Neo4j Morpheus: Interweaving Documents, Tables and Graph Data in Spark

Alastair Green, Mats Rydberg

Neo4j

#SAISDD9

Introduction

Mats Rydberg *Engineering Lead for Cypher for Apache Spark and Neo4j Morpheus, Cypher Language Group*

Alastair Green *Lead, Neo4j Query Languages team, PM for Cypher for Apache Spark/Neo4j Morpheus and Cypher for Gremlin*

Neo4j Morpheus

- **A product in gestation, based on Cypher for Apache Spark**
- Enriching Spark's graph capability
- Combining Spark SQL with Spark graph querying
- Interweaving graph, table and nested/document data
- Integrating Spark analytics and Neo4j operational data
- Advancing graph query language (GQL) features

Property Graphs meet Big Data

The Property Graph data model is becoming increasingly mainstream

- Cloud graph data services like Azure CosmosDB or Amazon Neptune
- Simple graph features in SQLServer 2017, multiple new graph DB products
- Read-only graph queries coming in the SQL:2020 standard
- Neo4j becoming common operational store in retail, finance, telcos … and more
- Increasing interest in graph algorithms over graph data as a basis for AI

Apache® Spark is the leading scale-out clustered memory solution for Big Data

- Spark 2: Data viewed as **tables** (DataFrames), processed by **SQL**,
- in **function chains**, using queries and user functions,
- transforming **immutable** tabular data sets

Property Graphs Nodes (Entities)

Node

- Represents an entity within the graph
- Can have *labels*

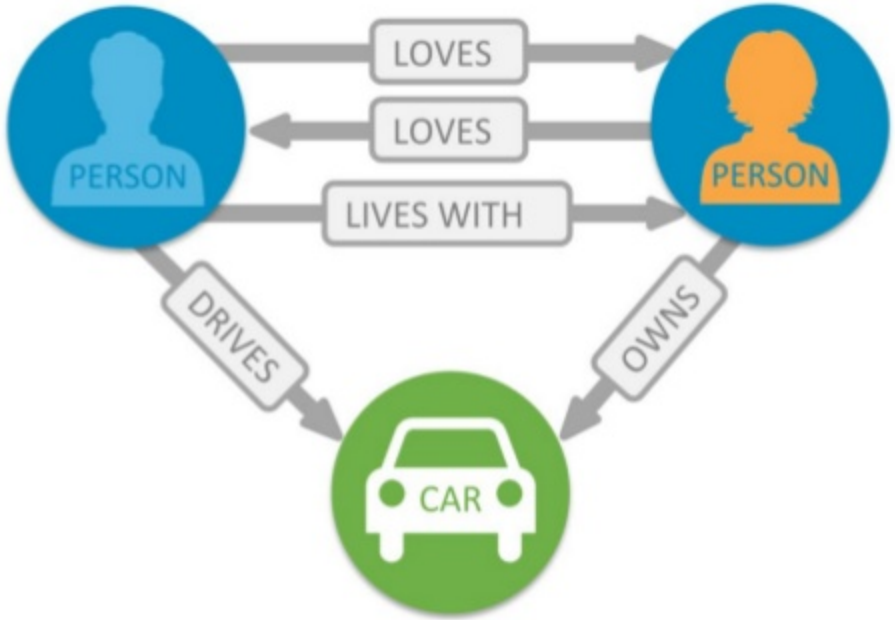
Property Graphs Relationships

Node

- Represents an entity within the graph
- Can have *labels*

Relationship

- Connects a start node with an end node
- Has one *type*



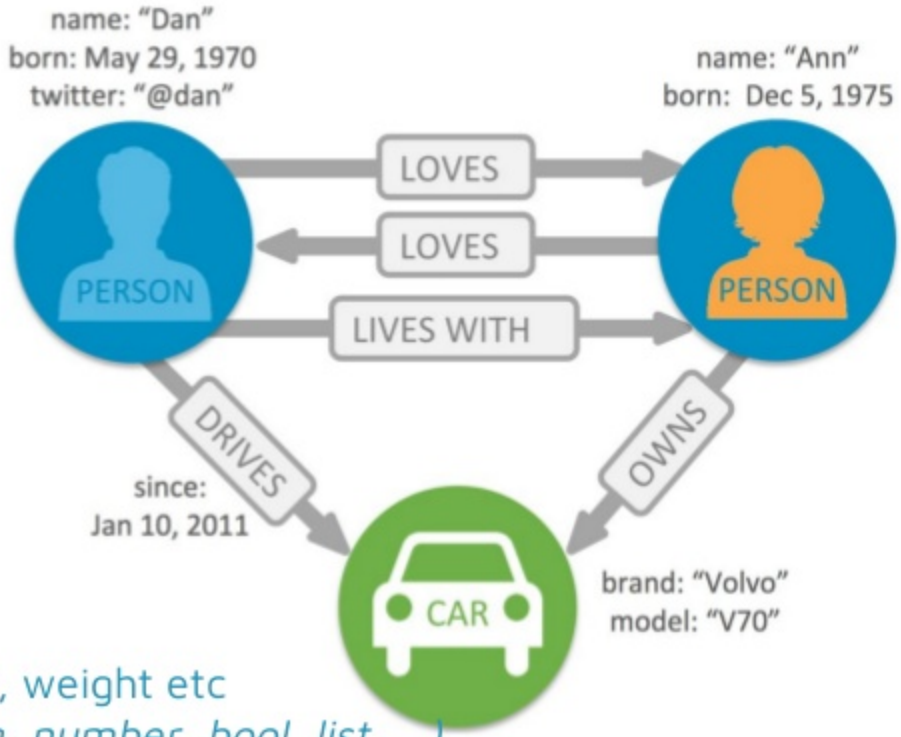
Property Graphs
Properties

Node

- Represents an entity within the graph
- Can have *labels*

Relationship

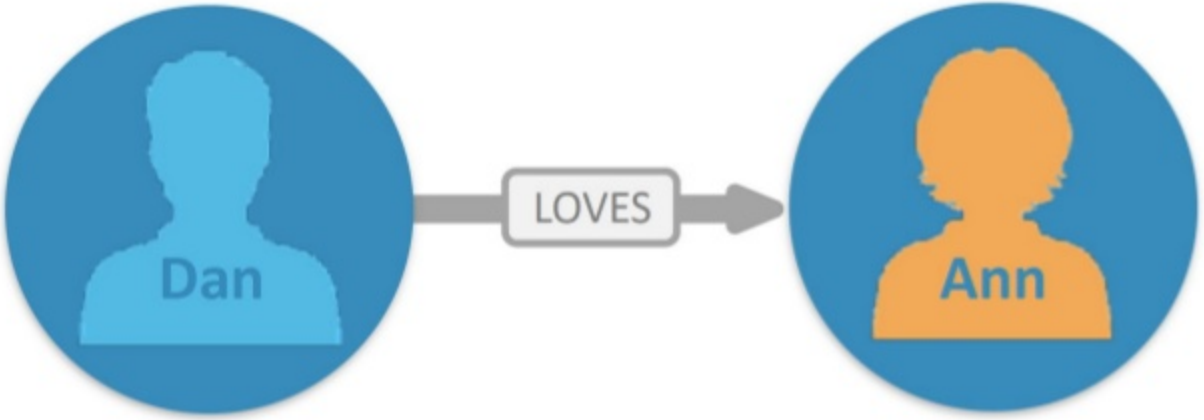
- Connects a start node with an end node
- Has one *type*



Property

- Describes a node/relationship: e.g. name, age, weight etc
- Key-value pair: *String* key; typed value *(string, number, bool, list, ...)*

Graph patterns

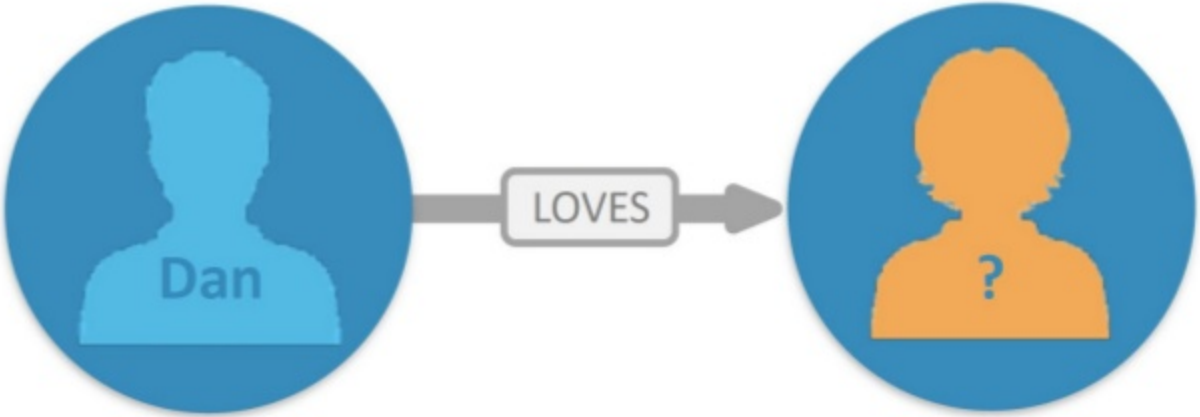


NODE — Relationship — NODE

(:Person { name:"Dan"}) -[:LOVES]-> (:Person { name:"Ann"})

LABEL — PROPERTY — LABEL — PROPERTY

Searching For Graph Patterns

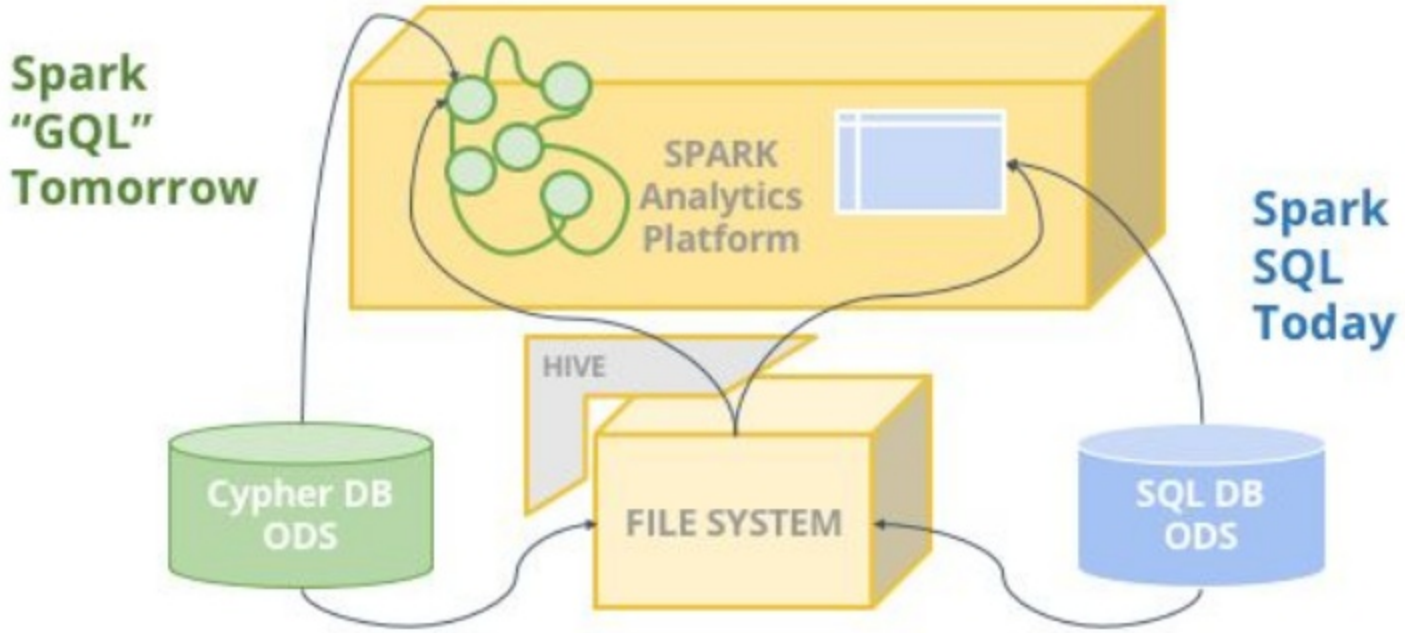


NODE — Relationship — NODE

```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

LABEL — PROPERTY — VARIABLE

Operational Tables and Graphs → Analytics



Spark: SQL and Cypher

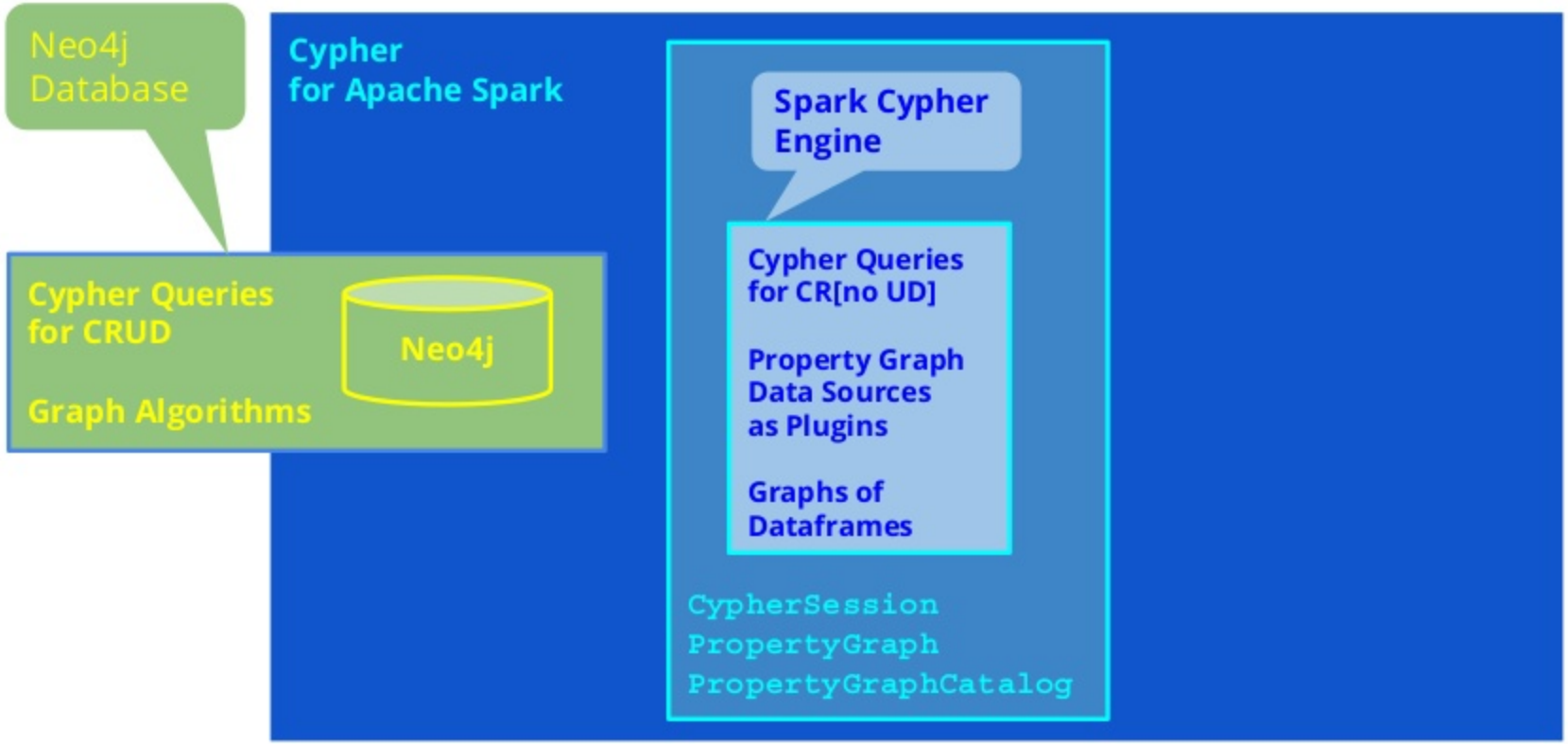
Apache® Spark is the leading scale-out clustered memory solution for Big Data

- Spark2: Data viewed as **tables** (Dataframes)
- With **StructType** schema to describe the type
- Processed by **SQL** and custom functions in **function chains**
- transforming input **immutable Dataframes** into output tables

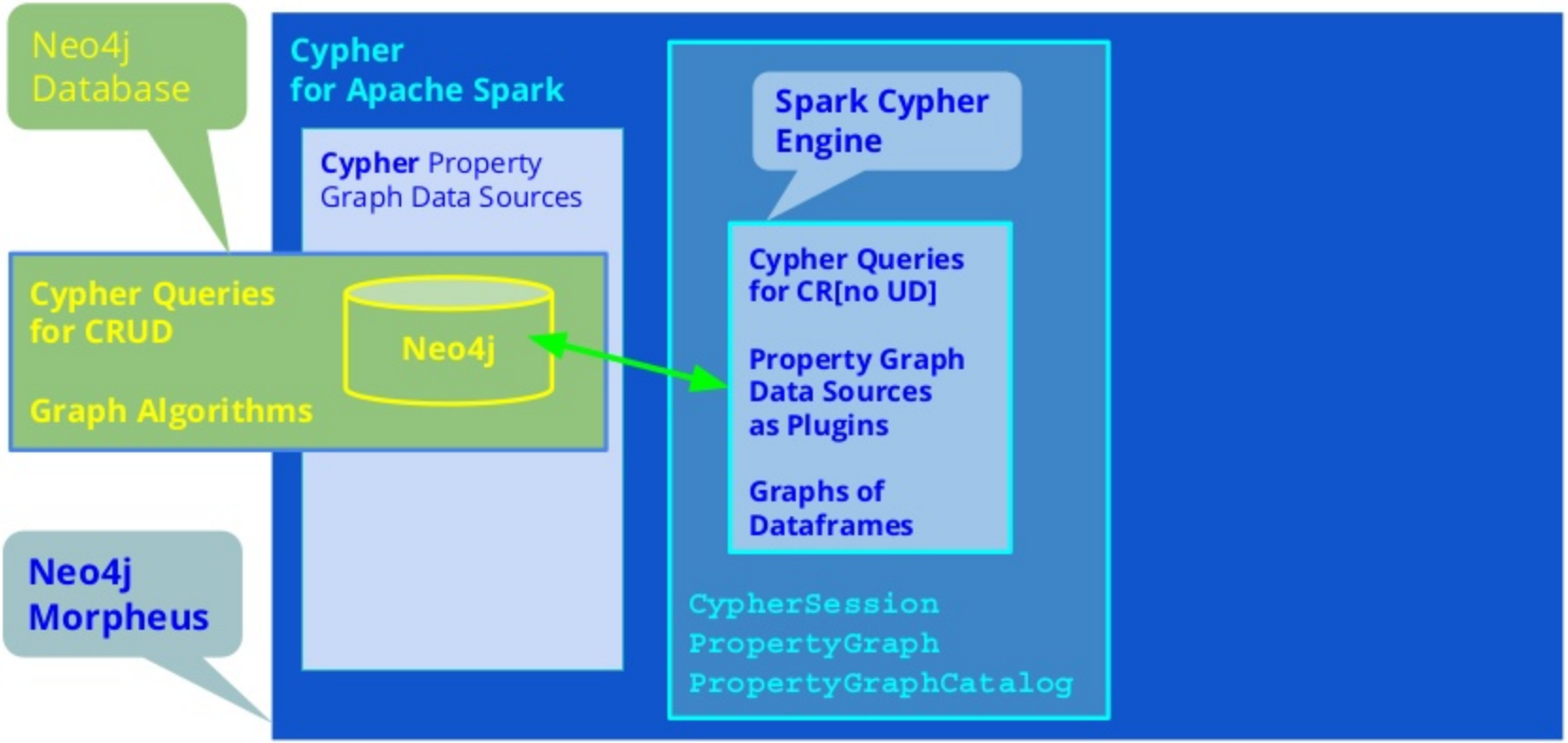
Cypher for Apache Spark (CAPS) mirrors the capabilities of Spark SQL

- Data viewed as **graphs** (made up of Dataframes)
- With **PropertyGraph.schema** to describe the graph type
- Processed by **Cypher** and user functions in **function chains** ,
- transforming input **immutable PropertyGraphs** into output graphs

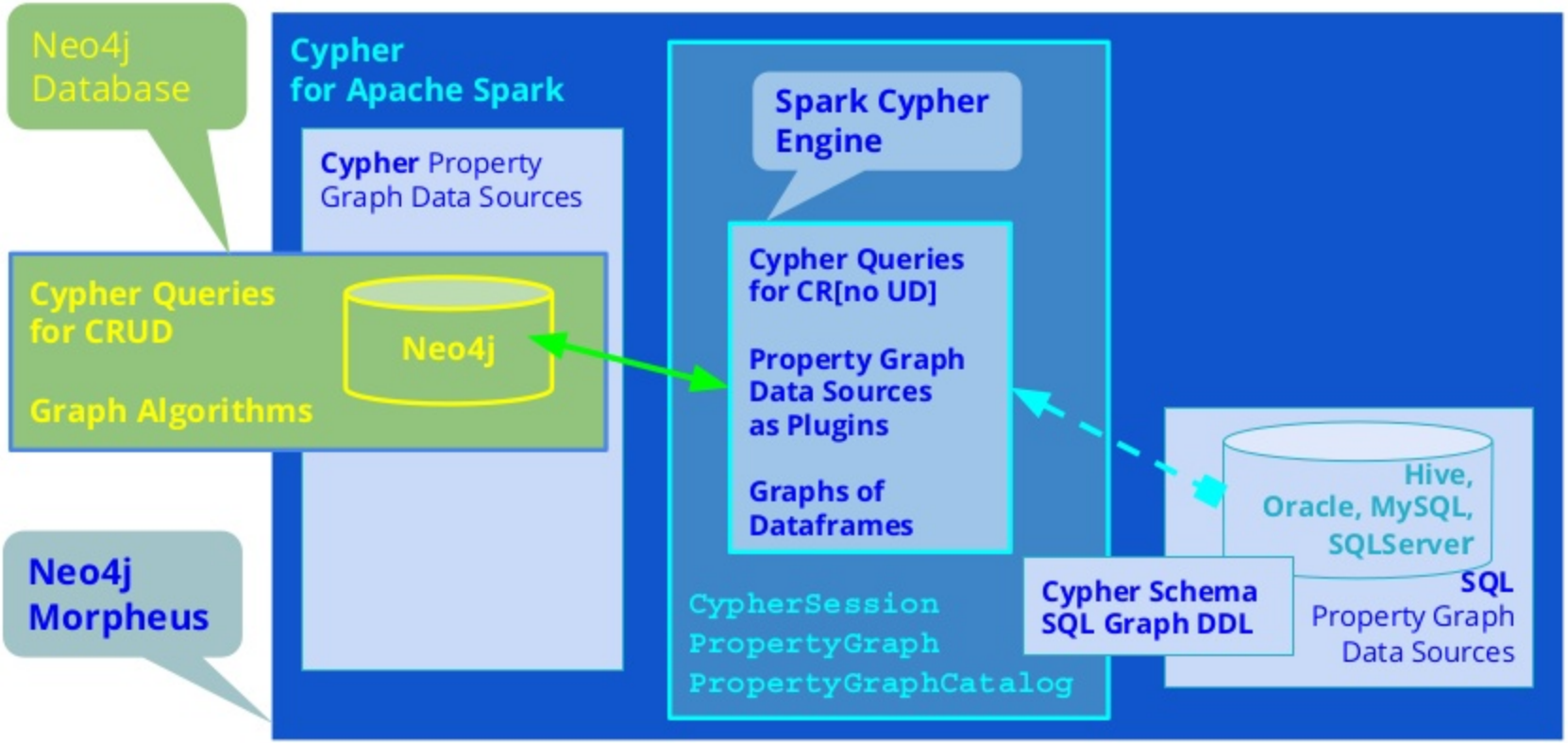
Neo4j Graph DB + Cypher for Apache Spark



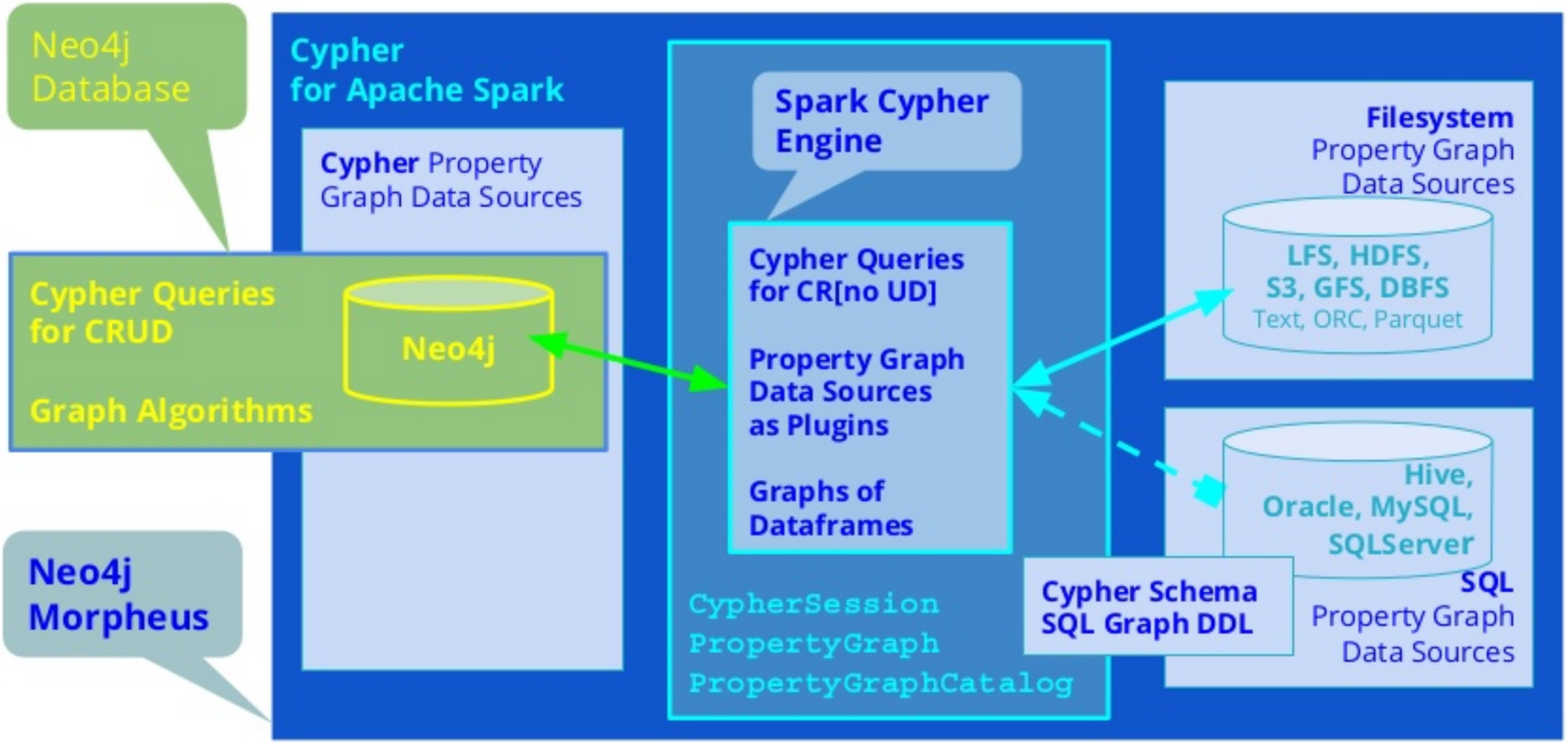
Cypher Property Graph Data Sources



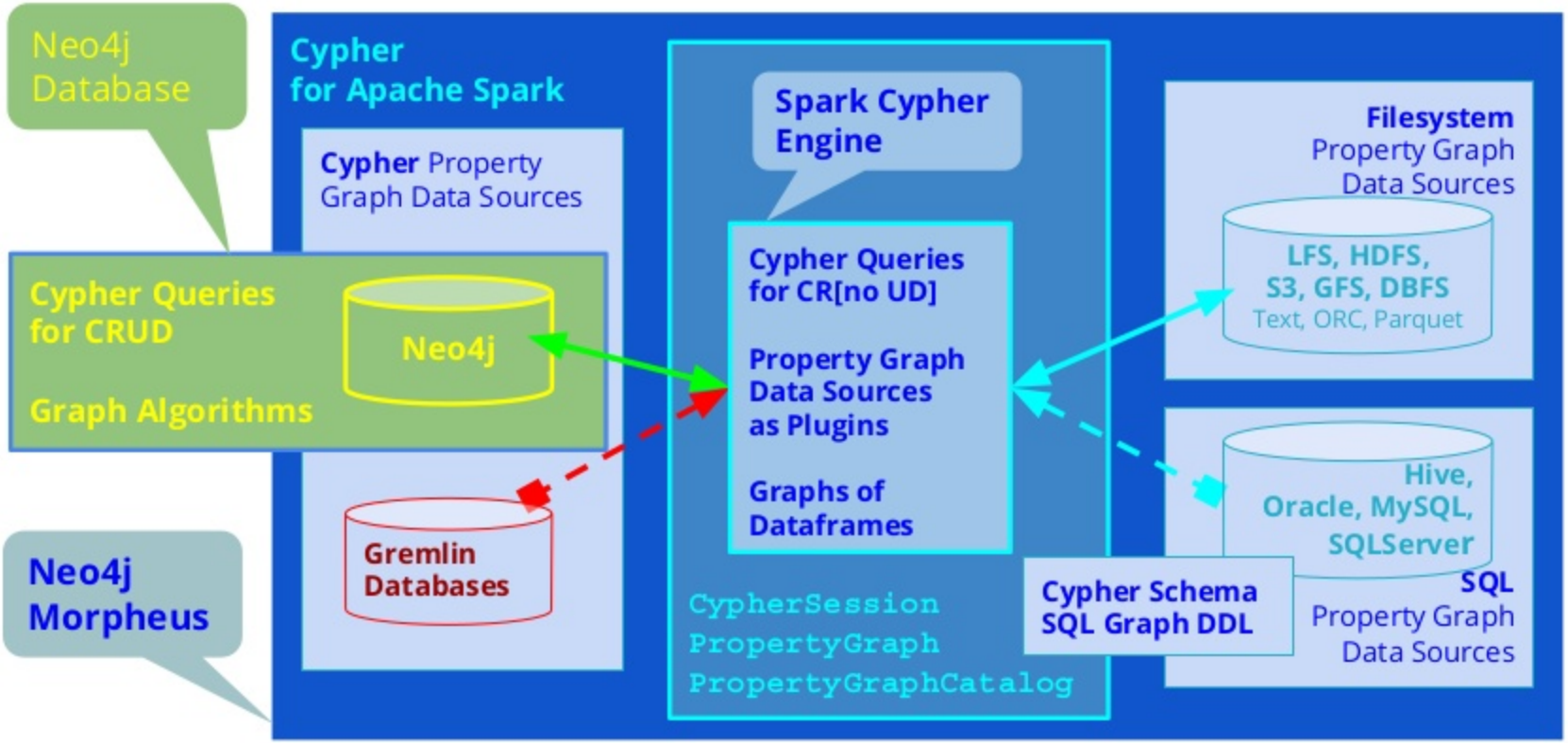
SQL Property Graph Data Source



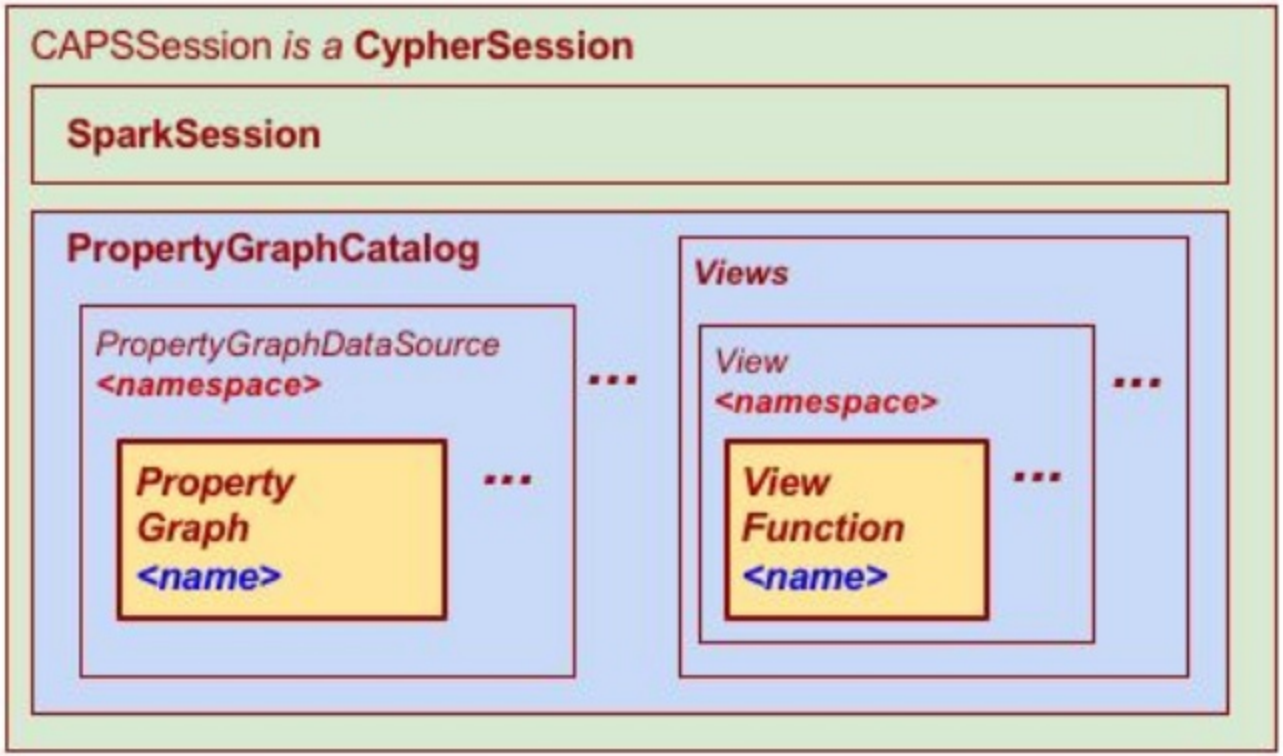
FS Property Graph Data Source



Future Property Graph Data Sources



Graphs and Views in the Catalog

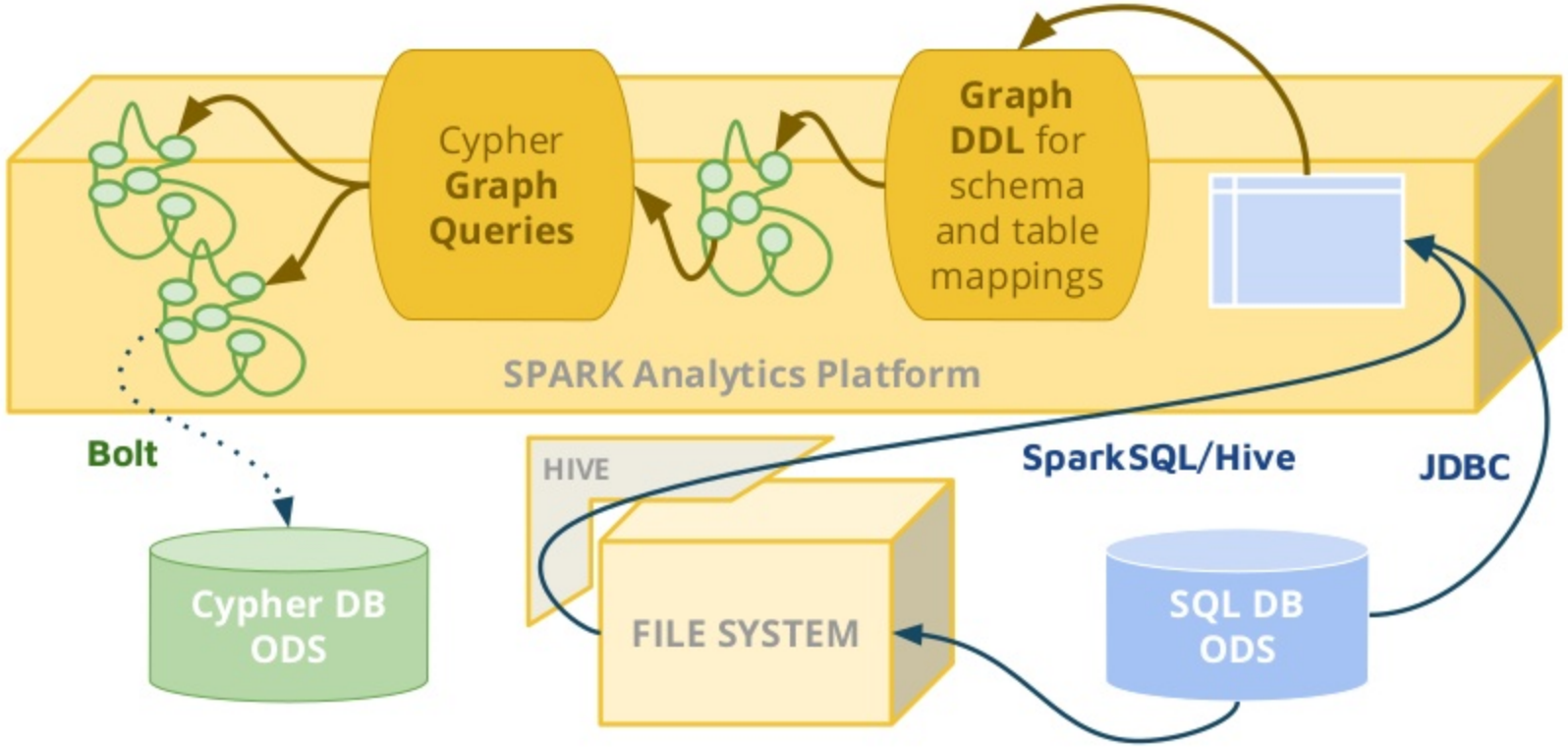



```
// named graph session.people
// view function session.peopleByCountry()
```

Querying (multiple) named graphs

```
// Session
implicit val session: CAPSSession = CAPSSession.local()
...

// Query
val result = session.cypher(
  """|FROM GRAPH socialNetwork
     |MATCH (p:Person)
     |FROM GRAPH products
     |MATCH (c:Customer)-[:ORDERED]->(i:Item) WHERE p.name = c.name
     |RETURN p.name, c.id, count(i.price) AS amount
  """.stripMargin)
result.show()
```

Turning Tables into Graphs



“Tables for Labels”

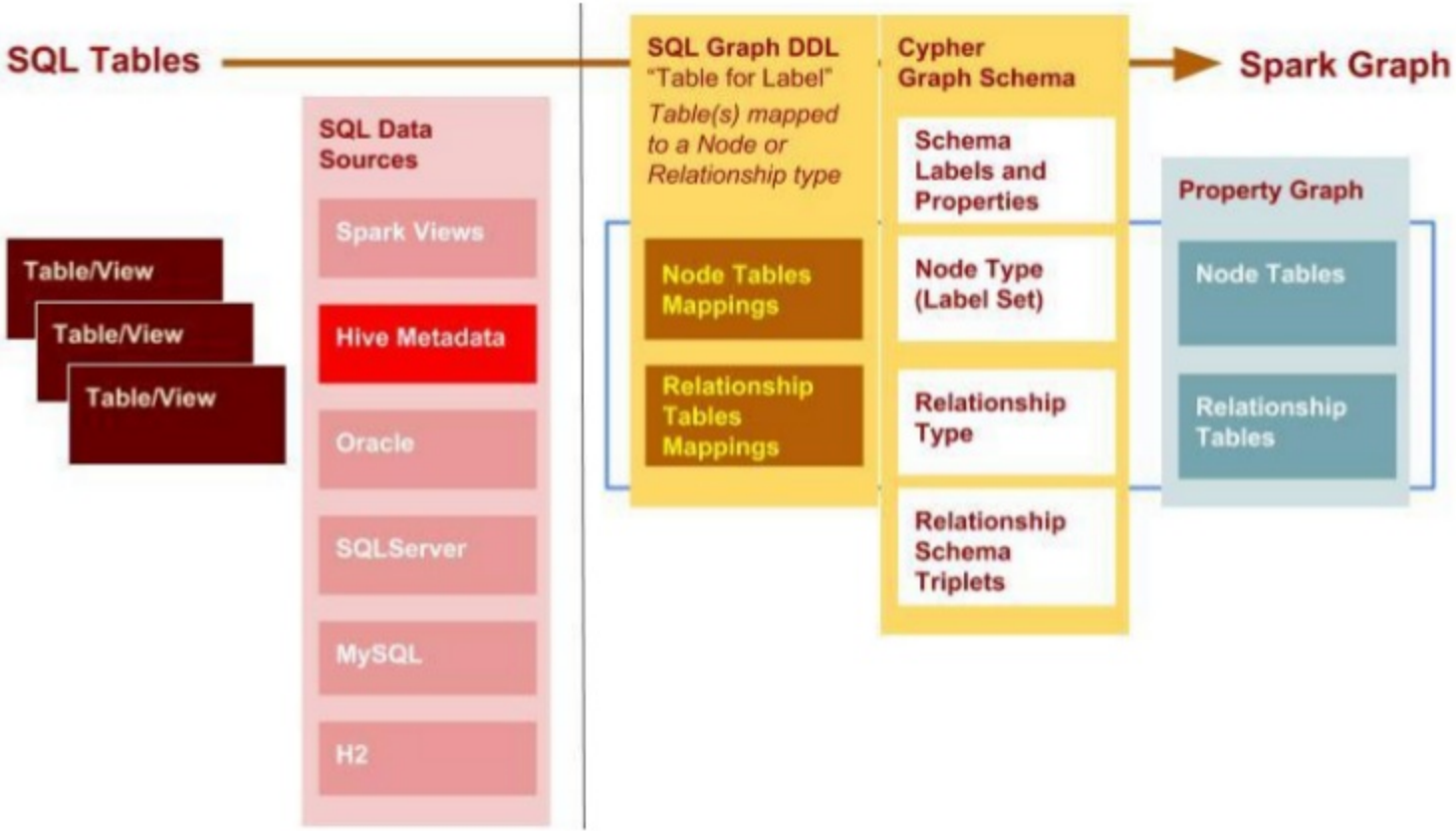
In Cypher for Apache Spark graphs have a **schema (graph type)**

The **schema** defines

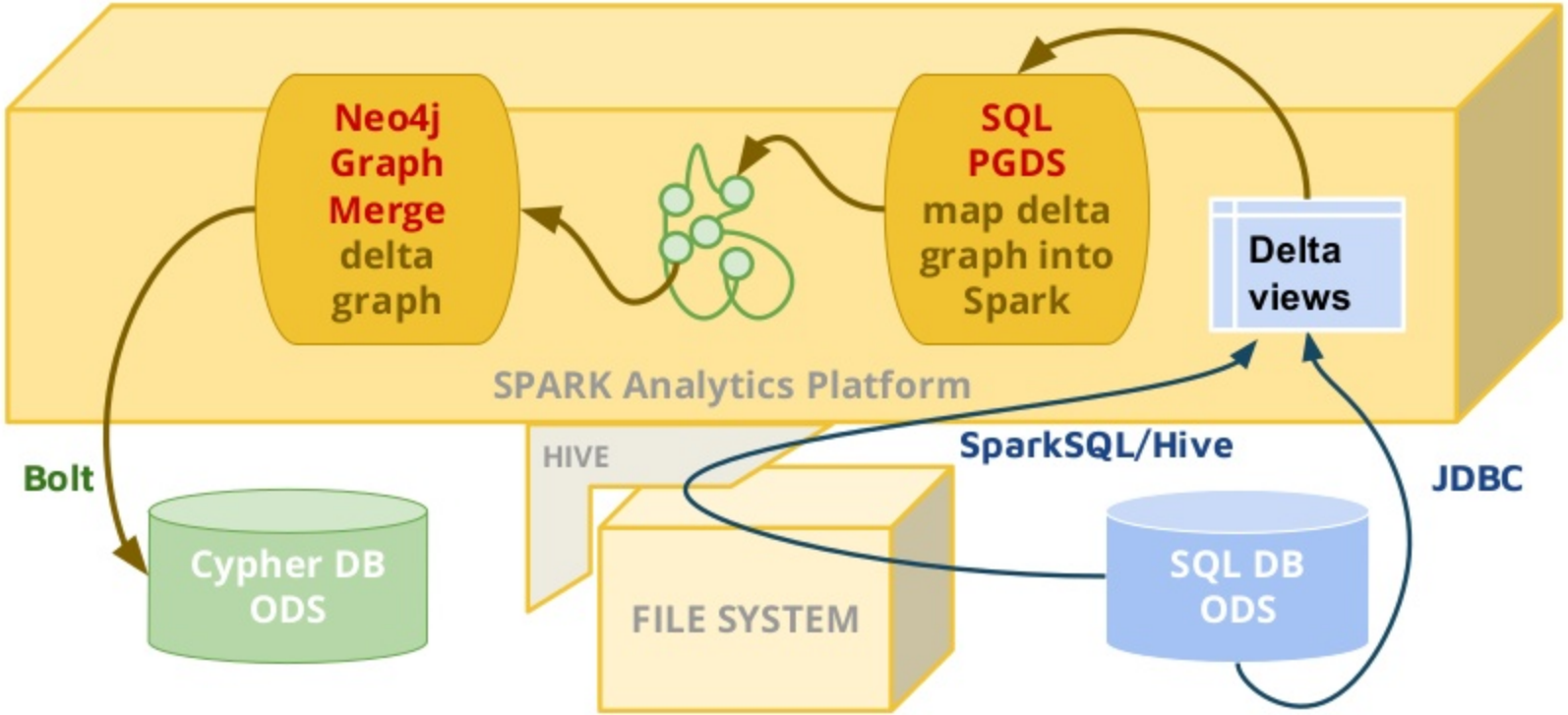
- The **properties** that belong to a **label**
- The **node types** and **relationship types** that occur in the graph
  - Node type is a **label set** (one or more labels → node type)
- **Relationship triplets** that permit node and relationship type combinations

A **graph instance** is made up of tables, one per node type/relationship type

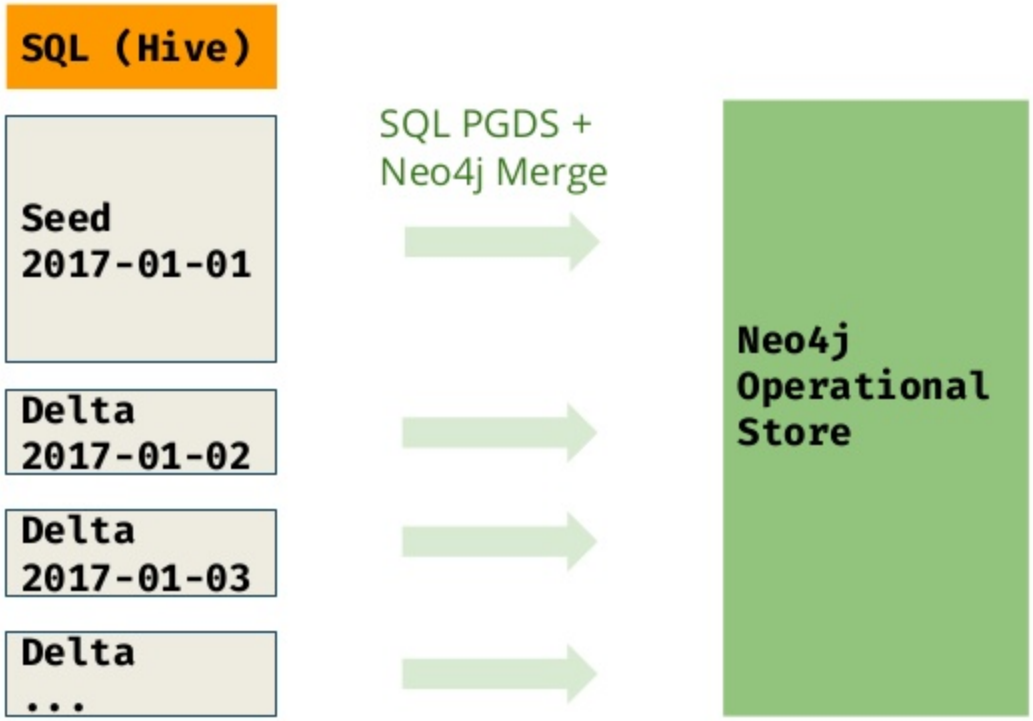
Mapping SQL tables into a Property Graph



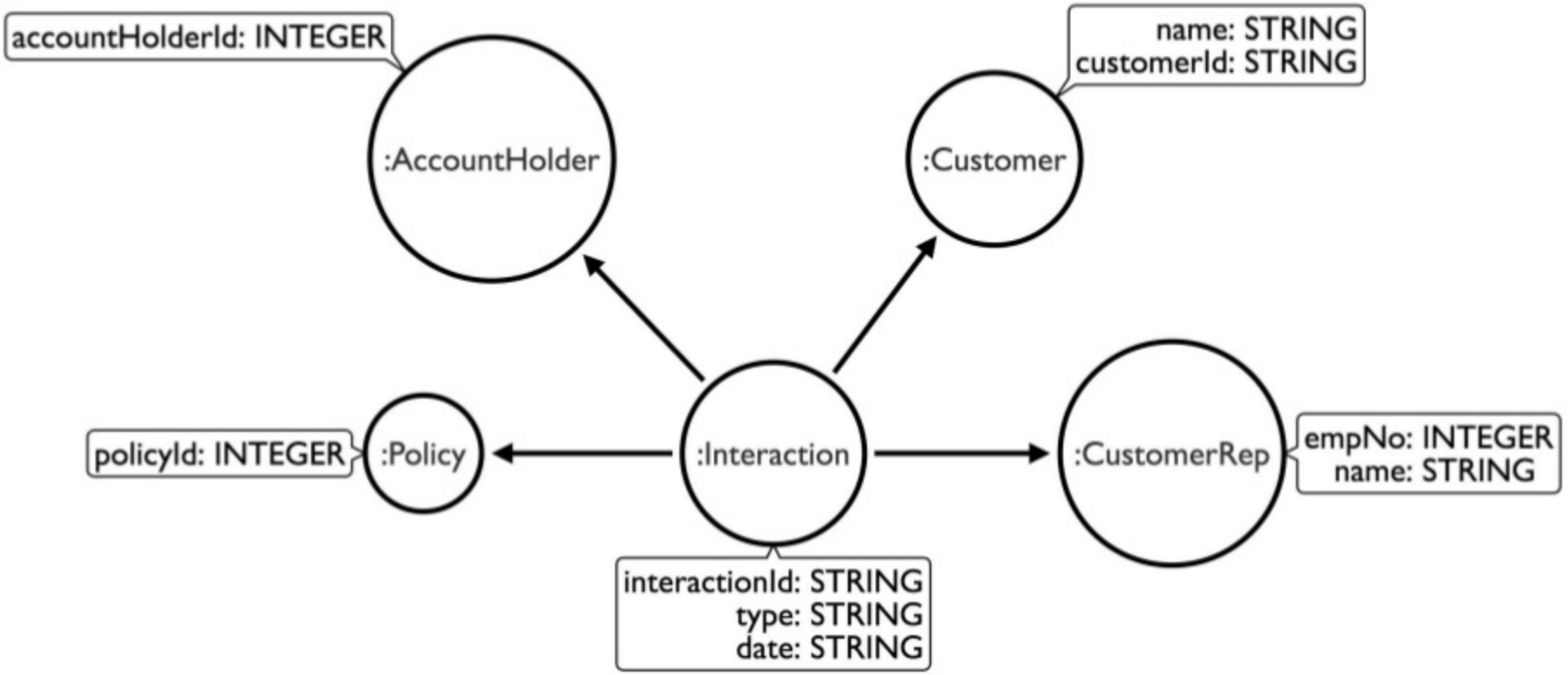
Synchronizing SQL data source with Neo4j



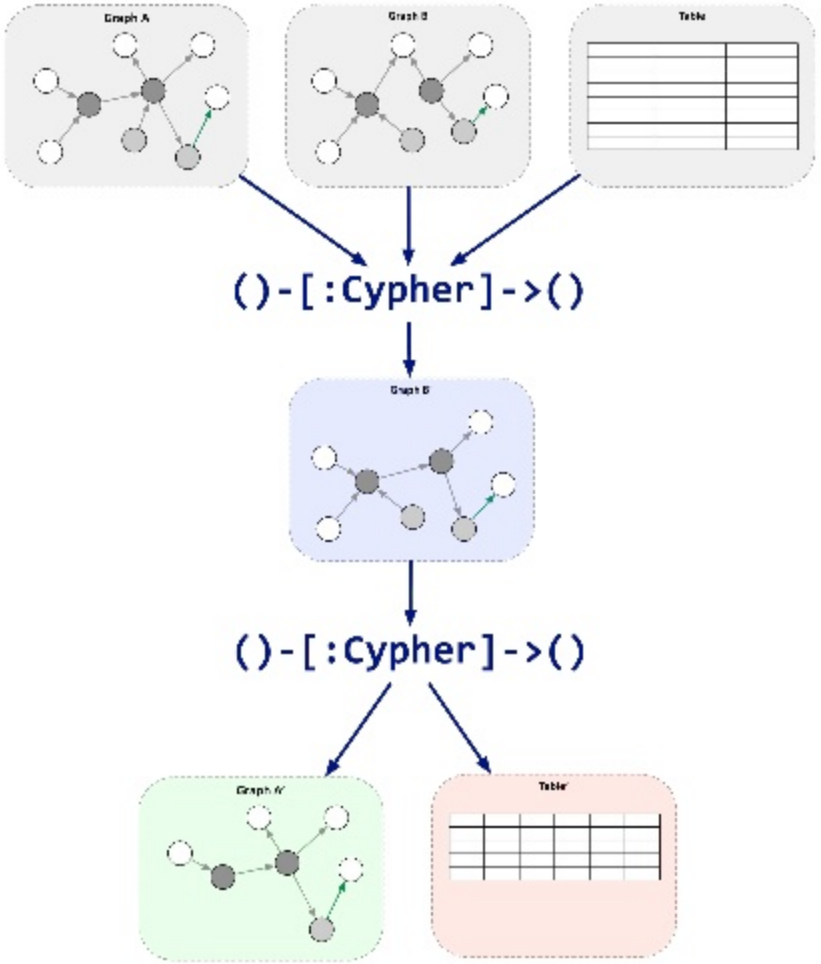
Demo: Customer360 incremental merge



Customer360 graph data model



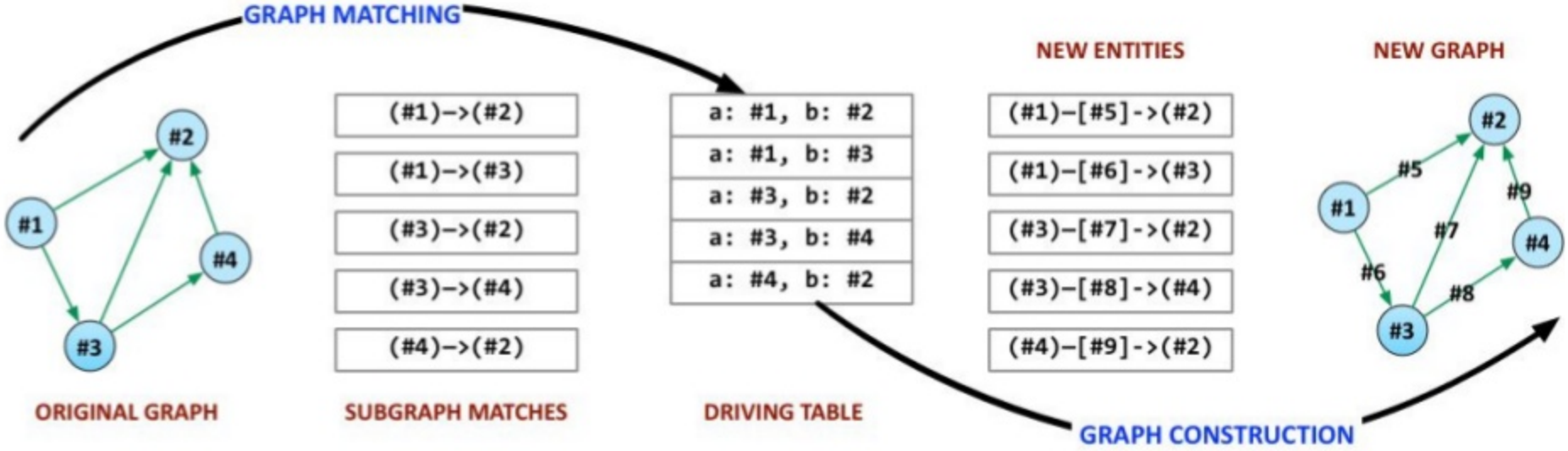
Query composition



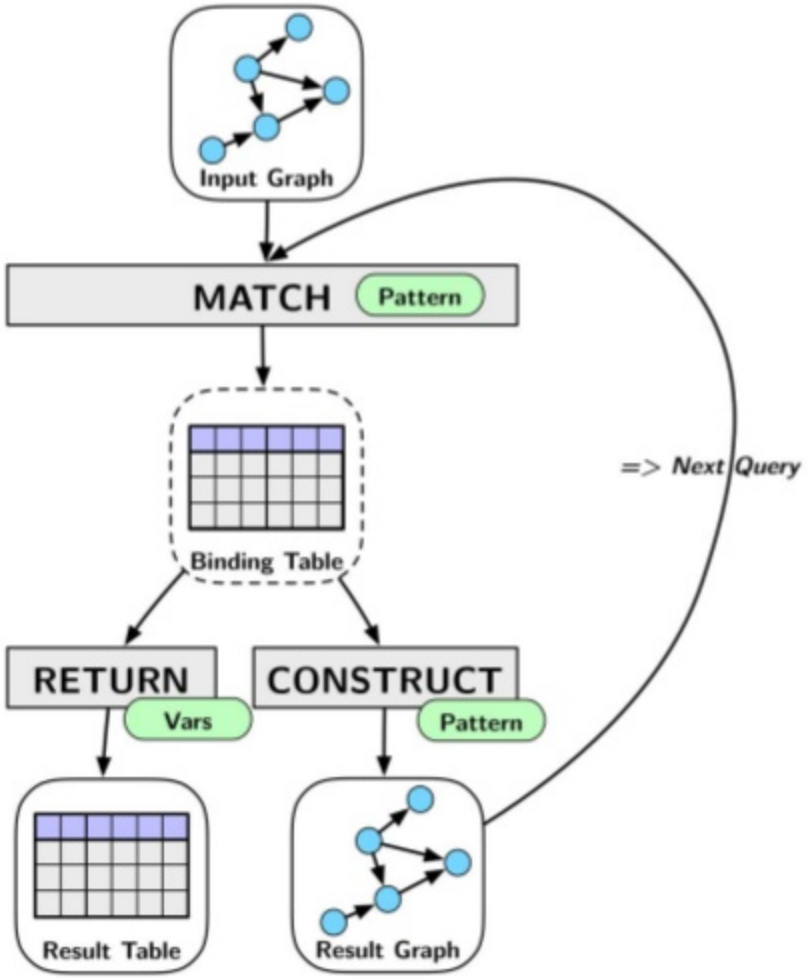
Query composition: graphs in, graphs out

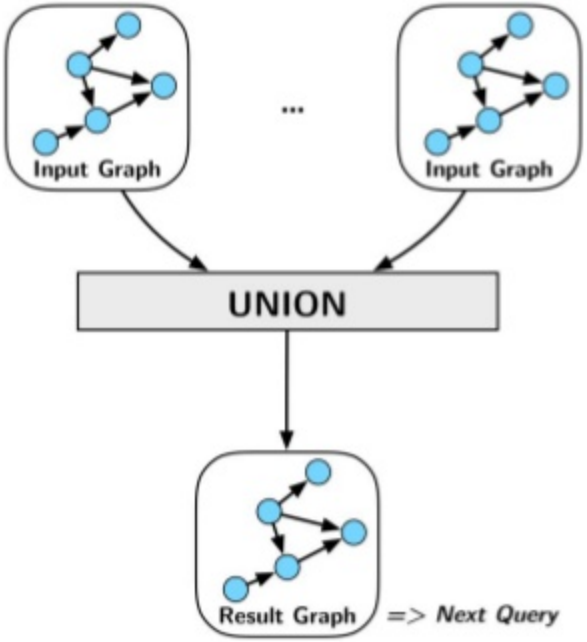
```
val graph = session.cypher("
  // Let's query two graphs...
  FROM GRAPH socialNetwork MATCH (p:Person)
  FROM GRAPH products MATCH (c:Customer) WHERE p.email = c.email

  // ...and construct a new graph from the result
  CONSTRUCT ON socialNetwork, products
  CREATE (p)-[:IS]->(c)
  RETURN GRAPH
").graph
```

Graph construction



Query composition





Cypher DDL to define schema objects

```
CATALOG CREATE GRAPH session.peopleUS {
    FROM session.people
    MATCH (us:Person)-[R*]-(n)
    WHERE us.nationality = 'USA'
    RETURN GRAPH OF *
}

CATALOG CREATE VIEW session.peopleByCountry($countryCode, $peopleGraph) {
    FROM $peopleGraph
    MATCH (us:Person)-[:R*]-(n)
    WHERE us.nationality = $countryCode
    RETURN GRAPH OF *
}
```

Using named graphs and view functions

```
// named graph
FROM session.peopleUS
MATCH (people:Person)
RETURN people.lastName

// view over graph
FROM session.peopleByCountry('USA', teradata.europe.people)
MATCH (people:Person)
RETURN people.lastName
```

Query composition: get data from new graph

```
// Let's query the graph we just constructed
val result = graph.cypher("
  MATCH (person:Person)-[:FRIEND_OF]-(friend:Person),
        (friend)-[:IS]->(customer:Customer),
        (customer)-[:BOUGHT]->(product:Product)
  RETURN DISTINCT product.title AS recommendation, person.name
  ORDER BY recommendation
")
result.show()
```

SQL and Cypher working together

```
// Query with Cypher
val results = socialNetwork.cypher("
  MATCH (a:Person)-[r:FRIEND_OF]->(b)
  RETURN a.name AS friend1, b.name AS friend2, r.since AS since
")

// Extract DataFrame representing the query result and register
results.records.asDataFrame.createOrReplaceTempView("friends")

// Query with SQL
spark.sql("
  SELECT friend1, friend2, since FROM friends ORDER BY since
").show()
```

Demo: Database snapshots for analytics

Cypher Graph Schema Labels and Properties

```
CREATE GRAPH snb WITH GRAPH SCHEMA (
  LABEL (Company),
  LABEL (Message {
      creationDate     : TIMESTAMP?,
      locationIP       : STRING?,
      content          : STRING?,
      length           : INTEGER?
  }),
  LABEL (LIKES {
    creationDate     : TIMESTAMP?
  }),
  ...
```

Cypher Graph Schema Node/Edge Types

```
// allowed node label sets (node types)
(Message, Post),
(Company),
(Country),
(Person),
...

// allowed relationship (edge) types
[LIKES],
[KNOWS],
[IS_LOCATED_IN],
...
```

Cypher Graph Schema Relationship Triplets

```
// relationship triplets
(Message)    <0 .. *>  -[IS_LOCATED_IN]->  <1>       (Country),
(Person)     <1>       -[LIKES]->          <0 .. *>  (Message),
(Company)    <0 .. *>  -[IS_LOCATED_IN]->  <1>       (Country),
(Person)     <0 .. *>  -[KNOWS]->          <0 .. *>  (Person),

// end of pure graph schema
```

SQL Graph DDL Mapping "Tables to Labels"

```
RELATIONSHIP LABEL SETS (
  (HAS_CREATOR)
    FROM "postHasCreator" edge
    START NODES
      LABEL SET (Message, Post)
      FROM "Post" start_nodes
      JOIN ON start_nodes.ID = edge."post"
    END NODES
      LABEL SET (Person)
      FROM "Person" end_nodes
      JOIN ON end_nodes.ID = edge."creator"

  ...
))
```

Relationship Label Set → **Relationship Type**

SQL **Relationship Source Table**

Node Label Set → **Node Type**

SQL **Node Source Table**

SQL PGQ and GQL

Morpheus SQL Property Graph Data Source **SQL Graph DDL**

A prototype of **SQL Property Graph Query** (SQL/PGQ)

Will be part of ISO SQL:202x

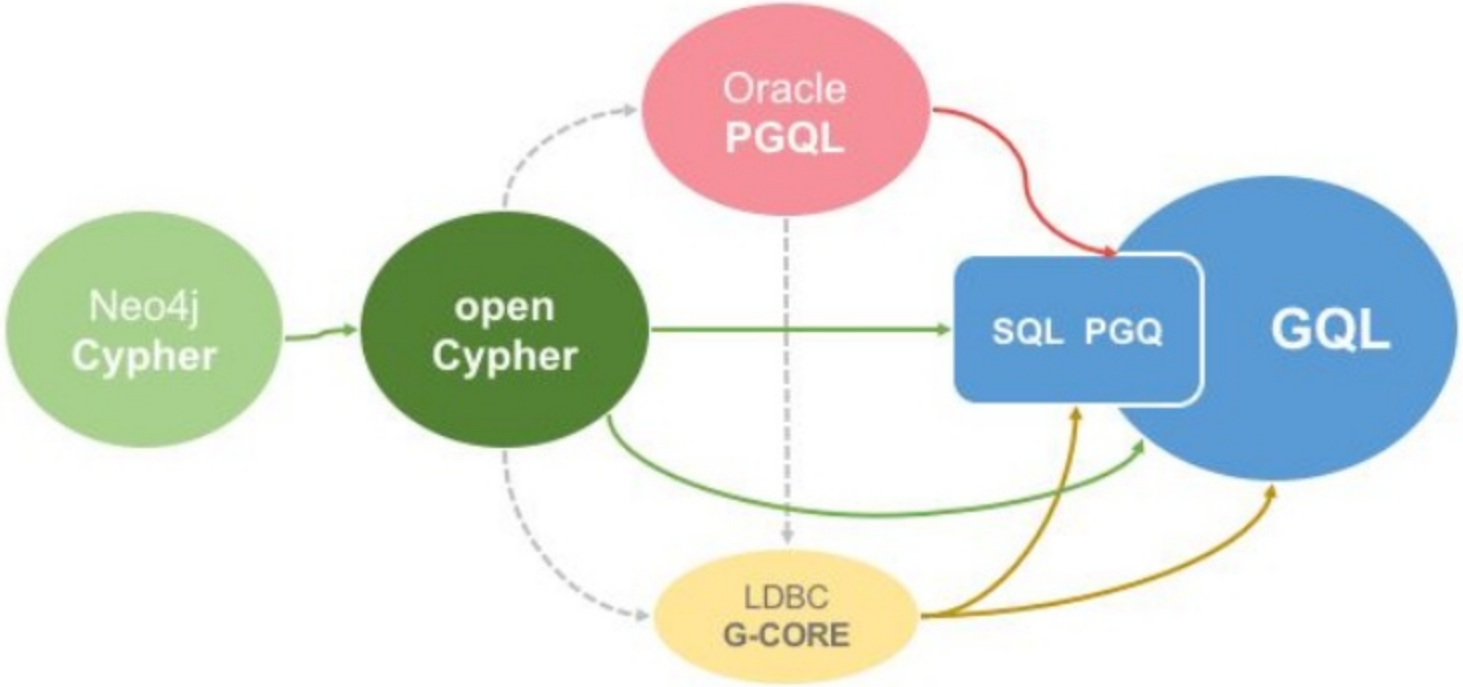
Morpheus SQL PGDS **Graph Schema** and **Composable Graph Queries**

"Prequel to GQL"

GQL is an initiative to create an **ISO international GQL standard alongside SQL**

SQL PGQ planned as a (major) subset of a wider "native" full CRUD GQL

From Cypher GQL



Spark SQL and Cypher → Spark SQL and GQL

Cypher for Apache Spark developed by Neo4j

- Source code under Apache 2.0 license
- **github.com/openCypher/cypher-for-apache-spark**

Considering **SPIP to bring enhanced graph support to Spark**

- Ongoing discussion with Databricks Apache Spark/GraphFrames developers
- Concept: DataFrame-based graphs and Cypher graph queries

Directions

- Cypher → **ISO GQL**
- Cypher for Spark → **Spark GQL + Spark SQL**

```
spark.sql
spark.gql
```

Neo4j Morpheus planned timeline

Neo4j Morpheus a new product, complementing the **Neo4j transactional DB**

- Commercially supported Cypher for Apache Spark
- Certified for Spark distributions and for SQL data sources

Limited access **Early Adopter release scheduled for end October**

Sign up to join EA programme https://neo4j.com/morpheus/

Documentation: https://neo4j.com/docs/morpheus-user-guide/preview/

To come

- Cypher language feature expansion
- Performance and scale testing in Q4/Q1

H1 2019 1.0 release as an extension to Neo4j Enterprise

Questions?