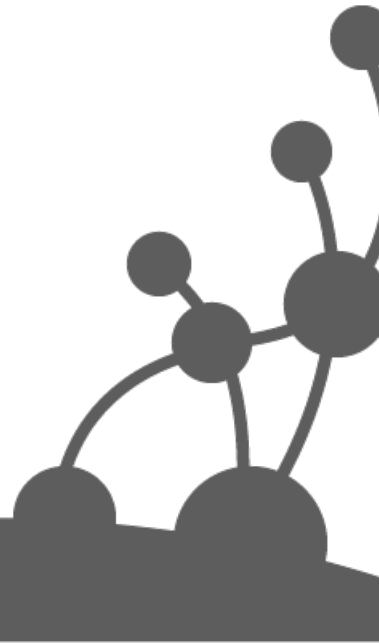
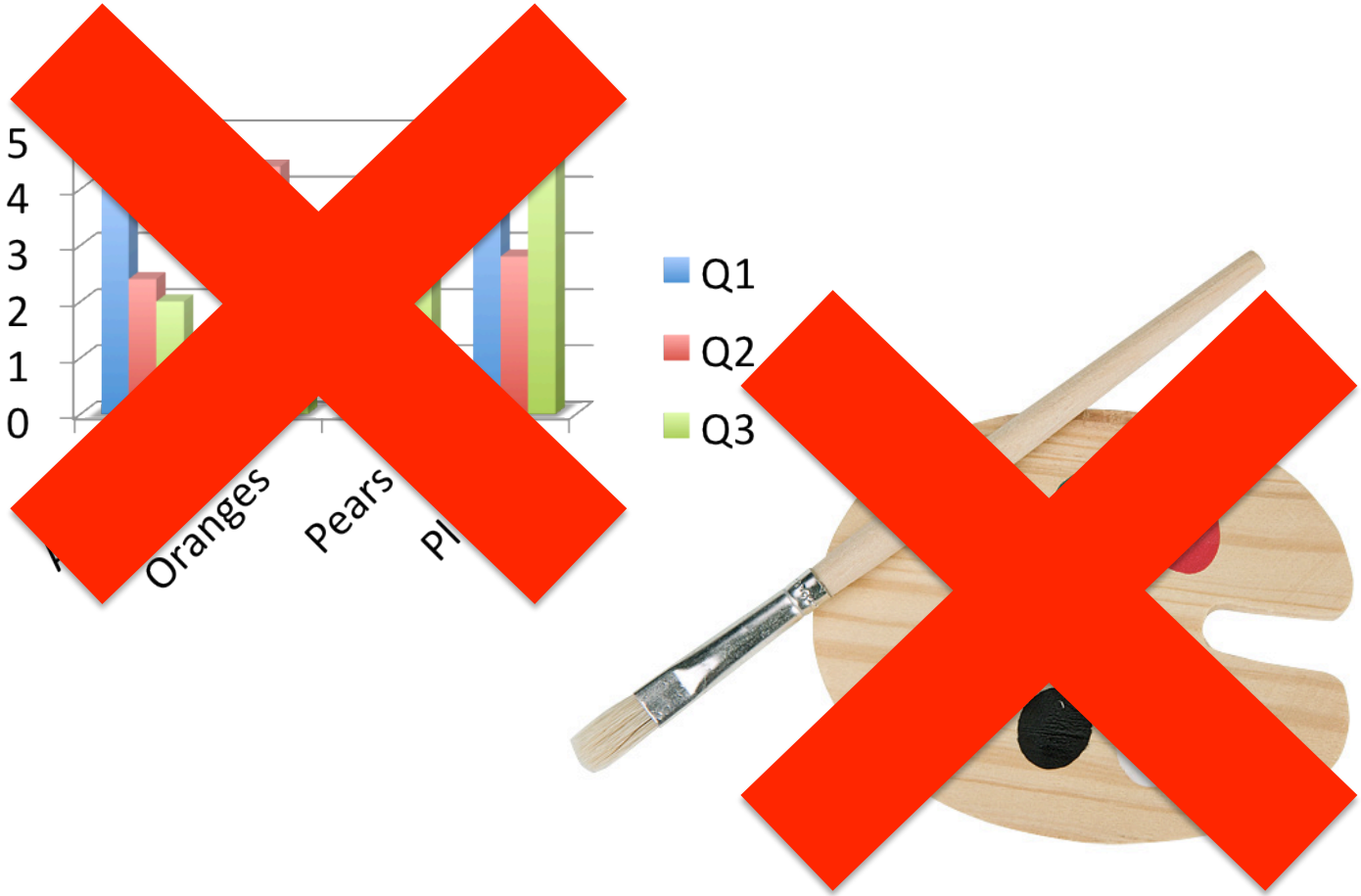
An Introduction to Doctor Who (and Neo4j)

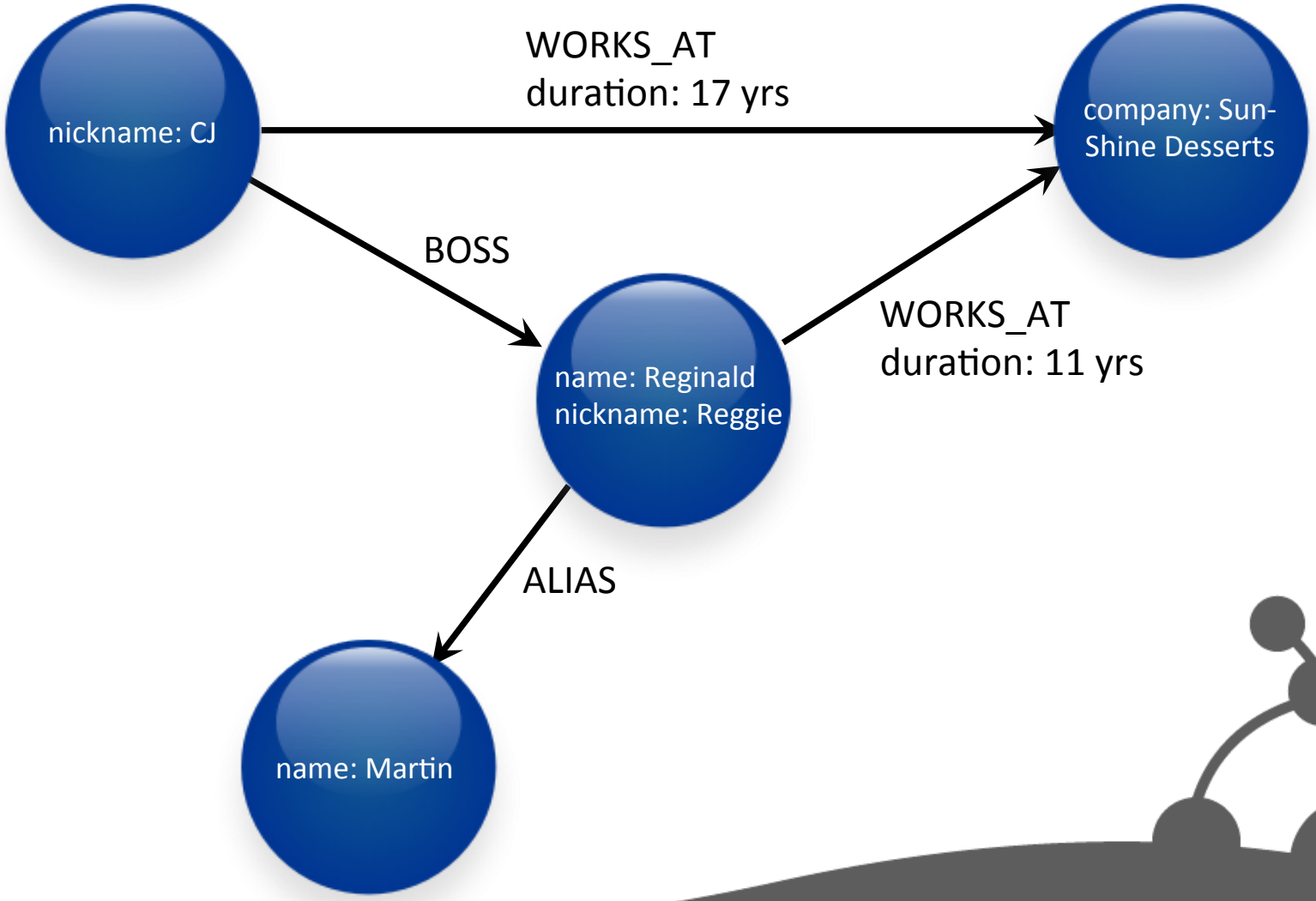
@iansrobinson
ian.robinson@neotechnology.com
#neo4j



Neo4j is a Graph Database

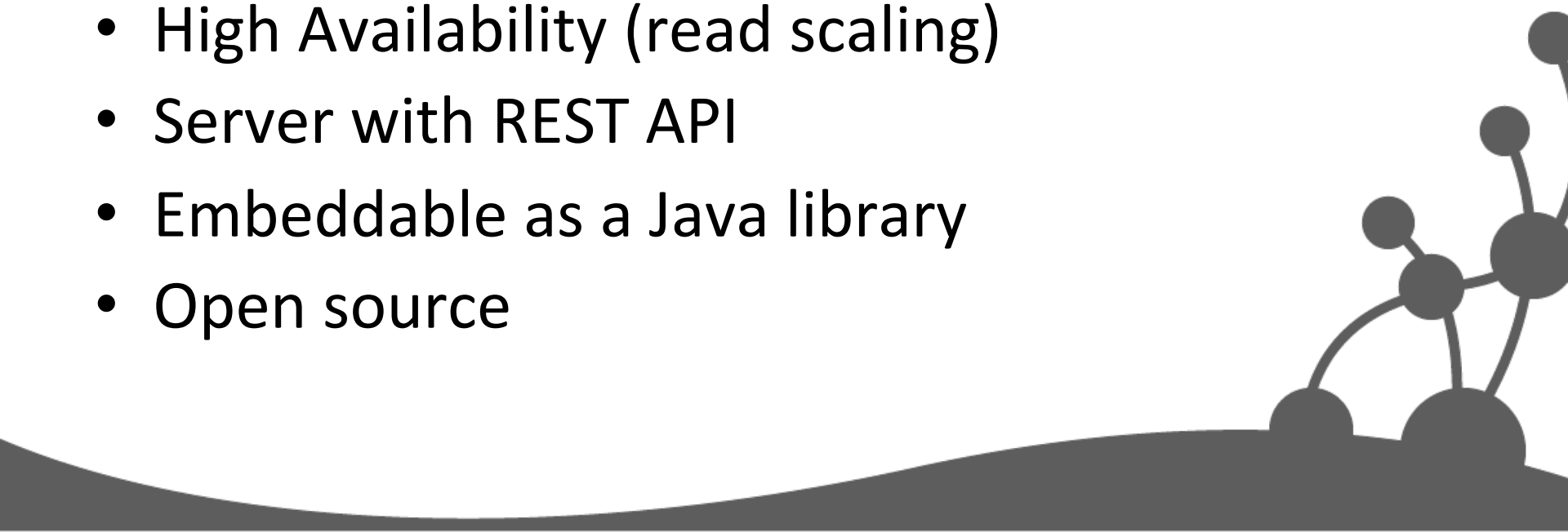


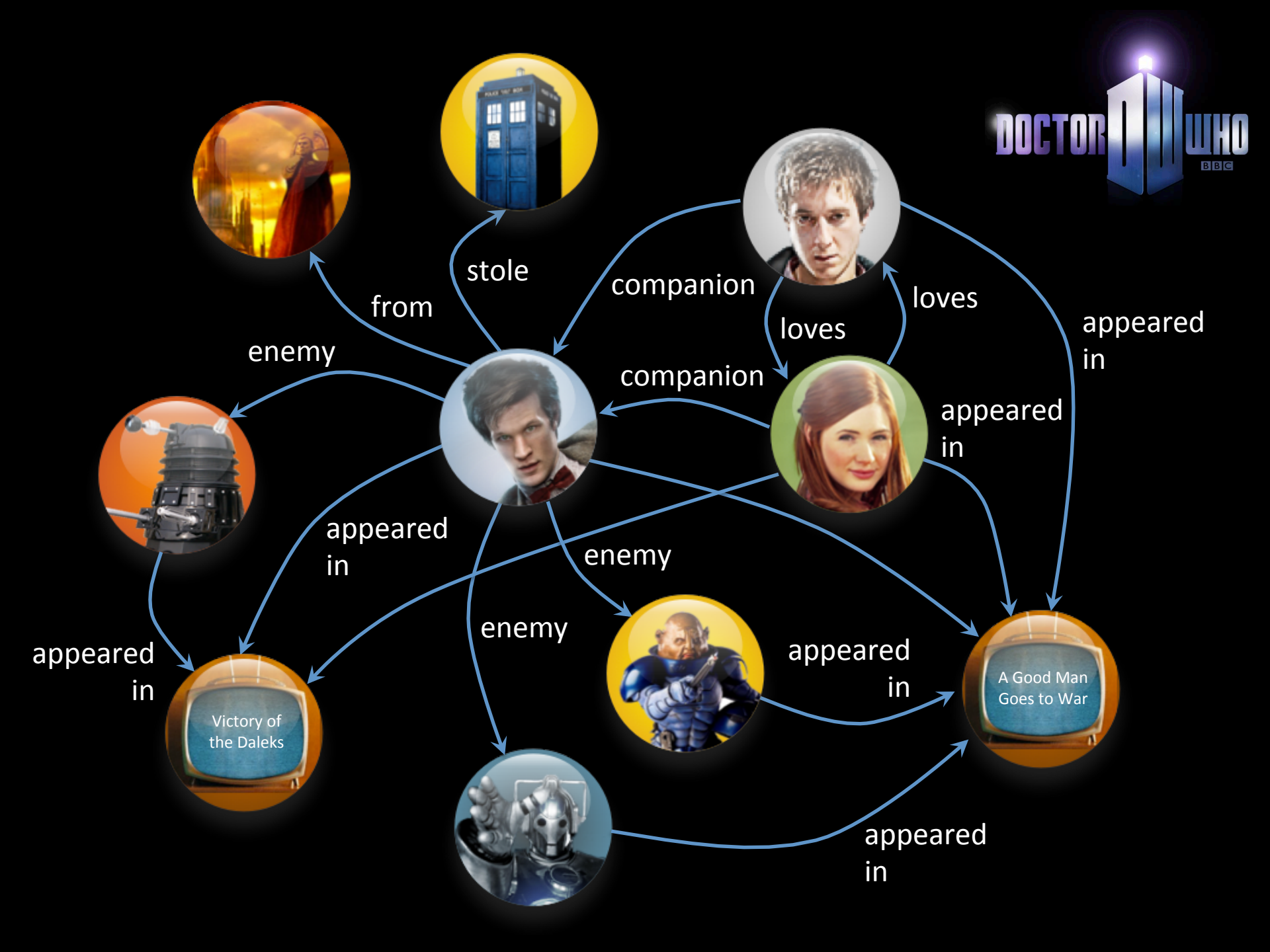
Property Graph



Features

- Semi-structured data
- Fully ACID
- 32 billion nodes, 32 billion relationships, 64 billion properties
- High Availability (read scaling)
- Server with REST API
- Embeddable as a Java library
- Open source




DOCTOR WHO
BBC
stole
from
companion
loves
loves
appeared in
enemy
companion
appeared in
appeared in
enemy
enemy
appeared in
appeared in
Victory of the Daleks
A Good Man Goes to War
appeared in

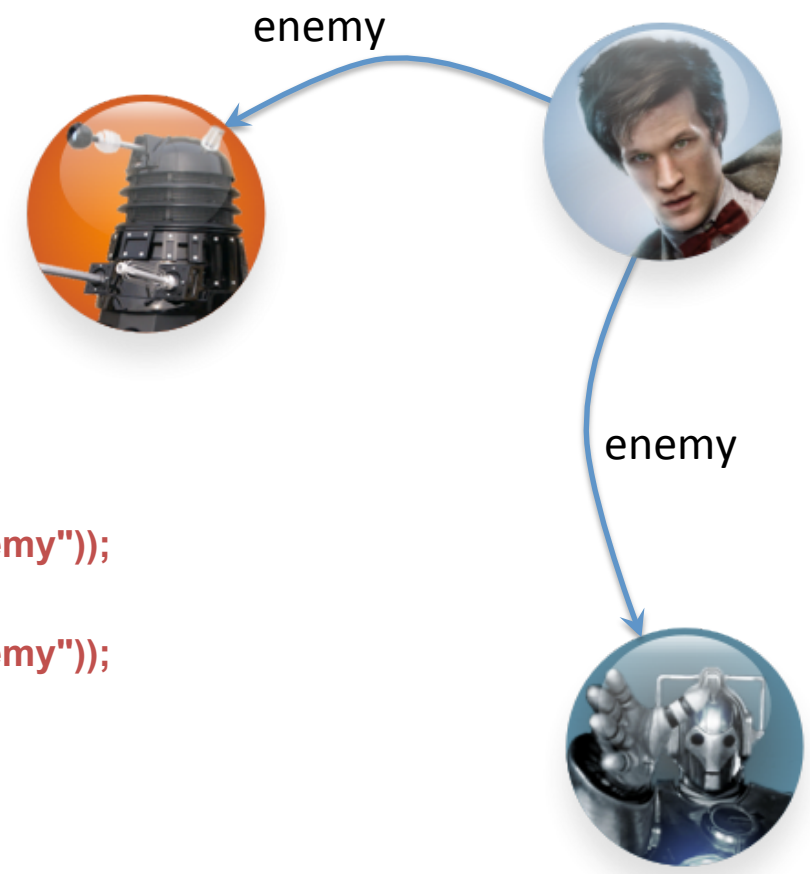
```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");
```

```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");
```

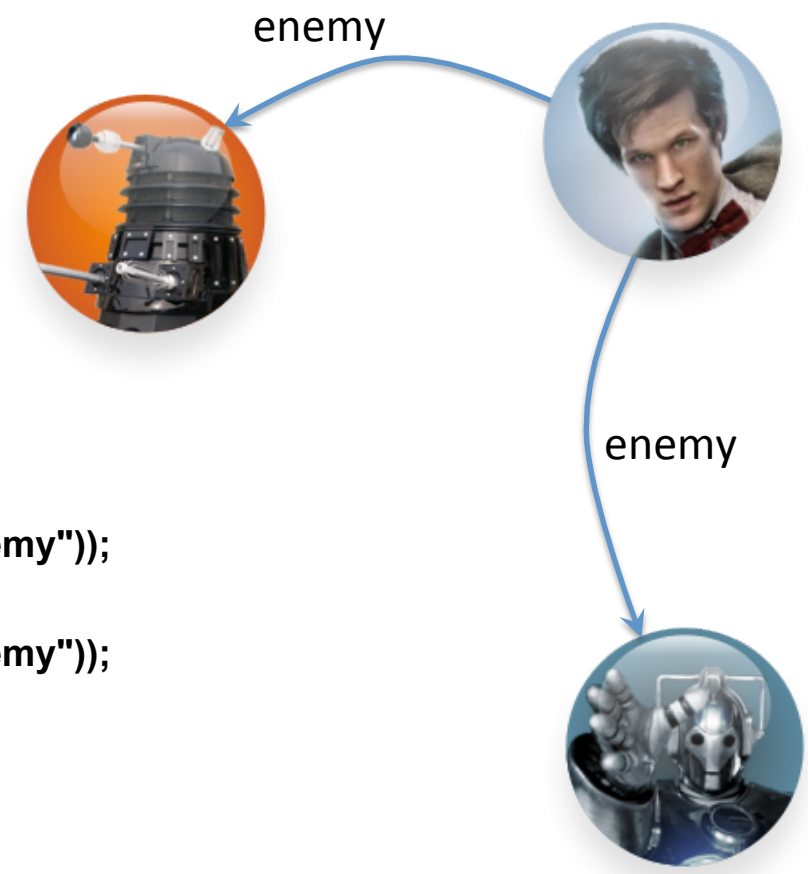
```
Node theDoctor = db.createNode();
theDoctor.setProperty("name", "The Doctor");

Node daleks = db.createNode();
daleks.setProperty("name", "Daleks");

Node cybermen = db.createNode();
cybermen.setProperty("name", "Cybermen");
```

```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");
```

```
Node theDoctor = db.createNode();
theDoctor.setProperty("name", "The Doctor");

Node daleks = db.createNode();
daleks.setProperty("name", "Daleks");

Node cybermen = db.createNode();
cybermen.setProperty("name", "Cybermen");

theDoctor.createRelationshipTo(daleks,
        DynamicRelationshipType.withName("enemy"));
theDoctor.createRelationshipTo(cybermen,
        DynamicRelationshipType.withName("enemy"));
```

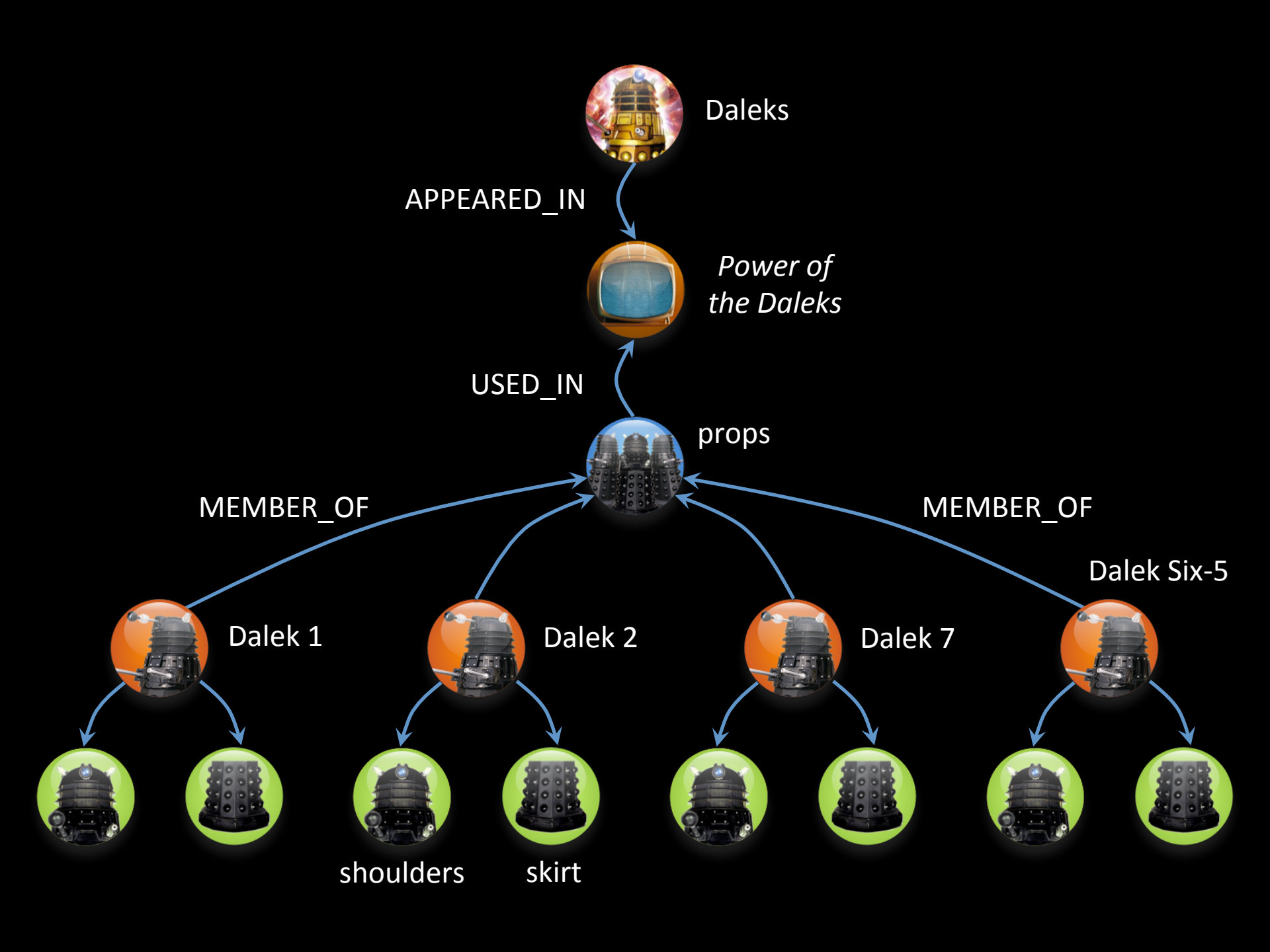


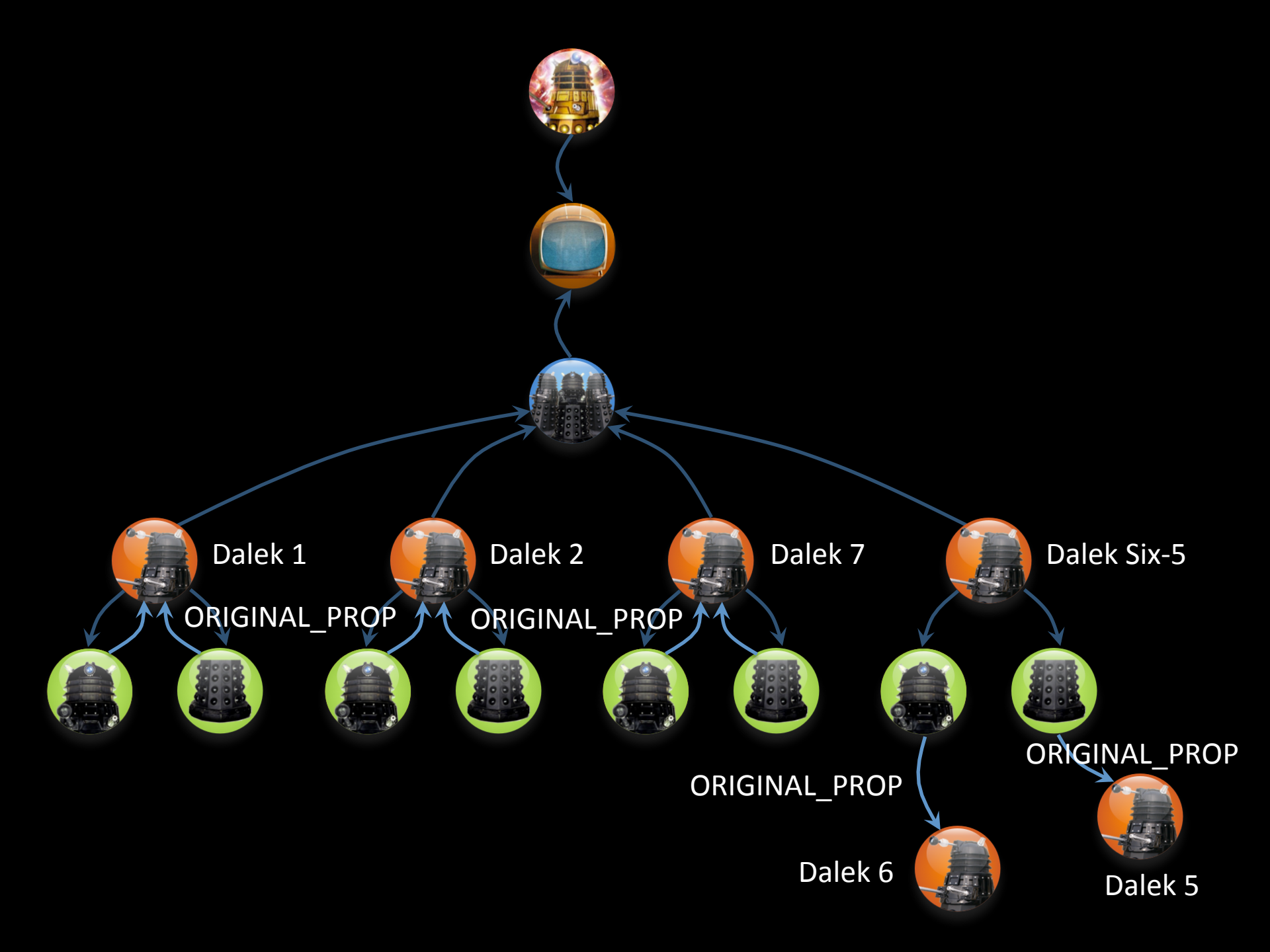
```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");

Transaction tx = db.beginTx();
try {
        Node theDoctor = db.createNode();
        theDoctor.setProperty("name", "The Doctor");

        Node daleks = db.createNode();
        daleks.setProperty("name", "Daleks");

        Node cybermen = db.createNode();
        cybermen.setProperty("name", "Cybermen");

        theDoctor.createRelationshipTo(daleks,
                DynamicRelationshipType.withName("enemy"));
        theDoctor.createRelationshipTo(cybermen,
                DynamicRelationshipType.withName("enemy"));

        tx.success();
} finally {
        tx.finish();
}
```

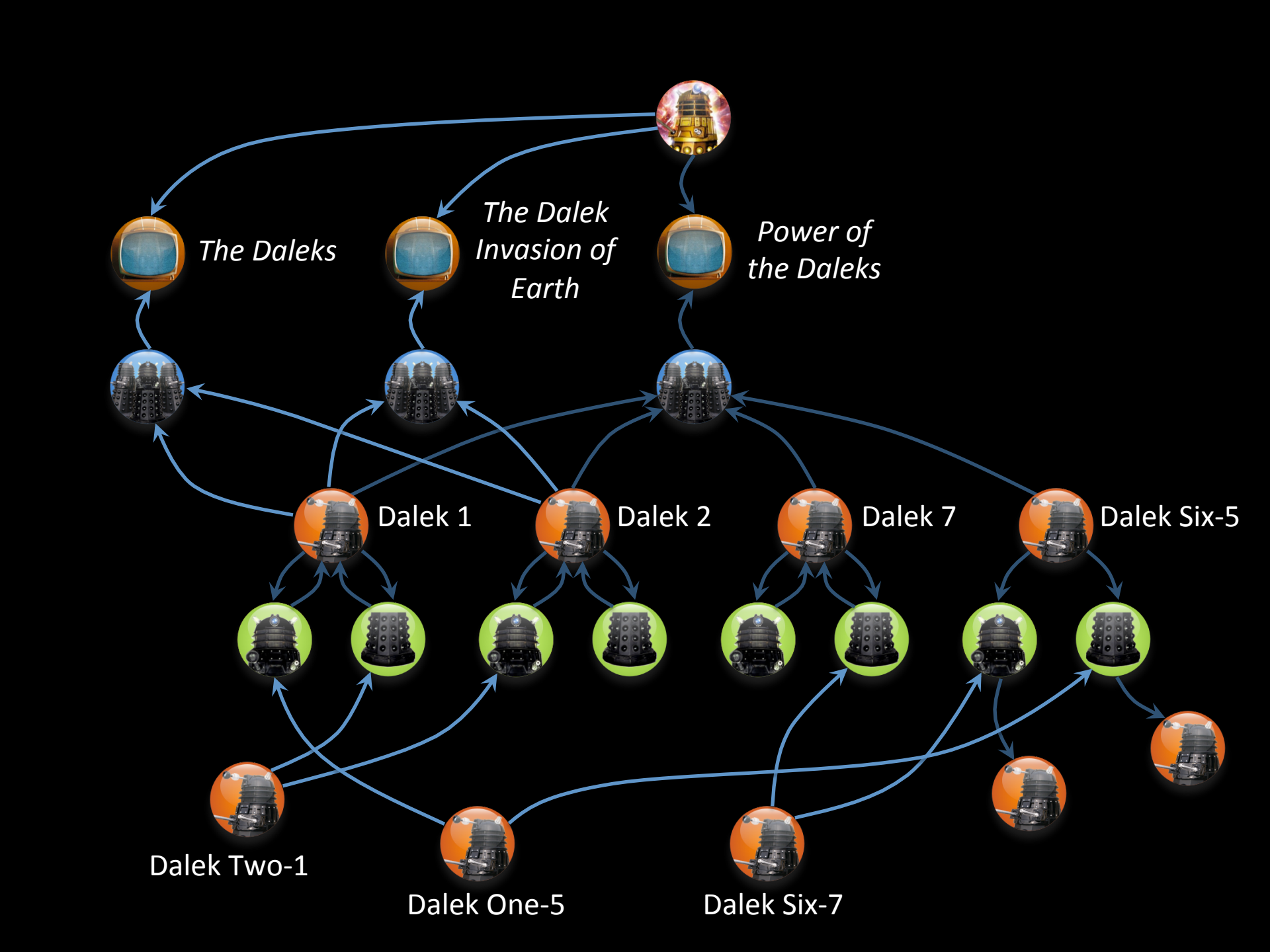


Dalek Props

http://www.dalek6388.co.uk/


Daleks
APPEARED_IN
Power of the Daleks
USED_IN
props
MEMBER_OF
MEMBER_OF
Dalek Six-5
Dalek 1
Dalek 2
Dalek 7
shoulders
skirt

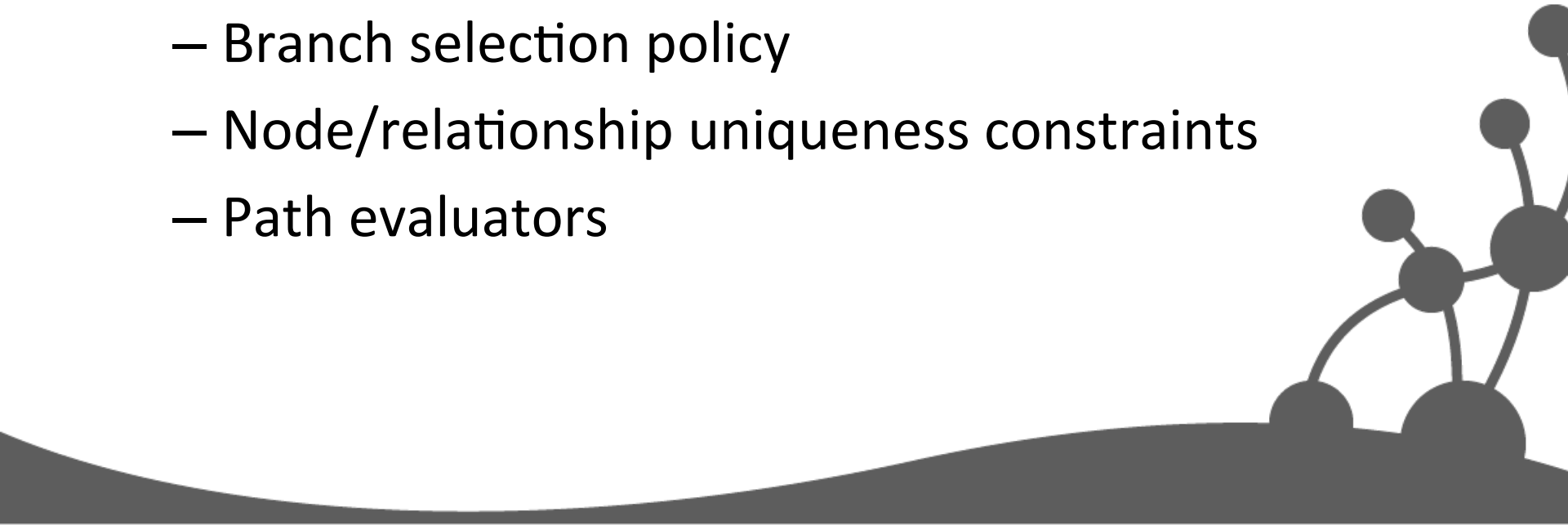

Dalek 1
Dalek 2
Dalek 7
Dalek Six-5
ORIGINAL_PROP
ORIGINAL_PROP
ORIGINAL_PROP
ORIGINAL_PROP
Dalek 6
Dalek 5


The Daleks
The Dalek Invasion of Earth
Power of the Daleks
Dalek 1
Dalek 2
Dalek 7
Dalek Six-5
Dalek Two-1
Dalek One-5
Dalek Six-7

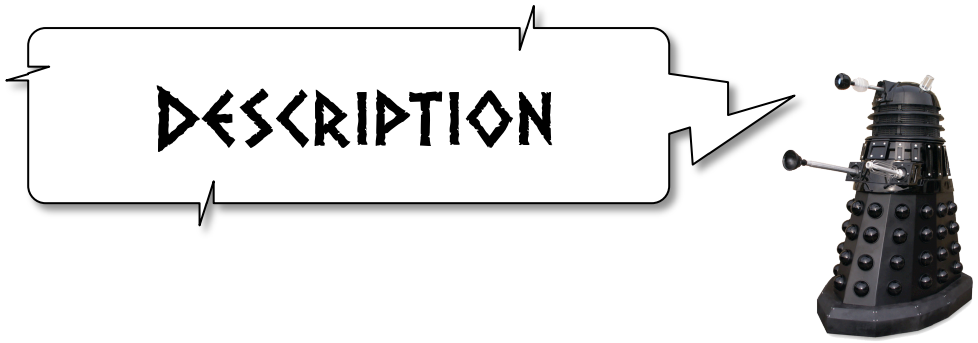
Supply Chain Traceability

Traversal Framework

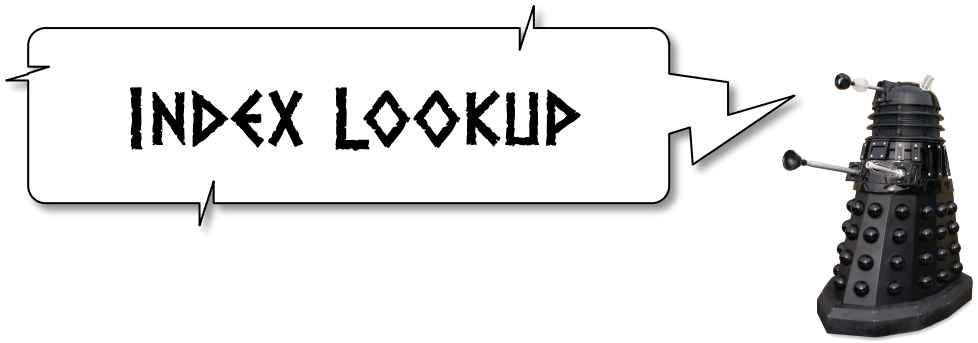
- Visits (and returns) nodes based on traversal description
- Powerful; can customize:
  - Relationships followed
  - Branch selection policy
  - Node/relationship uniqueness constraints
  - Path evaluators



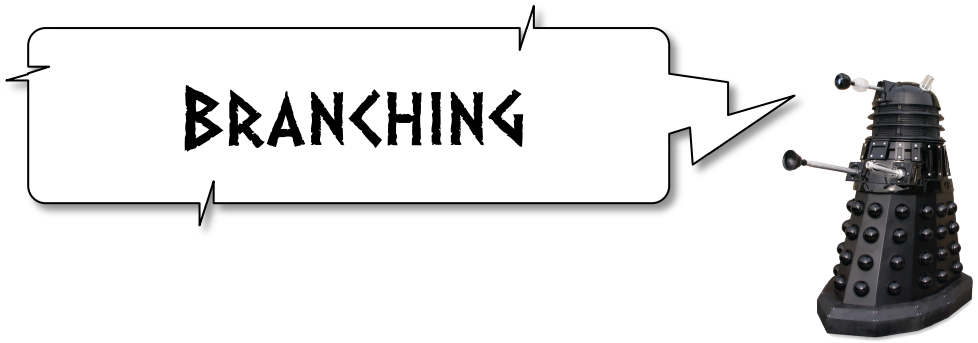
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



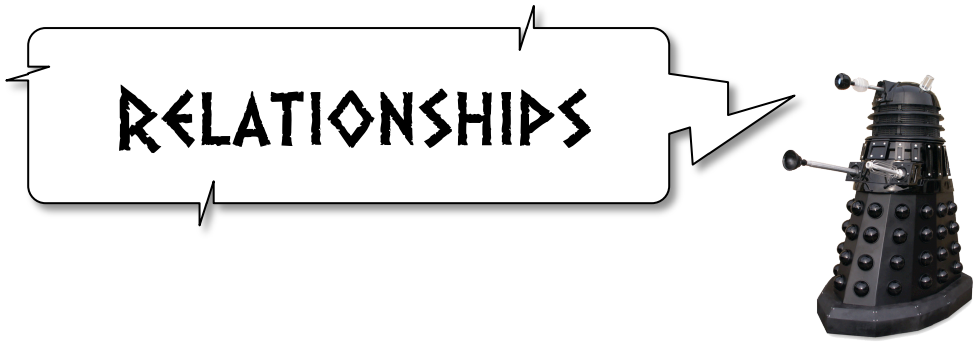
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



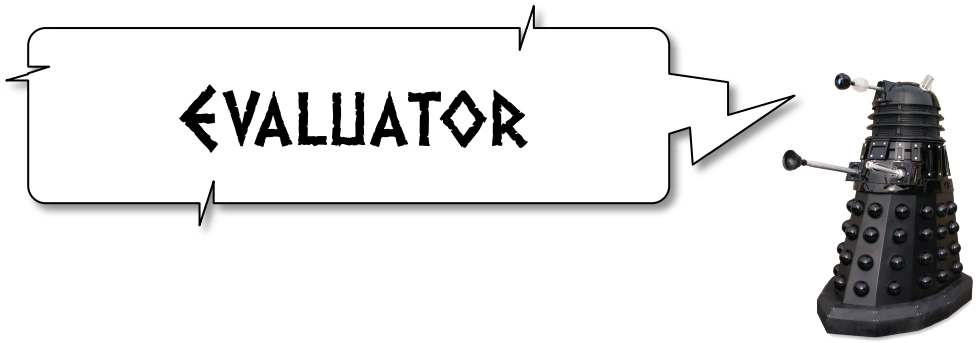
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



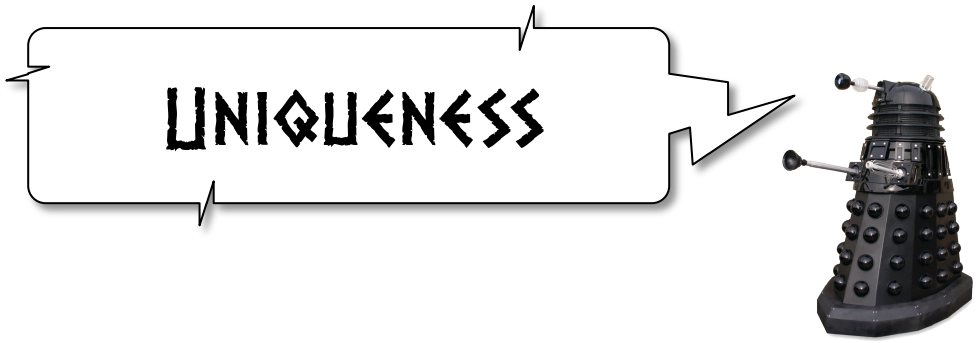
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



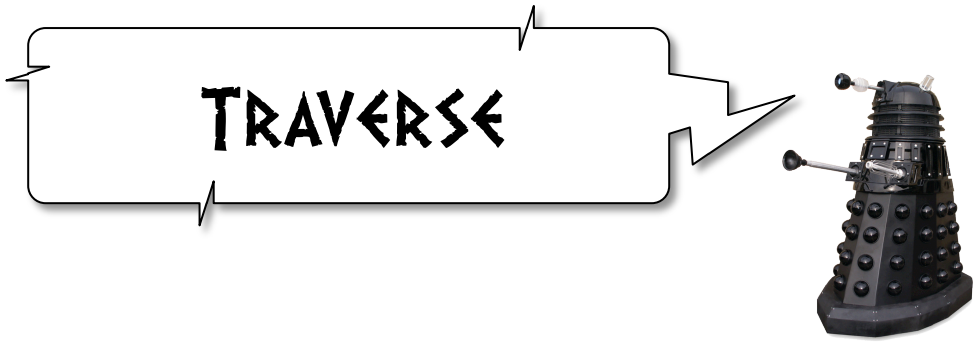
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



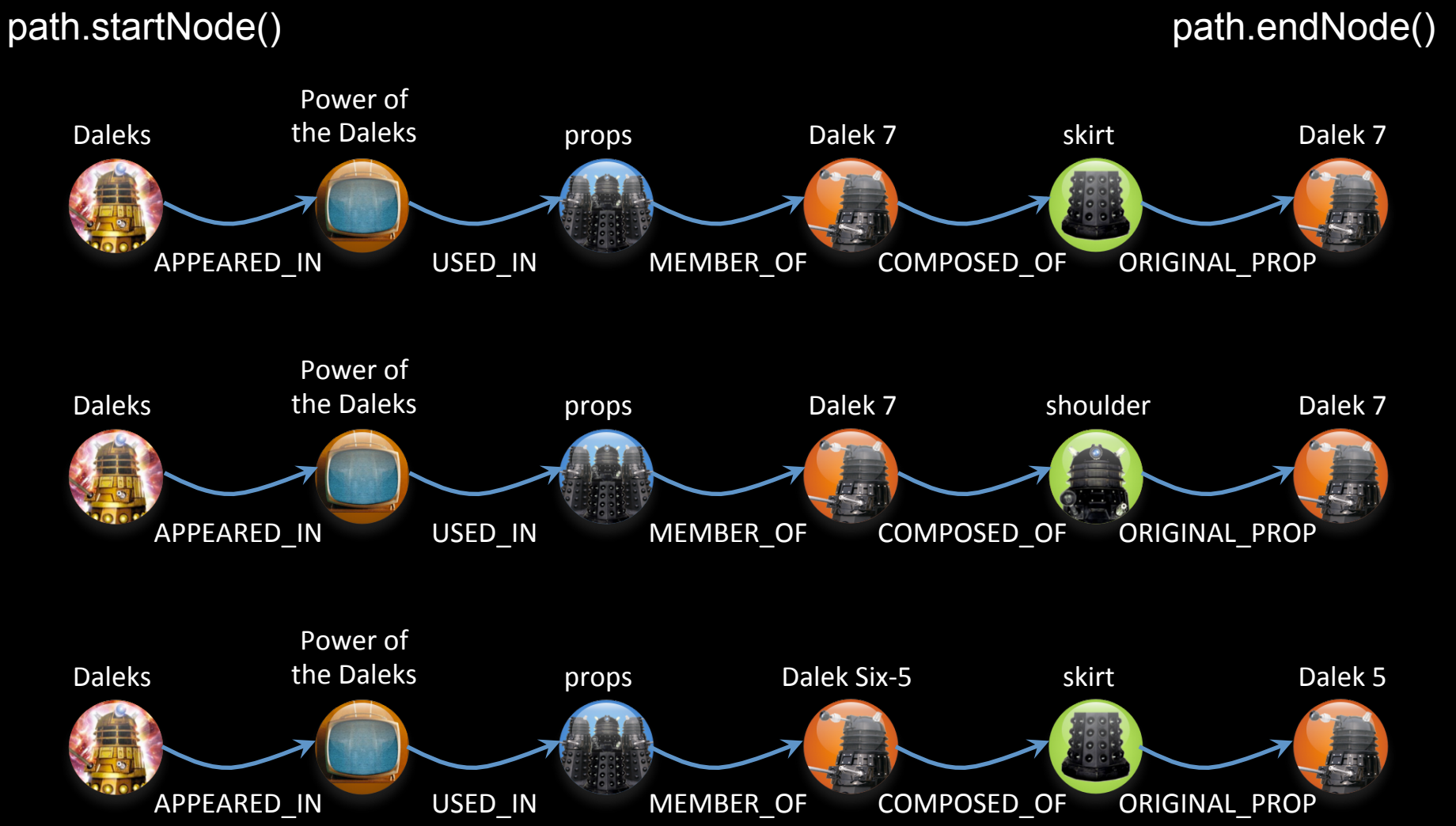
```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```



```
Node theDaleks = database.index().forNodes("species").get("name", "Dalek").getSingle();

Traverser traverser = Traversal.description()
    .depthFirst()
    .relationships(Rels.APPEARED_IN, Direction.OUTGOING)
    .relationships(Rels.USED_IN, Direction.INCOMING)
    .relationships(Rels.MEMBER_OF, Direction.INCOMING)
    .relationships(Rels.COMPOSED_OF, Direction.OUTGOING)
    .relationships(Rels.ORIGINAL_PROP, Direction.OUTGOING)
    .evaluator(new Evaluator() {
        @Override
        public Evaluation evaluate(Path path) {
            if (path.lastRelationship() != null && path.lastRelationship().isType(Rels.ORIGINAL_PROP)){
                return Evaluation.INCLUDE_AND_PRUNE;
            }
            return Evaluation.EXCLUDE_AND_CONTINUE;
        }
    })
    .uniqueness(Uniqueness.NONE)
    .traverse(theDaleks);
```

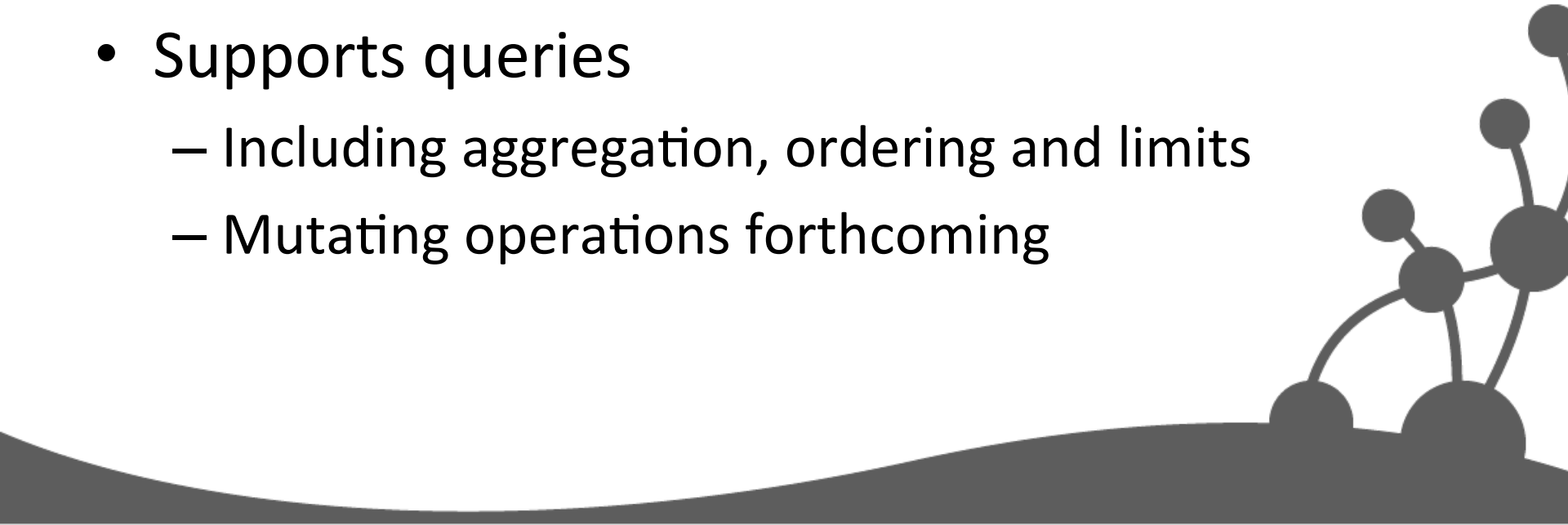


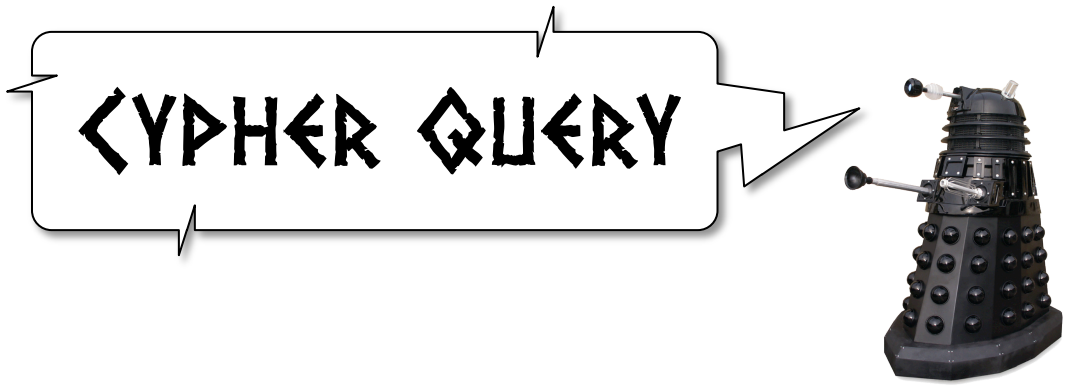
Iterable<Path>



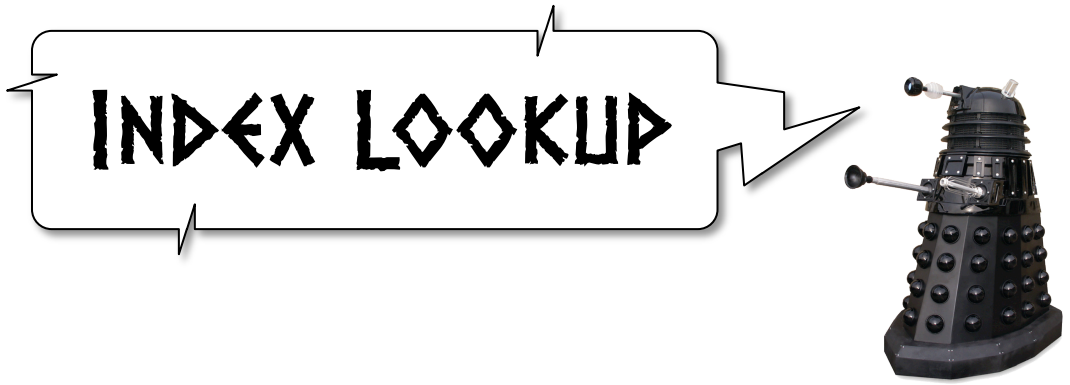
Cypher

- Declarative graph *pattern matching* language
  - "SQL for graphs"
- New in 1.4
  - Experimental; syntax changing rapidly
- Supports queries
  - Including aggregation, ordering and limits
  - Mutating operations forthcoming






```
start daleks=(species,name,'Dalek')
match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-
        ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
        (part)-[:ORIGINAL_PROP]->(originalprop)
return originalprop.name, part.type, count(episode.title)
order by count(episode.title) desc
limit 1
```



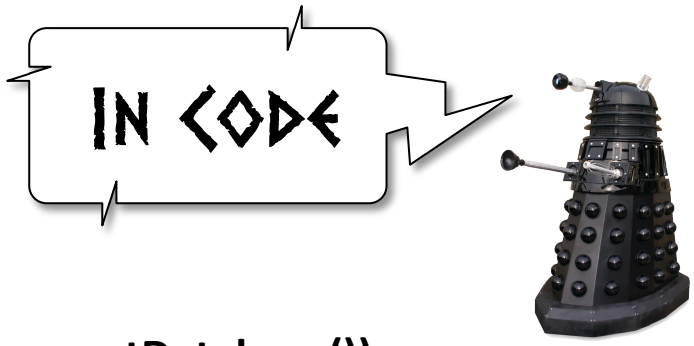

```
start daleks=(species,name,'Dalek')
match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-
      ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
      (part)-[:ORIGINAL_PROP]->(originalprop)
return originalprop.name, part.type, count(episode.title)
order by count(episode.title) desc
limit 1
```

MATCH NODES + RELATIONSHIPS

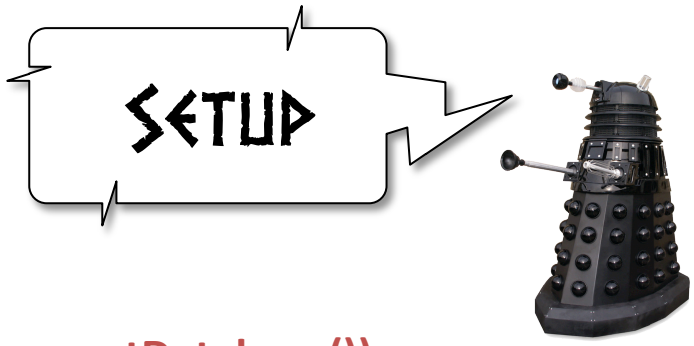
```
start daleks=(species,name,'Dalek')
match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-
      ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
      (part)-[:ORIGINAL_PROP]->(originalprop)
return originalprop.name, part.type, count(episode.title)
order by count(episode.title) desc
limit 1
```

Return Values

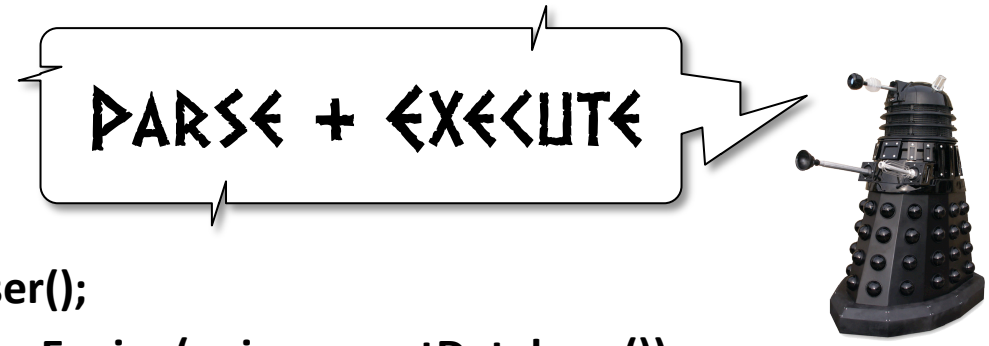
```
start daleks=(species,name,'Dalek')
match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-
      ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
      (part)-[:ORIGINAL_PROP]->(originalprop)
return originalprop.name, part.type, count(episode.title)
order by count(episode.title) desc
limit 1
```



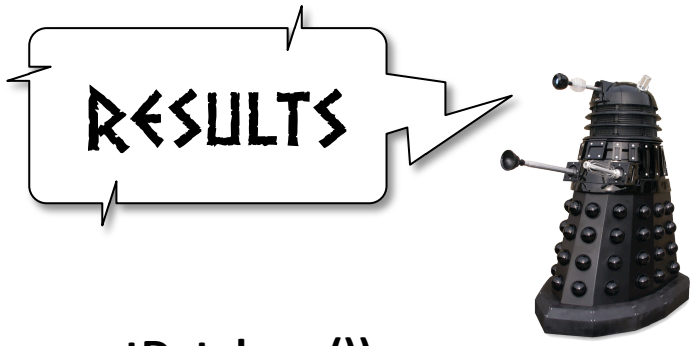

```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(universe.getDatabase());

String cql = "start daleks=(species,name,'Dalek')"
   + " match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()"
   + "-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)"
   + " return originalprop.name, part.type, count(episode.title)"
   + " order by count(episode.title) desc limit 1";

Query query = parser.parse(cql);
ExecutionResult result = engine.execute(query);

String originalProp = result.javaColumnAs("originalprop.name").next().toString();
String part = result.javaColumnAs("part.type").next().toString();
int episodeCount = (Integer) result.javaColumnAs("count(episode.title)").next();
```




```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(universe.getDatabase());

String cql = "start daleks=(species,name,'Dalek')"
   + " match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()"
   + "-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)"
   + " return originalprop.name, part.type, count(episode.title)"
   + " order by count(episode.title) desc limit 1";

Query query = parser.parse(cql);
ExecutionResult result = engine.execute(query);

String originalProp = result.javaColumnAs("originalprop.name").next().toString();
String part = result.javaColumnAs("part.type").next().toString();
int episodeCount = (Integer) result.javaColumnAs("count(episode.title)").next();
```




```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(universe.getDatabase());

String cql = "start daleks=(species,name,'Dalek')"
   + " match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()"
   + "-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)"
   + " return originalprop.name, part.type, count(episode.title)"
   + " order by count(episode.title) desc limit 1";

Query query = parser.parse(cql);
ExecutionResult result = engine.execute(query);

String originalProp = result.javaColumnAs("originalprop.name").next().toString();
String part = result.javaColumnAs("part.type").next().toString();
int episodeCount = (Integer) result.javaColumnAs("count(episode.title)").next();
```




```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(universe.getDatabase());

String cql = "start daleks=(species,name,'Dalek')"
   + " match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()"
   + "-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)"
   + " return originalprop.name, part.type, count(episode.title)"
   + " order by count(episode.title) desc limit 1";

Query query = parser.parse(cql);
ExecutionResult result = engine.execute(query);

String originalProp = result.javaColumnAs("originalprop.name").next().toString();
String part = result.javaColumnAs("part.type").next().toString();
int episodeCount = (Integer) result.javaColumnAs("count(episode.title)").next();
```

In Webadmin

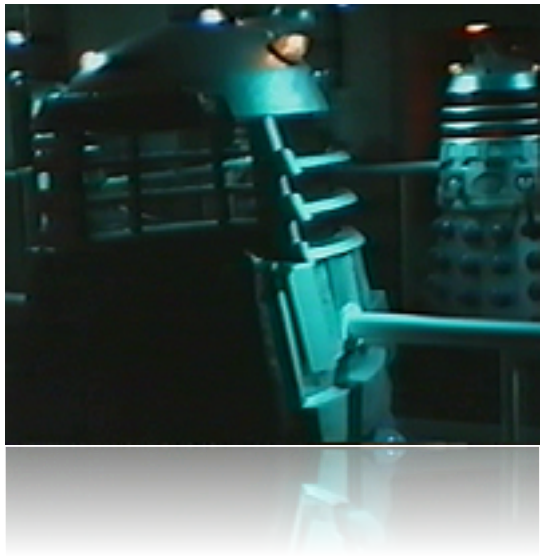
Overview **Dashboard** | Explore and edit **Data browser** | Power tool **Console** | Details **Server info** | Documentation

Gremlin | Cypher | HTTP

```
cypher> START daleks=(Species,species,'Dalek')
cypher> MATCH (daleks)-[:APPEARED_IN]->(episode)
cypher> <-[:USED_IN]-(props)
cypher> <-[:MEMBER_OF]-(prop)
cypher> -[:COMPOSED_OF]->(part)
cypher> -[:ORIGINAL_PROP]->(originalprop)
cypher> RETURN originalprop.name, part.type, COUNT(episode.title)
cypher> ORDER BY COUNT(episode.title) DESC
cypher> LIMIT 1
cypher>
==> +-------------------------------------------------------+
==> | originalprop.name | part.type | count(episode.title) |
==> +-------------------------------------------------------+
==> | Dalek 1           | shoulder  | 15                   |
==> +-------------------------------------------------------+
==> 1 rows, 86 ms
cypher>
```

The Hardest Working Prop Part

Dalek One’s shoulders



http://www.dalek6388.co.uk/

More Info

- Download:
  - http://neo4j.org/download/
- Tutorial:
  - https://github.com/jimwebber/neo4j-tutorial

Questions?

@iansrobinson
ian.robinson@neotechnology.com
#neo4j