# Graph Databases

# Graph Databases

- Data Model:
  - Nodes and Relationships
- Examples:
  - Neo4j, OrientDB, InfiniteGraph, AllegroGraph

# Graph Databases: Pros and Cons

- Pros:
  - Powerful data model, as general as RDBMS
  - Connected data locally indexed
  - Easy to query
  - Scales up reasonably well
- Cons
  - Sharding (lots of people working on this)

# Introduction to Graphs

# What are graphs good for?

- Recommendations
- Business intelligence
- Social computing
- Geospatial
- Systems management
- Web of things
- Genealogy
- Time series data
- Product catalogue
- Web analytics
- Scientific computing (especially bioinformatics)
- And much more!

# What is a Graph?

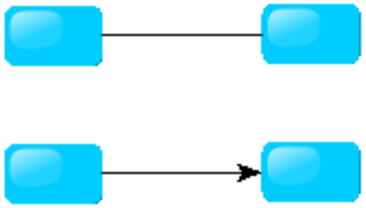
An abstract representation of a set of objects where some pairs are connected by links.


Object (Vertex, Node)

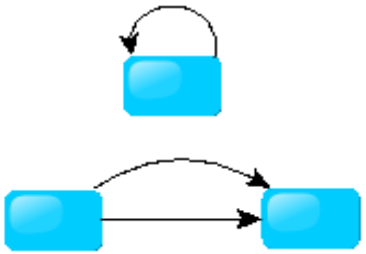
Link (Edge, Arc, Relationship)

# Different Kinds of Graphs

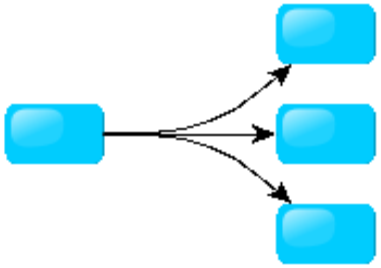
- Undirected Graph
- Directed Graph



- Pseudo Graph
- Multi Graph

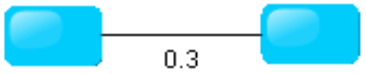


- Hyper Graph

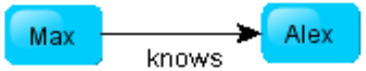


# More Kinds of Graphs

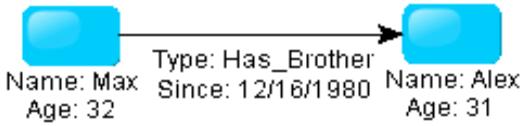
- Weighted Graph



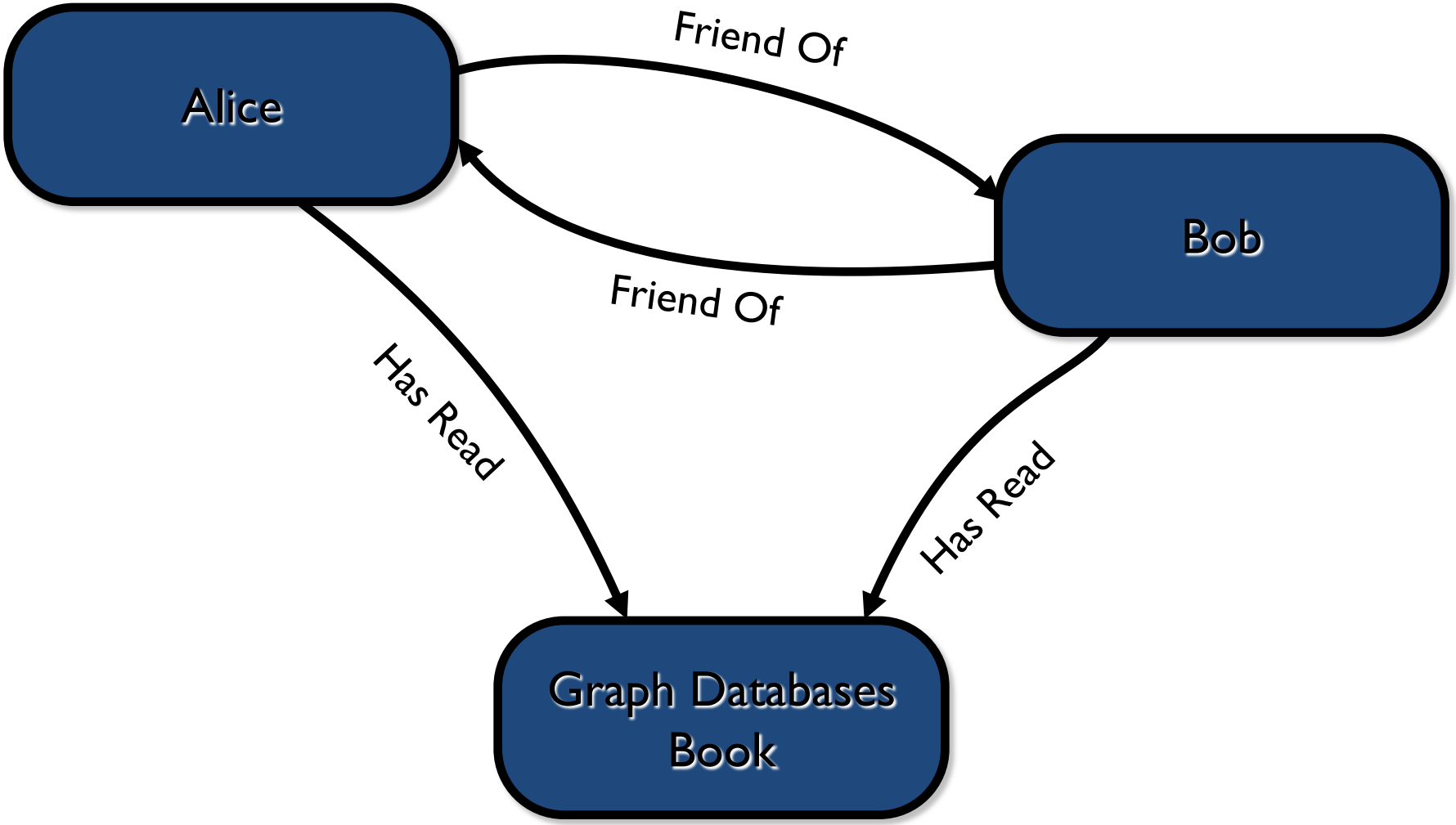

- Labeled Graph

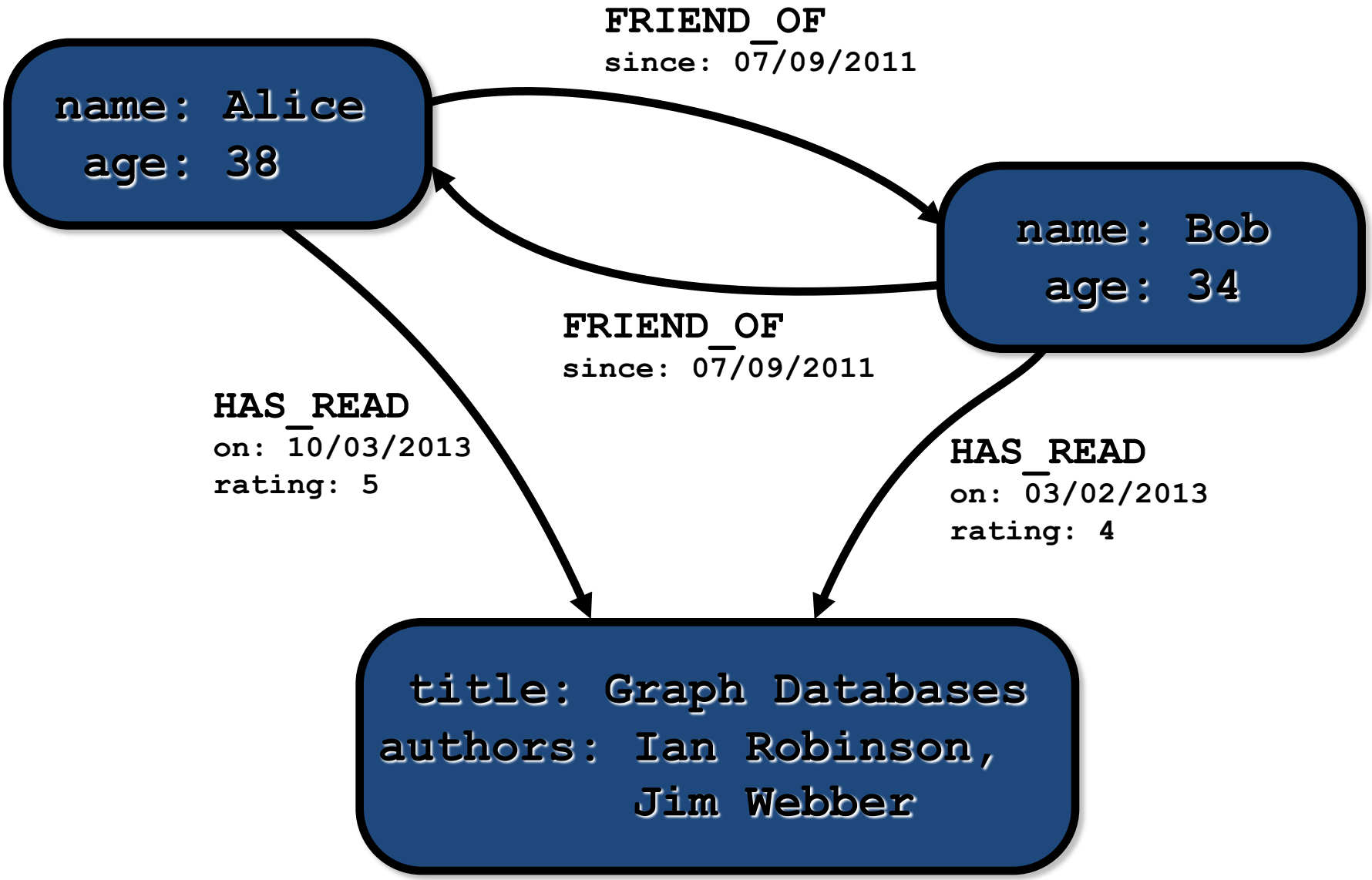



- Property Graph



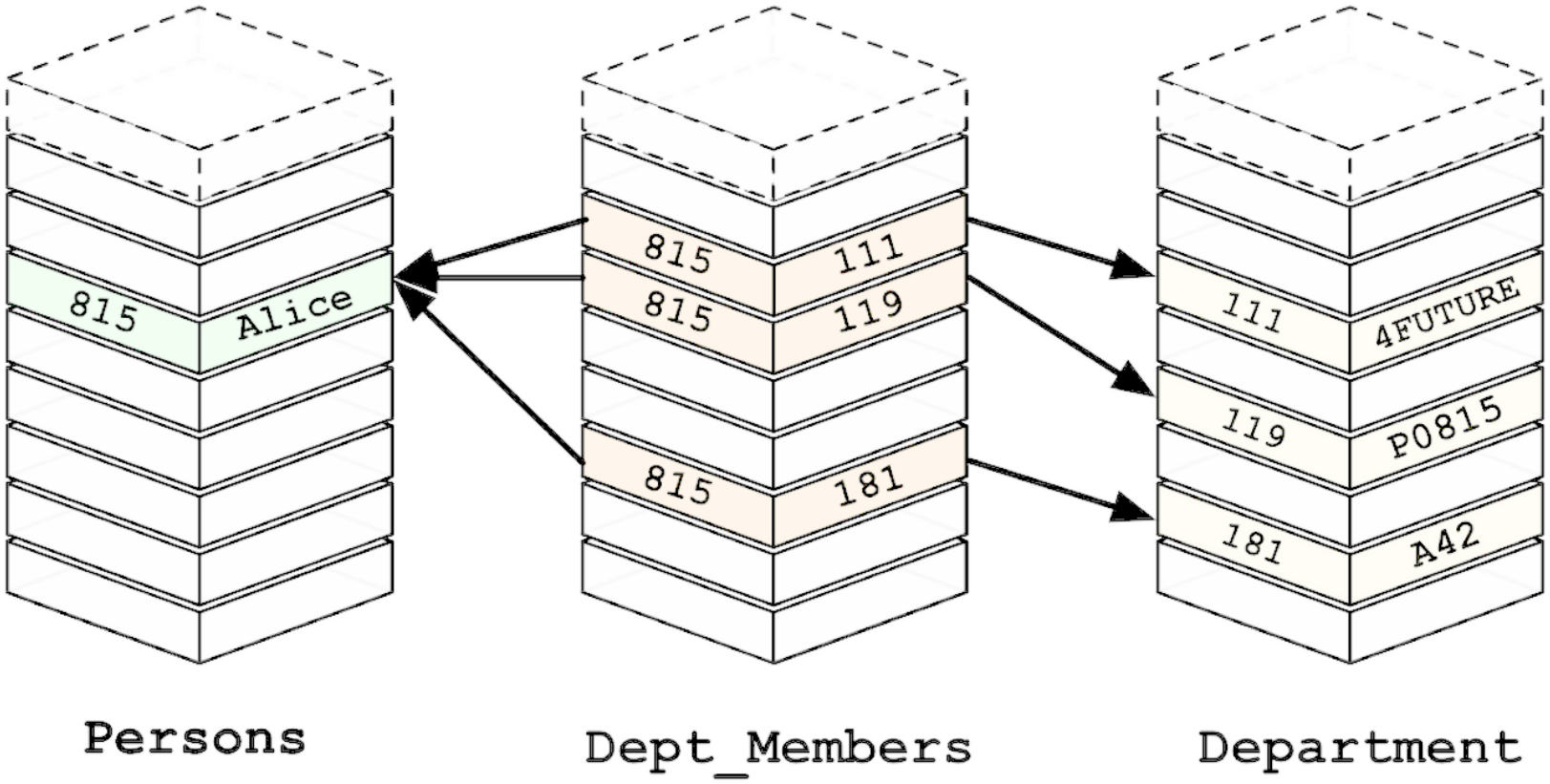
# Data in Graph Databases


Alice
Friend Of
Bob
Friend Of
Has Read
Has Read
Graph Databases
Book

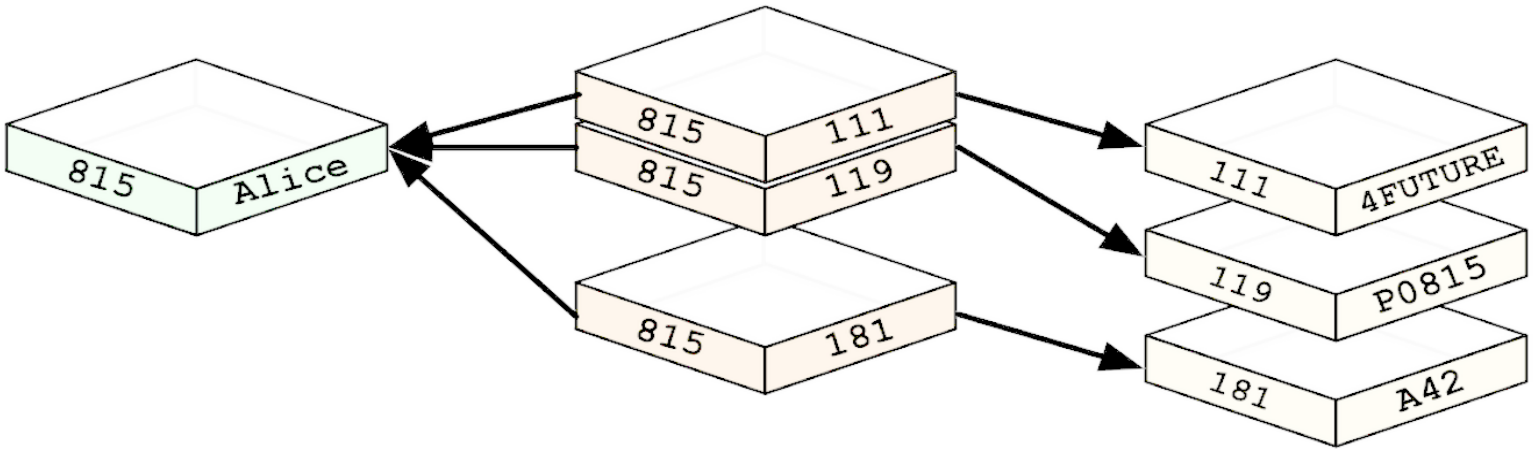

FRIEND_OF
since: 07/09/2011
name: Alice
age: 38
name: Bob
age: 34
FRIEND_OF
since: 07/09/2011
HAS_READ
on: 10/03/2013
rating: 5
HAS_READ
on: 03/02/2013
rating: 4
title: Graph Databases
authors: Ian Robinson,
Jim Webber

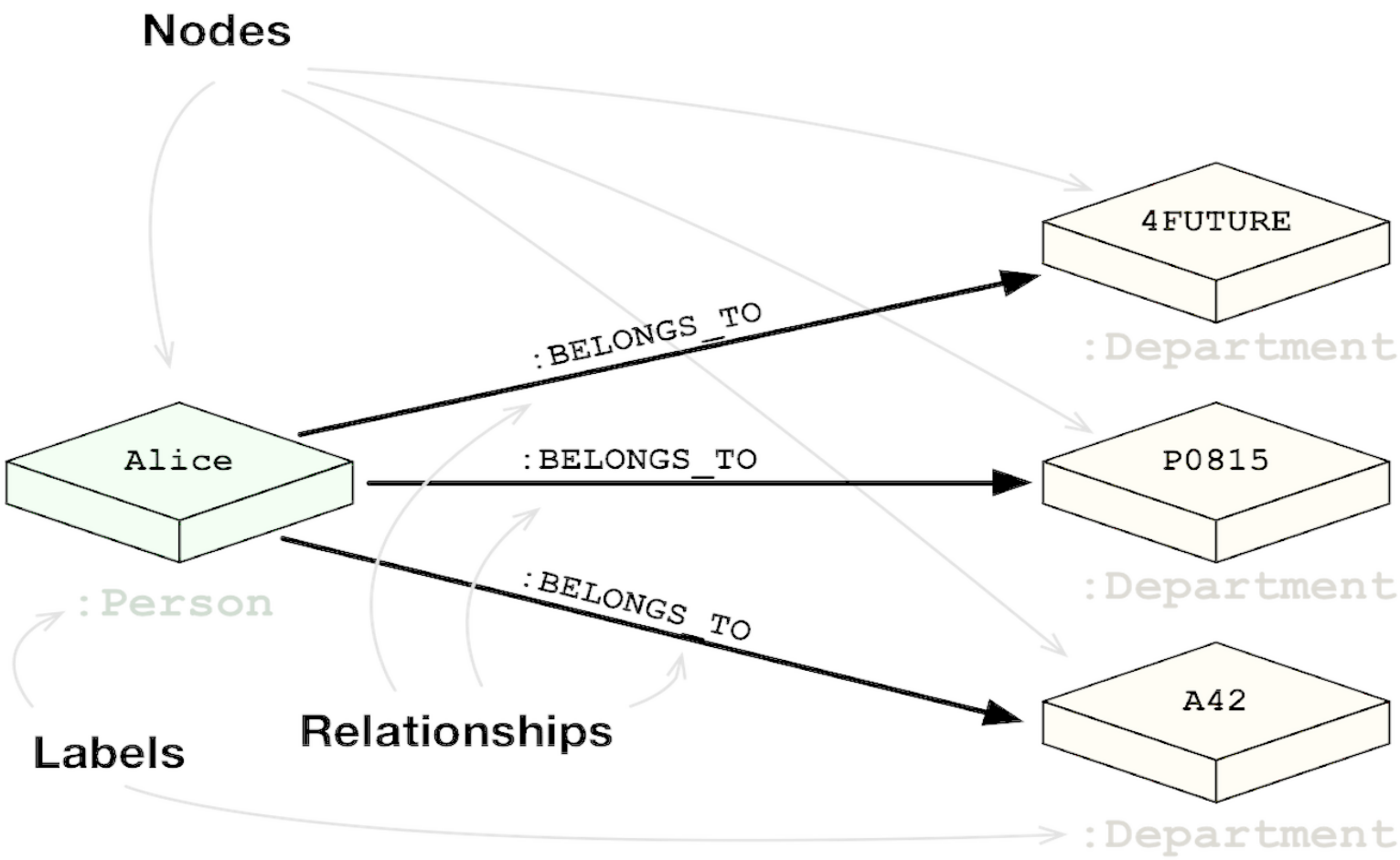
# Graphs & Relational Databases

# Relational Databases

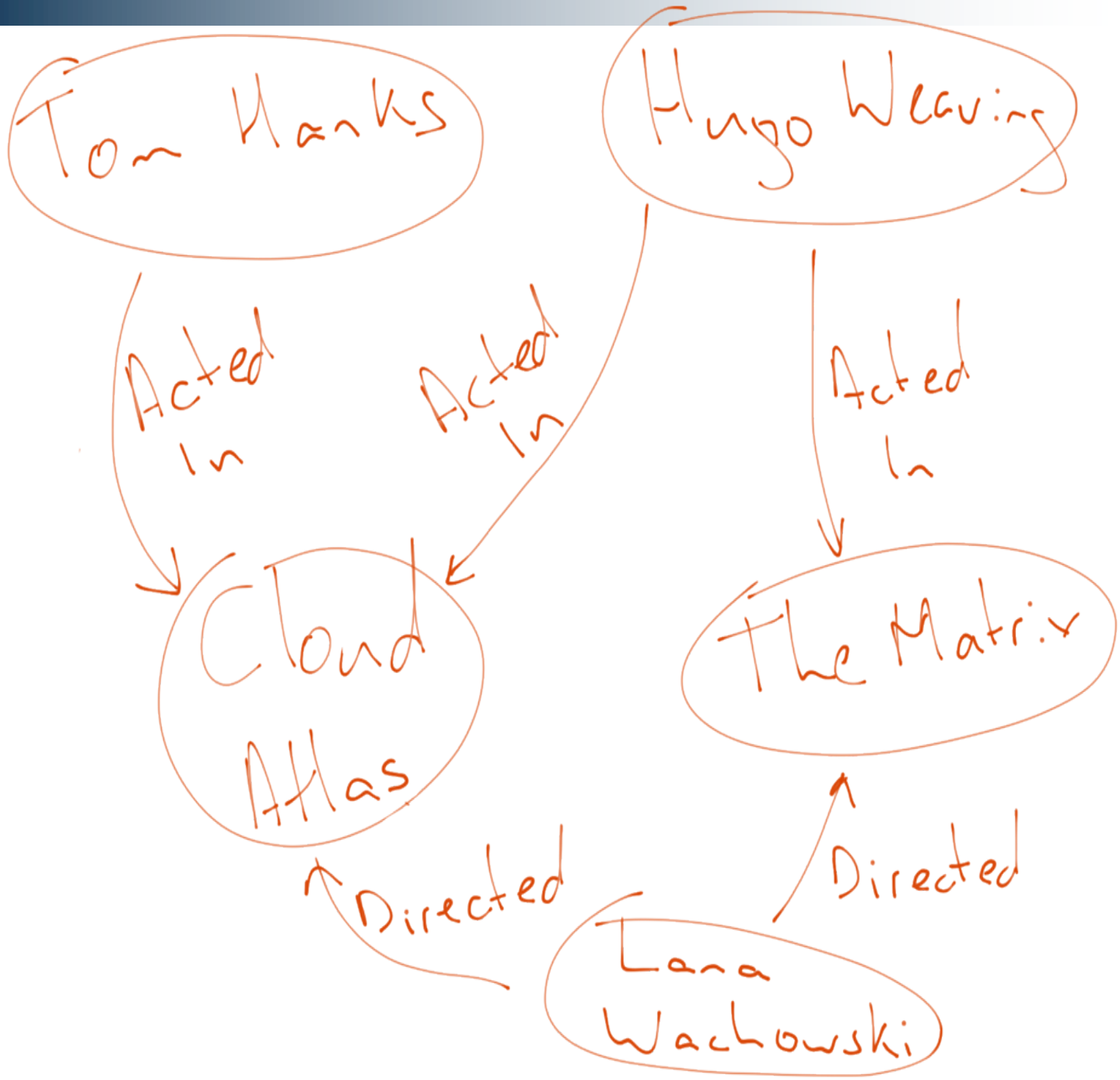


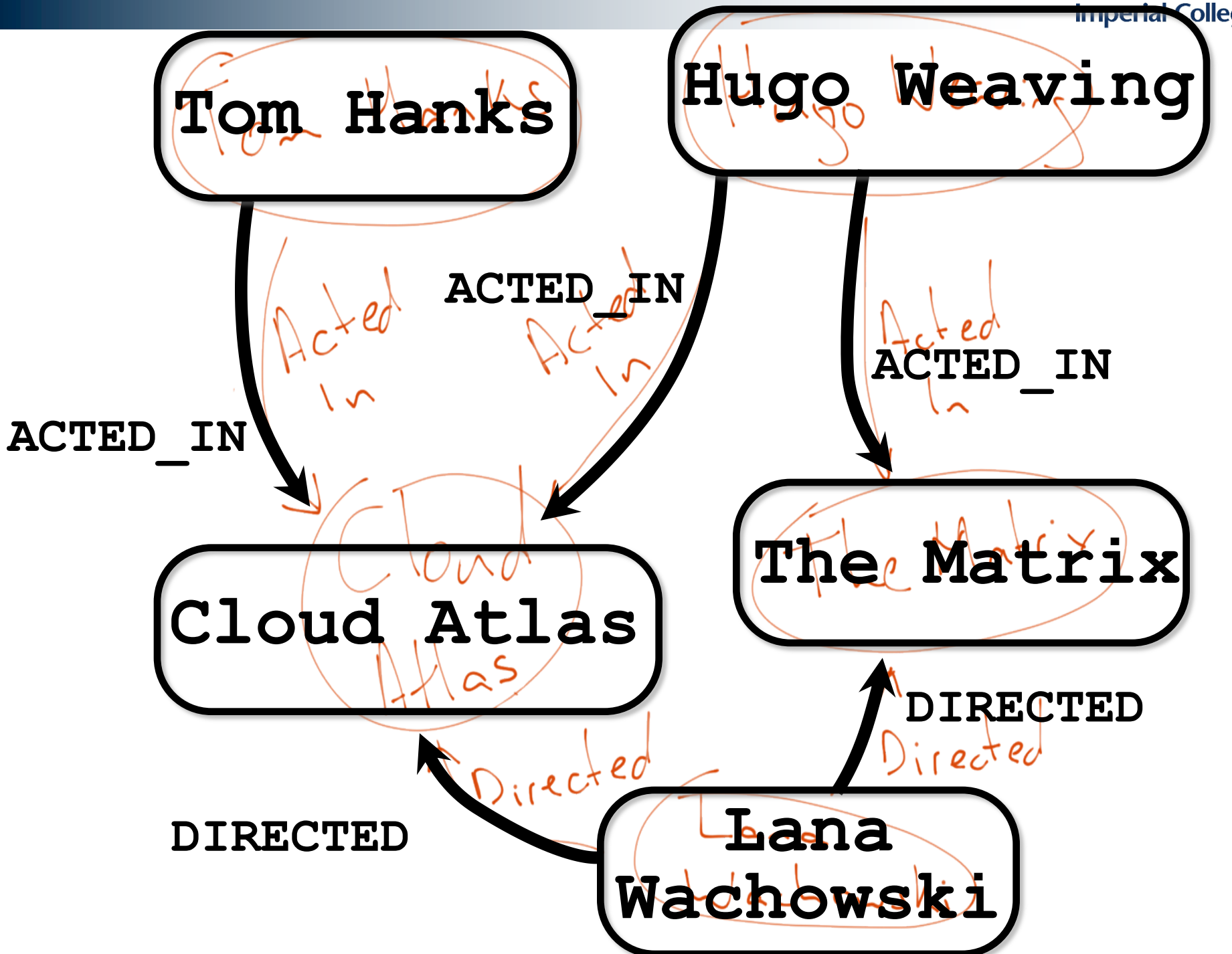
# Graph Databases

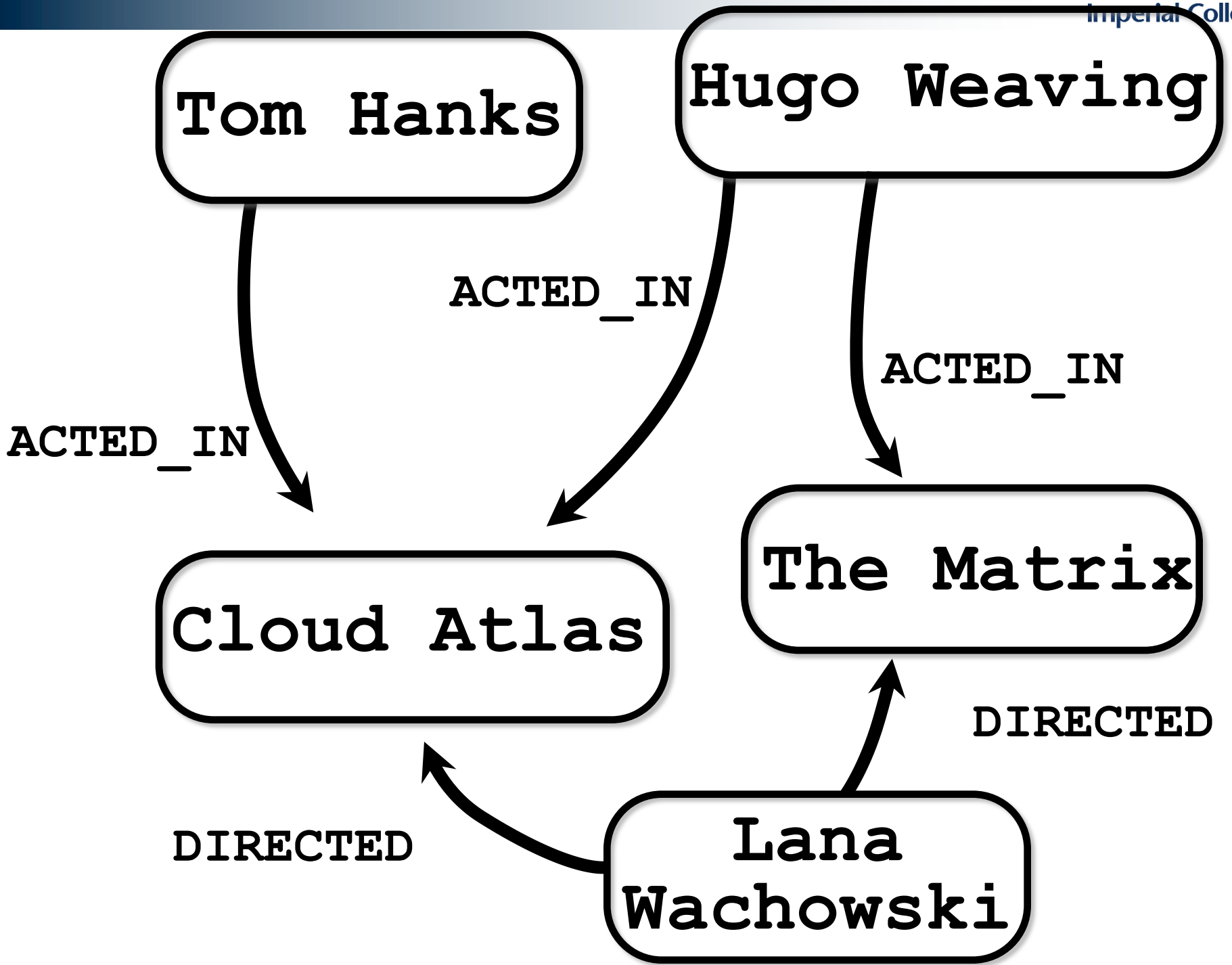


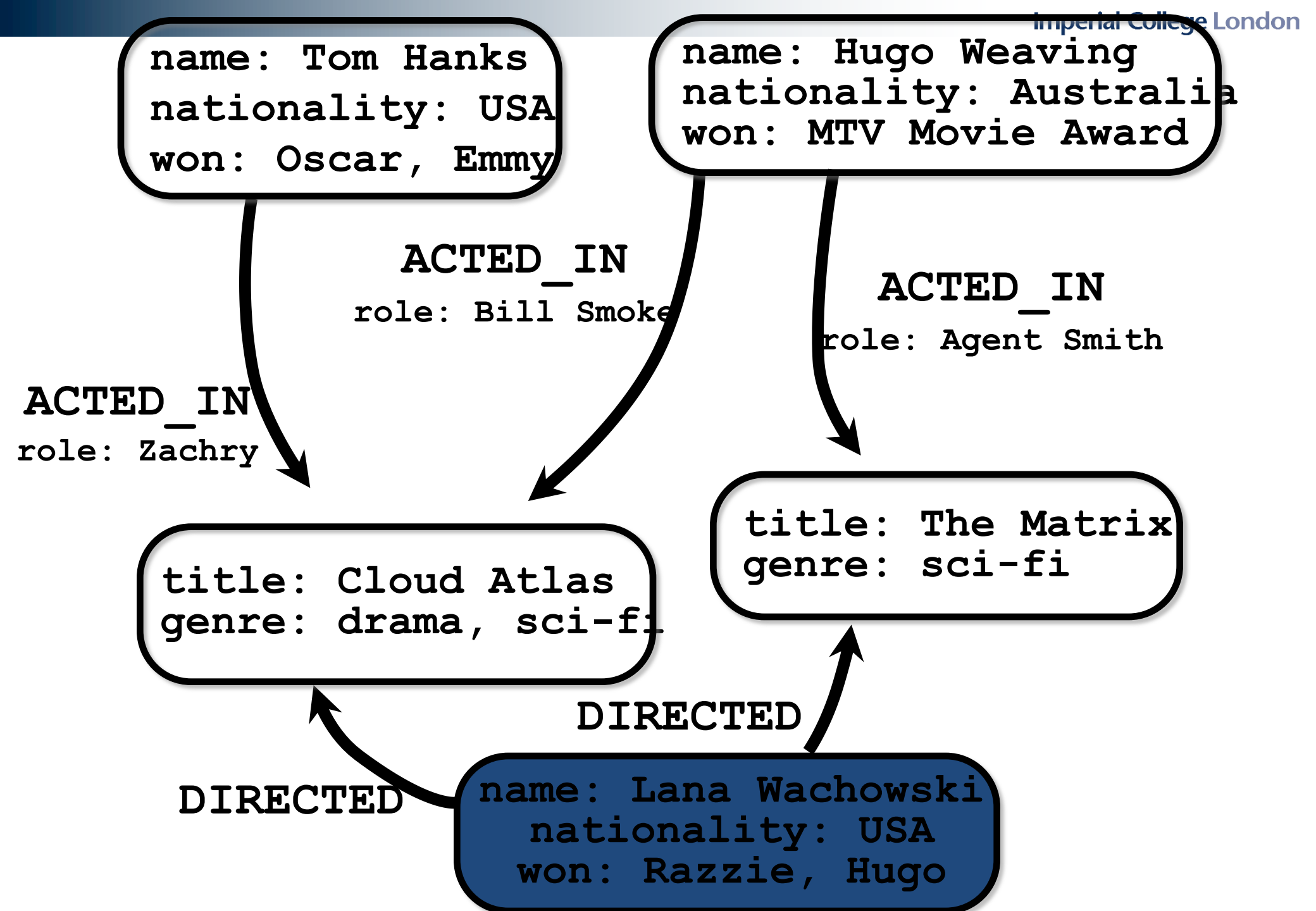

Nodes
4FUTURE
:Department
:BELONGS_TO
Alice
:BELONGS_TO
P0815
:Department
:Person
:BELONGS_TO
A42
Relationships
Labels
:Department

# Modeling with Graphs


Tom Hanks
Hugo Weaving
Acted In
Acted In
Acted In
Cloud Atlas
The Matrix
Directed
Directed
Lana Wachowski


Tom Hanks
Hugo Weaving
ACTED_IN
ACTED_IN
ACTED_IN
Cloud Atlas
The Matrix
DIRECTED
DIRECTED
Lana Wachowski


Tom Hanks
Hugo Weaving
ACTED_IN
ACTED_IN
ACTED_IN
Cloud Atlas
The Matrix
DIRECTED
DIRECTED
Lana Wachowski


name: Tom Hanks
nationality: USA
won: Oscar, Emmy
name: Hugo Weaving
nationality: Australia
won: MTV Movie Award
ACTED_IN
role: Bill Smoke
ACTED_IN
role: Agent Smith
ACTED_IN
role: Zachry
title: Cloud Atlas
genre: drama, sci-fi
title: The Matrix
genre: sci-fi
DIRECTED
DIRECTED
name: Lana Wachowski
nationality: USA
won: Razzie, Hugo

# Overview of Neo4j

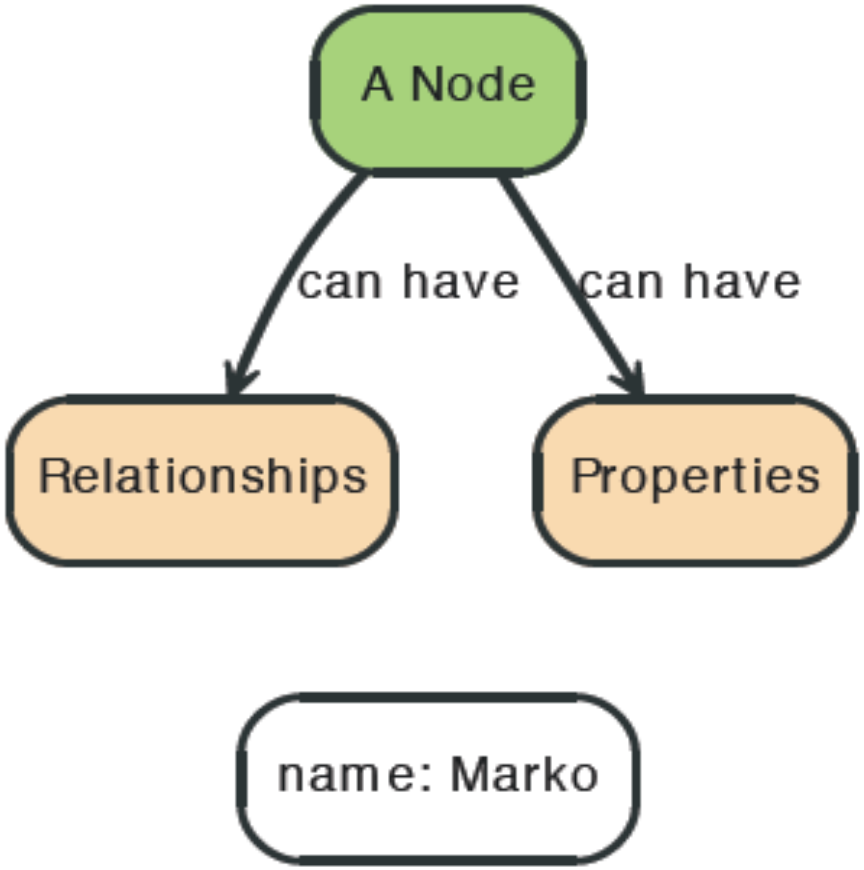
# What is a Graph Database?

- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups

# Translating to Neo4j

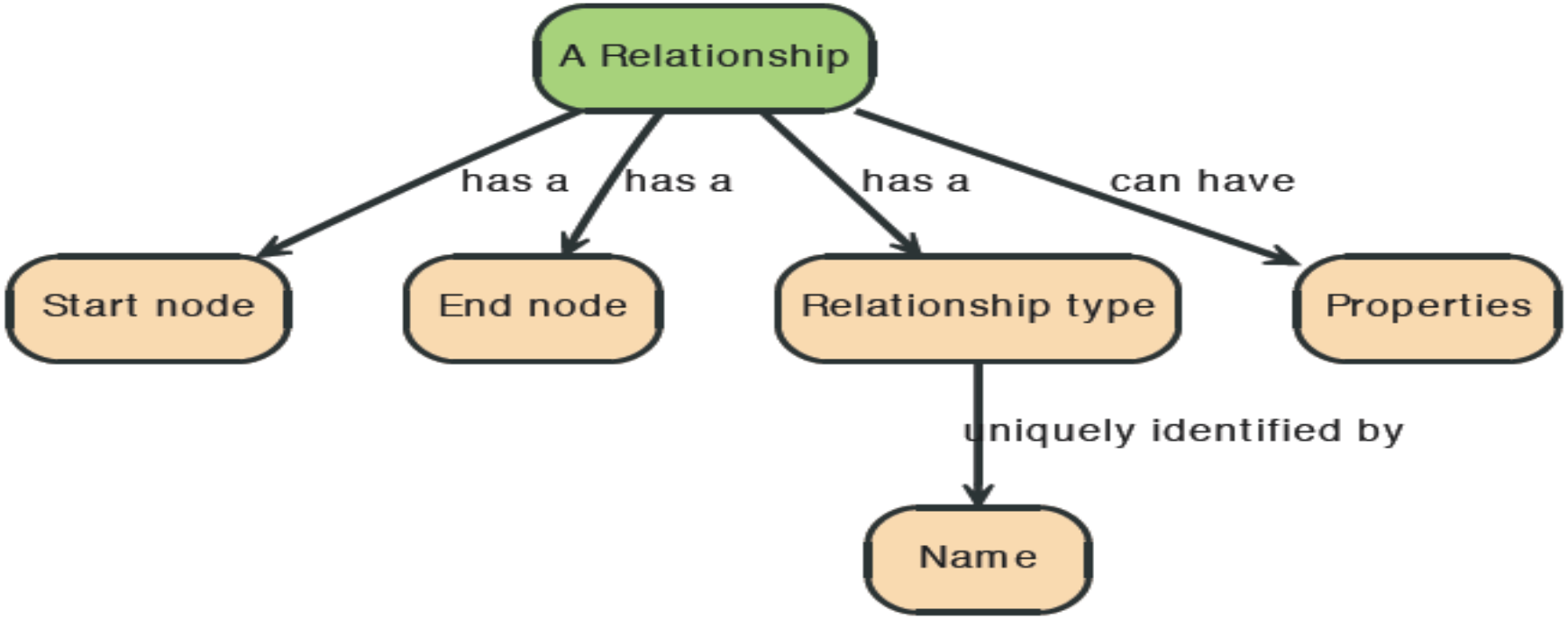
- Each entity table is represented by a label on nodes
- Each row in a entity table is a node
- Columns on those tables become node properties.
- Add unique constraints for business primary keys, add indexes for frequent lookup attributes

# Node in Neo4j

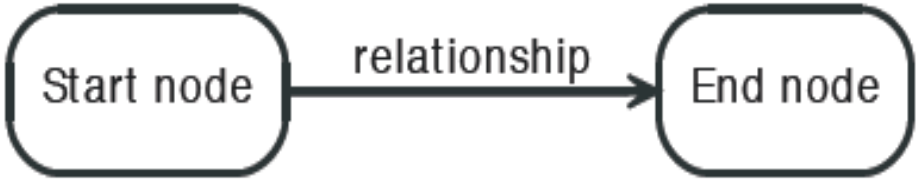


# Relationships in Neo4j

Relationships between nodes are a key part of Neo4j.



# Relationships in Neo4j



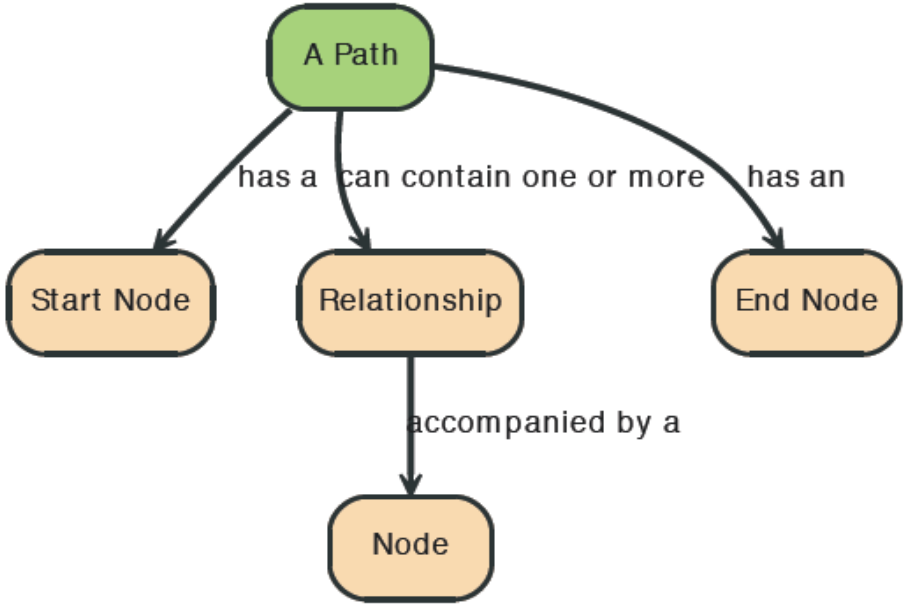


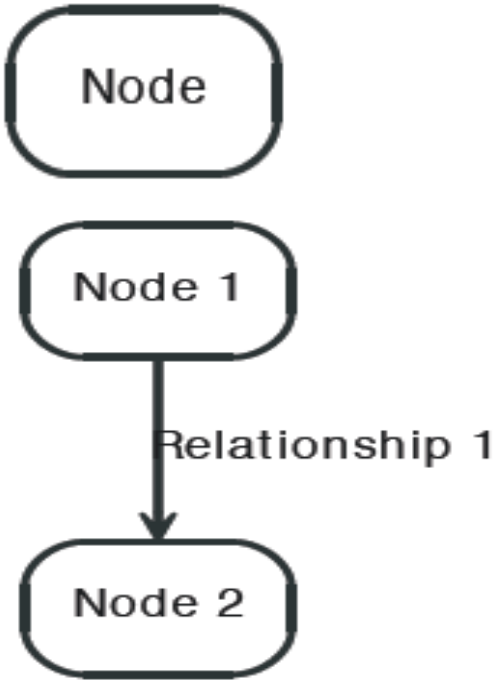
# Properties

- Both nodes and relationships can have properties.
- Properties are key-value pairs where the key is a string.
- Property values can be either a primitive or an array of one primitive type.
- For example String, int and int[] values are valid for properties.

# Paths in Neo4j

A path is one or more nodes with connecting relationships, typically retrieved as a query or traversal result.

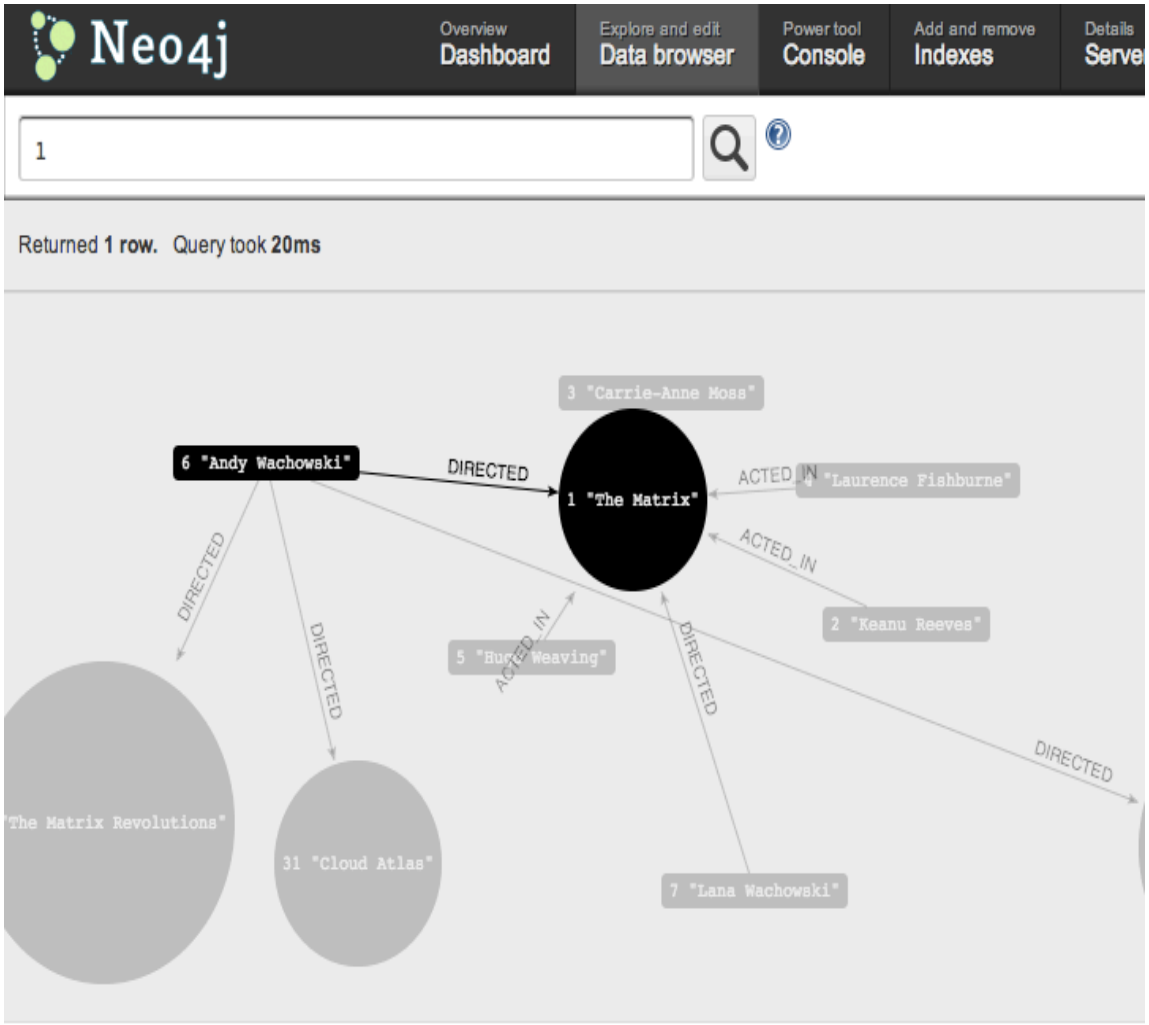


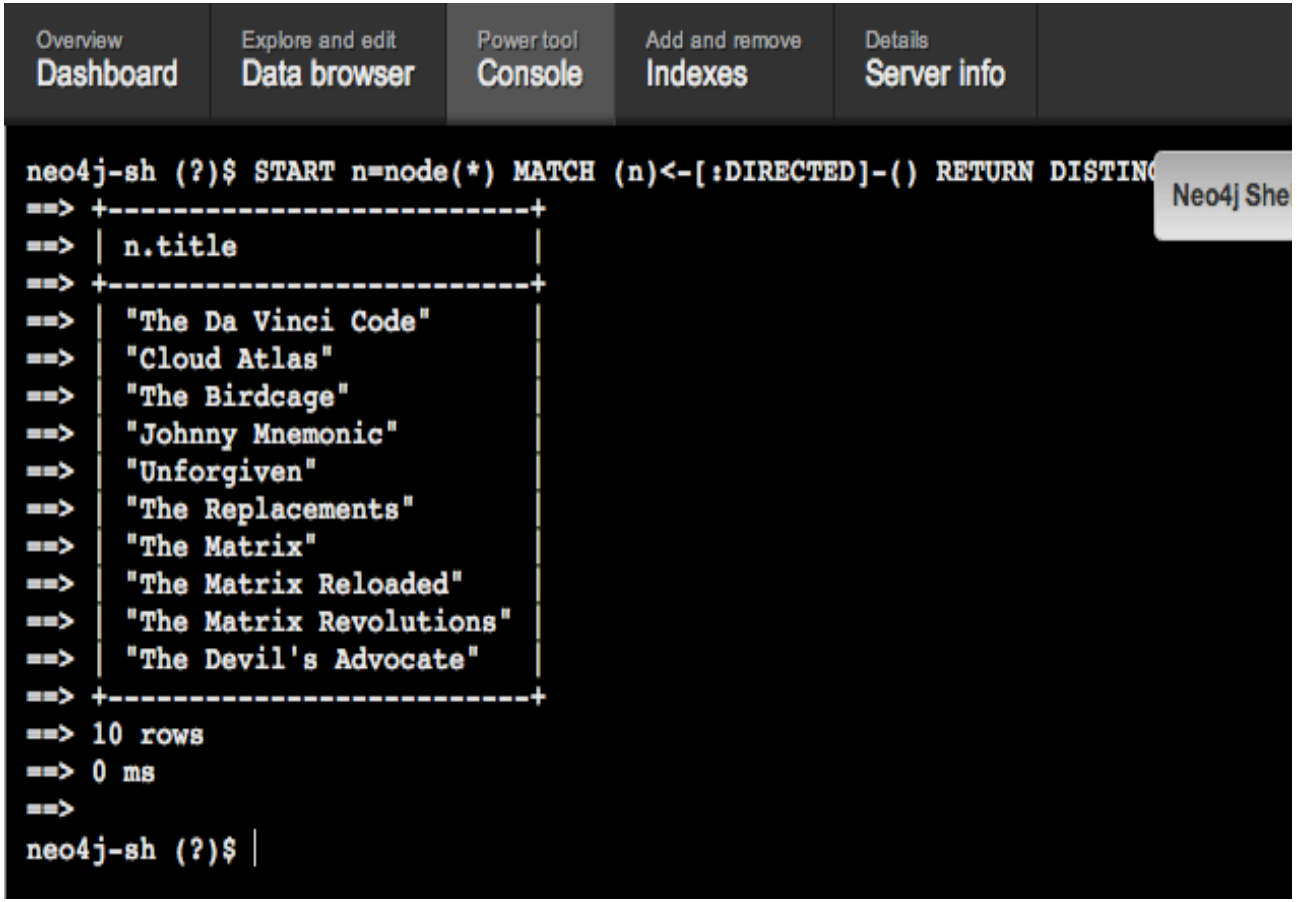
# Test Server

https://neo4j.com/lp/try-neo4j-sandbox/

# Neo4j Data Browser



# Neo4j Console



# Introduction to Cypher

- Declarative Pattern-Matching language
- SQL-like syntax
- Designed for graphs

# Two nodes, one relationship

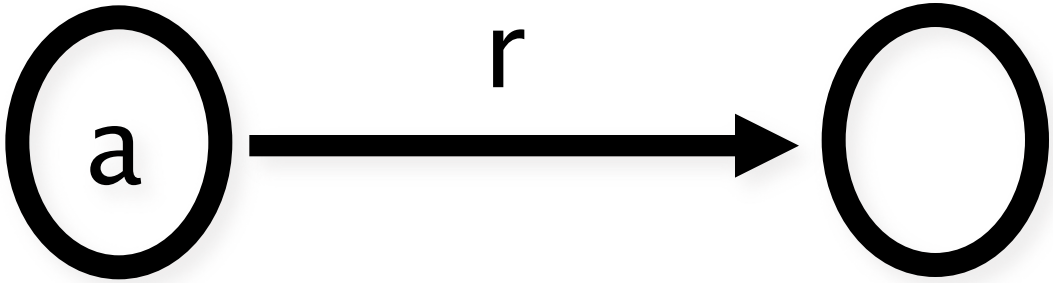
```
START a=node(*)
MATCH (a)-->(b)
RETURN a, b;
```

# Two nodes, one relationship

```
START a=node(*)
MATCH (a)-->()
RETURN a;
```


```
START a=node(*)
MATCH (a)-->()
RETURN a.name;
```

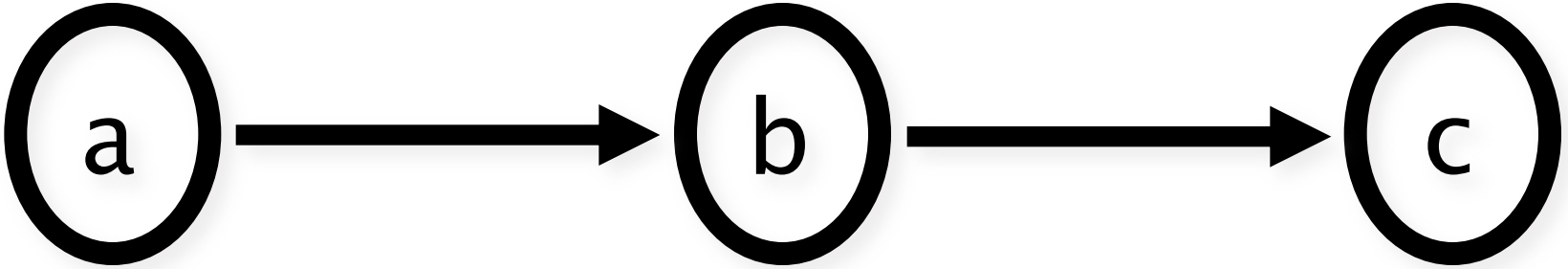
# Two nodes, one relationship




```
(a) -[r]-> ()
```

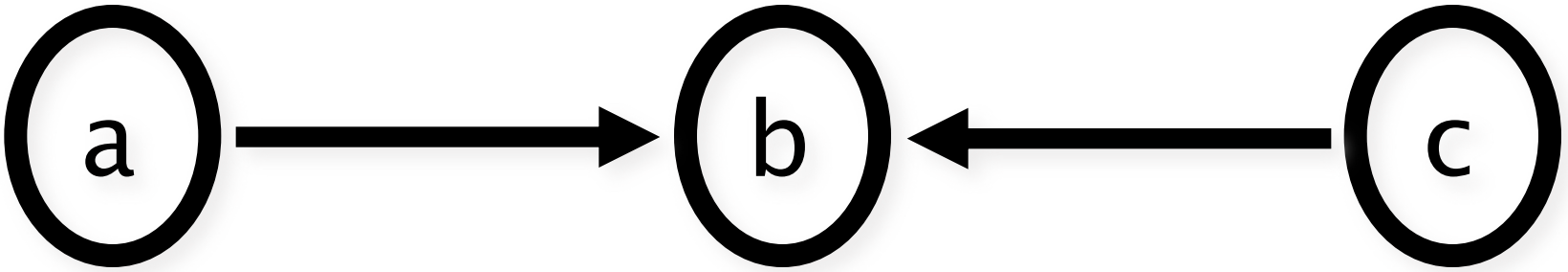

```
(a) -[:ACTED_IN]-> (m)
```

# Paths

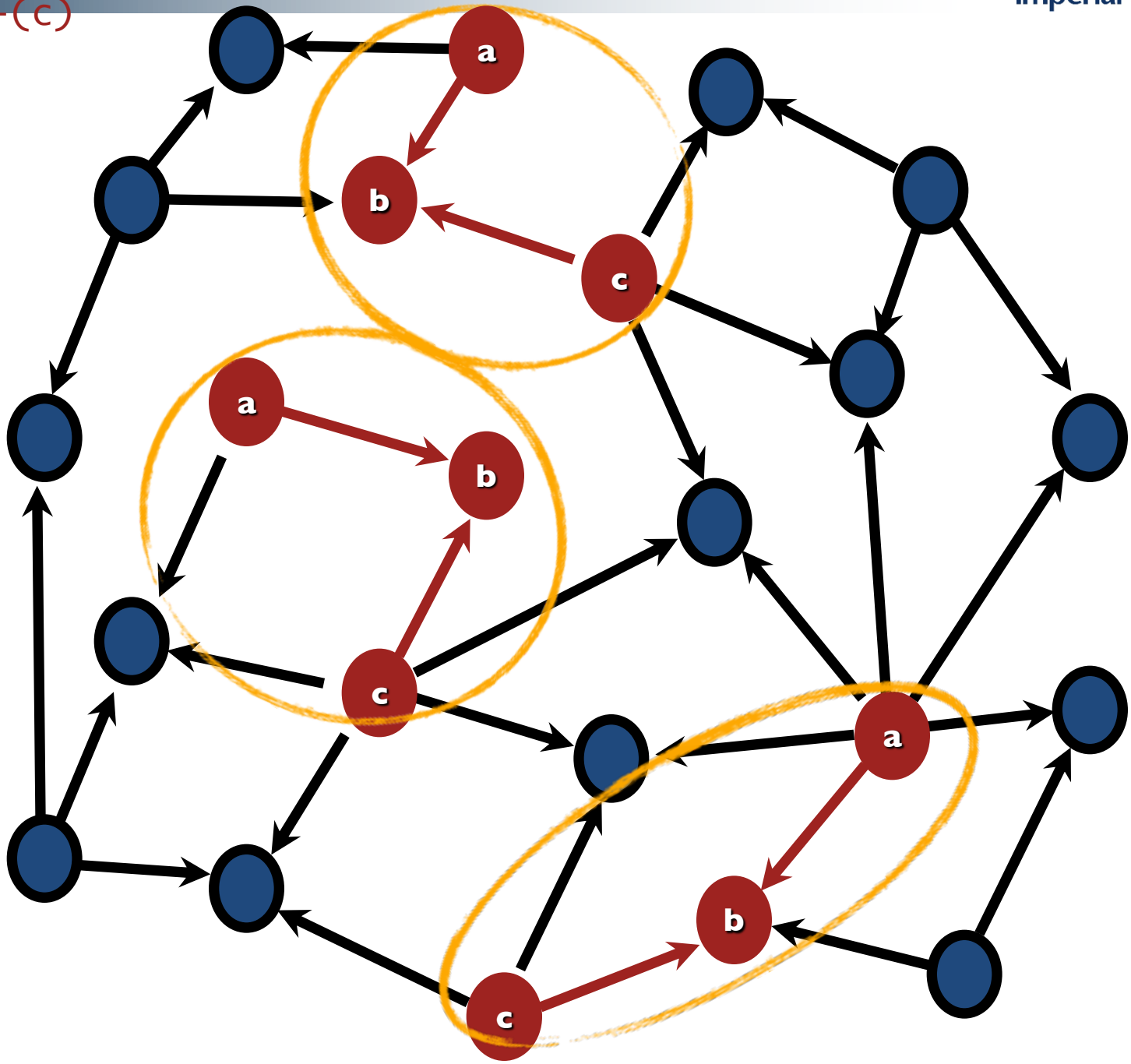



```
(a)-->(b)-->(c)
```

# Paths




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


a
b
c
a
b
c
a
b
c

# Paths

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name;
```

| a.name | m.title | d.name |
|---|---|---|
| “Keanu Reeves” | “The Matrix” | “Andy Wachowski” |
| “Keanu Reeves” | “The Matrix Reloaded” | “Andy Wachowski” |
| “Noah Wyle” | “A Few Good Men” | “Rob Reiner” |
| “Tom Hanks” | “Cloud Atlas” | “Andy Wachowski” |
| ... | ... | ... |

# Paths

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m), (m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name;
```


```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m),
(d)-[:DIRECTED]->(m)
RETURN a.name, m.title, d.name;
```

# Sort & Limit

```
START a=node(*)MATCH (a)-[:ACTED_IN]->(m)<-
[:DIRECTED]-(d)RETURN a.name, d.name,
count(*) AS count
ORDER BY(count) DESC
LIMIT 5;
```

# Aggregation

`count(x)` - add up the number of occurrences

`min(x)` - get the lowest value

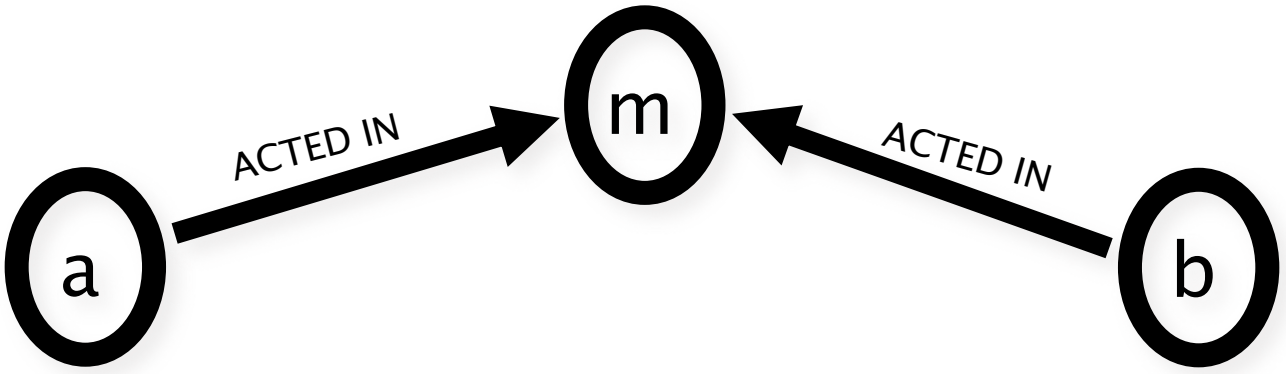
`max(x)` - get the highest value

`avg(x)` - get the average of a numeric value

`collect(x)` - collected all the occurrences into an array

# Aggregation

```
START a=node(*)
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, d.name, collect(m.title);
```



# Find a specific node (all-node query)

```
START n=node(*)
WHERE has(n.name) AND n.name = "Tom Hanks"
RETURN n;
```

WHERE - filter the results

has(n.name) - the name property must exist

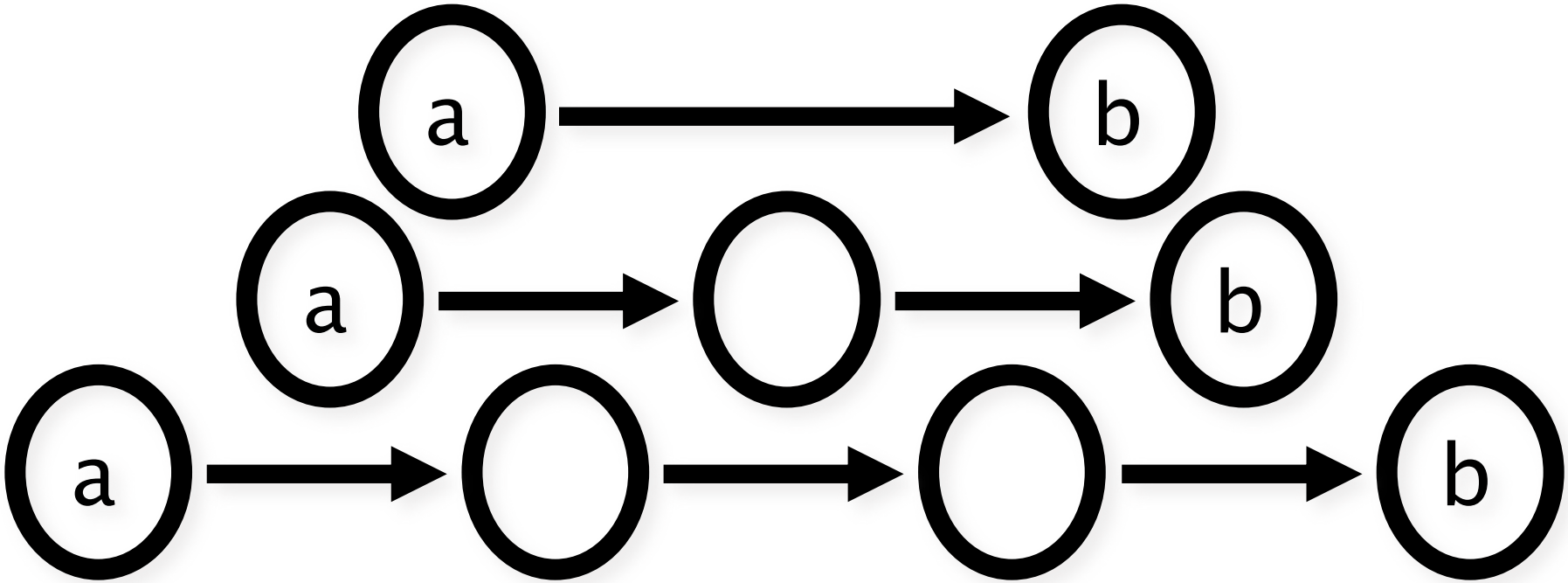
n.name = "Tom Hanks" - and have that value

# Matching multiple relationships

```
START a=node(*)
MATCH (a)-[:ACTED_IN|DIRECTED]->()<-
[:ACTED_IN|DIRECTED]-(b)
CREATE UNIQUE (a)-[:KNOWS]->(b);
```

*(Create KNOWS relationships between anyone, Actors or Directors, who worked together)*

# Variable length paths




```
(a)-[*1..3]->(b)
```

# Friends-of-Friends

```
MATCH (keanu)-[:KNOWS*2]->(fof)
WHERE keanu.name = “Keanu Reeves”
RETURN DISTINCT fof.name;
```

# Conditions

# Constraints on properties

```
MATCH (actor)-[r:ACTED_IN]->(movie)
WHERE "Neo" IN r.roles AND actor.name = “Keanu
Reeves”
RETURN DISTINCT movie.title;
```

*(Movies in which Keanu Reeves played Neo)*

# Constraints based on patterns

```
MATCH (gene)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(n)
WHERE gene.name = ”Gene Hackman”
RETURN DISTINCT n.name;
```

*(Actors who worked with Gene Hackman)*

# Constraints based on patterns

```
MATCH (gene)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(n)
WHERE (n)-[:DIRECTED]->() AND gene.name = ”Gene
Hackman”

RETURN DISTINCT n.name;
```

*(Actors who worked with Gene and were directors)*

# Updating Graphs with Cypher

# Creating nodes

```
CREATE ({title:"Mystic River", released:1993});
```

```
START a=node(*)
MATCH (a)
WHERE a.title = “Mystic River”
RETURN a
```

# Adding properties

```
START a=node(*)
MATCH (a)
WHERE a.title = “Mystic River”
SET movie.tagline = "We bury our sins here, Dave. We wash them
clean."
RETURN movie;
```

# Creating Relationships

```
CREATE UNIQUE (kevin)-[:ACTED_IN {roles:["Sean"]}]->(movie)
WHERE movie.title = “Mystic River” AND kevin.name = “Kevin Bacon”
```

```
MATCH (kevin)-[:ACTED_IN]->(movie)
WHERE kevin.name = “Kevin Bacon”
RETURN DISTINCT movie.title;
```

# Deleting relationships

```
MATCH (emil)-[r]->()
WHERE emil.name = “Emil Eifrem”
DELETE r;
```

# Deleting nodes and relationships

```
MATCH (emil)-[r]->()
WHERE emil.name = “Emil Eifrem”
DELETE r, emil;
```

# Deleting nodes and relationships

```
MATCH (emil)-[r?]->()
WHERE emil.name = “Emil Eifrem”
DELETE r, emil;
```