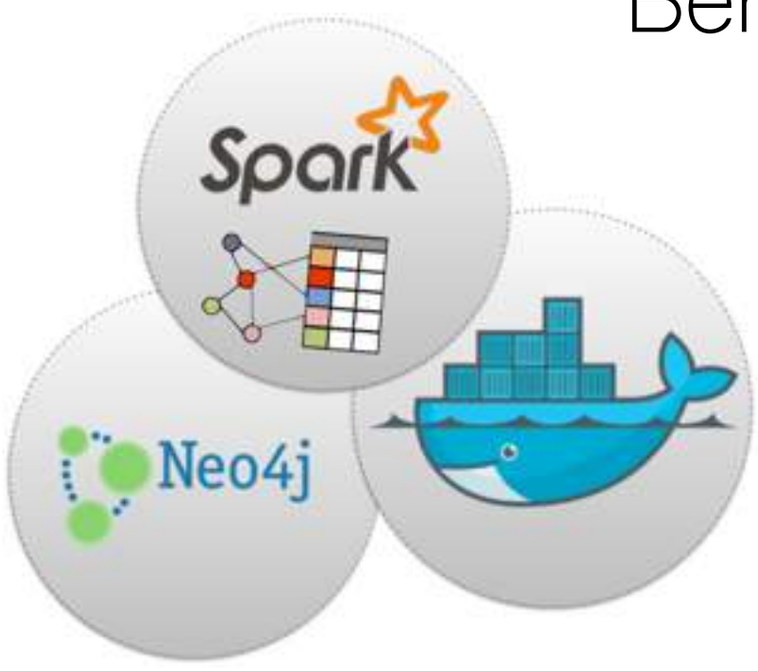
# Big Graph Analytics on Neo4j with Apache Spark

Michael Hunger

Original work by Kenny Bastani

Berlin Buzzwords, Open Stage



## My background

I only make it to the Open Stages :)
Probably because ~~Apache~~ Neo4j

Lead Developer Advocate
Neo4j Graph Database

Thank you all for coming!

# Advocacy + Development

Excited about Neo4j and graph database technology

And helping Neo4j Users to be happy and successful

But I like to code, so I work a lot on Neo4j integrations and performance improvements.

# Your background

Have you used Graph Databases ?
Or Neo4j ?
How about Hadoop ?
Or Spark?
And Spark's GraphX ?

Show of hands

# Agenda

What is this graph thing and why should I care?

Flurry of Intros: Neo4j, Spark, Docker

We're going to run PageRank on all human knowledge.

How ?

Meet Mazerunner, integrating Neo4j & Spark

# The Problem

It's hard to analyze graphs at scale

# The importance of graph algorithms

*PageRank* gives us Search

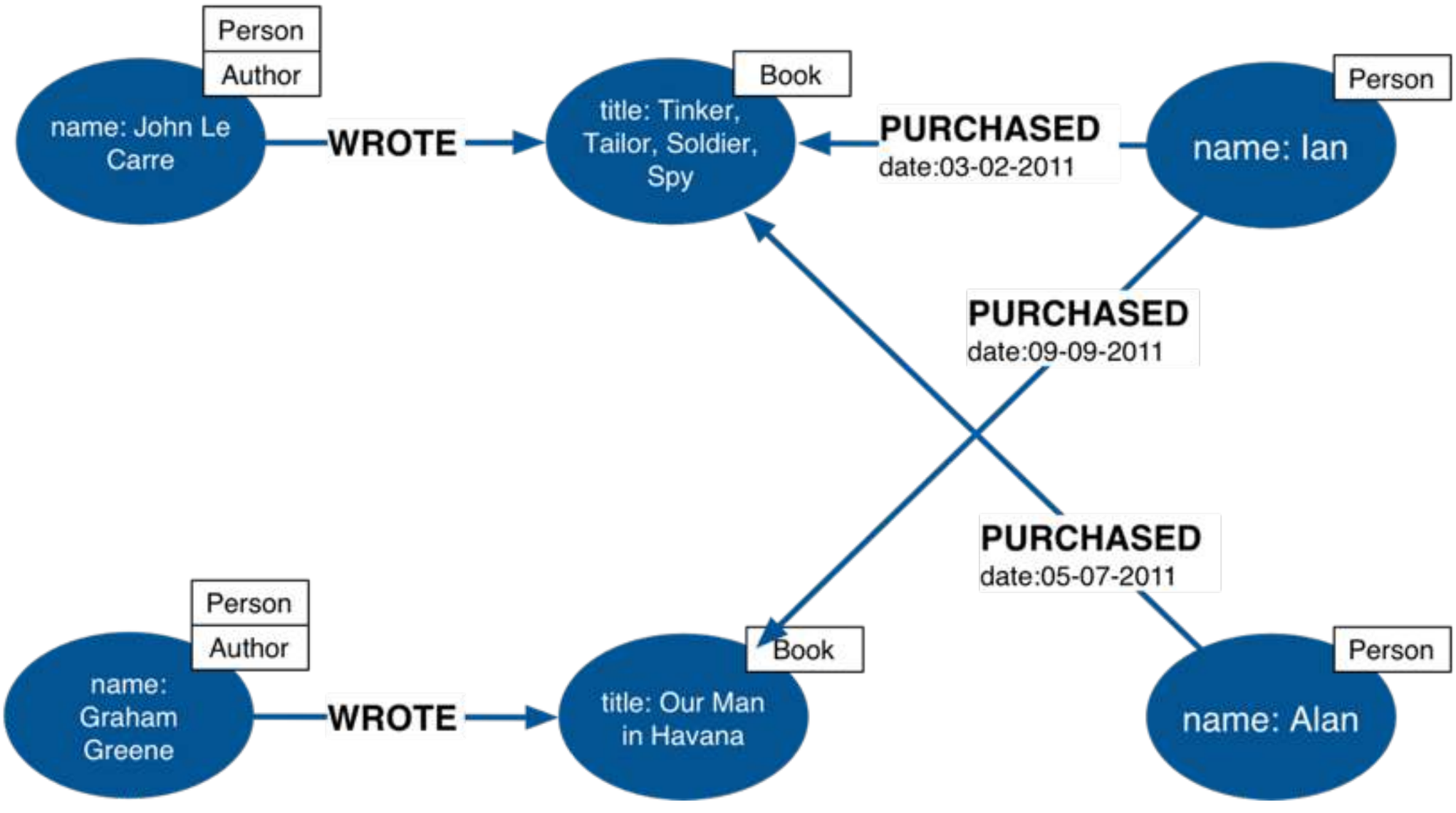
*Friend of a Friend* gives us Social

*Collaborative Filtering* gives us Recommendations

*Betweeness Centrality* gives us Relevancy

*Community Detection* gives us Structure

# What is a Graph?



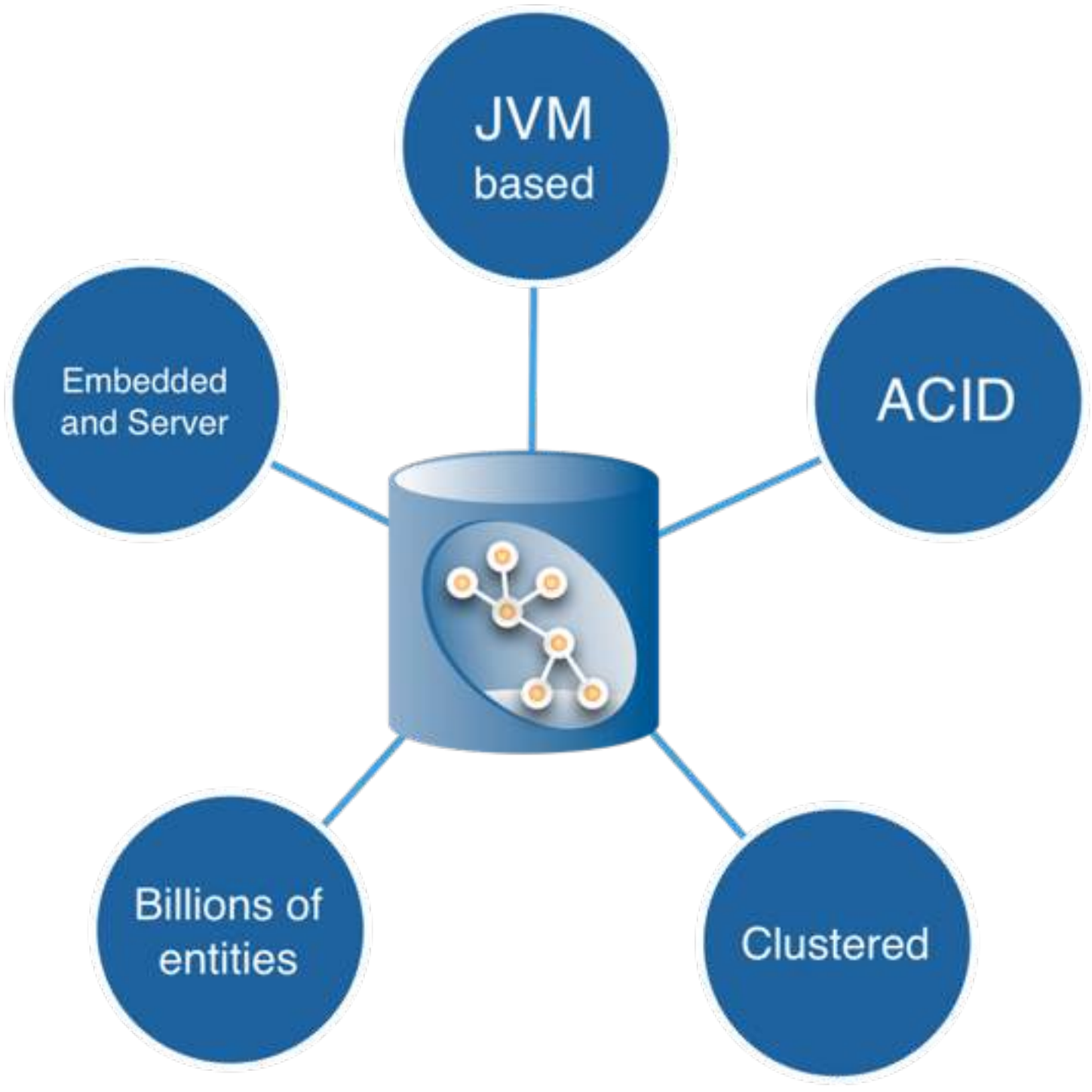

*Nodes* connected by *Relationships* with *Properties*

# Graph Databases - Made for Graphs

Graph Databases were built from the ground up to manage, story and query connected data at scale.

Serving as great **real-time** solution for focused **local search** and exploration.

# What is Neo4j?




*Most widely used, OSS*-Graph Database

Why is it so hard to do this stuff?

# Problem #1:

# Relational Databases - Tabular World

Relational databases store data in ways that make it difficult to extract graphs for analysis

# Problem #2:

# Graphs at Global Scale

Not just many rows (nodes) x 100M

But also many connections (fan out) x 100

And way more paths:

*nodes x rels* *steps* : 100M x $100^{10}$

# Graph Compute Engines - Made for Large Scale Graph Analytics

Graph Compute Engines provide batch or streaming graph computation based on optimized graph algorithm implementations.

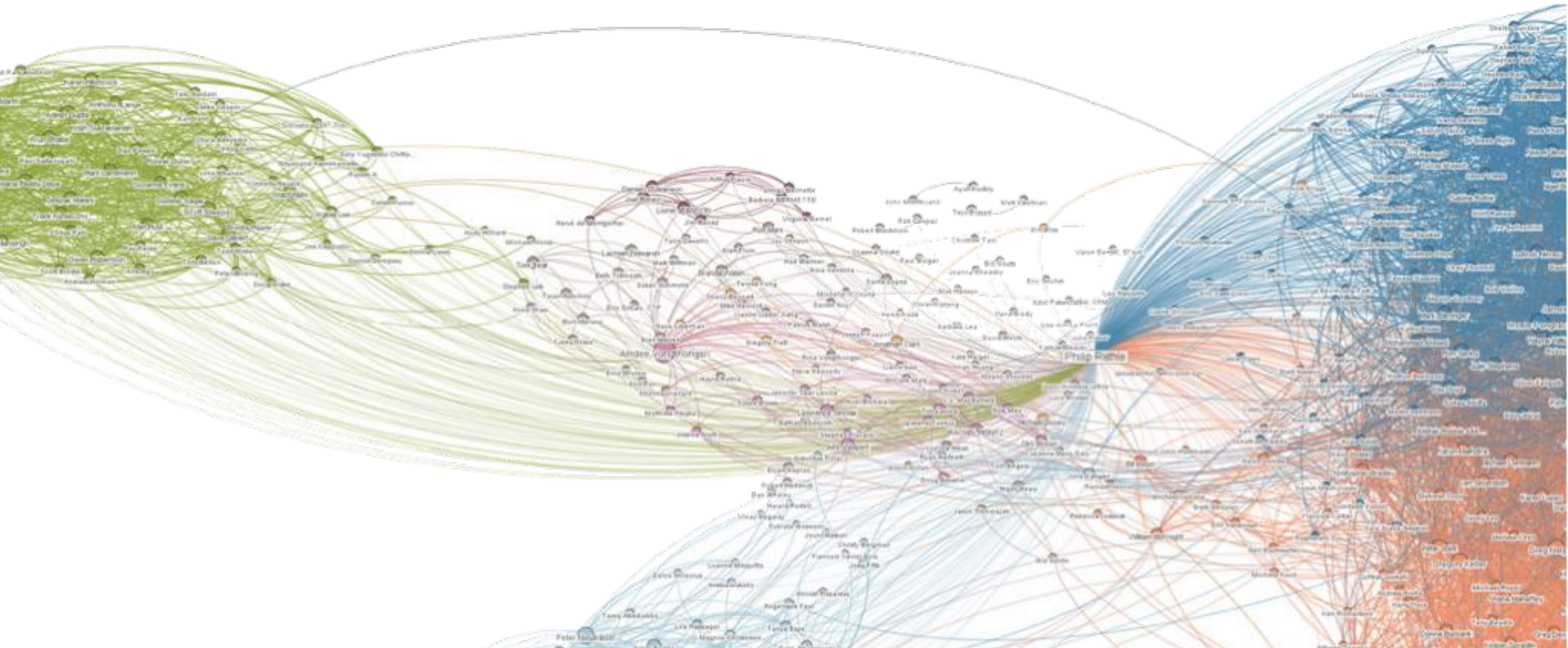
Serving as good **offline** solution for **global processing**. Often implement a scalable message passing algorithm like Pregel.

Examples: Pregel, Giraph, Faunus, Spark's GraphX

# Spectrum of Graph Processing

**Real-Time/ OLTP**

**Offline/ Batch**



# Hadoop MapReduce

There are several Graph Compute Engines on Hadoop.

Hadoop is great for batch processing of **non computationally expensive** iterative algorithms.

Bound by I/O performance of HDFS/HBase and by memory usage.

# Graph algorithms can be evil at scale

It depends on the complexity of your graph

How many strongly connected components you have

But since some graph algorithms like PageRank are iterative

You have to iterate from one stage and use the results of the previous stage

# It doesn't matter how many nodes you have in your cluster

For iterative graph algorithms the complexity of the graph will make you or break you

Graphs with high complexity need a lot of memory to be processed iteratively

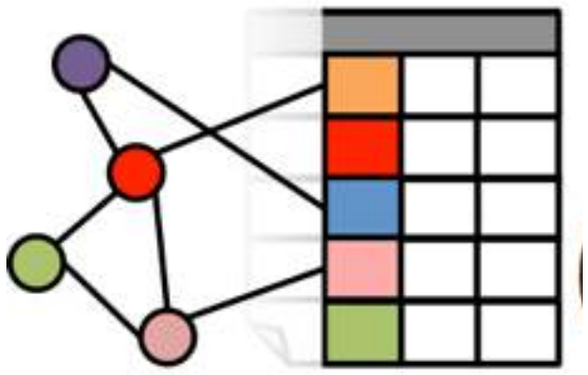
# Meet - Apache Spark

It's a scalable, fault-tolerant **in-memory** big data and machine learning platform.

Spark integrates well with Hadoop but is more modern and way faster in processing.

Also lets you do in-memory analytics on a graph dataset using the optimized GraphX library!

# Spark GraphX

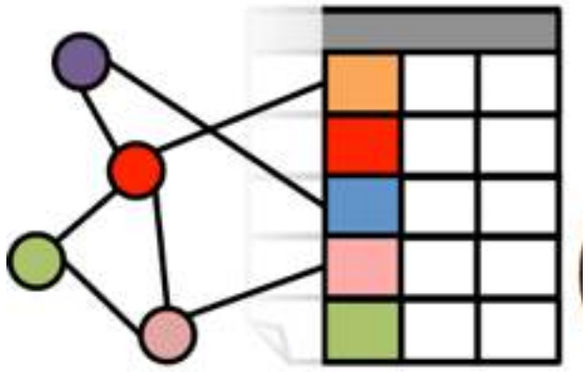


GraphX provides a Property Graph abstraction inside of Spark.

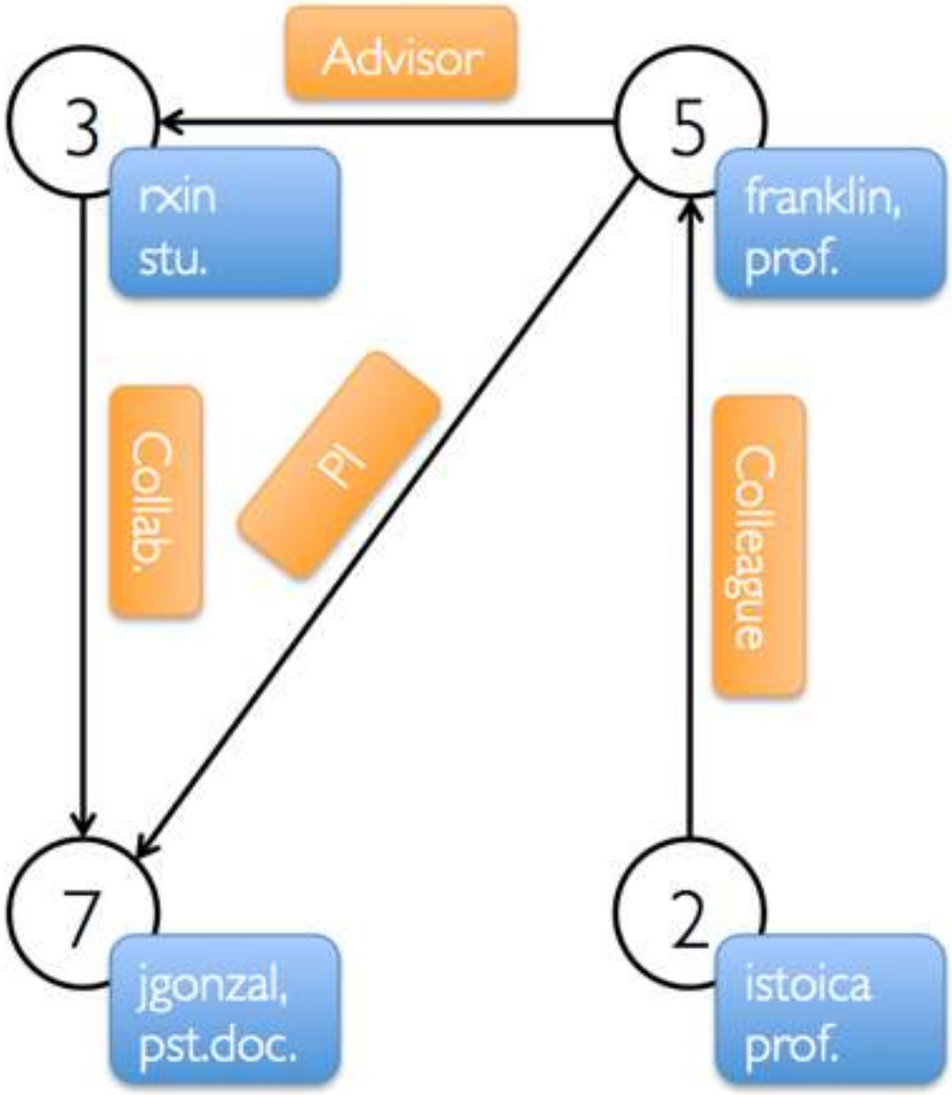
Based on a **node-table** ***(id, props)*** and a **relationship-table** ***(from,to,props)***.

Efficient mapping of those tables in Vertex and Edge abstractions.

# Spark GraphX



## Property Graph



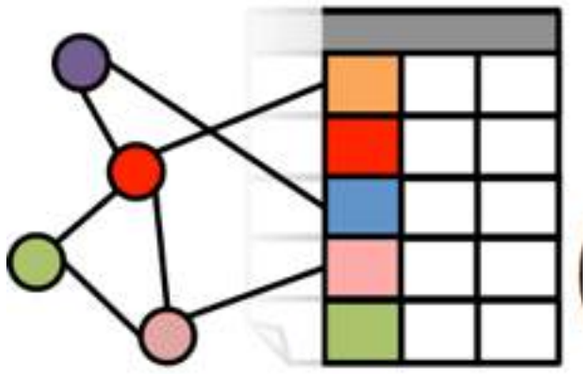

## Vertex Table

| 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 |

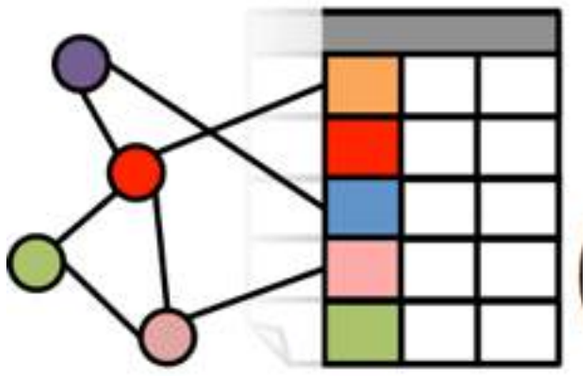
# Spark GraphX



Efficient implementation of Pregel messag
algorithm:

- pre-partition graphs
- push down predicates
- skip unused properties
- only process "active" nodes and edges

# Spark GraphX - Page Rank



```
val graph = GraphLoader.edgeListFile(“hdfs://web.txt”)

val prGraph = graph.joinVertices(graph.outDegrees)

val pageRank = prGraph.pregel(initialMessage = 0.0, iter = 10)

  ((oldV, msgSum) => 0.15 + 0.85 * msgSum,

          triplet => triplet.src.pr / triplet.src.deg,

     (msgA, msgB) => msgA + msgB)

pageRank.vertices.top(20) (Ordering.by(_._2)).foreach(println)
```

Open Source Project: *Mazerunner*

Graph Analytics with 2-way ETL

(Neo4j) <-[:WORKS_WITH]-> (Spark-GraphX)

# The basic idea is...

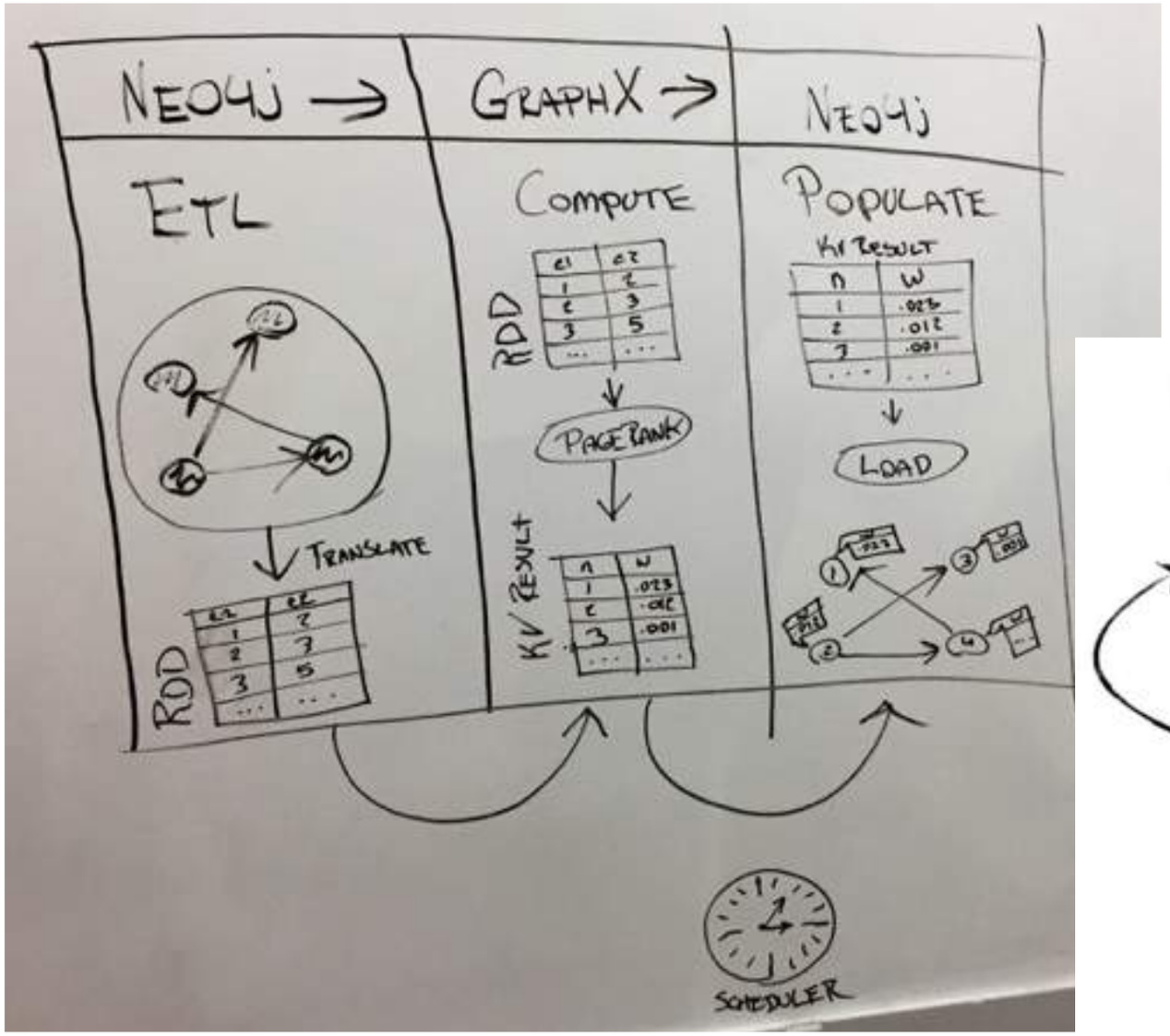
Graph databases plus ETL,

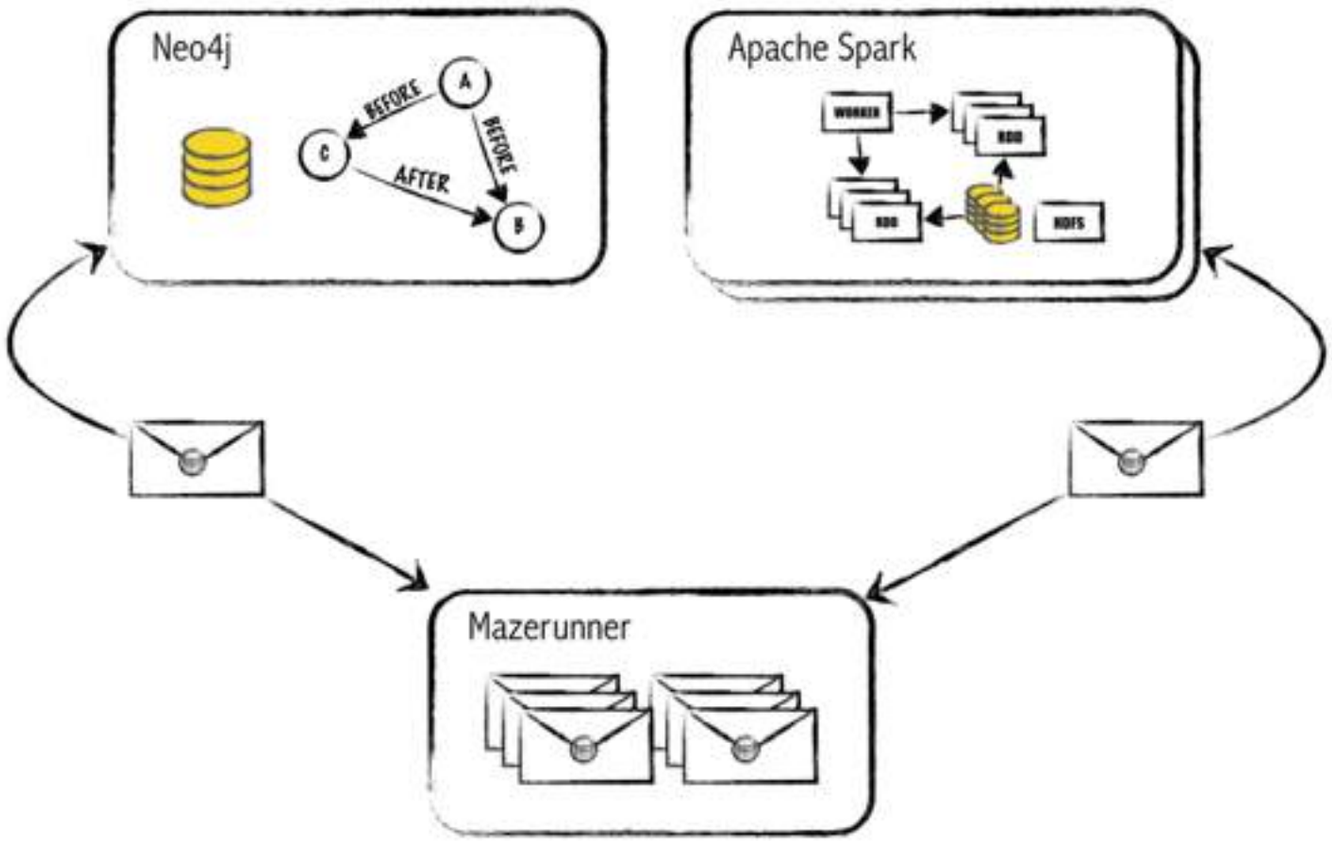
so you can **analyze** your data,

**write it back**, and

use it later in your realtime **graph queries**.

# What is Neo4j Mazerunner?



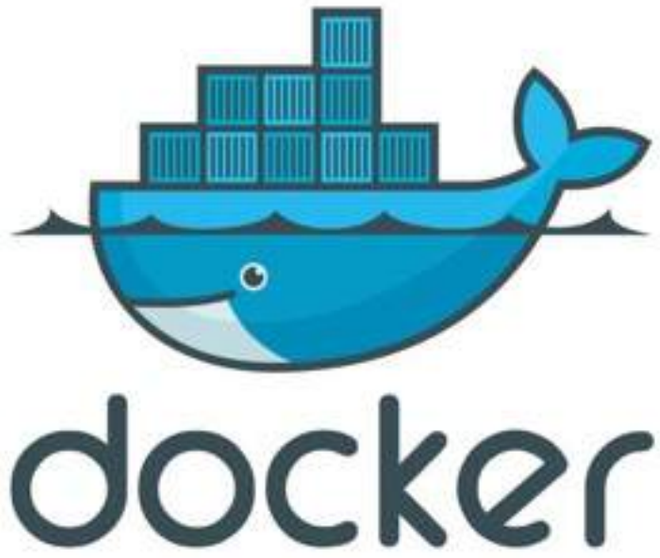

http://www.kennybastani.com/2014/11/using-apache-spark-and-neo4j-for-big.html

# What is Neo4j Mazerunner?

- Neo4j
- RabitMQ
- HDFS
- Spark / GraphX
- Docker

http://www.kennybastani.com/2014/11/using-apache-spark-and-neo4j-for-big.html

# Docker




Docker is a lightweight virtualization container engine that lets you easily package applications and deploy them with ease.

# Mazerunner runs on Docker Compose

You can pull it down and deploy it safely.

Just for your analytics and

then throw away the docker containers

while leaving the processed data behind.

# Now let's fire up Neo4j Mazerunner

Demo Goals:

- How to install and run Mazerunner using Docker Compose

- Run analysis job scheduler that extracts subgraphs, analyzes them, and pops the results back to Neo4j

# Demo: Part 0 - Docker Compose

```
# Start Spark-Neo4j using Docker Compose

docker run  --env DOCKER_HOST=$DOCKER_HOST \
            --env DOCKER_TLS_VERIFY=$DOCKER_TLS_VERIFY \
            --env DOCKER_CERT_PATH=/docker/cert \
            -v $DOCKER_CERT_PATH:/docker/cert \
            -ti kbastani/spark-neo4j up -d
```

http://www.kennybastani.com/2015/03/spark-neo4j-tutorial-docker.html

# Demo: Part 1 - DBPedia in Neo4j

```
1 // pages by degree
2 match (p:Page) return p, size((p)--()) as c order by c desc limit 25
```

```
$ match (p:Page) return p, size((p)--()) as c order by c desc limit 25
```

Graph | Rows

| | |
|---|---|
| title United States | 394690 |
| title Animal | 174353 |
| title France | 139765 |

Returned 25 rows in 15212 ms.

```
$ match path = (p:Page {title:"Berlin"})--(o) return path,size((o)--()) as c order by c desc ...
```

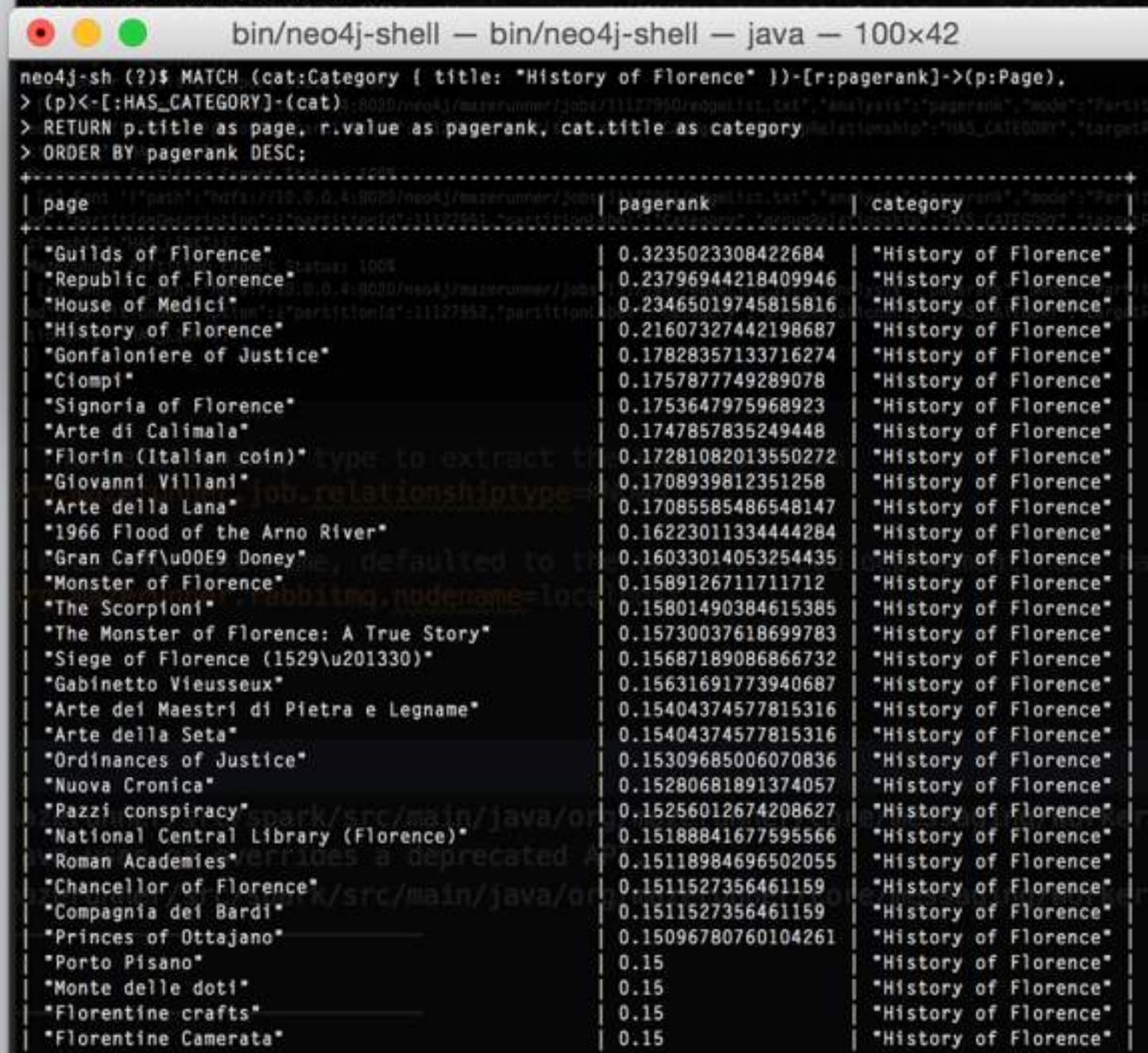
Graph | Rows

*(20) Berlin(1) Page(20)

*(25) Link(25)

# Demo: Part 2 - PageRank - Mazerunner






| page | pagerank | category |
|---|---|---|
| "Guilds of Florence" | 0.3235023308422684 | "History of Florence" |
| "Republic of Florence" | 0.23796944218409946 | "History of Florence" |
| "House of Medici" | 0.23465019745815816 | "History of Florence" |
| "History of Florence" | 0.21607327442198687 | "History of Florence" |
| "Gonfaloniere of Justice" | 0.17828357133716274 | "History of Florence" |
| "Ciompi" | 0.1757877749289078 | "History of Florence" |
| "Signoria of Florence" | 0.1753647975968923 | "History of Florence" |
| "Arte di Calimala" | 0.1747857835249448 | "History of Florence" |
| "Florin (Italian coin)" | 0.17281082013550272 | "History of Florence" |
| "Giovanni Villani" | 0.1708939812351258 | "History of Florence" |
| "Arte della Lana" | 0.17085585486548147 | "History of Florence" |
| "1966 Flood of the Arno River" | 0.16223011334444284 | "History of Florence" |
| "Gran Caff\u00E9 Doney" | 0.16033014053254435 | "History of Florence" |
| "Monster of Florence" | 0.1589126711711712 | "History of Florence" |
| "The Scorpioni" | 0.15801490384615385 | "History of Florence" |
| "The Monster of Florence: A True Story" | 0.15730037618699783 | "History of Florence" |
| "Siege of Florence (1529\u201330)" | 0.15687189086866732 | "History of Florence" |
| "Gabinetto Vieusseux" | 0.15631691773940687 | "History of Florence" |
| "Arte dei Maestri di Pietra e Legname" | 0.15404374577815316 | "History of Florence" |
| "Arte della Seta" | 0.15404374577815316 | "History of Florence" |
| "Ordinances of Justice" | 0.15309685006070836 | "History of Florence" |
| "Nuova Cronica" | 0.15280681891374057 | "History of Florence" |
| "Pazzi conspiracy" | 0.15256012674208627 | "History of Florence" |
| "National Central Library (Florence)" | 0.15188841677595566 | "History of Florence" |
| "Roman Academies" | 0.15118984696502055 | "History of Florence" |
| "Chancellor of Florence" | 0.1511527356461159 | "History of Florence" |
| "Compagnia dei Bardi" | 0.1511527356461159 | "History of Florence" |
| "Princes of Ottajano" | 0.15096780760104261 | "History of Florence" |
| "Porto Pisano" | 0.15 | "History of Florence" |
| "Monte delle doti" | 0.15 | "History of Florence" |
| "Florentine crafts" | 0.15 | "History of Florence" |
| "Florentine Camerata" | 0.15 | "History of Florence" |

# What's next?

Try it out.

Provide feedback!

Suggest improvements.

Become a committer to the project and let's make it better

GitHub Project — github.com/kbastani/neo4j-mazerunner
Kennys blog — www.kennybastani.com

# PageRank as native Neo4j Extension

Implement a Neo4j Server extension on Neo4j 2.3

Using the Kernel API

And parallelization

**Runs pagerank on dbpedia in 90s**

Thanks!

Questions ?

Find us online:

neo4j.com | @neo4j