SCALABALE GRAPH ANALYTICS WITH GRADOOP

ERHARD RAHM,
MARTIN JUNGHANNS, ANDRE PETERMANN, KEVIN GOMEZ, ERIC PEUKERT

UNIVERSITÄT LEIPZIG

GERMAN CENTERS FOR BIG DATA

Two Centers of Excellence for Big Data in Germany

- ScaDS Dresden/Leipzig
- Berlin Big Data Center (BBDC)

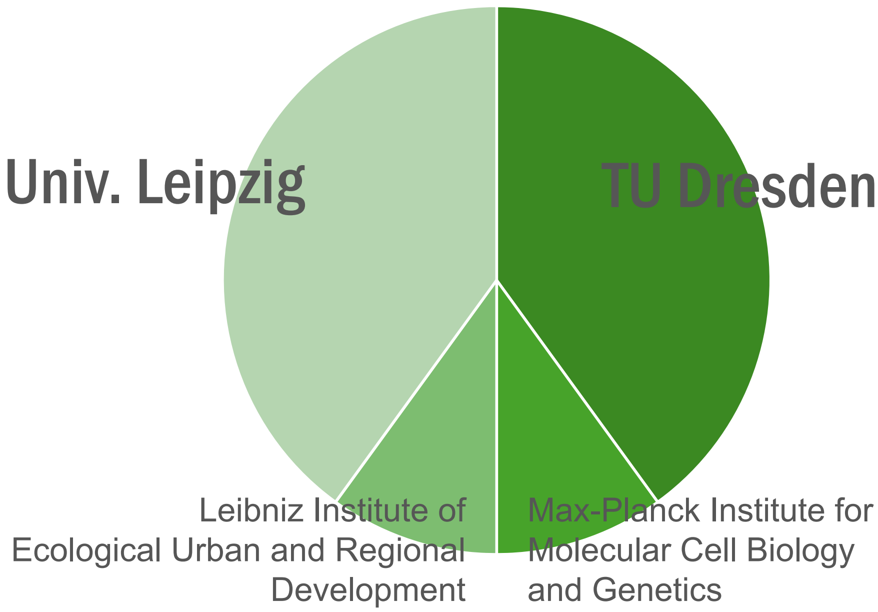
ScaDS Dresden/Leipzig (Competence Center for Scalable Data Services and Solutions Dresden/Leipzig)

- scientific coordinators: Nagel (TUD), Rahm (UL)
- start: Oct. 2014
- duration: 4 years (option for 3 more years)
- initial funding: ca. 5.6 Mio. Euro

ScaDS DRESDEN LEIPZIG GOALS

- Bundling and advancement of existing expertise on Big Data
- Development of Big Data Services and Solutions
- Big Data Innovations

ScaDS DRESDEN LEIPZIG FUNDED INSTITUTES



ASSOCIATED PARTNERS

- Avantgarde-Labs GmbH
- Data Virtuality GmbH
- E-Commerce Genossenschaft e. G.
- European Centre for Emerging Materials and Processes Dresden
- Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme
- Fraunhofer-Institut für Werkstoff- und Strahltechnik
- GISA GmbH
- Helmholtz-Zentrum Dresden - Rossendorf
- Hochschule für Telekommunikation Leipzig
- Institut für Angewandte Informatik e. V.
- Landesamt für Umwelt, Landwirtschaft und Geologie
- Netzwerk Logistik Leipzig-Halle e. V.
- Sächsische Landesbibliothek – Staats- und Universitätsbibliothek Dresden
- Scionics Computer Innovation GmbH
- Technische Universität Chemnitz
- Universitätsklinikum Carl Gustav Carus

STRUCTURE OF THE CENTER

Life sciences

Material and Engineering sciences

Environmental / Geo sciences

Digital Humanities

Business Data

Service center

Big Data Life Cycle Management and Workflows

Data Quality / Data Integration

Knowledge Extraktion

Visual Analytics

Efficient Big Data Architectures

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

RESEARCH PARTNERS

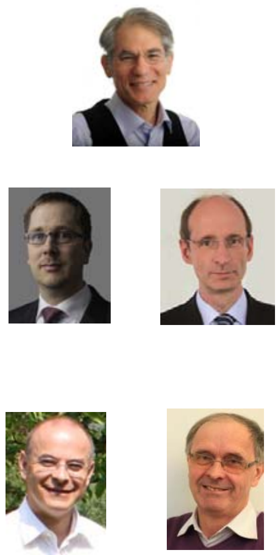
- Data-intensive computing W.E. Nagel
- Data quality / Data integration E. Rahm
- Databases W. Lehner, E. Rahm
- Knowledge extraction/Data mining
  C. Rother, P. Stadler, G. Heyer
- Visualization
  S. Gumhold, G. Scheuermann
- Service Engineering, Infrastructure
  K.-P. Fähnrich, W.E. Nagel, M. Bogdan

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

APPLICATION COORDINATORS

- Life sciences G. Myers
- Material / Engineering sciences M. Gude
- Environmental / Geo sciences J. Schanze
- Digital Humanities G. Heyer
- Business Data B. Franczyk



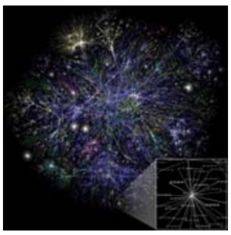
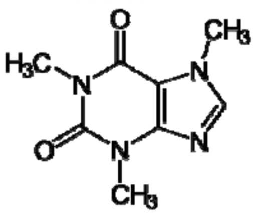
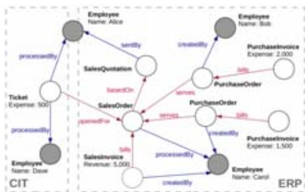
UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG AGENDA

- ScaDS Dresden/Leipzig
- Big Graph Data
  - Graph-based Business Intelligence with BIIG
  - basic approaches for graph data management/analysis
- GraDoop: Hadoop-based graph data management and analysis
  - Gradoop characteristics and architecture
  - Extended Property Graph Data Model (EPGM) / Graph operators
  - Distributed graph store
  - Sample workflows
- Summary and outlook

UNIVERSITÄT LEIPZIG

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. 31 billion triples

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

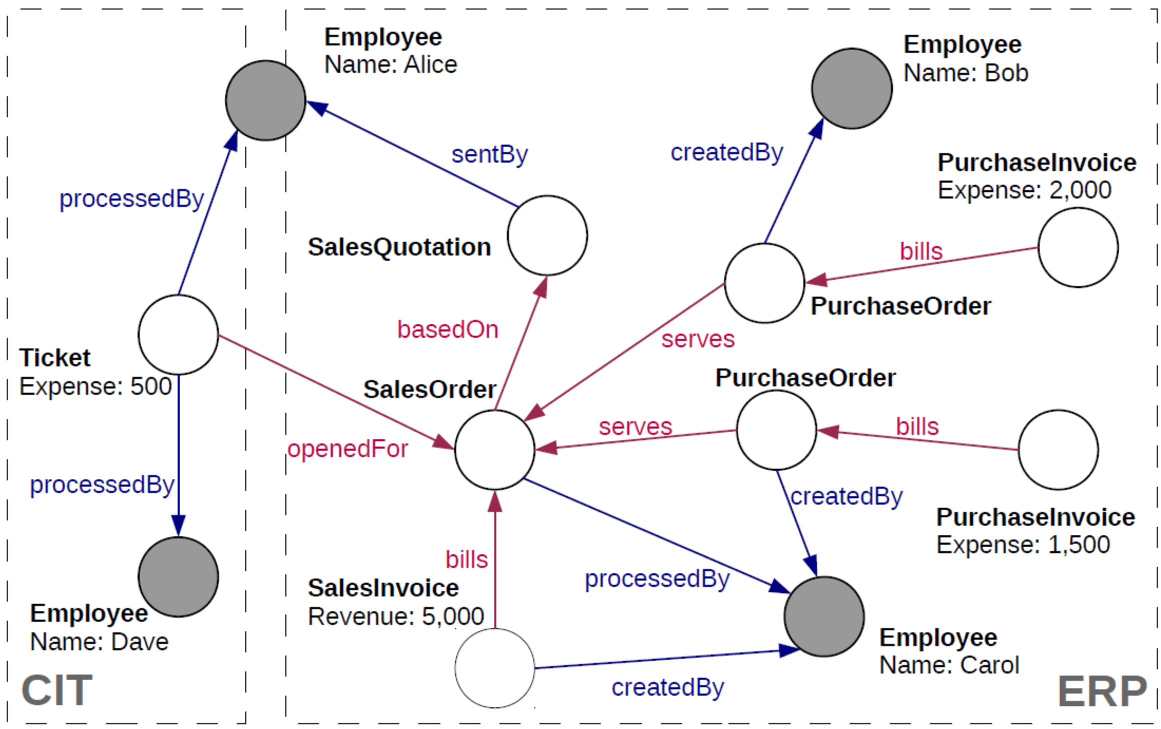
USE CASE: GRAPH-BASED BUSINESS INTELLIGENCE

- 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 approach (BIIIG)
  - integrate data sources within an instance graph by preserving original relationships between data objects (transactional and master data)
  - determine subgraphs (business transaction graphs) related to business activities
  - analyze subgraphs or entire graphs with aggregation queries, mining relationship patterns, etc.

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

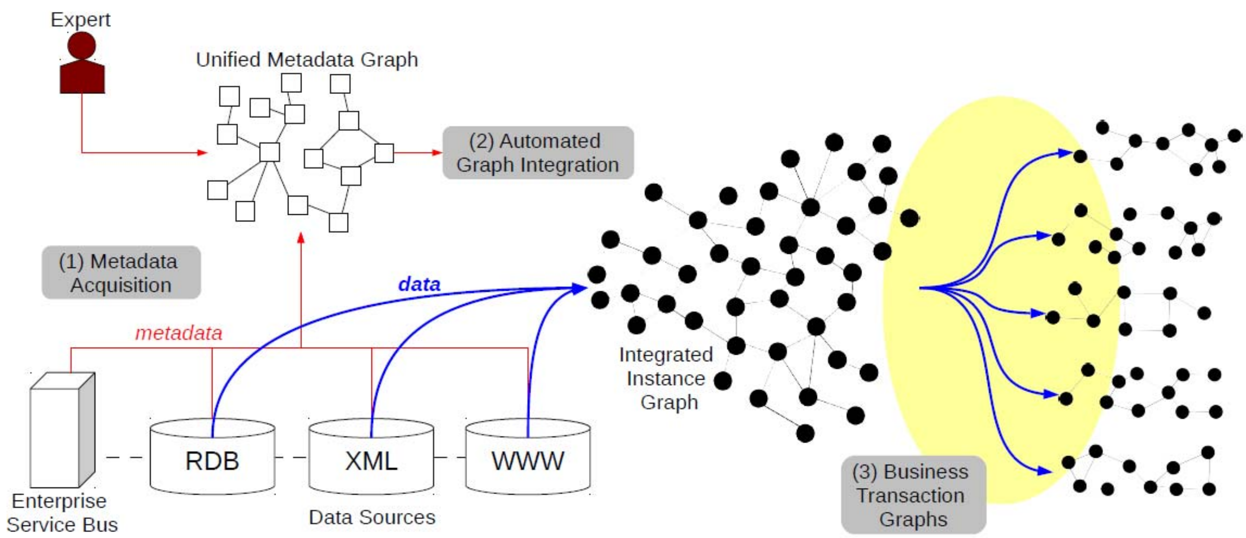
SAMPLE GRAPH



ScaDS DRESDEN LEIPZIG

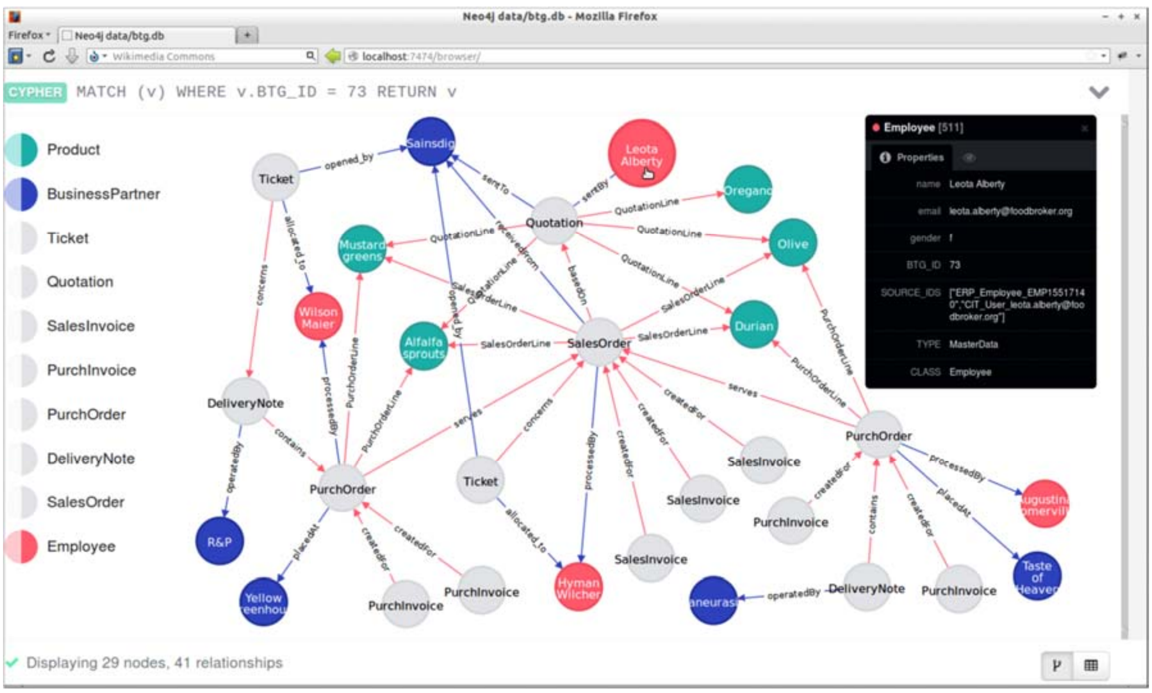
BIIIG DATA INTEGRATION AND ANALYSIS WORKFLOW

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



ScaDS DRESDEN LEIPZIG

SCREENSHOT FOR NEO4J IMPLEMENTATION



ScaDS DRESDEN LEIPZIG

GRAPH DATA MANAGEMENT

- Relational database systems
  - store vertices and edges in tables
  - utilize indexes, column stores, etc.
  - could be used as a basis (graph store) to implement graph operators
- 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

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

GRAPH DATA MANAGEMENT (2)

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

WHAT'S MISSING?

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

AGENDA

- ScaDS Dresden/Leipzig
- Big Graph Data
  - Graph-based Business Intelligence with BIIG
  - basic approaches for graph data management/analysis
- GraDoop: Hadoop-based graph data management and analysis
  - Gradoop characteristics and architecture
  - Extended Property Graph Data Model (EPGM) / Graph operators
  - Distributed graph store
  - Sample workflows
- Summary and outlook

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

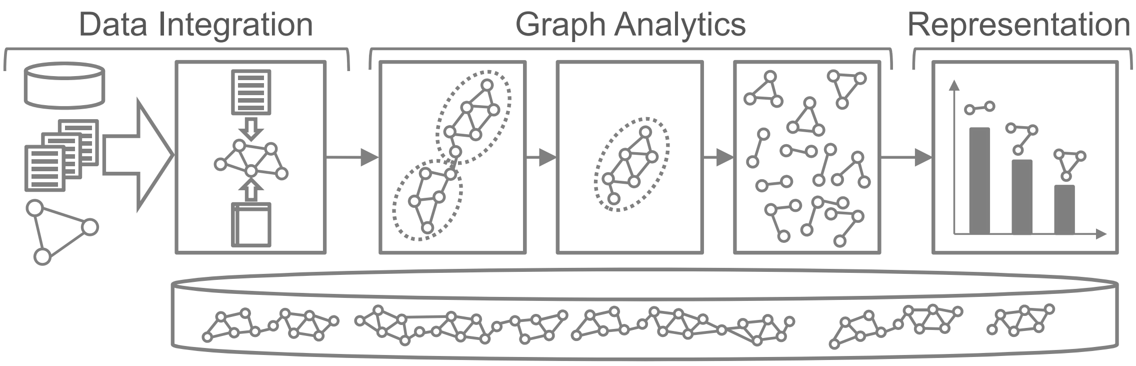
GRADOOP CHARACTERISTICS

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

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

END-TO-END GRAPH ANALYTICS

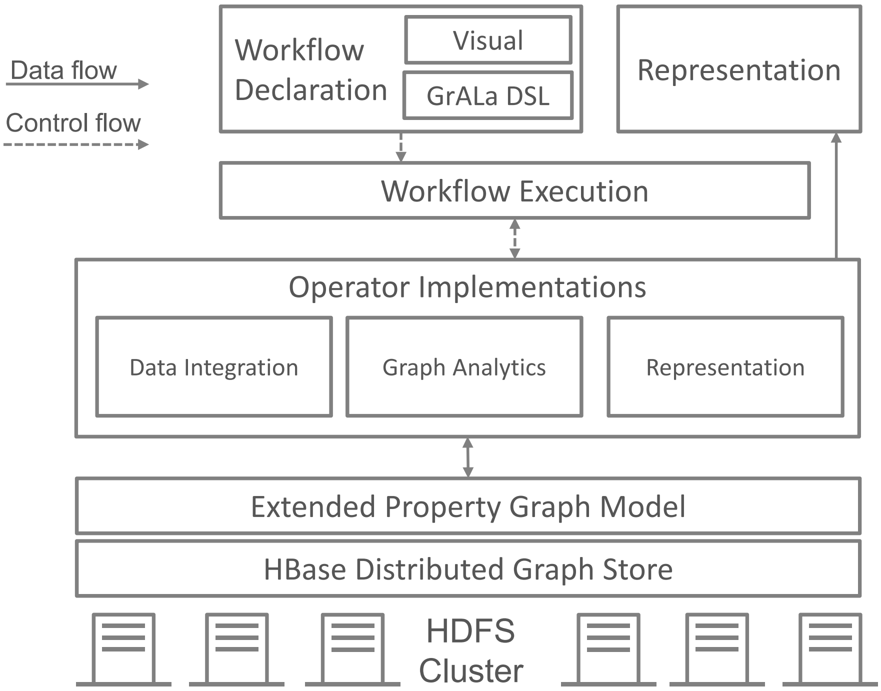


- **Integrate data** from one or more sources into a dedicated **graph storage** with **common graph data model**
- Definition of **analytical workflows** from **operator algebra**
- Result representation in **meaningful way**

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

HIGH LEVEL ARCHITECTURE



UNIVERSITÄT LEIPZIG

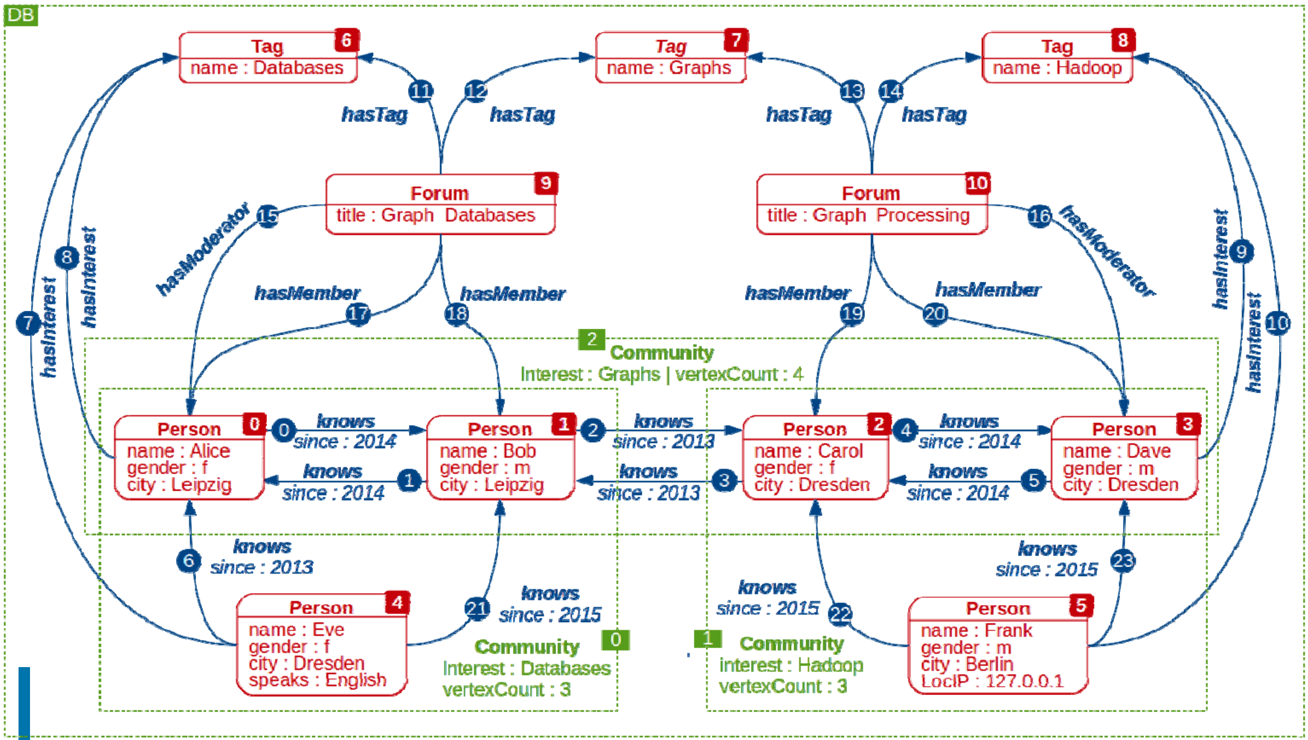
ScaDS DRESDEN LEIPZIG

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



$$DB_{EPGM} = \langle \mathcal{V}, \mathcal{E}, \mathcal{G}, \mathrm{T}, \tau, K, A, \kappa \rangle$$

ScaDS DRESDEN LEIPZIG

EXTENDED PROPERTY GRAPH MODEL

Vertex space

$\mathcal{V} = \{v_0, .., v_i\}$

Edge space

$\mathcal{E} = \{e_0, .., e_k\}$

$e_k = \langle v_i, v_j \mid v_i, v_j \in \mathcal{V} \rangle$

Logical graphs

$\mathcal{G} = \{G_{DB}, G_0, .., G_m\}$

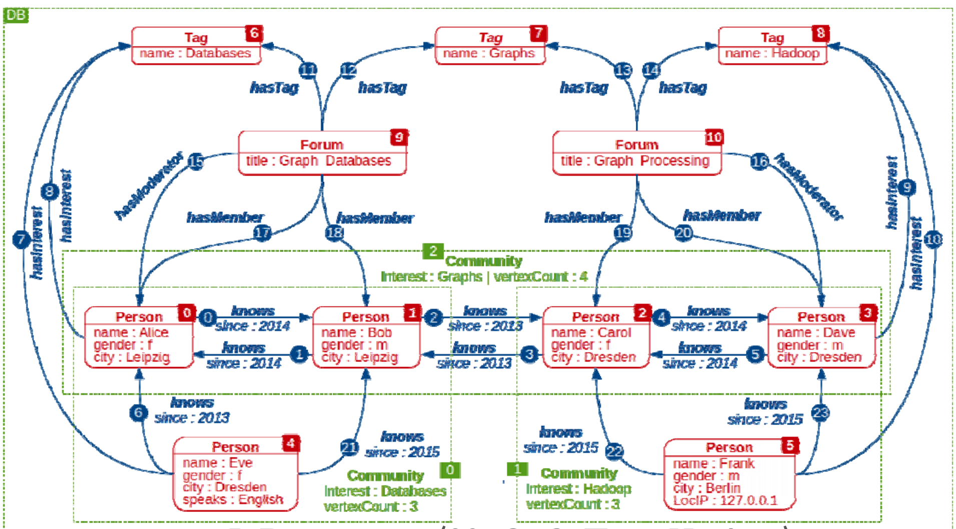
$G_m = \langle V, E \mid V \subseteq \mathcal{V} \wedge E \subseteq \mathcal{E} \rangle$

Type labels

$\tau : (\mathcal{V} \cup \mathcal{E} \cup \mathcal{G}) \rightarrow \mathrm{T}$

Properties

$\kappa : (\mathcal{V} \cup \mathcal{E} \cup \mathcal{G}) \times K \rightarrow A$



$$DB_{EPGM} = \langle \mathcal{V}, \mathcal{E}, \mathcal{G}, T, \tau, K, A, \kappa \rangle$$

ScaDS DRESDEN LEIPZIG

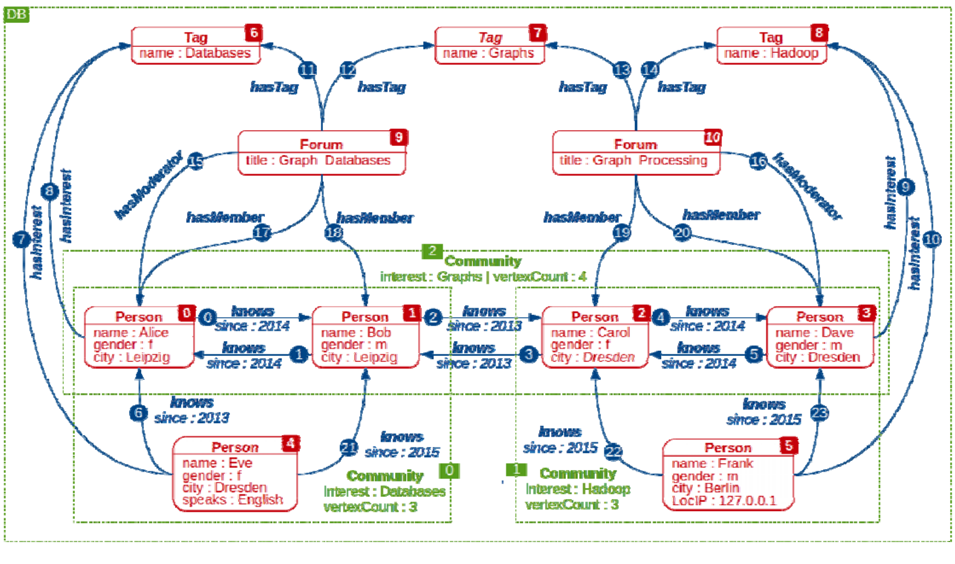
GRAPH OPERATORS

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

ScaDS DRESDEN LEIPZIG

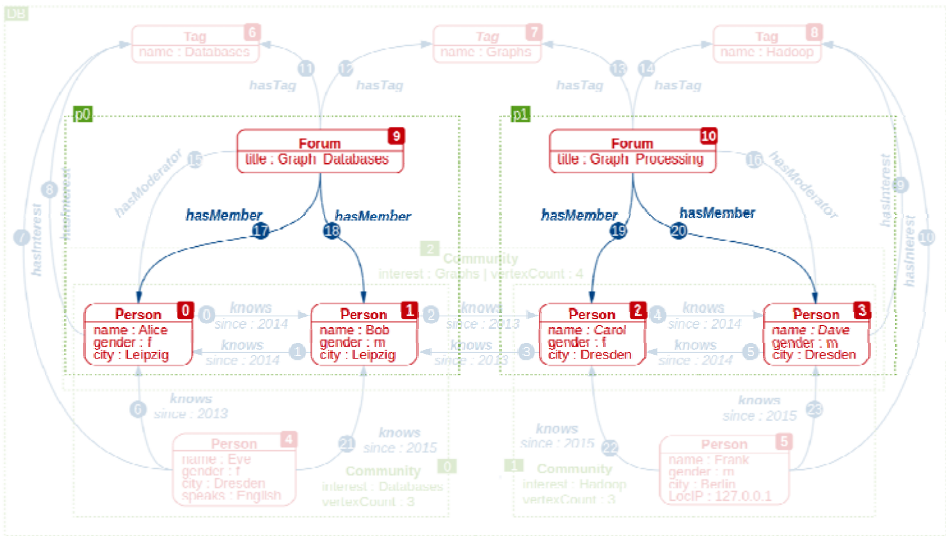
PATTERN MATCHING


```
1: pattern = new Graph("(a)<-d-(b)-e->(c)")
2: predicate = (Graph g => g.V[$a][:type] == "Person" &&
      g.V[$b][:type] == "Forum" &&
      g.V[$c][:type] == "Person" &&
      g.E[$d][:type] == "hasMember" &&
      g.E[$e][:type] == "hasMember")
3: result = db.match(pattern, predicate)
```



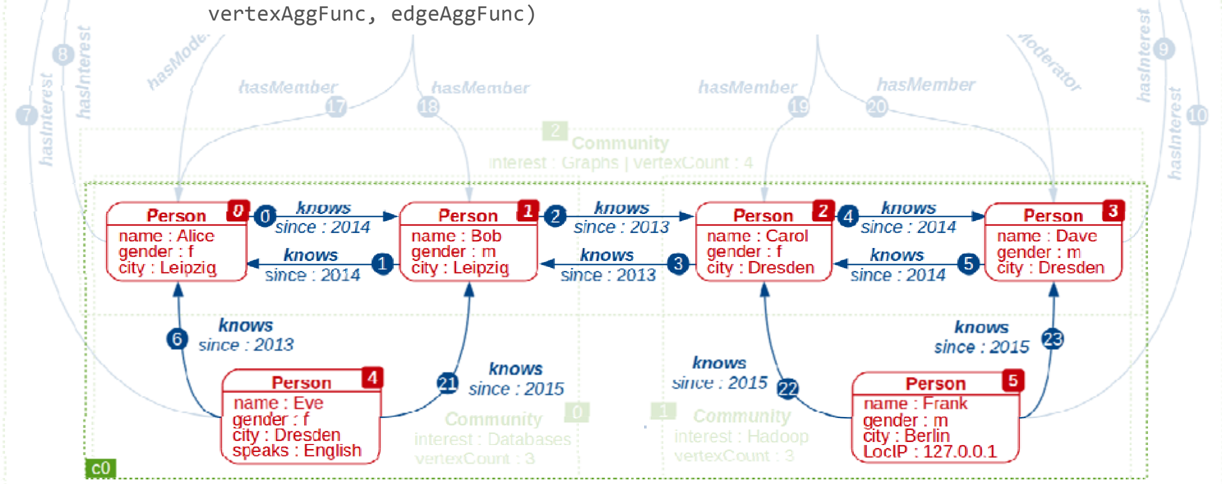
PATTERN MATCHING


```
pattern = new Graph("(a)<-d-(b)-e->(c)")
predicate = (Graph g => g.V[$a][:type] == "Person" &&
    g.V[$b][:type] == "Forum" &&
    g.V[$c][:type] == "Person" &&
    g.E[$d][:type] == "hasMember" &&
    g.E[$e][:type] == "hasMember")
result = db.match(pattern, predicate)
```



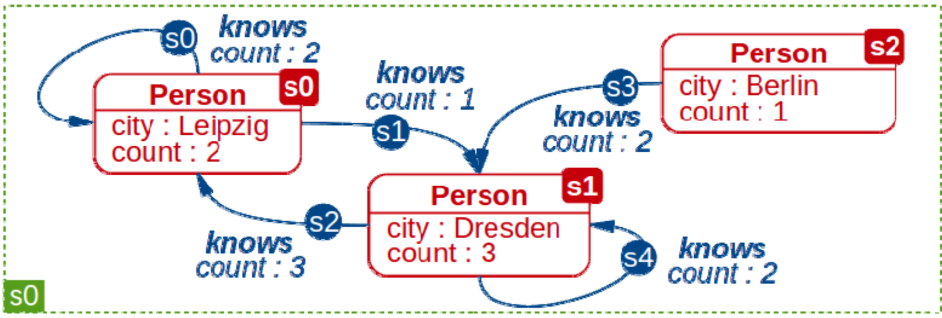
SUMMARIZATION

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



SUMMARIZATION

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



COLLECTION OPERATORS

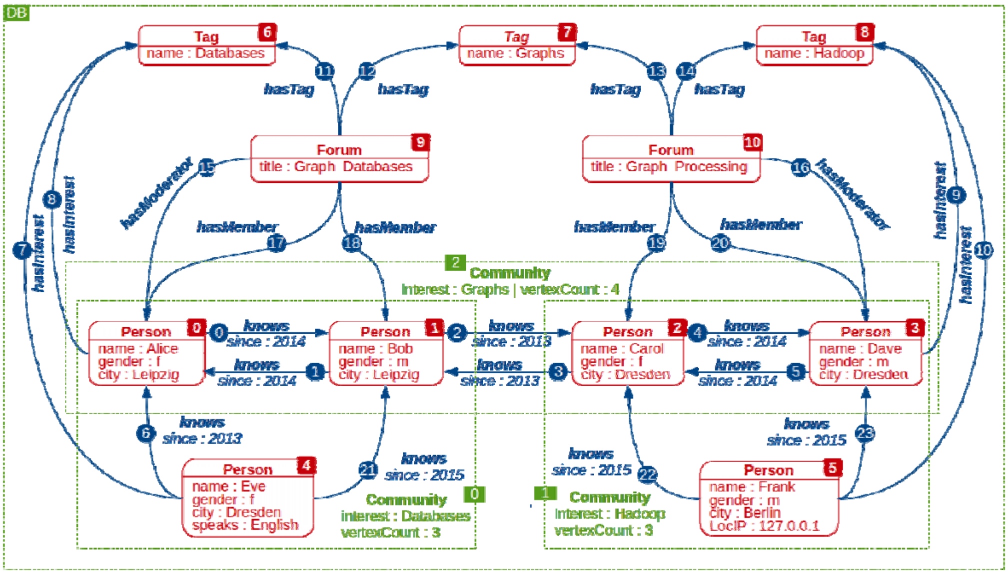
Operator	Definition	GrALa notation
collection		
Selection	$\sigma_\varphi : \mathcal{G}^n \to \mathcal{G}^n$	collection.select(predicate) : Collection
Distinct	$\delta : \mathcal{G}^n \to \mathcal{G}^n$	collection.distinct() : Collection
Sort by	$\xi_{k,d} : \mathcal{G}^n \to \mathcal{G}^n$	collection.sortBy(key,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

UNIVERSITÄT LEIPZIG

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)
```

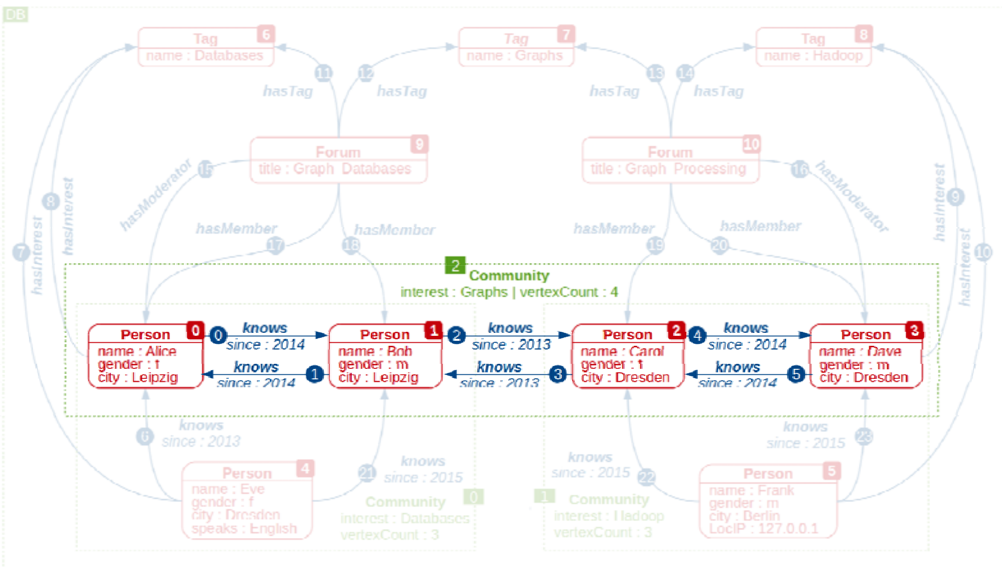


UNIVERSITÄT LEIPZIG

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)
```

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

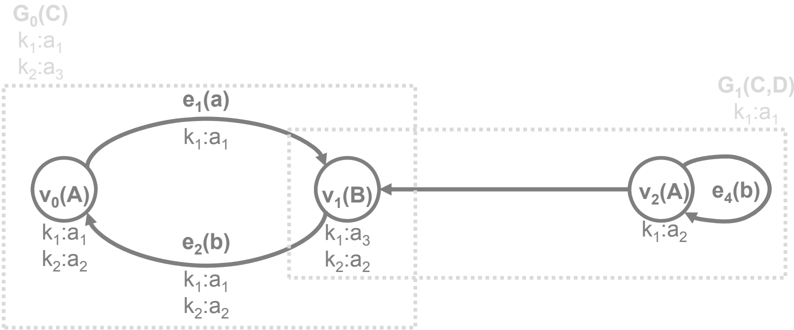


Table ´vertices´

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

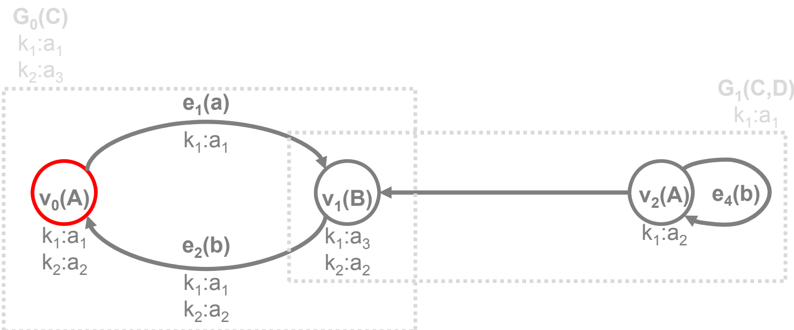


Table ´vertices´

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

VERTEX TABLE

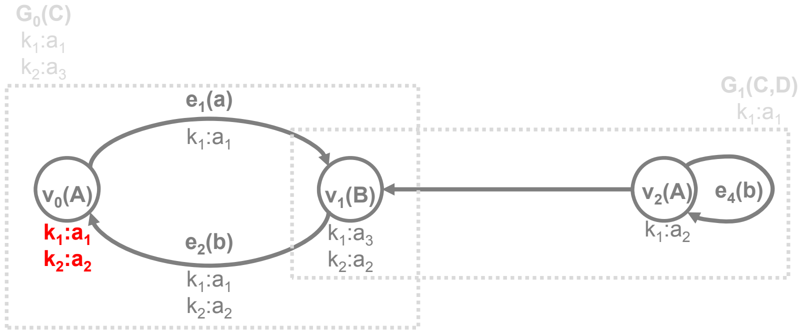


Table ´vertices´

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

VERTEX TABLE

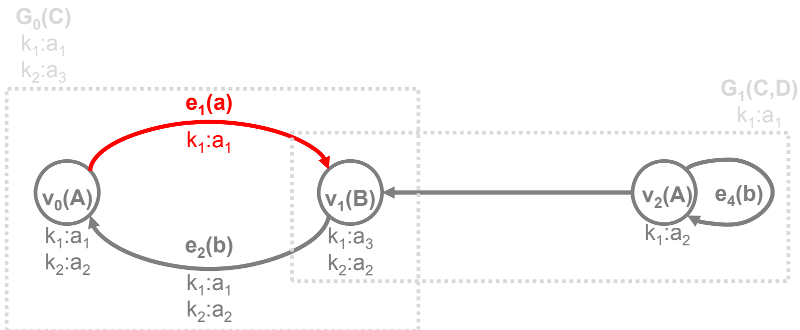


Table ´vertices´

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

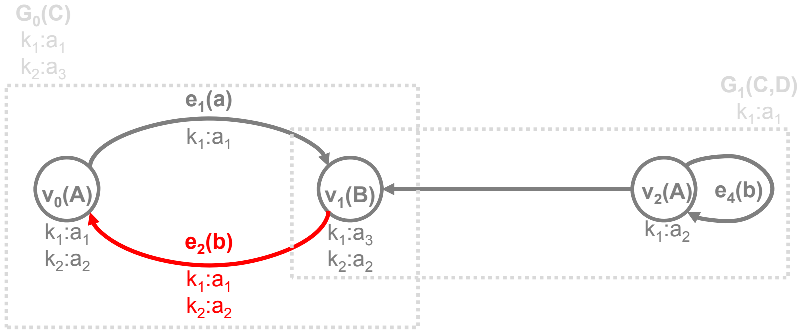
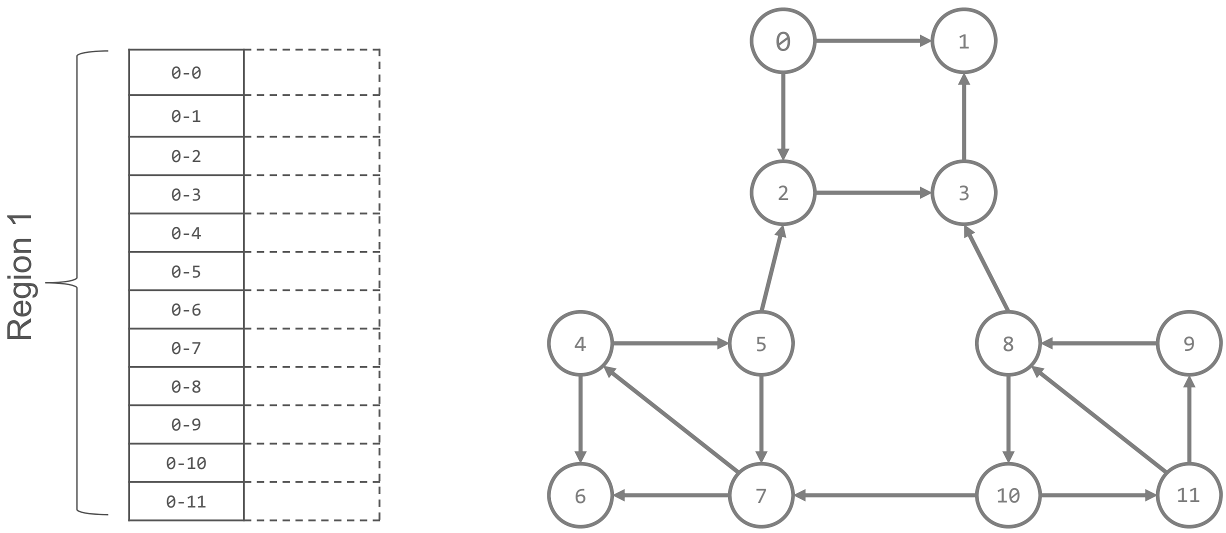


Table ´vertices´

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

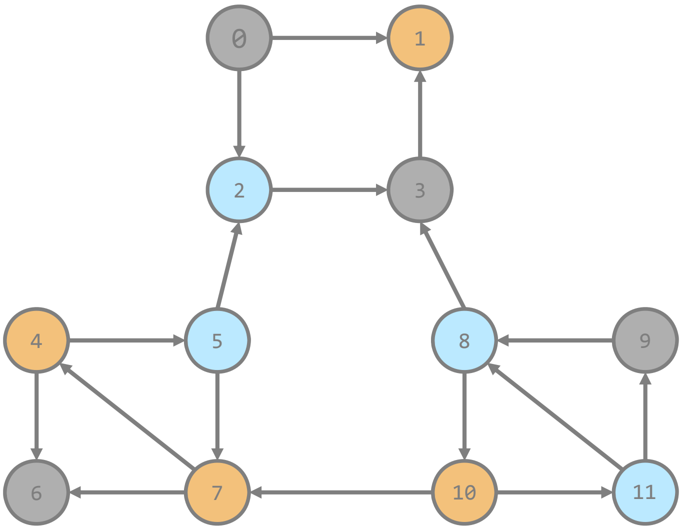
PARTITIONED VERTEX TABLE



UNIVERSITÄT LEIPZIG

PARTITIONED VERTEX TABLE

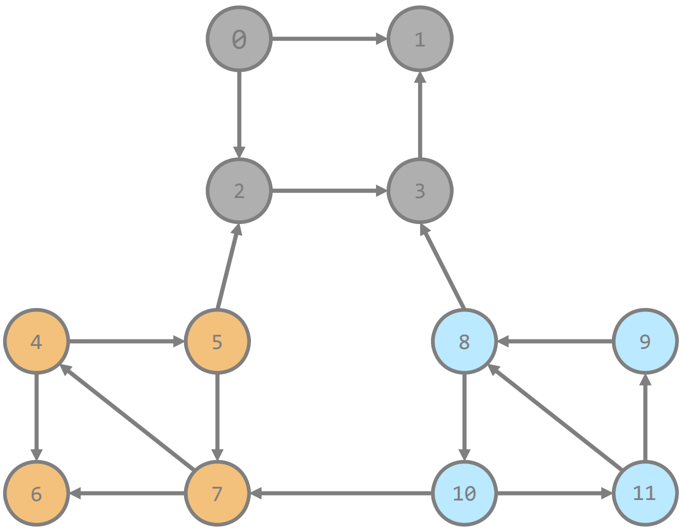
Region	Vertex	
Region 1	0-0	
	0-3	
	0-6	
	0-9	
Region 2	1-1	
	1-4	
	1-7	
	1-10	
Region 3	2-2	
	2-5	
	2-8	
	2-11	



UNIVERSITÄT LEIPZIG

PARTITIONED VERTEX TABLE

Region	Vertex	
Region 1	0-0	
	0-1	
	0-2	
	0-3	
Region 2	1-4	
	1-5	
	1-6	
	1-7	
Region 3	2-8	
	2-9	
	2-10	
	2-11	



UNIVERSITÄT LEIPZIG

GRAPH TABLE

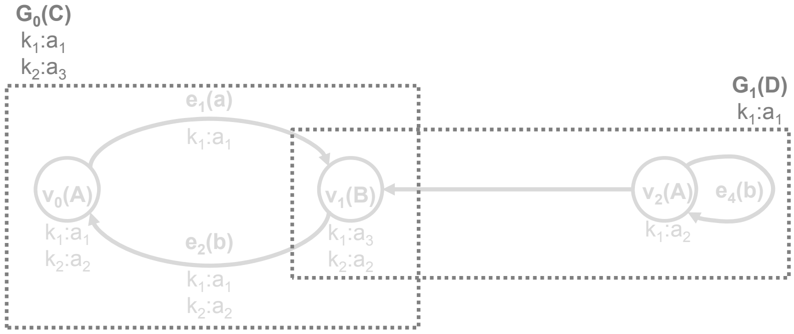


Table ´graphs´

0	meta		properties		edges	
	type	vertices	k_1	k_2	$0-0$	$0-1$
	C	$[0-0,0-1]$	$\langle t_1, a_1\rangle$	$\langle t_3, a_3\rangle$	$[\langle a, 0-1,0\rangle]$	$[\langle b, 0-0,0\rangle]$
1	meta		properties		edges	
	type	graphs	k_1		$0-1$	$0-2$
	D	$[0-1,0-2]$	$\langle t_1, a_1\rangle$		[]	$[\langle a, 0-1,0\rangle . \langle b, 0-2,1\rangle]$

UNIVERSITÄT LEIPZIG

GRAPH TABLE

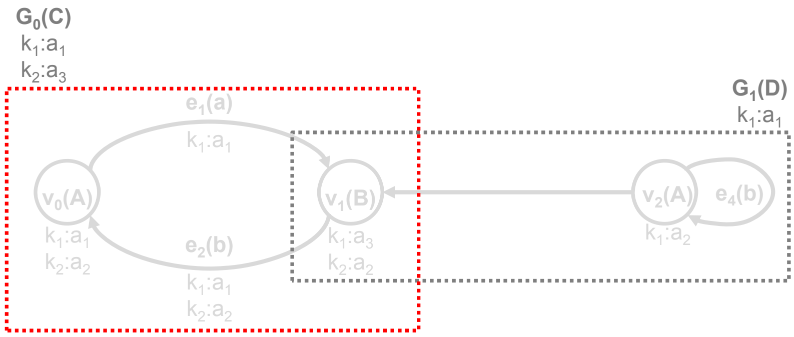


Table ´graphs´

0	meta		properties		edges	
	type	vertices	k_1	k_2	$0-0$	$0-1$
	C	$[0-0,0-1]$	$\langle t_1, a_1\rangle$	$\langle t_3, a_3\rangle$	$[\langle a, 0-1,0\rangle]$	$[\langle b, 0-0,0\rangle]$
1	meta		properties		edges	
	type	graphs	k_1		$0-1$	$0-2$
	D	$[0-1,0-2]$	$\langle t_1, a_1\rangle$		[]	$[\langle a, 0-1,0\rangle . \langle b, 0-2,1\rangle]$

GRAPH STORE

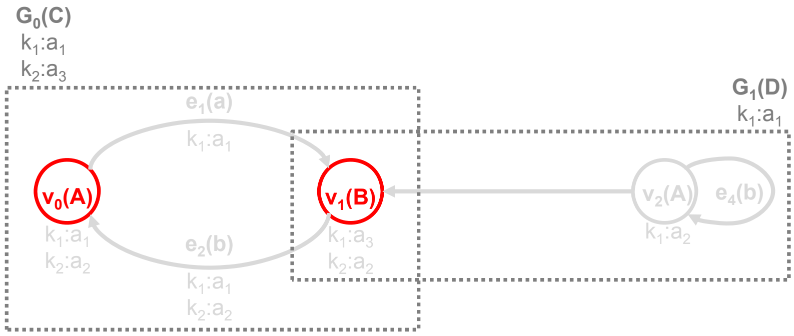


Table ´graphs´

0	meta		properties		edges	
	type	**vertices**	k_1	k_2	$\mathbf{0-0}$	$\mathbf{0-1}$
	C	$[\mathbf{0-0,0-1}]$	$\langle t_1, a_1\rangle$	$\langle t_3, a_3\rangle$	$[\langle a, 0-1{,}0\rangle]$	$[\langle b, 0-0{,}0\rangle]$
1	meta		properties		edges	
	type	graphs	k_1		$0-1$	$0-2$
	D	$[0-1{,}0-2]$	$\langle t_1, a_1\rangle$		$[]$	$[\langle a, 0-1{,}0\rangle, \langle b, 0-2{,}1\rangle]$

GRAPH TABLE

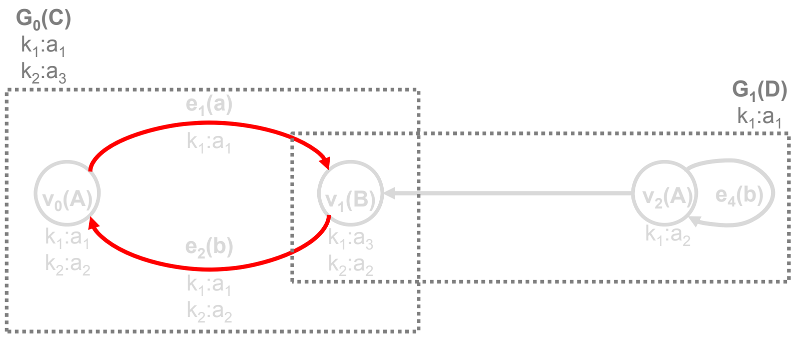


Table ´graphs´

0	meta		properties		edges	
	type	vertices	k_1	k_2	$0-0$	$0-1$
	C	$[0-0{,}0-1]$	$\langle t_1, a_1\rangle$	$\langle t_3, a_3\rangle$	$[\langle \mathbf{a, 0-1{,}0}\rangle]$	$[\langle \mathbf{b, 0-0{,}0}\rangle]$
1	meta		properties		edges	
	type	graphs	k_1		$0-1$	$0-2$
	D	$[0-1{,}0-2]$	$\langle t_1, a_1\rangle$		$[]$	$[\langle a, 0-1{,}0\rangle, \langle b, 0-2{,}1\rangle]$

ScaDS DRESDEN LEIPZIG

EXAMPLE GRALA WORKFLOWS

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

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

GRALA EXAMPLE : SUMMARIZED COMMUNITIES

```
// define pattern to extract persons and their "knows" relations
1: pattern = new Graph( "(a)-c->(b)" )
2: predicate = ( Graph g =>
     g.V[$a][:type] == "Person" &&
     g.V[$b][:type] == "Person" &&
     g.E[$c][:type] == "knows")
// find all matches inside the database
3: friendships = db.match( pattern , predicate )
// combine all matches to a single graph
4: knowsGraph = friendships.reduce( Graph g, Graph f => g.combine(f) )
// remove properties
5: knowsGraph = knowsGraph.project( Vertex v =>
     new Vertex(v[:type], {}), new Edge(e[:type], {}))
// extract communities, store community at vertex property "community"
6: knowsGraph = knowsGraph.callForGraph(
     :CommunityDetectionAlgorithm , {"propertyKey":"community"})
// summarize vertices based on their community
// count edges inside and between communities
7: summarizedCommunities = knowsGraph.summarize(
     {"community"},
     ((Vertex vSum, Set vertices) => vSum["count"] = |vertices|),
     {},
     ((Edge eSum, Set edges) => eSum["count"] = |edges|))
```

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

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

AGENDA

- ScaDS Dresden/Leipzig
- Big Graph Data
  - Graph-based Business Intelligence with BIIG
  - basic approaches for graph data management/analysis
- GraDoop: Hadoop-based graph data management and analysis
  - Gradoop characteristics and architecture
  - Extended Property Graph Data Model (EPGM) / Graph operators
  - Distributed graph store
  - Sample workflows
- Summary and outlook

ScaDS DRESDEN LEIPZIG

SUMMARY

- ScaDS Dresden/Leipzig
  - Research focus on data integration, knowledge extraction, visual analytics
  - broad application areas (scientific + business-related)
- Big Graph Data
  - high potential of graph analytics even for business data (BIIIG)
- GraDoop
  - end-to-end framework for graph data management and analytics
  - leverages Hadoop ecosystem including graph processing systems
  - extended property graph model (EPGM) with powerful operators
  - Gradoop store based on Hbase
  - initial implementation running

ScaDS DRESDEN LEIPZIG

GRADOOP OUTLOOK

- complete processing framework
  - implementation for all operators
  - implement more mining algorithms on EPGM
  - workflow execution layer
  - visualization
- automatic optimization of analysis workflows
- optimized graph partitioning approaches
- graph-based data integration
- ...

GRADOOP TEAM

- Graph Store / Workflow Execution / Graph Pattern Matching: Martin Junghanns (wiss. MA)
- BIIIG / Workflow Execution / Frequent Subgraph Mining: Andre Petermann (wiss. MA)
- RDF Graph Analytics: Markus Nentwig (wiss. MA)
- Gradoop + Flink: Niklas Teichmann (SHK)
- Graph Partitioning: Kevin Gómez (SHK/BA)
- Visual Workflow Definition: Simon Chill (MA)
- Graph Pattern Matching: Andreas Krause (MA)
- Frequent Subgraph Mining: Thomas Döring (MA)
- Graph Visualization: Ngoc Ha Tran (MA)

UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

REFERENCES

- Junghanns, M., Petermann, A., Gomez, K., Rahm, E.: *GRADOOP - Scalable Graph Data Management and Analytics with Hadoop*. Tech. report, Univ. of Leipzig, June 2015
- L. Kolb, E. Rahm: *Parallel Entity Resolution with Dedoop*. Datenbank-Spektrum 13(1): 23-32 (2013)
- L. Kolb, A. Thor, E. Rahm: Dedoop: *Efficient Deduplication with Hadoop*. PVLDB 5(12), 2012
- L. Kolb, A. Thor, E. Rahm: *Load Balancing for MapReduce-based Entity Resolution*. ICDE 2012: 618-629
- L. Kolb, Z. Sehili, E. Rahm: *Iterative Computation of Connected Graph Components with MapReduce*. Datenbank-Spektrum 14(2): 107-117 (2014)
- 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
- E. Rahm, W.E. Nagel: *ScaDS Dresden/Leipzig: Ein serviceorientiertes Kompetenzzentrum für Big Data*. Proc. GI-Jahrestagung 2014: 717