LDBC The graph & RDF benchmark reference

G-CORE: A Core for Future Graph Query Languages

Designed by the LDBC Graph Query Language Task Force

Hannes Voigt

hannes.voigt@tu-dresden.de

http://bit.ly/gcorelanguage · @LDBCouncil

My Background

- Ph.D. from TU Dresden
- Postdoc at Database System Group, TU Dresden

- Research in Graph Data Management, Schema Evolution, DBMS Schema Flexibility, Adaptive Indexing
- Member of LDBC Graph Query Language Taskforce
- Graph-related industry activities with SAP HANA Graph and openCypher

openCypher

Linked Data Benchmark Council (LDBC)

- LDBC is a non-profit organization dedicated to establishing benchmarks, benchmark practices and benchmark results for graph data management software.
- LDBC was established as an outcome of the LDBC EU project funded by the European Commission within the 7th Framework Programme (Grant Agreement No. 317548).

LDBC The graph & RDF benchmark reference

http://ldbcouncil.org/

LDBC Graph Query Language Task Force

- Recommend a query language core that could be incorporated in future versions of industrial graph query languages.
- Perform deep academic analysis of the expressiveness and complexity of evaluation of the query language
- Ensure a powerful yet practical query language

Academia	Industry
Renzo Angles, Universidad de Talca	Alastair Green, Neo4j
Marcelo Arenas, PUC Chile	Tobias Lindaaker, Neo4j
Pablo Barceló, Universidad de Chile	Marcus Paradies, SAP
Peter Boncz, CWI	Stefan Plantikow, Neo4j
George Fletcher, Eindhoven University of Technology	Arnau Prat, Sparsity
Claudio Gutierrez, Universidad de Chile	Juan Sequeda, Capsenta
Hannes Voigt, TU Dresden	Oskar van Rest, Oracle

LDBC

Result

G-CORE
A Core for Future Graph Query Languages

Designed by the LDBC Graph Query Language Task Force*

Renzo Angles
Universidad de Talca

Marcelo Arenas
PUC Chile

Pablo Barceló
DCC, Universidad de Chile

Peter Boncz
CWI, Amsterdam

George Fletcher
Technische Universiteit Eindhoven

Claudio Gutierrez
DCC, Universidad de Chile

Tobias Lindaaker
Neo4j

Marcus Paradies
SAP SE

Stefan Plantikow
Neo4j

Juan Sequeda
Capsenta

Oskar van Rest
Oracle

Hannes Voigt
Technische Universität Dresden

SIGMOD 2018

Preprint: http://bit.ly/gcorelanguage

LDBC

DISCLAIMER

WHAT IS NOT G-CORE

Graph Database System

Commercial/Proprietary

Another Standard

WHAT IS G-CORE

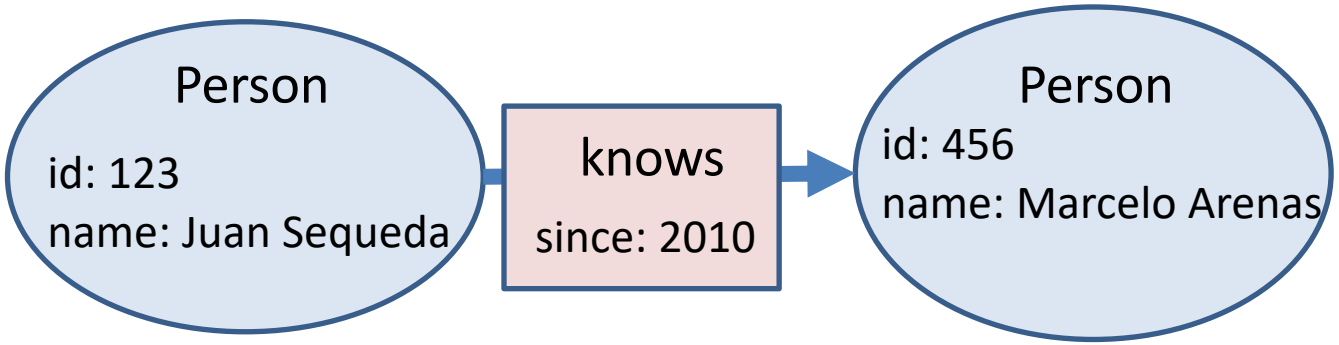
Graph Query Language

Designed by the LDBC Graph Query Language Task Force, consisting of members from industry and academia, intending to bring the best of both worlds to graph practitioners

Proposal which we hope will guide the evolution of both existing and future graph query languages, towards making them more useful, powerful and expressive

LDBC

Property Graph Data Model



Challenges and G-CORE Principles

- Composability →
  - Query results can be query inputs
  - Important for views & sub-queries
- Closed QL
  - Graphs as query input and output
  - Queries can be composed

Challenges and G-CORE Principles

- Composability →
  - Query results can be query inputs
  - Important for views & sub-queries
- Paths →
  - Fundamental to graphs
  - Increase the expressivity of the language

- Closed QL
  - Graphs as query input and output
  - Queries can be composed
- Paths are First Class Citizens
  - Graph model with path objects
  - Paths objects can have labels and properties

Challenges and G-CORE Principles

- Composability →
  - Query results can be query inputs
  - Important for views & sub-queries
- Paths →
  - Fundamental to graphs
  - Increase the expressivity of the language
- Capture a core →
  - Standards are difficult and politics
  - Foundation to develop next generation of languages

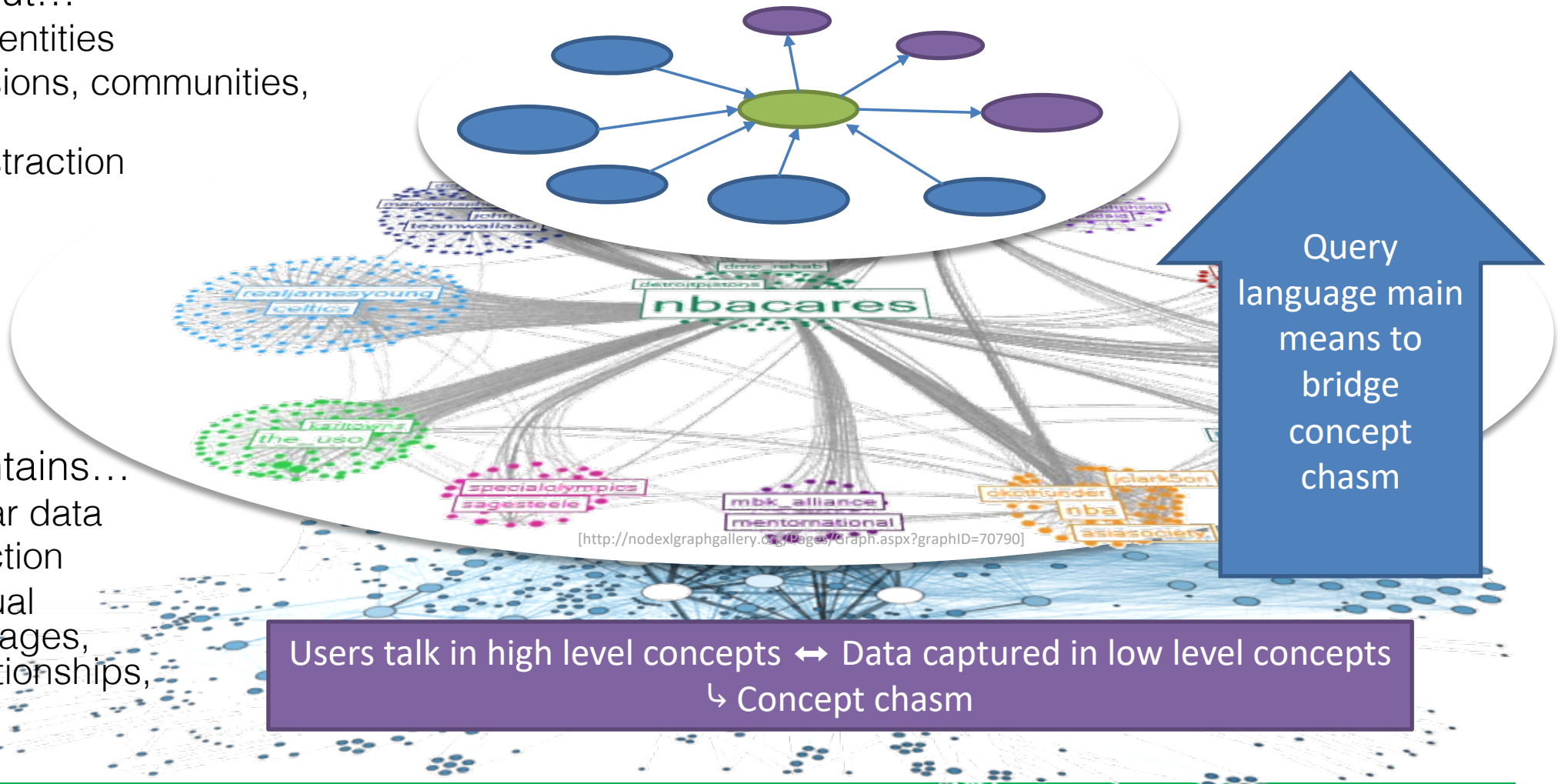
- Closed QL
  - Graphs as query input and output
  - Queries can be composed
- Paths are First Class Citizens
  - Graph model with path objects
  - Paths objects can have labels and properties
- Efficient Evaluation
  - Only features with tractable evaluation (in data complexity)
  - connects practical work with the foundational research

Closed QL and Query Composition

LDBC

Concept Chasm

- Users talk about…
  - Application entities
  - e.g. discussions, communities, topics, etc.
  - Multiple abstraction levels

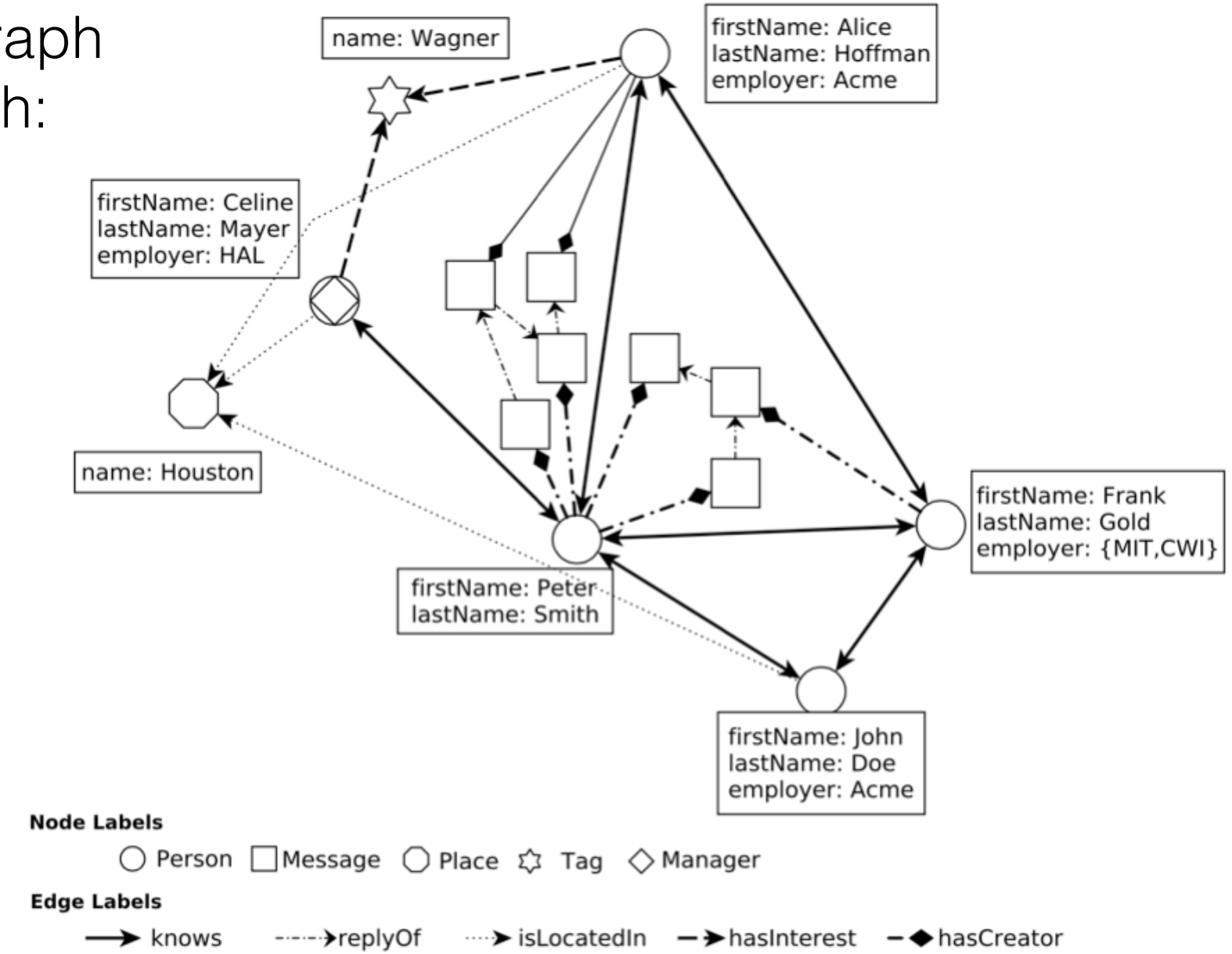


- Base data contains…
  - Fine granular data
  - Low abstraction
  - E.g. individual twitter messages, retweet relationships, etc.

LDBC

[Martin Grandjean, https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png, 2014]

Example Graph social_graph:



Always returning a graph

```
CONSTRUCT (n)
MATCH (n:Person) ON social_graph
WHERE n.employer = 'Acme'
```

Syntax inspired by Neo4j's Cypher and Oracle's PGQL

- **CONSTRUCT** clause: Every query returns a graph
- New graph with only nodes, namely those persons who work at Acme
- All the labels and properties that these person nodes had in `social_graph` are preserved in the returned result graph.

Multi-Graph Queries and Joins

- Simple data integration query

```
CONSTRUCT (c)<-[e:worksAt]-(n)
MATCH (c:Company) ON company_graph,
      (n:Person) ON social_graph
WHERE c.name IN n.employer
UNION social_graph
```

- Load company nodes into `company_graph`
- Create a unified graph (**`UNION`**) where employees and companies are connected with an edge labeled `worksAt`.

Graph Construction

- Normalize Data, turn property values into nodes

```
CONSTRUCT (n)-[y:worksAt]->
          (x:Company {name:=e})
MATCH (n:Person {employer=e}) ON social_graph
```

- The unbound destination node **x** would create a company node for each match result (binding).
- This is not what we want: we want only one company per unique name ... So ...

Graph Aggregation

```
CONSTRUCT (n)-[y:worksAt]->
          (x GROUP e :Company {name:=e})
MATCH (n:Person {employer=e}) ON social_graph
```

- Graph aggregation: **GROUP** clause in each graph pattern element
- Result: One company node for each unique value of e in the binding set is created

Graph Aggregation

```
CONSTRUCT (n)-[y:worksAt]->
   (x GROUP e :Company {name:=e, noEmpl:=COUNT(x)})
MATCH (n:Person {employer=e}) ON social_graph
```

- Graph aggregation: **GROUP** clause in each graph pattern element
- Result: One company node for each unique value of e in the binding set is created

Graph Augmentation

```
CONSTRUCT social_graph,
          (n)-[y:worksAt]->
          (x GROUP e :Company {name:=e})
MATCH (n:Person {employer=e}) ON social_graph
```

- With **CONSTRUCT** `social_graph,` … the query returns the graph with that identifier united with what is newly constructed for the given pattern
- No base data manipulation

Reachability over Paths

- Paths are demarcated with slashes `-/ /-`
- Regular path expression are demarcated with `< >`

```
CONSTRUCT (m)
MATCH (n:Person)-/<:knows+>/->(m:Person)
WHERE n.firstName = 'John' AND n.lastName = 'Doe'
  AND (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
```

- If we return just the node `(m)`, the `<:knows*>` path expression semantics is a reachability test

Existential Subqueries

```
CONSTRUCT (m)
MATCH (n:Person)-/<:knows+>/->(m:Person)
WHERE n.firstName = 'John' AND n.lastName = 'Doe'
  AND (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
```

Syntactical shorthand for existential subquery:

```
WHERE …
      EXISTS (
        CONSTRUCT ()
        MATCH (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
      )
```

LDBC

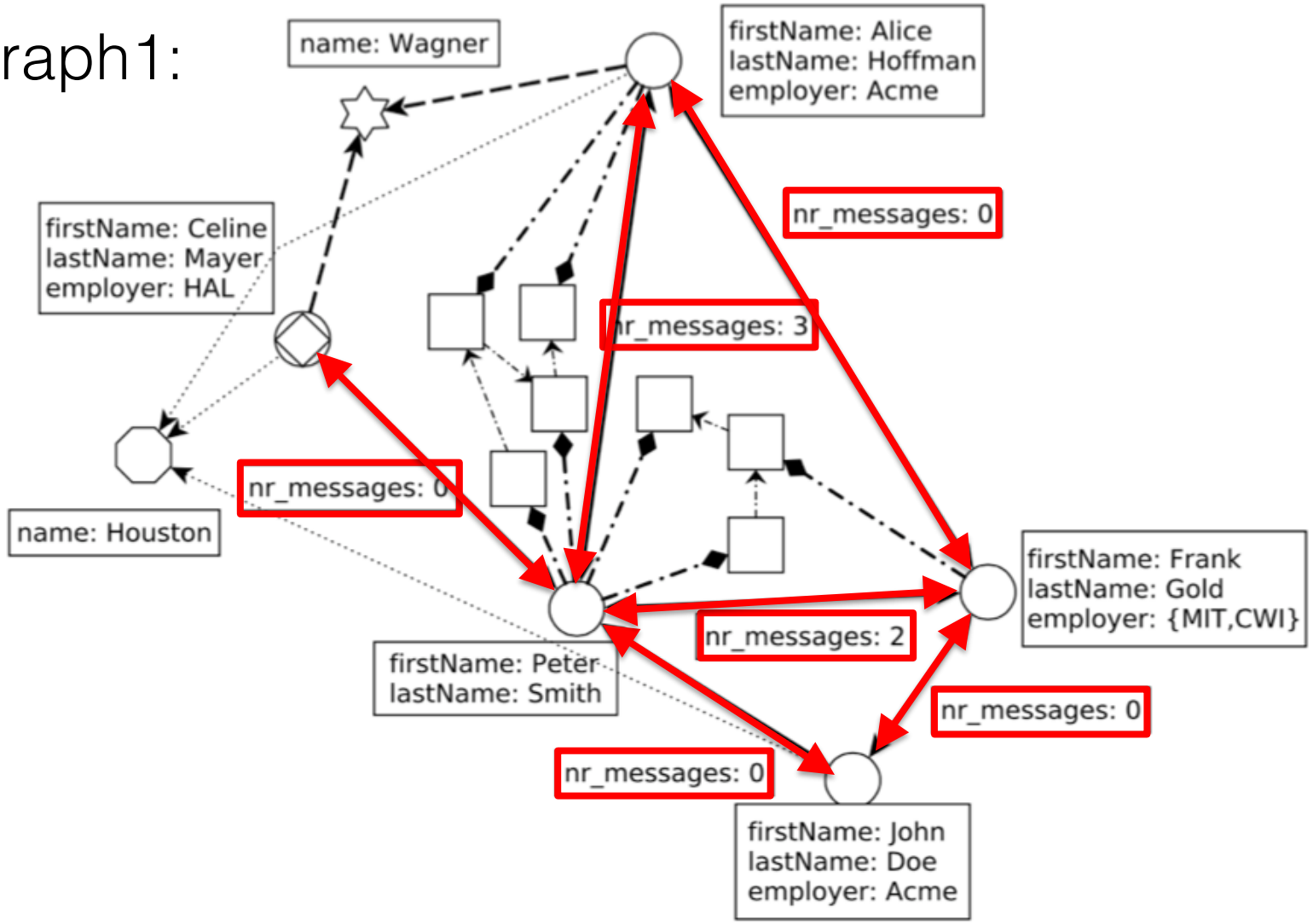
Views and Optionals

```
GRAPH VIEW social_graph1 AS (
  CONSTRUCT social_graph, (n)-[e]->(m)
        SET e.nr_messages := COUNT(*)
  MATCH (n)-[e:knows]->(m)
  WHERE (n:Person) AND (m:Person)
  OPTIONAL (n)<-[c1]-(msg1:Post),
           (msg1)-[:reply_of]-(msg2),
           (msg2:Post)-[c2]->(m)
     WHERE (c1:has_creator) AND (c2:has_creator)
)
```

- The view adds a `nr_messages` property to each `:knows` edge using **SET**
- `nr_messages` property contains the amount of messages that the two persons `n` and `m` have actually exchanged
- **`OPTIONAL`** matches, such that people who know each other but never exchanged a message still get a property `e.nr_messages=0`

LDBC

View social_graph1:



Paths as First Class Citizens

Storing Paths with `@p`

- Paths as first-class citizen
- Save the three shortest paths from John Doe towards other person who lives at his location, reachable over knows edges

```
CONSTRUCT (n)-/@p:localPeople {distance:=c}/->(m)
MATCH (n)-/3 SHORTEST p <:knows*> COST c/->(m)
WHERE n.firstName = 'John' AND n.lastName = 'Doe'
  AND (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
```

- `@` prefix indicates a stored path: query is delivering a graph with paths
- Path has label `:localPeople` and cost as property distance.
- Default cost of a path is its hop-count (length)
- **`COST`** allows to bind the cost value of path to a variable

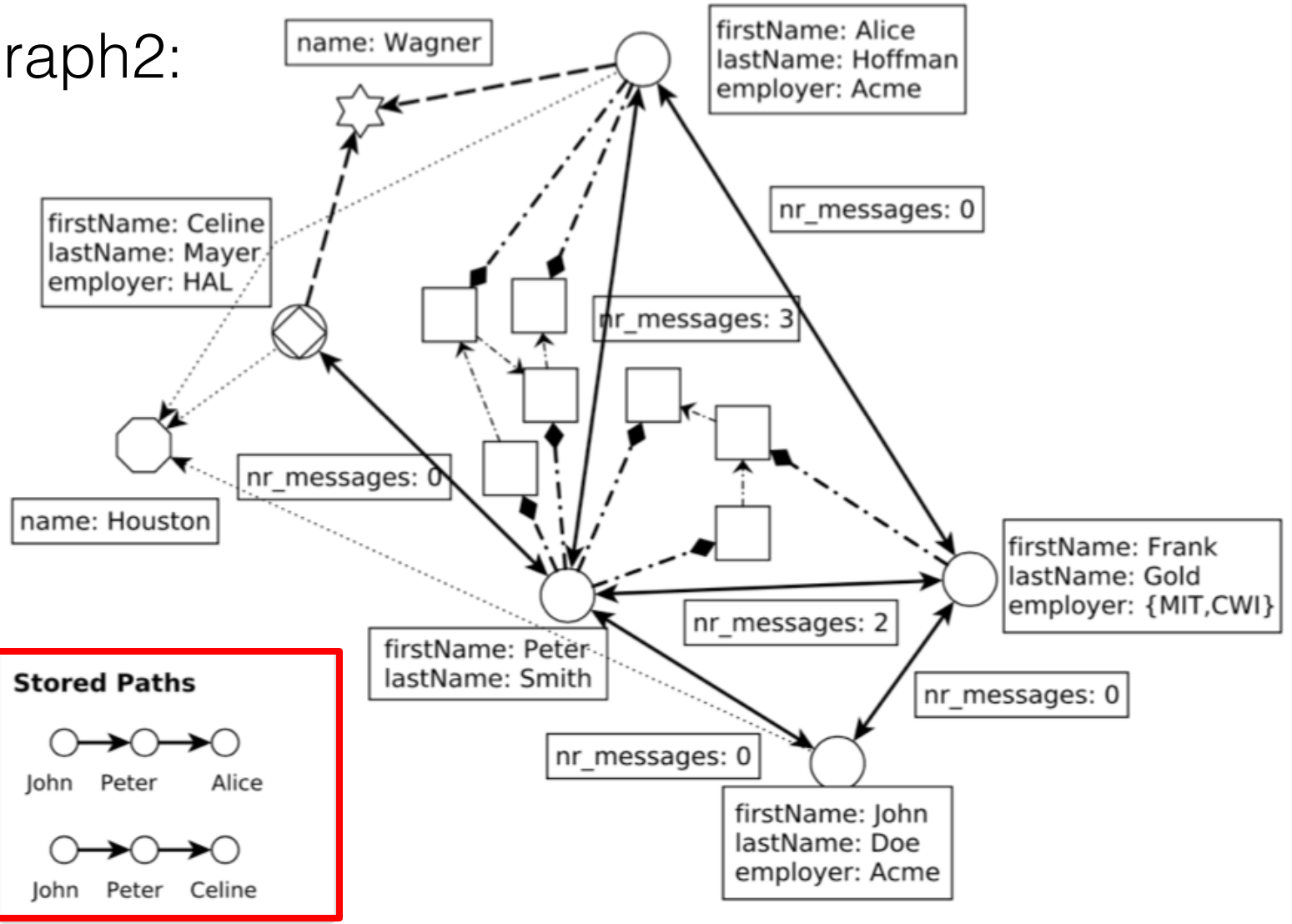
Weighted Shortest Paths

- John Doe wants to go to a Wagner Opera, but none of his friends likes Wagner.
- He thus wants to know which friend to ask to introduce him to a true Wagner lover who lives in his city (or to someone who can recursively introduce him).
- To optimize his chances for success, he prefers to try "friends" who actually communicate with each other.

```
GRAPH VIEW social_graph2 AS (
  PATH wKnows = (x)-[e:knows]->(y)
       WHERE NOT 'Acme' IN y.employer
       COST 1 / (1 + e.nr_messages)
  CONSTRUCT social_graph1, (n)-/@p:toWagner/->(m)
  MATCH (n:Person)-/p <~wKnows*>/->(m:Person) ON social_graph1
  WHERE (m)-[:hasInterest]->(:Tag {name='Wagner'})
    AND (n)-[:isLocatedIn]->()<-[:isLocatedIn]-(m)
    AND n.firstName = 'John' AND n.lastName = 'Doe'
)
```

- We look for the weighted shortest path over the `wKnows` ("weighted knows") path pattern towards people who like Wagner

View social_graph2:



Future Extensions of G-CORE

- Projecting tabular results
- Importing tabular data
- Binding table inputs
- Interpreting tables as graphs

Takeaway

- GCORE: A Closed, Composable, and Tractable Graph Query Language with Paths as First-Class Citizens
  - This work is the culmination of 2.5 years of intensive discussion between the LDBC Graph Query Language Task Force and members of industry and academia: Capsenta, HP, Huawei, IBM, Neo4j, Oracle, SAP and Sparsity.
  - We also thank the following people for their participation: Alex Averbuch, Hassan Chafi, Irini Fundulaki, Alastair Green, Josep Lluis Larriba Pey, Jan Michels, Raquel Pau, Arnau Prat, Tomer Sagi and Yinglong Xia
  - Tentative: LDBC Technical User Community Meeting in Austin June 8
  - http://bit.ly/gcorelanguage
  - https://github.com/ldbc/ldbc_gcore_parser