Large-Scale and Multi-Structured Databases

Neo4J Introduction

Prof Pietro Ducange

Ing. Alessio Schiavo

alessio.schiavo@phd.unipi.it

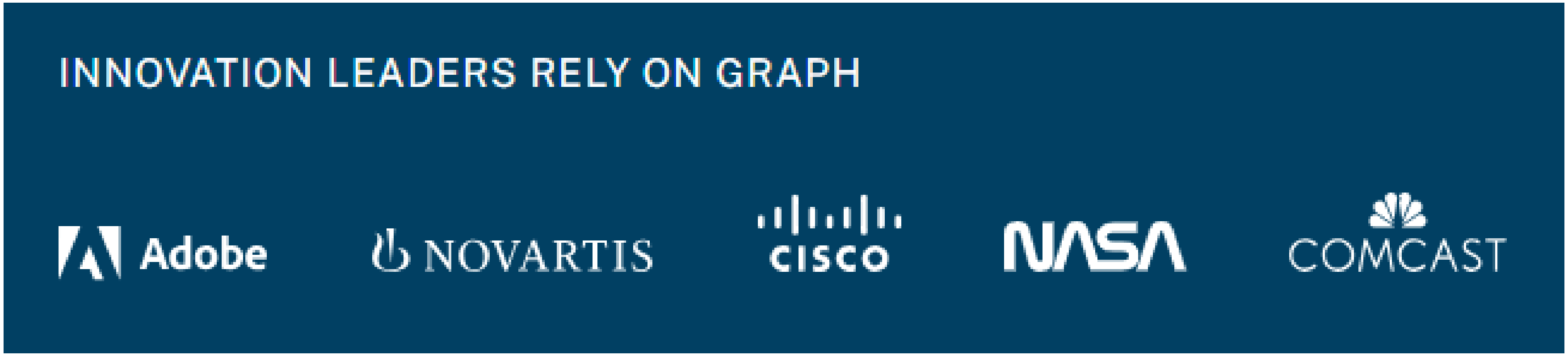
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

Copyright Issues

Most of the information included this presentation have been extracted from the official documentation of Neo4J (https://neo4j.com/docs/)



https://www.youtube.com/watch?v=urO5FyP9PoI

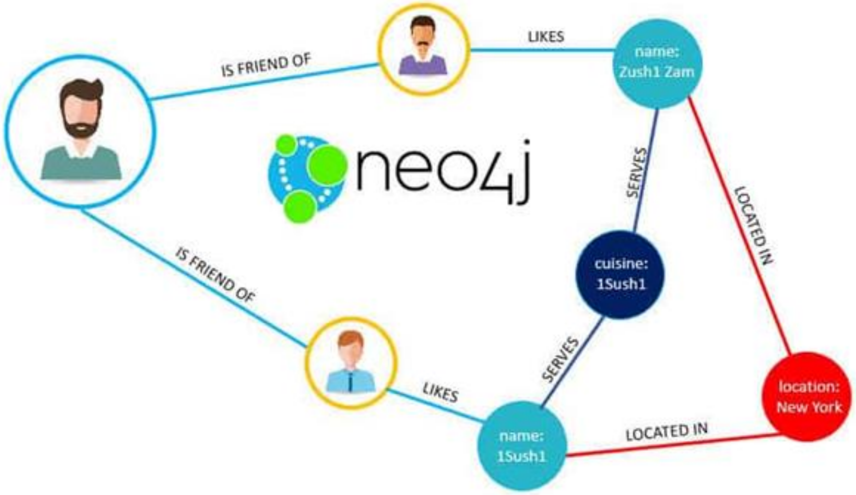
Neo4J

- **Neo4j** is a **native graph database**, built from the ground up to leverage not only data but also data *relationships.*
- Unlike traditional databases, which arrange data in rows, columns and tables, **Neo4j** has a flexible structure defined by ***stored relationships*** between data records.
- Each data record, or *node*, **stores direct pointers** to all the nodes it’s connected to.
- Neo4j's design allows to perform *queries with complex connections* orders of magnitude faster, and with more depth, than other databases.

Neo4J graph DB: main concepts

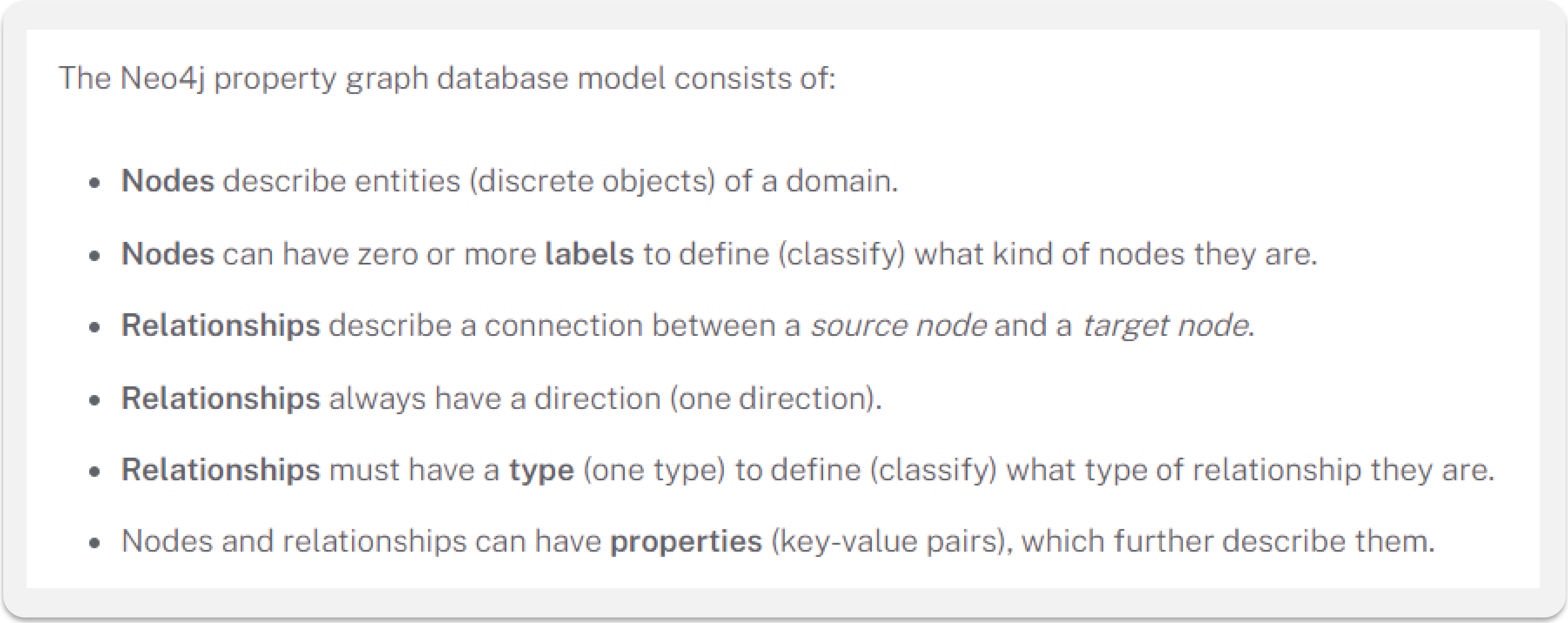
A Neo4J graph is the typical graph composed by:

- Nodes
- Labels
- Relationships
- Properties
- Indexes
- Constraints



https://neo4j.com/docs/getting-started/current/appendix/graphdb-concepts/

Neo4J graph DB: main concepts



The Neo4j property graph database model consists of:

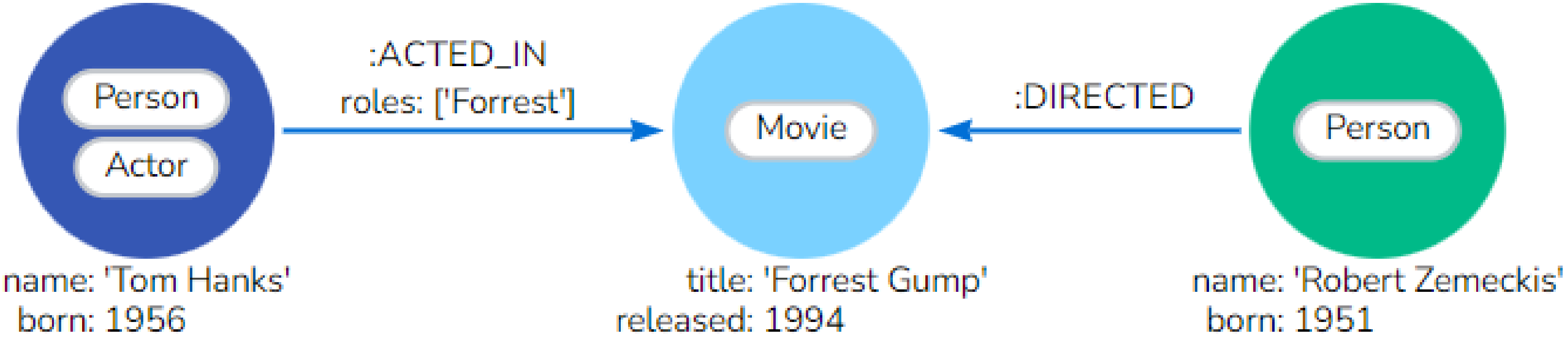
- **Nodes** describe entities (discrete objects) of a domain.
- **Nodes** can have zero or more **labels** to define (classify) what kind of nodes they are.
- **Relationships** describe a connection between a *source node* and a *target node*.
- **Relationships** always have a direction (one direction).
- **Relationships** must have a **type** (one type) to define (classify) what type of relationship they are.
- Nodes and relationships can have **properties** (key-value pairs), which further describe them.

Neo4J node

Nodes are the **entities** in the graph. A node stores data similarly to rows in DBMS and document/item in NoSQL

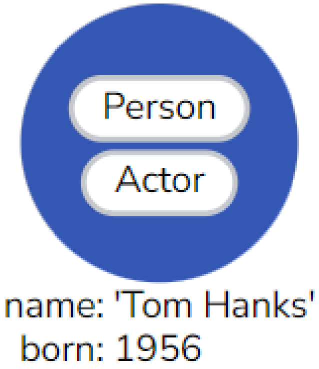
A node can:

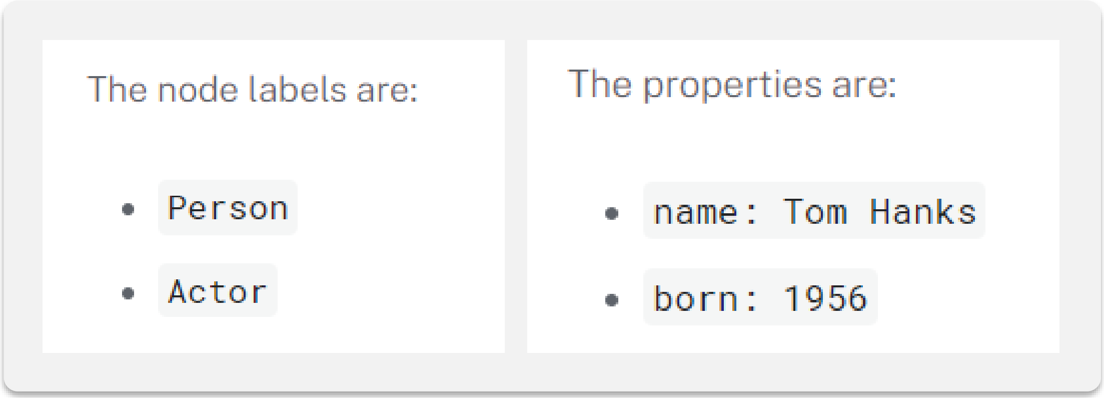
- have associated *properties* (schema-free)
- connect with other objects through a *relationship*
- be *labeled*
- be indexed



Neo4J label

- Labels are used to shape the domain by grouping nodes into sets where all nodes that have a certain label belongs to the same set.
- A node can have zero to many **labels to define** (classify) **what is the kind of that node**.
- Can be added and removed dynamically
- Conventionally expressed in CamelCase





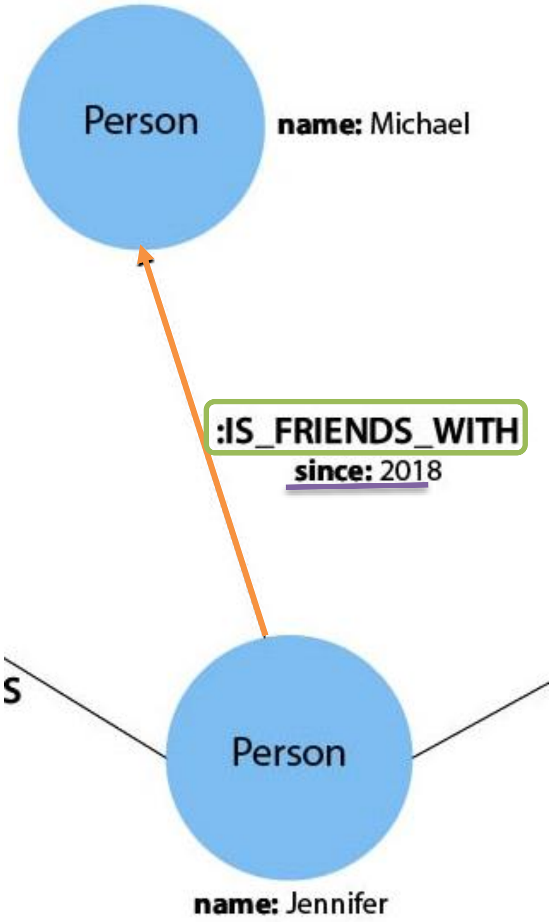
The node labels are:

- Person
- Actor

The properties are:

- name: Tom Hanks
- born: 1956

Neo4J relationship



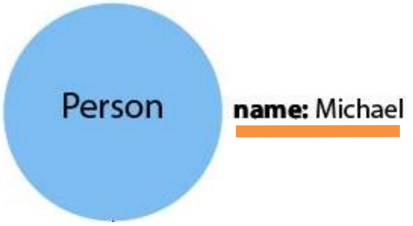
- A **relationship** connects *two* nodes.
- Relationships organize nodes into structures, allowing a graph to resemble a list, a tree, a map, or a compound entity
- A relationship must have exactly one *relationship type*. Typically expressed in **UPPER CASE**
- It can have associated *properties*
- Can be *added* and *removed* dynamically
- A node can have relationships to itself.

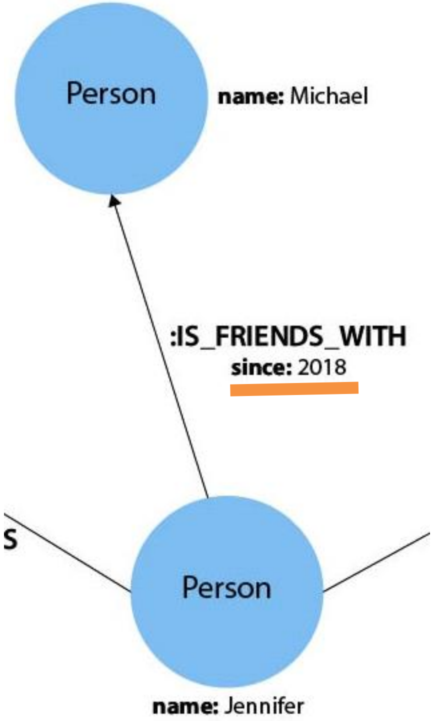
Neo4J property

Properties are name-value pairs that are used to **add qualities to nodes and relationships**.

Property types comprise:

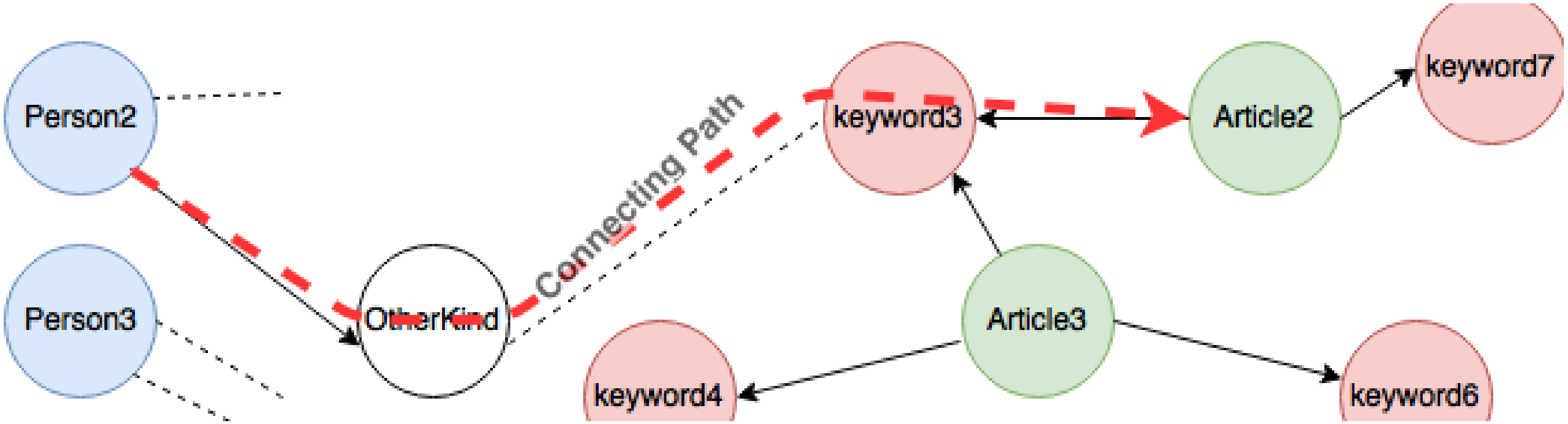
- **Number**, an abstract type, which has the subtypes Integer and Float
- **String**
- **Boolean**
- The spatial type **Point**
- Temporal types: **Date**, **Time**, **LocalTime**, **DateTime**, **LocalDateTime** and **Duration**



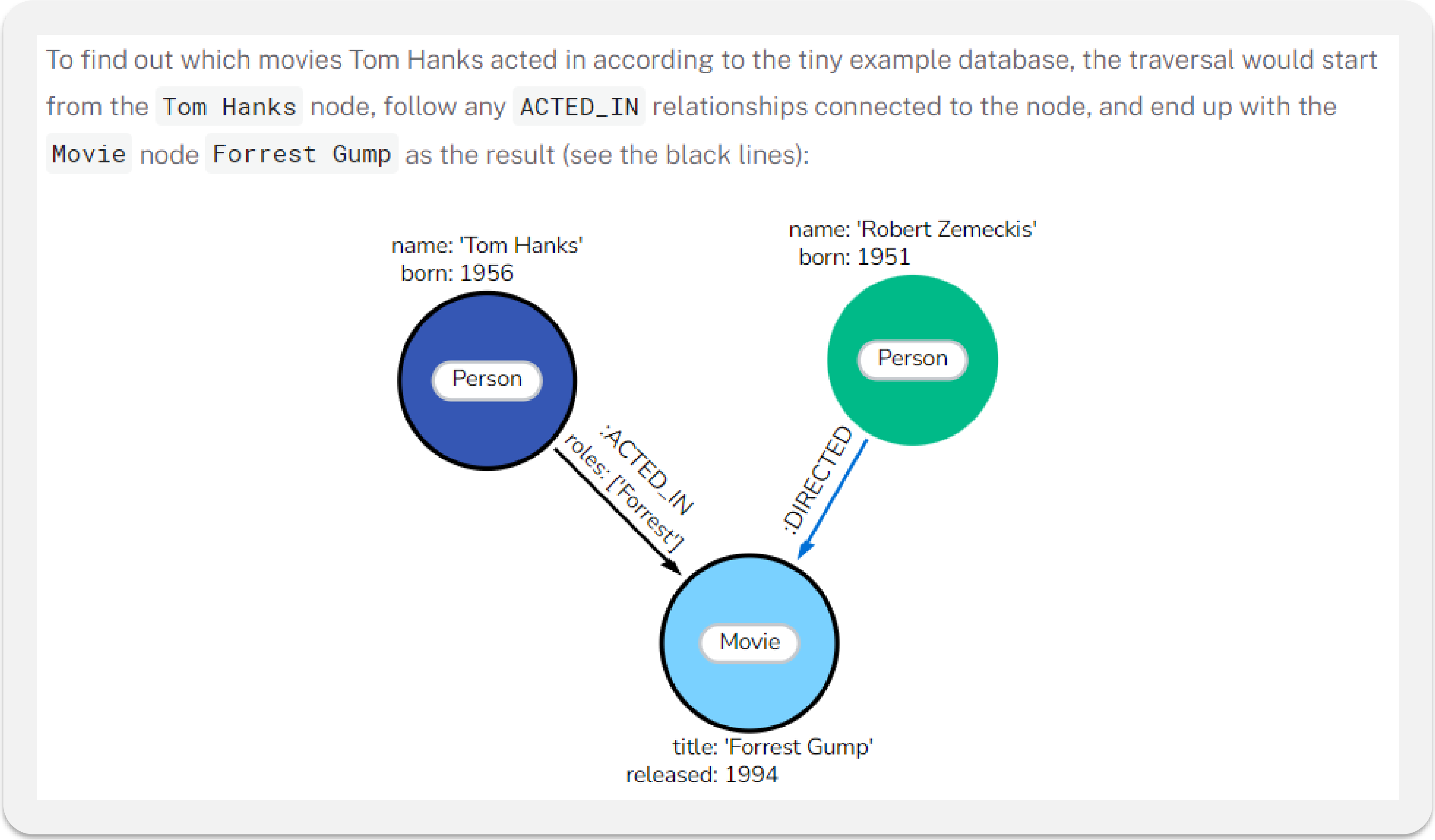


Neo4J traversal and path

- Traversing a graph means visiting nodes by following relationships according to some rules.
- The traversal result could be returned as a **path**

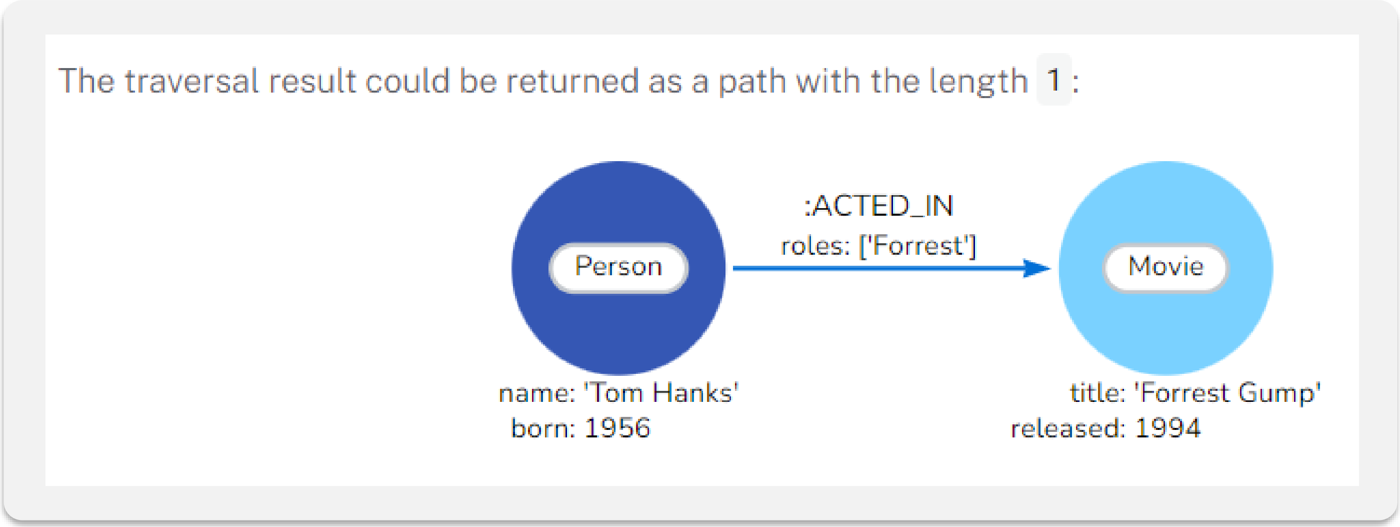


Neo4J traversal and path



To find out which movies Tom Hanks acted in according to the tiny example database, the traversal would start from the `Tom Hanks` node, follow any `ACTED_IN` relationships connected to the node, and end up with the `Movie` node `Forrest Gump` as the result (see the black lines):

Neo4J traversal and path



The traversal result could be returned as a path with the length `1`:

Neo4J indexes

The main reason for using **indexes** in a graph database is **to find the starting point of a graph traversal**. Once that starting point is found, the traversal relies on in-graph structures to achieve high performance.

An index can be:

- **Single-property**: the index refers to a single property for a given label. It can match ranges.
- **Composite**: the index refers to multiple properties for a given label. It can match only by equality.

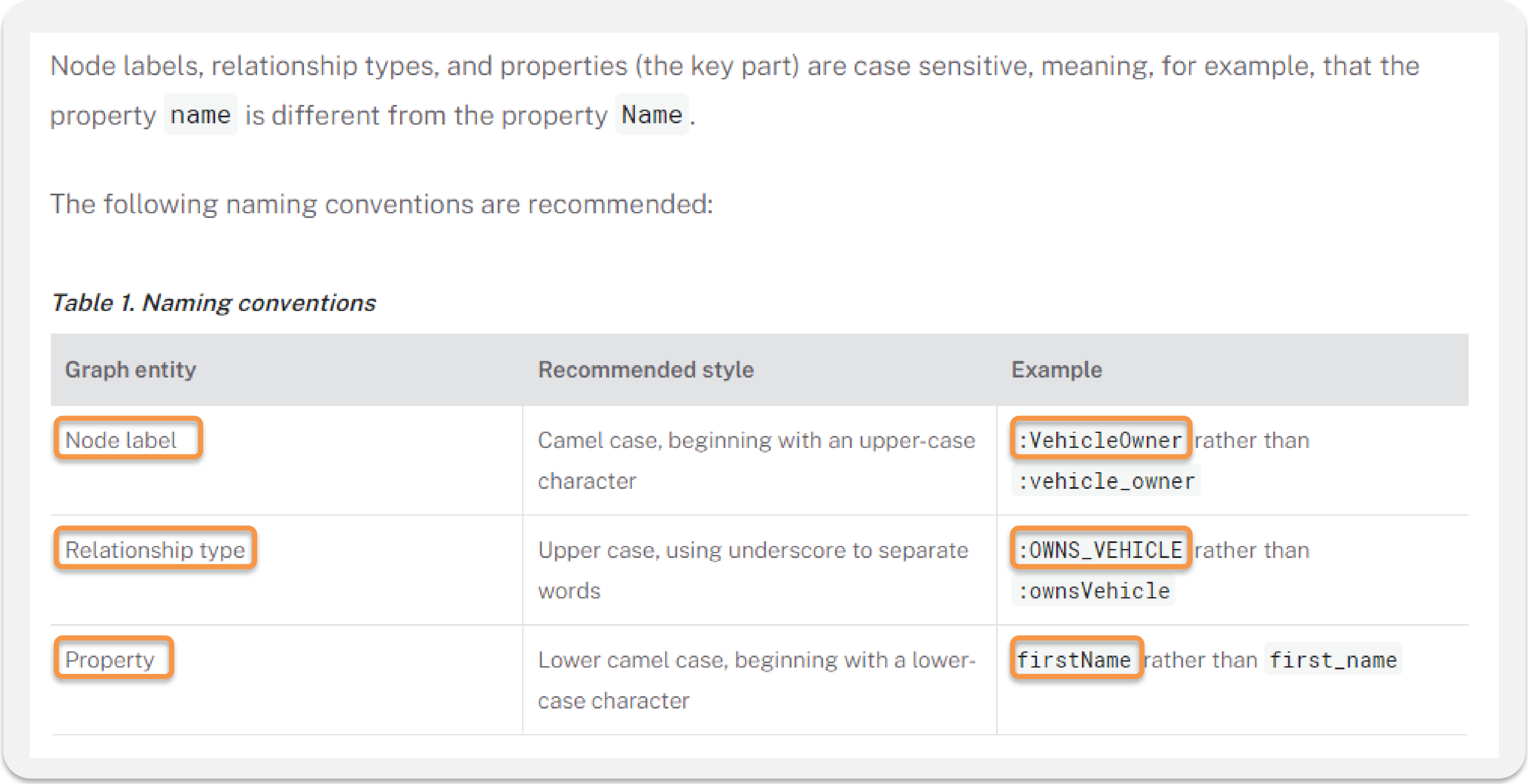
Neo4J Constraints

Constraints are used to make sure that the data adheres to the rules of the domain.

We can create constraints to:

- Enforce *uniqueness* (e.g. each Person node is unique)
- Enforce *existence* of a property in a node (e.g. each Person must have the property *name* defined)

Neo4J Naming Conventions



Node labels, relationship types, and properties (the key part) are case sensitive, meaning, for example, that the property `name` is different from the property `Name`.

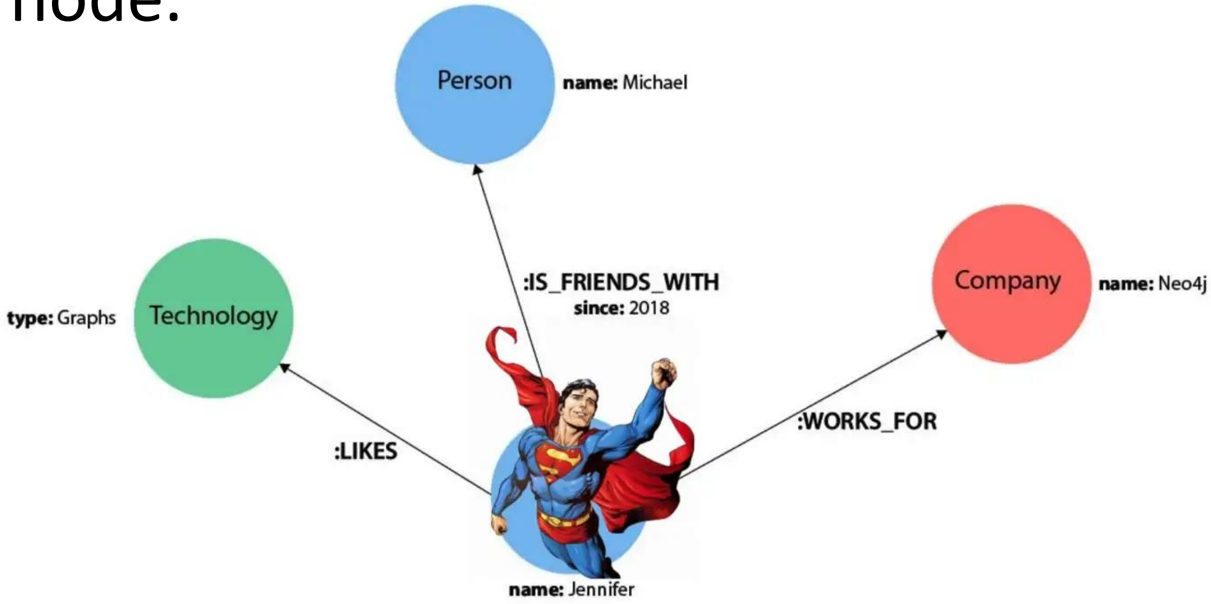
The following naming conventions are recommended:

Table 1. Naming conventions

Graph entity	Recommended style	Example
Node label	Camel case, beginning with an upper-case character	`:VehicleOwner` rather than `:vehicle_owner`
Relationship type	Upper case, using underscore to separate words	`:OWNS_VEHICLE` rather than `:ownsVehicle`
Property	Lower camel case, beginning with a lower-case character	`firstName` rather than `first_name`

Neo4J Supernodes

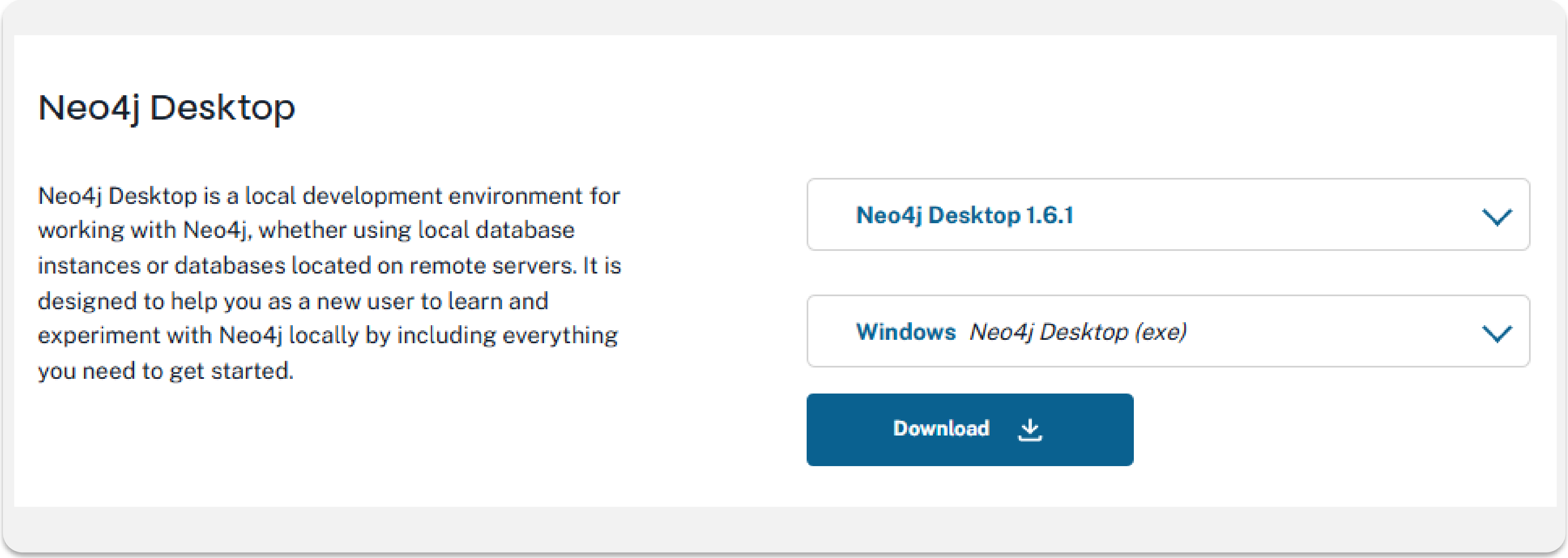
- **Supernodes** are nodes having a huge number of relationships (hundreds of thousands)
- They are problematic because they considerably slow down graph traversal when all relationships are to be traversed.
- For example, given a social media graph, a celebrity node is a super node.



Install Neo4J

- Download Neo4J
  from https://neo4j.com/deployment-center/
  You have 3 options:
  1. **Enterprise Edition**: paid license with 30 days free trial
  2. **Community Edition**: open-source and free. Less features than Enterprise Edition
  3. **Neo4J Desktop**: graphical installation available for Windows, Linux and MAC OSX. It is shipped with an enterprise edition key for developers.

Install Neo4J Desktop



Neo4j Desktop

Neo4j Desktop is a local development environment for working with Neo4j, whether using local database instances or databases located on remote servers. It is designed to help you as a new user to learn and experiment with Neo4j locally by including everything you need to get started.

https://neo4j.com/deployment-center/

Suggested Readings

Students are invited to read the official documentation of Neo4J

https://neo4j.com/docs/

https://neo4j.com/docs/getting-started/whats-neo4j/

https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/