UNIVERSITÄT LEIPZIG

ScaDS
DRESDEN LEIPZIG

SCALABLE GRAPH DATA ANALYTICS WITH GRADOOP

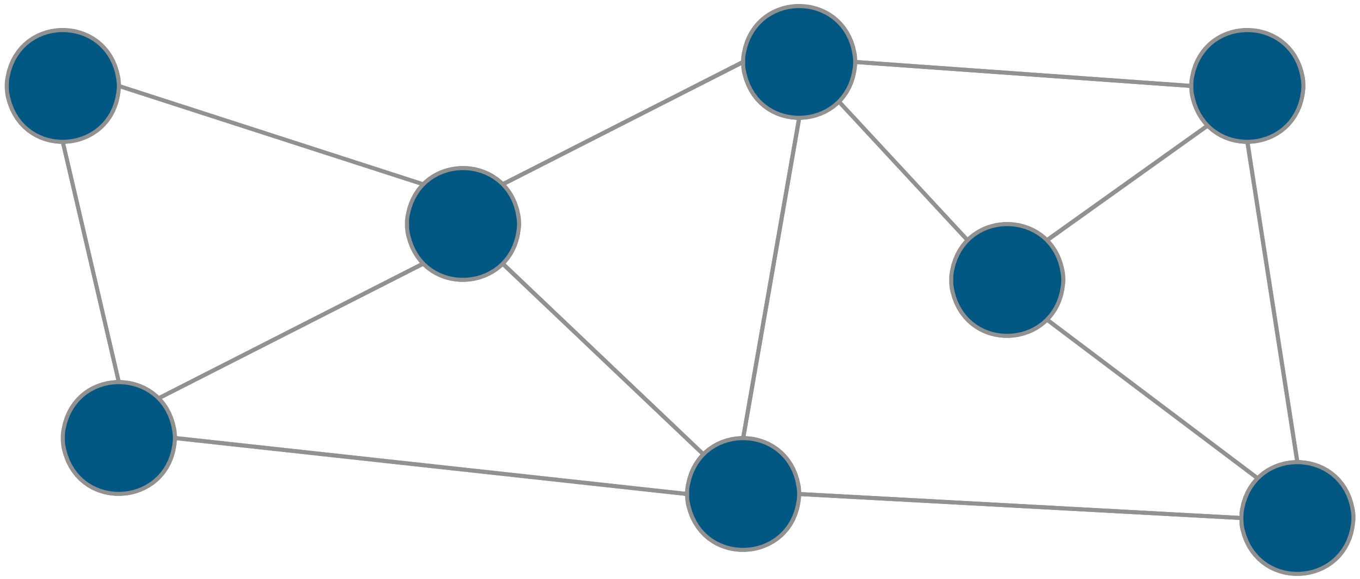
ERHARD RAHM,

MARTIN JUNGHANNS, ANDRÉ PETERMANN, ERIC PEUKERT

www.scads.de

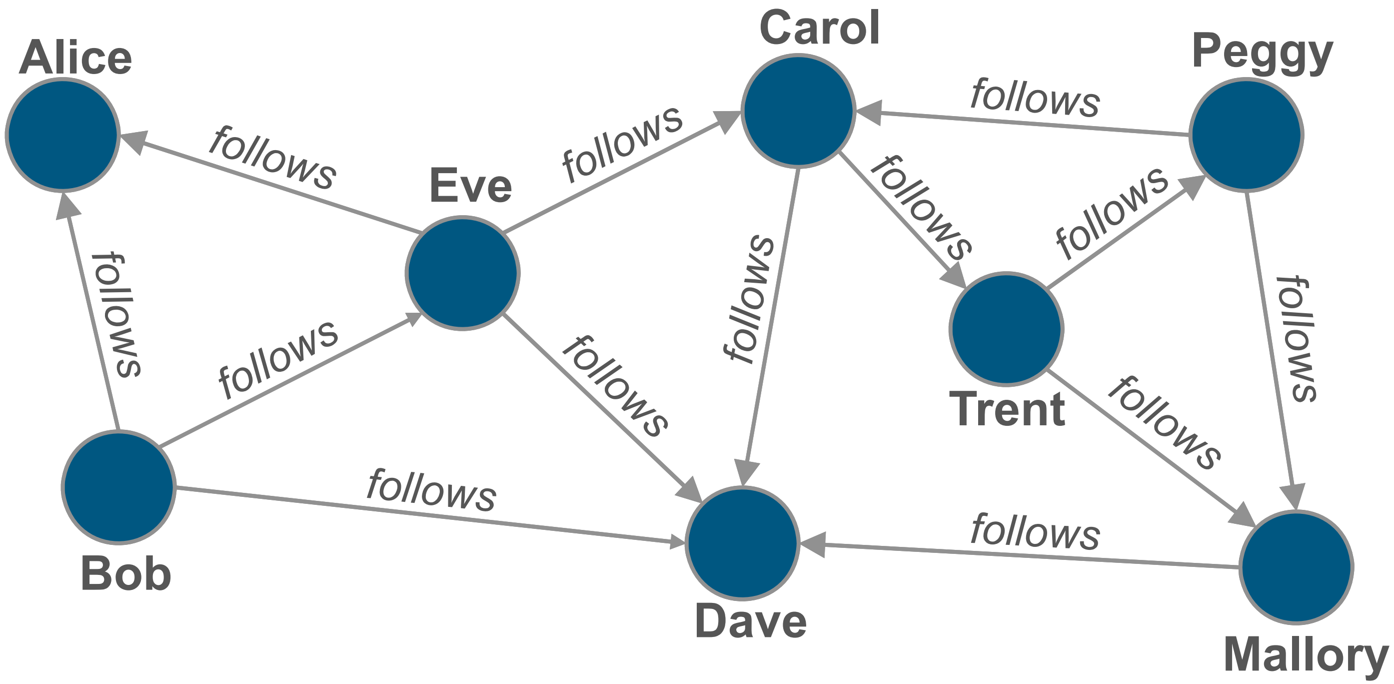
ScaDS DRESDEN LEIPZIG

"GRAPHS ARE EVERYWHERE"



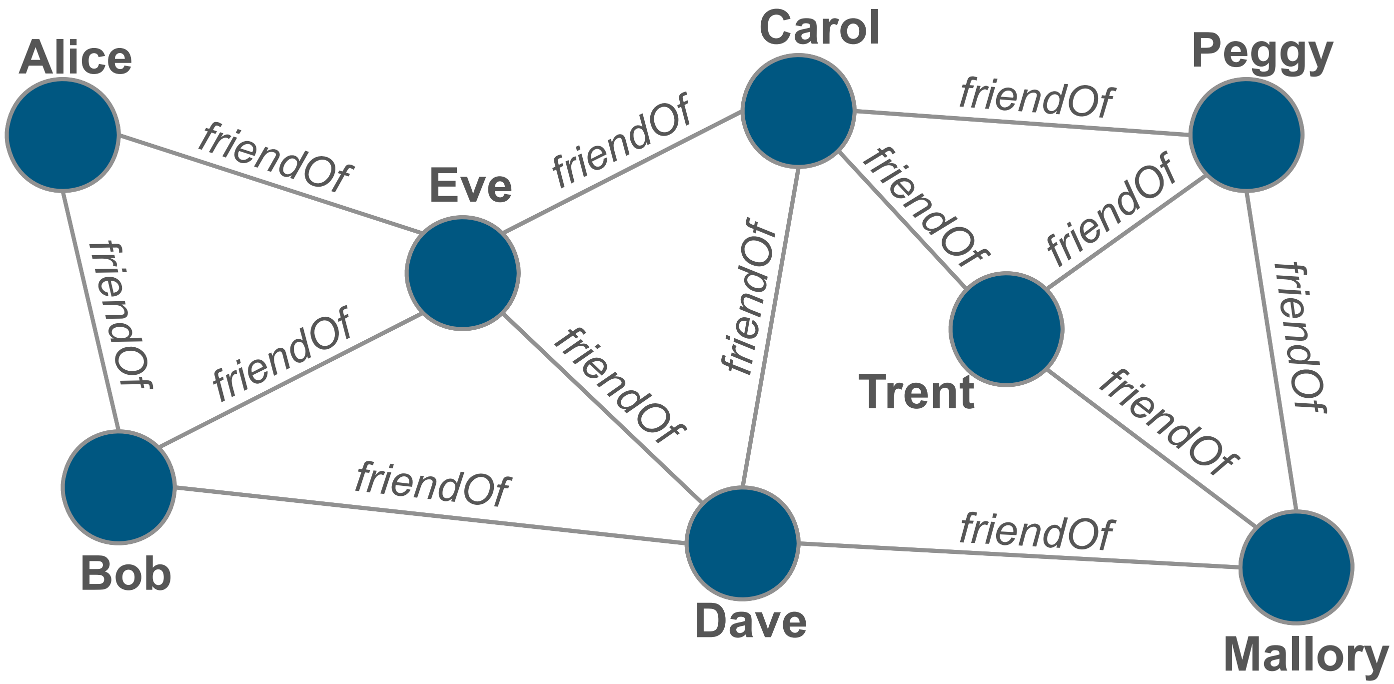
$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

“GRAPHS ARE EVERYWHERE”



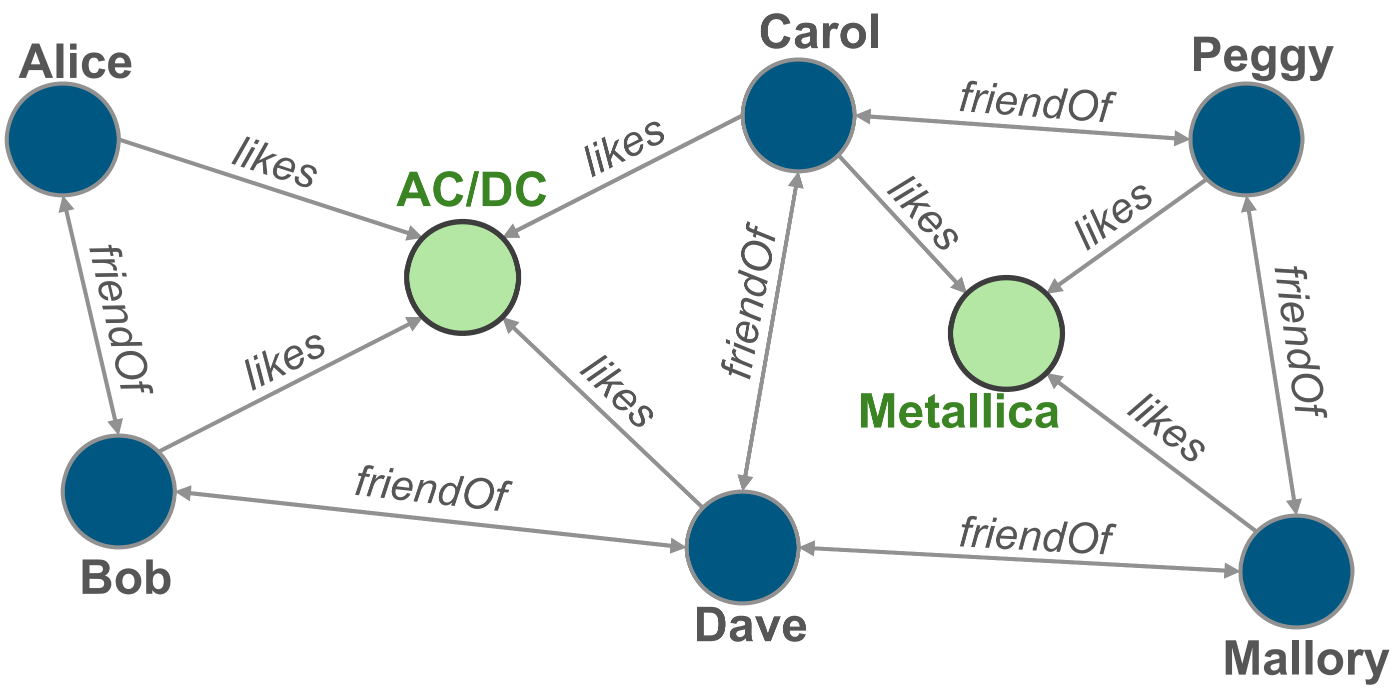
$$Graph = (\mathbf{Users}, Followers)$$

“GRAPHS ARE EVERYWHERE”



$$Graph = (\mathbf{Users}, Friendships)$$

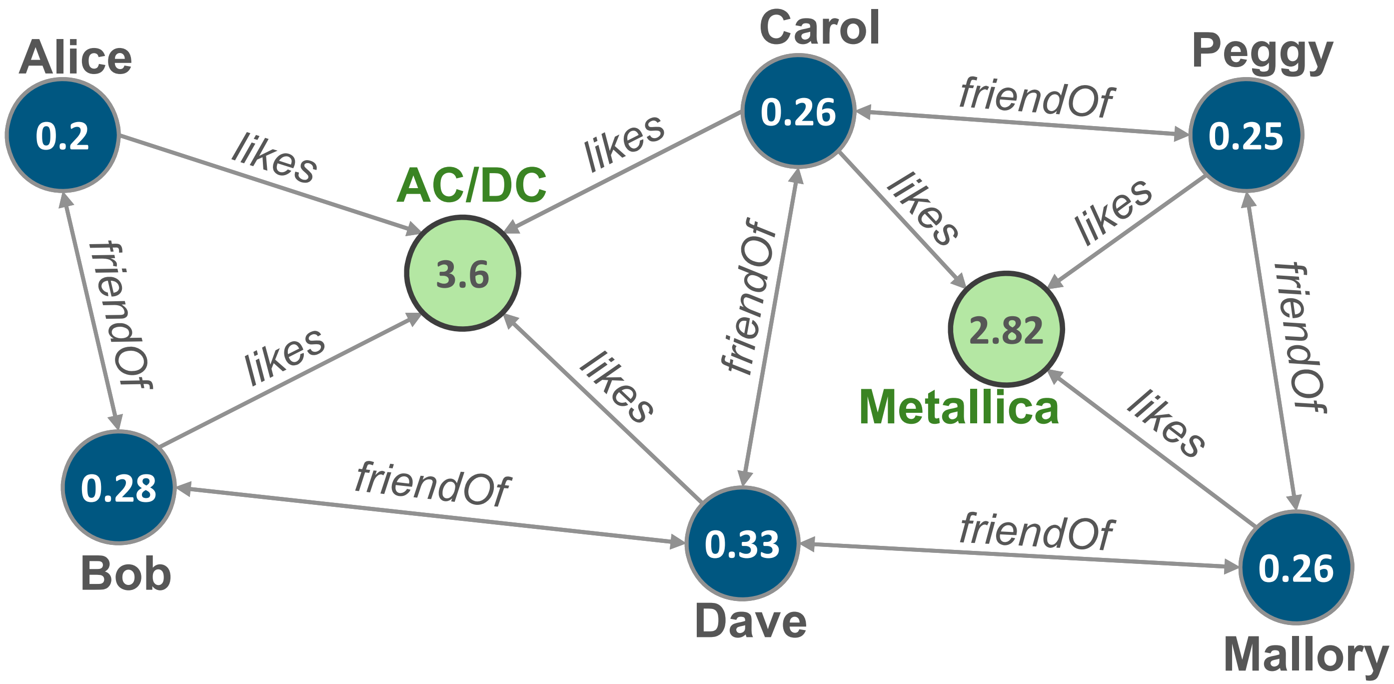
"GRAPHS ARE HETEROGENEOUS"



$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

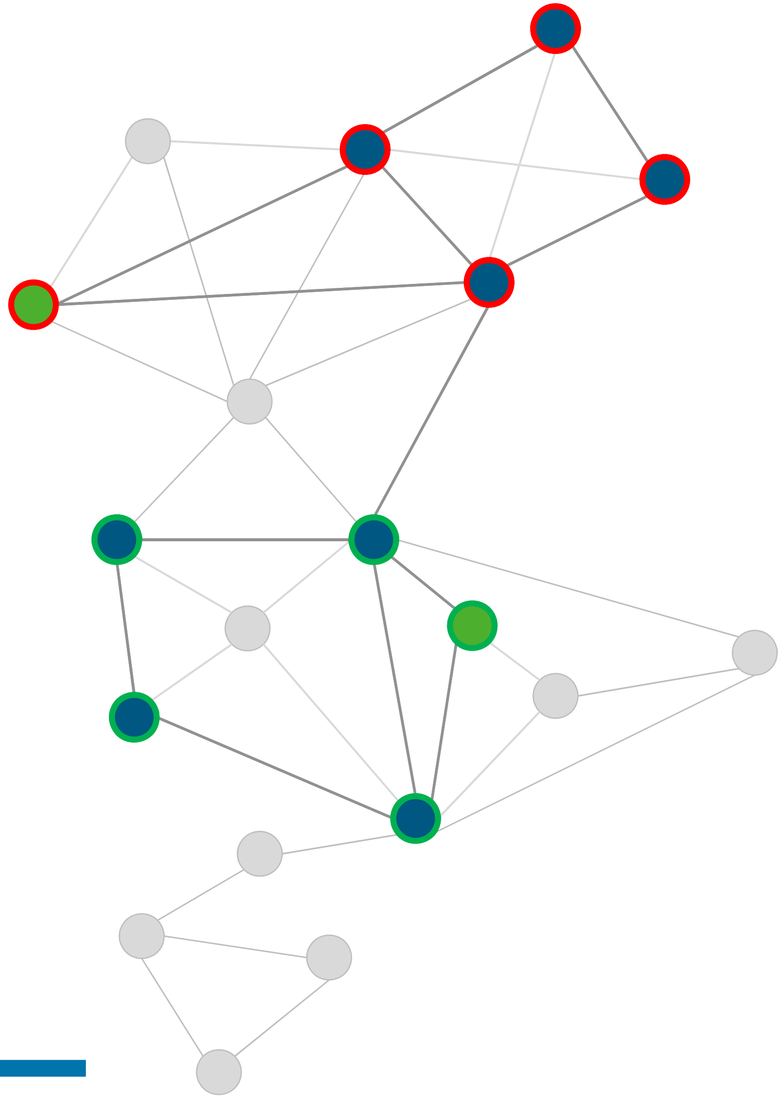
ScaDS DRESDEN LEIPZIG

"GRAPHS CAN BE ANALYZED"



$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

“GRAPHS CAN BE ANALYZED“



Assuming a social network

1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph

ScaDS DRESDEN LEIPZIG

GRAPH DATA ANALYTICS: REQUIREMENTS

- *all V challenges (volume, variety, velocity, veracity)*
- *ease-of-use*
- *high cost-effectiveness*
- powerful but easy to use graph data model
  - support for heterogeneous, schema-flexible vertices and edges
  - support for collections of graphs (not only 1 graph)
  - powerful graph operators
- graph-based integration of many data sources
- versioning and evolution (dynamic /temporal graphs)
- interactive, declarative graph queries
- scalable graph mining
- comprehensive visualization support

ScaDS DRESDEN LEIPZIG

COMPARISON

	Graph Database Systems Neo4j, OrientDB		
data model	rich graph models (PGM)		
focus	queries		
query language	yes		
graph analytics	no		
scalability	vertical		
Workflows	no		
persistency	yes		
dynamic graphs / versioning	no		
data integration	no		
visualization	(yes)		

ScaDS DRESDEN LEIPZIG

COMPARISON (2)

	Graph Database Systems Neo4j, OrientDB	Graph Processing Systems (Pregel, Giraph)	
data model	rich graph models (PGM)	generic graph models	
focus	queries	analytic	
query language	yes	no	
graph analytics	no	yes	
scalability	vertical	horizontal	
Workflows	no	no	
persistency	yes	no	
dynamic graphs / versioning	no	no	
data integration	no	no	
visualization	(yes)	no	

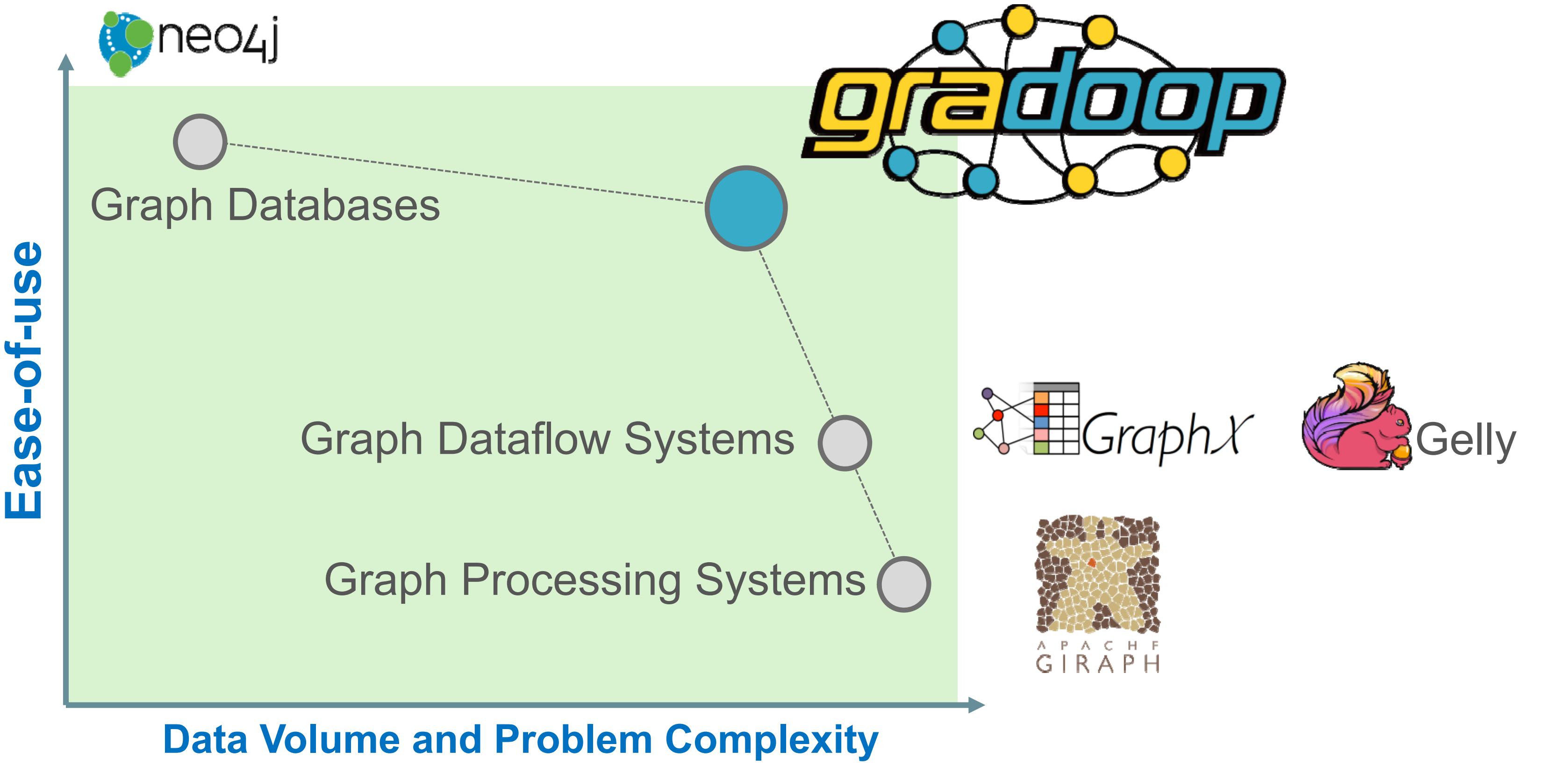
ScaDS DRESDEN LEIPZIG

COMPARISON (3)

	Graph Database Systems Neo4j, OrientDB	Graph Processing Systems (Pregel, Giraph)	Distributed Dataflow Systems (Flink Gelly, Spark GraphX)
data model	rich graph models (PGM)	generic graph models	generic graph models
focus	queries	analytic	analytic
query language	yes	no	no
graph analytics	no	yes	yes
scalability	vertical	horizontal	horizontal
Workflows	no	no	yes
persistency	yes	no	no
dynamic graphs / versioning	no	no	no
data integration	no	no	no
visualization	(yes)	no	no

WHAT'S MISSING?

An end-to-end framework and research platform for efficient, distributed and domain independent graph data management and analytics.


neo4j
gradoop
Graph Databases
Graph Dataflow Systems
Graph Processing Systems
GraphX
Gelly
APACHE GIRAPH
Ease-of-use
Data Volume and Problem Complexity

ScaDS DRESDEN LEIPZIG

AGENDA

- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems
- Gradoop
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Implementation
  - Evaluation
- Summary/Outlook

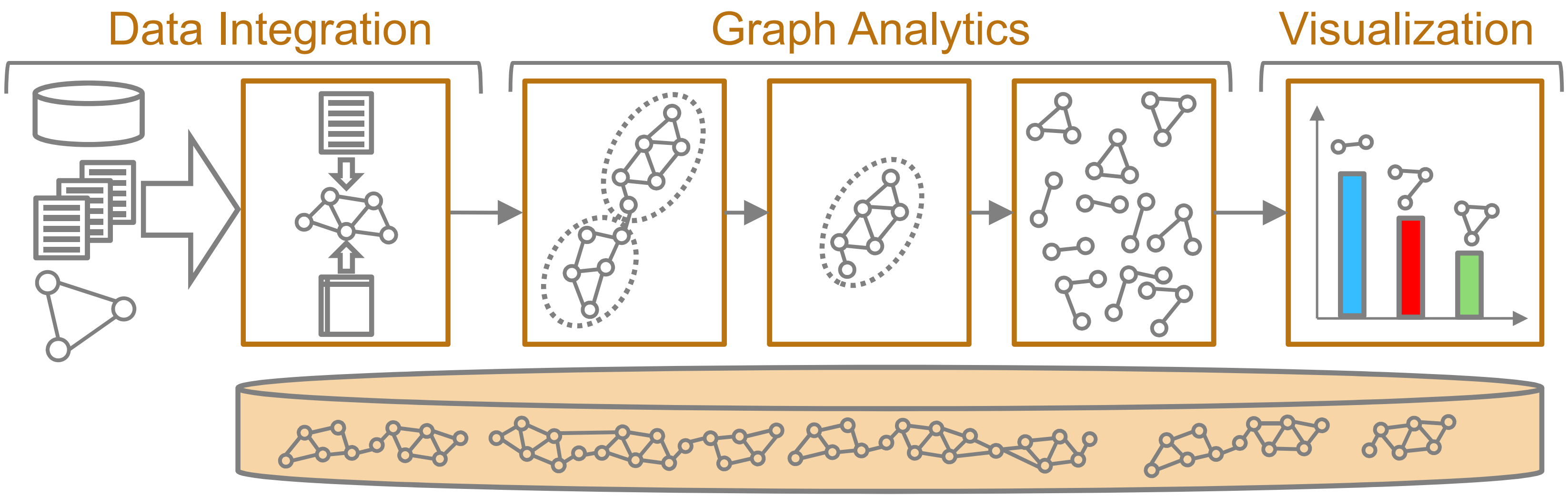
ScaDS DRESDEN LEIPZIG

GRADOOP CHARACTERISTICS

- Hadoop-based framework for graph data management and analysis
  - persistent graph storage in scalable distributed store (Hbase)
  - utilization of powerful dataflow system (Apache Flink) for parallel, in-memory processing
- Extended property graph data model (EPGM)
  - operators on graphs and sets of (sub) graphs
  - support for semantic graph queries and mining
- Declarative specification of graph analysis workflows
  - Graph Analytical Language - GrALa
- End-to-end functionality
  - graph-based data integration, data analysis and visualization
- Open-source implementation: www.gradoop.org

ScaDS DRESDEN LEIPZIG

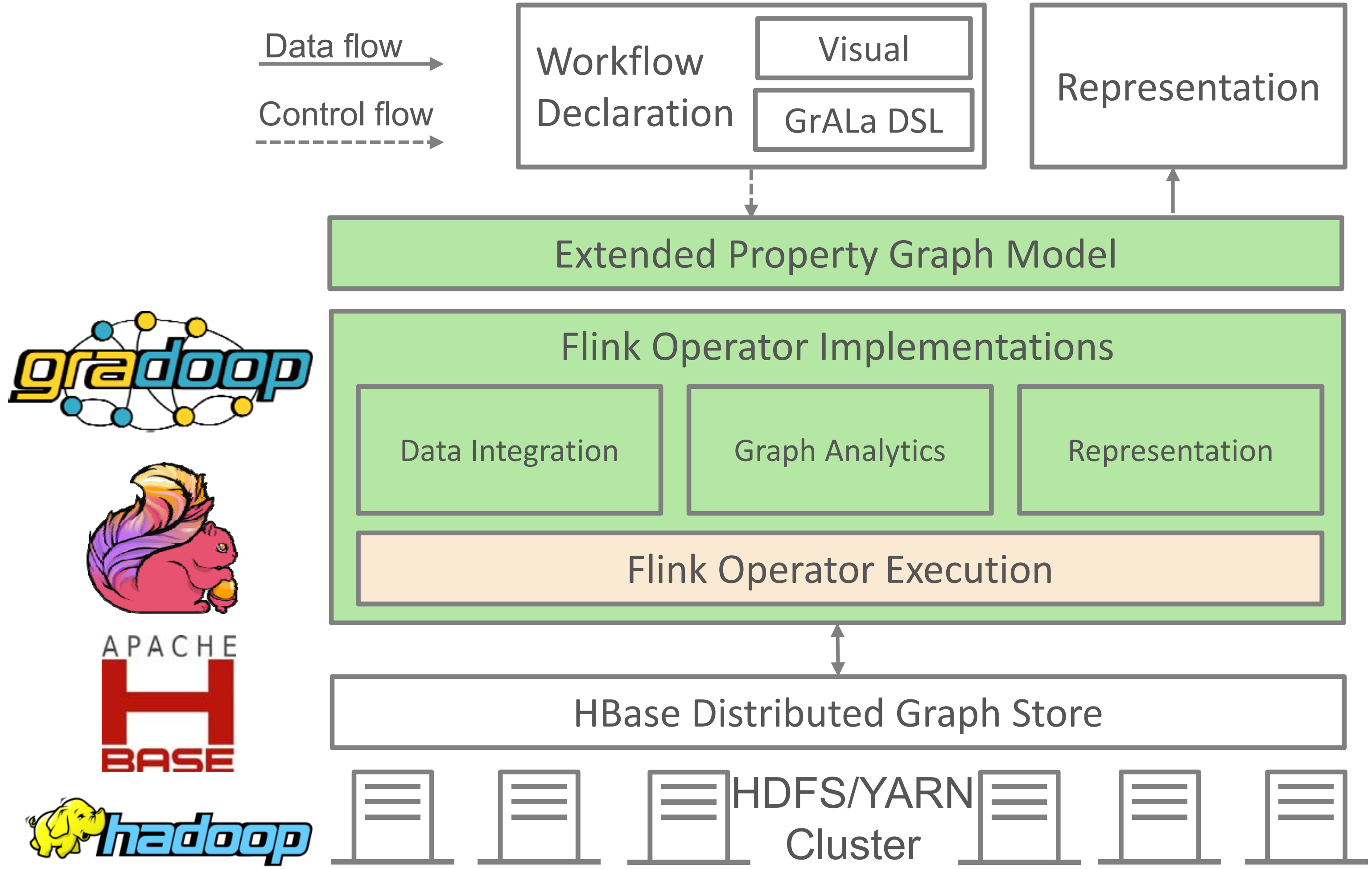
END-TO-END GRAPH ANALYTICS



- **integrate data** from one or more sources into a dedicated **graph store** with **common graph data model**
- definition of **analytical workflows** from **operator algebra**
- result representation in **meaningful way**

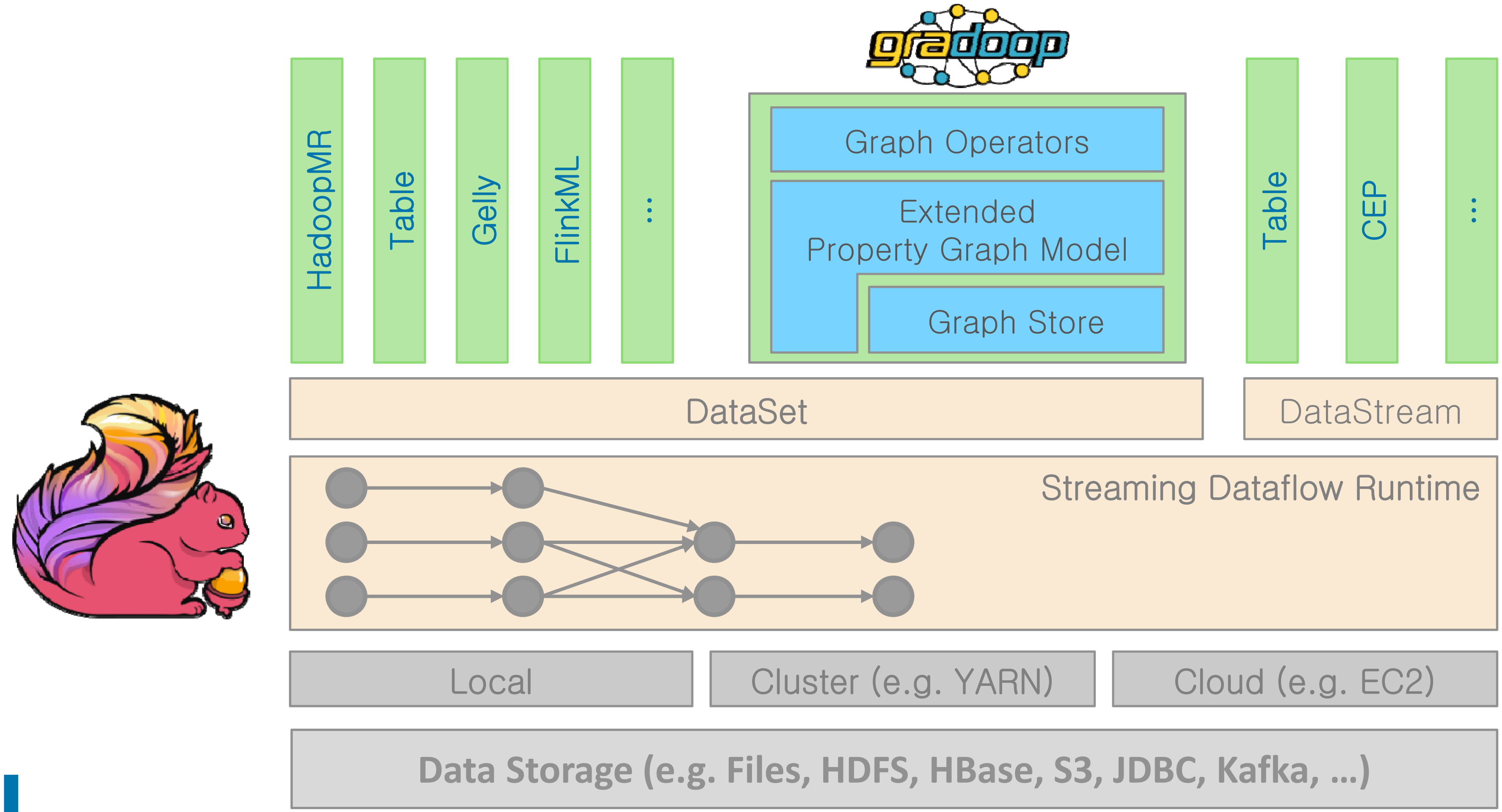
ScaDS DRESDEN LEIPZIG

HIGH LEVEL ARCHITECTURE



ScaDS DRESDEN LEIPZIG

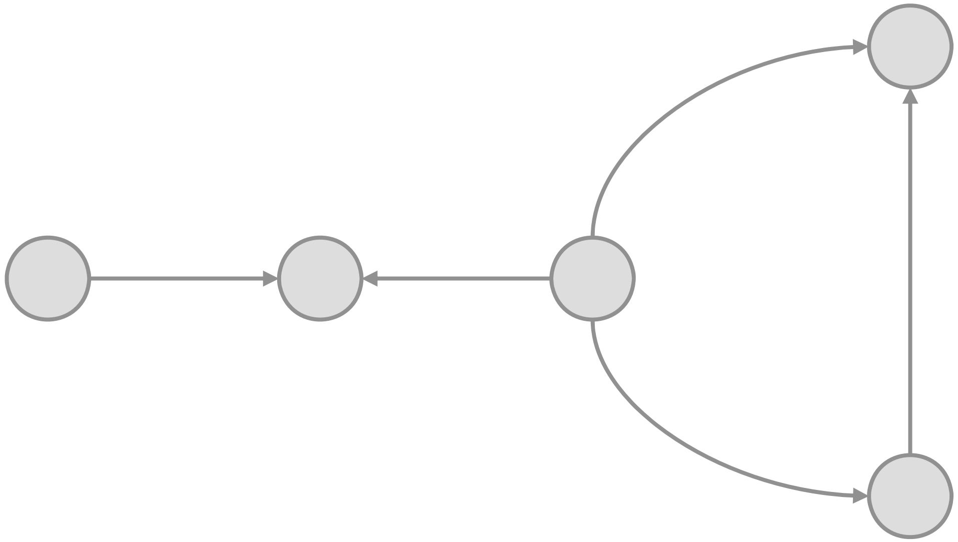
GRADOOP AS A FLINK EXTENSION



EXTENDED PROPERTY GRAPH MODEL (EPGM)

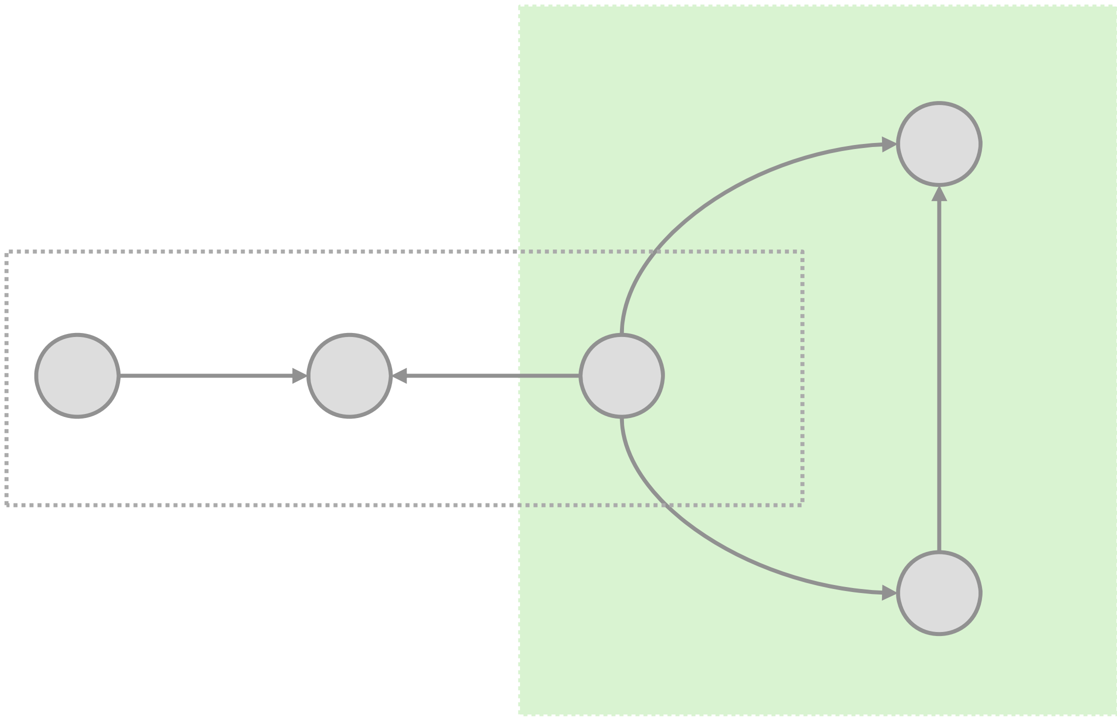
- includes PGM as special case
- support for collections of logical graphs / subgraphs
  - can be defined explicitly
  - can be result of graph algorithms / operators
- support for graph properties
- powerful operators on both graphs and graph collections
- Graph Analytical Language – GrALa
  - domain-specific language (DSL) for EPGM
  - flexible use of operators with application-specific UDFs
  - plugin concept for graph mining algorithms

- **Vertices and directed Edges**

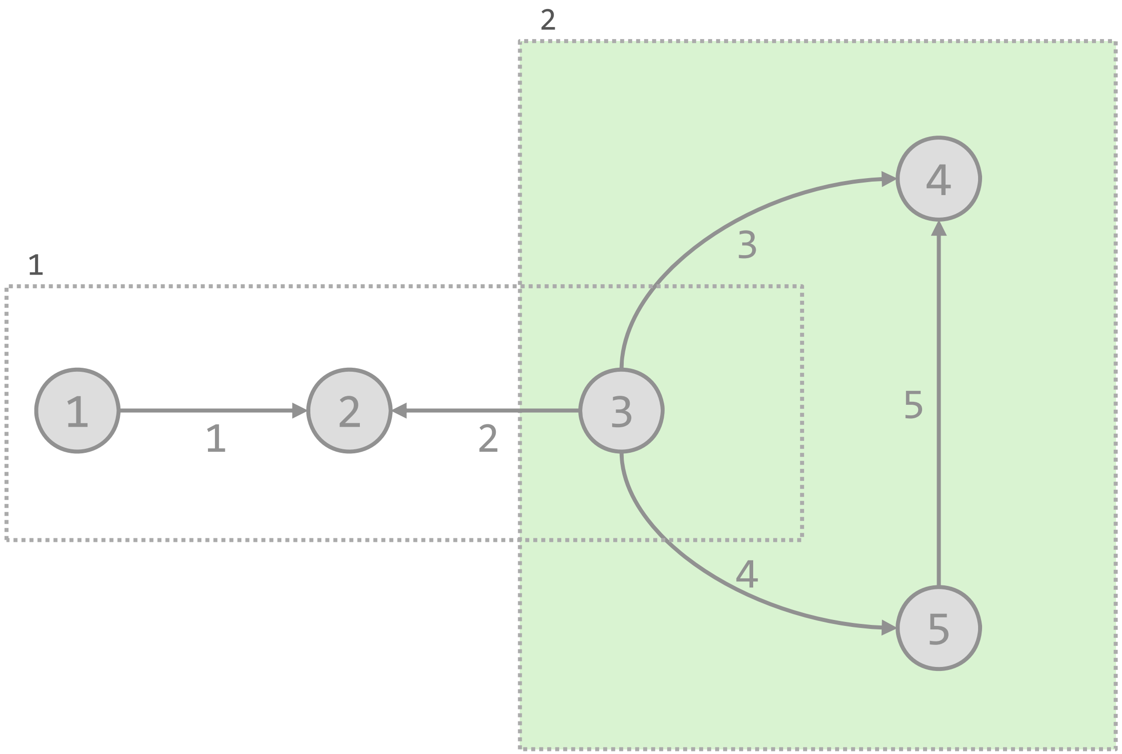


UNIVERSITÄT LEIPZIG

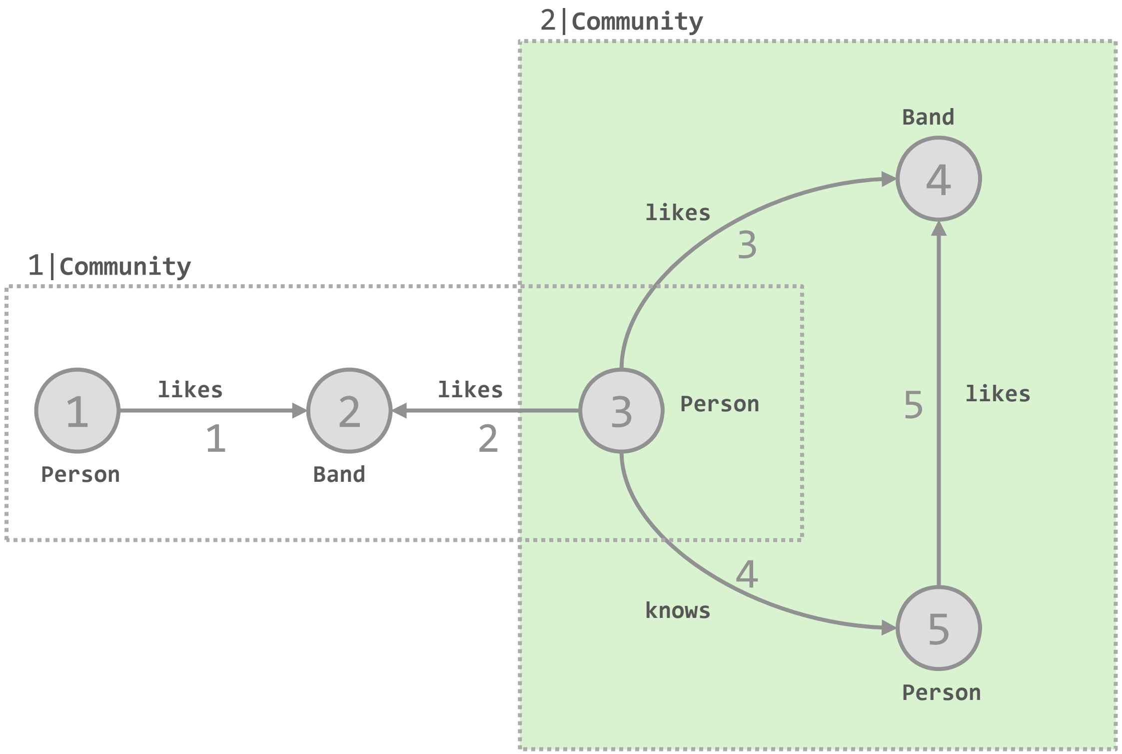
ScaDS DRESDEN LEIPZIG



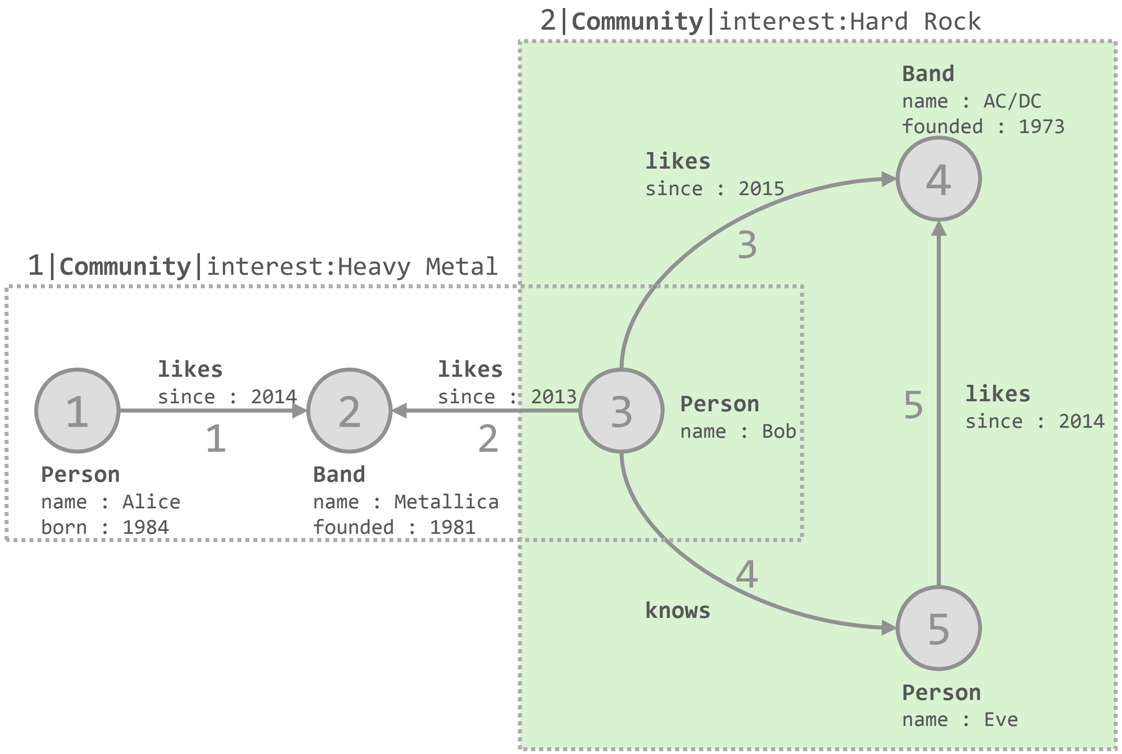
- Vertices and directed Edges
- **Logical Graphs**



- Vertices and directed Edges
- Logical Graphs
- **Identifiers**



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



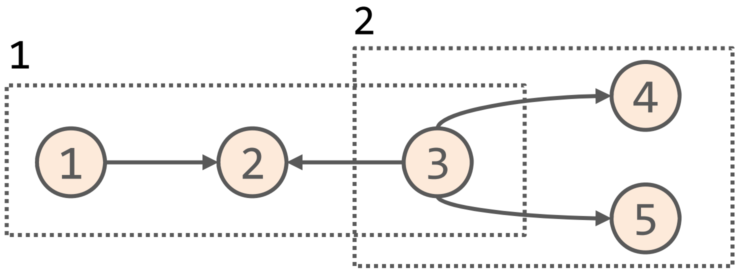
- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- **Properties**

Operators

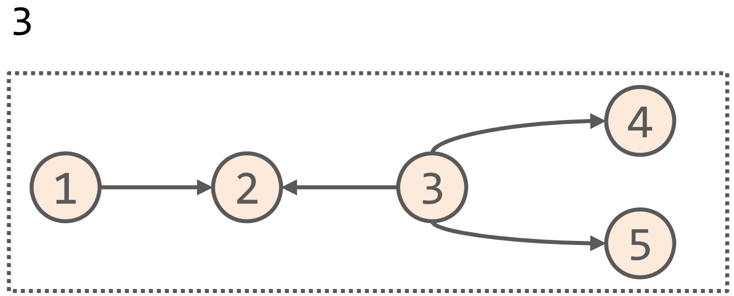
	Operators: Unary	Operators: Binary	Algorithms
Logical Graph	Aggregation	Combination	Gelly Library
	Pattern Matching	Overlap	BTG Extraction
	Transformation	Exclusion	Adaptive Partitioning
	Grouping	Equality	
	Subgraph		
	Call *		
Graph Collection	Selection	Union	Frequent Subgraphs
	Distinct	Intersection	
	Sort	Difference	
	Limit	Equality	
	Apply *		
	Reduce *		
	Call *		

* auxiliary

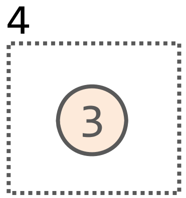
BASIC BINARY OPERATORS



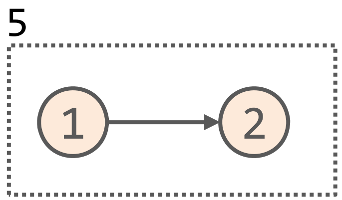
Combination



Overlap

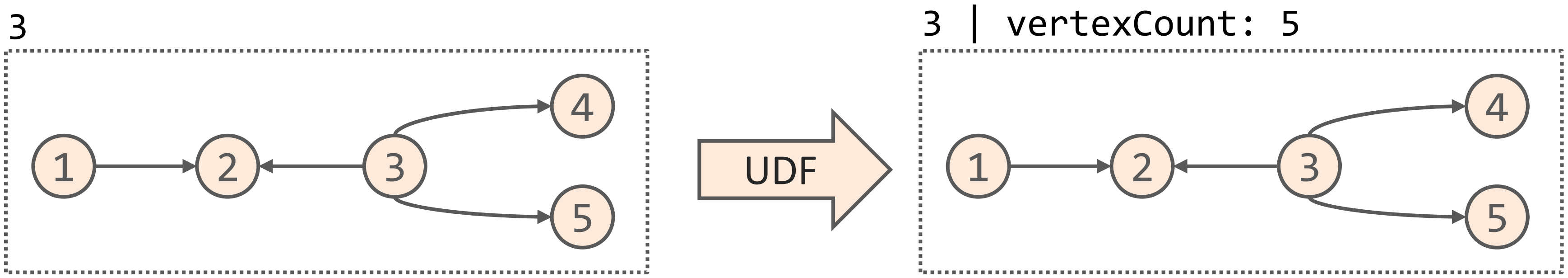


Exclusion




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

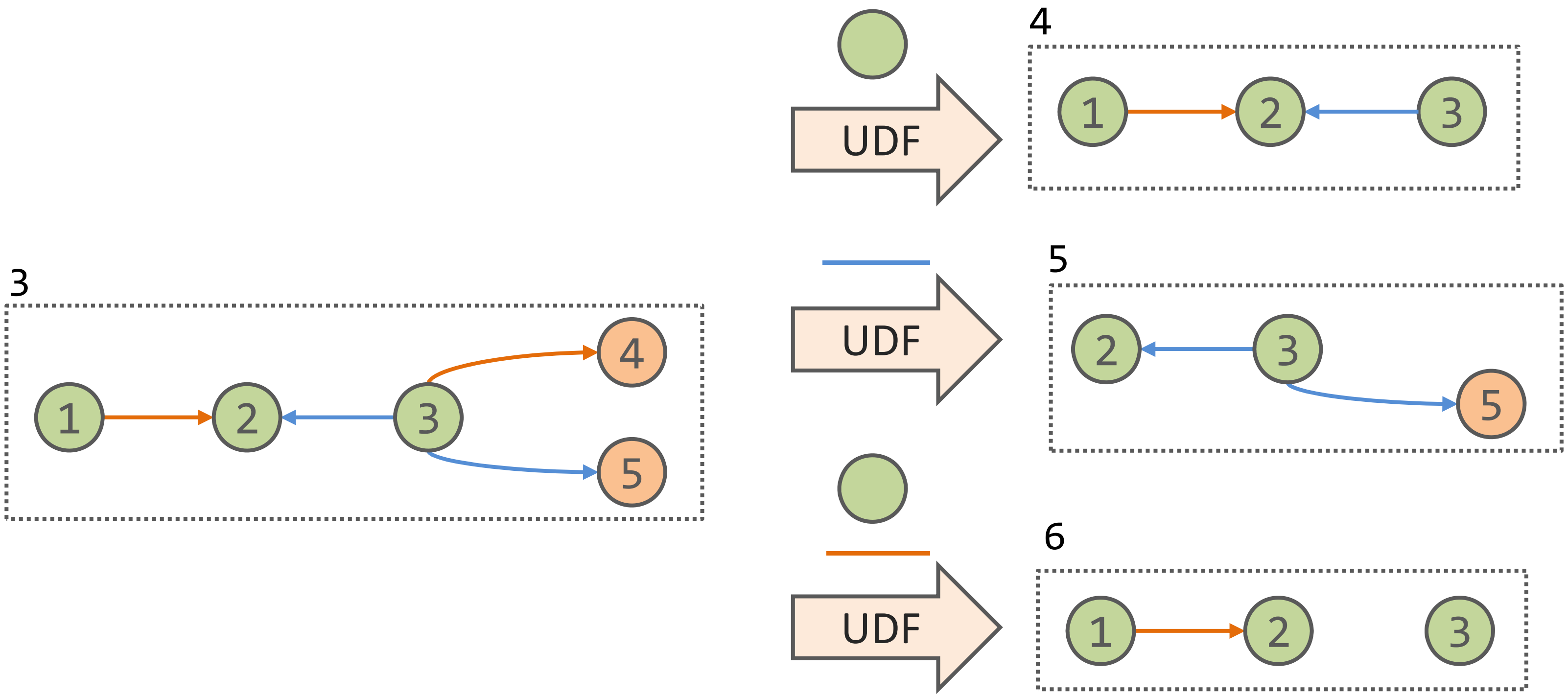
AGGREGATION




```
udf = (graph => graph['vertexCount'] = graph.vertices.size())
graph3 = graph3.aggregate(udf)
```

ScaDS DRESDEN LEIPZIG

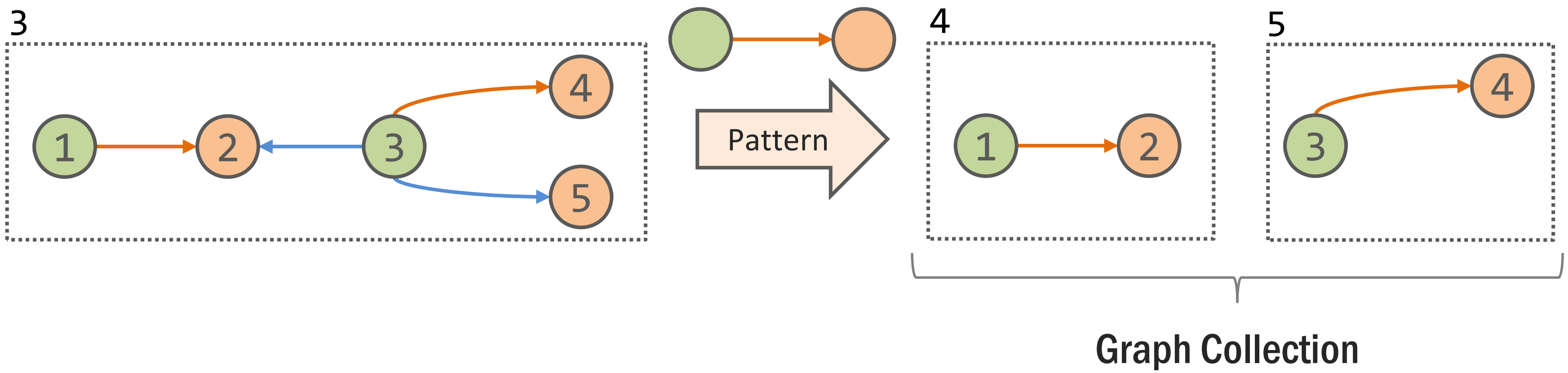
SUBGRAPH




```
LogicalGraph graph4 = graph3.subgraph((vertex => vertex[:label] == 'green'))
LogicalGraph graph5 = graph3.subgraph((edge => edge[:label] == 'blue'))
LogicalGraph graph6 = graph3.subgraph(
  (vertex => vertex[:label] == 'green'),
  (edge => edge[:label] == 'orange'))
```

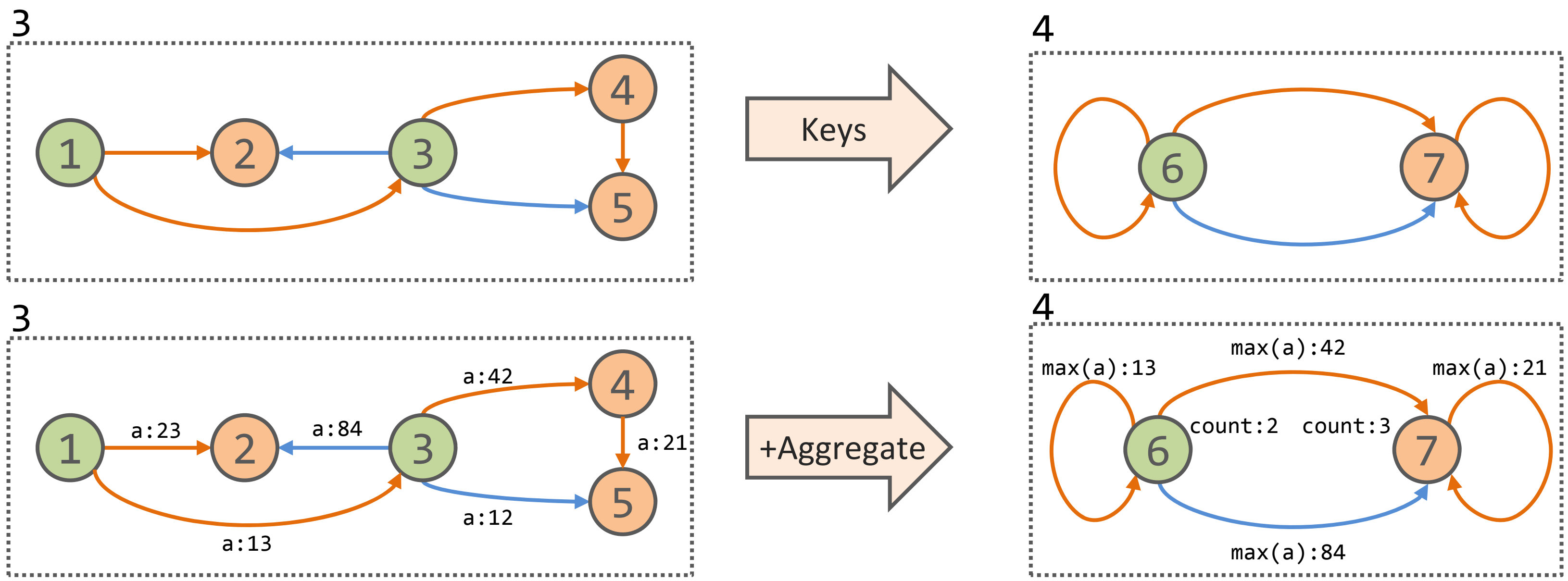
ScaDS DRESDEN LEIPZIG

PATTERN MATCHING




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

GROUPING

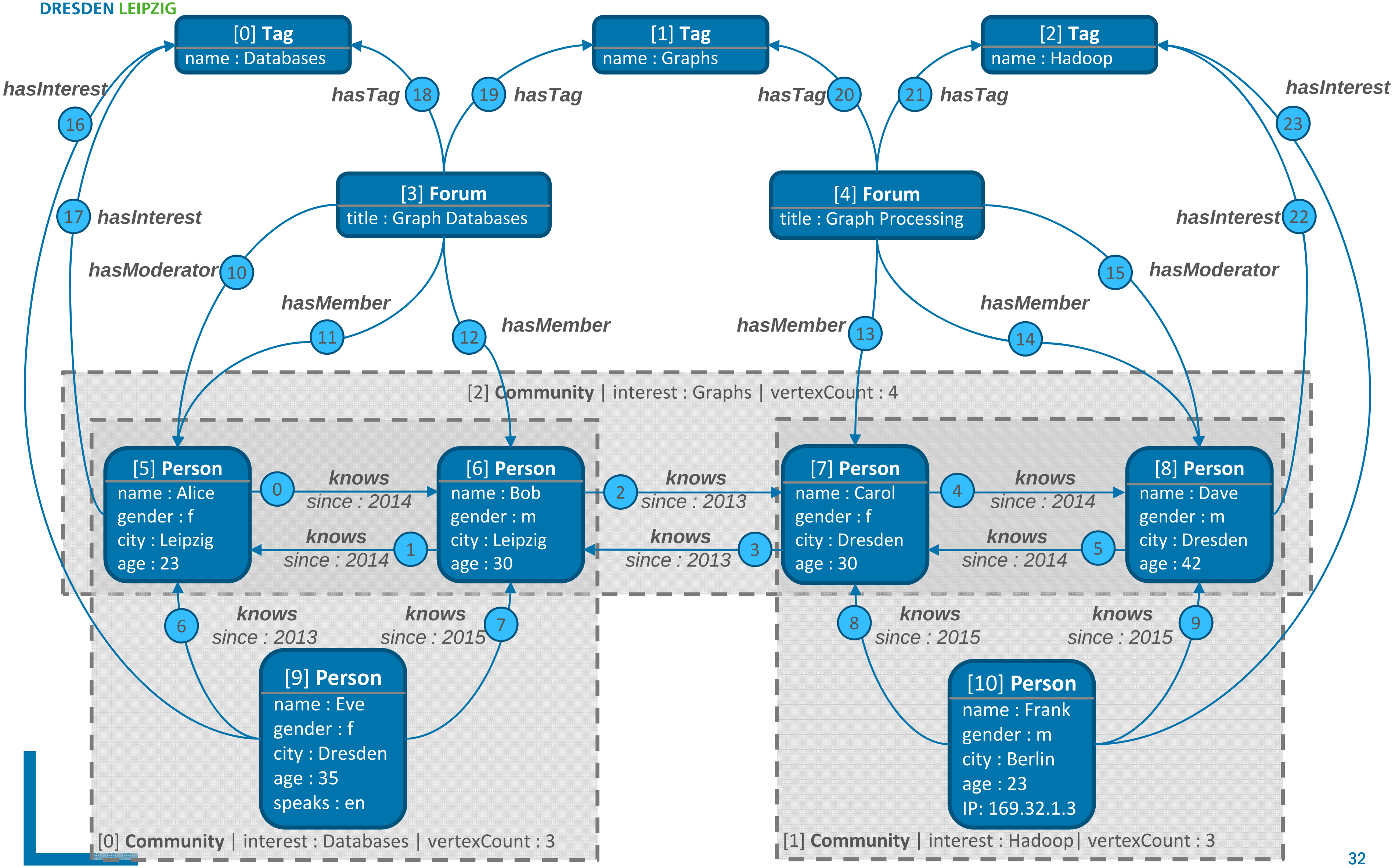



```
LogicalGraph grouped = graph3.groupBy(
  [:label], // vertex keys
  [:label]) // edge keys
LogicalGraph grouped = graph3.groupBy([:label], [COUNT()], [:label], [MAX('a')])
```

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

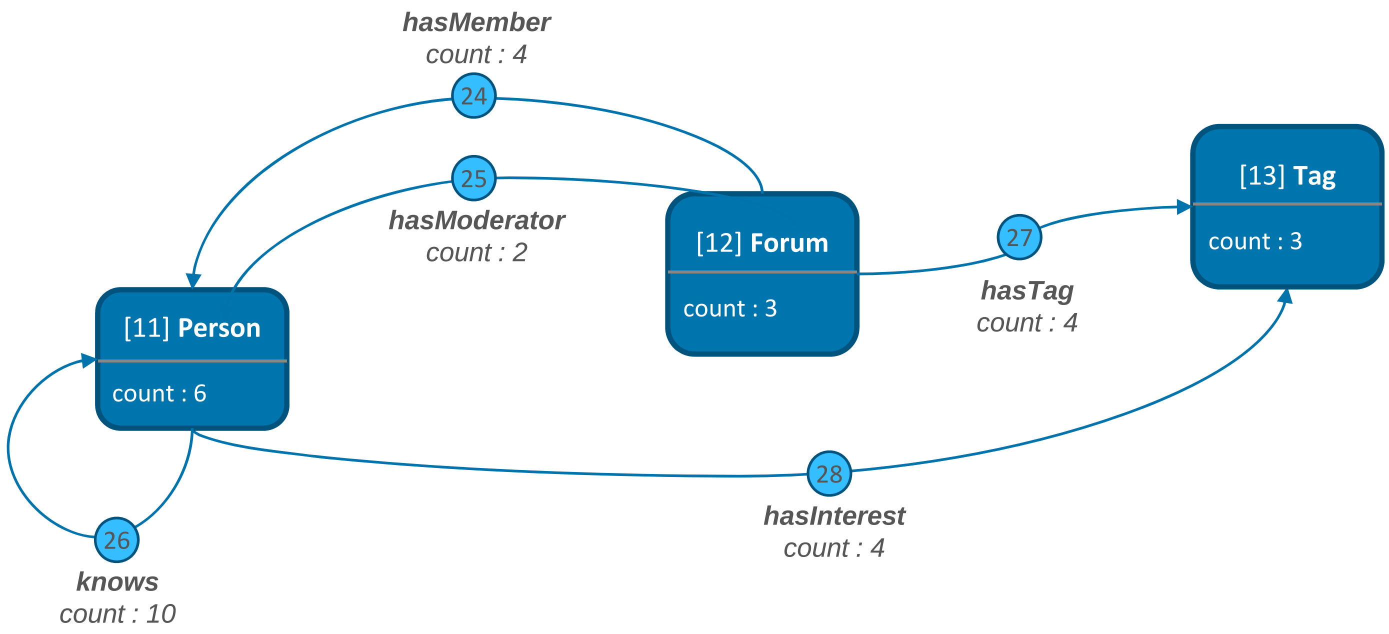
SAMPLE GRAPH



ScaDS DRESDEN LEIPZIG

GROUPING: TYPE LEVEL *(SCHEMA GRAPH)*

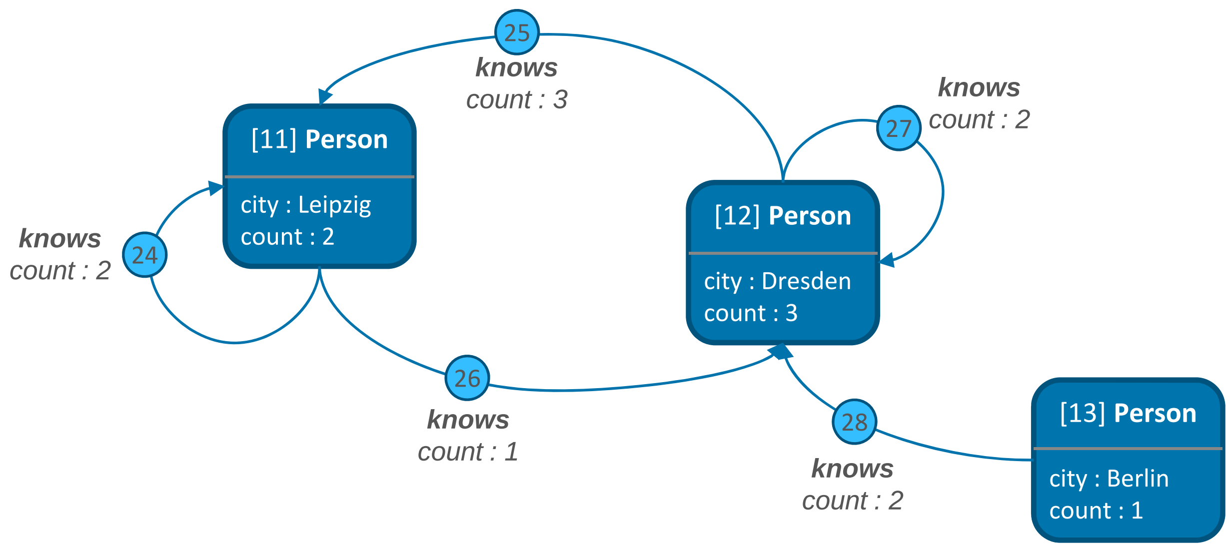
```
vertexGrKeys = [:label]
edgeGrKeys   = [:label]
sumGraph     = databaseGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
```



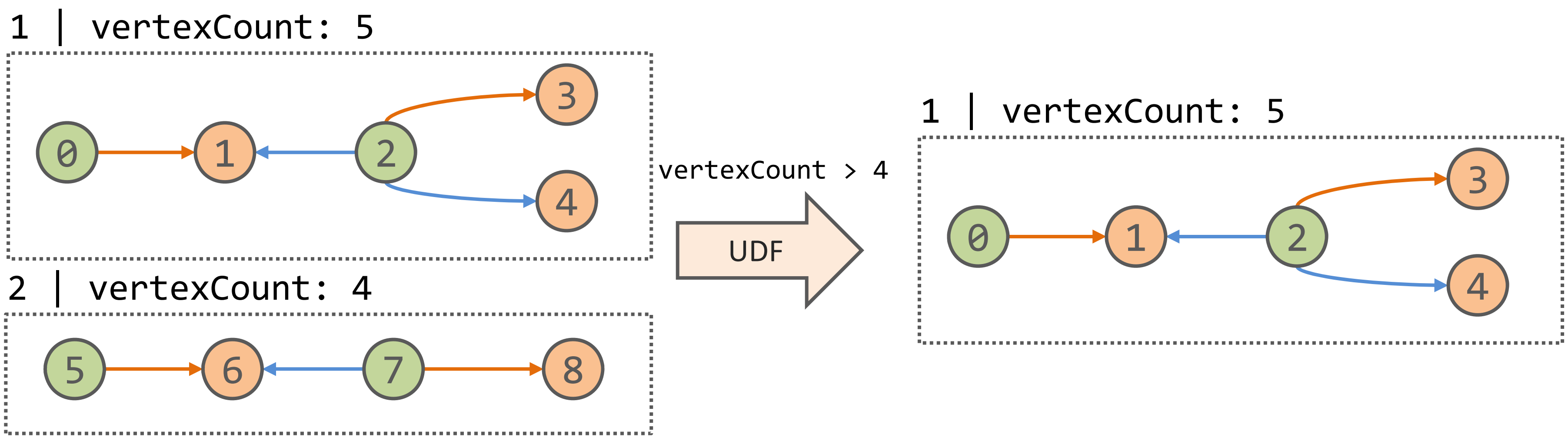
ScaDS DRESDEN LEIPZIG

GROUPING: PROPERTY-SPECIFIC

```
personGraph   = databaseGraph.subgraph((vertex => vertex[:label] == ‘Person’),
                                       (edge => edge[:label] == ‘knows’))
vertexGrKeys = [:label, “city”]
edgeGrKeys   = [:label]
sumGraph     = personGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
```



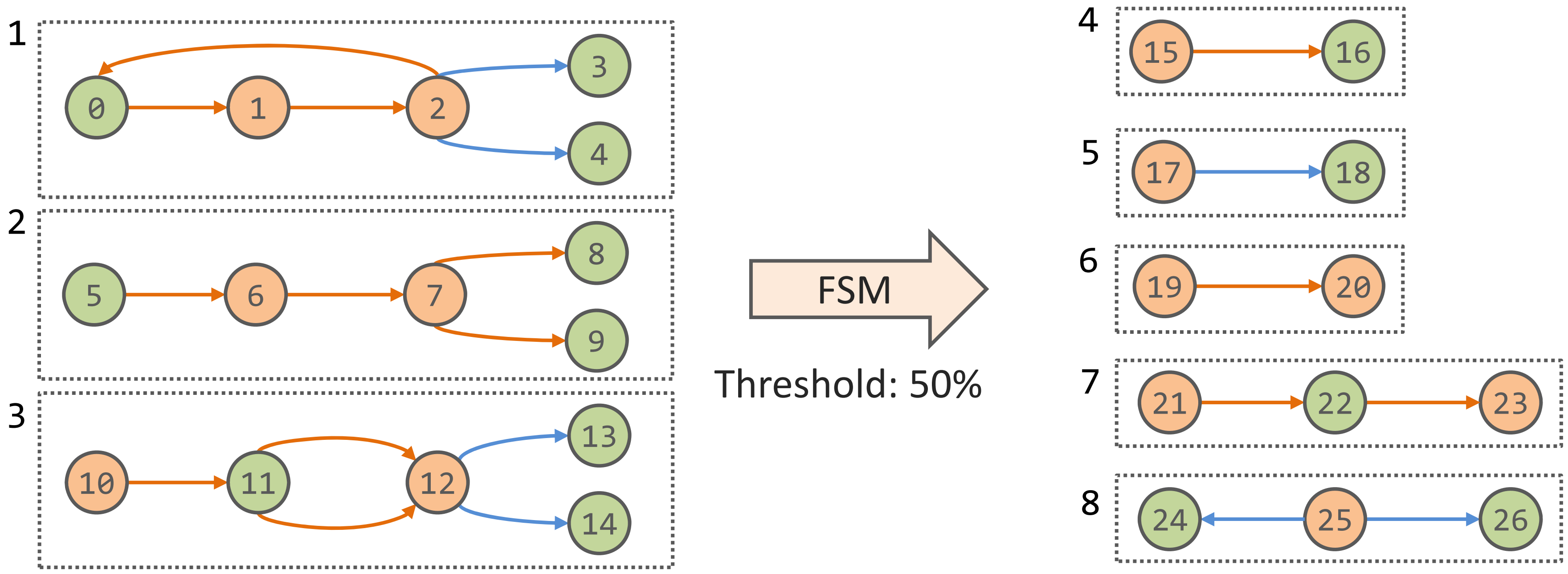
SELECTION




```
GraphCollection filtered = collection.select((graph => graph['vertexCount'] > 4));
```

ScaDS DRESDEN LEIPZIG

CALL (E.G. FREQUENT SUBGRAPHS)




```
GraphCollection frequentPatterns = collection.callForCollection(new TransactionalFSM(0.5))
```

Implementation

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>

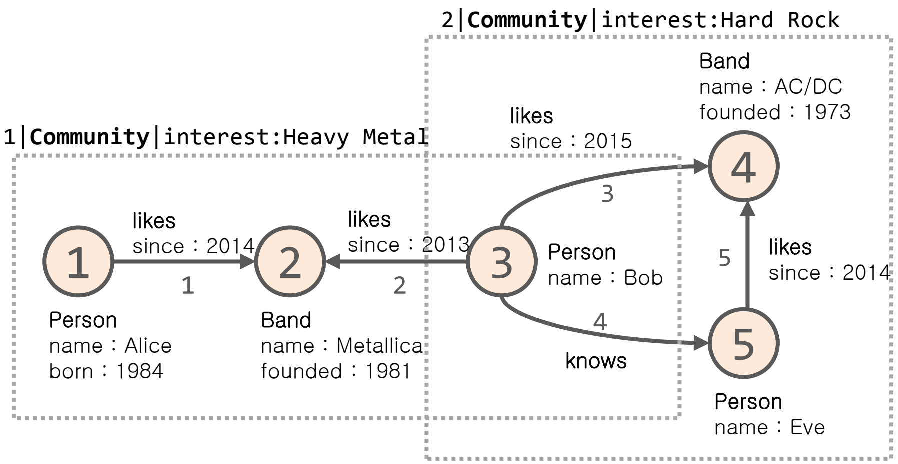
EPGMVertex

Id	Label	Properties	Graphs

```
GradoopId := UUID        String    PropertyList  := List<Property>          GradoopIdSet := Set<GradoopId>
             128-bit               Property      := (String, PropertyValue)
                                   PropertyValue := byte[]
```

ScaDS DRESDEN LEIPZIG

GRAPH REPRESENTATION: EXAMPLE



DataSet<EPGMGraphHead>

Id	Label	Properties
1	Community	{interest:Heavy Metal}
2	Community	{interest:Hard Rock}

DataSet<EPGMVertex>

Id	Label	Properties	Graphs
1	Person	{name:Alice, born:1984}	{1}
2	Band	{name:Metallica,founded:1981}	{1}
3	Person	{name:Bob}	{1,2}
4	Band	{name:AC/DC,founded:1973}	{2}
5	Person	{name:Eve}	{2}

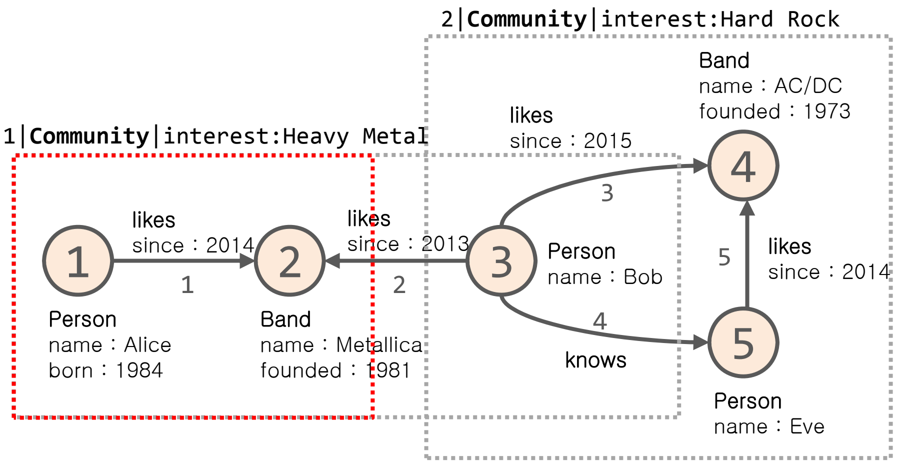
DataSet<EPGMEdge>

Id	Label	Source	Target	Properties	Graphs
1	likes	1	2	{since:2014}	{1}
2	likes	3	2	{since:2013}	{1}
3	likes	3	4	{since:2015}	{2}
4	knows	3	5	{}	{2}
5	likes	5	4	{since:2014}	{2}

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

OPERATOR IMPLEMENTATION



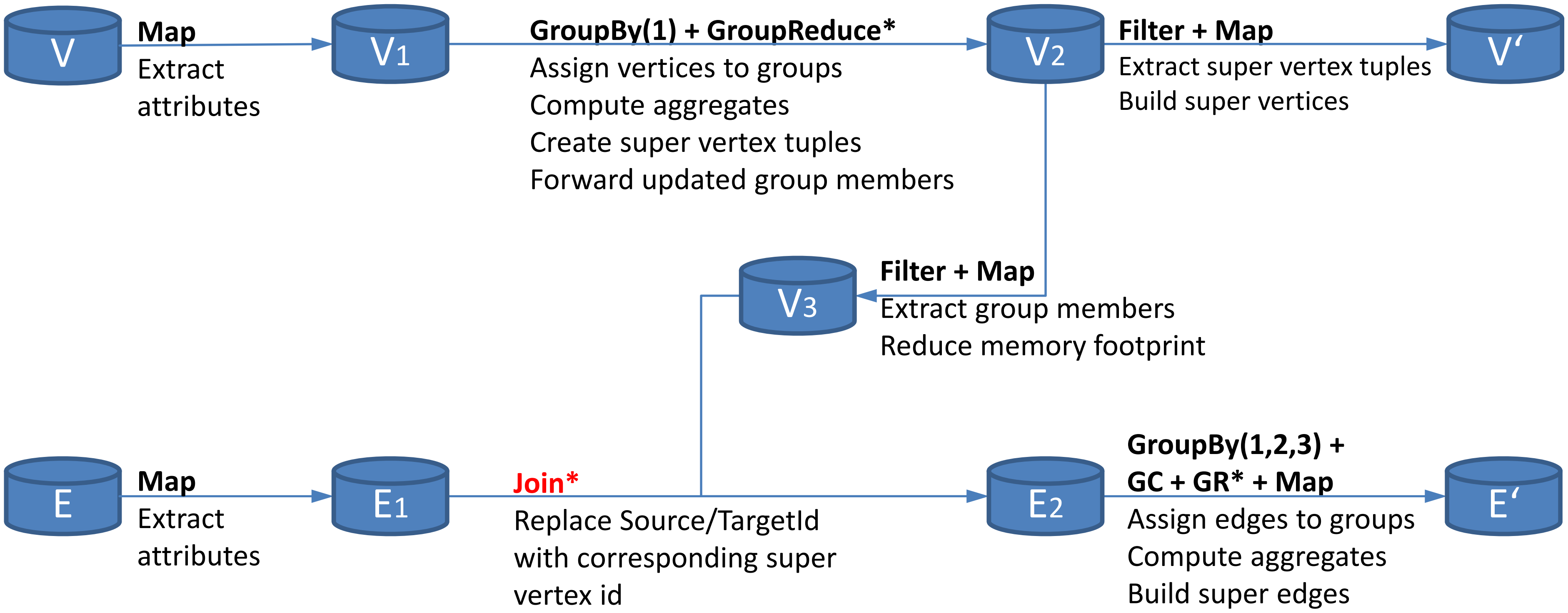
Exclusion

```
// input: firstGraph (G[1]), secondGraph (G[2])

DataSet<GradoopId> graphId = secondGraph.getGraphHead()
   .map(new Id<G>());

DataSet<V> newVertices = firstGraph.getVertices()
   .filter(new NotInGraphBroadcast<V>())
   .withBroadcastSet(graphId, GRAPH_ID);

DataSet<E> newEdges = firstGraph.getEdges()
   .filter(new NotInGraphBroadcast<E>())
   .withBroadcastSet(graphId, GRAPH_ID)
   .join(newVertices)
   .where(new SourceId<E>().equalTo(new Id<V>())
   .with(new LeftSide<E, V>())
   .join(newVertices)
   .where(new TargetId<E>().equalTo(new Id<V>())
   .with(new LeftSide<E, V>());
```

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

IMPLEMENTATION OF GRAPH GROUPING



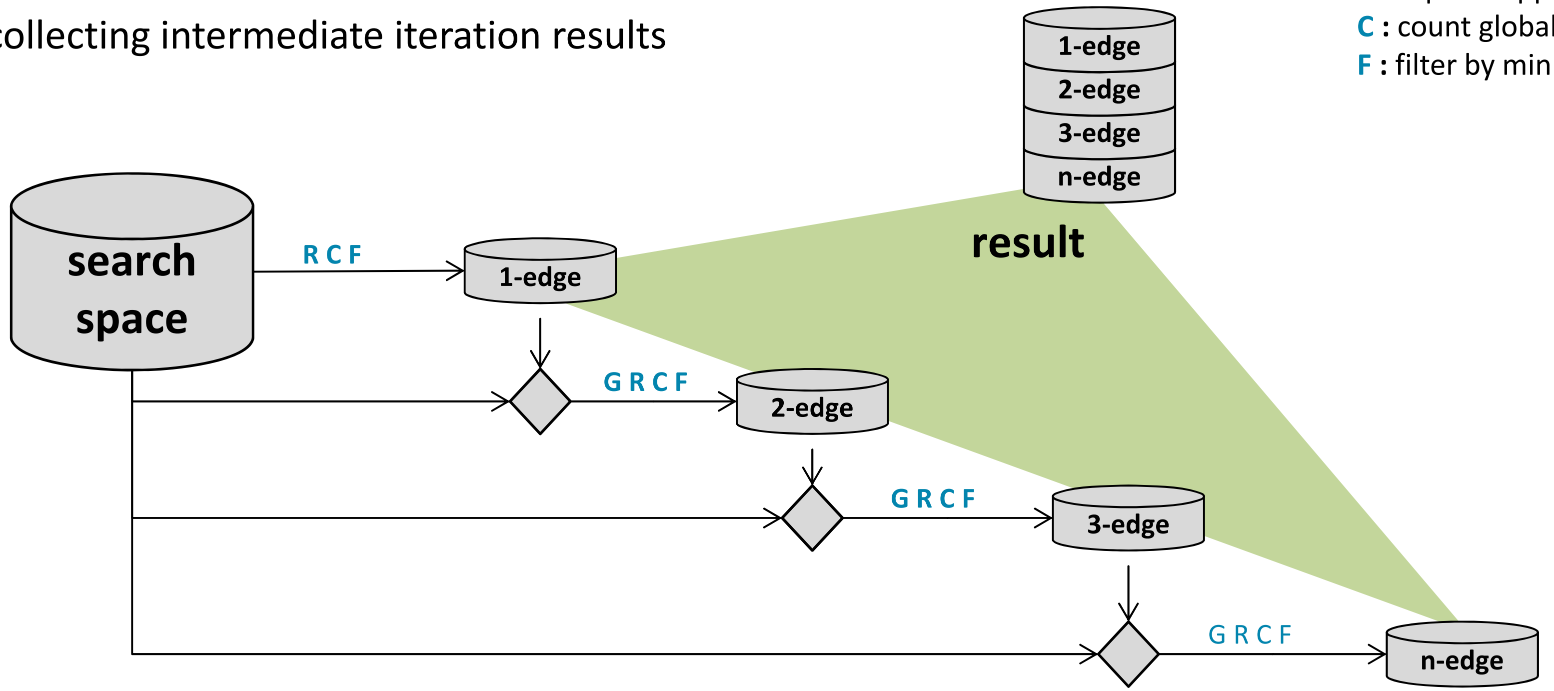
*requires worker communication

ScaDS DRESDEN LEIPZIG

ITERATIVE COMPUTATION OF FREQUENT SUBGRAPHS

collecting intermediate iteration results

G : grow frequent patterns
R : report supported patterns
C : count global frequency
F : filter by min frequency

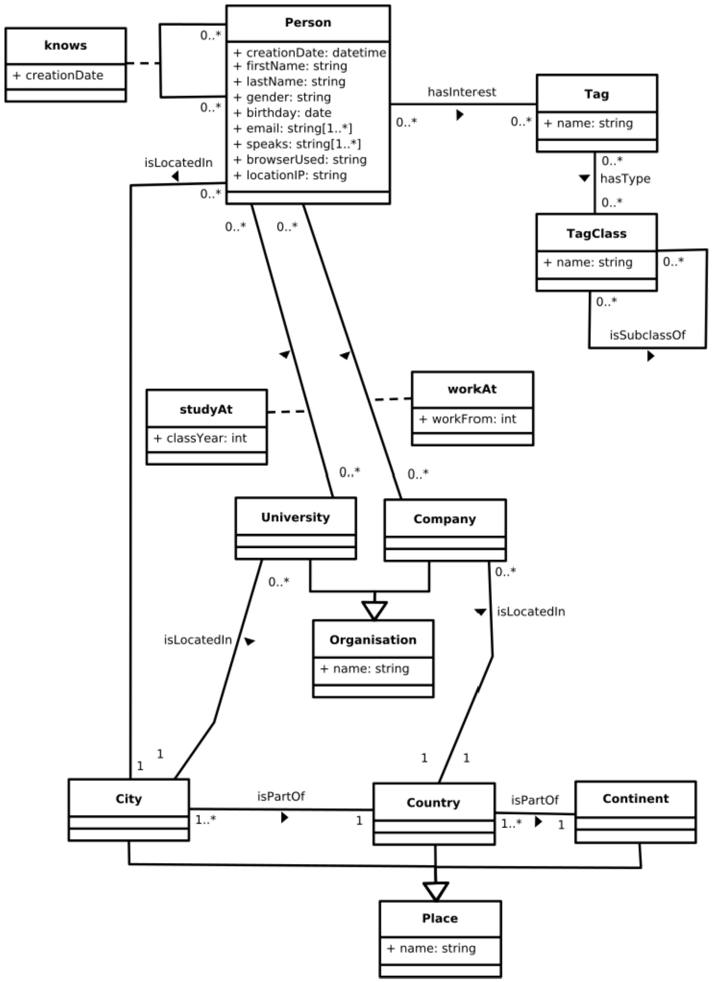


Evaluation

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

TEST WORKFLOW: SUMMARIZED COMMUNITIES



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

ScaDS DRESDEN LEIPZIG

TEST WORKFLOW: SUMMARIZED COMMUNITIES

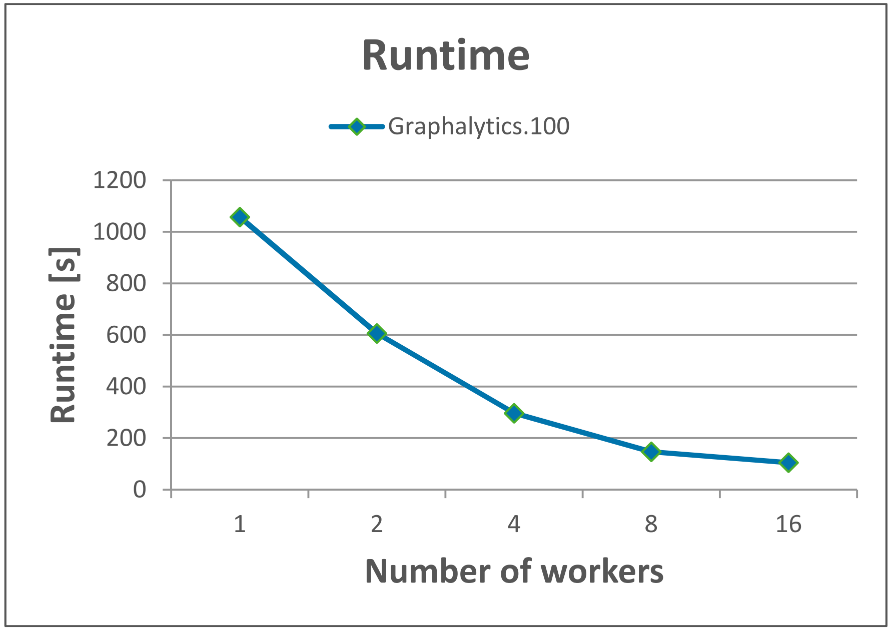
1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities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

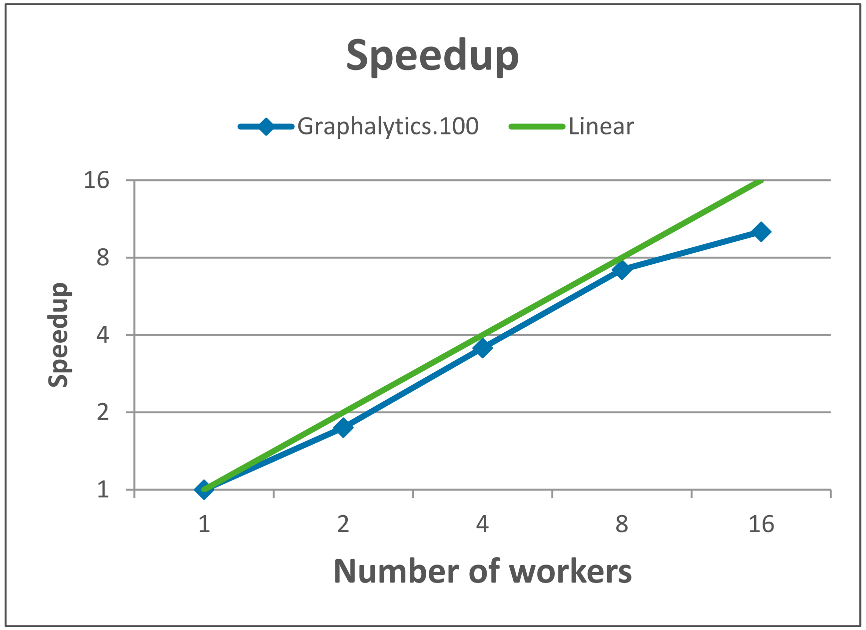
```
return socialNetwork
  // 1) extract subgraph
  .subgraph((vertex) → {
      return vertex.getLabel().toLowerCase().equals(person);
  }, (edge) → { return edge.getLabel().toLowerCase().equals(knows); })
  // project to necessary information
  .transform((current, transformed) → { return current; }, (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>());
```

https://git.io/vgozj

ScaDS DRESDEN LEIPZIG

BENCHMARK RESULTS

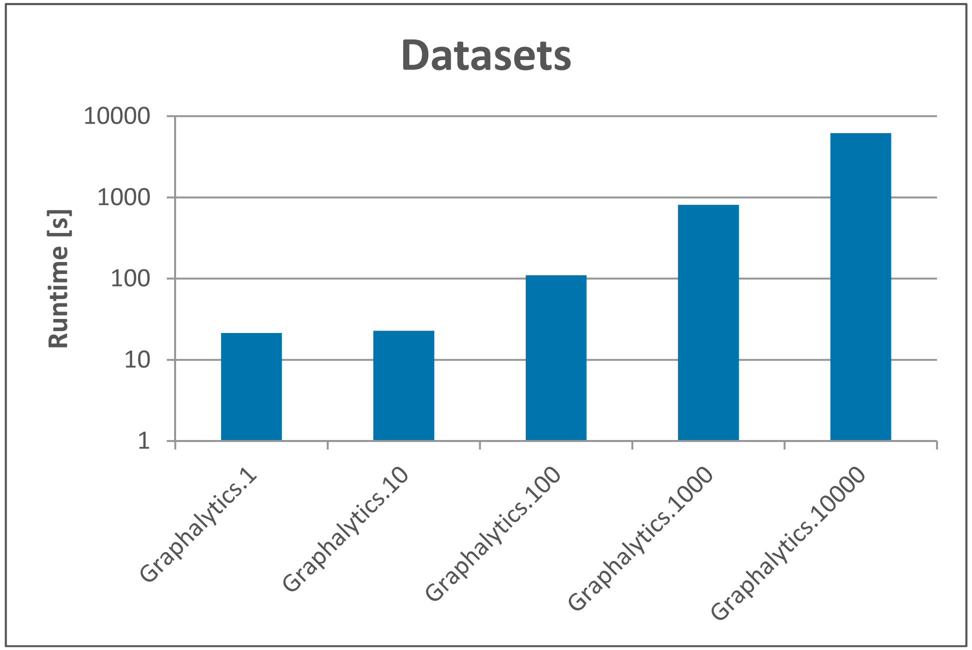




Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 16x Intel(R) Xeon(R) 2.50GHz (6 Cores)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT

BENCHMARK RESULTS 2

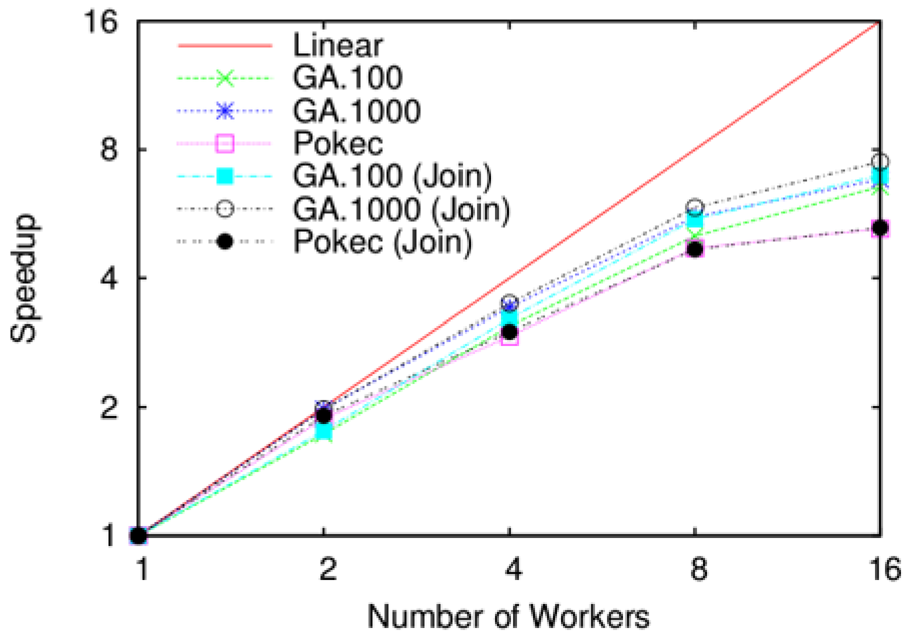


Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

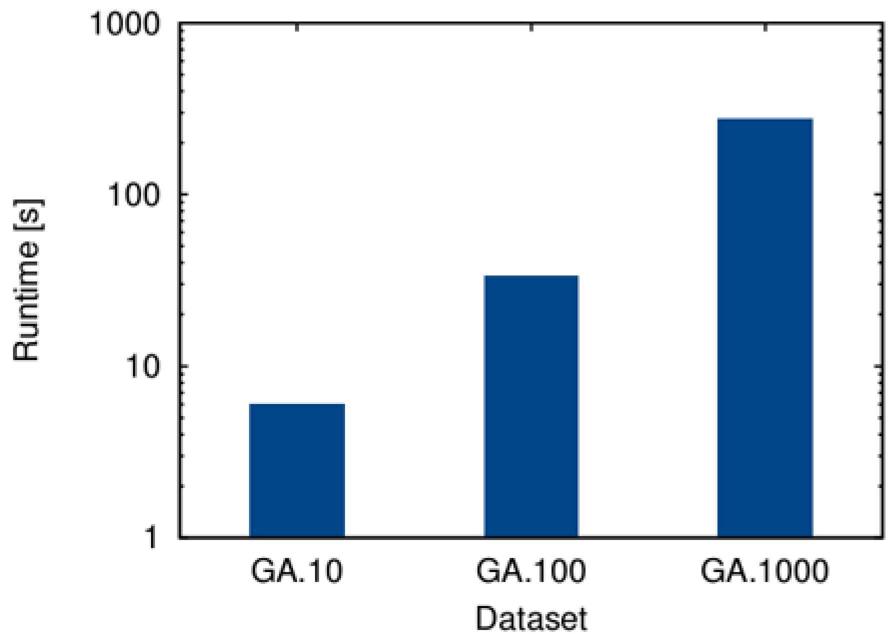
- 16x Intel(R) Xeon(R) 2.50GHz (6 Cores)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT

ScaDS DRESDEN LEIPZIG

EVALUATION OF GROUPING: SCALABILITY



Speedup for grouping on type



Runtime for grouping on type

AGENDA

- Intro Graph Analytics
  - Graph data
  - Graph databases vs graph processing systems
- Gradoop
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Use cases
  - Evaluation
- Summary/Outlook

ScaDS DRESDEN LEIPZIG

SUMMARY

- Big Graph Analytics
  - Hadoop-based graph processing frameworks based on generic graphs
  - Spark/Flink: batch/streaming-oriented workflows (rather than interactive OLAP)
  - graph collections not generally supported
  - generally missing: graph-based data integration, built-in support for dynamic graph data
- GraDoop (www.gradoop.org)
  - open-source infrastructure for entire processing pipeline: graph acquisition, storage, integration, transformation, analysis (queries + graph mining), visualization
  - extended property graph model (EPGM) with powerful operators (e.g., grouping, pattern matching) and support for graph collections
  - leverages Hadoop ecosystem
    - Apache HBase for permanent graph storage
    - Apache Flink to implement operators
  - ongoing implementation

ScaDS DRESDEN LEIPZIG

COMPARISON

	Graph Database Systems Neo4j, OrientDB	Graph Processing Systems (Pregel, Giraph)	Distributed Dataflow Systems (Flink Gelly, Spark GraphX)	gradoop
data model	rich graph models (PGM)	generic graph models	generic graph models	Extended PGM
focus	queries	analytic	analytic	analytic
query language	yes	no	no	no
graph analytics	no	yes	yes	yes
scalability	vertical	horizontal	horizontal	horizontal
Workflows	no	no	yes	yes
persistency	yes	no	no	yes
dynamic graphs / versioning	no	no	no	no
data integration	no	no	no	(yes)
visualization	(yes)	no	no	limited

LESSONS LEARNED ABOUT FLINK

- instrumental to develop Gradoop in relatively short time
- elegant and intuitive DataSet API
- very good out-of-the-box performance for non-custom types
- stumbling blocks
  - collecting intermediate results during iterations requires non-intuitive workarounds
  - missing possibility to reuse datasets in data flow programs
  - missing multicast operator with multiple outputs of possibly different types (to replace filter hierarchies causing duplication of previous ouputs)
  - missing support for theta-joins (e.g., via user-defined join predicates)
  - missing adaptive configuration of parallelism (e.g., to keep data local as long as possible)

OUTLOOK / CHALLENGES

- Graph-based data integration
  - unified approach for knowledge graphs and regular data graphs
  - holistic data integration for many sources
- Graph analytics
  - automatic optimization of analysis workflows
  - optimized graph partitioning approaches
  - visualization of graphs and analysis results
  - interactive graph analytics
  - dynamic graph data

ScaDS DRESDEN LEIPZIG

REFERENCES

- M. Junghanns, A. Petermann, K. Gomez, E. Rahm: *GRADOOP - Scalable Graph Data Management and Analytics with Hadoop*. Tech. report (Arxiv), Univ. of Leipzig, 2015
- M. Junghanns, A. Petermann, N. Teichmann, K. Gomez, E. Rahm: *Analyzing Extended Property Graphs with Apache Flink*. Proc. ACM SIGMOD workshop on Network Data Analytics (NDA), 2016
- M. Junghanns, A. Petermann, N. Neumann, E. Rahm: *Management and Analysis of Big Graph Data: Current Systems and Open Challenges.* In: Big Data Handbook (eds.: S. Sakr, A. Zomaya) , Springer, 2017 (to appear)
- A. Petermann, M. Junghanns, S. Kemper, K. Gomez, N.Teichmann, E. Rahm: *Graph Mining for Complex Data Analytics.* Proc. ICDM 2016 (Demo paper)
- A. Petermann, M. Junghanns, R. Müller, E. Rahm: *BIIIG : Enabling Business Intelligence with Integrated Instance Graphs*. Proc. 5th Int. Workshop on Graph Data Management (GDM 2014)
- A. Petermann, M. Junghanns, R. Müller, E. Rahm: *Graph-based Data Integration and Business Intelligence with BIIIG.* Proc. VLDB Conf., 2014
- A. Petermann, M. Junghanns, R. Müller, E. Rahm: *FoodBroker - Generating Synthetic Datasets for Graph-Based Business Analytics*. Proc. 5th Int. Workshop on Big Data Benchmarking (WBDB), 2014
- A. Petermann; M. Junghanns: *Scalable Business Intelligence with Graph Collections*. it - Information Technology Special Issue: Big Data Analytics, 2016