NORTHWESTERN
UNIVERSITY

MSiA 430: Software Tools for BDA

Spring 2023

GraphDB – Part II

Instructor: Goce Trajcevski

TA: Elliot Gardner

NORTHWESTERN
UNIVERSITY

- Intro/Motivation
- Basics of Graph Theory
  - Representations
  - Algorithms
- **Relational vs. GraphDB**
- **Basics of GraphDB**
  - **…via actual Cypher…**
- **GraphDB/Neo4j cont.**
  - **Context and more Cypher;**
  - **Notes from practical use-cases;**
    …when to use which…

Another reading: Ian Robinson (et al.): Graph Databases (2nd edition) Online Access provided by Northwestern Library…

- Aggregate vs. Connected Data Models

Aggregate Oriented Model

NORTHWESTERN UNIVERSITY

"There is a significant downside - the whole approach works really well when data access is aligned with the aggregates, but what if you want to ***look at the data in a different way****?*
Order entry naturally stores orders as aggregates, but analyzing product sales cuts across the aggregate structure.
The advantage of not using an aggregate structure is that it allows you to slice and dice your data different ways for different audiences."

Martin Fowler

Connected Data Model

*The **connected data model is based on fine grained elements that are richly connected**, the emphasis is on **extracting many dimensions and attributes as elements**.*
Connections are cheap and can be used not only for the domain-level relationships but also for additional structures that allow efficient access for different use-cases.

The fine grained model requires a external scope for mutating operations (e.g., Transactions).

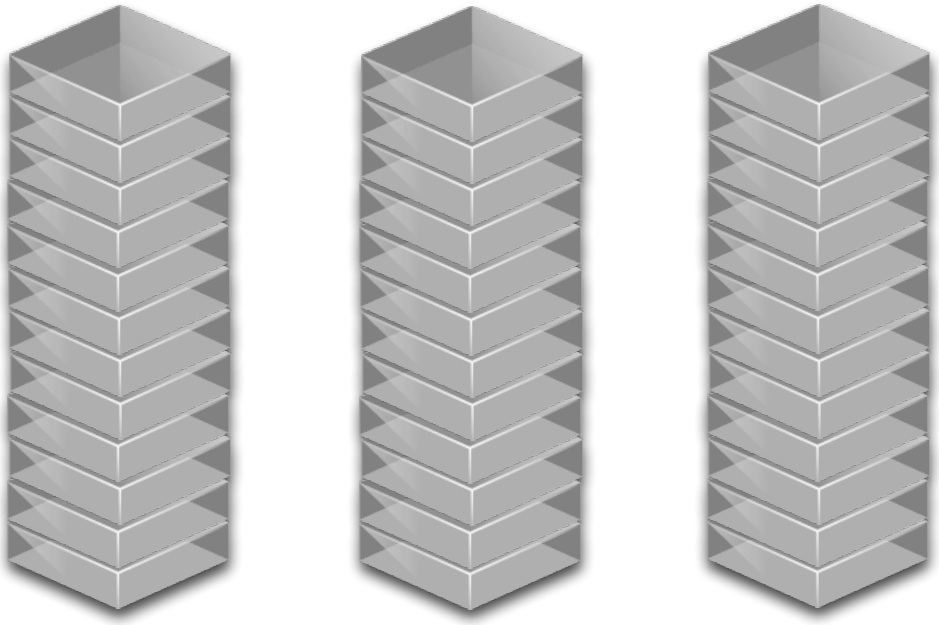
Michael Hunger

NORTHWESTERN
UNIVERSITY

Relational vs. Graph

Relational vs. Graph

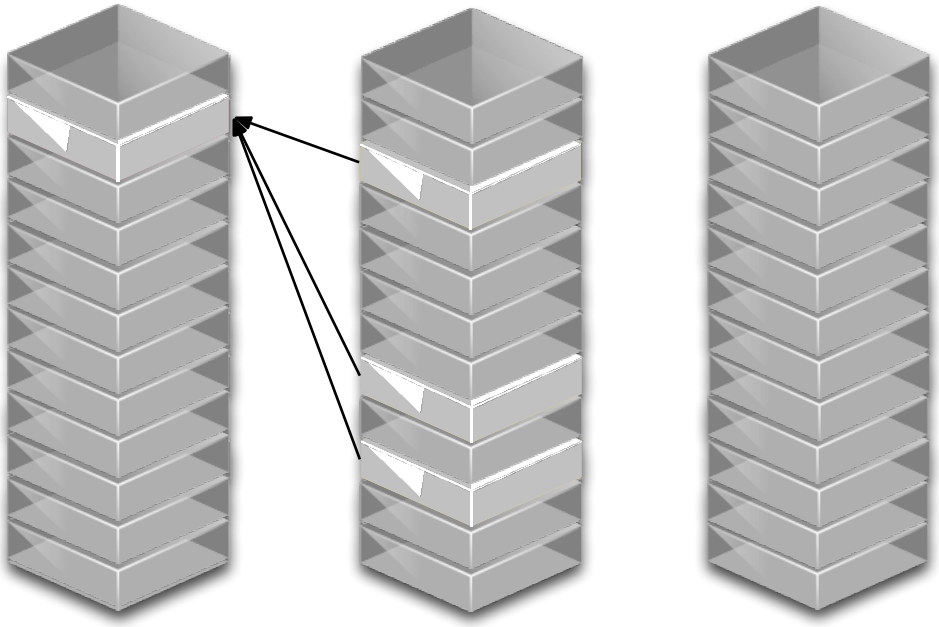
You know relational



`foo` `foo_bar` `bar`

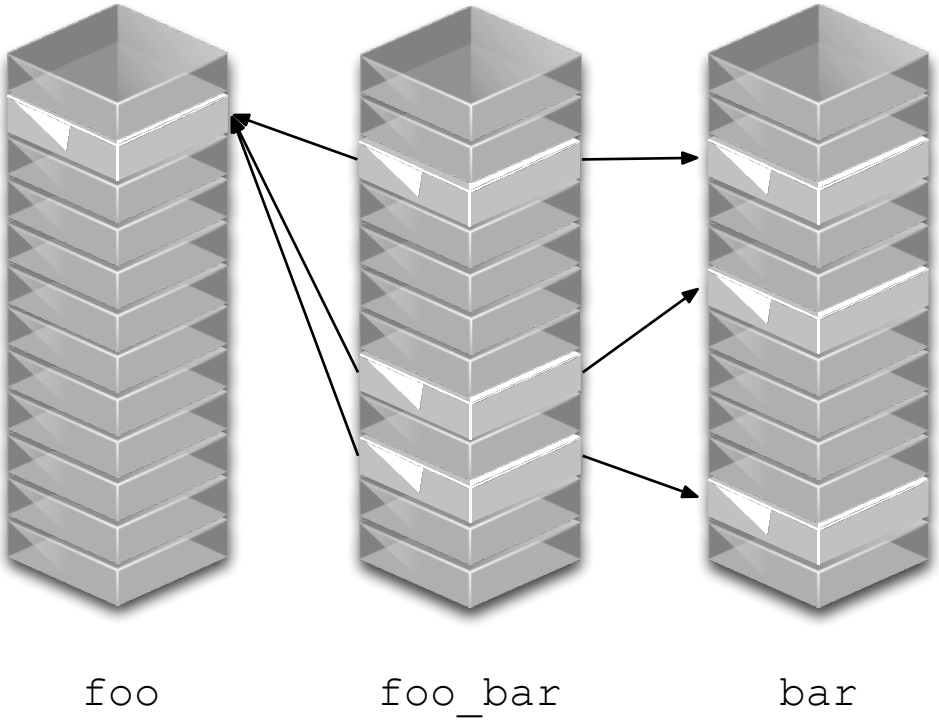
Relational vs. Graph

You know relational



Relational vs. Graph

You know relational

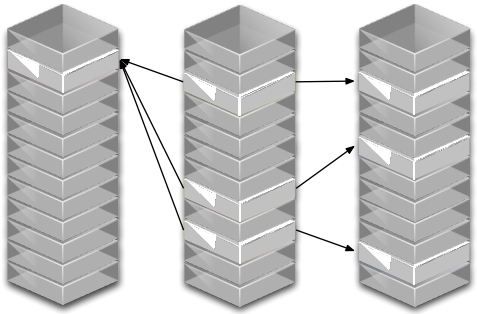


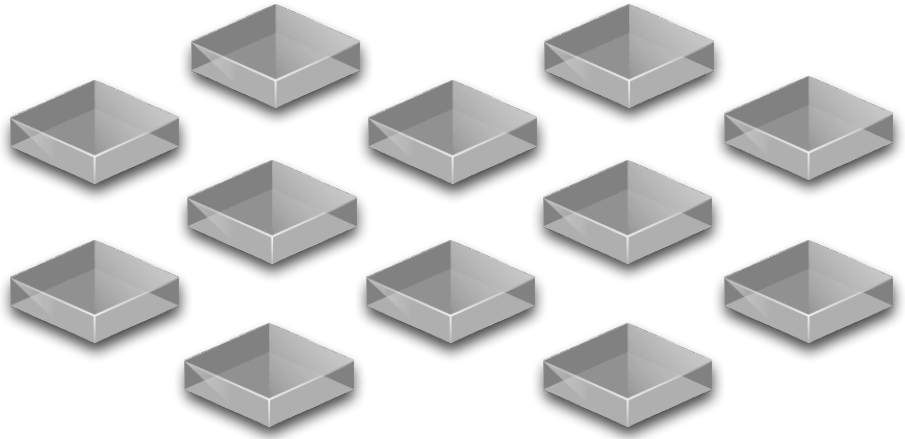
Relational vs. Graph

NORTHWESTERN UNIVERSITY

You know relational

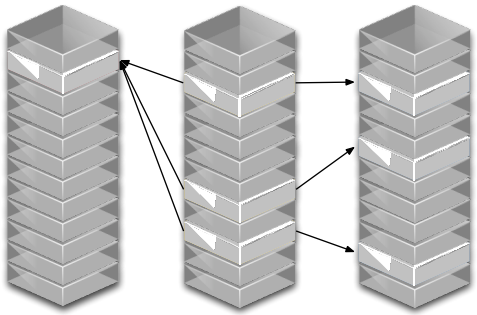
...now consider a graph

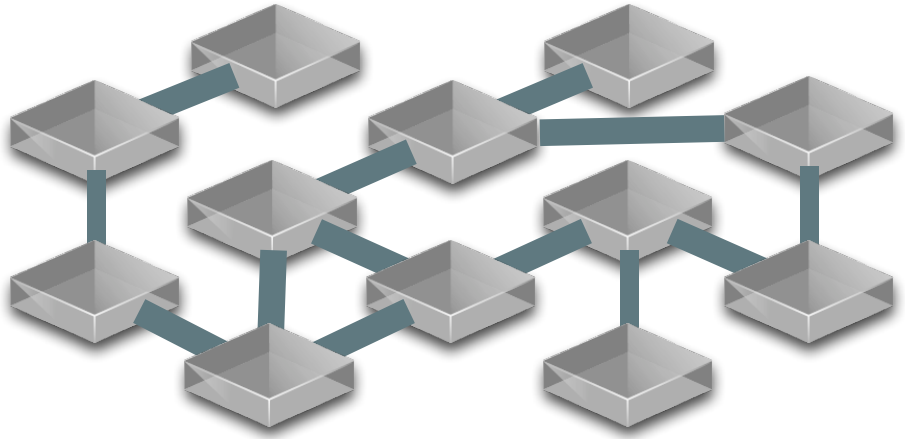




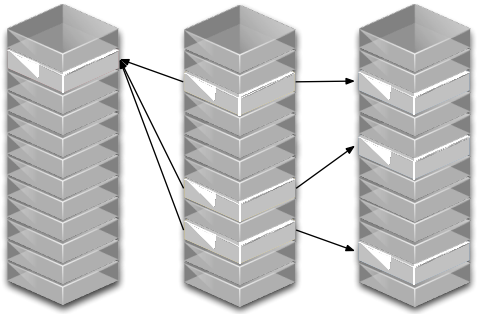
You know relational

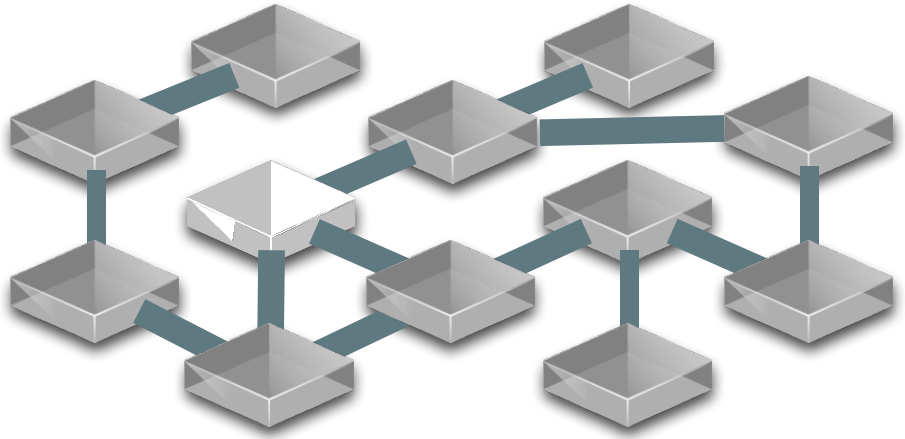
...now consider a graph



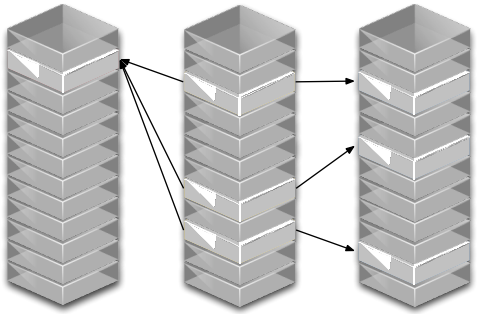


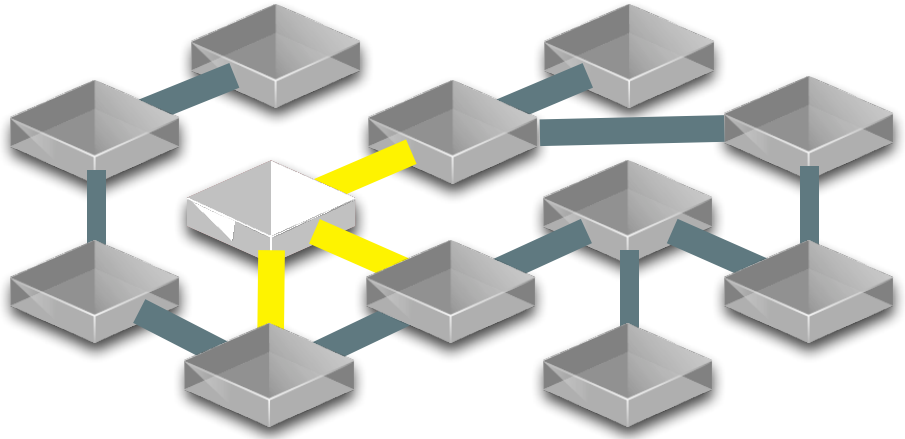
Relational vs. Graph





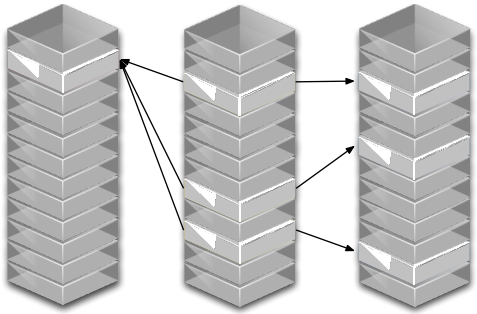
Relational vs. Graph

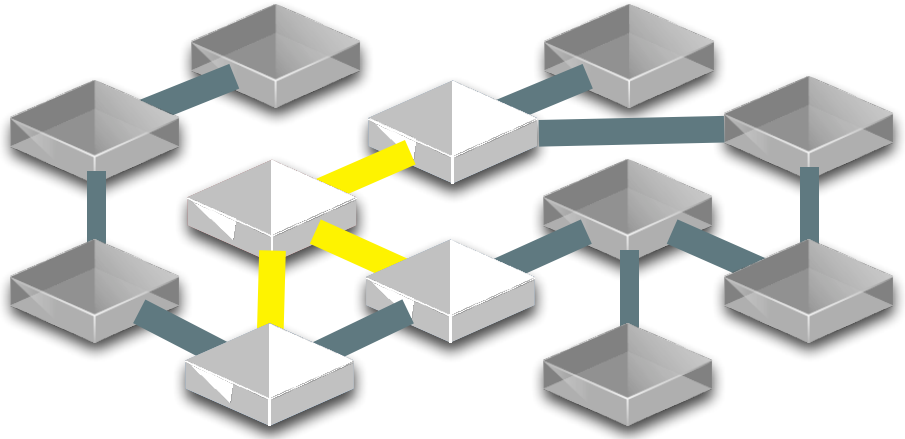




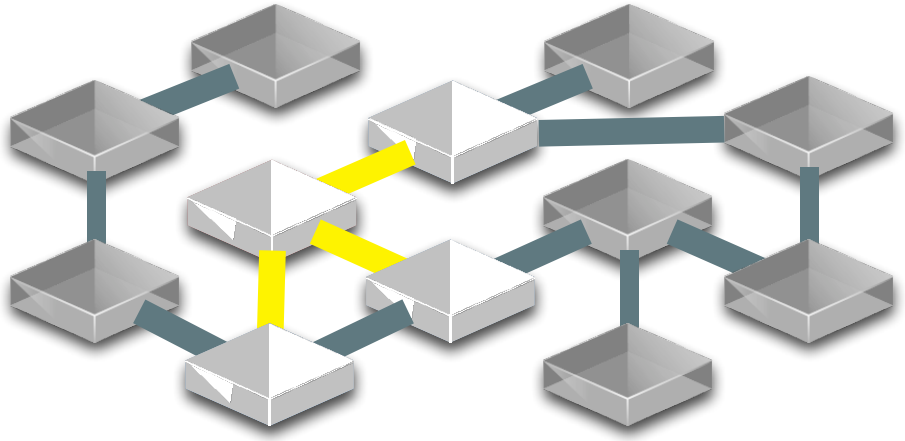
You know relational

...now consider a graph





Relational vs. Graph



Think of Relational Tables and a DB Query

NORTHWESTERN UNIVERSITY

EmpID	Name	PictureRef
4951870	John Doe	s3://acme-pics/4951870.png
9765207	Jane Smith	s3://acme-pics/9765207.png
4150915	Shyam Bhatt	s3://acme-pics/
7566243	Kathryn Bate	

EmpID	Manager ID	StartDate	EndDate
4951870	9765207	20170101	null
9765207	7566243	20150130	null
4150915	8795882	20141215	20150312
7566243	8509238	20120605	20140124

EmpID	Building	Office
4951870	1200	124A
9765207	1300	187D
4150915	45	432C

Retrieve (the pictures of) all the employees who have offices in the same buildings as their managers

Think of a Relational DB Query

NORTHWESTERN UNIVERSITY

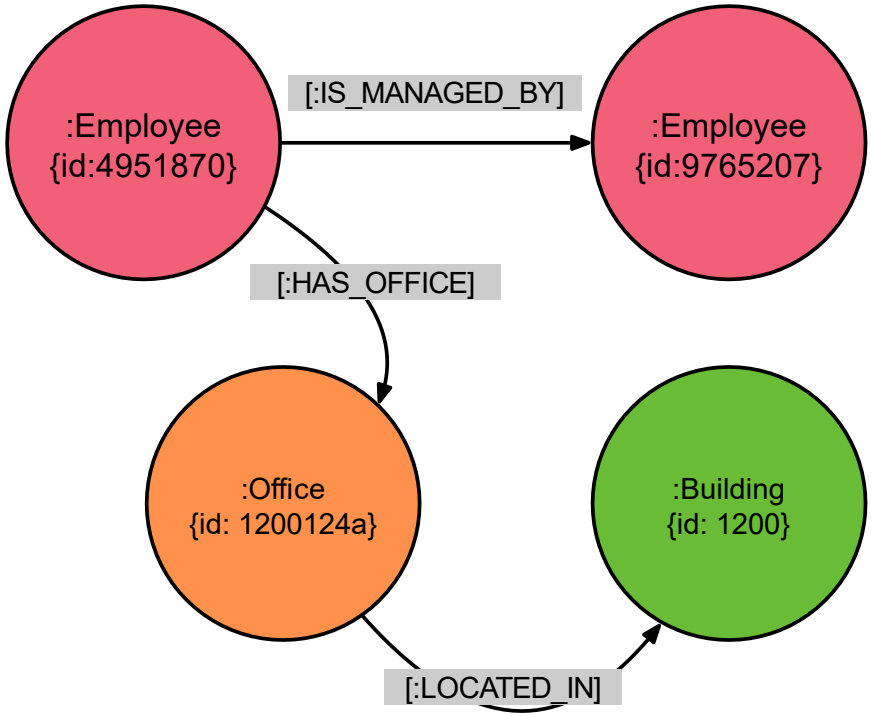
Retrieve (the pictures of) all the employees who have offices in the same buildings as their managers

EmpID	Name	PictureRef	EmpID	Manager ID	StartDate	EndDate
4951870	John Doe	s3://acme-pics/4951870.png	51870	9765207	20170101	null
9765207	Jane Smith	s3://acme-pics/9765207.png	65207	7566243	20150130	null
Shyam Bhatt				8795882	20141215	20150312
Kathryn Bates				8509238	20120605	20140124

EmpID	Building	Office
4951870	1200	124A
9765207	1300	187D
4150915	45	432C

Too many Index Lookups; Very Expensive!

(Partial) Graph View

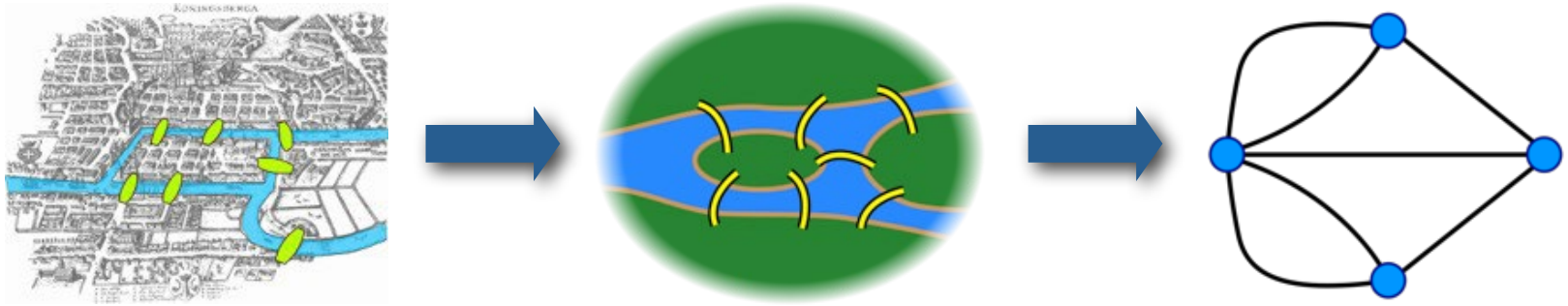


1 Index Lookup
(find :Employee nodes)

Then Index-Free Adjacency

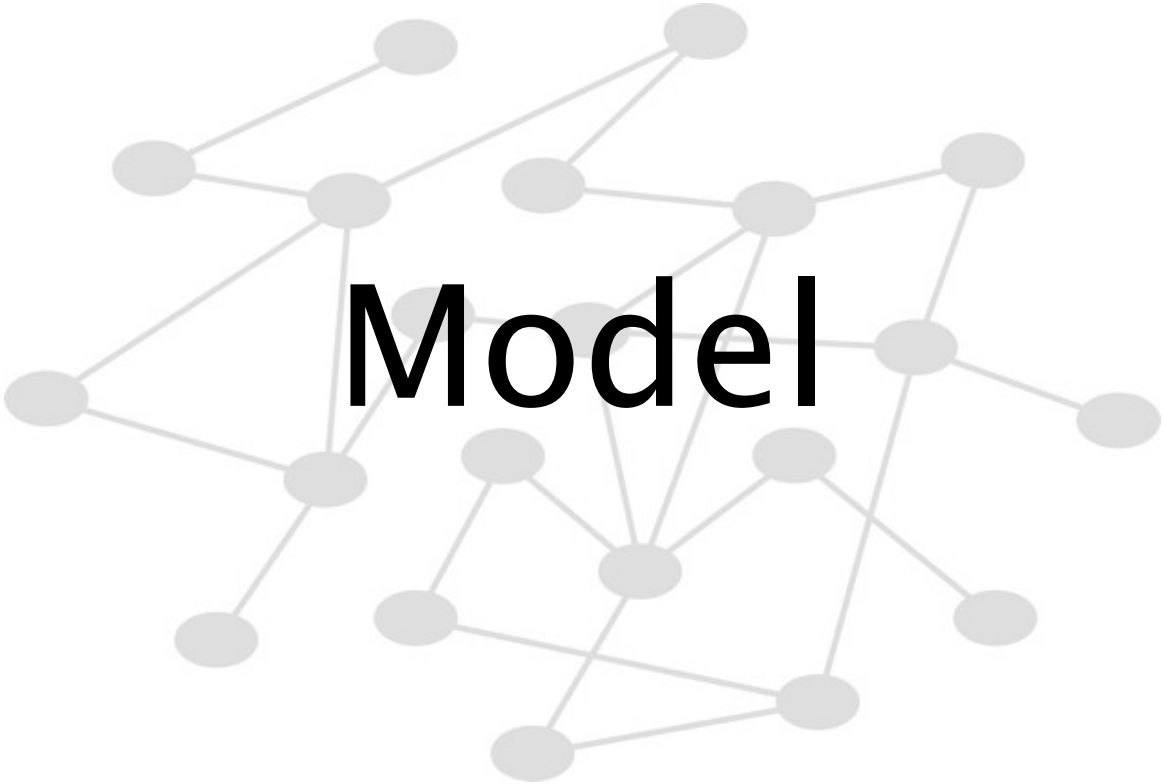
Models

Purposeful abstraction of a domain designed to satisfy particular application/ end-user goals

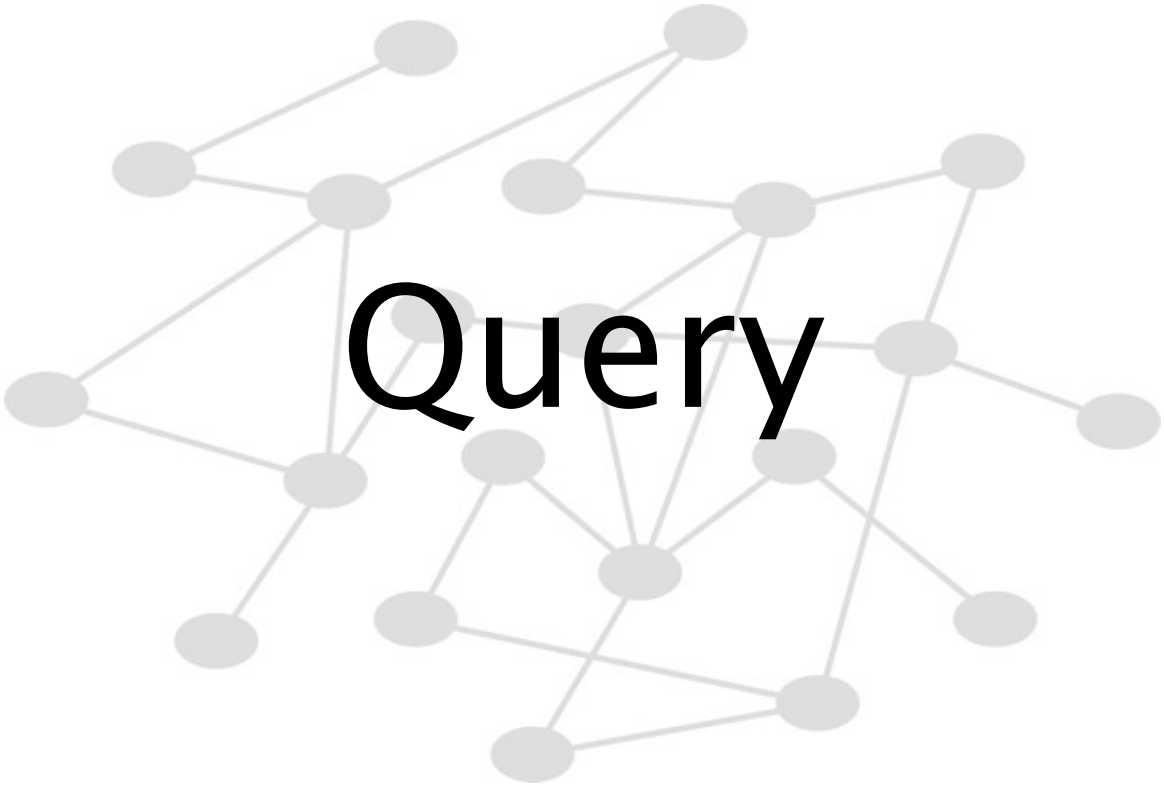


Images:
en.wikipedia.org

Design for Queryability



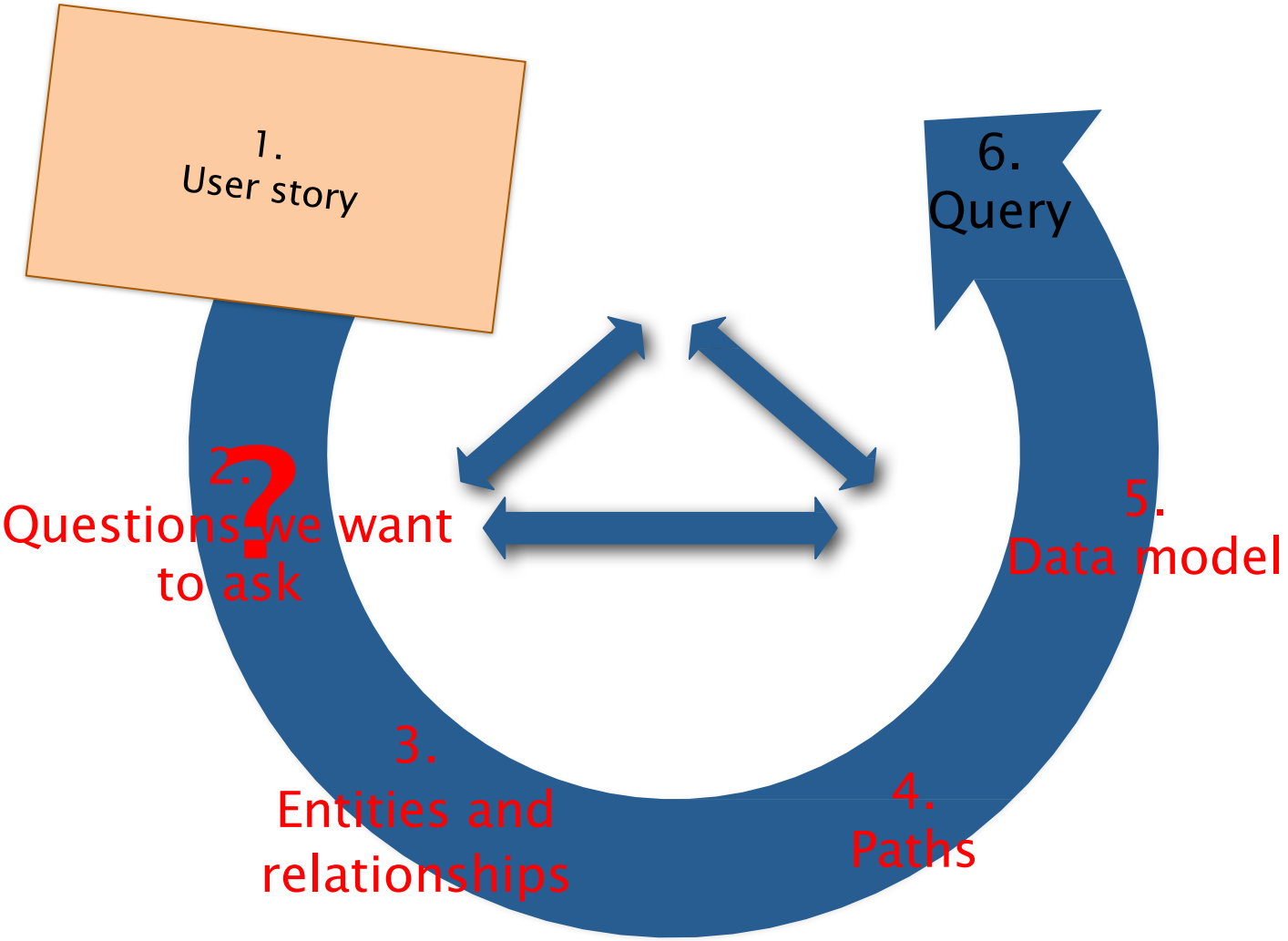
Design for Queryability



(Recall…): Methodology

1. Identify application/end-user goals
2. Figure out what questions to ask of the domain
3. Identify entities in each question
4. Identify relationships between entities in each question
5. Convert entities and relationships to paths These become the basis of the data model
6. Express questions as graph patterns These become the basis for queries

From User Story to Model and Query



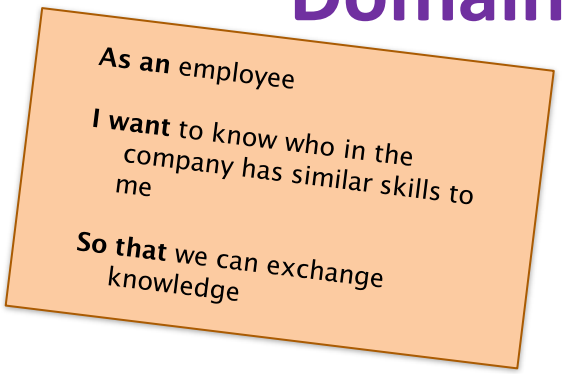
1. Application/End-User Goals

As an employee

I want to know who in the company has similar skills to me

So that we can exchange knowledge

2. Questions To Ask of the Domain



Which people, who work for the same company as me, have similar skills to me?

3. Identify Entities

NORTHWESTERN UNIVERSITY

Which people, who work for the same company as me, have similar skills to me?

Person

Company

Skill

4. Identify Relationships Between Entities

Which people, who work for the same company as me, have similar skills to me?

Person WORKS_FOR Company

Person HAS_SKILL Skill

NORTHWESTERN
UNIVERSITY

5. Convert to Cypher Paths

Person WORKS_FOR Company

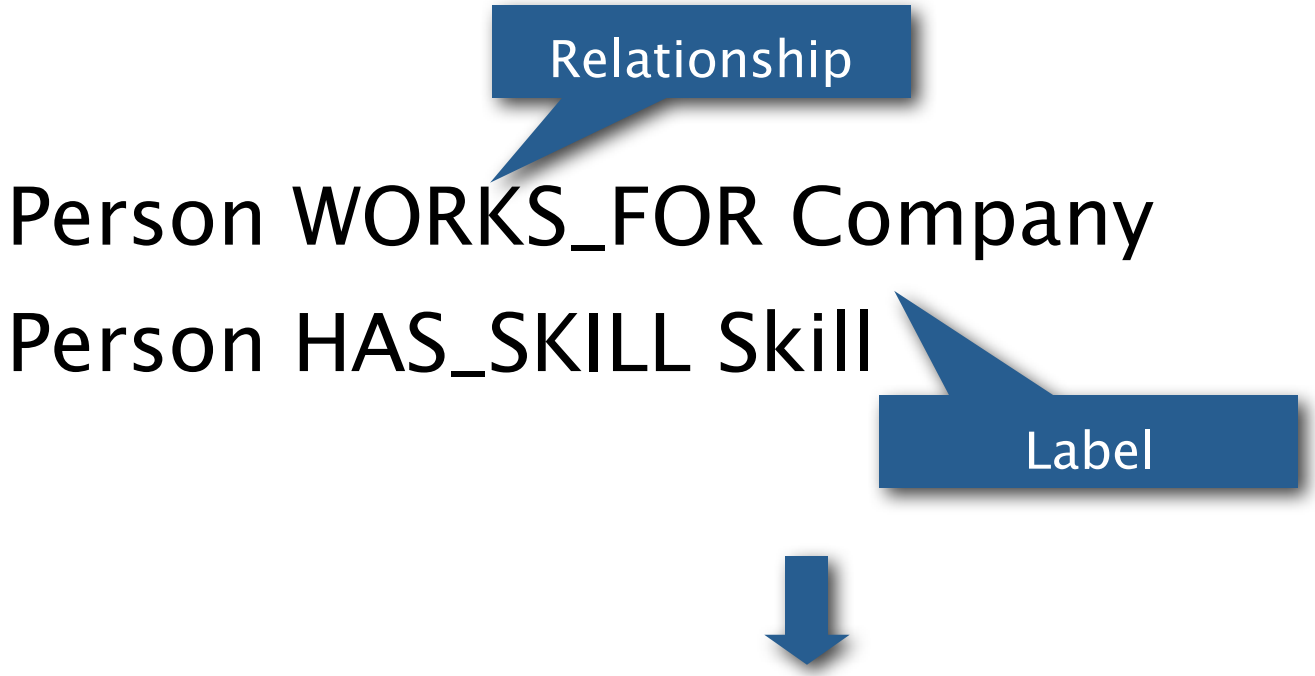
Person HAS_SKILL Skill

5. Convert to Cypher Paths

Person WORKS_FOR Company

Person HAS_SKILL Skill

5. Convert to Cypher Paths



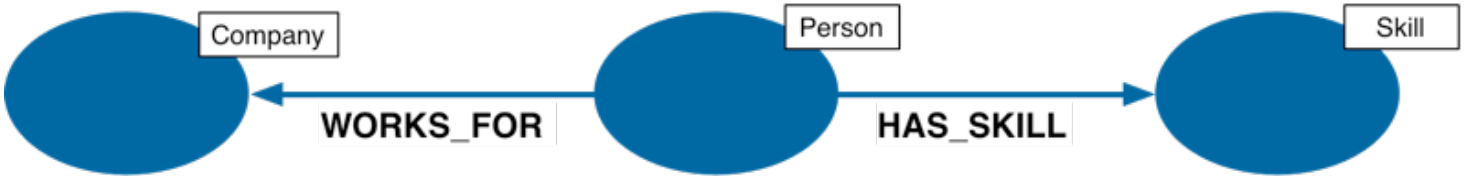

```
(:Person)-[:WORKS_FOR]->(:Company),
(:Person)-[:HAS_SKILL]->(:Skill)
```

Consolidate Paths

NORTHWESTERN UNIVERSITY

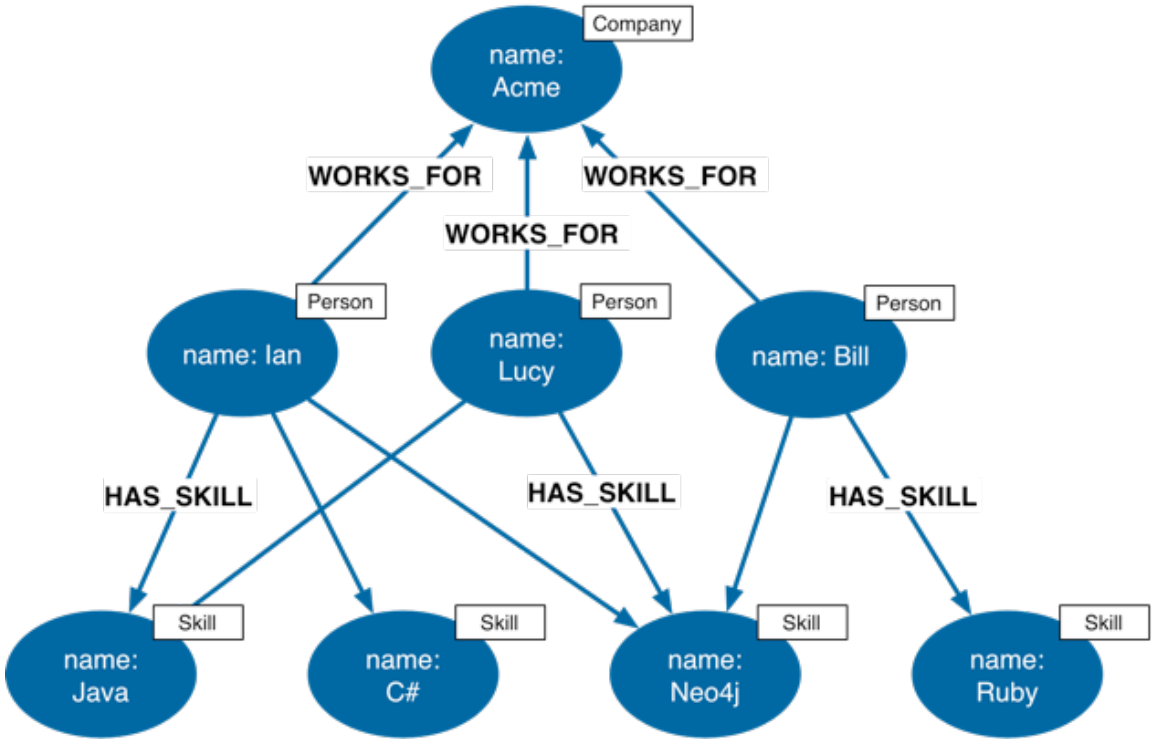
```
(:Person)-[:WORKS_FOR]->(:Company),
(:Person)-[:HAS_SKILL]->(:Skill)
```

```
(:Company)<-[:WORKS_FOR]-(:Person)-[:HAS_SKILL]->(:Skill)
```



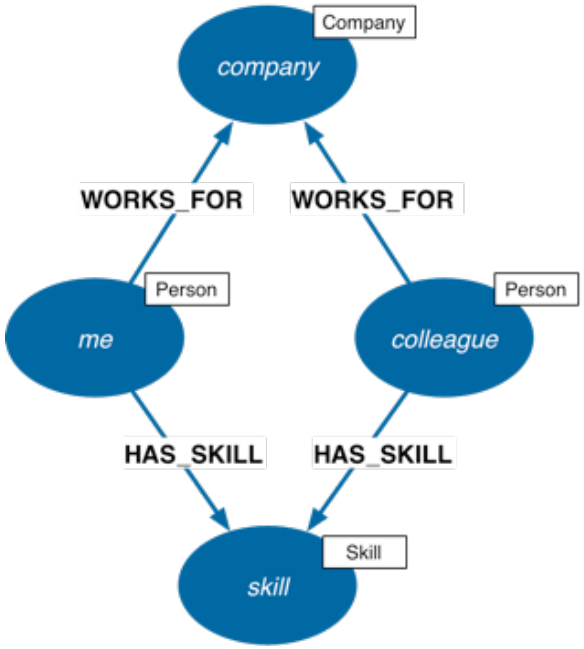
Candidate Data Model

```
(:Company)<-[:WORKS_FOR]-(:Person)-[:HAS_SKILL]->(:Skill)
```



6. Express Question as Graph Pattern

Which people, who work for the same company as me, have similar skills to me?

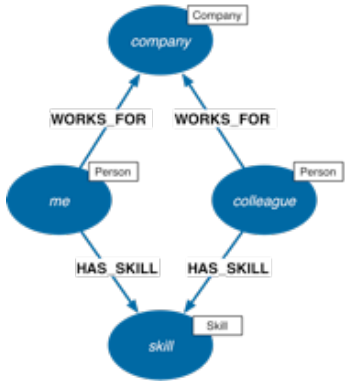


Cypher Query

Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)

WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



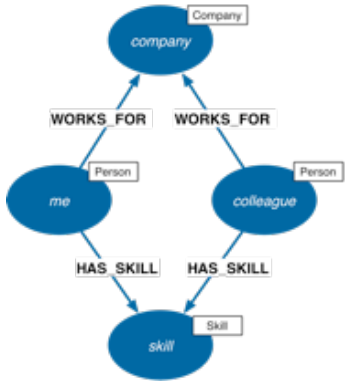
Graph Pattern

NORTHWESTERN UNIVERSITY

Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)

WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

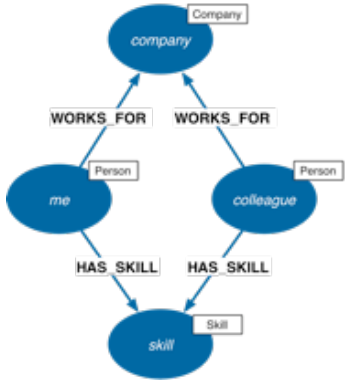


Anchor Pattern in Graph

NORTHWESTERN UNIVERSITY

Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

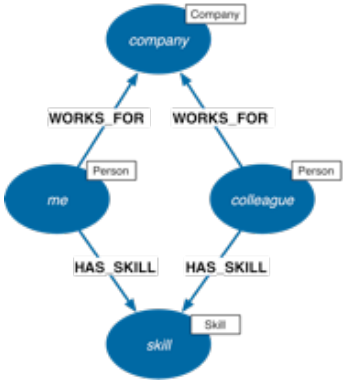


Create Projection of Results

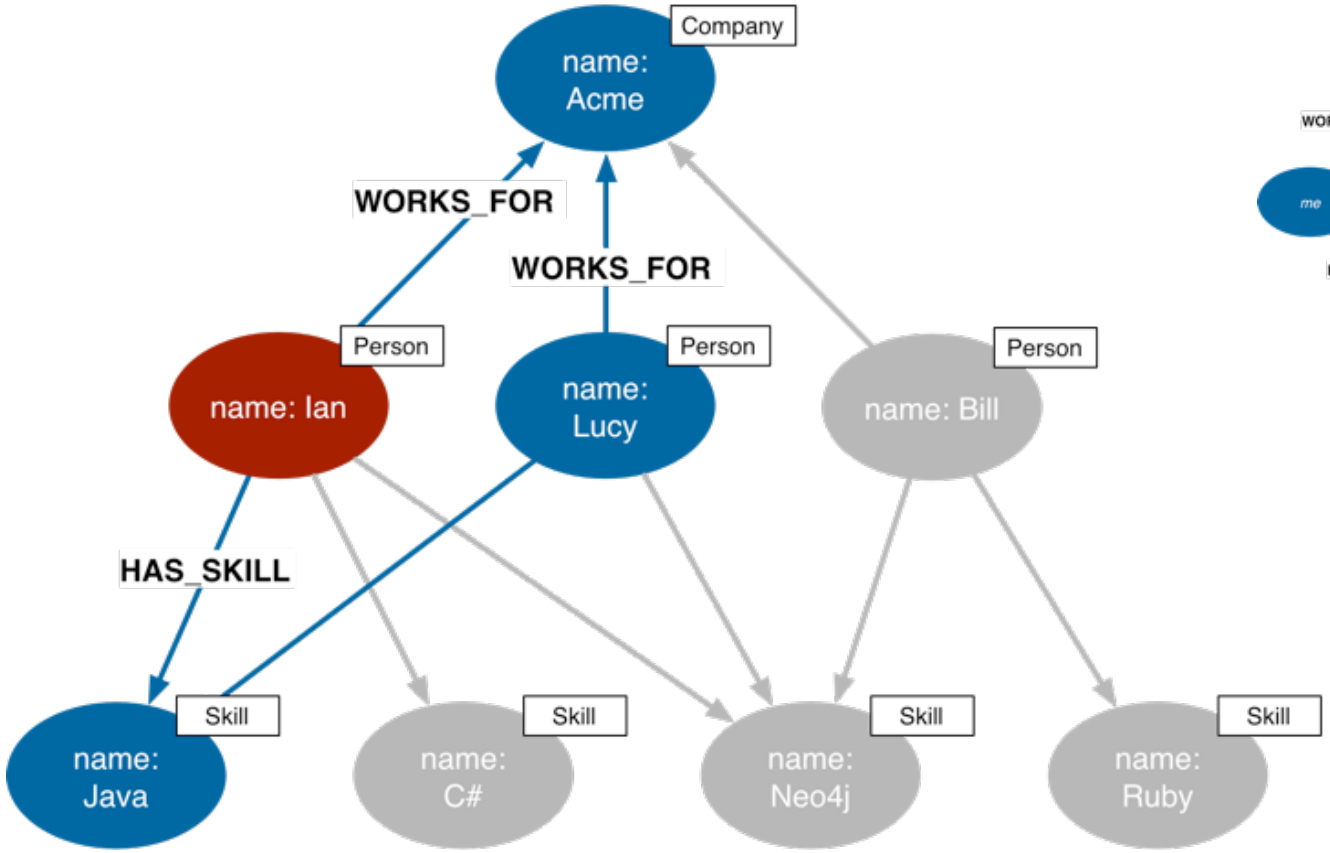
NORTHWESTERN UNIVERSITY

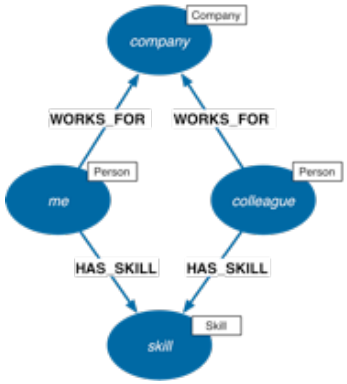
Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

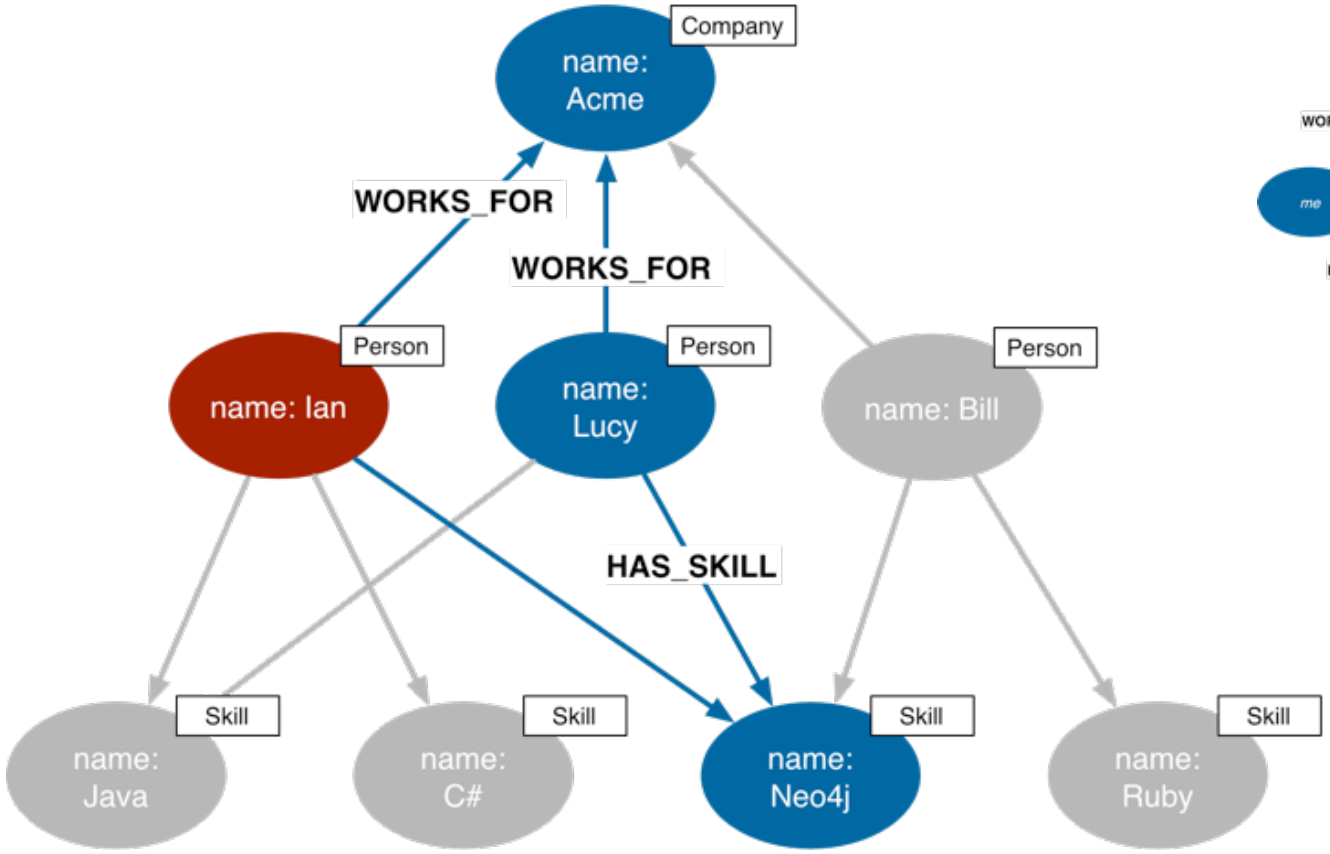


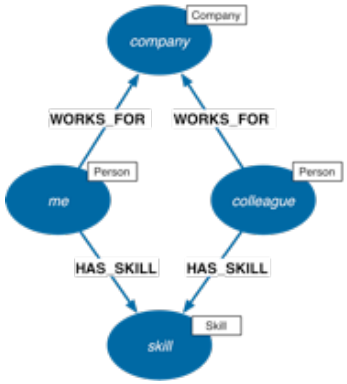
First Match


Company
name: Acme
WORKS_FOR
WORKS_FOR
Person
name: Ian
Person
name: Lucy
Person
name: Bill
HAS_SKILL
Skill
name: Java
Skill
name: C#
Skill
name: Neo4j
Skill
name: Ruby

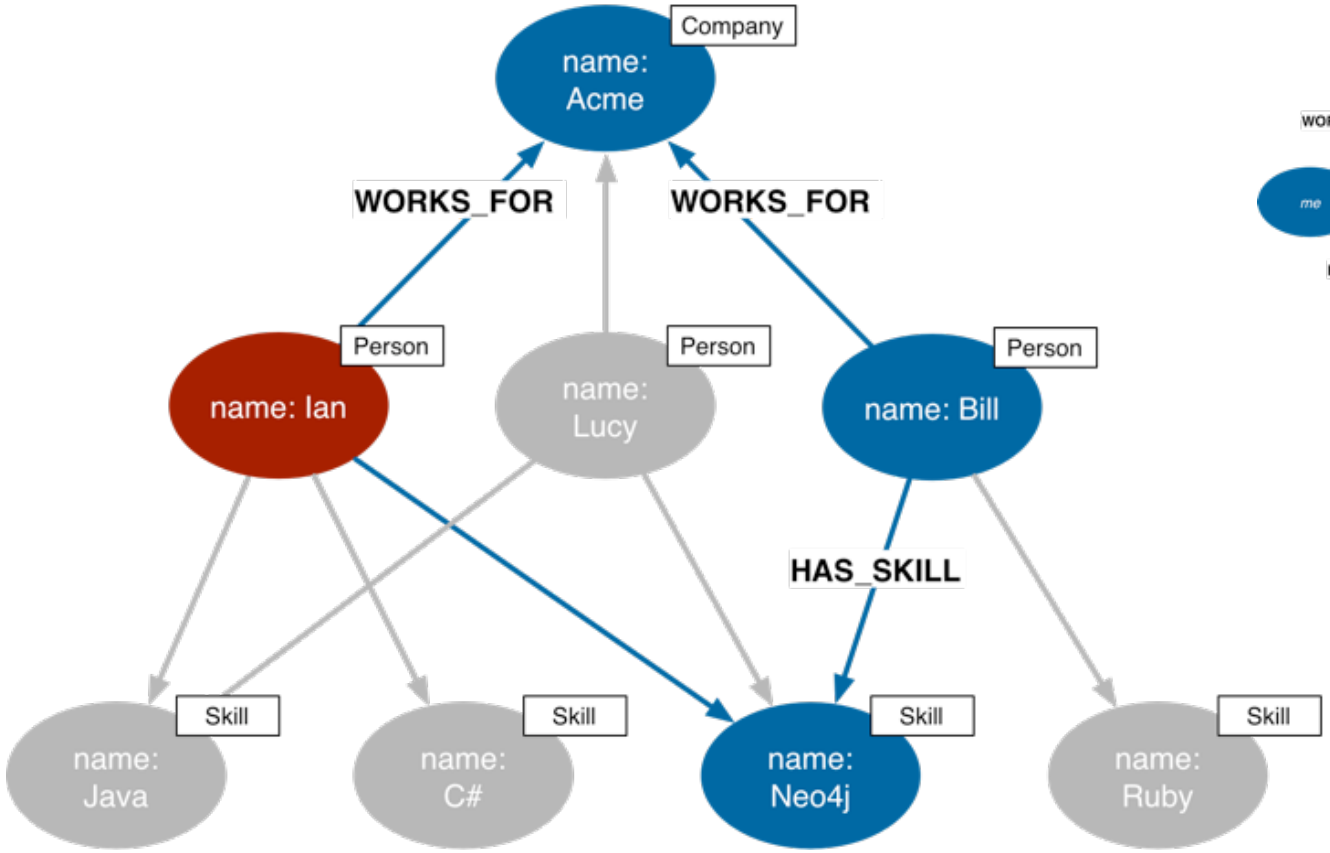

Company
company
WORKS_FOR
WORKS_FOR
Person
me
Person
colleague
HAS_SKILL
HAS_SKILL
Skill
skill

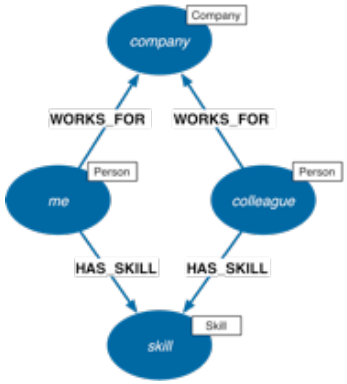
Second Match





Third Match


Company
name: Acme
WORKS_FOR
WORKS_FOR
Person
name: Ian
Person
name: Lucy
Person
name: Bill
HAS_SKILL
Skill
name: Java
Skill
name: C#
Skill
name: Neo4j
Skill
name: Ruby


Company
company
WORKS_FOR
WORKS_FOR
Person
me
Person
colleague
HAS_SKILL
HAS_SKILL
Skill
skill

Running the Query

```
+-----------------------------------+
| name    | score | skills          |
+-----------------------------------+
| "Lucy" | 2       ["Java","Neo4j"] |
| "Bill" | 1       ["Neo4j"]        |
+-----------------------------------+
2 rows
```

From User Story to Model and Query

NORTHWESTERN UNIVERSITY

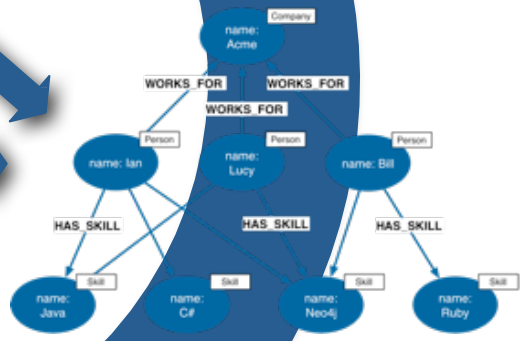
As an employee

I want to know who in the company has similar skills to me

So that we can exchange knowledge

```
MATCH (company)<-[:WORKS_FOR]-(me:Person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

Which people, who work for the same company as me, have similar skills to me?



Person WORKS_FOR Company

Person HAS_SKILL Skill

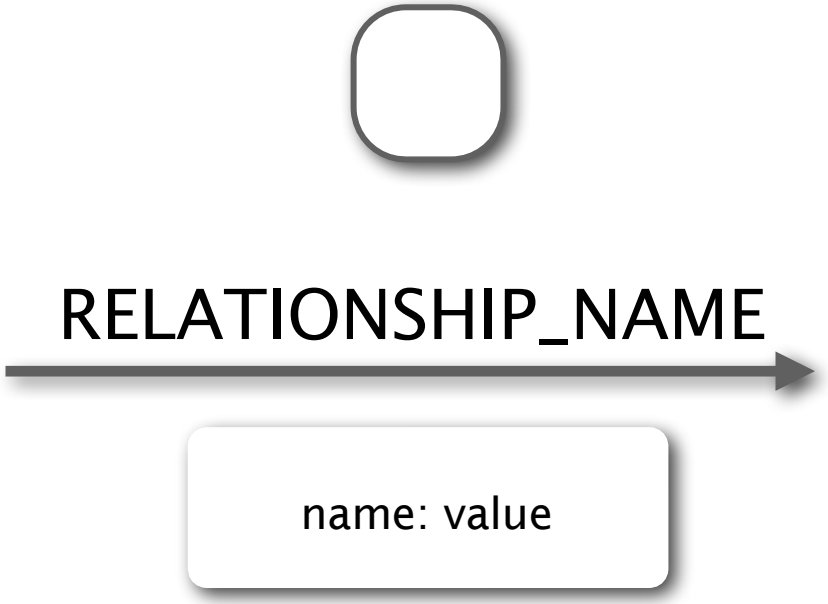
```
(:Company)<-[:WORKS_FOR]-(:Person)-[:HAS_SKILL]->(:Skill)
```

NORTHWESTERN
UNIVERSITY

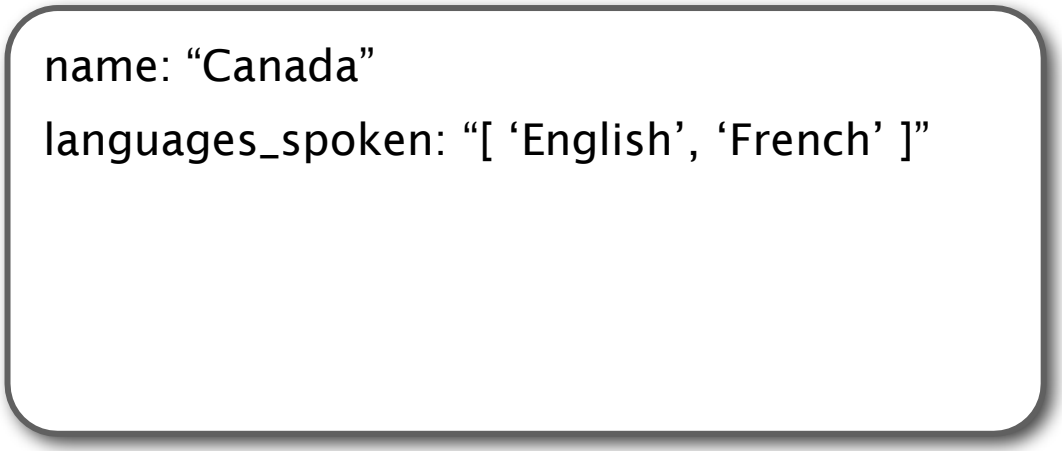
Paradigm

Building blocks

- Nodes
- Relationships
- Properties



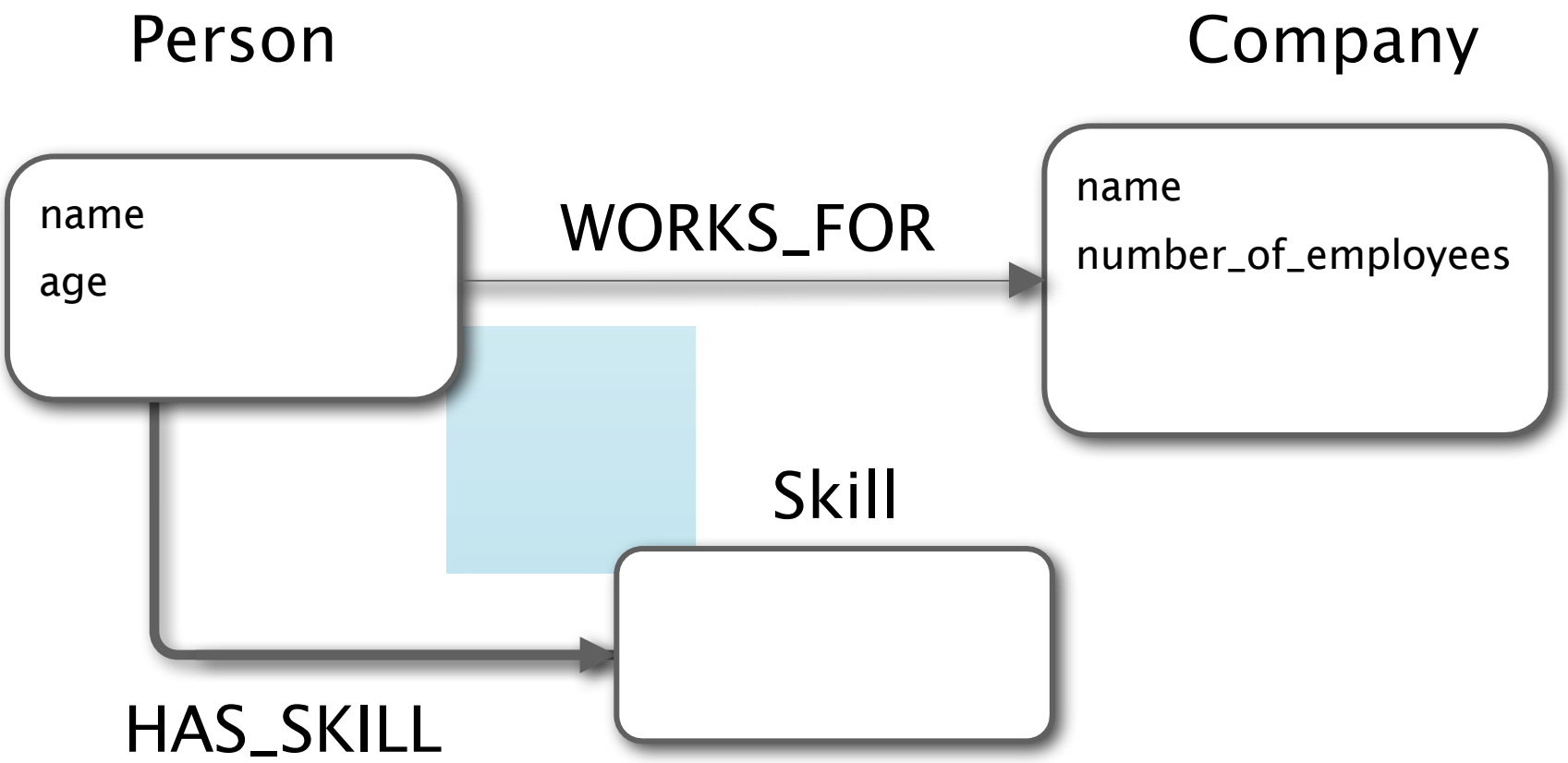
Anti-pattern: rich properties



Normalize Nodes

Anti-Pattern: Node represents multiple concepts

Normalize into separate concepts



Challenge: Property or Relationship?

NORTHWESTERN UNIVERSITY

- Can every property be replaced by a relationship?
- Should every entity with the same property values be connected?

Object Mapping

- Similar to how you would map objects to a relational database, using an ORM such as Hibernate
- Generally simpler and easier to reason about
- Examples
  - Java: Spring Data Neo4j
  - Ruby: Active Model
- Why Map?
  - Do you use mapping because you are scared of SQL?
  - Following DDD, could you write your repositories directly against the graph API?

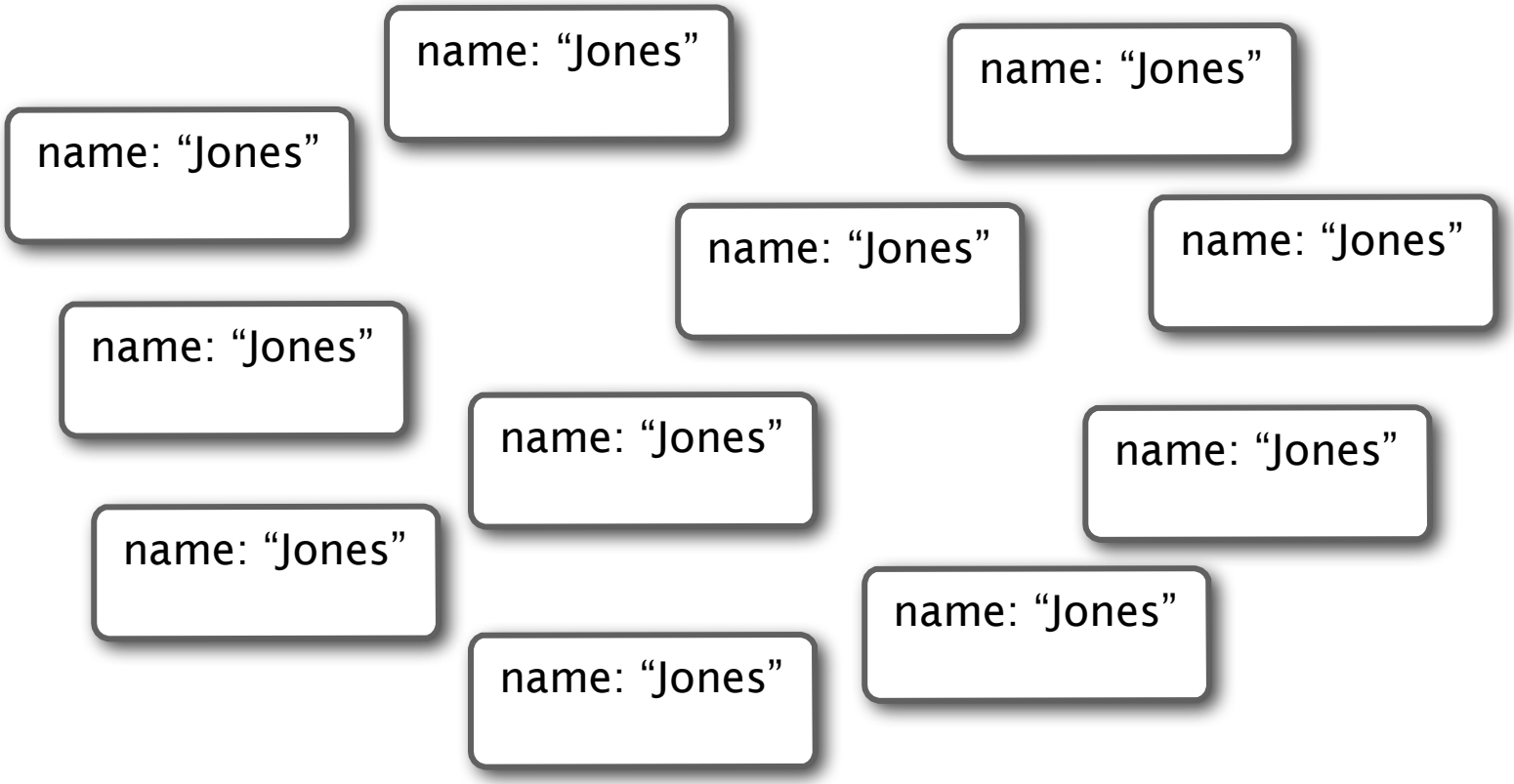
NORTHWESTERN
UNIVERSITY

CONNECT for fast access In-Graph Indices

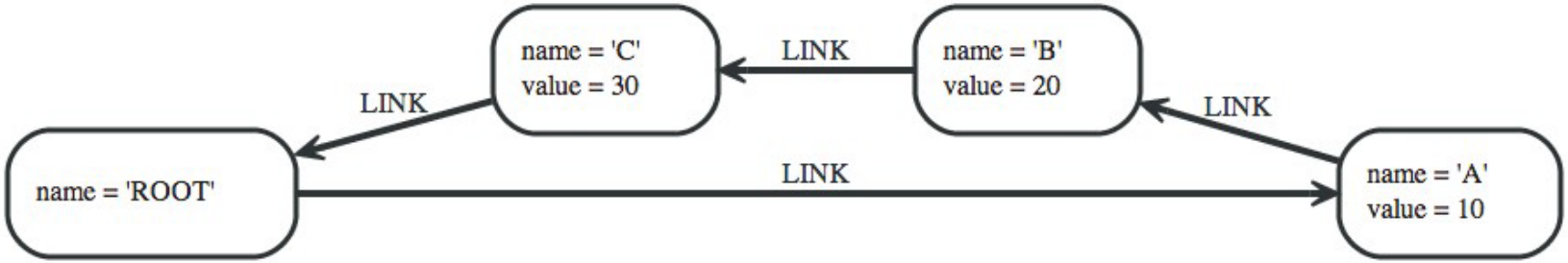
Relationships for querying

- like in other databases
  - same structure for different use-cases (OLTP and OLAP) doesn‘t work
  - graph allows: add more structures
- Relationships should the primary means to access nodes in the database
- Traversing relationships is cheap – that’s the whole design goal of a graph database
- Use lookups only to find starting nodes for a query

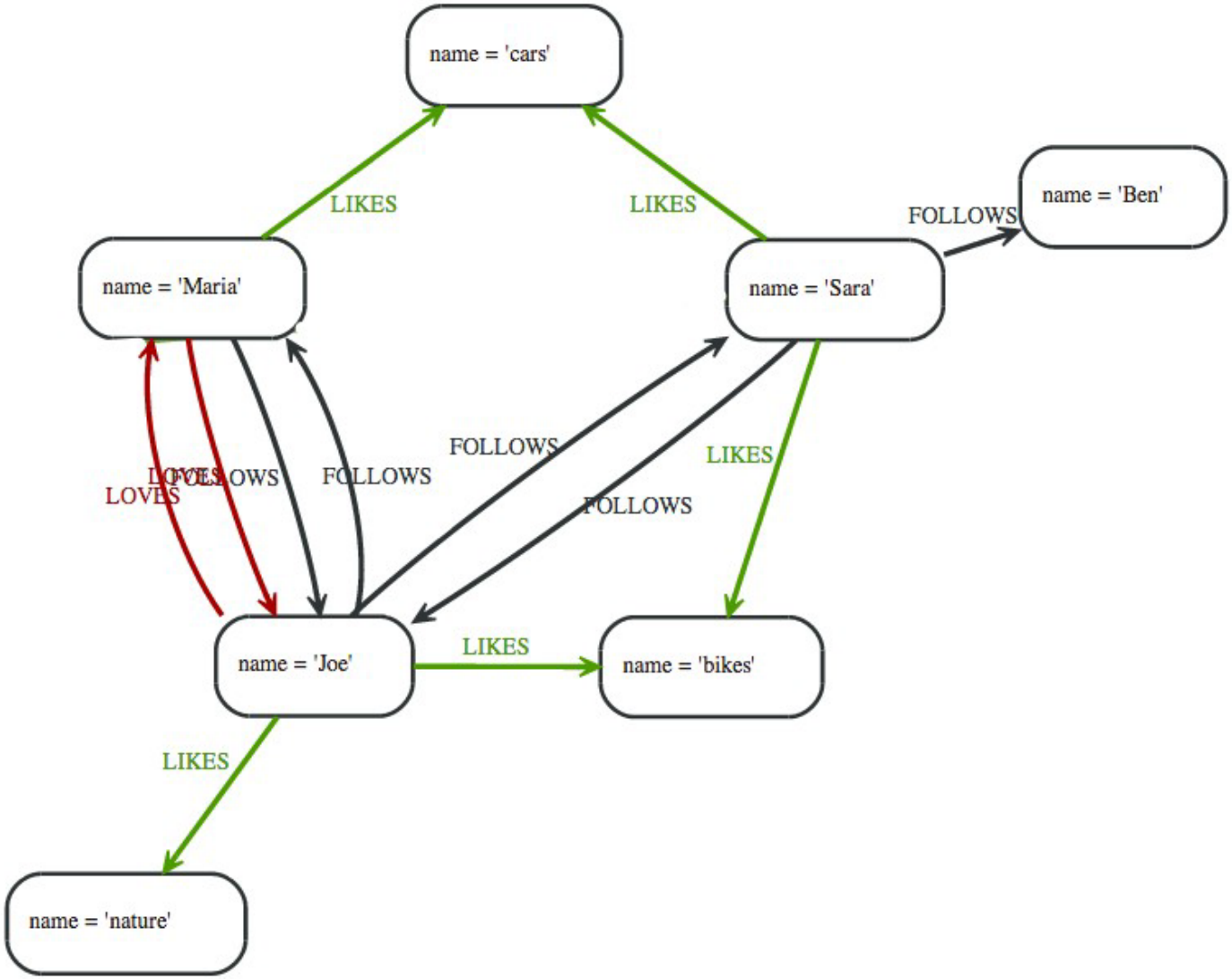
Anti-pattern: unconnected graph



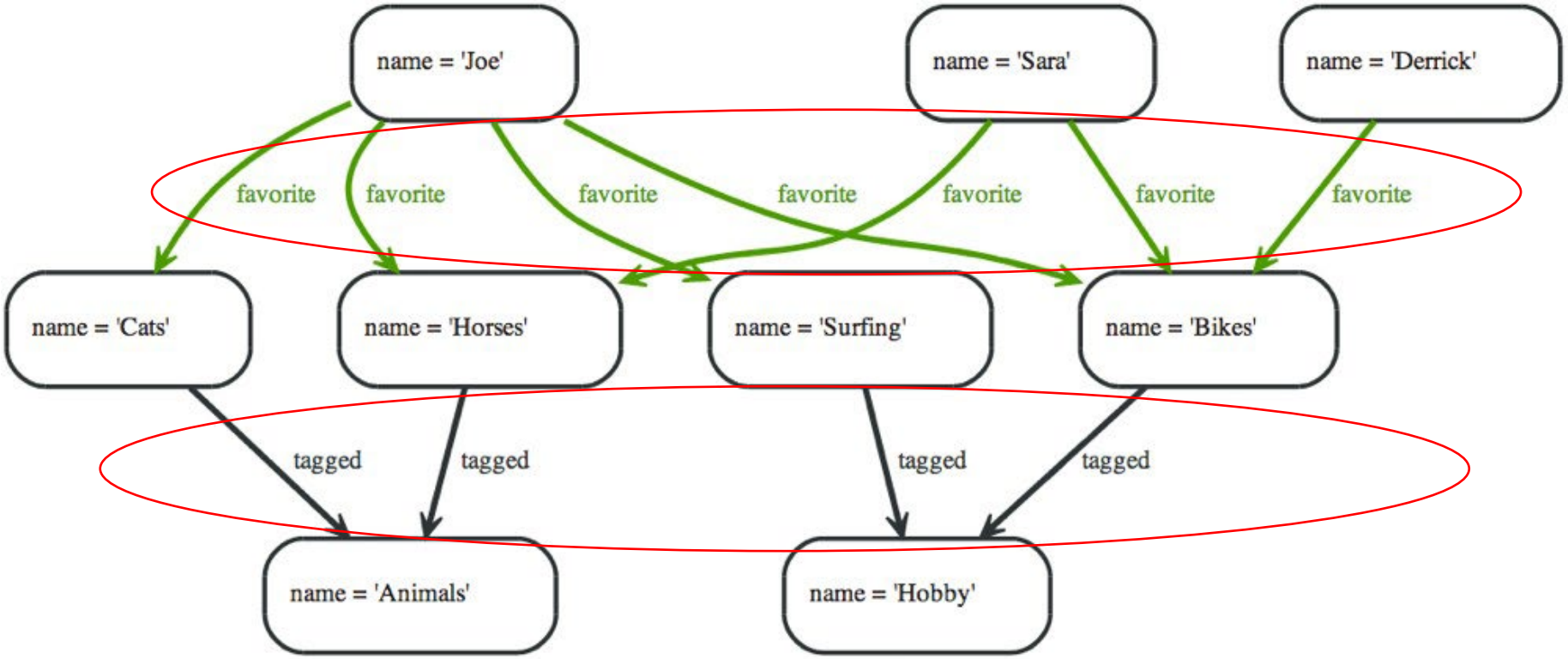
Pattern: Linked List



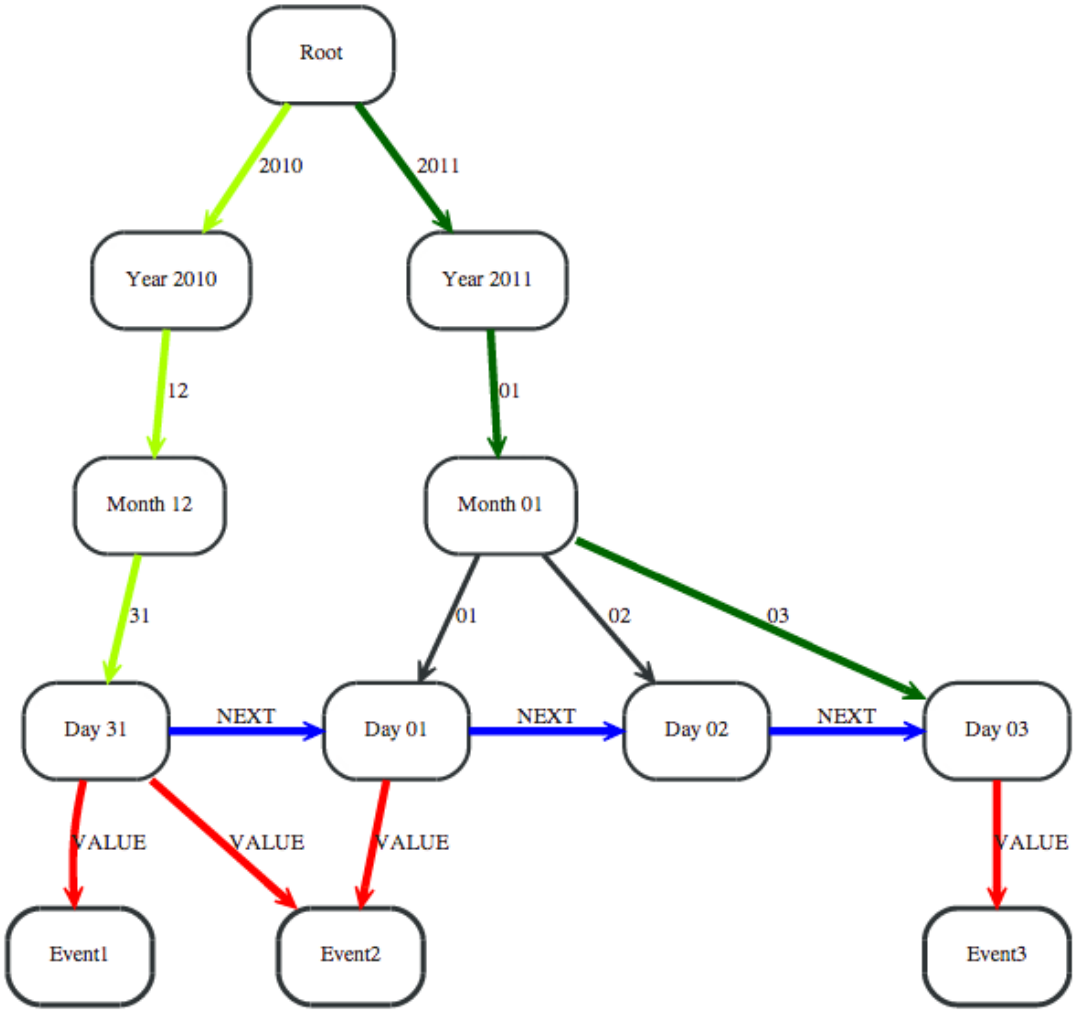
Pattern: Multiple Relationships



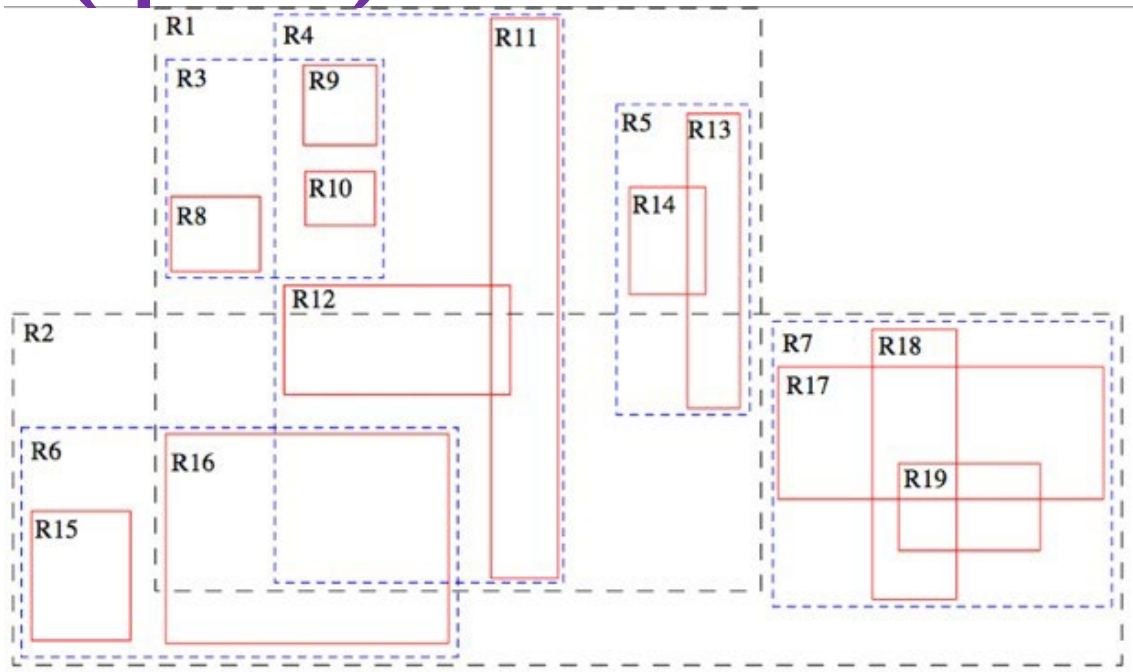
Pattern-Trees: Tags and Categories

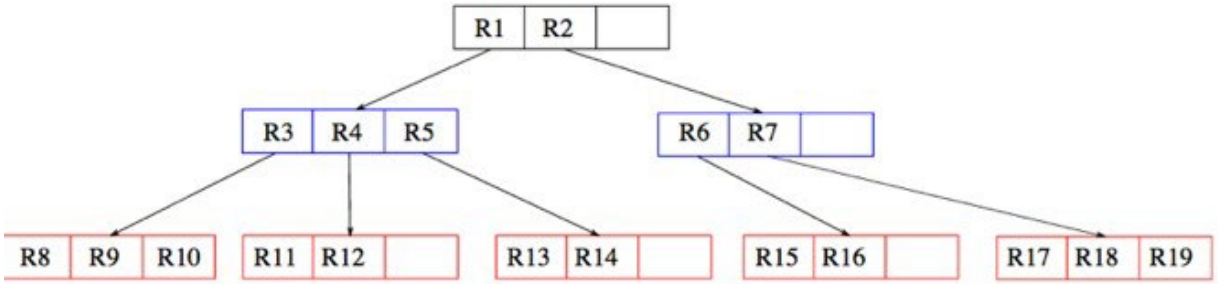


ADDENDUM: Pattern-Tree: Multi-Level-Tree

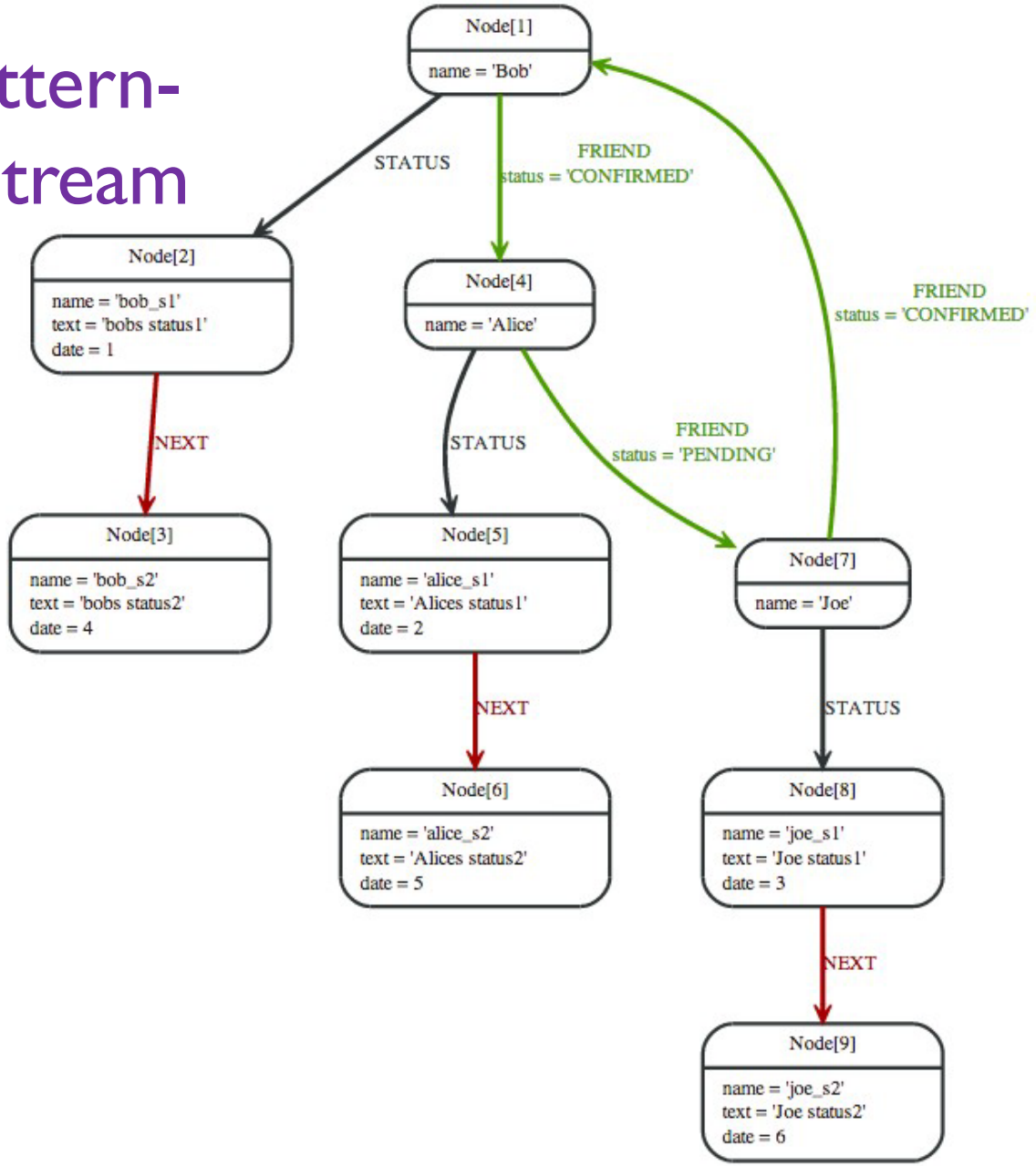


ADDENDUM: Pattern-Trees: R-Tree (spatial)





Example of Pattern-tree: Activity Stream



NORTHWESTERN
UNIVERSITY

- Intro/Motivation
- Basics of Graph Theory
  - Representations
  - Algorithms
- Relational vs. GraphDB
- Basics of GraphDB
  - …via actual Cypher…
- **GraphDB/Neo4j cont.**
  - **Context and more Cypher;**
  - **Notes from practical use-cases;**
    …when to use which…

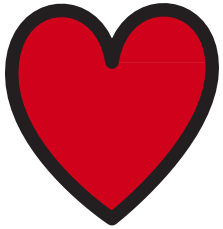
A bit of semantics and semantic web...

Recall: …GraphDB Gains

- Easy to model and store relationships
- Performance of relationship traversal remains constant with growth in data size
- Queries are shortened and more readable
- Adding additional properties and relationships can be done on the fly - no migrations

CYPHER

Ann

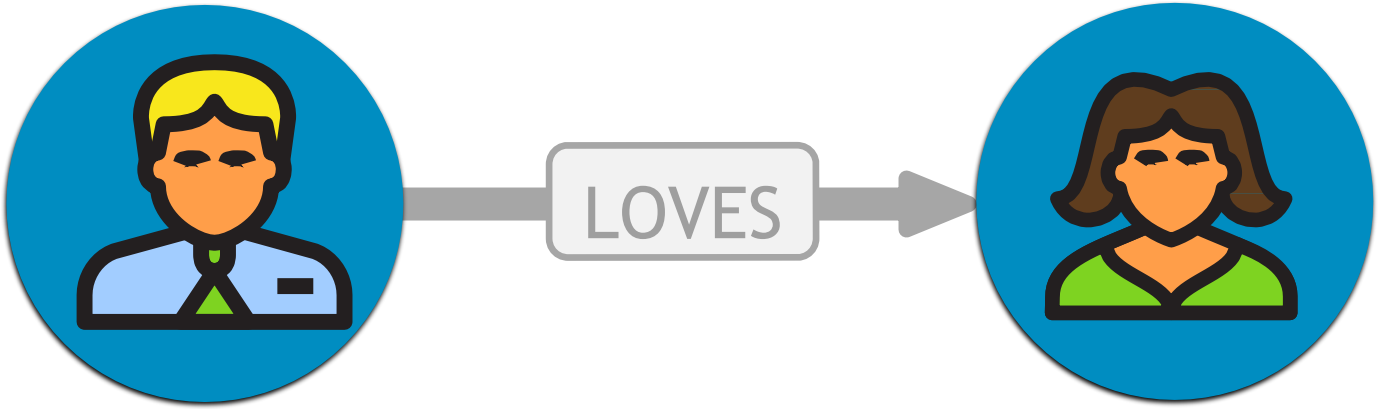


Loves

Dan

NORTHWESTERN UNIVERSITY

Property Graph Model



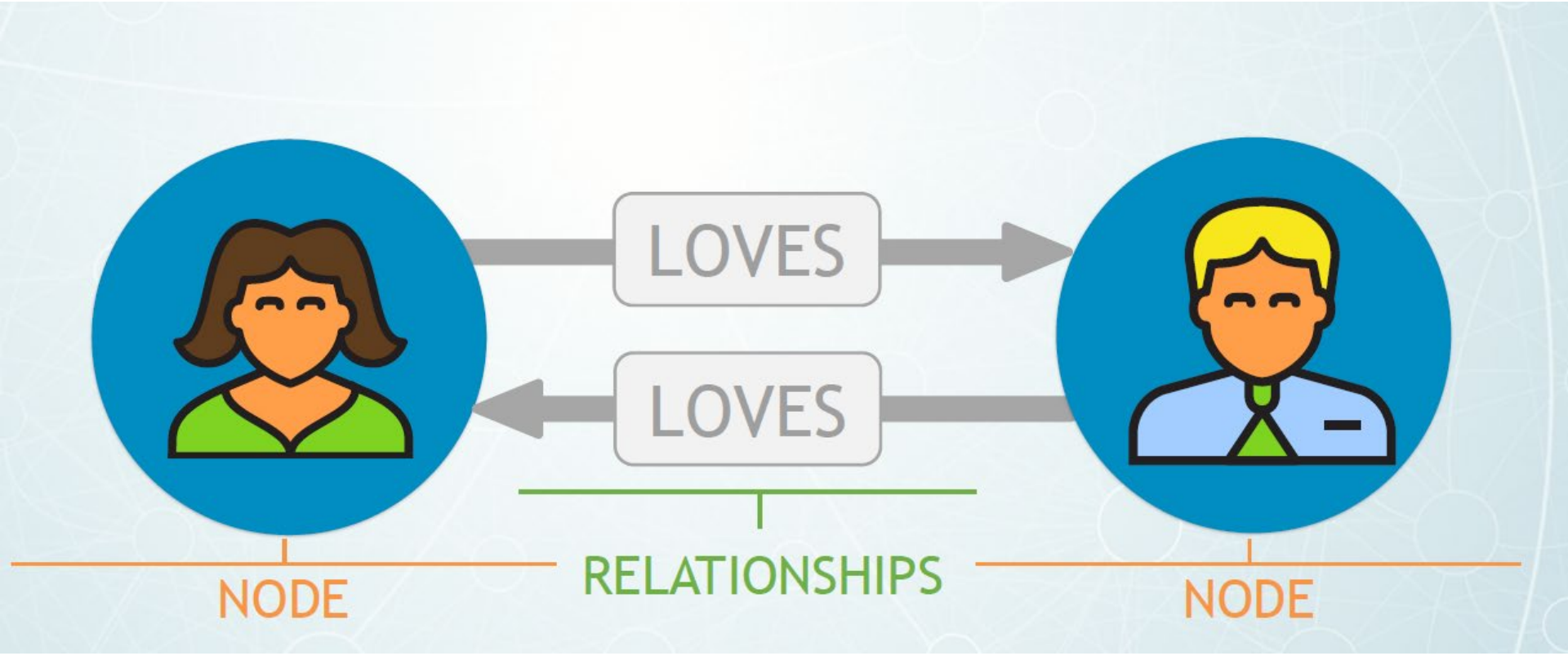
NODE NODE

CREATE (:Person { name:“Dan”}) - [:LOVES]-> (:Person { name:“Ann”})

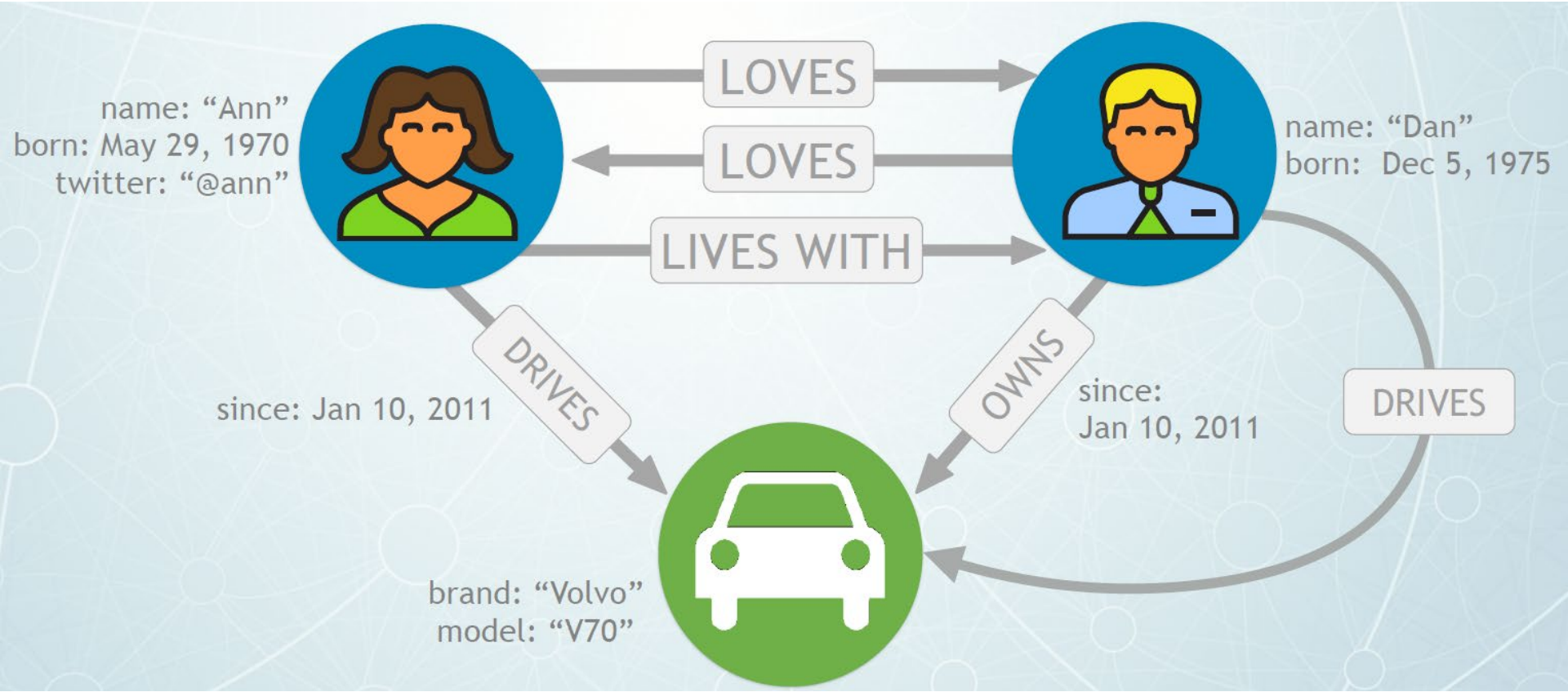
LABEL PROPERTY LABEL PROPERTY

NOTE: relationships are directional…

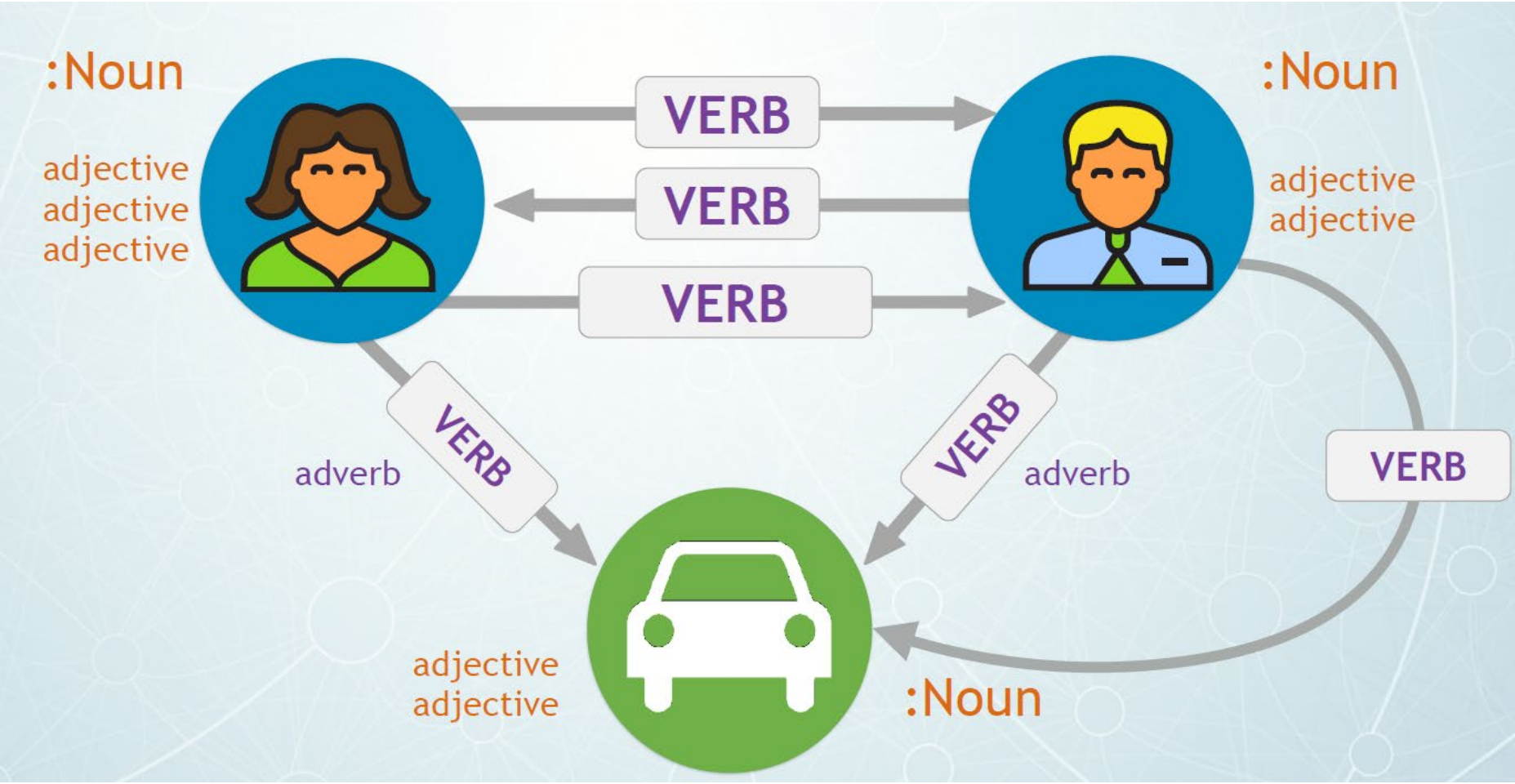
NORTHWESTERN UNIVERSITY



Of course: more details…



Similar to ER design…



Once again, basic components:

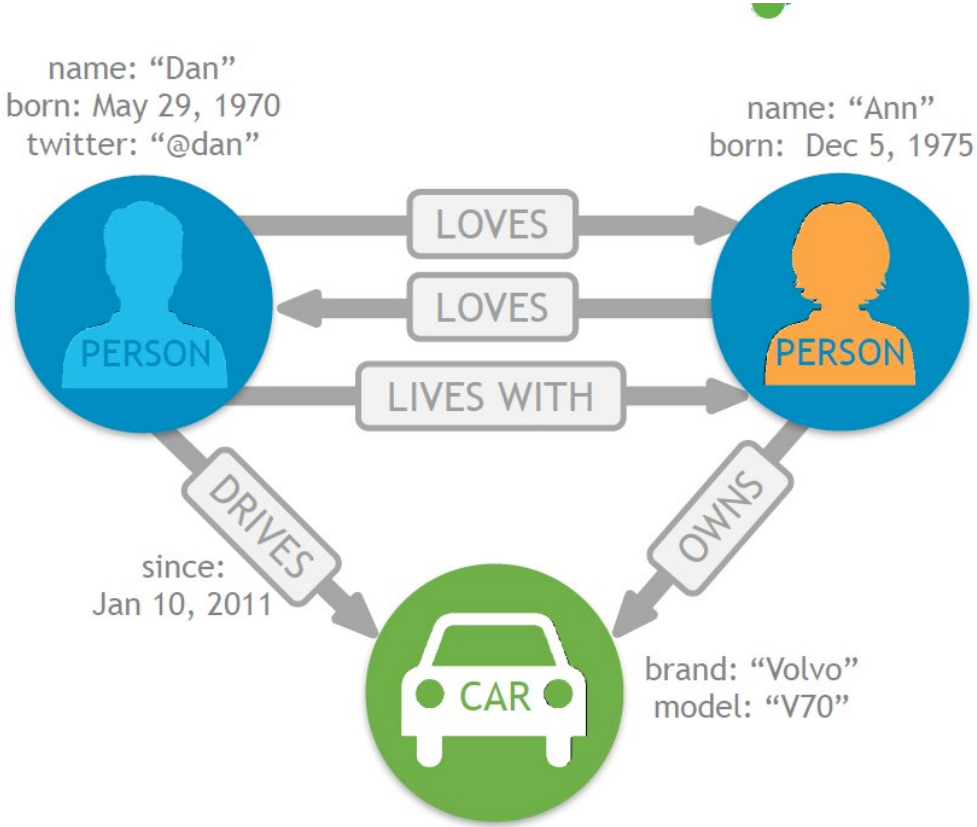
NORTHWESTERN UNIVERSITY

Nodes

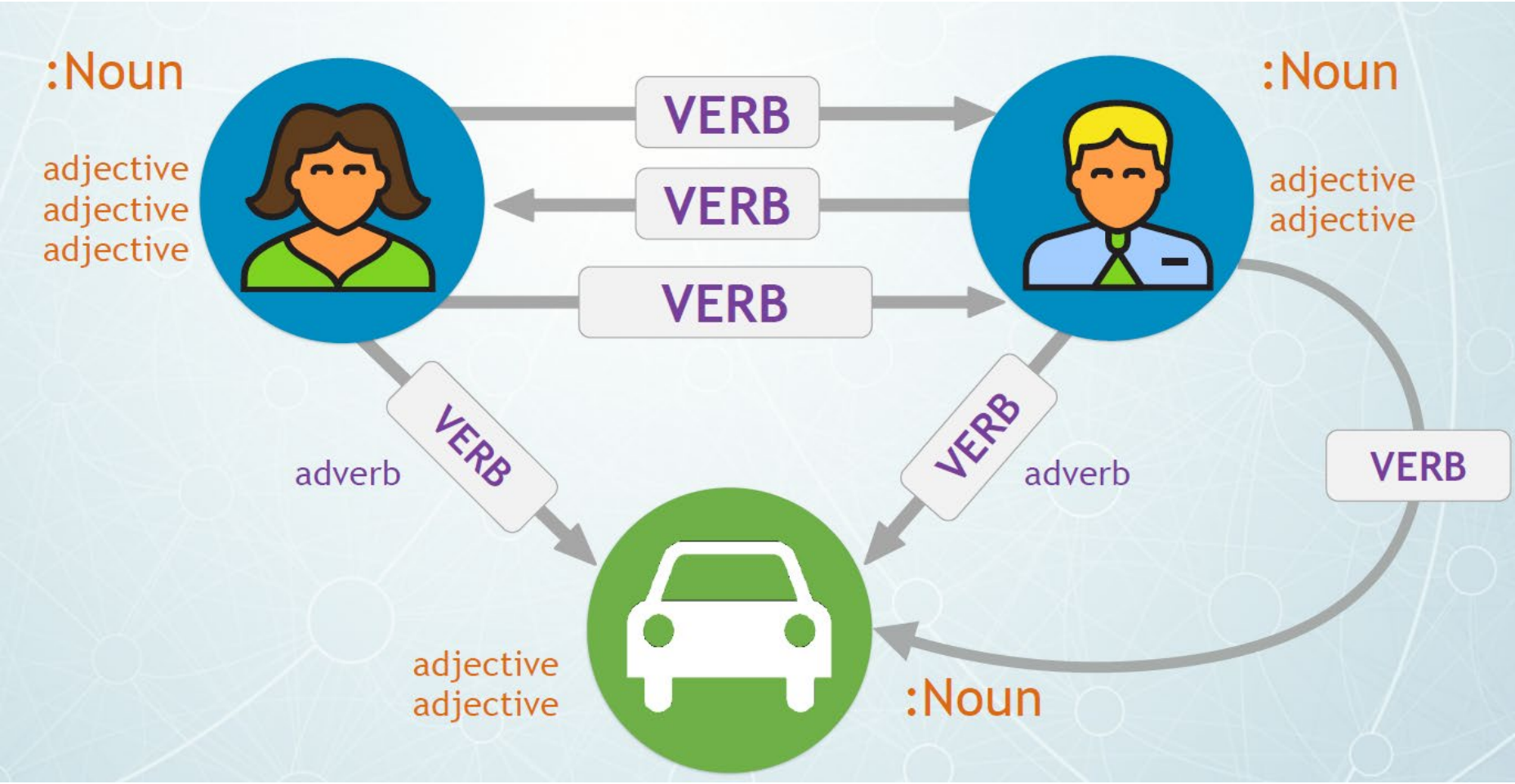
- The objects in the graph
- Can have name-value *properties*
- Can be *labeled*

Relationships

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



Recall: Similar to ER design…



Recall: basic components…

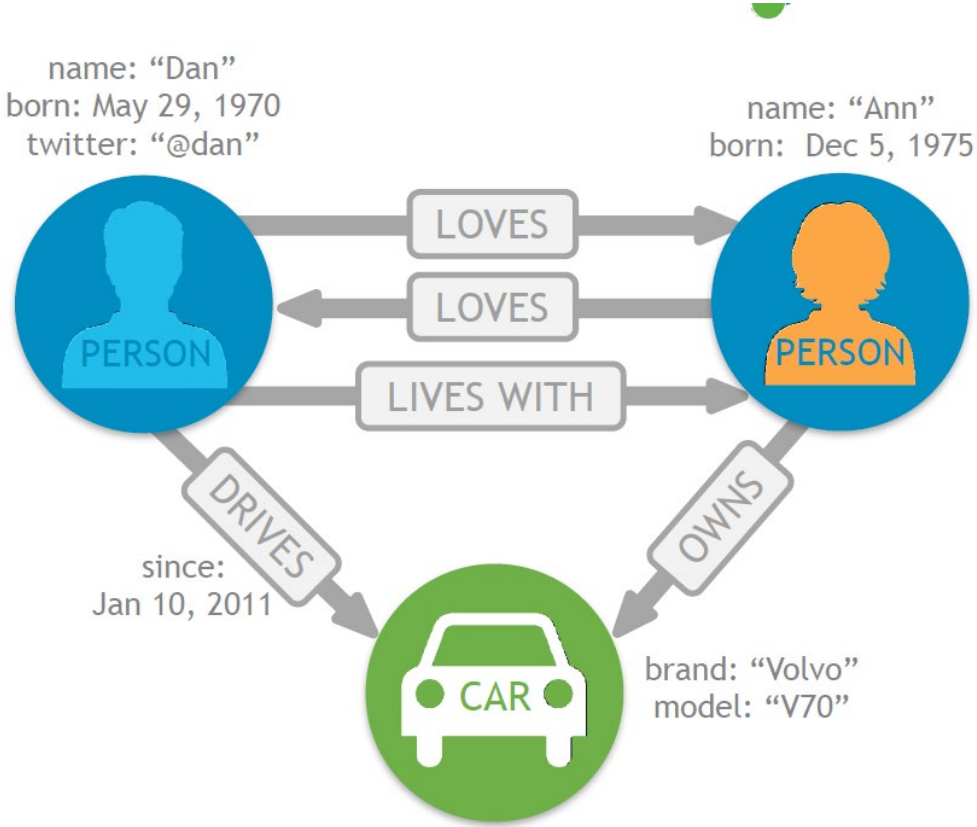
NORTHWESTERN UNIVERSITY

Nodes

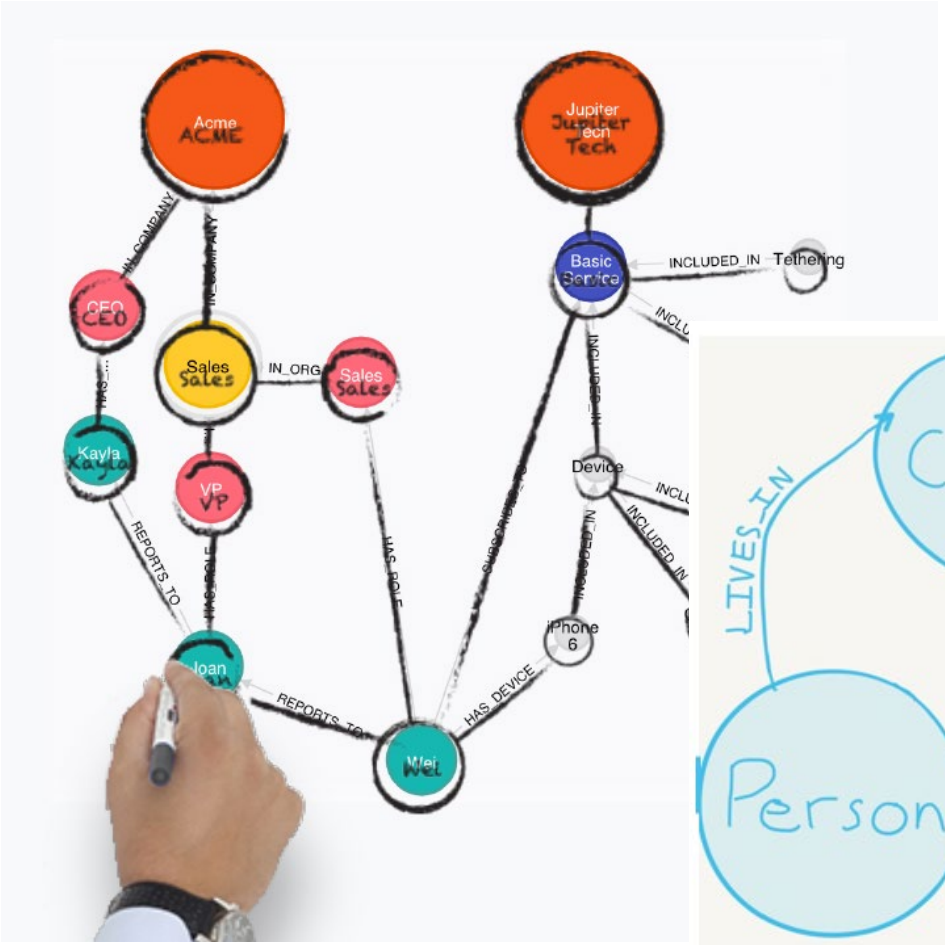
- The objects in the graph
- Can have name-value *properties*
- Can be *labeled*

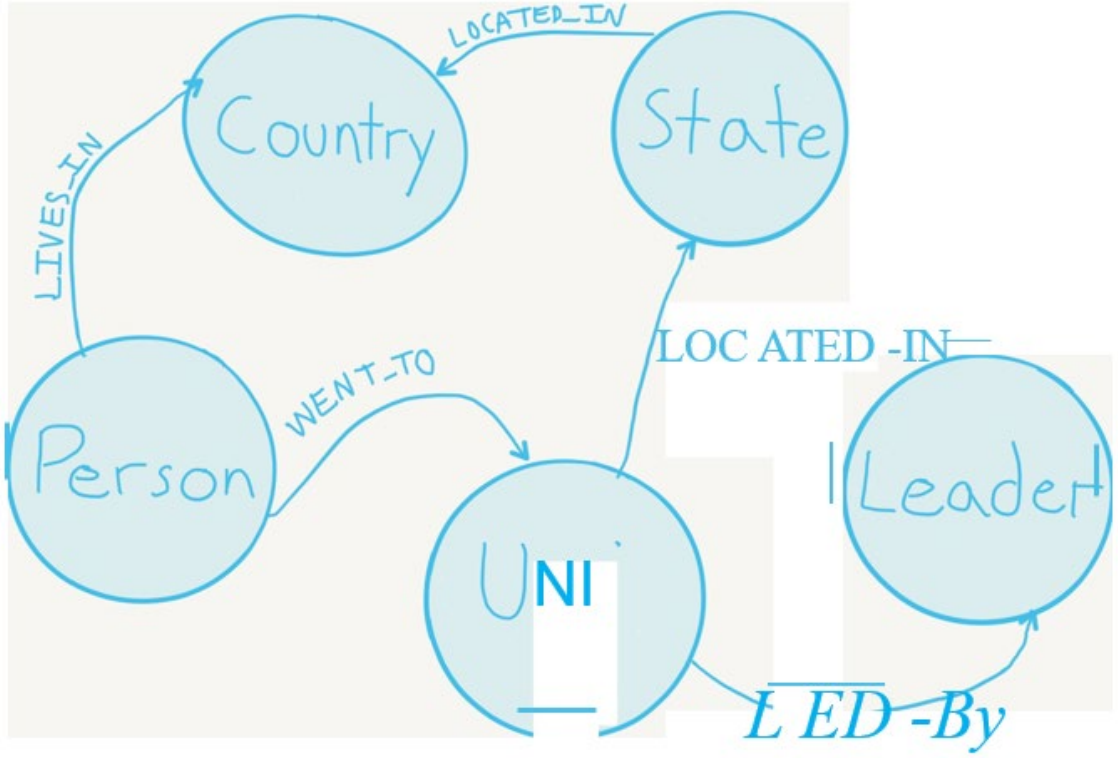
Relationships

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

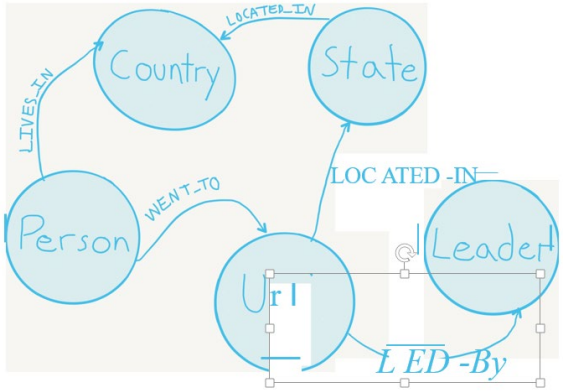


Simple AND INTUITIVE design (model)



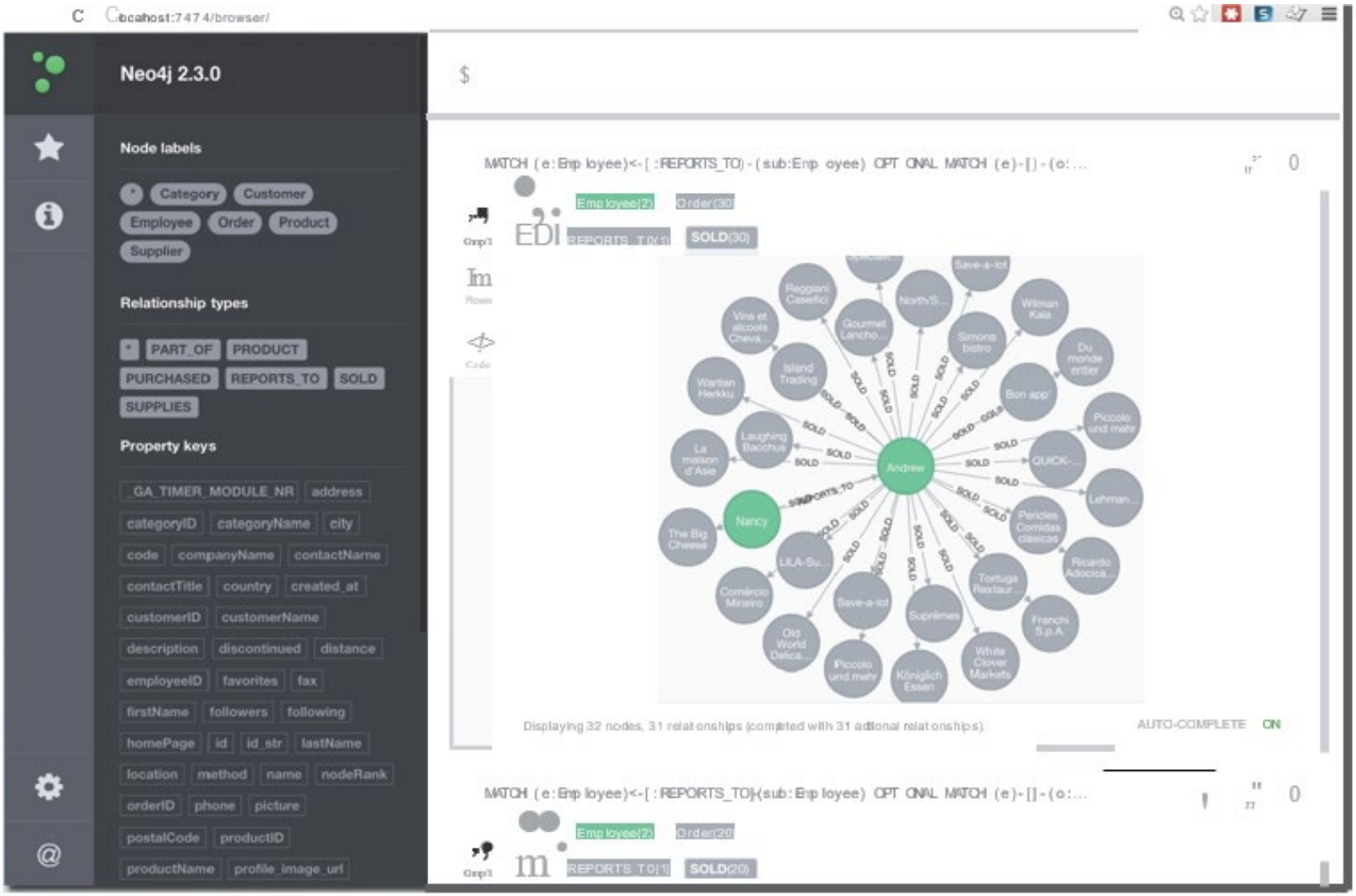


Simple design -> simple querying



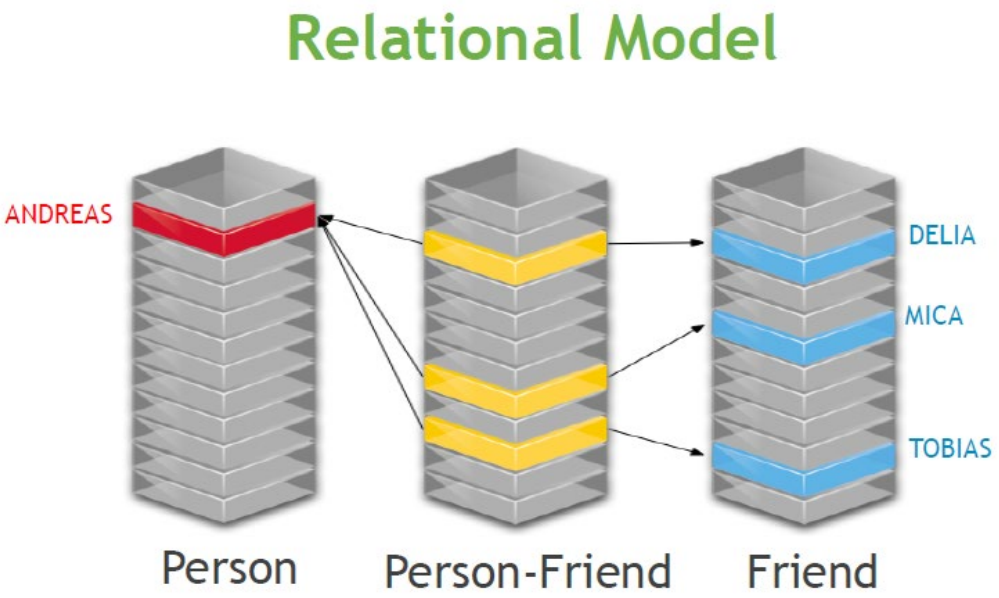

```
MATCH
(p:Person)-[:WENT_TO]->(u:Uni),
(p)-[:LIVES_IN]->(c:Country),
(u)-[:LED_BY]->(l:Leader),
(u)-[:LOCATED_IN]->(s:State)
WHERE s.abbr = 'CT'
RETURN
p.name,
c.country, c.leader, p.hair, u.name, l.name, s.abbr
```

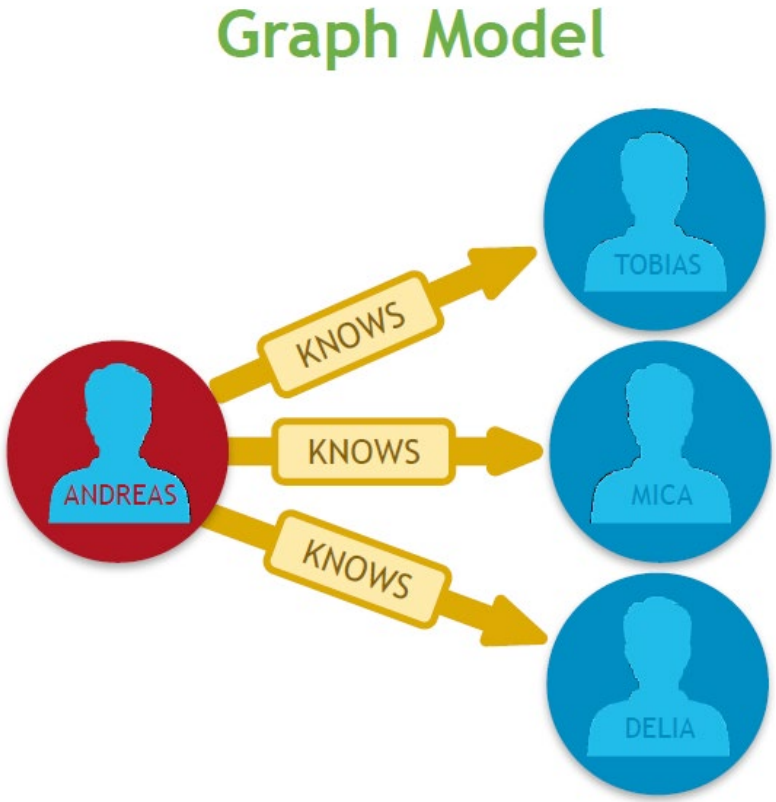
Simple UI


localhost:7474/browser/
Neo4j 2.3.0
Node labels
Category
Customer
Employee
Order
Product
Supplier
Relationship types
PART_OF
PRODUCT
PURCHASED
REPORTS_TO
SOLD
SUPPLIES
Property keys
address
categoryID
categoryName
city
code
companyName
contactName
contactTitle
country
created_at
customerID
customerName
description
discontinued
distance
employeeID
favorites
fax
firstName
followers
following
homePage
id
id_str
lastName
location
method
name
nodeRank
orderID
phone
picture
postalCode
productID
productName
profile_image_url
$
MATCH (e:Employee)<-[:REPORTS_TO]-(sub:Employee) OPT ONAL MATCH (e)-[]-(o:...
Employee(2)
Order(30)
SOLD(30)
Andrew
Nancy
Displaying 32 nodes, 31 relationships (completed with 31 additional relationships)
AUTO-COMPLETE ON
MATCH (e:Employee)<-[:REPORTS_TO]-(sub:Employee) OPT ONAL MATCH (e)-[]-(o:...
Employee(2)
Order(20)
REPORTS_TO(1)
SOLD(20)

Recall: Relational vs. Graph…

NORTHWESTERN UNIVERSITY

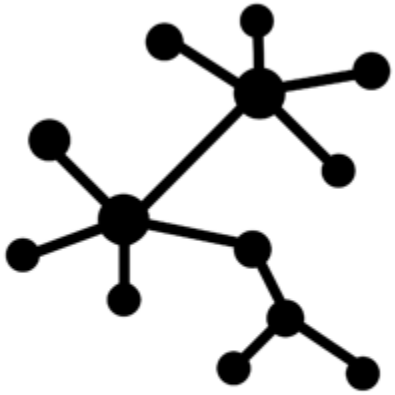




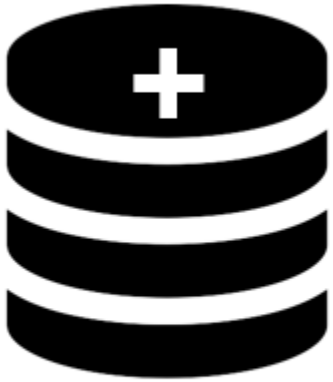
How to use Neo4J

NORTHWESTERN UNIVERSITY

- Rather simple…



CREATE MODEL

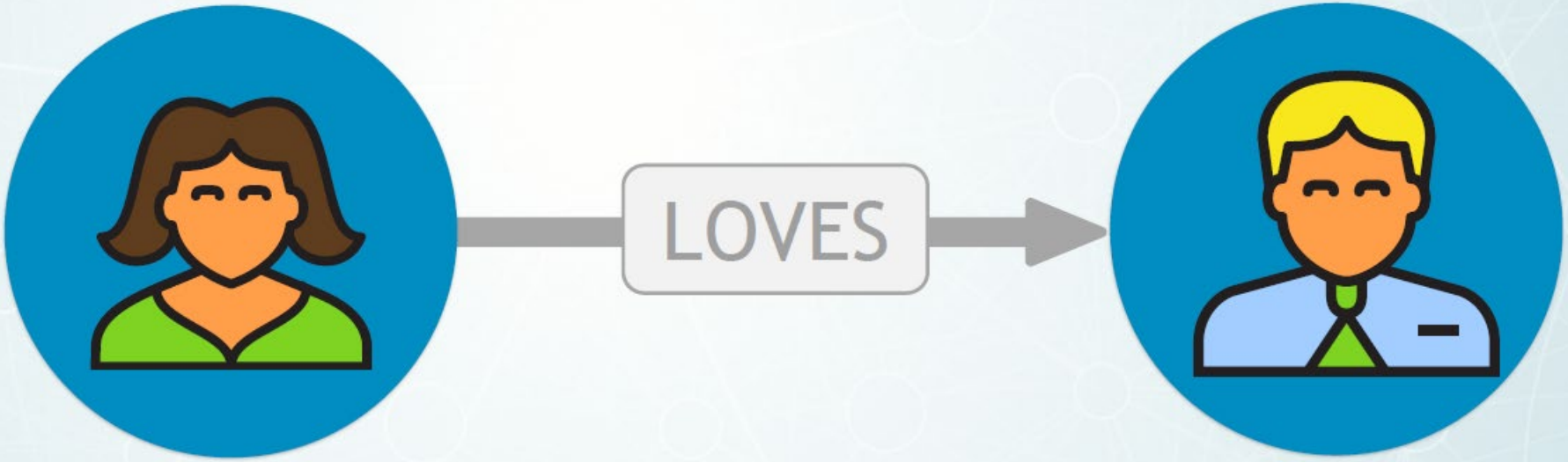


LOAD DATA

QUERY DATA

Creating GraphDB in Cypher

NORTHWESTERN UNIVERSITY



NODE NODE

```
CREATE (:Person { name:“Ann”} ) - [:LOVES]-> (:Person { name:“Dan”} )
```

LABEL PROPERTY LABEL PROPERTY

Bidirectional…

```
MATCH (:Person { name:“Ann”} ) - [:FB_FRIENDS] -> (:Person { name:“Dan”} )

MATCH (:Person { name:“Ann”} ) - [:FB_FRIENDS] - (:Person { name:“Dan”} )
```

Loading data in Neo4j

- Options:

1: CSV

orders.csv

employees | customers | products.c | orders.csv | order-deta | regions.cs | shippers.c | suppliers.c | territories | categories | employee-

```
1  OrderID,CustomerID,EmployeeID,OrderDate,RequiredDate,ShippedDate,ShipVia,Freigh
   t,ShipName,ShipAddress,ShipCity,ShipRegion,ShipPostalCode,ShipCountry
2  10248,VINET,5,1996-07-04 00:00:00.000,1996-08-01 00:00:00.000,1996-07-16
   00:00:00.000,3,32.38,Vins et alcools Chevalier,59 rue de
   l'Abbaye,Reims,NULL,51100,France
3  10249,TOMSP,6,1996-07-05 00:00:00.000,1996-08-16 00:00:00.000,1996-07-10
   00:00:00.000,1,11.61,Toms Spezialitäten,Luisenstr.
   48,Münster,NULL,44087,Germany
4  10250,HANAR,4,1996-07-08 00:00:00.000,1996-08-05 00:00:00.000,1996-07-12
   00:00:00.000,2,65.83,Hanari Carnes,Rua do Paço, 67,Rio de Janeiro,RJ,05454-
   876,Brazil
5  10251,VICTE,3,1996-07-08 00:00:00.000,1996-08-05 00:00:00.000,1996-07-15
   00:00:00.000,1,41.34,Victuailles en stock,2, rue du
   Commerce,Lyon,NULL,69004,France
6  10252,SUPRD,4,1996-07-09 00:00:00.000,1996-08-06 00:00:00.000,1996-07-11
   00:00:00.000,2,51.30,Suprêmes délices,Boulevard Tirou, 255,Charleroi,NULL,B-
   6000,Belgium
7  10253,HANAR,3,1996-07-10 00:00:00.000,1996-07-24 00:00:00.000,1996-07-16
```

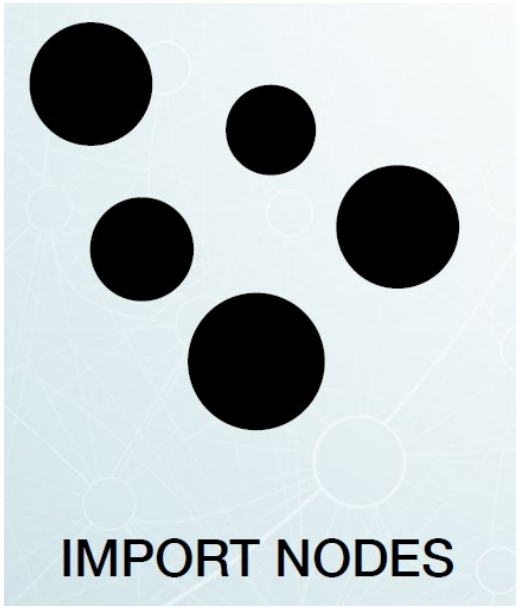
Line 1, Column 1 Tab Size: 4 Plain Text

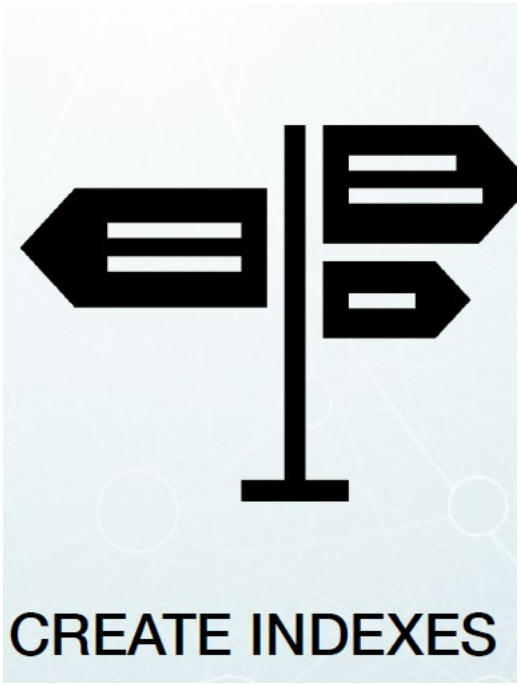
Loading data in Neo4j

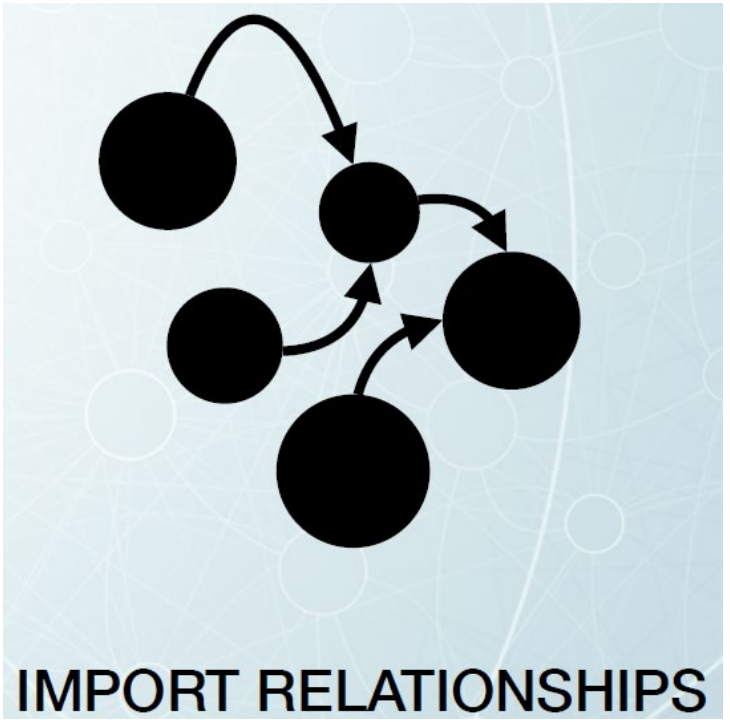
NORTHWESTERN UNIVERSITY

- Options:

2: Load multiple objects







Loading data in Neo4j

NORTHWESTERN UNIVERSITY

```
// Create customers
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "https://
raw.githubusercontent.com/neo4j-contrib/developer-resources/
gh-pages/data/northwind/customers.csv" AS row
CREATE (:Customer {companyName: row.CompanyName, customerID:
row.CustomerID, fax: row.Fax, phone: row.Phone});

// Create products
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "https://
raw.githubusercontent.com/neo4j-contrib/developer-resources/
gh-pages/data/northwind/products.csv" AS row
CREATE (:Product {productName: row.ProductName, productID:
row.ProductID, unitPrice: toFloat(row.UnitPrice)});
```

Loading data in Neo4j – not that bad…

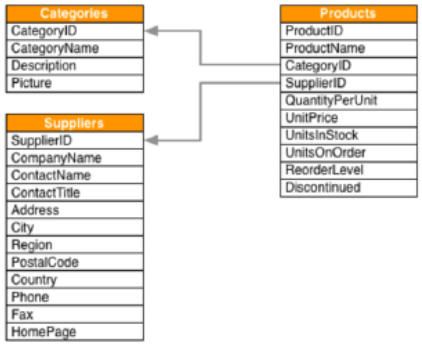
NORTHWESTERN UNIVERSITY

```
:play northwind graph
```

Product Catalog

Northwind sells food products in a few categories, provided by suppliers. Let's start by loading the product catalog tables.

The load statements to the right require public internet access. `LOAD CSV` will retrieve a CSV file from a valid URL, applying a Cypher statement to each row using a named map (here we're using the name `row`).



:help cypher LOAD CSV

Load records

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
  n.unitPrice = toFloat(row.unitPrice),
  n.unitsInStock = toInt(row.unitsInStock), n.unitsOnOrder = toInt(row.unitsOnOrder),
  n.reorderLevel = toInt(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Create indexes

```
CREATE INDEX ON :Product(productID)
```

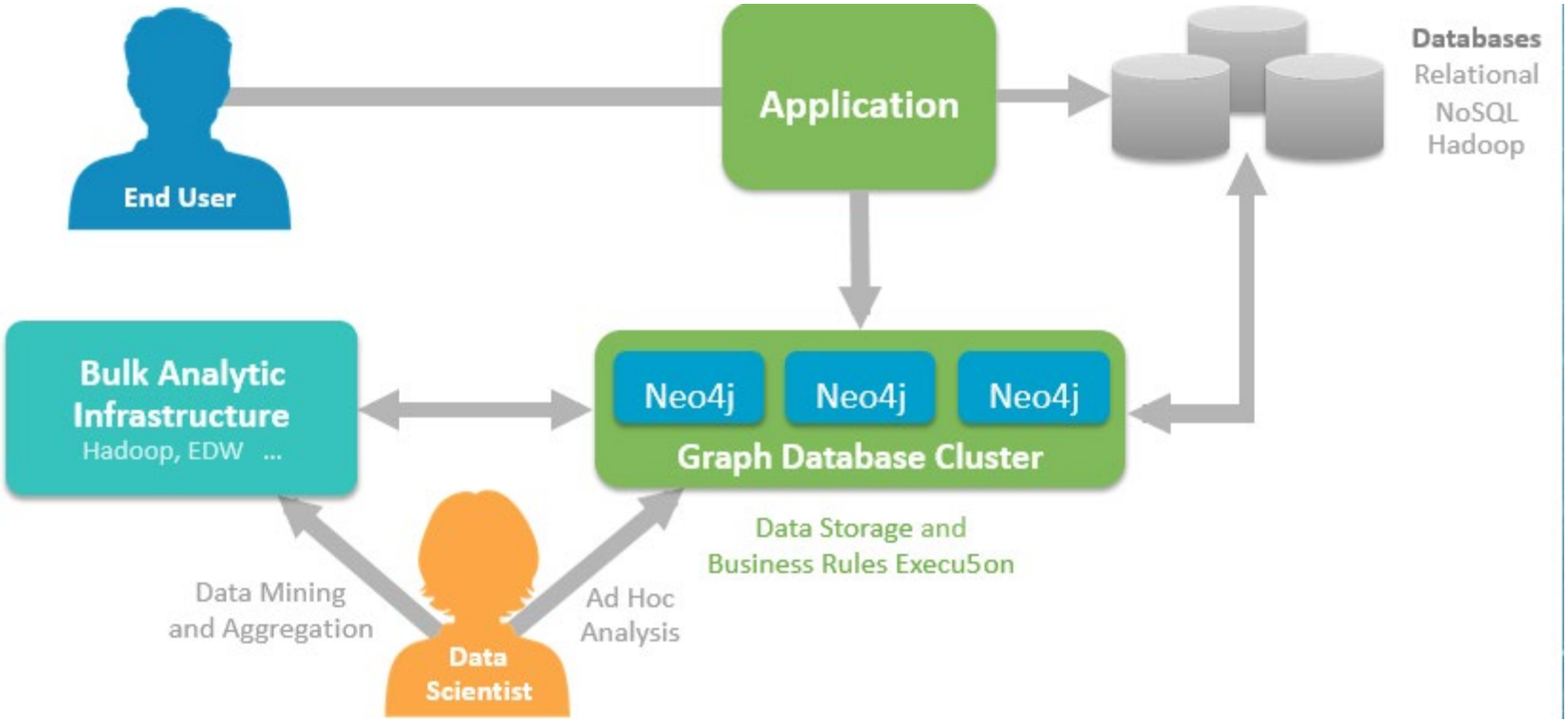
```
CREATE INDEX ON :Category(categoryID)
```

```
CREATE INDEX ON :Supplier(supplierID)
```

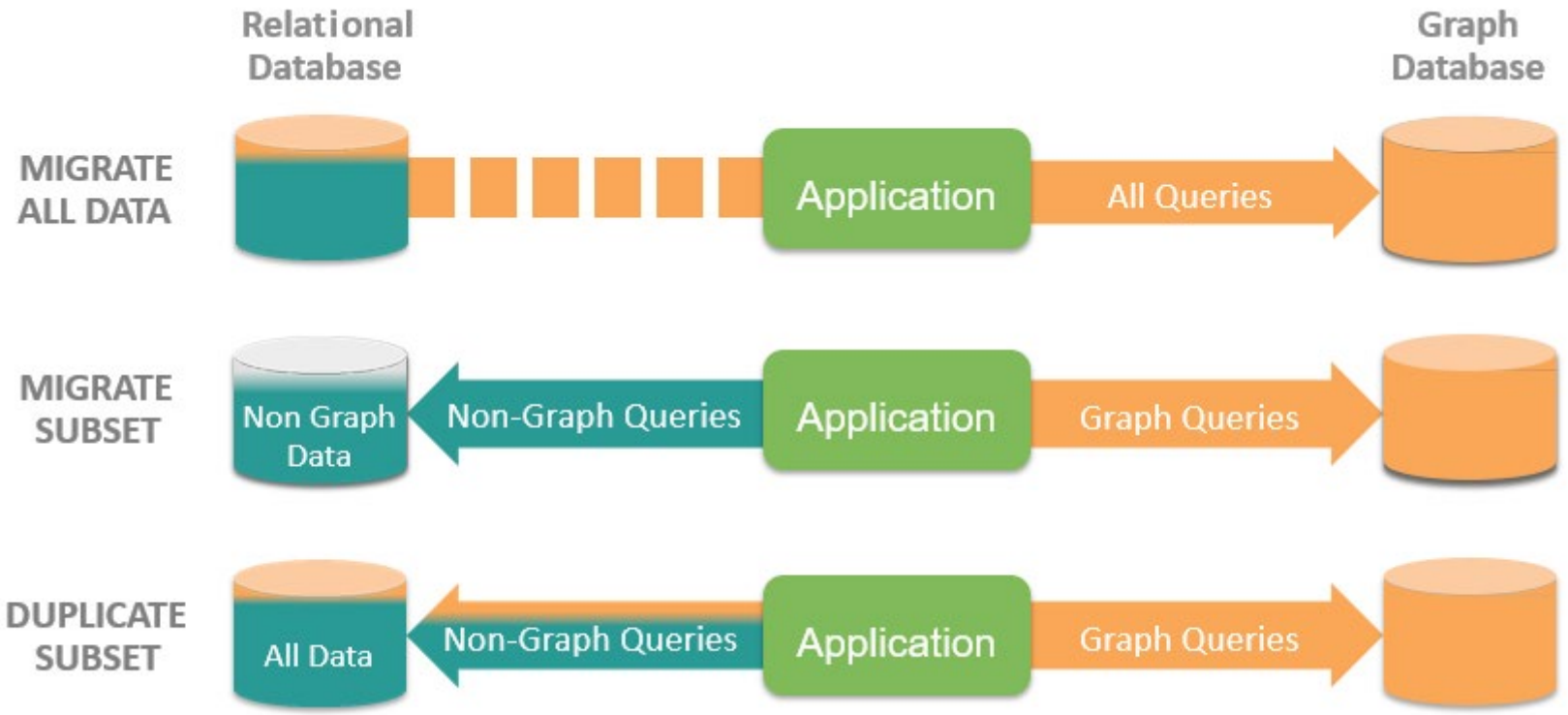
NOTE: 4.5 million nodes and their relationship from Dbpedia loads in < 100 seconds

Options…

NORTHWESTERN UNIVERSITY

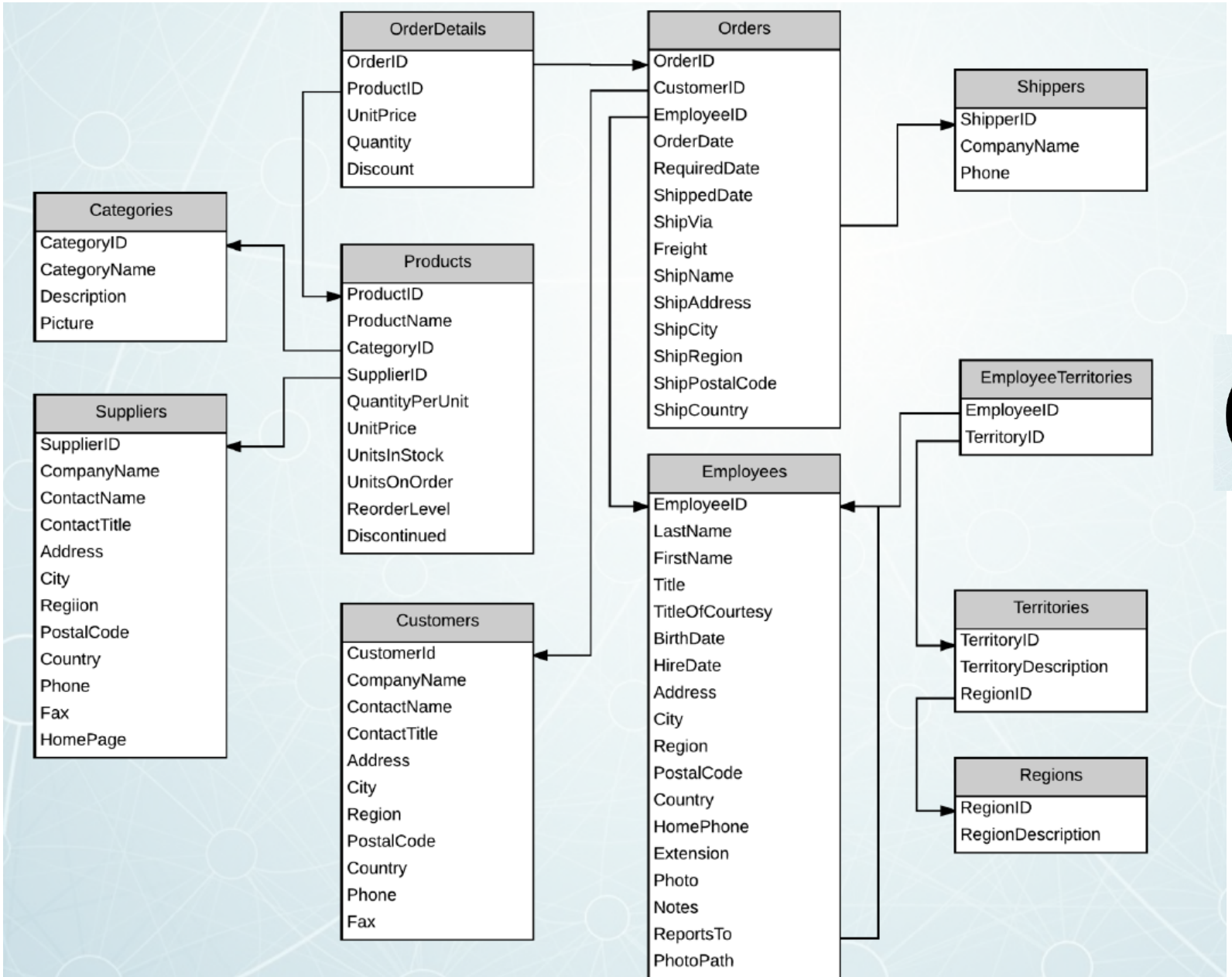


Options… Data Migration

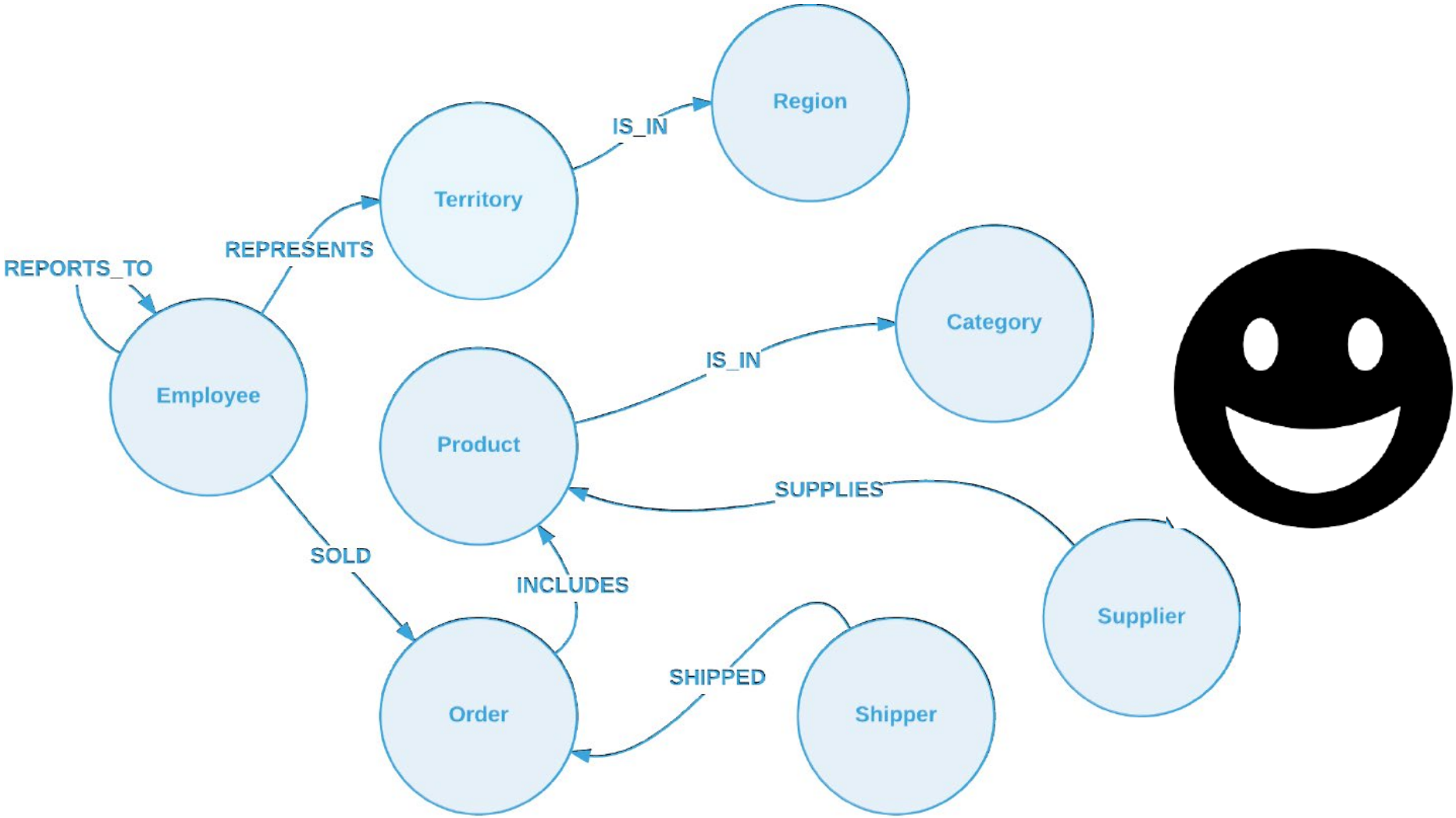


Detailed case… “good old” Northwind

NORTHWESTERN UNIVERSITY

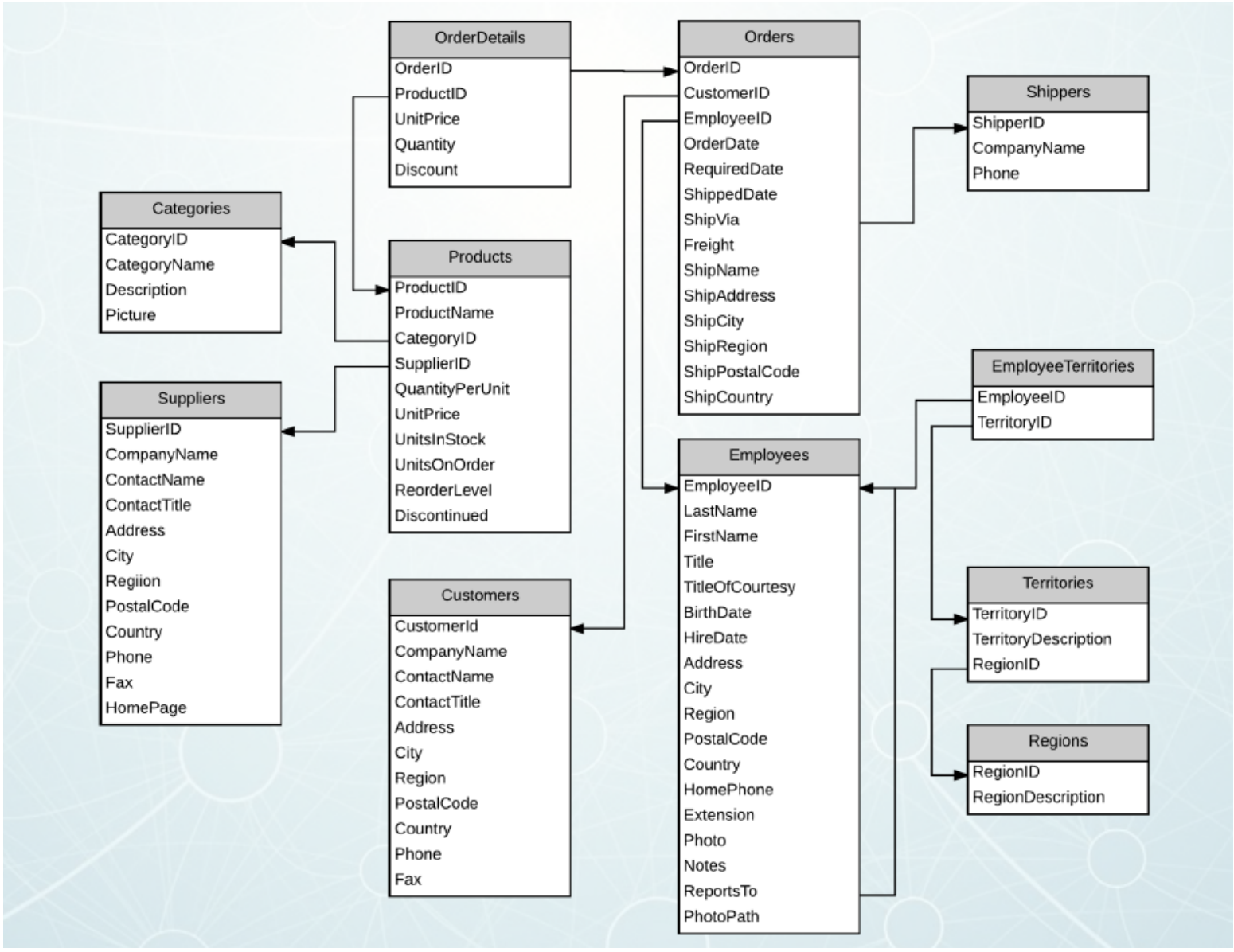


Northwind – GraphDB model



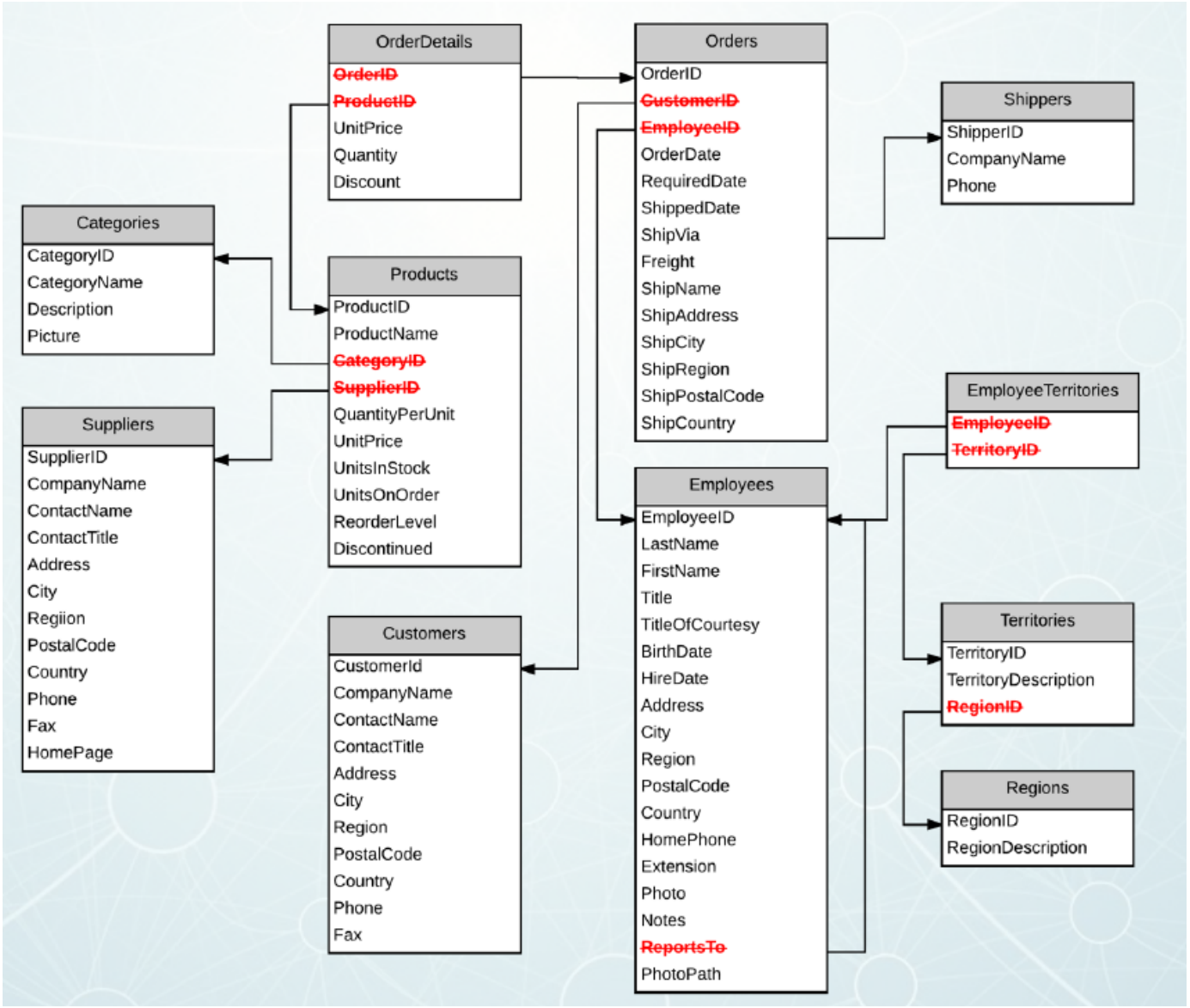
How to go from RDB to GraphDB?

Start with the ER diagram…



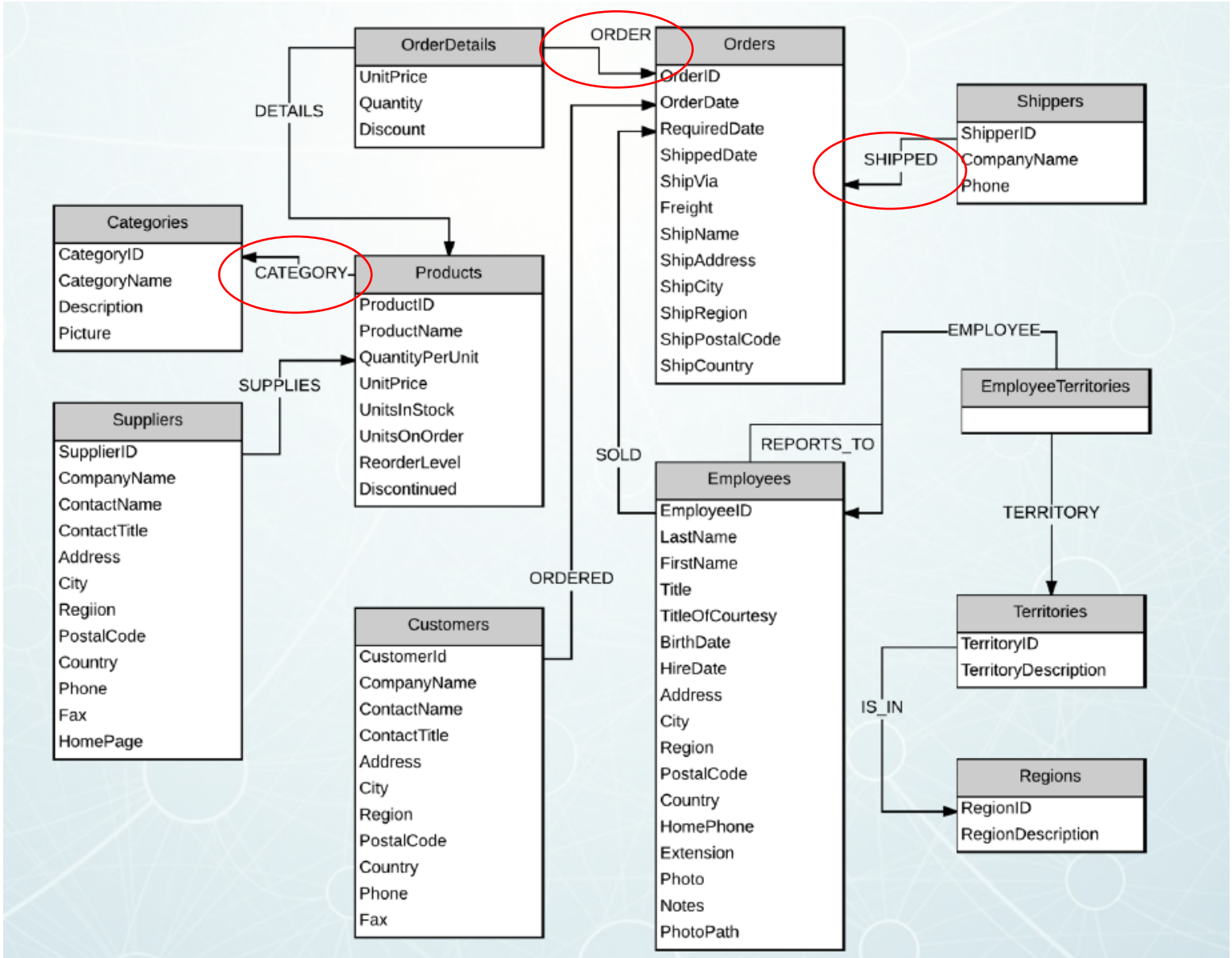
How to go from RDB to GraphDB?

- Locate the foreign keys



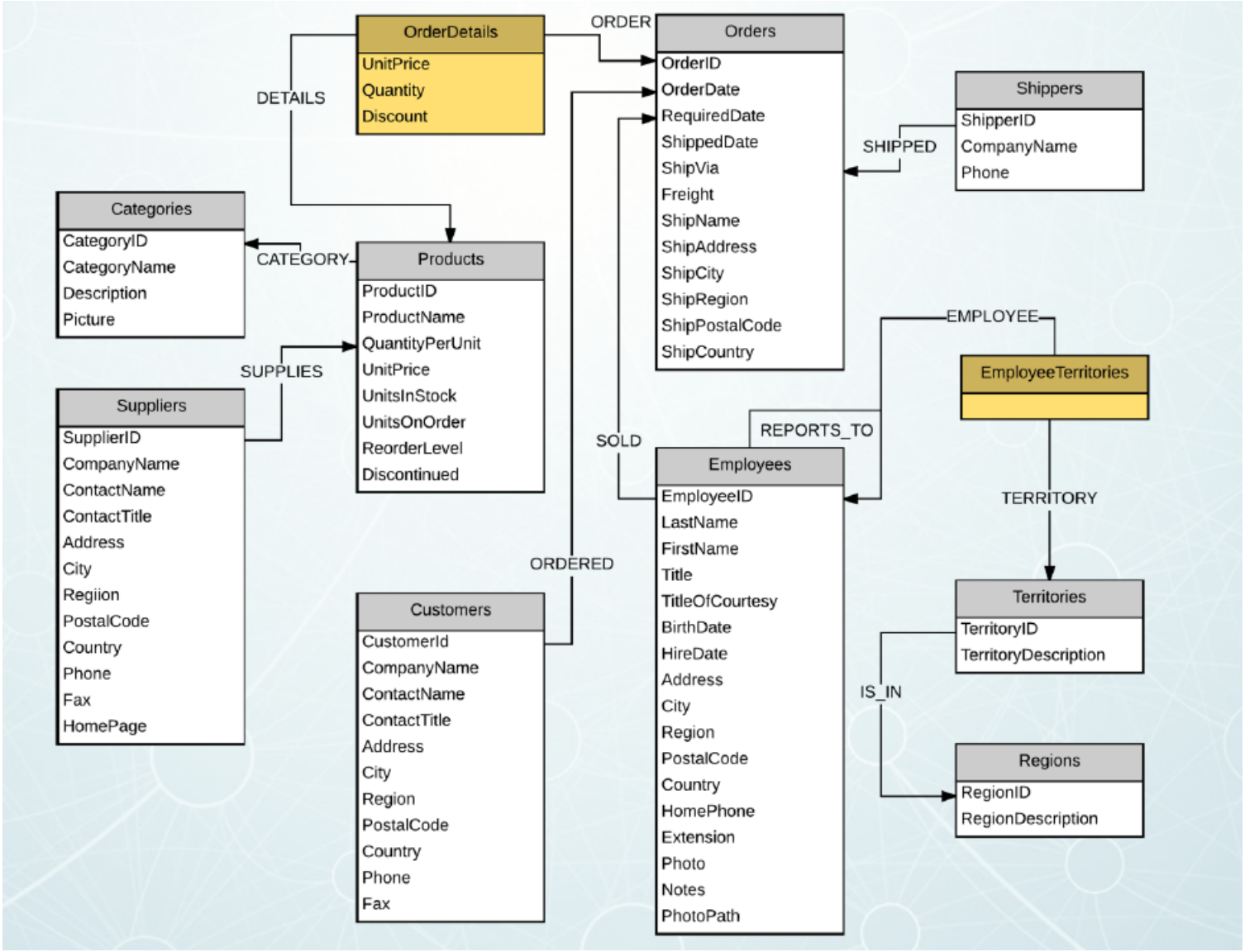
How to go from RDB to GraphDB…

- Then, drop the foreign keys, and label the relationships



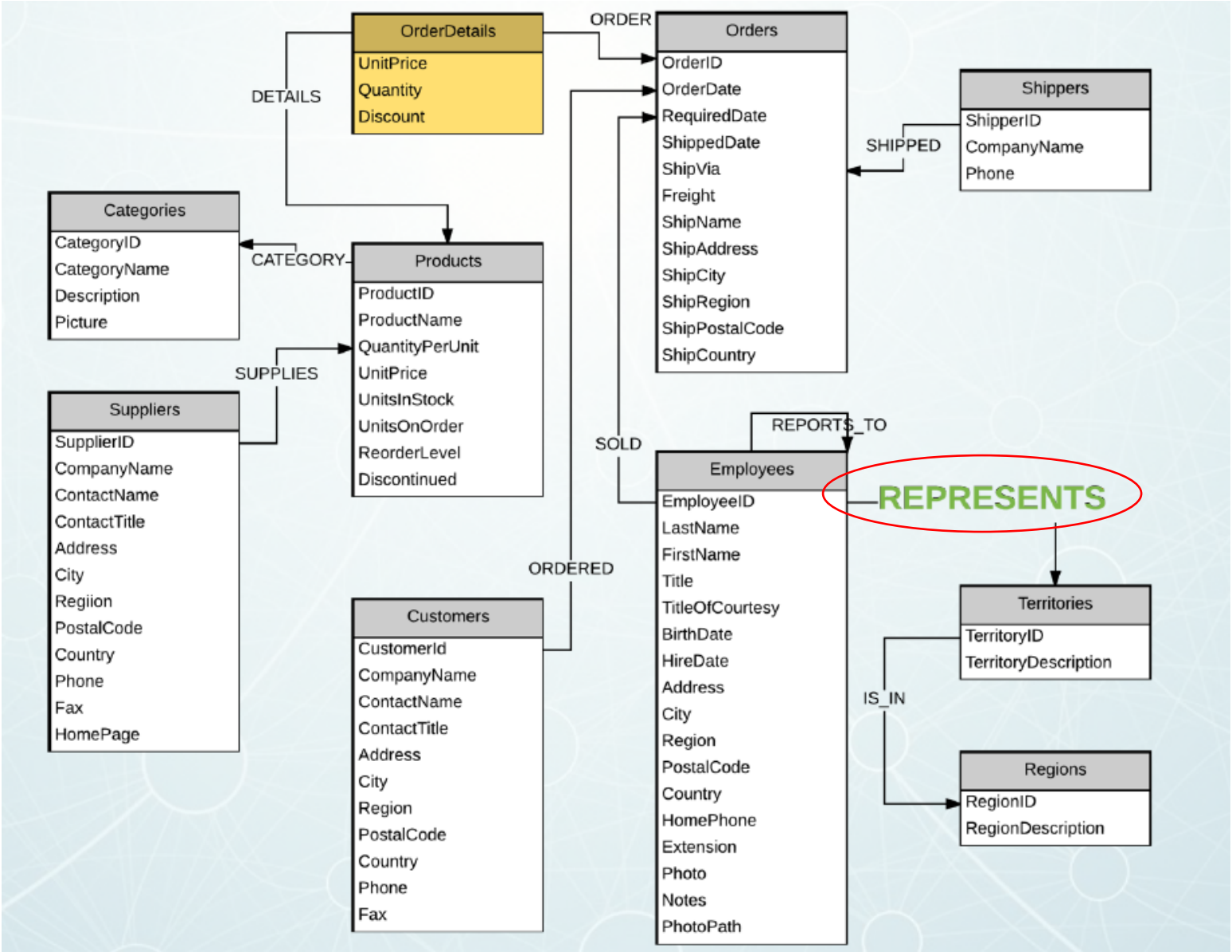
How to go from RDB to GraphDB

- Next step: find the JOIN-ing tables (i.e., M-to-M)



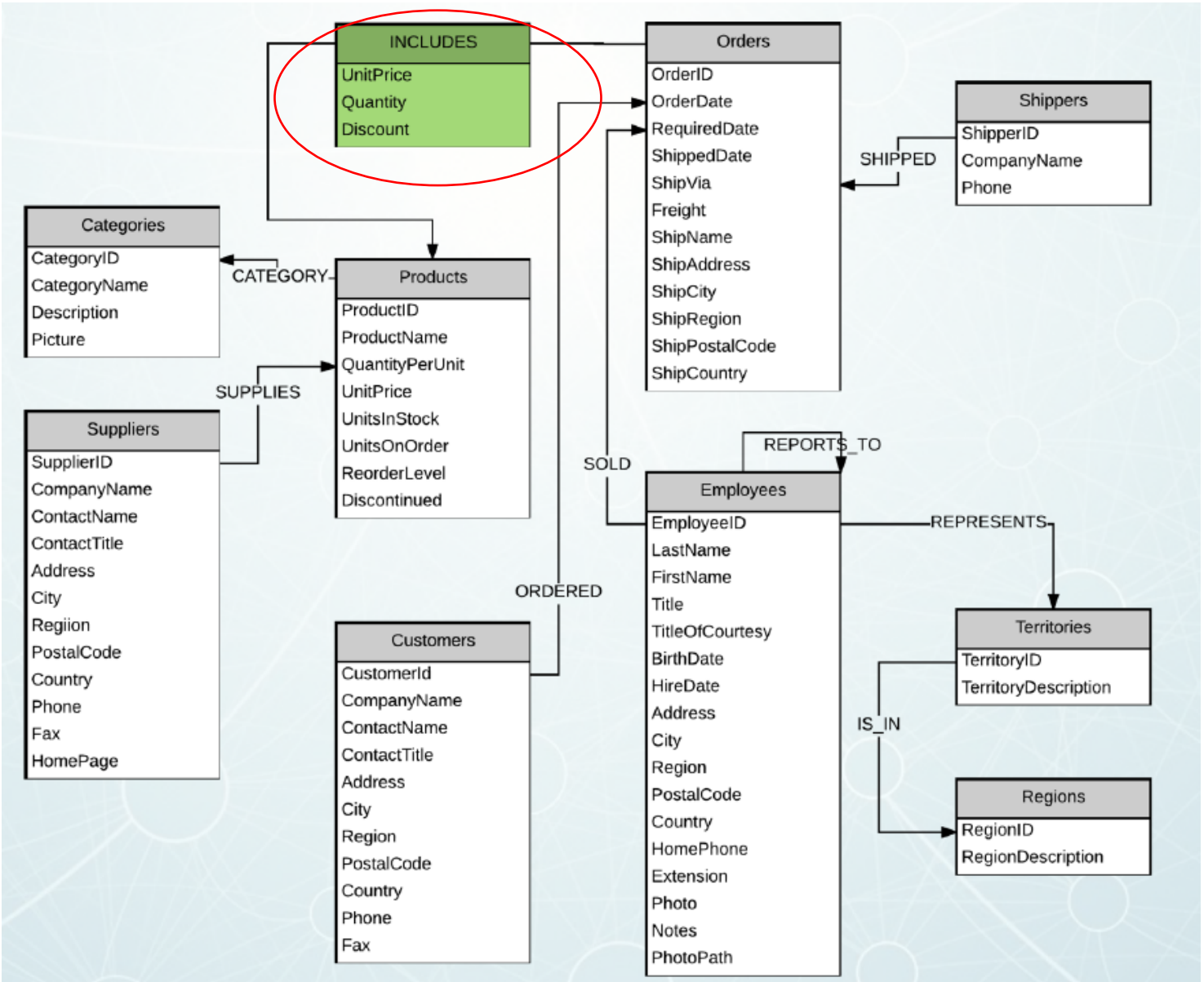
How to go from RDB to GraphDB…

- If join-table has no attributes, make it a relationship…



How to go from RDB to GraphDB

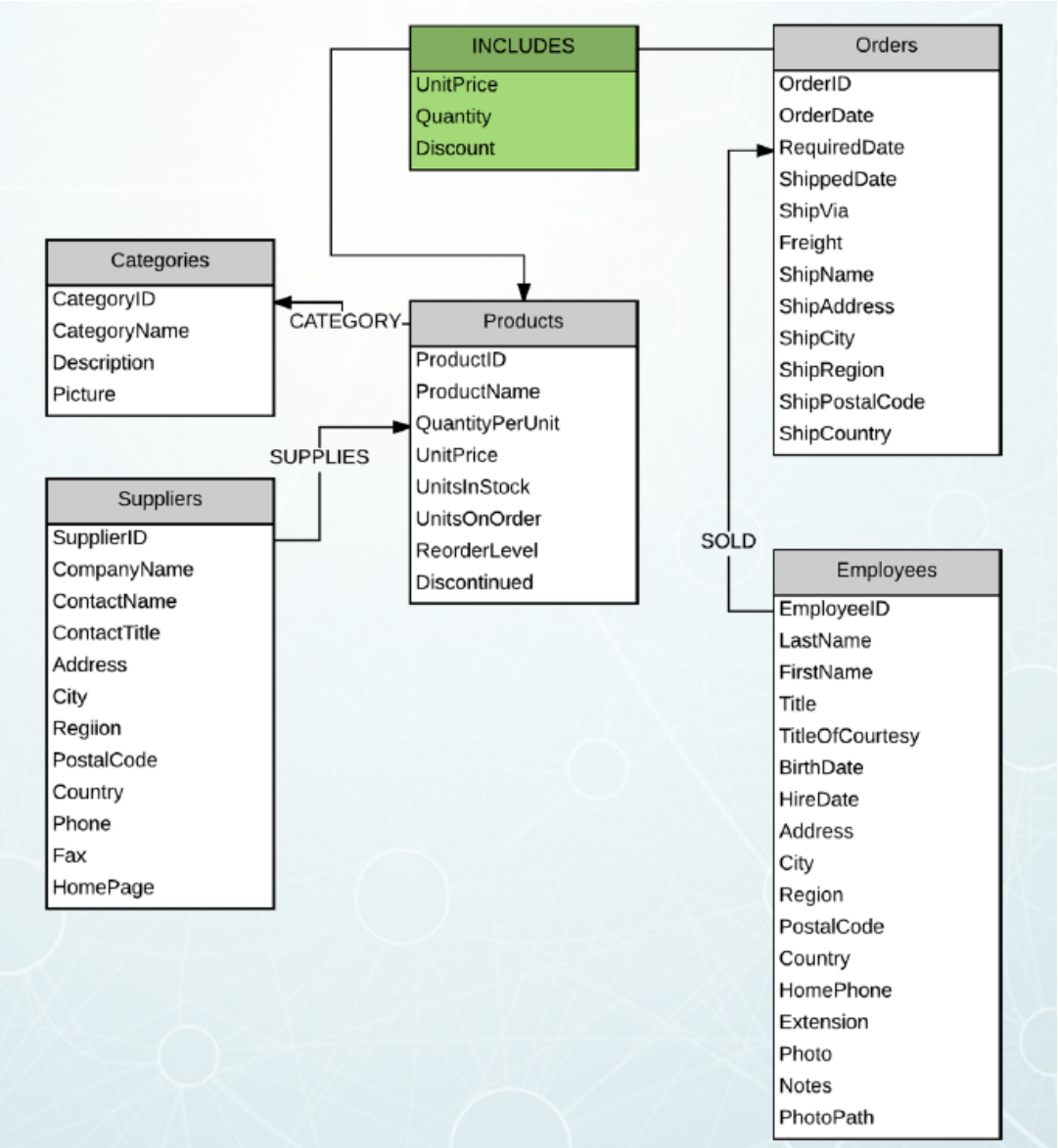
- If there are attributes -> relationship with properties…



How to go from RDB to GraphDB… and then…

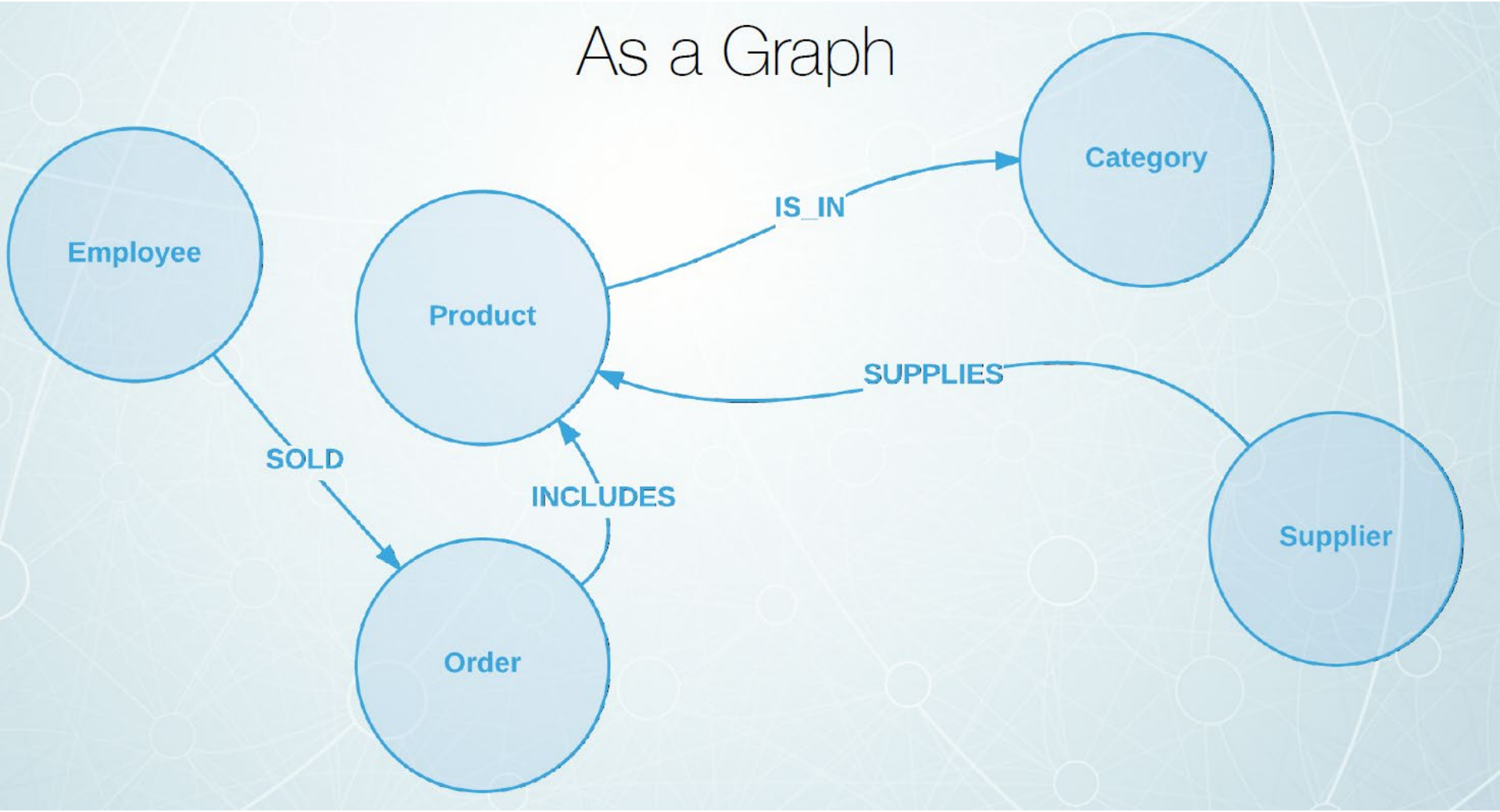
NORTHWESTERN UNIVERSITY

- Consider the subset:



How to go from RDB to GraphDB… and then…

NORTHWESTERN UNIVERSITY

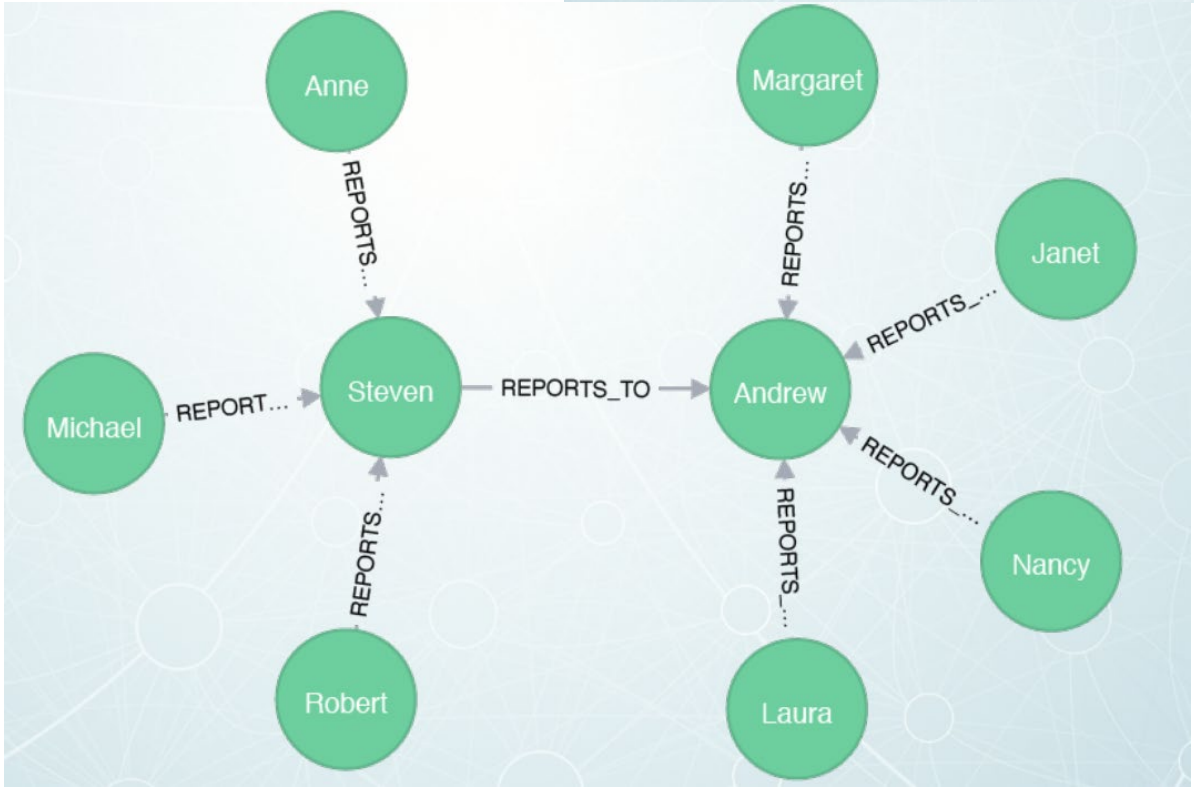


GraphDB… and then…

- **Querying Examples with Neo4J**

Querying Examples:

- Consider:



Querying Examples:

NORTHWESTERN UNIVERSITY

- Q: Who do employees report to?

```
MATCH
  (e:Employee)<-[:REPORTS_TO]-(sub:Employee)
RETURN
   e.employeeID AS managerID,
   e.firstName AS managerName,
   sub.employeeID AS employeeID,
   sub.firstName AS employeeName;
```

Querying Examples:

- Answer:

```
$ MATCH (e:Employee)<-[:REPORTS_TO]-(sub:Employee) RETURN e.employeeID AS...
```

Rows

Code

managerID	managerName	employeeID	employeeName
2	Andrew	1	Nancy
2	Andrew	3	Janet
2	Andrew	4	Margaret
2	Andrew	5	Steven
2	Andrew	8	Laura
5	Steven	9	Anne
5	Steven	6	Michael
5	Steven	7	Robert

Returned 8 rows in 111 ms.

Querying Examples:

- Who is Robert's boss (i.e., Robert reports to)

```
MATCH
 p=(e:Employee)<-[:REPORTS_TO]-(sub:Employee)
WHERE
  sub.firstName = 'Robert'
RETURN
  p
```

Querying Examples:

- What is Robert's hierarchy of reporting

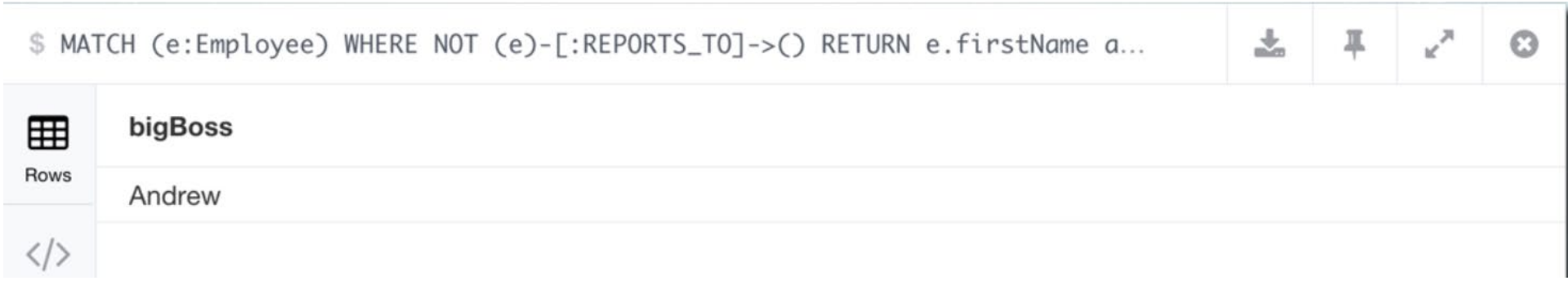
```
MATCH
 p=(e:Employee)<-[:REPORTS_TO*]-(sub:Employee)
WHERE
  sub.firstName = 'Robert'
RETURN
  p
```

Querying Examples:

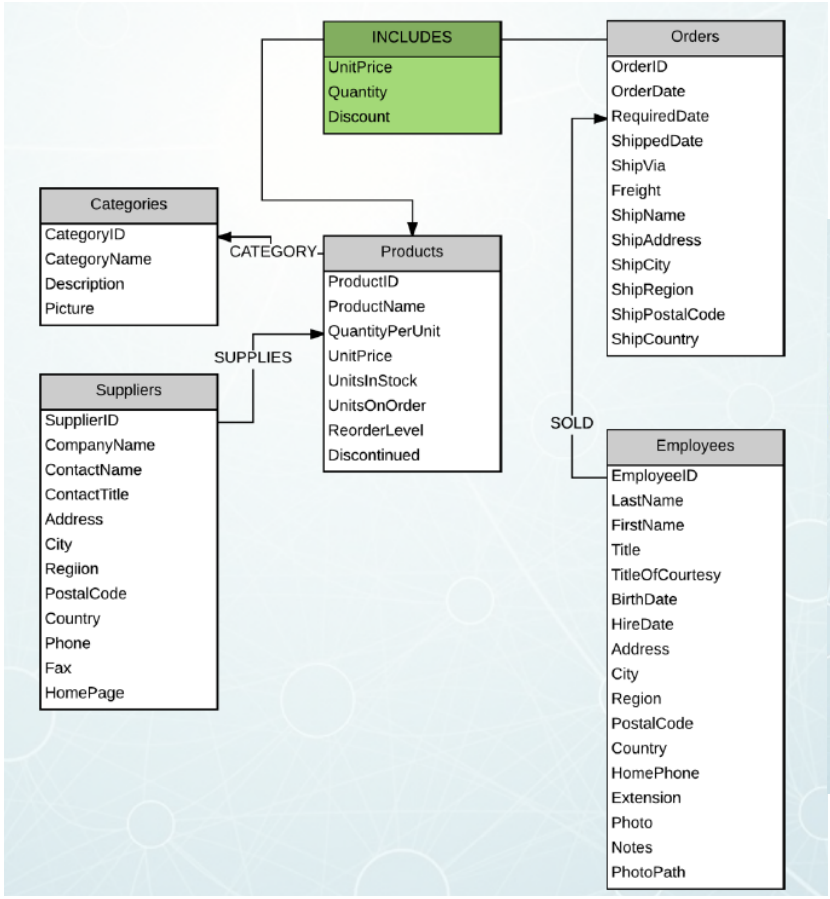
NORTHWESTERN UNIVERSITY

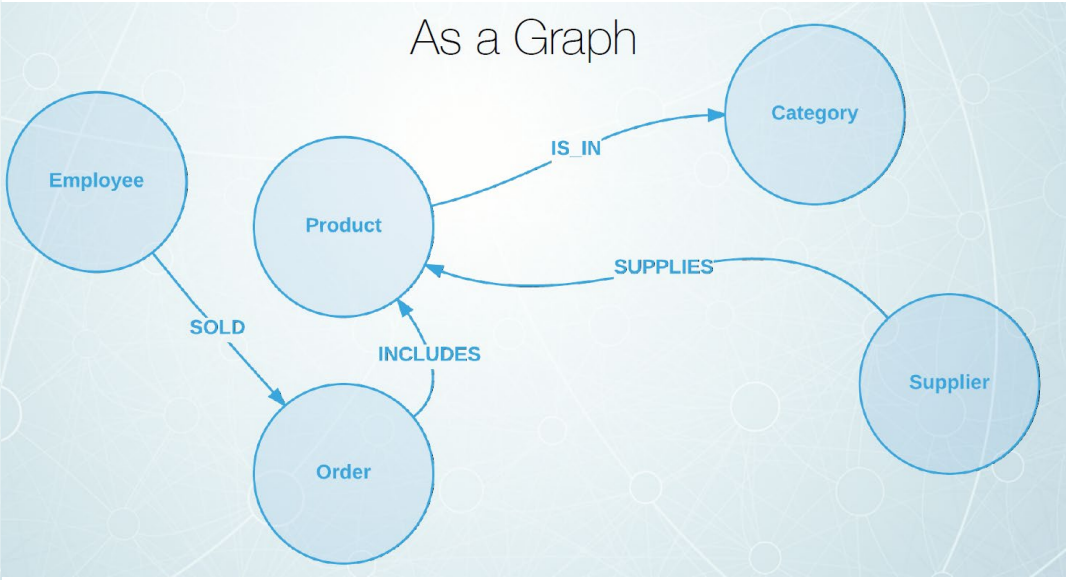
- Who is the ultimate-boss

```
MATCH
  (e:Employee)
WHERE
  NOT (e)-[:REPORTS_TO]->()
RETURN
  e.firstName as bigBoss
```



Back to Northwind… Query Example:





- Which other products have the chocolate sellers sold

Back to Northwind… Query Example:

```
MATCH
  (choc:Product {productName: 'Chocolade'})
   <-[:INCLUDES]-(:Order)<-[:SOLD]-(employee),
  (employee)-[:SOLD]->(o2)-[:INCLUDES]->(other:Product)
RETURN
  employee.firstName,
  other.productName,
  COUNT(DISTINCT o2) as count
ORDER BY
  count DESC
LIMIT 5;
```

```
$ MATCH (choc:Product {productName: 'Chocolade'}) <-[:INCLUDES]-(:Order)<...
```

Rows

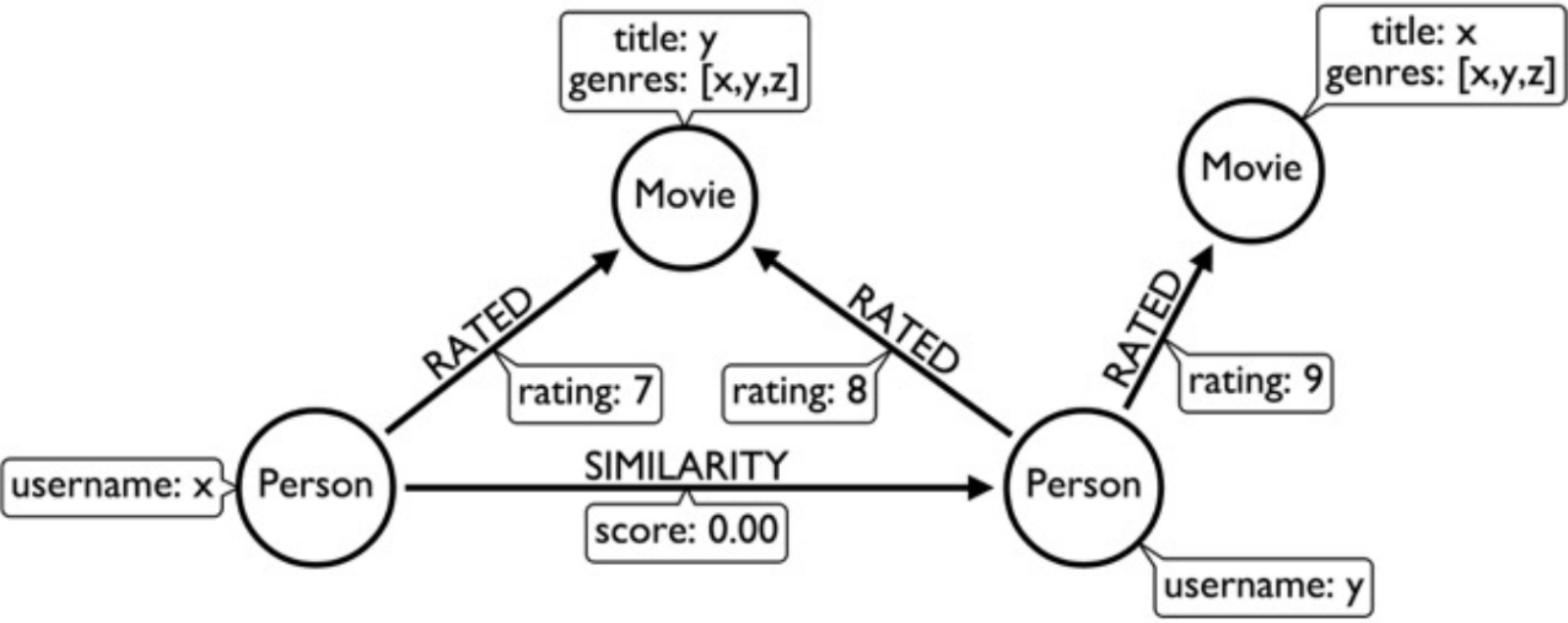
Code

employee.firstName	other.productName	count
Margaret	Gnocchi di nonna Alice	14
Janet	Gumbär Gummibärchen	12
Nancy	Flotemysost	12
Margaret	Pâté chinois	12
Nancy	Camembert Pierrot	11

Returned 5 rows in 319 ms.

Querying Examples:

- Movies GraphDB data model… (“schema”)



Which movies to man between 25 and 34 like

Querying Examples:

- Movies GraphDB

NORTHWESTERN UNIVERSITY

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WHERE u.age = 25 AND u.gender = "M" AND r.stars > 3
RETURN m.title, COUNT(r) AS cnt
ORDER BY cnt DESC
LIMIT 10
```

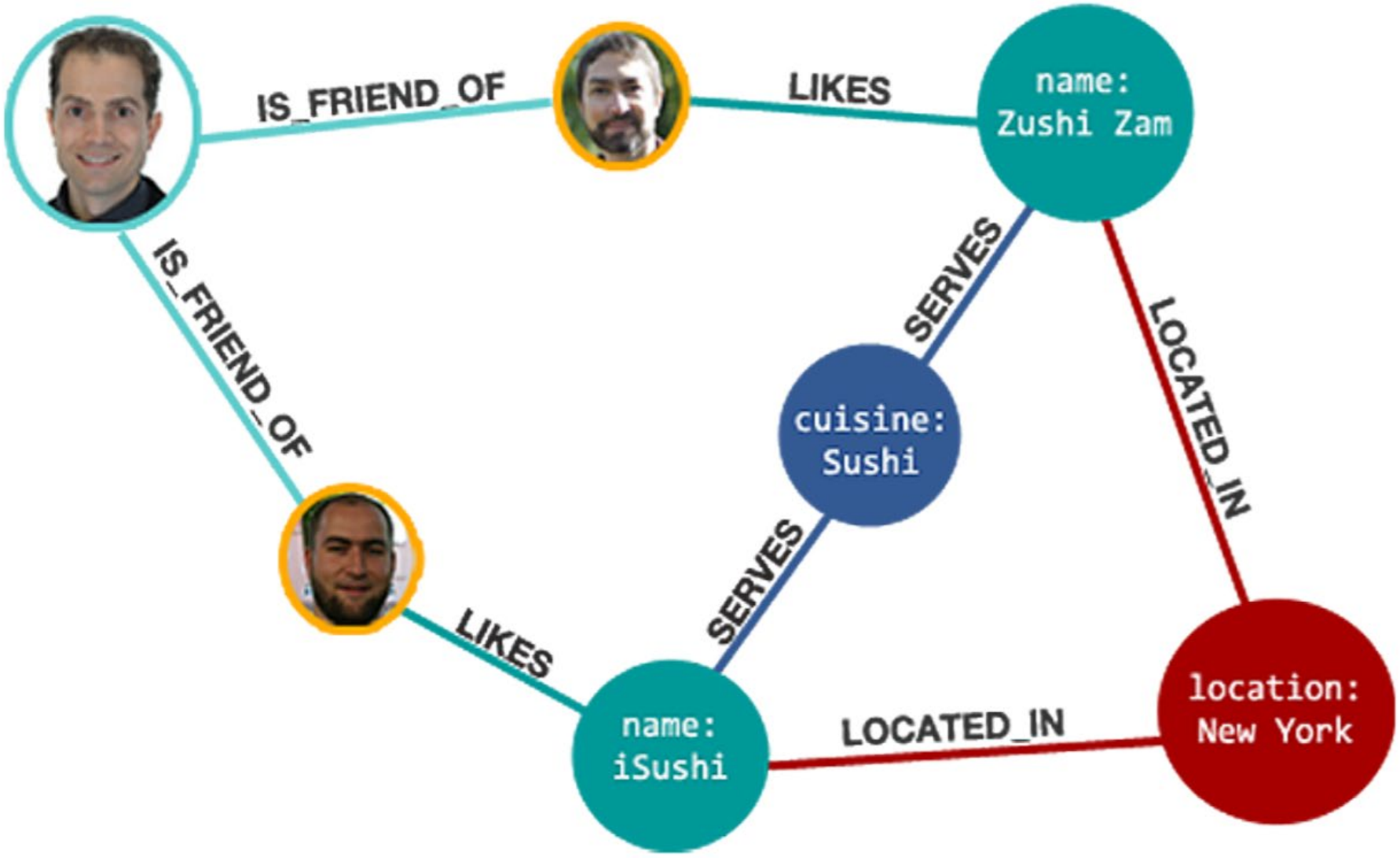
movies.title	cnt
"American Beauty (1999)"	388
"Star Wars: Episode V - The Empire Strikes Back (1980)"	370
"Star Wars: Episode IV - A New Hope (1977)"	366
"Terminator 2: Judgment Day (1991)"	328
"Silence of the Lambs, The (1991)"	326
"Raiders of the Lost Ark (1981)"	323
"Matrix, The (1999)"	320
"Saving Private Ryan (1998)"	313
"Braveheart (1995)"	300
"Star Wars: Episode VI - Return of the Jedi (1983)"	296

Querying Examples

- What are the top 25 movies that I haven't seen, with a same genre as "Toy Story", and given high rankings by people who liked "Toy Story"

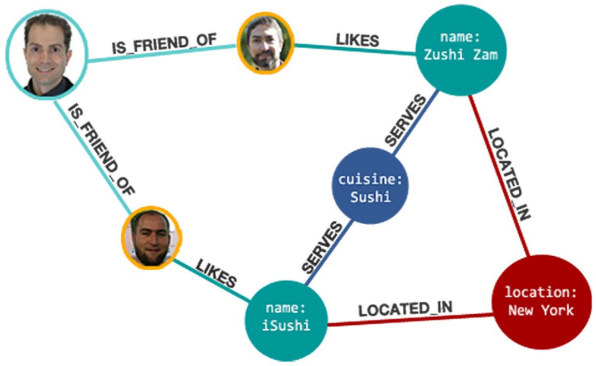
```
MATCH (watched:Movie {title:"Toy Story"})<-[r1:RATED]- () -[r2:RATED]->(unseen:Movie)
WHERE r1.rating > 7 AND r2.rating > 7
AND watched.genres = unseen.genres
AND NOT( (:Person {username:"maxdemarzi"}) -[:RATED|WATCHED]-> (unseen) )
RETURN unseen.title, COUNT(*)
ORDER BY COUNT(*) DESC
LIMIT 25
```

Querying Examples:



Which sushi restaurants in New York does my friend like

Querying Examples:




```
MATCH (me:Person)-[:IS_FRIEND_OF]->(friend),
    (friend)-[:LIKES]->(restaurant),
       (restaurant)-[:LOCATED_IN]->(city:Location),
       (restaurant)-[:SERVES]->(cuisine:Cuisine)

WHERE me.name = 'Philip' AND city.location='New York' AND
      cuisine.cuisine='Sushi'

RETURN restaurant.name
```

NORTHWESTERN
UNIVERSITY

Overview of Use-Cases

[A]CL from Hell

NORTHWESTERN UNIVERSITY

- Customer:
  - leading consumer utility company with tons and tons of users
- Goal:
  - comprehensive access control administration for customers
- Benefits:
  - Flexible and dynamic architecture
  - Exceptional performance
  - Extensible data model supports new applications and features
  - Low cost

[A]CL from Hell

NORTHWESTERN UNIVERSITY

- Customer:
  - leading consumer utility company with tons and tons of users
- Goal:
  - comprehensive access control administration for customers
- Benefits:
  - Flexible and dynamic architecture
  - Exceptional performance
  - Extensible data model supports new applications and features
  - Low cost

- **A Reliable access control administration system for 5 million customers, subscriptions and agreements**
- **Complex dependencies between groups, companies, individuals, accounts, products, subscriptions, services and agreements**
- **Broad and deep graphs (master customers with 1000s of customers, subscriptions & agreements)**

[A]CL from Hell

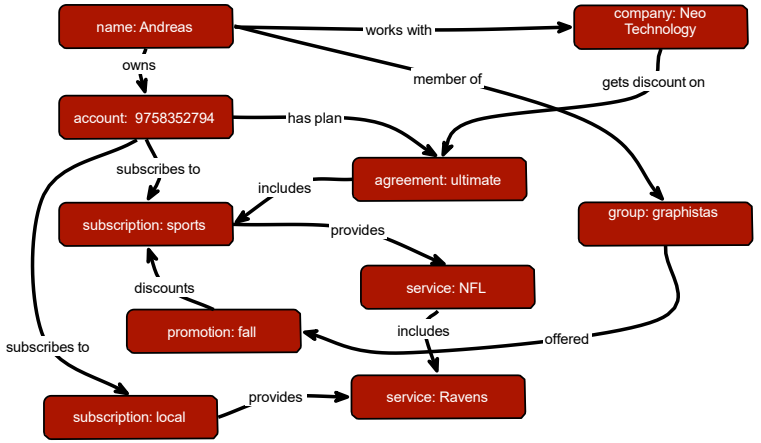
NORTHWESTERN UNIVERSITY

- Customer:
  - leading consumer utility company with tons and tons of users
- Goal:
  - comprehensive access control administration for customers

- Benefits:
  - Flexible and dynamic architecture
  - Exceptional performance
  - Extensible data model supports new applications and features
  - Low cost

- A Reliable access control administration system for

  5 million customers, subscriptions and agreements
- Complex dependencies between groups, companies, individuals, accounts, products, subscriptions, services and agreements
- Broad and deep graphs (master customers with 1000s of customers, subscriptions & agreements)



[B]Timely Recommendations

NORTHWESTERN UNIVERSITY

- Customer:
  - a professional social network
  - 35 millions users, adding 30,000+ each day
- Goal: up-to-date recommendations

- Benefits:
  - Scalable solution with real-time end-user experience
  - Low maintenance and reliable architecture
  - 8-week implementation

[B]Timely Recommendations

- Customer:
  - a professional social network
  - 35 millions users, adding 30,000+ each day
- Goal: up-to-date recommendations

- Scalable solution with real-time end-user experience
- Low maintenance and reliable architecture
- 8-week implementation

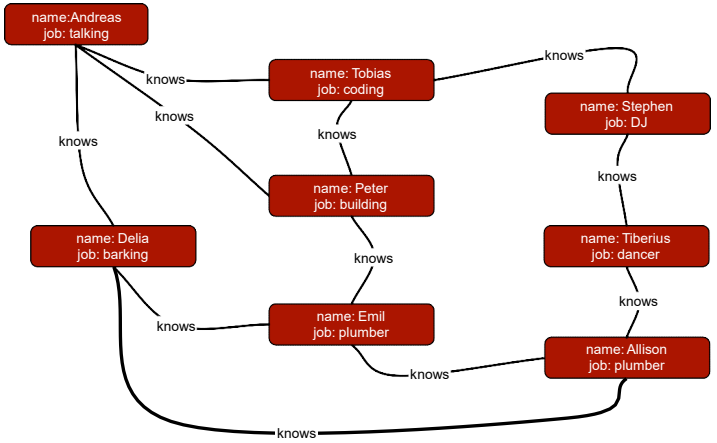
- **Problem:**
  - **Real-time recommendation imperative to attract new users and maintain positive user retention**
  - **Clustered MySQL solution not scalable or fast enough to support real-time requirements**
- **Upgrade from running a batch job**
  - **initial hour-long batch job**
  - **but then success happened, and it became a day**
  - **then two days**
- **With Neo4j, real time recommendations**

[B]Timely Recommendations

NORTHWESTERN UNIVERSITY

- Customer:
  - a professional social network
  - 35 millions users, adding 30,000+ each day
- Goal: up-to-date recommendations
- Problem:
  - Real-time recommendation imperative to attract new users and maintain positive user retention
  - Clustered MySQL solution not scalable or fast enough to support real-time requirements
- Upgrade from running a batch job
  - initial hour-long batch job
  - but then success happened, and it became a day
  - then two days
- With Neo4j, real time recommendations

- Scalable solution with real-time end-user experience
- Low maintenance and reliable architecture
- 8-week implementation



[C] Collaboration on Global Scale

NORTHWESTERN UNIVERSITY

- Customer: a worldwide software leader
  - highly collaborative end-users

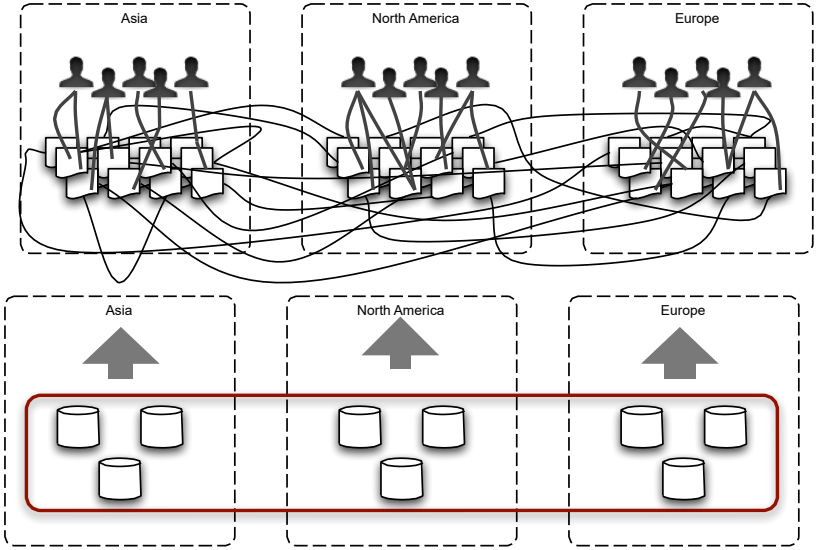
Goal: offer an online platform for global collaboration

- Highly flexible data analysis
- Sub-second results for large, densely-connected data
- User experience - competitive advantage

- Massive amounts of data tied to members, user groups, member content, etc. all interconnected

- Infer collaborative relationships through user-generated content

Worldwide Availability



Industry: Communications
Use case: Master Data Management
San Jose, CA

Cisco HMP

NORTHWESTERN UNIVERSITY

Background

- One of the world's largest communications equipment manufacturers
- #91 Global 2000. $44B in annual sales.
- Needed a system that could accommodate its master data hierarchies in a performant way
- HMP is a Master Data Management system at whose heart is Neo4j. Data access services available 24x7 to applications companywide

Business problem

- Sales compensation system had become unable to meet Cisco's needs
- Existing Oracle RAC system had reached its limits:
  - Insufficient flexibility for handling complex organizational hierarchies and mappings
  - "Real-time" queries were taking > 1 minute!
- Business-critical "P1" system needs to be continually available, with zero downtime

Solution & Benefits

- Cisco created a new system: the Hierarchy Management Platform (HMP)
- Allows Cisco to manage master data centrally, and centralize data access and business rules
- Neo4j provided "Minutes to Milliseconds" performance over Oracle RAC, serving master data in real time
- The graph database model provided exactly the flexibility needed to support Cisco's business rules
- HMP so successful that it has expanded to include product hierarchy

Industry: Communications
Use case: Social gaming
Frankfurt, Germany

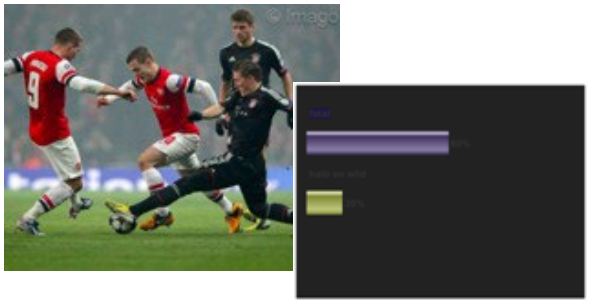
Deutsche Telecom

Background

- Europe's largest communications company
- Provider of mobile & land telephone lines to consumers and businesses, as well as internet services, television, and other services

> 236,000	50	> 58 bn. €
Employees worldwide in 2011	Countries	Revenue in 2011

Interactive Television



Business problem

- The Fanorakel application allows fans to have an interactive experience while watching sports
- Fans can vote for referee decisions and interact with other fans watching the game
- Highly connected dataset with real-time updates
- Queries need to be served real-time on rapidly changing data
- One technical challenge is to handle the very high spikes of activity during popular games

Solution & Benefits

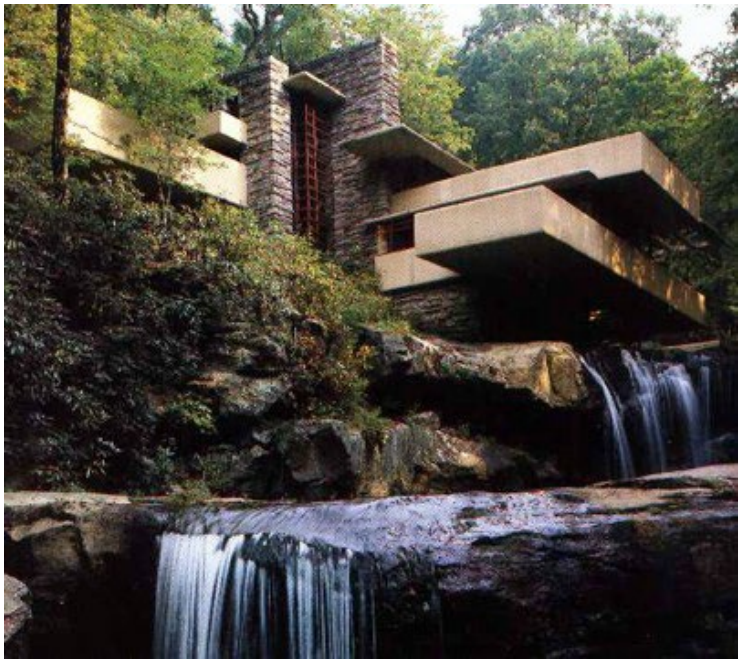
- Interactive, social offering gives fans a way to experience the game more closely
- Increased customer stickiness for Deutsche Telekom
- A completely new channel for reaching customers with information, promotions, and ads
- Clear competitive advantage

SUMMARY:

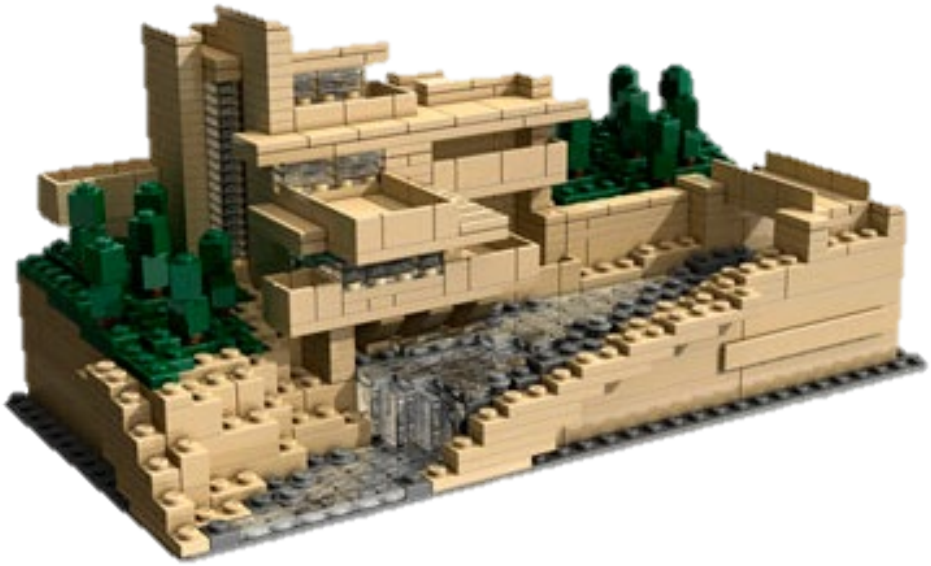
Model

NORTHWESTERN UNIVERSITY

Real World



Model



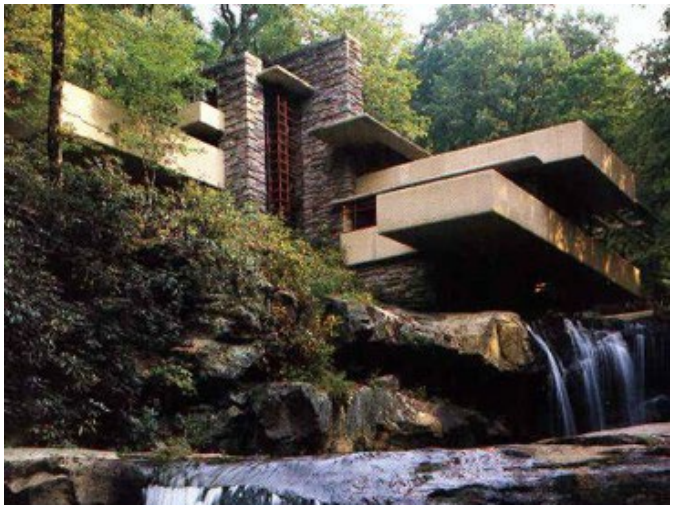
Model mis-match

NORTHWESTERN UNIVERSITY

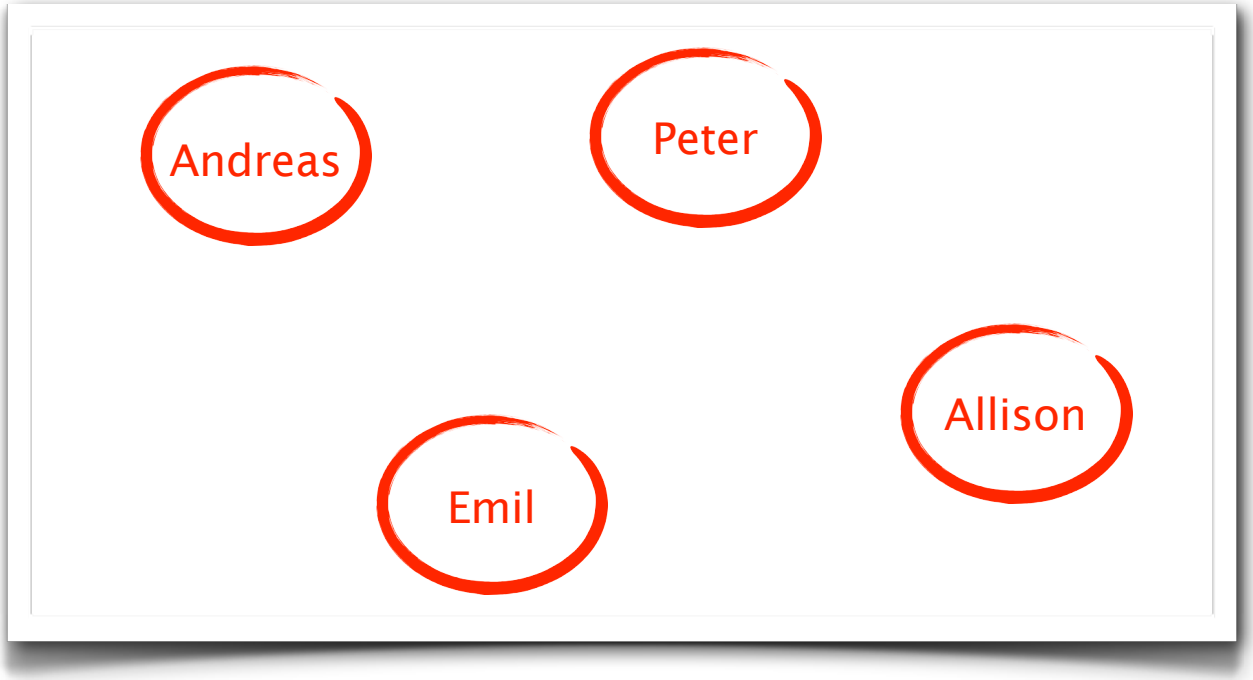
Application Model

Database Model

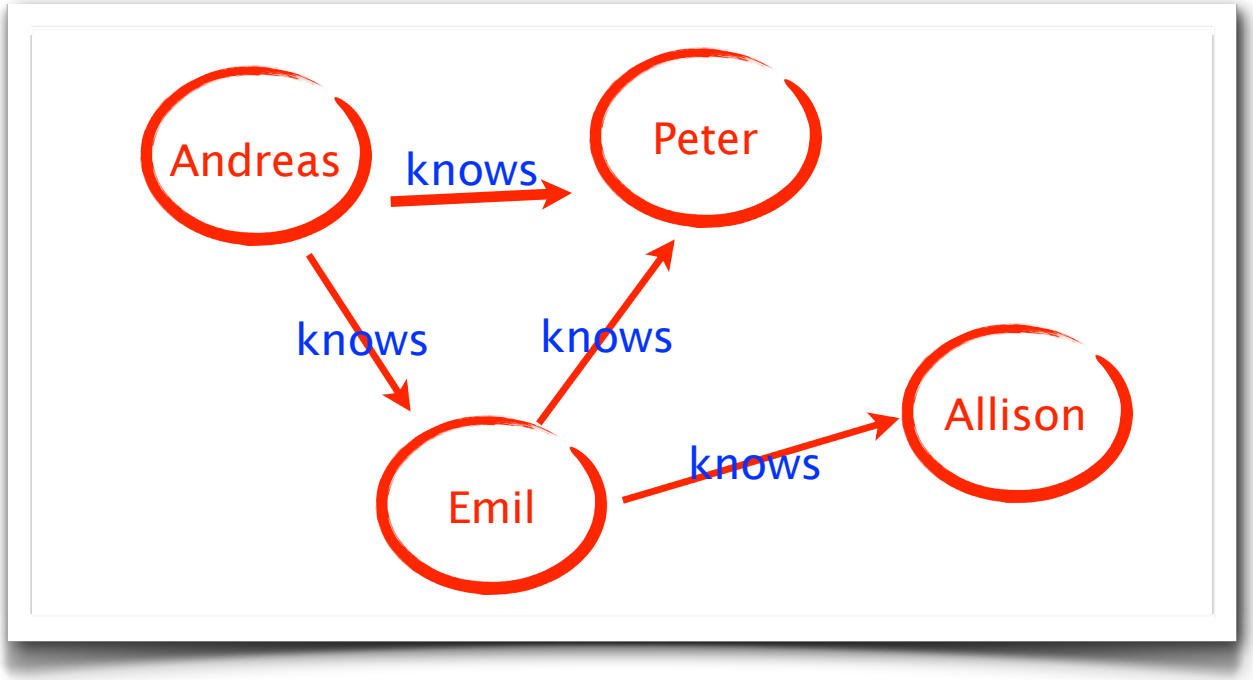
Trinity of models



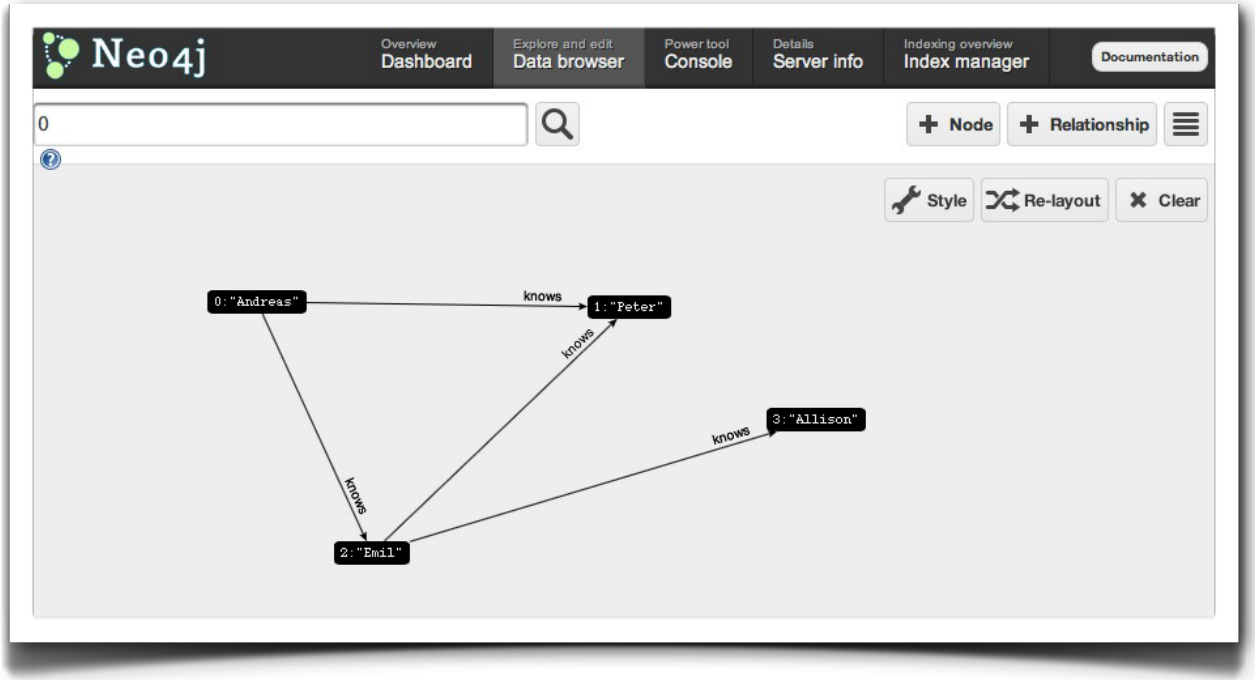
GraphDB: Whiteboard --> Data



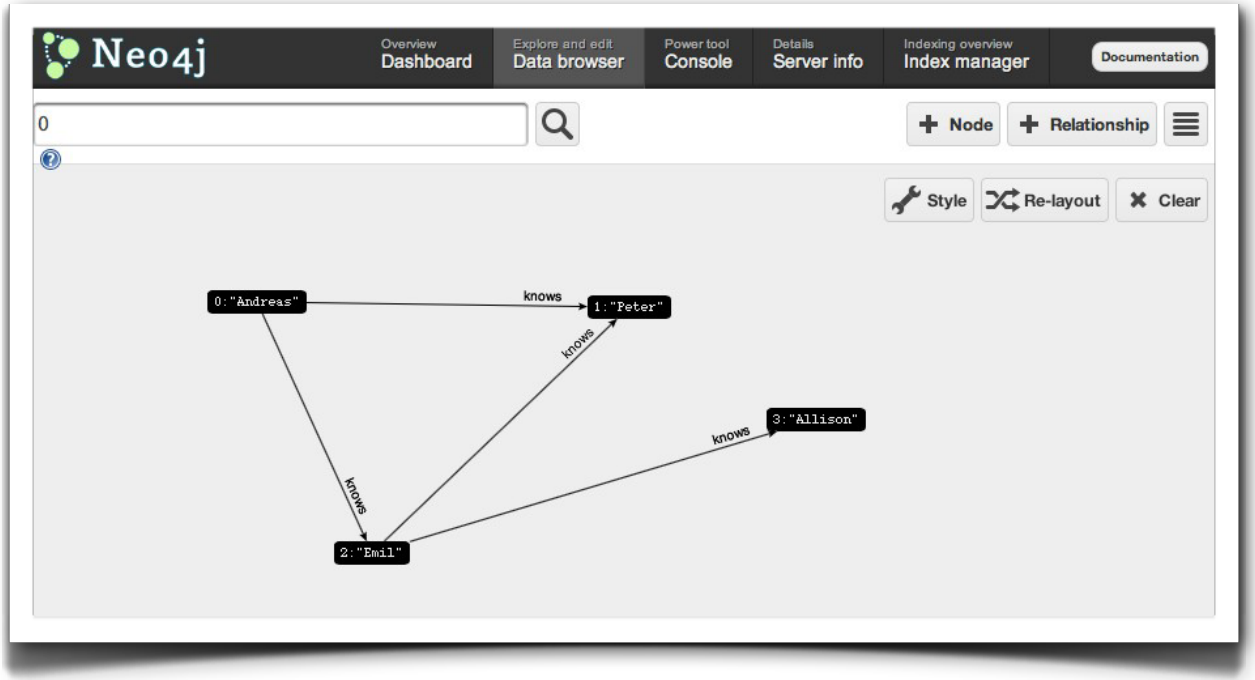
GraphDB: Whiteboard --> Data



GraphDB: Whiteboard --> Data



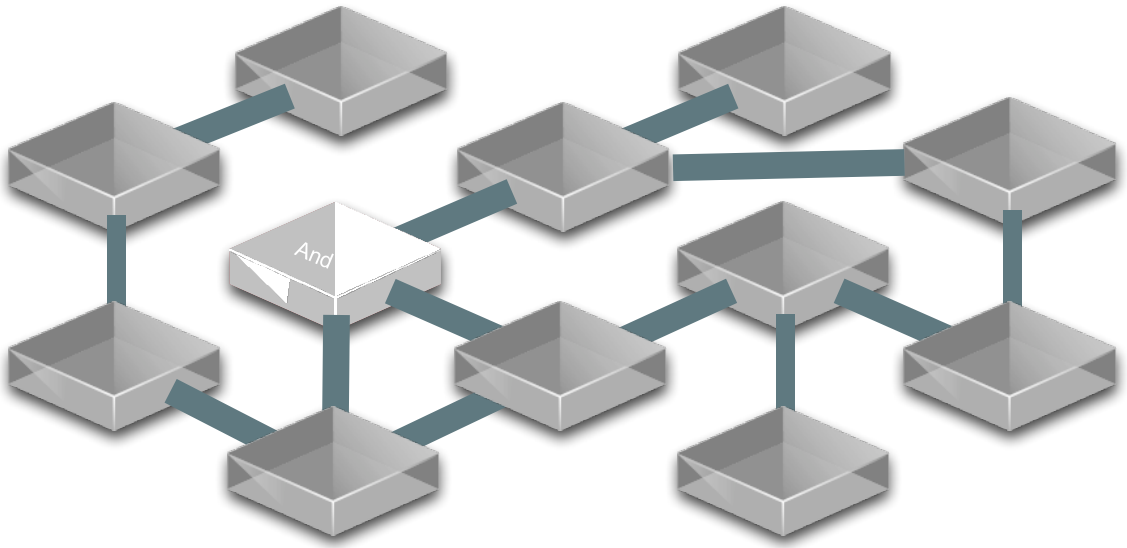
Whiteboard --> Data AND Query




```
// Cypher query - friend of a friend
start n=node(0)
match (n)-->()-[:KNOWS|LIKES]->(foaf)
WHERE NOT((n)[:KNOWS]-->(foaf))
return foaf
```

Processing: You traverse the graph

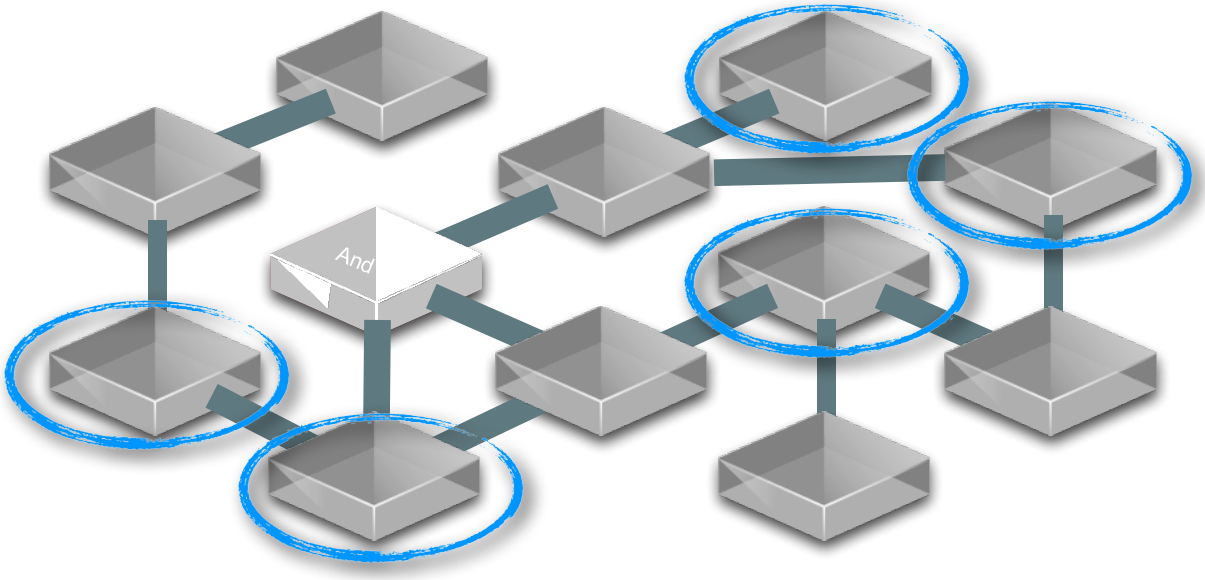
```
// lookup starting point in an index
START n=node:People(name = 'Andreas')
```



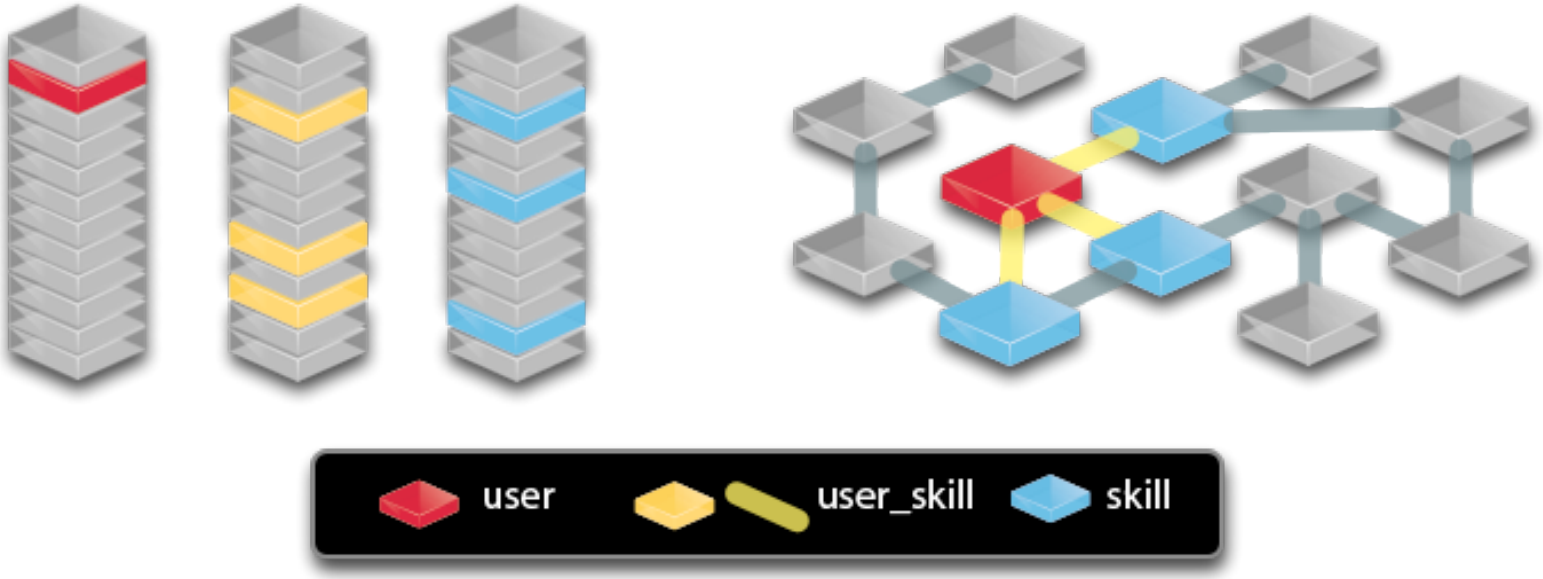
Processing: You traverse the graph

- // lookup starting point in an index
- START n=node:People(name =

```
// then traverse to find results
START me=node:People(name = 'Andreas'
MATCH (me)-[:FRIEND]-(friend)-[:FRIEND]-(friend2)
RETURN friend2
```



So, in summary:




```
START user = node(1)
MATCH user -[user_skill]-> skill
RETURN skill, user_skill
```

```
SELECT skills.*, user_skill.*
FROM users
JOIN user_skill ON users.id = user_skill.user_id
JOIN skills ON user_skill.skill_id = skill.id WHERE users.id = 1
```

Well, a little more “summary”

- When to use which

A little more summary… and caution…

NORTHWESTERN UNIVERSITY

- RDMBS: relational databases, the powerhouse of software applications since the 1980s, work well when your data is predictable and fits well into tables, columns, rows, and wherever queries are not very join-intensive.
- Another type of database, optimized for connected data: the graph database. Graph databases are compelling because they enable companies to make sense of the masses of connected data that exist today.
- **Walking, not joining**
  - Graph databases are very adept at working with not just single points of information, but also evolving relationship networks.
  - Particularly well when the relationships inside your data are important and your queries depend on exploring and exploiting them.

A little more summary… and caution…

NORTHWESTERN UNIVERSITY

- GraphDB: relationship information is a first-class entity.
  - Also: flexible for adding new nodes and relationships without compromising existing network or expensively migrating your data. Highly efficient when it comes to query performance, even for deep and complex queries
- Connections between nodes directly link in such a way that relating data becomes a simple matter of following connections.
  - Avoids join index lookup performance problem by specifying connections at insert time, so that the data graph can be walked rather than calculated at query time.
- Index-free adjacency allows queries to traverse millions of nodes per second
  - Response times several orders of magnitude faster than with relational databases for connected queries (e.g., friend-of-friend/shortest path).

A little more summary… and caution…

NORTHWESTERN UNIVERSITY

- So does it make sense for CIOs to use both relational databases and graph databases?
  - should they standardize across the enterprise on one or the other?
  - Today it makes pragmatic sense to use both. Each model has their pros and cons; as the enterprise IT user typically has a wide set of problems it needs to solve, there is no single database or database model that is best at everything.
- current applications for graph databases include fraud detection, real-time recommendation engines, master data management, network and IT operations, and identity and access management
- No formal approach to delineating which is best. How can you tell when the situation is right for graph databases over RDBMS?

A little more summary… and caution…

NORTHWESTERN UNIVERSITY

- What are the biggest drawbacks to using graph databases?
  - The number of graph databases available is growing (good news for the developer)
  - the technology is still fairly new compared to relational software, which has now existed for a (few) full generation(s). It takes time to build a solid database market after all, regardless of data model.
- Transactions, recovery and durability are features that you would take for granted when a database may not be working as expected—or worse, not be present at all.
  - Many graph database implementations are still young, it may be a good idea to first verify that core features work as advertised.
- Some graph databases only offer the graph model, but the underlying implementation is backed by a traditional, relational or other type of NoSQL database.
  - Can impact runtime behavior as queries may get translated into joins. Be clear about what you are getting.

Graphs? Not really, but sort of…

NORTHWESTERN UNIVERSITY

- The Semantic Web…

The Web (common perception)

NORTHWESTERN UNIVERSITY

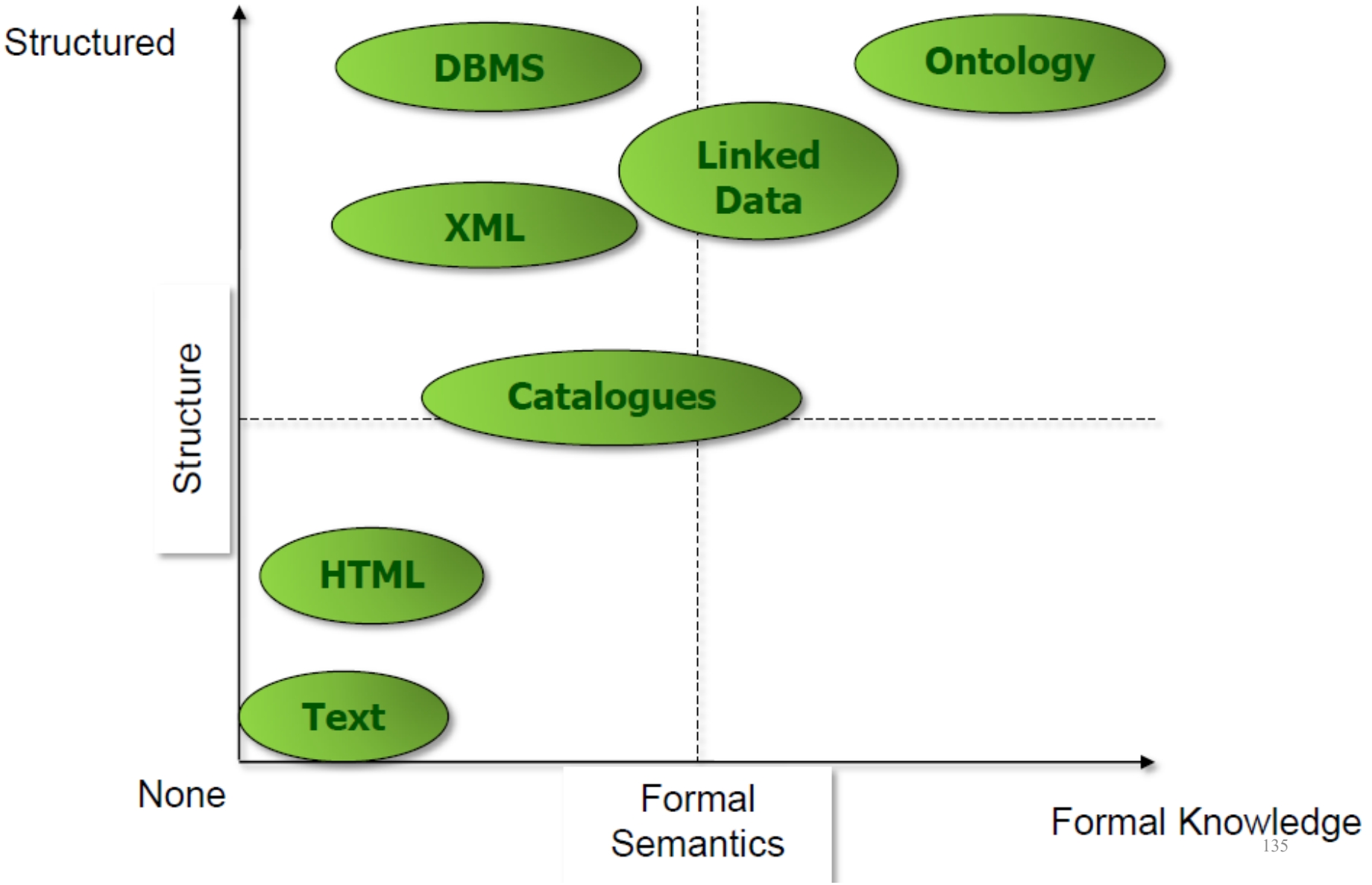
- *Target consumers*: humans
  - – web 2.0 mashups provide *some* improvement
  - – Rules about the *structure* and *visualisation* of information, but not about its intended meaning
  - – Intelligent agents can't easily use the information
- *Granularity*: document
  - – One giant distributed *filesystem* of documents
  - – One *document* can link to other documents
- *Integration & reuse*: very limited
  - – Cannot be easily automated
  - – Web 2.0 mashups provide *some* improvement

Web: limitations

NORTHWESTERN UNIVERSITY

- Finding information
- Data granularity
- Resource identification
- Data aggregation & reuse
- Data integration
  - Recall: ETL
- Inference of new information
  - Recall: Analytics queries in MDX…

Types of Data…



Smarter Web

NORTHWESTERN UNIVERSITY

- *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."* (Tim Berners-Lee, 2001)
-
- *"PricewaterhouseCoopers believes a Web of data will develop that fully augments the document Web of today. You'll be able to find and take* ***pieces of data*** *sets* ***from different places****, aggregate them without warehousing, and* ***analyze them in a more straightforward, powerful way*** *than you can now." (PWC, May 2009)*

The Semantic Web

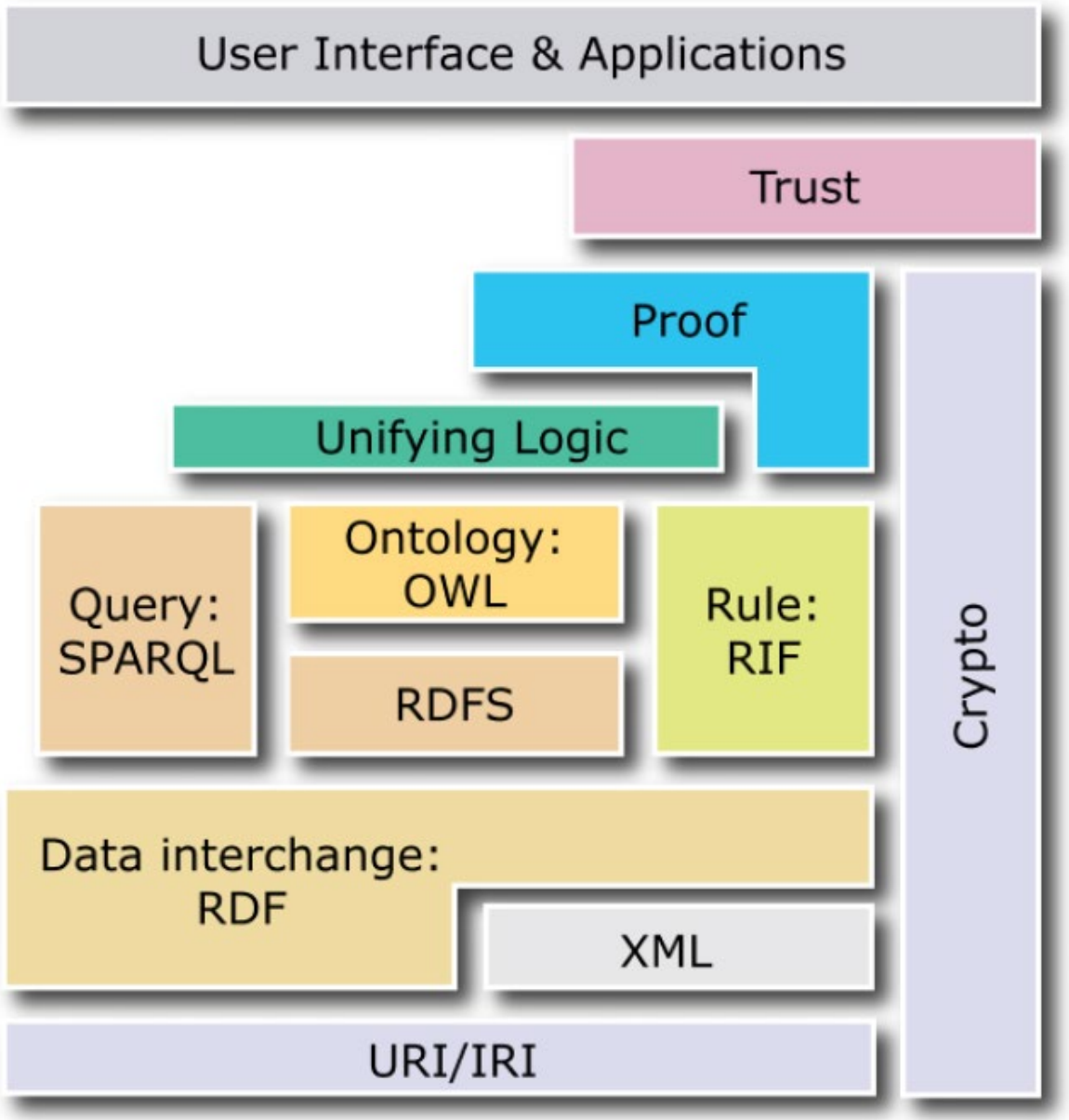
NORTHWESTERN UNIVERSITY

- *Target consumers*: intelligent agents
  - – Explicit specification of the *intended meaning* information
  - – Intelligent agents can make use the information
- • *Granularity*: resource/fact
  - – One giant distributed *database* of facts about resources
  - – One *resource* can be linked (related) to other resources
- ***Integration & reuse***: easier
  - – Resources have unique identifiers
  - – With explicit semantics transformation & integration can be automated

W3C: Semantic Web

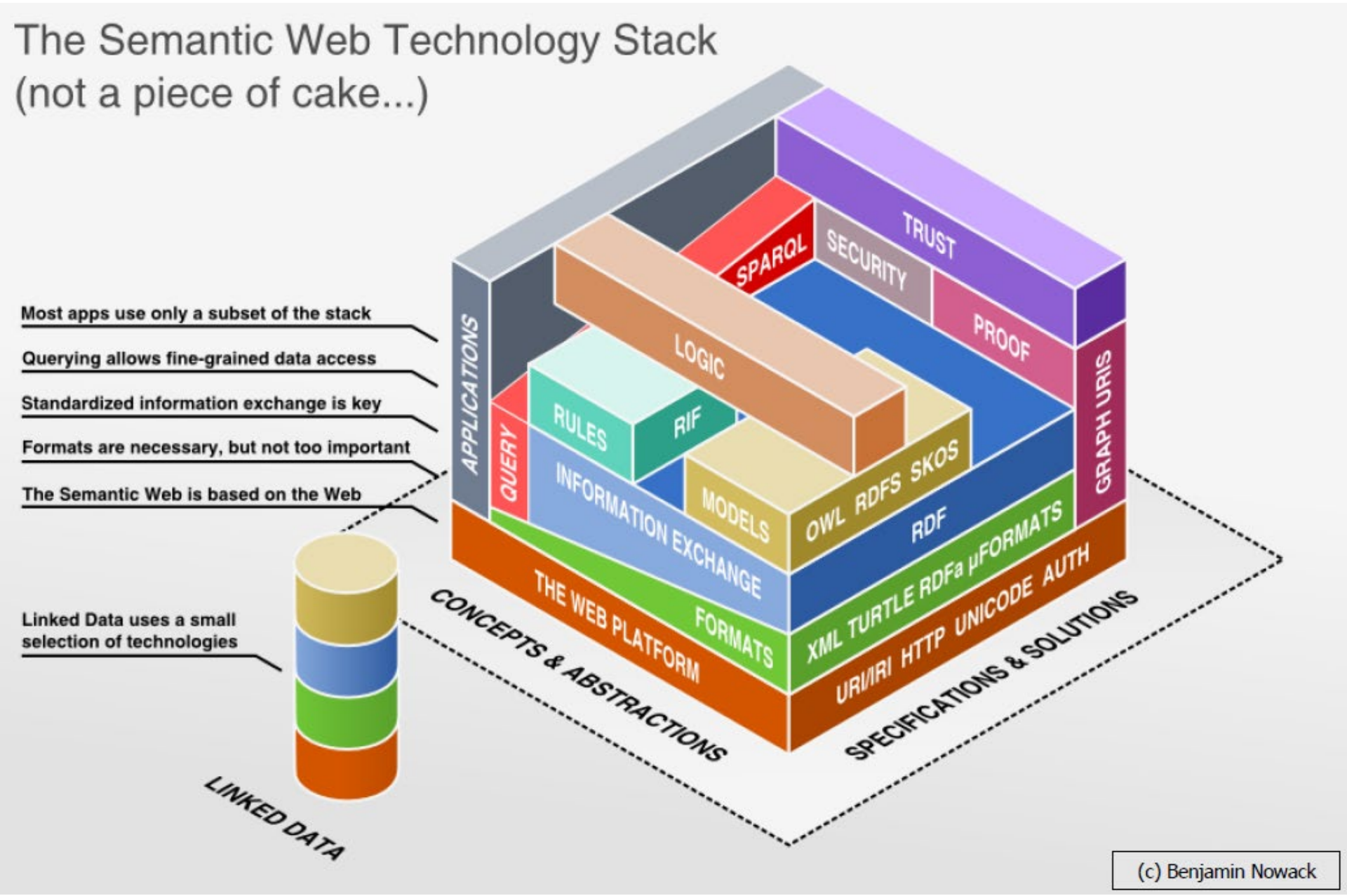
- Extend principles of the Web from documents to data
- Data should be accessed using the general Web architecture (e.g., URI-s, protocols, …)
- Data should be related to one another just as documents are already
- Creation of a common framework that allows:
  - – Data to be shared and reused across applications
  - – Data to be processed automatically
  - – New relationships between pieces of data to be inferred

Semantic Web Layers…



Semantic Web: more details…

NORTHWESTERN UNIVERSITY



Semantic Web: Ontologies

NORTHWESTERN UNIVERSITY

- A *formal* specification that provides sharable and reusable knowledge representation
  - – Examples:
  - taxonomies, thesauri, topic maps, formal ontologies
- An ontology specification includes
  - – Description of the *concepts* in some domain and their properties
  - – Description of the possible *relationships* between concepts and the *constraints* on how the relationships can be used
  - – Sometimes, the *individuals (*members of concepts)

Semantic Web: Ontology Dimensions

NORTHWESTERN

	dimension	examples
Semantic	**Degree of structure and formality**	• Informal (specified in natural language) • taxonomy or topic hierarchy • very formal - unambiguous description of terms and axioms
	Expressiveness of the representation language	• different logic formalisms have different expressivity (and computational complexity)
	granularity	• simple taxonomies and hierarchies • detailed property descriptions, rules and restrictions
Pragmatic	**Intended use**	• data integration (of disparate datasources) • represent a natural language vocabulary (lexical ontology) • categorization and classification
	Role of automated reasoning	• is inference of new knowledge required? • simple reasoning (class/subclass transitivity inference) vs. complex reasoning (classification, theorem proving)
	Descriptive vs. prescriptive	• descriptive – less strict characterization, • prescriptive – strict characterization
	Design methodology	• bottom-up vs. top-down
	governance	• are there legal and regulatory implications • is provenance required?

Ontologies: Example

NORTHWESTERN UNIVERSITY

```
class Person

class Woman
    subClassOf          #Person

class Man
    subClassOf          #Person
    complementOf        #Woman
```

```
property hasParent
    domain              #Person
    range               #Person
    maxCardinality      2

property hasChild
    inverseOf           #hasParent

property hasSpouce
    domain              #Person
    range               #Person
    maxCardinality      1
    symmetric
```

```
individual John
    instanceOf      #Man

individual Mary
    instanceOf      #Woman
    hasSpouce       #John

individual Jane
    instance Of     #Woman
    hasParent       #John
    hasParent       #Mary
```

Hmmm:

- Properties
- relationships
- …

Resource Description Framework (RDF)

NORTHWESTERN UNIVERSITY

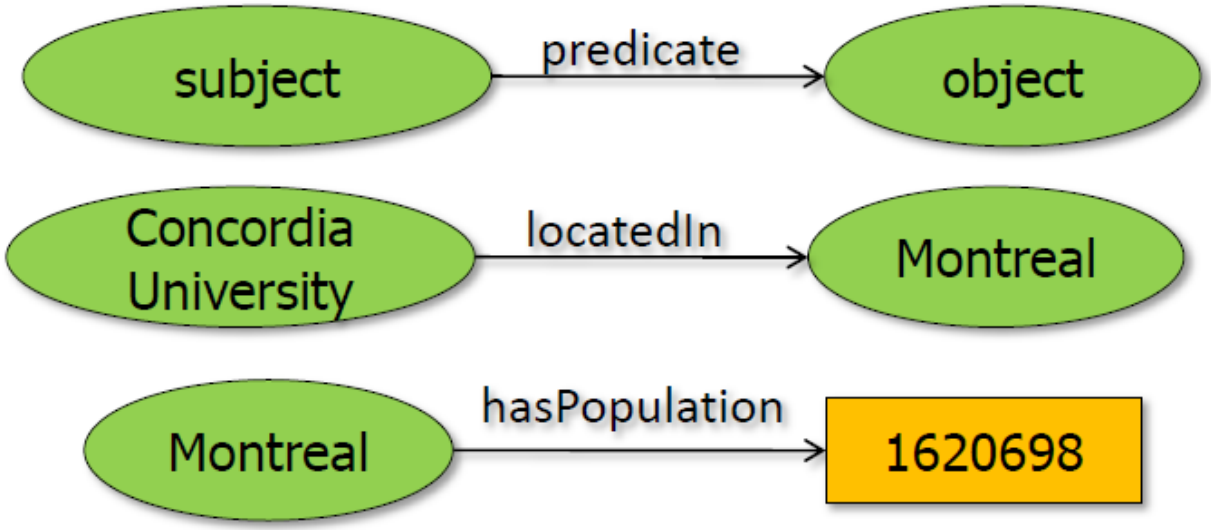
- A simple data model for
  - – Formally describing the *semantics* of information in a machine accessible way
  - – representing meta-data (data about data)

- A set of representation syntaxes
  - – XML (standard) but also N3, Turtle, …
- Building blocks
  - – *Resources* (with unique identifiers)
  - – *Literals*
  - – Named *relations* between pairs of resources (or a resource and a literal)

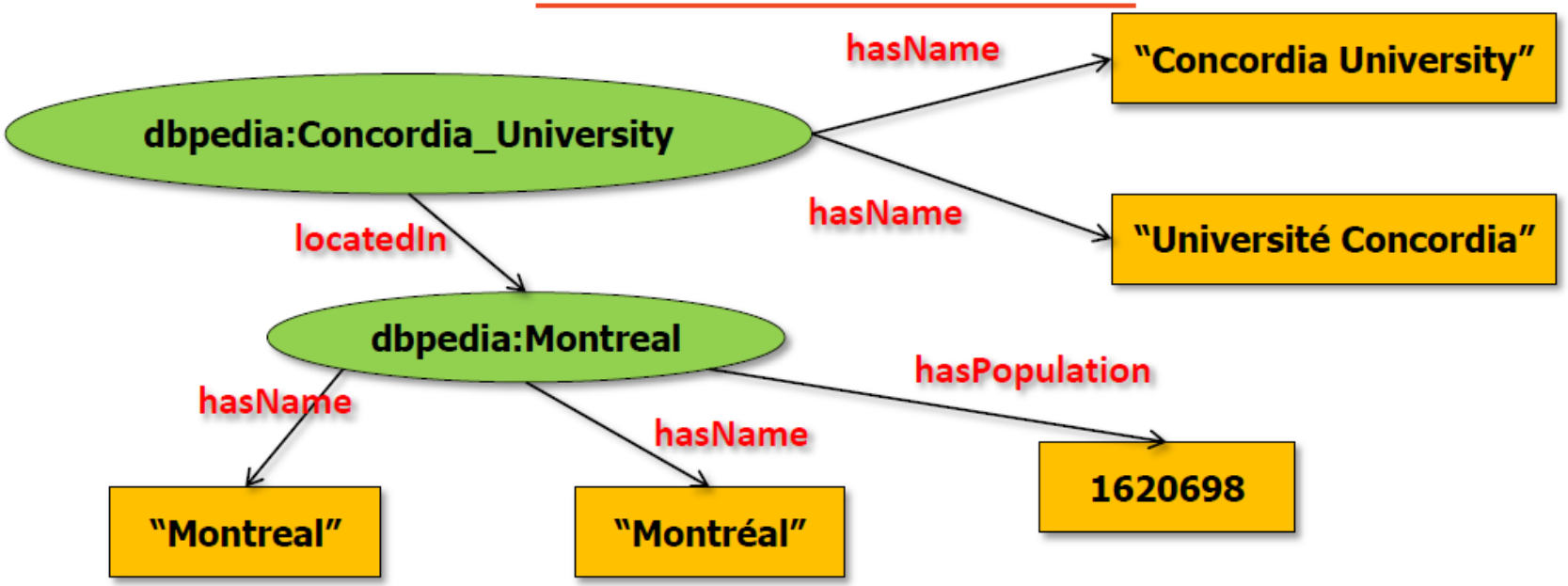
RDF

NORTHWESTERN UNIVERSITY

- Everything is a triple
  - **Subject** (resource), **Predicate** (relation), **Object** (resource or literal)
- The RDF graph is a collection of triples



RDF Example



Subject	Predicate	Object
http://dbpedia.org/resource/Montreal	hasName	"Montreal"
http://dbpedia.org/resource/Montreal	hasPopulation	1620698
http://dbpedia.org/resource/Montreal	hasName	"Montréal"
http://dbpedia.org/resource/Concordia_University	**locatedIn**	**http://dbpedia.org/resource/Montreal**
http://dbpedia.org/resource/Concordia_University	hasName	"Concordia University"
http://dbpedia.org/resource/Concordia_University	hasName	"Université Concordia"

RDF Features

NORTHWESTERN UNIVERSITY

- Simple but expressive data model
- Global identifiers of all resources (URIs)
  - – Reduces ambiguity
- Easier incremental data integration
  - – Can handle incomplete information (Open World Assumption)
- Schema agility
- ***Graph structure***
  - – Suitable for a large class of tasks
  - – Data merging is easier

SPARQL

- SQL-like query language for RDF data
- Simple protocol for querying remote databases over HTTP
- Query types
  - – ***select*** – projections of variables and expressions
  - – ***construct*** – create triples (or graphs)
  - – ***ask*** – whether a query returns results (result is true/false)
  - – ***describe*** – describe resources in the graph

```
SELECT ?var1 ?var2
WHERE {
  triple-pattern1 .
  triple-pattern2 .
  {{triple-pattern3} UNION {triple-pattern4}}
  OPTIONAL {triple-pattern5}
  FILTER (filter-expr)
}
ORDER BY DESC (?var1)
LIMIT 100
```

SPARQL

NORTHWESTERN
UNIVERSITY

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbp-ont: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?university ?students
WHERE {
   ?university rdf:type dbpedia:University ;
               dbp-ont:numberOfStudents ?students ;
               dbp-ont:city dbpedia:Montreal .
  FILTER (?students > 5000)
}
ORDER BY DESC (?students)
```

FactForge

SPARQL Query

Results for PREFIX rdf:<http://www.w3... (6)

View as Exhibit Downlo

university	students
dbpedia:Con_U	43944
dbpedia:HEC_Montreal	12000
dbpedia:Collège_Ahuntsic	10100
dbpedia:John_Molson_School_of_Business	8026
dbpedia:CEGEP_Vanier	6100
dbpedia:Cégep_du_Vieux_Montréal	6100

ASIDE: Semantic Similarity

- What is it measuring?
- Concept pairs
  - Assign a numeric value that quantifies how similar or related two concepts are
- Not words
- Must know concept underlying a word form
  - Cold may be temperature or illness
- Concept Mapping
- Word Sense Disambiguation

ASIDE: Semantic Similarity

- Being able to organize concepts by their similarity or relatedness to each other is a fundamental operation in the human mind,
  - and in many problems in Natural Language Processing and Artificial Intelligence
- If we know a lot about X, and if we know Y is similar to X, then a lot of what we know about X may apply to Y
  - Use X to explain or categorize Y

ASIDE: Semantic Similarity

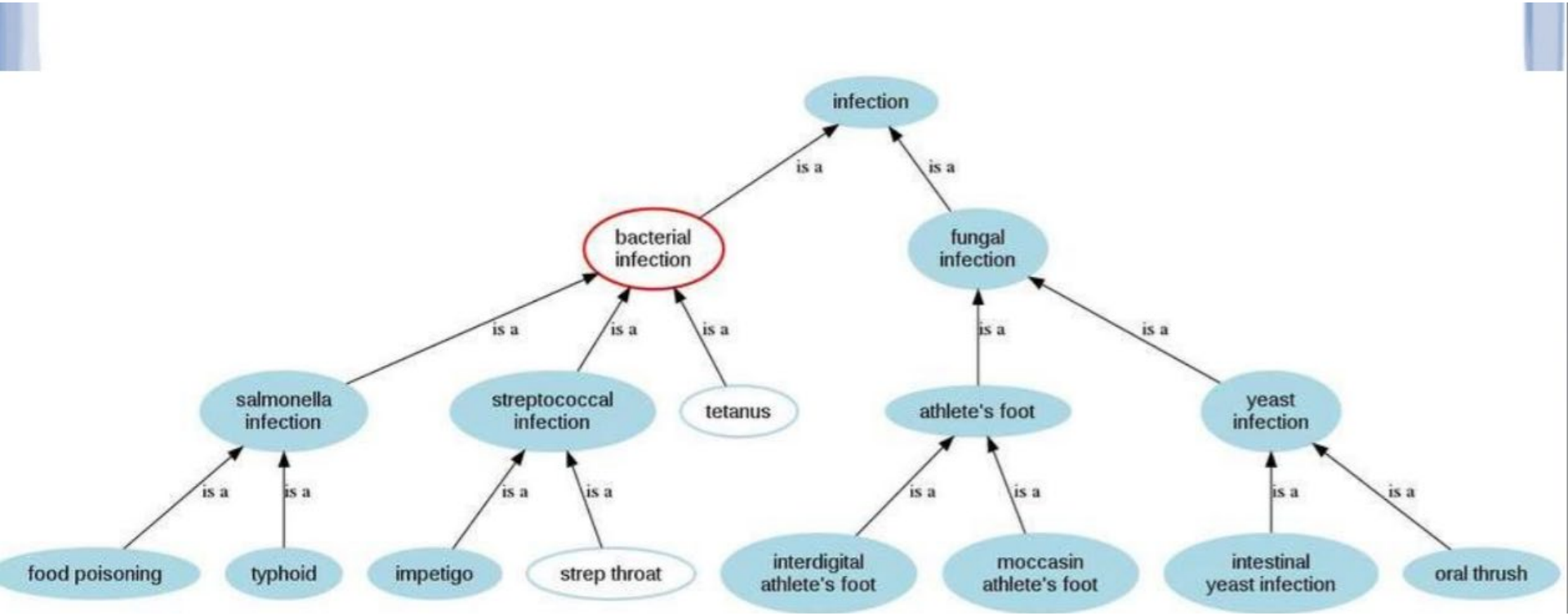
NORTHWESTERN UNIVERSITY

- Similar vs. Related…
  - Similarity based on is-a relations
- How much is X like Y?
  - Share ancestor in is-a hierarchy
  - LCS : least common subsumer
- Closer / deeper the ancestor the more similar
  - Tetanus and strep throat are similar both are kinds-of bacterial infections

ASIDE: Semantic Similarity

NORTHWESTERN UNIVERSITY

- LCS: Least Common Subsumer



ASIDE: Semantic Similarity

- Similarity vs. Relatedness
  - Relatedness more general
- How much is X related to Y?
  - E.g., penguin is related to Antartic

- Many ways to be related
- is-a, part-of, treats, affects, symptom-of, ...
  - Tetanus and deep cuts are related but they really aren't similar (deep cuts can cause tetanus)
- All similar concepts are related, but not all related concepts are similar

ASIDE: Semantic Similarity

NORTHWESTERN UNIVERSITY

- Measures:
- Path length
- Path + Depth in Hierarchy
- Path + Information Content