Neo4 o quê?

A practical guide to Graph Databases

@ederign

/tiagobento

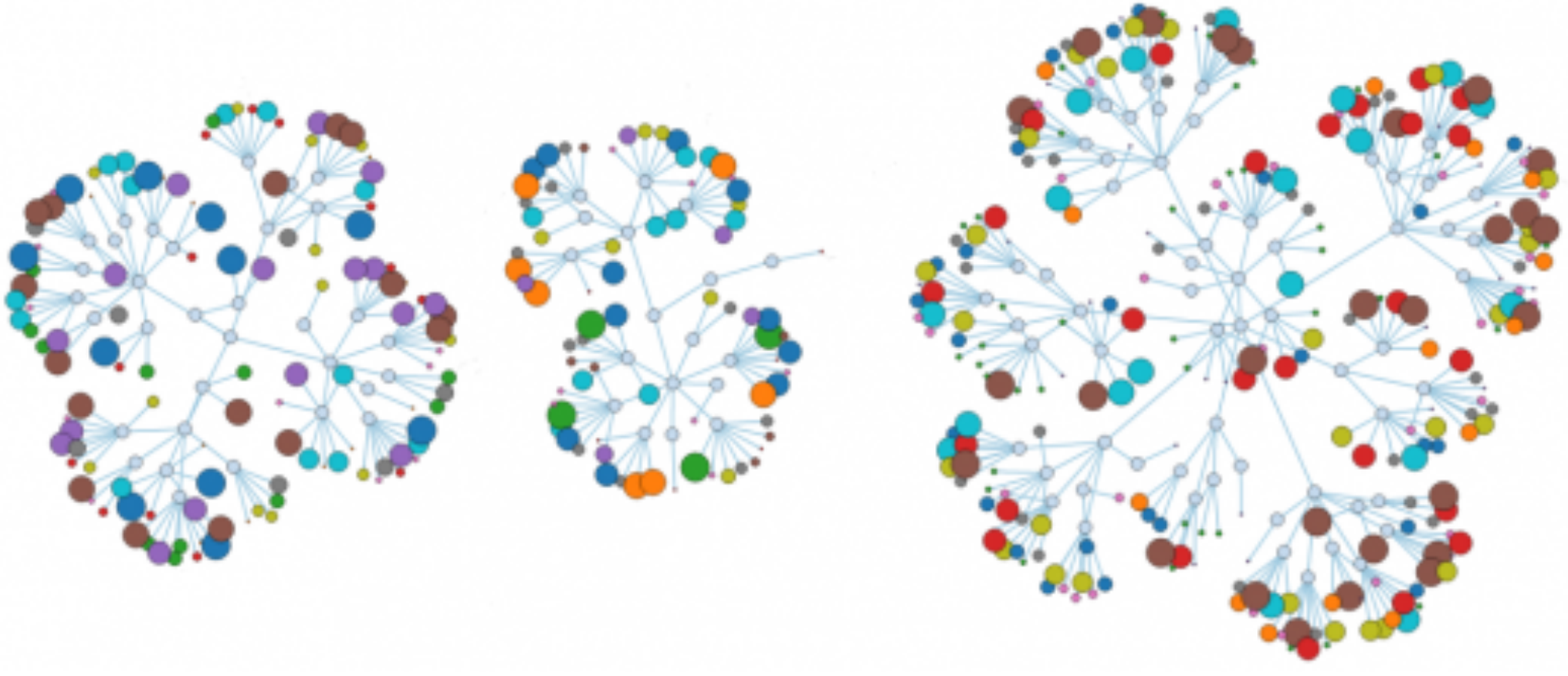
BIG DATA

“Every 2 days we create as much information as we did up to 2003”

Eric Schmidt, Google

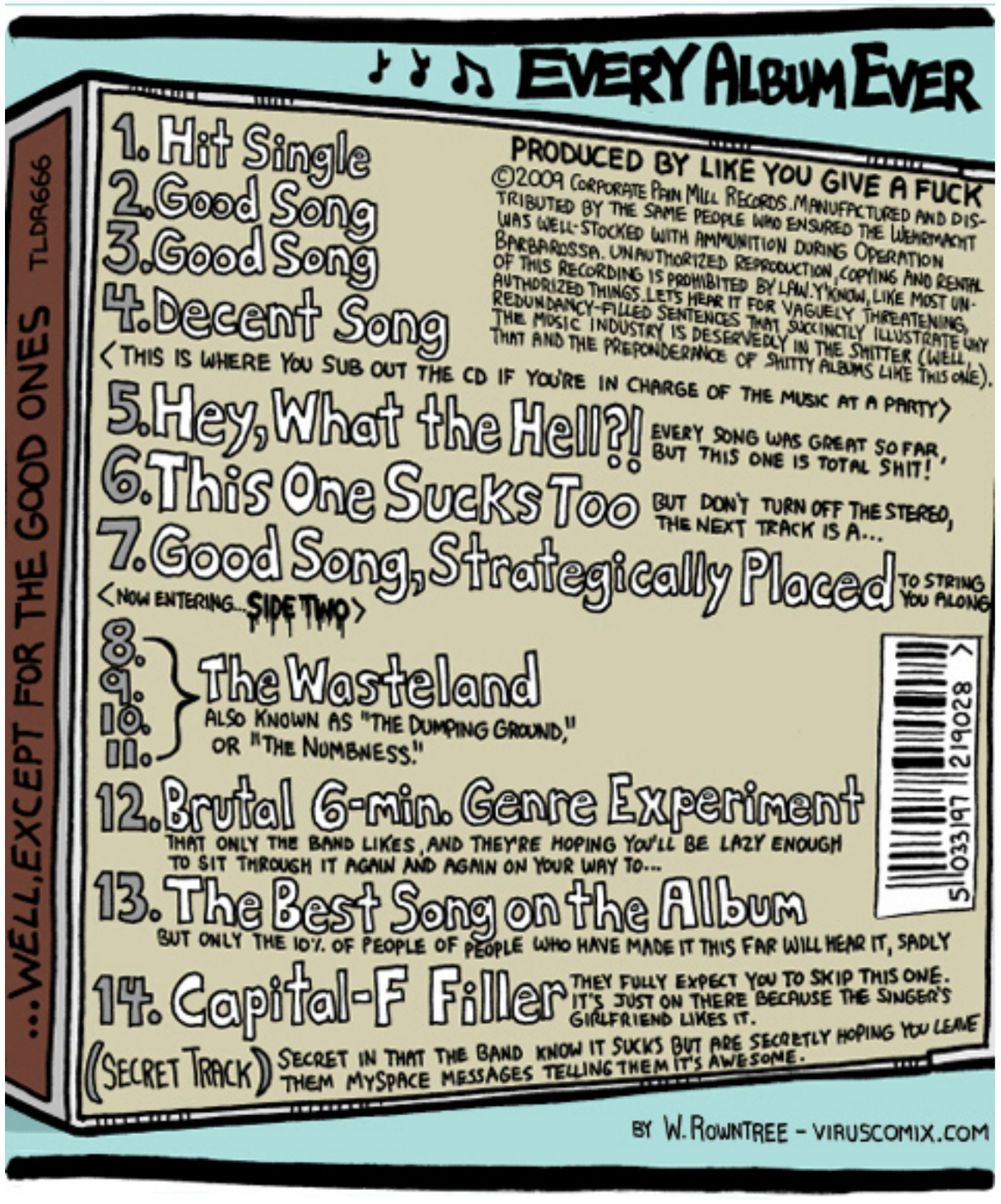
Our data is more **connected**

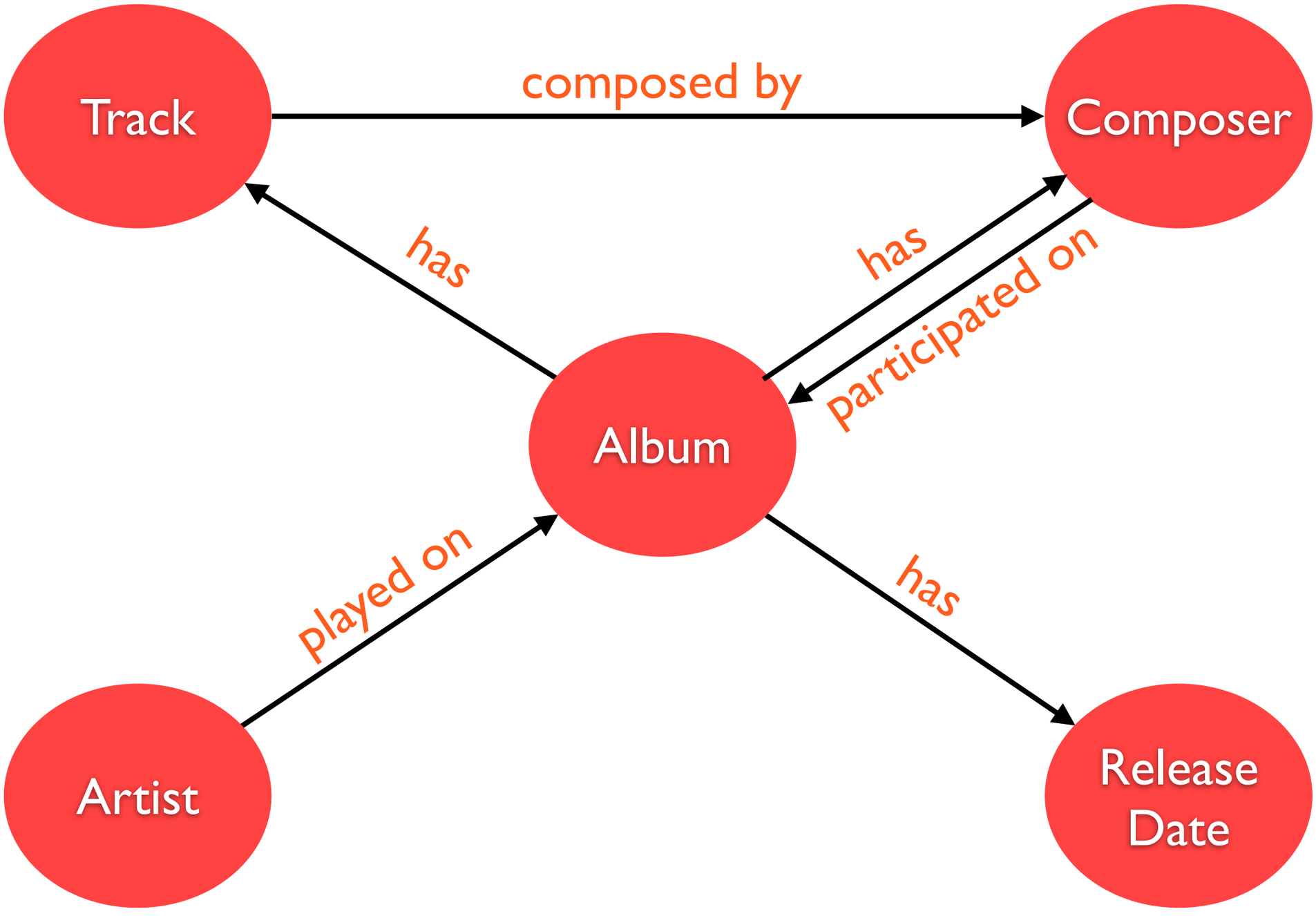
Text (content)
HyperText (added pointers)
RSS (joined those pointers)
Blogs (added pingbacks)

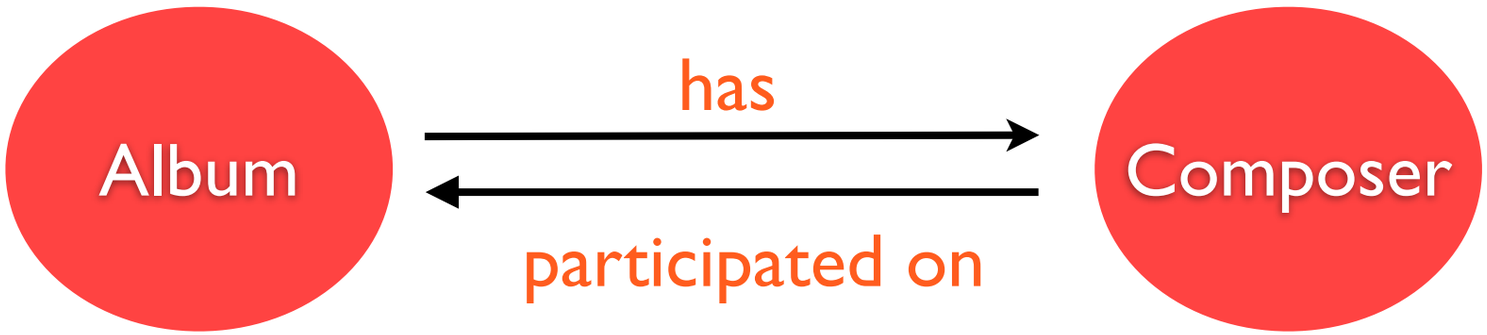


Data is more **complex** and **semi-structured**

If you tried to collect all the data of every music album ever made, how would you model it?




Track
composed by
Composer
has
has
participated on
Album
played on
has
Artist
Release Date

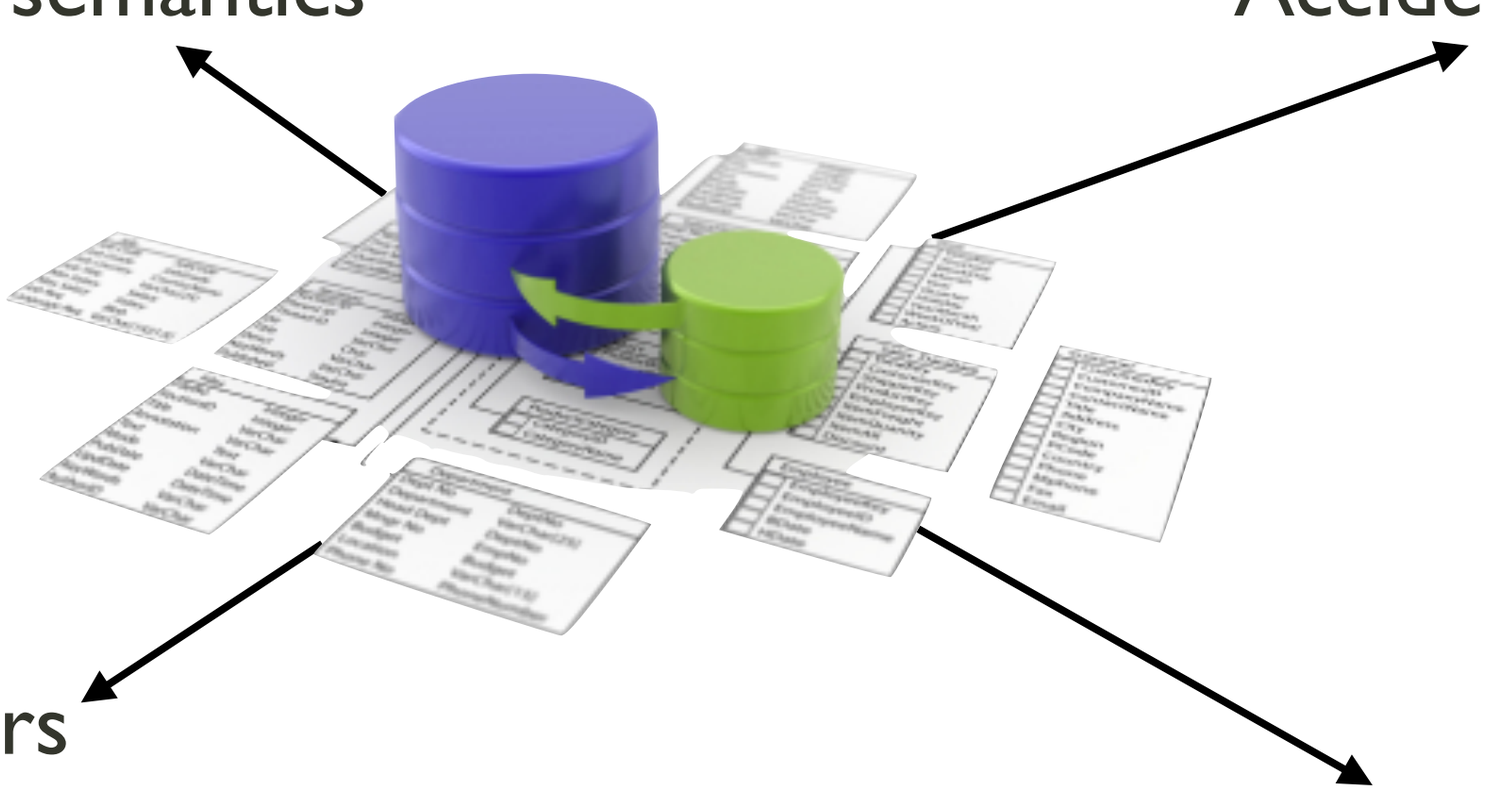

Album
has
participated on
Composer


Album
id
Name
release_date
art
AlbumComposer
album_id
Composer_id
Composer
id
name
birth

You have to **stretch** your data
(and your domain model)
to fit in relational databases

Lose semantics

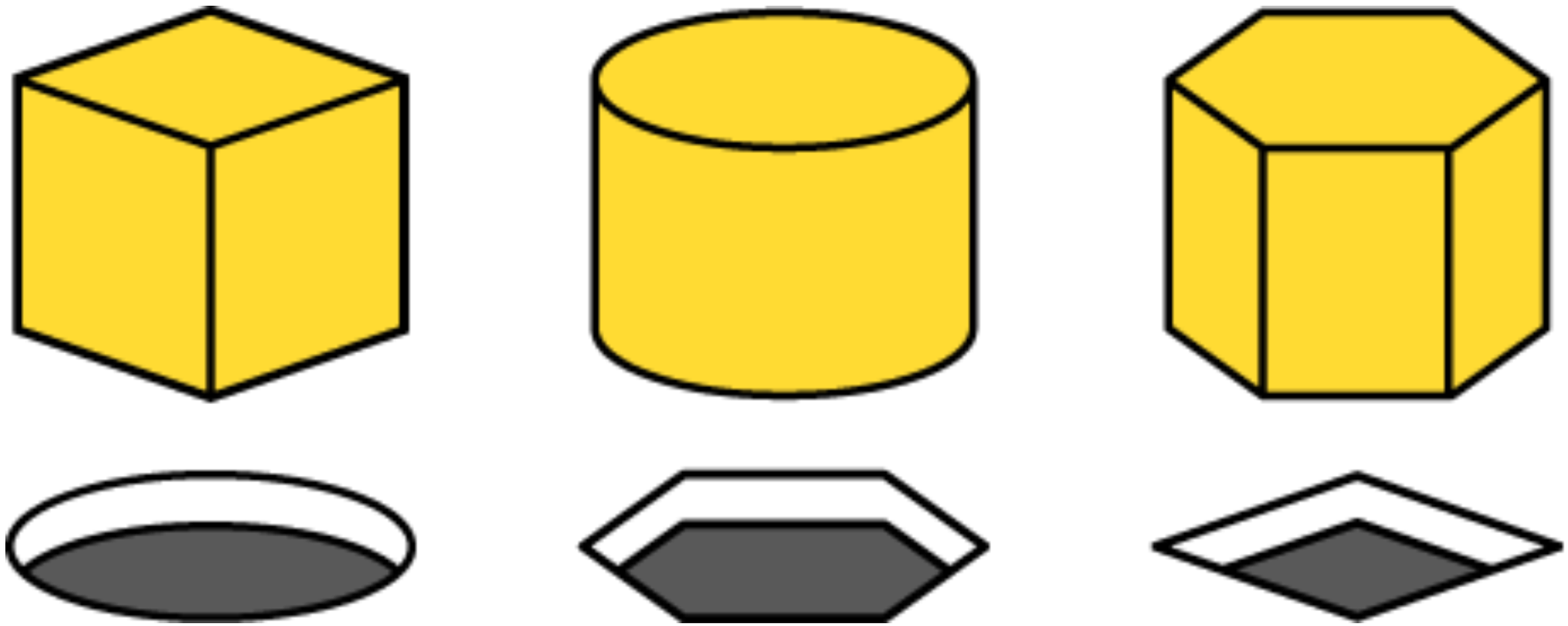
Accidental complexity



Tears

Object-relational impedance mismatch

There is **no**
"one size fits all"
approach

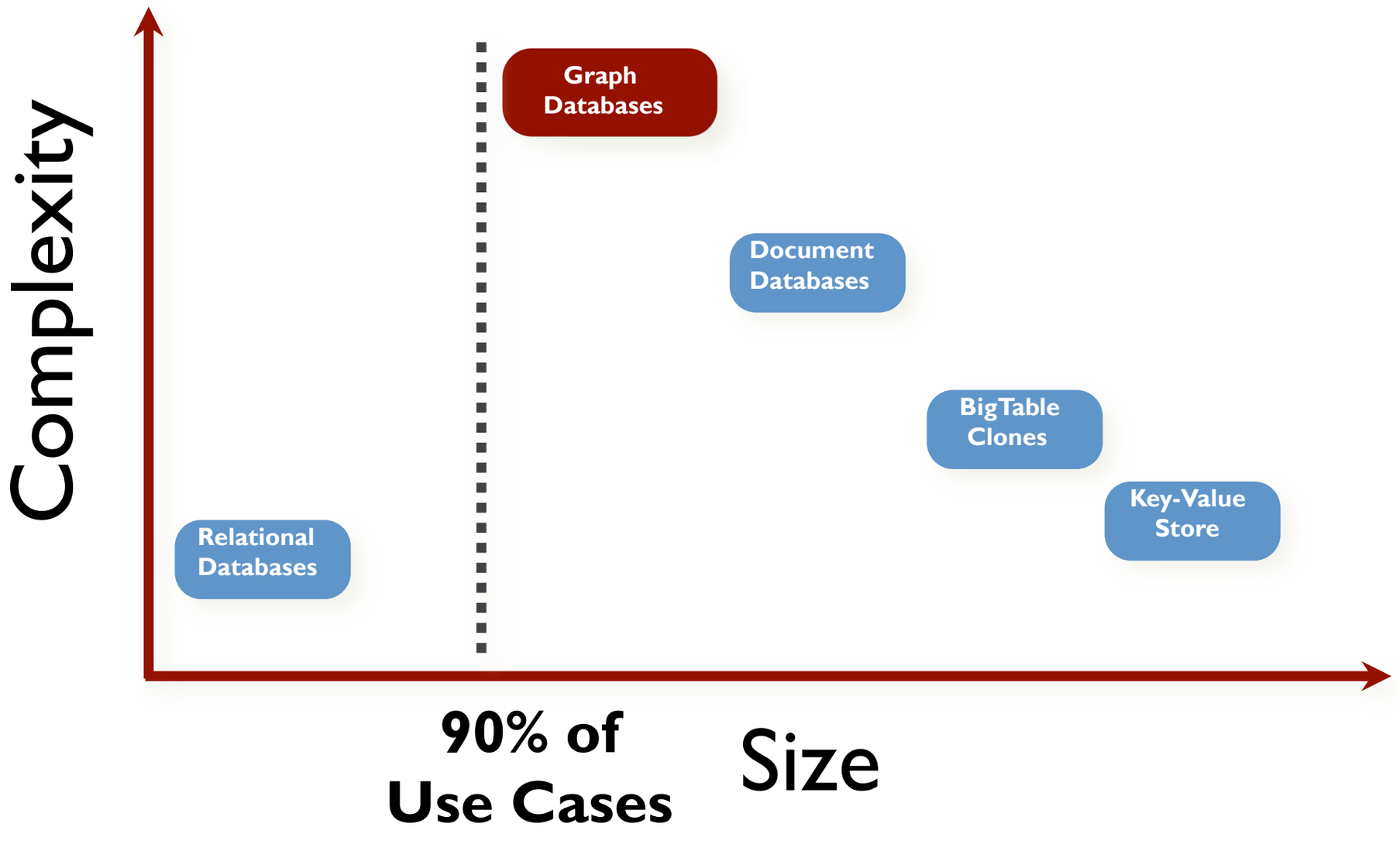


Not Only SQL

Key-Value
Column Family
Document
Graph

The right **tool** for the right **job**

Don't be a hipster



Why should I use a graph database?

Graphs are everywhere

Best fits for

Highly connected data (*social networks*)

Recommendations (e-commerce)

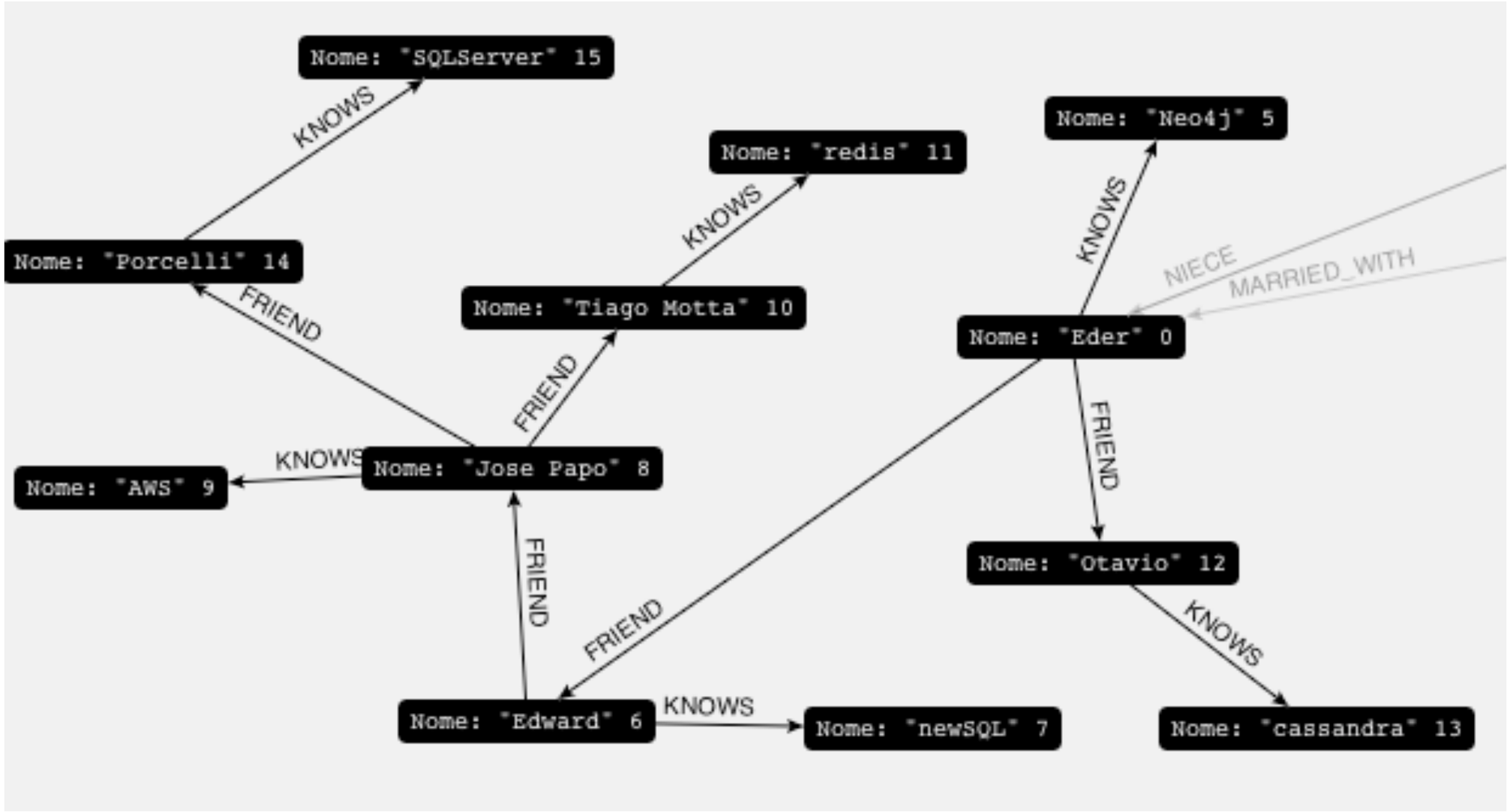
Path Finding (how do i know you)

A* (Least Cost path)

Data First Schema (bottom-up)

Schema Evolution

Property graph



The world is connected

What is a graph database?

A database with an explicit graph structure

Each node knows its adjacent nodes

As the number of the nodes increases,
the cost of a local step (or hop) remains the same

Plus a index for lookups

Neo4j

THE graph database

Graph Database + Lucene Index

Property Graph

Embeddable and server

REST interface

Stable

Full ACID (atomicity, consistency, isolation, durability)

High Availability (with Enterprise Edition)

32 Billion Nodes
32 Billion Relationships
64 Billion Properties

Schema free

Social network "path exists"

~1k persons
~ 50 friends/persons
pathExists(a,b)
depth 4

	# persons	query time
Relational database	1000	2000ms
Neo4j	1000	2ms
Neo4j	1000000	2ms

If you've ever

Joined more than 7 tables together

Modeled a graph in a table

Fells icky when need to "adapt" your
ER model to fit on a DB

Tried to write some crazy view/stored procedure with multiple recursive self an inner joins

You should use

Neo4j

Neo4j

Starting the server

Embedded

```
<dependency>
        <groupId>org.neo4j</groupId>
        <artifactId>neo4j</artifactId>
        <version>1.9</version>
</dependency>
```

```
GraphDatabaseService graphDb = new GraphDatabaseFactory()
                                    .newEmbeddedDatabase(DB_PATH);
```

Standalone

```
neo4j-community-1.9 tiagobento$ bin/neo4j start
```

```
Starting Neo4j Server...WARNING: not changing user
process [80169]... waiting for server to be ready..... OK.
```

```
Go to http://localhost:7474/webadmin/
for administration interface.
```

Neo4j

Overview
Dashboard

Explore and edit
Data browser

Power tool
Console

Add an
Index

14	13	23	3
nodes	properties	relationships	relationship types

Year | One month | One week | One day | 6 hours | 30 minutes

Overview
Dashboard

Explore and edit
Data browser

Power tool
Console

Add an
Index

```
START root=node(0) // Start with the
                   // reference node
RETURN root        // and return it.

// Hit CTRL+ENTER to execute
```

Returned **14 rows.** Query took **776ms**

n
Node 0
Node 1
Node 2
Node 3

Manipulating data

Cypher

Neo4j's Query Language

```
CREATE

( _1 { name: "Radiohead" }),
( _2 { name: "The Black Keys" }),
( _3 { name: "Joy Division" }),
( _4 { name: "Los hermanos" }),
( _5 { name: "Oasis" }),
( _6 { name: "Daft Punk" }),
( _7 { name: "Felguk" }),
( _8 { name: "Deadmau5" }),

( eder { name: "Eder Ignatowicz", age: 22 }),
( tiago { name: "Tiago Bento", age: 19 }),
```

Tiago Bento

Radiohead

The Black Keys

Joy Division

Los hermanos

Oasis

Daft Punk

Deadmau5

Felguk

Eder Ignatowicz

```
eder-[:LIKES]->_8,
eder-[:LIKES]->_7,
eder-[:LIKES]->_6,
eder-[:LIKES]->_5,
eder-[:LIKES]->_4,

tiago-[:LIKES]->_6,
tiago-[:LIKES]->_5,
tiago-[:LIKES]->_4,
tiago-[:LIKES]->_3,
tiago-[:LIKES]->_2,
tiago-[:LIKES]->_1

RETURN *
```

Tiago Bento

Radiohead

The Black Keys

Joy Division

Los hermanos

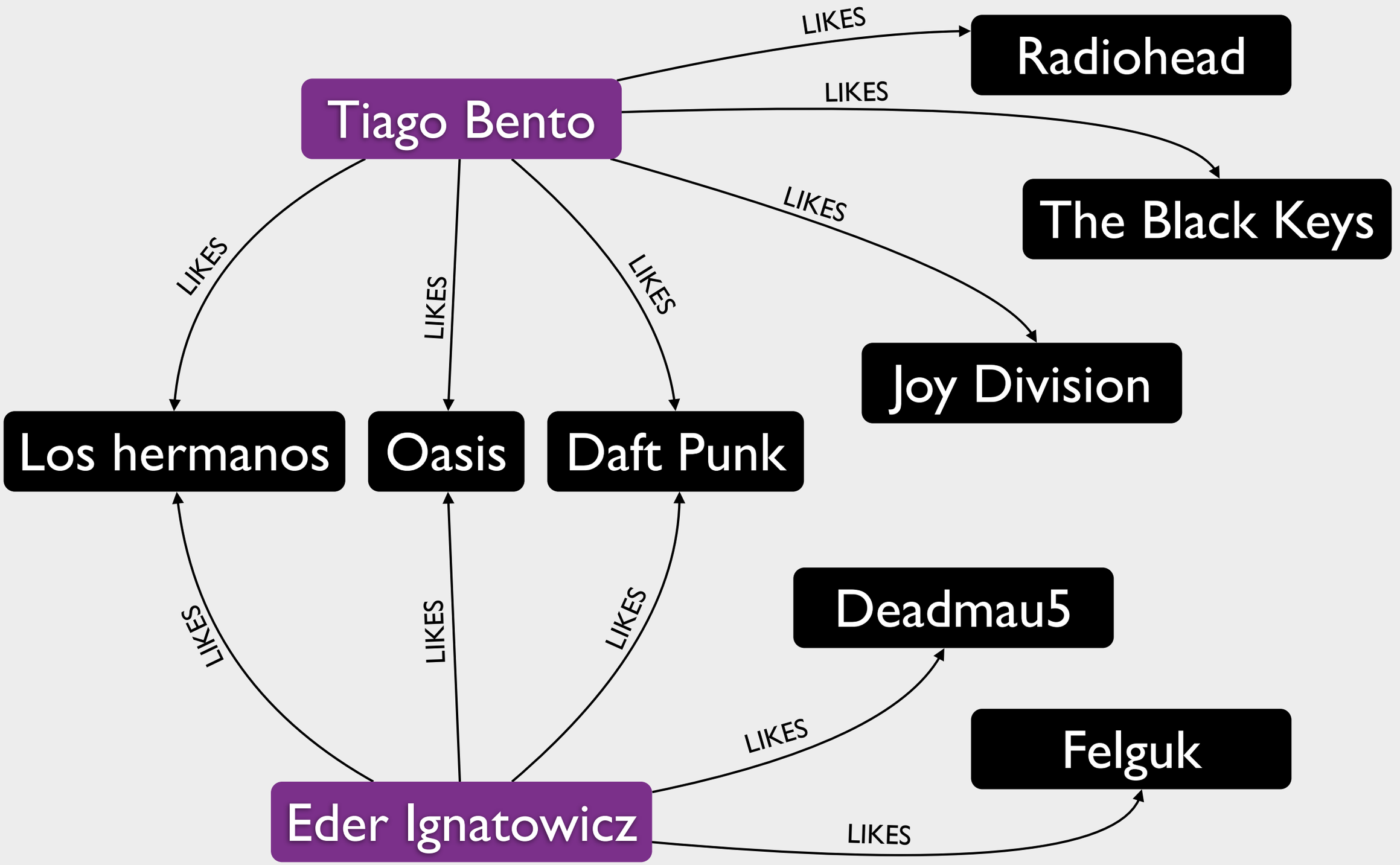
Oasis

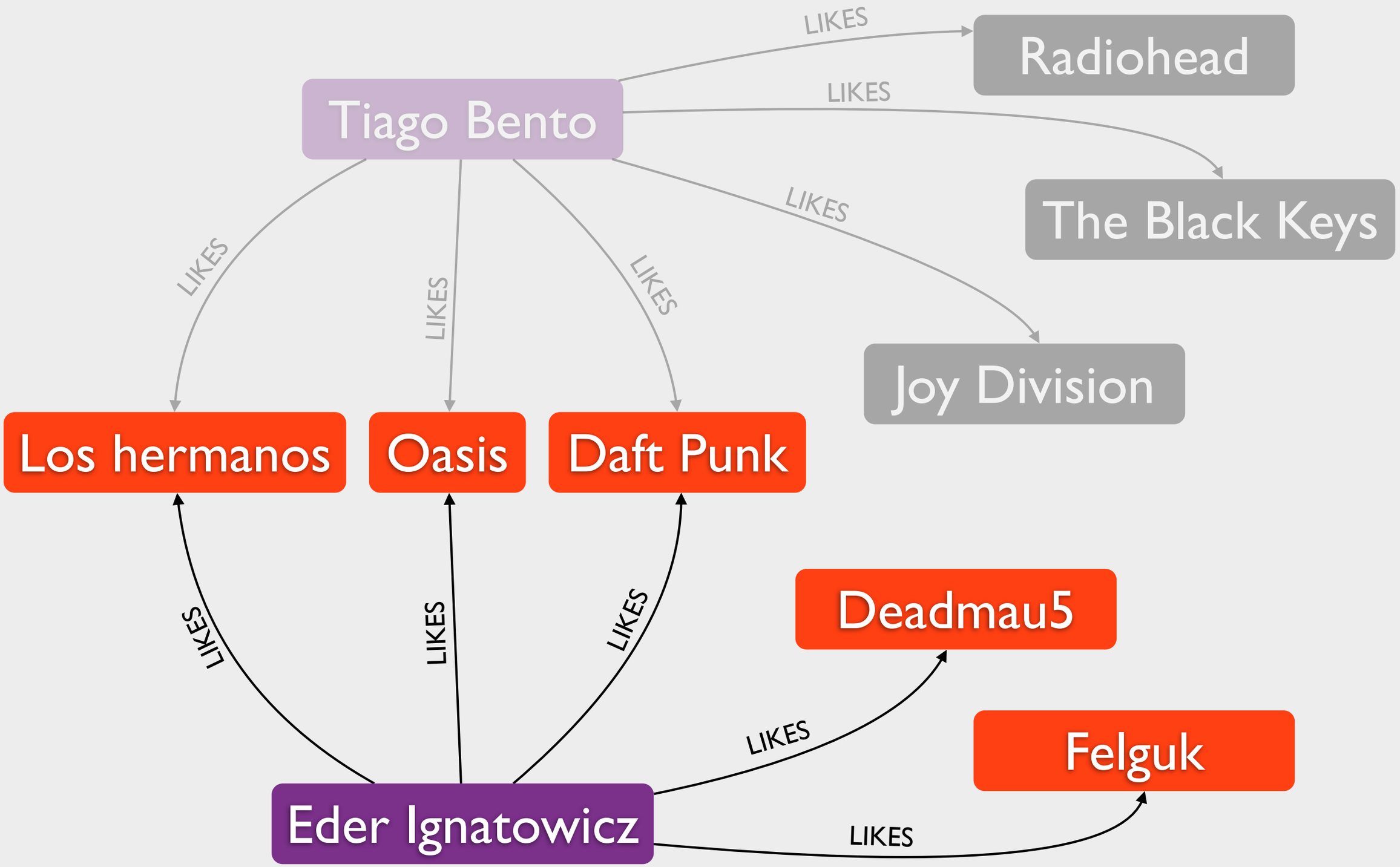
Daft Punk

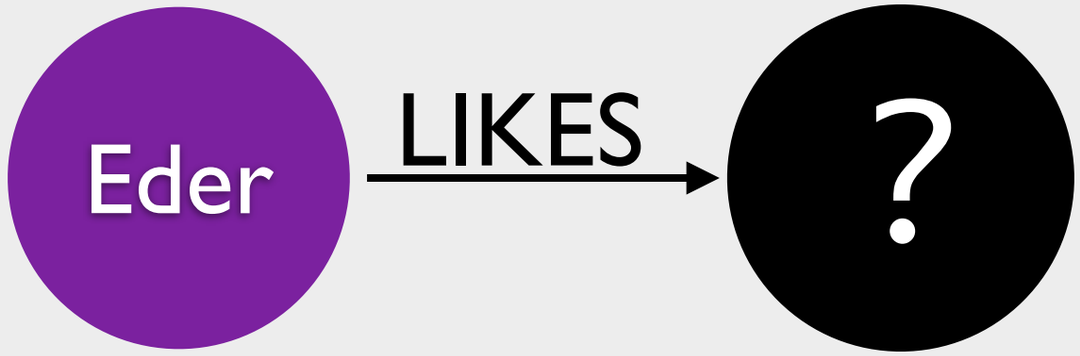
Deadmau5

Felguk

Eder Ignatowicz

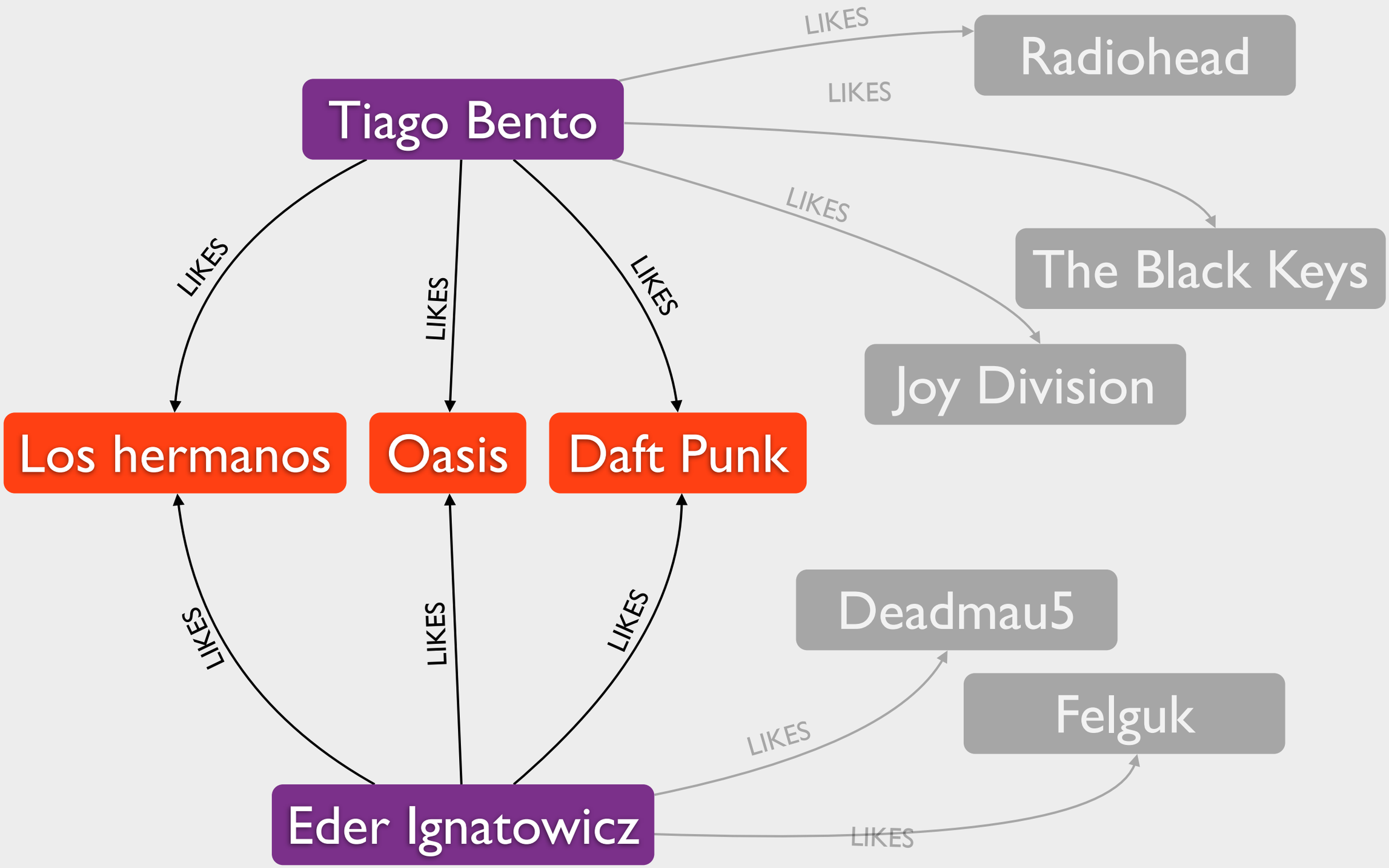

Tiago Bento
LIKES
Radiohead
LIKES
The Black Keys
LIKES
Joy Division
LIKES
Daft Punk
LIKES
Oasis
LIKES
Los hermanos
Eder Ignatowicz
LIKES
Deadmau5
LIKES
Felguk
LIKES
LIKES
LIKES
LIKES


Tiago Bento
LIKES
Radiohead
LIKES
The Black Keys
LIKES
Joy Division
LIKES
LIKES
LIKES
Los hermanos
Oasis
Daft Punk
Eder Ignatowicz
LIKES
LIKES
LIKES
Deadmau5
LIKES
Felguk
LIKES


Eder
LIKES
?

```
START  ed=node(9)
MATCH  (ed)-[:LIKES]->(a)
RETURN collect(a.name);
```

```
[ "Deadmau5", "Felguk", "Daft Punk", "Oasis", "Los hermanos" ]
```


Tiago Bento
LIKES
Radiohead
LIKES
LIKES
The Black Keys
LIKES
Joy Division
LIKES
LIKES
LIKES
Los hermanos
Oasis
Daft Punk
LIKES
LIKES
LIKES
Deadmau5
Felguk
Eder Ignatowicz
LIKES
LIKES


Eder
LIKES
?
LIKES
Tiago

Musical compatibility

```
START  ti=node(10), ed=node(9)
MATCH  (ed)-[:LIKES]->(a)<-[:LIKES]-(ti)
RETURN collect(a.name);
```

```
[ "Daft Punk", "Oasis", "Los hermanos" ]
```

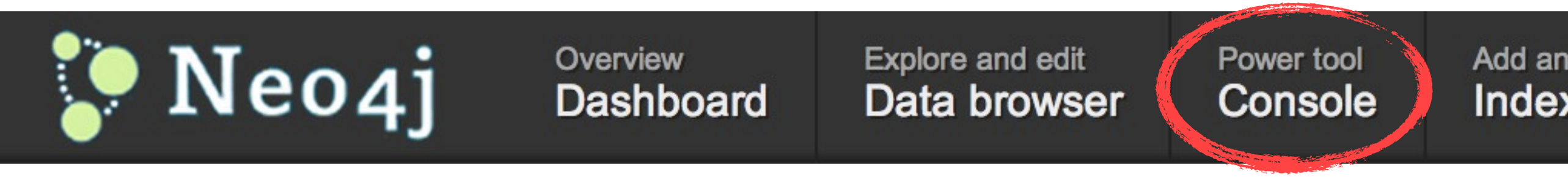
Musical compatibility

```
START  ti=node(10), ed=node(9)
MATCH  (ed)-[:LIKES]->(a)<-[:LIKES]-(ti)
WHERE  a.name =~ "(?i)D.*"
RETURN collect(a.name);
```

```
[ "Daft Punk" ]
```

Gremlin

Chaining navigation


Neo4j
Overview
Dashboard
Explore and edit
Data browser
Power tool
Console
Add an
Index


Neo4J Shell
Gremlin
HTTP
==>
==> \,,,/
==> (o o)
==> -----oOOo-(_)-oOOo-----
==>
==> Available variables:
==>   g = (neo4jgraph[EmbeddedGraphDatabase [data/graph.db]]
==> , null) out = (java.io.PrintStream@2393846a
==> , null)

Neo4j

Overview
Dashboard

Explore and edit
Data browser

Power tool
Console

Add and
Index

```
==>
==>          \,,,/
==>          (o o)
==> -----oOOo-(_)-oOOo-----
==>
==> Available variables:
==>   g = (neo4jgraph[EmbeddedGraphDatabase [data/graph.db]]
==> , null)  out = (java.io.PrintStream@2393846a
==> , null)
```

Neo4J Shell | Gremlin | HTTP

Gremlin

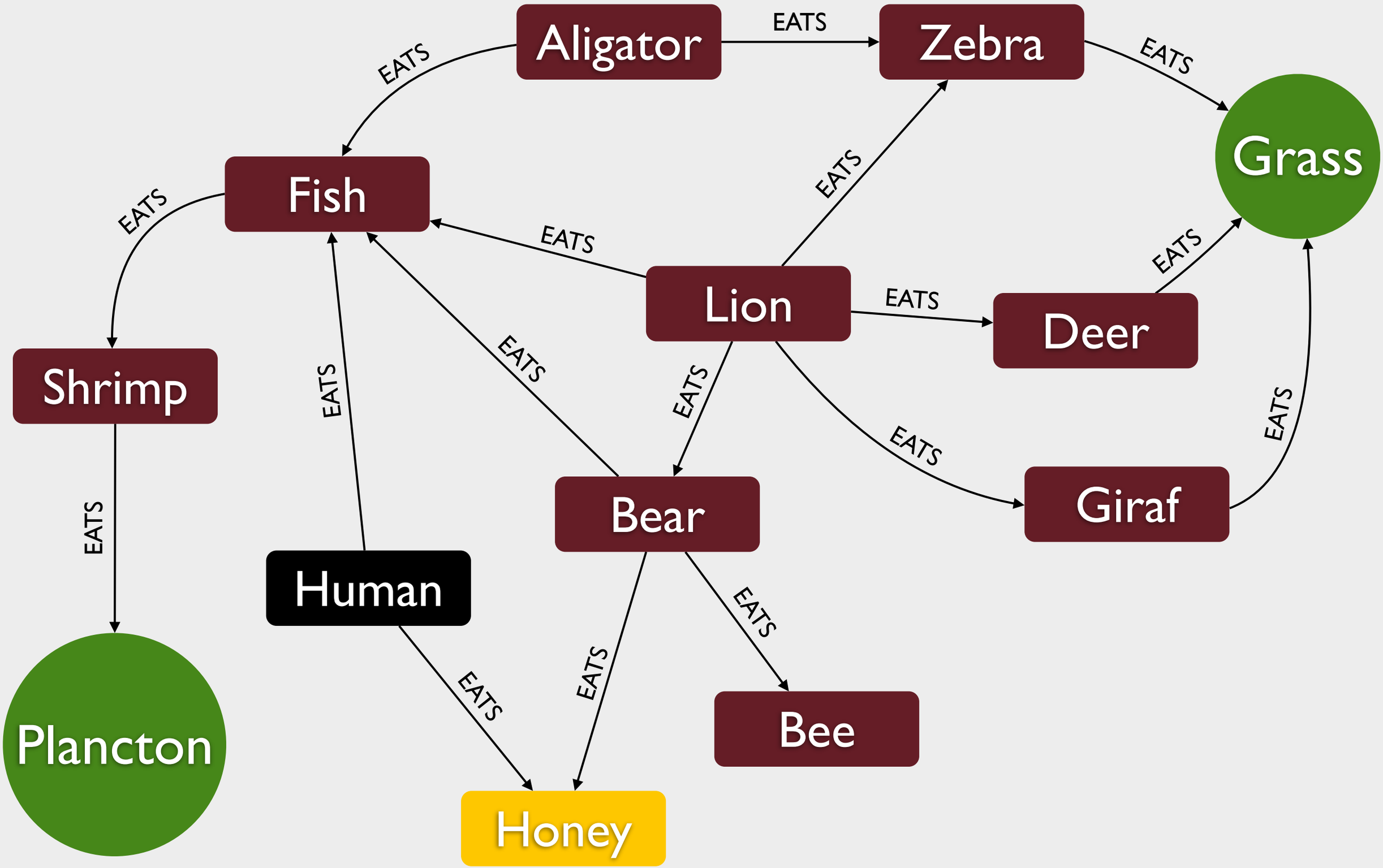
```
==>
==>           \,,,/
==>           (o o)
==> -----oOOo-(_)-oOOo-----
==>
==> Available variables:
==>   g = (neo4jgraph[EmbeddedGraphDatabase [data/graph.db]]
==> , null)  out = (java.io.PrintStream@14c55ea
==> , null)
gremlin>
```

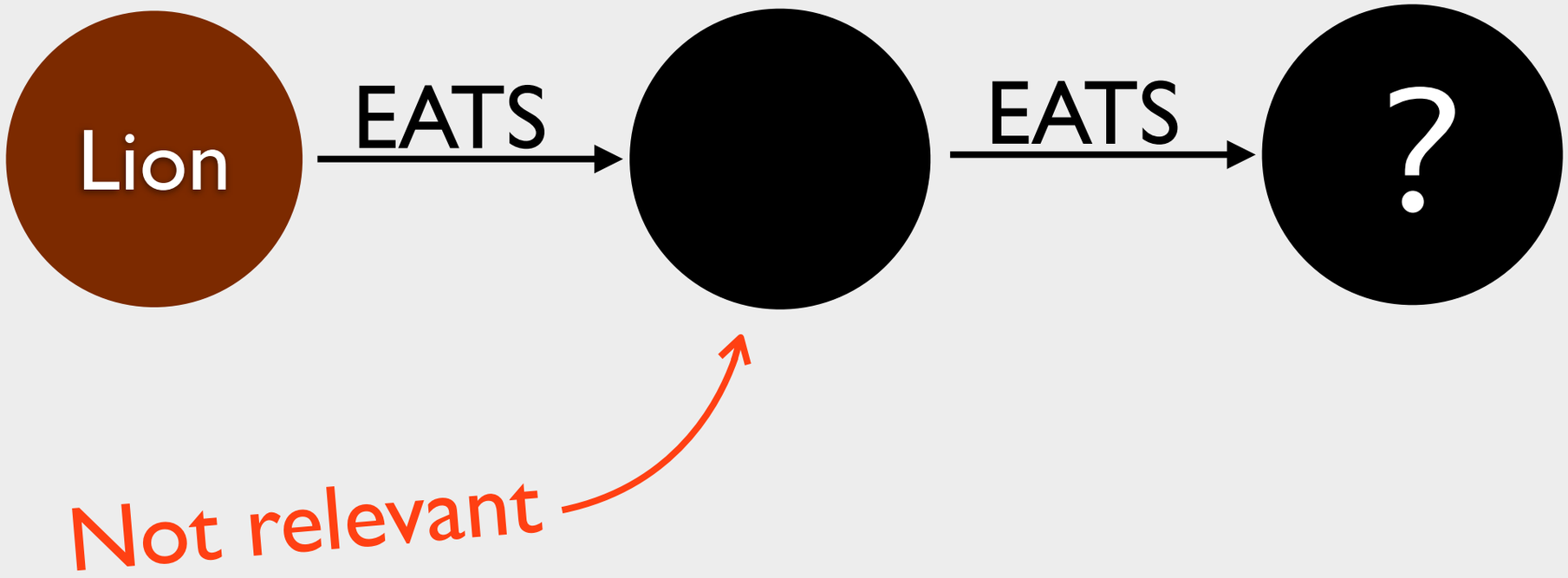
```
g.addVertex(1, [name: "Lion", size: "Big" ]);
g.addVertex(2, [name: "Aligator", size: "Big" ]);
g.addVertex(3, [name: "Zebra", size: "Big" ]);
g.addVertex(4, [name: "Deer", size: "Big" ]);
g.addVertex(5, [name: "Giraf", size: "Huge" ]);
```

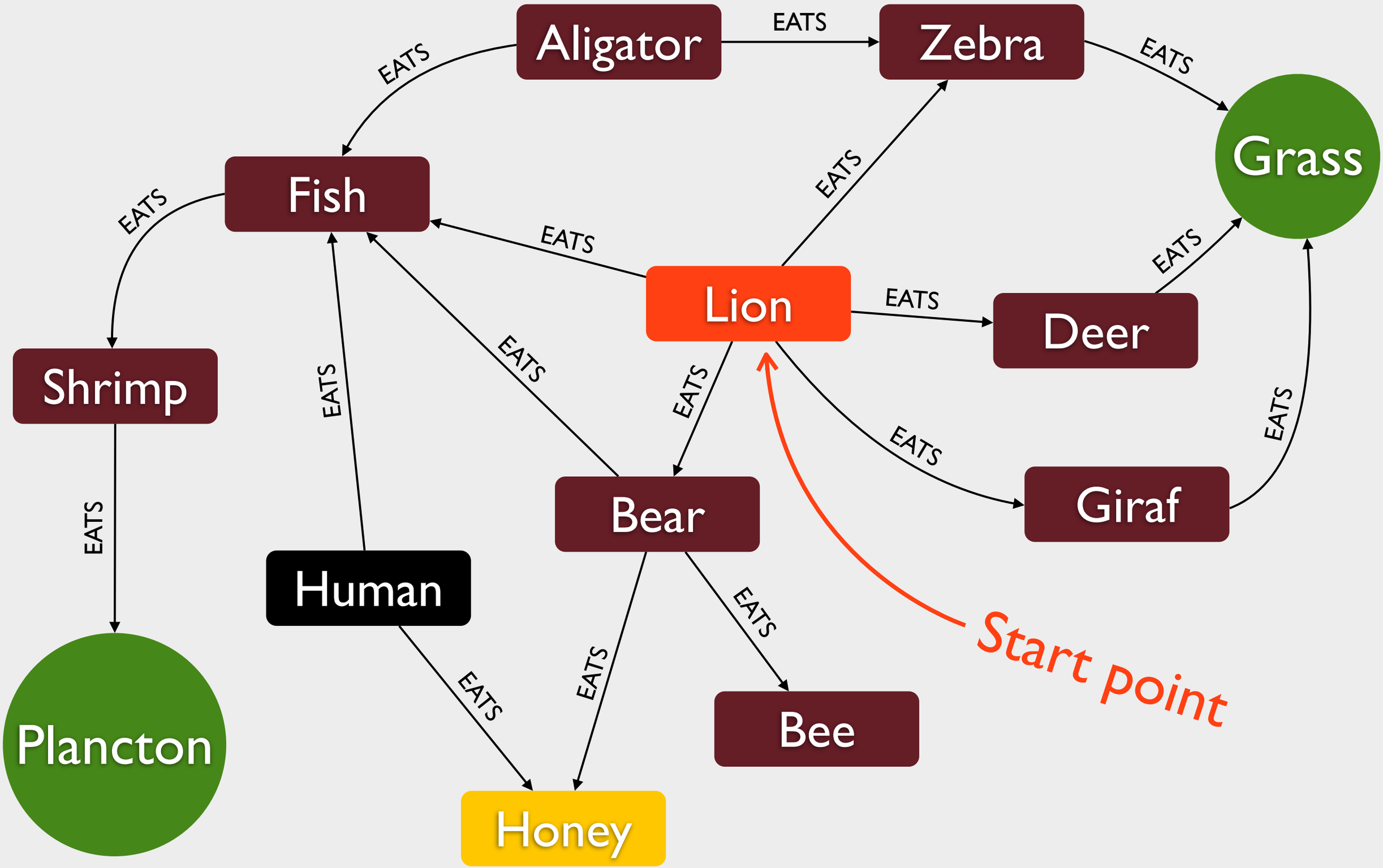
...

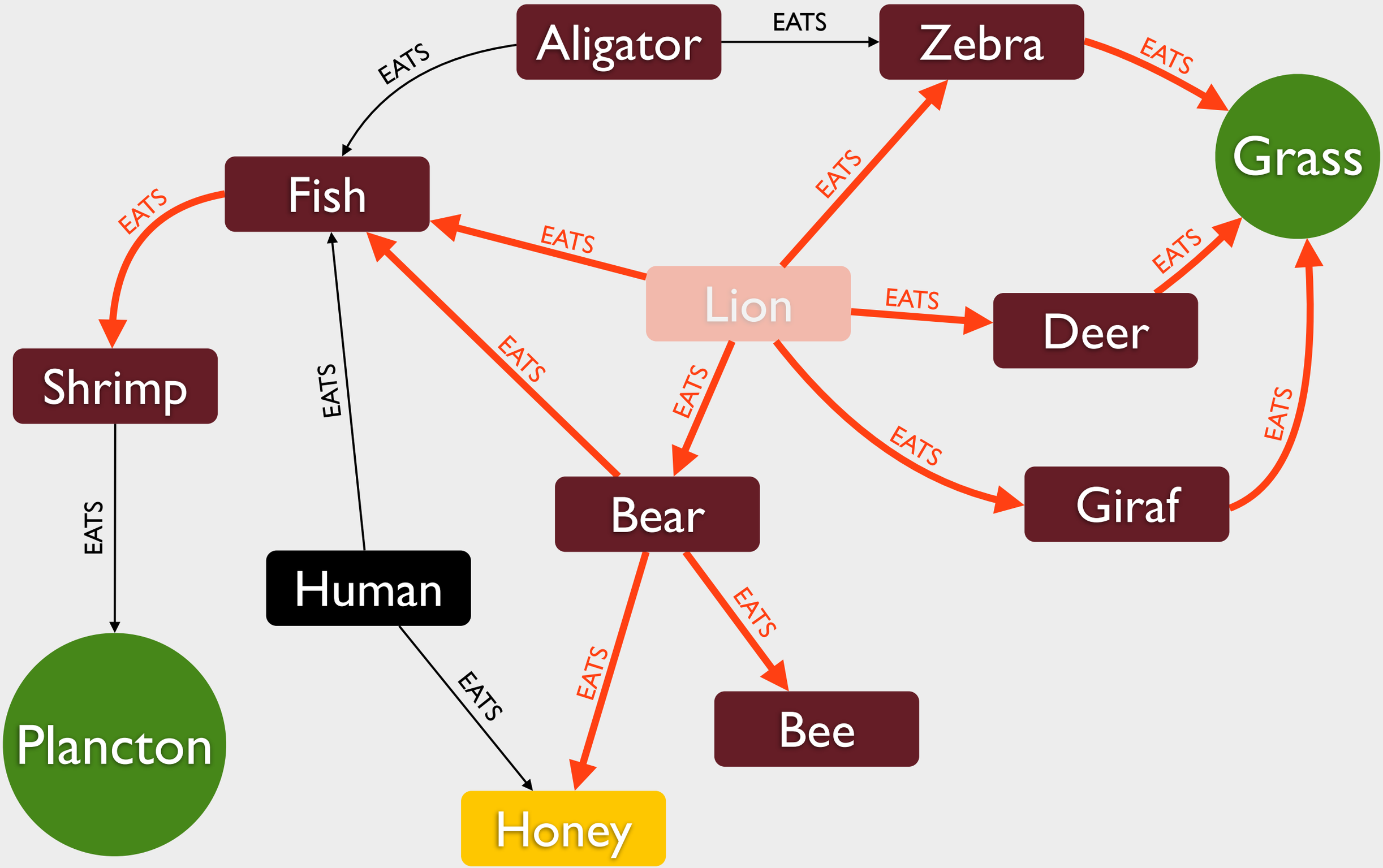
```
g.addEdge(g.v(1), g.v(3), 'EATS');
g.addEdge(g.v(1), g.v(4), 'EATS');
g.addEdge(g.v(1), g.v(5), 'EATS');
g.addEdge(g.v(1), g.v(7), 'EATS');
g.addEdge(g.v(1), g.v(6), 'EATS');
```

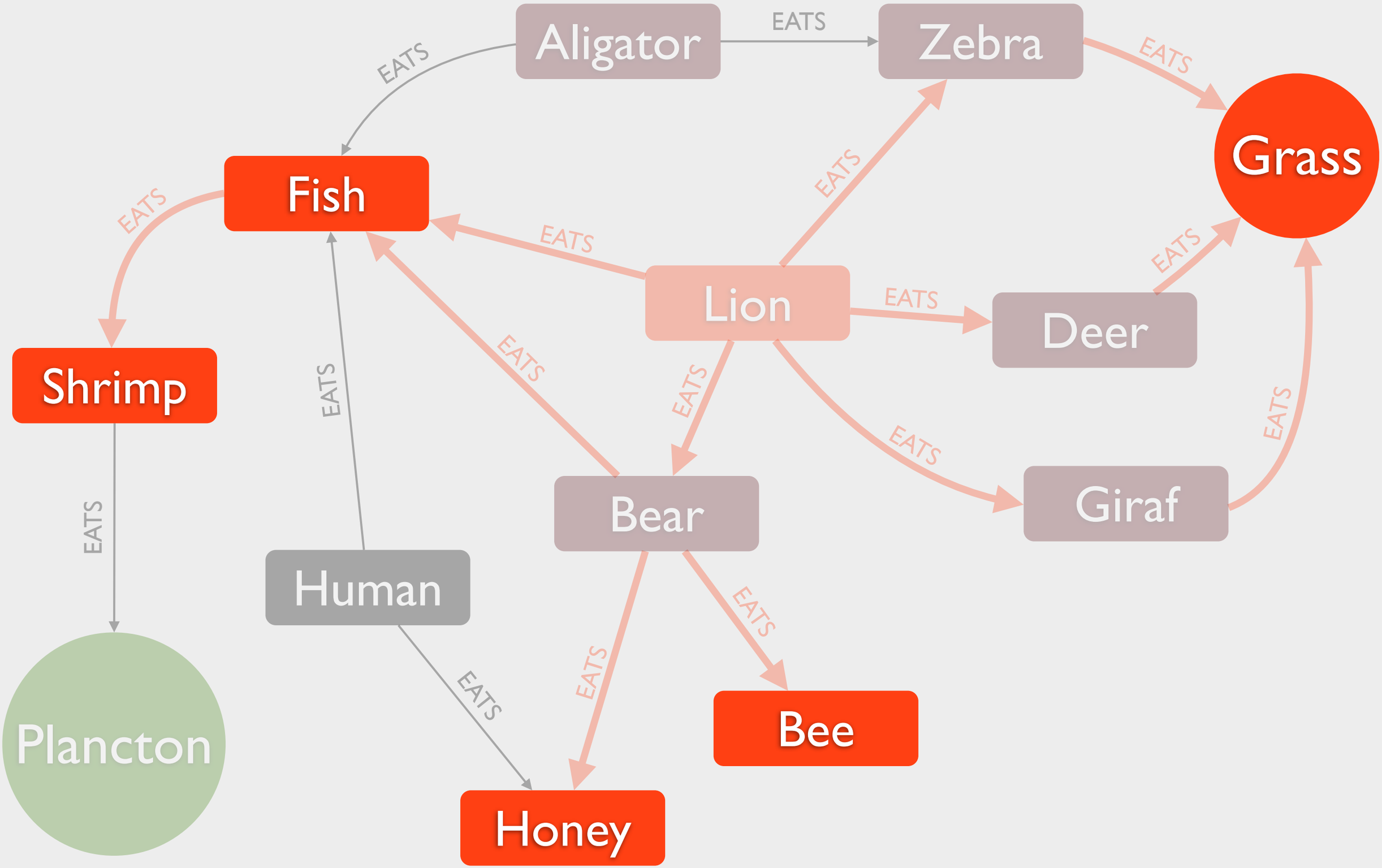
...


Aligator
Zebra
Grass
Fish
Lion
Deer
Shrimp
Bear
Giraf
Human
Bee
Plancton
Honey
EATS


Lion
EATS
EATS
?
Not relevant


Aligator
Zebra
Grass
Fish
Lion
Deer
Shrimp
Bear
Giraf
Human
Bee
Plancton
Honey
EATS
Start point


Aligator
EATS
Zebra
EATS
Grass
EATS
Fish
EATS
EATS
Lion
EATS
Deer
EATS
Shrimp
EATS
EATS
EATS
EATS
EATS
EATS
Giraf
EATS
EATS
Bear
EATS
Human
EATS
EATS
Bee
Plancton
Honey


Aligator
EATS
Zebra
EATS
Grass
EATS
Fish
EATS
EATS
EATS
Lion
EATS
Deer
EATS
Shrimp
EATS
EATS
EATS
EATS
EATS
EATS
Giraf
EATS
Bear
EATS
EATS
Human
EATS
Bee
Plancton
Honey

Lion's ID

```
g.v(1)
.out('EATS').out('EATS')
.name.unique().sort()
```

```
Bee
Fish
Grass
Honey
Shrimp
```

Lion's ID

```
g.v(1)
.out('EATS')
.loop(1){it.loops == 2}
.name.unique().sort()
```

```
Bee
Fish
Grass
Honey
Shrimp
```

Java API

Cypher Queries Execution

Traversal Framework

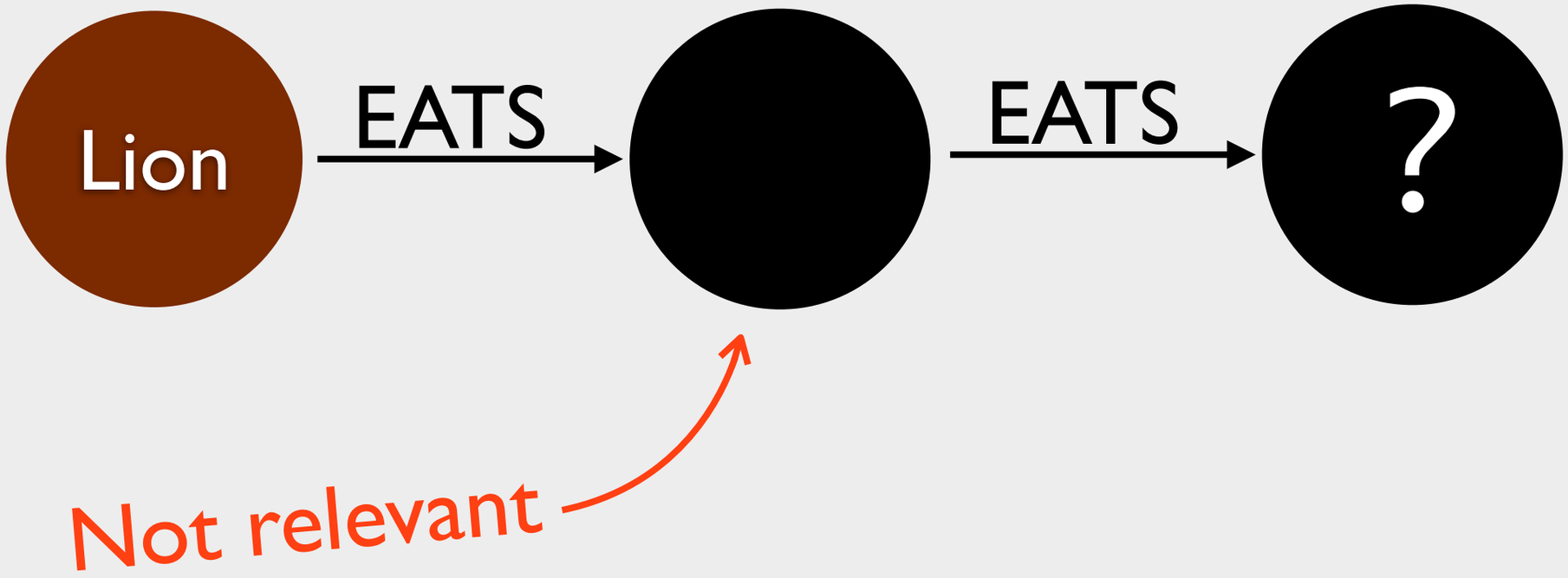
Cypher

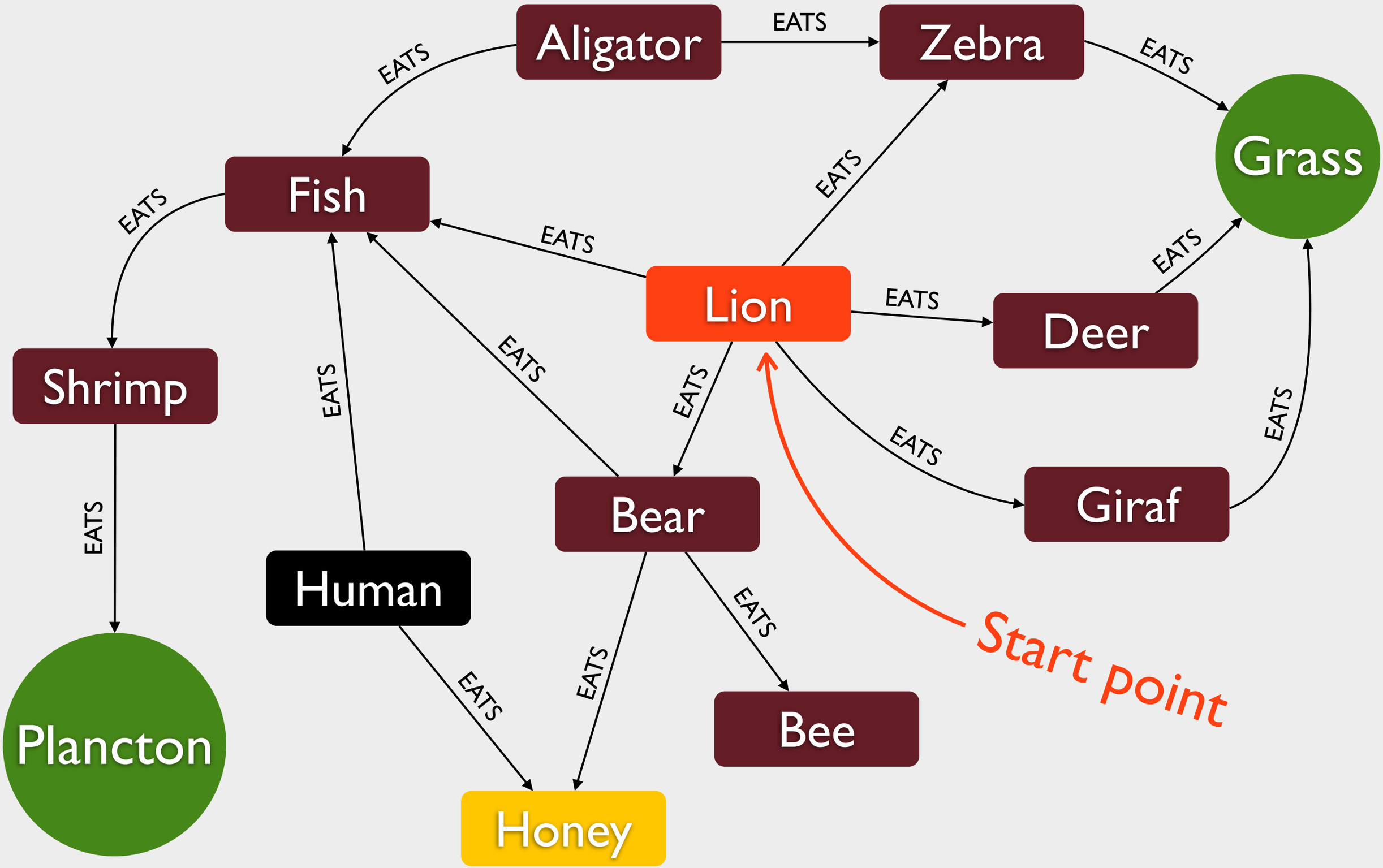
```
ExecutionEngine ee = new ExecutionEngine(graphDb);
ExecutionResult result = ee.execute("START n=node(1) RETURN n;");

Iterator<Node> columns = result.columnAs("n");
for (Node node : IteratorUtil.asIterable(columns)) {
    System.out.println(node.getProperty("name"));
}
```

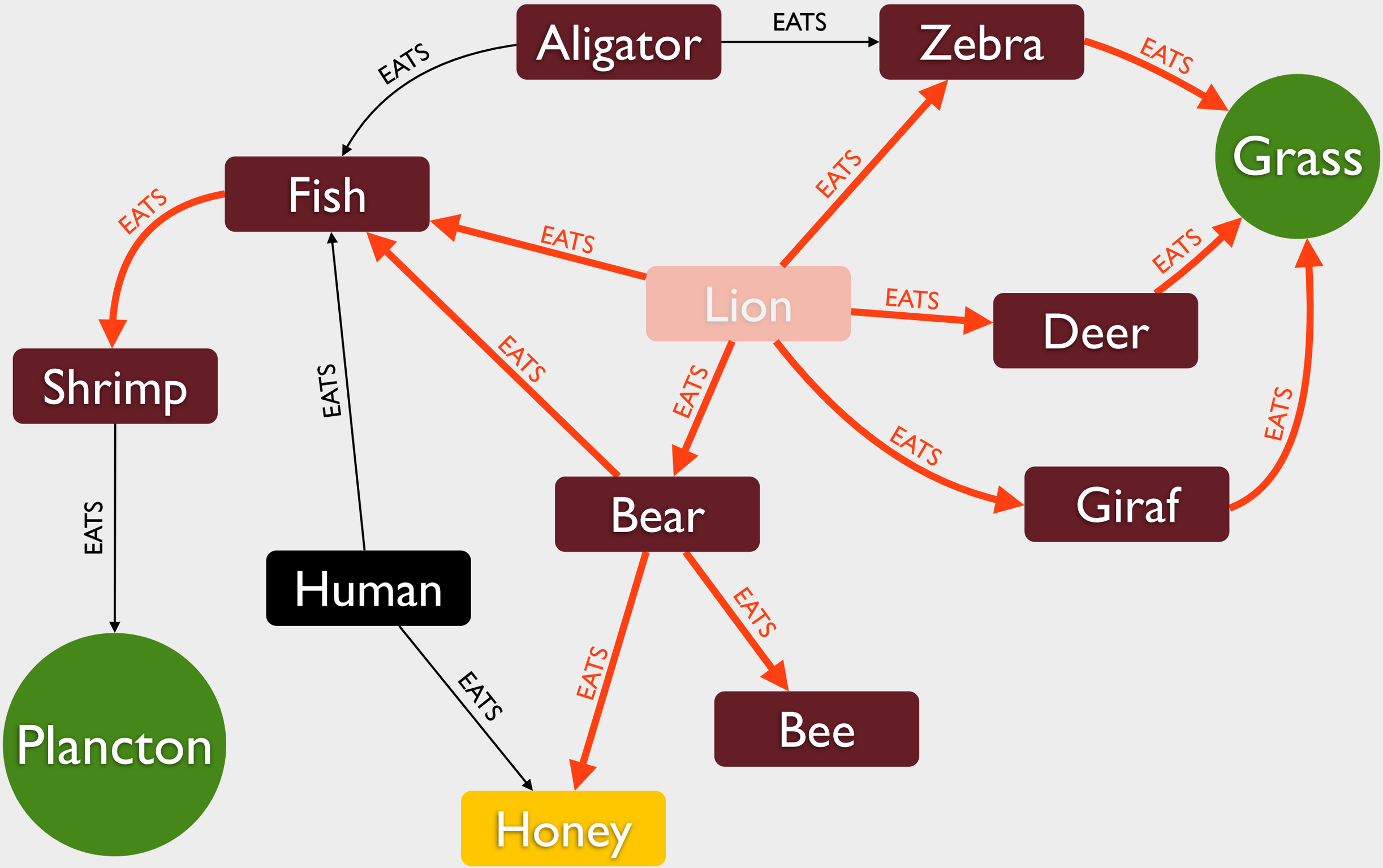
```
First node
```

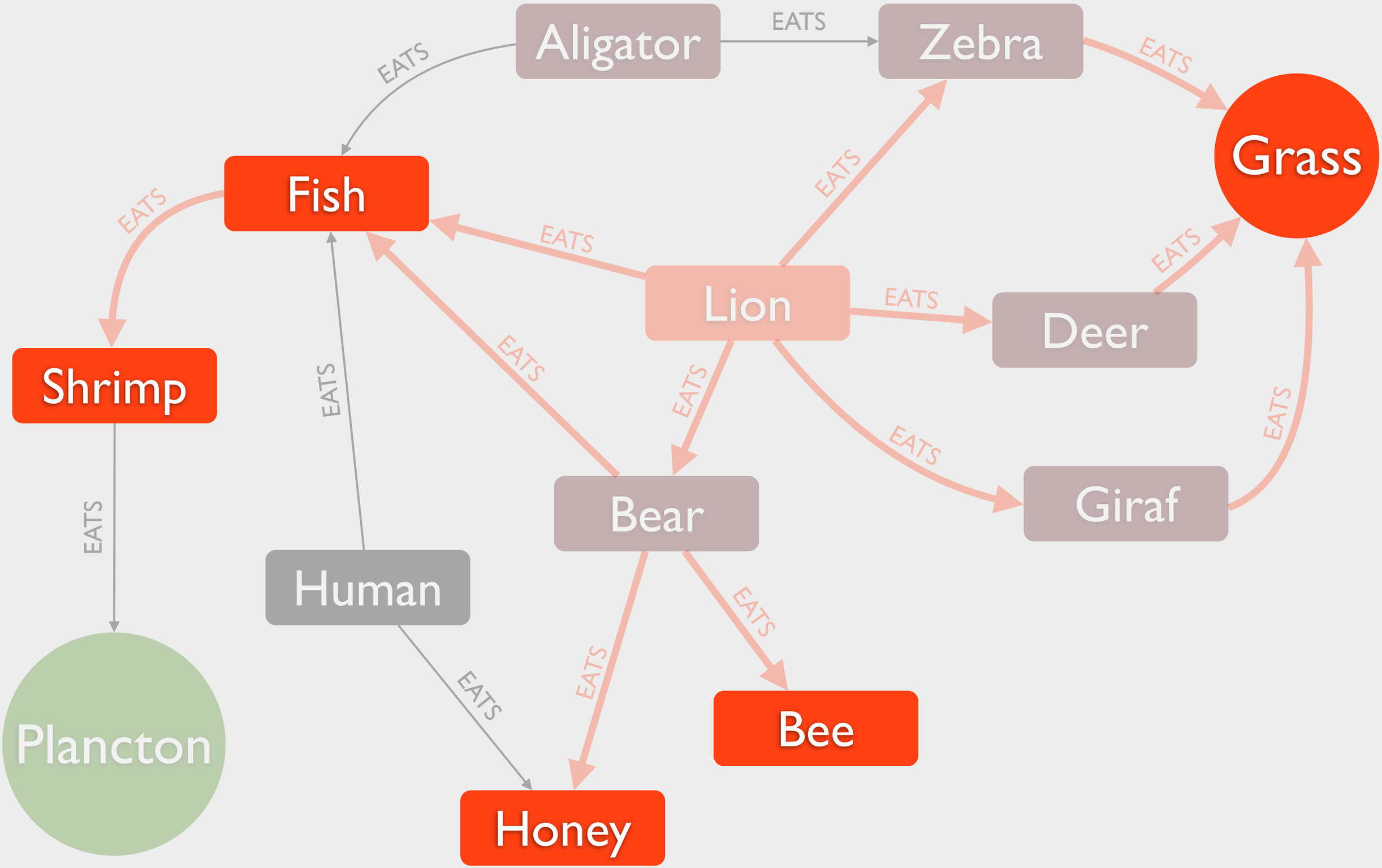
Traversal Framework

Traverse through your data


Lion
EATS
EATS
?
Not relevant


Aligator
Zebra
Grass
Fish
Lion
Deer
Shrimp
Bear
Giraf
Human
Bee
Plancton
Honey
EATS
Start point


Aligator
EATS
Zebra
EATS
Grass
EATS
Fish
EATS
EATS
Lion
EATS
Deer
EATS
Shrimp
EATS
EATS
EATS
EATS
EATS
EATS
Giraf
EATS
EATS
Bear
EATS
Human
EATS
EATS
Bee
Plancton
Honey


Aligator
EATS
Zebra
EATS
Grass
EATS
Fish
EATS
EATS
EATS
Lion
EATS
Deer
EATS
Shrimp
EATS
EATS
EATS
EATS
EATS
EATS
Giraf
EATS
Bear
EATS
EATS
Human
EATS
Bee
Plancton
Honey

Food chain

```
Node lion = graphDb.getNodeById(LION_ID);

TraversalDescription td = Traversal.description()
      .depthFirst()
      .relationships(RelationshipTypes.EATS, Direction.OUTGOING)
      .evaluator(Evaluators.atDepth(2))
      .uniqueness(Uniqueness.NODE_LEVEL);
```

```
for (Node node : td.traverse(lion).nodes()) {
   System.out.println(node.getProperty("name"));
}
```

```
Bee
Honey
Fish
Shrimp
Grass
```

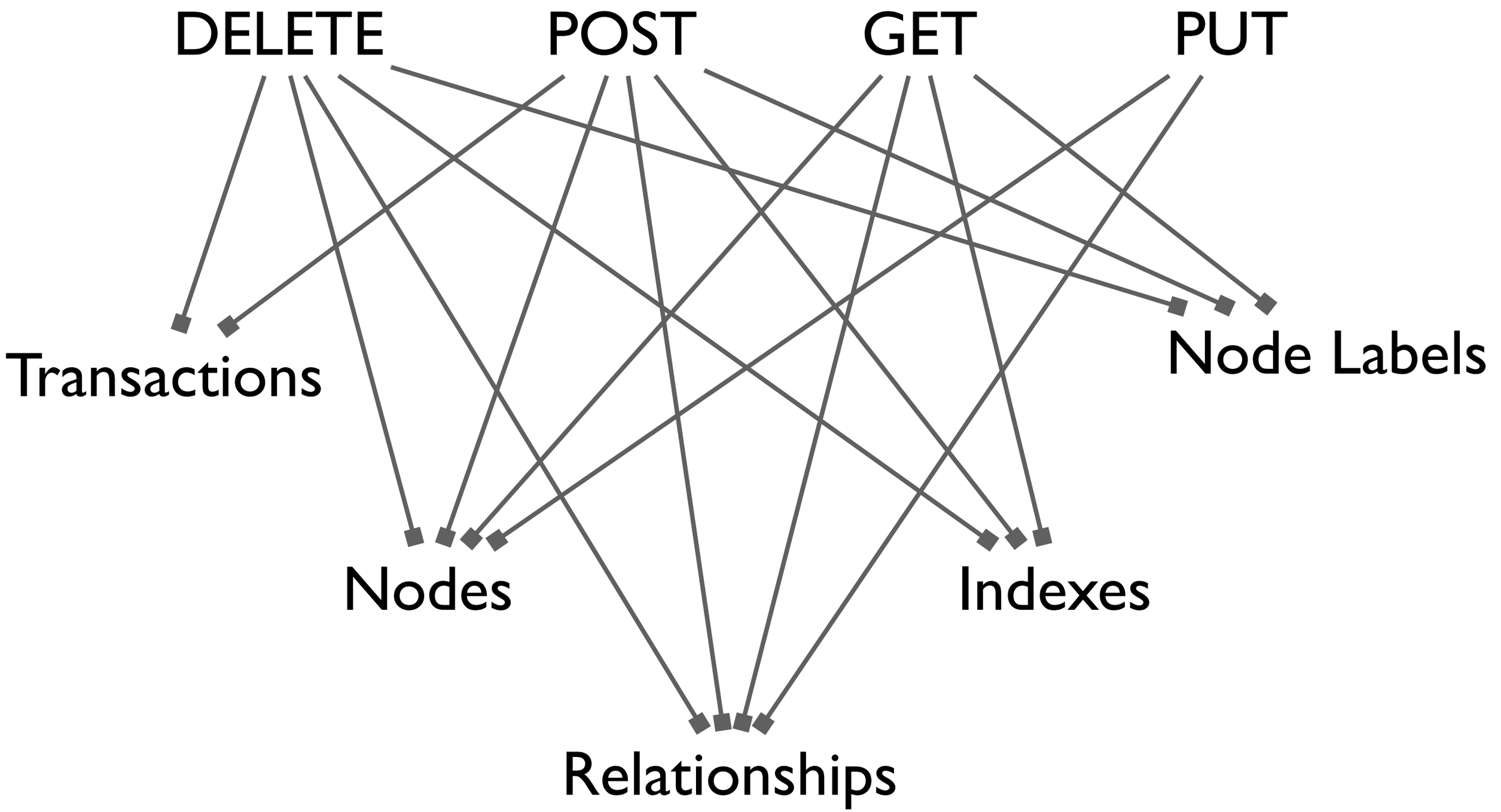
Food chain

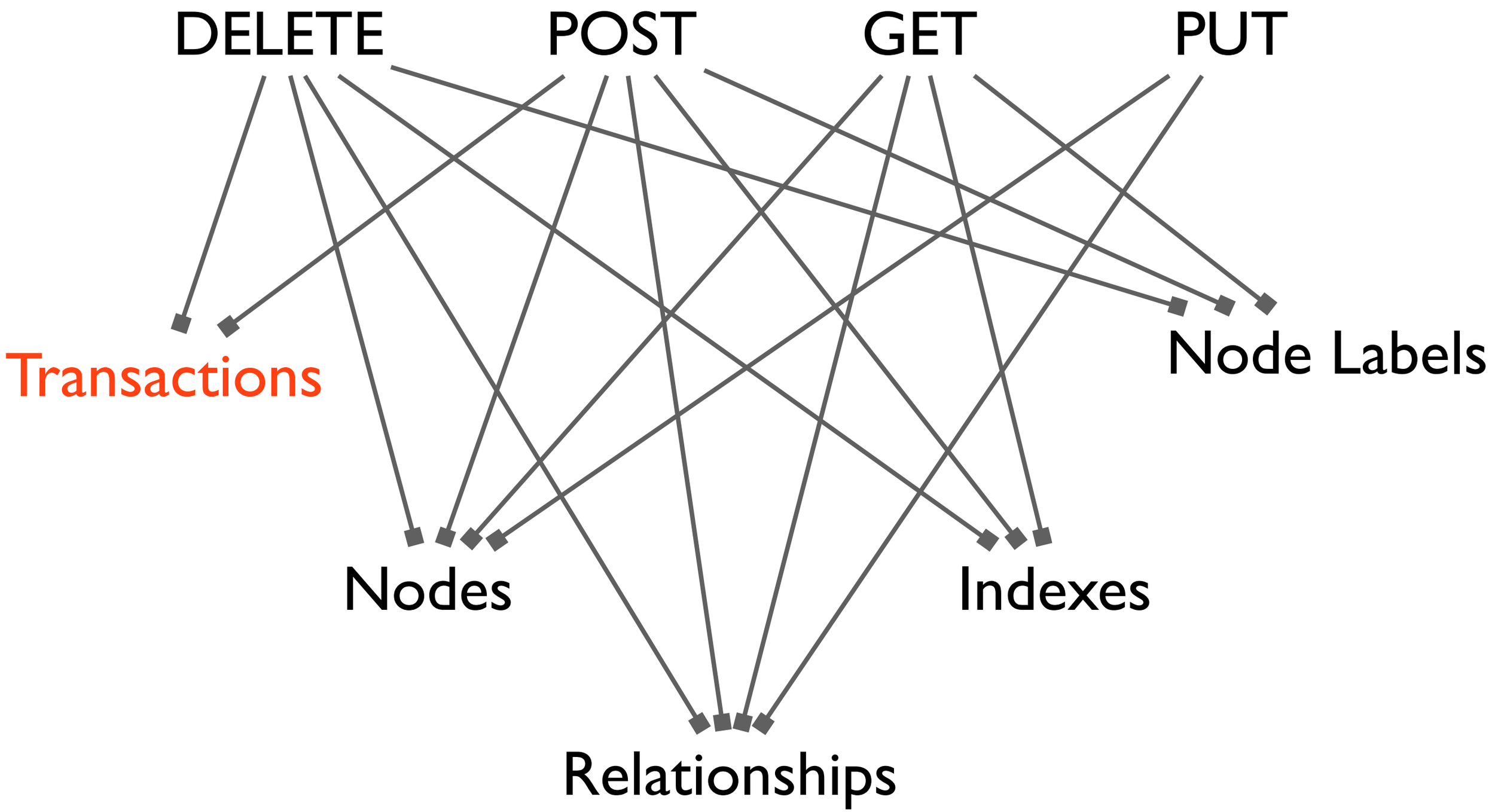
```
Node lion = graphDb.getNodeById(LION_ID);

TraversalDescription td = Traversal.description()
      .depthFirst()
      .relationships(RelationshipTypes.EATS, Direction.OUTGOING)
      .evaluator(Evaluators.atDepth(2))
      .uniqueness(Uniqueness.NODE_LEVEL);

for (Path path : td.traverse(lion)) {
   System.out.println(path);
}
```

```
(1)--[EATS,4]-->(6)--[EATS,22]-->(9)
(1)--[EATS,4]-->(6)--[EATS,11]-->(13)
(1)--[EATS,4]-->(6)--[EATS,10]-->(7)
(1)--[EATS,3]-->(7)--[EATS,12]-->(8)
(1)--[EATS,2]-->(5)--[EATS,9]-->(11)
```

REST API

Full power through HTTP Requests


DELETE
POST
GET
PUT
Transactions
Node Labels
Nodes
Indexes
Relationships


DELETE
POST
GET
PUT
Transactions
Node Labels
Nodes
Indexes
Relationships

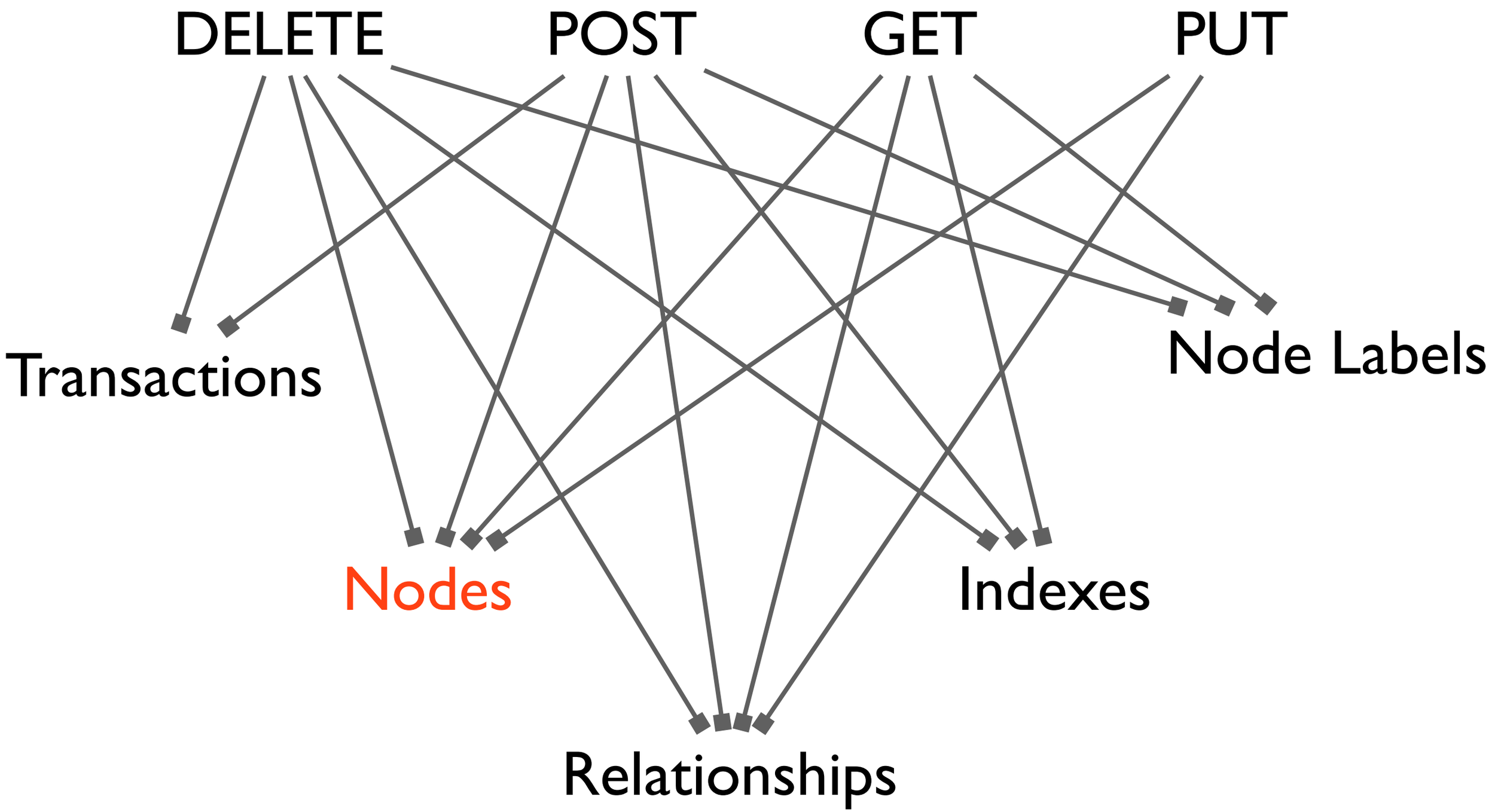
```
POST http://localhost:7474/db/data/transaction
```

```
POST http://localhost:7474/db/data/transaction/7
{
  "statements" : [ {
    "statement" : "CREATE n RETURN n"
  } ]
}
```

```
POST http://localhost:7474/db/data/transaction/7
{
  "statements" : [ {
    "statement" : "START n=node(1) DELETE n"
  } ]
}
```

```
POST http://localhost:7474/db/data/transaction/7/commit
```

TRANSACTION


DELETE
POST
GET
PUT
Transactions
Node Labels
Nodes
Indexes
Relationships

- POST http://localhost:7474/db/data/node
- Accept: application/json
- Content-Type: application/json

```
{
  "foo" : "bar"
}
```

- 201: Created
- Content-Length: 1156
- Content-Type: application/json
- Location: http://localhost:7474/db/data/node/5


```
{
  "extensions" : {
  },
  "paged_traverse" : "http://localhost:7474/db/data/node/5/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "labels" : "http://localhost:7474/db/data/node/5/labels",
  "outgoing_relationships" : "http://localhost:7474/db/data/node/5/relationships/out",
  "traverse" : "http://localhost:7474/db/data/node/5/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/5/properties/{key}",
  "all_relationships" : "http://localhost:7474/db/data/node/5/relationships/all",
  "self" : "http://localhost:7474/db/data/node/5",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/5/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/5/relationships/in",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/in/{-list|&|types}",
  "create_relationship" : "http://localhost:7474/db/data/node/5/relationships",
  "data" : {
    "foo" : "bar"
  }
}
```

- POST http://localhost:7474/db/data/cypher
- Accept: application/json
- Content-Type: application/json

```
{
  "query" :
  "START x  = node:node_auto_index(name={startName})
   MATCH path = (x-[r]-friend)
   WHERE friend.name = {name}
   RETURN TYPE(r)",
  "params" : {
    "startName" : "Tiago Bento",
    "name" : "Eder Ignatowicz"
  }
}
```

- 200: OK
- Content-Type: application/json

```
{
  "columns" : [ "TYPE(r)" ],
  "data" : [ [ "know" ] ]
}
```


Neo4j