Evolving Cypher for Processing Multiple Graphs

Stefan Plantikow

stefan.plantikow@neotechnology.com

Lead openCypher Language Group (CLG)
Neo Technology

About this talk

- This is a proposal on how to add support for working with multiple graphs to the Cypher property graph query language, as part of the openCypher project.

- It is informational only: none of the ideas and proposed features presented in the following material are a part of the Cypher standard nor are they available from Neo Technology in any product offering, nor does this presentation represent any commitment that Neo Technology is going to provide such features in the future.

1st OpenCypher Implementer's Meeting (oCIM) Yesterday

- 35 participants from
  SAP, IBM, Oracle, Bitnine, Redis, Neueda, Dgraph, Memgraph, Neo,
  Universities Leipzig, Budapest, Dresden, Edinburgh

- Announcements
  - Neo4j's Cypher frontend to be APL 2.0 licensed
  - TCK and Grammar releases
  - Collaboration on formal semantics with the University of Edinburgh
  - New Cypher implementations
  - Proposal for **openCypher Implementers Group (oCIG)**

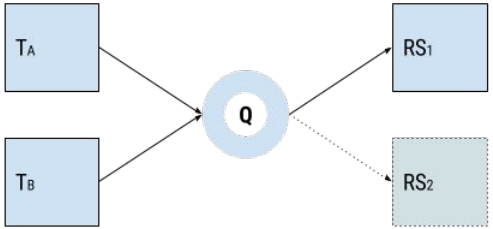
Cypher 2017

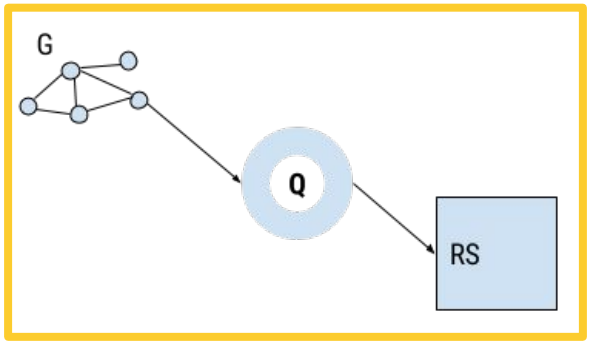
- Most widely-used declarative query language for property graphs
- Continuously adding new features
  - Conjunctive Regular Path Queries (CRPQs)
  - Additional kinds of subqueries (correlated and scalar subqueries)
  - Additional uniqueness modes (homo- and isomorphic matching)
  - New kinds of constraints
  - New data types and functions
  - Improved text search
  - ...
- Yet, still returns only returns records...

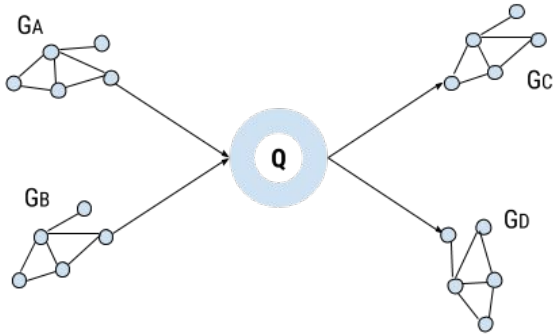
T → T … G → T … G → G’

The world of data management is often tabular in mindset or appearance

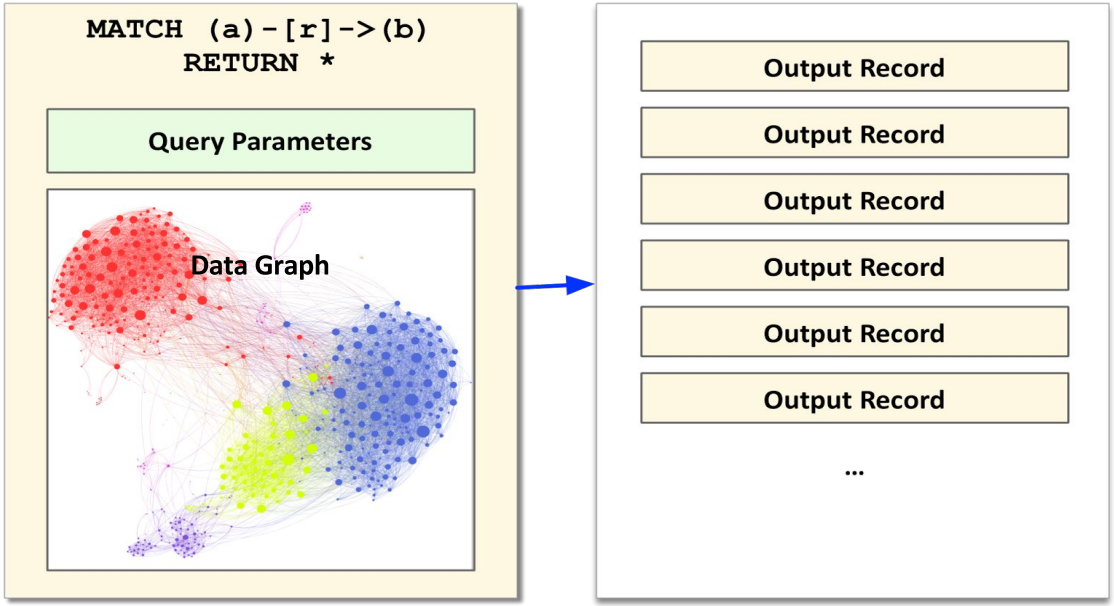
Stores, visualization, reports, spreadsheets, **Cypher result sets** …



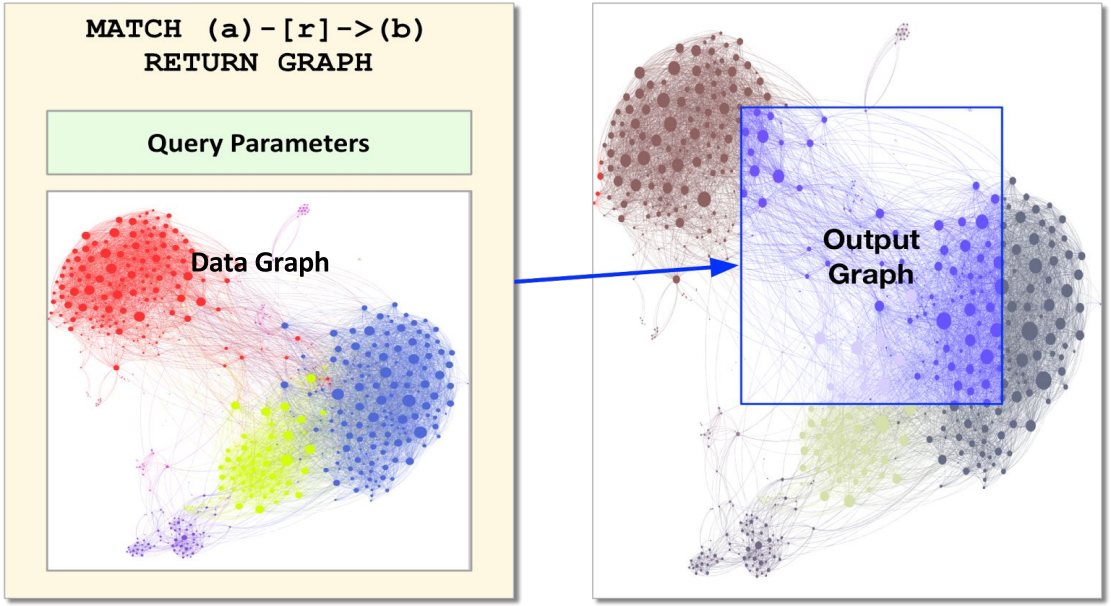




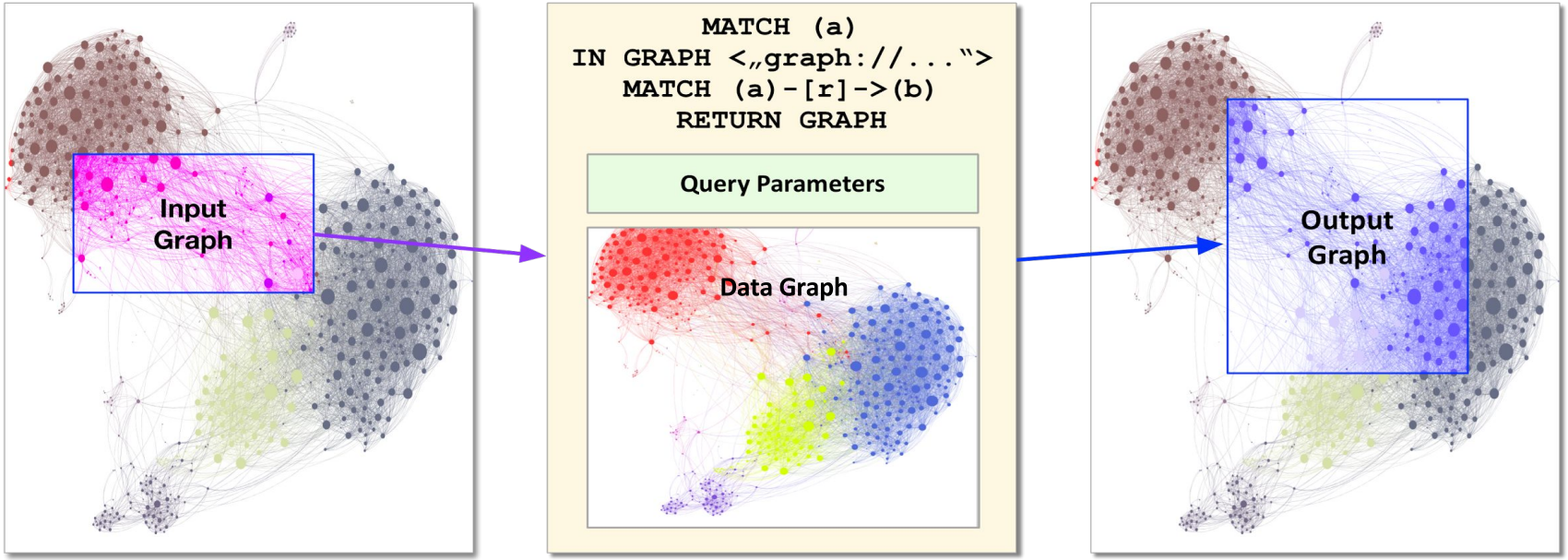
From producing records (i.e. G->T)



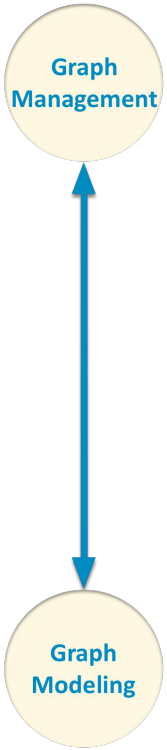
...to producing graphs G->G'....



...to consuming & combining & producing graphs: G->...->G'

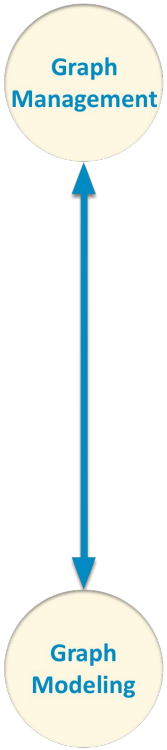


Why Multiple Graphs?



- Management of multiple graphs inside a GDBMS
  - Federation across organizational boundaries
  - Natural Sharding and Partitioning
    - e.g. by country, region etc.
  - Structuring the graph data set for operational purposes
  - System graph
  - Access control
  - Snapshots
  - Versioning

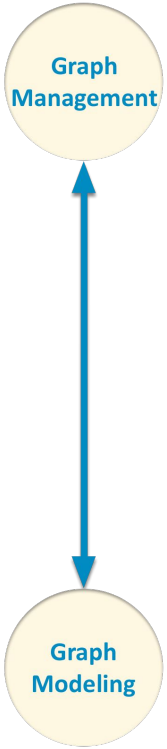
Why Multiple Graphs?



- Views: Transform, filter, aggregate graphs inside the GDBMS
  - **<G>⟶<G'>**
  - Application data provisioning
  - Incremental maintenance of aggregates
  - Analytical processing in big data systems

- Graph Visualization
  - How to return relevant entities in a systematic fashion?

Why Multiple Graphs?



- Inter-graph operations
  - What is the difference graph between now and yesterday?
  - How do these two cliques of social graphs intersect?
  - How to contract parts of a larger graph to see the bigger picture?
  - Updating graphs

- Graphs as a modeling tool
  - How do we represent a route in the graph?
    (e.g. Travel trips, Bus routes)
  - How do we relate larger parts of the same graph to each other?
    (e.g. Fraud ring tracking)

Impact of support for querying multiple graphs

Physical model Where are graphs (nodes, relationships) stored? How are they addressed?

Logical model How to add discrete multiple graphs to the Property Graph Model?

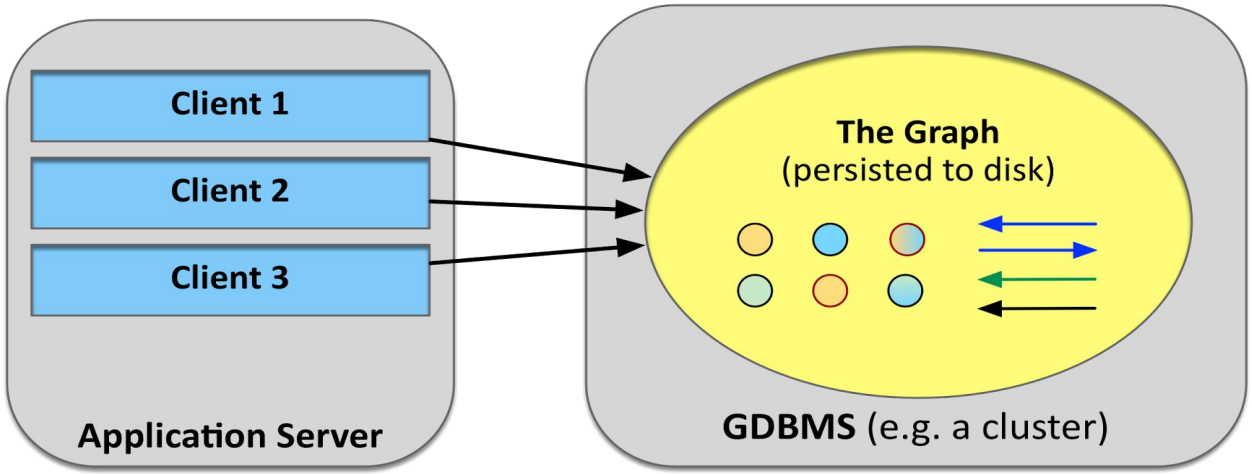
Language model How are graphs represented in Cypher? As values? Between operators?

Client model How are graphs returned to the client?

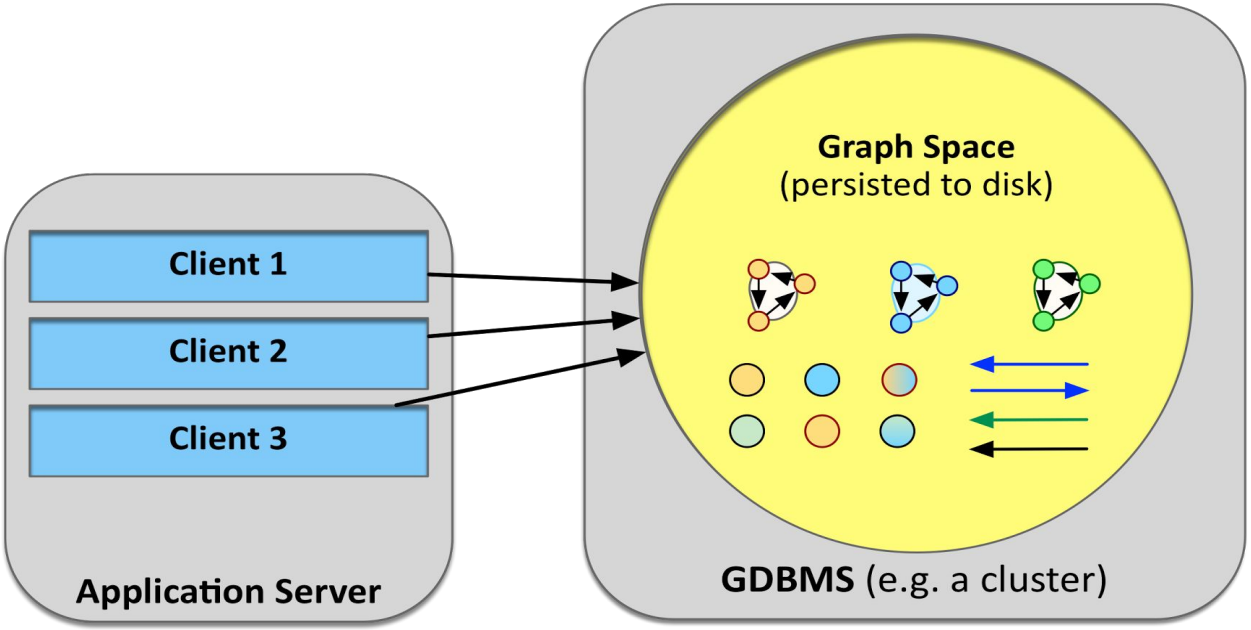
Lateral Impact on existing features and useability

Physical model today: Single Graph

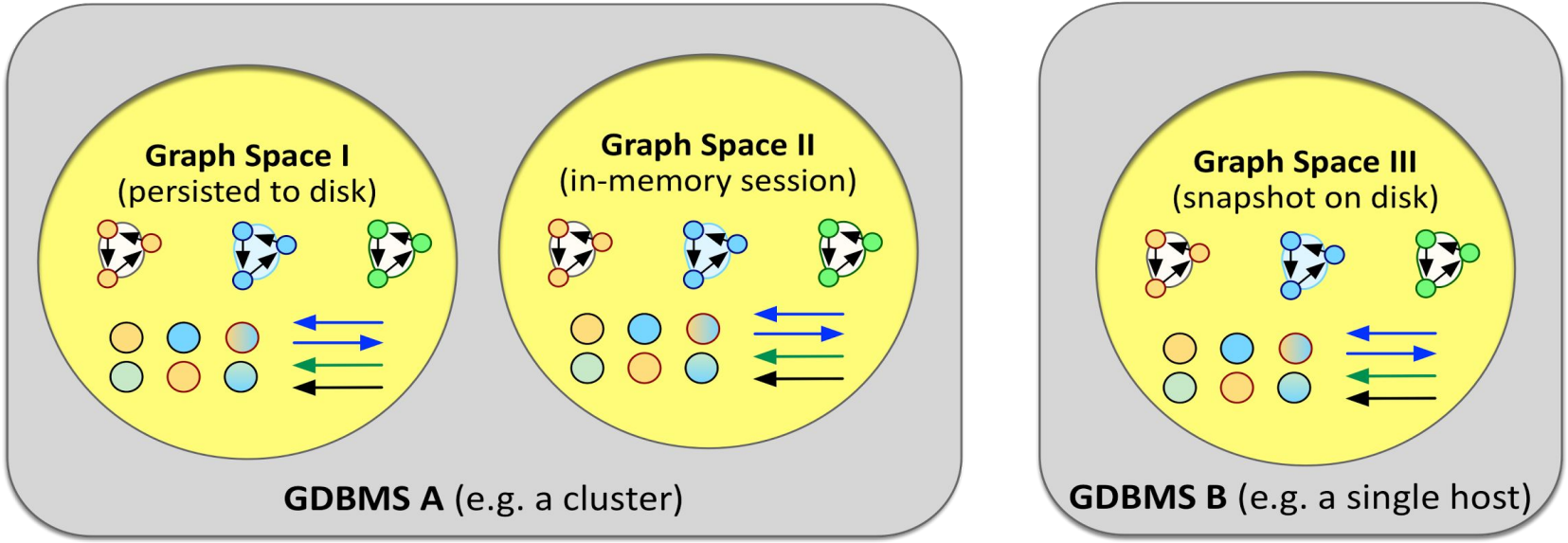
Entity = Node | Relationship



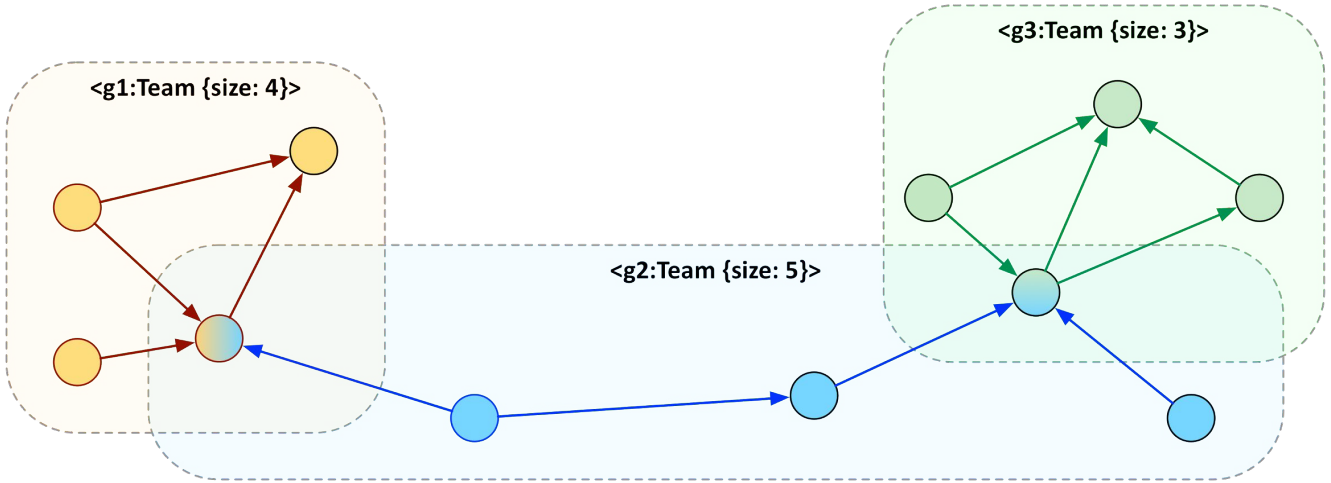
Physical model: Graph Space



Physical model: Multiple Graph Spaces



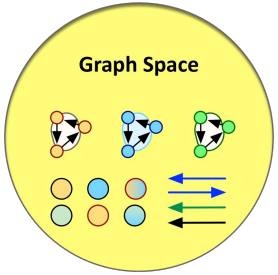
Logical Model: Graphs as entity containers



- Graphs are 1st class entities with
  - Identity
  - Labels
  - Properties

- Graphs may **contain** nodes
- Graphs may **contain** relationships (including start and end nodes)
- Each node or relationship must be **contained** in at least one graph

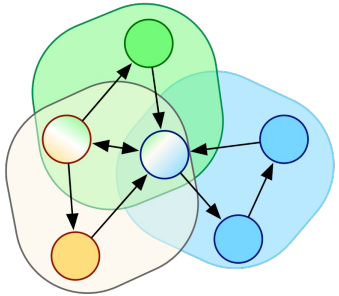
Existence ≠ Containment



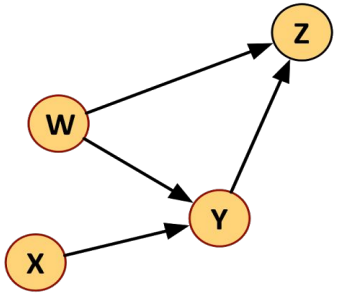
An entity **exists** in a single *associated* graph space.

A node or a relationship **is contained**

in at least one or more graphs.

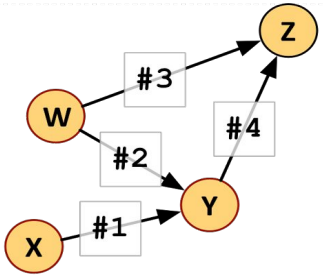


Hello, world of graphs!




```
MATCH (a)-[r]->(b)
RETURN GRAPH
```

Streaming Graphs




```
MATCH (a)-[r]->(b)
RETURN GRAPHS *
```

```
{ a:X, r:#1, b:Y  X -#1-> Y }
{ a:W, r:#2, b:Y  W -#2-> Y }
{ a:W, r:#3, b:Z  W -#3-> Z }
{ a:Y, r:#4, b:Z  Y -#4-> Z }
```

- Stream of **Records**

  ⇒ Stream of **Graph Records ("g-records")**

- **Current Graph Record** `this`

  ≠

  Current Data Graph & `data`

  Associated Graph Space

- Return each match as a graph

```
MATCH (a)-[r]->(b)
RETURN GRAPHS *
```

Key Ideas

- Switch to streaming graphs ("g-records") instead of records
  - Relies on supporting any value as a property in implementations
  - Entity values are references (like in Cypher today)

- Provide easy aggregation of the graph elements in projections
  - **`RETURN GRAPH`** computes the union over all g-records
    (vs. **`RETURN GRAPHS *`** which just returns them individually)
  - Extends naturally to variants using other set operations, like **`INTERSECT`**

- Provide easy nesting and unnesting in projections
  - Nesting Graphs are stored as properties of the g-record
  - Unnesting Replace g-record from graph stored as a property

Graph Patterns

```
1.  RETURN GRAPH <"graph://icij.org/panama">
2.  RETURN GRAPH <"graph://wikipedia.org/edits" { lang: "de" }>

3.  MATCH <g:Autobahn:Road> WHERE g.limit > 120
    RETURN GRAPHS FROM g

4.  IN GRAPH <"graph://my-app.com/my-graph1">
    MATCH (a)-[r]->(b)
    RETURN GRAPH
```

Querying Graphs

*Default **data** graph is provided by session*

```
1.  MATCH (a:Person)-[:KNOWS*3]-(c:Person)
    WHERE EXISTS IN GRAPH <"graph://my-app.com/my-graph2"> {
        (a)-[:ALUMNI_OF]->(:University)<-[:ALUMNI_OF]-(c)
    }
    RETURN GRAPH *
```

*Matching 0..n multiple graphs from **data***

```
2.  MATCH <g:Industry>
    WHERE EXISTS IN GRAPH <"graph://ipcc/studies/resources"> {
        MATCH (r:Resource)-[l:MEASURED]->(:Study) WHERE l.scarce = 'very'
        MATCH IN GRAPH g (r)-[:DEPENDS_ON*]->(:Manufacturer)
    }
    RETURN GRAPHS FROM g
```

*Replace g-record with **g***

Set Operations

1.
```
MATCH <g1:Network {country: "DE"}> RETURN GRAPHS *
INTERSECT GRAPHS
MATCH <g2:Network {country: "SE"}> RETURN GRAPHS *
```

2.
```
MATCH {
    MATCH <g_today "graph://.../2017-Feb-08"> RETURN GRAPH
    EXCEPT GRAPHS
    MATCH <g_yesterday "graph://.../2017-Feb-07"> RETURN GRAPH
}
RETURN GRAPH
```

3. Additional Set Operations `UNION, EXCLUSIVE UNION, ...`

4. Value-level Set Operations `RETURN g_today - g_yesterday`

Querying Inline Views

```
IN GRAPH {
    MATCH UNIQUE NODES (a)-[:KNOWS]-(b)-[:KNOWS]-(c)-[:KNOWS]-(a)
    RETURN GRAPHS
    UNION
    MATCH UNIQUE NODES (a:Java)-[:KNOWS]-(b:Scala)-[:KNOWS]-(c:Java)
    WHERE a.city = b.city AND b.city = c.city
    RETURN GRAPHS
}
MATCH (a:Scala)-[:KNOWS]-(b)
RETURN a, count(b) AS deg ORDER BY deg ASC
```

Updating Graphs

- Creating and deleting graphs
- Setting and removing properties and labels
- Adding and removing nodes and relationships to and from graphs
- Merging into graphs according to some uniqueness constraint
- Copying graphs

```
MATCH (a)-[:KNOWS]->(b)
CREATE GRAPH <g'> {
    CREATE (p:Person {name: a.first})
    MERGE (p)-[:KNOWS]->({total: sum(b)})
}
```

Further Steps

- Actively exploring
  - Alternative approaches (**CIR-2017-182**)
  - Relationship to type theory (Graph pattern types)
  - Relationship to stream processing (StreamSQL, Borealis)
- Generalize the notion of **data graph** vs **this graph** to multiple graphs
- Implementation

Summary

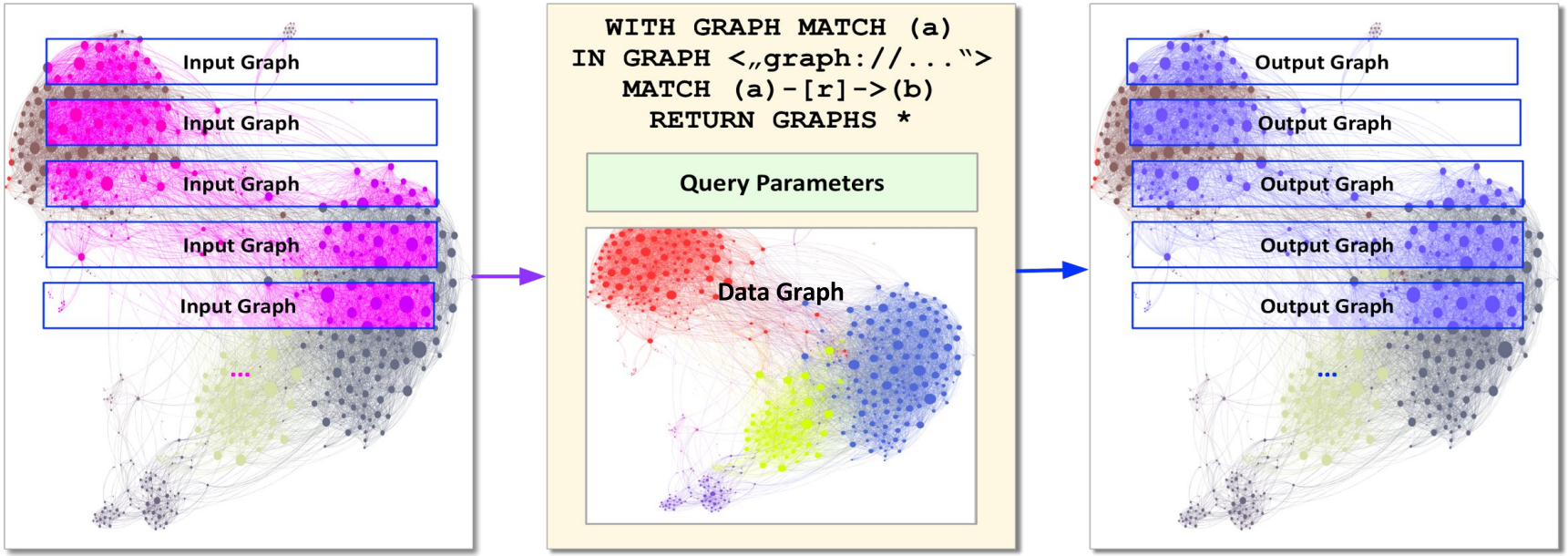
Multiple graphs are the next evolutionary step for Cypher

- **Extends the property graph model** while preserving its original characteristics
  - Ease of use
  - *Schema optionality*
  - *Data integration*
- **Enables** analytics, views, graph management, modeling, visualization, *graph streams*
- **Great fit** builds on Cypher's human-readable, *application-oriented* language design

Waiting to be explored federation, views, syntax alternatives, lifetime, snapshots, planning, ...

CIR-2017-182

Cypher 2018: Let's build the world of multiple graphs!



CIR-2017-182

Thank you

http://opencypher.org

https://github.com/opencypher/openCypher