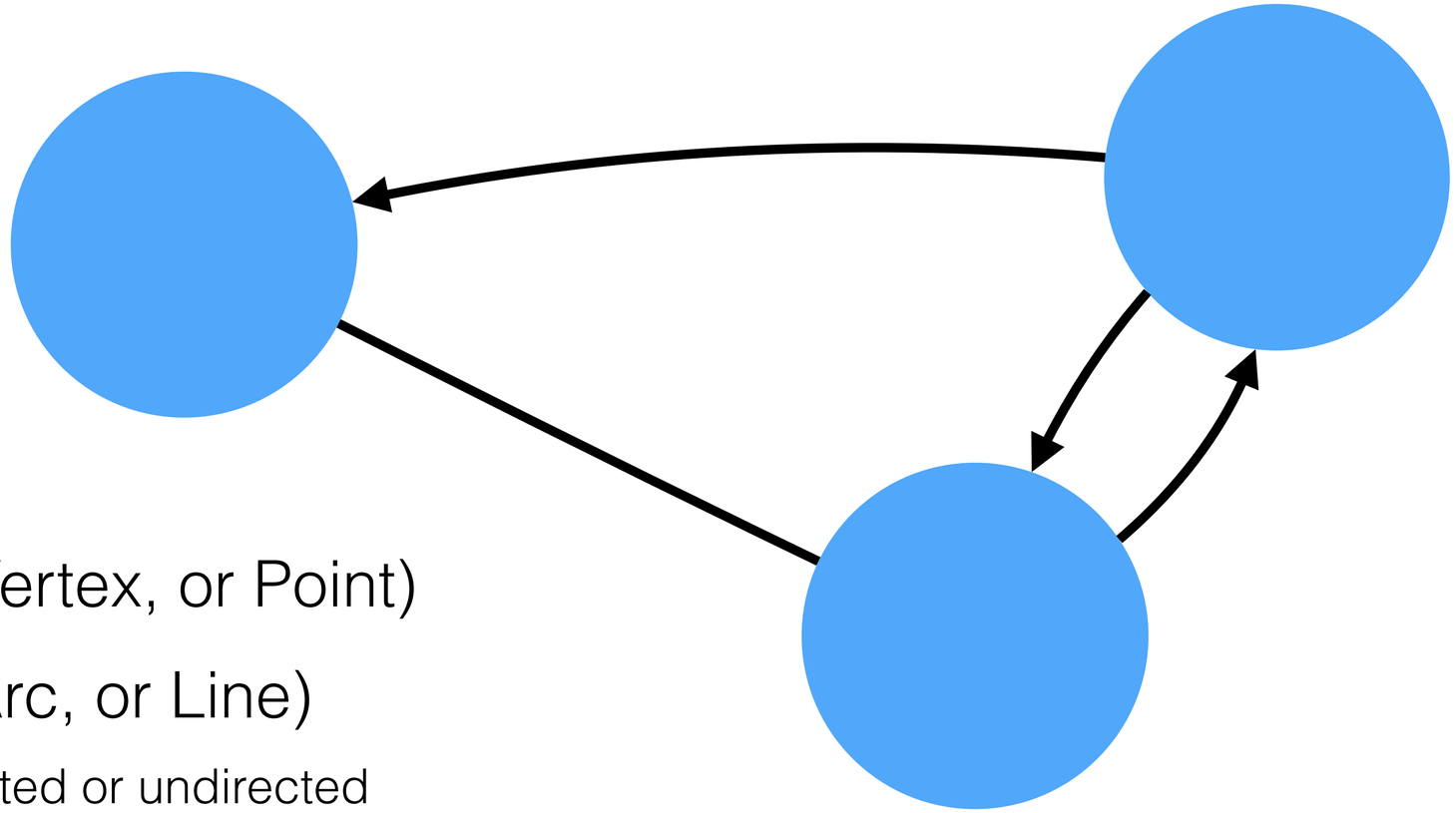
Introduction to Graph Databases with Neo4j

Michael Moussa

What is a Graph?

Structure that models relationships between objects



Node (or Vertex, or Point)

Edge (or Arc, or Line)

*Can be directed or undirected

Graph Theory

Study of graphs and their applications

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

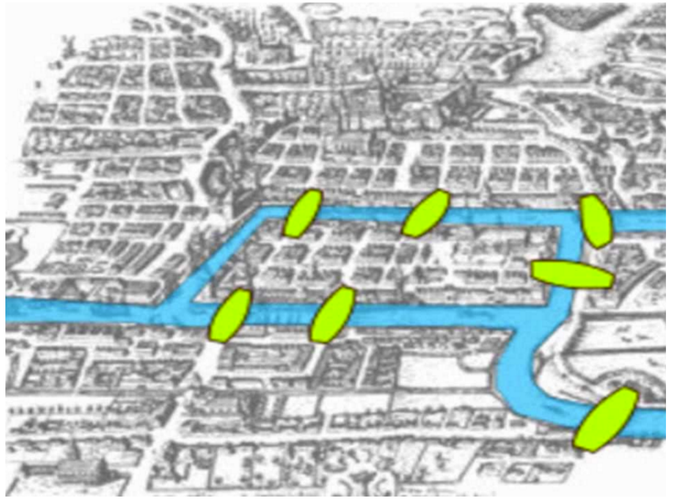
Leonhard Euler (1707-1783) considered the "father" of graph theory

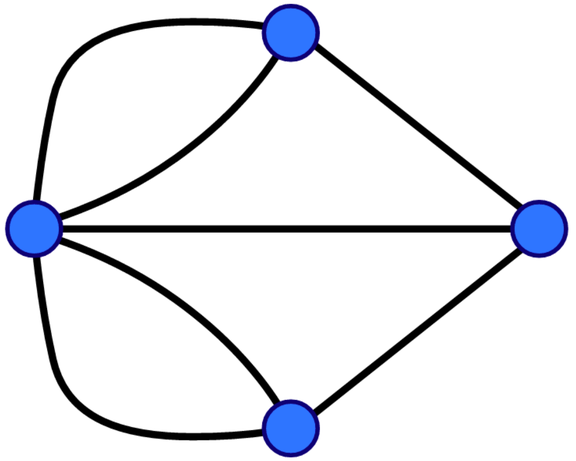
Seven Bridges of Königsberg

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

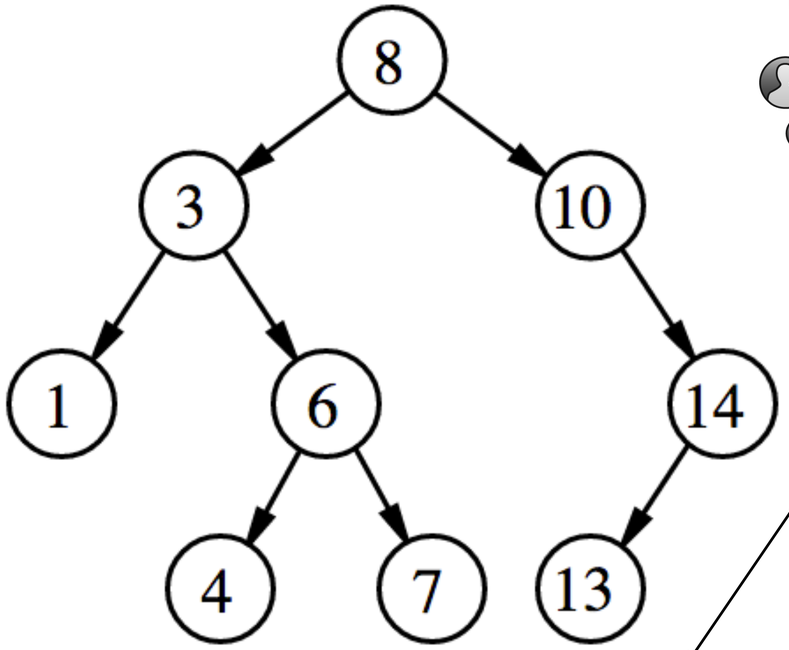
Euler proved in 1736 that the answer is "No".

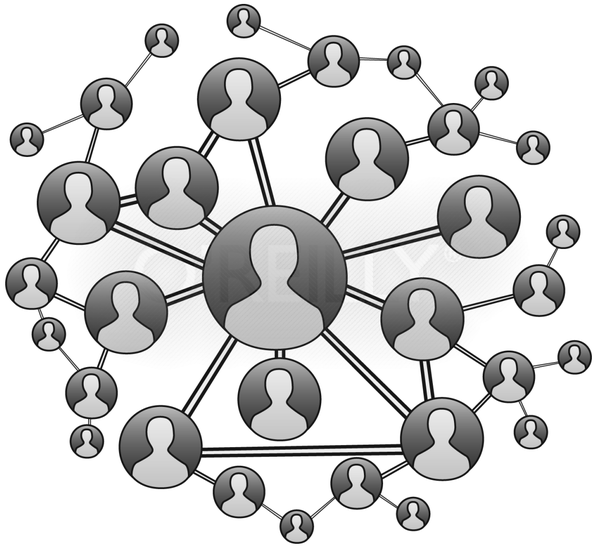
His solution is the first theorem of graph theory.

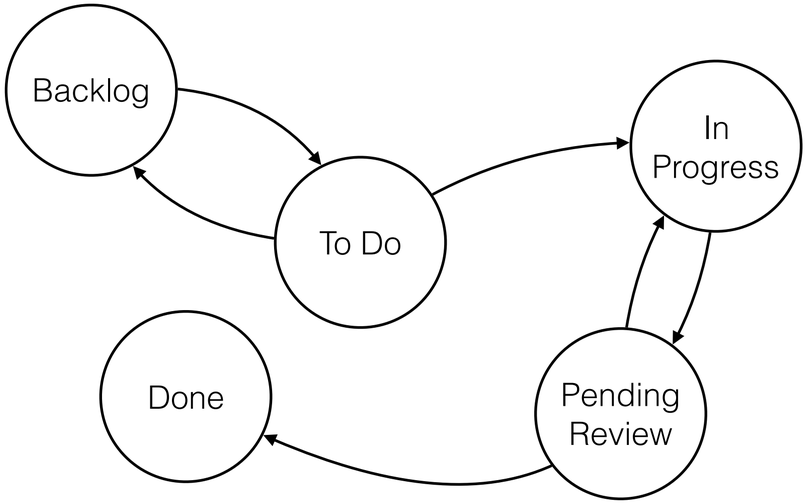


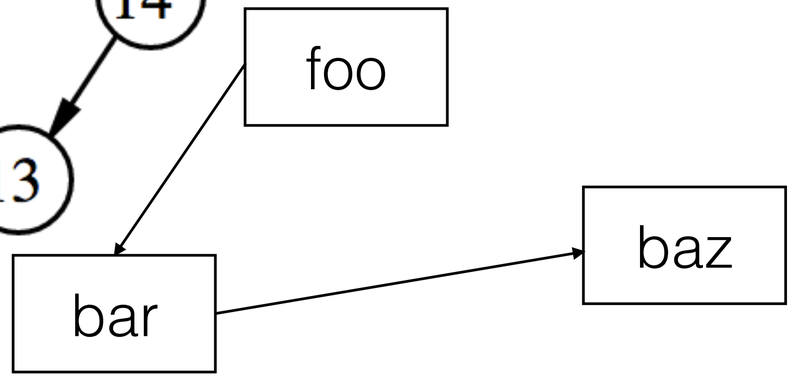


Graphs are Everywhere!



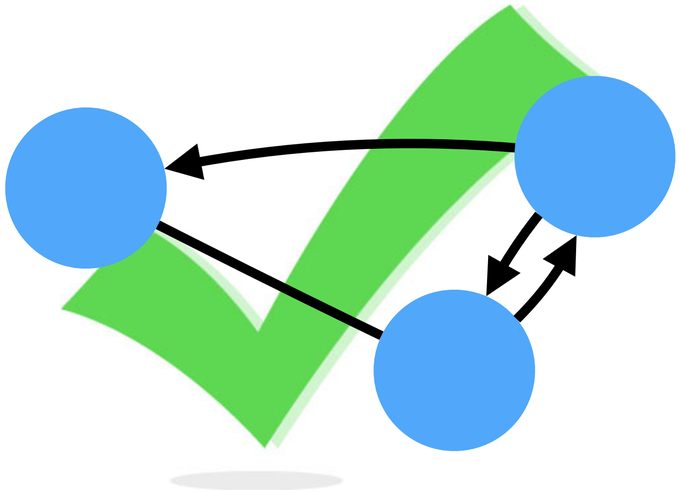






- Driving directions
- Production recommendations
- Financial fraud detection
- Identity and access management

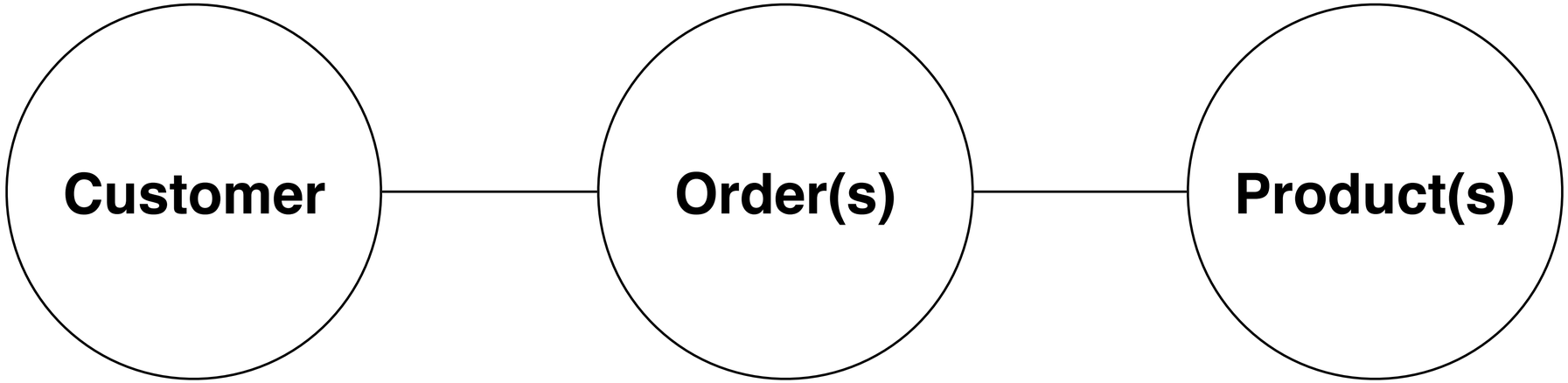
Graph Database



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

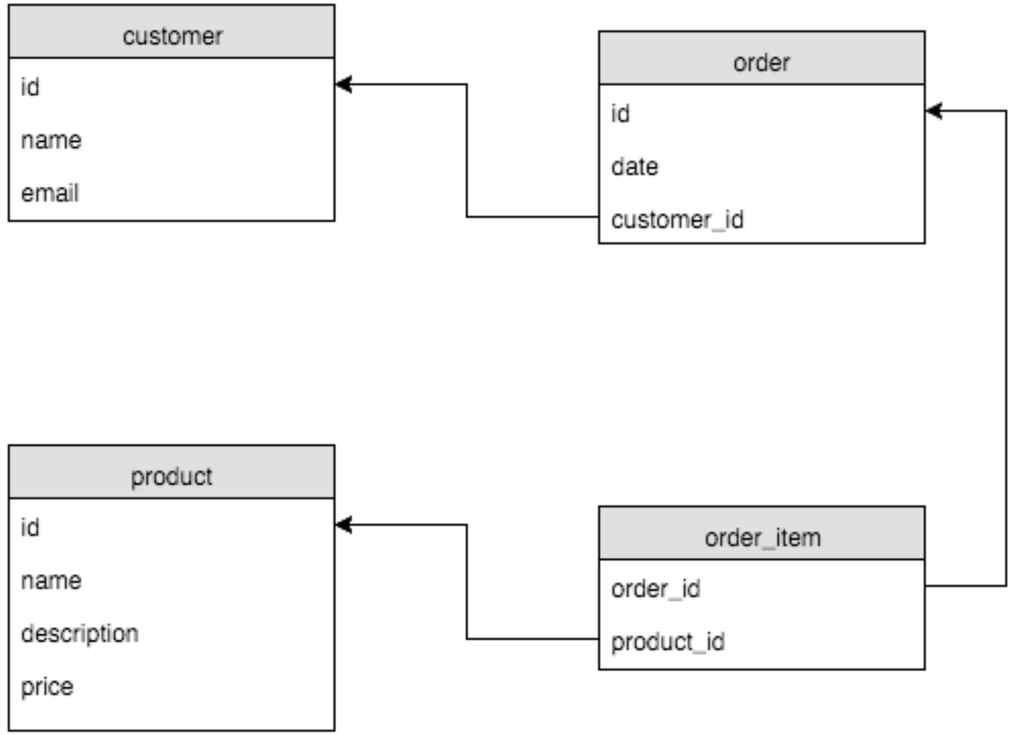
E-commerce Example

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



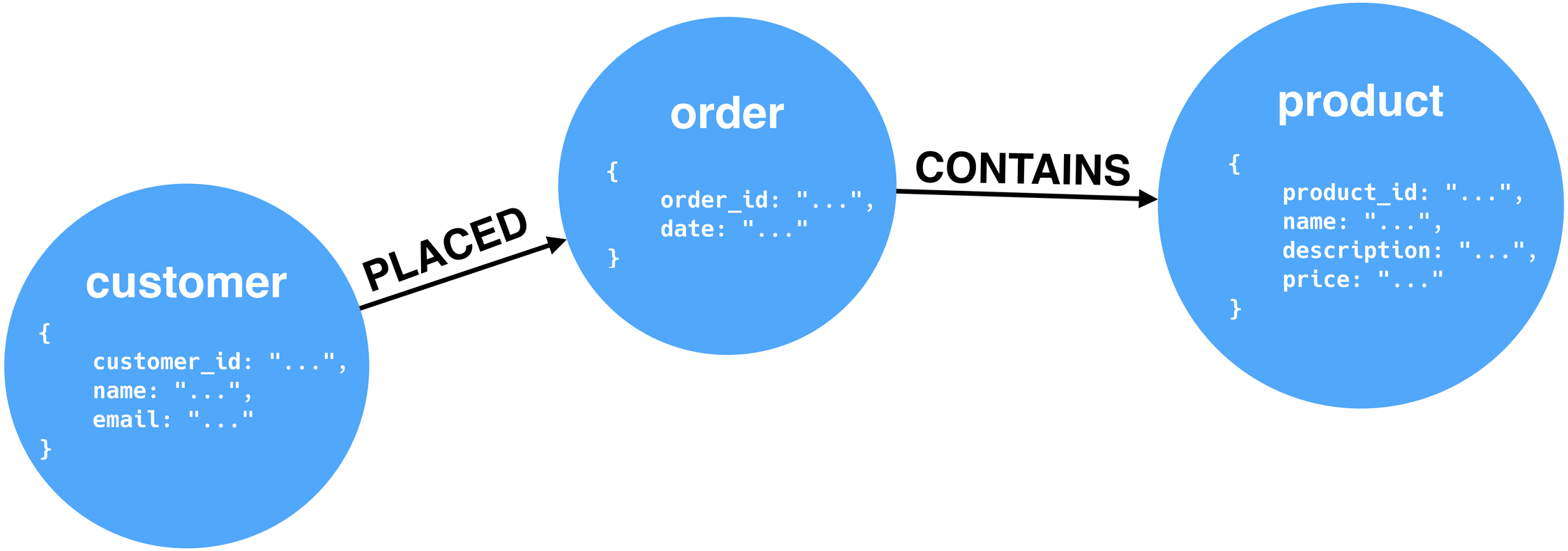
E-commerce Example

Modeling in a relational database



E-commerce Example

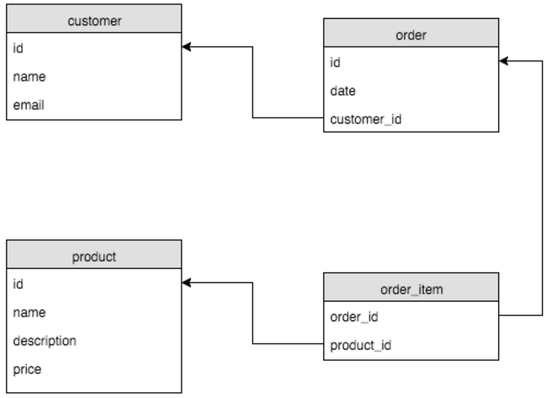
Modeling in a graph database



E-commerce Example

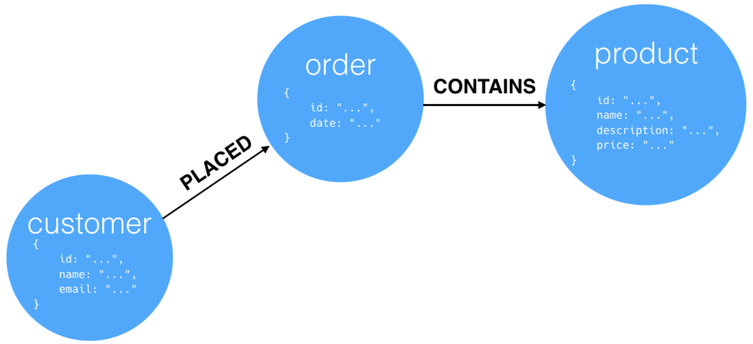
RDBMS

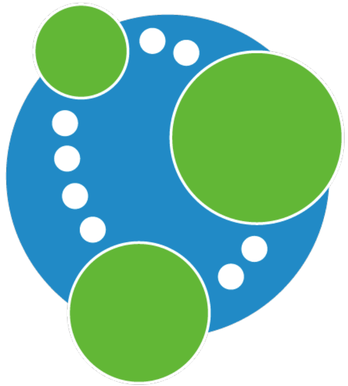
- Highly-structured
- Entities related via JOIN tables and foreign keys



Graph DB

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





neo4j

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**.

Capacity

Nodes: ~34 billion

Relationships: ~34 billion

Properties: 68 billion - 274 billion (depending on datatype)

Relationship types: 65,000

Capacity

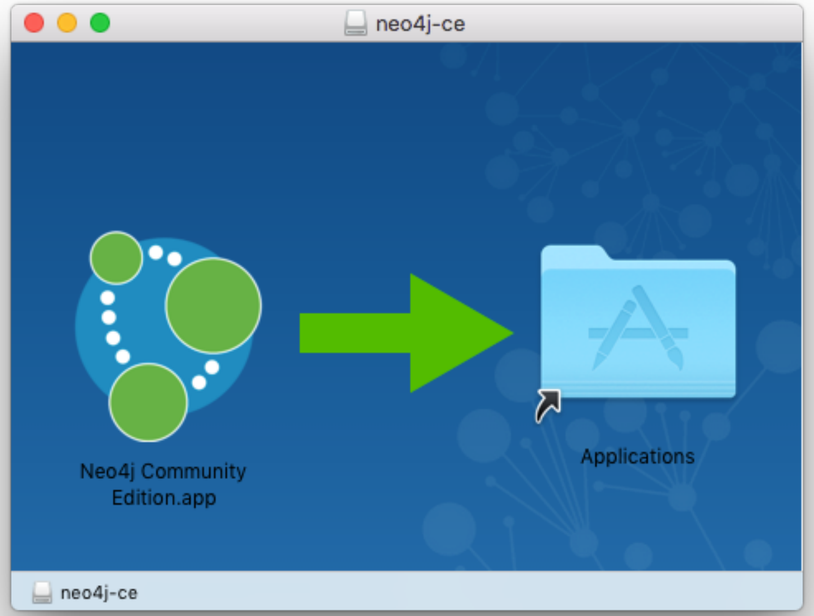
1,000,000,000,000,000 Nodes


onefinestay
ZEPHYR HEALTH INC
CISCO
53
gamesys
Global 500 Media
Global 500 Logistics
doximity
Linking Medical Minds
SNAP INTERACTIVE
mallowstreet
Lufthansa Systems
Walmart
WANDERU
TOMTOM
telenor
SharePractice
meetic
tallcat
AXON ACTIVE
Focusing on your decisions
webshots
wallpaper & screensaver
InfoJobs
zeebox
ebay
megree
Glowbl
DingLicom
Mapillary
Global 500 Manufacturing
CrunchBase
eHarmony
pitney bowes
Global 2000 Adidas
wazoku
classmates
hp
Hinge
Care.com
There for you
LIFECHURCH.TV
Aol.
UBS
maaii
Let's connect
shutl
entropy
CareerArcGroup
spring
UNIFIED
M
CenturyLink
NII
aikux.com
Curaspan HEALTH GROUP
ePals
<fuseworks/>
gen
dop
MONSANTO
ALERTLOGIC
Security. Compliance. Cloud.
Legacy.com
VIRTUAL INSTRUMENTS
ice
janssen
kitedesk
NexLP
OPEN Tree of Life
ThingWorx
A PTC Business
ECOLUTIS
PERIGEE
graphAlchemist
WILLERSOFT
Pro Integra S.L.
SERENA
SODIFRANCE
timeBlend!
Zenoss
LA BANQUE POSTALE
myWave
workjam
nine#
LinkedIn
die Bayerische
Accenture
StarHub
noble group
SMARTMONSTERS
Pondera
SUPPLYHUB
compete
EQUILAR
Spree
NATIONAL GEOGRAPHIC
Livestation
NMBS
SNCB
KiwiRail
3
Three.com
YELAGO
wayblazer
HealthUnlocked
PERIGEE

Cypher

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

Installing Neo4j



http://neo4j.com/download/

```
// 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
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

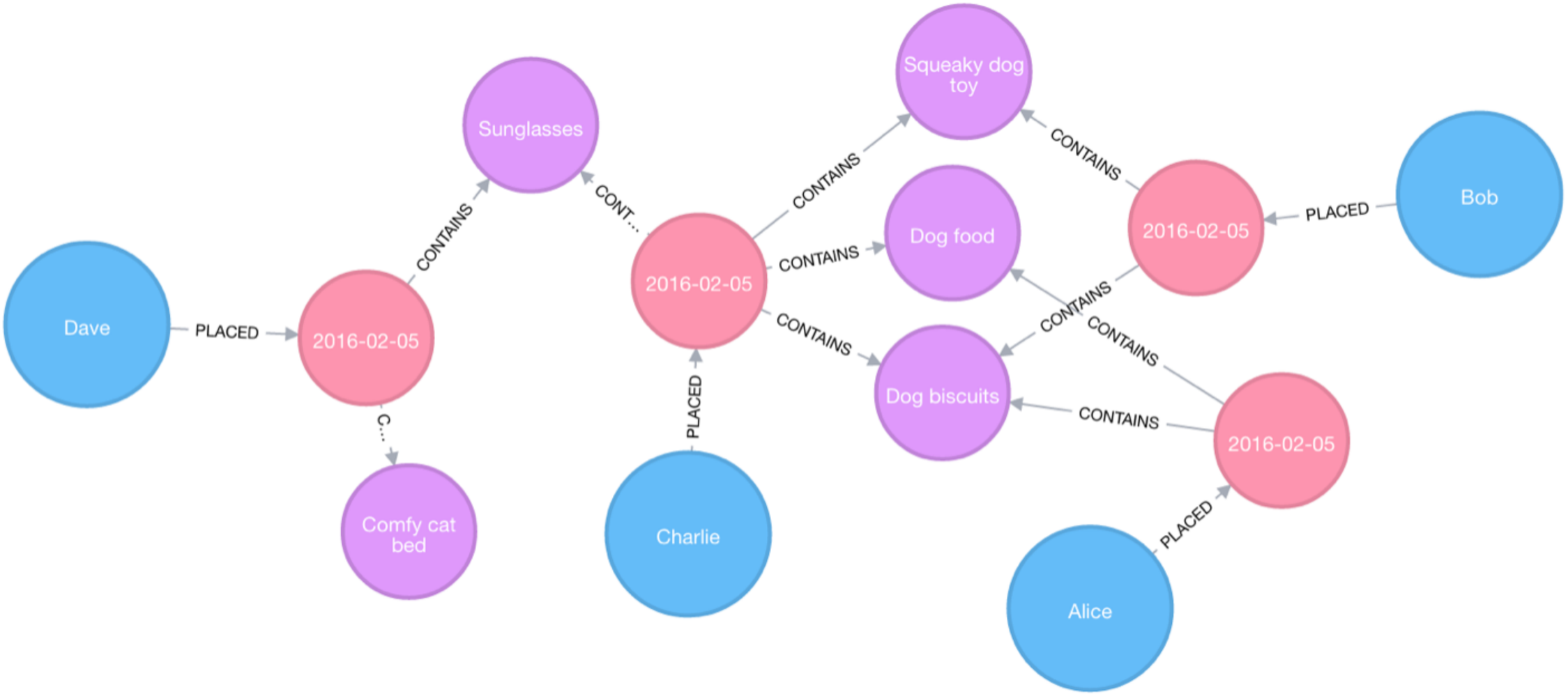
```
// Charlie orders dog food, dog biscuits,
// a squeaky dog toy, and sunglasses
MATCH (customer:Customer {email: "charlie@example.com"}),
      (product:Product)
WHERE product.product_id IN [12, 34, 56, 90]
MERGE (customer)-[:PLACED]->(order:Order {date: "2016-02-05"})
MERGE (order)-[:CONTAINS]->(product);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

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

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


Dave
PLACED
2016-02-05
CONTAINS
Sunglasses
C...
Comfy cat bed
CONT...
2016-02-05
PLACED
Charlie
CONTAINS
Squeaky dog toy
CONTAINS
Dog food
CONTAINS
Dog biscuits
CONTAINS
Squeaky dog toy
CONTAINS
2016-02-05
PLACED
Bob
CONTAINS
Dog food
CONTAINS
Dog biscuits
CONTAINS
2016-02-05
PLACED
Alice

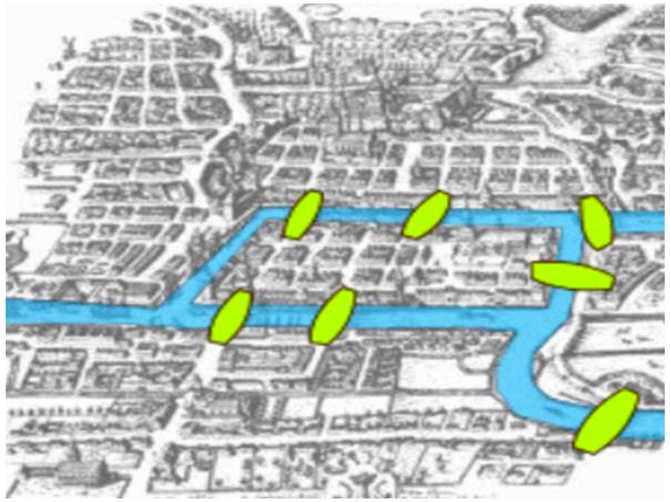
```
// Someone is browsing the dog biscuit item (product_id: 34)
// What else should we recommend that they buy?
MATCH (order:Order)-[:CONTAINS]->(product:Product {product_id: 34}),
      (order)-[:CONTAINS]->(recommendation:Product)
WHERE NOT recommendation = product
RETURN recommendation.name,
   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

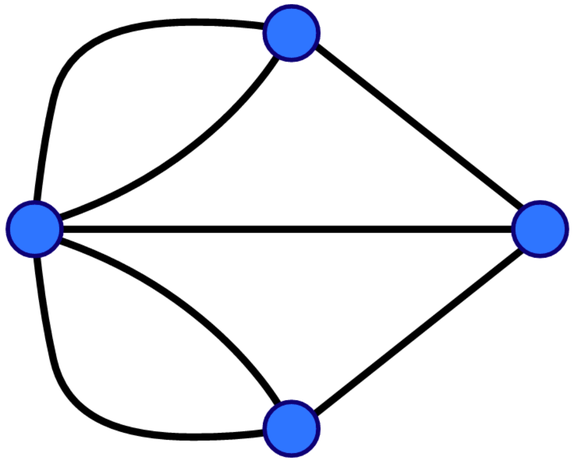
recommendation.name	recommended
Dog food	2
Squeaky dog toy	2
Sunglasses	1

```
// 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?

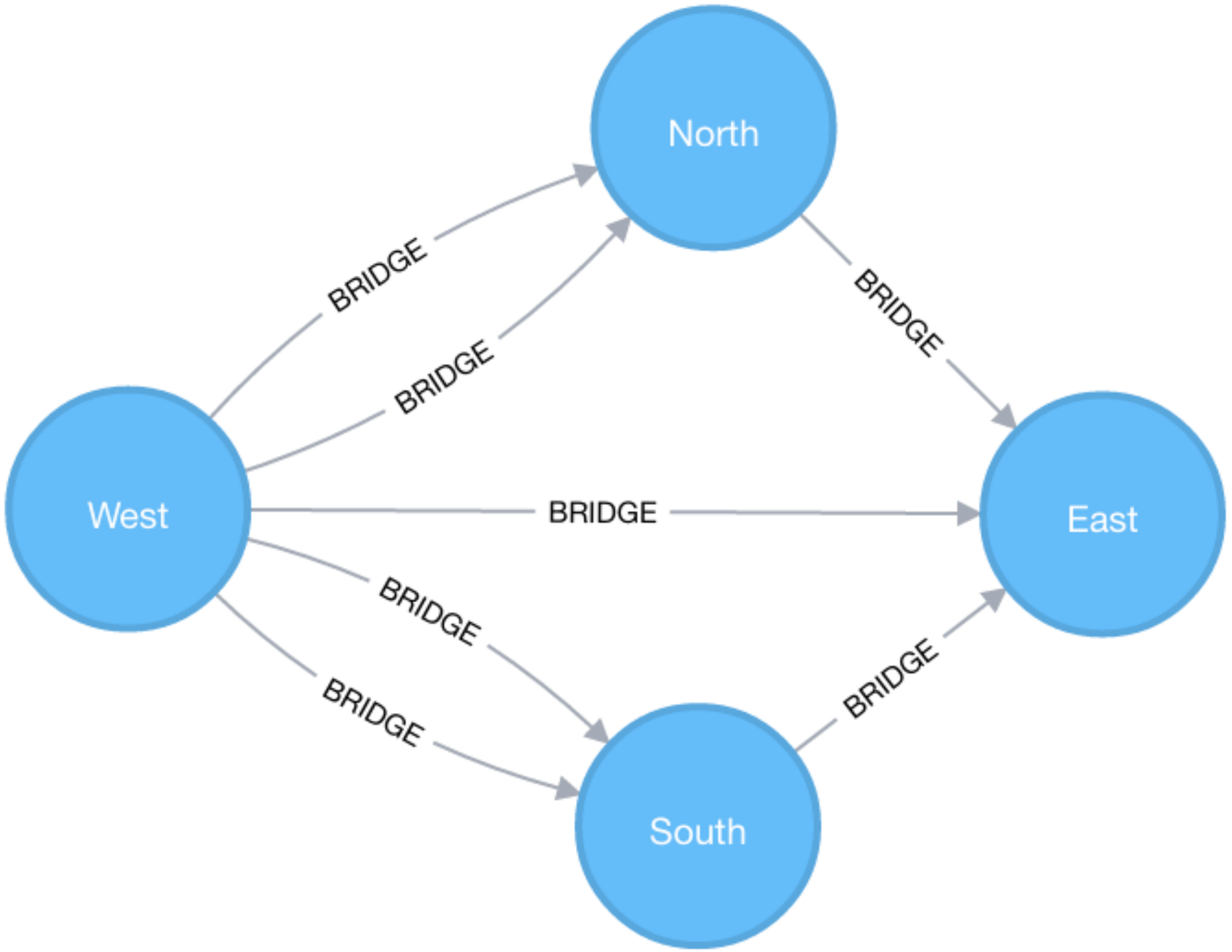




```
CREATE (west:Land {name: "West"}), (north:Land {name: "North"}),
       (east:Land {name: "East"}), (south:Land {name: "South"}),

       (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;
```

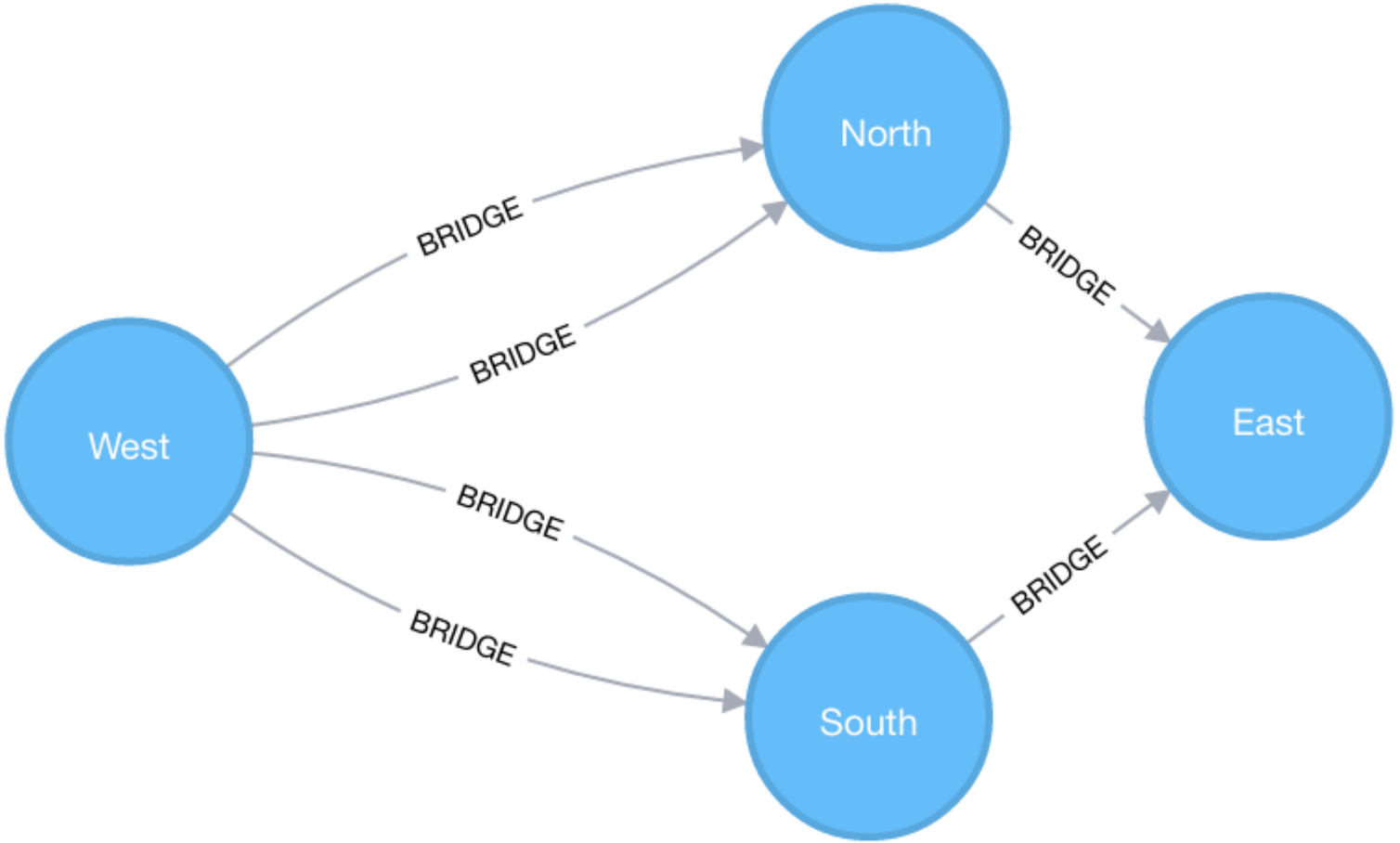

North
BRIDGE
BRIDGE
BRIDGE
West
BRIDGE
East
BRIDGE
BRIDGE
BRIDGE
South

```
// Since the Seven Bridges of Königsberg has no solution,
// the following query will return no rows.
MATCH (:Land)-[b1:BRIDGE]-(:Land)-[b2:BRIDGE]-
      (:Land)-[b3:BRIDGE]-(:Land)-[b4:BRIDGE]-
      (:Land)-[b5:BRIDGE]-(:Land)-[b6:BRIDGE]-
      (:Land)-[b7:BRIDGE]-(:Land)
RETURN b1, b2, b3, b4, b5, b6, b7;
```

(no rows)

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

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


North
BRIDGE
BRIDGE
BRIDGE
West
BRIDGE
East
BRIDGE
South

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

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

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
```

It's a REST API behind the scenes...

```
$ curl --header "Authorization: Basic <.....>" http://localhost:7474/db/data/
{
  "extensions" : { },
  "node" : "http://localhost:7474/db/data/node",
  "node_index" : "http://localhost:7474/db/data/index/node",
  "relationship_index" : "http://localhost:7474/db/data/index/relationship",
  "extensions_info" : "http://localhost:7474/db/data/ext",
  "relationship_types" : "http://localhost:7474/db/data/relationship/types",
  "batch" : "http://localhost:7474/db/data/batch",
  "cypher" : "http://localhost:7474/db/data/cypher",
  "indexes" : "http://localhost:7474/db/data/schema/index",
  "constraints" : "http://localhost:7474/db/data/schema/constraint",
  "transaction" : "http://localhost:7474/db/data/transaction",
  "node_labels" : "http://localhost:7474/db/data/labels",
  "neo4j_version" : "2.3.2"
}
```

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

```php
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

```php
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;
}
```

```php
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;
}
```

```php
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

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

Find Bob

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

```php
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
  - http://neo4j.com/docs/stable
- Cypher Refcard
  - http://neo4j.com/docs/stable/cypher-refcard
- Online Training
  - http://neo4j.com/graphacademy/online-training
- Neo4j Certification
  - http://neo4j.com/graphacademy/neo4j-certification
- Neo4j Bookshelf
  - https://neo4j.com/books/
- GraphGists
  - http://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/d9ead

(in case you missed it...)

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

THANKS!