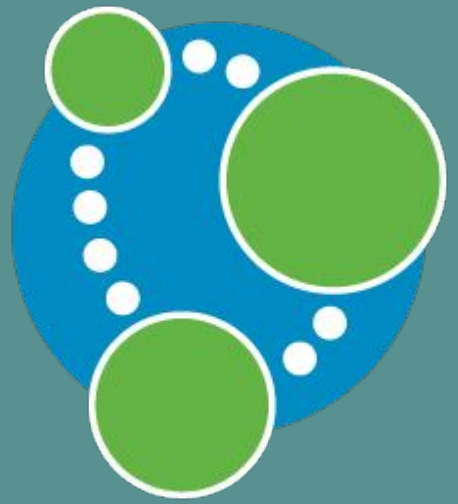


neo4j

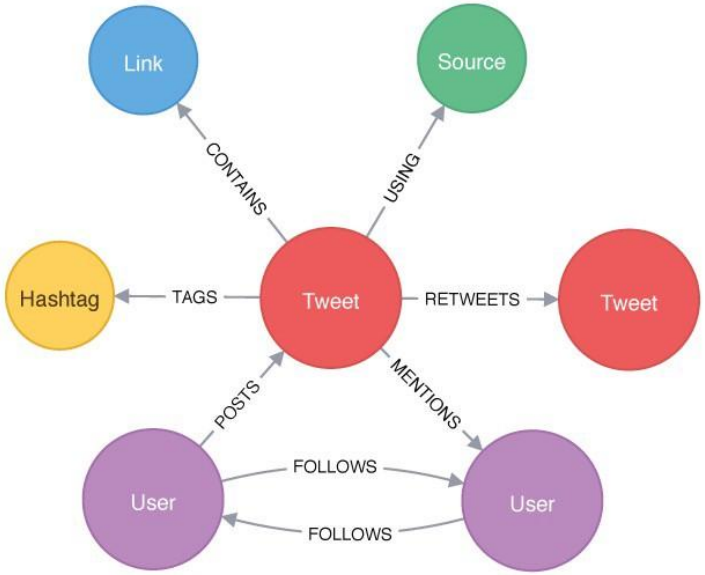
Mike Schumacher & Lucas Gehlen

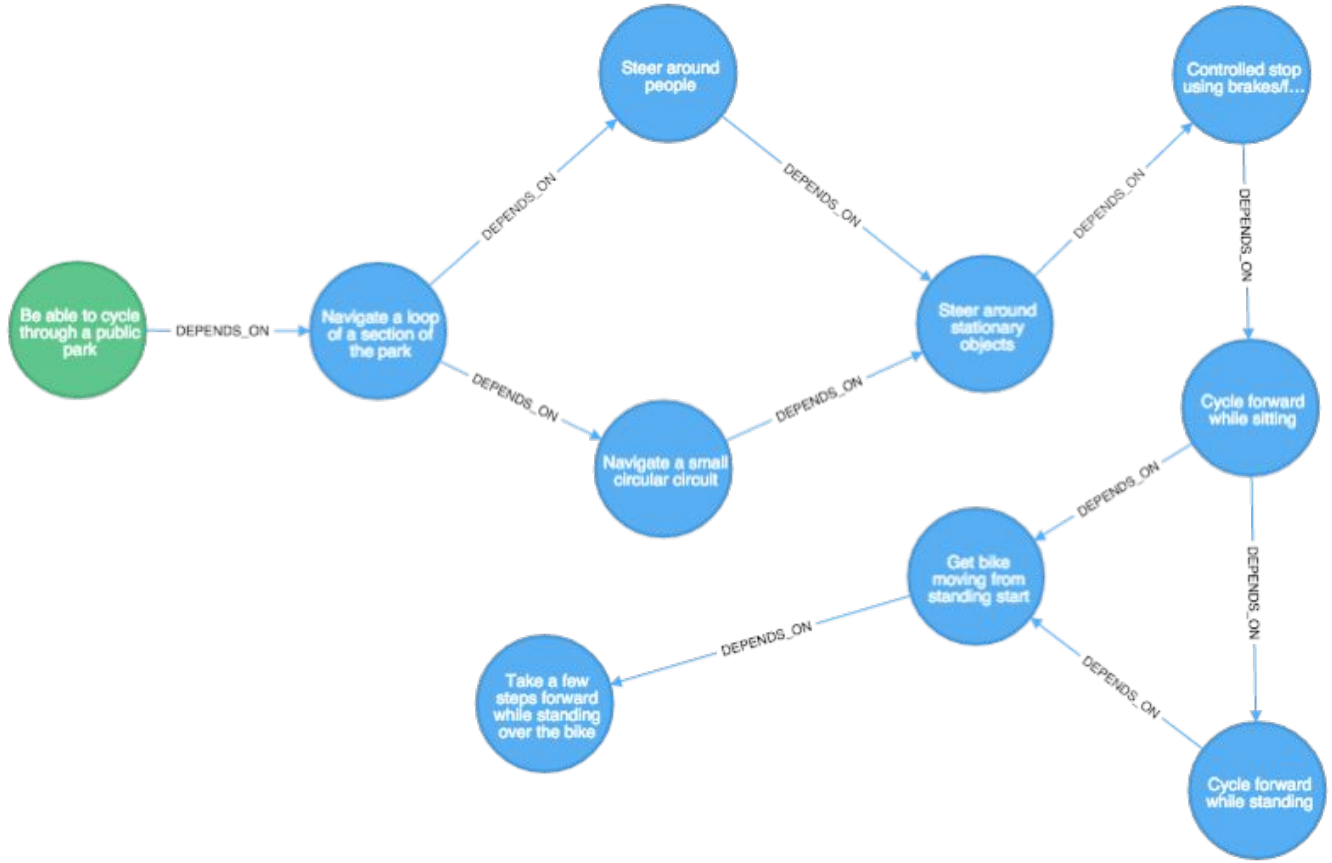
Agenda

- What is Neo4j?
- NOSQL
- Neo4j in action
- Example RDBMS to Graphs
- The Property Graph Data Model
- Cypher
- Assignment 1: First steps with cypher
- RDBMS vs. Graph DB
- Assignment 2: Beer DB
- Functions in cypher
- Assignment 3: Queries with functions
- Java and Neo4j

What is Neo4j?

- Open-Source graph database
- In Java!
- NoSQL based





No… NO… NOSQL

- One of 4 NoSQL DB types
- No Tables
- Nodes+Relationships

Computer

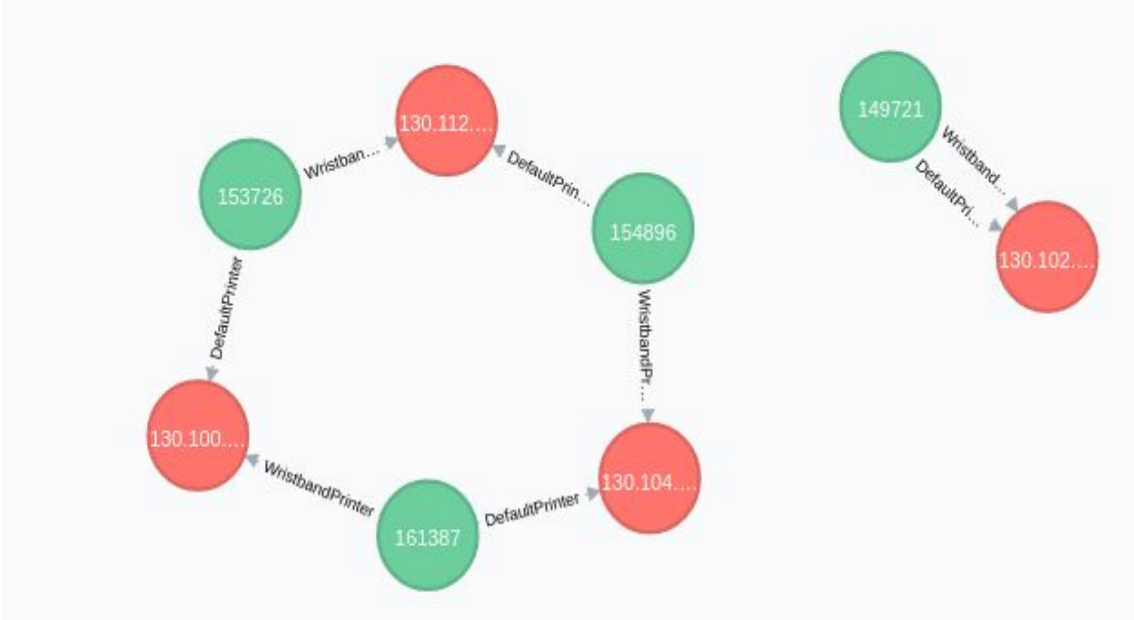
Asset	DefaultPrinter	WristbandPrinter
149721	1	1
153726	4	3
154896	3	2
161387	2	4

Printer

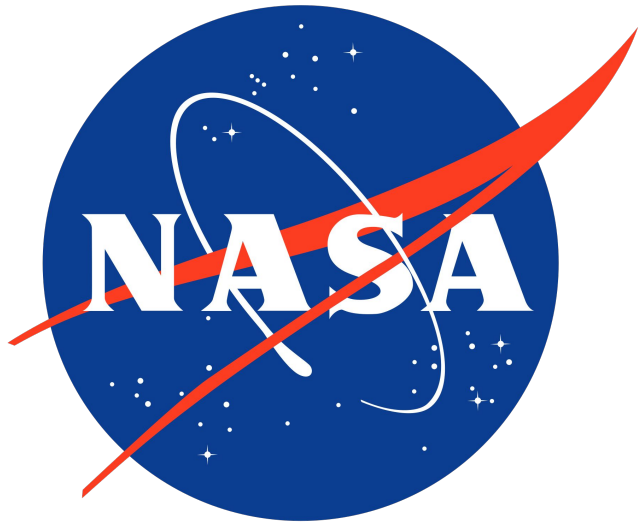
PrinterID	Path
1	130.102.198.12
2	130.104.200.25
3	130.112.201.26
4	130.100.100.9

Computer_Printer

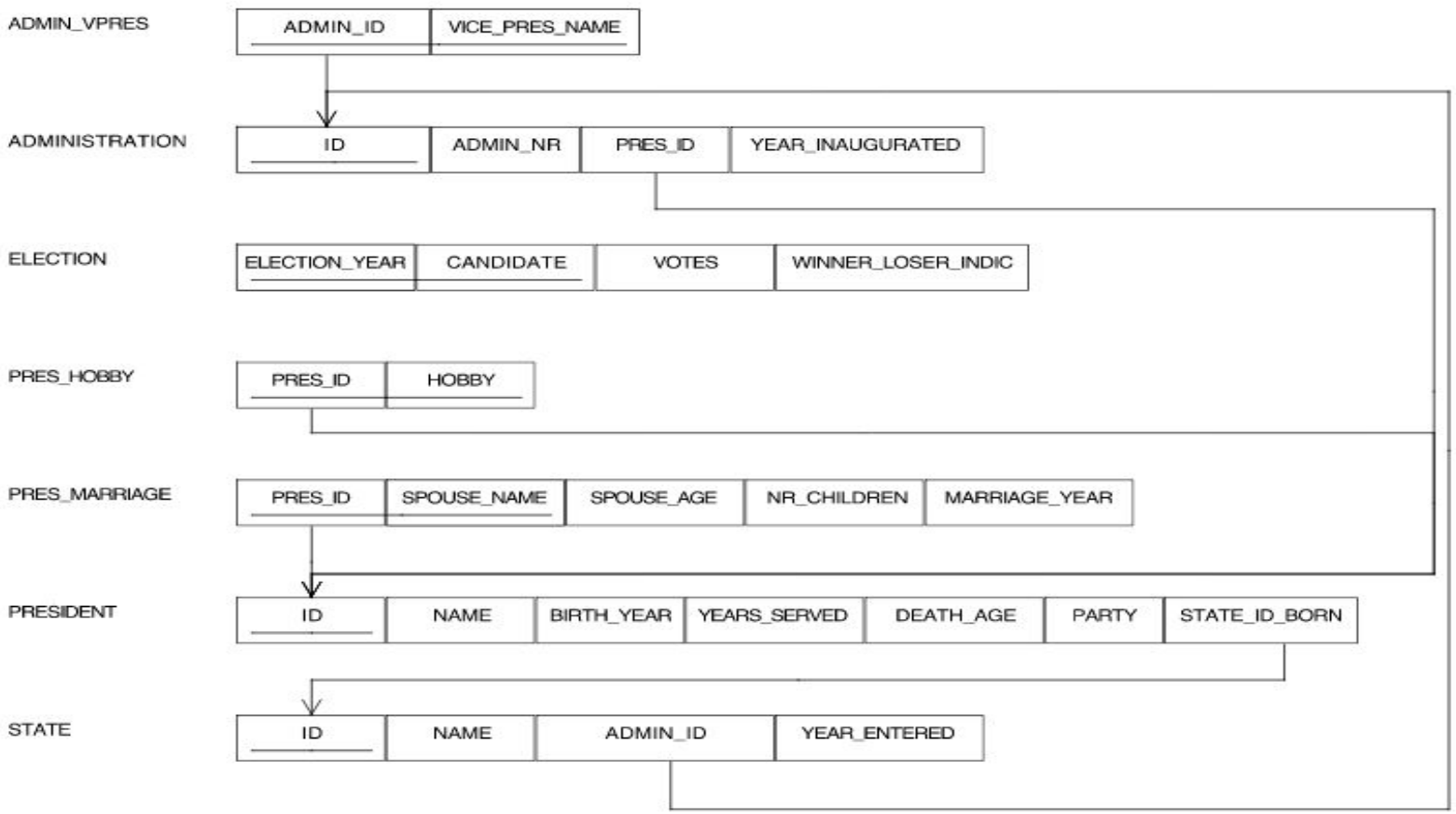
Asset	DefaultPrinter	WristbandPrinter
149721	130.102.198.12	130.102.198.12
153726	130.100.100.9	130.112.201.26
154896	130.112.201.26	130.104.200.25
161387	130.104.200.25	130.100.100.9



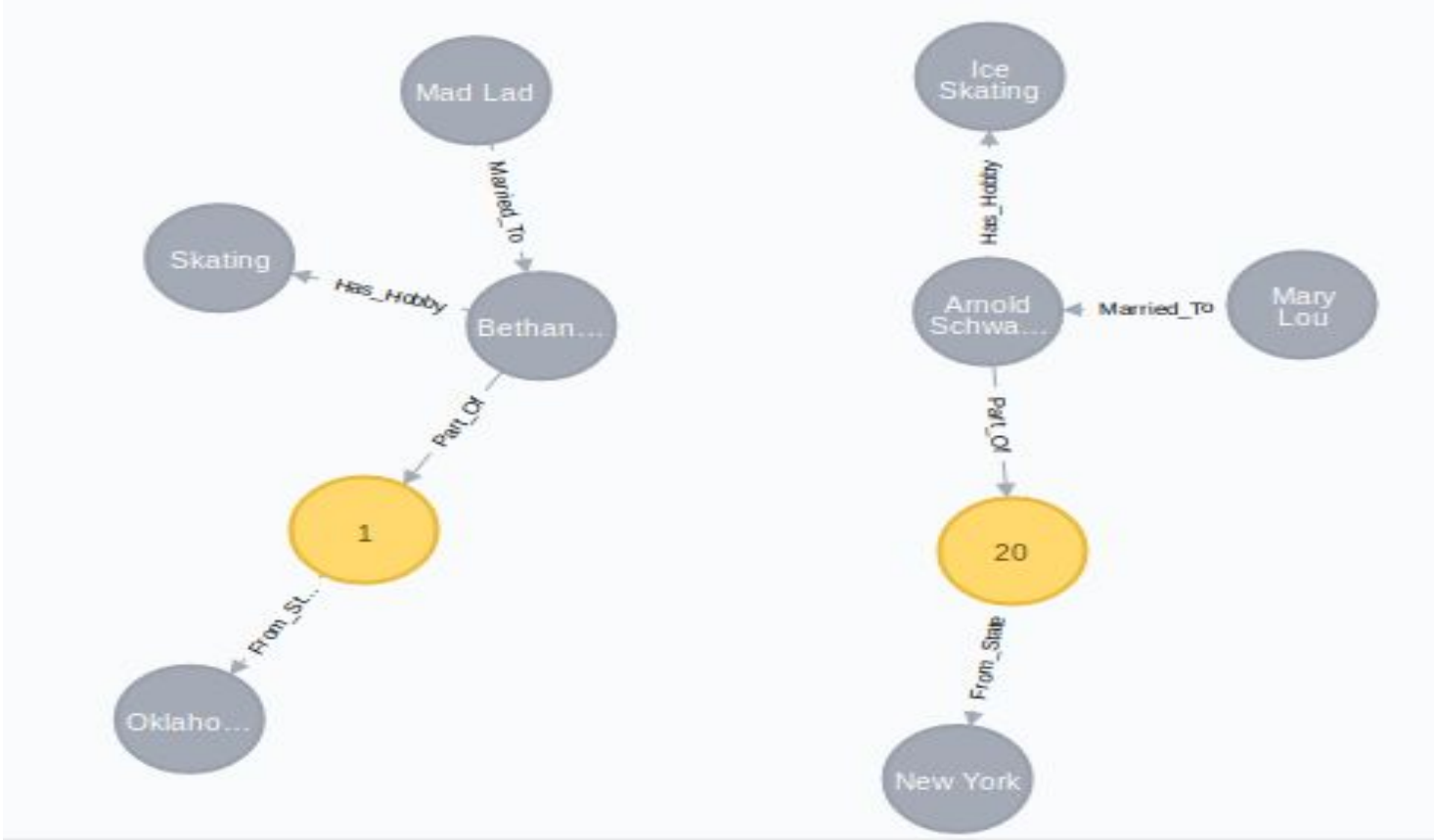
Neo4J in action



Example RDBMS to Graphs

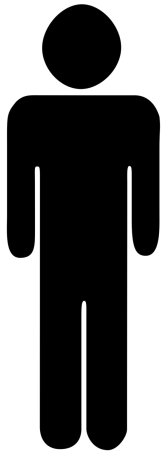


Example RDBMS to Graphs

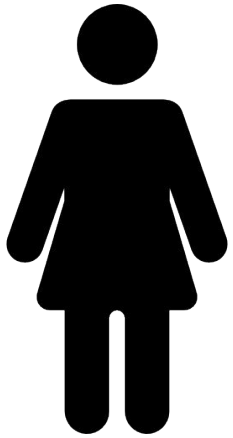


The Property Graph Data Model

James is married to Mary

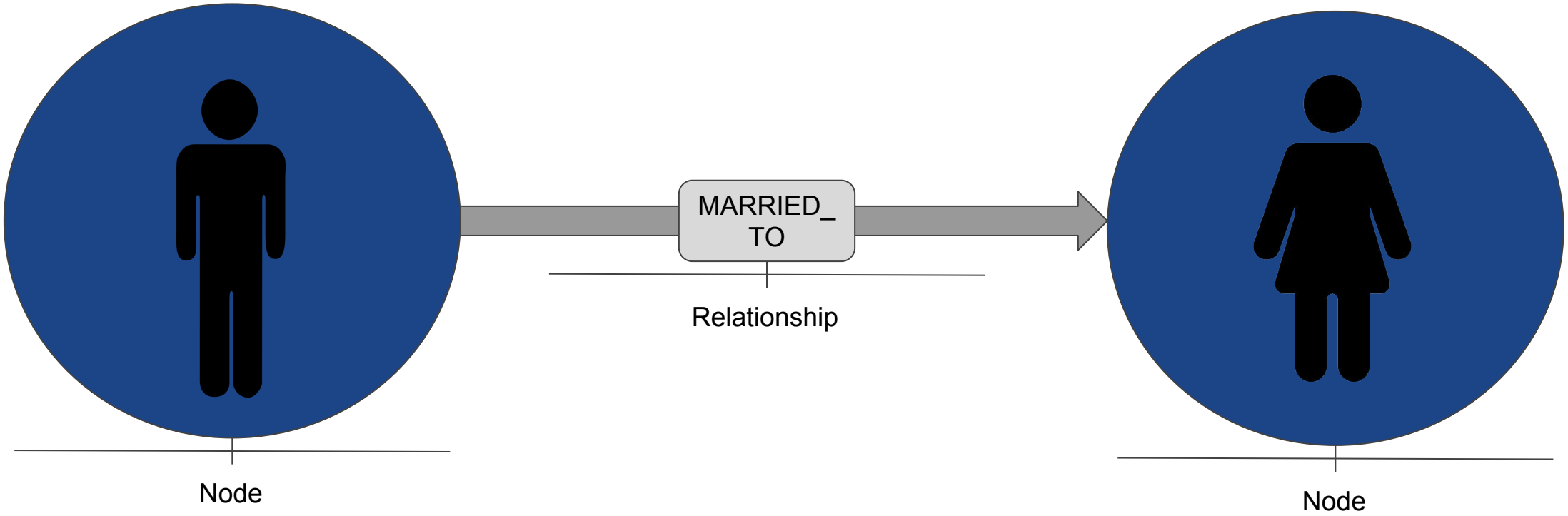


James

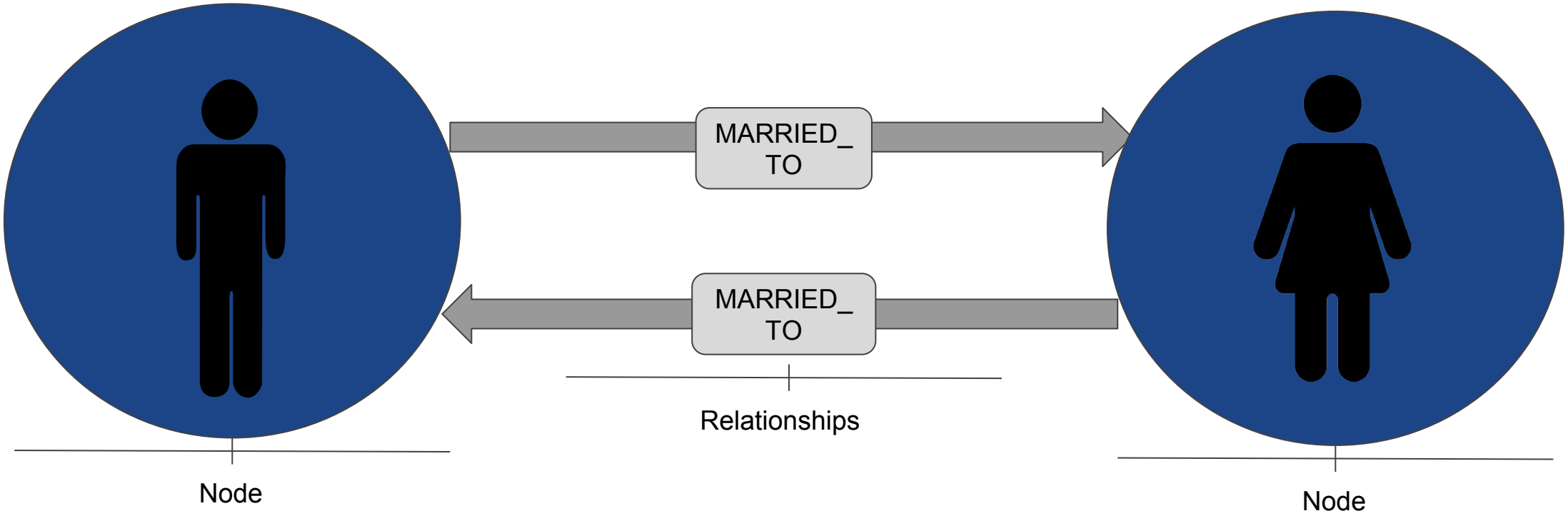


Mary

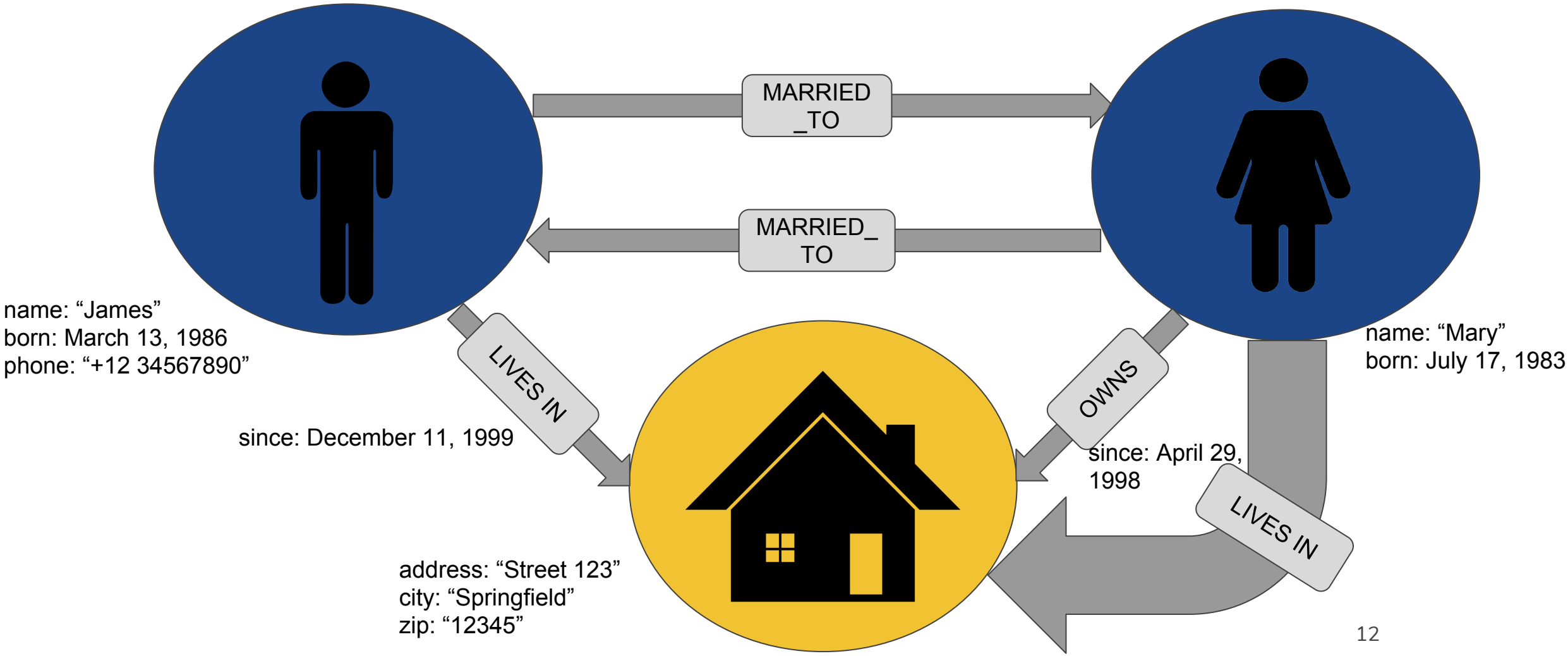
James is married to Mary



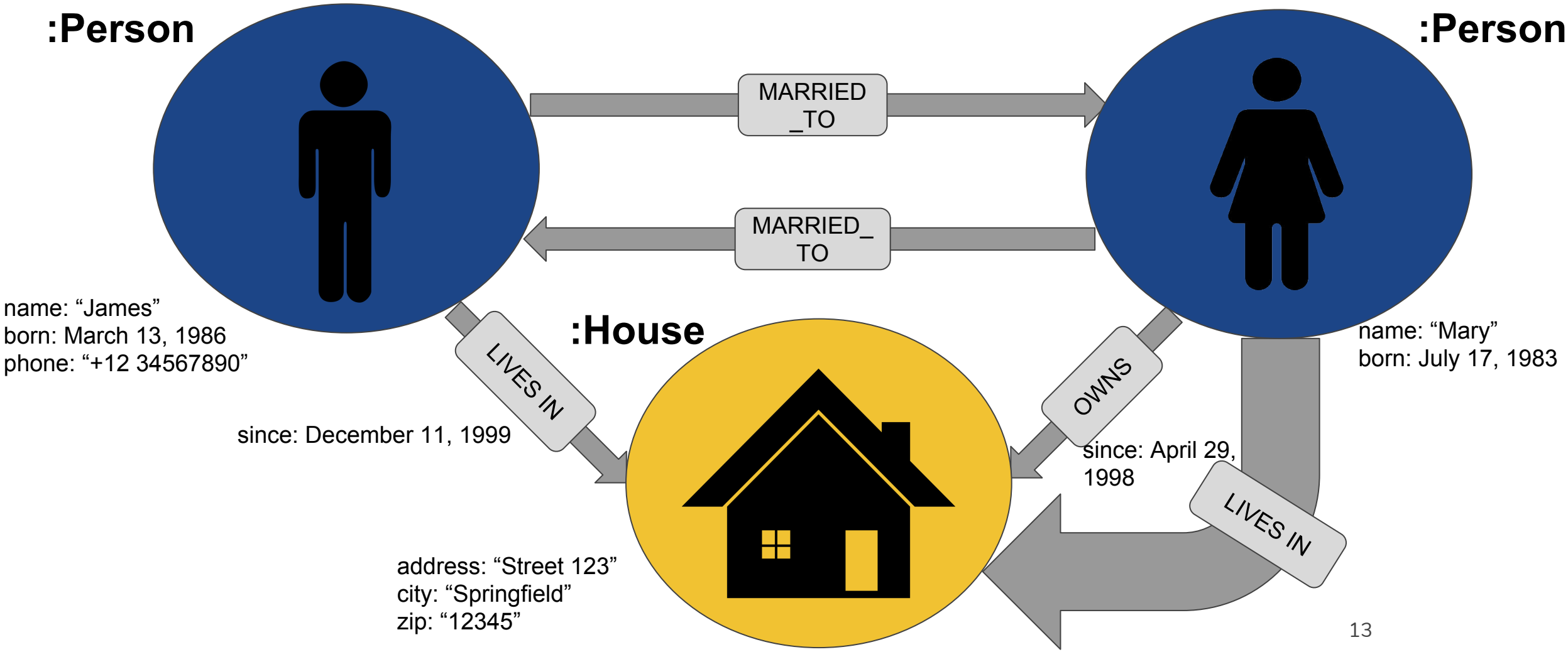
Relationships are Directional



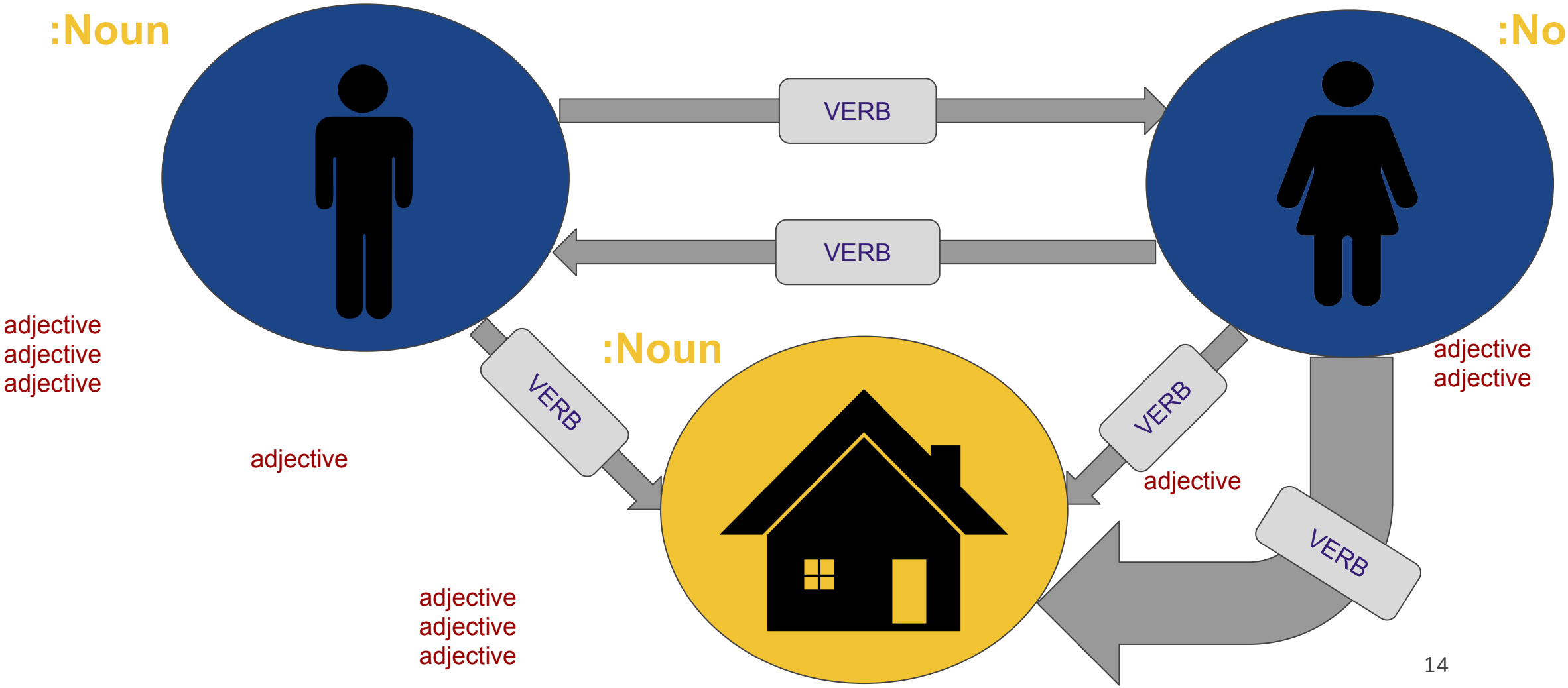
Detailed Property Graph



Labeled Property Graph

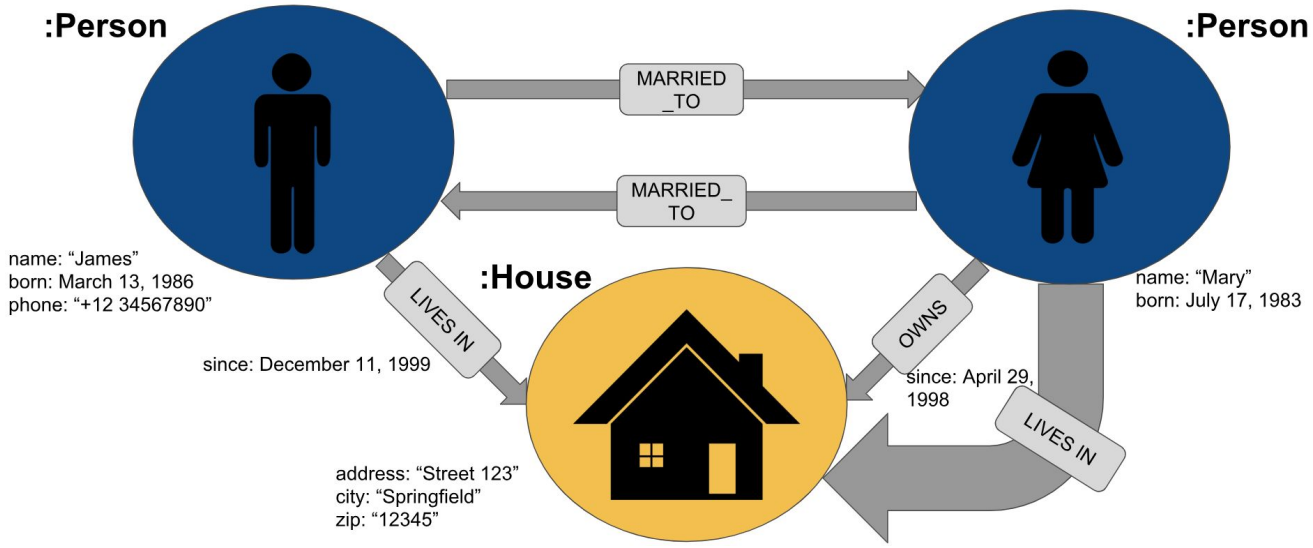


Mapping to Languages



Property Graph Model Components

- Nodes
  - Objects in the graph
  - Can have name-value properties
  - can be labeled
- Relationships
  - Relate nodes by type and direction
  - Can have name-value properties



Cypher

SQL for graphs

About Cypher

- Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax

About Cypher

- Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax

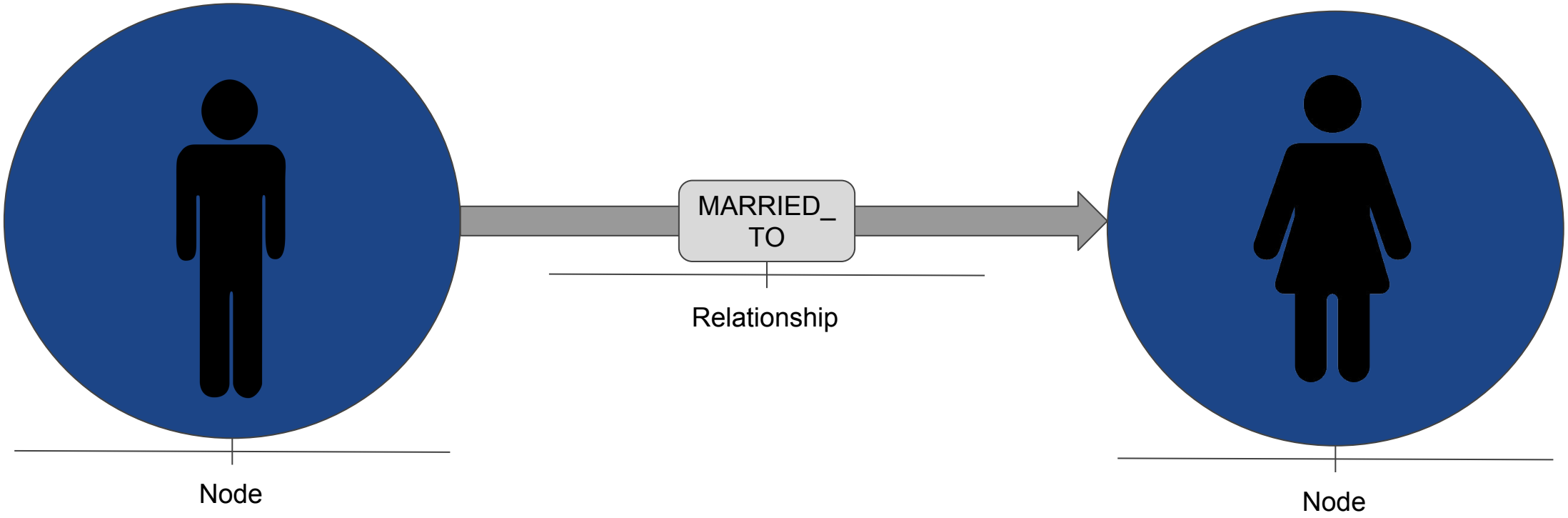
About Cypher

- Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax

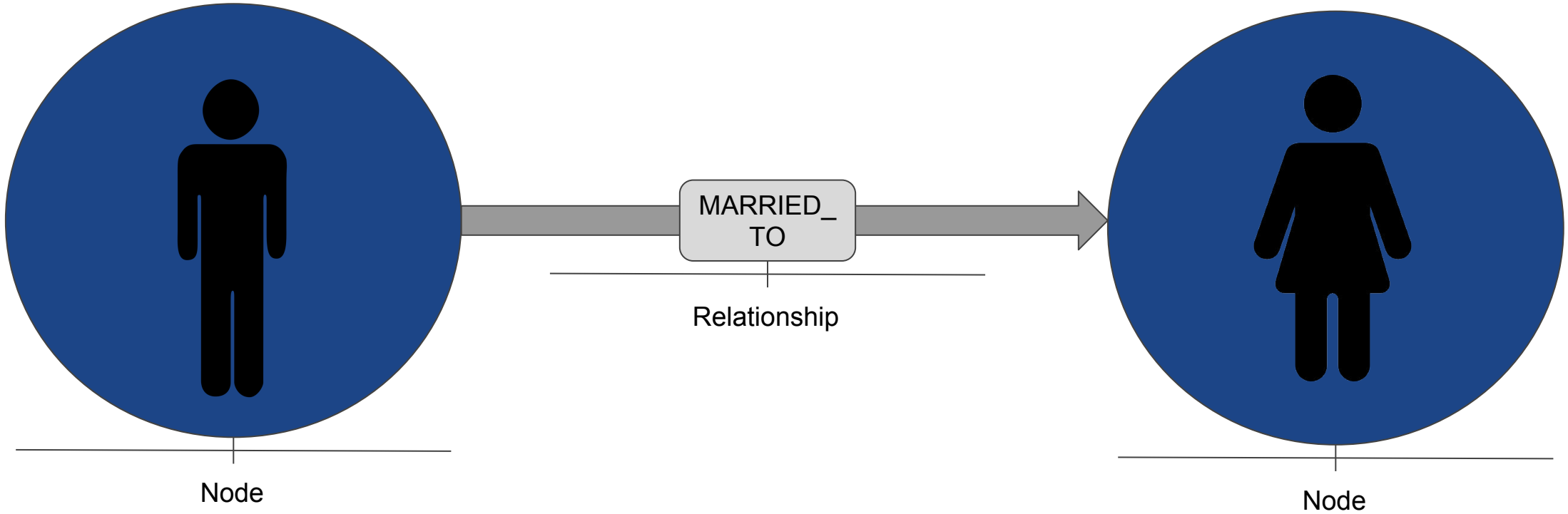
CYPHER

```
(a)-[LOVES]->(b)
```

Create - Property Graph Model

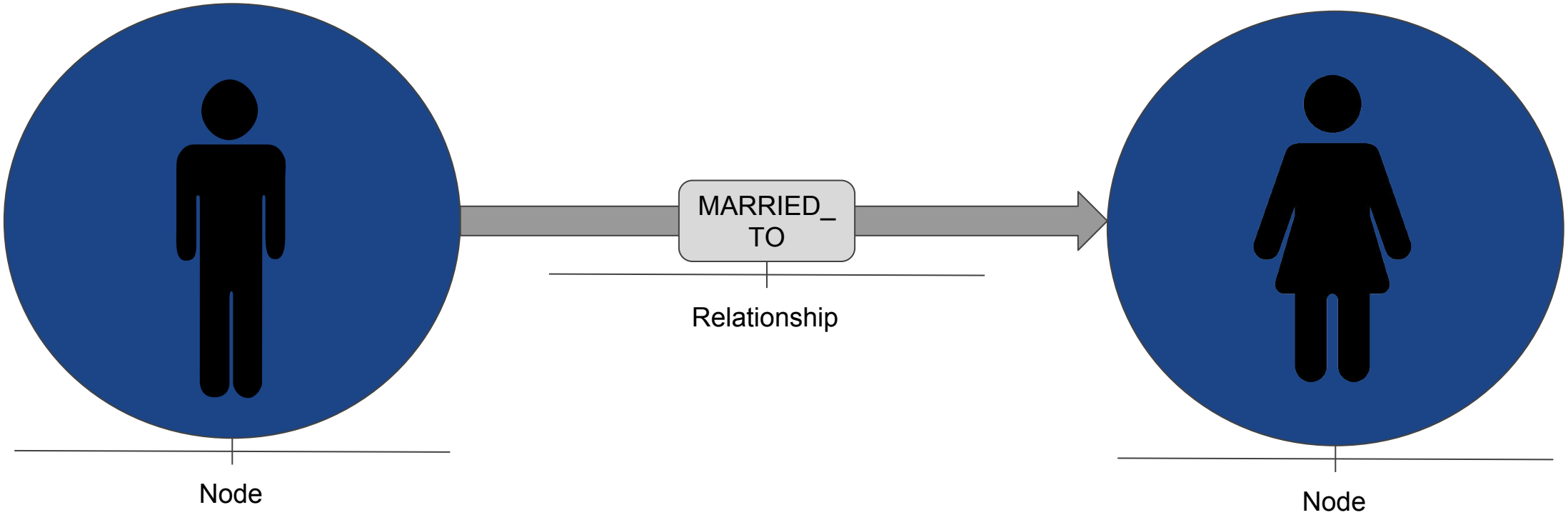


Create - Property Graph Model



CREATE (**:Person {name:"James"}**) - [**:MARRIED_TO**] -> (**:Person {name:"Mary"}**)

Create - Property Graph Model



CREATE (**:LABEL {PROPERTY:VALUE}**) - [**:RELATIONSHIP**] -> (**:LABEL {PROPERTY:VALUE}**)

Social Example

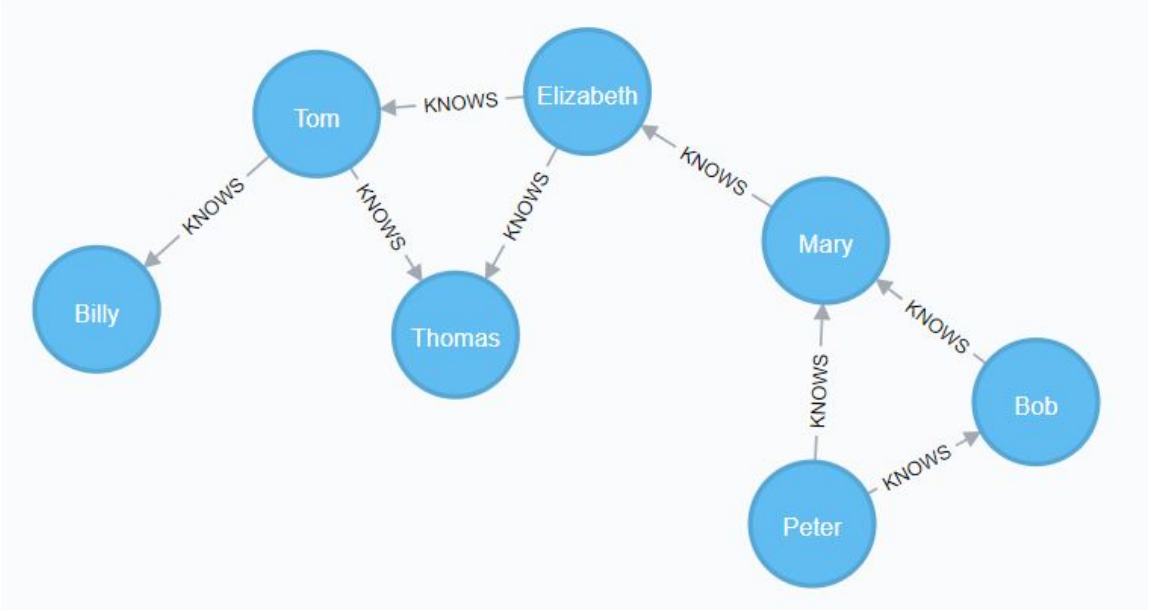
Social Graph - Create

```
CREATE
    (peter:Person {name:"Peter"}),
    (bob:Person {name:"Bob"}),
    (mary:Person {name:"Mary"}),
    (elizabeth:Person {name:"Elizabeth"}),
    (tom:Person {name:"Tom"}),
    (thomas:Person {name:"Thomas"}),
    (billy:Person {name:"Billy"}),

    (peter)-[:KNOWS]->(bob),
    (peter)-[:KNOWS]->(mary),
    (bob)-[:KNOWS]->(mary),
    (mary)-[:KNOWS]->(elizabeth),
    (elizabeth)-[:KNOWS]->(tom),
    (elizabeth)-[:KNOWS]->(thomas),
    (tom)-[:KNOWS]->(thomas),
    (tom)-[:KNOWS]->(billy)
```

Social Graph - Create

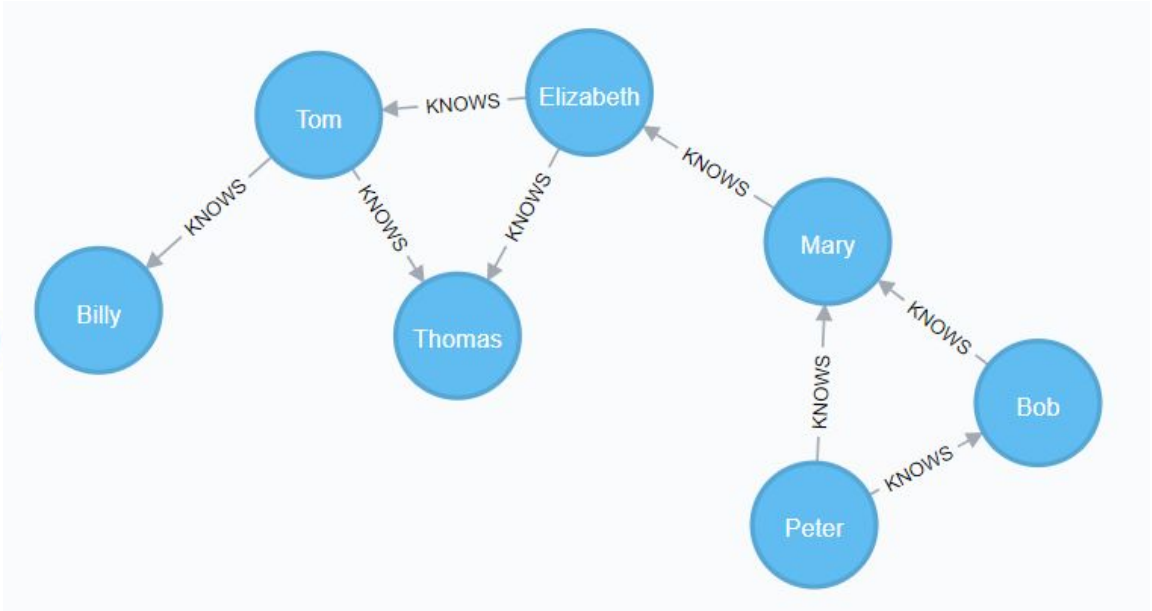
```
CREATE
    (peter:Person {name:"Peter"}),
    (bob:Person {name:"Bob"}),
    (mary:Person {name:"Mary"}),
    (elizabeth:Person {name:"Elizabeth"}),
    (tom:Person {name:"Tom"}),
    (thomas:Person {name:"Thomas"}),
    (billy:Person {name:"Billy"}),

    (peter)-[:KNOWS]->(bob),
    (peter)-[:KNOWS]->(mary),
    (bob)-[:KNOWS]->(mary),
    (mary)-[:KNOWS]->(elizabeth),
    (elizabeth)-[:KNOWS]->(tom),
    (elizabeth)-[:KNOWS]->(thomas),
    (tom)-[:KNOWS]->(thomas),
    (tom)-[:KNOWS]->(billy)
```

Result:



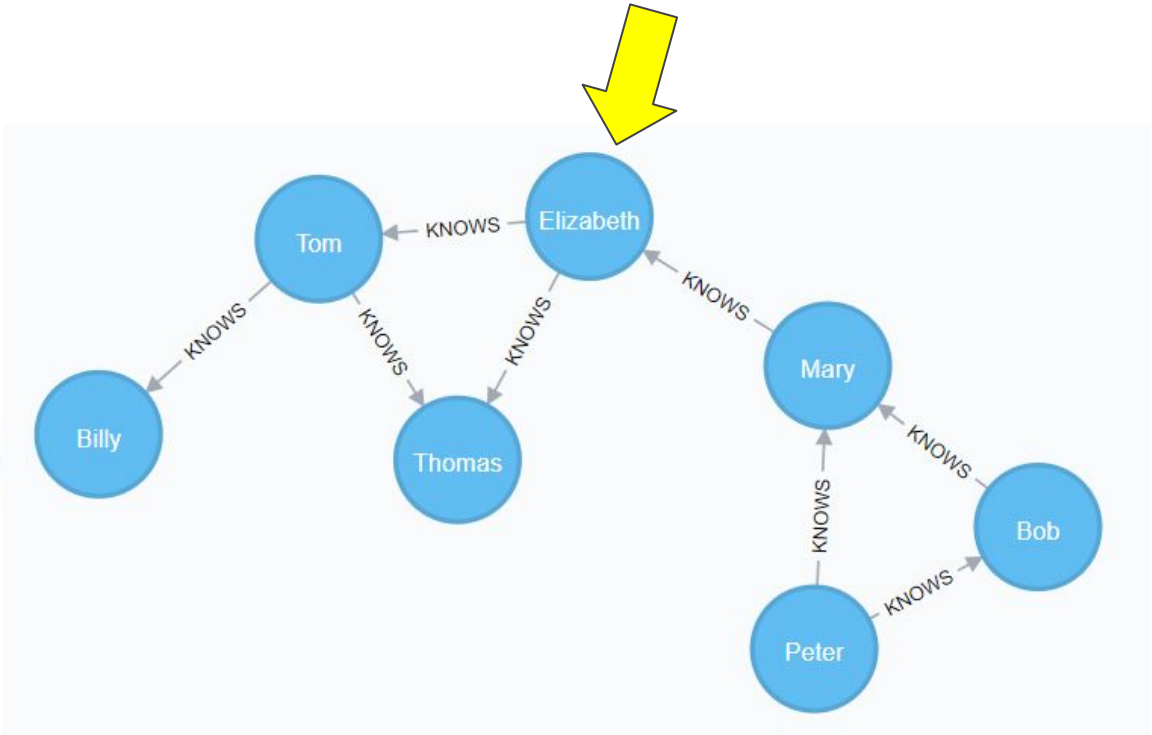
Social Graph - Friends of Peter's Friends

```
MATCH (person)-[:KNOWS]-(friend),
      (friend)-[:KNOWS]-(foaf)
WHERE person.name = "Peter"
AND NOT (person)-[:KNOWS]-(foaf)
RETURN foaf
```



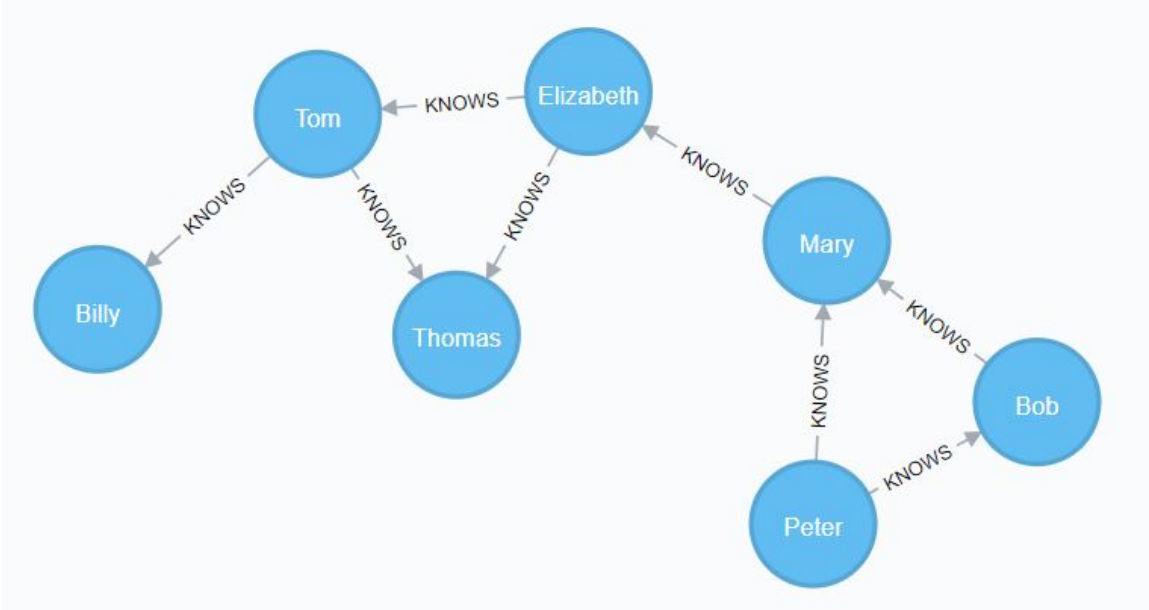
Social Graph - Friends of Peter's Friends

```
MATCH (person)-[:KNOWS]-(friend),
      (friend)-[:KNOWS]-(foaf)
WHERE person.name = "Peter"
AND NOT (person)-[:KNOWS]-(foaf)
RETURN foaf
```



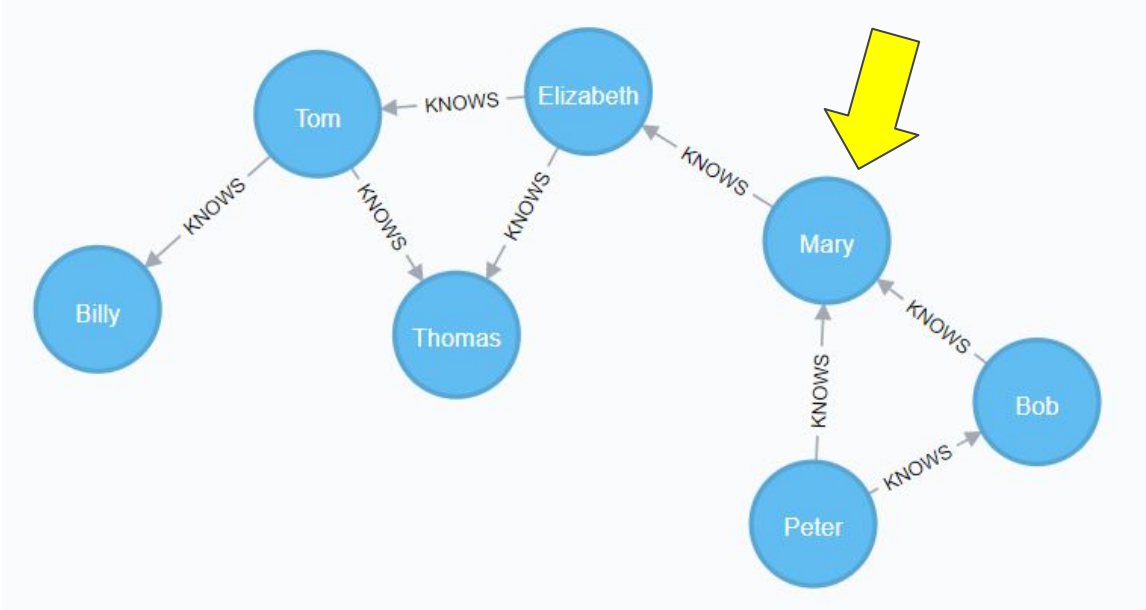
Social Graph - Common Friends

```
MATCH   (personA)-[:KNOWS]-(friend),
        (personB)-[:KNOWS]-(friend)
        WHERE personA.name="Peter"
        AND personB.name="Bob"
        RETURN friend
```



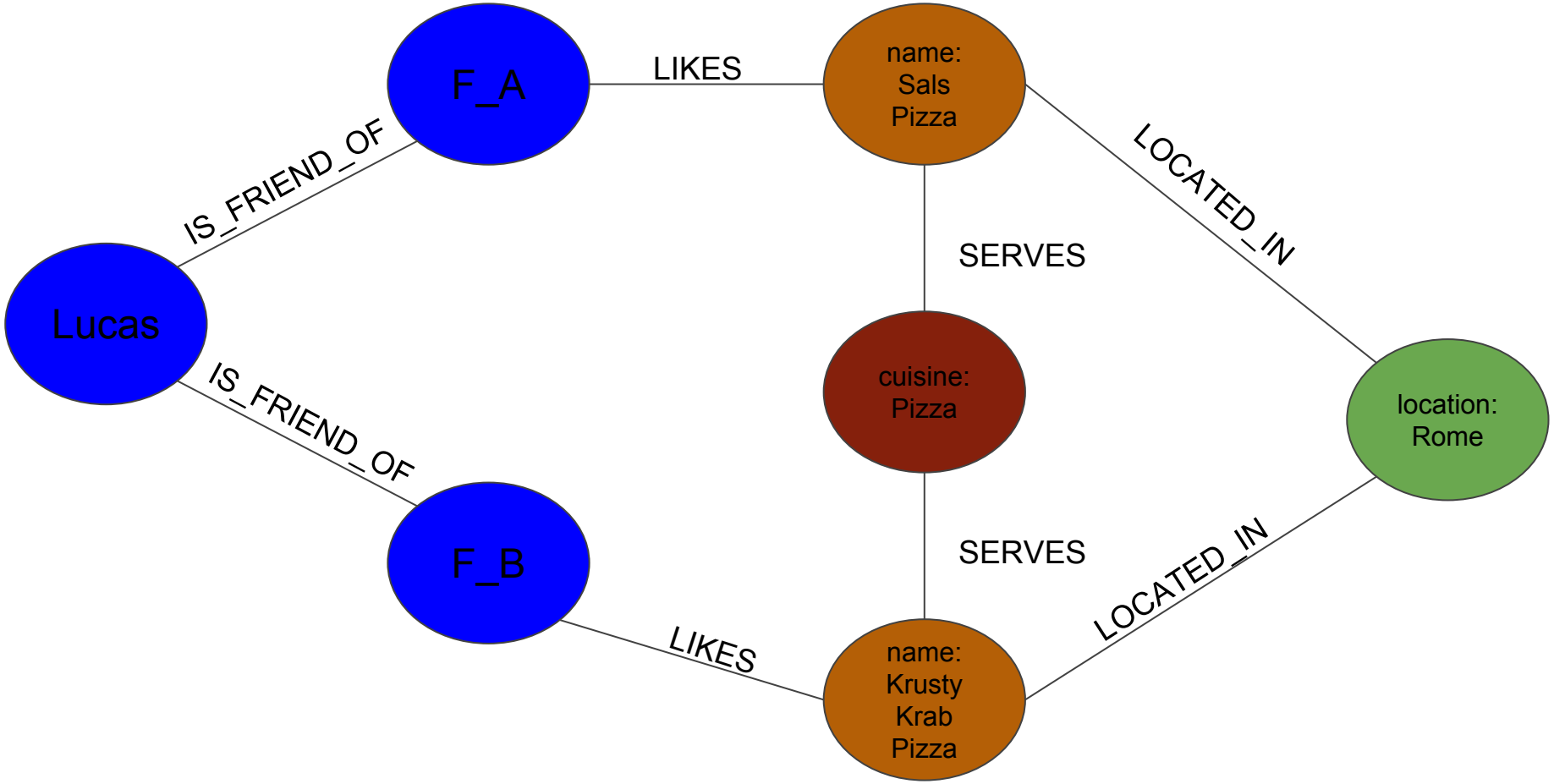
Social Graph - Common Friends

```
MATCH   (personA)-[:KNOWS]-(friend),
        (personB)-[:KNOWS]-(friend)
        WHERE personA.name="Peter"
        AND personB.name="Bob"
        RETURN friend
```



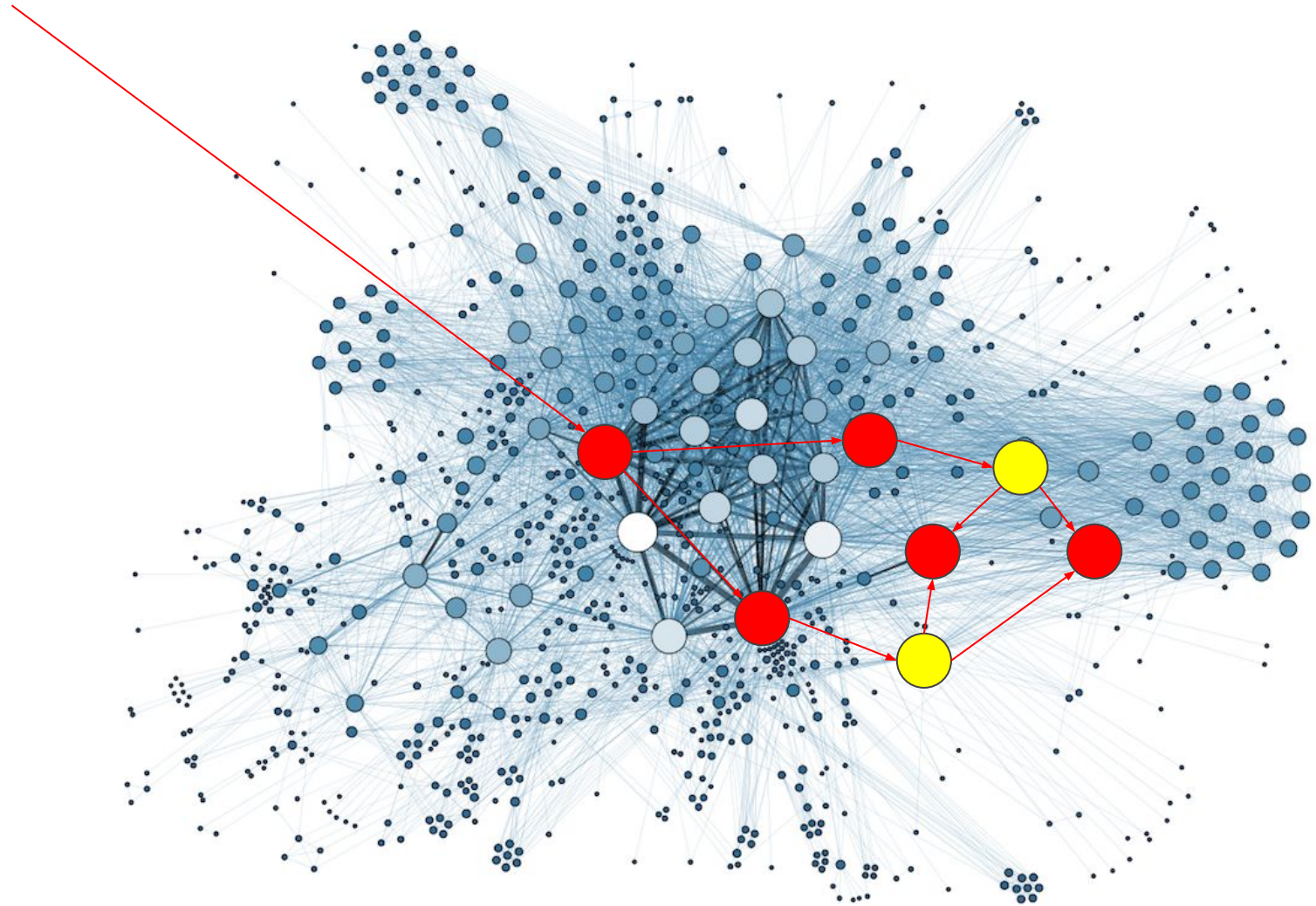
Restaurant Example


Pizza Restaurants in Rome, Rome my friends like
Pizza Restaurants in Rome, Rome my friends like


F_A
LIKES
name: Sals Pizza
IS_FRIEND_OF
LOCATED_IN
SERVES
Lucas
cuisine: Pizza
location: Rome
IS_FRIEND_OF
F_B
SERVES
LOCATED_IN
LIKES
name: Krusty Krab Pizza


```
MATCH   (person)-[:IS_FRIEND_OF]->(friend),
        (friend)-[:LIKES]->(restaurant),
        (restaurant)-[:LOCATED_IN]->(city),
        (restaurant)-[:SERVES]->(dish)
       WHERE person.name="Lucas"
       AND city.name="Rome"
       AND dish.name="Pizza"
       RETURN restaurant
```


Pizza Restaurants in Rome, Rome my friends like
Pizza Restaurants in Rome, Rome my friends like



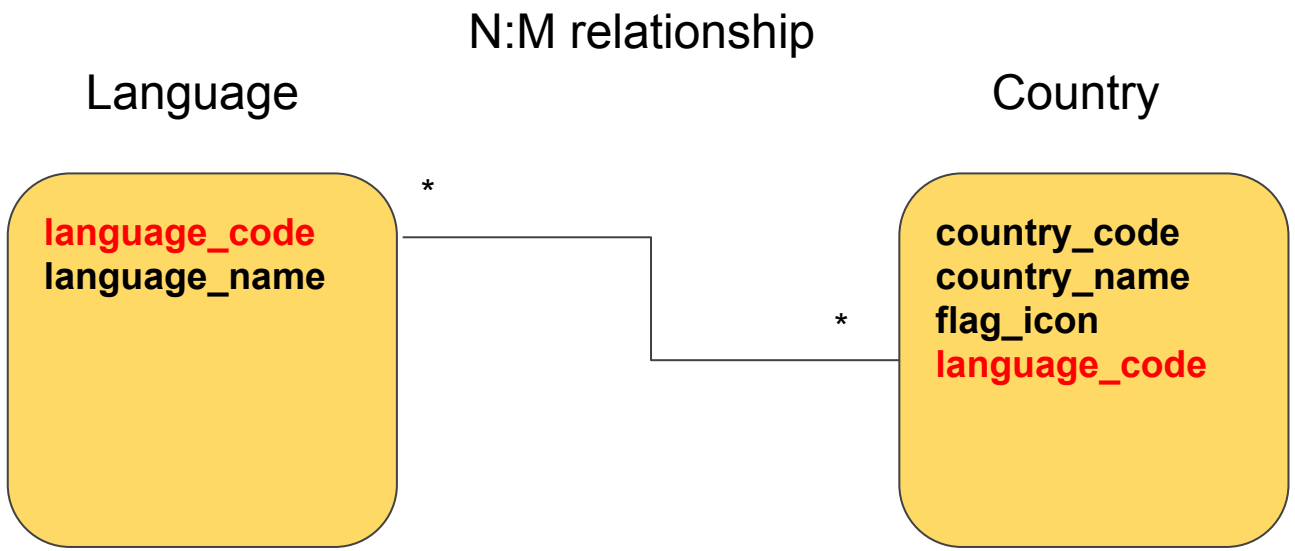
Assignment 1: First steps with Cypher

1. Start a docker container with an empty database with the following command:
   a. **docker run --publish=7474:7474 --publish=7687:7687 neo4j:latest**
2. open a browser and navigate to "**localhost:7474**"
3. **neo4j/neo4j** username/password
4. Choose a new password
5. Create a node of type "Person" with a property name that is set to your name.
6. Create a node of type "Hobby" with a property name that is set is to something that you like to do (e.g. "Programming", "Reading", ...).
7. Create a new relationship between those two nodes called "LIKES".
8. Create some friends of type "Person" with a property name that have the relationship "FRIENDS" with your own node.
9. Find all your friends in the database.
10. Give at least one friend a hobby.

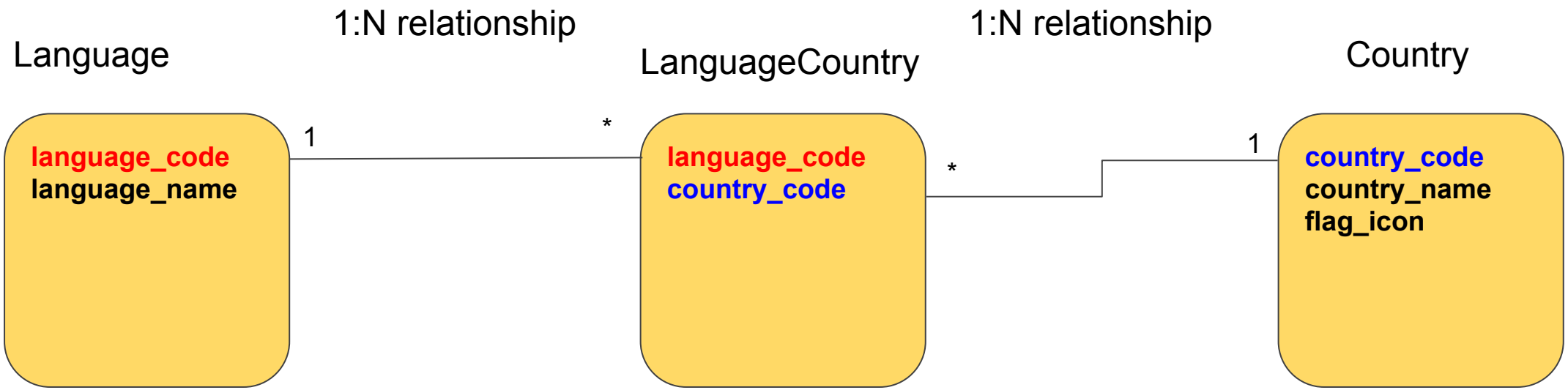
RDBMS vs. Graph DB

What language do they speak here?

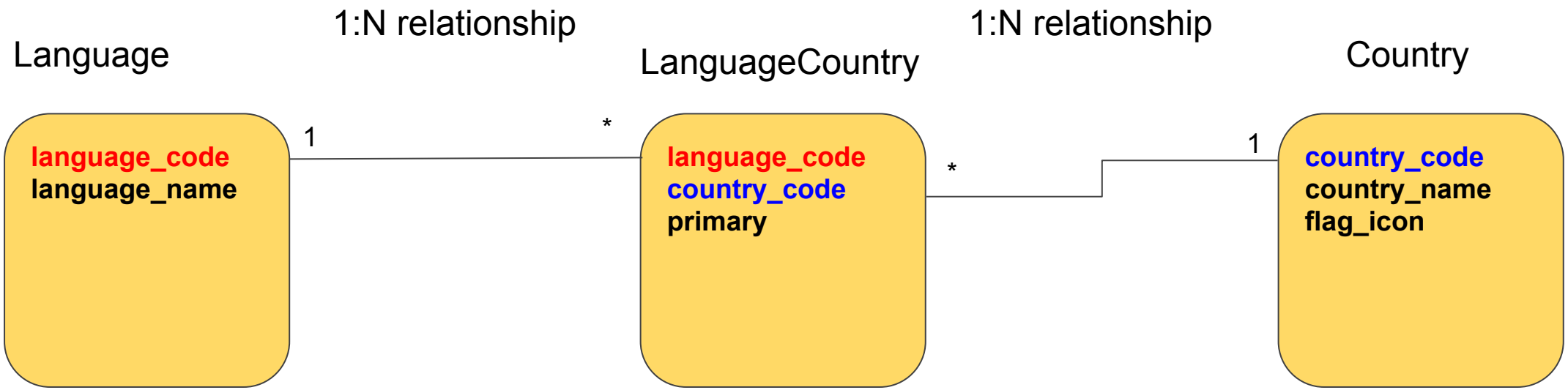
Languages in Tables (1)



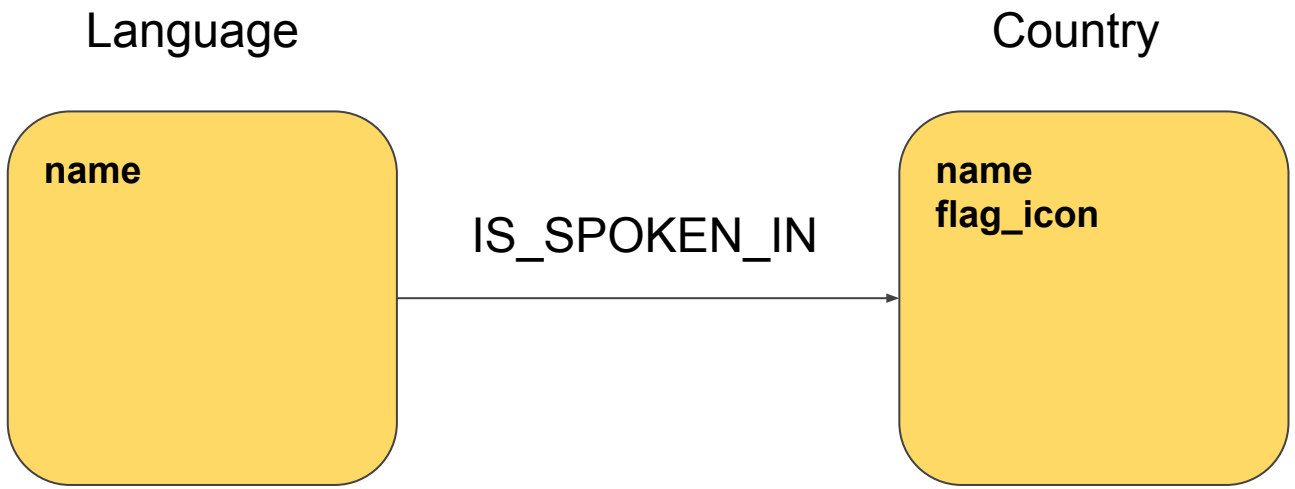
Languages in Tables (2)



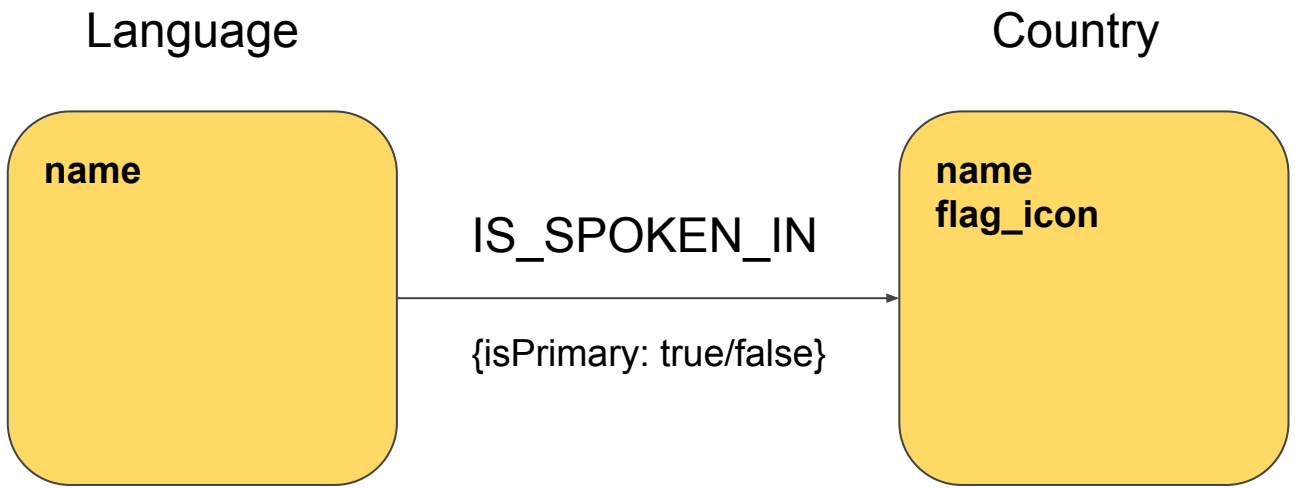
Languages in Tables (3)



Languages in Graphs (1)



Languages in Graphs (2)



Performance

- Performance definition:
  - With rising nodes same effort for one “step” in the graph
- Social Graph with ~50 “friends” for each user
- Search for 4th degree friends after warmup of the database

Performance

- Performance definition:
  - With rising nodes same effort for one “step” in the graph
- Social Graph with ~50 “friends” for each user
- Search for 4th degree friends after warmup of the database

DB	USERS	TIME
Relational	1000	2000 ms
Neo4j	1000	2 ms
Neo4j	1000000	2 ms

Assignment 2: Beer DB

1. clone the github repository "**git clone** https://github.com/sebivenlo/Neo4JGraphDB.git"
2. navigate to the "assignment2" folder and run the following command:

   Linux + Mac:

```
docker run \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/(path to assignment2 folder)/neo4j/data:/data \
neo4j:latest
```

   Windows Users: Use the absolute path **"C:/../assignment2/neo4j/data:/data"**

   If all else fails:

   start an empty neo4j database using the command from assignment 1 and copy paste

   the data.cypher script in there.

   The rest of the assignment 2 can be found in the readme.adoc on github
   https://github.com/sebivenlo/Neo4JGraphDB

Functions CYPHER

- Different groups of functions
  - Predicate functions(all,any,exists)
  - Scalar functions(head,last,size)
  - Aggregate functions(avg,count,max)
  - List functions(keys,reverse,tail)
  - Numeric Mathematical functions(abs,floor,round)
  - Logarithmic Mathematical functions(e,exp,log)
  - Trigonometric Mathematical functions(sin,cos,pi)
  - String functions(substring,split,toUpper)
  - Date functions(date,duration,localtime)
- shortestPath((Node1)-[reltype]-(Node2))
  - assignable: p=shortestPath
  - p>1 means a path longer than being directly related

Assignment 3 - Queries with functions

1. Open the Neo4J Browser as shown in Assignment 1
2. Delete the old database by executing “MATCH (n) DETACH DELETE n”
3. In the Neo4J Browser, execute “:play movies”
4. Click on the script shown in the feed
5. Execute the script
6. Run “MATCH (n) RETURN n” to get an Overview of the data
7. Look at the readme of this workshops repository to see the queries of this assignment

Java Part MIKE

Thank you for listening