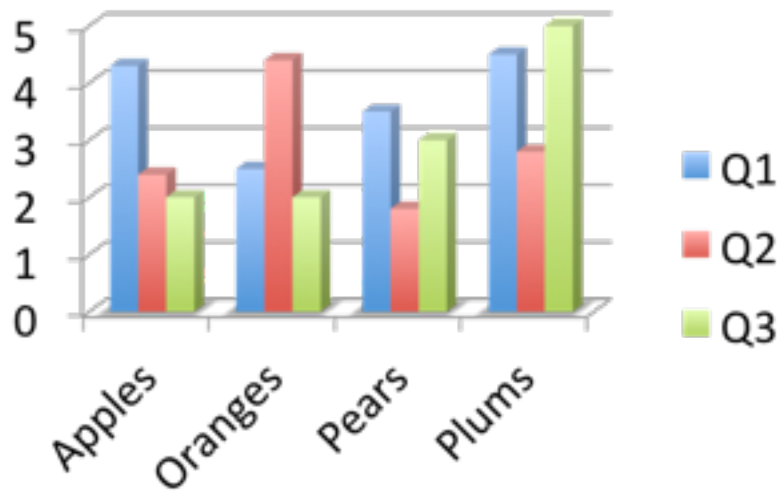
An Introduction to Neo4j

@iansrobinson
ian.robinson@neotechnology.com

Neo4j is a Graph Database

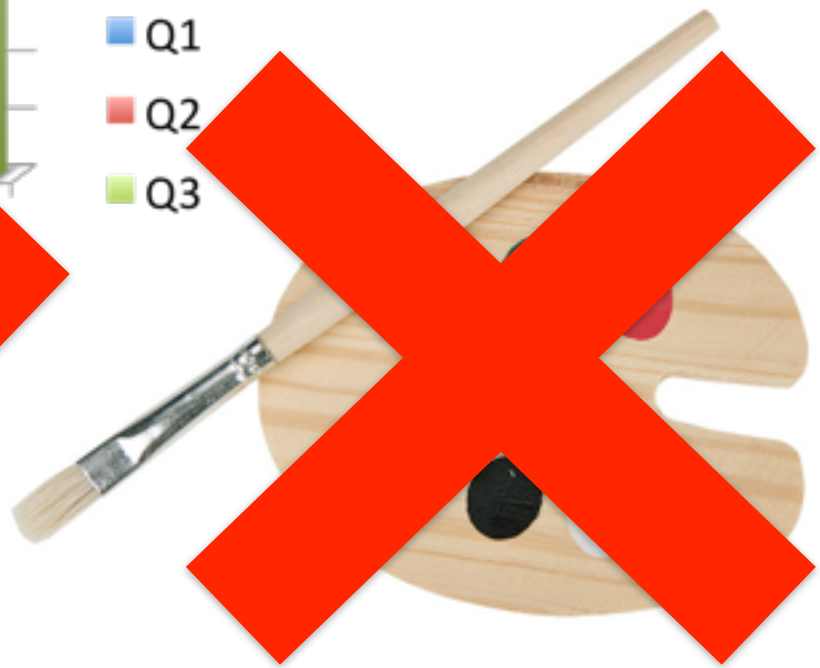
Neo4j is a Graph Database



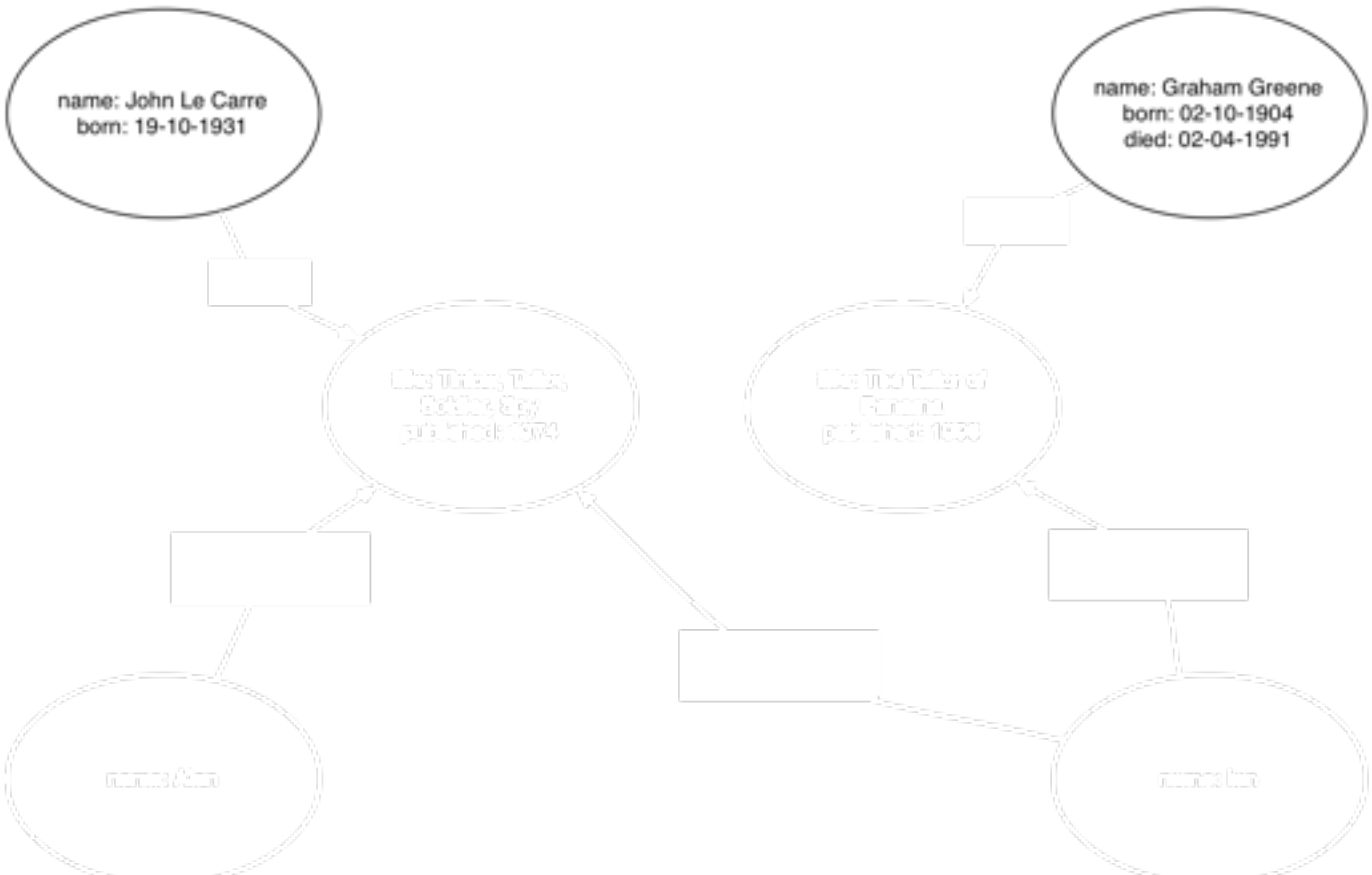
Neo4j is a Graph Database

Neo4j is a Graph Database

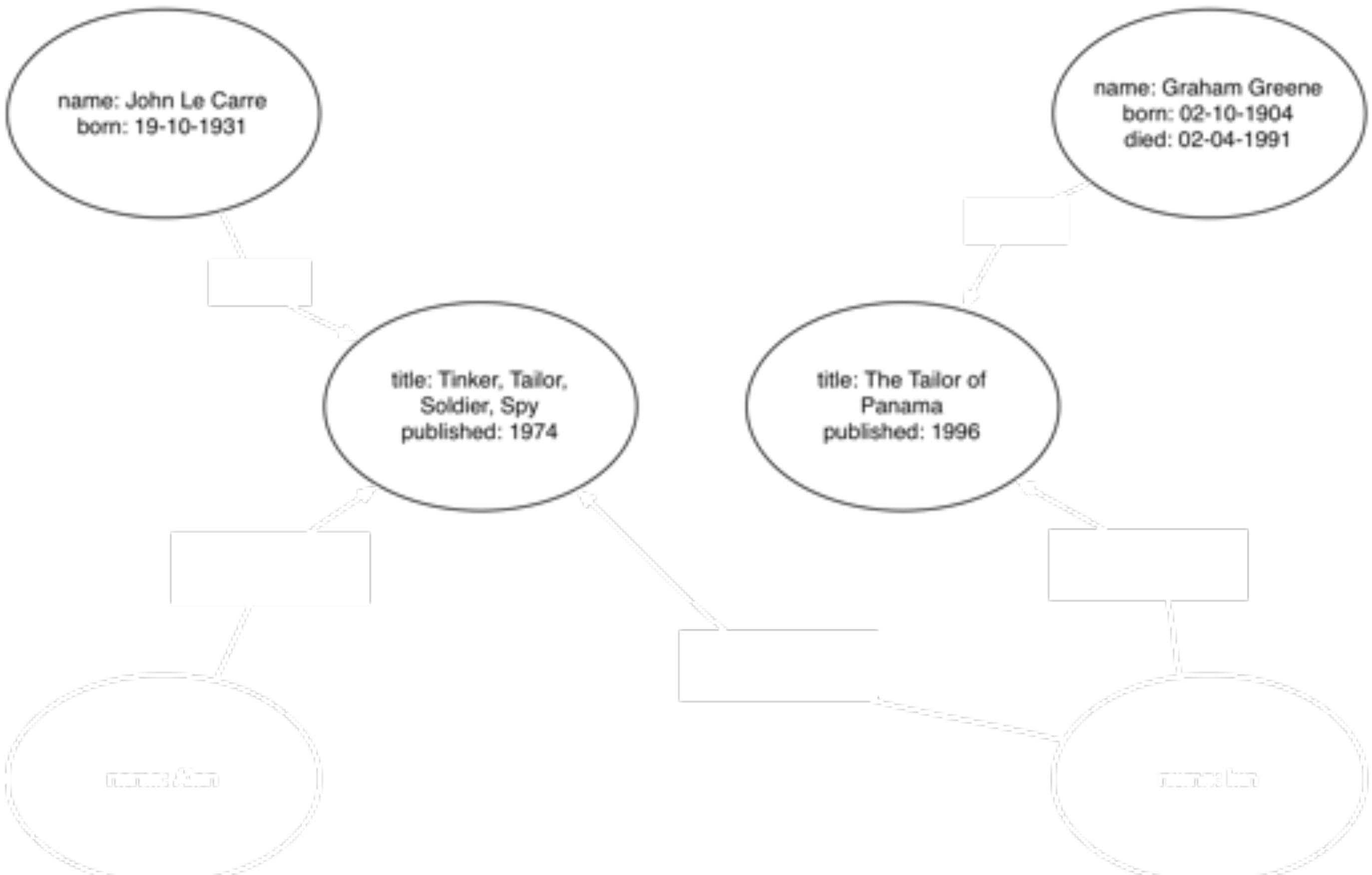
Neo4j is a Graph Database



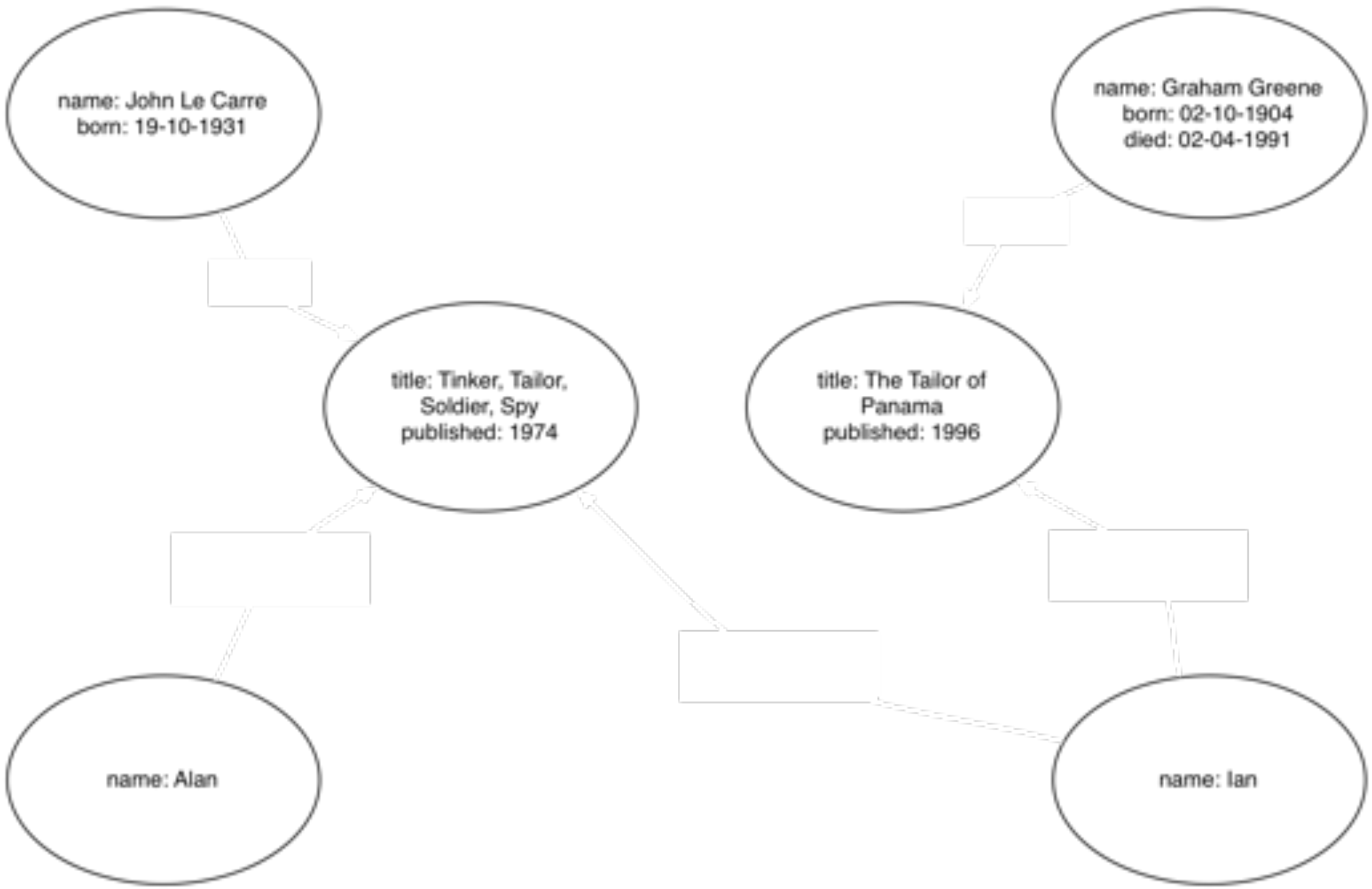
Nodes & Properties



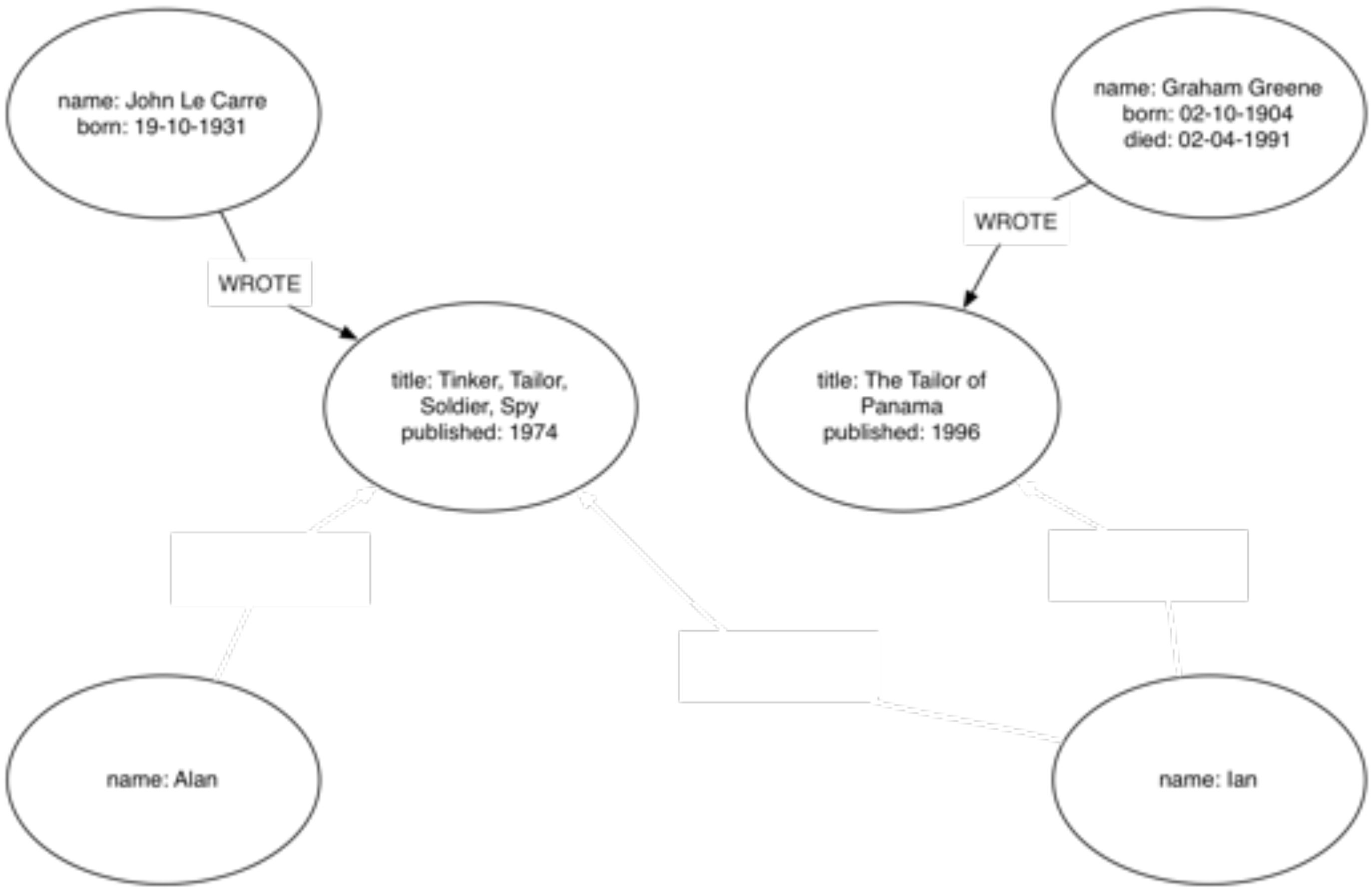
Nodes & Properties



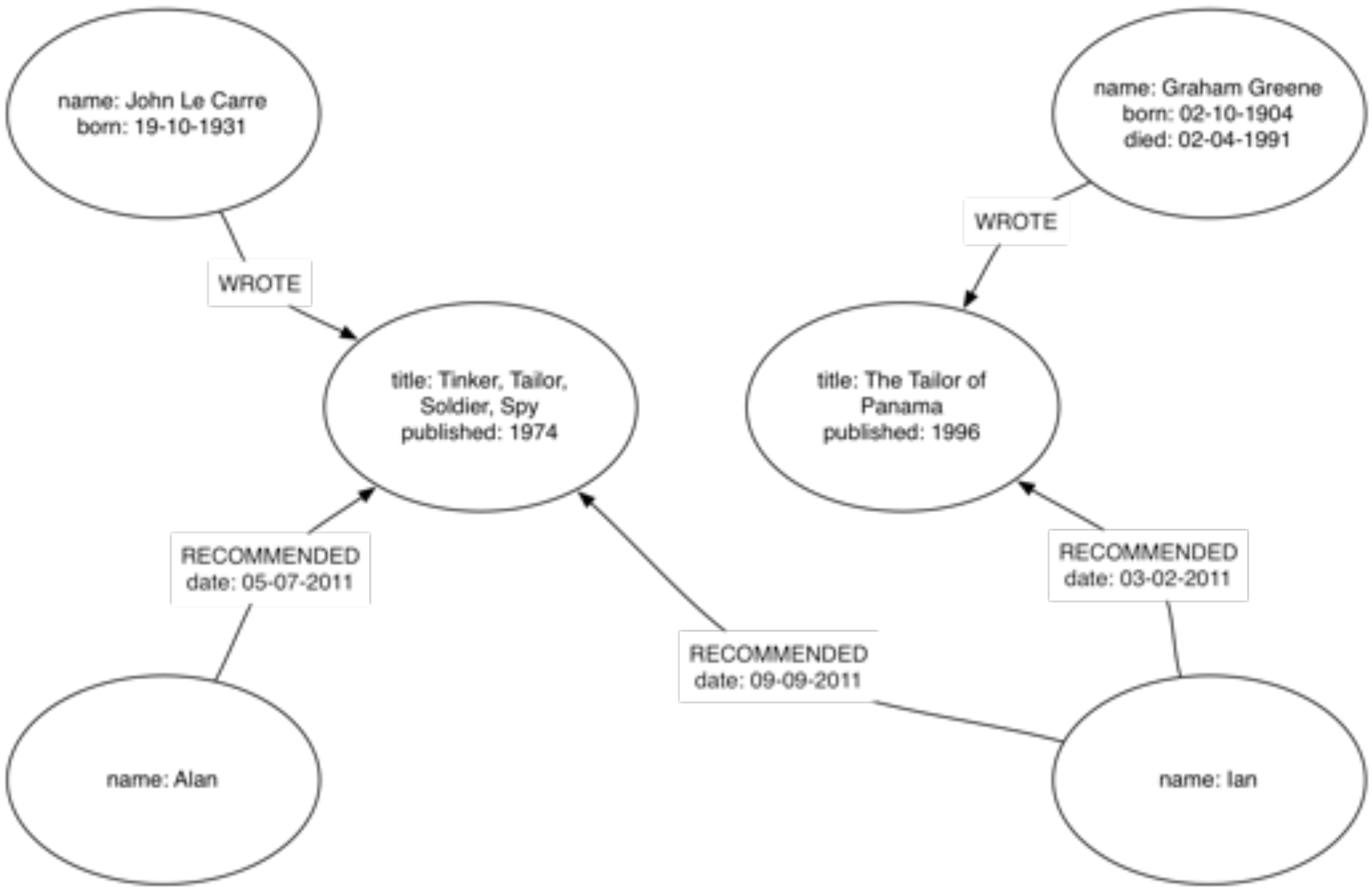
Nodes & Properties



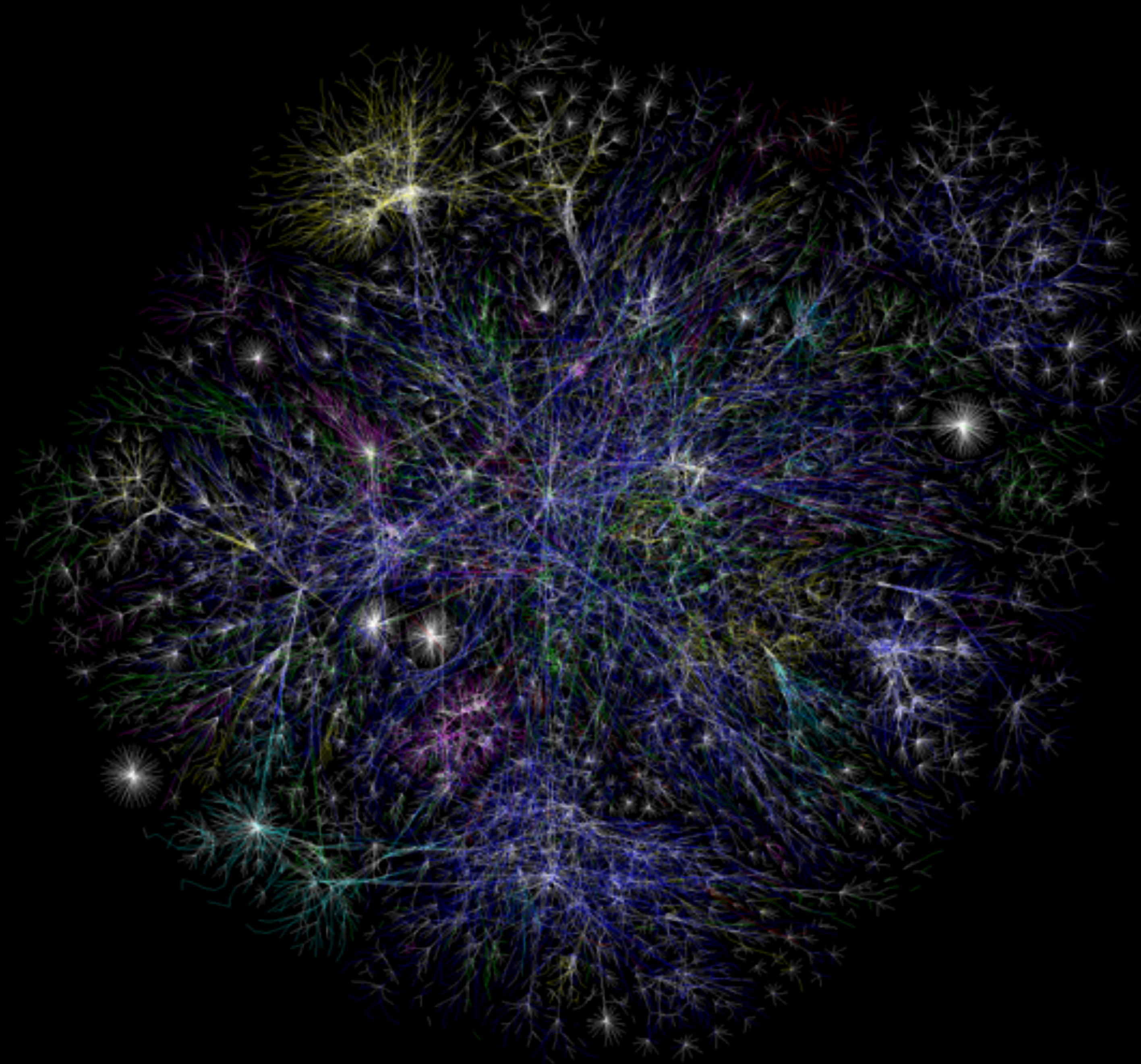
Relationships

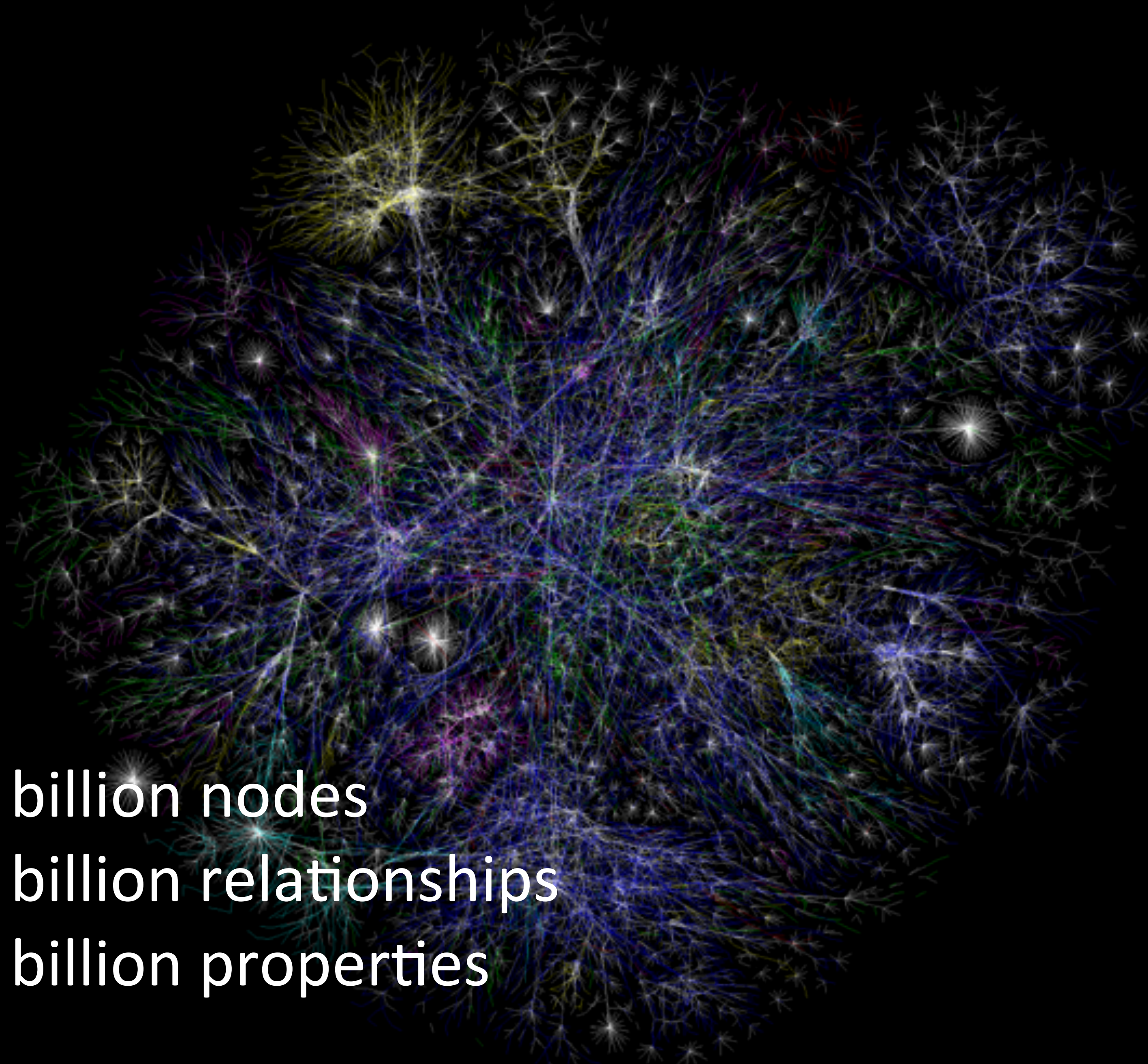


Relationships

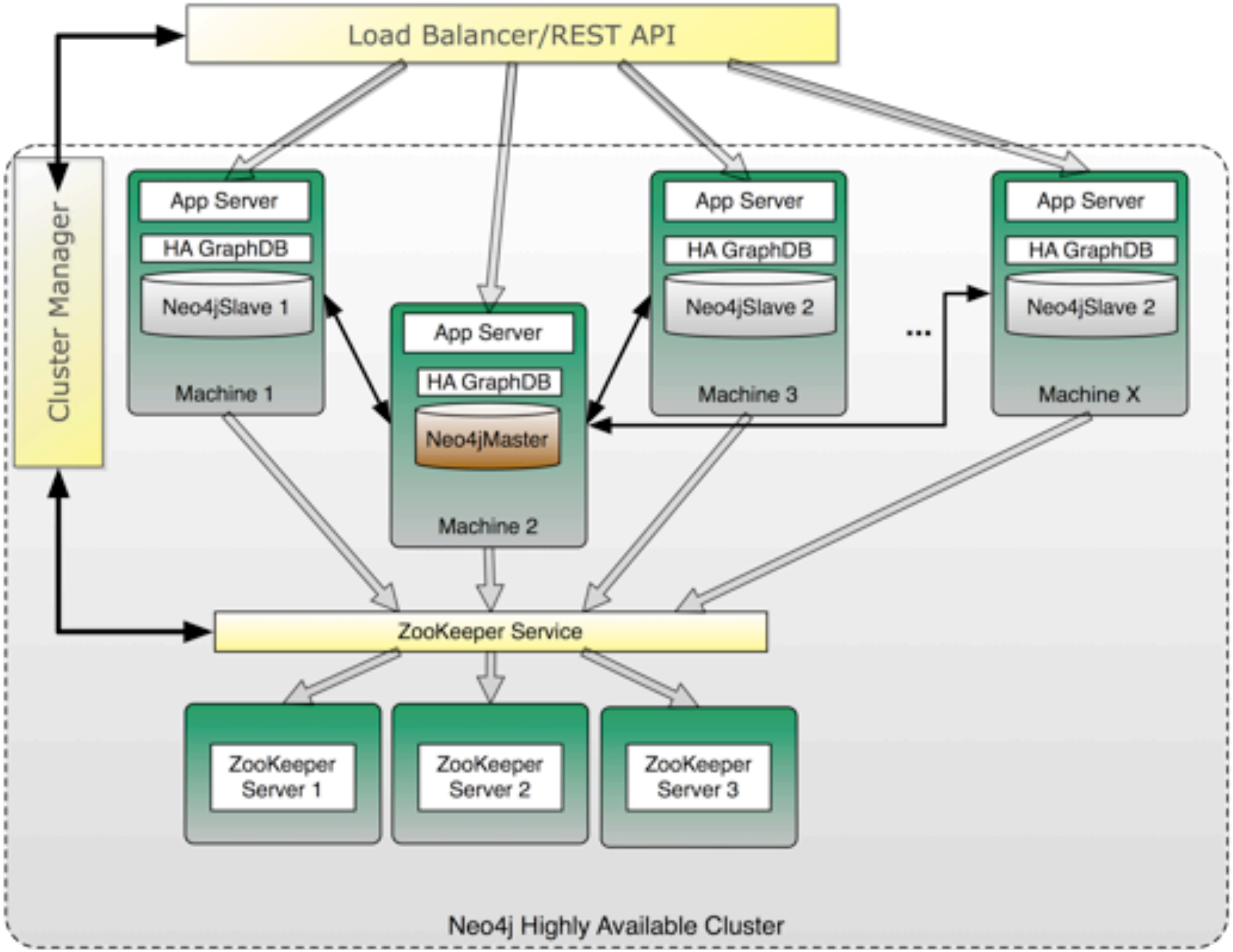


Neo4j



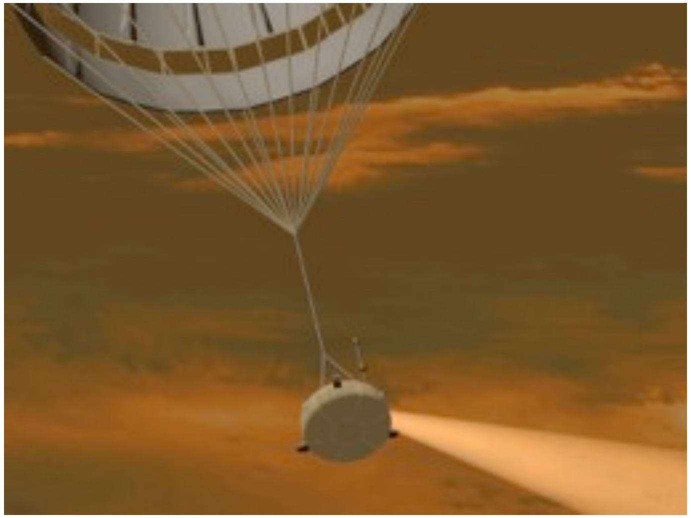


32 billion nodes
32 billion relationships
64 billion properties


Load Balancer/REST API
Cluster Manager
App Server
HA GraphDB
Neo4jSlave 1
Machine 1
App Server
HA GraphDB
Neo4jMaster
Machine 2
App Server
HA GraphDB
Neo4jSlave 2
Machine 3
...
App Server
HA GraphDB
Neo4jSlave 2
Machine X
ZooKeeper Service
ZooKeeper Server 1
ZooKeeper Server 2
ZooKeeper Server 3
Neo4j Highly Available Cluster

I ♥ ACID

How do I query the data?



http://opfm.jpl.nasa.gov/

http://news.xinhuanet.com

Why Neo4j?

Why Neo4j?

Schema-free

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

ACID transactions

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

ACID transactions

- Durable, consistent data

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

ACID transactions

- Durable, consistent data

Performance

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

ACID transactions

- Durable, consistent data

Performance

- Millions of 'joins' per second

Why Neo4j?

Schema-free

- Complex, densely-connected datasets

ACID transactions

- Durable, consistent data

Performance

- Millions of ‘joins’ per second
- Consistent query times as dataset grows

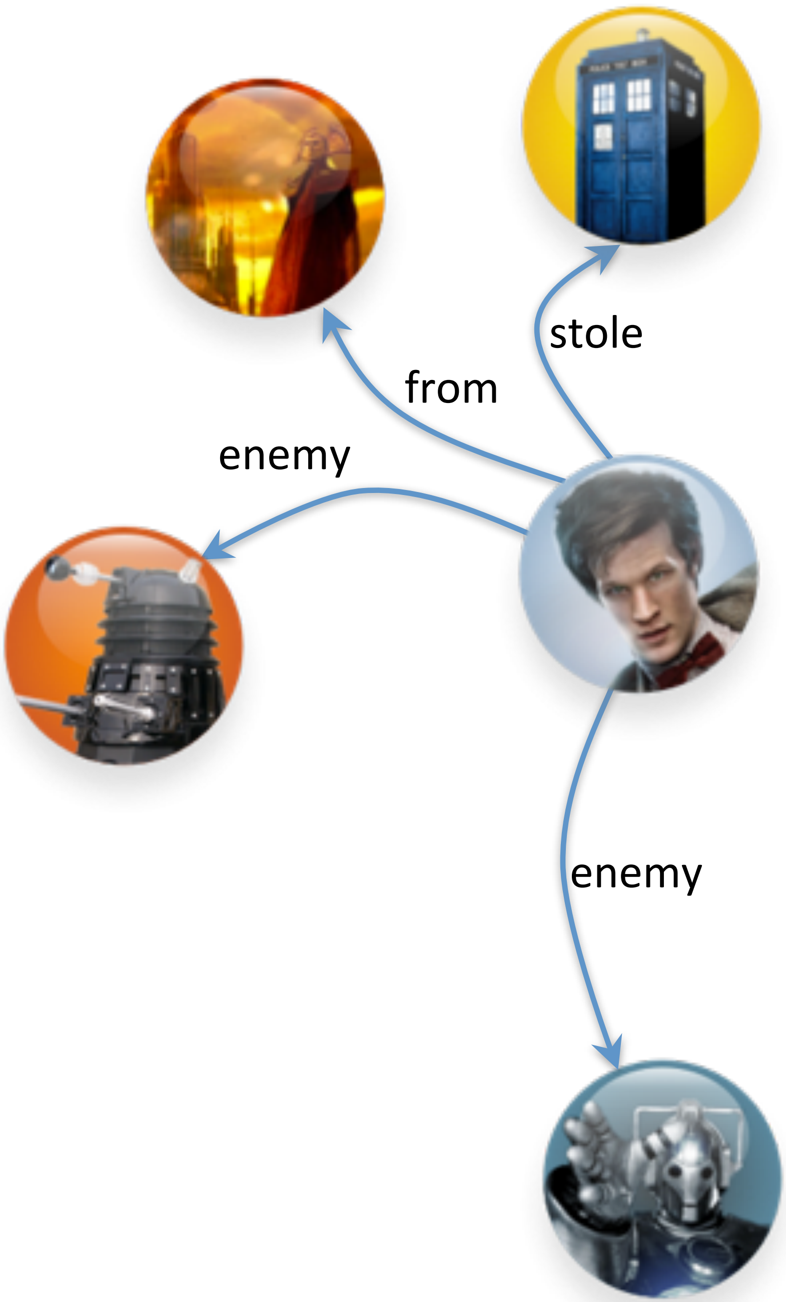
DOCTOR WHO

BBC


from

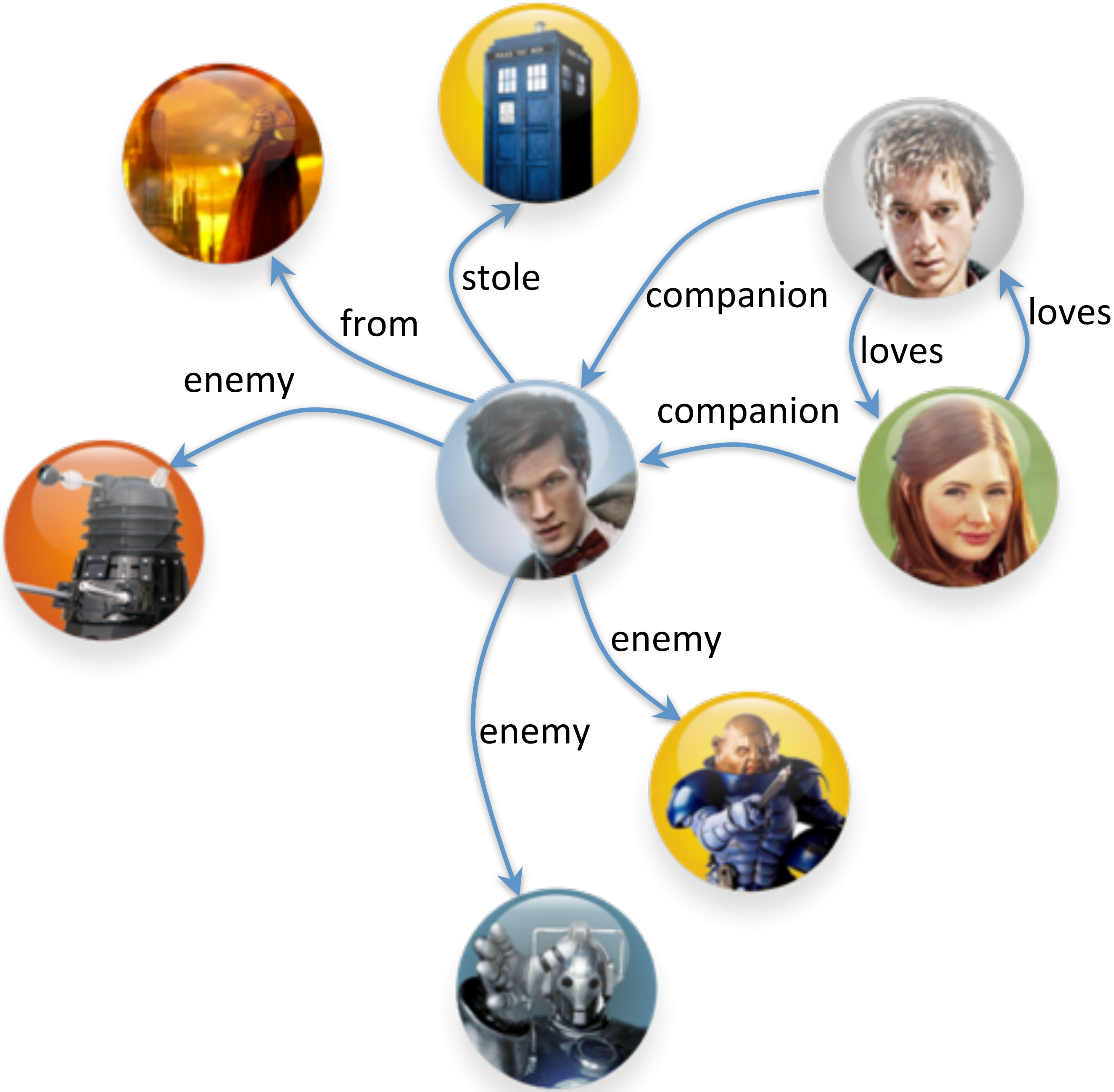

stole
from


stole
from
enemy

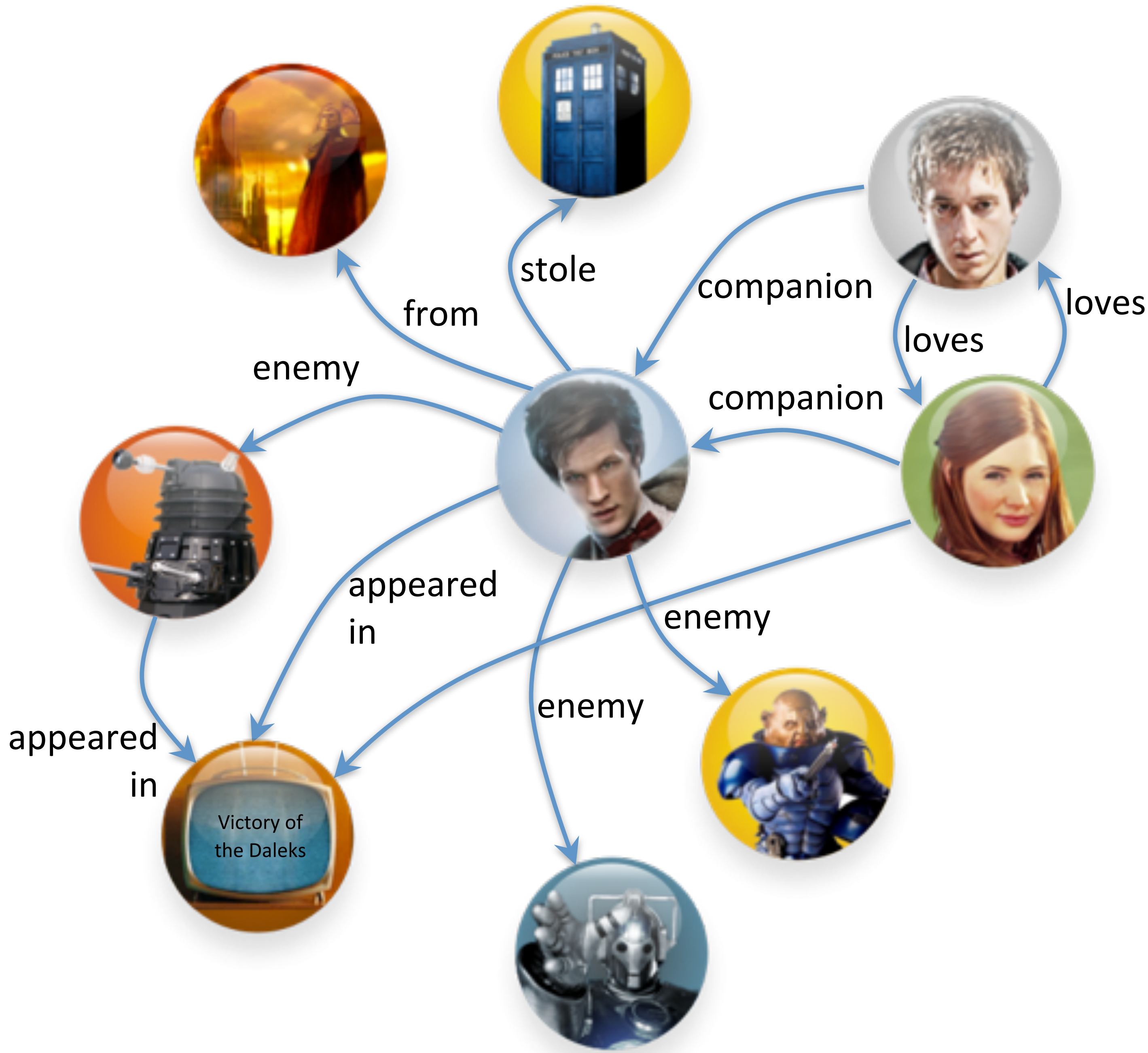

stole
from
enemy
enemy

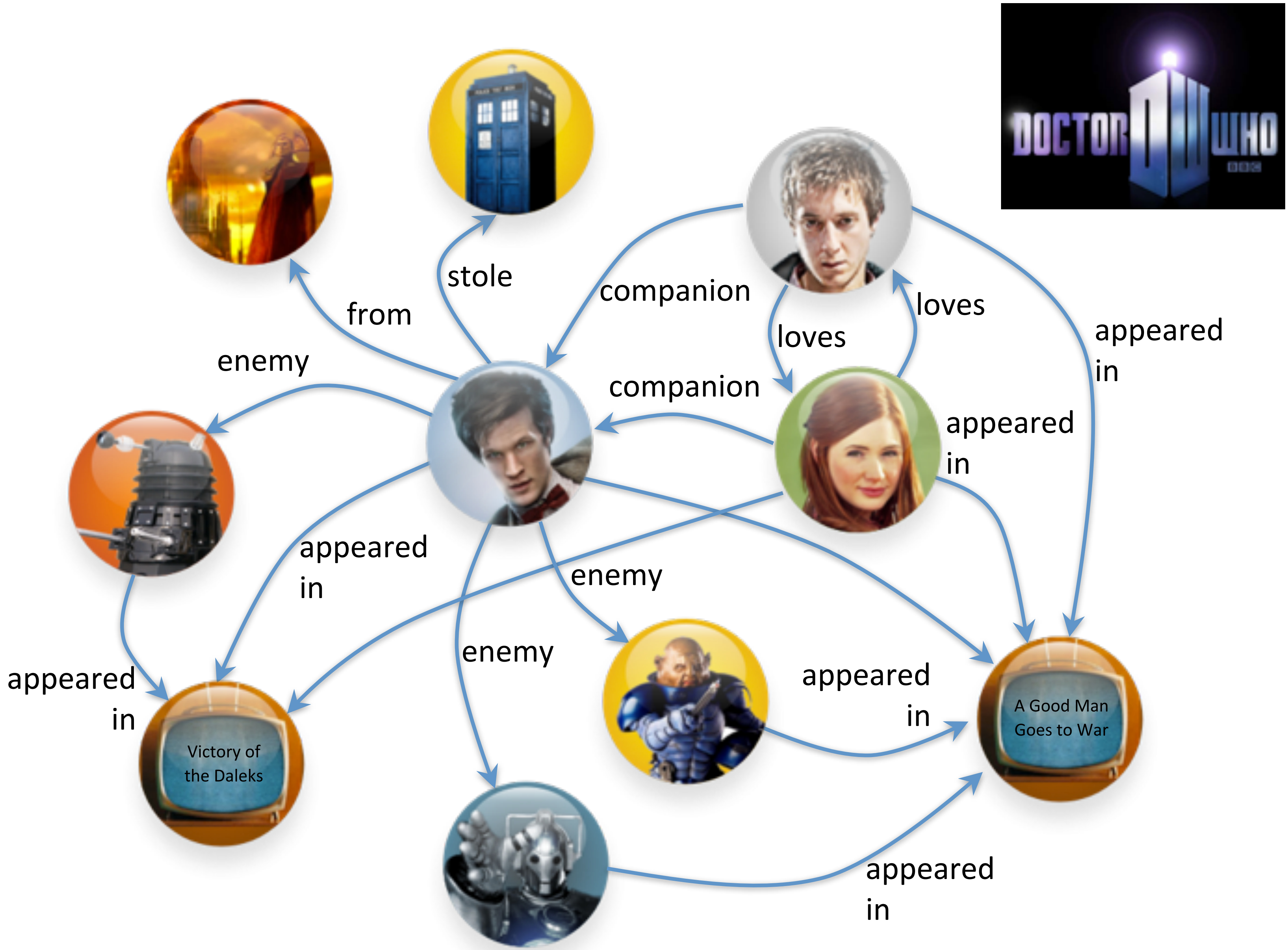

stole
from
enemy
enemy
enemy

stole
from
companion
enemy
companion
enemy
enemy


stole
from
companion
loves
loves
enemy
companion
enemy
enemy

DOCTOR WHO


stole
from
companion
loves
loves
enemy
companion
appeared in
enemy
enemy
appeared in
Victory of the Daleks



DOCTOR WHO
stole
companion
from
loves
loves
appeared in
enemy
companion
appeared in
appeared in
enemy
appeared in
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");
```

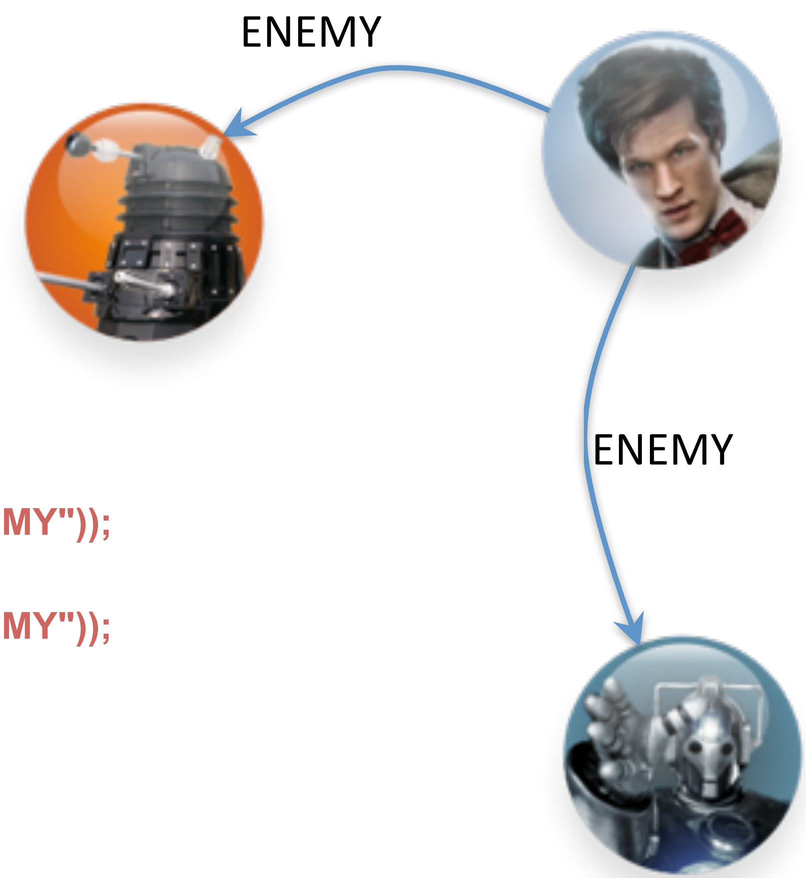
```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");

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"));
```



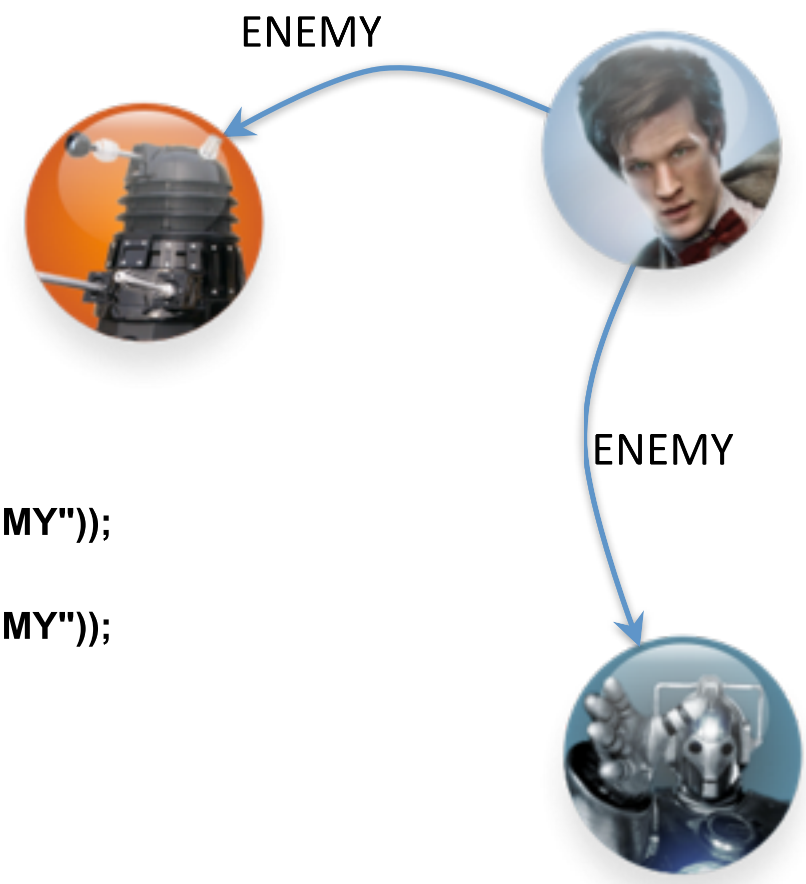
```
GraphDatabaseService db = new EmbeddedGraphDatabase("/data/drwho");

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"));
```



Cypher

(daleks)<-[:ENEMY]-(doctor)-[:ENEMY]->(cybermen)

Cypher

(doctor)-[:ENEMY]->(daleks),

(doctor)-[:ENEMY]->(cybermen)

Create Graph Using Cypher

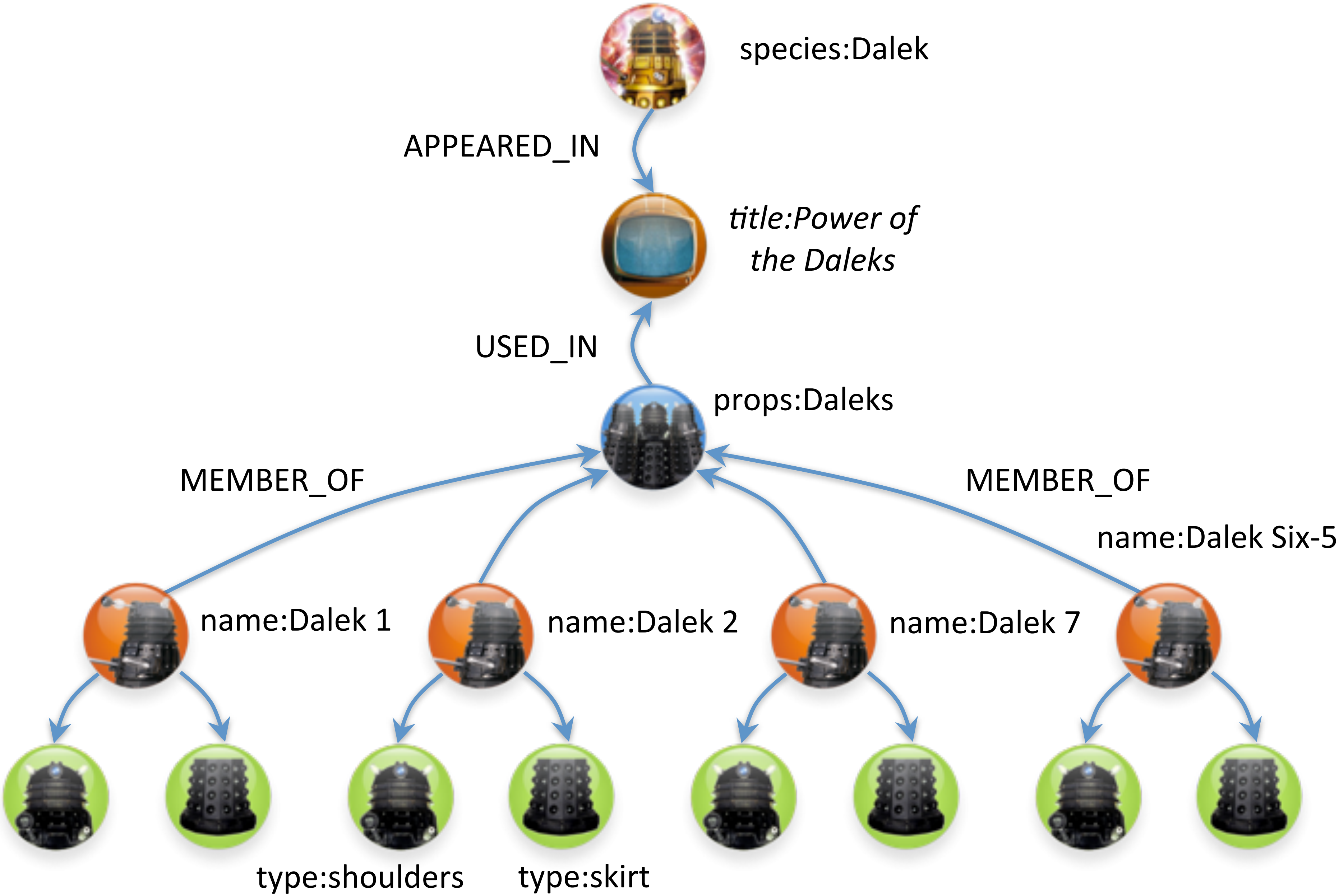

```
CREATE doctor = {name : 'The Doctor'},
       daleks = {name : 'Daleks'},
       cybermen = {name : 'Cybermen'},
       doctor-[:ENEMY]->daleks,
       doctor-[:ENEMY]->cybermen
RETURN doctor;
```

Create Graph Using Cypher

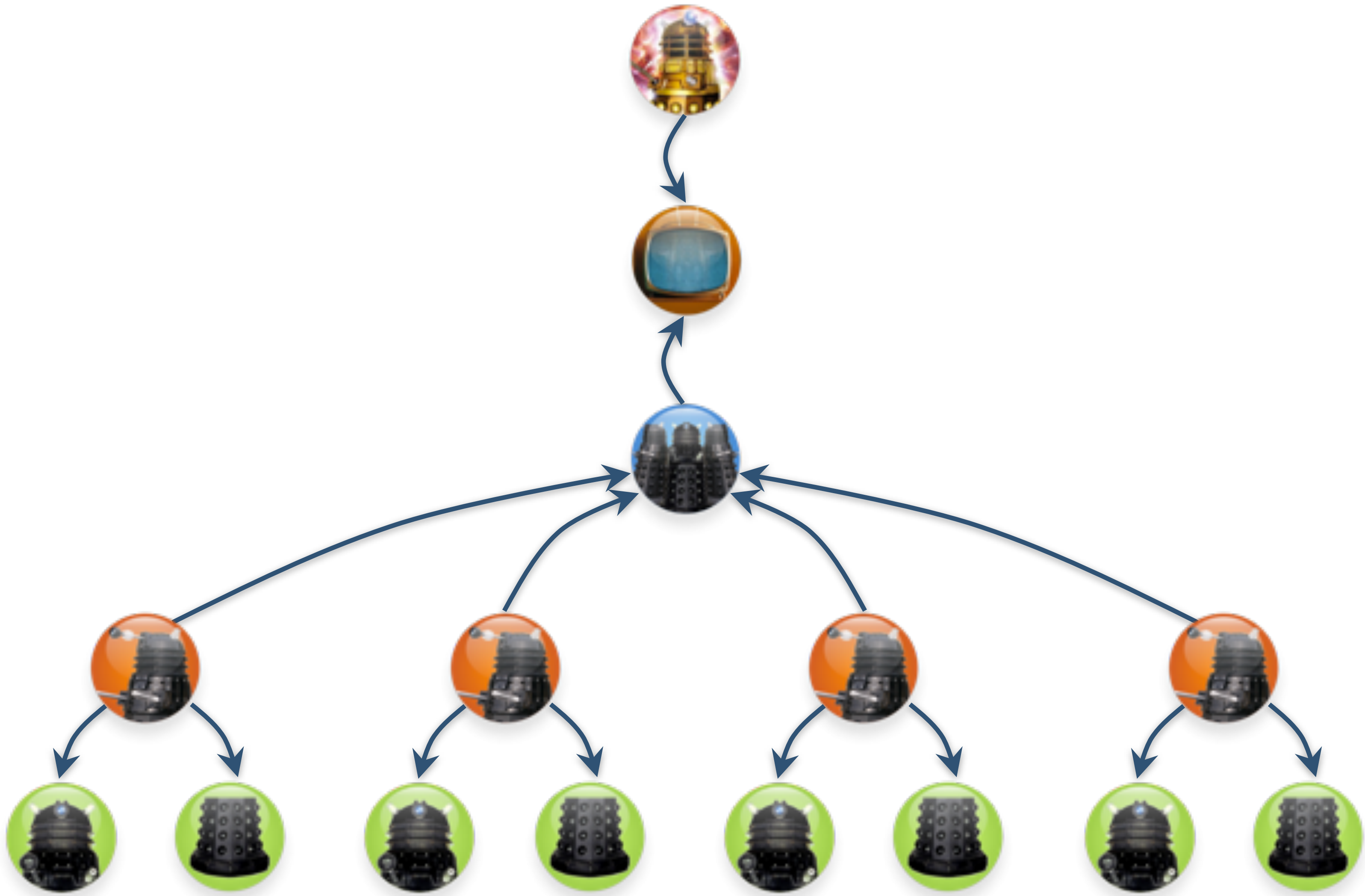

```
CREATE doctor = {name : 'The Doctor'},
       daleks = {name : 'Daleks'},
       cybermen = {name : 'Cybermen'},
       doctor-[:ENEMY]->daleks,
       doctor-[:ENEMY]->cybermen
RETURN doctor;
```

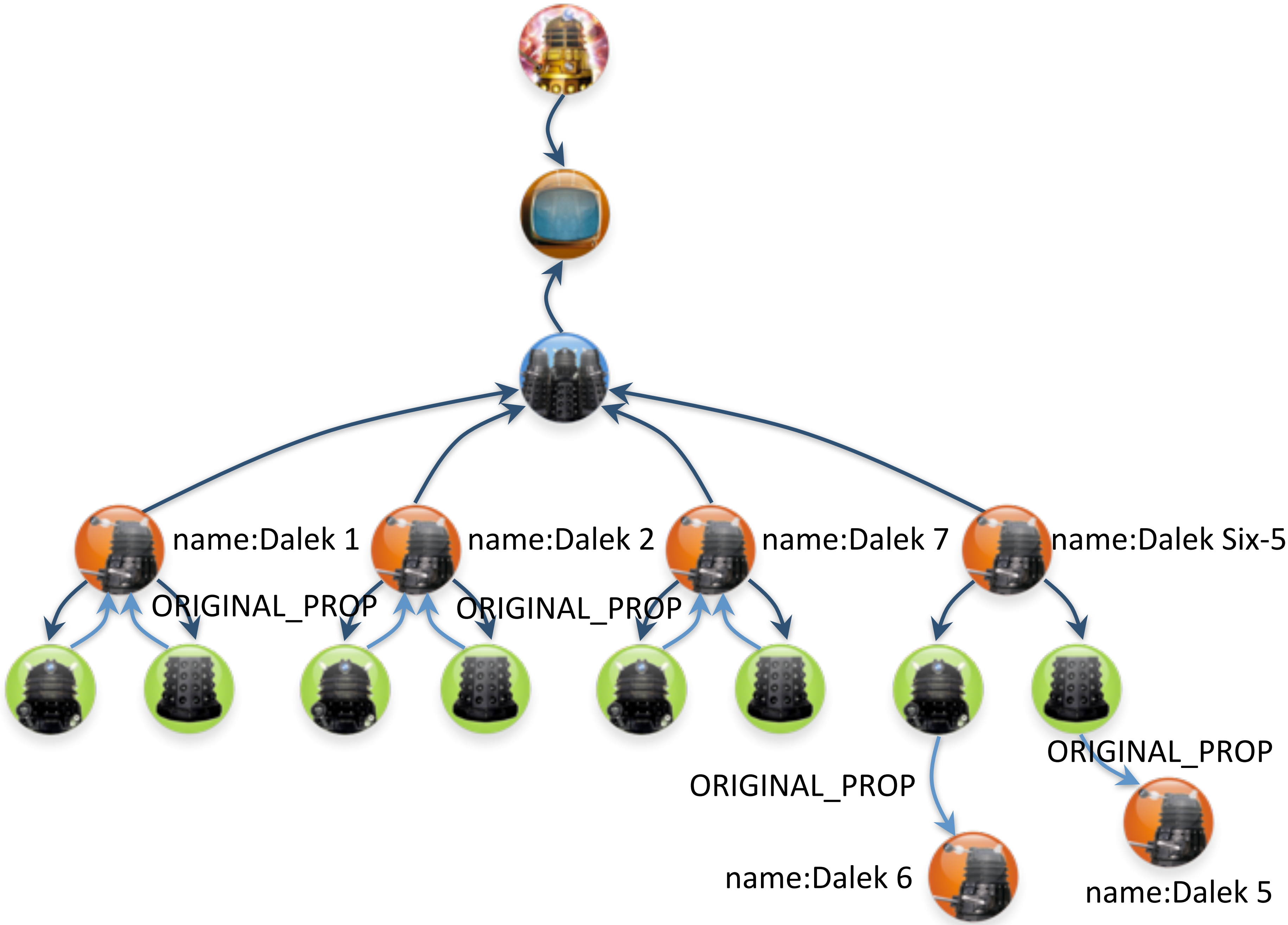
Dalek Props

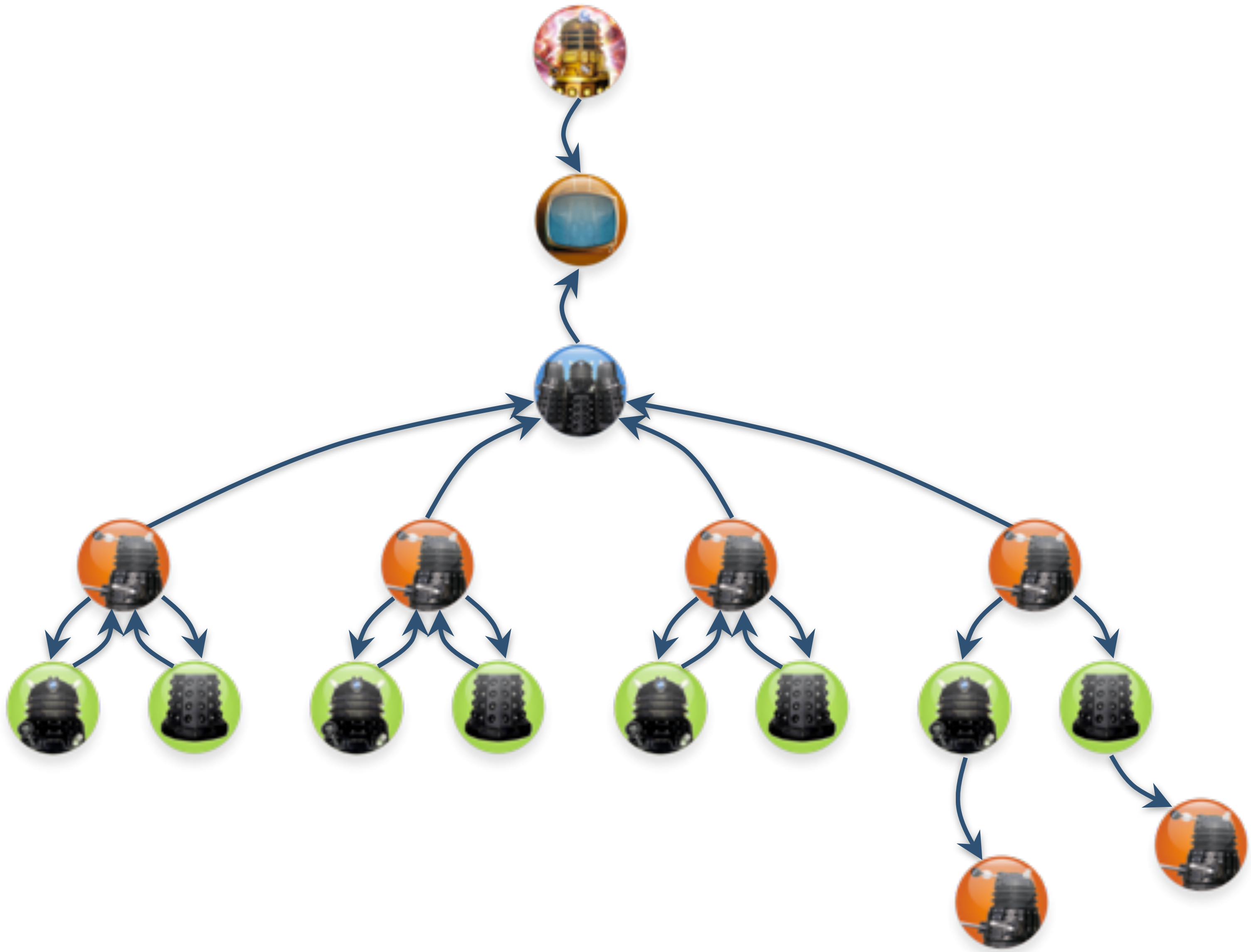
http://www.dalek6388.co.uk/

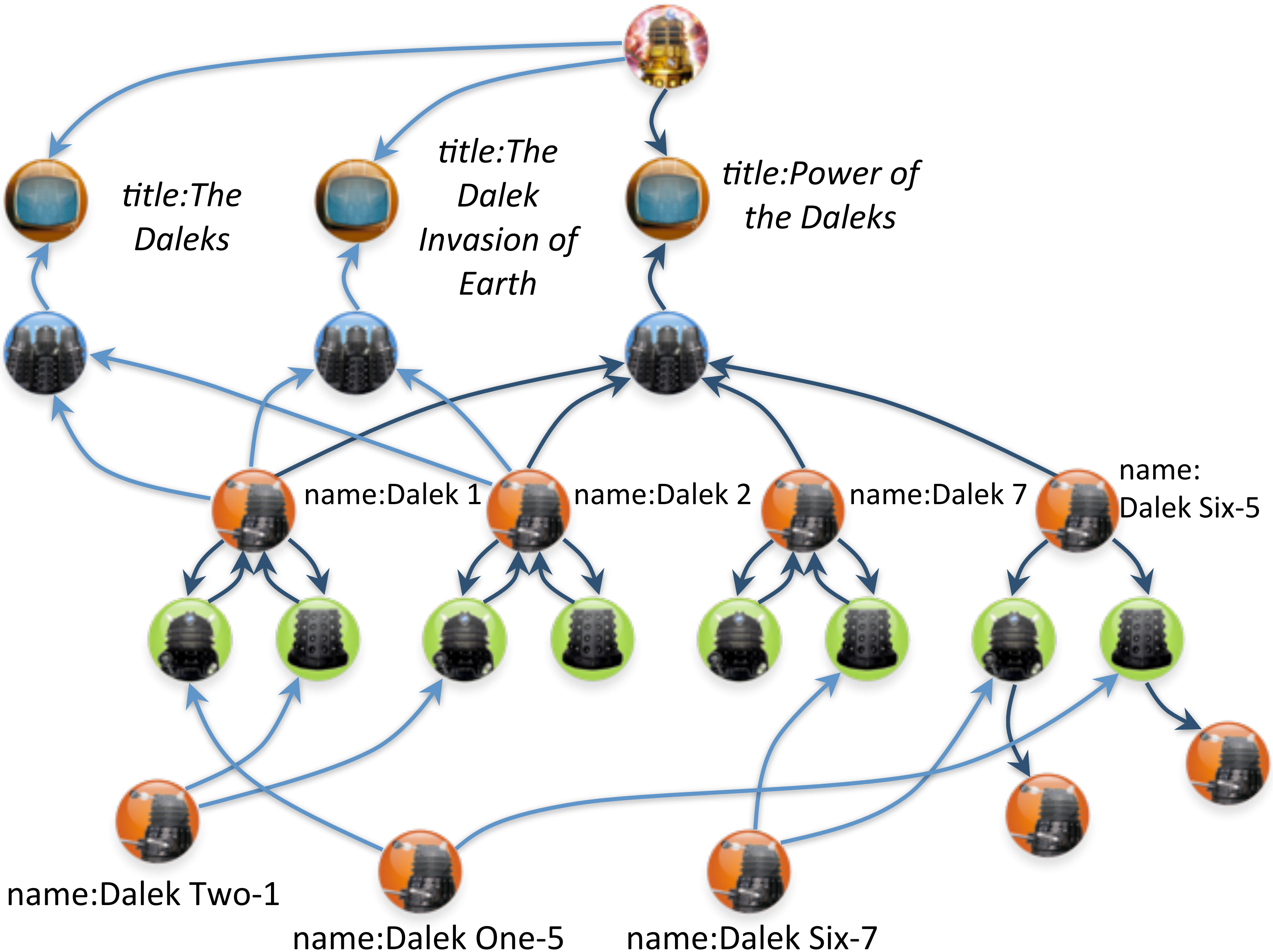


species:Dalek
APPEARED_IN
title:Power of the Daleks
USED_IN
props:Daleks
MEMBER_OF
MEMBER_OF
name:Dalek Six-5
name:Dalek 1
name:Dalek 2
name:Dalek 7
type:shoulders
type:skirt

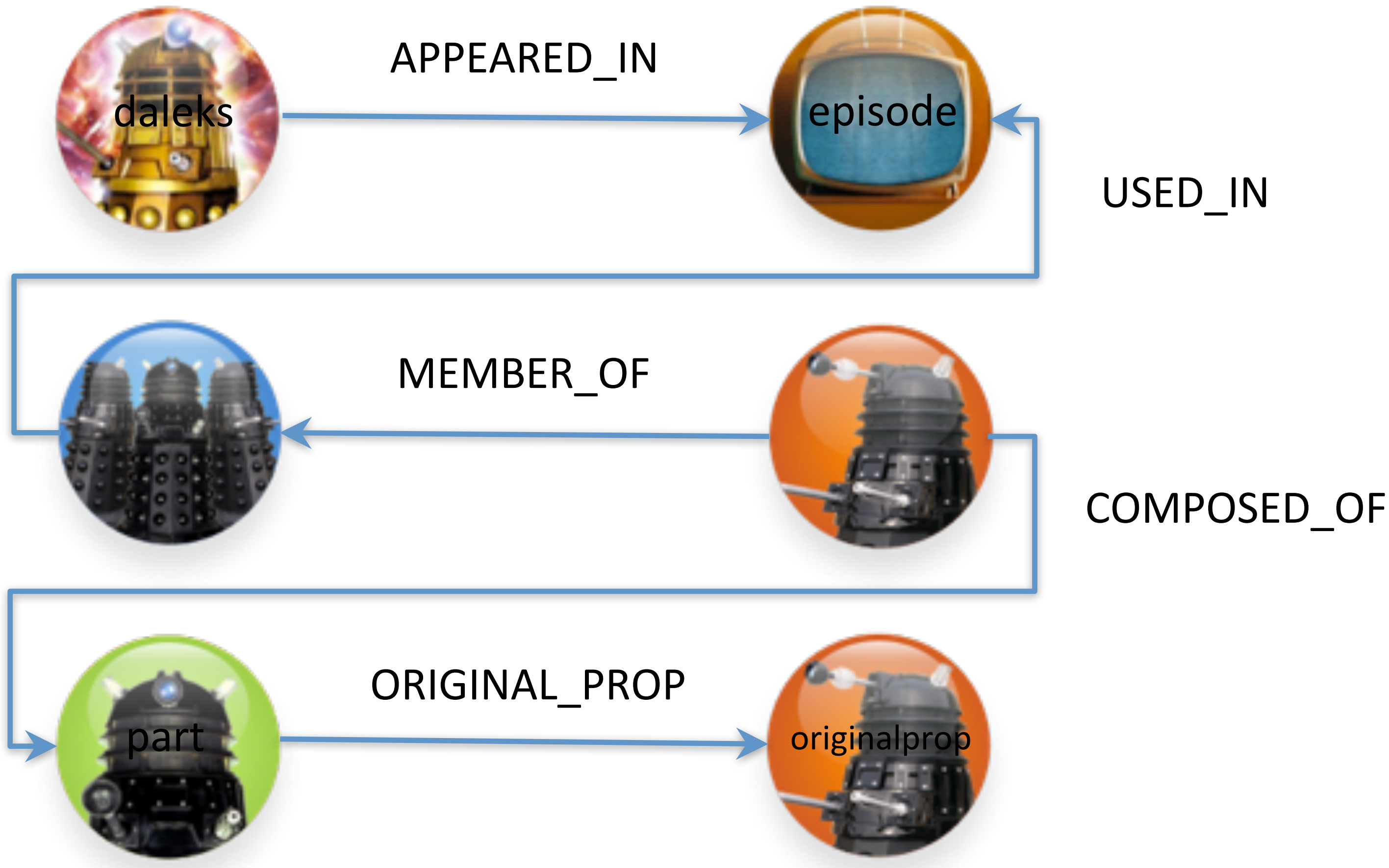


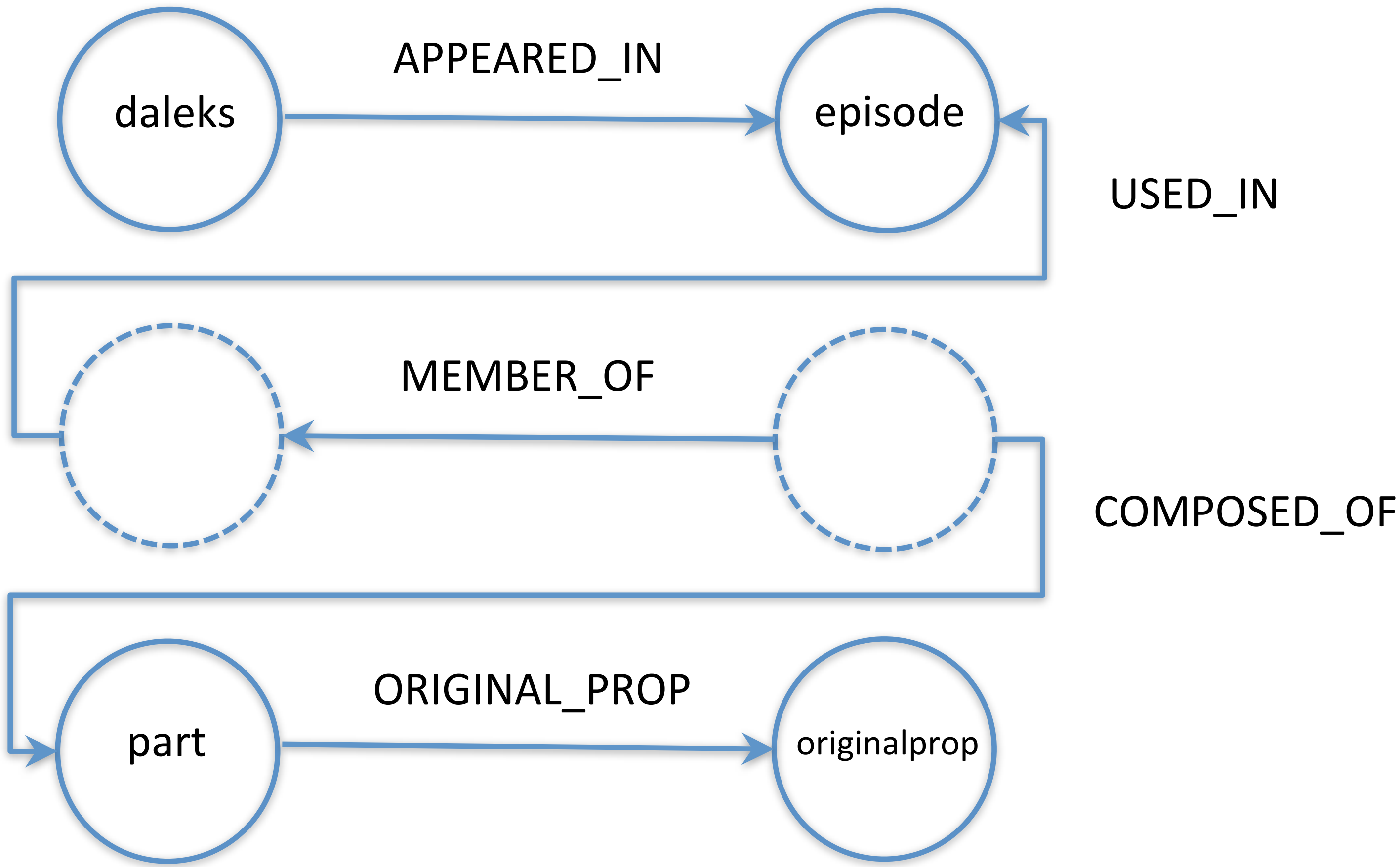

name:Dalek 1
name:Dalek 2
name:Dalek 7
name:Dalek Six-5
ORIGINAL_PROP
ORIGINAL_PROP
ORIGINAL_PROP
ORIGINAL_PROP
name:Dalek 6
name:Dalek 5



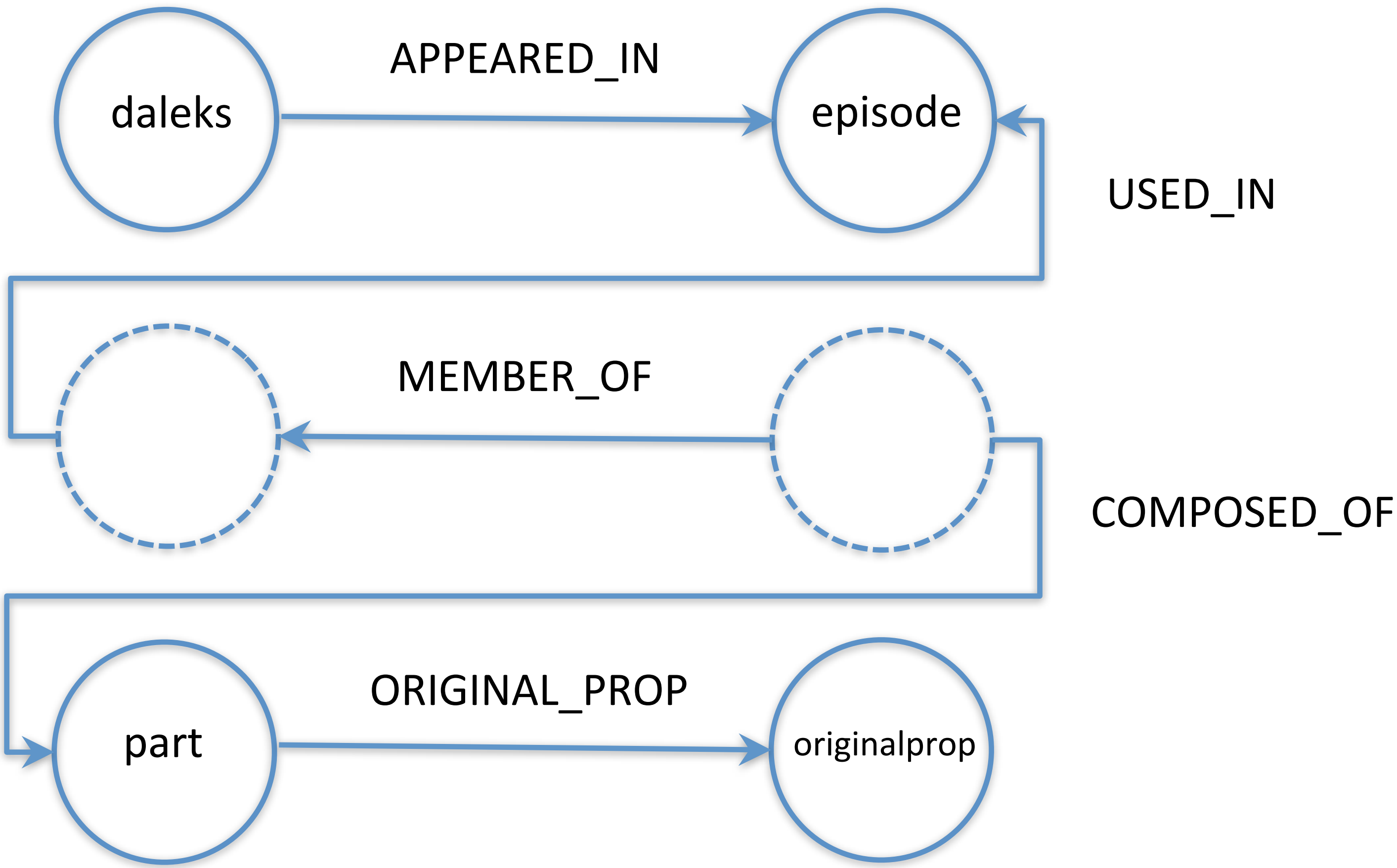

title:The Daleks
title:The Dalek Invasion of Earth
title:Power of the Daleks
name:Dalek 1
name:Dalek 2
name:Dalek 7
name: Dalek Six-5
name:Dalek Two-1
name:Dalek One-5
name:Dalek Six-7

Supply Chain Traceability

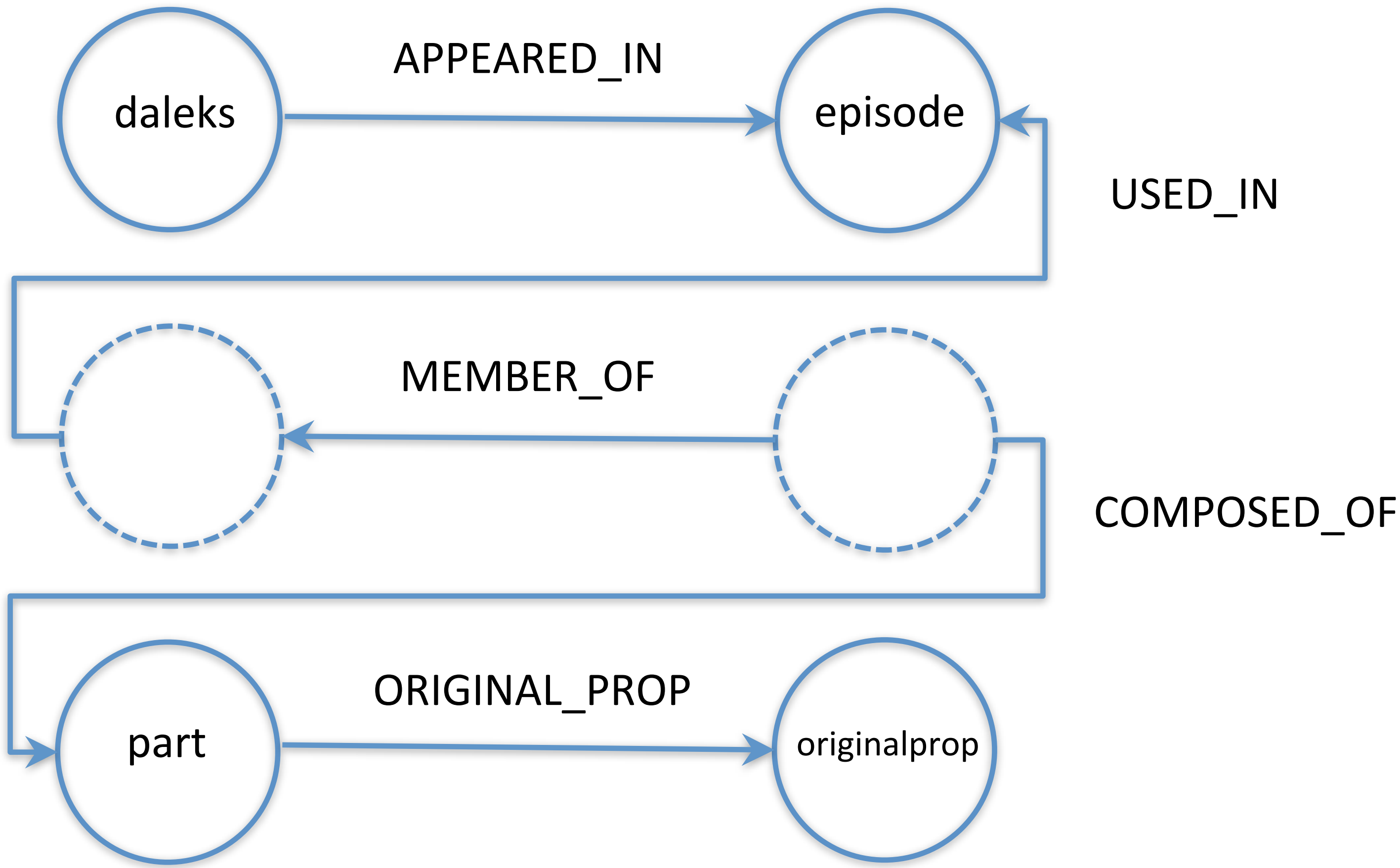

APPEARED_IN
daleks
episode
USED_IN
MEMBER_OF
COMPOSED_OF
ORIGINAL_PROP
part
originalprop


daleks
APPEARED_IN
episode
USED_IN
MEMBER_OF
COMPOSED_OF
part
ORIGINAL_PROP
originalprop

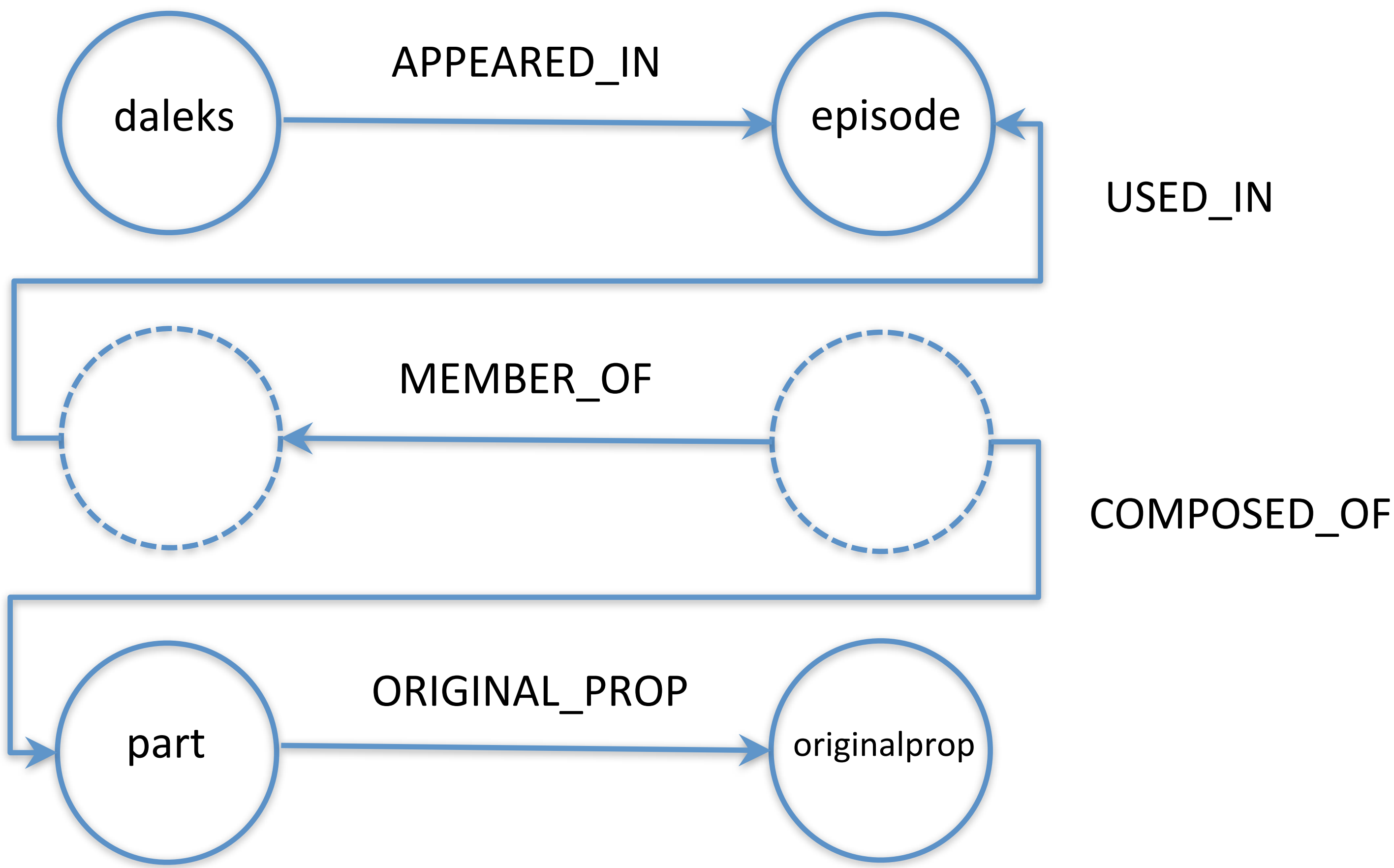
(daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-



(daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->



(daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)



Cypher Query

```
START  daleks=node:species(species='Dalek')
MATCH  daleks-[:APPEARED_IN]->episode<-[:USED_IN]-
       ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
       part-[:ORIGINAL_PROP]->originalprop
RETURN originalprop.name AS prop,
       part.type AS part,
       count(episode) AS episode_count,
       collect(episode.title) AS episodes
ORDER BY episode_count desc
LIMIT 1;
```

Index Lookup

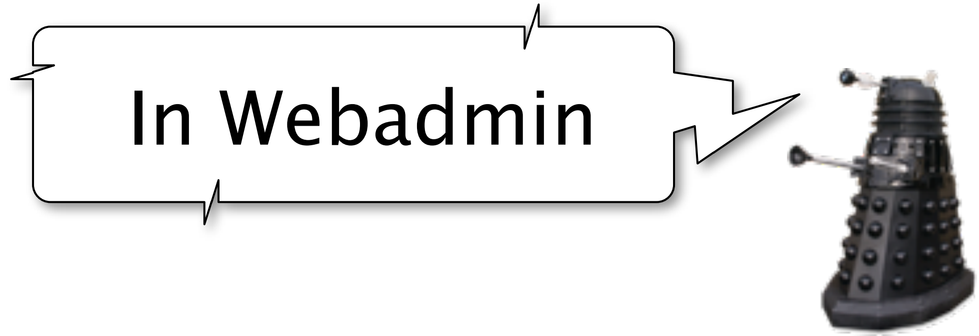
```
START  daleks=node:species(species='Dalek')
MATCH  daleks-[:APPEARED_IN]->episode<-[:USED_IN]-
       ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
       part-[:ORIGINAL_PROP]->originalprop
RETURN originalprop.name AS prop,
       part.type AS part,
       count(episode) AS episode_count,
       collect(episode.title) AS episodes
ORDER BY episode_count desc
LIMIT 1;
```

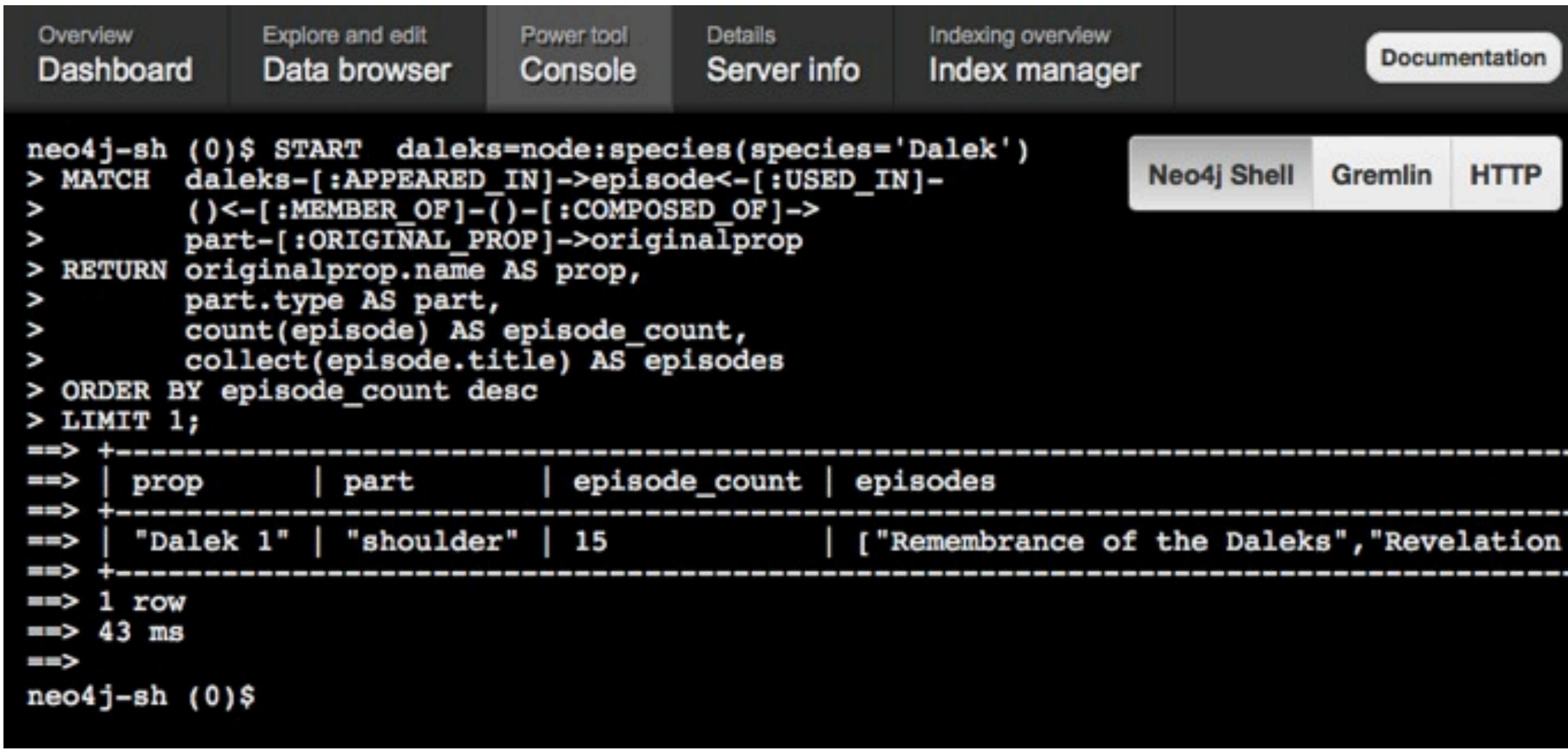
Match Nodes & Relationships

```
START  daleks=node:species(species='Dalek')
MATCH  daleks-[:APPEARED_IN]->episode<-[:USED_IN]-
    ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
    part-[:ORIGINAL_PROP]->originalprop
RETURN originalprop.name AS prop,
    part.type AS part,
    count(episode) AS episode_count,
    collect(episode.title) AS episodes
ORDER BY episode_count desc
LIMIT 1;
```

Return Values

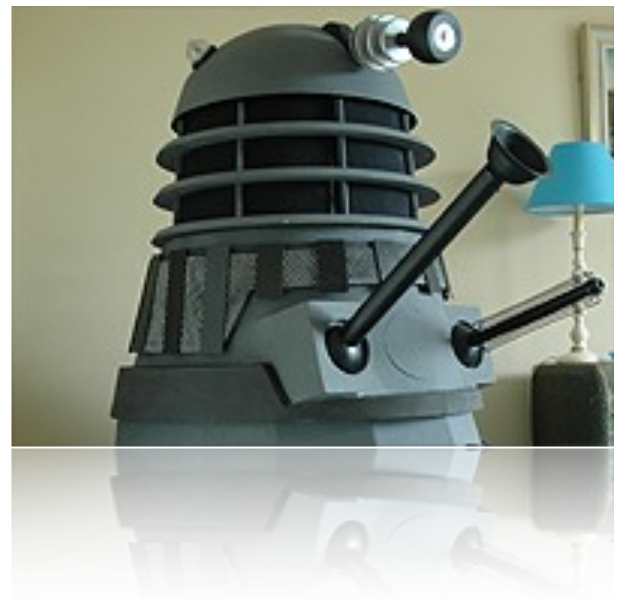
```
START  daleks=node:species(species='Dalek')
MATCH  daleks-[:APPEARED_IN]->episode<-[:USED_IN]-
       ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
       part-[:ORIGINAL_PROP]->originalprop
RETURN originalprop.name AS prop,
       part.type AS part,
       count(episode) AS episode_count,
       collect(episode.title) AS episodes
ORDER BY episode_count desc
LIMIT 1;
```


In Webadmin


Overview
Dashboard
Explore and edit
Data browser
Power tool
Console
Details
Server info
Indexing overview
Index manager
Documentation
Neo4j Shell
Gremlin
HTTP
neo4j-sh (0)$ START daleks=node:species(species='Dalek')
> MATCH daleks-[:APPEARED_IN]->episode<-[:USED_IN]-
>   ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
>   part-[:ORIGINAL_PROP]->originalprop
> RETURN originalprop.name AS prop,
>   part.type AS part,
>   count(episode) AS episode_count,
>   collect(episode.title) AS episodes
> ORDER BY episode_count desc
> LIMIT 1;
==> | prop | part | episode_count | episodes
==> | "Dalek 1" | "shoulder" | 15 | ["Remembrance of the Daleks","Revelation
==> 1 row
==> 43 ms
neo4j-sh (0)$

The Hardest Working Prop Part

Dalek One's shoulders



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

Download

http://neo4j.org/download/

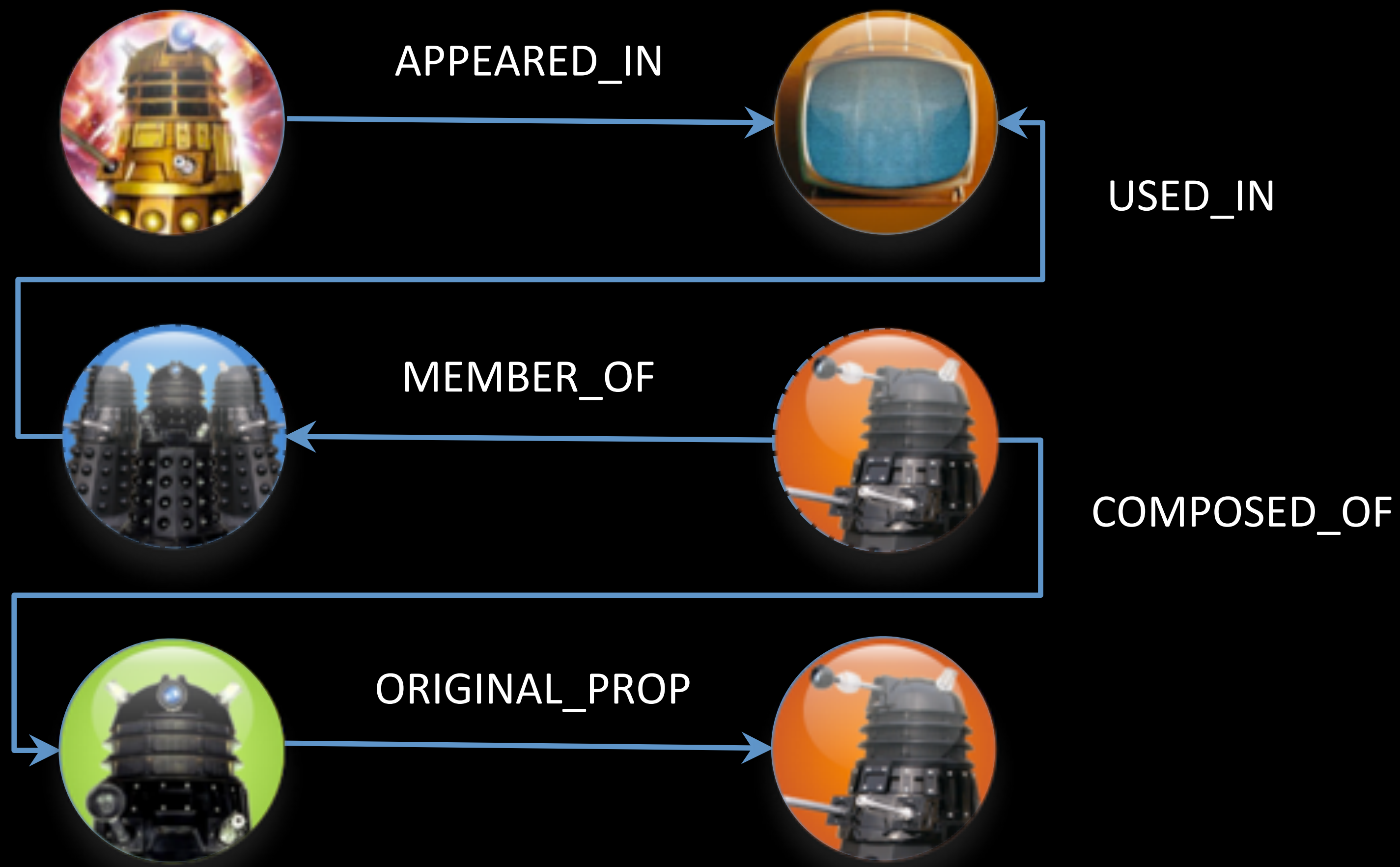
Tutorial

https://github.com/jimwebber/neo4j-tutorial

Questions?

@iansrobinson
ian.robinson@neotechnology.com

match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->(part)-[:ORIGINAL_PROP]->(originalprop)



```
match (daleks)-[:APPEARED_IN]->(episode)<-[:USED_IN]-
    ()<-[:MEMBER_OF]-()-[:COMPOSED_OF]->
    (part)-[:ORIGINAL_PROP]->(originalprop)
```