Spark Tutorial 1

Theoretical notions and practice

Goals of Spark

Works well

- with GBs, TBs, or PBs or data
- With different storage media (RAM, HDDs, SSDs)
- from low-latency streaming jobs to long batch jobs

Motivations: Sort Benchmark

Originally sponsored by Jim Gray to measure advancements in software and hardware in 1987

Participants often used purpose-built hardware/software to compete

- Large companies: IBM, Microsoft, Yahoo, ...
- Academia: UC Berkeley, UCSD, MIT, ...

Sort Benchmark

DIAPRISM Hardware Sorter
— Sort a Million Records Within a Second —

Shinsuke Azuma, Takao Sakuma, Tetsuya Takeo, Takaaki Ando, Kenji Shirai

Mitsubishi Electric Corp.

Mitsubishi Electric Corp. has released the DIAPRISM hardware sorter for OLAP (On-Line Analytical Processing) and data warehouse applications. It enables high-speed sorting on a commercial PC server. The sorter employs a pipeline merge sort algorithm. It consists of multiple sort processors connected linearly and necessary amount of memory for each processor. The architecture of the sort processor is capable of sorting 4GB of data at one time. The sorter has achieved Datamation benchmark record, sorting a million 100-byte records in 0.998 second.

1MB -> 100MB -> 1TB (1998) -> 100TB (2009)

Sorting 100 TB and 1 PB

Participated in the most challenging category

- Daytona GraySort
- Sort 100TB in a fault-tolerant manner
- 1 trillion records generated by a 3rd-party harness (10B key + 90B payload)

Set a new world record (tied with UCSD)

- Saturated throughput of 8 SSDs and 10Gbps network
- 1st time public cloud + open source won

Sorted 1PB on their own to push scalability

On-Disk Sort Record: Time to sort 100TB

2013 Record: Hadoop

2100 machines

72 minutes

2014 Record: Spark

207 machines

23 minutes

Also sorted 1PB in 4 hours

dbTrento

Is Map Reduce paradigm sufficient ?

No!!!

- Map and reduce operations are too strict
- Write and read only on disk (too slow)
- New needs: **graph analysis**, **machine learning**, etc.

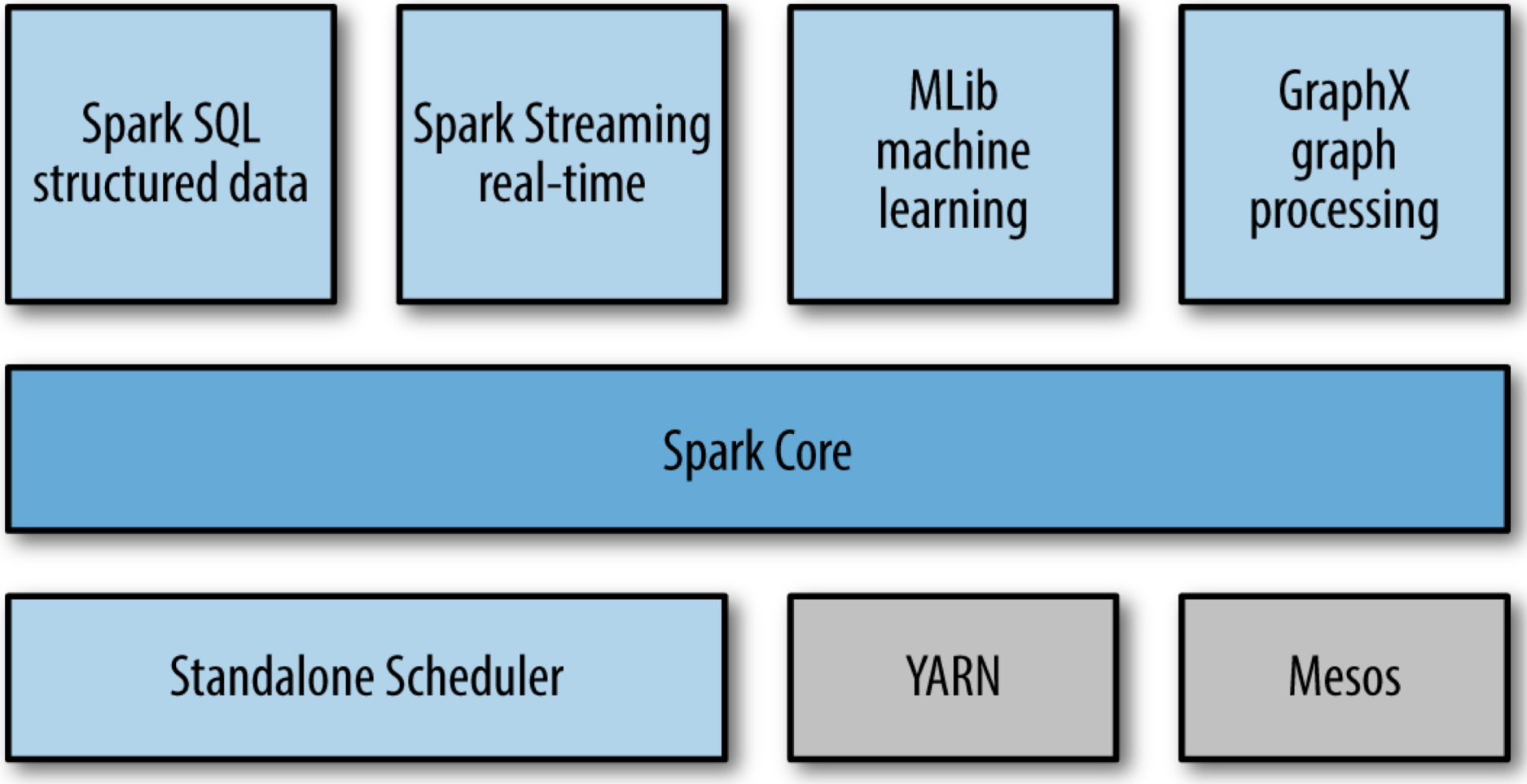
New engines

- **Flink**
- **Storm**
- **Spark**

Advantages

- New operations (**flatmap**, **cogroup**, **join**, ...)
- In memory computation (up to 10x-100x than Hadoop)

Spark Ecosystem



Spark Core

Spark basics

Resilient Distributed Dataset (RDDs)

- Parallel fault-tolerant collection of data
- Created from:
  - a collection and then parallelized
  - an external source: S3, HDFS, Cassandra, ...

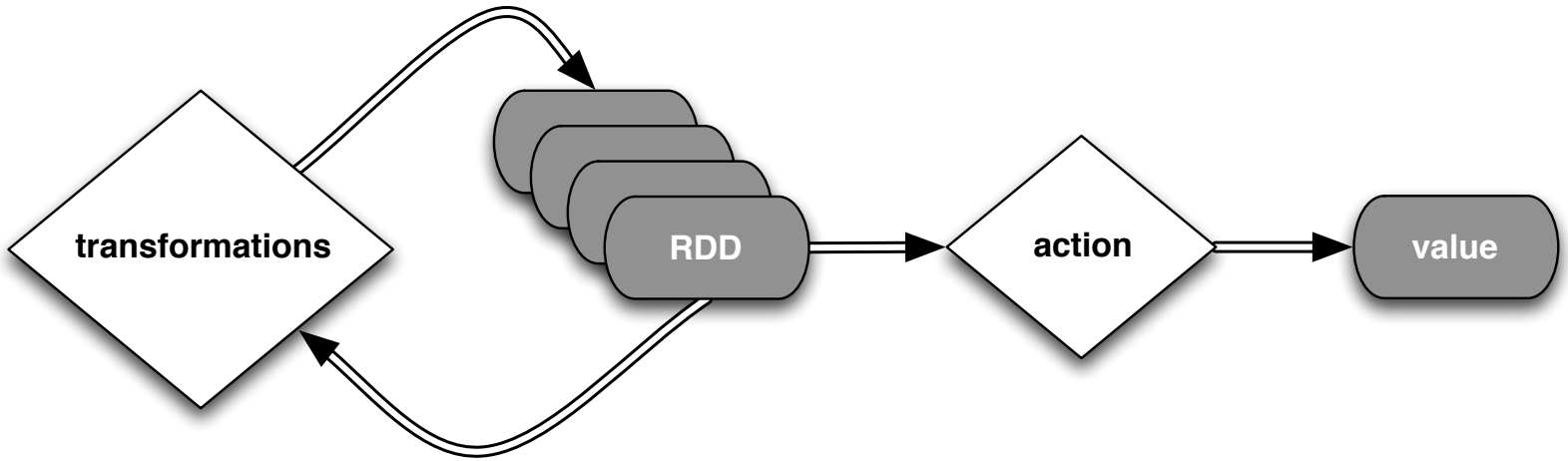
RDD operations

- **Transformations**
  - Lazy evaluation
  - Evaluated only when an action require it
- **Actions**
  - Force compute previous transformations

Spark basics

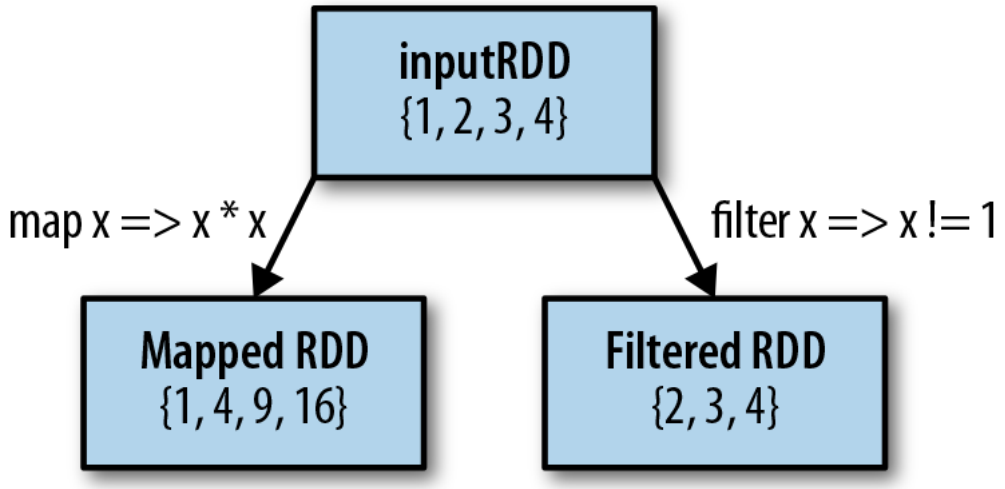
Resilient Distributed Dataset (RDDs)

- Parallel fault-tolerant collection of data
- Created from:
  - a collection and then parallelized
  - an external source: S3, HDFS, Cassandra, ...



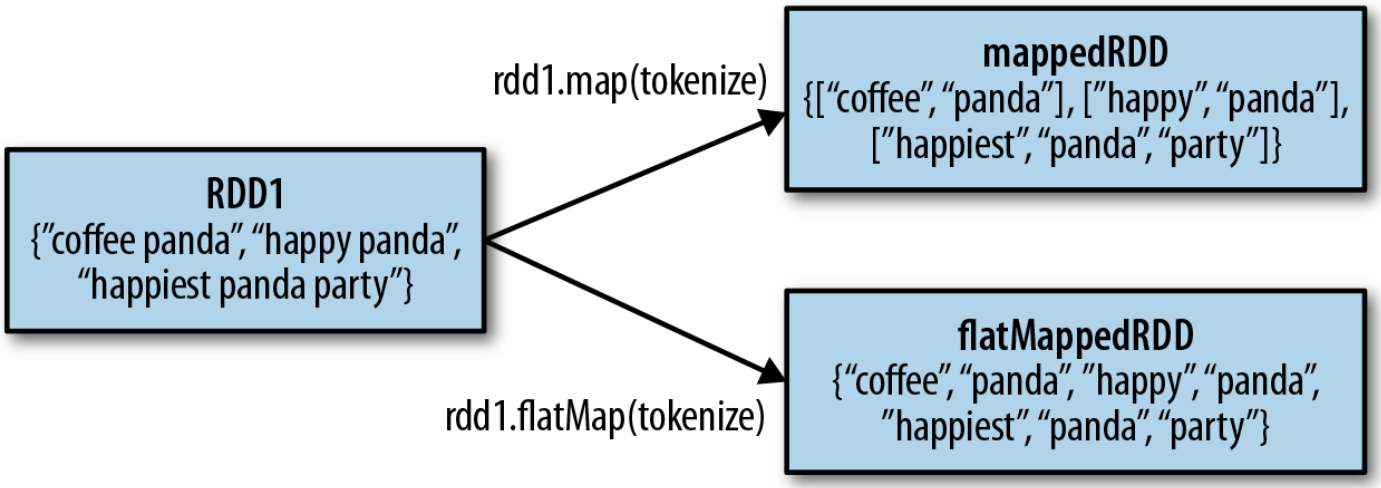
Spark Transformations

Map & Filter



Map vs. FlatMap

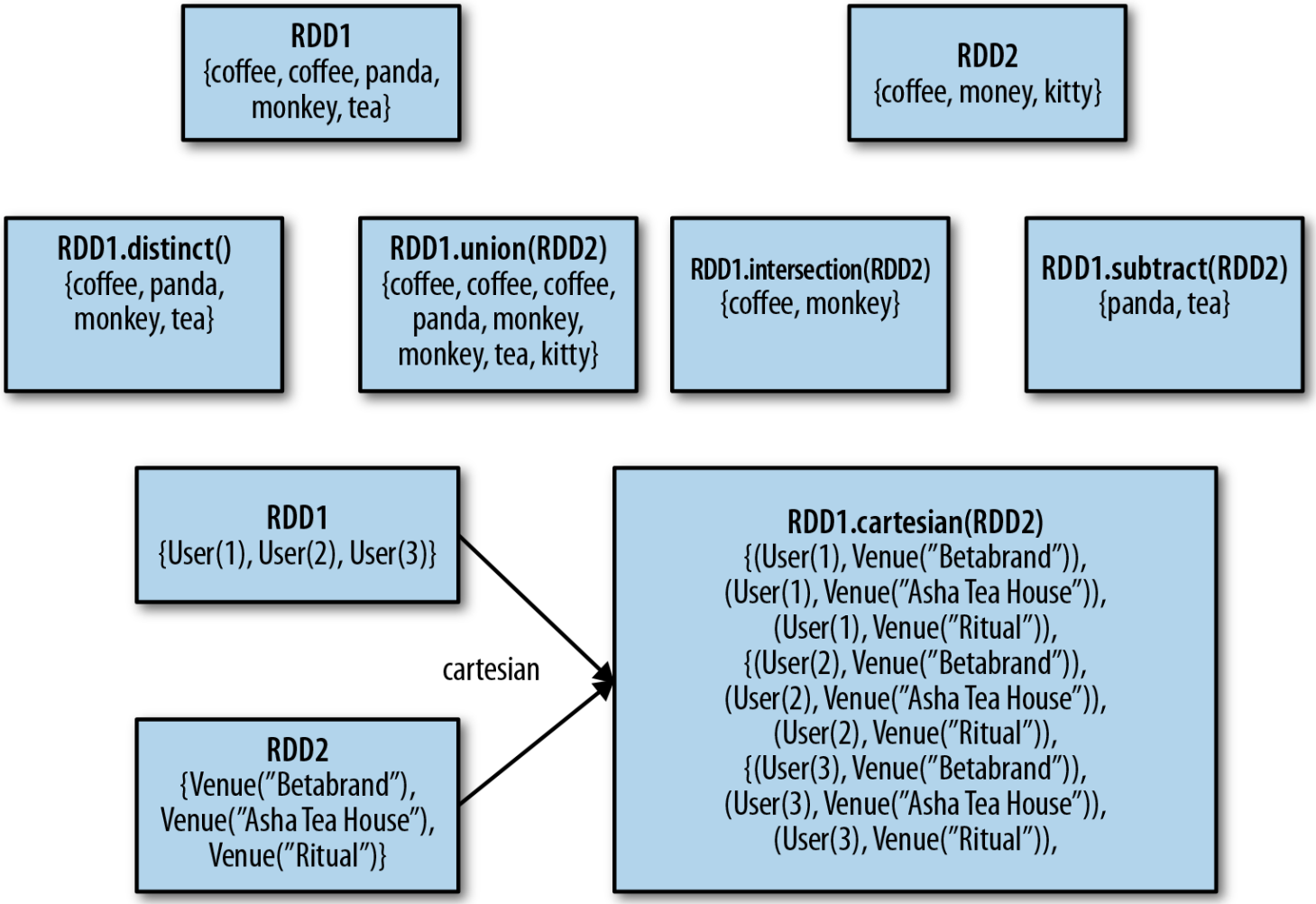
tokenize("coffee panda") = List("coffee", "panda")



dbTrento

Spark Transformations

Some “list like” operations



Spark Actions

Resilient RDD containing {1, 2, 3, 3}

Some operations:

Operation	Purpose	Example	Results
collect()	Return all elements	rdd.collect()	{1, 2, 3, 3}
count()	Count all elements	rdd.count()	4
countByValue()	Count elements by value	rdd.countByValue()	{(1, 1), (2, 1), (3, 2)}
take(num)	Return a number of elements equal to num	rdd.take(2)	{1, 2}
top(num)	Return the top N values	rdd.top(2)	{3, 3}
reduce(func)	Reduce	rdd.reduce((x, y) => x + y)	9

Load data into RDDs

Parallelized Collections

- Existing collections from your driver program
- Variable, Lists, Sets

External Datasets

- **S3**
- HDFS
- Cassandra, MongoDB, and others
- **MySQL, Postegres, others**

RDDs and Partitions

Partitions

- RDDs partitioned through different nodes
- Number of partitions is an important value
- Typically 2-4 partitions per CPU
- Done automatically by default

Transformations

- **By default applied on the RDD**
- You can apply transformations on partitions
- User defined partitions functions
- Transformation (***map*** vs **mapPartition**)

Actions and Transformations: Lazy Computation

- Transformations create a new dataset from an existing one (map is a transformation)
- Actions return a value to the driver program after running a computation on the dataset (reduce is an action)

- Transformations are lazy, it means that they are only computed when an action requires a result to be returned to the driver program

x = rdd.map(…).map(…)	No computation
print(x.reduce(…))	The computation starts from rdd
print(x.map(…).count())	The computation starts from rdd

Lazy evaluation drawbacks

Drawbacks

- Many partitions are computed multiple times
- Longer execution time
- Seems so nice from the outside

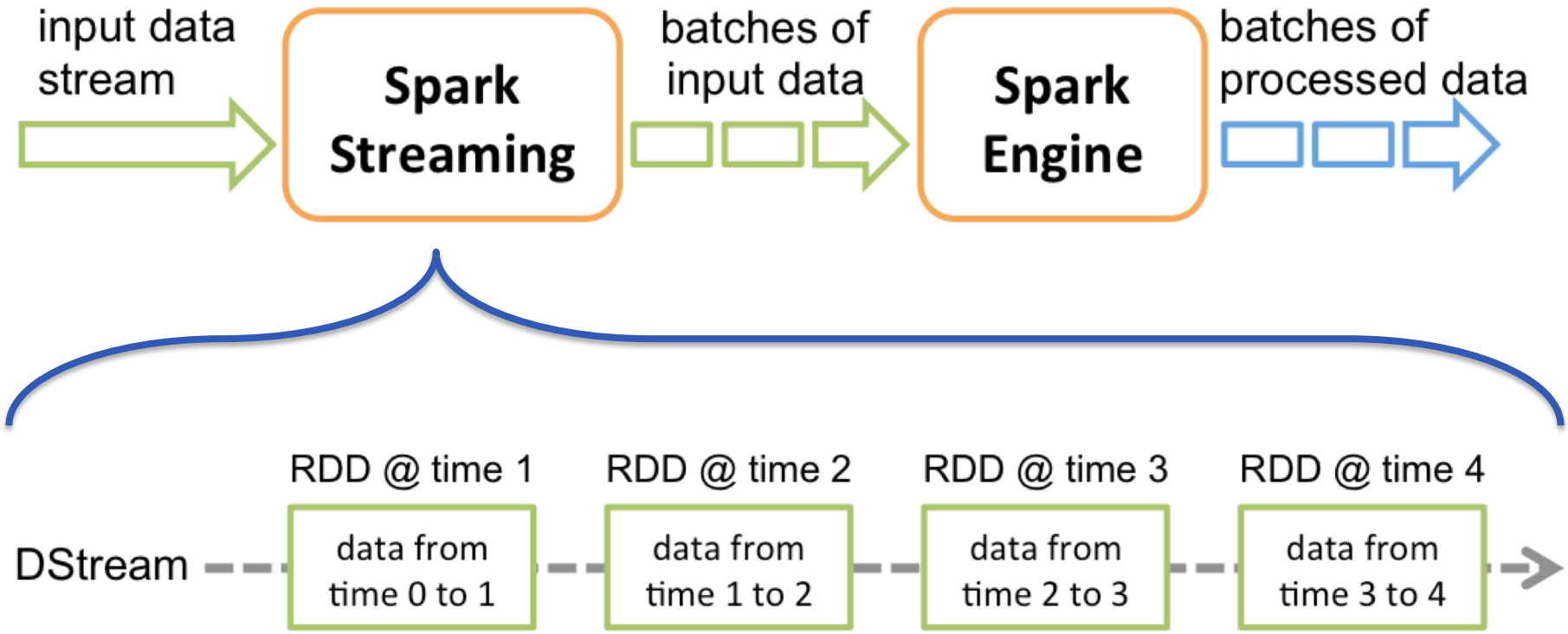
Solution

- **RDD Persistence**
- Caching RDDs in memory across different operations
- Significant speed up (at least 10x)
- Different level of persistence (Main memory, local disk, mixed)

Spark Streaming

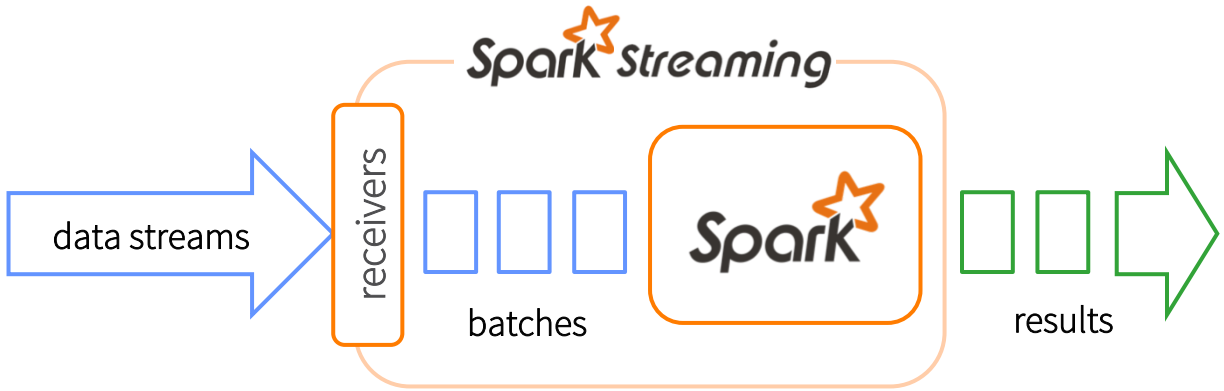
General overview

Architecture



Main features

- *Receivers* receive data streams and chop them up into batches



- High level api (joins, sliding windows, etc)
- Fault-tolerant
- Integration (MILib, SQL, DataFrames, GraphX)

Spark SQL & DataFrames

About it

- Spark SQL
  - Runs SQL / HiveQL queries
  - Connect existing BI tools to Spark through JDBC
  - Bindings in Python, Scala, Java, and R

- DataFrames & Datasets
  - A distributed collection of rows organized into named columns.
  - An abstraction for selecting, filtering, aggregating and plotting structured data (cf. R, Pandas).

Write less code: Input & Output

Unified interface to reading/writing data in a variety of formats

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

read and **write** functions create new builders for doing I/O

Builder methods specify:
- Format
- Partitioning
- Handling of existing data

load(...), **save(...)** or **saveAsTable(...)** finish the I/O specification

High level operations

Solve common problems concisely using DataFrame functions:

- Selecting columns and filtering
- Joining different data sources
- Aggregation (count, sum, average, etc)
- Plotting results with Pandas

E.g. Compute an average


```
private IntWritable one =
  new IntWritable(1)
private IntWritable output =
  new IntWritable()
proctected void map(
    LongWritable key,
    Text value,
    Context context) {
  String[] fields = value.split("\t")
  output.set(Integer.parseInt(fields[1]))
  context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()

protected void reduce(
    IntWritable key,
    Iterable<IntWritable> values,
    Context context) {
  int sum = 0
  int count = 0
  for(IntWritable value : values) {
     sum += value.get()
     count++
    }
  average.set(sum / (double) count)
  context.Write(key, average)
}
```


```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [x.[1], 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

E.g. Compute an average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
    .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
    .collect()
```

Using DataFrames

```
sqlCtx.table("people") \
    .groupBy("name") \
    .agg("name", avg("age")) \
    .collect()
```

Using API Docs

- Python
- Scala
- Java
- R

Spark MILib

Mllib's Mission

- Capable of learning from large-scale datasets
- Easy to build machine learning applications

- Good coverage of algorithms

classification	feature extraction, selection
regression	statistics
clustering	linear algebra
recommendation	frequent itemsets

Example: Text Classification

Goal: Given a text document, predict its topic.

Features

```
Subject: Re: Lexan Polish?
Suggest McQuires #1 plastic
polish.  It will help somewhat
but nothing will remove deep
scratches without making it
worse than it already is.
McQuires will do something...
```

Label

1: about science
0: not about science

CTR, inches of rainfall, ...

text, image, vector, ...

Example: Training & Testing

Training

Given labeled data:
RDD of (features, label)

```
Subject: Re: Lexan Polish?
Suggest McQuires #1 plastic
polish.  It will help...
```
Label 0

```
Subject: RIPEM FAQ
RIPEM is a program which
performs Privacy Enhanced...
```
Label 1

...

Learn a model.

Testing/Production

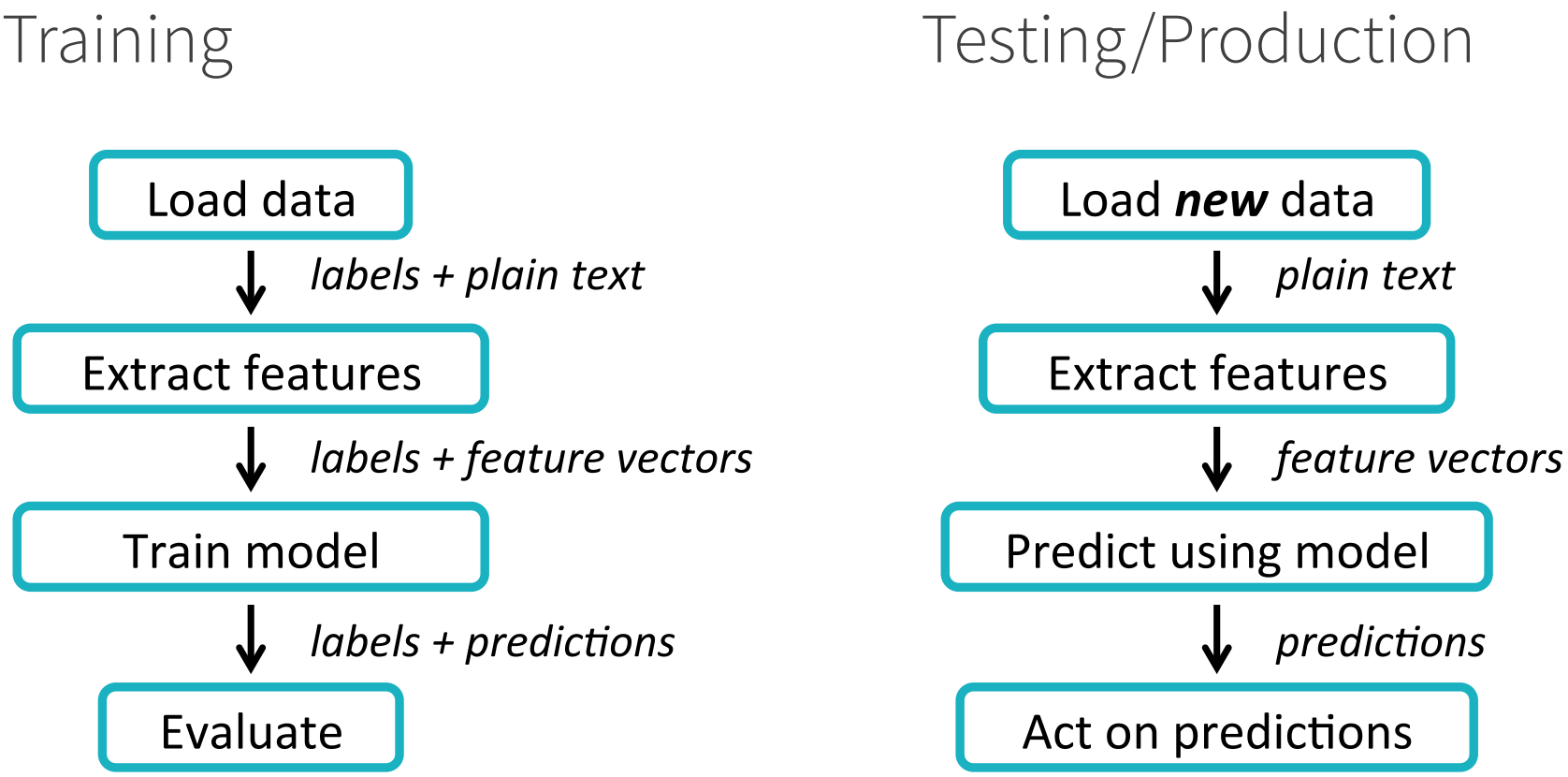
Given new unlabeled data:
RDD of features

```
Subject: Apollo Training
The Apollo astronauts also
trained at (in) Meteor...
```
→ Label 1

```
Subject: A demo of Nonsense
How can you lie about
something that no one...
```
→ Label 0

Use model to make predictions.

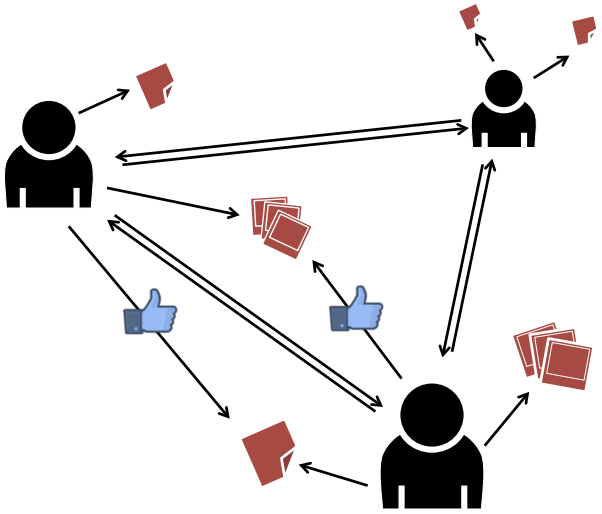
Example: ML Workflow



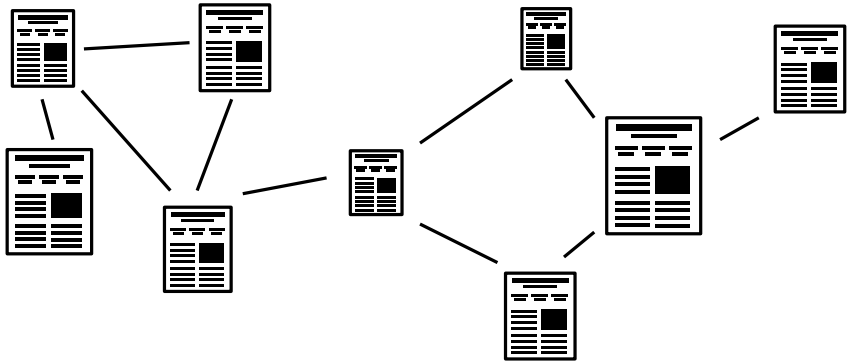
Spark GraphX

Graph Everywhere

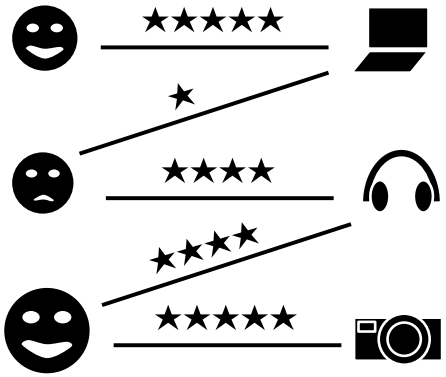
Social Networks



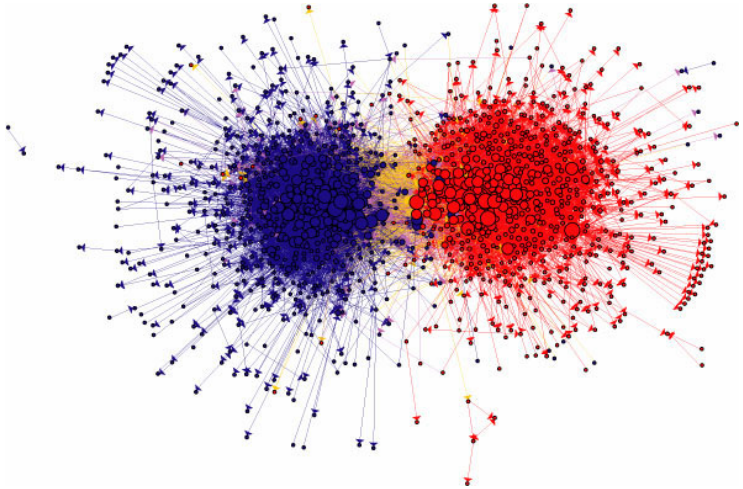
Web Graphs



User-Item Graphs

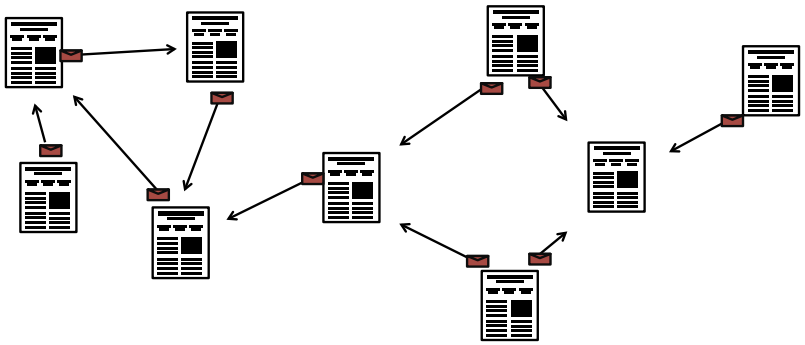


Web Graphs

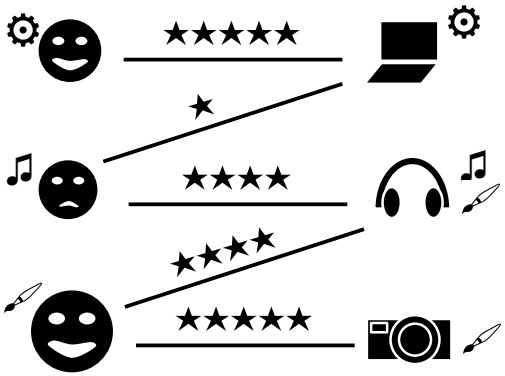


Algorithms everywhere

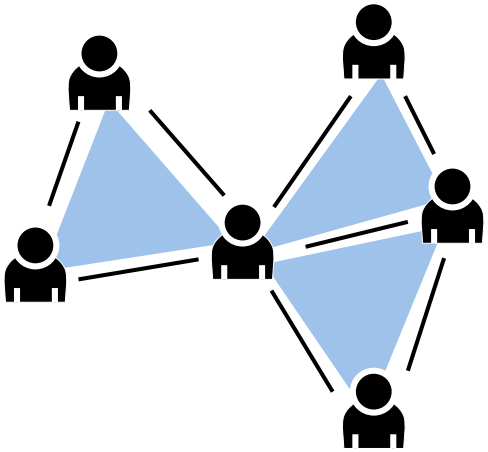
PageRank



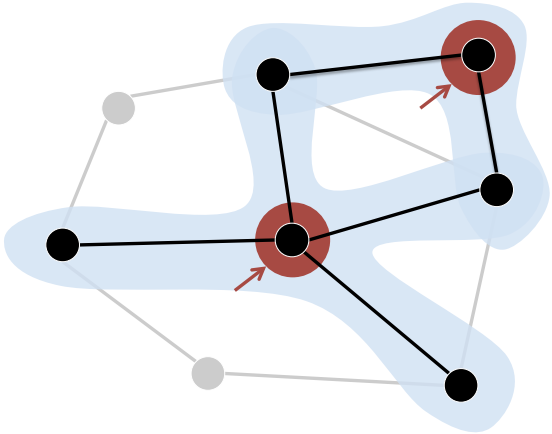
Collaborative Filtering



Triangle Counting



The Graph-Parallel Pattern



List of algorithms

Collaborative Filtering
- » Alternating Least Squares
- » Stochastic Gradient Descent
- » Tensor Factorization

Structured Prediction
- » Loopy Belief Propagation
- » Max-Product Linear Programs
- » Gibbs Sampling

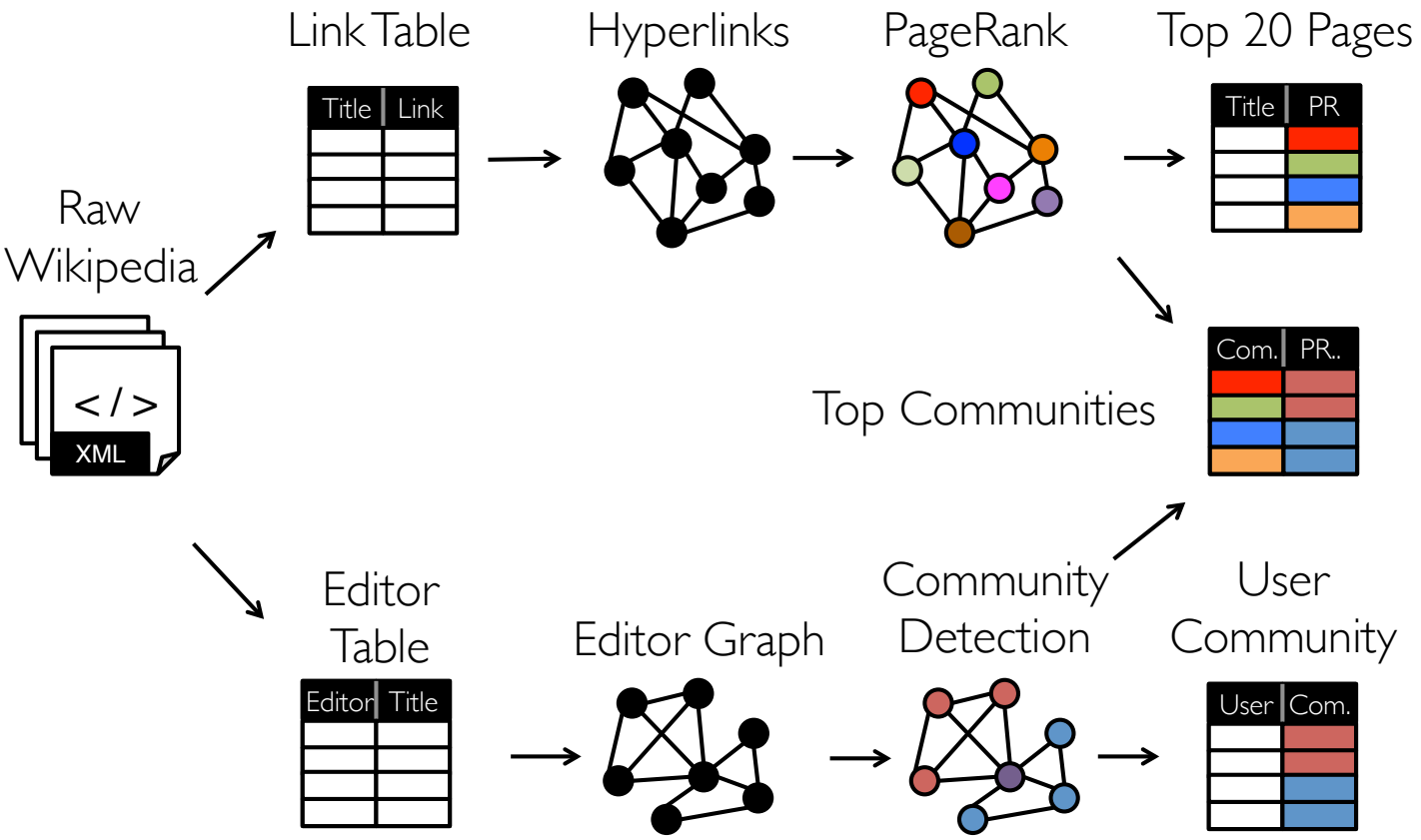
Semi-supervised ML
- » Graph SSL
- » CoEM

Community Detection
- » Triangle-Counting
- » K-core Decomposition
- » K-Truss

Graph Analytics
- » PageRank
- » Personalized PageRank
- » Shortest Path
- » Graph Coloring

Classification
- » Neural Networks

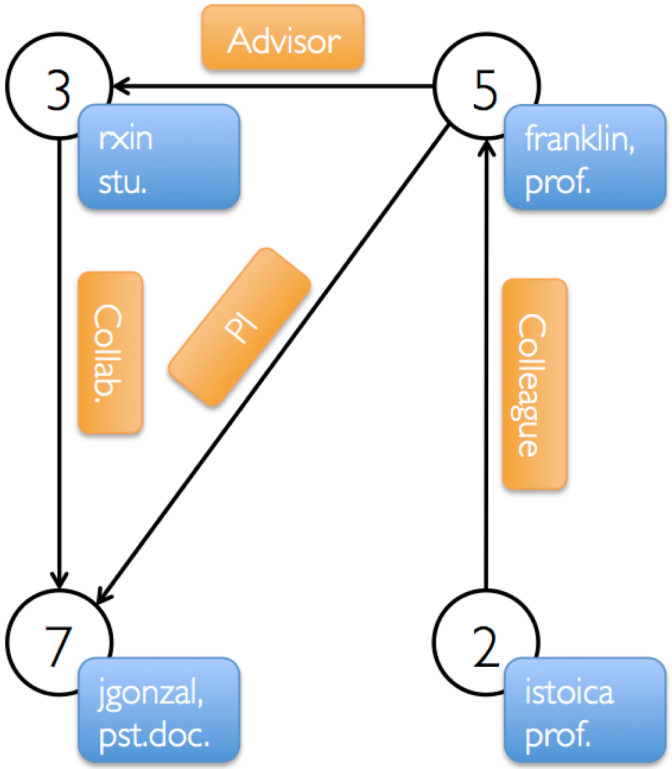
Modern Analytics



dbTrento

Graph ideas

Property Graph

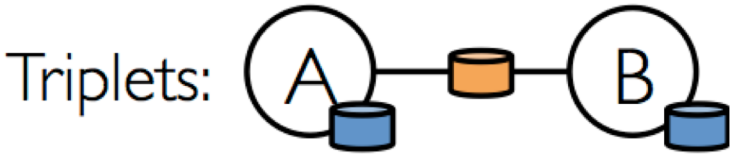


Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI



Contacts

For any problem, send a mail to:

daniele.foroni@unitn.it