Cypher-based Graph Pattern Matching in Apache Flink

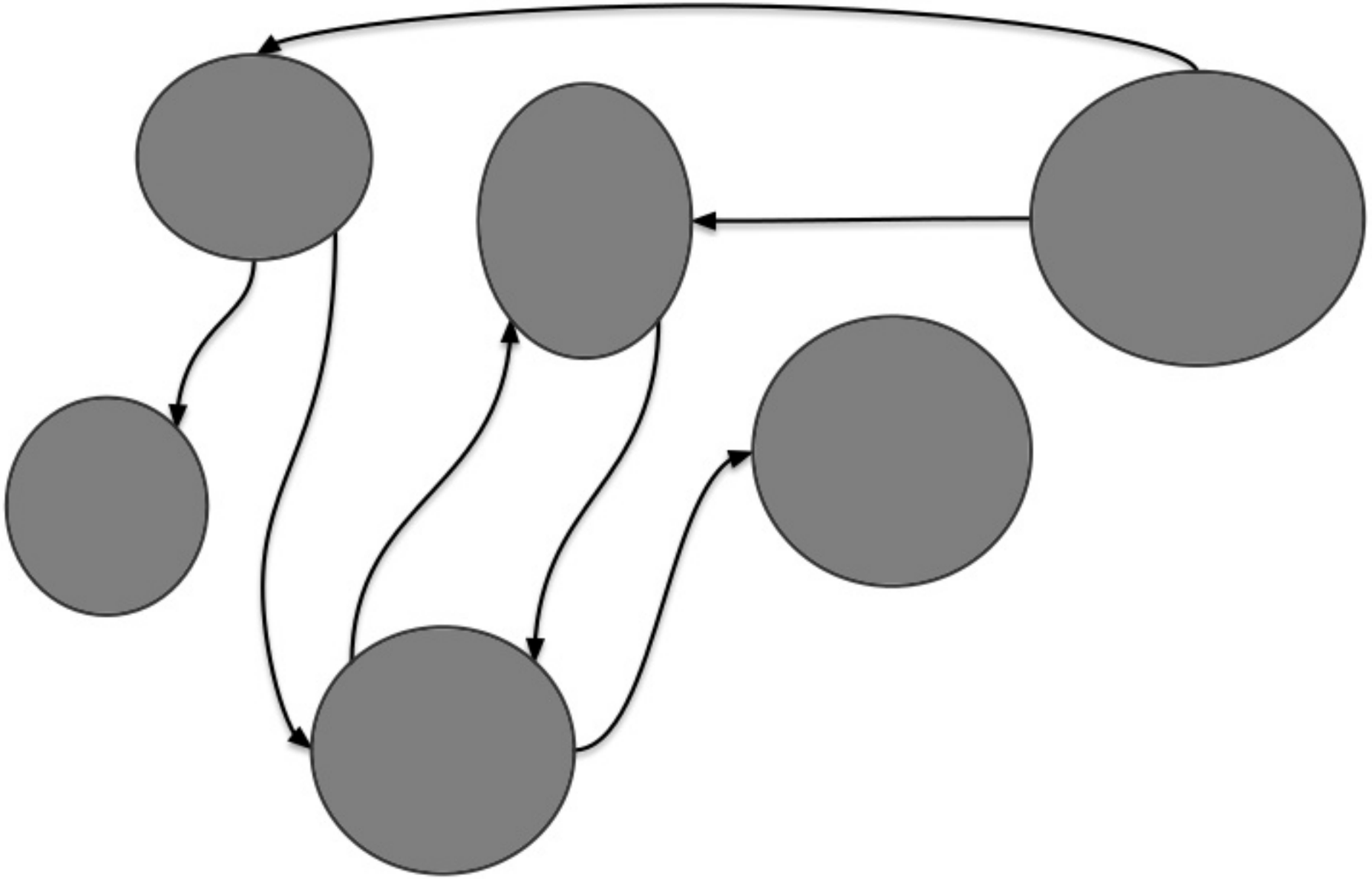
About us

Martin Junghanns
Software Engineer @ Neo4j
PhD Student @ University of Leipzig

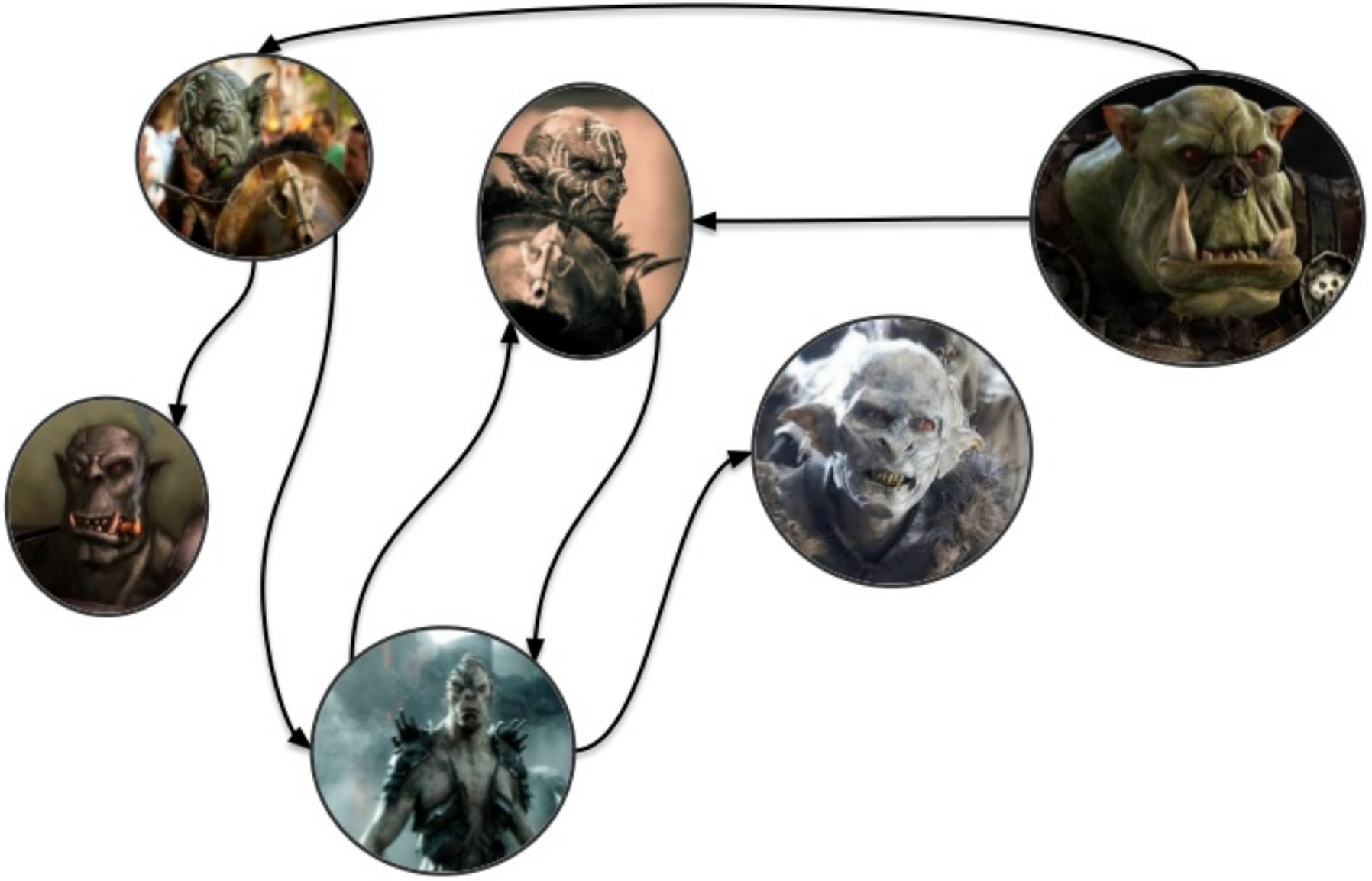
Max Kiessling
Software Engineer @ Neo4j

MOTIVATION

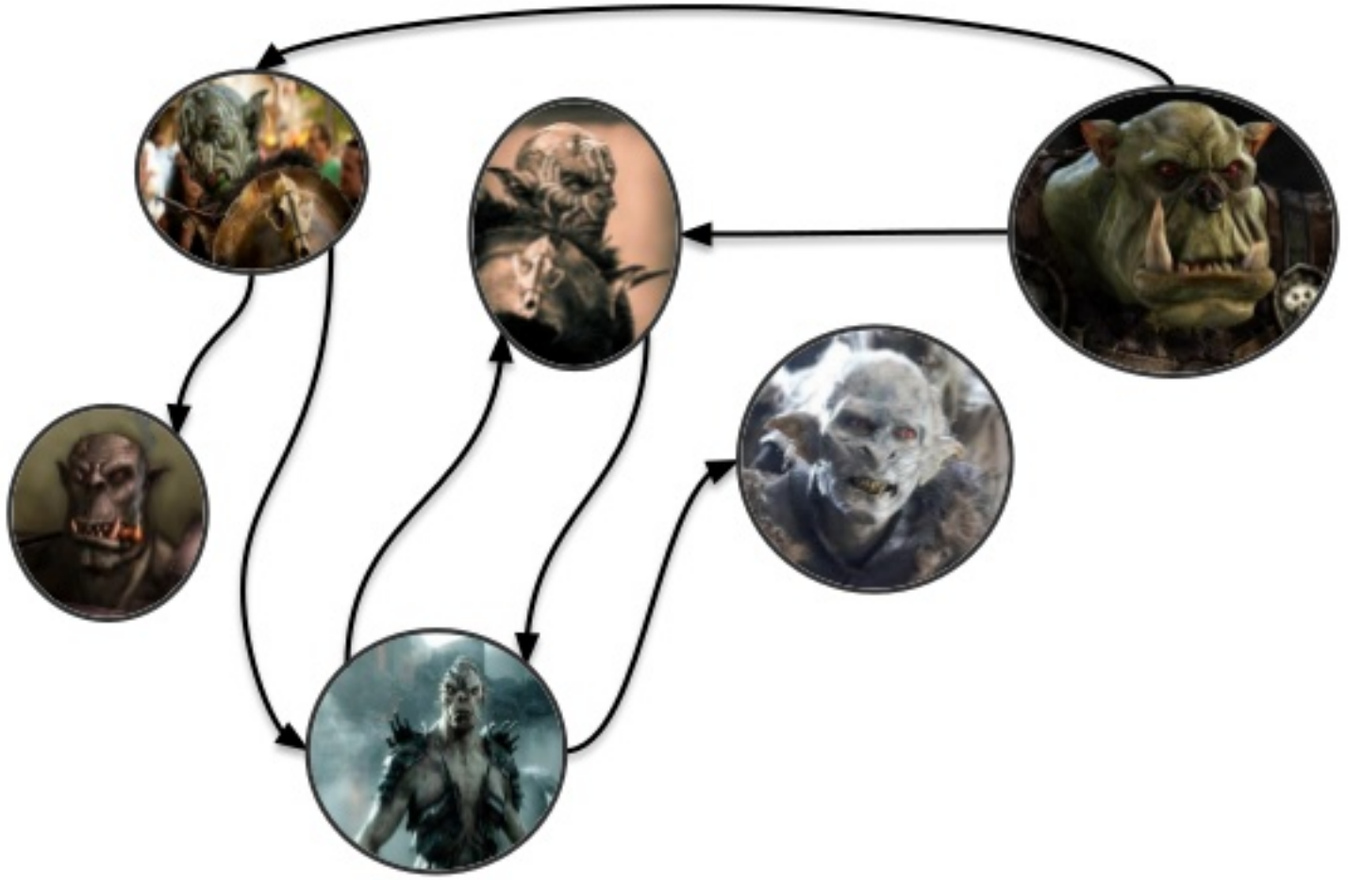
Graph = Set of Vertices + Set of Edges



A Real World Example - OrcBook



What Can You Do With It?



Sauron's Data Analyst: "Who are the closest enemies of each Orc?"

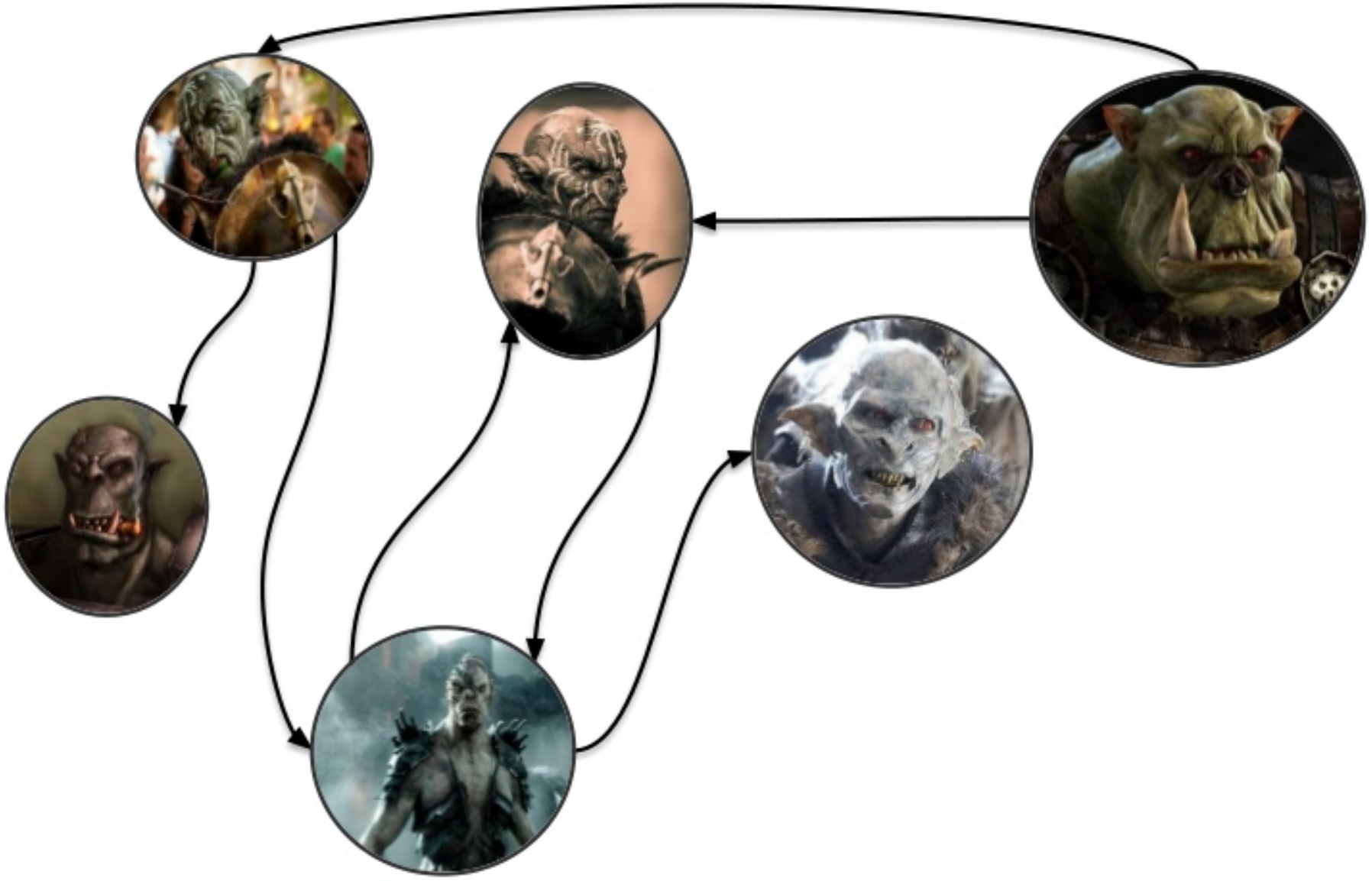

```
1 MATCH (a:Orc)-[:hates*1..2]->(b:Orc)
2 RETURN a,b
```

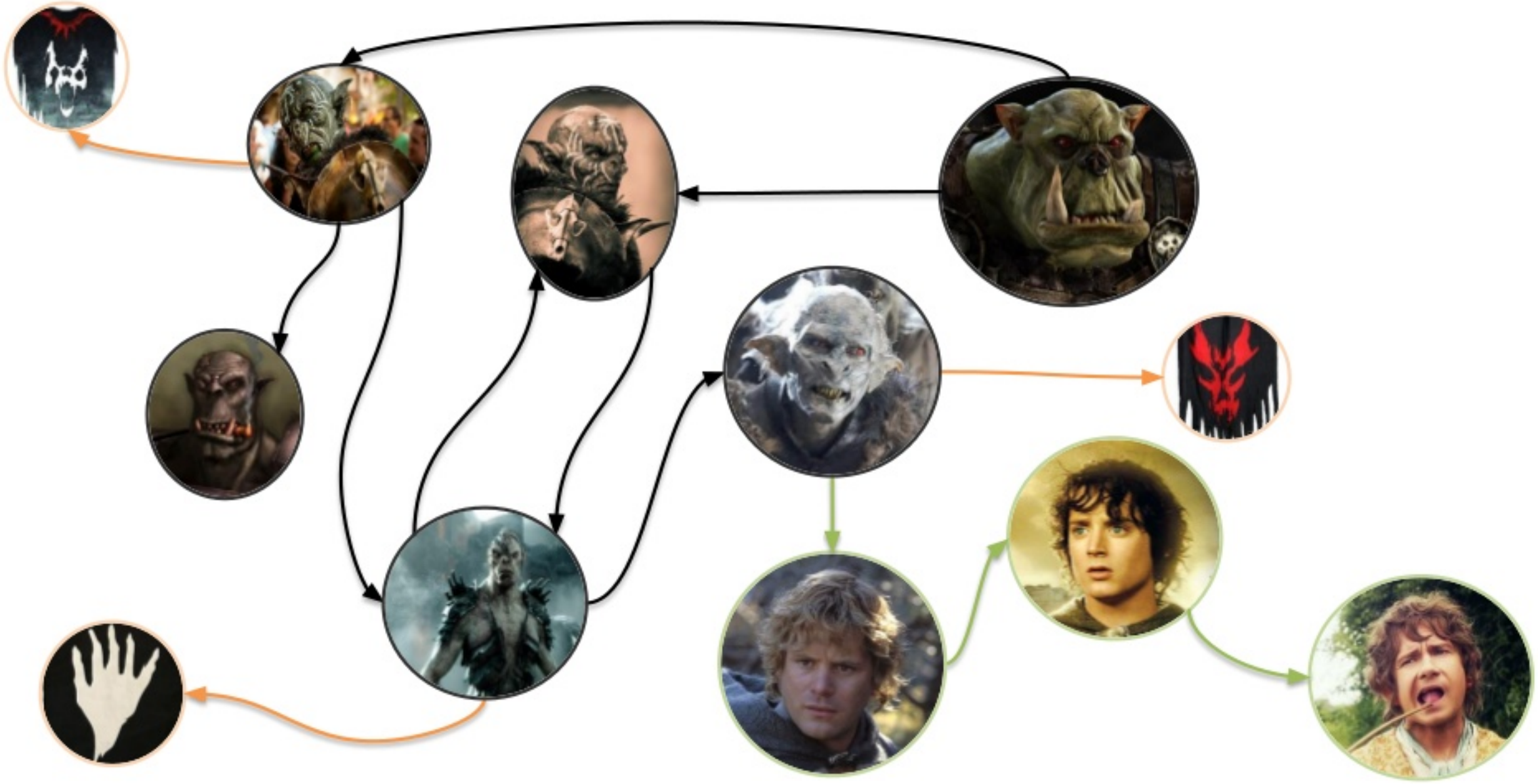
Cypher

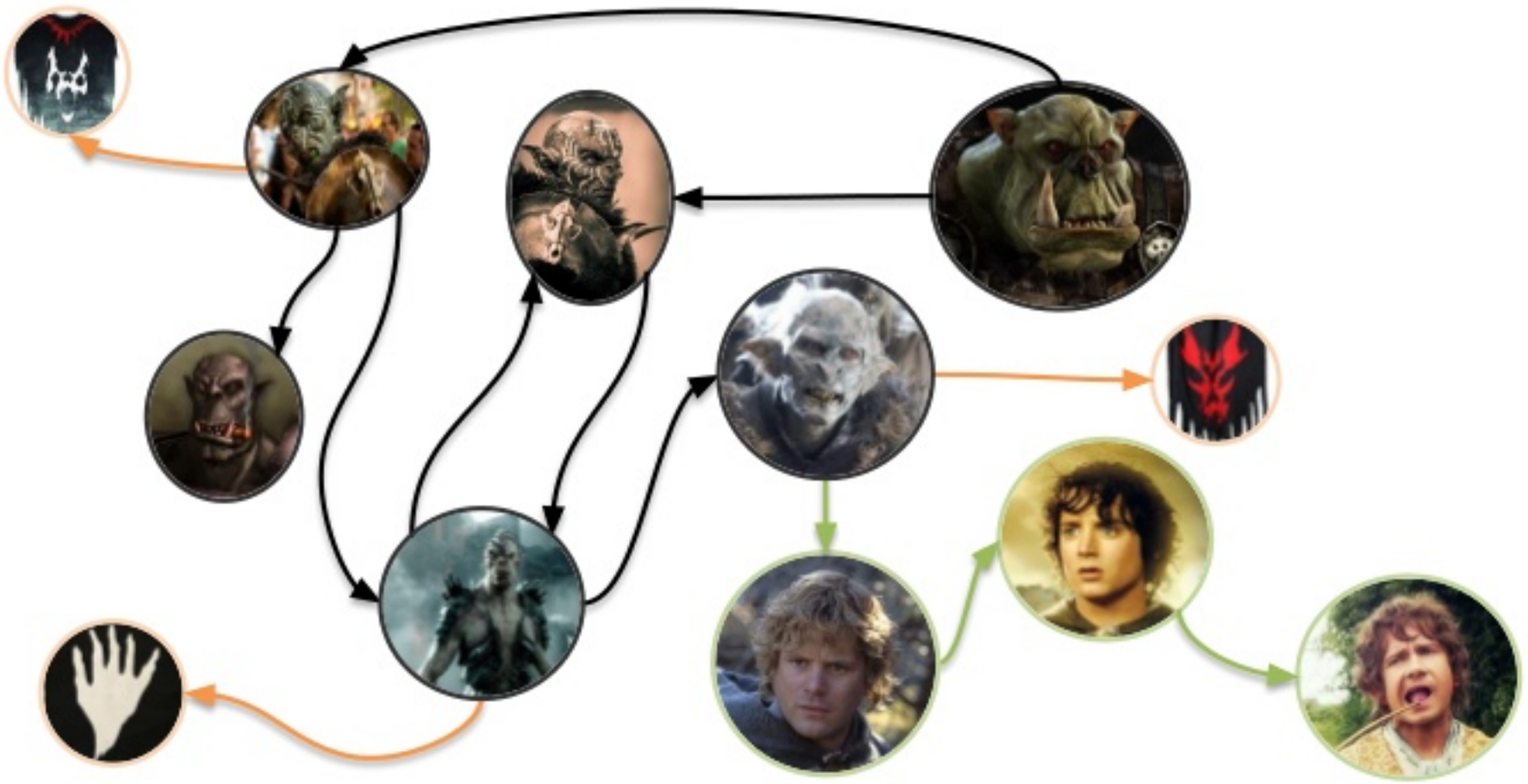
Flink Gelly

```java
/**
 * Traverse the Graph and find all nodes that are connected via "hates"-edges within 2 hops
 */
public class GellyTraversal {
  public static void main(String[] args) throws Exception {

    ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
    DataSet<Vertex<Long, Tuple2<Set<String>, Map<String, String>>>> vertices = env.fromElements();
    DataSet<Edge<Long, Tuple3<Long, String, Map<String, String>>>> edges = env.fromElements();

    Graph<Long, Tuple2<Set<String>, Map<String, String>>,Tuple3<Long, String, Map<String, String>>> g
      = Graph.fromDataSet(vertices, edges, env);

    Graph<Long, Set<Long>, Tuple3<Long, String, Map<String, String>>> inputGraph =
      g.mapVertices(v -> new HashSet<>());

    Graph<Long, Set<Long>, Tuple3<Long, String, Map<String, String>>> withNeighbours =
      inputGraph.runVertexCentricIteration(
        new ComputeFunction<Long, Set<Long>, Tuple3<Long, String, Map<String, String>>, Set<Long>>() {
          @Override
          public void compute(Vertex<Long, Set<Long>> vertex, MessageIterator<Set<Long>> messages)
            throws Exception {

            Set<Long> neighbours = vertex.getValue();
            for(Set<Long> msg : messages) {
              neighbours.addAll(msg);
            }

            if(neighbours != vertex.getValue()) {
              setNewVertexValue(neighbours);
              Set<Long> neighboursWithSelf = Sets.newHashSet(neighbours);
              neighboursWithSelf.add(vertex.getId());
              for (Edge<Long, Tuple3<Long, String, Map<String, String>>> e : getEdges()) {
                neighbours.add(vertex.getId());
                if (e.getValue().f1.equals("hates")) {
                  sendMessageTo(e.getSource(), neighboursWithSelf);
                }
              }
            }
          }
        },
        new MessageCombiner<Long, Set<Long>>() {
          @Override
          public void combineMessages(MessageIterator<Set<Long>> messages) throws Exception {
            sendCombinedMessage(
              StreamSupport.stream(messages.spliterator(), parallel:false)
                .flatMap(Collection::stream)
                .collect(Collectors.toSet())
            );
          }
        },
        2
    );

    List<Tuple2<Long,Long>> results = withNeighbours.getVertices().flatMap(
      (FlatMapFunction<Vertex<Long, Set<Long>>, Tuple2<Long, Long>>) (vertex, collector) -> vertex
        .getValue()
        .stream()
        .map(neighbour -> Tuple2.of(vertex.getId(), neighbour))
        .forEach(collector::collect)).collect();

    System.out.println(results);
  }
}
```

```
Graph<Long, Set<Long>, Tuple3<Long, String, Map<String, String>>> withNeighbours =
  inputGraph.runVertexCentricIteration(
    new ComputeFunction<Long, Set<Long>, Tuple3<Long, String, Map<String, String>>, Set<Long>>() {
      @Override
      public void compute(Vertex<Long, Set<Long>> vertex, MessageIterator<Set<Long>> messages)
        throws Exception {

        Set<Long> neighbours = vertex.getValue();
        for(Set<Long> msg : messages) {
          neighbours.addAll(msg);
        }

        if(neighbours != vertex.getValue()) {
          setNewVertexValue(neighbours);
          Set<Long> neighboursWithSelf = Sets.newHashSet(neighbours);
          neighboursWithSelf.add(vertex.getId());
          for (Edge<Long, Tuple3<Long, String, Map<String, String>>> e : getEdges()) {
            neighbours.add(vertex.getId());
            if (e.getValue().f1.equals("hates")) {
              sendMessageTo(e.getSource(), neighboursWithSelf);
            }
          }
        }
      }
    },
    new MessageCombiner<Long, Set<Long>>() {
      @Override
      public void combineMessages(MessageIterator<Set<Long>> messages) throws Exception {
        sendCombinedMessage(
          StreamSupport.stream(messages.spliterator(), parallel: false)
            .flatMap(Collection::stream)
            .collect(Collectors.toSet())
        );
      }
    },
    2
);
```







Sauron's Data Analyst: "Which two clan leaders hate each other and one of them knows Frodo over one to ten hops?"

Cypher

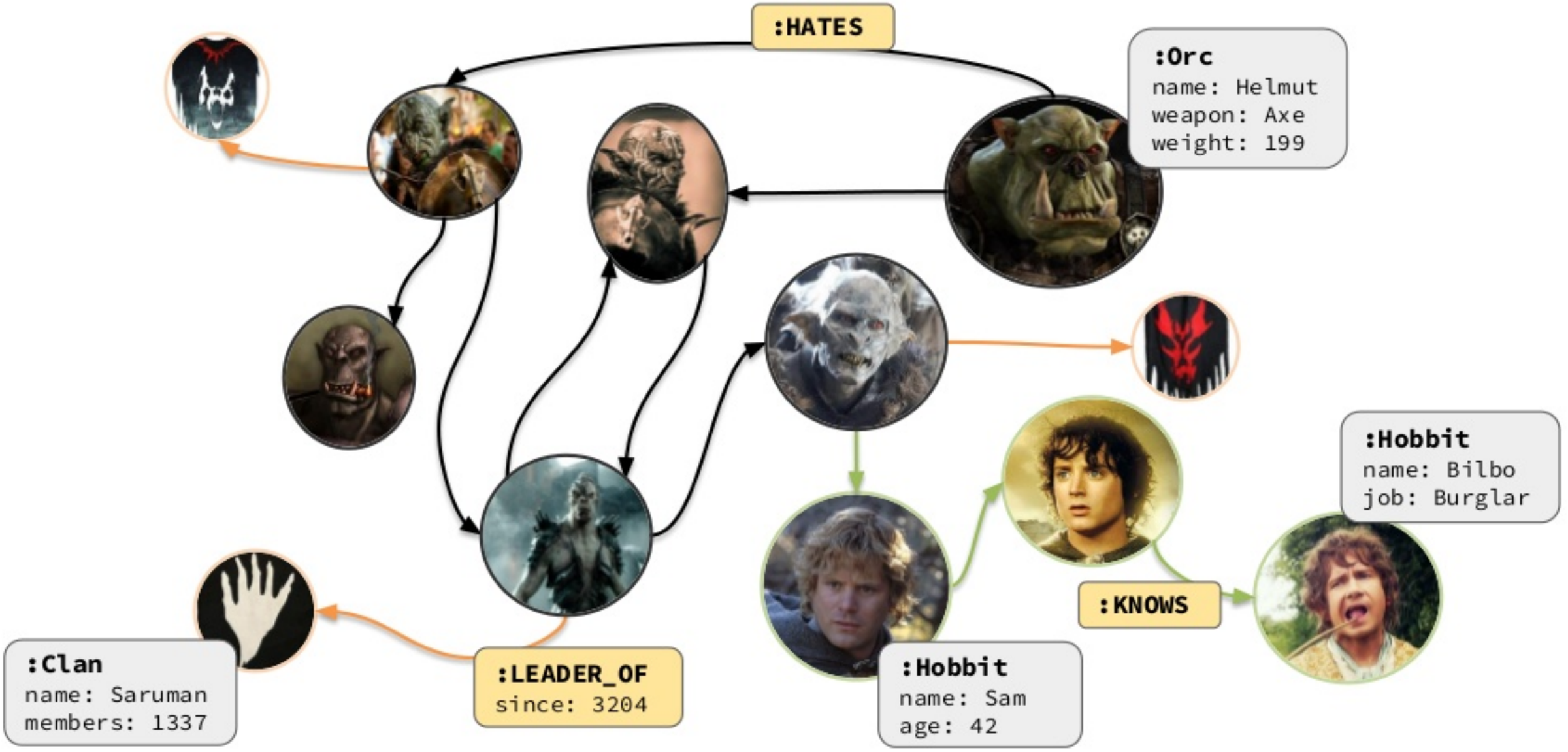
```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```

Flink Gelly

or any other non-declarative
graph processing system

CYPHER

Graph Fundamentals - Property Graphs



WHAT IS CYPHER?

Neo4j's declarative graph query language

- Used to insert, update and retrieve data from Neo4j
- Designed to be easily understood by people with SQL background
- Support for Pattern Matching, Filtering, Aggregation, Projection
- Results are (multidimensional) Tables

Specified, maintained and extended in the **openCypher** project
by several academic and industry contributors.

http://www.opencypher.org/

Cypher Query Syntax

Find all vertices with label *Clan* and assign them to *c1*

Traverse incoming edges of type *LEADER_OF*

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```

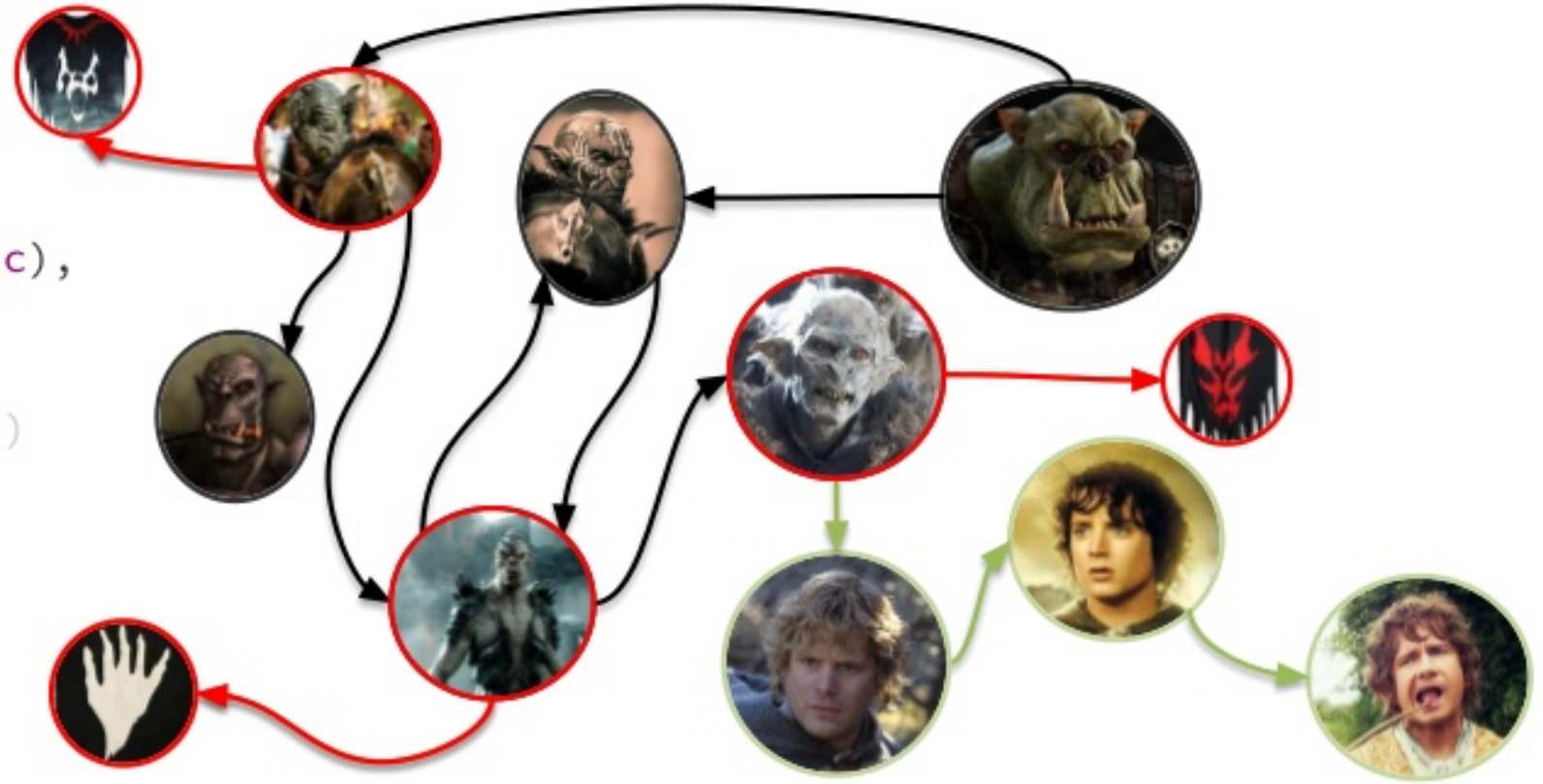
Describes the pattern that should be matched

Filters the match results

Specifies which fields to return

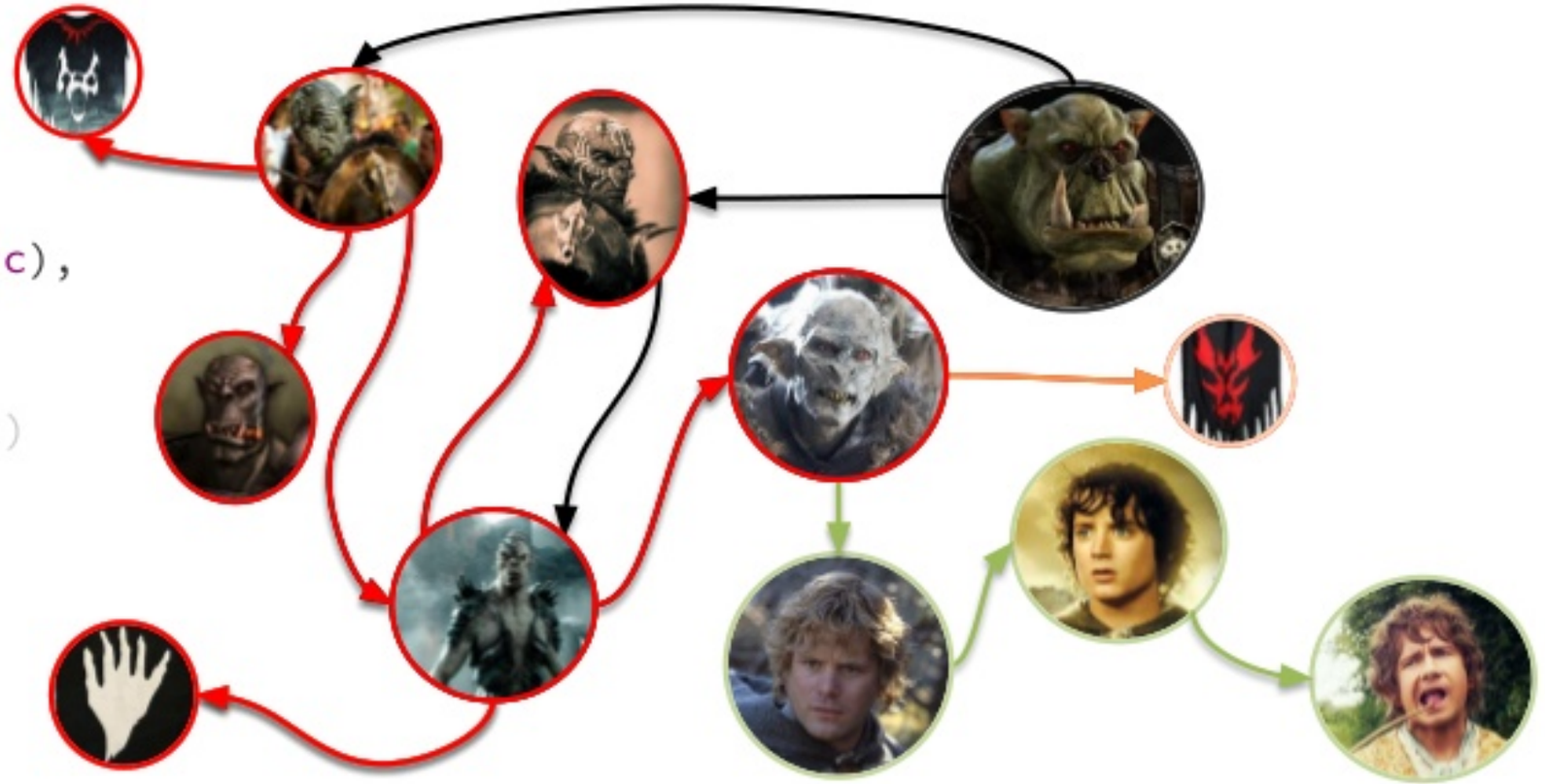
CYPHER QUERY SYNTAX

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```



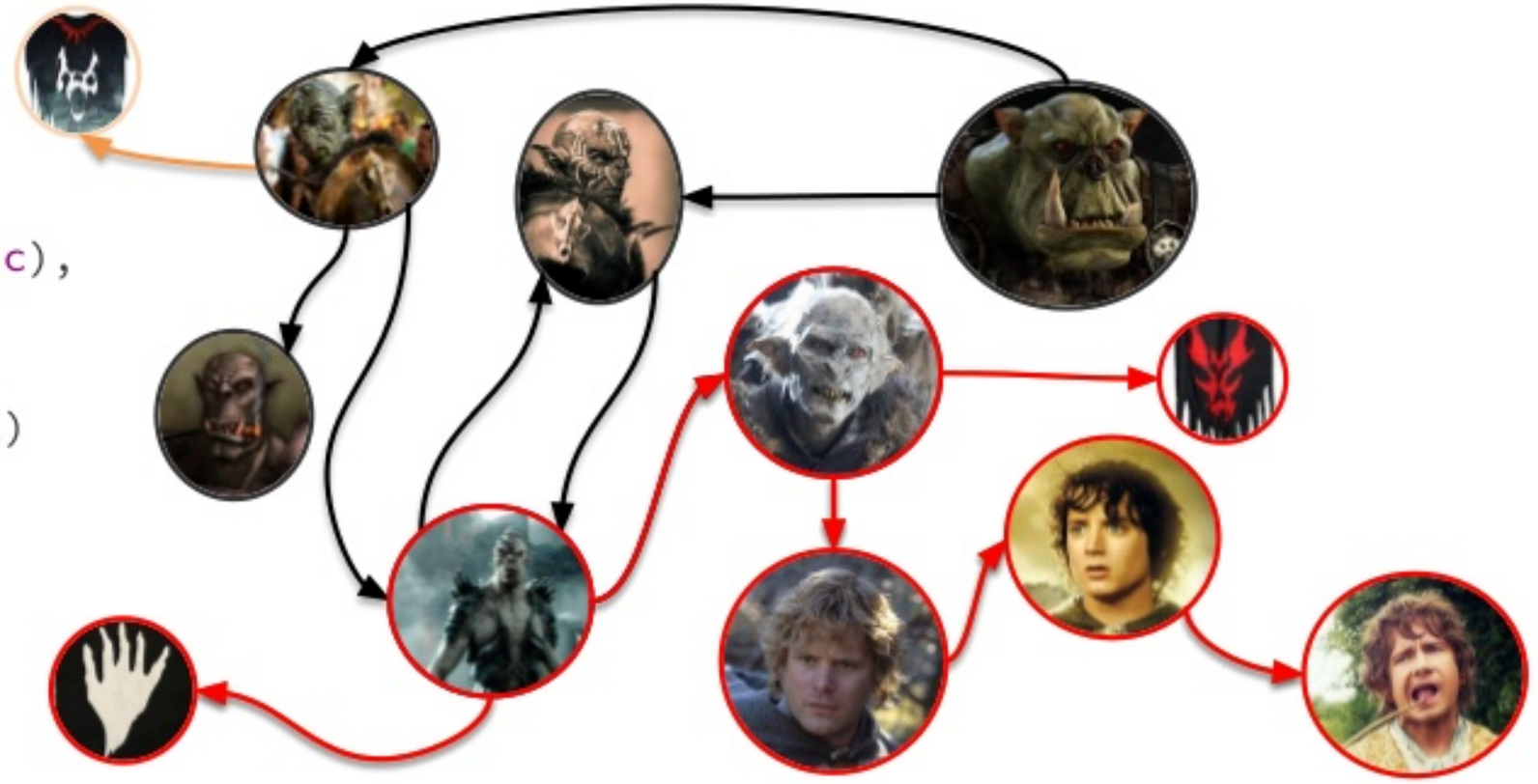
CYPHER QUERY SYNTAX

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```



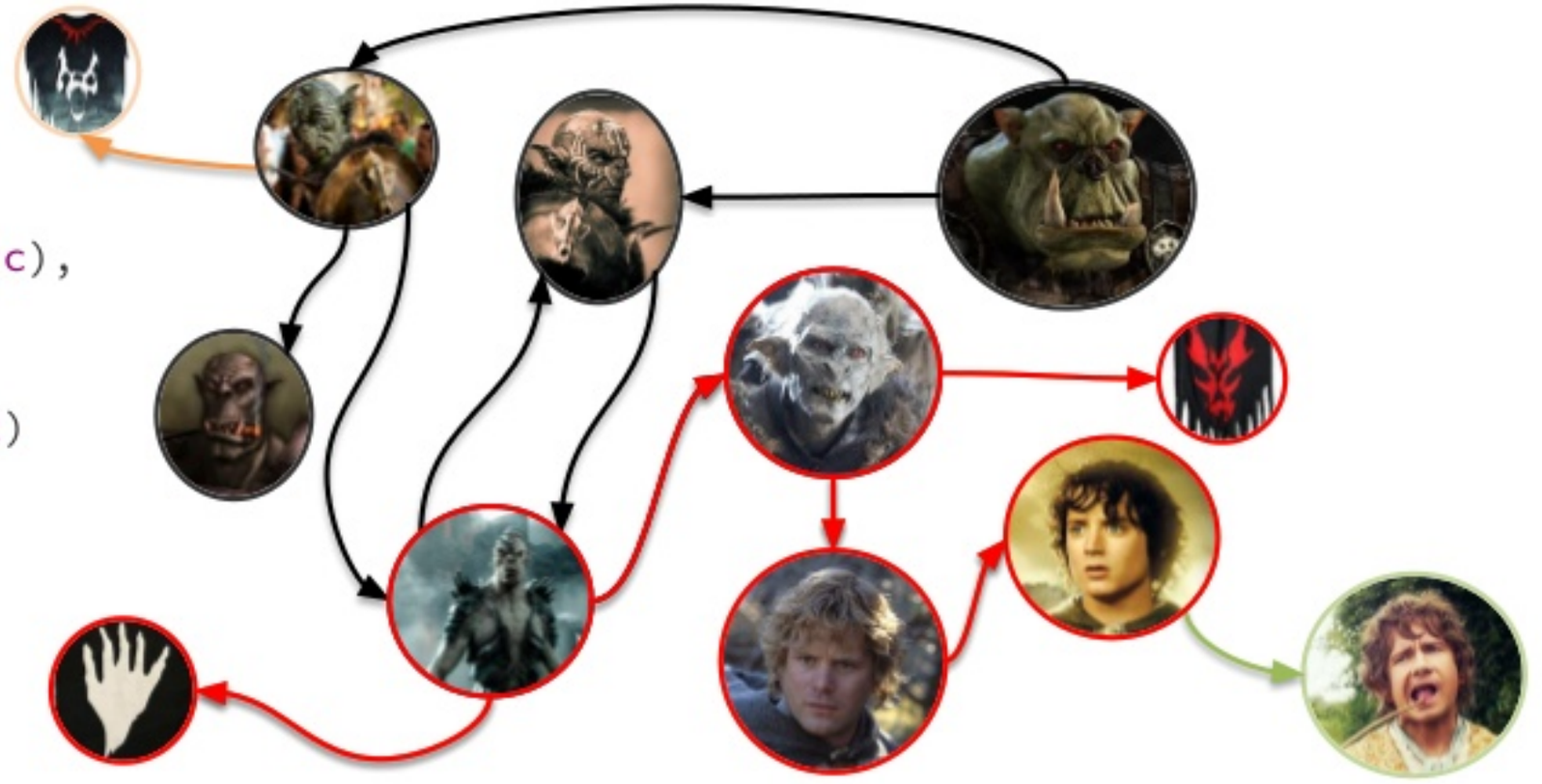
CYPHER QUERY SYNTAX

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```



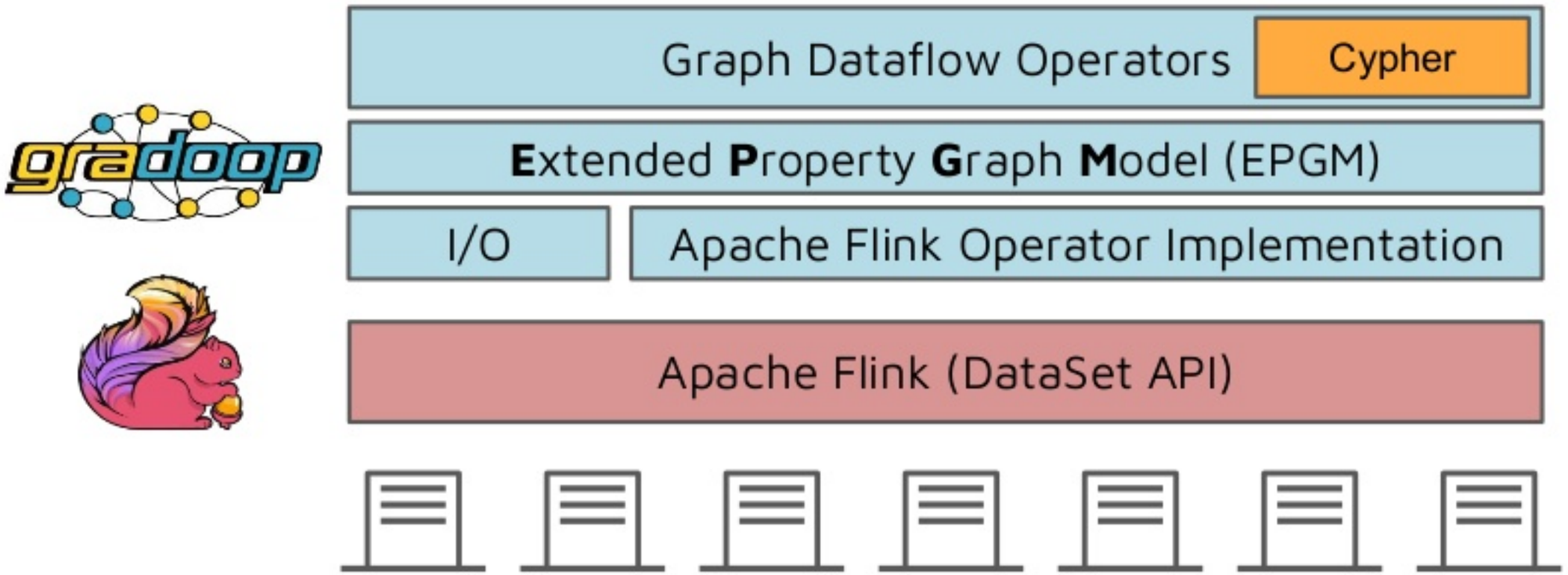
CYPHER QUERY SYNTAX

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```



Cypher Operator in Gradoop

„An open-source **graph dataflow framework** for **declarative analytics** of **heterogeneous** graph data."

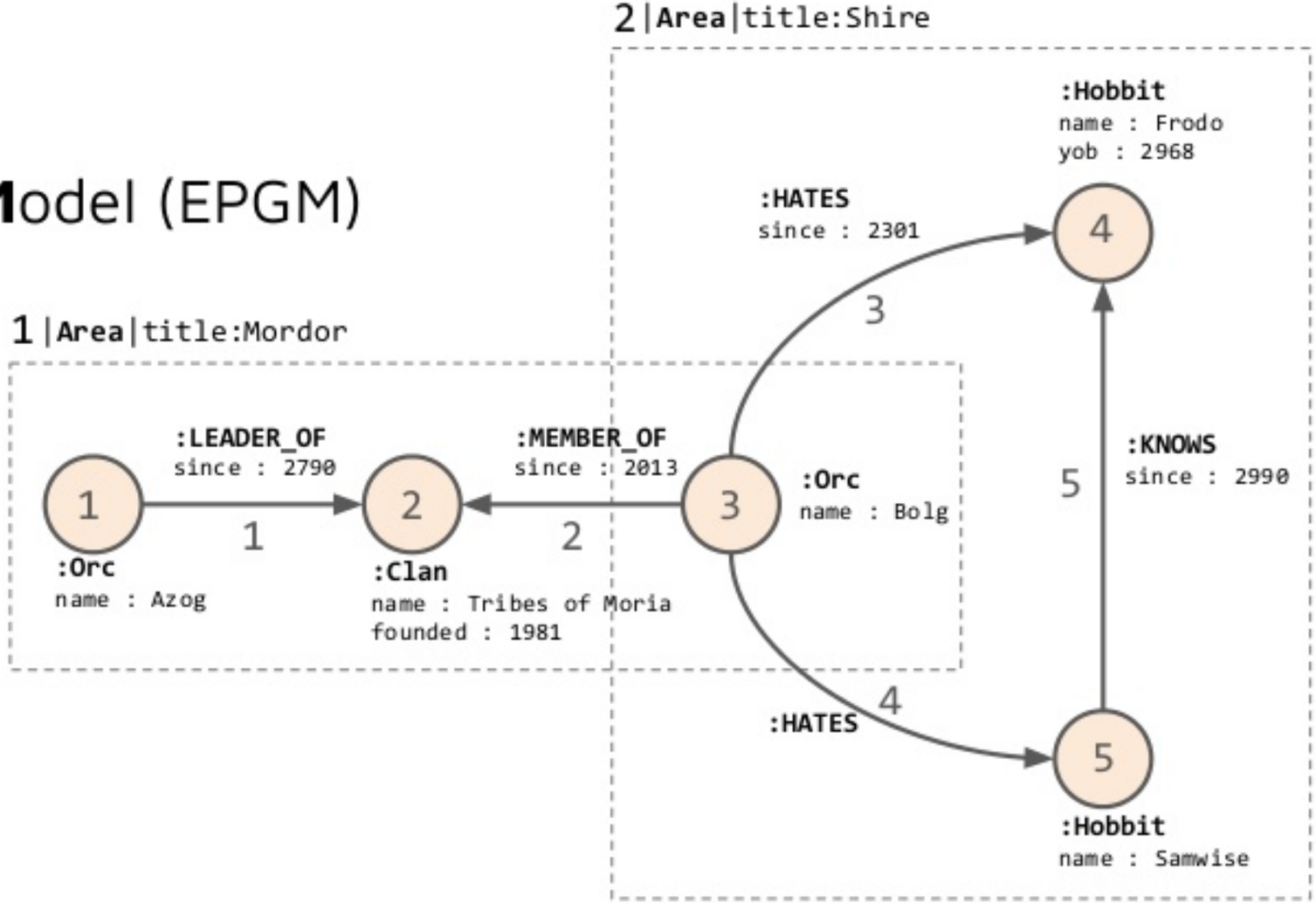


Extended **P**roperty **G**raph **M**odel (EPGM)

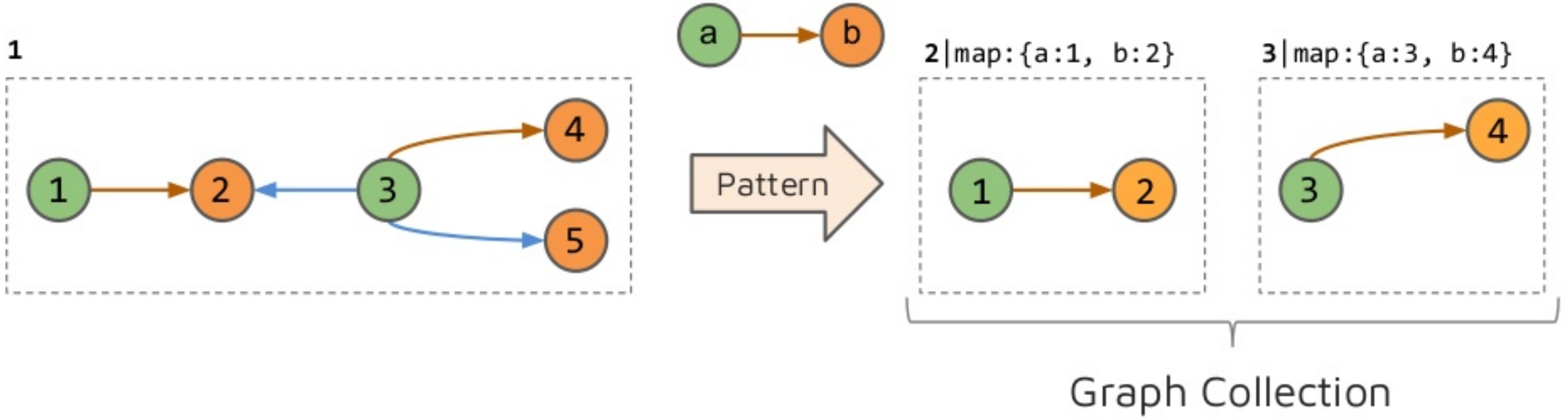
= Property Graph Model

+ Logical Graphs

+ Graph Transformations
  - Subgraph
  - Aggregation
  - Transformation
  - Grouping
  - **Cypher**
  - ...

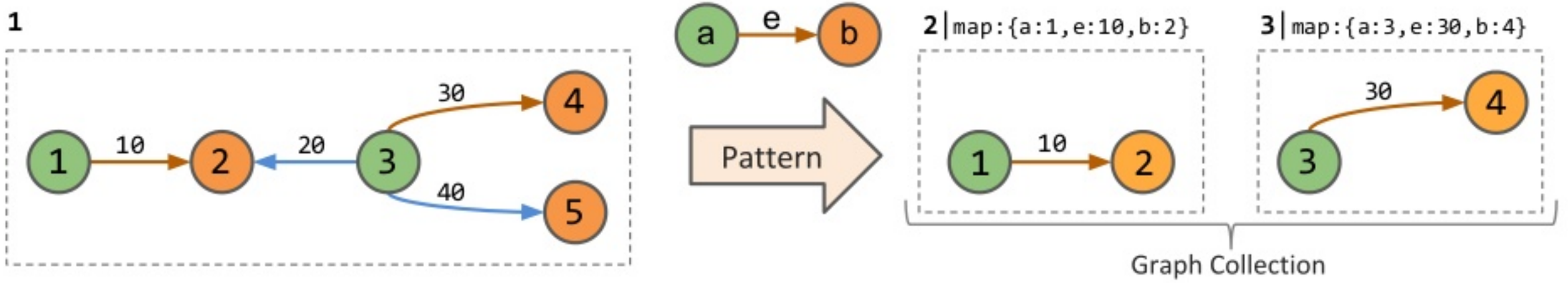


CYPHER OPERATOR




```
LogicalGraph graph1 = new CSVDataSource("hdfs:///path/to/graph", conf).getLogicalGraph();
String pattern = "MATCH (a:Green)-[:orange]->(b:Orange)";
GraphCollection collection = graph1.cypher(pattern);
```

INTERNAL REPRESENTATION



DataSet<Vertex>

id	label
1	Green
2	Orange
3	Green
4	Orange
5	Orange

DataSet<Edge>

id	src	trgt	label
10	1	2	ORANGE
20	3	2	BLUE
30	3	4	ORANGE
40	3	5	BLUE

DataSet<GraphHead>

id	label	properties
2	Green	map:{a:1, e:10, b:2}
3	Orange	map:{a:3, e:30, b:4}

DataSet<Vertex>

id	label	graphs
1	Green	{2}
2	Orange	{2}
3	Green	{3}
4	Orange	{4}

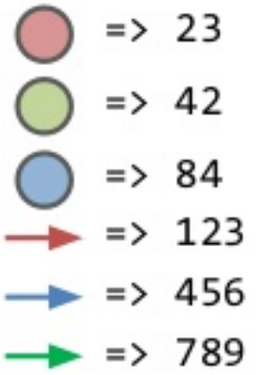
DataSet<Edge>

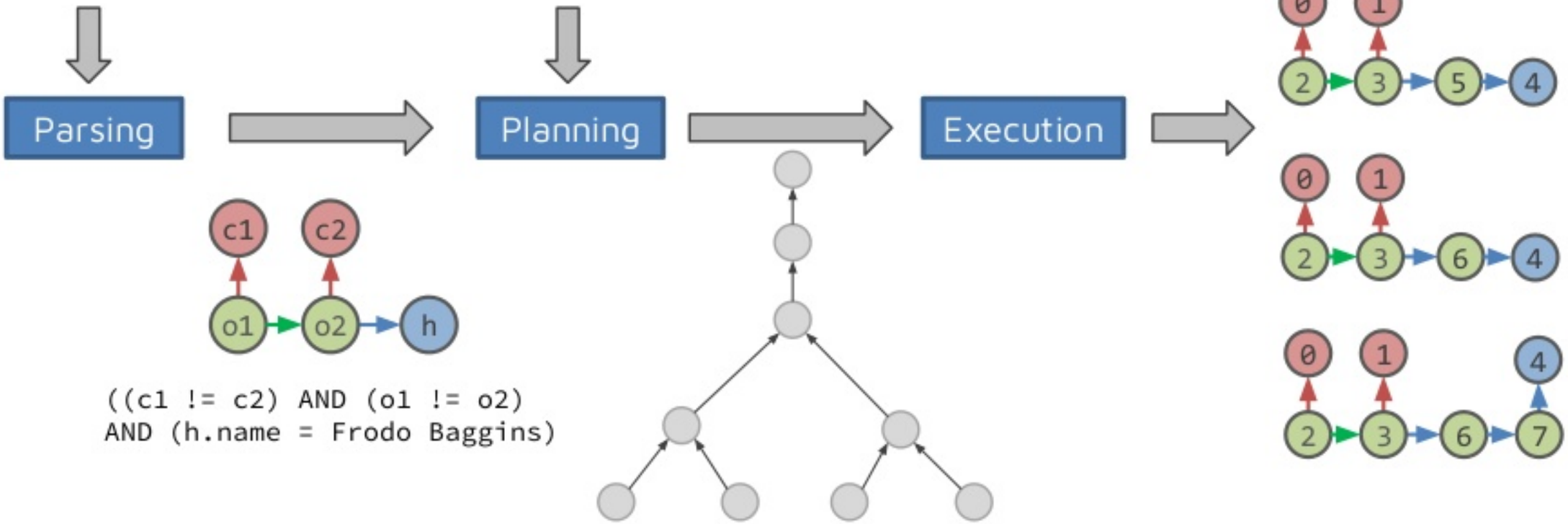
id	src	trgt	label	graphs
10	1	2	ORANGE	{2}
30	3	4	ORANGE	{3}

CYPHER ENGINE

QUERY OVERVIEW

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```





Parsing and Query Rewriting

```
MATCH (o1:Orc)-[:KNOWS]->(h:Hobbit)
WHERE (o1.weapon = "Axe" AND o1.weight > h.weight)
    OR o1.weapon = "Sword"
```



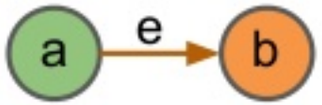
```
(o1.weapon = "Axe" AND o1.weight > h.weight)
          OR o1.weapon = "Sword"
```

Query Graph

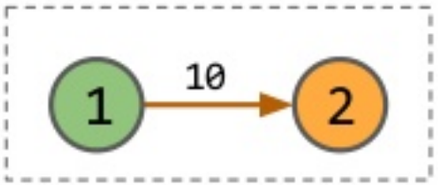

```
(o1.weapon = "Axe" OR o1.weapon = "Sword")
                    AND
(o1.weight > h.weight OR o1.weapon = "Sword")
```

Intermediate Result Representation - Embedding

Embedding = Mapping between Query graph and Input (Sub-)Graph



Embedding f
f(a) = 1
f(e) = 10
f(b) = 2



QUERY OPERATORS – FILTER AND PROJECT

Filter

WHERE h.name = 'Frodo'

name: Frodo
height: 1.22m
gender: male
city: Bag End

Project

[]

id	label	properties
1	Orc	{...}
2	Clan	{...}
3	Hobbit	{...}
...	...	...

$\sigma_{label='Hobbit' \wedge name='Frodo'}$

h.id	h.name	h.height	...
31	Frodo	1.22	...

$\pi_{h.id}$

h.id
31

DataSet<Vertex> → DataSet<Embedding>

vertices.**flatMap**(FilterAndProject)

Query operators – Join Embeddings

JoinEmbeddings

```
Left:  (c1:Clan)<-[:HAS_LEADER]-(o1:Orc)
Right: (o1:Orc)-[:HATES]->(o2.Orc)
```

c.id	_e1.id	o1.id
51	11	2
52	12	3
...	...	...

o1.id	_e2.id	o2.id
2	13	5
3	14	3
...	...	...

$$L \bowtie_{o1.id} R$$

Combine

Check for distinctiveness

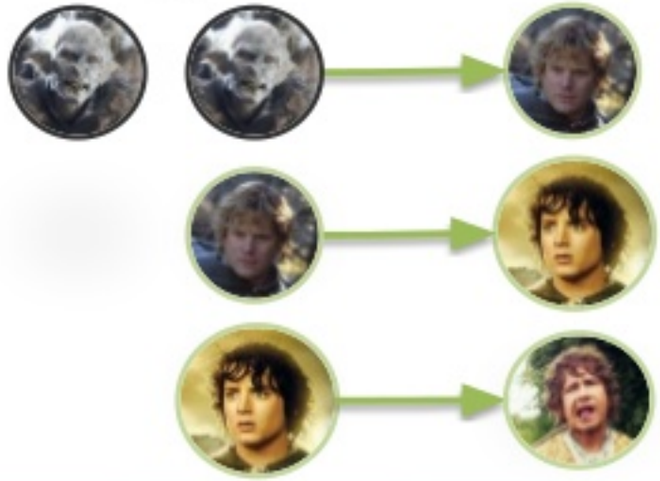
c.id	_e1.id	o1.id	_e2.id	o2.id
51	11	2	13	5
52	12	3	14	3
...	...	...	...	...

```
DataSet<Embedding> lhs
DataSet<Embedding> rhs
```

```
lhs.flatJoin(rhs).with(Combine)
```

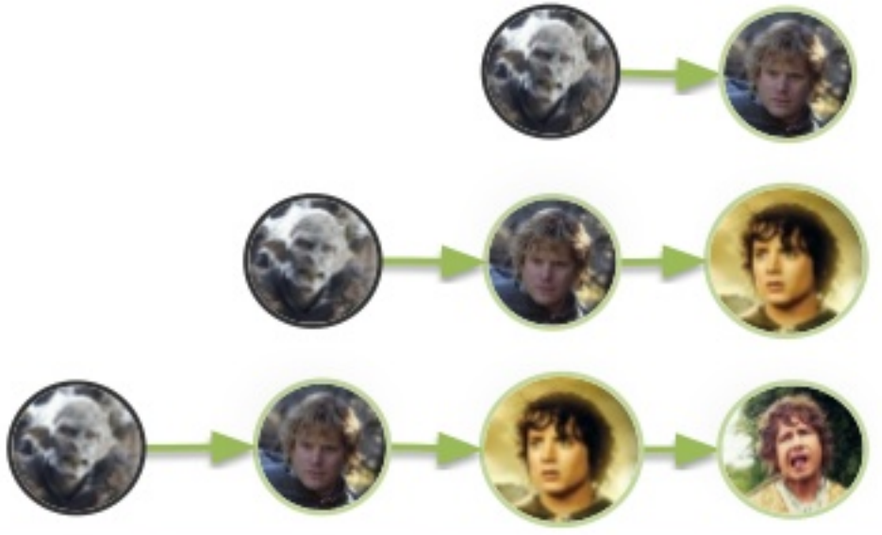
```
DataSet<Embedding>
```

Query operators – Expand Embeddings



ExpandEmbeddings

```
Left:  (o2:Orc)
Edges: (o2)-[:KNOWS*1..10]->(h:Hobbit)
```



o2.id
5

_e3.sid	_e3.id	_e3.tid
5	26	31
31	27	32
32	28	33

$$L \bowtie_{o2.id = e3.sid} E$$

$$E' \bowtie_{e.tid = e3.sid} E$$

Combine

Check for vertex/edge isomorphism

o2.id	_e3.id	h.id
3	[26]	31
3	[26,31,27]	32
3	[26,31,27,32,28]	33

```
DataSet<Embedding> lhs

DataSet<Embedding> rhs
```

BulkIteration(ws = lhs.**join**(rhs))

```
filteredPaths = ws.filter(filterByLength)
newPaths      = filteredPaths.flatJoin(rhs, combine)
nextWs        = ws.union(newPaths)
```

DataSet<Embedding>

QUERY OPERATORS – FILTER EMBEDDINGS

FilterEmbeddings

o1 != o2

o1.sid	_e2.id	o2.tid
2	13	5
3	14	3
...	...	...

$$\sigma_{o1.id \neq o2.id}$$

o1.sid	_e2.id	o2.tid
2	13	5
...	...	...

`DataSet<Embedding>` → `embeddings.filter(ByPredicate)` → `DataSet<Embedding>`

Cost-based greedy query planning

- Problem: Query can be computed in a factorial number of ways
  - Goal: Find a way (plan) with minimal / low computation costs

- Use statistics about the input graph
  - Vertex-/Edge counts by label, i.e., label distributions
  - Distinct value counts (source, target) by edge label
  - Property value distributions

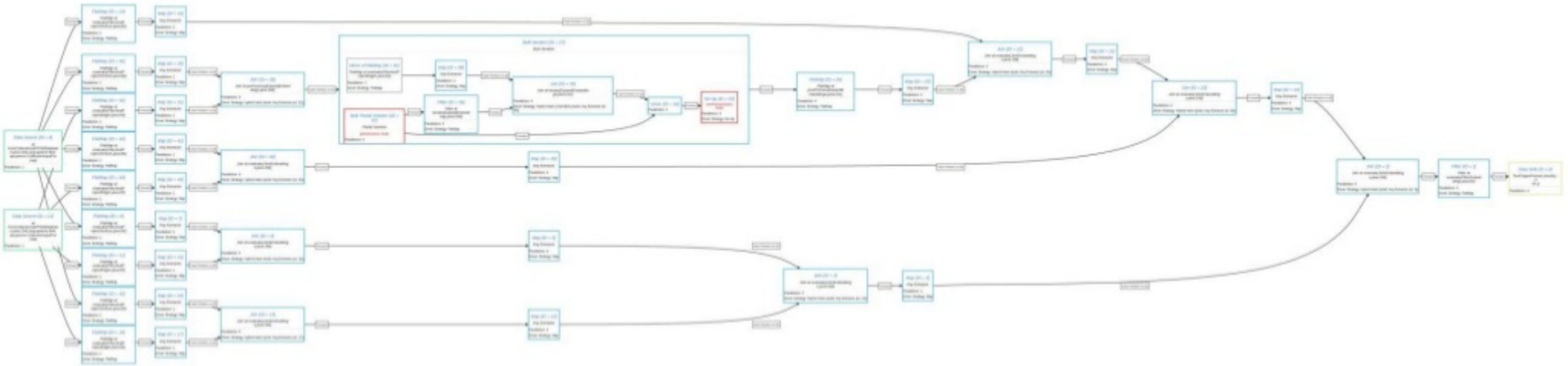
- Cost calculation for computing intermediate results
  - Primarily based on join result estimation
  - Filters and projections are evaluated as early as possible

- Planner iteratively builds a physical query plan
  - Greedy: picks plan with minimum cost with each iteration

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```

```
PlanTableEntry | type: GRAPH | all-vars: [...] | proc-vars: [...] | attr-vars: [] | est-card: 23 | prediates: () | Plan :
|-FilterEmbeddingsNode{filterPredicate=((c1 != c2) AND (o1 != o2))}
|.|-JoinEmbeddingsNode{joinVariables=[o2], vertexMorphism=H, edgeMorphism=I}
|.|.|-JoinEmbeddingsNode{joinVariables=[o1], vertexMorphism=H, edgeMorphism=I}
|.|.|.|-JoinEmbeddingsNode{joinVariables=[c1], vertexMorphism=H, edgeMorphism=I}
|.|.|.|.|-FilterAndProjectVerticesNode{vertexVar=c1, filterPredicate=((c1.label = Clan)), projectionKeys=[]}
|.|.|.|.|-FilterAndProjectEdgesNode{sourceVar='o1', edgeVar='_e0', targetVar='c1', filterPredicate=((_e0.label = leaderOf)), projectionKeys=[]}
|.|.|.|-JoinEmbeddingsNode{joinVariables=[o1], vertexMorphism=H, edgeMorphism=I}
|.|.|.|.|-FilterAndProjectVerticesNode{vertexVar=o1, filterPredicate=((o1.label = Orc)), projectionKeys=[]}
|.|.|.|.|-FilterAndProjectEdgesNode{sourceVar='o1', edgeVar='_e1', targetVar='o2', filterPredicate=((_e1.label = hates)), projectionKeys=[]}
|.|.|-JoinEmbeddingsNode{joinVariables=[o2], vertexMorphism=H, edgeMorphism=I}
|.|.|.|-JoinEmbeddingsNode{joinVariables=[h], vertexMorphism=H, edgeMorphism=I}
|.|.|.|.|-FilterAndProjectVerticesNode{vertexVar=h, filterPredicate=((h.label = Hobbit) AND (h.name = Frodo Baggins)), projectionKeys=[]}
|.|.|.|.|-ExpandEmbeddingsNode={startVar='o2', pathVar='_e3', endVar='h', lb=1, ub=10, direction=OUT, vertexMorphism=H, edgeMorphism=I}
|.|.|.|.|.|-FilterAndProjectVerticesNode{vertexVar=o2, filterPredicate=((o2.label = Orc)), projectionKeys=[]}
|.|.|.|.|.|-FilterAndProjectEdgesNode{sourceVar='o2', edgeVar='_e3', targetVar='h', filterPredicate=((_e3.label = knows)), projectionKeys=[]}
|.|.|.|-JoinEmbeddingsNode{joinVariables=[c2], vertexMorphism=H, edgeMorphism=I}
|.|.|.|.|-FilterAndProjectVerticesNode{vertexVar=c2, filterPredicate=((c2.label = Clan)), projectionKeys=[]}
|.|.|.|.|-FilterAndProjectEdgesNode{sourceVar='o2', edgeVar='_e2', targetVar='c2', filterPredicate=((_e2.label = leaderOf)), projectionKeys=[]}
```

```
MATCH (c1:Clan)<-[:LEADER_OF]-(o1:Orc),
      (o1)-[:HATES]->(o2:Orc),
      (o2)-[:LEADER_OF]->(c2:Clan),
      (o2)-[:KNOWS*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 AND o1 = o2)
      AND h.name = "Frodo"
RETURN o1.name, o2.name
```



DEMO

FUTURE WORK

- Optimizations
  - DP-Planner
  - Improve cost model (more statistics, Flink optimizer hints)
  - Reuse of intermediate results
- Support more Cypher features
  - e.g. Aggregation and Functions
- Introduce new Cypher features
  - e.g. regular path queries

Further reading / Contributing

Gradoop: http://www.gradoop.com

Demo: https://github.com/dbs-leipzig/gradoop_demo

Paper: https://event.cwi.nl/grades/2017/03-Junghanns.pdf

Neo4j: https://neo4j.com/

openCypher: http://www.openCypher.org

UNIVERSITÄT LEIPZIG

Q & A