ScaDS
DRESDEN LEIPZIG

SCALABLE GRAPH ANALYTICS WITH GRADOOP AND BIIIG

MARTIN JUNGHANNS, ANDRE PETERMANN, ERHARD RAHM

ScaDS DRESDEN LEIPZIG

RESEARCH ON GRAPH ANALYTICS

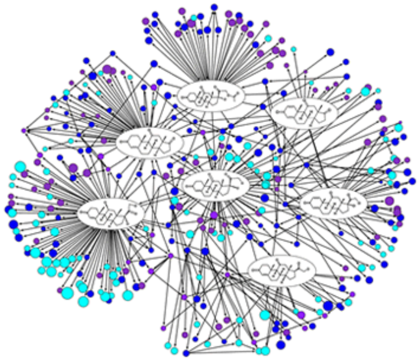
- Graph Analytics on Hadoop (Gradoop)
  - Distributed graph data management
  - Rich graph data model with powerful operators
  - Domain independent

- Business Intelligence with Integrated Instance Graphs (BIIIG)
  - Graph-based data integration
  - Graph OLAP, Mining and visualization
  - Improved Scalability on Gradoop

„GRAPHS ARE EVERYWHERE“ AND LARGE

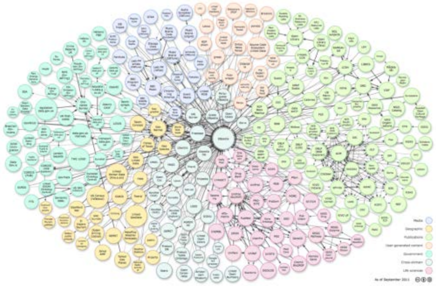
Social science

Life science

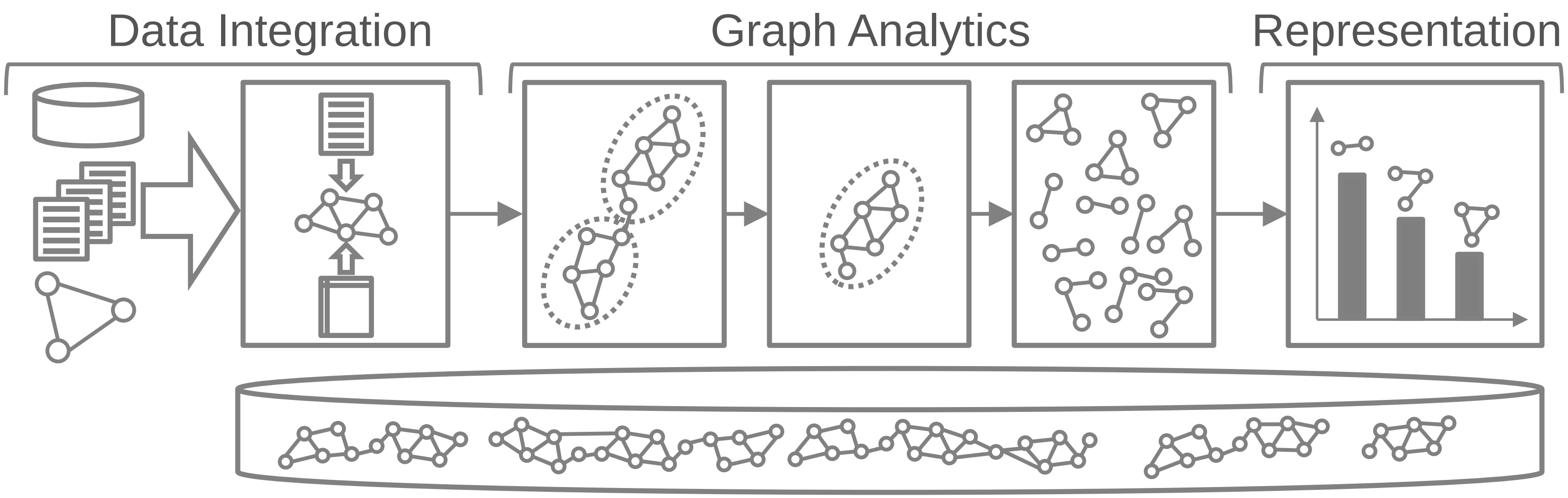


Engineering

Information science



END-TO-END GRAPH ANALYTICS



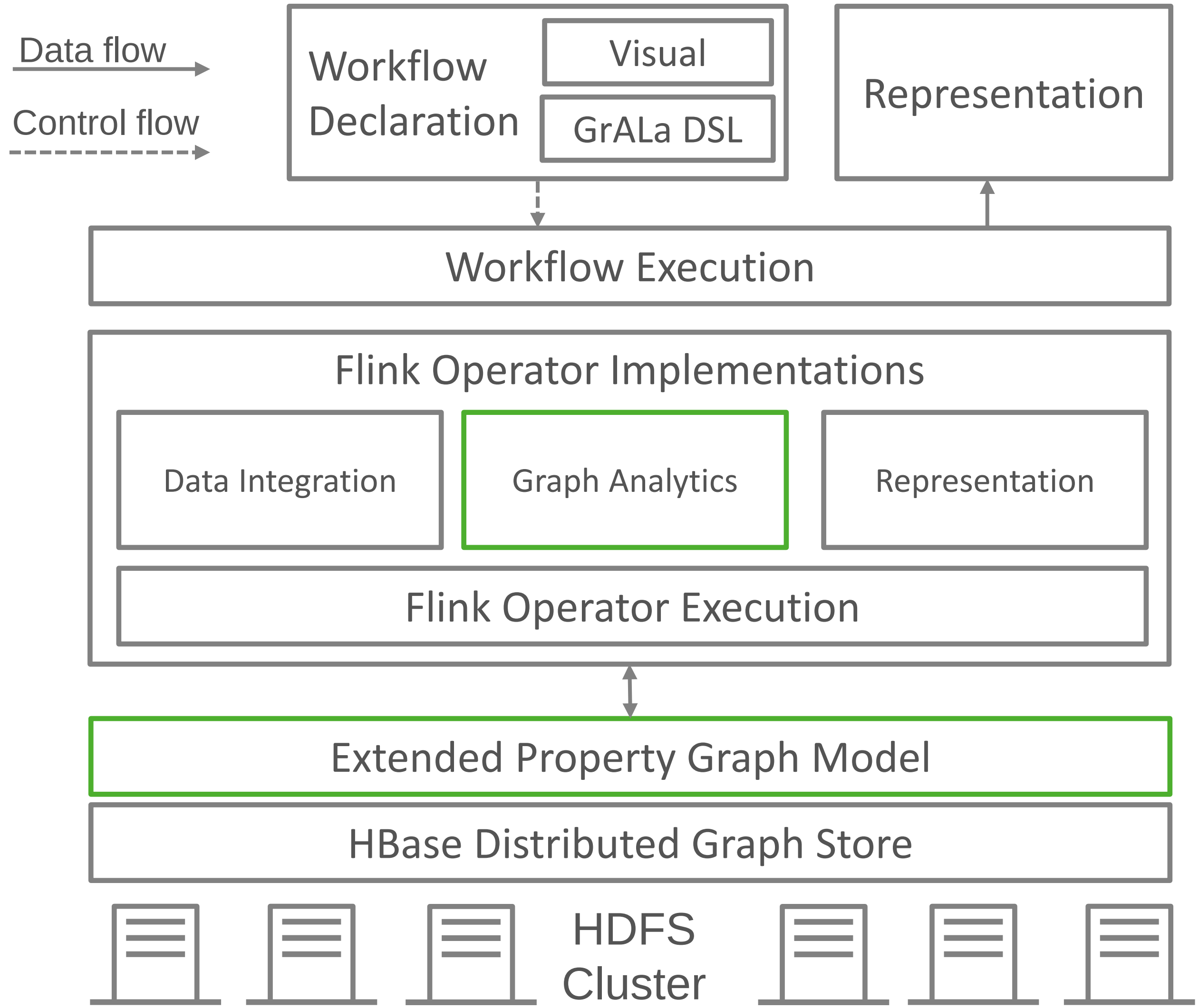
- **Integrate data** from one or more sources into a dedicated **graph storage** with **common graph data model**
- Definition of **analytical workflows** from **operator algebra**
- Result representation in **meaningful way**

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

gradoop

ScaDS DRESDEN LEIPZIG

HIGH LEVEL ARCHITECTURE



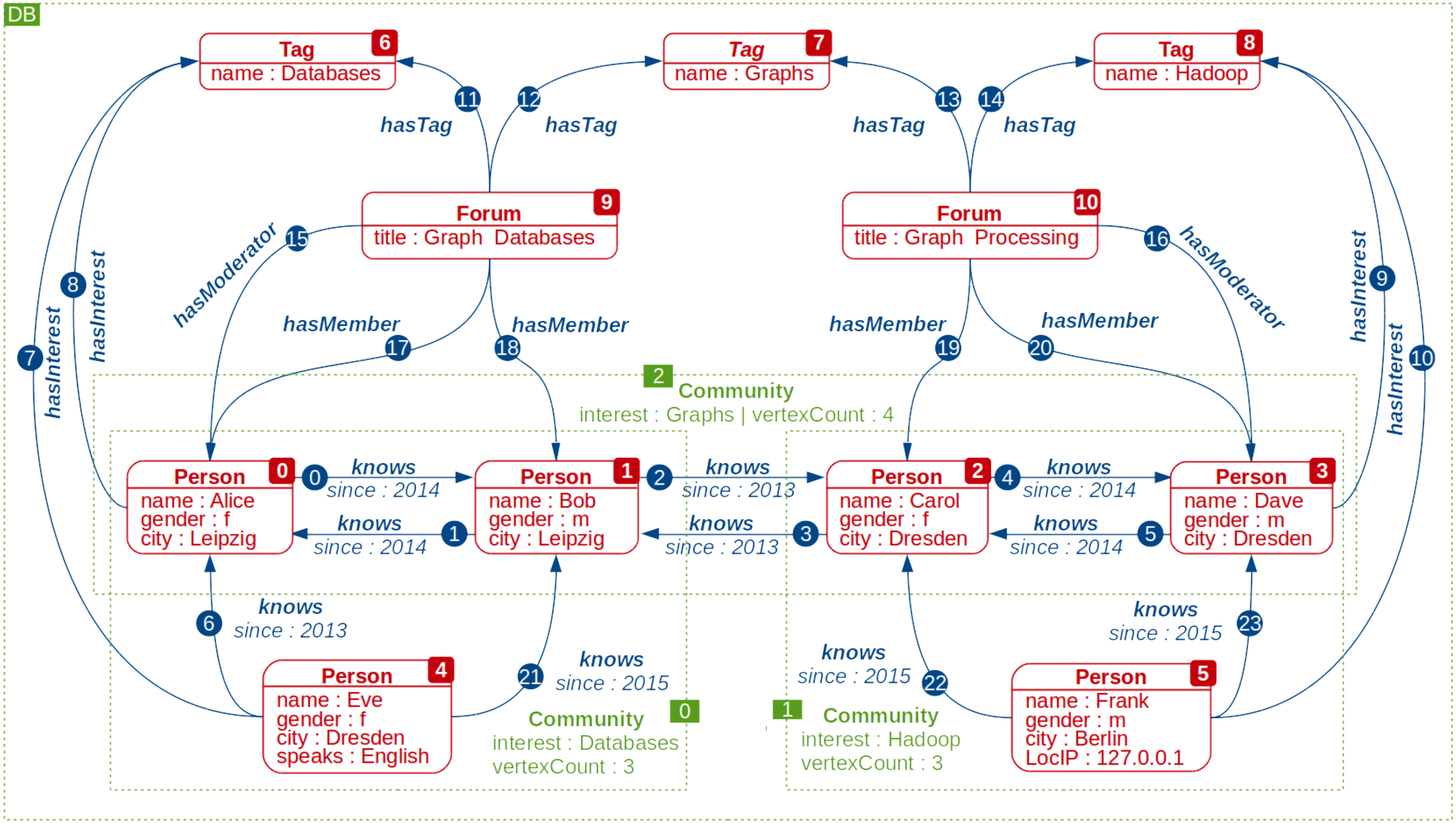
ScaDS DRESDEN LEIPZIG

DATA MODEL - REQUIREMENTS

1. Simple but powerful
   - intuitive graphs are flat structures of vertices and binary edges
2. Logical graphs
   - support of multiple, possibly overlapping graphs in one database is advantageous for analytical applications
3. Attributes and type labels
   - type labels and custom properties for vertices, edges and graphs
4. Parallel edges and loops
   - allow multiple relations between two vertices and self-connected relations

ScaDS DRESDEN LEIPZIG

EXTENDED PROPERTY GRAPH MODEL



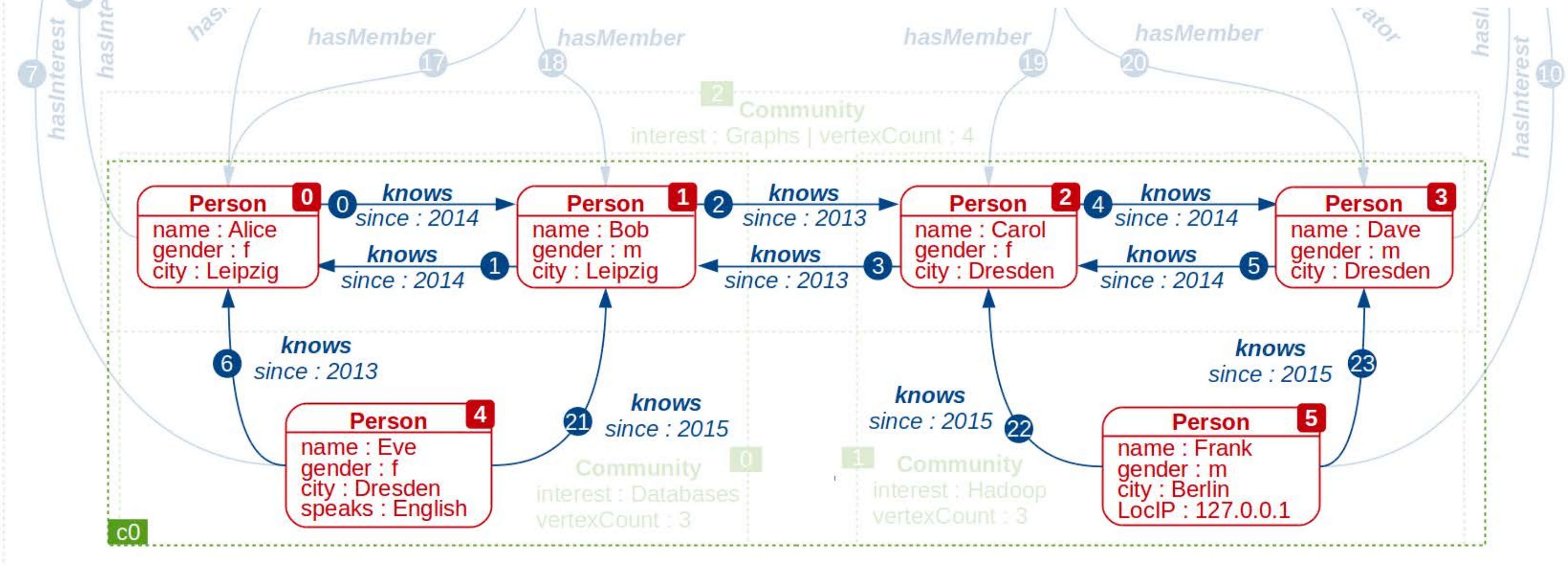
GRAPH OPERATORS

Operator	Definition	GrALa notation
unary		
Pattern Matching	$\mu_{G^*,\varphi} : \mathcal{G} \rightarrow \mathcal{G}^n$	graph.match(patternGraph,predicate) : Collection
Aggregation	$\gamma_a : \mathcal{G} \rightarrow \mathcal{G}$	graph.aggregate(propertyKey,aggregateFunction) : Graph
Projection	$\pi_{\upsilon,\epsilon} : \mathcal{G} \rightarrow \mathcal{G}$	graph.project(vertexFunction,edgeFunction) : Graph
Summarization	$\varsigma_{\upsilon,\epsilon} : \mathcal{G} \rightarrow \mathcal{G}$	graph.summarize(vertexGroupKeys, vertexAggregateFunction, edgeGroupKeys,edgeAggregateFunction) : Graph
binary		
Combination	$\sqcup : \mathcal{G}^2 \rightarrow \mathcal{G}$	graph.combine(otherGraph) : Graph
Overlap	$\sqcap : \mathcal{G}^2 \rightarrow \mathcal{G}$	graph.overlap(otherGraph) : Graph
Exclusion	$- : \mathcal{G}^2 \rightarrow \mathcal{G}$	graph.exclude(otherGraph) : Graph

ScaDS DRESDEN LEIPZIG

WORKFLOW EXAMPLE: SUMMARIZATION

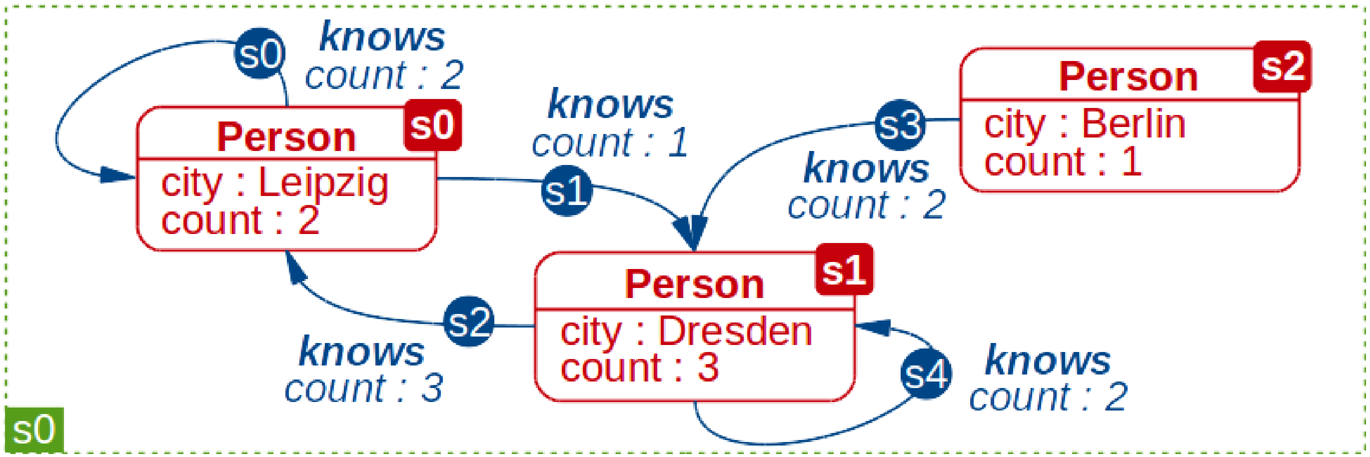
```
1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = {:type, “city”}
3: edgeGroupingKeys = {:type}
4: vertexAggFunc = (Vertex vSum, Set vertices => vSum[“count”] = |vertices|)
5: edgeAggFunc = (Edge eSum, Set edges => eSum[“count”] = |edges|)
6: sumGraph = personGraph.summarize(vertexGroupingKeys, edgeGroupingKeys,
     vertexAggFunc, edgeAggFunc)
```



ScaDS DRESDEN LEIPZIG

WORKFLOW EXAMPLE: SUMMARIZATION

```
1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = {:type, “city”}
3: edgeGroupingKeys = {:type}
4: vertexAggFunc = (Vertex vSum, Set vertices => vSum[“count”] = |vertices|)
5: edgeAggFunc = (Edge eSum, Set edges => eSum[“count”] = |edges|)
6: sumGraph = personGraph.summarize(vertexGroupingKeys, edgeGroupingKeys,
     vertexAggFunc, edgeAggFunc)
```



ScaDS DRESDEN LEIPZIG

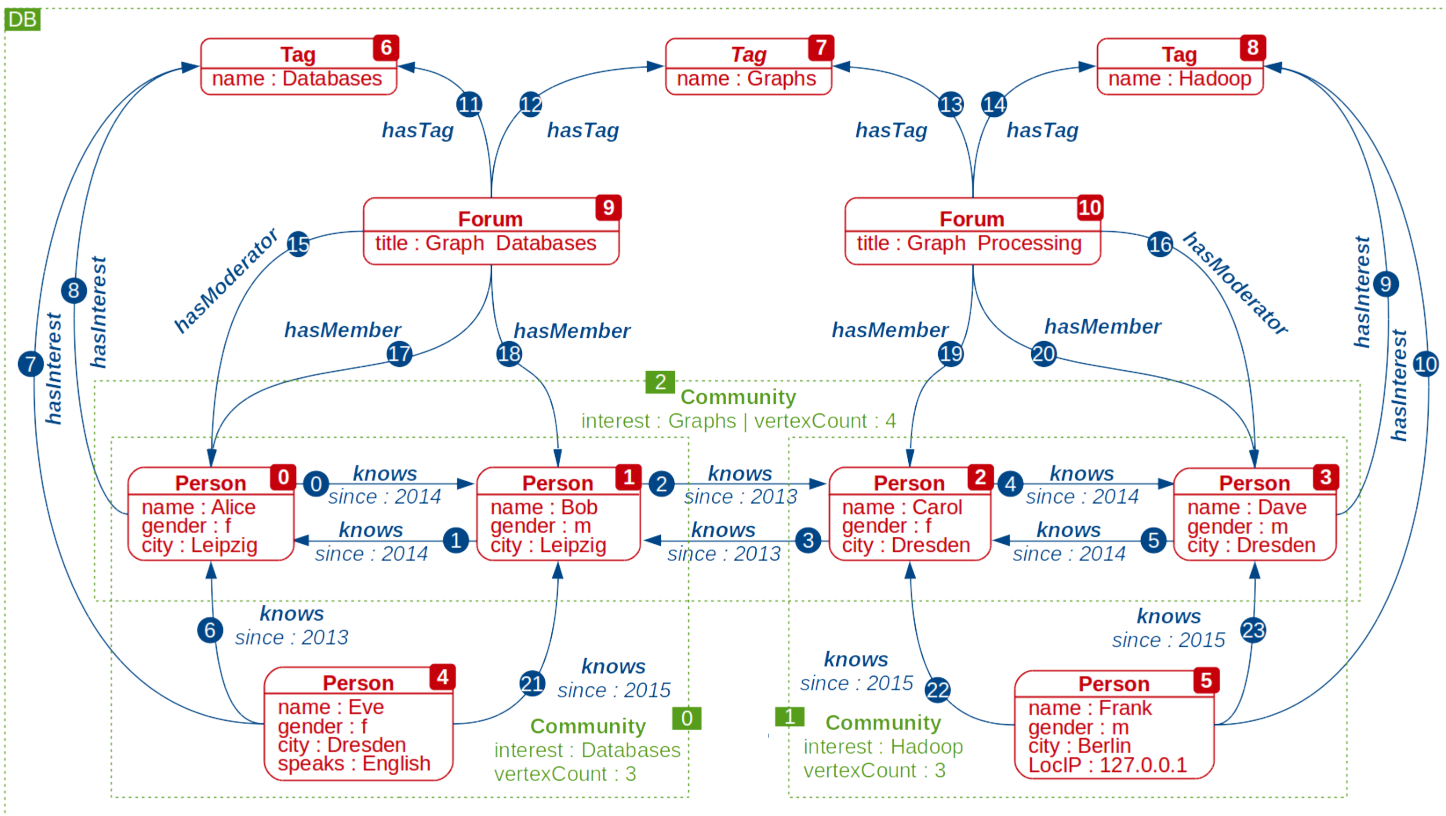
COLLECTION OPERATORS

Operator	Definition	GrALa notation
collection		
Selection	$\sigma_\varphi : \mathcal{G}^n \rightarrow \mathcal{G}^n$	collection.select(predicate) : Collection
Distinct	$\delta : \mathcal{G}^n \rightarrow \mathcal{G}^n$	collection.distinct() : Collection
Sort by	$\xi_{k,d} : \mathcal{G}^n \rightarrow \mathcal{G}^n$	collection.sortBy(key, [:asc|:desc]) : Collection
Top	$\beta_n : \mathcal{G}^n \rightarrow \mathcal{G}^n$	collection.top(limit) : Collection
Union	$\cup : (\mathcal{G}^n)^2 \rightarrow \mathcal{G}^n$	collection.union(otherCollection) : Collection
Intersection	$\cap : (\mathcal{G}^n)^2 \rightarrow \mathcal{G}^n$	collection.intersect(otherCollection) : Collection
Difference	$\setminus : (\mathcal{G}^n)^2 \rightarrow \mathcal{G}^n$	collection.difference(otherCollection) : Collection
auxiliary		
Apply	$\lambda_o : \mathcal{G}^n \rightarrow \mathcal{G}^n$	collection.apply(unaryGraphOperator) : Collection
Reduce	$\rho_o : \mathcal{G}^n \rightarrow \mathcal{G}$	collection.reduce(binaryGraphOperator) : Graph
Call	$\eta_{a,P} : \mathcal{G} \cup \mathcal{G}^n \rightarrow \mathcal{G} \cup \mathcal{G}^n$	[graph|collection].callFor[Graph|Collection](algorithm,parameters) : [Graph|Collection]

ScaDS DRESDEN LEIPZIG

SELECTION

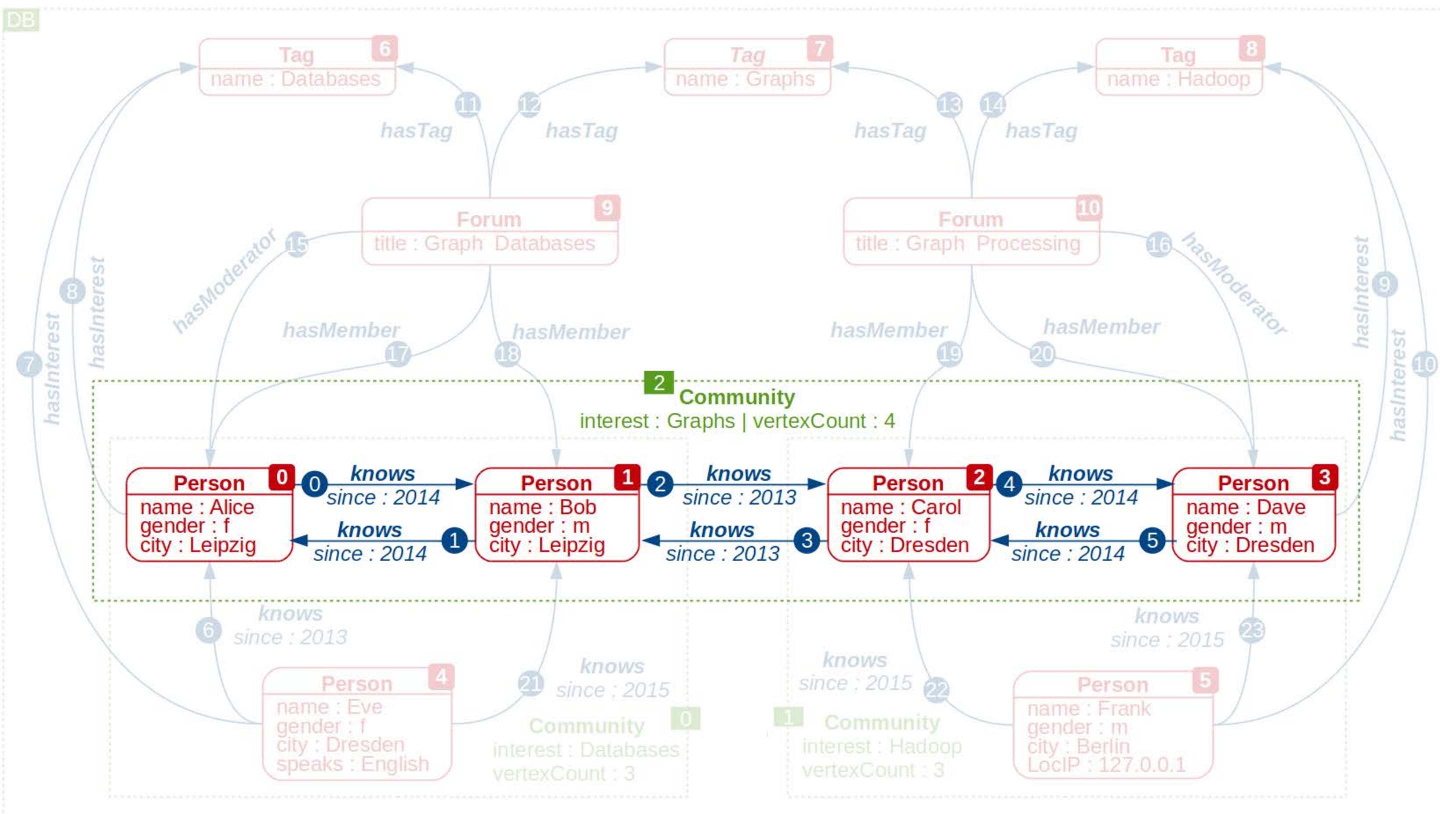
```
1: collection = <db.G[0],db.G[1],db.G[2]>
2: predicate = (Graph g => |g.V| > 3
3: result = collection.select(predicate)
```



ScaDS DRESDEN LEIPZIG

SELECTION

```
collection = <db.G[0],db.G[1],db.G[2]>
predicate = (Graph g => |g.V| > 3
result = collection.select(predicate)
```



FLINK IMPLEMENTATION STATE

Operator	Implementation	Operator	Implementation
unary		**collection**	
Pattern Matching		Selection	
Aggregation		Distinct	
Projection		Sort by	
Summarization		Top	
binary		Union	
Combination		Intersection	
Overlap		Difference	
Exclusion		**auxiliary**	
		Apply	
		Reduce	
		Call	

ScaDS DRESDEN LEIPZIG

SUMMARY & ROADMAP: GRADOOP

- Summary
  - end-to-end framework for graph data management and analytics
  - extended property graph model (EPGM) with powerful operators
  - initial implementation running (HBase, MapReduce and Giraph)
- Roadmap
  - WIP: workflow execution layer (Flink, Spark, ...)
  - WIP: reference implementation for all operators
  - optimized graph partitioning approaches
  - graph-based data integration (DeDoop)

BIIIG ON GRADOOP

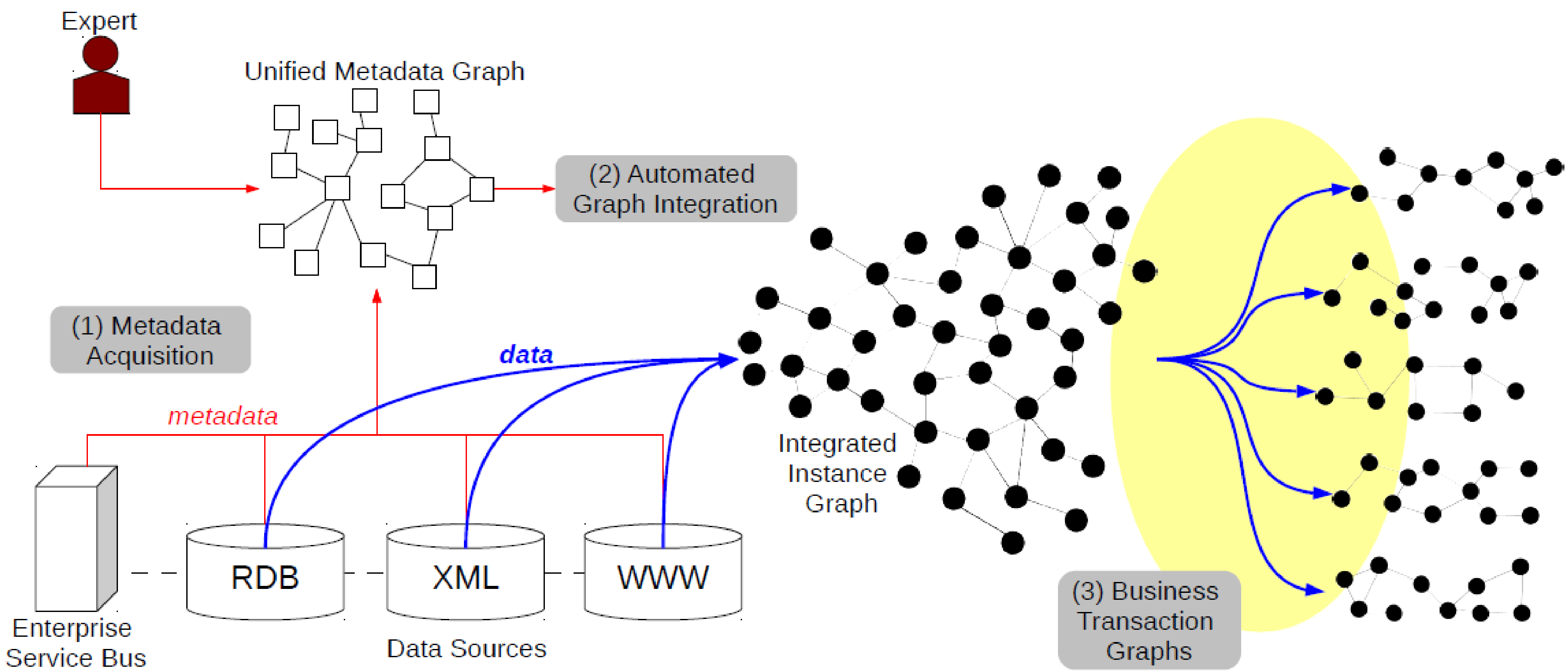
- Fitting data model

- Complex Analytics composed of Gradoop Operators

- Example: Cluster Characteristic Patterns in Business Process Executions
  - Quantify clusters by business measure (e.g., profitable and lossy)
  - Characteristic = frequent within one but not in other clusters

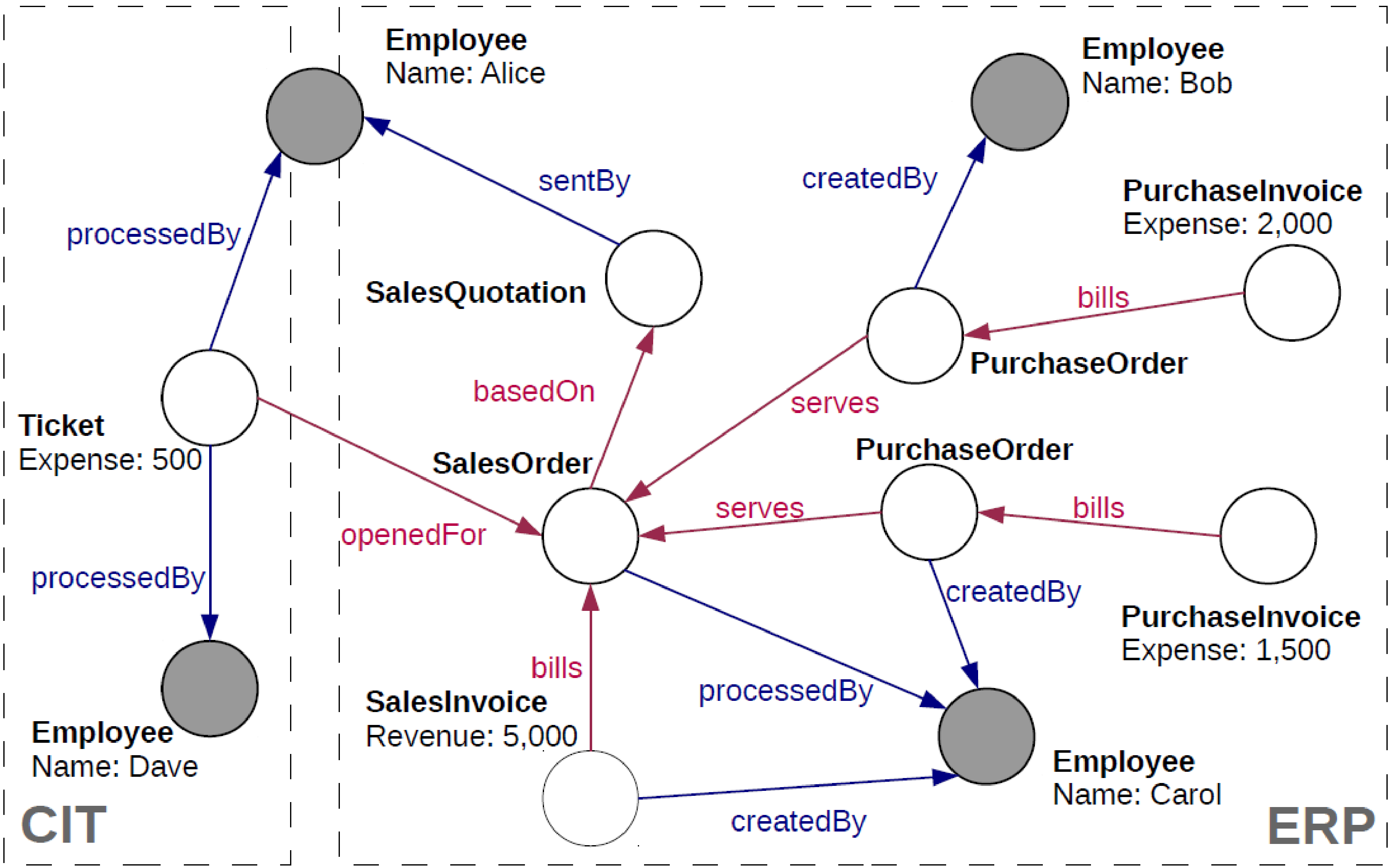
ScaDS DRESDEN LEIPZIG

BIIG OVERVIEW

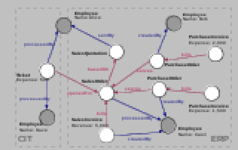


ScaDS DRESDEN LEIPZIG

BUSINESS TRANSACTION GRAPH



CLUSTER-CHARACTERISTIC PATTERNS



BTG 1

BTG 2

BTG 3

BTG 4

BTG 5

BTG 6

BTG n

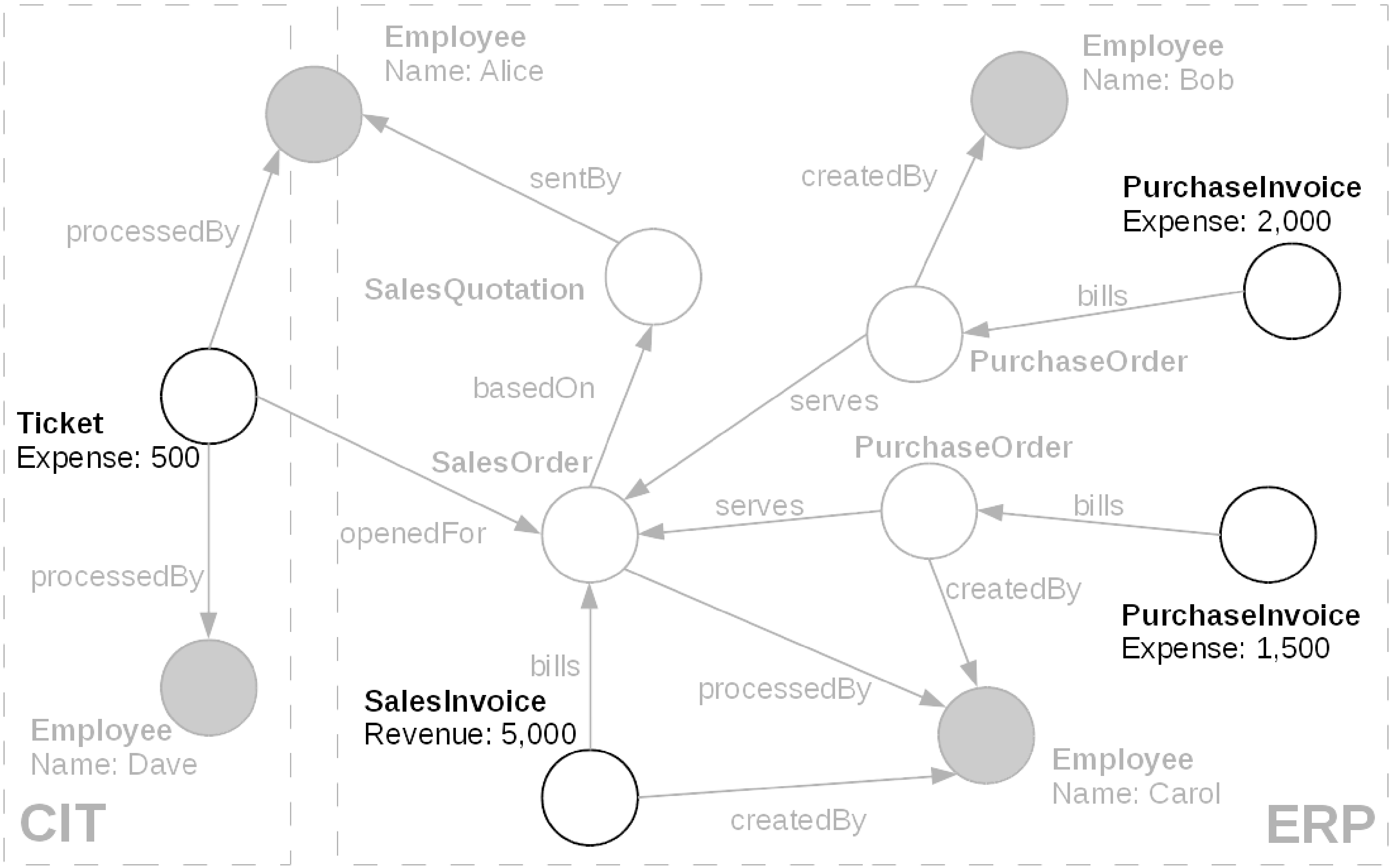
ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS

```
// generate base collection
btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )
```

ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



ScaDS DRESDEN LEIPZIG

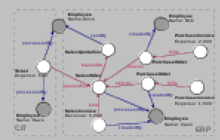
CLUSTER-CHARACTERISTIC PATTERNS

```
// generate base collection

btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )

// aggregate profit

aggFunc = ( Graph g =>
    g.V.values("Revenue").sum() - g.V.values("Expense").sum()
)
```

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



BTG 1

BTG 2

BTG 3

BTG 4

BTG 5

BTG 6

BTG n

Total Rev.	Expenses	Net Profit
5,000	- 3,000	2,000
9,000	- 3,000	6,000
2,000	- 1,500	500
4,000	- 4,500	- 500
5,000	- 7,000	- 2,000
10,000	- 15,000	-5,000
8,000	- 4,000	4,000

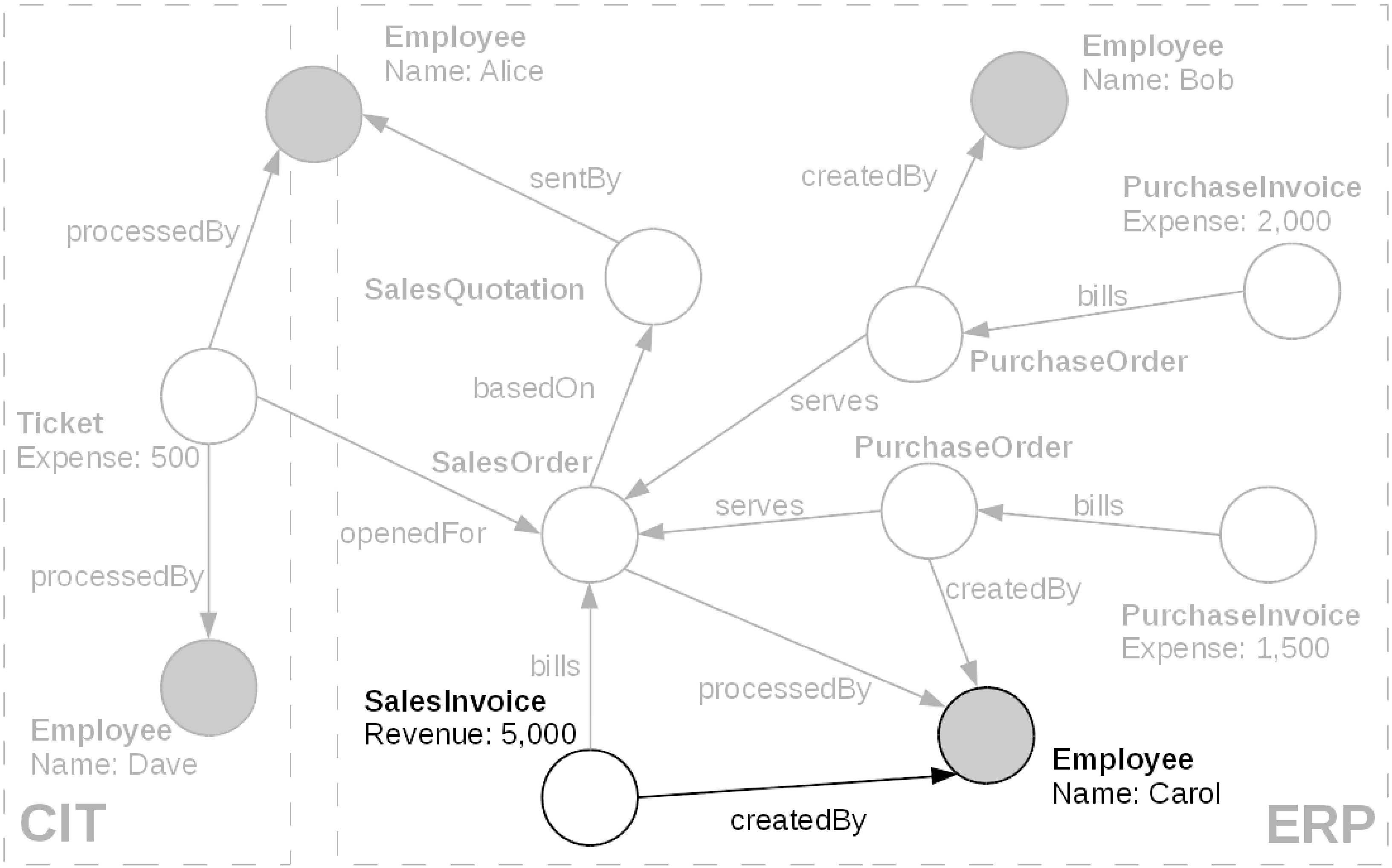
ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS

```
// generate base collection

btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )

// aggregate profit

aggFunc = ( Graph g =>
    g.V.values("Revenue").sum() - g.V.values("Expense").sum()
)

btgs = btgs.apply( Graph g =>
    g.aggregate( "Profit" , aggFunc )
)
```

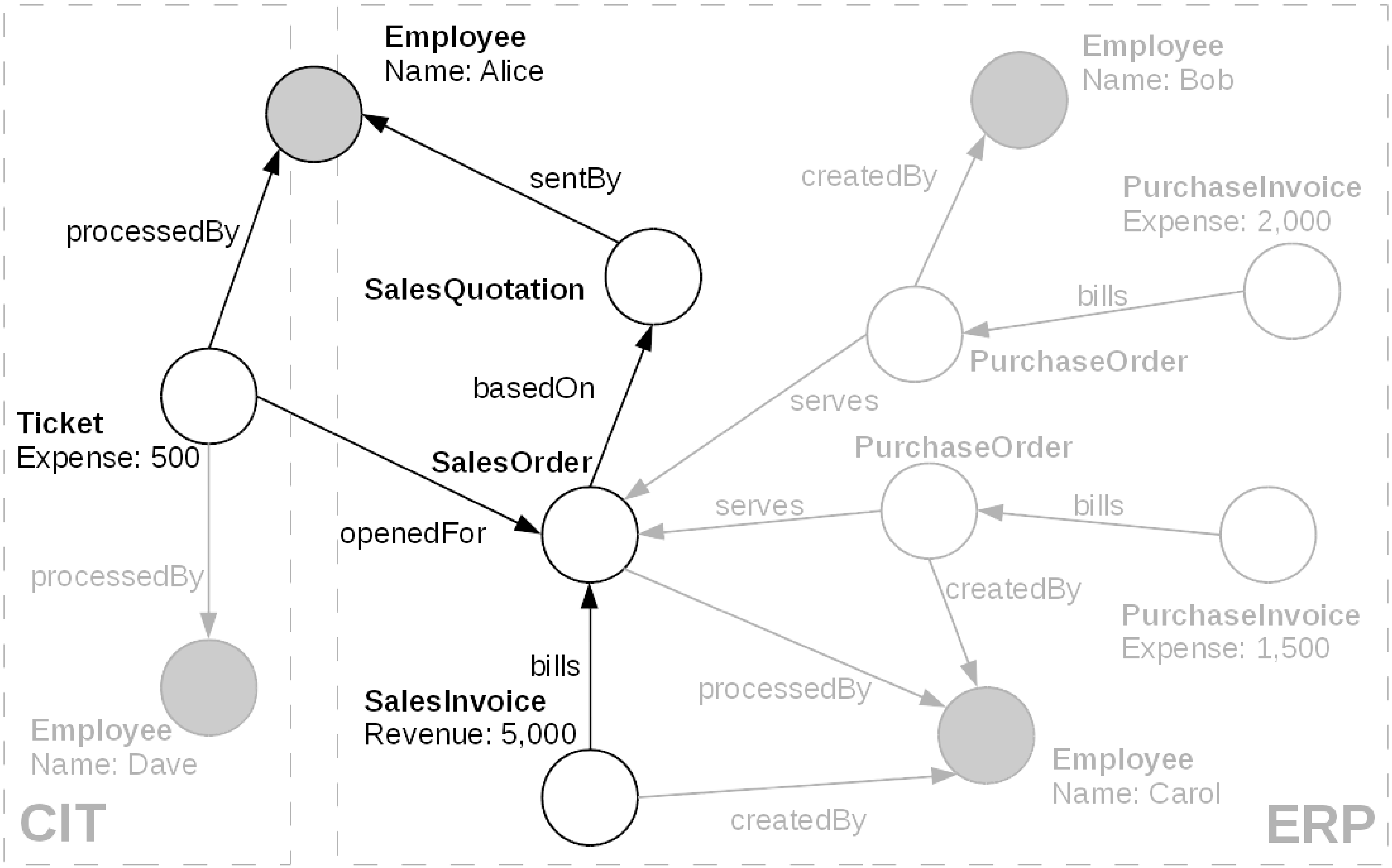
ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



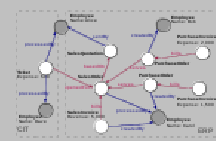
ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS

```
// specific projection

vertexFunc = (Vertex v => new Vertex(
    (v[“IsMasterData”] ? v[“SourceID”] : v[:type]) ,
    {“Result”:v[“Result”]}
)
edgeFunc = (Edge e => new Edge(
    (e[:type]) , {}
)
btgs = btgs.apply( Graph g =>
    g.project( vertexFunc , edgeFunc )
)
```

ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



BTG 1

BTG 2

BTG 3

BTG 4

BTG 5

BTG 6

BTG n

Total Rev.	Expenses	Net Profit
5,000	- 3,000	2,000
9,000	- 3,000	6,000
2,000	- 1,500	500
4,000	- 4,500	- 500
5,000	- 7,000	- 2,000
10,000	- 15,000	-5,000
8,000	- 4,000	4,000

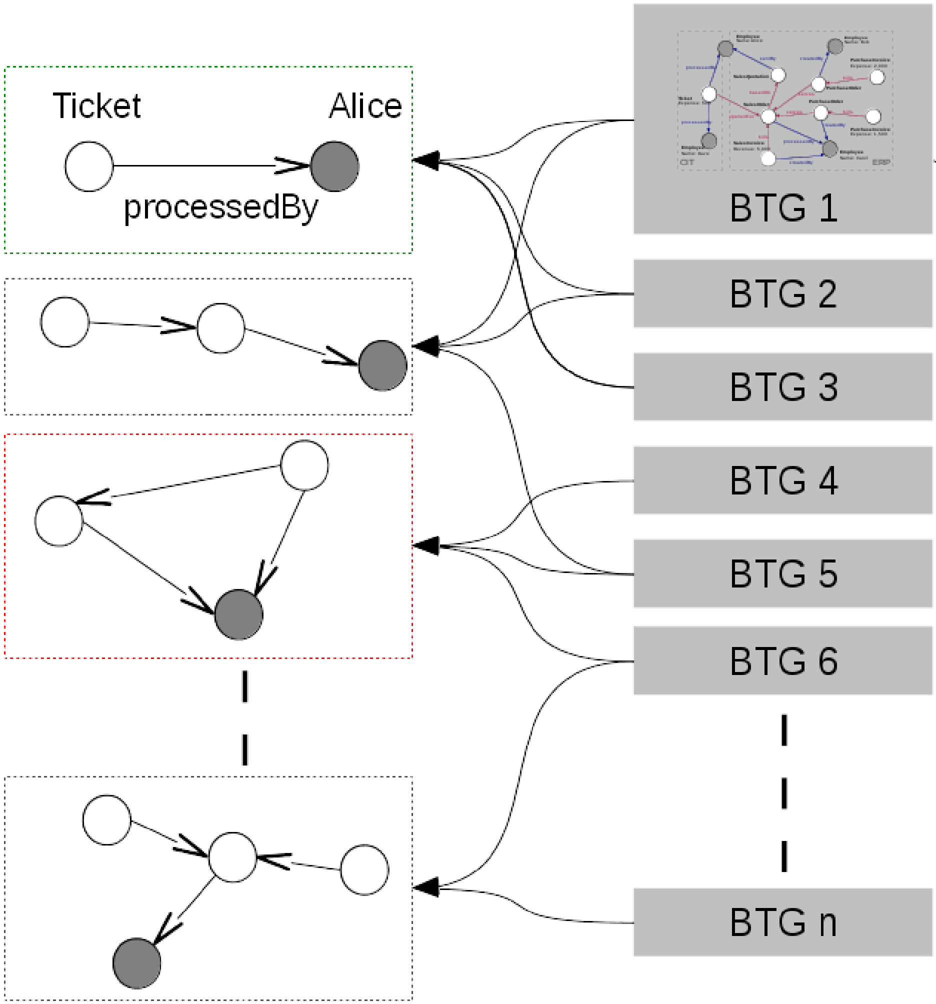
ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS

```
// select profit and loss clusters

proftitBtgs = btgs.select( Graph g => g[“Result”] >= 0 )
lossBtgs = btgs.difference(profitBtgs)
```

ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS



Total Rev.	Expenses	Net Profit
5,000	- 3,000	2,000
9,000	- 3,000	6,000
2,000	- 1,500	500
4,000	- 4,500	- 500
5,000	- 7,000	- 2,000
10,000	- 15,000	-5,000
8,000	- 4,000	4,000

ScaDS DRESDEN LEIPZIG

CLUSTER-CHARACTERISTIC PATTERNS

```
// select profit and loss clusters
proftitBtgs = btgs.select( Graph g => g[“Result”] >= 0 )
lossBtgs = btgs.difference(profitBtgs)

profitFreqPats = proftitBtgs.callForCollection(
    :FrequentSubgraphs , {“Threshold”:0.7}
)
lossFreqPats = lossBtgs.callForCollection(
    :FrequentSubgraphs , {“Threshold”:0.7}
)

// determine cluster characteristic patterns
trivialPats = profitFreqPats.intersect(lossFreqPats)
profitCharPatterns = profitFreqPats.difference(trivialPats)
lossCharPatterns = lossFreqPats.difference(trivialPats)
```

ScaDS DRESDEN LEIPZIG

SUMMARY & ROADMAP: BIIG

- Summary
  - Graph-based business intelligence framework
  - Graph transformations of business information systems
  - Concept of Business Transaction Graphs
- Roadmap
  - WIP: distributed frequent pattern mining
  - Summarization-based Graph OLAP
  - Meaningful result representation
  - Real-world evaluation

ScaDS DRESDEN LEIPZIG

REFERENCES

- **Junghanns, M., Petermann, A., Gomez, K., Peukert, E., Rahm, E.: *GRADOOP - Scalable Graph Data Management and Analytics with Hadoop*. Tech. report, Univ. of Leipzig, June 2015**
- Kolb L., E. Rahm: *Parallel Entity Resolution with Dedoop*. Datenbank-Spektrum 13(1): 23-32 (2013)
- Kolb L., A. Thor, E. Rahm: Dedoop: *Efficient Deduplication with Hadoop*. PVLDB 5(12), 2012
- Kolb L., A. Thor, E. Rahm: *Load Balancing for MapReduce-based Entity Resolution*. ICDE 2012: 618-629
- Kolb L., Z. Sehili, E. Rahm: *Iterative Computation of Connected Graph Components with MapReduce*. Datenbank-Spektrum 14(2): 107-117 (2014)
- **Petermann A., M. Junghanns, R. Müller, E. Rahm: *BIIIG : Enabling Business Intelligence with Integrated Instance Graphs*. Proc. 5th Int. Workshop on Graph Data Management (GDM 2014)**
- **Petermann A., 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
- Jindal, A. et.al.: *Vertexica: your relational friend for graph analytics!*. PVLDB 7(13), 2014
- Rudolf, M. et.al.: *The Graph Story of the SAP HANA Database.* BTW, 2013

ScaDS
DRESDEN LEIPZIG

Thank you!

www.gradoop.org
www.biiig.org