Neo4j & Cypher

Content

- Neo4j
- Cypher
- Hands-on

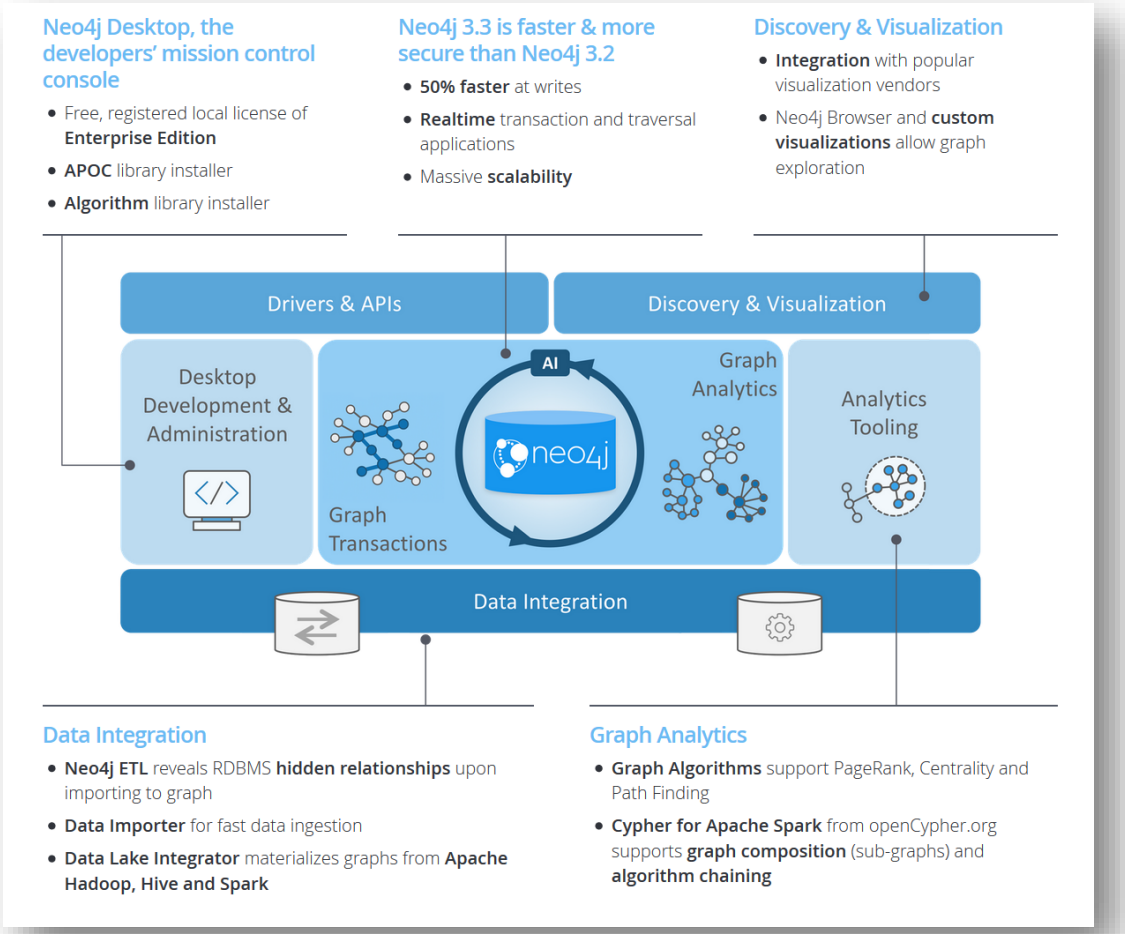
Content

- Neo4j
- Cypher
- Hands-on

Neo4j

What is Neo4j?

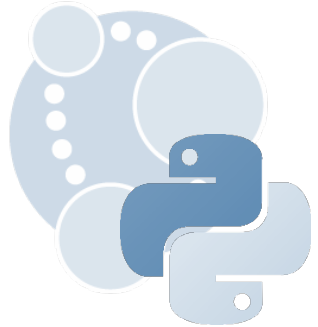
- Neo4j is a **Database** and DBMS
  use it to reliably store information and find it later
- Neo4j's data model is a **Labeled Property Graph**
- Neo4j is a **native graph-based** database
  ➞ uses native graph storage that is specifically designed to store and manage graphs
- Neo4j has **drivers** for the most popular programing languages



Neo4j

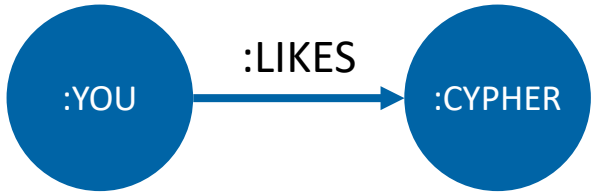
Neo4j and Python

```
# import neo4j module
import neo4j

# create connection/client/driver
driver = neo4j.GraphDatabase.driver("uri", auth=("user", "password"))

# define method to retrieve all friends of a person
def print_friends_of(tx, name):
    query = "MATCH (a:Person)-[:KNOWS]->(f) WHERE a.name={name} RETURN f.name"
    for record in tx.run(query, name=name):
        print(record["f.name"])

# get all friends of "Alice"
with driver.session() as session:
    session.read_transaction(print_friends_of, "Alice")

# close connection to the database
driver.close()
```



Content

- Neo4j
- Cypher
- Hands-on



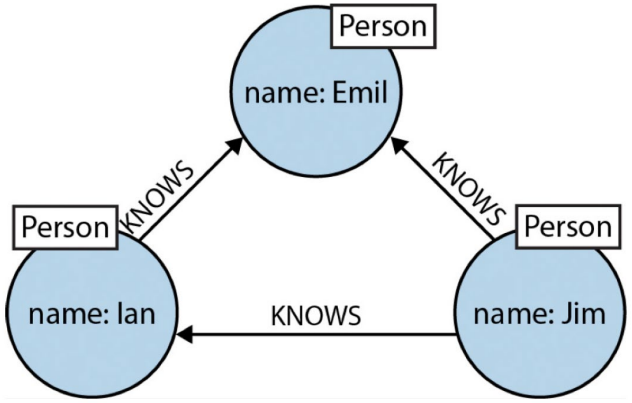

```
(:YOU)-[:LIKES]->(:CYPHER)
```

Cypher

What is Cypher?

- Cypher is Neo4j's graph **query language** (SQL for graphs!)
- Cypher is a declarative query language: it describes **what** you are interested in, **not how** it is acquired
- Cypher is meant to be very **readable** and **expressive**

Cypher is inspired by a number of different approaches and builds upon established practices for expressive querying. Many of the keywords like WHERE and ORDER BY are inspired by SQL. Pattern matching borrows expression approaches from SPARQL. Some of the collection semantics have been borrowed from languages such as Haskell and Python.

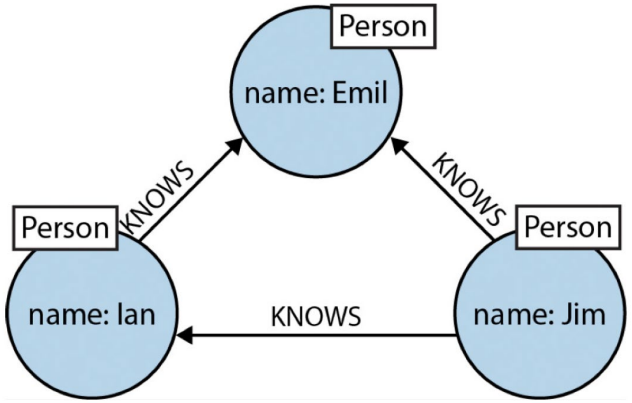



```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```

Cypher

Basic Syntax

- **Nodes** are «drawn» in parentheses:
  `()`
- **Edges** are «drawn» using pairs of dashes with greater-than or less-than signs and square brackets in between:
  `-[]->`
  `<-[]-`




```
()-[]->()
```
basically everything

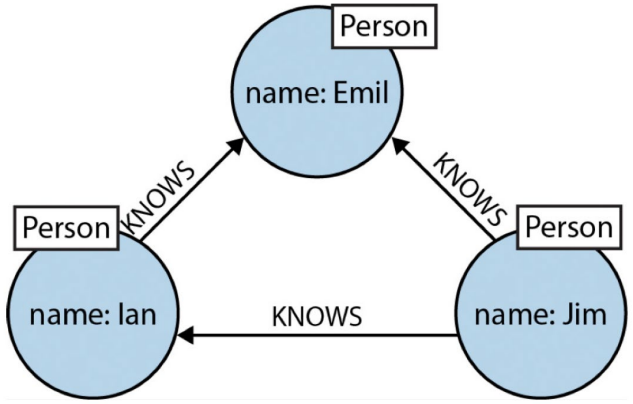
```
(:Person)-[:KNOWS]->(:Person)
```
with Labels

```
(:Person {name:'Jim'})-[:KNOWS]->(:Person {name:'Ian'})
```
with Labels and Properties

Cypher

Example (1/2)

The MATCH clause



Result
b
Emil
Ian

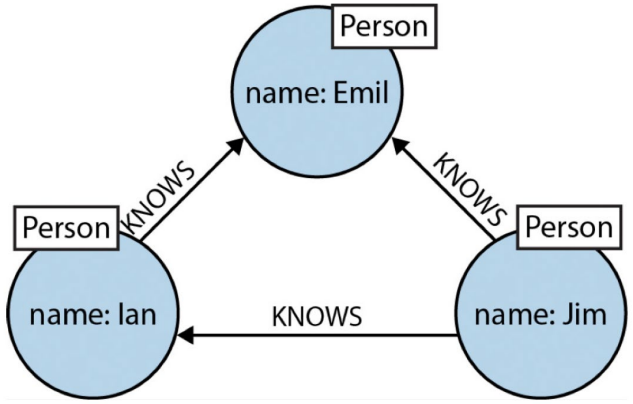
```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)
RETURN b
```

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```

Cypher

Example (2/2)

The MATCH clause



Result

b	c
Ian	Emil

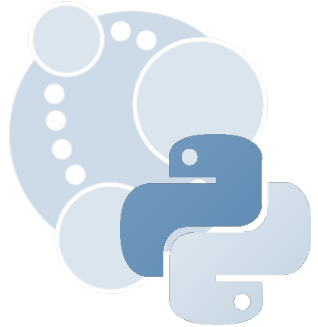
```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)
RETURN b
```

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```

Neo4j

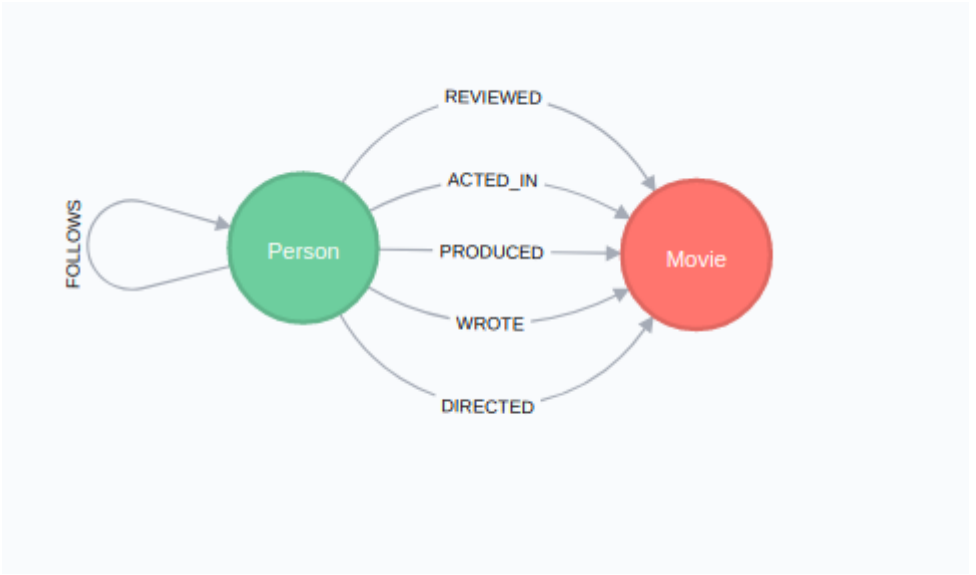
Neo4j and Python

```
# import neo4j module
import neo4j

# create connection/client/driver
driver = neo4j.GraphDatabase.driver("uri", auth=("user", "password"))

# define method to retrieve all friends of a person
def print_friends_of(tx, name):
    query = "MATCH (a:Person)-[:KNOWS]->(f) WHERE a.name={name} RETURN f.name"
    for record in tx.run(query, name=name):
        print(record["f.name"])

# get all friends of "Alice"
with driver.session() as session:
    session.read_transaction(print_friends_of, "Alice")

# close connection to the database
driver.close()
```



Content

- Neo4j
- Cypher
- Hands-on



Hands-on

Neo4j Browser