UNIVERSITÄT LEIPZIG

ScaDS DRESDEN LEIPZIG

GRADOOP: Scalable Graph Analytics with Apache Flink

Martin Junghanns

@kc1s

Leipzig University

FOSDEM 16 .org

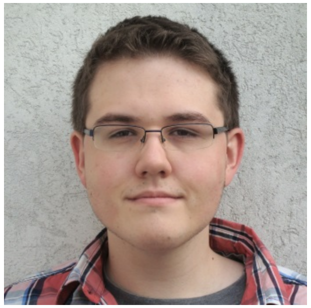
Brussels 30 & 31 January

About the speaker and the team

Martin, PhD Student

André, PhD Student

Prof. Dr. Erhard Rahm
Database Chair

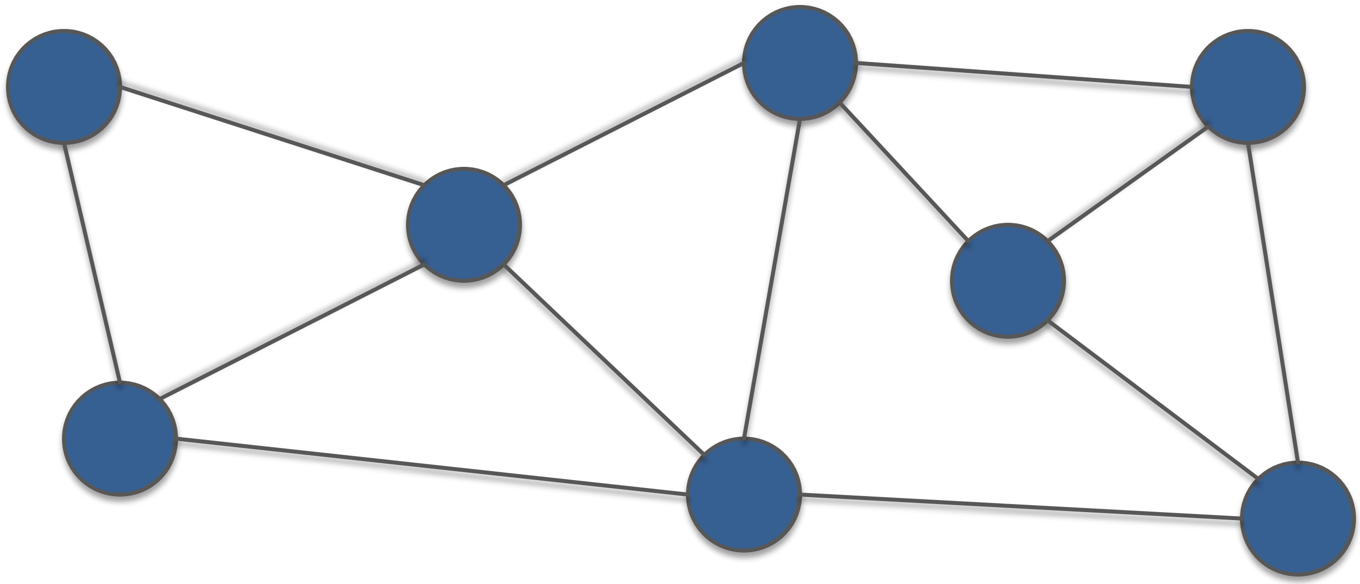


Niklas, M.Sc. Student

Kevin, M.Sc. Student

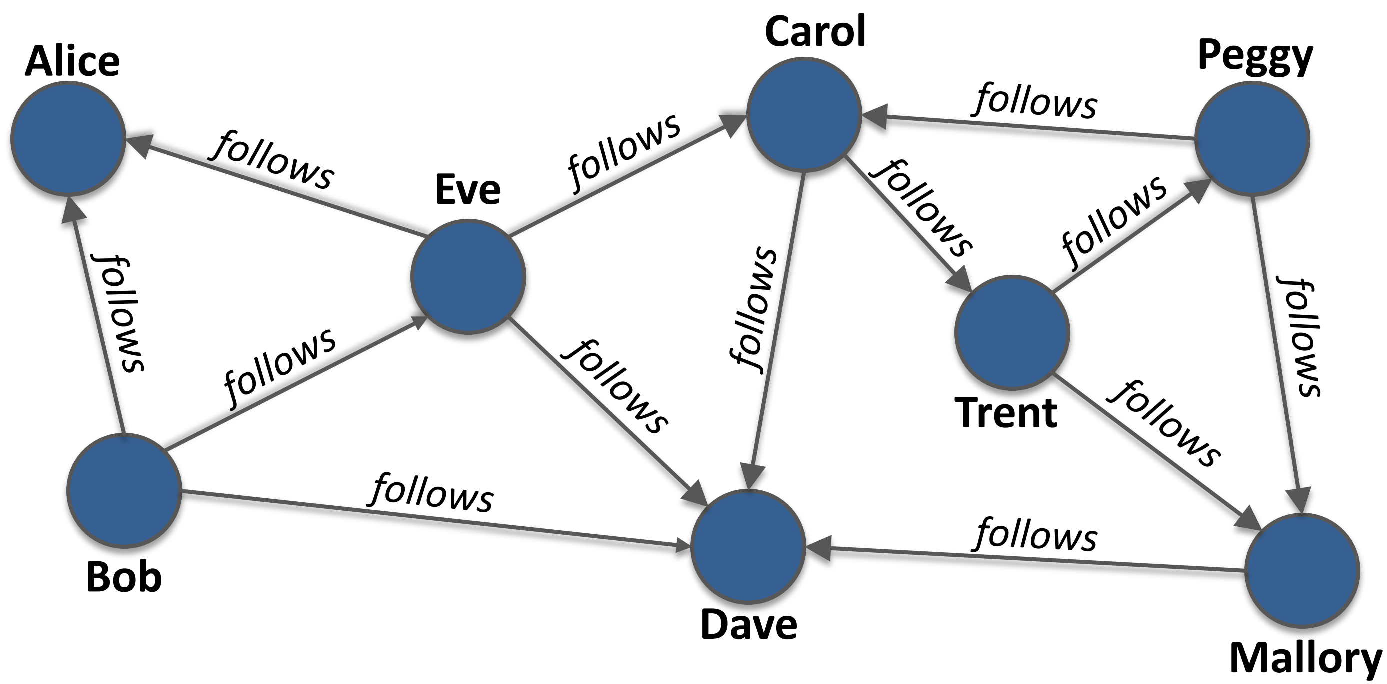
Motivation

“Graphs are everywhere”



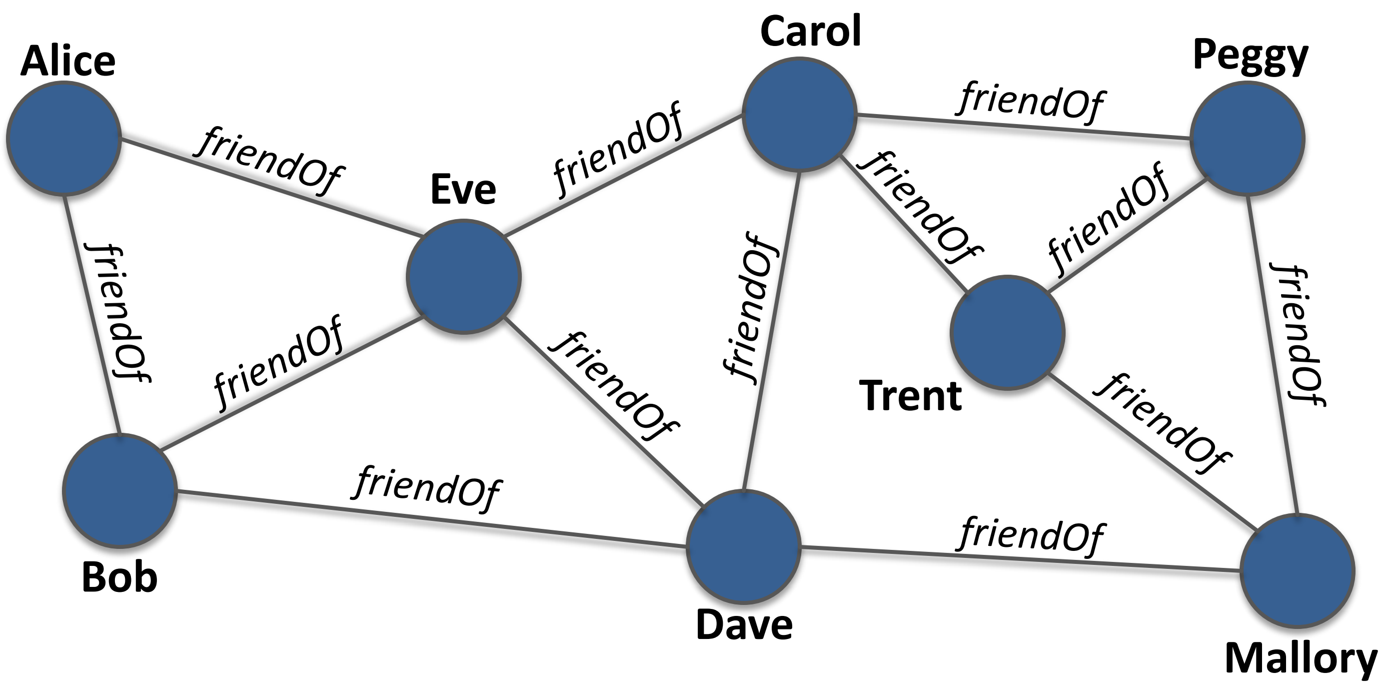
$$\boldsymbol{G}raph = (\boldsymbol{V}ertices, \boldsymbol{E}dges)$$

“Graphs are everywhere”



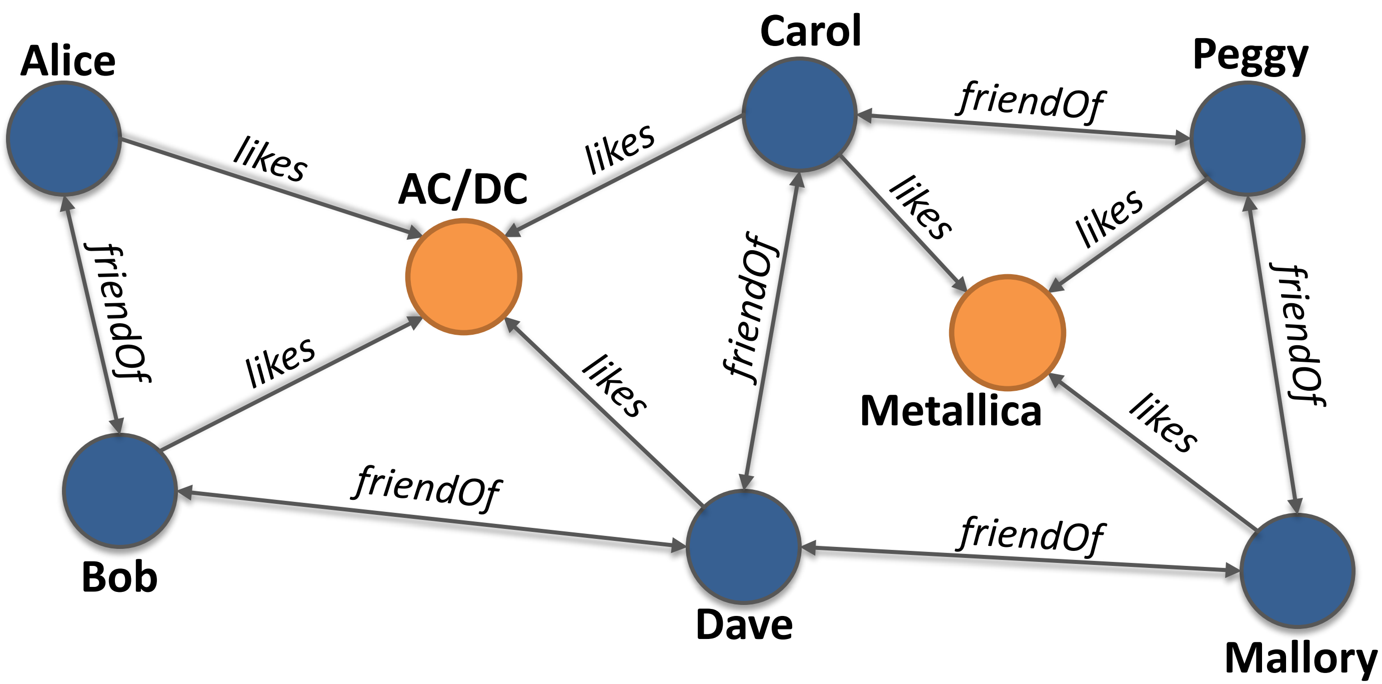
$$Graph = (\mathbf{Users}, Followers)$$

"Graphs are everywhere"



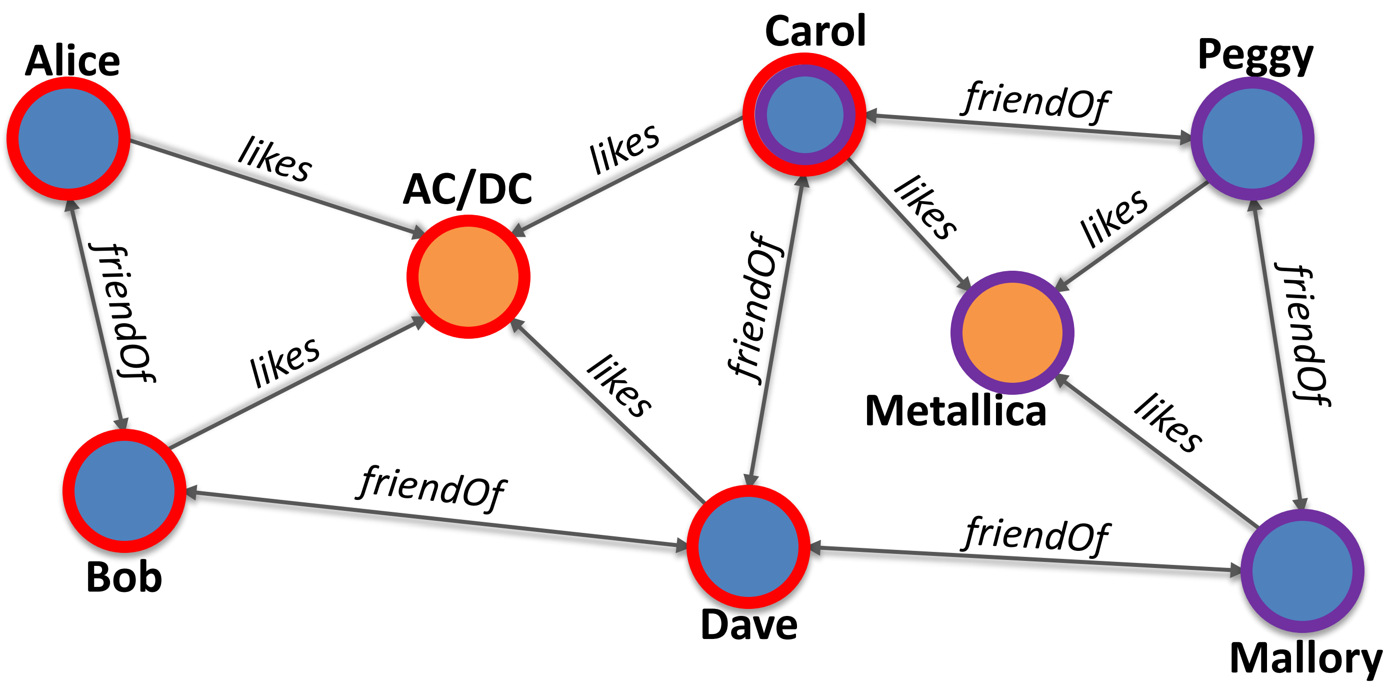
$$Graph = (\mathbf{Users}, Friendships)$$

“Graphs are heterogeneous”



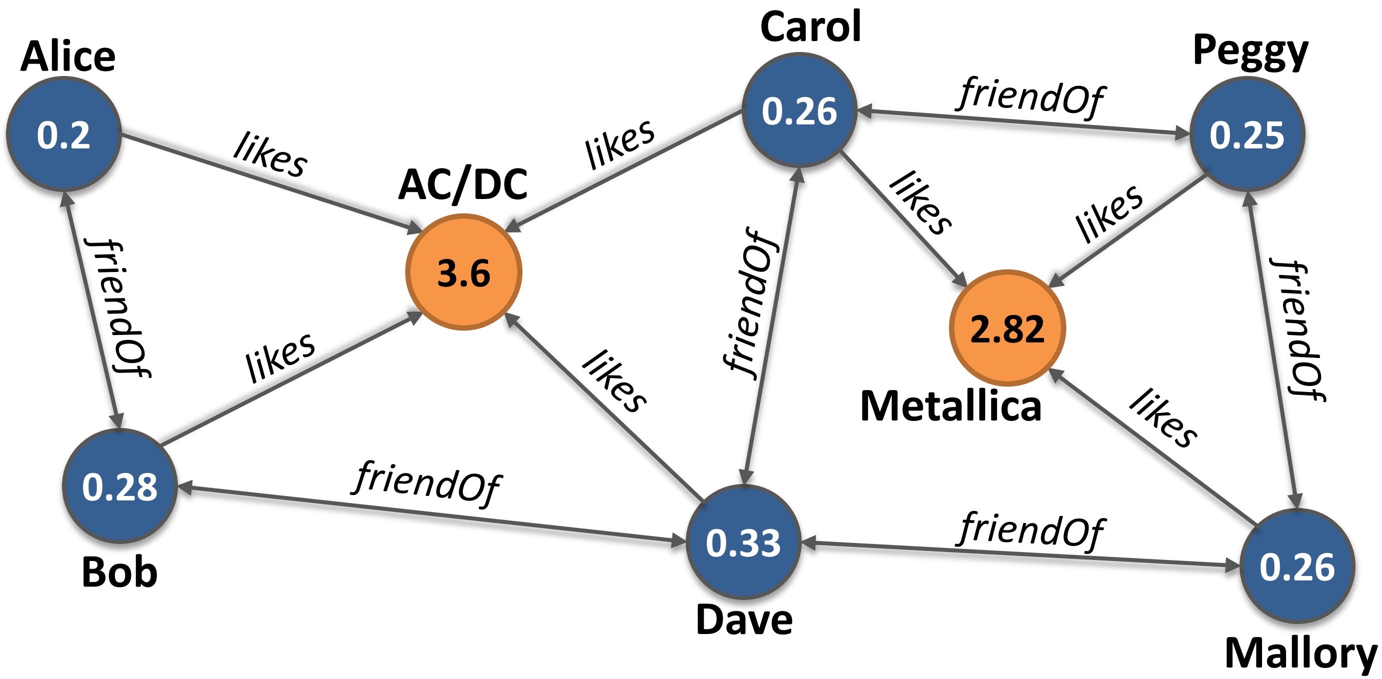
$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

“Graphs can be analyzed”



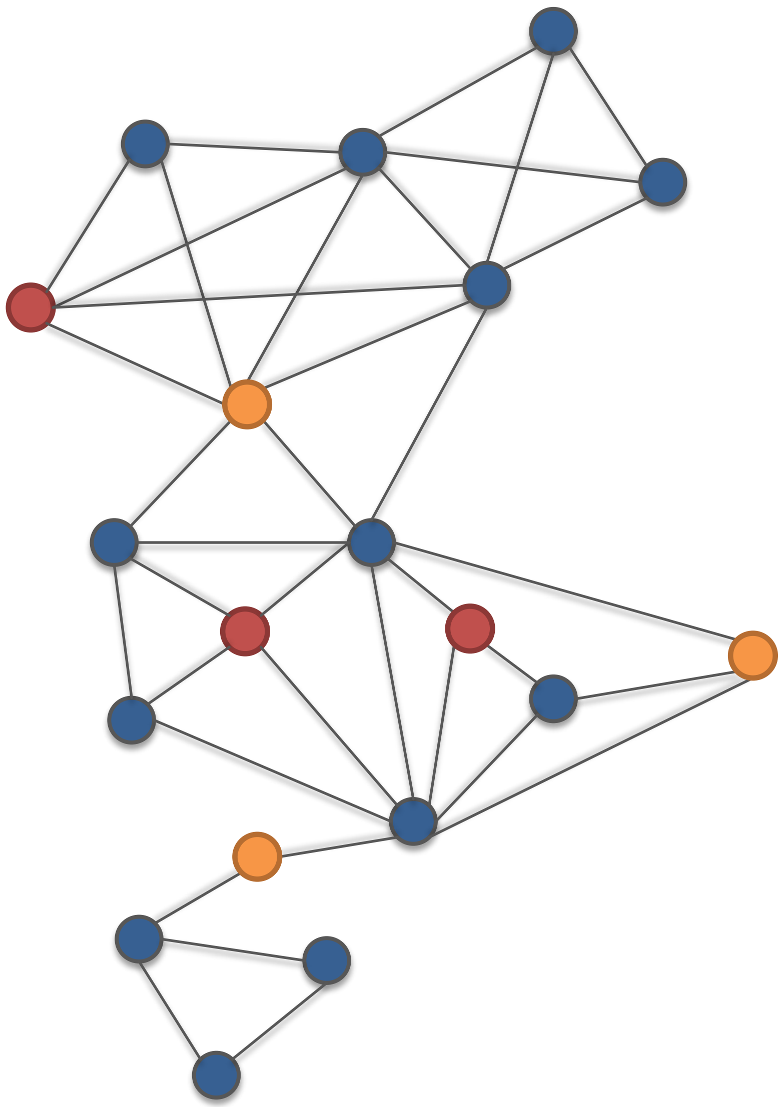
$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

“Graphs can be analyzed”



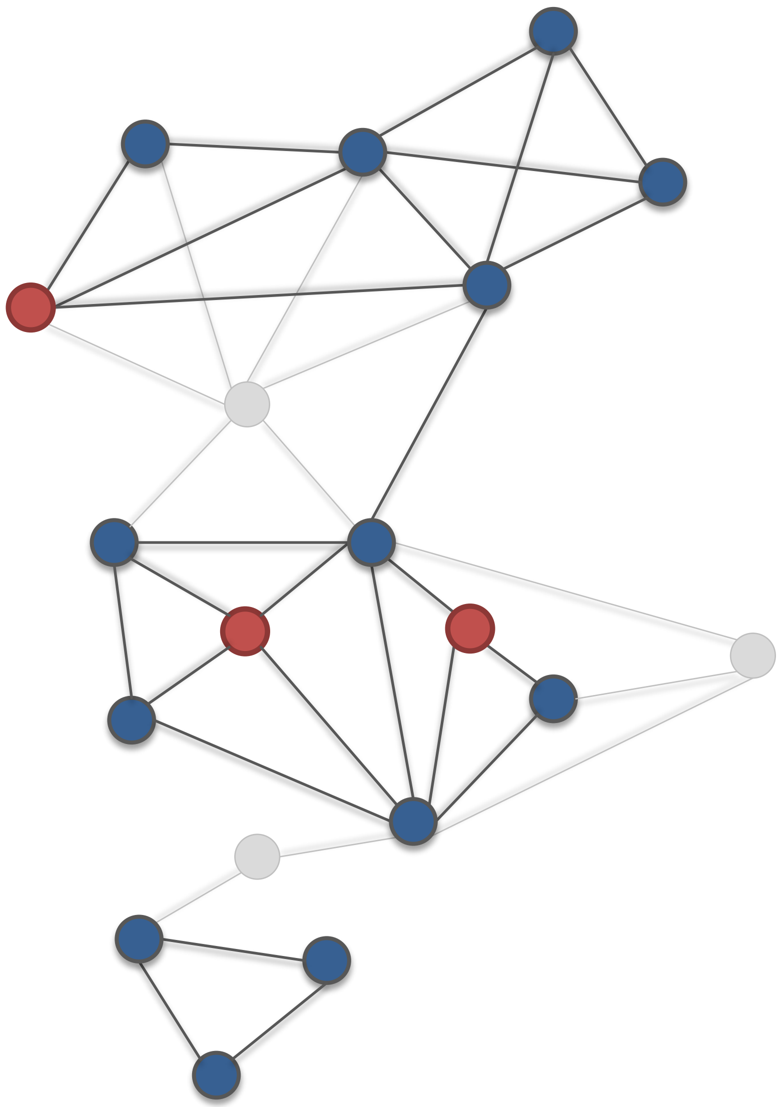
$$Graph = (\mathbf{Users} \cup \mathbf{Bands}, Friendships \cup Likes)$$

“Graphs can be analyzed“



Assuming a social network

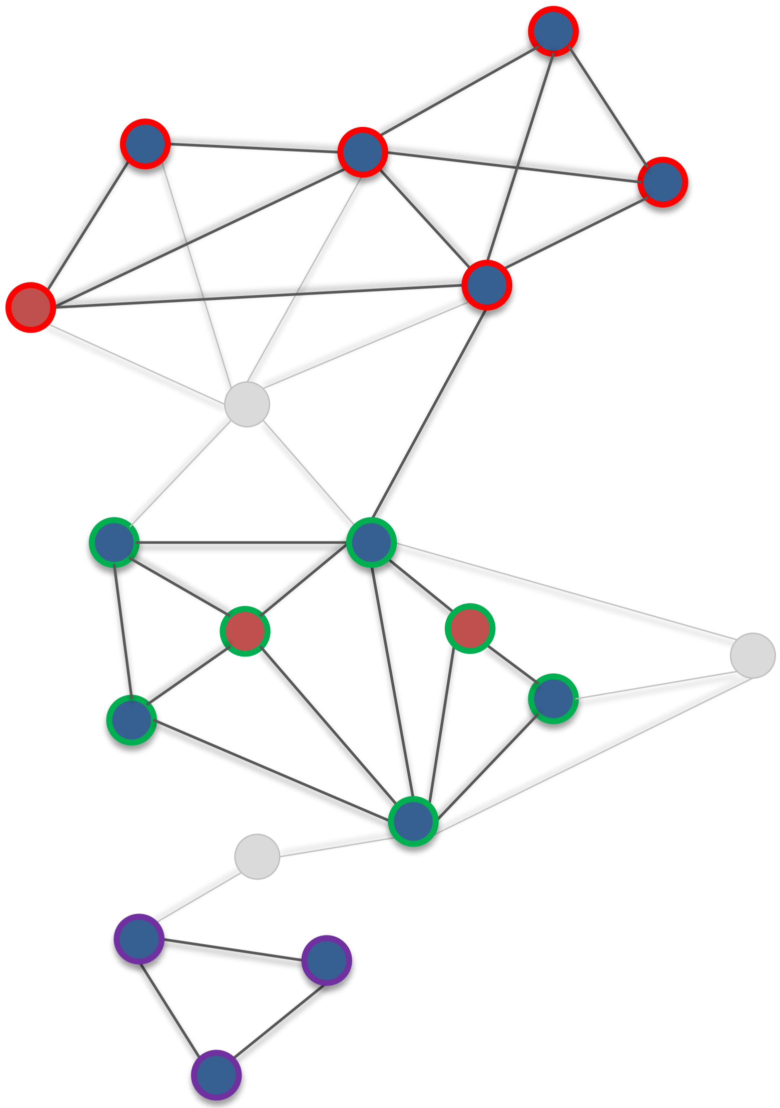
“Graphs can be analyzed“



Assuming a social network

1. Determine subgraph

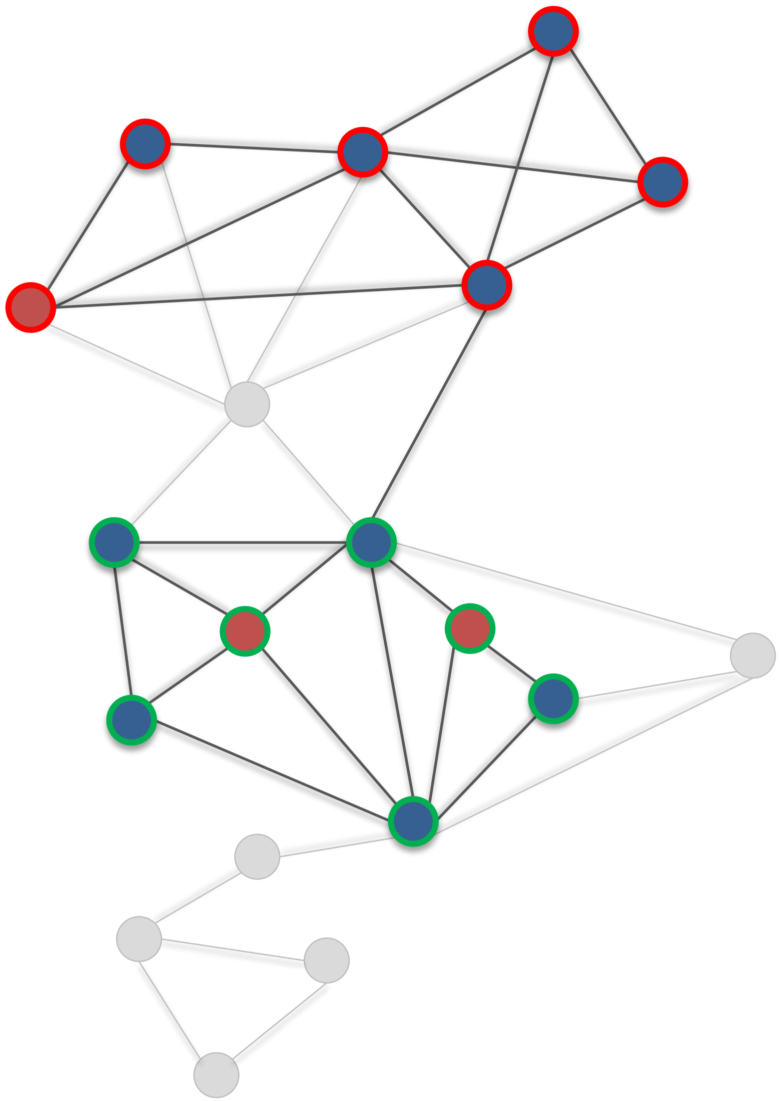
“Graphs can be analyzed“



Assuming a social network

1. Determine subgraph
2. Find communities

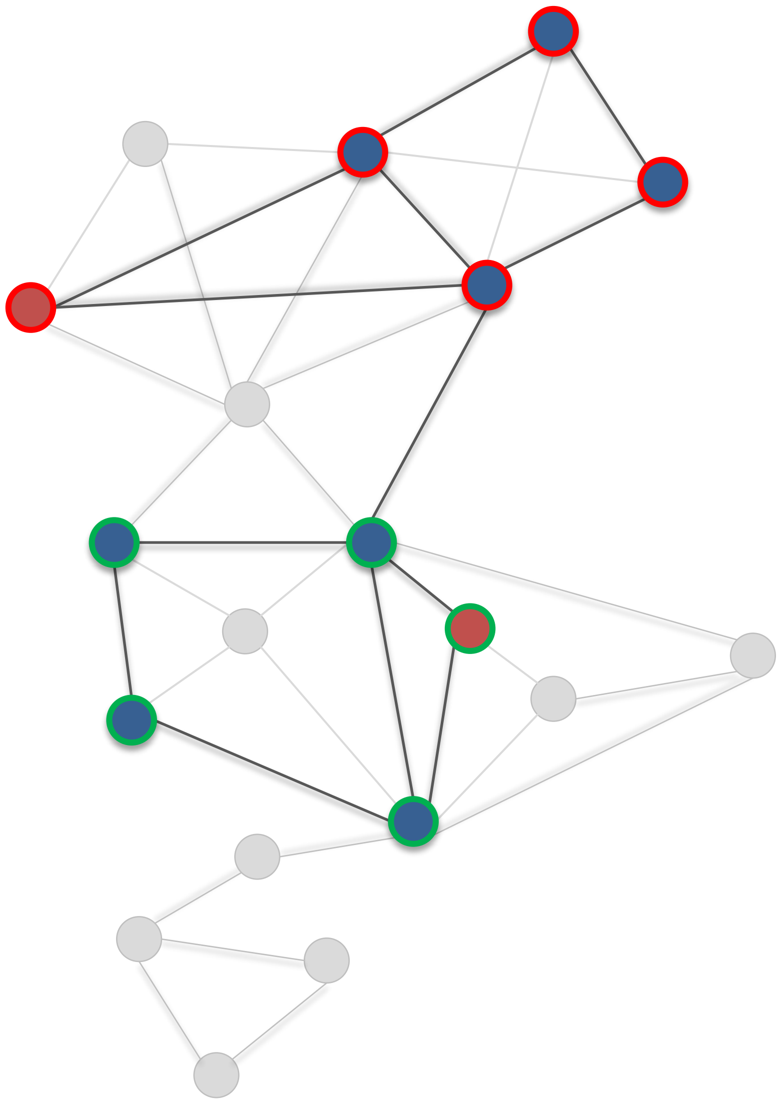
“Graphs can be analyzed“



Assuming a social network

1. Determine subgraph
2. Find communities
3. Filter communities

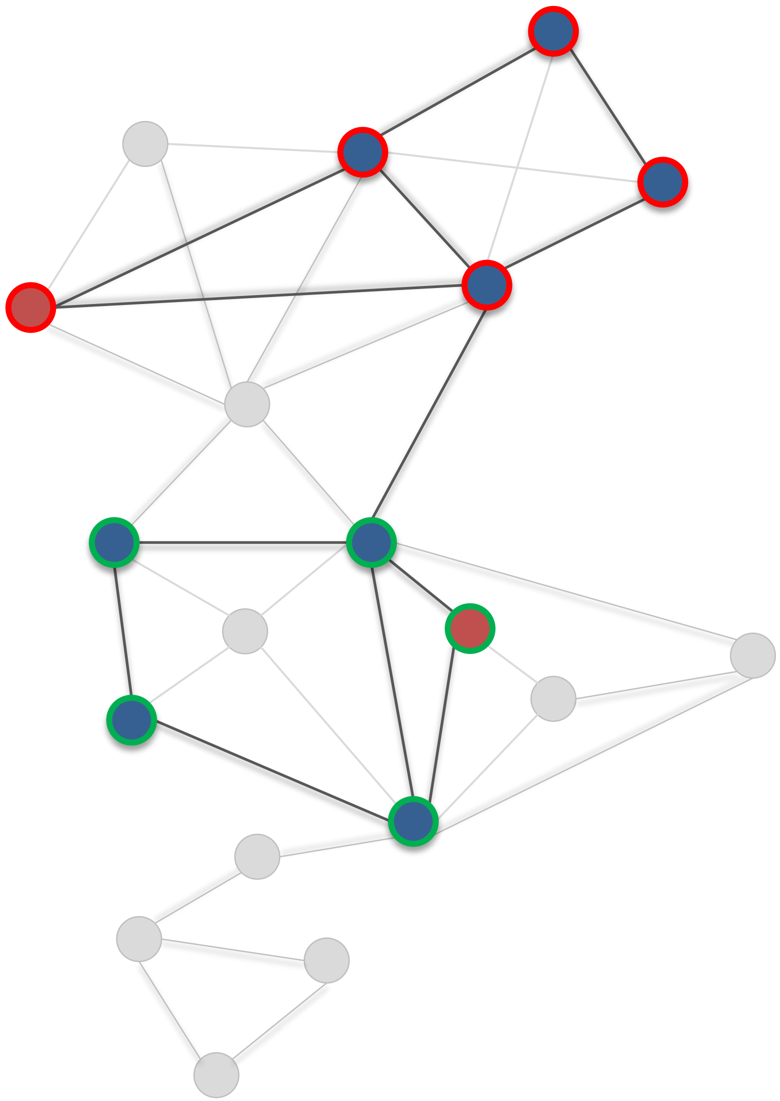
“Graphs can be analyzed“



Assuming a social network

1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph

“Graph data models must be expressive“



Assuming a social network

- **Heterogeneous data**

1. Determine subgraph

- **Apply graph transformation**

2. Find communities

- **Handle collections of graphs**

3. Filter communities

- **Aggregation, Selection**

4. Find common subgraph

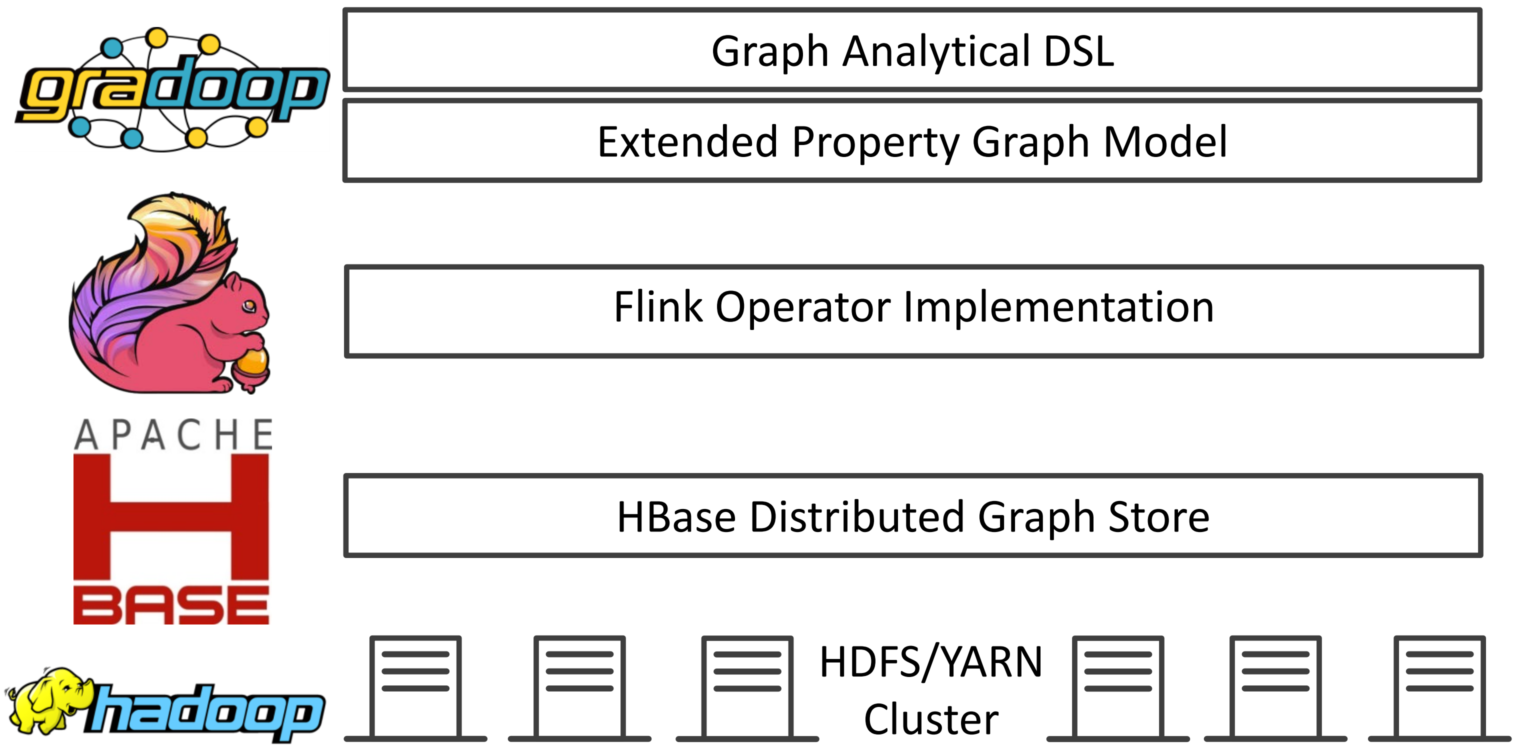
- **Apply dedicated algorithms**

„And let‘s not forget …“

"...Graphs are large"

A framework and research platform for **efficient**, **distributed** and domain independent graph data **management** and **analytics**.

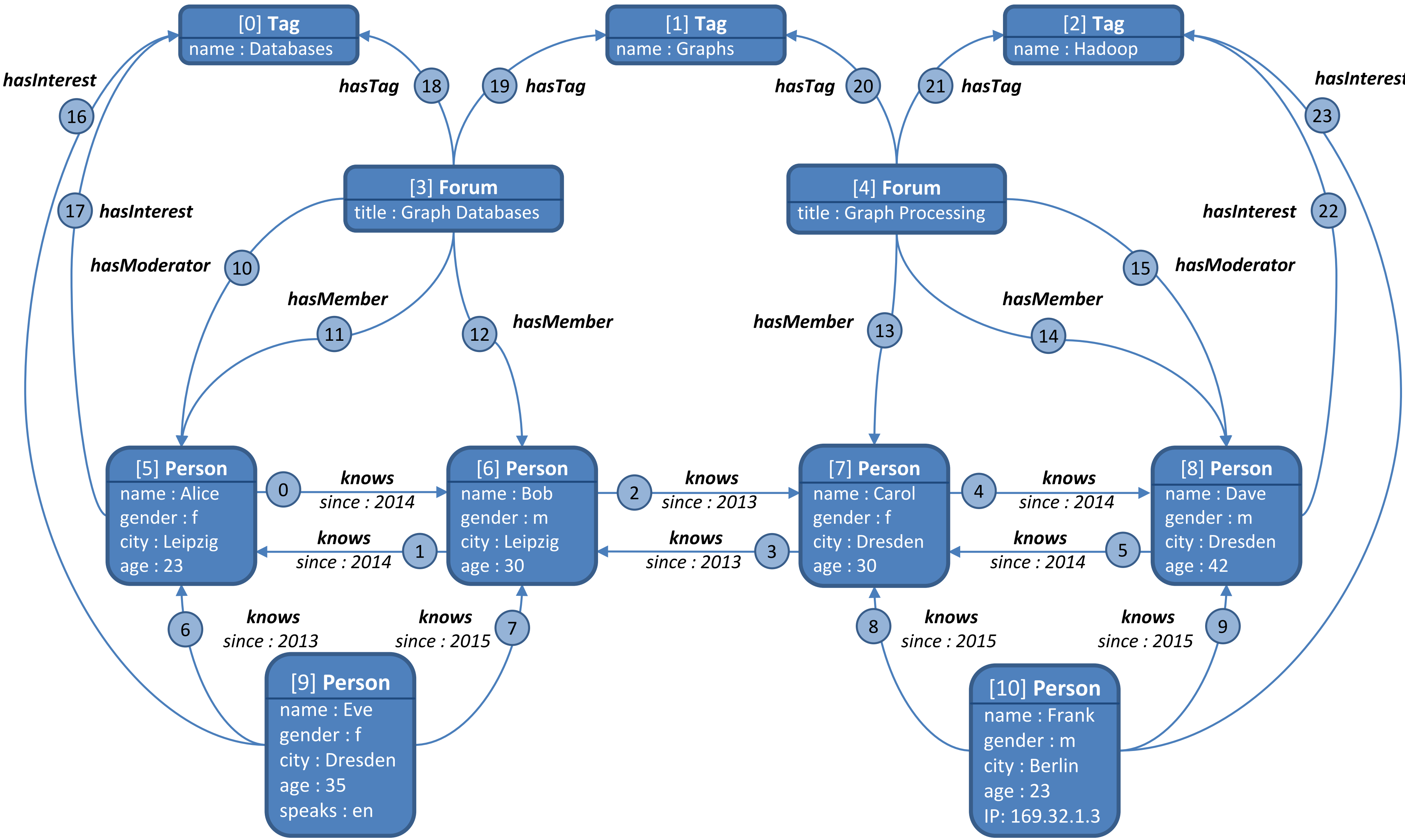
High Level Architecture



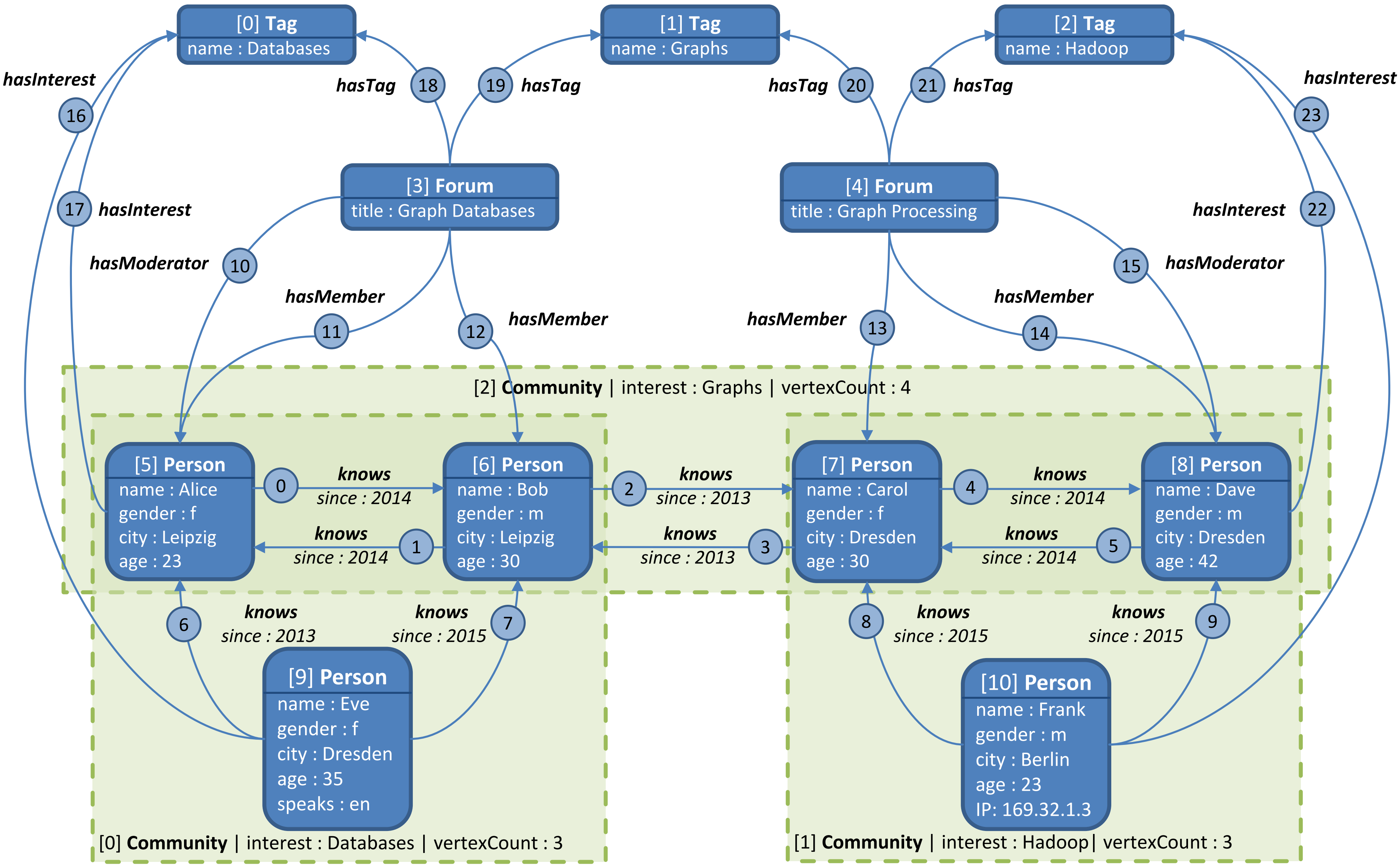
- Java
- 25K (33K) LOC
- GPLv3

Extended Property Graph Model (EPGM)

EPGM – Graph Representation



EPGM – Graph Representation



EPGM – Operators and Algorithms

	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 *		

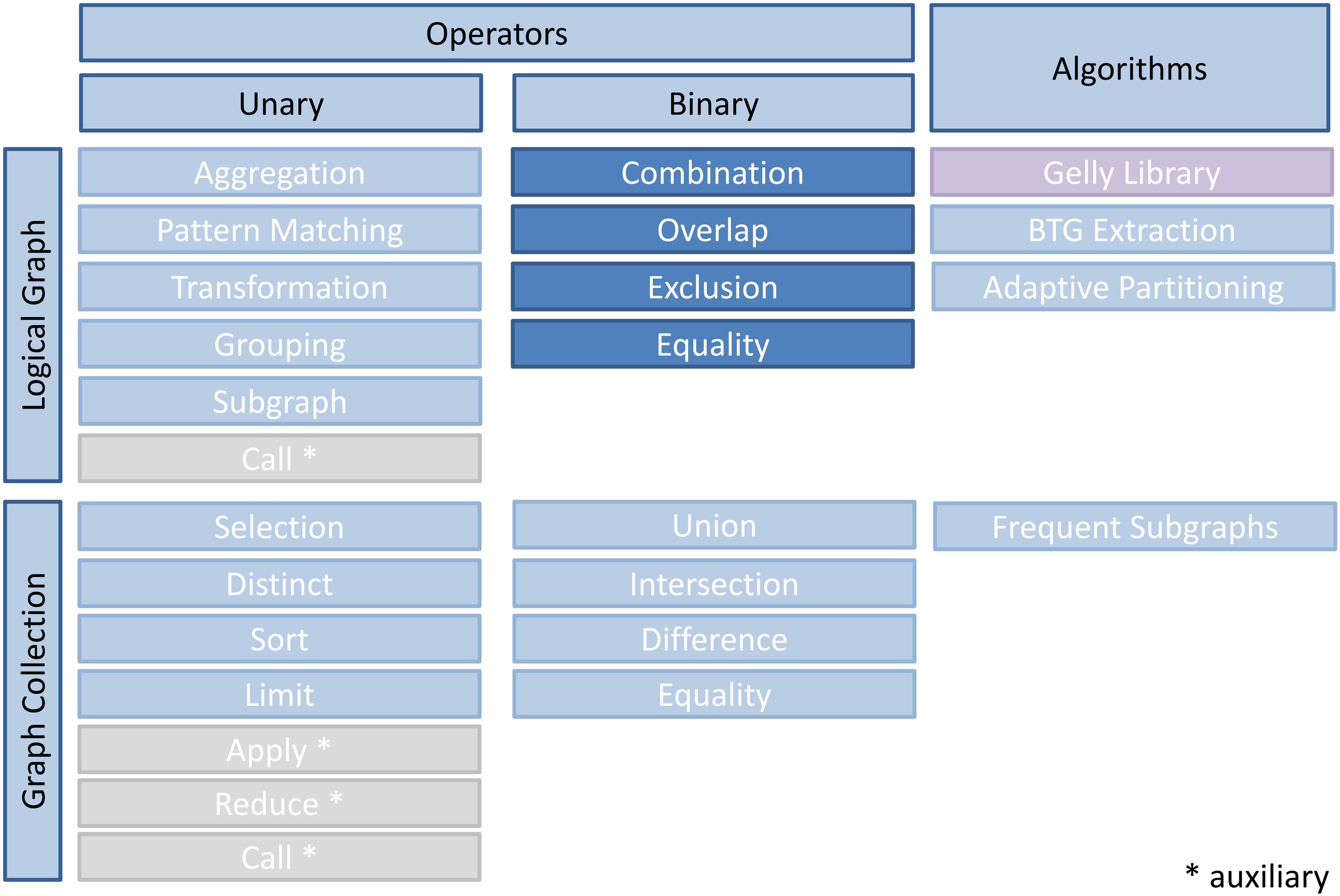
* auxiliary

EPGM – Operators and Algorithms

	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 *		

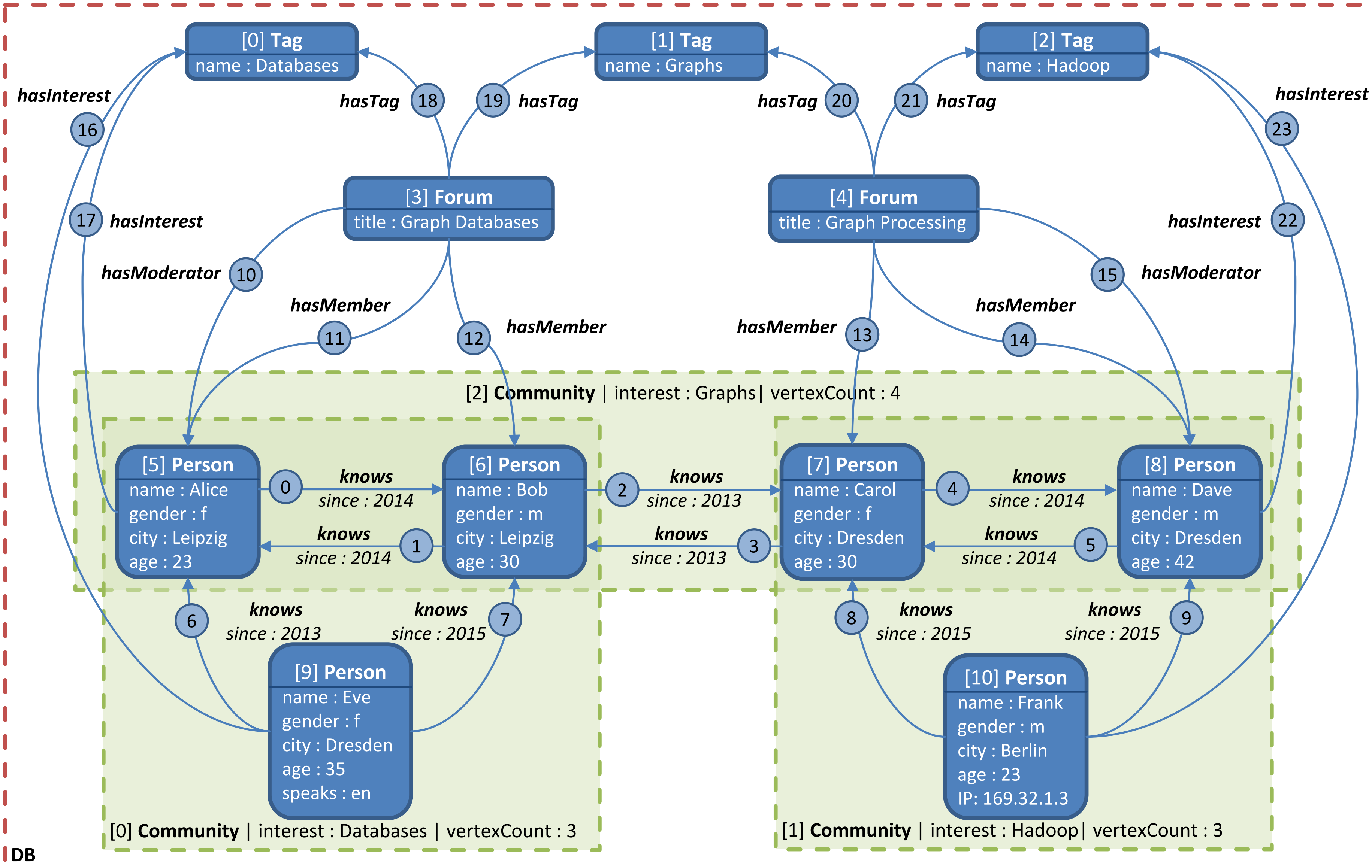
* auxiliary

EPGM – Operators and Algorithms



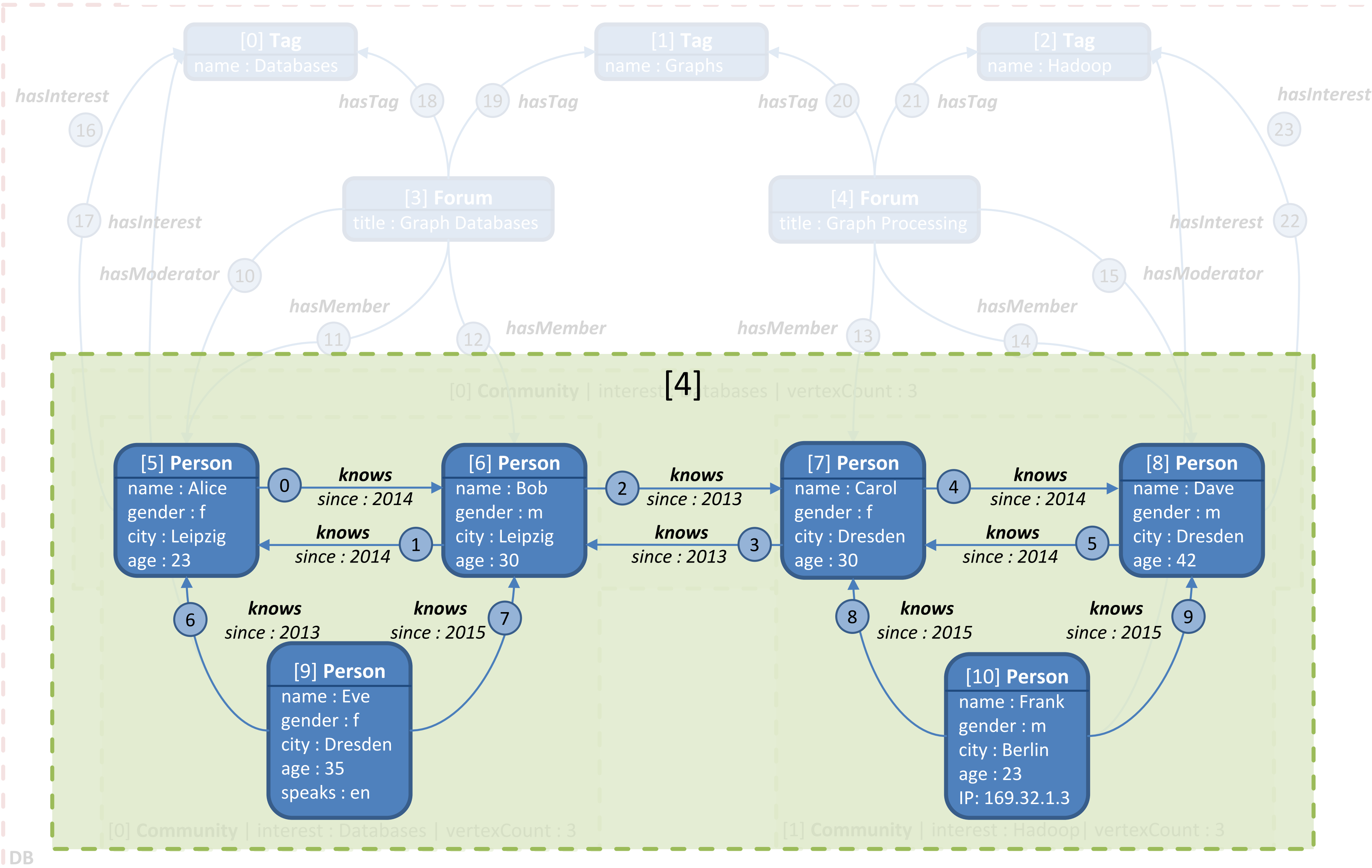
Combination

```
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])
```



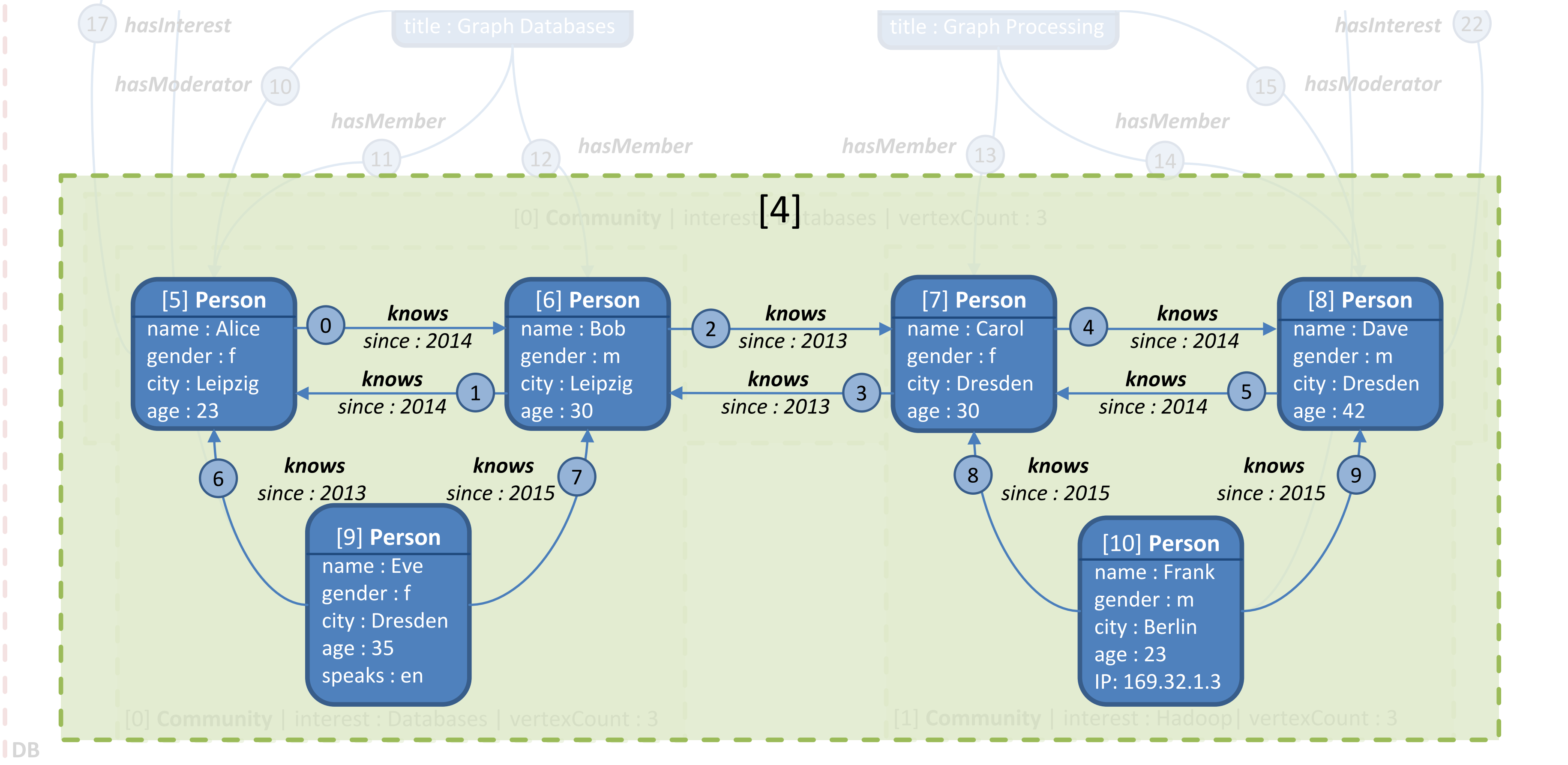
EPGM – Operators and Algorithms

	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

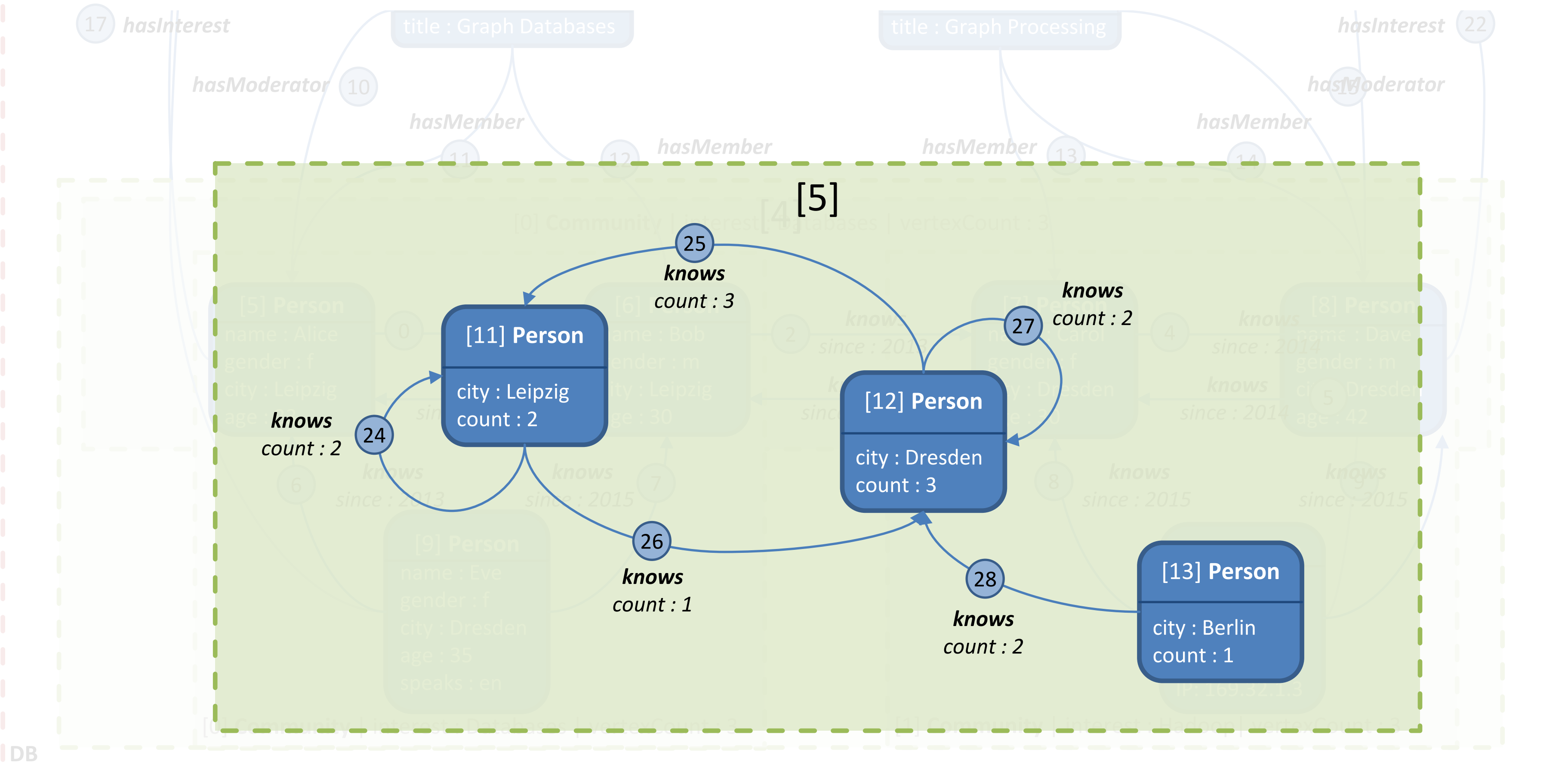
Combination + Grouping

```
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)
```



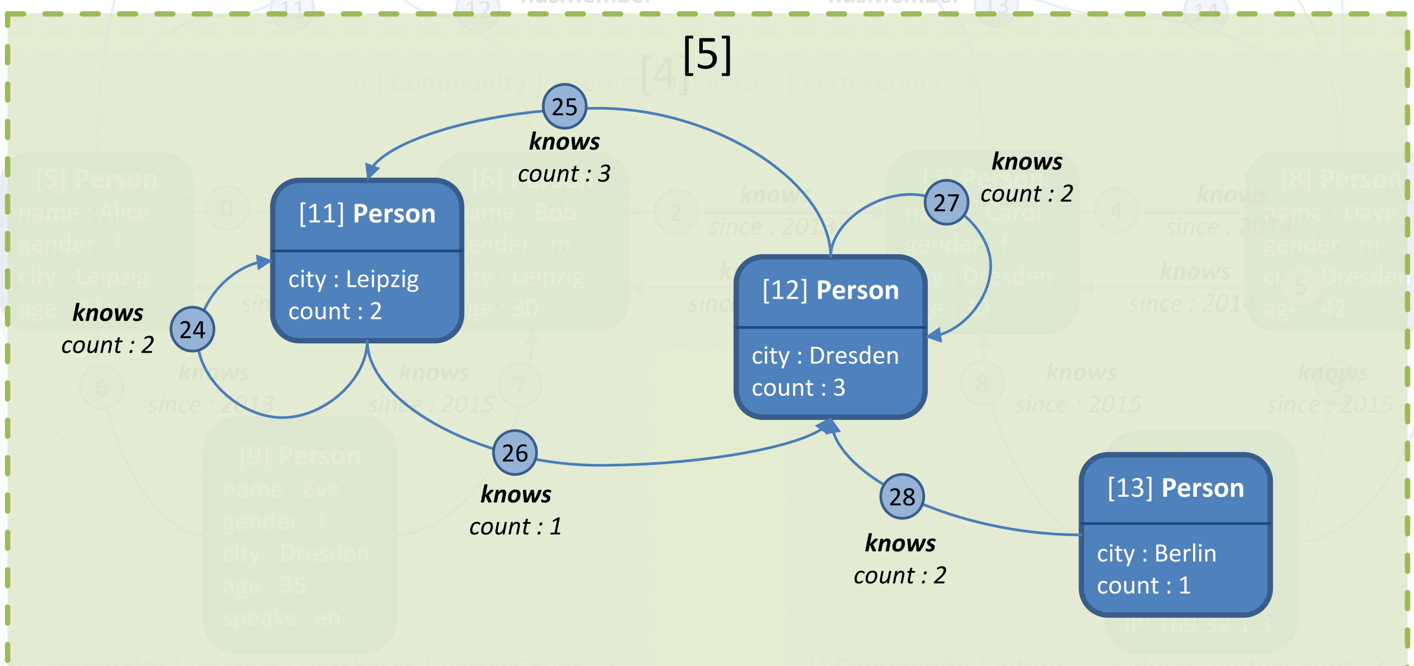
Combination + Grouping

```
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)
```



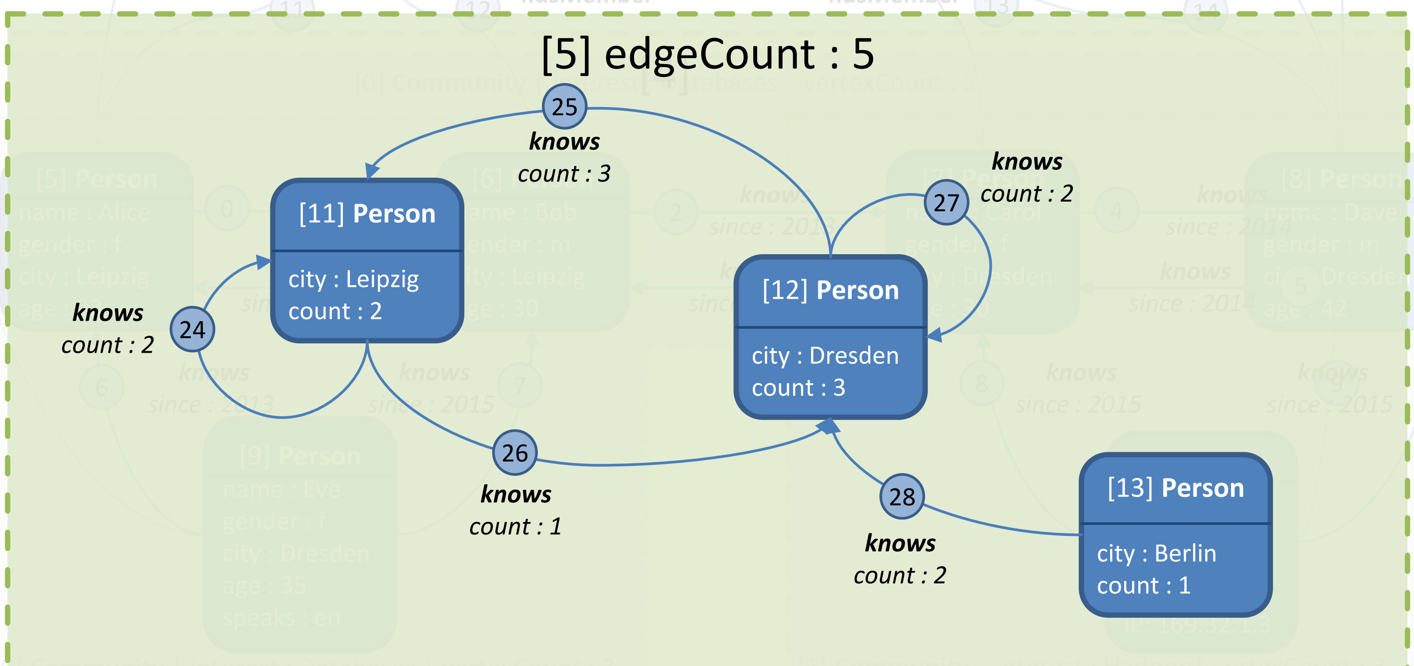
Combination + Grouping + Aggregation

```
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)
7: aggFunc = (g => |g.E|)
8: aggGraph = sumGraph.aggregate(“edgeCount”, aggFunc)
```



Combination + Grouping + Aggregation

```
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)
aggFunc = (g => |g.E|)
aggGraph = sumGraph.aggregate(“edgeCount”, aggFunc)
```



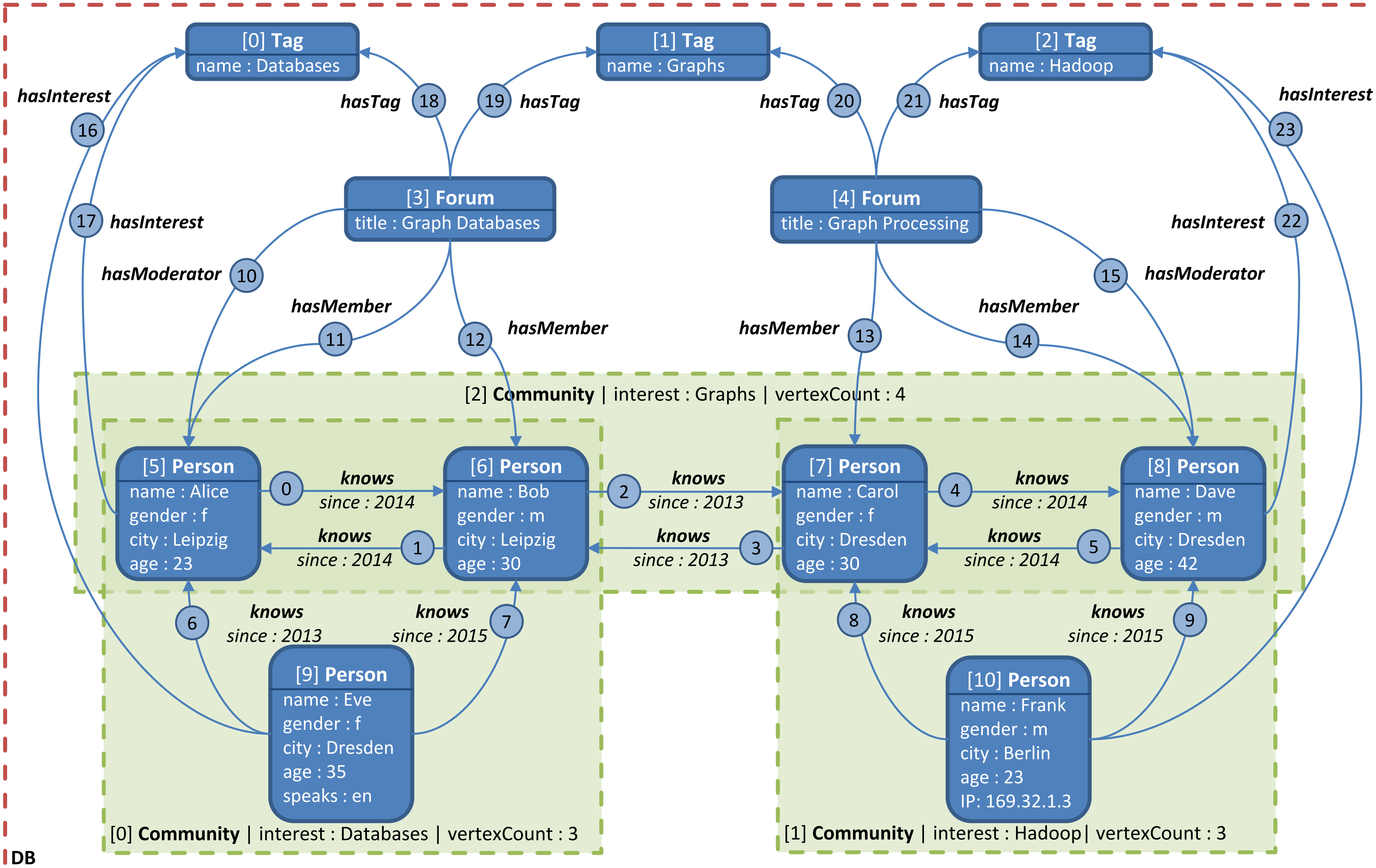
EPGM – Operators and Algorithms

	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

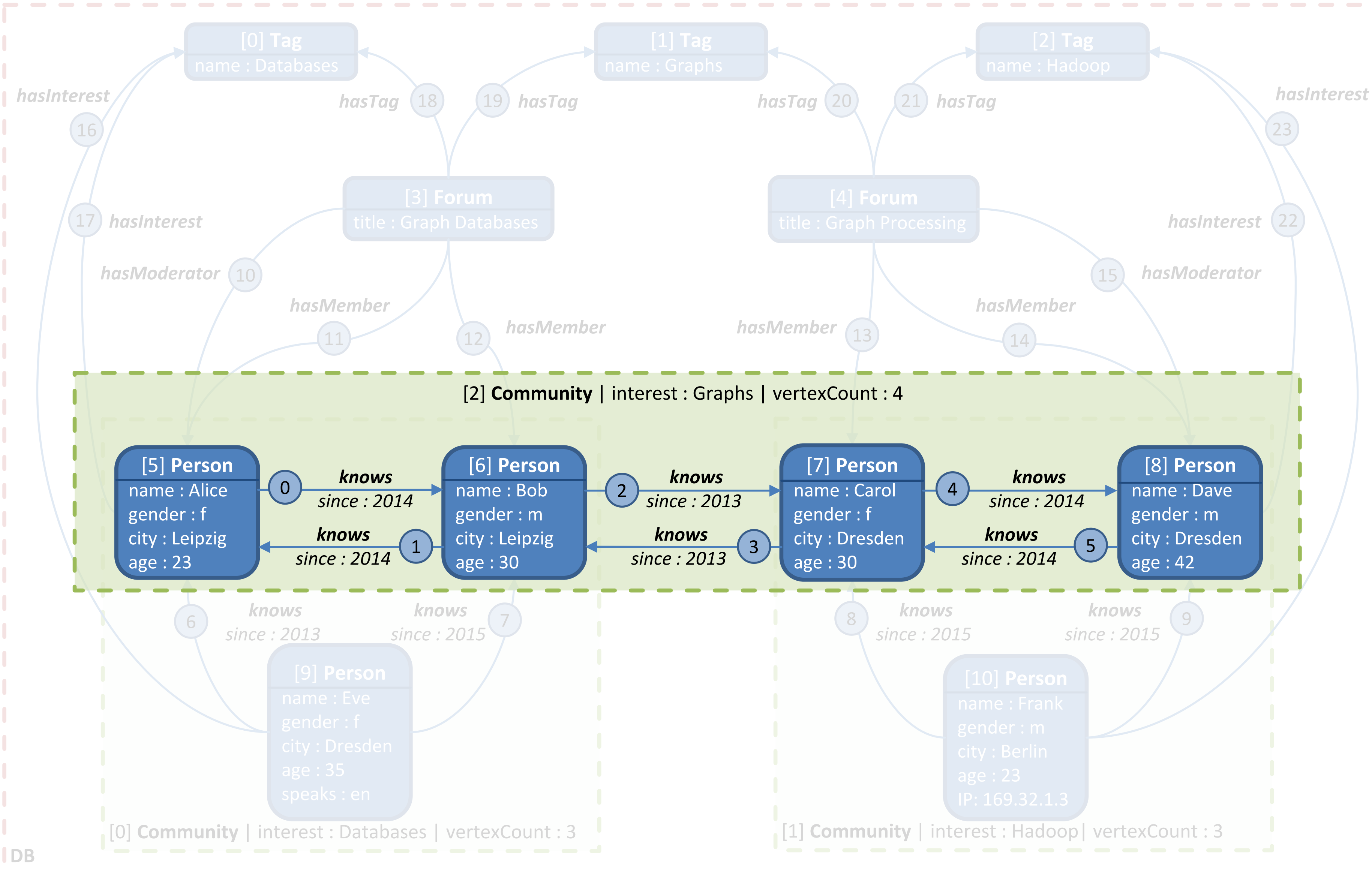
Selection

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



Selection

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



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

Apache Flink

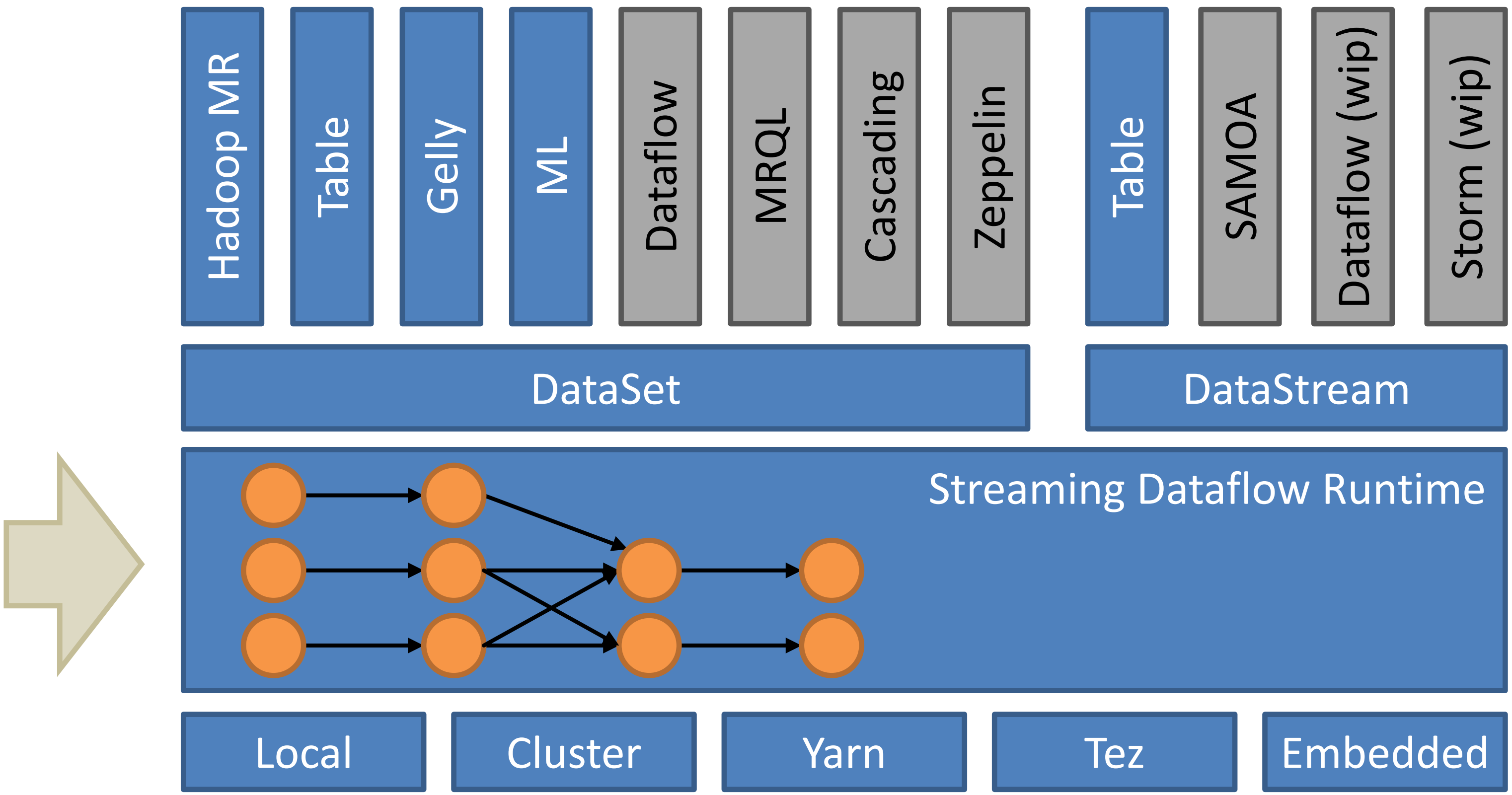
Apache Flink

„Streaming Dataflow Engine that provides

- *data distribution,*
- *communication,*
- and *fault tolerance*

for distributed computations over data streams.“

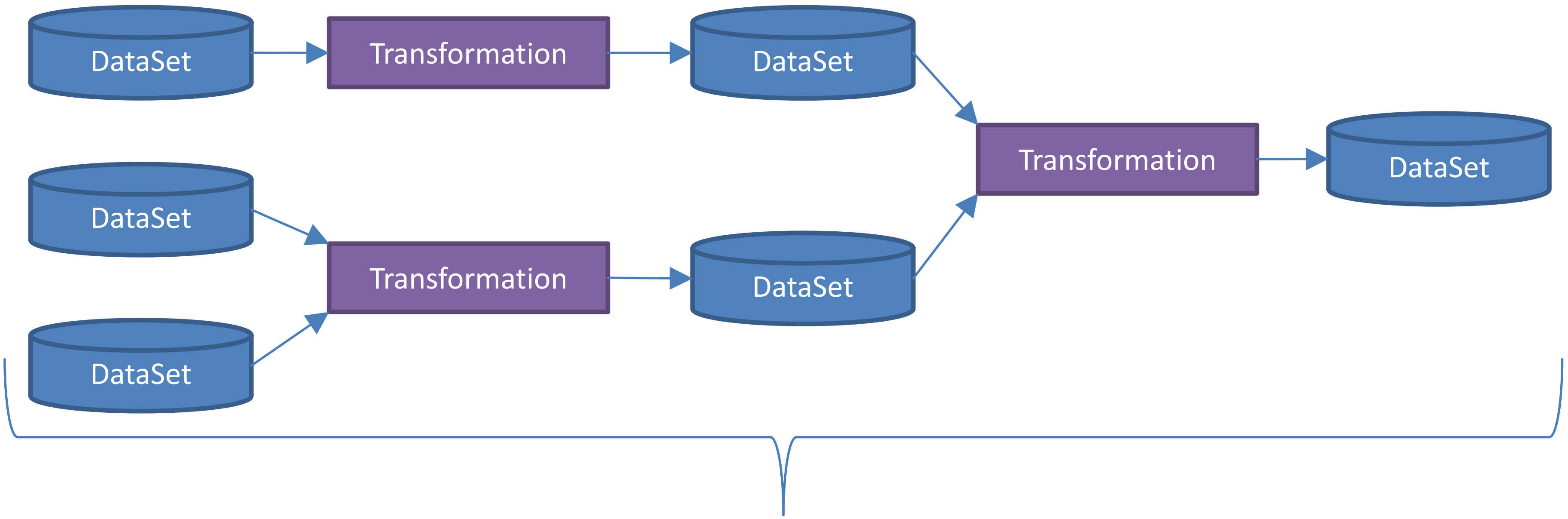
HDFS
LocalFS
HBase
JDBC
Kafka
RabbitMQ
Flume
(Neo4j)



http://www.slideshare.net/robertmetzger1/apache-flink-meetup-munich-november-2015-flink-overview-architecture-integrations-and-use-case

Apache Flink – DataSet API

- **DataSet** := Distributed Collection of Data
- **Transformation** := Operation applied on DataSet
- **Flink Program** := Composition of Transformations



Flink Program

Apache Flink – DataSet Transformations

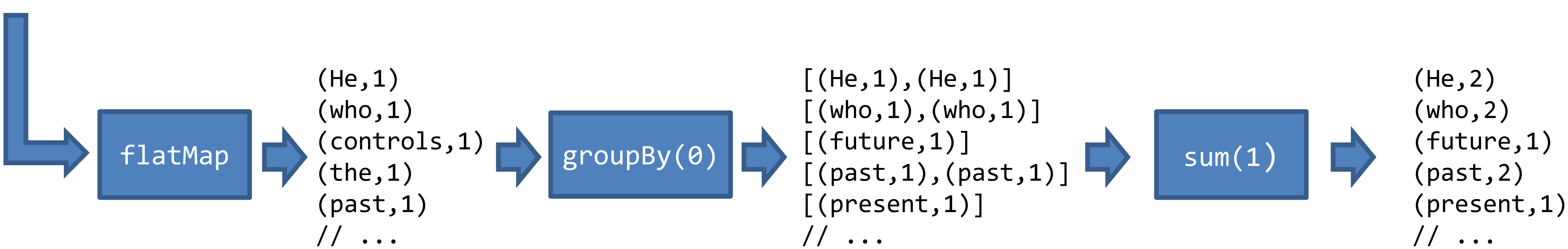
- aggregate
- coGroup
- cross
- distinct
- filter
- first-N
- flatMap
- groupBy
- join
- leftOuterJoin
- rightOuterJoin
- fullOuterJoin
- map
- mapPartition
- project
- reduce
- reduceGroup
- union

- iterate
- iterateDelta

The „Hello World“ of Big Data – Word Count

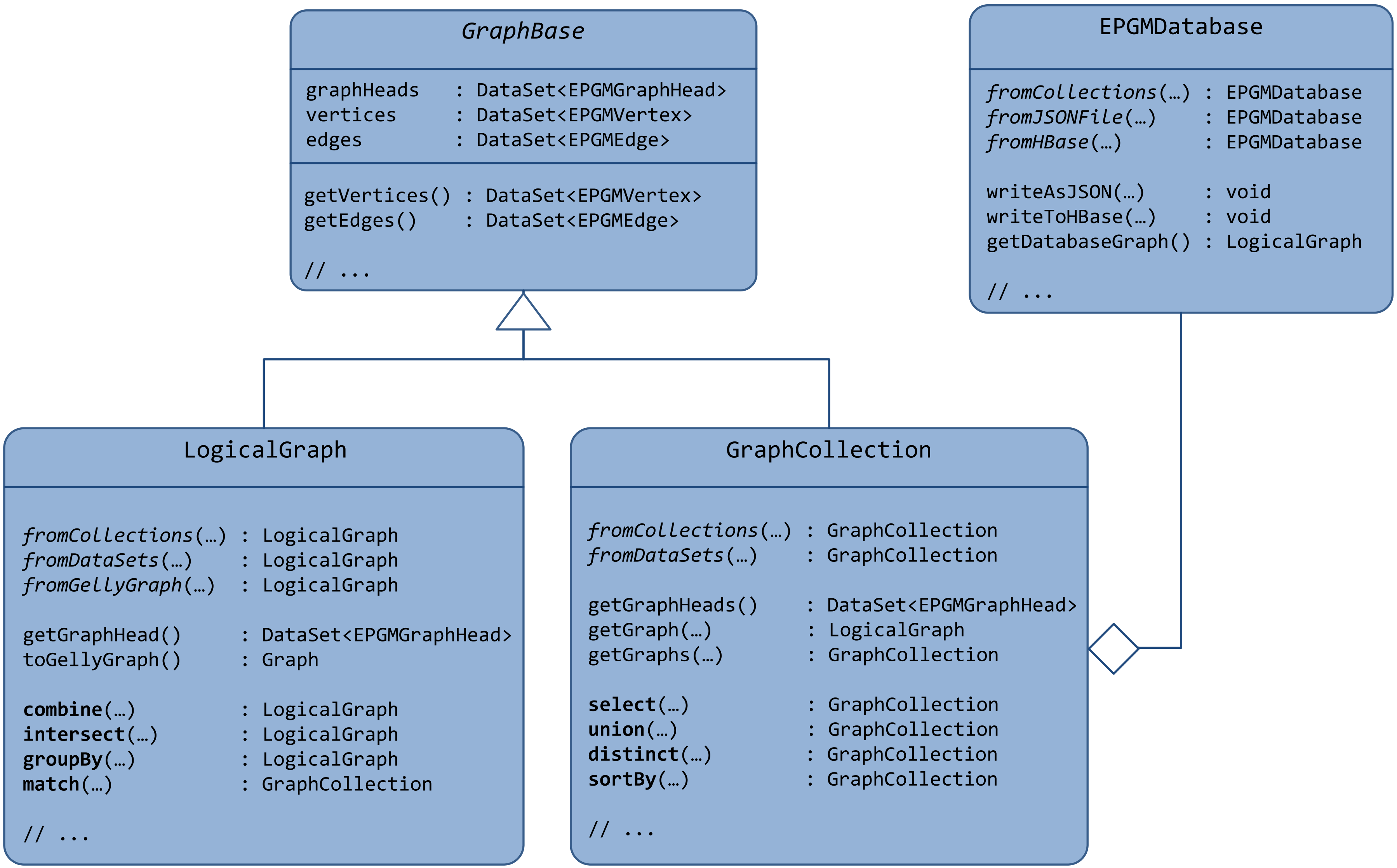
```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

DataSet<String> text = env.fromElements( // or env.readTextFile(„hdfs://…“)
        „He who controls the past controls the future.“,
        „He who controls the present controls the past.“);

DataSet<Tuple2<String, Integer>> wordCounts = text
        .flatMap(new LineSplitter()) // splits the line and outputs (word, 1) tuples
        .groupBy(0)
        .sum(1);

wordCounts.print(); // trigger execution
```

„He who controls the past controls the future.“
„He who controls the present controls the past.“



EPGM on Apache Flink

EPGM on Apache Flink – User facing API



EPGM on Apache Flink – DataSets

EPGMGraphHead

Id	Label	Properties

POJO → DataSet<EPGMGraphHead>

EPGMVertex

Id	Label	Properties	Graphs

POJO → DataSet<EPGMVertex>

EPGMEdge

Id	Label	Properties	SourceId	TargetId	Graphs

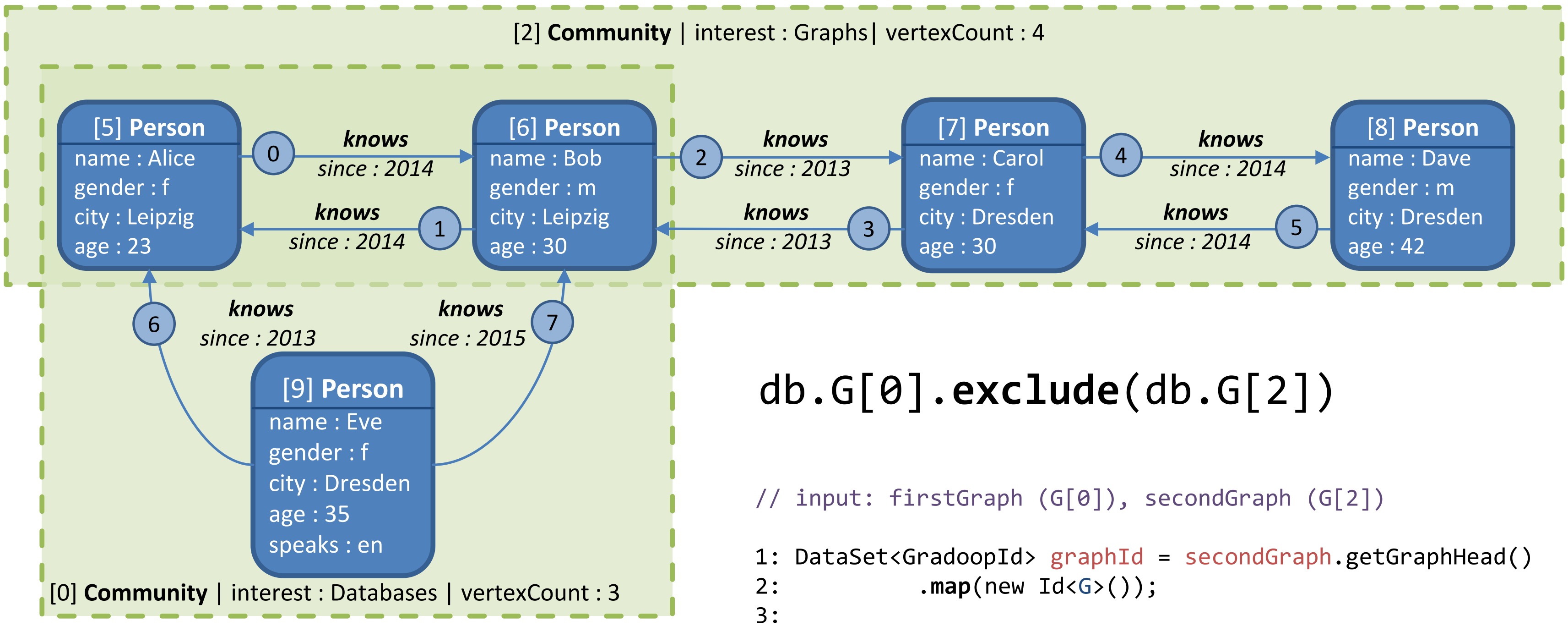
POJO → DataSet<EPGMEdge>

EPGMVertex

Id	Label	Properties	Graphs

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

EPGM on Apache Flink – Exclusion




```
db.G[0].exclude(db.G[2])
```

```
// input: firstGraph (G[0]), 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>());
```

EPGM on Apache Flink – Exclusion

```
graphId =
    secondGraph.getGraphHead()
```

Id	Label	Properties	
2	Community	interest: Graphs	vertexCount: 4

```
    .map(new Id<G>());
```

Id
2

```
newVertices =
    firstGraph.getVertices()
```

Id	Label	Properties			Graphs
5	Person	name: Alice	gender: f	...	[0, 2]
6	Person	name: Bob	gender: m	...	[0, 2]
9	Person	name: Eve	gender: f	...	[0]

```
    .filter(new NotInGraphBroadCast<V>())
    .withBroadcastSet(graphId, GRAPH_ID);
```

Id	Label	Properties			Graphs
9	Person	name: Eve	gender: f	...	[0]

EPGM in Apache Flink – Exclusion

```
newEdges =
     firstGraph.getEdges()
```

Id	Label	SourceId	TargetId	Properties	Graphs
0	knows	5	6	since: 2014	[0, 2]
1	knows	6	5	since: 2014	[0, 2]
6	knows	9	5	since: 2013	[0]
7	knows	9	6	since: 2015	[0]

```
    .filter(new NotInGraphBroadCast<E>())
    .withBroadcastSet(graphId, GRAPH_ID)
```

Id	Label	SourceId	TargetId	Properties	Graphs
6	knows	9	5	since: 2013	[0]
7	knows	9	6	since: 2015	[0]

```
    .join(newVertices)
    .where(new SourceId<E>().equalTo(new Id<V>())
```

Id	Label	SourceId	TargetId	...	Id	Label	...
6	knows	**9**	5	...	**9**	Person	...
7	knows	**9**	6	...	**9**	Person	...

```
    .with(new LeftSide<E, V>())
```

Id	Label	SourceId	TargetId	...
6	knows	**9**	5	...
7	knows	**9**	6	...

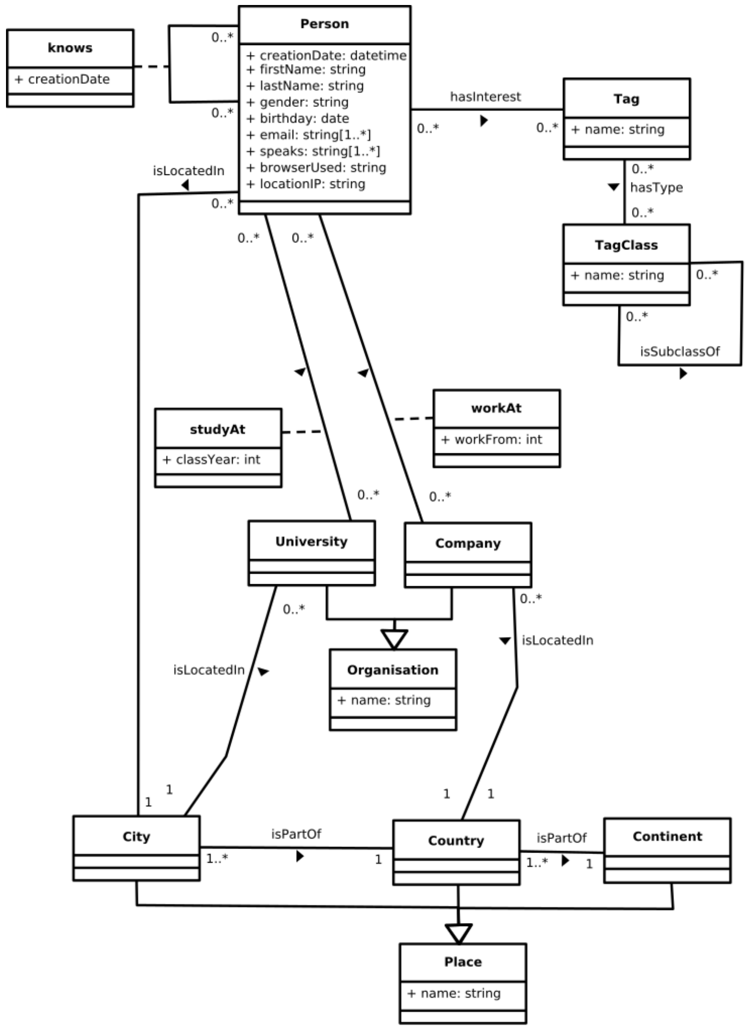
```
    .join(newVertices)
    .where(new TargetId<E>().equalTo(new Id<V>())
    .with(new LeftSide<E, V>());
```

Id	Label	SourceId	TargetId	...	Id	Label	...

Id	Label	SourceId	TargetId	...

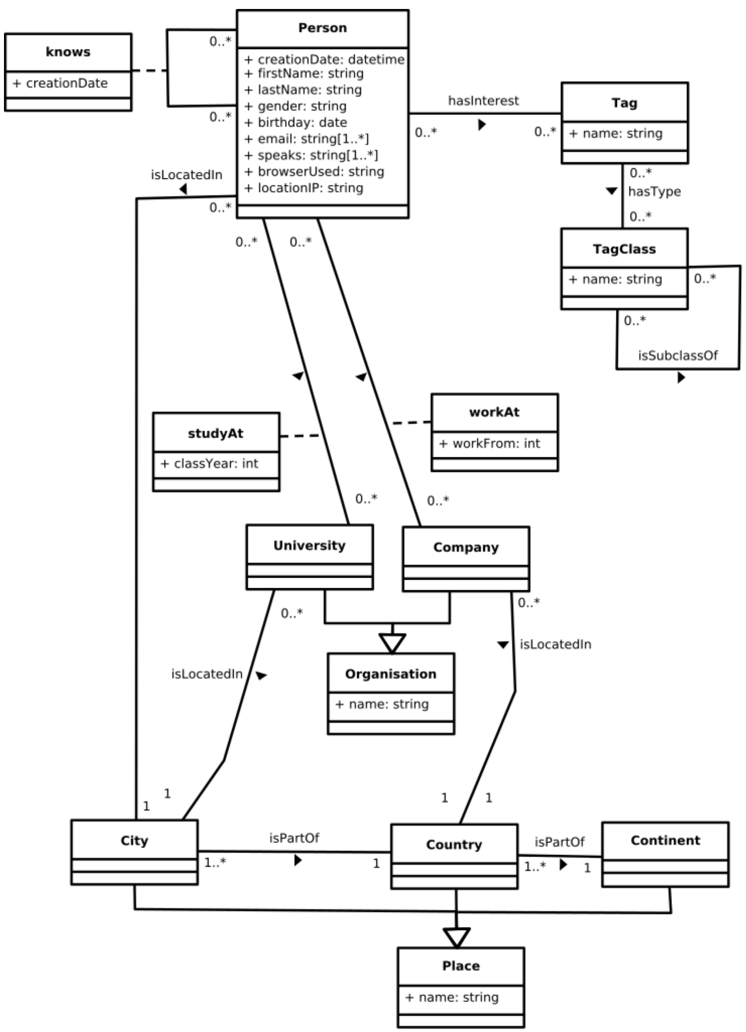
Social Network Example

LDBC Social Network Data



http://ldbcouncil.org/

LDBC Social Network Data



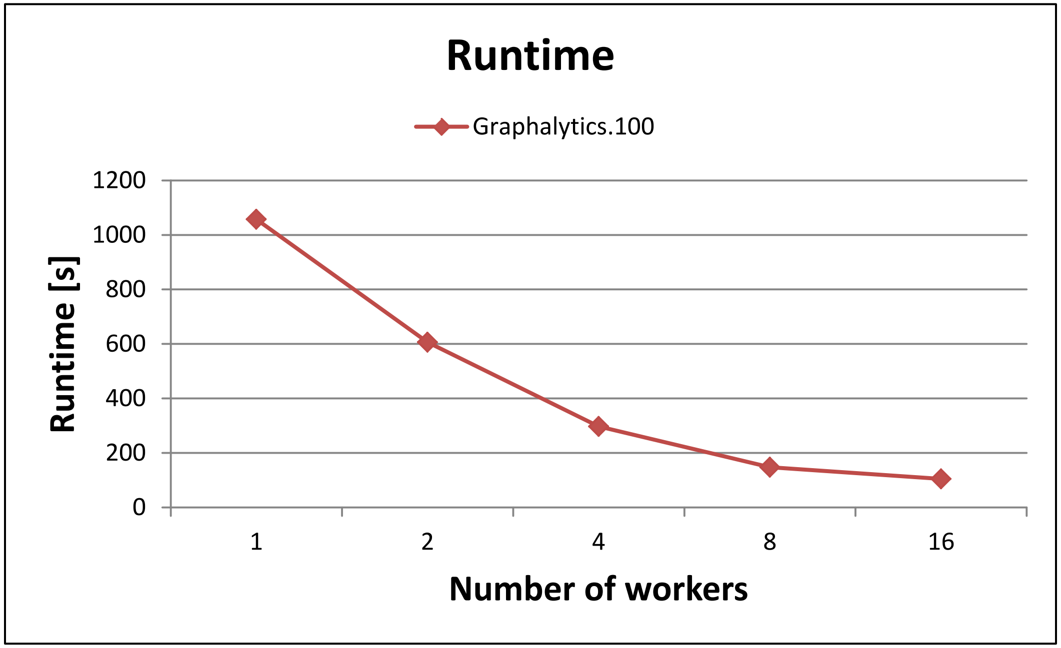

```
socialNetwork
 .subgraph(
  (v => v.label == ‘Person‘),
  (e => e.label == ‘knows‘))
 .transform(
  (gIn, gOut => gOut = gIn),
  (vIn, vOut => {
   vOut.label      = vIn.label,
   vOut[‘city‘]    = vIn[‘city‘]
   vOut[‘gender‘] = vIn[‘gender‘]
   vOut[‘key‘]     = vIn[‘birthday‘]}),
  (eIn, eOut) => 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|))
```

LDBC Social Network Data

```
return socialNetwork
  // 1) extract subgraph
  .subgraph((v) → {
      return v.getLabel().equals(person);
  }, (e) → { return e.getLabel().equals(knows); })
  // 2) project to necessary information
  .transform(
    (gIn, gOut) → {
      return gIn;
    }, (vIn, vOut) → {
        vOut.setLabel(vIn.getLabel());
        vOut.setProperty(city, vIn.getPropertyValue(city));
        vOut.setProperty(gender, vIn.getPropertyValue(gender));
        vOut.setProperty(label, vIn.getPropertyValue(birthday));
        return vOut;
    }, (eIn, eOut) → {
        eOut.setLabel(eIn.getLabel());
        return eOut;
  })
  // 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() > minClusterSize;
  })
  // 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>());
```

```
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|))
```

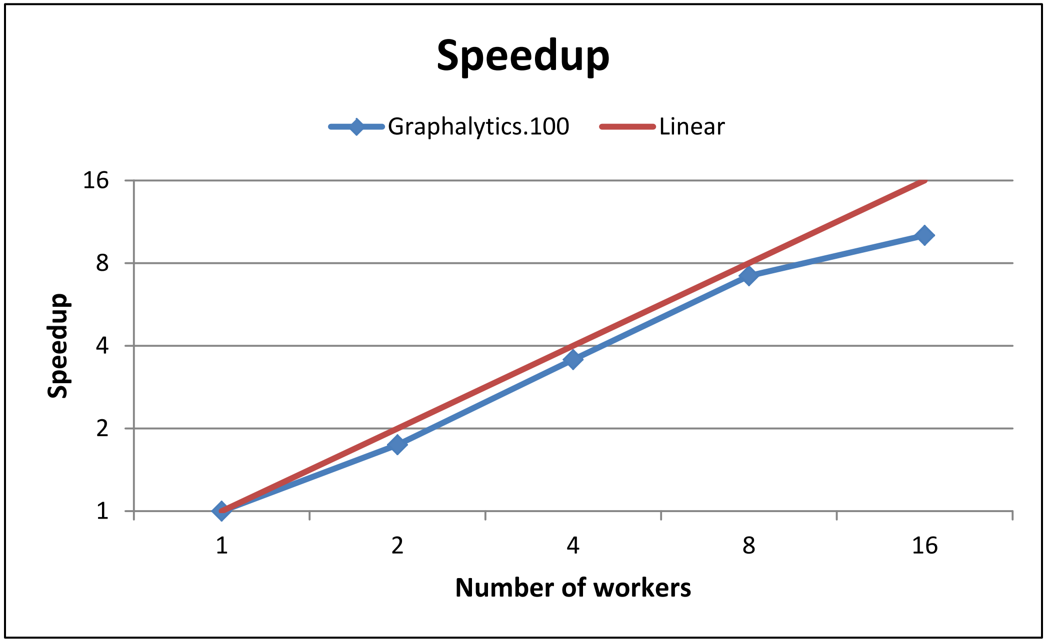
Benchmark Results



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
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

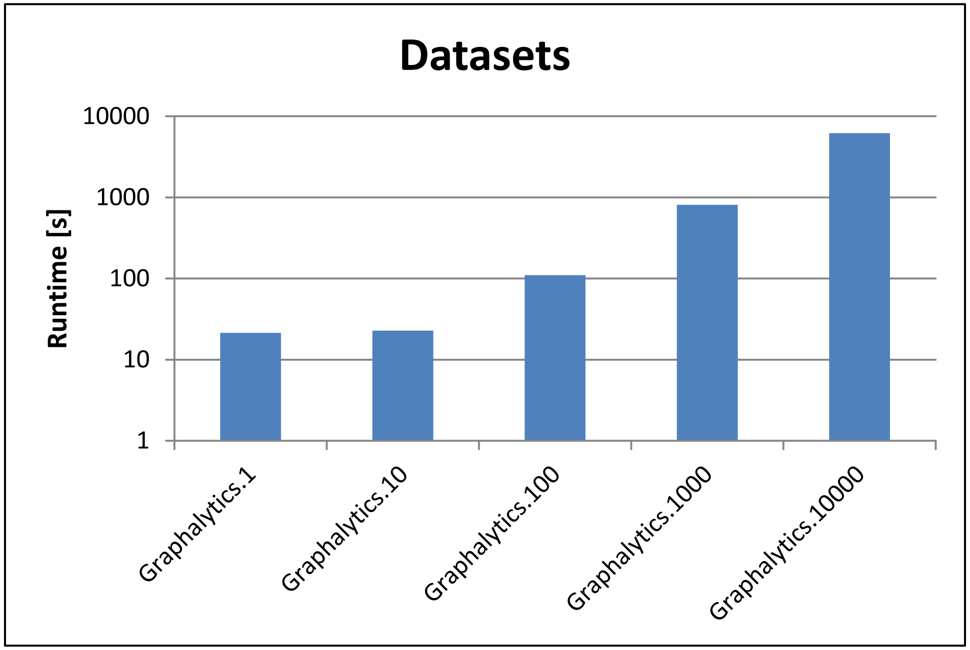
Benchmark Results



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
    - slots (per worker) 12
    - jobmanager.heap.mb 2048
    - taskmanager.heap.mb 40960

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
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

Current State & Future Work

Current State

	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

Current State

- 0.0.1 First Prototype (May 2015)
  - Hadoop MapReduce and Giraph for operator implementations
  - Too much complexity
  - Performance loss through serialization in HDFS/HBase
- 0.0.2 Using Flink as execution layer (June 2015)
  - Basic operators
- 0.1 December 2015
  - System-side identifiers (UUID)
  - Improved property handling
  - More operator implementations (e.g., Equality, Bool operators)
  - Code refactoring
- 0.2-SNAPSHOT
  - Graph Pattern Matching
  - Frequent Subgraph Mining
  - Memory optimization (96-bit ID, Dictionary Encoding, …)
  - Tuple Implementation

Contributions to Flink

- FLINK-2411 Add basic graph summarization algorithm
- FLINK-2590 DataSetUtils.zipWithUniqueID creates duplicate Ids
- FLINK-2905 Add intersect method to Graph class
- FLINK-2910 Combine tests for binary graph operators
- *FLINK-2941 Implement a neo4j - Flink/Gelly connector*
- FLINK-2981 Update README for building docs
- FLINK-3064 Missing size check in GroupReduceOperatorBase leads to NPE
- FLINK-3118 Check if MessageFunction implements ResultTypeQueryable
- FLINK-3122 Generalize value type in LabelPropagation
- *FLINK-3272 Generalize vertex value type in ConnectedComponents*

- Flink Forward (October 2015)
- Meetup Big Data Usergroup Saxony (December 2015)

Contributions welcome!

- Code
  - Operator implementations
  - Performance Tuning
  - Extend HBase Storage

- People
  - Bachelor / Master Thesis
  - Open PhD positions in Leipzig, Germany

- **Data!** and Use Cases
  - We are researchers, we assume ...

Thank you!

UNIVERSITÄT LEIPZIG

ScaDS
DRESDEN LEIPZIG

www.gradoop.com

https://flink.apache.org
http://ldbcouncil.org/

http://dbs.uni-leipzig.de/file/GradoopTR.pdf
http://dbs.uni-leipzig.de/file/biiig-vldb2014.pdf

https://github.com/dbs-leipzig/gradoop
https://github.com/s1ck/gdl
https://github.com/s1ck/ldbc-flink-import
https://github.com/s1ck/flink-neo4j

FOSDEM 16 .org
Brussels 30 & 31 January