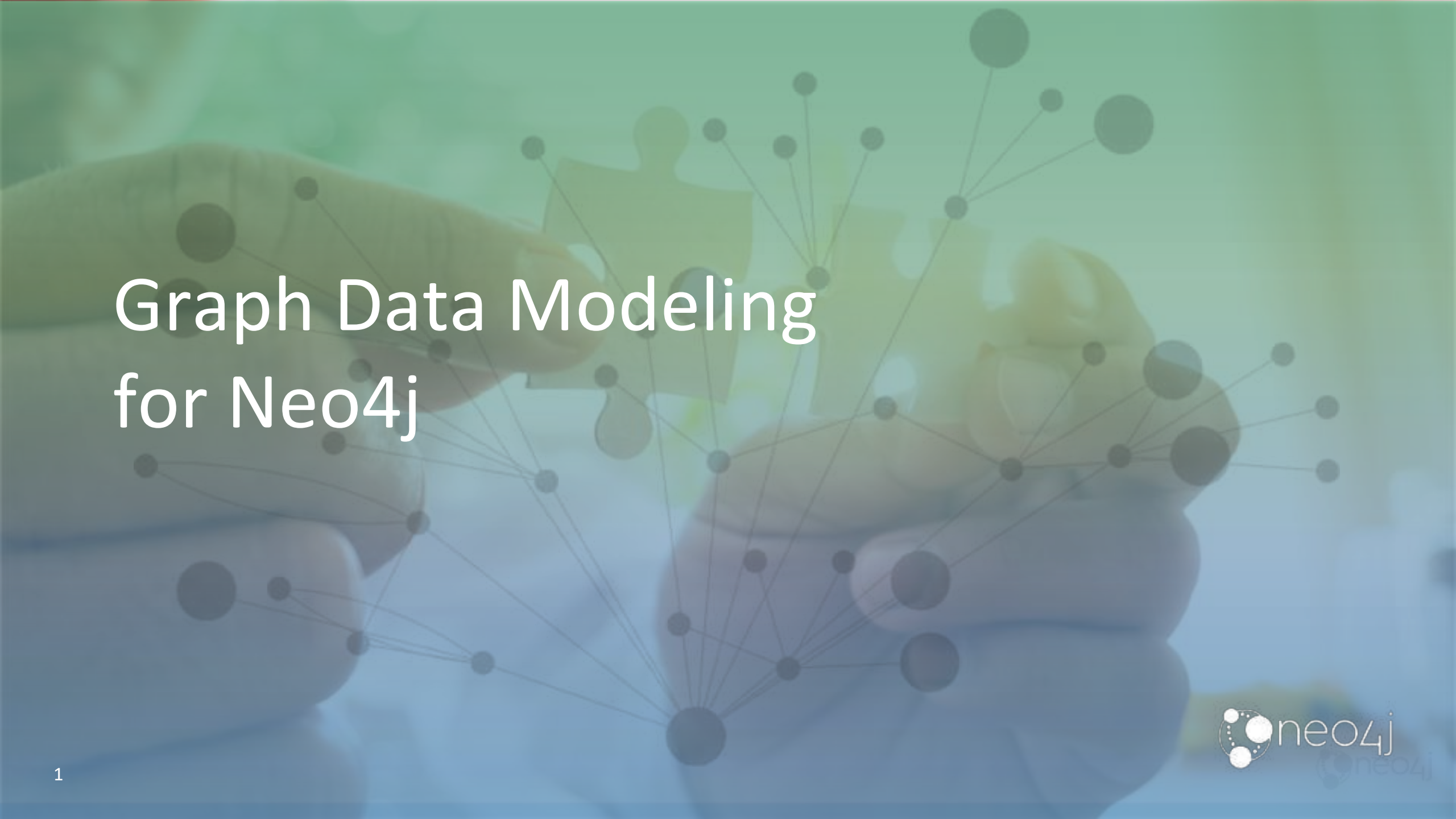
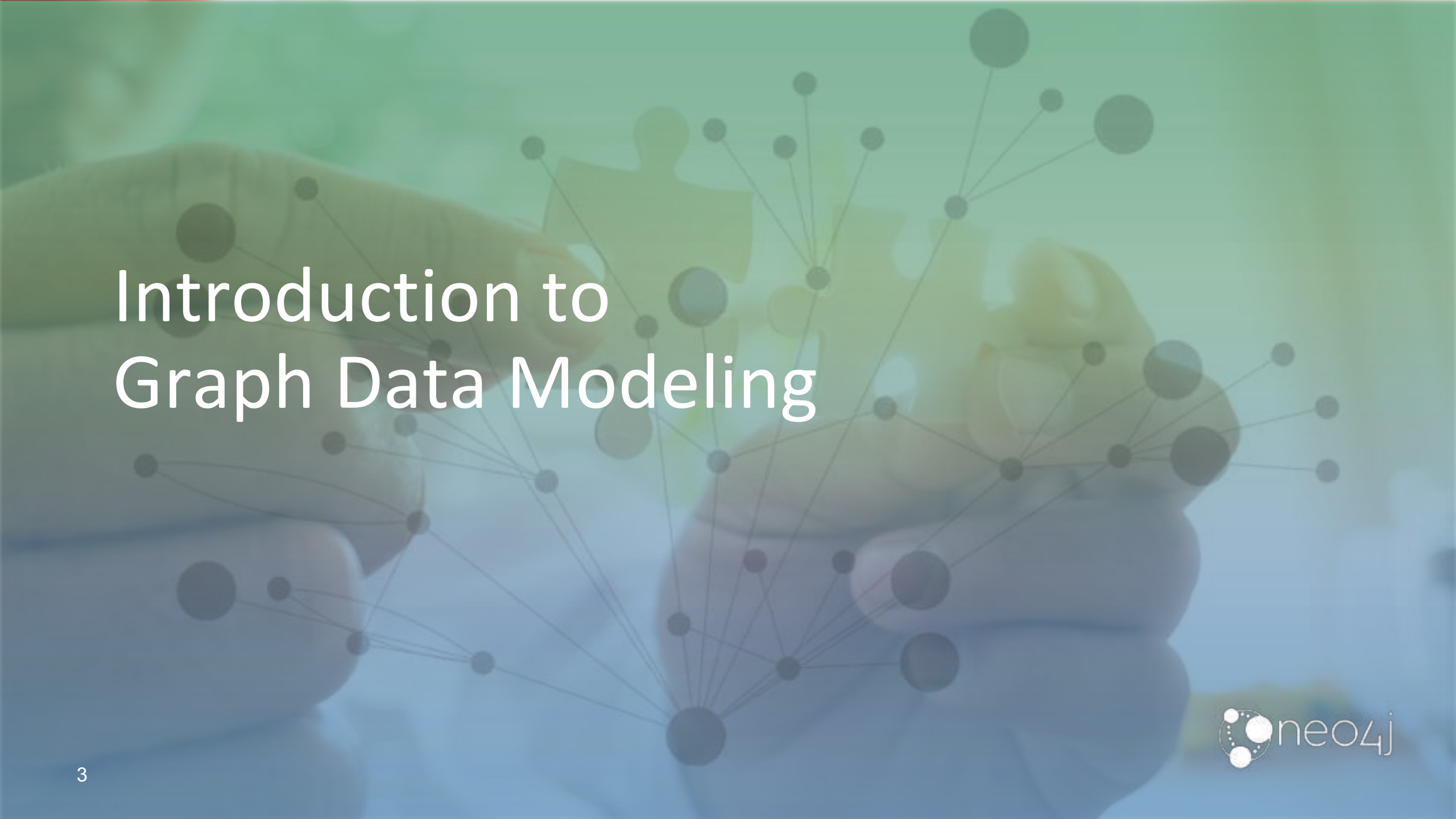
Graph Data Modeling for Neo4j



Lesson Overview

1. Introduction to Graph Data Modeling
2. Designing the Initial Graph Data Model
3. Graph Data Modeling Core Principles
4. Common Graph Structures
5. Refactoring and Evolving a Graph Data Model


Introduction to
Graph Data Modeling

Introduction to Graph Data Modeling

- Graph data modeling
- Neo4j graph data modeling
- Tools
- Workflow

Graph data modeling

What is Graph Data Modeling?

Graph data modeling is a collaborative effort by stakeholders including developers

Stakeholders include:

- Business analysts
- Architects
- Managers
- Project leaders

The application domain is analyzed by stakeholders and developers

- They develop a data model for use with Neo4j
- Stakeholders must …
  - Understand the domain
  - Be prepared to ask detailed questions on business operations

Introduction to Graph Data Modeling

- Graph data modeling
- Neo4j graph data modeling
- Tools
- Workflow

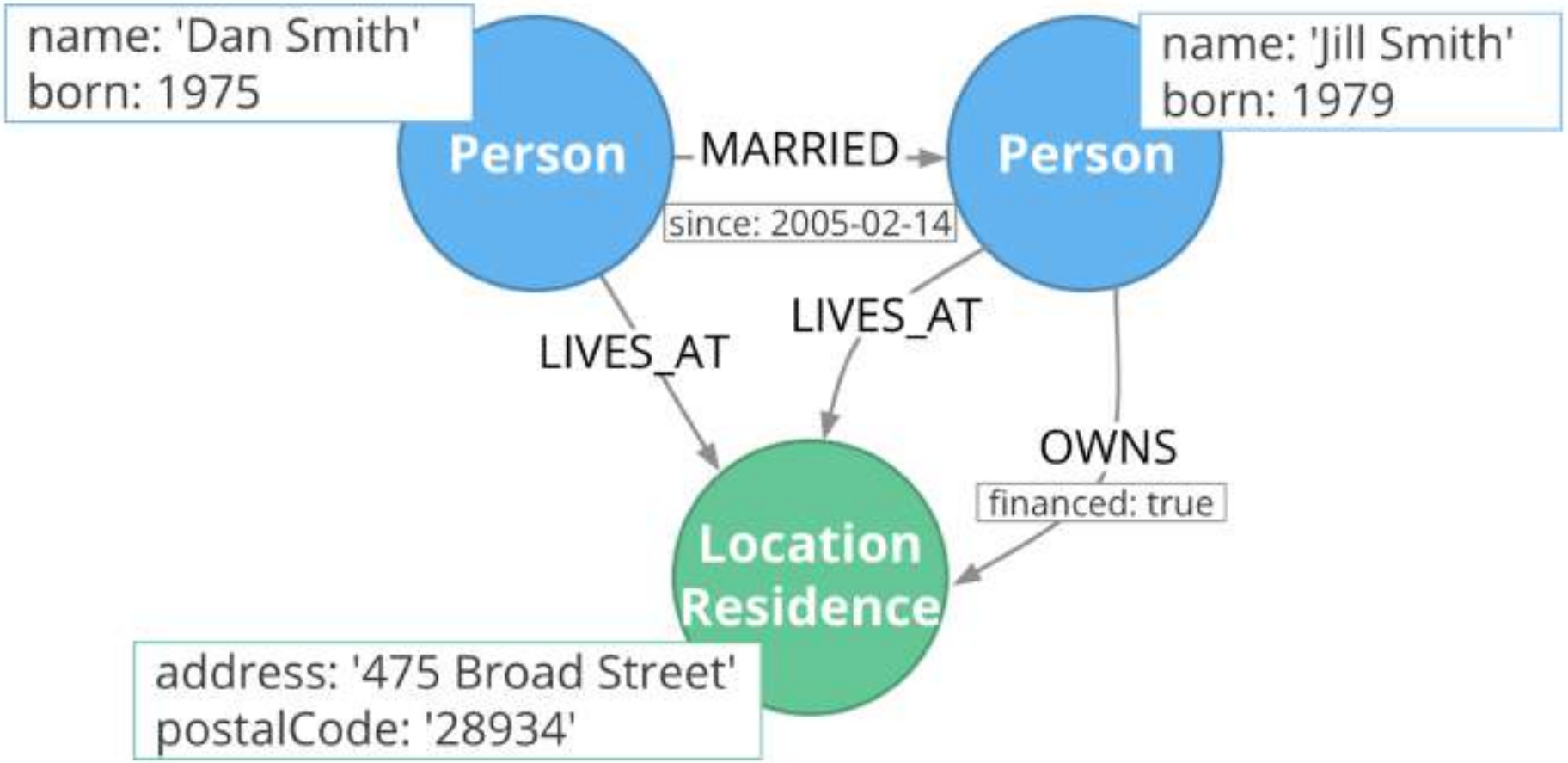
Neo4j graph data modeling

Neo4j Supports Graph Data Modeling

- Neo4j is a full-featured graph database
  - It includes tools used to create **property graphs**
- It supports application access in retrieving data for business use cases by **traversing the graph**

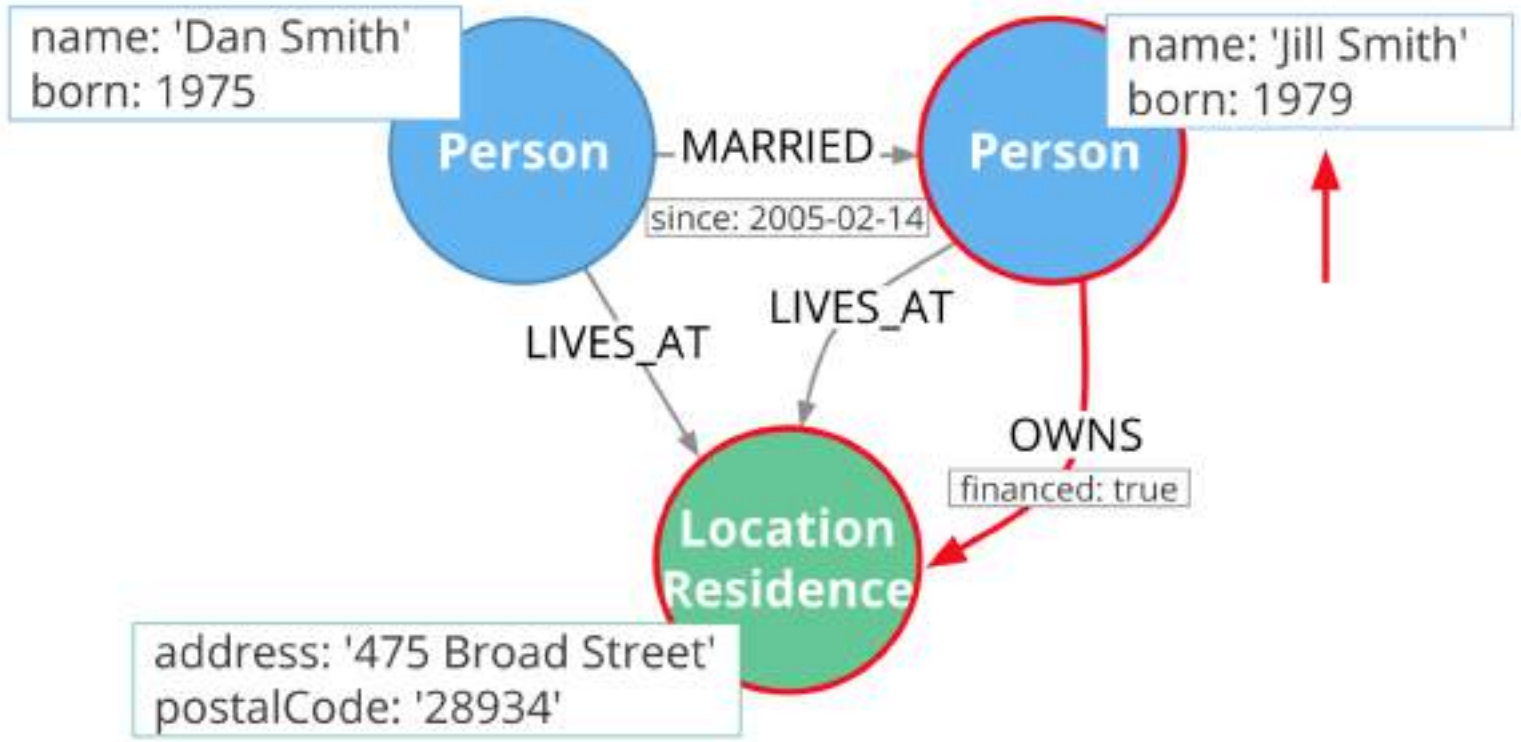
Neo4j Property Graph Model

- Nodes (Entities)
- Relationships
- Properties
- Labels



Graph Traversal

```
MATCH (r:Residence)<-[:OWNS]-(p:Person)
WHERE r.address = '475 Broad Street'
RETURN p
```

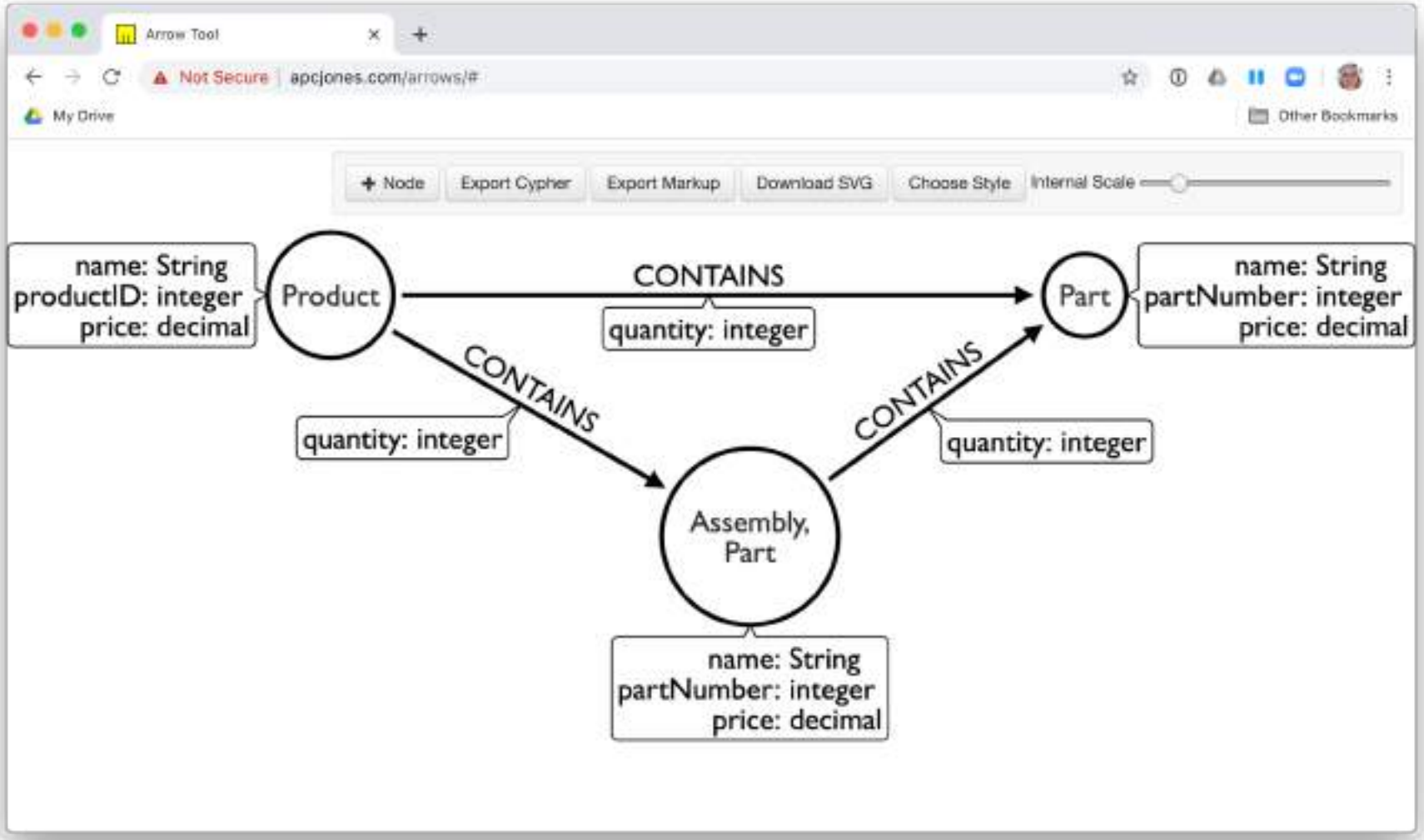


Introduction to Graph Data Modeling

- Graph data modeling
- Neo4j graph data modeling
- Tools
- Workflow

Tools

Graph Data Modeling Tools (arrows.app)



Guided Exercise: Using the Arrow tool

https://youtu.be/NB184T-S46w

Introduction to Graph Data Modeling

- Graph data modeling
- Neo4j graph data modeling
- Tools
- Workflow

Workflow

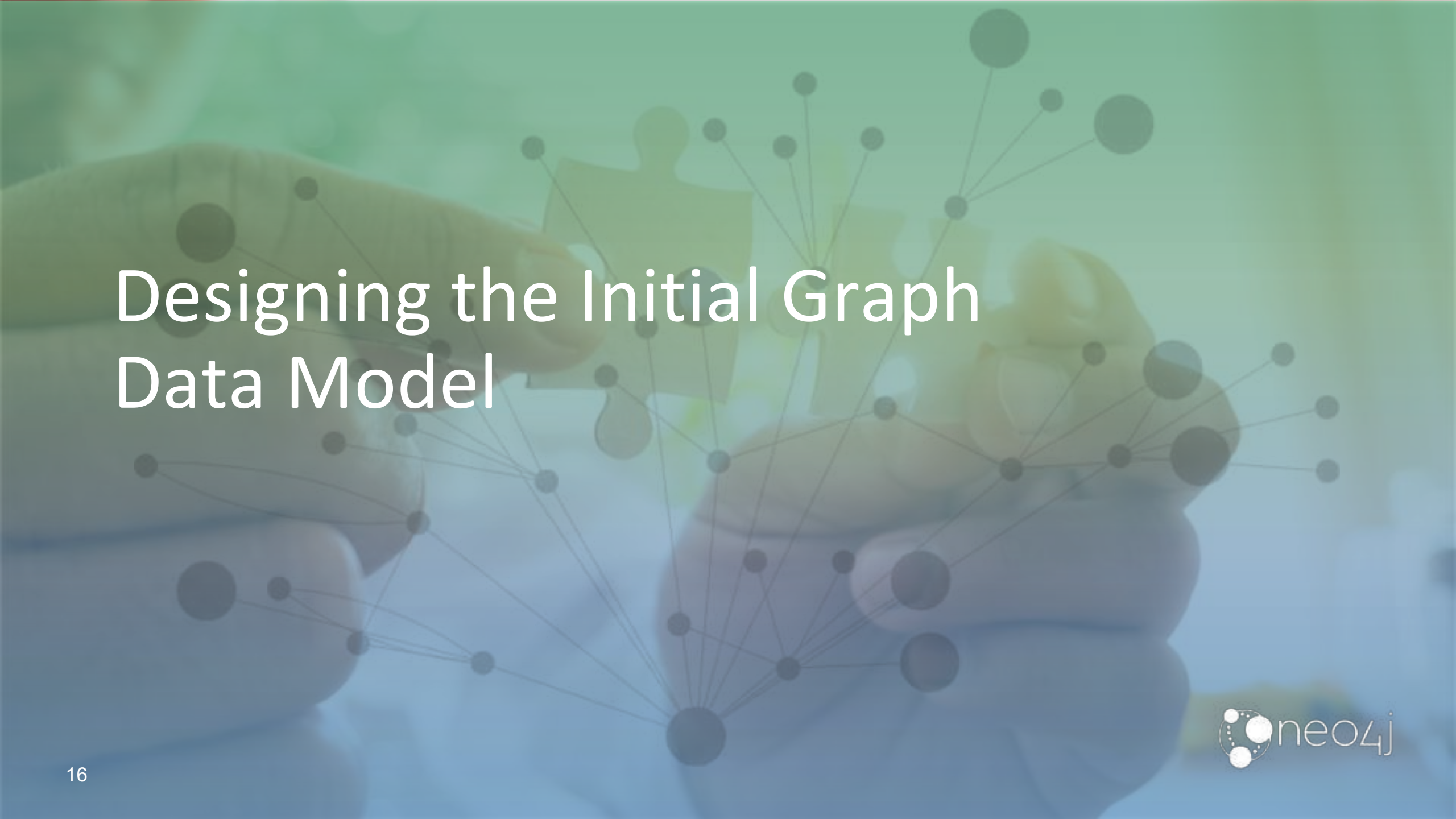
Workflow for Graph Data Modeling

Step	Description	Stakeholders	Developers
1.	Build the initial graph data model.	✔	✔
2.	Create and profile Cypher queries to support the model.		✔
3.	Create data in the database to support the model.		✔
4.	Identify additional questions for the application.	✔	✔
5.	Modify the graph data model to support new questions.		✔
6.	Refactor the database to support the revised graph data model.		✔
7.	Create and profile Cypher queries to support the revised model.		✔
	Repeat Steps 4-7.	✔	✔

Summary

You should now be able to:

- **Describe the purpose graph data modeling**
- **Describe how Neo4j supports graph data modeling**
- **Describe the tools available for graph data modeling**
- **Describe the workflow for graph data modeling**



Designing the Initial Graph Data Model

In This Module You'll Learn ...

How to ...

- Describe the domain for a model
- Define the questions for the domain
- Identify entities from the questions for the domain
- Model the domain using the Arrow Tool to
- Identify the connections between entities
- Describe how to test the initial model

Designing the Initial Data Model

1. Understand the domain
2. Create high-level sample data
3. Define specific questions for the application
4. Identify entities
5. Identify connections between entities
6. Test the questions against the model
7. Test scalability

Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

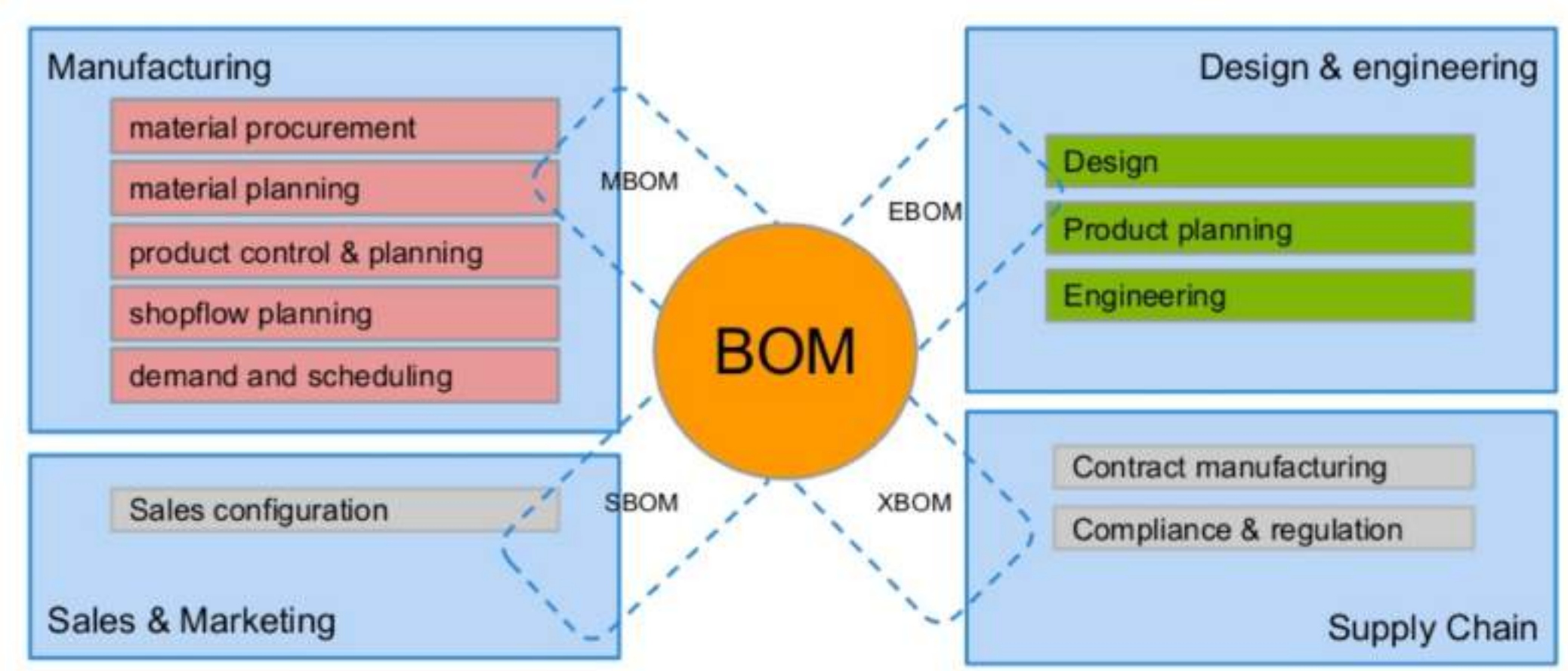
1. Understand the domain

Step 1: Understanding the Domain

- Describe the application
- Identify the stakeholders
- Identify the developers that will develop the application
- Identify the users of the application (both people and systems)
- Enumerate the use cases
- Stakeholders agreed upon all use cases

Note: Stakeholders do not need any knowledge of the underlying implementation

Example Domain: Bill of Materials



BOM connects all engineering and non-engineering processes

Example BOM Use Cases

System produces ...

1. A list of parts to make a product
2. A list of products that can be made with available parts
3. A list of parts that are made with other parts
4. **User picks** parts to make a product

System creates ...

1. A price for a product based upon the part prices
2. A list of parts that need to be ordered

Notes:

- A product or part can be made of multiple parts of the same type
- Some parts are made from other parts (sub-assembly)

Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

2. Create sample data

BOM High-level Sample Data

Products	Parts	Assemblies	Notes
Wood table 40"	Wood top 40"	Leg assembly	Has 4 legs
Deluxe wood table 40"	Glass top 40"	Leg assembly	Has 4 legs
Wood table 60"	Wood top 60"	Leg assembly	Has 6 legs, table brace
Deluxe wood table 60"	Glass top 60"	Leg assembly	Has 6 legs, table brace
	Leg		
	Leg foot		
	M20 bolt		
	M20 nut		
	Leg plate		Uses 2 bolts/nuts per leg
	Table brace		

Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

3. Define specific questions

Sample Questions for the BOM

1. What parts are needed to make Wood table 40"?
2. Do we have enough parts to make 100 Deluxe wood table 60"?
3. What products require a table brace?
4. How much will the parts cost to make product Wood table 60"?

Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

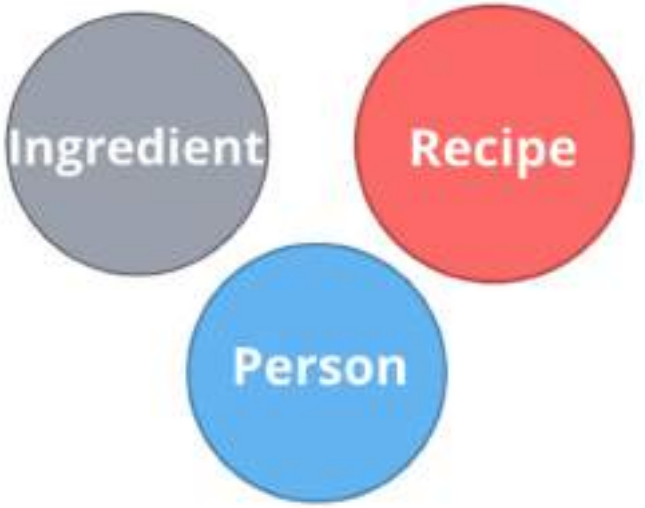
4. Identify nodes

Identify Entities from Questions

Entities are the **nouns** in the application questions:

1. What **ingredients** are used in a **recipe**?

1. Who is married to this **person**?



- The generic nouns often become labels in the model
- Use domain knowledge deciding how to further group or differentiate entities

Note: Enterprise Edition no limit to the number entities (nodes)
Community Edition limit of 34B entities

Define Properties

Two purposes for properties:

1. Unique identification
2. Answering application questions

Otherwise properties are decoration

- These properties and the associated data should not be added

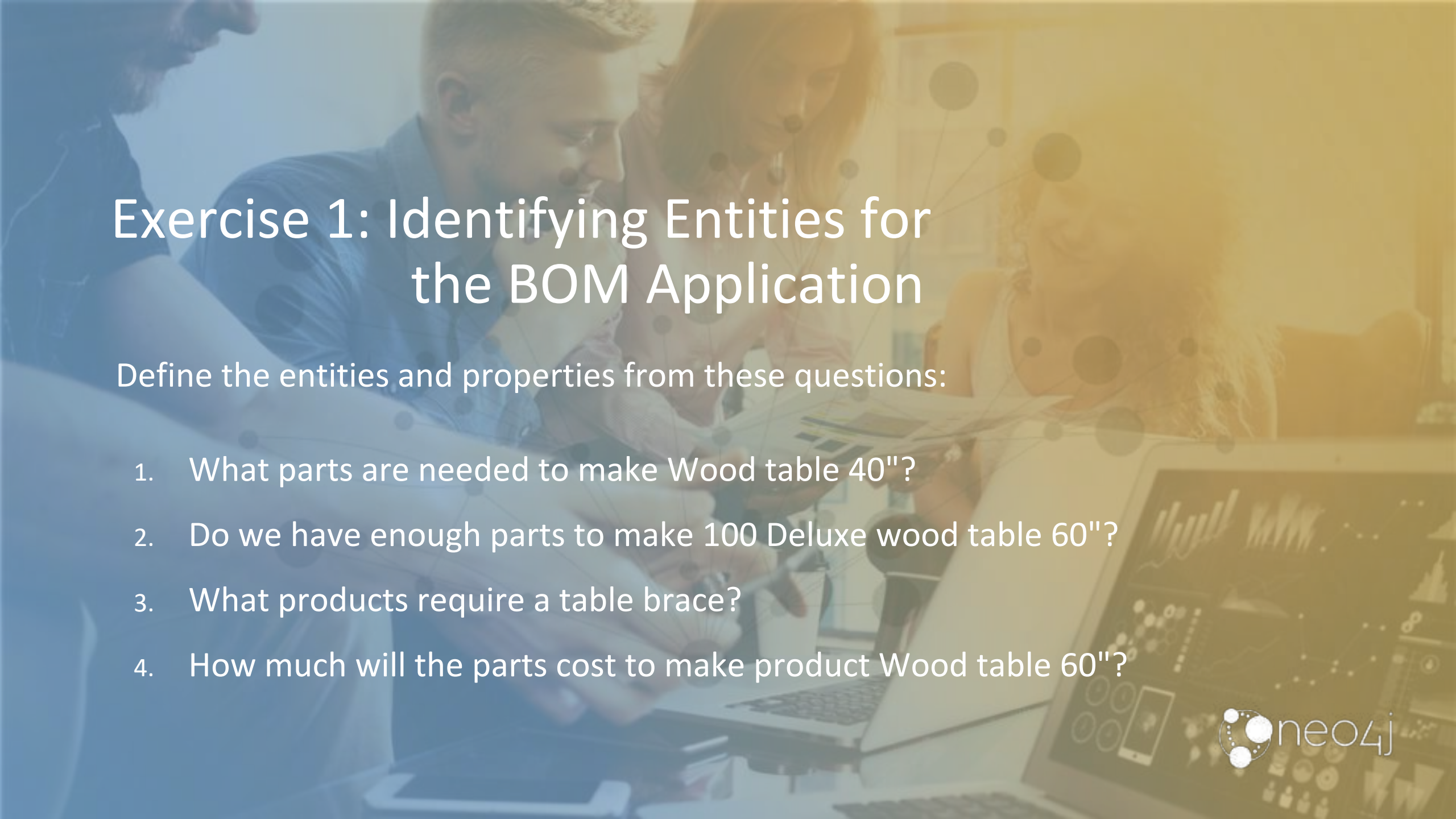
Properties are used for:

- Anchoring (where to begin the query)
- Traversing the graph (navigation)
- Returning data from the query

Exercise 1: Identifying Entities for the BOM Application

Define the entities and properties from these questions:

1. What parts are needed to make Wood table 40"?
2. Do we have enough parts to make 100 Deluxe wood table 60"?
3. What products require a table brace?
4. How much will the parts cost to make product Wood table 60"?



Exercise 1: Application Questions

- What parts are needed to make Wood table 40"?
- Do we have enough parts to make 100 Deluxe wood table 60"?
- What products require a table brace?
- How much will the parts cost to make product Wood table 60"?

Exercise 1 Solution

Part

- name
- partNumber
- price

Part, Assembly

- name
- partNumber
- price

Product

- name
- productId

Exercise 2: Creating the BOM Entity Model in the Arrow Tool

Use the entities identified earlier for the BOM application and create them in the Arrow tool

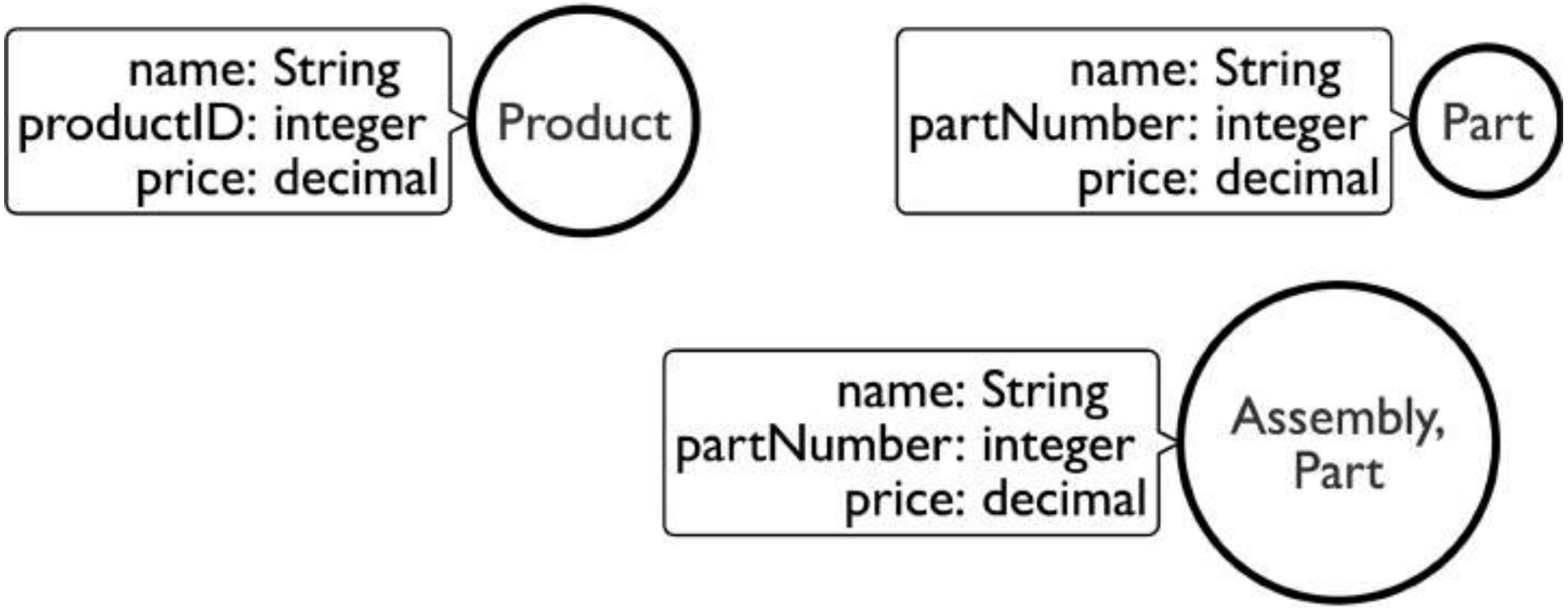
Make sure to include properties for the nodes and specify the types for the properties, rather than values

http://www.apcjones.com/arrows/

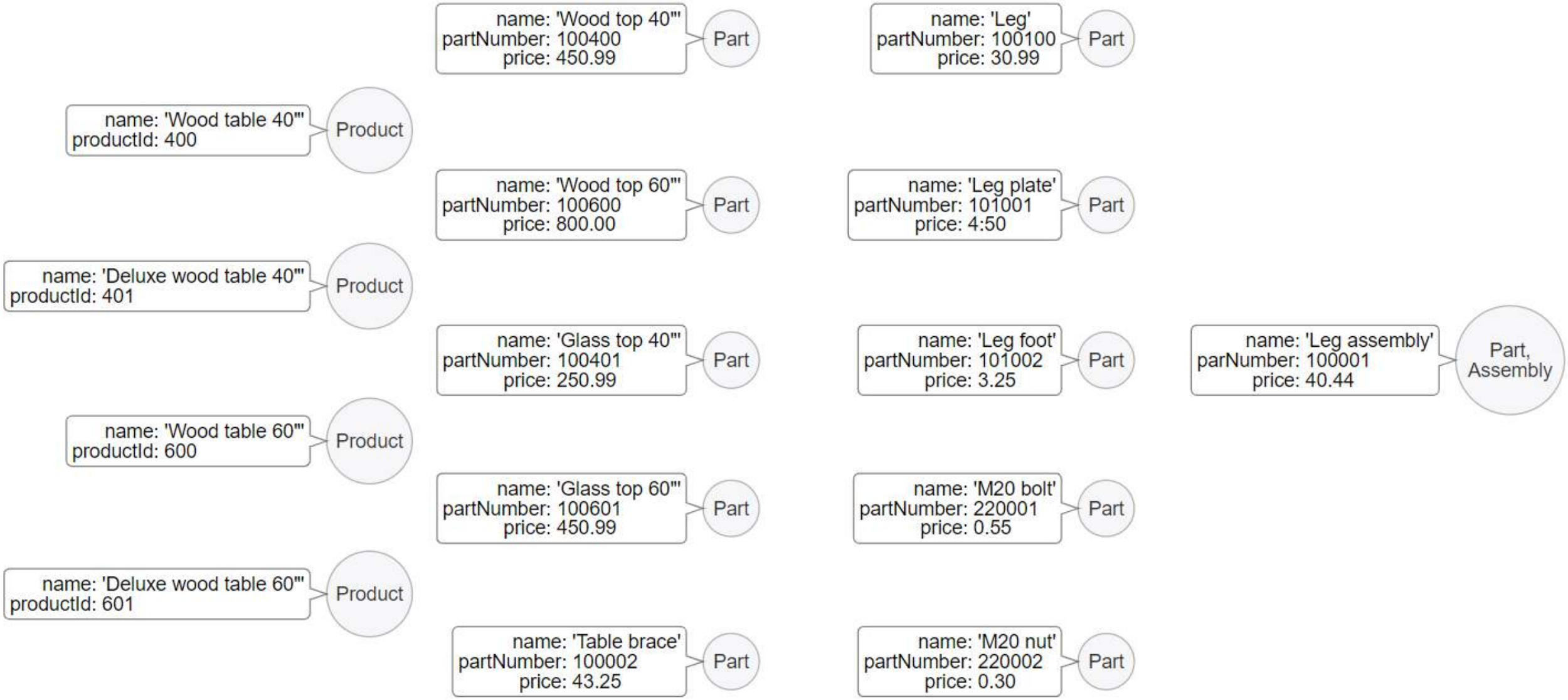
neo4j

Exercise 2 Solution

In Arrow tool your solution look like similar to this:



Exercise 2 Sample Data



Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

5. Identify connections

Identify Connections Between Entities

Connections are the **verbs** in the application questions:

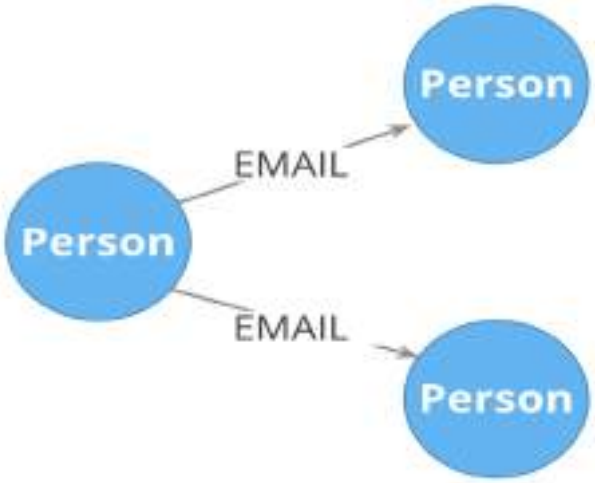
- What ingredients are **used** in a recipe?

- Who is **married** to this person?

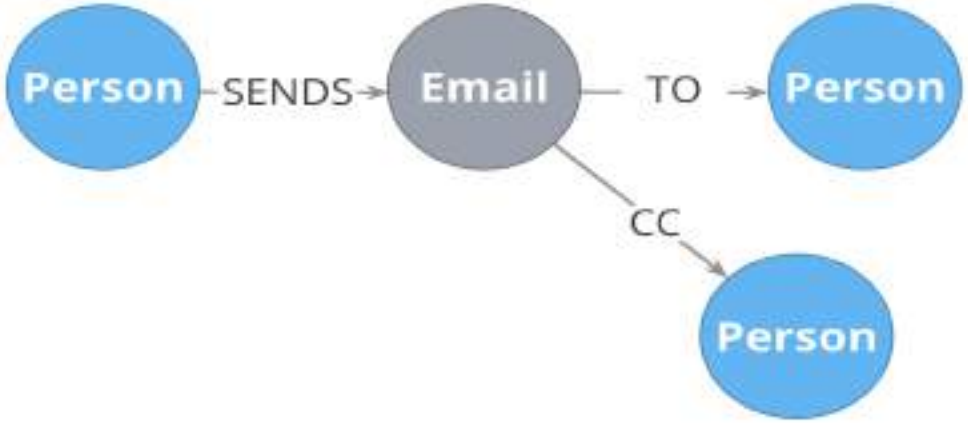
Naming Relationships

- Stakeholders must agree upon name (type) for the relationship
- Avoid names that could be construed as nouns (for example email)

Do not do this:



Instead do this:



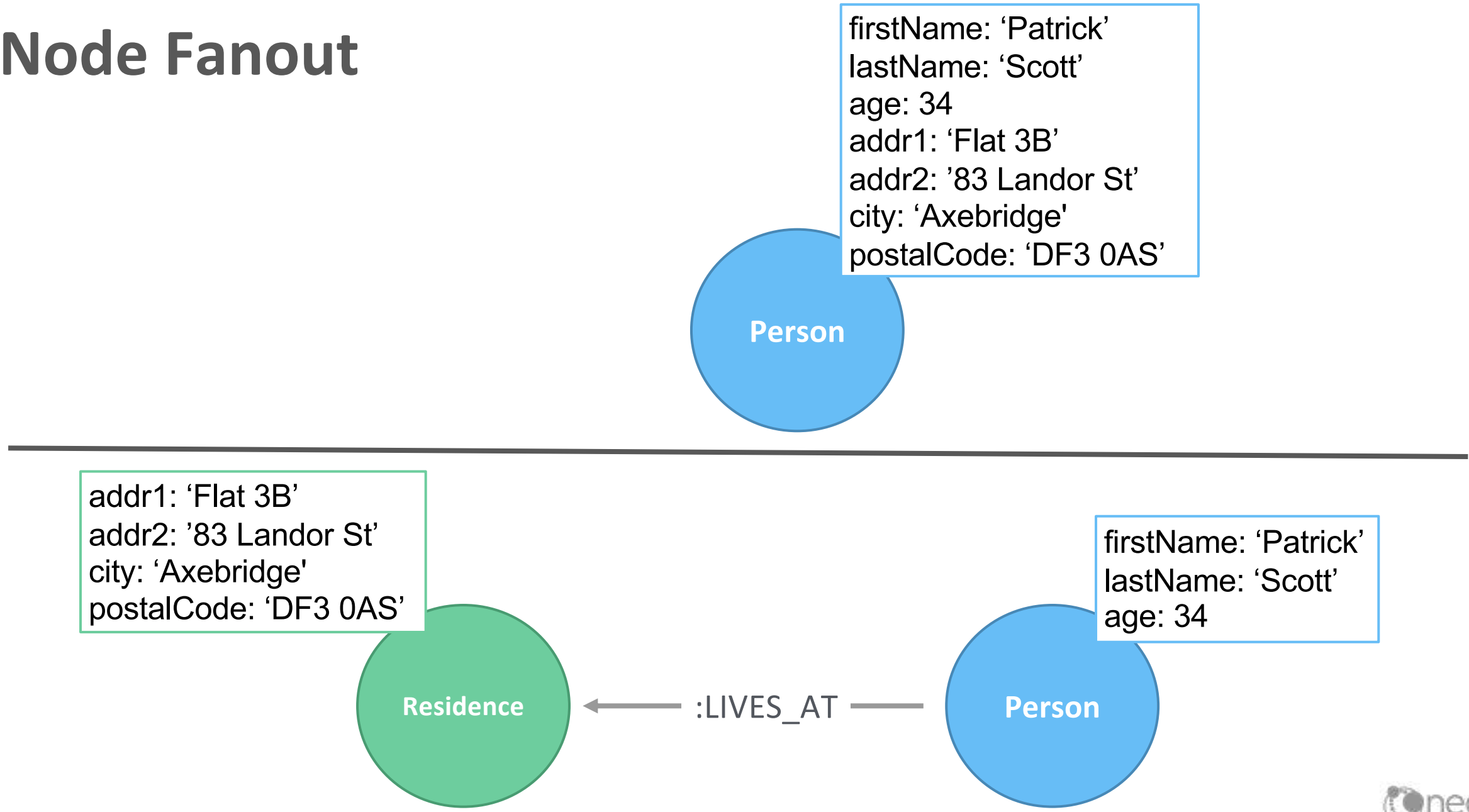
Direction and Type

Direction and **type** are required for **relationships**

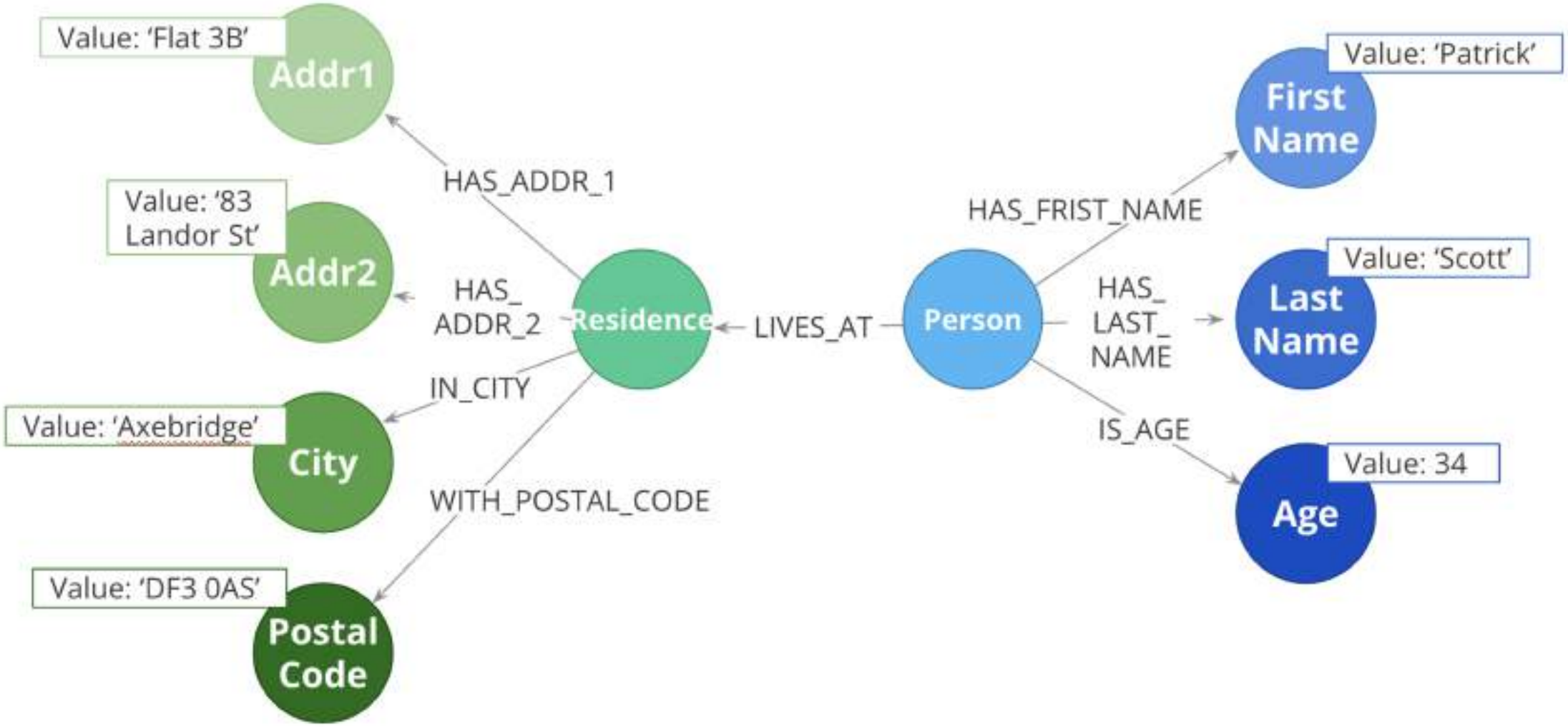
Select direction and type based on expected questions:

1. What episode follows ‘The Ark in Space’? (**NEXT**⟶)
2. What episode came before ‘Genesis of the Daleks’? (⟵**PREVIOUS**)

Node Fanout


firstName: ‘Patrick’
lastName: ‘Scott’
age: 34
addr1: ‘Flat 3B’
addr2: ’83 Landor St’
city: ‘Axebridge'
postalCode: ‘DF3 0AS’
Person
addr1: ‘Flat 3B’
addr2: ’83 Landor St’
city: ‘Axebridge'
postalCode: ‘DF3 0AS’
Residence
:LIVES_AT
firstName: ‘Patrick’
lastName: ‘Scott’
age: 34
Person

How Much Node Fanout?



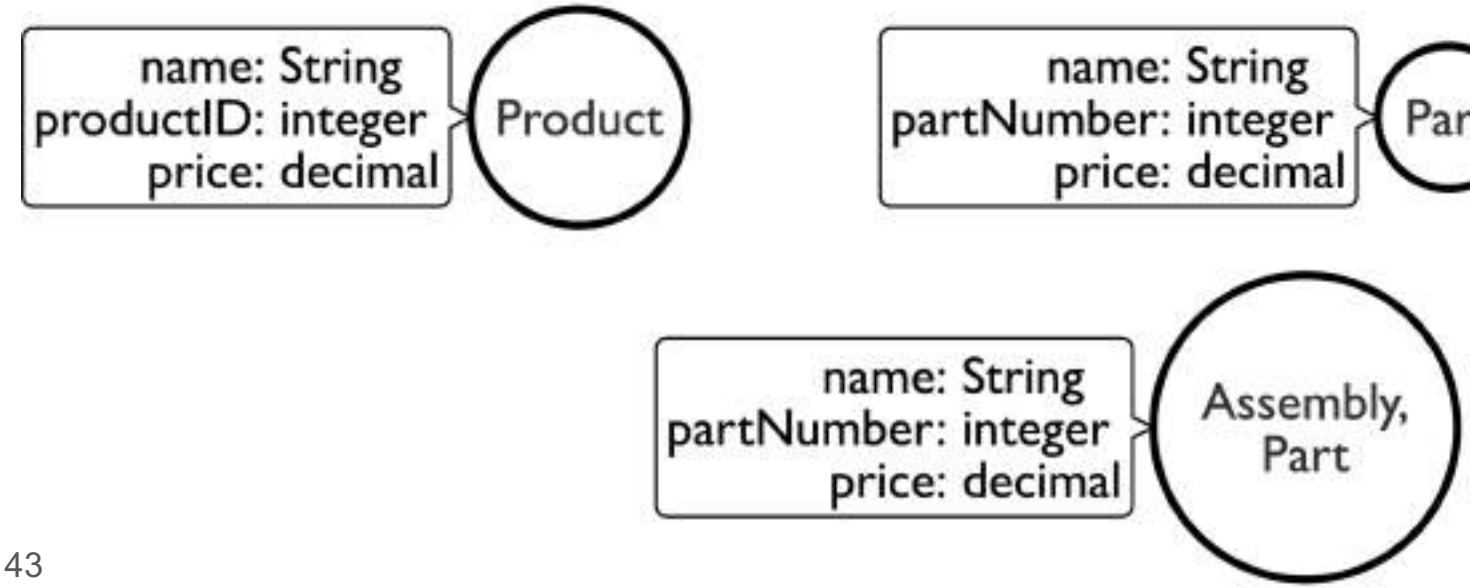
Exercise 3: Adding Relationships to the model

Instructions on the next slide ...

Exercise 3 Instructions

Questions to answer for the BOM application:

1. What parts are needed to make Wood table 40"?
2. Do we have enough parts to make 100 Deluxe wood table 60"?
3. What products require a table brace?
4. How much will the parts cost to make product Wood table 60"?



The entity model

Use the Arrow tool to add relationships

Exercise 3: Connections

Product -CONTAINS-> Part

- quantity

Product -CONTAINS-> Assembly

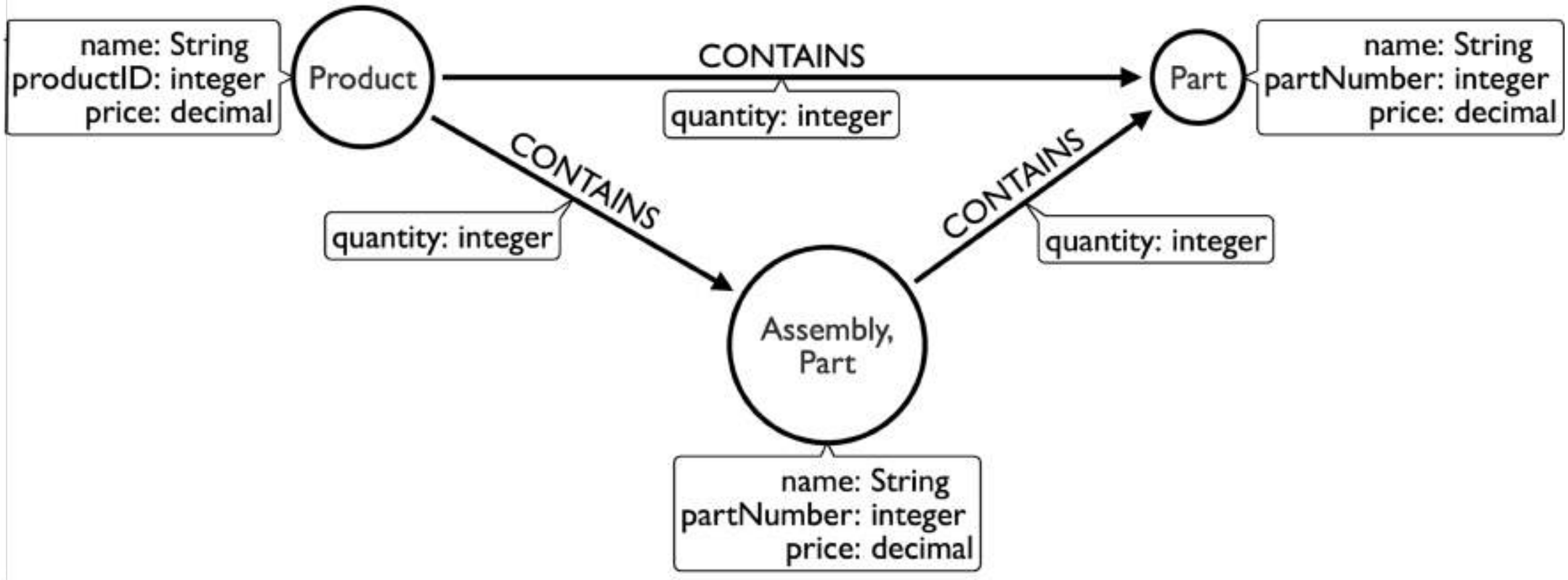
- quantity

Assembly -CONTAINS-> Part

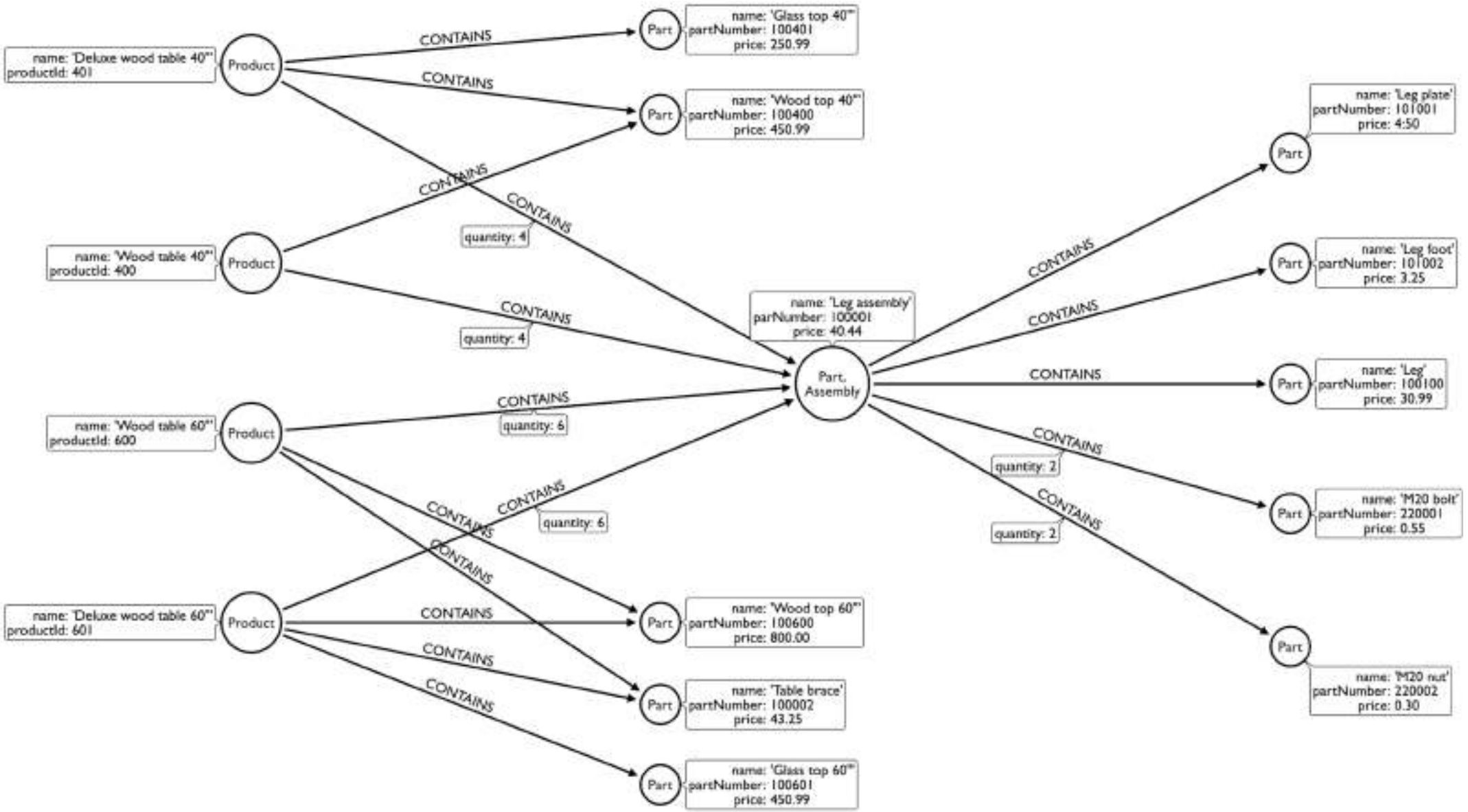
- quantity

Exercise 3 Solution

Your graph data model should look similar to this with relationships added:



Model Showing Graph of Sample Data



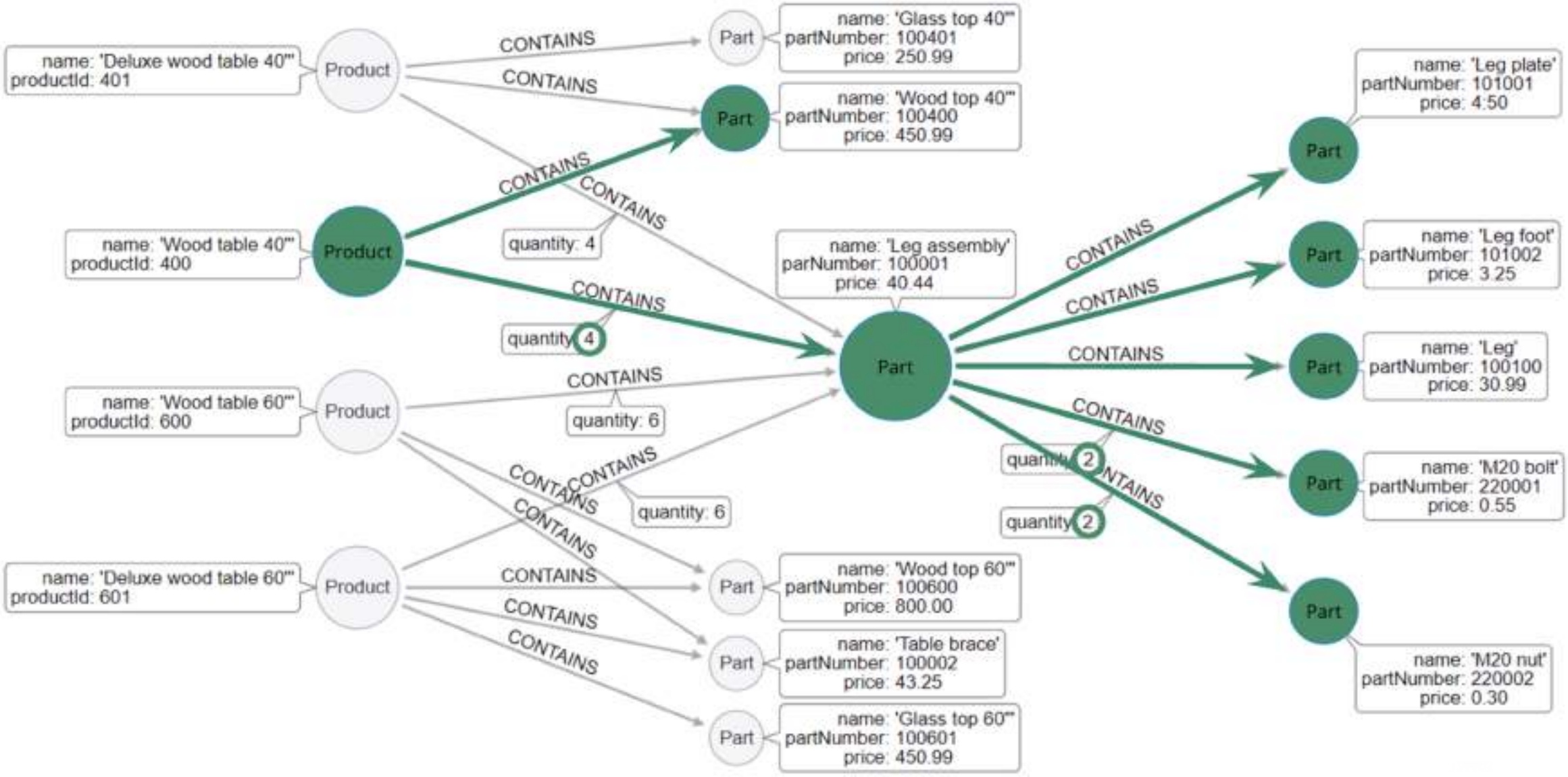
Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

6. Test the questions against the model

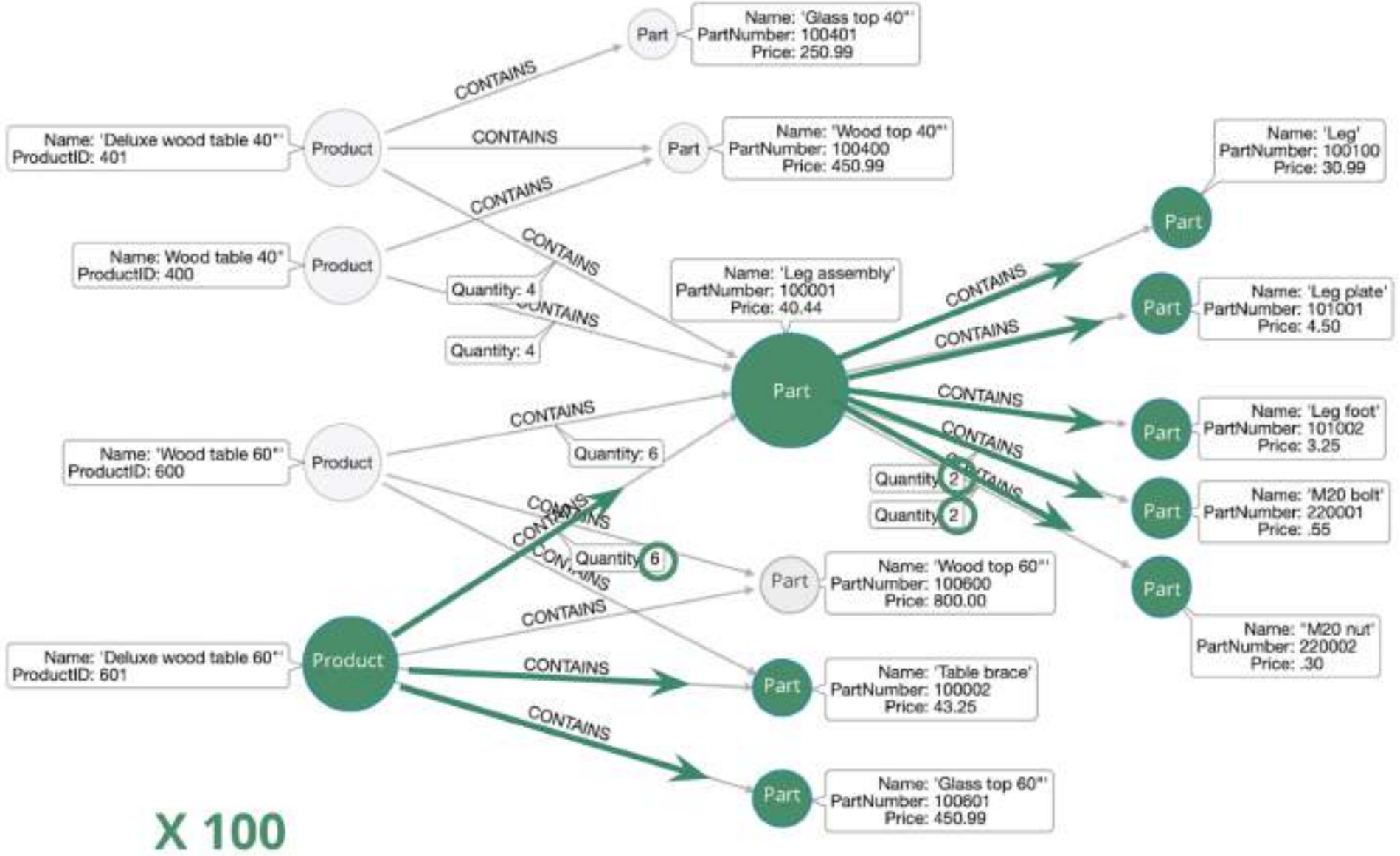
Testing the Model - 1

What parts are needed to make Wood table 40"?



Testing the Model - 2

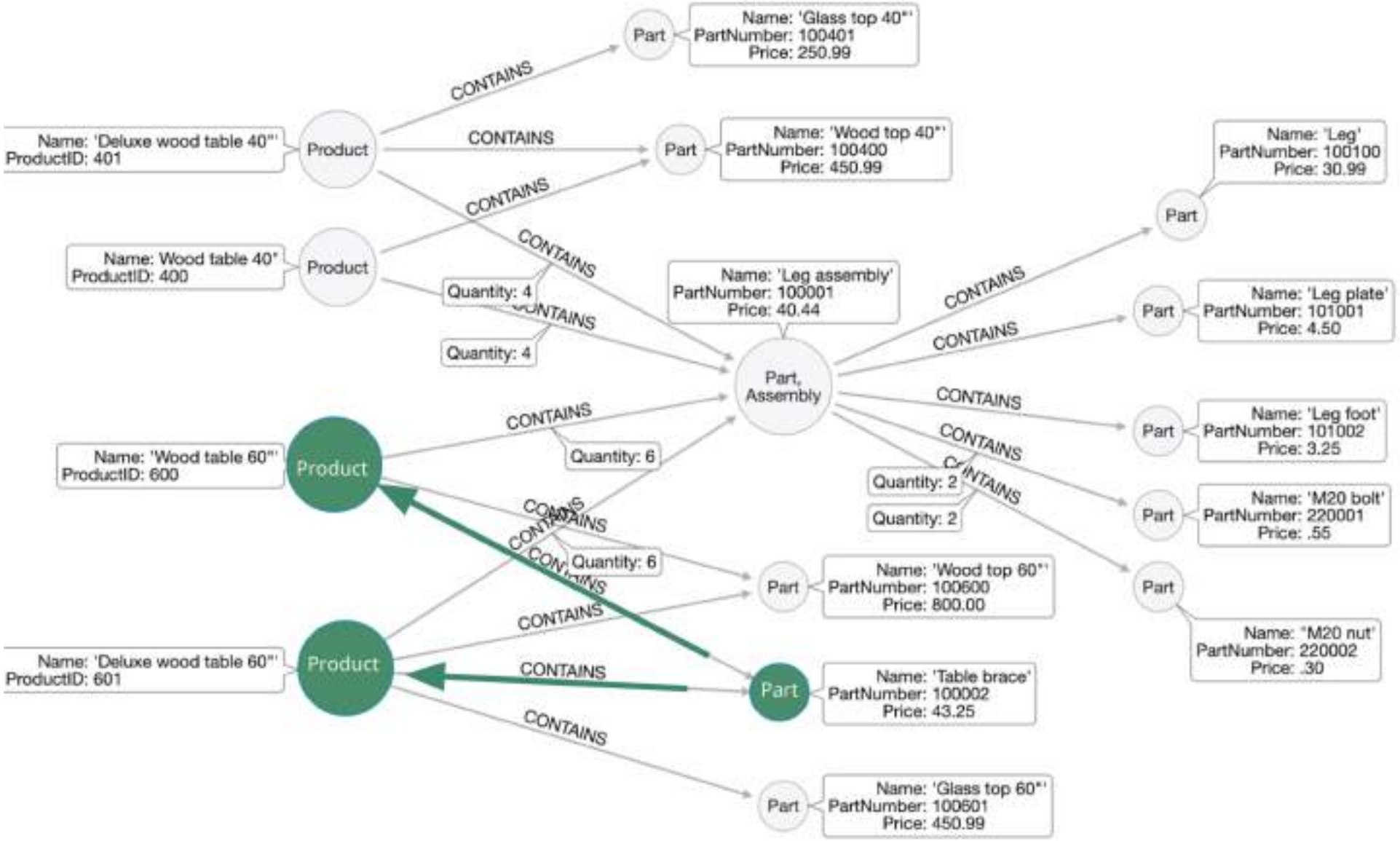
Do we have enough parts to make 100x Deluxe wood table 60"?



X 100

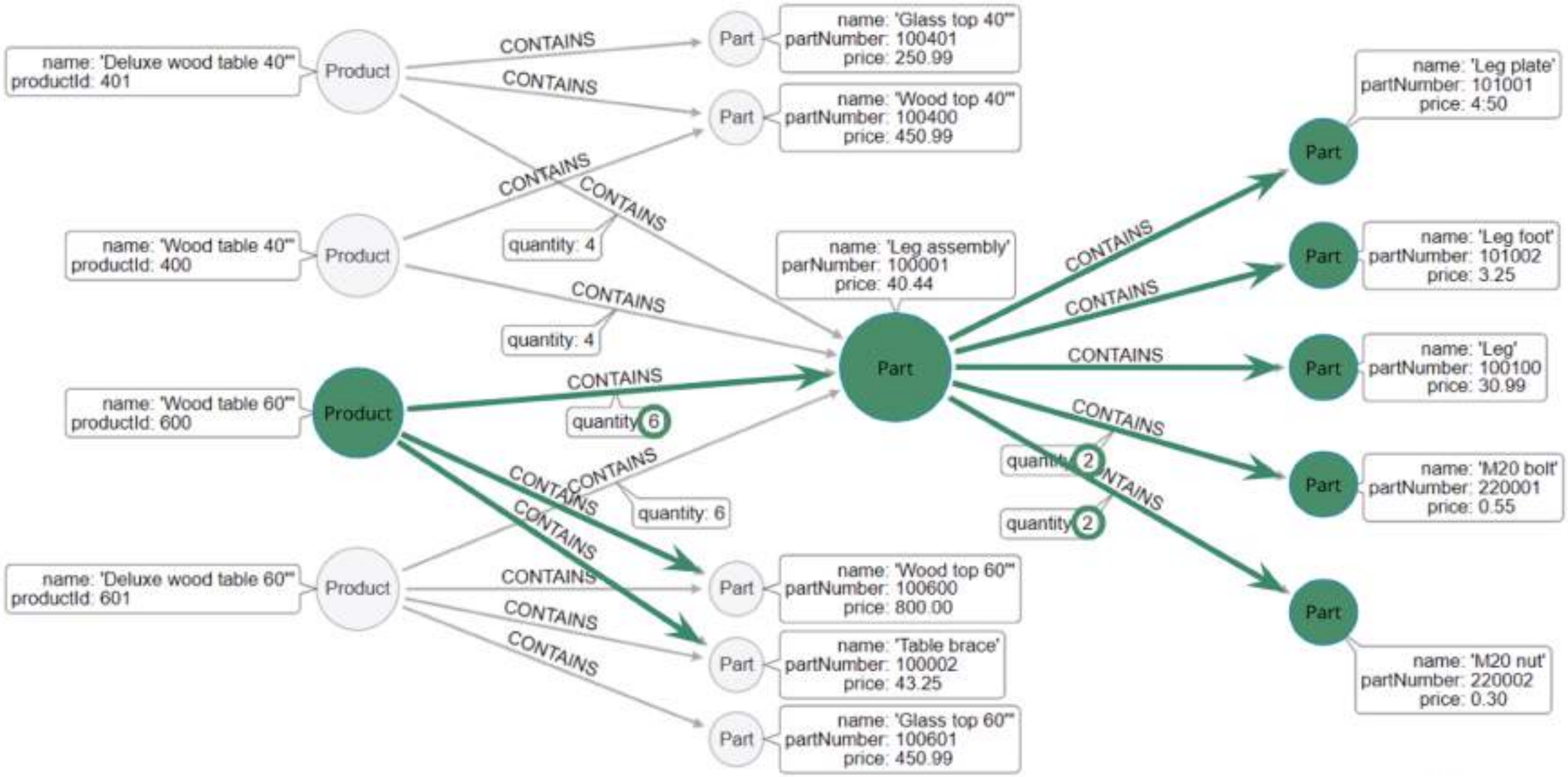
Testing the Model - 3

What products require a table brace?



Testing the Model - 4

How much will the parts cost to make Wood table 60"?

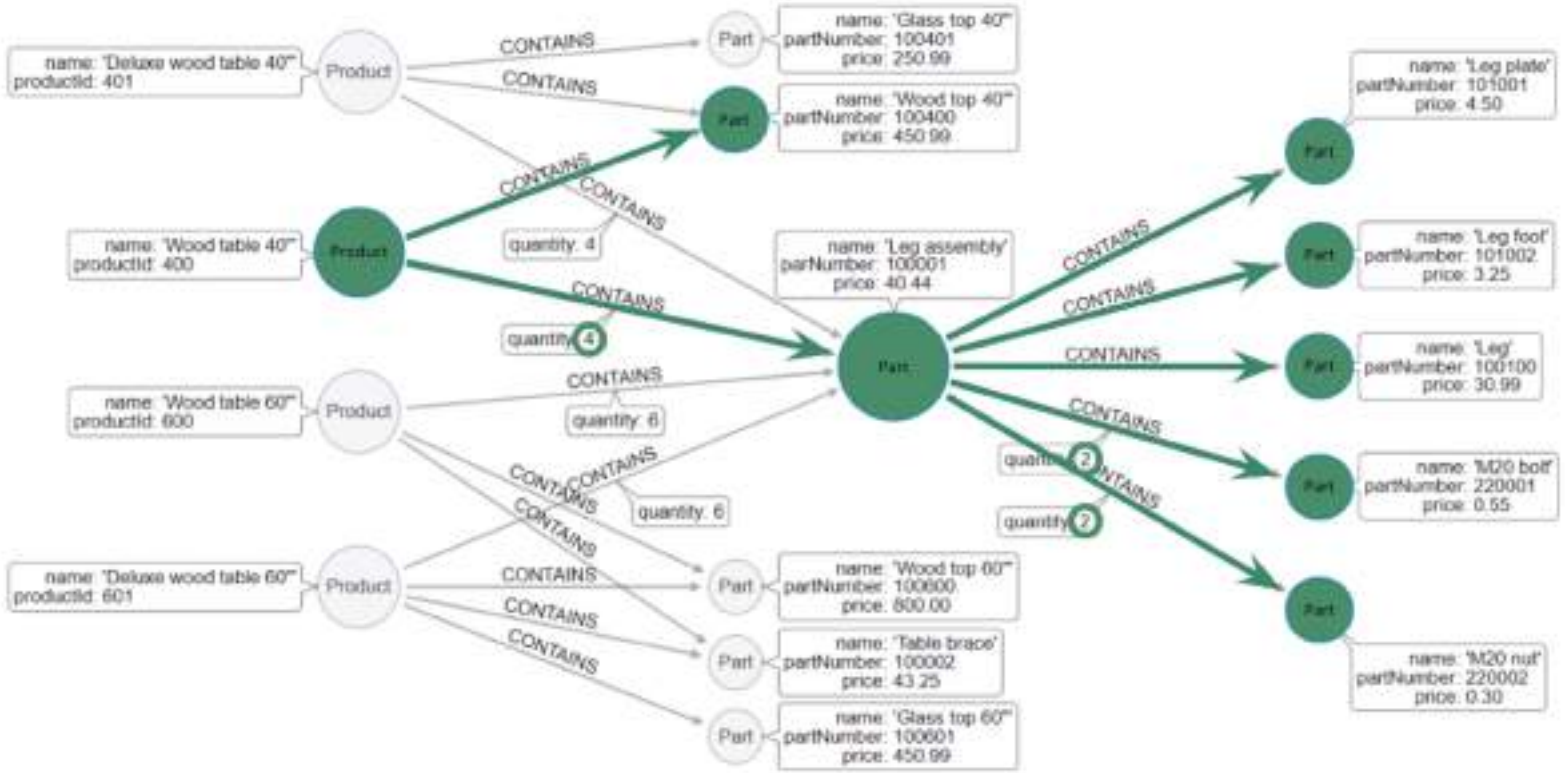


Designing the Initial Model

1. Understand the domain
2. Create sample data
3. Define specific questions
4. Identify nodes
5. Identify connections
6. Test the questions against the model
7. Test scalability

7. Test scalability

Testing Scalability



Scalability questions:

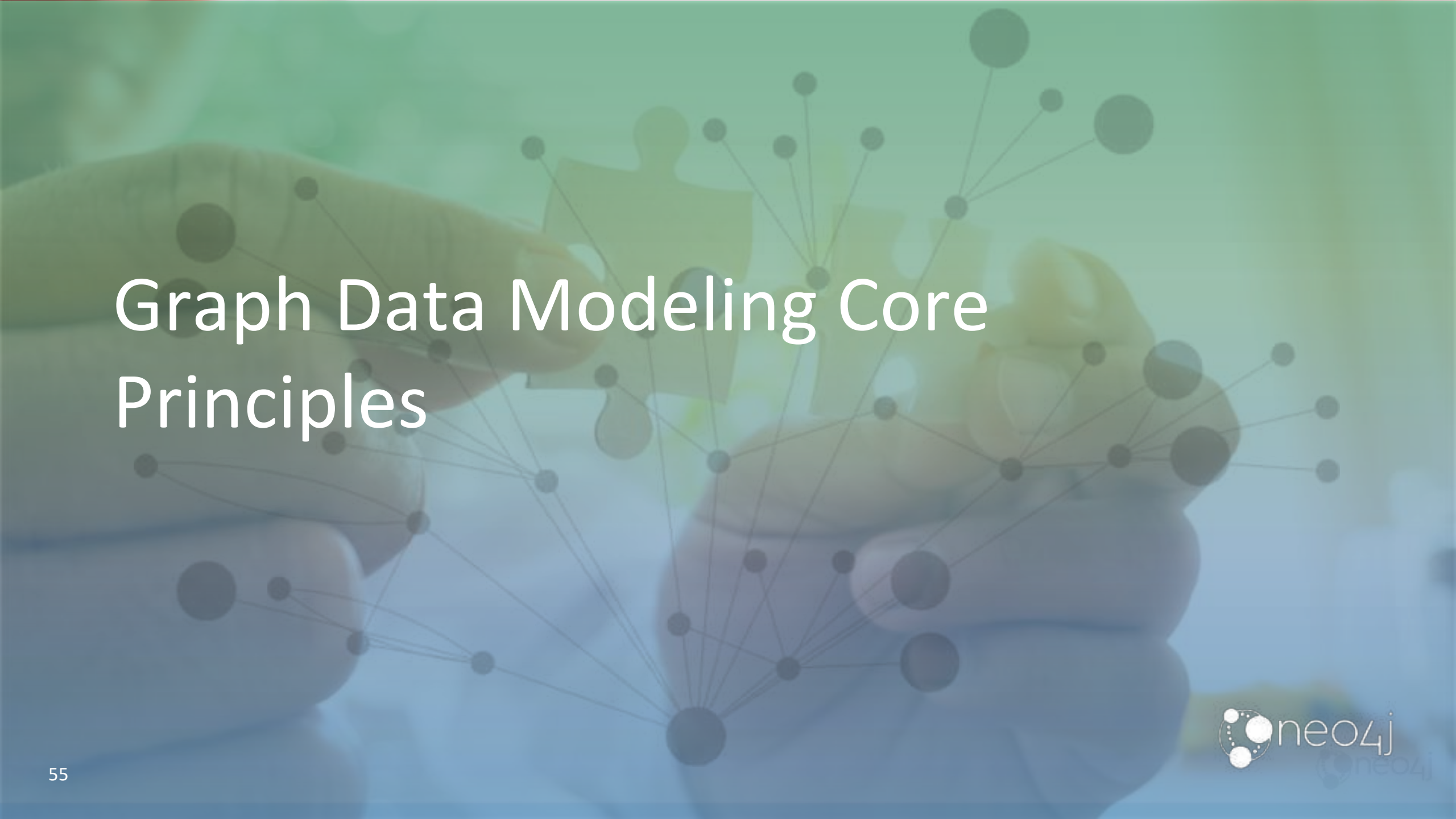
- How many products?
- How many parts?
- How often are products added?
- How often do prices change?
- Are prices based upon time?
- Is inventory part of the model?

Summary

You should now be able to:

- Describe the domain for a model
- Define the questions for the domain
- Identify entities from the questions for the domain
- Use the Arrows Tool to model the domain
- Identify the connections between entities
- Describe how to test the initial model

Graph Data Modeling Core Principles



In This Module You'll Learn ...

How to ...

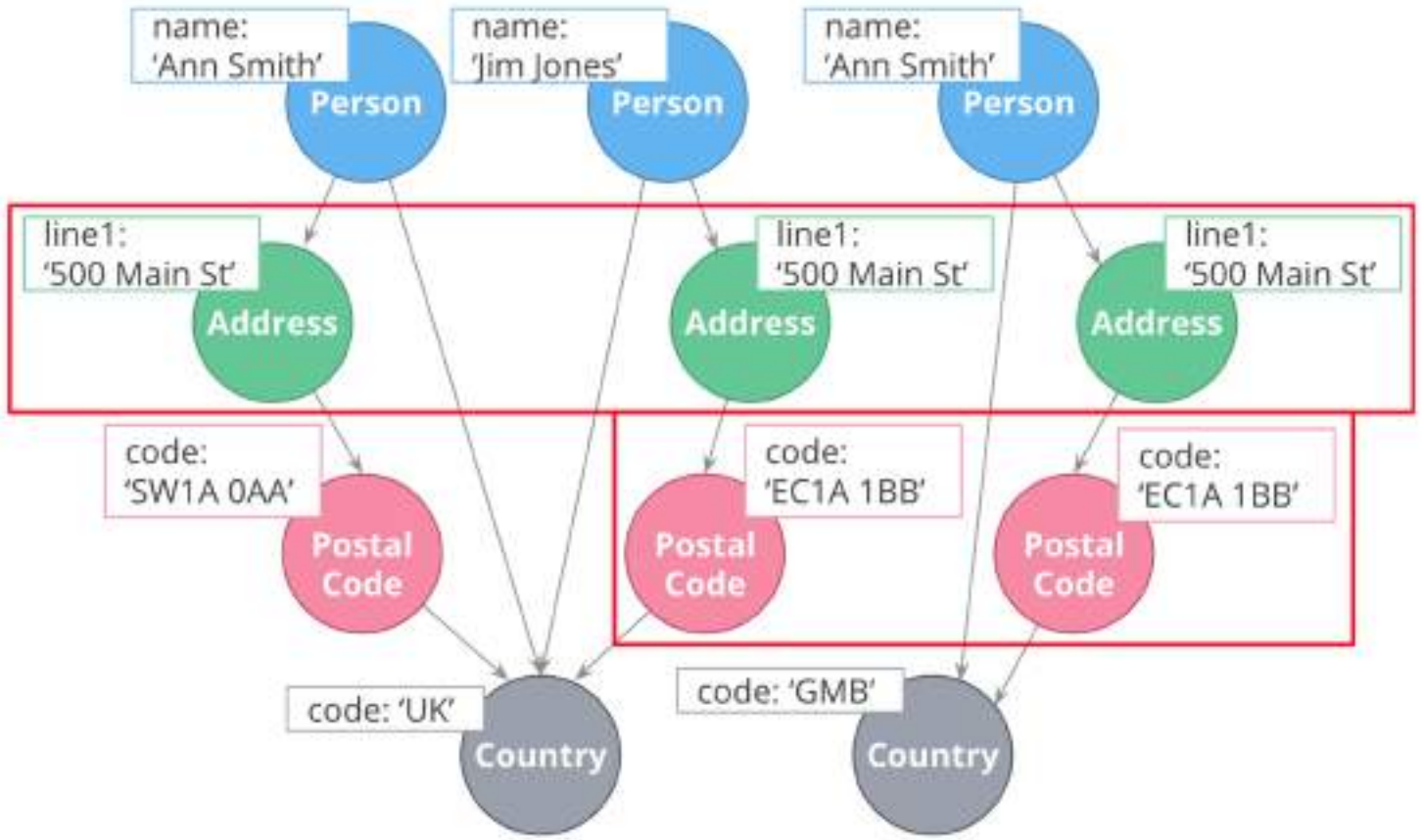
- Describe graph data modeling best practices for modeling:
  - Nodes (entities)
  - Relationships
  - Properties
- Describe data accessibility

Graph Modeling Core Principles

- Nodes
  - Uniqueness
  - Fanout
- Relationships
  - Naming best practices
  - Semantic redundancy
  - Types vs. Properties
- Properties
- Data object accessibility

Node Best Practices

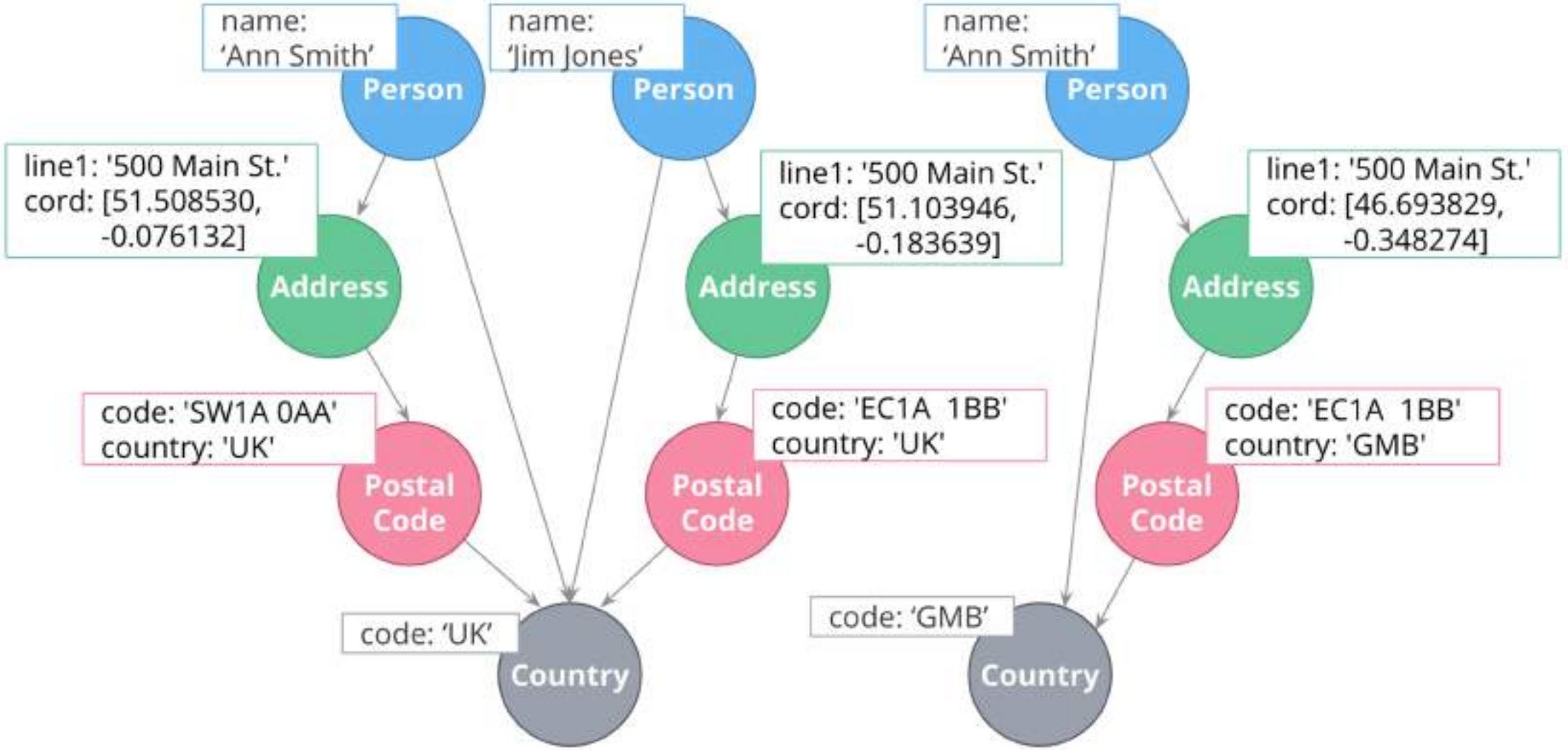
Uniqueness of Nodes: Before



Notes:

- Country nodes are considered **super nodes** (a node with lots of fan-in or fan-out)
- Use caution when using them in a design
- Be aware of queries that might select all paths in or out of a super node

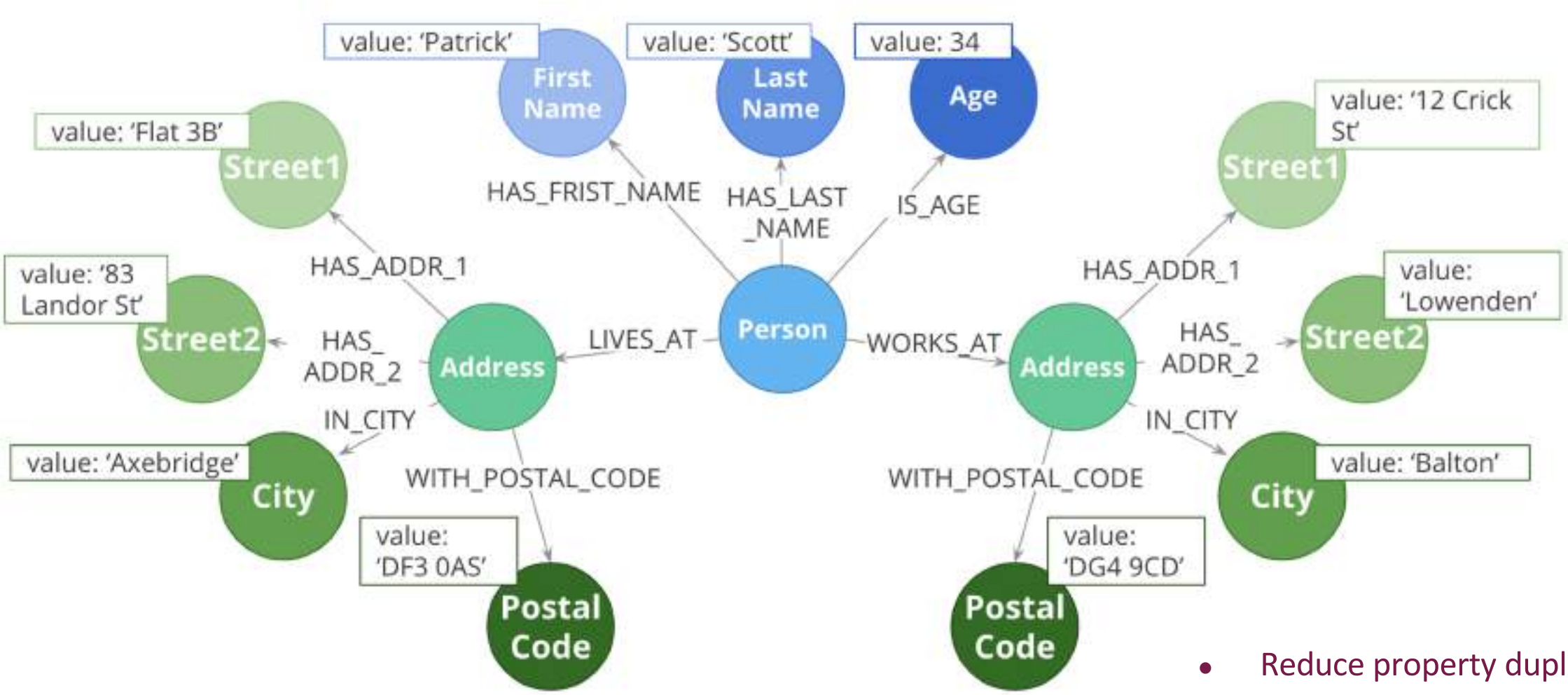
Uniqueness of Nodes: After



Complex Data

firstName: 'Patrick'
lastName: ‘Scott’
age: 34
homeAddress: [‘Flat 3B’, ‘83 Landor St.’, ‘Axebridge’, ‘DF3 OAS’]
workAddress: [‘Acme Ltd.’, ‘12 Crick St.’, ‘Balton’, ‘DG4 9CD’]

Use Fanout Judiciously for Complex Data



- Reduce property duplication
- Reduce gather-and-inspect

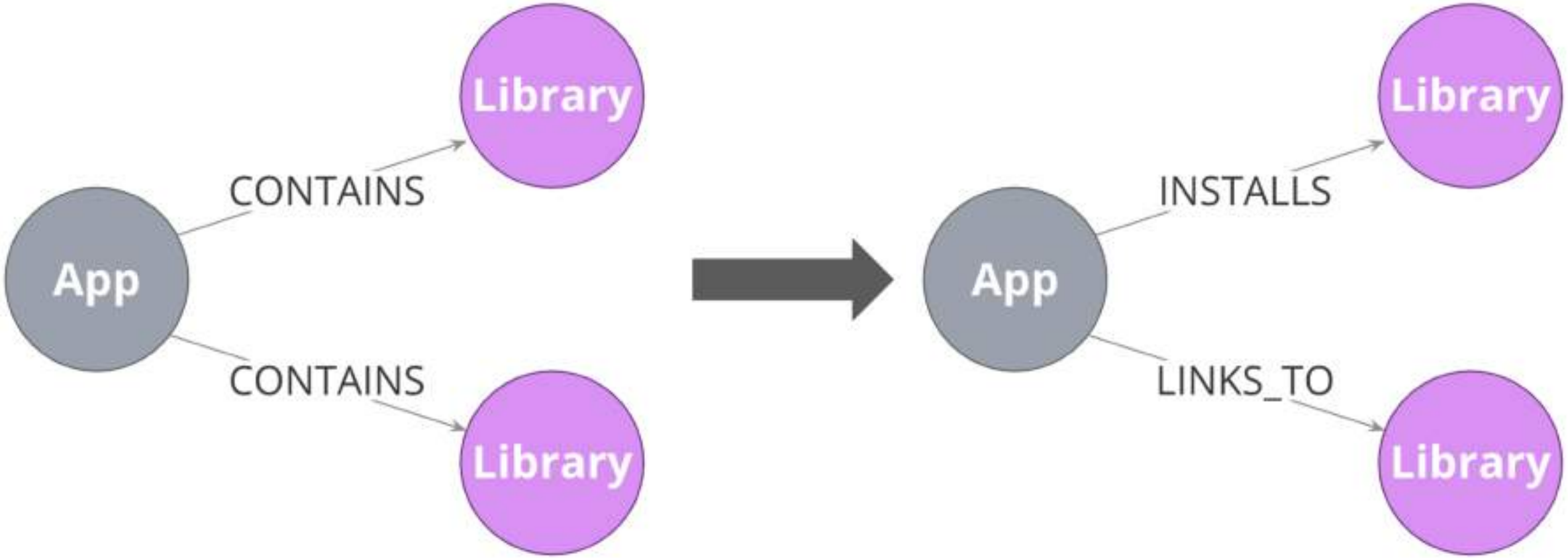
Relationship Best Practices

Best Practices
for Modeling Relationships

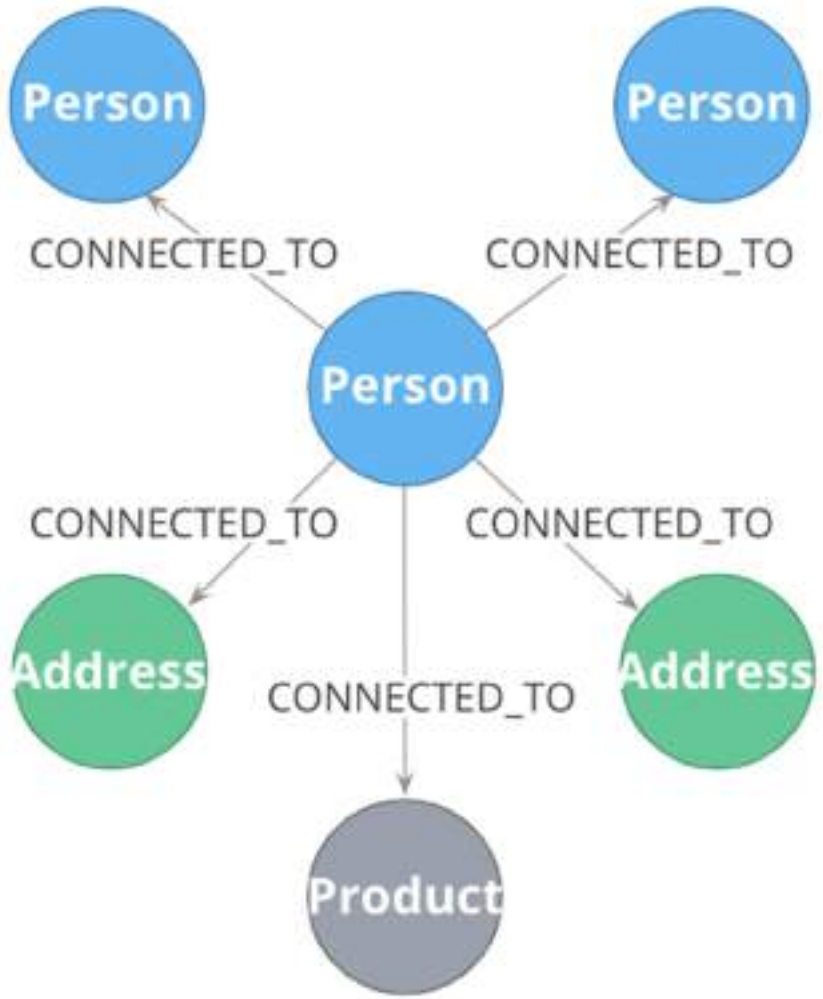
Data models should address:

- Using specific relationship types
- Using types vs. properties
- Reducing symmetric relationships

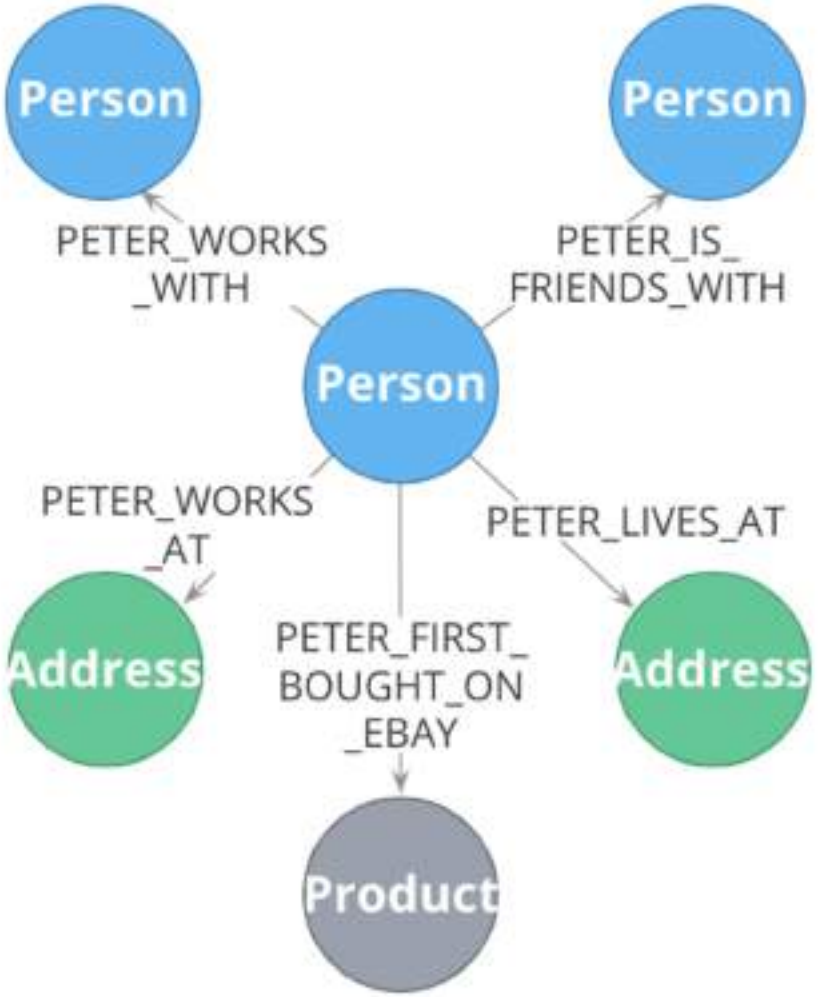
Using Specific Relationship Types



But Not Too Specific



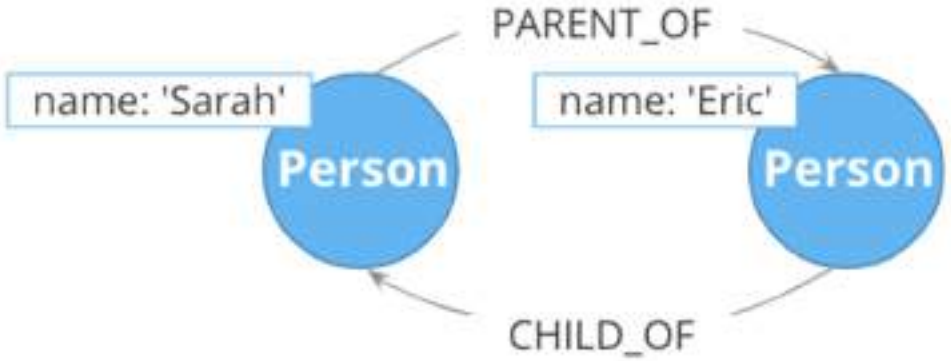
Not specific enough



Too specific

Do Not Use Symmetric Relationships

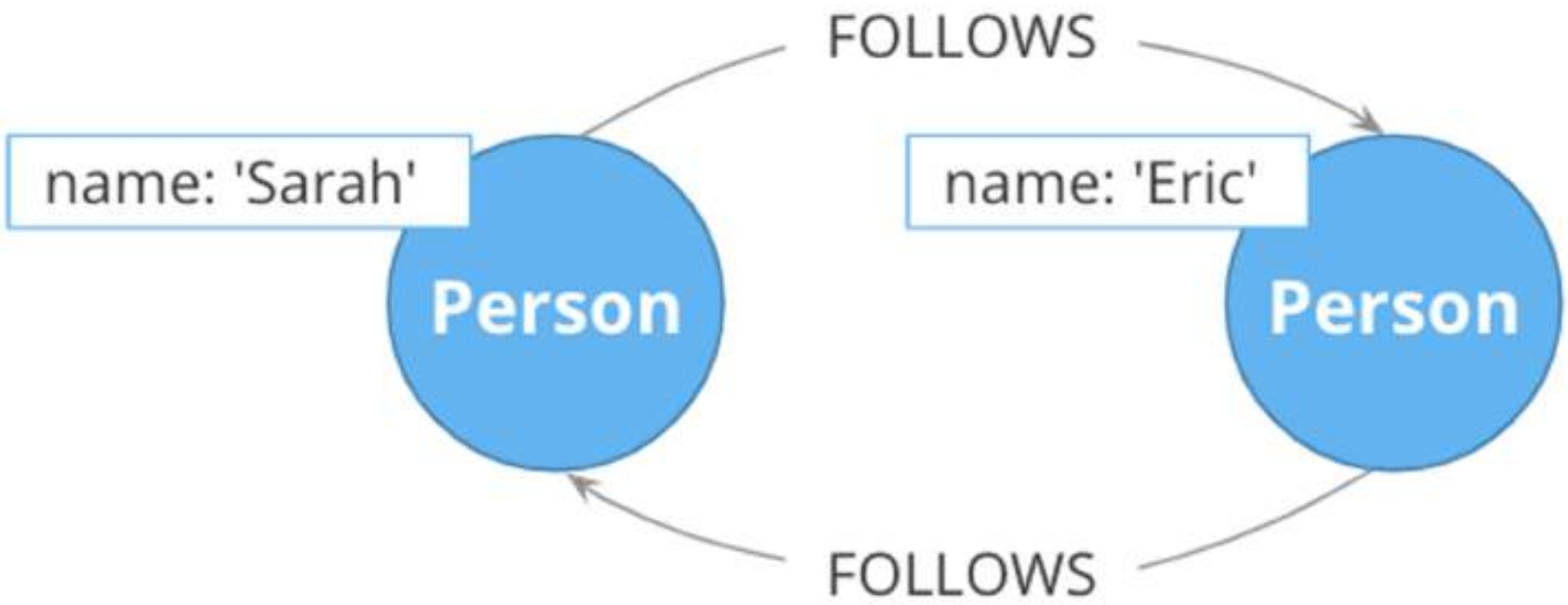
You should never do this:



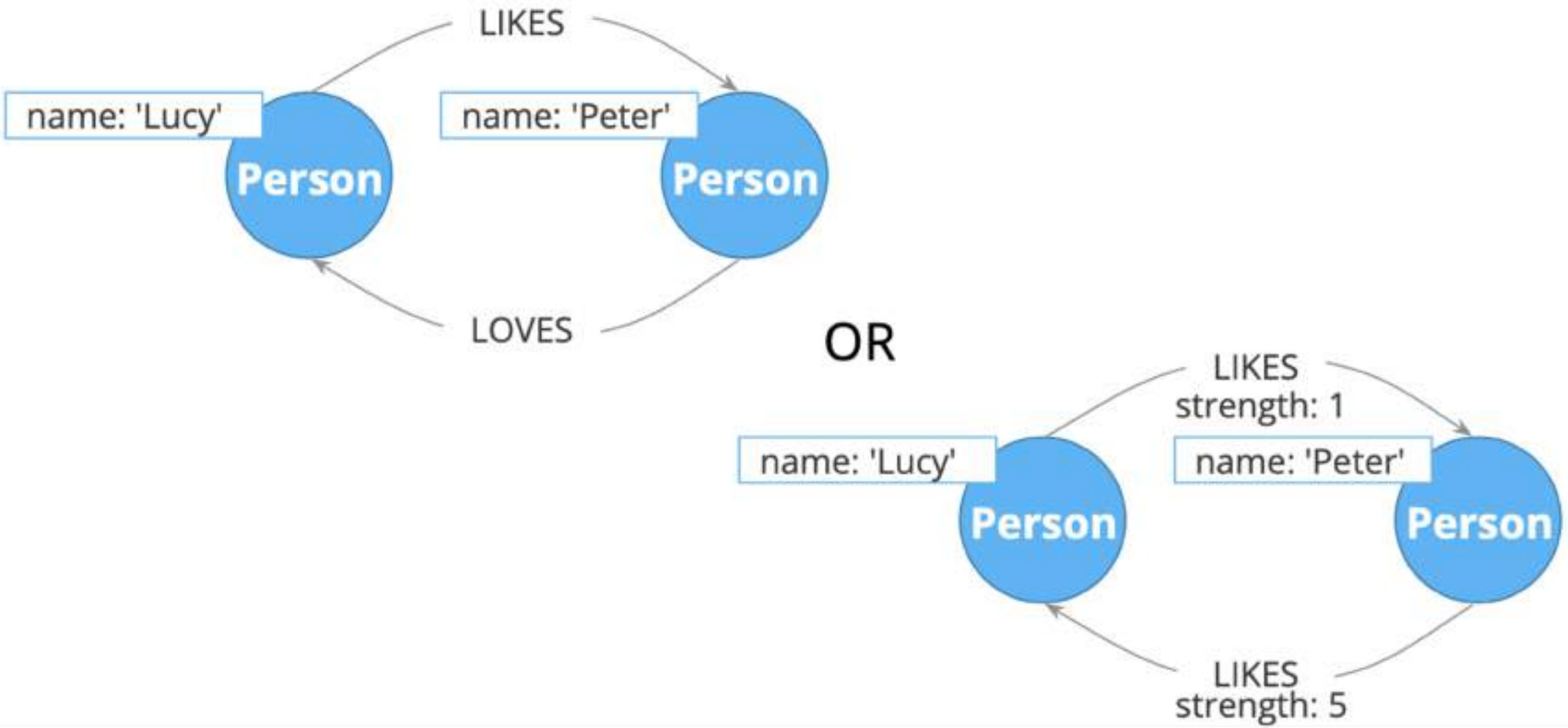
Do one of these:

or

Semantics of Symmetry are Important



Using Types vs. Properties



Property
Best Practices

Property Best Practices

- Property lookups have a cost
- Parsing a complex property adds more cost

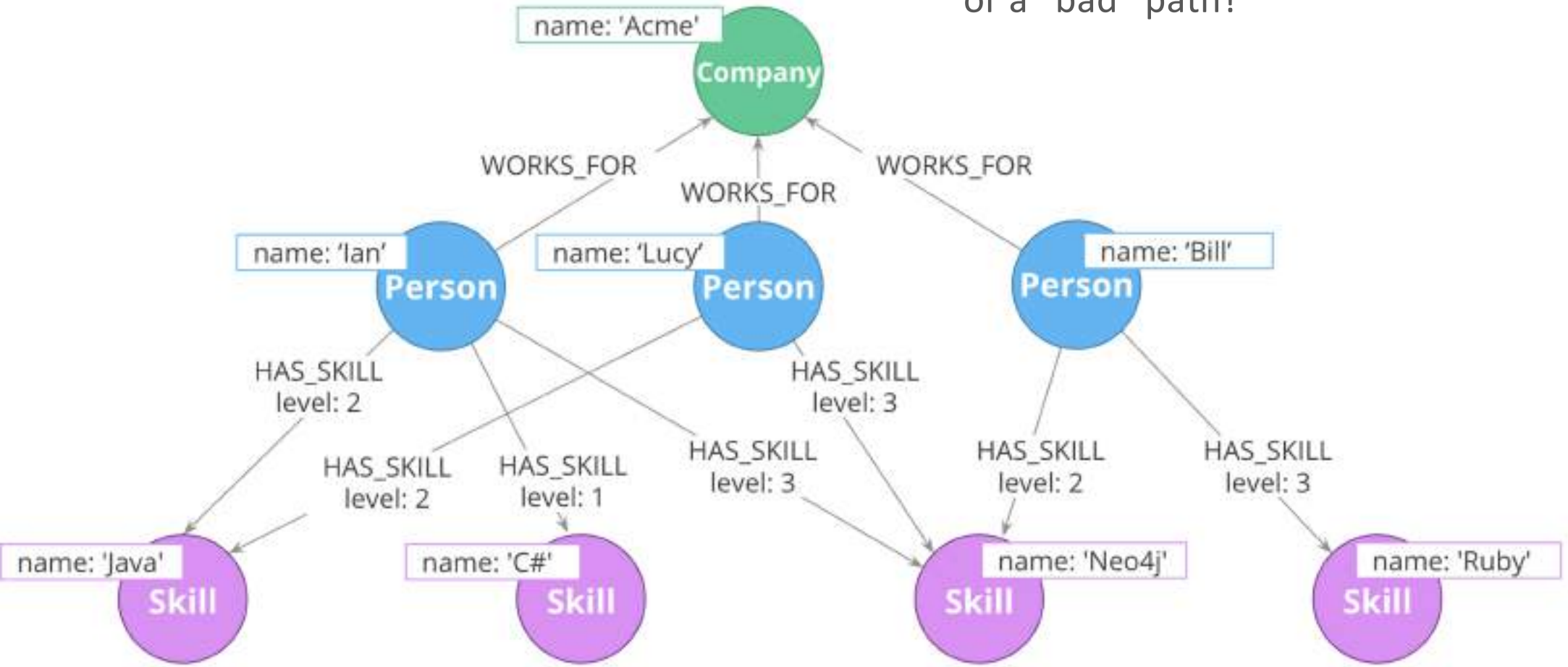
- Anchors and properties used for traversal should be as simple as possible

Data Object Accessibility Best Practices

neo4j

Data Accessibility

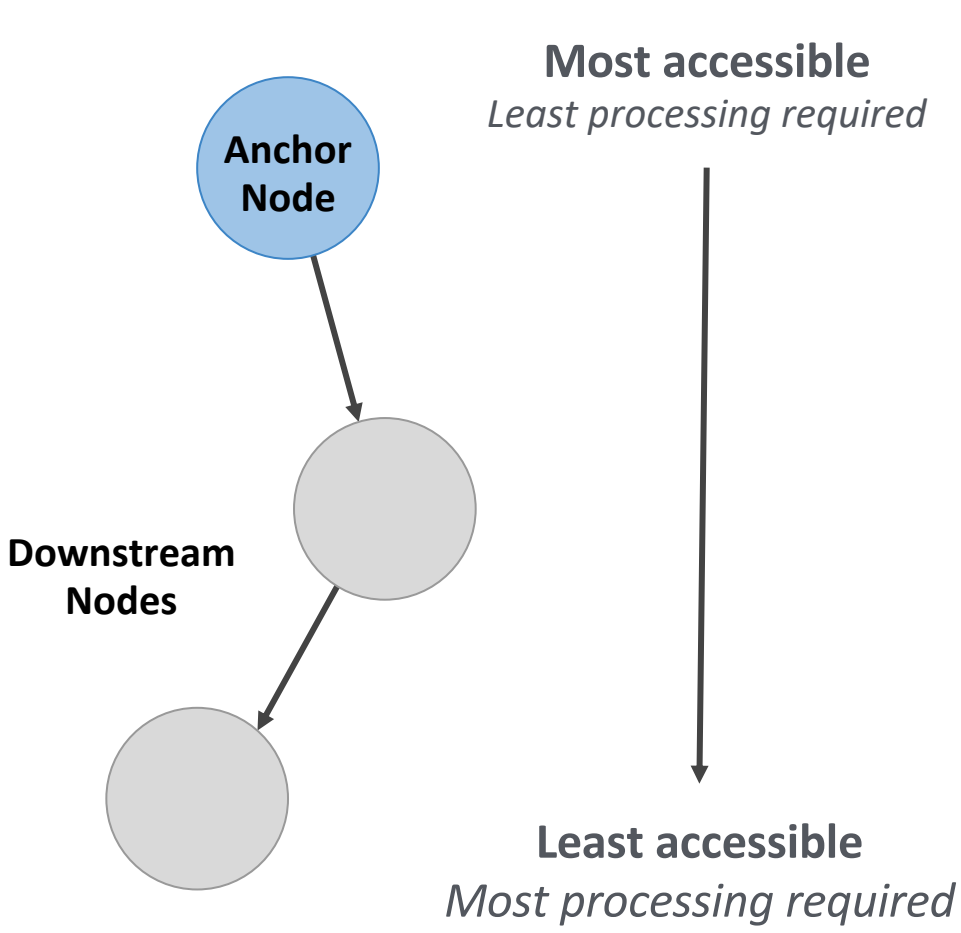
For each query, how much work must Neo4j do to evaluate if the traversal represents a “good” or a “bad” path?



neo4j

Hierarchy of Accessibility

For each data object, how much work must Neo4j do to evaluate if this is a “good” path or a “bad” one?



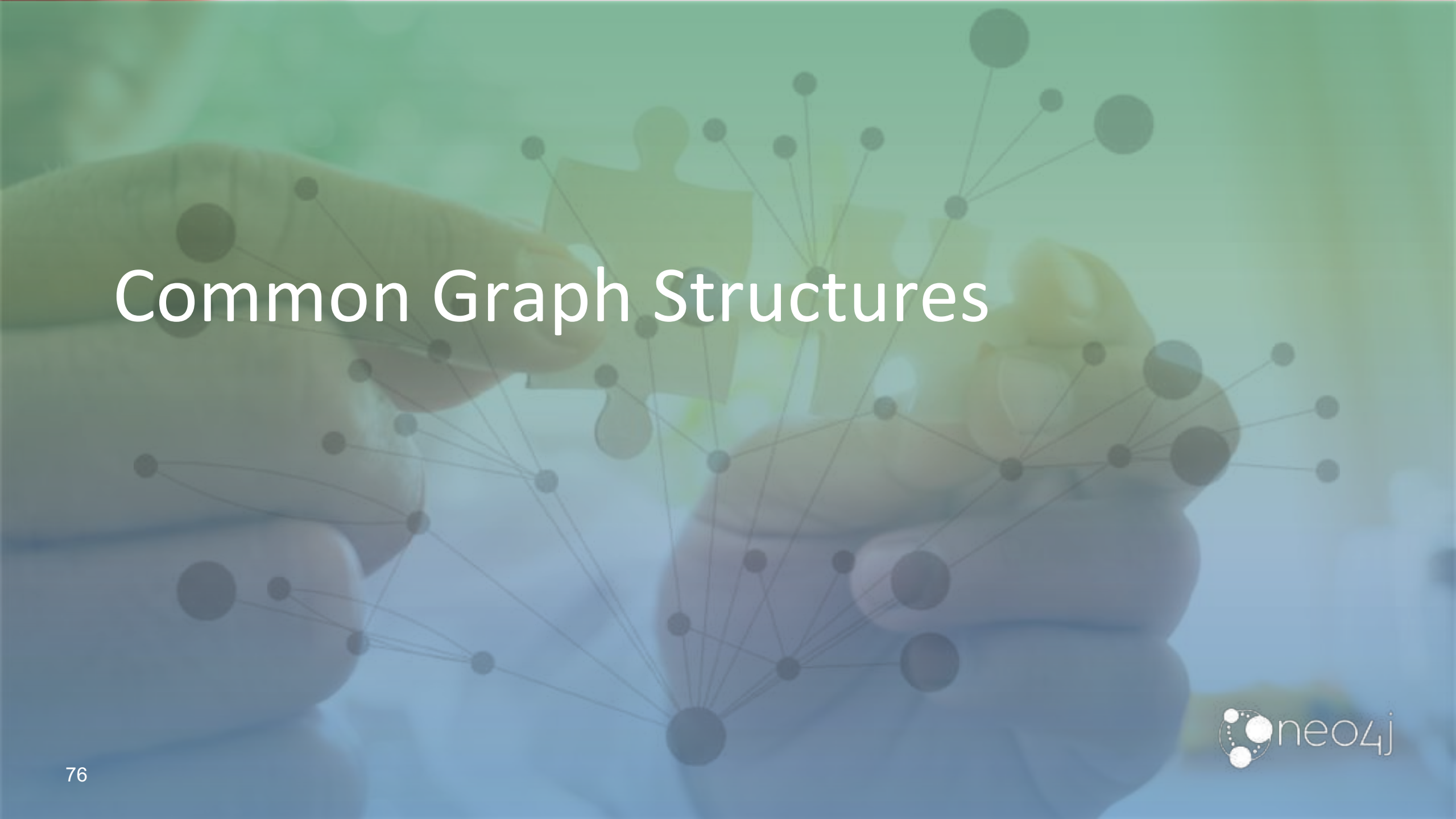
1. Anchor node label
   Anchor node properties (indexed)
2. Relationship type
3. Anchor node properties (non-indexed)
4. Downstream node labels
5. Relationship properties
   Downstream node properties

Summary

You should now be able to:

- **Describe graph data modeling best practices for modeling:**
  - **Nodes (entities)**
  - **Relationships**
  - **Properties**
- **Describe data accessibility**

Common Graph Structures



In This Module You'll Learn ...

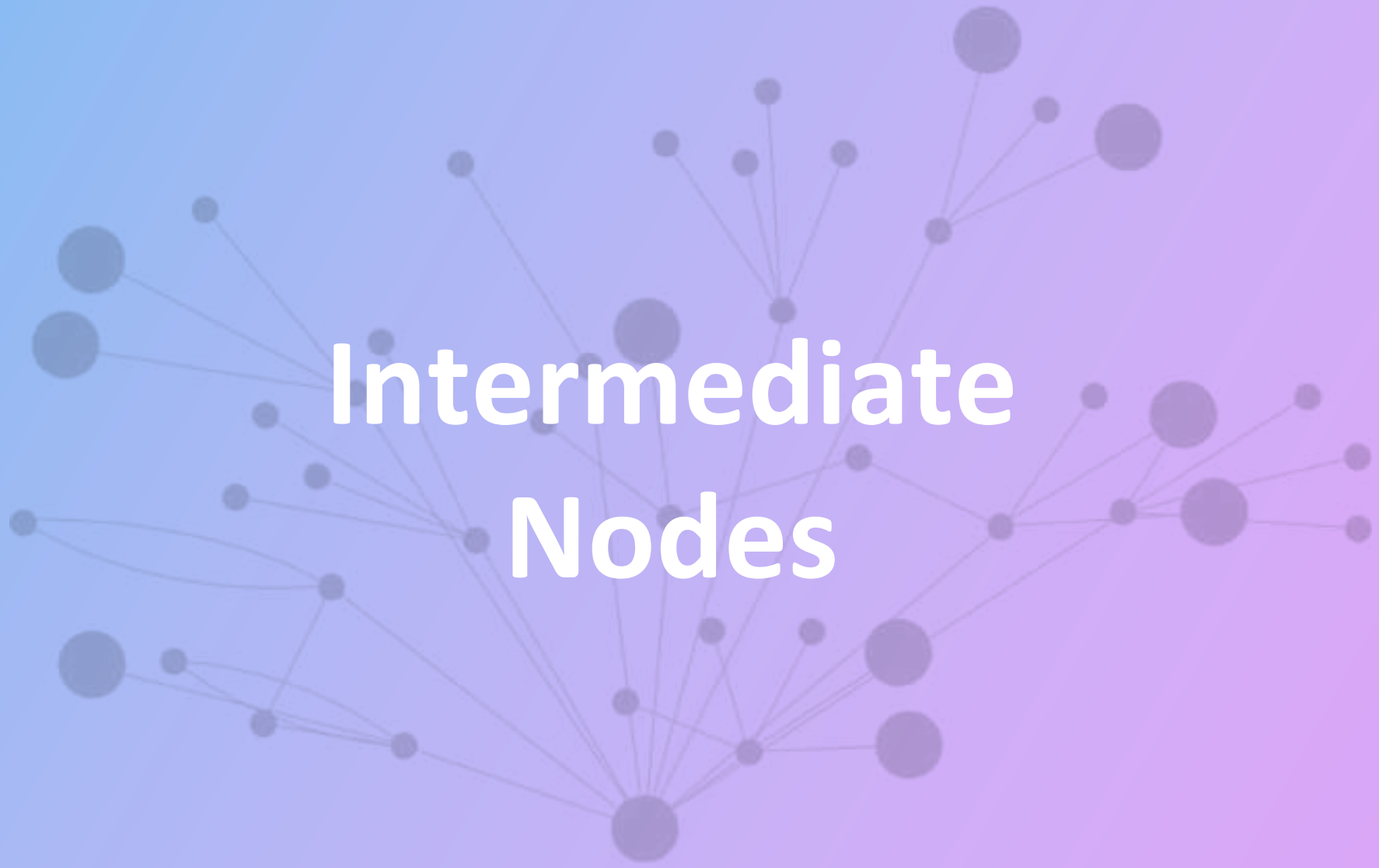
How to ...

- Describe common graph structures used in modeling:
  - Intermediate nodes
  - Linked lists
  - Timeline trees
  - Multiple structures in a single graph

Common Graph Structures

- Intermediate node
- Linked list
- Timeline tree
- Multiple structures in a single model

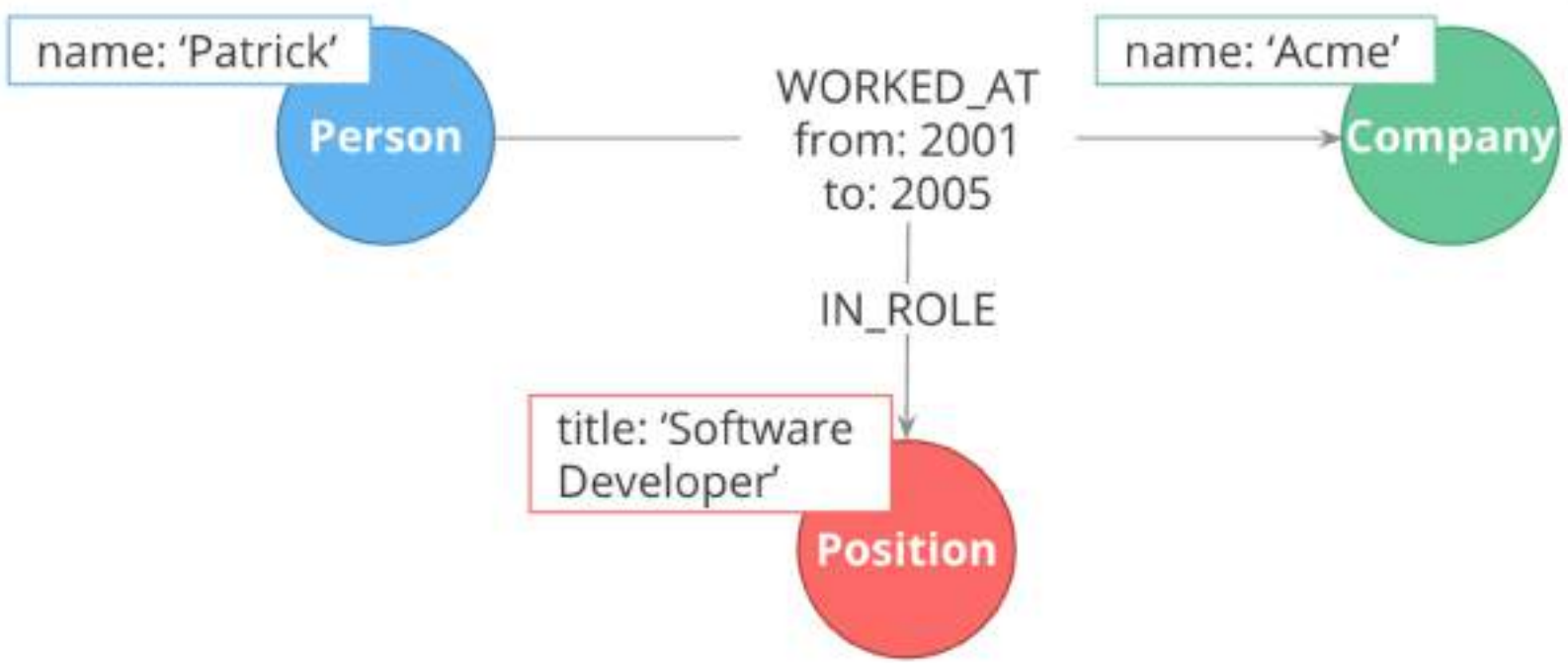
Intermediate Nodes



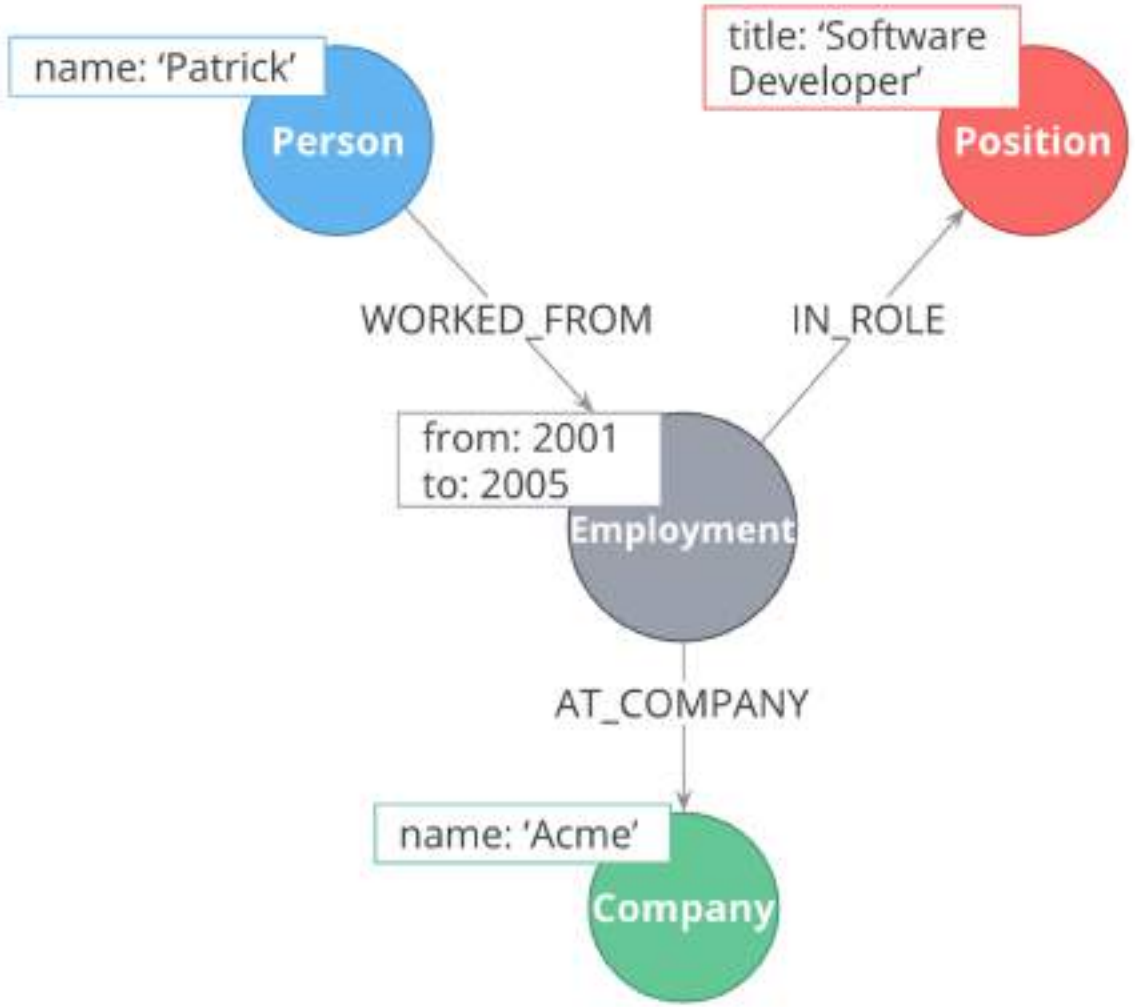
Intermediate Nodes

Create intermediate nodes when you need to:

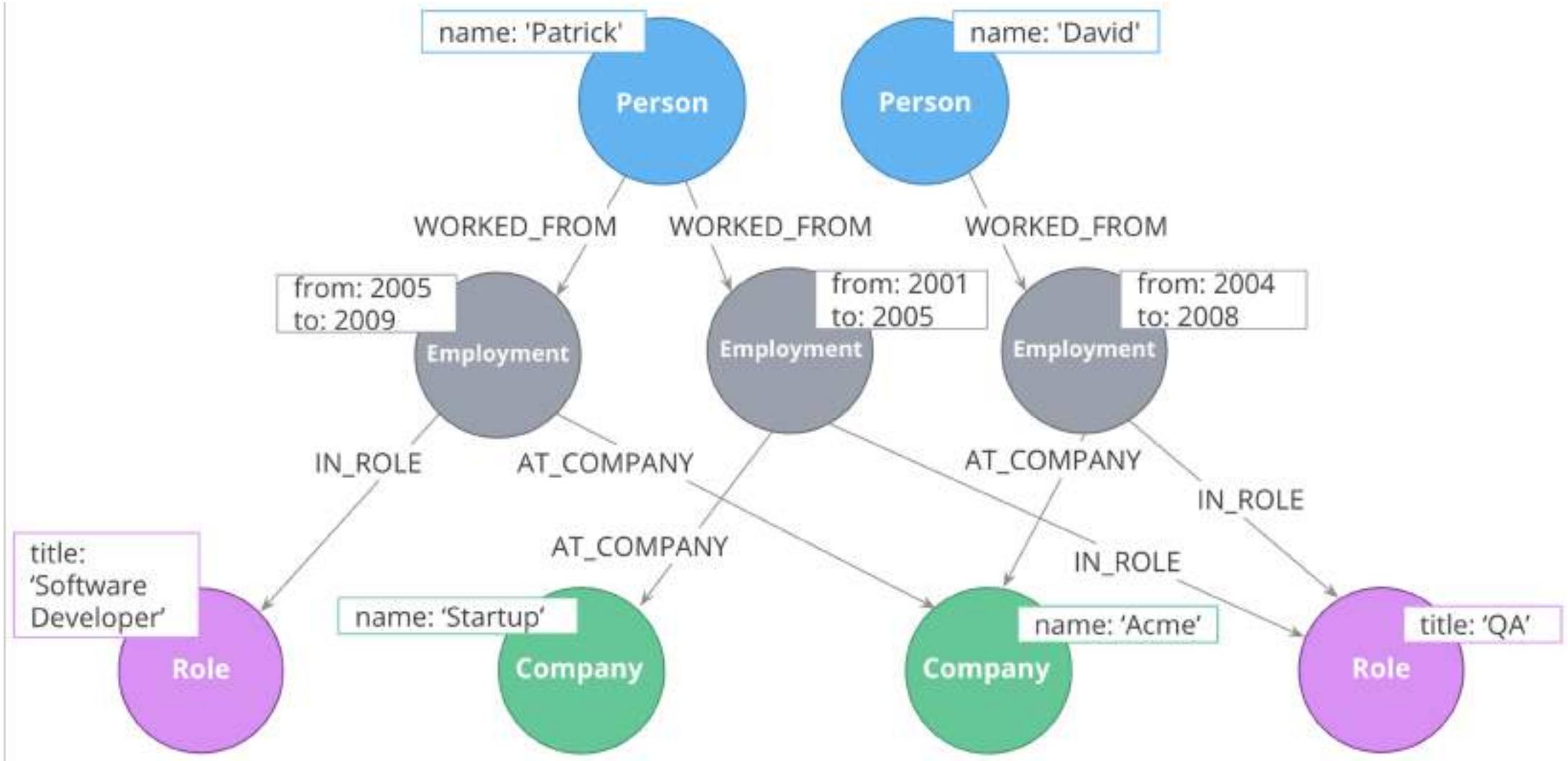
- Connect more than two nodes in a single context
- Relate something to a relationship



Using Intermediate Nodes

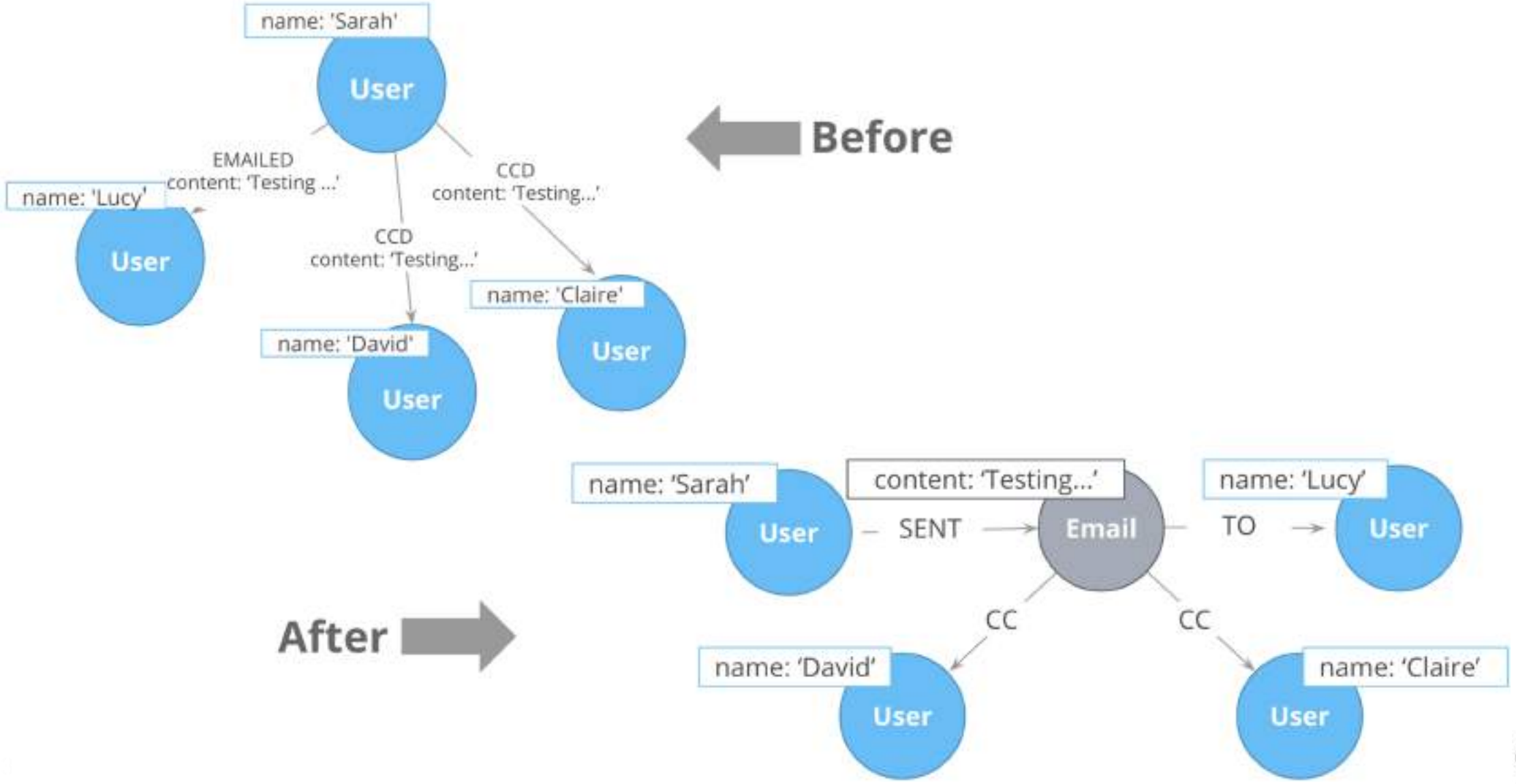


Intermediate Nodes: Sharing Context

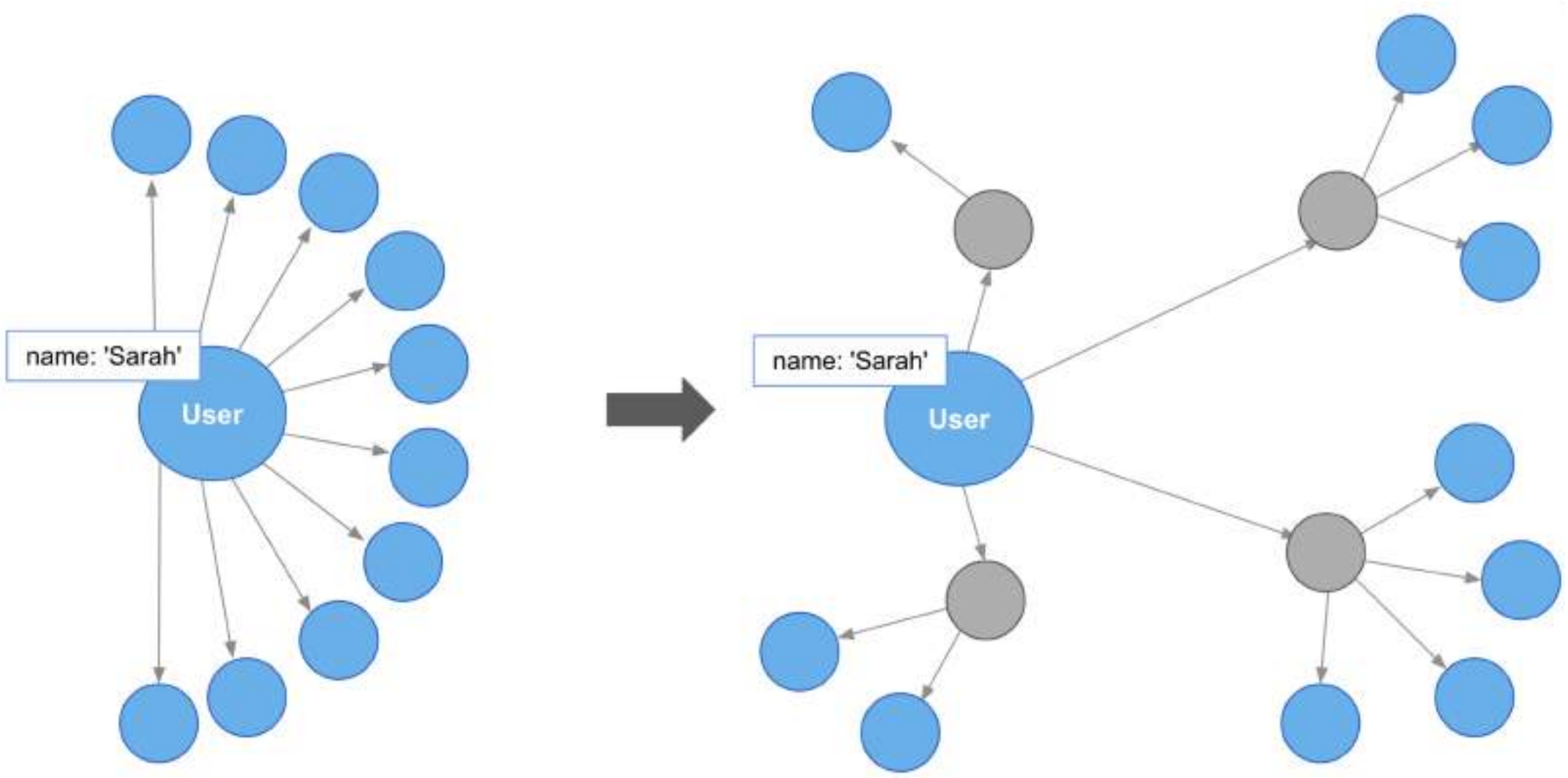


neo4j

Intermediate Nodes: Sharing Data



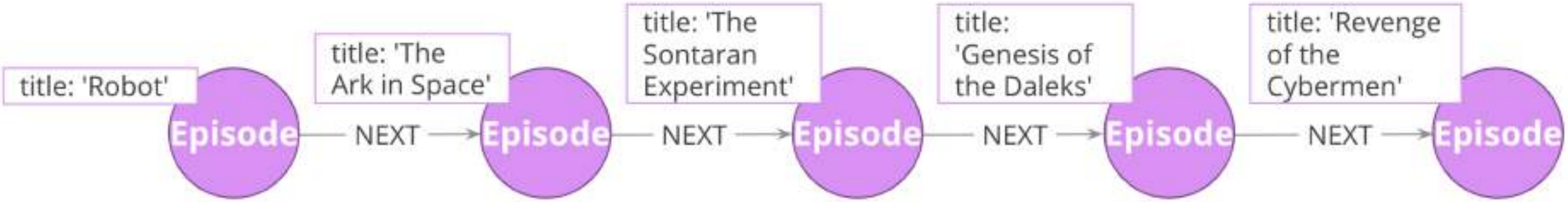
Intermediate Nodes: Organizing Data



Linked Lists

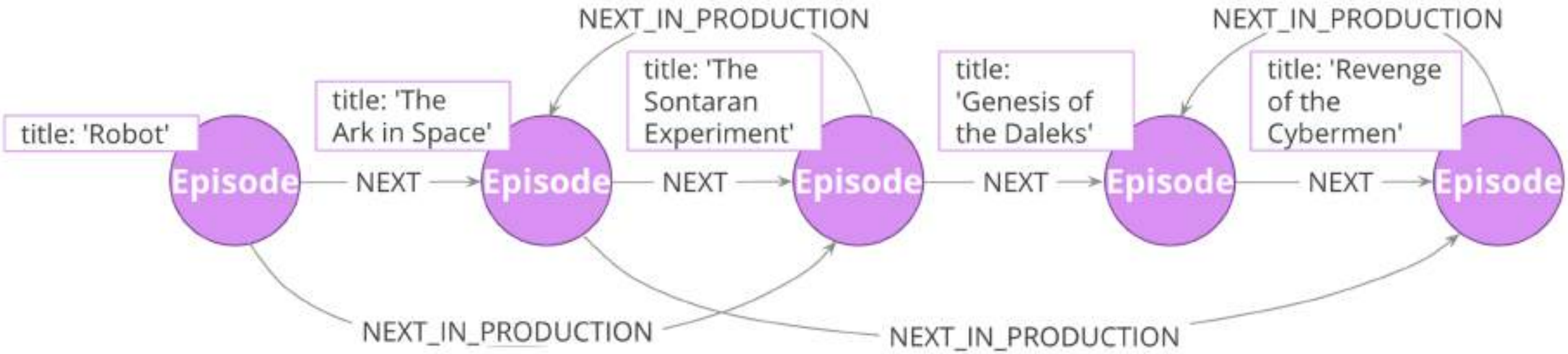
Linked Lists

Episodes of the Dr.Who series:



Do NOT do this (doubly-linked list):

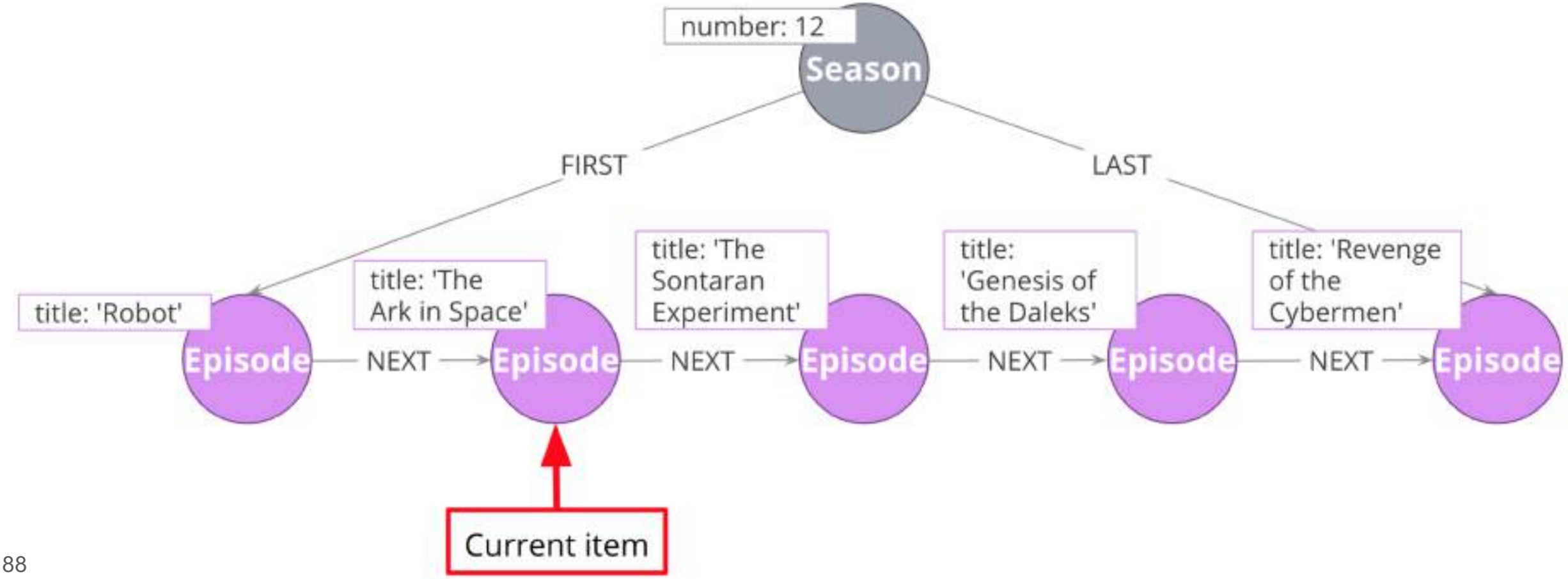
Interleaved Linked List



Head and Tail of Linked List

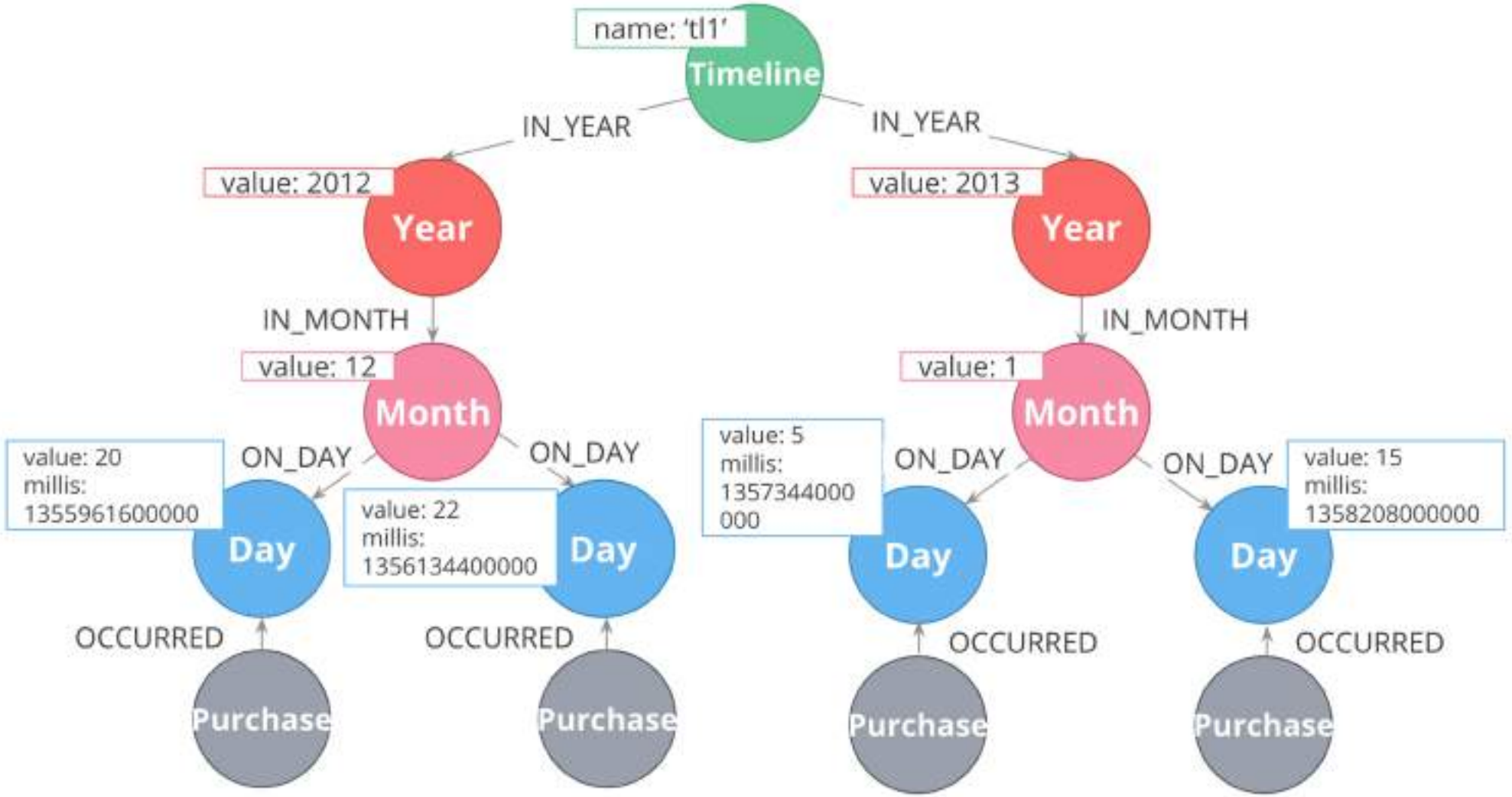
Some possible use cases:

- Add episodes as they are broadcast
- Maintain pointer to first and last episodes
- Find all broadcast episodes
- Find latest broadcast episode



Timeline Tree

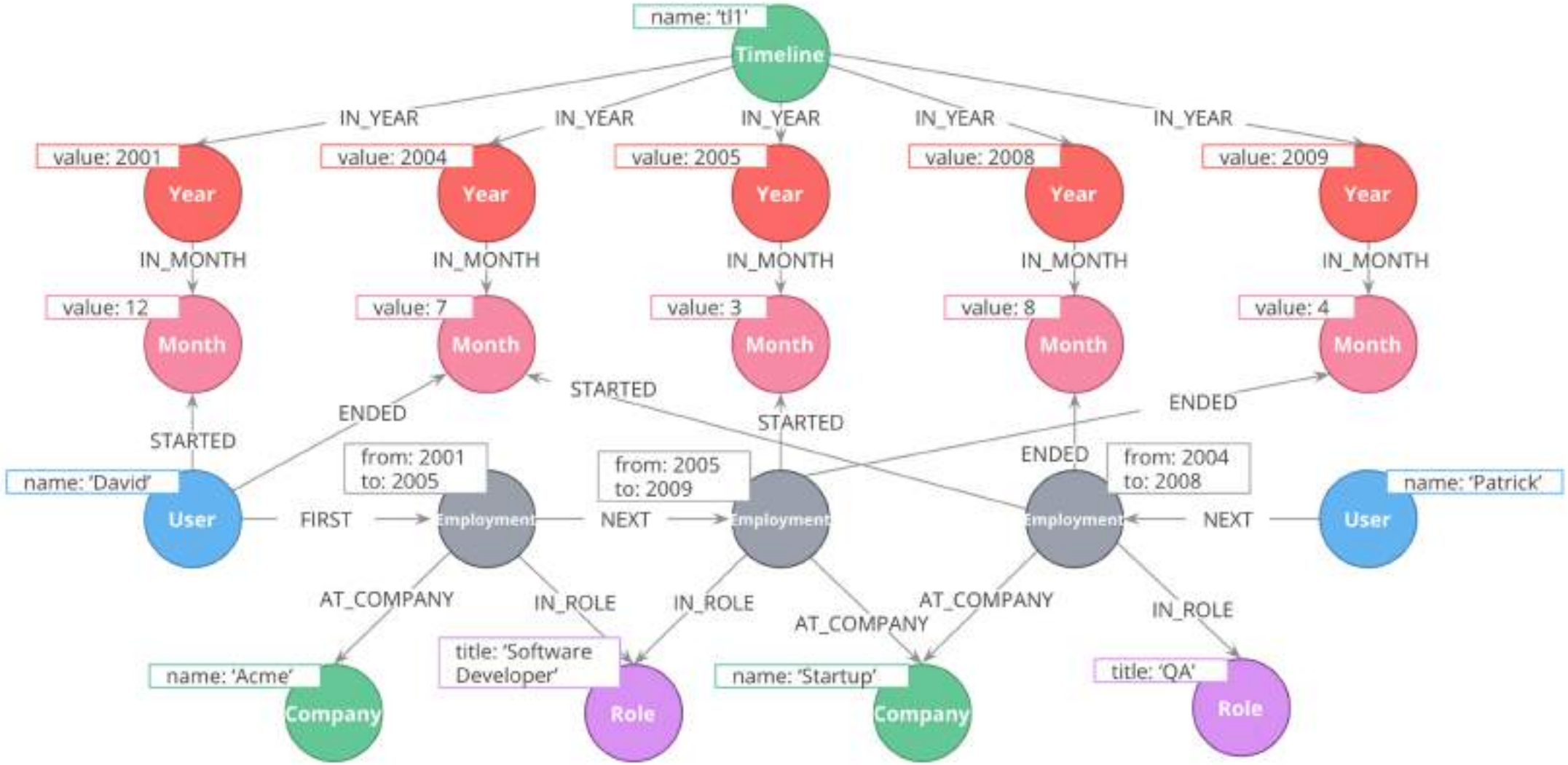
Timeline Tree



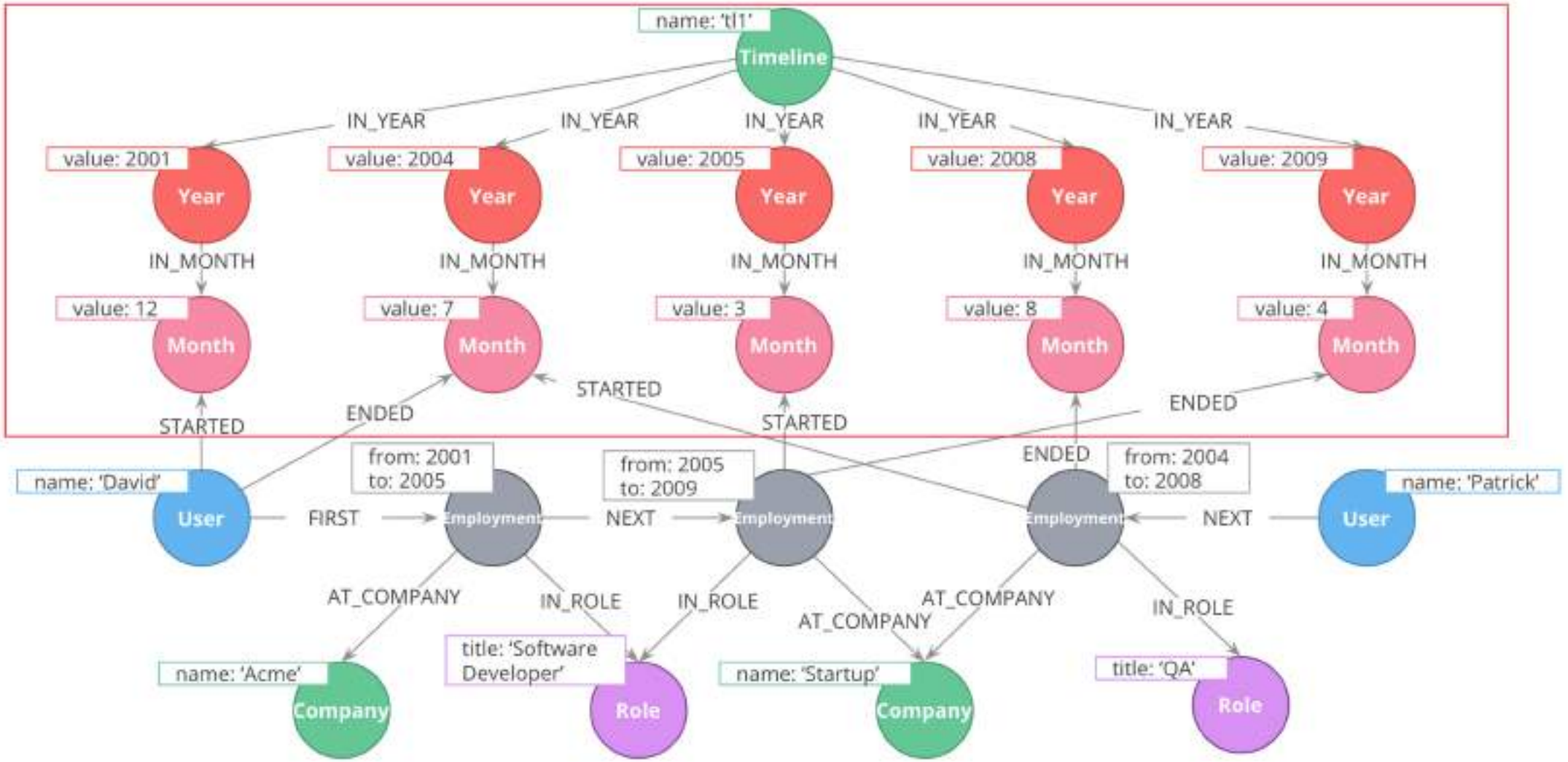
neo4j

Using Multiple Structures

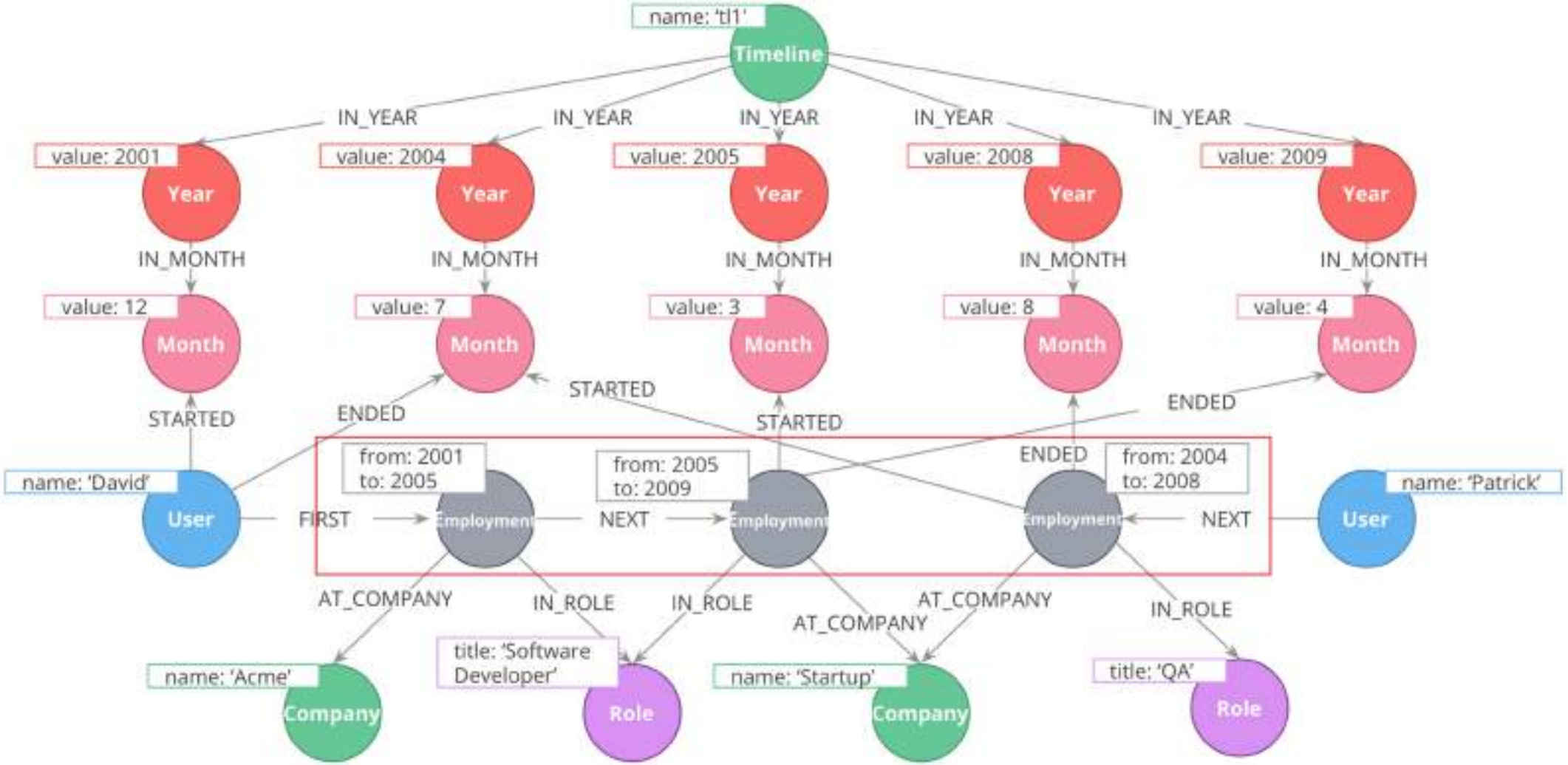
Using Multiple Structures



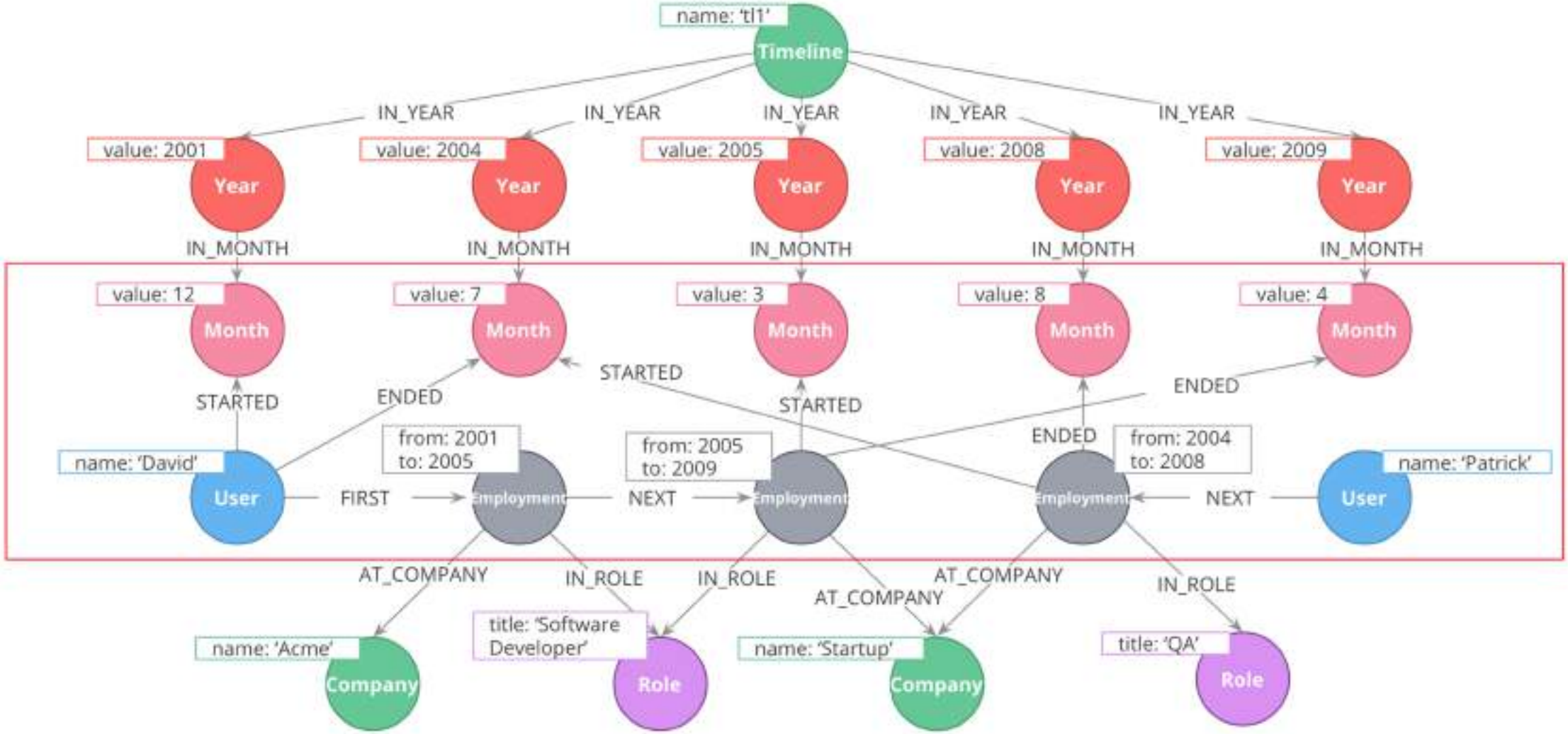
Using the Timeline Tree



Using Intermediate Nodes



Using Linked Lists



Exercise 4: Model a Learning Management System (LMS)

Given a description of the domain, sample data, and the application questions:

1. Create the model (entities and connections) using the Arrow tool.
2. Add sample data to the model using the Arrow tool and confirm questions can be answered using the model.

Exercise 4: The Domain

- There are many courses in the LMS, each of which contains a number of lessons that must be completed in a specific order.
- Every course grants a certificate upon completion.
- This certificate has a term of validity. When it expires, students must take the course again.
- Students can enroll in as many simultaneous courses as they want to.
- When a student logs in and chooses a course, the LMS must send them to their latest unfinished lesson.

Exercise 4: Sample Data

Courses	Lessons	Certificate
Introduction to Neo4j	Graph Theory, Graph Databases, Basic Cypher	2-year validity
Neo4j for Developers	Graph Theory, Property Graph, Graph Databases	6-month validity

Students	Completed Courses	In-Progress Courses
Alice	Introduction to Neo4j (2016), Introduction to Neo4j (2018)	Introduction to Neo4j (lesson 1)
Dan		Introduction to Neo4j (lesson 3), Neo4j for Developers (lesson 2)

Exercise 4: Application Questions

1. Which lesson(s) is Dan currently working on?
2. What are Alice's current certifications?
3. Which lessons are in the Neo4j for Developers course?
4. What is the last lesson in the Introduction to Neo4j course?
5. Which lesson follows Graph Theory in the Neo4j for Developers course?
6. Who has completed Introduction to Neo4j?

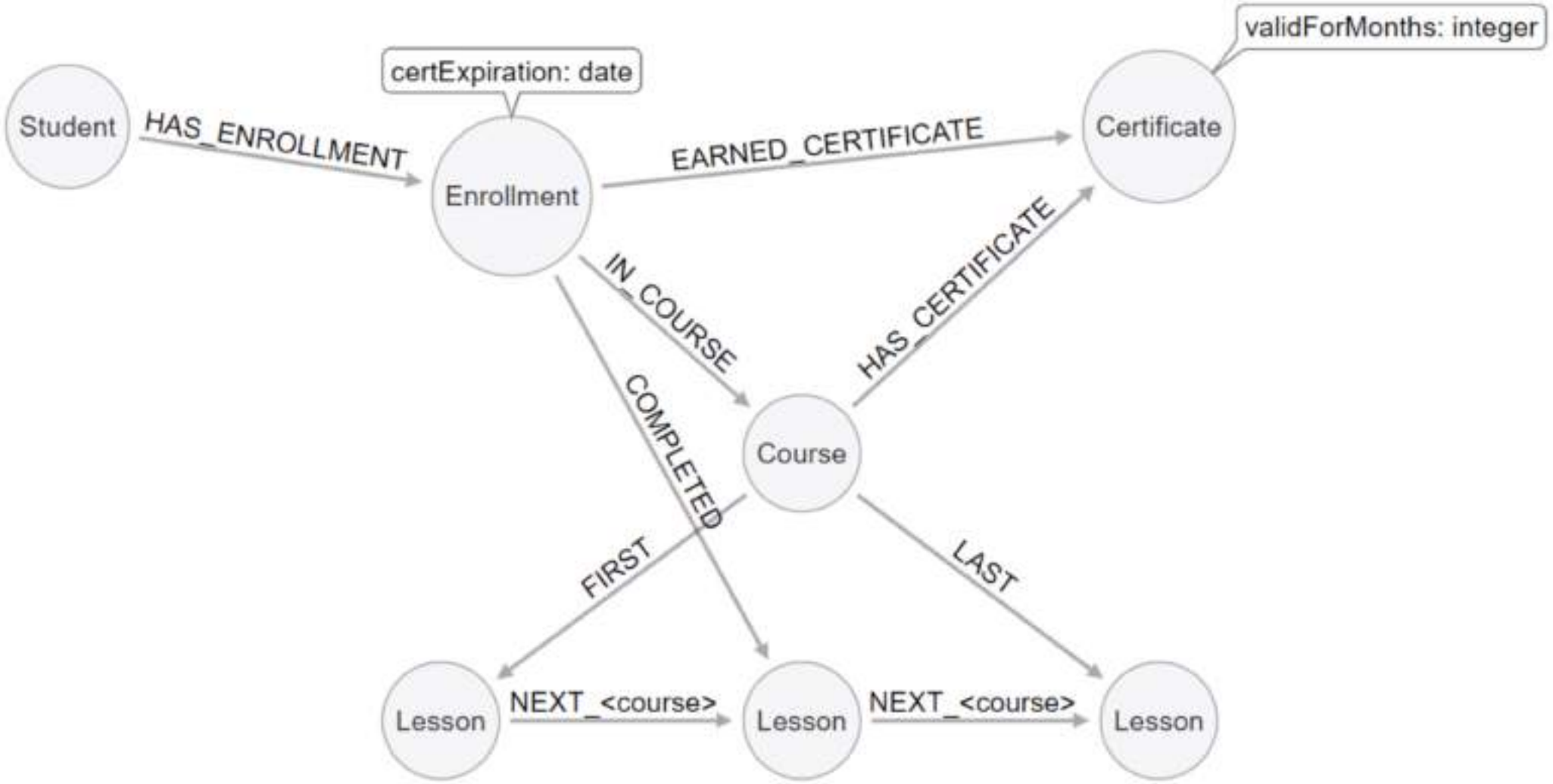
Exercise 6

- Which lesson(s) is Dan currently working on?
- What are Alice's current certifications?
- Which lessons are in the **Neo4j for Developers** course?
- What is the last lesson in the **Introduction to Neo4j** course?
- Which lesson follows **Graph Theory** in the **Neo4j for Developers** course?
- Who has completed **Introduction to Neo4j**?

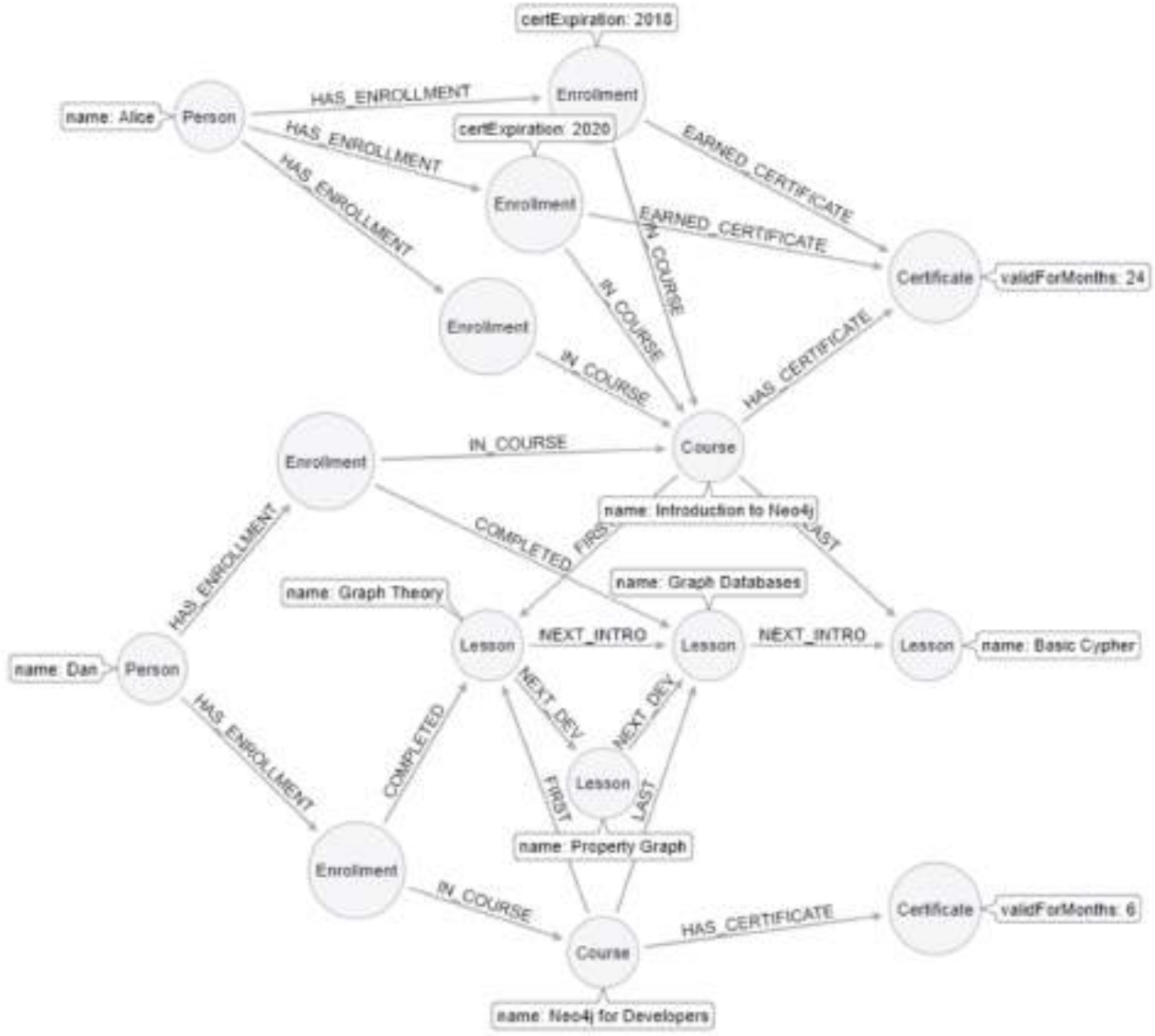
Courses	Lessons	Certificate
Introduction to Neo4j	Graph Theory, Graph Databases, Basic Cypher	2-year validity
Neo4j for Developers	Graph Theory, Property Graph, Graph Databases	6-month validity

Students	Completed Courses	In-Progress Courses
Alice	Introduction to Neo4j (2016), Introduction to Neo4j (2018)	Introduction to Neo4j (lesson 1)
Dan		Introduction to Neo4j (lesson 3), Neo4j for Developers (lesson 2)

Exercise 4 Solution: Application Model



Exercise 4 Solution: Sample Data



1. Which lesson(s) is Dan currently working on?
2. What are Alice's current certifications?
3. Which lessons are in the **Neo4j for Developers** course?
4. What is the last lesson in the **Introduction to Neo4j** course?
5. Which lesson follows **Graph Theory** in the **Neo4j for Developers** course?
6. Who has completed **Introduction to Neo4j**?

Summary

You should now be able to:

Describe common graph structures used in modeling:

- **Intermediate nodes**
- **Linked lists**
- **Timeline trees**
- **Multiple structures in a single graph**

Refactoring and Evolving a Model

About This Module

At the end of this module, you should be able to:

- Describe why you would refactor a graph data model
- Refactor a model to:
  - Eliminate duplicate data in nodes
  - Use node labels rather than properties
  - Extract property values to create nodes

What is Refactoring?

Refactoring is the process of …

- Changing the **data structure** ...
- **Without altering** its semantic meaning

Most of the time …

- Refactoring involves **moving data** from one structure to another

Sometimes refactoring involves …

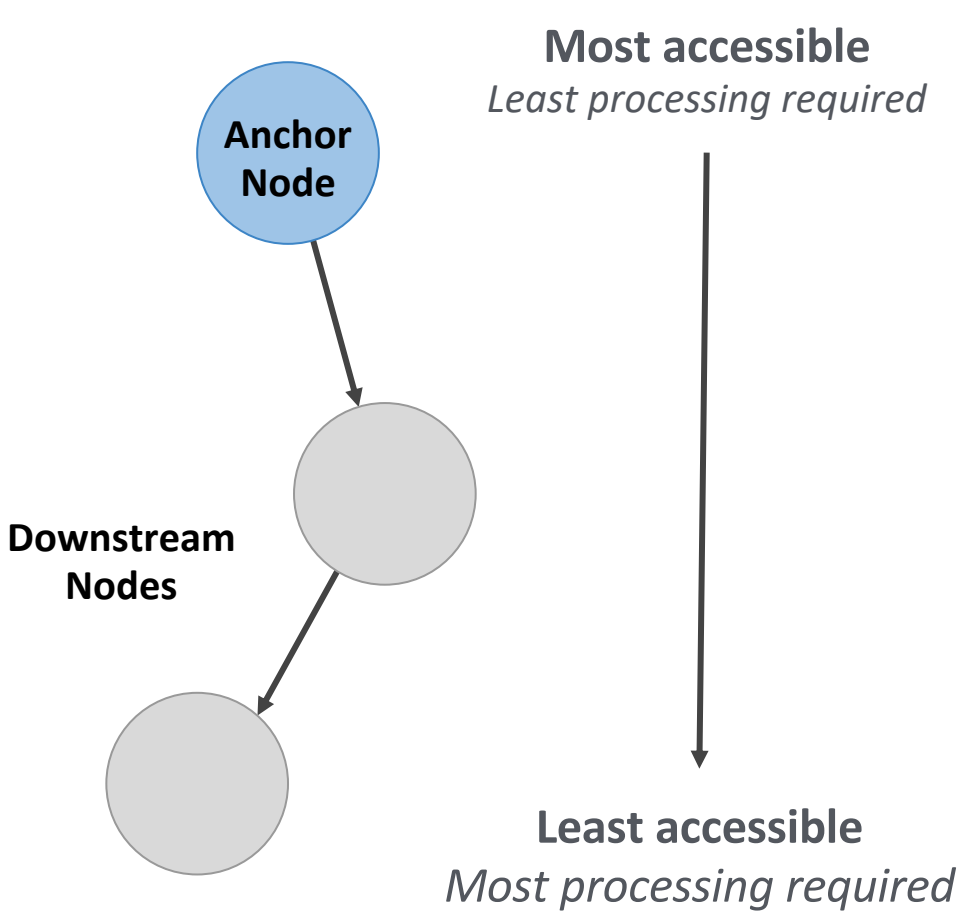
- **Adding additional data** from other sources

The most common type of refactoring is ...

- Restructure the graph to use a property value
- A property value is used to create a label, a node, or a relationship

Hierarchy of Accessibility

For each data object, how much work must Neo4j do to evaluate if this is a “good” path or a “bad” one?



1. Anchor node label
   Anchor node properties (indexed)
2. Relationship type
3. Anchor node properties (non-indexed)
4. Downstream node labels
5. Relationship properties
   Downstream node properties

Why Refactor?

Data models can be optimized
For **one** of **four** things:

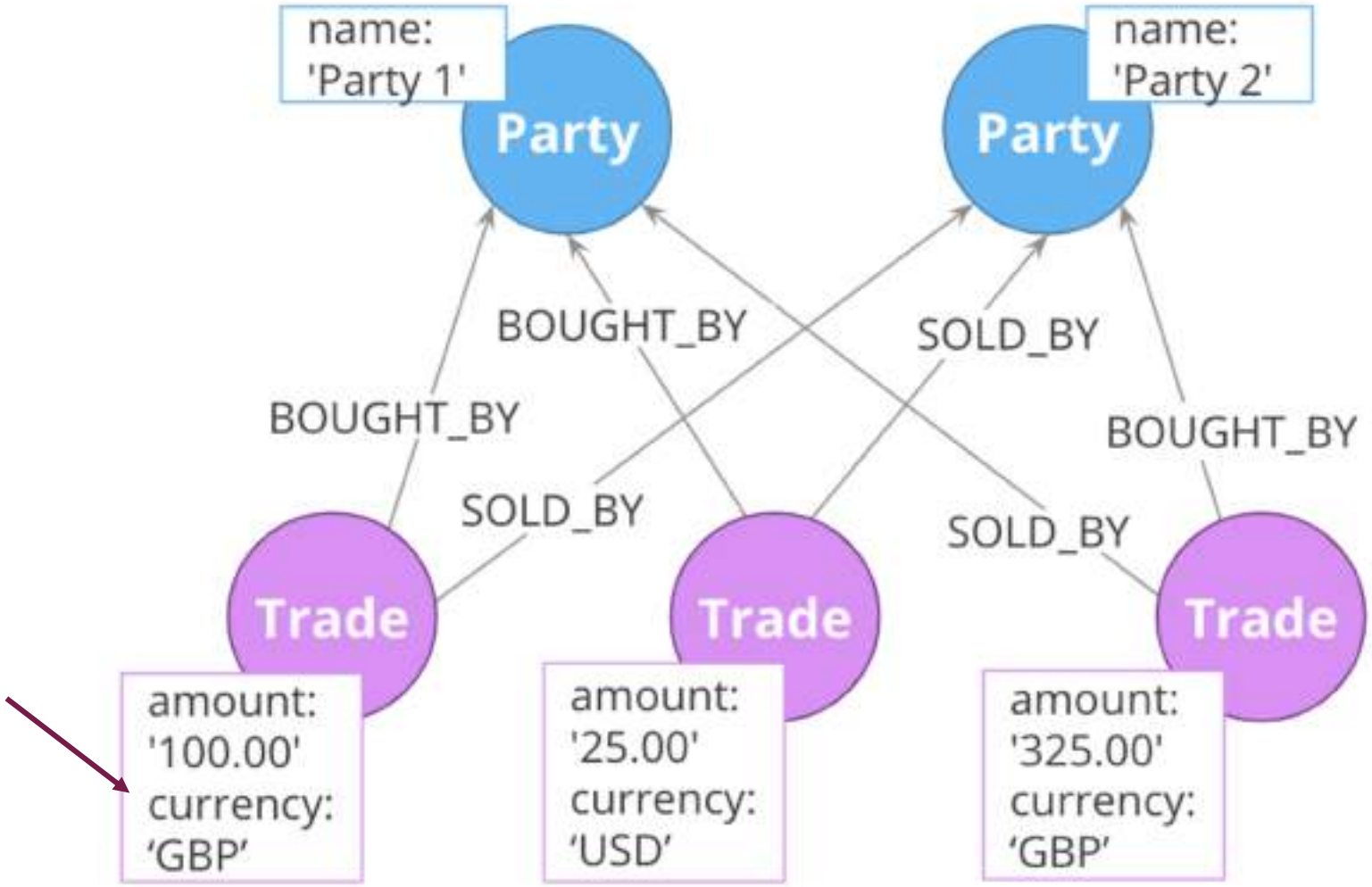
- Query performance
- Model simplicity & intuitiveness
- Query simplicity (i.e., simpler Cypher strings)
- Easier data updates

Note: Improving behavior in one of these areas frequently involves sacrifices in others

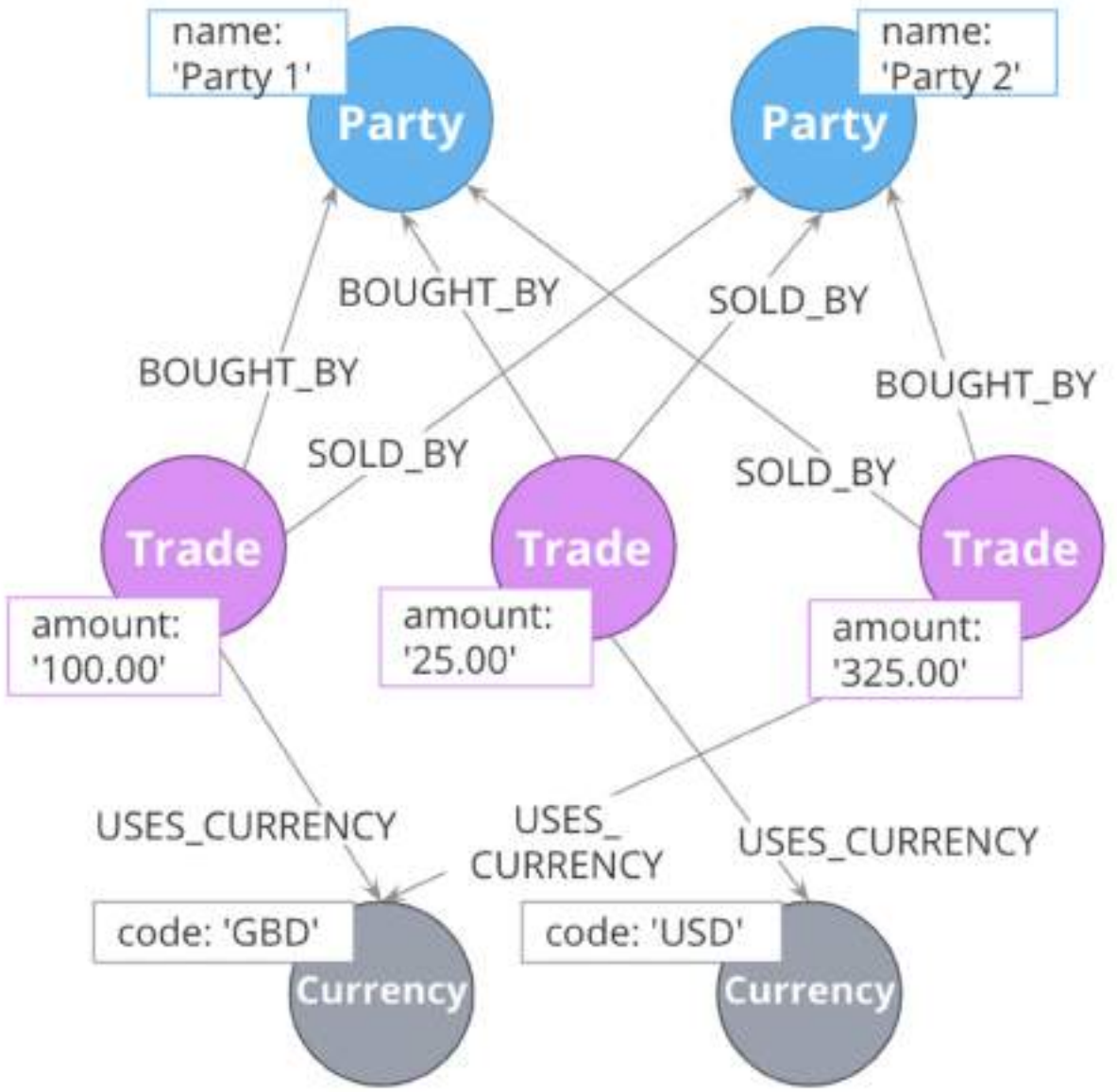
Another **important** reason to refactor is …

- to accommodate new application questions in the **same model**

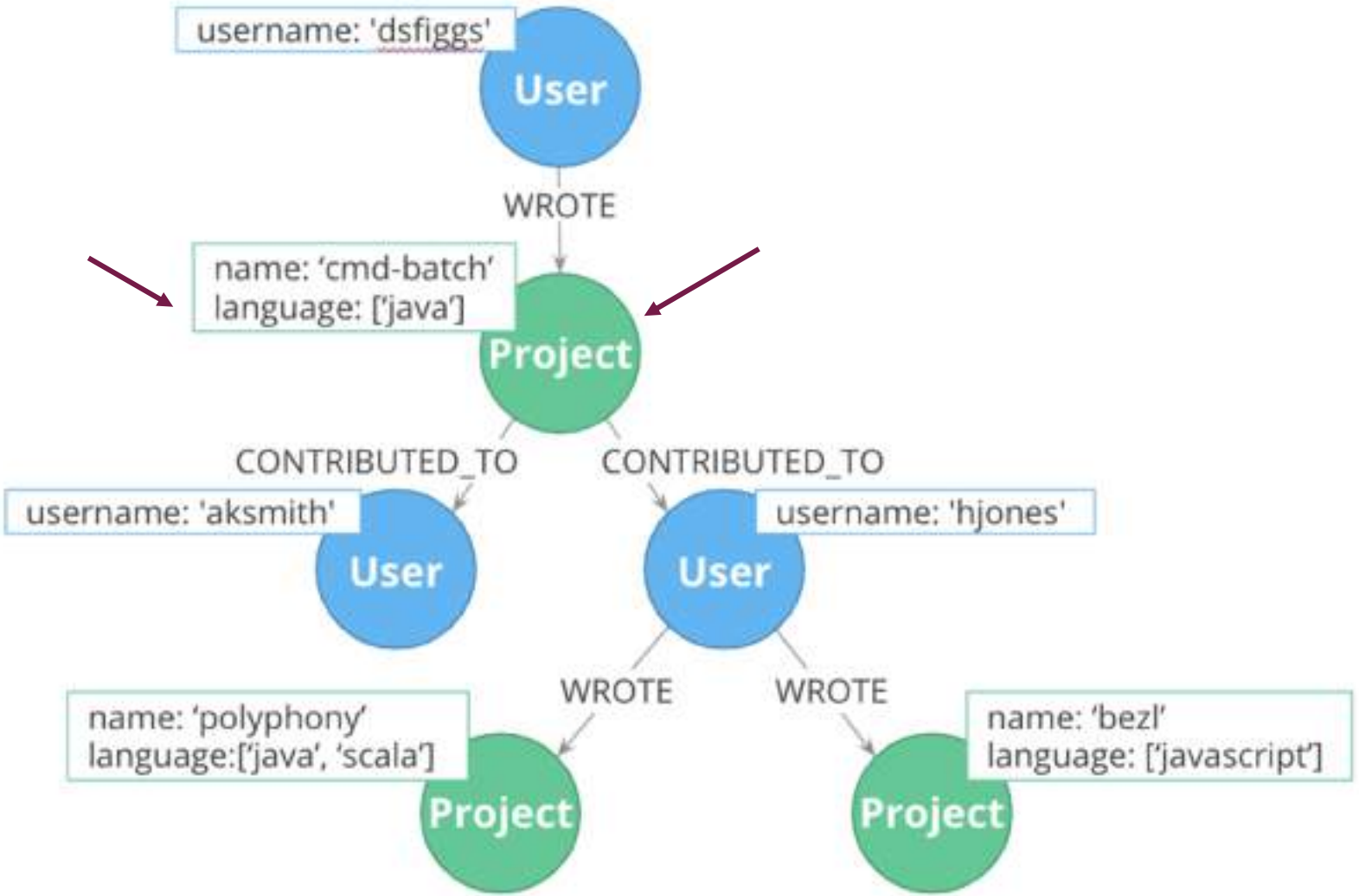
Goal: Eliminate Duplicate Data in Properties



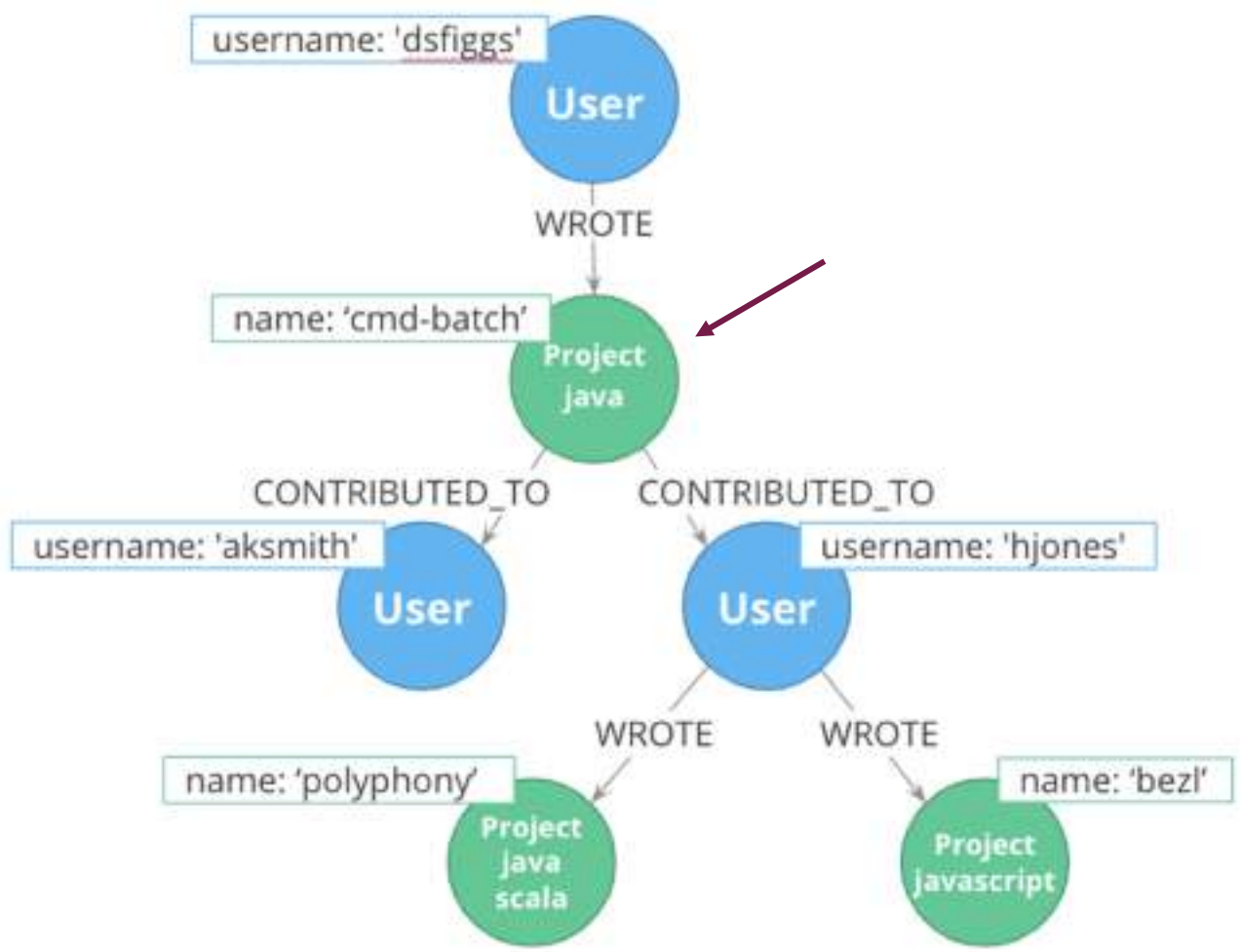
Refactor Example: Extracting Nodes From Properties



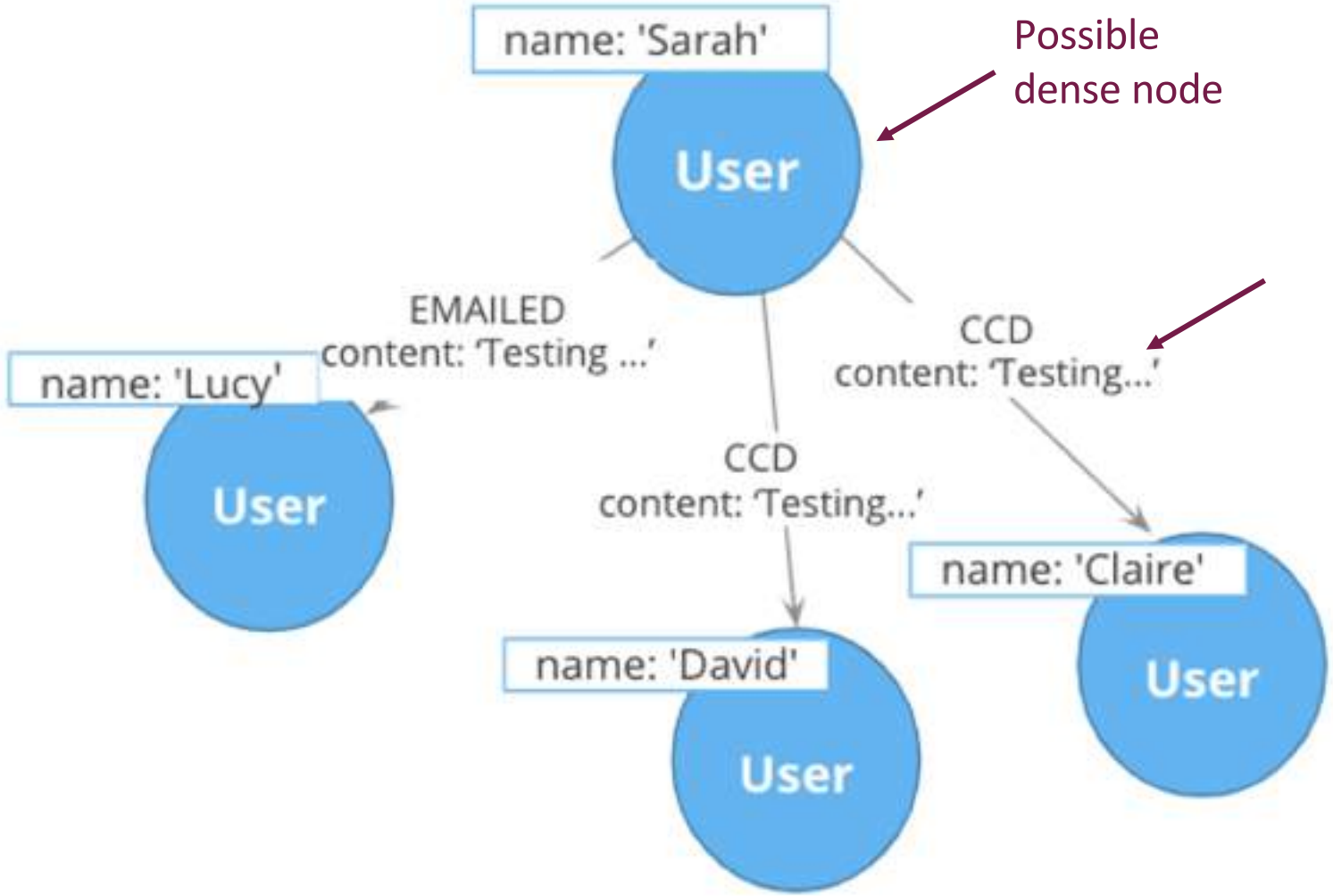
Goal: Use Labels Instead of Property Values



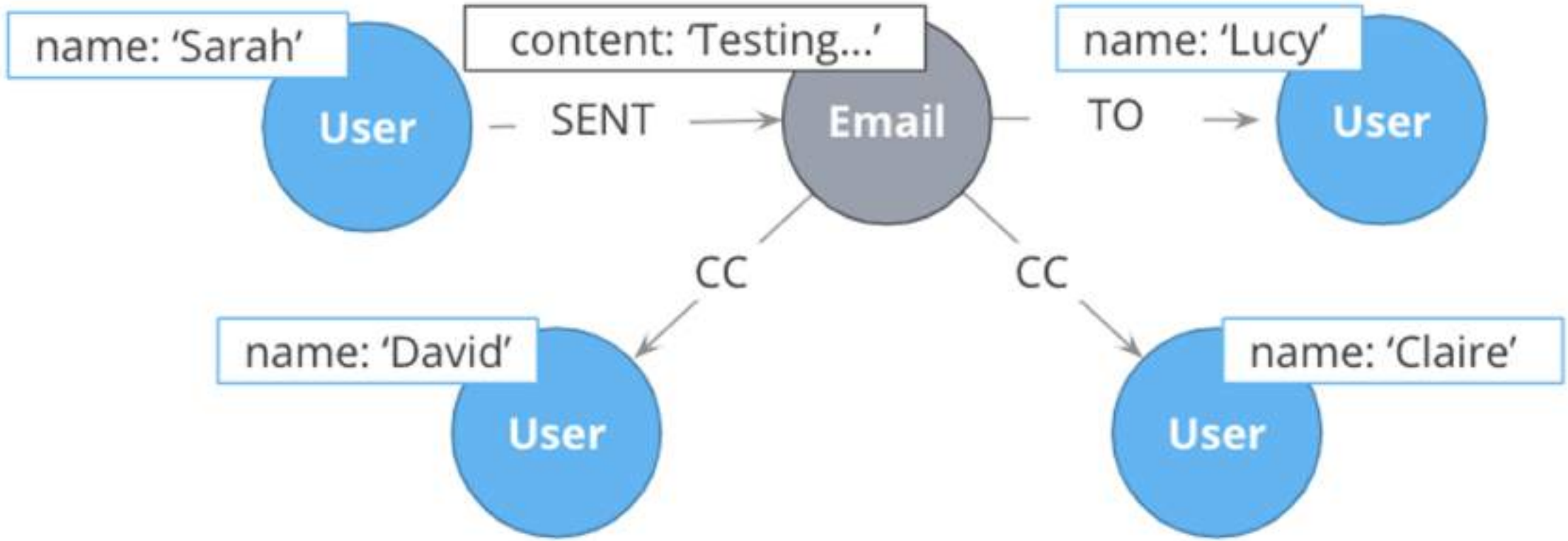
Refactor Example: Turn Property Values Into Labels for Nodes



Goal: Use Nodes Instead of Properties for relationships

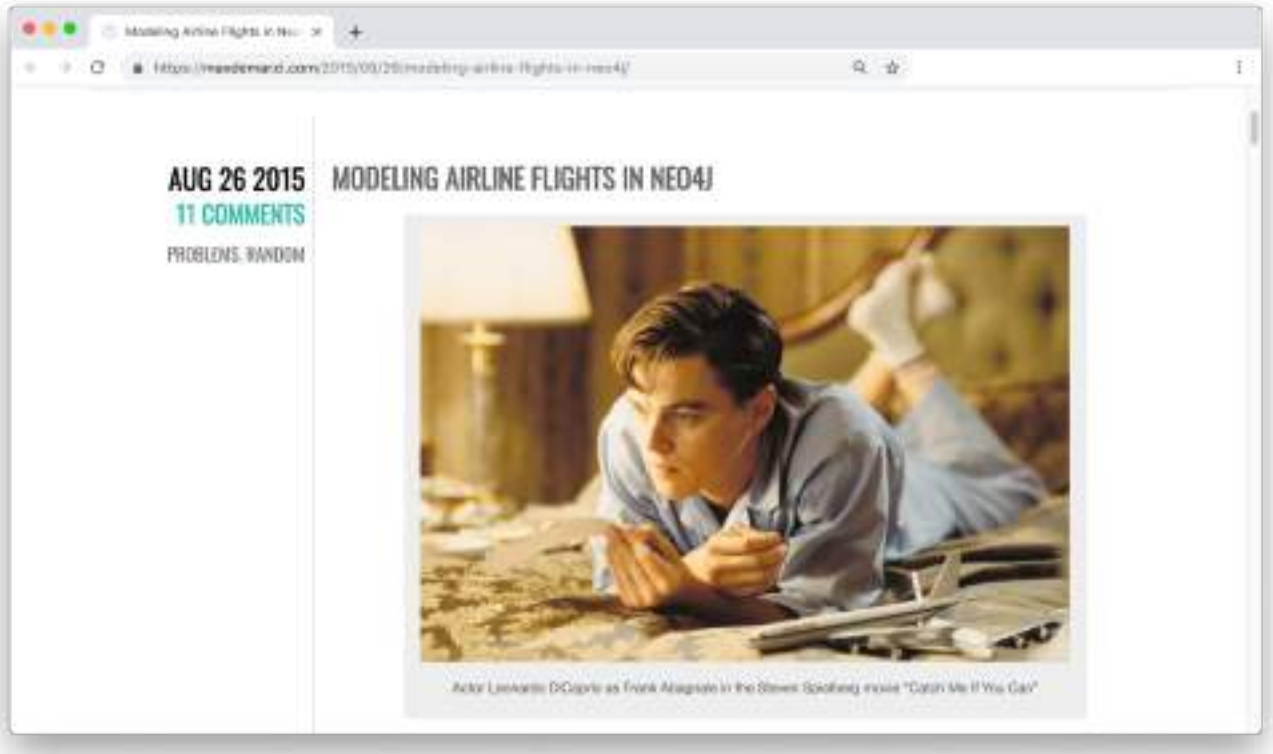


Refactor: Extract Nodes from Relationship Properties



Refactoring Example

Refactoring example:
Modeling airline flights



Credit: Max De Marzi https://maxdemarzi.com/2015/08/26/modeling-airline-flights-in-neo4j/

Initial Question for Our Model

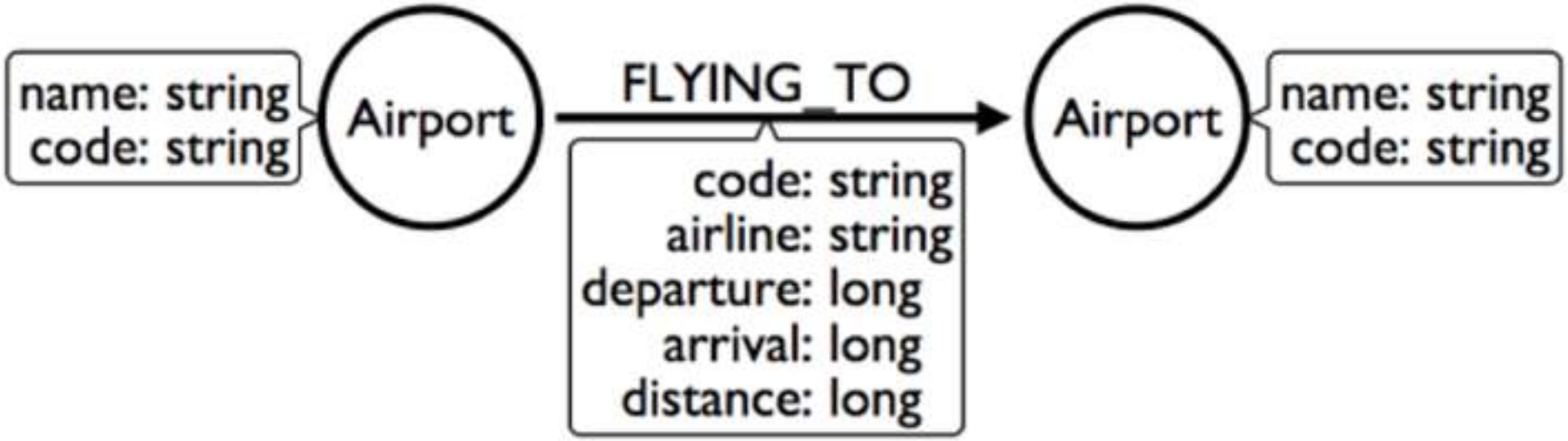
Question: What **flights** will take me from **Malmo** to **New York** on **Friday**?

Ask yourself:

- What are the entities?
- What are the connections between the entities?
- What properties do we need?

Initial Model

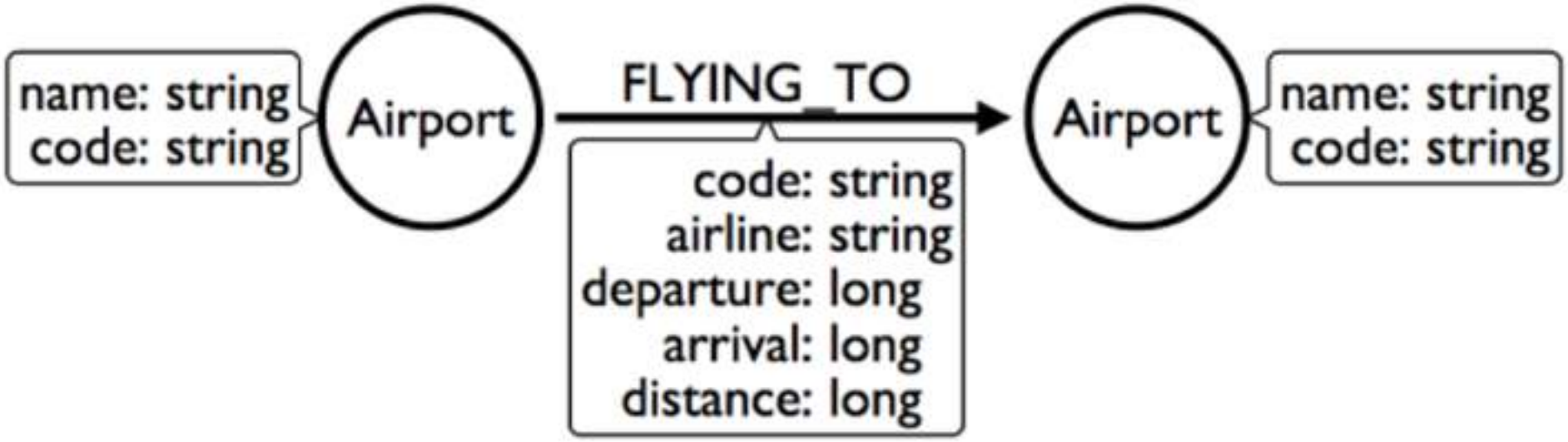
Question: What **flights** will take me from **Malmo** to **New York** on **Friday**?



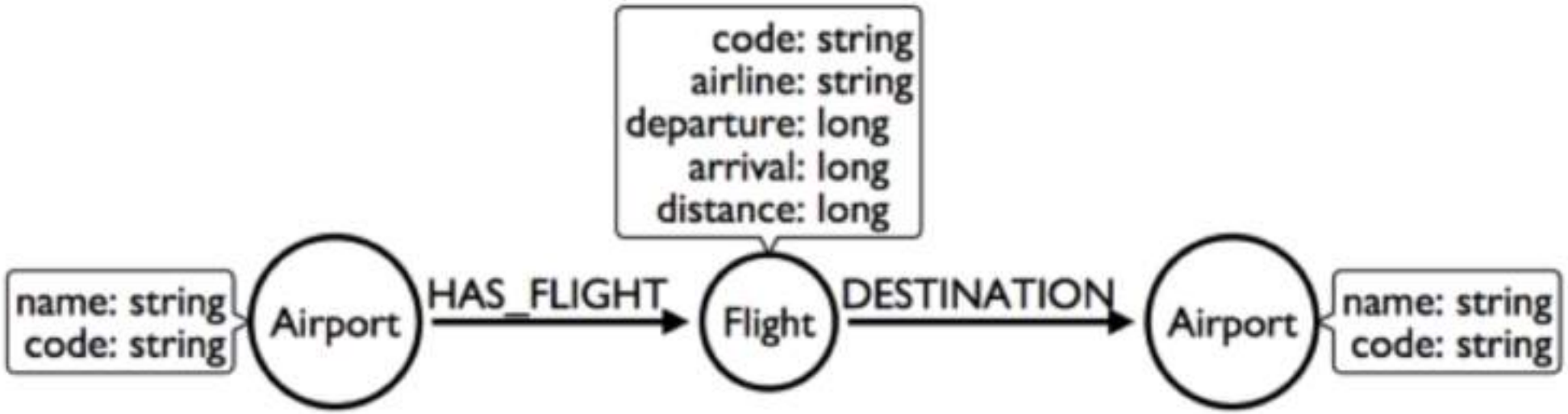
Initial Model

Question: What **flights** will take me from **Malmo** to **New York** on **Friday**?

New Question: Mom is on **flight AY189**. When will she land?



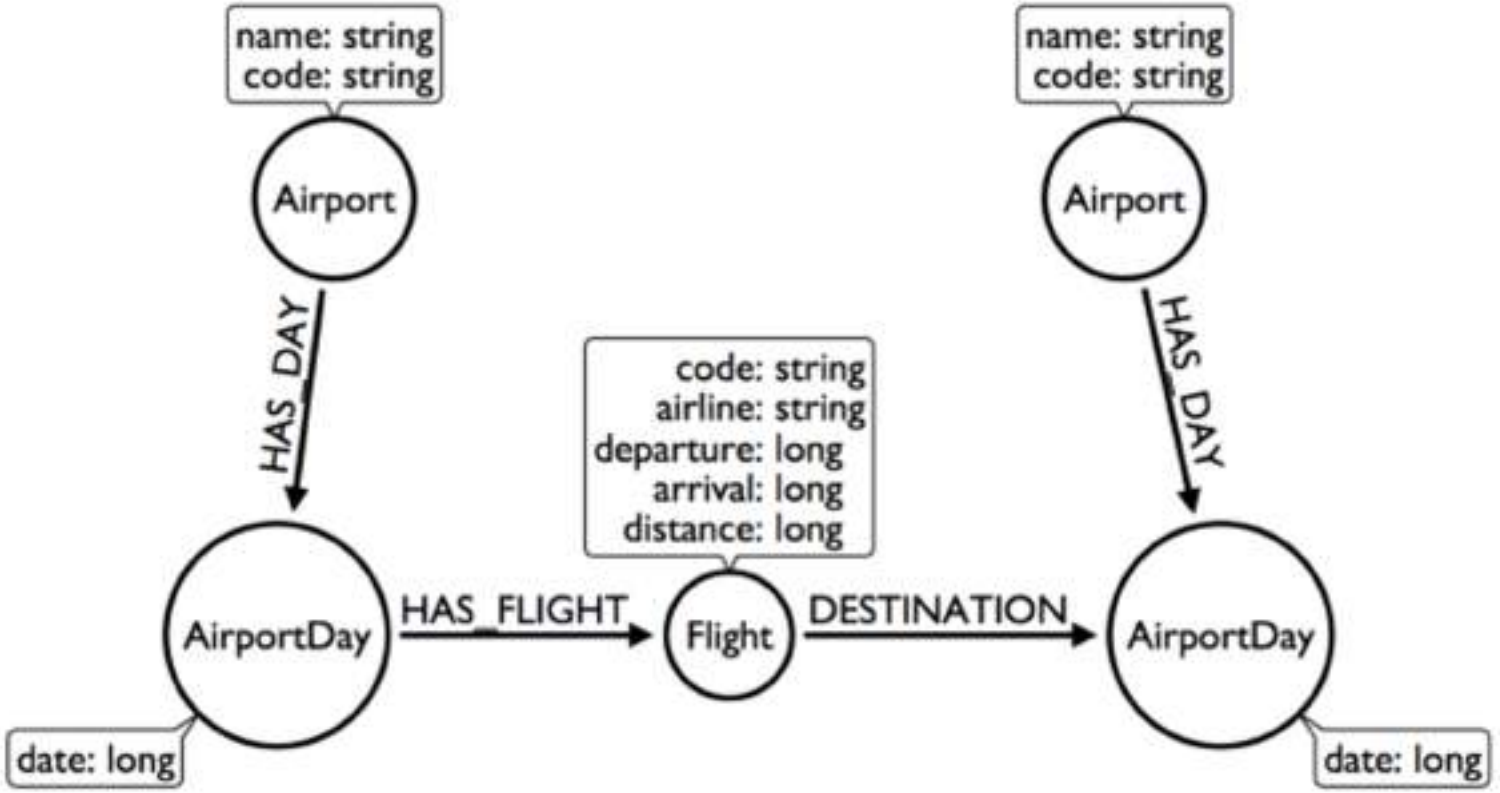
Refactor: Create Intermediate Flight Nodes



Question 1: What flights will take me from Malmo to New York on Friday?

Question 2: Mom is on flight AY189. When will she land?

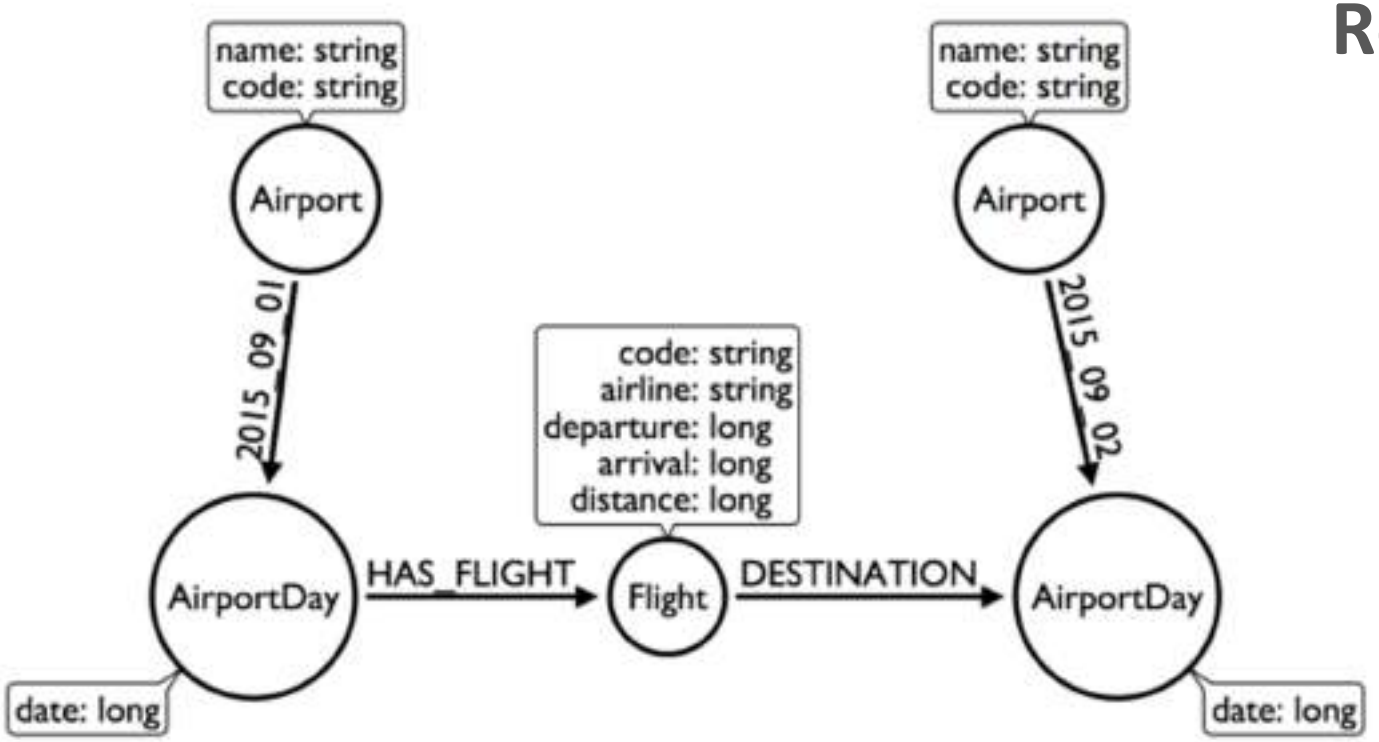
Refactor: Create AirportDay Intermediate Nodes



Question 1: What flights will take me from Malmo to New York on Friday?

Question 2: Mom is on flight AY189. When will she land?

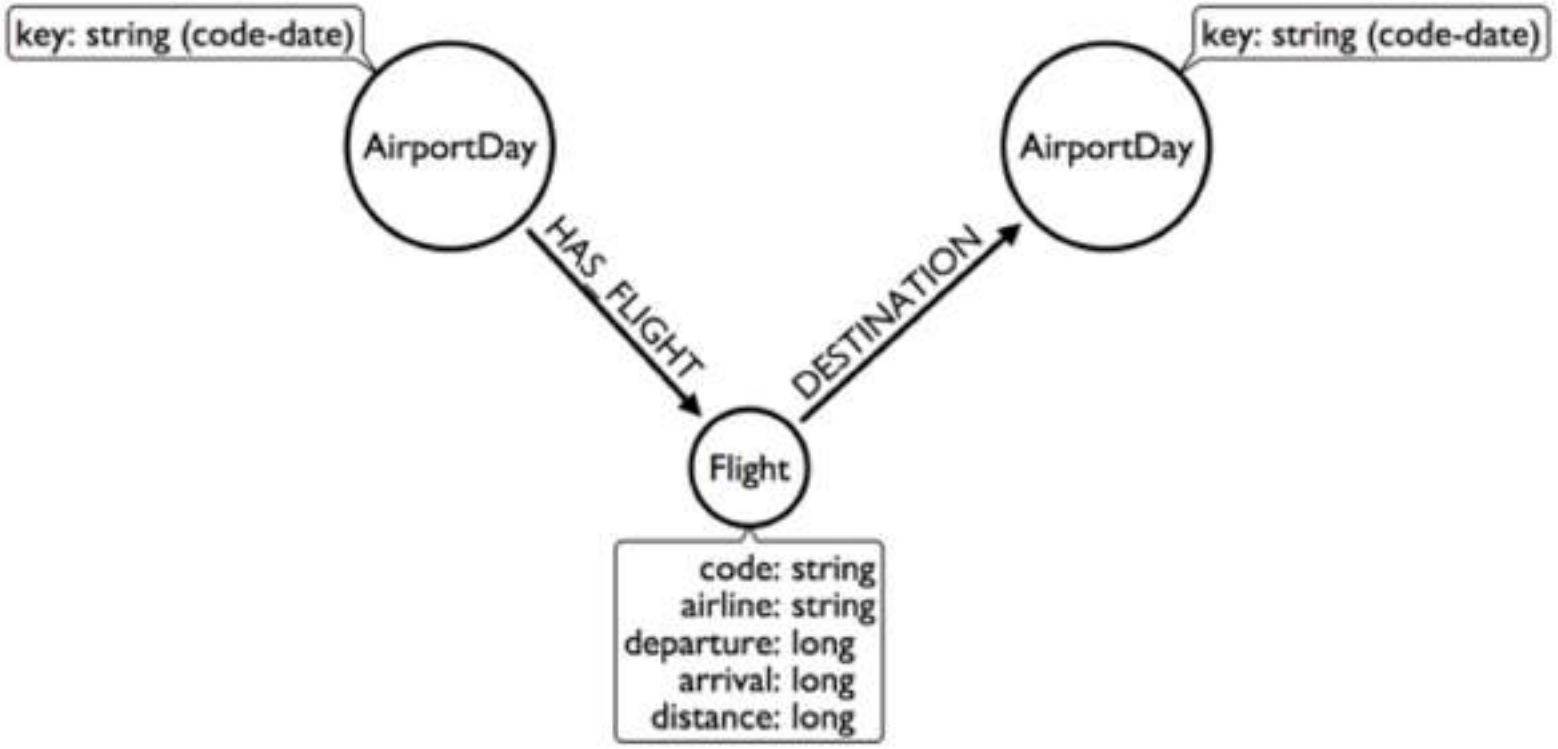
Possible Refactor: Change Relationship Type to Date



Question 1: What flights will take me from Malmo to New York on Friday?

Question 2: Mom is on flight AY189. When will she land?

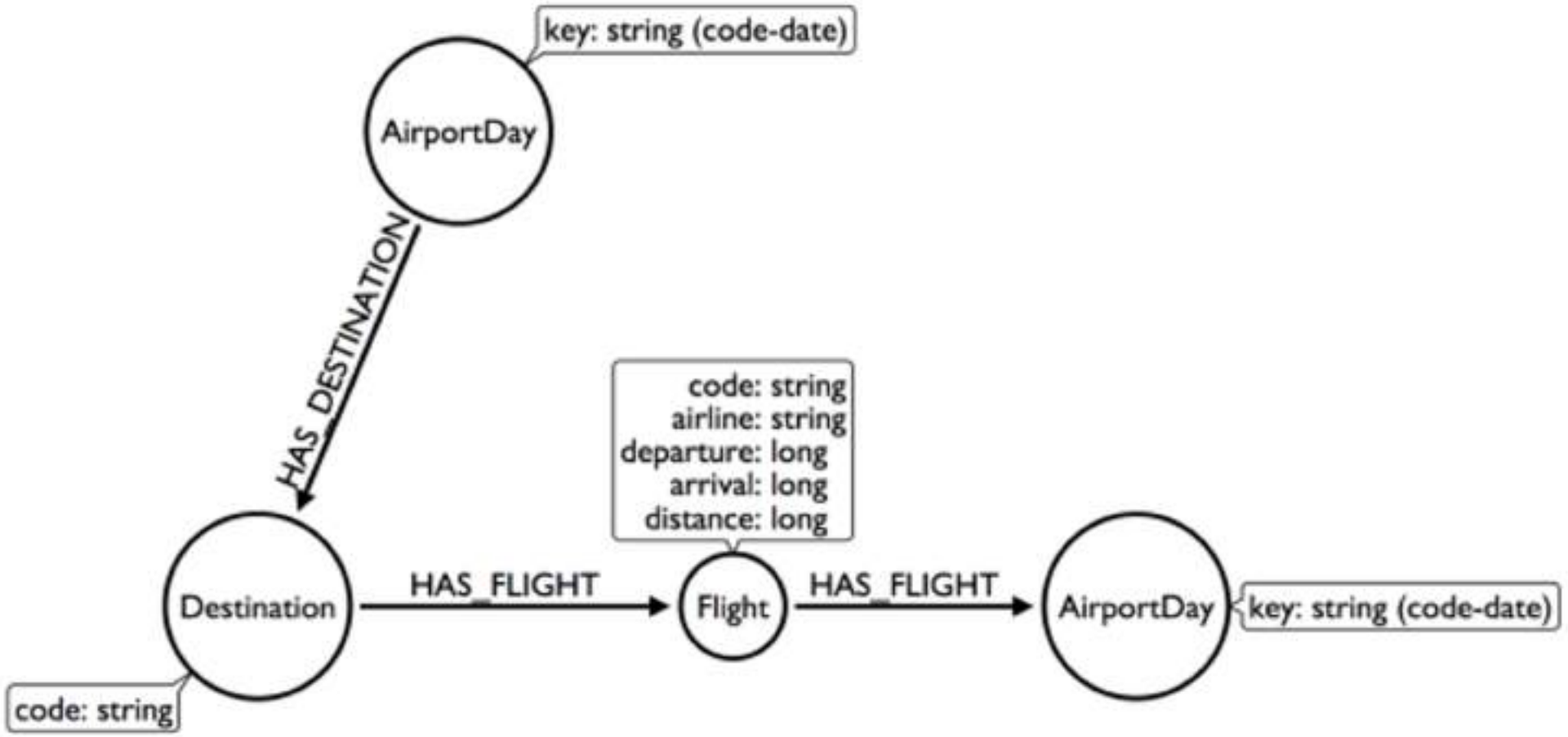
Possible Refactor: Remove Airport Nodes



Question 1: What flights will take me from Malmo to New York on Friday?

Question 2: Mom is on flight AY189. When will she land?

Refactor: Add Destination Intermediate Nodes



Question 1: What flights will take me from Malmo to New York on Friday?

Question 2: Mom is on flight AY189. When will she land?

Summary

You should now be able to:

- **Describe why you would refactor a graph data model**
- **Refactor a model to:**
  - **Eliminate duplicate data in nodes**
  - **Use node labels rather than properties**
  - **Extract property values to create nodes**