Graph Analytics using Spark 2.0

ANA PAULA APPEL

RENAN FRANCISCO SANTOS SOUZA

Outline

Part 1: Spark

- Overview, RDD, DataFrames, Mllib

Part 2: Graph Analytics theory

- Motivations, algorithms, graph data science

Part 3: Graph Analytics in Spark 2.0

- GraphX, GraphFrames

What is Spark?

Open source fast cluster computing technology

Like other cluster computing frameworks (*e.g.*, Hadoop MapReduce), it provides a set of high level abstractions

- Facilitate use of cluster's resources
- Fault tolerance
- Task parallelism

The main innovation in Spark:

- **In-memory cluster computing** → increasing processing speed of a distributed application.

General-purpose, domain-independent

- It is designed to cover a wide range of workload: batch applications, iterative algorithms, interactive ad-hoc queries, and streaming.

It reduces the management burden of maintaining separate tools for distributed data processing, analysis, and visualization.

Features of Apache Spark

Speed

- Faster and easier development than other MR frameworks
- Faster processing speed – It reduces disk IO and makes extensive use of in-memory data processing

Supports multiple languages and libraries

- Java, Scala, Python, R, SQL, Mlib, GraphX, [put your library here]
- Over 80 high-level operators for interactive analysis and batch dataflow processing
  - Map, reduce, filter, join, union, difference, sort, etc

Advanced Analytics

- Interactive SQL queries
- Streaming data
- Machine learning
- Graph algorithms

Evolution of Apache Spark

Spark was born as a research project at UC Berkeley's AMPLab in 2009.

Open sourced in 2010

Apache project in 2013

Apache Spark became a top level Apache project in 2014.

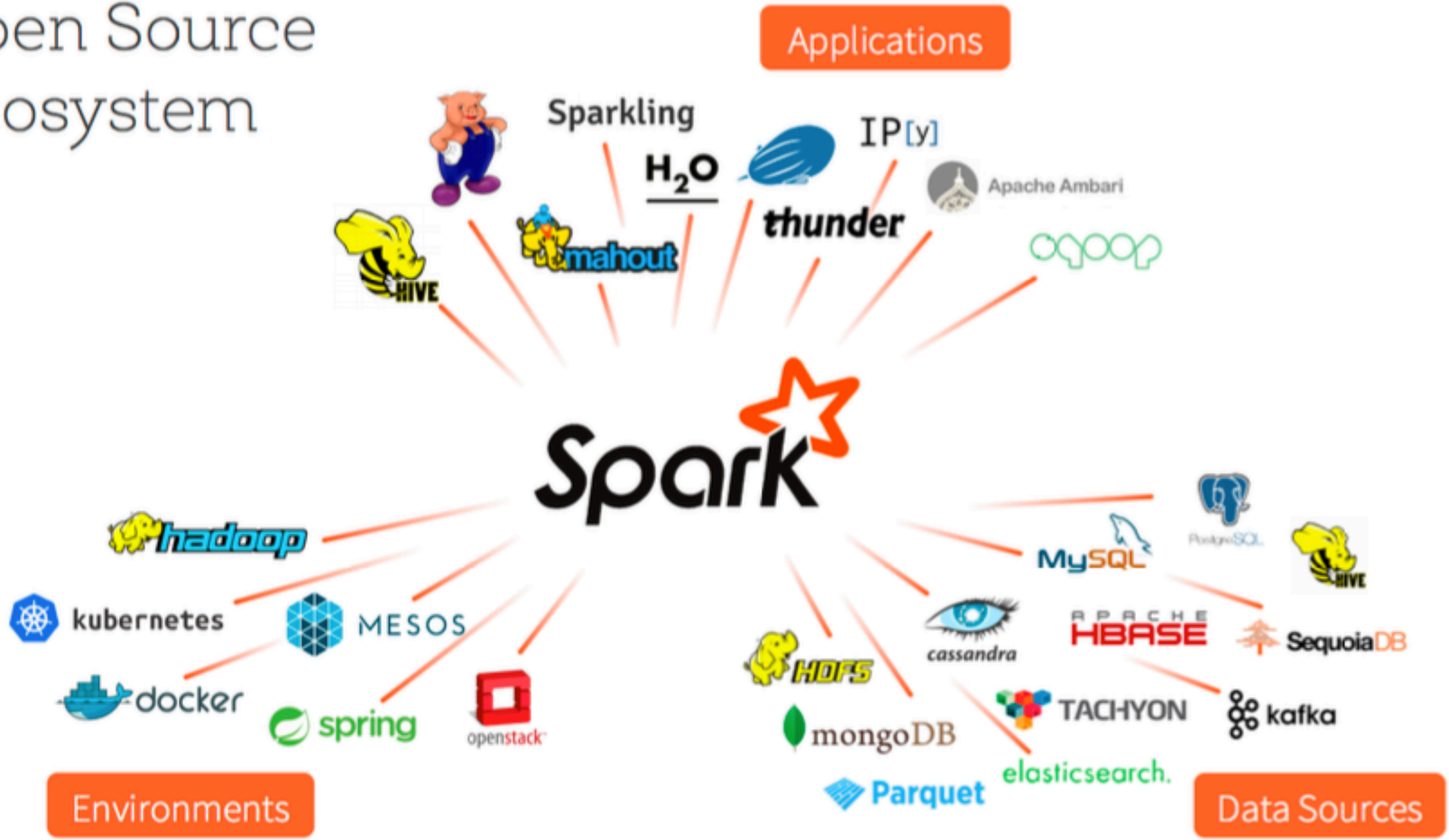
Other companies (*e.g.*, IBM) use Spark on top a big data cloud platform and provide data processing and analysis as a service.

Today it is the most active open source big data processing project

Apache Spark @ GitHub

- Over 17k commits
- 41 releases
- Almost 1k contributors

Open Source Ecosystem



Spark vs. Hadoop

Spark is not a modified version of Hadoop

It is not dependent on Hadoop

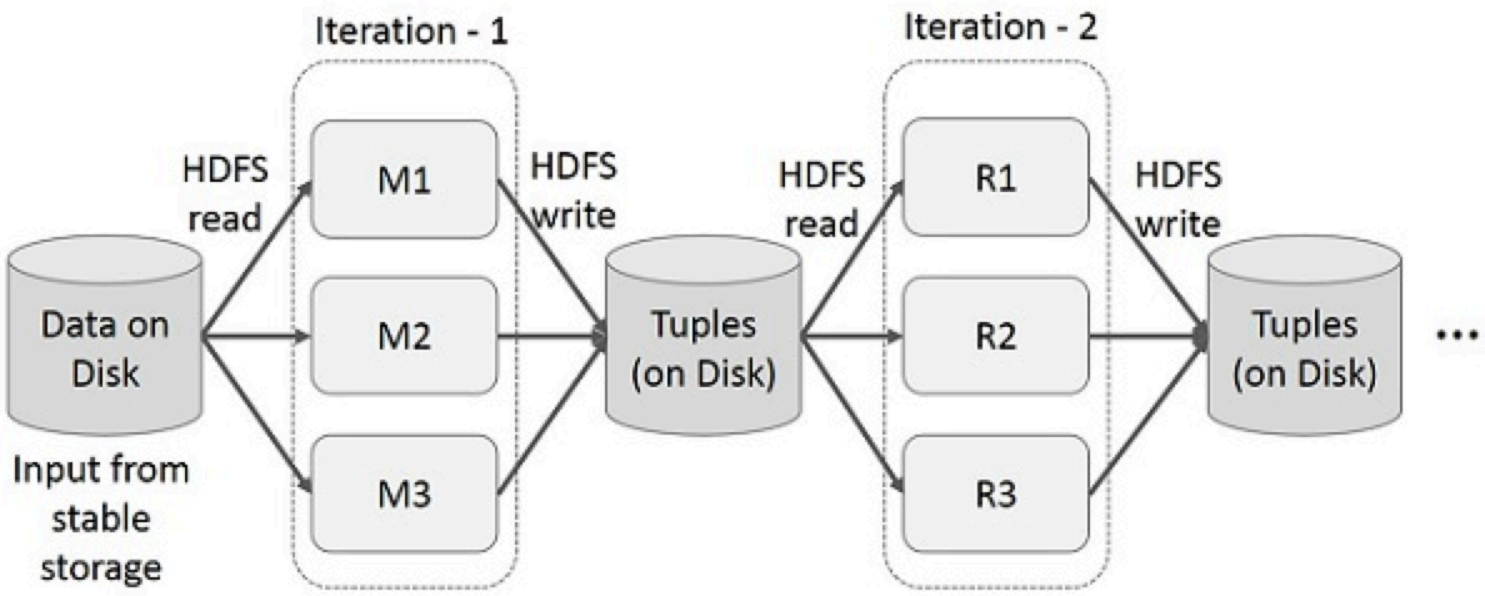
It needs to read data from somewhere

- File systems
  - Local file system (local mode deployment)
  - Shared (*e.g.*, NFS or data storage on a SAN)
  - Distributed (e.g., Hadoop Distributed File System – **HDFS – the most common choice in Spark clusters**)
- Data management systems
  - E.g., Hive, HBase, MySQL, Postgres
- Web services that provide data
  - E.g., Twitter streaming API

Iterative operations in Hadoop MR

To reuse data between MapReduce jobs, Hadoop needs to write to HDFS

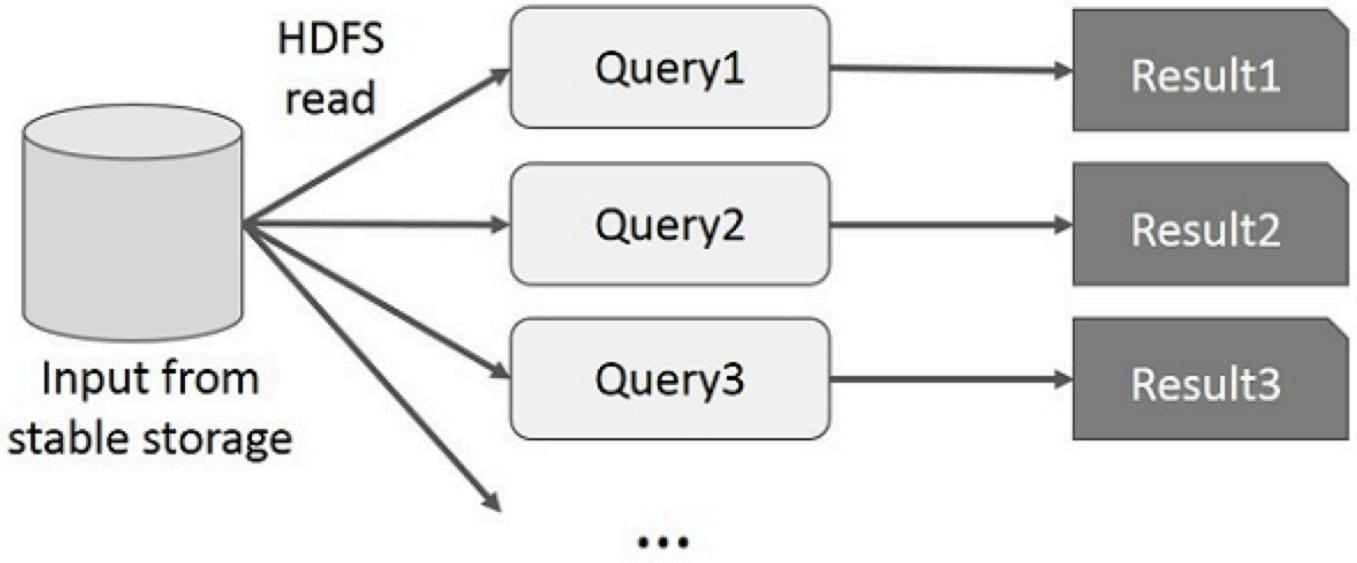
This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



Interactive operations in Hadoop MR

User runs ad-hoc queries

Each query in a dataset does HDFS I/O, which can dominate application execution time



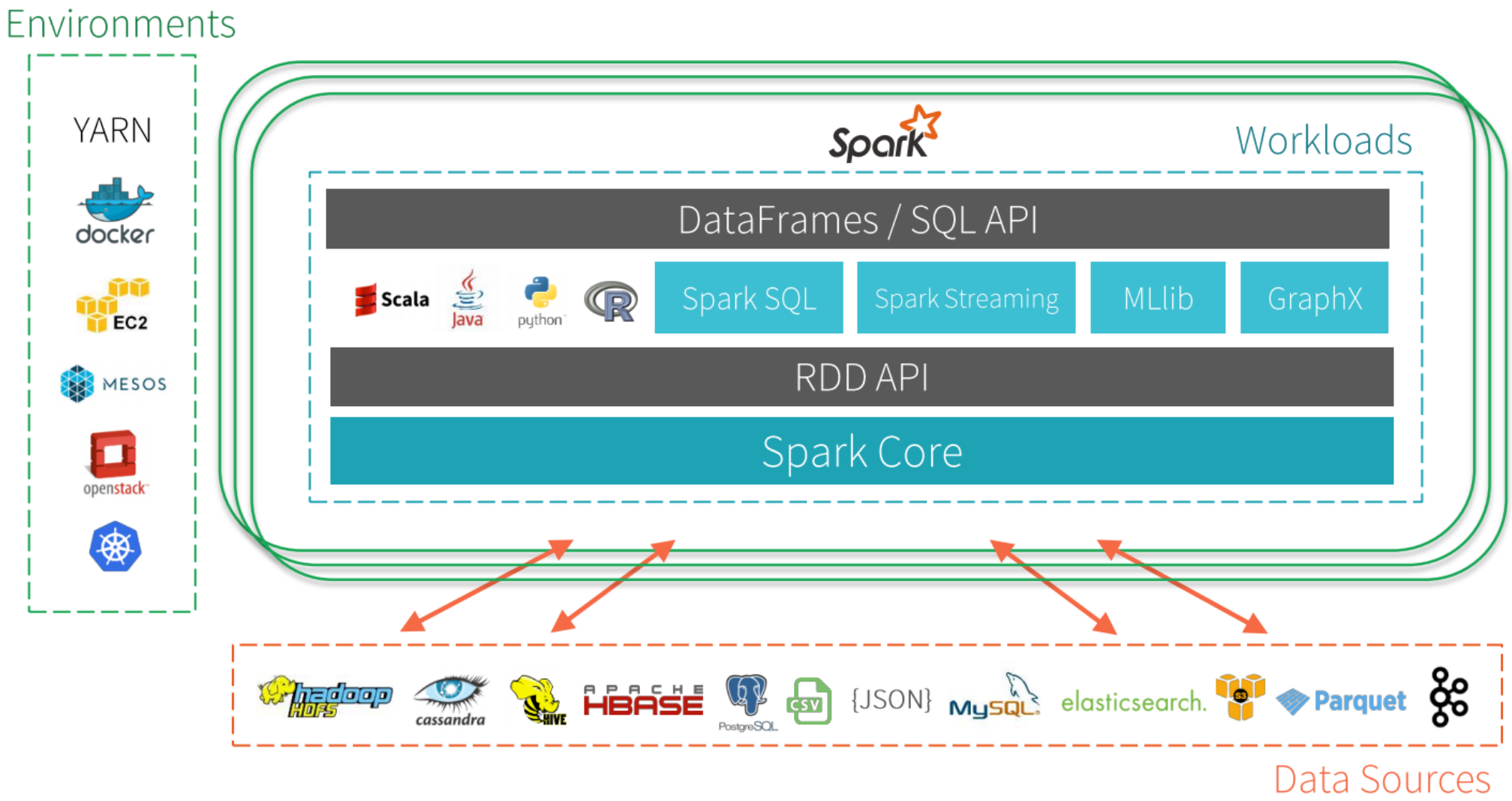
Spark vs. Hadoop MR

"Spark runs programs up to 100x faster than Hadoop MR in memory" ?

	Which is faster?	Main reason
Word count (200GB)	Spark 2.7x	Spark's map phase optimizations
k-means (70GB)	Spark 5x	- Iterative algorithms require intermediary data to be reused at each iteration. - Hadoop MR stores intermediary data on HDFS - Caching as RDDs reduces CPU and IO overheads - Less intermediary data materialization on HDFS
Linear regression (370GB)		
PageRank (42M *v*, 1.5B *e*, 24GB)		
Sort (500GB)	Hadoop MR 2x	MR has less network overheads in reduce phase

Shui et al., *Clash of the titans: MapReduce vs. Spark for large scale data analytics.* PVLDB, 2015

Spark Architecture



Resilient Distributed Dataset (RDD)

The main Spark component

An abstraction on top of distributed memory

Efficient data reuse and fault tolerance

RDD is a distributed dataset composed of data tuples

Any dataset operated in Spark is an RDD

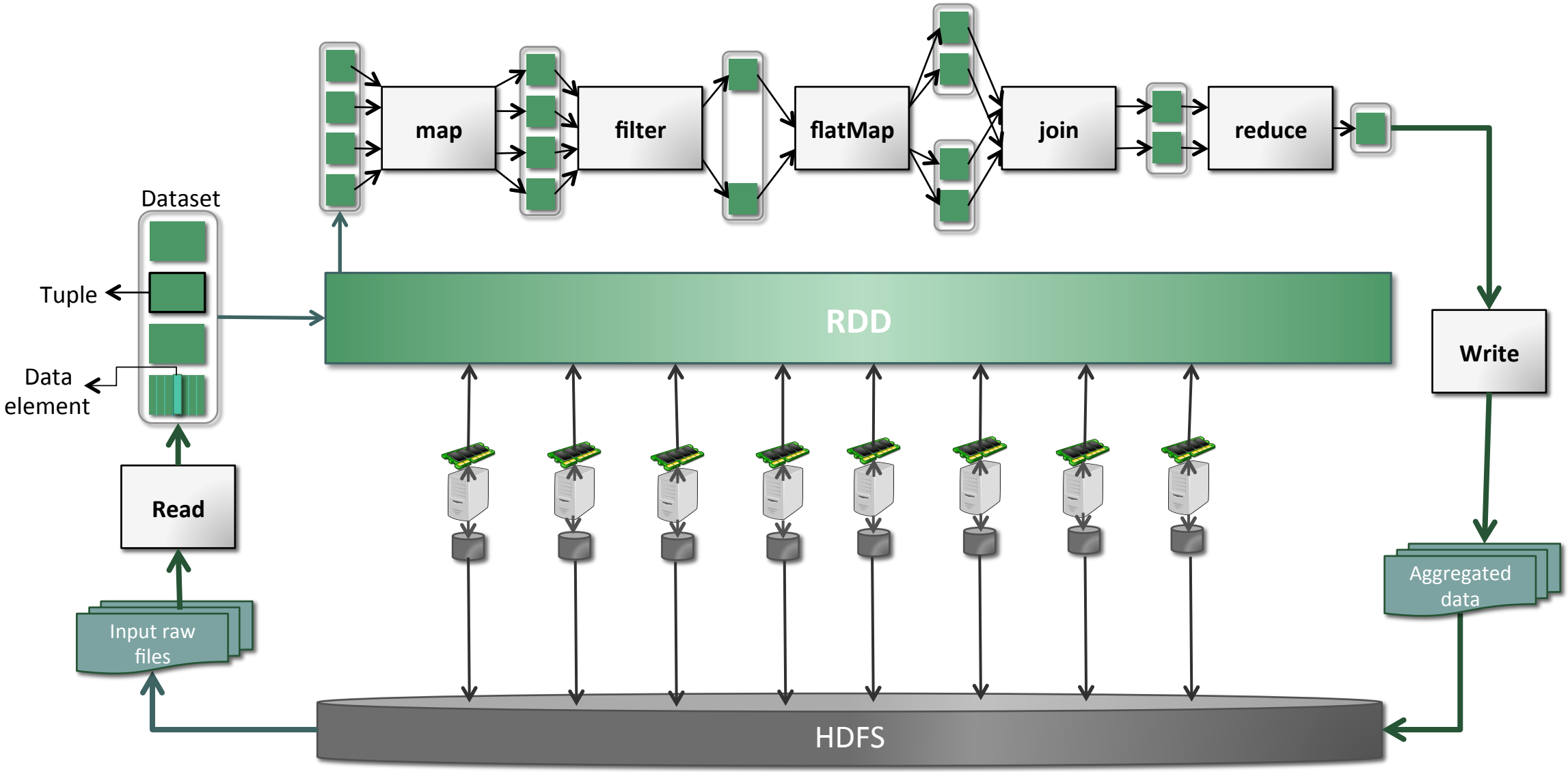
Resilient Distributed Dataset (RDD)

Distributed & Parallel

- It efficiently distributes a same data operation on all tuples of a dataset in parallel

Dataflows of linked data operations

- Each data operations process an input dataset and generates intermediary and output datasets


map
filter
flatMap
join
reduce
Dataset
Tuple
Data element
RDD
Write
Read
Aggregated data
Input raw files
HDFS

Data sharing using Spark RDD

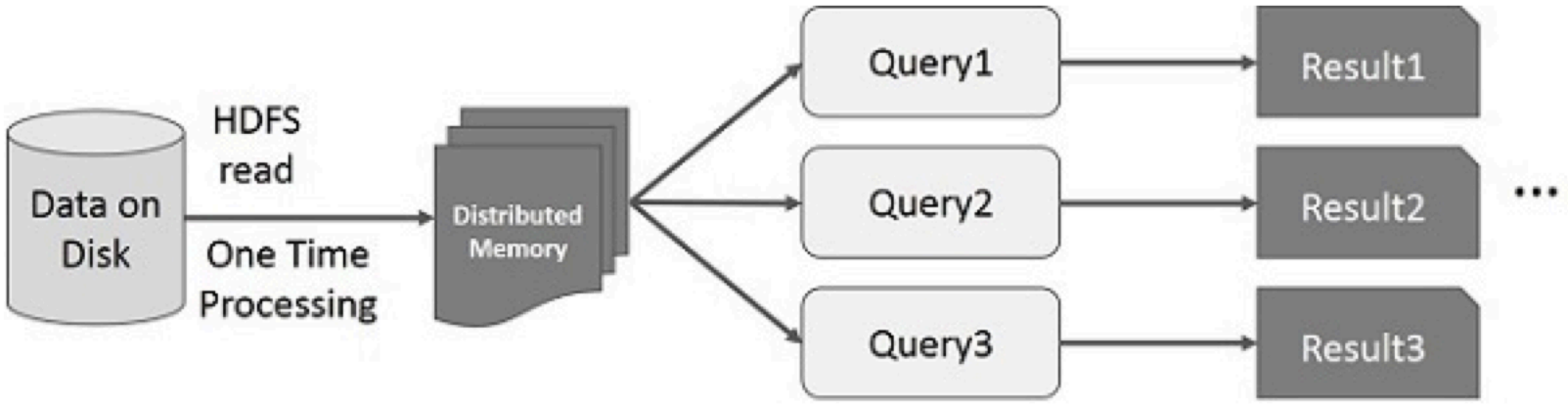
- RDD stores intermediary data between the jobs
- If the distributed memory is not sufficient, Spark will store on the disk.
- Data sharing in memory is 10 to 100 times faster than network and disk.

Interactive operations in Spark RDD

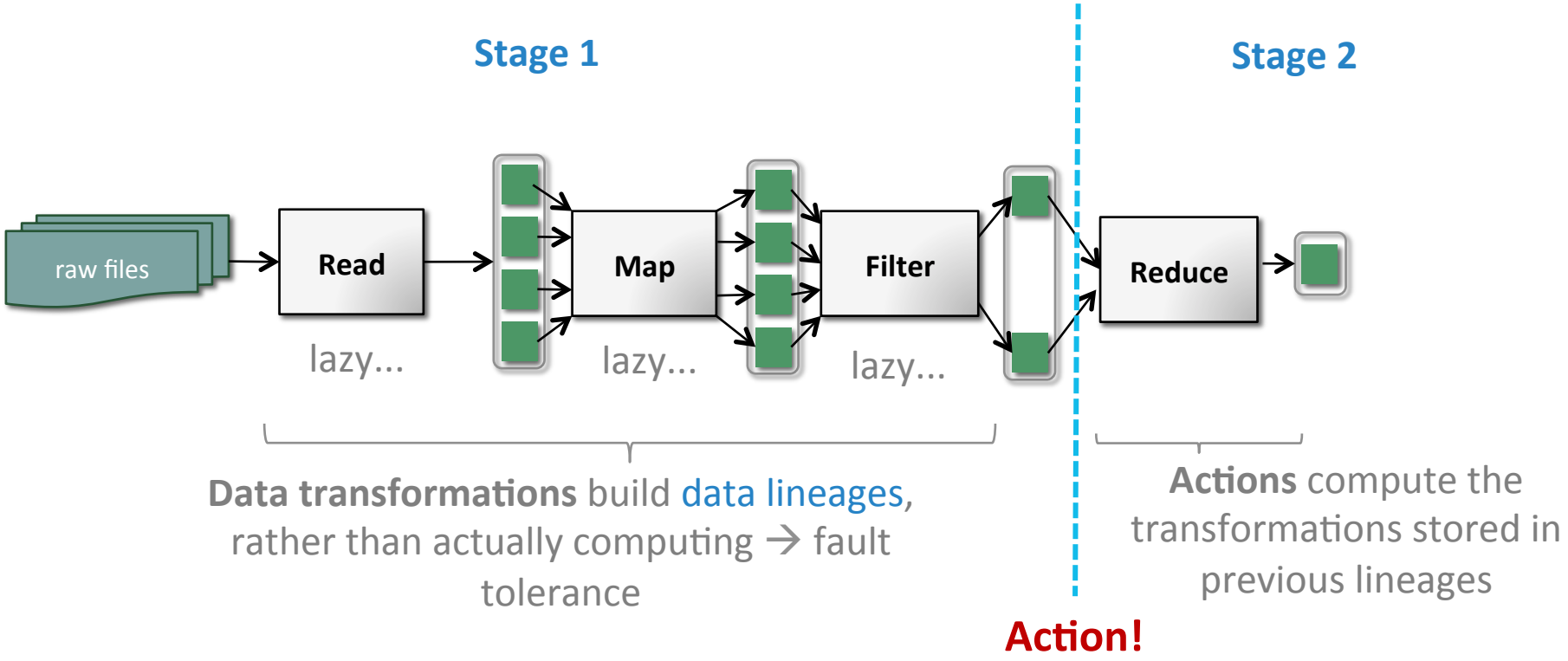
For different ad-hoc queries running on the dataset, data can be kept in memory

It is also possible to persist an RDD in memory. Spark will keep the data elements for faster access, for the next time you query it.

It is also easy to persist RDDs on disk



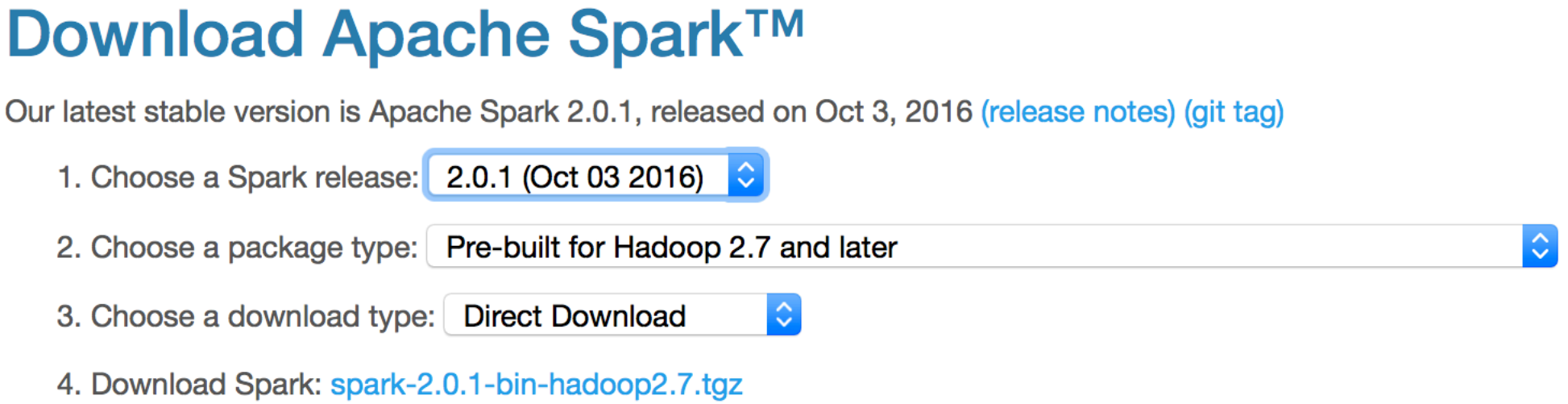
Lazy evaluation: data transformations vs. actions



Programming in Spark

Download & run

http://spark.apache.org/downloads.html



Untar and run in an interactive shell

- Scala: $SPARK_HOME/bin/spark-shell
- Python: $SPARK_HOME/bin/pyspark

Spark APIs

Programming languages

- Scala
- Java
- Python

Built-in libraries

- MLlib
- SparkR
- SparkSQL
- GraphX

RDD interface

- Your library here

MOST USED SPARK COMPONENTS

69% | Spark SQL

62% | DataFrames

58% | MLlib + GraphX

48% | Streaming

75%

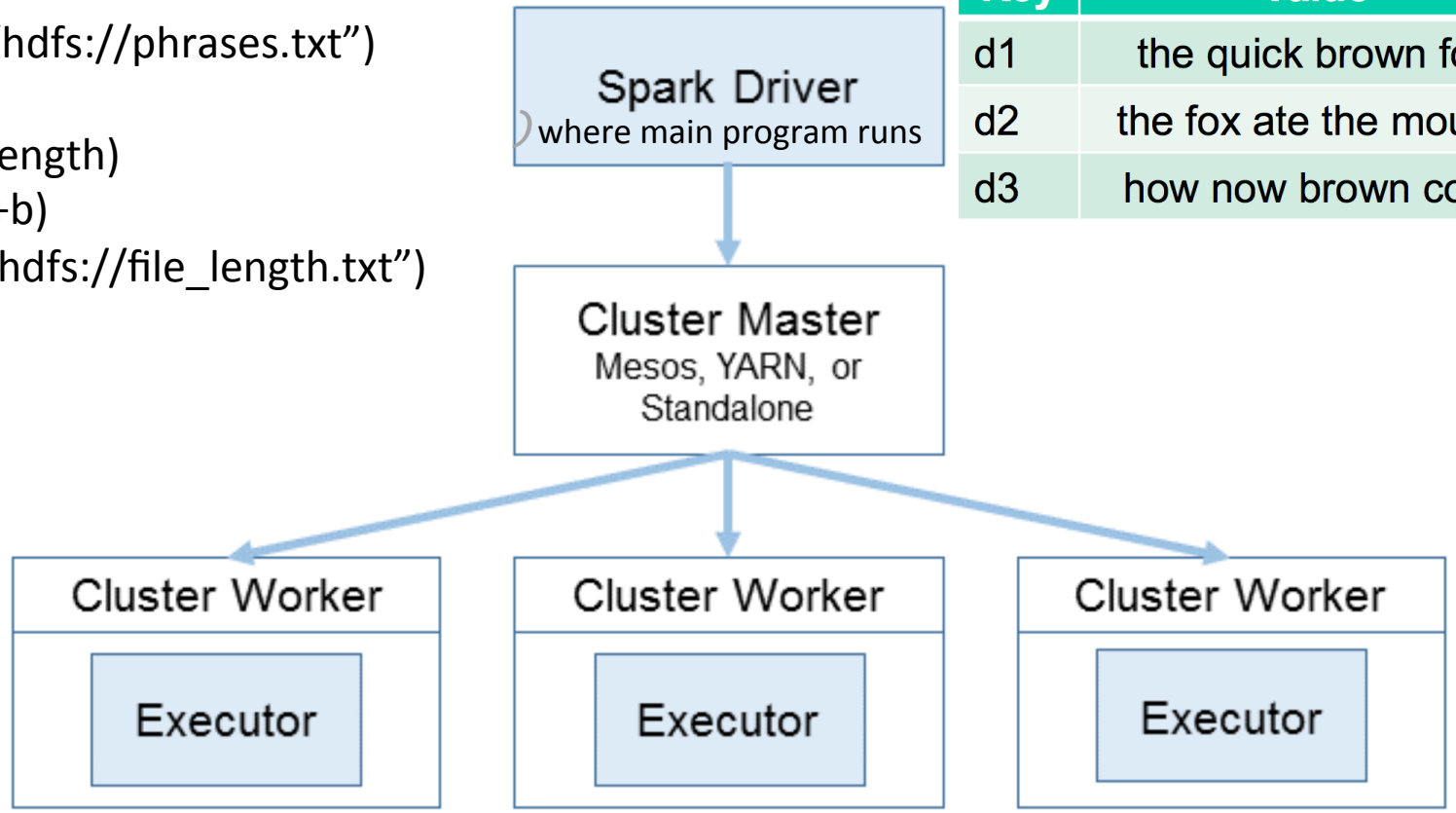
of Spark users are using two or more Spark components.

51% of Spark users are using three or more Spark components.

Spark Cluster

```
val myFirstRDD = sc.textFile("hdfs://phrases.txt")

myFirstRDD .map(line=>line.length)
           .reduce((a,b)=>a+b)
           .saveAsTextFile("hdfs://file_length.txt")
```

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow



Scala vs Python in pure Spark

SCALA

```
val rdd = sc.textFile(“hdfs://vectors.csv”)

rdd.map{ x=>
     val arr = x.split(",”);
     val sumSqr = arr.map(_.toFloat)
                          .map(x=>x*x)
                          .reduce(_+_);
     Math.sqrt(sumSqr)
 }.filter(_>25.0)
 .map(x=>x*0.42)
 .reduce(_+_)
```

PYTHON

```
def norm(str):
    arr_float = map(float, str.split(","))
    squares = [(x*x) for x in arr_float]
    sum_squares = sum(squares)
    return sqrt(sum_squares)

rdd = sc.textFile(“hdfs://vectors.csv”)

rdd.map(lambda x: norm(x))
      .filter(lambda x: x>25.0)
      .map(lambda x: x*0.42)
      .reduce(lambda x,y: x+y)
```

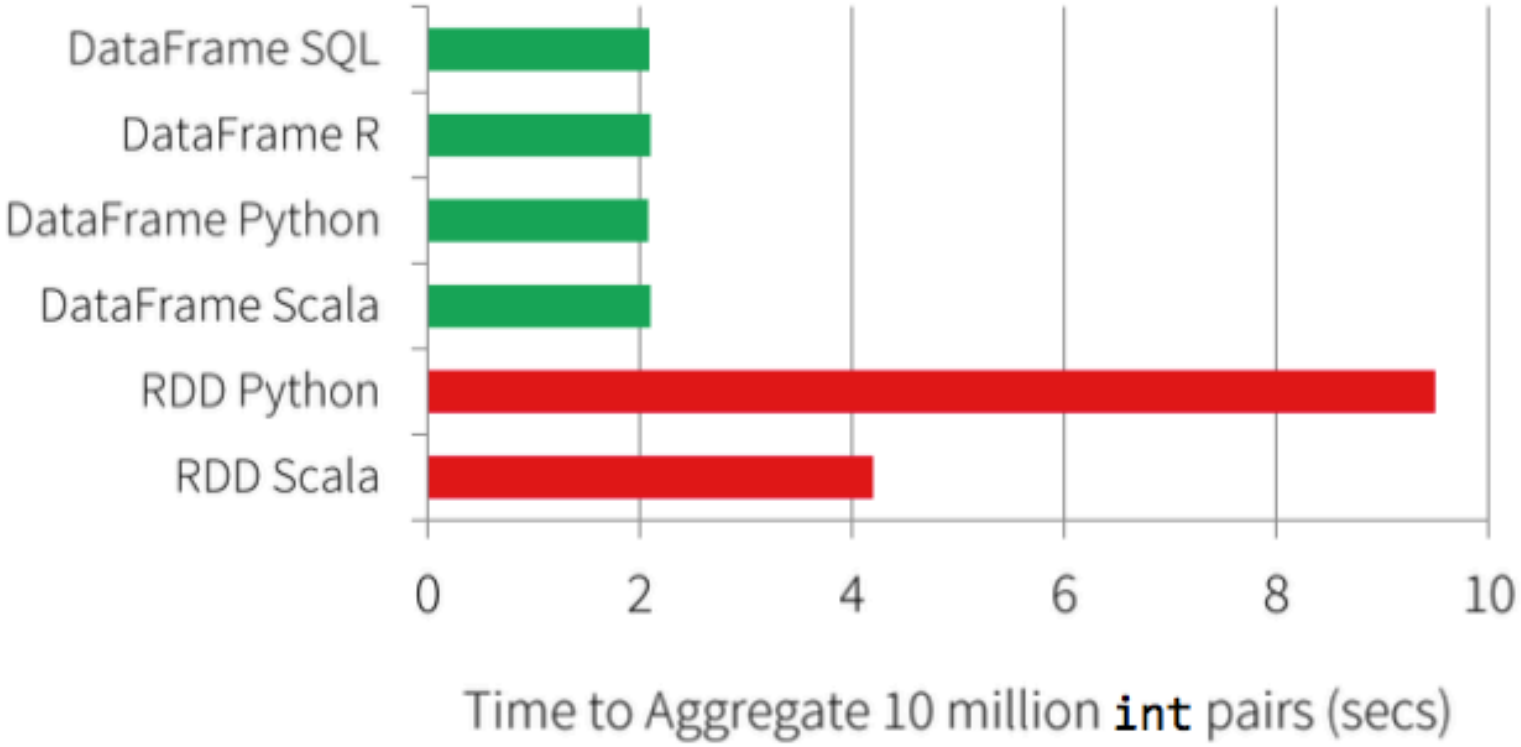
Datasets and DataFrames

- Higher level abstractions on top of the RDDs
- *New* standard ways to work with data in Spark 2.0
- Organized into named columns
- Equivalent to a table in RDBMS or a data frame in R/Python
- Rich optimizations for efficient data parallelism
- Available in Scala, Java, Python, and R

DataFrames

- **Can be constructed from a wide array of sources:**
  - Raw unstructured text, CSV, JSON files on disk
  - Compressed files
  - Parquet files
  - Structured data systems (Hive, MySQL, Postgres, etc)
  - Existing RDDs

Not Just Less Code: Faster implementation & performance



https://gist.github.com/rxin/c1592c133e4bccf515dd

RDDs

Collections of Native JVM Objects

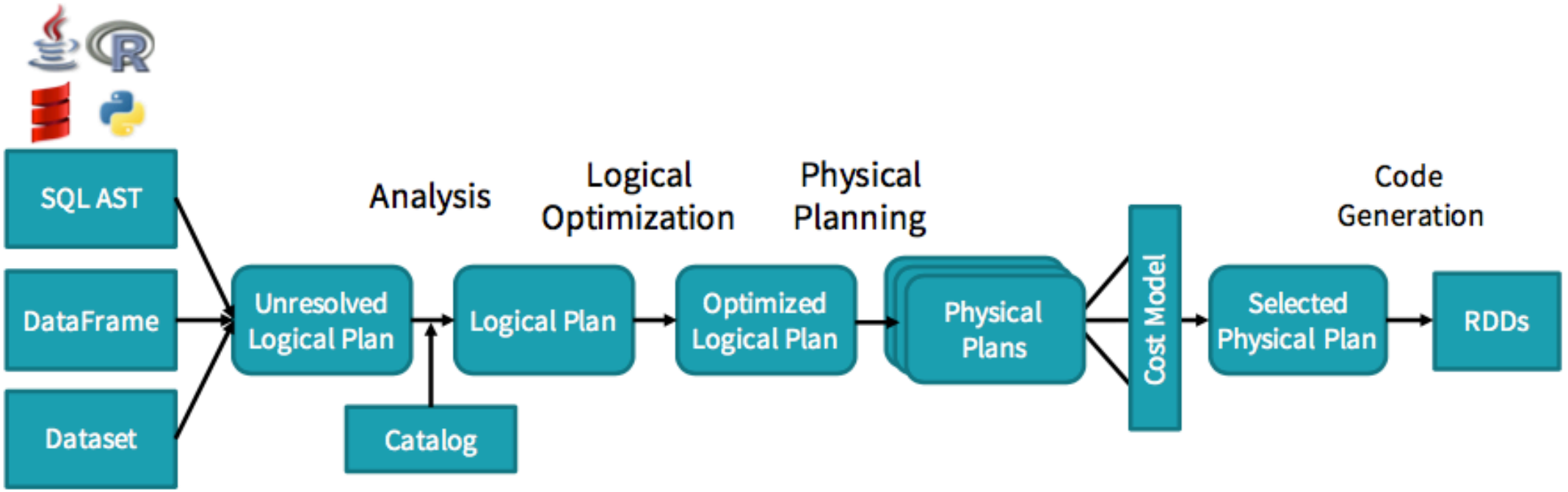
- Compile-time type-safety
- Easy to express certain types of logic
- Lots of existing code + users
- Lower level control of Spark

DataFrames / SQL

Structured Binary Data (Tungsten)

- Lower memory pressure (gc & space)
- Memory accounting (avoids OOMs)
- Faster sorting / hashing / serialization
- More opportunities for automatic optimization

Shared Optimization & Execution



DataFrames, Datasets and SQL
share the same optimization/execution pipeline

SparkSession: easier way to work with DF

SPARK 2.0

```
val spark = SparkSession.builder
                        .appName(”MyApp")
                        .master("local")
                        .getOrCreate()

val df = spark.read.json(“people.json”)

//df.select(“name”).show()

df.registerTempTable(“people”)

spark.sql(“select name from people”).show()
```

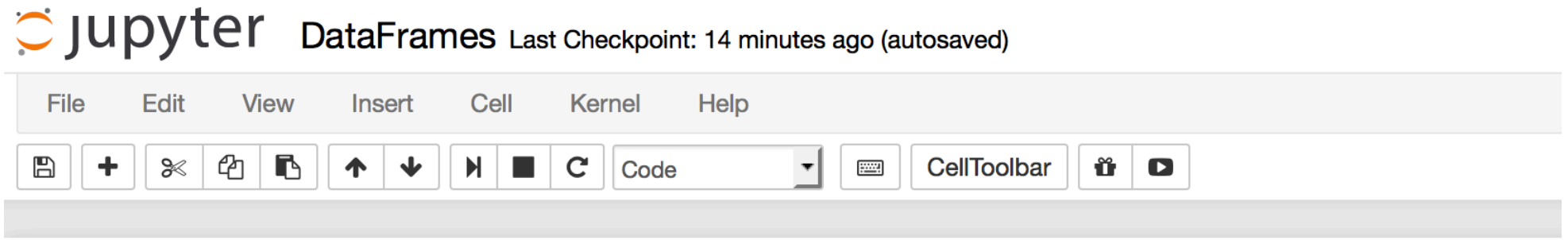
OLDER WAY

```
val conf = new SparkConf()
                        .setAppName(“MyApp”)
                        .setMaster(“local”)

val sc = new SparkContext(conf)

val sqlContext = new SQLContext(sc)

val df = sqlContext.read.json(“people.json”)

//df.select(“name”).show()

df.registerTempTable(“people”)

sqlContext.sql(“select name from
people”).show()
```

Interactive environment for data analysis with PySpark

http://jupyter.readthedocs.io/en/latest/install.html


```
export PYSPARK_DRIVER_PYTHON=$ANACONDA_HOME/bin/jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook --NotebookApp.open_browser=False --NotebookApp.ip='*' --NotebookApp.port=8888"
export XDG_RUNTIME_DIR=""
```



In [11]:
```
cat /home/bigdata/jupyter_notebooks/corpus.csv
```

```
id,x,y,eclass
1,-5,-2,0
2,-3,-1,0
3,-6,-0.3,0
4,-1.3,-45.1,0
5,-5.6,13.1,0
6,-9.48,1.4,0
7,-50.2,-55.21,0
8,-13.4,-4.1,0
9,-5.5,-5.41,0
10,-42.2,-1.40,0
11,-9,-1.52,0
12,-20.4,-3.1,0
13,-24.3,-5.51,0
14,5,42,1
15,3,21,1
16,6,0.3,1
17,1.3,45.1,1
18,5.6,13.1,1
```

jupyter DataFrames (unsaved changes)

File Edit View Insert Cell Kernel Help Python [Root]

Code CellToolbar

In [2]:
```
from pyspark.sql import SparkSession
import json
spark = SparkSession\
    .builder\
    .appName("PythonSQL")\
    .getOrCreate()
```

In [3]:
```
df = spark.read.option('header',True).csv('corpus.csv')
```

In [4]:
```
df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- x: string (nullable = true)
 |-- y: string (nullable = true)
 |-- eclass: string (nullable = true)
```

In [ ]:

```
In [1]: HDFS_URL = 'hdfs://brl-sda-skm.sl.cloud9.ibm.com:9000'
```

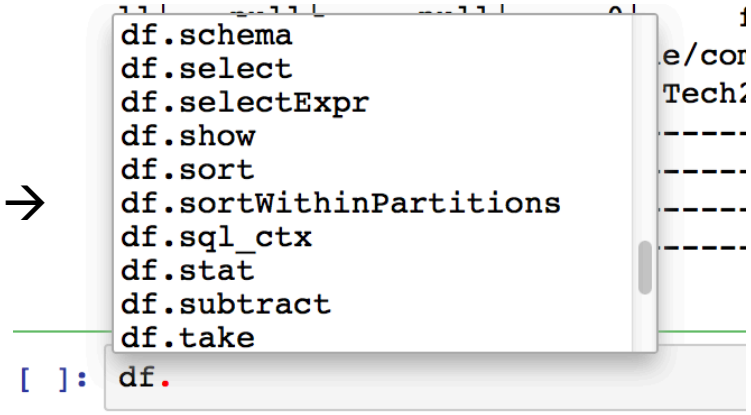
```
In [1]: r = sc.textFile(HDFS_URL+'/reddit/submissions/RS_2016-02.bz2')
```

```
In [2]: r.first()
```

```
Out[2]: u'{"from":null,"created_utc":1454284800,"saved":false,"num_comments":0,"p
ermalink":"/r/Gear4Sale/comments/43lprb/wts_tech21_sansamp_bass_driver_17
5_shipped/","from_id":null,"domain":"self.Gear4Sale","id":"43lprb","url":
"https://www.reddit.com/r/Gear4Sale/comments/43lprb/wts_tech21_sansamp_ba
ss_driver_175_shipped/","stickied":false,"over_18":false,"author_flair_te
xt":null,"edited":false,"quarantine":false,"author":"[deleted]","archived
":false,"secure_media_embed":{},"subreddit":"Gear4Sale","ups":5,"secure_m
edia":null,"score":5,"locked":false,"subreddit_id":"t5_2s4dp","selftext":
"[deleted]","downs":0,"link_flair_css_class":null,"media_embed":{},"hide_
score":false,"name":"t3_43lprb","title":"[WTS] Tech21 SansAmp Bass Driver
 - $175 Shipped","author_flair_css_class":null,"is_self":true,"thumbnail"
:"default","retrieved_on":1459148160,"from_kind":null,"media":null,"disti
nguished":null,"gilded":0,"link_flair_text":null}'
```

```
In [7]: df = spark.read.json('reddit.json')

In [8]: df.printSchema()

root
 |-- archived: boolean (nullable = true)
 |-- author: string (nullable = true)
 |-- author_flair_css_class: string (nullable = true)
 |-- author_flair_text: string (nullable = true)
 |-- created_utc: long (nullable = true)
 |-- distinguished: string (nullable = true)
 |-- domain: string (nullable = true)
 |-- downs: long (nullable = true)
 |-- edited: boolean (nullable = true)
 |-- from: string (nullable = true)
 |-- from_id: string (nullable = true)
 |-- from_kind: string (nullable = true)
 |-- gilded: long (nullable = true)
 |-- hide_score: boolean (nullable = true)
 |-- id: string (nullable = true)
 |-- is_self: boolean (nullable = true)
 |-- link_flair_css_class: string (nullable = true)
 |-- link_flair_text: string (nullable = true)
 |-- locked: boolean (nullable = true)
 |-- media: string (nullable = true)
 |-- name: string (nullable = true)
 |-- num_comments: long (nullable = true)
 |-- over_18: boolean (nullable = true)
 |-- permalink: string (nullable = true)
 |-- quarantine: boolean (nullable = true)
 |-- retrieved_on: long (nullable = true)
```

Autocomplete →




```
In [10]: df.limit(1).show()
```

```
+--------+---------+--------------------+----------------+-----------+------------+--------------+-----+------+--
--+-------+---------+------+----------+------+-------+--------------------+--------------+------+-----+---------+---
--------+-------+--------------------+---------+-----------+-----+-----+-----------+---------+--------+---------+
-----------+---------+--------------------+---+--------------------+
|archived|   author|author_flair_css_class|author_flair_text|created_utc|distinguished|      domain|downs|edited|fr
om|from_id|from_kind|gilded|hide_score|    id|is_self|link_flair_css_class|link_flair_text|locked|media|     name|num
_comments|over_18|           permalink|quarantine|retrieved_on|saved|score|secure_media| selftext|stickied|subreddit|
subreddit_id|thumbnail|               title|ups|                 url|
+--------+---------+--------------------+----------------+-----------+------------+--------------+-----+------+--
--+-------+---------+------+----------+------+-------+--------------------+--------------+------+-----+---------+---
--------+-------+--------------------+---------+-----------+-----+-----+-----------+---------+--------+---------+
-----------+---------+--------------------+---+--------------------+
|   false|[deleted]|                null|            null| 1454284800|         null|self.Gear4Sale|    0| false|nu
ll|   null|     null|     0|     false|43lprb|   true|                null|          null| false| null|t3_43lprb|
        0|  false|/r/Gear4Sale/comm...|     false|  1459148160|false|    5|        null|[deleted]|   false|Gear4Sale|
    t5_2s4dp|  default|[WTS] Tech21 Sans...|  5|https://www.reddi...|
+--------+---------+--------------------+----------------+-----------+------------+--------------+-----+------+--
--+-------+---------+------+----------+------+-------+--------------------+--------------+------+-----+---------+---
--------+-------+--------------------+---------+-----------+-----+-----+-----------+---------+--------+---------+
```

Run SQL on your data without loading into an RDBMS

```
In [11]: df = spark.read.json('reddit.json')

In [12]: df.registerTempTable("submission")

In [14]: rs = spark.sql("select * from submission")

In [15]: rs
Out[15]: DataFrame[archived: boolean, author: string, author_flair_css_class: string, author_flair_text: string, created_utc:
         bigint, distinguished: string, domain: string, downs: bigint, edited: boolean, from: string, from_id: string, from_ki
         nd: string, gilded: bigint, hide_score: boolean, id: string, is_self: boolean, link_flair_css_class: string, link_fla
         ir_text: string, locked: boolean, media: string, name: string, num_comments: bigint, over_18: boolean, permalink: str
         ing, quarantine: boolean, retrieved_on: bigint, saved: boolean, score: bigint, secure_media: string, selftext: string
         , stickied: boolean, subreddit: string, subreddit_id: string, thumbnail: string, title: string, ups: bigint, url: str
         ing]

In [23]: spark.sql("select score, url from submission").show()
+-----+--------------------+
|score|                 url|
+-----+--------------------+
|    5|https://www.reddi...|
+-----+--------------------+
```

In [2]:
```
from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName("PythonSQL")\
    .getOrCreate()
HDFS_URL = 'hdfs://cluster.br.ibm.com:9000'
```

In [3]:
```
comments = spark.read.json(HDFS_URL+'/home/bigdata/reddit/comments/*')
comments.registerTempTable("comment")
```

Large (hundreds of GB) compressed files
1 JSON per line

In [4]:
```
submissions = spark.read.json(HDFS_URL+'/home/bigdata/reddit/submissions/RC*')
submissions.registerTempTable("submission")
```

In [7]:
```
rs = sparl.sql("""
SELECT
    s.id as submission_id,
    substring(c.parent_id, 4, 100) as clean_parent_id,
    s.author as sub_author,
    c.author as com_author
FROM
    submission s,
    comment c
WHERE
    s.author != '[deleted]' and
    c.author != '[deleted]' and
    s.id = substring(c.parent_id, 4, 100)
""")
```

Graph person replies to person:
Author of a comment replies to author of a submission

In [8]:
```
rs.write.save(HDFS_URL+'/home/bigdata/reddit/parquet/joined.parquet', format='parquet')
```

Spark MLLib

Spark MLib

- linear SVM and logistic regression
- classification and regression tree
- multinomial naive Bayes
- random forest and gradient-boosted trees
- linear regression with L1- and L2-regularization
- isotonic regression
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power
- iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

KMEANS IN MLLIB

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Evaluate clustering by computing Within Set Sum of Squared Errors
Assign point to cluster centre

Pipelines

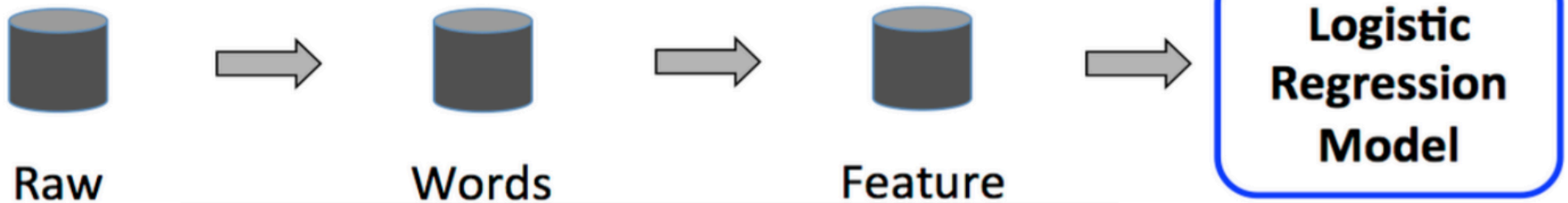
In machine learning, it is common to run a sequence of algorithms to process and learn from data.

- E.g., a simple text document processing workflow might include several stages:
  - Split each document's text into words.
  - Convert each document's words into a numerical feature
  - vector.
  - Normalization
  - Learn a prediction model using the feature vectors and labels.
  - Spark ML represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this

simple workflow as a running example in this section.

Pipeline (Estimator)

Pipeline.fit()



Raw text

Words

Feature

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

Pipeline-Prediction

```
# Prepare test documents, which are unlabeled.
Document = Row("id", "text")
test = sc.parallelize([(4, "spark i j k"),
                       (5, "l m n"),
                       (6, "spark hadoop spark"),
                       (7, "apache hadoop")]) \
    .map(lambda x: Document(*x)).toDF()

# Make predictions on test documents and print columns of interest.
prediction = cvModel.transform(test)
selected = prediction.select("id", "text", "prediction")
for row in selected.collect():
    print(row)

sc.stop()
```

```
Row(id=4, text=u'spark i j k', prediction=1.0)
Row(id=5, text=u'l m n', prediction=0.0)
Row(id=6, text=u'spark hadoop spark', prediction=1.0)
Row(id=7, text=u'apache hadoop', prediction=0.0)
```

Docs

http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/functions.html

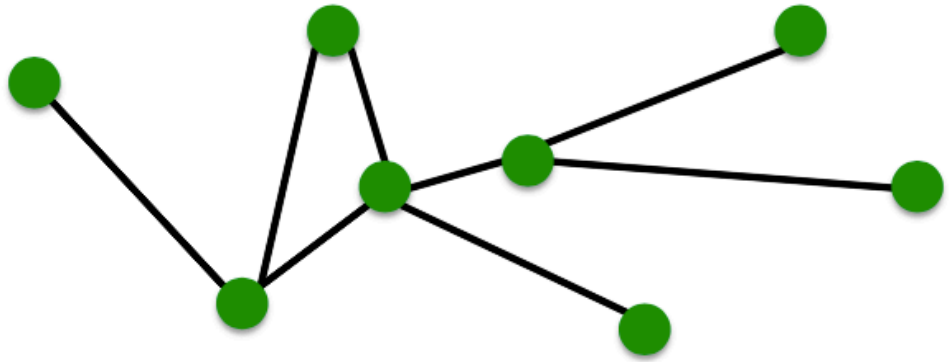
http://spark.apache.org/docs/latest/programming-guide.html

http://spark.apache.org/docs/latest/sql-programming-guide.html

http://spark.apache.org/docs/latest/ml-guide.html

Graph Analytics

Components of a network



Objects: nodes, vertices — N

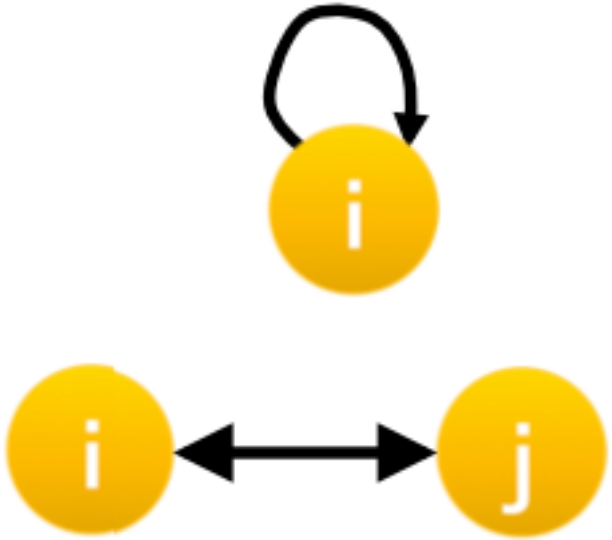
Interactions: links, edges — E

System: network, graph — $G(N,E)$

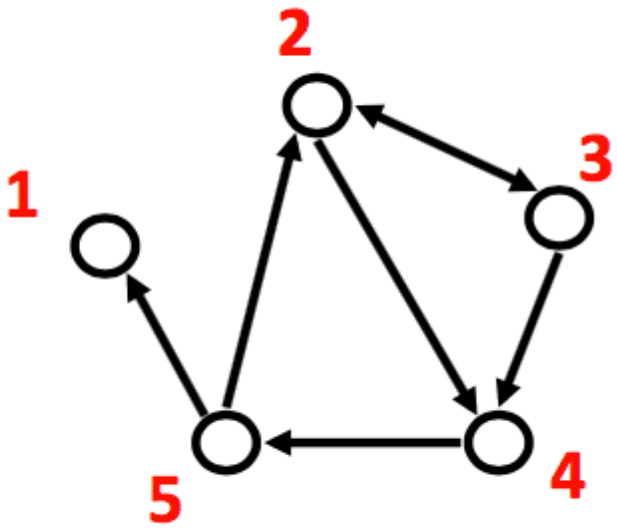
Adjacency Matrix

Representing edges (who is adjacent to whom) as a matrix

- Aij =1 if node *i* has an edge to node *j*
  = 0 if node *i* does not have an edge to j
- Aii = 0 unless the network has self-loops
- Aij = Aji if the network is undirected, or if *i* and *j* share a reciprocated edge



Adjacency Matrix

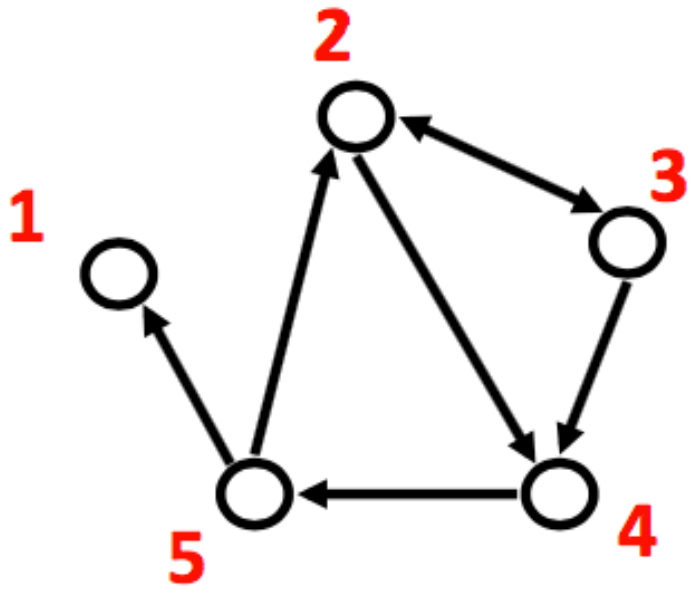


$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Edge List

Edge list

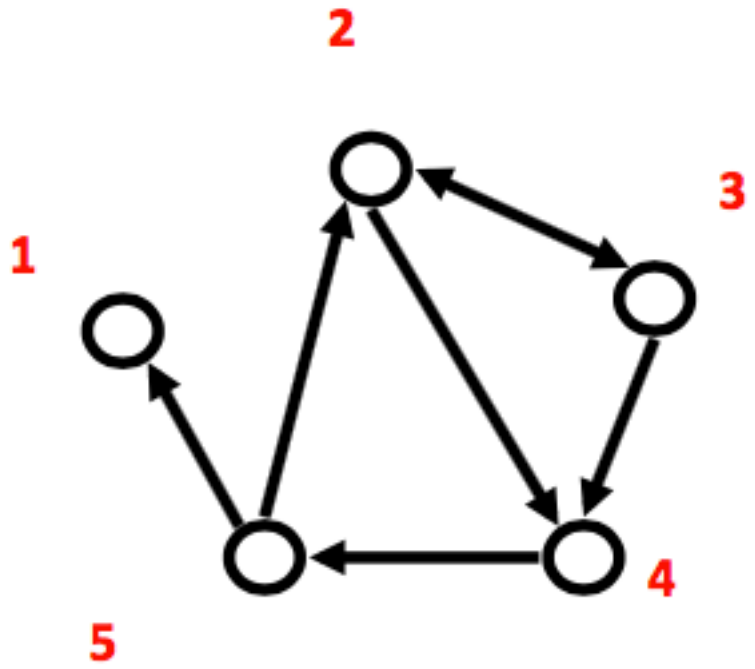
- 2, 3
- 2, 4
- 3, 2
- 3, 4
- 4,5
- 5, 2
- 5, 1



Adjacency List

Adjacency list

- is easier to work with if network is
  - Large
  - sparse
- quickly retrieve all neighbors for a node
  - 1:
  - 2: 3 4
  - 3: 2 4
  - 4: 5
  - 5: 1 2



Why Networks? Why Now?

Universal language for describing complex data

- Networks from science, nature, and technology are more similar than one would expect

Shared vocabulary between fields

- Computer Science, Social science, Physics, Economics, Statistics, Biology

Data availability (computational challenges)

- Web/mobile, bio, health, and medical

Impact!

- Social networking, Social media, Drug design

Networks: Size Matters

Network data: Orders of magnitude

436-node network of email exchange at a corporate research lab [Adamic-Adar, SocNets '03]

43,553-node network of email exchange at an university [Kossinets-Watts, Science '06]

4.4-million-node network of declared friendships on a blogging community [Liben-Nowell et al., PNAS '05]

240-million-node network of communication on Microsoft Messenger [Leskovec-Horvitz, WWW '08]

800-million-node Facebook network [Backstrom et al. '11]

Networks Really Matter

If you want to understand the spread of diseases, **you need to figure out who will be in contact with whom**

If you want to understand the structure of the Web, **you have to analyze the 'links'.**

If you want to understand dissemination of news or evolution of science, **you have to follow the flow.**

Reasoning about Networks

What do we hope to achieve from studying networks?

- Patterns and statistical **properties** of network data
- **Design principles** and **models**
- **Understand** why networks are organized the way they are
  - Predict behavior of networked systems

Reasoning about Networks

How do we reason about networks?

- **Empirical:** Study network data to find organizational principles
  - How do we measure and quantify networks?

Mathematical models: Graph theory and statistical models

- Models allow us to understand behaviors and distinguish surprising from expected phenomena

Algorithms for analyzing graphs

- Hard computational challenges

Networks: Structure & Process

What do we study in networks?

- **Structure and evolution:**
  - What is the structure of a network?
  - Why and how did it come to have such structure?
- **Processes and dynamics:**
  - Networks provide “skeleton” for spreading of information, behavior, diseases
  - How do information and diseases spread?

Networks: Online

Communication networks:

- Intrusion detection, fraud
- Churn prediction

Social networks:

- Link prediction, friend recommendation
- Social circle detection, community detection
- Social recommendations
- Identifying influential nodes, Information virality

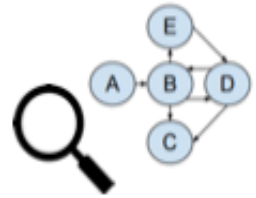
Information networks:

- Navigational aids

Graph Analytics in Spark 2.0

Graph Algorithms

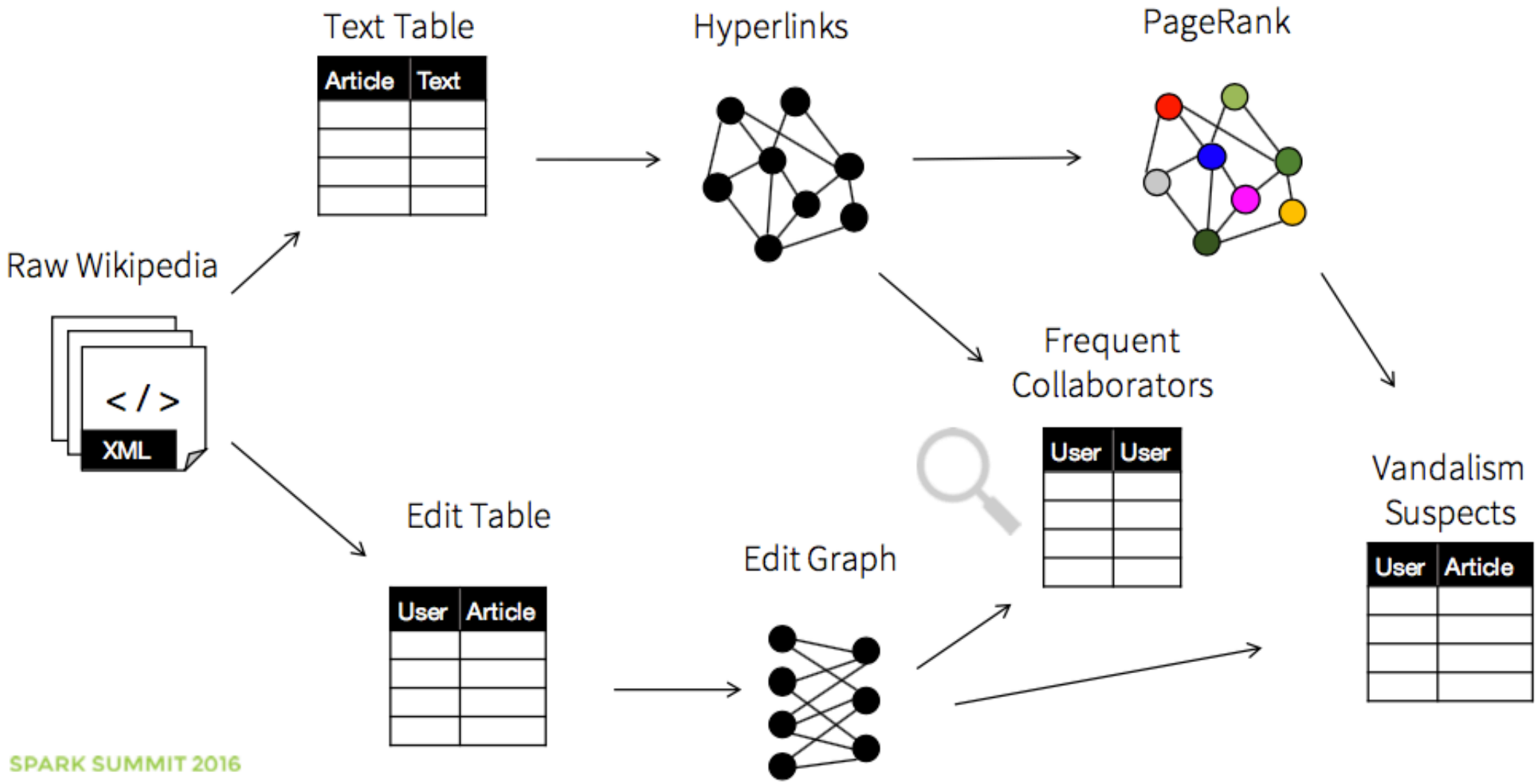
Graph Queries



GraphFrames API

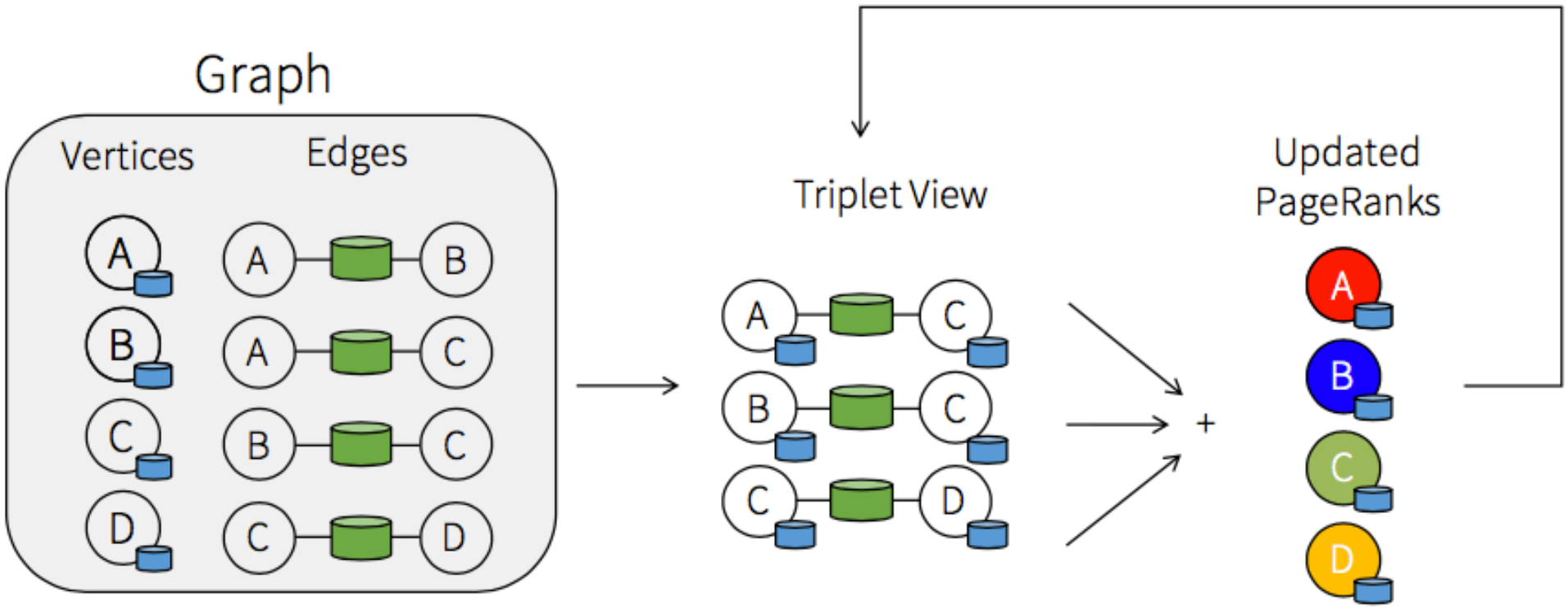
Pattern Query Optimizer

Spark SQL


Text Table
Article
Text
Hyperlinks
PageRank
Raw Wikipedia
< / >
XML
Frequent
Collaborators
User
User
Vandalism
Suspects
User
Article
Edit Table
User
Article
Edit Graph

GraphX

- Component of Spark for graph-parallel computation.
- It extends RDD by introducing a new graph abstraction
  - a directed multigraph with properties attached to each vertex and edge.
- GraphX exposes a variant of the Pregel API.
  - At a high level the Pregel operator in GraphX is a bulk-synchronous parallel messaging abstraction constrained to the topology of the graph.
- GraphX exposes a variant of the GraphLab API.



GraphX: Triplet view enabled efficient message-passing algorithms

Building a Graph on GraphX

```
val vertexArray = Array(
  (1L, ("Alice", 28)),
  (2L, ("Bob", 27)),
  (3L, ("Charlie", 65)),
  (4L, ("David", 42)),
  (5L, ("Ed", 55)),
  (6L, ("Fran", 50))
  )
val edgeArray = Array(
  Edge(2L, 1L, 7),
  Edge(2L, 4L, 2),
  Edge(3L, 2L, 4),
  Edge(3L, 6L, 3),
  Edge(4L, 1L, 1),
  Edge(5L, 2L, 2),
  Edge(5L, 3L, 8),
  Edge(5L, 6L, 3)
  )
val vertexRDD = sc.parallelize(vertexArray)
val edgeRDD = sc.parallelize(edgeArray)
val graph = Graph(vertexRDD, edgeRDD)
```

Building a Graph on GraphX

```
val articles = sc.textFile("articles.txt")
val links = sc.textFile("links.txt")
val vertices = articles.map { line =>
  val fields = line.split('\t')
  (fields(0).toLong, fields(1))
}
val edges = links.map { line =>
  val fields = line.split('\t')
  Edge(fields(0).toLong, fields(1).toLong, 0)
}
val graph = Graph(vertices, edges, "").cache()
```

Another Example

```
// run graph analytics
val g: Graph[String, Int] = Graph(nodes, edges)
val r = g.pageRank(0.0001).vertices
r.join(nodes).sortBy(_._2._1, ascending=false).foreach(println)

// define a reduce operation to compute the highest degree vertex
def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
  if (a._2 > b._2) a else b
}

// compute the max degrees
val maxInDegree: (VertexId, Int)  = g.inDegrees.reduce(max)
val maxOutDegree: (VertexId, Int) = g.outDegrees.reduce(max)
val maxDegrees: (VertexId, Int)   = g.degrees.reduce(max)

// connected components
val scc = g.stronglyConnectedComponents(10).vertices
node.join(scc).foreach(println)
```

Graph Algorithms

Find important vertices
- PageRank

Find paths between sets of vertices
- Breadth-first search (BFS)
- Shortest paths

Find groups of vertices (components, communities)
- Connected components
- Strongly connected components
- Label Propagation Algorithm (LPA)

Other
- Triangle counting
- SVDPlusPlus

GraphX Library

Overview

- General-purpose graph processing library
- Optimized for fast distributed computing
- Library of algorithms: PageRank, Connected Components, etc.

Challenges

- No Java, Python APIs
- Lower-level RDD-based API (vs. DataFrames)
- Cannot use recent Spark optimizations: Catalyst query optimizer, Tungsten memory management

GraphFrames

- Goal: DataFrame-based graphs on Apache Spark
- Simplify interactive queries
- Support motif-finding for structural pattern search
- Benefit from DataFrame optimizations
- Collaboration between Databricks, UC Berkeley & MIT + Now with community contributors!

What are GraphFrames?

- *GraphX is to RDDs as GraphFrames are to DataFrames.*
- GraphFrames represent graphs: vertices (e.g., users) and edges (e.g., relationships between users). If you are familiar with GraphX, then GraphFrames will be easy to learn. The key difference is that GraphFrames are based upon Spark DataFrames, rather than RDDs.
- GraphFrames also provide powerful tools for running queries and standard graph algorithms.

SparkPackages Feedback Register a package Login Find a package

graphframes (homepage)

GraphFrames: DataFrame-based Graphs

@graphframes / ★★★★★ (7)

This is a prototype package for DataFrame-based graphs in Spark. Users can write highly expressive queries by leveraging the DataFrame API, combined with a new API for motif finding. The user also benefits from DataFrame performance optimizations within the Spark SQL engine.

Tags

3 graph 2 DataFrame

How to [+]

Include this package in your Spark Applications using:

spark-shell, pyspark, or spark-submit

```
> $SPARK_HOME/bin/spark-shell --packages graphframes:graphframes:0.2.0-spark2.0-s_2.11
```

Releases

GraphFrames

"vertices" DataFrame

- 1 vertex per Row
- id: column with unique ID

id	City	State
"JFK"	"New York"	NY
"SEA"	"Seattle"	WA

"edges" DataFrame

- 1 edge per Row
- src, dst: columns using IDs from vertices.id

src	dst	delay	tripID
"JFK"	"SEA"	45	1058923
"DFW"	"SFO"	-7	4100224

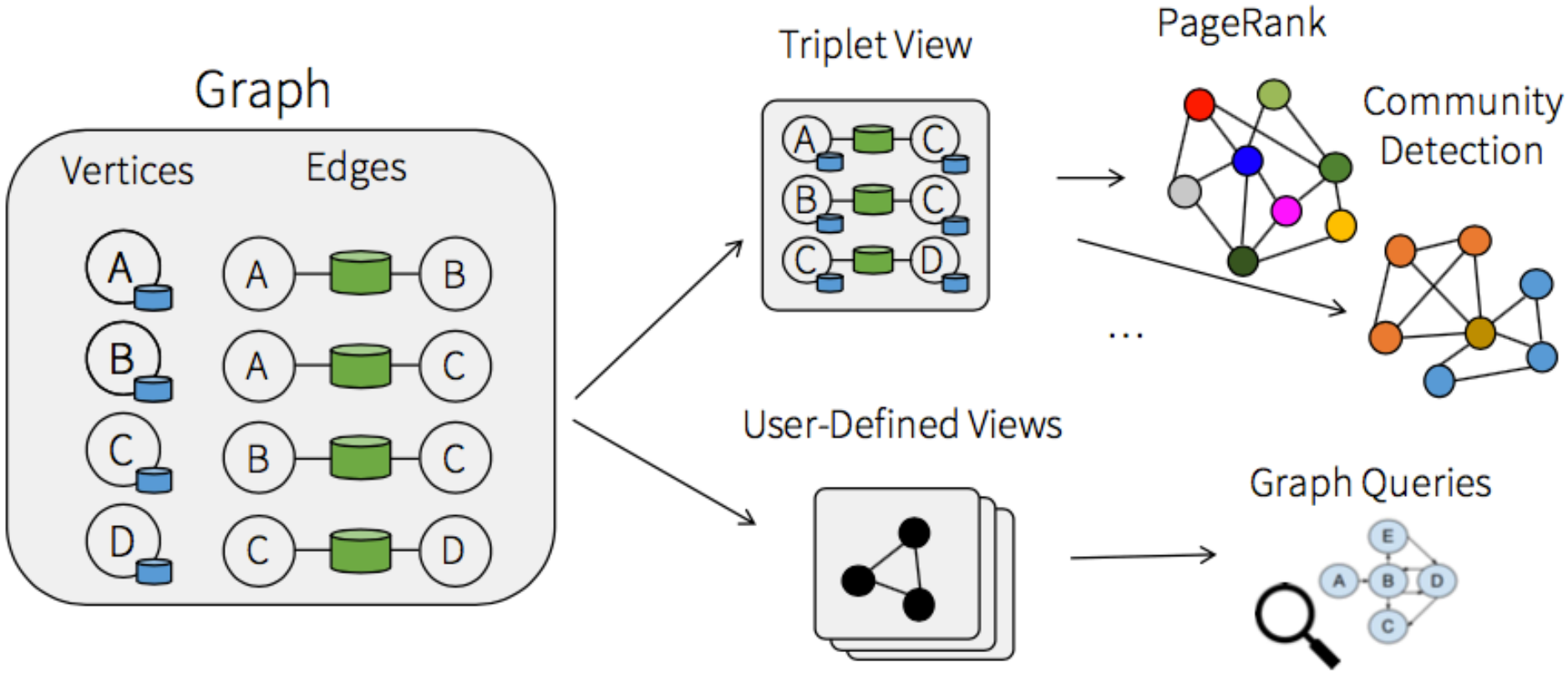
Extra columns store vertex or edge data (a.k.a. attributes or properties).

GraphFrames API

Unifies graph algorithms, graph queries, and DataFrames

Available in Scala, Java, and Python

```
class GraphFrame {
  def vertices: DataFrame
  def edges: DataFrame

  def find(pattern: String): DataFrame
  def registerView(pattern: String, df: DataFrame): Unit

  def degrees(): DataFrame
  def pageRank(): GraphFrame
  def connectedComponents(): GraphFrame
  ...
}
```



GraphFrames: User-defined views enable efficient graph queries

Algorithm Implementations

Mostly wrappers for GraphX

- PageRank
- Shortest paths
- Connected components
- Strongly connected components
- Label Propagation Algorithm (LPA)
- SVDPlusPlus

Some algorithms implemented using DataFrames

- Breadth-first search
- Triangle counting

APIs: Scala, Java, Python

- API available from all 3 languages
- First time GraphX functionality has been available to Java & Python users
- Problems
  - Java-friendliness is currently in alpha.
  - Python does not have aggregate Messages
- (for implementing your own graph algorithms).

Saving & loading graphs

```
from __future__ import print_function
from pyspark.sql import Row, SparkSession
from pyspark.sql.functions import *
from graphframes import *
import os

caminho = "/Users/anaappel/spark2/graphframes/IA.csv"

spark = SparkSession.builder.appName("TutorialGraphFrames").getOrCreate()

sep = ';'

df = spark.read.csv(caminho, None, sep, None, None, None, None, True)
nos = df.select(df['src'])
nos = nos.select(df['src'].cast('int')).withColumnRenamed('src', 'id').dropDuplicates()

arestas = df.select('*')


g = GraphFrame(nos,arestas)
```

GraphFrames vs. GraphX

	GraphFrames	GraphX
Built on	DataFrames	RDDs
Languages	Scala, Java, Python	Scala
Use cases	Queries & algorithms	Algorithms
Vertex IDs	Any type (in Catalyst)	Long
Vertex/edge attributes	Any number of DataFrame columns	Any type (VD, ED)
Return types	GraphFrame or DataFrame	Graph[VD, ED], or RDD[Long, VD]

GraphX compatibility

Simple conversions between GraphFrames & GraphX.

```
val g: GraphFrame = ...

// Convert GraphFrame → GraphX
val gx: Graph[Row, Row] = g.toGraphX

// Convert GraphX → GraphFrame
val g2: GraphFrame = GraphFrame.fromGraphX(gx)
```

Vertex & edge attributes are Rows in order to handle non-Long IDs

Wrapping existing GraphX code: See Belief Propagation example:
https://github.com/graphframes/graphframes/blob/master/src/main/scala/org/graphframes/examples/BeliefPropagation.scala

Data Representation

- Adjacency matrix
- Edgelist
- Adjacency list

Power Law

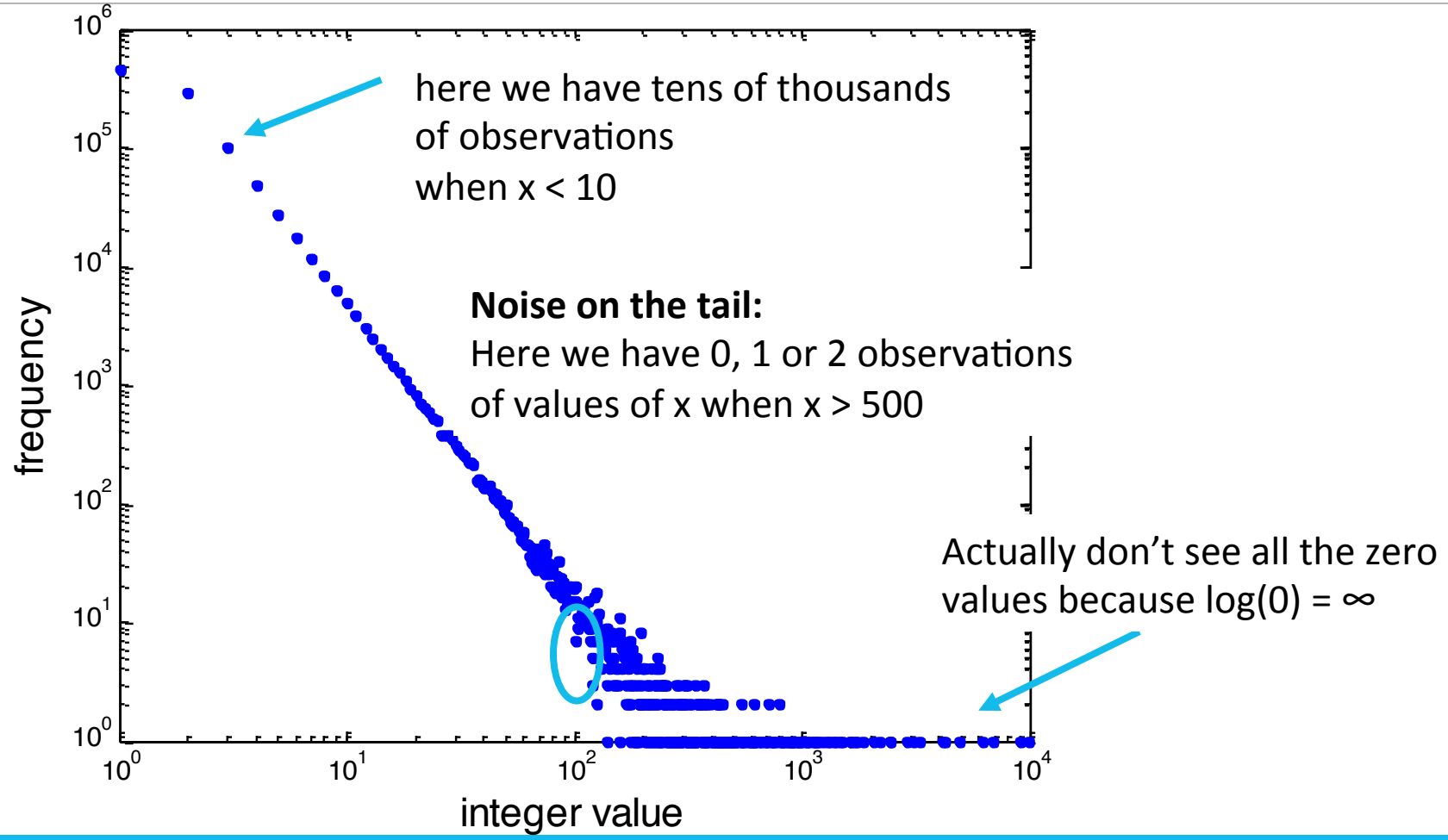
Uma distribuição que segue uma lei de potência é uma distribuição na forma:

$$p(x) = a * x^{-\gamma}$$

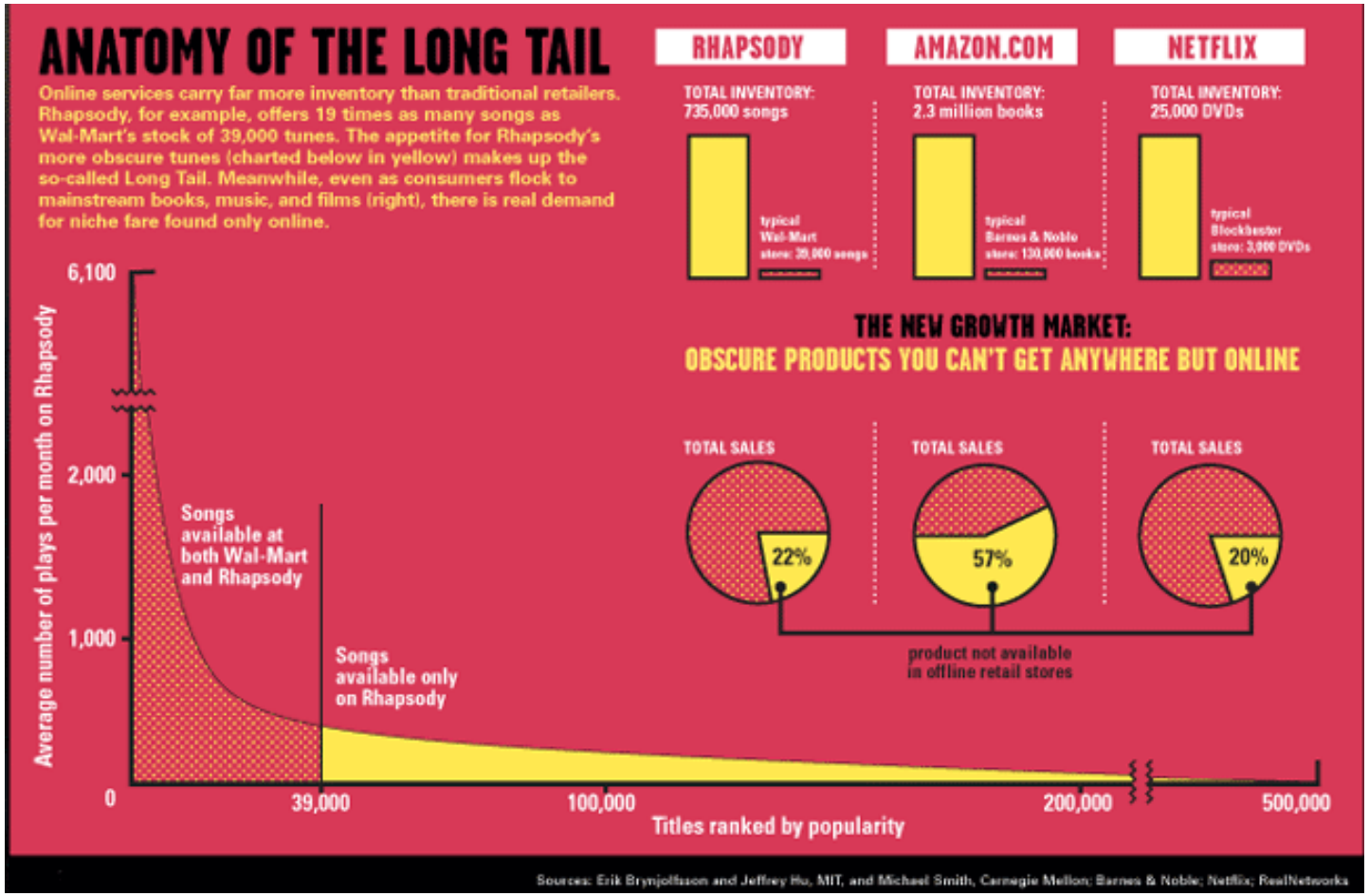
p(x) é a probabilidade de x ocorrer, sendo a uma constante de proporcionalidade e γ é o expoente da lei de potência

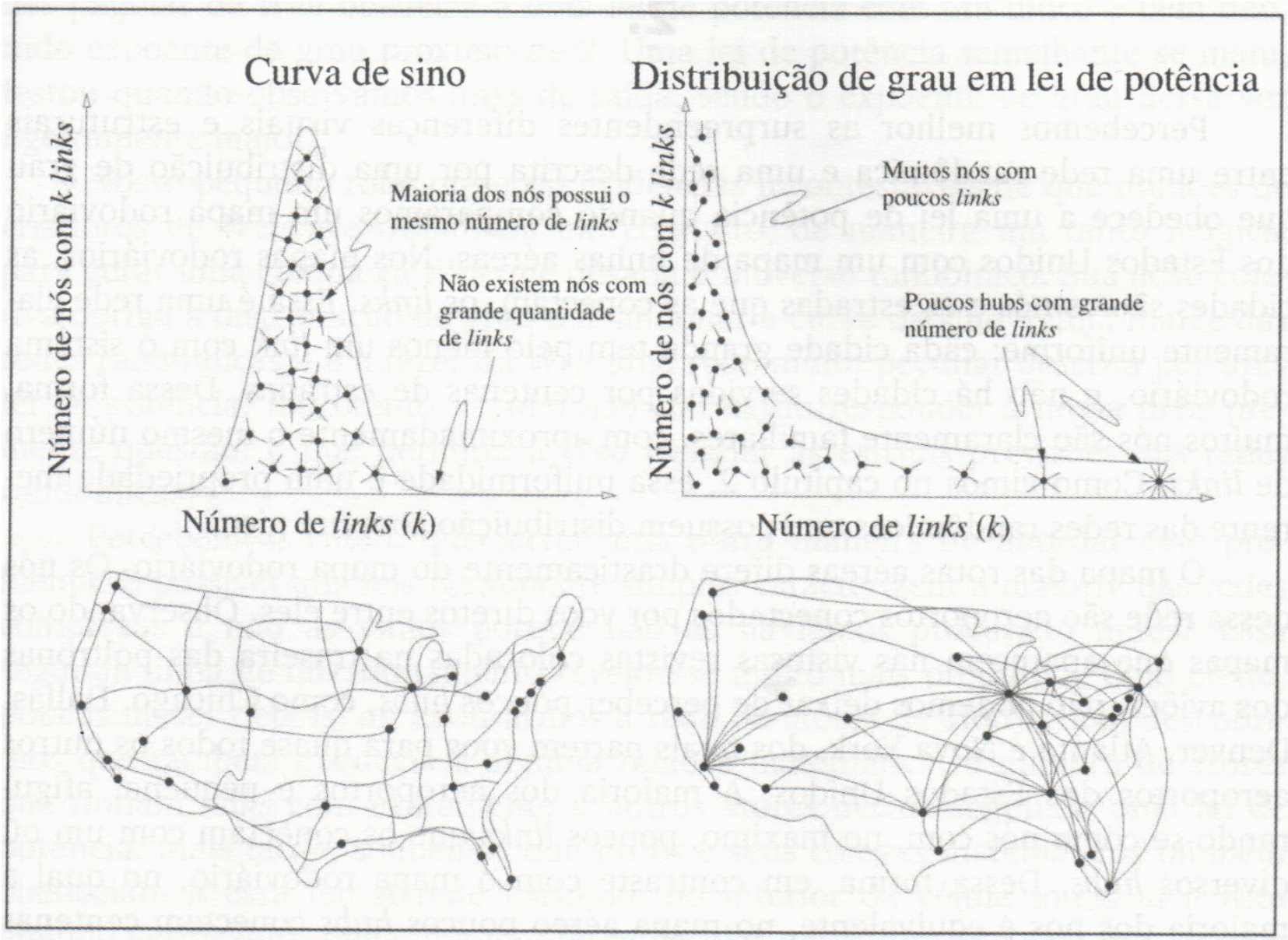
A. Clauset, C.R. Shalizi, and M.E.J. Newman, "Power-law distributions in empirical data" *SIAM Review* **51**(4), 661-703 (2009).

Power Law


here we have tens of thousands
of observations
when x < 10
Noise on the tail:
Here we have 0, 1 or 2 observations
of values of x when x > 500
Actually don't see all the zero
values because log(0) = ∞
frequency
integer value
10^0
10^1
10^2
10^3
10^4
10^5
10^6

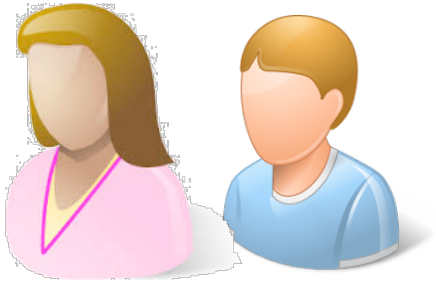
Anatomy of the Long Tail




Curva de sino
Distribuição de grau em lei de potência
Número de nós com k links
Maioria dos nós possui o mesmo número de links
Não existem nós com grande quantidade de links
Número de links (k)
Número de nós com k links
Muitos nós com poucos links
Poucos hubs com grande número de links
Número de links (k)

Stakeholders

Médicos



Pacientes

Seguradora

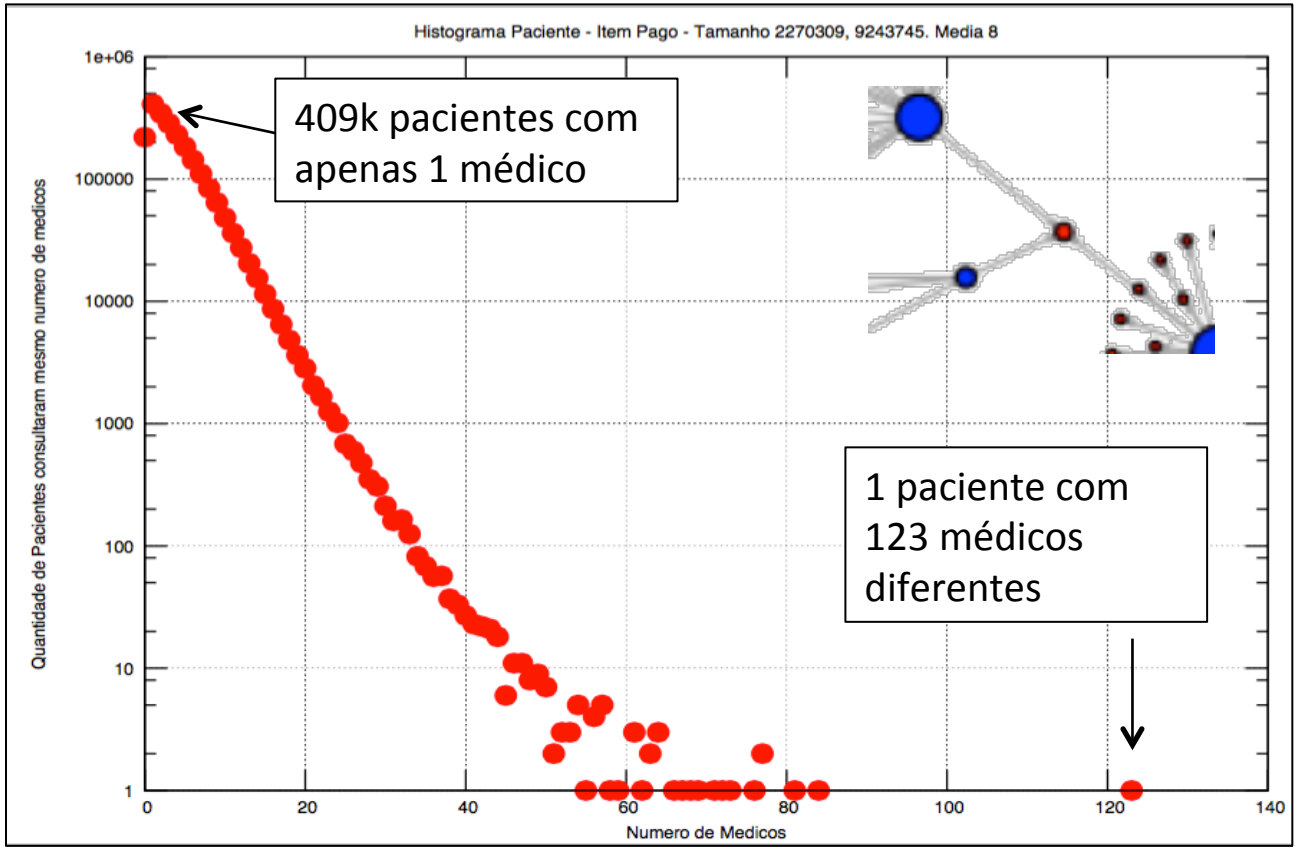
Prestadores

Reembolsos

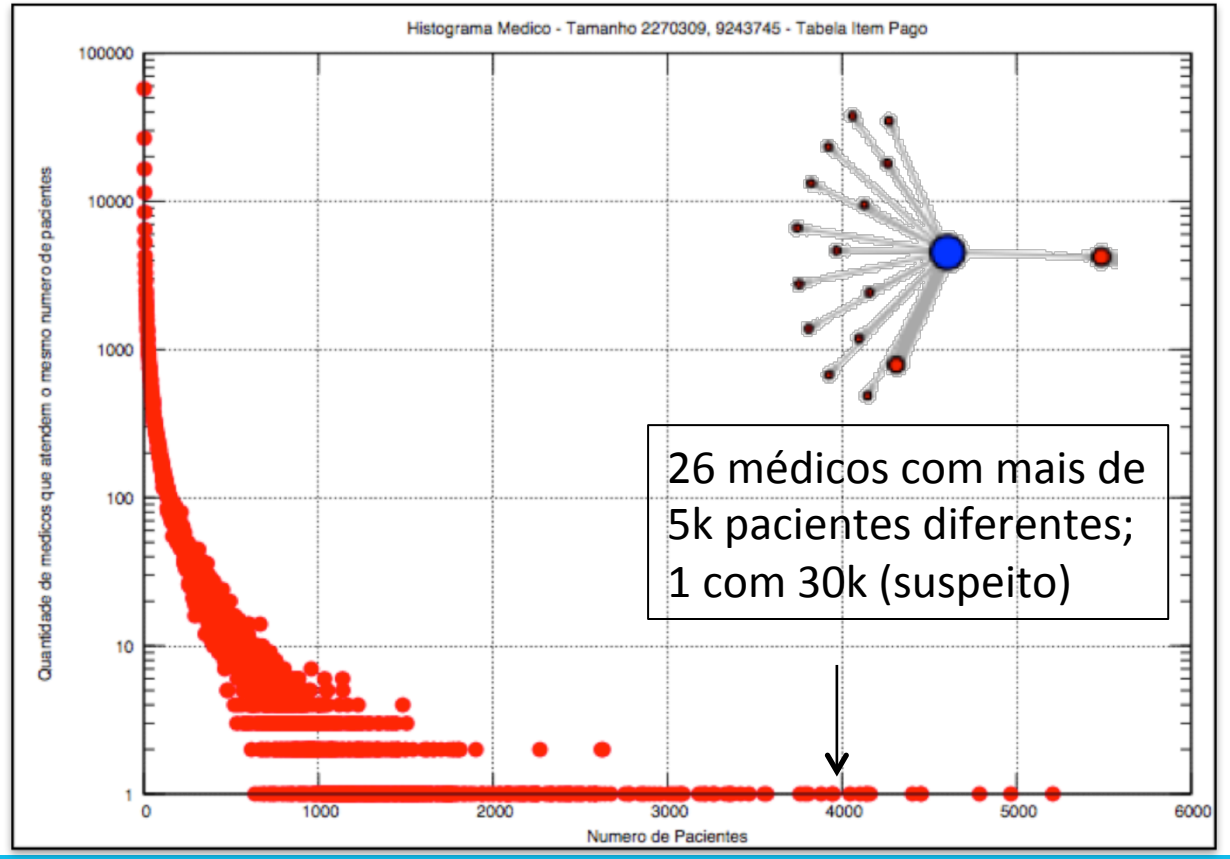
Serviços da área da saúde

Distribuição de grau de médicos e pacientes

Histograma de pacientes



Histograma de médicos



Computing Metrics

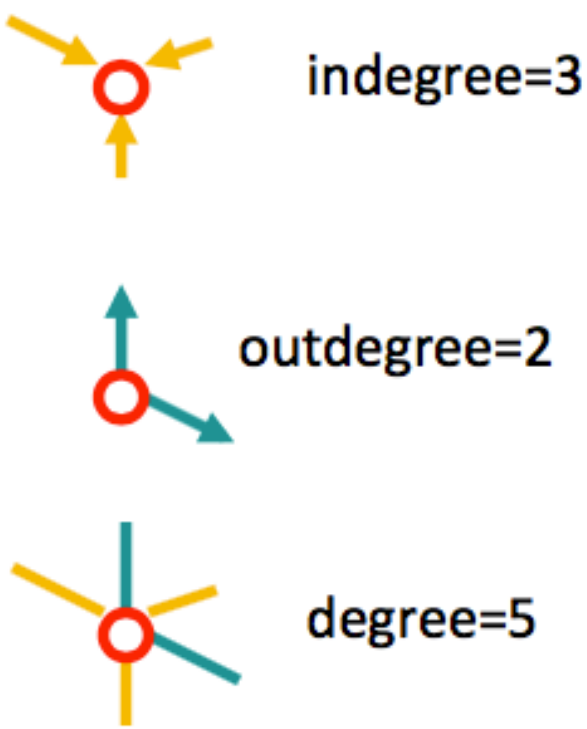
Degree & Degree Distribution

Connected Components

Nodes

Node network properties

- from immediate connections
  - In-degree
    how many directed edges (arcs) are incident on a node
  - Out-degree
    how many directed edges (arcs) originate at a node
  - degree (in or out)
    number of edges incident on a node



Graph in Spark

```
from __future__ import print_function
from pyspark.sql import Row, SparkSession
from pyspark.sql.functions import *
from graphframes import *
import os

caminho = "/Users/anaappel/spark2/graphframes/IA.csv"
#caminho = "/Users/anaappel/spark2/graphframes/soc-Epinions.txt"

spark = SparkSession.builder.appName("TutorialGraphFrames").getOrCreate()

sep = ';'

df = spark.read.csv(caminho, None, sep, None, None, None, None, True)
nos = df.select(df['src'])
nos = nos.select(df['src'].cast('int')).withColumnRenamed('src', 'id').dropDuplicates()

arestas = df.select('*')


g = GraphFrame(nos,arestas)
```

Node degree from Matrix Values

Outdegree(i) = $\sum_{j=1}^{n} A_{ij}$

A = $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$

example: outdegree for node 3 is 2, which we obtain by summing the number of non-zero entries in the 3rd *row*

$$\sum_{j=1}^{n} A_{3j}$$

Indegree(j) = $\sum_{i=1}^{n} A_{ij}$

A = $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$

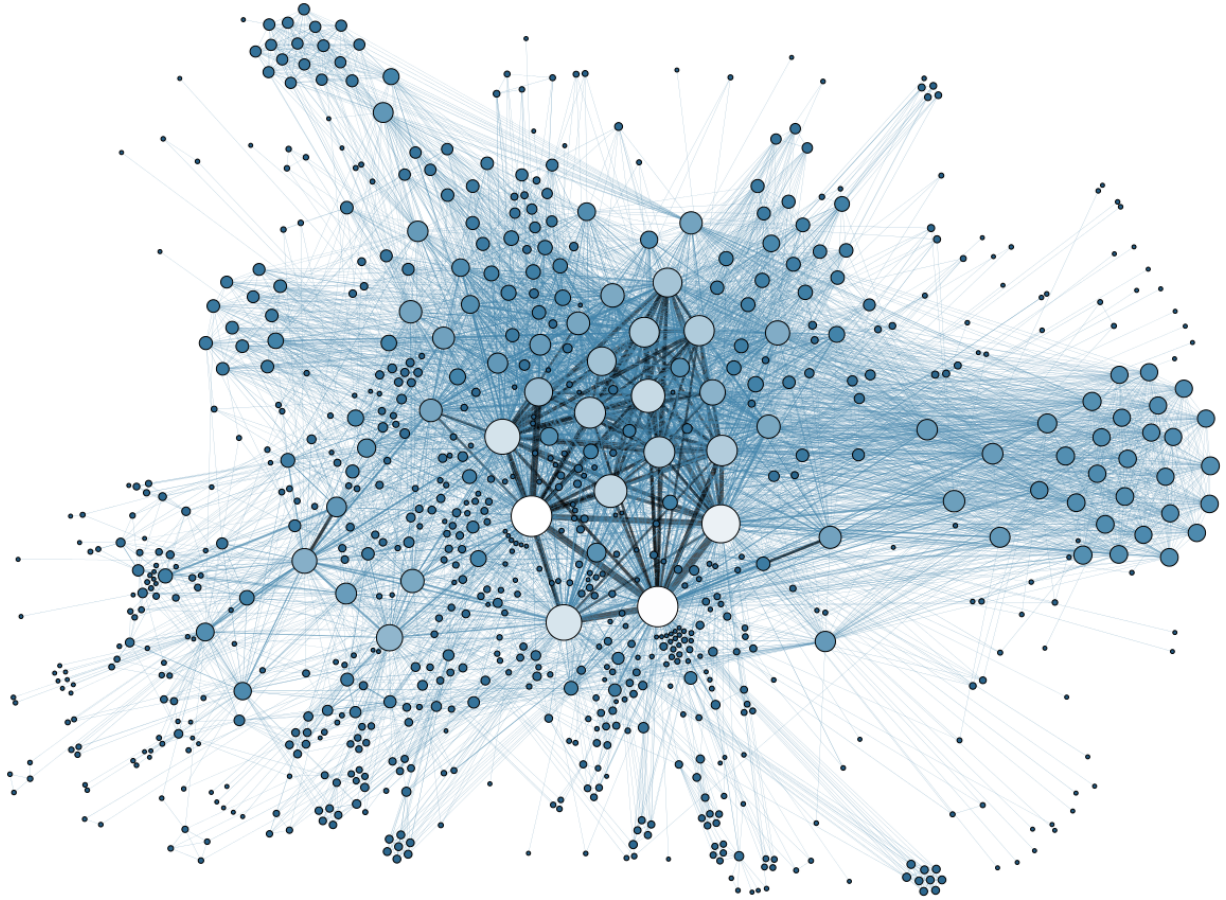
example: the indegree for node 3 is 1, which we obtain by summing the number of non-zero entries in the 3rd *column*

$$\sum_{i=1}^{n} A_{i3}$$

Degree in Spark

```
grau = {}
#g.inDegrees.show()
graus = g.inDegrees
grausLista = graus.orderBy(graus.inDegree, ascending=False).collect()
for row in grausLista:
    if(grau.has_key(row['inDegree'])):
        grau[row['inDegree']]+=1
    else:
        grau[row['inDegree']] = 1

with open(os.path.join(os.getcwd(), "degrees distribution.csv"), 'w') as doc:
    doc.write("degree;amount\n")
    for key in grau:
       doc.write(str(key)+';'+str(grau[key])+'\n')
```

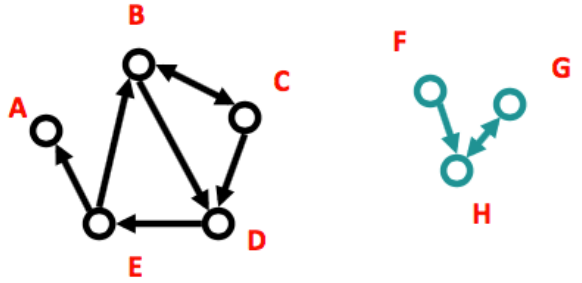
Is Everything Connected?



Connected Components

Strongly connected components

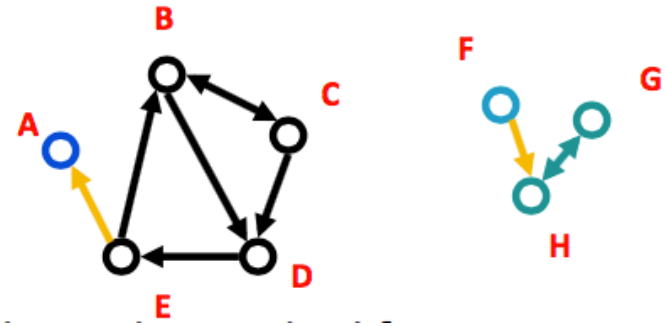
- Each node within the component can be reached from every other node in the component by following directed links
  - B C D E
  - A
  - G H
  - F



Weakly connected components:

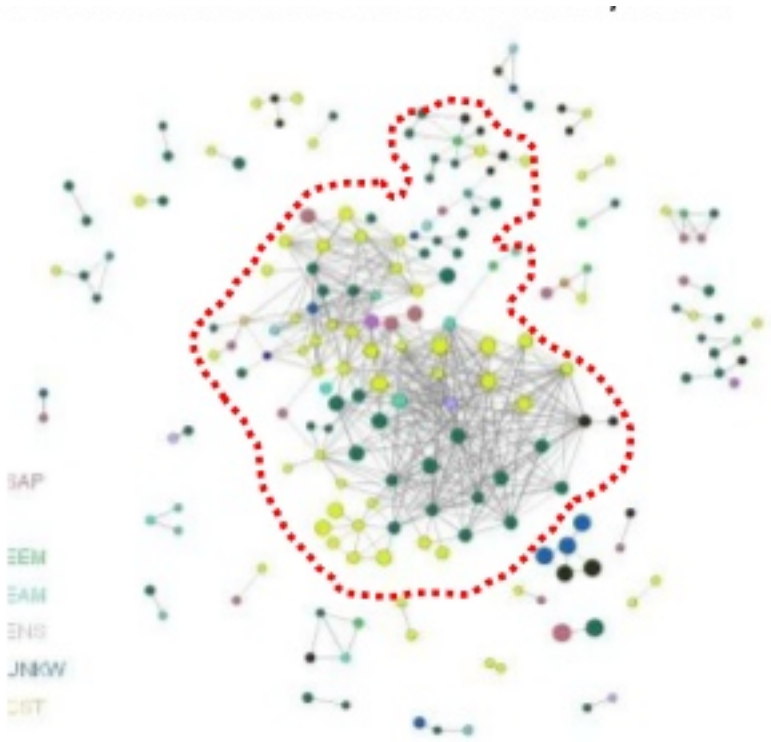
- every node can be reached from every other node by following links in either direction
  - ABCDE
  - G H F

In undirected networks one talks simply about ‘connected components’



Giant Component

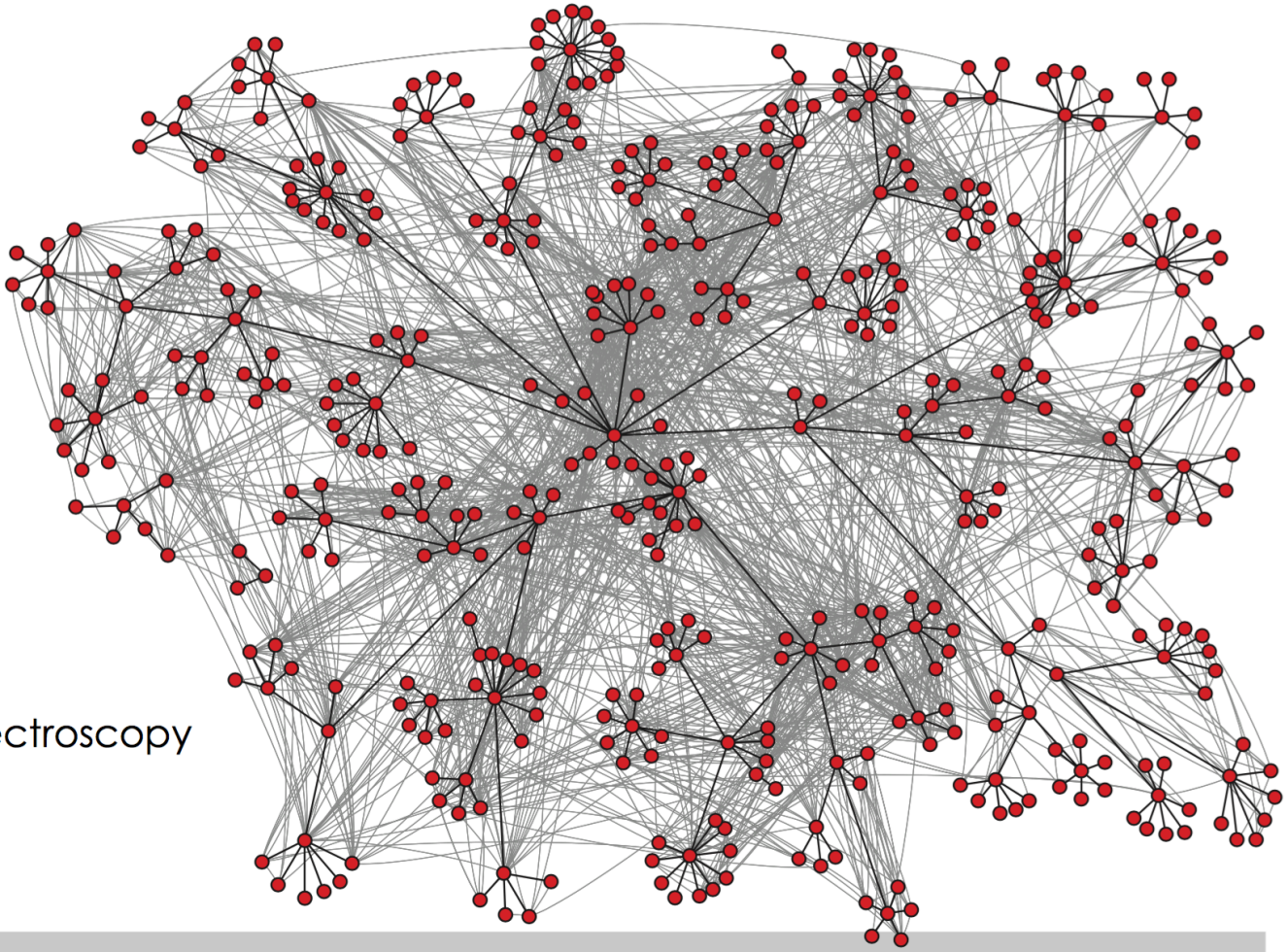
if the largest component encompasses a significant fraction of the graph, it is called the **giant component**



Connected Component in Spark

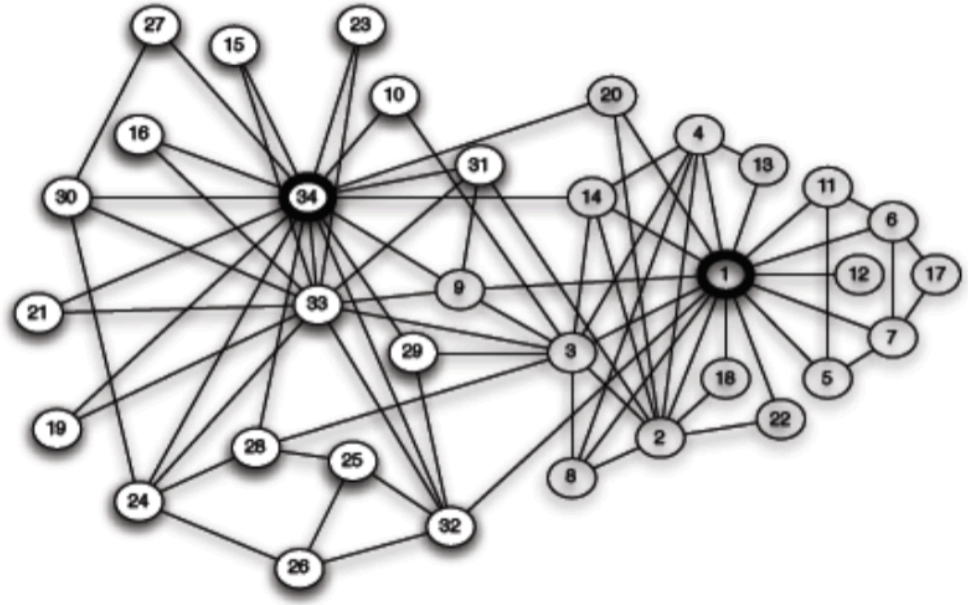
```
comp = {}
size = {}
g.connectedComponents().show()
for row in g.connectedComponents().collect():
    if(comp.has_key(row['component'])):
        comp[row['component']]+=1
    else:
        comp[row['component']]=1

for key in comp:
    if(size.has_key(comp[key])):
        size[comp[key]]+=1
    else:
        size[comp[key]]=1

with open(os.path.join(os.getcwd(), "connected components distribution.csv"), 'w') as doc:
    doc.write("size;count\n")
    for key in size:
       doc.write(str(key)+';'+str(size[key])+'\n')
```

Why look for community structure?

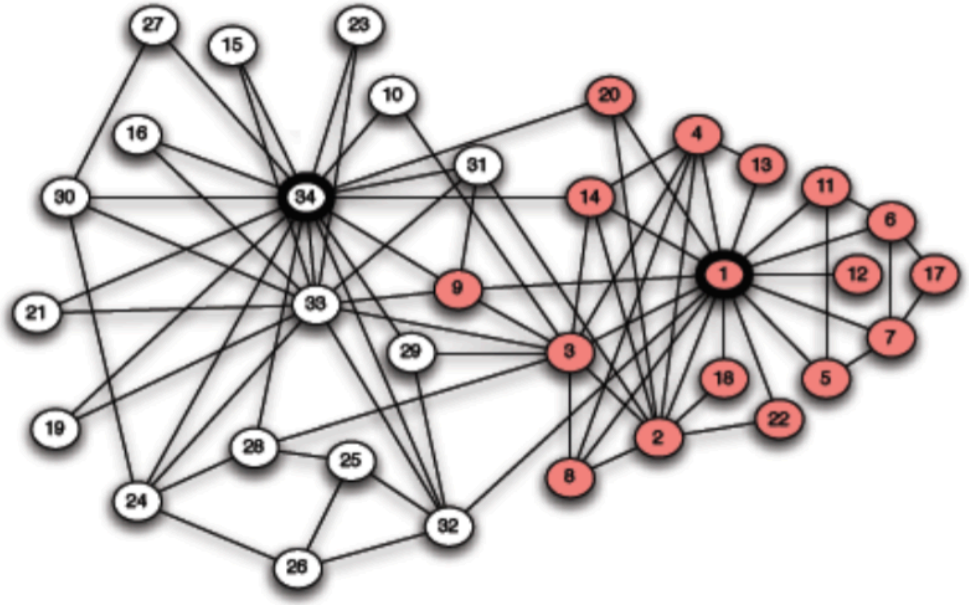


example:
email spectroscopy

Zachary Karate Club



(a) *Karate club network*



(b) *After a split into two clubs*

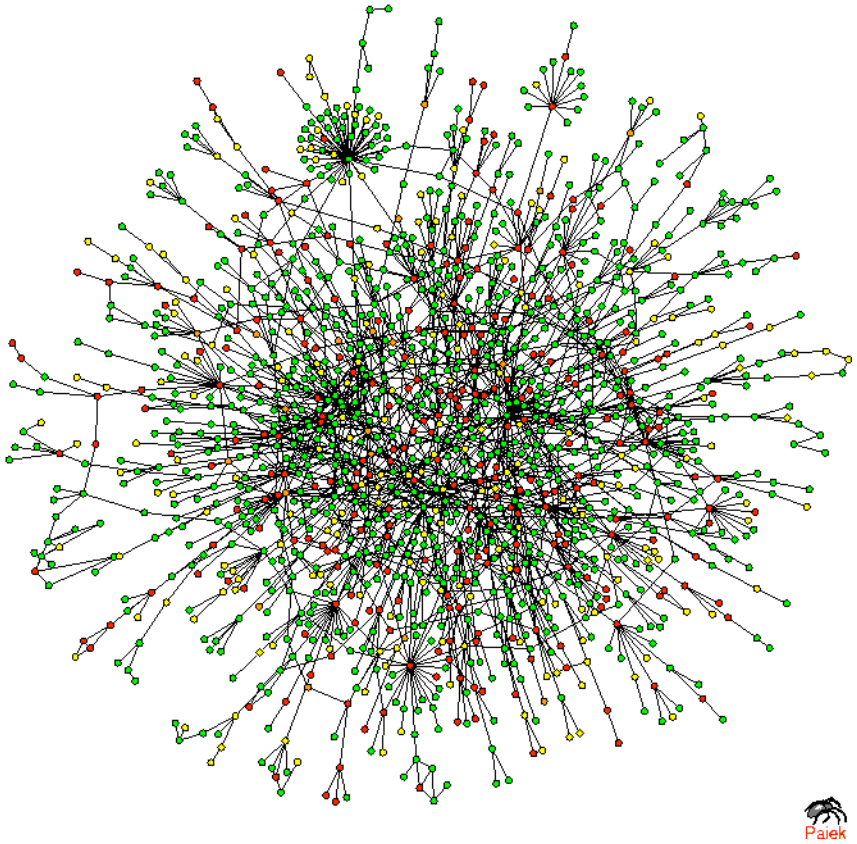
Why do it: gain understanding

- Gain understanding of networks
- Discover communities of practice
- Measure isolation of groups
- Understand opinion dynamics / adoption

Why do it: store & compute

If your network needs to be distributed accross the world or over many machines, you want to minimize the number of edges in-between

Community Structure



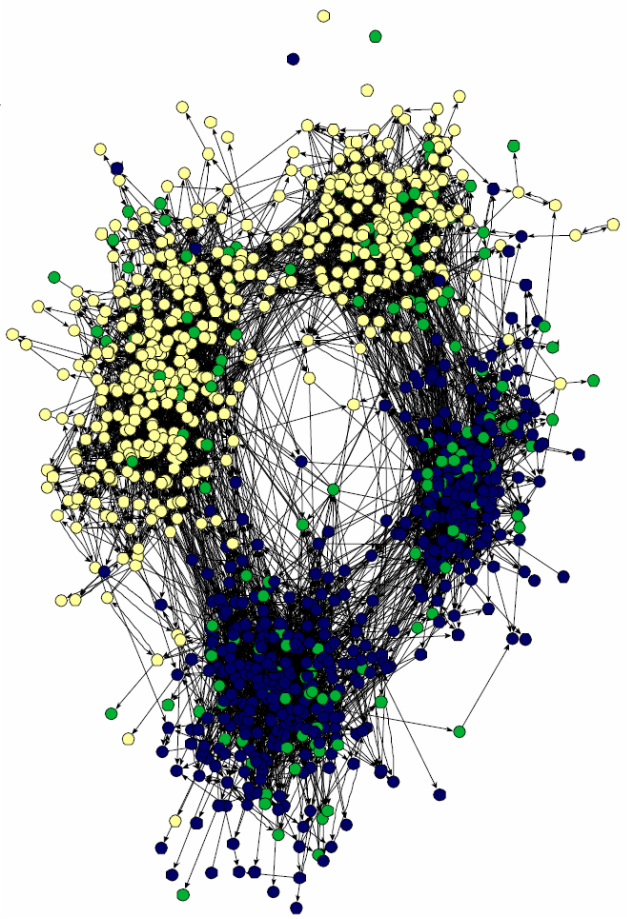
Communities

Many networks present a community structure

- Groups of nodes that have a higher number of connections within each other than with nodes that belong to other groups
- People are naturally grouped basing on their interests, age, occupation, ...

How to find communities:

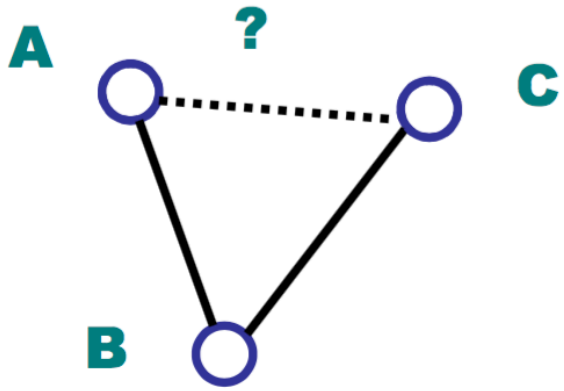
- Spectral clustering
- Hierarchical clustering based on connections
- Block model
- Partitioning



Rede de amizade de crianças em uma escola

Transitivity, triadic closure, clustering

- Transitivity:
  - if A is connected to B and B is connected to C
  - what is the probability that A is connected to C? my friends' friends are likely to be my friends



Counting Triangles in Spark

```
triangs = g.triangleCount()
triangsList = triangs.orderBy(triangs['count'], ascending=False)
triangsList.show()
for row in triangsList.collect():
    if(quant.has_key(row['count'])):
        quant[row['count']]+=1
    else:
        quant[row['count']] = 1

with open(os.path.join(os.getcwd(), "triangles distribution.csv"), 'w') as doc:
    doc.write("triangles;count\n")
    for key in quant:
       doc.write(str(key)+';'+str(quant[key])+'\n')
```

Local clustering coefficient (Watts&Strogatz 1998)

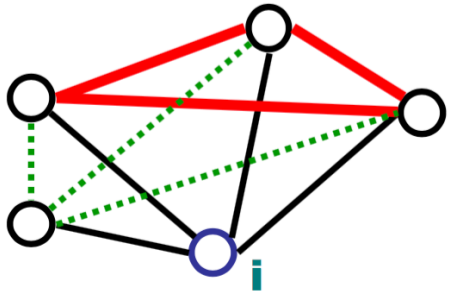
- For a vertex I
  - The fraction pairs of neighbors of the node that are themselves connected
  - Let *ni* be the number of neighbors of vertex *i*

$$C(v_i) = \frac{2 * \Delta(v_i)}{d(v_i) * (d(v_i) - 1)}$$

Clustering

Global clustering coefficient
3 x number of triangles in the graph number of connected triples of vertices

$$C(\mathcal{G}) = \frac{1}{N} * \sum_{i=1}^{N} C(v_i)$$



link present
link absent

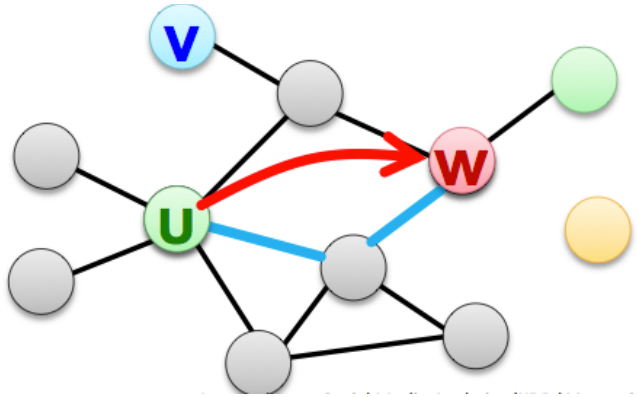
$n_i = 4$
max number of connections:
4*3/2 = 6
3 connections present
$C_i = 3/6 = 0.5$

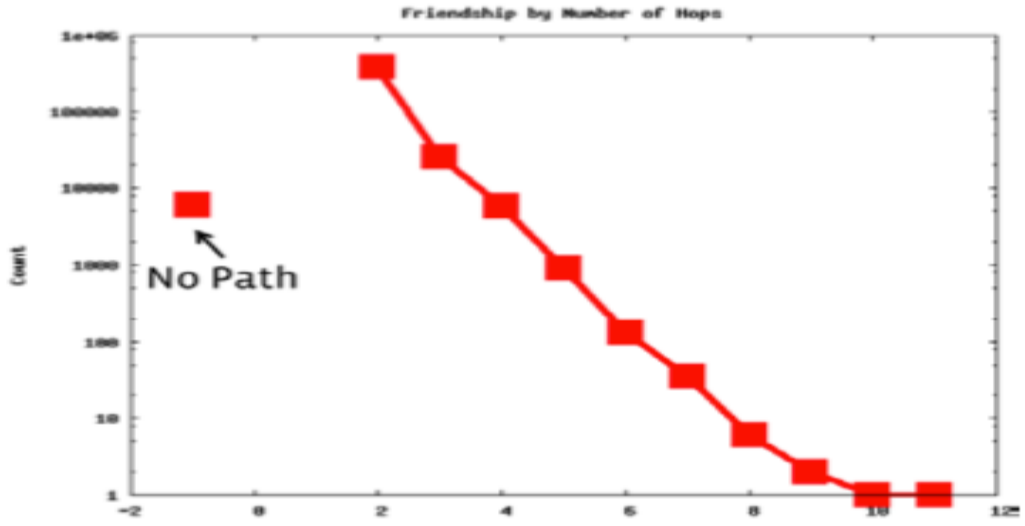
Link Prediction

People on Facebook who you may know

92% of new friendships on Facebook are friend-of-a-friend

Friends in common help





Link Prediction

Given a snapshot of a social network

Question: infer which new interactions among its members are likely to occur in the near future [Liben-Nowell and Kleinberg, 2004, 2007]

Link Prediction

Link prediction. A network is changing over time. Given a snapshot of a network at time t, predict edges added in the interval (t,t')

Link completion (missing links identification). Given a network, infer links that are consistent with the structure, but missing (find unobserved edges)

Link reliability. Estimate the reliability of given links in the graph.

Predictions: link existence, link weight, link type

Scoring Algorithm

Link prediction by proximity scoring

1. For each pair of nodes compute proximity (similarity) score c(v1,v2)
2. Sort all pairs by the decreasing score
3. Select top n pairs (or above some threshold) as new links

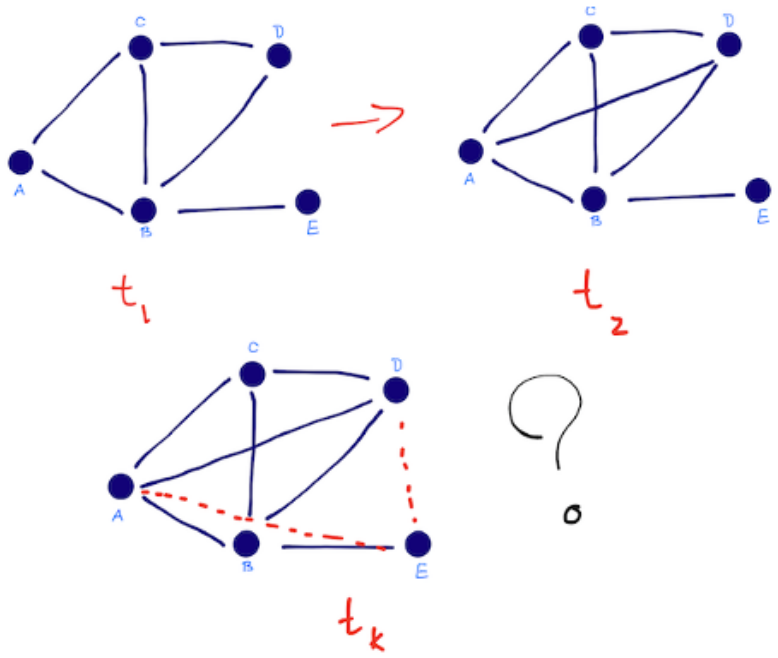
$G = \langle V, E \rangle$ ← Social Network

$e = \langle u, v \rangle \in E$ ← Interaction between u and V

$G[t_0, t_1]$ ← Given Subgraph as training set

$G[t_2, t_3]$ ← Infer new Edges/used for testing

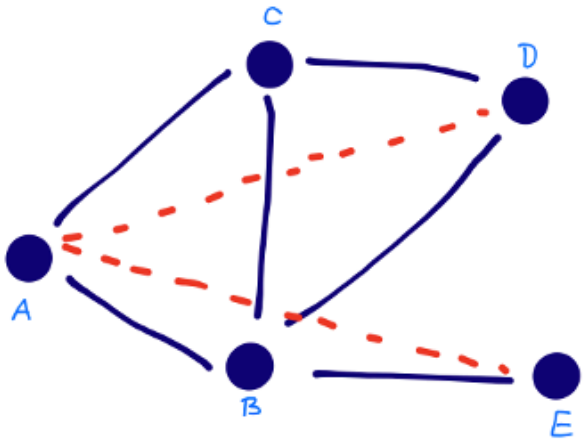
Score(u,v) ← Liklihood that u and v share an edge (Proximity or Similarity)



Common Neighbors

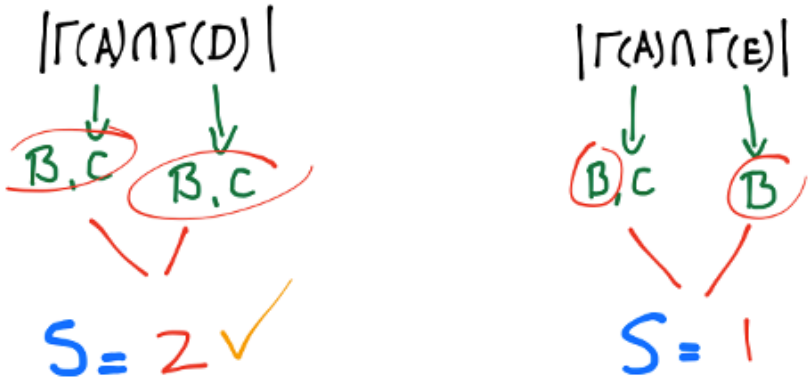
The common-neighbors predictor captures the notion that two strangers who have a common friend may be introduced by that friend.

This introduction has the effect of “closing a triangle” in the graph and feels like a common mechanism in real life.



$$Score(x,y) = |\Gamma(x) \cap \Gamma(y)|$$

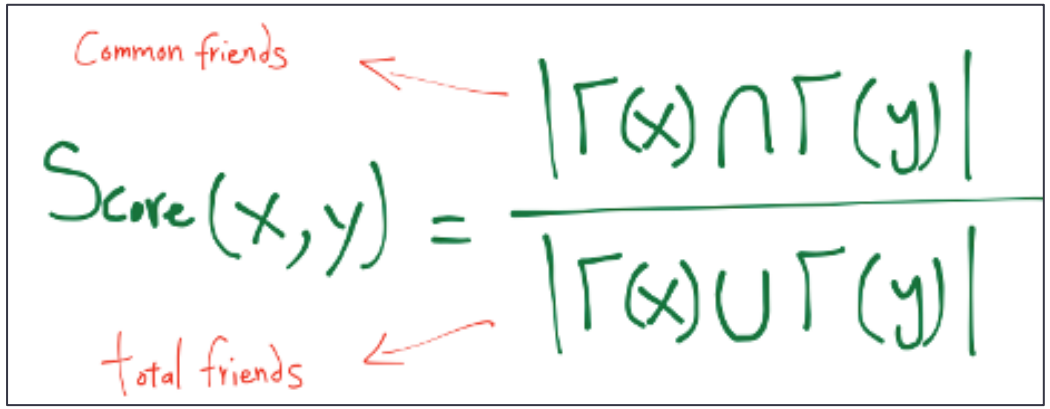
Neighbors of x



Jaccard's Coefficient

The Jaccard coefficient—a similarity metric that is commonly used in information retrieval—measures the probability that both x and y have a feature f, for a randomly selected feature f that either x or y has.

If we take "features" here to be neighbors, then this measure captures the intuitively appealing notion that the proportion of the coauthors of x who have also worked with y (and vice versa) is a good measure of the similarity of x and y.



Adamic/Adar

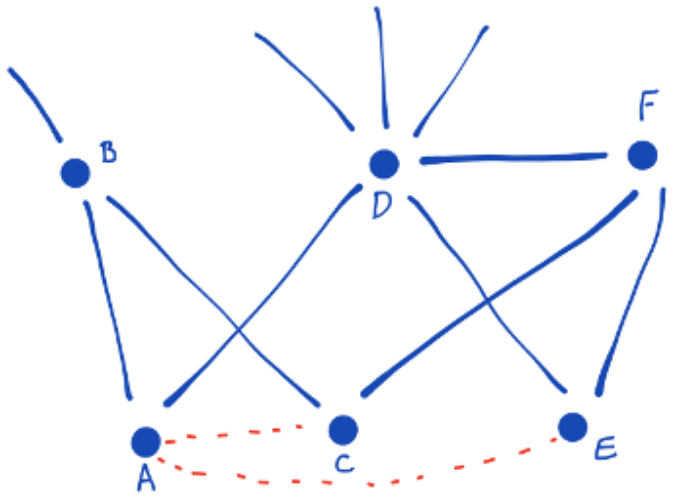
This measure refines the simple counting of common features by weighting rarer features more heavily.

For x and y to be introduced by a common friend z, person z will have to choose to introduce the pair ⟨x,y⟩ from (choose |Γ(z)| with 2) pairs of his friends; thus an unpopular person (someone with not a lot of friends) may be more likely to introduce a particular pair of his friends to each other.

$$Score(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log|\Gamma(z)|}$$

Frequency of z

Weighting rare features more heavily



$\Gamma(A) \cap \Gamma(C) = B$

$\frac{1}{\log(\Gamma(B))} = \frac{1}{\log 3} = 2.09$

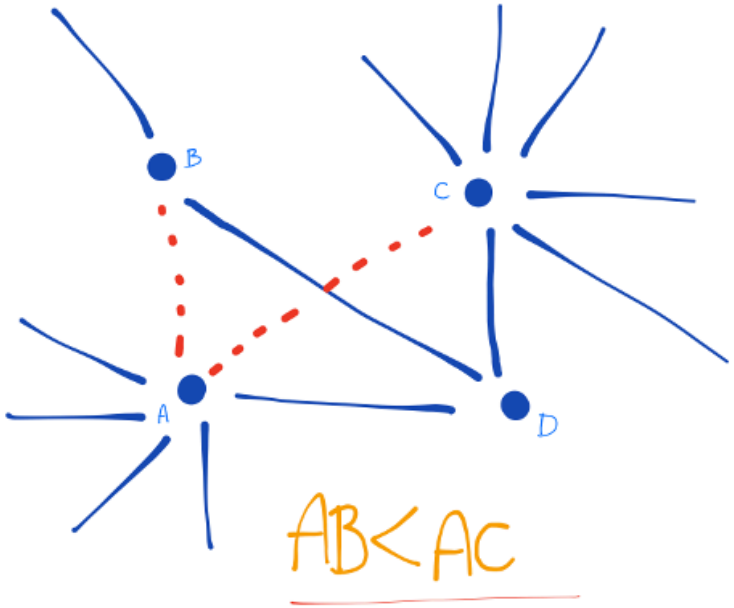
$\Gamma(A) \cap \Gamma(E) = D$

$\frac{1}{\log(\Gamma(D))} = \frac{1}{\log 6} = 1.2$

Preferential Attachment

One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in s ıe rich get richer. We estimate how ”rich” our two vertices between the number of friends (|Γ(x)|) or followers ea

$$Score(x,y) = |\Gamma(x)| \cdot |\Gamma(y)|$$

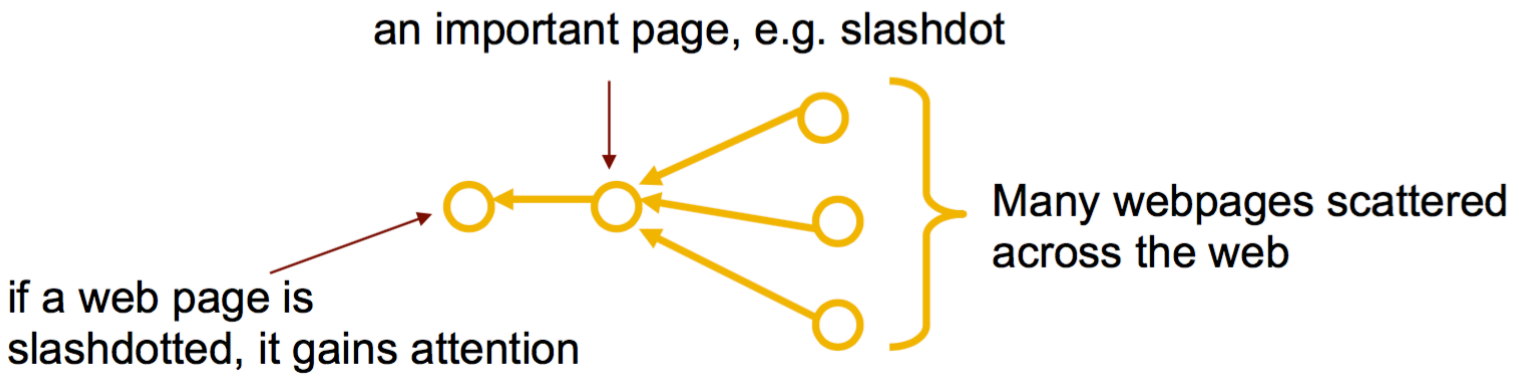


Centrality

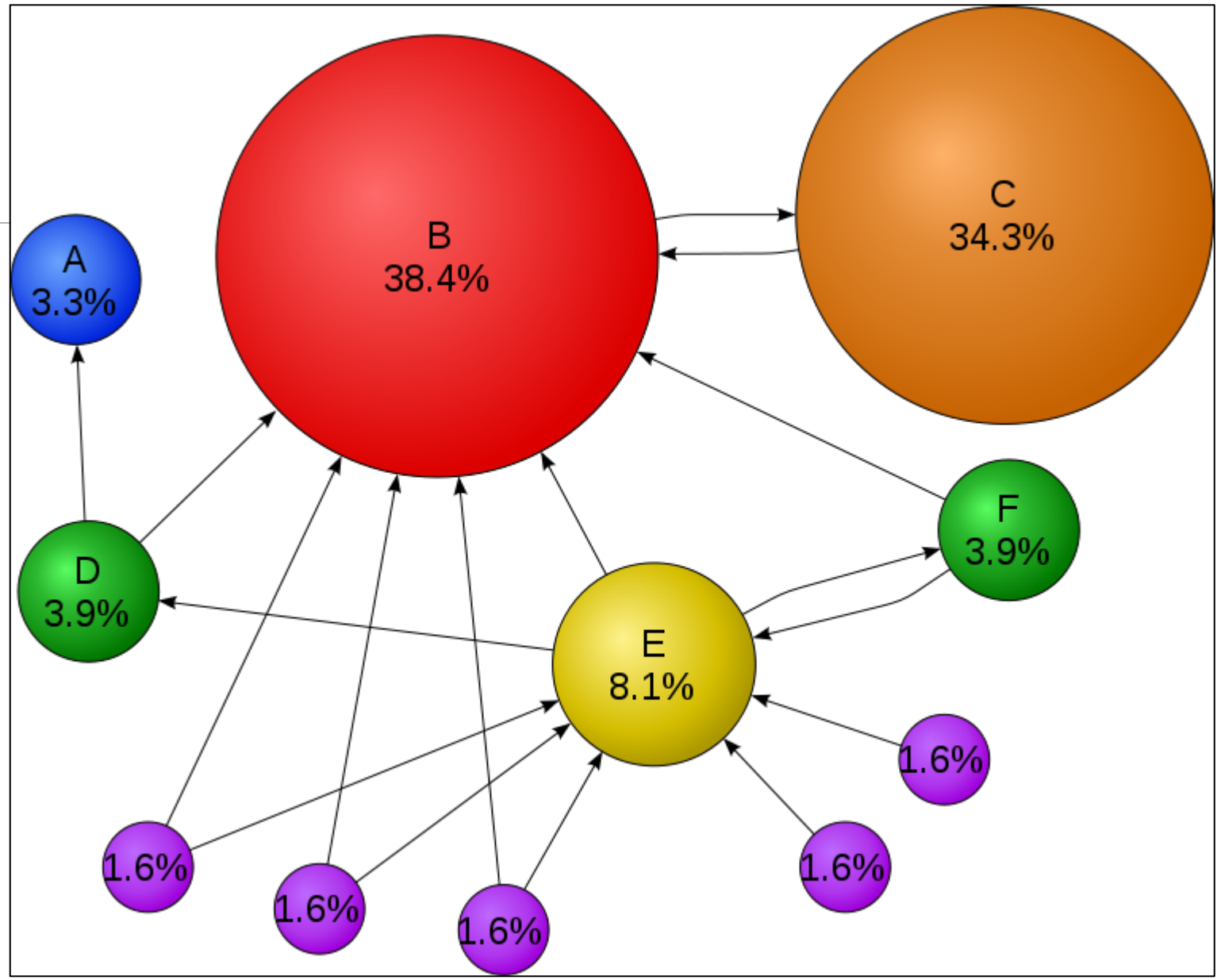
- Node centrality can reveal the relative importance of nodes within the network
- Choose a measure appropriate to the question you are asking

Eigenvector centrality in directed Networks

- PageRank (centrality) brings order to the Web:
  - it's not just the pages that point to you, but how many pages point to those pages, etc.
  - more difficult to artificially inflate centrality with a recursive definition



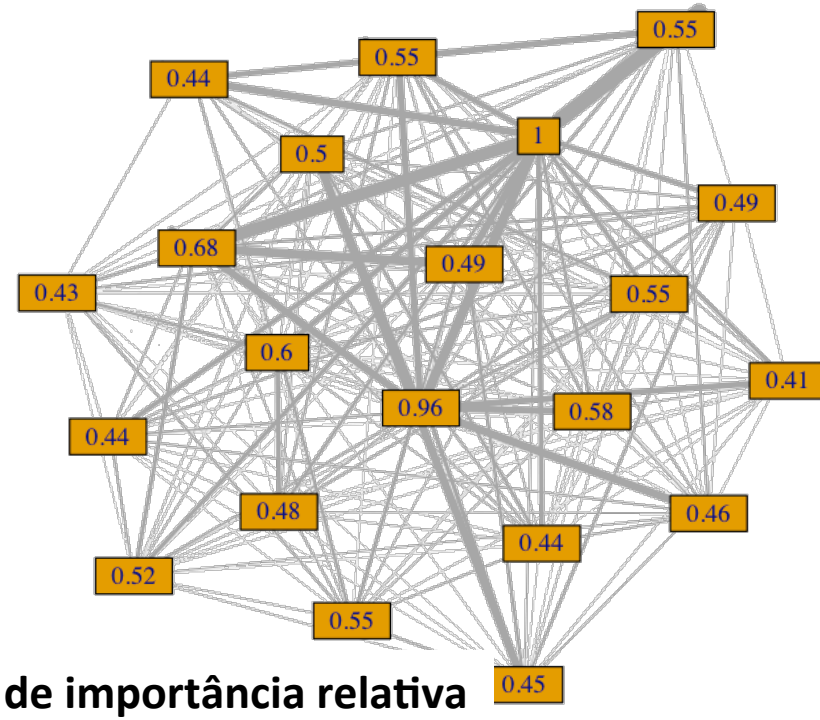
PageRank



2. Papel dos médicos na rede

Resultados

- Calculamos a métrica para a base toda, e para 5 quartis.
- A evolução por quartis do topo dos rankings de médicos (métrica *eigenvalue*) varia pouco demonstrando estabilidade da métrica.
- No subconjunto de médicos com maior valor da métrica, observamos um nível muito forte de compartilhamento de pacientes.
- O tamanho desse subconjunto varia entre dezenas e centenas de médicos.
- Medimos a correlação de métrica entre vizinhos: *corr*=0.38.



Métrica de importância relativa

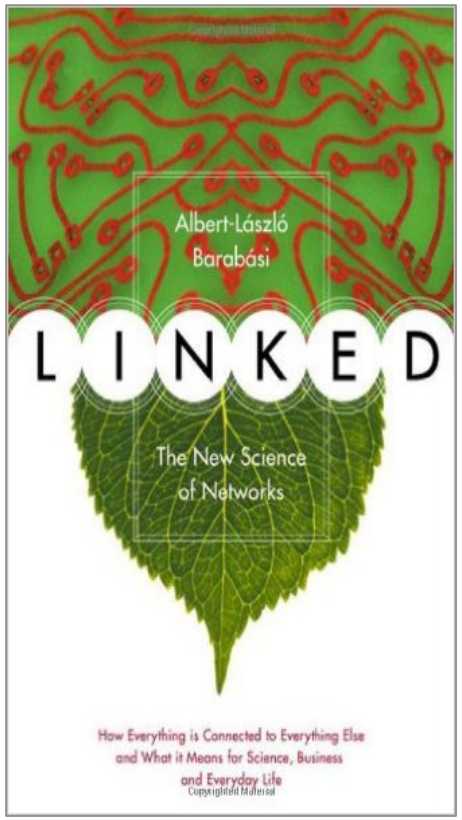
2Q 2014

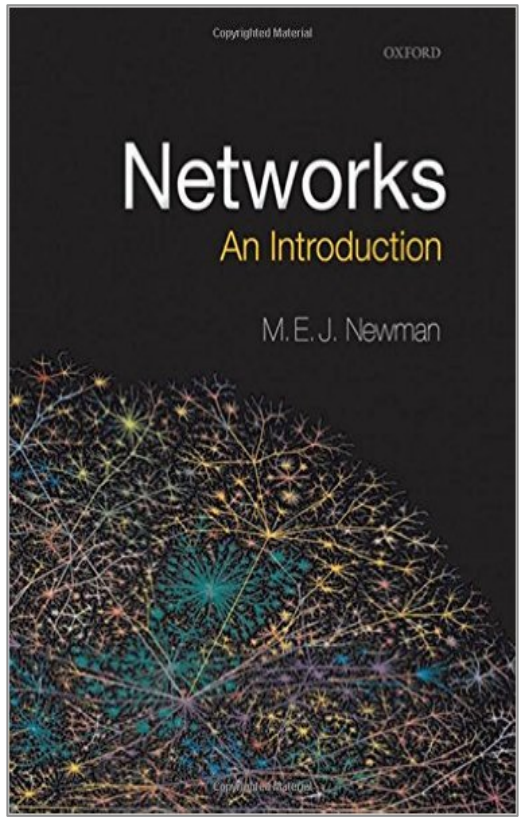
Médico	*eigen*	Rank	Grau
M_{SP} 153	1	1	253
M_{SP} 139	0.55	8	335

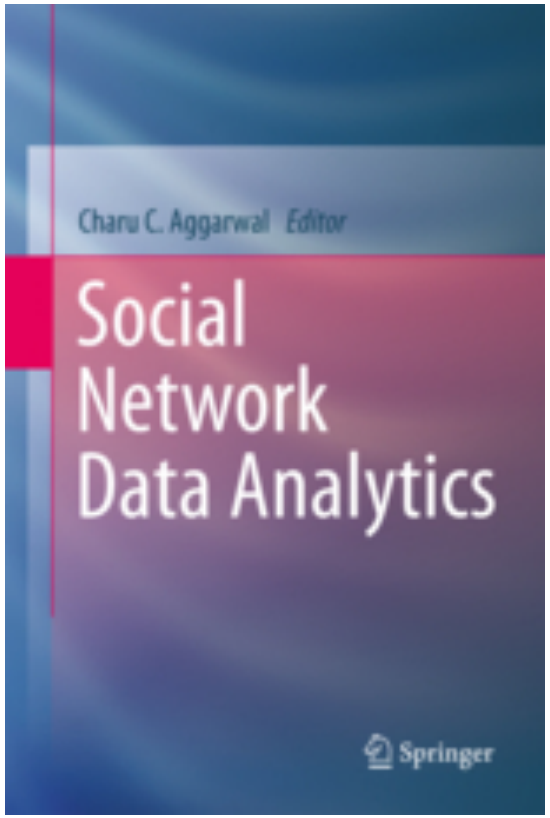
PageRank on Spark

```
results = g.pageRank(resetProbability=0.15, tol=0.01)
results.vertices.select("id", "pagerank").show()
```

Complex Networks books

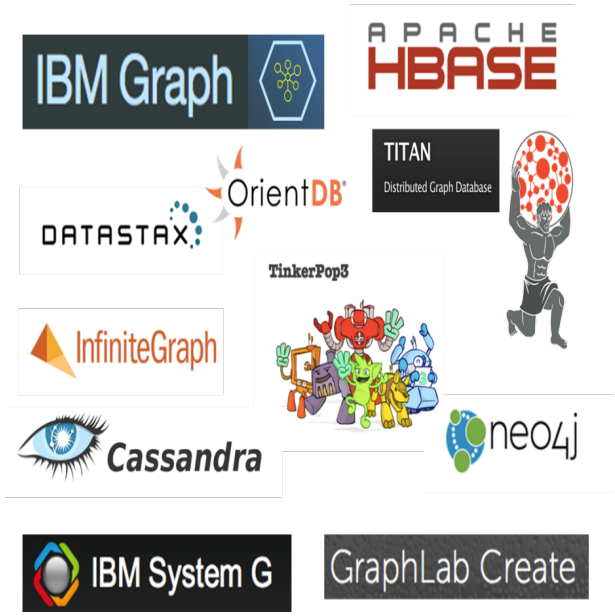




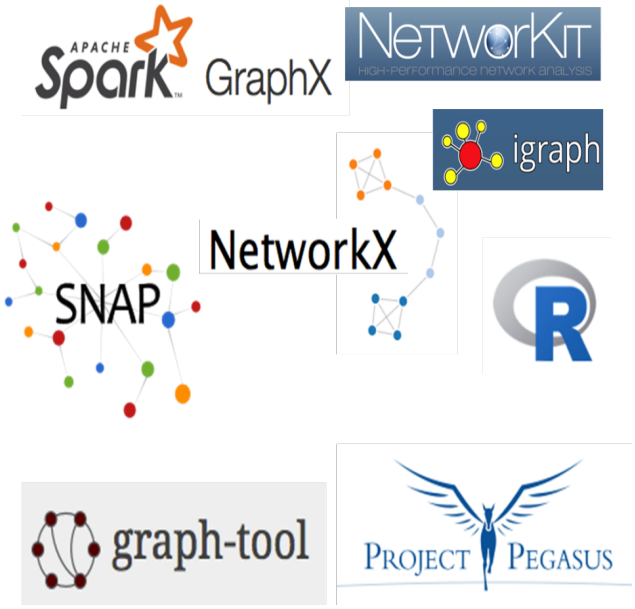


Several Tool to analyse Graphs

GRAPH ANALYTICS



Database



Engines

Visualization

Scalability

Current status

DataFrame-based parts benefit from DataFrame scalability + performance optimizations (Catalyst, Tungsten).

GraphX wrappers areas fast as GraphX (+conversion over head).

WIP

• GraphX has optimizations which are not yet ported to GraphFrames. • See next slide...

WIP optimizations

Join elimination

- GraphFrame algorithms require lots of joins.
- Not all joins are necessary

Solution:

- Vertex IDs serve as unique keys.
- Tracking keys allows Catalyst to eliminate some joins.

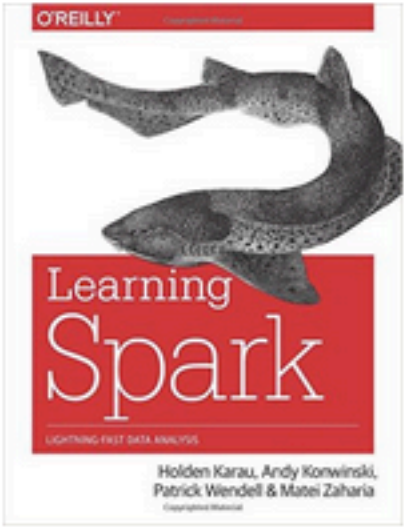
Materialized views

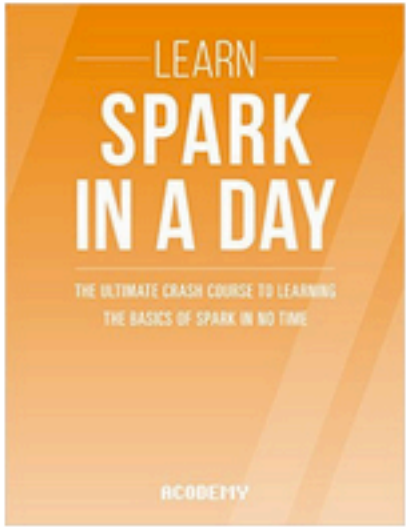
- Data locality for common use cases
- Message-passing algorithms often need "triplet view" (src, edge, dst)

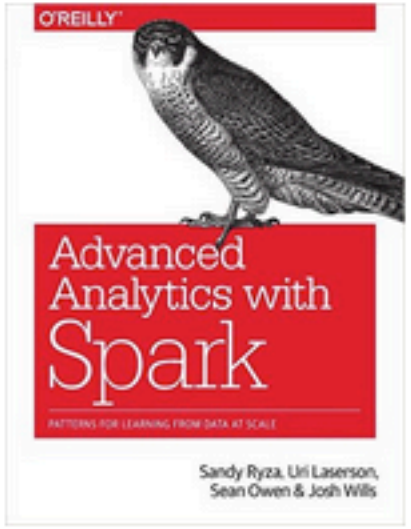
Solution:

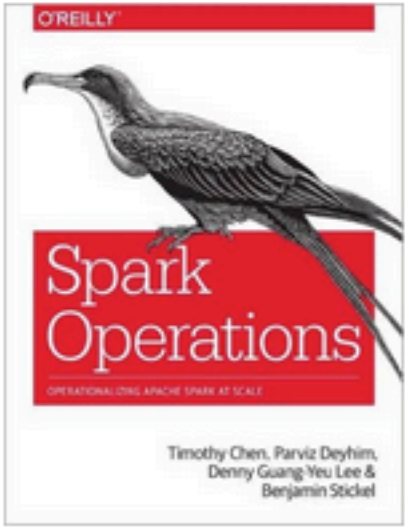
- Materialize specific views
- Analogous to GraphX's "replicated vertex view"

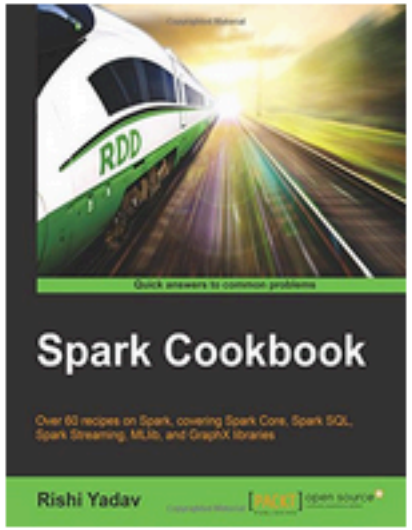
For more info & benchmark results, see Ankur Dave's SSE 2016 talk.
https://spark-summit.org/east-2016/events/graphframes-graph-queries-in-spark-sql/

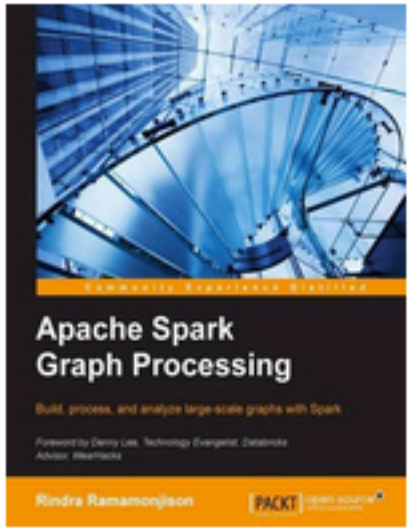

O'REILLY
Learning
Spark
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

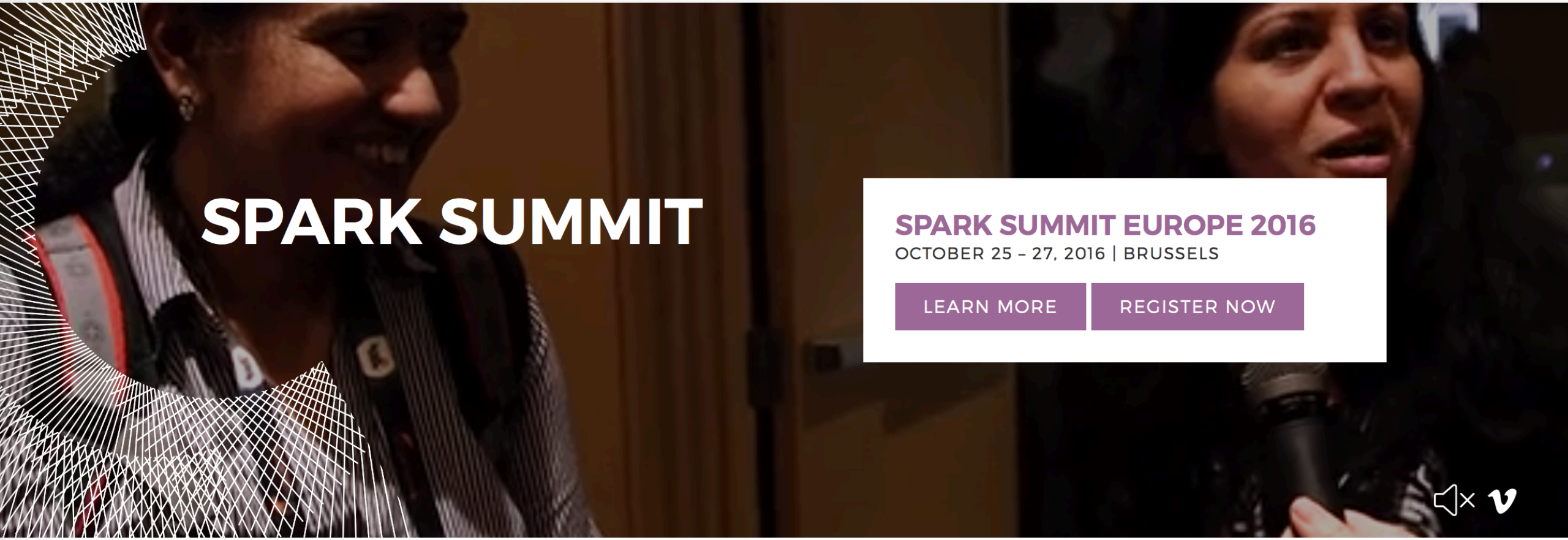

LEARN
SPARK
IN A DAY
THE ULTIMATE CRASH COURSE TO LEARNING
THE BASICS OF SPARK IN NO TIME
ACODEMY


O'REILLY
Advanced
Analytics with
Spark
Sandy Ryza, Uri Laserson,
Sean Owen & Josh Wills


O'REILLY
Spark
Operations
Timothy Chen, Parviz Deyhim,
Denny Guang-Yeu Lee &
Benjamin Stickel


RDD
Spark Cookbook
Rishi Yadav
PACKT open source


Apache Spark
Graph Processing
Rindra Ramamonjison
PACKT open source



BOSTON

SPARK SUMMIT EAST

FEBRUARY 7 – 9, 2017
CALL FOR PRESENTATIONS

SAN FRANCISCO

SPARK SUMMIT

JUNE 6 – 8, 2016
WATCH THE TALKS

UP NEXT BRUSSELS

SPARK SUMMIT EUROPE

OCTOBER 25 – 27, 2016
REGISTER NOW

References

- Large Scale Distributed Data Science using Apache Spark – KDD 2015
- Spark Summit presentation
- Spark Apache Web Site
- Zaharia, M, et al. 2012. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12).

Thank You!!! ;-)

Graph Analytics using Spark 2.0

ANA PAULA APPEL (APAPPEL@BR.IBM.COM)

RENAN FRANCISCO SANTOS SOUZA (RFSOUZA@BR.IBM.COM)

Apply if you like!!!

https://ibm.referrals.selectminds.com/jobs/research-staff-member-ndash%3B-social-data-analytics-126168