


# Introduction to Graph Databases with Neo4j

Michael Moussa

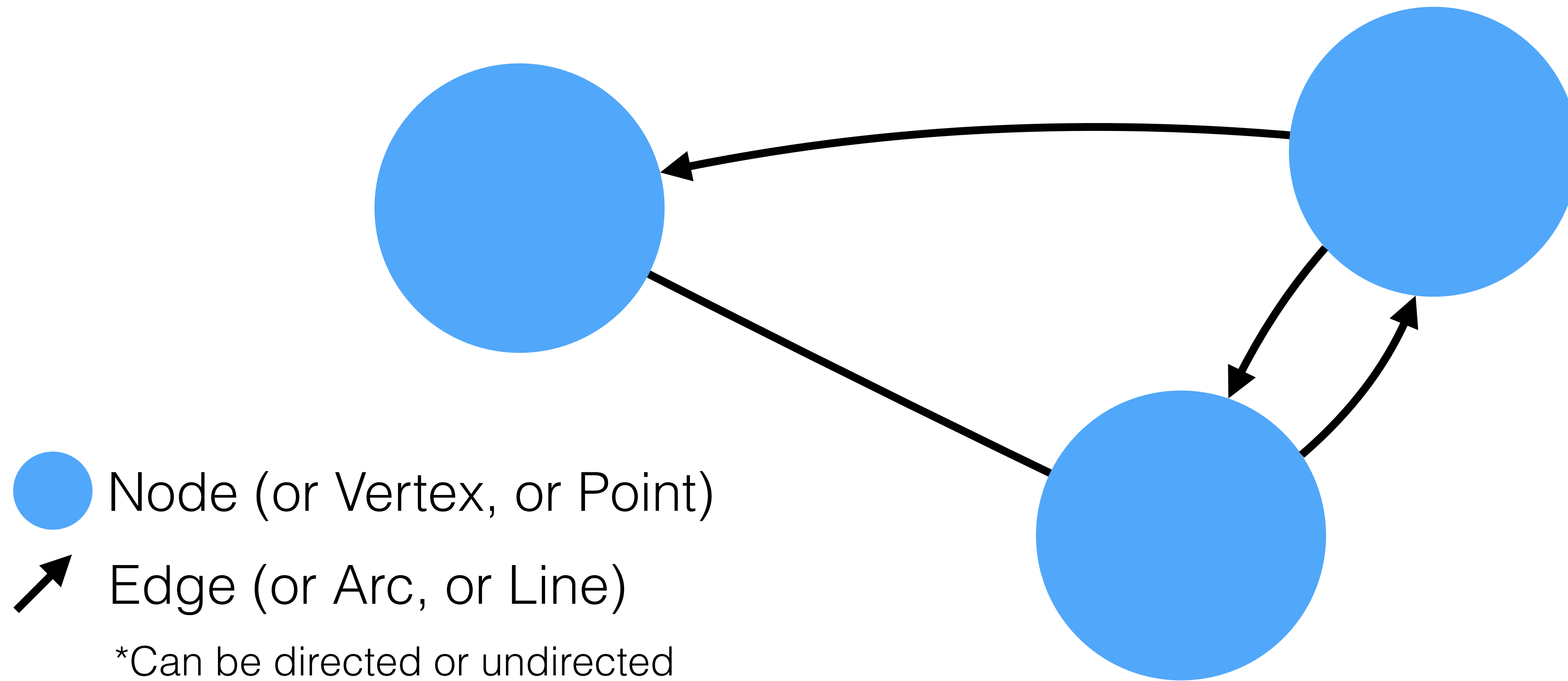


# About Me

- Web application developer for 17 years
- Solutions Architect at  **rackspace**  
the #1 managed cloud company
- Zend Expressive maintainer & general open source contributor

# What is a Graph?

Structure that models relationships between objects





# Graph Theory

Study of graphs and their applications

Used to model various problems in biology, chemistry, physics, etc.

Leonhard Euler (1707-1783) considered the "father" of graph theory



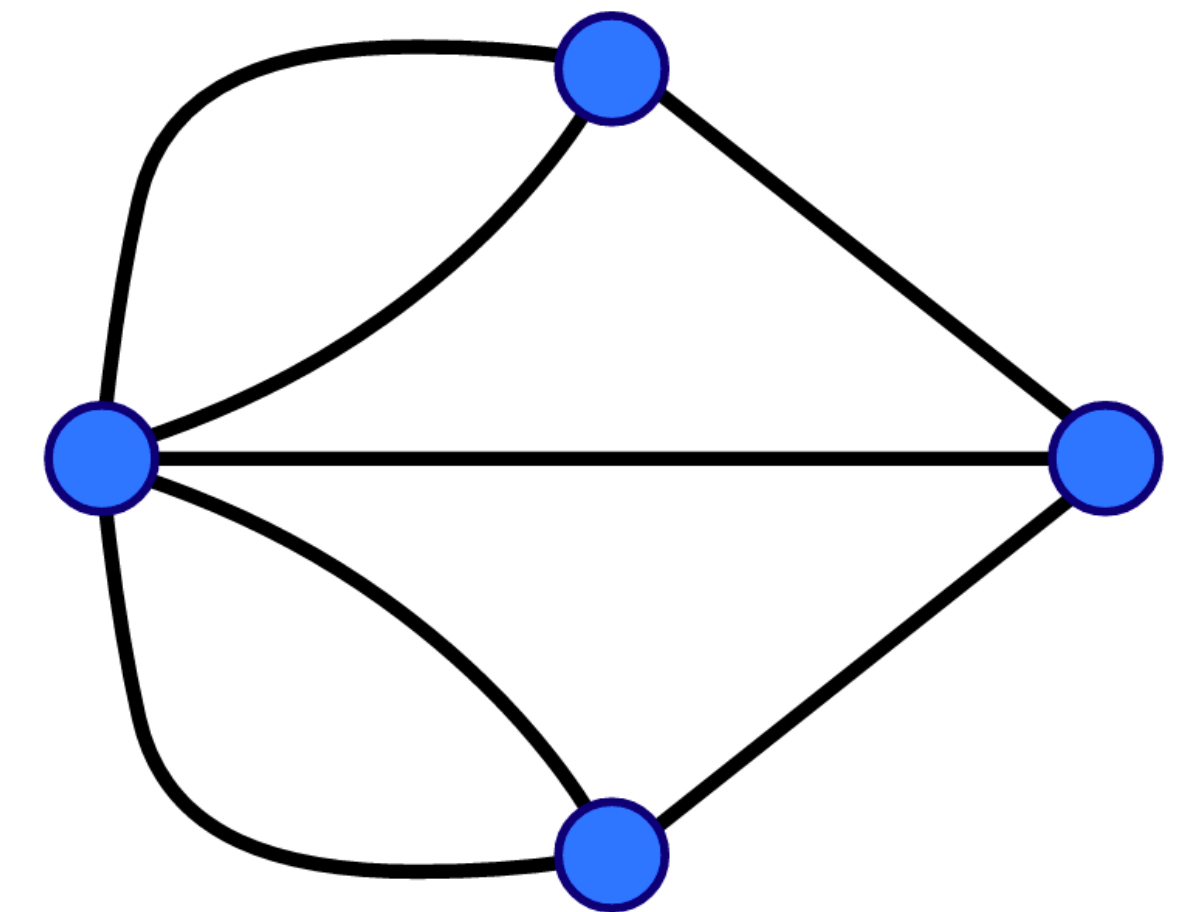
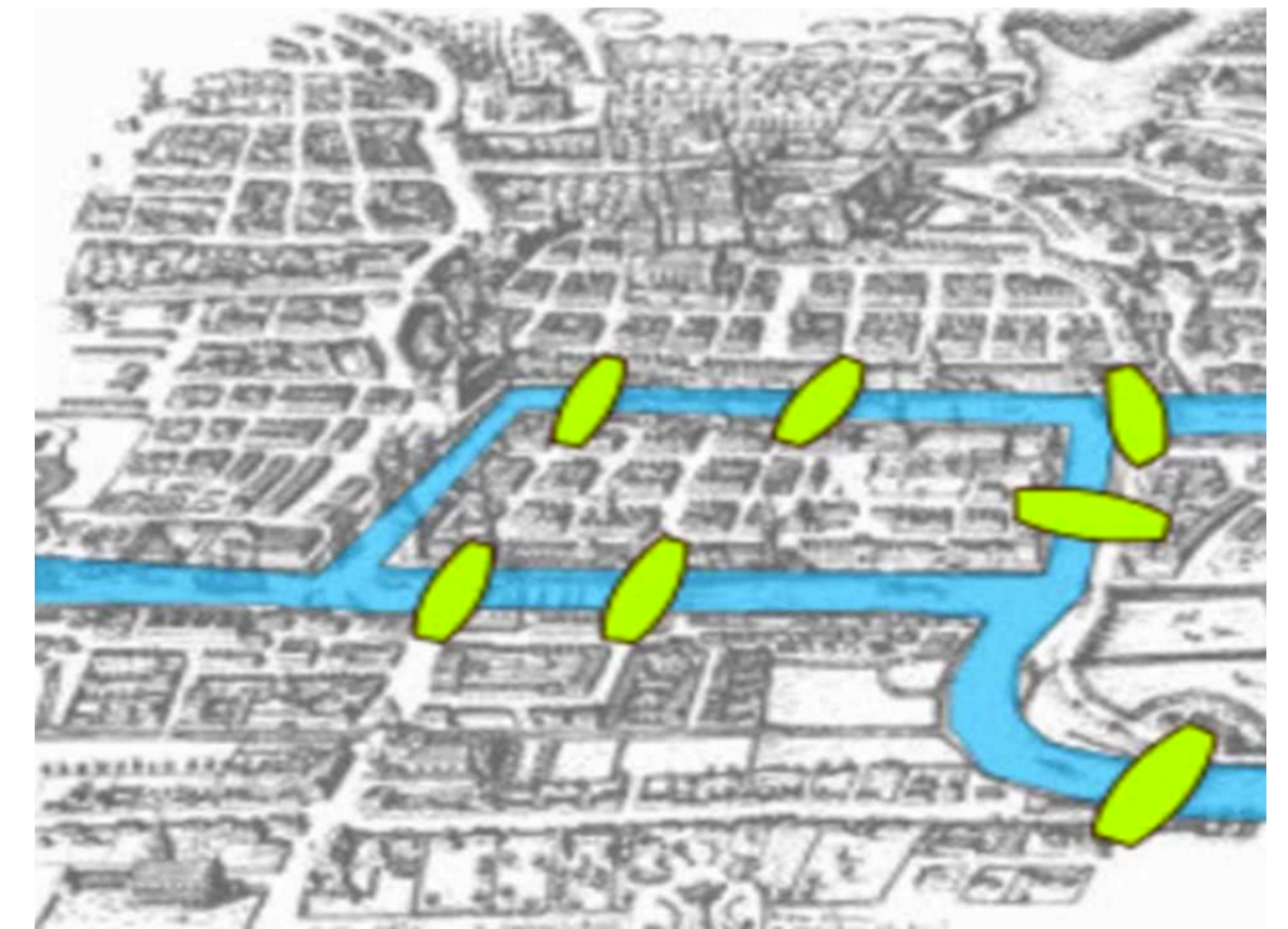


# Seven Bridges of Königsberg

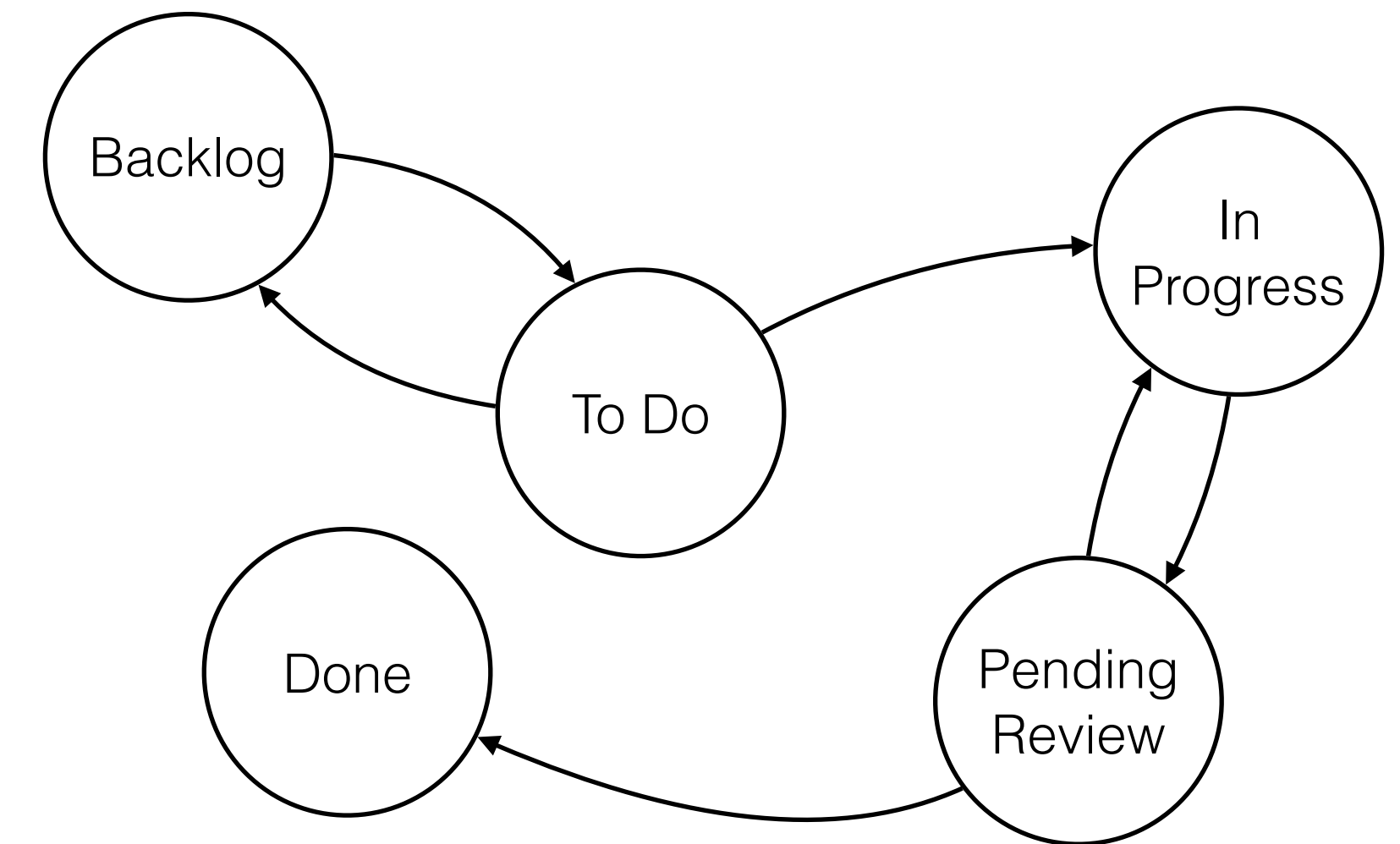
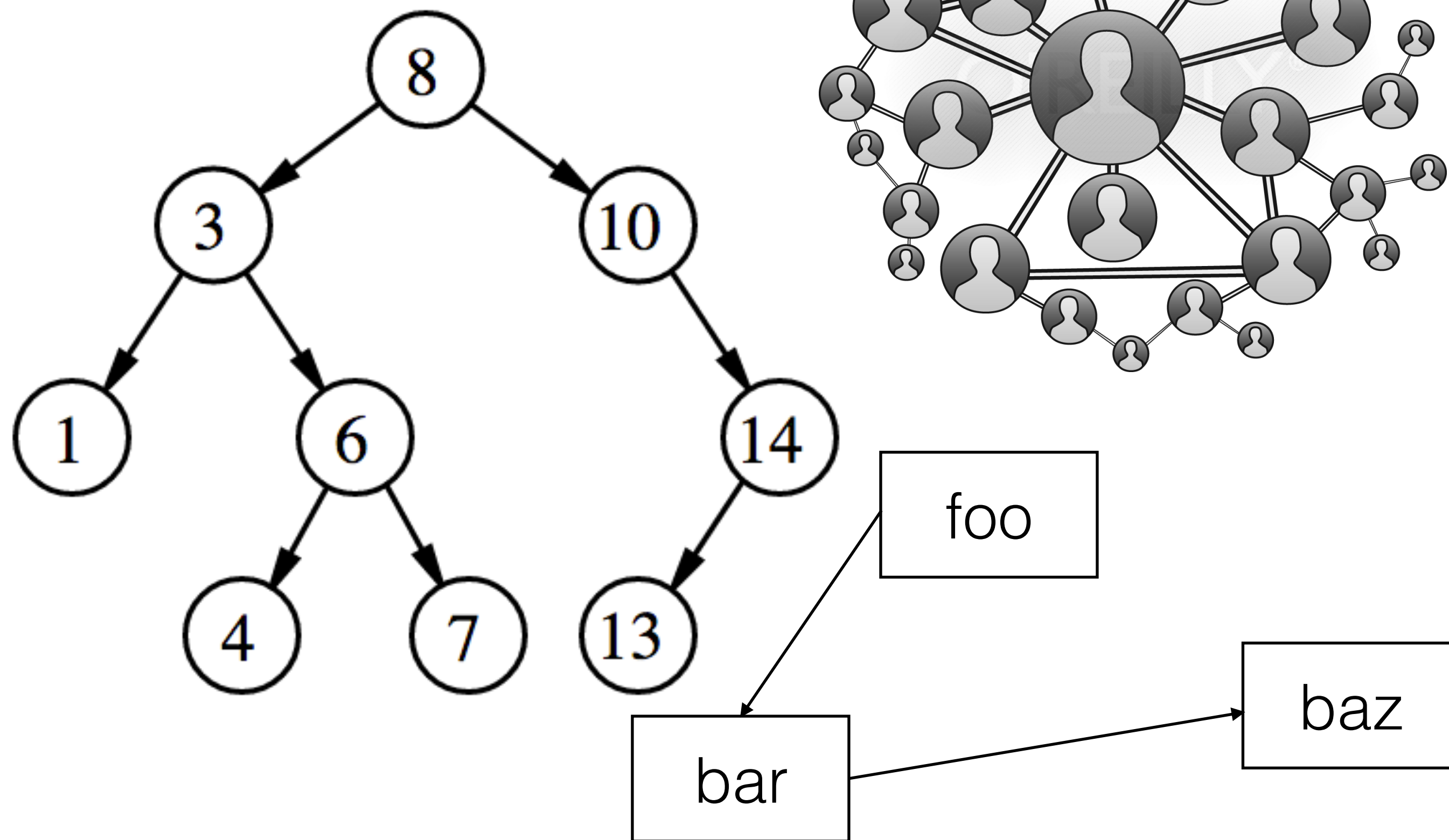
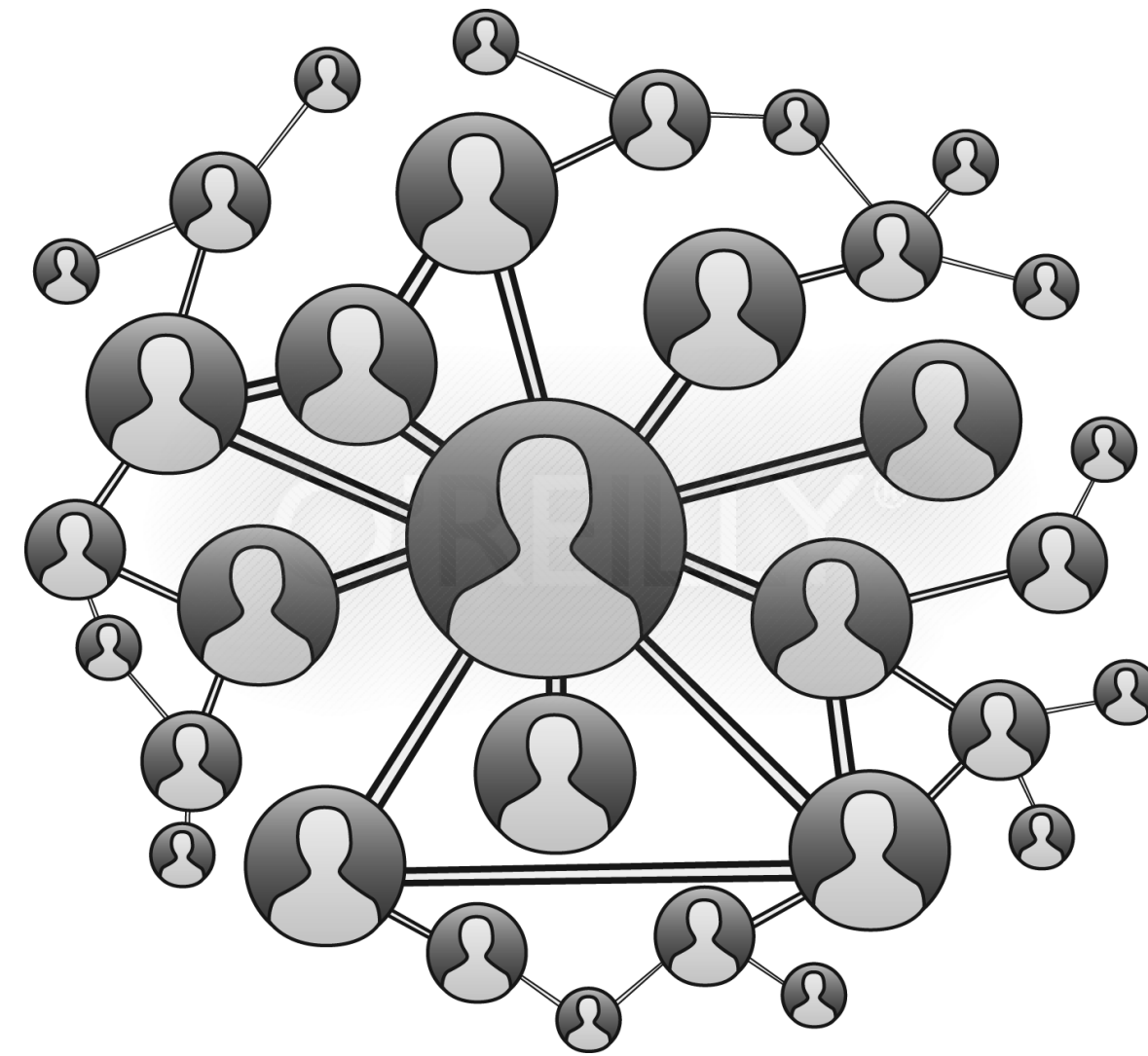
Can you walk through the city crossing each and every bridge exactly once?

Euler proved in 1736 that the answer is "No".

His solution is the first theorem of graph theory.

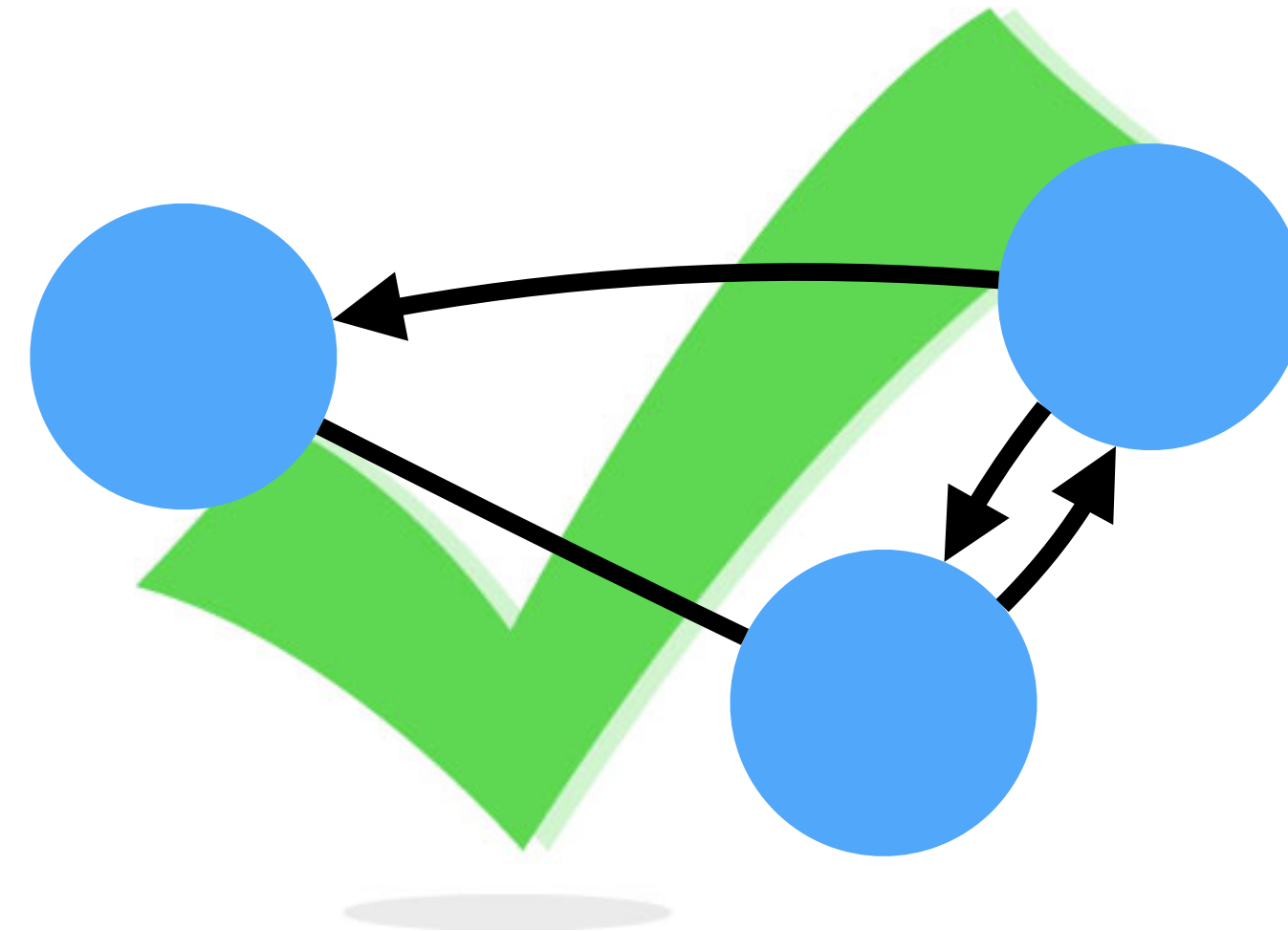


# Graphs are Everywhere!



- Driving directions
- Production recommendations
- Financial fraud detection
- Identity and access management

# Graph Database

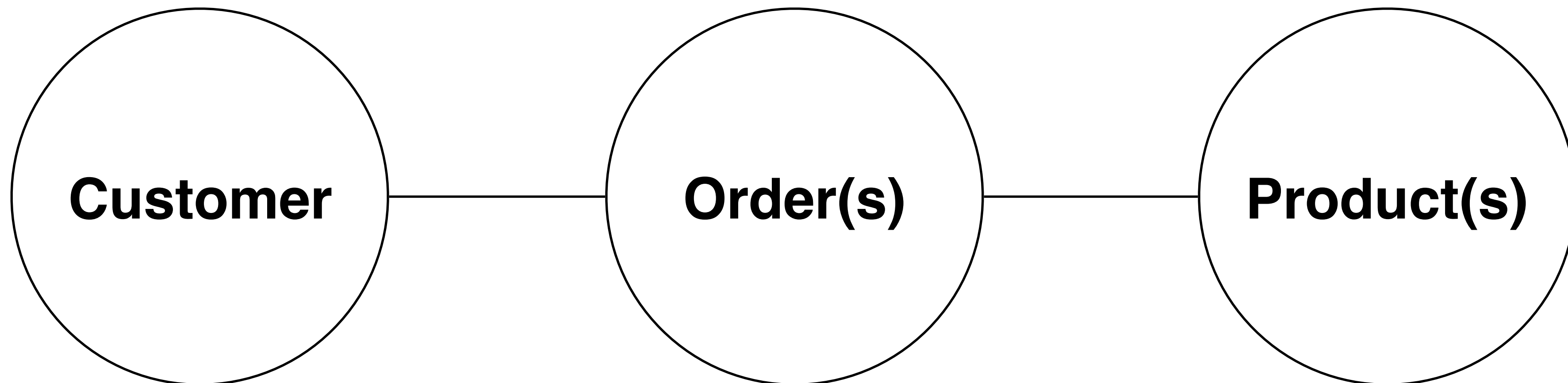


- Store information as nodes and relationships
- Relationships between data are "first-class citizens"



# E-commerce Example

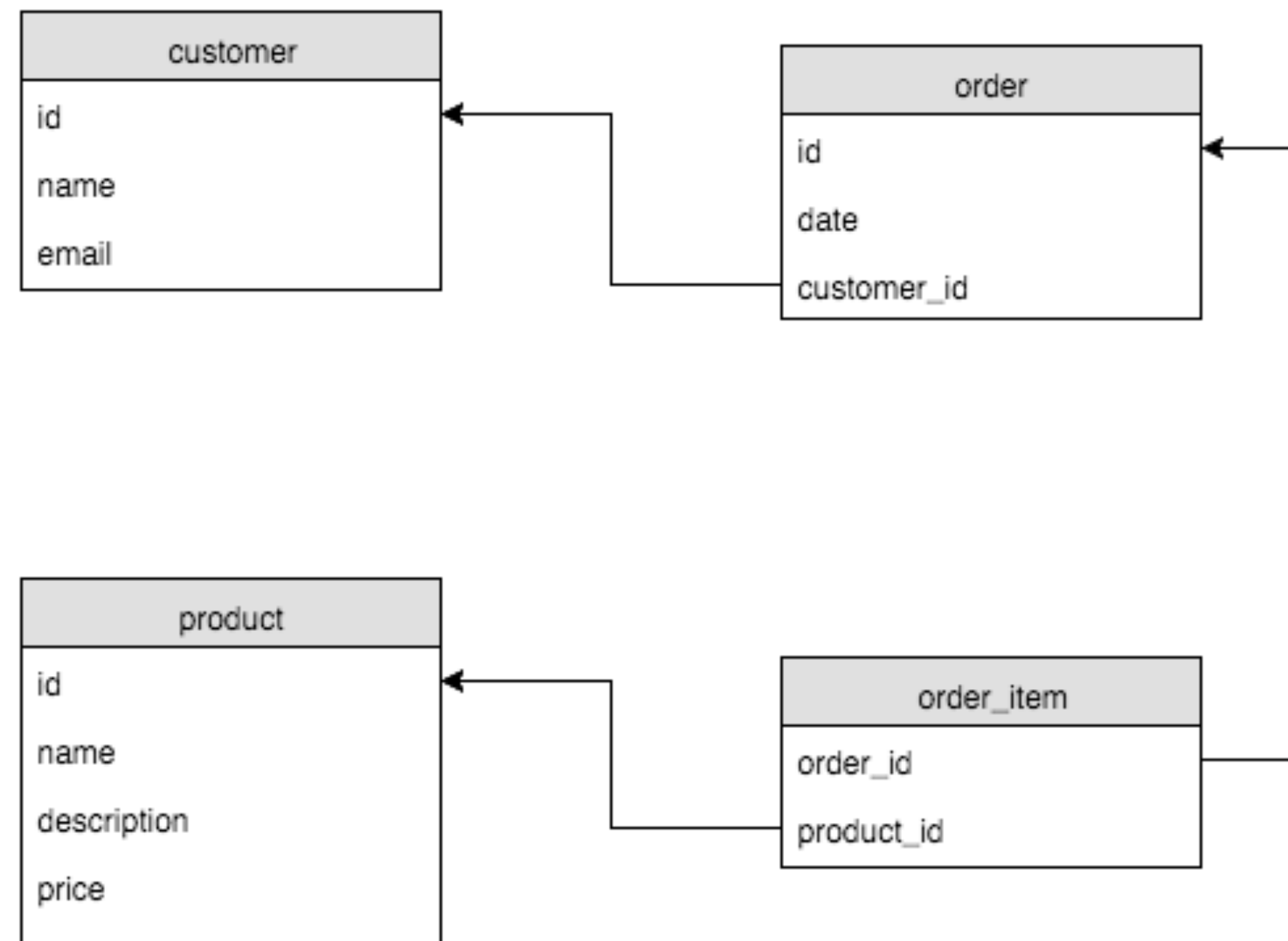
A **customer** places **order(s)** consisting of **product(s)**





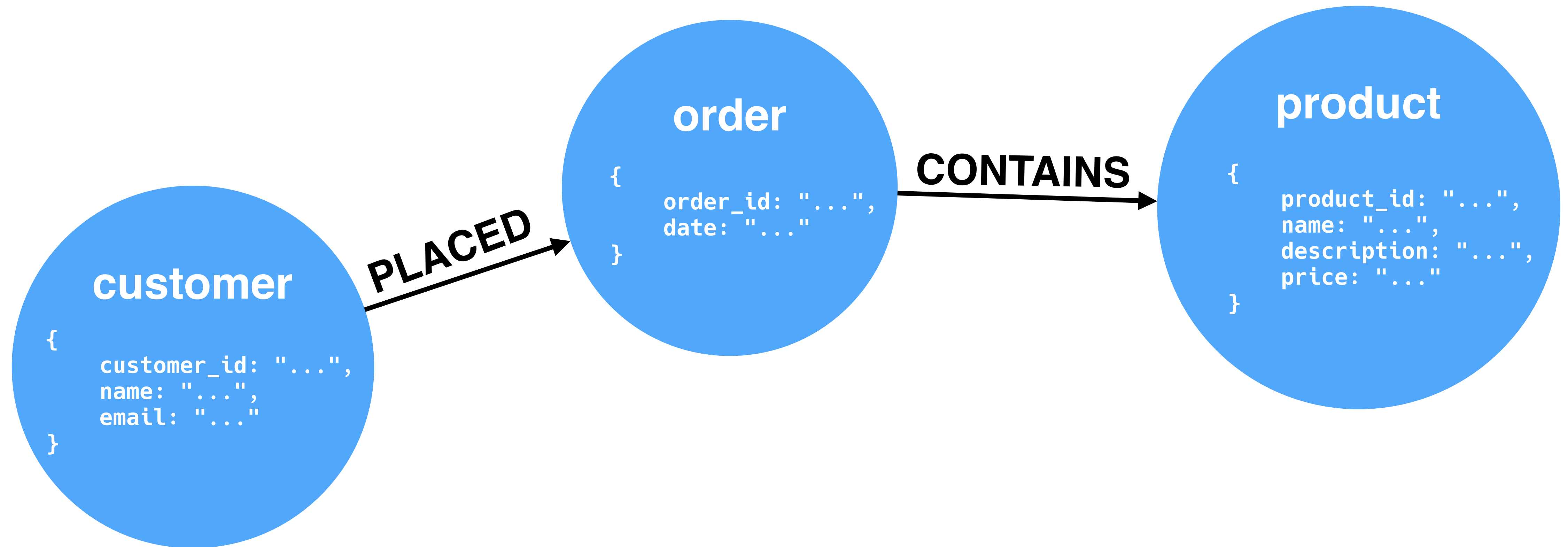
# E-commerce Example

Modeling in a relational database



# E-commerce Example

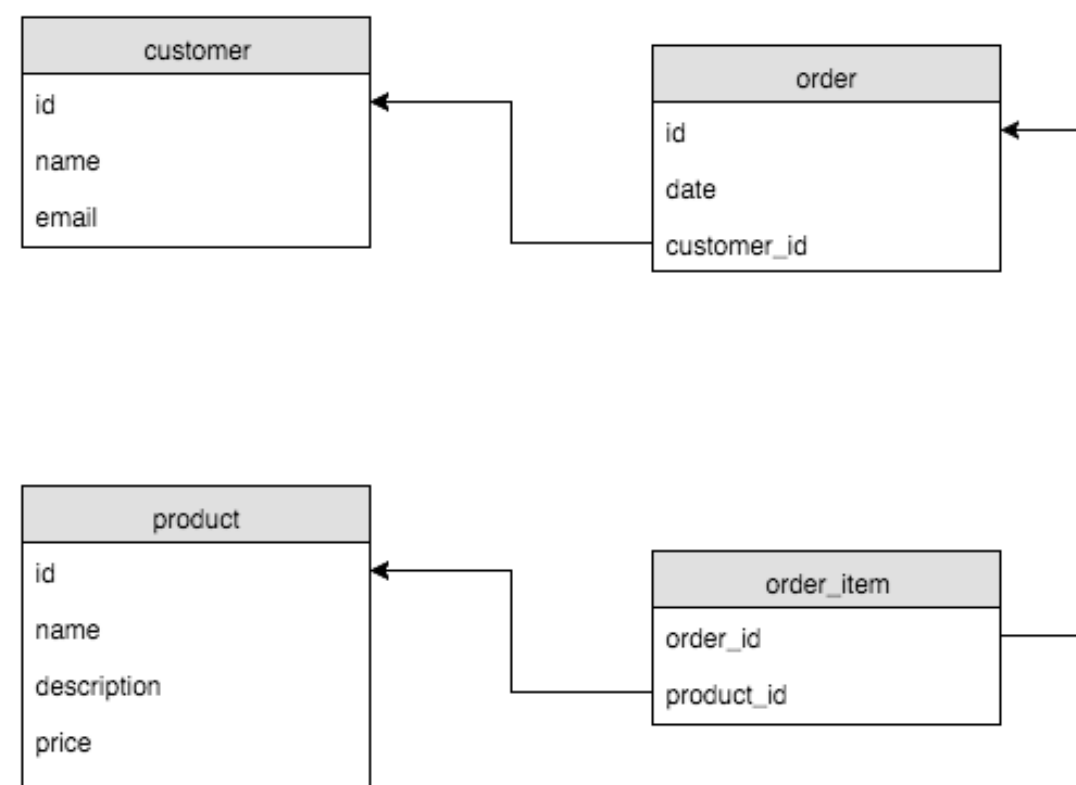
Modeling in a graph database



# E-commerce Example

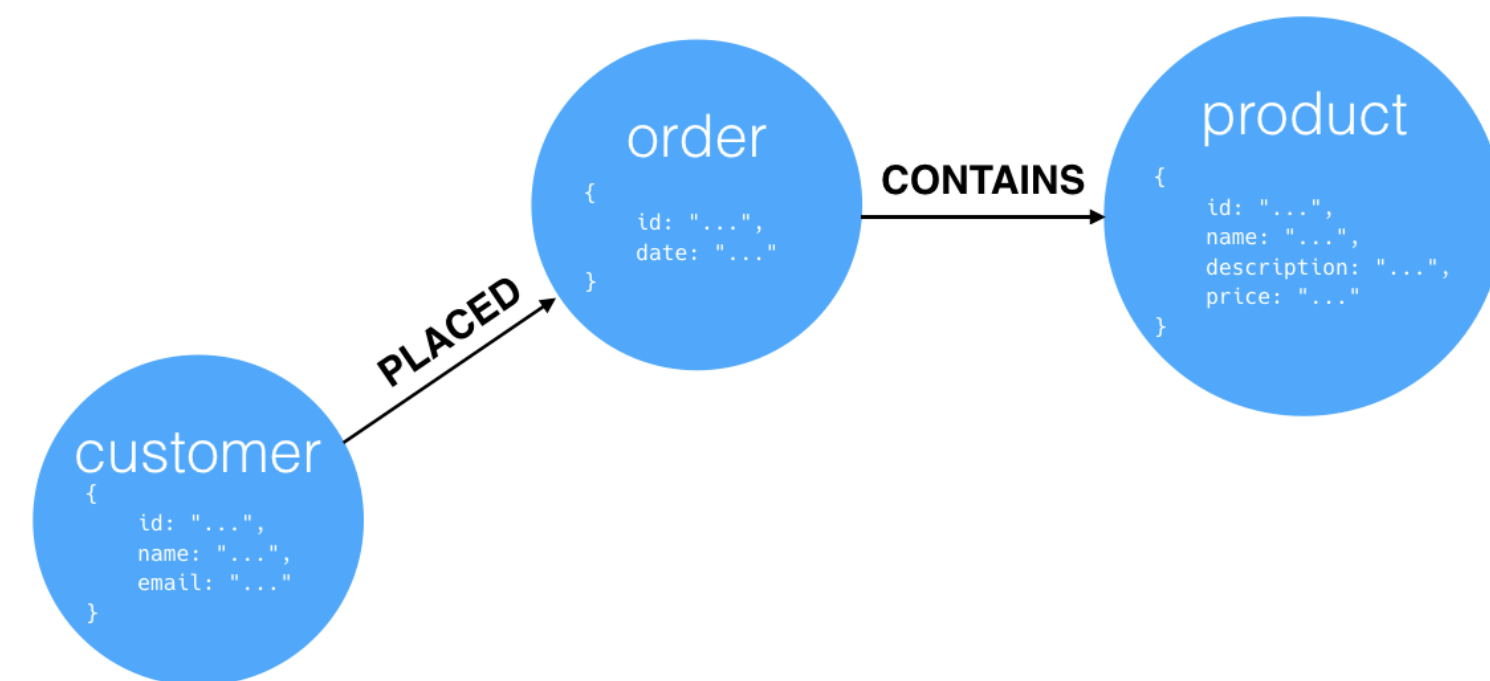
## RDBMS

- Highly-structured
- Entities related via JOIN tables and foreign keys



## Graph DB

- Schemaless, flexible
- Relationships are as important as the data itself







The world's most popular graph database

- Native graph database
- Open source
- ACID compliant
- Powerful and expressive query language
- Excellent documentation
- Active community

# Labeled Property Graph

- Entities are **nodes** containing various **properties**.
- **Relationships** connect those nodes to others, and are enhanced by **properties** of their own.
- Nodes are grouped with like nodes using **labels**.

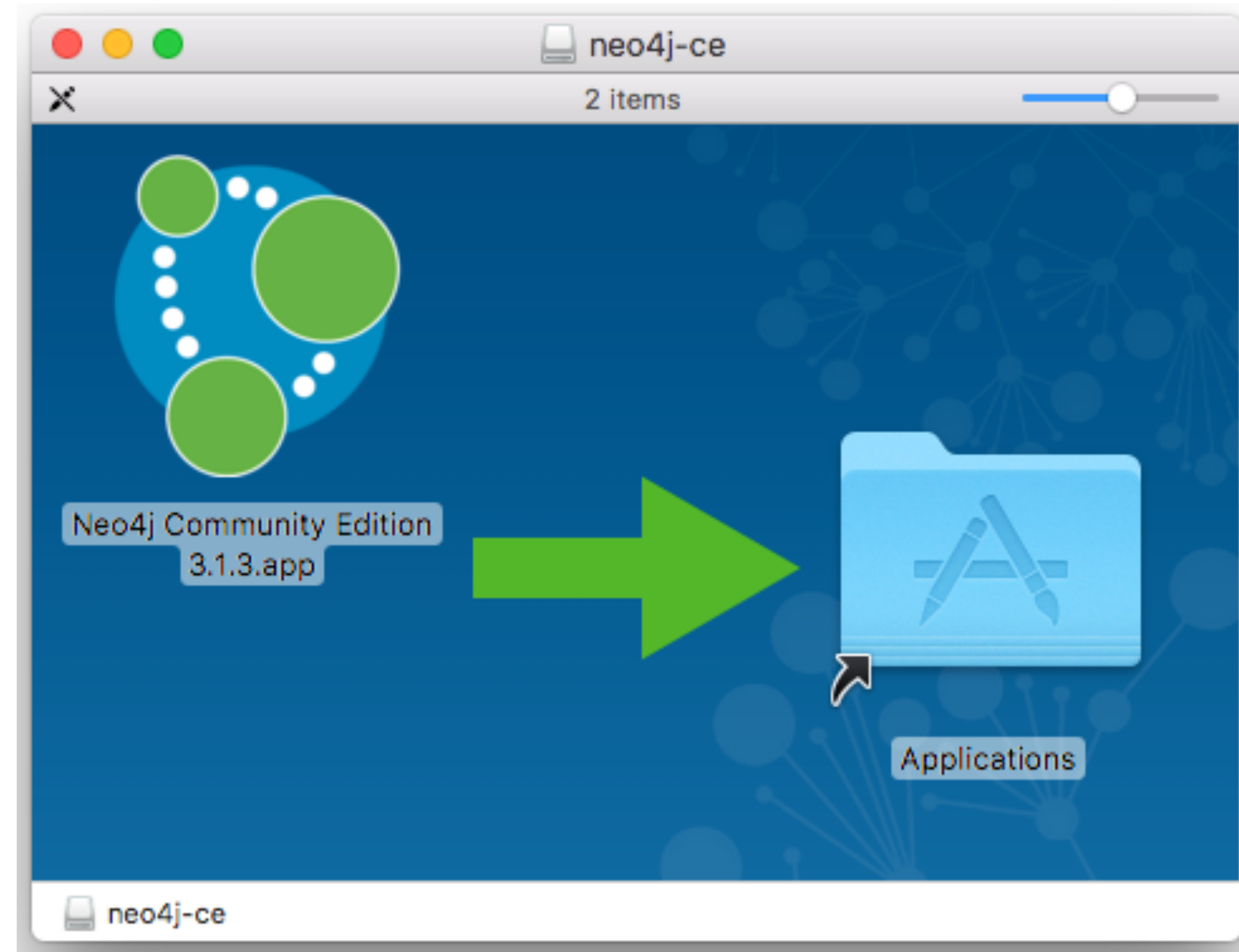




# Cypher

- Neo4j's query language
- Declarative
- SQL-inspired
- Open source!
  - <http://opencypher.org>


# Installing Neo4j



<https://neo4j.com/download/>

# Neo4j Online Sandbox


### Recommendations



Generate personalized real-time recommendations using a dataset of movie reviews.

[Launch Sandbox](#)


### Trumpworld



Explore connections in and around the Trump Administration using this dataset from BuzzFeed.

[Launch Sandbox](#)


### Twitter



If signed into Neo4j Sandbox using Twitter, this Sandbox will allow you to Graph your Twitter network.

[Launch Sandbox](#)


### Legis-Graph



US Congress modeled as a Graph - bills, votes, members, and more.

[Launch Sandbox](#)

### Blank Sandbox



Blank Sandbox. Load your own data with LOAD CSV or create data from scratch.

[Launch Sandbox](#)

<https://neo4j.com/sandbox-v2/>

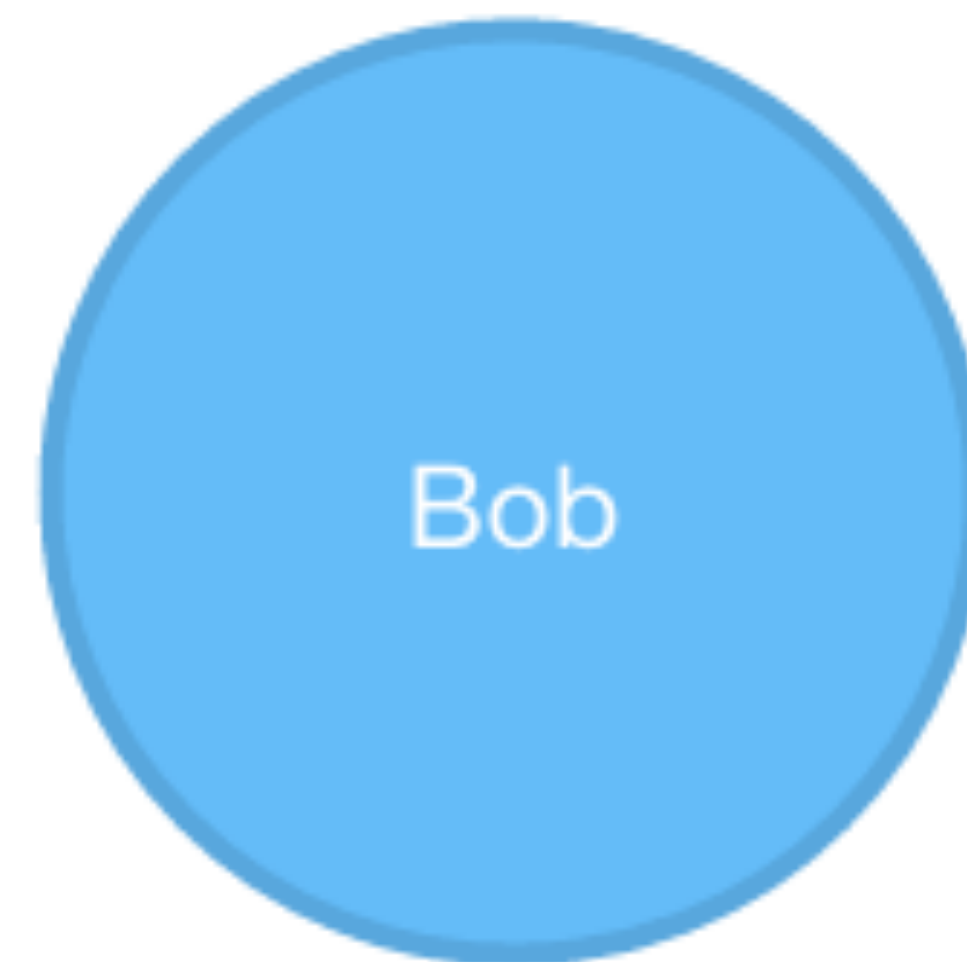


// Create some products

```
CREATE ( :Product {product_id: 12, name: "Dog food", price: 20,  
                  description: "The most delicious dog food ever"}),  
      ( :Product {product_id: 34, name: "Dog biscuits", price: 5,  
                  description: "The yummiest dog biscuits ever"}),  
      ( :Product {product_id: 56, name: "Squeaky dog toy", price: 10,  
                  description: "A fun squeaky toy"}),  
      ( :Product {product_id: 78, name: "Comfy cat bed", price: 30,  
                  description: "Soft and claw-proof"}),  
      ( :Product {product_id: 90, name: "Sunglasses", price: 15,  
                  description: "It has nothing to do with pets"});
```

Added 5 labels, created 5 nodes, set 20 properties

```
// Create some customers
CREATE (alice:Customer {name: "Alice", email: "alice@example.com"}),
      (bob:Customer {name: "Bob", email: "bob@example.com"}),
      (:Customer {name: "Charlie", email: "charlie@example.com"}),
      (:Customer {name: "Dave", email: "dave@example.com"})
RETURN alice, bob;
```





```
// Alice orders dog food and dog biscuits
MATCH (customer:Customer), (product:Product)
WHERE customer.email = "alice@example.com"
      AND product.product_id IN [12, 34]
MERGE (customer)-[:PLACED]->(order:Order {date: "2016-02-05"})
MERGE (order)-[:CONTAINS]->(product);
```

Added 1 label, created 1 node, set 1 property, created 3 relationships

```
// Bob orders dog biscuits and a squeaky dog toy
MATCH (customer:Customer {email: "bob@example.com"}),
      (product1:Product {product_id: 34}),
      (product2:Product {product_id: 56})
MERGE (customer)-[:PLACED]->(order:Order {date: "2016-02-05"})
MERGE (order)-[:CONTAINS]->(product1)
MERGE (order)-[:CONTAINS]->(product2);
```

Added 1 label, created 1 node, set 1 property, created 3 relationships

```
// Charlie orders dog food, dog biscuits,  
// a squeaky dog toy, and sunglasses  
MATCH (customer:Customer {email: "charlie@example.com"}),  
      (product:Product)  
WHERE product.product_id IN [12, 34, 56, 90]  
MERGE (customer)-[:PLACED]->(order:Order {date: "2016-02-05"})  
MERGE (order)-[:CONTAINS]->(product);
```

Added 1 label, created 1 node, set 1 property, created 5 relationships

```
// Dave orders sunglasses and a comfy cat bed
MATCH (customer:Customer {email: "dave@example.com"},
      (product1:Product {product_id: 78}),
      (product2:Product {product_id: 90})
MERGE (customer)-[:PLACED]->(order:Order {date: "2016-02-05"})
MERGE (product1)<-[:CONTAINS]-(order)-[:CONTAINS]->(product2);
```

Added 1 label, created 1 node, set 1 property, created 3 relationships

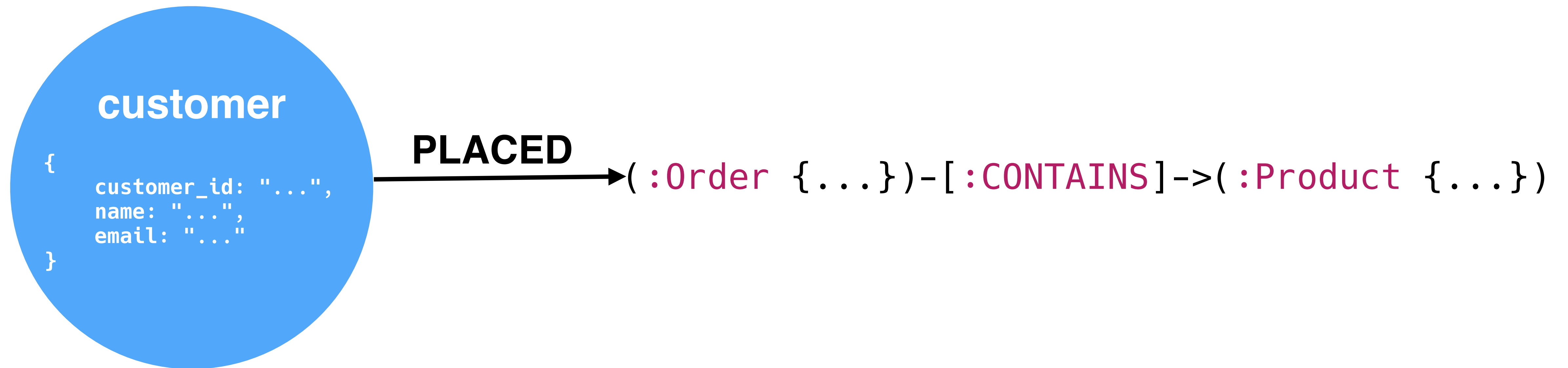


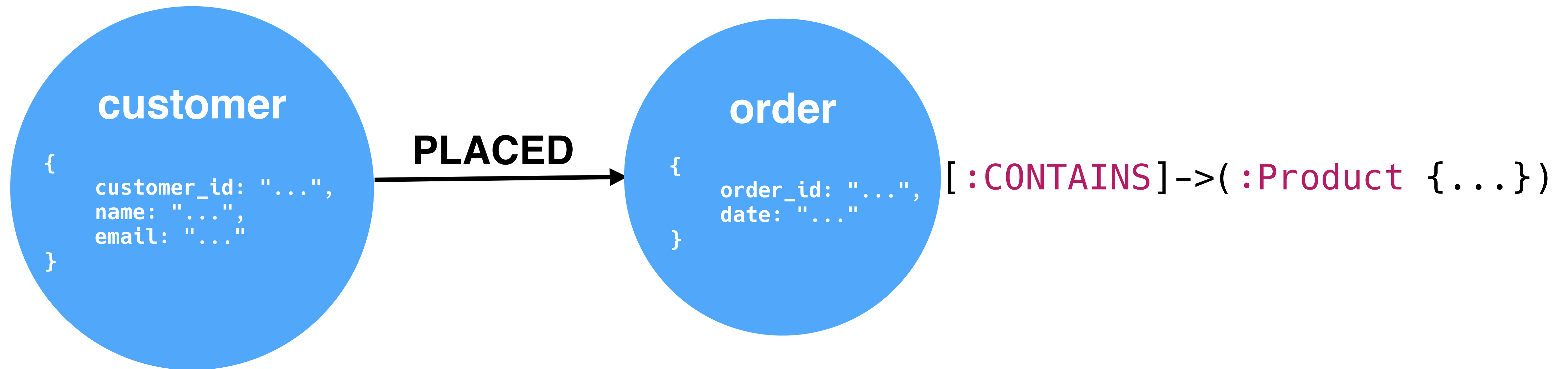
( :Customer { . . . } ) - [ :PLACED ] -> ( :Order { . . . } ) - [ :CONTAINS ] -> ( :Product { . . . } )

## customer

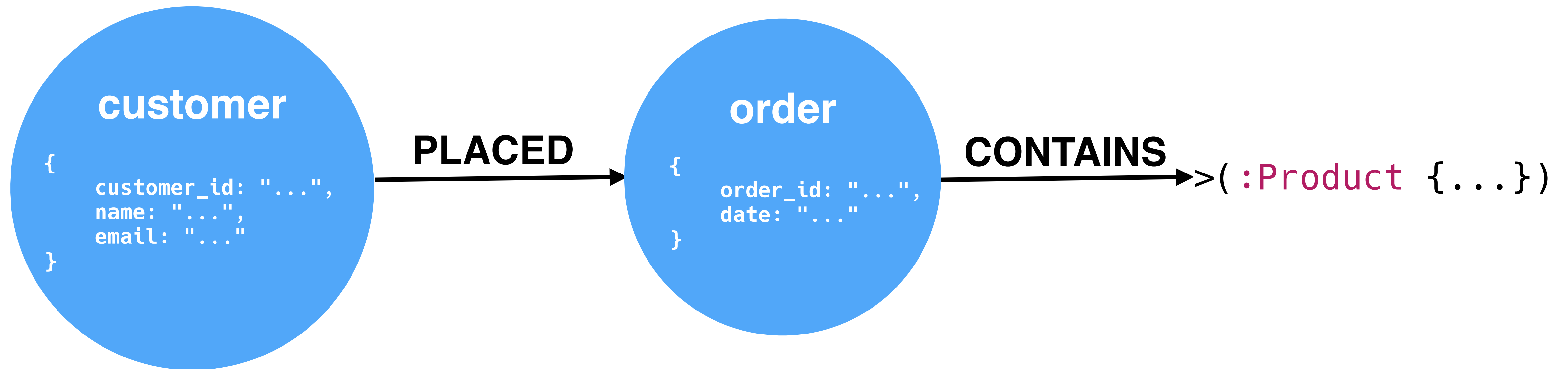
```
{  
  customer_id: "...",  
  name: "...",  
  email: "..."  
}
```

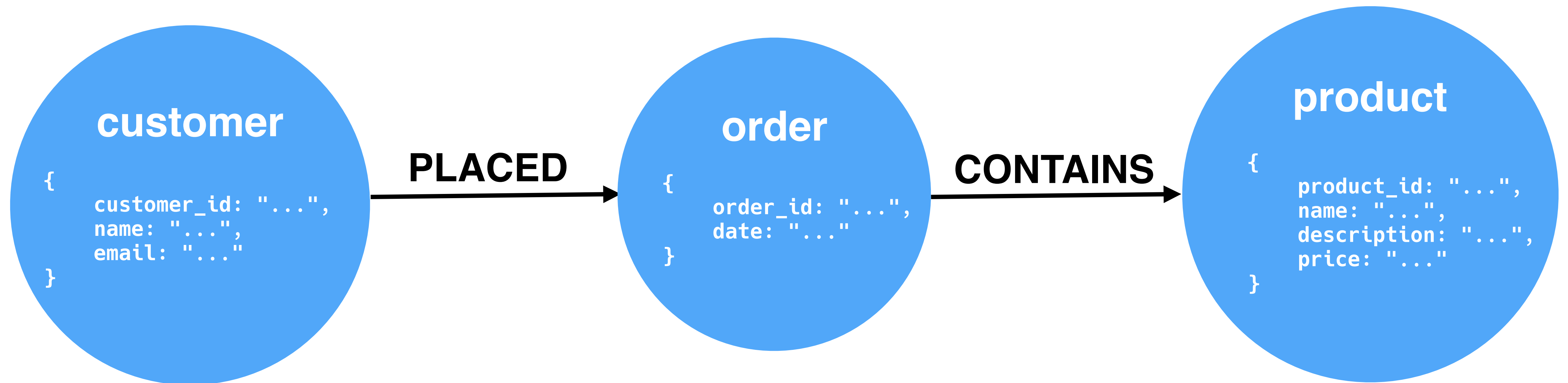
`- [ :PLACED ] -> ( :Order { ... } ) - [ :CONTAINS ] -> ( :Product { ... } )`



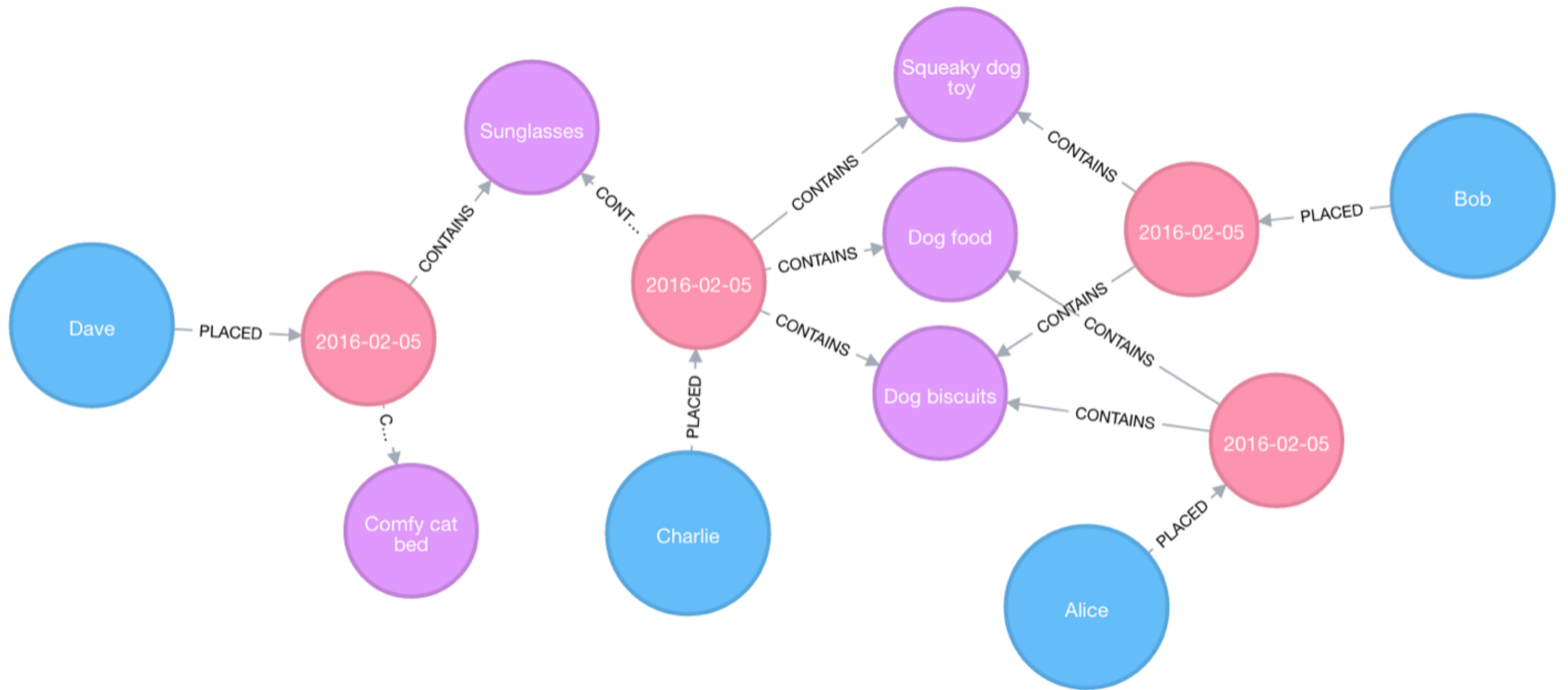








```
// Let's see what we've created.  
MATCH (n)  
RETURN n  
LIMIT 50;
```





```
// Someone is browsing the dog biscuit item (product_id: 34)
// What else should we recommend that they buy?
MATCH (order:Order)-[:CONTAINS]->(product:Product {product_id: 34}),
      (order)-[:CONTAINS]->(recommendation:Product)
WHERE NOT recommendation = product
RETURN recommendation.name,
       COUNT(recommendation) as recommended
ORDER BY recommended DESC;
```

**recommendation.name**

**recommended**

Dog food

2

Squeaky dog toy

2

Sunglasses

1

```
// Alice is browsing the dog biscuit page.
// What should we recommend that she order?
MATCH (alice:Customer {email: 'alice@example.com'})-[*2]->(p:Product)
WITH COLLECT(p.product_id) AS excludedProducts
MATCH (customer:Customer)-[:PLACED]->
      (order:Order)-[:CONTAINS]->(product:Product {product_id: 34}),
      (order)-[:CONTAINS]->(recommendation:Product)
WHERE NOT recommendation.product_id IN excludedProducts
RETURN recommendation.name, COUNT(recommendation) as recommended
ORDER BY recommended DESC;
```

**recommendation.name**

**recommended**

Squeaky dog toy

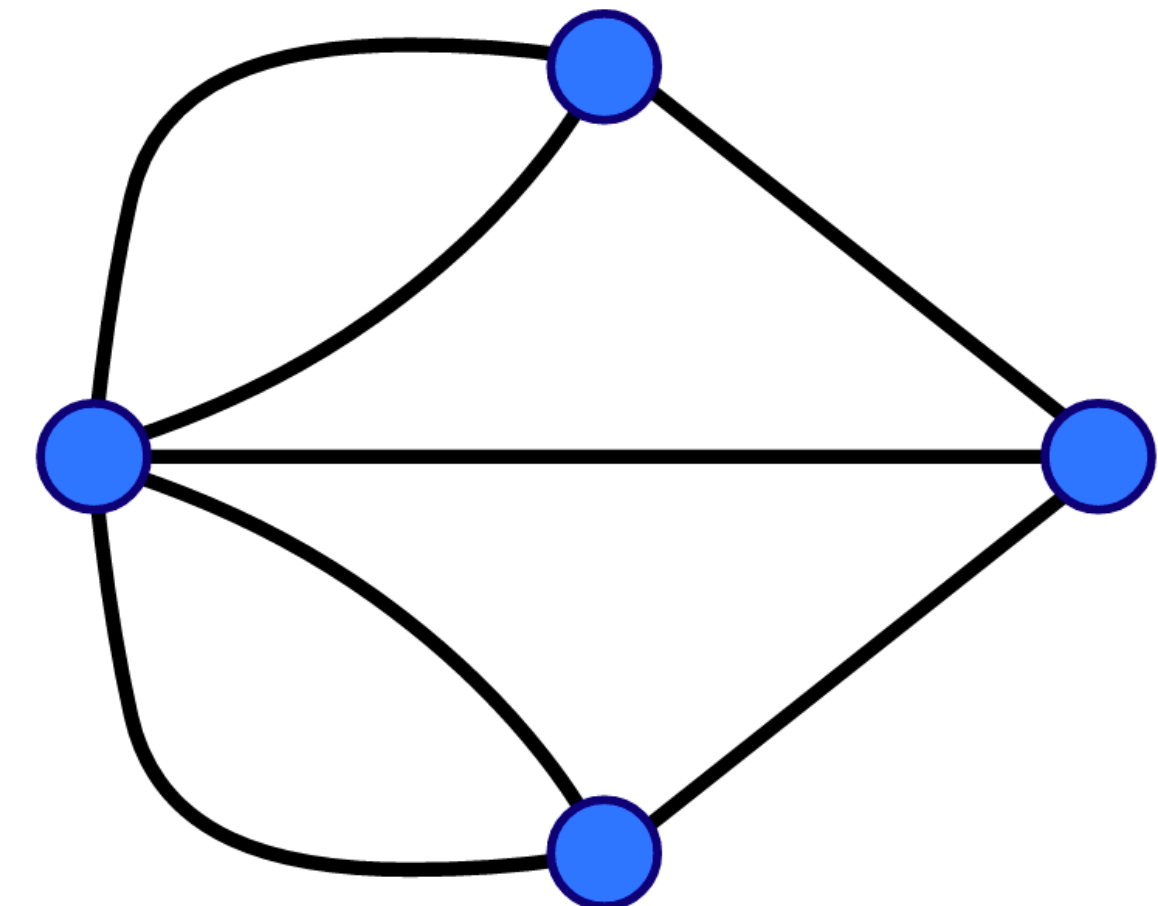
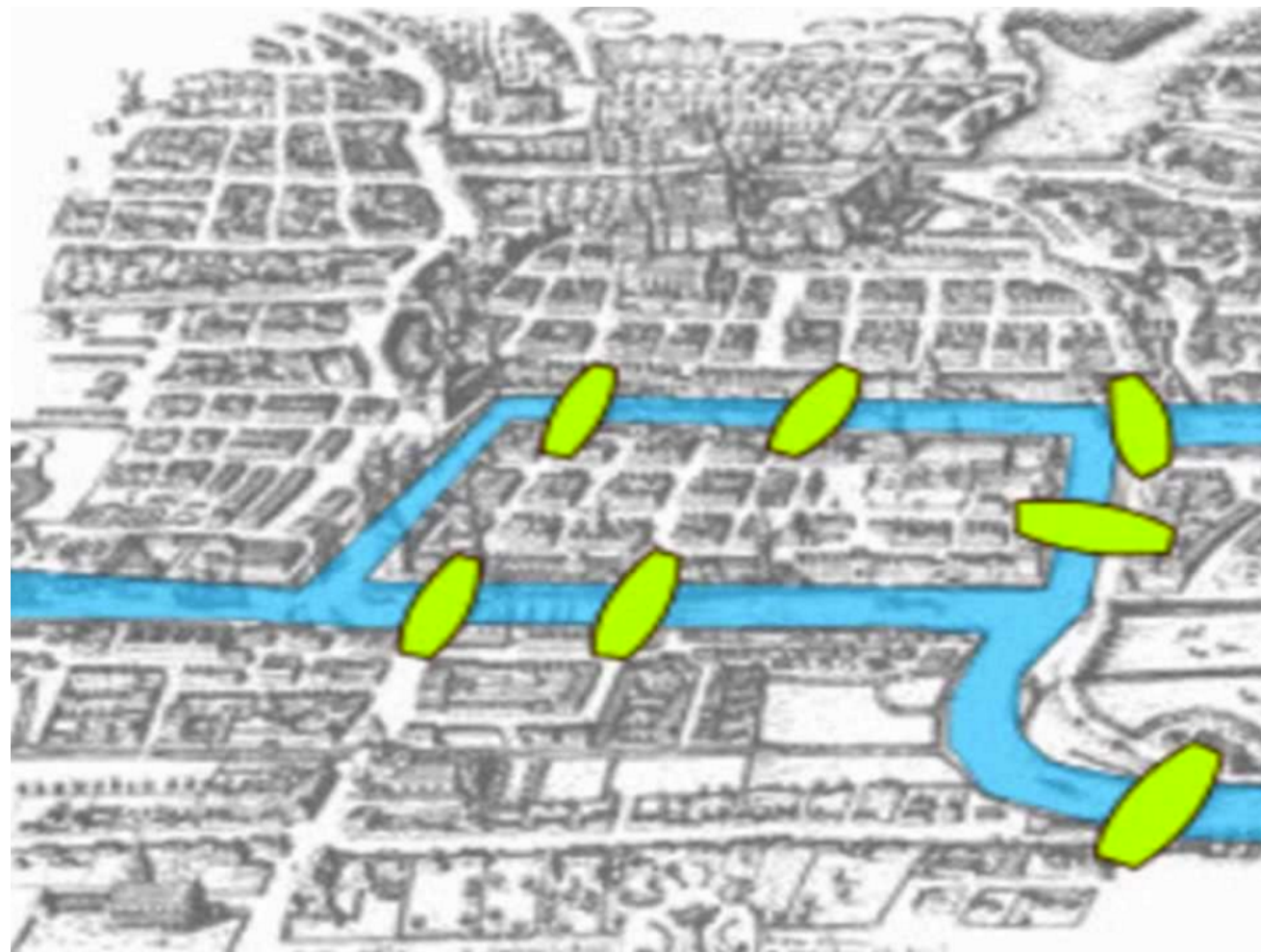
2

Sunglasses

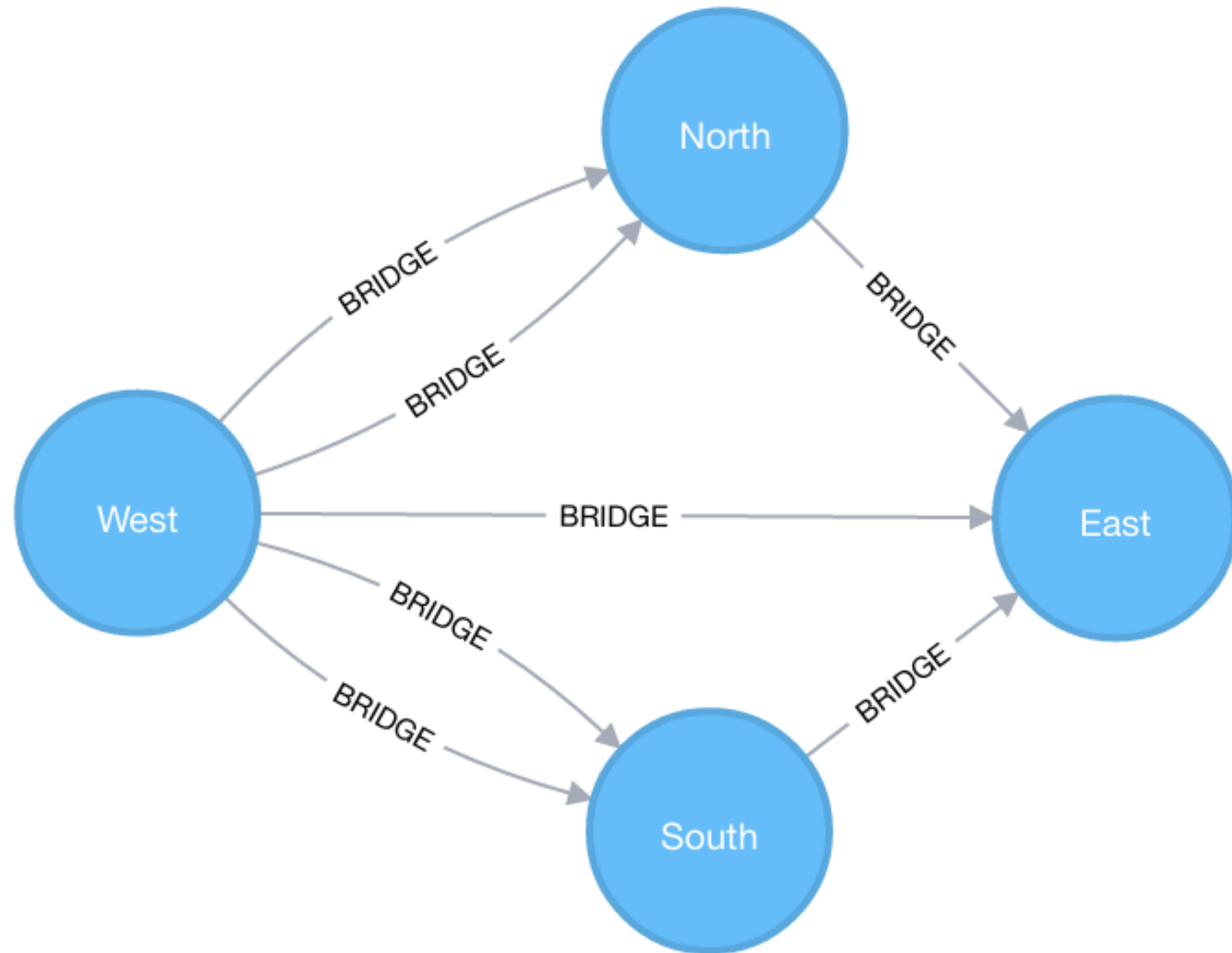
1



# Remember the Seven Bridges of Königsberg?



```
CREATE (west:Land {name: "West"}), (north:Land {name: "North"}),  
      (east:Land {name: "East"}), (south:Land {name: "South"}),  
  
      (west)-[b1:BRIDGE {name: "West to North (1)"}]->(north),  
      (west)-[b2:BRIDGE {name: "West to North (2)"}]->(north),  
      (north)-[b3:BRIDGE {name: "North to East"}]->(east),  
      (west)-[b4:BRIDGE {name: "West to South (1)"}]->(south),  
      (west)-[b5:BRIDGE {name: "West to South (2)"}]->(south),  
      (west)-[b6:BRIDGE {name: "West to East"}]->(east),  
      (south)-[b7:BRIDGE {name: "South to East"}]->(east)  
  
RETURN west, north, south, east, b1, b2, b3, b4, b5, b6, b7;
```



```
// Since the Seven Bridges of Königsberg has no solution,  
// the following query will return no rows.
```

```
MATCH path=( :Land )-[ b1:BRIDGE ]-( :Land )-[ b2:BRIDGE ]-  
           ( :Land )-[ b3:BRIDGE ]-( :Land )-[ b4:BRIDGE ]-  
           ( :Land )-[ b5:BRIDGE ]-( :Land )-[ b6:BRIDGE ]-  
           ( :Land )-[ b7:BRIDGE ]-( :Land )  
RETURN NODES(path);
```

```
// Since the Seven Bridges of Königsberg has no solution,  
// the following query will return no rows.
```

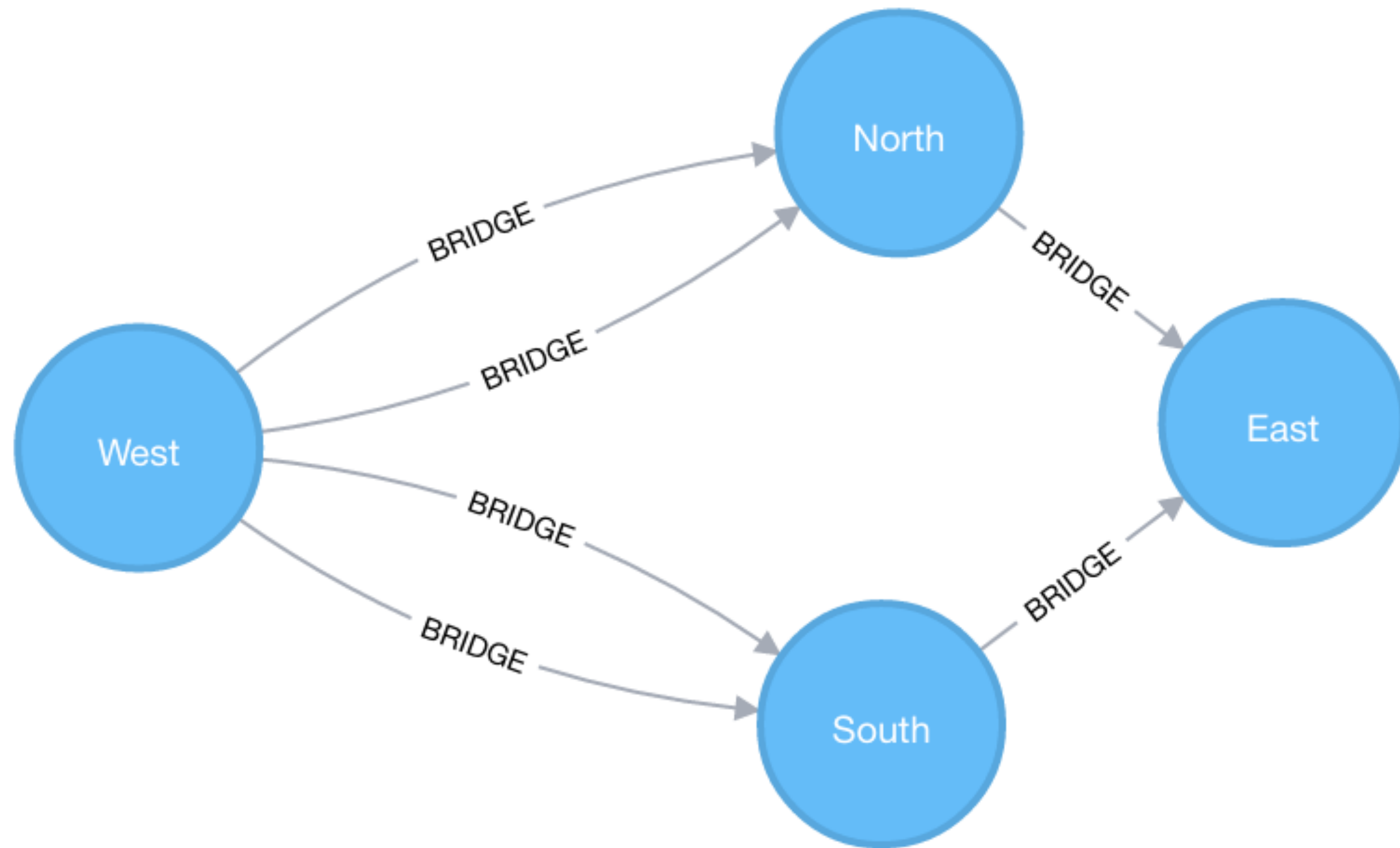
```
MATCH path=( :Land )-[ b1:BRIDGE ]-( :Land )-[ b2:BRIDGE ]-  
           ( :Land )-[ b3:BRIDGE ]-( :Land )-[ b4:BRIDGE ]-  
           ( :Land )-[ b5:BRIDGE ]-( :Land )-[ b6:BRIDGE ]-  
           ( :Land )-[ b7:BRIDGE ]-( :Land )  
RETURN NODES(path);
```

(no rows)



```
// Let's remove a bridge...  
MATCH ( )-[b:BRIDGE {name: "West to East"}]-( )  
DELETE b;
```

```
// Try again!
MATCH path=( :Land )-[ b1:BRIDGE ]-( :Land )-[ b2:BRIDGE ]-
          ( :Land )-[ b3:BRIDGE ]-( :Land )-[ b4:BRIDGE ]-
          ( :Land )-[ b5:BRIDGE ]-( :Land )-[ b6:BRIDGE ]-
          ( :Land )
RETURN NODES(path);
```



```
// Show us the way!
MATCH (land:Land)-[b1:BRIDGE]-( :Land)-[b2:BRIDGE]-
      ( :Land)-[b3:BRIDGE]-( :Land)-[b4:BRIDGE]-
      ( :Land)-[b5:BRIDGE]-( :Land)-[b6:BRIDGE]-( :Land)
RETURN land.name, b1.name, b2.name, b3.name,
       b4.name, b5.name, b6.name;
```





land.name	b1.name	b2.name	b3.name	b4.name	b5.name	b6.name
South	West to South (1)	West to South (2)	South to East	North to East	West to North (2)	West to North (1)
South	West to South (2)	West to South (1)	South to East	North to East	West to North (2)	West to North (1)
North	North to East	South to East	West to South (1)	West to North (1)	West to North (2)	West to South (2)
North	West to North (1)	West to South (1)	South to East	North to East	West to North (2)	West to South (2)
North	North to East	South to East	West to South (2)	West to North (1)	West to North (2)	West to South (1)
North	West to North (1)	West to South (2)	South to East	North to East	West to North (2)	West to South (1)
South	West to South (1)	West to South (2)	South to East	North to East	West to North (1)	West to North (2)
South	West to South (2)	West to South (1)	South to East	North to East	West to North (1)	West to North (2)
North	North to East	South to East	West to South (1)	West to North (2)	West to North (1)	West to South (2)
North	West to North (2)	West to South (1)	South to East	North to East	West to North (1)	West to South (2)
North	North to East	South to East	West to South (2)	West to North (2)	West to North (1)	West to South (1)
North	West to North (2)	West to South (2)	South to East	North to East	West to North (1)	West to South (1)
North	West to North (1)	West to South (1)	West to South (2)	West to North (2)	North to East	South to East

land.name

b1.name

b2.name

b3.name

b4.name

b5.name

b6.name

south

West to South (1)

West to South (2)

South to East

North to East

West to North (2)

West to North (1)

South	West to South (2)	West to North (1)	West to North (2)	West to South (1)	South to East	North to East
South	West to South (2)	West to North (2)	West to North (1)	West to South (1)	South to East	North to East
South	South to East	North to East	West to North (1)	West to South (1)	West to South (2)	West to North (2)
South	West to South (1)	West to North (1)	North to East	South to East	West to South (2)	West to North (2)
South	South to East	North to East	West to North (2)	West to South (1)	West to South (2)	West to North (1)
South	West to South (1)	West to North (2)	North to East	South to East	West to South (2)	West to North (1)
North	West to North (1)	West to North (2)	North to East	South to East	West to South (2)	West to South (1)
North	West to North (2)	West to North (1)	North to East	South to East	West to South (2)	West to South (1)
South	South to East	North to East	West to North (1)	West to South (2)	West to South (1)	West to North (2)
South	West to South (2)	West to North (1)	North to East	South to East	West to South (1)	West to North (2)
South	South to East	North to East	West to North (2)	West to South (2)	West to South (1)	West to North (1)
South	West to South (2)	West to North (2)	North to East	South to East	West to South (1)	West to North (1)
North	West to North (1)	West to North (2)	North to East	South to East	West to South (1)	West to South (2)
North	West to North (2)	West to North (1)	North to East	South to East	West to South (1)	West to South (2)

```
MATCH (:Land)-[bridges:BRIDGE*6]-(:Land)
RETURN EXTRACT(b in bridges | b.name);
```







[West to South (2), West to North (2), West to North (1), West to South (1), South to East, North to East]
[West to South (2), West to North (1), West to North (2), West to South (1), South to East, North to East]
[West to South (1), West to North (2), West to North (1), West to South (2), South to East, North to East]
[West to South (1), West to North (1), West to North (2), West to South (2), South to East, North to East]
[South to East, North to East, West to North (2), West to South (2), West to South (1), West to North (1)]
[West to South (2), West to North (2), North to East, South to East, West to South (1), West to North (1)]
[South to East, North to East, West to North (2), West to South (1), West to South (2), West to North (1)]
[West to South (1), West to North (2), North to East, South to East, West to South (1), West to North (1)]
[West to South (2), West to North (1), North to East, South to East, West to South (1), West to North (2)]
[South to East, North to East, West to North (1), West to South (1), West to South (2), West to North (2)]
[West to South (1), West to North (1), North to East, South to East, West to South (2), West to North (2)]
[West to South (2), West to South (1), South to East, North to East, West to North (1), West to North (2)]
[West to South (1), West to South (2), South to East, North to East, West to North (1), West to North (2)]

[illegible]

# PHP?

# Neo4j-PHP-Client

<https://github.com/graphaware/neo4j-php-client>

```
composer require graphaware/neo4j-php-client
```



# Neo4j PHP OGM

<https://github.com/graphaware/neo4j-php-ogm>

```
composer require graphaware/neo4j-php-ogm
```

# Neo4j Bolt PHP

<https://github.com/graphaware/neo4j-bolt-php>

```
composer require graphaware/neo4j-bolt
```

# Create the Bolt client

```
$client = \GraphAware\Neo4j\Client\ClientBuilder::create( )  
    ->addConnection(  
        'bolt',  
        'bolt://neo4j:password@localhost:7687'  
    )  
    ->build( );
```

# Or REST, if you must

```
$client = \GraphAware\Neo4j\Client\ClientBuilder::create( )  
    ->addConnection(  
        'default',  
        'http://neo4j:password@localhost:7474'  
    )  
    ->build( );
```

# Look up Bob's orders

```
$result = $client->run(  
    'MATCH (customer:Customer {email: { email }})-[:PLACED]->  
        (order)-[:CONTAINS]->(product)  
    RETURN customer.name, order.date,  
        COLLECT(product.name) AS product_list',  
    [ 'email' => 'bob@example.com' ]  
);
```

# List Bob's orders

```
foreach ( $result->getRecords( ) as $record ) {  
    echo sprintf(  
        "%s ordered [%s] on %s\n",  
        $record->get( 'customer.name' ),  
        implode( ', ', $record->get( 'product_list' ) ),  
        $record->get( 'order.date' )  
    );  
}
```



```
$ php neo4j-client-example.php  
Bob ordered [Squeaky dog toy, Dog biscuits] on 2016-02-05
```

# OGM

```
use GraphAware\Neo4j\OGM\Annotations as OGM;

/** @OGM\Node(label="Customer") */
class Customer {
    /** @OGM\GraphId() */
    public $id;
    /** @OGM\Property(type="string") */
    public $email;
    /** @OGM\Property(type="string") */
    public $name;
    /** @OGM\Relationship(type="PLACED", direction="OUTGOING",
        *      targetEntity="Order", mappedBy="customer",
        *      collection=true) */
    public $orders;
}
```

```
use GraphAware\Neo4j\OGM\Annotations as OGM;

/** @OGM\Node(label="Order") */
class Order {
    /** @OGM\GraphId() */
    public $id;
    /** @OGM\Property(type="string") */
    public $date;
    /** @OGM\Relationship(type="CONTAINS", direction="OUTGOING",
     *      targetEntity="Product", collection=true) */
    public $products;
    /** @OGM\Relationship(type="PLACED", direction="INCOMING",
     *      targetEntity="Customer", collection=false) */
    public $customer;
}
```

```
use GraphAware\Neo4j\OGM\Annotations as OGM;

/** @OGM\Node(label="Product") */
class Product {
    /** @OGM\GraphId() */
    public $id;
    /** @OGM\Property(type="string") */
    public $date;
    /** @OGM\Property(type="string") */
    public $description;
    /** @OGM\Property(type="string") */
    public $name;
    /** @OGM\Property(type="int") */
    public $price;
    /** @OGM\Property(type="int") */
    public $productId;
}
```

# Create the EntityManager

```
$entityManager = \GraphAware\Neo4j\OGM\EntityManager::create(  
    'bolt://neo4j:password@localhost:7687'  
);
```



# Find Bob

```
$customer = $entityManager  
    ->getRepository(Customer::class)  
    ->findOneBy( ['email' => 'bob@example.com'] );
```

```
foreach ( $customer->orders as $order ) {  
    $products = array_reduce(  
        $order->products->toArray( ),  
        function ( $productNames, $product ) {  
            $productNames[] = $product->name;  
            return $productNames;  
        }  
    );  
    echo sprintf(  
        "%s ordered [%s] on %s\n",  
        $customer->name, implode( ' , ', $products ), $order->date  
    );  
}
```

```
$ php neo4j-ogm-example.php  
Bob ordered [Squeaky dog toy, Dog biscuits] on 2016-02-05
```

# Reference Material

- Official Documentation
  - <https://neo4j.com/docs/stable>
- Cypher Refcard
  - <https://neo4j.com/docs/stable/cypher-refcard>
- Online Training
  - <https://neo4j.com/graphacademy/online-training>
- Neo4j Certification
  - <https://neo4j.com/graphacademy/neo4j-certification>
- Neo4j Bookshelf
  - <https://neo4j.com/books/>
- GraphGists
  - <https://neo4j.com/graphgists>
- Slack
  - <https://neo4j-users-slack-invite.herokuapp.com>
- Twitter
  - <https://twitter.com/neo4j>

**USE THE RIGHT TOOL  
FOR THE JOB**

**<https://joind.in/talk/5b436>**

(in case you missed it...)





```
MATCH (:Attendee)-[:HAS]->(q:Question)
RETURN q.text;
```

# Thanks!