ScaDS
DRESDEN LEIPZIG

UNIVERSITÄT
LEIPZIG

SCALABLE GRAPH DATA MANAGEMENT AND ANALYTICS WITH GRADOOP

Erhard Rahm

gradoop

ScaDS DRESDEN LEIPZIG PROJECT

UNIVERSITÄT LEIPZIG

SPONSORED BY THE Federal Ministry of Education and Research

- second funding period (until 9/2021); ScaDS2
  - ScaDS started in Oct. 2014
  - BMBF funding: 4+3 years
- collaborative data science research and application
- directors: W.E. Nagel (TUD), E Rahm (UL)
- funded partners in ScaDS2:

TECHNISCHE UNIVERSITÄT DRESDEN

UNIVERSITÄT LEIPZIG

Leibniz Institute of Ecological Urban and Regional Development

CBG Max Planck Institute of Molecular Cell Biology and Genetics

HELMHOLTZ CENTRE FOR ENVIRONMENTAL RESEARCH – UFZ

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

ScaDS DRESDEN LEIPZIG

STRUCTURE SCADS2

UNIVERSITÄT LEIPZIG

Application Areas

- Life Science & E-Health
- Business Applications
- Environmental Sciences
- Matter / Energy / Chemistry
- Material Sciences/Engineering
- Digital Humanities

Service Center

Visual Analytics

- Scalable Visual Analytics
- Immersive Visual Interaction

Big Data Integration & Analytics

- Big Data Integration
- Data Analytics

Scalable and Secure Data Platforms

- Scalable Architectures
- Hardware-based Data Security

ScaDS DRESDEN LEIPZIG

SCADS RESULTS

UNIVERSITÄT LEIPZIG

- 5 survey articles on ScaDS results in database journal (March 2019)

Big Data Competence Center ScaDS Dresden/Leipzig: Overview and selected research activities

Erhard Rahm[1] · Wolfgang E. Nagel[2] · Eric Peukert[1] · René Jäkel[2] · Fabian Gärtner[1] · Peter F. Stadler[1] · Daniel Wiegreffe[1] · Dirk Zeckzer[1] · Wolfgang Lehner[2]

Received: 5 November 2018 / Accepted: 17 December 2018

Abstract
Since its launch in October 2014, the Competence Center for *Scal*able *D*ata Services and *S*olutions (ScaDS) Dresden/Leipzig carries out collaborative research on Big Data methods and their use in challenging data science applications of different domains, leading to both general, and application-specific solutions and services. In this article, we give an overview about the structure of the competence center, its primary goals and research directions. Furthermore, we outline selected research results on scalable data platforms, distributed graph analytics, data augmentation and integration and visual analytics. We also briefly report on planned activities for the second funding period (2018-2021) of the center.

Keywords Big Data · Data science · Data management

1 Introduction

The Competence Center for *Scal*able *D*ata Services and *S*olutions (ScaDS) Dresden/Leipzig [25] is one of two German Big Data competence centers that the Federal Ministry of Education and Research (BMBF) established in 2014 after a competitive selection process (the second center is the Berlin Big Data Center [4]). Initial funding has been for four years and in 2018 the BMBF extended the funding for a second phase until Sep. 2021. The funded partners in phase 1 are the two Saxonian universities TU Dresden and University of Leipzig as well as two application partners,

This work was supported by the German Federal Ministry of Education and Research (BMBF, Grant No.: 01IS14014A-D) by funding the competence center for Big Data "ScaDS Dresden/Leipzig"

Erhard Rahm
rahm@informatik.uni-leipzig.de

Wolfgang E. Nagel
wolfgang.nagel@tu-dresden.de

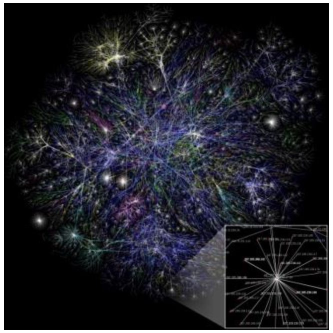
„GRAPHS ARE EVERYWHERE“

Social science

Facebook
- ca. 1.3 billion users
- ca. 340 friends per user

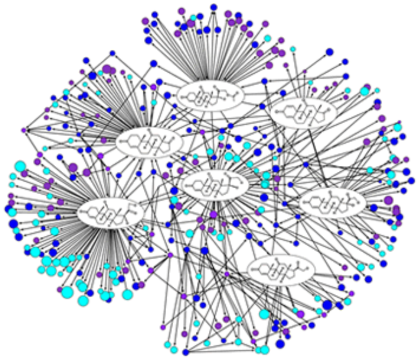
Twitter
- ca. 300 million users
- ca. 500 million tweets per day

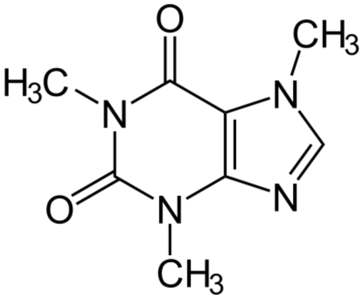
Engineering



Internet
- ca. 2.9 billion users

Life science



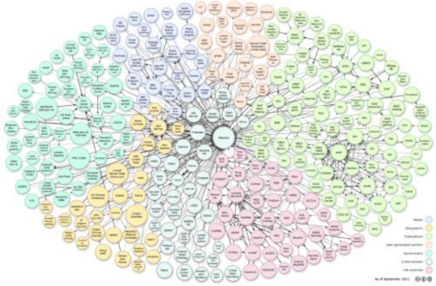


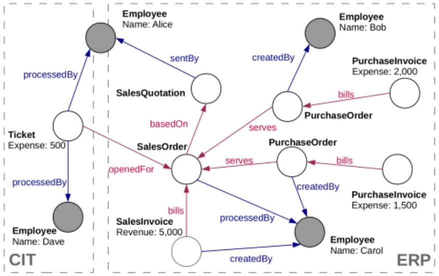
Gene (human)
- 20,000-25,000
- ca. 4 million individuals

Patients
- > 18 millions (Germany)

Illnesses
- > 30.000

Information science

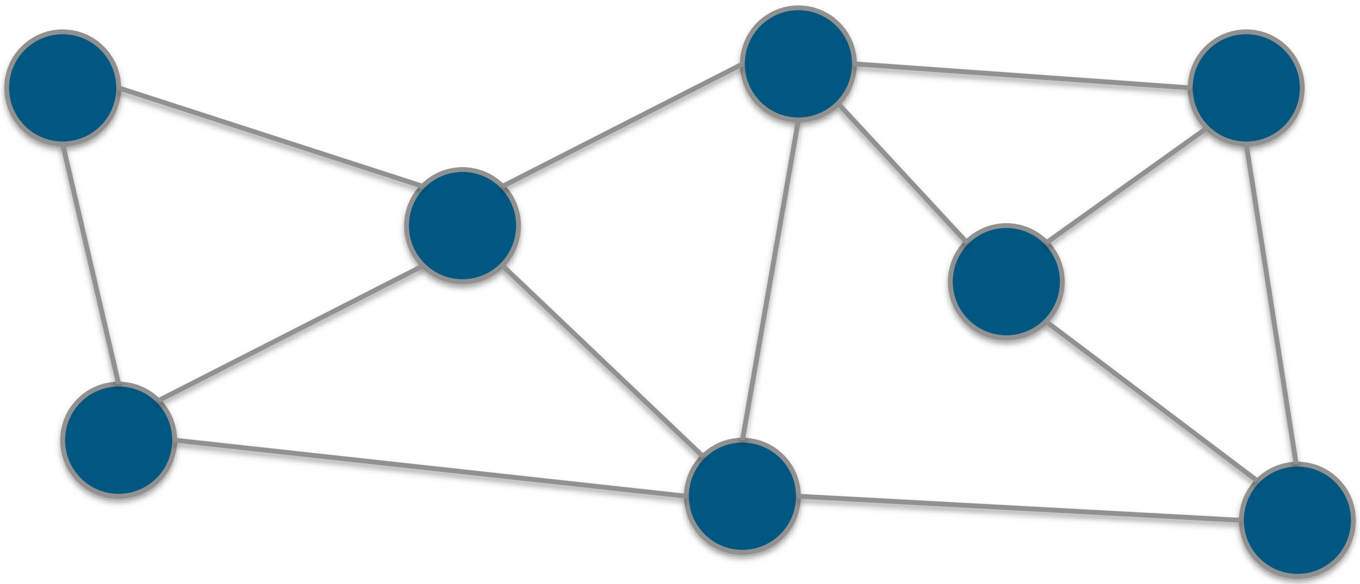




World Wide Web
- ca. 1 billion Websites

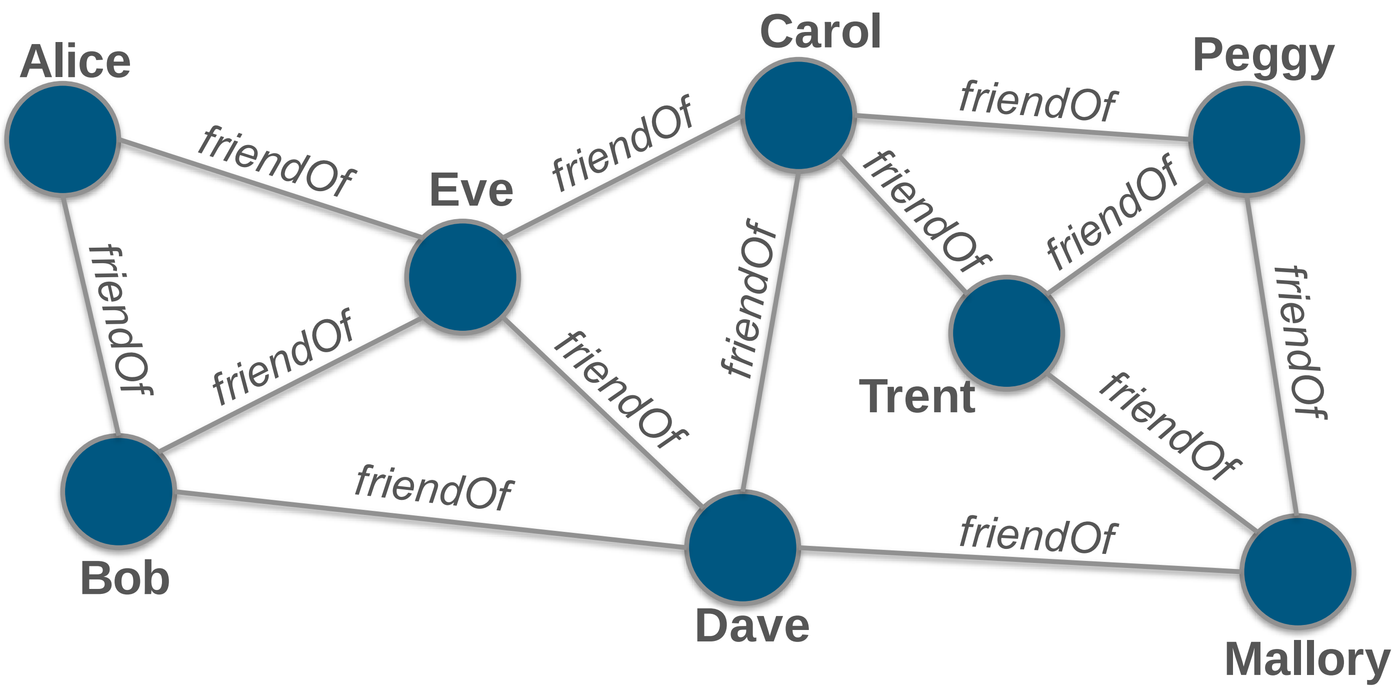
LOD-Cloud
- ca. 90 billion triples

“GRAPHS ARE EVERYWHERE”



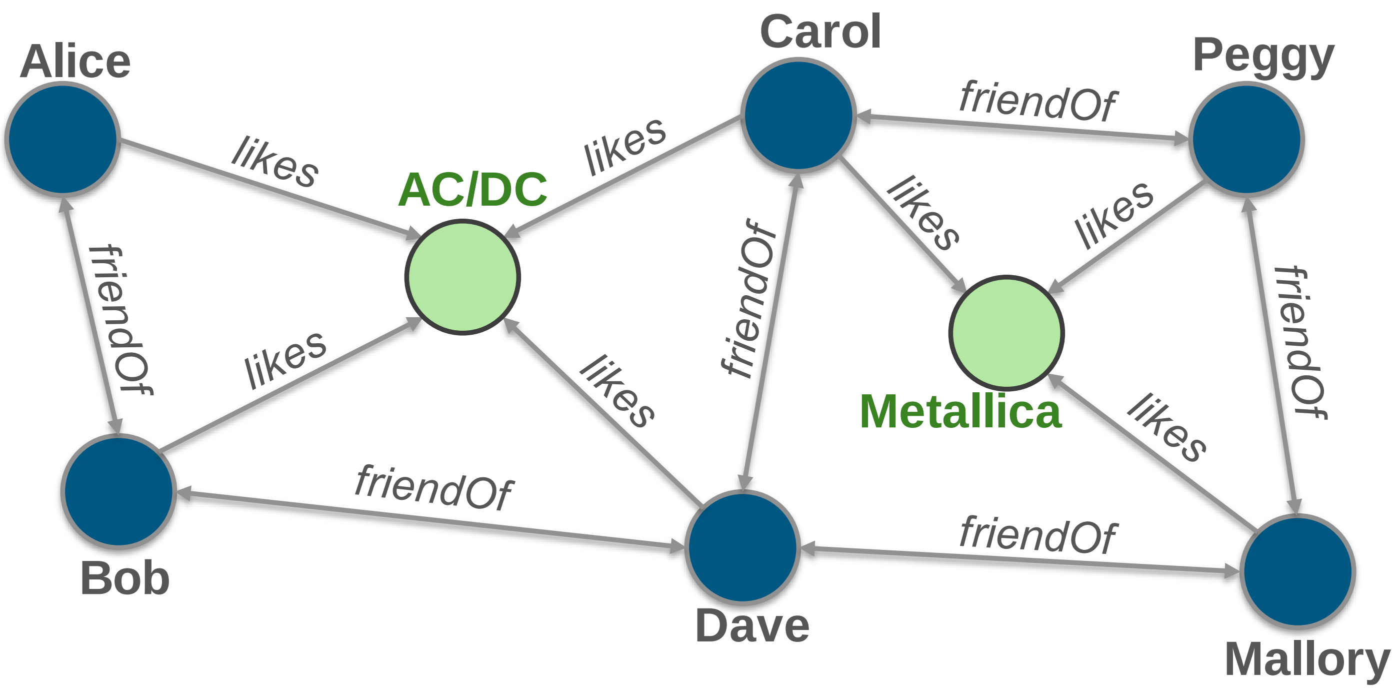
$$\boldsymbol{Graph} = (\boldsymbol{Vertices}, \boldsymbol{Edges})$$

“GRAPHS ARE EVERYWHERE”



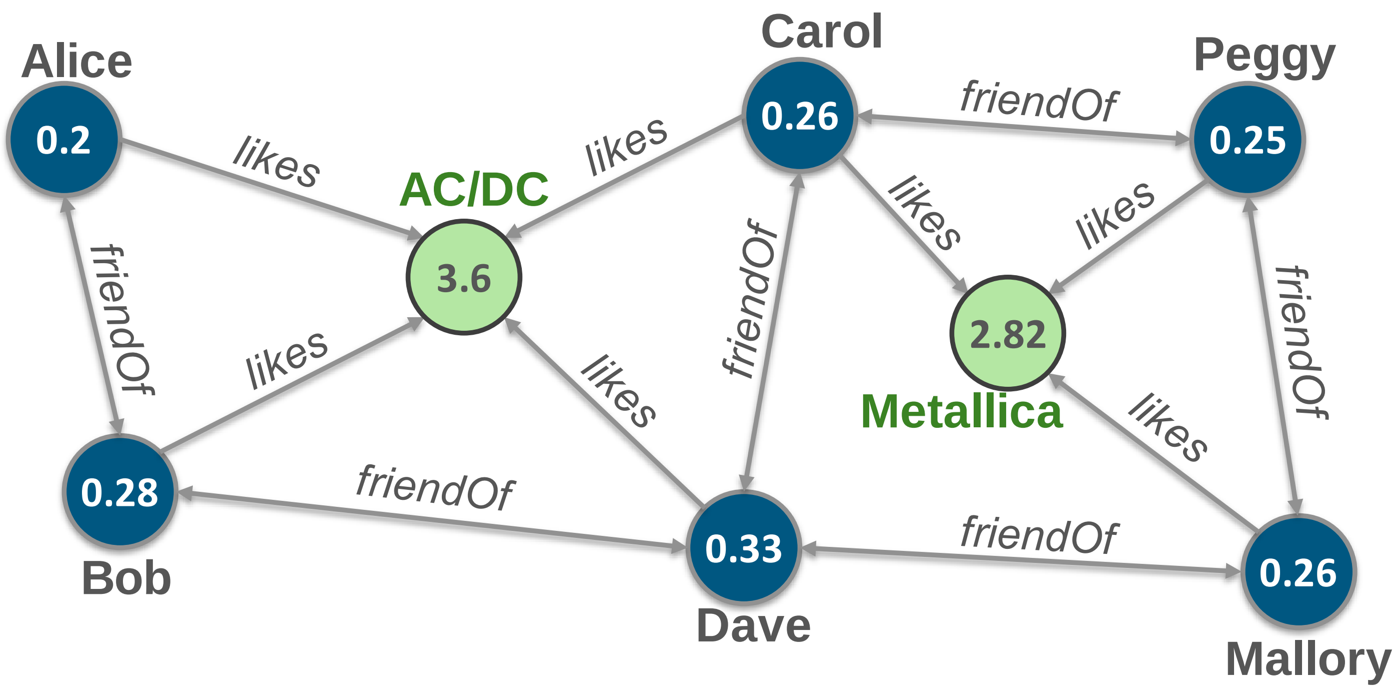
$$Graph = (\mathbf{Users}, Friendships)$$

“GRAPHS ARE HETEROGENEOUS”



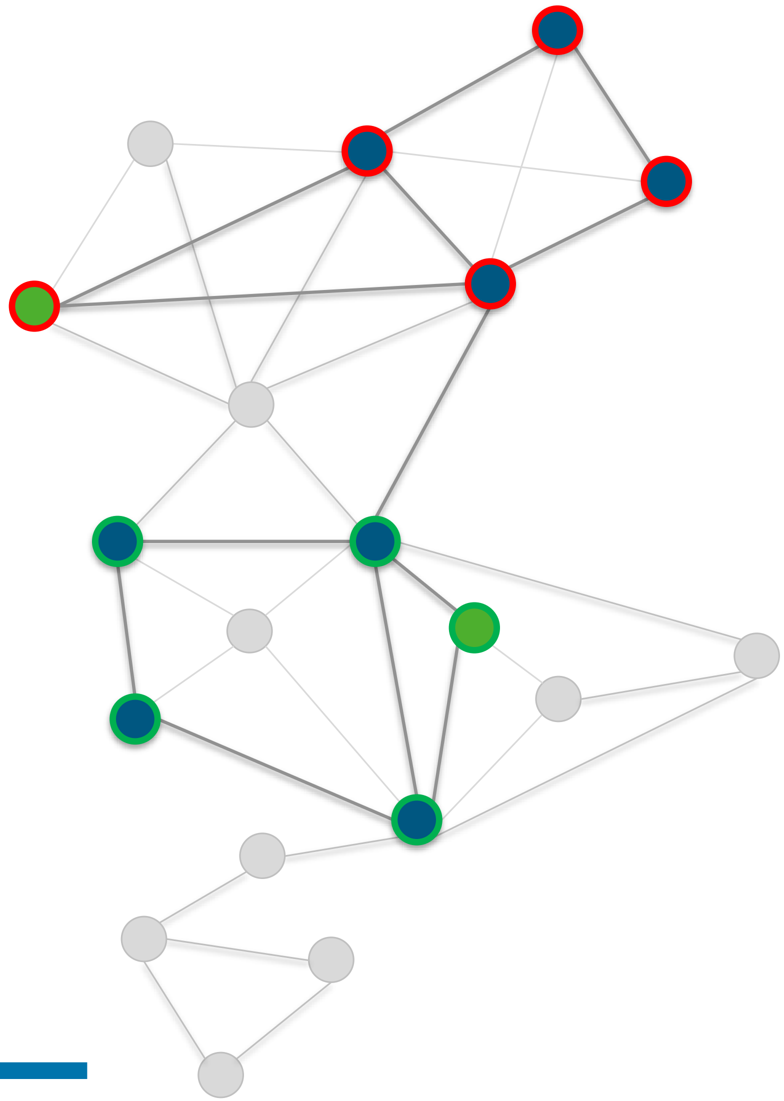
$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

“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

UNIVERSITÄT LEIPZIG

- 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
- powerful query and analysis capabilities
  - interactive, declarative graph queries
  - scalable graph mining and machine learning
- high performance and scalability
- persistent graph storage and transaction support
- graph-based integration of many data sources
- versioning and evolution (dynamic /temporal graphs)
- comprehensive visualization support

COMPARISON

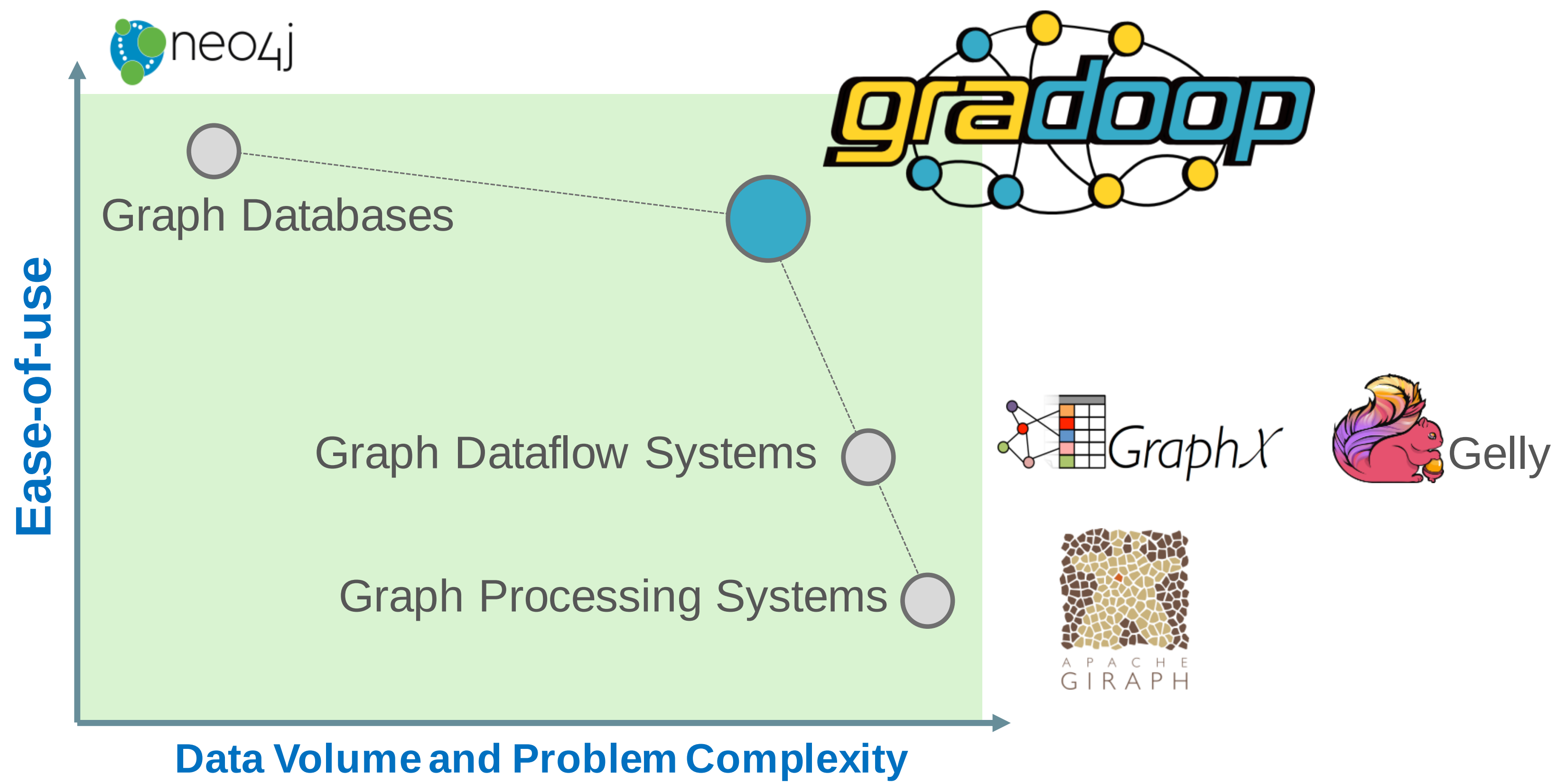
	Graph Database Systems Neo4j, OrientDB		
data model	rich graph models (PGM)		
focus	queries		
query language	yes		
graph analytics	(no)		
scalability	vertical		
analysis workflows	no		
persistency	yes		
dynamic graphs / versioning	no		
data integration	no		
visualization	(yes)		

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	
analysis workflows	no	no	
persistency	yes	no	
dynamic graphs / versioning	no	no	
data integration	no	no	
visualization	(yes)	no	

COMPARISON (3)

	Graph Database Systems Neo4j, OrientDB	Graph Processing Systems (Pregel, Giraph)	Graph 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
analysis workflows	no	no	yes
persistency	yes	no	no
dynamic graphs / versioning	no	no	no
data integration	no	no	no
visualization	(yes)	no	no


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

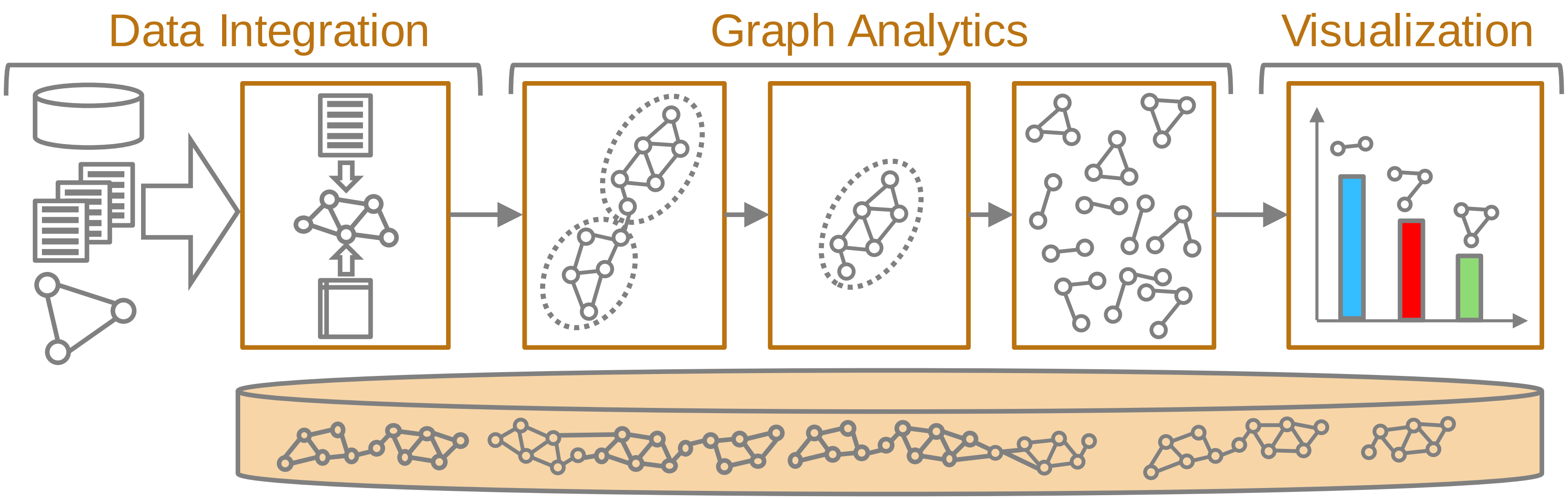
AGENDA

- ScaDS Dresden/Leipzig
- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems
- Gradoop approach
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Implementation and performance evaluation
- Ongoing work
  - Graph-based data integration
    - graph transformations
    - multi-source matching (FAMER)
  - Temporal graphs
- Conclusions

GRADOOP CHARACTERISTICS

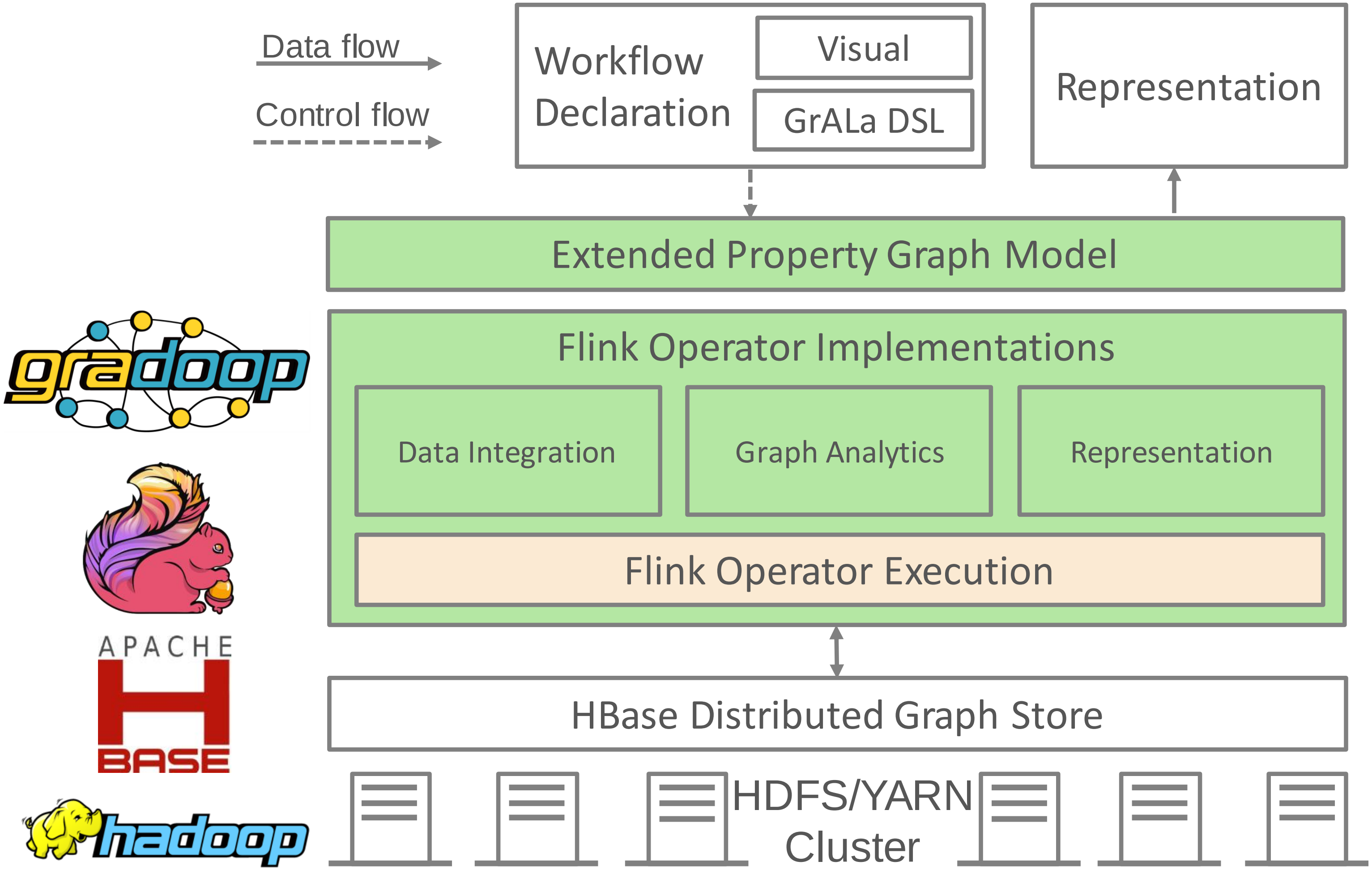
- Hadoop-based framework for graph data management and analysis
  - persistent graph storage in scalable distributed store (Hbase or Accumulo)
  - utilization of Apache Flink for parallel, in-memory processing
- Extended property graph data model (EPGM)
  - operators on graphs and sets of (sub) graphs
  - support for semantic graph queries (grouping, pattern matching/Cypher, ...)
  - support for graph mining (frequent subgraph mining, clustering, ...)
- 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
- integration into KNIME

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**

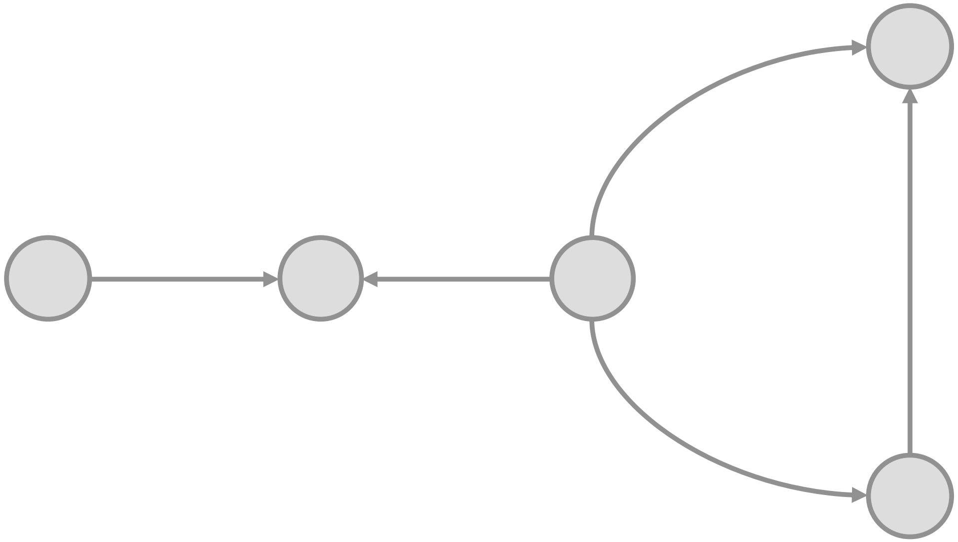
HIGH LEVEL ARCHITECTURE



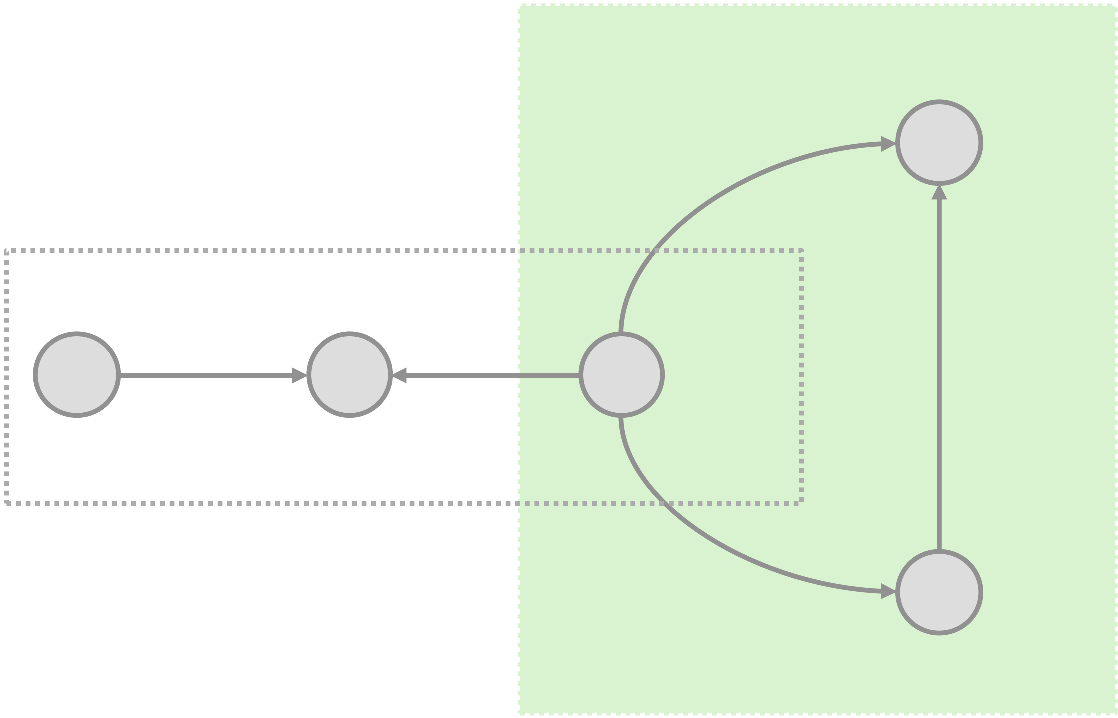
EXTENDED PROPERTY GRAPH MODEL (EPGM)

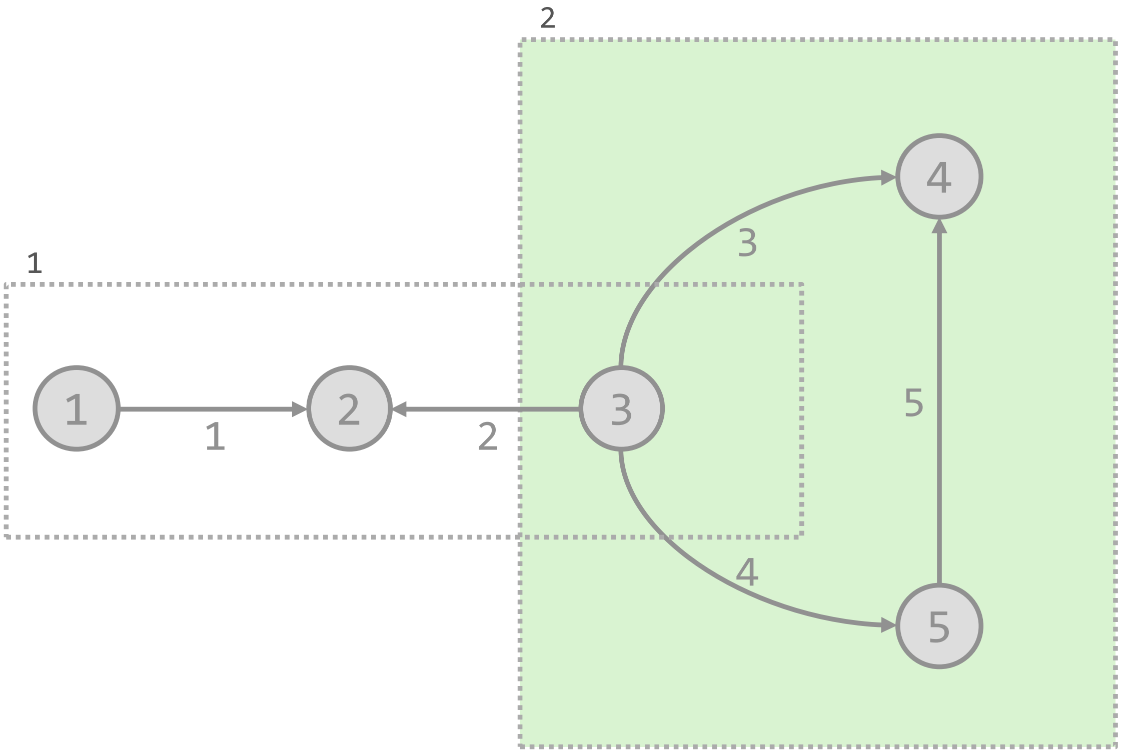
- includes PGM as special case
- support for collections of logical graphs / subgraphs
  - can be defined explicitly
  - can be result of graph algorithms / operators
- support for graph properties
- powerful operators on both graphs and graph collections
- Graph Analytical Language – GrALa
  - domain-specific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

- **Vertices and directed Edges**

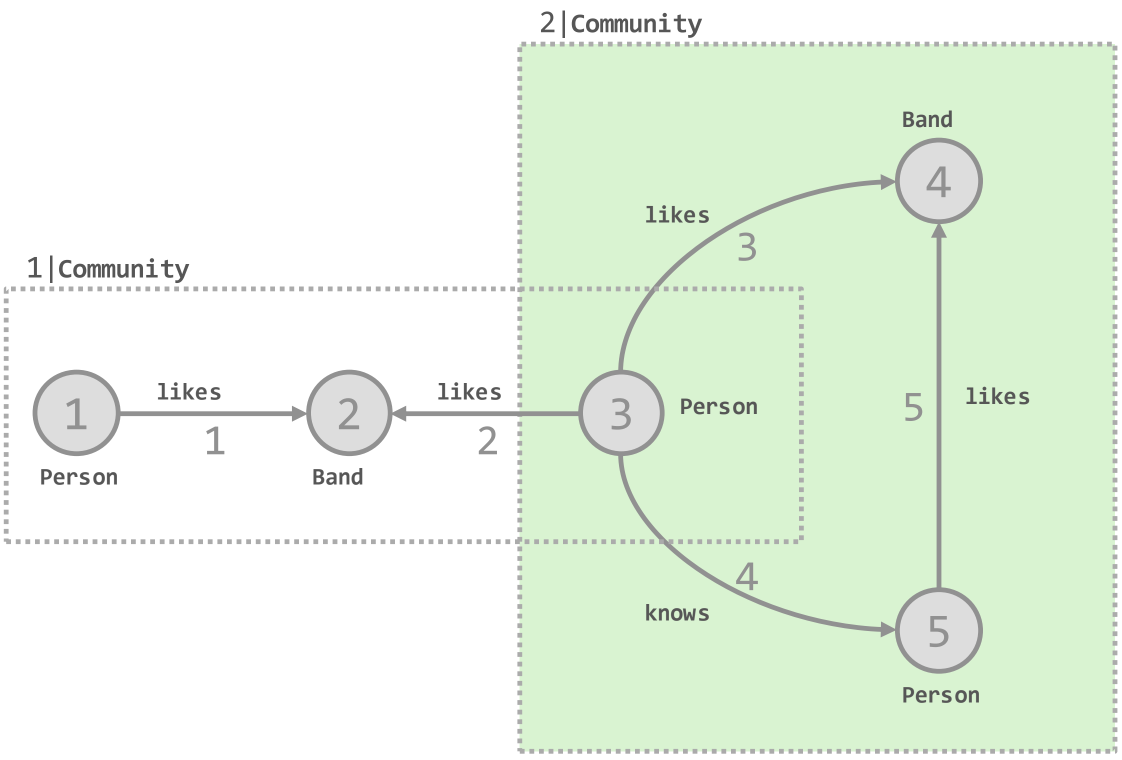


- Vertices and directed Edges
- **Logical Graphs**

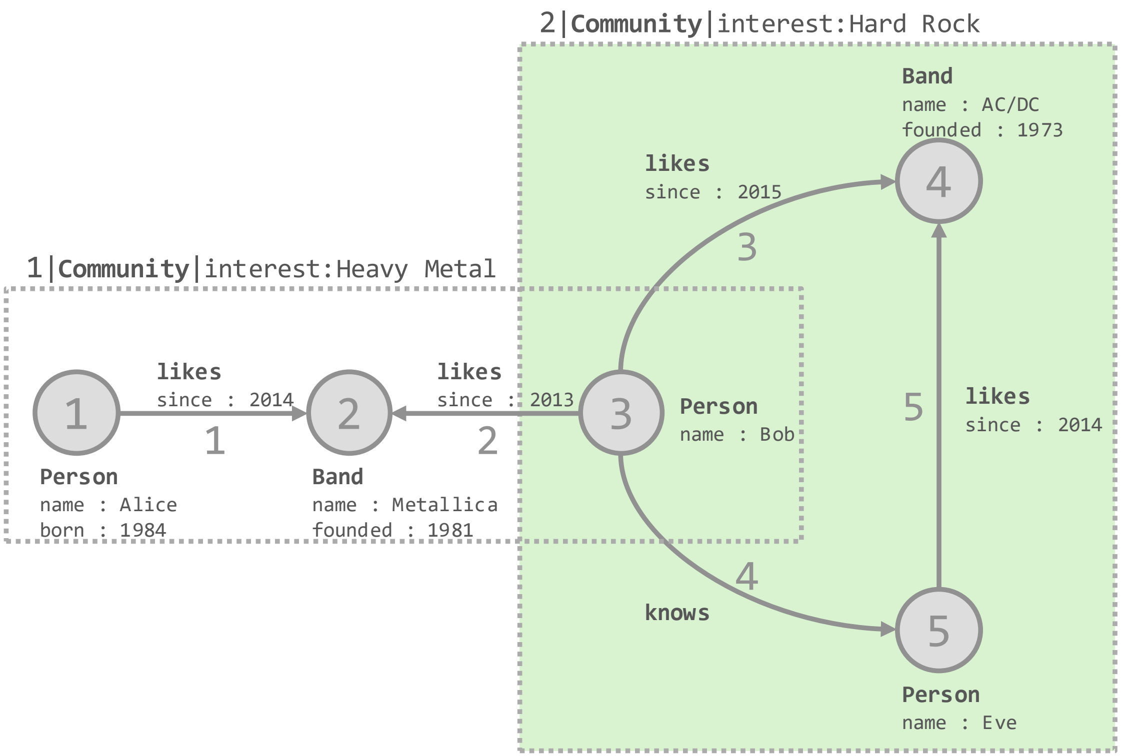




- Vertices and directed Edges
- Logical Graphs
- **Identifiers**



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

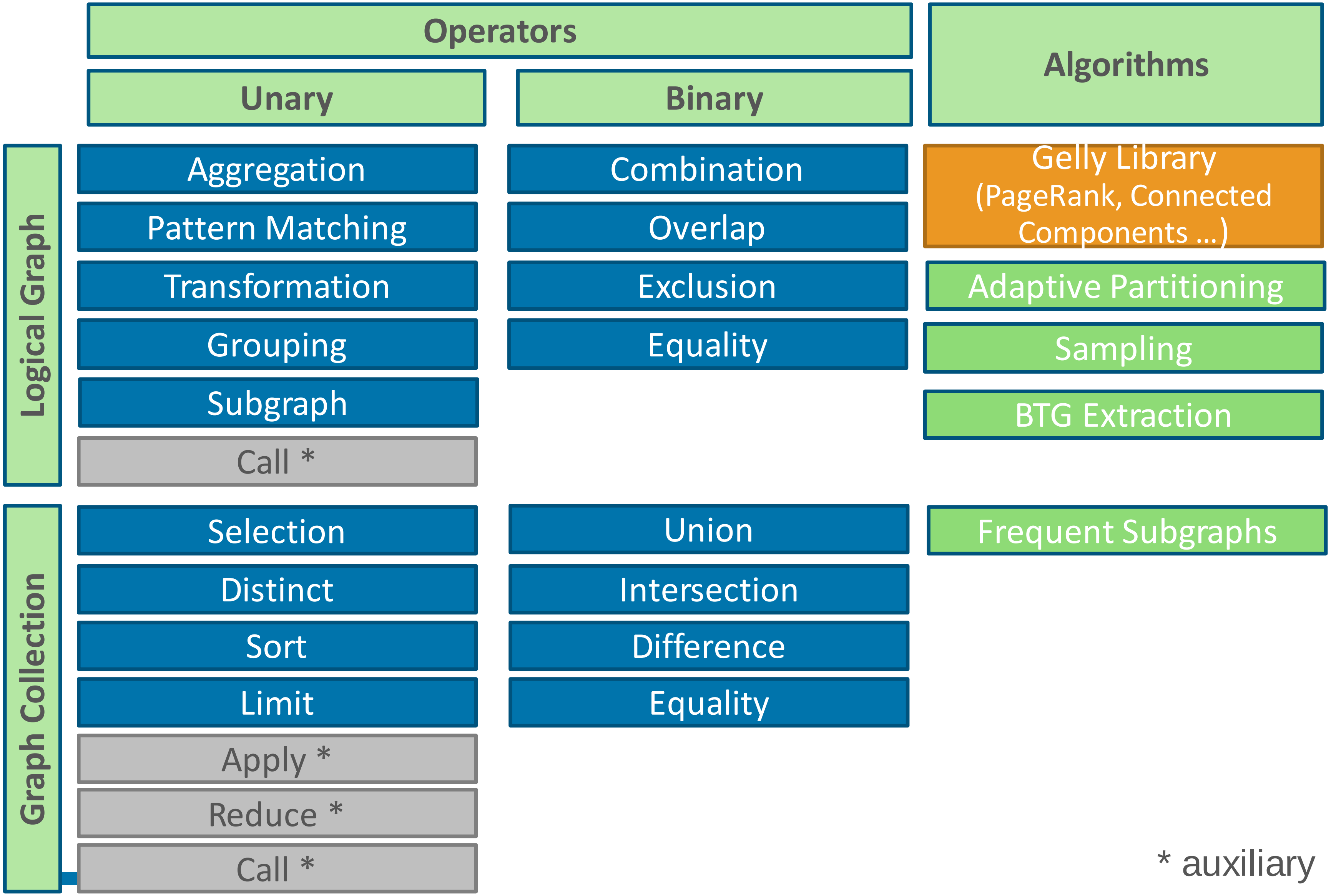


- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- **Properties**

Operators

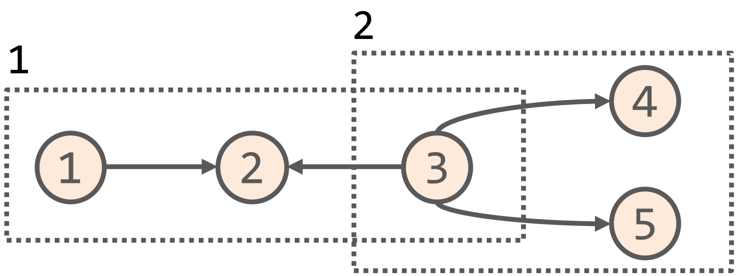
ScaDS DRESDEN LEIPZIG

UNIVERSITÄT LEIPZIG

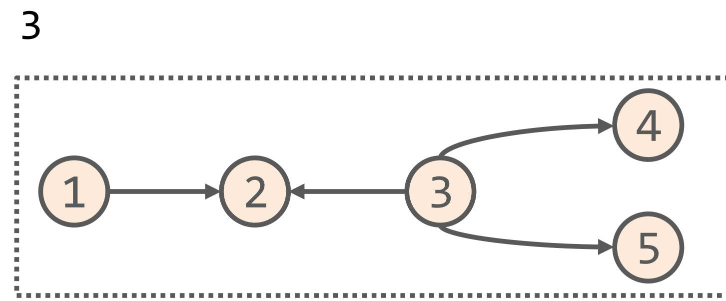


* auxiliary

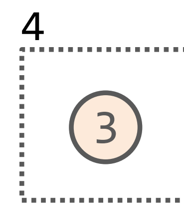
BASIC BINARY OPERATORS



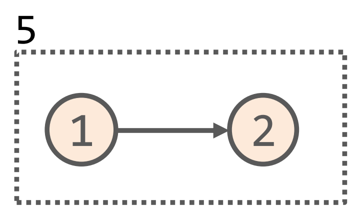
Combination



Overlap

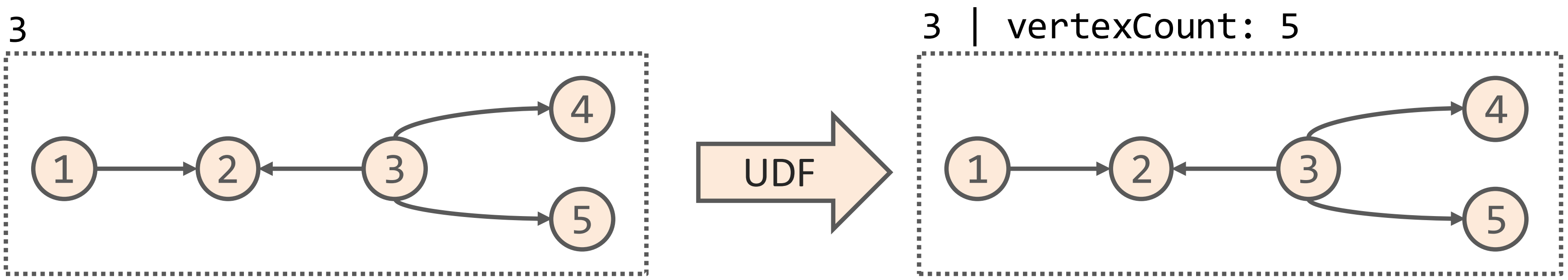


Exclusion



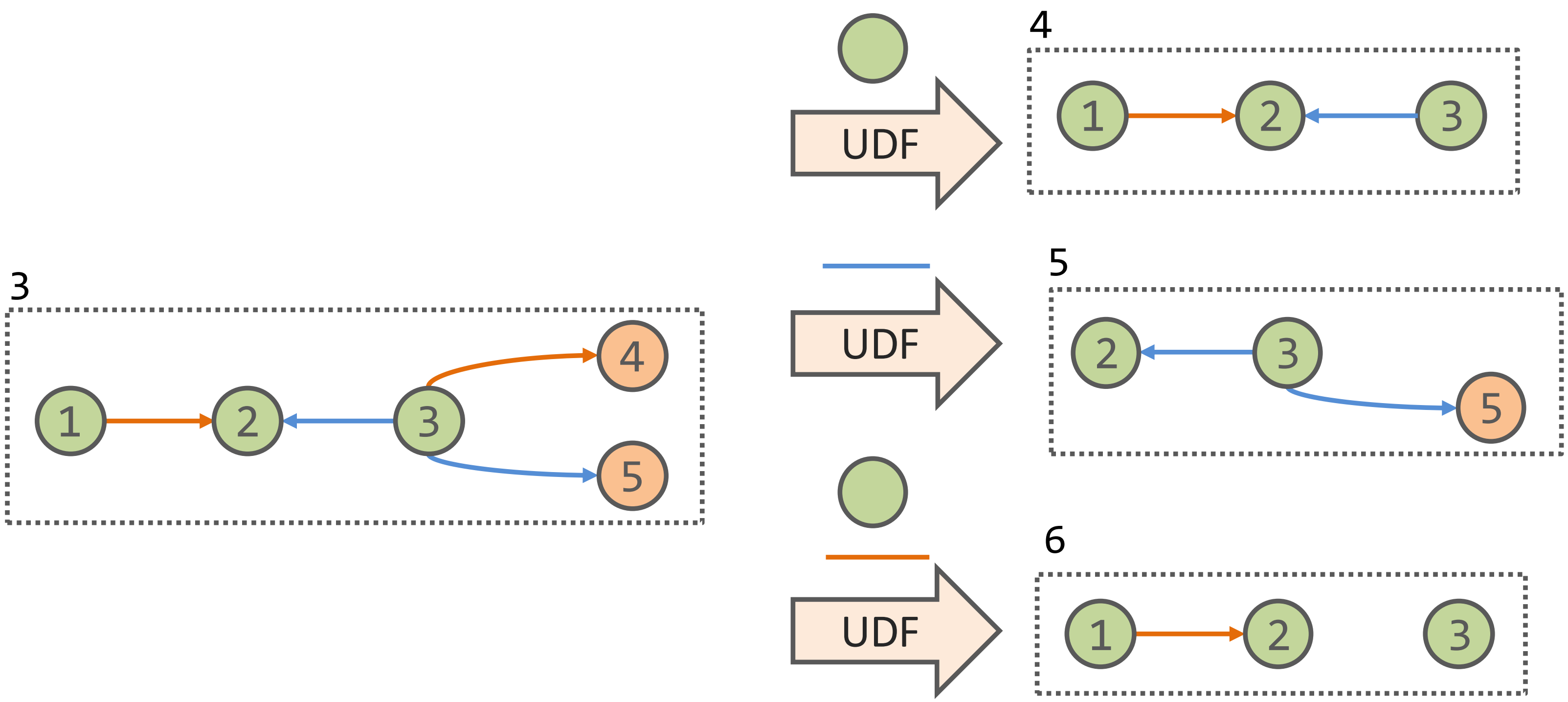

```
LogicalGraph graph3 = graph1.combine(graph2);
LogicalGraph graph4 = graph1.overlap(graph2);
LogicalGraph graph5 = graph1.exclude(graph2);
```

AGGREGATION



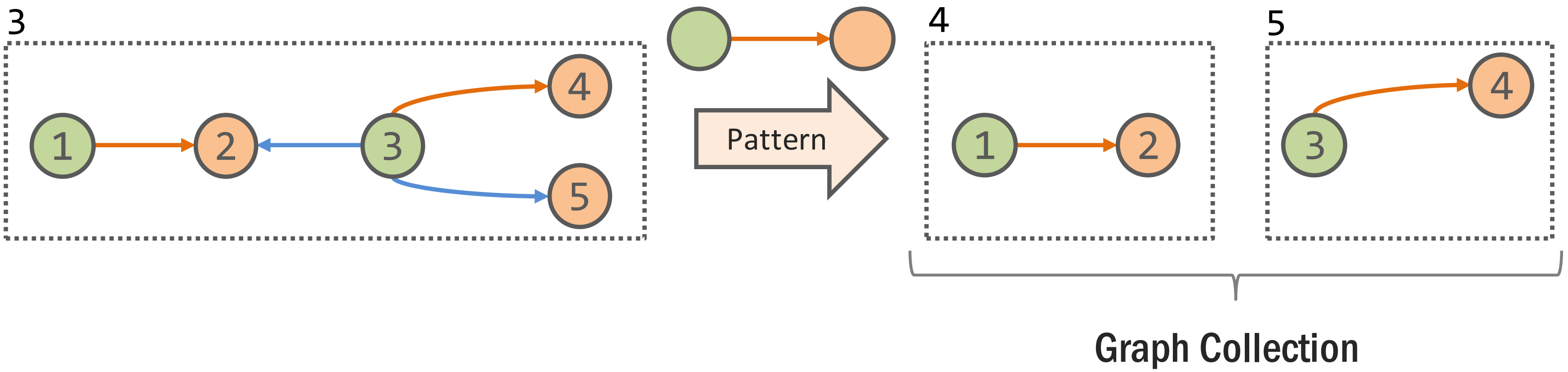

```
udf = (graph => graph['vertexCount'] = graph.vertices.size())
graph3 = graph3.aggregate(udf)
```

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’))
```

PATTERN MATCHING



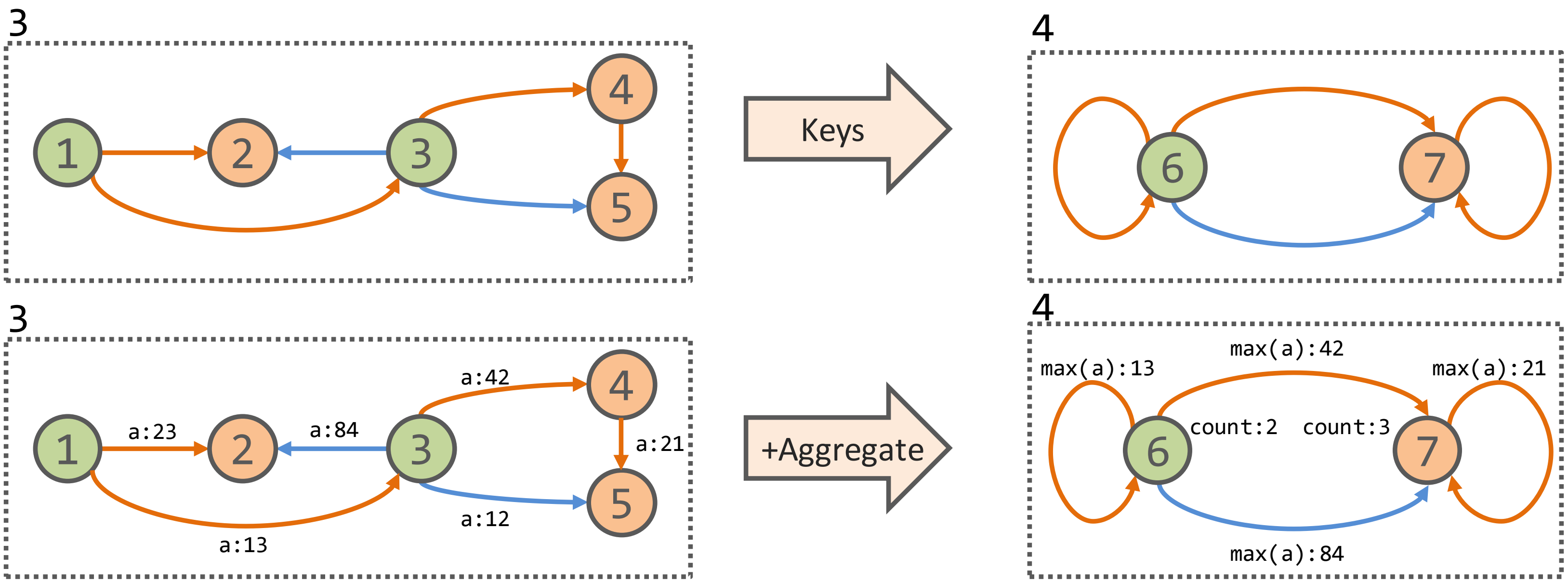

```
GraphCollection collection = graph3.match("(:Green)-[:orange]->(:Orange)");
```

- support of *Cypher query language* for pattern matching*

```
q =  "MATCH (p1: Person ) -[e: knows *1..3] ->( p2: Person)
       WHERE p1.gender <> p2 .gender RETURN *"
GraphCollection matches = g.cypher (q)
```

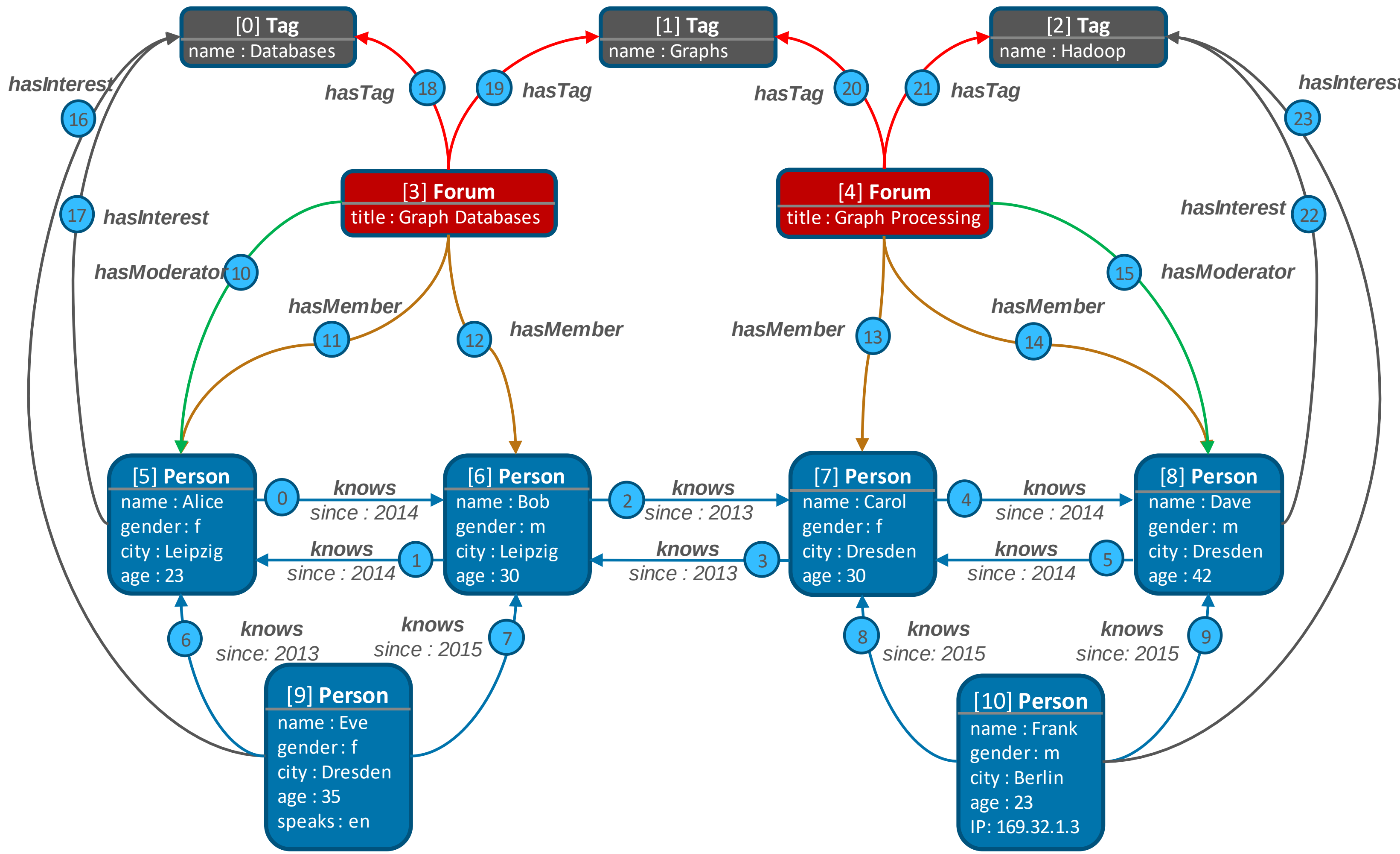
* Junghanns et al.: *Cypher-based Graph Pattern Matching in Gradoop*. Proc. GRADES 2017

GROUPING



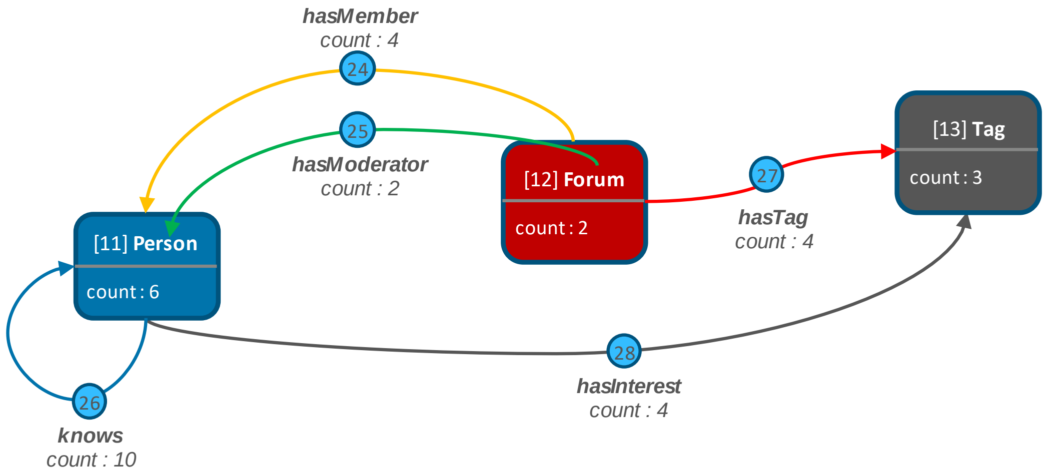

```
LogicalGraph grouped = graph3.groupBy(
  [:label], // vertex keys
  [:label]) // edge keys
LogicalGraph grouped = graph3.groupBy([:label], [COUNT()], [:label], [MAX('a')])
```

SAMPLE GRAPH



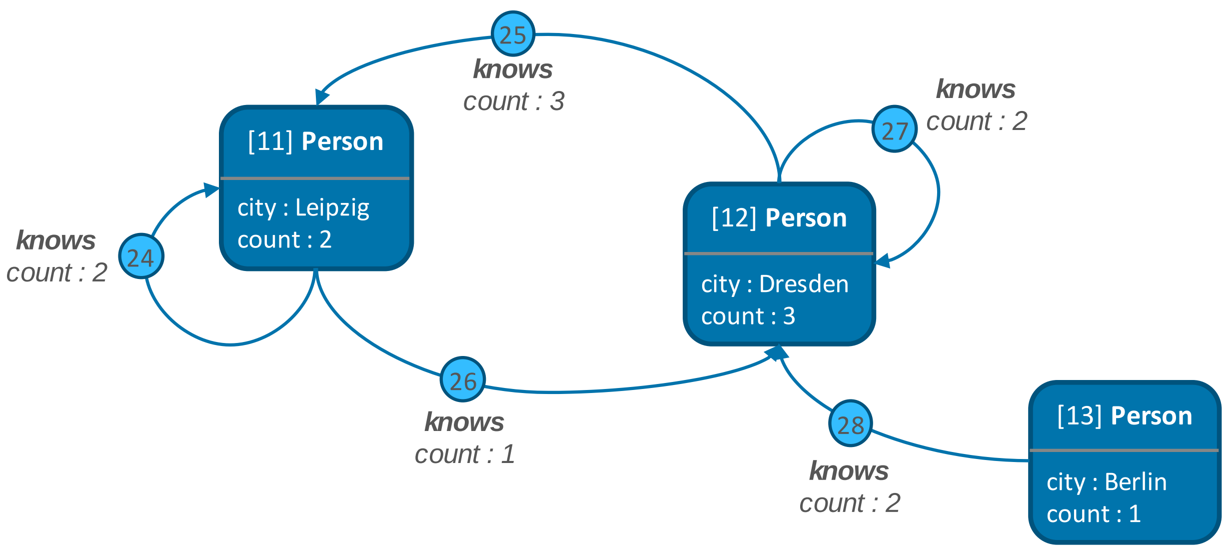
GROUPING: TYPE LEVEL *(SCHEMA GRAPH)*

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

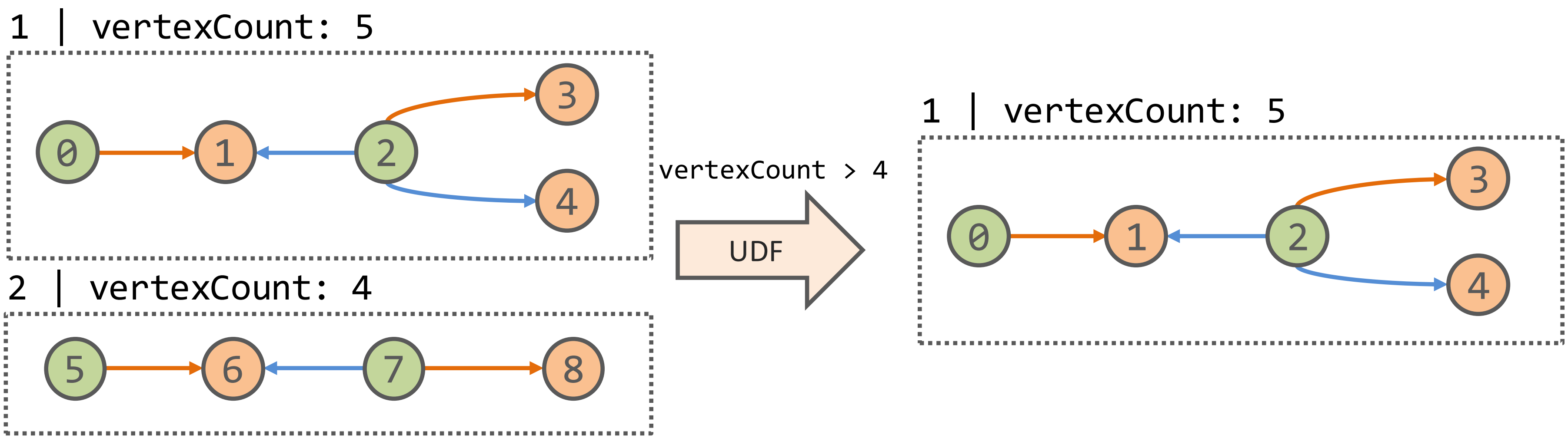


GROUPING: PROPERTY-SPECIFIC

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

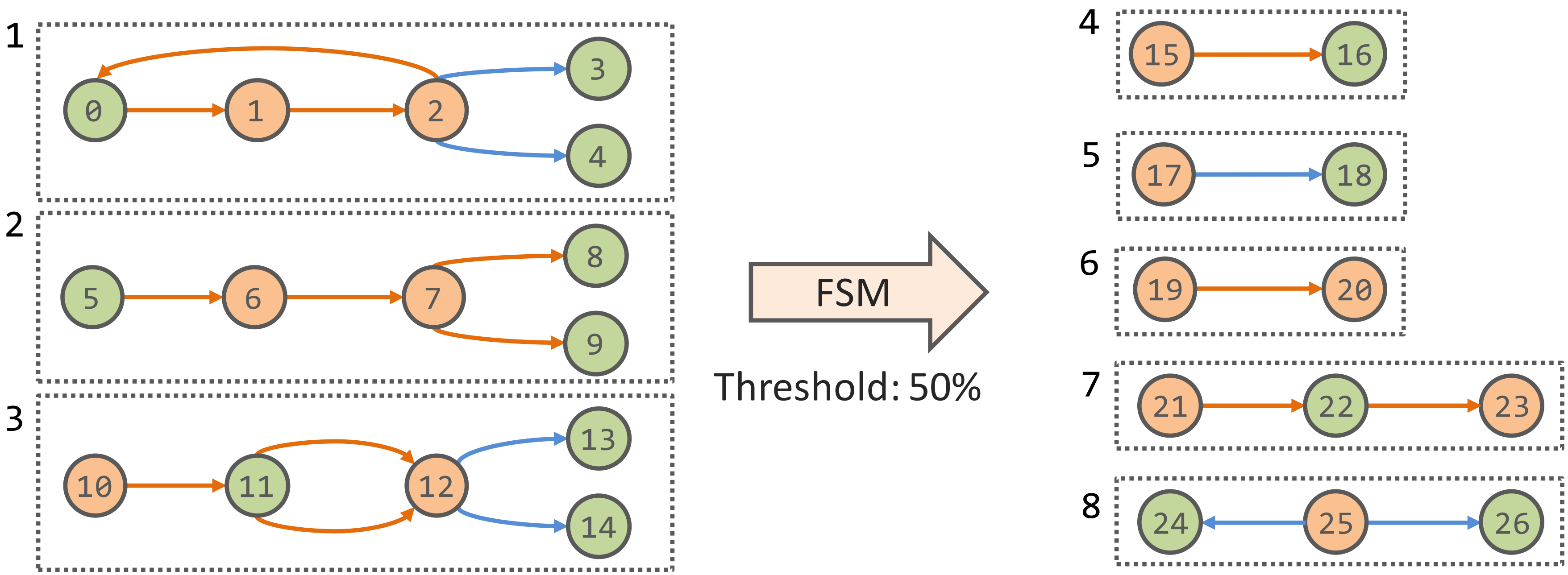


SELECTION



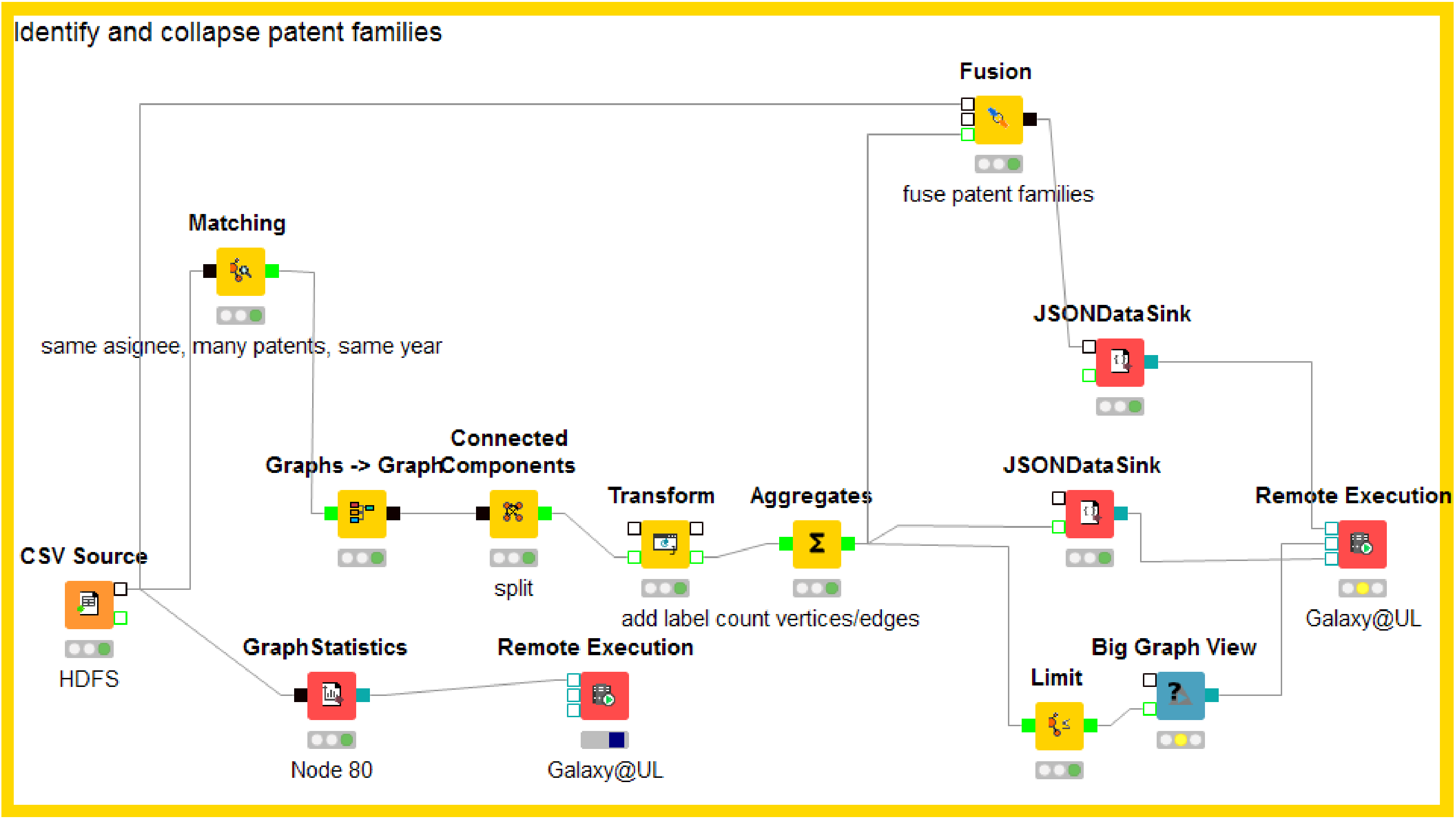

```
GraphCollection filtered = collection.select((graph => graph['vertexCount'] > 4));
```

CALL (E.G., FREQUENT SUBGRAPHS)

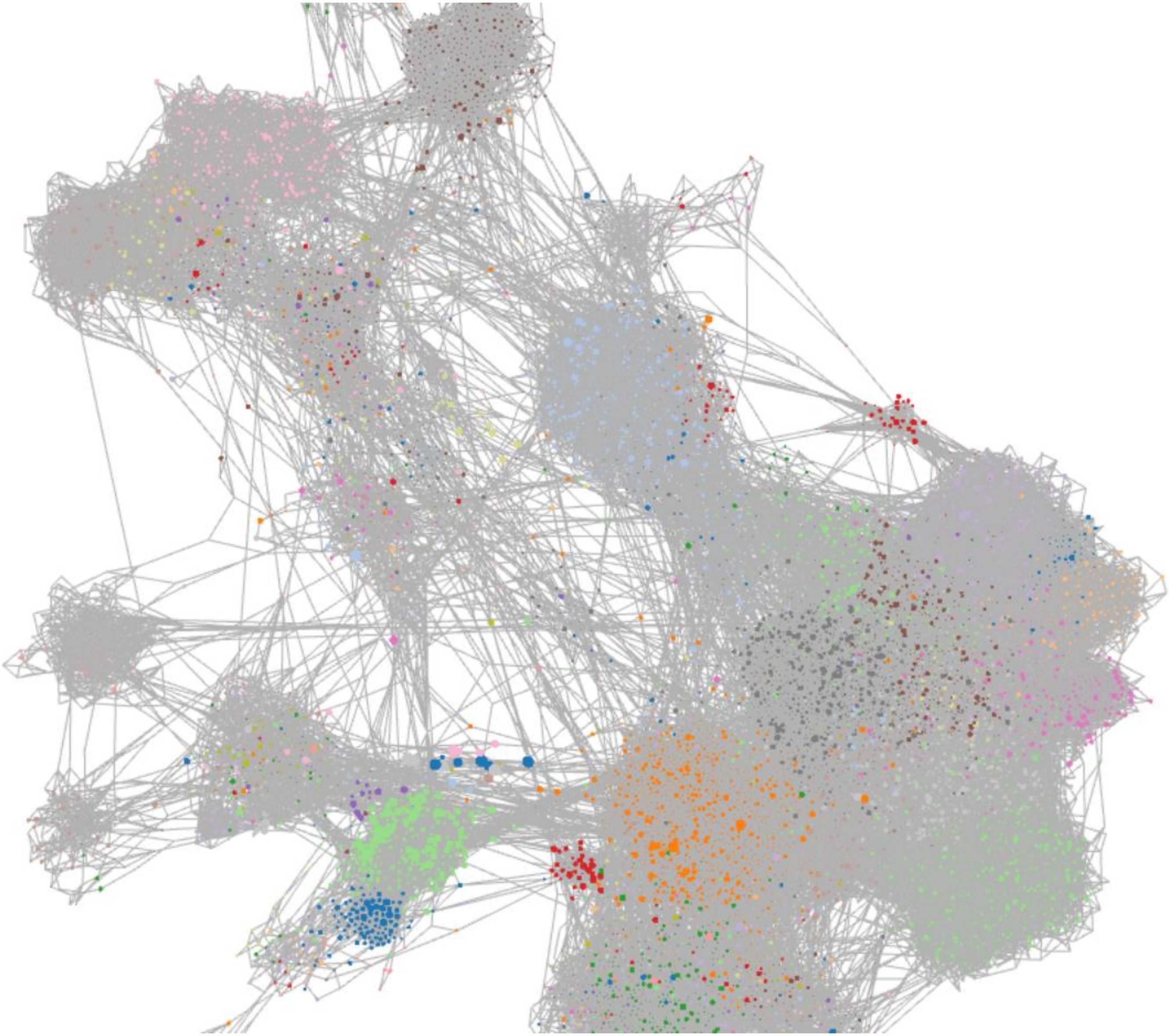



```
GraphCollection frequentPatterns = collection.callForCollection(new TransactionalFSM(0.5))
```

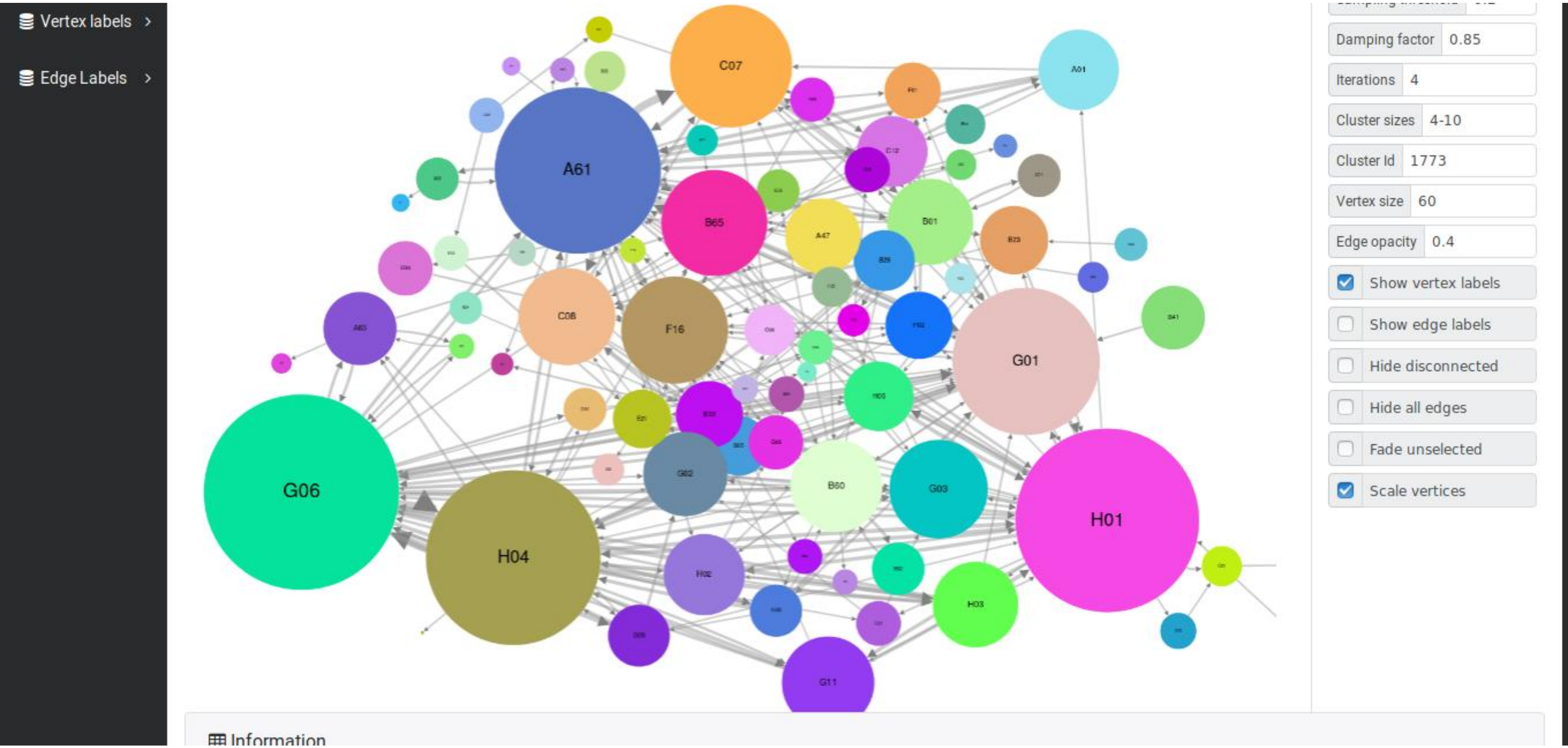
COMBINATION GRADOOP WITH KNIME



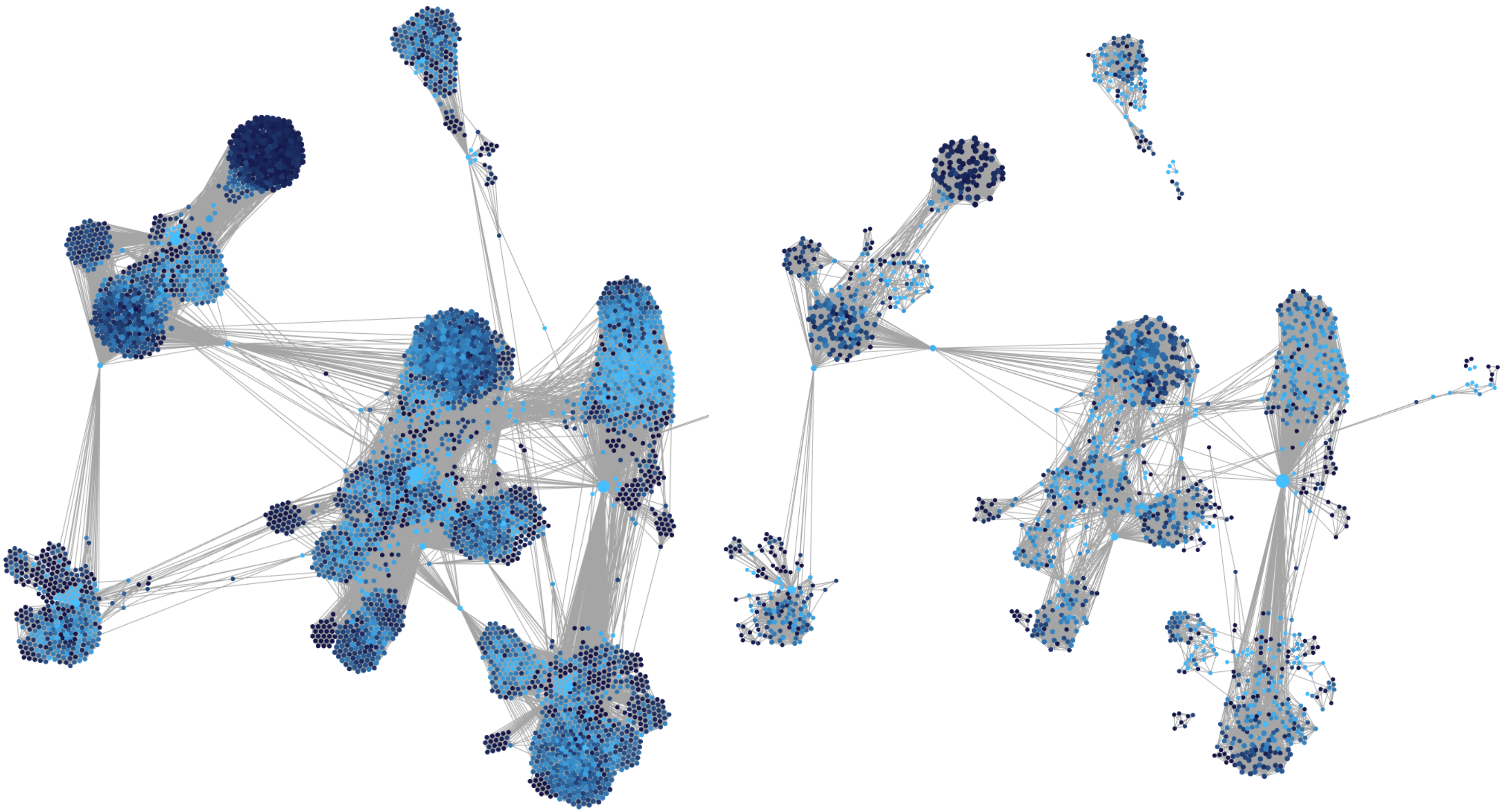
VISUALIZATION (PATENT CITATIONS)



VISUALIZATION (GROUPED PATENT CITATIONS)



SAMPLING (FACEBOOK, RANDOM WALK)



Implementation and evaluation

ScaDS DRESDEN LEIPZIG

GRAPH REPRESENTATION

UNIVERSITÄT LEIPZIG

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>

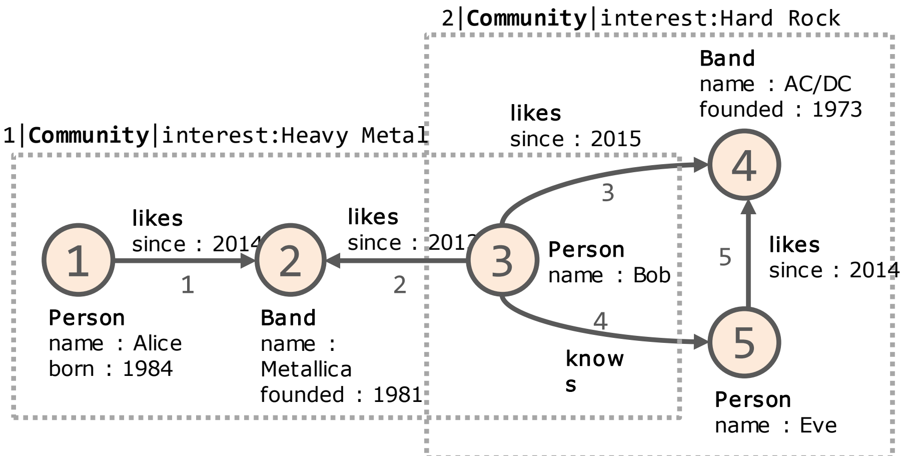
EPGMVertex

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

ScaDS DRESDEN LEIPZIG

GRAPH REPRESENTATION: EXAMPLE

UNIVERSITÄT LEIPZIG



DataSet<EPGMGraphHead>

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

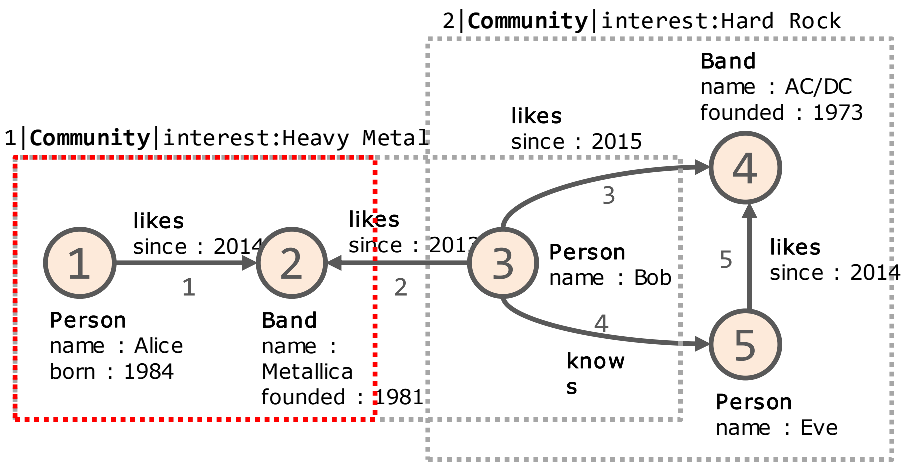
DataSet<EPGMVertex>

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}

OPERATOR IMPLEMENTATION



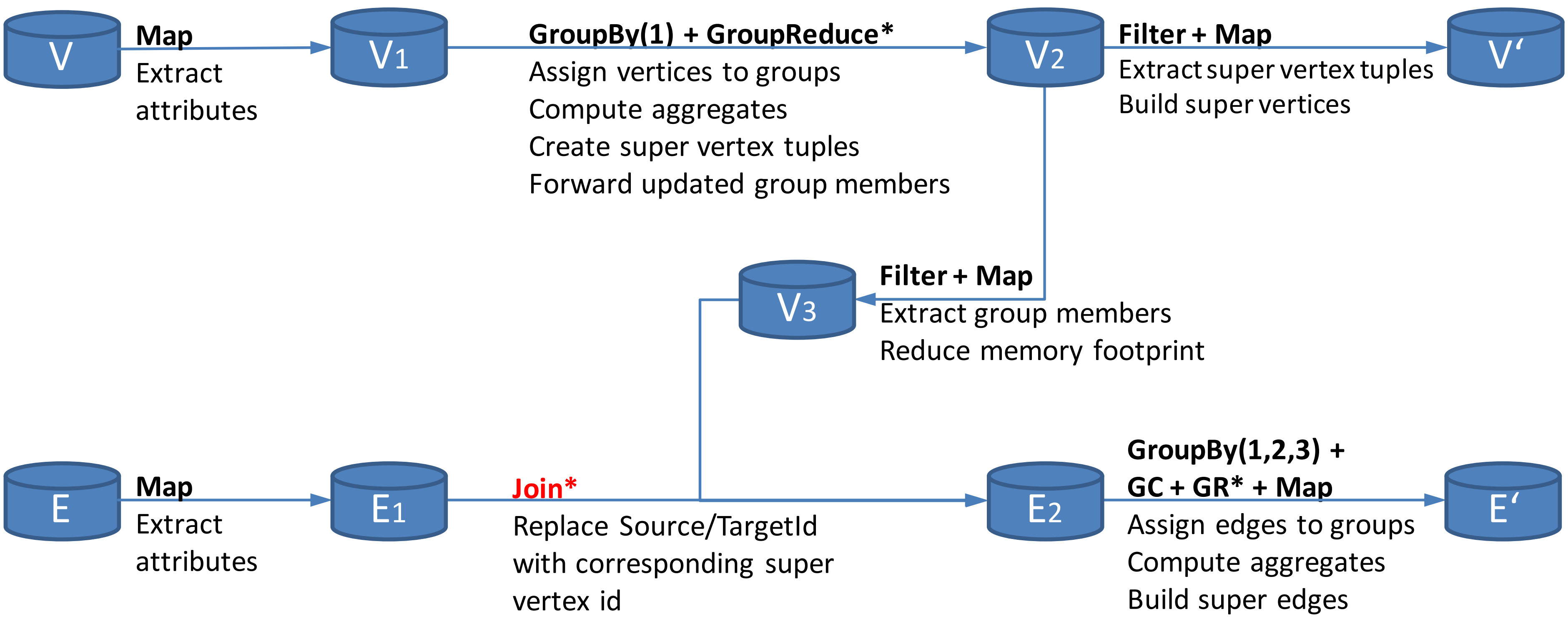
Exclusion

```
// 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>());
```

ScaDS DRESDEN LEIPZIG

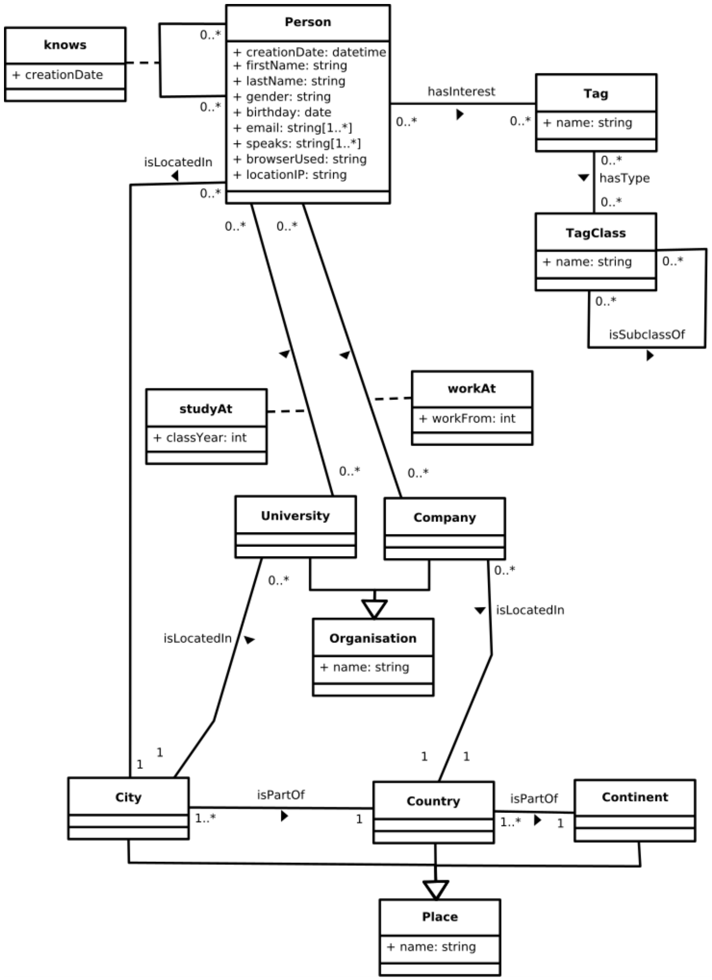
UNIVERSITÄT LEIPZIG

IMPLEMENTATION OF GRAPH GROUPING



*requires worker communication

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

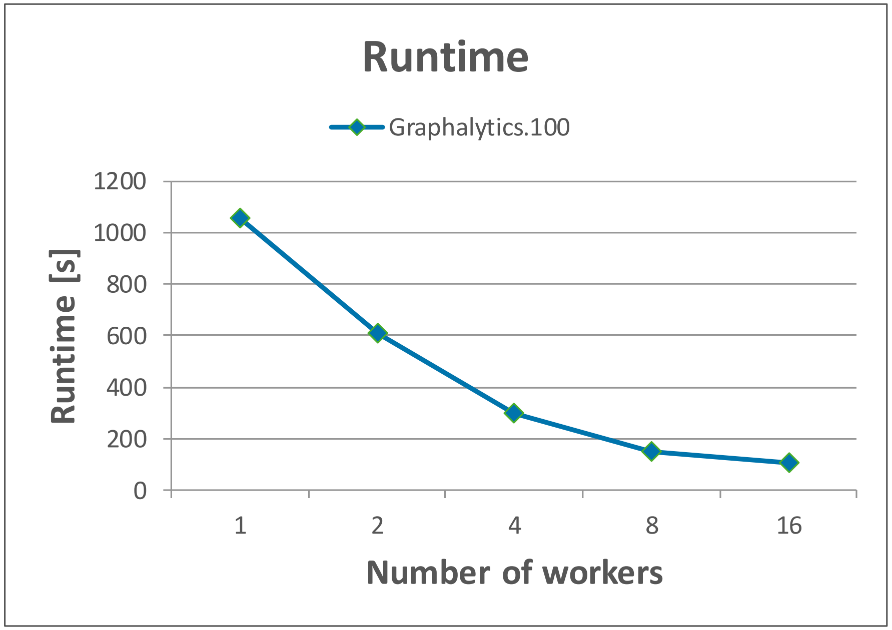
TEST WORKFLOW: SUMMARIZED COMMUNITIES

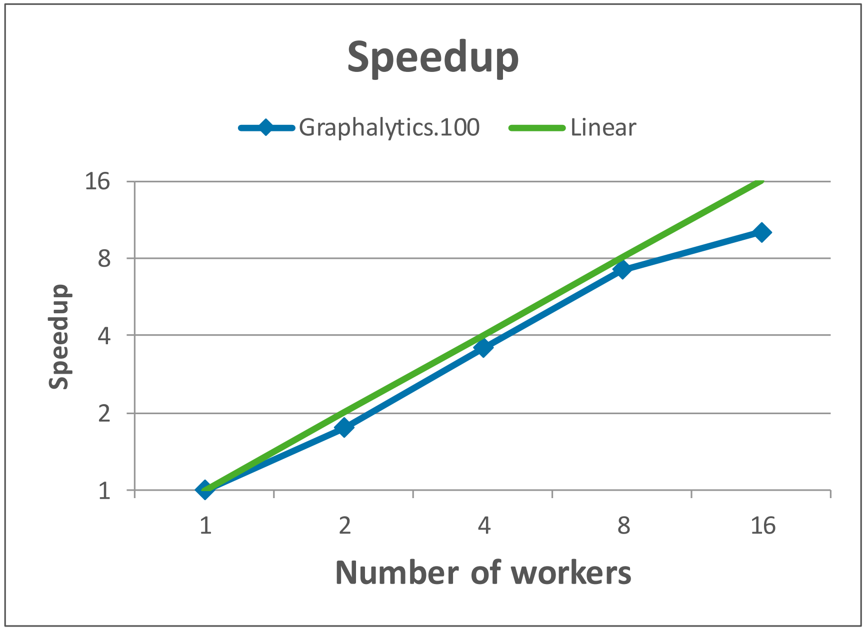
1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities 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

BENCHMARK RESULTS

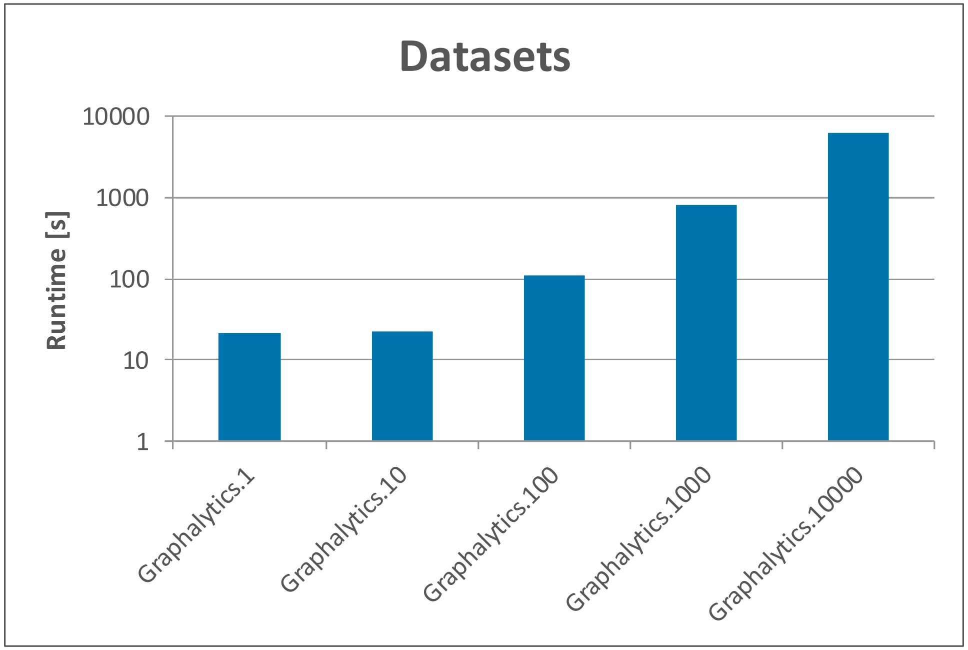




Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 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

COMPARISON

	Graph Database Systems Neo4j, OrientDB	Graph Processing Systems (Pregel, Giraph)	Graph 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	(yes)
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	ongoing work
data integration	no	no	no	ongoing work
visualization	(yes)	no	no	yes

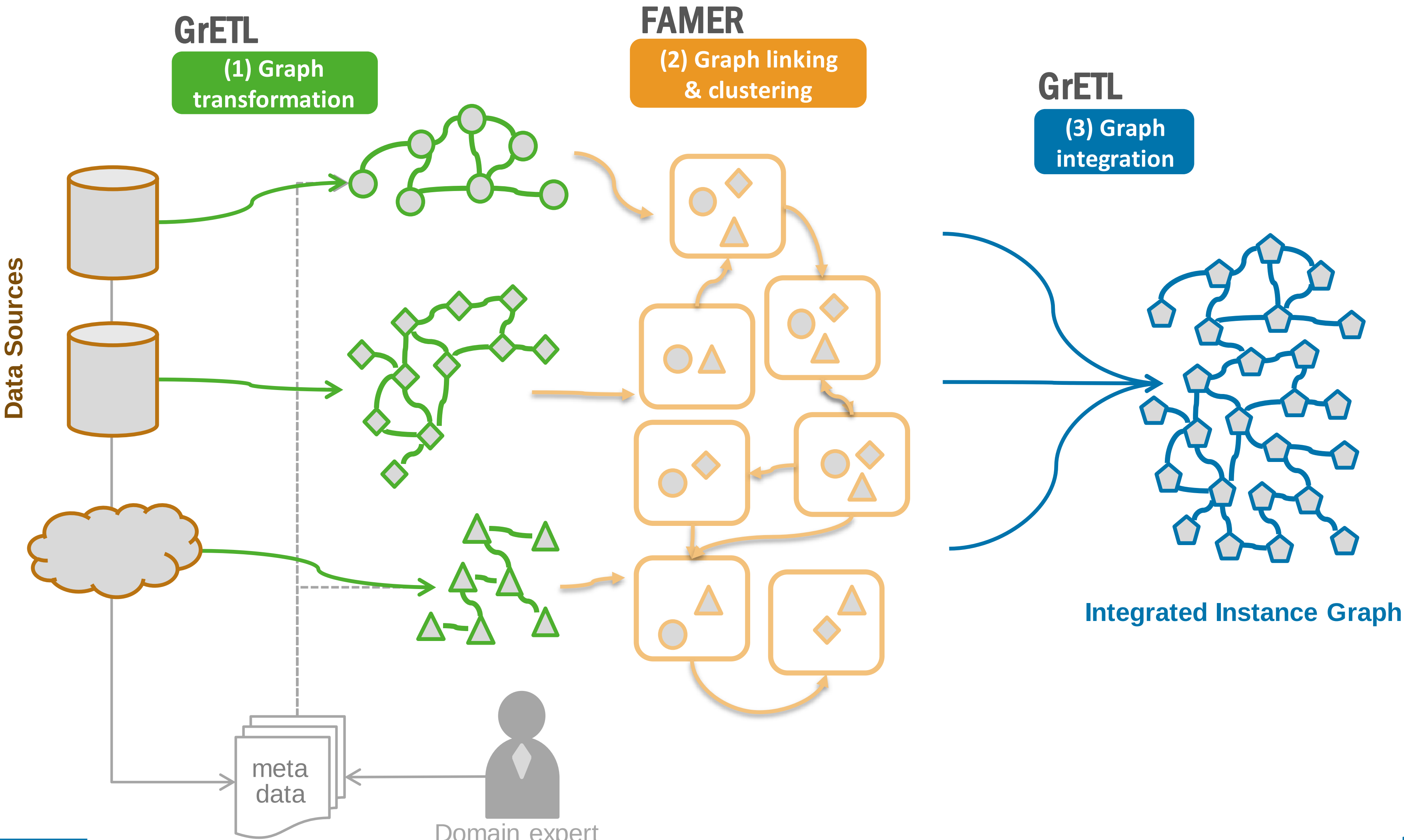
AGENDA

- ScaDS Dresden/Leipzig
- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems
- Gradoop approach
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Implementation and performance evaluation
- Ongoing work
  - Graph-based data integration
    - graph transformations
    - multi-source matching (FAMER)
  - Temporal graphs
- Conclusions

GRAPH-BASED DATA INTEGRATION

- need to integrate diverse data from different sources (or from data lake) into semantically expressive graph representation
  - for later graph analysis
  - for constructing knowledge graphs
- traditional tasks for data acquisition, data transformation & cleaning, schema / entity matching, entity fusion, data enrichment / annotation
- new challenges
  - many data sources (pairwise linking of sources not sufficient)
  - match and fuse both entities and relationships
  - several entity and relationship types
  - more complex preparatory data transformations to resolve structural heterogeneity in input sources/graphs

GRAPH DATA INTEGRATION WORKFLOW



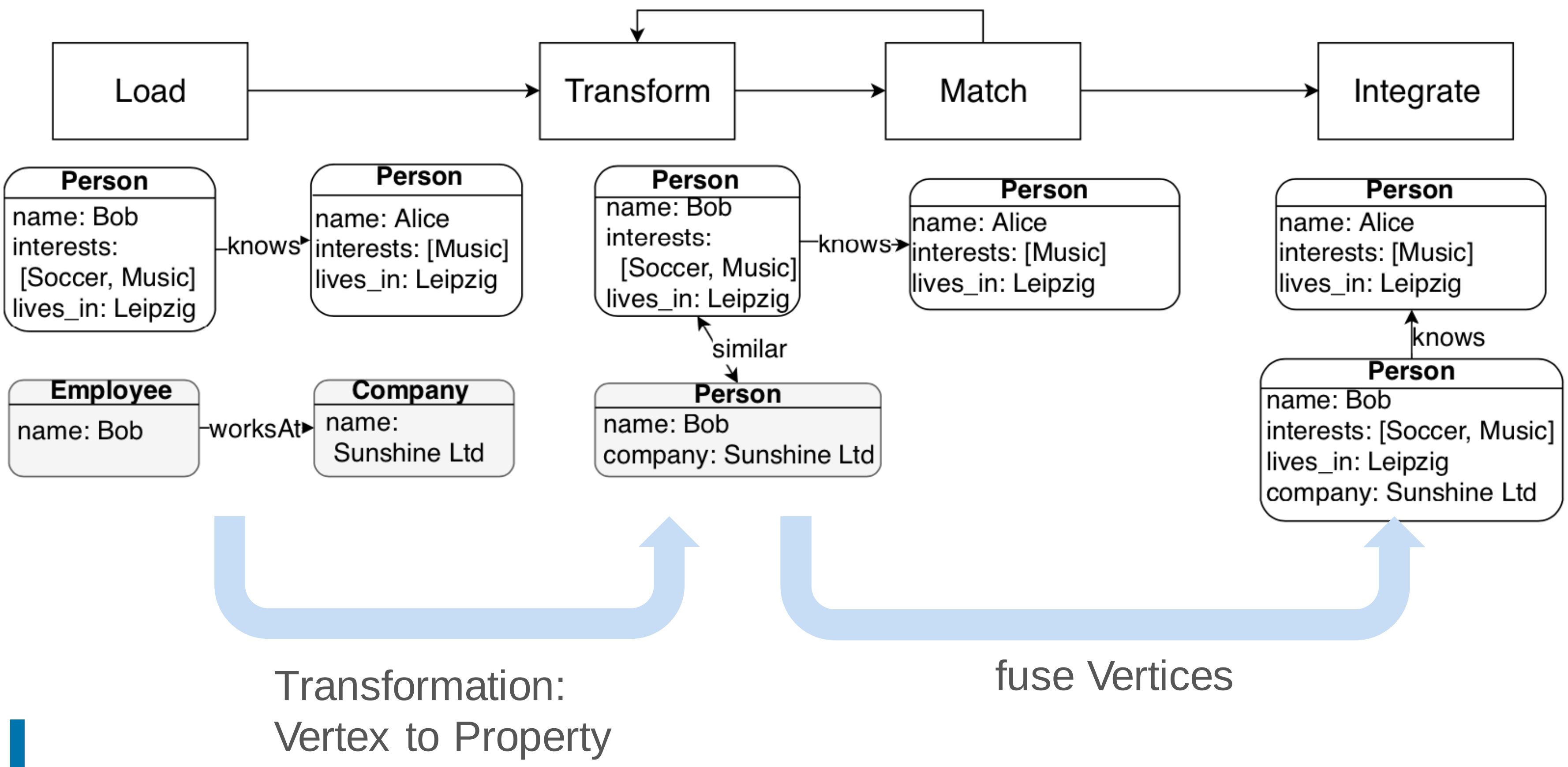
GRAPH TRANSFORMATION OPERATORS*

- high level operators implemented on top of Apache Flink as addition to Gradoop

Operator	GrALa
Property To Vertex	`graph.propertyToVertex(label, propertyName, newLabel, newPropertyName, edgeConfig, condense)`
Vertex to Property	`graph.vertexToProperty(label, edgeConfig)`
Vertex To Edge	`graph.vertexToEdge(vertexLabel, newEdgeLabel)`
Edge To Vertex	`graph.edgeToVertex(edgeLabel, newVertexLabel, edgeLabelSourceToNew, edgeLabelNewToTarget)`
Connect Neighbors	`graph.connectNeighbors(vertexLabel, edgeDirection, neighborVertexLabel, newEdgeLabel)`
Invert Edge	`graph.invertEdge(label, newLabel)`
Cluster Fusion	`graph.fuse(fusionConfig)`
Grouping	`graph.groupBy(vertexGroupingKeys, edgeGroupingKeys)`
Cypher Construct	`graph.query(patternQuery, constructionQuery)`

* M. Kricke, E. Peukert, E. Rahm: *Graph transformations in Gradoop*. Proc. BTW 2019

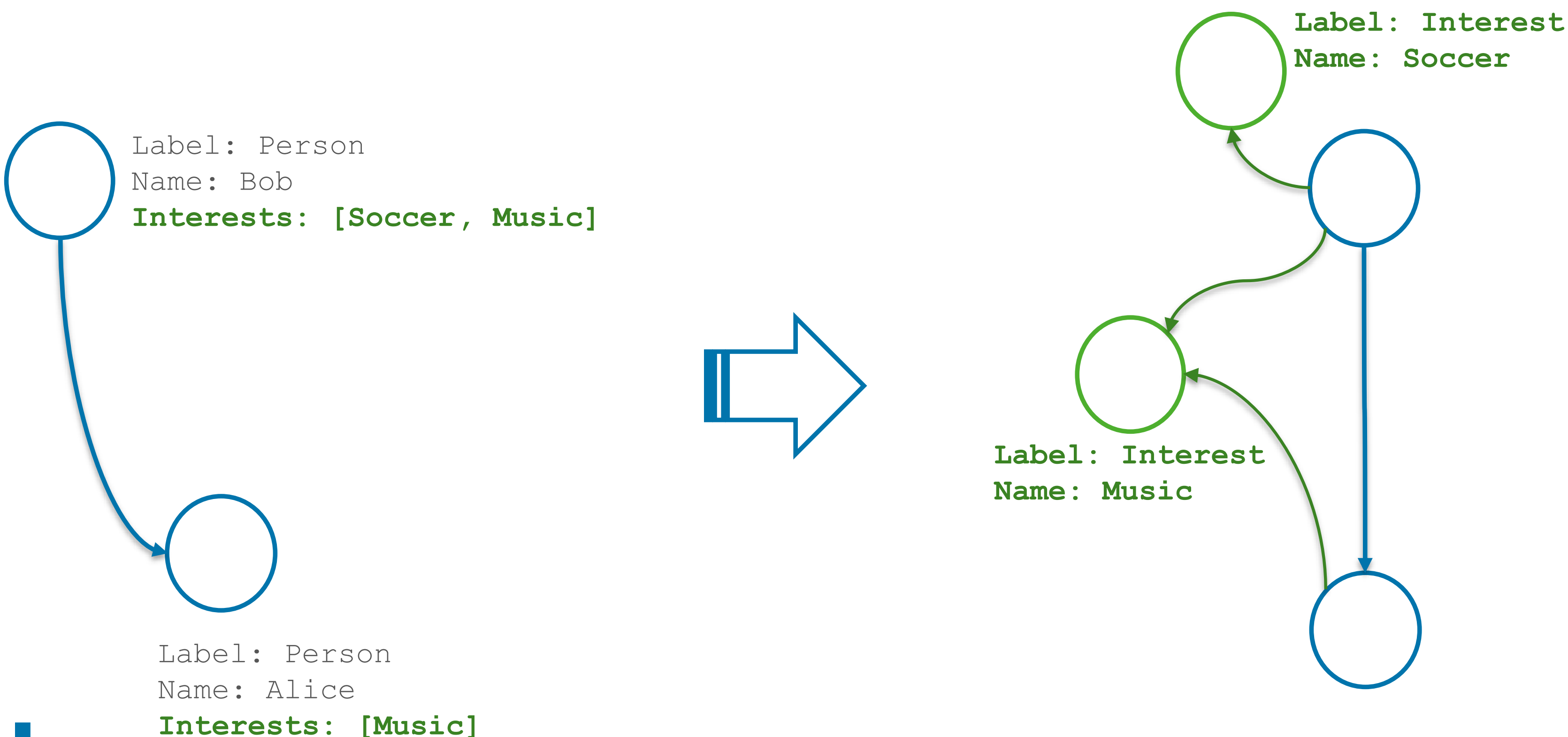
TYPICAL WORKFLOW



PROPERTY TO VERTEX

Pseudocode:

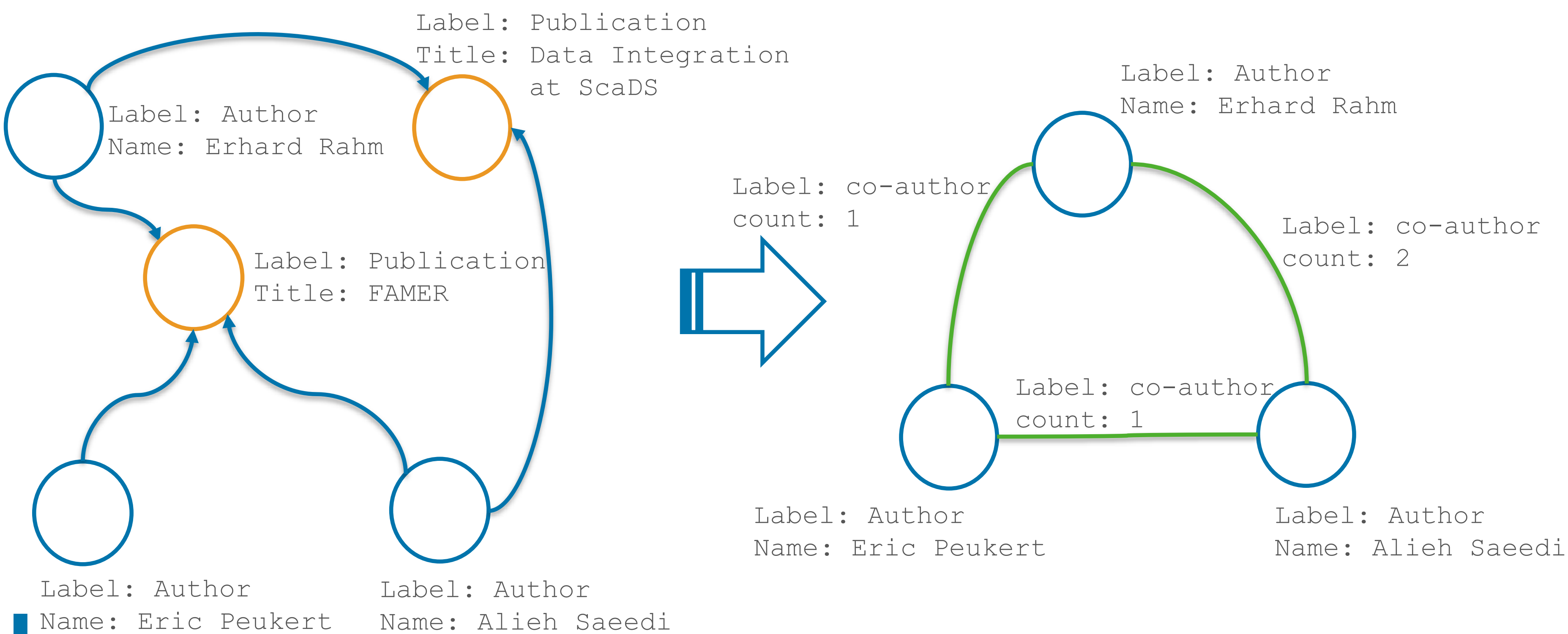
```
inputGraph
  .propertyToVertex(Person, Interests, Interest, Name)
```



CONNECT NEIGHBORS

Pseudocode:

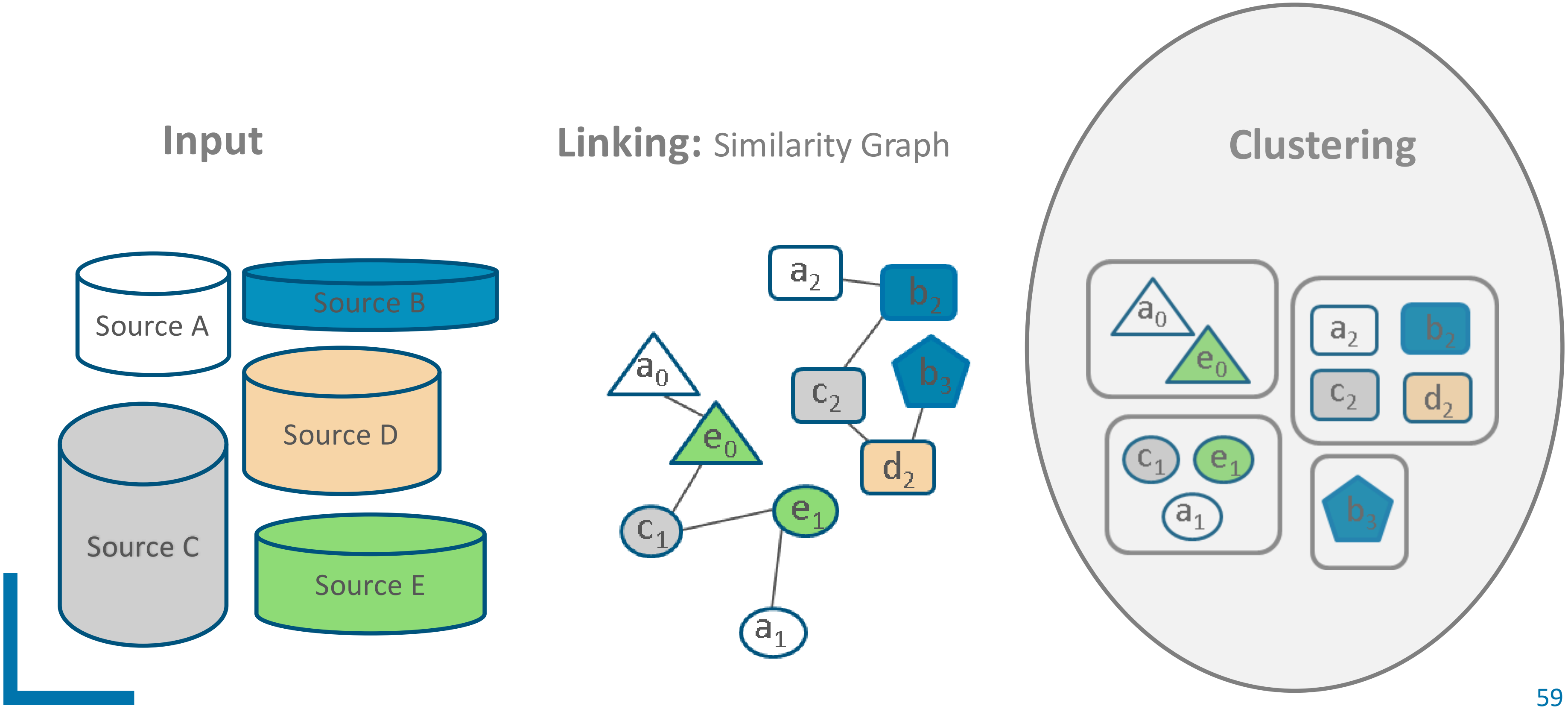
```
inputGraph
  .connectNeighbors(Publication, Author, co-author)
  .fuseEdges(co-author, count, SUM)
  .vertexInducedSubgraph(ByLabel(Author))
```



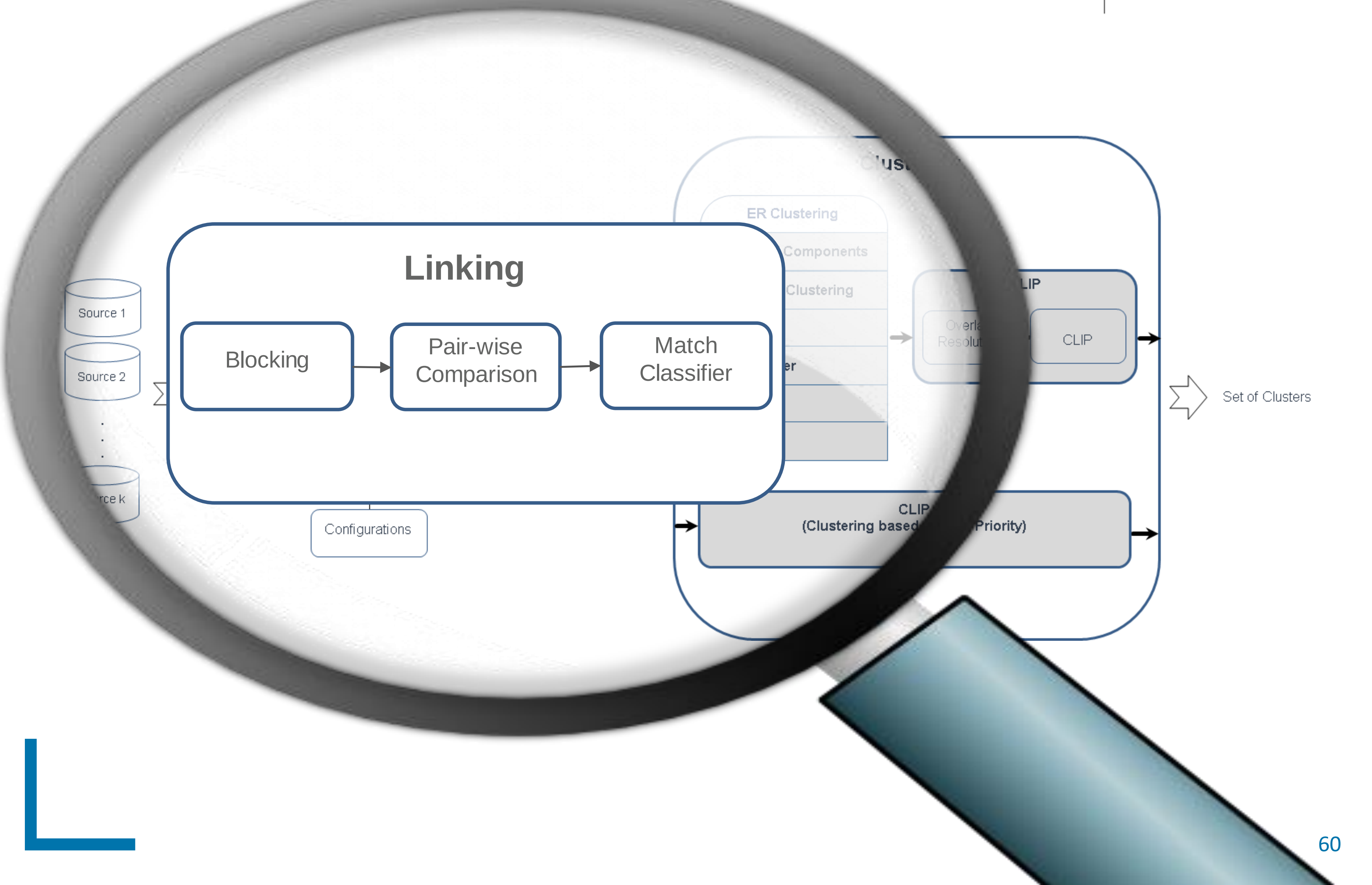
ENTITY LINKING AND CLUSTERING

FAst **M**ulti-source **E**ntity **R**esolution System

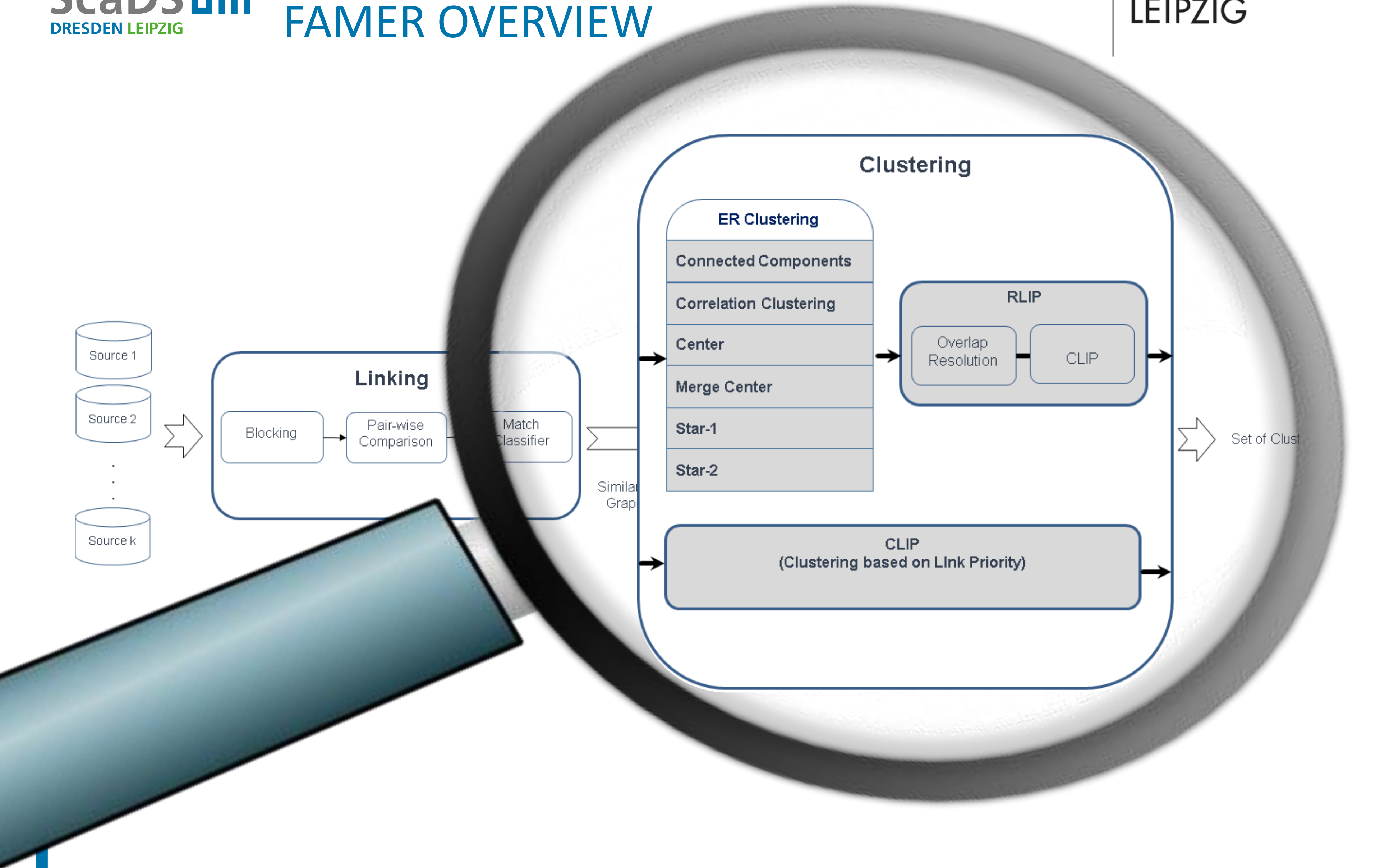
- scalable linking & clustering for many sources



FAMER OVERVIEW



FAMER OVERVIEW

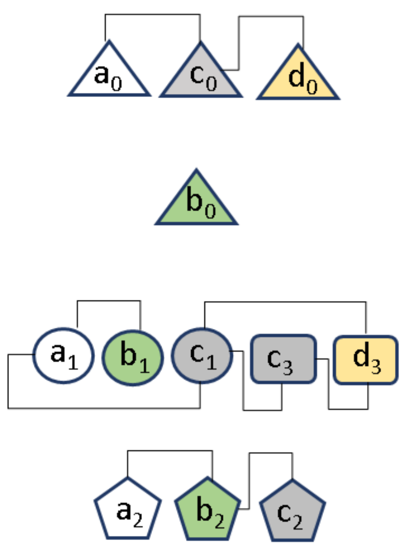


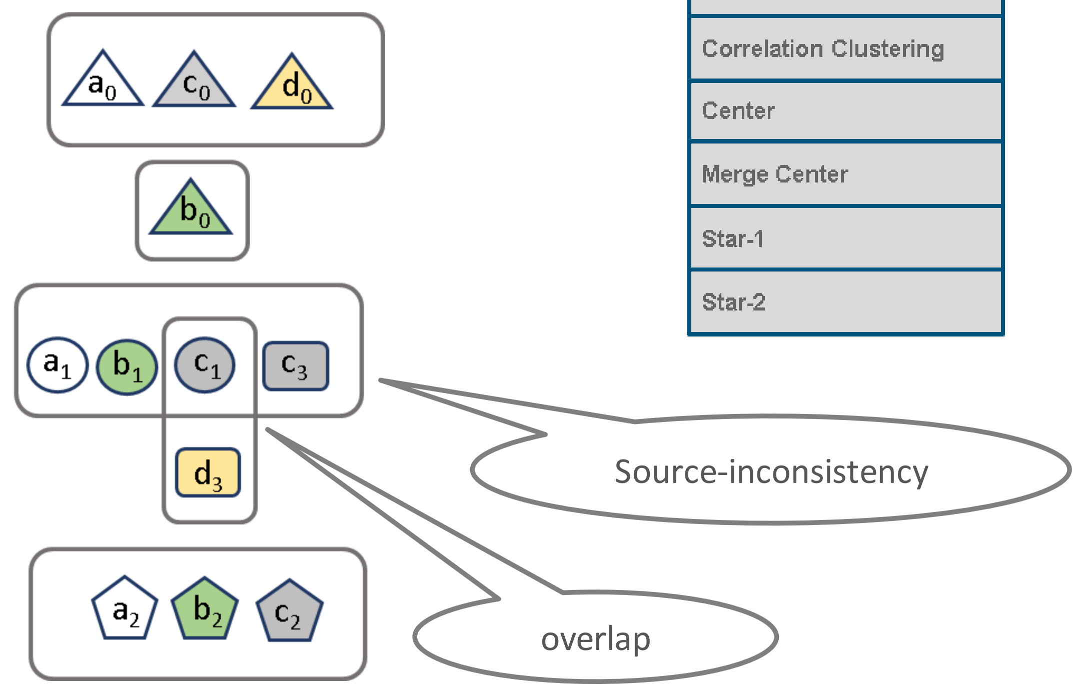
ScaDS DRESDEN LEIPZIG

FAMER CLUSTERING

UNIVERSITÄT LEIPZIG

Similarity graph



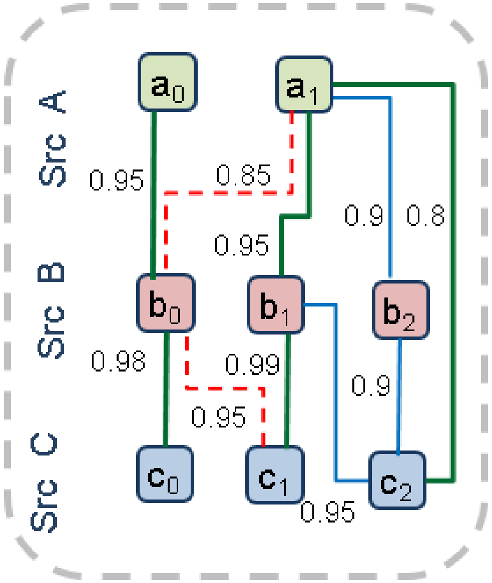


ER Clustering

- Connected Components
- Correlation Clustering
- Center
- Merge Center
- Star-1
- Star-2

LINK STRENGTH

- Link Strength
  - Strong
  - Normal
  - Weak

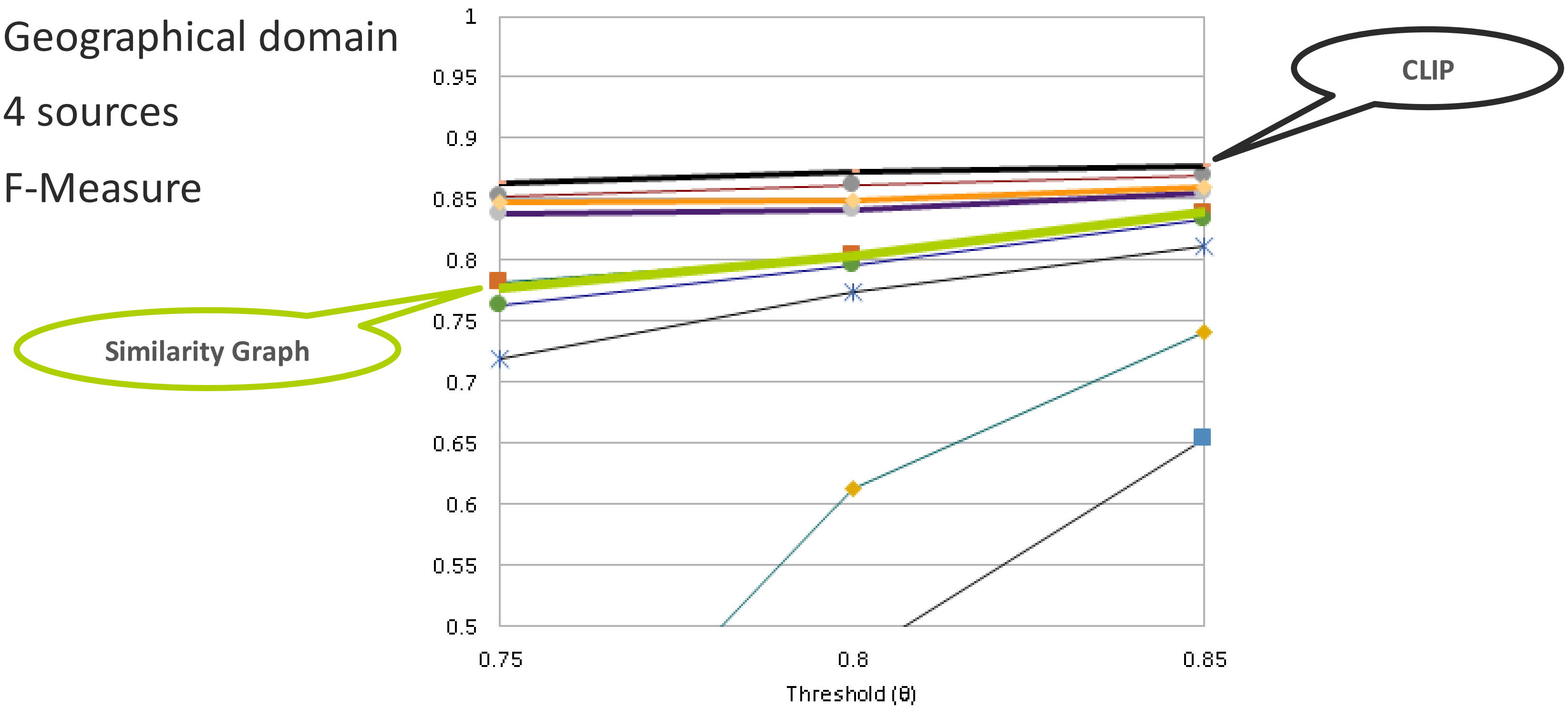


ScaDS DRESDEN LEIPZIG

CLIP QUALITY

UNIVERSITÄT LEIPZIG

- Geographical domain
- 4 sources
- F-Measure



ScaDS DRESDEN LEIPZIG

UNIVERSITÄT LEIPZIG

DYNAMIC GRAPHS

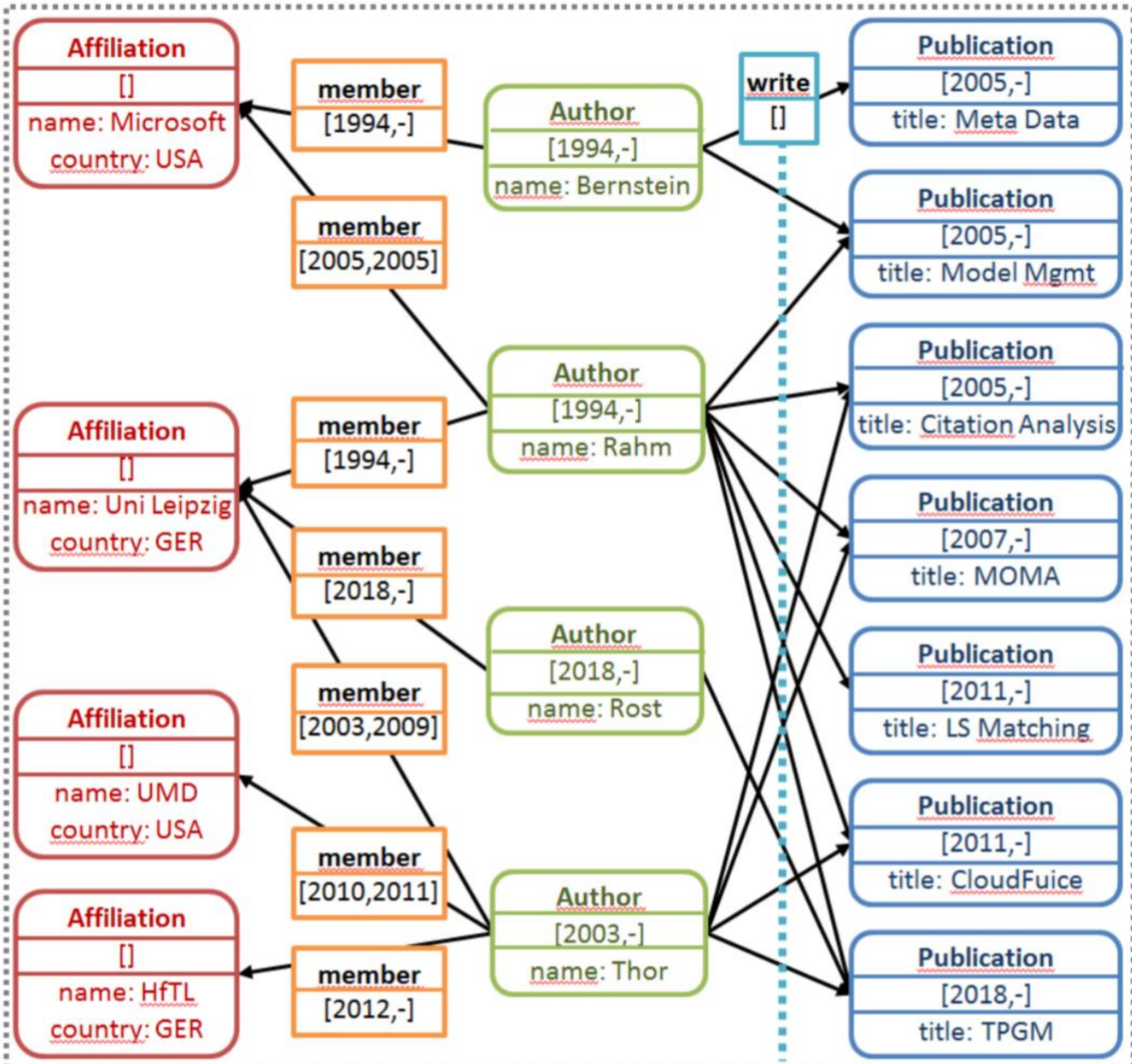
- current graph databases and graph processing systems focus on static graphs
- real graphs like social networks, citation networks, road networks etc change over time
  - graph elements are continuously added, removed or updated
  - slowly evolving networks vs. streaming graph data (e.g., sensor data)
- analytical questions are often time-related
  - as-of queries on past states (snapshots)
  - change/evolution analysis ...
  - need to efficiently update/refresh analysis results (graph metrics, communities/clusters, ...)
- need of scalable approaches for managing and analyzing temporal and stream graphs

ScaDS DRESDEN LEIPZIG

TEMPORAL GRAPHS IN GRADOOP

UNIVERSITÄT LEIPZIG

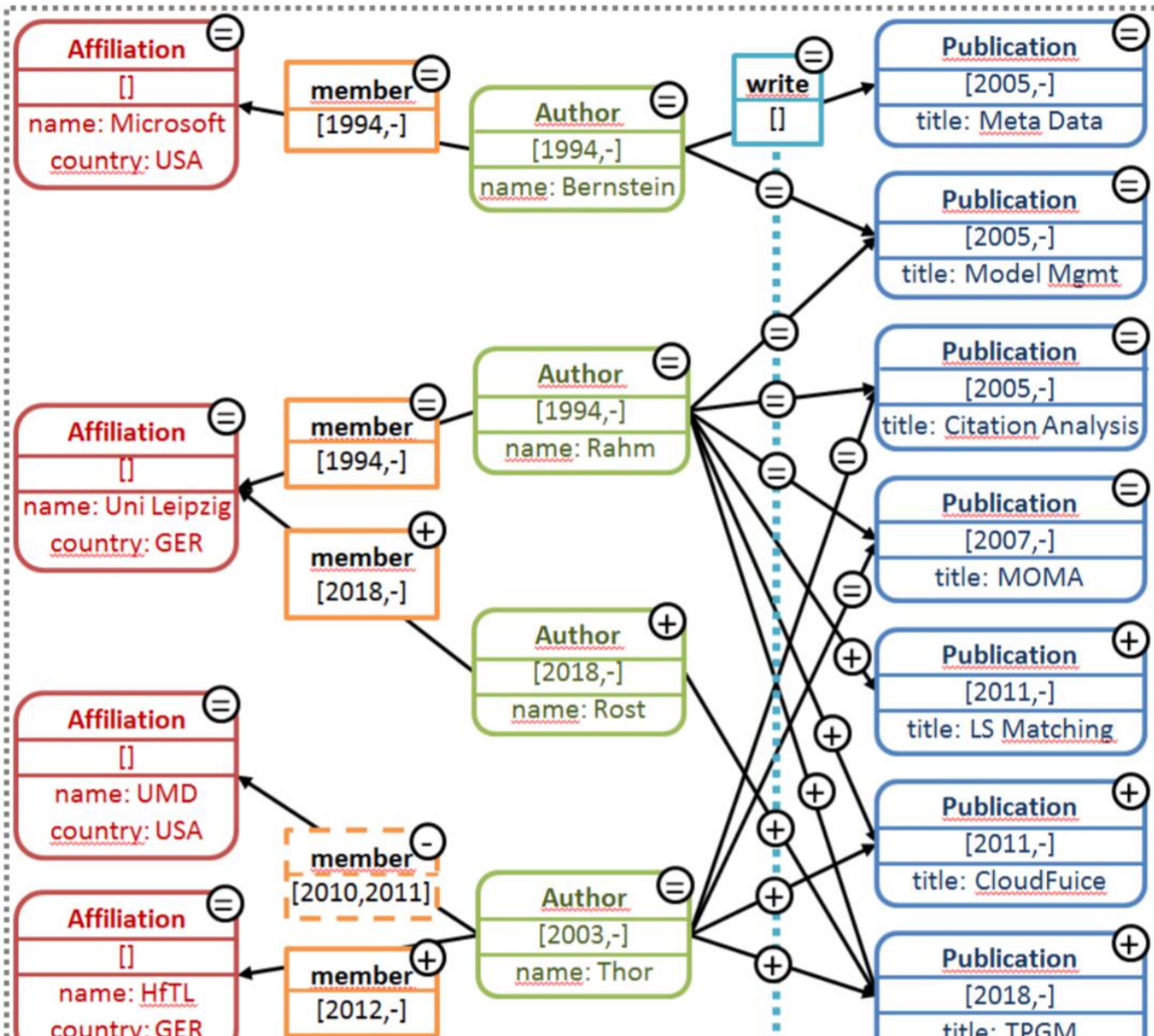
- support for bitemporal graphs
  - time intervals for valid time (val-from, val-to) and transaction time (tx-from, tx-to) for vertices, edges and graphs
  - *valid time* provided by user, *tx time* is system-provided
  - similar to temporal support in SQL:2011
- changes to existing operators
  - time predicates (as-of, between, overlap, precedes/suceeds …) for subgraph, pattern matching, grouping …
- new operators
  - snapshot extraction (as-of subgraph)
  - graph diff (between two snaphshots)


Affiliation
[]
name: Microsoft
country: USA
member
[1994,-]
Author
[1994,-]
name: Bernstein
write
[]
Publication
[2005,-]
title: Meta Data
member
[2005,2005]
Publication
[2005,-]
title: Model Mgmt
Author
[1994,-]
name: Rahm
Publication
[2005,-]
title: Citation Analysis
Affiliation
[]
name: Uni Leipzig
country: GER
member
[1994,-]
member
[2018,-]
Publication
[2007,-]
title: MOMA
Author
[2018,-]
name: Rost
Publication
[2011,-]
title: LS Matching
member
[2003,2009]
Affiliation
[]
name: UMD
country: USA
Publication
[2011,-]
title: CloudFuice
member
[2010,2011]
Author
[2003,-]
name: Thor
Affiliation
[]
name: HfTL
country: GER
member
[2012,-]
Publication
[2018,-]
title: TPGM

EXAMPLE QUERIES

- affiliation memberships with a duration of less than 3 years
  - graph.subgraph (null, e -> e.label = 'member' AND YEAR(e.to)-YEAR(e.from) < 3
- authors who had an US affiliation in 2017 (temporal pattern matching)
  - (a:Author)-[m:member]->(f:Affiliation country : USA) WHERE m.asOf(2017)
- graph snapshot as of 2010
  - graph.snapshot (asOf(2010))
- graph difference
  - graph.diff (asOf(2010),asOf(2019))

GRAPH DIFF BETWEEN 2010 AND NOW



AGENDA

- ScaDS Dresden/Leipzig
- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems
- Gradoop approach
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Implementation and performance evaluation
- Ongoing work
  - Graph-based data integration
    - graph transformations
    - multi-source matching (FAMER)
  - Temporal graphs
- Conclusions

SUMMARY

- Big Graph Analytics
  - many alternatives, but limitations
  - graph collections not generally supported
  - Insufficient support for graph-based data integration and 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 (Hbase, Apache Flink)
  - Integration into Knime

OUTLOOK / CHALLENGES

- Graph-based data integration
  - Integration matching for multiple vertex and edge types
  - Incemental addition of new data (sources)
  - maintenance of knowledge graphs
- Dynamic data
  - Implementation of new operators
  - Support for stream data
  - Graph mining on dynamic graphs
- Machine learning on graphs
  - use of graph embeddings, e.g., for approximate pattern matching
  - better predictions by using contextual data ...

REFERENCES

- M. Junghanns, M. Kießling, A. Averbuch, A. Petermann, E. Rahm: *Cypher-based Graph Pattern Matching in Gradoop*. Proc. ACM SIGMOD workshop on Graph Data Management Experiences and Systems (GRADES), 2017
- M. Junghanns, M. Kießling, N. Teichmann, K. Gomez, A. Petermann, E. Rahm: *Declarative and distributed graph analytics with GRADOOP*. PVLDB 2018
- M. Junghanns, A. Petermann, K. Gomez, E. Rahm: *GRADOOP - Scalable Graph Data Management and Analytics with Hadoop*. Arxiv, 2015
- M. Junghanns, A. Petermann, M. 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
- M. Junghanns, A. Petermann, E. Rahm: *Distributed Grouping of Property Graphs with GRADOOP.* Proc. BTW, 2017
- 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. Kricke, E. Peukert, E. Rahm: *Graph transformations in Gradoop*. Proc. BTW 2019
- M. Nentwig, E. Rahm: *Incremental Clustering on Linked Data*. Proc. ICDMW 2018
- 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. ICDEW 2014
- A. Petermann, M. Junghanns, R. Müller, E. Rahm: *Graph-based Data Integration and Business Intelligence with BIIIG.* PVLDB 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, E. Rahm: *DIMSpan - Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems.* Proc. BDCAT 2017
- E. Rahm: *The Case for Holistic Data Integration*. Proc. ADBIS, 2016
- E. Rahm et al.: *Big Data Competence Center ScaDS Dresden/Leipzig: Overview and selected research activities*. Datenbank-Spektrum 2019
- C. Rost, A. Thor, E. Rahm: *Temporal graph analysis using Gradoop*. Proc. BTW workshops, 2019
- M.A. Rostami, M. Kricke, E. Peukert, S. Kühne, M. Wilke, S. Dienst, E. Rahm: *BIGGR: Bringing Gradoop to Applications.* Datenbank-Spektrum 2019
- A. Saeedi, M. Nentwig, E. Peukert, E. Rahm: *Scalable Matching and Clustering of Entities with FAMER*. CSIMQ 2018
- A. Saeedi, E. Peukert, E. Rahm: *Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution*. Proc. ADBIS, LNCS 10509, 2017
- A. Saeedi, E. Peukert, E. Rahm: *Using Link Features for Entity Clustering in Knowledge Graphs.* Proc. ESWC 2018