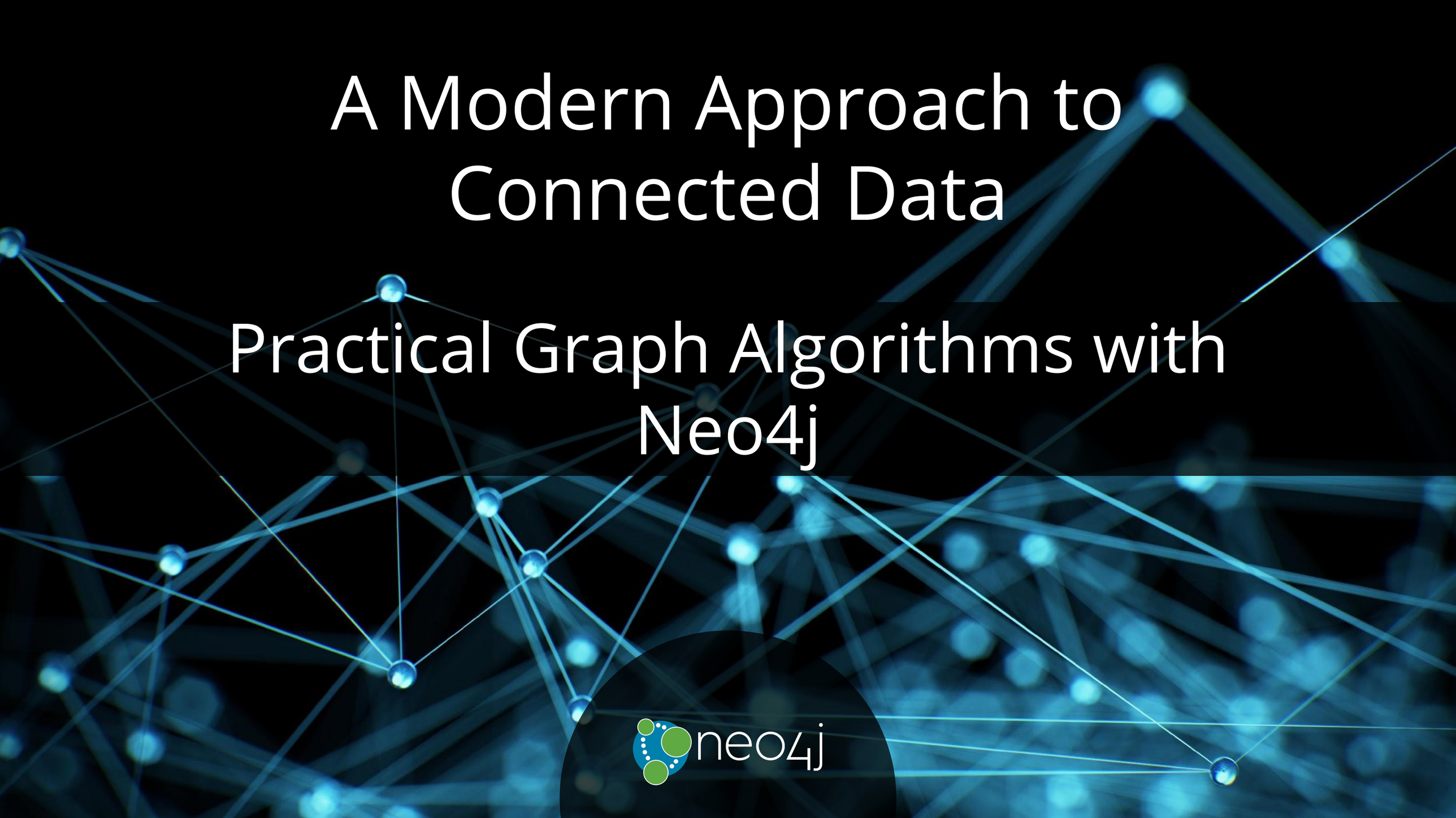
A Modern Approach to Connected Data

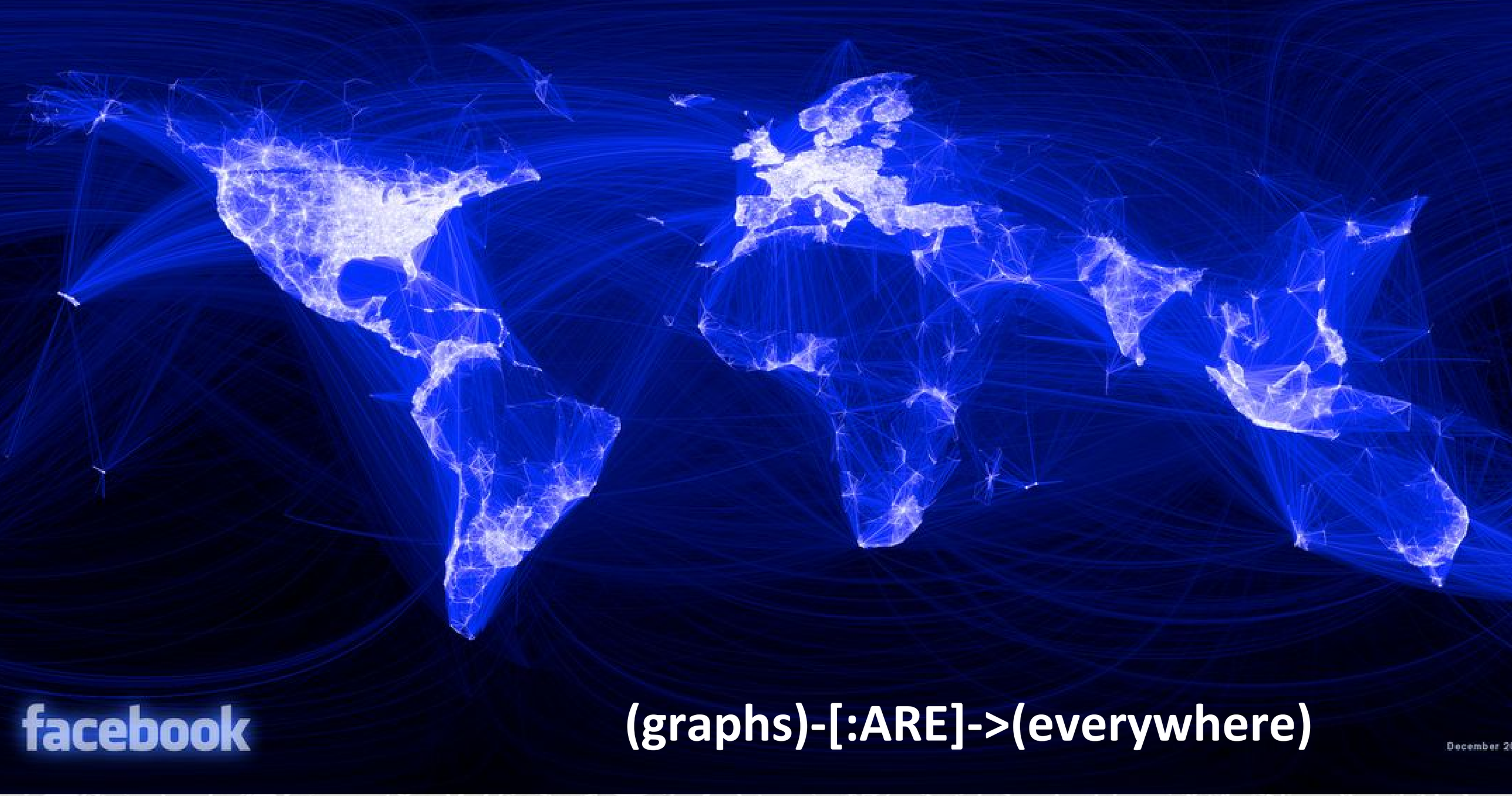
Practical Graph Algorithms with Neo4j



About me

Head of Developer Relations Engineering

Follow & tweet at me @mesirii


facebook
(graphs)-[:ARE]->(everywhere)

Value from Data Relationships

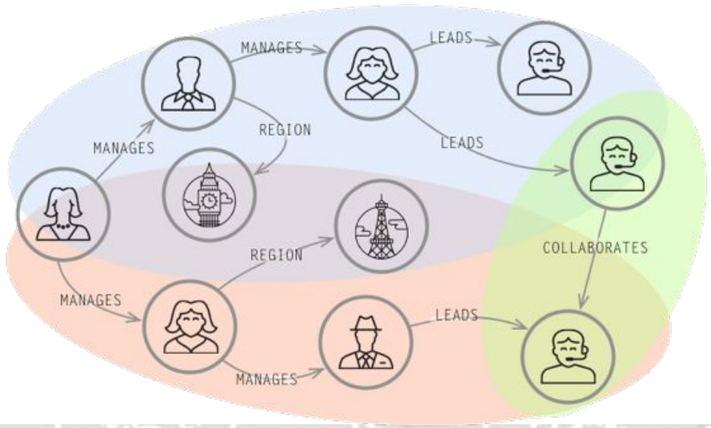
Common Current Use Cases

Internal Applications

Master Data Management

Network and
IT Operations

Fraud Detection

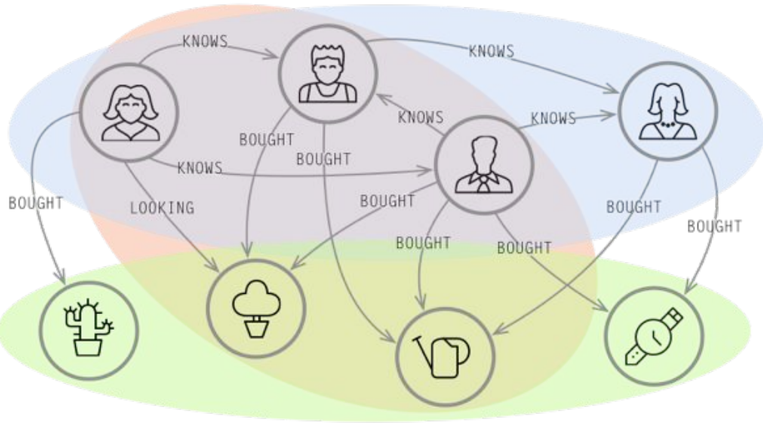


Customer-Facing Applications

Real-Time Recommendations

Graph-Based Search

Identity and
Access Management



Insights from Algorithms

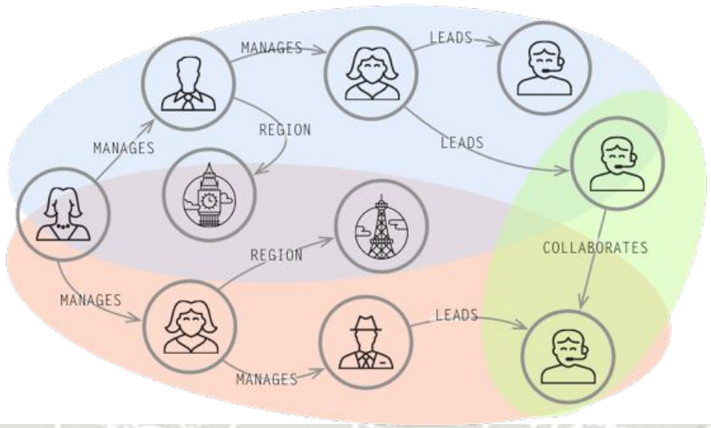
Improving all use-cases

Graph Algorithms

Relevance

Clustering

Structural Insights



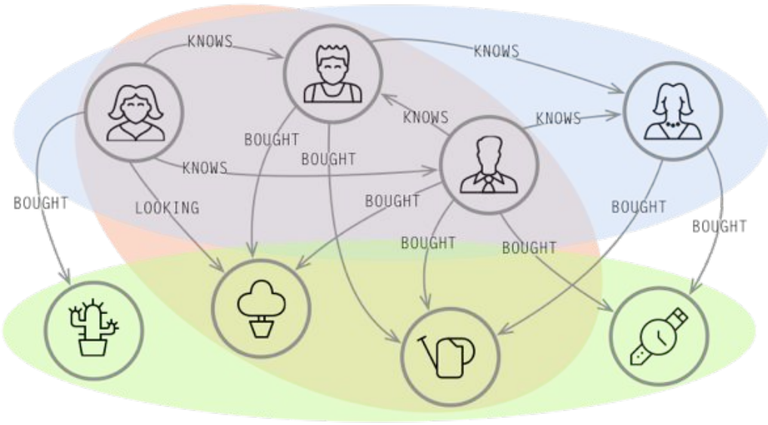
Machine Learning

Classification, Regression

NLP, Struct/Content Pred

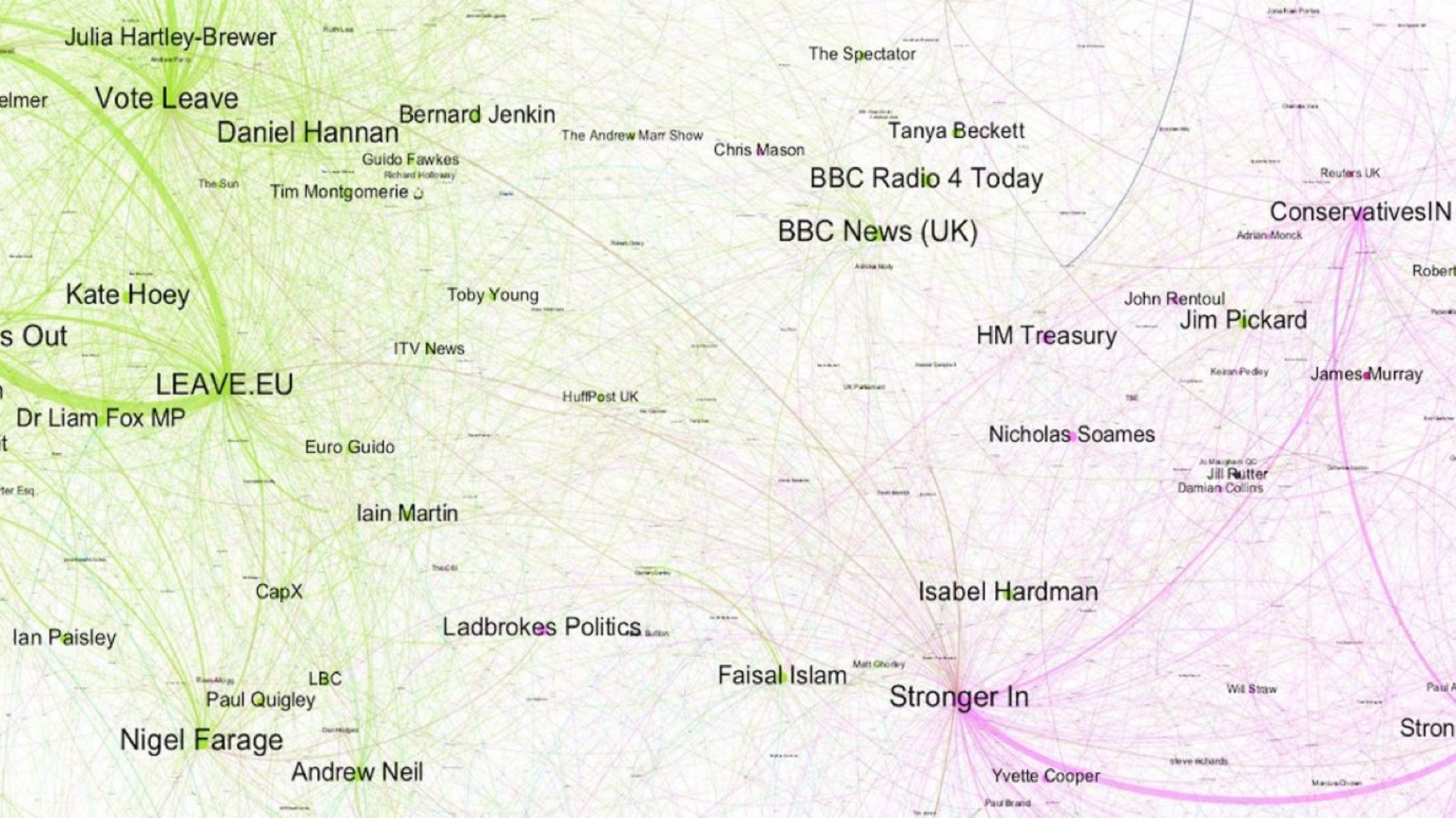
NN <-> Graph

Graph As Compute

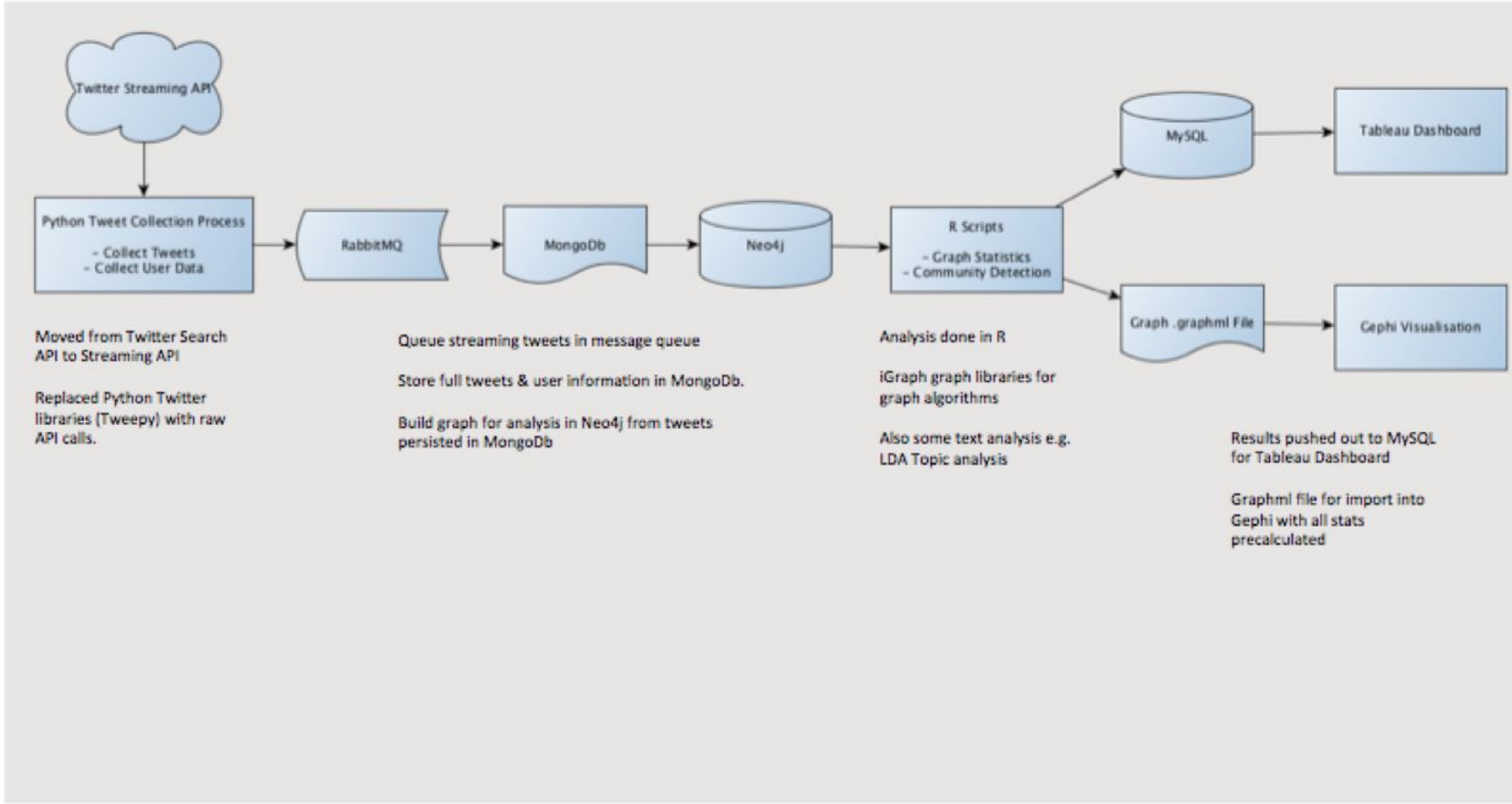


Example: Twitter Analytics Right Relevance

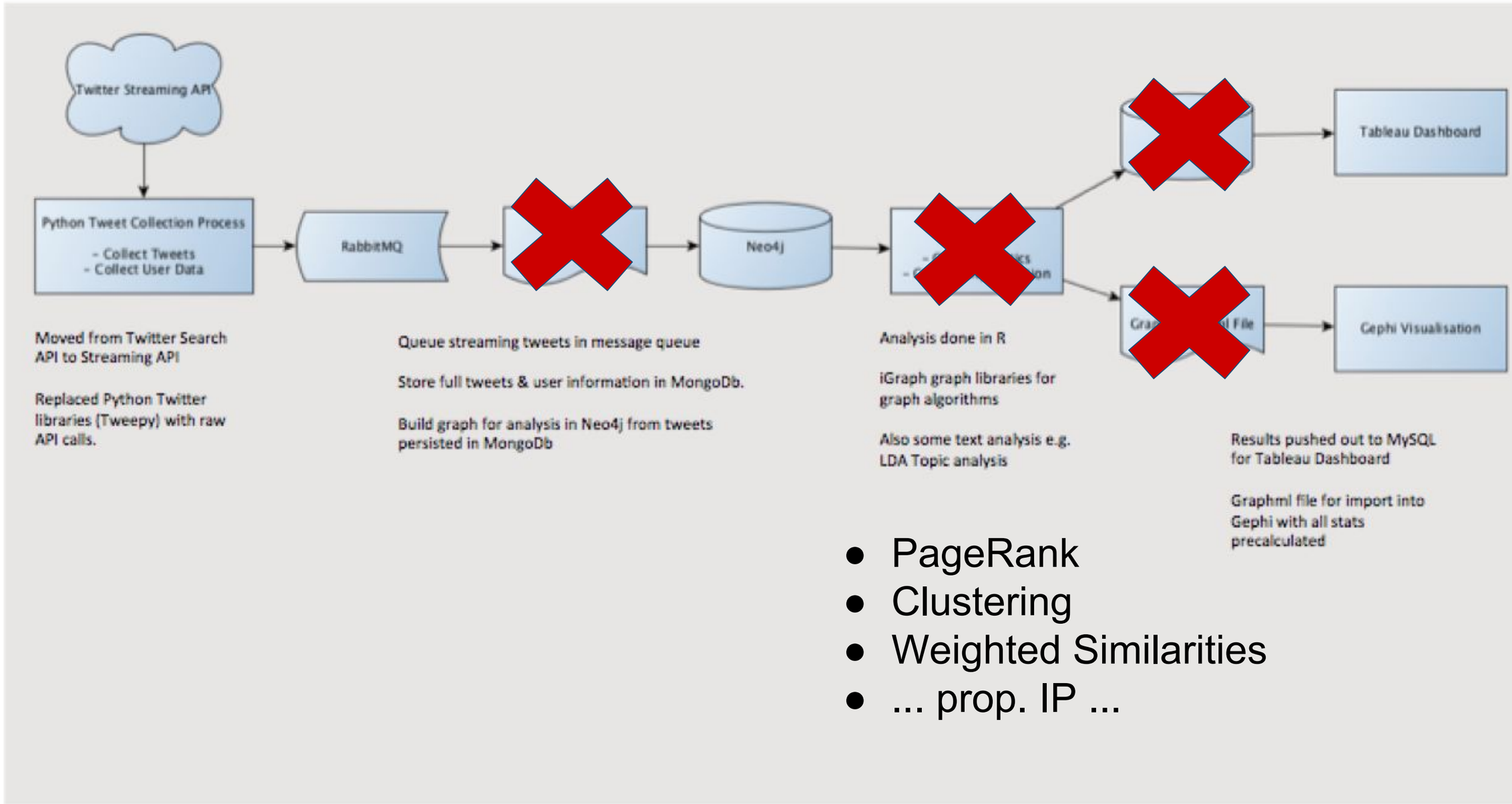
neo4j.com/blog/graph-algorithms-make-election-data-great-again


Julia Hartley-Brewer
The Spectator
Vote Leave
Bernard Jenkin
Daniel Hannan
The Andrew Marr Show
Tanya Beckett
Chris Mason
Guido Fawkes
BBC Radio 4 Today
The Sun
Tim Montgomerie
Reuters UK
ConservativesIN
BBC News (UK)
Kate Hoey
Toby Young
John Rentoul
Jim Pickard
HM Treasury
ITV News
LEAVE.EU
James Murray
HuffPost UK
Dr Liam Fox MP
Nicholas Soames
Euro Guido
Jill Rutter
Damian Collins
Iain Martin
CapX
Isabel Hardman
Ladbrokes Politics
Ian Paisley
Faisal Islam
LBC
Stronger In
Paul Quigley
Nigel Farage
Andrew Neil
Yvette Cooper

Custom Pipeline Neo4j <-> R



Goal: Simplification & Performance



- PageRank
- Clustering
- Weighted Similarities
- ... prop. IP ...

Applicable Tools

- Store Tweets in Neo4j (e.g. 300M)
- Use APOC + **Graph Algorithms** for processing
- Use NLP Algorithms / Procedures (e.g. from GraphAware)
- Neo4j Tableau (WDC) Connector
- Use APOC for streaming results to Gephi

Graph Algorithms

Project Goals

- high performance graph algorithms
- user friendly (procedures)
- support graph projections
- augment OLTP with OLAP
- integrate efficiently with live Neo4j database
  (read, write, Cypher)
- common Graph API to write your own

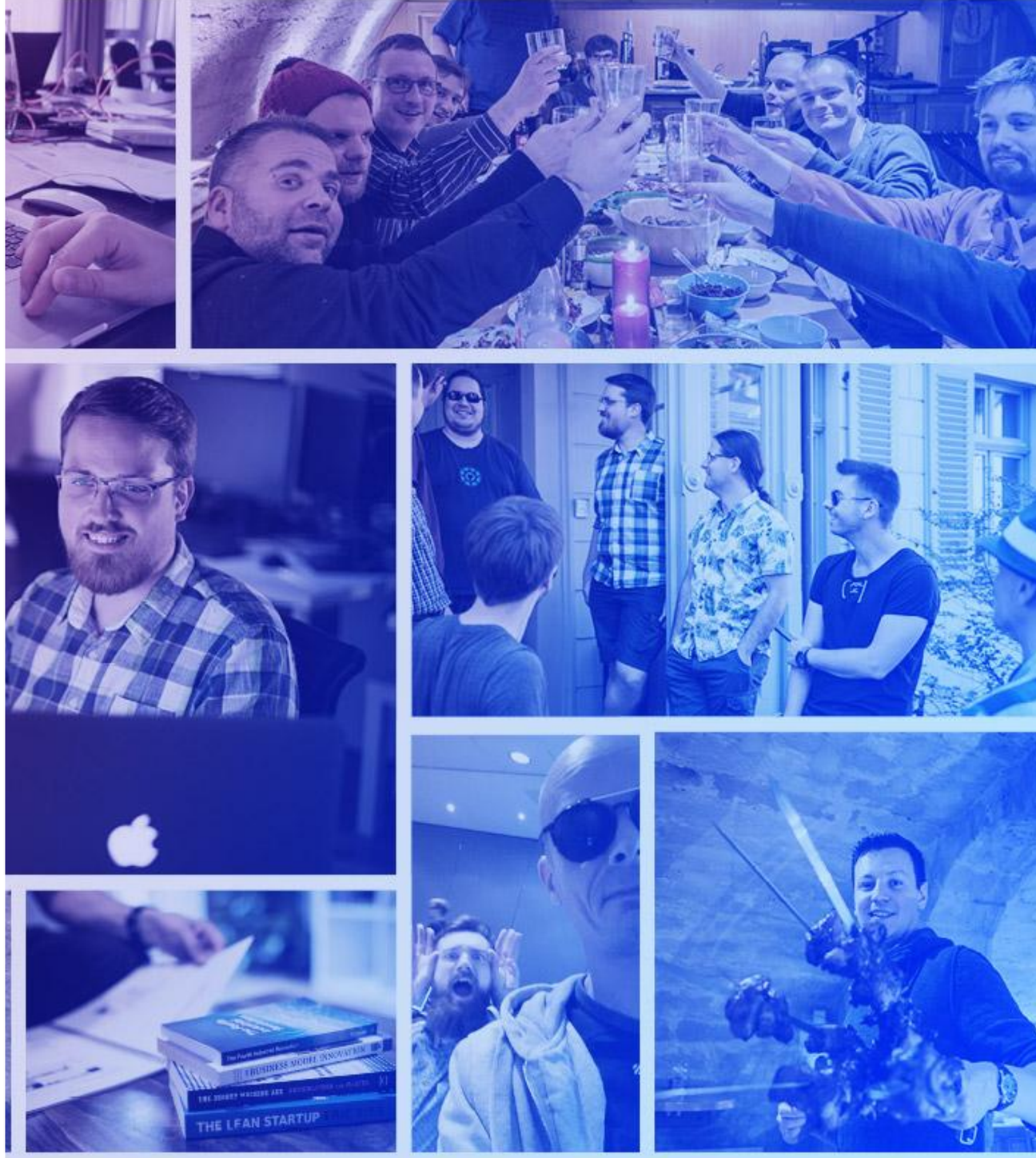
Kudos

Development

- Martin Knobloch (AVGL)
- Paul Horn (AVGL)

Documentation

- Tomaz Bratanic



Usage

1. Call as Cypher procedure
2. Pass in specification (Label, Prop, Query) and configuration
3. ~.stream variant returns (**a lot**) of results

```
CALL algo.<name>.stream('Label','TYPE',{conf})
YIELD nodeId, score
```

4. non-stream variant writes results to graph returns statistics

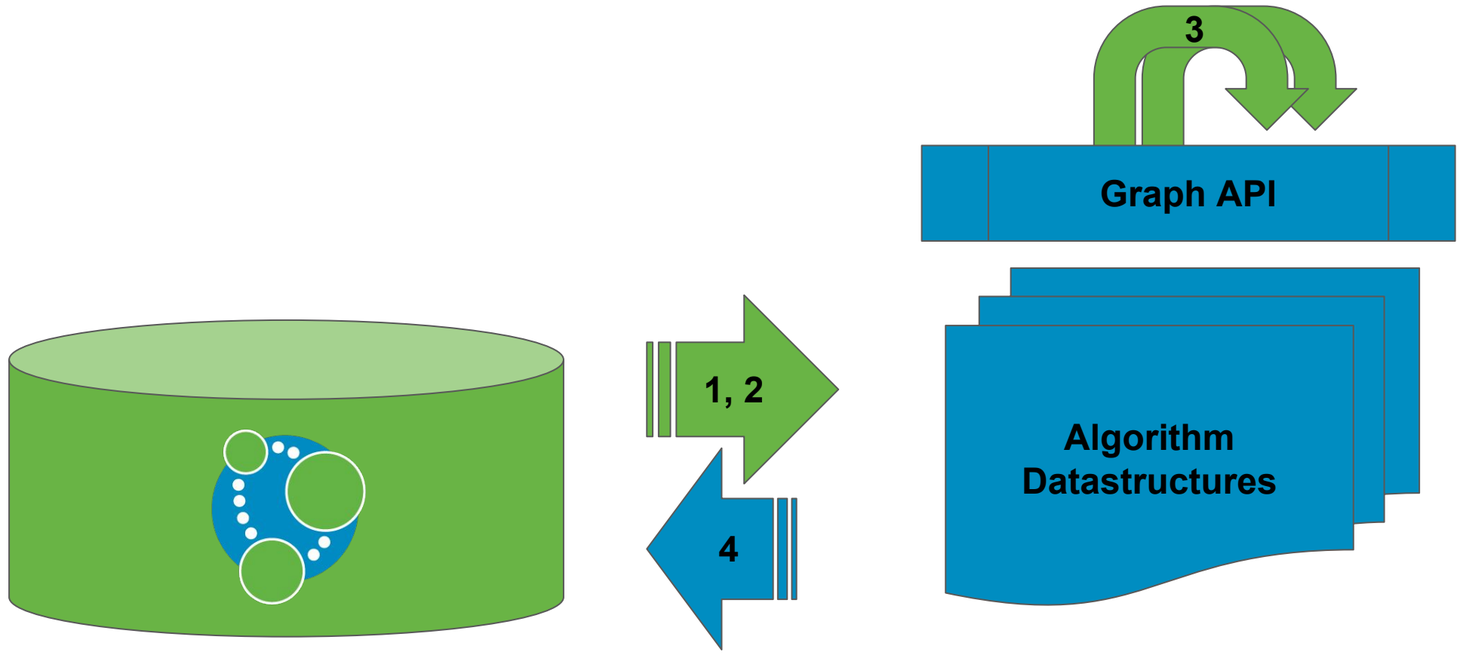
```
CALL algo.<name>('Label','TYPE',{conf});
```

5. Cypher projection:
   pass in Cypher for node- and relationship-lists

```
CALL algo.<name>(
'MATCH ... RETURN id(n)',
'MATCH (n)-->(m) RETURN id(n), id(m)',
{graph:'cypher'})
```

Architecture

1. Load Data in parallel from Neo4j
2. Store in efficient Datastructure
3. Run Graph Algorithm in parallel using Graph API
4. Write data back in parallel



The Algorithms

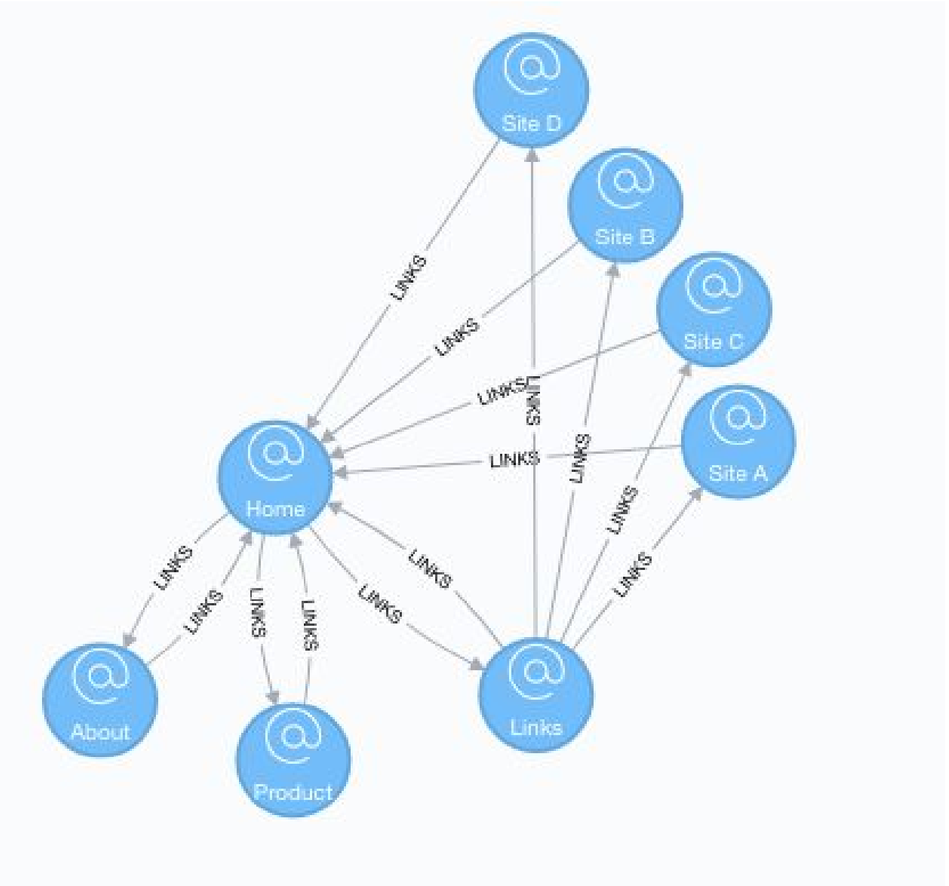
Centralities

- PageRank (baseline)
- Betweeness
- Closeness
- Degree

Example: PageRank

```
CALL algo.pageRank.stream
('Page', 'LINKS', {iterations:20, dampingFactor:0.85})
YIELD node, score
RETURN node,score order by score desc limit 20
```

```
CALL algo.pageRank('Page', 'LINKS',
{iterations:20, dampingFactor:0.85,
write: true,writeProperty:"pagerank"})

YIELD nodes, iterations, loadMillis, computeMillis,
writeMillis, dampingFactor, write, writeProperty
```



https://neo4j-contrib.github.io/neo4j-graph-algorithms/#_page_rank

The Algorithms

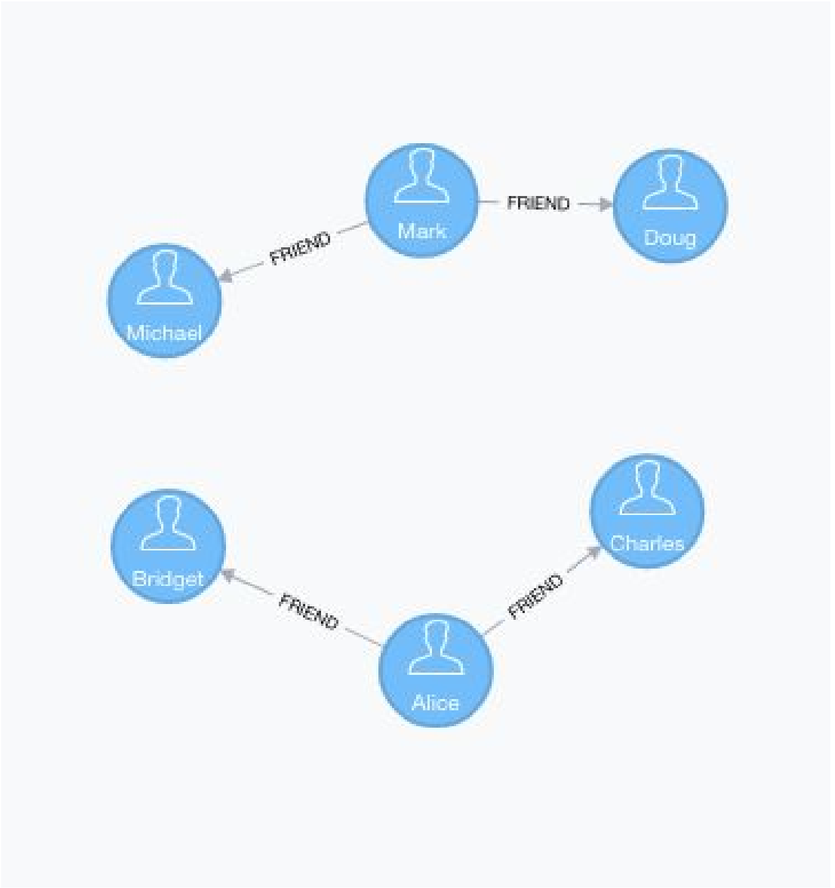
Clustering

- Label Propagation
- Louvain (Phase2)
- Union Find / WCC
- Strongly Connected Components
- Triangle-Count/Clustering Coeff

Example: UnionFind (CC)

```
CALL algo.unionFind.stream('User', 'FRIEND', {})
YIELD nodeId,setId
RETURN setId, count(*)
ORDER BY count(*) DESC LIMIT 100;
```

```
CALL algo.unionFind('User', 'FRIEND',
{write:true, partitionProperty:'partition'})
YIELD nodes, setCount, loadMillis, computeMillis,
      writeMillis
```



https://neo4j-contrib.github.io/neo4j-graph-algorithms/#_connected_components

The Algorithms

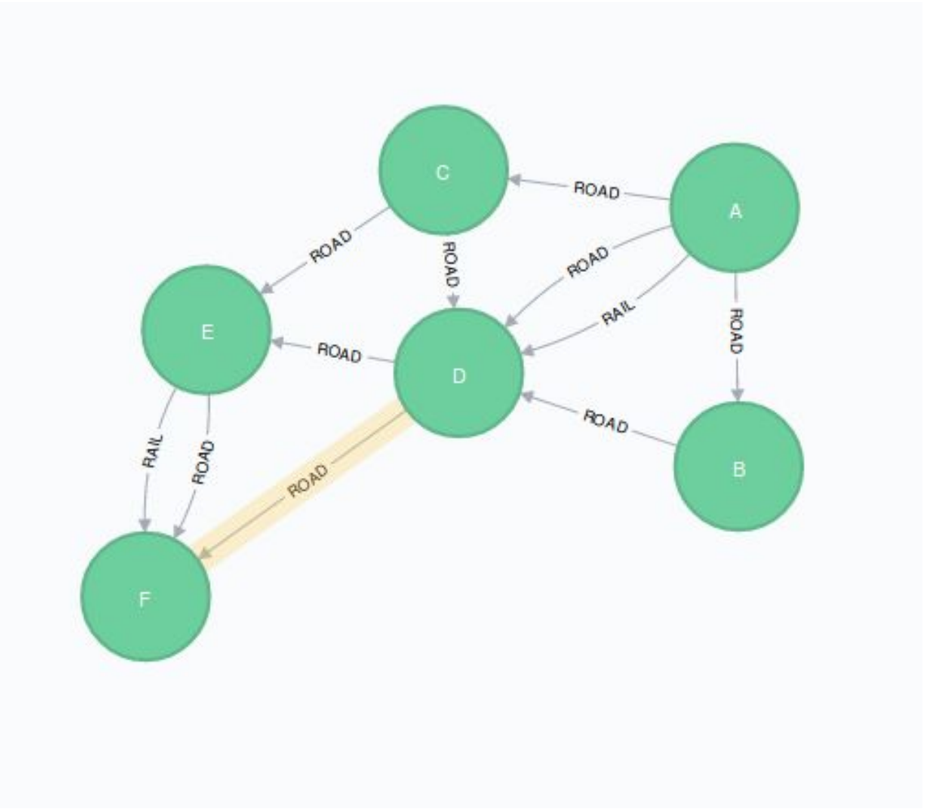
Path-Expansion / Traversal

- Single Short Path
- All-Nodes SSP
- Parallel BFS / DFS

Example: UnionFind (CC)

```
MATCH(start:Loc {name:'A'})
CALL algo.deltaStepping.stream(start, 'cost', 3.0)
YIELD nodeId, distance
RETURN nodeId, distance ORDER LIMIT 20
```

```
MATCH(start:Node {name:'A'})
CALL algo.deltaStepping(start, 'cost', 3.0,
{write:true, writeProperty:'ssp'})
YIELD nodeCount, loadDuration, evalDuration,
      writeDuration
RETURN *
```



https://neo4j-contrib.github.io/neo4j-graph-algorithms/#_all_pairs_and_single_source_shortest_path

DEMO

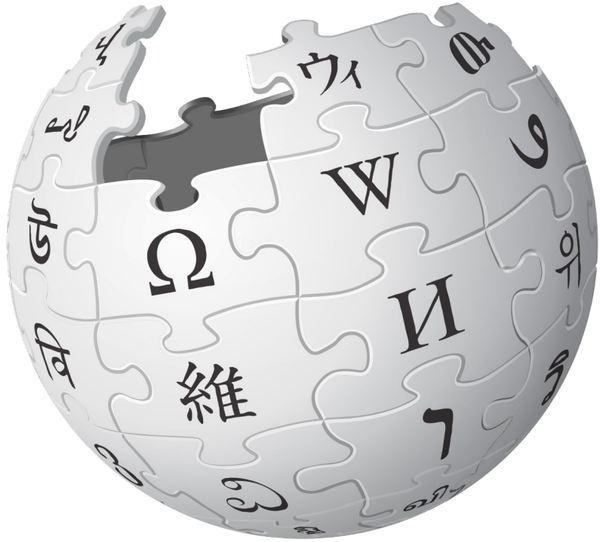
Datasets

- Neo4j Community Graph
  - 280k nodes, 1.4m relationships
  - Centralities, Clustering, Grouping
- DBPedia
  - 11m nodes, 116m relationships
  - Page Rank, unionFind
- Bitcoin
  - 1.7bn nodes, 2.7bn rels
  - degree distribution
  - pageRank, unionFind

Neo4j Community Graph

- Neo4j Community activity from Twitter, GitHub, StackOverflow
- Let's look at tweets
- Tweet-PageRank
- Projection -> mention-user-network
  - centralities
  - clustering
  - grouping

DBPedia



- Shallow Copy of Wikipedia
- Just (Page) -[:Link]-> (Page)

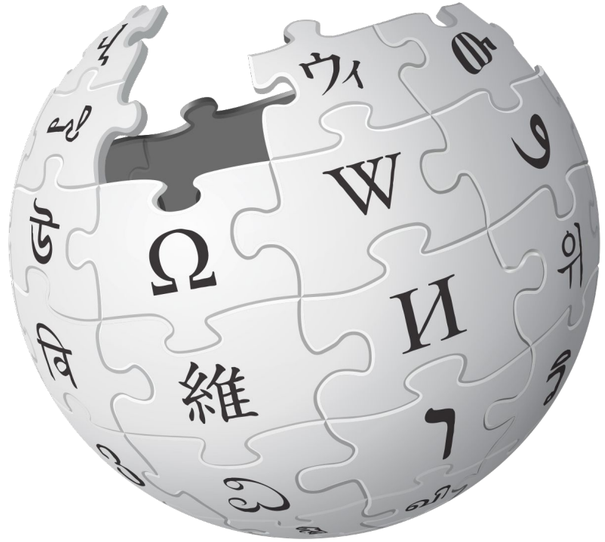
```
CALL algo.pageRank.stream('Page', 'Link', {iterations:5}) YIELD node, score
WITH * ORDER BY score DESC LIMIT 5
RETURN node.title, score;
```

```
+-------------------------------------+
| node.title                | score   |
+-------------------------------------+
| "United States"           | 13349.2 |
| "Animal"                  | 6077.77 |
| "France"                  | 5025.61 |
| "List of sovereign states" | 4913.92 |
| "Germany"                 | 4662.32 |
+-------------------------------------+
5 rows
46247 ms
```

DBPedia

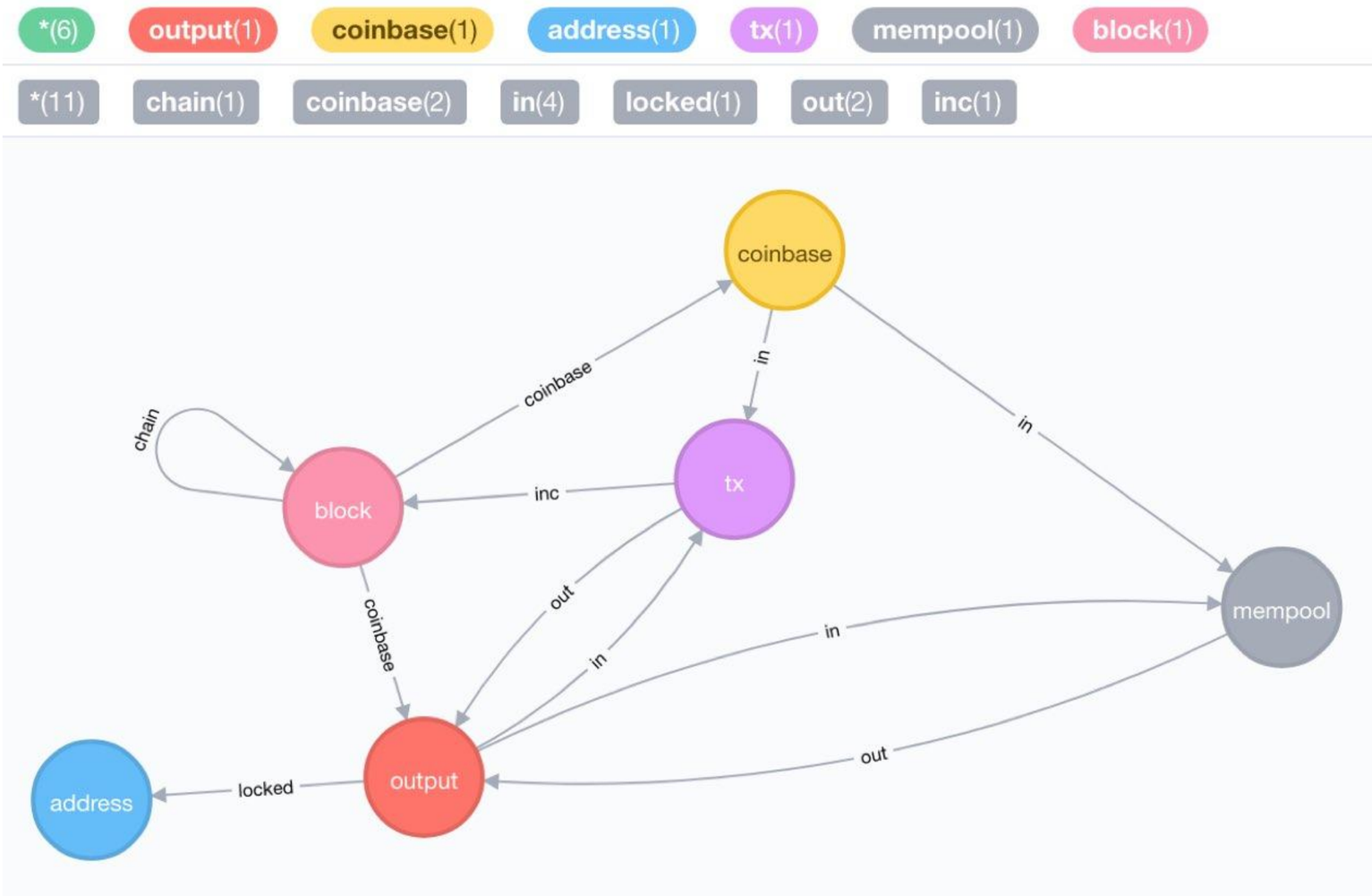
```
CALL algo.pageRank('Page', 'Link', {write:true,iterations:20});
```

```
+-------------------------------------------------------------------------------------+
| nodes    | iter | loadMillis | computeMillis | writeMillis | damping | writeProperty |
+-------------------------------------------------------------------------------------+
| 11474730 | 20   | 34106      | 9712          | 1810        | 0.85    | "pagerank"    |
+-------------------------------------------------------------------------------------+
1 row
47888 ms
```



BitCoin

- Full Copy of the BitCoin BlockChain
  - from learnmeabitcoin.com
  - thanks Greg Walker
  - (see Online Meetup)
- 1.7bn nodes, 2.7bn rels
  - 474k blocks,
  - 240m tx,
  - 280m addresses
  - 650m outputs



BitCoin

- distribution of "locked" relationships for "addresses"
  - = participation in transactions

```
call apoc.stats.degrees('<locked');
+--------------------------------------------------------------------------------------------------------------------------------------+
| type      | direction | total     | p50 | p75 | p90 | p95 | p99 | p999 | max     | min | mean                    |
+--------------------------------------------------------------------------------------------------------------------------------------+
| "locked" | "INCOMING" | 654662356 | 0   | 0   | 1   | 1   | 2   | 28   | 1891327 | 0   | 0.37588608290716047 |
+--------------------------------------------------------------------------------------------------------------------------------------+
1 row
308619 ms
```

BitCoin

- Inferred network of addresses, via transaction and output
- (a)<-[:locked]-(o)-[:in]->(tx)-[:out]->(o2)-[:locked]->(a2)
- use cypher projections
- 1M outputs (24s) to start with, connected addresses via tx
- 10M outputs (296s)

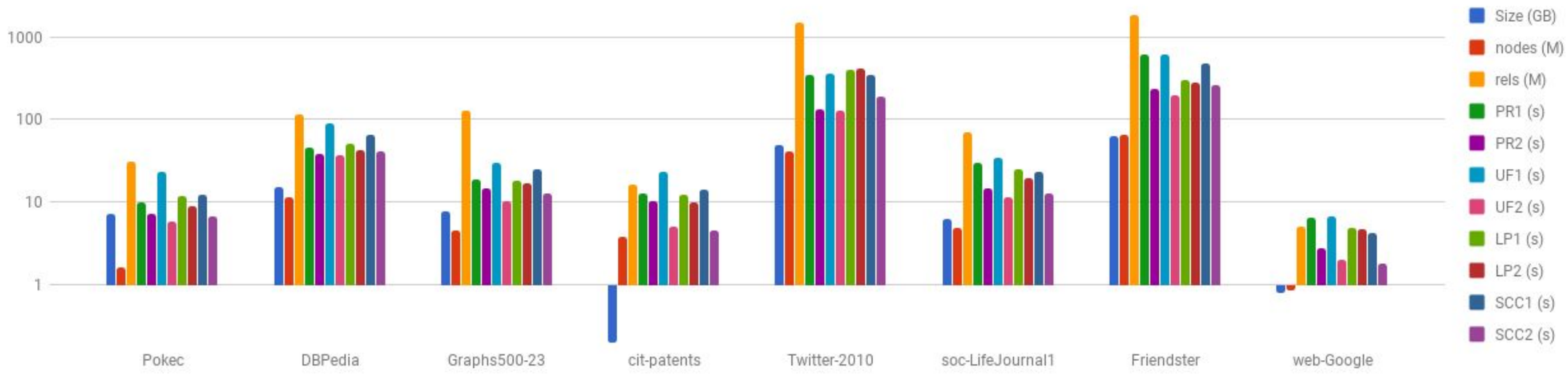
```
call algo.unionFind.stream(
'match (o:output)-[:locked]->(a) with a limit 10000000
 return id(a) as id',

'match (o:output)-[:locked]->(a) with o,a limit 10000000
 match (o)-[:in]->(tx)-[:out]->(o2)-[:locked]->(a2)
 return id(a) as source, id(a2) as target, count(tx) as value',

{graph:'cypher'}) YIELD setId, nodeId
RETURn setId, count(*) as size
ORDER BY size DESC LIMIT 10;
```

```
+------------------+
| setId   | size   |
+------------------+
| 5036    | 4409420 |
| 6295282 | 1999    |
| 5839746 | 1488    |
| 9356302 | 833     |
| 6560901 | 733     |
| 6370777 | 637     |
| 8101710 | 392     |
| 5945867 | 369     |
| 2489036 | 264     |
| 1703620 | 203     |
+------------------+
10 rows
296109 ms
```

Timing

First release July 2017

- neo4j.com/blog/efficient-graph-algorithms-neo4j



Second Release Sept/Oct 2017

- huge graphs, additional algorithms, bugfixes

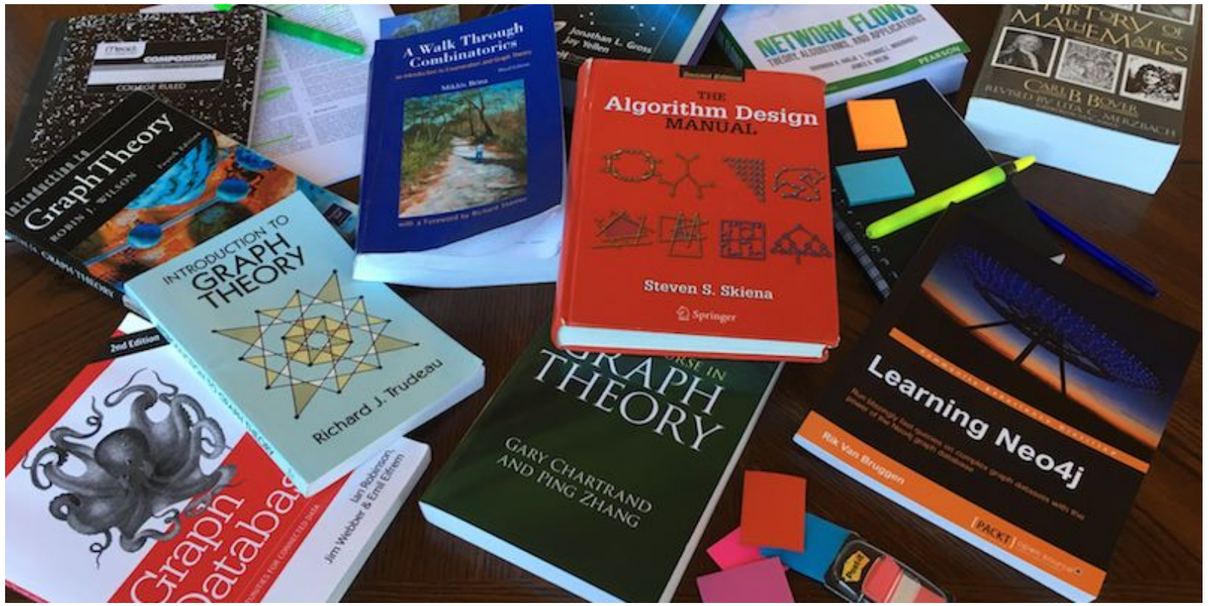
More examples

- Launch & Perf:
- Docs: neo4j-contrib.github.io/neo4j-graph-algorithms
- Tomaz Bratanic: tbgraph.wordpress.com (docs, social, tube, GoT)
- Community Graph: github.com/community-graph
- Twitter Analytics:

  neo4j.com/blog/graph-algorithms-make-election-data-great-again

Reading Material

- Thanks to Amy Hodler
- 13 Top Resources on Graph Theory
- neo4j.com/blog/
  top-13-resources-graph-theory-algorithms/



Please get in touch!!

We need your feedback & use-cases!

neo4j.com/slack -> #neo4j-graph-algorithms

github.com/neo4j-contrib/neo4j-graph-algorithms

michael@neo4j.com


The Sky is the Limit
Questions?

Don't forget! Subscribe & Like

youtube.com/c/neo4j