Multiple Graph Processing

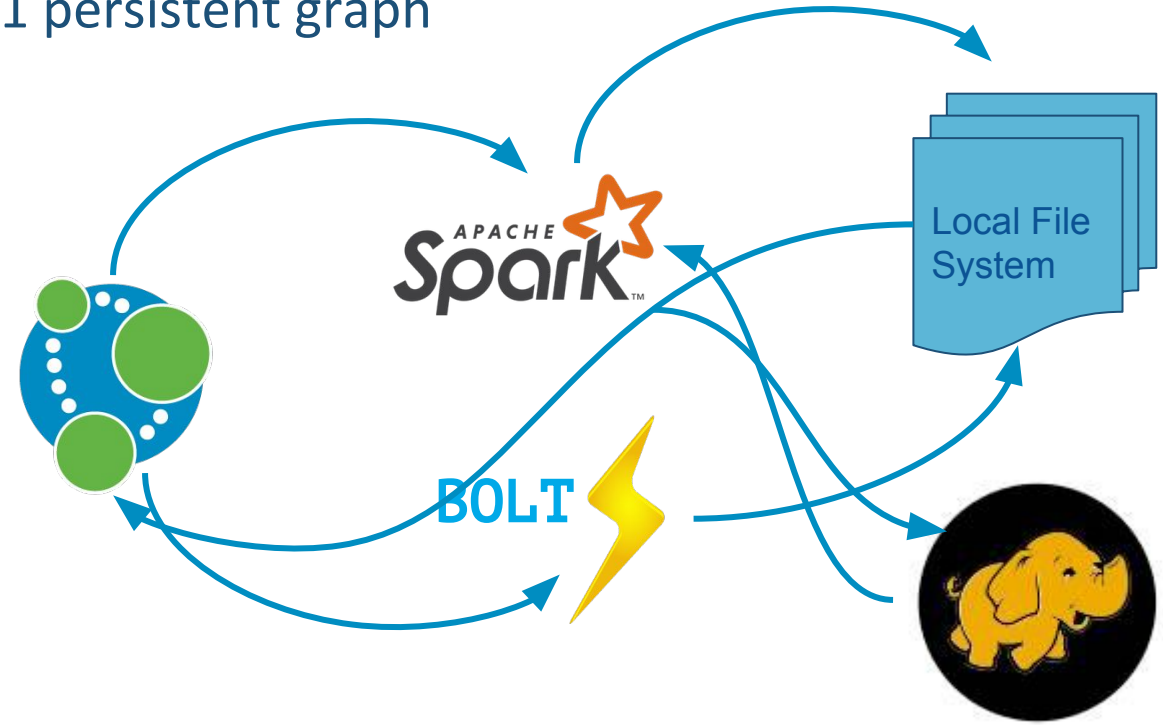
Naming and Addressing

Alastair Green

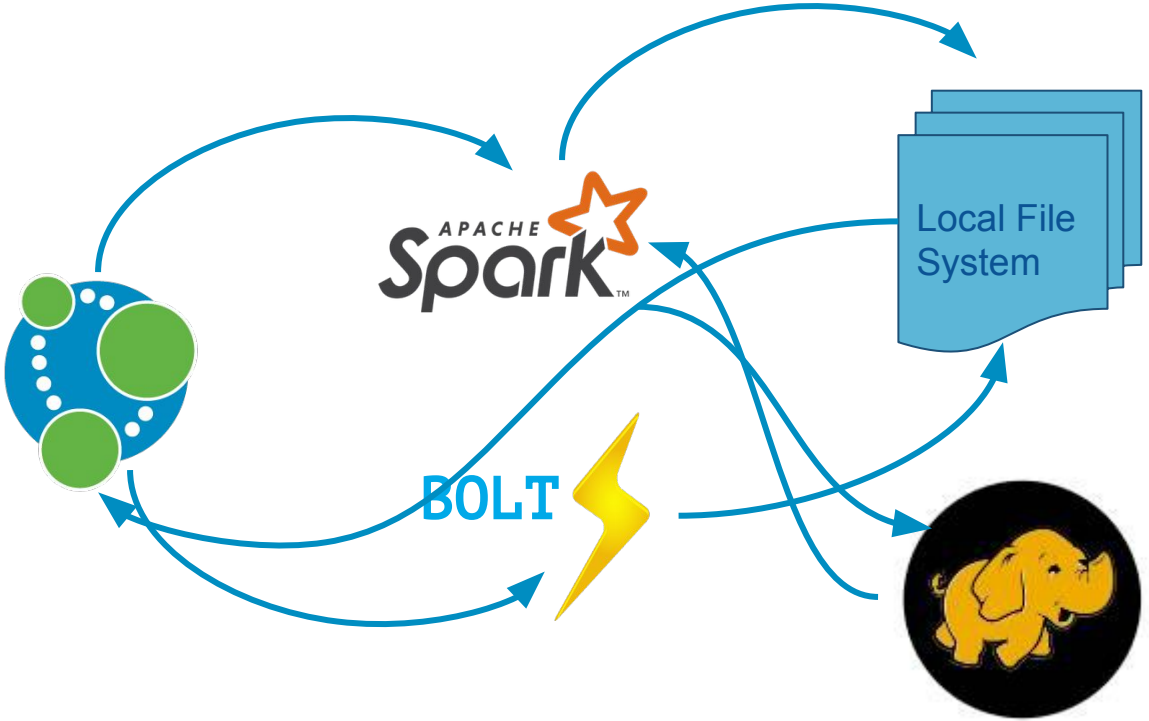
Neo Technology Cypher Language Group

Problem #1 Loading and saving

> 1 Store ⟶ > 1 persistent graph



Problem #1 Loading and saving



Neo4j instance *= 1 graph*

Neo4j ***CSV fileset*** *= 1 graph*

.graphml *= 1 graph*

.nq *= n graphs*

GraphX/Frame conventional ***V+E files*** *= 1 graph*

Containers and storage (rep + encoding) standards

Storage type	Container type	URI scheme	Storage Format	PG?
Neo4j	Neo4j Server	bolt://	Opaque	Y
CSV fileset	LFS, HDFS ...	file://, hdfs://	Under-specified	Y
.graphml	LFS, HDFS ...	file://, hdfs://	Well-specified	N
GraphX/Frames V+E files	HDFS, LFS, ...	file://, hdfs://	Under-specified	N
N-Quads	LFS, HDFS, ...	file://, hdfs://	Well-specified	N
...				

URI [+ directory/file convention] [+ PG convention]

How do we know where to find edges and vertices?

```
file:///path/to/the/data/devices/edges/*.csv | .txt
file:///path/to/the/data/devices/vertices/*.csv
```

How do we know where to find labels/types in *.csv?

What happens if we want to model graph properties?

- These questions have to be answered by (small) specifications
- Conventional/assumed answers may not be systematic or complete

`graph://` URIs

A URI scheme that can

- incorporate other URI schemes
- register/refer to sub-specifications or URI scheme extensions that define PG conventions
- allow the hierarchical path convention used by many URI schemes to be used to locate/identify containers or “spaces”
- allows every path to terminate with the unqualfied **name of a graph**

```
graph://bolt/'bolt+routing://neo4j-server:4567/social-networks/europe'
```

The meaning/depth of the path is implementation determined

It might imply folders, partitions or instances or tenants … or nothing

Loading and saving graphs in Cypher queries*

```
IN GRAPH
        EuropeSocialNetwork
AT
        graph://bolt/'bolt+routing://neo4j-server:4567/social-networks/europe'
MATCH
        (p1)-[:friends]-(p2)-[:friends]-(p3)-[:friends]-(p1)
CREATE
        (t:FriendsTriangle),
        (t)-[:contains]->(p1),
        (t)-[:contains]->(p2),
        (t)-[:contains]->(p3)
IN GRAPH
        FriendsTriangles
AT
      graph://graphml+pg/'hdfs://my-big-data/europe/snapshots/2017-08-23/friendsTriangles'
```

Load

Save

*** Example use case and pattern concepts from LDBC QL TF input by Hannes Voigt**

Or perhaps … or as well

Explicit

```
GRAPH [europe]
AT
        graph://bolt/'bolt+routing://neo4j-server:4567/social-networks/europe'

IN GRAPH
        europe
MATCH
        (p1)-[:friends]-(p2)-[:friends]-(p3)-[:friends]-(p1)
```

Implicit?

```
DEFAULT GRAPH AT
        graph://bolt/'bolt+routing://neo4j-server:4567/social-networks/europe'

MATCH
        (p1)-[:friends]-(p2)-[:friends]-(p3)-[:friends]-(p1)
```

Anonymous

Problem #2 Graphs in the scope of queries

Cypher queries are made up of *parts*

- Intermediate results are chained from part to part
- We will want to allow those results to be graphs
- There could be more than one, so names are needed
- And it might be good to allow intermediate graphs to be persisted

`WITH` or `IN`?

A named graph could have global scope in the query (cf. current Cypher anon graph)

This would mean it could always be referred to in a later part

And also used in `RETURN` as a final result

Problem #3 Graphs in the scope of sub-queries

A sub-query must be able to return a graph

```
IN
       {IN GRAPH someGraph MATCH CREATE ... AS GRAPH} // anonymous
MATCH
       ...
```

and it might be useful to name it (and perhaps persist it) for later reference

```
DECLARE DEFAULT GRAPH AT
   graph://bolt/'bolt+routing://neo4j-server:4567/social-networks/europe'

IN
       {MATCH ... CREATE ... AS GRAPH g [AT <url>]} // named [optionally saved]
MATCH
       ...
```

Problem #4 Queries returning graphs

Persisting a graph is not the same as returning a graph to a caller

- Critical to allowing graph queries to be part of function chains
- May be more than one graph that results from a query
- If so then each graph needs a name

```
CREATE
        (t:FriendsTriangle), (t)-[:contains]->(p1),
        (t)-[:contains]->(p2),(t)-[:contains]->(p3)
IN GRAPH
        FriendsTriangles
RETURN
        FriendsTriangles
```

Problem #5 Views

Logically we compose queries as functions over views ***q(v(G))***

```
IN GRAPH
        EuropeSocialNetwork
MATCH
        (p1)-[:friends]-(p2)-[:friends]-(p3)-[:friends]-(p1)
CREATE
        (t:FriendsTriangle), (t)-[:contains]->(p1),
        (t)-[:contains]->(p2), (t)-[:contains]->(p3)
IN GRAPH
        FriendsTriangles AS VIEW

IN GRAPH
        FriendsTriangles
MATCH
        ...
```