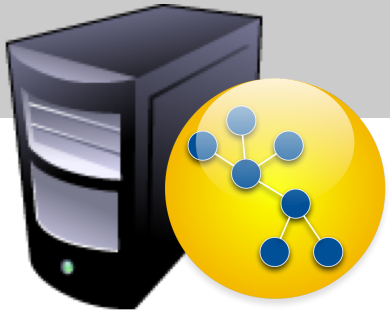
Test-Driven Data Modeling With Graphs

@ianSrobinson
ian@neotechnology.com

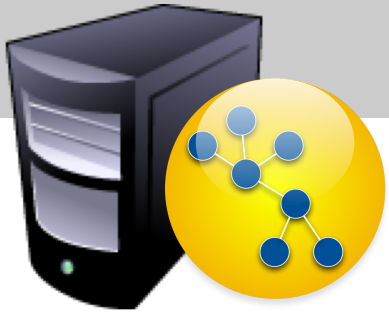
#neo4j

Outline

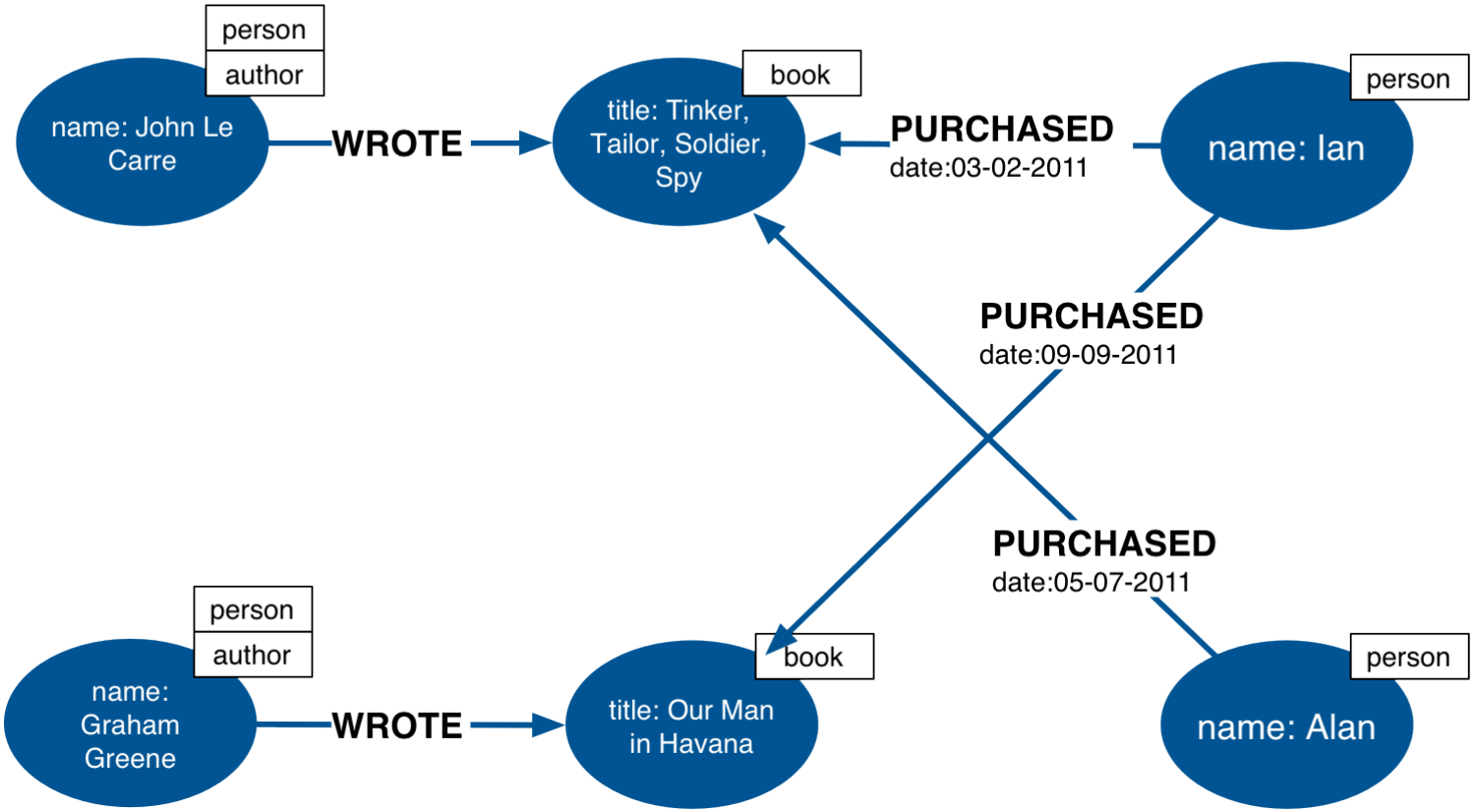


- Data modeling with graphs
- Neo4j application architecture options
- Testing your data model

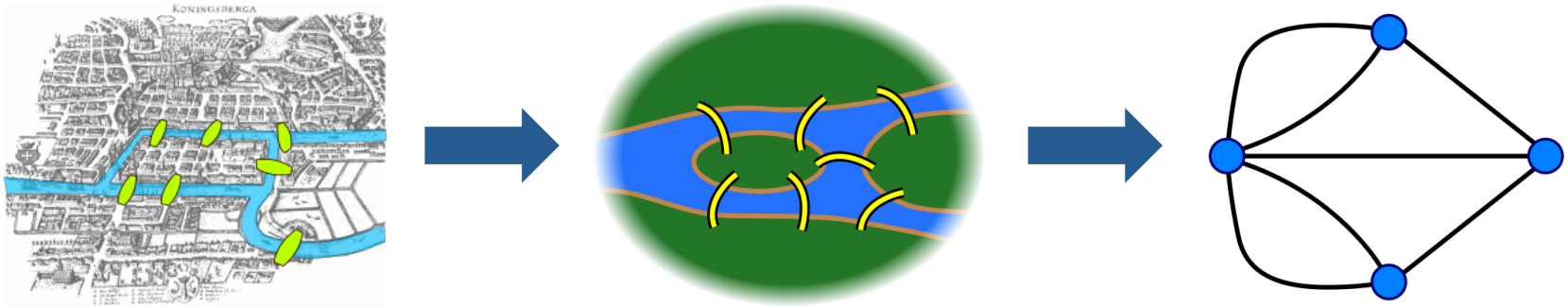
Graph data modeling



Property Graph Data Model



Models



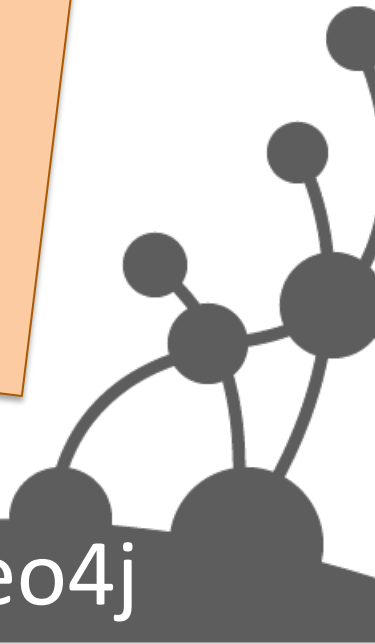
Images: en.wikipedia.org

User stories

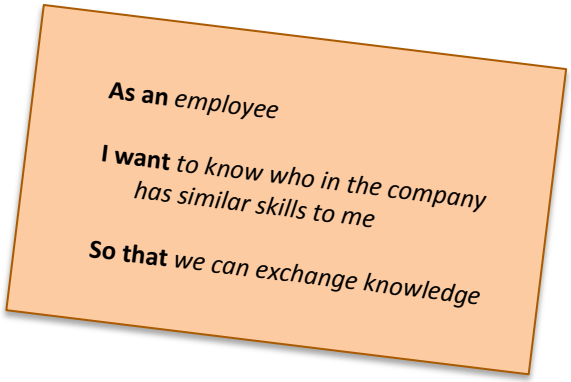
As an *employee*

I want *to know who in the company has similar skills to me*

So that *we can exchange knowledge*



Derive questions



Which people, who work for the same company as me, have similar skills to me?

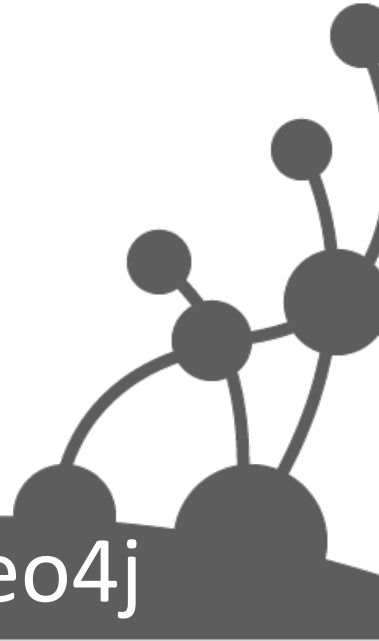
Identify entities

Which people, who work for the same company as me, have similar skills to me?

person

company

skill

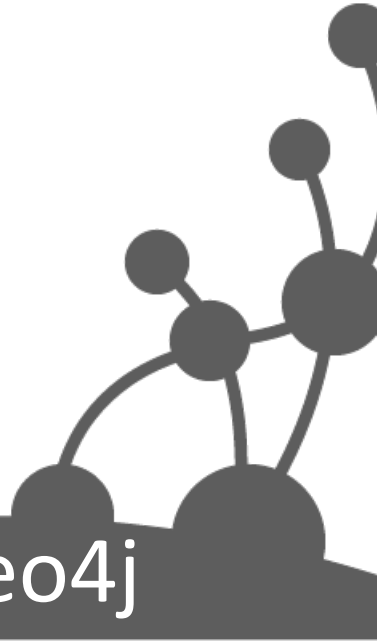


Identify relationships between entities

Which people, who work for the same company as me, have similar skills to me?

person WORKS_FOR company

person HAS_SKILL skill

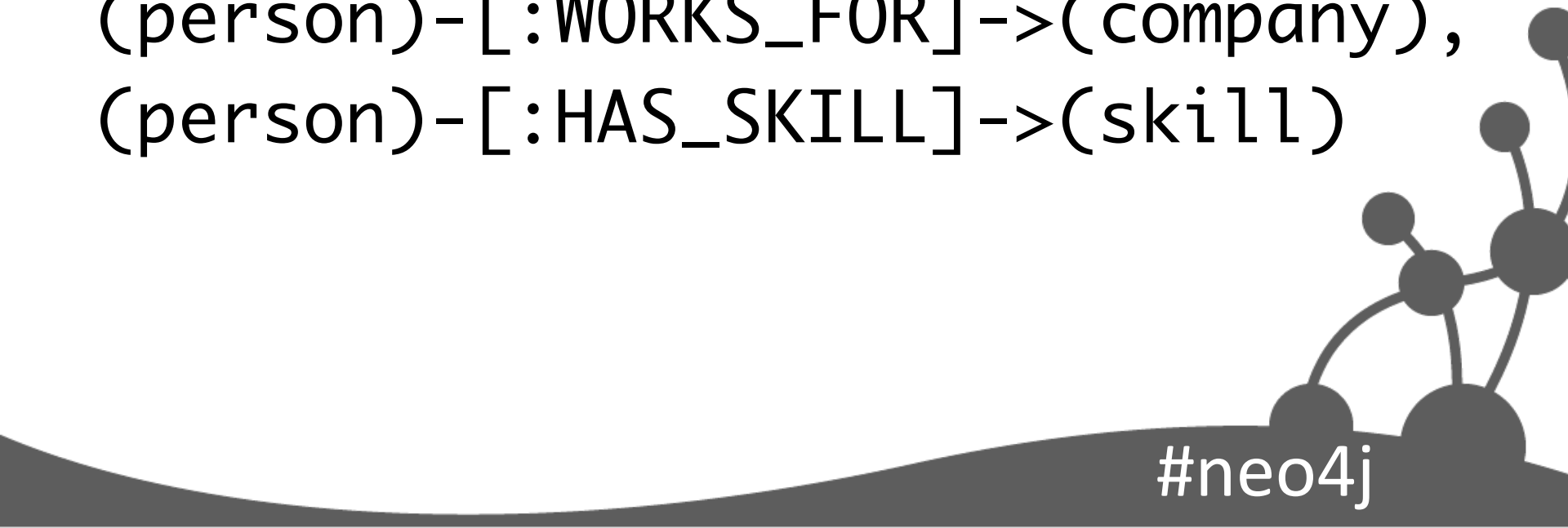


Convert to Cypher paths

person WORKS_FOR company

person HAS_SKILL skill

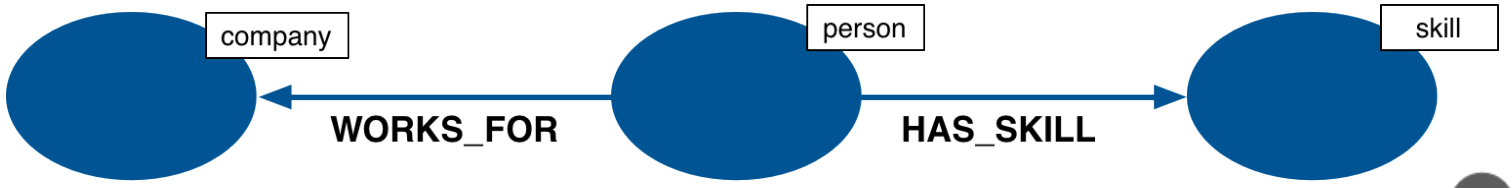
```
(person)-[:WORKS_FOR]->(company),
(person)-[:HAS_SKILL]->(skill)
```



Cypher paths

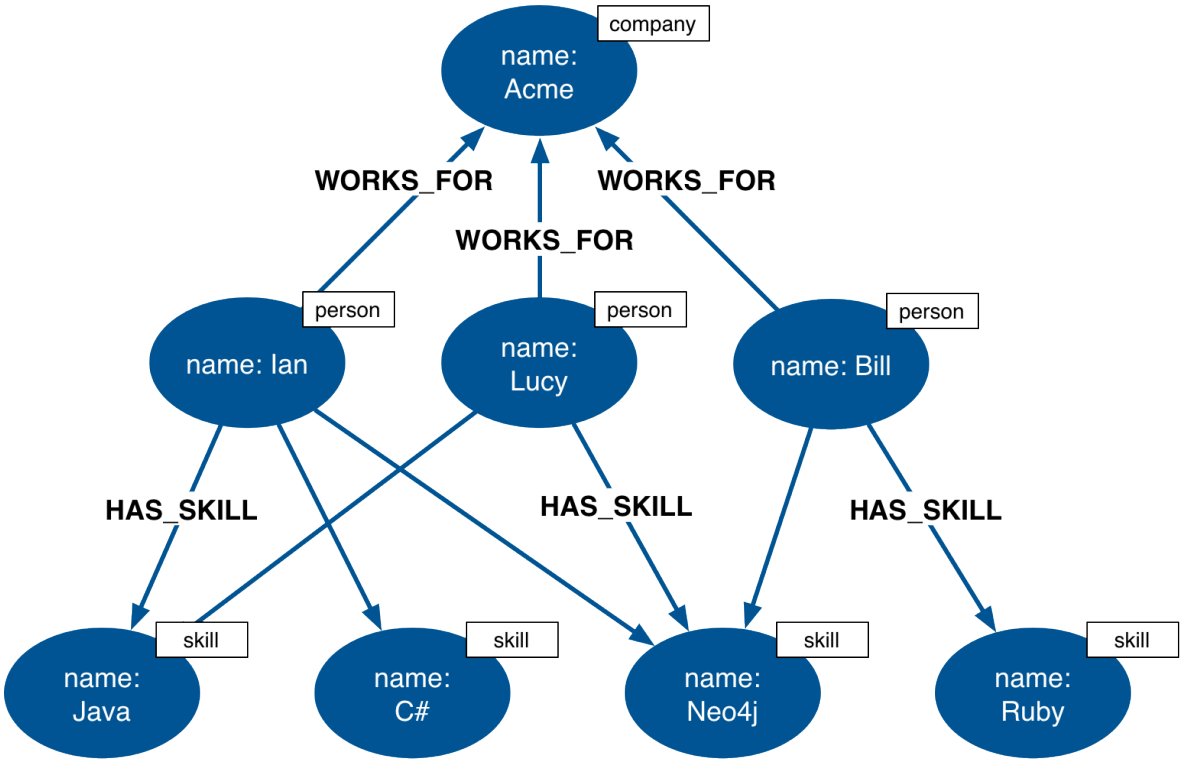
```
(person)-[:WORKS_FOR]->(company),
(person)-[:HAS_SKILL]->(skill)
```

```
(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)
```



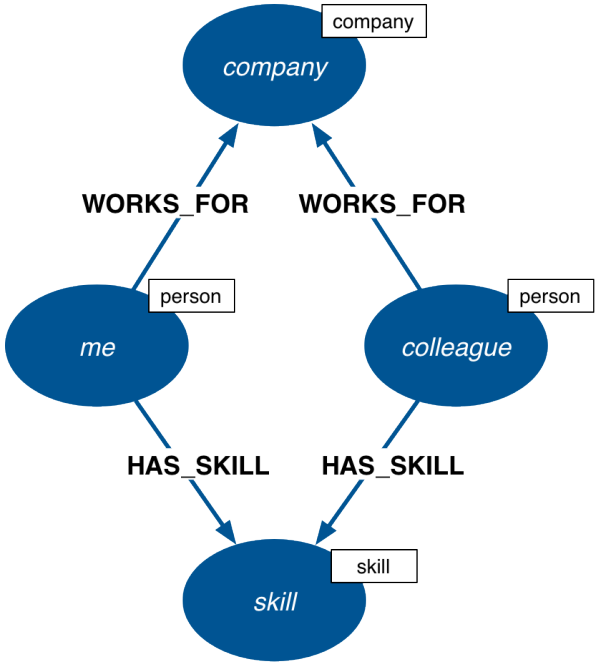
Data model

```
(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)
```



Formulating question as graph pattern

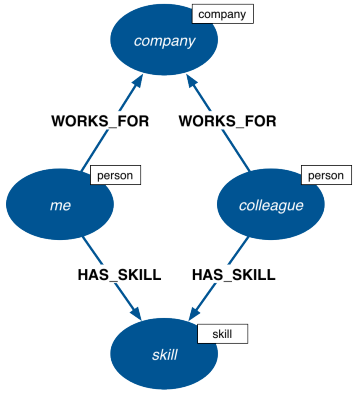
Which people, who work for the same company as me, have similar skills to me?



Cypher query

Which people, who work for the same company as me, have similar skills to me?

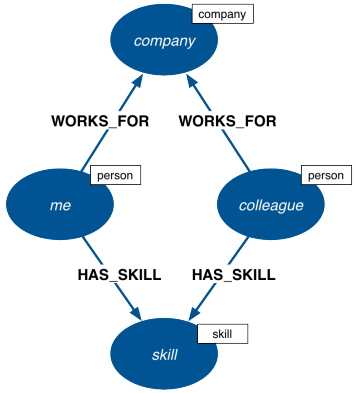
```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



Graph pattern

Which people, who work for the same company as me, have similar skills to me?

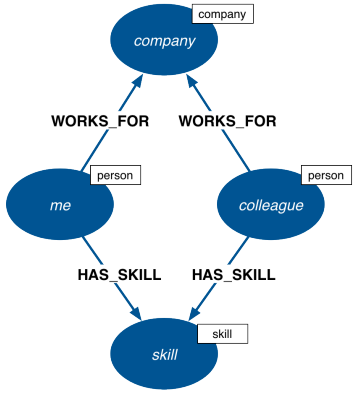
```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



Anchor pattern in graph

Which people, who work for the same company as me, have similar skills to me?

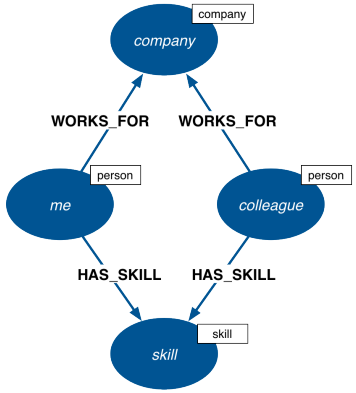
```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



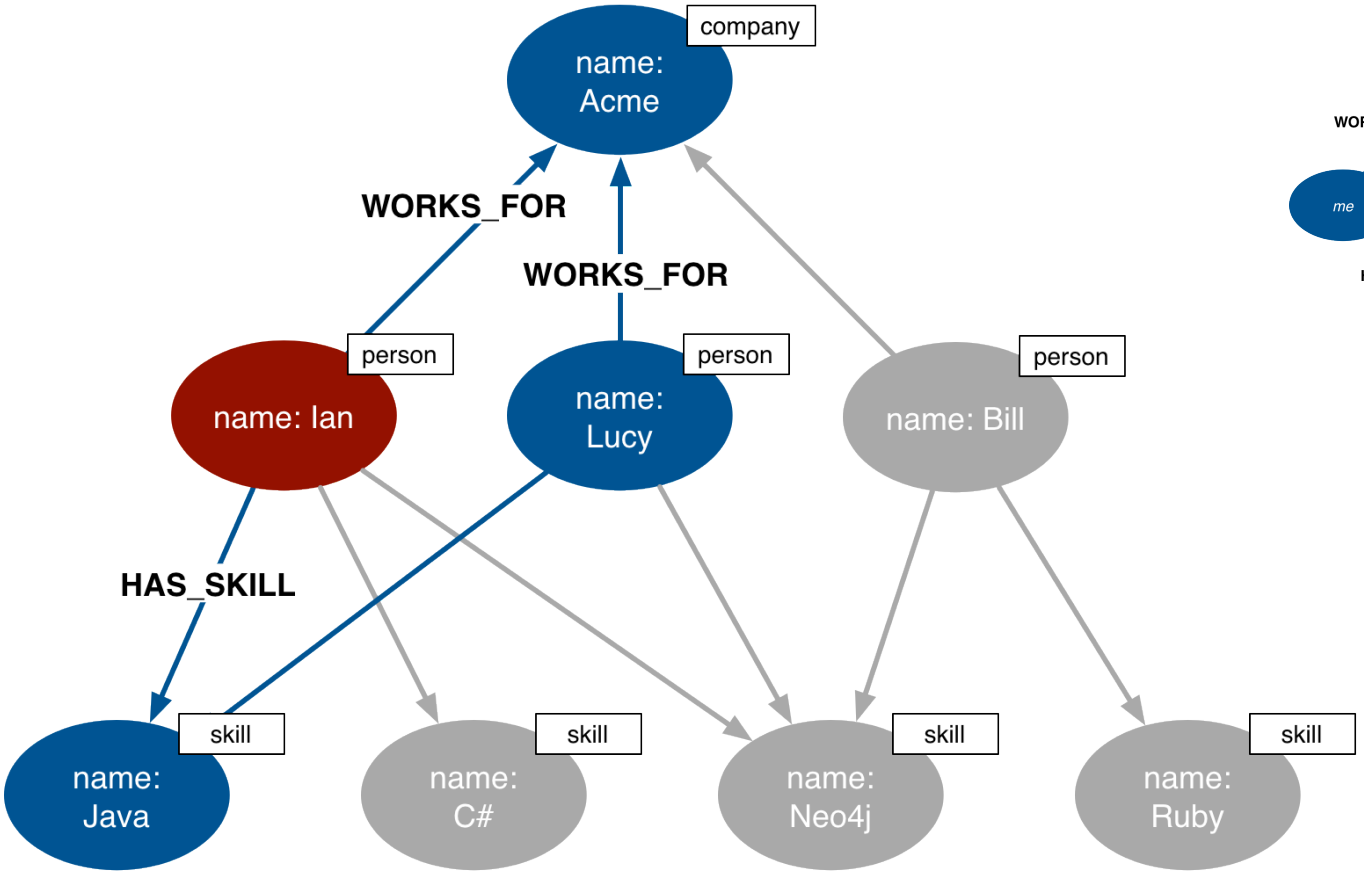
Create projection of results

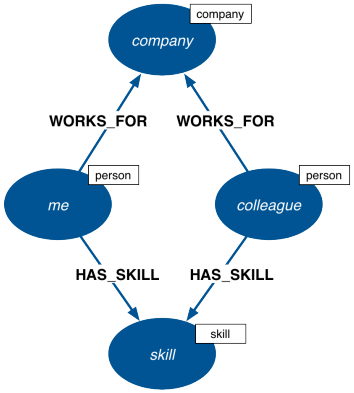
Which people, who work for the same company as me, have similar skills to me?

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

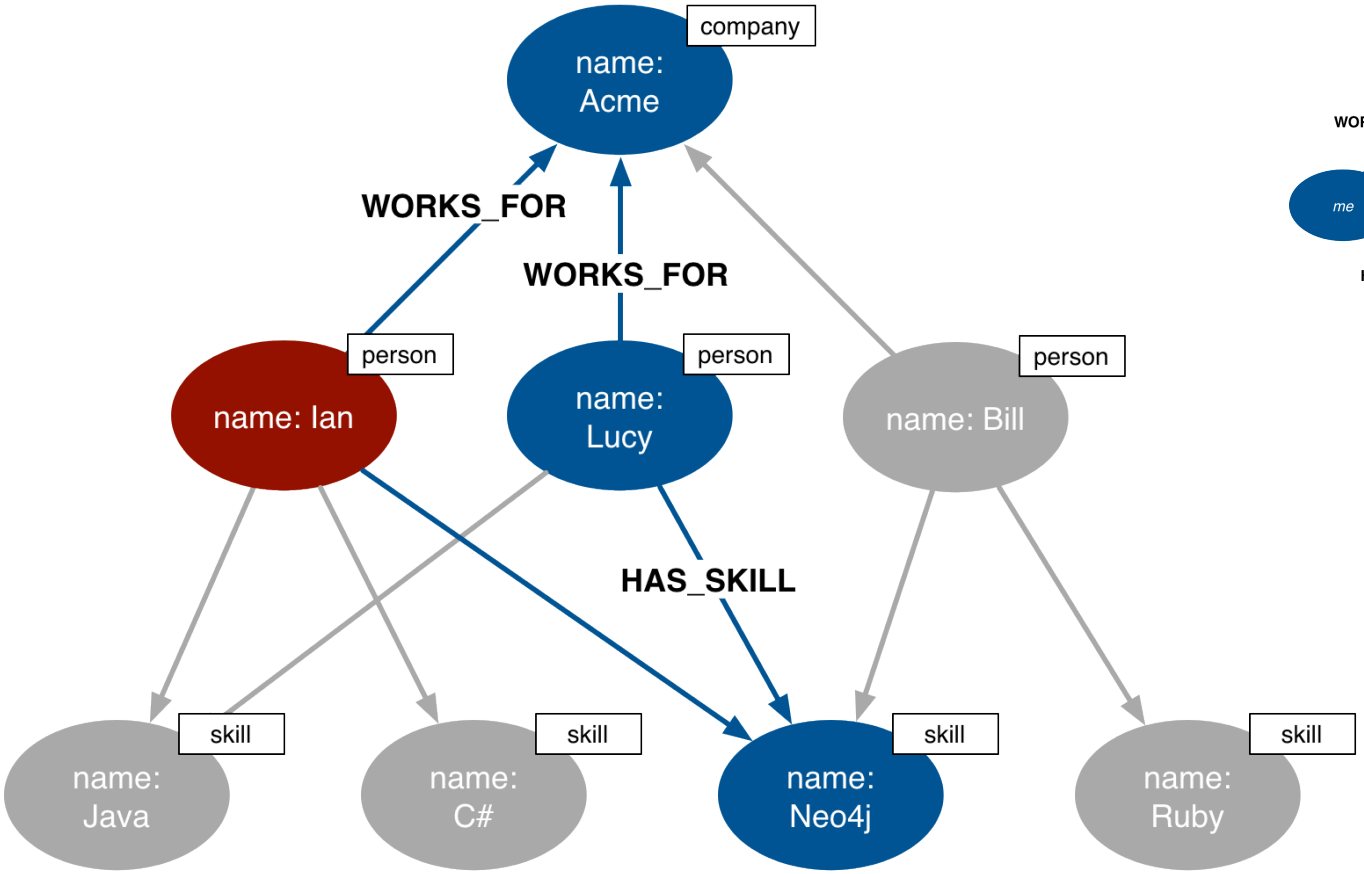


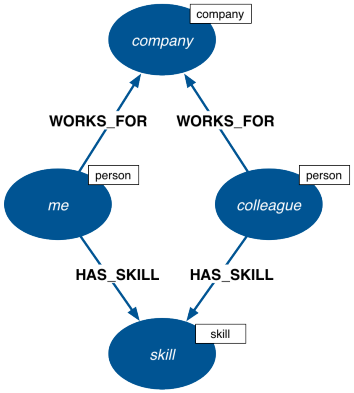
First match


company
name:
Acme
WORKS_FOR
WORKS_FOR
person
name: Ian
person
name:
Lucy
person
name: Bill
HAS_SKILL
skill
name:
Java
skill
name:
C#
skill
name:
Neo4j
skill
name:
Ruby

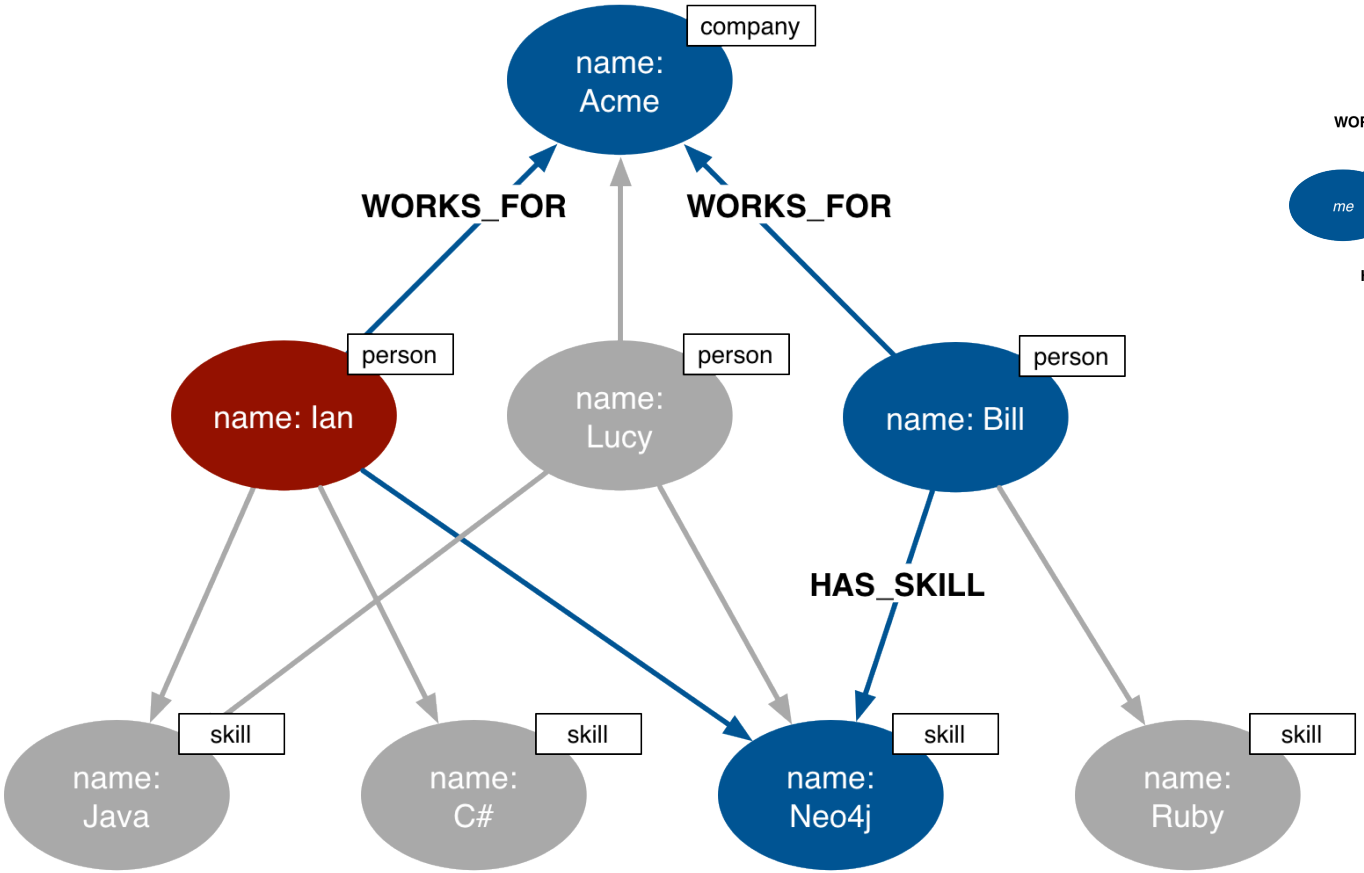

company
company
WORKS_FOR
WORKS_FOR
person
me
person
colleague
HAS_SKILL
HAS_SKILL
skill
skill

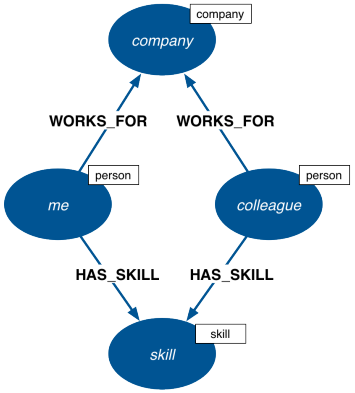
Second match





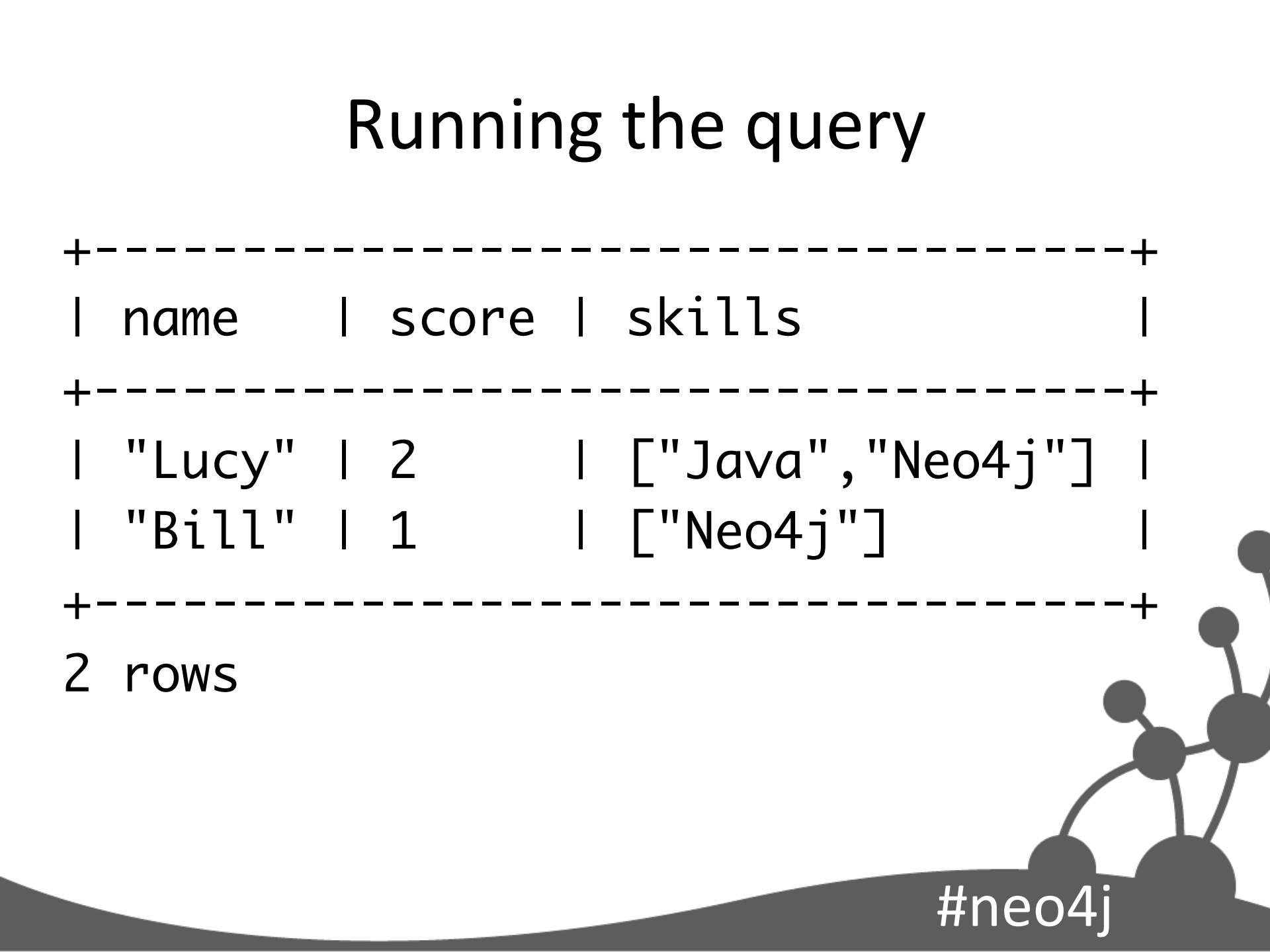
Third match


company
name: Acme
WORKS_FOR
WORKS_FOR
person
name: Ian
person
name: Lucy
person
name: Bill
HAS_SKILL
skill
name: Java
skill
name: C#
skill
name: Neo4j
skill
name: Ruby


company
company
WORKS_FOR
WORKS_FOR
person
me
person
colleague
HAS_SKILL
HAS_SKILL
skill
skill

Running the query

```
+-----------------------------------+
| name    | score | skills          |
+-----------------------------------+
| "Lucy" | 2     | ["Java","Neo4j"] |
| "Bill" | 1     | ["Neo4j"]        |
+-----------------------------------+
2 rows
```



From user story to model

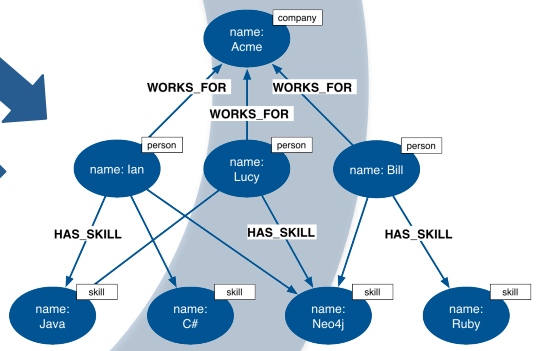
As an *employee*

I want *to know who in the company has similar skills to me*

So that *we can exchange knowledge*

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
      (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name = {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```

Which people, who work for the same company as me, have similar skills to me?



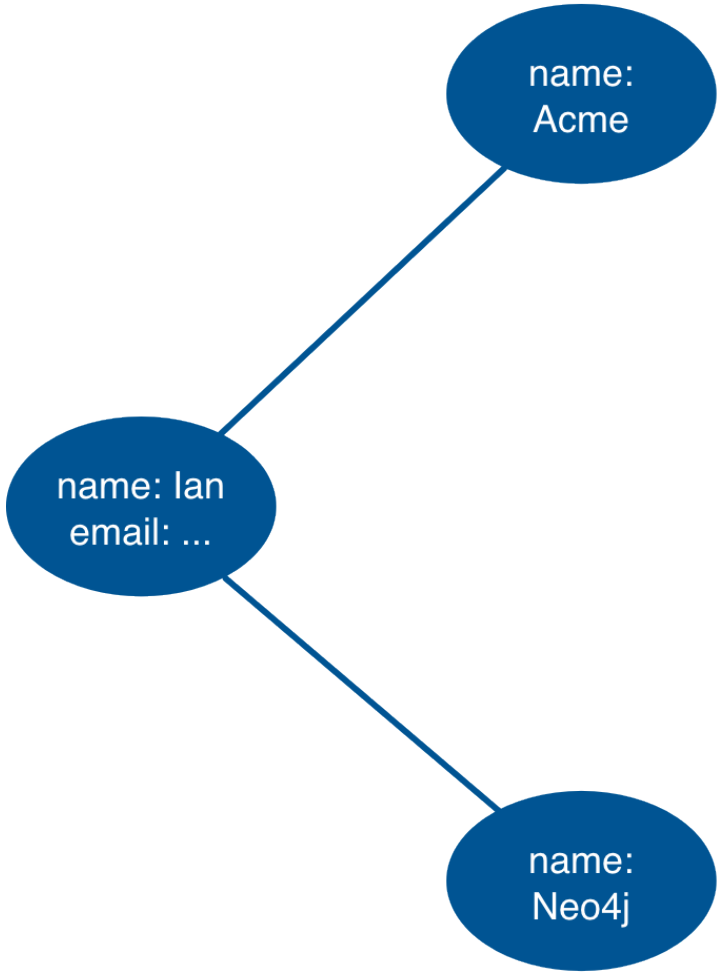
person WORKS_FOR company

person HAS_SKILL skill

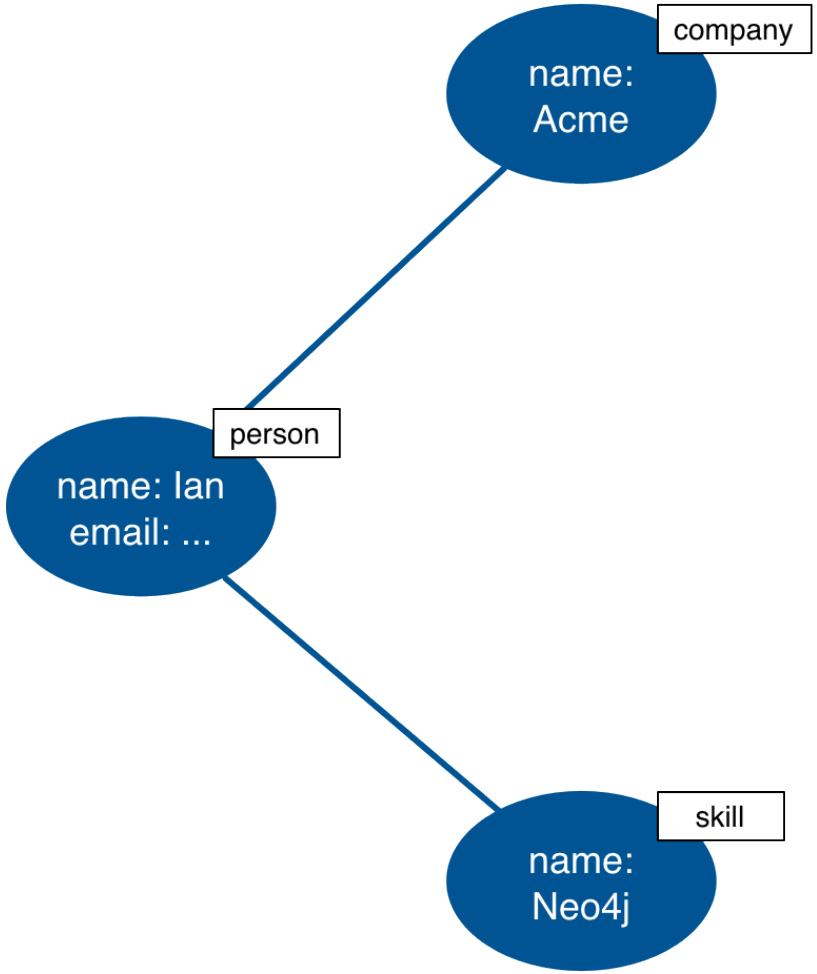
```
(company)<-[:WORKS_FOR]-(person)-[:HAS_SKILL]->(skill)
```

#neo4j

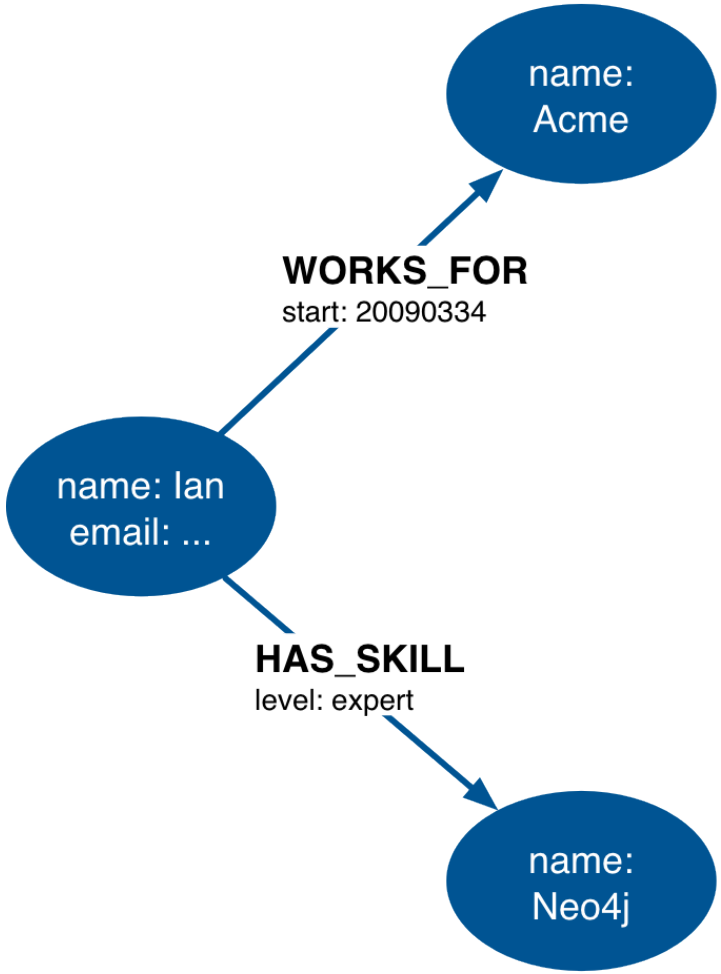
Nodes for things



Labels for grouping

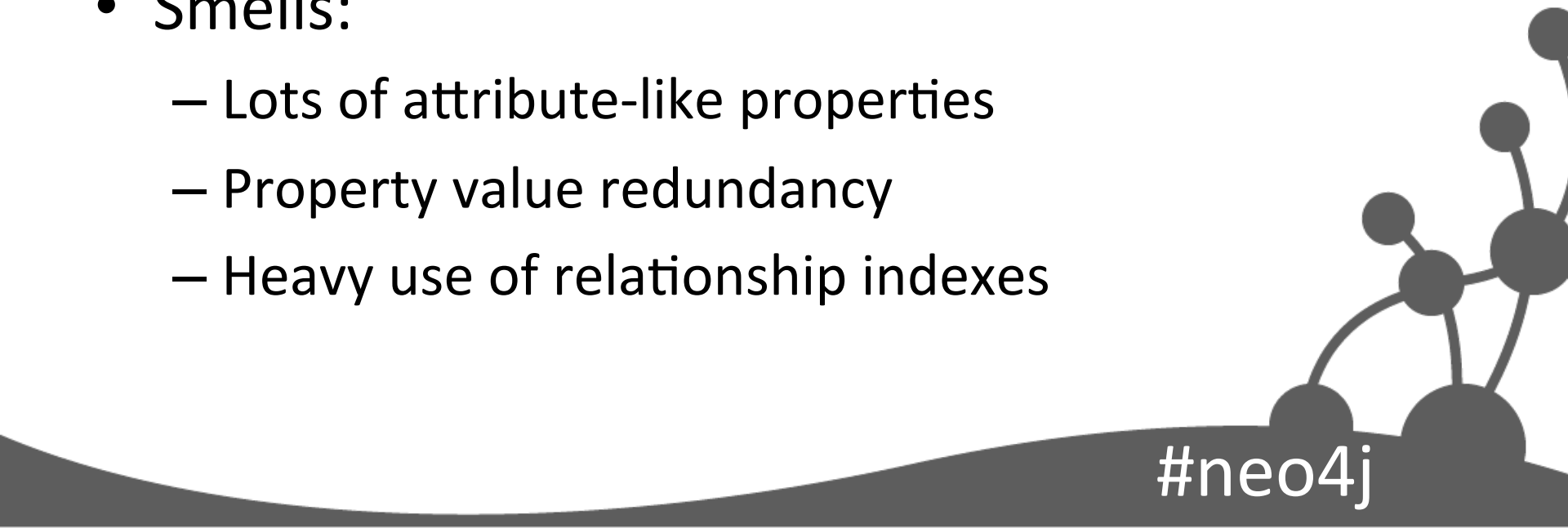


Relationships for structure

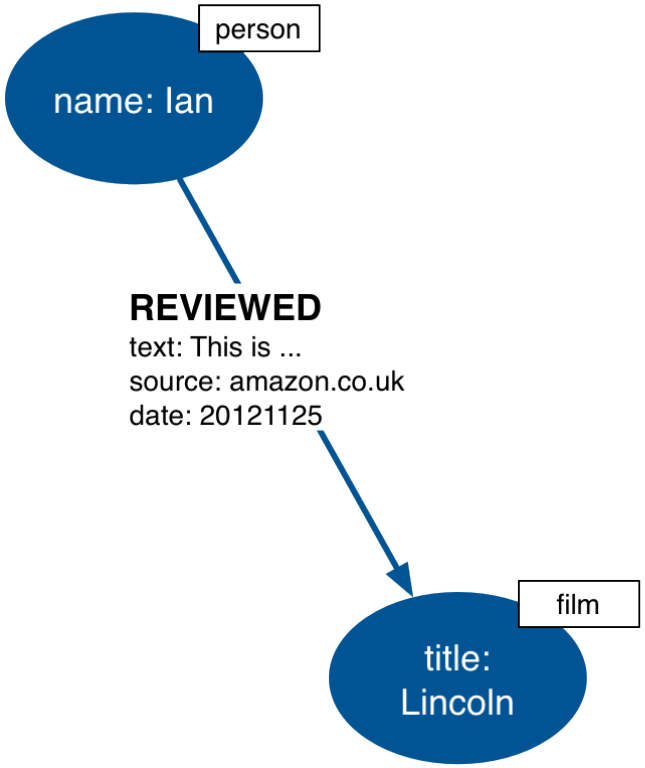


Don’t model entities as relationships

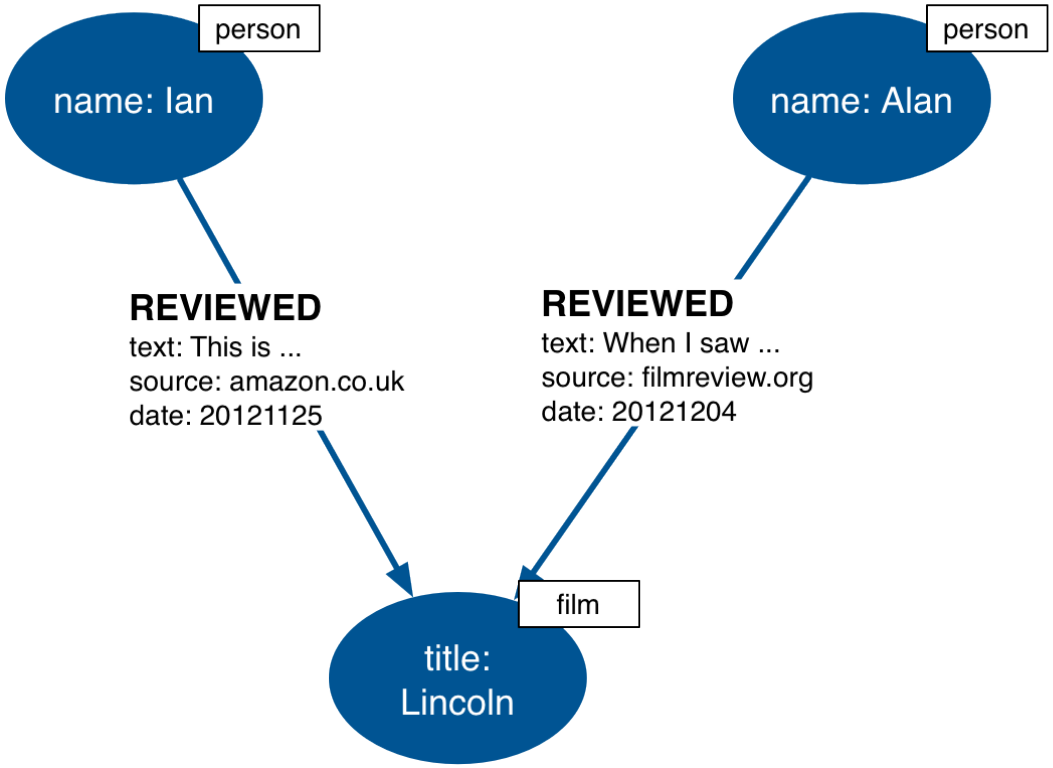
- Limits data model evolution
  - Unable to associate more entities
- Entities sometimes hidden in a verb
- Smells:
  - Lots of attribute-like properties
  - Property value redundancy
  - Heavy use of relationship indexes



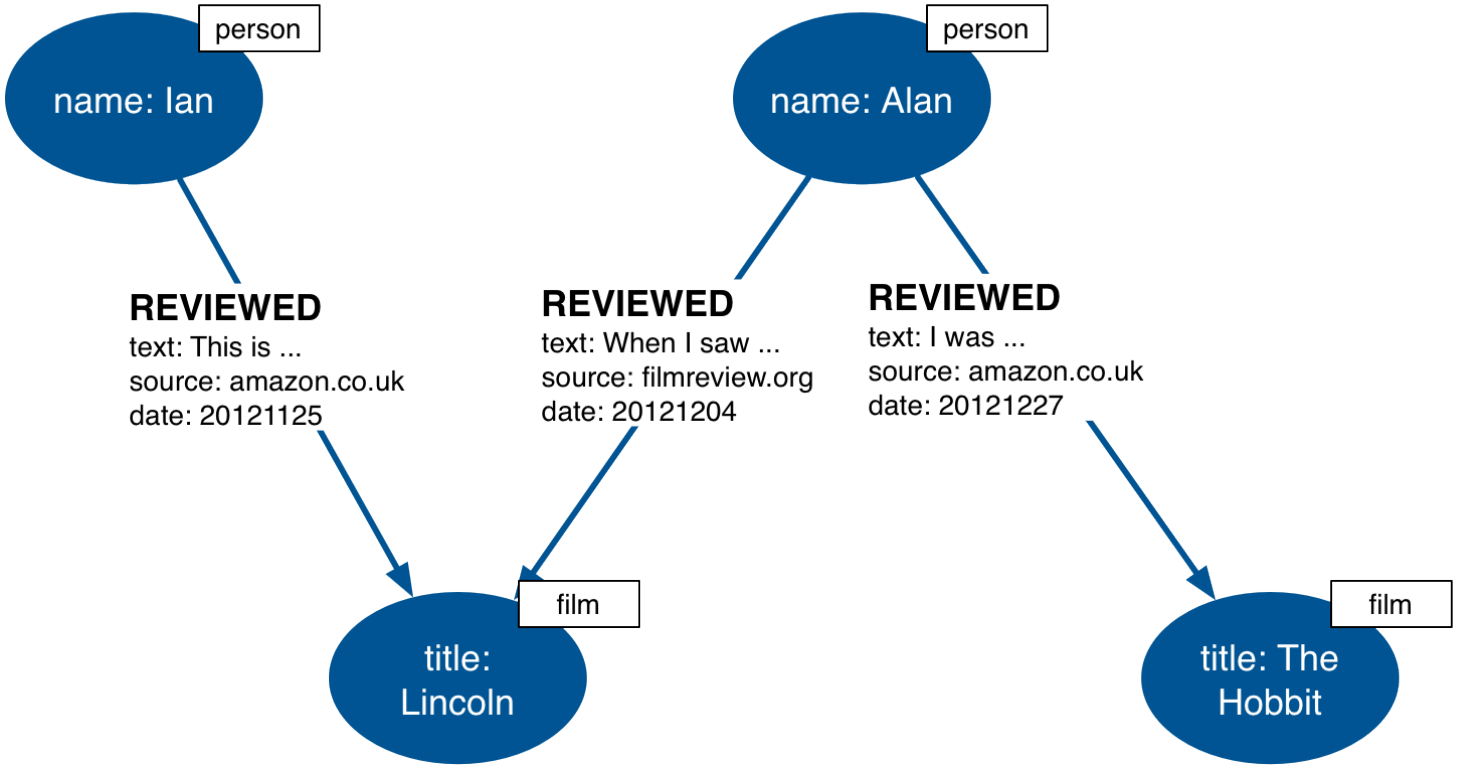
Example: Reviews



Add another review

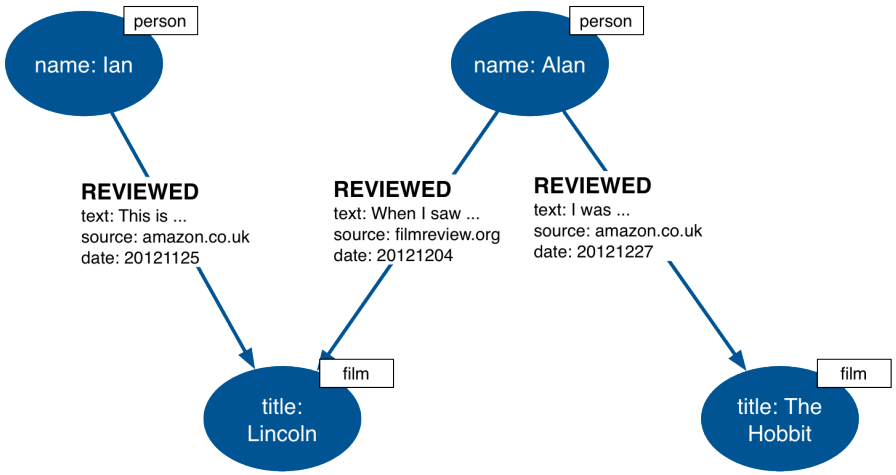


And another

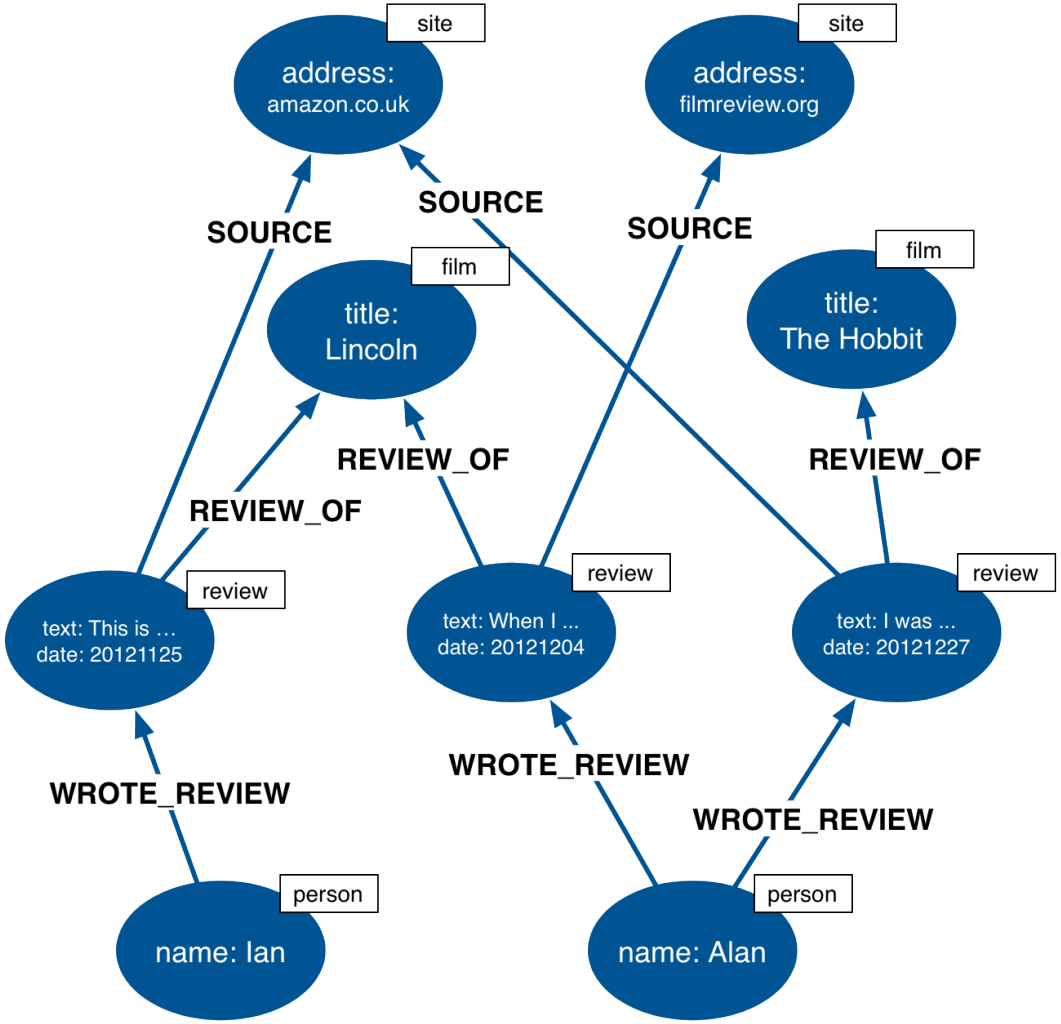

person
name: Ian
person
name: Alan
REVIEWED
text: This is ...
source: amazon.co.uk
date: 20121125
REVIEWED
text: When I saw ...
source: filmreview.org
date: 20121204
REVIEWED
text: I was ...
source: amazon.co.uk
date: 20121227
film
title: Lincoln
film
title: The Hobbit

Problems

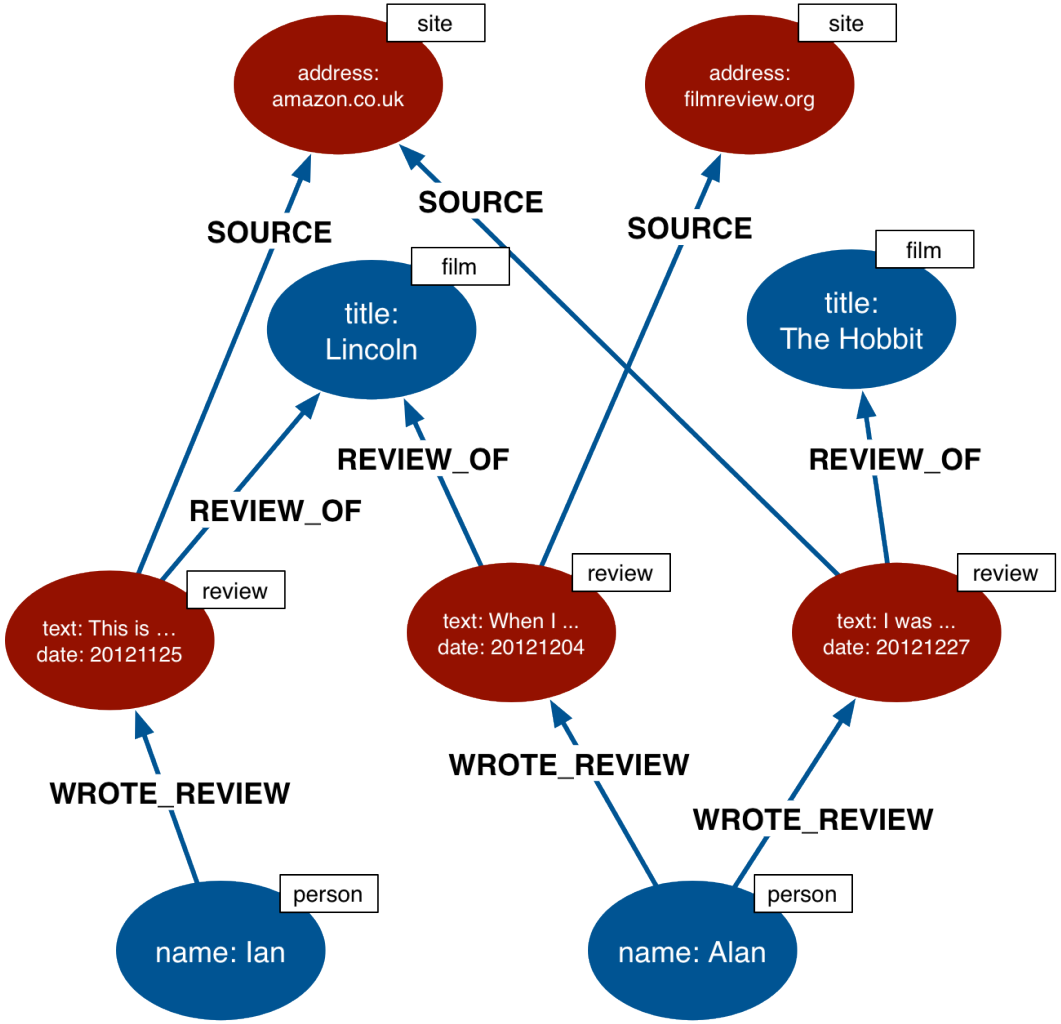
- Redundant data (2 x amazon.co.uk)
- Difficult to find reviews for source
- Users can't comment on reviews



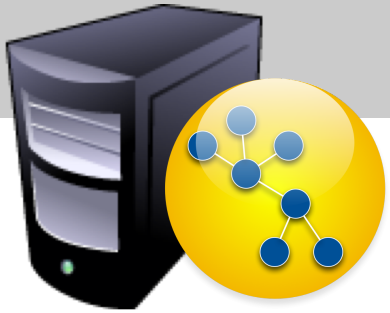
Revised model



Model actions in terms of products

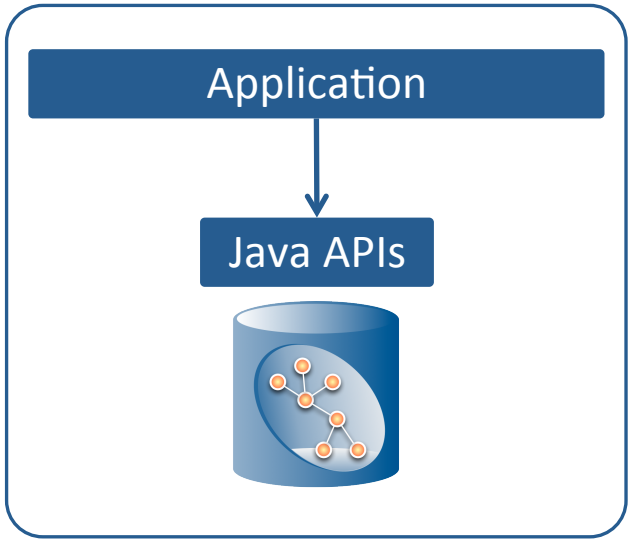


Application architectures



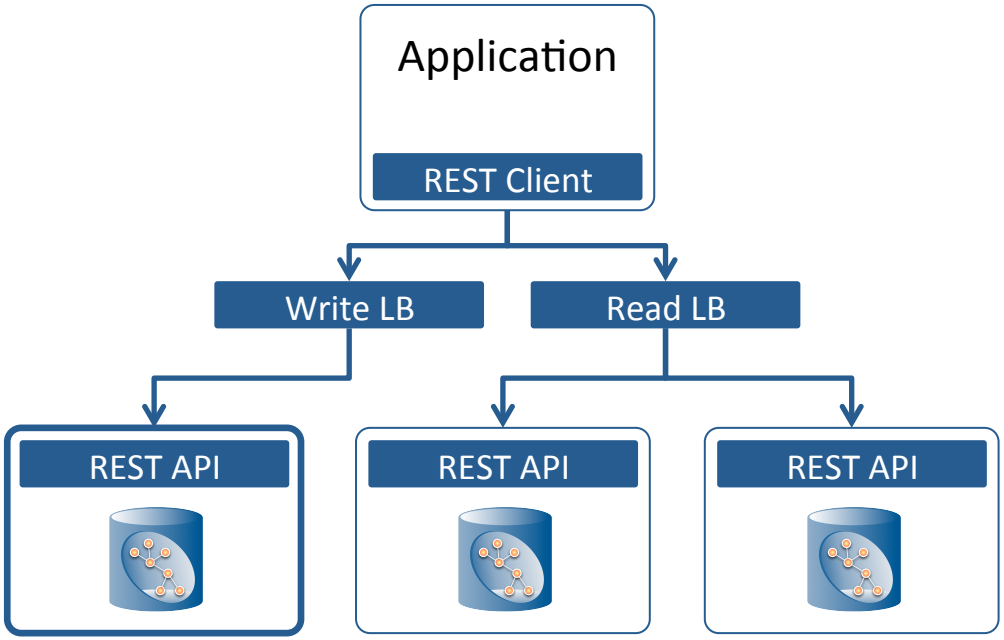
Embedded

- Host in Java process
- Access to Java APIs



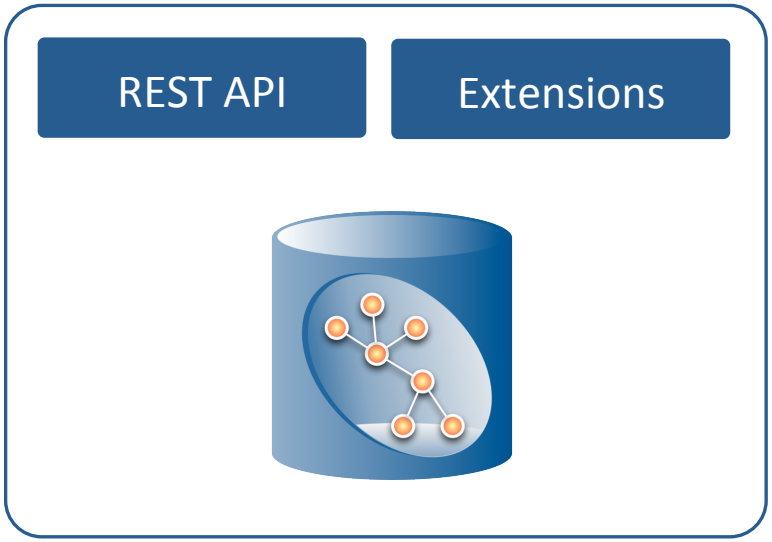
Server

- HTTP/JSON interface
- Server wraps embedded instance

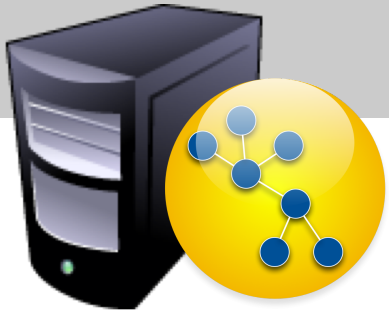


Server Extensions

- Encapsulate complex server-side logic
- Control HTTP request/response format

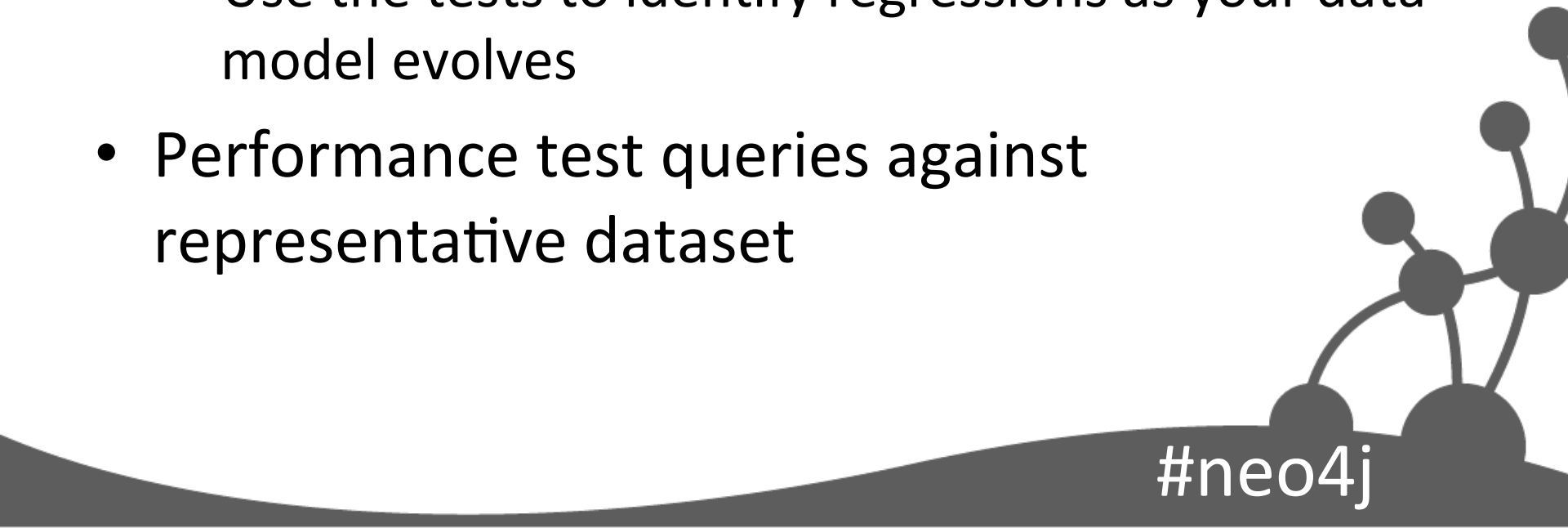


Testing

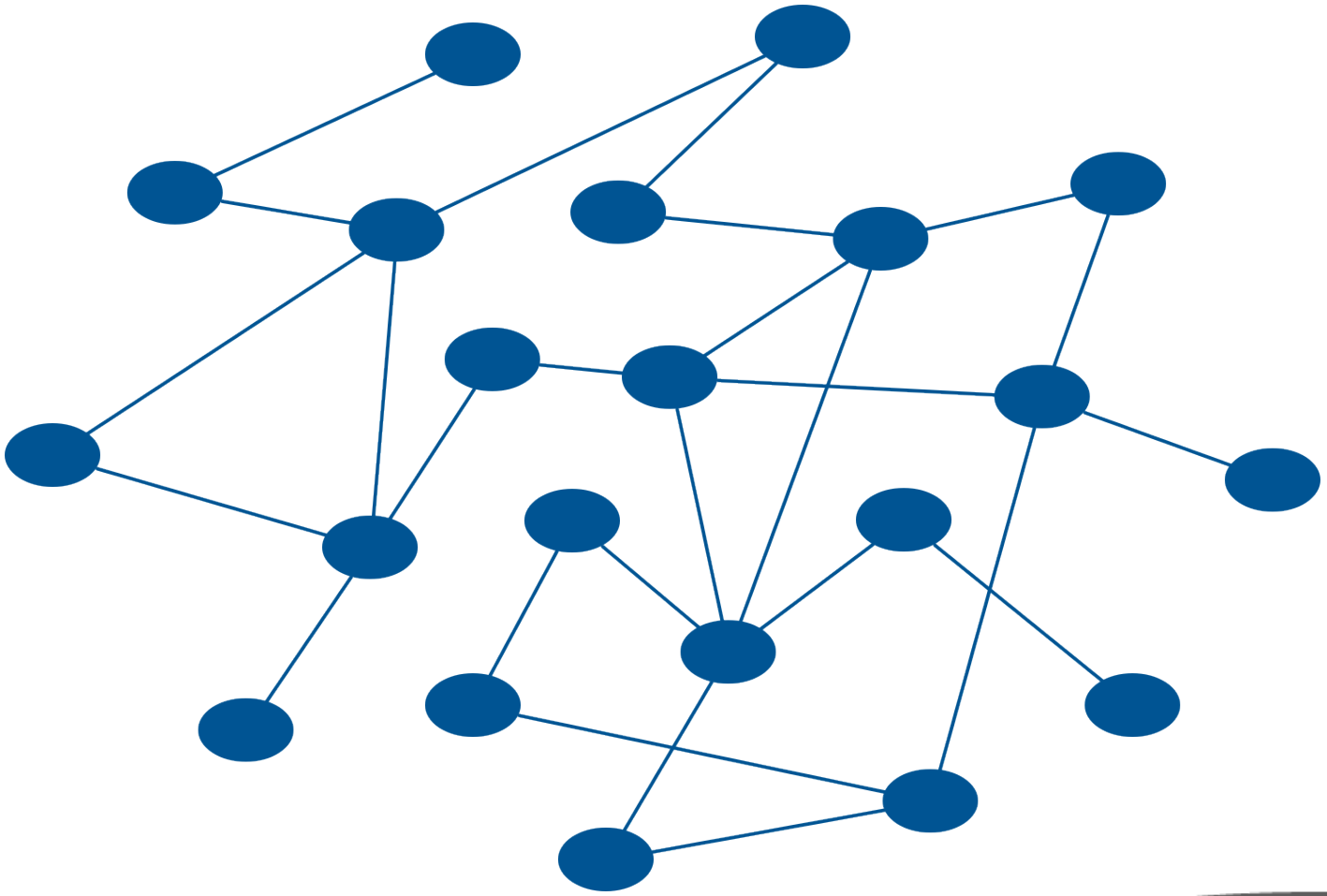


Test-driven data modeling

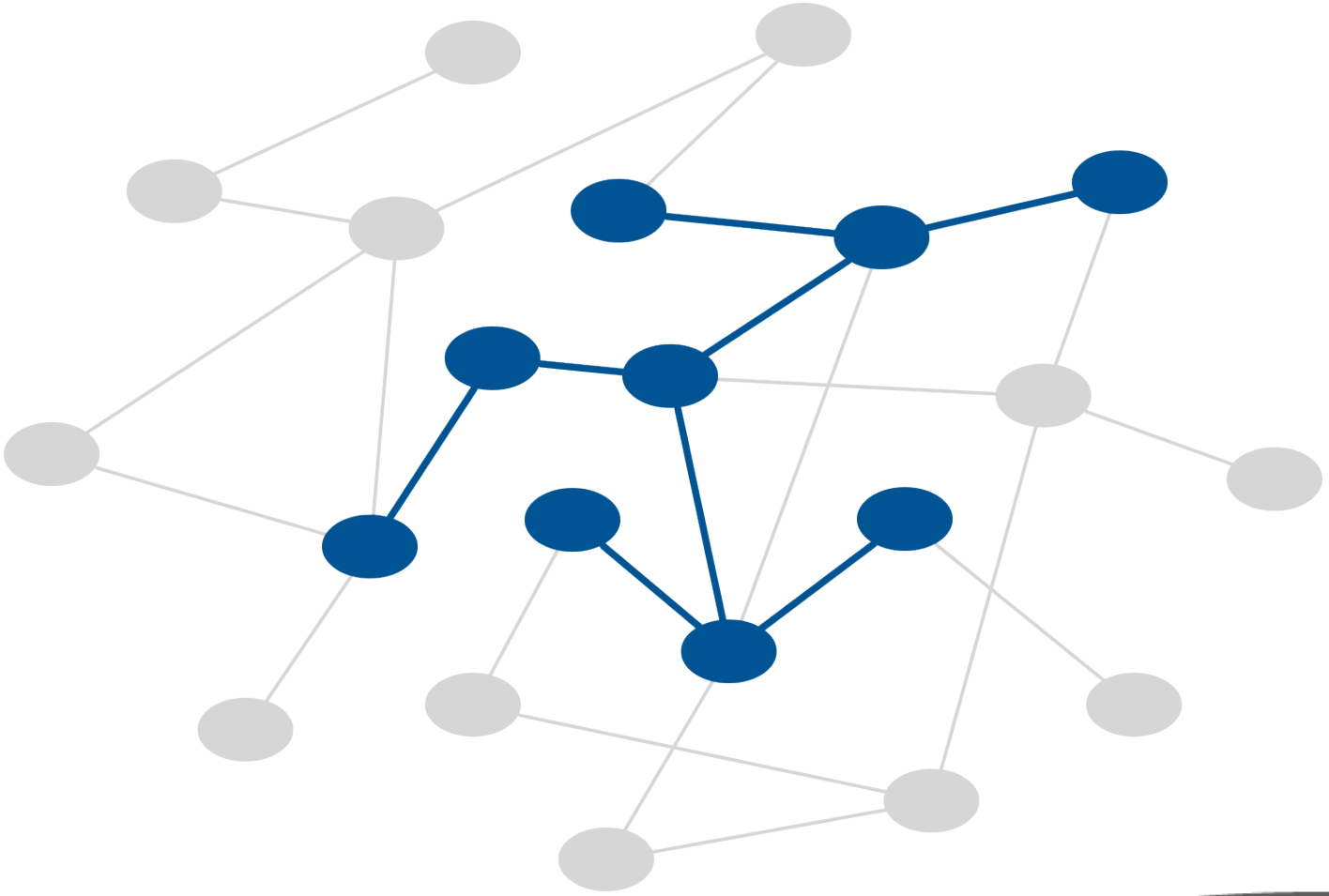
- Unit test with small, well-known datasets
  - Inject small graphs to test individual queries
  - Datasets express understanding of domain
  - Use the tests to identify regressions as your data model evolves
- Performance test queries against representative dataset



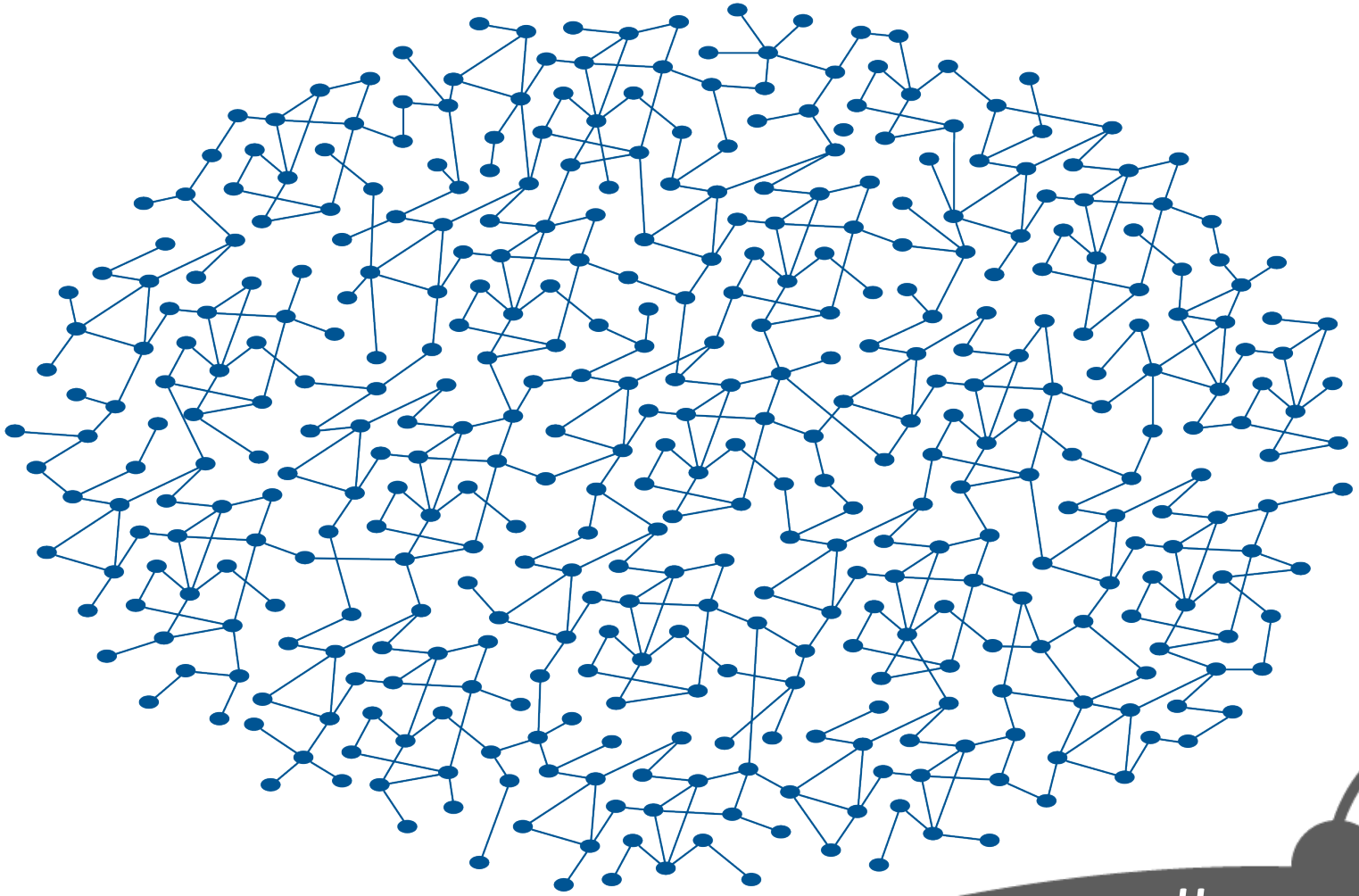
Query times proportional to size of subgraph searched



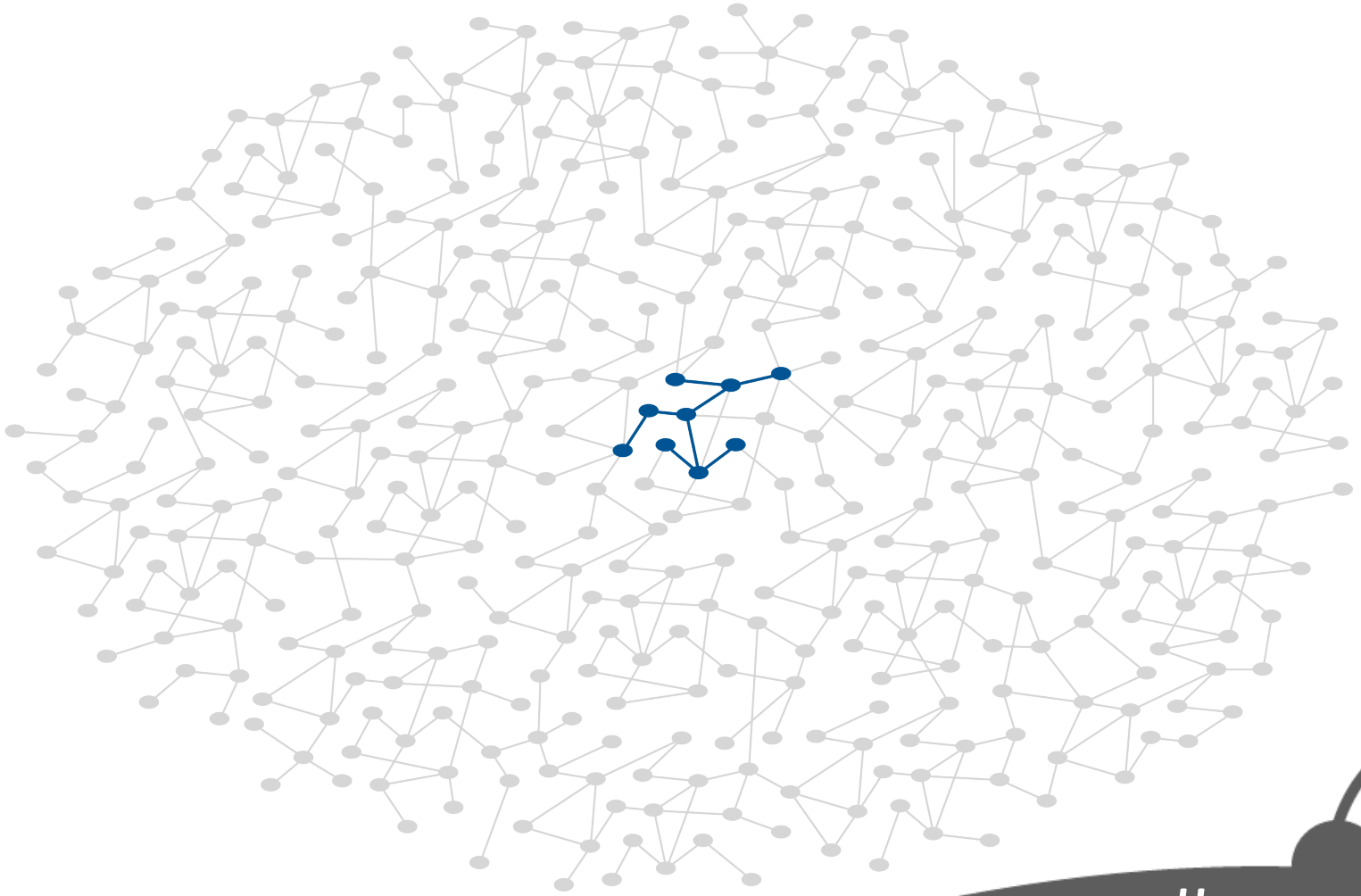
Query times proportional to size of subgraph searched



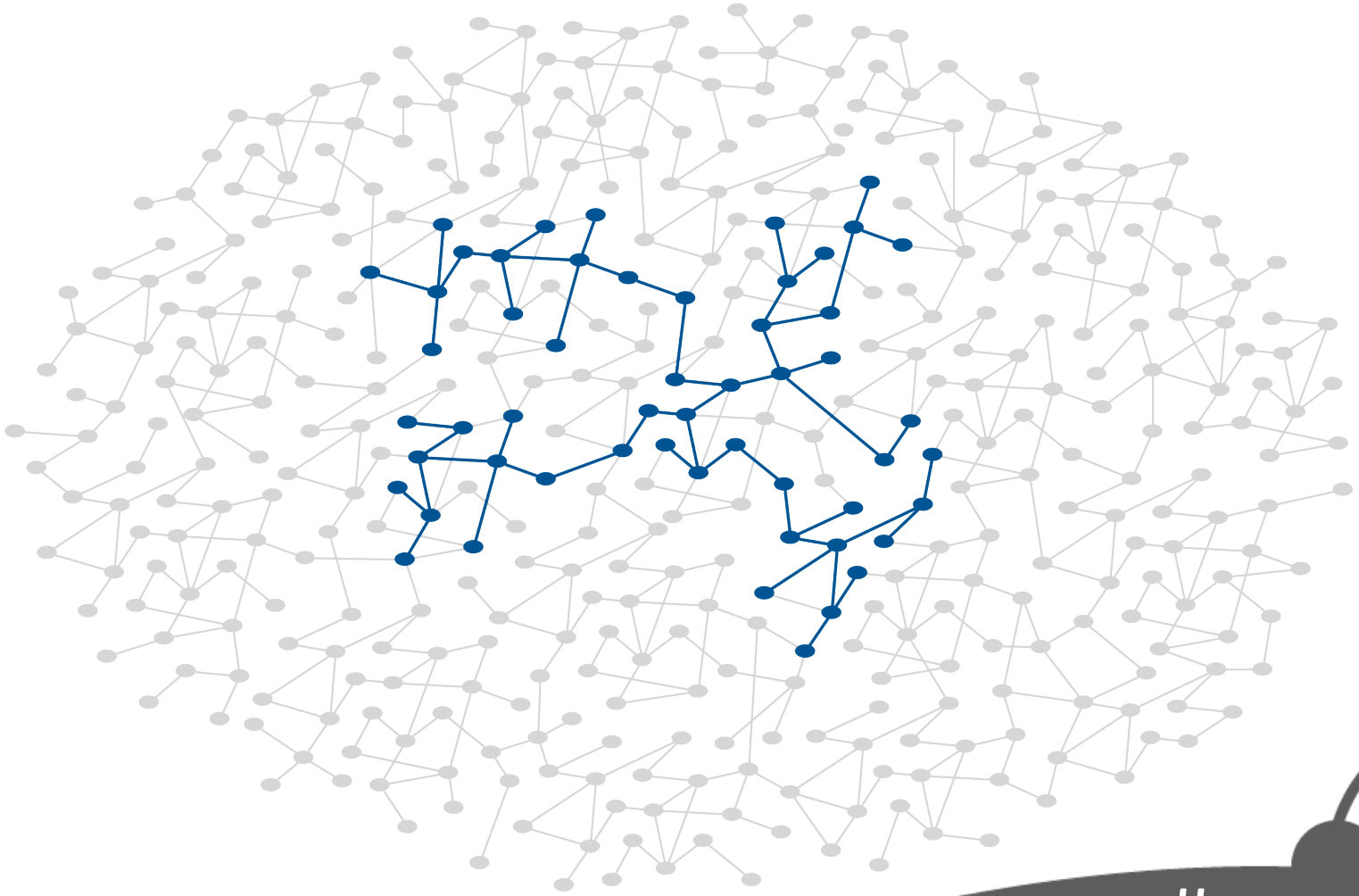
Query times proportional to size of subgraph searched



Query times remain constant …

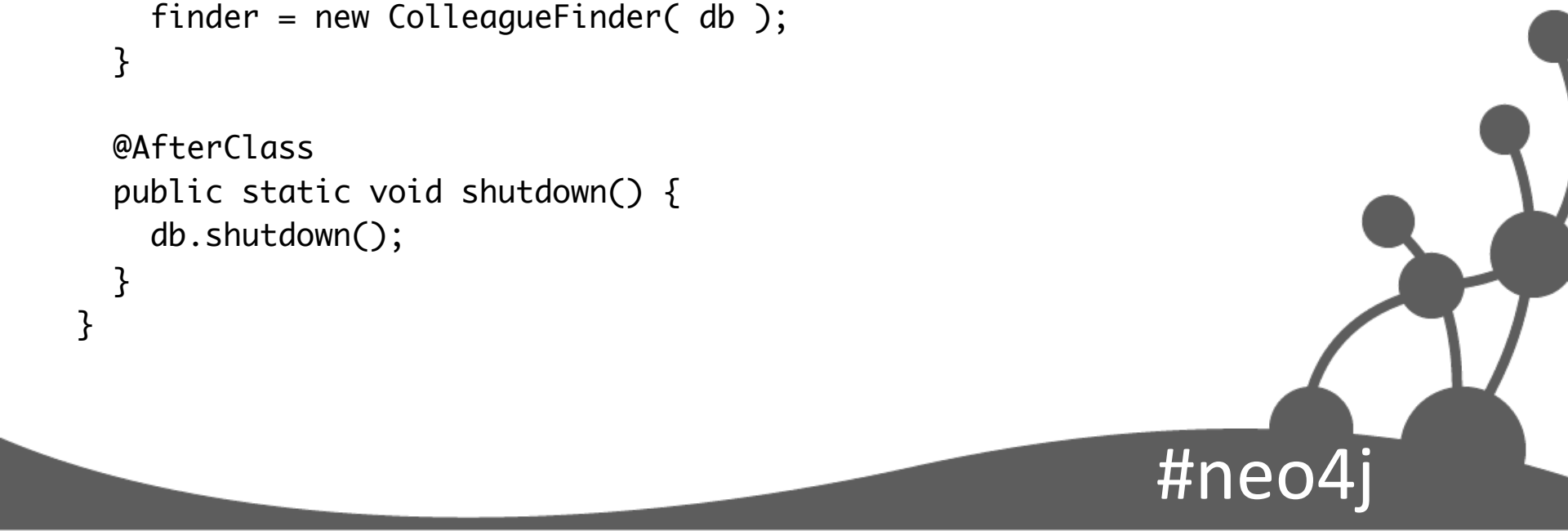


… unless subgraph seached grows



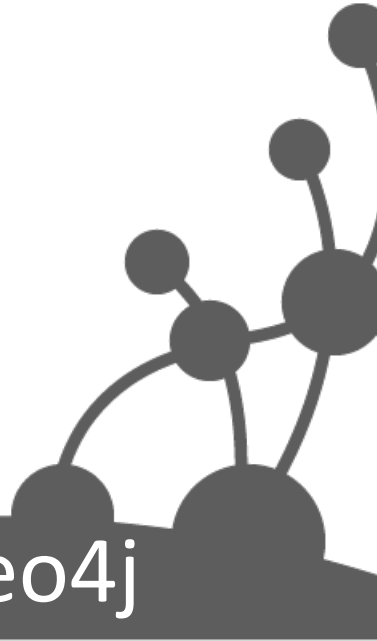
Unit test fixture

```
public class ColleagueFinderTest {

  private static GraphDatabaseService db;
  private static ColleagueFinder finder;

  @BeforeClass
  public static void init() {
    db = new TestGraphDatabaseFactory().newImpermanentDatabase();
    ExampleGraph.populate( db );
    finder = new ColleagueFinder( db );
  }

  @AfterClass
  public static void shutdown() {
    db.shutdown();
  }
}
```



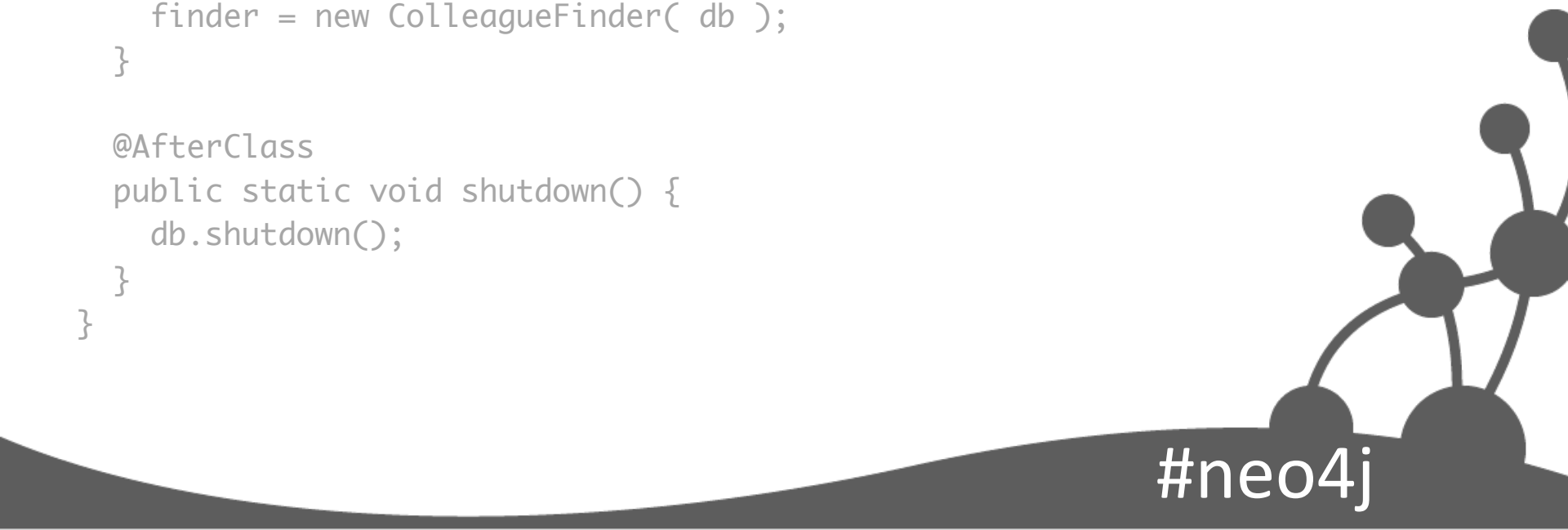
Create database

```
public class ColleagueFinderTest {

  private static GraphDatabaseService db;
  private static ColleagueFinder finder;

  @BeforeClass
  public static void init() {
    db = new TestGraphDatabaseFactory().newImpermanentDatabase();
    ExampleGraph.populate( db );
    finder = new ColleagueFinder( db );
  }

  @AfterClass
  public static void shutdown() {
    db.shutdown();
  }
}
```



Populate graph

```
public class ColleagueFinderTest {

  private static GraphDatabaseService db;
  private static ColleagueFinder finder;

  @BeforeClass
  public static void init() {
    db = new TestGraphDatabaseFactory().newImpermanentDatabase();
    ExampleGraph.populate( db );
    finder = new ColleagueFinder( db );
  }

  @AfterClass
  public static void shutdown() {
    db.shutdown();
  }
}
```



Create object under test

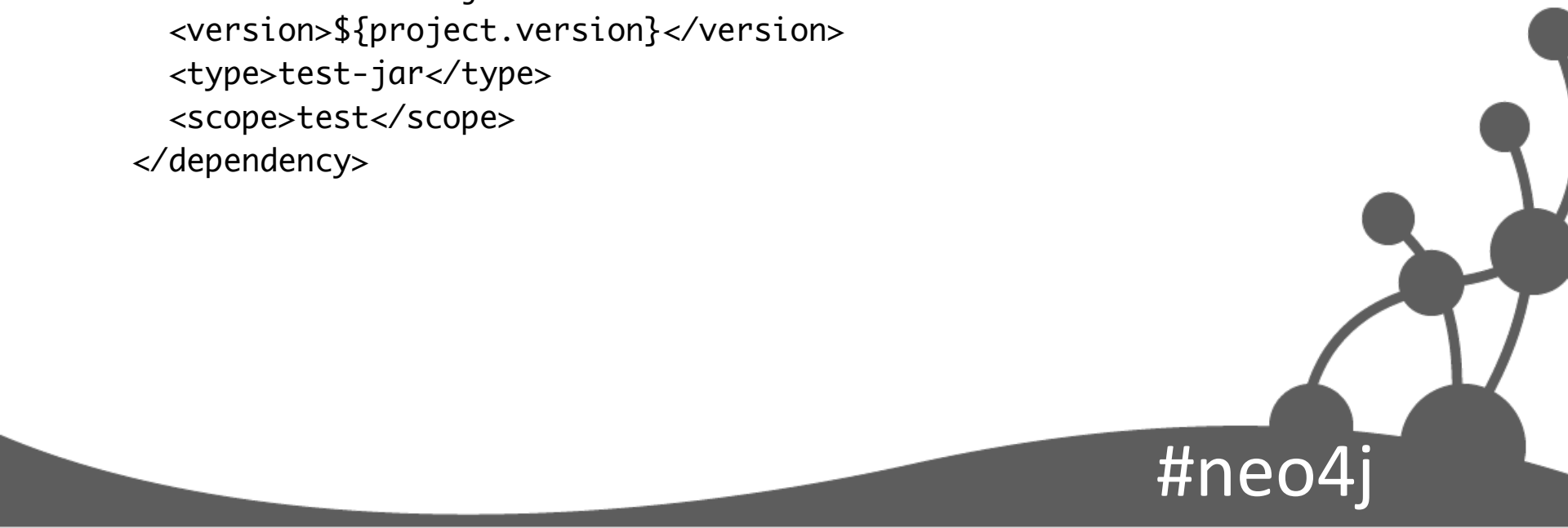
```
public class ColleagueFinderTest {

  private static GraphDatabaseService db;
  private static ColleagueFinder finder;

  @BeforeClass
  public static void init() {
    db = new TestGraphDatabaseFactory().newImpermanentDatabase();
    ExampleGraph.populate( db );
    finder = new ColleagueFinder( db );
  }

  @AfterClass
  public static void shutdown() {
    db.shutdown();
  }
}
```

Inject database

ImpermanentGraphDatabase

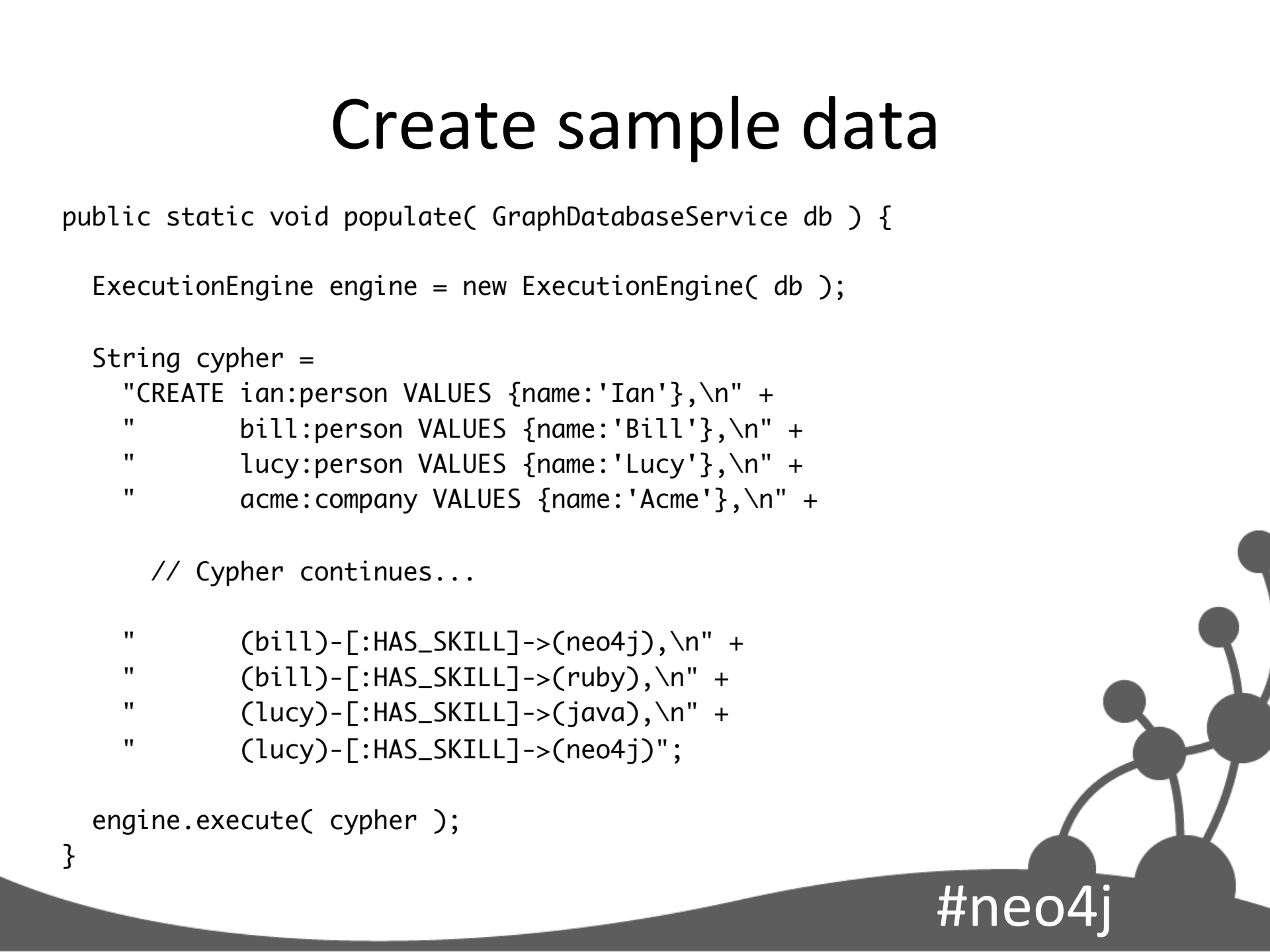
- In-memory
- For testing only

```
<dependency>
  <groupId>org.neo4j</groupId>
  <artifactId>neo4j-kernel</artifactId>
  <version>${project.version}</version>
  <type>test-jar</type>
  <scope>test</scope>
</dependency>
```



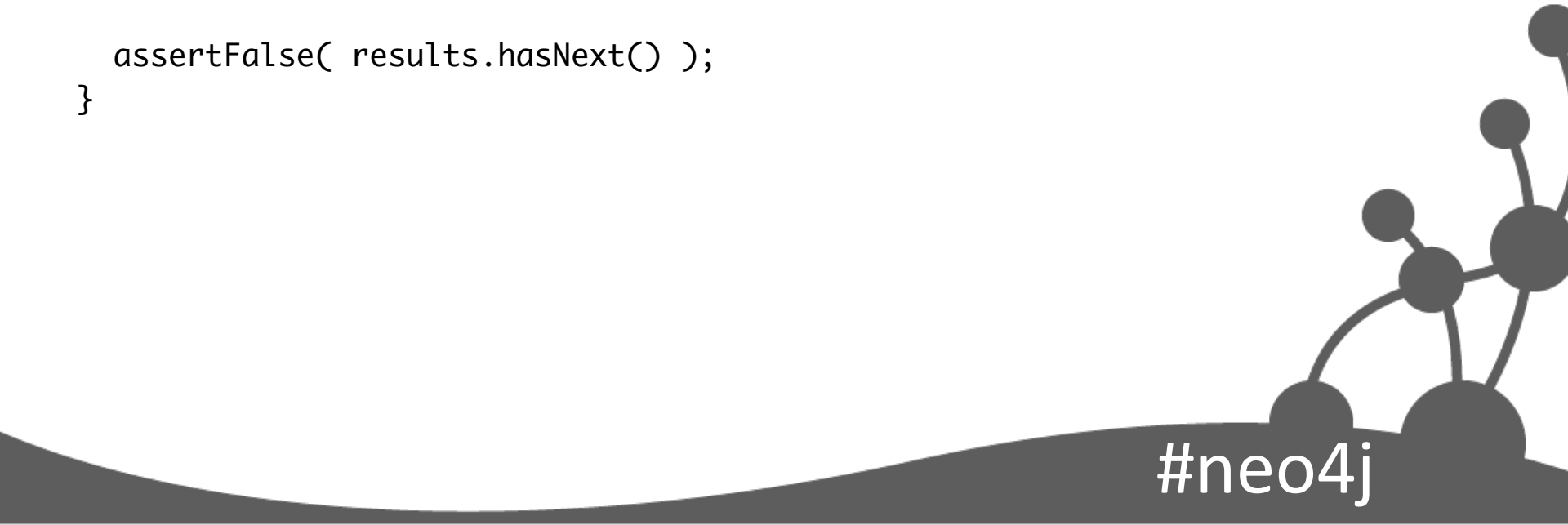
Create sample data

```
public static void populate( GraphDatabaseService db ) {

  ExecutionEngine engine = new ExecutionEngine( db );

  String cypher =
    "CREATE ian:person VALUES {name:'Ian'},\n" +
    "       bill:person VALUES {name:'Bill'},\n" +
    "       lucy:person VALUES {name:'Lucy'},\n" +
    "       acme:company VALUES {name:'Acme'},\n" +

      // Cypher continues...

    "       (bill)-[:HAS_SKILL]->(neo4j),\n" +
    "       (bill)-[:HAS_SKILL]->(ruby),\n" +
    "       (lucy)-[:HAS_SKILL]->(java),\n" +
    "       (lucy)-[:HAS_SKILL]->(neo4j)";

  engine.execute( cypher );
}
```



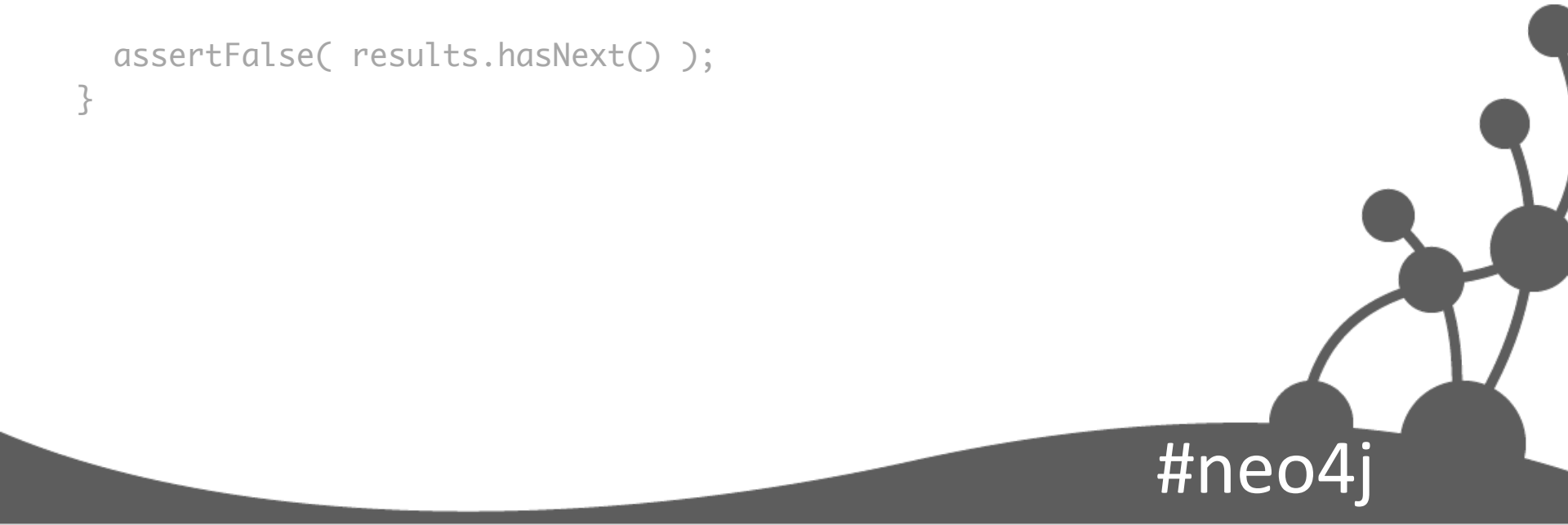
Unit test

```
@Test
public void shouldFindColleaguesWithSimilarSkills() throws Exception {

  // when
  Iterator<Map<String, Object>> results = finder.findFor( "Ian" );

  // then
  assertEquals( "Lucy", results.next().get( "name" ) );
  assertEquals( "Bill", results.next().get( "name" ) );

  assertFalse( results.hasNext() );
}
```



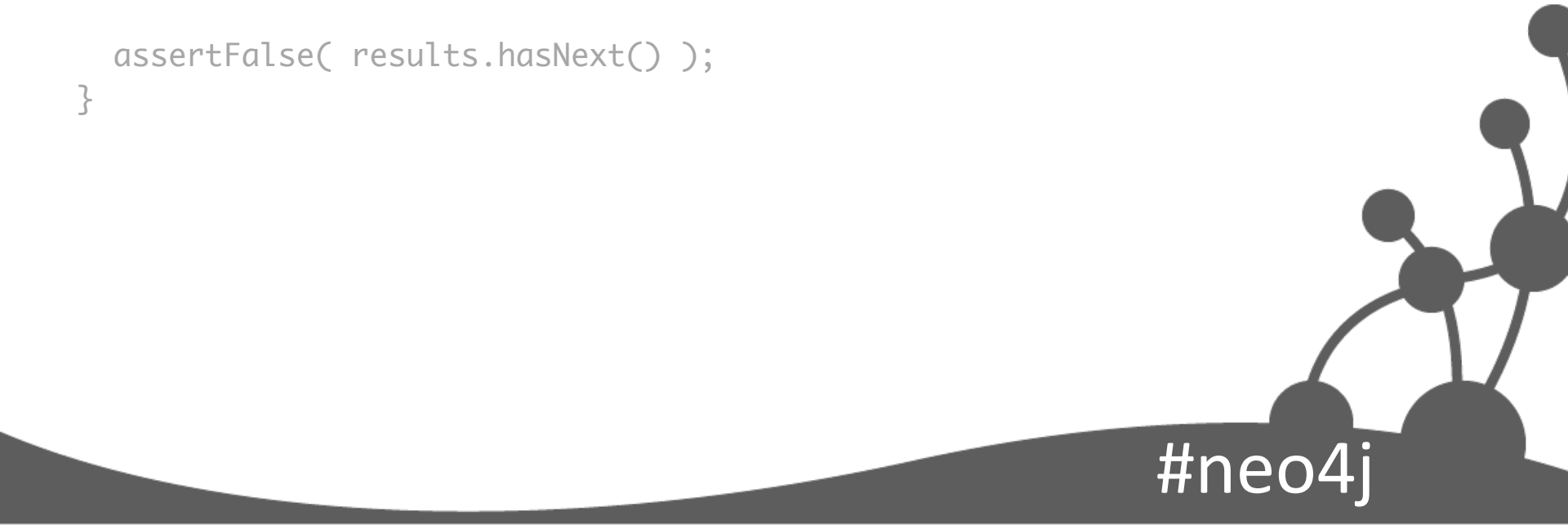
Execute unit under test

```
@Test
public void shouldFindColleaguesWithSimilarSkills() throws Exception {

  // when
  Iterator<Map<String, Object>> results = finder.findFor( "Ian" );

  // then
  assertEquals( "Lucy", results.next().get( "name" ) );
  assertEquals( "Bill", results.next().get( "name" ) );

  assertFalse( results.hasNext() );
}
```



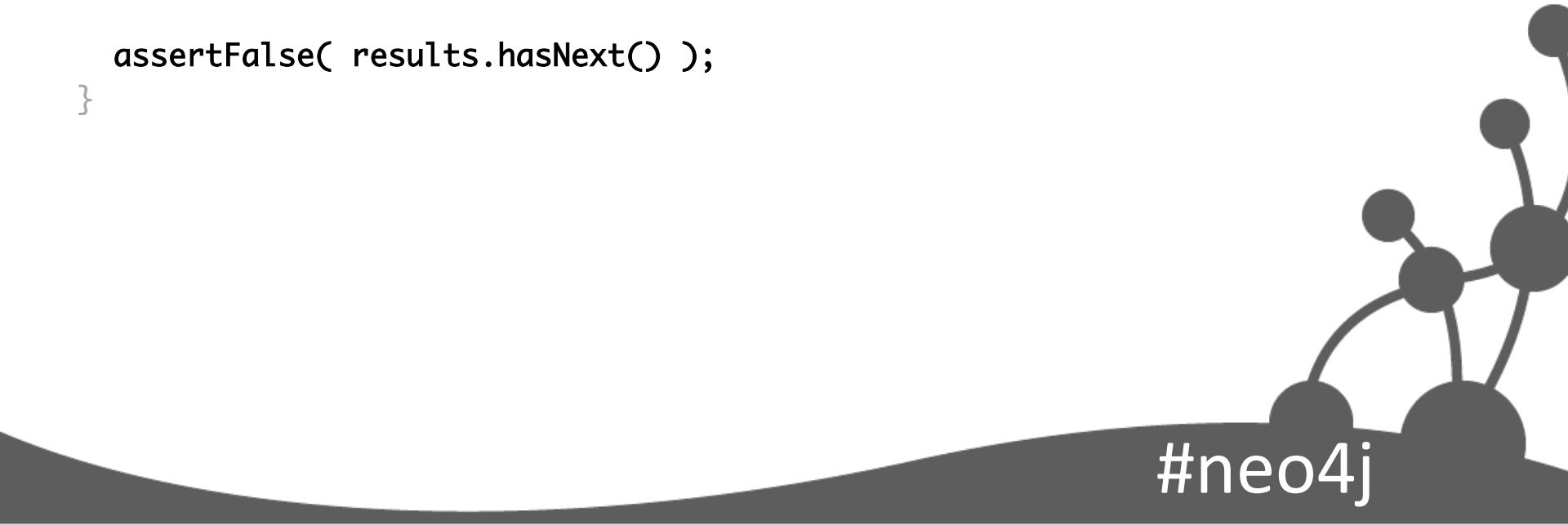
Assert results

```
@Test
public void shouldFindColleaguesWithSimilarSkills() throws Exception {

  // when
  Iterator<Map<String, Object>> results = finder.findFor( "Ian" );

  // then
  assertEquals( "Lucy", results.next().get( "name" ) );
  assertEquals( "Bill", results.next().get( "name" ) );

  assertFalse( results.hasNext() );
}
```



Ensure no more results

```
@Test
public void shouldFindColleaguesWithSimilarSkills() throws Exception {

  // when
  Iterator<Map<String, Object>> results = finder.findFor( "Ian" );

  // then
  assertEquals( "Lucy", results.next().get( "name" ) );
  assertEquals( "Bill", results.next().get( "name" ) );

  assertFalse( results.hasNext() );
}
```



Object under test

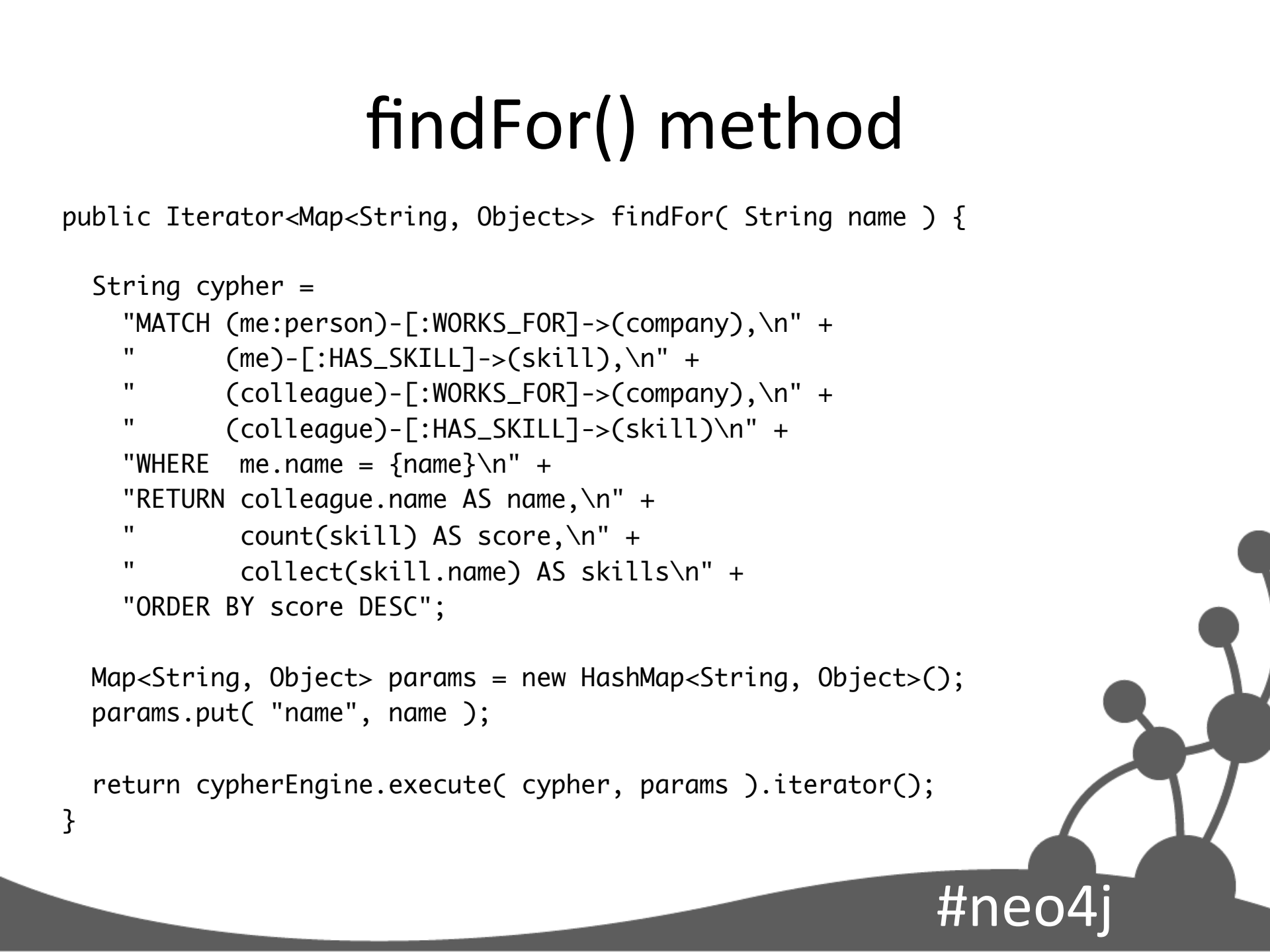
```
public class ColleagueFinder {

  private final ExecutionEngine cypherEngine;

  public ColleagueFinder( GraphDatabaseService db ) {
      this.cypherEngine = new ExecutionEngine( db );
  }

  public Iterator<Map<String, Object>> findFor( String name ) {
      ...
  }
}
```

Inject database

```
public class ColleagueFinder {

  private final ExecutionEngine cypherEngine;

  public ColleagueFinder( GraphDatabaseService db ) {
      this.cypherEngine = new ExecutionEngine( db );
  }

  public Iterator<Map<String, Object>> findFor( String name ) {
      ...
  }
}
```

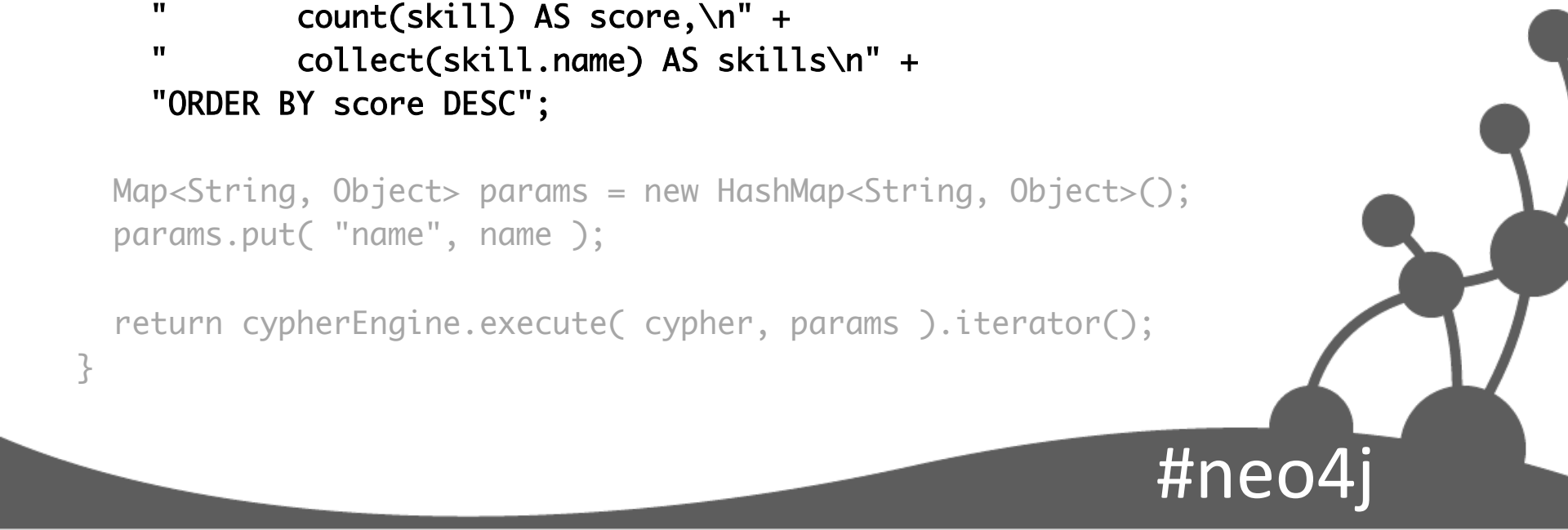
findFor() method

```
public Iterator<Map<String, Object>> findFor( String name ) {

  String cypher =
    "MATCH (me:person)-[:WORKS_FOR]->(company),\n" +
    "      (me)-[:HAS_SKILL]->(skill),\n" +
    "      (colleague)-[:WORKS_FOR]->(company),\n" +
    "      (colleague)-[:HAS_SKILL]->(skill)\n" +
    "WHERE  me.name = {name}\n" +
    "RETURN colleague.name AS name,\n" +
    "       count(skill) AS score,\n" +
    "       collect(skill.name) AS skills\n" +
    "ORDER BY score DESC";

  Map<String, Object> params = new HashMap<String, Object>();
  params.put( "name", name );

  return cypherEngine.execute( cypher, params ).iterator();
}
```



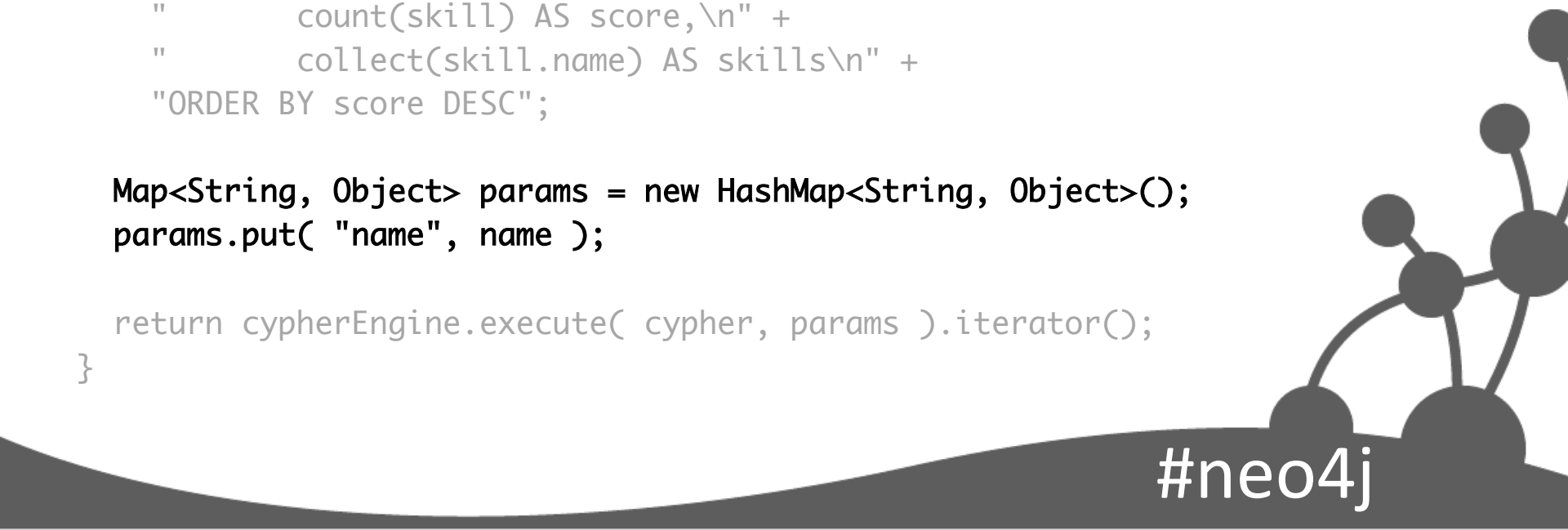
Cypher query

```
public Iterator<Map<String, Object>> findFor( String name ) {

  String cypher =
    "MATCH (me:person)-[:WORKS_FOR]->(company),\n" +
    "      (me)-[:HAS_SKILL]->(skill),\n" +
    "      (colleague)-[:WORKS_FOR]->(company),\n" +
    "      (colleague)-[:HAS_SKILL]->(skill)\n" +
    "WHERE  me.name = {name}\n" +
    "RETURN colleague.name AS name,\n" +
    "       count(skill) AS score,\n" +
    "       collect(skill.name) AS skills\n" +
    "ORDER BY score DESC";

  Map<String, Object> params = new HashMap<String, Object>();
  params.put( "name", name );

  return cypherEngine.execute( cypher, params ).iterator();
}
```



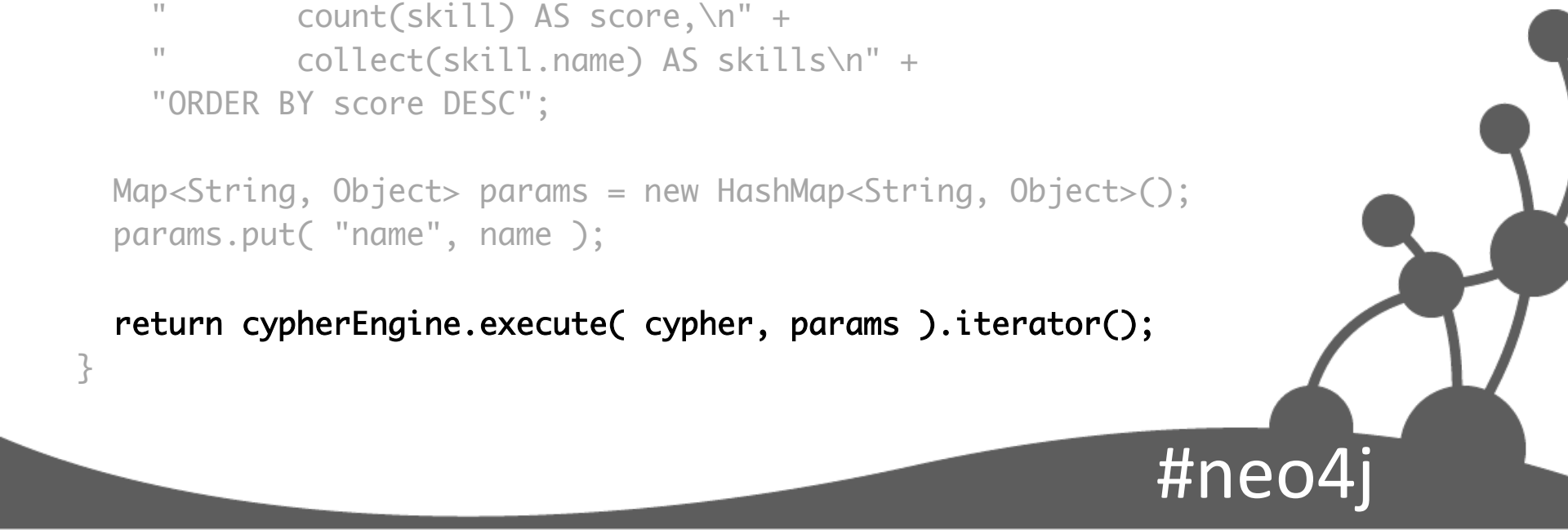
Parameterized query

```
public Iterator<Map<String, Object>> findFor( String name ) {

  String cypher =
    "MATCH (me:person)-[:WORKS_FOR]->(company),\n" +
    "      (me)-[:HAS_SKILL]->(skill),\n" +
    "      (colleague)-[:WORKS_FOR]->(company),\n" +
    "      (colleague)-[:HAS_SKILL]->(skill)\n" +
    "WHERE  me.name = {name}\n" +
    "RETURN colleague.name AS name,\n" +
    "       count(skill) AS score,\n" +
    "       collect(skill.name) AS skills\n" +
    "ORDER BY score DESC";

  Map<String, Object> params = new HashMap<String, Object>();
  params.put( "name", name );

  return cypherEngine.execute( cypher, params ).iterator();
}
```



Execute query

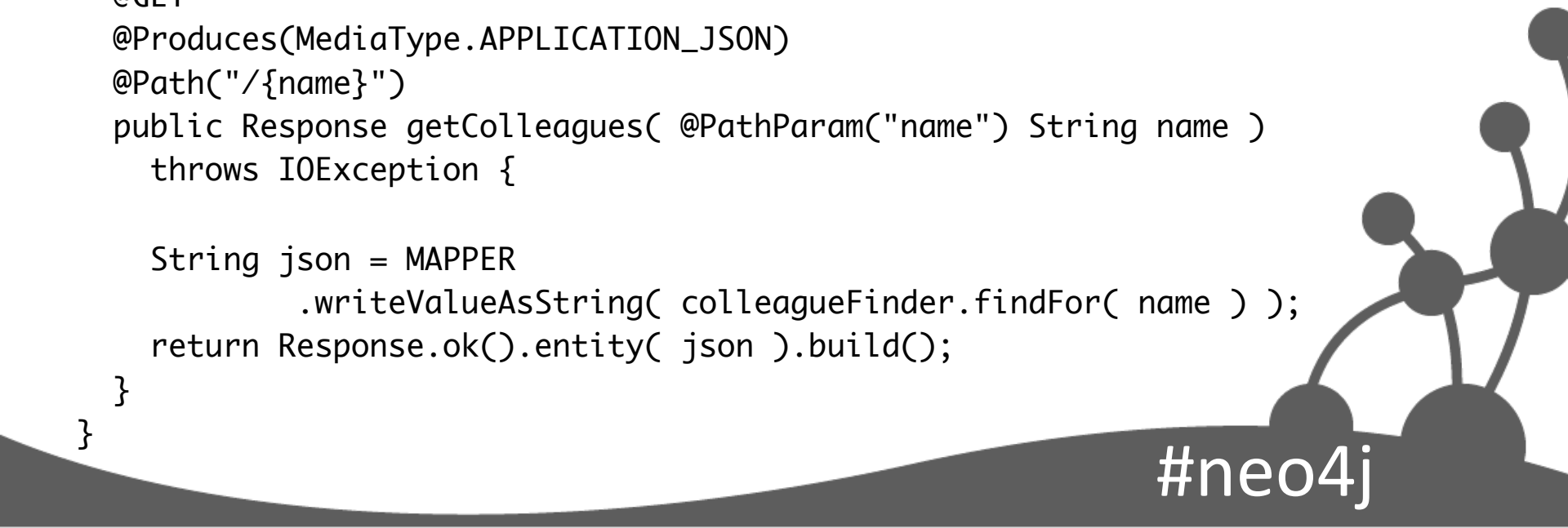
```
public Iterator<Map<String, Object>> findFor( String name ) {

  String cypher =
    "MATCH (me:person)-[:WORKS_FOR]->(company),\n" +
    "      (me)-[:HAS_SKILL]->(skill),\n" +
    "      (colleague)-[:WORKS_FOR]->(company),\n" +
    "      (colleague)-[:HAS_SKILL]->(skill)\n" +
    "WHERE  me.name = {name}\n" +
    "RETURN colleague.name AS name,\n" +
    "       count(skill) AS score,\n" +
    "       collect(skill.name) AS skills\n" +
    "ORDER BY score DESC";

  Map<String, Object> params = new HashMap<String, Object>();
  params.put( "name", name );

  return cypherEngine.execute( cypher, params ).iterator();
}
```



#neo4j

Unmanaged extension

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
            .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

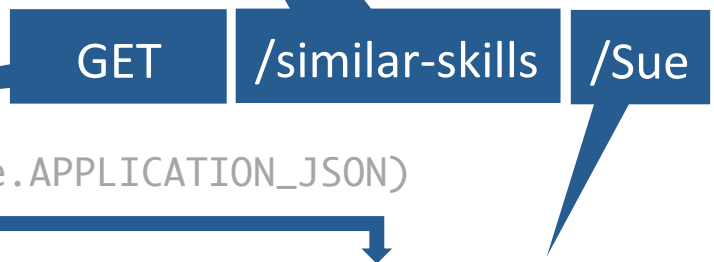


JAX-RS annotations

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
            .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

Map HTTP request to object+method

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
            .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```



Database injected by server

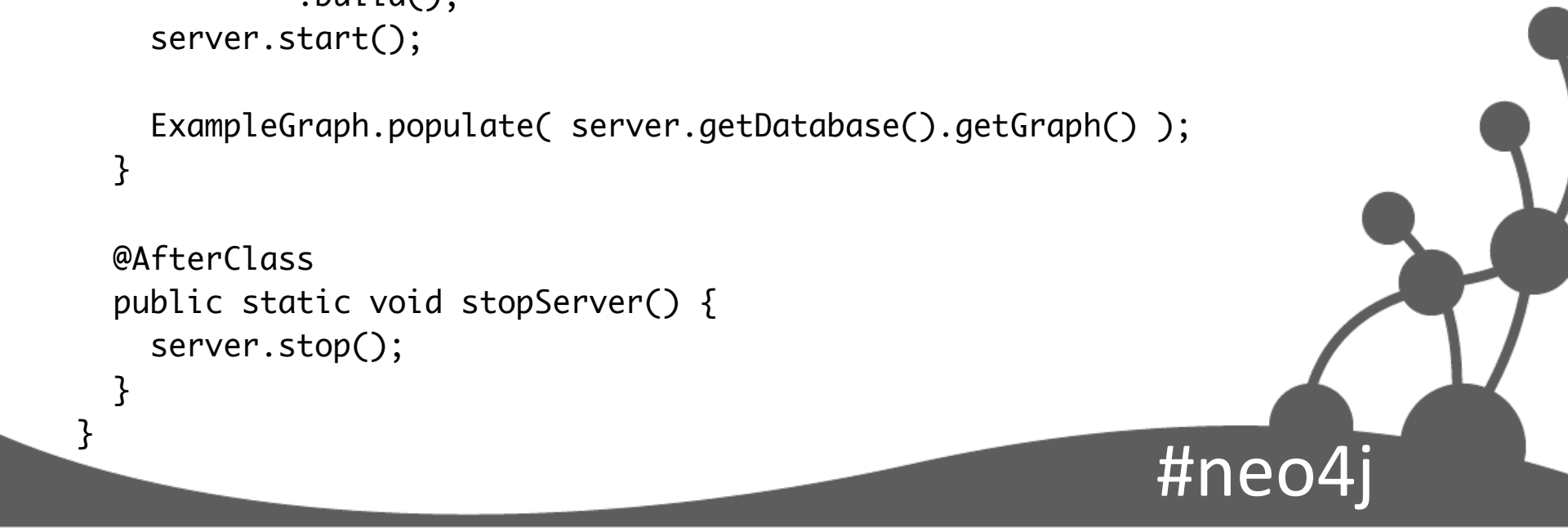
```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
            .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

Generate and format response

```
@Path("/similar-skills")
public class ColleagueFinderExtension {
  private static final ObjectMapper MAPPER = new ObjectMapper();
  private final ColleagueFinder colleagueFinder;

  public ColleagueFinderExtension( @Context GraphDatabaseService db ) {
    this.colleagueFinder = new ColleagueFinder( db );
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{name}")
  public Response getColleagues( @PathParam("name") String name )
    throws IOException {

    String json = MAPPER
            .writeValueAsString( colleagueFinder.findFor( name ) );
    return Response.ok().entity( json ).build();
  }
}
```

Extension test fixture

```
public class ColleagueFinderExtensionTest {
  private static CommunityNeoServer server;

  @BeforeClass
  public static void startServer() throws IOException
  {
    server = ServerBuilder.server()
            .withThirdPartyJaxRsPackage(
                    "org.neo4j.good_practices", "/colleagues" )
            .build();
    server.start();

    ExampleGraph.populate( server.getDatabase().getGraph() );
  }

  @AfterClass
  public static void stopServer() {
    server.stop();
  }
}
```



Build and configure server

```
public class ColleagueFinderExtensionTest {
  private static CommunityNeoServer server;

  @BeforeClass
  public static void startServer() throws IOException
  {
    server = ServerBuilder.server()
            .withThirdPartyJaxRsPackage(
                    "org.neo4j.good_practices", "/colleagues" )
            .build();
    server.start();

    ExampleGraph.populate( server.getDatabase().getGraph() );
  }

  @AfterClass
  public static void stopServer() {
    server.stop();
  }
}
```

Start server

```
public class ColleagueFinderExtensionTest {
  private static CommunityNeoServer server;

  @BeforeClass
  public static void startServer() throws IOException
  {
    server = ServerBuilder.server()
            .withThirdPartyJaxRsPackage(
                    "org.neo4j.good_practices", "/colleagues" )
            .build();
    server.start();

    ExampleGraph.populate( server.getDatabase().getGraph() );
  }

  @AfterClass
  public static void stopServer() {
    server.stop();
  }
}
```

Populate database

```
public class ColleagueFinderExtensionTest {
  private static CommunityNeoServer server;

  @BeforeClass
  public static void startServer() throws IOException
  {
    server = ServerBuilder.server()
            .withThirdPartyJaxRsPackage(
                    "org.neo4j.good_practices", "/colleagues" )
            .build();
    server.start();

    ExampleGraph.populate( server.getDatabase().getGraph() );
  }

  @AfterClass
  public static void stopServer() {
    server.stop();
  }
}
```

#neo4j

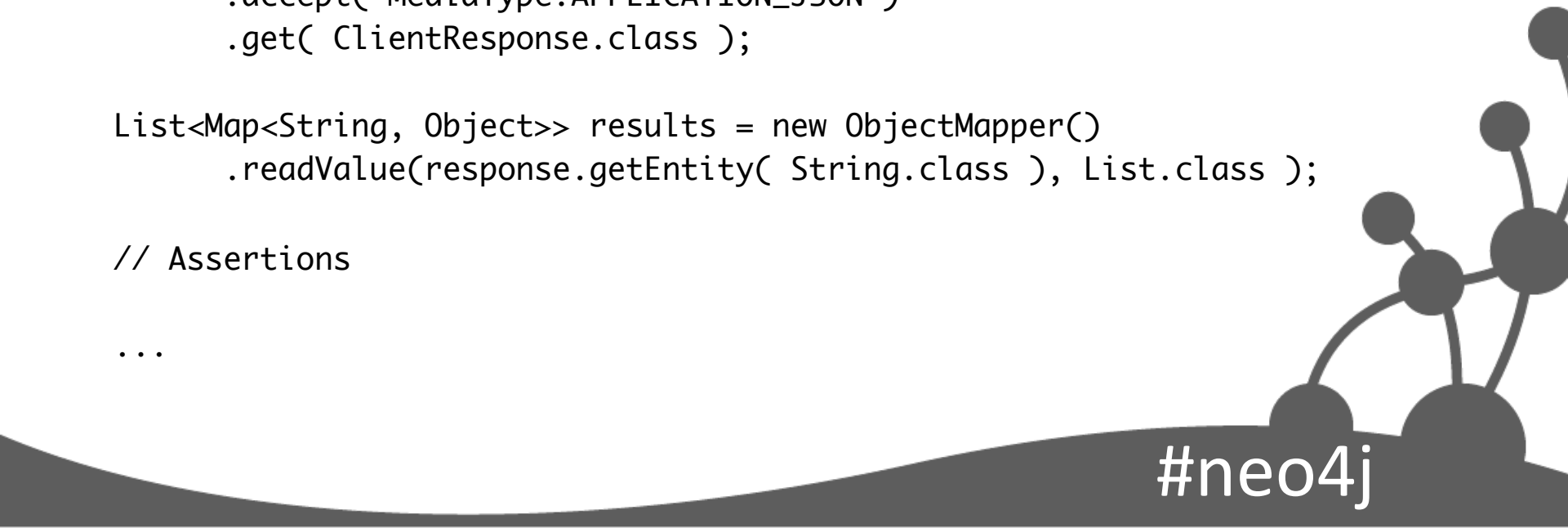
ServerBuilder

- Programmatic configuration

```
<dependency>
  <groupId>org.neo4j.app</groupId>
  <artifactId>neo4j-server</artifactId>
  <version>${project.version}</version>
  <type>test-jar</type>
</dependency>
```

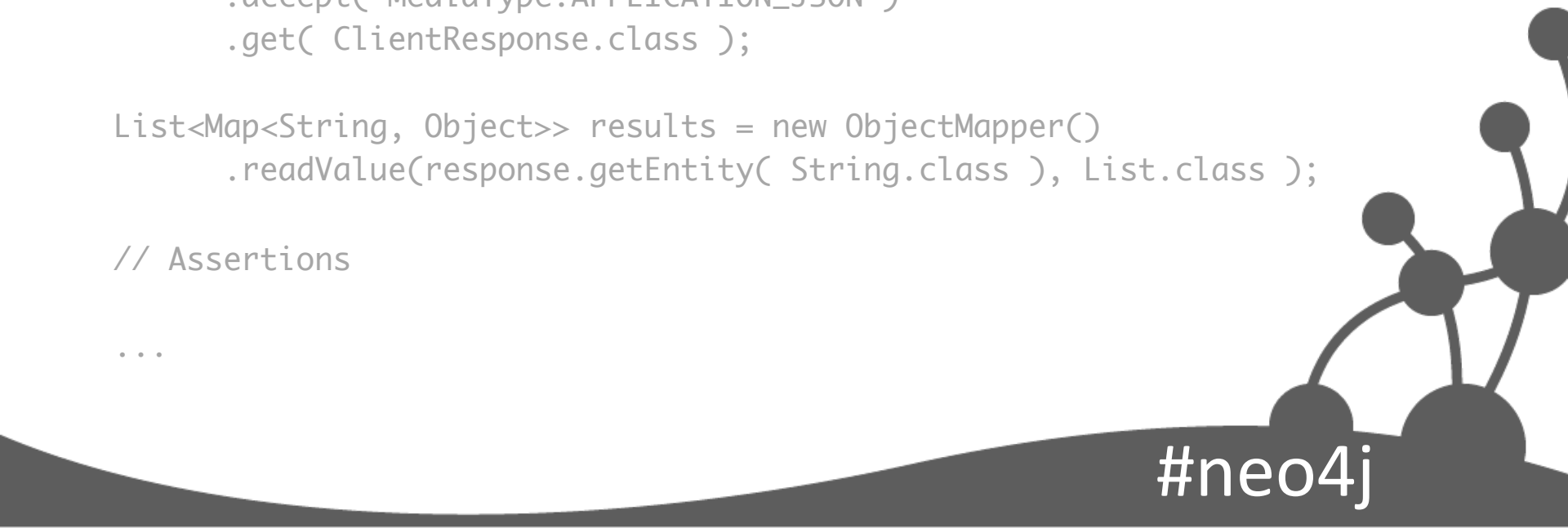
Testing extensions

```
@Test
public void shouldReturnColleaguesWithSimilarSkills() throws Exception {

  Client client = Client.create( new DefaultClientConfig() );

  WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

  ClientResponse response = resource
        .accept( MediaType.APPLICATION_JSON )
        .get( ClientResponse.class );

  List<Map<String, Object>> results = new ObjectMapper()
        .readValue(response.getEntity( String.class ), List.class );

  // Assertions

  ...
```



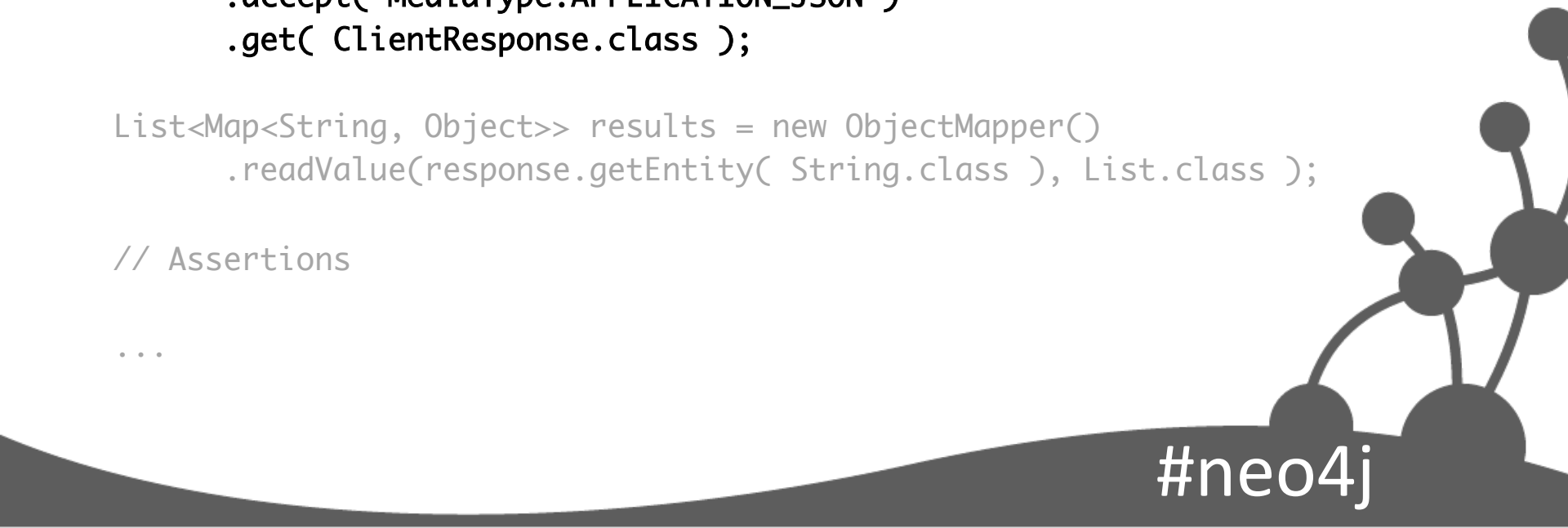
Create HTTP client

```
@Test
public void shouldReturnColleaguesWithSimilarSkills() throws Exception {

  Client client = Client.create( new DefaultClientConfig() );

  WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

  ClientResponse response = resource
        .accept( MediaType.APPLICATION_JSON )
        .get( ClientResponse.class );

  List<Map<String, Object>> results = new ObjectMapper()
        .readValue(response.getEntity( String.class ), List.class );

  // Assertions

  ...
```



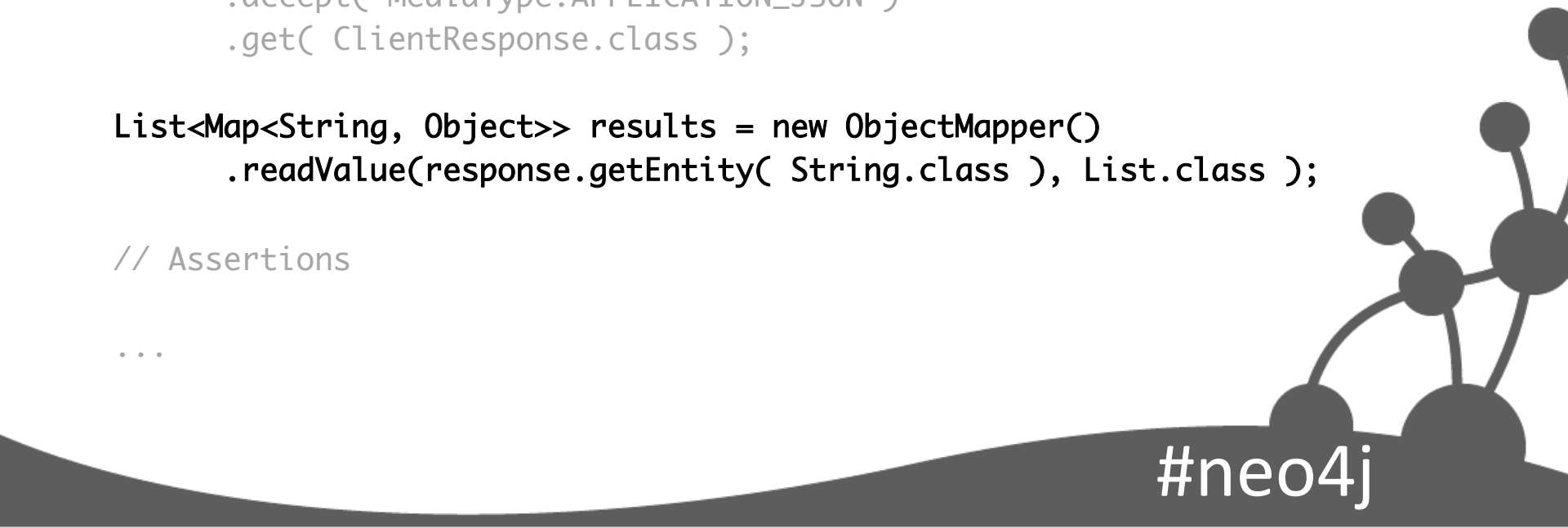
Issue request

```
@Test
public void shouldReturnColleaguesWithSimilarSkills() throws Exception {

  Client client = Client.create( new DefaultClientConfig() );

  WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

  ClientResponse response = resource
        .accept( MediaType.APPLICATION_JSON )
        .get( ClientResponse.class );

  List<Map<String, Object>> results = new ObjectMapper()
        .readValue(response.getEntity( String.class ), List.class );

  // Assertions

  ...
```



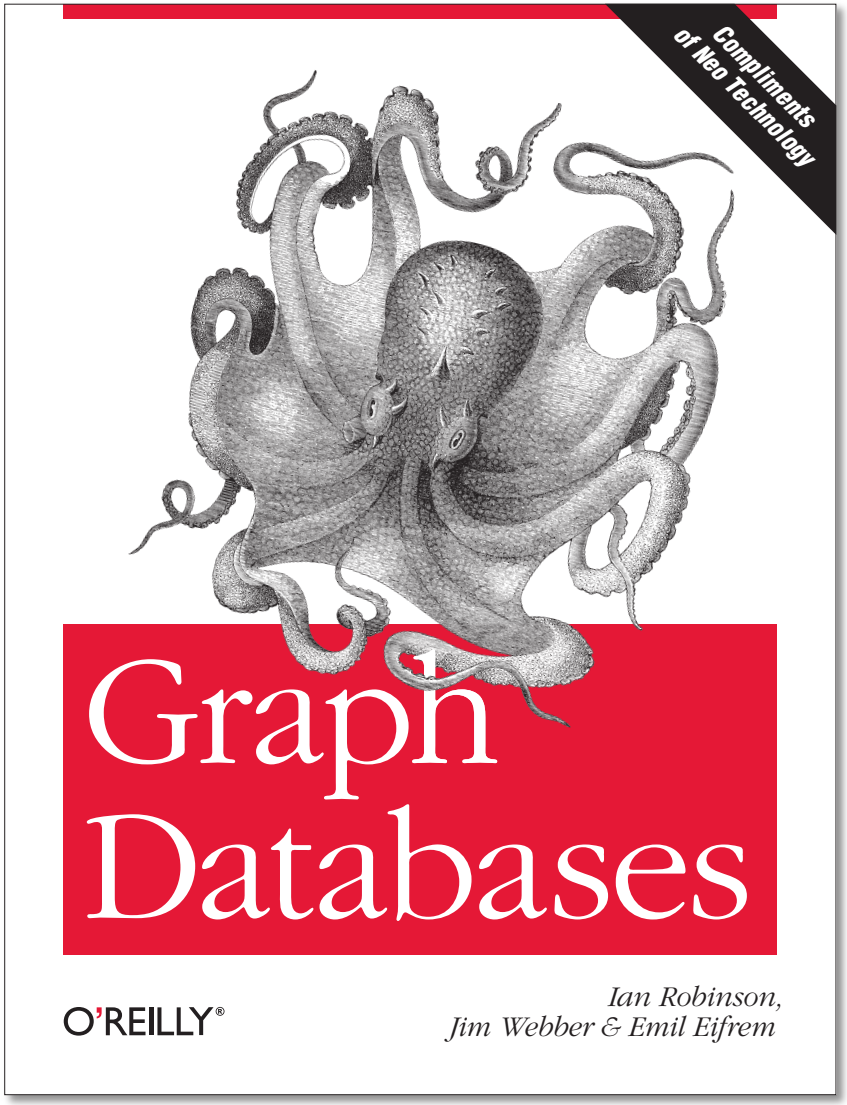
Parse response

```
@Test
public void shouldReturnColleaguesWithSimilarSkills() throws Exception {

  Client client = Client.create( new DefaultClientConfig() );

  WebResource resource = client
        .resource( "http://localhost:7474/colleagues/similar-skills/Ian" );

  ClientResponse response = resource
        .accept( MediaType.APPLICATION_JSON )
        .get( ClientResponse.class );

  List<Map<String, Object>> results = new ObjectMapper()
        .readValue(response.getEntity( String.class ), List.class );

  // Assertions

  ...
```



Assert results

```
  ...

  assertEquals( 200, response.getStatus() );
  assertEquals( MediaType.APPLICATION_JSON,
                response.getHeaders().get( "Content-Type" ).get( 0 ) );

  assertEquals( "Lucy", results.get( 0 ).get( "name" ) );
  assertThat( (Iterable<String>) results.get( 0 ).get( "skills" ),
              hasItems( "Java", "Neo4j" ) );
}
```



Thank you

@ianSrobinson
ian@neotechnology.com
github.com/iansrobinson

http://graphdatabases.com

#neo4j