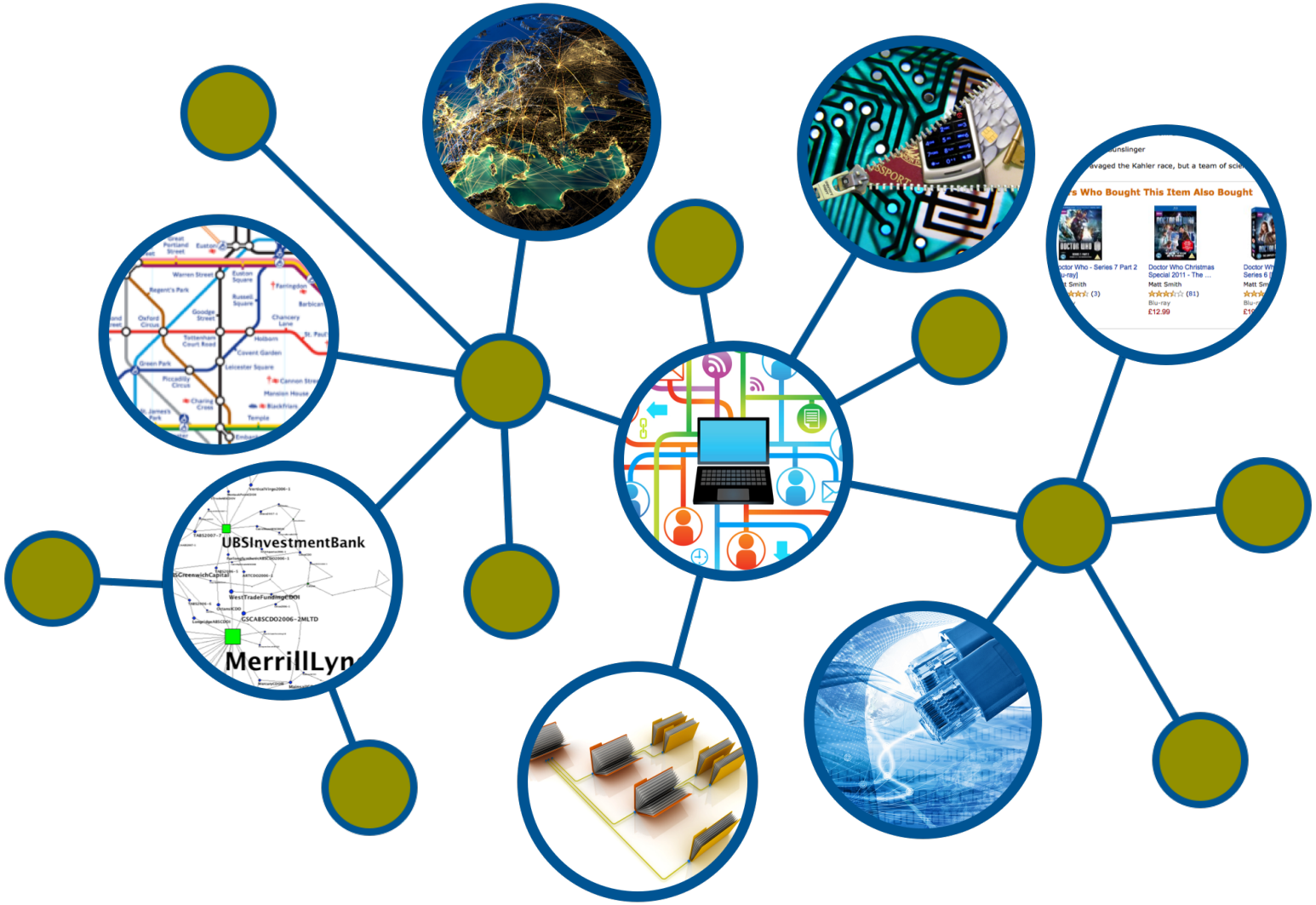
Let Me Graph That For You

@ianSrobinson
ian@neotechnology.com

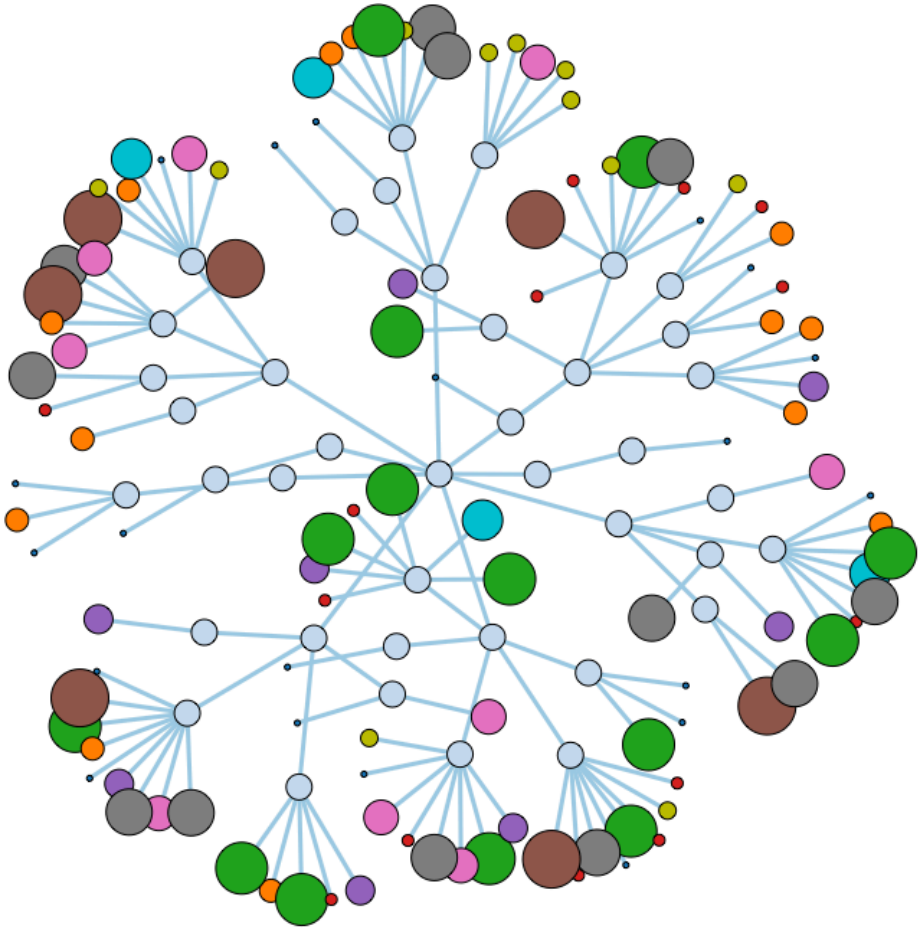
@neo4j

complexity = f(size, variable structure, connectedness)



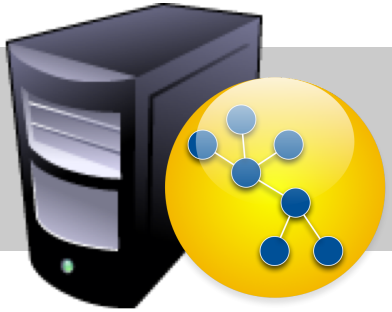
@neo4j

Graphs Are Everywhere

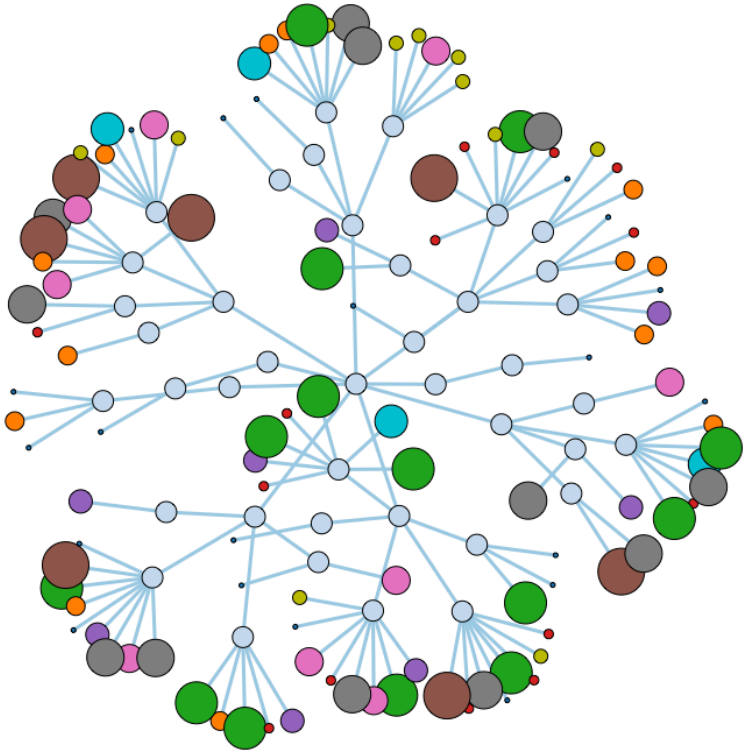


@neo4j

Graph Databases

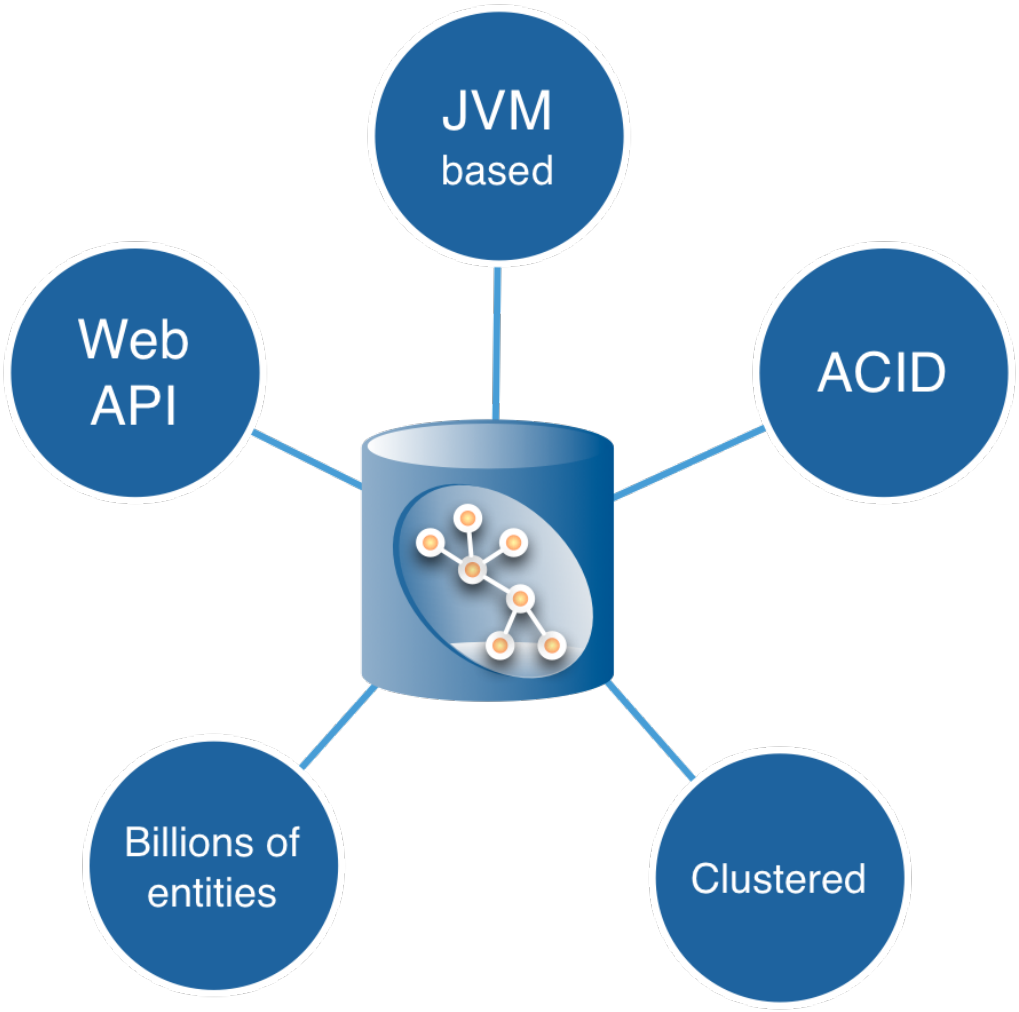


- Store
- Manage
- Query



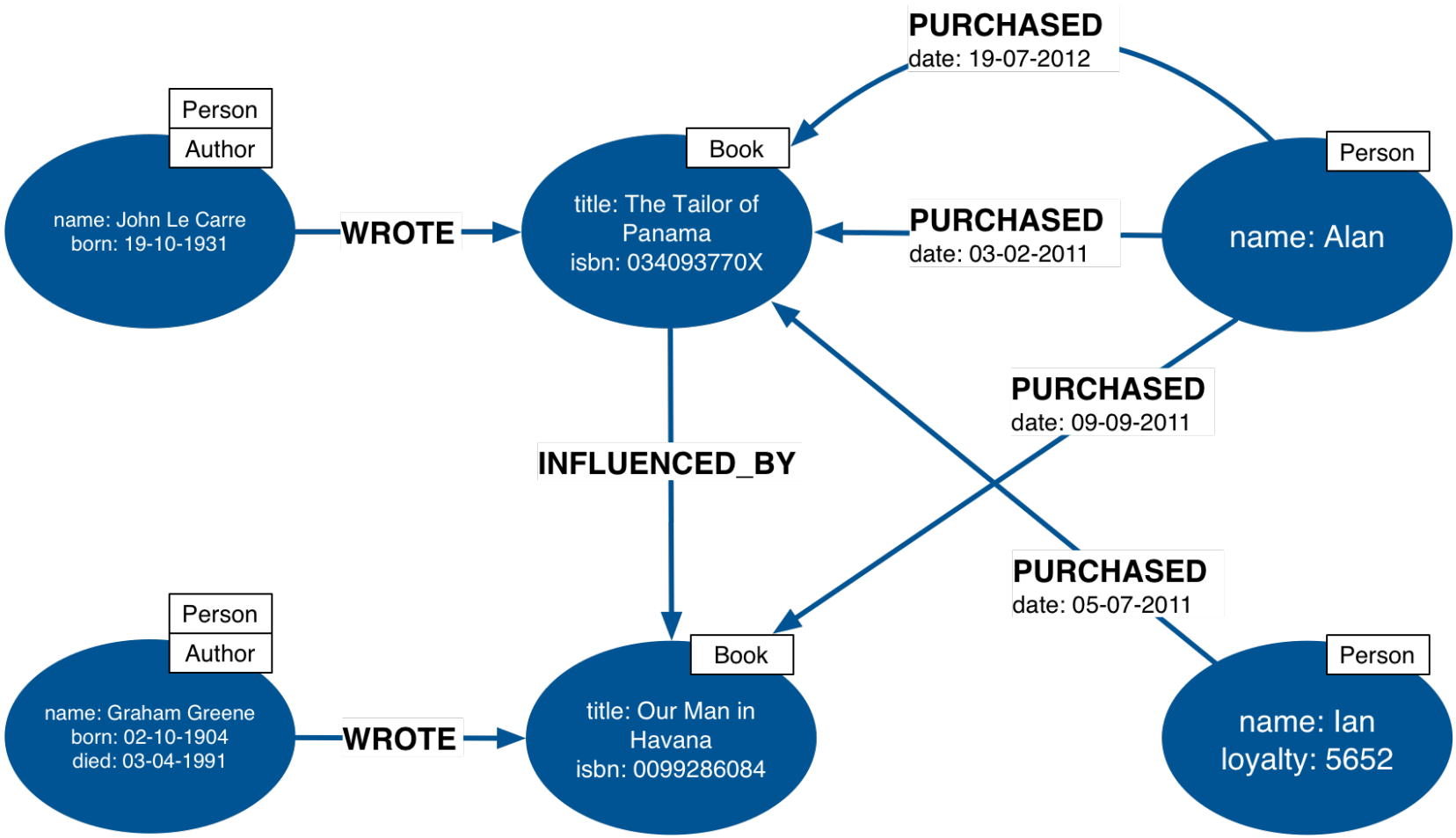
@neo4j

Neo4j is a Graph Database



@neo4j

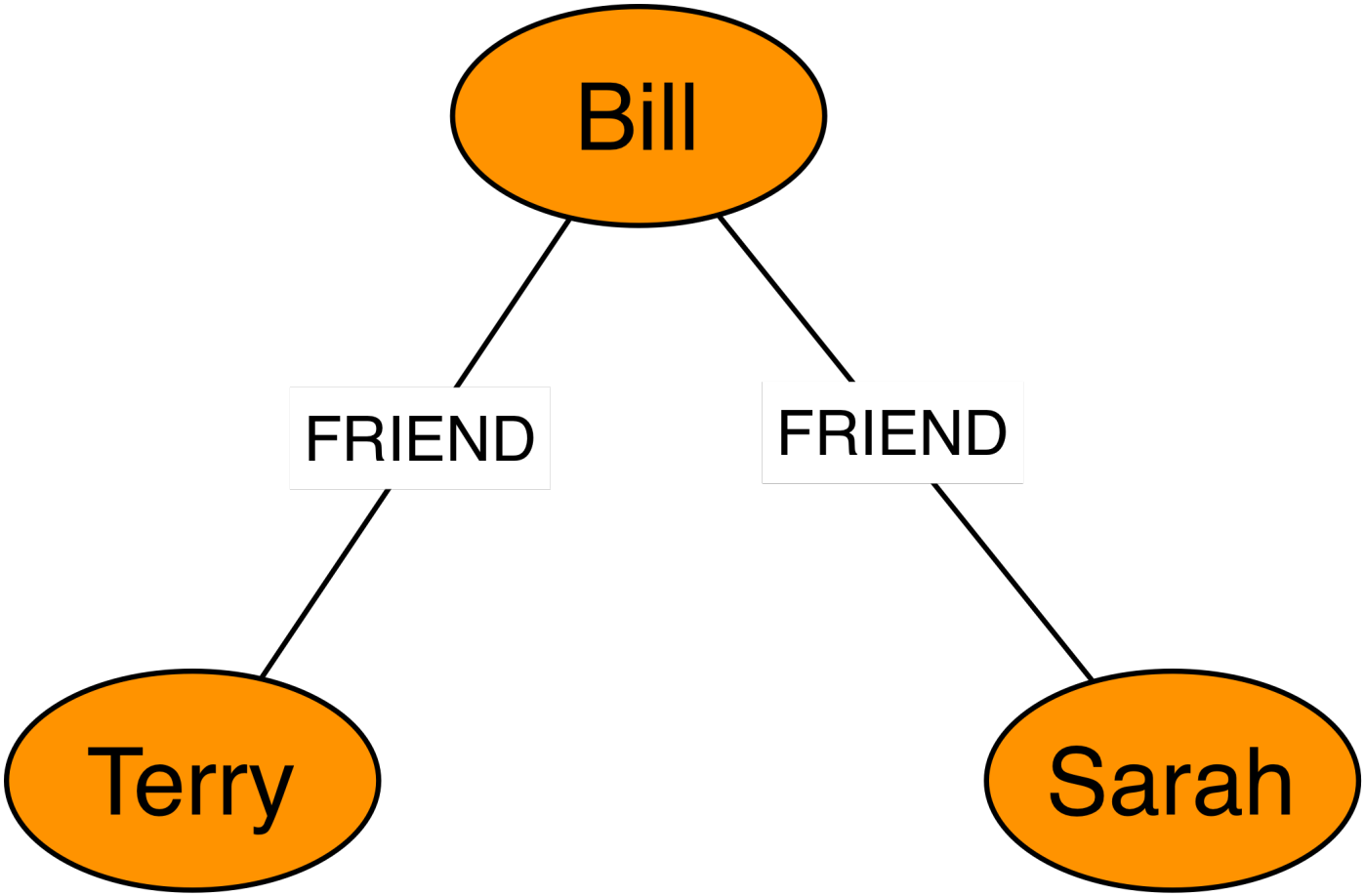
Labeled Property Graph



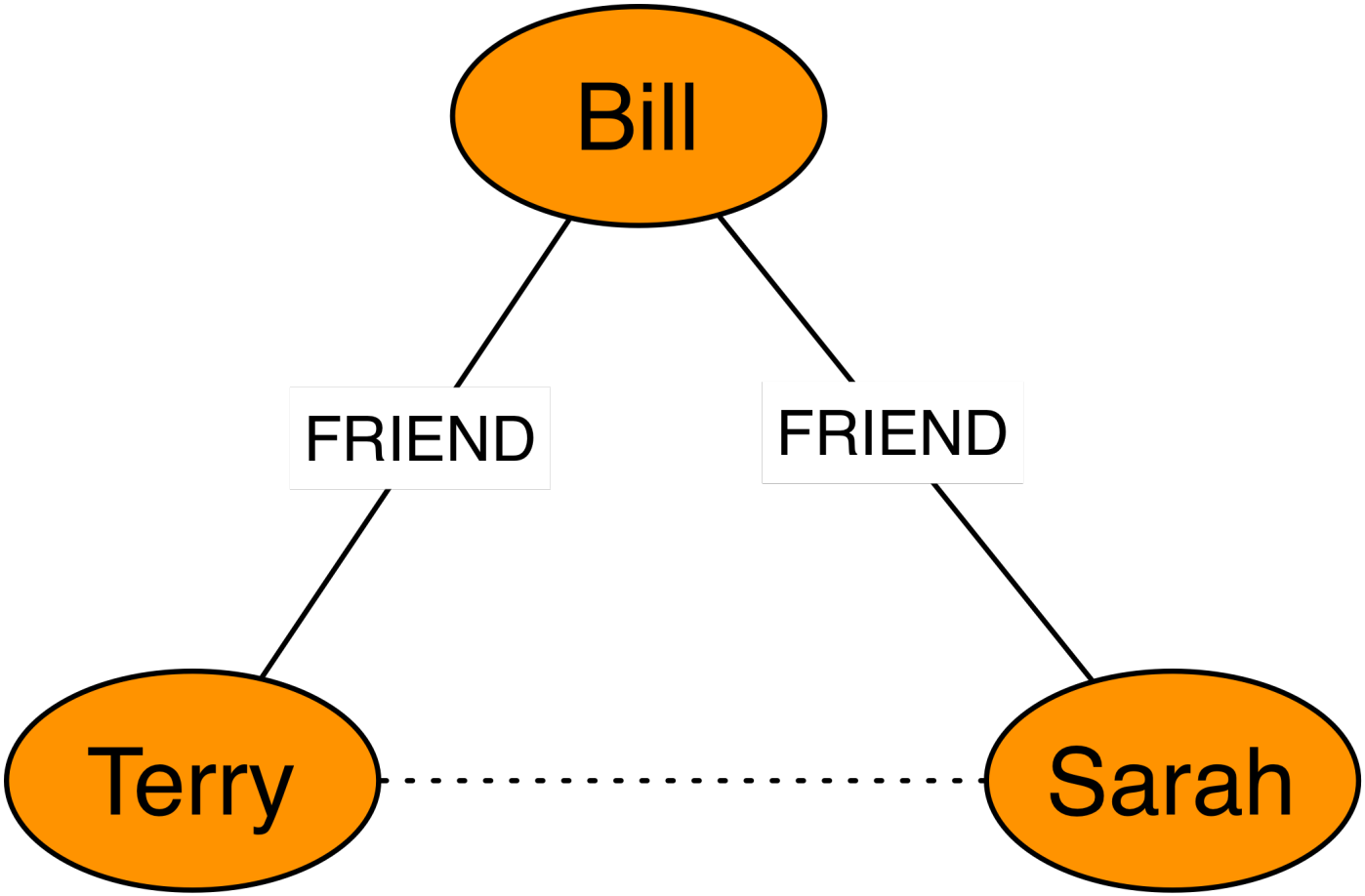
@neo4j

Making Connections

Triadic Closure – Closing Triangles

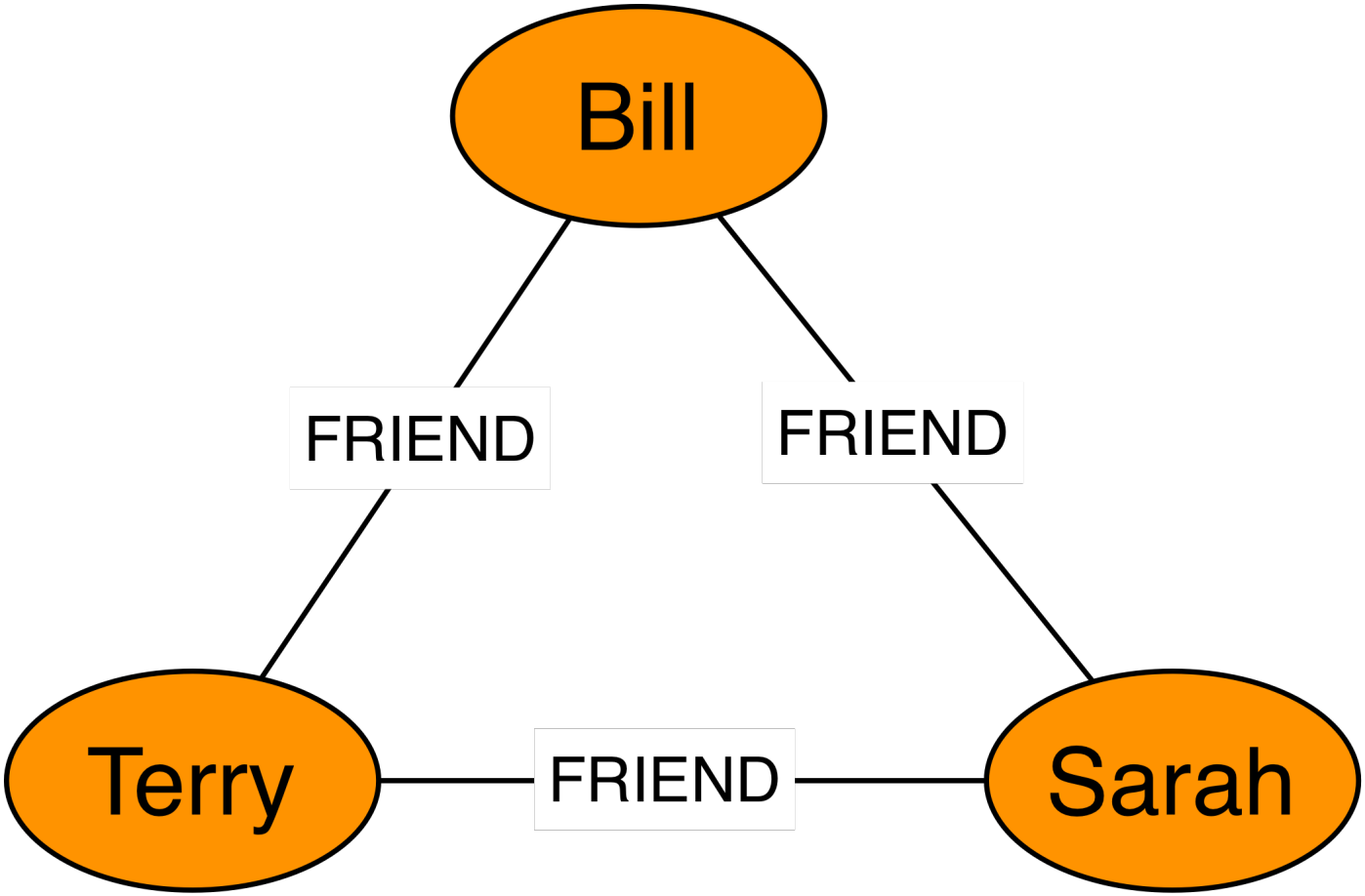


Triadic Closure – Closing Triangles

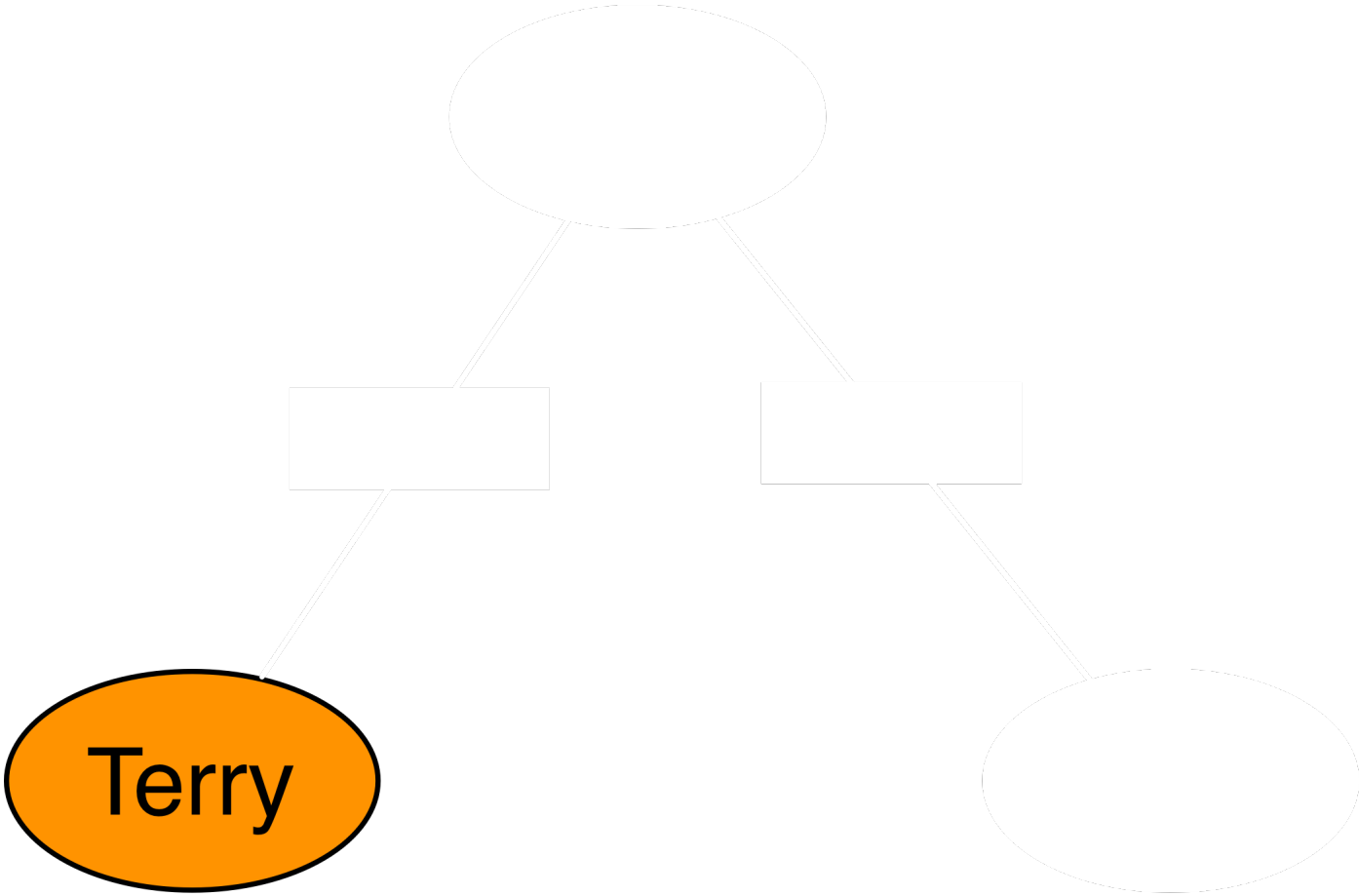


@neo4j

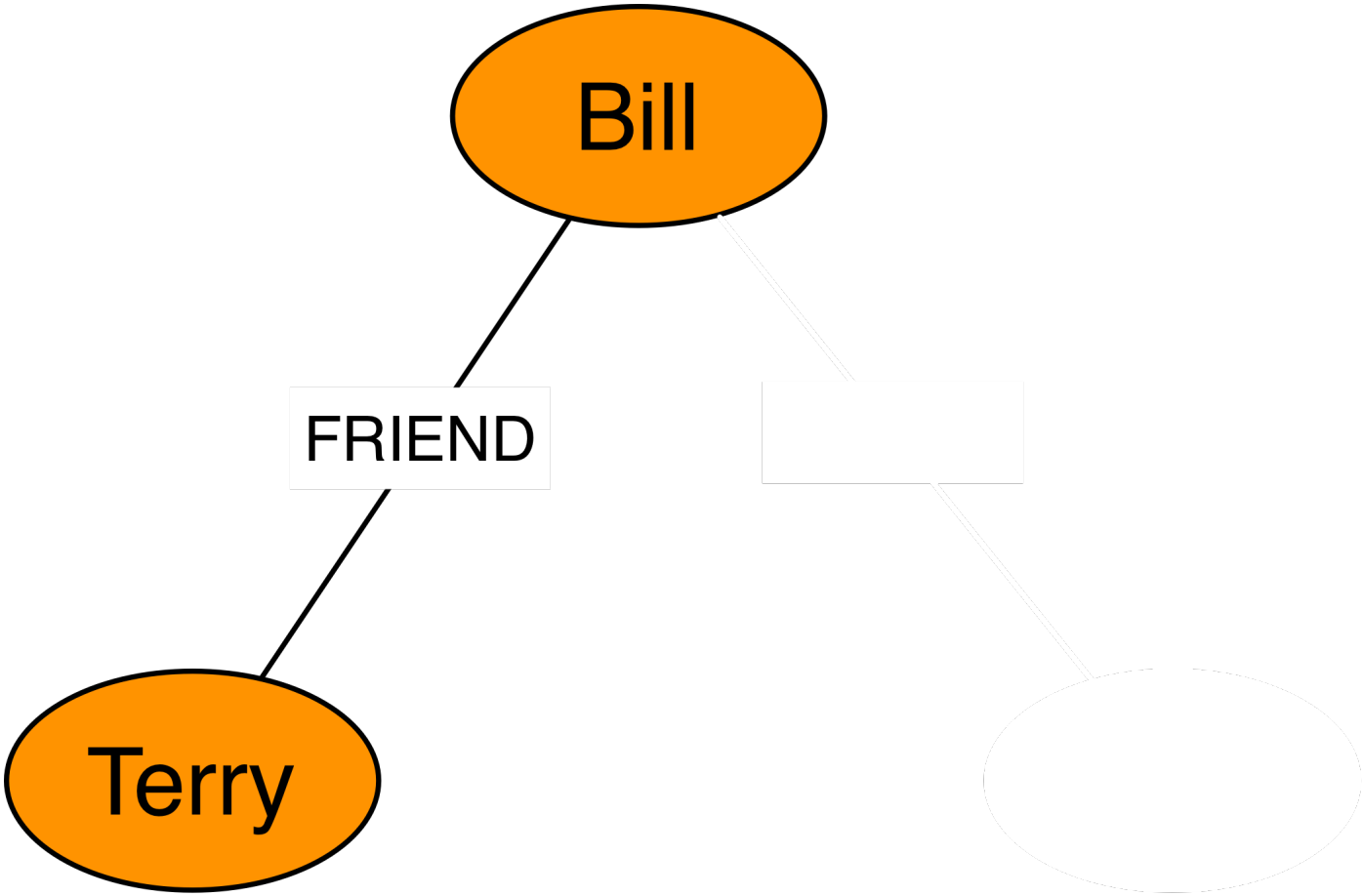
Triadic Closure – Closing Triangles



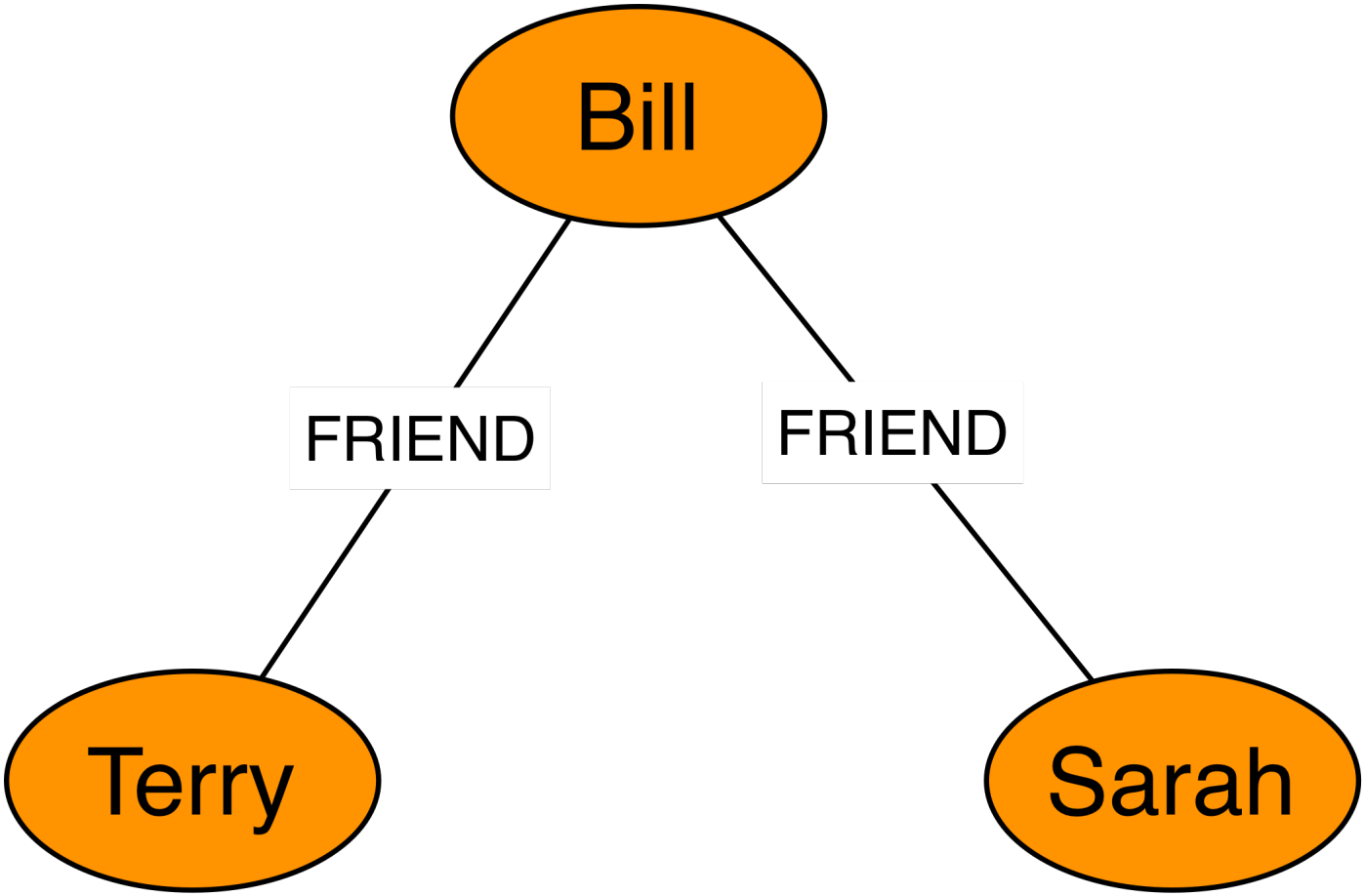
Recommending New Connections



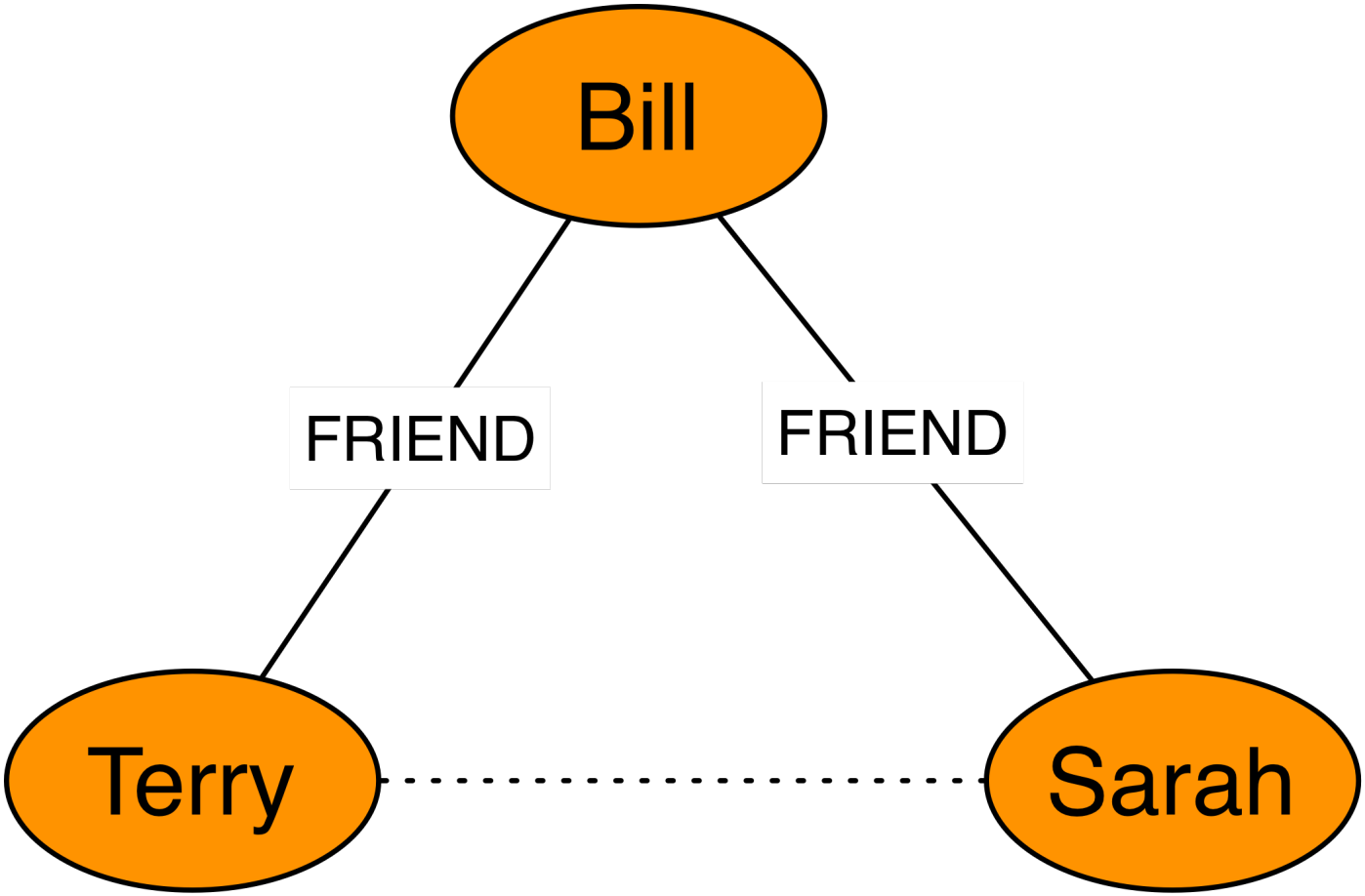
Immediate Friendships



Means and Motive



Recommendation



Recommend New Connections

```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

Find Terry

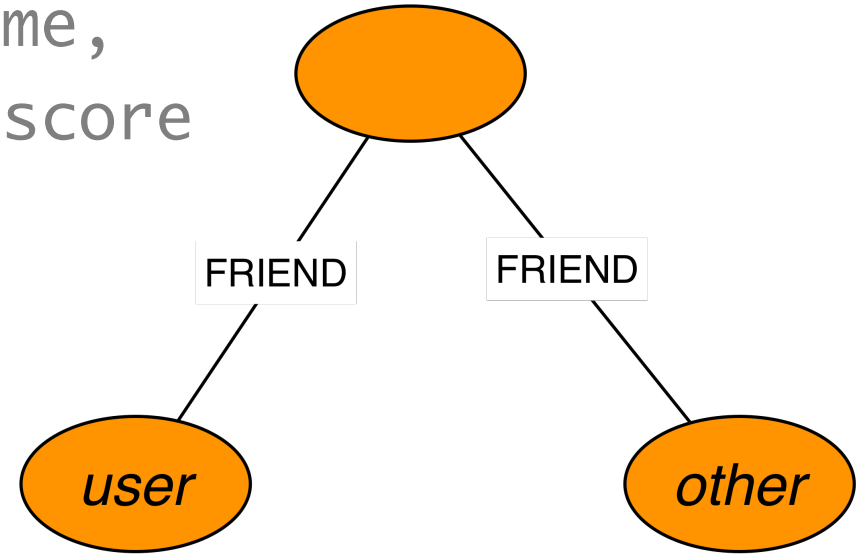
```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

Find Terry's Friends' Friends

```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

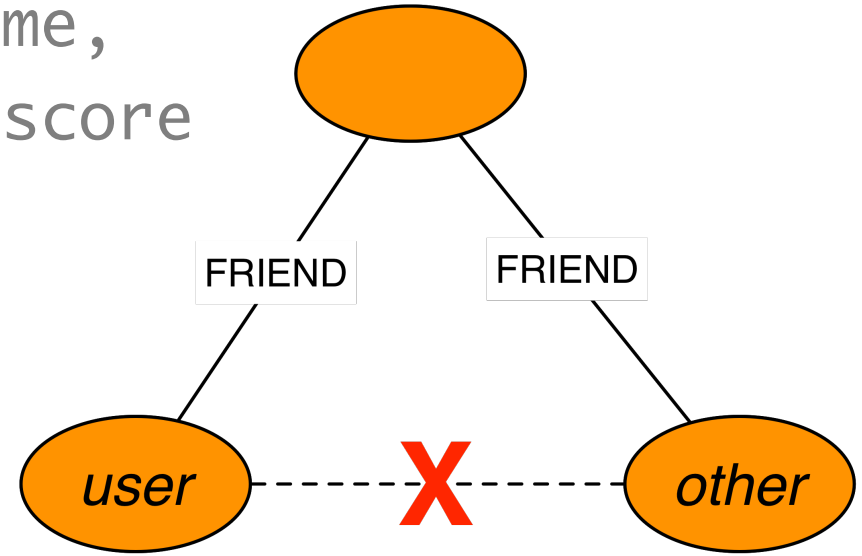
Find Terry's Friends' Friends

```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```



…Who Terry Doesn't Know

```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```



Count Matches Per Person

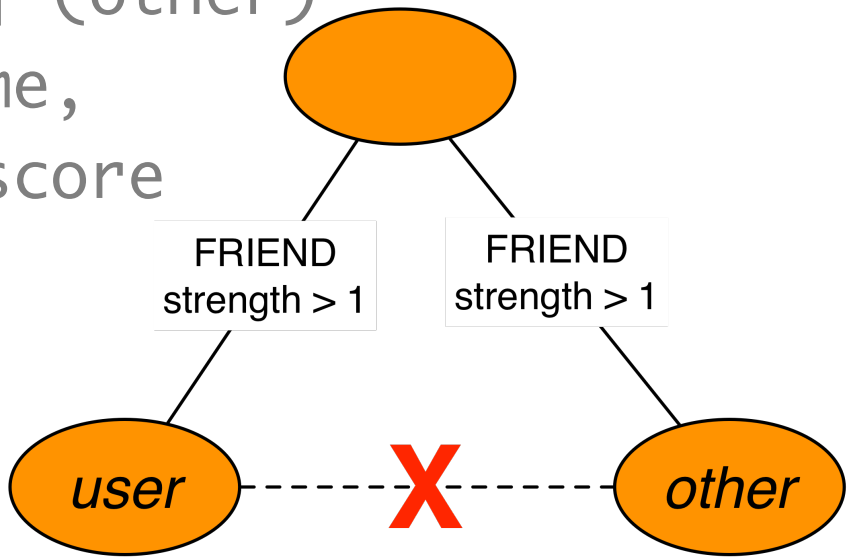
```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

Return The Results

```
MATCH (user:User{name:'Terry'})
      -[:FRIEND*2]-
      (other:User)
WHERE NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

Taking Account of Friendship Strength

```
MATCH (user:User{name:'Terry'})
      -[rels:FRIEND*2]-
      (other:User)
WHERE ALL(r IN rels WHERE r.strength > 1)
AND NOT (user)-[:FRIEND]-(other)
RETURN other.name AS name,
       COUNT(other) AS score
ORDER BY score DESC
```

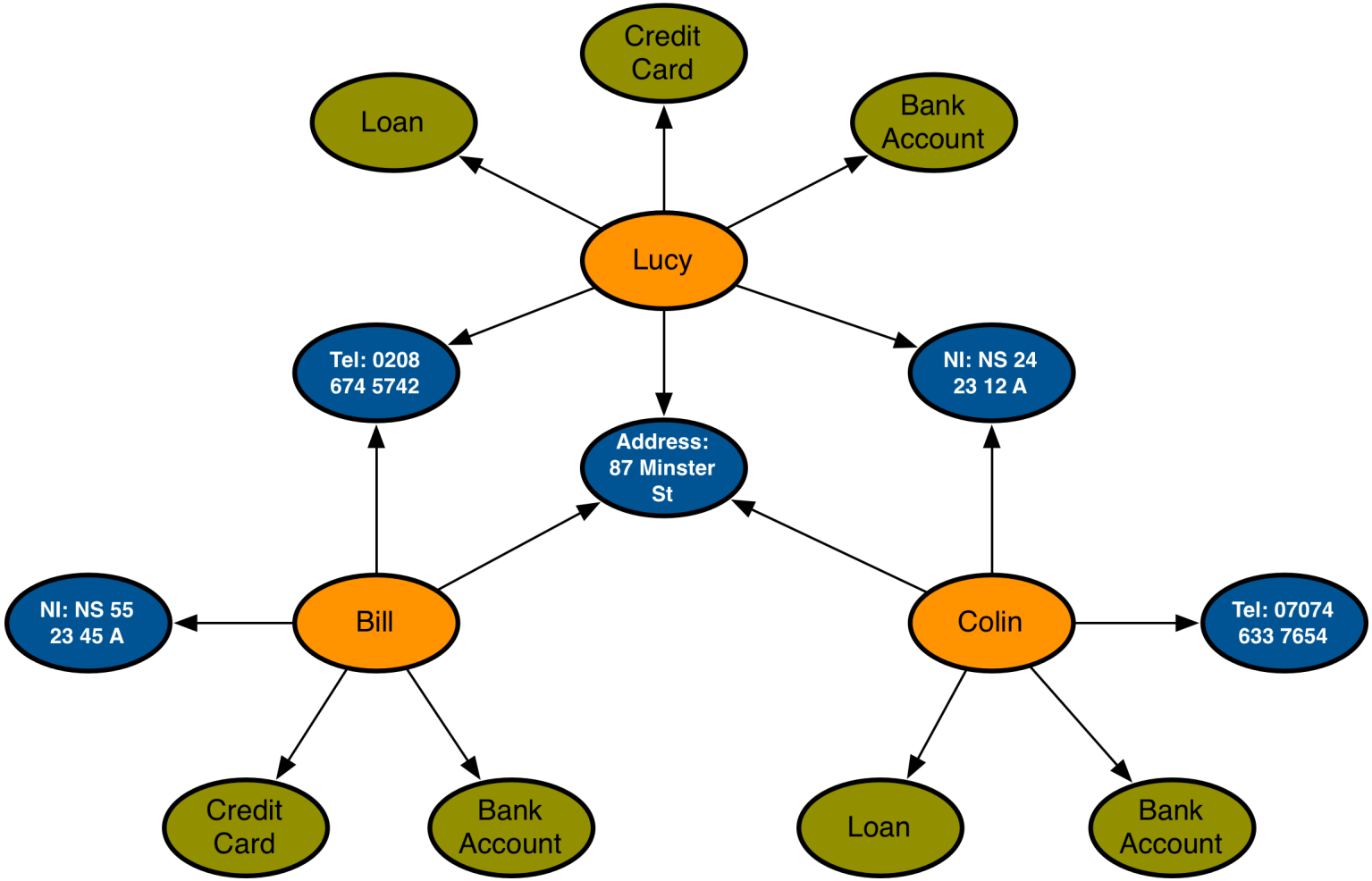


Nowhere To Hide

First-Party Fraud

- Fraudsters apply for credit
  - No intention of repaying
- Appear normal until they "burst out"
  - Clear out accounts
- Fraud ring
  - Share bits of identity (NI, address, telephone)
  - Coordinated "burst out"

Fraud Ring



@neo4j

Query

- Create new applicant
- Connect applicant to identity info
  - Reuse existing identify info where possible

Then

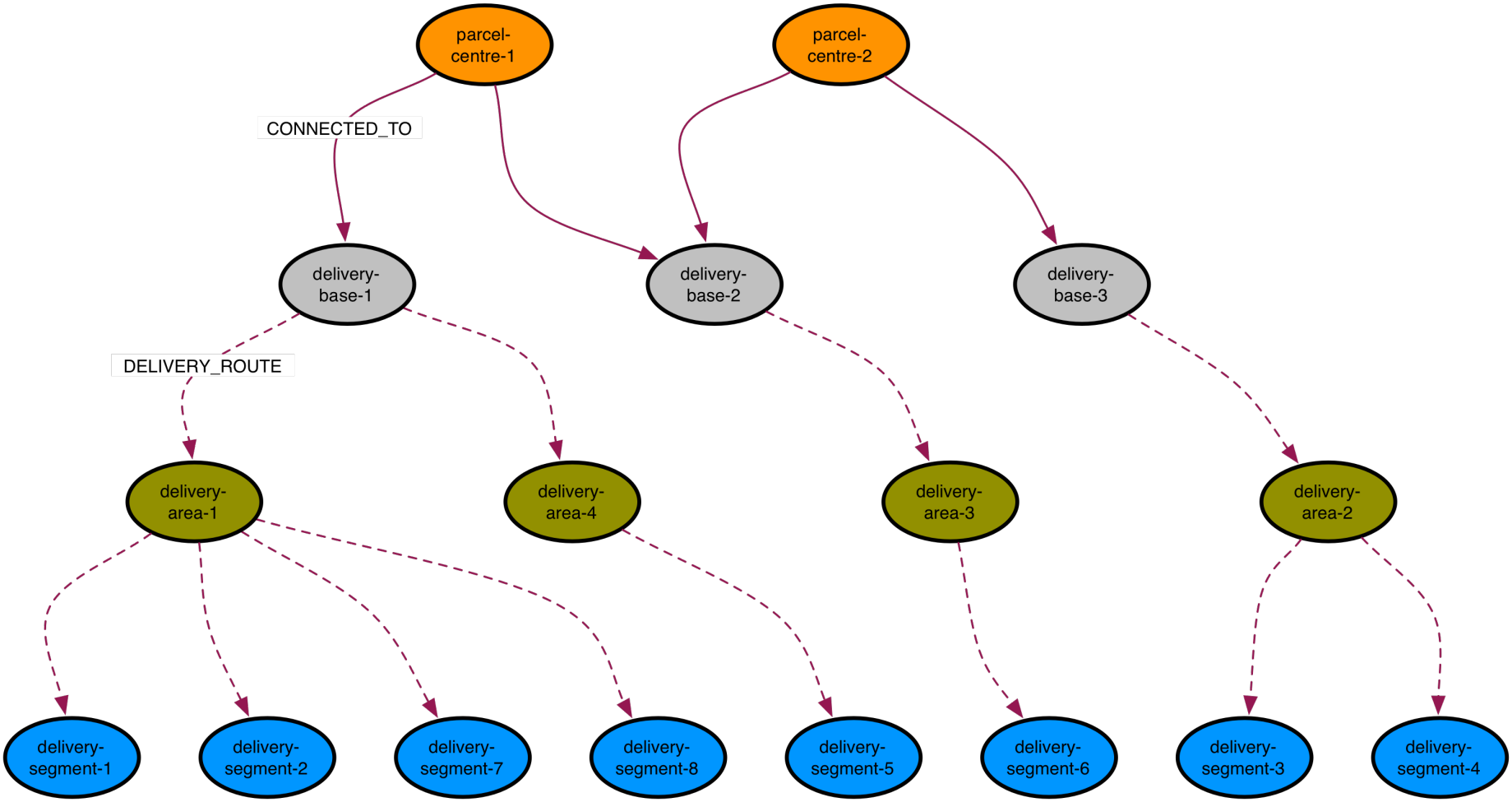
- Select applicant's identity info
- Crawl surrounding graph
  - Look for expansive clusters of account holders

Path Calculations

Problem

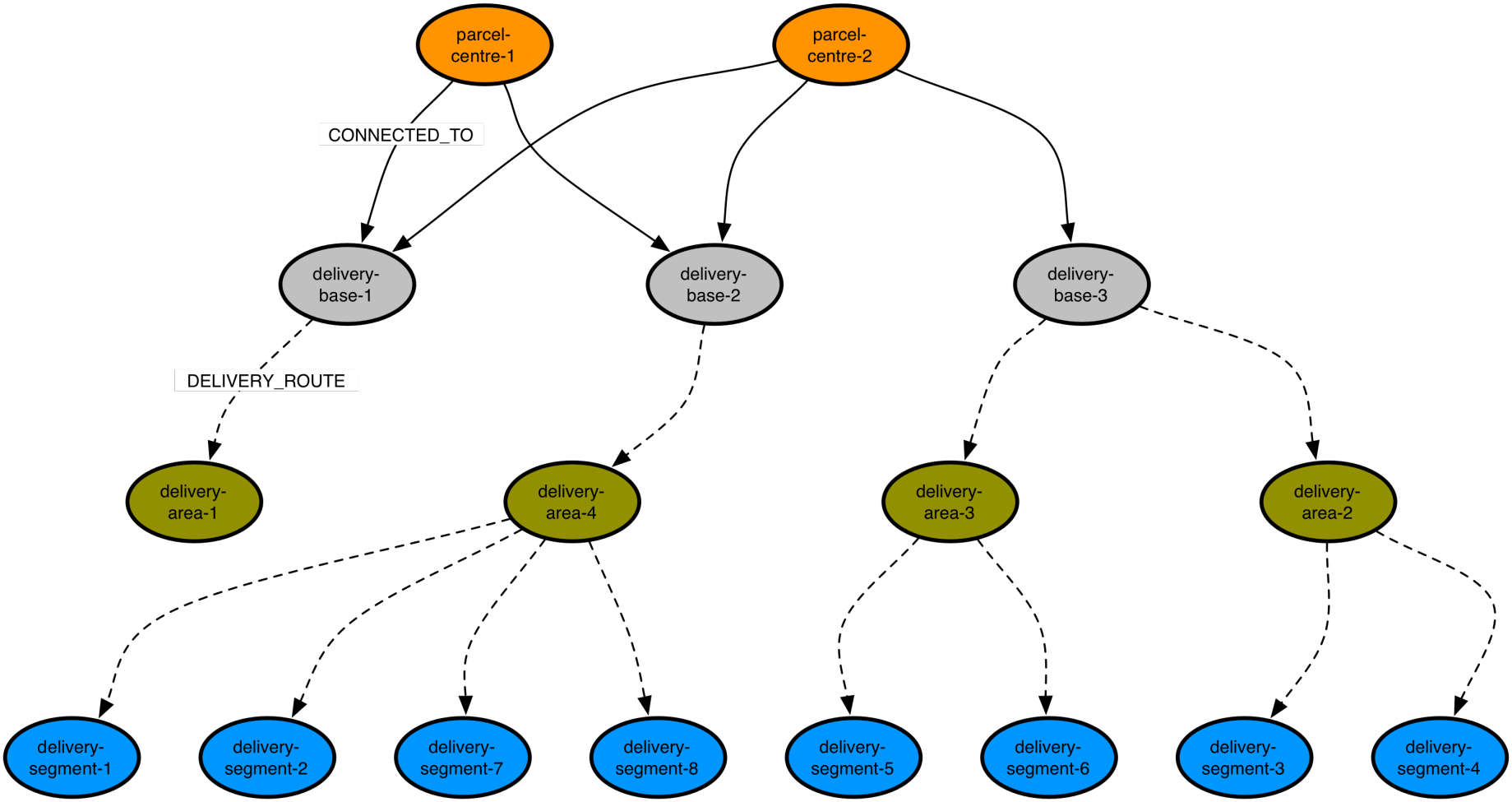
- Increase in parcel traffic
  - Amazon, eBay
  - Current infrastructure can't cope
- Calculate optimal route
  - Under 40ms
  - Routes vary over time
- Numbers:
  - 2000-3000 parcels per second
  - 25 national parcel centres, 2 million postcodes, 30 million address

Period 1



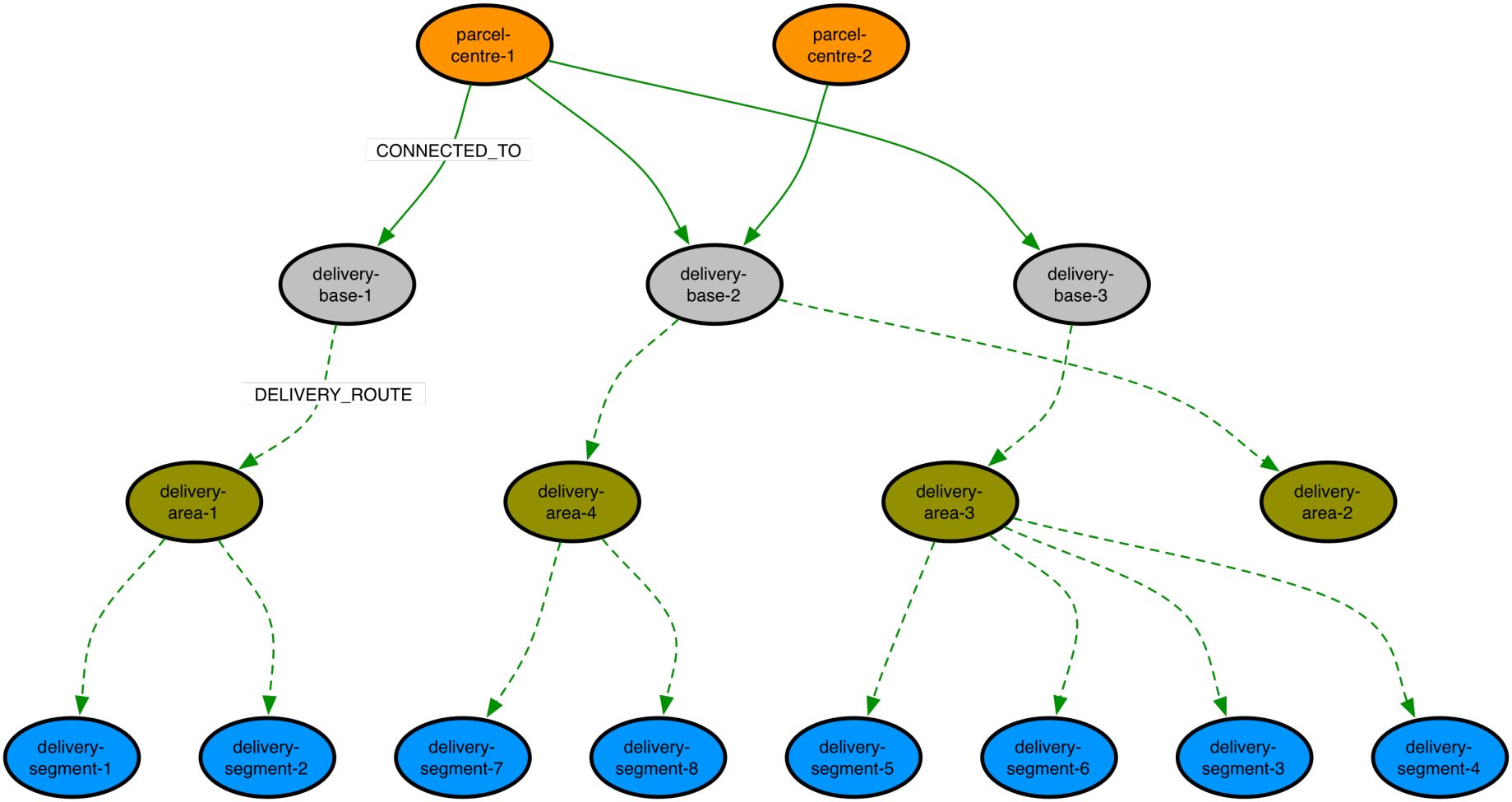
@neo4j

Period 2



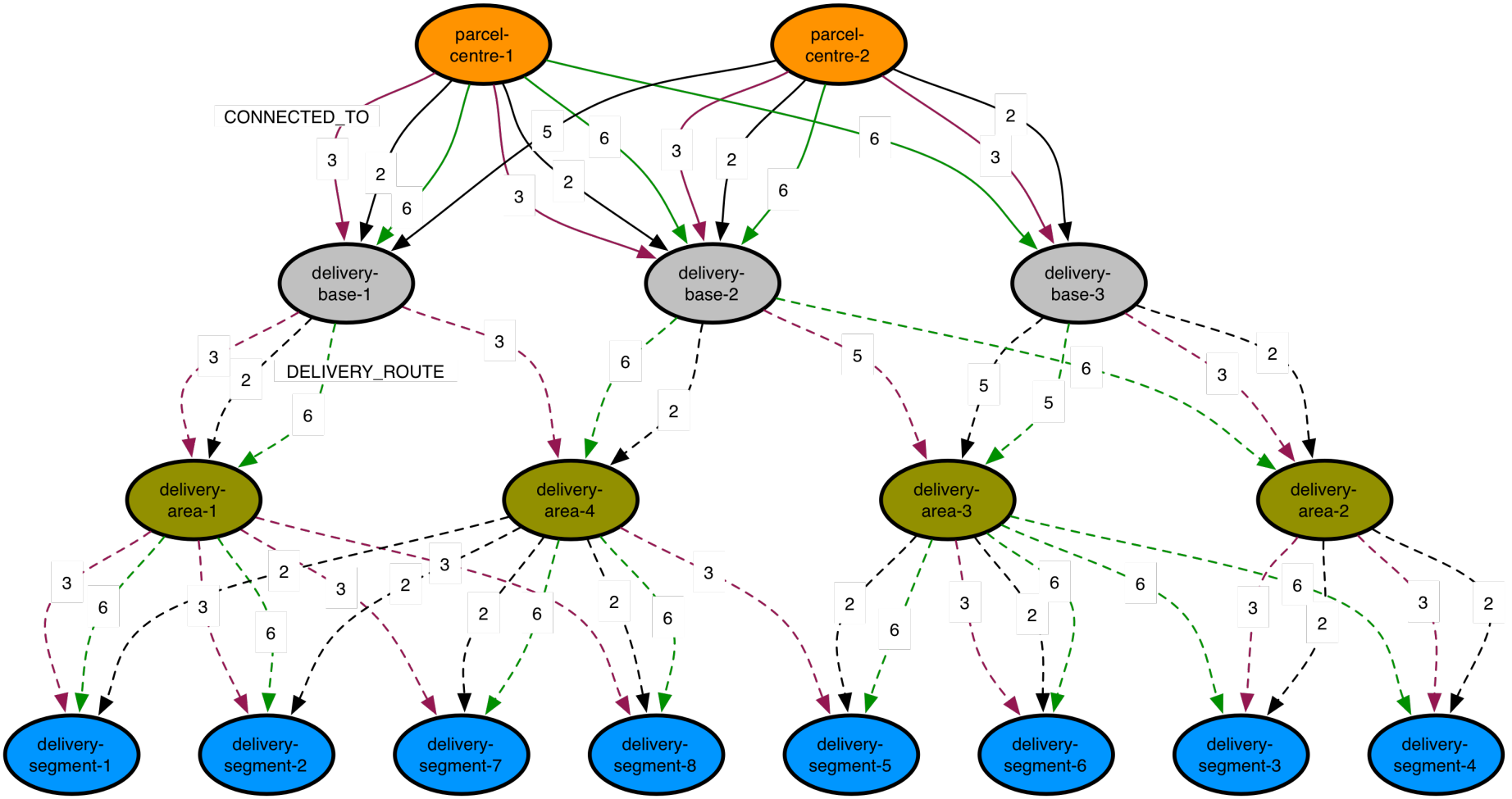
@neo4j

Period 3

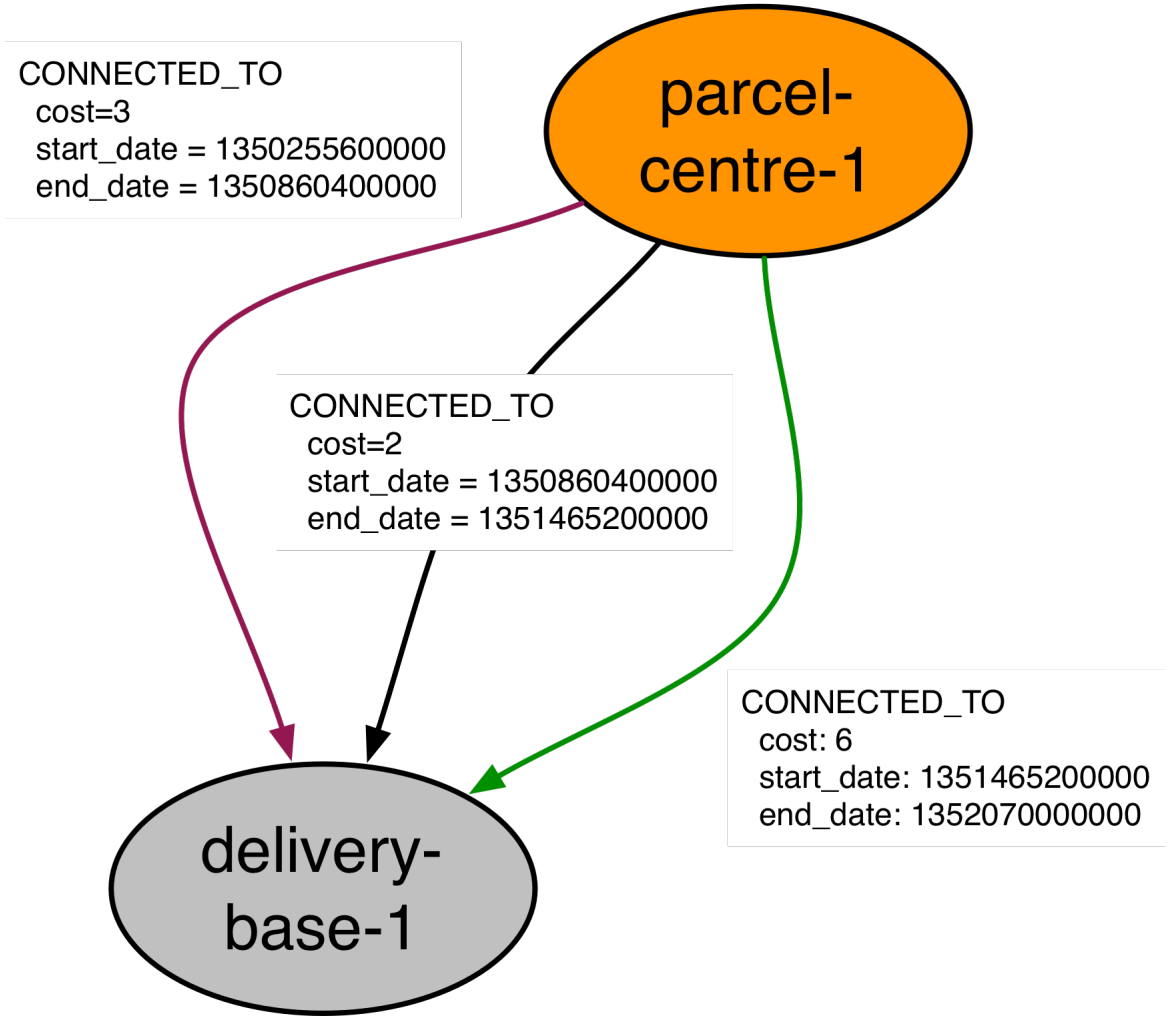


@neo4j

The Full Graph

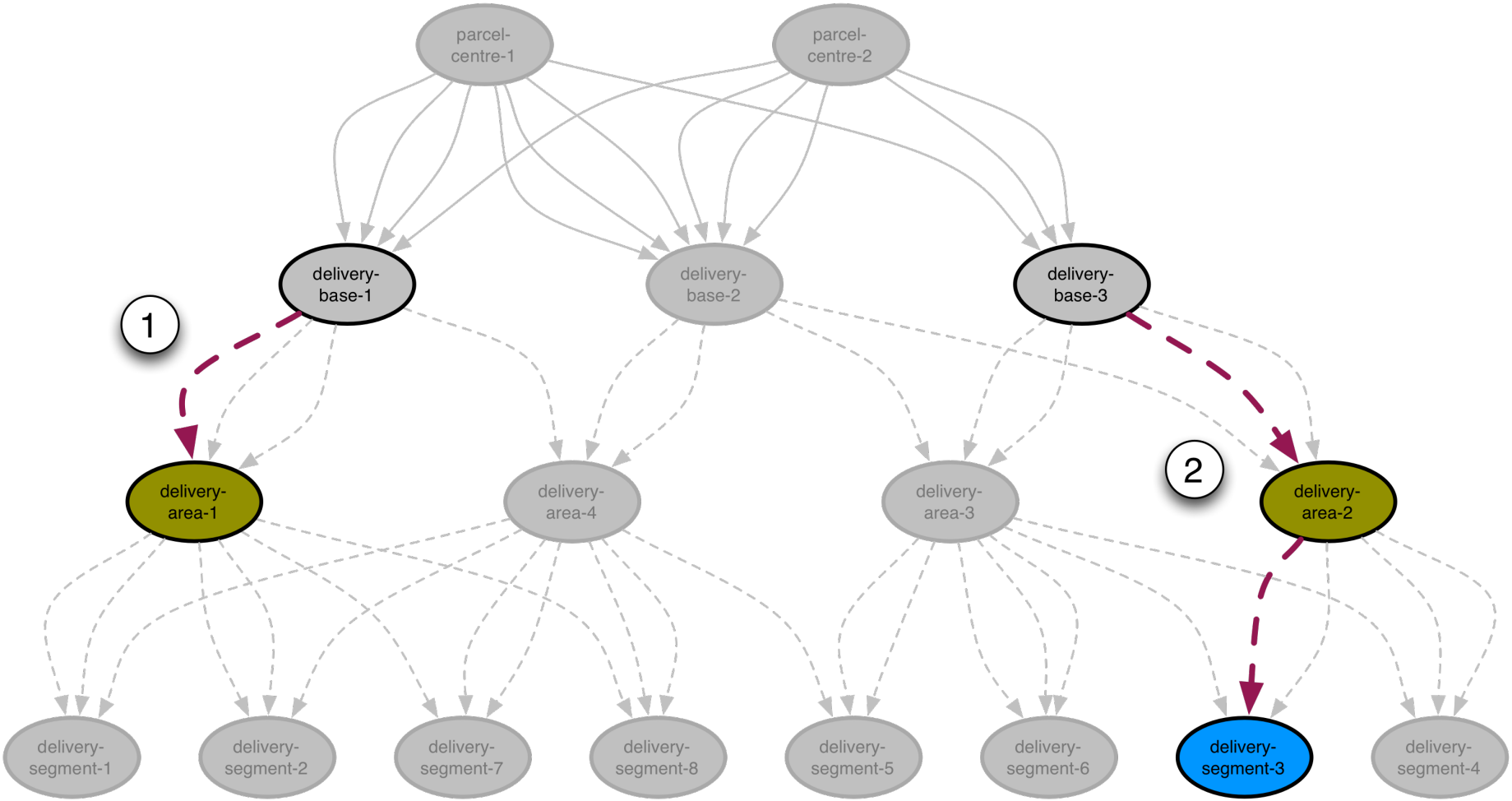


@neo4j


CONNECTED_TO
cost=3
start_date = 1350255600000
end_date = 1350860400000
parcel-centre-1
CONNECTED_TO
cost=2
start_date = 1350860400000
end_date = 1351465200000
CONNECTED_TO
cost: 6
start_date: 1351465200000
end_date: 1352070000000
delivery-base-1

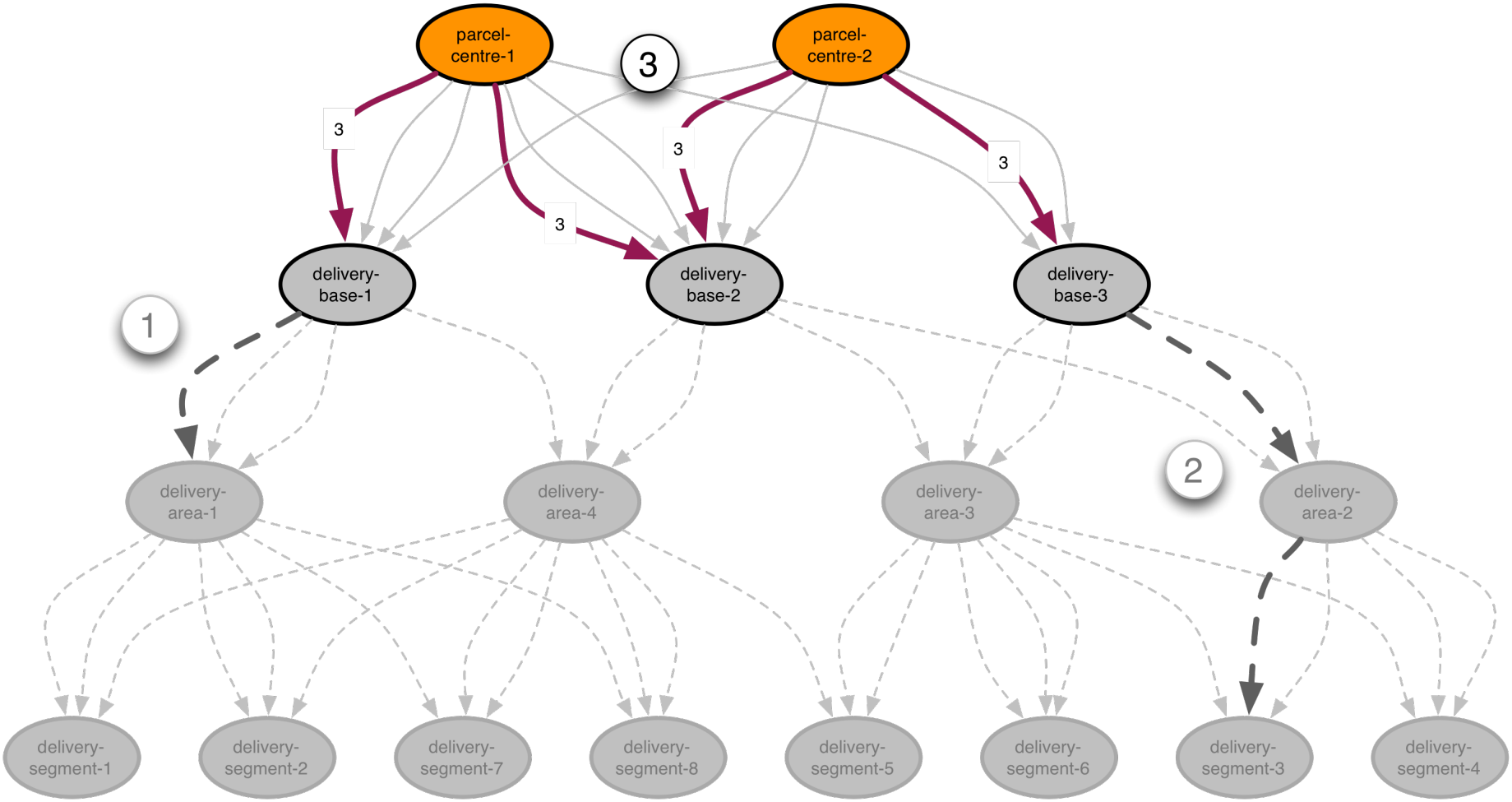
@neo4j

Steps 1 and 2



@neo4j

Step 3


parcel-centre-1
parcel-centre-2
3
3
3
3
3
delivery-base-1
delivery-base-2
delivery-base-3
1
2
delivery-area-1
delivery-area-4
delivery-area-3
delivery-area-2
delivery-segment-1
delivery-segment-2
delivery-segment-7
delivery-segment-8
delivery-segment-5
delivery-segment-6
delivery-segment-3
delivery-segment-4

@neo4j

Paths

```
MATCH  path=(from{name:'X'})
             -[:CONNECTED_TO*1..4]->
            (to{name:'Y'})
RETURN path AS shortestPath,
       reduce(weight=0,
              r in relationships(path) |
              weight + r.weight) AS total
ORDER BY total ASC
LIMIT 1
```

Match Variable-Length Path

```
MATCH  path=(from{name:'X'})
             -[:CONNECTED_TO*1..4]->
            (to{name:'Y'})
RETURN path AS shortestPath,
       reduce(weight=0,
              r in relationships(path) |
              weight + r.weight) AS total
ORDER BY total ASC
LIMIT 1
```

Calculate Path Weight

```
MATCH  path=(from{name:'X'})
             -[:CONNECTED_TO*1..4]->
            (to{name:'Y'})
RETURN path AS shortestPath,
       reduce(weight=0,
              r in relationships(path) |
              weight + r.weight) AS total
ORDER BY total ASC
LIMIT 1
```

Return Shortest Weighted Path

```
MATCH  path=(from{name:'X'})
             -[:CONNECTED_TO*1..4]->
            (to{name:'Y'})
RETURN path AS shortestPath,
       reduce(weight=0,
              r in relationships(path) |
              weight + r.weight) AS total
ORDER BY total ASC
LIMIT 1
```

Full Query

```
MATCH (s:Location {name:{startLocation}}),
      (e:Location {name:{endLocation}})
MATCH upLeg = (s)<-[:DELIVERY_ROUTE*1..2]-(db1)
WHERE all(r in relationships(upLeg)
          WHERE r.start_date <= {intervalStart}
          AND r.end_date >= {intervalEnd})
WITH  e, upLeg, db1
MATCH downLeg = (db2)-[:DELIVERY_ROUTE*1..2]->(e)
WHERE all(r in relationships(downLeg)
          WHERE r.start_date <= {intervalStart}
          AND r.end_date >= {intervalEnd})
WITH  db1, db2, upLeg, downLeg
MATCH topRoute = (db1)<-[:CONNECTED_TO]-()-[:CONNECTED_TO*1..3]-(db2)
WHERE all(r in relationships(topRoute)
          WHERE r.start_date <= {intervalStart}
          AND r.end_date >= {intervalEnd})
WITH  upLeg, downLeg, topRoute,
      reduce(weight=0, r in relationships(topRoute) | weight+r.cost) AS score
      ORDER BY score ASC
      LIMIT 1
RETURN (nodes(upLeg) + tail(nodes(topRoute)) + tail(nodes(downLeg))) AS route
```

neo4j.com/online_course

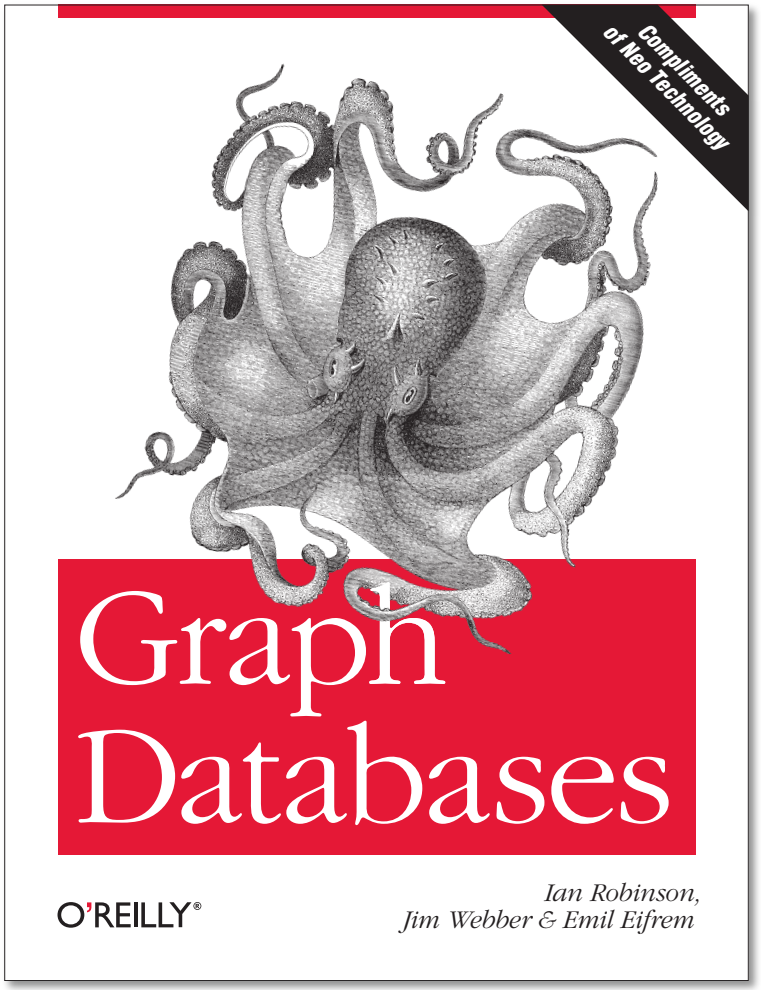
Online Training: Getting Started with Neo4j

Learn Neo4j at your own pace and time with our free online training course. Get introduced to graph databases, learn the core functionality of Neo4j, and practice Cypher with this engaging and interactive course.

Get started today »

@neo4j

graphdatabases.com



Thank you

@ian**S**robinson
#neo4j

@neo4j