# Query languages 2 (NDBI006) Graph Databases

J. Pokorný
MFF UK

# Content

- Introduction
- Graph database technology
- Categories of graph databases
- Limitations of graphs databases
- Modelling graph databases
- Conclusions
- References

# Introduction

- Distinguished characteristic of the domain:
    - relationship-rich data
    - relationships are first-class citizens in graph databases
- Graph databases are focused on:
    - efficiently store and query highly connected data
- Two basic types of graph data stores:
    - one graph
    - collections of graphs
- Application areas: geospatial processing, social networks analysis, biology systems, traffic networks, healthcare, retail, semantic associations, etc.
- Our examples: mostly inspired by Neo4j[1]

[1] http://neo4j.com

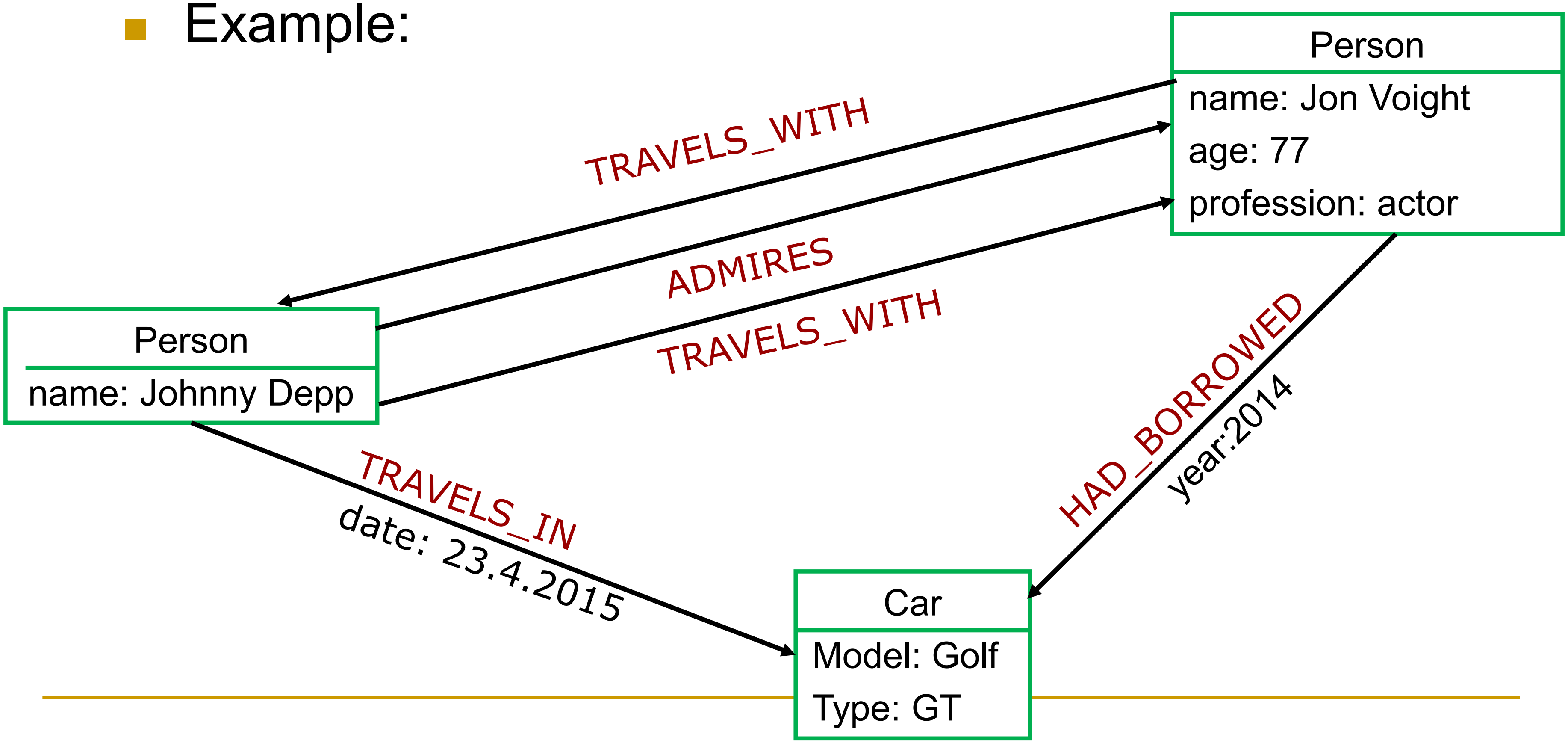
# Graph data model

- (labelled) property graph model
  - entities (nodes)
  - properties (attributes)
  - labels (types)
  - relationships (edges)
    - direction,
    - start node,
    - end node
  - identifiers

Entities and relationships can hold any number of properties, nodes and edges can be tagged with labels. Both nodes and edges are defined by a unique identifier.

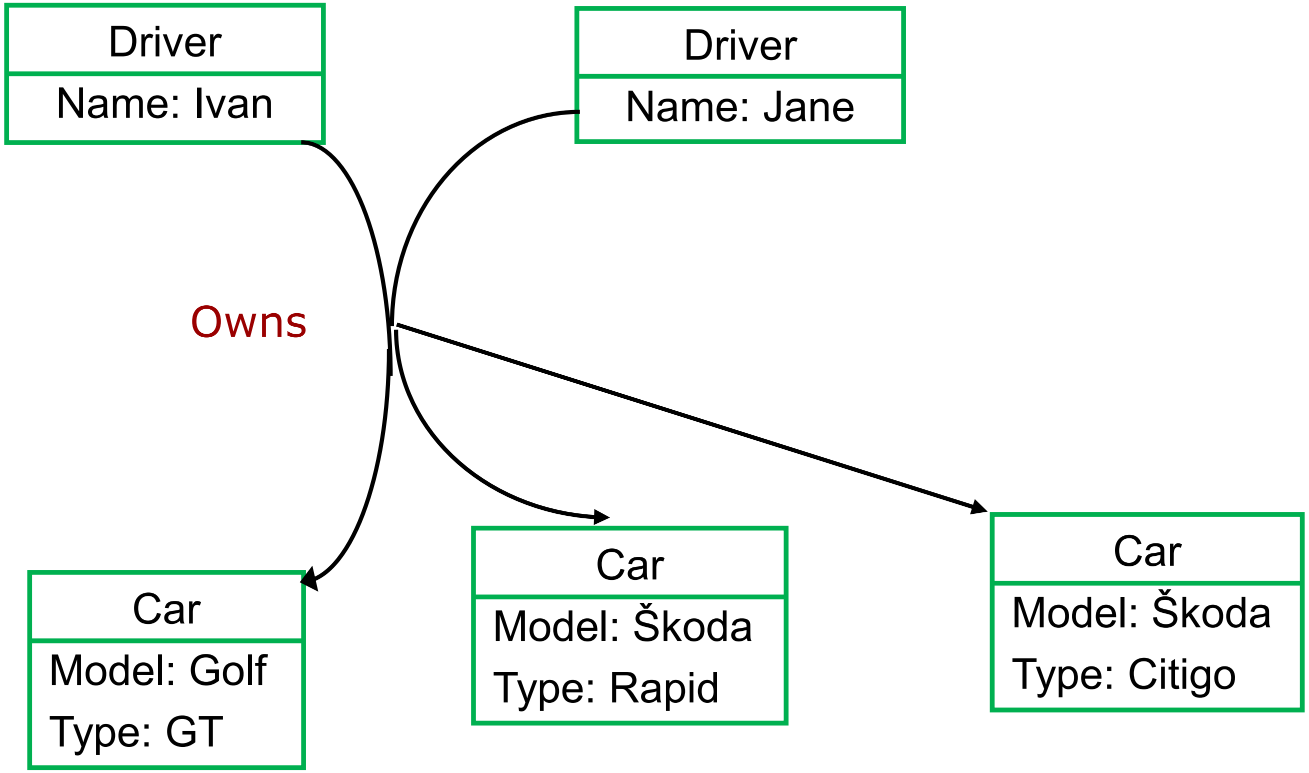
# Graph data model

- In graph-theoretic notions:
  labelled and directed attributed multigraphs
  - Example:

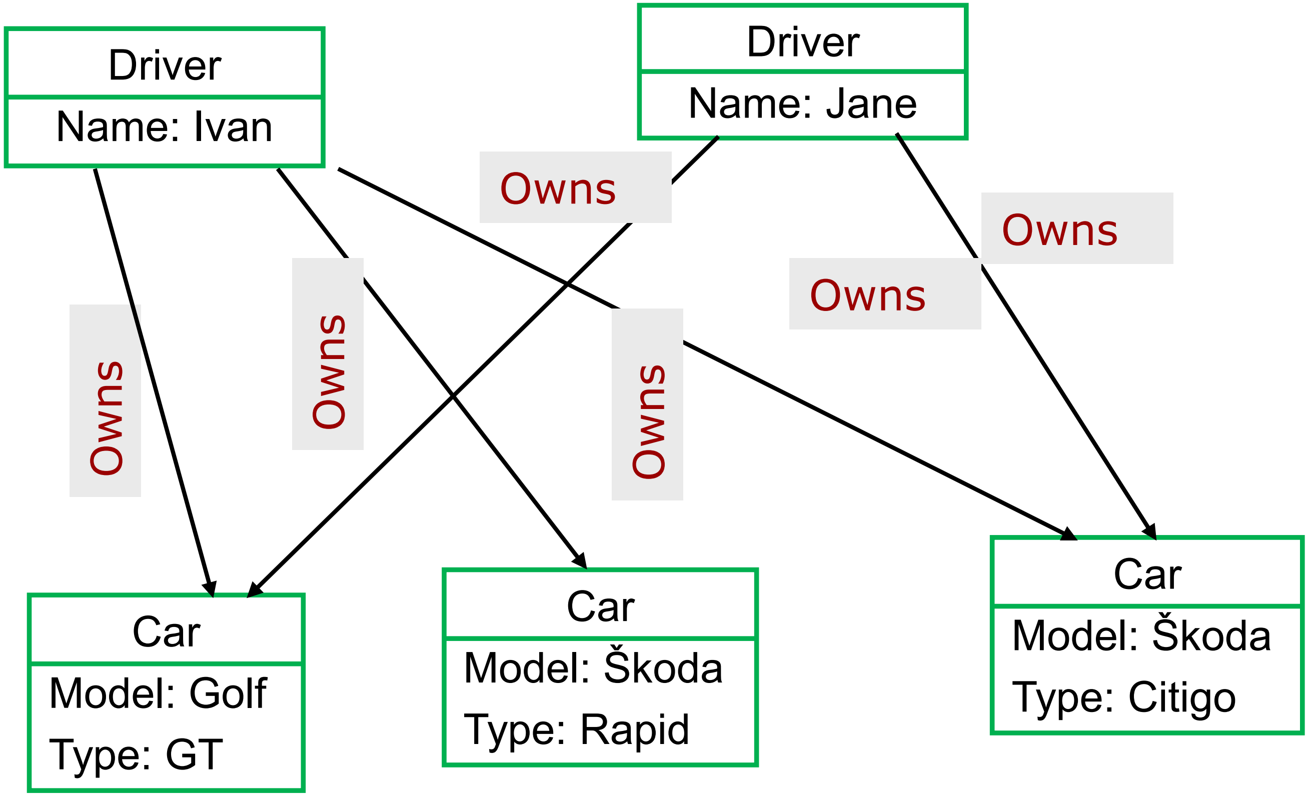


# Graph data model

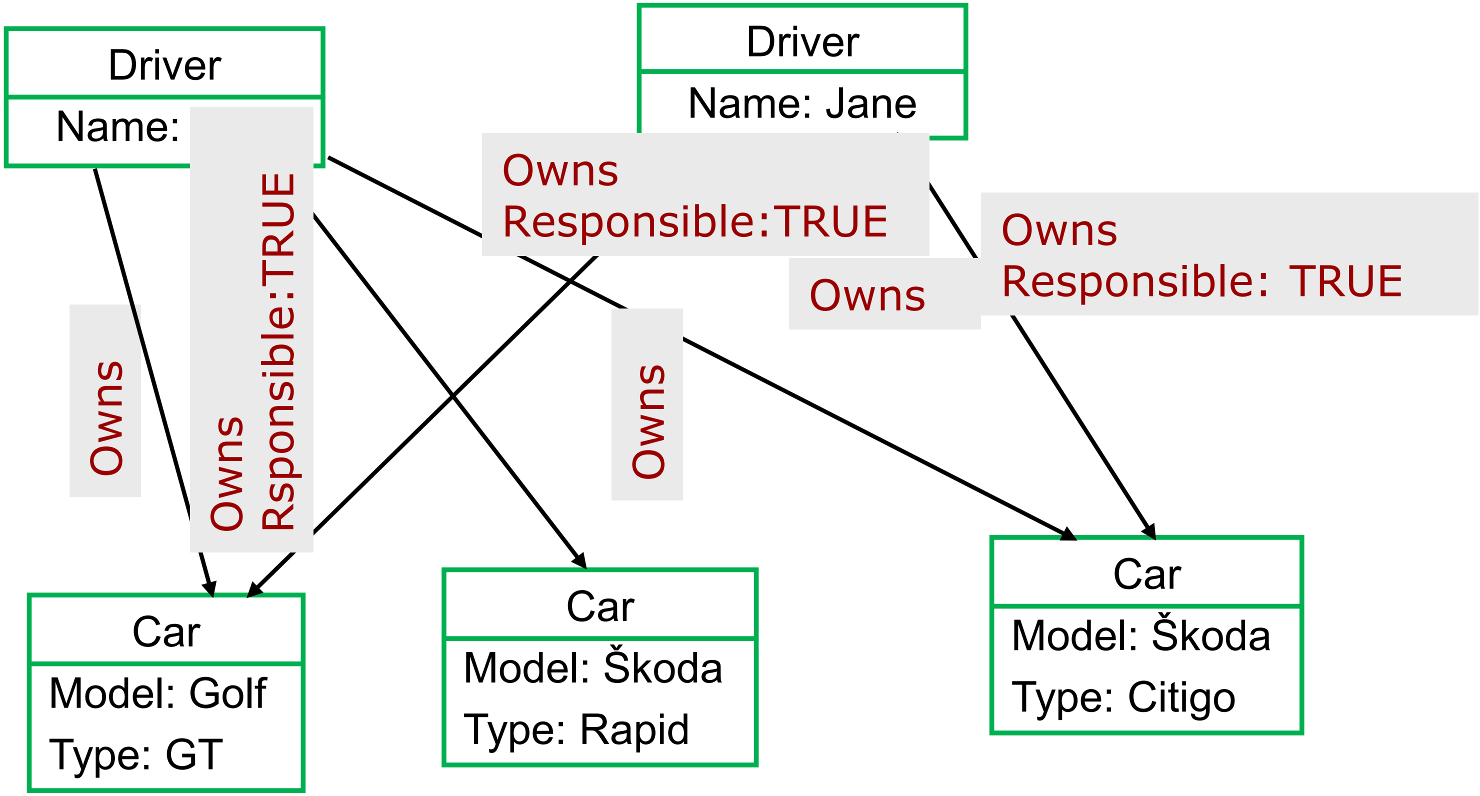
- hypergraphs
  - hyperedge connects an arbitrary set of nodes



# Equivalent multigraph



# Multigraph with more semantics



# More formally

Definition: Database graph $G = (V, E, N, \Sigma, \varphi, \lambda, A, Att)$ is labelled and directed attribute multigraph, where $V$ is finite set of nodes with identifiers drawn from an infinite alphabet $N$, $E$ is a set of edges and $\varphi$ is an incidence function mapping from $E$ into $V \times V$. Edge labels are drawn from the finite set of symbols $\Sigma$, $\lambda$ is a function from $E$ into $\Sigma$ labelling edges. $A$ is a set of attributes (properties) represented by couples $(A_i, value_{ij})$. $Att$ is a mapping assigning to each node/edge a subset (event. empty) of attributes from $A$. Identifiers of nodes are called also labels (node labels).

Note: The definition accepts database graphs with different attribute sets for nodes/edges of the same types. It occurs in practice, especially in GDBMS without schema. Often attribute domains are defined. Then $value_{ij} \in dom(A_i)$.

# Why graph databases?

- Traditional solutions:
  - Relational databases – SQL with joins
  - Relational databases – SQL with Common Table Expressions
    - relatively simple for trees and acyclic graphs
    - more complicated in cyclic graphs
  - Datalog – is able, e.g., to cover conjunctive regular path queries
    - less implementations, appropriate rather for small graphs
    - Trend: `renaissance' for Datalog (e.g. DATOMIC[2] – distributed DBMS with ACID, joins, …)
  - XML databases
    - require XML data model for graphs

[2] http://www.datomic.com/

# Why graph databases?

- Graph databases are focused on:
  - be flexible in usage data models behind graphs used,
  - exceptional performances for local reads, by traversing the graph.
- Graph databases are often included among NoSQL databases
- Trend: graph databases + graph-based analytics on
  - Big Graphs
  - large, unstructured datasets

# Graph database technology

- Graph storage
- Graph querying
- Scalability
- Transaction processing

- Terminology:
  - Graph Database Management Systems (GDBMS)
  - Graph Databases (GDB) /* rather for OLTP */
  - graph processing tools /* rather for OLAP*/
  - today: notions GDBMS and „graph database“ are used interchangeable (see the NoSQL world)

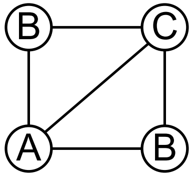
# Graph storage

- index-free adjacency: every node is directly linked to its neighbour node
  - appropriate for local graph queries
- data structures:
  - lists
  - bitmaps (+compression)
- graph-based indices:
  - indexing methods: path-based, graph based, tree-based
  - strategy for query processing: filtering-verification fashion
- non-native solutions:
  - column store in Virtuoso Universal Server
  - other DBMS as back-end storage, e.g., MySQL in FlockDB (stores adjacency lists)

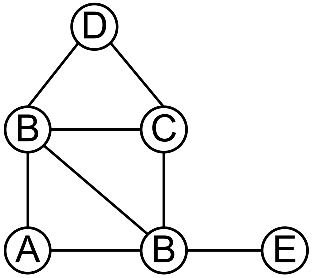
# Graph-based indices

Example (Sasha et al, 2002):

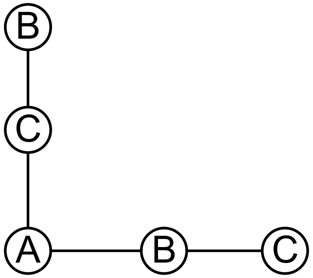
- assumption: undirected graphs, no edge labels
- index based on paths
  - Do enumeration of all paths of the length <= L of all graphs in DB,
  - For each path store the number of its occurrences in all graphs in DB into the hash table.



$g_1$



$g_2$

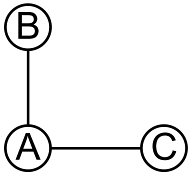


$g_3$

| key | $g_1$ | $g_2$ | $g_3$ |
|---|---|---|---|
| h(CA) | 1 | 0 | 1 |
| … | | | |
| h(ABCB) | 2 | 2 | 0 |

# Graph-based indices

- Query:



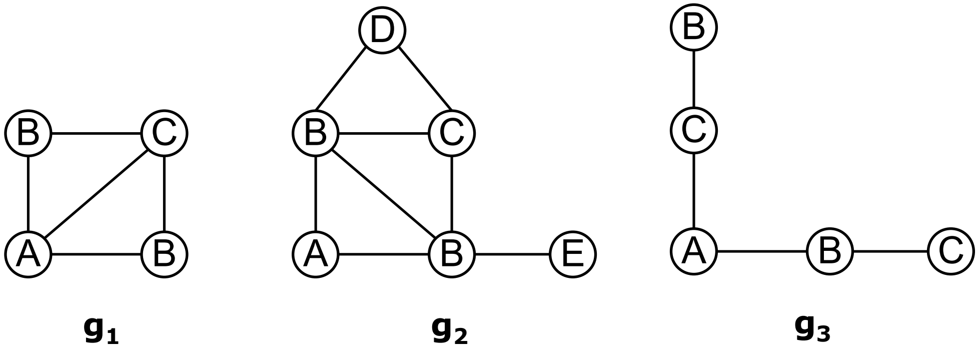
AB:1
AC:1
BAC:1

- Find candidates with index,
- eliminate graphs, where the number of occurrences of a path is smaller than in the query graph,
- execute and verify, i.e. check isomorphism.

⇨ Candidates = {g1, g3}

⇩

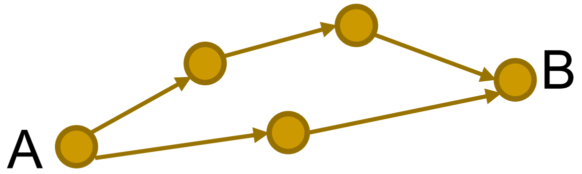
verification



| key | $g_1$ | $g_2$ | $g_3$ |
|---|---|---|---|
| h(CA) | 1 | 0 | 1 |
| … | | | |
| h(ABCB) | 2 | 2 | 0 |

# Graph querying

- Query capabilities come from the associated graph model
- Types of queries:
  - k-hop queries



2-hop distance between A and B

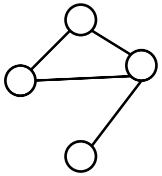
  - point querying - looking for a node based on its properties or through its identifier
  - finding tuples of points – nodes connected by paths
    - conjunctive queries with regular path(s)
  - subgraph and supergraph queries
  - breadth-first/depth-first search,
  - path and shortest path finding,

# Graph querying

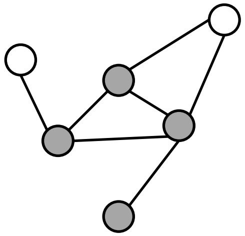
- least-cost path (see algorithm Dijkstra, A*)
- finding cliques or dense subgraphs,
- finding strong connected components
- tree pattern queries

- Other types of queries:
  - approximate matching in Big Graphs
  - structural similarity queries

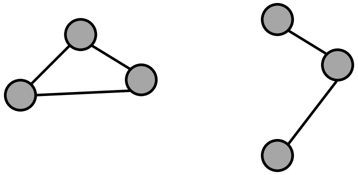
# Graph querying



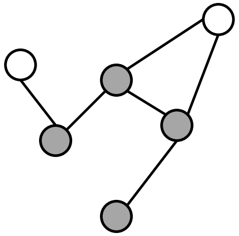
guery graph *q*



q as a subgraph
(exact) match



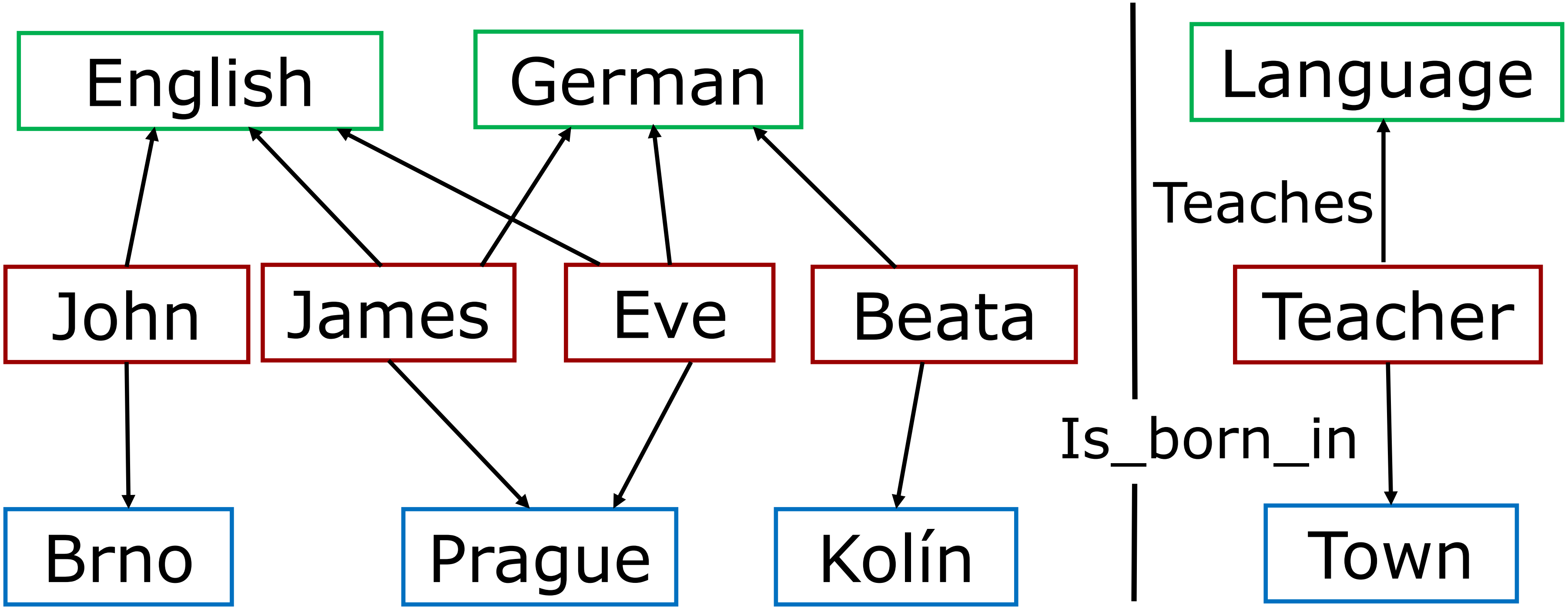
*q* as a supergraph query



*q* as a similarity query
(requires a similarity measure)

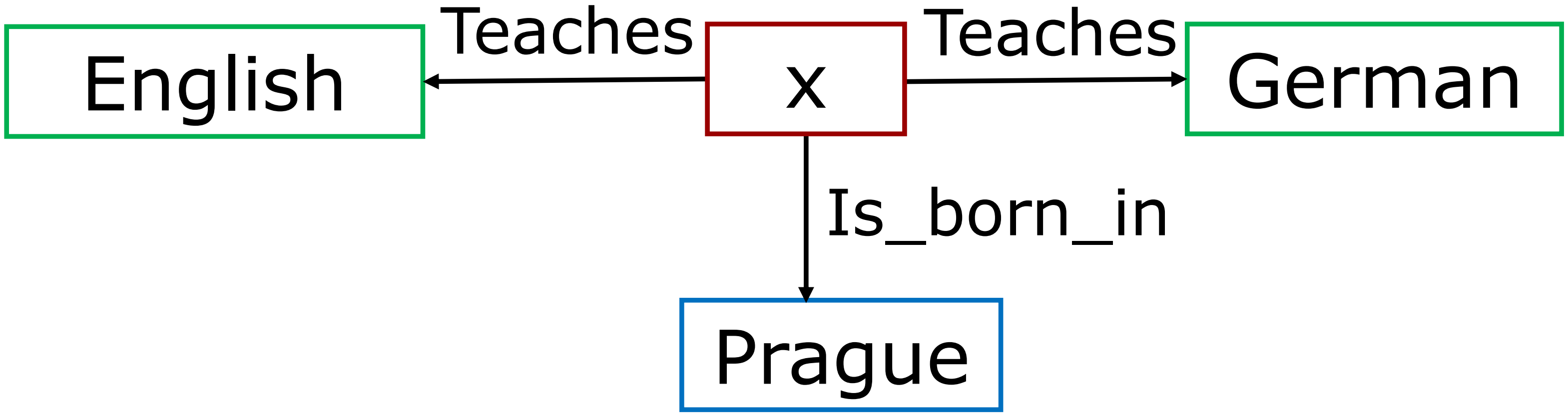
# Example

Teachers teach languages, teachers are born in towns



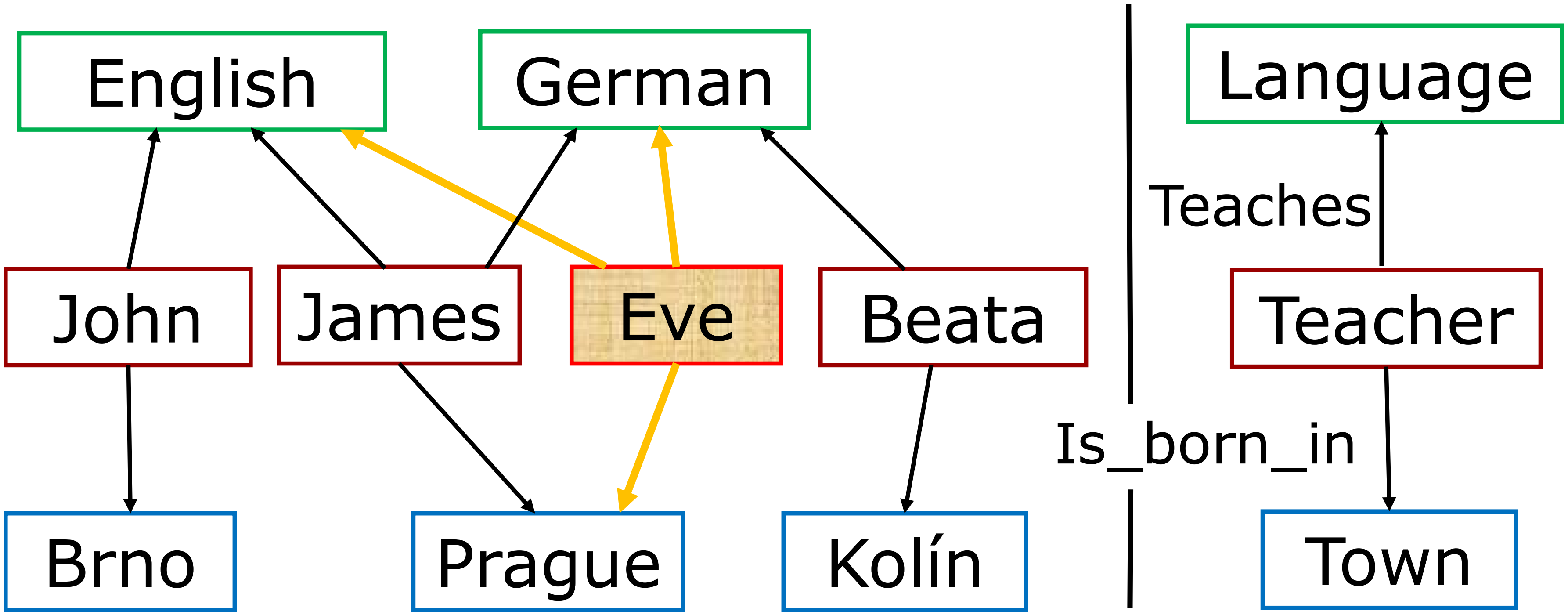
# Example

Query: Find teachers born in Prague who teach English and German.

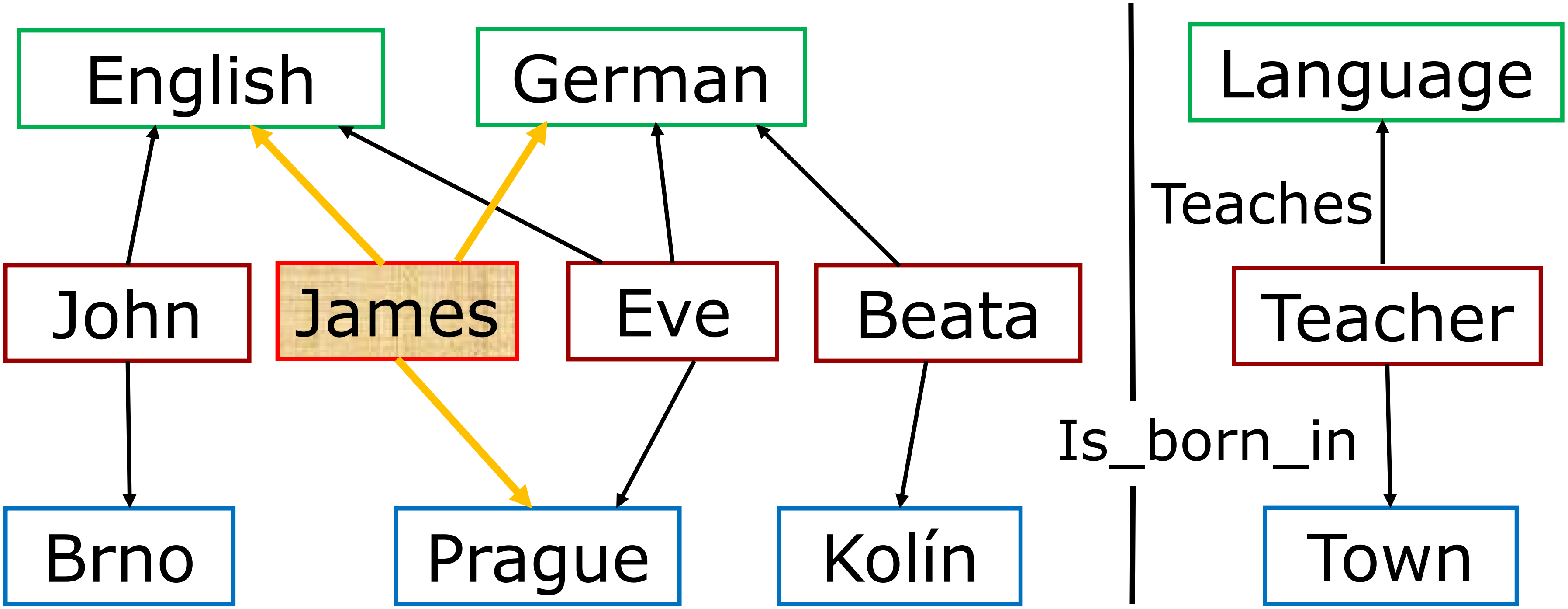


where x is a variable.
This is a language with graph patterns (e.g., G, GraphLog)

# Matching in two subgraphs



# Matching in two subgraphs



# Graph querying

Let Σ be an alphabet of edge labels.

- A conjunctive query over Σ is an expression of the form

$$Q(z_1,\ldots z_n) \leftarrow (x_1, a_1, y_1),\ldots,(x_m, a_m, y_m),\ m \geq 1$$

where $x_i$ and $y_i$ are node variables or constants, $a_i \in \Sigma$, $z_i$ is an $x_j$ or $y_j$,

- A conjunctive regular path query over Σ is an expression of the form

$$Q(z_1,\ldots,z_n) \leftarrow (x_1, r_1, y_1),\ldots,(x_m, r_m, y_m),\ m \geq 1,$$

where $r_i$ is a regular expression over Σ and $z_i$ is an $x_j$ or $y_j$, $1 \leq j \leq m$.

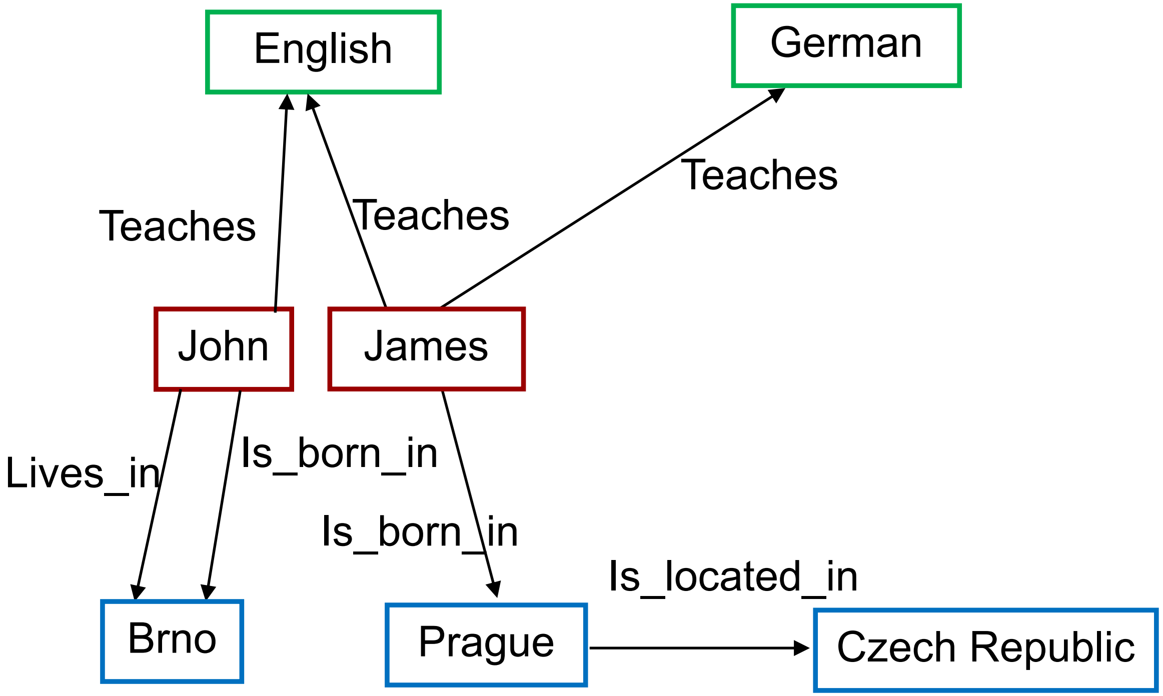
Ex.: node types: Teacher, Language, Town;

edge types: Teaches, Is_born_in, Has_a_nationality, Lives_in, and Is_located_in

# Graph querying

Query: Find teachers teaching English and German and places associated to them

$$Q(x, y) \leftarrow (x, \text{Teaches}, \text{German}), (x, \text{Teaches}, \text{English}),$$
$$(x, \text{Has\_a\_nationality} \mid ((\text{Is\_born\_in} \mid \text{Lives\_in}).\text{Is\_located\_in}^{*}), y)$$



# Graph querying

- A extended conjunctive regular path query over Σ

Example: Find x and y, where the path from y to x is the same as from y to Rohnovi?

$ans(x, y) \leftarrow (James, \pi, y), (x, \pi, y), (\Sigma^*\pi)$

where π is a path variable, Σ* denotes any sequence of edge labels.

Note: usable in RDF for comparing of semantic associations. This is the relationship between the paths.

# Graph querying

- A extended conjunctive regular path query over $\Sigma$

Example: Find x and y, where the path from y to x is the same as from y to Rohn?

$\mathrm{ans}(x, y) \leftarrow (\mathrm{James}, \pi, y), (x, \pi, y), (\Sigma^*\pi)$

where $\pi$ is a path variable, $\Sigma^*$ denotes any sequence of edge labels.

path labeling is from $E^*$

Note: usable in RDF for comparing of semantic associations. This is the relationship between the paths.

# Graph querying

- Powerful query languages

  Examples:

  - Cypher (in Neo4j GDBMS) – declarative, inspired by the SQL
  - Gremlin – rather procedural graph Language
- Issues:
  - often iterative computation for complex queries – not easy, e.g., with Map Reduce
    - Subgraph isomorphism problem is NP-complete
    - Maximum common subgraph problem is NP-complete;
    - Maximum common subgraph isomorphism problem is NP-hard
    - Evaluation of query with simple regular path is NP-Complete Problem
  - complex calculation of the query result – a transformation of the original graph
  - graph visualization

# Example

- Neo4j
  - data: nodes and relationships + identifiers + properties in a key-value form,
  - values: primitive or an array of one primitive type,
  - nodes cannot reference themselves directly
  - graph processing: mostly random data access
  - high-level query language Cypher
  - advanced query features: built-in graph algorithms
    - shortestPath
    - allSimplePaths
    - allPaths
    - dijkstra (optionally with cost_property and default_cost parameters)

# Example

Inspired by SQL:

CREATE: creates a node, a relationship; e.g.

- Example: CREATE (n: Person {name: 'Jane'}) // Person is a label, n is a variable for the new node

identifiers - names assigned to parts of a graph: n, A, r, name, person

Note: it is actually a variable

START: (optional) entry points in a graph (by index or ID) for a graph pattern

- Ex.: START n = node (*), START n = node (3),
  START n = node: Person (name = ‘Jane')

querying
by example

MATCH: graph pattern, bound to the START entry points, is specified by one or more paths separated by ","

Notation: nodes: (a) or () (anonymous nodes)
edges - ->, <- -, - -

# Example

- related nodes (without direction) pomocí - -
- related nodes entering to (coming out) with  - - > (< - -)
- relationships entering  to (coming out) with a variable, or by the name of the relationship (labeled edge), e.g.,

```
(a) - [r] -> (b)
(a) - [:friend] -> (b)
(a) - [*] -> (b),                  /* path lenghts is arbitrary*/
(a) - [*1..4] -> (b),              /* path lenghts is max. 4 */
(a) - [:is_father_of | playing_in*1..2] -> (b),
(a)-->()-->(b)
```

joint paths, e.g.,

```
(a)- - > (b) - - > (c),
```

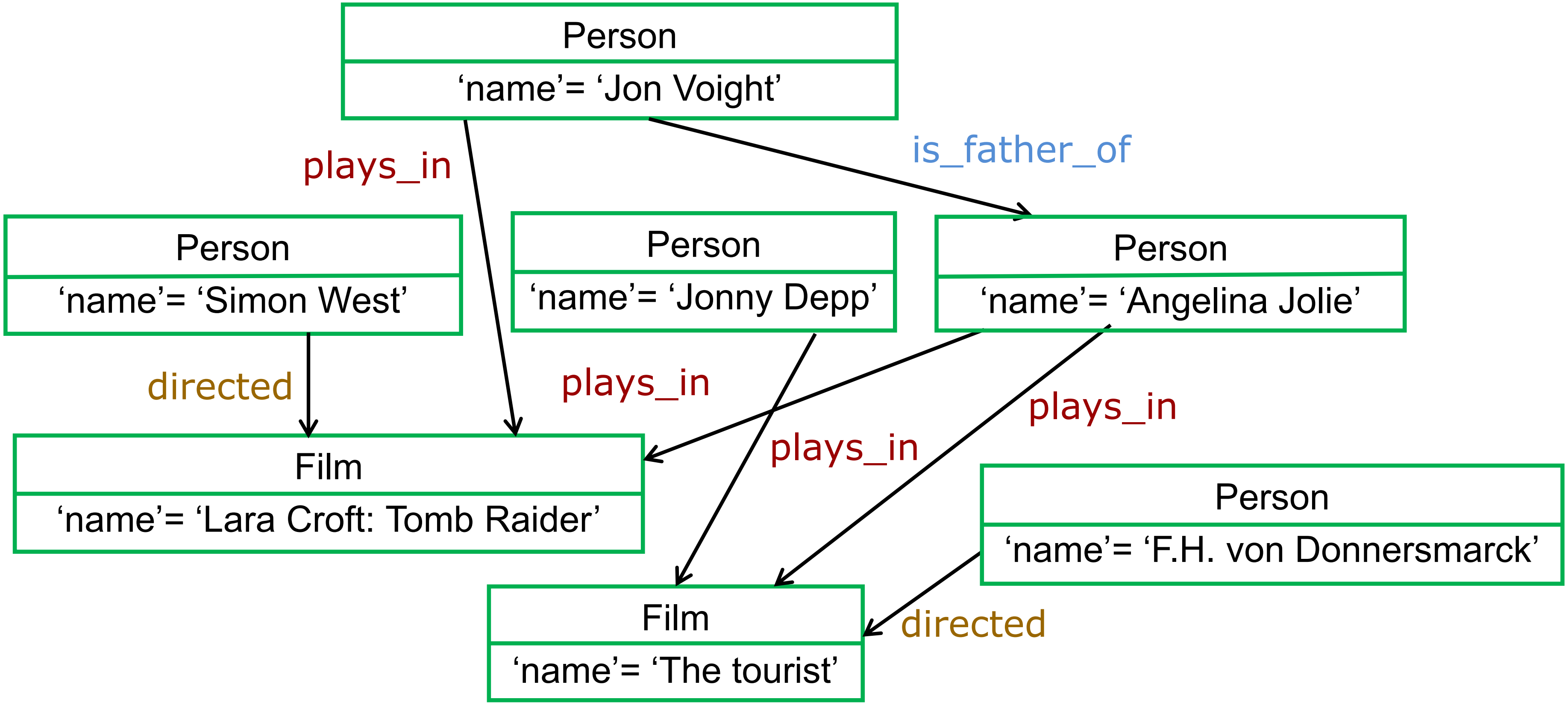
Functions: e.g. nodes(p) (relationships(path) ) returns all nodes (relationships) on the path p

# Example

- WHERE: filtering criteria (AND, OR, NOT, comparison, regular expression, …)
- RETURN: answer form
- ORDER BY: (similar to SQL) DESC, …
- LIMIT: output limit (number of rows)
- Aggregation functions: COUNT, SUM, AVG, MAX, MIN, COLLECT

- Advanced querying: embedded graph algorithms
  - shortestPath
  - allSimplePaths
  - allPaths
  - dijkstra (optimally with cost_property and default_cost parameters)

# Examples in Cypher



Person
'name'= 'Jon Voight'

plays_in

is_father_of

Person
'name'= 'Simon West'

Person
'name'= 'Jonny Depp'

Person
'name'= 'Angelina Jolie'

directed

plays_in

plays_in

plays_in

Film
'name'= 'Lara Croft: Tomb Raider'

Person
'name'= 'F.H. von Donnersmarck'

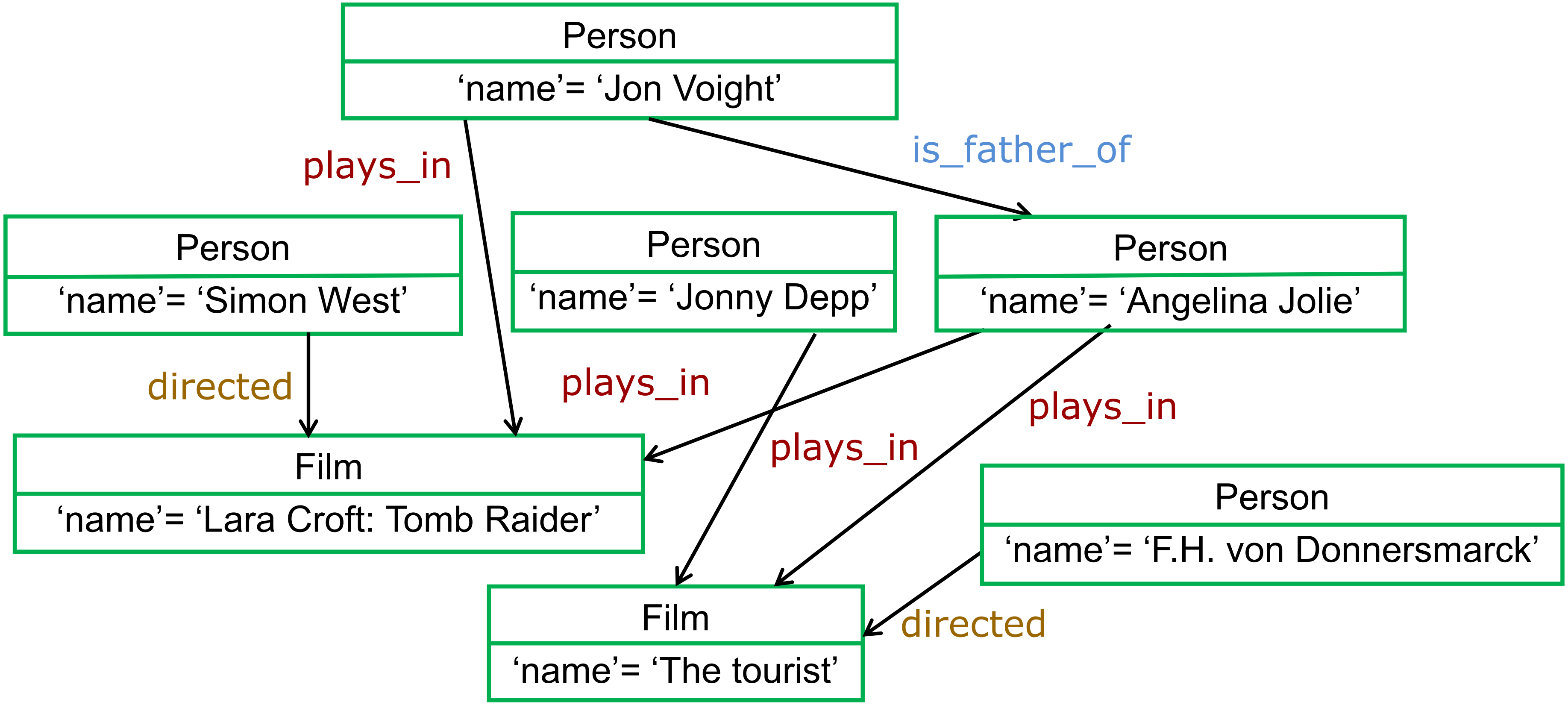
Film
'name'= 'The tourist'

directed

Query1: Find films related to Angelina Jolie

```
START x=node:Person(name ='Angelina Jolie')
MATCH (x)- ->(film)
RETURN film.name
```

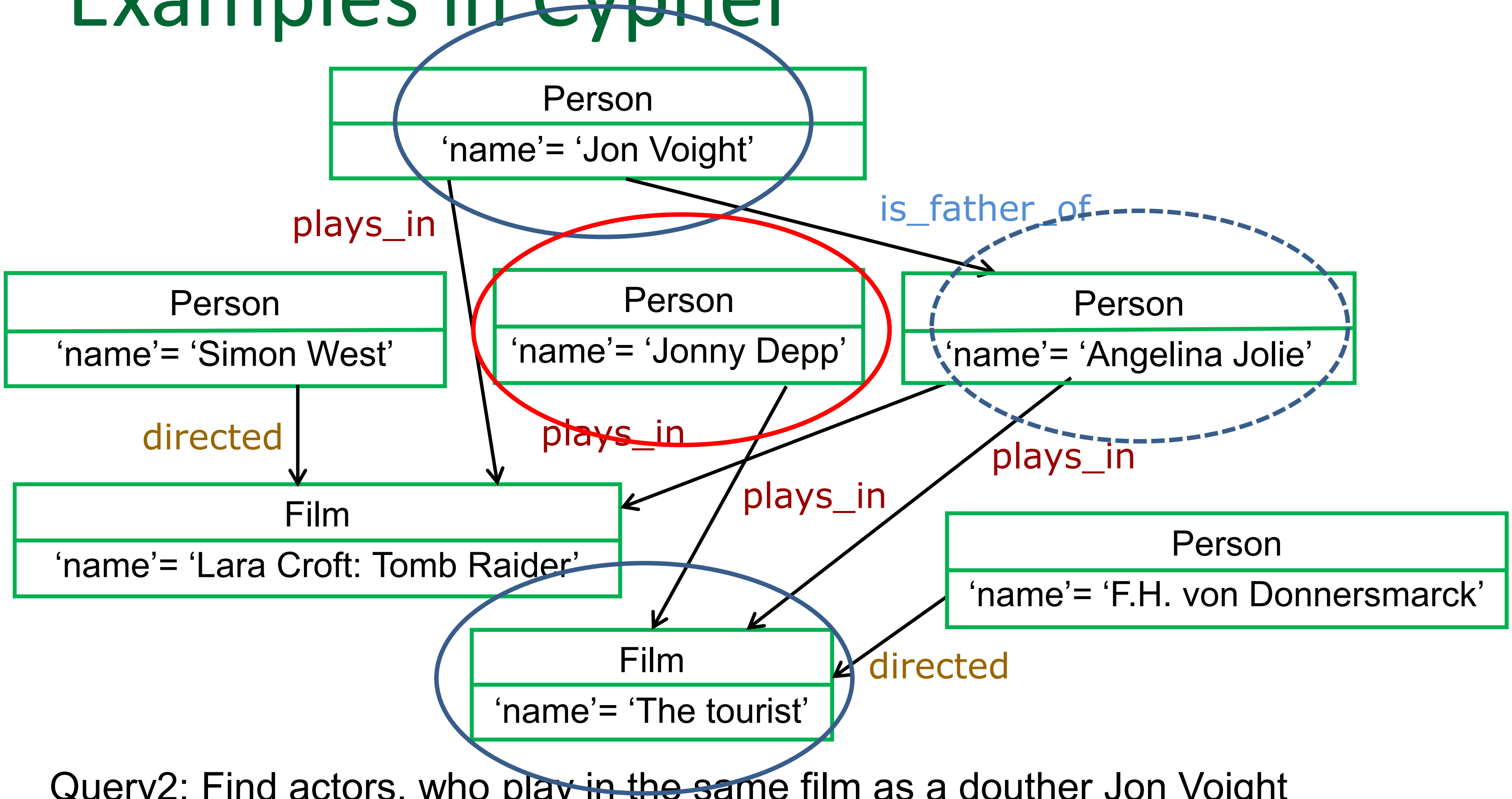
# Examples in Cypher



Query2: Find actors, who play in the same film as a douther Jon Voight

```
MATCH actor1-[:plays_in]->film<-[:plays_in]-()<-[:is_father_of]-actor2
WHERE actor2.name = 'Jon Voight'
RETURN actor1.name
```

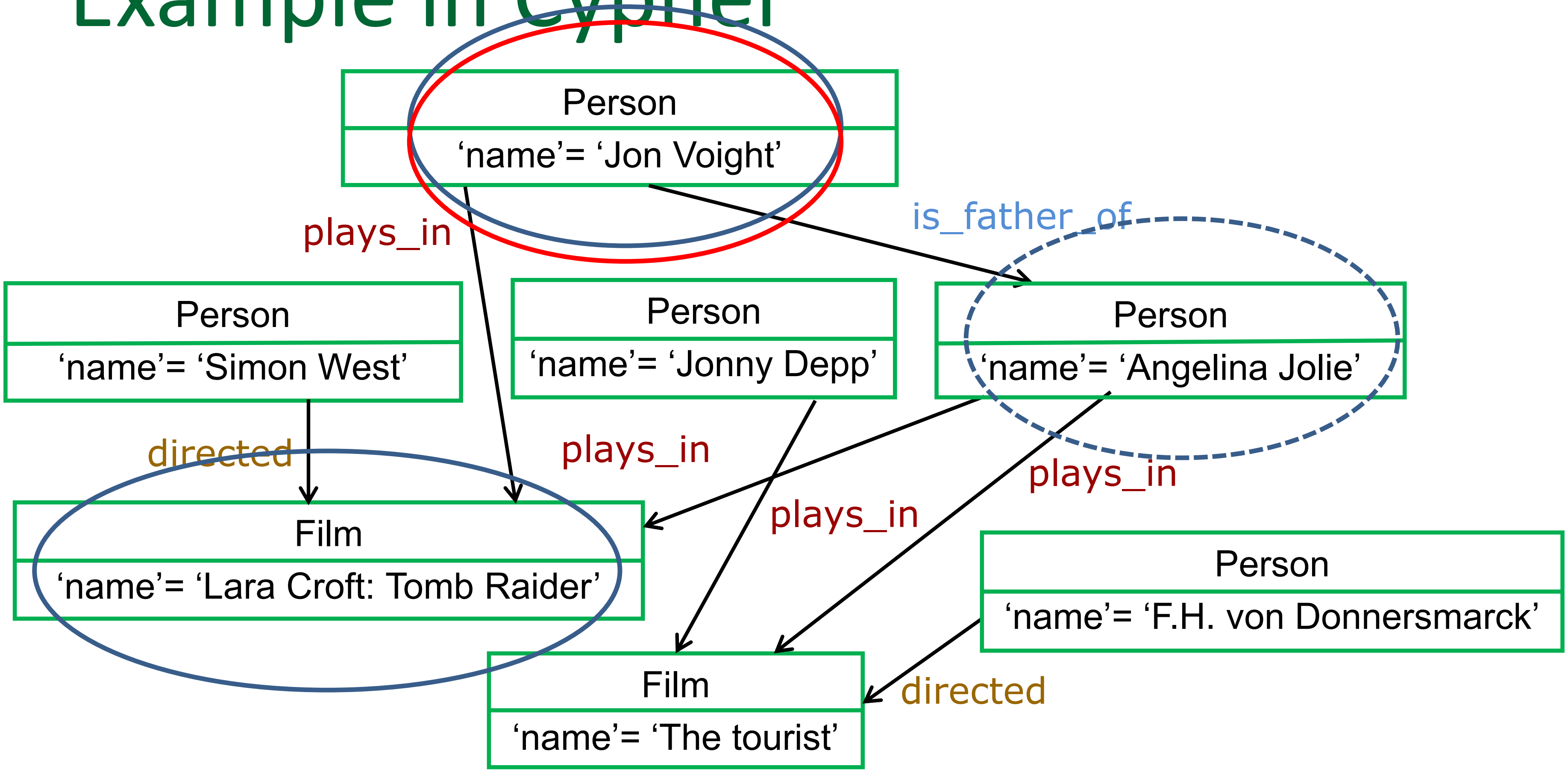
# Examples in Cypher



Query2: Find actors, who play in the same film as a douther Jon Voight

```
MATCH actor1-[:plays_in]->film<-[:plays_in]-()<-[:is_father_of]-actor2
WHERE actor2.name = 'Jon Voight'
RETURN actor1.name
```

# Example in Cypher



Query2: Find actors, who play in the same film as a douther Jon Voight

```
MATCH actor1-[:plays_in]->film<-[:plays_in]-()<-[:is_father_of]-actor2
WHERE actor2.name = 'Jon Voight'
RETURN actor1.name
```

# Scalability

- Scaling for:
  - large datasets – no problem – graphs of order $10^{10}$ in Neo4j
  - read performance – no problem – e.g., Neo4j is focused on read performance
  - write performance – no problem in the case of vertical scaling, but when horizontal scaling is necessary
- Example:
  - Titan[2] is a highly scalable OLTP GDBMS optimized for thousands of users concurrently accessing and updating one Big Graph.

[2] http://thinkaurelius.github.io/titan/

# Scalability

- Sharding (or graph partitioning) - distributing it across multiple machines
- Issues:
  - much more difficult than scaling the simpler data in other NoSQL databases
  - Goal: to avoid having relationships that span machines as much as possible ⇒ minimum point-cut problem (NP-hard)

    In practice: connected graphs can mutate rapidly and unpredictably at runtime

# Transaction processing

GDB are often optimized and focused on

- CRUD (create, read, update, delete) operations,
- query processing - reporting, data warehousing, and real-time analytics,
- batch mode analytics or data discovery.

- not all GDB are fully ACID
- BASE properties often considered in the context of NoSQL DB - variant not too appropriate for graphs
- partitioning and replication strategies to maximise the locality of the processing
- Example: Neo4j uses master-slave replication

# Categories of graph databases

Examples of GDBMSs:

- general purpose GDBMS
  - distributed: Sparksee (originally DEC), InfiniteGraph, Titan, GraphBase
  - centralized: Neo4j
- special GDBMS
  - Web-oriented: InfoGrid, FlockDB
  - multimodel: OrientDB (graph + documents + SQL)
  - hypergraphs: HyperGraphDB
  - triplestores: AllegroGraph …
- low-level platforms
  - Pregel, Giraph

# Triplestores

- Storage for RDF data: subject (with) - predicate (P) – object (O)
- logical level: a table, sometimes other columns are added: graph name (N), ID
- Examples: AllegroGraph, BrightStarDB, Bigdata, SparkleDB
- More advanced: AllegroGraph, GraphDB™ support reasoning and ontology modelling
- hybrid solution: Virtuoso Universal Server (RDF data, relations, XML, text)
- General observation: not yet suitable for storing truly large data sets efficiently

# Pregel and Giraph

- Bulk Synchronous Processing (BSP) model is used to the design, analysis and implementation of parallel algorithms there.
  - powerful generalization of MapReduce (MR)
- Pregel - a system for large-scale graph processing on distributed cluster of commodity ma-chines. It is based on BSP.
- Giraph extends Pregel. It utilizes Apache MR framework implementation to process graphs.
  - Now: for Currently used at Facebook to analyze the social graph
- Pregel and Giraph do not use a graph database for storage.

# RDBMS vs. Graph databases

- Example by E. Eifrem (CEO of Graph Database company Neo) about testing speed of the "friends of friends" query
  - three levels depth: GDB beat the relational one by a factor of 150,
  - four levels depth: the GDB bested the relational one by a factor of 1000.

# DB-Engines Ranking of GDBMSs

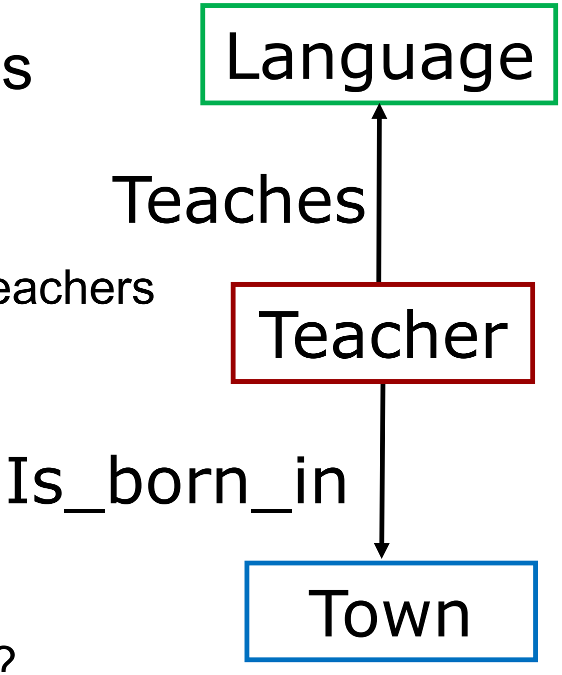
See http://db-engines.com/en/ranking/graph+dbms
31 systems in ranking, April 2019

| | | Score | |
|---|---|---|---|
| 1. | Neo4j | 49.49 | |
| 2. | Microsoft Azure Cosmos DB | 26.28 | /multi-model/ |
| 3. | OrientDB | 6.19 | /multi-model/ |
| 4. | Arrango | 4.29 | /multi-model/ |
| 5. | Virtuoso | 3.31 | /multimodel/ |
| 6. | Amazone Neptune | 1.39 | /multi-model/ |
| 7. | JanusGraph | 1.38 | |
| 8. | Giraph | 1.20 | |
| 9. | Dgraph | 1.08 | |
| 10. | GraphDB | 0.97 | /multi-model/ |

# Graph databases modelling

- Graph database schema

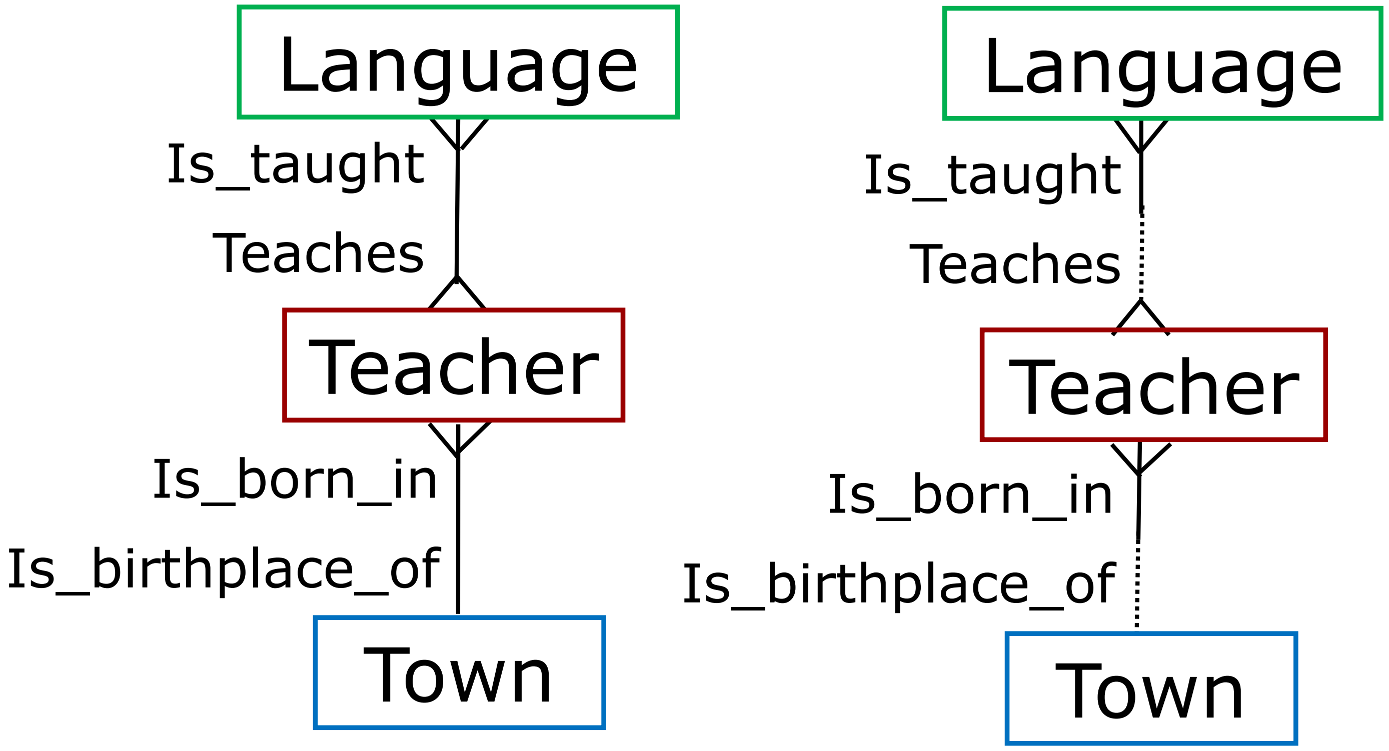
  specifies: structure + property definitions
  + integrity constraints (IC)
  - Examples of properties:
    - Name for Language, Birth_year, #T_ID for Teachers
    - Room for Teaches



- „weak“ GDB schema
  - does not contain cardinalities,
    - Does teacher teach more languages?
    - Can be a language taught by more teachers?
  - does not contain dependencies among properties,
  - contains partially: property domains, simple property constraints.

# Graph databases modelling

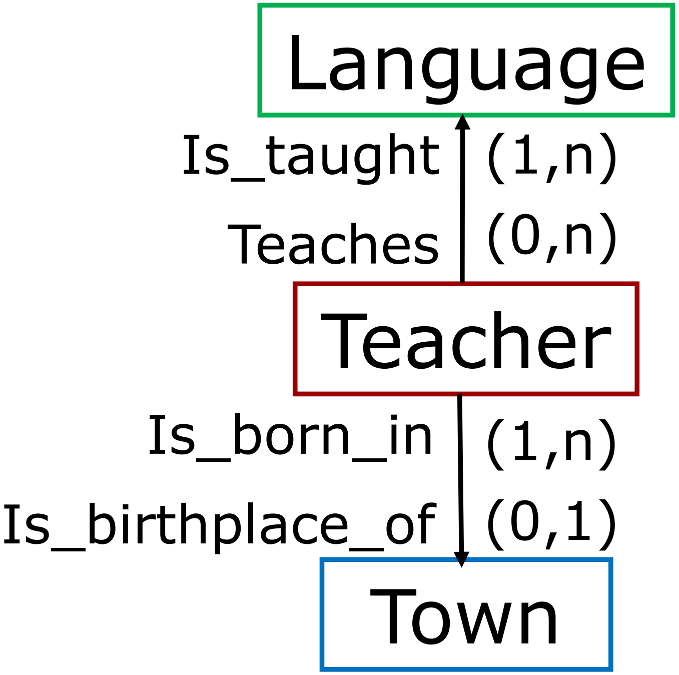
- Graph conceptual schema



# Graph database modelling

- Graph conceptual schema

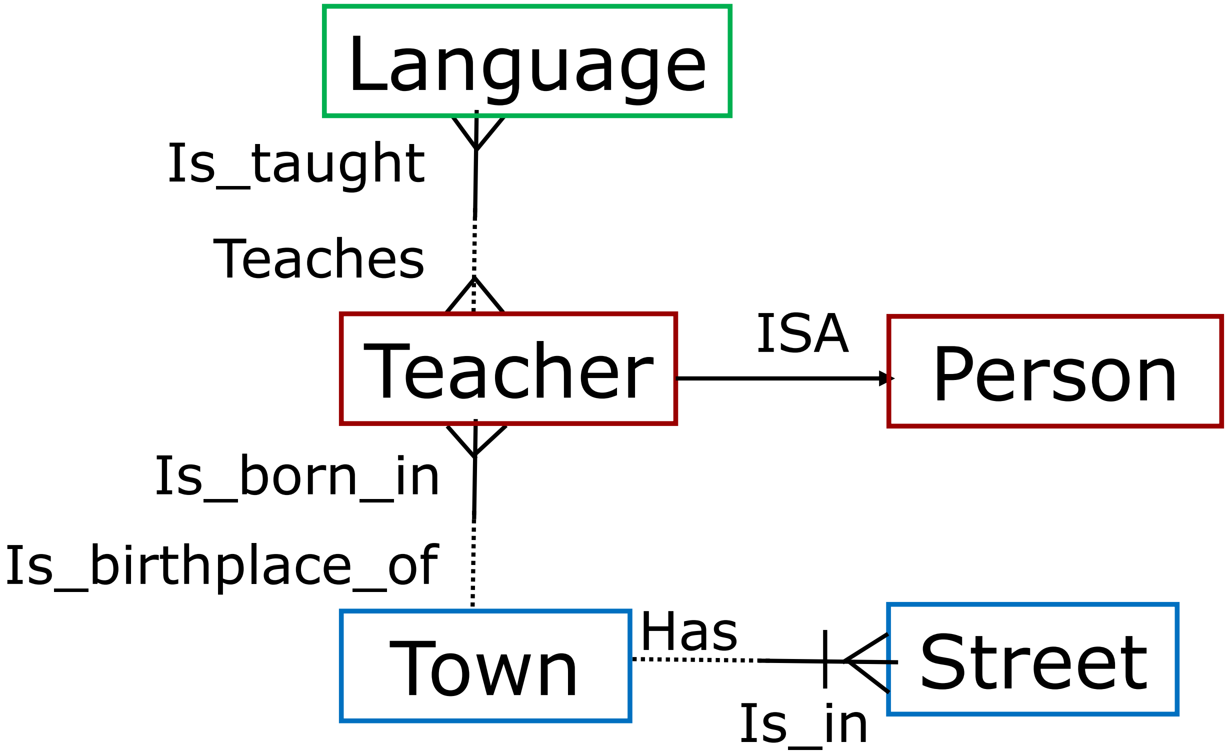
  variant with min-max ICs



# Graph database modelling

- Graph conceptual schema

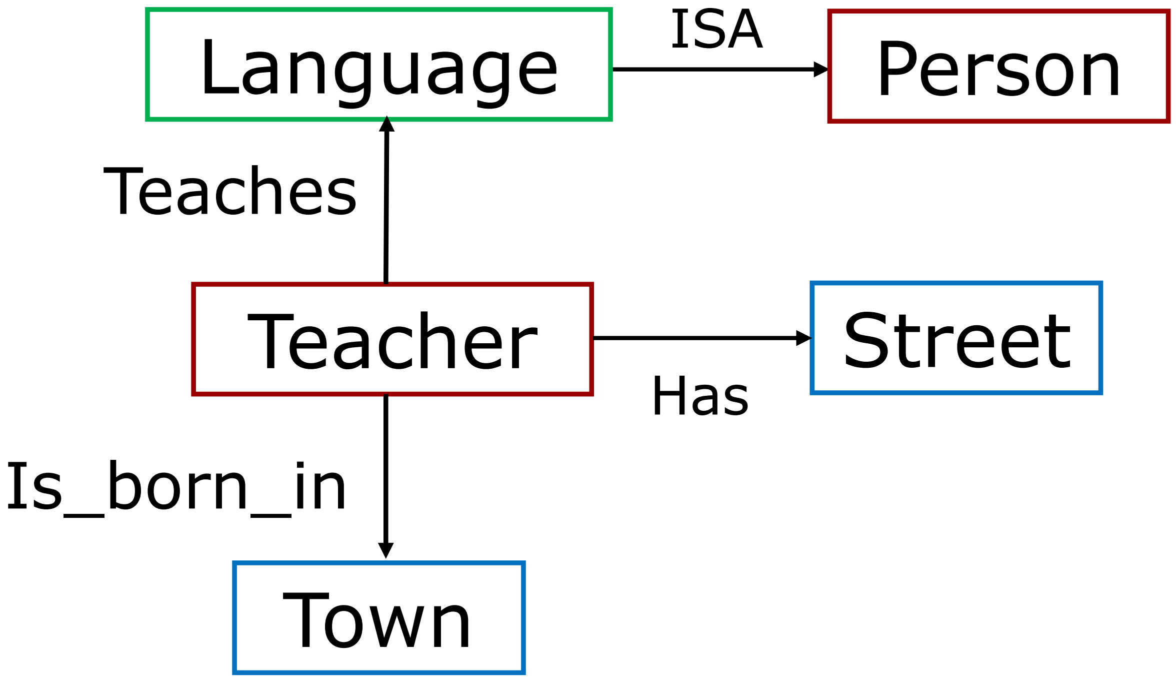
variant with ISA-hierarchiees and weak entities



# Graph database modelling

- Associated graph database schema

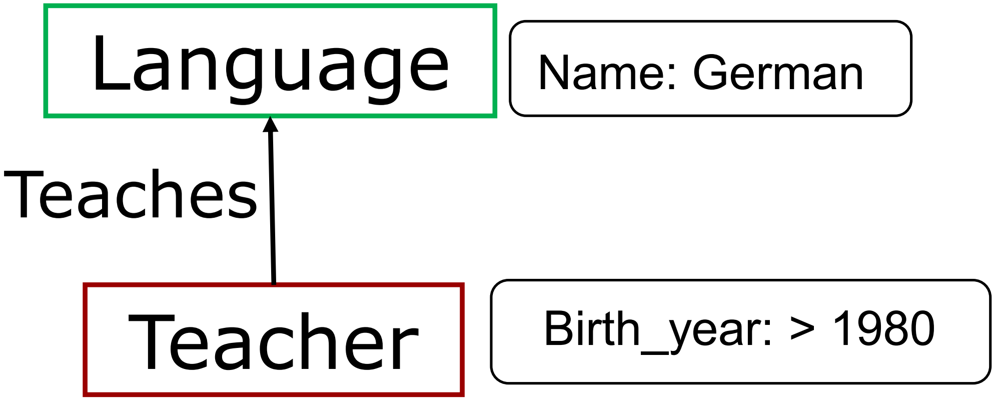
  variant with ISA-hierarchies and weak entities



# Graph database modelling

- (database) integrity constraint using a pattern
- Ex.: GDBMS GRAD (has a database model, schema-less)

  IC: Each teacher teaching German is born after 1980.



# Mapping: conceptual → database

- Transforms the graph conceptual schema into an equivalent (or nearly equivalent) graph database schema.
- The resulted schema is not given uniquely. It is necessary to decide:
  - for edge direction of a given relationship type,
  - for the edge label.
- For ISA entities and weak entities it necessary to consider different keys of participated entity types (they are inherited from higher hierarchy members and from strong entity types).

# Schemas and practice

The approach of Neo4j:

- Neo4j is schema-less.
- some possibilities of IC definitions:
  - CREATE CONSTRAINT ON (Teacher:Teacher)

    ASSERT Teacher.#T_ID IS UNIQUE,
  - CREATE CONSTRAINT ON (Teacher:Teacher)

    ASSERT exists(Teacher.Birth_year),

  i.e. all nodes with a certain label have a certain property.
  - CREATE CONSTRAINT ON ()-[teaches:Teaches]-()

    ASSERT exists(teaches.Room),

  i.e. all relationships with a certain type have a certain property.

# Schemas and practice

The approach of OrientDB: the role of graph database schema can be precisely specified

- schema-full - enables strict-mode at a class-level and sets all fields as mandatory.
- schema-less - enables classes with no properties. Default is non-strict-mode, meaning that records can have arbitrary fields.
- schema-hybrid - enables classes with some fields, but allows records to define custom fields. This is also sometimes called schema-mixed.

- Graph database schema can sometimes be disadvantageous, e.g., in dynamic environment, in quickly changing application domain, etc.

# Limitations of graphs databases

- Functionality restrictions
- Big Analytics requirements
- Other challenges

# Functionality restrictions

- Declarative querying: Most commercial GDB cannot be queried using a declarative language.
- Data partitioning: Most graph databases do not include the functionality to partition and distribute data in a computer network. It is difficult to partition a graph in a way that would not result in most queries having to access multiple partitions.
- Vectored operations: They support a procedure which sequentially writes data from multiple buffers to a single data stream or reads data from a data stream to multiple buffers. It seems that it is not the case in GDBs today.

# Functionality restrictions

- Model restrictions:
  - data schema and ICs definitions are restricted in GDBs. Therefore, data inconsistencies can quickly reduce their usefulness.
  - graph model itself can be restricted. For example, Neo4j nodes cannot reference themselves directly. Sometimes self-reference can be required.
- Querying restrictions: For example, FlockDB overcomes the difficulty of horizontal scaling the graph by limiting the complexity of graph traversal.
  - does not allow multi-hop graph walks, so it cannot do a full "transitive closure".
  - enables very fast and scalable processing of 1-hop queries.

# Big Analytics requirements

- High cost of some queries: Most real-world graphs are highly dynamic and often generate large volumes of data at a very rapid rate. A challenge is how to store the historical trace compactly while still enabling efficient execution of point queries and global or neighbourhood-centric analysis tasks.
- Real time processing: graph data discovery takes place essentially in batch environments, e.g., in Giraph. Some products aimed at data discovery and complex analytics that will operate in real-time.
- Indexes size: path-based and graph-based approaches suffer from large index size, substantial index construction overhead and expensive query processing cost
- Graph extraction: how to efficiently extract a graph, or a collection of graphs, from non-graph data stores.

# Big Analytics requirements

- Complex graph algorithms are needed in practice. The ideal GDB should understand analytic queries that go beyond *k*-hop queries for small *k*.
  - Experiment[1]: networks with 256 million edges + 4 fundamental graph algorithms + 12 GDBMS. The most popular GDBMS have reached the worst results in these tests.
- Parallelisation: when the data is too big to handle on one server. What to do when the data does not all fit into the memory available. Due to the fact that partitioning a graph is a problem, most GDBs do not provide shared nothing parallel queries on very large graphs.
- Heterogeneous and uncertain graph data: important when data sets need to be semantically integrated in order to be effectively queried or analysed.

[1] McColl, R., et al: A Performance Evaluation of Open Source Graph Databases. Proc. of PPAA ’14, pp. 11-18. ACM, NY (2014)

# Other challenges

- Design of GDB: Similarly to traditional databases, some attempts to develop design models and tools occur in last time. See, e.g., starting from a conceptual schema in ER-model.
- More user-friendly querying. Due to their complex schemas and a variety of information descriptions, it becomes very hard to formulate a query that can be properly processed by the existing systems.
- Graph pattern matching: new semantics and algorithms for graph pattern matching over distributed graphs are in development.
- Developing heuristics for some hard graph problems:
  - Example: partitioning of large-scale dynamic graph data for efficient distributed processing (the classical graph partition problem is NP-hard)

# Other challenges

- Keyword Search on Graph Representation of Data: Problem: to find a (closely) connected set of nodes that together match all given keywords. How to extend it to ontologies?
- Visualization: Improvement of human-data interaction is fundamental, particularly a visualization of large-scale graph data, and of query and analysis results.
- Need for benchmarks:
  - The benchmarks built, e.g., for RDF data, are mostly focused on scaling and not on querying.
  - Benchmarks covering a variety of graph analysis tasks are missing. They would help towards evaluating and comparing the expressive power and the performance of different graph databases and frameworks.

# Other challenges

- Graph streams processing: with goal to compute properties of a graph without storing the entire graph.
- Compressing graphs: matching without decompression is possible. Combining parallelism with compressing or partitioning is also very interesting.

# Conclusions

- Technological trends:
  - Usage of graphics processors (GPUs)
  - Overcome the problem to run graphical DB on a single machine
  - parallelization of graph databases
- Usual question:

  What to choose?

  - There is no single answer: everything depends on how the data will be used.
  - If necessary, two separate persistence solutions can coexist for each use case (polyglot persistence).

# References

- Sasha et al: Algorithmics and Applications of Tree and Graph Searching PODS'02
- Cypher: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html
- The Neo4j Documentation:

  https://neo4j.com/docs/
- Pokorný, J., Snášel, V.:  Big Graph Storage, Processing and Visualization. Chapter 12 in: Graph-Based Social Media Analysis, I. Pitas (Ed.), Chapman and Hall/CRC, pp. 403 – 430, 2015.