Graph-based!

Recommenders 101

Caroline Harbitz

This workshop is…

- A basic overview of traditional recommendation algorithms
- A hands-on introduction to graph-powered recommendations

This workshop is not about...

- In-depth analyses of recommendation algorithms
- Graph theory
- Implementing a graph database application
- Bashing RDBMS and aggregate stores
- nordstrom.com's recommendations

Before we get started: some useful links

- Clone (or download) this workshop's github repository

  https://github.com/cterp/rec101-galvanize-workshop.git

- Neo4j documentation

  http://neo4j.com/docs/stable/

- Intro to Cypher

  http://neo4j.com/developer/cypher-query-language/

- Neo4j ref card

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

Part 2: Neo4j graphs

Workshop outline

Part 1: Recommendation algorithms

Part 2: Graph database basics and queries

Part 3: Build your own simple recommender

Customers also considered

Tide HE Turbo Powder Laundry Detergent
★★★☆☆ 33
$12.57

$12.57

HEC Simply Clean & Fresh...
★★★★☆ 95

Walmart

amazon.com

...ently Viewed Items and Featured Recommendations

...ng history

Holiday Wonderland 300-Count Clear Christmas Mini Light Set
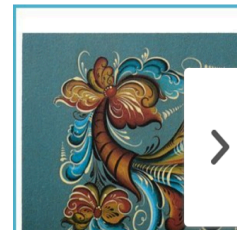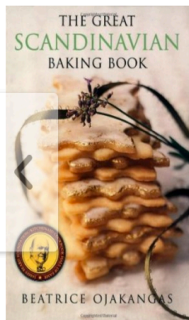★★★★☆ 293
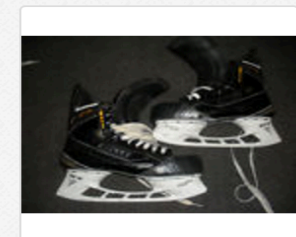$21.99 ✓Prime

Res...
28.5-Inche...
★★★★☆
$22.45 ✓Prime

Kurt Adler 10 Light...

...pular on eBay

ebay™

Free shipping

You might also like...

Recommendations for You in Books

THE GREAT SCANDINAVIAN BAKING BOOK
BEATRICE OJAKANGAS

AUNT HILDUR'S EXCELLENT NORWEGIAN RECIPES

Scandinavian CHRISTMAS

SCANDINAVIAN classic BAKING
PAT SINCLAIR

Bestsellers

○ ●

Madden 'Troopa' Boot
Was: $99.95
Now: $79.90 20% OFF
★★★★½ (1864)

Converse Chuck Taylor® 'Shoreline' Sneaker (Women) (Regular Retail Price: $49.95)
Add to cart to see price
★★★★½ (258)

Dolce Vita 'Garim' Wedge Bootie (Women) (Nordstrom Exclusive)
Was: $109.95
Now: $79.90 25% OFF
★★★★½ (7)

(Women)
Was: $169.95
Now: $149.90 10% OFF
★★★★½ (1598)

Bootie (Women) (Nordstrom Exclusive)
Was: $149.95
Now: $99.90 33% OFF
★★★★½ (289)

NORDSTROM

Used Bauer Supreme TotalOne MX3 Pro...
$79.99
1 bid

Part 1: Recommenders

Part 1: Recommenders

Part 1: Recommenders
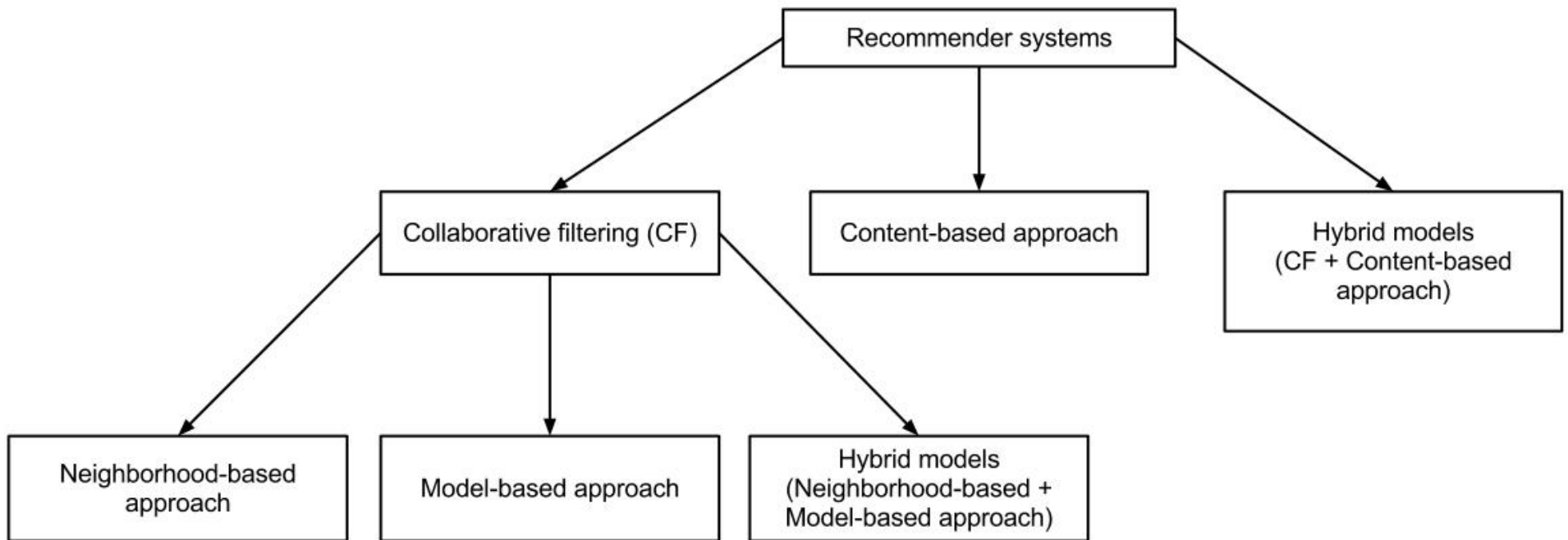
Recommendation process

Purpose: personalize each customer's online store.

1. Find potential recommendations
2. Narrow down potential recommendations
3. Hide irrelevant recommendations
4. Measure the quality of the recommendations

Part 1: Recommenders

Collaborative filtering

Predict a person's affinity for something by connecting that person to other people with similar tastes.

Part 1: Recommenders

Collaborative filtering: user-user

- Compute "distance" between all users
- Neighborhoods
- Pitfalls:
  - Neighborhood quality
  - Online calculations

Collaborative filtering: item-item

- Example: amazon.com

- Find items that customers tend to purchase together

- Calculate the similarity between a single product and all related products

Collaborative filtering: item-item

For each item in product catalog I_1

   For each customer C who purchased I_1

      For each item I_2 purchased by customer C

         Record that a customer purchased I_1 and I_2

  For each item I_2

    Compute the similarity between I_1 and I_2

Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." Internet Computing, IEEE 7.1 (2003): 76-80.

Collaborative filtering: Limitations

- Rigid

- Scaling

- Recommendation quality

- Data sparsity

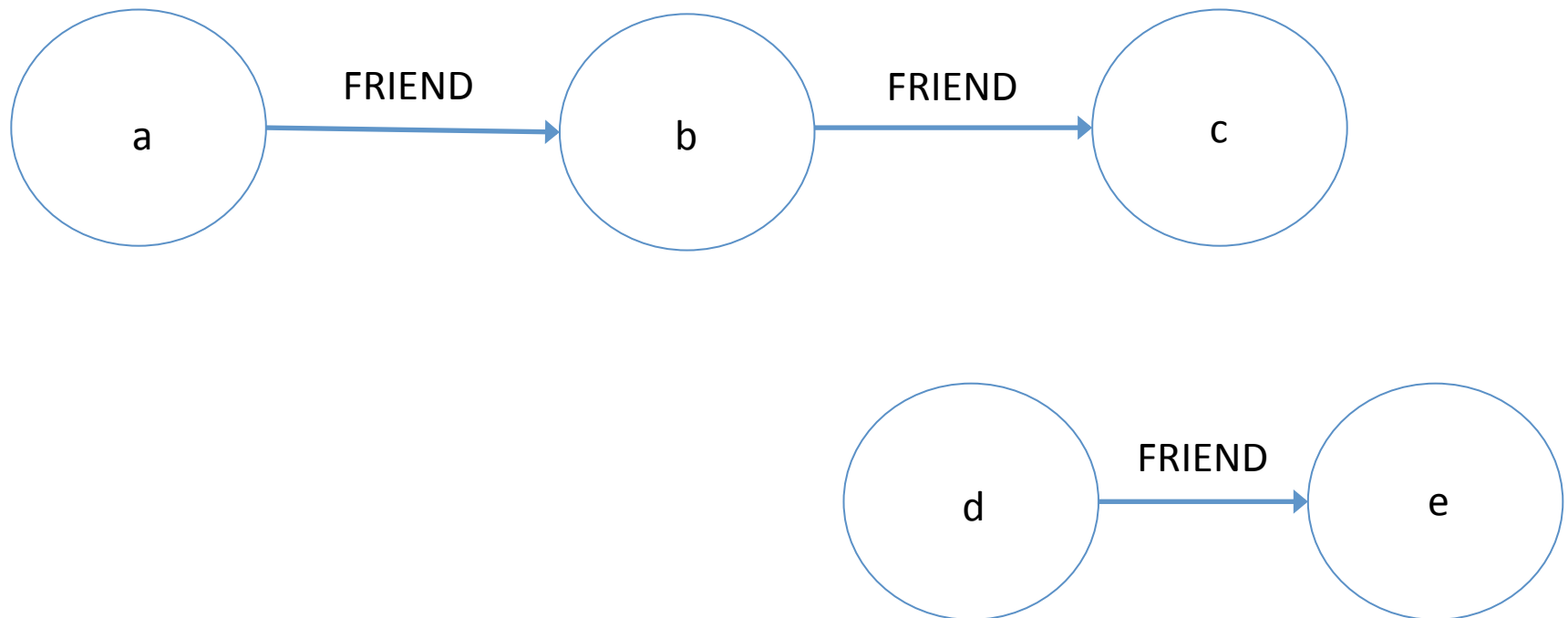# Part 2: Neo4j graphs

- A graph: a collection of *nodes* and *edges*
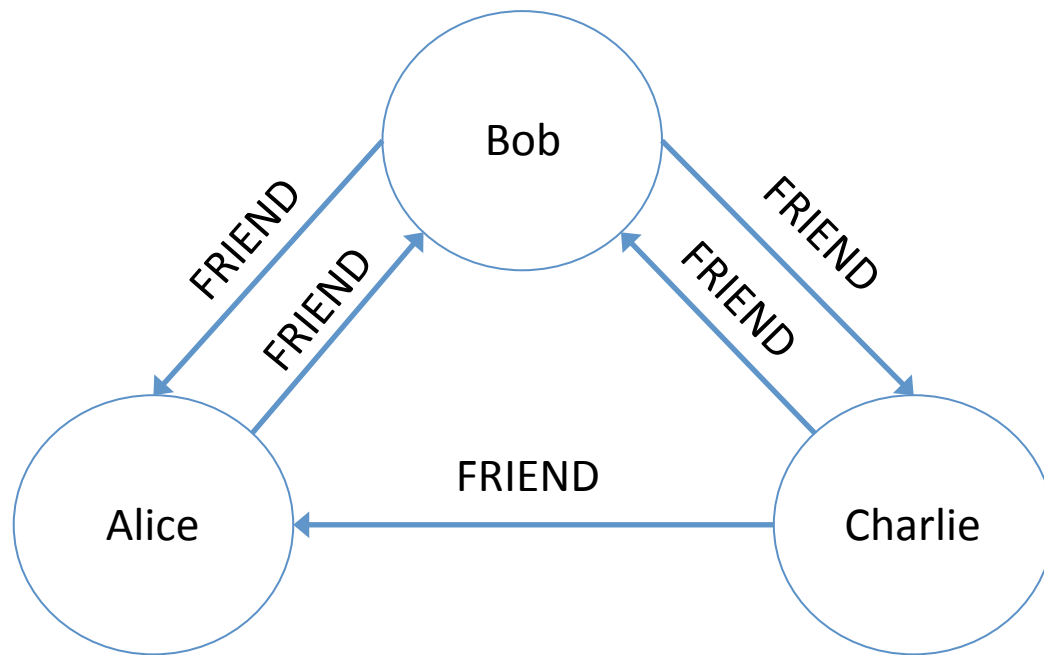
A simple property graph.

4 building blocks of a Neo4j graph

1. <u>Nodes</u>: entities

2. <u>Relationships</u>: connect nodes

3. <u>Properties</u>: entity attributes, relationship qualities, and metadata

4. <u>Labels</u>: group nodes by role

Part 2: Neo4j graphs

What is a traversal?

Ok, so what?

- Graphs let you forget about:
  - Primary and foreign key constraints
  - Expensive joins
  - Expensive reciprocal queries
  - Sparse tables with nullable columns

- Graphs are all about connectedness:
  - Store direct references to neighboring nodes
  - Relationships are first-class citizens of graph data model
  - Connect your data as the domain dictates rather than a schema imposed upfront

Part 2: Neo4j graphs

Limitations of this approach

- Your graph is only as good as your data <u>model</u>
- Neo4j's implementation optimizes traversals (OLTP) at the expense of non-traversal queries (OLAP)
- Not a standard industry tool

Why neo4j?

- Large user community

- Easy to use

- Documentation

Part 2: Neo4j graphs

Cypher

- Neo4j's graph query language
  - DSL to query a Neo4j graph database
- In a nutshell: describe paths in the graph by using ASCII art

Cypher examples

- Nodes: ()

- Relationships: []

- Directions: -> and <-

- Graph pattern examples:

  ()-[]-()
  ()-[]->()
  ()<-[]-()

ASCII art describes the graph pattern of a Cypher query.

Frequently used statements

MATCH

- Draws the query pattern
- MATCH (node1:Label1)-[rel:REL_TYPE]->(node2:Label2)

RETURN

- Equivalent to SQL SELECT

WHERE

- Filter your pattern matching results

WITH

- Chain query parts; similar to Unix pipe. RETURN.

Start your Neo4j servers

From your Neo4j directory, run:

```
$ ./bin/neo4j start
```
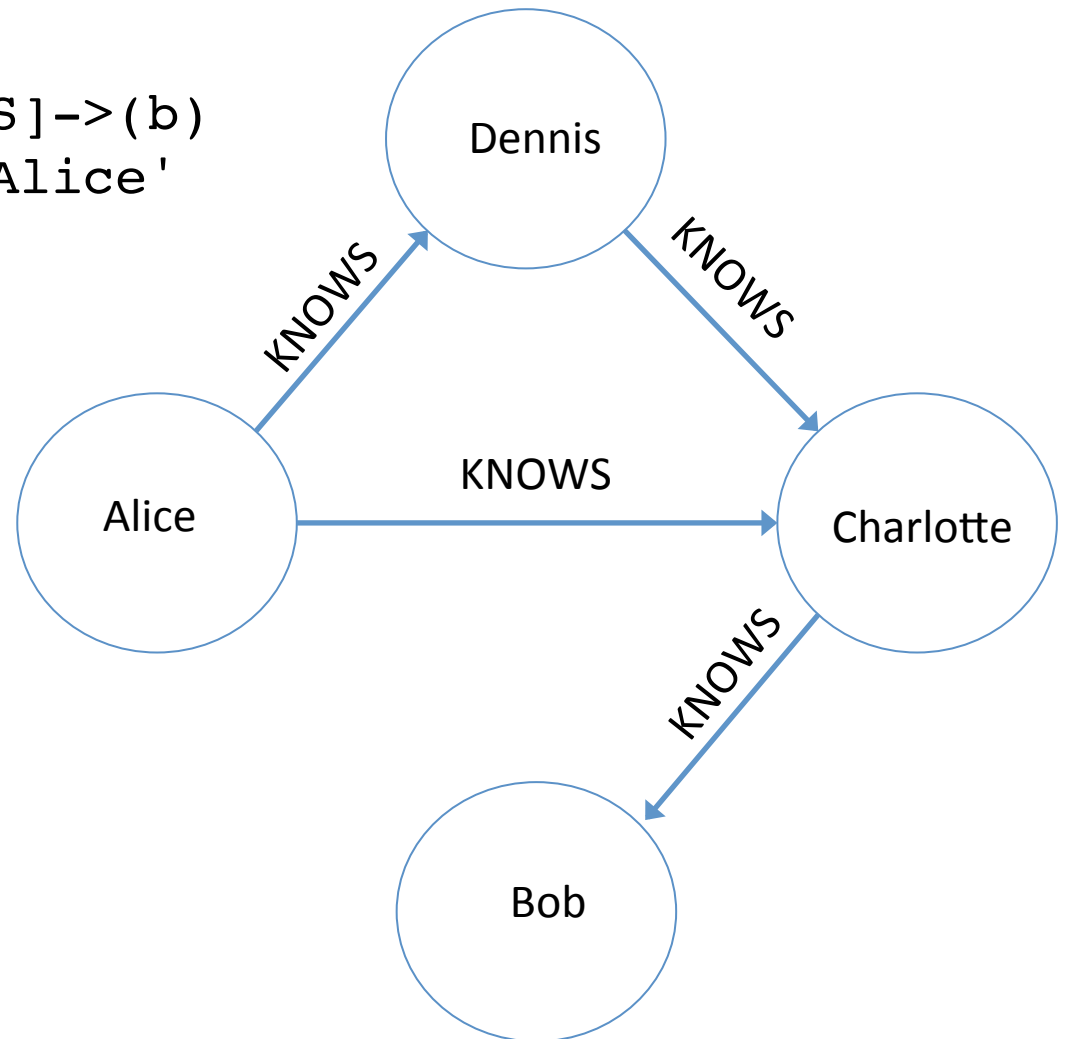
Then, in your browser go to:

http://localhost:7474/browser/

Run 'part_2.cypher' in the browser

Part 2: Neo4j graphs

Example 1: find Alice's friends

```
MATCH (a:Person)-[:KNOWS]->(b)
WHERE a.person_name = 'Alice'
RETURN b.person_name;
```

What happens when you return just 'b'?

Equivalent queries

```
MATCH (a:Person)-[:KNOWS]->(b)
WHERE a.person_name = 'Alice'
RETURN b.person_name;


MATCH (a:Person)-[:KNOWS]->(b:Person)
WHERE a.person_name = 'Alice'
RETURN b.person_name;


MATCH (a)-[:KNOWS]->(b)
WHERE a.person_name = 'Alice'
RETURN b.person_name;


MATCH (a:Person {person_name: 'Alice'})-[:KNOWS]
    ->(b:Person)
RETURN b.person_name;
```
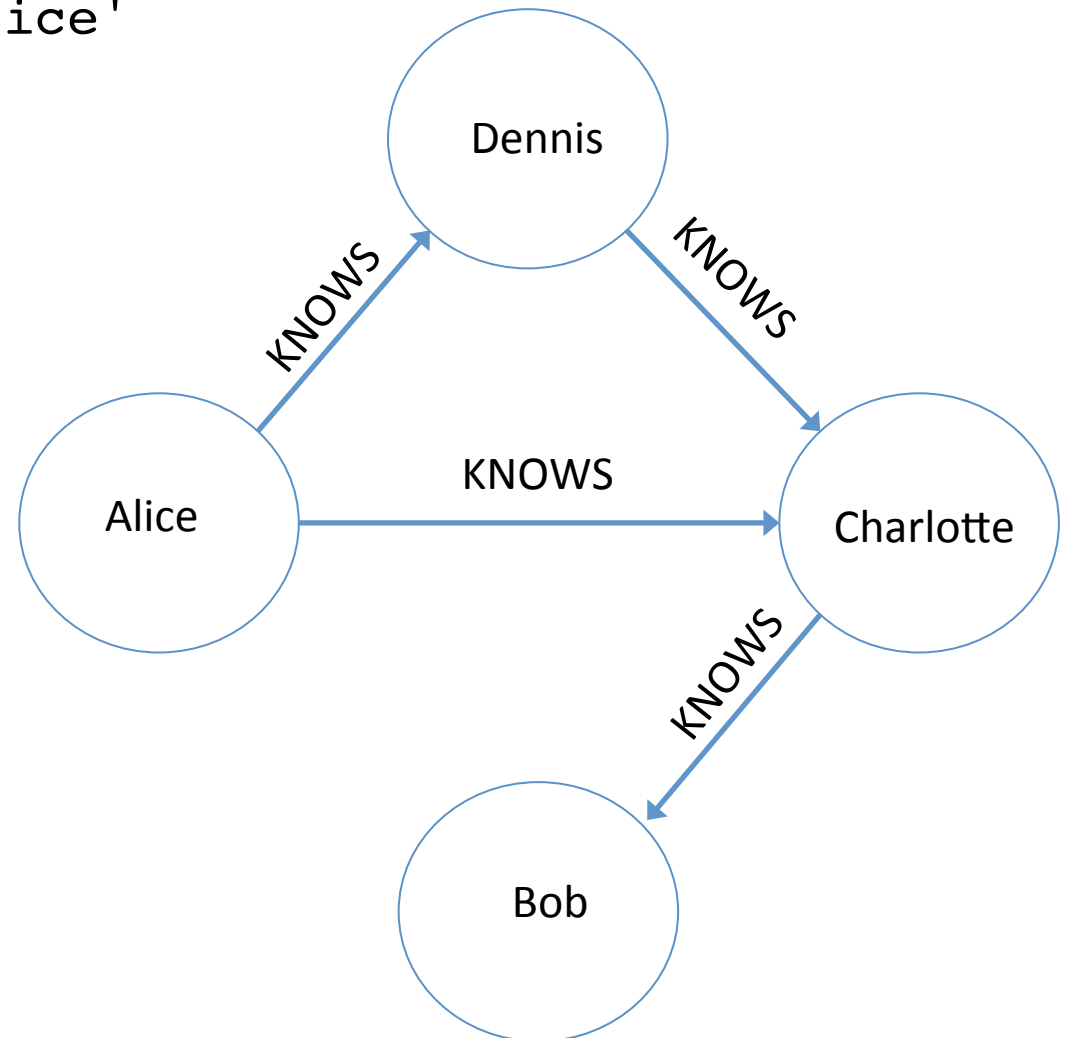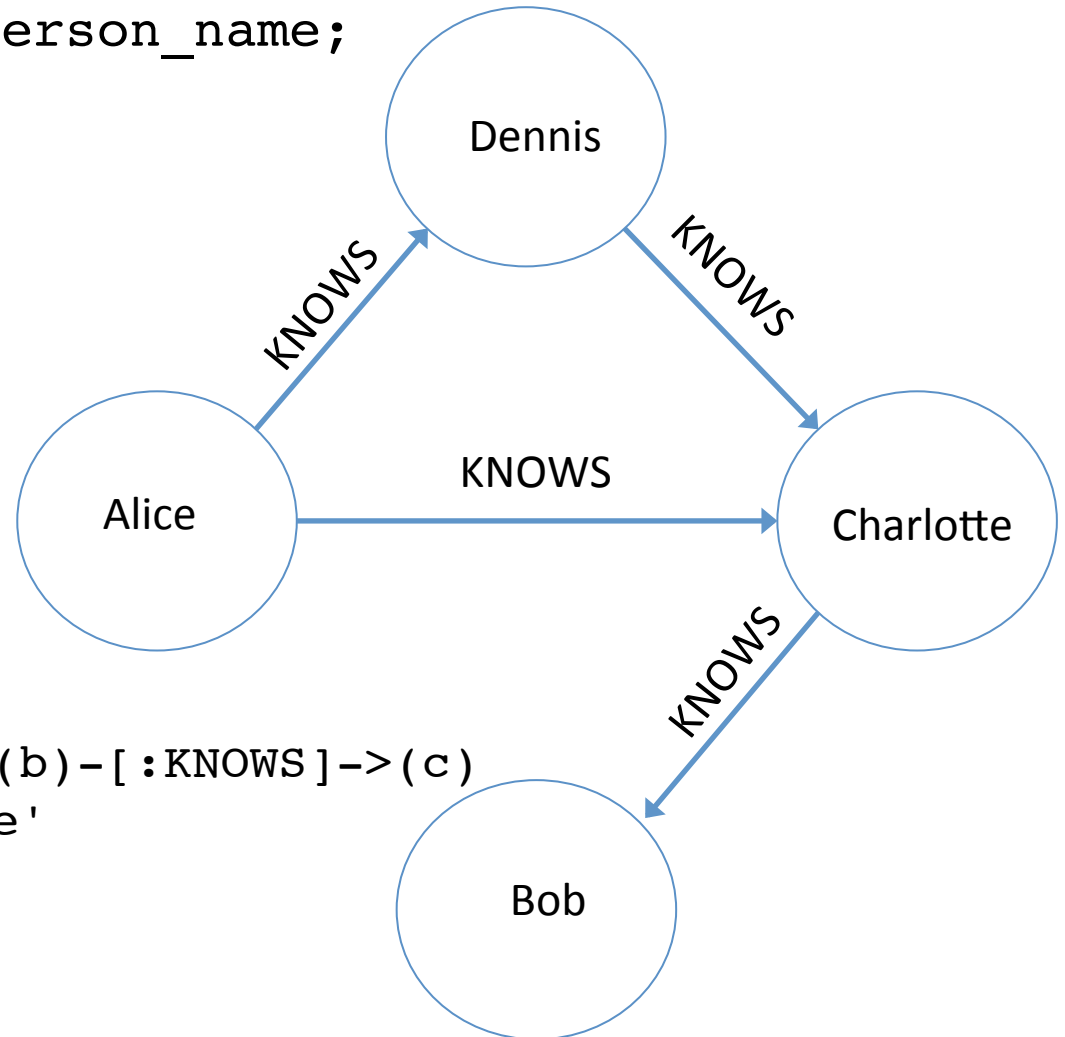
Example 2: find Alice's mutual friend

```
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c),
(a)-[:KNOWS]->(c)
WHERE a.person_name = 'Alice'
RETURN b.person_name;
```
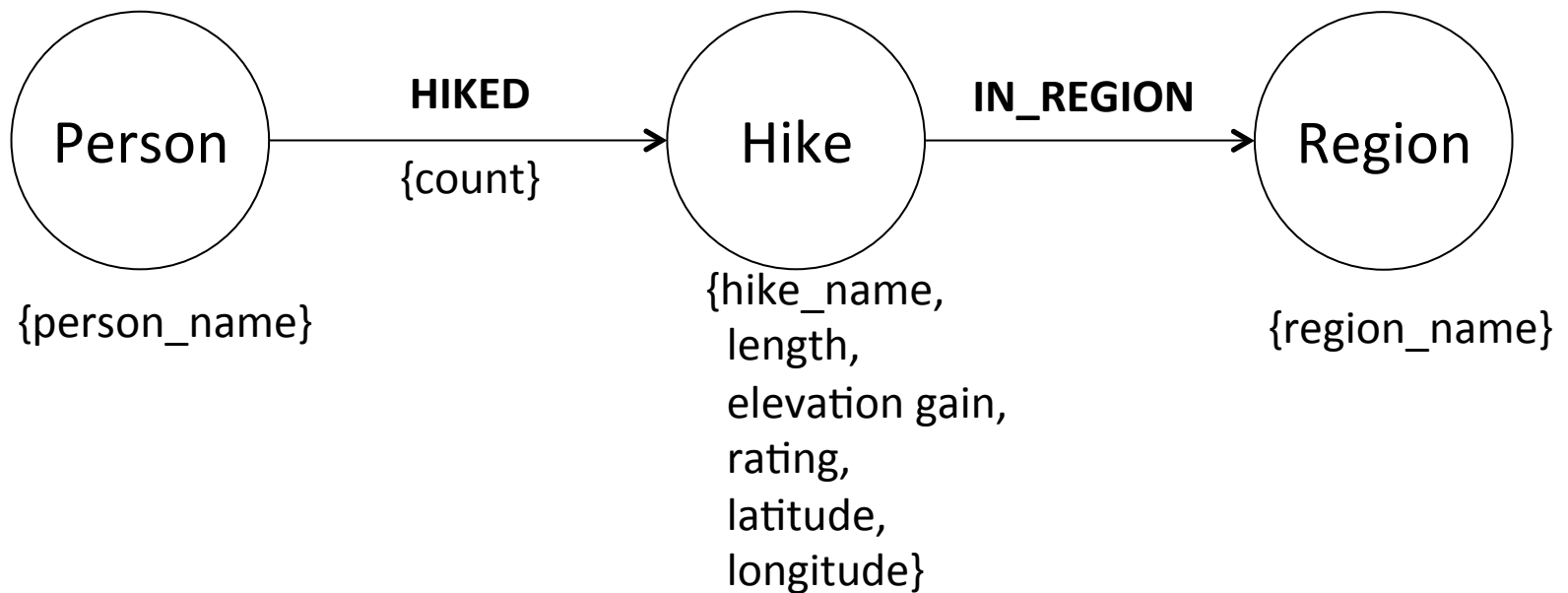
Example 3: find Alice's friend-of-friends

```
MATCH (p:Person {person_name: 'Alice'})-
    [:KNOWS*..2]->(friend_of_friend)
WHERE NOT (p)-[:KNOWS]-(friend_of_friend)
RETURN friend_of_friend.person_name;
```

```
// Equivalent query
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c)
WHERE a.person_name = 'Alice'
AND NOT (a)-[:KNOWS]->(c)
RETURN c.person_name;
```


Part 2: Neo4j graphs

Part 3: Build your own recommender

Data model

Person {person_name} —HIKED {count}→ Hike {hike_name, length, elevation gain, rating, latitude, longitude} —IN_REGION→ Region {region_name}

Let's gradually build up our recommender

1. Universally popular hikes
2. A person's favorite hikes
3. Hikes that similar people liked

Take a moment to run these in the browser:
1. 'clean_database.cypher'
2. 'import_hikes.cypher'

Warm-up queries

Find…

- All the hikes on the Olympic Peninsula
- All the hikes with more than 4000 feet of elevation gain (**hint**: use toInt())

1. Universally popular hikes

Let's define "popular" as the hikes that our hikers hiked most often.

Write a query that finds the most popular hikes.

2. A robot's favorite* hikes

Write a query that lists Bender's top 5 favorite (most hiked) hikes.

3. Suggest new* hikes to Bender

Write a query that:

1. takes Bender's 5 favorite hikes,

2. finds other people that liked those hikes, and

3. returns the favorite hikes of those other people.

Hint: WITH will be useful.

3. Suggest hikes to Bender

- Take the favorite hikes,
- And find other hikers who hiked Bender's favorite hikes:
- We need to take Bender out of his recommendations:
- Now find those hikers' favorite hikes that are not Bender's favorite hikes:
- Now take out Bender's favorite hikes and return some recommendations:

Part 3: Build your own recommender

Improvements

- Score recommendations and boost/penalize some scores
- Remove irrelevant or lowly rated recommendations
  - Blacklist, filter
- Measure the quality of recommendations
- Different graph implementation

Part 3: Build your own recommender

What I hope you've gained

- Appreciation for a new data structure
- Insight into the complicated world of recommendations

Wrap-up

Non-technical concerns

- $$$

- Computers are not creative:
  - Apple and Spotify hire DJs to compile some of their playlists

- Machines make educated guesses but can't understand music or appreciate the utility of an item

Wrap-up