

# Introduction to Graph Database with Neo4j

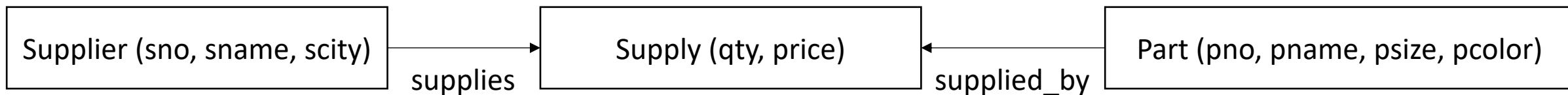
Zeyuan Hu

Dec. 4th 2020

Austin, TX

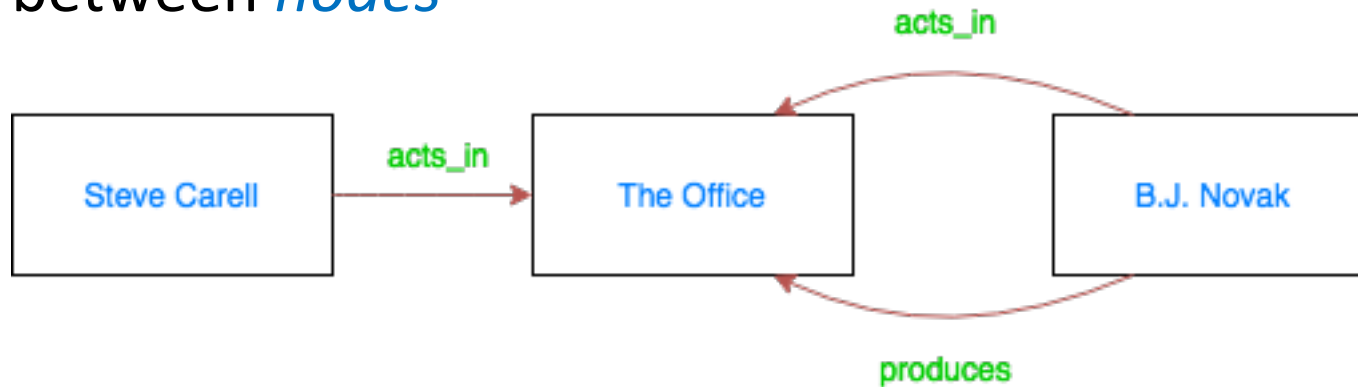
# History

- Lots of logical data models have been proposed in the history of DBMS
  - Hierarchical (IMS), Network (CODASYL), Relational, etc
- What Goes Around Comes Around
  - Graph database uses data models that are “spirit successors” of Network data model that is popular in 1970’s.
  - CODASYL = Committee on Data Systems Languages



# Edge-labelled Graph

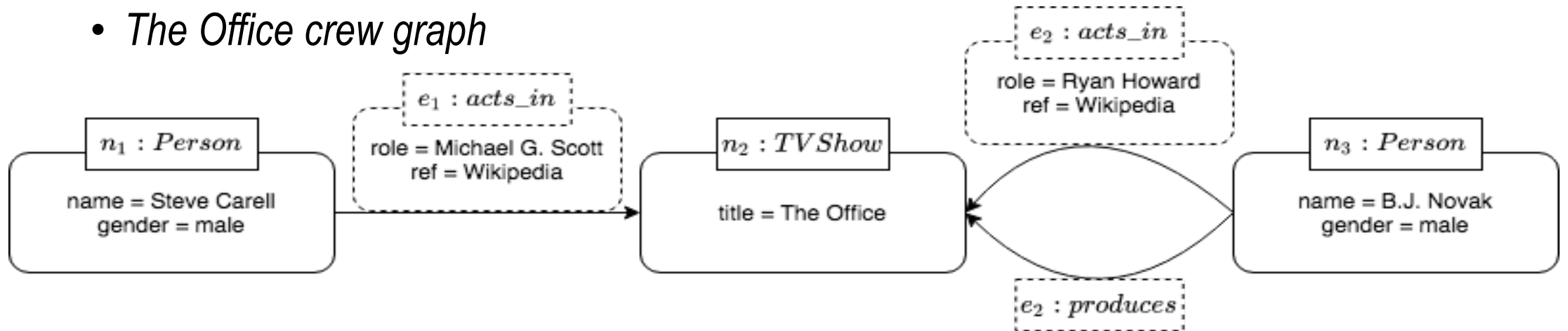
- We assign *labels* to *edges* that indicate the different types of relationships between *nodes*



- *Nodes* = {Steve Carell, The Office, B.J. Novak}
- *Edges* = {(Steve Carell, acts\_in, The Office), (B.J. Novak, produces, The Office), (B.J. Novak, acts\_in, The Office)}
- Basis of *Resource Description Framework (RDF)* aka. “Triplestore”

# The Property Graph Model

- Extends Edge-labelled Graph with **labels**
  - Both edges and nodes can be labelled with a set of property-value pairs **attributes** directly to each edge or node.
  - *The Office crew graph*



- Node  $n_1$  has **node label** *Person* with **attributes**: `<name, Steve Carell>`, `<gender, male>`
- Edge  $e_1$  has **edge label** *acts\_in* with **attributes**: `<role, Michael G. Scott>`, `<ref, Wikipedia>`

# Property Graph Against Other Players

- *v.s. Edge-labelled Graph Model*

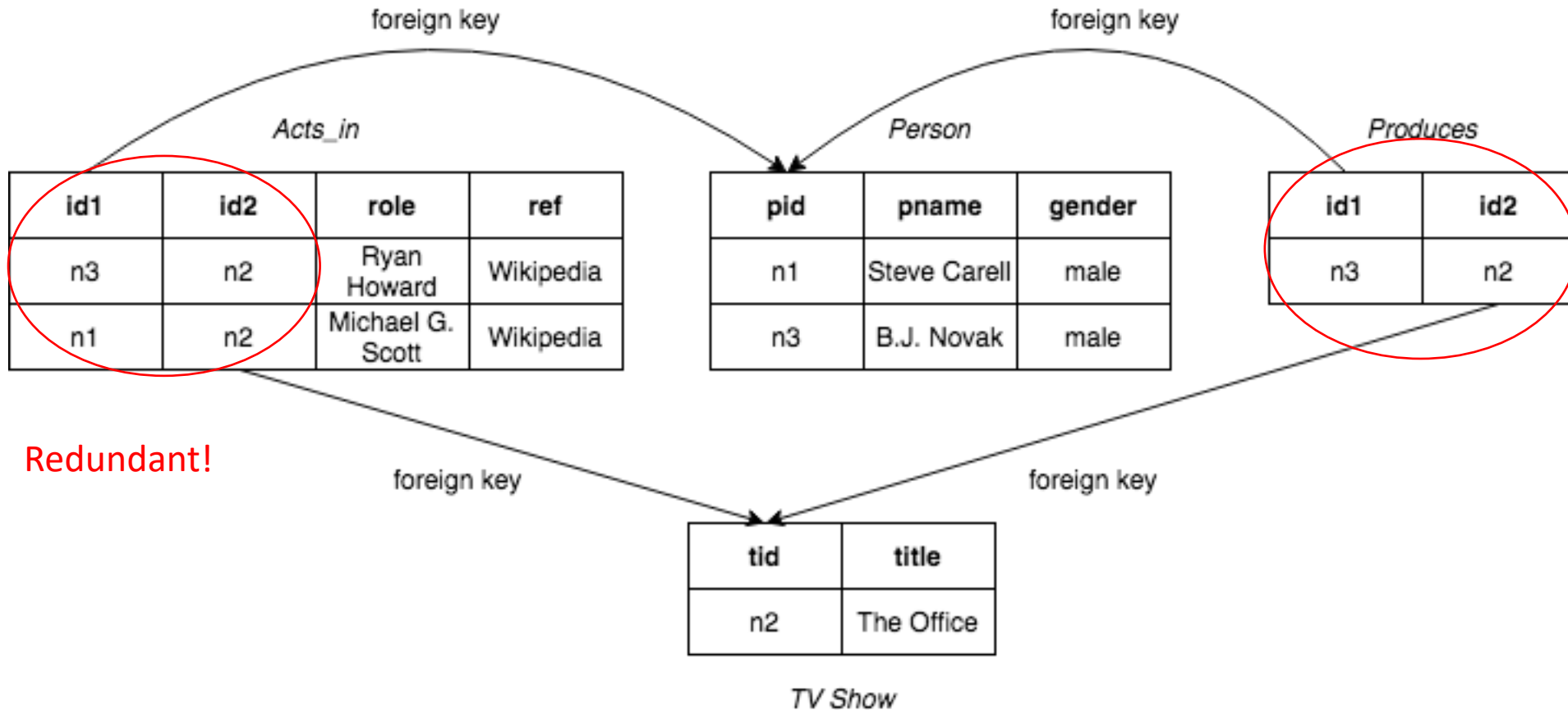
- Having node labels as part of the model can offer a more direct abstraction that is easier for users to query and understand
  - *Steve Carell and B.J. Novak can be labelled as Person*
- Suitable for scenarios where various new types of meta-information may regularly need to be added to edges or nodes

- *v.s. Relational Model*

- Graph Structure is more intuitive than a collection of tables (e.g., HW7 org chart)
- Avoid repetitive data storage from user perspective (e.g., primary key & foreign key)
- Enable same relation name with different attributes
  - `CREATE TABLE TVSHOW(title, year);`
  - `CREATE TABLE TVSHOW(title, year, production_company);` // Not possible!

# Same Data, Different Model

- The same data represented in relational model



# Neo4j

- Neo4j is a graph database that uses *property graph* data model with a query language called *Cypher*
- In graph database domain, there is no standard query language (yet). Many vendor-dependent flavors
  - **SPARQL** for RDF
  - **Cypher**, **Gremlin**, etc. for property graph
  - *Ex: Find co-stars of The Office*

```
PREFIX : <http://ex.org/#>
SELECT ?x1 ?x2
WHERE {
  ?x1 :acts_in ?x3 . ?x1 :type :Person .
  ?x2 :acts_in ?x3 . ?x2 :type :Person .
  ?x3 :title "The Office" . ?x3 :type :TVSHOW .
  FILTER(?x1 != ?x2)
}
```

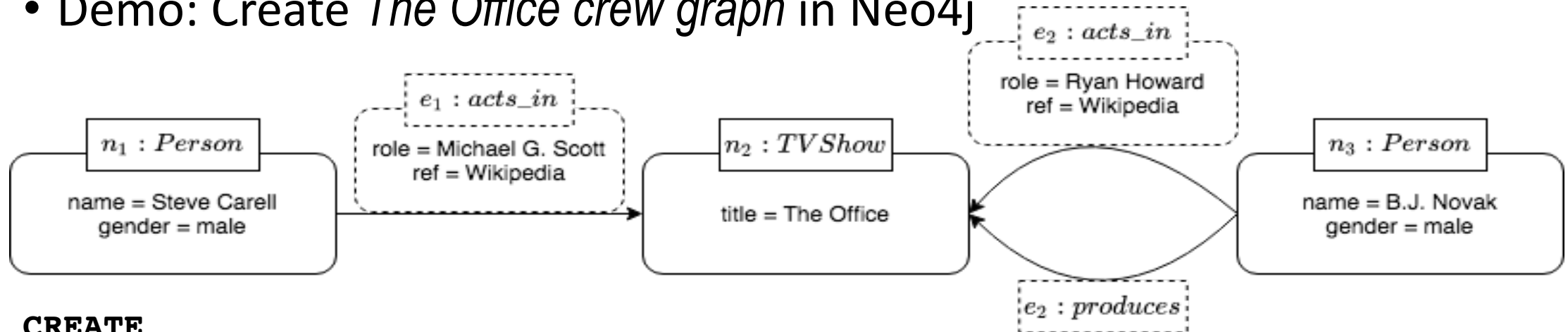
```
MATCH (x1:Person) -[:acts_in]->
      (:TVSHOW {title: "The Office"})
      <-[:acts_in]- (x2:Person)
RETURN x1, x2
```

```
g.V().has("TVSHOW", "title", "The Office").
in('acts_in').hasLabel("Person").
values("name")
```

- There has been ongoing standardization effort – Graph Query Language (GQL)

# First Property Graph with Neo4j

- Demo: Create *The Office* crew graph in Neo4j



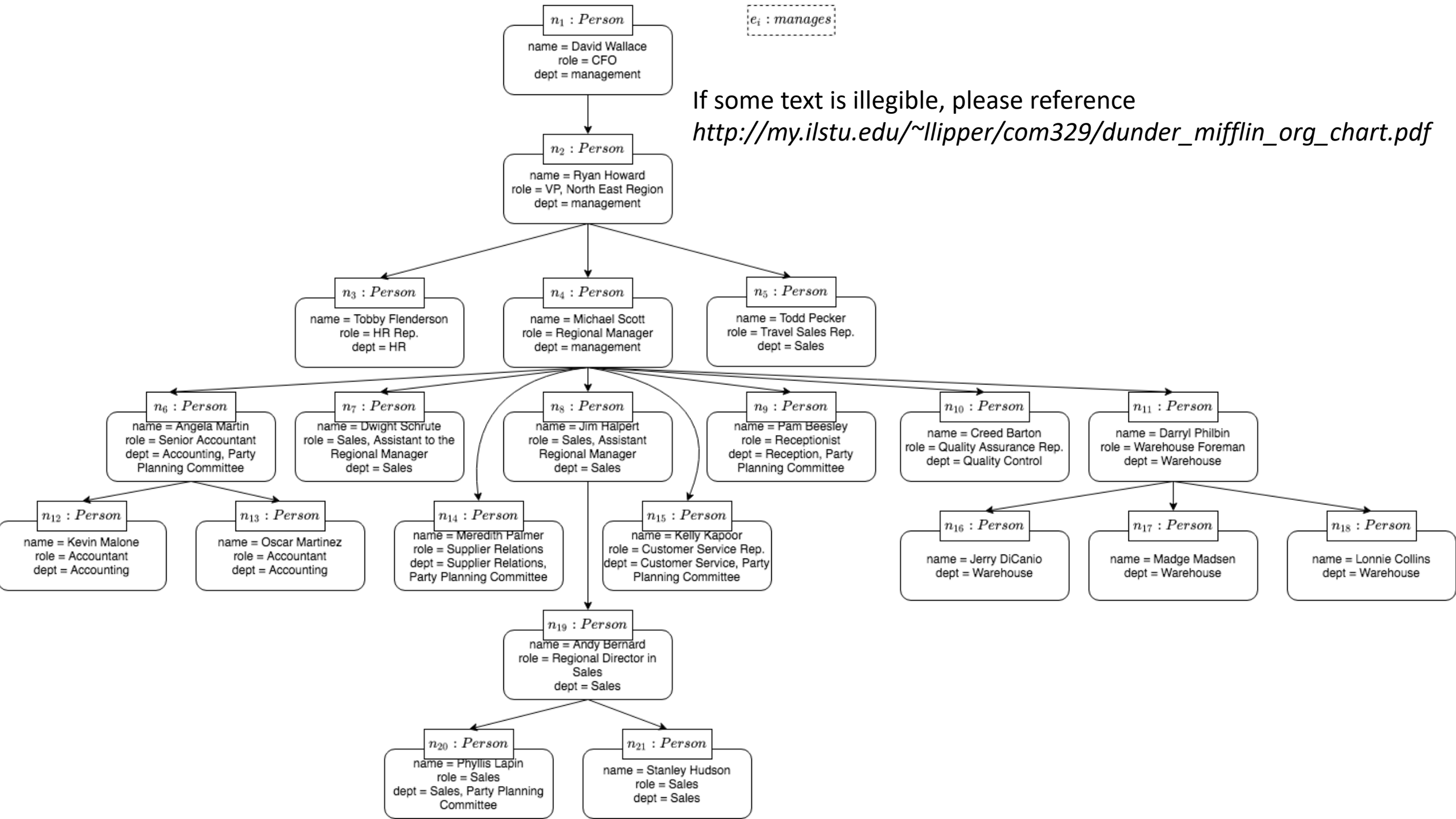
## CREATE

```
(n1:Person {name: "Steve Carell", gender: "male"}),  
(n2:Person {name: "B.J. Novak", gender: "male"}),  
(n3:TVShow {title: "The Office"}),  
(n1)-[:acts_in {role: "Michael G. Scott", ref: "Wikipedia"}]->(n3),  
(n2)-[:acts_in {role: "Ryan Howard", ref: "Wikipedia"}]->(n3),  
(n2)-[:produces]->(n3);
```



# Let's Practice

- Let's create the org. chart of Dunder Mifflin, Scranton Branch <sup>1</sup>
- All edges have labels  $e_i$ : *manages* with  $i$  being numbers from 1 to  $n$ , the number of edges
- Some useful commands & notes
  - See the graph - `MATCH (n) RETURN n LIMIT 50`
  - Delete the graph - `MATCH (n) DETACH DELETE n`
  - To create list of values, use "[ ]"
    - For example, `role: ["Sales", "Assistant Regional Manager"]`



$e_i : manages$

If some text is illegible, please reference [http://my.ilstu.edu/~llipper/com329/dunder\\_mifflin\\_org\\_chart.pdf](http://my.ilstu.edu/~llipper/com329/dunder_mifflin_org_chart.pdf)

# Graph Query Languages

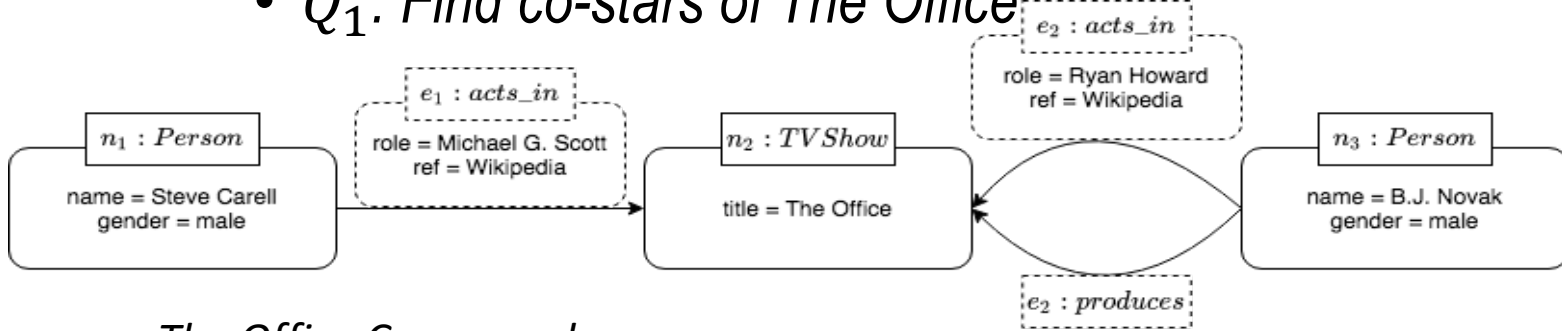
- Two important usage patterns for graph query languages:
  - Graph Pattern Matching
  - Graph Navigation
- We'll focus on Cypher in this tutorial. However, any significant graph query languages will have these two important patterns in their languages.

# Graph Pattern Matching

- Graph Pattern Matching

- A *match* is a mapping from variables to constants such that when the mapping is applied to the given pattern, the result is, roughly speaking, contained within the original graph (i.e., subgraph).

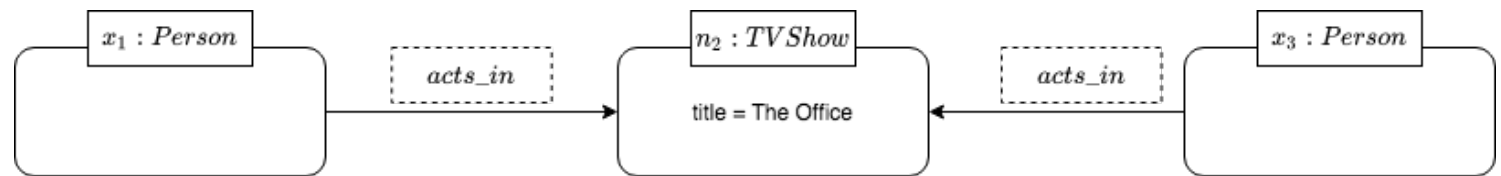
- $Q_1$ : Find co-stars of *The Office*



The Office Crew graph

$x_1$	$x_2$
Steve Carell	B.J. Novak

Result set (i.e., *matching*) for  $Q_1$



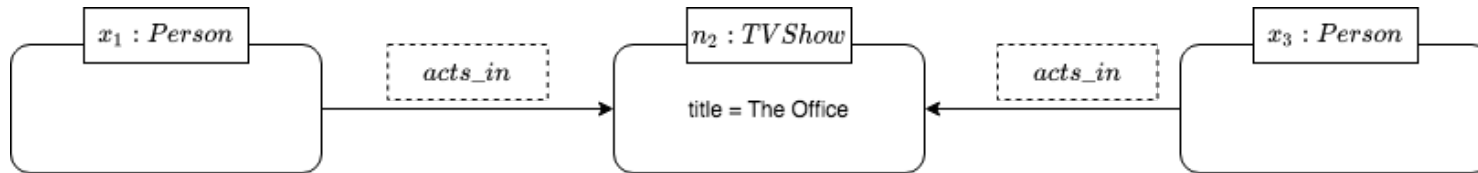
graph pattern for  $Q_1$

# Graph Pattern Matching Semantics

- Different query languages may have different evaluation rule for the input query graph pattern
  - No constraint at all (**Homomorphism-based semantics**)
    - *Ex: distinct variables can be mapped to same constants*
  - Certain types of variables are restricted to match distinct constants in the database (**Isomorphism-based semantics**)
    - No-repeated-anything semantics
      - *Variables mapped to nodes and edges have to be distinct*
    - No-repeated-node semantics
      - *Variables mapped to nodes have to be distinct*
    - No-repeated-edge semantics
      - *Variables mapped to edges have to be distinct*
- Another angle: Set vs. Bag
- Different languages have different semantics

# Graph Pattern Matching in Cypher

- Cypher has no-repeated-edges, bags semantics
- $Q_1$ : Find co-stars of The Office



```
MATCH (x1:Person) -[:acts_in]-> (:TVSHOW {title: "The Office"}) <-[:acts_in]- (x2:Person)  
RETURN x1, x2
```

Graph pattern  $x_1$  has to connect to TVSHOW node through an incoming edge with label `acts_in`

We want to match variable  $x_1$  to node with type Person

- Cypher manual:

- <https://neo4j.com/docs/cypher-manual/current/syntax/patterns/>

# Example

- Who's inside Party Planning Committee?

```
MATCH (p:Person)
WHERE "Party Planning Committee" in p.dept
return p.name
```

- How many people does Michael directly manage? (*hint: use count ( )* )

```
MATCH (p:Person)<-[:manages]-(n:Person)
WHERE n.name = "Michael Scott"
RETURN count(p)
```

- Get the Dunder Mifflin employees that are on the same level as “Michael Scott”

```
MATCH p1 = (n:Person)<-[:manages]-(p:Person)
MATCH p2 = (m:Person)<-[:manages]-(p:Person)
WHERE length(p1) = length(p2) AND m.name <> n.name AND n.name = "Michael Scott"
RETURN m
```

# Let's Practice

- Find all the employees that are directly managed by someone that reports to Michael

```
MATCH (p {name: 'Michael Scott'})-[:manages]->()-[:manages]->(fof)
RETURN fof.name
```

- Does Michael directly manage more employees than Jim Halpert?

```
MATCH (p:Person)<-[:manages]-(n:Person)
WHERE n.name = "Michael Scott"
WITH count(p) AS c1
MATCH (p:Person)<-[:manages]-(m:Person)
WHERE m.name = "Jim Halpert"
RETURN c1 > count(p)
```

Each MATCH ... WHERE can be thought as a SELECT ... FROM ... WHERE

```
MATCH (p:Person)<-[:manages]-(n:Person)
WHERE n.name = "Michael Scott"
MATCH (q:Person)<-[:manages]-(m:Person)
WHERE m.name = "Darryl Philbin"
RETURN p.name, q.name
```



# Same Data, Different Model

- Let's query the same data in Relational Model

empID	name	role	dept	mgrID
1	David Wallace	{ "CFO" }	{ "management" }	
2	Ryan Howard	{ "VP, North East Region" }	{ "management" }	1
3	Tobby Flenderson	{ "HR Rep." }	{ "HR" }	2
4	Michael Scott	{ "Regional Manager" }	{ "management" }	2
5	Stanley Hudson	{ "Sales Rep." }	{ "Sales" }	2
6	Dwight Krehbiel	{ "Sales Rep." }	{ "Sales" }	2
7	Angela Martin	{ "Sales Rep." }	{ "Sales" }	2
8	Kevin Conner	{ "Sales Rep." }	{ "Sales" }	2
9	Meredith Platter	{ "Sales Rep." }	{ "Sales" }	2
10	Timothy Underhill	{ "Sales Rep." }	{ "Sales" }	2
11	Erin Burdette	{ "Sales Rep." }	{ "Sales" }	2
12	Phyllis Phloog	{ "Sales Rep." }	{ "Sales" }	2
13	Joey Green	{ "Sales Rep." }	{ "Sales" }	2
14	Michael Van der Bilt	{ "Sales Rep." }	{ "Sales" }	2
15	Erin Cartwright	{ "Sales Rep." }	{ "Sales" }	2
16	Greg Hepler	{ "Sales Rep." }	{ "Sales" }	2
17	Michael Scott	{ "Regional Manager" }	{ "management" }	2
18	Stanley Hudson	{ "Sales Rep." }	{ "Sales" }	2
19	Dwight Krehbiel	{ "Sales Rep." }	{ "Sales" }	2
20	Angela Martin	{ "Sales Rep." }	{ "Sales" }	2
21	Kevin Conner	{ "Sales Rep." }	{ "Sales" }	2
22	Meredith Platter	{ "Sales Rep." }	{ "Sales" }	2
23	Timothy Underhill	{ "Sales Rep." }	{ "Sales" }	2
24	Erin Burdette	{ "Sales Rep." }	{ "Sales" }	2
25	Phyllis Phloog	{ "Sales Rep." }	{ "Sales" }	2
26	Joey Green	{ "Sales Rep." }	{ "Sales" }	2
27	Michael Van der Bilt	{ "Sales Rep." }	{ "Sales" }	2
28	Erin Cartwright	{ "Sales Rep." }	{ "Sales" }	2
29	Greg Hepler	{ "Sales Rep." }	{ "Sales" }	2
30	Michael Scott	{ "Regional Manager" }	{ "management" }	2

- Actual schema and data see "sql-ex-2.sql"

```
MATCH p1 = (n:Person)<-[:manages]-(p:Person)
MATCH p2 = (m:Person)<-[:manages]-(p:Person)
WHERE length(p1) = length(p2) AND m.name <>
n.name AND n.name = "Michael Scott"
RETURN m
```

# Same Data, Different Model

- Get the Dunder Mifflin employees that are on the same level as “Michael Scott”

```
with recursive samelevel(s1, s2, s3, s4) as (  
  (select a1.name, a1.mgrID, a2.name, a2.mgrID  
    from dunderMifflin a1, dunderMifflin a2  
    where a1.mgrID = a2.mgrID)  
  union  
  (select a1.name, a1.mgrID, a2.name, a2.mgrID  
    from dunderMifflin a1, dunderMifflin a2, samelevel l1  
    where a1.mgrID = l1.s2 and a2.mgrID = l1.s4)  
) select l2.s3 from samelevel L2 where l2.s1 = 'Michael Scott' and l2.s1 <> l2.s3;
```

Base case: if two people are at the same level, their manager has to be the same.

Recursion: Same idea as base case but use the base relation and the result table we just computed in base case.

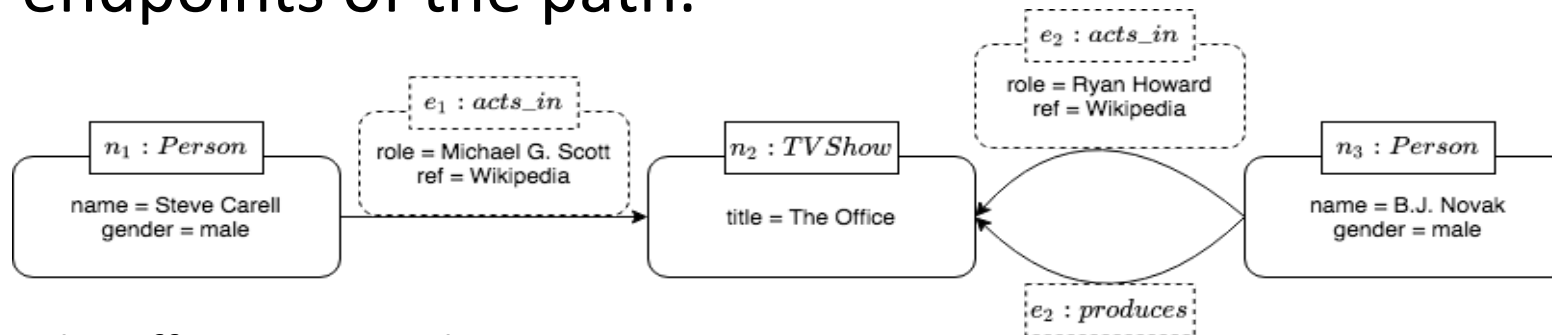
# Graph Navigation

- A mechanism provided by graph query languages to *navigate* the topology of the data.
- Two important query classes:
  - Path Query
  - Path Query + Graph Pattern Matching (i.e., navigational graph pattern)

# Path Query

Often represented using Regular Expressions

- Path query has the general form  $P = x \xrightarrow{\alpha} y$  where  $\alpha$  specifies conditions on the paths we wish to retrieve and  $x$  and  $y$  are the endpoints of the path.



*The Office Crew graph*

- $Q_1$ : Find co-stars of The Office

$$P := x \xrightarrow{\text{acts\_in} \cdot \text{acts\_in}^-} y$$

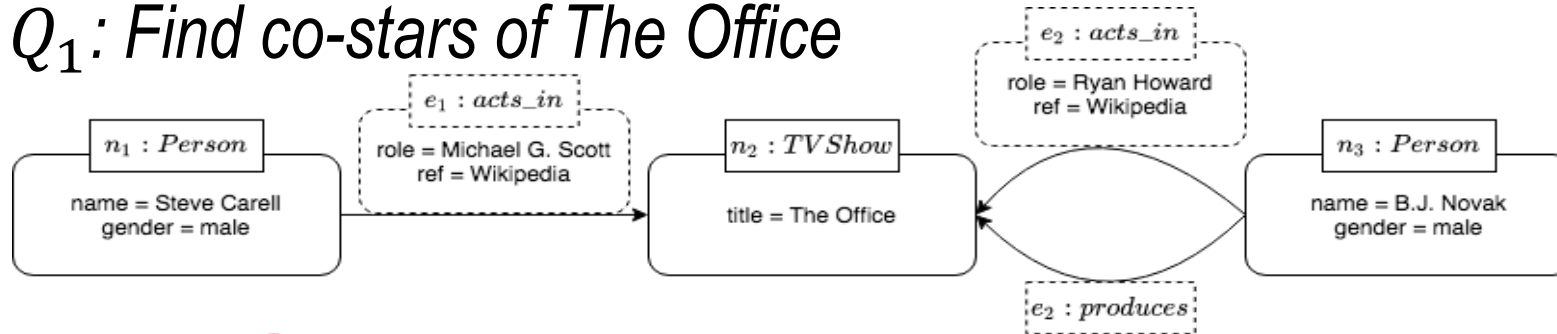
Edge has direction!

# Path Query Semantics

- There are different semantics for path query evaluation:
  - **Arbitrary path semantics**
    - All paths are considered
    - Useful when user only cares about whether there is a path or pairs of nodes are connected by such paths
  - **Shortest path semantics**
    - Only paths of minimal length that satisfy  $\alpha$  in  $P$  are considered
    - Useful when we want to find shortest path for a pair of nodes
  - **No-repeated-node semantics**
    - All matching paths with each node appears once in the path (i.e., simple path)
  - **No-repeated-edge semantics**
    - All matching paths with each edge appears once in the path

# Path Query in Cypher

- Cypher has no-repeated-edge, bags semantics
- $Q_1$ : Find co-stars of The Office



```
MATCH path=(p:Person)-[:acts_in]->(:TVShow)<-[:acts_in]-(q:Person)
return path
```

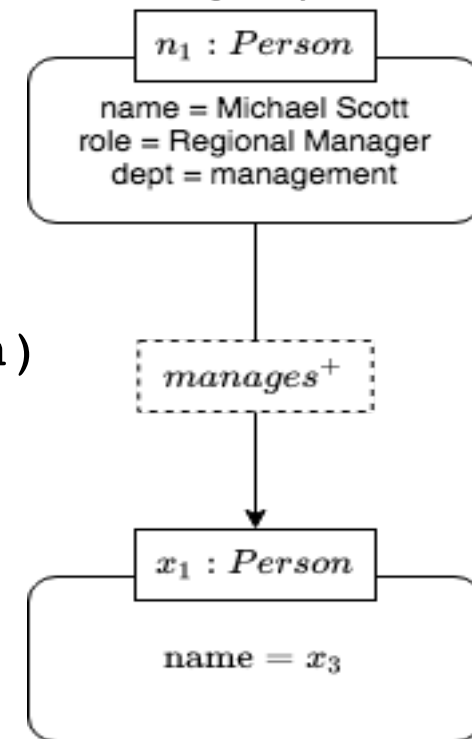
- Nothing new but we return a path now!

"path"
[{"gender": "male", "name": "Steve Carell"}, {"ref": "Wikipedia", "role": "Michael G. Scott"}, {"title": "The Office"}, {"title": "The Office"}, {"ref": "Wikipedia", "role": "Ryan Howard"}, {"gender": "male", "name": "B.J. Novak"}]
[{"gender": "male", "name": "B.J. Novak"}, {"ref": "Wikipedia", "role": "Ryan Howard"}, {"title": "The Office"}, {"title": "The Office"}, {"ref": "Wikipedia", "role": "Michael G. Scott"}, {"gender": "male", "name": "Steve Carell"}]

# Navigational Graph Pattern in Cypher

- We can combine path query with graph pattern matching by allowing edge labels in the graph pattern to be paths
- Q2: *Find all the people that Michael Scott manages*

```
MATCH path=(p:Person)-[:manages*1..]->(q:Person)
WHERE p.name = "Michael Scott"
return q.name
```



- Resources: <https://neo4j.com/docs/cypher-manual/current/syntax/patterns/#cypher-pattern-relationship>

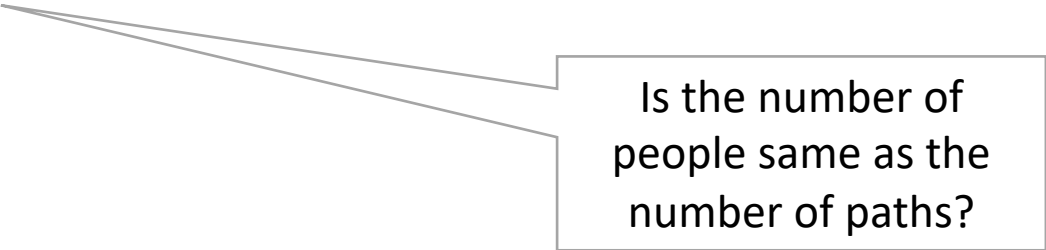
# Let's Practice

- Does Jim Halpert manage Phyllis Lapin?

```
MATCH path=(p:Person)-[:manages*1..]->(q:Person)
WHERE p.name = "Jim Halpert" and q.name = "Phyllis Lapin"
return count(path)
```

- Find all people that are indirectly managed by Michael Scott

```
MATCH path=(p1:Person {name: "Michael Scott"})-[:manages*1..]->(
    -[:manages*1..]->(p2:Person)
return collect(distinct p2)
```



Is the number of people same as the number of paths?



# Graph Algorithms in Cypher

- Cypher and many graph query languages allow user to directly embed graph algorithms inside the query
- *Q3: Find the shortest path between David Wallace and Andy Bernard*

```
MATCH path = shortestPath(  
  (p:Person {name: "David Wallace"})-[:manages*1..]- (q:Person {name: "Andy Bernard"}))  
RETURN path
```

# Conclusion

- Introduced Edge-label Graph, Property Graph
  - Discussed their difference with each other and with Relational Model
- Introduced graph query languages
  - SPARQL for RDF (i.e., Edge-label Graph), Gremlin and Cypher for Property Graph
  - Introduced three important usage patterns in graph query languages
    - Graph Pattern Matching
    - Path Query
    - Navigational Graph Pattern Matching
  - Demonstrated and practiced those usage patterns in Cypher with Neo4j

# Moving Forward

- Gremlin
  - <https://kelvinlawrence.net/book/Gremlin-Graph-Guide.html>
  - <https://tinkerpop.apache.org/docs/current/tutorials/getting-started/>
- Contrast among Cypher, SQL, and Datalog on the same data
  - <https://github.com/xxks-kkk/Code-for-blog/tree/master/2020/sql-datalog-cypher>
- Code for this tutorial
  - <https://github.com/xxks-kkk/Code-for-blog/tree/master/2020/intro-to-graphdb-with-neo4j>
- Slides available
  - <https://zhu45.org/introduction-to-graph-database-with-neo4j.pdf>