UNIVERSITÄT LEIPZIG

ScaDS
DRESDEN LEIPZIG

SCALABLE AND PRIVACY-PRESERVING DATA INTEGRATION
- PART 3 -

ERHARD RAHM

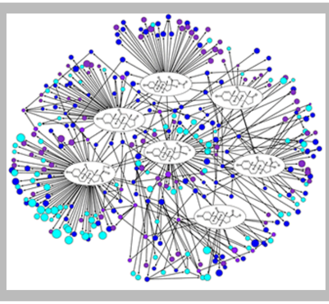
MARTIN JUNGHANNS, ANDRÉ PETERMANN

UNIVERSITY OF LEIPZIG

www.scads.de

AGENDA

- ScaDS Dresden/Leipzig
- Big Data Integration
  - Scalable entity resolution / link discovery
  - Large-scale schema/ontology matching
  - Holistic data integration
- Privacy-preserving record linkage
  - Encryption of sensitive information
  - PPRL with linkage unit
  - Secure multi-party approaches
- Graph-based data integration and analytics
  - Introduction
  - Graph-based data integration / business intelligence (BIIG)
  - Hadoop-based graph analytics (GRADOOP)

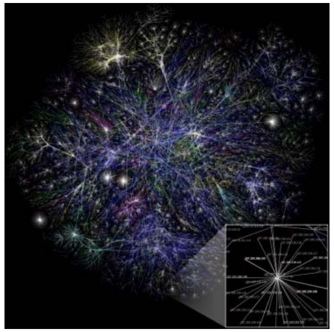


ScaDS DRESDEN LEIPZIG

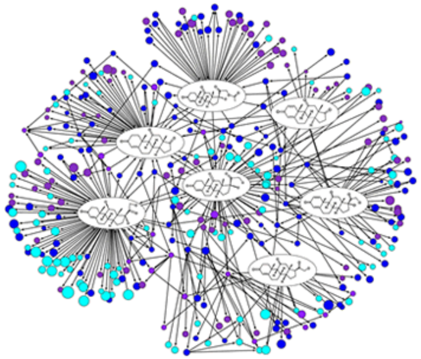
„GRAPHS ARE EVERYWHERE“

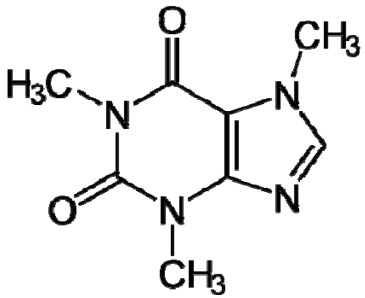
Social science

Engineering

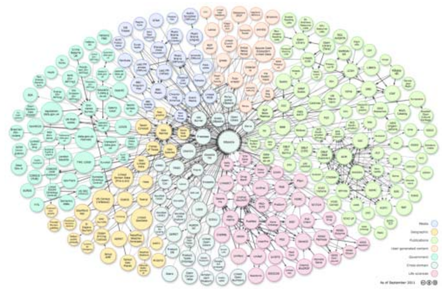


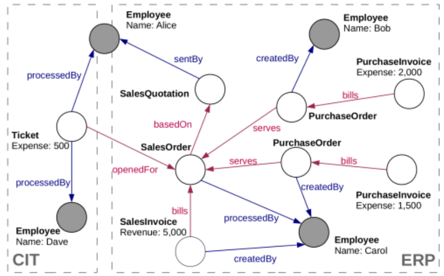
Life science





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

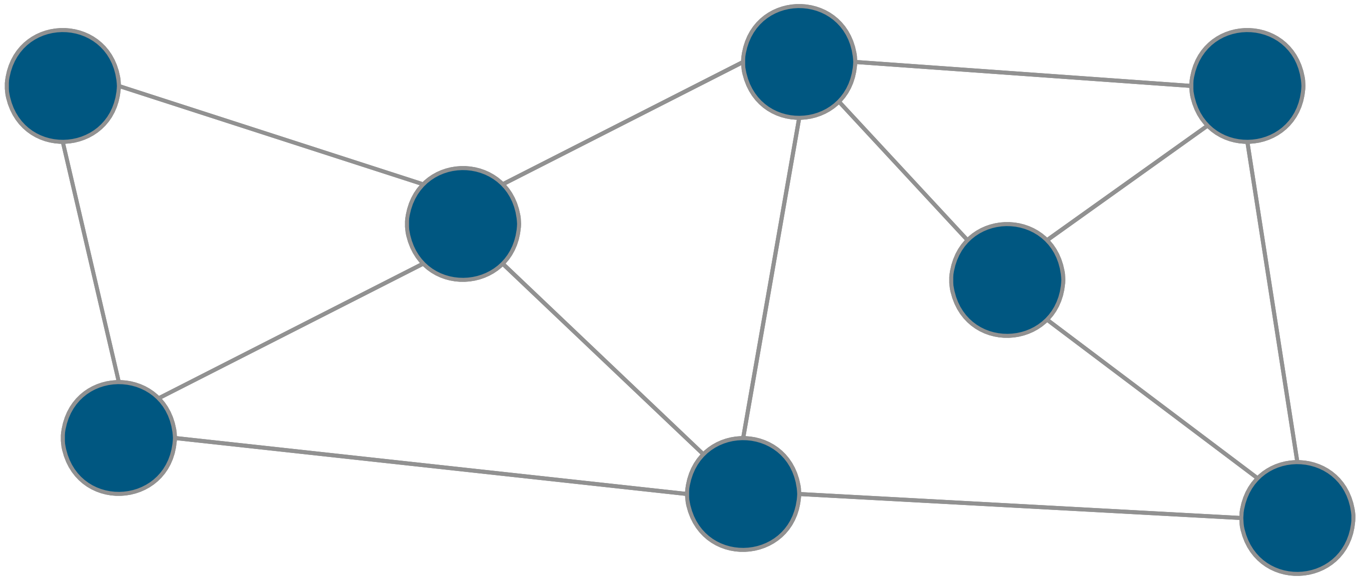
Patients
- > 18 millions (Germany)

Illnesses
- > 30.000

World Wide Web
- ca. 1 billion Websites

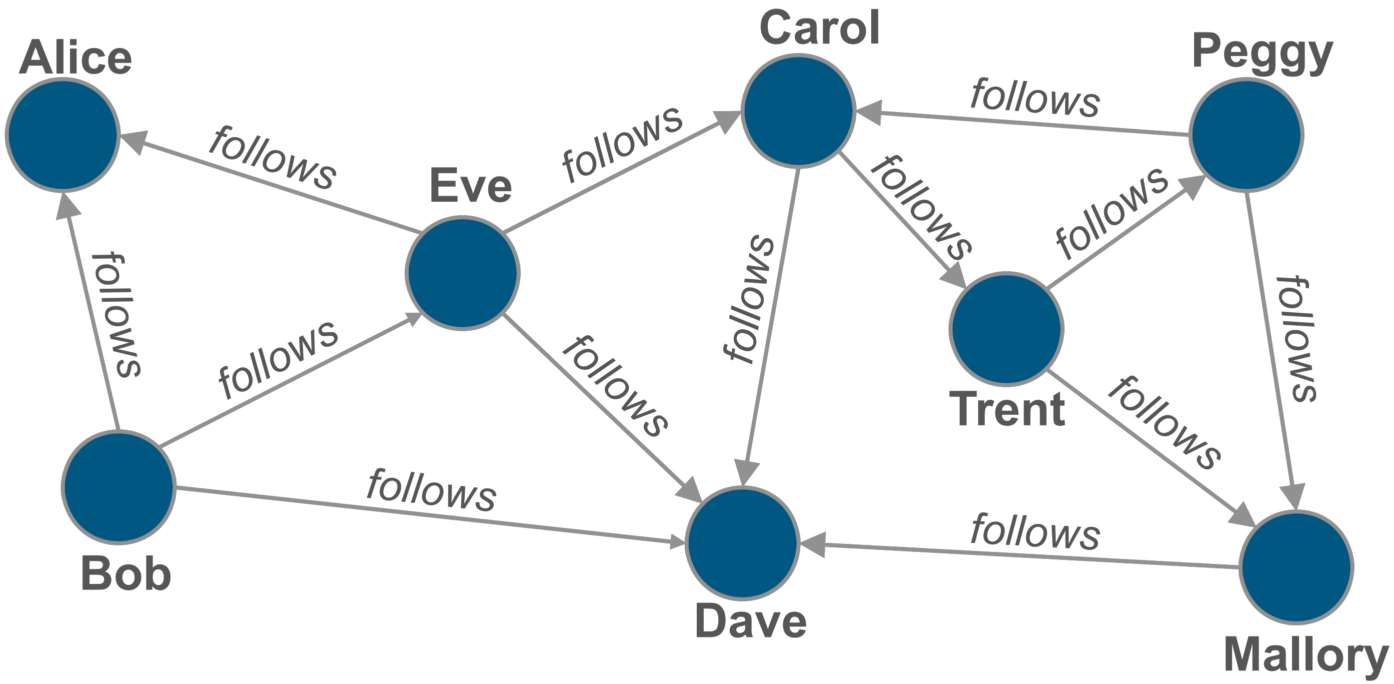
LOD-Cloud
- ca. 90 billion triples

“GRAPHS ARE EVERYWHERE”



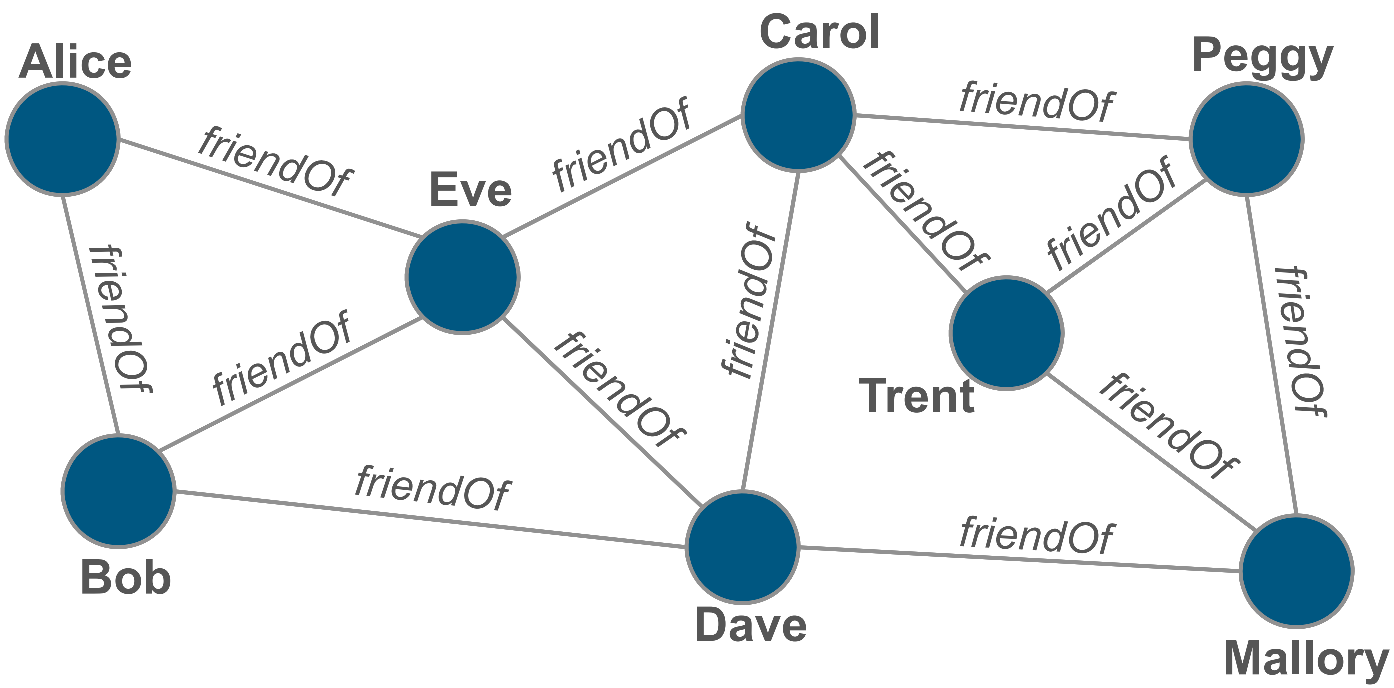
$$\boldsymbol{Graph} = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

"GRAPHS ARE EVERYWHERE"



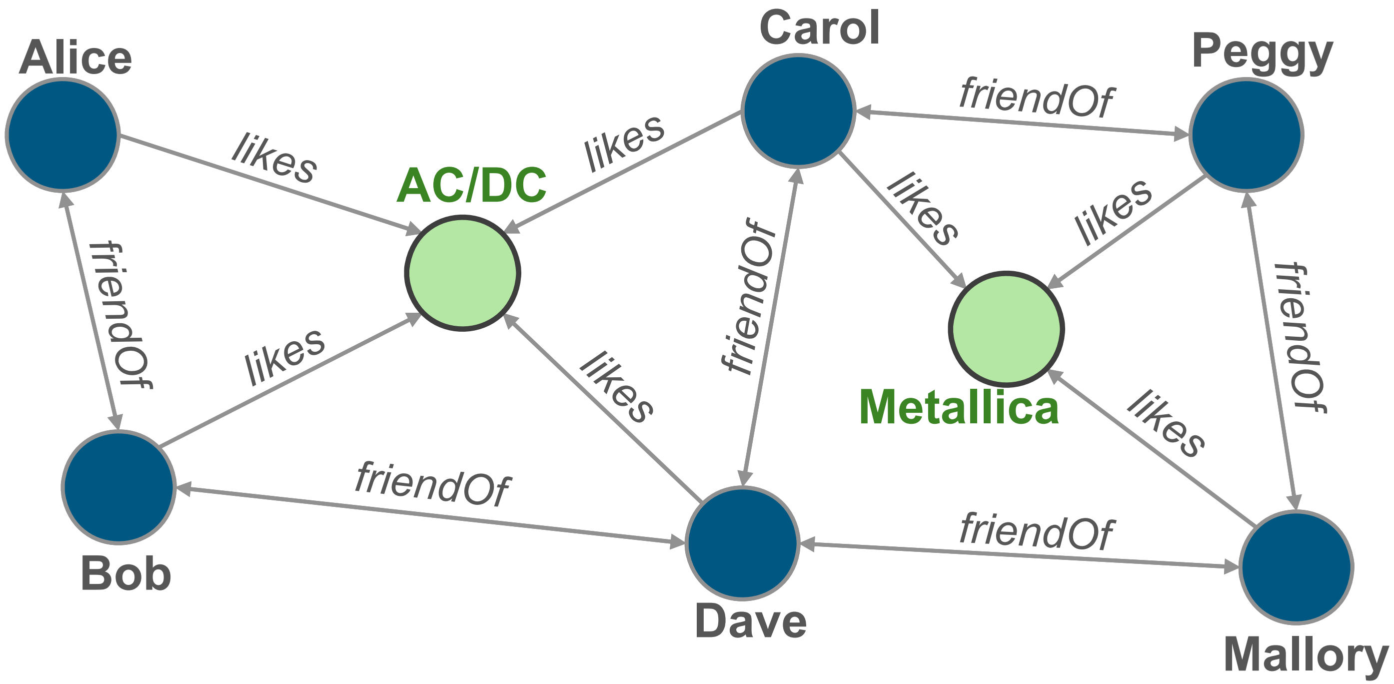
$$Graph = (\mathbf{Users}, Followers)$$

"GRAPHS ARE EVERYWHERE"



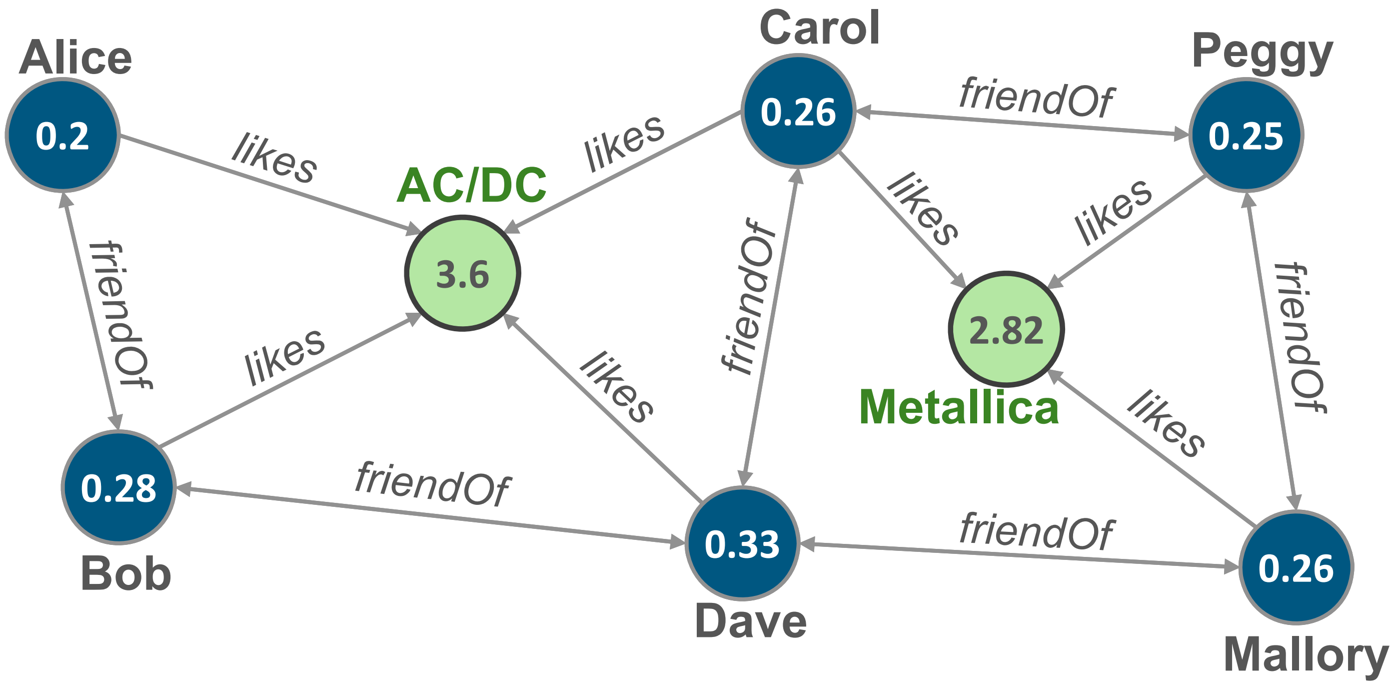
$$Graph = (\mathbf{Users}, Friendships)$$

“GRAPHS ARE HETEROGENEOUS”



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

"GRAPHS CAN BE ANALYZED"



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

ScaDS DRESDEN LEIPZIG

GRAPH DATA MODEL: REQUIREMENTS

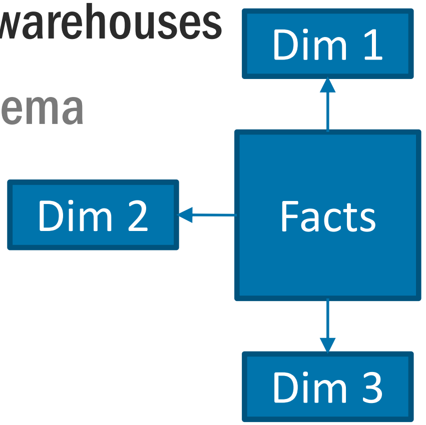
- Support for heterogeneous vertices and edges
- Good semantic expressiveness
  - Typed (labeled) vertices and edges
  - Properties for vertices and edges
  - Support for collections of graphs (not only 1 graph)
- Flexibility: (semi-) structured data without strict schema
- Analysis support
  - powerful graph operators / queries
- Easy usability

Property Graph Data Model (PGM) supports most requirements

ScaDS DRESDEN LEIPZIG

GRAPH-BASED DATA INTEGRATION

- Graphs are not only useful to represent and analyze existing networks / graph data
- Support easy linking / integration of existing data sources
  - utilized in LOD based on semantic web technology / RDF
  - can be utilized for other data models such as PGM
  - Enables graph-based analysis on relational databases and other data sources, e.g., for business intelligence
- Business intelligence usually based on relational data warehouses
  - enterprise data is integrated within dimensional schema
  - analysis limited to predefined relationships
  - no support for relationship-oriented data mining



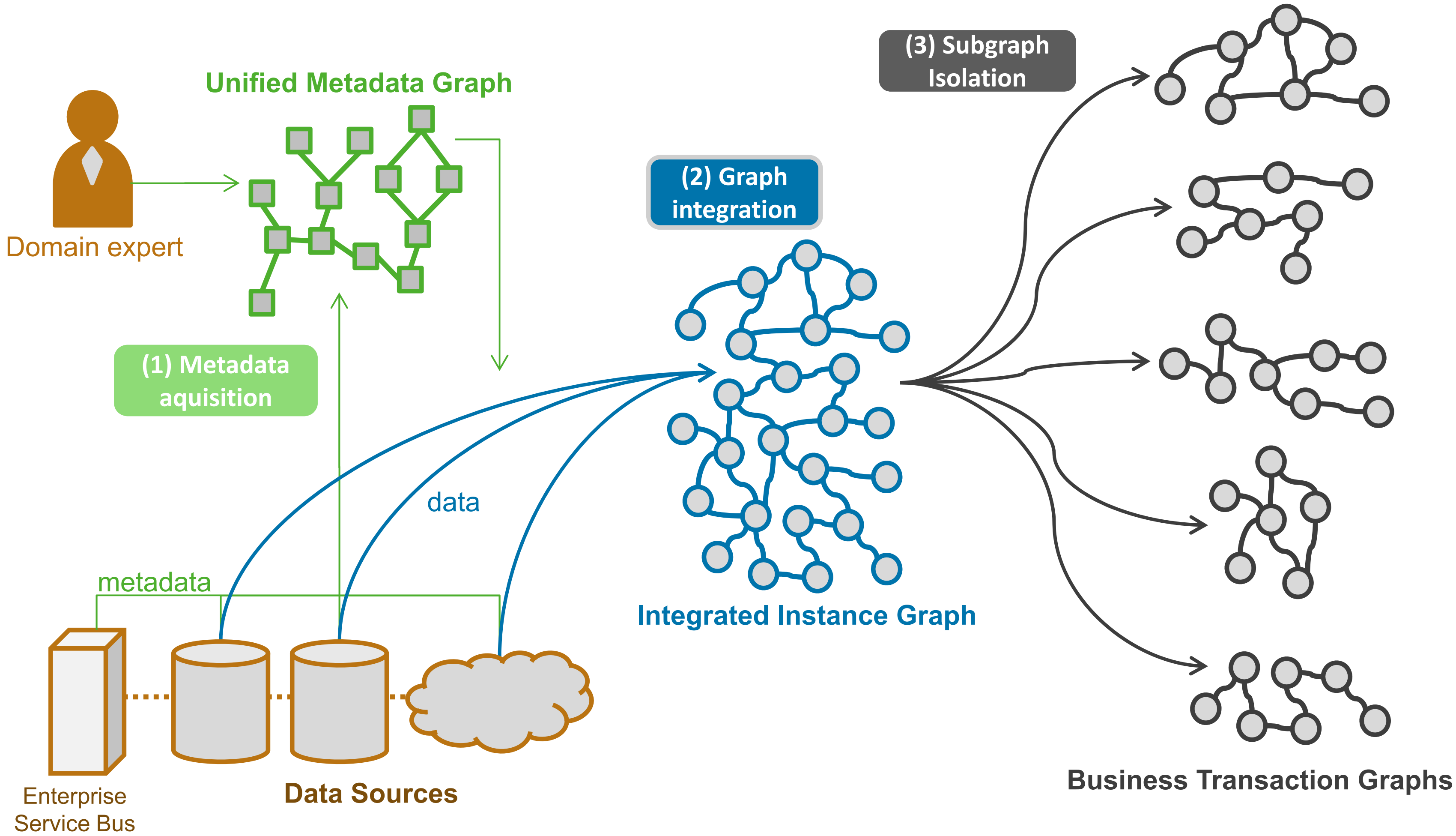
GRAPH-BASED BUSINESS INTELLIGENCE WITH BIIIG

- BIIIG: Business Intelligence on Integrated Instance Graphs
- Heterogeneous data sources are integrated within an instance graph by preserving original relationships between data objects
  - transactional and master data
- Largely automated extraction of metadata and instance data and transformation into graphs
  - fusion of matching entities and relations
- Extraction of subgraphs (business transaction graphs) related to interrelated business activities
- Analysis of graphs/subgraphs with aggregation queries, pattern mining etc.

ScaDS DRESDEN LEIPZIG

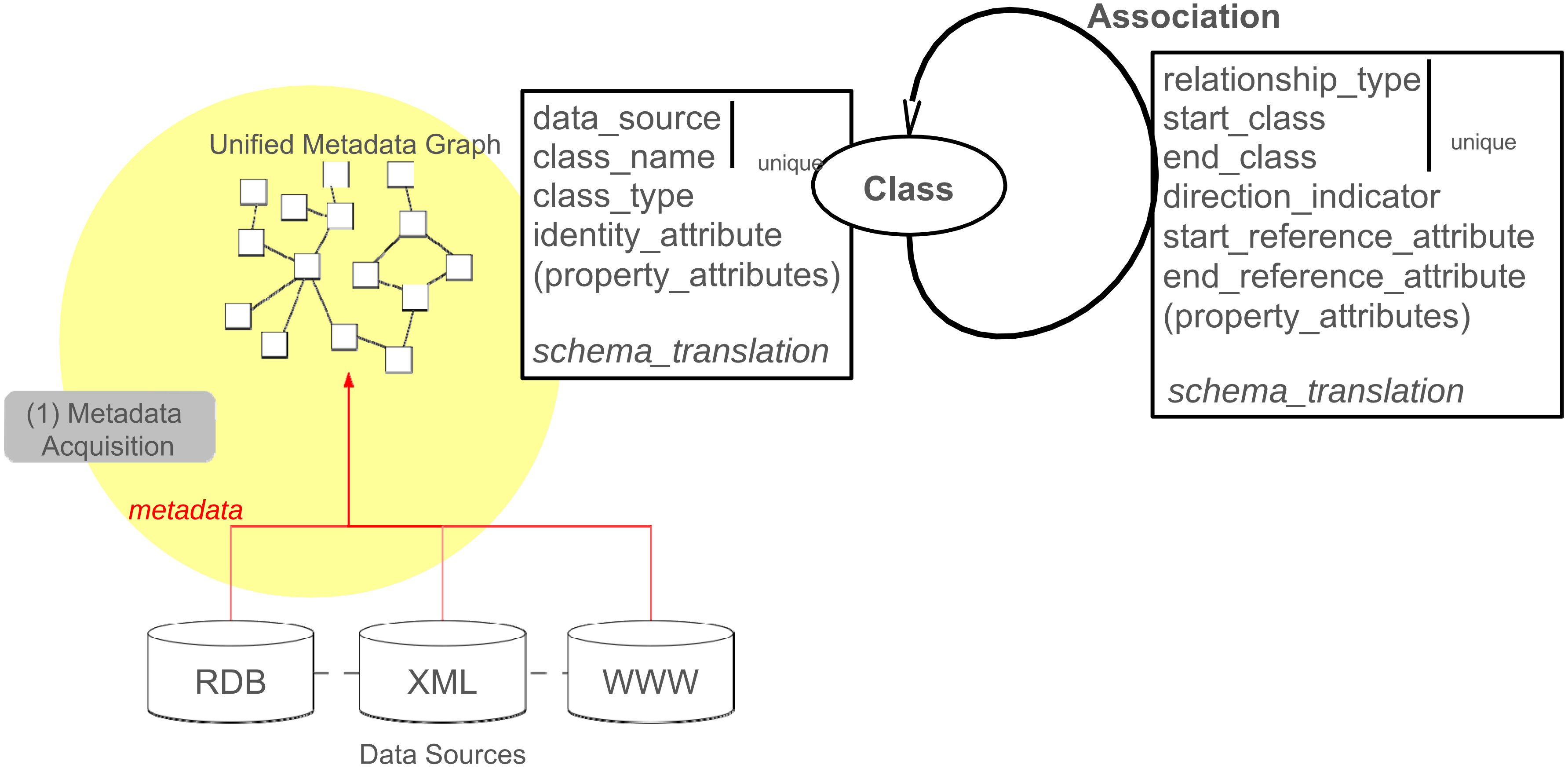
BIIG DATA INTEGRATION AND ANALYSIS WORKFLOW

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



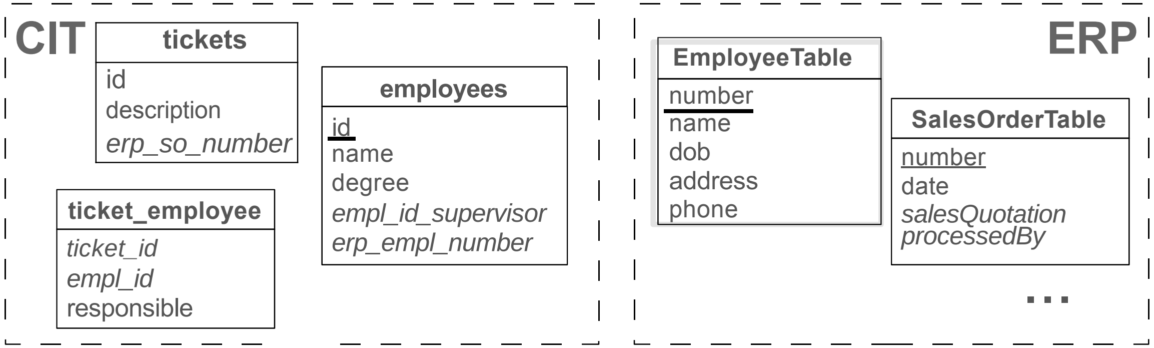
ScaDS DRESDEN LEIPZIG

METADATA REPRESENTATION WITH UMG



ScaDS DRESDEN LEIPZIG

METADATA EXTRACTION (2)



Schema translations:

Class **Employee**

```
SELECT number, name,
dob, address, phone
FROM CIT.employees
```

Class **Ticket**

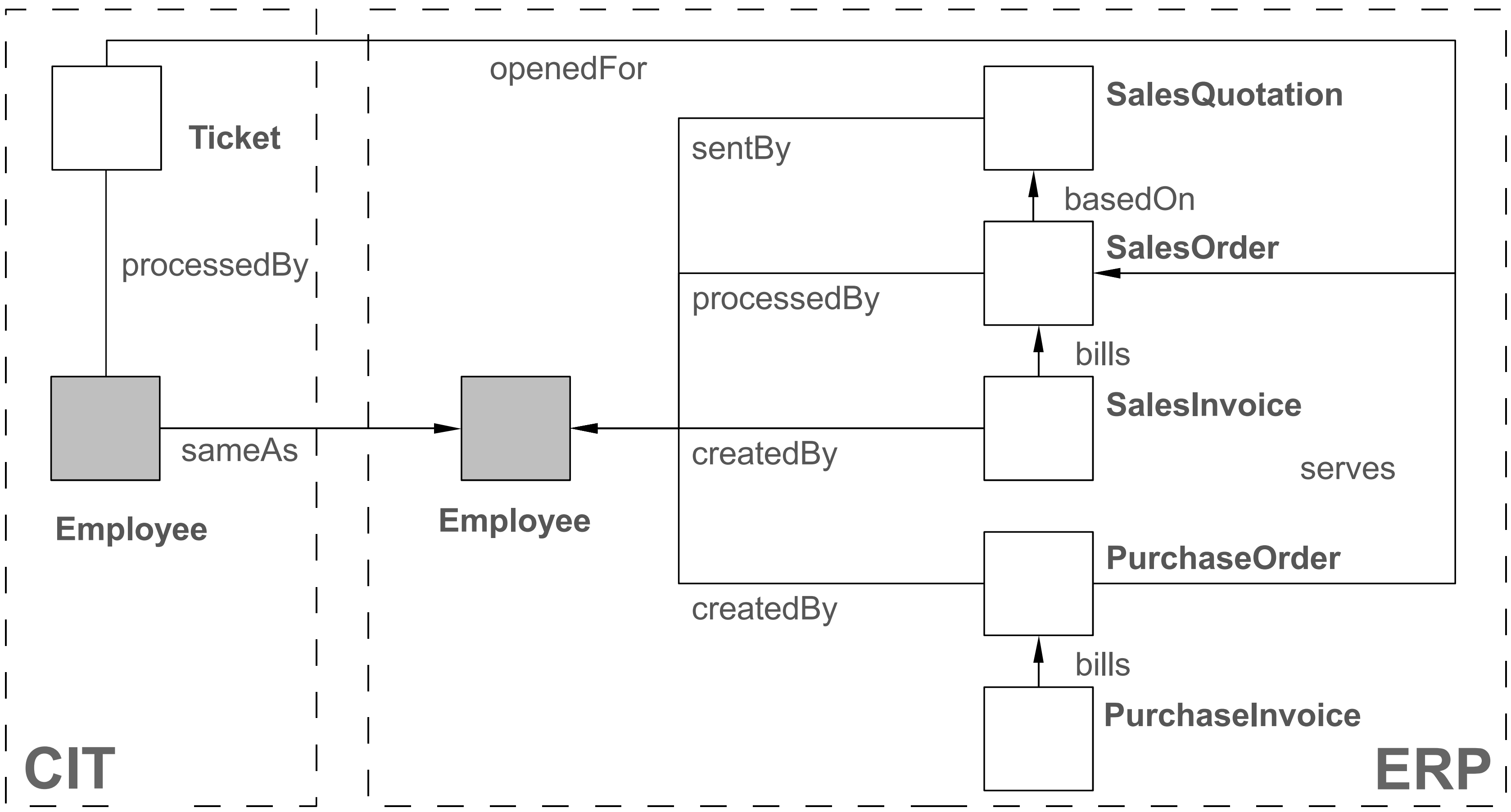
```
SELECT id, description
FROM CIT.tickets
```

Association **processedBy** (m:n)

```
SELECT ticket_id, empl_id
FROM CIT.ticket_employee
```

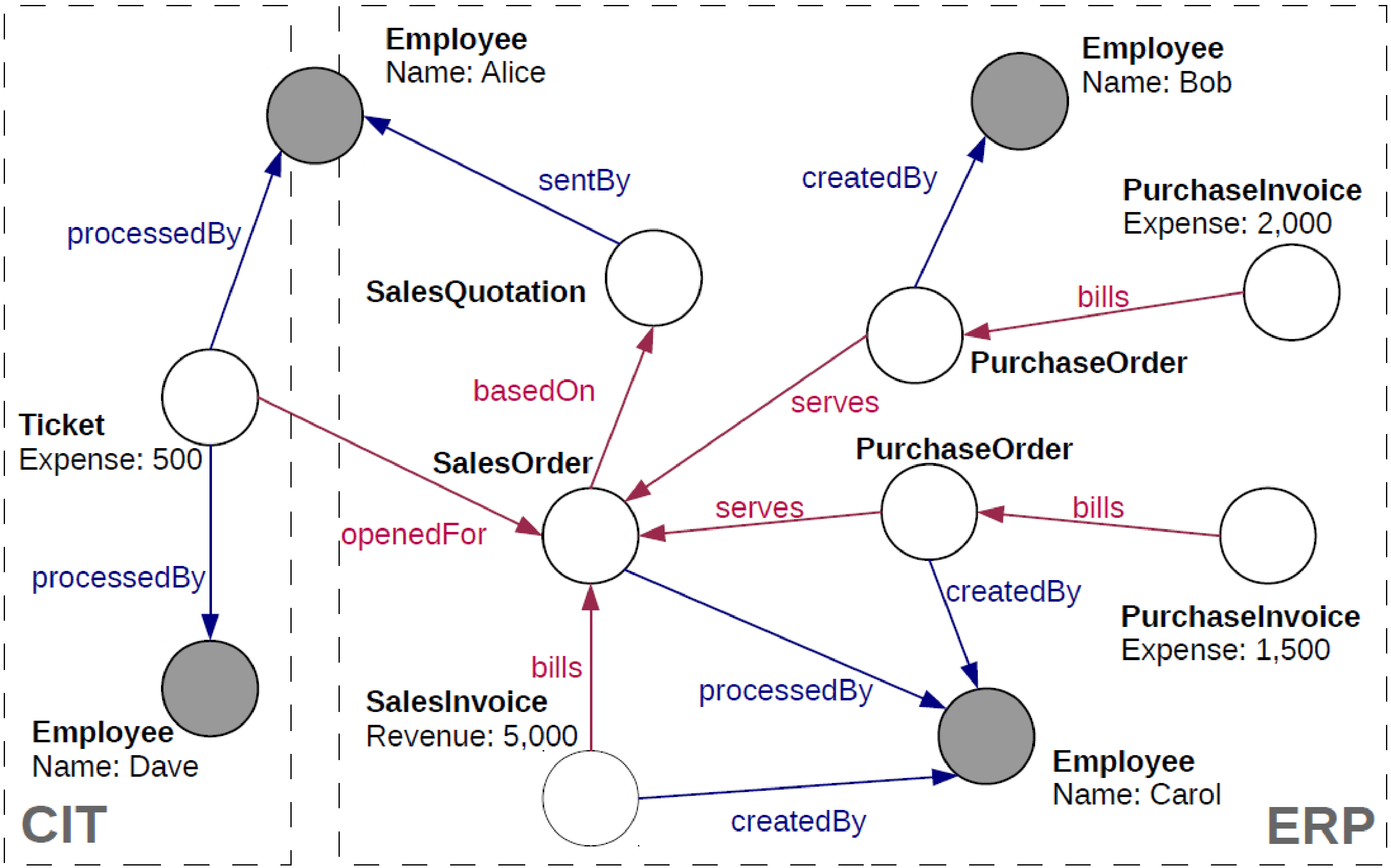
ScaDS DRESDEN LEIPZIG

RESULTING UMG (UNIFIED METADATA GRAPH)



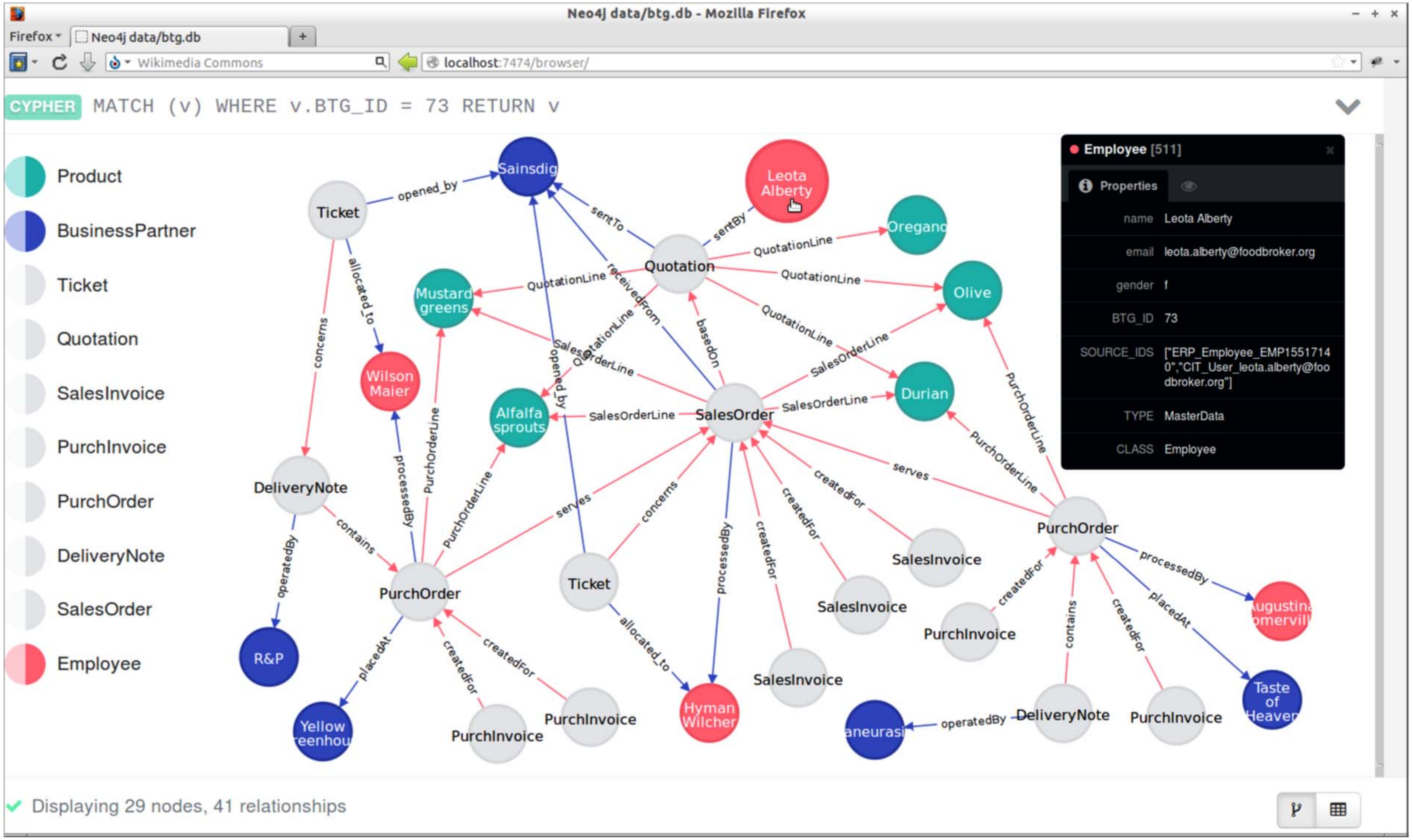
ScaDS DRESDEN LEIPZIG

SAMPLE INSTANCE GRAPH



ScaDS DRESDEN LEIPZIG

SCREENSHOT OF NEO4J IMPLEMENTATION



ScaDS DRESDEN LEIPZIG

GRAPH DATA MANAGEMENT

- Relational database systems
  - can be used to implement a graph store
  - SQL alone insufficient for graph processing (need for graph operators and graph mining)
- RDF data management
  - flexible management of semantic web data
  - Data integration support (linking of entities / concepts)
  - SPARQL query processing
  - insufficient scalability of triple stores
  - insufficient support for graph mining
- Graph database system, e.g. Neo4J
  - use of ***property graph data model***: vertices and edges have arbitrary set of properties (represented as key-value pairs)
  - focus on simple transactions and queries
  - insufficient scalability
  - insufficient support for graph mining

ScaDS DRESDEN LEIPZIG

GRAPH DATA MANAGEMENT (2)

- Parallel *graph processing* systems, e.g., Google Pregel, Apache Giraph
  - in-memory storage of graphs in shared nothing clusters
  - parallel processing of general graph algorithms, e.g., page rank, connected components, …
  - little support for semantically expressive graphs
  - no end-to-end approach with data integration and persistent graph storage
- Newer approaches (Apache Spark, Apache Flink):
  - analysis workflow with graph operators
  - no end-to-end solution with data integration and persistent graph storage

ScaDS DRESDEN LEIPZIG

COMPARISON

	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	transactional	analytic	analytic
Query Language	Yes	No	No
Persistency	Yes	No	No
Scalability	Vertical	Horizontal	Horizontal
Workflows	No	No	**Yes**
Data Integration	**No**	**No**	**No**
Graph Analytics	**No**	**Yes**	**Yes**
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.

ScaDS DRESDEN LEIPZIG

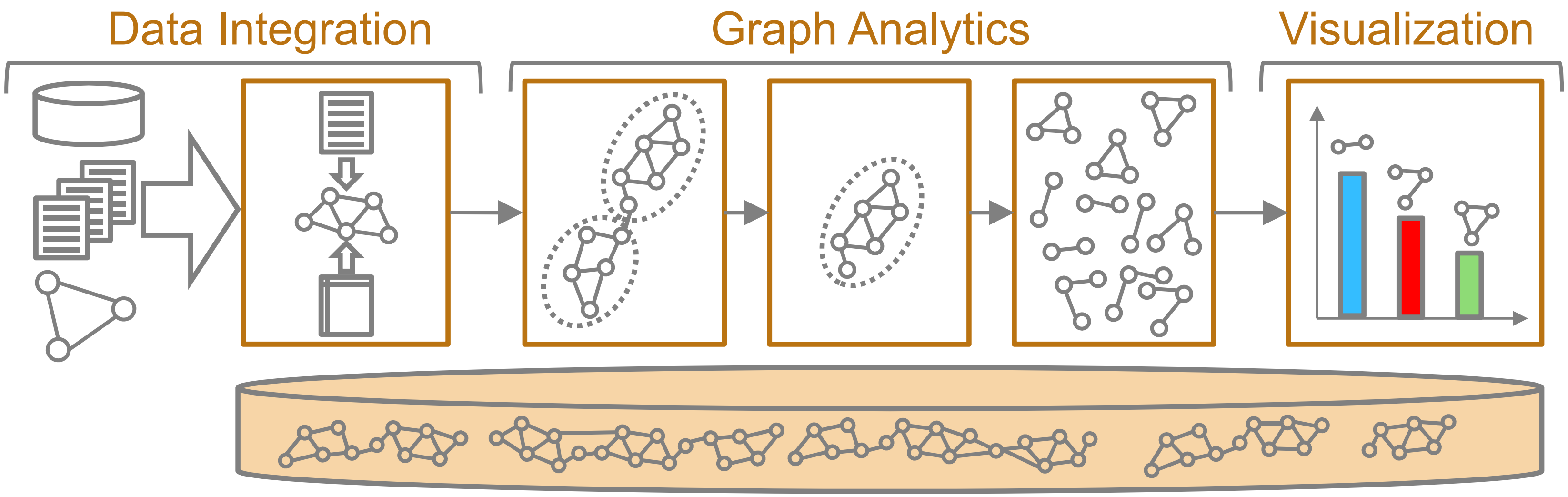
GRADOOP CHARACTERISTICS

- Hadoop-based framework for graph data management and analysis
- Graph storage in scalable distributed store, e.g., HBase
- Extended property graph data model (EPGM)
  - operators on graphs and sets of (sub) graphs
  - support for semantic graph queries and mining
- Leverages powerful components of Hadoop ecosystem
  - initially: Apache HBase, MapReduce, Apache Giraph
  - now: Apache HBase, Apache Flink, ...
- New functionality for graph-based processing workflows and graph mining

www.gradoop.org

ScaDS DRESDEN LEIPZIG

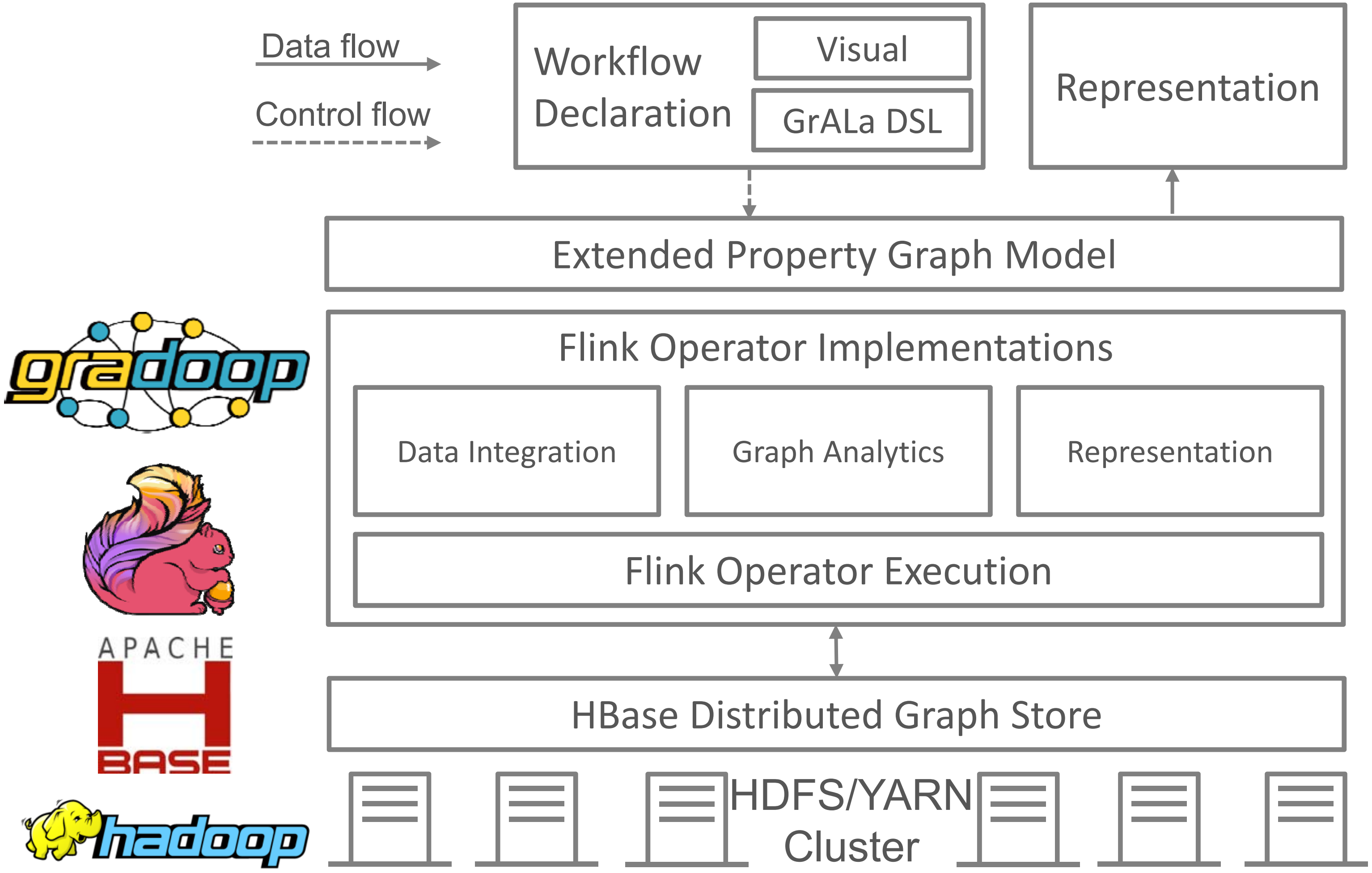
END-TO-END GRAPH ANALYTICS



- **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



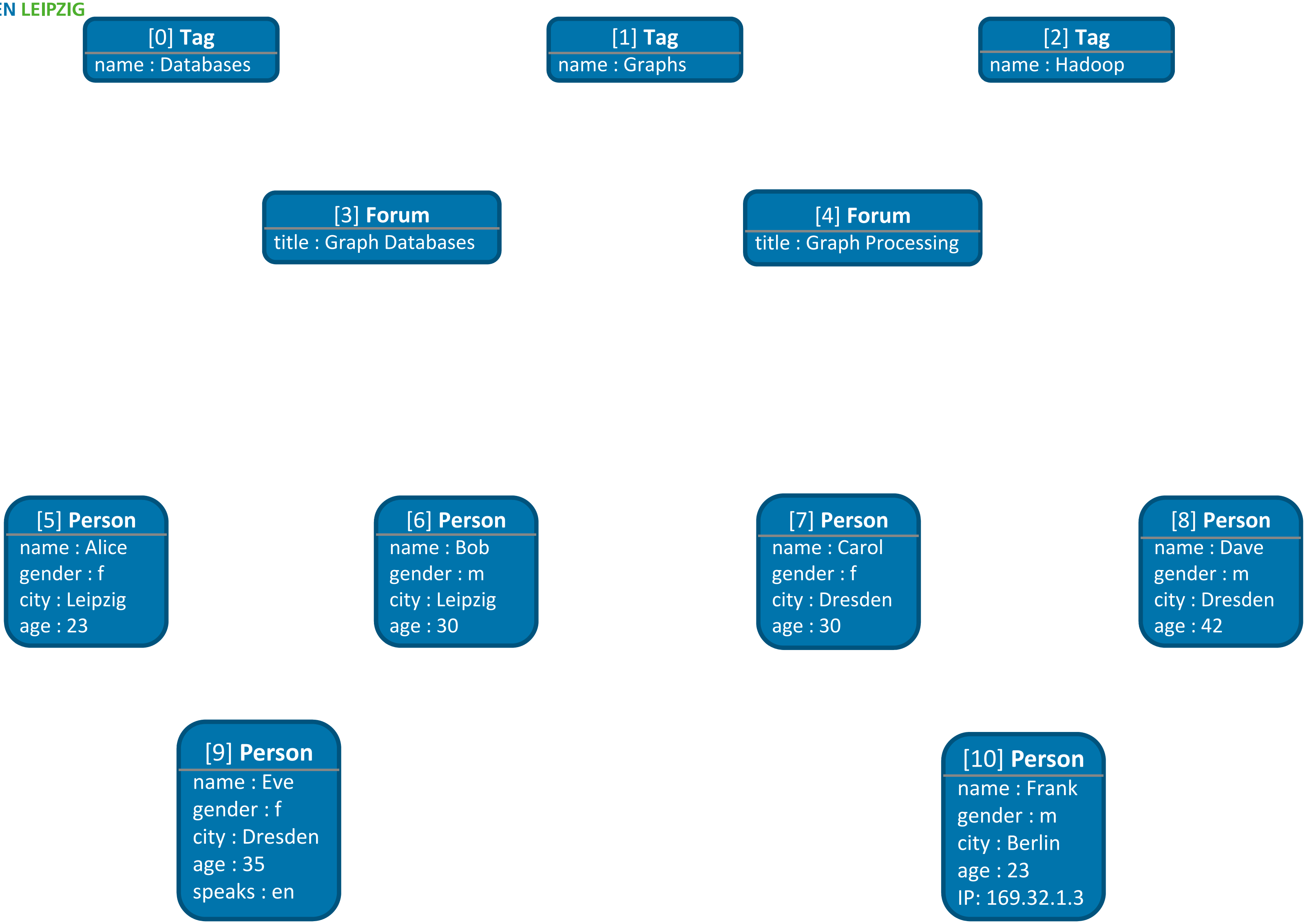
EXTENDED PROPERTY GRAPH MODEL (EPGM)

- 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
- Operators on both graphs and graph collections

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

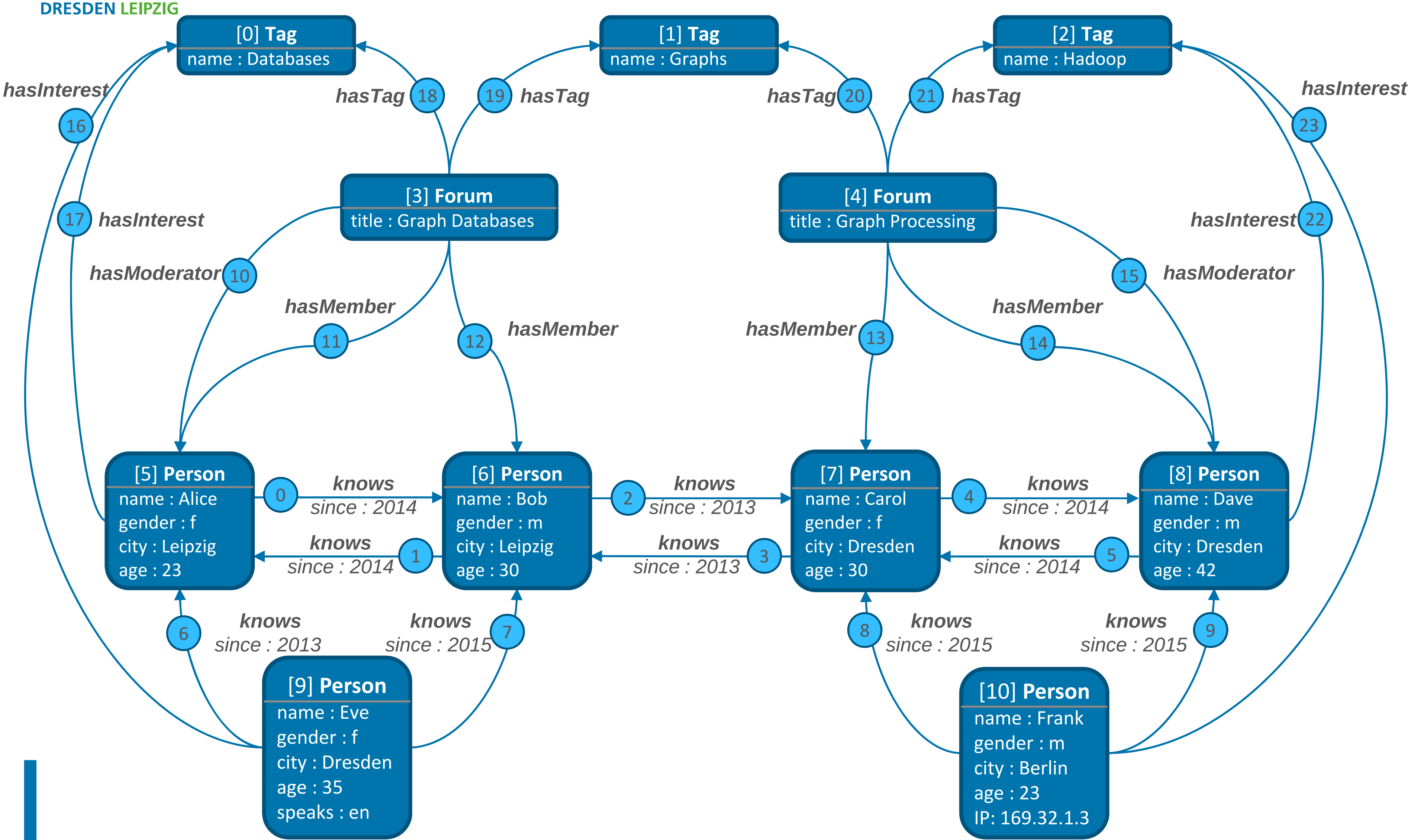
EPGM – GRAPH REPRESENTATION



UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

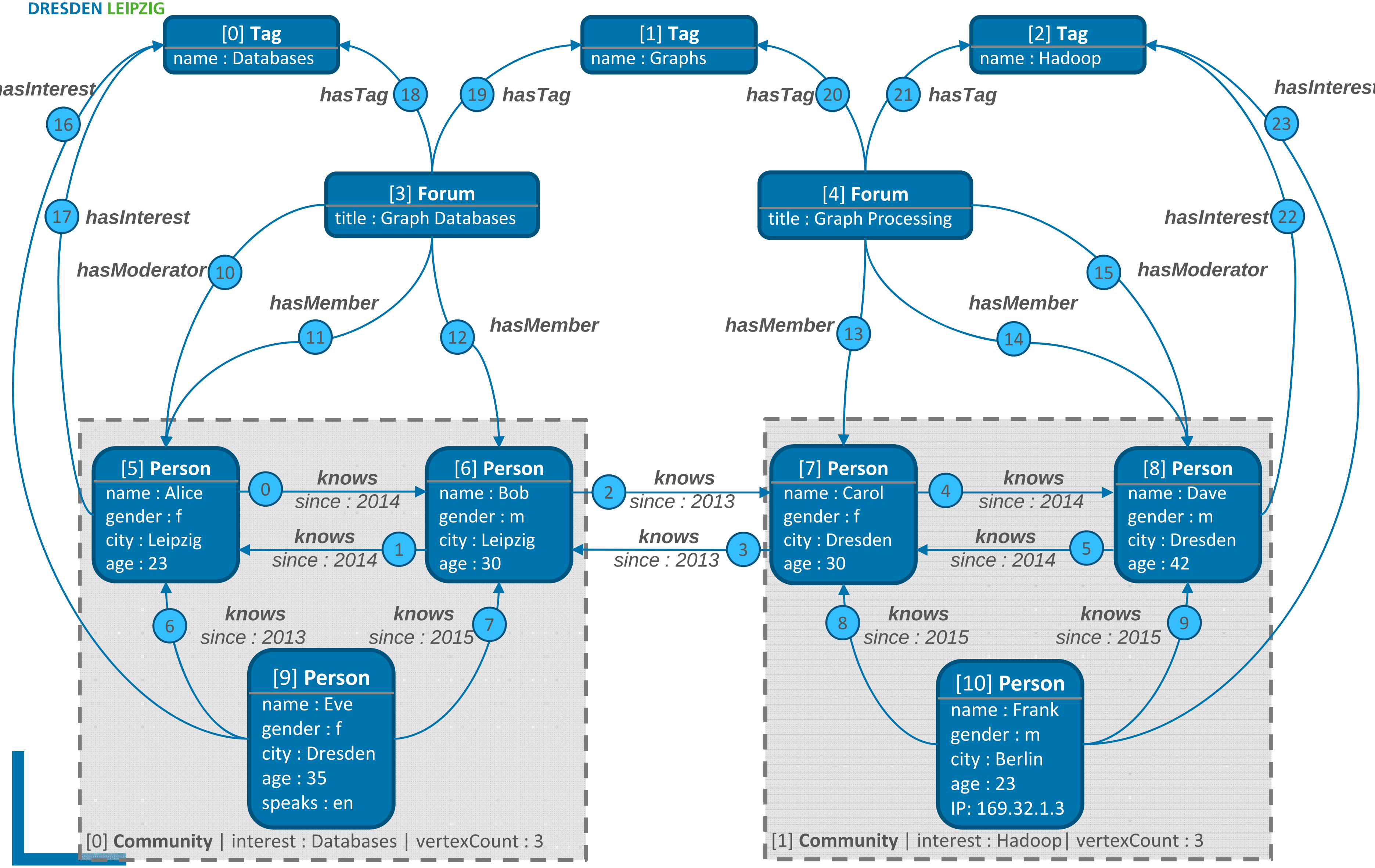
EPGM – GRAPH REPRESENTATION



UNIVERSITÄT LEIPZIG

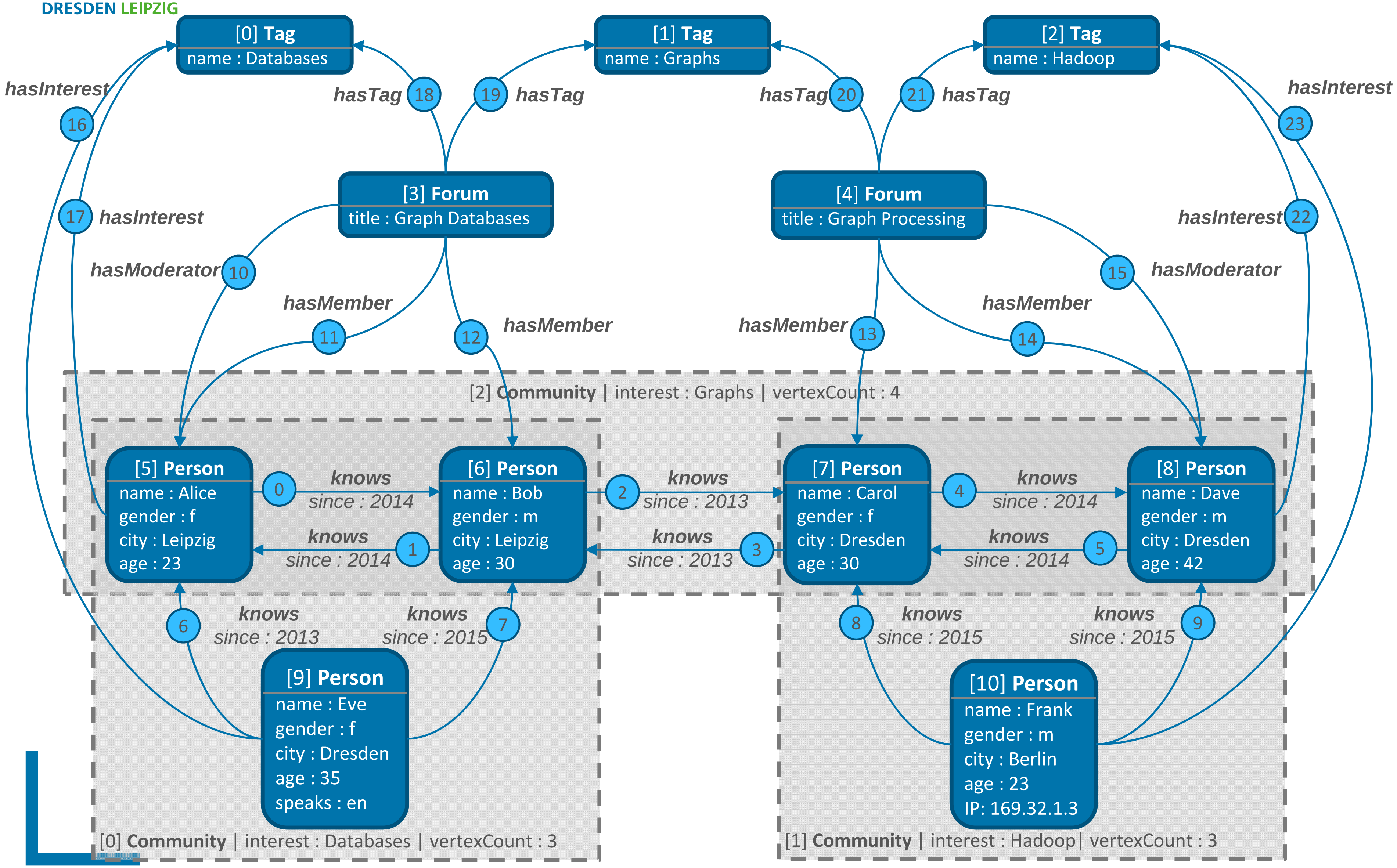
ScaDS DRESDEN LEIPZIG

EPGM – GRAPH REPRESENTATION



ScaDS DRESDEN LEIPZIG

EPGM – GRAPH REPRESENTATION



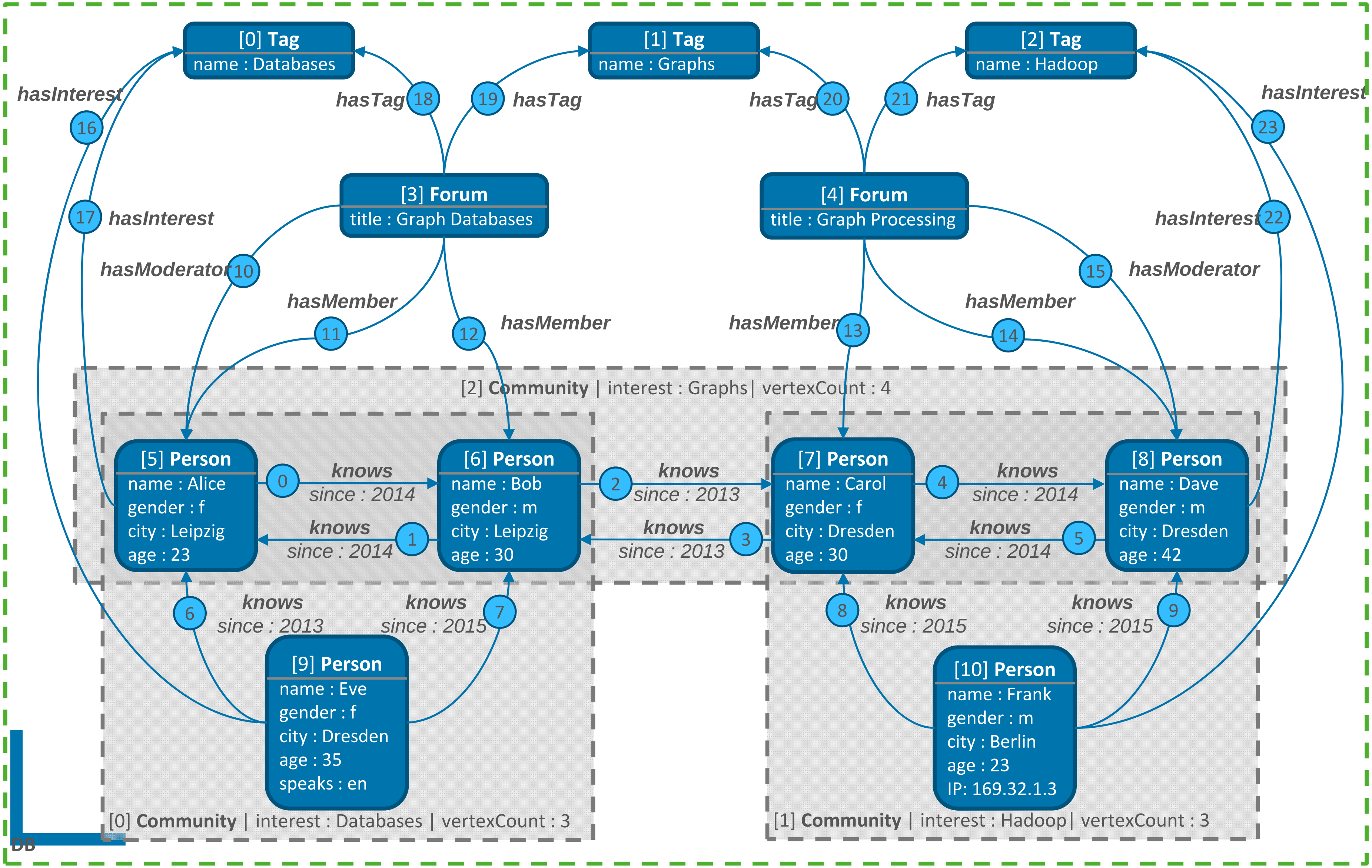
EPGM – OPERATORS AND ALGORITHMS

	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

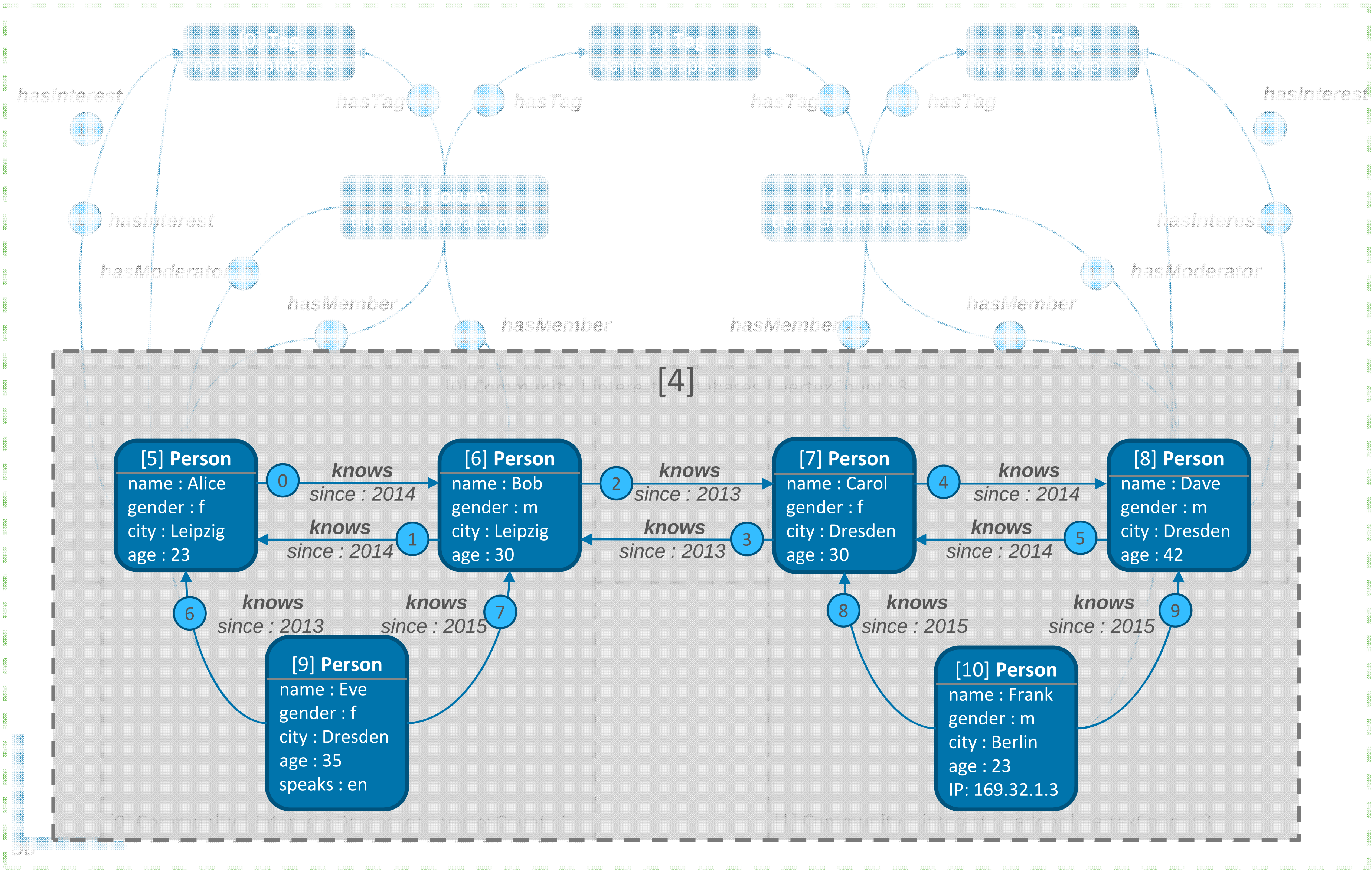
COMBINATION

```
1: personGraph =
db.G[0].combine(db.G[1]).combine(db.G[2])
```



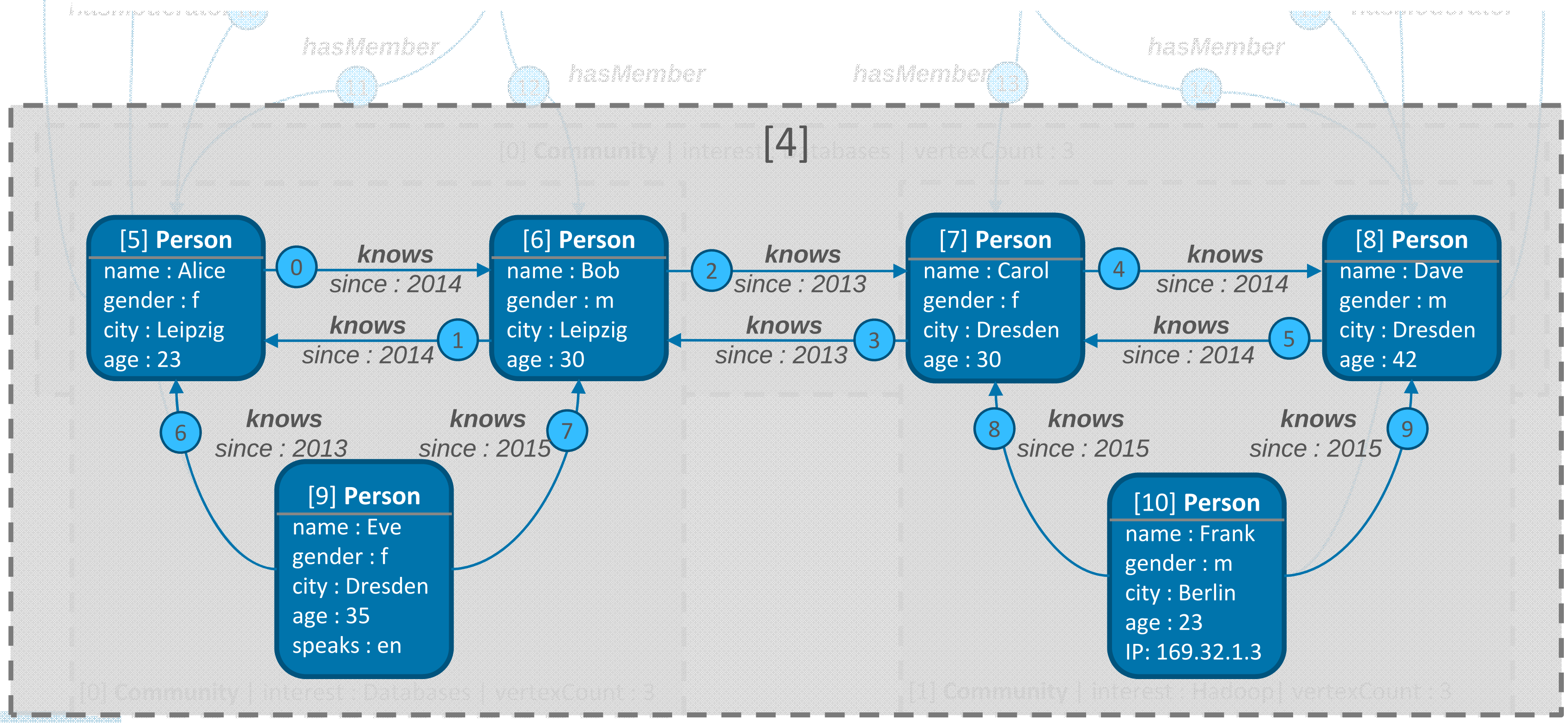
COMBINATION

```
1: personGraph =
db.G[0].combine(db.G[1]).combine(db.G[2])
```



GROUPING (GRAPH SUMMARIZATION)

```
personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
vertexGroupingKeys = [:label, “city”]
edgeGroupingKeys = [:label]
vertexAggFunc = (superVertex, vertices => superVertex[“count”] = |vertices|)
edgeAggFunc = (superEdge, edges => superEdge[“count”] = |edges|)
sumGraph = personGraph.groupBy(vertexGroupingKeys, vertexAggFunc,
  edgeGroupingKeys, edgeAggFunc)
```

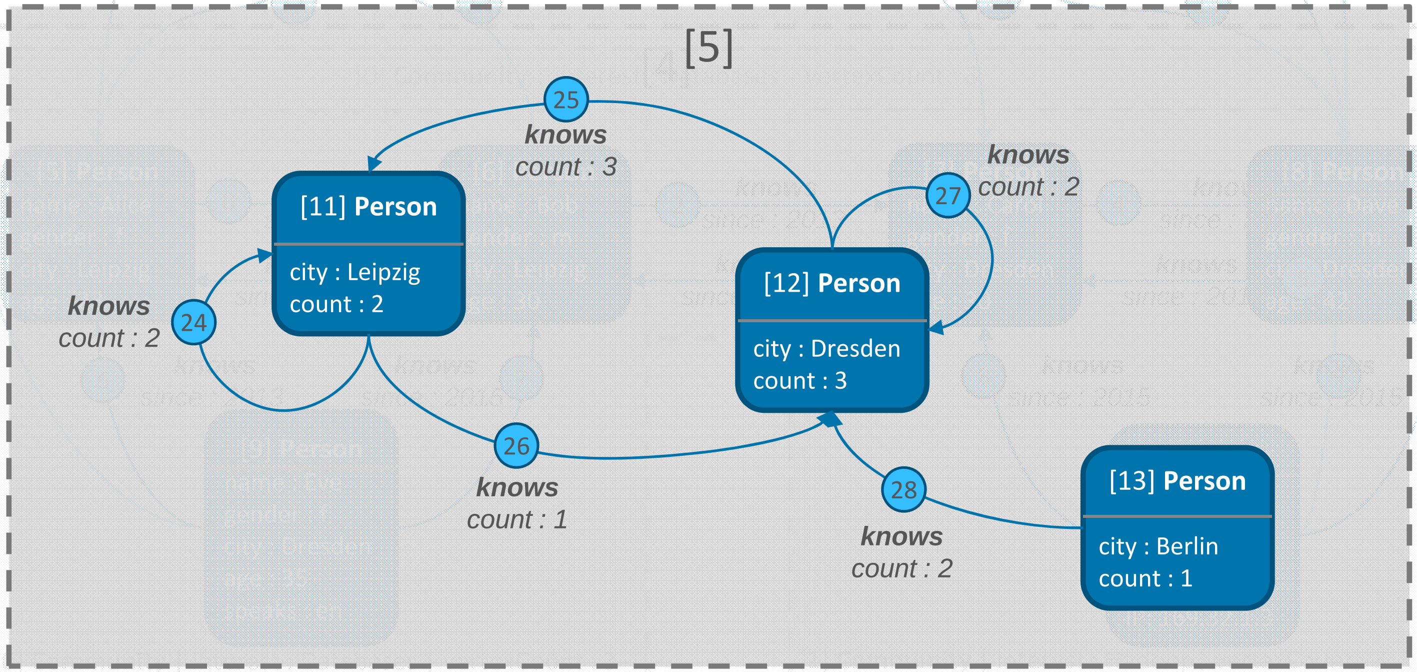


UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

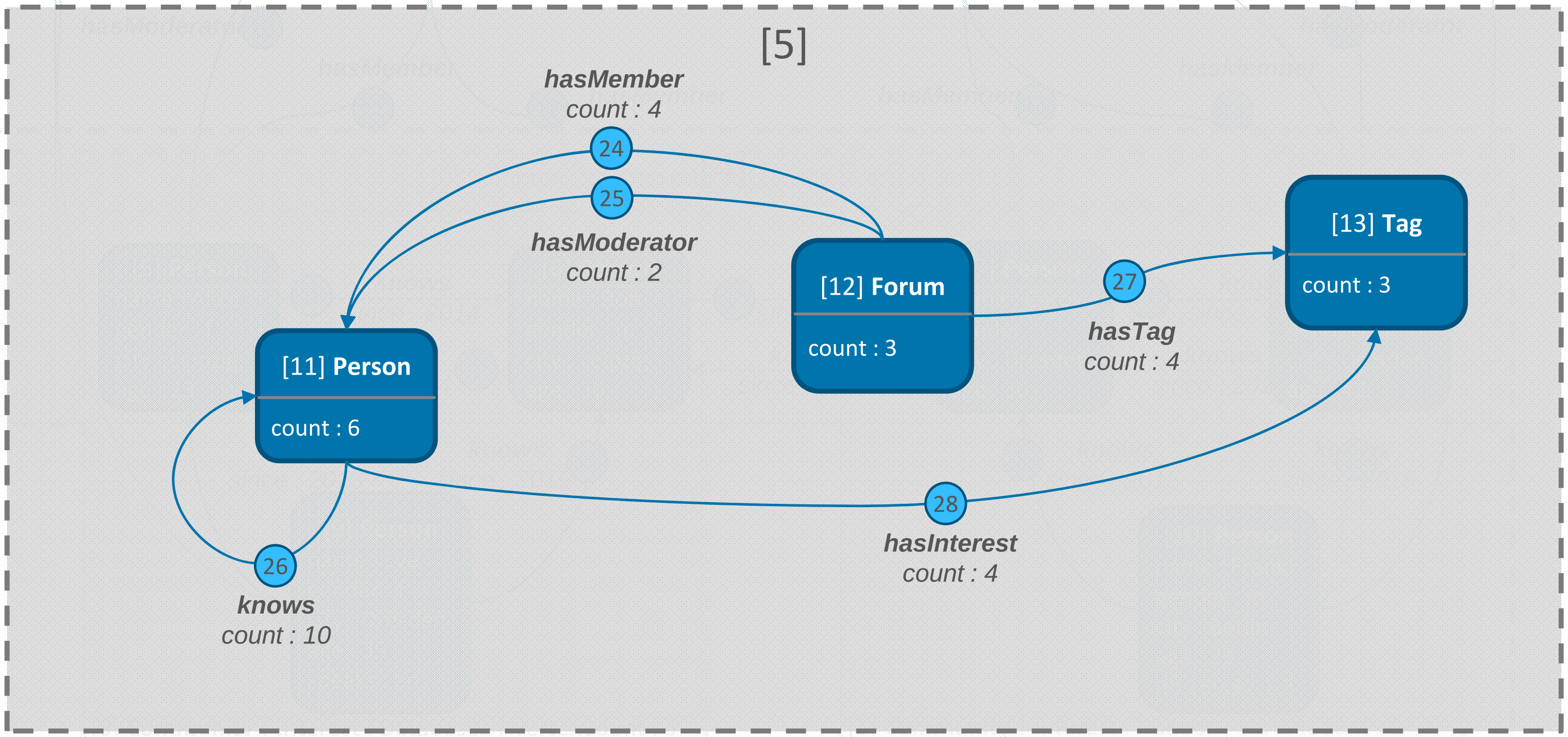
GROUPING (2)

```
1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = [:label, “city”]
3: edgeGroupingKeys = [:label]
4: vertexAggFunc = (superVertex, vertices => superVertex[“count”] = |vertices|)
5: edgeAggFunc = (superEdge, edges => superEdge[“count”] = |edges|)
6: sumGraph = personGraph.groupBy(vertexGroupingKeys, vertexAggFunc,
     edgeGroupingKeys, edgeAggFunc)
```



GROUPING: TYPE LEVEL *(SCHEMA GRAPH)*

```
1: vertexGroupingKeys = [:label]
2: edgeGroupingKeys = [:label]
3: vertexAggFunc = (superVertex, vertices => superVertex["count"] = |vertices|)
4: edgeAggFunc = (superEdge, edges => superEdge["count"] = |edges|)
5: sumGraph = databaseGraph.groupBy(vertexGroupingKeys, vertexAggFunc,
     edgeGroupingKeys, edgeAggFunc)
```

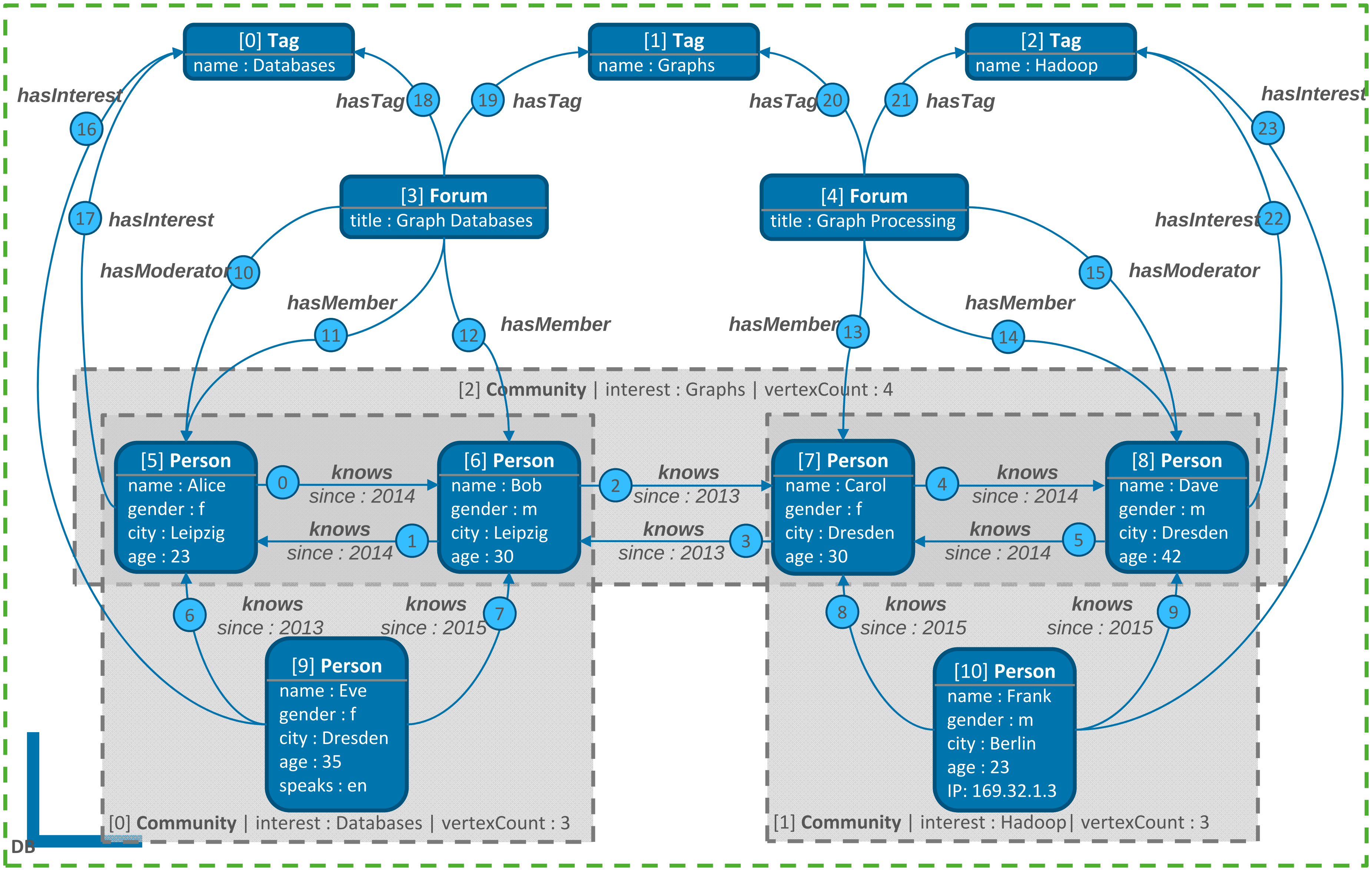


UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

SELECTION

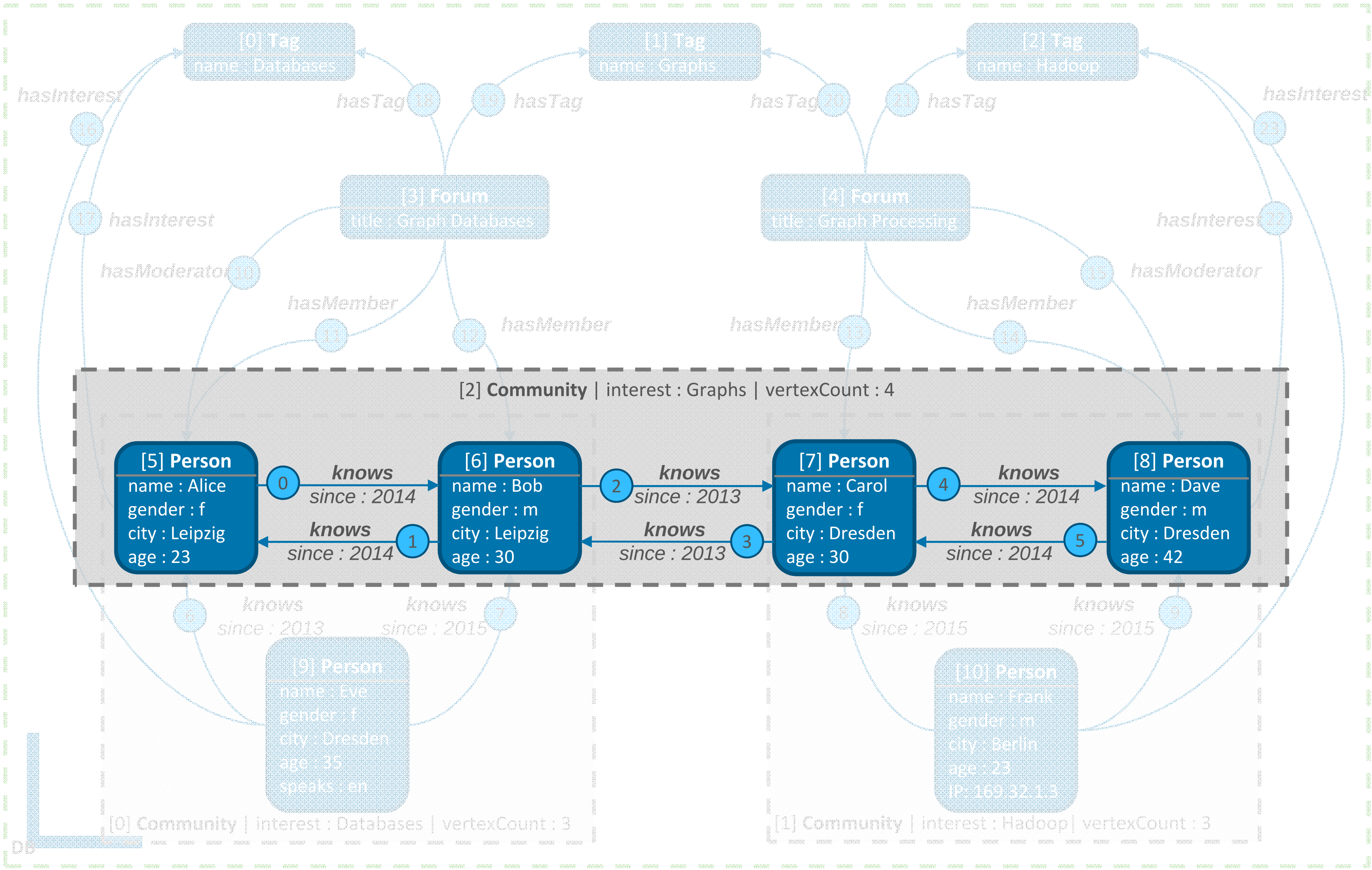
```
1: resultColl =
db.G[0,1,2].select((g => g[“vertexCount”] > 3))
```



ScaDS DRESDEN LEIPZIG

SELECTION

```
1: resultColl =
db.G[0,1,2].select((g => g["vertexCount"] > 3))
```



ScaDS DRESDEN LEIPZIG

DISTRIBUTED GRAPH STORE

1. Large-scale graphs
   - Support for real-world graphs with millions of vertices and billions of edges
2. Graph partitioning
   - Efficient data distribution to balance load and minimize communication during computation
3. Data versioning
   - Enable time-based graph analytics on properties and graph structure
4. Fault tolerance
   - Prevent data loss in case of cluster failures

DISTRIBUTED GRAPH STORE – HBASE

- Open Source implementation of Google BigTable
- **Distributed**, persistent, **sparse**, **multidimensional** sorted map based on HDFS
- Data distribution based on row key (i.e., horizontal **partitioning**)
- **Flexible** storage layout (handles only `byte[]`, **no types, no schema**)
- **Fault tolerancy** through data replication (HDFS)
- **Data versioning** on cell level

HTable

sorted

<table>
<tr><td rowspan="3">row key 1</td><td colspan="2">Column family 1</td><td colspan="2">Column family 2</td></tr>
<tr><td colspan="2">Column identifier</td><td>C. identifier</td><td>C. identifier</td></tr>
<tr><td colspan="2">versioned value</td><td>v. value</td><td>v. value</td></tr>
<tr><td rowspan="3">row key 2</td><td colspan="2">Colum family 1</td><td colspan="2">Colum family 2</td></tr>
<tr><td>C. Identifier</td><td>C. identifier</td><td colspan="2">Column identifier</td></tr>
<tr><td>v. value</td><td>v. value</td><td colspan="2">versioned value</td></tr>
</table>

Cell: <rowkey>.<column_family>.<column_identifier>[.<version>]

ScaDS DRESDEN LEIPZIG

VERTEX TABLE

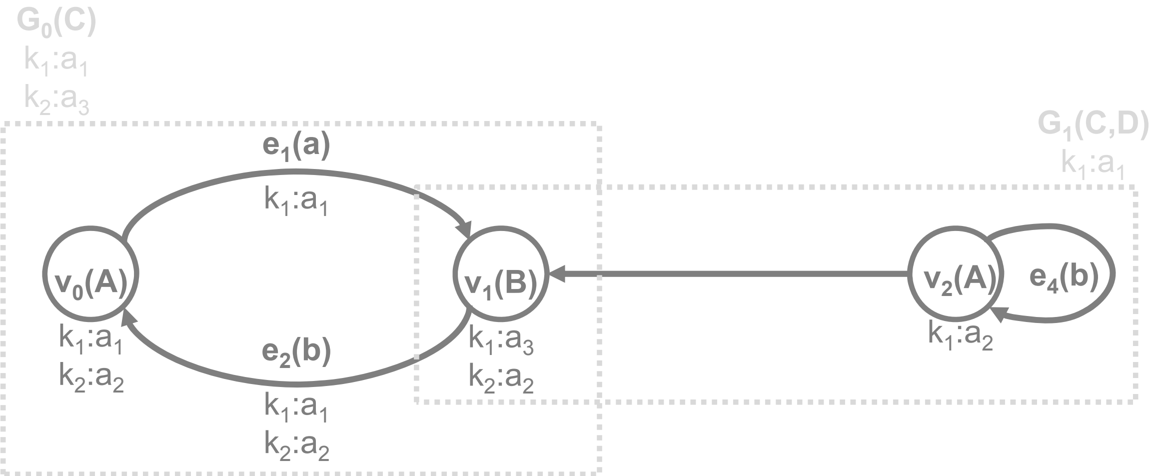


Table ´vertices´

<table>
<tr><td rowspan="3">0-0</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle a, 0-1{,}0 \rangle$</td><td colspan="2">$\langle b, 0-1{,}0 \rangle$</td></tr>
<tr><td>A</td><td>1</td><td>[0]</td><td>$\langle t_1, a_1 \rangle$</td><td>$\langle t_2, a_2 \rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1 \rangle)]$</td><td colspan="2">$[(k_1, \langle t_1, a_1 \rangle), (k_2, \langle t_2, a_2 \rangle)]$</td></tr>
<tr><td rowspan="3">0-1</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle b, 0-0{,}0 \rangle$</td><td>$\langle a, 0-0{,}0 \rangle$</td><td>$\langle a, 0-2{,}0 \rangle$</td></tr>
<tr><td>B</td><td>1</td><td>[0,1]</td><td>$\langle t_2, a_3 \rangle$</td><td>$\langle t_2, a_2 \rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1 \rangle), (k_2, \langle t_2, a_2 \rangle)]$</td><td>$[(k_1, \langle t_1, a_1 \rangle)]$</td><td>[]</td></tr>
<tr><td rowspan="3">0-2</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td colspan="2">k_1</td><td>$\langle a, 0-1{,}0 \rangle$</td><td>$\langle b, 0-2{,}1 \rangle$</td><td colspan="2">$\langle b, 0-2{,}1 \rangle$</td></tr>
<tr><td>A</td><td>2</td><td>[1]</td><td colspan="2">$\langle t_2, a_2 \rangle$</td><td colspan="2">[]</td><td>[]</td><td>[]</td></tr>
</table>

ScaDS DRESDEN LEIPZIG

VERTEX TABLE

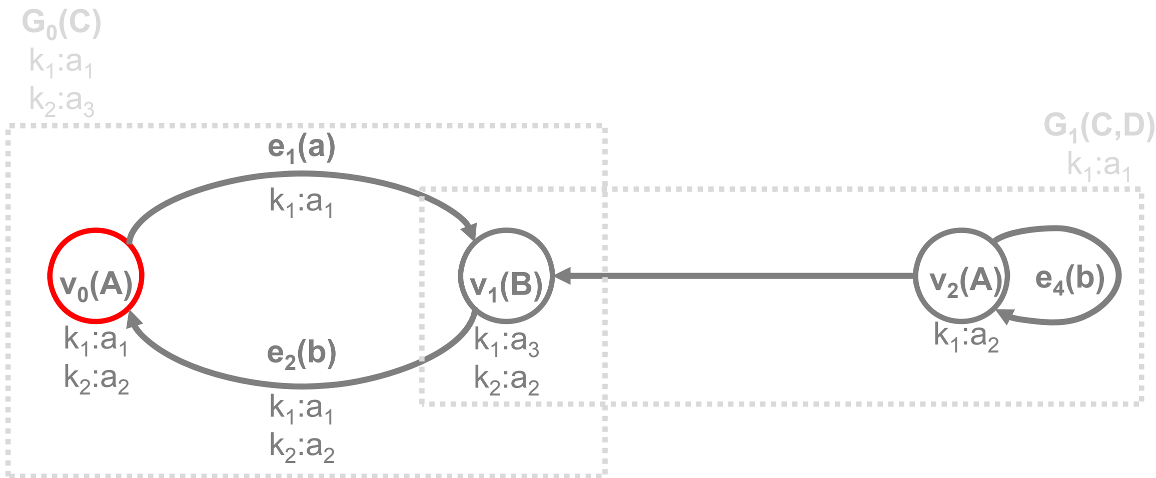


Table ´vertices´

<table>
<tr><td rowspan="3">0-0</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle a, 0-1{,}0\rangle$</td><td colspan="2">$\langle b, 0-1{,}0\rangle$</td></tr>
<tr><td>A</td><td>1</td><td>[0]</td><td>$\langle t_1, a_1\rangle$</td><td>$\langle t_2, a_2\rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1\rangle)]$</td><td colspan="2">$[(k_1, \langle t_1, a_1\rangle), (k_2, \langle t_2, a_2\rangle)]$</td></tr>
<tr><td rowspan="3">0-1</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle b, 0-0{,}0\rangle$</td><td>$\langle a, 0-0{,}0\rangle$</td><td>$\langle a, 0-2{,}0\rangle$</td></tr>
<tr><td>B</td><td>1</td><td>[0,1]</td><td>$\langle t_2, a_3\rangle$</td><td>$\langle t_2, a_2\rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1\rangle), (k_2, \langle t_2, a_2\rangle)]$</td><td>$[(k_1, \langle t_1, a_1\rangle)]$</td><td>[]</td></tr>
<tr><td rowspan="3">0-2</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td colspan="2">k_1</td><td>$\langle a, 0-1{,}0\rangle$</td><td>$\langle b, 0-2{,}1\rangle$</td><td colspan="2">$\langle b, 0-2{,}1\rangle$</td></tr>
<tr><td>A</td><td>2</td><td>[1]</td><td colspan="2">$\langle t_2, a_2\rangle$</td><td colspan="2">[]</td><td>[]</td><td>[]</td></tr>
</table>

ScaDS DRESDEN LEIPZIG

VERTEX TABLE

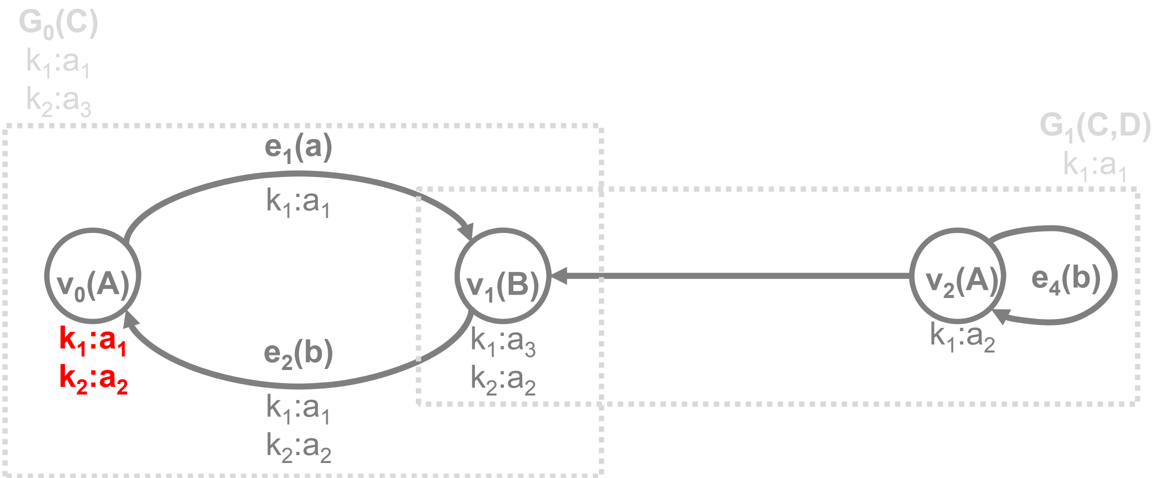


Table ´vertices´

0-0	meta			properties		out edges		in edges	
	type	idx	graphs	$\boldsymbol{k_1}$	$\boldsymbol{k_2}$	$\langle a,0-1,0\rangle$		$\langle b,0-1,0\rangle$	
	A	1	[0]	$\boldsymbol{\langle t_1,a_1\rangle}$	$\boldsymbol{\langle t_2,a_2\rangle}$	$[(k_1,\langle t_1,a_1\rangle)]$		$[(k_1,\langle t_1,a_1\rangle),(k_2,\langle t_2,a_2\rangle)]$	
0-1	meta			properties		out edges		in edges	
	type	idx	graphs	k_1	k_2	$\langle b,0-0,0\rangle$		$\langle a,0-0,0\rangle$	$\langle a,0-2,0\rangle$
	B	1	[0,1]	$\langle t_2,a_3\rangle$	$\langle t_2,a_2\rangle$	$[(k_1,\langle t_1,a_1\rangle),(k_2,\langle t_2,a_2\rangle)]$		$[(k_1,\langle t_1,a_1\rangle)]$	[]
0-2	meta			properties		out edges		in edges	
	type	idx	graphs	k_1		$\langle a,0-1,0\rangle$	$\langle b,0-2,1\rangle$	$\langle b,0-2,1\rangle$	
	A	2	[1]	$\langle t_2,a_2\rangle$		[]		[]	[]

ScaDS DRESDEN LEIPZIG

VERTEX TABLE

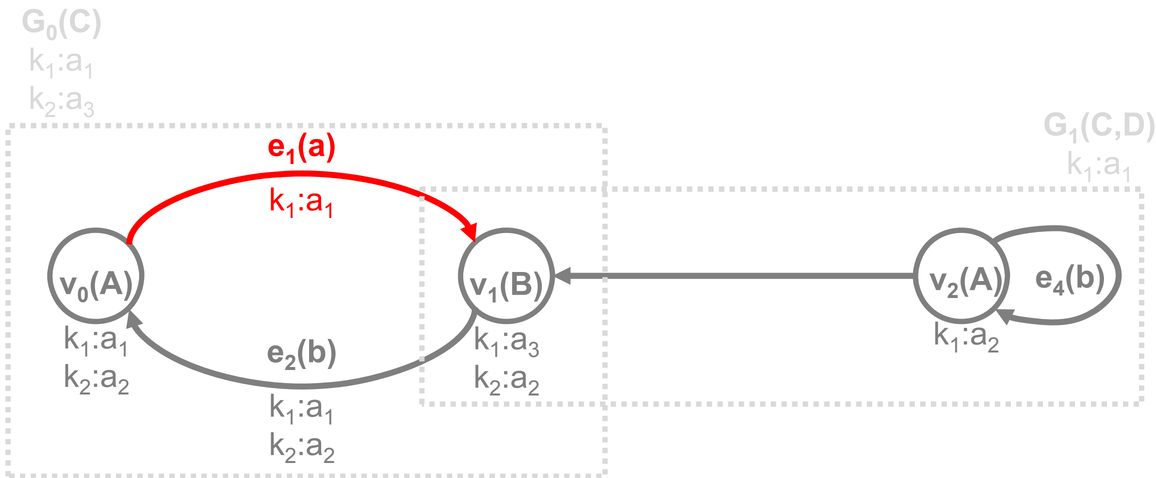


Table ´vertices´

<table>
<tr><td rowspan="3">0-0</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle \mathbf{a}, \mathbf{0}-\mathbf{1}, \mathbf{0} \rangle$</td><td colspan="2">$\langle b, 0-1,0 \rangle$</td></tr>
<tr><td>A</td><td>1</td><td>[0]</td><td>$\langle t_1, a_1 \rangle$</td><td>$\langle t_2, a_2 \rangle$</td><td colspan="2">$[(\mathbf{k_1}, \langle \mathbf{t_1}, \mathbf{a_1} \rangle)]$</td><td colspan="2">$[(k_1, \langle t_1, a_1 \rangle), (k_2, \langle t_2, a_2 \rangle)]$</td></tr>
<tr><td rowspan="3">0-1</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle b, 0-0,0 \rangle$</td><td>$\langle \mathbf{a}, \mathbf{0}-\mathbf{0}, \mathbf{0} \rangle$</td><td>$\langle a, 0-2,0 \rangle$</td></tr>
<tr><td>B</td><td>1</td><td>[0,1]</td><td>$\langle t_2, a_3 \rangle$</td><td>$\langle t_2, a_2 \rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1 \rangle), (k_2, \langle t_2, a_2 \rangle)]$</td><td>$[(\mathbf{k_1}, \langle \mathbf{t_1}, \mathbf{a_1} \rangle)]$</td><td>[]</td></tr>
<tr><td rowspan="3">0-2</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td colspan="2">k_1</td><td>$\langle a, 0-1,0 \rangle$</td><td>$\langle b, 0-2,1 \rangle$</td><td colspan="2">$\langle b, 0-2,1 \rangle$</td></tr>
<tr><td>A</td><td>2</td><td>[1]</td><td colspan="2">$\langle t_2, a_2 \rangle$</td><td colspan="2">[]</td><td>[]</td><td>[]</td></tr>
</table>

ScaDS DRESDEN LEIPZIG

VERTEX TABLE

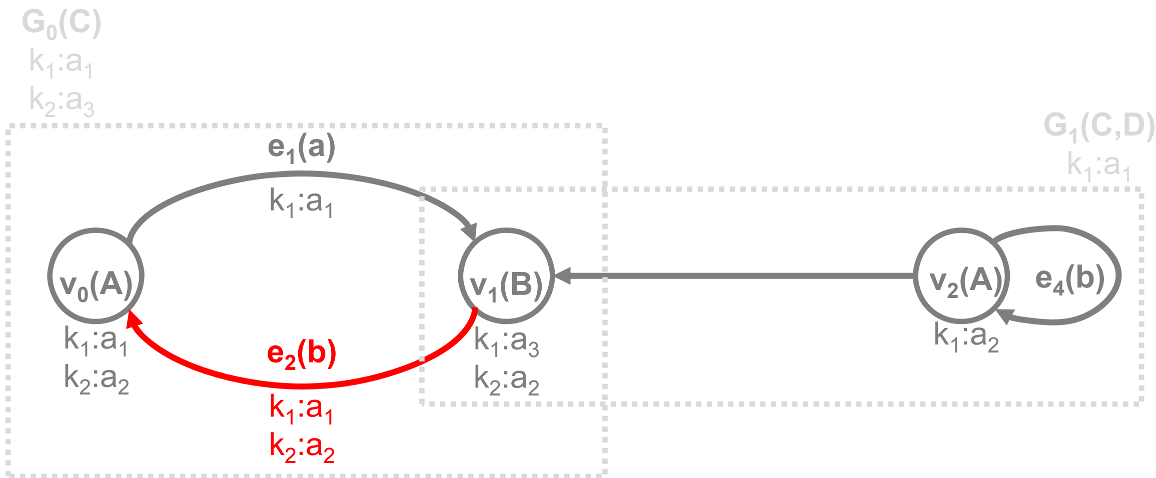


Table ´vertices´

<table>
<tr><td rowspan="3">0-0</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle a, 0-1,0\rangle$</td><td colspan="2">$\langle \boldsymbol{b}, \mathbf{0}-\mathbf{1}, \mathbf{0}\rangle$</td></tr>
<tr><td>A</td><td>1</td><td>[0]</td><td>$\langle t_1, a_1\rangle$</td><td>$\langle t_2, a_2\rangle$</td><td colspan="2">$[(k_1, \langle t_1, a_1\rangle)]$</td><td colspan="2">$[(\boldsymbol{k_1}, \langle \boldsymbol{t_1}, \boldsymbol{a_1}\rangle), (\boldsymbol{k_2}, \langle \boldsymbol{t_2}, \boldsymbol{a_2}\rangle)]$</td></tr>
<tr><td rowspan="3">0-1</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td>k_1</td><td>k_2</td><td colspan="2">$\langle \boldsymbol{b}, \mathbf{0}-\mathbf{0}, \mathbf{0}\rangle$</td><td>$\langle a, 0-0,0\rangle$</td><td>$\langle a, 0-2,0\rangle$</td></tr>
<tr><td>B</td><td>1</td><td>[0,1]</td><td>$\langle t_2, a_3\rangle$</td><td>$\langle t_2, a_2\rangle$</td><td colspan="2">$[(\boldsymbol{k_1}, \langle \boldsymbol{t_1}, \boldsymbol{a_1}\rangle), (\boldsymbol{k_2}, \langle \boldsymbol{t_2}, \boldsymbol{a_2}\rangle)]$</td><td>$[(k_1, \langle t_1, a_1\rangle)]$</td><td>[]</td></tr>
<tr><td rowspan="3">0-2</td><td colspan="3">meta</td><td colspan="2">properties</td><td colspan="2">out edges</td><td colspan="2">in edges</td></tr>
<tr><td>type</td><td>idx</td><td>graphs</td><td colspan="2">k_1</td><td>$\langle a, 0-1,0\rangle$</td><td>$\langle b, 0-2,1\rangle$</td><td colspan="2">$\langle b, 0-2,1\rangle$</td></tr>
<tr><td>A</td><td>2</td><td>[1]</td><td colspan="2">$\langle t_2, a_2\rangle$</td><td colspan="2">[]</td><td>[]</td><td>[]</td></tr>
</table>

ScaDS DRESDEN LEIPZIG

EXAMPLE GRALA WORKFLOWS

1. Business Intelligence
   - Top Revenue Subgraph
   - Find the common subgraph of the top 100 revenue business transaction graphs
2. Social Network Analysis
   - "Summarized Communities"
   - Find communities by label propagation
   - Filter communities by number of users
   - Summarize vertices per community
     and edges between community members

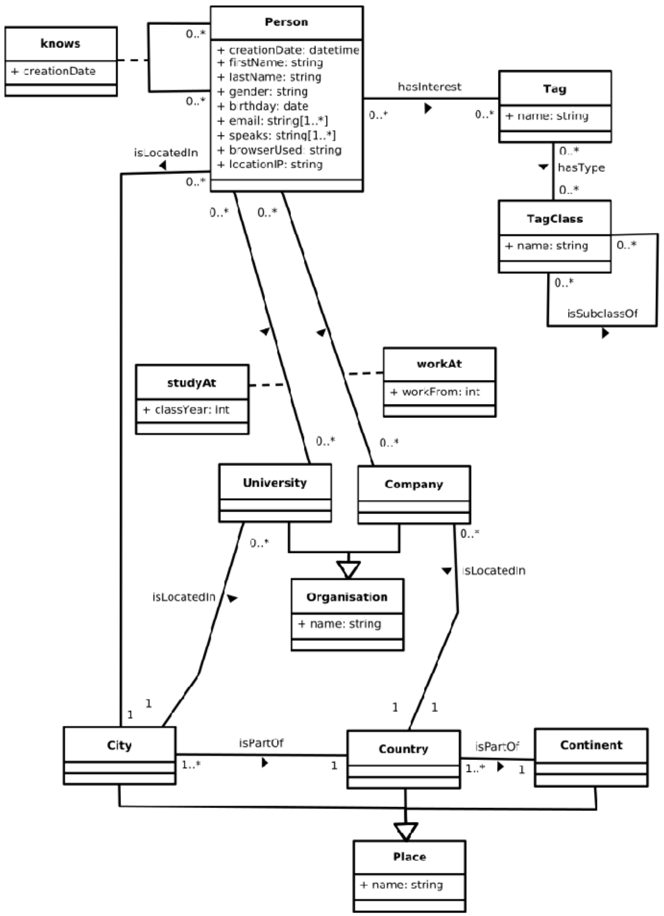
GRALA EXAMPLE : TOP REVENUE SUBGRAPH

```
// compute logical graphs
1: btgs = db.callForCollection( :BusinessTransactionGraphs , {} )
// define and apply aggregate function (number of invoices per graph)
2: aggFuncInvoiceCount = ( Graph g =>
     |g.V.filter( Vertex v => v[:type] == "Invoice")|)
3: btgs = btgs.apply(
     Graph g => g.aggregate( "invoiceCount",aggFuncInvoiceCount) )
// select logical graphs with at least one invoice
4: invBtgs = btgs.select(
     Graph g => g["invoiceCount"] > 0)
// define and apply aggregate function (revenue per graph)
5: aggFuncRevenue = ( Graph g =>
     g.V.values("revenue").sum())
6: invBtgs = invBtgs.apply(
     Graph g => g.aggregate( "revenue",aggFuncRevenue) )
// sort graphs by revenue and return top 100
7: topBtgs = invBtgs.sortBy( “revenue“ , :desc ).top( 100 )
// compute overlap to find master data objects (e.g., Employees)
8: topBtgOverlap = invBtgs.reduce(
     Graph g, Graph h => g.overlap(h))
```

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

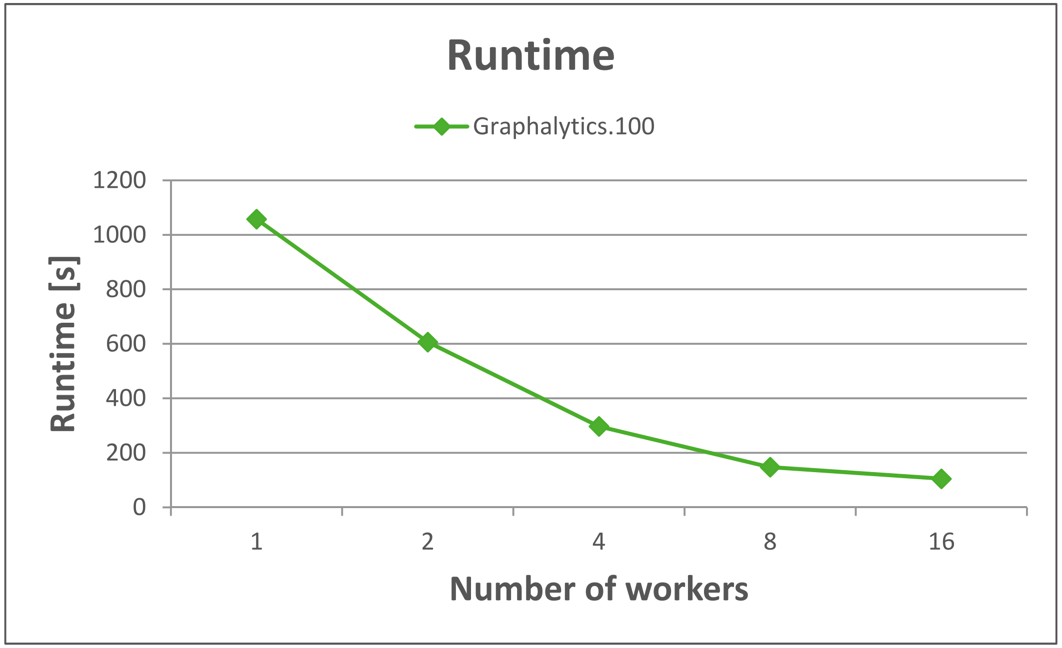
SUMMARIZED COMMUNITIES




```
socialNetwork
 .subgraph(
  (v => v.label == ‘Person‘),
  (e => e.label == ‘knows‘))
 .transform(
  (gIn, gOut => gOut = gIn),
  (vIn, vOut => {
   vOut.label      = vIn.label,
   vOut[‘city‘]    = vIn[‘city‘]
   vOut[‘gender‘] = vIn[‘gender‘]
   vOut[‘key‘]     = vIn[‘birthday‘]}),
  (eIn, eOut) => eOut.label = eIn.label))
 .callForCollection(:LabelPropagation, [‘key‘, 4])
 .apply(g =>
  g.aggregate(‘vertexCount‘, (h => |h.V|))
 .select(g => g[‘vertexCount‘] > 50000)
 .reduce(g, h => g.combine(h))
 .groupBy(
  [‘city‘,‘gender‘], (superVertex, vertices =>
   superVertex[‘count‘] = |vertices|),
  [], (superEdge, edges =>
   superEdge[‘count‘] = |edges|)
 .aggregate(‘vCount‘, (g => |g.V|))
 .aggregate(‘eCount‘, (g => |g.E|))
```

ScaDS DRESDEN LEIPZIG

BENCHMARK RESULTS

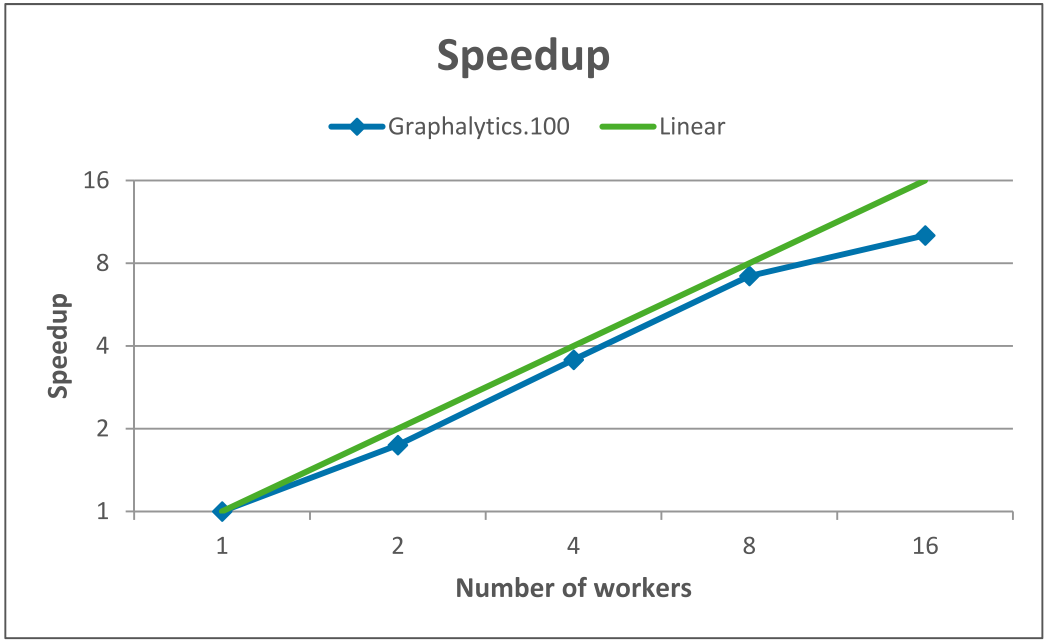


Dataset	# Vertices	# Edges
Graphalytics.1	61,613	2,026,082
Graphalytics.10	260,613	16,600,778
Graphalytics.100	1,695,613	147,437,275
Graphalytics.1000	12,775,613	1,363,747,260
Graphalytics.10000	90,025,613	10,872,109,028

- 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

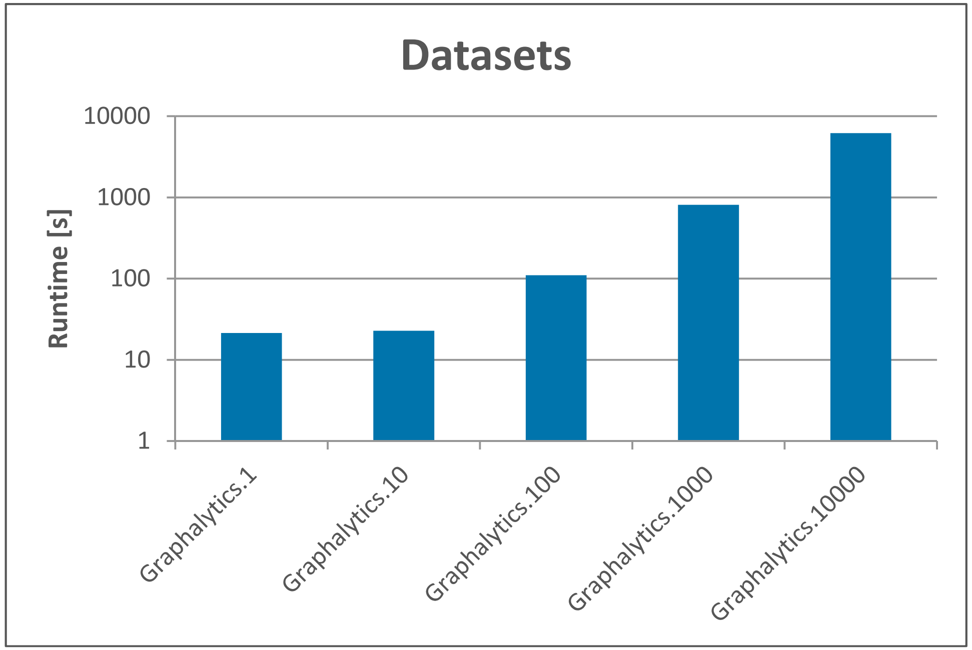
BENCHMARK RESULTS 2



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

BENCHMARK RESULTS 3



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

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

ScaDS DRESDEN LEIPZIG

SUMMARY

- Graph-based data integration
  - centralized „linked data“ using PGM rather than RDF
  - data/metadata extraction and transformation into graphs
  - linking / matching + fusion
- Big Graph Analytics
  - high potential even for business intelligence (BIIIG)
  - Hadoop-based graph processing frameworks based on generic graphs
  - Spark/Flink: batch-oriented workflows (rather than OLAP)
  - Graph collections not generally supported

ScaDS DRESDEN LEIPZIG

SUMMARY (2)

- GraDoop
  - 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) and support for graph collections
  - leverages Hadoop ecosystem
    - Apache HBase for permanent graph storage
    - Apache Flink to implement operators
  - ongoing implementation

ScaDS DRESDEN LEIPZIG

OUTLOOK / CHALLENGES

- Graph-based data integration
  - unified approach for knowledge graphs and regular data graphs
  - evaluate/improve scalability and data quality
- Graph analytics
  - automatic optimization of analysis workflows
  - optimized graph partitioning approaches
  - load balancing
  - interactive graph analytics
  - visualization of graphs and analysis results

ScaDS DRESDEN LEIPZIG

REFERENCES

- R. Angles, C. Guiterrez. *Survey of Graph Database Models*. ACM Comput. Surv., 2008
- M. Junghanns, A. Petermann, K. Gomez, E. Rahm: *GRADOOP - Scalable Graph Data Management and Analytics with Hadoop*. Tech. report, Univ. of Leipzig, June 2015
- M. Junghanns, A. Petermann, N. Teichmann, K. Gomez, E. Rahm: *Analyzing Extended Property Graphs with Apache Flink*. Tech. report, Univ. of Leipzig, Feb. 2016
- 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
- Petermann, A.; Junghanns, M.; Müller, R.; Rahm, E.: *FoodBroker - Generating Synthetic Datasets for Graph-Based Business Analytics*. Proc. 5th Int. Workshop on Big Data Benchmarking (WBDB), 2014
- R.S. Xin, J.E. Gonzales, M.J. Franklin, I.Stoica: *GraphX: A resilient distributed graph system on Spark*. GRADES, 2013
- Z. Wang et. al. *Pagrol: Parallel Graph OLAP over Large-scale Attributed Graphs*. ICDE, 2014
- P. Zhao et. al. Graph Cube: *On Warehousing and OLAP Multidimensional Networks*. Sigmod, 2011