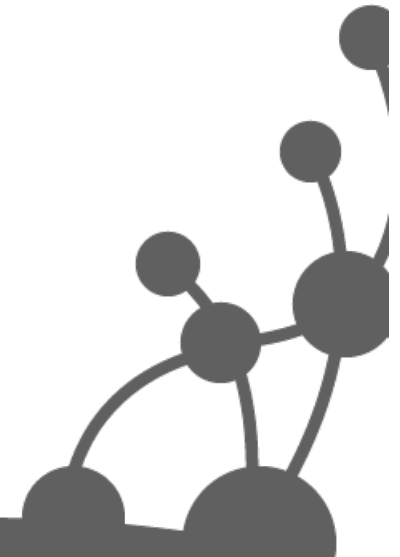


A Little Graph Theory for the Busy Developer

Dr. Jim Webber

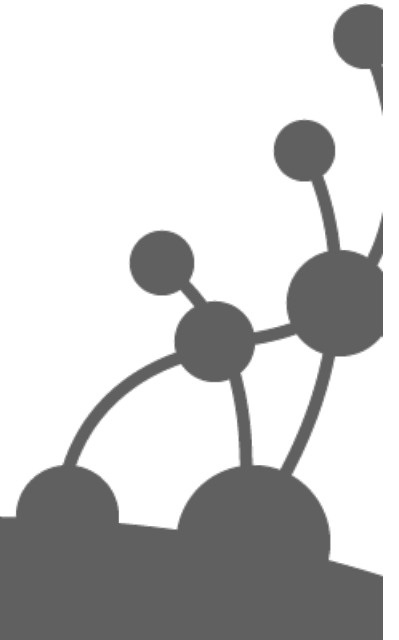
Chief Scientist, Neo Technology

@jimwebber



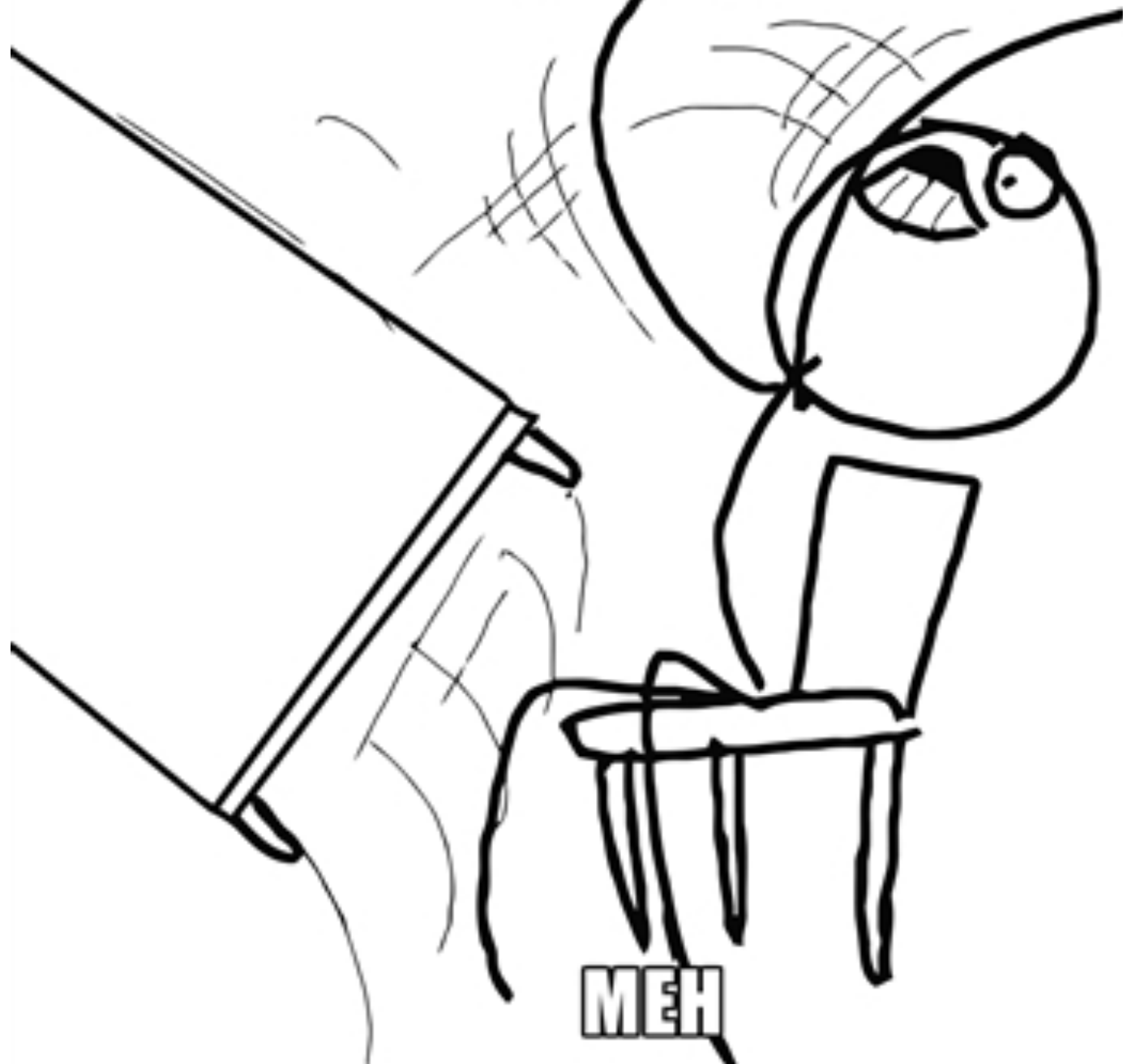
Roadmap

- Imprisoned data
- Graph models
- Graph theory
 - Local properties, global behaviours
 - Predictive analytics
- Graph matching
 - Predictive, real-time analytics for fun and profit
- Fin





TABLES?



MEH





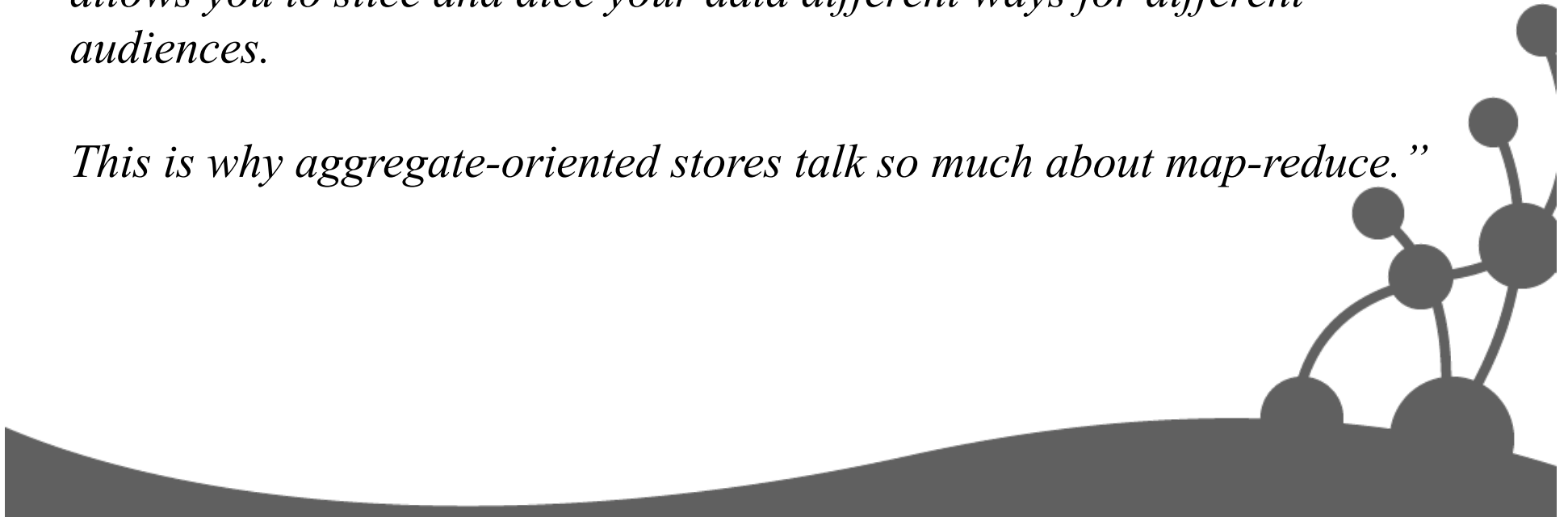


Aggregate-Oriented Data

<http://martinfowler.com/bliki/AggregateOrientedDatabase.html>

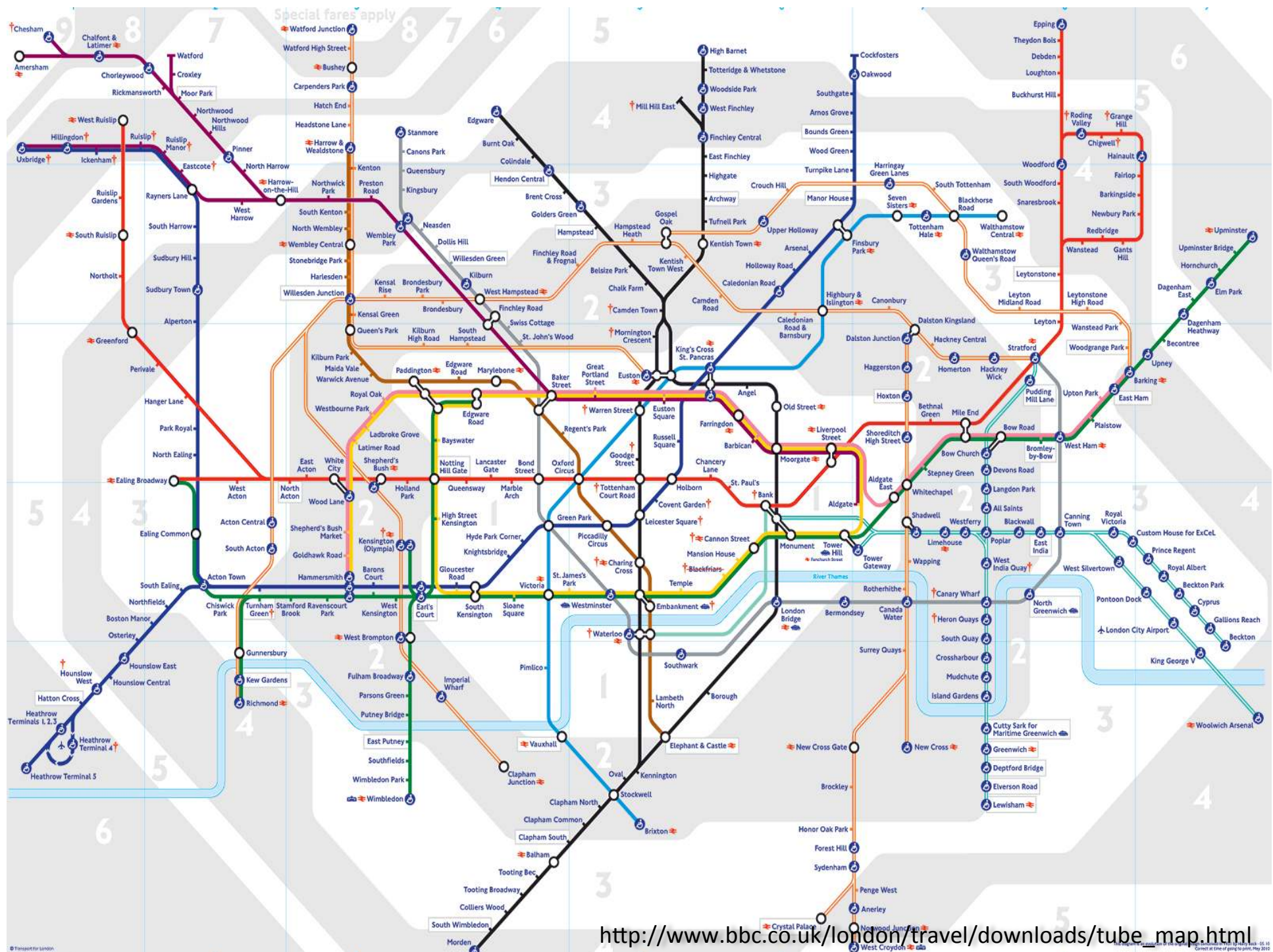
“There is a significant downside - the whole approach works really well when data access is aligned with the aggregates, but what if you want to look at the data in a different way? Order entry naturally stores orders as aggregates, but analyzing product sales cuts across the aggregate structure. The advantage of not using an aggregate structure in the database is that it allows you to slice and dice your data different ways for different audiences.

This is why aggregate-oriented stores talk so much about map-reduce.”



complexity = f(size, connectedness, uniformity)

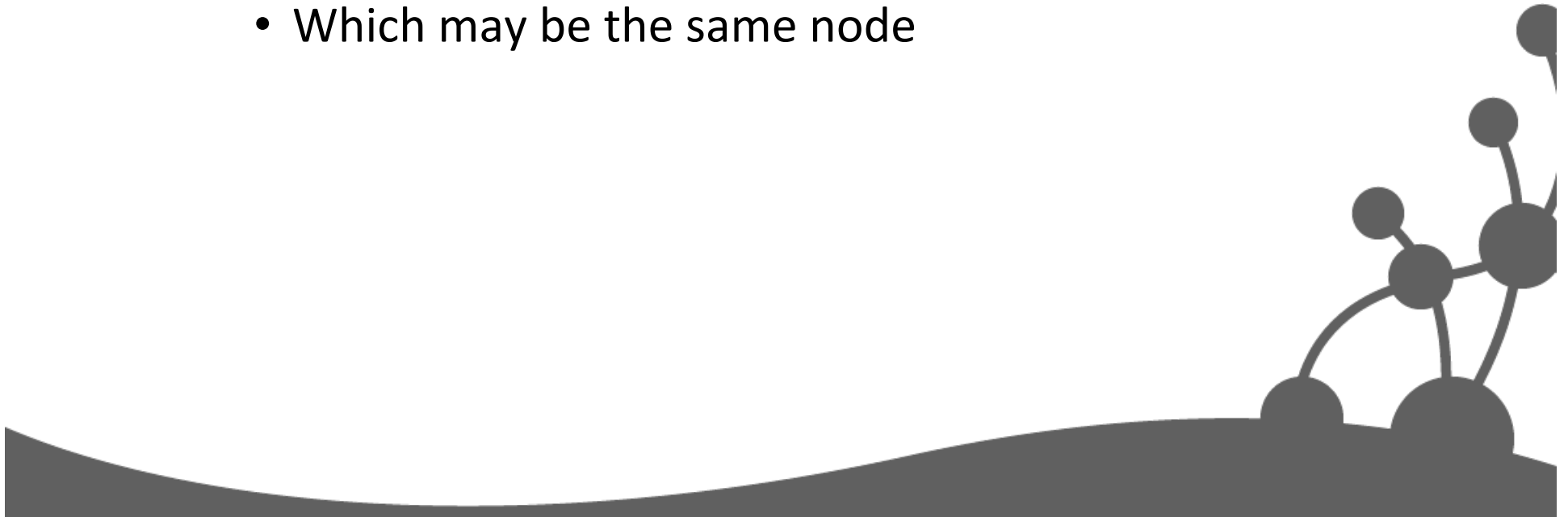




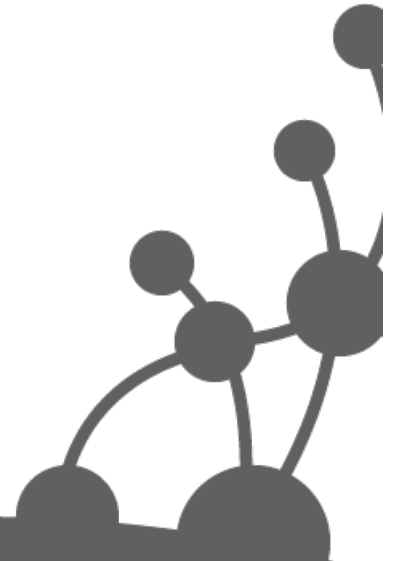
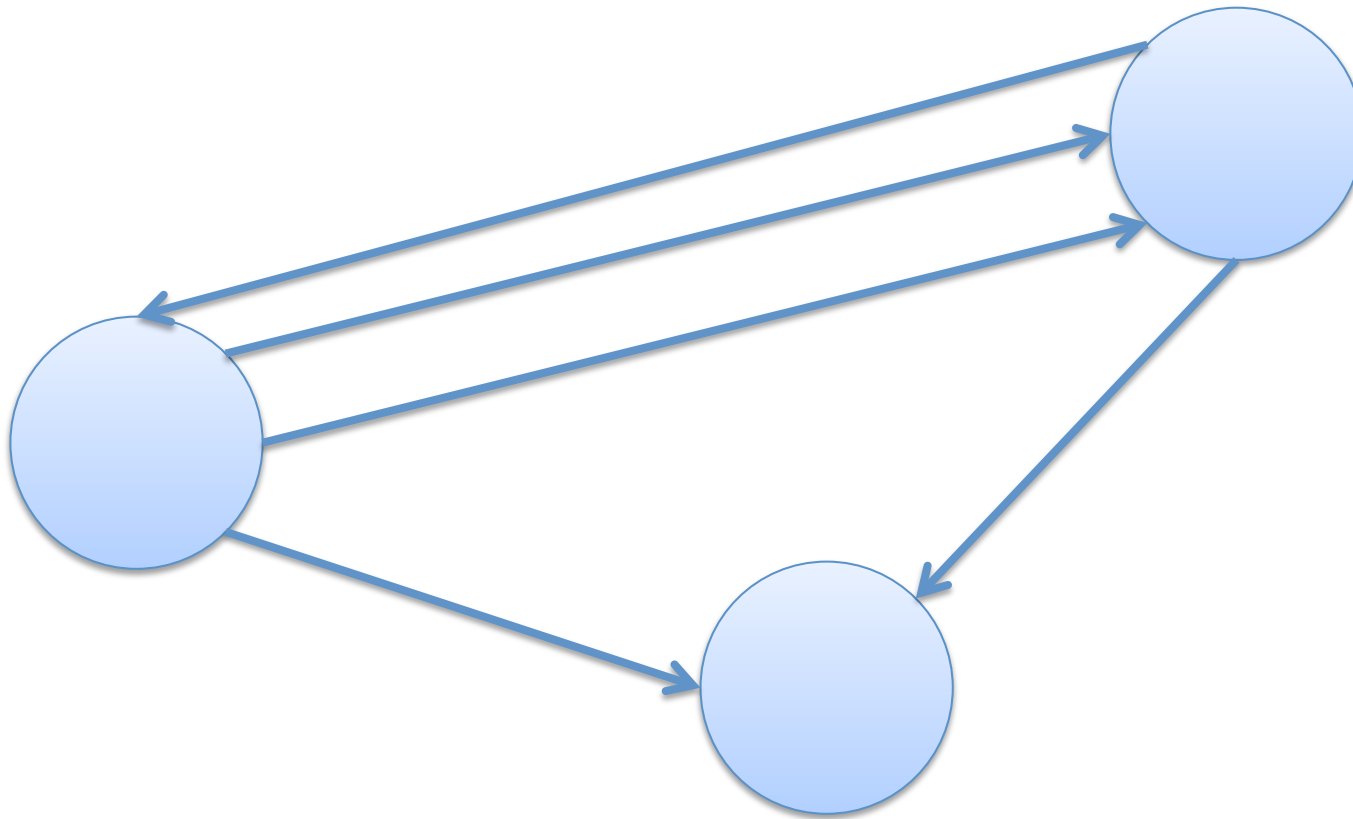
http://www.bbc.co.uk/london/travel/downloads/tube_map.html

Property graphs

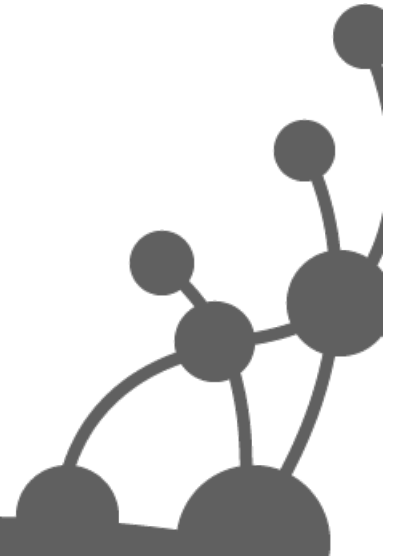
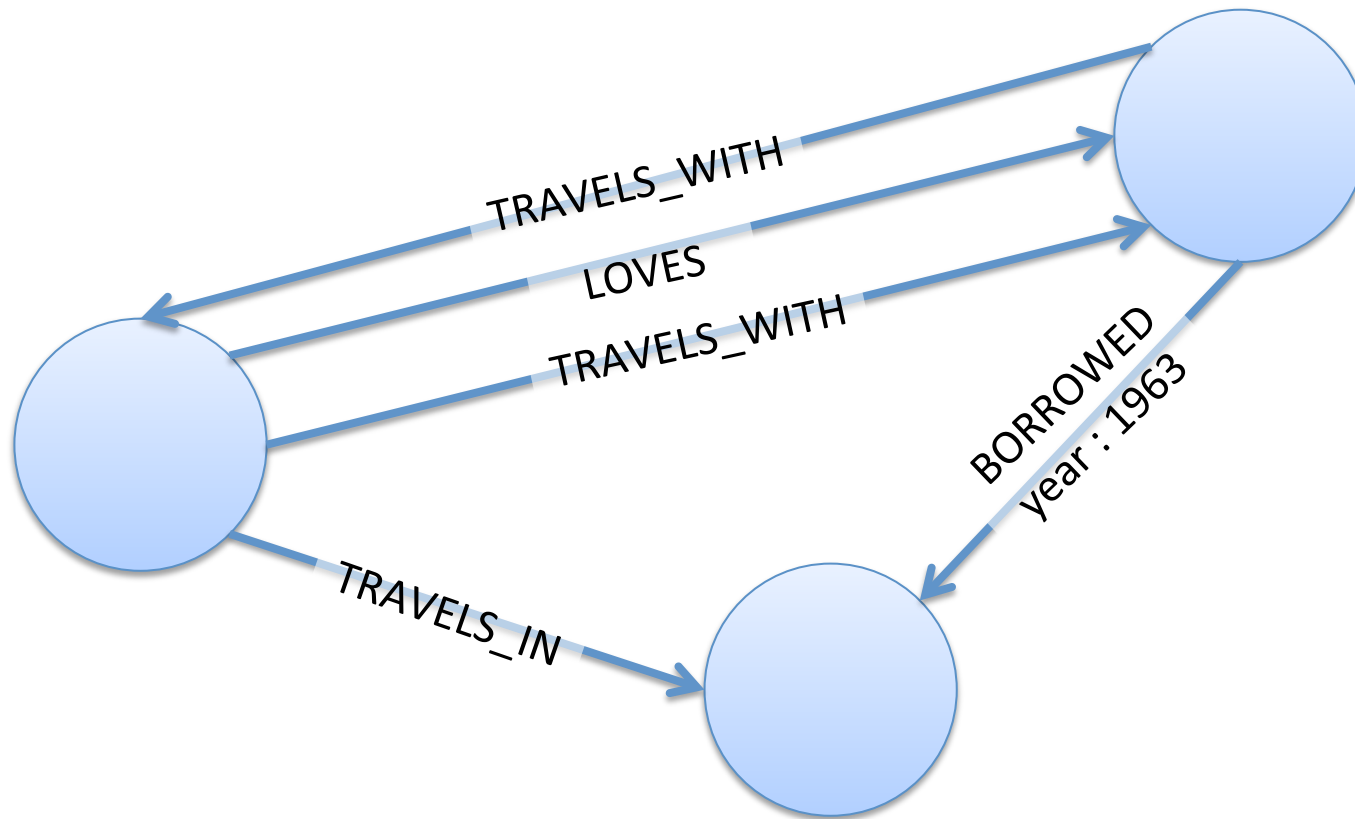
- Property graph model:
 - Nodes with properties
 - Named, directed relationships with properties
 - Relationships have exactly one start and end node
 - Which may be the same node



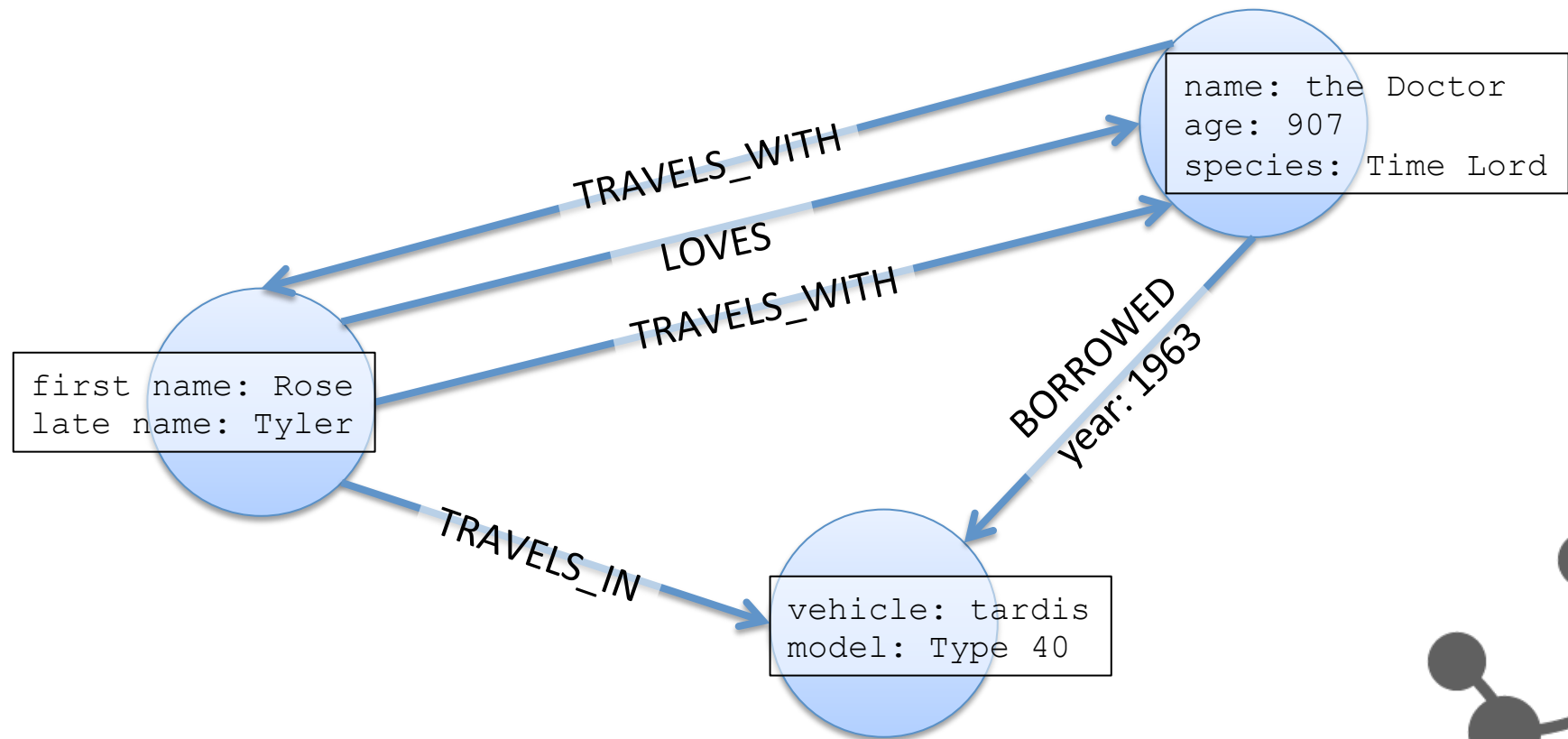
Property Graph Model



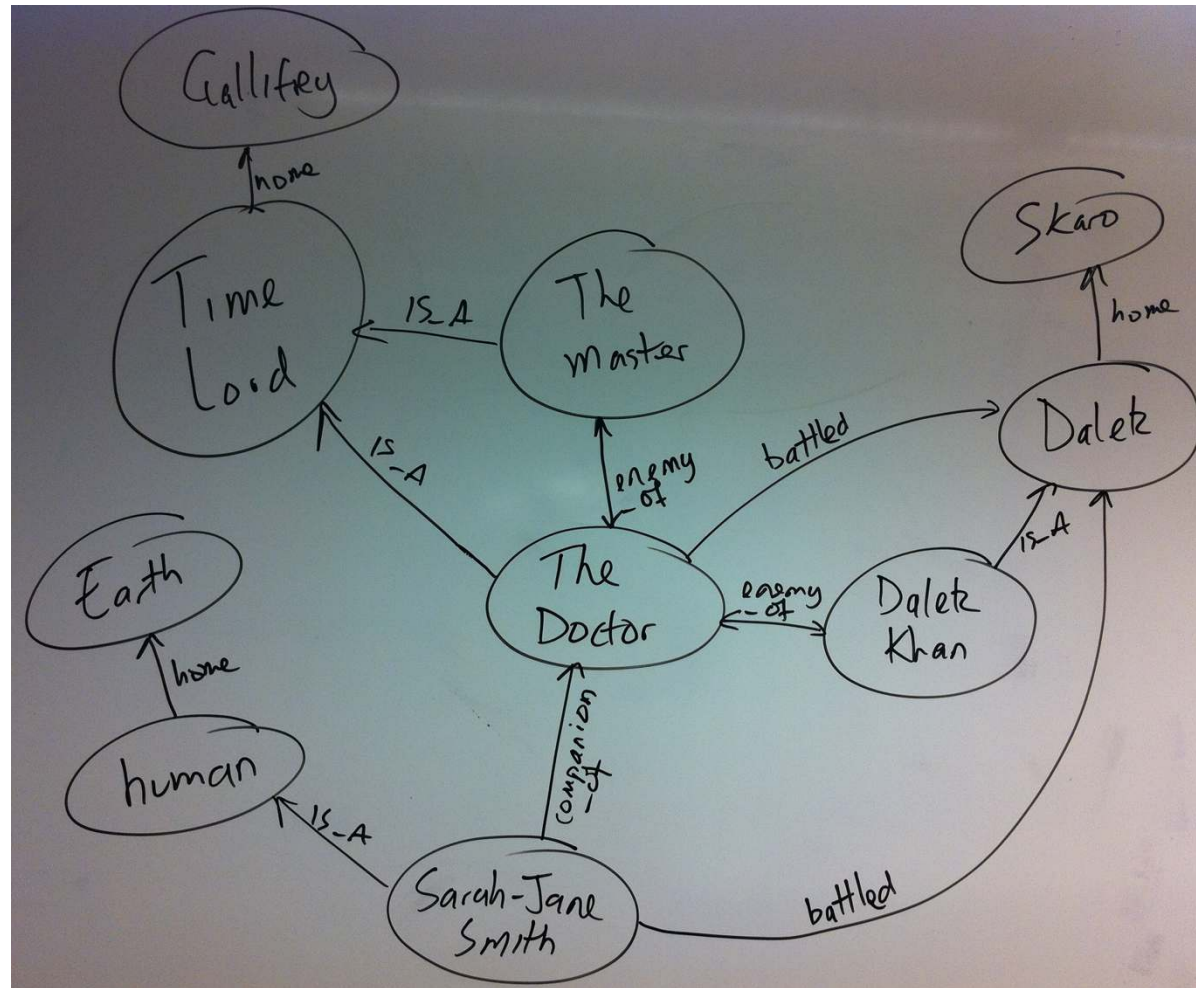
Property Graph Model



Property Graph Model



Property graphs are very whiteboard-friendly

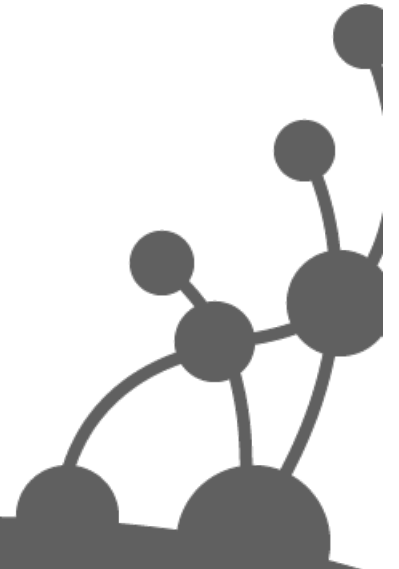


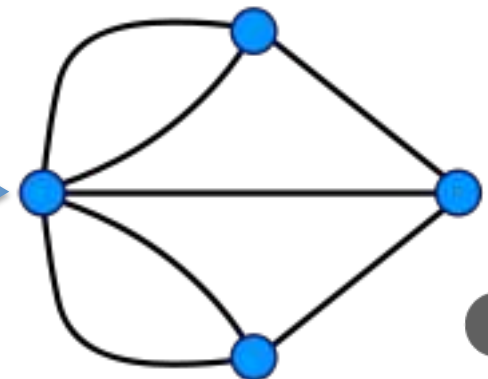
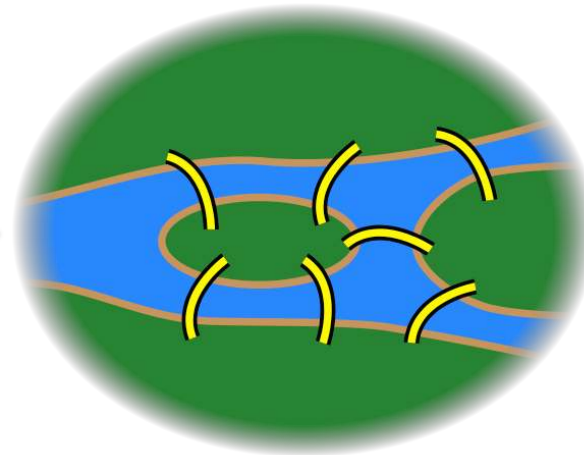
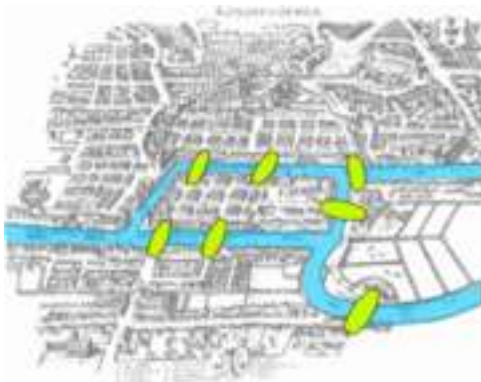




Meet Leonhard Euler

- Swiss mathematician
- Inventor of Graph Theory (1736)





DULWICH SUPERMARKET & OFF LICENCE

ENGLISH , TURKISH , GREEK , MEDITERRANEAN FOOD

18

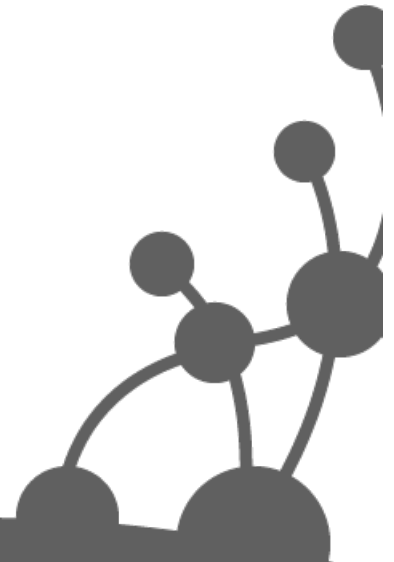
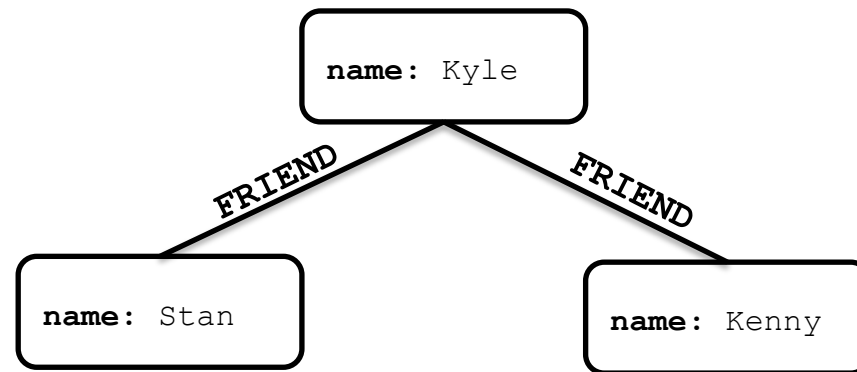
DELICATESSEN & ORGANICS

TEL : 020 8299 2214

FRESHLY CUT SANDWICHES · BILTONG

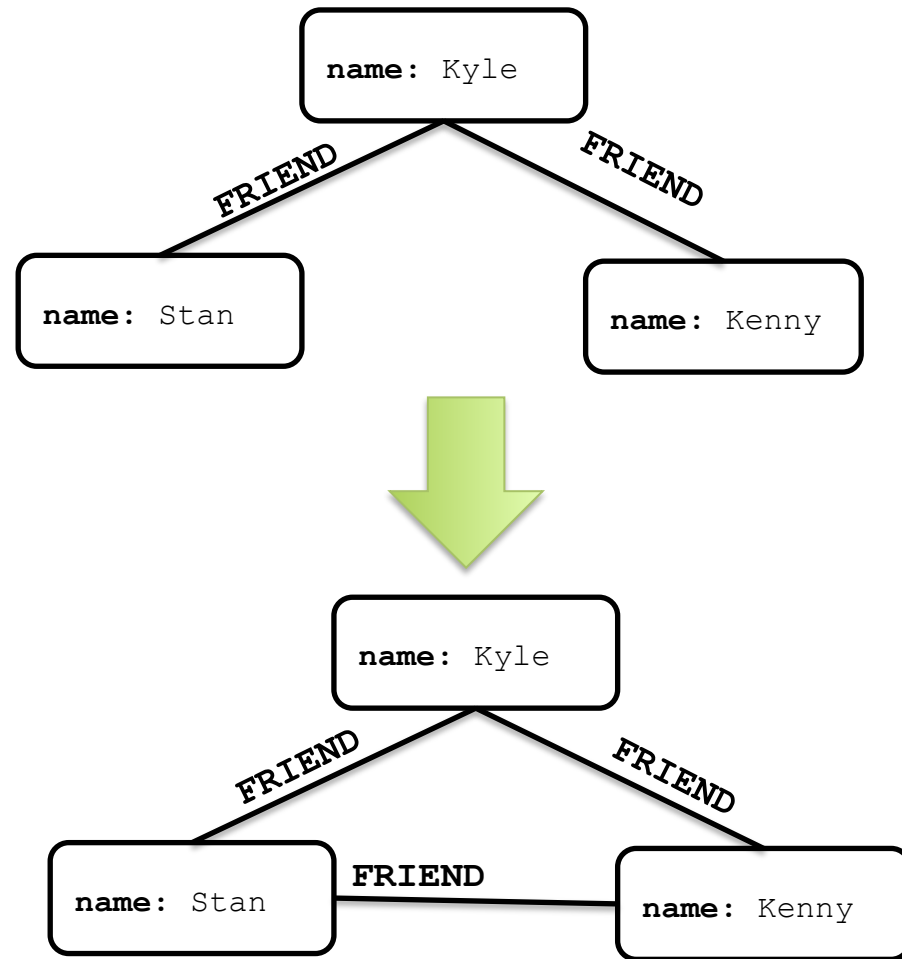


Triadic Closure

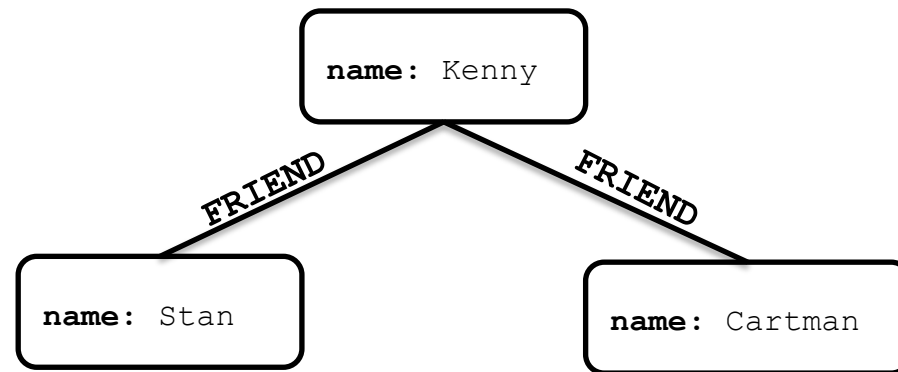


Triadic Closure

(strong relationship)

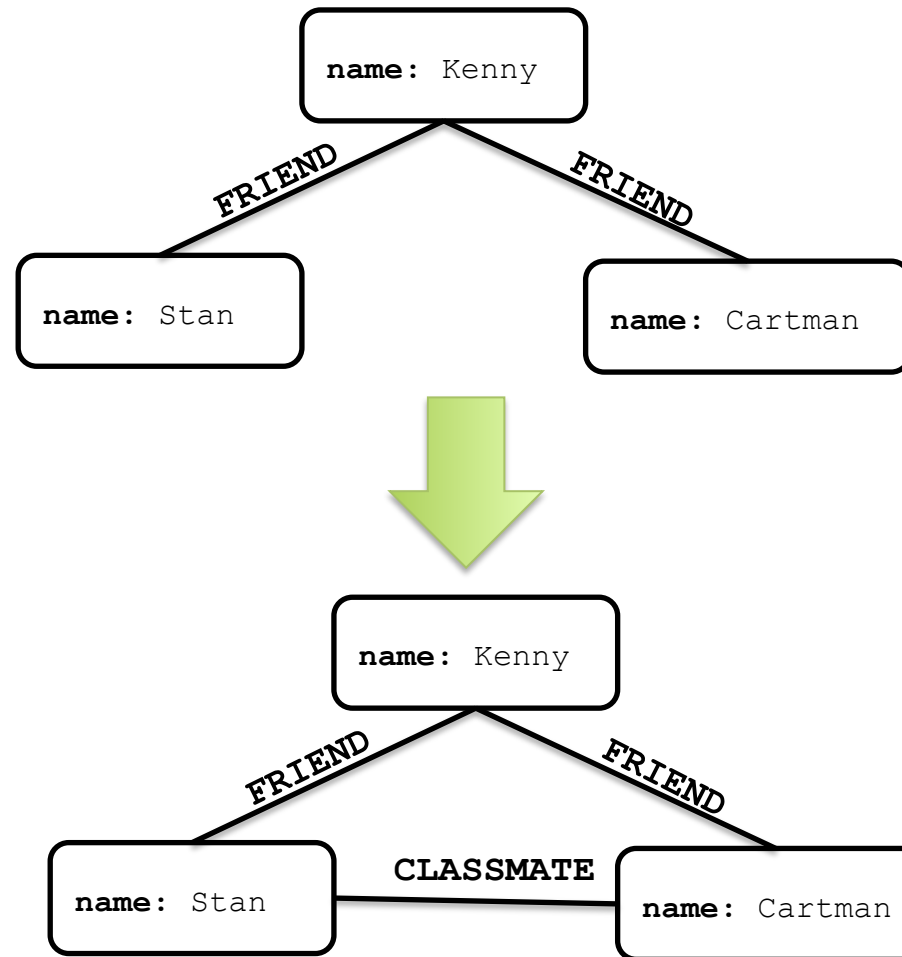


Triadic Closure

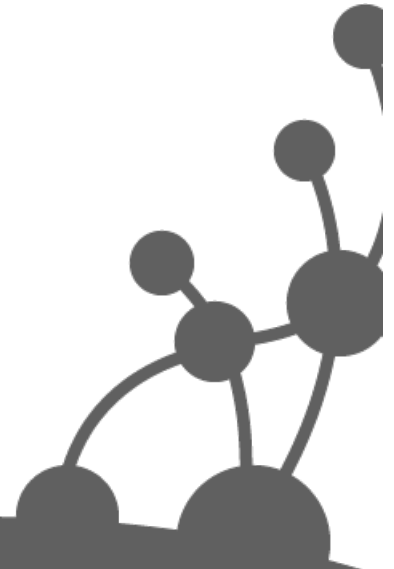
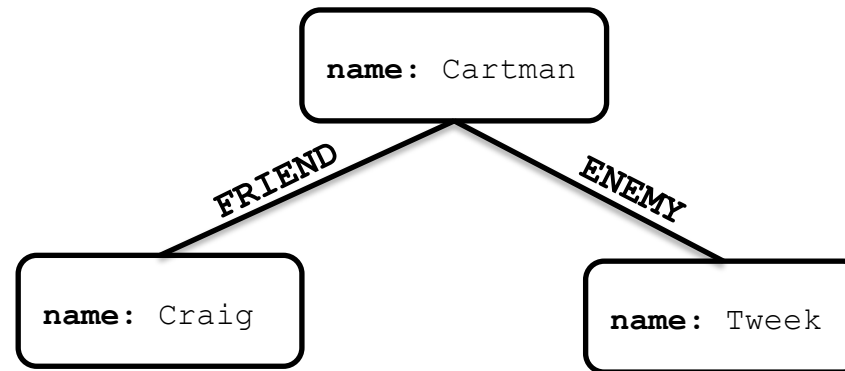


Triadic Closure

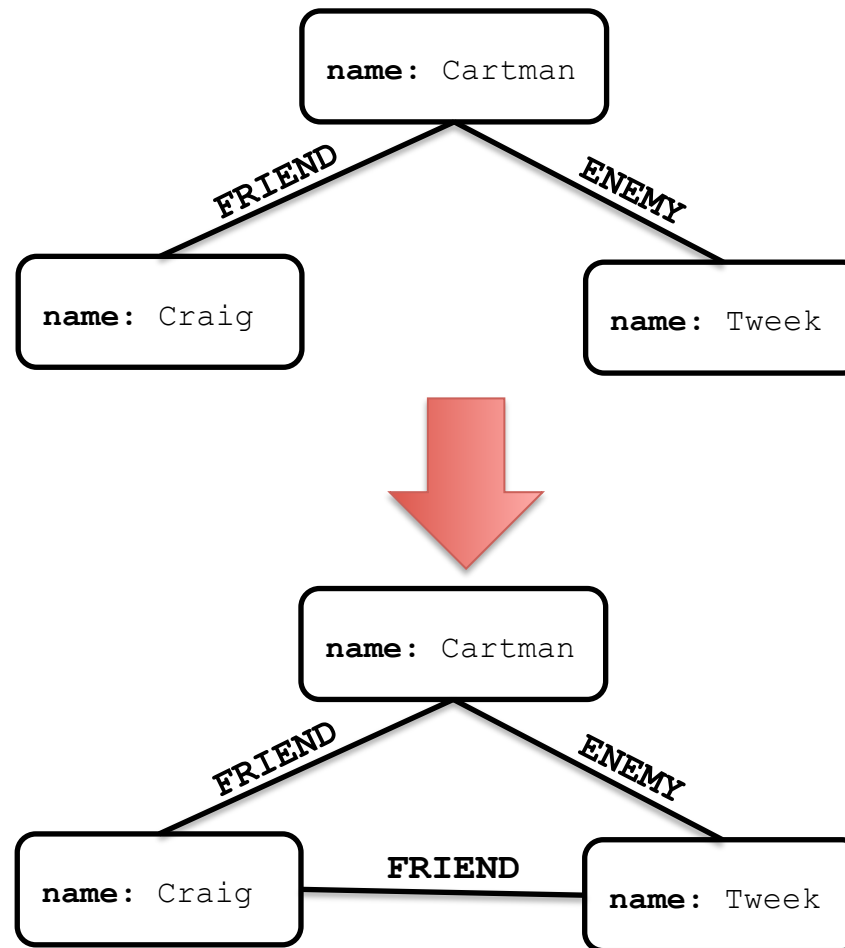
(weak relationship)



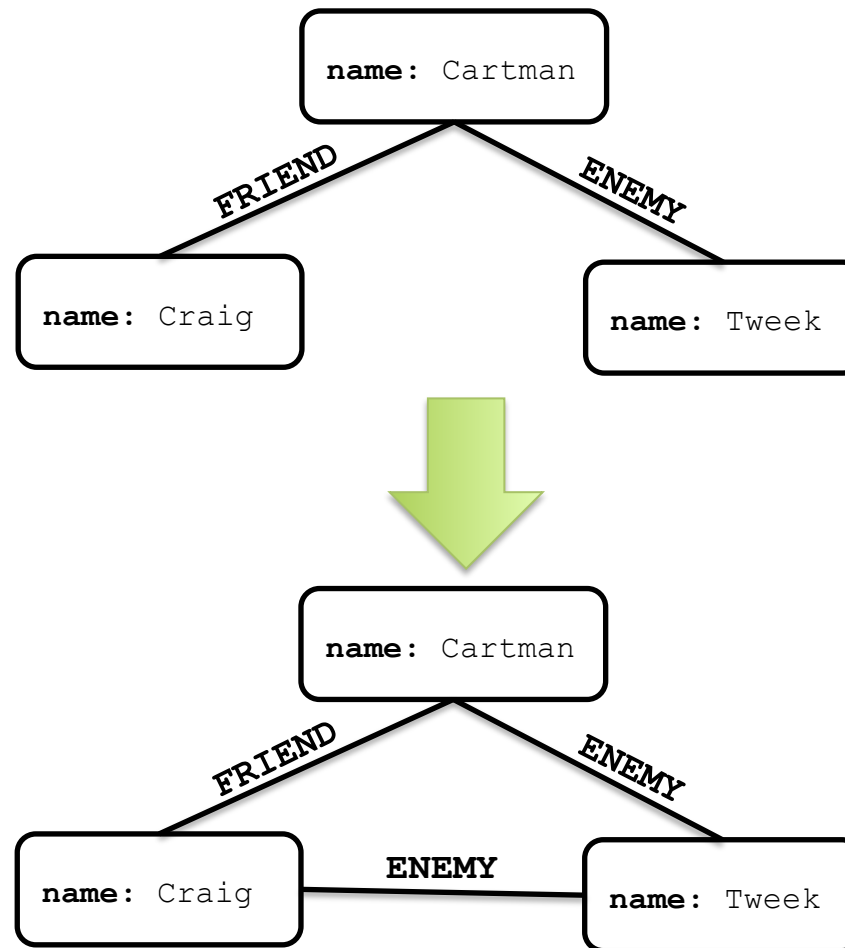
Structural Balance



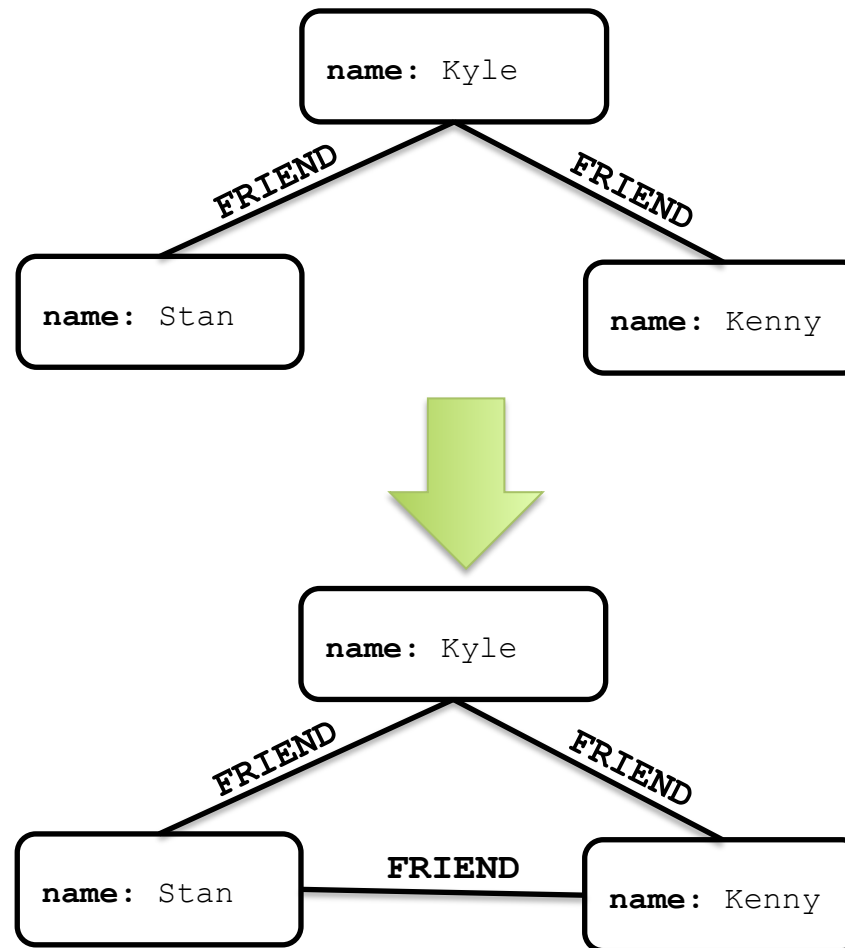
Structural Balance



Structural Balance

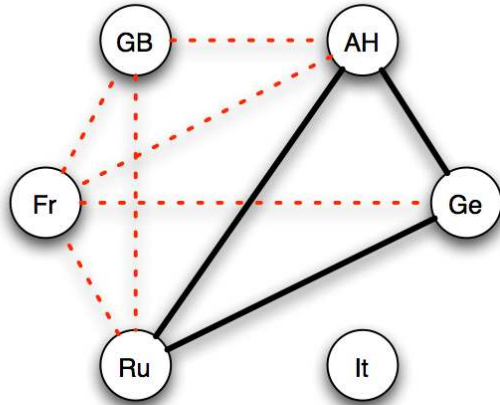


Structural Balance

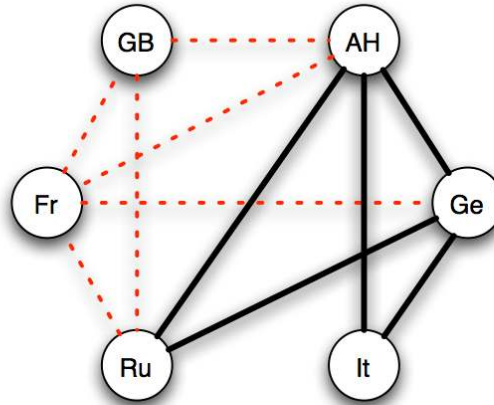


WWI caused by balanced closures?

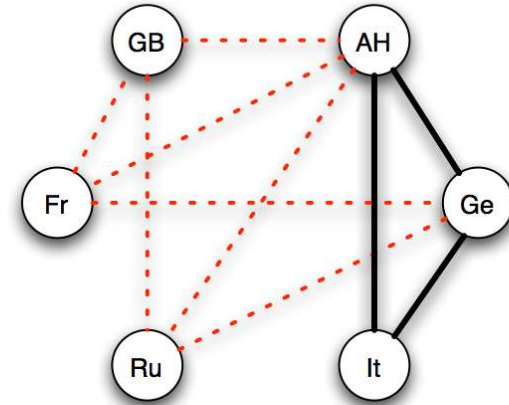
[Easley and Kleinberg]



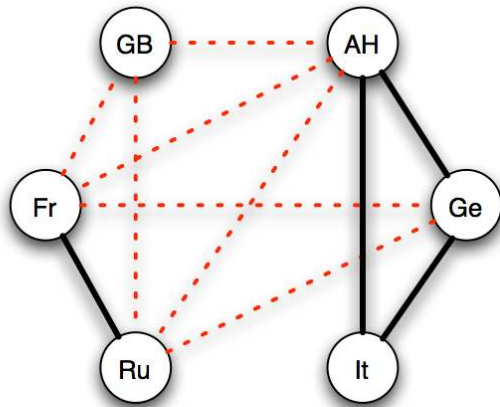
(a) *Three Emperors' League 1872–81*



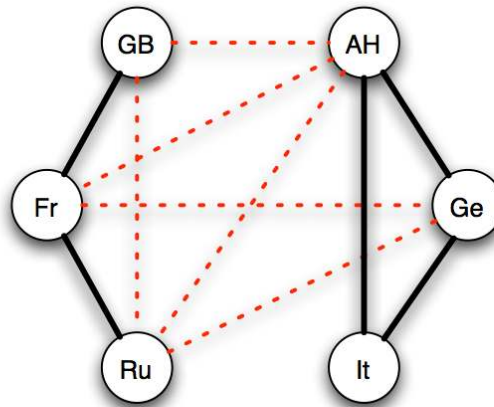
(b) *Triple Alliance 1882*



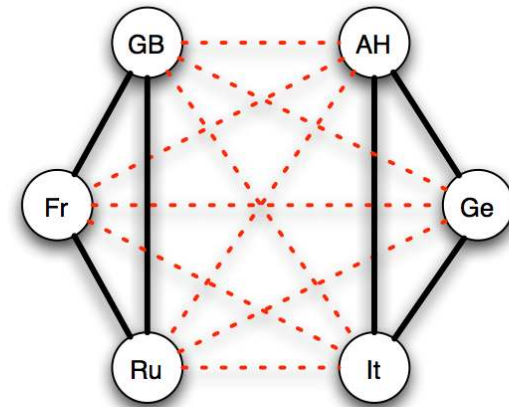
(c) *German-Russian Lapse 1890*



(d) *French-Russian Alliance 1891–94*

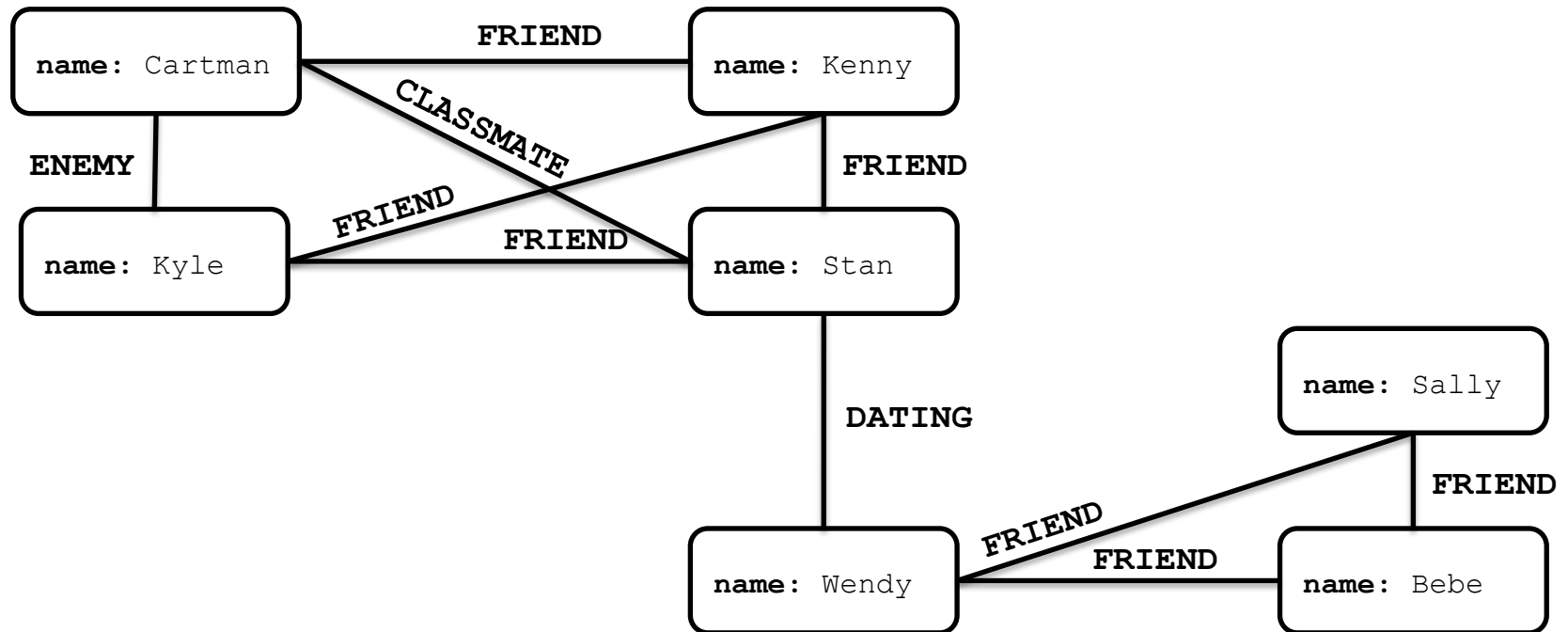


(e) *Entente Cordiale 1904*



(f) *British Russian Alliance 1907*

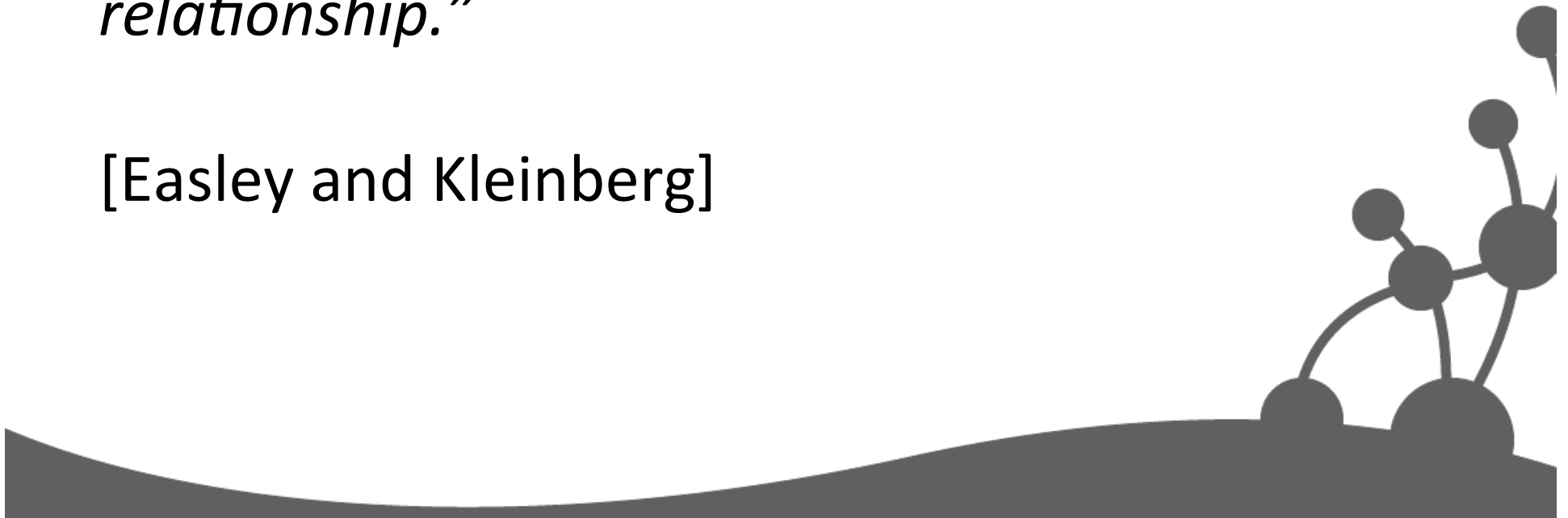
Local Bridges



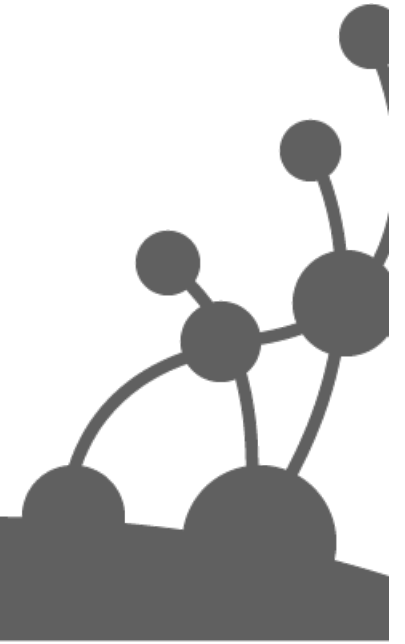
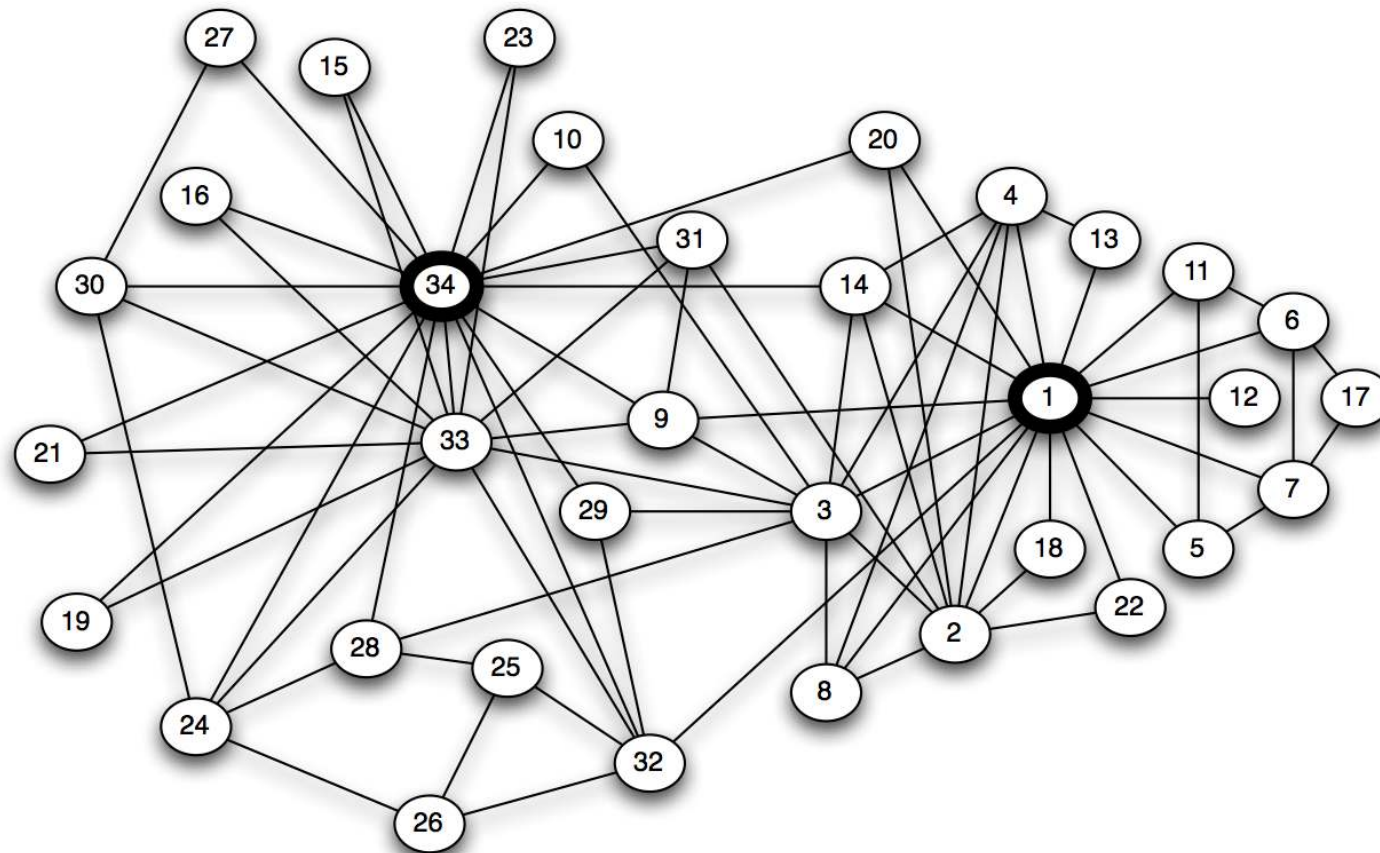
Local Bridge Property

*“If a node **A** in a network satisfies the Strong Triadic Closure Property and is involved in at least two strong relationships, then any local bridge it is involved in must be a weak relationship.”*

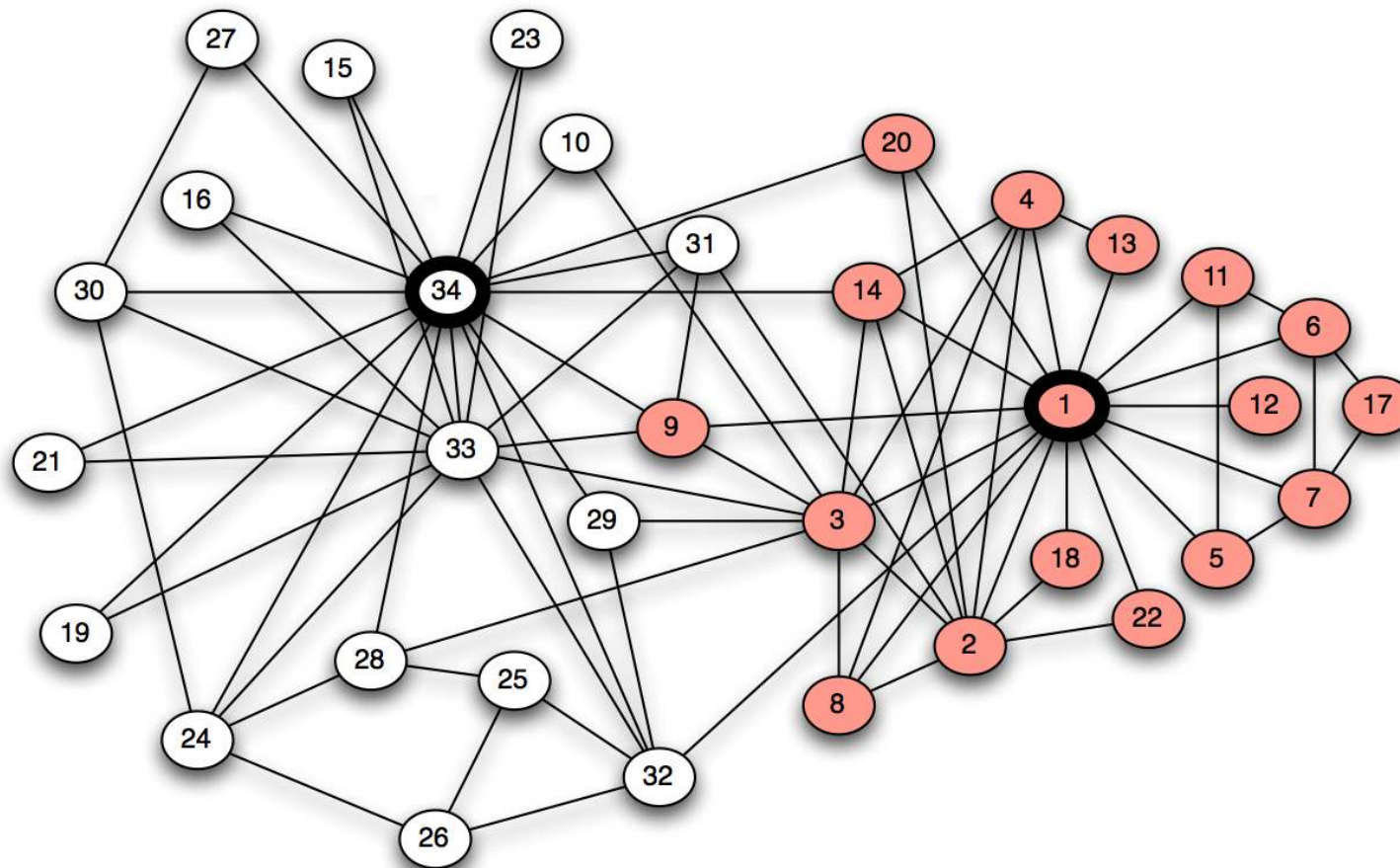
[Easley and Kleinberg]



University Karate Club

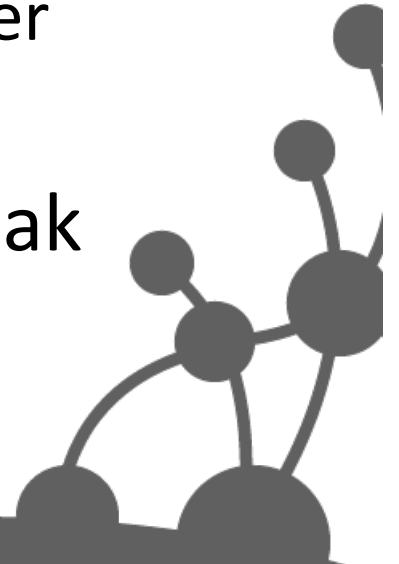


University Karate Clubs



Graph Partitioning

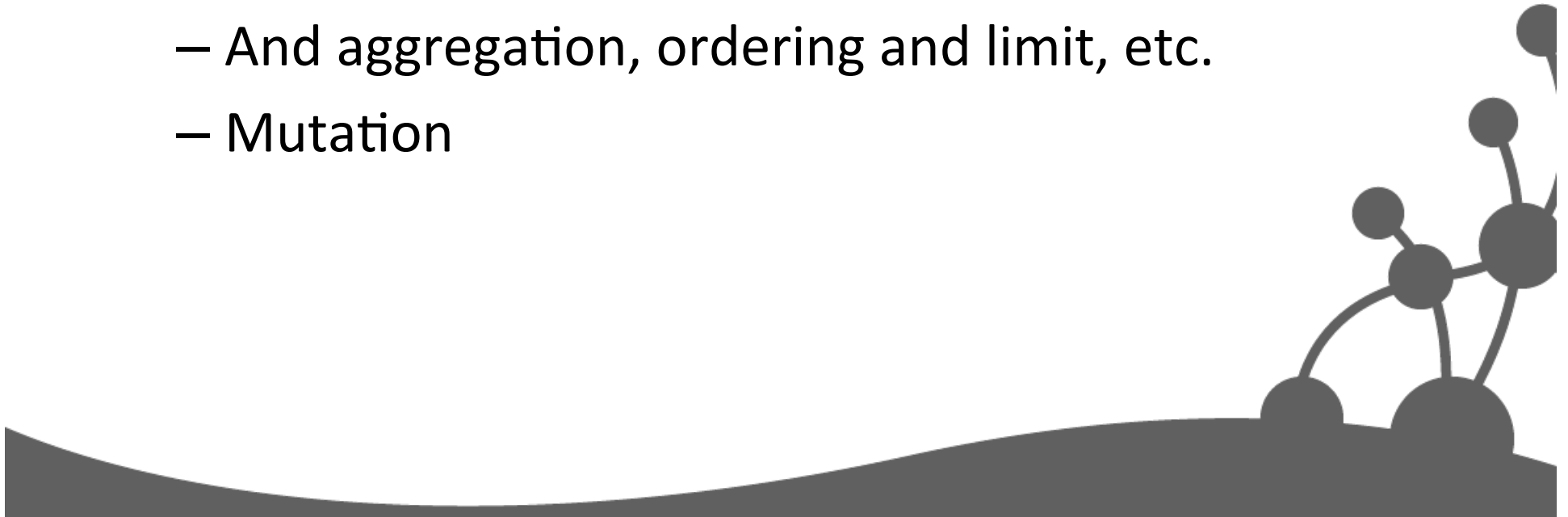
- (NP) Hard problem, so not great for massive graphs
 - Recursively remove the spanning links between dense regions
 - Or recursively merge nodes into ever larger “subgraph” nodes
- Can use this to (almost) predict the break up of the karate club!





Cypher

- Declarative graph pattern matching language
 - “SQL for graphs”
 - Columnar results
- Supports graph matching queries
 - And aggregation, ordering and limit, etc.
 - Mutation

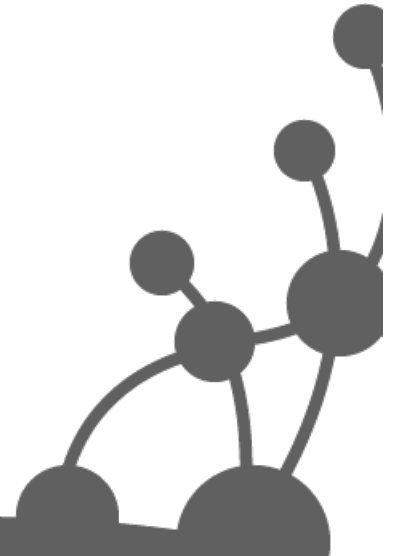
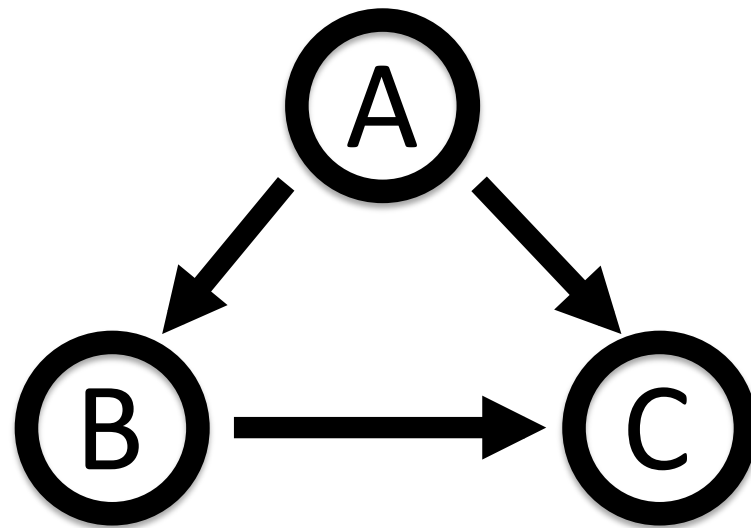


Cypher is Declarative

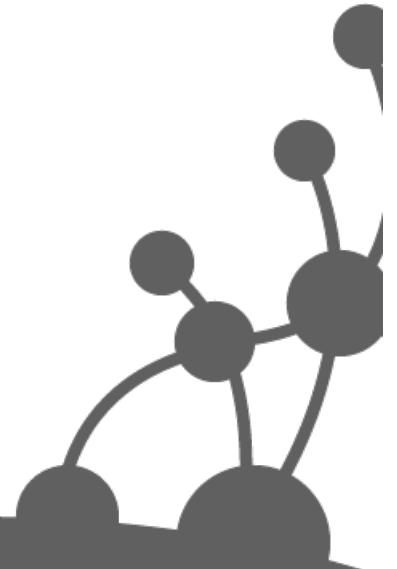
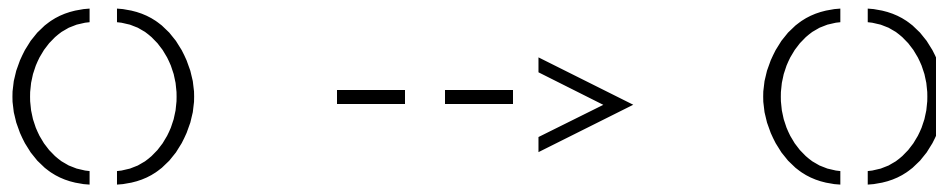
- Imperative
 - follow relationship
 - breadth-first vs depth-first
 - explicit algorithm
- Declarative
 - specify starting point
 - specify desired outcome
 - algorithm adaptable
 - based on query



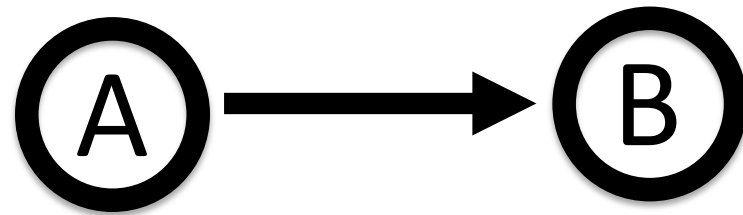
Cypher is a pattern matching language



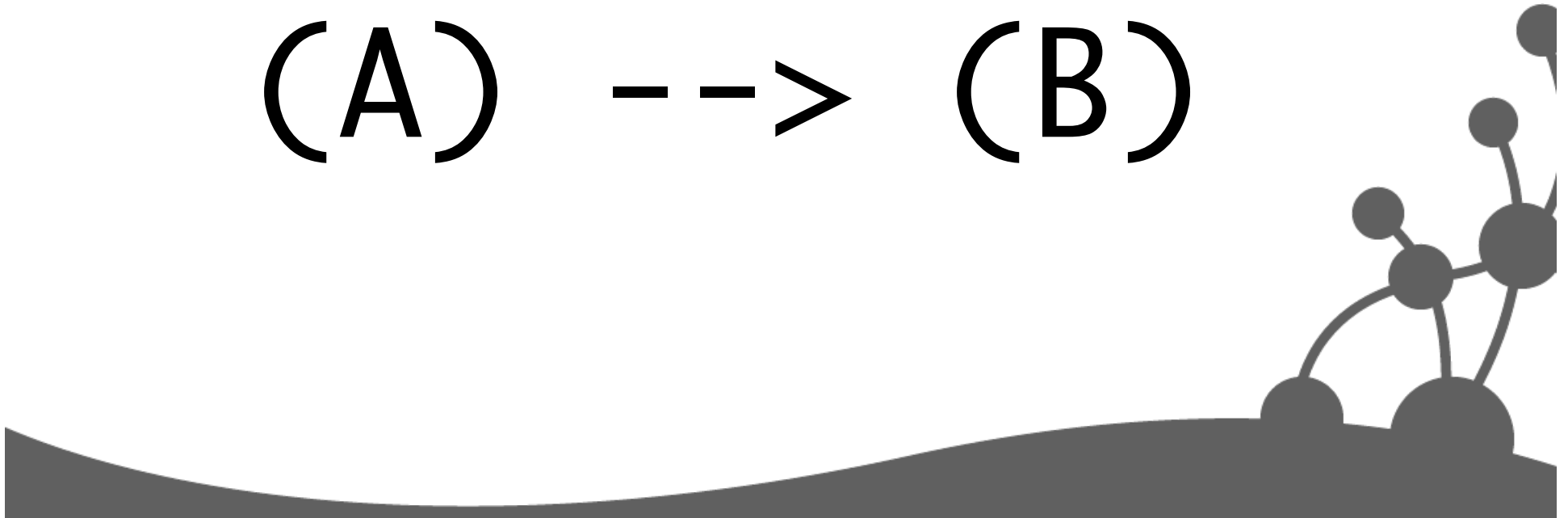
Un-named Nodes & Rels



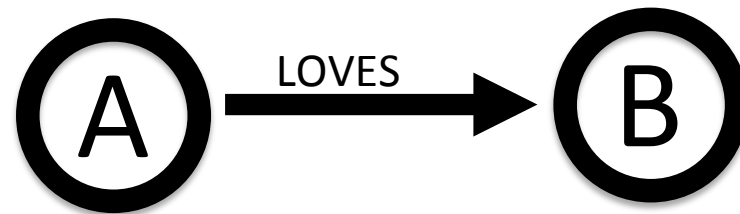
Un-named Relationship



$(A) \dashrightarrow (B)$



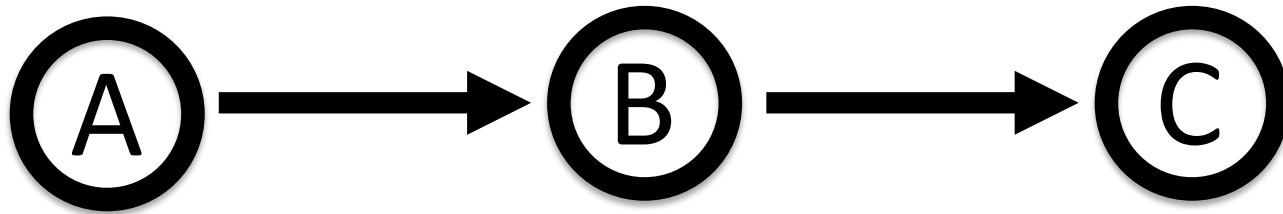
ASCII Art Patterns



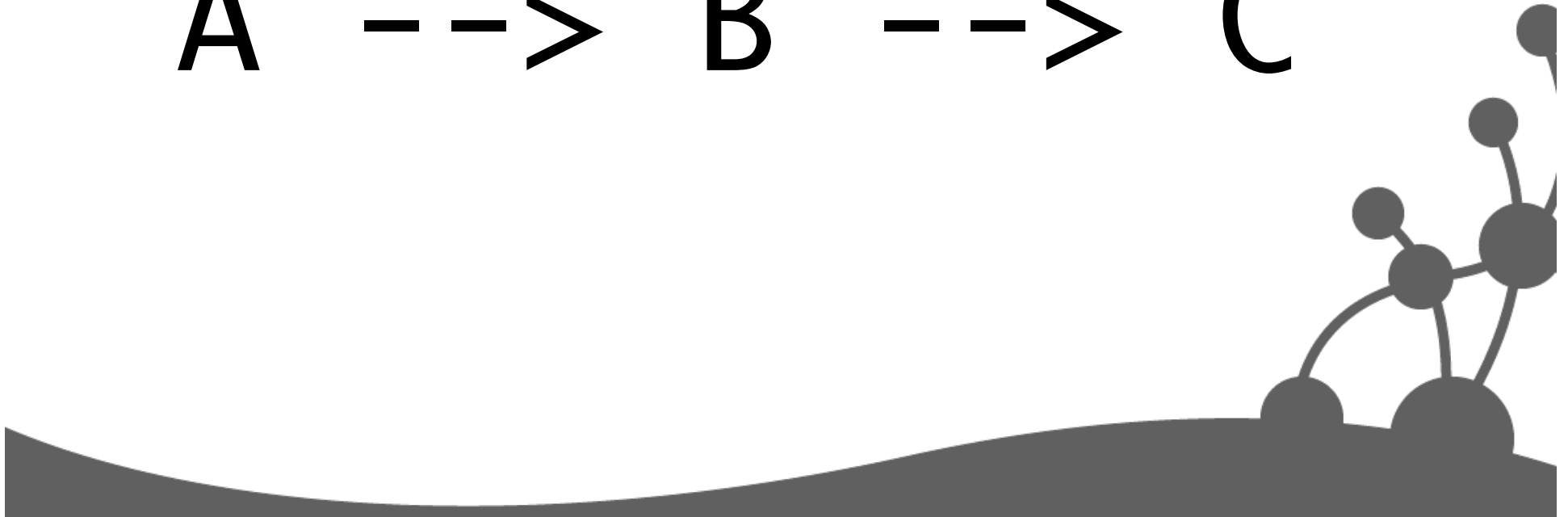
A -[:LOVES]-> B



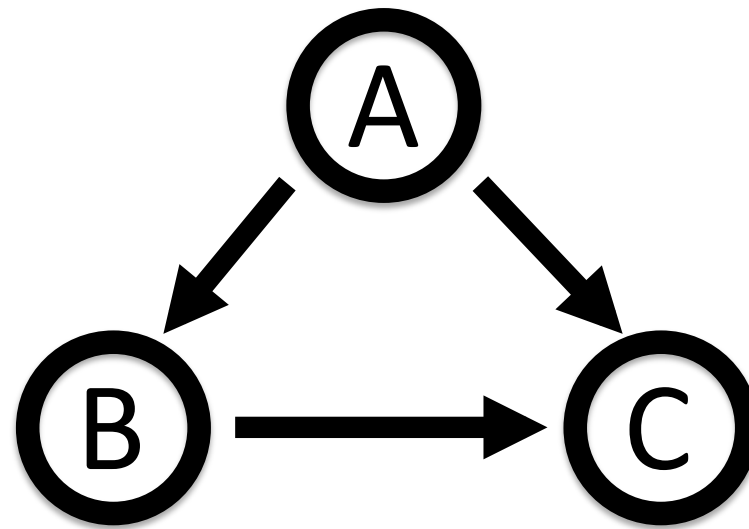
ASCII Art Patterns



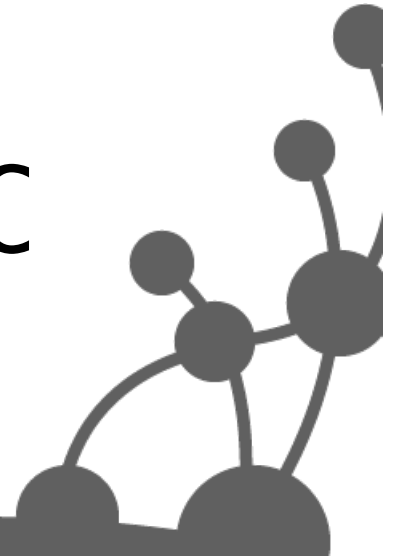
A --> B --> C



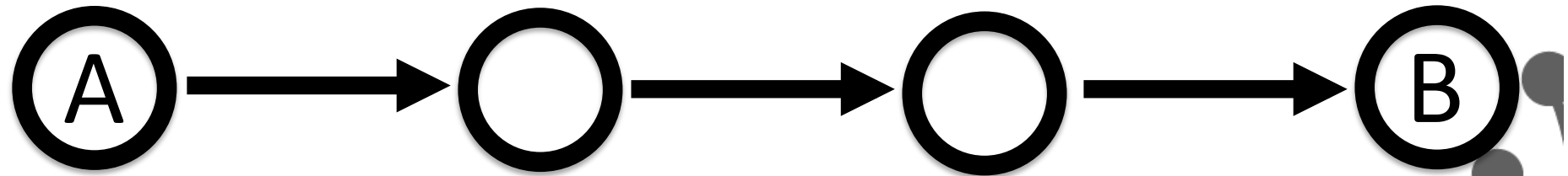
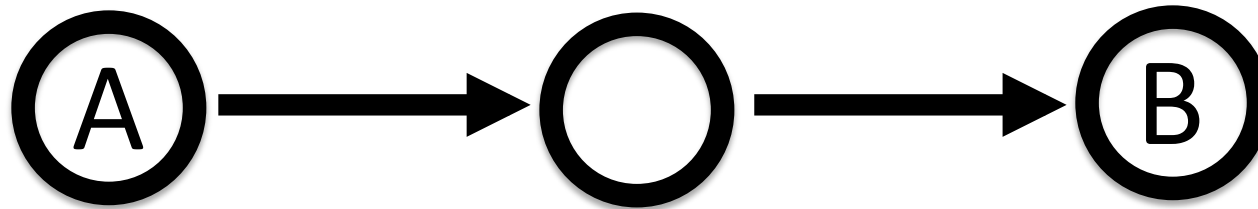
ASCII Art Patterns



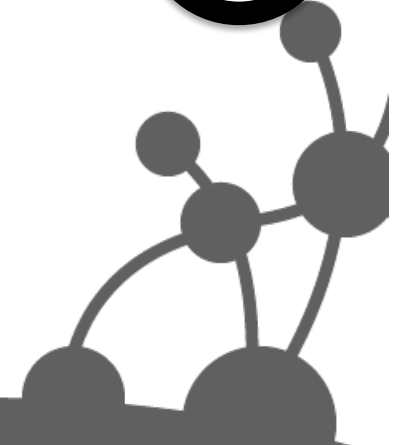
A --> B --> C, A --> C
A --> B --> C <-- A



Variable Length Paths



$A - [*] -> B$

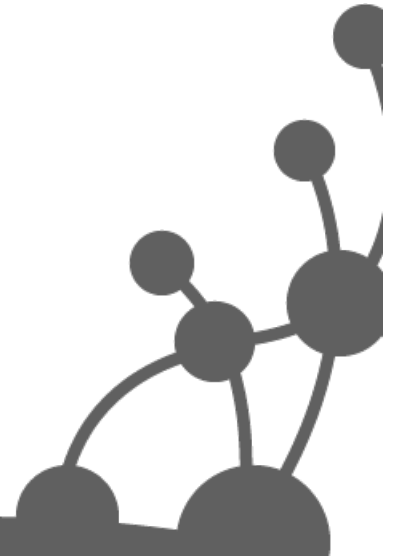


Design Decisions

- Optional relationships



A - [?] -> B



SQL Familiarity

- select
 - from
 - where
 - group by
 - order by
- start
 - match
 - where
 - return



Using Indexes

Index name

Key = 'value'

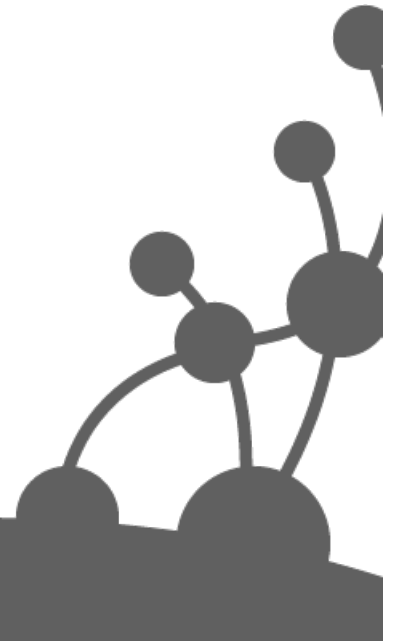
- **Lookup:**

```
start daleks = node:species(species='Dalek')  
return daleks
```

- **Query:**

```
start monsters = node:species('species:S*n')  
return monsters
```

Lucene query
string



Match (ASCII art for graphs)

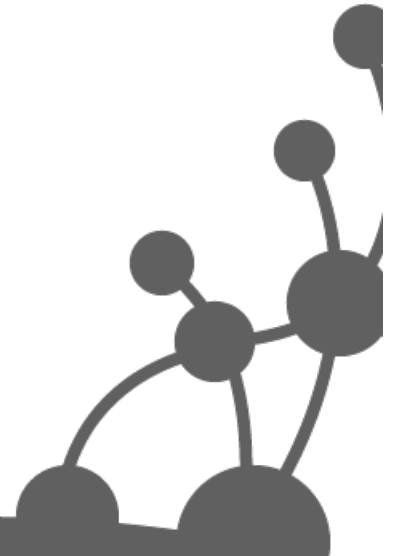
Bound node

Bound node

```
start daleks = node:species(species='Dalek')  
match daleks-[:APPEARED_IN]->episode  
return episode
```

Relationship
name

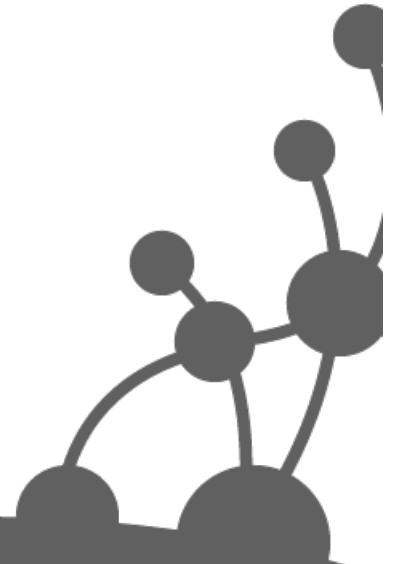
Direction



'Circles and arrows'

Optional
brackets

```
start daleks = node:species(species='Dalek')  
match (daleks) - [:APPEARED_IN] -> (episode)  
return episode
```



Directionality

No direction
arrow

```
start daleks = node:species(species='Dalek')  
match (daleks)-[:APPEARED_IN]-(episode)  
return episode
```



Example Query

- The top 5 most frequently appearing companions:

```
start doctor=node:characters(character = 'Doctor')
match (doctor)<-[:COMPANION_OF]-(companion)
      -[:APPEARED_IN]->(episode)
return companion.character, count(episode)
order by count(episode) desc
limit 5
```

Start node from
index

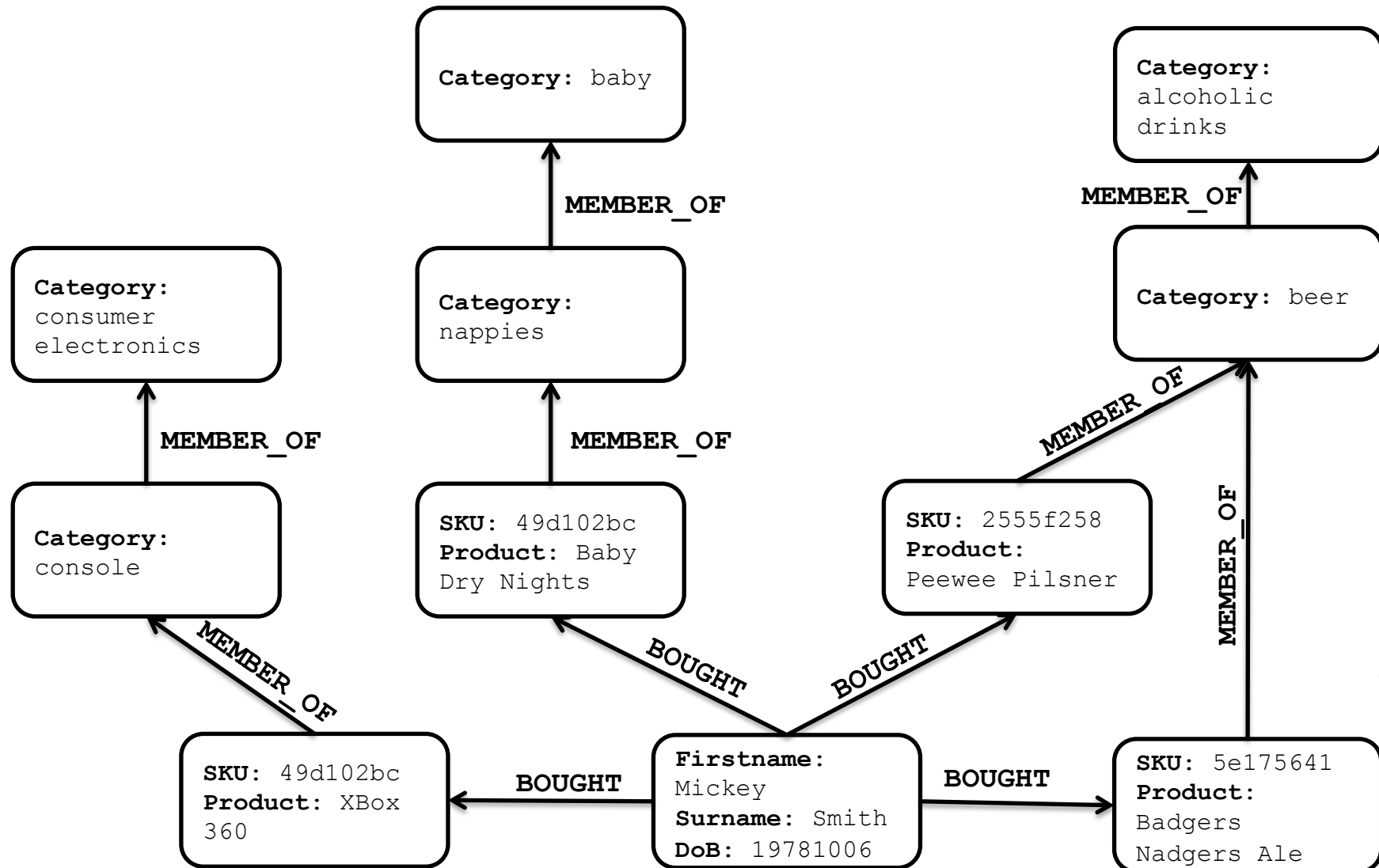
Subgraph
pattern

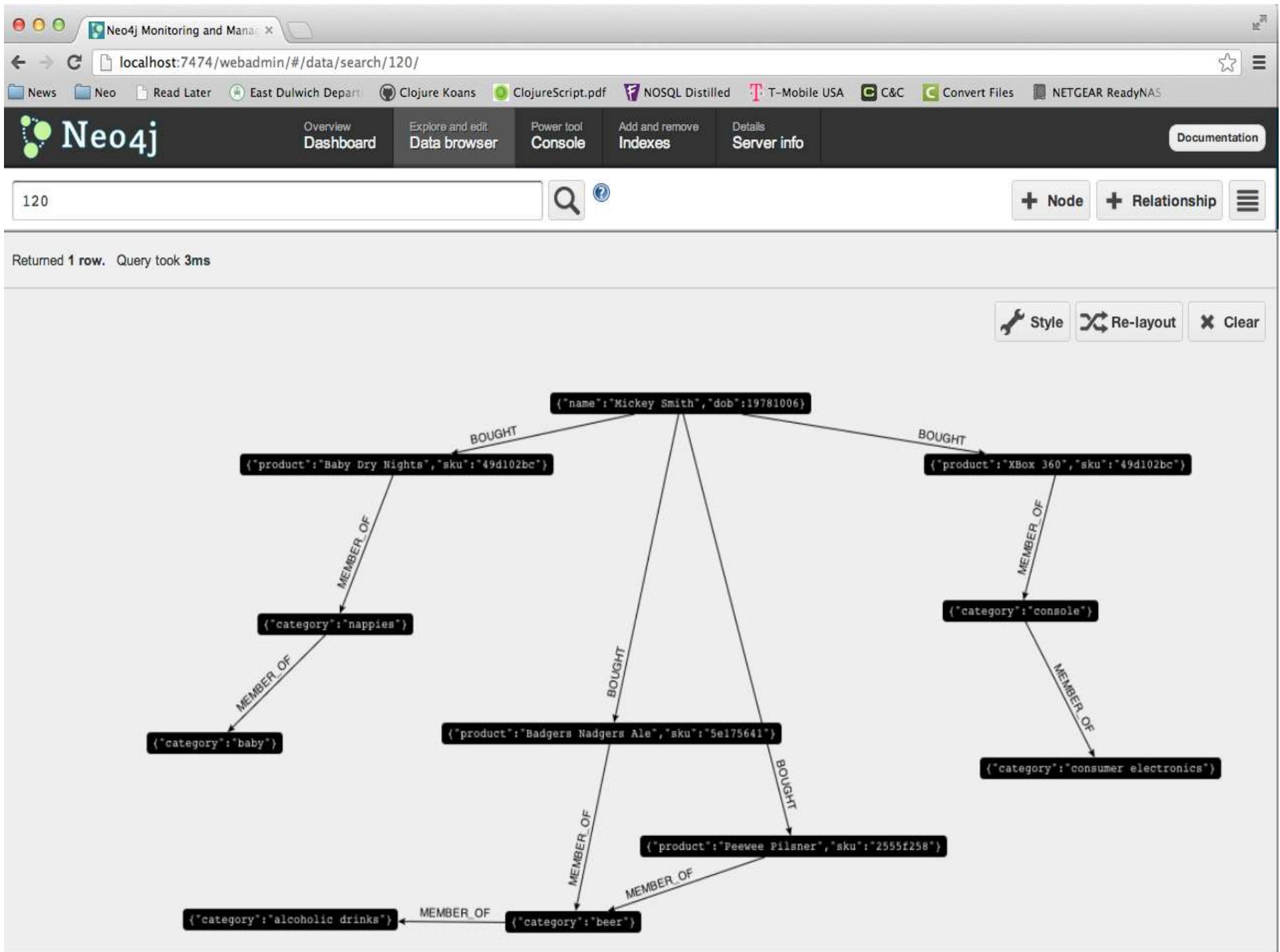
Accumulates
rows by episode

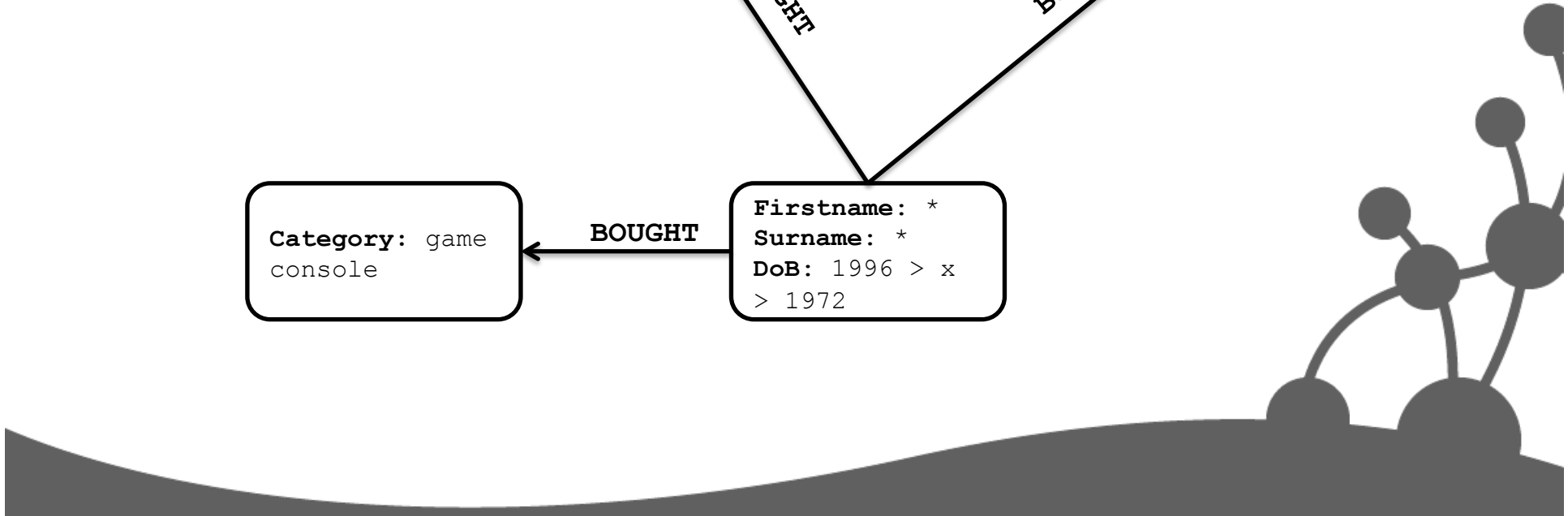
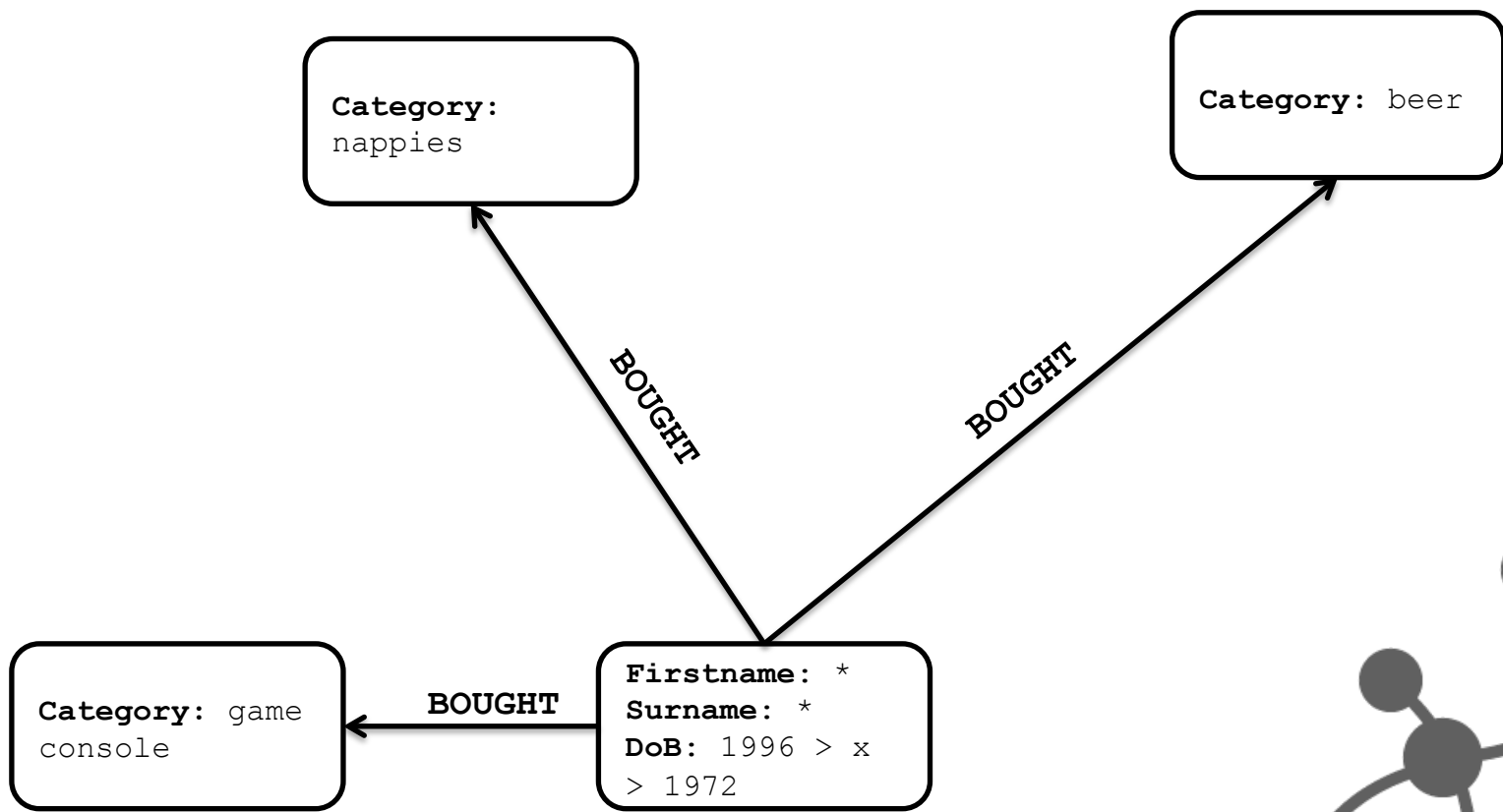
Limit returned
rows

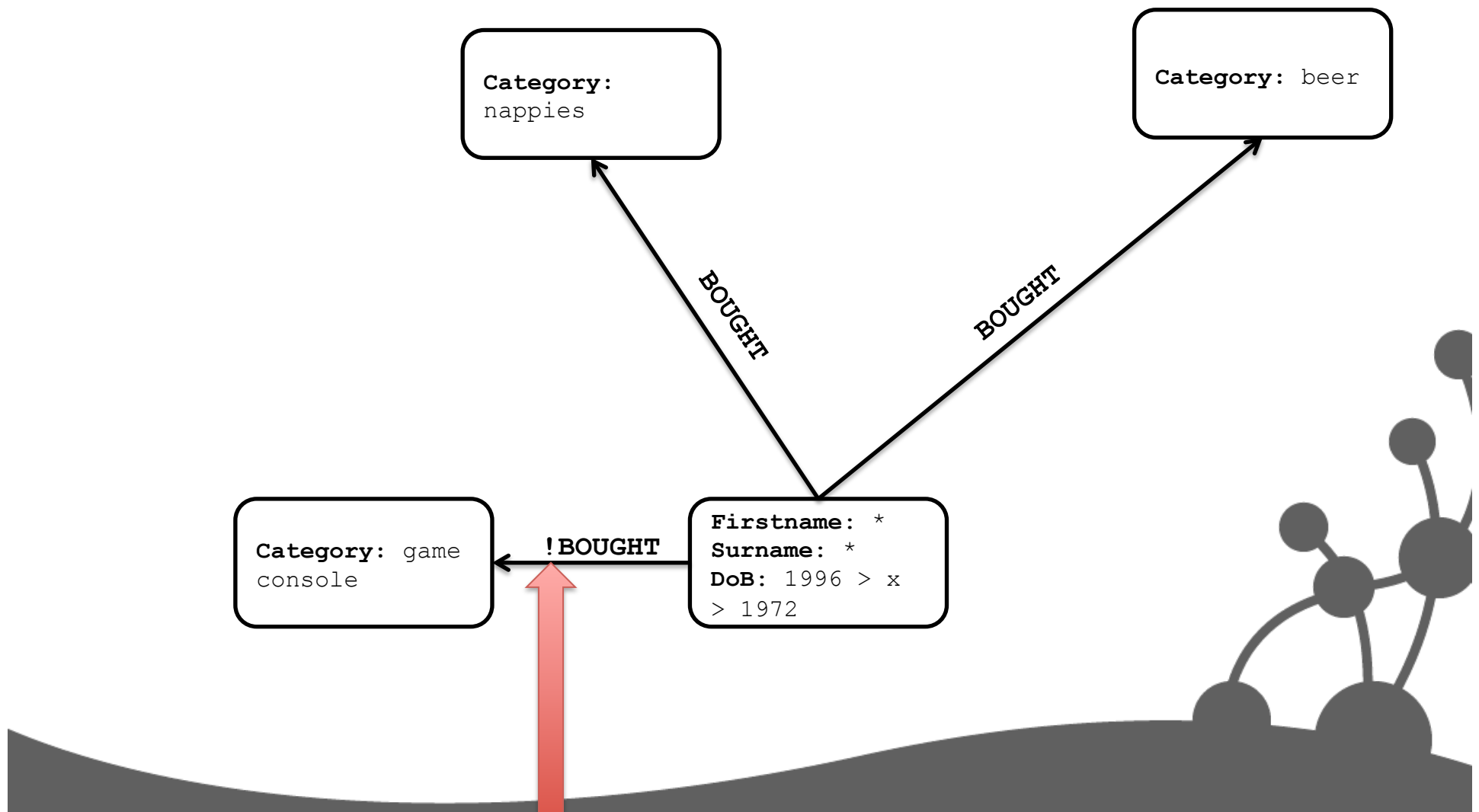


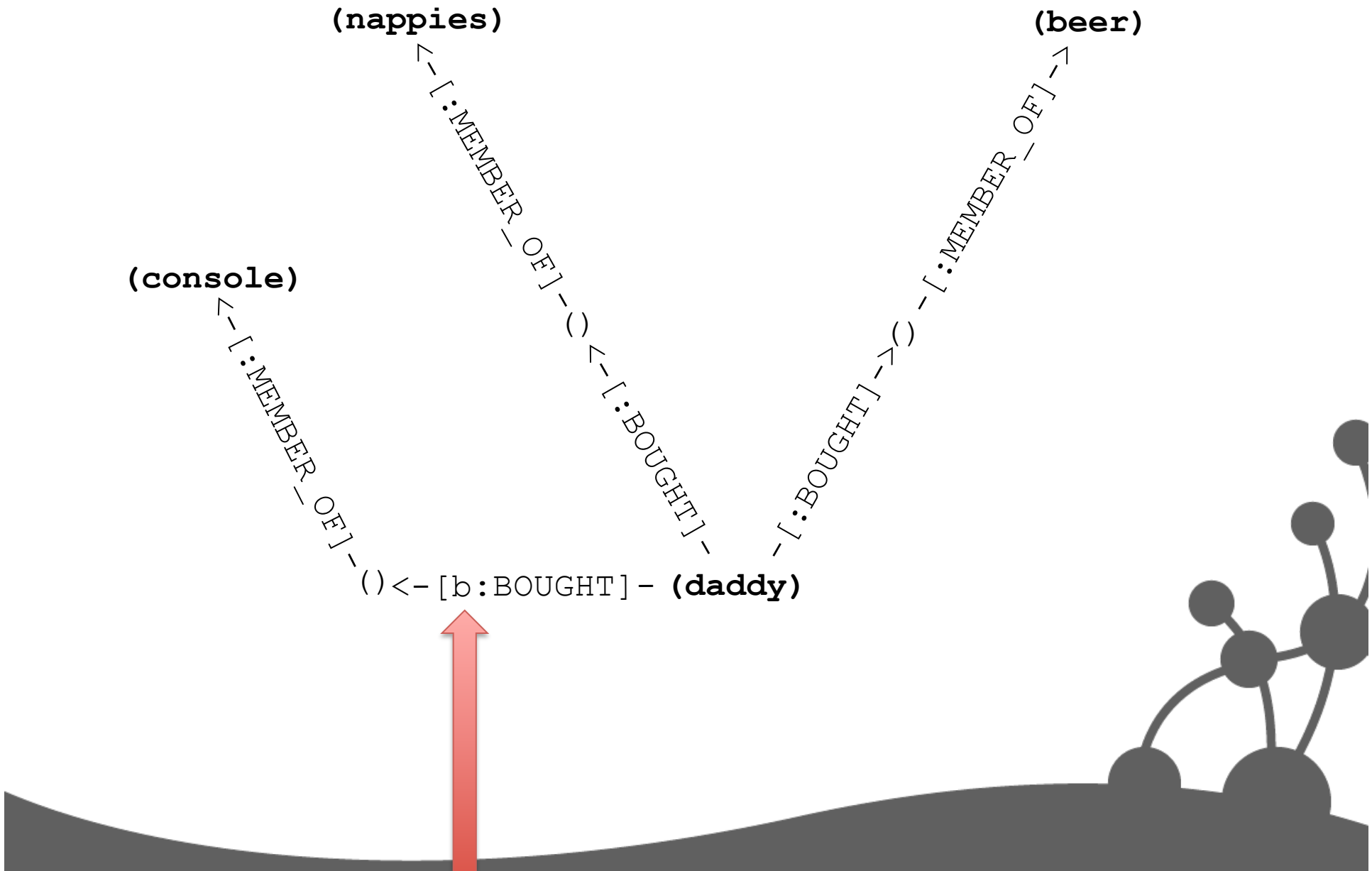












Flatten the graph

```
(daddy) - [:BOUGHT] -> () - [:MEMBER_OF] -> (nappies)  
(daddy) - [:BOUGHT] -> () - [:MEMBER_OF] -> (beer)  
(daddy) - [b:BOUGHT] -> () - [:MEMBER_OF] -> (console)
```



Wrap in a Cypher MATCH clause

```
MATCH (daddy) -[:BOUGHT]->()-[:MEMBER_OF]->(nappies),  
(daddy) -[:BOUGHT]->()-[:MEMBER_OF]->(beer),  
(daddy) -[b:BOUGHT]->()-[:MEMBER_OF]->(console)
```



Cypher WHERE clause

```
MATCH (daddy)-[:BOUGHT]->()-[:MEMBER_OF]->(nappies),  
      (daddy)-[:BOUGHT]->()-[:MEMBER_OF]->(beer),  
      (daddy)-[b:BOUGHT]->()-[:MEMBER_OF]->(console)  
WHERE b is null
```



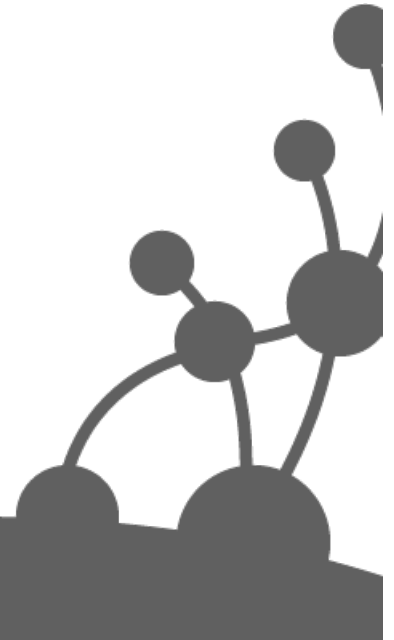
Full Cypher query

```
START beer=node:categories(category='beer'),  
      nappies=de:categories(category='nappies'),  
      xbox=node:products(product='xbox 360')
```

```
MATCH (daddy)-[:BOUGHT]->()-[:MEMBER_OF]->(beer),  
      (daddy)-[:BOUGHT]->()-[:MEMBER_OF]->(nappies),  
      (daddy)-[b?:BOUGHT]->(xbox)
```

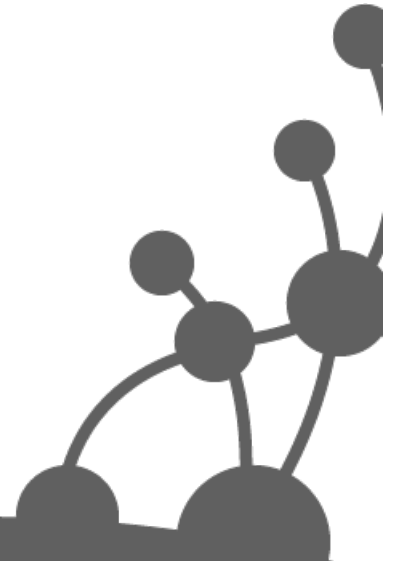
```
WHERE b is null
```

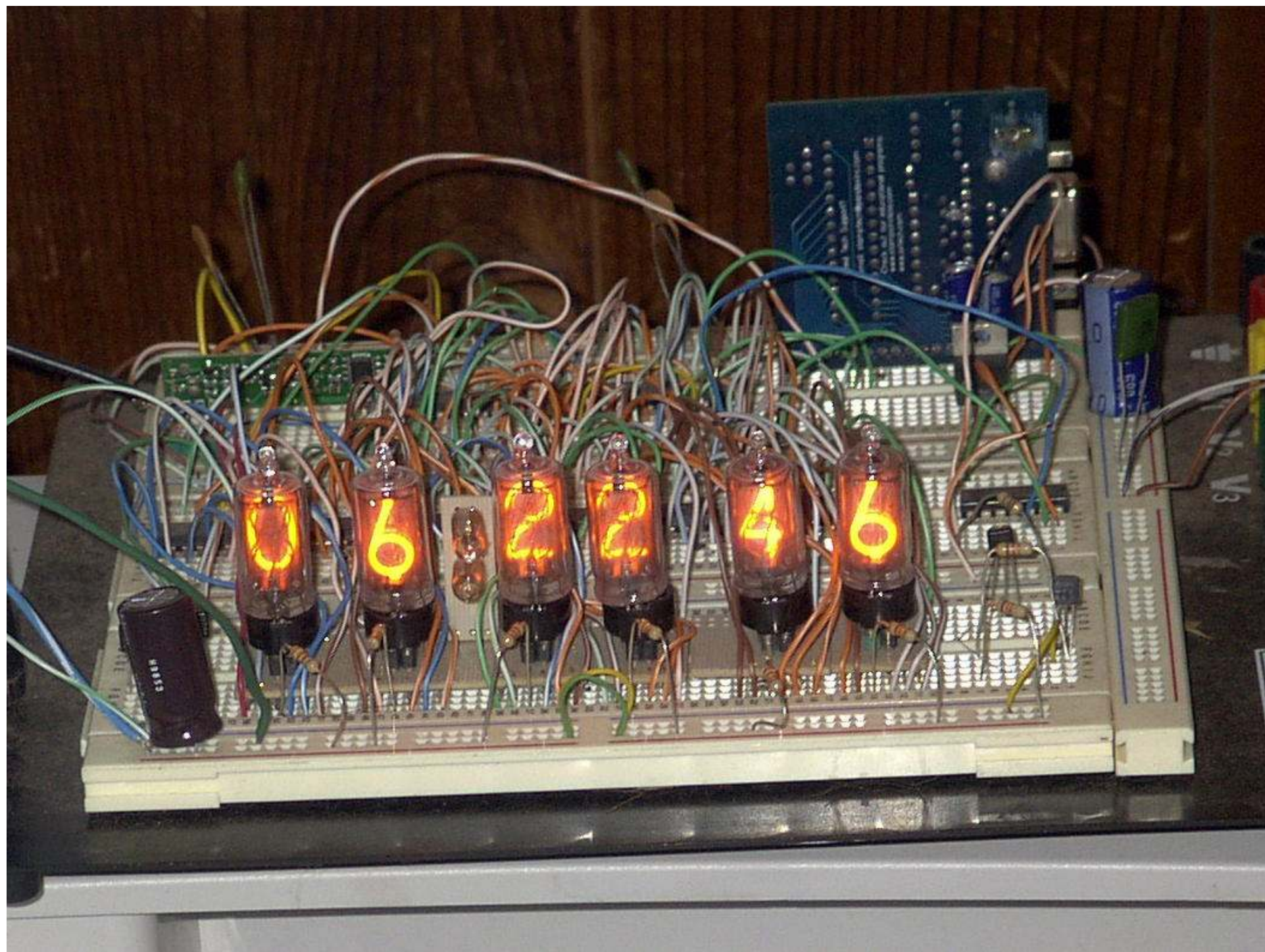
```
RETURN distinct daddy
```



Results

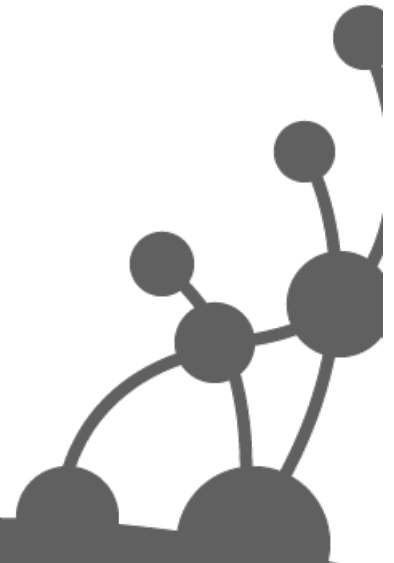
```
==> +-----+
==> | daddy                                     |
==> +-----+
==> | Node[15]{name:"Rory Williams",dob:19880121} |
==> +-----+
==> 1 row
==> 6 ms
==>
neo4j-sh (0)$
```





What are graphs good for?

- Recommendations
- Business intelligence
- Social computing
- Geospatial
- MDM
- Systems management
- Web of things
- Genealogy
- Time series data
- Product catalogue
- Web analytics
- Scientific computing (especially bioinformatics)
- Indexing your *slow* RDBMS
- And much more!





Thanks for listening

Neo4j: <http://neo4j.org>

Neo Technology: <http://neotechnology.com>

Me: @jimwebber

