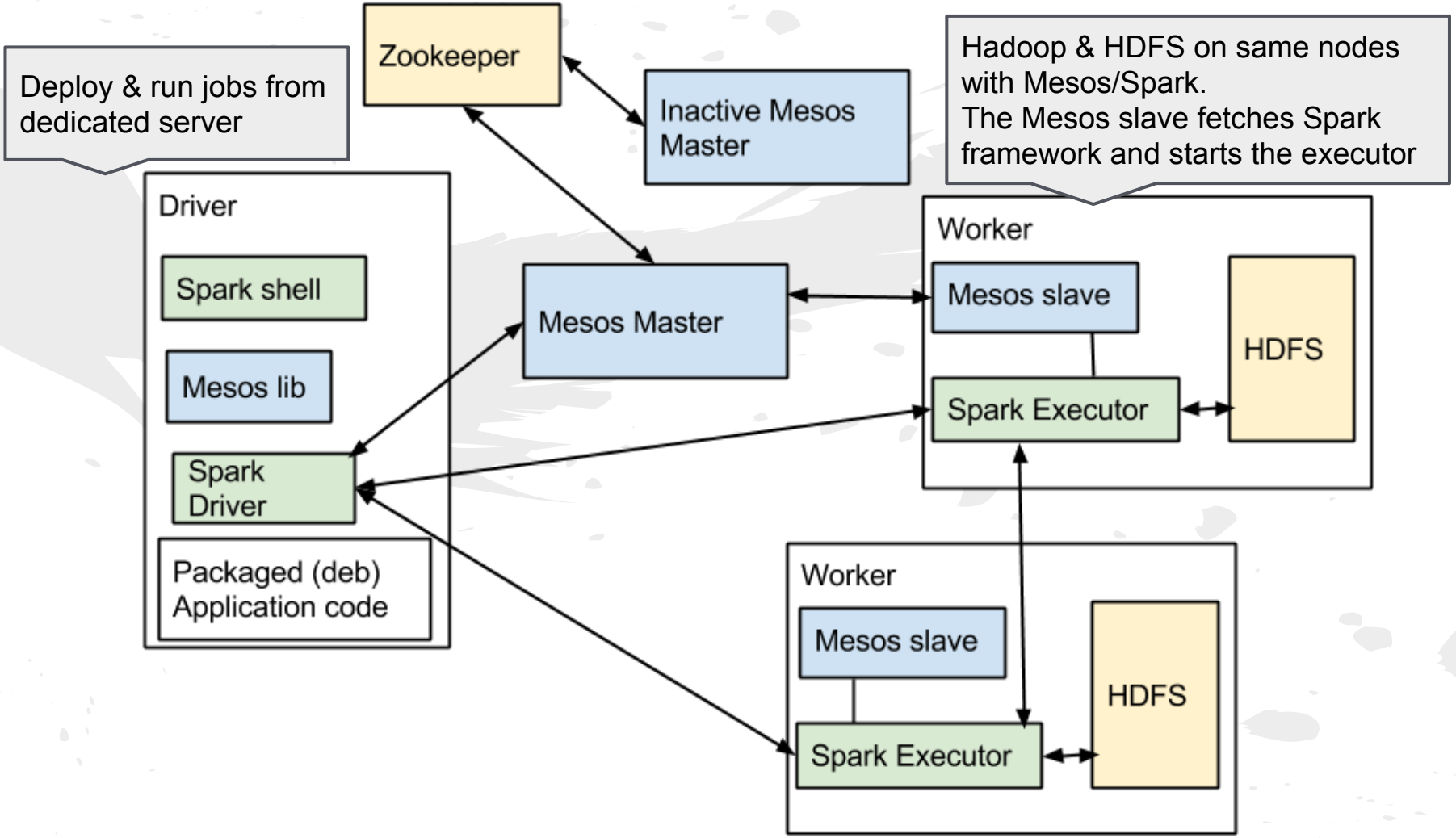
- Cluster manager providing resources (CPU & RAM) isolation and sharing them between applications
- Strong isolation with Linux container groups
- Flexible resource request model:

    => Mesos makes resource offers

    => the framework decides to accept/reject

    => the framework tells what task to run and what amount of the resources will be used

- We can run other frameworks (ex: our services platform)


Deploy & run jobs from dedicated server
Zookeeper
Inactive Mesos Master
Hadoop & HDFS on same nodes with Mesos/Spark.
The Mesos slave fetches Spark framework and starts the executor
Driver
Spark shell
Mesos lib
Spark Driver
Packaged (deb) Application code
Mesos Master
Worker
Mesos slave
HDFS
Spark Executor
Worker
Mesos slave
HDFS
Spark Executor

# Some repetitive tasks

- Repeating the creation of a new project with packaging, scripts, config, etc. is boring

- Most jobs take as input Avro files from HDFS (Sqoop export, Event logs) and output Avro

- They often have common structure and testing utilities

- Want to get latest data or all data in a time range

# Creating Sparky

- To improve our productivity we created Sparky, a thin library around Spark

=> read/write Avro from HDFS, work with partitioned data by time, testing, benchmarking ML utilities (ROC & AUC)

=> a job structure allowing to standardize jobs design and add some abstractions around SparkContext and job configuration - could allow us to have shared RDD (Job Server from Ooyala?)

=> a maven archetype to generate new spark projects ready for production (debian packaging, scripts, configuration and job template)

# Problems we had

- “No more space left…” => spark.local.dir pointing to /tmp … oops

- NotSerializableException
    => closures referencing variables outside the function scope
    => classes not implementing Serializable, Enable Kryo serialization

- Kryo throwing ArrayOutOfBoundsException for large objects, increase spark.kryoserializer.buffer.mb

- Loosing still in use broadcasted data due to spark.cleaner.ttl

# Problems we had

- Reading Avro GenericRecord instead of SpecificRecord

    => Avro is not finding the generated classes in your application

    => the spark assembly with hadoop includes avro, it is loaded by a different classloader than the one used for the application

    => tweaking with Avro API to use the task classloader

    => alternative solution: repackage Spark without avro and include avro in your application fat jar

# Things we must do

- Use a scheduler such as Chronos, allowing to handle inter job dependencies

- Migrate Hadoop jobs to use Mesos or YARN (and use the same cluster manager for Spark jobs)

- Automatic job output validation features

    ex: check output size is close to expected

# Systems we built with Spark

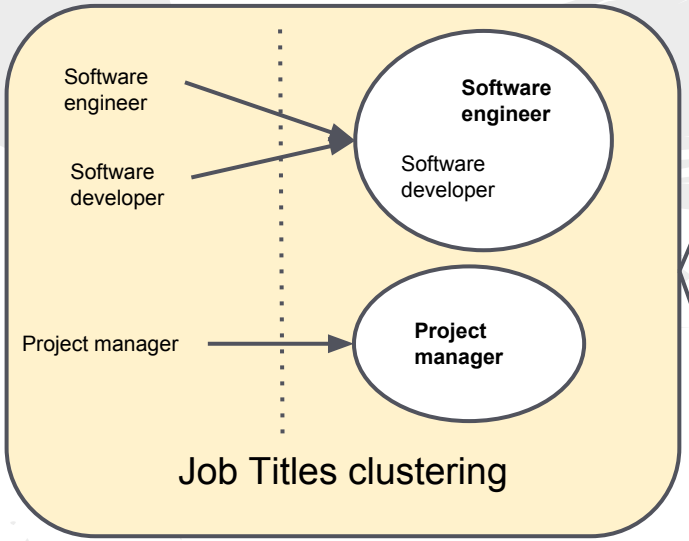
**Job offer click prediction (in emails)**

Members segmentation & targeting

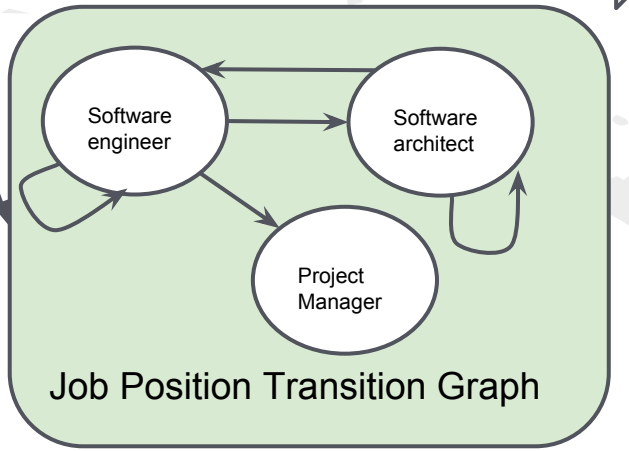
# Job offer click prediction

- Increase click rates on job offers in our career mail
- Current solution is using Solr and searches best matches in textual content between members and job offers
- POC a click prediction engine using ML algorithms and Spark
- Achieving ~5-7% click increase in our career mail

# Algorithm overview




Compute input variables:
- Distance between Member attributes and Job offer attributes
- Job offer click count, etc




Linear regression to score offers:
x = input variables
y = boolean, has clicked

model: $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$

# Spark/HDFS/HBase at the core of our system

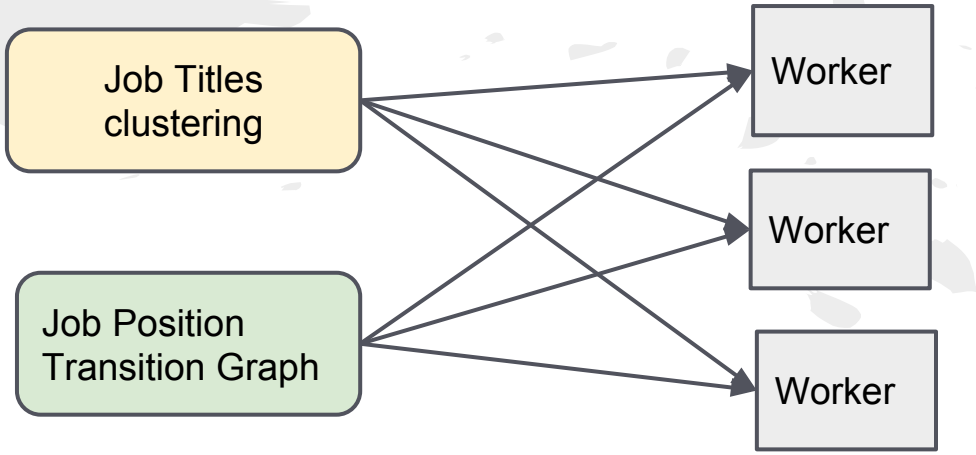
- The prediction engine is made of several Spark jobs working together

- The job computing the predictions does not have algorithmic logic:
    => fetches past predictions/clicks from HBase
    => uses an algorithm implementation and trains it with the data
    => stores the predictions back to HBase

- Our algorithms implement a contract (train, predict) and contain all the logic
    => allows us to plug new algorithms in the prediction job
    => and to have a automatic benchmark system to measure algorithms performance (really nice! :))

# Spark/HDFS/HBase at the core of our system

The broadcast feature of Spark helped us a lot to share small to medium size data, by broadcasting it to all workers



# Spark/HDFS/HBase at the core of our system

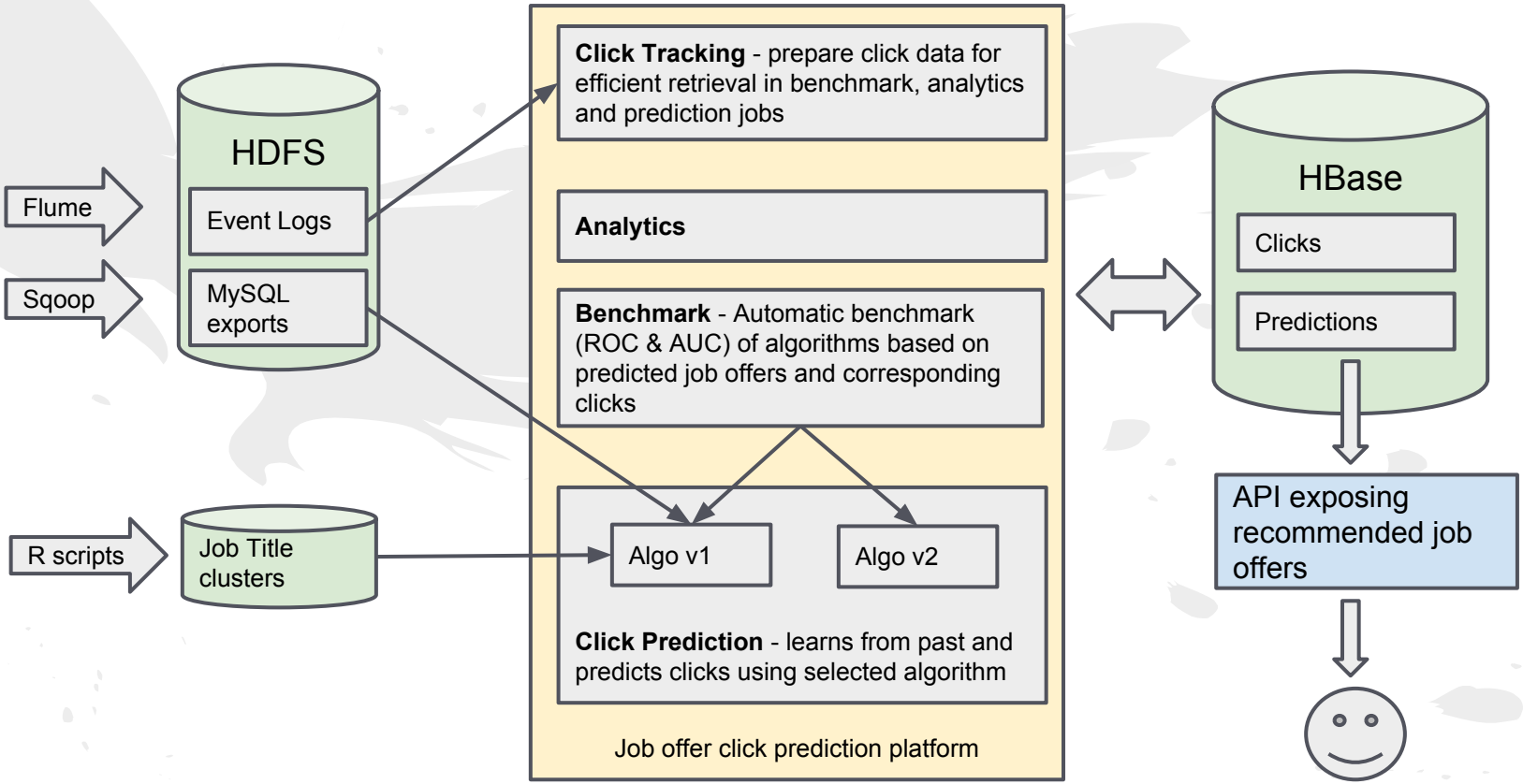
- We stored only Avro on HDFS, partitioned by date

- We stored only primitive types (int, long, byte) in HBase and designed RowKeys allowing us to do efficient partial range scans

=> decile#prediction_date#memberId, c:click_target#click_time, targetId

=> the decile was used as sharding key

=> prediction_date and click_target were used to do partial range scans for constructing benchmark datasets and filter the specific kind of events we wanted to predict (click on the job offer, click on the company name, etc)


Flume
Sqoop
HDFS
Event Logs
MySQL exports
R scripts
Job Title clusters
Click Tracking - prepare click data for efficient retrieval in benchmark, analytics and prediction jobs
Analytics
Benchmark - Automatic benchmark (ROC & AUC) of algorithms based on predicted job offers and corresponding clicks
Algo v1
Algo v2
Click Prediction - learns from past and predicts clicks using selected algorithm
Job offer click prediction platform
HBase
Clicks
Predictions
API exposing recommended job offers

# Systems we built with Spark

Job offer click prediction (in emails)

**Members segmentation & targeting**

# Member Segmentation

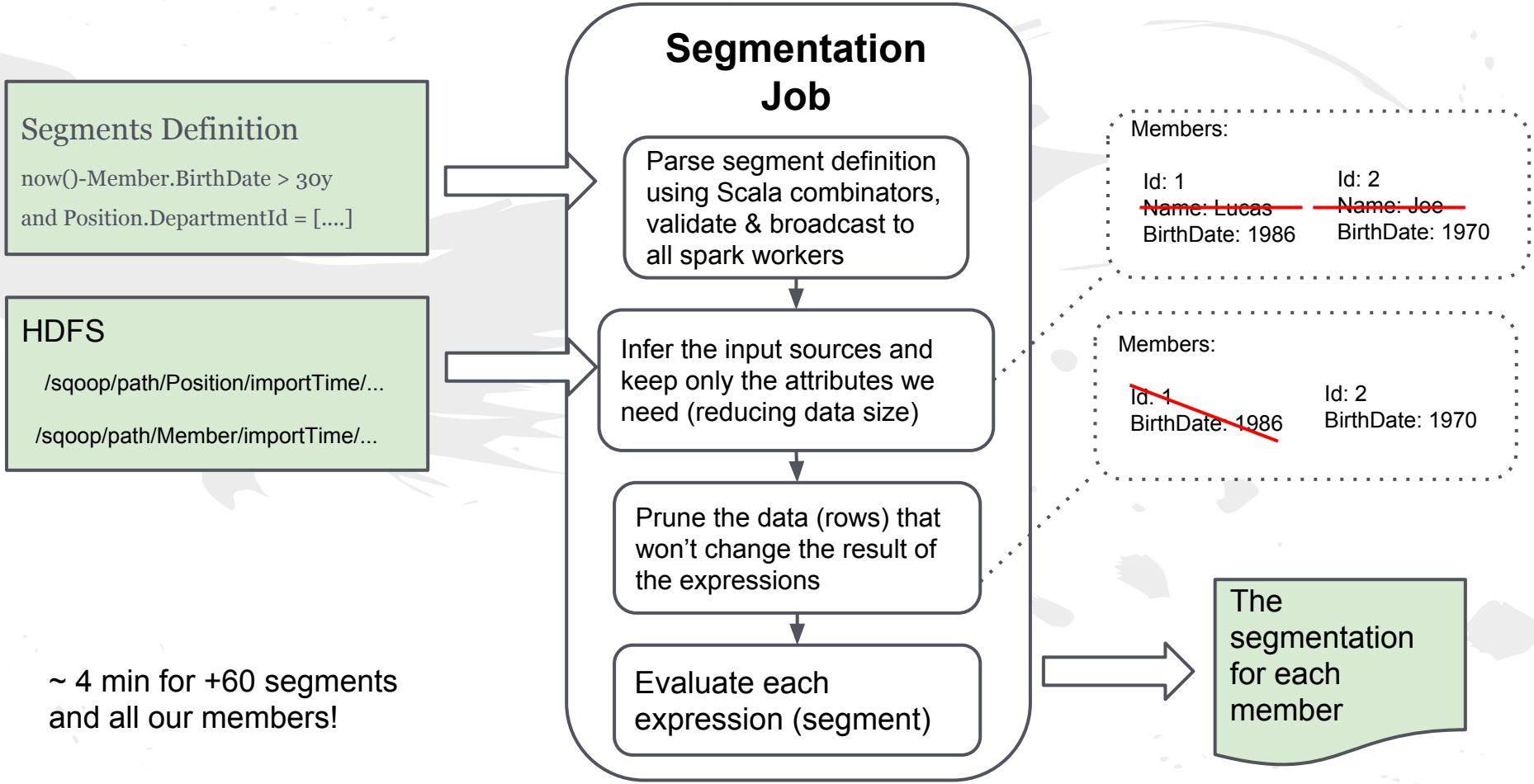
- What is a segment? ex: All male customers older than 30 working in IT

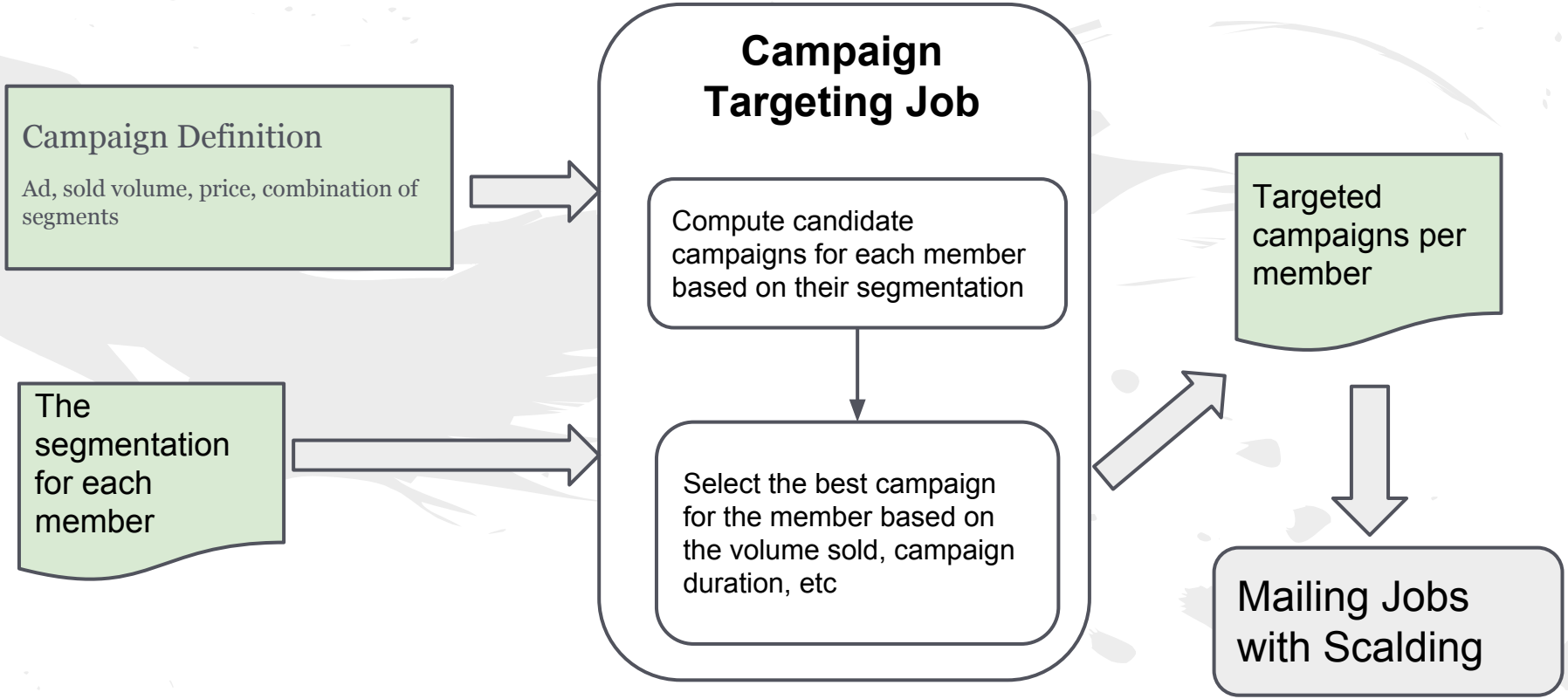
- Our traffic team responsible for doing campaign targeting was using MySQL

- Their queries were taking hours to complete

- Was not industrialized, thus not usable outside the ad targeting scope

- Our goal: build a system that allows to
    => easily express segments
    => compute the segmentation for all our members quickly
    => use it to do targeting in emails, select AB testing target, etc


Segments Definition
now()-Member.BirthDate > 30y
and Position.DepartmentId = [....]
HDFS
/sqoop/path/Position/importTime/...
/sqoop/path/Member/importTime/...
Segmentation Job
Parse segment definition using Scala combinators, validate & broadcast to all spark workers
Infer the input sources and keep only the attributes we need (reducing data size)
Prune the data (rows) that won't change the result of the expressions
Evaluate each expression (segment)
Members:
Id: 1
Name: Lucas
BirthDate: 1986
Id: 2
Name: Joe
BirthDate: 1970
Members:
Id: 1
BirthDate: 1986
Id: 2
BirthDate: 1970
The segmentation for each member
~ 4 min for +60 segments and all our members!


Campaign Definition
Ad, sold volume, price, combination of segments
The segmentation for each member
Campaign Targeting Job
Compute candidate campaigns for each member based on their segmentation
Select the best campaign for the member based on the volume sold, campaign duration, etc
Targeted campaigns per member
Mailing Jobs with Scalding

# Next steps

- Build inverted indexes for computed segmentation (ElasticSearch?)

- Expose the indexes to BI for analytics purpose

- Expose the indexes to Viadeo customers allowing them to create targeted campaigns online

# Questions?

Or send them at cepoi.eugen@gmail.com

# Thanks :) _

*Viadeo Data mining & machine learning team*
*@EugenCepoi*