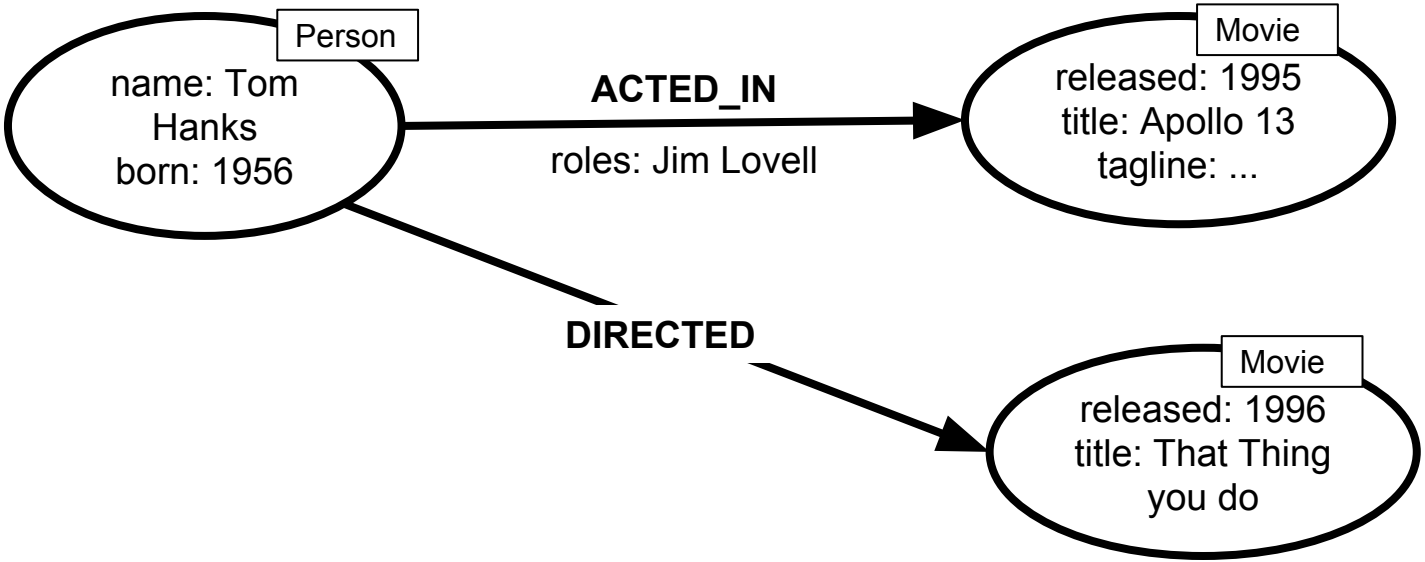
Neo4j Fundamentals Training

Welcome everyone

Please say hi to each other.
We'll spend the day together.

Our Data Model for Today - Movies and Actors



Logistics

- WIFI-SSID: **some-ssid** Password: **some-password**
- Install last stable Neo4j version from USB / neo4j.com/download
- 1 morning break, lunch, 1 afternoon break
- Working together today
  - Pair on the exercises, that works much better, help each other!
  - Ask relevant questions immediately, raise your hand if you have problems
  - Keep larger questions to the breaks / end
  - Use the scratch-pad google doc: bit.ly/_fundamentals
  - Join neo4j.com/slack and go to the #training-fundamentals channel
- Please fill out the feedback form: bit.ly/neo-survey (gives a discount code)

What are we going to do today?

- Intro to graphs and Neo4j 30’
- First steps in Cypher 30’
- Importing the movie data 30’
- Querying the movie graph 90’
  - `:play http://guides.neo4j.com/fundamentals/`
- Build and run a movie graph application 60’
- Relational to Graph 30’
- Show what you've learned 60’
- Resources and Q&A 30’

Intro to graphs and Neo4j

What are we going to do?

- Why graphs?
- What do we use graphs for?
- Intro to the property graph model
- The whiteboard model is the model
- Querying the graph

Why graphs?

What are people using graph databases for?

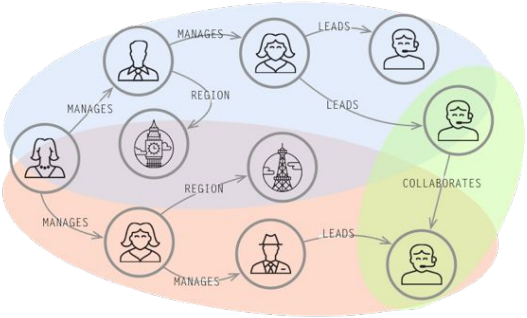
Use Cases

Internal Applications

Master Data Management

Network and
IT Operations

Fraud Detection

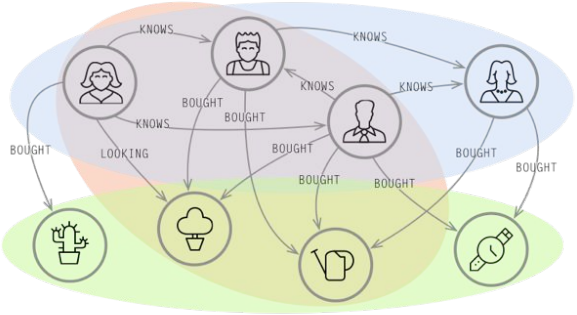


Customer-Facing Applications

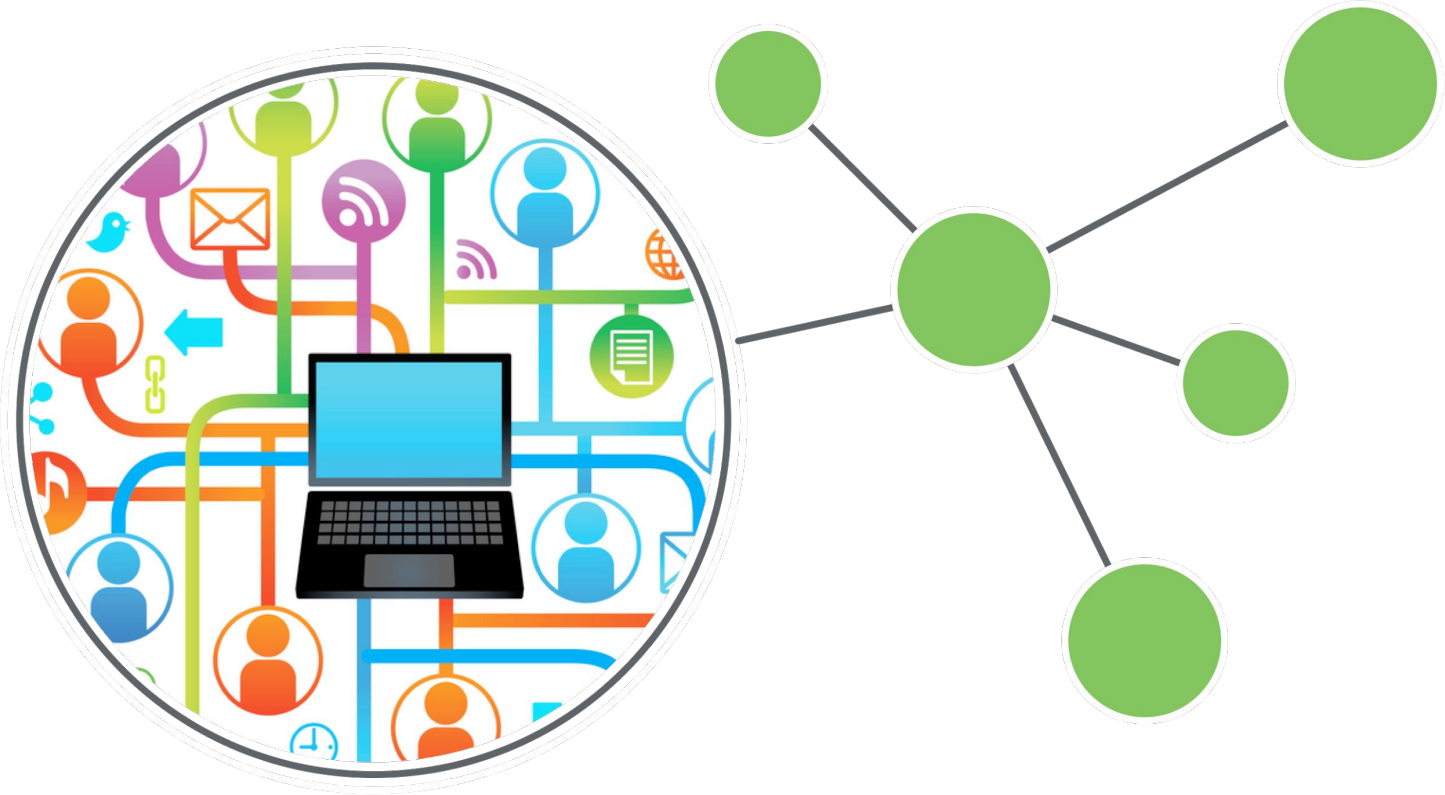
Real-Time Recommendations

Graph-Based Search

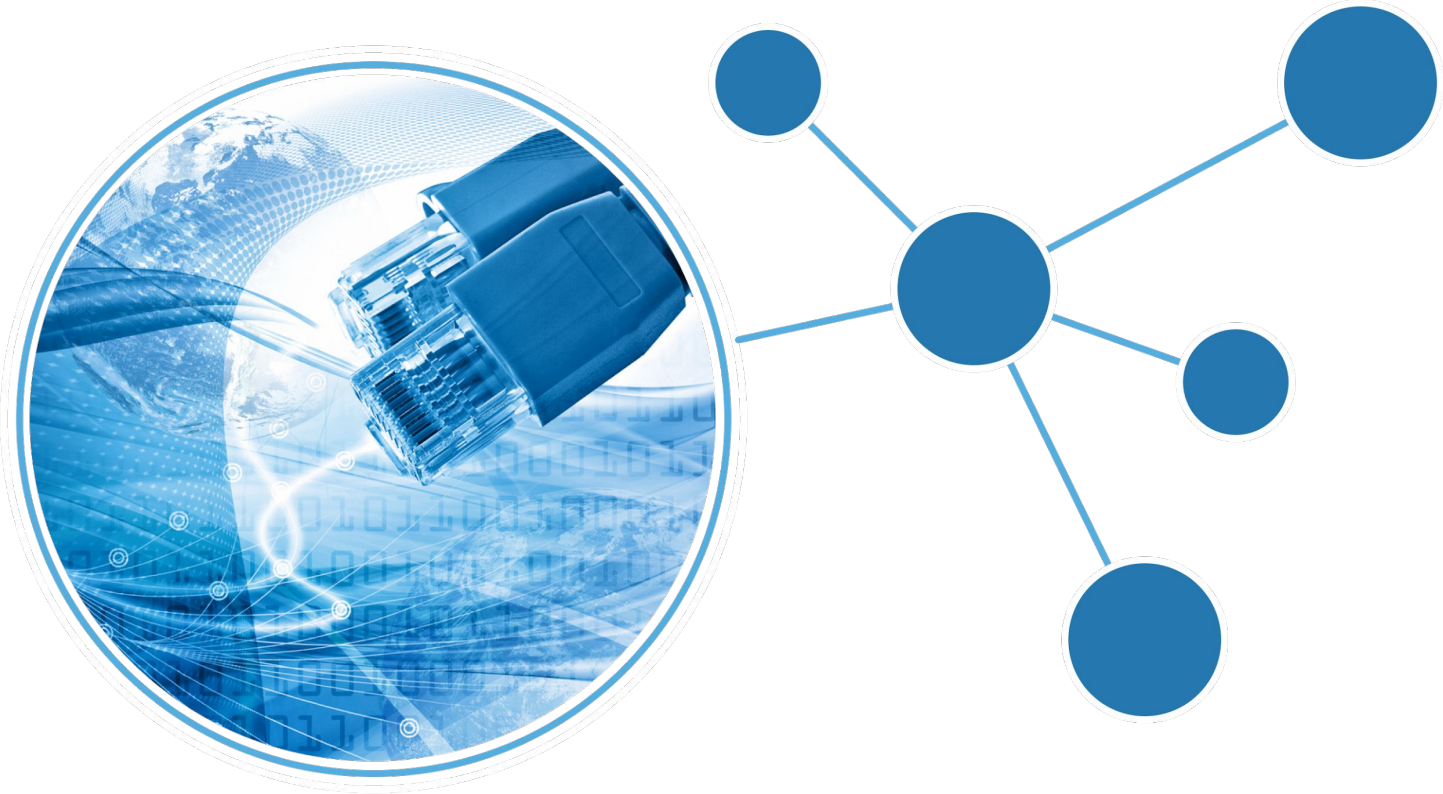
Identity and
Access Management



Social Network

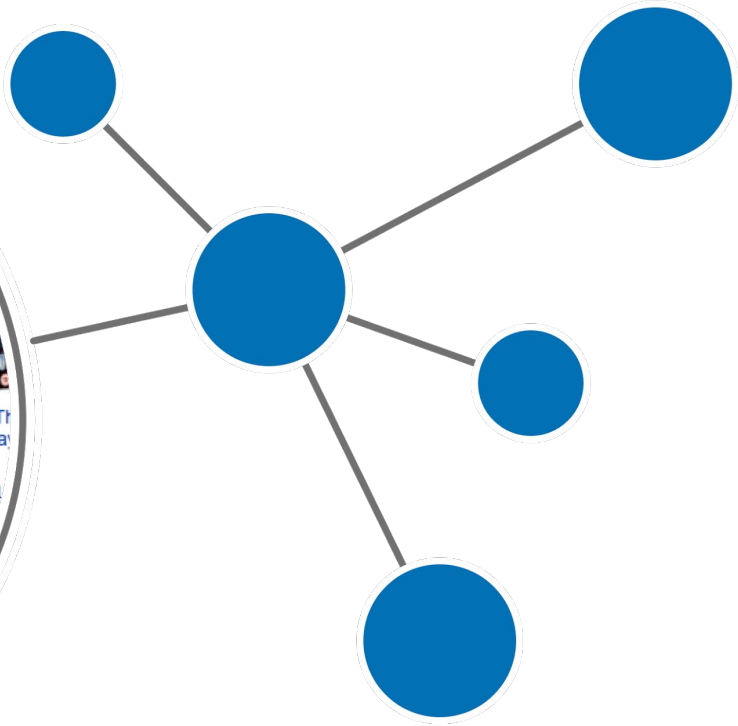


Impact Analysis

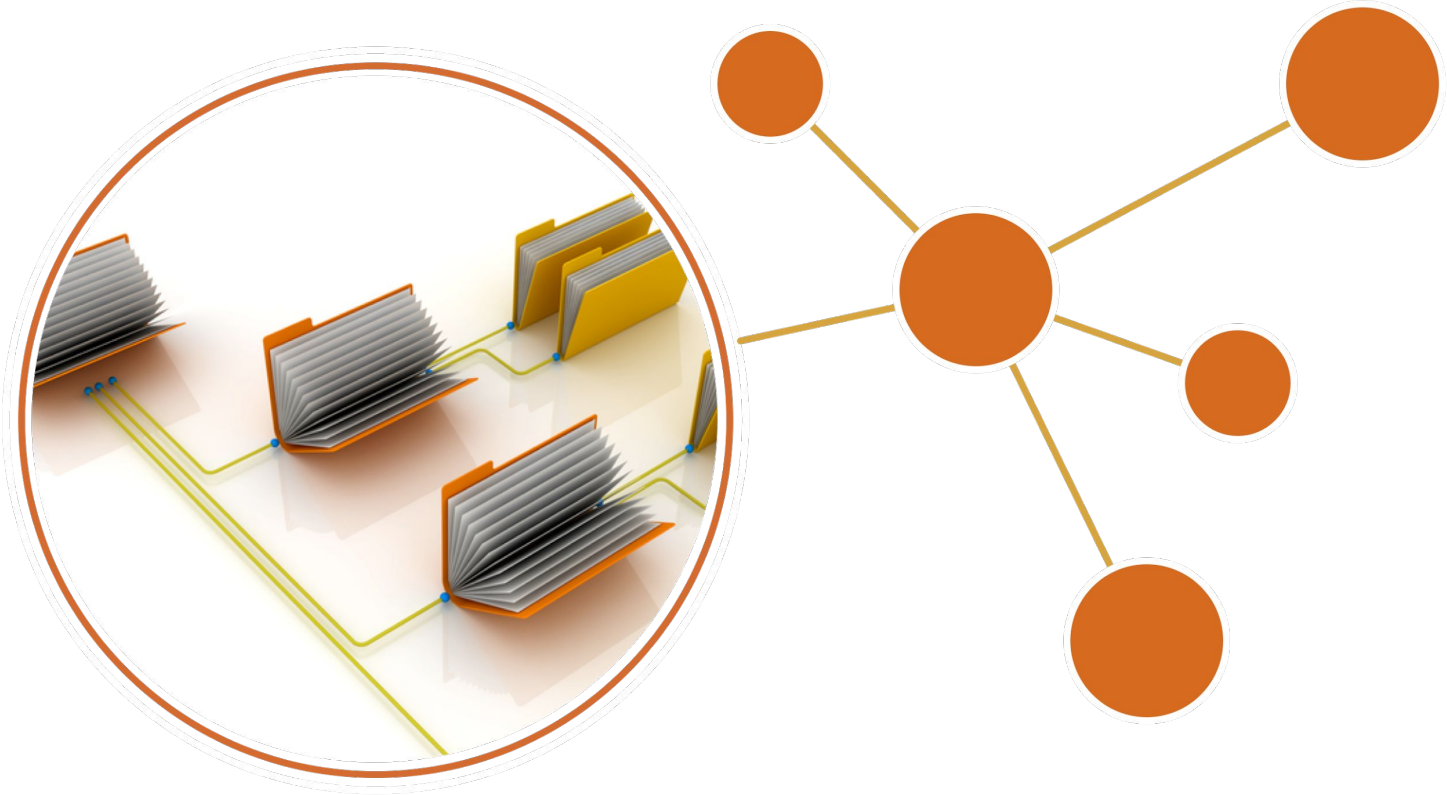


Logistics & Routing

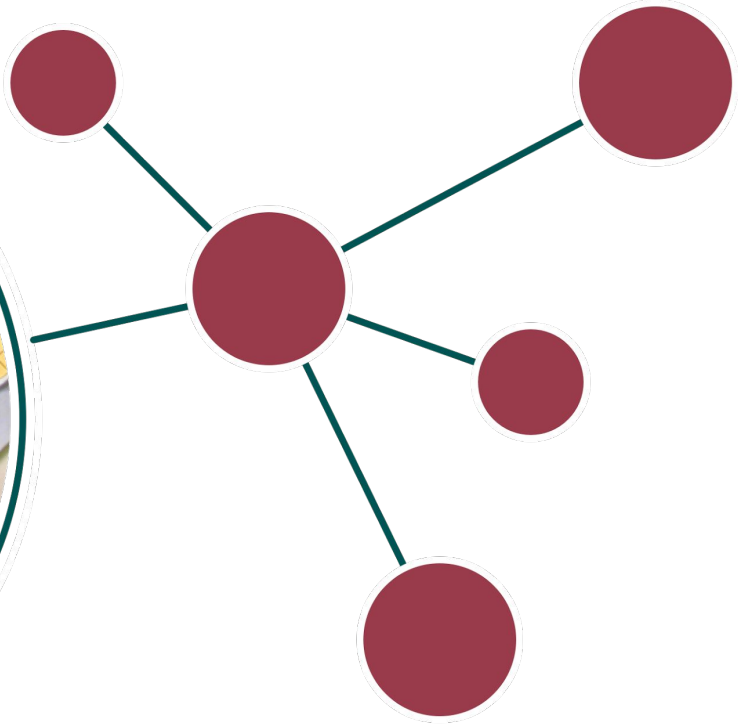
Recommendations



Access Control



Fraud Analysis



Intro to the property graph model

Neo4j Fundamentals

- Nodes
- Relationships
- Properties
- Labels

Property Graph Model Components

Nodes

- Represent the objects in the graph
- Can be *labeled*

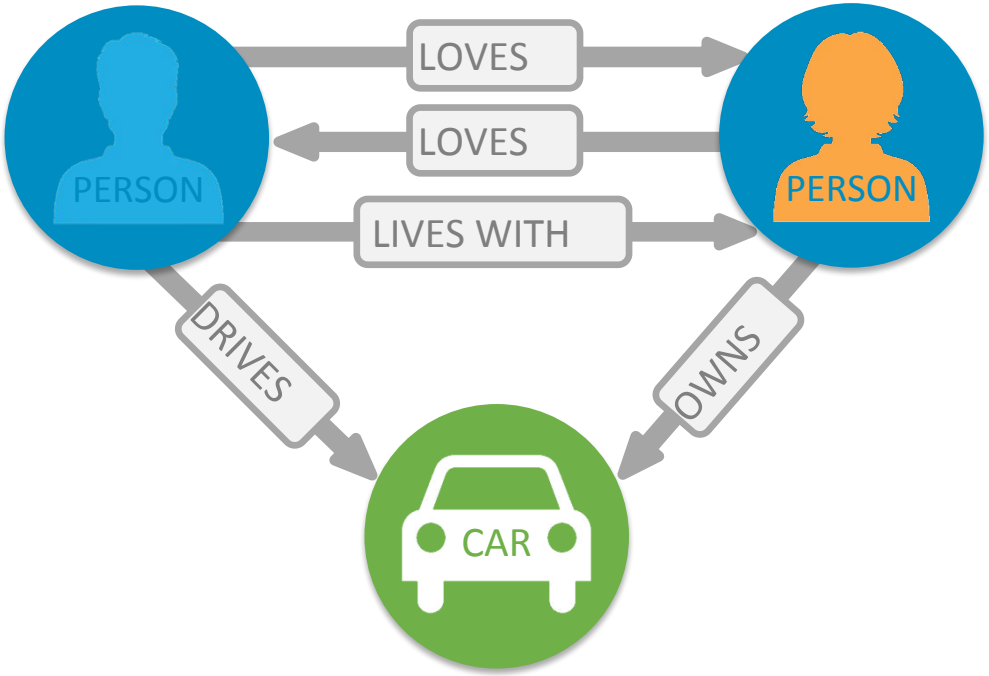
Property Graph Model Components

Nodes

- Represent the objects in the graph
- Can be *labeled*

Relationships

- Relate nodes by *type* and *direction*



Property Graph Model Components

Nodes

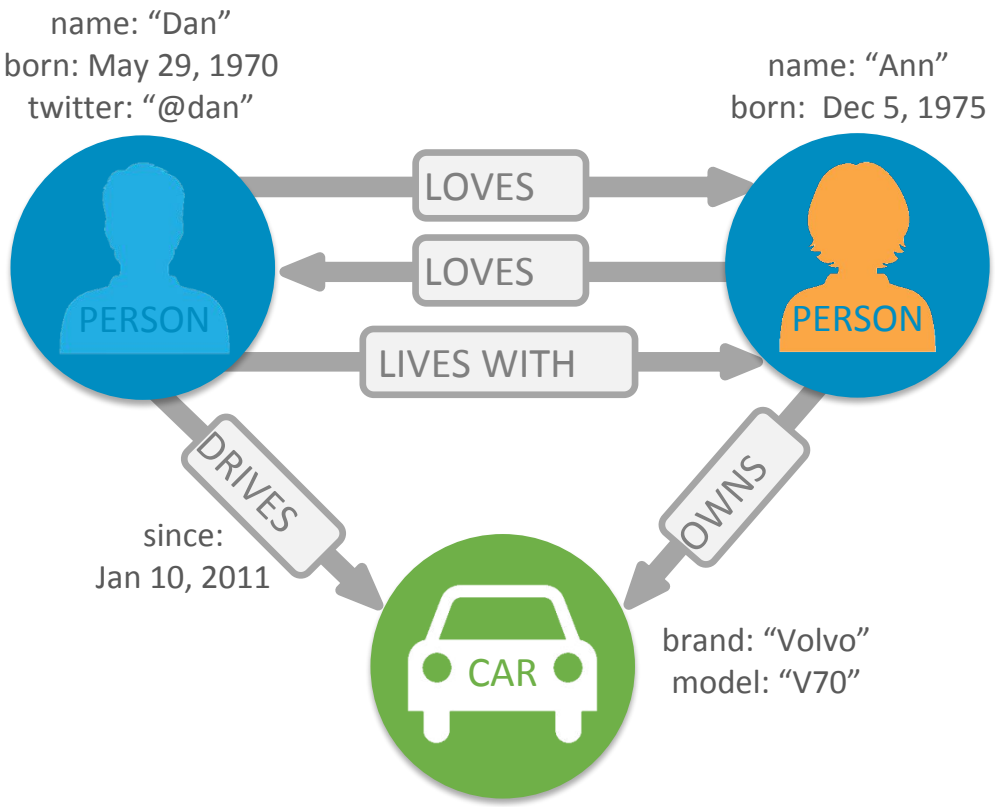
- Represent the objects in the graph
- Can be *labeled*

Relationships

- Relate nodes by *type* and *direction*

Properties

- Name-value pairs that can go on nodes and relationships.

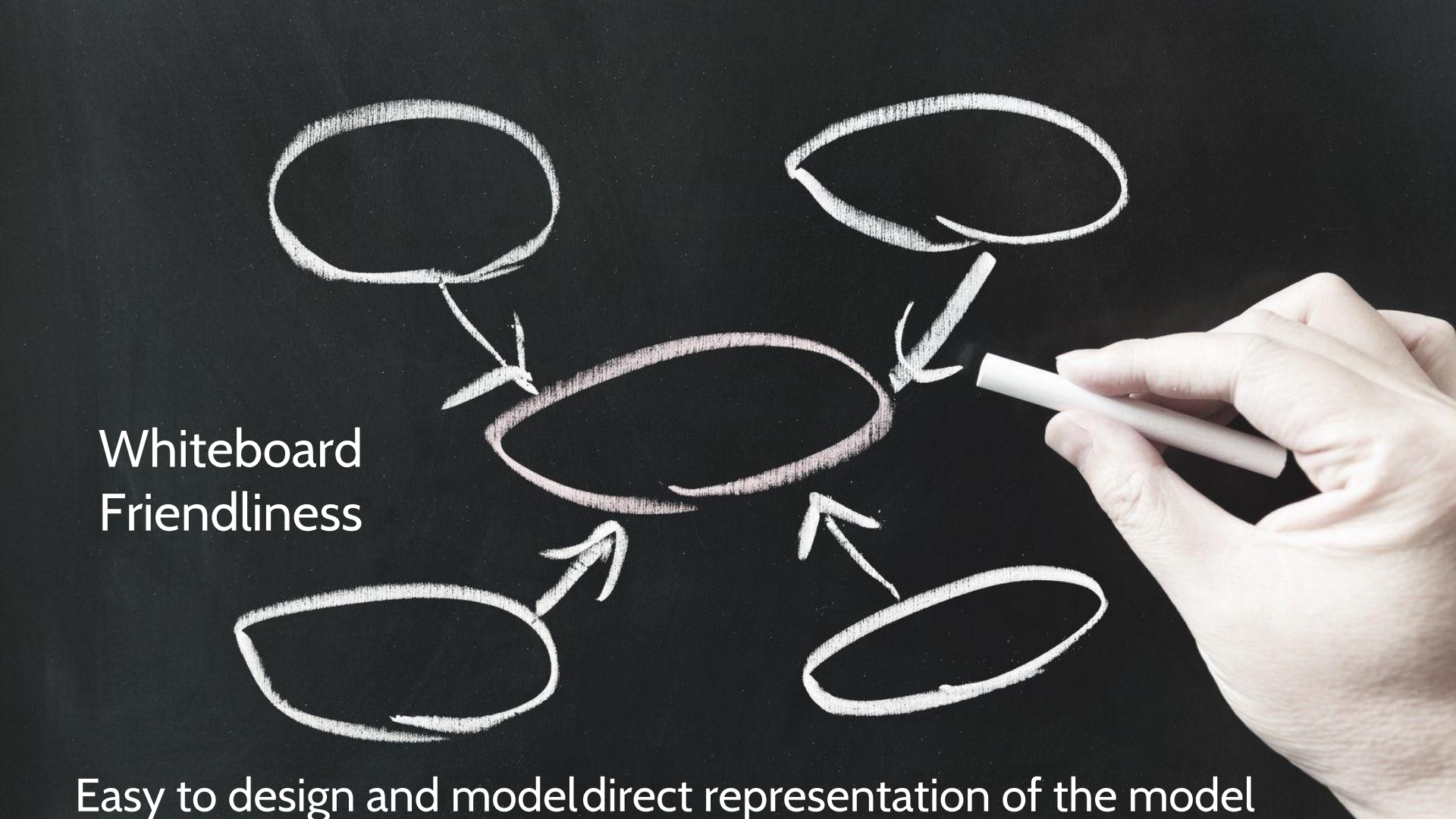


Exercise: Graphs are all around us

Can you identify nodes and relationships in the room?

Summary of the graph building blocks

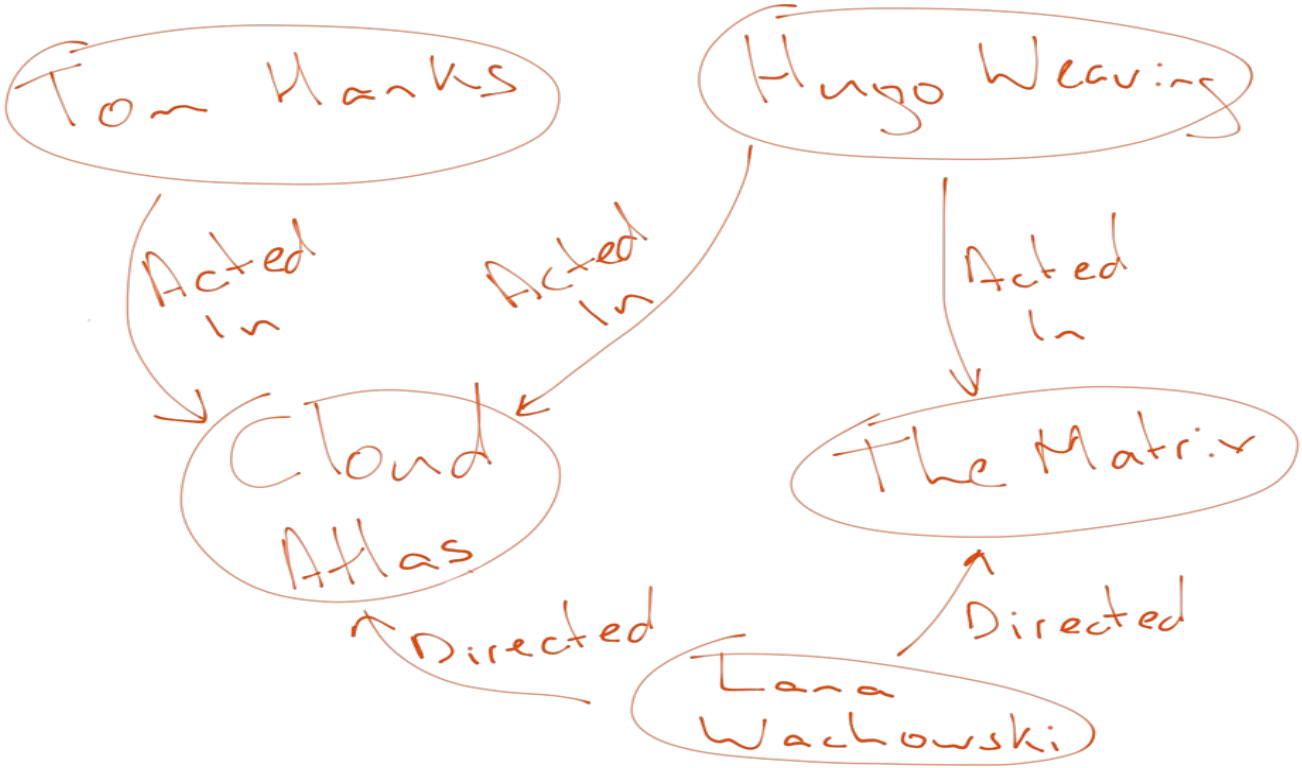
- **Nodes** - Entities and complex value types
- **Relationships** - Connect entities and structure domain
- **Properties** - Entity attributes, relationship qualities, metadata
- **Labels** - Group nodes by role



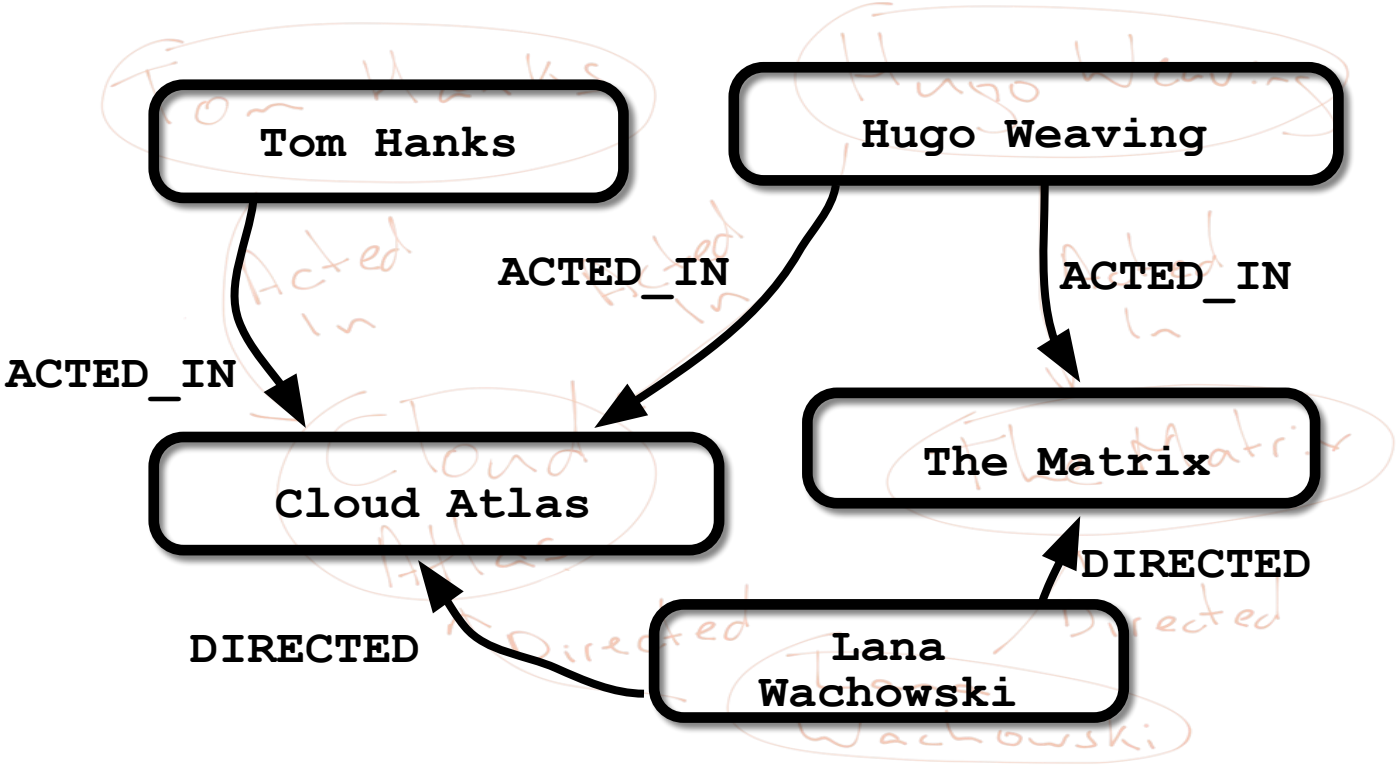
Whiteboard
Friendliness

Easy to design and model direct representation of the model

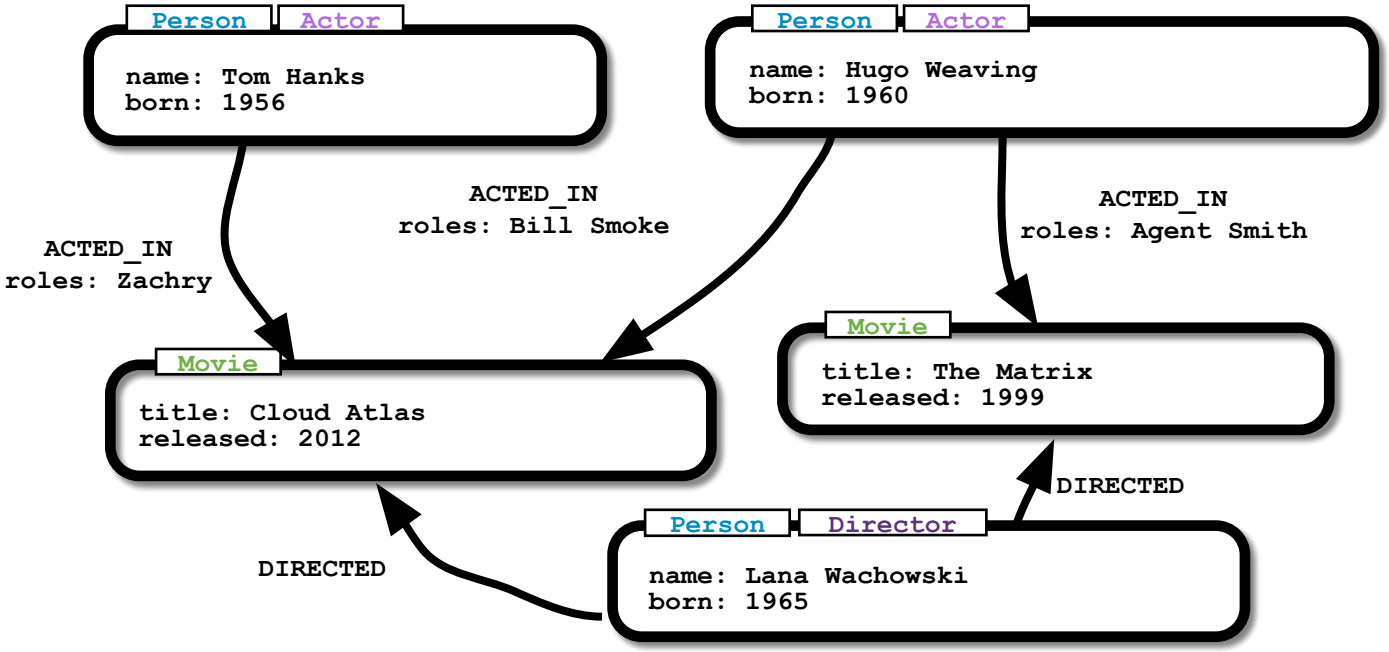
Whiteboard friendliness



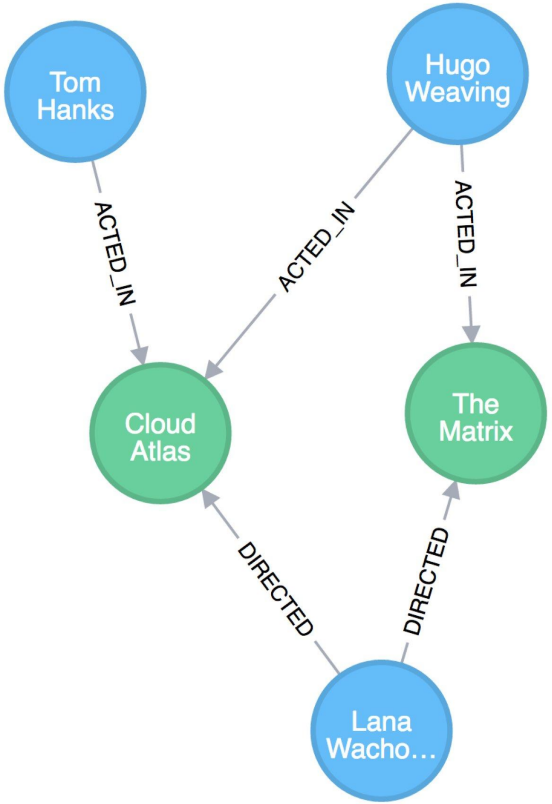
Whiteboard friendliness



Whiteboard friendliness

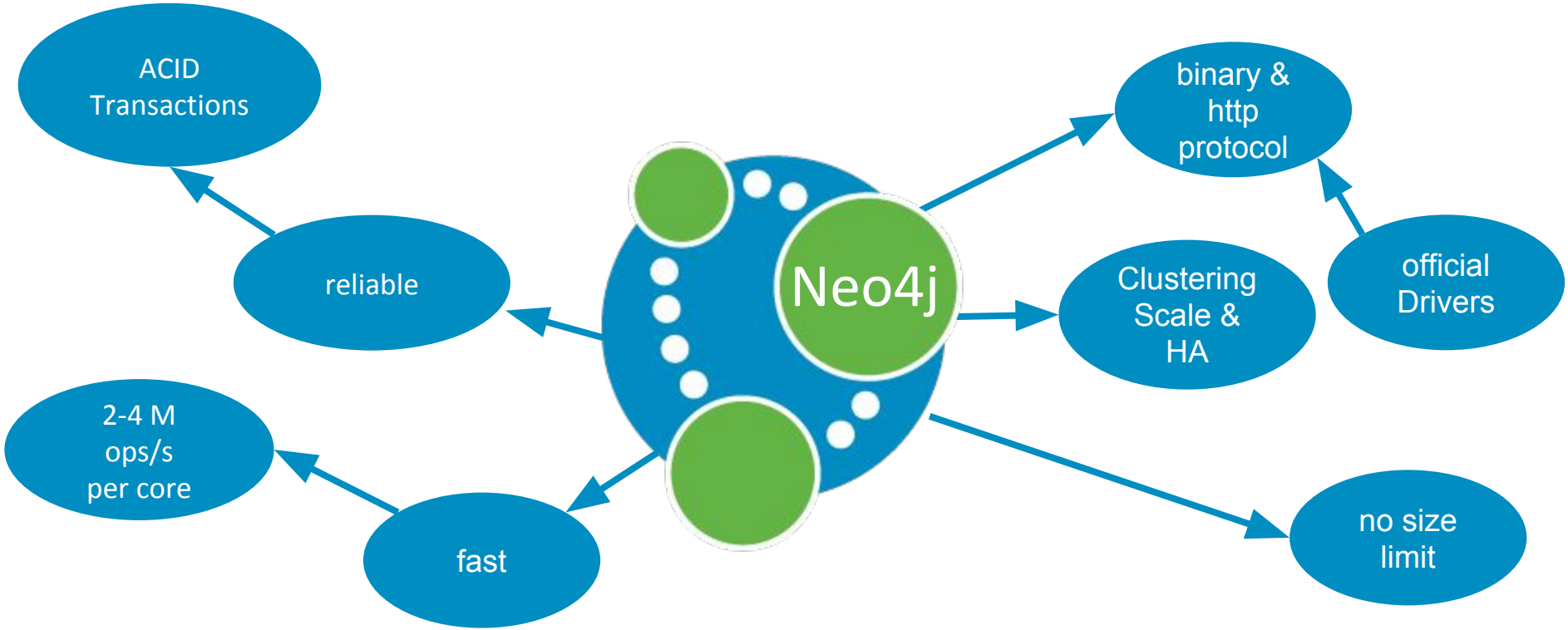


Whiteboard friendliness

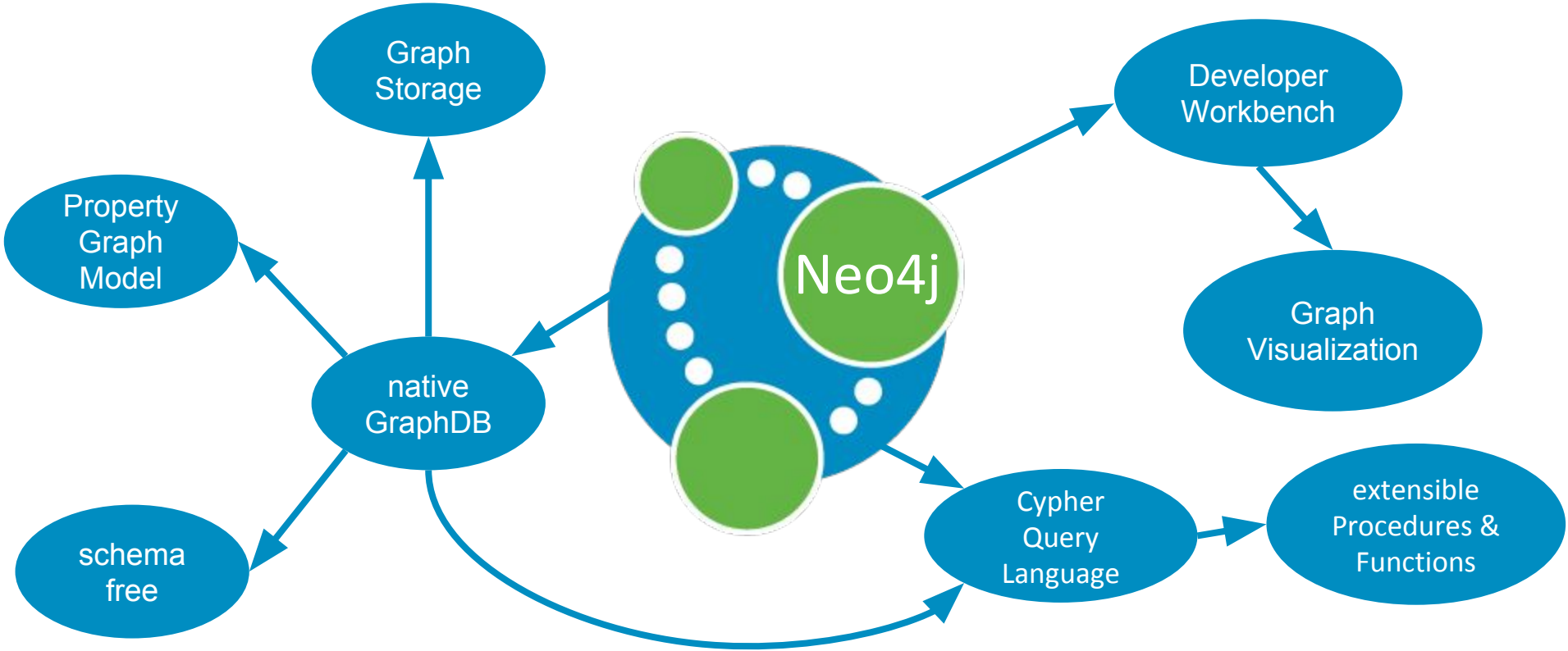


Neo4j is a Graph Database

Neo4j is a database



Neo4j is a graph database



Graph Querying

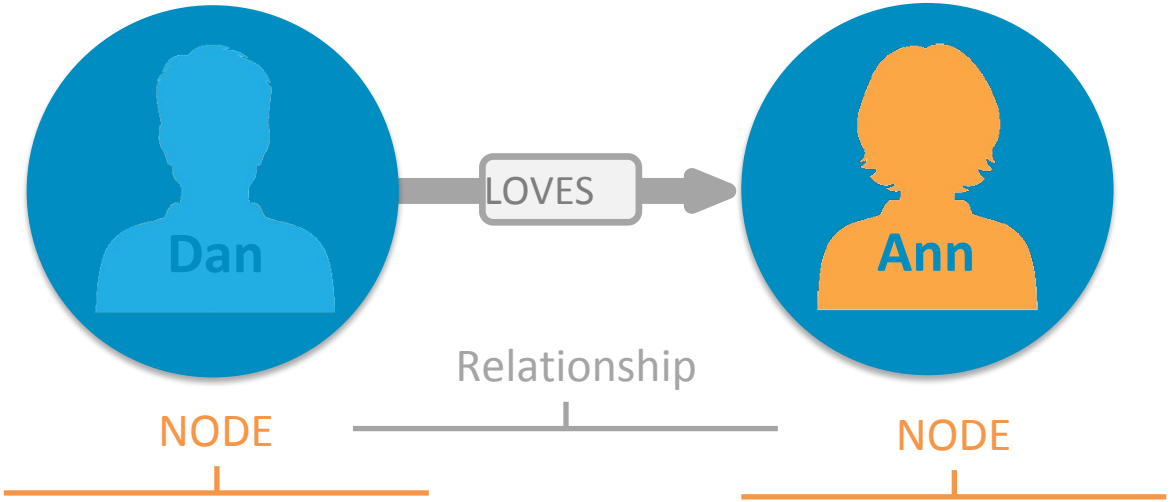
Cypher

A pattern matching query language made for graphs

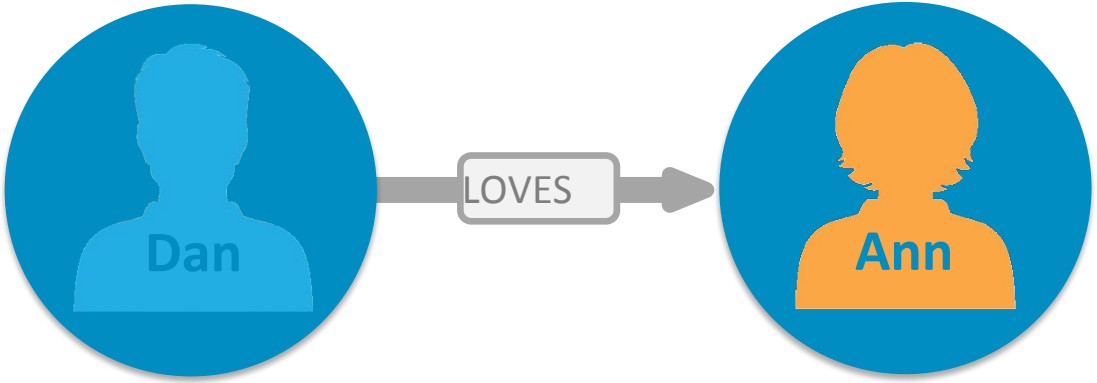
Cypher

- Declarative
  - You tell Cypher what you want, not how to do it
- Expressive
  - Optimize syntax for reading
- Pattern Matching
  - Patterns are easy for your human brain
- Idempotent
  - Updates should be expressed idempotently

Pattern in our Graph Model



Cypher: Express Graph Patterns

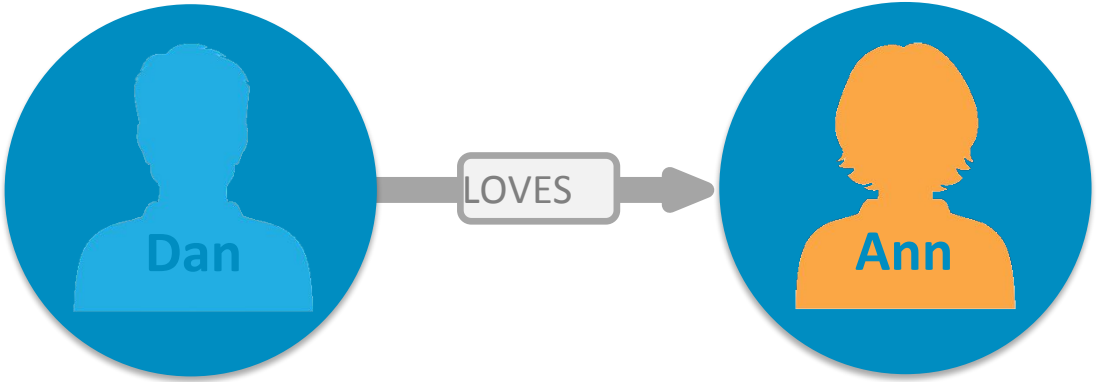


NODE | Relationship | NODE

(:Person { name:"Dan"}) -[:LOVES]-> (:Person { name:"Ann"})

LABEL | PROPERTY | LABEL | PROPERTY

Cypher: CREATE Graph Patterns

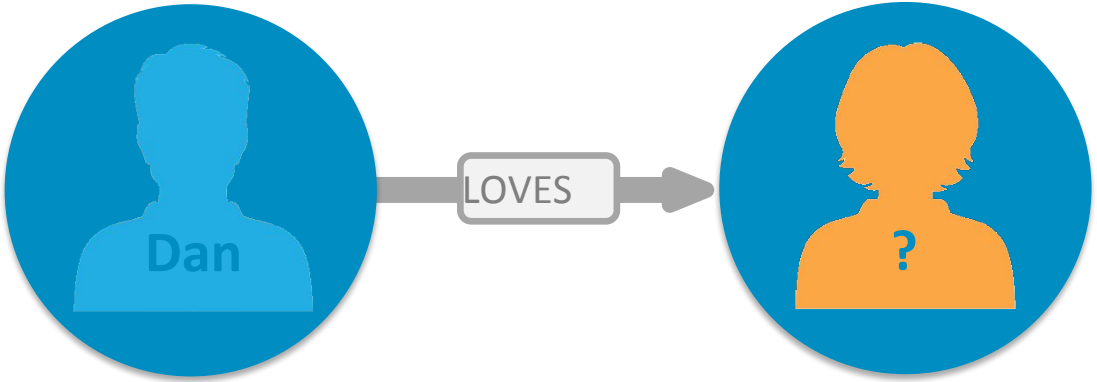


NODE | Relationship | NODE

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

LABEL | PROPERTY | LABEL | PROPERTY

Cypher: MATCH Graph Patterns



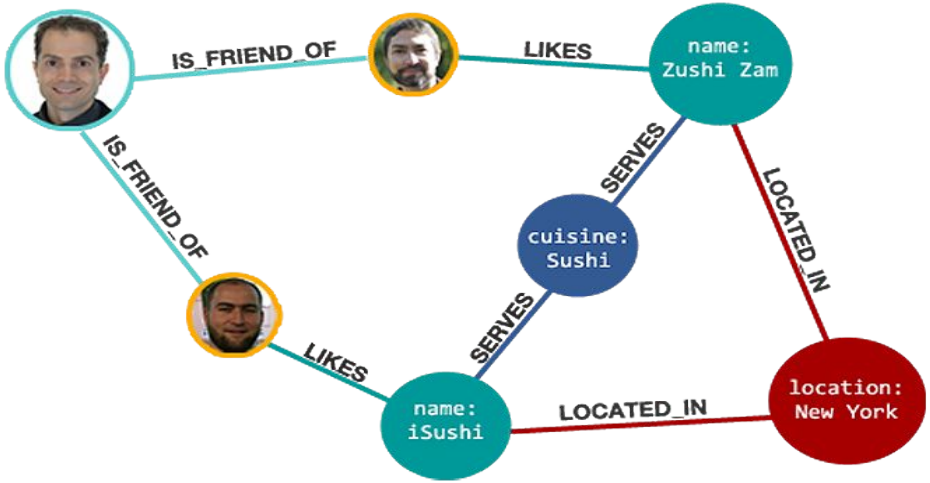
NODE | Relationship | NODE

MATCH (:Person { name:"Dan"}) -[:LOVES]-> (whom) **RETURN** whom

LABEL | PROPERTY | VARIABLE

A graph query example

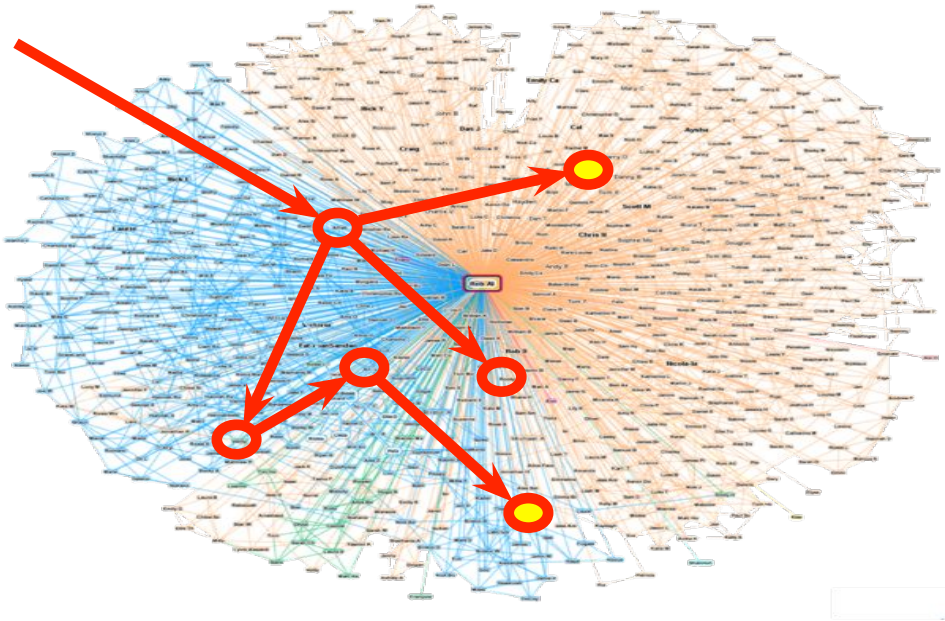
A social recommendation



A social recommendation

```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location),
      (restaurant)-[:SERVES]->(type:Cuisine)
WHERE person.name = 'Philip'
AND loc.location='New York'
AND type.cuisine='Sushi'
RETURN restaurant.name
```

A social recommendation



End of Module
Intro to graphs and Neo4j

Questions ?

First steps in Cypher

Cypher the Graph Query Language

- Made for Graphs (create, update, query)
- All about Patterns
- Using Ascii Art for Patterns
- Declarative, Readable, Expressive
- Powerful Capabilities
- Open Language

Ascii Art: Nodes

`()` or `(n)`

- Surrounded with parentheses
- Use an alias to refer to our node later

`(n:Label)`

- Specify a Label, starts with a colon `:`
- Group nodes by roles or types, think of labels as tags

`(n:Label {prop: 'value'})`

- Nodes can have any properties

Ascii Art: Relationships

neo4j

`-->` or `-[r:TYPE]->`

- Wrapped with hyphens & square brackets
- Like labels, a relationship type starts with a colon `:`

`<  >` Specify the direction of the relationship

`-[:KNOWS {since: 2010}]->`

- Relationships can have properties too!

Ascii Art: Relationships

```
(n:Label {prop:'value'})-[:TYPE]->(m:Label)
```

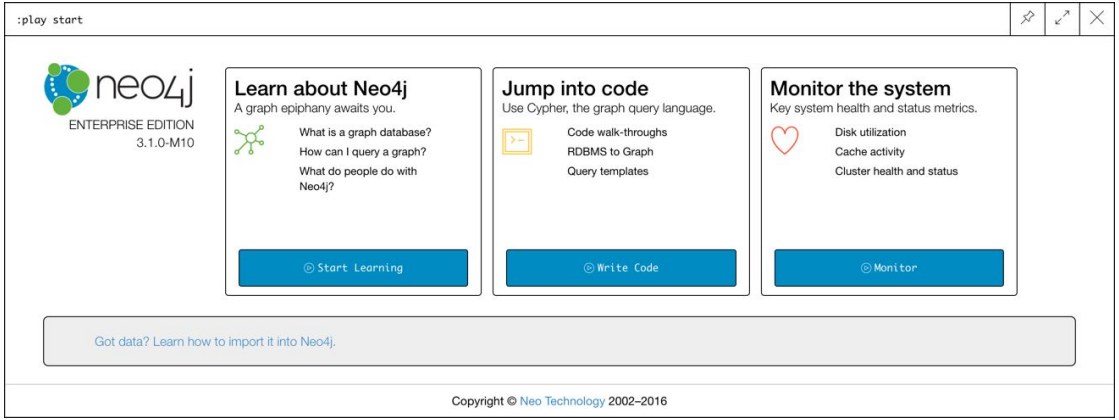
- Mini Graph Examples, that we look for
- Add variables for later use

```
(p1:Person {name:'Alice'})-[:KNOWS]->(p2:Person {name:'Bob')
```

Intro to Cypher

Make sure you've got Neo4j running

1. Start the server.
2. It should be running on: http://localhost:7474
3. Log-in with default credentials
   - user: **neo4j** password: **neo4j**
4. Chose a new password



We're good to go!

Useful resources for this section

- Cypher Reference Card

  http://neo4j.com/docs/cypher-refcard/current/

- Neo4j Developer Pages

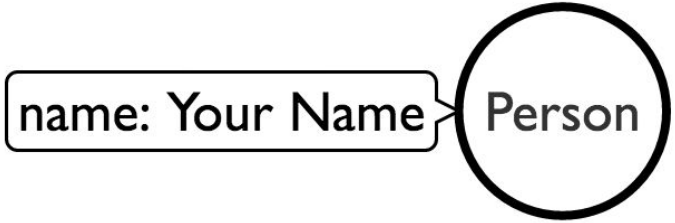
  http://neo4j.com/developer/cypher

- Neo4j Documentation

  http://neo4j.com/docs

Let's try it - CREATE myself!

CREATE a person node within the graph to represent yourself!



Label of `:Person`

Property `:name` should be `'Your Name'`

Welcome to the graph

CREATE statement:

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

Properties are stored as key-value pairs `{key:'Value'}`

Strings, numbers, booleans, and arrays.

Try running the create statement again

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

After you've done that run the following query

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

OOOPS! We created two of me, that's not good.

Constraints

Unique Constraints

We create unique constraints to:

- ensure uniqueness
- allow fast lookup of nodes which match these (label,property) pairs.

Unique Constraints

We create unique constraints to:

- ensure uniqueness
- allow fast lookup of nodes which match these (label,property) pairs.

```
CREATE CONSTRAINT ON (label:Label)
ASSERT label.property IS UNIQUE
```

Creating a constraint

Try creating a constraint on `:Person(name)`:

```
CREATE CONSTRAINT ON (p:Person)
ASSERT p.name IS UNIQUE;
```

Creating a constraint

Try creating a constraint on `:Person(name)`:

```
CREATE CONSTRAINT ON (p:Person)
ASSERT p.name IS UNIQUE;
```

```
Unable to create CONSTRAINT ON ( person:Person ) ASSERT person.name IS UNIQUE:
Multiple nodes with label `Person` have property `name` = 'Your Name':
  node(292)
  node(293)
```

OOOPS! Our data violates the constraint we're trying to create.

Creating a constraint

Get rid of my nodes:

```
MATCH (n)
DETACH DELETE n;
```

Now try to create the constraint again:

```
CREATE CONSTRAINT ON (p:Person)
ASSERT p.name IS UNIQUE;
```

Try running the create statement again

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

Now try it again:

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

Try running the create statement again

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

Now try it again:

```
CREATE (p:Person {name: 'Your Name'})
RETURN p;
```

```
Node 0 already exists with label Person and property "name"=[Your Name]
```

Graph the group

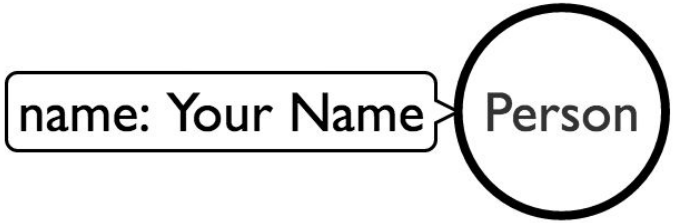
Let's do it together

- Let’s create a **graph of our group** together!
- Open your browser to http://l.neo4j.org/trainer
- And type along with me.
- **Don't be afraid, this is just a sandbox in a playground.**

Your turn: CREATE a node for yourself

CREATE statement:

```
CREATE (p:Person {name: 'Your Name'})
RETURN p
```



Let's find ourselves: `CREATE -> MATCH`

```
MATCH (p:Person {name: 'Your Name'})
RETURN p;
```

SET - and update properties

Find yourself and **SET** your `city` as a property.

```
MATCH (p:Person {name: 'Your Name'})
SET p.city = 'Your City'
RETURN p;
```

Teaser for the course

List people with names and cities

```
MATCH (p:Person)
RETURN p.name, p.city;
```

Aggregation by City

```
MATCH (p:Person)
RETURN p.city, collect(p.name) AS people;
```

Top Cities

```
MATCH (p:Person)
RETURN p.city, collect(p.name) AS people, count(*) AS number
ORDER BY number DESC
LIMIT 3;
```

MERGE - Find Or Create

Let's create ourselves twice. What happens now?

```
CREATE (p:Person {name: 'Your Name'});
```

MERGE to the rescue:

- **MATCH**es the **whole** pattern to find it
- If not found **CREATE** the pattern
- Relies on constraint for strong guarantees

```
MERGE (p:Person {name: 'Your Name'})
RETURN p;
```

Creating Relationships

Most challenging task of the course:
Ask your **right** neighbour for their name.
Find the pair of you:

```
MATCH (p1:Person {name: 'Your Name'})
MATCH (p2:Person {name: 'Your Neighbour'})
RETURN p1, p2;
```

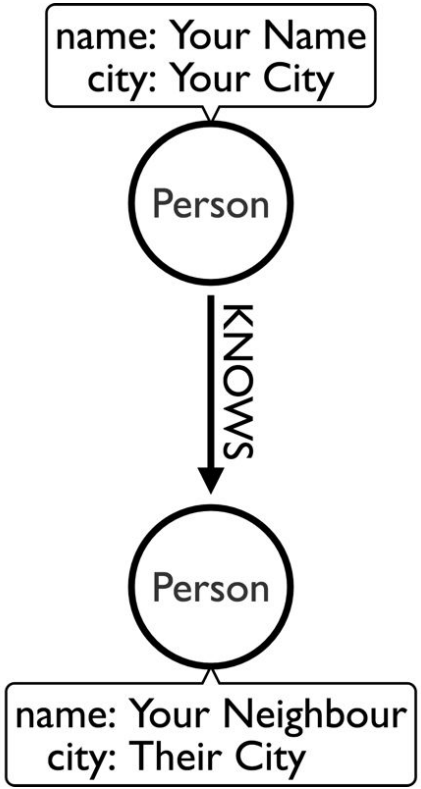
Or in one statement:

```
MATCH (p1:Person {name: 'Your Name'}),
      (p2:Person {name: 'Your Neighbour'})
RETURN p1, p2;
```

CREATEing Relationships

Create a relationship between you and your neighbour:

```
MATCH (p1:Person {name: 'Your Name'})
MATCH (p2:Person {name: 'Your Neighbour'})
CREATE (p1)-[:KNOWS]->(p2);
```



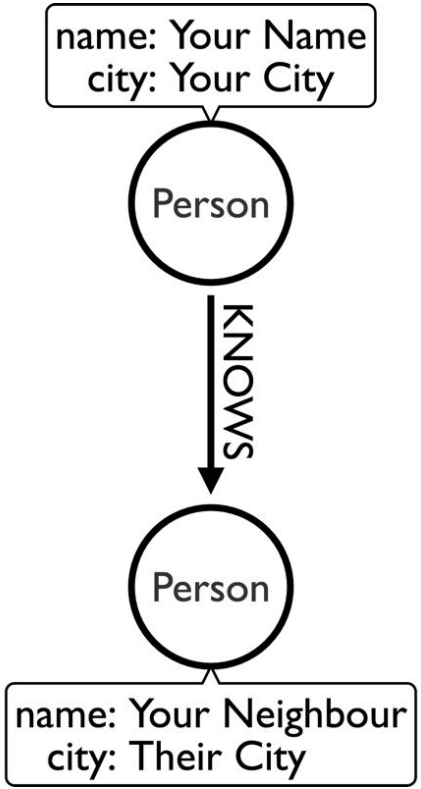
MERGEing Relationships

To avoid **duplicate** Relationships. Use **MERGE**

```
MATCH (p1:Person {name: 'Your Name'})
MATCH (p2:Person {name: 'Your Neighbour'})
MERGE (p1)-[:KNOWS]->(p2);
```

Relationship is created uniquely by

- type
- direction (can be left off)
- properties



Cleaning Up

Let's clean up and delete ourselves

```
MATCH (p:Person {name: 'Your Name'})
DELETE p;
```

OOPS. No dangling pointers. Relationships have to go as well.

```
MATCH (p:Person {name: 'Your Name'})
DETACH DELETE p;
```

End of Module
First steps in Cypher

Questions ?

Data Import with Cypher

Setup our Domain Data

What options do we have to create our data ?

1. Create the statements manually in an editor
2. Save as a Script (`movies.cypher`)
3. And paste or drag it into the Neo4j Browser
4. Execute

In the **real world** we would import the graph from CSV-files.

Let's do that!

Data Entry is Tedious

Meet LOAD CSV

`LOAD CSV` is an ETL Power Tool

- Load CSV data from a **http** or **file** URL
- Provides a **stream** of records (lists or maps)
- Create, update or extend graph structures with the full power of Cypher
- Transactional operations on existing graph in running server
- Transform and convert CSV values
- Allows us to import **into** our graph model
- Up to 10M nodes & relationships work well

This is our CSV Data

movies.csv

title	released	tagline
The Matrix	1999	Welcome to the Real World
Cloud Atlas	2012	Everything is connected
Sleepless in Seattle	1993	"What if someone you never met, someone you never saw, someone you never knew was the only someone for you?"
Apollo 13	1995	"Houston, we have a problem."

This is our CSV Data

people.csv

name	born
Tom Hanks	1956
Keanu Reeves	1964
Meg Ryan	1961
Clint Eastwood	1930

This is our CSV Data

actors.csv

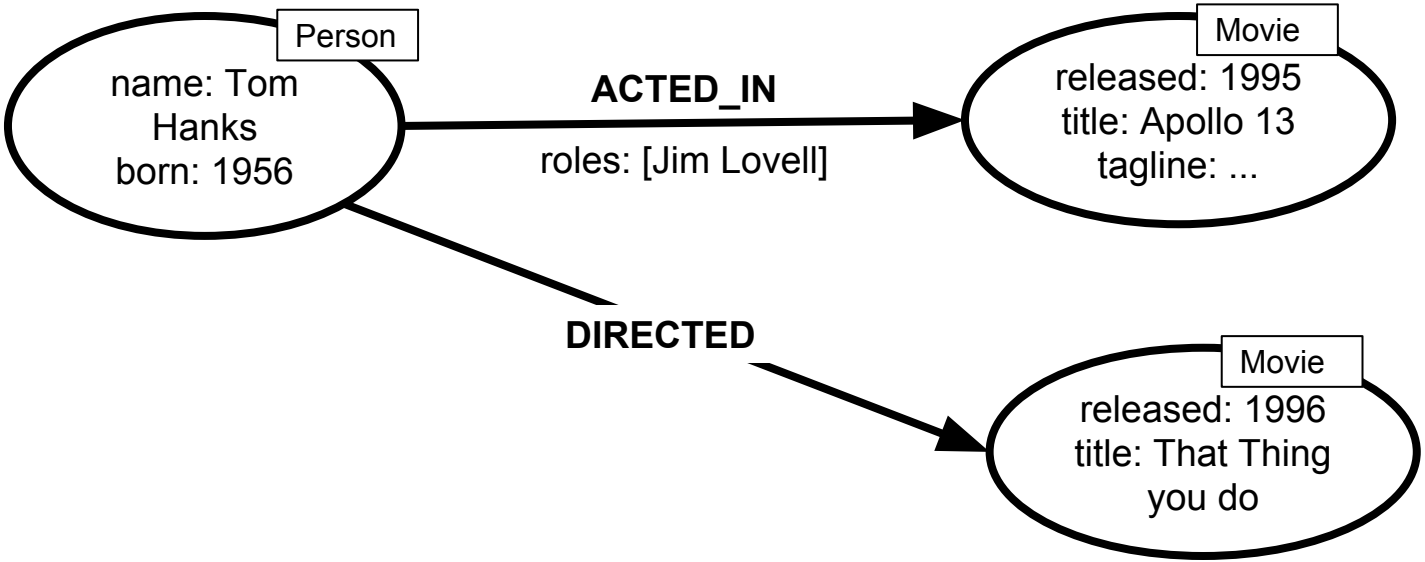
movie	roles	person
Sleepless in Seattle	Sam Baldwin	Tom Hanks
Cloud Atlas	Zachry;Dr. Henry Goose;Isaac Sachs;Dermot Hoggins	Tom Hanks
The Matrix	Neo	Keanu Reeves
Sleepless in Seattle	Annie Reed	Meg Ryan

This is our CSV Data

directors.csv

movie	**person**
Sleepless in Seattle	Nora Ephron
Cloud Atlas	Lilly Wachowski
Cloud Atlas	Tom Tykwer
The Matrix	Lana Wachowski

Remember our Data Model - Movies and Actors



Tabular CSV records to Graph structure

for each record from a file:

- either **Create Node** (Person, Movie)

```
CREATE (:Person {name:row.name, born: toInt(row.born)});
```

- or **Find** start- and end-node (Person, Movie) and **Create Relationship**

```
MATCH (m:Movie {title:row.movie}), (p:Person {name:row.person})
CREATE (p)-[:ACTED_IN {roles:split(r.roles)} ]->(m);
```

Reading data from CSV with Cypher

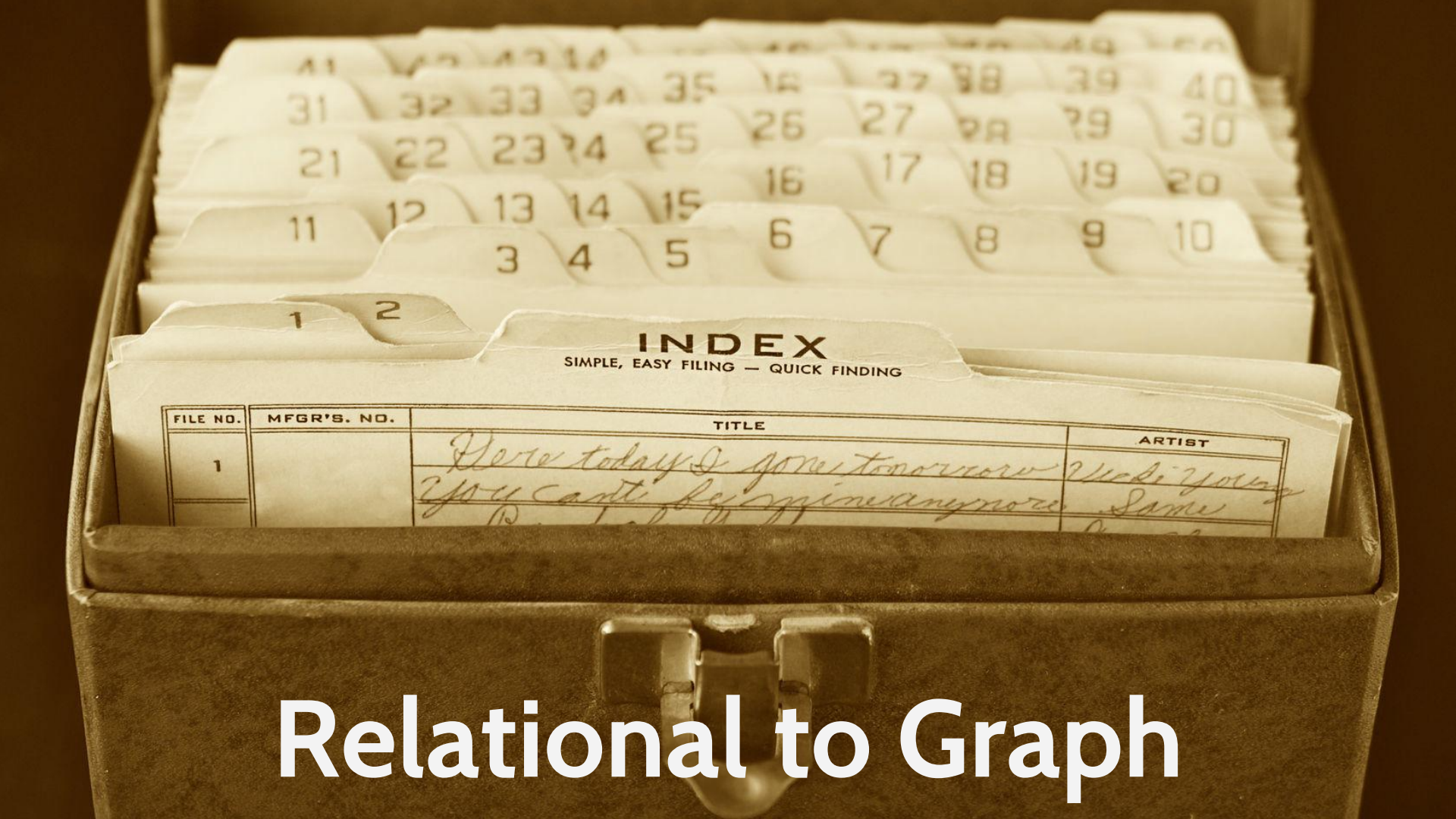
```
[USING PERIODIC COMMIT] // optional transaction batching

LOAD CSV     // load csv data

WITH HEADERS // optionally use first header row as keys in "row" map

FROM "url"   // file:// URL relative to $NEO4J_HOME/import or http://

AS row       // return each row of the CSV as list of strings or map

FIELDTERMINATOR ";" // alternative field delimiter

... rest of the Cypher statement ...
```

Ok, let's try it:

```
:play http://guides.neo4j.com/fundamentals/import.html
```

End of Module
Data Import With Cypher

Questions?

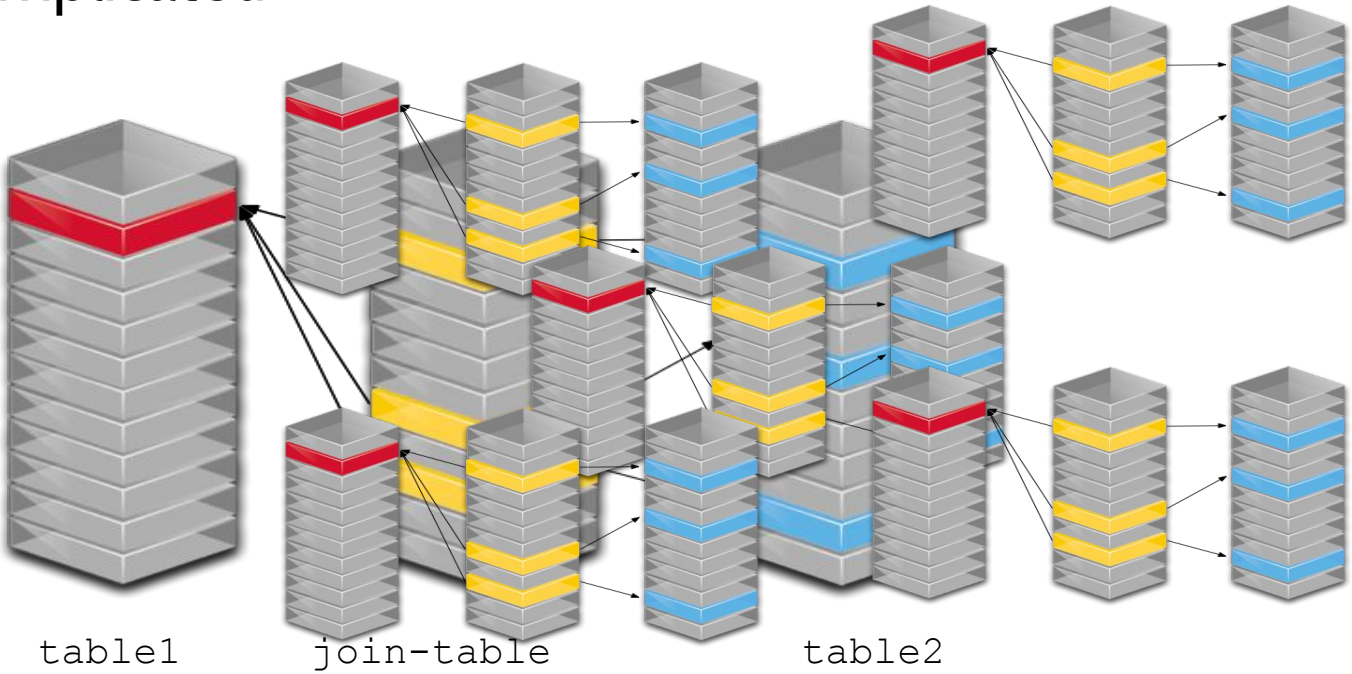


Relational to Graph

Relational to Graph

Relational is simple

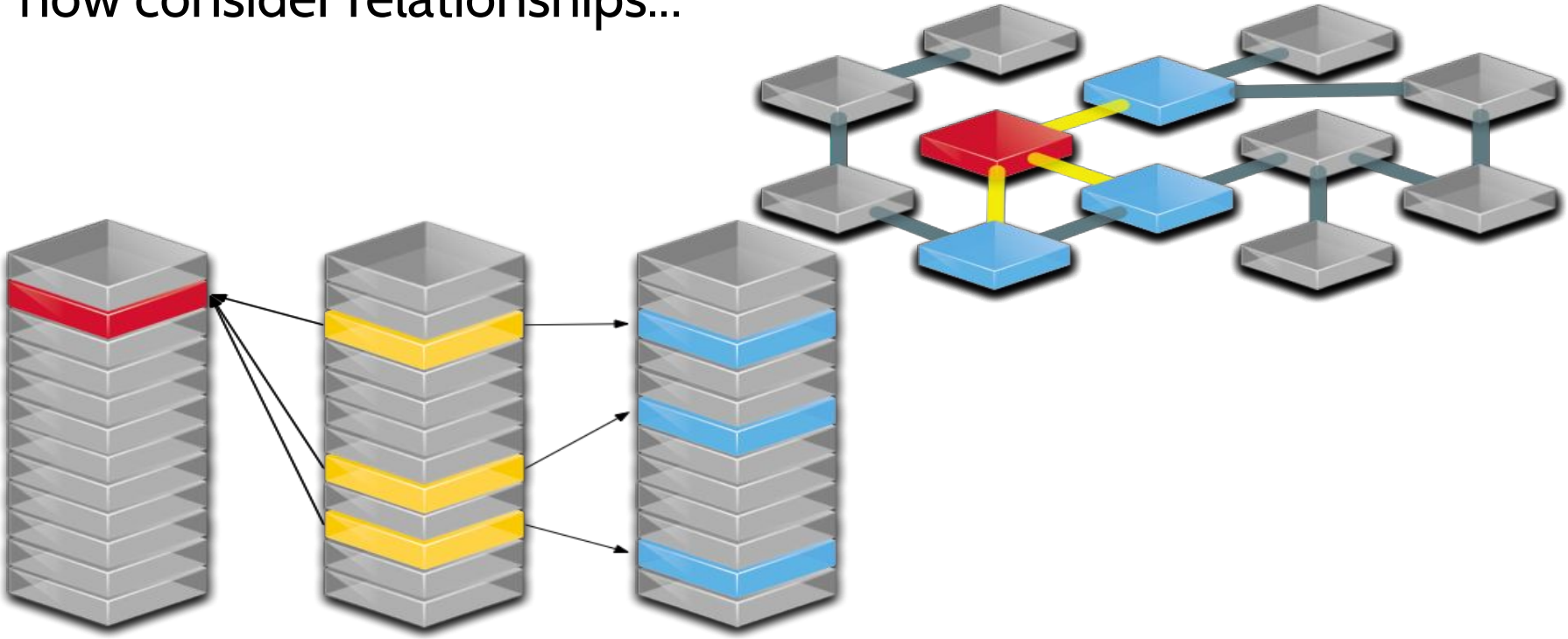
until it gets complicated ...



Relational to Graph

You know relational

now consider relationships...



Normalized Relational Model to Graph Model

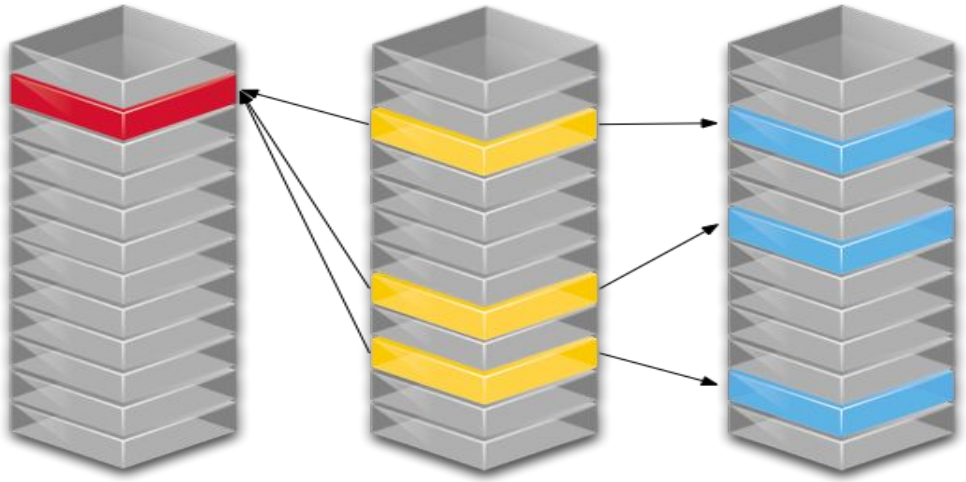
- Entity-Tables become Nodes
- Foreign Keys become Relationships
- Link Tables become Relationships
- Remove artificial Primary Keys and Foreign Keys

Querying your Data

You know how to query
a
Relational Database?

Just use SQL

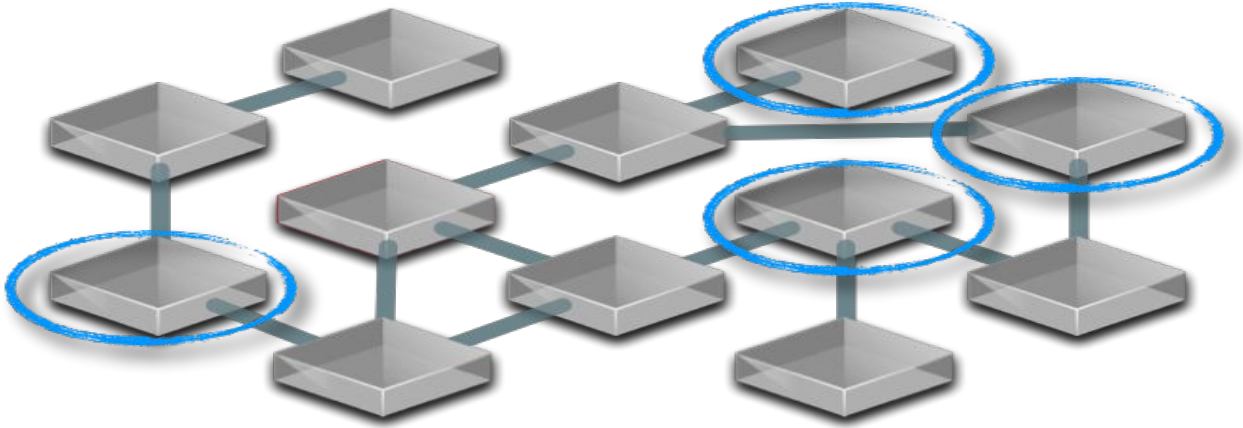
```
select movie.title
from actors join actor_movie on (...) join movies on (...)
where actors.name = "Andreas"
```



How to query a graph?

You traverse the graph

```
// find starting points
MATCH (:Person {name:'Andreas'})
// then traverse the relationships
       -[:ACTED_IN]->(movie:Movie)
      <-[:ACTED_IN]- (other:Person)
// and return results
RETURN other
```



RDBMS Can’t Handle Relationships Well

- Cannot model or store data and relationships without complexity
- Performance degrades with number and levels of relationships, and database size
- Query complexity grows with need for JOINs
- Adding new types of data and relationships requires schema redesign, increasing time to market

… making traditional databases inappropriate when data relationships are valuable in real-time

Express Complex Queries Easily with Cypher

Find all reports and how many people they manage,
each up to 3 levels down

Cypher Query

```
MATCH (boss)-[:MANAGES*0..3]->(mgr)

WHERE boss.name = "John Doe" AND
      (mgr)-[:MANAGES]->()
RETURN mgr.name AS Manager,
  size((mgr)-[:MANAGES*1..3]->()) AS Total
```

SQL Query

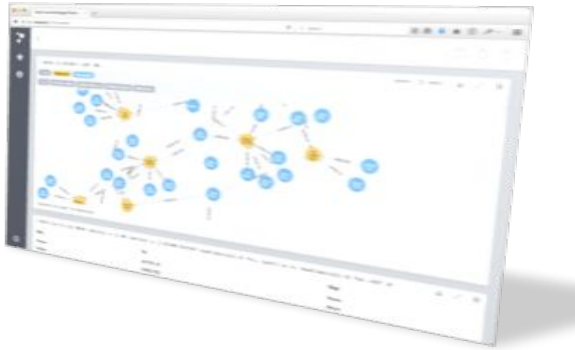
```
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
  FROM person_reportee manager
  WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
  SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT depth1Reportees.pid AS directReportees,
count(depth2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
  FROM(
  SELECT reportee.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.pid AS directReportees, count(L2Reportees.directly_manages)
AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT L2Reportees.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
)
```

Unlocking Value from Your Data Relationships

- Model your data naturally as a graph of data and relationships
- Drive graph model from domain and use-cases
- Use relationship information in real-time to transform your business
- Add new relationships on the fly to adapt to your changing requirements

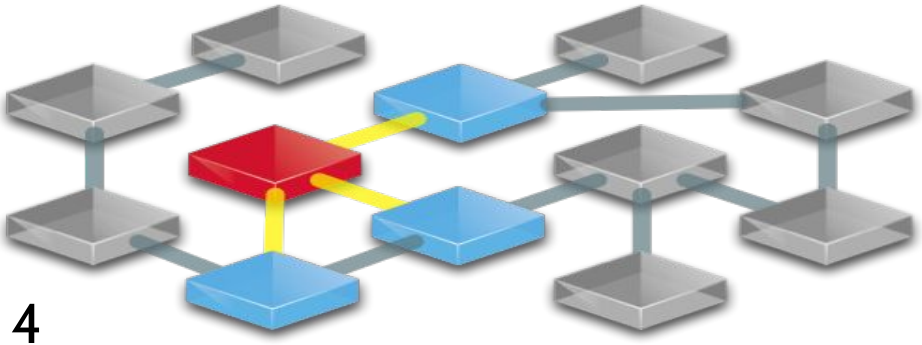
High Query Performance with a Native Graph DB

- Relationships are first class citizen
- No need for joins, just follow pre-materialized relationships of nodes
- Query & Data-locality – navigate out from your starting points
- Only load what's needed
- Aggregate and project results as you go
- Optimized disk and memory model for graphs



Looks different. Who cares?

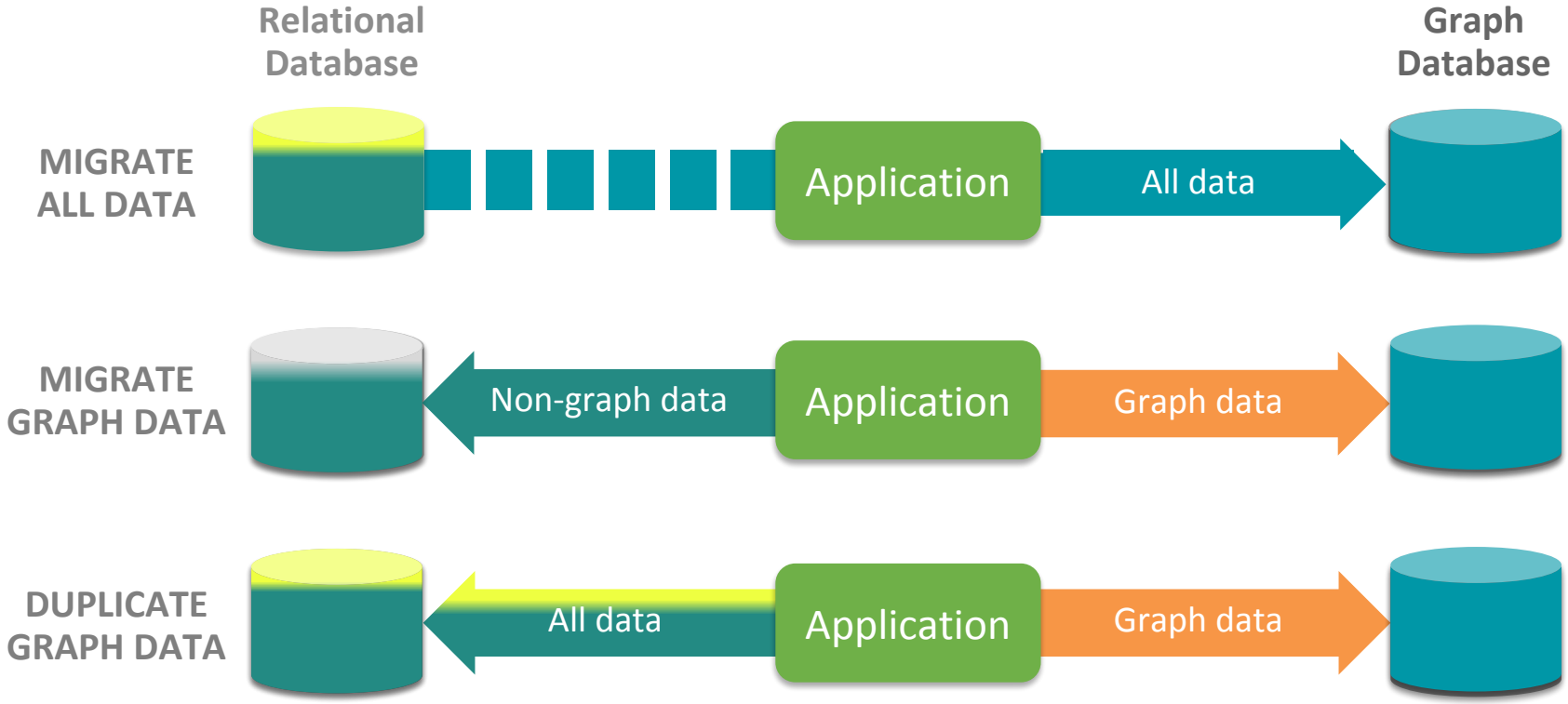
- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O



	# persons	query time
Relational database	1.000	2000ms
Neo4j	1.000	2ms
Neo4j	1.000.000	2ms

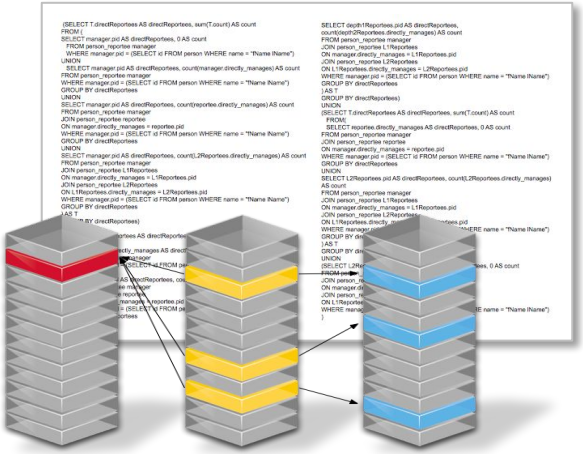
Data Migration / Import

Three Ways to Migrate Data to Neo4j



Accessing Relational Data

- ***Dump to CSV*** all relational database have the option to dump query results and tables to CSV
- ***Access with DB-Driver*** access DB with JDBC/ODBC or other driver to pull out selected datasets
- ***Use built-in or external endpoints*** some databases expose HTTP-APIs or can be integrated (DataClips)
- ***Use ETL-Tools*** existing ETL Tools can read from relational and write to Neo4j e.g. via JDBC

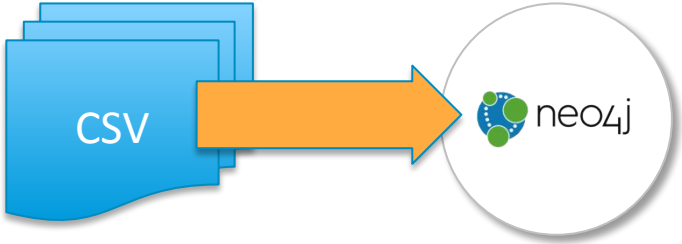


Getting Data into Neo4j: CSV

10
M

Cypher-Based "LOAD CSV" Capability

- Transactional (ACID) writes
- Initial and incremental loads of up to 10 million nodes and relationships
- From HTTP and Files
- Power of Cypher
  - Create and Update Graph Structures
  - Data conversion, filtering, aggregation
  - Destructuring of Input Data
- Transaction Size Control
- Also via Neo4j-Shell / Cypher-Shell

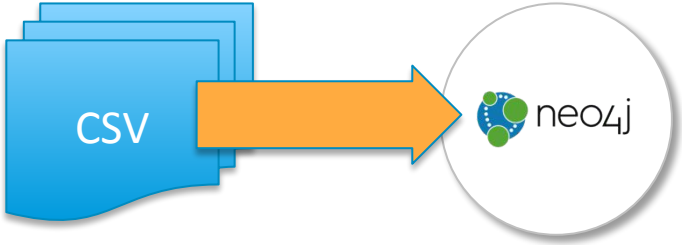


Getting Data into Neo4j: CSV

Command-Line Bulk Loader *neo4j-import*

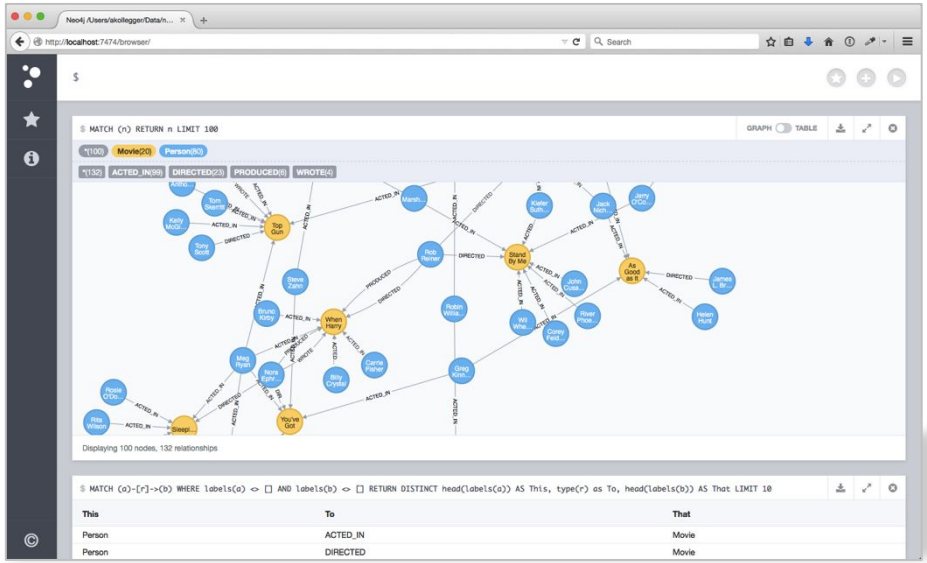
100 B

- For initial database population
- Scale across CPUs and disk performance
- Efficient RAM usage
- Split- and compressed file support
- For loads up to 10B+ records
- Up to 1M records per second



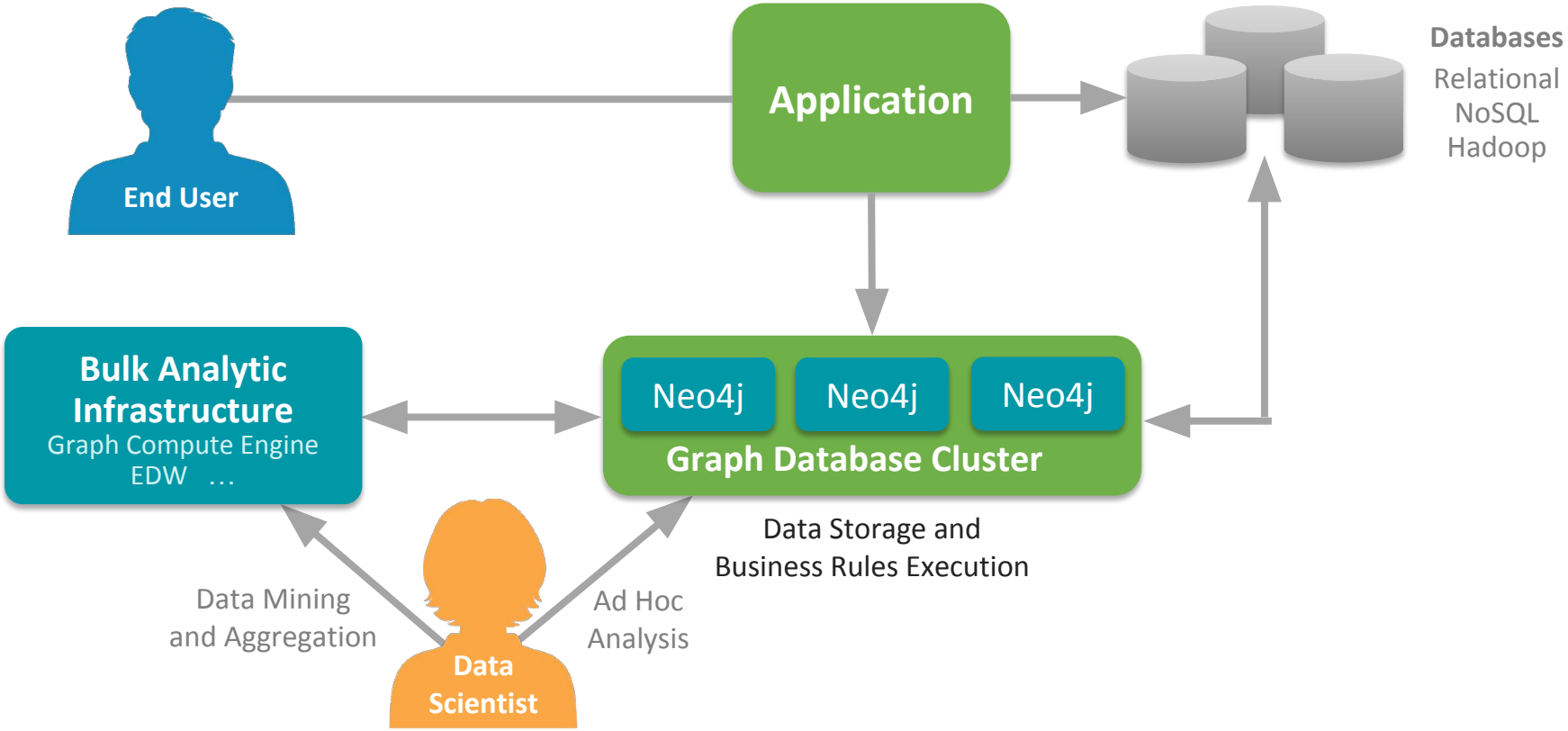
Import Performance: Some Numbers

- Cypher Import 10k-10M records
- Import 100K-100M records per second transactionally
- Bulk import tens of billions of records in a few hours



Architecture / Development

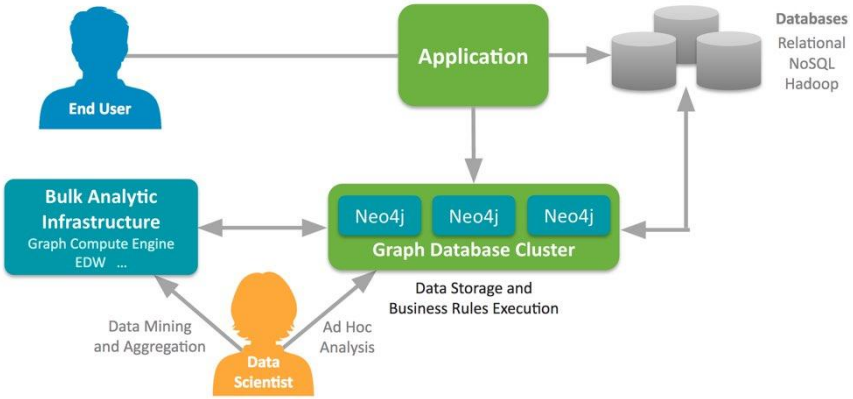
Neo4j Fits into Your Enterprise Environment



Using Neo4j from your Application

In your persistence layer, switch to

- Native Neo4j Drivers with Cypher
- Community Drivers
- Neo4j-JDBC Driver
- Object-graph-mapping library
  - neo4j-ogm
  - Spring Data Neo4j
  - py2neo
  - neoclient (PHP)



- Other libraries available for analytics pipeline, ETL and BI tools

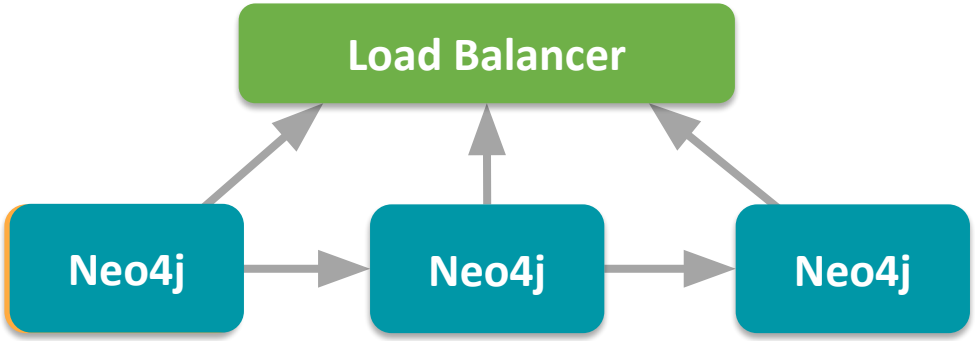
Neo4j Clustering
Architecture Optimized for Speed & Availability at Scale

Clustering Features

- Master-slave replication with ***master re-election*** and ***failover***
- Each instance has its own ***local cache***
- Horizontal scaling & disaster recovery

Performance Benefits

- ***No network hops*** within queries
- ***Real-time operations*** with fast and consistent response times
- ***Cache sharding*** spreads cache across cluster for very large graphs



End of
Relational to Graph

Questions?

Develop an Application

With Neo4j & Cypher

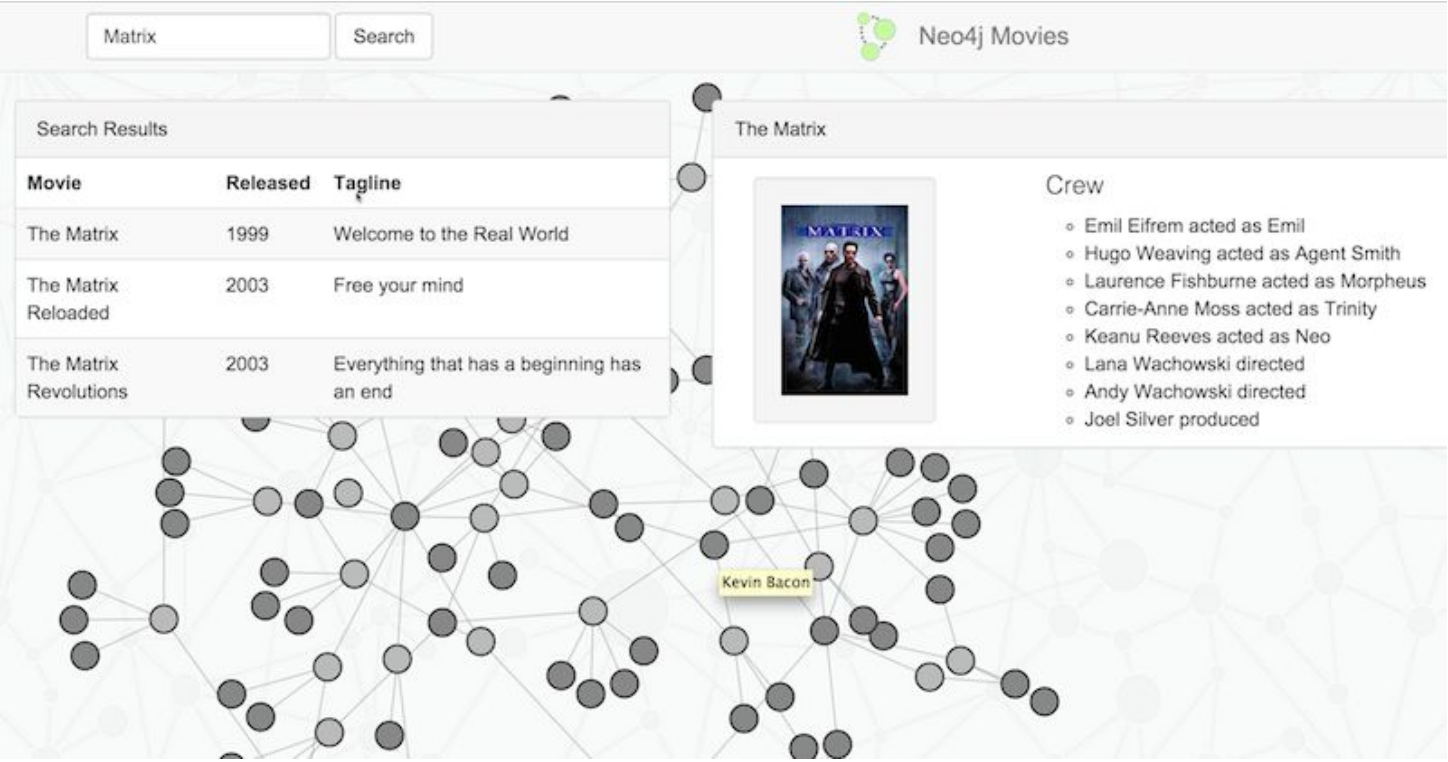
Available APIs

- Remote with Drivers
  - Bolt Binary Protocol (since Neo4j 3.0)
  - Officially Supported Drivers
  - Cypher transactional HTTP Endpoint
- Native Java API
  - User Defined Procedures
  - Execute Cypher
  - Core Java API

More Information on Drivers

- Neo4j Language Guides
  - http://neo4j.com/developer/language-guides
  - Official: Java, Spring, Python, .Net, JavaScript
  - Community: Ruby, PHP, Go, R, Perl, Elixir, Haskell …
- Movie Example Project
  - http://neo4j.com/developer/example-project
- Example Project Implementations
  - https://github.com/neo4j-examples?query=movie
- Clone & Run

Example Project



http://my-neo4j-movies-app.herokuapp.com

Example Project

- Sample Movie Dataset `:play movies`
- same simple jquery-ajax html frontend
- 3 endpoints
  - movie search
  - single movie detail display
  - graph visualization
- backends implemented with each of the language drivers

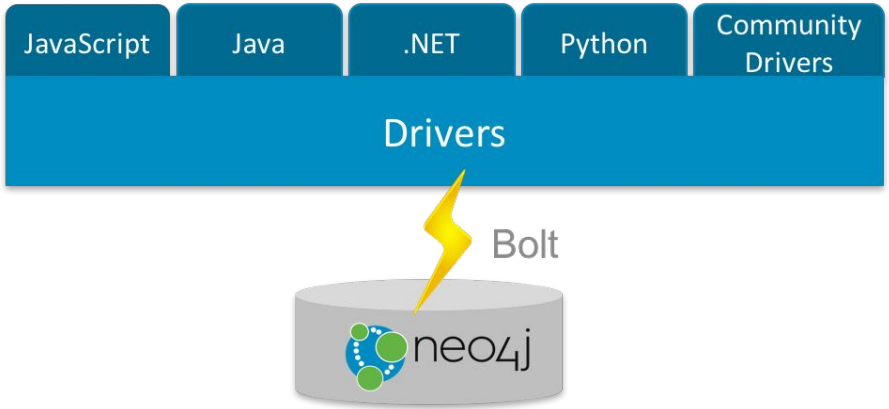
Exercise: Run one sample application

1. Prep your local Neo4j server `:play movies`
2. Grab one example from:

   https://github.com/neo4j-examples?query=movie
3. Git Clone it (or download ZIP)
4. Build and run it against your local server
5. Bonus: Deploy to Heroku / Azure / AWS

Bolt Binary Protocol

- High performance protocol
- Versioned Protocol and API
- Based on Packstream
- Auth and TLS
- Pooling
- Official Drivers: Java, Python, .Net, JavaScript



Java Driver

```
import org.neo4j.driver.v1.*;

Driver driver = GraphDatabase.driver( "bolt://localhost", AuthTokens.basic( "neo4j", "neo4j" )
);
Session session = driver.session();

session.run( "CREATE (a:Person {name:'Arthur', title:'King'})" );

StatementResult result = session.run( "MATCH (a:Person) WHERE a.name = 'Arthur' RETURN a.name AS
name, a.title AS title" );
while ( result.hasNext() )
{
    Record record = result.next();
    System.out.println( record.get( "title" ).asString() + " " + record.get("name").asString()
);
}

session.close();
driver.close();
```

github.com/neo4j/neo4j-java-driver

org.neo4j.driver:neo4j-java-driver:1.0.5

JavaScript Driver

```
var neo4j = require('neo4j-driver').v1;

var driver = neo4j.driver("bolt://localhost", neo4j.auth.basic("neo4j", "neo4j"));
var session = driver.session();
session
  .run( "CREATE (a:Person {name:'Arthur', title:'King'})" )
  .then( function()
  {
    return session.run( "MATCH (a:Person) WHERE a.name = 'Arthur' RETURN a.name AS name, a.title
AS title" )
  })
  .then( function( result ) {
    console.log( result.records[0].get("title") + " " + result.records[0].get("name") );
    session.close();
    driver.close();
  })
```

github.com/neo4j/neo4j-javascript-driver

npm install neo4j-driver

Python Driver

```
from neo4j.v1 import GraphDatabase, basic_auth

  driver = GraphDatabase.driver("bolt://localhost", auth=basic_auth("neo4j", "neo4j"))
  session = driver.session()

  session.run("CREATE (a:Person {name:'Arthur', title:'King'})")
  query = "MATCH (a:Person) WHERE a.name = 'Arthur' RETURN a.name AS name, a.title AS title"
  result = session.run(query)
  for record in result:
      print("%s %s" % (record["title"], record["name"]))

  session.close()
```

github.com/neo4j/neo4j-python-driver

```
pip install neo4j-driver
```

.Net Driver

```
using Neo4j.Driver.V1;

using (var driver = GraphDatabase.Driver("bolt://localhost", AuthTokens.Basic("neo4j",
"neo4j")))
using (var session = driver.Session())
{
    session.Run("CREATE (a:Person {name:'Arthur', title:'King'})");
    var result = session.Run("MATCH (a:Person) WHERE a.name = 'Arthur' RETURN a.name AS name,
a.title AS title");

    foreach (var record in result)
    {
        Output.WriteLine($"{record["title"].As<string>()} {record["name"].As<string>()}");
    }
}
```

github.com/neo4j/neo4j-dotnet-driver

```
PM> Install-Package Neo4j.Driver-1.0.3
```

Extend Cypher - User Defined Procedures

- User Defined Procedures
- 80:20 high performance operations
- Methods annotated with `@Procedure` [`@PerformsWrites`]
- Returns `Stream` of DTOs
- Automatic Transaction Scope
- Callable within Cypher
- Deploy jar in `$NEO4J_HOME/plugins directory`
- Use of embedded Java-APIs

Neo4j Procedure (Embedded) API - Cypher

```
@Context GraphDatabaseService db;

...

String query =

"MATCH (:Person {name:{name}})-[:ACTED_IN]-(m:Movie) RETURN m.title as title";


Result result = db.execute(query, map("name","Clint Eastwood"));

for (Map<String,Object> row : result) {

   String dept = (String)row.get(dept);

   ...

}
```

User Defined Procedure Example

```
@Context GraphdatabaseService db;

public Stream<NodeResult> lookup(@Name("ids") List<Long> ids) {
   return ids.map( i -> new NodeResult(db.getNodeById(i)));
}
public static class NodeResult {
    public final Node node;
}
CALL proc.lookup([1,2,3]) YIELD node
RETURN node
```

Neo4j Procedure API - Nodes & Rels

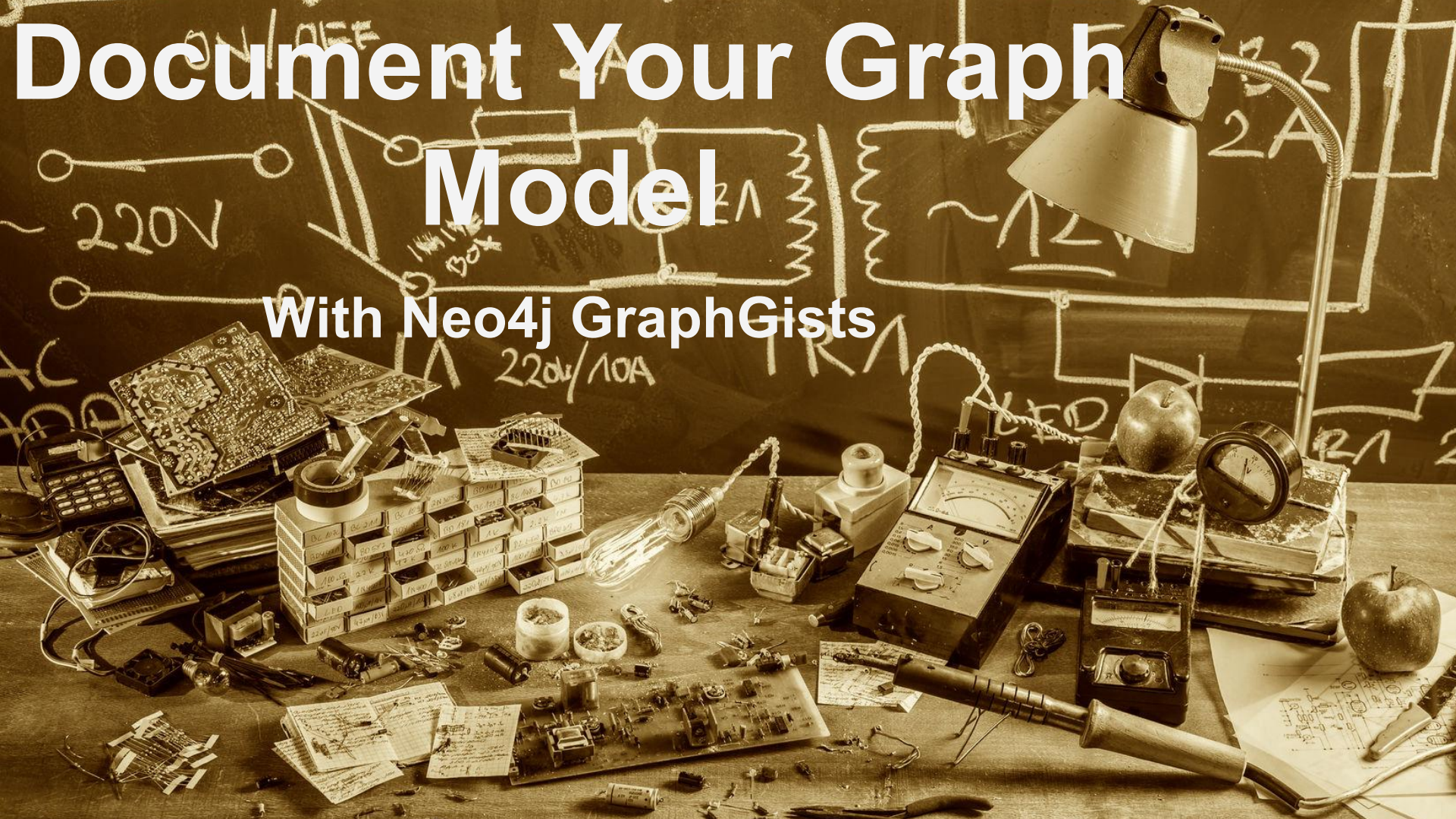
```
@Context GraphDatabaseService db;

...

   Node john        = db.createNode(Label.label("Person"));

                      john.setProperty("name","John Claude Van Damme");

   Node movie       = db.findNode(Label.label("Movie"),"title","Karate Tiger"));

   Relationship rel = john.createRelationshipTo(movie,

                                                RelationshipType.withName("ACTED_IN"));
```

End of
Develop an Application

Questions ?

Document Your Graph Model

With Neo4j GraphGists



Graph Model and Example Sharing & Collaboration

A tool to:

- quickly sketch out graph models
- document and discuss use-cases
- incrementally build the graph
- life queries with error checking
- shareable and collaborative
- easy to write, read, manage and version

Where can you use Graph Gists?

- Documentation
- Evolution
- Discussion
- Feedback
- Versioning
- Sandbox
- Answering Questions
- much more …

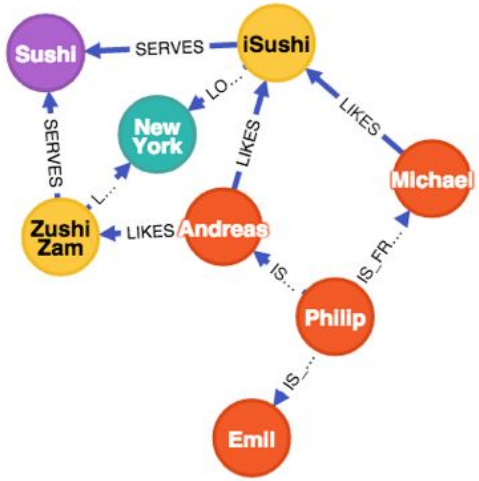
What is a Graph Gist?

simple text file with:

- a graph domain model (Cypher)
- describing text and pictures
- example queries run against the model (Cypher)
- tabular and graph result rendering
- interactively executable
- written in plain text AsciiDoc Markup (similar to MarkDown)
- rendered in your Browser

Example: Restaurant Recommendations

Recommendation Graphgist



Philips Friends

Query 2

```
MATCH (philip:Person {name:"Philip"})-[:IS_FRIEND_OF]-(person)
RETURn person.name
```

Test run OK

person.name

Setup the Graph Example

Header

```
== Restaurant Recommendations
```

Prose

```
 We want to demonstrate how easy it is to model a domain as a graph and answer questions
  in almost natural language.
 Here we use a Domain of restaurants which serve cuisines and are located in a Cit
```

Image

```
 image::https://dl.dropboxusercontent.com/u/14493611/sushi_restaurants_nyc.svg[]

 == Setup: Creating Friends, Restaurants in Cities and their Cusines

 //setup
 [source,cypher]
 ----
```

Cypher Code Block - **Setup**

```
 CREATE (philip:Person {name:"Philip"})-[:IS_FRIEND_OF]->(emil:Person {name:"Emil"}),
        (philip)-[:IS_FRIEND_OF]->(michael:Person {name:"Michael"}),
        (philip)-[:IS_FRIEND_OF]->(andreas:Person {name:"Andreas"})
 ...
 ----
```

Render **Full Graph**

```
 //graph
```

Render Use-Case Queries

Header

```
=== Restaurants in NYC and their cusines

 [source,cypher]
 ----
 MATCH (nyc:City {name:"New York"})<-[:LOCATED_IN]
       -(restaurant)-[:SERVES]->(cusine)
 RETURN nyc, restaurant, cusine
 ----

 //table
 //graph_result
```

Cypher Code Block - Query

Render **Tabular Result**

Render **Graph Result**

Let’s create a GraphGist together

Collaborate on a GraphGist

1. Open this Google Doc: http://bit.ly/neo_gdoc
2. It is rendered in this GraphGist: http://bit.ly/neo_gist

Collaborate on a GraphGist

1. Now let's add ourselves again like we did in the beginning.
2. Add yourself as a `Person` with a `name`
3. with your `city` to the setup section
4. Give yourself an variable
5. Refresh the GraphGist
6. Create `KNOWS` relationship to your neighbor
7. Add more use-case queries for finding patterns and aggregation on city
8. Add prose and an image

See which Graph Use Cases are documented

Spend 10 minutes looking at the existing GraphGists that interest you.

Also check their sources!

Go To:

http://neo4j.com/graphgists

Task: Tell us afterwards which GraphGist you found most interesting and why?

5 Example Use Cases

Let’s look at some interesting GraphGist examples in detail

- Bank Fraud Detection
- Books Management Graph
- Analyzing Offshore Leaks
- Network Dependency Graph
- Job Recommendation System

Create GraphGists Yourself

- Go to the GraphGists Author Portal http://portal.graphgist.org
- See the full GraphGists collection for inspiration or useful templates
- Check out the "What is a GraphGist" page for syntax details
- Look at the blank Template

End of
Documenting Graph Models

Questions?

More Ways to Learn about Neo4j

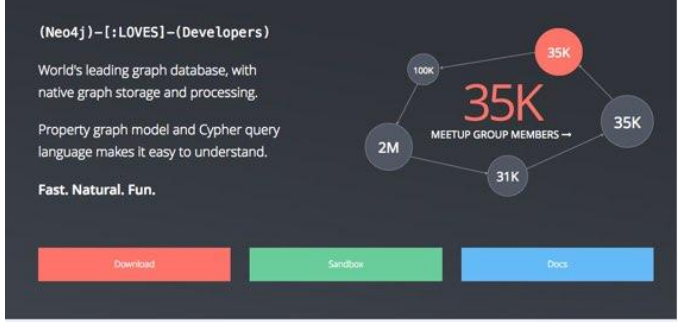
Resources & Links

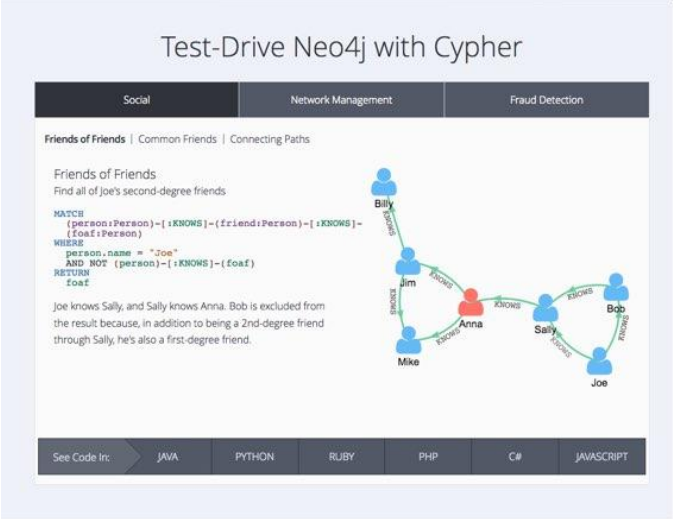
13-14 OCTOBER
SAN FRANCISCO
GRAPHCONNECT SF 2016
Connecting the World of Graphs
REGISTER

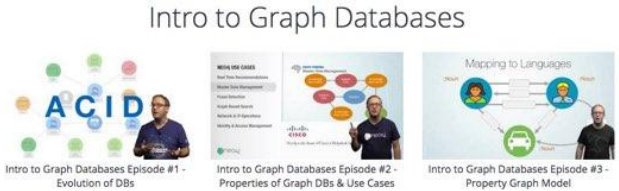
Developer Pages

neo4j.com/developer

Intro to Graphs, RDBMS to Graphs
Data Import & Data Modeling
Language Guides & Drivers
Visualization
Neo4j Ecosystem



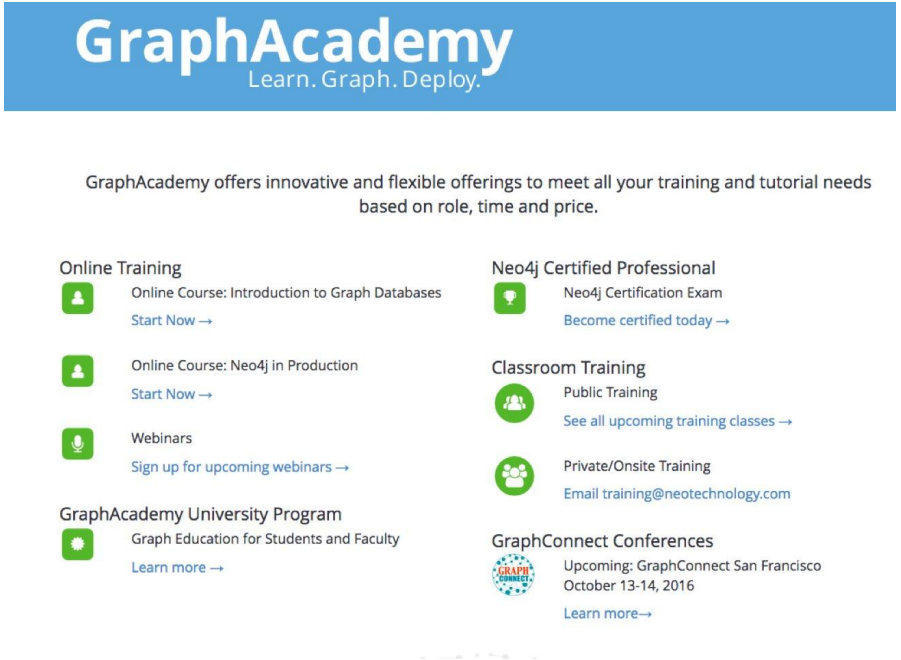




GraphAcademy

neo4j.com/graphacademy

Online & Classroom Training
University Program
Certification
Webinars
Graph Days, GraphTalks



Neo4j Certified Professional

neo4j.com/graphacademy/neo4j-certification/

1 hour certificate exam covers Cypher, Data Modeling, Clustering, etc.

Cypher Reference Card

neo4j.com/docs/cypher-refcard/current/

Developer Documentation

Neo4j Documentation

Neo4j 3.0 Docs

Neo4j Developer Manual	HTML
Neo4j Developer Manual for .NET	PDF
Neo4j Developer Manual for Java	PDF
Neo4j Developer Manual for JavaScript	PDF
Neo4j Developer Manual for Python	PDF
Neo4j Operations Manual	HTML | PDF
Cypher Refcard	HTML | PDF
Neo4j Java Developer Reference	HTML | PDF
Neo4j REST API documentation	HTML | PDF
Neo4j Object Graph Mapping OGM Library	HTML | PDF

neo4j.com/docs

neo4j.com/docs/developer-manual/current
neo4j.com/docs/operations-manual/current

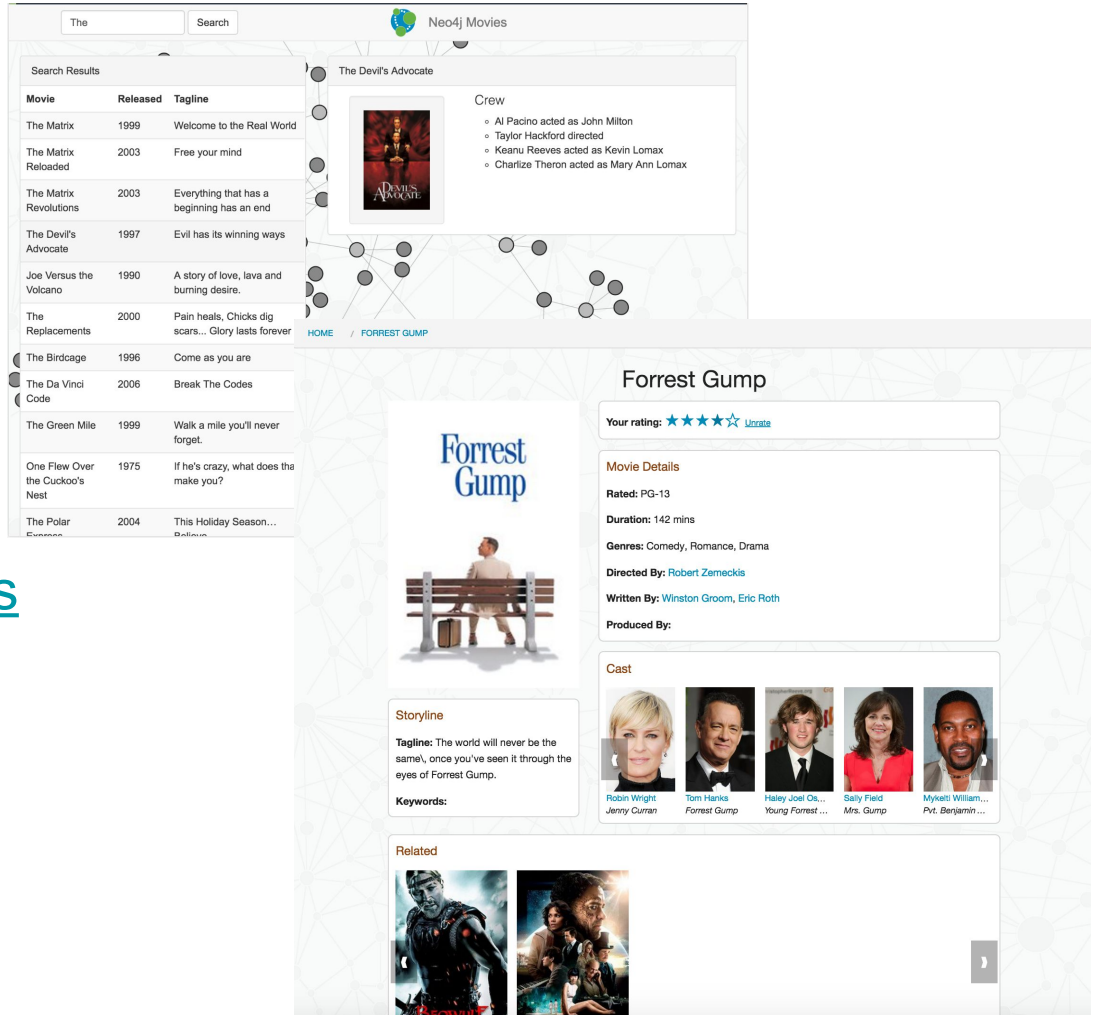
Knowledge Base

neo4j.com/developer/kb

- Frequently Asked Questions and Answers
- Canonical Source
- Maintained by our Support and Field team
- Tagged,
- Version specific

Example Applications

github.com/neo4j-examples

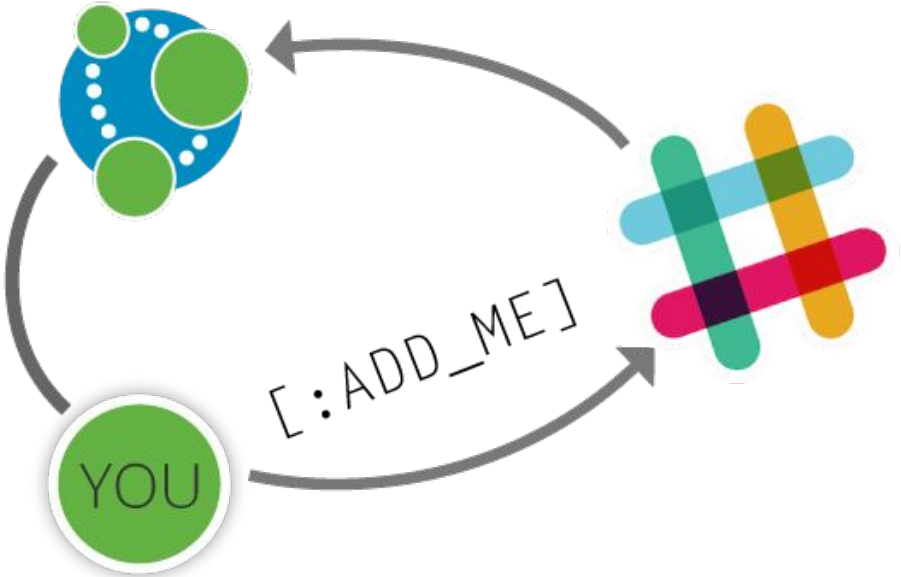


Meetup Groups

neo4j.meetup.com

neo4j-users Slack

neo4j.com/slack



GitHub

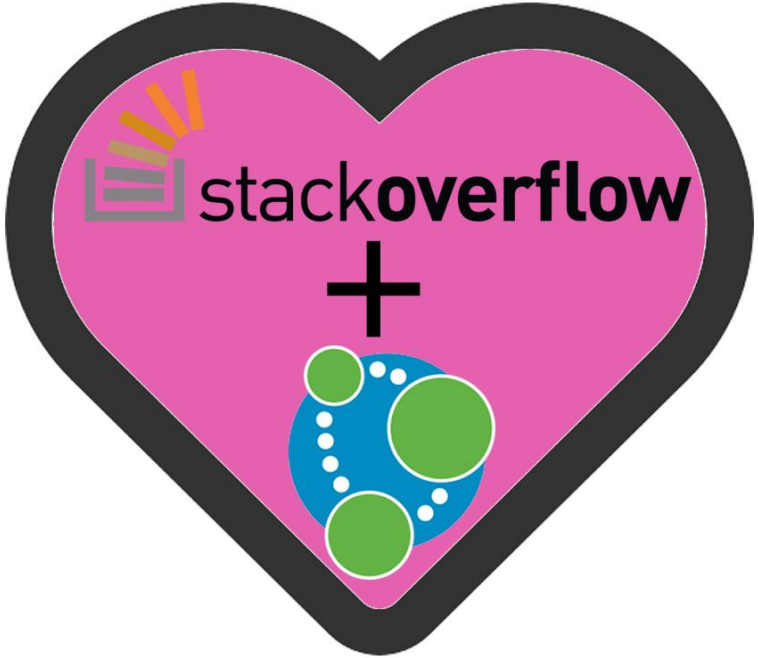
github.com/neo4j

github.com/neo4j-contrib

github.com/neo4j-examples

StackOverflow

stackoverflow.com/questions/tagged/neo4j

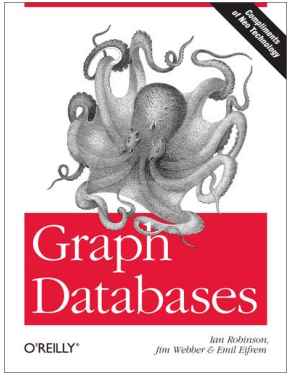


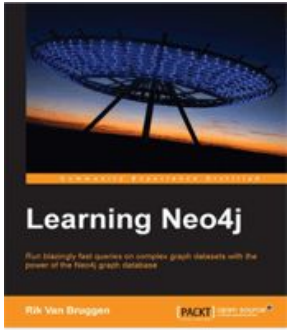
Books

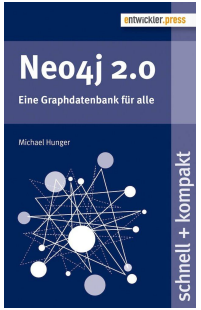
neo4j.com/books

Free, high value e-books

Third party publications







Beginning Neo4j

Learn More →

Building Web Applications with Python and Neo4j

Learn More →

Connected

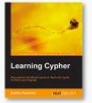
Learn More →

Good Relationships

Learn More →

Graph Databases (Japanese Version)

Learn More →



Learning Cypher

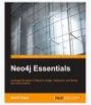
Learn More →

Neo4j Cookbook

Learn More →

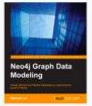
Neo4j Des données et des graphes – Prise en main (2e édition)

Learn More →



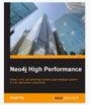
Neo4j Essentials

Learn More →



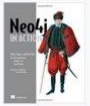
Neo4j Graph Modeling

Learn More →



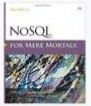
Neo4j High Performance

Learn More →



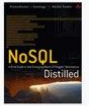
Neo4j in Action

Learn More →



NoSQL

Learn More →



NoSQL Distilled

Learn More →

Practical Neo4j

Learn More →

Professional Services

neo4j.com/professional-services

- Neo4j Enterprise (faster, scalable, supported)
- Service & Support (24/7, other SLAs)
- Workshops, On-Site Training, Proof of Concept
- Dedicated Consulting
- Huge Partner Network
- We don't bite, ping your local rep for a relaxed chat

Talk to us!

neo4j.com/contact-us

- Have an interesting project, use-case?
  - We can help you promote it with articles, talks, webinars
- Wrote a tool, library, connector?
  - We love to make it known in our community
- Have questions about licensing & pricing?
  - Reach out with your questions, we help you clarify
- Want to work on Neo4j?
  - We offer many cool jobs, please apply

End of
More Ways to Learn about Neo4j

Questions?