LDBC *The graph & RDF benchmark reference*

9th TUC Meeting
SAP Headquarters
Walldorf, Germany

Distributed Graph Flows: Cypher on Flink and Gradoop

Max Kießling
University of Leipzig & Intern @ Neo Technology

UNIVERSITÄT LEIPZIG

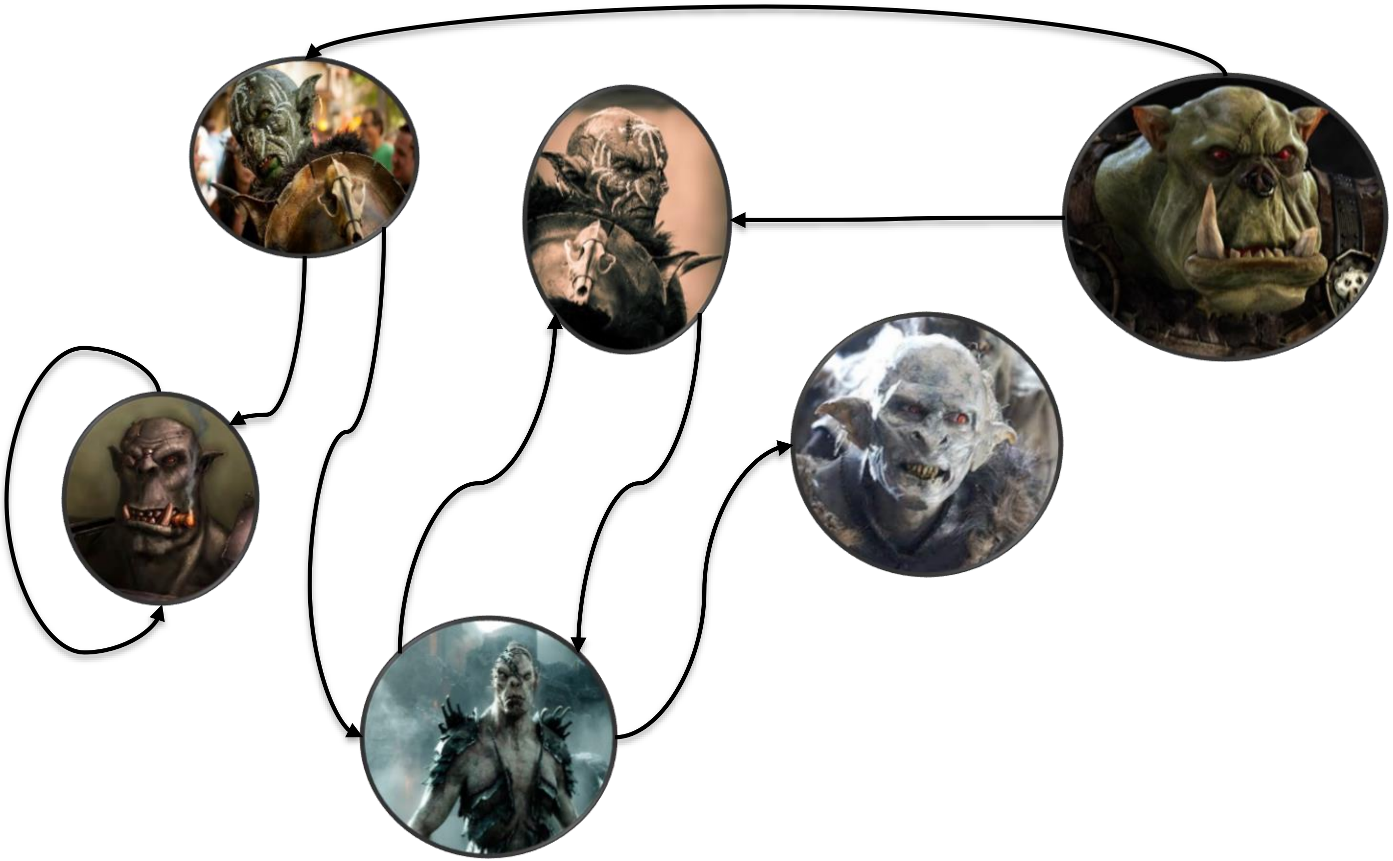
neo4j

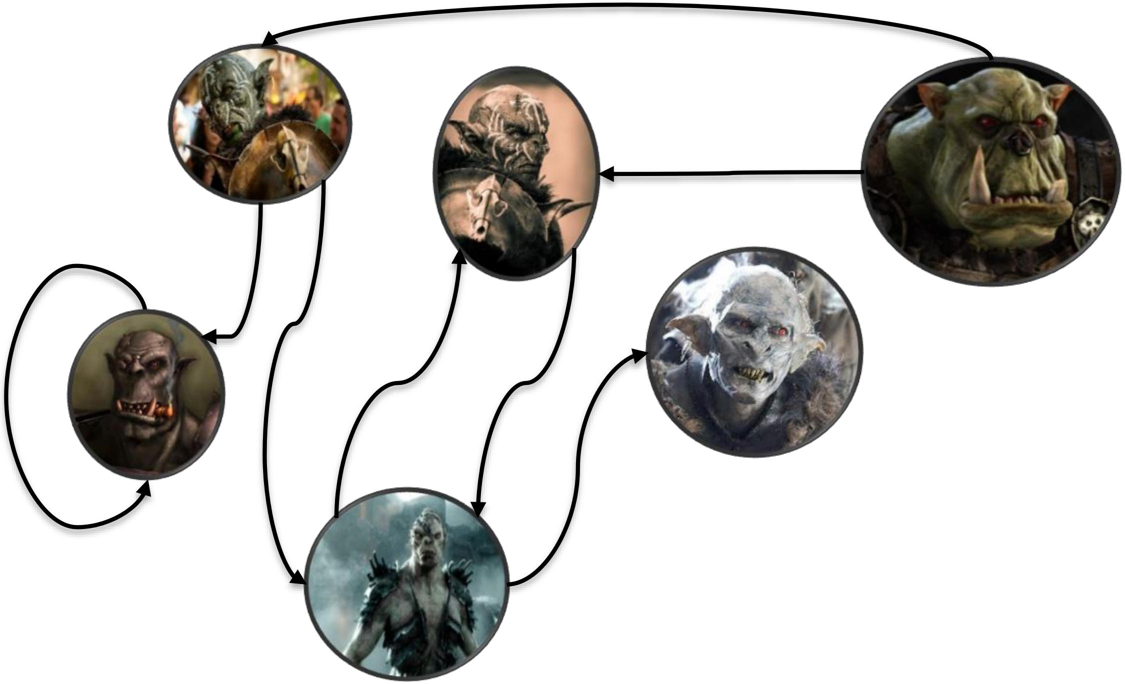
Motivation

Cypher on Gradoop

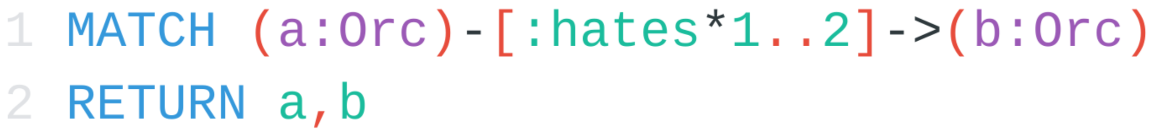
Conclusion

Motivation





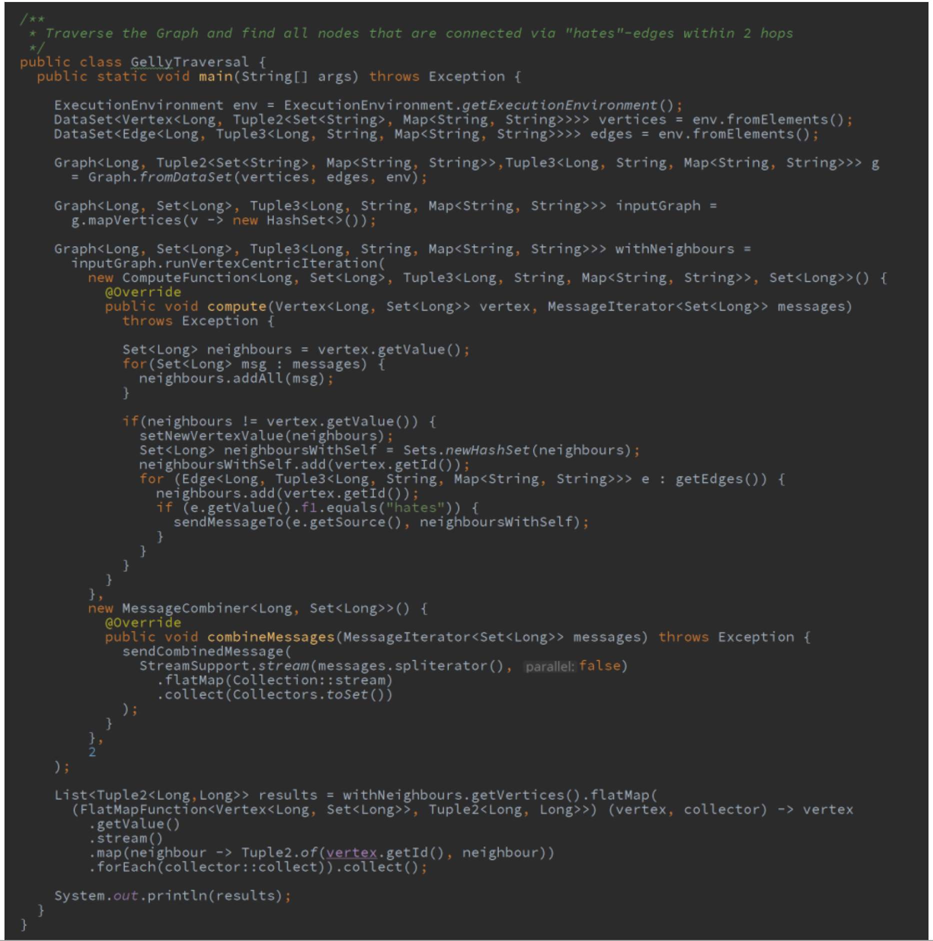
„Who are the closest enemies of each Orc?“



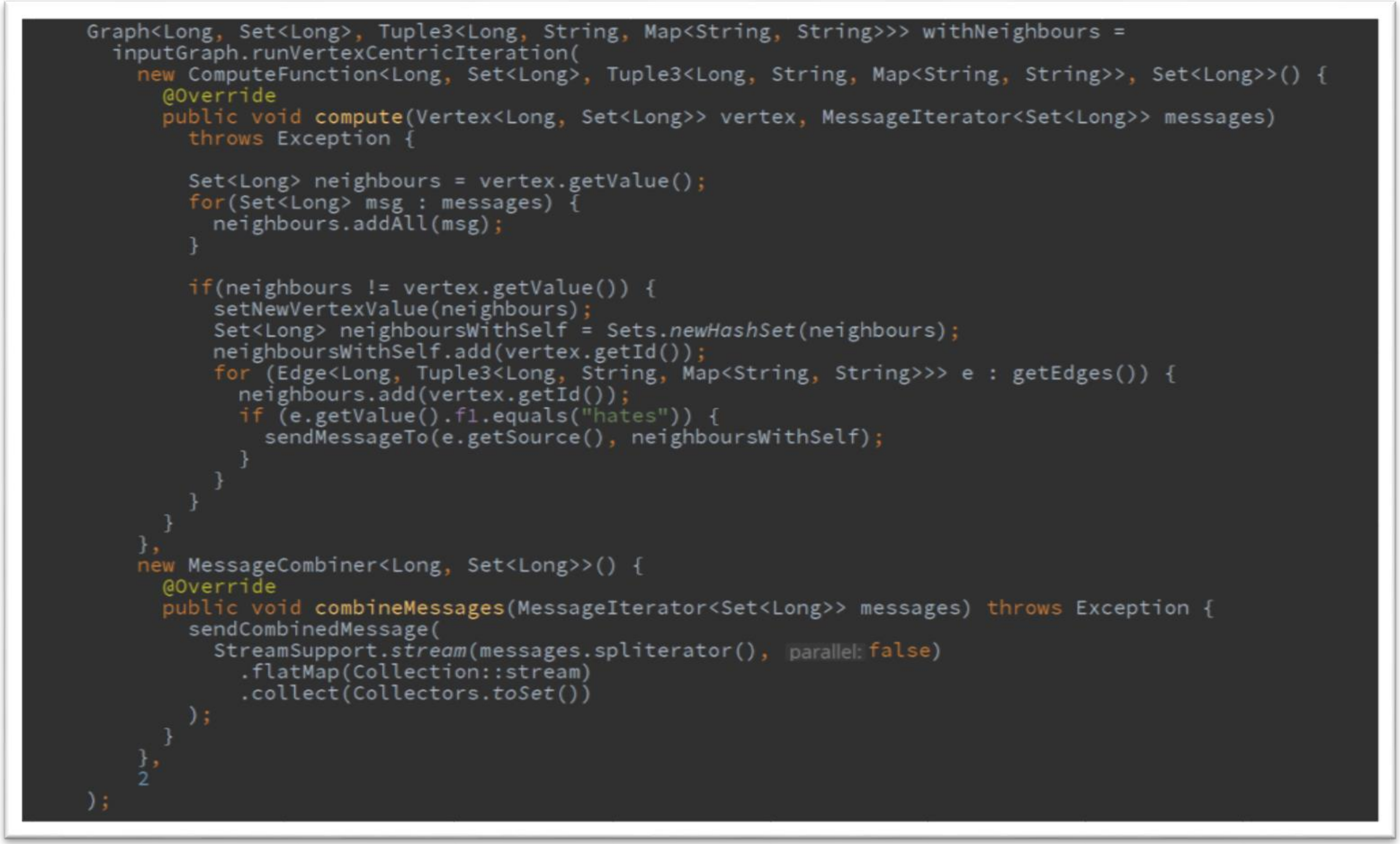
```
MATCH (a:Orc)-[:hates*1..2]->(b:Orc)
RETURN a,b
```

Cypher

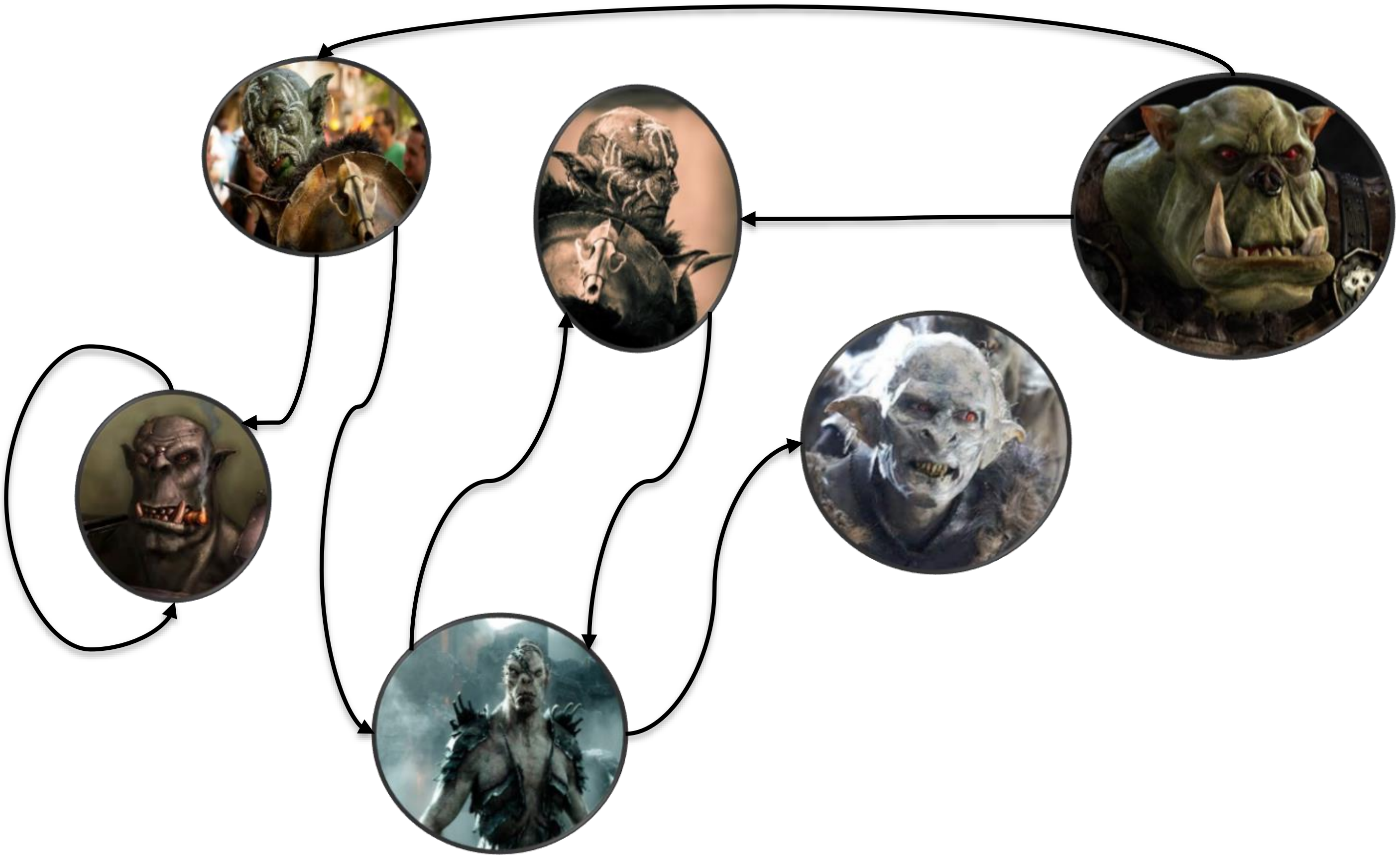
Flink Gelly

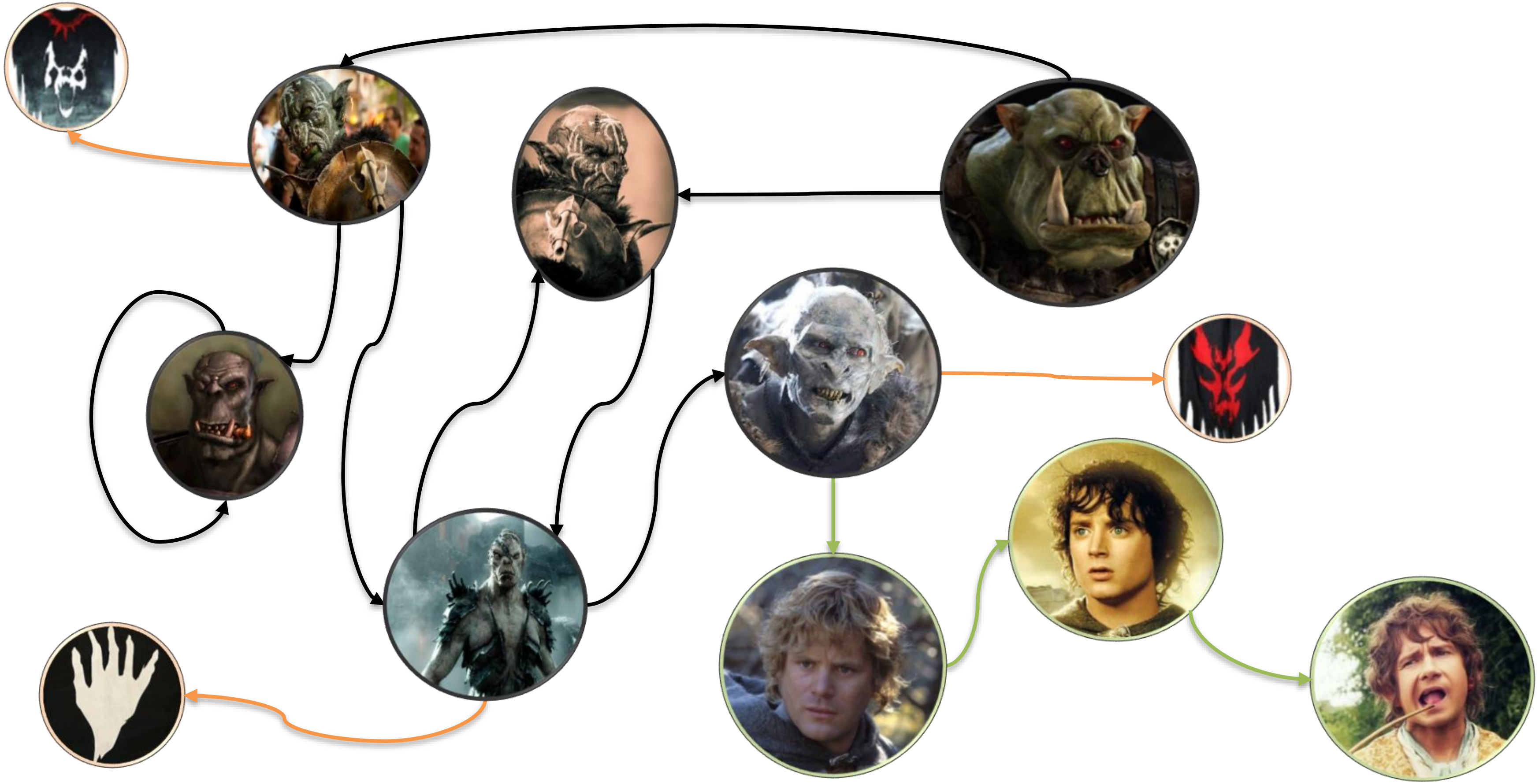


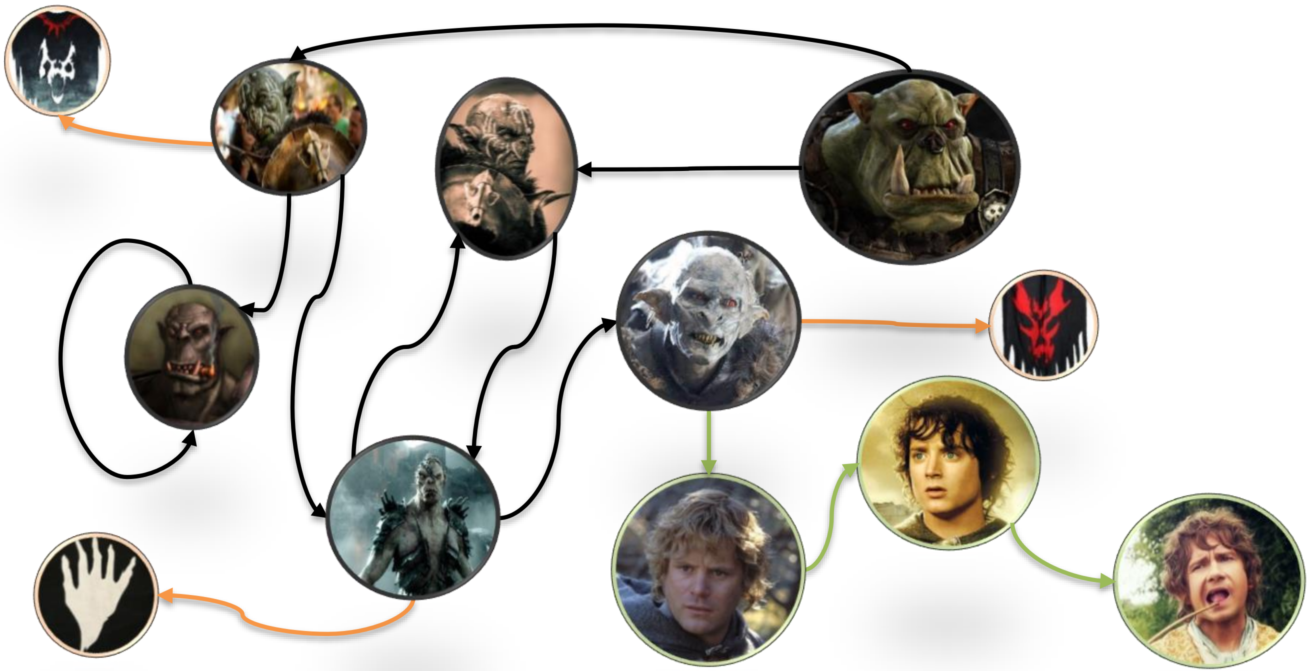
```java
/**
 * Traverse the Graph and find all nodes that are connected via "hates"-edges within 2 hops
 */
public class GellyTraversal {
  public static void main(String[] args) throws Exception {

    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
);
```

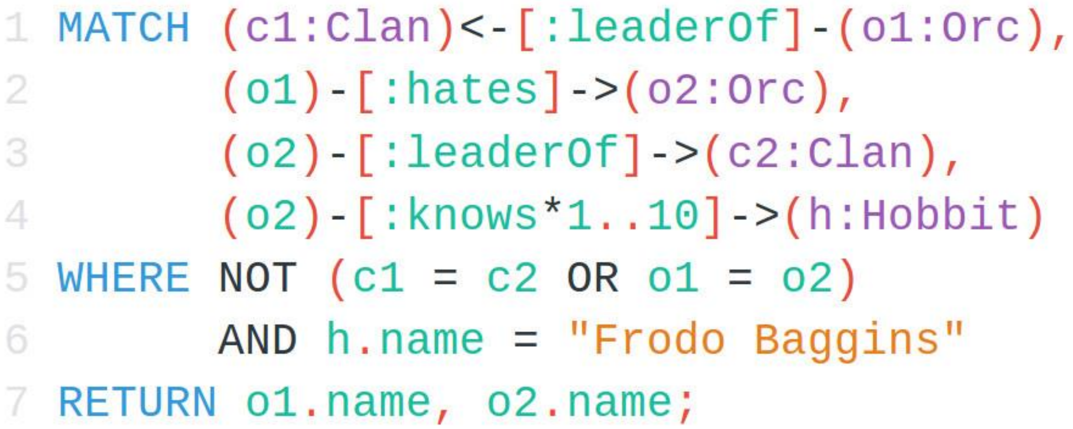




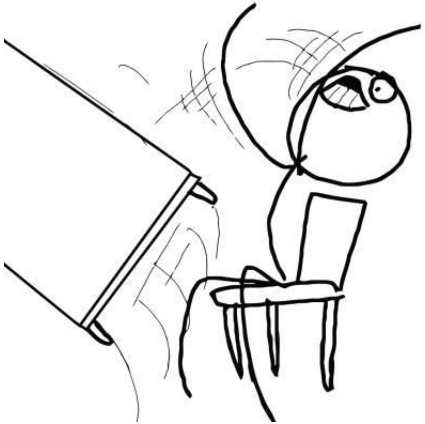


„Which two clan leaders hate each other and one of them knows Frodo over one to ten hops?“

Cypher



```
MATCH (c1:Clan)<-[:leaderOf]-(o1:Orc),
      (o1)-[:hates]->(o2:Orc),
      (o2)-[:leaderOf]->(c2:Clan),
      (o2)-[:knows*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 OR o1 = o2)
      AND h.name = "Frodo Baggins"
RETURN o1.name, o2.name;
```

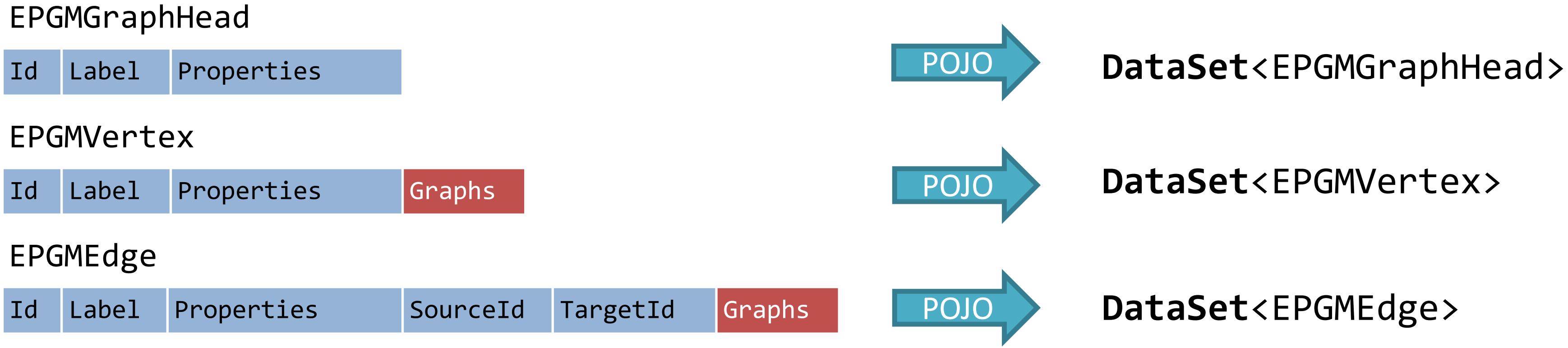


Flink Gelly
(or any other non-declarative graph processing system)

Cypher on Gradoop

Overview Implementation Flink Execution

EPGM Graph Representation



EPGMGraphHead

Id Label Properties

POJO

DataSet<EPGMGraphHead>

EPGMVertex

Id Label Properties Graphs

POJO

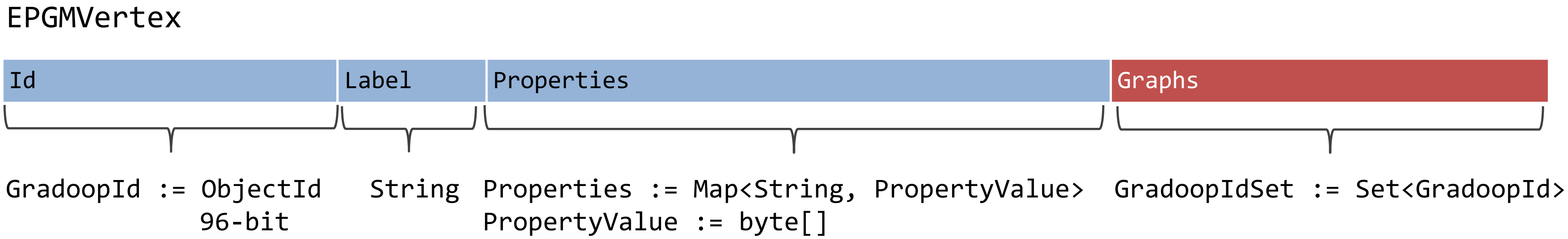
DataSet<EPGMVertex>

EPGMEdge

Id Label Properties SourceId TargetId Graphs

POJO

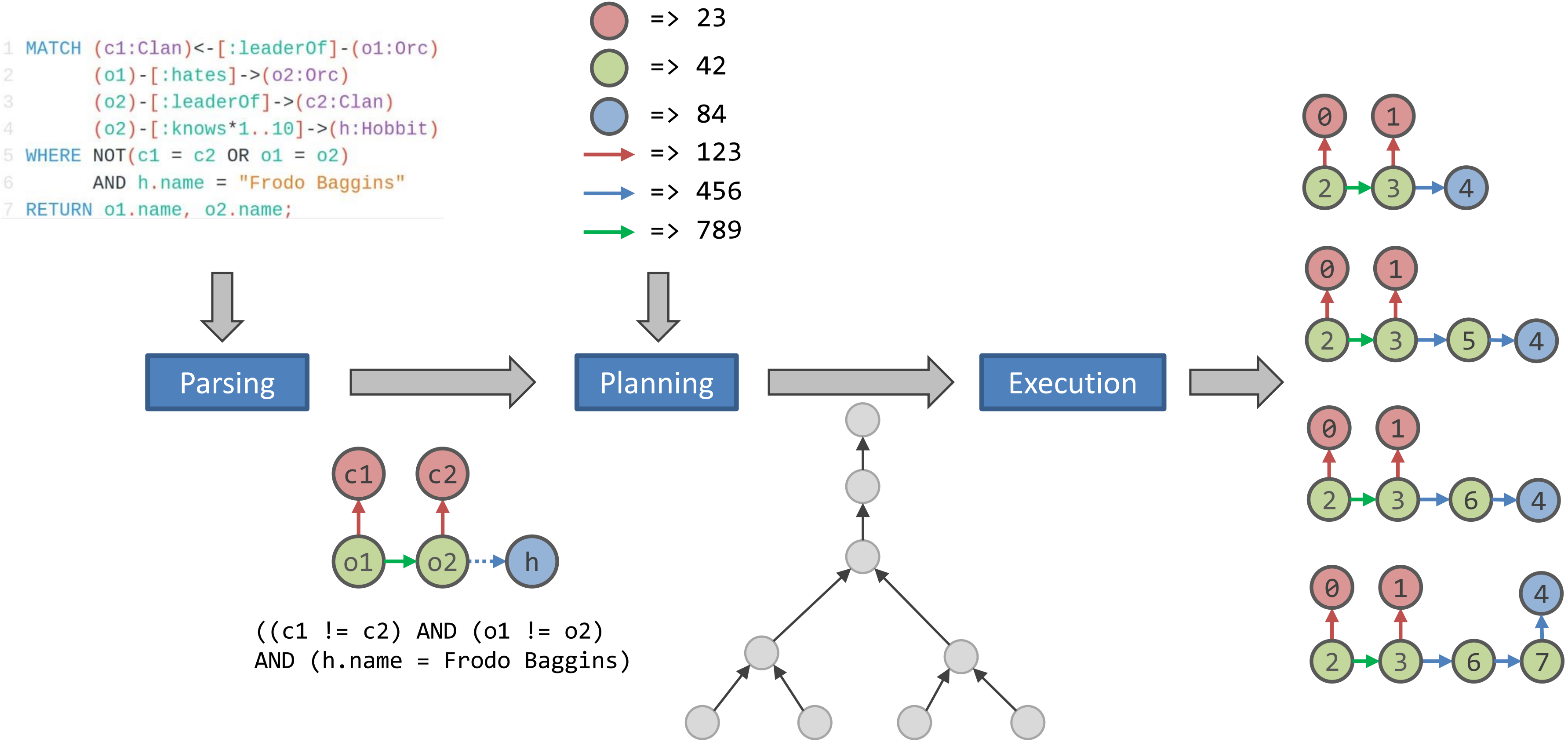
DataSet<EPGMEdge>



EPGMVertex

Id Label Properties Graphs

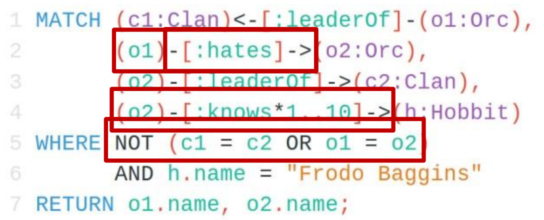
```
GradoopId := ObjectId      String  Properties := Map<String, PropertyValue>  GradoopIdSet := Set<GradoopId>
             96-bit                PropertyValue := byte[]
```



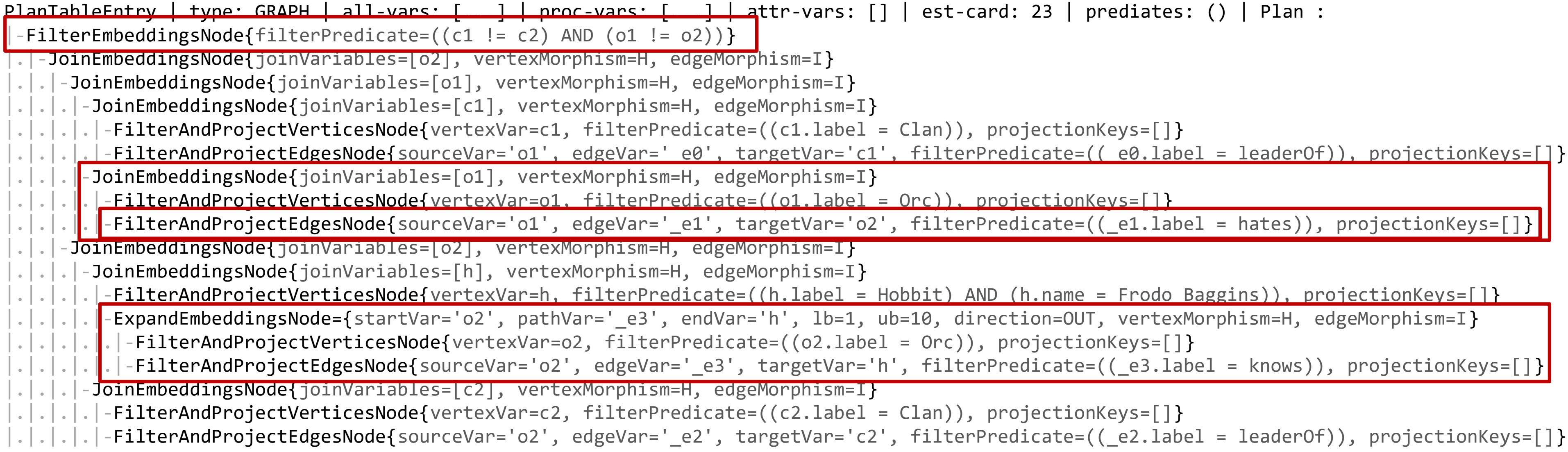
```
MATCH (c1:Clan)<-[:leaderOf]-(o1:Orc)
      (o1)-[:hates]->(o2:Orc)
      (o2)-[:leaderOf]->(c2:Clan)
      (o2)-[:knows*1..10]->(h:Hobbit)
WHERE NOT(c1 = c2 OR o1 = o2)
      AND h.name = "Frodo Baggins"
RETURN o1.name, o2.name;
```

🔴 => 23
🟢 => 42
🔵 => 84
→ => 123
→ => 456
→ => 789

((c1 != c2) AND (o1 != o2)
AND (h.name = Frodo Baggins)

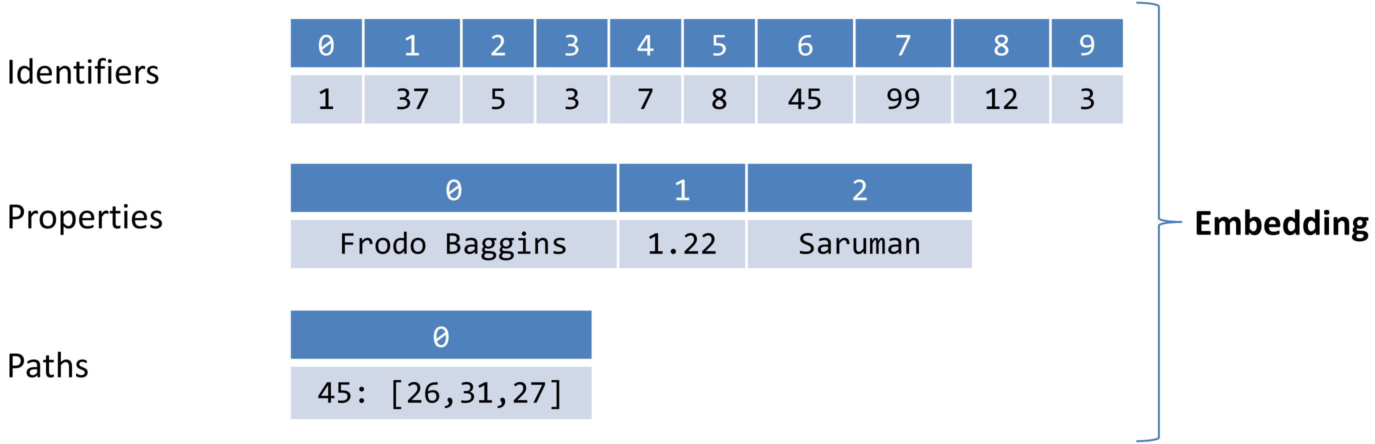


```
MATCH (c1:Clan)<-[:leaderOf]-(o1:Orc),
      (o1)-[:hates]->(o2:Orc),
      (o2)-[:leaderOf]->(c2:Clan),
      (o2)-[:knows*1..10]->(h:Hobbit)
WHERE NOT (c1 = c2 OR o1 = o2)
      AND h.name = "Frodo Baggins"
RETURN o1.name, o2.name;
```



```
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=[]}
```

Embedding - Data structure used for intermediate results



Identifiers

0	1	2	3	4	5	6	7	8	9
1	37	5	3	7	8	45	99	12	3

Properties

0	1	2
Frodo Baggins	1.22	Saruman

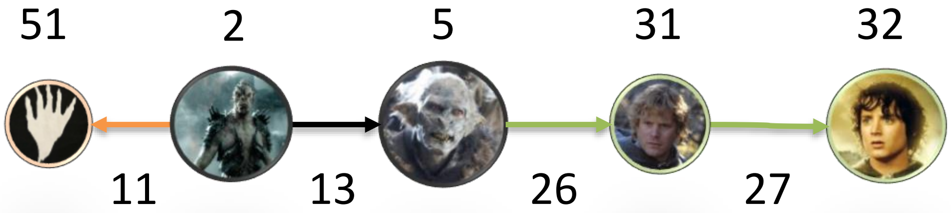
Paths

0
45: [26,31,27]

EmbeddingMetaData – Stores information about the embedding content

Mapping : Variable -> ID Column `{h: 0, e1: 1, o2: 5, ...}`

Mapping : Variable.Property -> Property Column `{h.name: 0, h.height: 1, c1.name: 2, ...}`



0	1	2	3	4	5	6
2	11	51	13	5	45	32

0	1
Azog	Gorbag

0
45: [26,31,27]

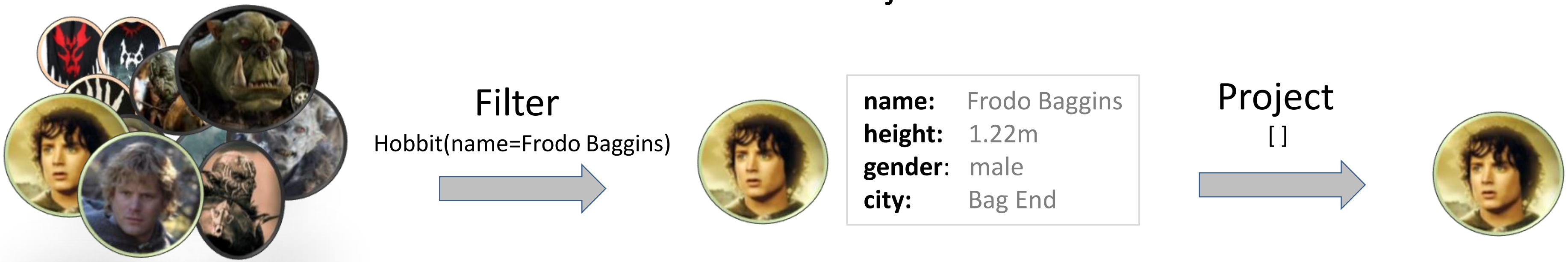
Entry Mapping

```
{o1: 0, e1: 1, c1: 2, e2: 3, o2: 4, e3: 5, h: 6}
```

Property Mapping

```
{o1.name: 0, o2.name: 1}
```

FilterAndProject



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

id	Properties
1	{…}
2	{…}
3	{…}
…	…

$$\sigma_{\substack{Label=Hobbit \\ \wedge name=Frodo}}(V)$$

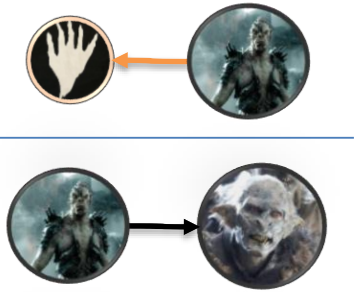
h.id	h.name	h.height	…
31	Frodo	1.22	…

$$\pi_{h.Id}(V')$$

h.id
32

`DataSet<Vertex>` → `DataSet<Embedding>`

FlatMap(Vertex -> Embedding)



JoinEmbeddings

```
Left: (c1:Clan)<-[:hasLeader]-(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

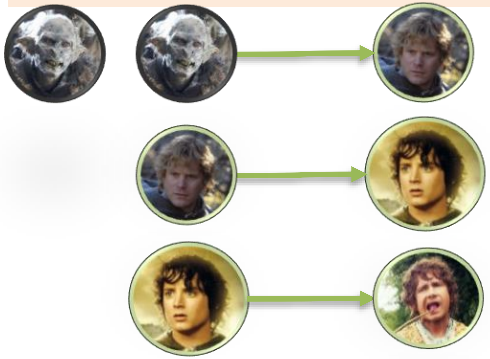
Check for vertex/edge isomorphism,
Remove duplicate entries

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

```
DataSet<Embedding>
DataSet<Embedding>
```

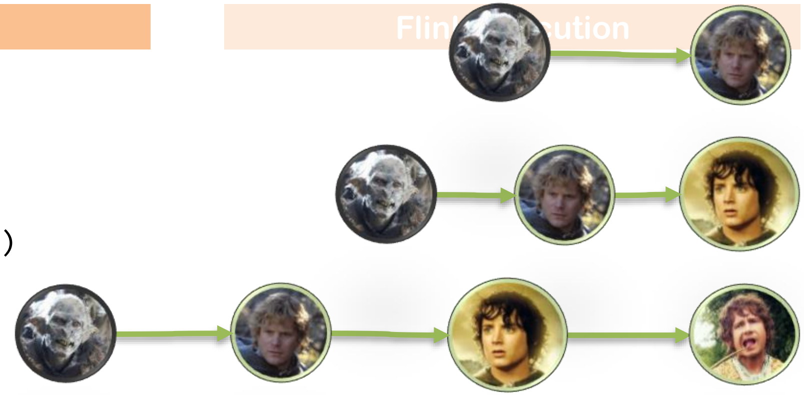
```
FlatJoin(lhs, rhs -> combine(lhs, rhs))
```

```
DataSet<Embedding>
```



ExpandEmbeddings

```
Left: (o2:Orc)
Edge: (o2)-[:knows*1..10]->(h)
```



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

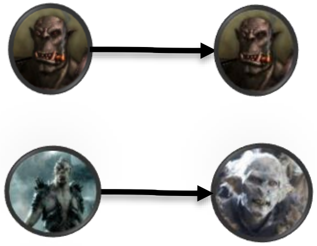
o2.id	_e3.id	h.id
3	[26]	31
3	[26,31,27]	32
3	[26,31,27,32,28]	33

BulkIteration

```
DataSet<Embedding>

DataSet<Embedding>
```

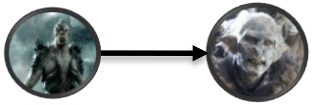
```
FlatJoin(lhs, rhs ->
 combine(lhs, rhs))
```

```
DataSet<Embedding>
```



FilterEmbeddings

o1 != o2



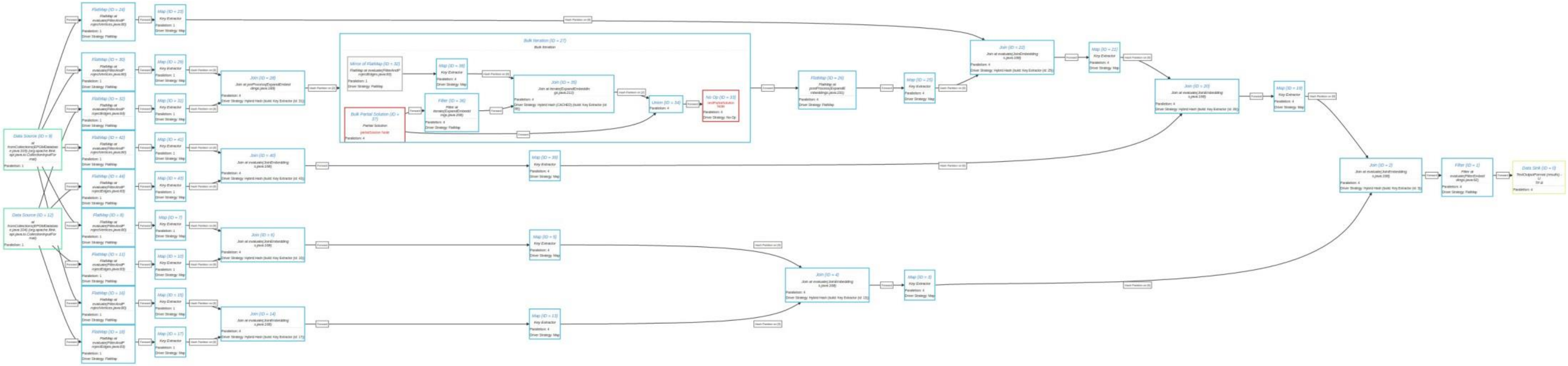
o1.id	_e2.id	o2.id
2	13	5
3	14	3
…	…	…

$$\sigma_{o1.id\ !=o2.id}(E)$$

o1.id	_e2.id	o2.id
2	13	5
…	…	…

`DataSet<Embedding>` → **`Filter`**`(embedding, predicate)` → `DataSet<Embedding>`

```
MATCH (c1:Clan)<-[:leaderOf]-(o1:Orc)
      (o1)-[:hates]->(o2:Orc)
      (o2)-[:leaderOf]->(c2:Clan)
      (o2)-[:knows*1..10]->(h:Hobbit)
WHERE NOT(c1 = c2 OR o1 = o2)
      AND h.name = "Frodo Baggins"
RETURN o1.name, o2.name;
```



Open Tasks and Future Work

- Implement Cypher Technology Compatibility KIT (TCK) integration tests
- Benchmarking
  - Implement and evaluate LDBC benchmarking queries
- Optimizations
  - DP-Planner
  - Improve cost model (more statistics, Flink optimizer hints)
  - Reuse of intermediate results
  - Consider graph partitioning
- Support more Cypher features
  - e.g. Aggregation and Functions
- Introduce new Cypher features
  - e.g. regular path queries

Summary

- Cypher on Gradoop
    - Covering many Cypher features (variable length paths, predicates)
    - Query execution engine incl. Greedy cost-based optimizer
    - Physical operators mapped to Flink transformations

Gradoop:	http://www.gradoop.com
Neo4j:	https://neo4j.com/
Apache Flink:	https://flink.apache.org/
openCypher:	http://www.opencypher.org/

Thank you!

UNIVERSITÄT LEIPZIG neo4j gradoop ScaDS DRESDEN LEIPZIG