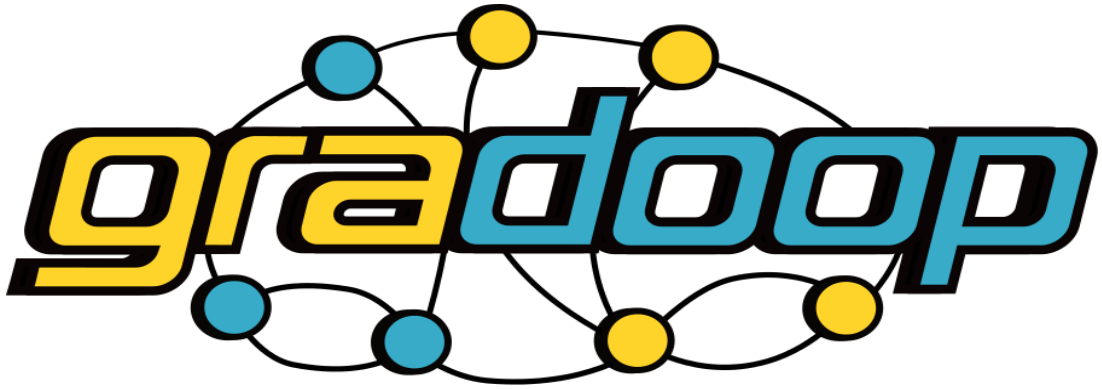
9th LDBC TUC Meeting
9-10 February 2017
Walldorf, Germany

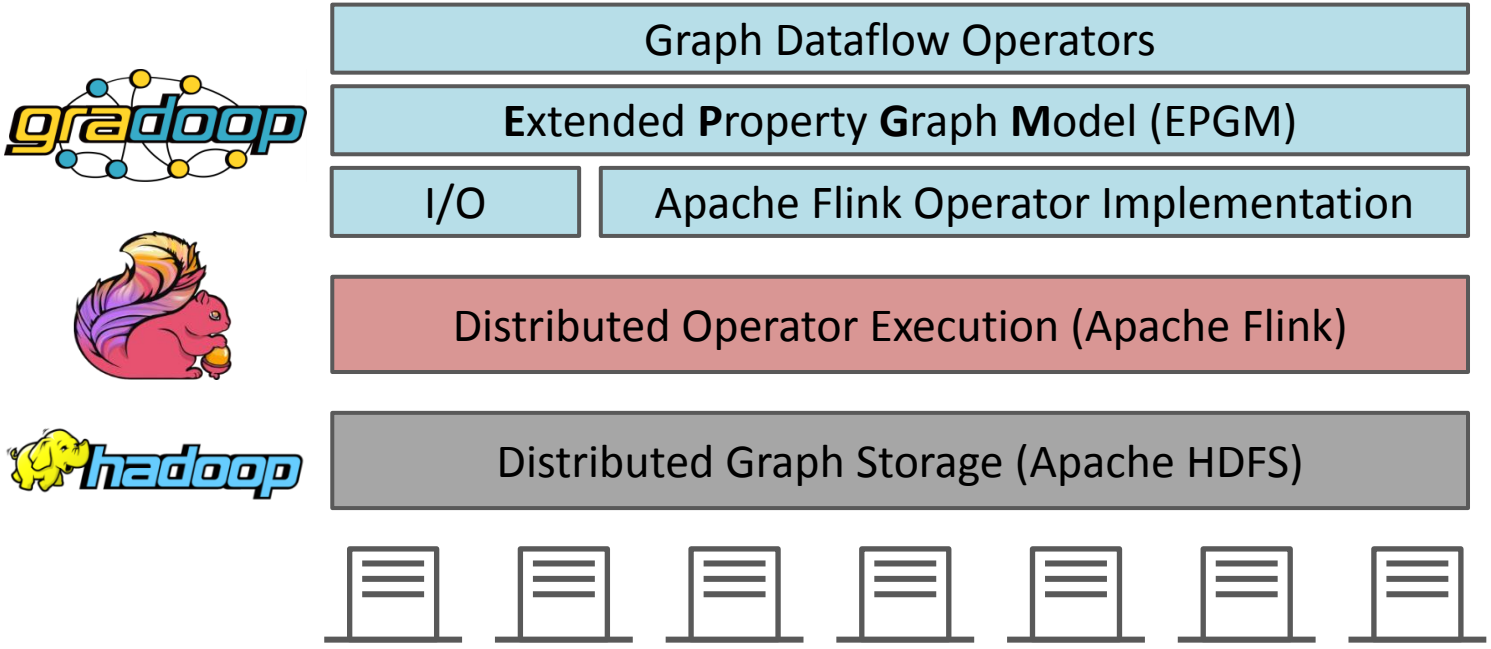
Distributed Graph Analytics with Gradoop

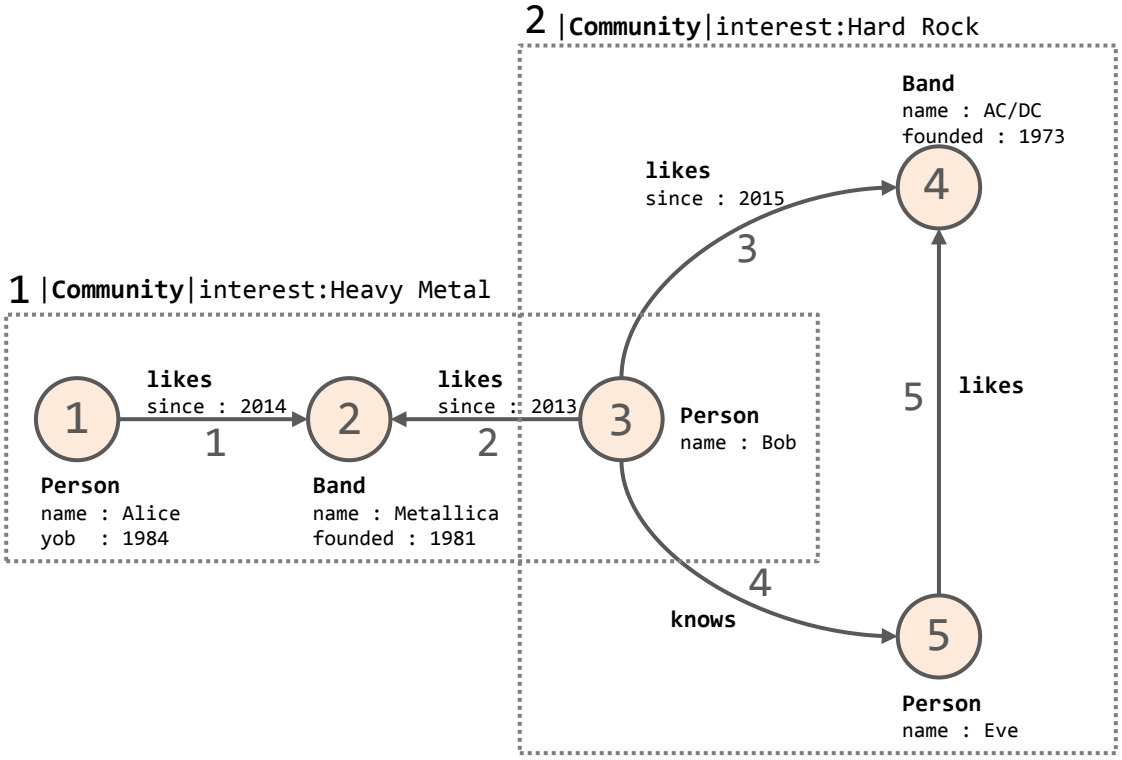
Martin Junghanns
University of Leipzig – Database Research Group

UNIVERSITÄT LEIPZIG



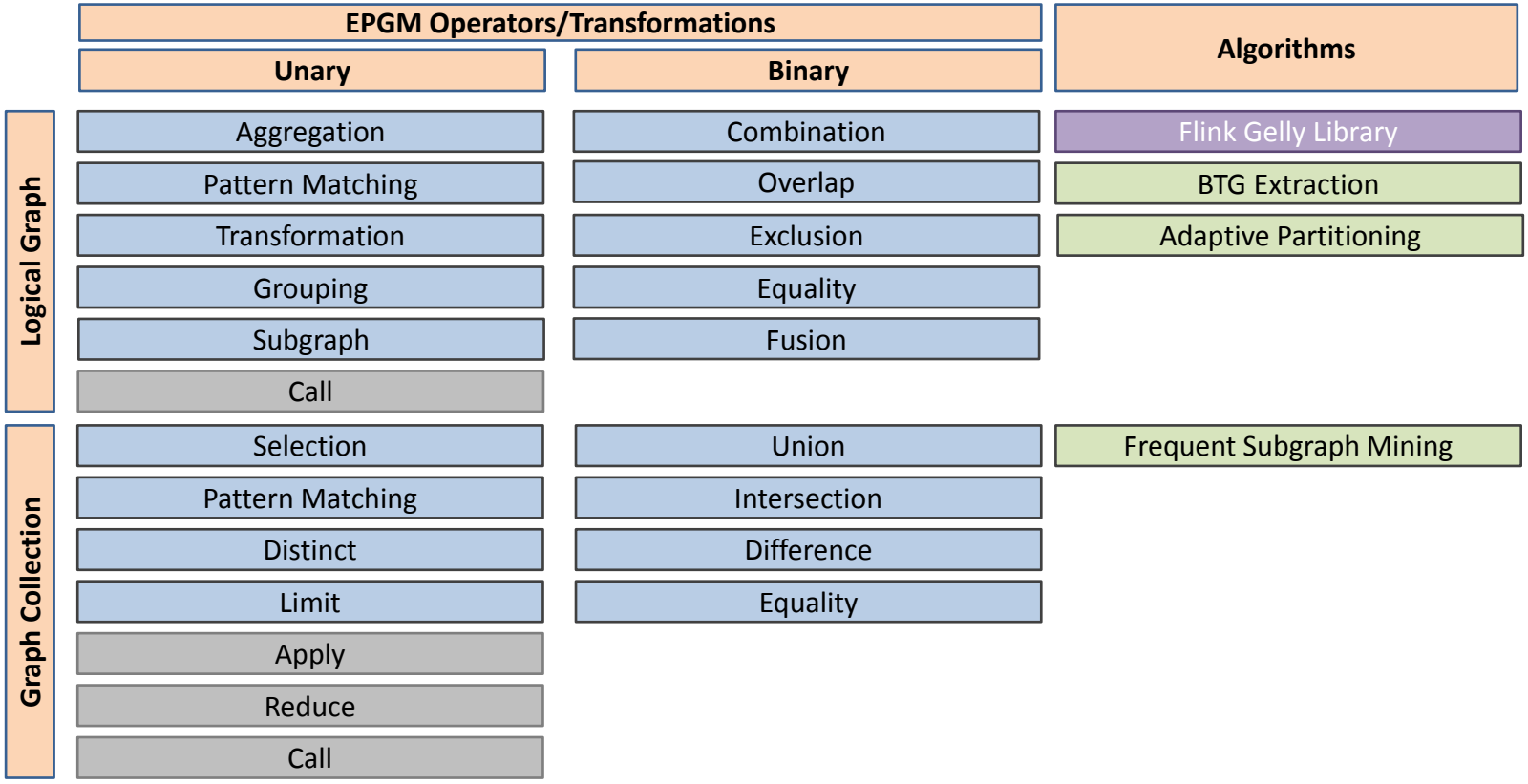
„An open-source **graph dataflow system** for **declarative analytics** of **heterogeneous** graph data.“


Graph Dataflow Operators
Extended Property Graph Model (EPGM)
I/O
Apache Flink Operator Implementation
Distributed Operator Execution (Apache Flink)
Distributed Graph Storage (Apache HDFS)

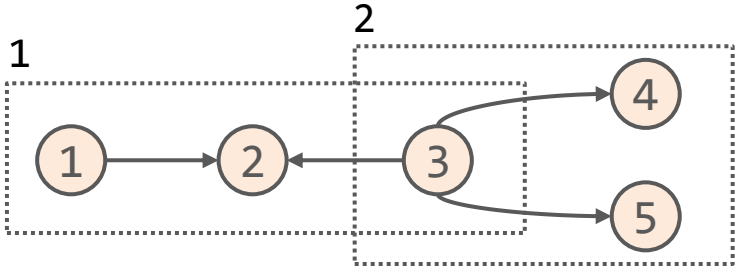


- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- Properties

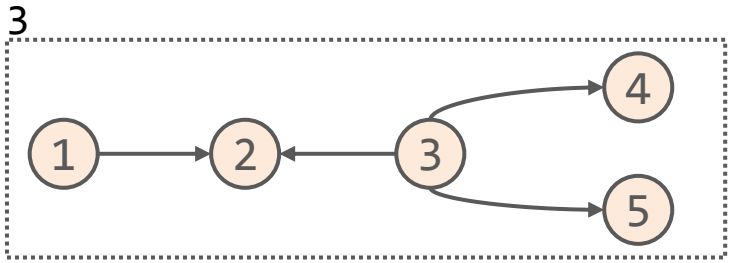
Extended **P**roperty **G**raph **M**odel (EPGM)


EPGM Operators/Transformations
Unary
Binary
Algorithms
Logical Graph
Aggregation
Pattern Matching
Transformation
Grouping
Subgraph
Call
Combination
Overlap
Exclusion
Equality
Fusion
Flink Gelly Library
BTG Extraction
Adaptive Partitioning
Graph Collection
Selection
Pattern Matching
Distinct
Limit
Apply
Reduce
Call
Union
Intersection
Difference
Equality
Frequent Subgraph Mining

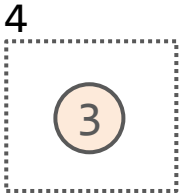
Basic Binary Operators



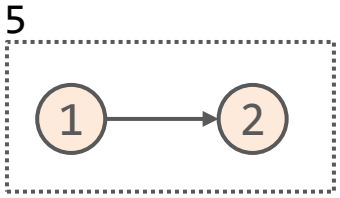
Combination



Overlap

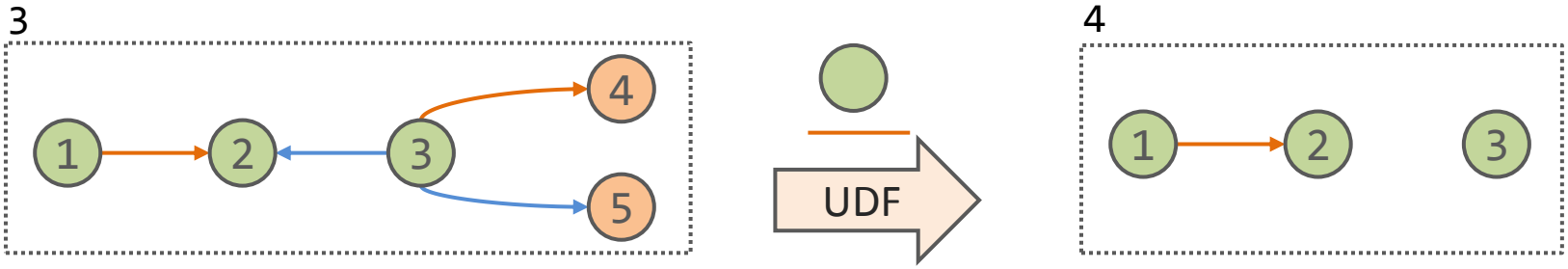


Exclusion



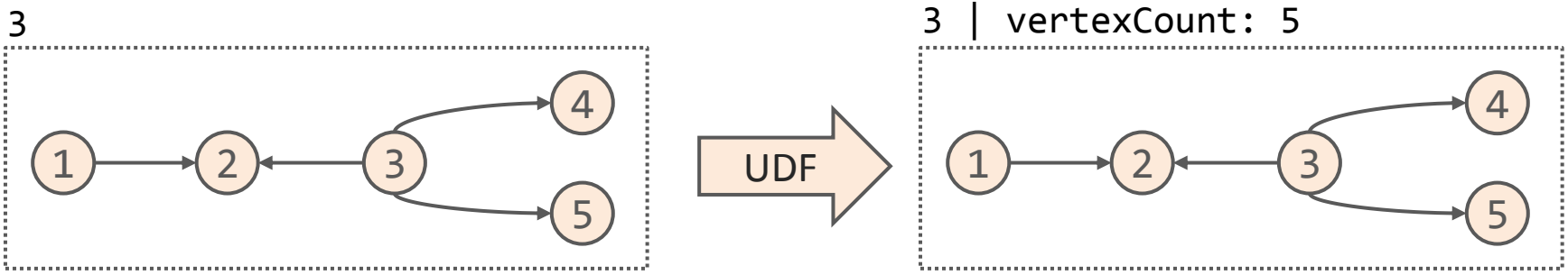

```
LogicalGraph graph3 = graph1.combine(graph2);
LogicalGraph graph4 = graph1.overlap(graph2);
LogicalGraph graph5 = graph1.exclude(graph2);
```

Subgraph



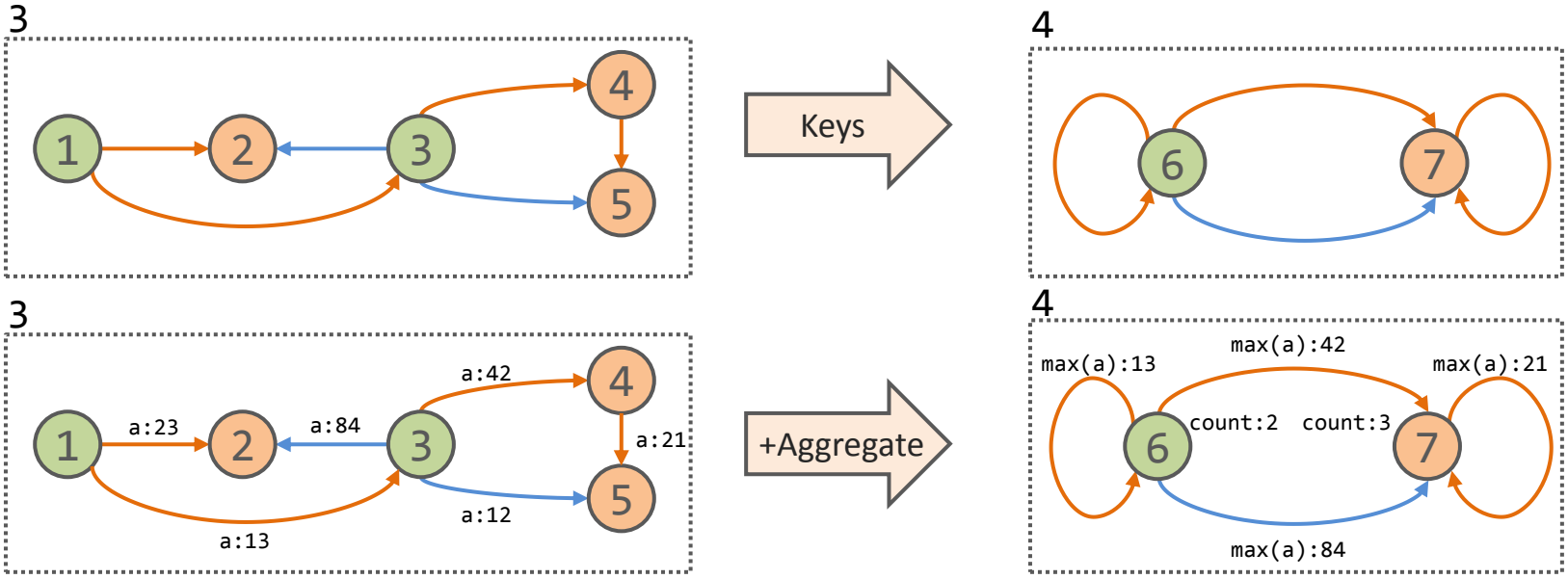

```
LogicalGraph graph4 = graph3.subgraph(
  (vertex => vertex.getLabel().equals('Green')),
  (edge   => edge.getLabel().equals('orange')));
```

Aggregation



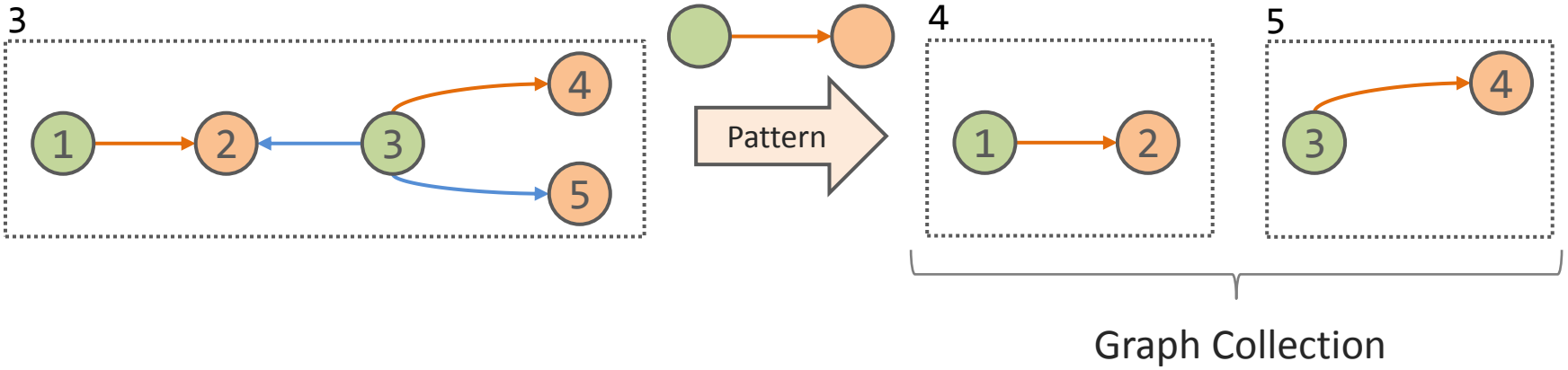

```
graph3 = graph3.aggregate(new VertexCount());
```

Grouping



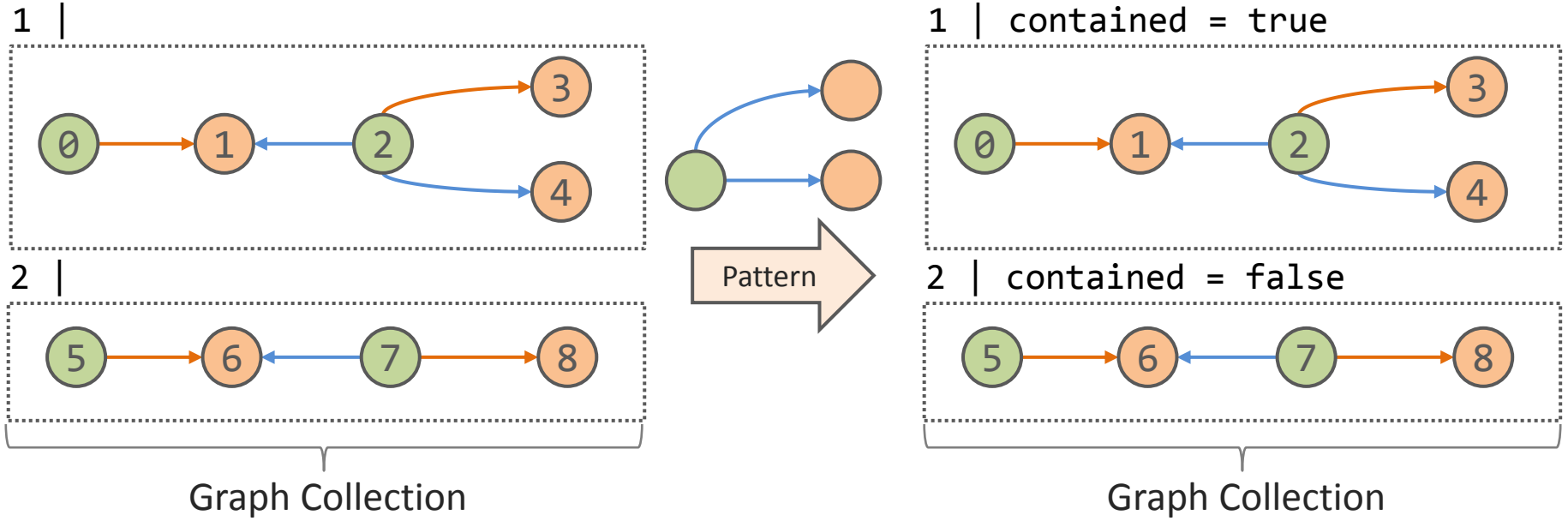

```
LogicalGraph grouped = graph3.groupBy()
  .useVertexLabel()
  .useEdgeLabel()
  .addVertexAggregate(new CountAggregator())
  .addEdgeAggregate(new MaxAggregator('a'));
```

Pattern Matching (Single Graph Input)



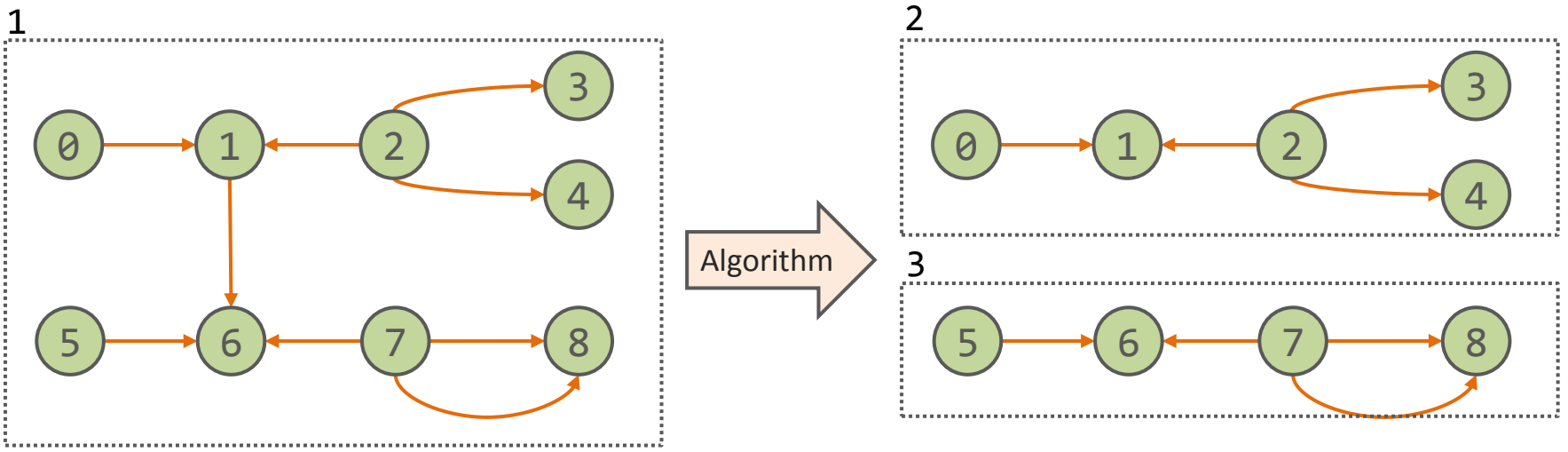

```
GraphCollection collection = graph3.match('(:Green)-[:orange]->(:Orange)');
```

Pattern Matching (Graph-Transaction Setting)



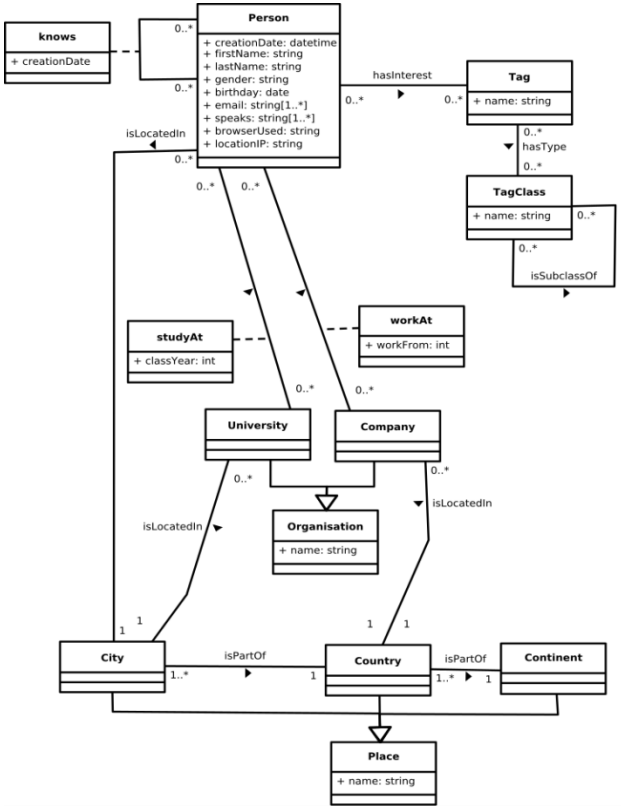

```
GraphCollection filtered = coll.match('(v0:Green)-[:blue]->(:Orange),(v0)-[:blue]->(:Orange)');
```

Call (e.g. Clustering)




```
GraphCollection clustering = graph.callForCollection(new ClusteringAlgorithm())
```

Overall performance [3]



http://ldbcouncil.org/

1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities to a single graph
7. **Group** graph by Persons *location* and *gender*
8. **Aggregate** vertex and edge count of grouped graph

Overall performance [3]

```
return socialNetwork
  // 1) extract subgraph
  .subgraph((vertex) → {
      return vertex.getLabel().toLowerCase().equals(person);
  }, (edge) → { return edge.getLabel().toLowerCase().equals(knows); })
  // project to necessary information
  .transform((current, transformed) → { return current; }, (current, transformed) → {
      transformed.setLabel(current.getLabel());
      transformed.setProperty(city, current.getPropertyValue(city));
      transformed.setProperty(gender, current.getPropertyValue(gender));
      transformed.setProperty(label, current.getPropertyValue(birthday));
      return transformed;
  }, (current, transformed) → {
      transformed.setLabel(current.getLabel());
      return transformed;
  })
  // 3a) compute communities
  .callForGraph(new GellyLabelPropagation<GraphHeadPojo, VertexPojo, EdgePojo>(maxIterations, label))
  // 3b) separate communities
  .splitBy(label)
  // 4) compute vertex count per community
  .apply(new ApplyAggregation<>(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>()))
  // 5) select graphs with more than minClusterSize vertices
  .select((g) → { return g.getPropertyValue(vertexCount).getLong() > threshold; })
  // 6) reduce filtered graphs to a single graph using combination
  .reduce(new ReduceCombination<GraphHeadPojo, VertexPojo, EdgePojo>())
  // 7) group that graph by vertex properties
  .groupBy(Lists.newArrayList(city, gender))
  // 8a) count vertices of grouped graph
  .aggregate(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>())
  // 8b) count edges of grouped graph
  .aggregate(edgeCount, new EdgeCount<GraphHeadPojo, VertexPojo, EdgePojo>());
```

1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities to a single graph
7. **Group** graph by Persons *location* and *gender*
8. **Aggregate** vertex and edge count of grouped graph

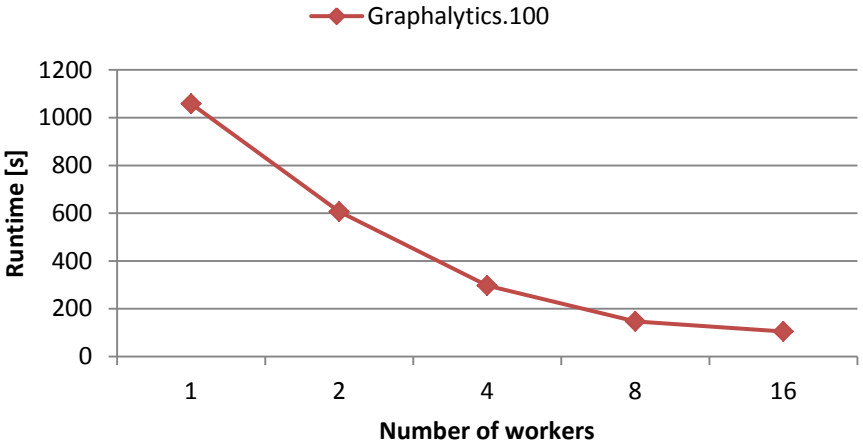
https://git.io/vD2Ii

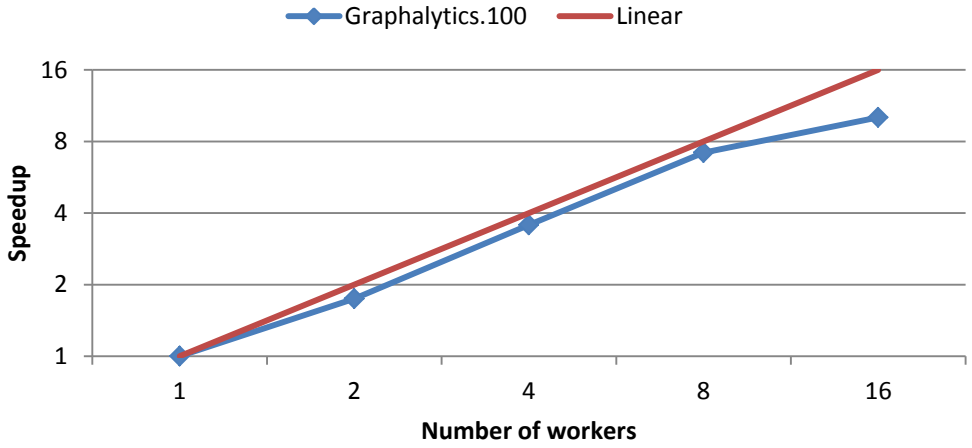
Overall performance [3]

Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

Overall performance [3]

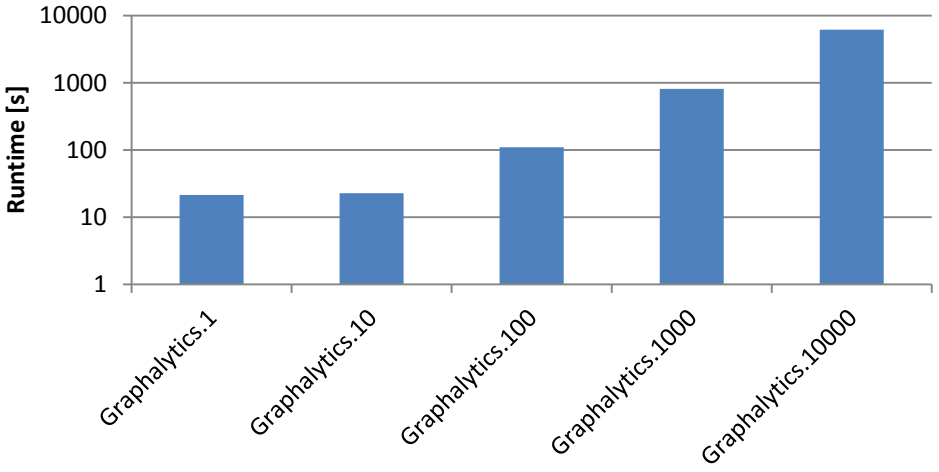




Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

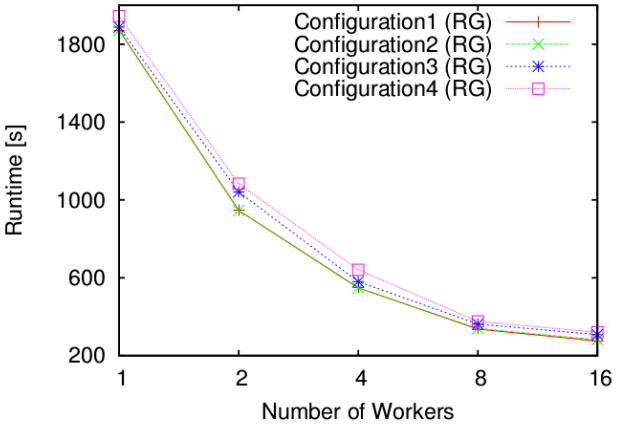
Overall performance [3]



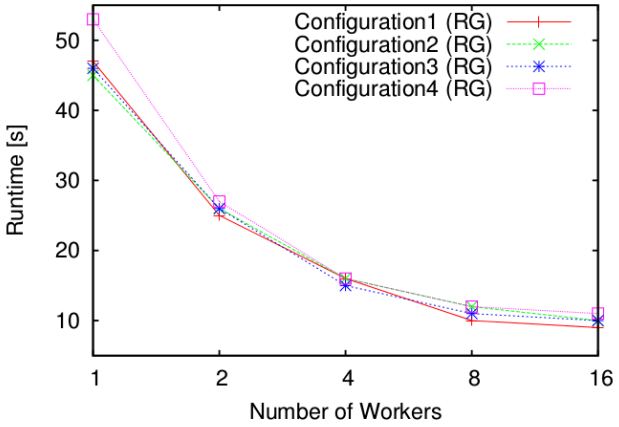
Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

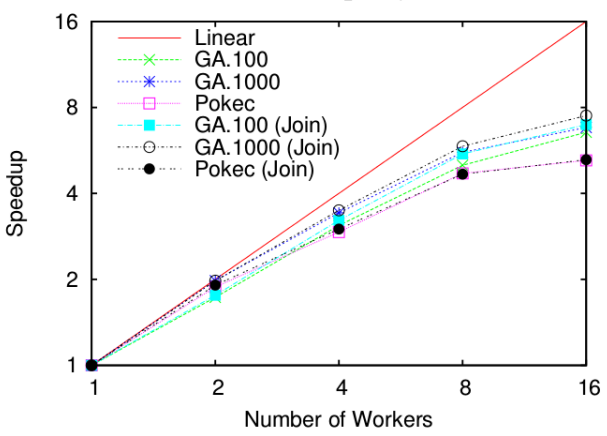
Grouping [1]



(a) Runtime on Graphalytics 1000



(b) Runtime on Pokec



(c) Speedup for configuration 1 (RG)

Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Pokec	1,632,803	30,622,564

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- **Flink 1.0.3**
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

- Extended Property Graph Model
  - Schema flexible: Type Labels and Properties
  - Logical Graphs / Graphs Collection
- Graph and Collection Operators
  - Combination to analytical workflows
- Implemented on Apache Flink
  - Built-in scalability
  - Combine with other libraries

[1] Junghanns, M.; Petermann, A.; K.; Rahm, E., „**Distributed Grouping of Property Graphs with Gradoop**“, Proc. BTW Conf. , 2017.

[2] Petermann, A.; Junghanns, M.; Kemper, S.; Gomez, K.; Teichmann, N.; Rahm, E., „**Graph Mining for Complex Data Analytics** “, Proc. ICDM Conf. (Demo), 2016.

[3] Junghanns, M.; Petermann, A.; Teichmann, N.; Gomez, K.; Rahm, E., „**Analyzing Extended Property Graphs with Apache Flink**“, Int. Workshop on Network Data Analytics (NDA), SIGMOD, 2016.

[4] Petermann, A.; Junghanns, M., „**Scalable Business Intelligence with Graph Collections**“, it – Special Issue on Big Data Analytics, 2016.

[5] Petermann, A.; Junghanns, M.; Müller, M.; Rahm, E., „**Graph-based Data Integration and Business Intelligence with BIIIG**“, Proc. VLDB Conf. (Demo), 2014.

www.gradoop.com

UNIVERSITÄT LEIPZIG

gradoop

ScaDS DRESDEN LEIPZIG