UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

SCALABLE GRAPH ANALYTICS

ERHARD RAHM

www.scads.de

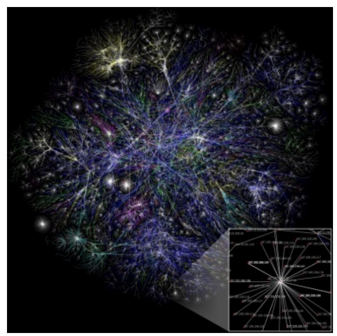
Management and Analysis of Big Graph Data: Current Systems and Open Challenges

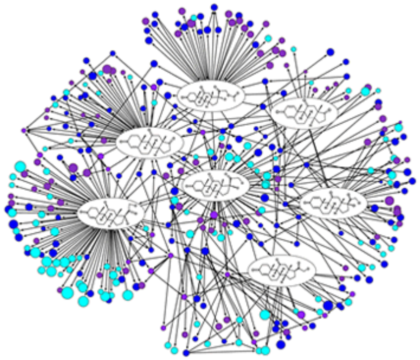
Martin Junghanns, André Petermann, Martin Neumann and Erhard Rahm

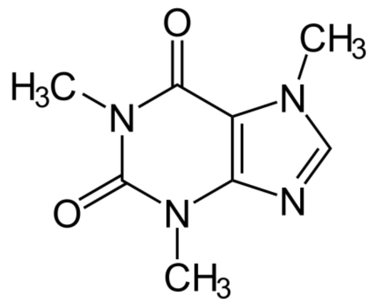
Abstract Many big data applications in business and science require the management and analysis of huge amounts of graph data. Suitable systems to manage and to analyze such graph data should meet a number of challenging requirements including support for an expressive graph data model with heterogeneous vertices and

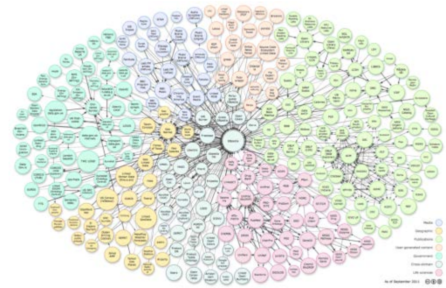
„GRAPHS ARE EVERYWHERE“

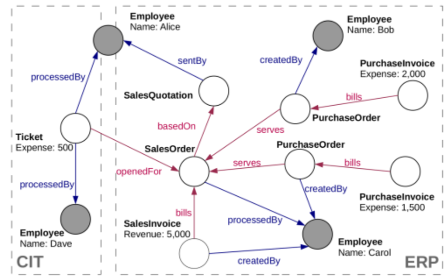
Social science	Engineering	Life science	Information science











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

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

Internet
- ca. 2.9 billion users

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

Patients
- > 18 millions (Germany)

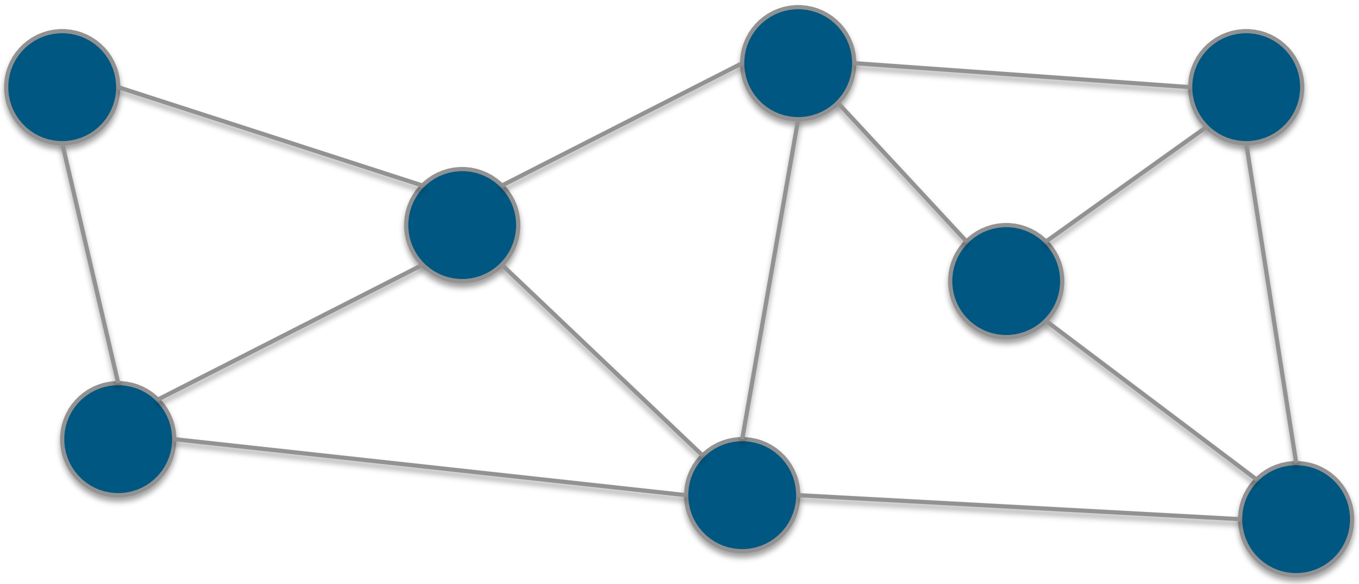
Illnesses
- > 30.000

World Wide Web
- ca. 1 billion Websites

LOD-Cloud
- ca. 90 billion triples

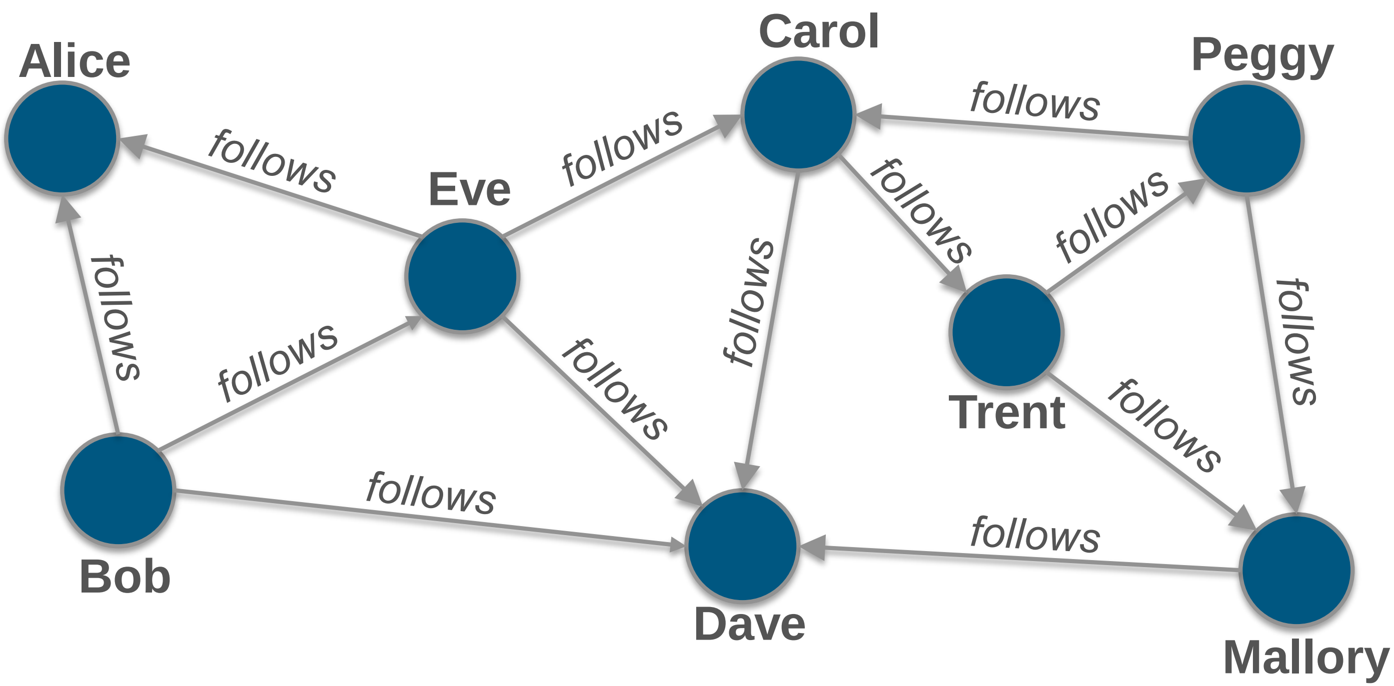
ScaDS DRESDEN LEIPZIG

"GRAPHS ARE EVERYWHERE"



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

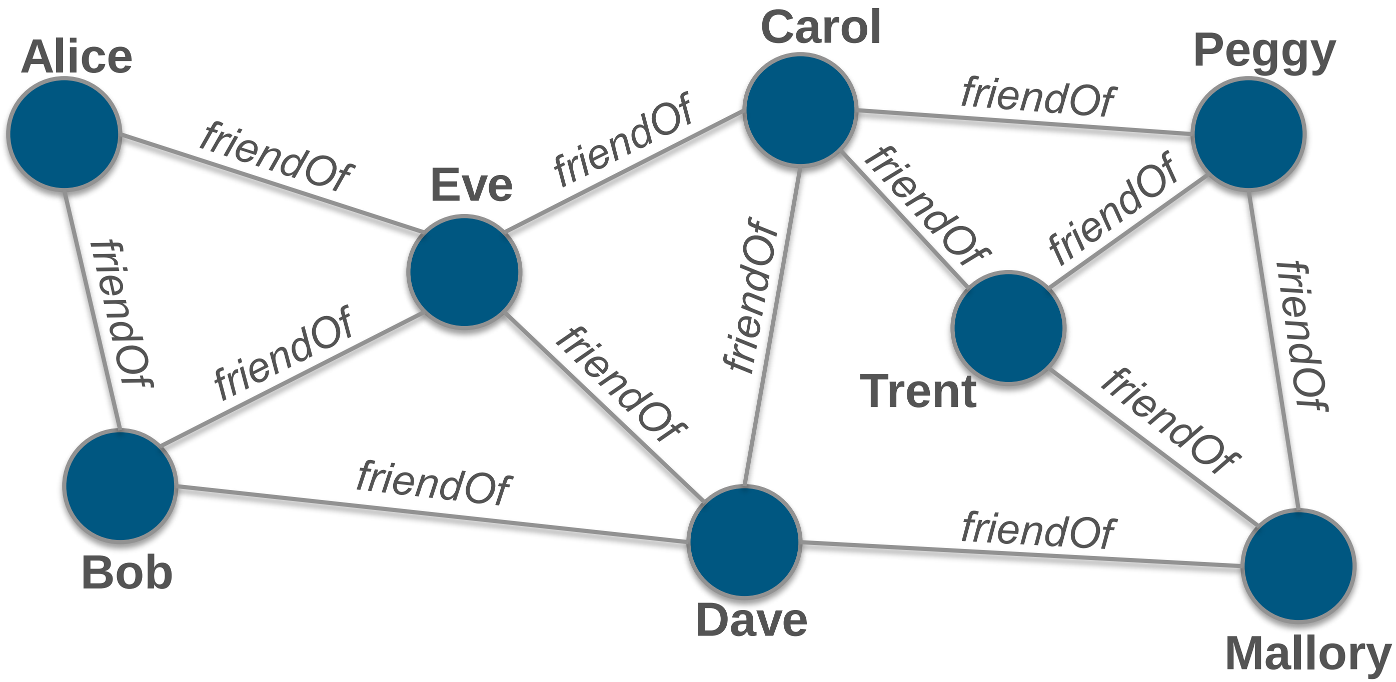
“GRAPHS ARE EVERYWHERE”



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

ScaDS DRESDEN LEIPZIG

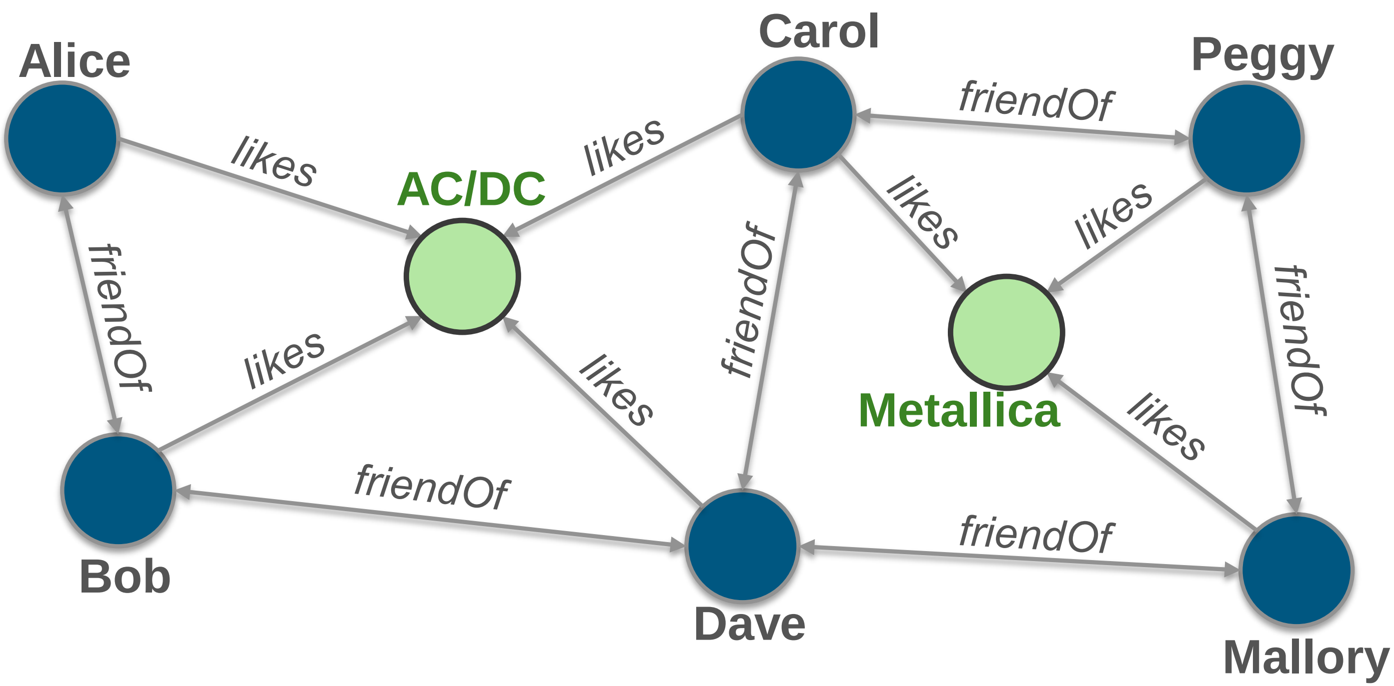
"GRAPHS ARE EVERYWHERE"



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

ScaDS DRESDEN LEIPZIG

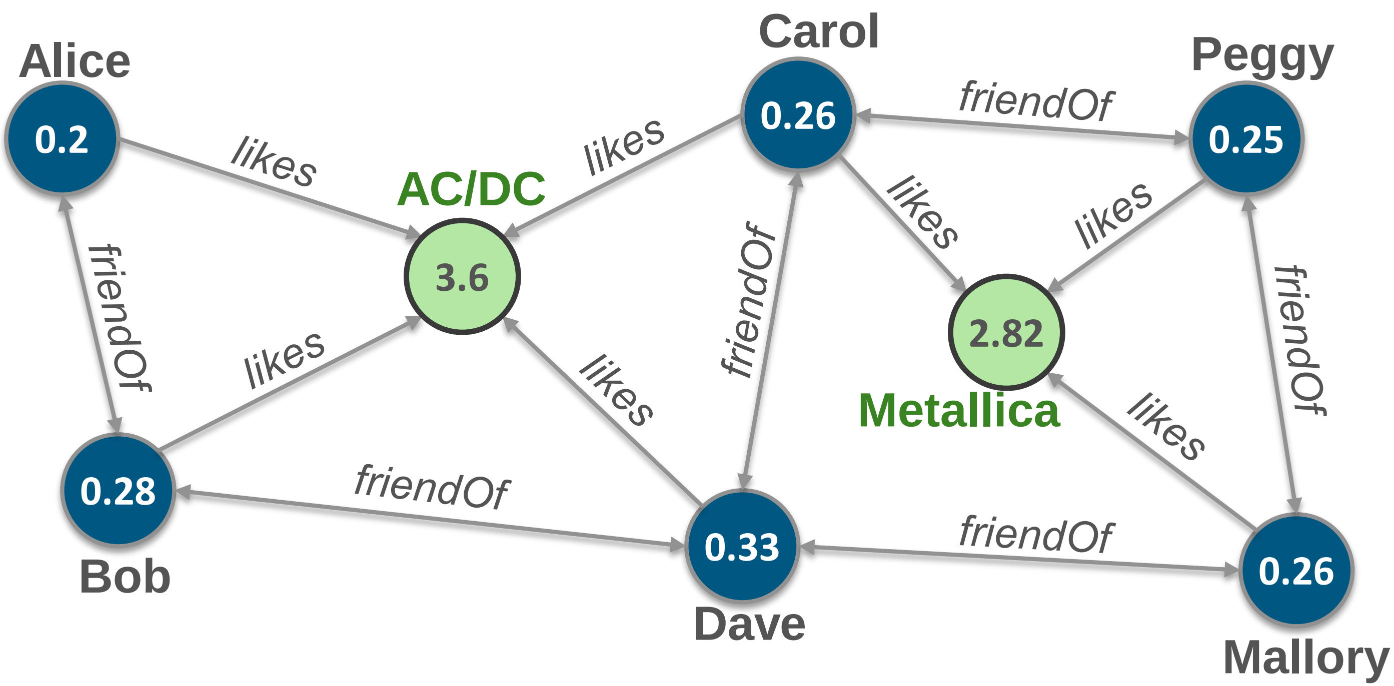
"GRAPHS ARE HETEROGENEOUS"



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

ScaDS DRESDEN LEIPZIG

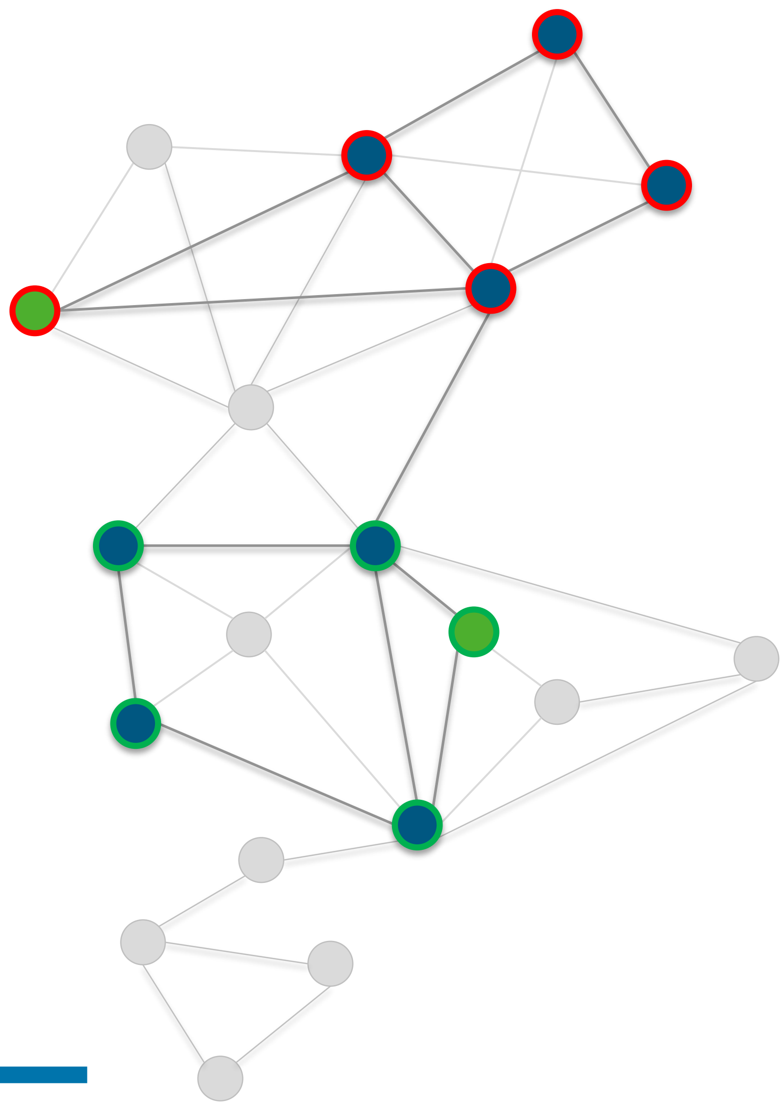
"GRAPHS CAN BE ANALYZED"



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

ScaDS DRESDEN LEIPZIG

"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: HIGH-LEVEL REQUIREMENTS

- *all „V“ challenges*
  - *volume (scalability)*
  - *Variety (support for heterogenous data / data integration)*
  - *Velocity (dynamically changing graph data)*
  - *veracity (high data quality)*
  - *value (improved business value)*
- *ease-of-use*
- *high cost-effectiveness*

ScaDS DRESDEN LEIPZIG

GRAPH DATA ANALYTICS: REQUIREMENTS

- 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
- 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

ScaDS DRESDEN LEIPZIG

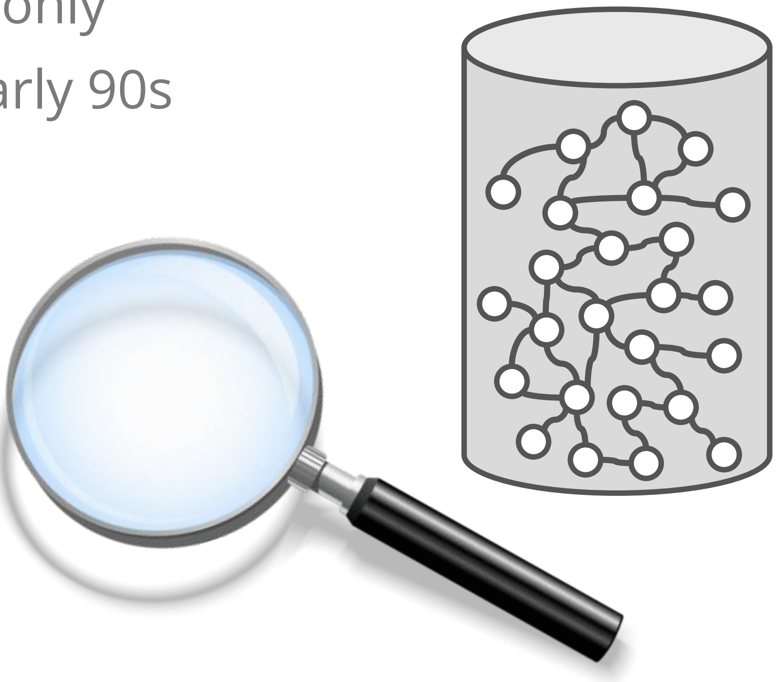
AGENDA

- Motivation
  - graph data
  - requirements
- Graph data systems
  - graph database systems
  - distributed graph processing systems (Pregel, etc.)
  - distributed graph dataflow systems (GraphX, Gelly)
- Gradoop
  - architecture
  - Extended Property Graph Model (EPGM)
  - implementation and performance evaluation
- Open challenges

ScaDS DRESDEN LEIPZIG

GRAPH DATABASES

- First Generation:
  - research prototypes only
  - peak popularity in early 90s
- Second Generation:
  - NoSQL movement
  - commercial systems



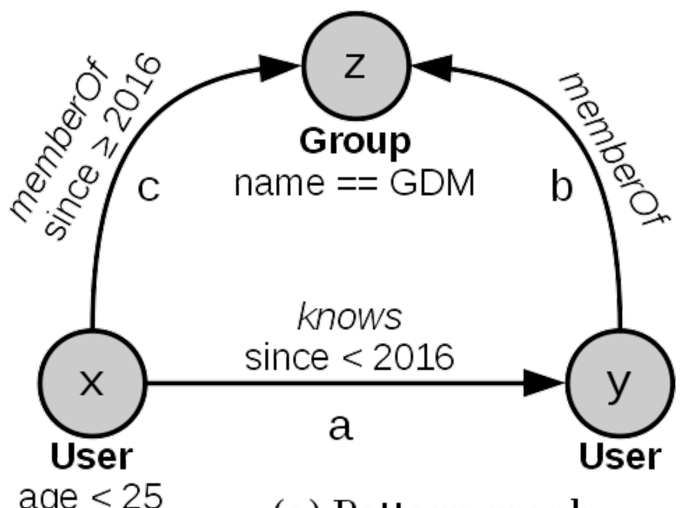
ScaDS DRESDEN LEIPZIG

RECENT GRAPH DATABASE SYSTEMS

- graph data model
  - mostly property graphs, RDF or generic graphs
  - different vertex and edge data
  - graph operators (traversal, pattern matching) / queries
- application scope
  - mostly queries/OLTP on small graph portions
  - some support for analytical queries/computations (analyze whole graph, e.g., page rank)

PROPERTY GRAPH MODEL

- popular data model for commercial graph DBMS
  - de-facto industry standard TinkerPop
  - query languages Gremlin (TinkerPop) and Cypher (Neo4j, openCypher)
- query example (pattern matching)



(a) Pattern graph

```
MATCH  (x:User)-[a:knows]->(y:User),
       (x)-[b:memberOf]->(z:Group),
       (y)-[c:memberOf]->(z)
WHERE  x.age < 25 AND z.name = 'GDM' AND
       a.since < 2016 AND c.since >= 2016
RETURN y.name, x.name
```

(b) Cypher

ScaDS DRESDEN LEIPZIG

SYSTEM COMPARISON

System	Data Model: RDF/SPARQL	Data Model: PGM/TinkerPop	Data Model: Generic	Scope: OLTP/Queries	Scope: Analytics	Storage: Approach	Storage: Replication	Storage: Partitioning
Apache Jena TBD	✓/✓			✓		native		
AllegroGraph	✓/✓			✓		native	✓	
MarkLogic	✓/✓			✓		native	✓	✓
Ontotext GraphDB	✓/✓			✓		native	✓	
Oracle Spatial and Graph	✓/✓			✓		native	✓	
Virtuoso	✓/✓			✓		relational	✓	✓
TripleBit	✓/✓			✓		native		
Blazegraph	✓/✓	✓/✓		✓	✓	native RDF	✓	✓
IBM System G	✓/✓	✓/✓	✓	✓	✓	native PGM, wide column store	✓	✓
Stardog	✓/✓	✓/✓		✓	○	native RDF	✓	
SAP Active Info. Store		✓/-		✓		realtional		
ArangoDB		✓/✓		✓		document store	✓	✓
InfiniteGraph		✓/✓		✓		native	✓	✓
Neo4j		✓/✓		✓		native	✓	
Oracle Big Data		✓/✓			✓	key value store	✓	✓
OrientDB		✓/✓		✓		document store	✓	✓
Sparksee		✓/✓		✓		native	✓	
SQLGraph		✓/✓		✓		relational		
Titan		✓/✓		✓	○	wide column store, key value store	✓	✓
HypergraphDB			✓	✓		native		

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

ScaDS DRESDEN LEIPZIG

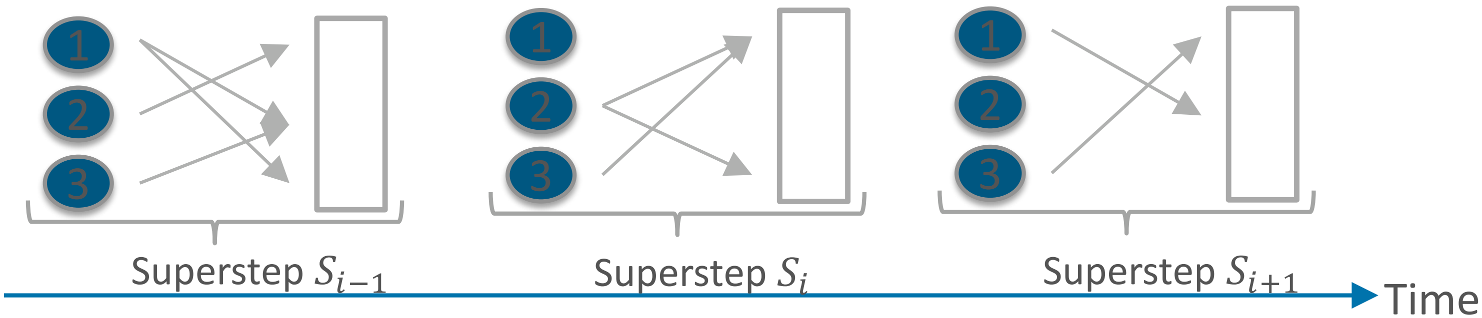
GRAPH PROCESSING SYSTEMS

- goal: better support for scalable/distributed graph mining
  - page rank, connected components, clustering, frequent subgraphs, …
  - mostly generic graphs only (e.g., directed multigraphs)
- early approaches based on MapReduce
  - iterative processing via control program and multiple MR programs
  - unintuitive programming and limited performance (high communication and I/O costs)
- „newer“ computation models pioneered by Google Pregel
  - vertex-centric programming („Think Like a Vertex“)
  - Bulk-synchronous-parallel (BSP) computation
  - In-memory storage of graph data

ScaDS DRESDEN LEIPZIG

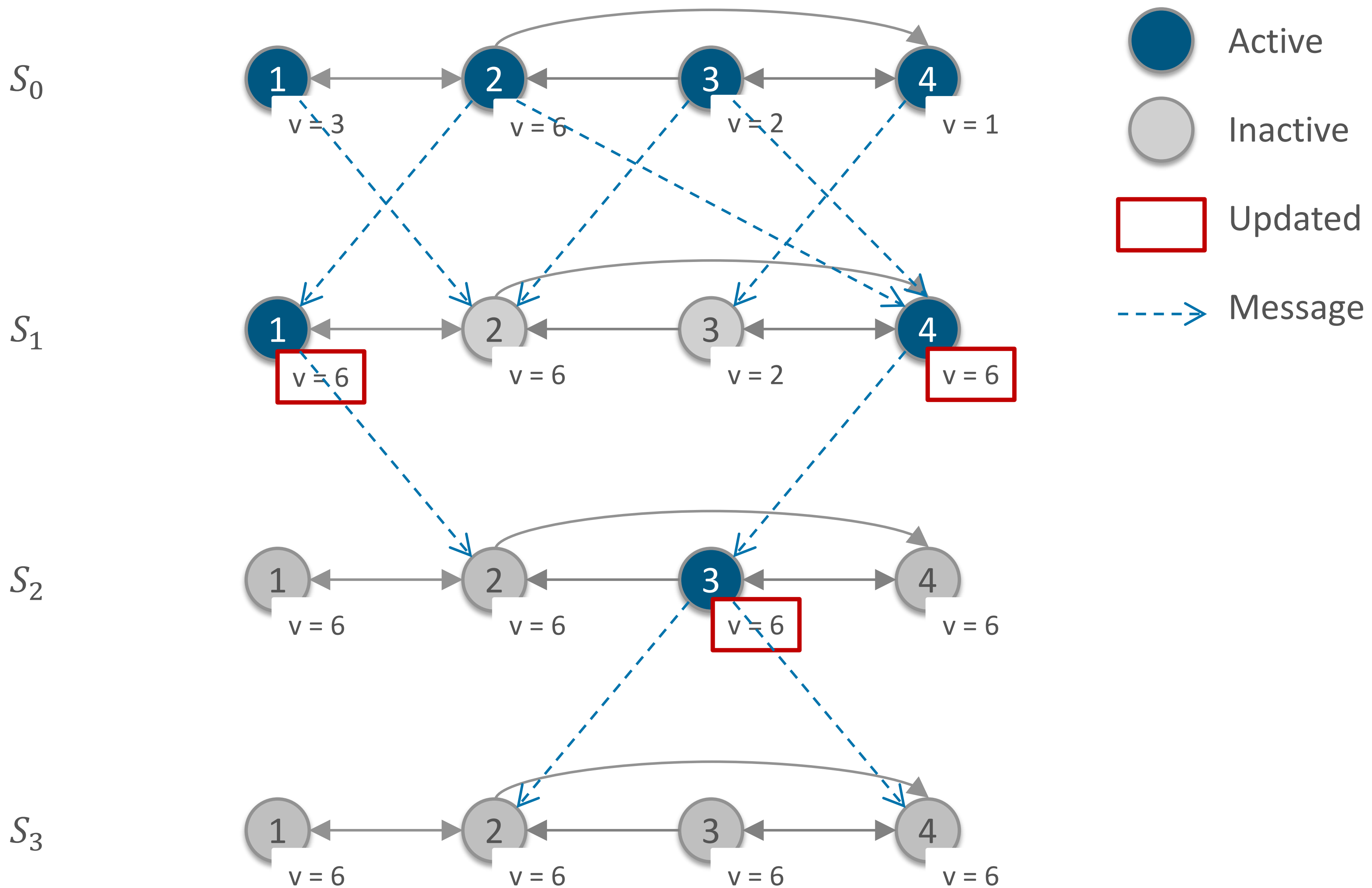
VERTEX-CENTRIC PROCESSING

- parallel and synchronized execution of vertex *compute* function
- vertex keeps state about itself
- compute function
  - reads incoming messages,
  - updates vertex state (value)
  - sends information to neighboring vertices
- vertices can deactivate themselves (call voteToHalt() function)
- iterative execution within *supersteps* until there are no active vertices or messages anymore (bulk-synchronus-parallel execution)



ScaDS DRESDEN LEIPZIG

EXAMPLE – MAXIMUM VALUE



ScaDS DRESDEN LEIPZIG

ALTERNATE MODELS

- alternate execution models
  - partition-centric ("Think-like-a-graph"): synchronized execution of compute functions for entire partions (all vertices on one worker)
  - asynchronous: to avoid many idle vertices/workers with skewed degree distributions

- Gather-Apply-Scatter (GAS) programming model
  - **gather** function: aggregates/combines messages
  - **apply** function: preprocesses incoming messages and updates vertex state
  - **scatter** function: uses vertex state to produce outgoing messages
  - Goals: reduce network traffic and better workload balancing for graphs graphs with highly skewed degree distribution

- Scatter-Gather programming model
  - user provides **vertex** and **edge** functions:
  - vertex function uses all incoming messages to modify vertex value
  - edge function uses vertex value to generate a message
  - susceptible to execution skew (like vertex-centric)

GRAPH PROCESSING SYSTEMS

	Language	Computation Model	BSP	async.	Agg.	Add	Remove	Comb.
Pregel	C++	Pregel	✓		✓	✓	✓	✓
Giraph	Java	Pregel	✓		✓			✓
GPS	Java	Pregel	✓		✓			✓
Mizan	C++	Pregel	✓	✓	✓	✓		n.a.
GraphLab	C++	GAS	✓	✓	✓	✓		n.a.
GraphChi	C++, Java	Pregel		✓	✓	✓	✓	n.a.
Signal/Collect	Java	Scatter-Gather	✓	✓				n.a.
Chaos	Java	Scatter-Gather	✓	✓				n.a.
Giraph++	Java	Partiton-centric	✓		✓	✓	✓	✓
GraphX	Scala, Java	GAS	✓		✓			n.a.
Gelly	Scala, Java	GSA, Scatter-Gather, Pregel	✓		✓	✓	✓	n.a.

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

GRAPH DATAFLOW SYSTEMS

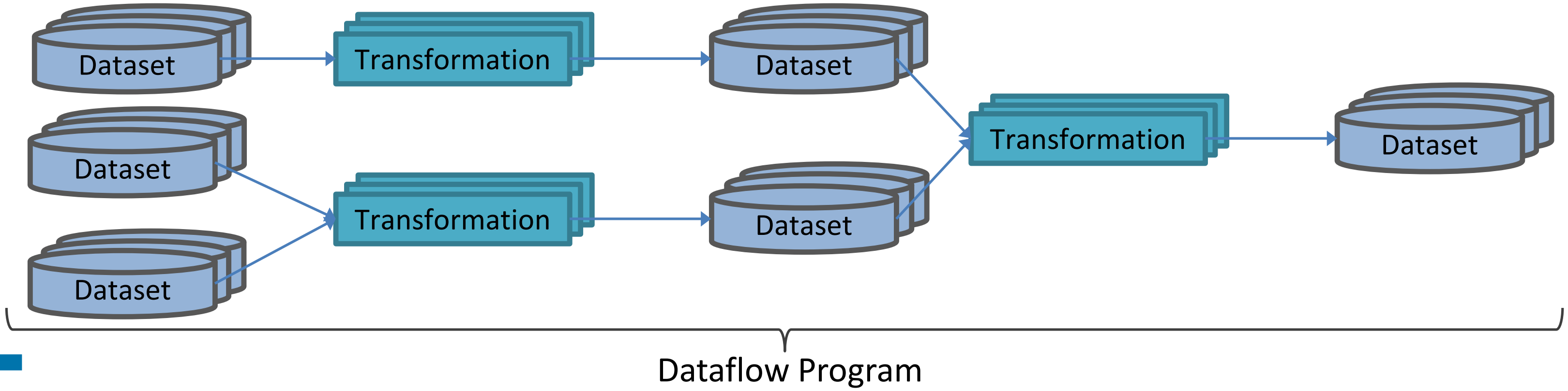
- Graph processing systems are specialized systems
  - tailored programming abstractions for fast execution of a single iterative graph algorithm
- complex analytical problems often require the combination of multiple techniques, e.g.:
  - creation of combined graph structures from different sources (data extraction, transformation and integration)
  - different analysis steps: queries, iterative graph processing, machine learning, …
- Dataflow systems can combine such tasks within dataflow programs/workflows/scripts for distributed execution
  - 1^{st} generation: MapReduce workflows
  - Apache Spark/Flink: in-memory dataflow systems

ScaDS DRESDEN LEIPZIG

DATAFLOW SYSTEMS

- Distributed in-memory dataflow systems (e.g., Apache Spark, Apache Flink)
  - general-purpose operators (e.g. map, reduce, filter, join) => **transformations**
  - specialized libraries (e.g. machine learning, graph analysis)
  - holistic view enables optimizations (operator reordering, caching, etc.)

- **Dataset** := distributed collection of data objects
- **Transformation** := operation on datasets (higher-order function)
- **Dataflow Programm** := composition of transformations



GRAPH DATAFLOW SYSTEMS

- Graph abstraction on top of a dataflow system (e.g., Gelly on Apache Flink and GraphX on Apache Spark)
  - generic graph representation
  - graph operations / transformations / processing
- Graph transformations / operations
  - **mutation**: adding / removing of vertices and edges
  - **map**: modification of vertex and edge values
  - **subgraph**: find subgraph for user-defined vertex / edge predicates
  - **join**: combination of vertex / edge datasets with other datasets
  - **union/difference/intersect:** combine two graphs into one
- Graph processing
  - Gelly implements **Pregel**, **GAS**, **Scatter-Gather** by using native Flink iteration functions
  - GraphX implements **GAS** based on Spark Iterations

Gelly

GraphX

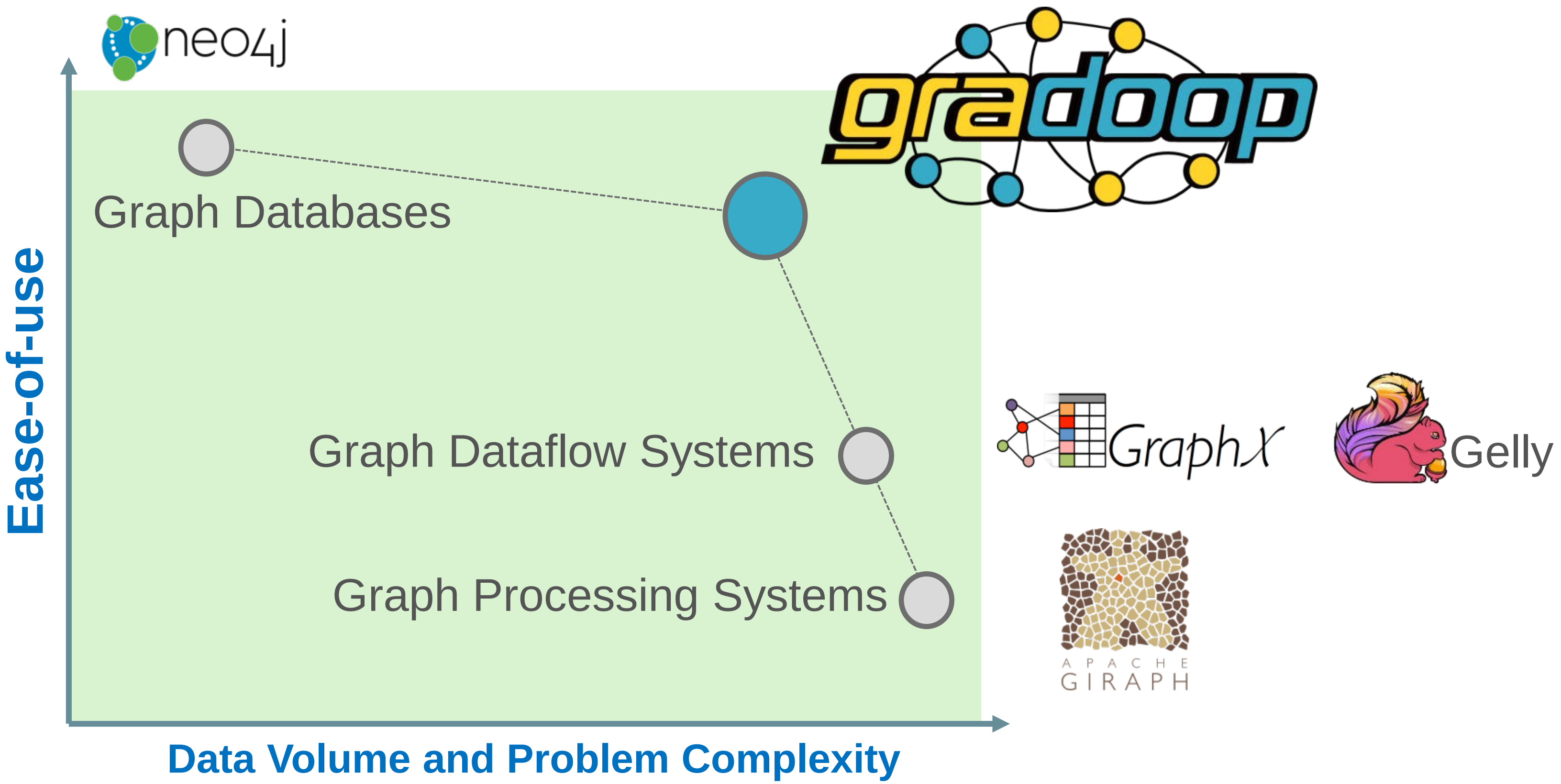
ScaDS DRESDEN LEIPZIG

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
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 for scalable (distributed) graph data management and analytics supporting a rich graph data model and queries


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

- Motivation
  - graph data
  - requirements
- Graph data systems
  - graph database systems
  - distributed graph processing systems (Pregel, etc.)
  - distributed graph dataflow systems (GraphX, Gelly)
- Gradoop
  - architecture
  - Extended Property Graph Model (EPGM)
  - implementation and performance evaluation
- Open challenges

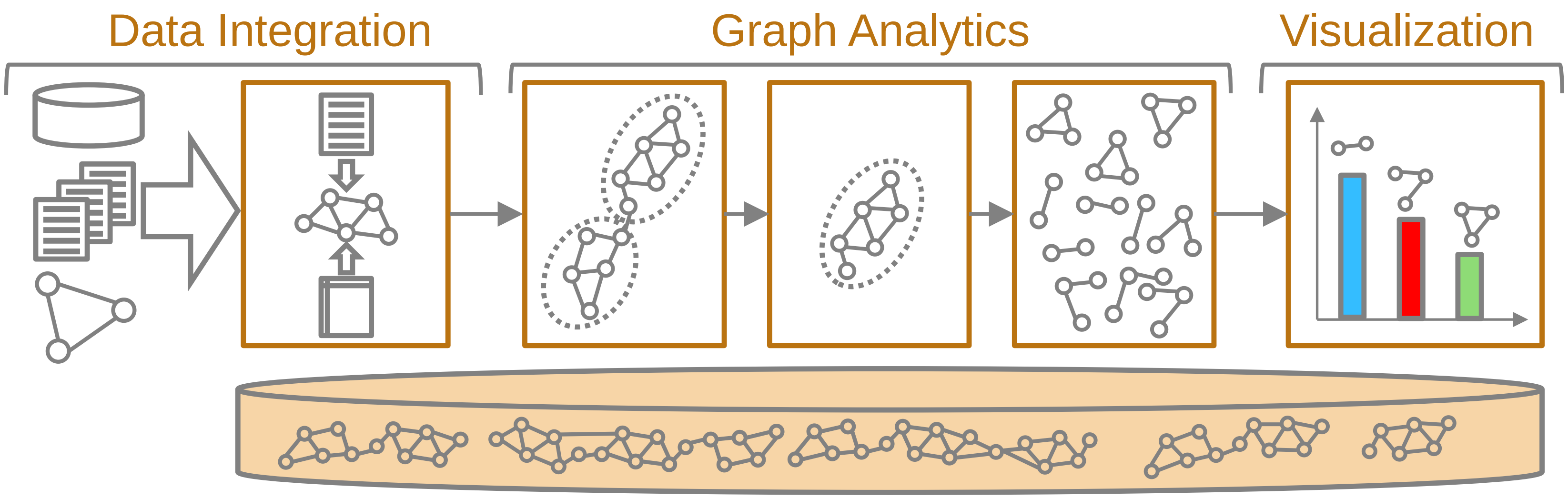
ScaDS DRESDEN LEIPZIG

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

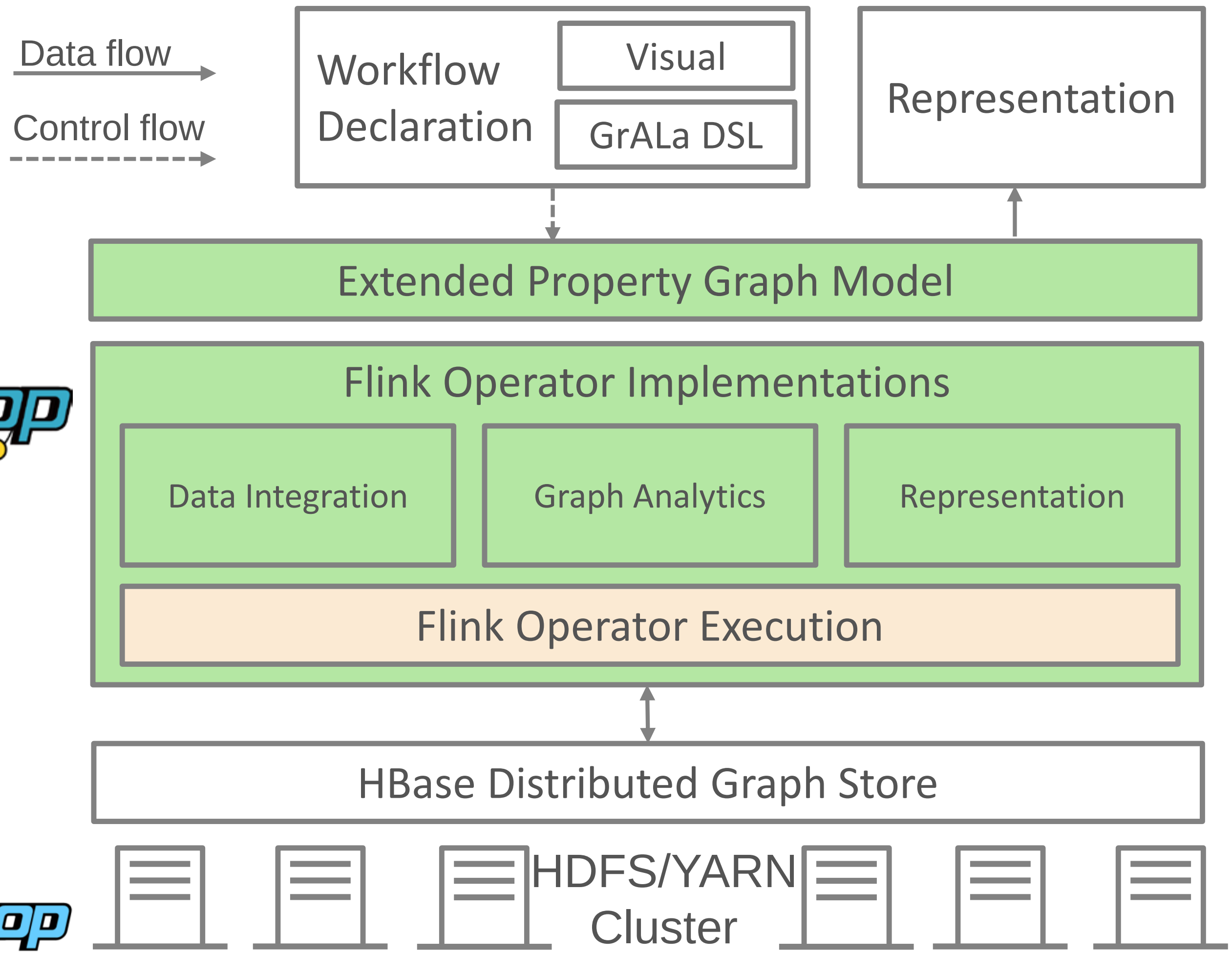
END-TO-END GRAPH ANALYTICS



- **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

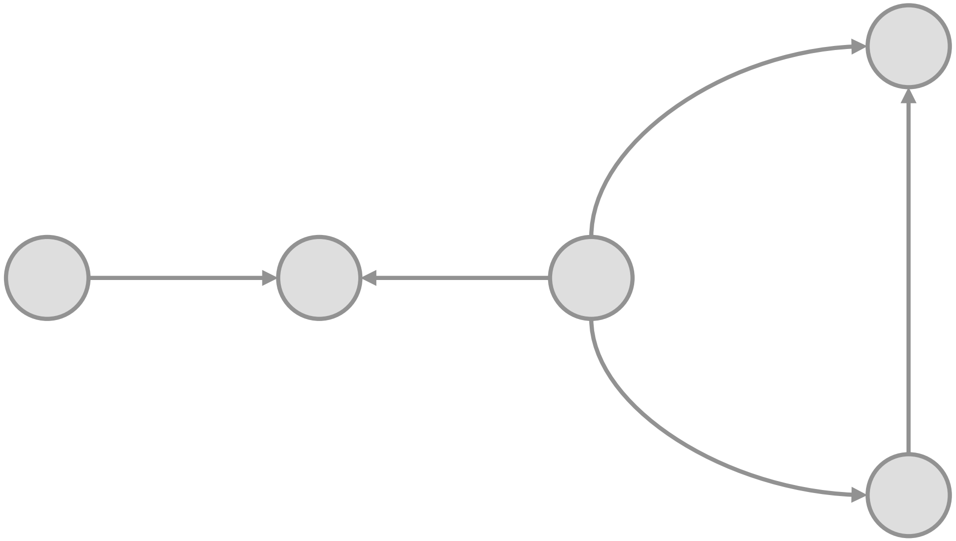
HIGH LEVEL ARCHITECTURE

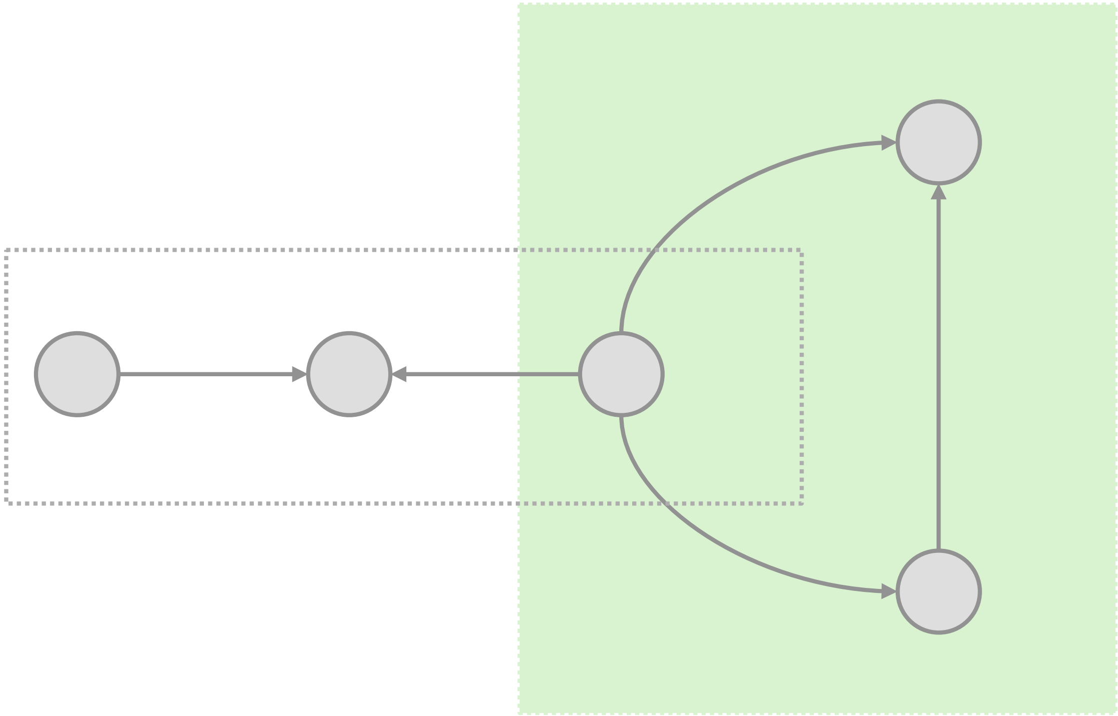


EXTENDED PROPERTY GRAPH MODEL (EPGM)

- 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

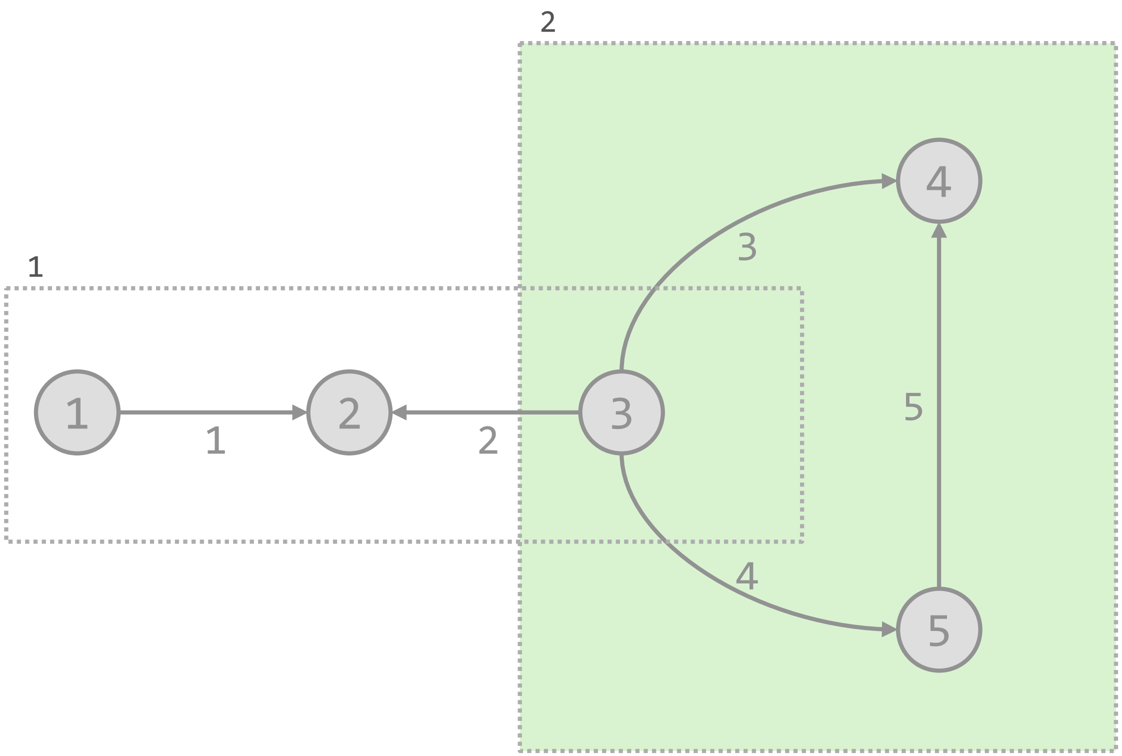
- **Vertices and directed Edges**



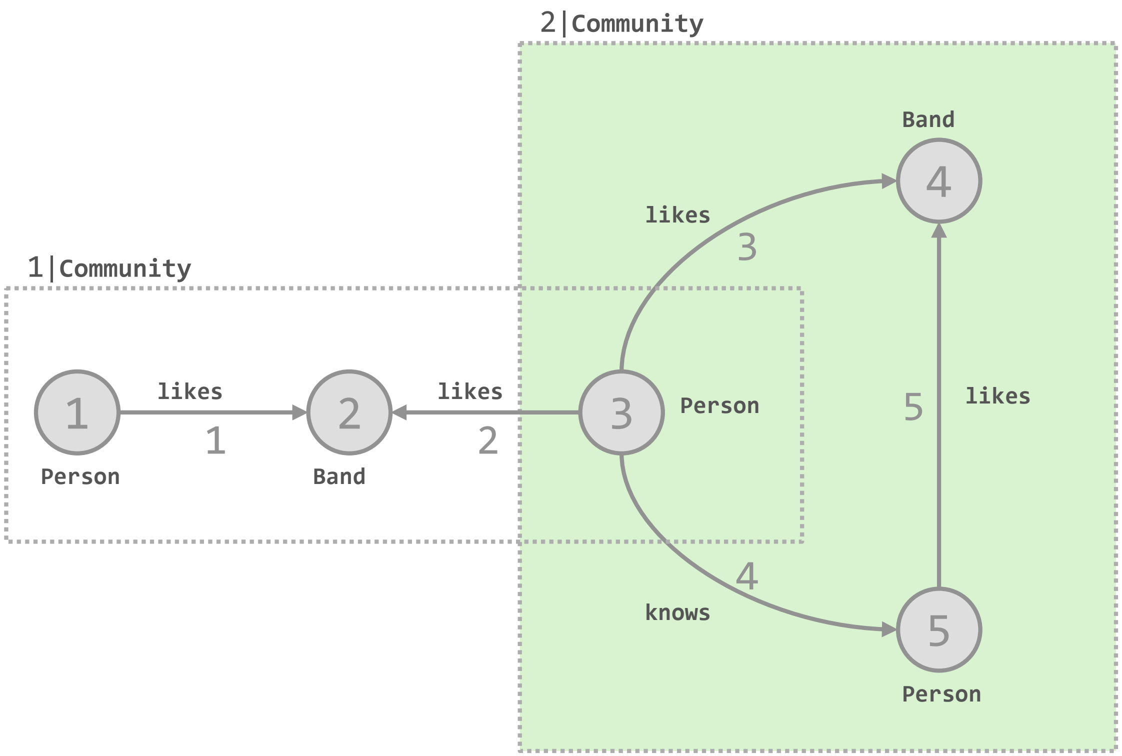


- Vertices and directed Edges
- **Logical Graphs**

ScaDS
DRESDEN LEIPZIG

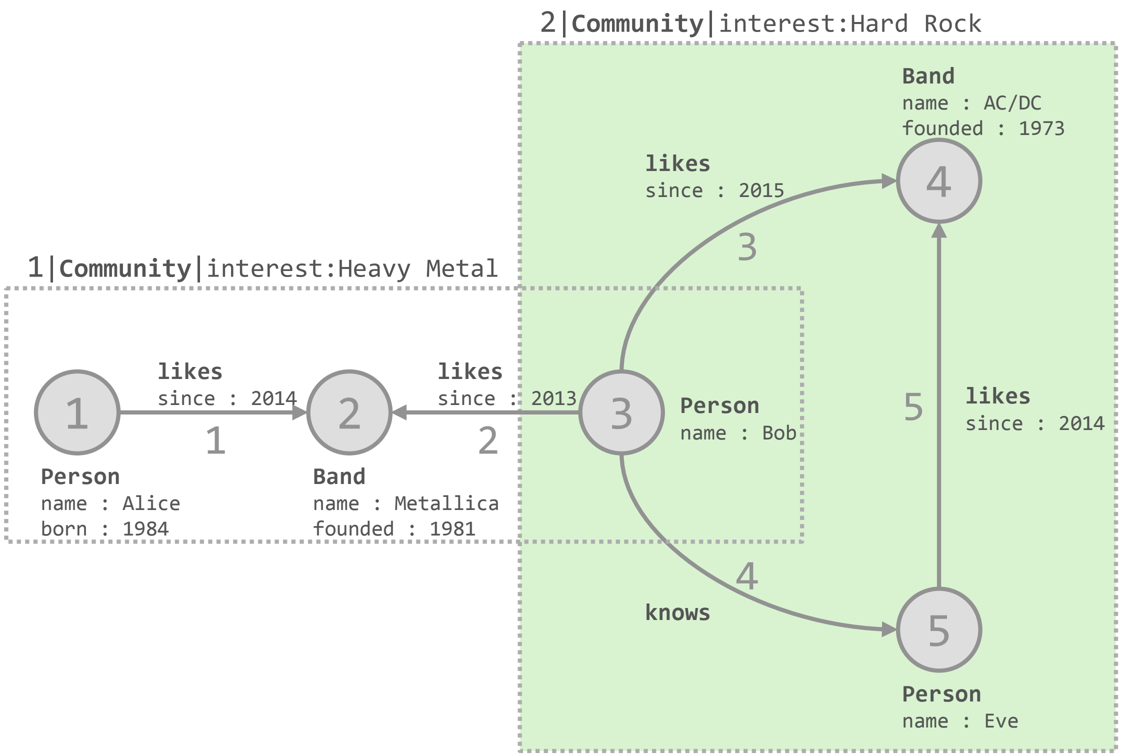


- Vertices and directed Edges
- Logical Graphs
- **Identifiers**



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

ScaDS DRESDEN LEIPZIG



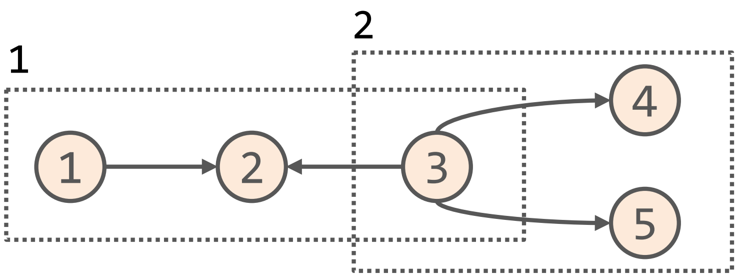
- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- **Properties**

Operators

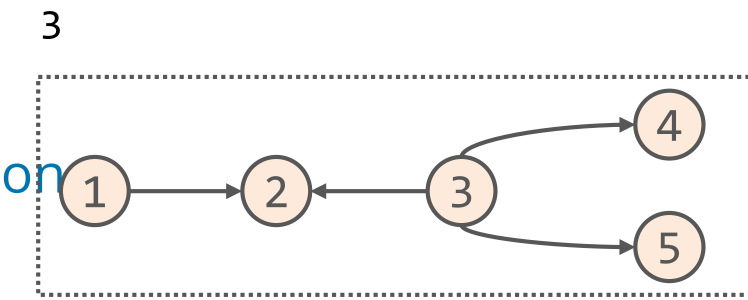
	Operators		Algorithms
	Unary	Binary	
Logical Graph	Aggregation	Combination	Gelly Library
	Pattern Matching	Overlap	BTG Extraction
	Transformation	Exclusion	Adaptive Partitioning
	Grouping	Equality	
	Subgraph		
	Call *		
Graph Collection	Selection	Union	Frequent Subgraphs
	Distinct	Intersection	
	Sort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

* auxiliary

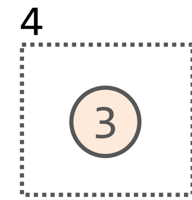
BASIC BINARY OPERATORS



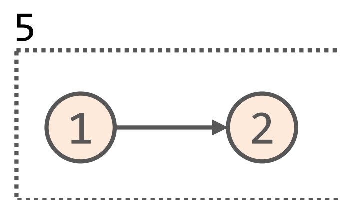
Combination



Overlap



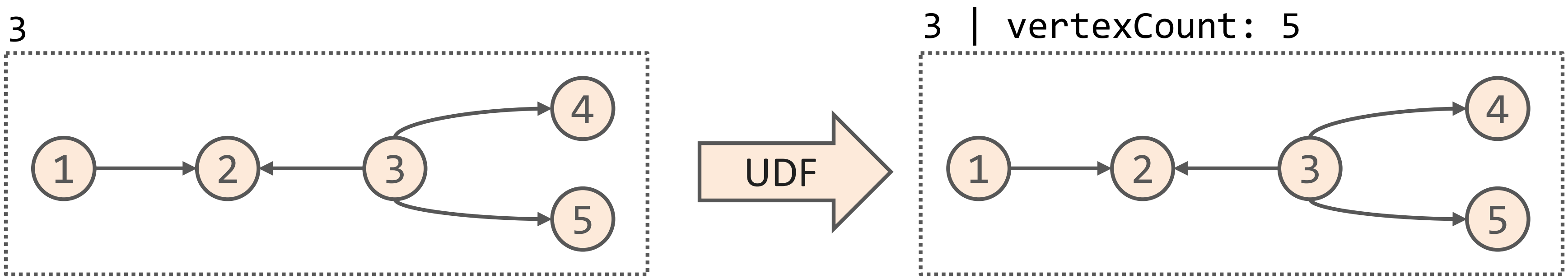
Exclusion




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

ScaDS DRESDEN LEIPZIG

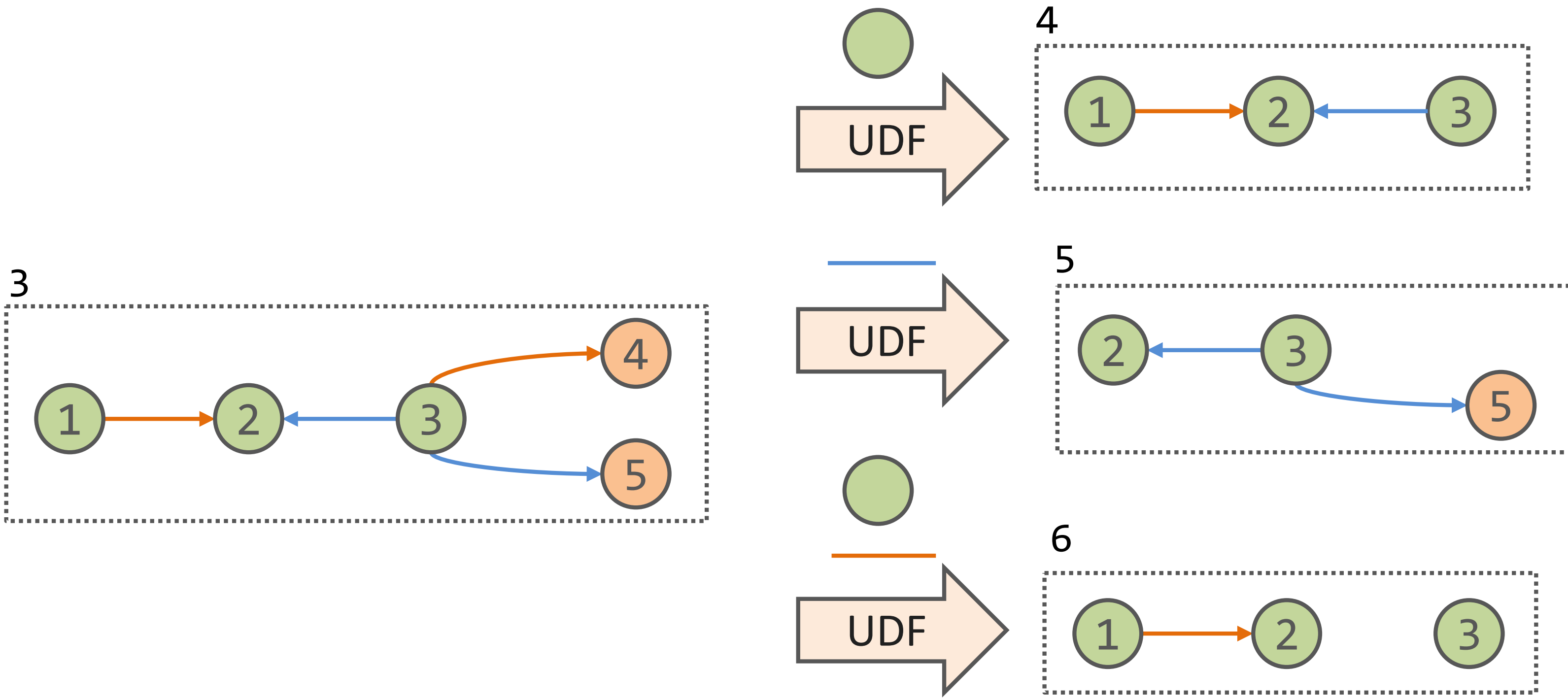
AGGREGATION




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

ScaDS DRESDEN LEIPZIG

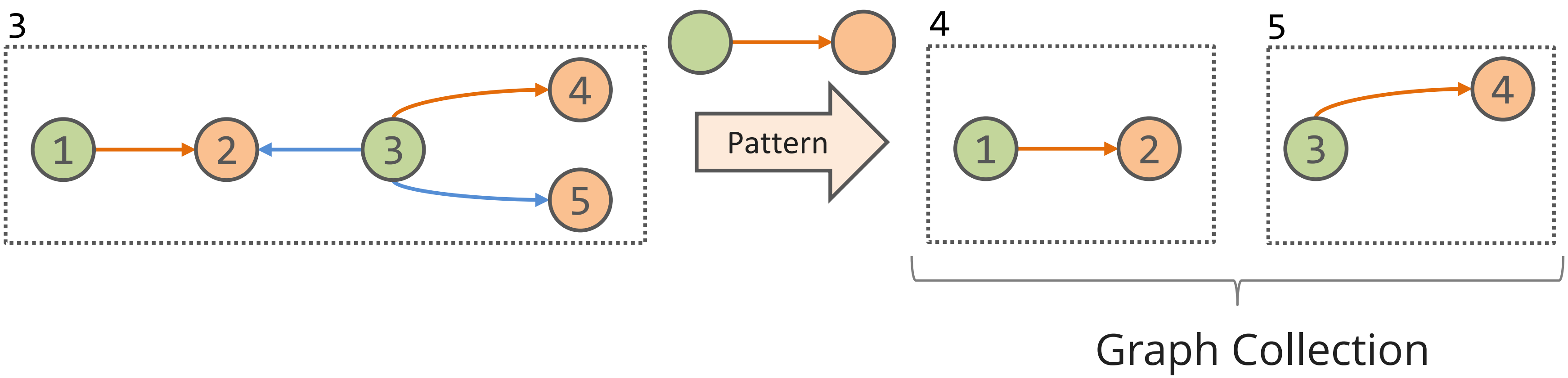
SUBGRAPH




```
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)");
```

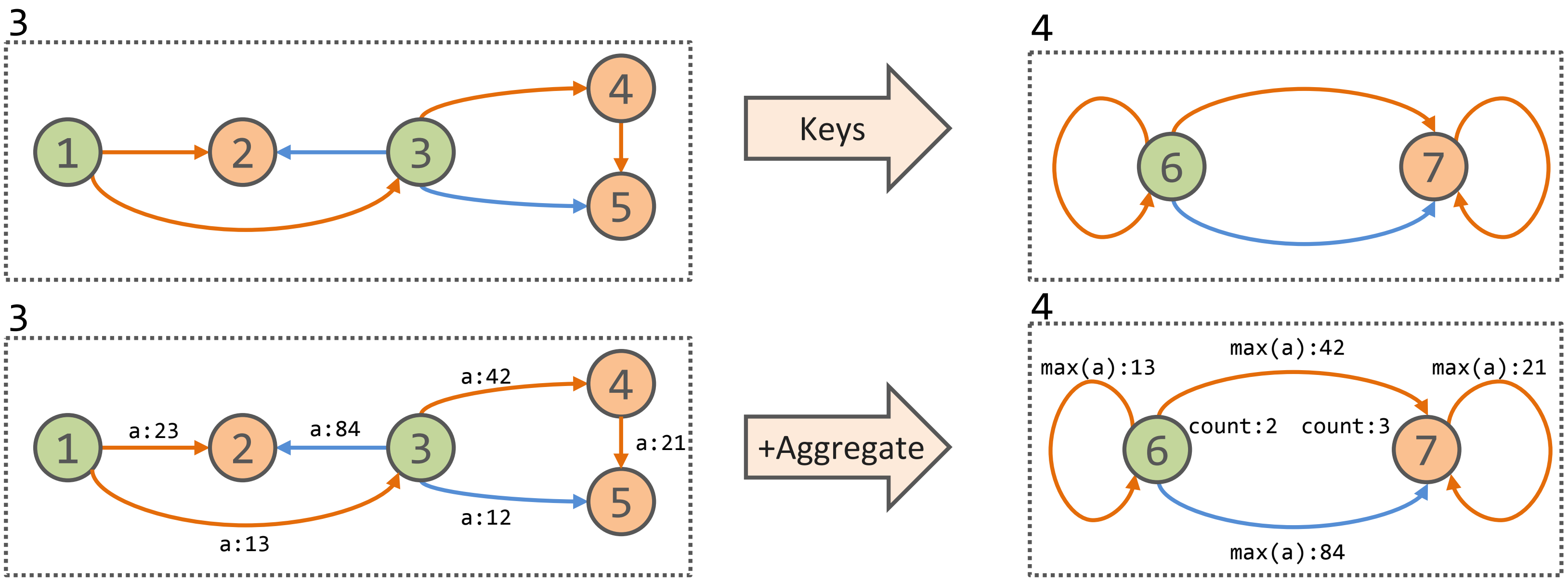
- **new:** support of *Cypher query language* for pattern matching*

```
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

ScaDS DRESDEN LEIPZIG

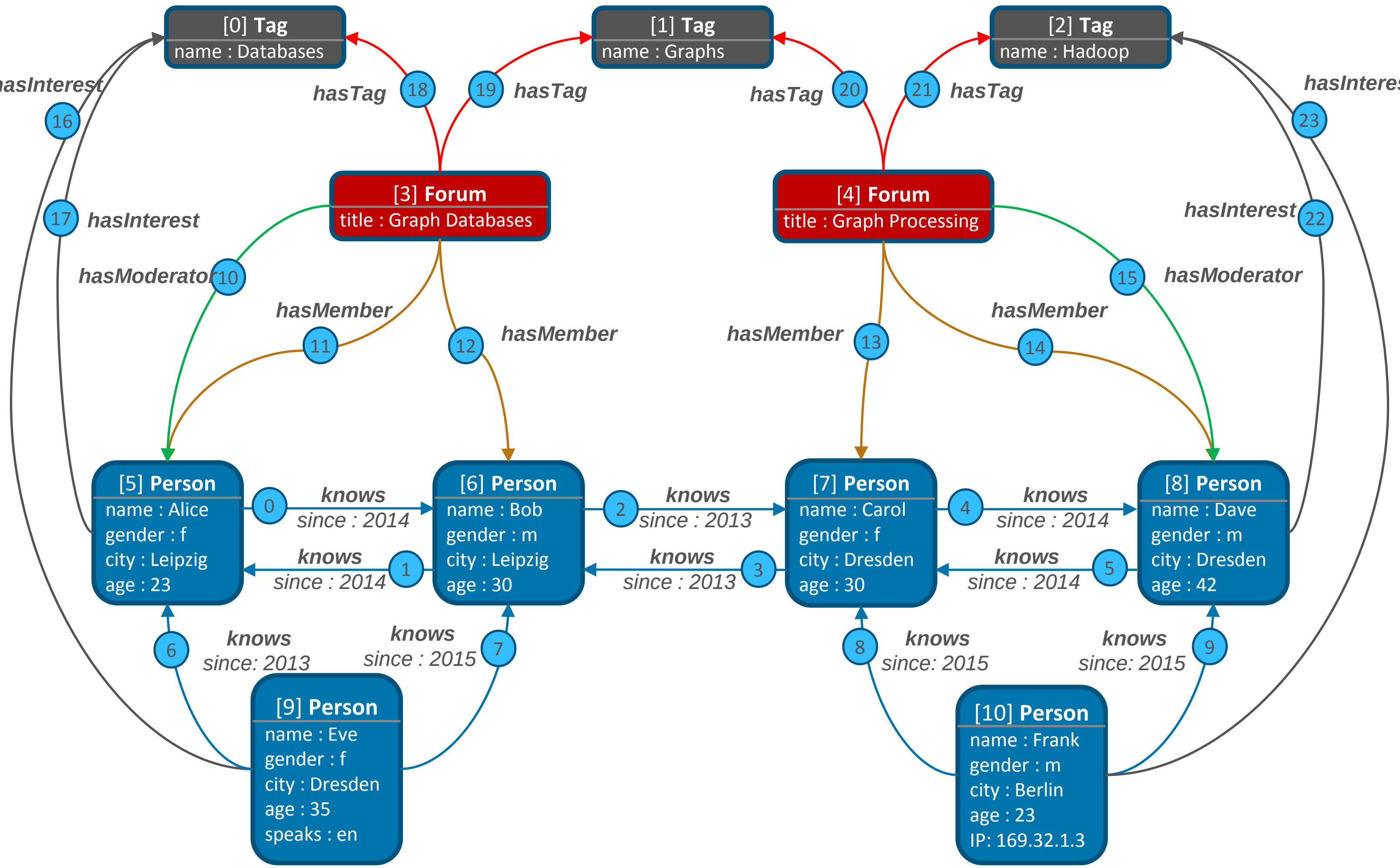
GROUPING




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

ScaDS DRESDEN LEIPZIG

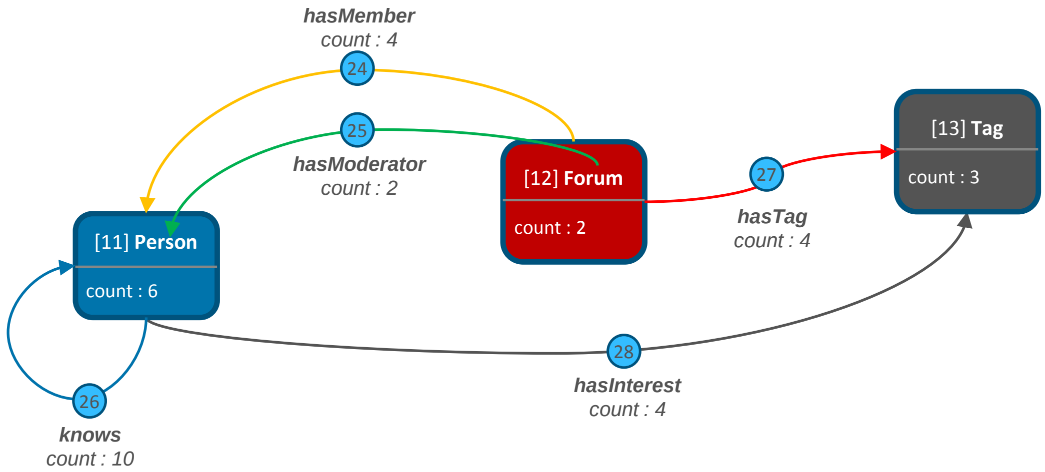
SAMPLE GRAPH



ScaDS DRESDEN LEIPZIG

GROUPING: TYPE LEVEL *(SCHEMA GRAPH)*

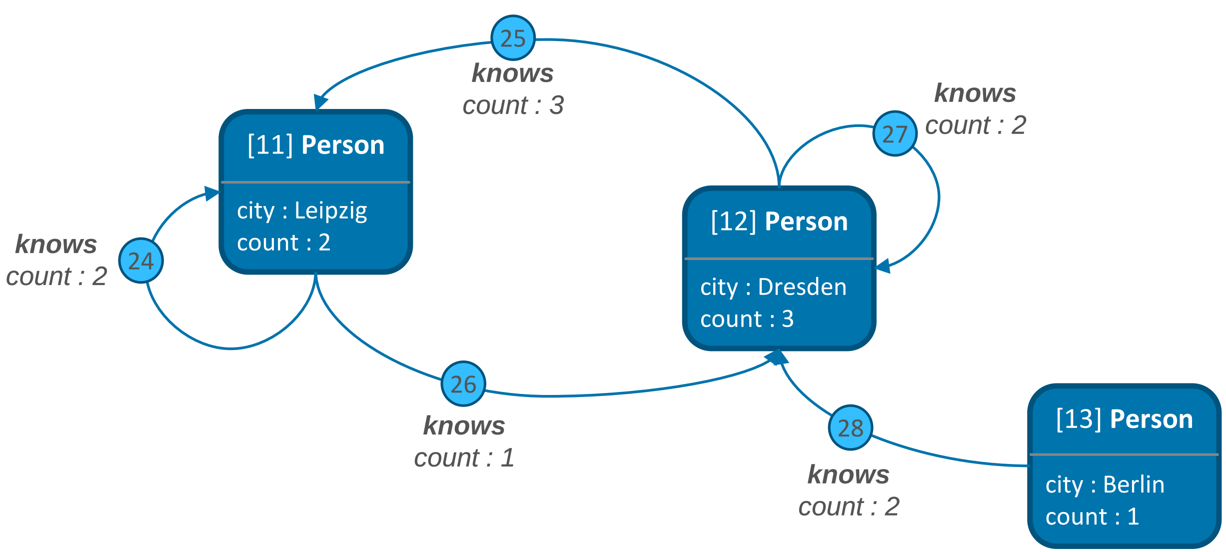
```
vertexGrKeys = [:label]
edgeGrKeys   = [:label]
sumGraph     = databaseGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
```



ScaDS DRESDEN LEIPZIG

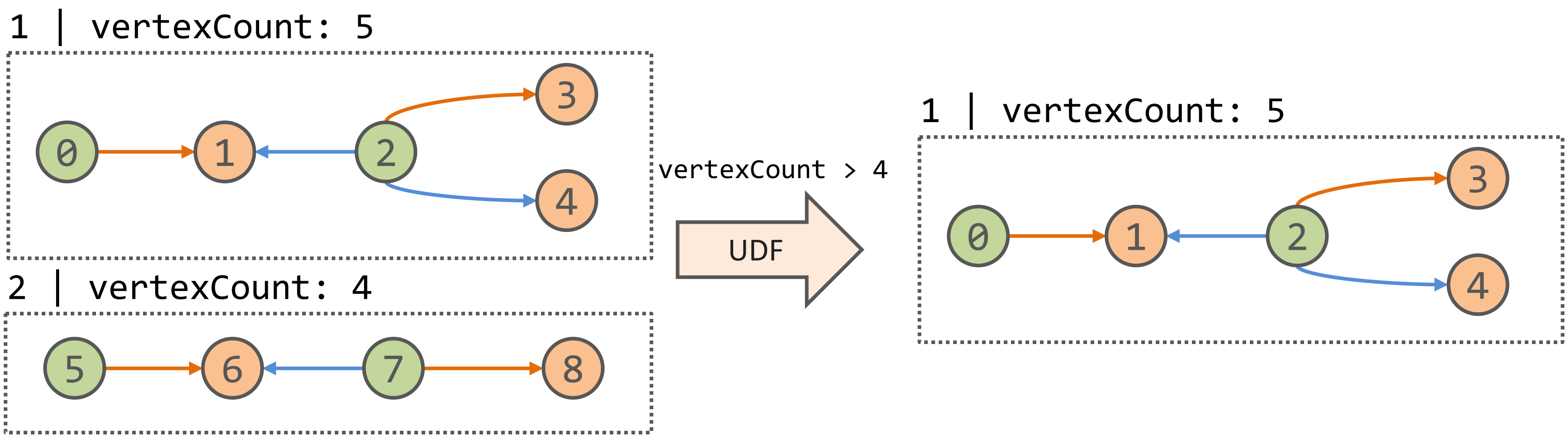
GROUPING: PROPERTY-SPECIFIC

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



ScaDS DRESDEN LEIPZIG

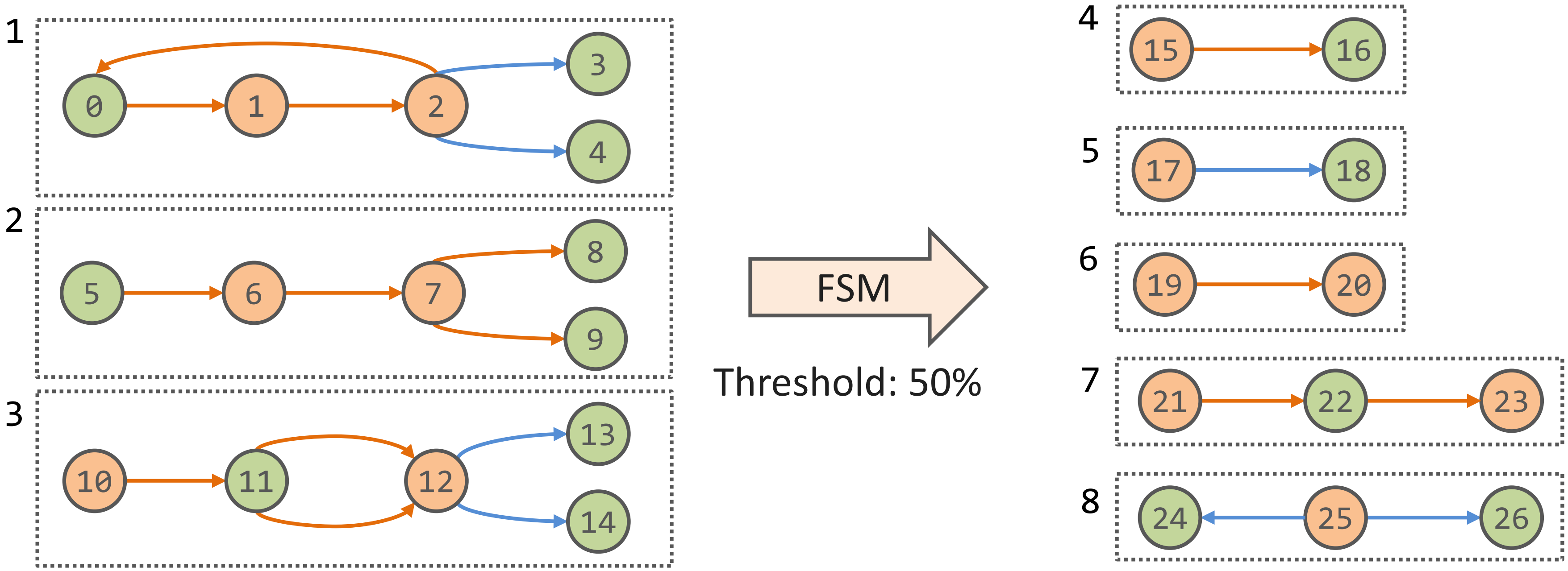
SELECTION




```
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 and evaluation

GRAPH REPRESENTATION

EPGMGraphHead

Id	Label	Properties

POJO → **DataSet**<EPGMGraphHead>

EPGMVertex

Id	Label	Properties	Graphs

POJO → **DataSet**<EPGMVertex>

EPGMEdge

Id	Label	Properties	SourceId	TargetId	Graphs

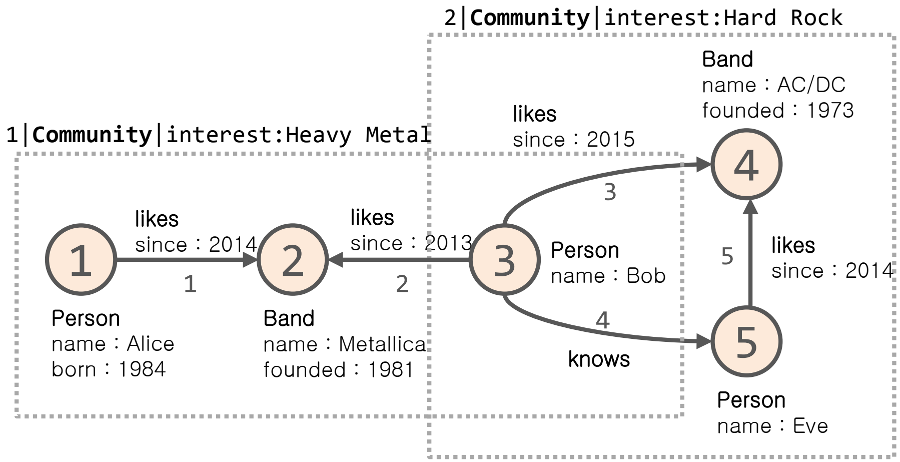
POJO → **DataSet**<EPGMEdge>

EPGMVertex

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

ScaDS DRESDEN LEIPZIG

GRAPH REPRESENTATION: EXAMPLE



DataSet<EPGMGraphHead>

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

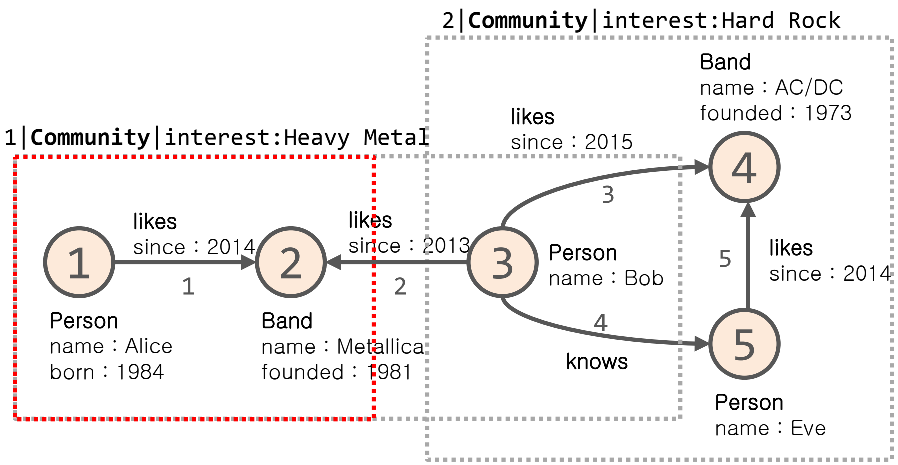
DataSet<EPGMVertex>

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}

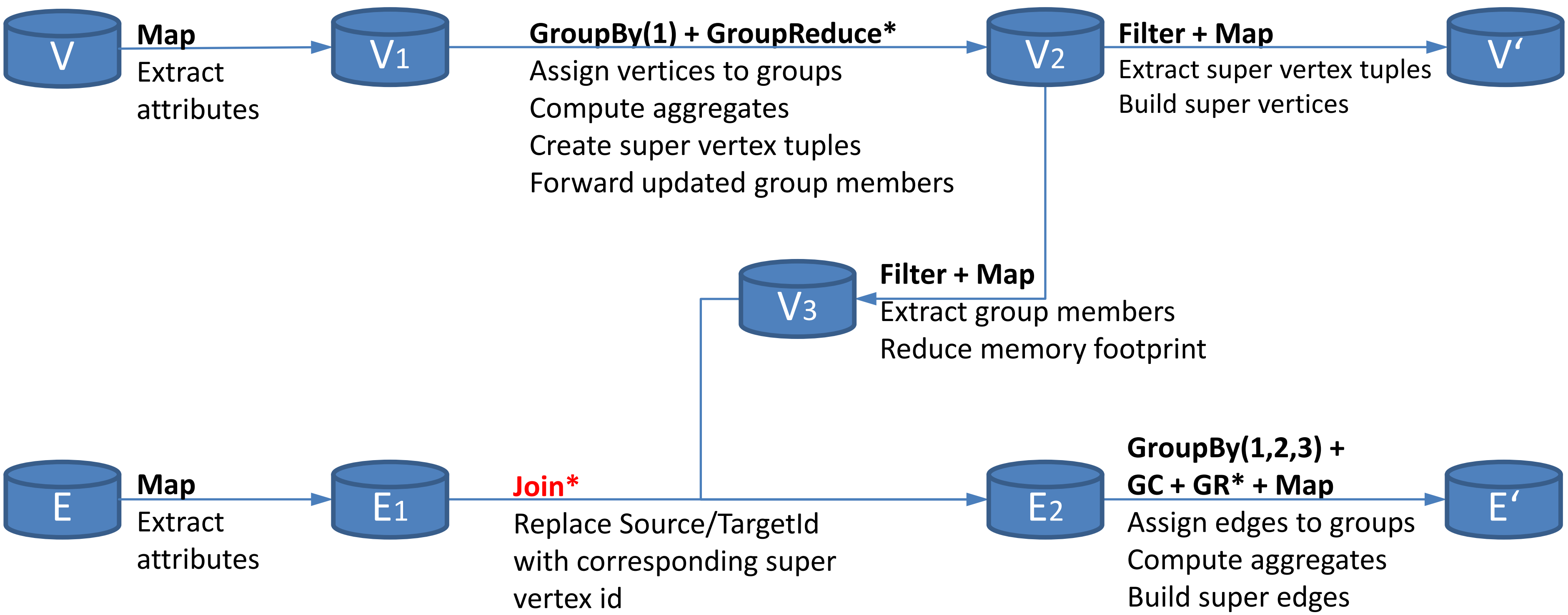
OPERATOR IMPLEMENTATION



Exclusion

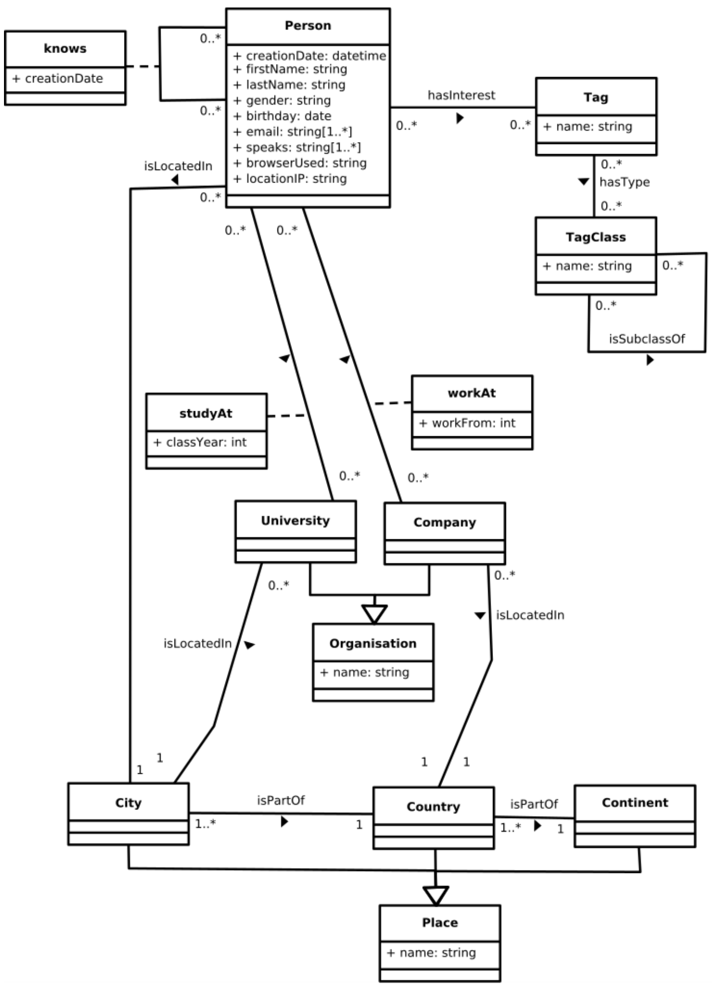
```
// input: firstGraph (G[1]), secondGraph (G[2])

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>());
```

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

ScaDS DRESDEN LEIPZIG

TEST WORKFLOW: SUMMARIZED COMMUNITIES

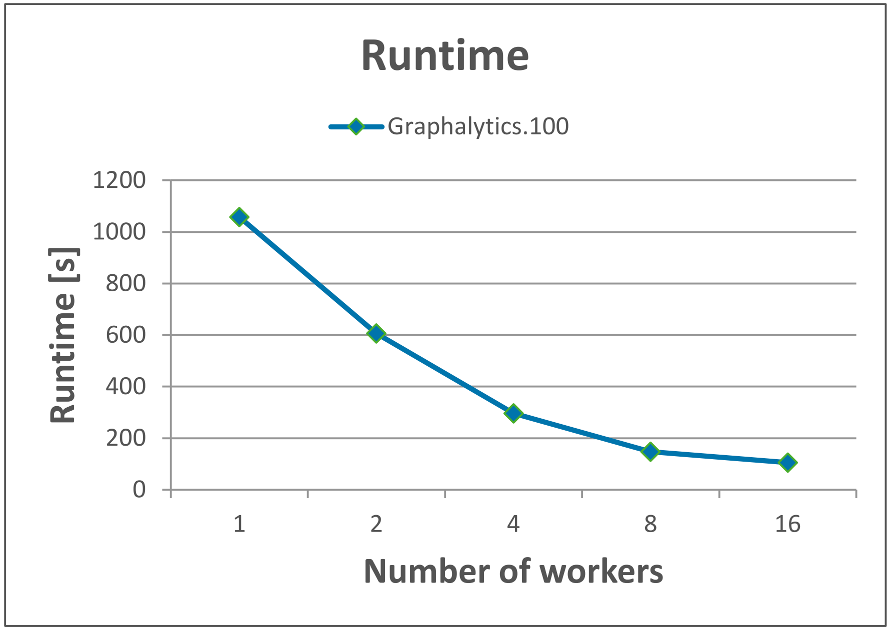
1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities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

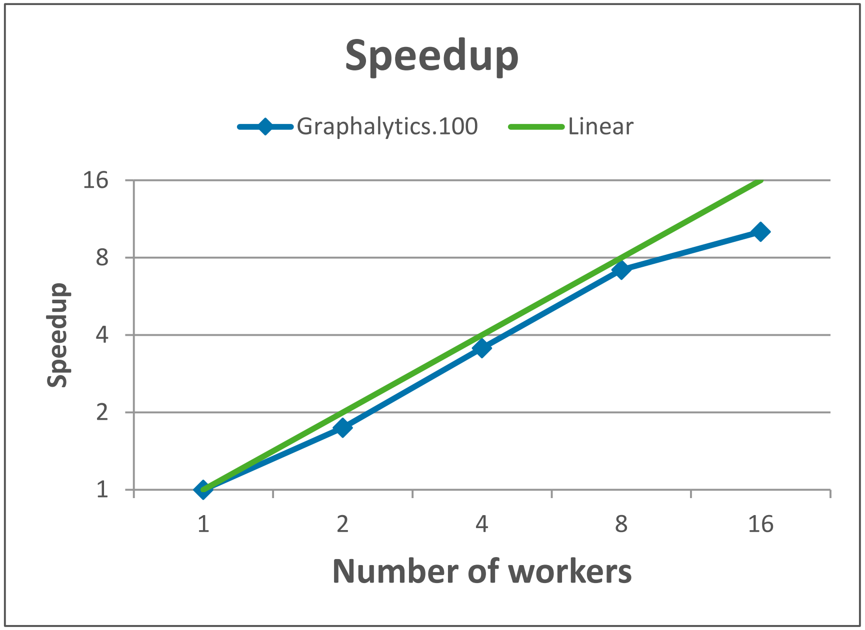
```
return socialNetwork
  // 1) extract subgraph
  .subgraph((vertex) → {
      return vertex.getLabel().toLowerCase().equals(person);
  }, (edge) → { return edge.getLabel().toLowerCase().equals(knows); })
  // project to necessary information
  .transform((current, transformed) → { return current;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



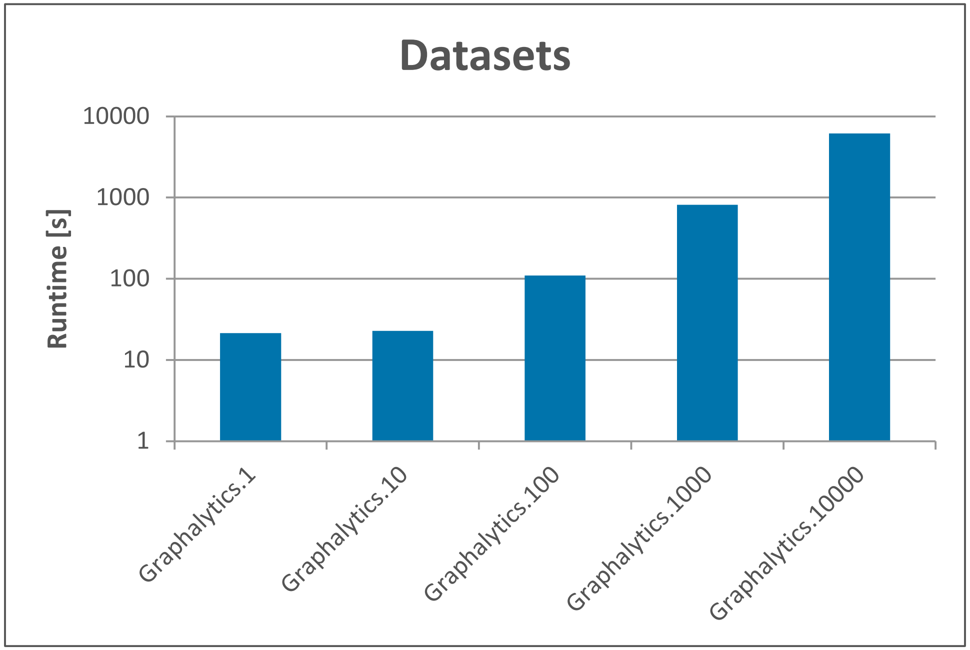


Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

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

ScaDS DRESDEN LEIPZIG

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

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	no
data integration	no	no	no	(yes)
visualization	(yes)	no	no	limited

ScaDS DRESDEN LEIPZIG

AGENDA

- Motivation
  - graph data
  - requirements
- Graph data systems
  - graph database systems
  - distributed graph processing systems (Pregel, etc.)
  - distributed graph dataflow systems (GraphX, Gelly)
- Gradoop
  - architecture
  - Extended Property Graph Model (EPGM)
  - implementation and performance evaluation
- Open challenges

ScaDS DRESDEN LEIPZIG

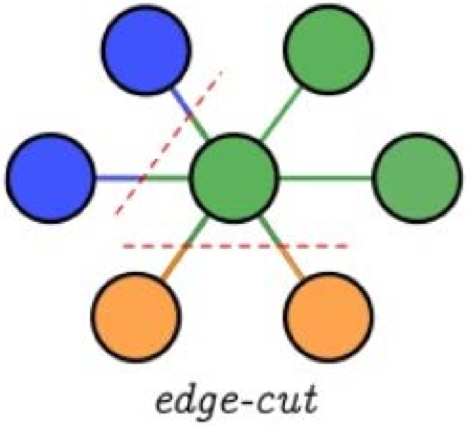
CHALLENGES

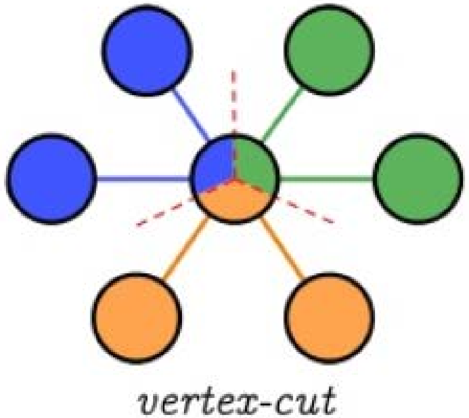
- Graph data allocation and partitioning
- Benchmarking and evaluation of graph data systems
- Graph-based data integration and knowledge graphs
- Analysis of dynamic graphs
- Interactive graph analytics

ScaDS DRESDEN LEIPZIG

GRAPH DATA ALLOCATION / PARTITIONING

- distributed graph processing depends on suitable graph allocation/partitioning
  - minimize communication for graph analysis
  - load balancing
- goal: balanced vertex distribution with minimal number of edges between partitions (edge cut)
  - vertex cut: balanced edge distribution with minimal replication of vertices (PowerGraph, Spark GraphX)





ScaDS DRESDEN LEIPZIG

GRAPH DATA ALLOCATION / PARTITIONING (2)

- hash-based vertex partitioning prevalent but not optimal
  - vertex neighbors frequently in different partitions -> high communication overhead
- multilevel graph partitioning (e.g., METIS)
  - expensive to determine / static
- newer approaches for adaptive allocation
  - Stanton/Kliot (KDD2012), Mondal/Deshpande (Sigmod2012), Huang/Abadi (VLDB2016)

ScaDS DRESDEN LEIPZIG

BENCHMARKING AND EVALUATION

- many comparative evaluations between graph DBMS and graph processing systems (Han -VLDB14, Lu -VLDB14, ...)
  - many differences in considered systems. workloads, configurations, etc
  - early systems using Map/reduce or Giraph are outperformed by newer graph processing systems
  - few results for Spark GraphX, Flink Gelly
- Benchmark efforts for graph data analysis
  - e.g., LinkBench, LDBC, gMark
  - only few results so far

ScaDS DRESDEN LEIPZIG

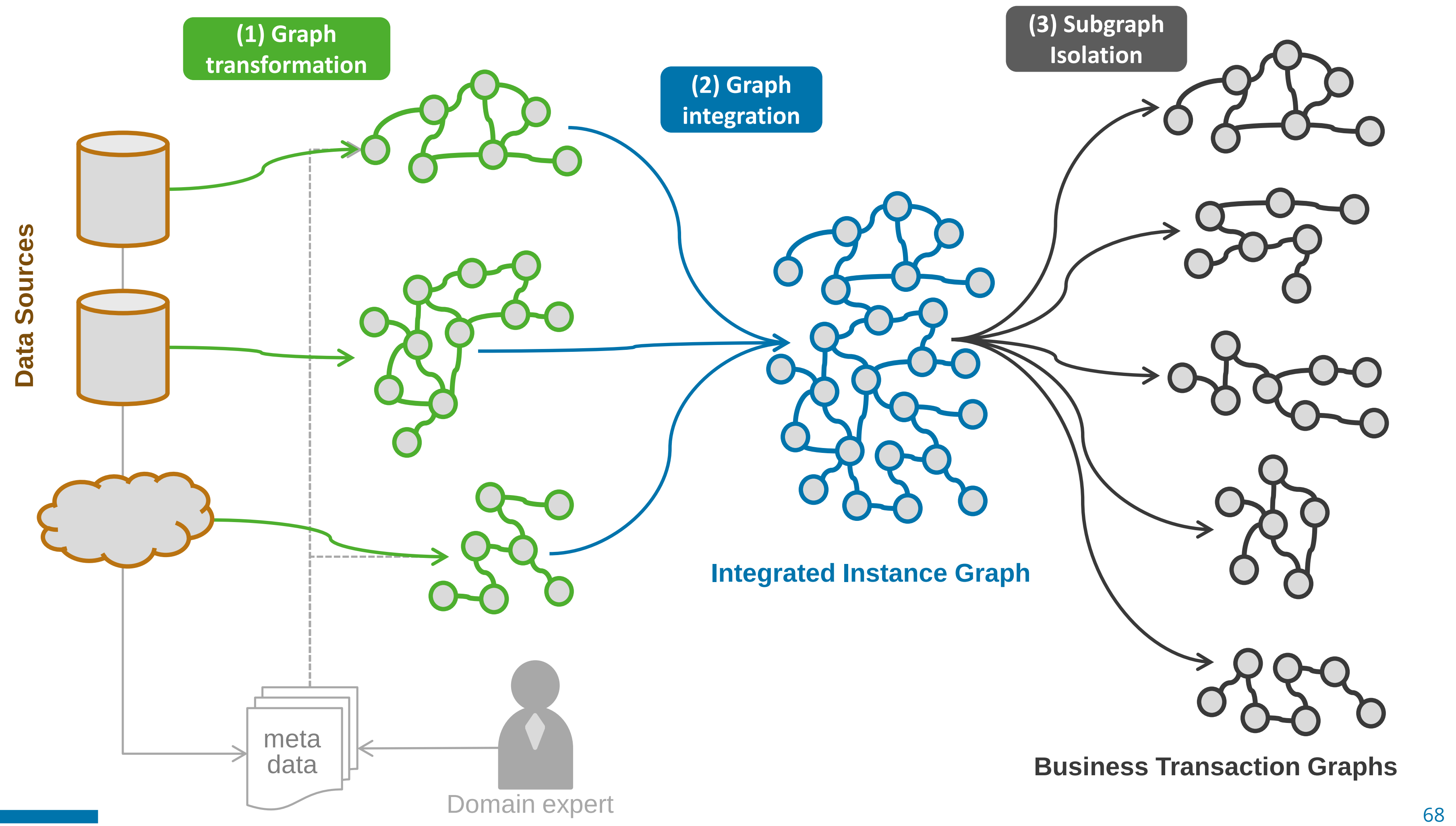
GRAPH-BASED DATA INTEGRATION

- need to integrate diverse data from different sources (or from data lake) into semantically expressive graph representation
  - for later graph analysis
  - for representing background knowledge (knowledge graphs)
- traditional tasks for data acquisition, data transformation, data cleaning, schema / entity matching, entity fusion, data enrichment / annotation
- most previous work for RDF data, but not for property graphs

ScaDS DRESDEN LEIPZIG

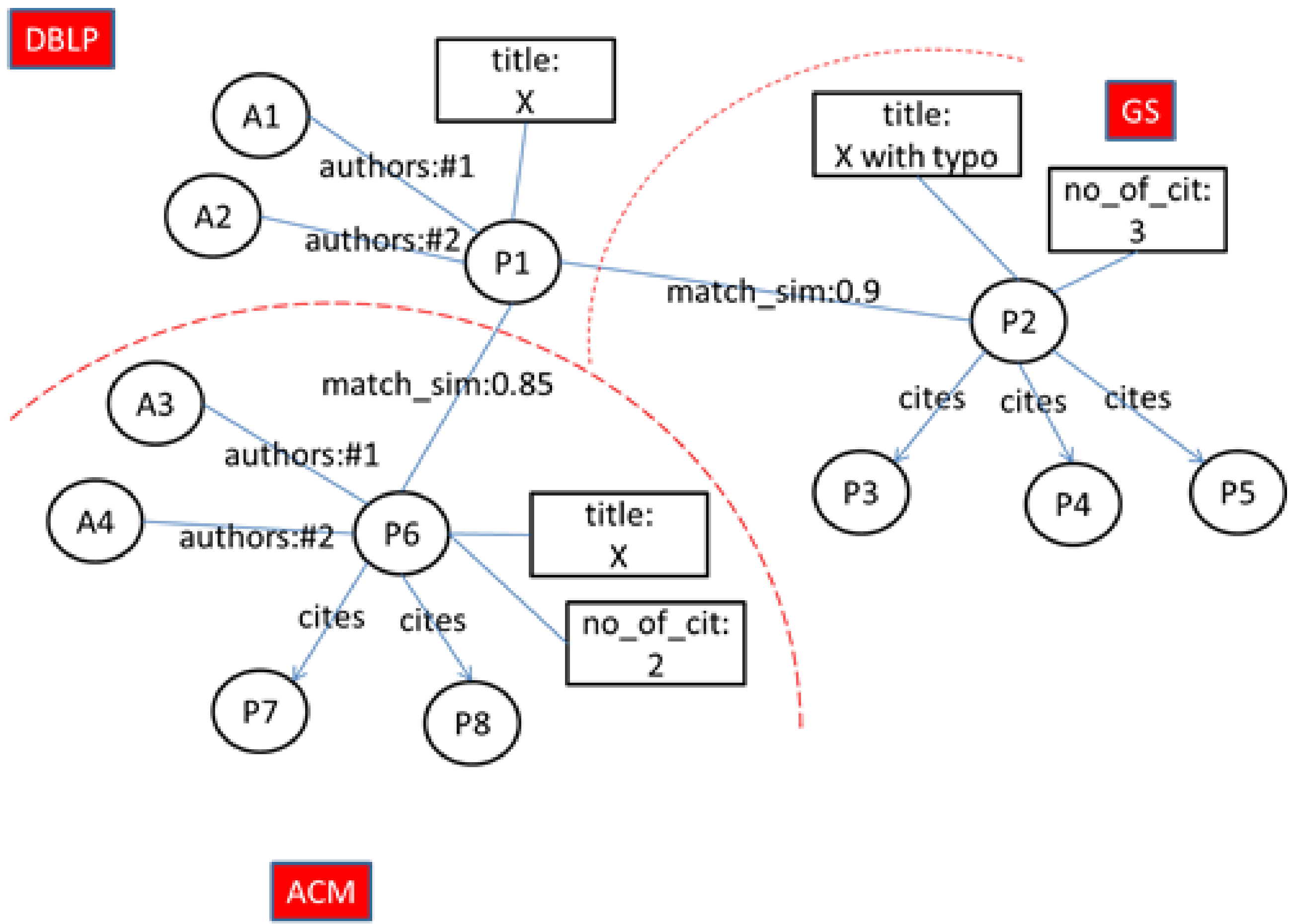
BIIIG DATA INTEGRATION WORKFLOW

„**B**usiness **I**ntelligence on **I**ntegrated **I**nstance **G**raphs (BIIIG)“ (PVLDB 2014)



ScaDS DRESDEN LEIPZIG

INTEGRATION SCENARIO



source: Andreas Thor

ScaDS DRESDEN LEIPZIG

DYNAMIC GRAPHS

- graphs like social networks, citation networks, road networks etc change over time
  - need to efficiently update/refresh analysis results (graph metrics, communities/clusters, ...)
  - streaming networks vs slowly evolving networks
  - fast stream analysis vs. analysis of series of graph snapshots
- many initial studies on specific aspects but no comprehensive system for analysis of dynamic graphs

ScaDS DRESDEN LEIPZIG

INTERACTIVE GRAPH ANALYTICS

- need to support both interactive graph queries / exploration + graph mining
- OLAP-like graph analysis functionality
  - Multi-level, multidimensional grouping and aggregation
  - need for extended (nested) graph model?
- visual analytics for big graphs
  - data reduction techniques for visualization (sampling, multi-level grouping, ...)

ScaDS DRESDEN LEIPZIG

AGENDA

- Motivation
  - graph data
  - requirements
- Graph data systems
  - graph database systems
  - distributed graph processing systems (Pregel, etc.)
  - distributed graph dataflow systems (GraphX, Gelly)
- Gradoop
  - architecture
  - Extended Property Graph Model (EPGM)
  - implementation and performance evaluation
- Open challenges

Thank you!

ScaDS DRESDEN LEIPZIG

REFERENCES

- **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**

Gradoop

- 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, 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, E. Rahm: *Distributed Grouping of Property Graphs with GRADOOP.* Proc. BTW, 2017
- A. Petermann; M. Junghanns: *Scalable Business Intelligence with Graph Collections*. it - Information Technology Special Issue: Big Data Analytics, 2016
- 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, E. Rahm: *DIMSpan - Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems.* arXiv 2017