

# Neo4j Tutorial

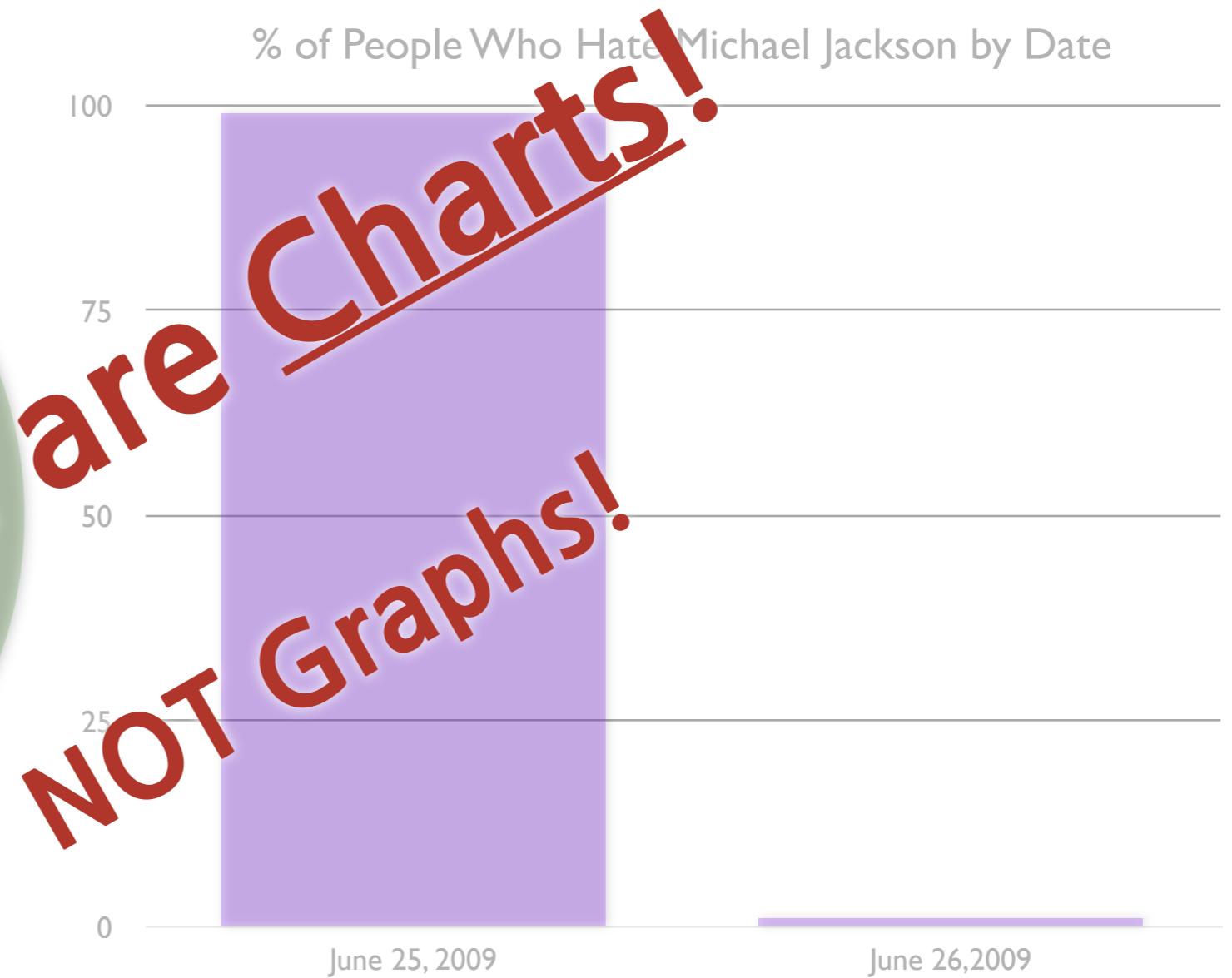
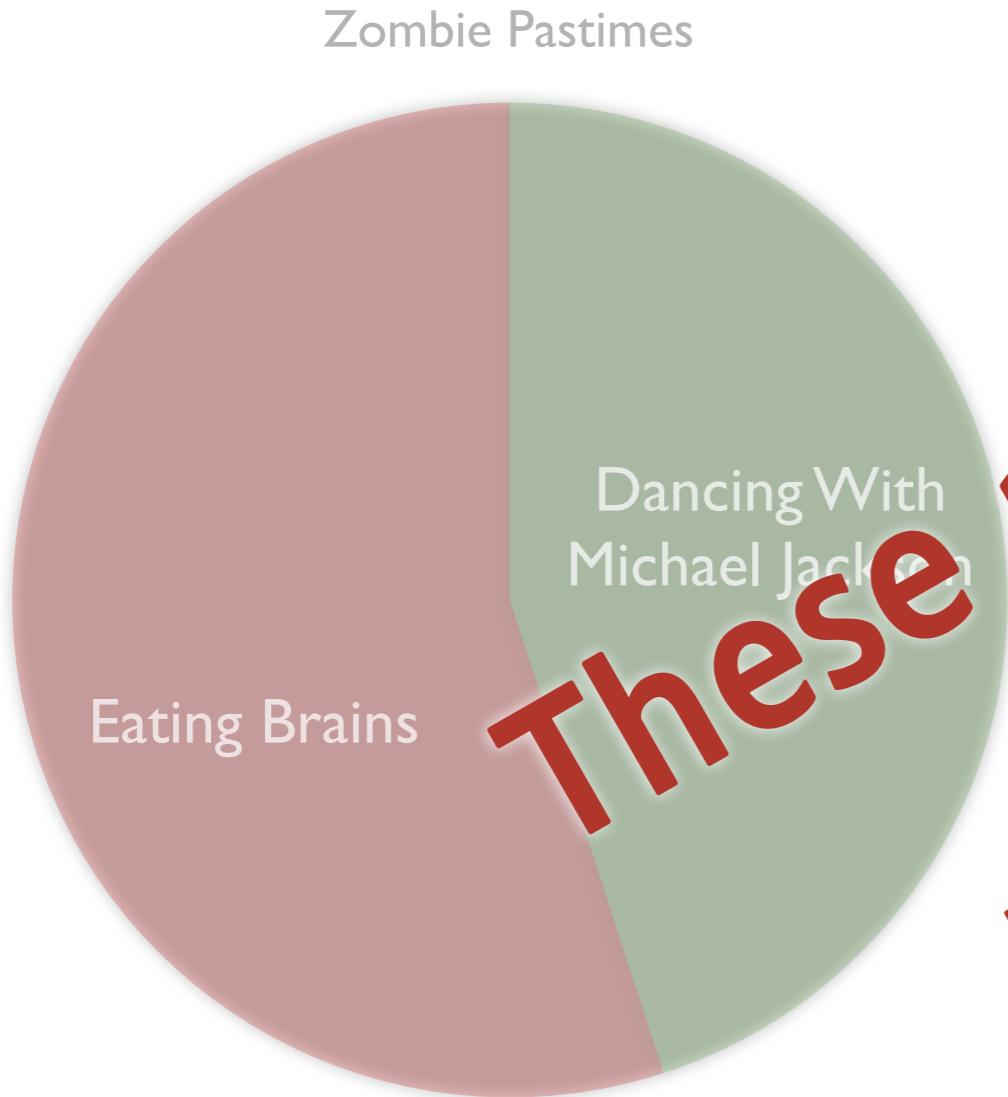
(you) - [:know] ->(Neo4j )





# (Introduction to Graphs)

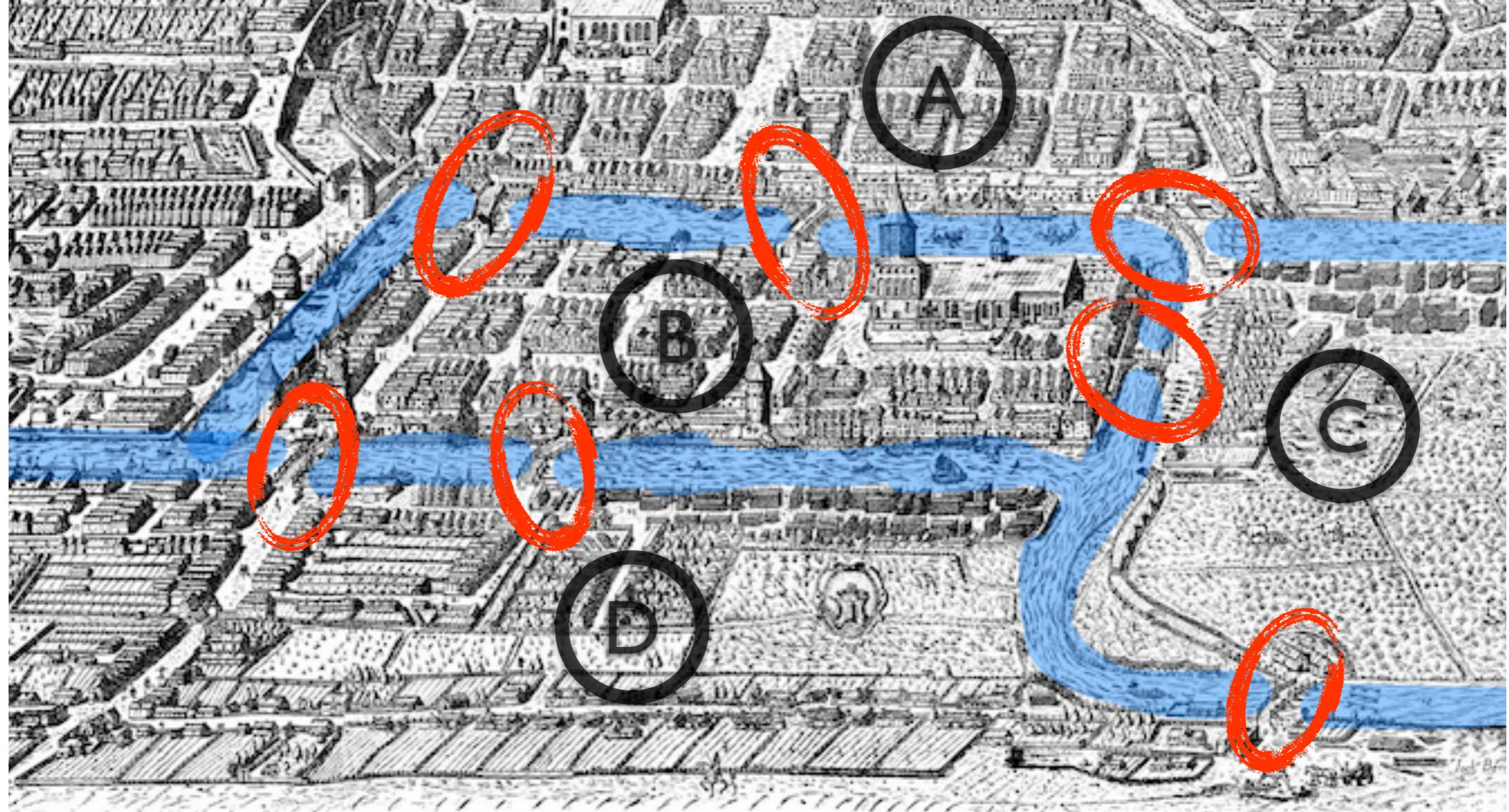
**But first...  
an important message...**

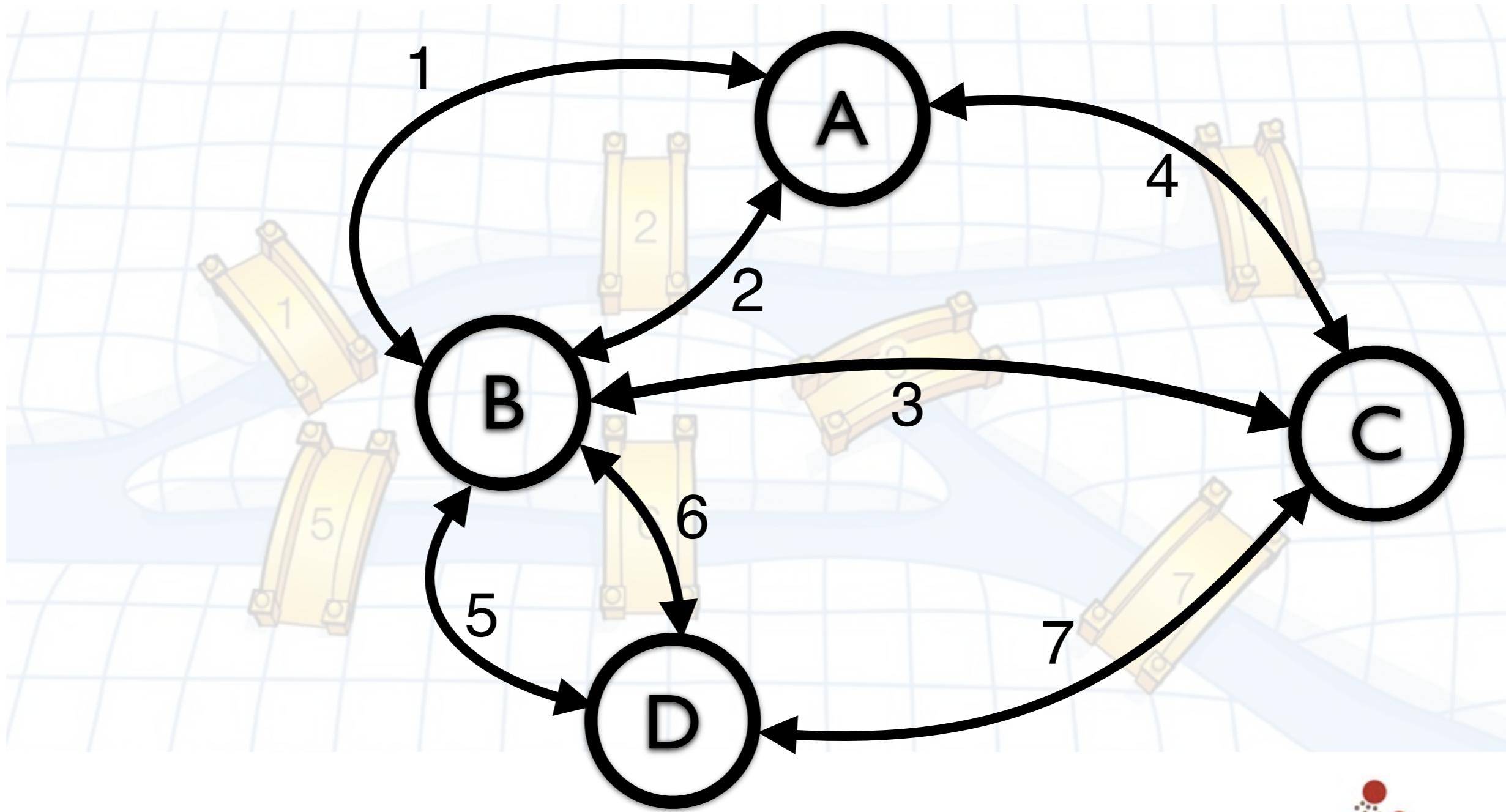




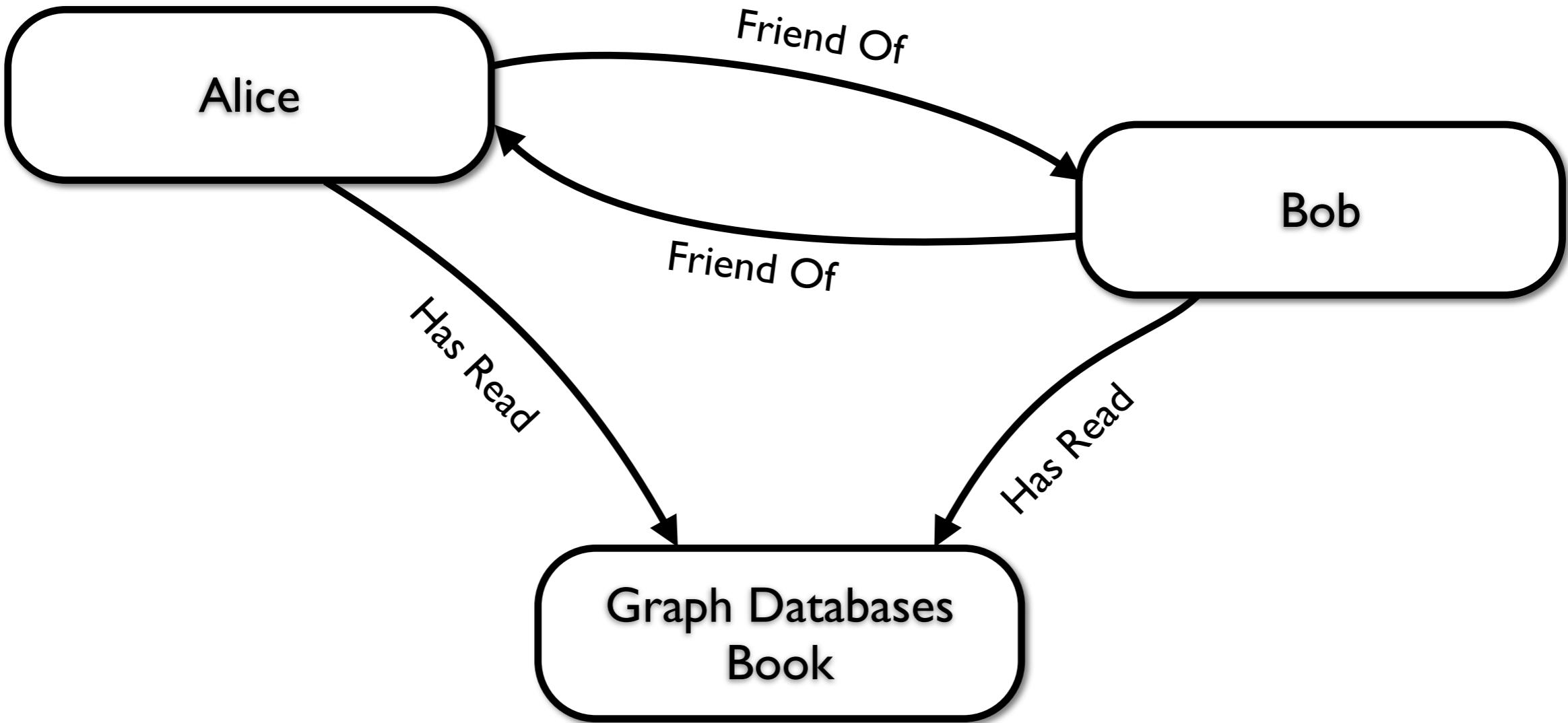
# Leonhard Euler 1707-1783

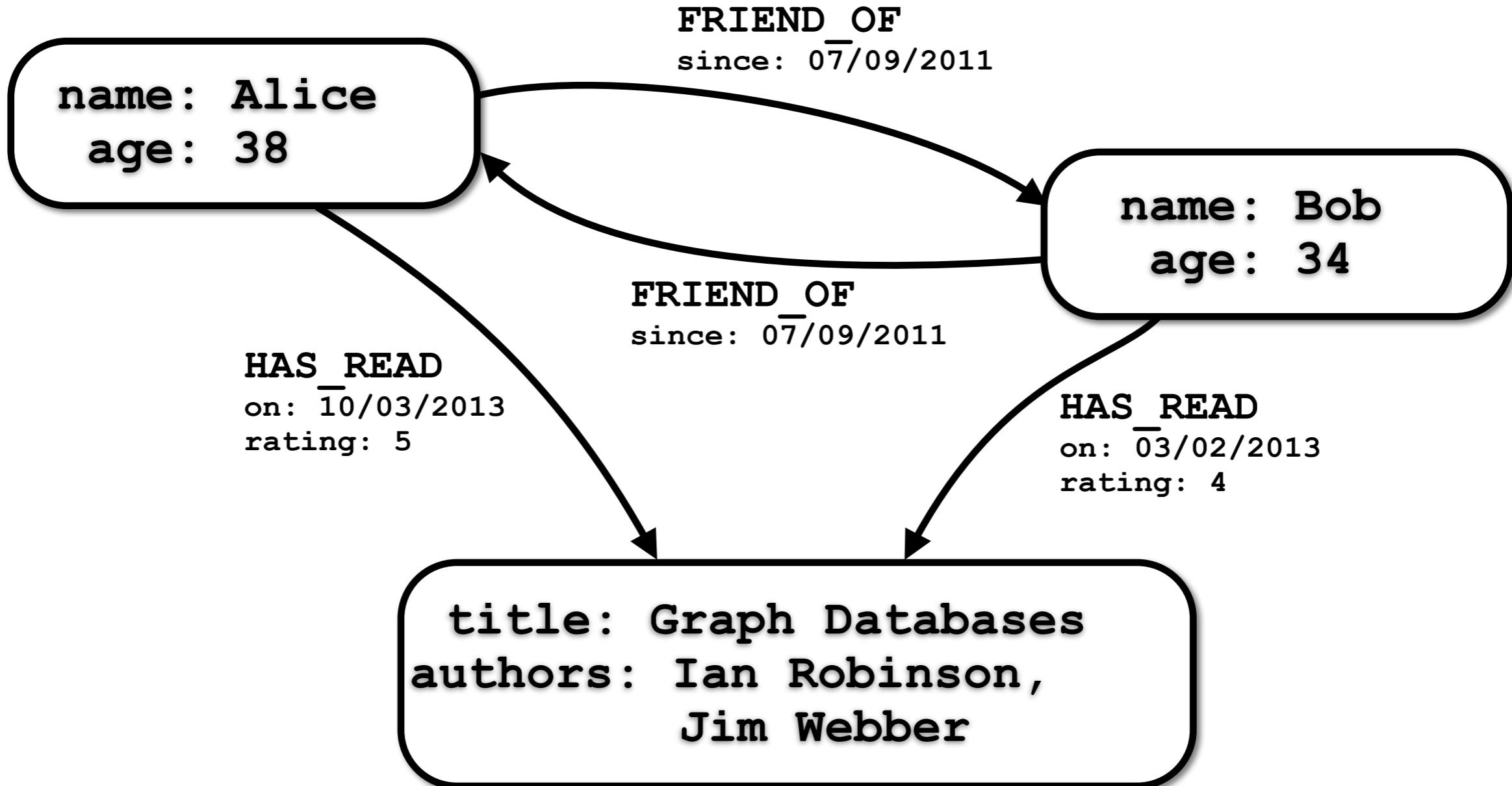
# Königsberg (Prussia) - 1736





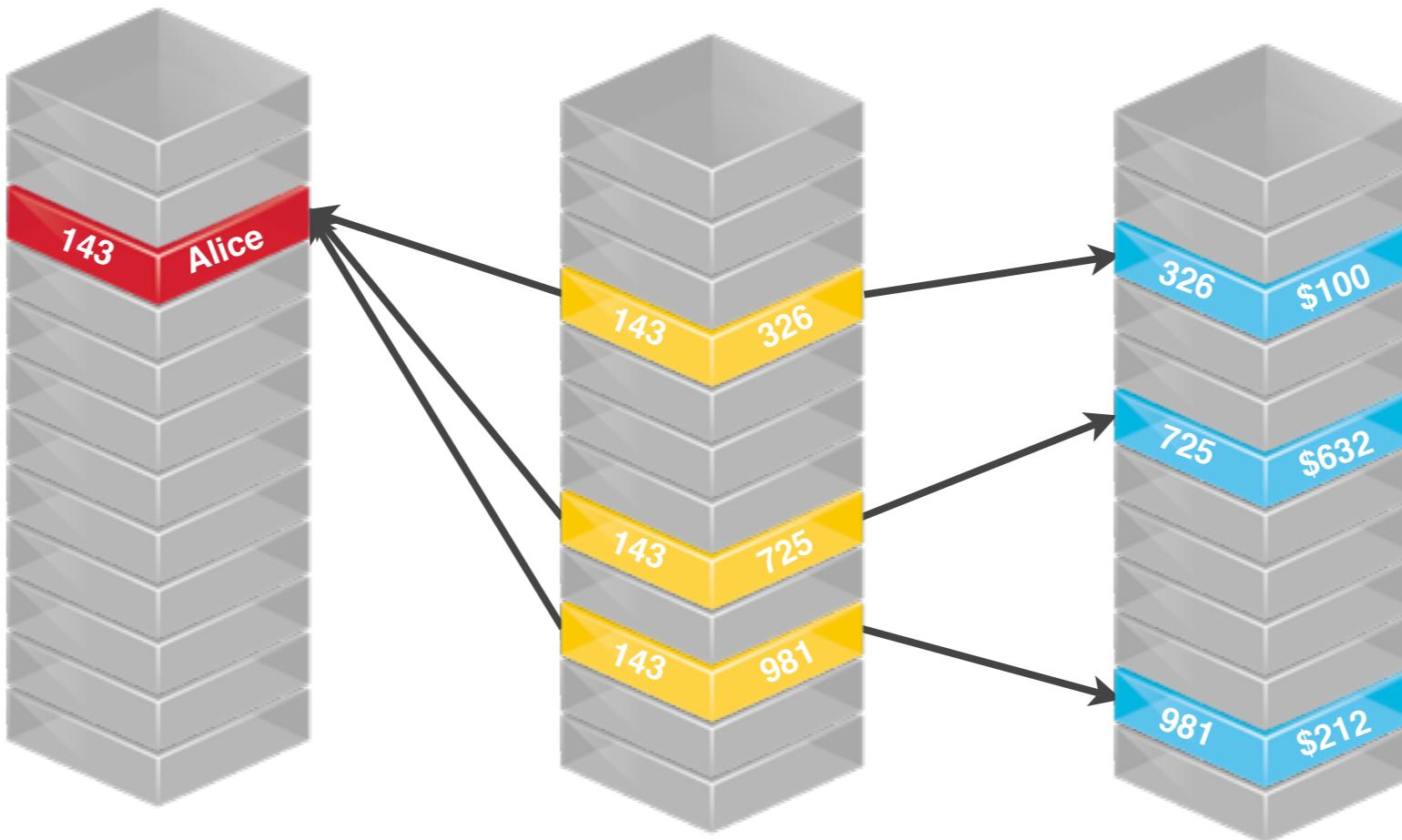
# So what about data?







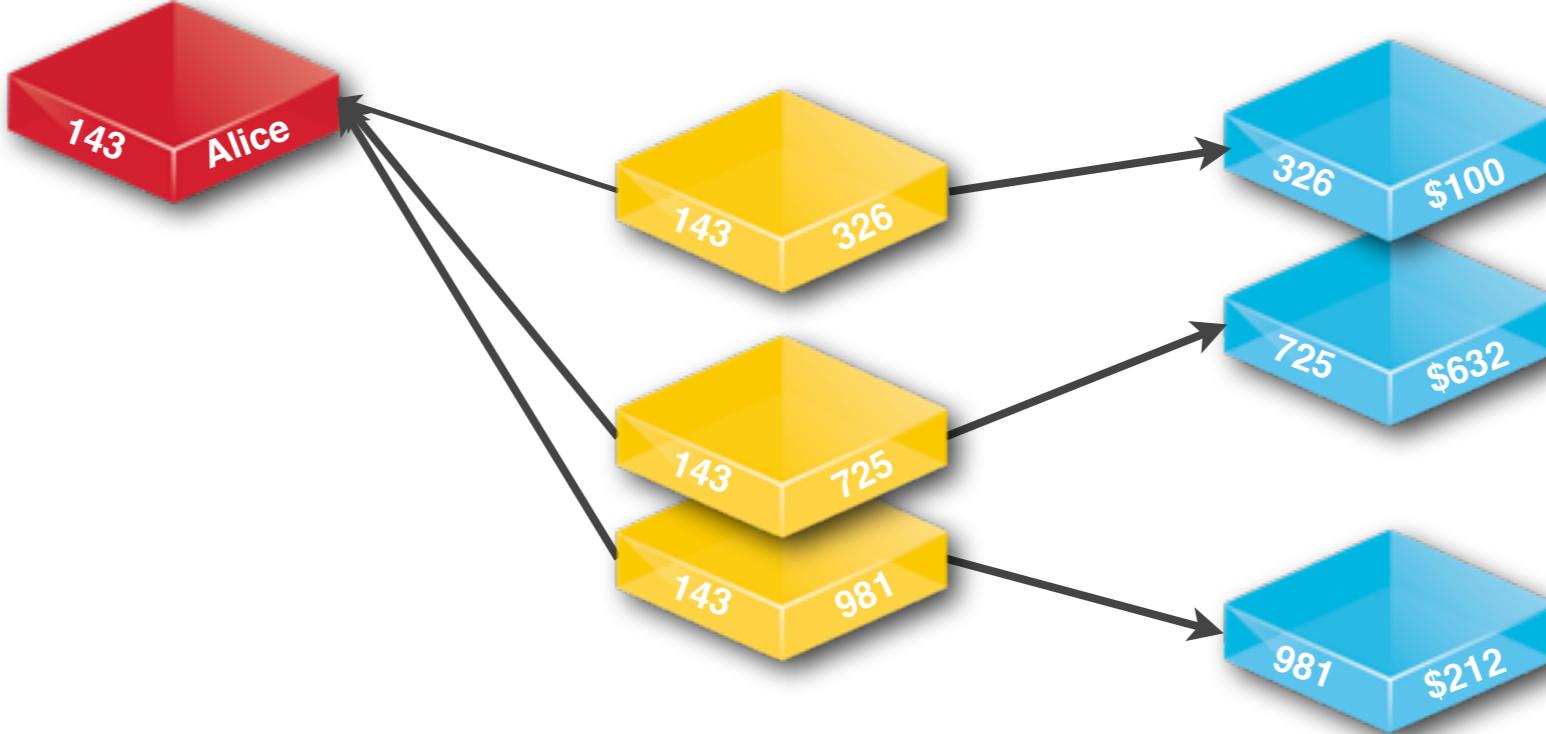
# Segue: Graphs in a Relational World

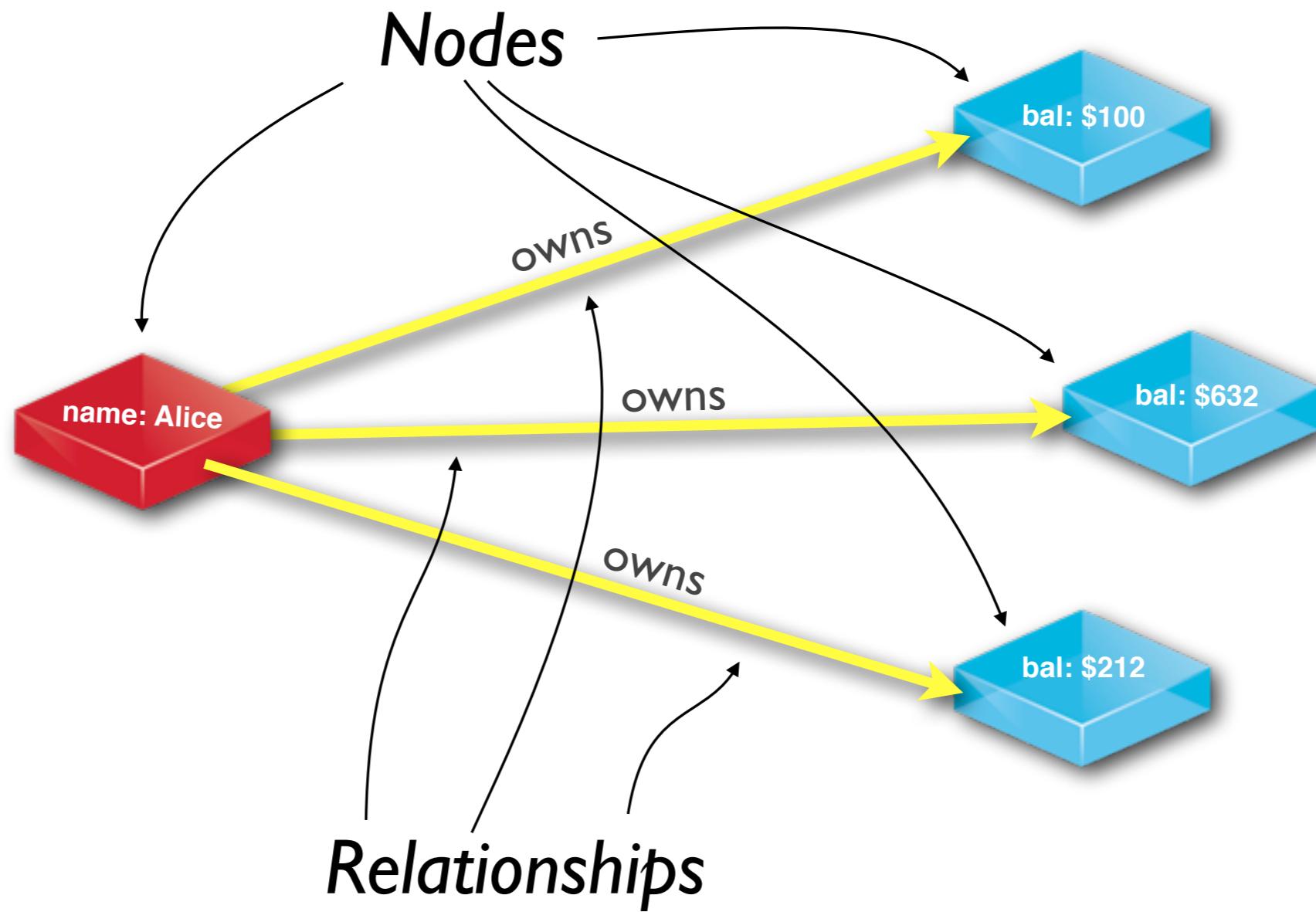


**Customers**

**Customer\_Accounts**

**Accounts**

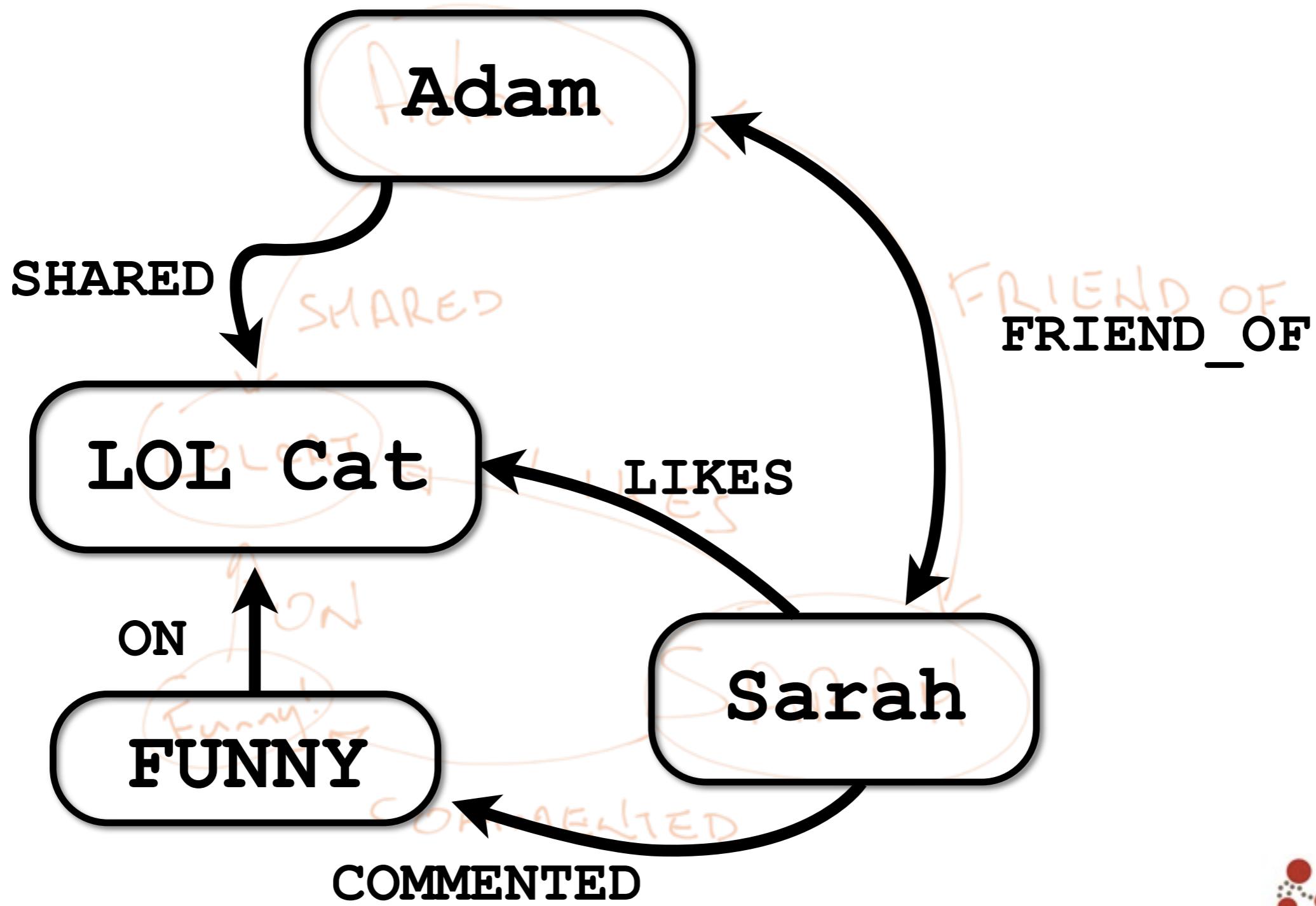






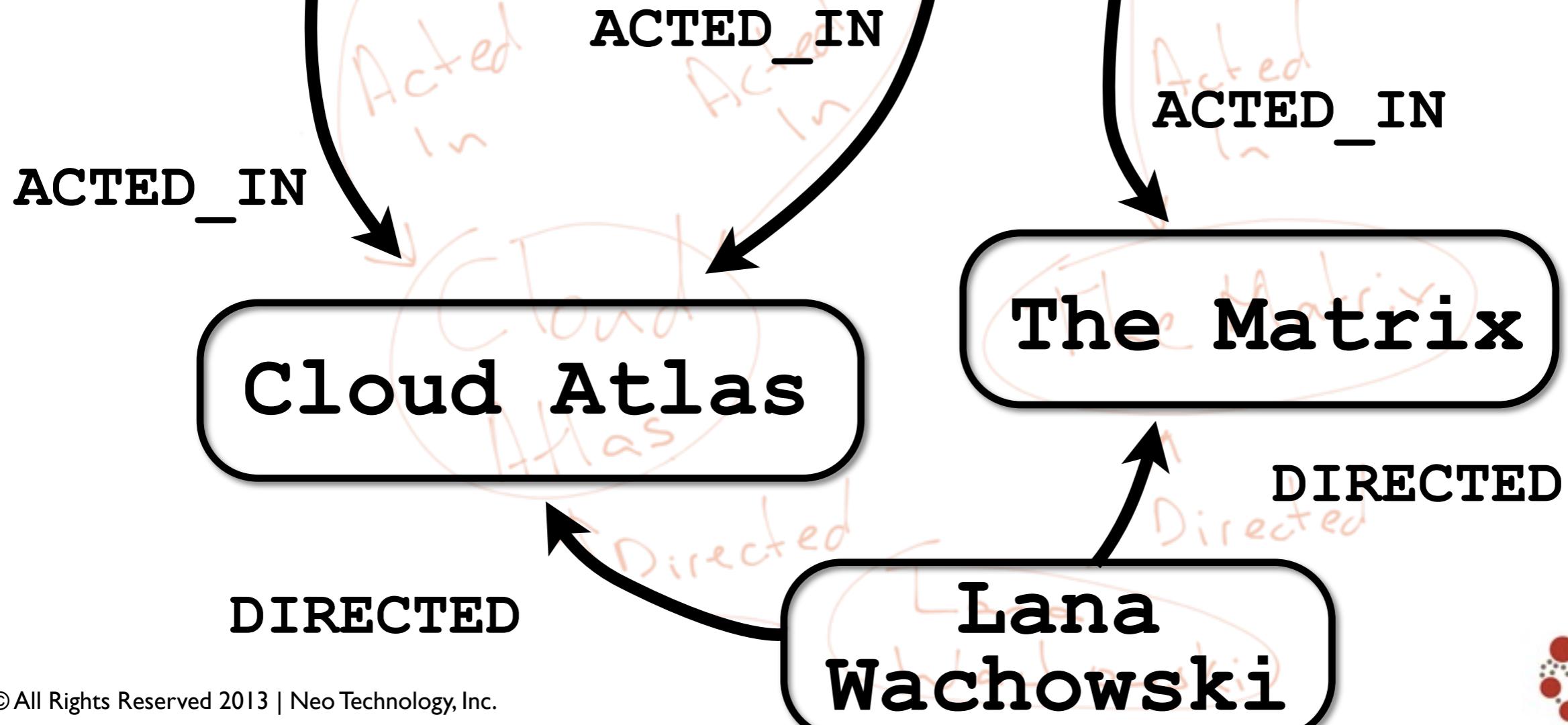
# (Modeling with Graphs)

# Start with a whiteboard



Tom Hanks

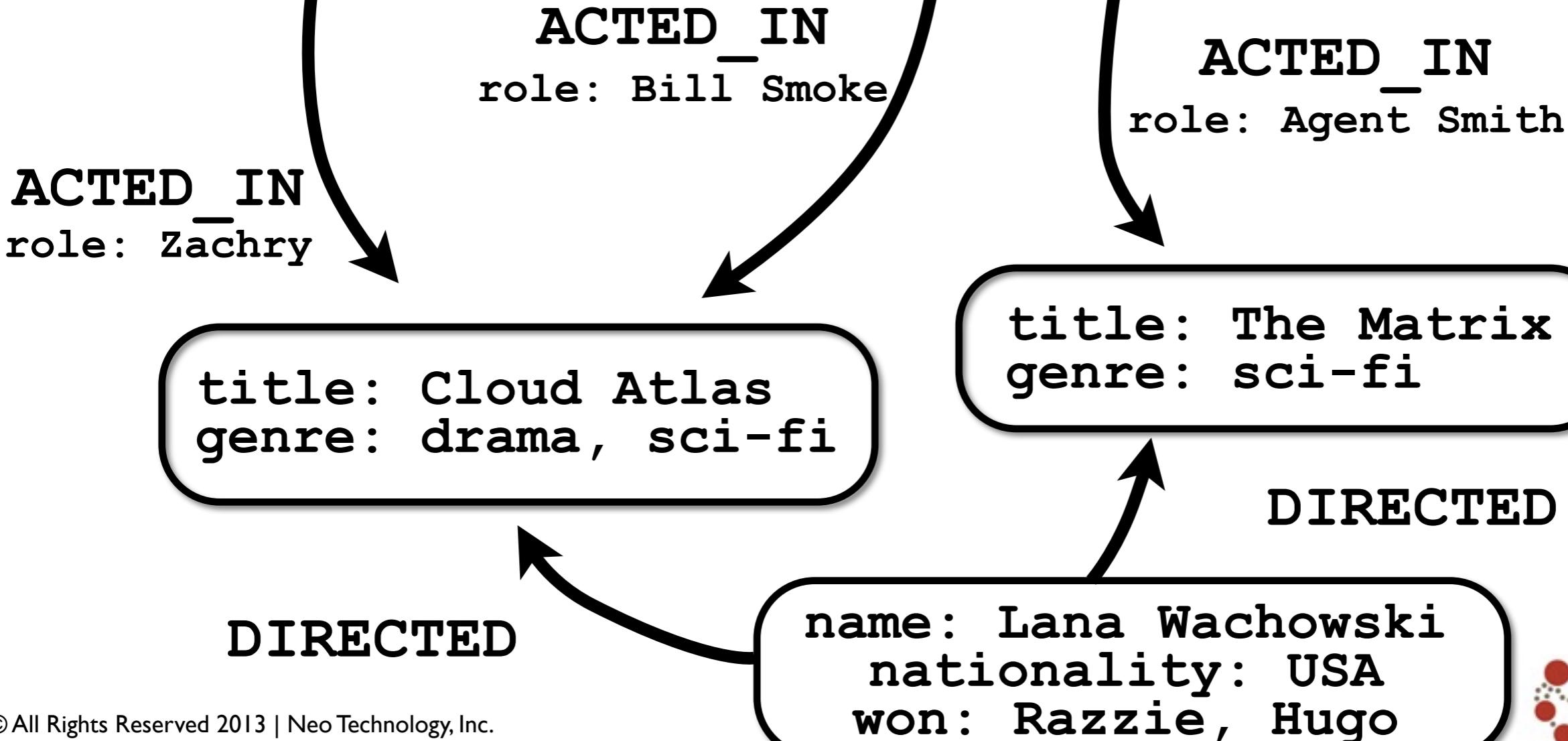
Hugo Weaving



# Model Incrementally

**name:** Tom Hanks  
**nationality:** USA  
**won:** Oscar, Emmy

**name:** Hugo Weaving  
**nationality:** Australia  
**won:** MTV Movie Award





# (Brief tour of Neo4j)

`bin/neo4j`

`neo4j` itself

`bin/neo4j-shell`

a command-line shell

`conf/`

configuration files

`data/`

graph data, logs



## Mac OS X

1

```
./bin/neo4j start
```

2

```
./bin/neo4j-shell < sample/movies.cyp
```

3

```
open http://localhost:7474/
```

## Windows

1

```
bin\neo4j start
```

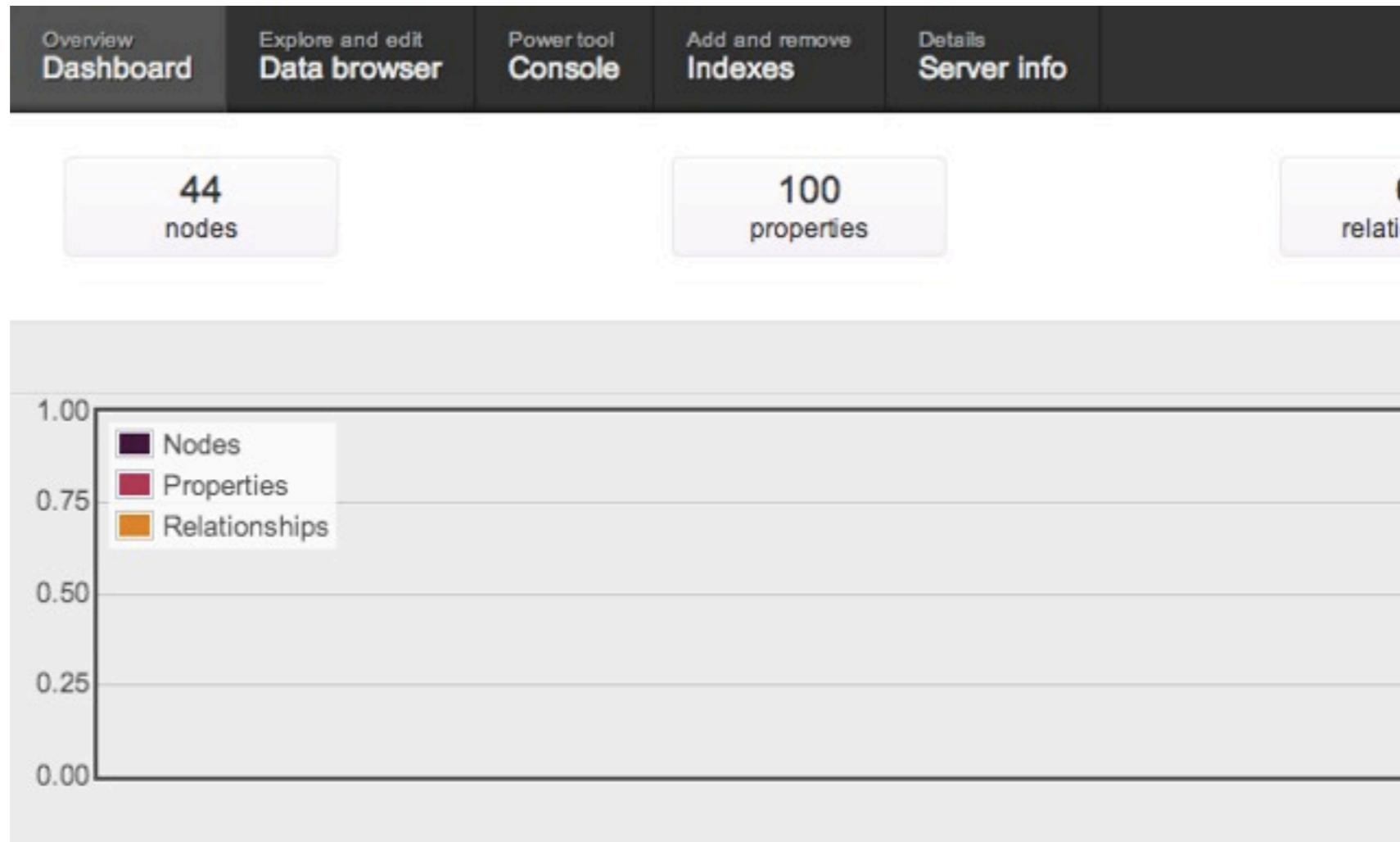
2

```
java -jar gumdrop.jar --file sample\movies.cyp
```

3

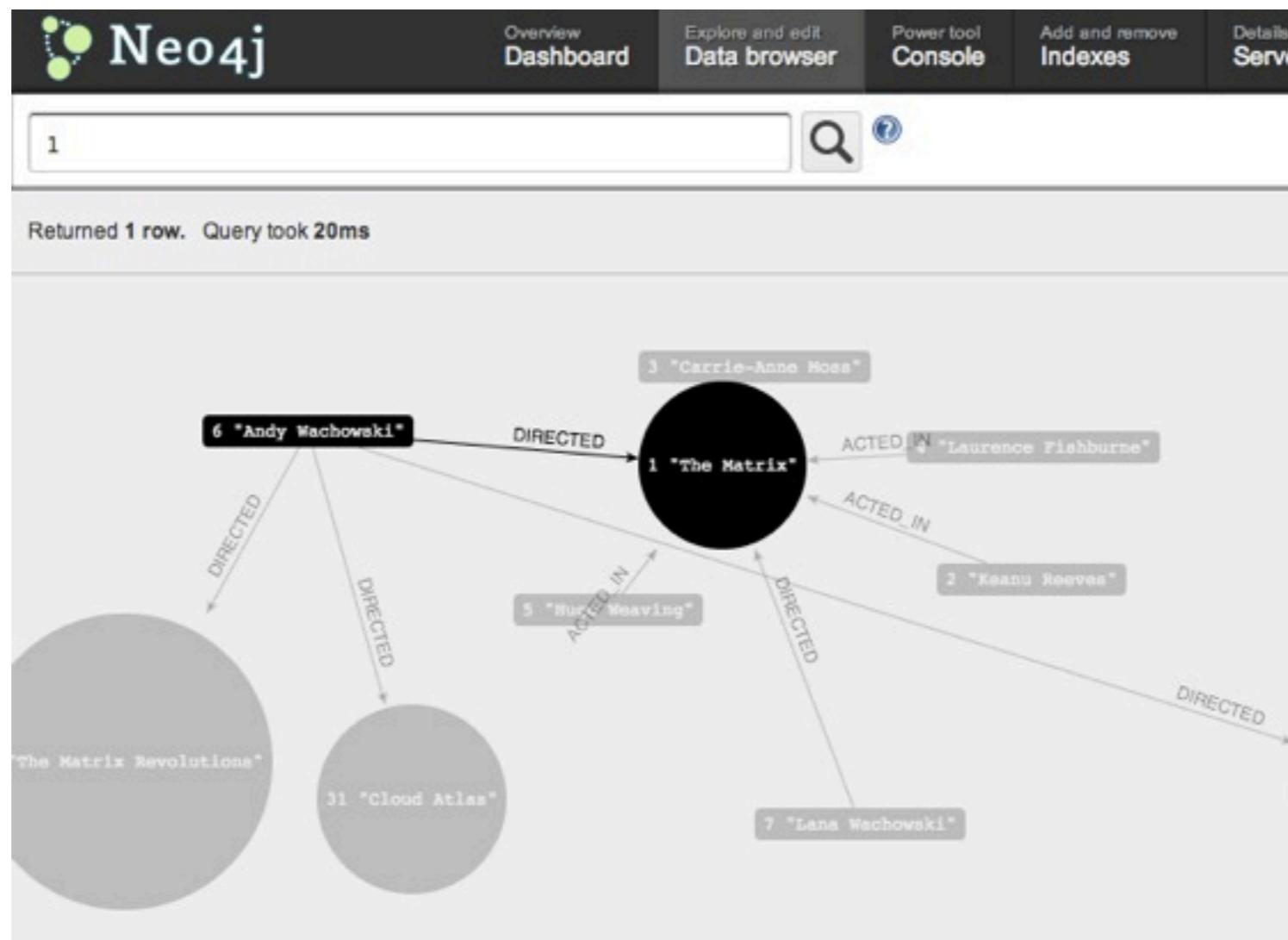
```
start http://localhost:7474/
```

# Neo4j Dashboard - for monitoring of key stats



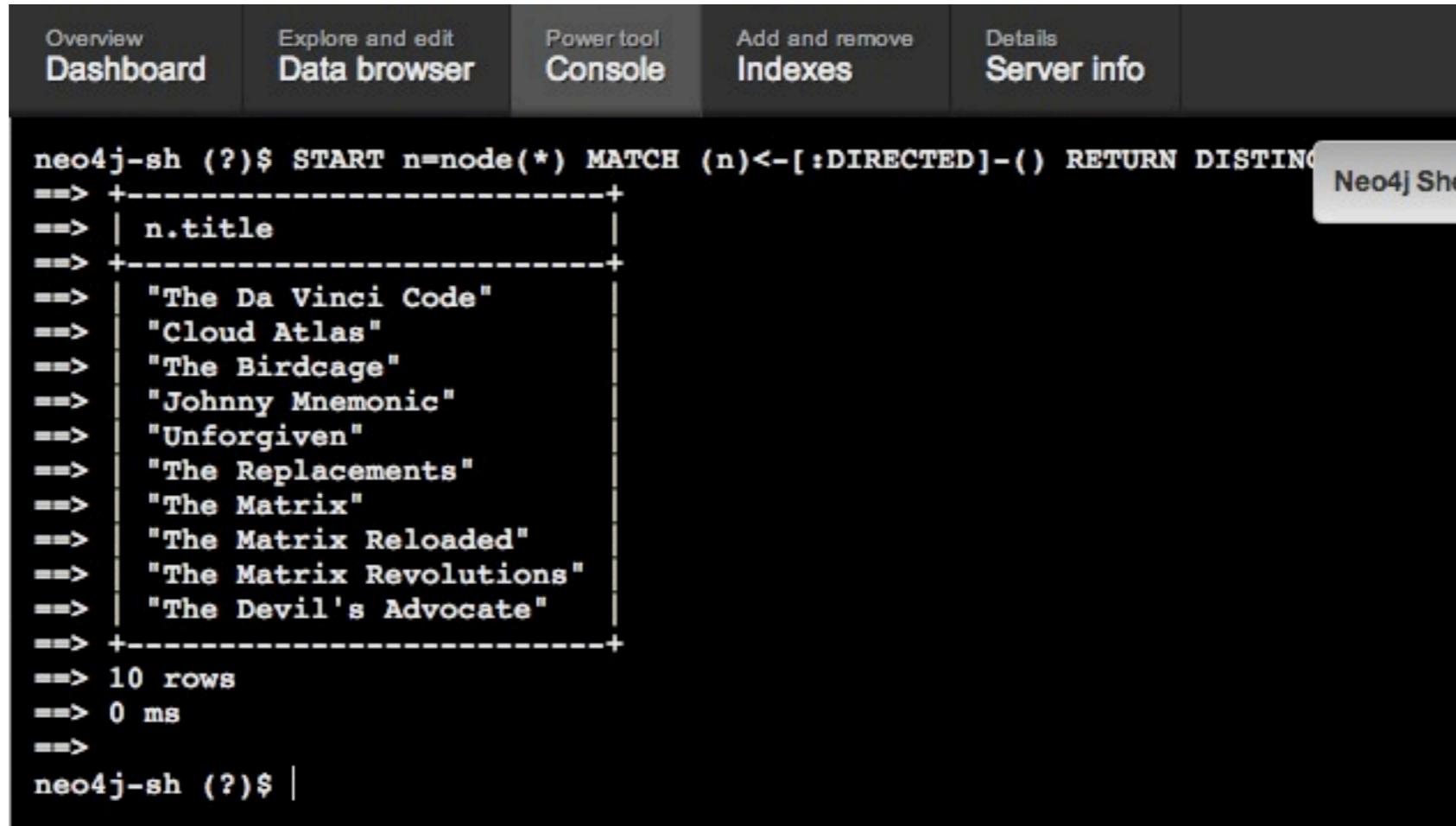
<http://localhost:7474/>

# Neo4j Data browser - to inspect or visualize data



<http://localhost:7474/>

# Neo4j Console - for running commands and queries



The screenshot shows the Neo4j Console interface with a dark theme. The top navigation bar includes links for Overview, Dashboard, Explore and edit Data browser, Power tool Console, Add and remove Indexes, Details Server info, and a dropdown menu. The main area displays a command-line session:

```
neo4j-sh (?)$ START n=node(*) MATCH (n)<-[ :DIRECTED ]-() RETURN DISTINCT n.title
==> +-----+
==> | n.title
==> +-----+
==> | "The Da Vinci Code"
==> | "Cloud Atlas"
==> | "The Birdcage"
==> | "Johnny Mnemonic"
==> | "Unforgiven"
==> | "The Replacements"
==> | "The Matrix"
==> | "The Matrix Reloaded"
==> | "The Matrix Revolutions"
==> | "The Devil's Advocate"
==> +-----+
==> 10 rows
==> 0 ms
==>
neo4j-sh (?)$ |
```

A tooltip labeled "Neo4j Shell" is visible near the top right of the console area.

<http://localhost:7474/>

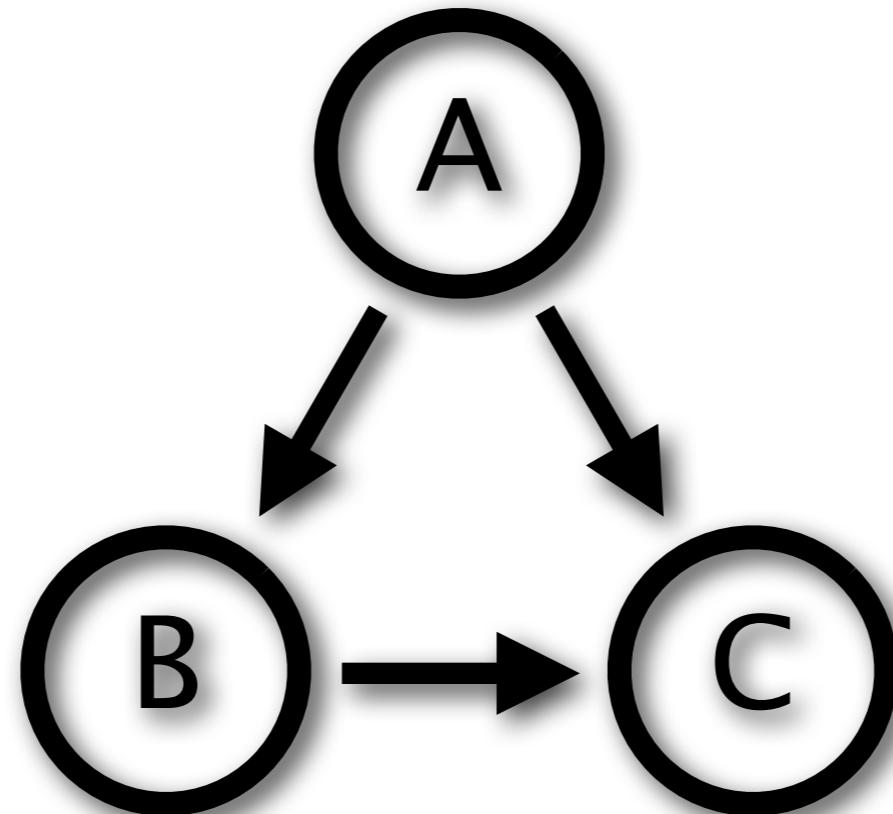


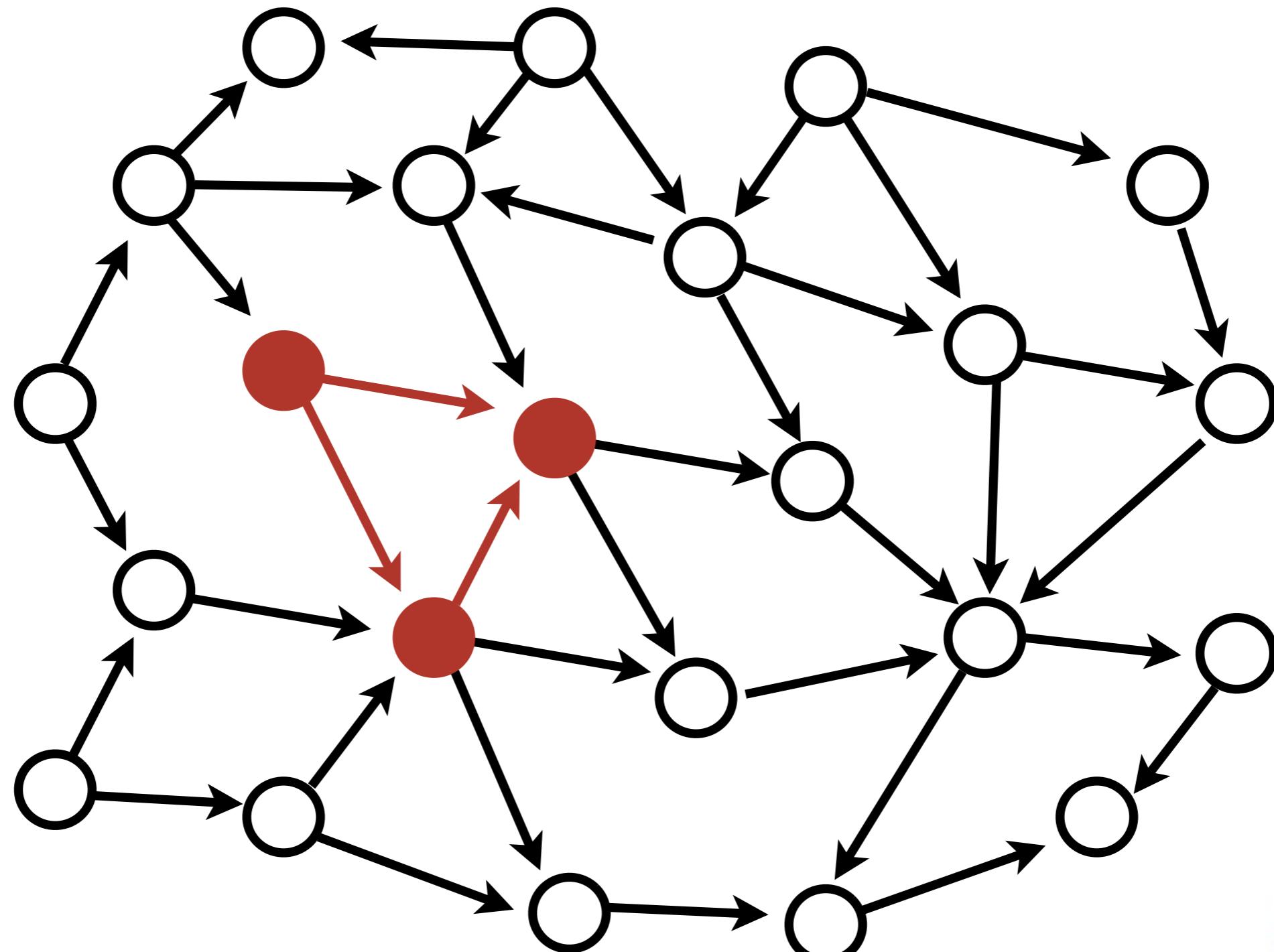
# (Introduction to Cypher)

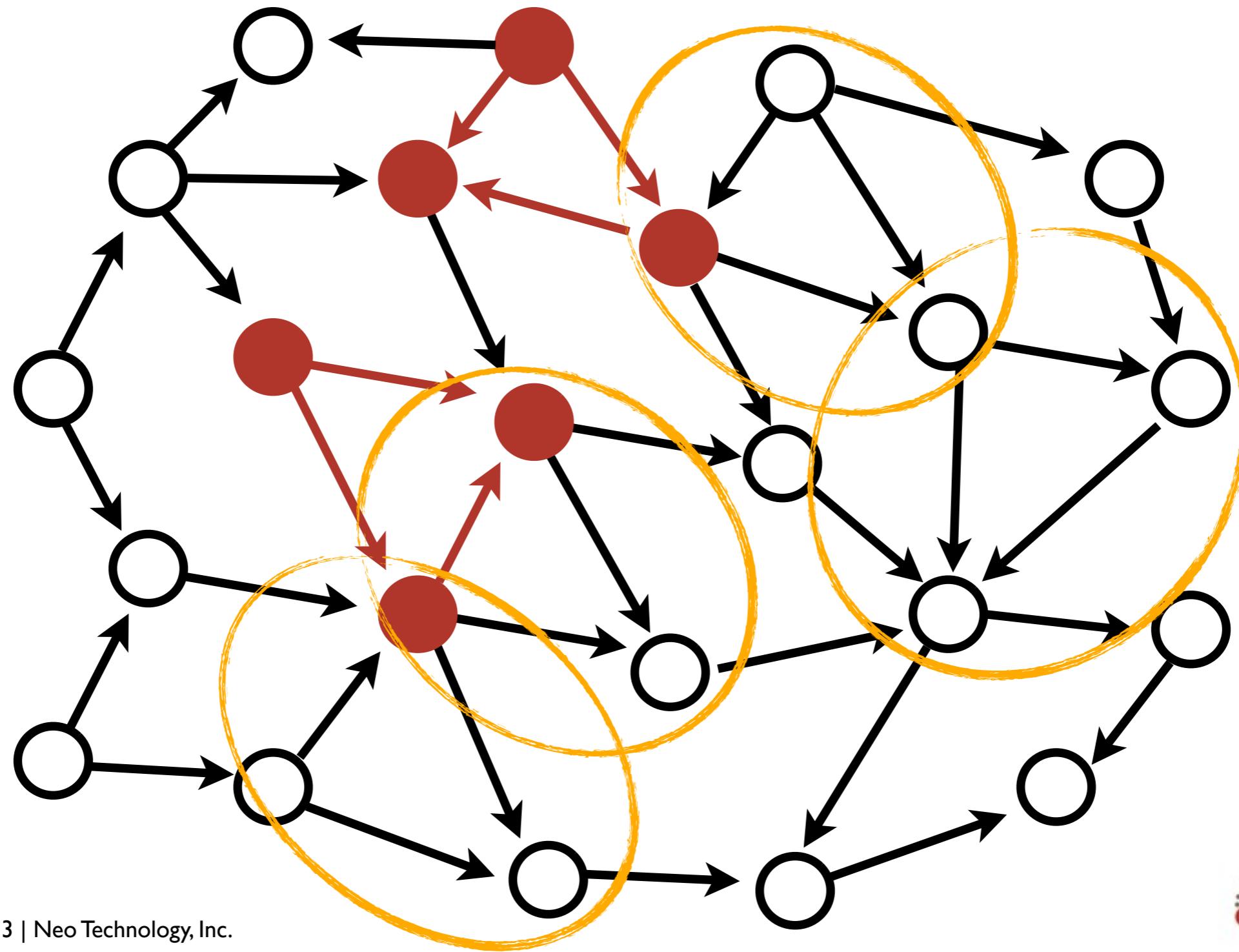
# Cypher is Neo4j's graph query language

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

# It's all about Patterns

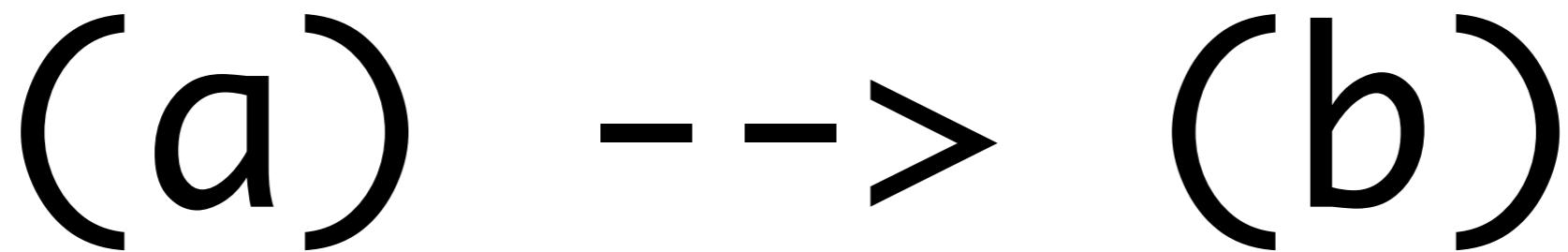






# How?

# Two nodes, one relationship

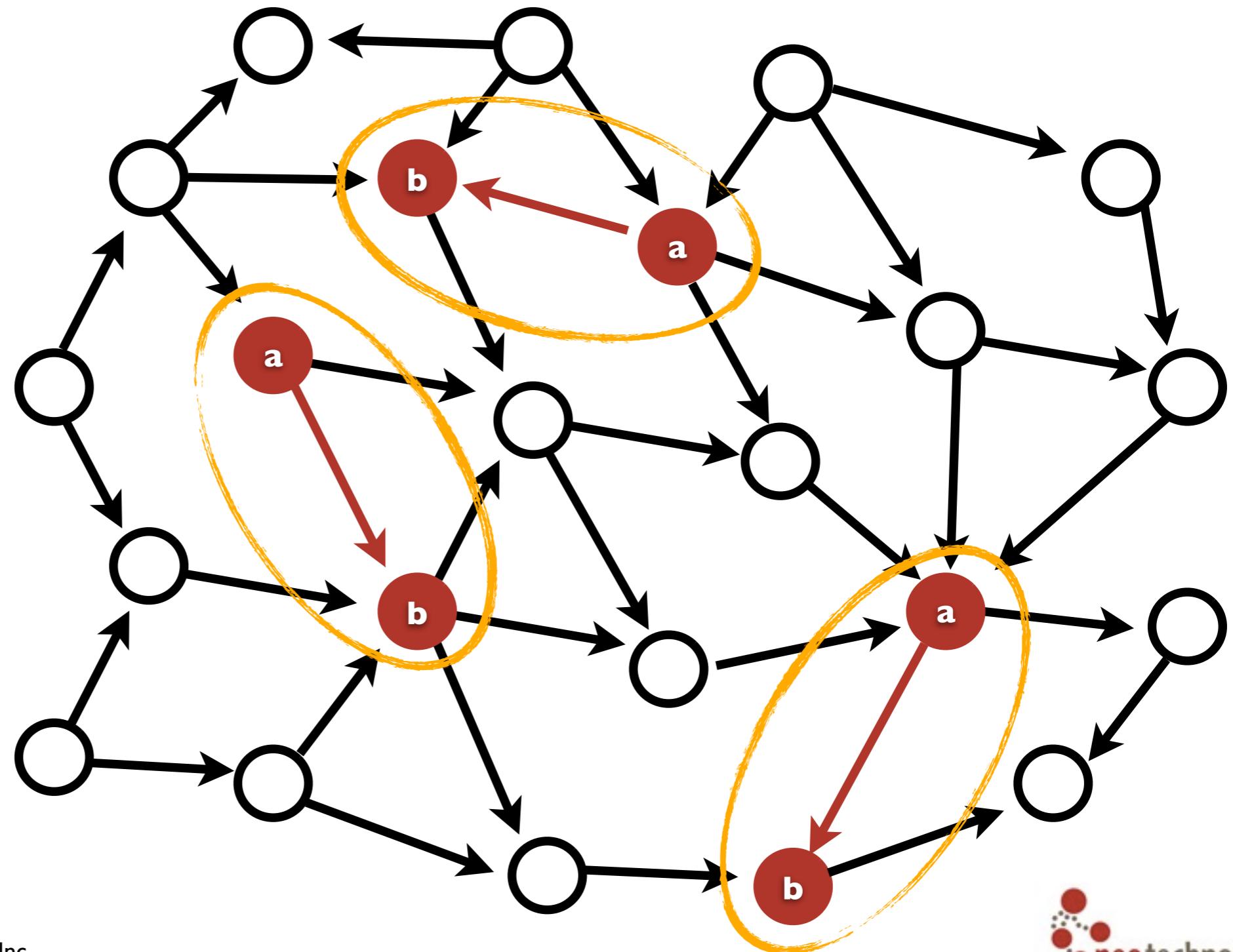


# Two nodes, one relationship

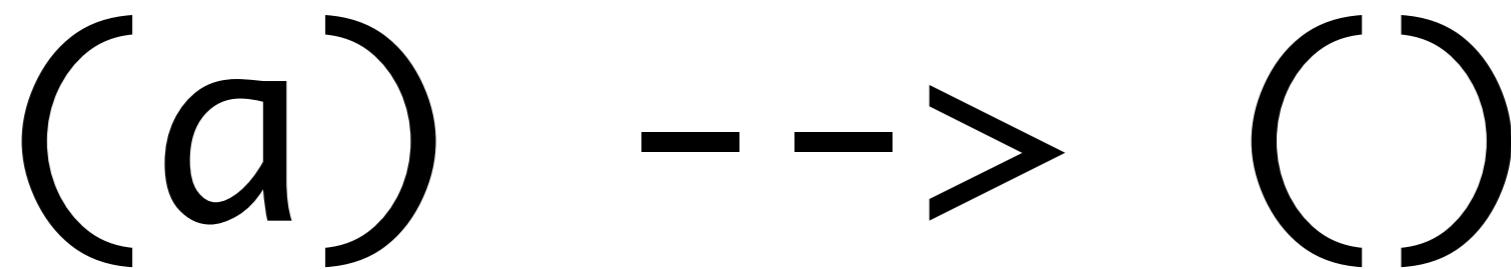
```
START a=node(*)  
MATCH (a) -.->(b)  
RETURN a, b;
```



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



# Two nodes, one relationship



# Two nodes, one relationship

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

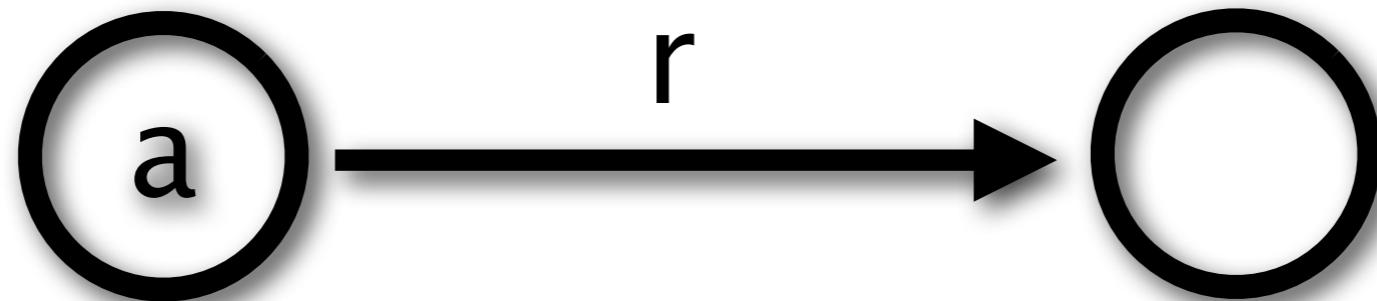


# Two nodes, one relationship

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



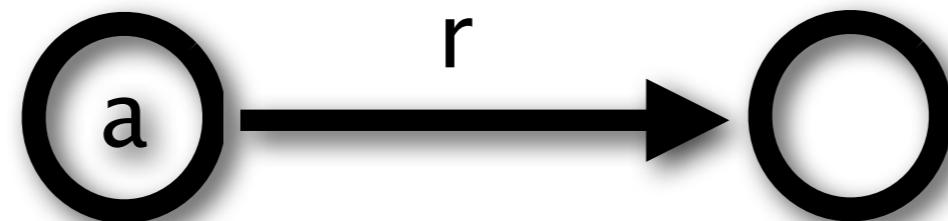
# Two nodes, one relationship



(a) - [r] -> ()

# Two nodes, one relationship

```
START a=node(*)  
MATCH (a)-[r]->()  
RETURN a.name, type(r);
```



# Two nodes, one relationship



(a) -[:ACTED\_IN]-> (m)

# Two nodes, one relationship

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

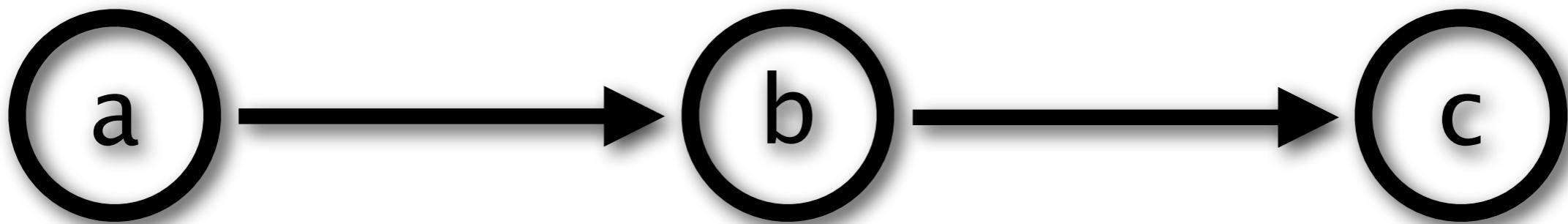


# Two nodes, one relationship

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

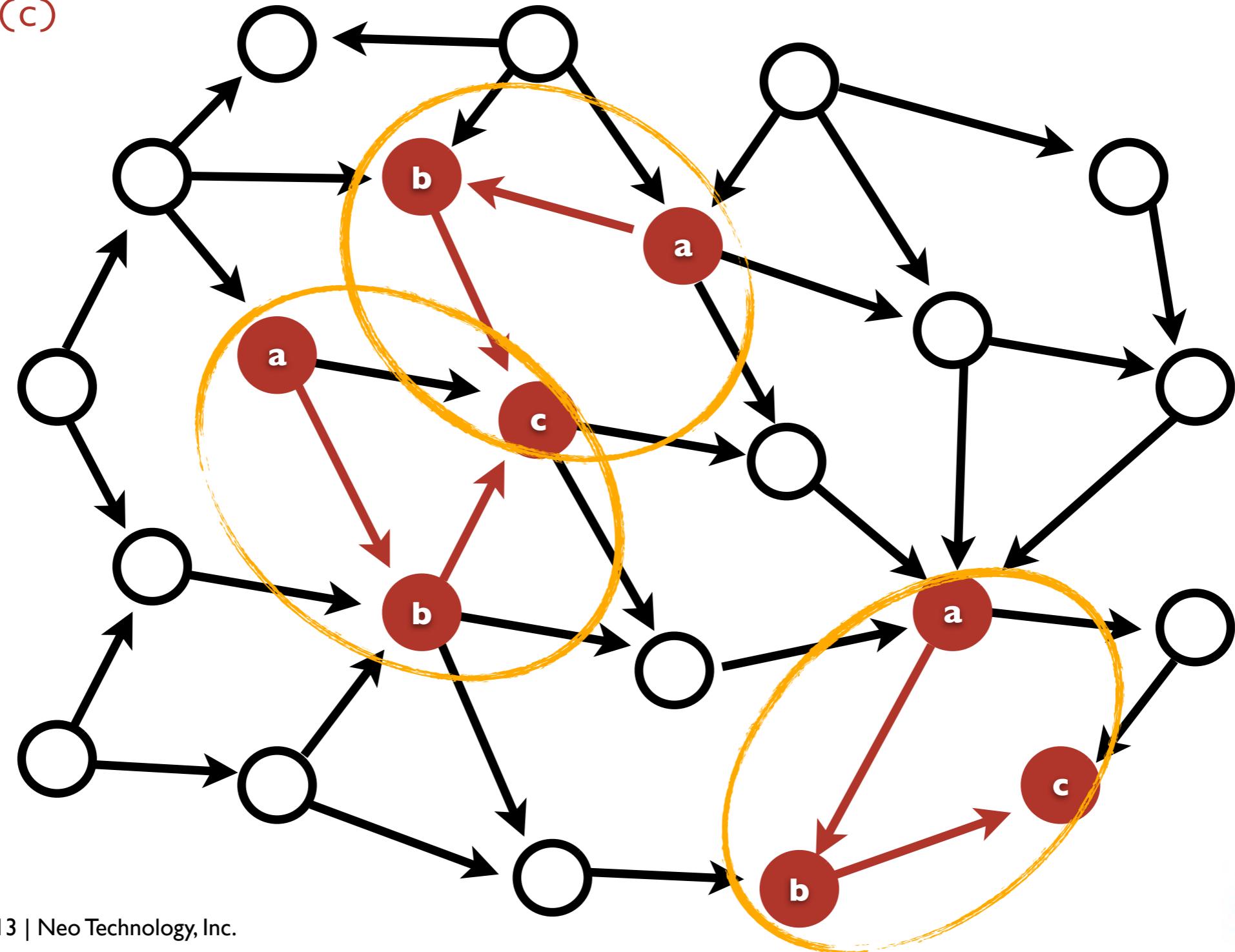


# Paths

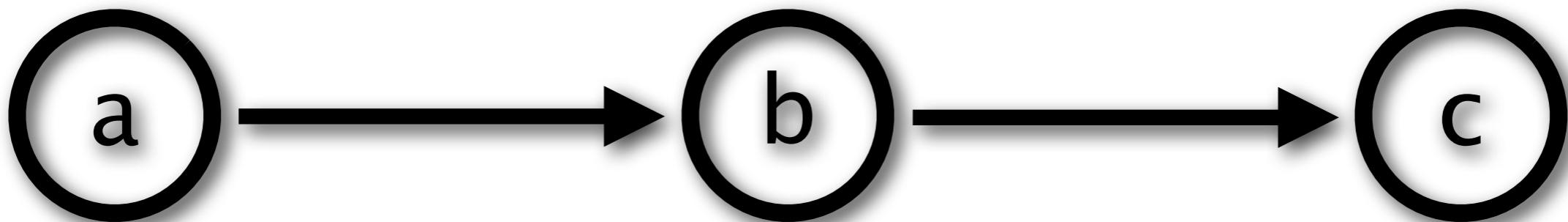


(a) - -> (b) - -> (c)

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

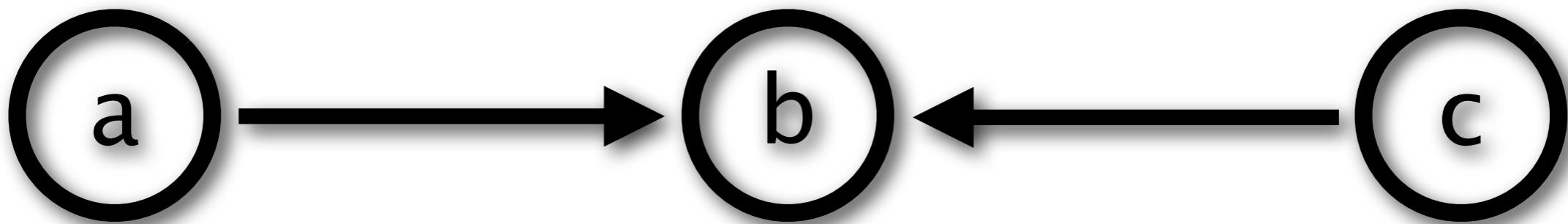


# Paths



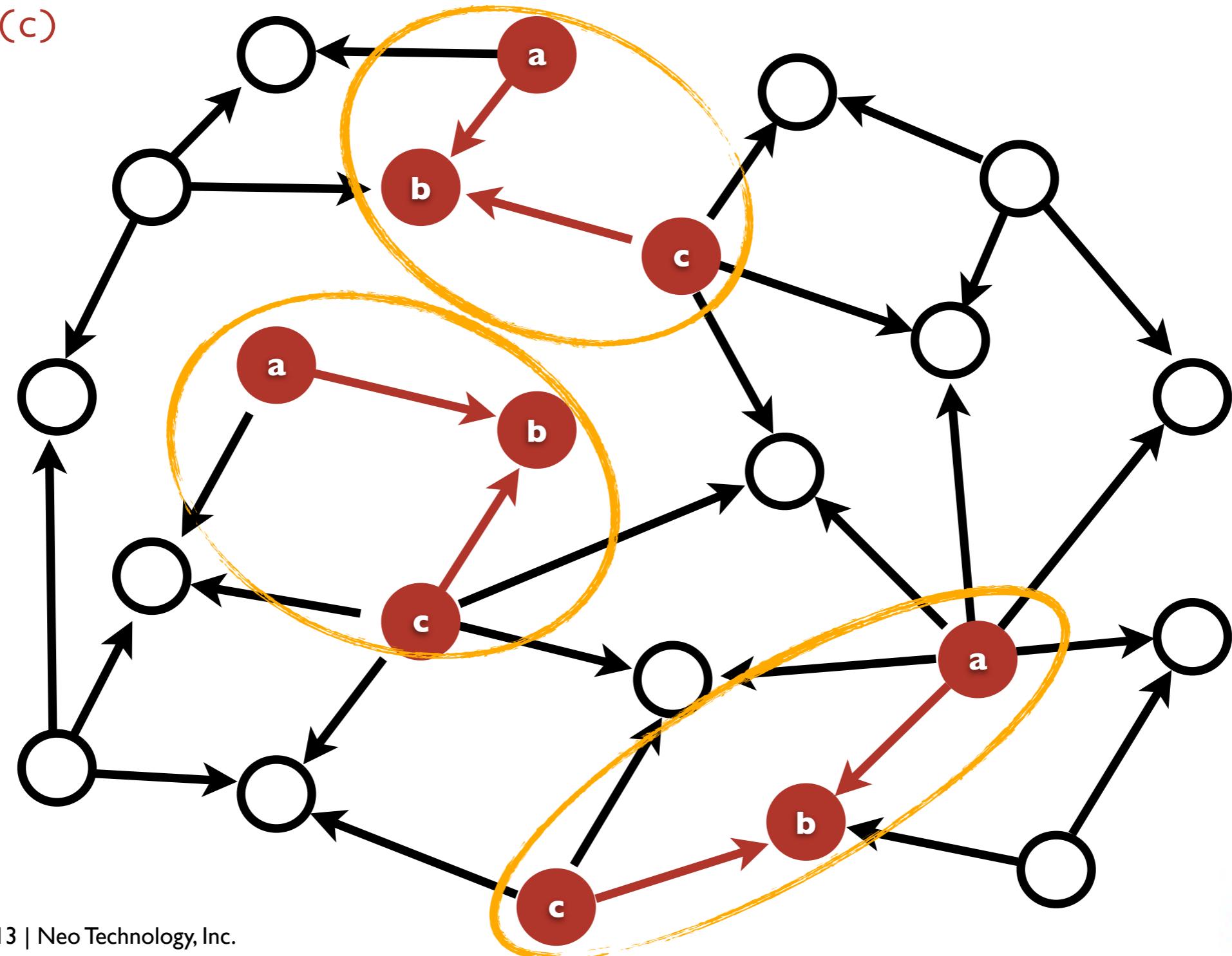
(a) - -> (b) - -> (c)

# Paths



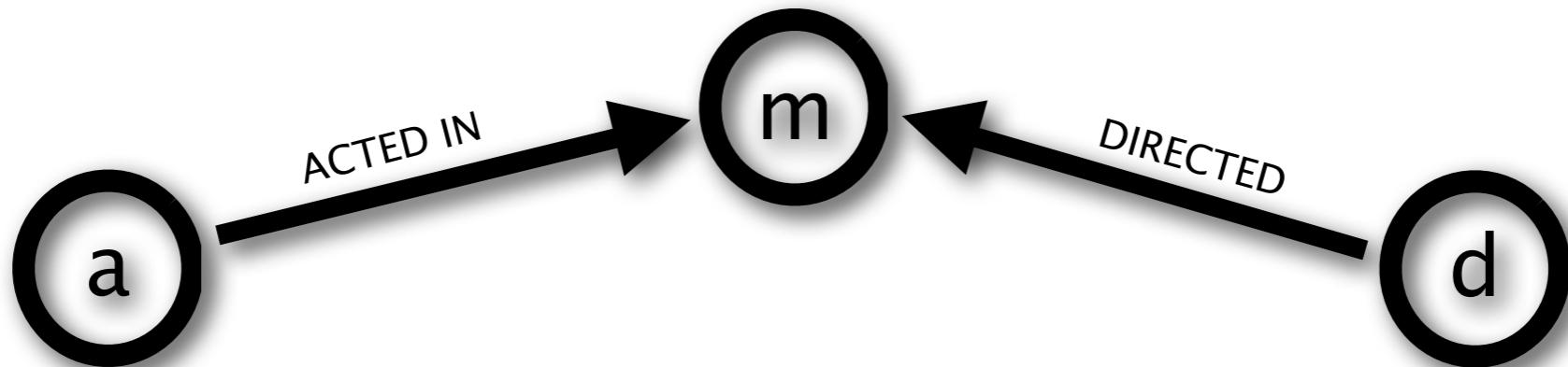
(a) - -> (b) <-- (c)

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

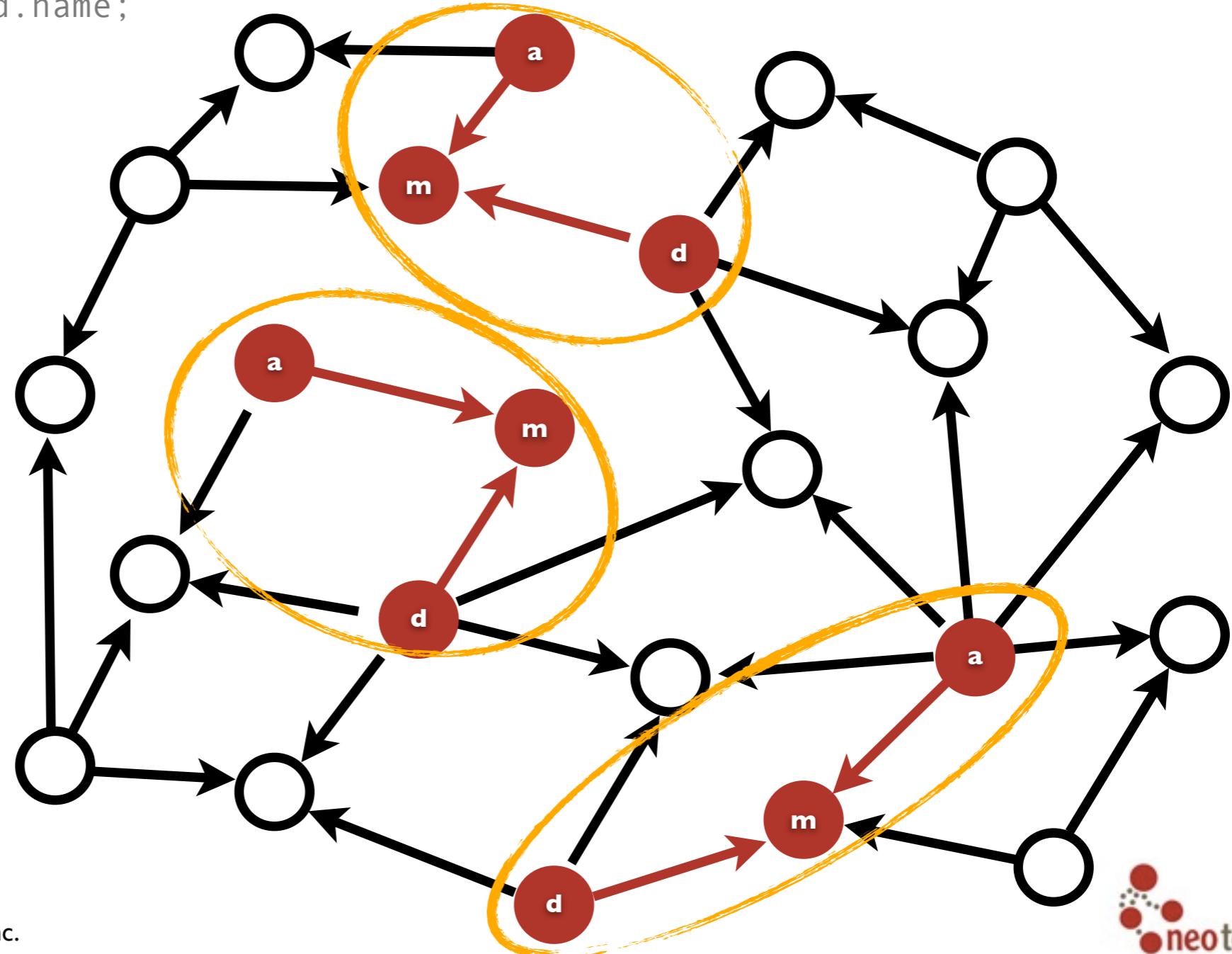


# Paths

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



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



# 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)<-[:DIRECTED]-(d)  
RETURN a.name AS actor, m.title AS movie,  
       d.name AS director;
```

| actor          | movie                 | director         |
|----------------|-----------------------|------------------|
| “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;
```



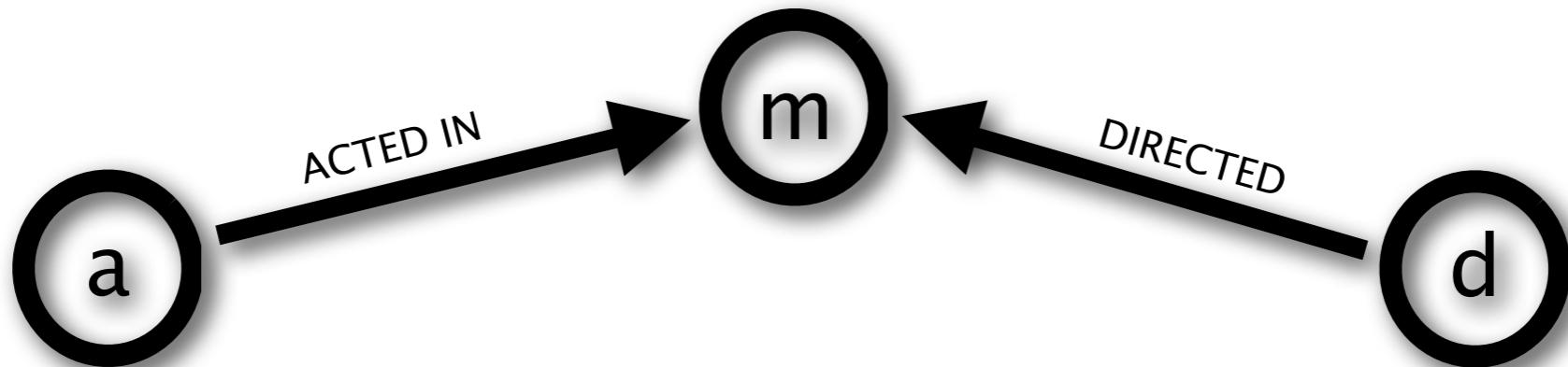
# Paths

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



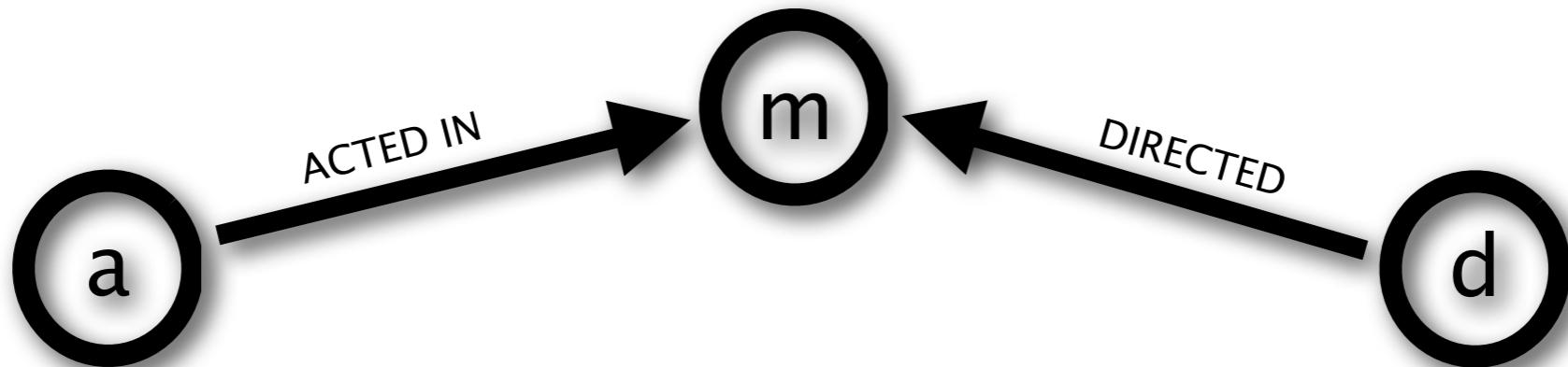
# Paths

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



# Paths

```
START a=node(*)  
MATCH p=(a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)  
RETURN nodes(p);
```



# Paths

START

a=node(\*)

MATCH p1=(a)-[:ACTED\_IN]->(m) , p2=(d)-[:DIRECTED]->(m)

RETURN p1, p2;



# Aggregation

```
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” |
| ...            | ...                   | ...              |

# Aggregation

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

| a.name         | d.name           | count(*) |
|----------------|------------------|----------|
| “Aaron Sorkin” | “Rob Reiner”     | 2        |
| “Keanu Reeves” | “Andy Wachowski” | 3        |
| “Hugo Weaving” | “Tom Tykwer”     | 1        |
| ...            | ...              | ...      |

# Aggregation

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

| a.name         | d.name           | count(m) |
|----------------|------------------|----------|
| “Aaron Sorkin” | “Rob Reiner”     | 2        |
| “Keanu Reeves” | “Andy Wachowski” | 3        |
| “Hugo Weaving” | “Tom Tykwer”     | 1        |
| ...            | ...              | ...      |

# LAB

Which directors also acted in their movie?

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

# Unique relationships in paths

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

# Sort & Limit

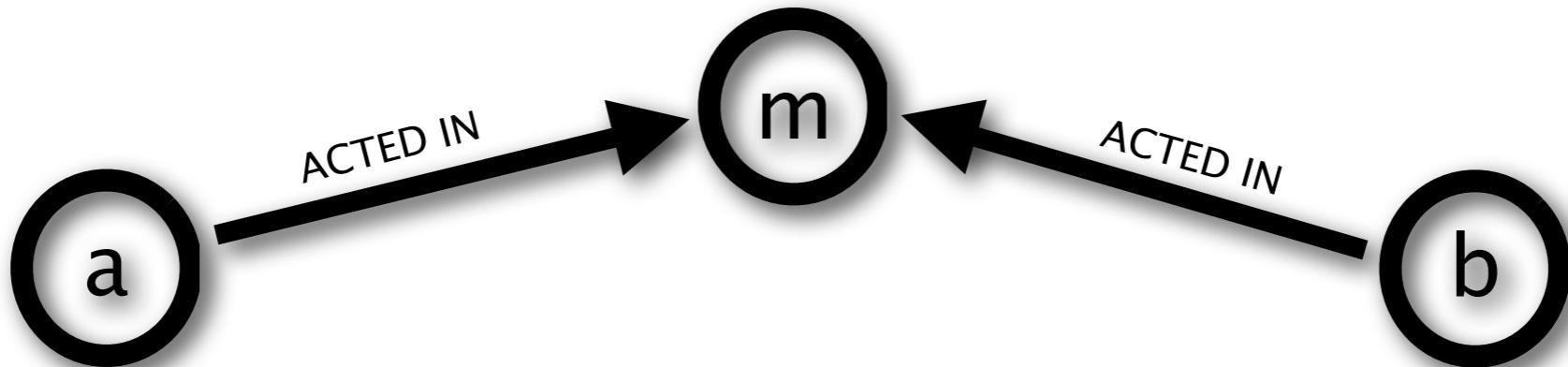
```
START a=node(*)  
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)  
RETURN a.name, d.name, count(*) AS count  
ORDER BY(count) DESC  
LIMIT 5;
```

# 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);
```



# Starting somewhere

# All-nodes Query

```
START n=node(*) RETURN n;
```

**START** - clause for looking up starting points

**node(\*)** - all nodes in the graph

**RETURN n** - clause to specify data to return

# Find a specific node (all-node query)

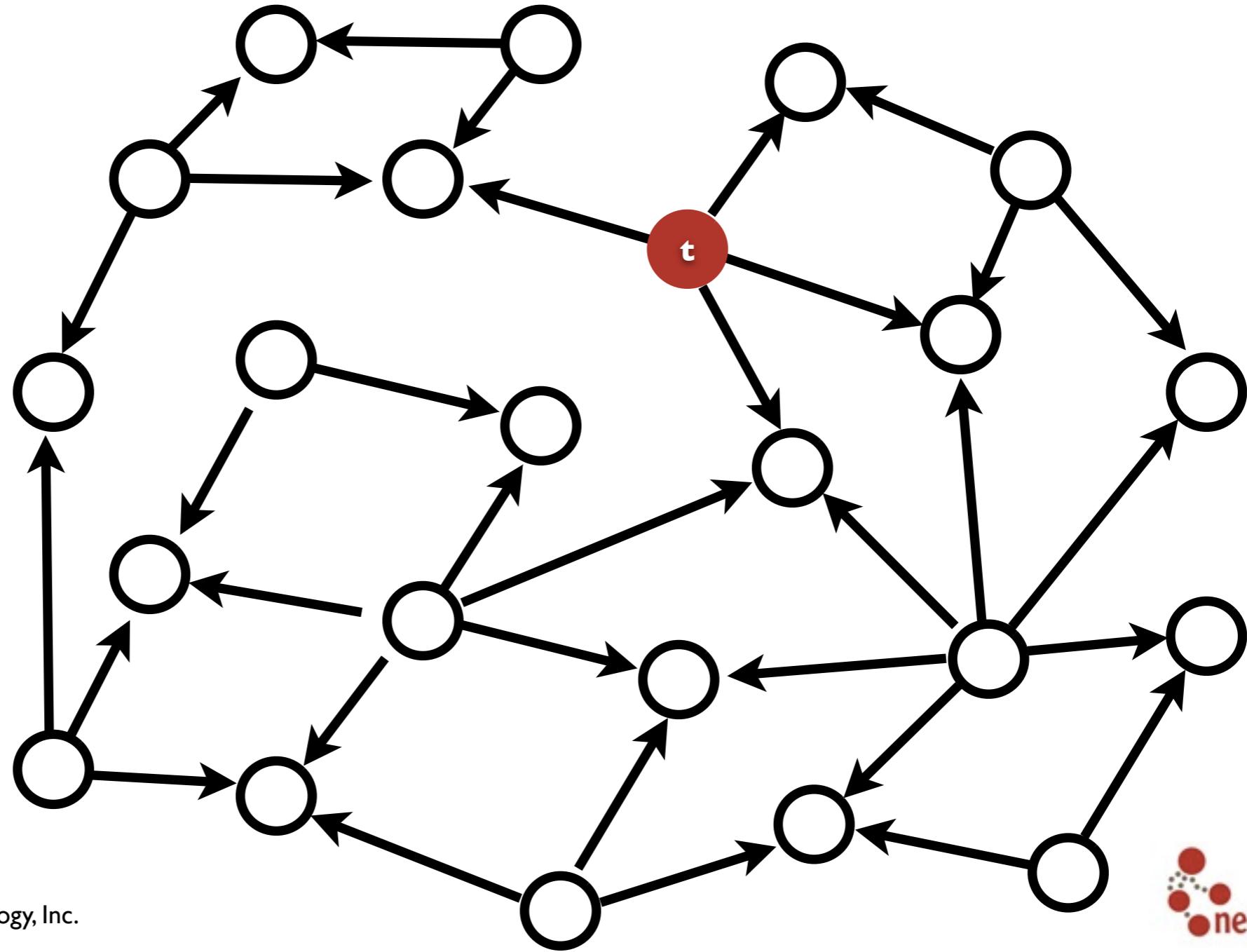
```
START n=node(*)  
WHERE has(n.name) AND n.name = "Tom Hanks"  
RETURN n;
```

WHERE - filter the results

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

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

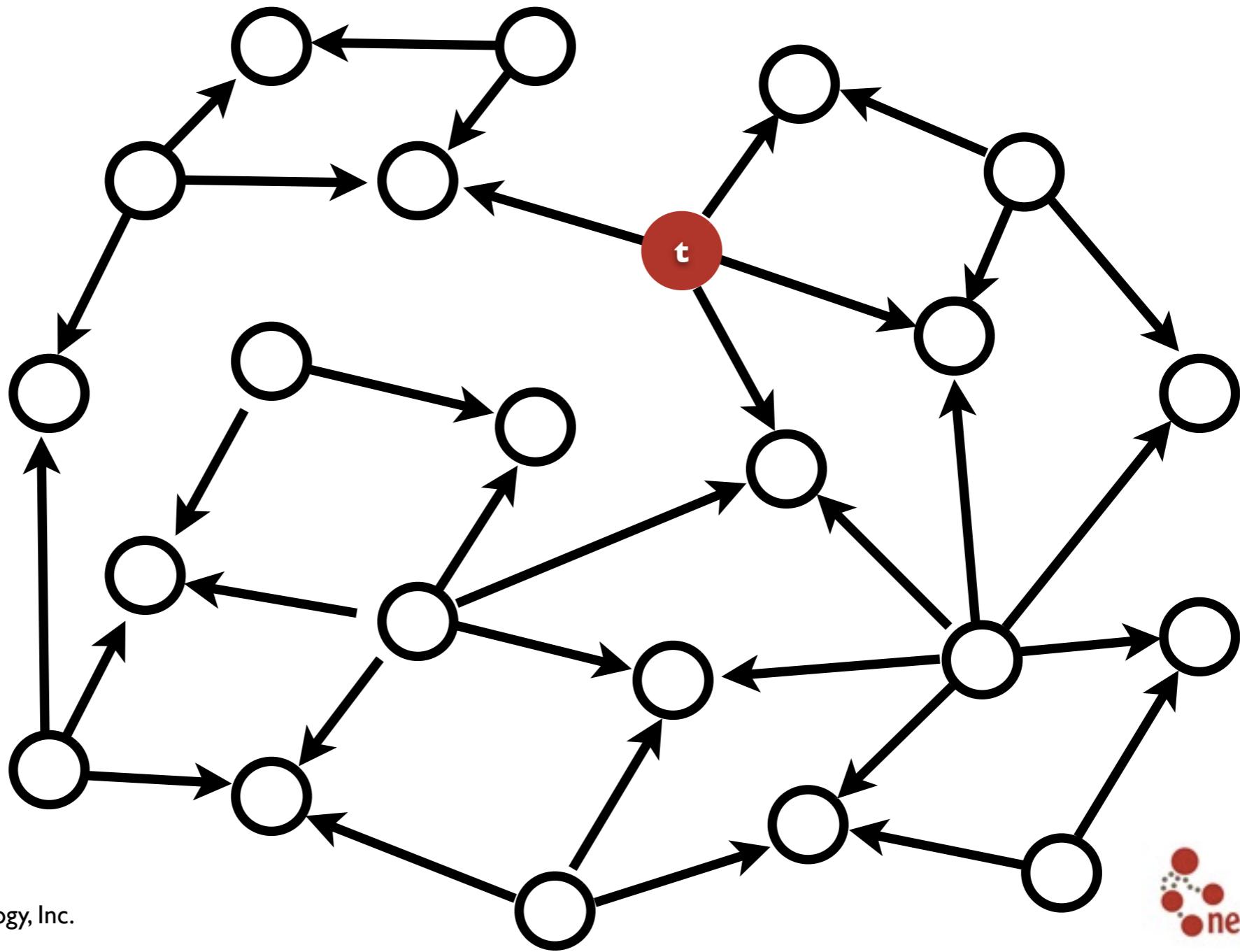
```
START n=node(*)  
WHERE has(n.name) AND n.name = "Tom Hanks"  
RETURN n;
```



# Find a specific node (auto-index query)

```
START tom=node:node_auto_index(name="Tom Hanks")
RETURN tom;
```

```
START keanu=node:node_auto_index(name="Keanu Reeves")
RETURN keanu;
```

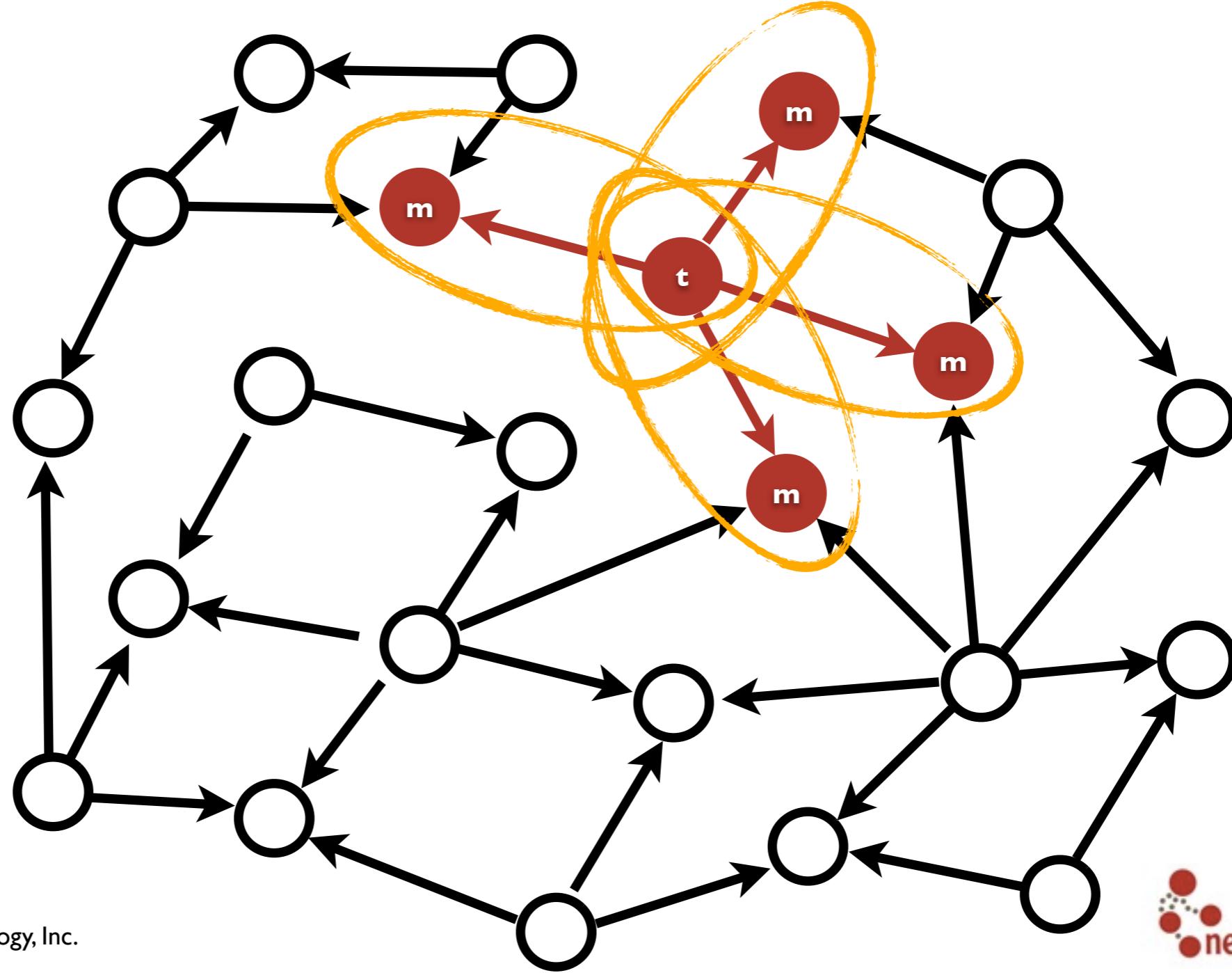


# Start with a specific node (auto-indexed)

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)
RETURN movie.title;
```

*(Movies featuring Keanu)*

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)
RETURN movie.title;
```

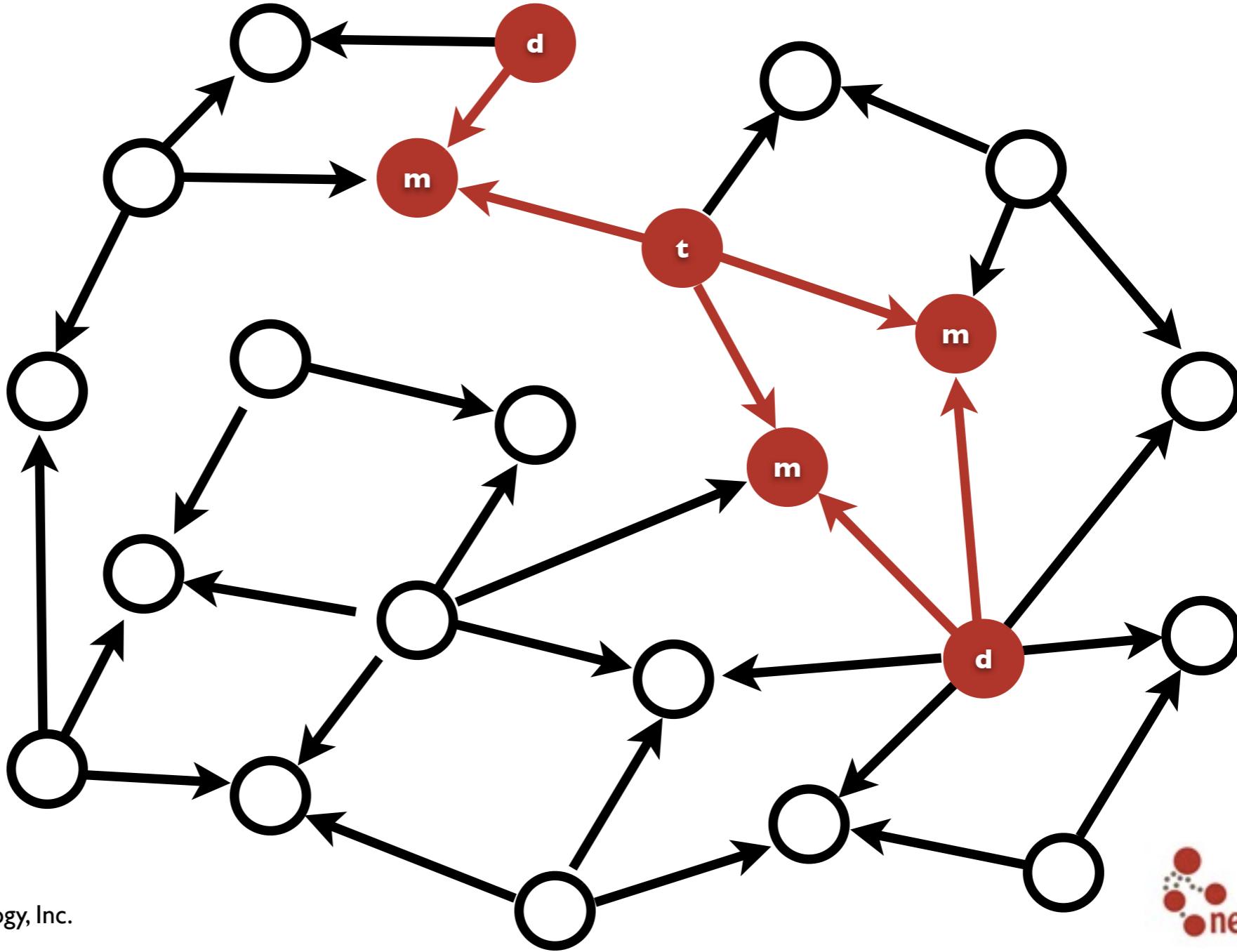


# Start with a specific node (auto-indexed)

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->()-<[:DIRECTED]-(director)
RETURN director.name;
```

*(Directors who worked with Tom Hanks)*

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)- [:ACTED_IN] -> () <- [:DIRECTED] - (director)
RETURN director.name;
```



# Start with a specific node (auto-indexed)

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->()-<[:DIRECTED]-(director)
RETURN DISTINCT director.name;
```

*(Directors who worked with Tom Hanks)*

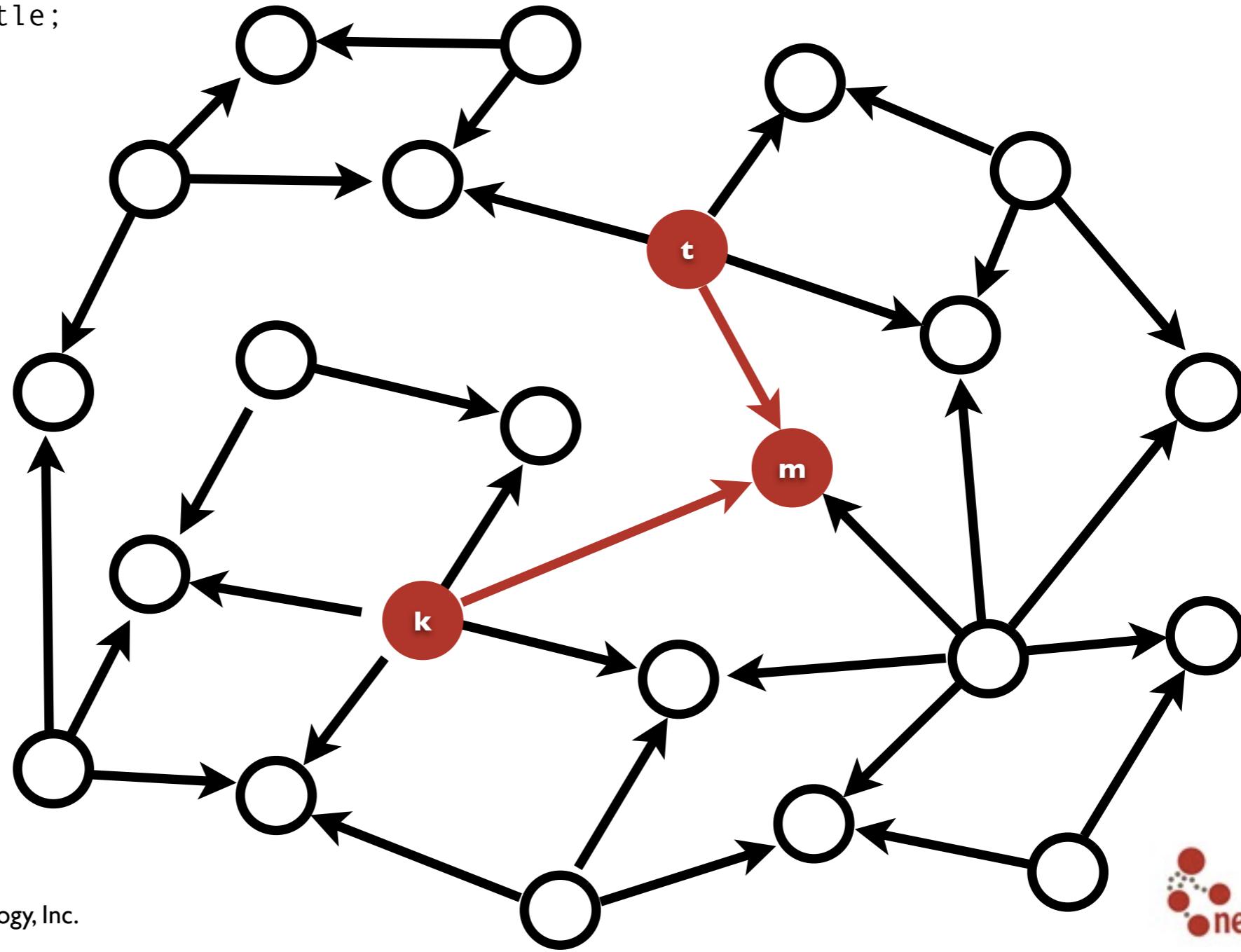
# Start with multiple nodes (auto-indexed)

```
START tom=node:node_auto_index(name="Tom Hanks") ,  
     kevin=node:node_auto_index(name="Kevin Bacon")  
MATCH (tom)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(kevin)  
RETURN DISTINCT movie.title;
```

*(Movies featuring both Tom Hanks and Kevin Bacon)*



```
START tom=node:node_auto_index(name="Tom Hanks"),  
kevin=node:node_auto_index(name="Kevin Bacon")  
MATCH (tom)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(kevin)  
RETURN DISTINCT movie.title;
```



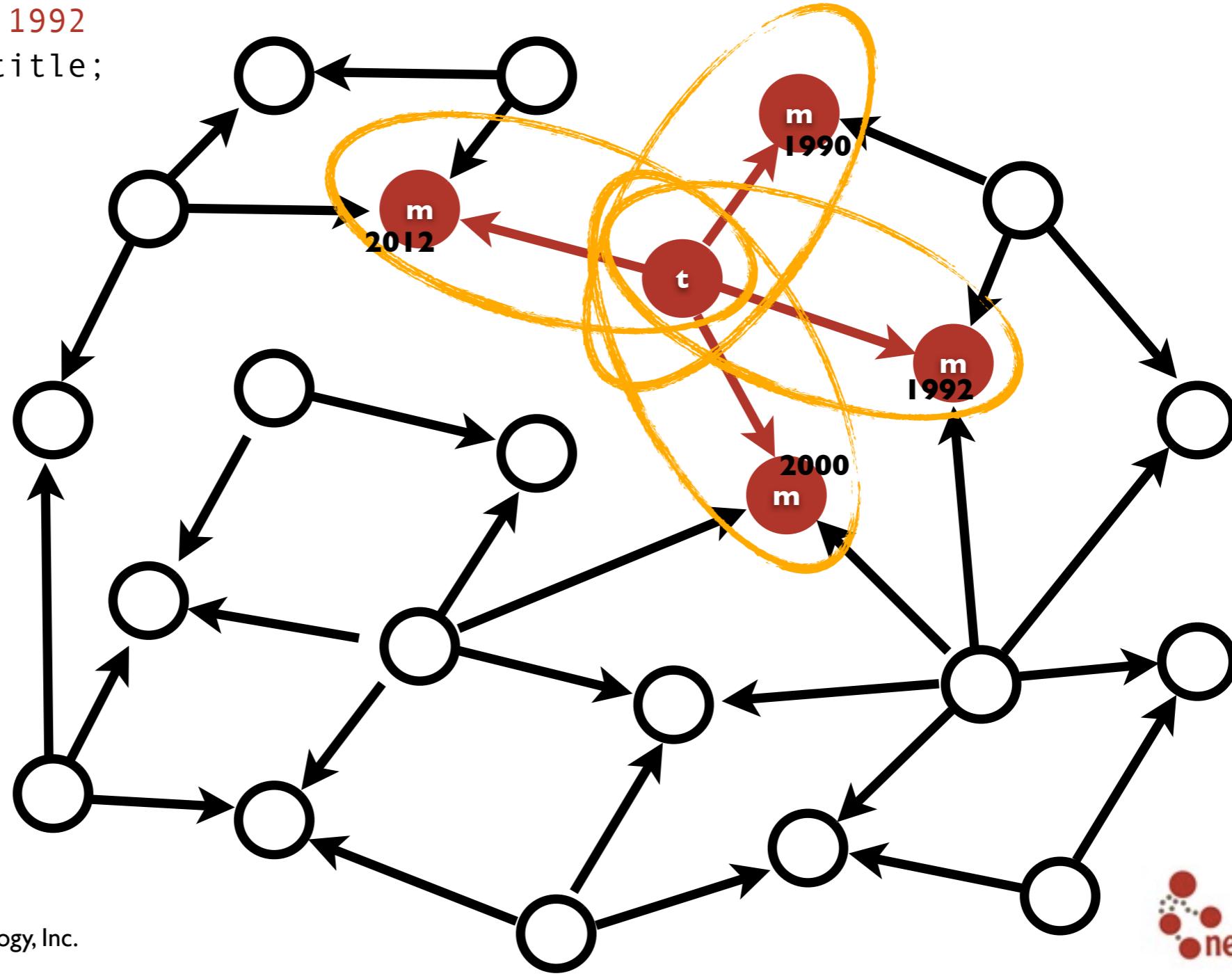
# Conditions

# Constraints on properties

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)
WHERE movie.released < 1992
RETURN DISTINCT movie.title;
```

*(Movies in which Tom Hanks acted, that were released before 1980)*

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom) - [:ACTED_IN] ->(movie)
WHERE movie.released < 1992
RETURN DISTINCT movie.title;
```



# Constraints on properties

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

*(Movies in which Keanu Reeves played Neo)*

# Constraints based on comparisons

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(a)
WHERE a.born < tom.born
RETURN DISTINCT a.name;
```

*(Actors who worked with Tom and are older than he was)*

# Constraints based on comparisons

```
START tom=node:node_auto_index(name="Tom Hanks")
MATCH (tom)-[:ACTED_IN]->(movie)<-[:ACTED_IN]-(a)
WHERE a.born < tom.born
RETURN DISTINCT a.name, (tom.born - a.born) AS diff;
```

*(Actors who worked with Tom and are older than he was)*



# Constraints based on patterns

```
START gene=node:node_auto_index(name="Gene Hackman")
MATCH (gene) - [:ACTED_IN] ->(movie)<- [:ACTED_IN] - (n)
RETURN DISTINCT n.name;
```

*(Actors who worked with Gene Hackman)*

# Constraints based on patterns

```
START gene=node:node_auto_index(name="Gene Hackman")
MATCH (gene) - [:ACTED_IN] -> (movie) <- [:ACTED_IN] - (n)
WHERE (n) - [:DIRECTED] -> ()
RETURN DISTINCT n.name;
```

*(Actors who worked with Gene and were directors of their own films)*

# Constraints based on patterns

```
START keanu=node:node_auto_index(name="Keanu Reeves") ,  
      hugo=node:node_auto_index(name="Hugo Weaving")  
MATCH (keanu) - [:ACTED_IN] -> (movie) <- [:ACTED_IN] - (n)  
WHERE NOT (hugo)- [:ACTED_IN] -> (movie)  
RETURN DISTINCT n.name;
```

*(Actors who worked with Keanu, but never worked with Hugo)*

# LAB

Who are the five busiest actors?

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

# ADVANCED LAB

Recommend 3 actors that Keanu Reeves should work with (but hasn't).

```
START keanu=node:node_auto_index(name="Keanu Reeves")
MATCH (keanu)-[:ACTED_IN]->()-<[:ACTED_IN]-(c),
      (c)-[:ACTED_IN]->()-<[:ACTED_IN]-(coc)
WHERE NOT((keanu)-[:ACTED_IN]->()-<[:ACTED_IN]-(coc))
AND coc <> keanu
RETURN coc.name, count(coc)
ORDER BY count(coc) DESC
LIMIT 3;
```



# Coffee? Lunch?

```
START kevin=node:node_auto_index(name="Kevin Bacon")
MATCH (kevin)-[:ACTED_IN]->(movie)
RETURN DISTINCT movie.title;
```

*(Movies featuring Kevin Bacon)*



# (Updating Graphs with Cypher)

# Creating nodes

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

```
START movie=node:node_auto_index(title="Mystic River")
RETURN movie;
```

# Adding properties

```
START movie=node:node_auto_index(title="Mystic River")
SET movie.tagline = "We bury our sins here, Dave. We wash them clean."
RETURN movie;
```

# Changing properties

```
START movie=node:node_auto_index(title="Mystic River")
SET movie.released = 2003
RETURN movie;
```

# Creating Relationships

```
START movie=node:node_auto_index(title="Mystic River") ,  
    kevin=node:node_auto_index(name="Kevin Bacon")  
CREATE UNIQUE (kevin)-[:ACTED_IN {roles:["Sean"]}] ->(movie);
```

```
START kevin=node:node_auto_index(name="Kevin Bacon")  
MATCH (kevin)-[:ACTED_IN]->(movie)  
RETURN DISTINCT movie.title;
```



# LAB

## Change Kevin Bacon's role in Mystic River from "Sean" to "Sean Devine"

```
START movie=node:node_auto_index(title="Mystic River") ,  
      kevin=node:node_auto_index(name="Kevin Bacon")  
MATCH (kevin)-[r:ACTED_IN]->(movie)  
SET r.roles = ["Sean Devine"]  
RETURN r.roles;
```

# LAB

## Change Kevin Bacon's role in Mystic River from "Sean" to "Sean Devine"

```
START movie=node:node_auto_index(title="Mystic River") ,  
      kevin=node:node_auto_index(name="Kevin Bacon")  
MATCH (kevin)-[r:ACTED_IN]->(movie)  
SET r.roles = filter(n in r.roles : n <> "Sean") + "Sean Devine"  
RETURN r.roles;
```

# LAB

## Add Clint Eastwood as the director of Mystic River

```
START movie=node:node_auto_index(title="Mystic River"),  
    clint=node:node_auto_index(name="Clint Eastwood")  
CREATE UNIQUE (clint)-[:DIRECTED]->(movie);
```

# LAB

List all the characters in the movie “The Matrix”

```
START matrix=node:node_auto_index(title="The Matrix")
MATCH (matrix)<- [r:ACTED_IN] -()
RETURN r.roles;
```

```
START matrix=node:node_auto_index(title="The Matrix")
MATCH (matrix)<- [r:ACTED_IN] - (a)
WHERE "Emil" IN r.roles
RETURN a;
```

# Deleting nodes

```
START emil=node:node_auto_index(name="Emil Eifrem")
DELETE emil;
```

# Deleting relationships

```
START emil=node:node_auto_index(name="Emil Eifrem")
MATCH (emil)-[r]->()
DELETE r;
```

# Deleting nodes and relationships

```
START emil=node:node_auto_index(name="Emil Eifrem")
MATCH (emil)-[r]->()
DELETE r, emil;
```

# Deleting nodes and relationships

```
START emil=node:node_auto_index(name="Emil Eifrem")
MATCH (emil)-[r?]->()
DELETE r, emil;
```

# LAB

Add KNOWS relationships between all actors who  
were in the same movie

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



# Coffee?

# (More Cypher)

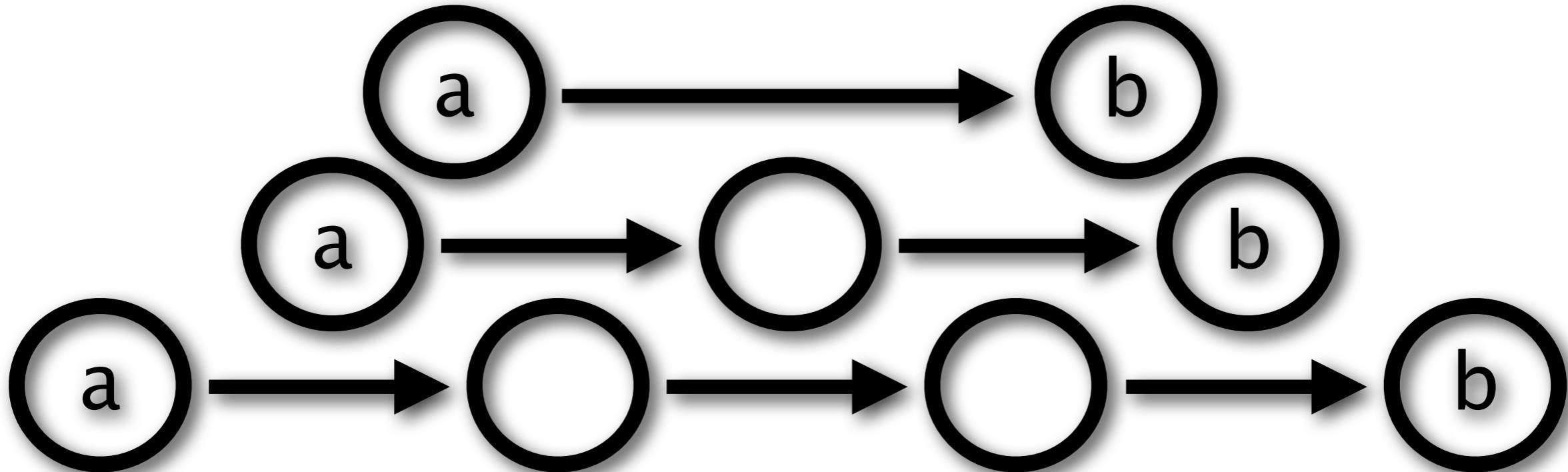
# Matching multiple relationships

```
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

```
START keanu=node:node_auto_index(name="Keanu Reeves")
MATCH (keanu) - [:KNOWS*2] ->(fof)
RETURN DISTINCT fof.name;
```

# LAB

Return Friends-of-Friends who are not immediate friends

```
START keanu=node:node_auto_index(name="Keanu Reeves")
MATCH (keanu) - [:KNOWS*2] ->(fof)
WHERE NOT((keanu) - [:KNOWS] - (fof))
RETURN DISTINCT fof.name;
```

# Bacon Number!

```
START charlize=node:node_auto_index(name="Charlize Theron") ,  
      bacon=node:node_auto_index(name="Kevin Bacon")  
MATCH p=shortestPath((charlize)- [:KNOWS*] ->(bacon))  
RETURN length(rels(p));
```

# LAB

Return the names of the people joining  
Charlize to Kevin.

```
START charlize=node:node_auto_index(name="Charlize Theron"),  
      bacon=node:node_auto_index(name="Kevin Bacon")  
MATCH p=shortestPath((charlize)-[:KNOWS*]->(bacon))  
RETURN extract(n in nodes(p) : n.name);
```

# (Graphs are Everywhere)

Neo4j Case Studies



Deutsche  
Telekom

mozilla

viadeo



Lufthansa

research  
now™  
an e-Rewards company

accenture

*High performance. Delivered.*

Pitney Bowes

CareerArcGroup  
The Social Acceleration Network™

teachscape

InfoJobs  
EMPLEO

splink | tuvo delicioso sport.

squidoo

M800

FIFTYTHREE

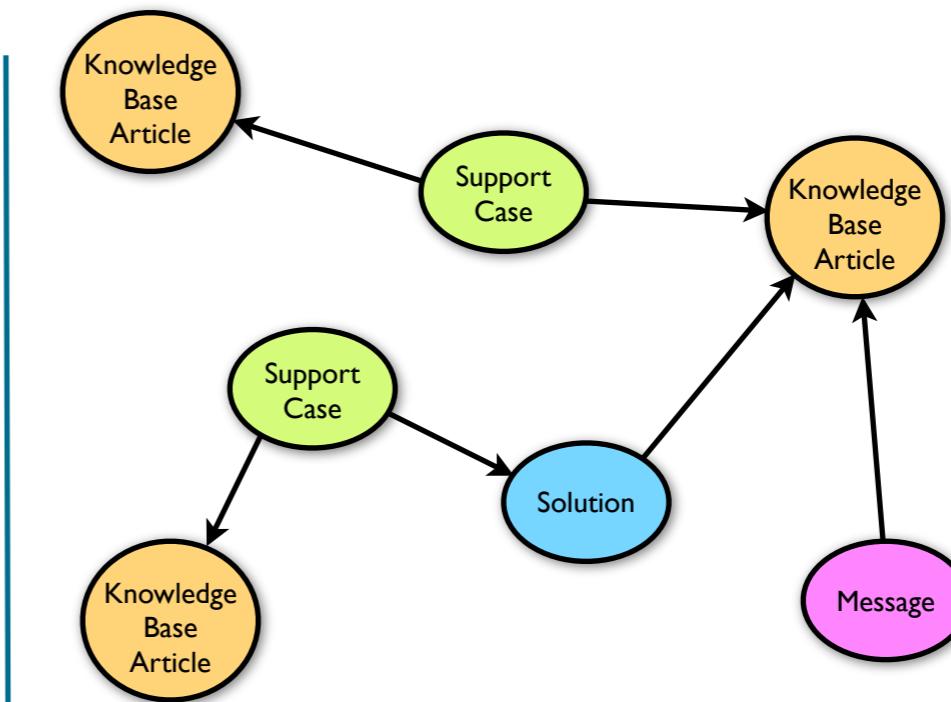
DRW TRADING GROUP

CHIP  
ONLINE

and more.

## Background

- Cisco.com serves customer and business customers with Support Services
- Needed real-time recommendations, to encourage use of online knowledge base
- Cisco had been successfully using Neo4j for its internal master data management solution.
  - Identified a strong fit for online recommendations



## Business problem

- Call center volumes needed to be lowered by improving the efficacy of online self service
- Leverage large amounts of knowledge stored in service cases, solutions, articles, forums, etc.
- Problem resolution times, as well as support costs, needed to be lowered

## Solution & Benefits

- Cases, solutions, articles, etc. continuously scraped for cross-reference links, and represented in Neo4j
- Real-time reading recommendations via Neo4j
- Neo4j Enterprise with HA cluster
- The result: customers obtain help faster, with decreased reliance on customer support



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

**Cisco HMP**

## 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 "PI" 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

## Background

- One of the world's largest logistics carriers
- Projected to outgrow capacity of old system
- New parcel routing system
  - Single source of truth for entire network
  - B2C & B2B parcel tracking
  - Real-time routing: up to 5M parcels per day



## Business problem

- 24x7 availability, year round
- Peak loads of 2500+ parcels per second
- Complex and diverse software stack
- Need predictable performance & linear scalability
- Daily changes to logistics network: route from any point, to any point

## Solution & Benefits

- Neo4j provides the ideal domain fit:
  - a logistics network is a graph
- Extreme availability & performance with Neo4j clustering
- Hugely simplified queries, vs. relational for complex routing
- Flexible data model can reflect real-world data variance much better than relational
- “Whiteboard friendly” model easy to understand

Industry: Online Job Search  
Use case: Social / Recommendations



Sausalito, CA

GlassDoor

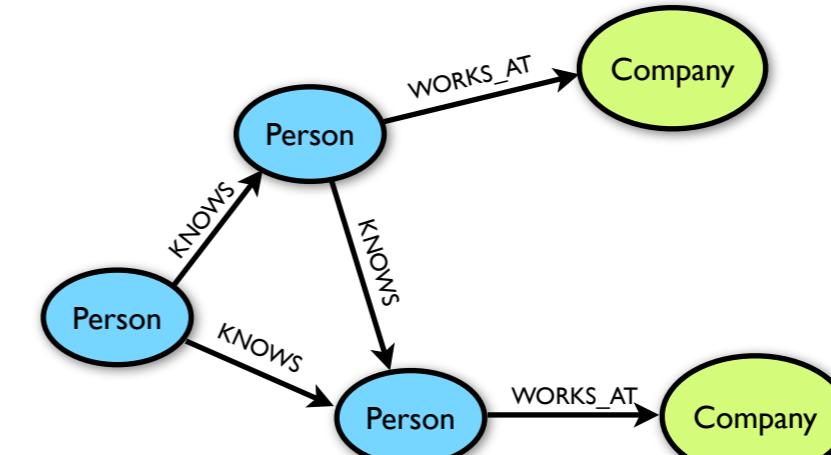
## Background

- Online jobs and career community, providing anonymized inside information to job seekers



## Business problem

- Wanted to leverage known fact that most jobs are found through personal & professional connections
- Needed to rely on an existing source of social network data. Facebook was the ideal choice.
- End users needed to get instant gratification
- Aiming to have the best job search service, in a very competitive market



## Solution & Benefits

- First-to-market with a product that let users find jobs through their network of Facebook friends
- Job recommendations served real-time from Neo4j
- Individual Facebook graphs imported real-time into Neo4j
- Glassdoor now stores > 50% of the entire Facebook social graph
- Neo4j cluster has grown seamlessly, with new instances being brought online as graph size and load have increased



Industry: Web/ISV

Use case: Content Management, Social, Access Control

San Jose, CA

Adobe

Adobe

## Background

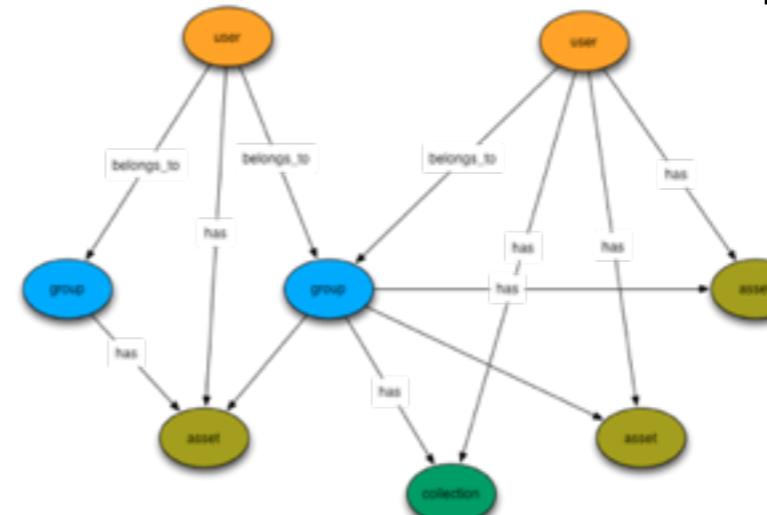
- One of the ten largest software companies globally
- \$4B+ in revenue. Over 11,000 employees.
- Launched Creative Cloud in 2012, allowing its Creative Suite users to collaborate via the Cloud



## Business problem

- Adobe needed a highly robust and available, 24x7 distributed global system, supporting collaboration for users of its highest revenue product line
- Storing creative artifacts in the cloud meant managing access rights for (eventually) millions of users, groups, collections, and pieces of content
- Complex access control rules controlling who was connected to whom, and who could see or edit what, proved a significant technical challenge

## User-Content-Access Graph



## Solution & Benefits

- Selected Neo4j to meet very aggressive project deadlines. The flexibility of the graph model, and performance, were the two major selection factors.
- Easily evolve the system to meet tomorrow's needs
- Extremely high availability and transactional performance requirements. 24x7 with no downtime.
- Neo4j allows consistently fast response times with complex queries, even as the system grows
- First (and possibly still only) database cluster to run across three Amazon EC2 regions: U.S., Europe, Asia



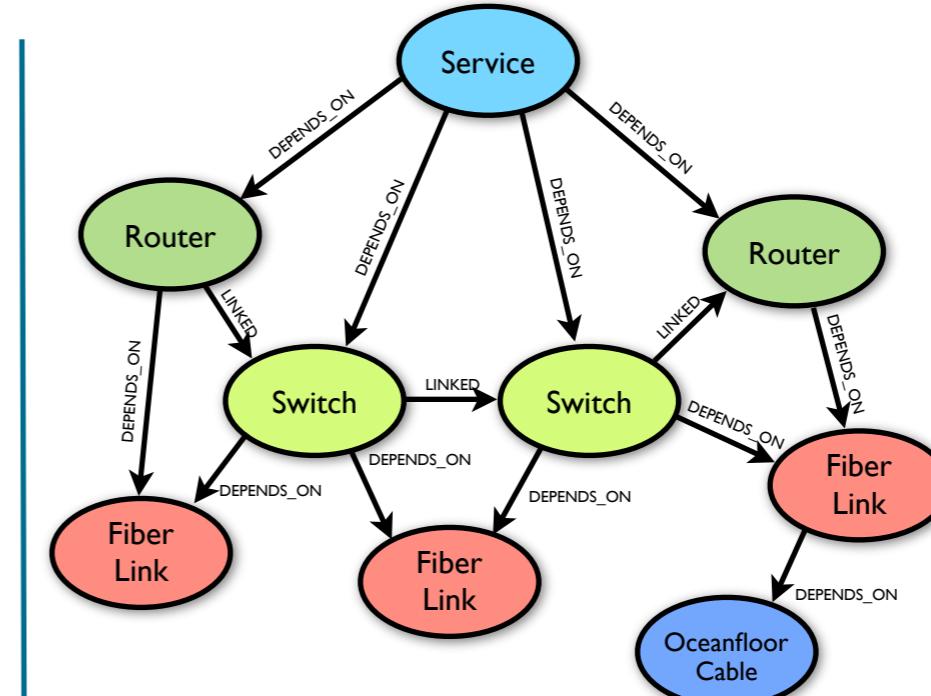
Industry: Communications  
Use case: Network Management

Paris, France

SFR

## Background

- Second largest communications company in France
- Part of Vivendi Group, partnering with Vodafone



## Business problem

- Infrastructure maintenance took one full week to plan, because of the need to model network impacts
- Needed rapid, automated “what if” analysis to ensure resilience during unplanned network outages
- Identify weaknesses in the network to uncover the need for additional redundancy
- Network information spread across > 30 systems, with daily changes to network infrastructure
- Business needs sometimes changed very rapidly

## Solution & Benefits

- Flexible network inventory management system, to support modeling, aggregation & troubleshooting
- Single source of truth (Neo4j) representing the entire network
- Dynamic system loads data from 30+ systems, and allows new applications to access network data
- Modeling efforts greatly reduced because of the near 1:1 mapping between the real world and the graph
- Flexible schema highly adaptable to changing business requirements

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  
Employees worldwide in 2011

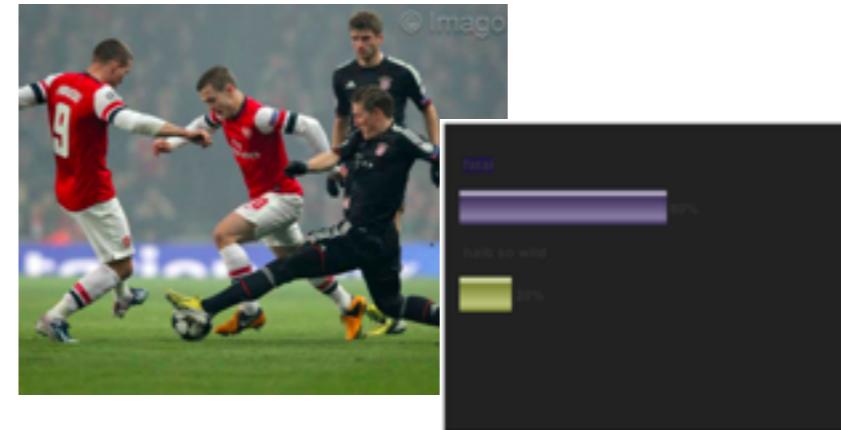
50  
Countries

> 58 bn. €  
Revenue in 2011

## 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

## Interactive Television Programming



## 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



Industry: Web/ISV, Communications

Use case: Network Management

Global (U.S., France)

**Hewlett Packard**

## Background

- World's largest provider of IT infrastructure, software & services
- HP's Unified Correlation Analyzer (UCA) application is a key application inside HP's OSS Assurance portfolio
- Carrier-class resource & service management, problem determination, root cause & service impact analysis
- Helps communications operators manage large, complex and fast changing networks



## Business problem

- Use network topology information to identify root problems causes on the network
- Simplify alarm handling by human operators
- Automate handling of certain types of alarms Help operators respond rapidly to network issues
- Filter/group/eliminate redundant Network Management System alarms by event correlation

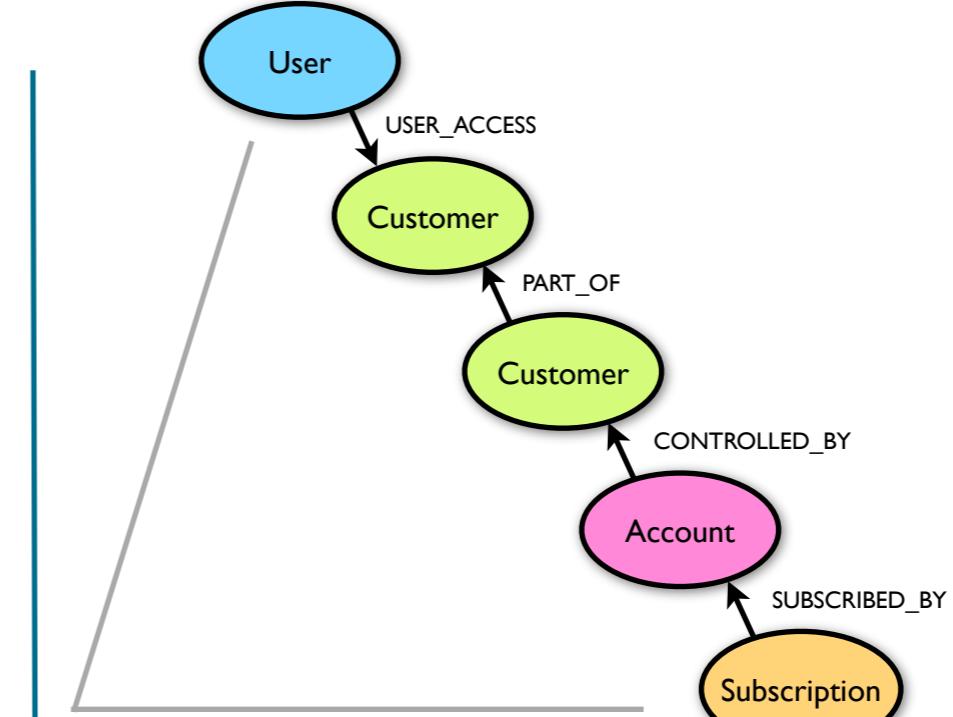
## Solution & Benefits

- Accelerated product development time
- Extremely fast querying of network topology
- Graph representation a perfect domain fit
- 24x7 carrier-grade reliability with Neo4j HA clustering
- Met objective in under 6 months



## Background

- 10th largest Telco provider in the world, leading in the Nordics
- Online self-serve system where large business admins manage employee subscriptions and plans
- Mission-critical system whose availability and responsiveness is critical to customer satisfaction



## Business problem

- Degrading relational performance. User login taking minutes while system retrieved access rights
- Millions of plans, customers, admins, groups.  
Highly interconnected data set w/massive joins
- Nightly batch workaround solved the performance problem, but meant data was no longer current
- Primary system was Sybase. Batch pre-compute workaround projected to reach 9 hours by 2014: longer than the nightly batch window

## Solution & Benefits

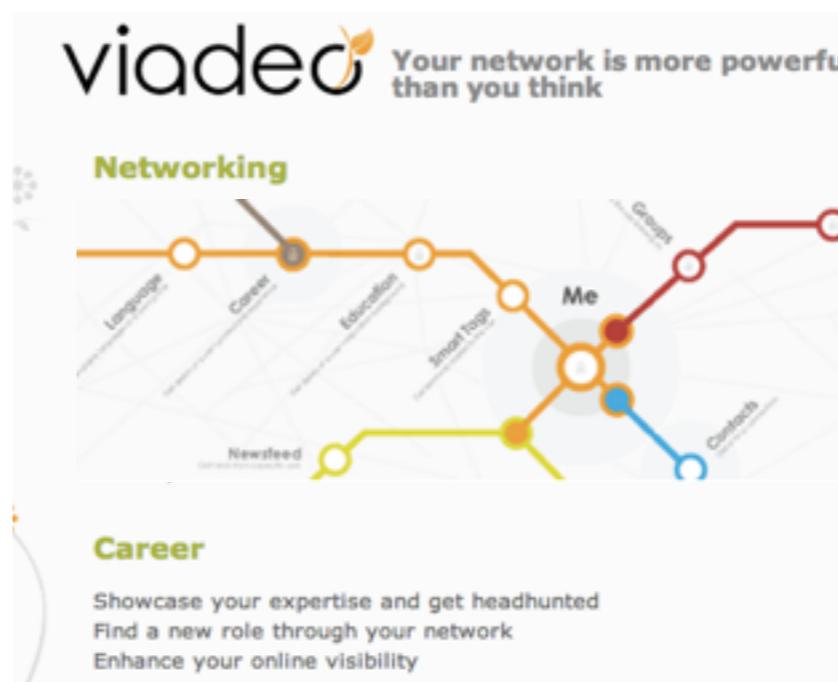
- Moved authorization functionality from Sybase to Neo4j
- Modeling the resource graph in Neo4j was straightforward, as the domain is inherently a graph
- Able to retire the batch process, and move to real-time responses: measured in milliseconds
- Users able to see fresh data, not yesterday's snapshot
- Customer retention risks fully mitigated

Industry: Professional Social Network  
Use case: Social, Recommendations  
Silicon Valley & France



## Background

- World's second-largest professional network (after LinkedIn)
- 50M members. 30K+ new members daily.
- Over 400 staff with offices in 12 countries



## Business problem

- Business imperative for real-time recommendations: to attract new users and retain existing ones
- Key differentiator: show members how they are connected to any other member
- Real-time traversals of social graph not feasible with MySQL cluster. Batch precompute meant stale data.
- Process taking longer & longer: > 1 week!

## Solution & Benefits

- Neo4j solution implemented in 8 weeks with 3 part-time programmers
- Able to move from batch to real-time: improved responsiveness with up-to-date data.
- Viadeo (at the time) had 8M members and 35M relationships.
- Neo4j cluster now sits at the heart of Viadeo's professional network, connecting 50M + professionals

Industry: Communications  
Use case: Social, Mobile



Hong Kong  
**Maaii**

## Background

- Hong Kong based telephony infrastructure provider (aka M800 aka Pop Media)
- Exclusive China Mobile partner for international toll-free services. SMS Hub & other offerings
- 2012 Red Herring Top 100 Global Winner



## Business problem

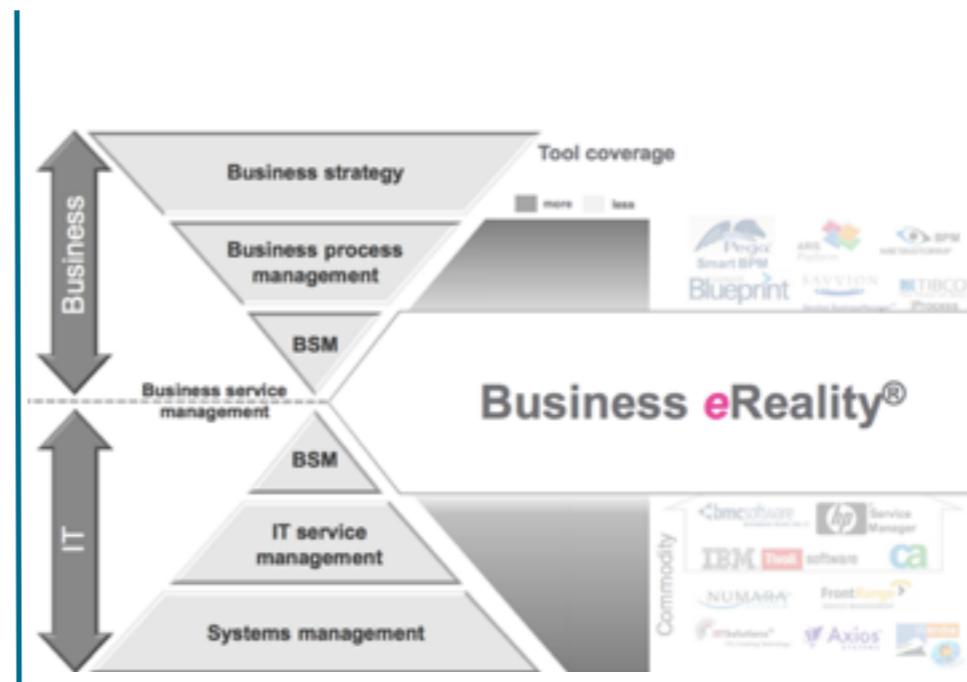
- Launched a new mobile communication app “Maaii” allowing consumers to communicate by voice & text (Similar to Line, Viber, Rebtel, VoxOx...)
- Needed to store & relate devices, users, and contacts
- Import phone numbers from users’ address books. Rapidly serve up contacts from central database to the mobile app
- Currently around 3M users w/200M nodes in the graph

## Solution & Benefits

- Quick transactional performance for key operations:
  - friend suggestions (“friend of friend”)
  - updating contacts, blocking calls, etc.
  - etc.
- High availability telephony app uses Neo4j clustering
- Strong architecture fit: Scala w/Neo4j embedded

## Background

- Junisphere AG is a Zurich-based IT solutions provider
- Founded in 2001. Profitable. Self funded.
- Software & services.
- Novel approach to infrastructure monitoring:  
Starts with the end user, mapped to business processes and services, and dependent infrastructure



## Business problem

- “Business Service Management” requires mapping of complex graph, covering: business processes--> business services--> IT infrastructure
- Embed capability of storing and retrieving this information into OEM application
- Re-architecting outdated C++ application based on relational database, with Java

## Solution & Benefits

- Actively sought out a Java-based solution that could store data as a graph
- Domain model is reflected directly in the database:
  - “No time lost in translation”
  - “Our business and enterprise consultants now speak the same language, and can model the domain with the database on a 1:1 ratio.”
- Spring Data Neo4j strong fit for Java architecture

## Background

- Teachscape, Inc. develops online learning tools for K-12 teachers, school principals, and other instructional leaders.
- Teachscape evaluated relational as an option, considering MySQL and Oracle.
- Neo4j was selected because the graph data model provides a more natural fit for managing organizational hierarchy and access to assets.



## Business problem

- Neo4j was selected to be at the heart of a new architecture.
- The user management system, centered around Neo4j, will be used to support single sign-on, user management, contract management, and end-user access to their subscription entitlements.

## Solution & Benefits

- **Domain and technology fit**
  - simple domain model where the relationships are relatively complex. Secondary factors included support for transactions, strong Java support, and well-implemented Lucene indexing integration
- **Speed and Flexibility**
  - The business depends on being able to do complex walks quickly and efficiently. This was a major factor in the decision to use Neo4j.
- **Ease of Use**
  - accommodate efficient access for home-grown and commercial off-the-shelf applications, as well as ad-hoc use.
  - Extreme availability & performance with Neo4j clustering
  - Hugely simplified queries, vs. relational for complex routing
  - Flexible data model can reflect real-world data variance much better than relational
  - “Whiteboard friendly” model easy to understand

## Background

- Bioinformatics company offering gene sequencing "as a service" (over the web)
- Provider of genomic information services
- Needed a new platform to support storage & retrieval of sequenced genomes in the cloud



## Business problem

- Neo4j is used to store metadata about each sequenced genome (including a pointer to the sequenced genome itself, which is a binary file stored on Amazon S3), and to support search and other forms of information processing against the genomic data.
- graph database was chosen because “Our specific domain maps naturally onto graph paradigm”.

## Solution & Benefits

- **Domain fit**
  - Domain naturally lends itself to a graph representation.
  - Graph model determined to be a perfect fit.
- **Agility & Performance**
  - Saved time with Neo4j as compared to the alternatives.
  - Queries “practically write themselves.”
- **Solution Completeness**
  - “Neo4j is incomparably better than other graph databases.”



# (More about Neo4j)

Neo4j offers three APIs:

1. Cypher for most work
2. REST for management
3. Plugin API for special cases

# Language Drivers

Friends of Neo4j speak many languages, and work in many frameworks.



## Neo4j REST API

Neo4j Team

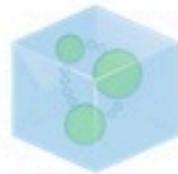
Discoverable, language-neutral data access from anything that can send HTTP requests. You could write a whole application with just bash scripts and curl.



## Spring Data Neo4j

Neo4j Team

Familiar POJO-based development, enabling object-to-graph mapping using annotations. Amazingly simple, with full graph power just a traversal query away.



## Java API

Neo4j Team

For intimate access, talk directly to Neo4j's graph engine directly in your JVM based application. Full feature parity with Neo4j Server, including HA clustering.



## neo4j.rb

Andreas Ronge

Ruby on Rails? Try coasting along graph paths with Neo4j. Everything you know and love, wrapped with graph glory.



## Neography

Max de Marzi

For native Ruby access to Neo4j, Neography provides a thin, elegant wrapper around the REST API.



## Neo4jPHP

Josh Adell

Neo4jPHP provides an API that is both intuitive and flexible, and it takes advantage of 'under-the-hood' performance enhancements, such as caching and lazy-loading.

<http://www.neo4j.org/develop/drivers>

Traverses 1,000,000+ relationships / second  
on commodity hardware

# Scale: 34 Bn nodes/relationships

# Licensing options



# (Next steps)

# The rise of the graph

# It's not just social

Find the graphs in your domain,  
and model them

# Get involved in the community

# Stack Overflow



Find answers or reach to fellow  
developers with questions.

[Ask Neo4j questions »](#)

<http://stackoverflow.com/questions/tagged/neo4j>

## Neo4j Google Group



Share your experiences and expertise  
with fellow graphistas.

[Join now »](#)

<http://groups.google.com/group/neo4j>

## GitHub Issues



Encountered an issue with Neo4j?  
Submit it here.

<https://github.com/neo4j/neo4j/issues>

## Meetups / User Groups



Neo4j meetups are worldwide. Make a connection or start a new group.

[Join a Meetup »](#)

<http://neo4j.meetup.com/>

# Graphistas World Map



Add yourself to the graphistas world map and let it become a smaller place.

[Add yourself »](#)

<http://www.neo4j.org/participate/contributors#map>



# (Thank You)

<http://www.neo4j.org>

<http://groups.google.com/group/neo4j>

<http://www.neo4j.org/develop>