Intro to Neo4j and Graph Databases

Edited by: Muazzam Siddiqui

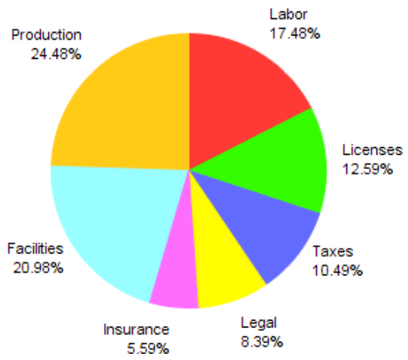
Original Content: **William Lyon**

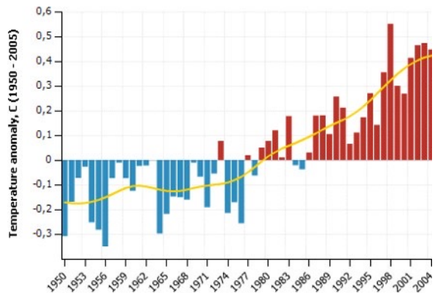
Presented at:**Neo4j Webinar March 2016**

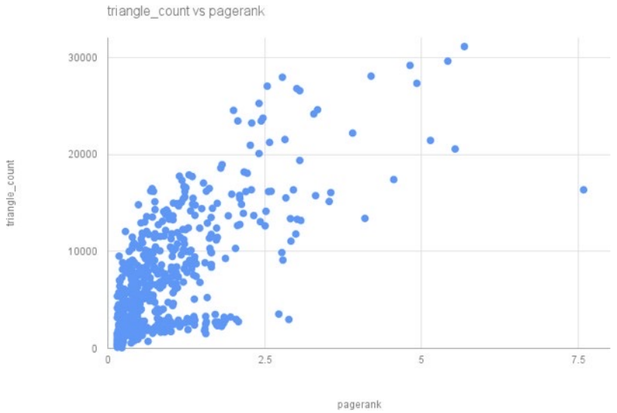
Agenda

- What is a graph [database]?
- Use cases - why graphs?
- Neo4j product overview
  - Labeled property graph data model
  - Cypher query language
- RDBMS to graph
- Resources
- Questions?

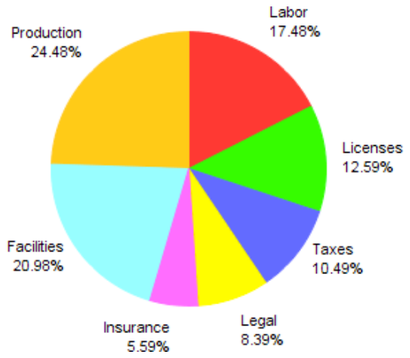
Chart

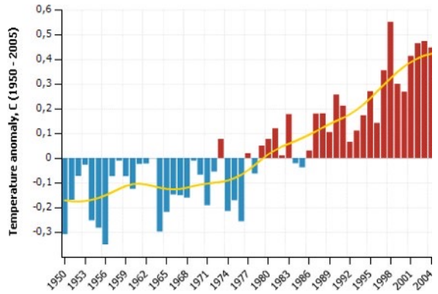

Labor
17.48%
Licenses
12.59%
Taxes
10.49%
Legal
8.39%
Insurance
5.59%
Facilities
20.98%
Production
24.48%

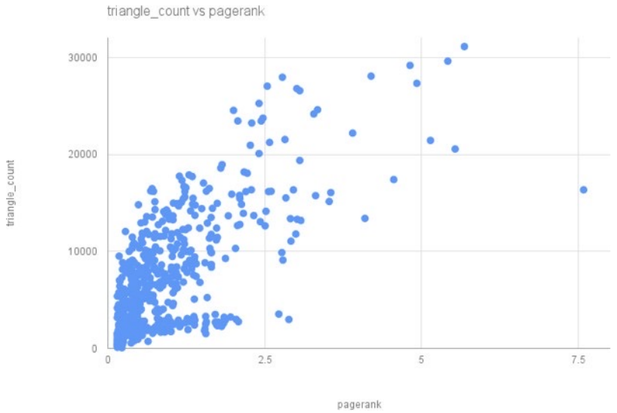

Temperature anomaly, C (1950 - 2005)
0,6
0,5
0,4
0,3
0,2
0,1
0
-0,1
-0,2
-0,3
1950
1953
1956
1959
1962
1965
1968
1971
1974
1977
1980
1983
1986
1989
1992
1995
1998
2001
2004


triangle_count vs pagerank
triangle_count
30000
20000
10000
0
0
2.5
5
7.5
pagerank

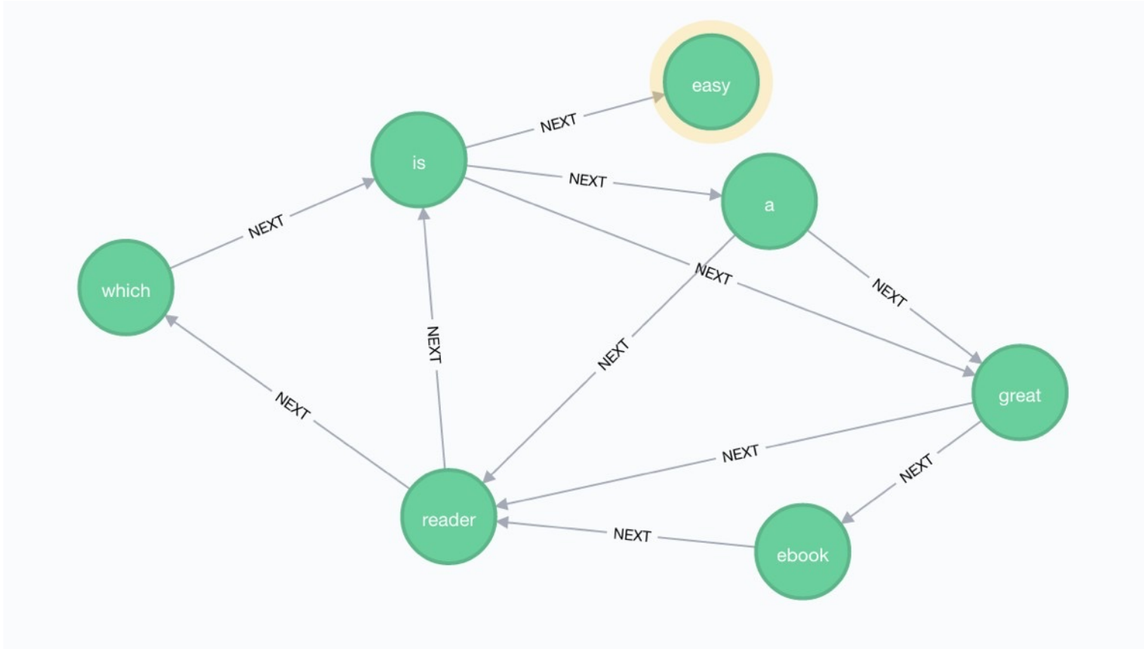
Chart

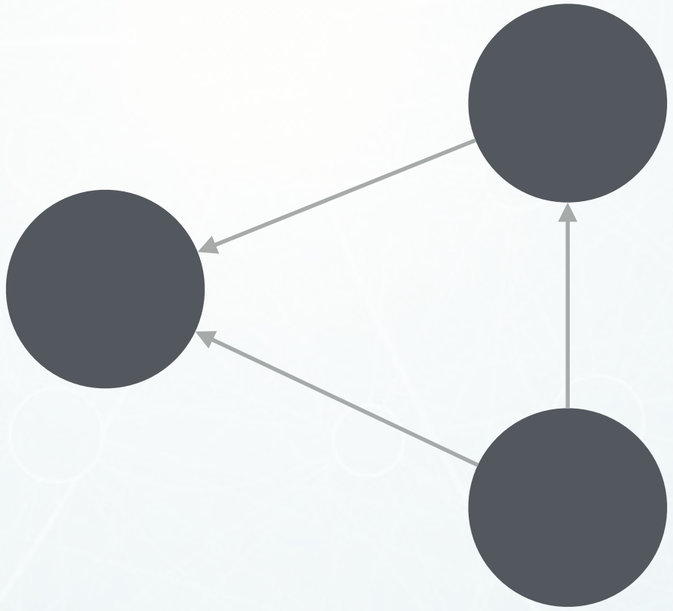

Production
24.48%
Labor
17.48%
Licenses
12.59%
Taxes
10.49%
Legal
8.39%
Insurance
5.59%
Facilities
20.98%


Temperature anomaly, C (1950 - 2005)
0,6
0,5
0,4
0,3
0,2
0,1
0
-0,1
-0,2
-0,3
1950
1953
1956
1959
1962
1965
1968
1971
1974
1977
1980
1983
1986
1989
1992
1995
1998
2001
2004

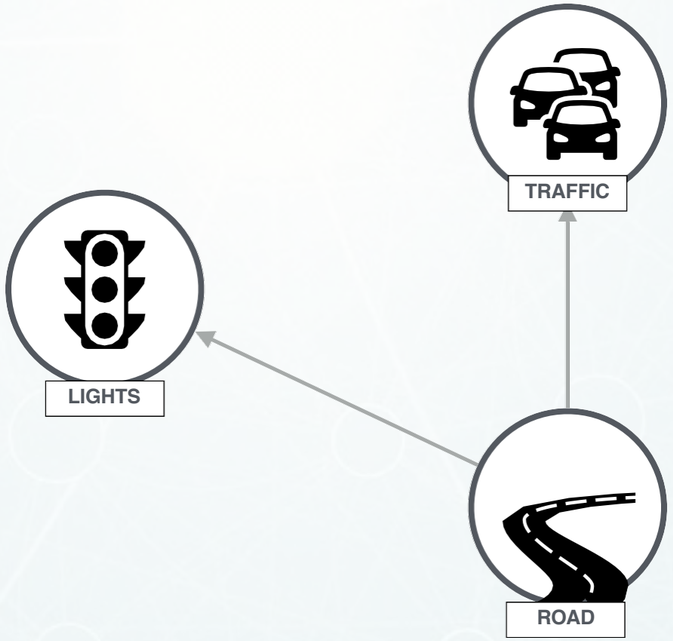

triangle_count vs pagerank
triangle_count
30000
20000
10000
0
0
2.5
5
7.5
pagerank

Graph

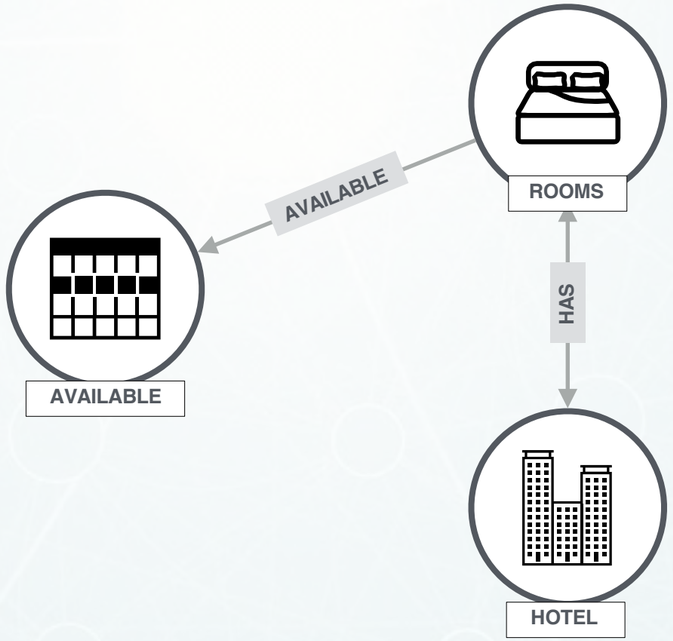

easy
is
a
which
great
reader
ebook
NEXT



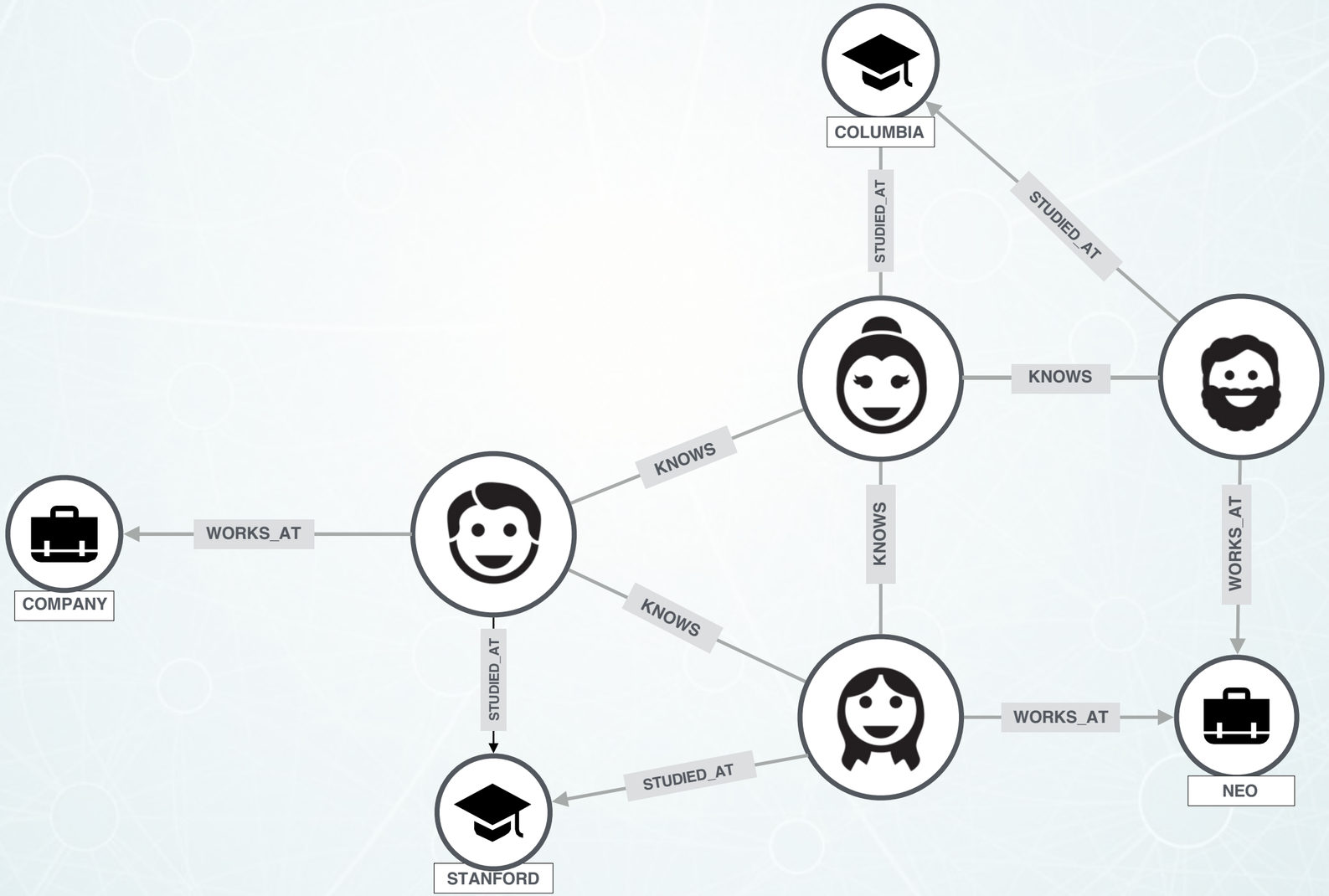
A Graph Is Connected Data



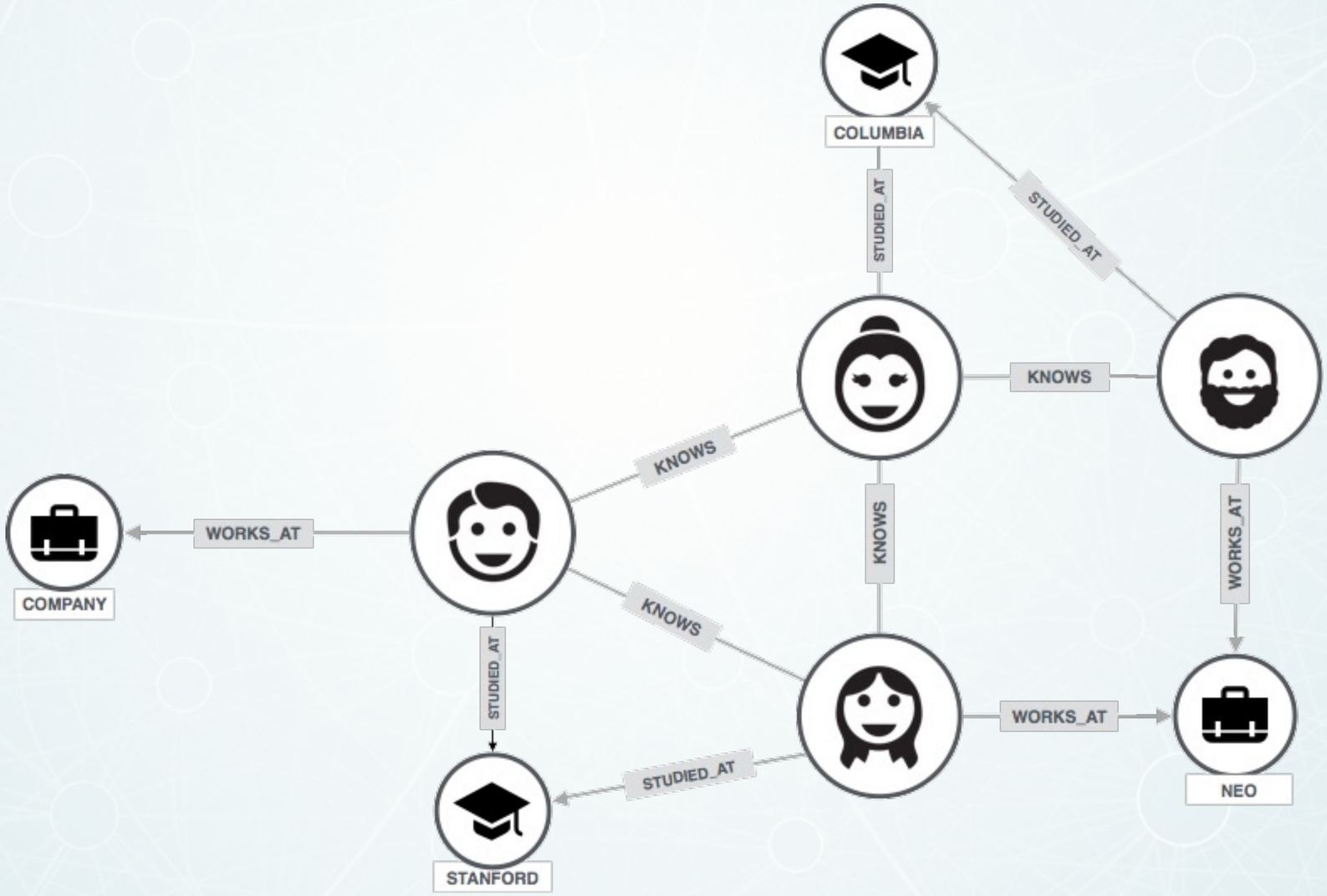
A Graph Is Connected Data



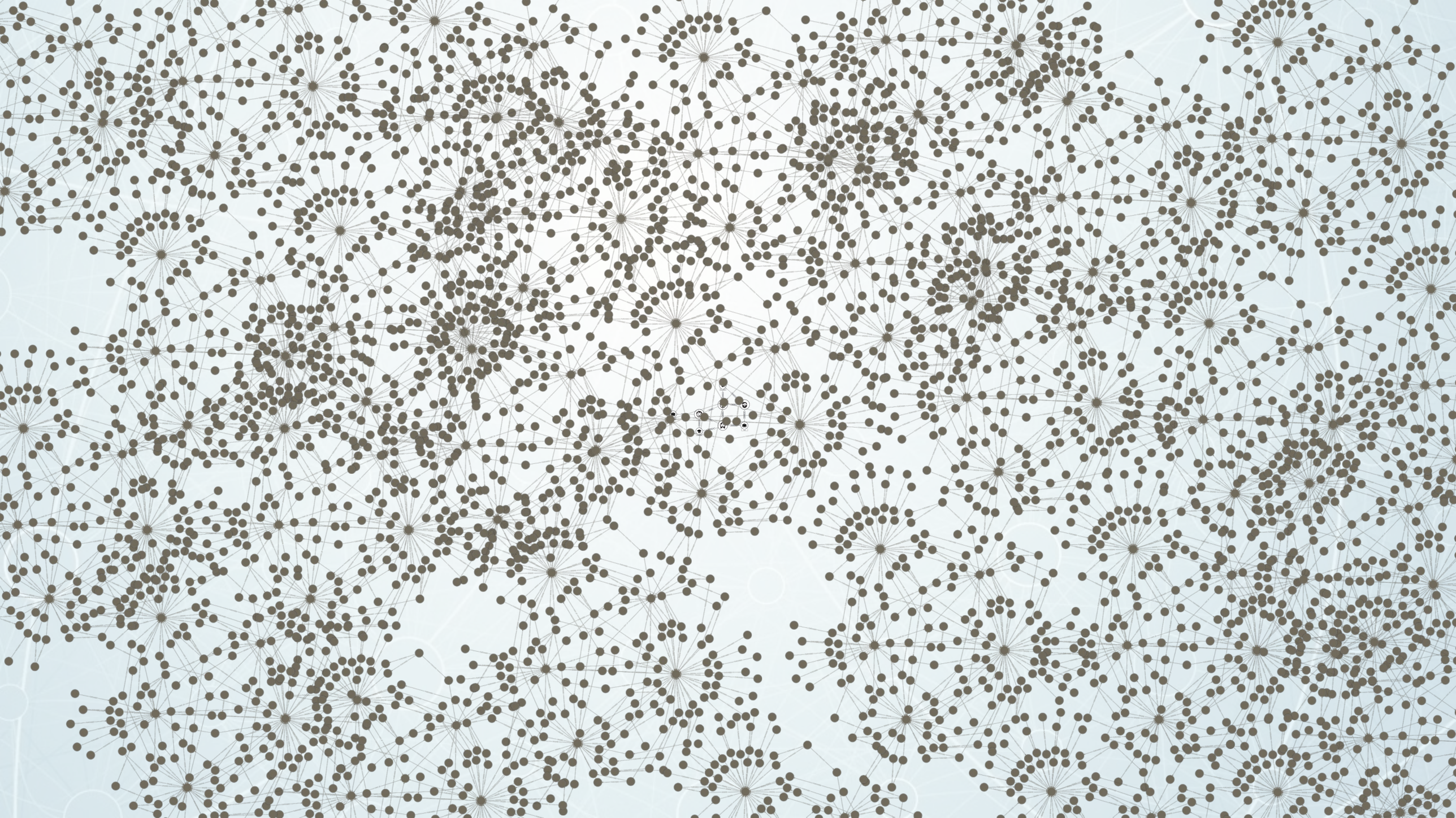
A Graph Is Connected Data



A Graph Is Connected Data

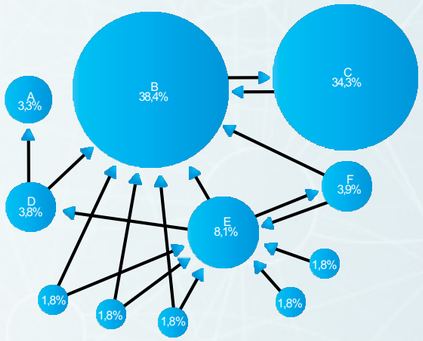


A Graph Is Connected Data

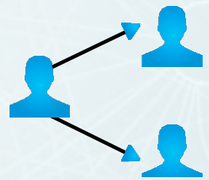


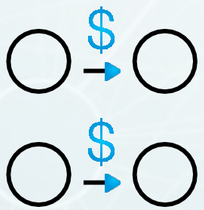
Use of Graphs has created some of the most successful companies in the world

Google

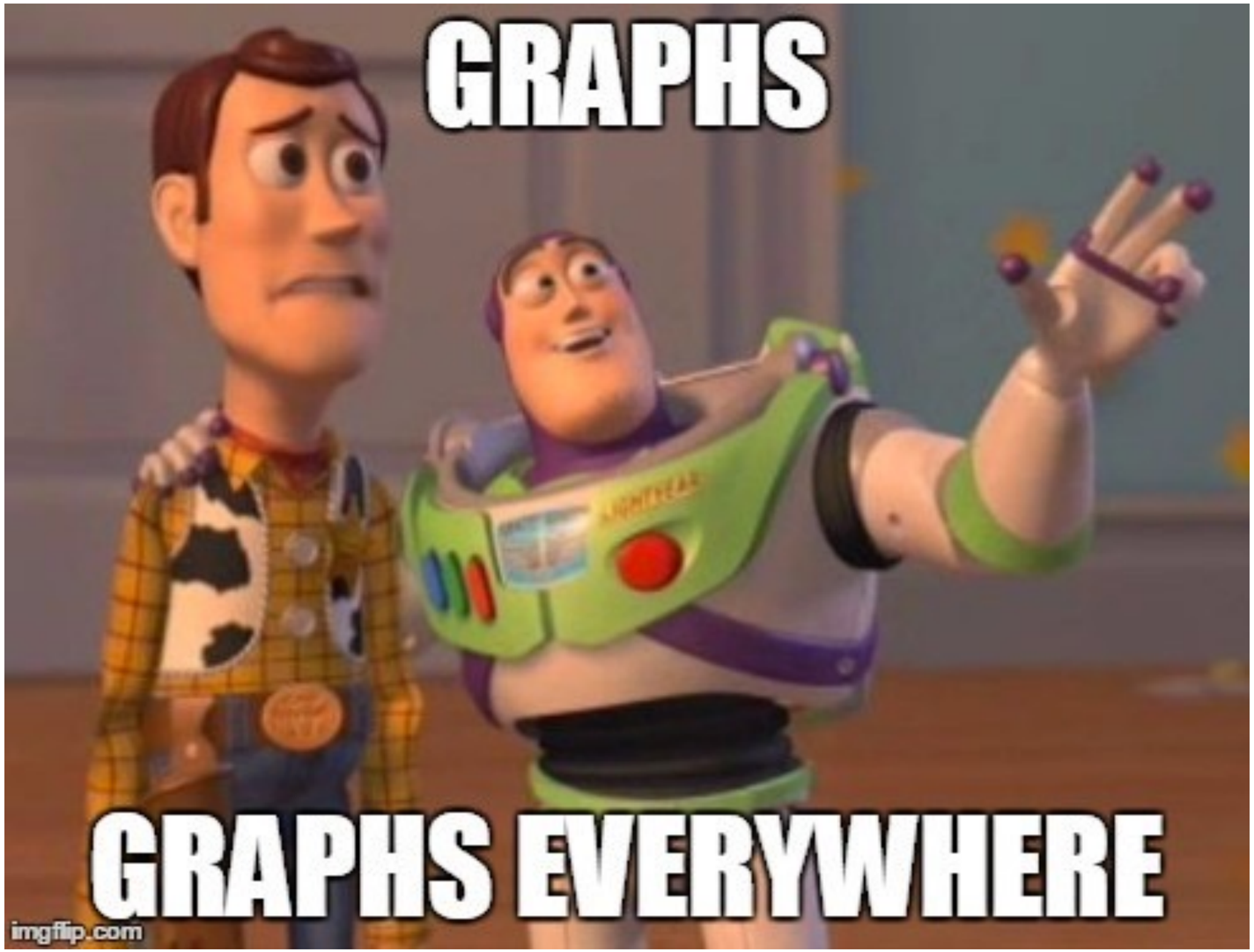


LinkedIn





PayPal


GRAPHS
GRAPHS EVERYWHERE
imgflip.com

NEO4j (Graph DBs) USE CASES

Real Time Recommendations

Graph Based Search

Network & IT-Operations

NEO4j USE CASES

Real Time Recommendations

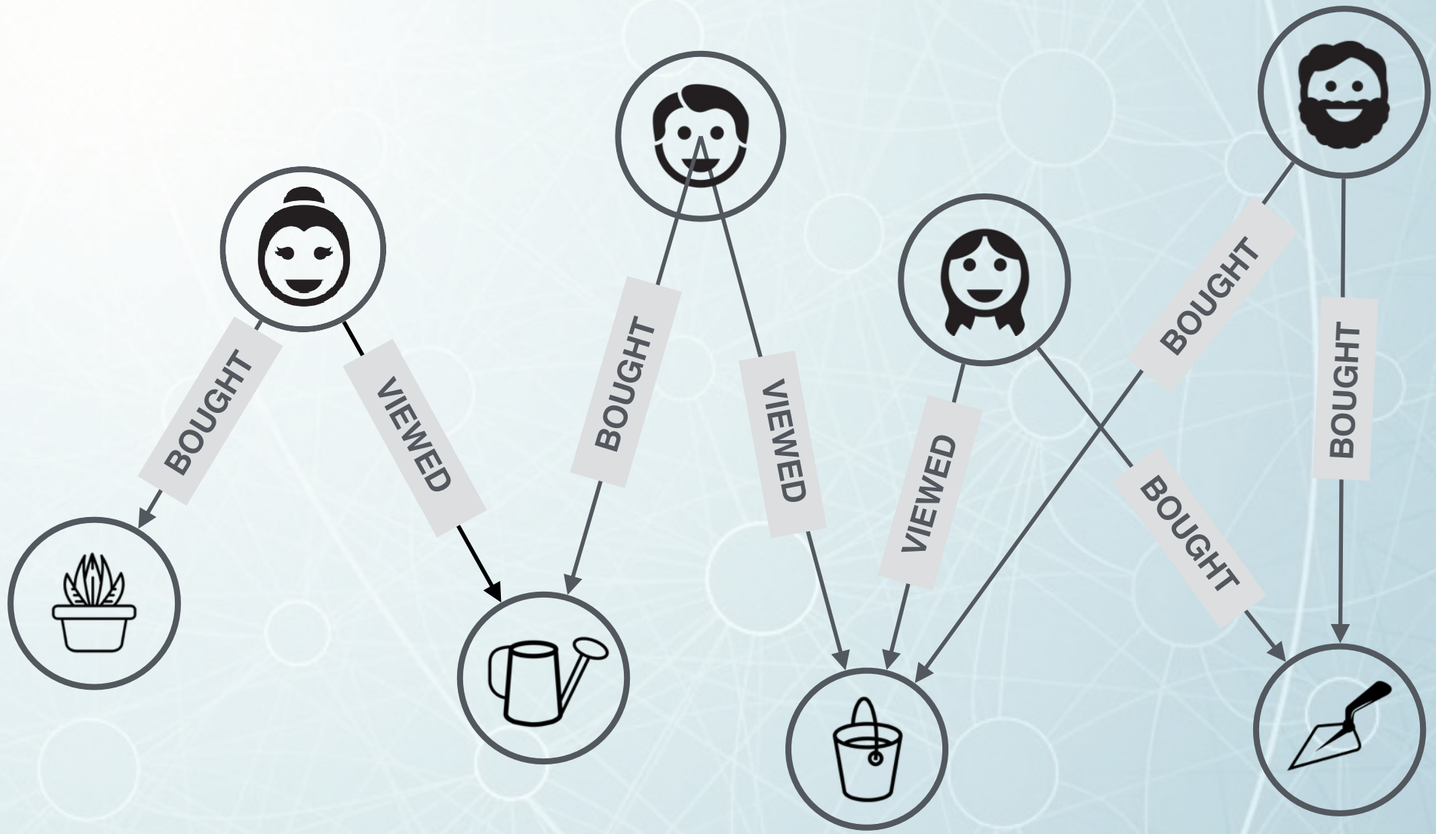
Graph Based Search

Network & IT-Operations

Identity & Access Management

neo4j

GRAPH THINKING:
Real Time Recommendations

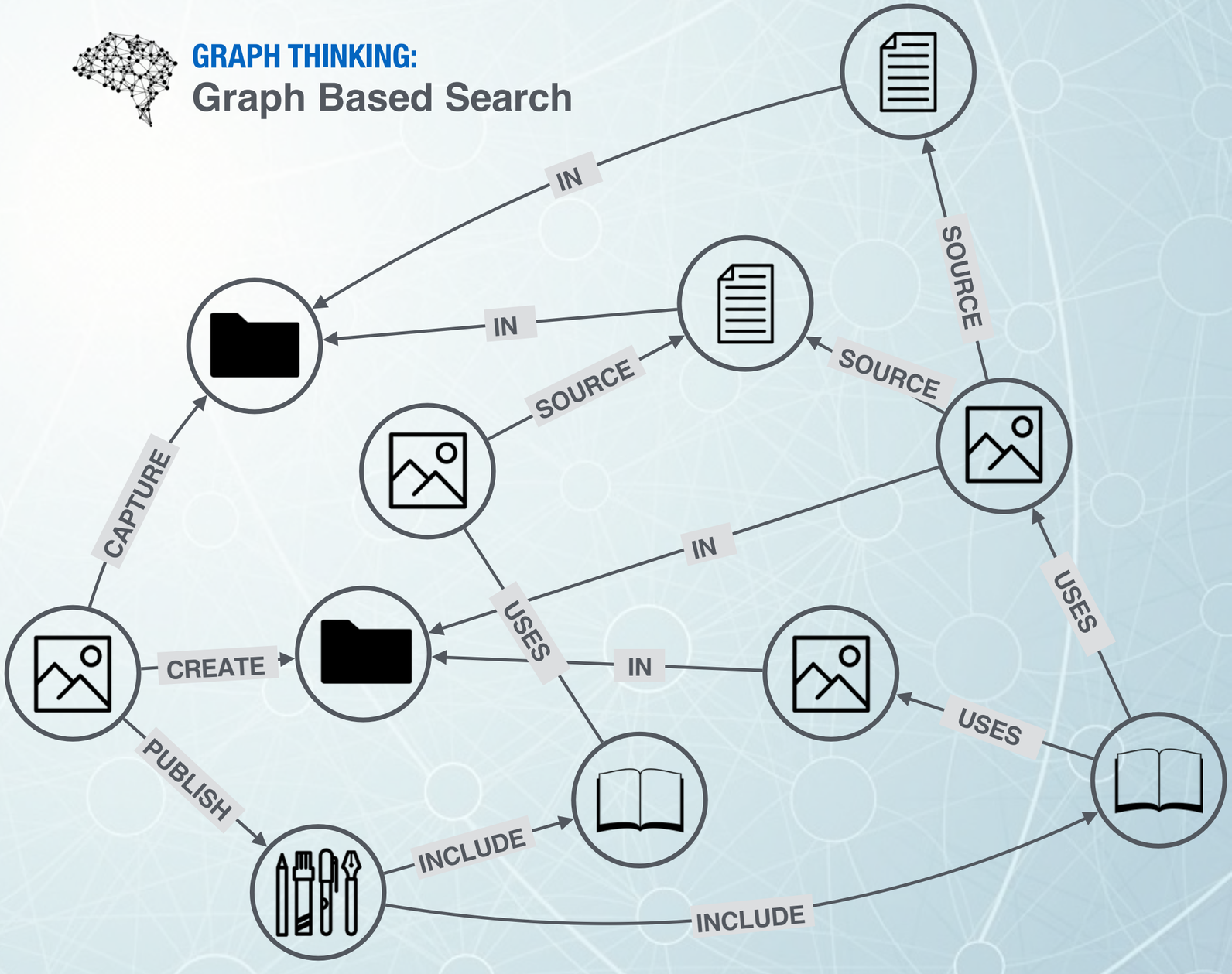


NEO4j USE CASES

Real Time Recommendations

Graph Based Search

Network & IT-Operations



neo4j

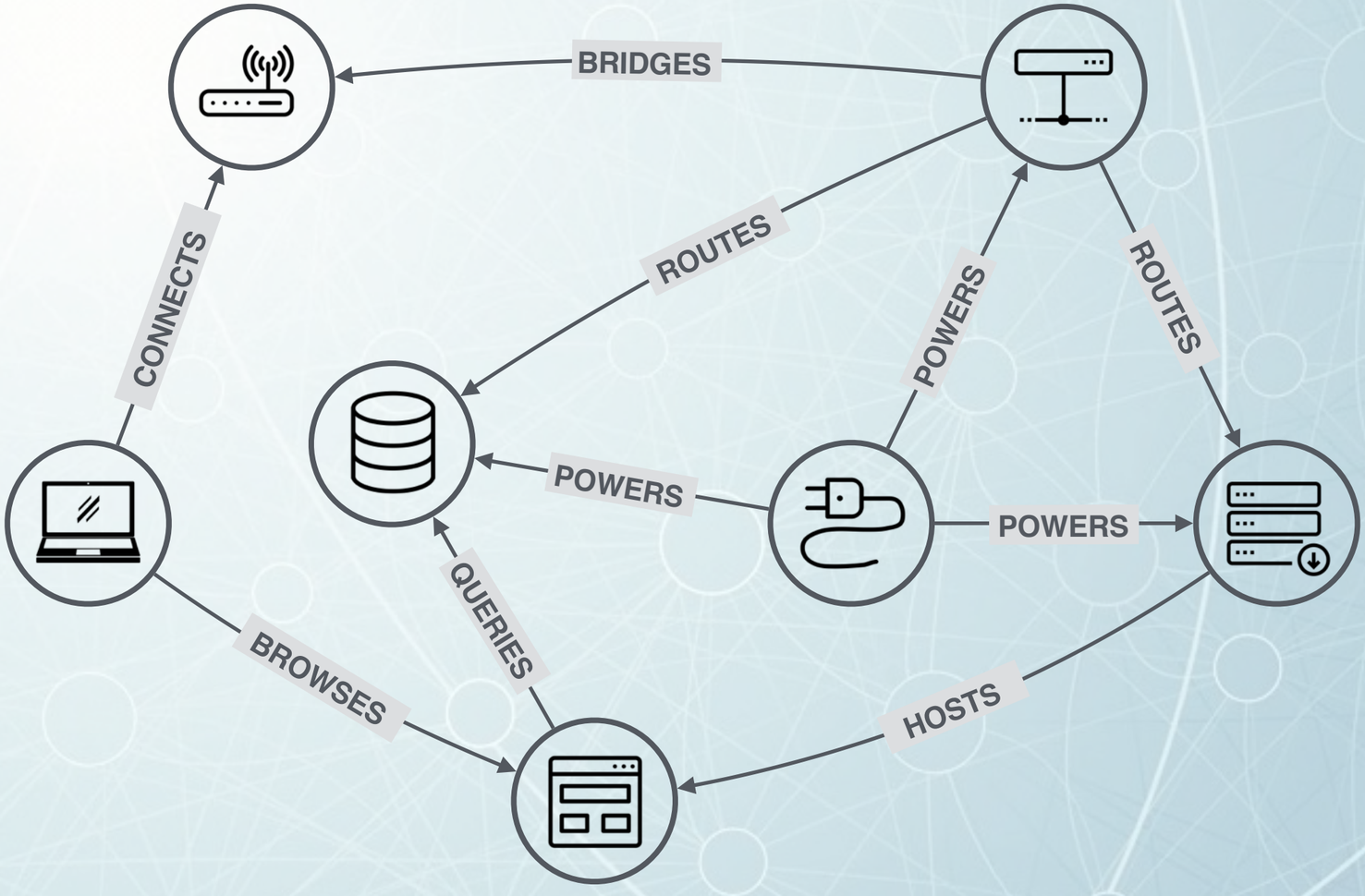
NEO4j USE CASES

Real Time Recommendations

Graph Based Search

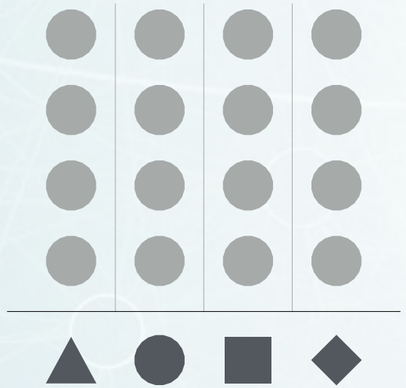
Network & IT-Operations

GRAPH THINKING:
Network & IT-Operations



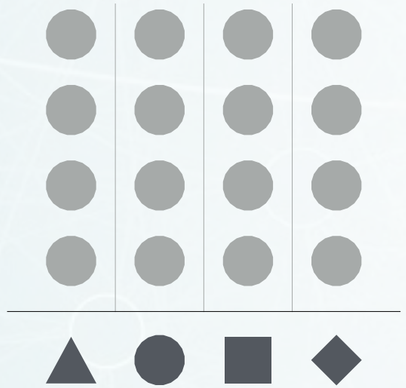
neo4j

A way of representing data



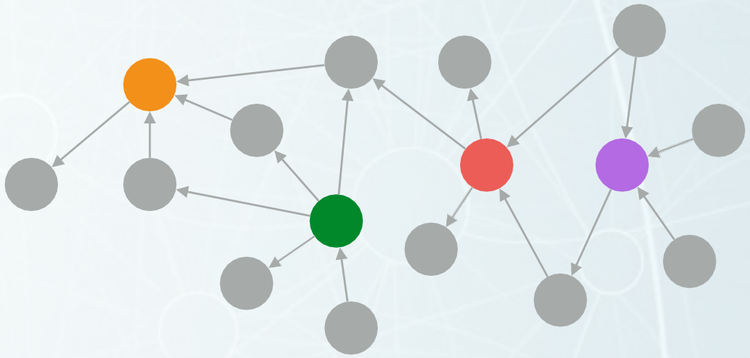
Relational Database

A way of representing data



Relational Database

Graph Database



Good for:

- Well-understood data structures that don’t change too frequently
- Known problems involving discrete parts of the data, or minimal connectivity

Good for:

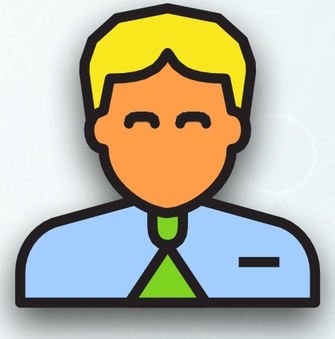
- Dynamic systems: where the data topology is difficult to predict
- Dynamic requirements: the evolve with the business
- Problems where the relationships in data contribute meaning & value

THE PROPERTY GRAPH DATA MODEL

Ann Loves Dan

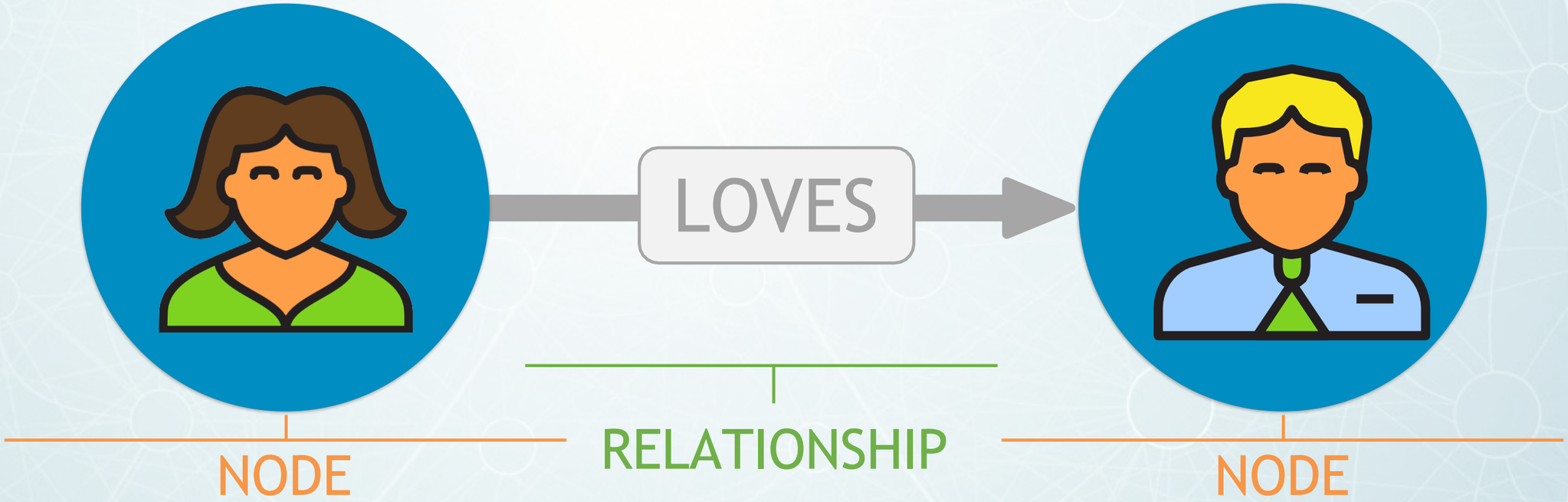
Ann

Loves

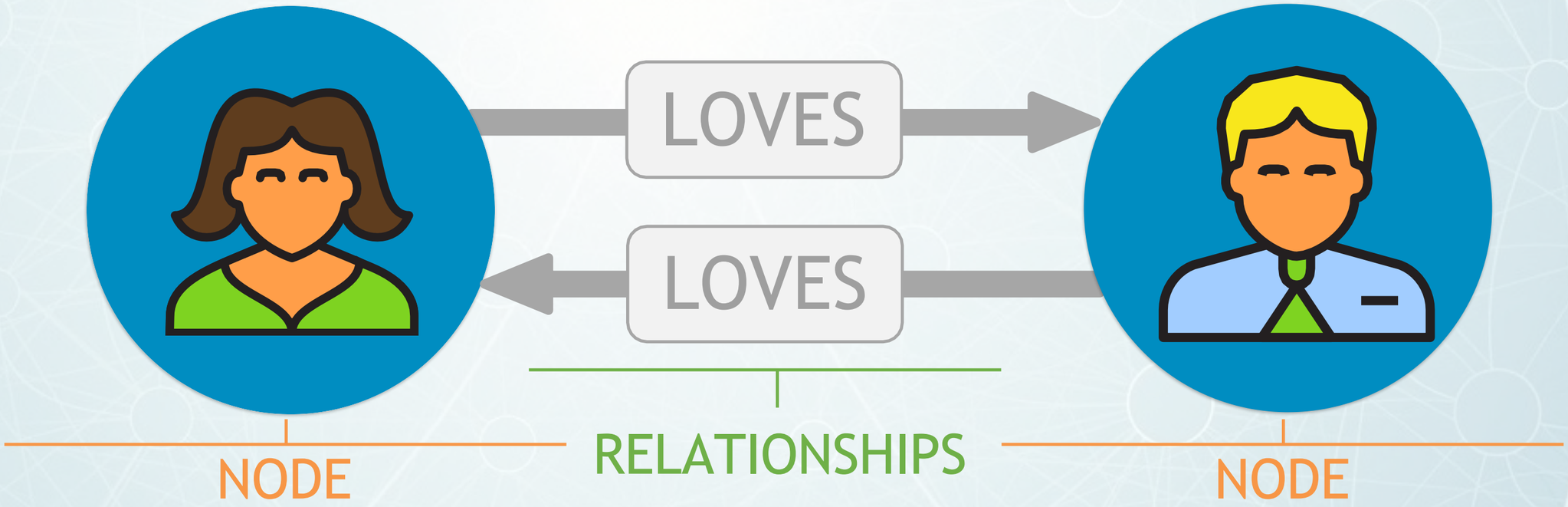


Dan

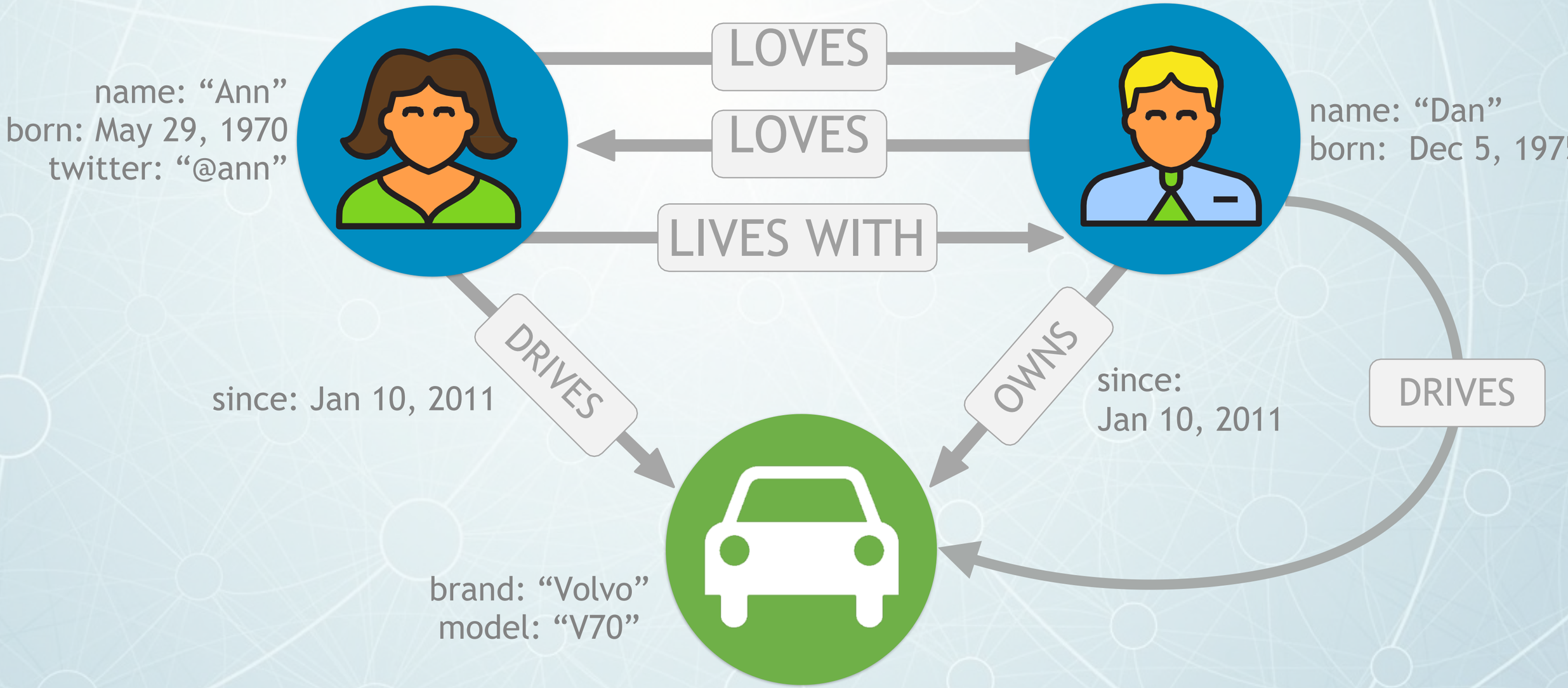
Ann Loves Dan



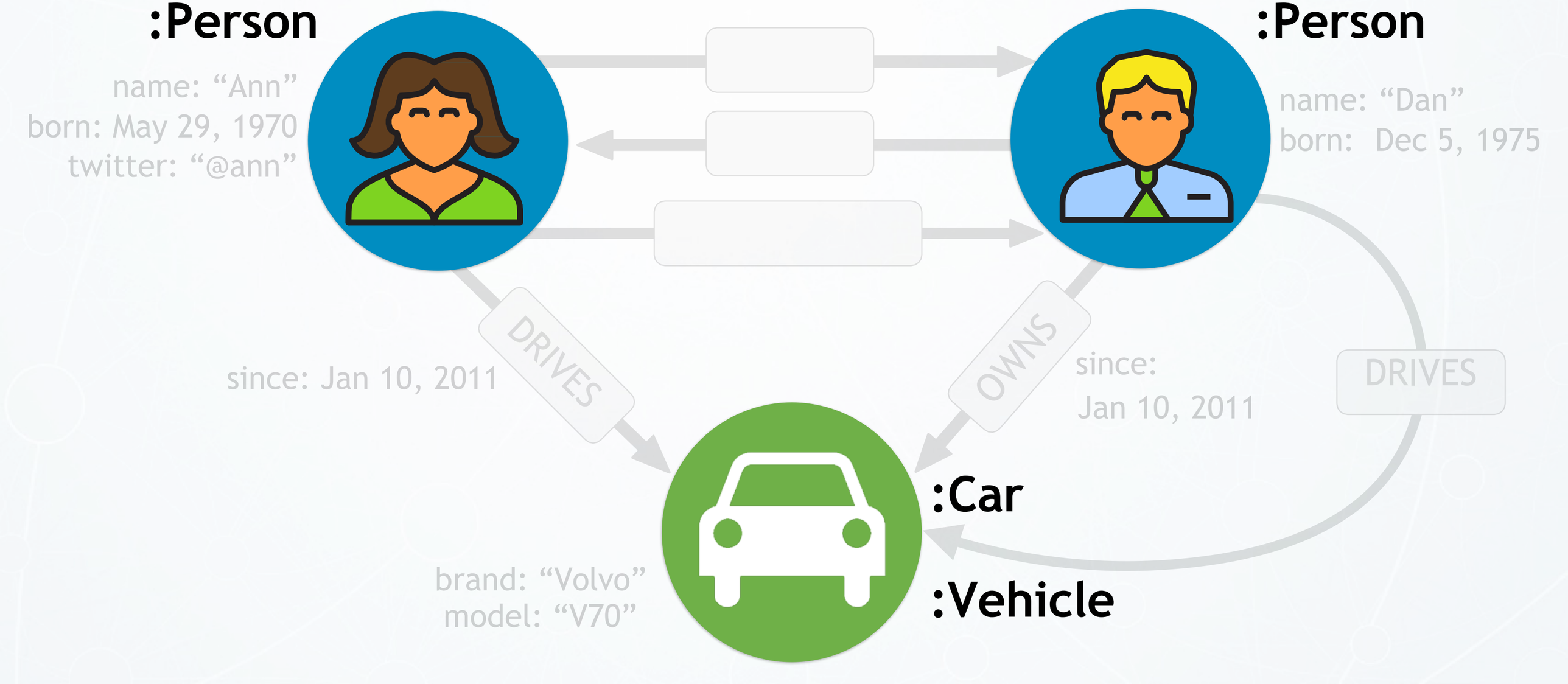
Relationships are Directional



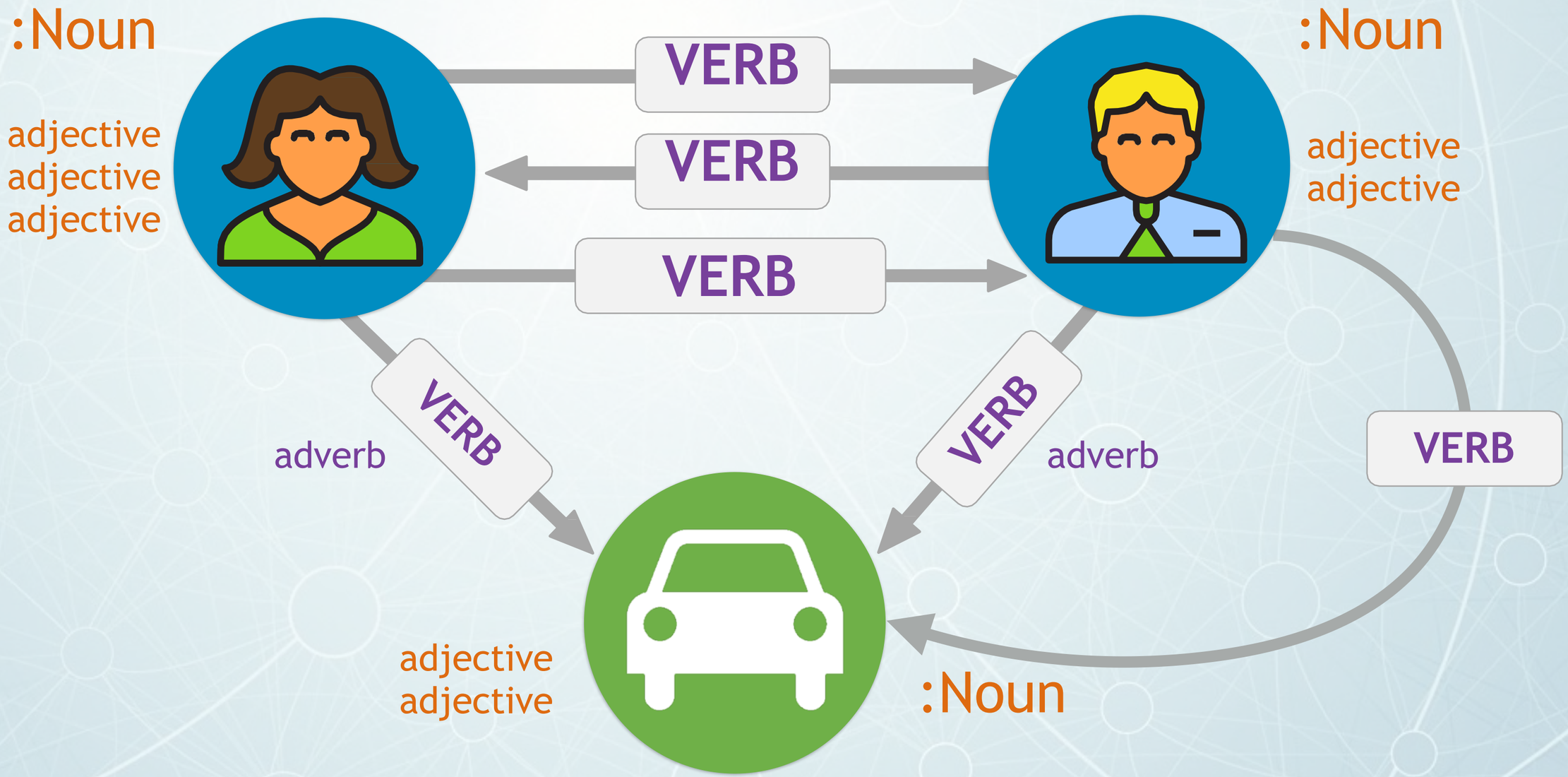
Detailed Property Graph



Labeled Property Graph



Mapping to Languages



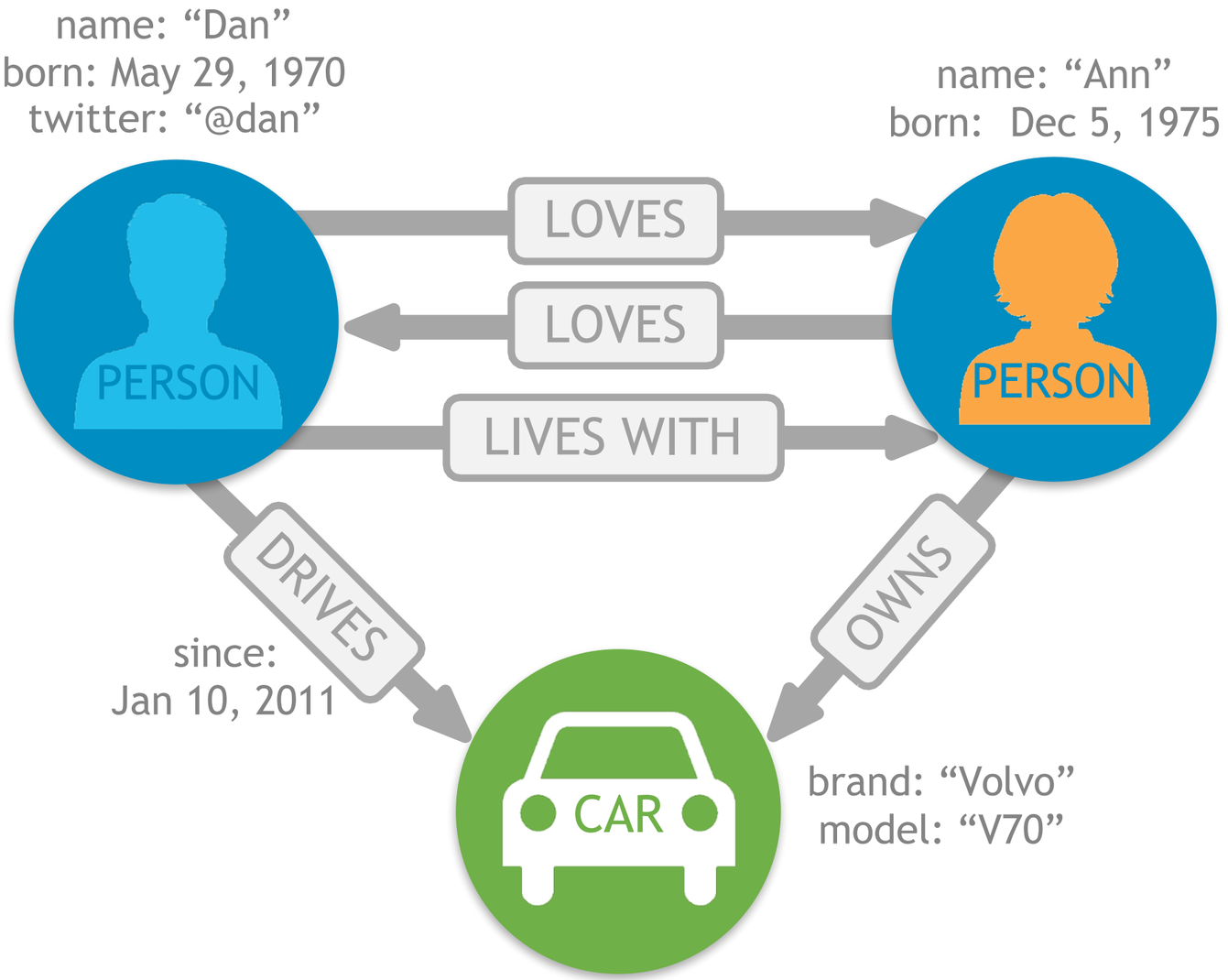
Property Graph Model Components

Nodes

- The entities in the graph
- Can have name-value *properties*
- Can be *labeled*

Relationships

- Relate nodes by type and direction
- Can have name-value *properties*



WHY GRAPHS?

Intuitivness

Speed

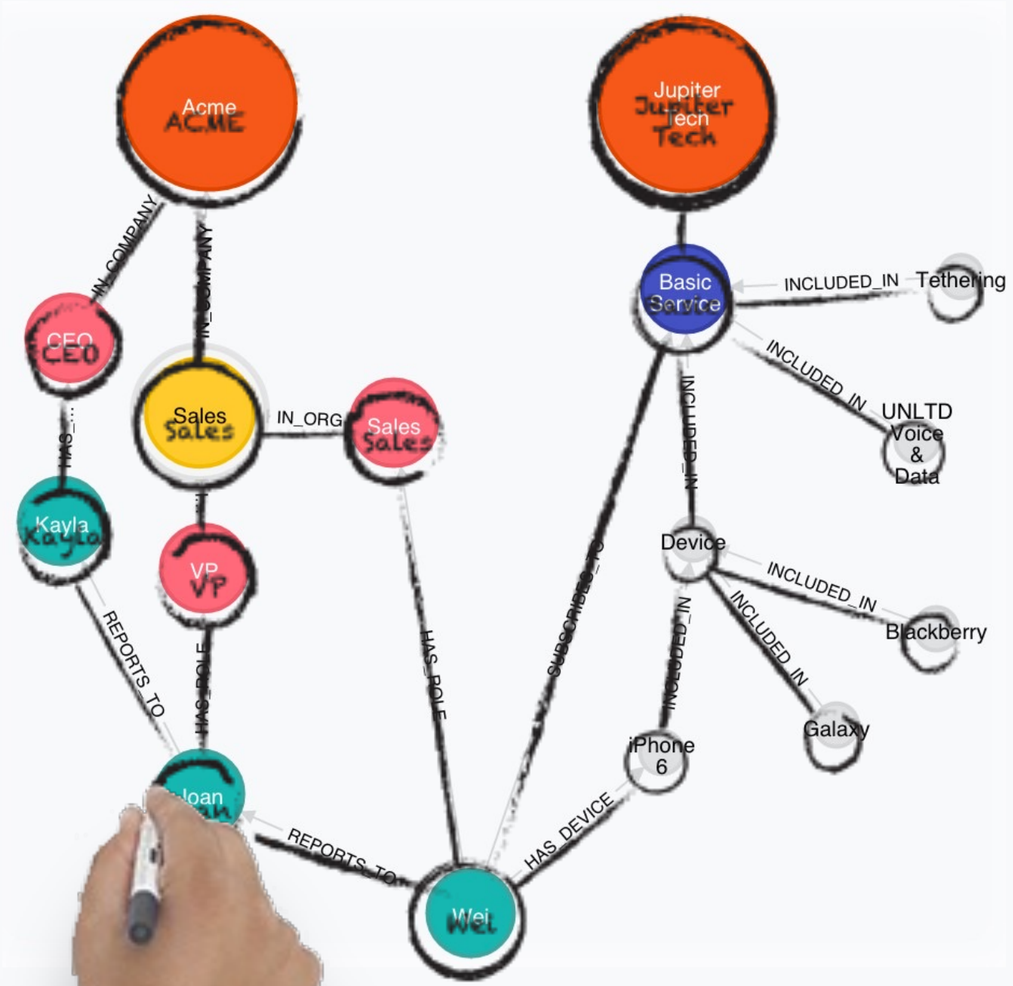
Agility

Intuitiveness

Speed

Agility

Intuitiveness



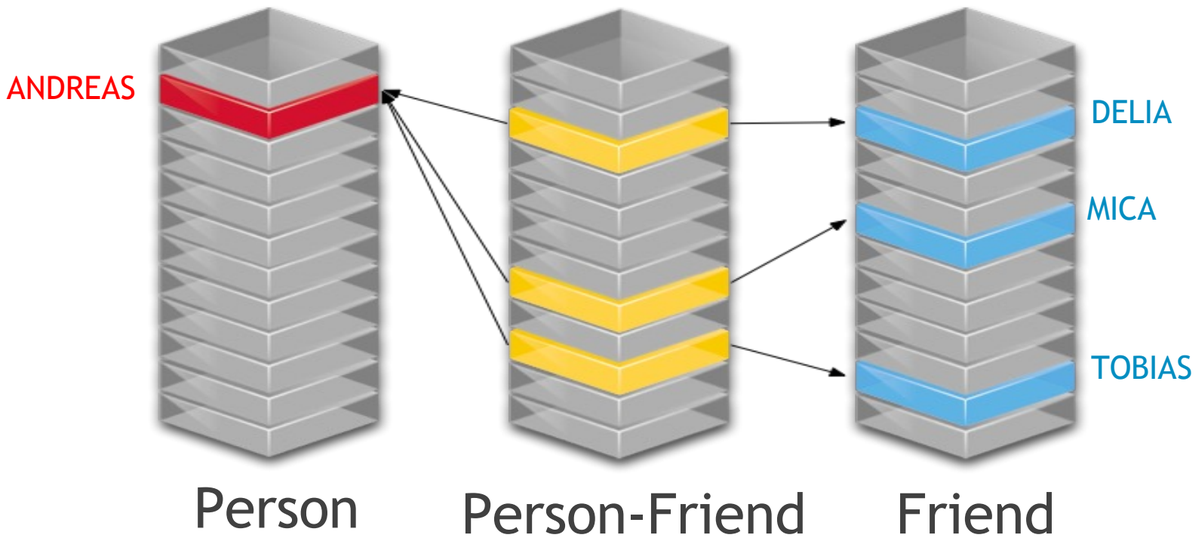
Intuitivness

Speed

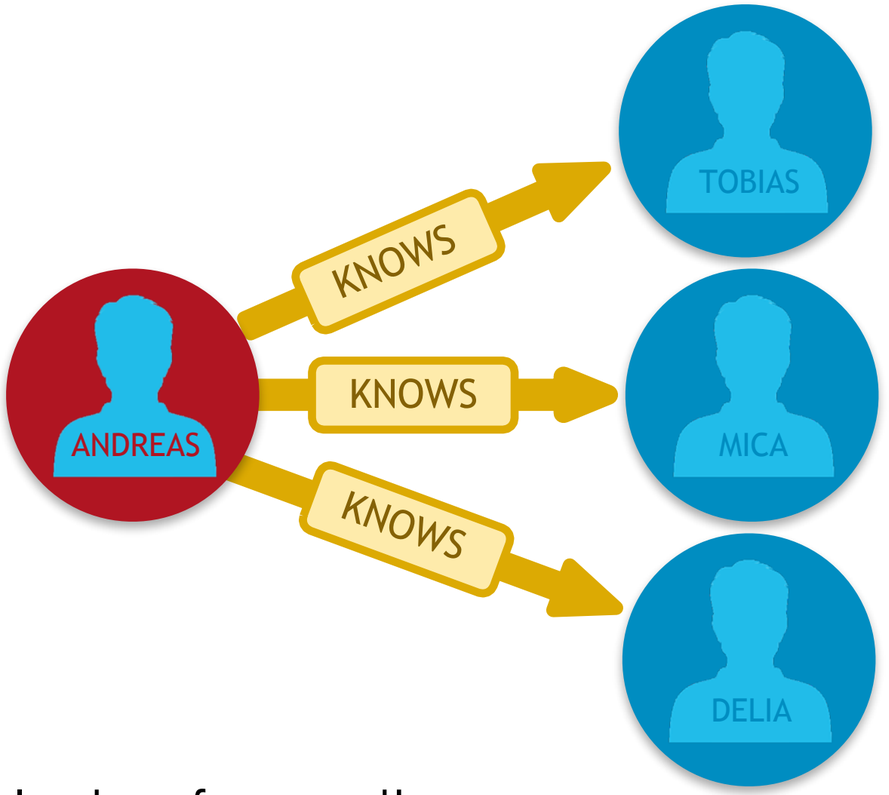
Agility

Relational Versus Graph Models

Relational Model



Graph Model



Index free adjacency

Speed

ebay™

"We found Neo4j to be literally **thousands of times faster** than our prior MySQL solution, with queries that require 10-100 times less code. Today, Neo4j provides eBay with functionality that was previously impossible."

- Volker Pacher, Senior Developer

"Minutes to milliseconds" performance

Queries up to 1000x faster than RDBMS or other NoSQL

Intuitivness

Speed

Agility

Agility =

A Naturally Adaptive Model

+

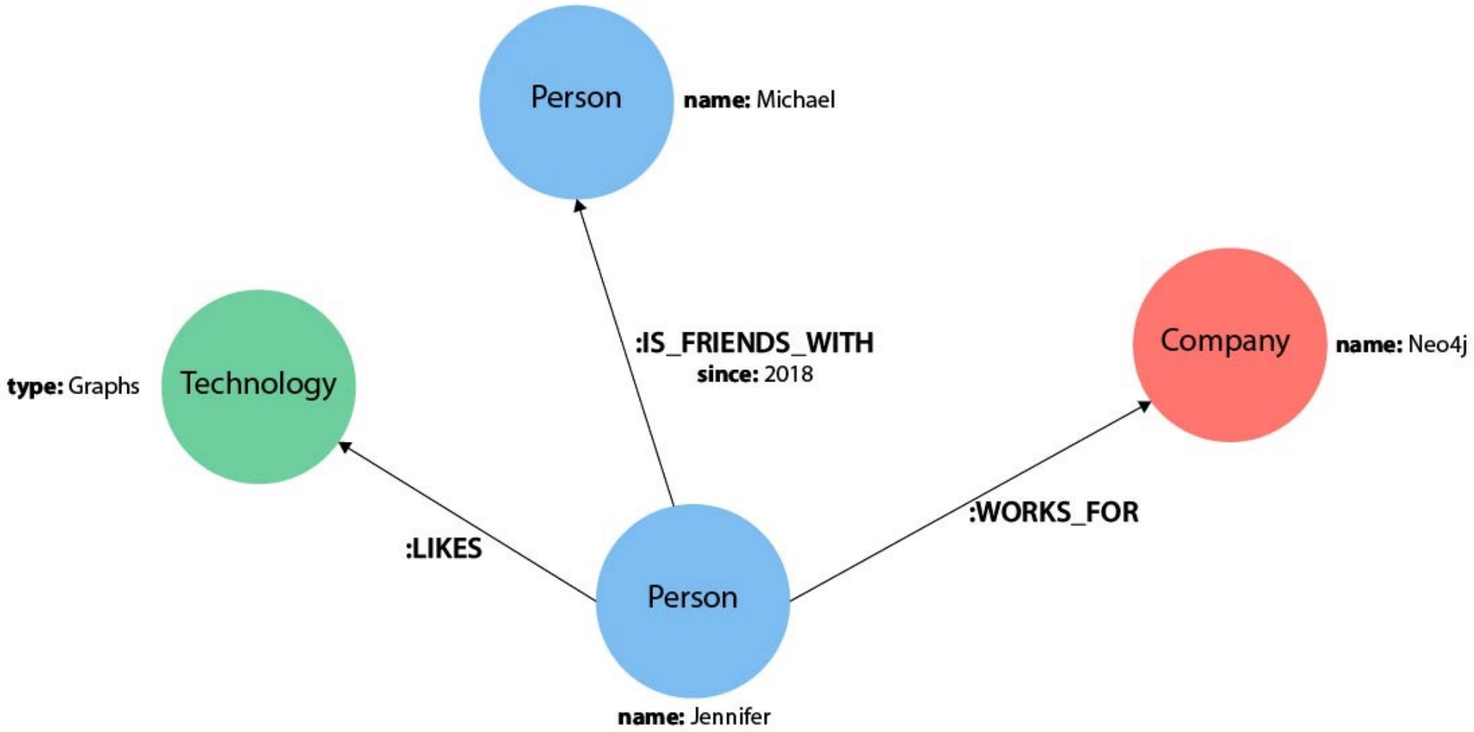
A Query Language Designed for Connectedness

CYPHER

SQL for graphs

(Very) Brief Cypher Tutorial

Property Graph



Cypher specification (Nodes)

Nodes

```
(: Person)
(: Person {name: ‘Jennifer’,
           age: 30})
(: Person {name: ‘Michael’,
           hobby: [‘soccer’, ‘cycling’]})
(: Technology {type: ‘Graphs’})
(: Company {name: ‘Neo4j’,
             headquarter: ‘Bloomington’})
```

Cypher specification (Relationships)

Jennifer likes Graphs

```
(:Person {name: 'Jennifer') -[:LIKES]-> (:Technology {type: 'Graphs'})
```

Jennifer likes Graphs since 1990

```
(:Person {name: 'Jennifer') -[:LIKES {since: 1990}]-> (:Technology {type: 'Graphs'})
```

Cypher Query Language (MATCH and RETURN)

- MATCH searches for an existing node, relationship, label, property, or pattern in the database.
  - find all node labels in the database,
  - search for a particular node,
  - find all the nodes with a particular relationship,
  - look for patterns of nodes and relationships etc
- RETURN specifies what values or results you might want to return.
  - You can return nodes, relationships, node and relationship properties, or patterns in your query results.

Cypher queries (MATCH and RETURN)

Find the labeled Person nodes in the graph:

```
MATCH (p: Person)
RETURN p
```

Find the Person node with name Jennifer:

```
MATCH (p: Person {name: 'Jennifer'})
RETURN p
```

Find which companies Jennifer works for:

```
MATCH (:Person {name: 'Jennifer'}) –[:WORKS_FOR]-> (company: Company)
RETURN company
```

Cypher queries (MATCH and RETURN)

Find the name of the company Jennifer works for:

```
MATCH (:Person {name: 'Jennifer'}) –[:WORKS_FOR]-> (company: Company)
RETURN company.name
```

Since when did Jennifer work for Neo4J

```
MATCH (:Person {name: 'Jennifer'}) –[w:WORKS_FOR]-> (: Company {name:'Neo4j'})
RETURN w.since
```

Cypher queries (MATCH and RETURN)

```
Find the types of relationships that exist from Jennifer to Michael
```

```
MATCH (:Person {name: 'Jennifer'}) –[r]-> (:Person {name: 'Michael'})
RETURN type(r)
```

```
Find the types of relationships that exist between Jennifer and
Michael
```

```
MATCH (:Person {name: 'Jennifer'}) -[r]- (:Person {name: 'Michael'})
RETURN type(r)
```

Cypher queries (WHERE)

```
Find the name of each person who worked for a company since 1990
```

```
MATCH (p: Person)  – [w: WORKS_FOR] ->  (: Company)
WHERE w.since >= 1990
RETURN p.name
```

```
Find the name of each person who worked for a company since 1990
and who has Jennifer as a friend
```

```
MATCH (p: Person) -[w: WORKS_FOR]->(: Company),
          (p) -[ :IS_FRIENDS_WITH]->(:Person {name: `Jennifer'})
WHERE w.since >= 1990
RETURN p.pname
```

Cypher queries (more complex patterns)

```
Find the name of each person who works for a company and who has a
friend:

                MATCH (p: Person) –[:WORKS_FOR]-> (:Company),
                          (p: Person) –[:IS_FRIENDS_WITH] -> (:Person)
                RETURN p.name

Alternatively,

  MATCH (:Company) <–[:WORKS_FOR]- (p: PERSON) -[:IS_FRIENDS_WITH]->
  (:Person)
  RETURN p.pname
```

Cypher queries (grouping and aggregation)

Find the id and name of each person along with of his or her friends

```
MATCH (p: Person) –[:IS_FRIENDS_WITH]-> (f: Person)
RETURN id(p), p.name, count(f)
```