



UNIVERSITÄT
LEIPZIG



Europäische Union
Europäischer Fonds für
regionale Entwicklung
Europäischer
Sozialfonds

Europa fördert Sachsen.
EFRE
ESF



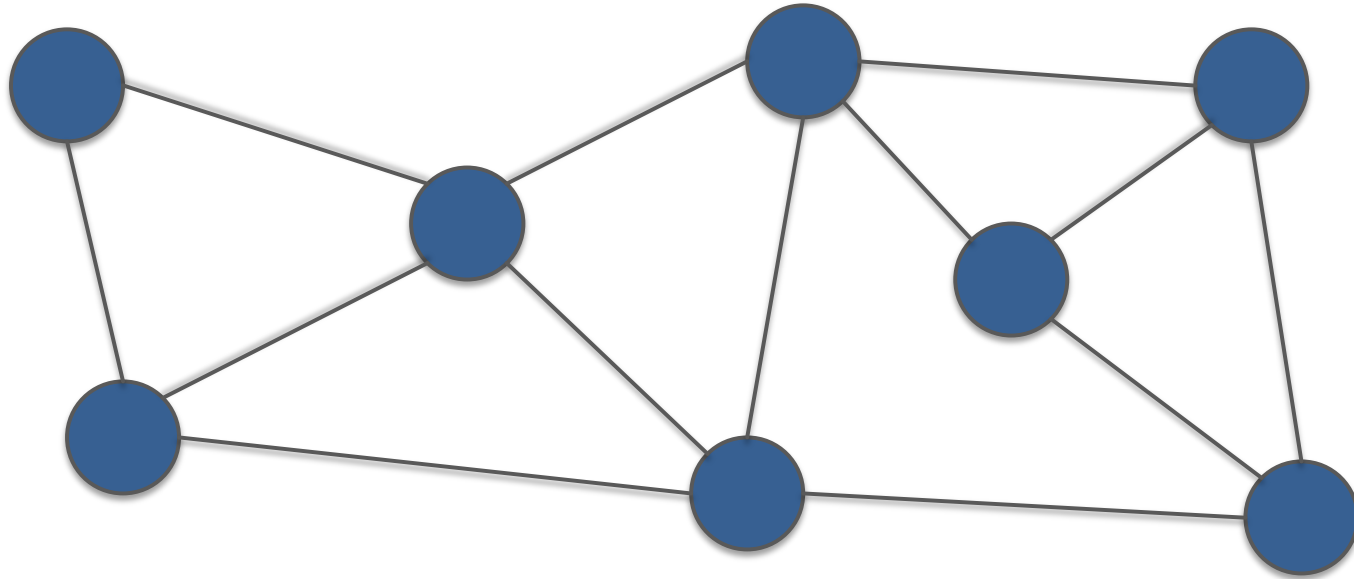
Gradoop - Scalable Graph Analytics with Apache Flink @ GRIDKA 2019

28th August 2019

Kevin Gómez
University of Leipzig

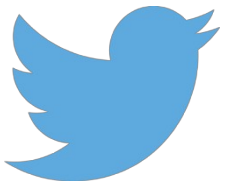
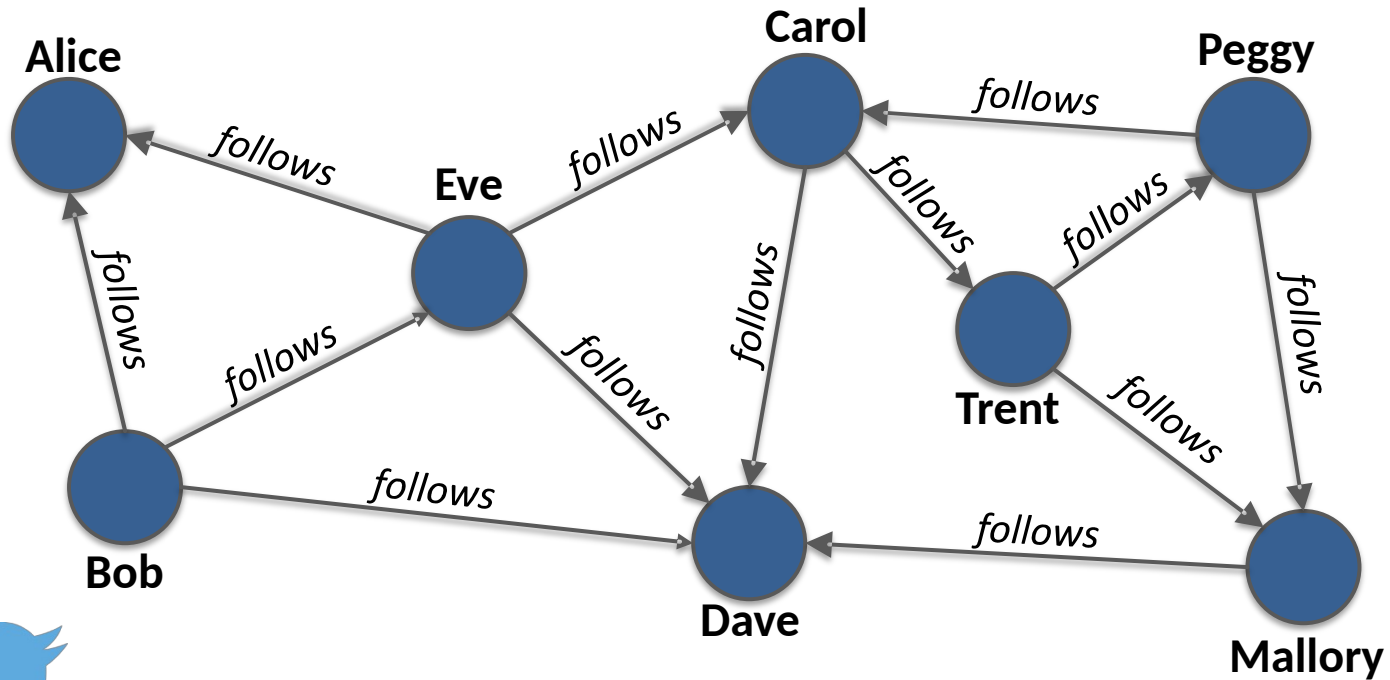
Motivation

„Graphs are everywhere“



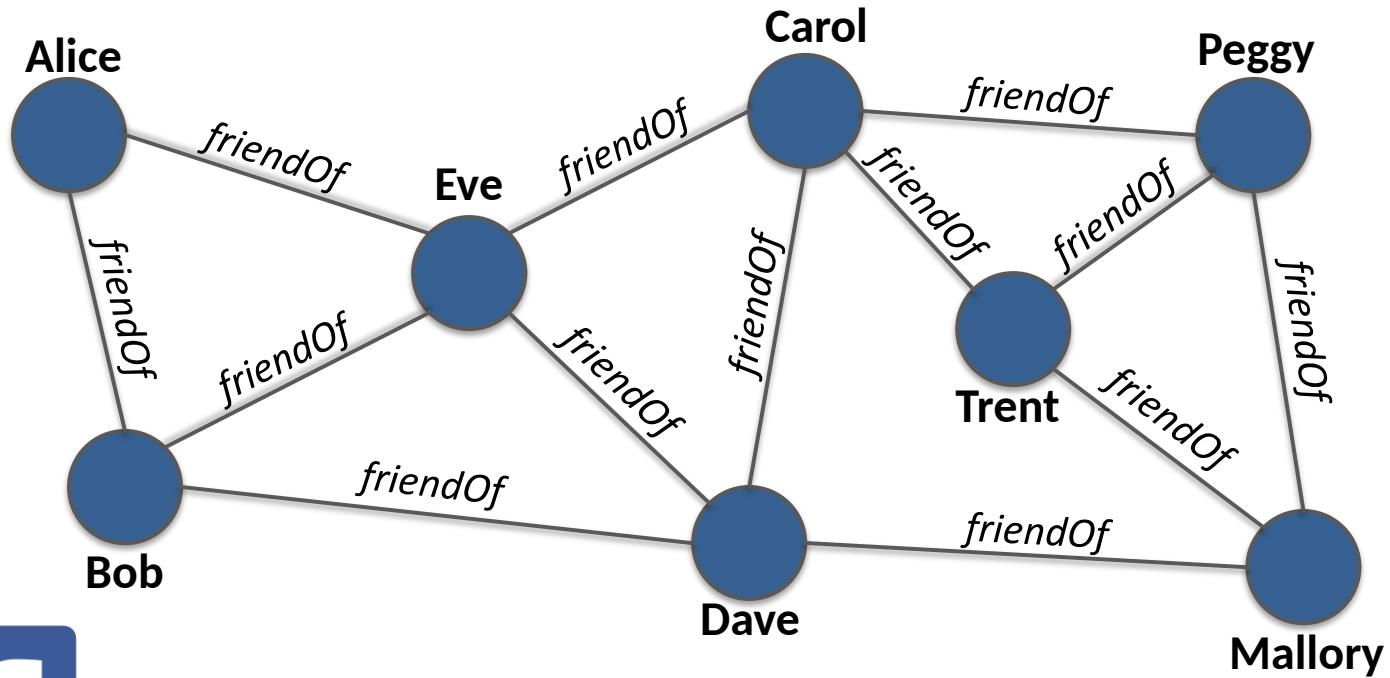
Graph = (*Vertices*, *Edges*)

„Graphs are everywhere“



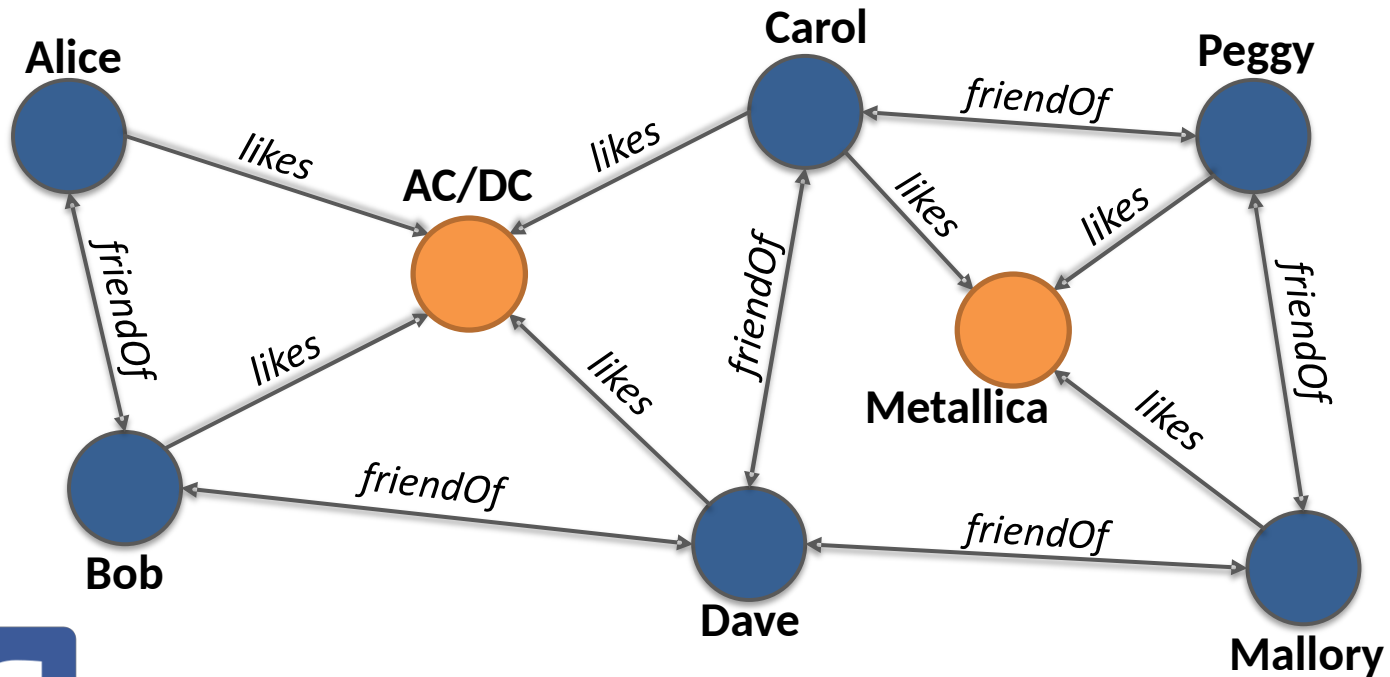
$Graph = (Users, Followers)$

„Graphs are everywhere“



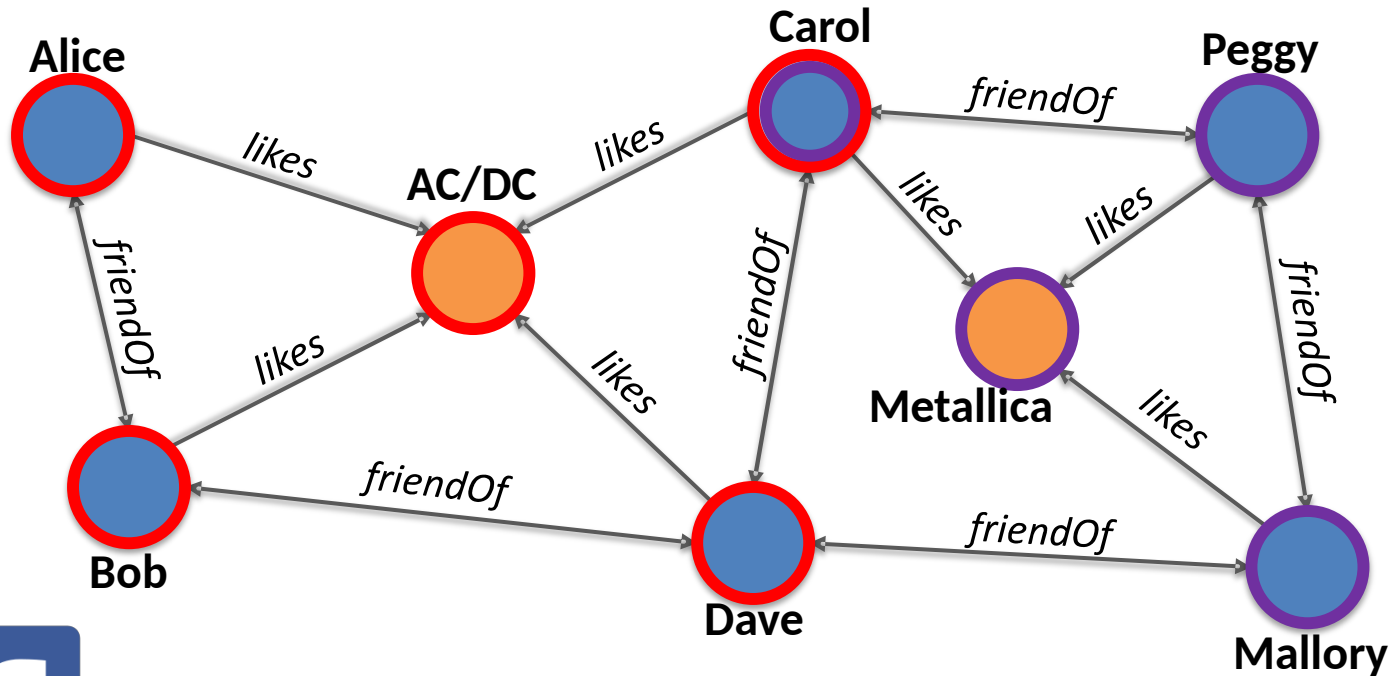
$Graph = (Users, Friendships)$

„Graphs are heterogeneous“



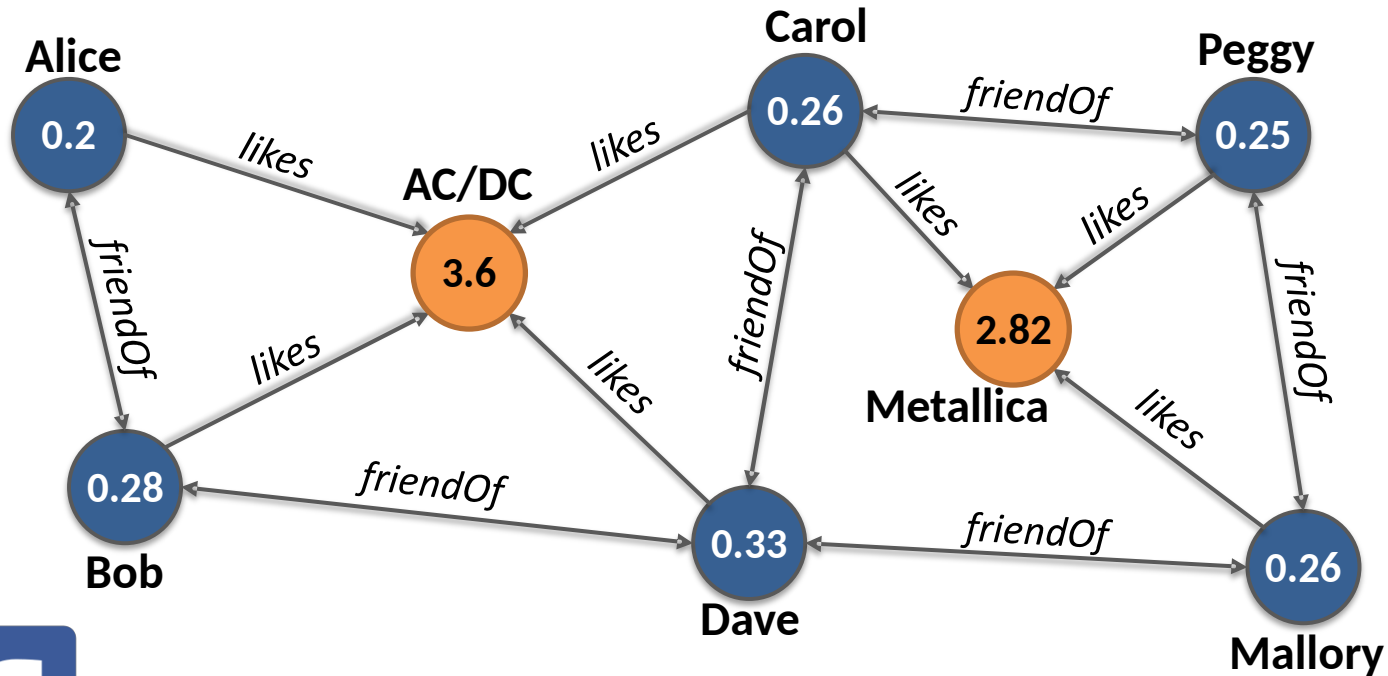
$Graph = (Users \cup Bands, Friendships \cup Likes)$

„Graphs can be analyzed“



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

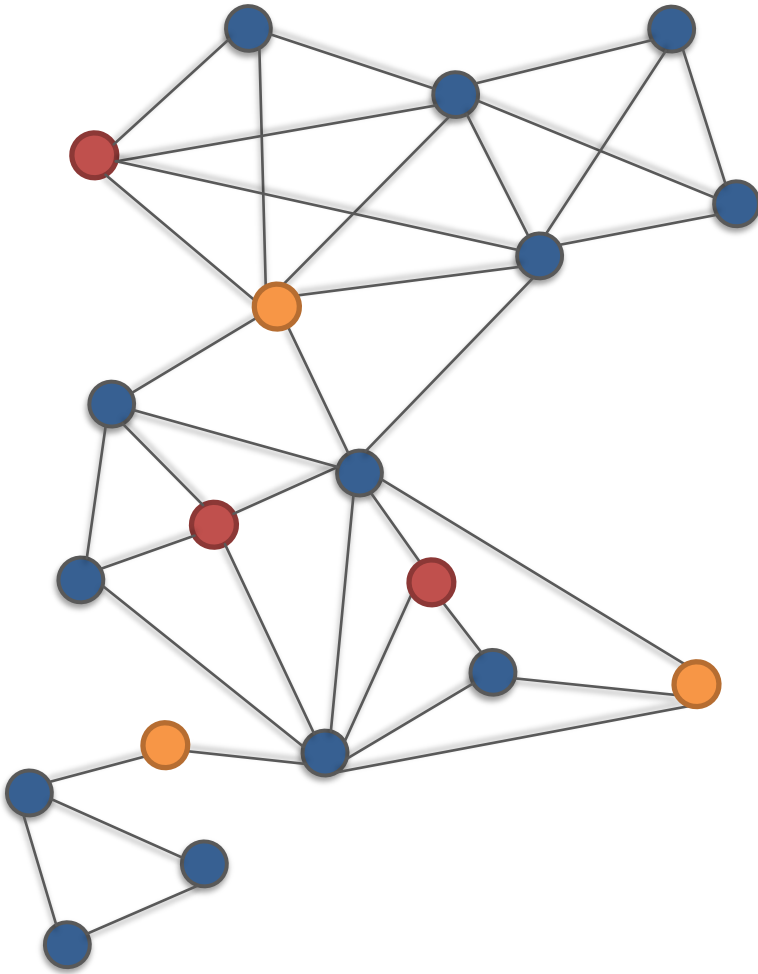
„Graphs can be analyzed“



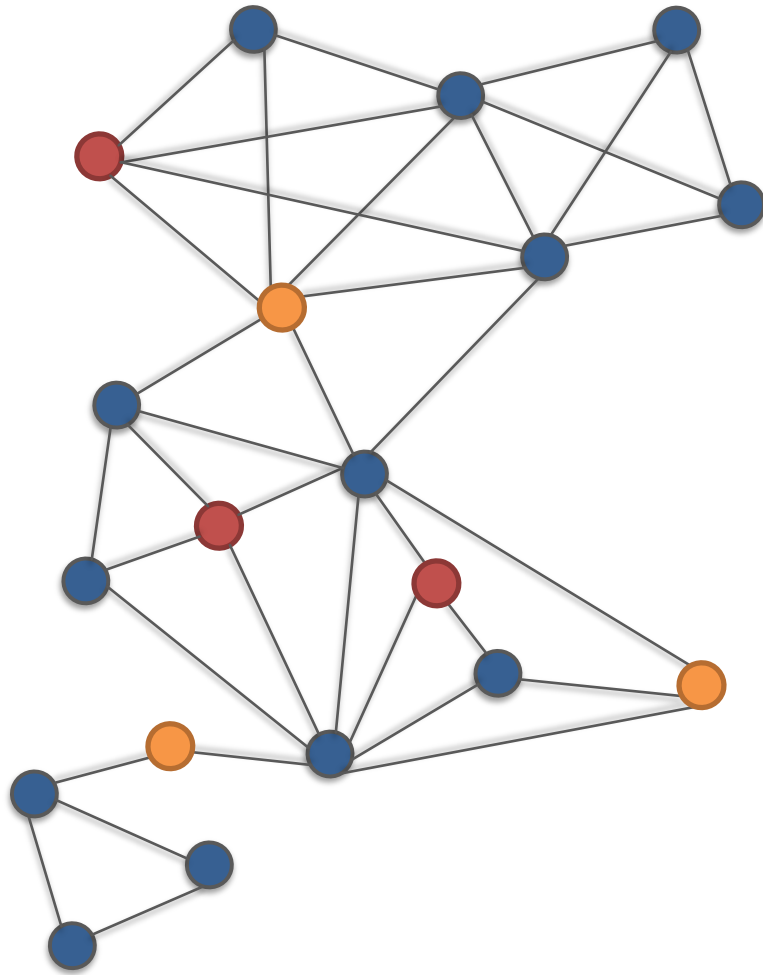
$Graph = (Users \cup Bands, Friendships \cup Likes)$

„Graphs can be analyzed“

Assuming a social network



„Graphs can be analyzed“



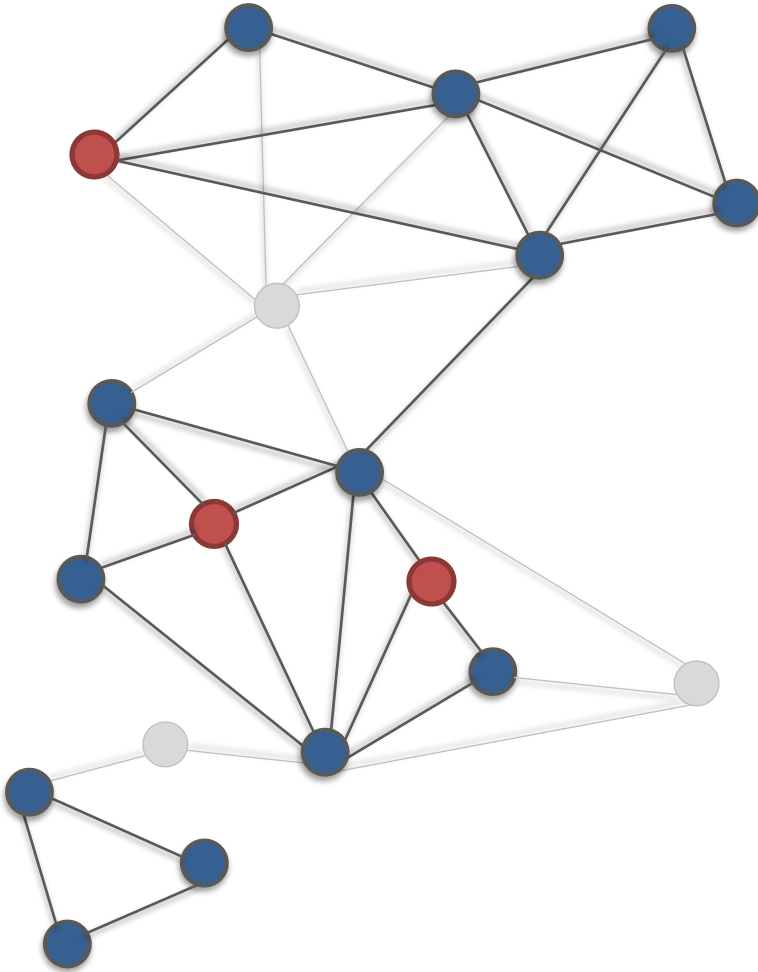
Assuming a social network

1. Determine subgraph

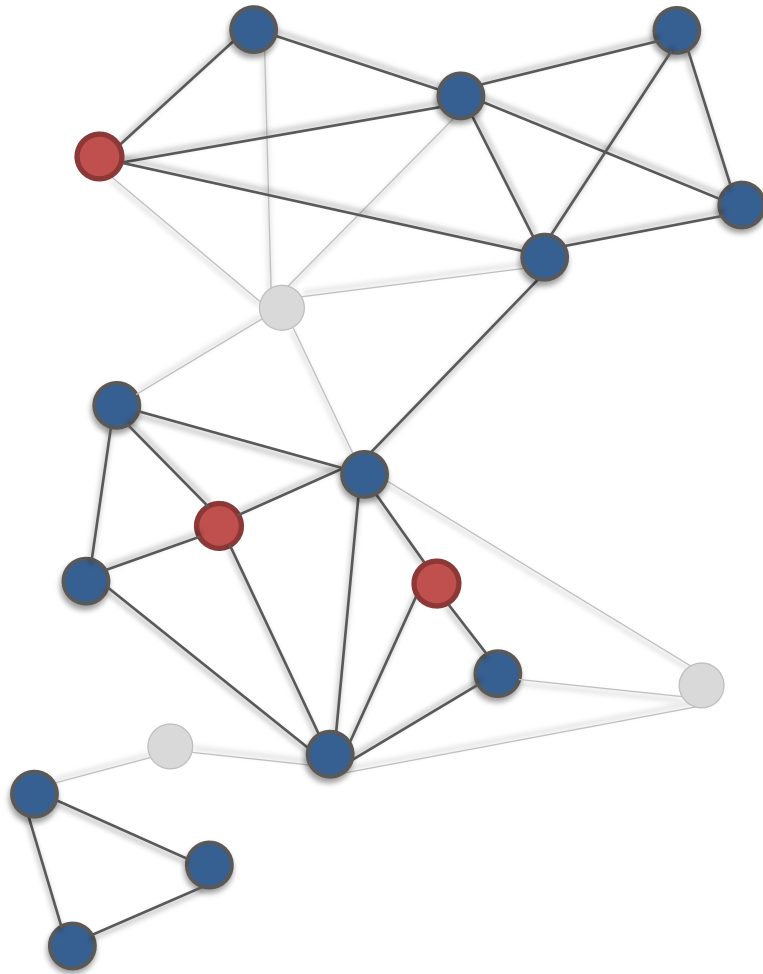
„Graphs can be analyzed“

Assuming a social network

1. Determine subgraph



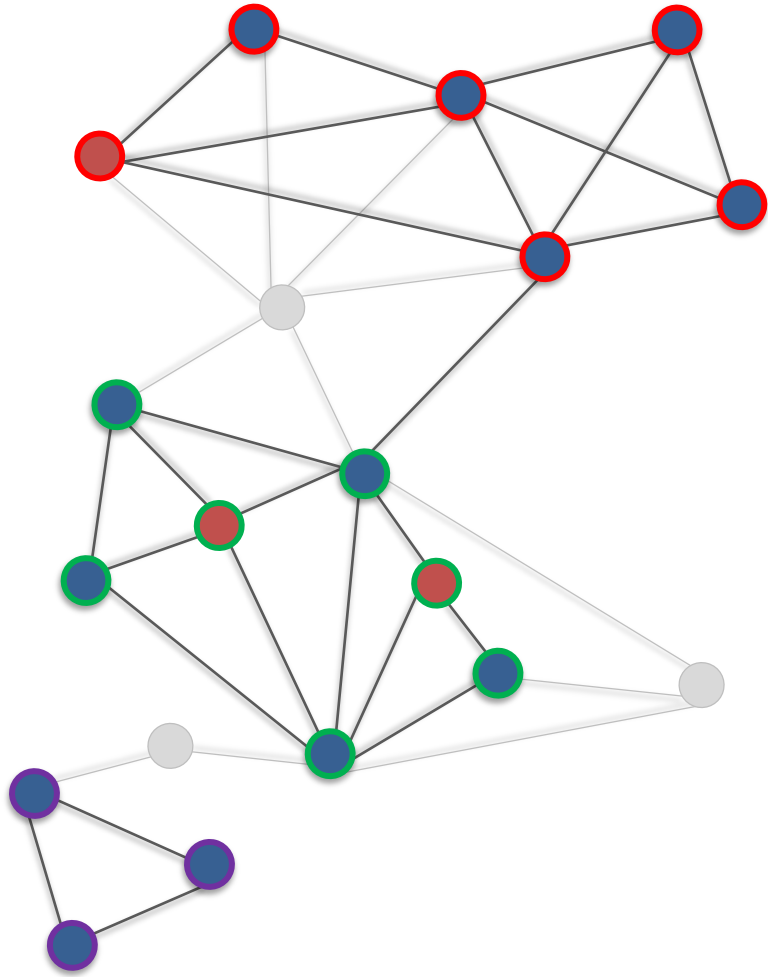
„Graphs can be analyzed“



Assuming a social network

1. Determine subgraph
2. Find communities

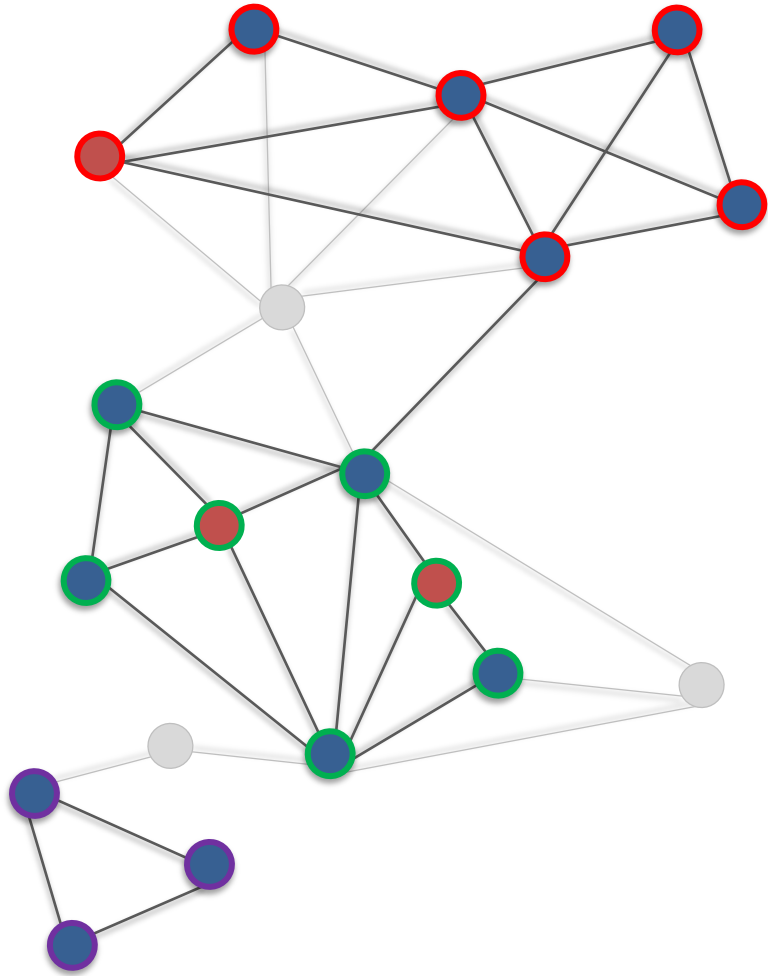
„Graphs can be analyzed“



Assuming a social network

1. Determine subgraph
2. Find communities

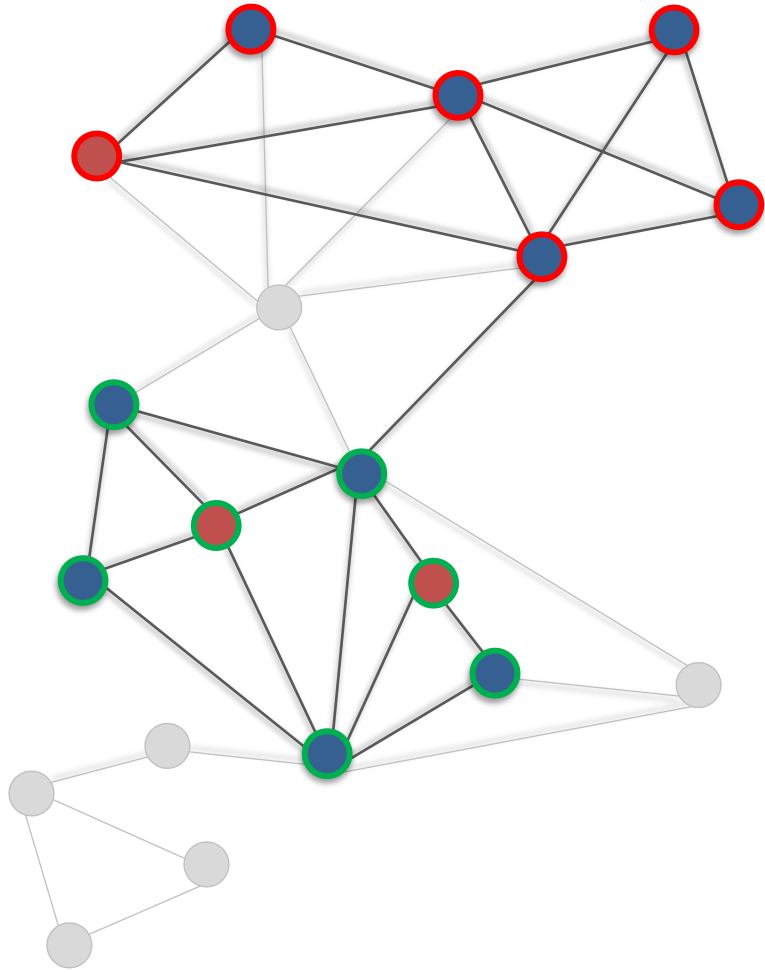
„Graphs can be analyzed“



Assuming a social network

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

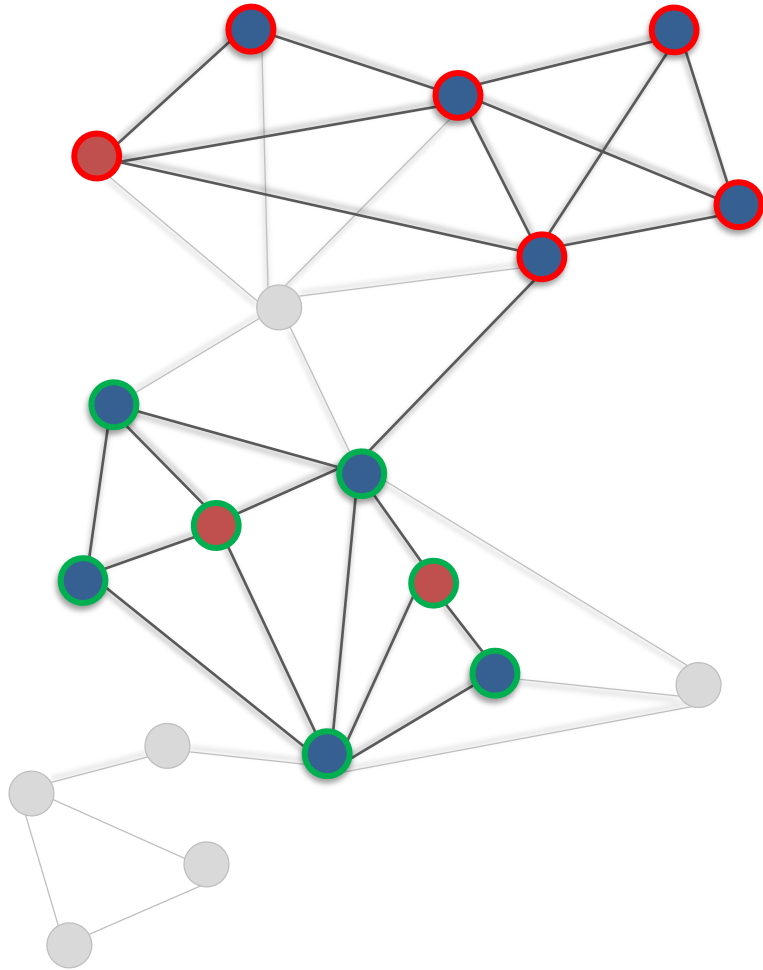
„Graphs can be analyzed“



Assuming a social network

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

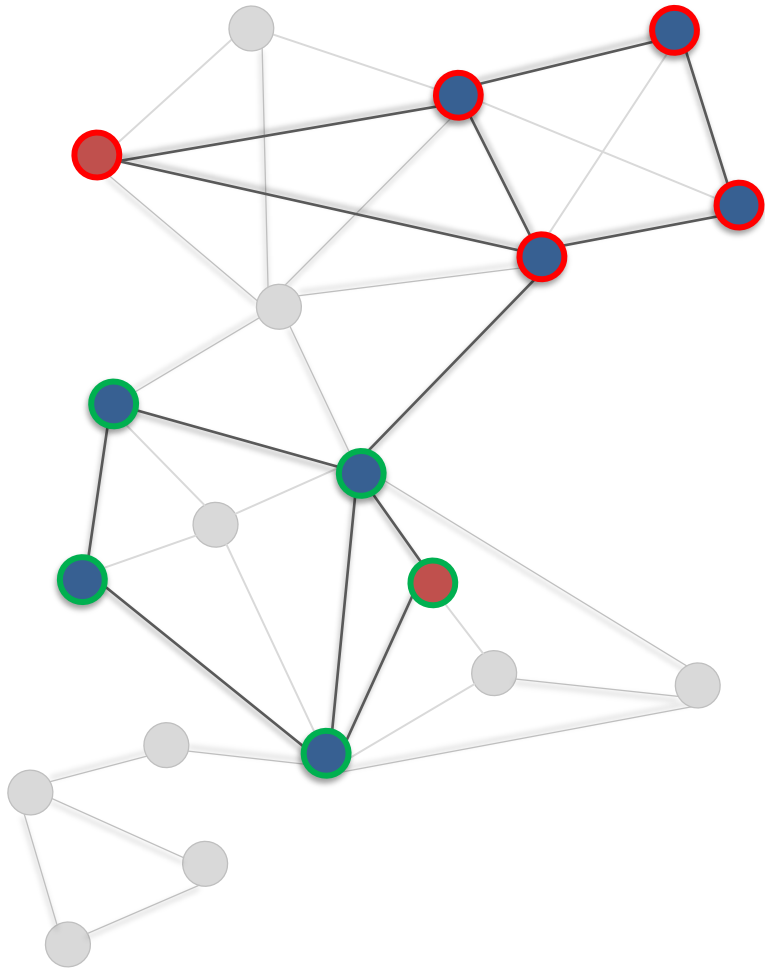
„Graphs can be analyzed“



Assuming a social network

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

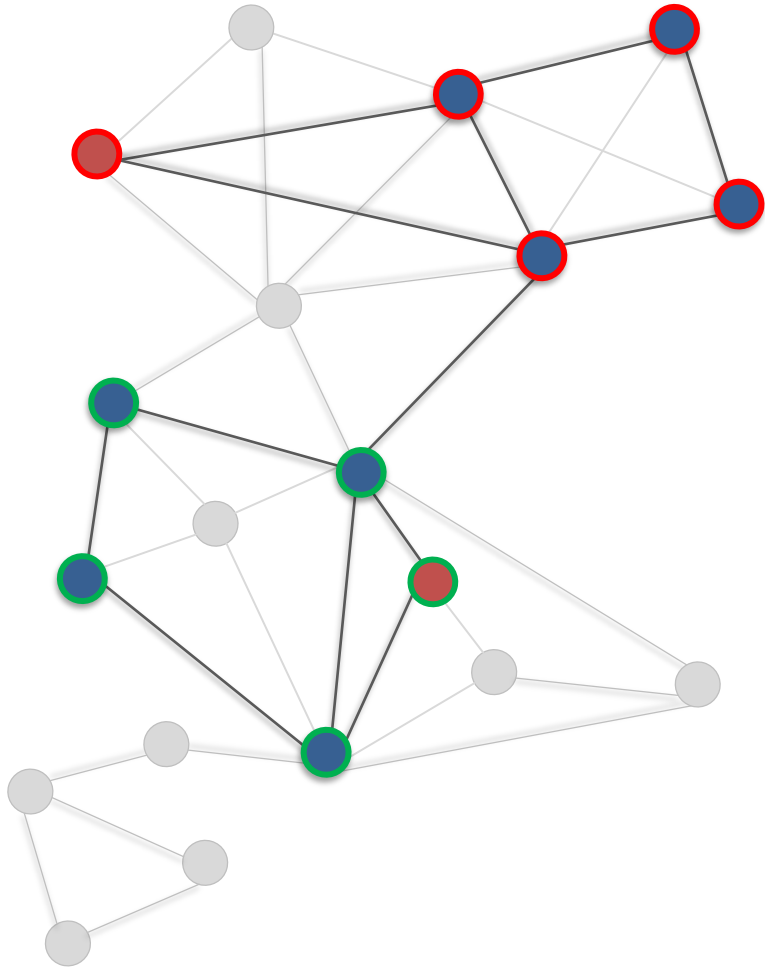
„Graphs can be analyzed“



Assuming a social network

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

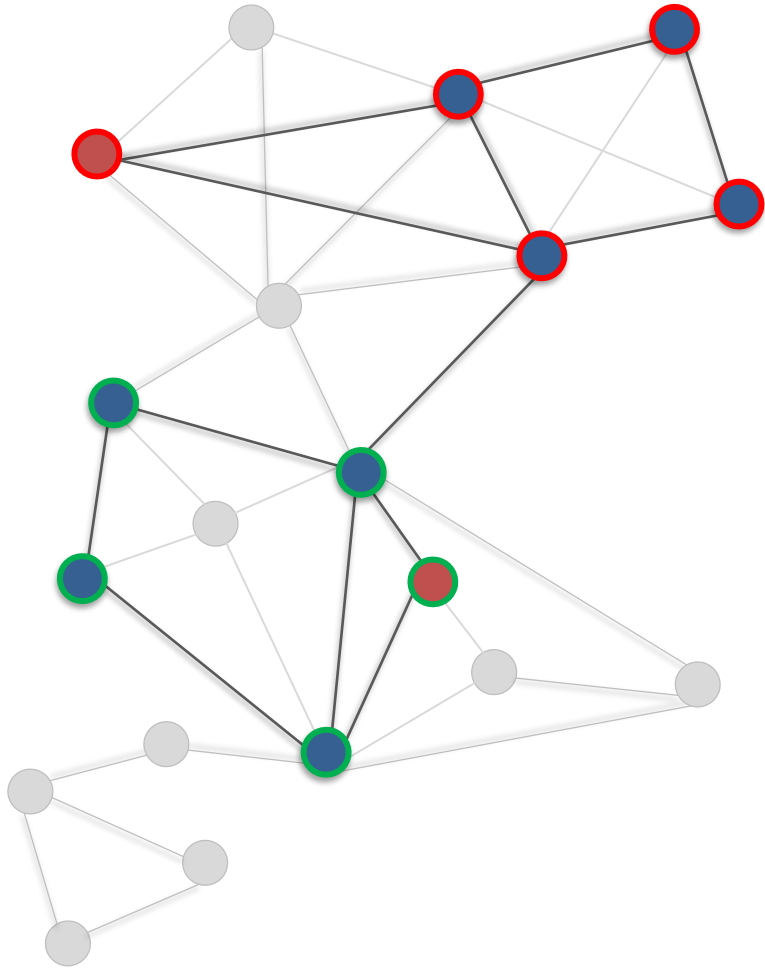
„Graphs can be analyzed“



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 algorithm**

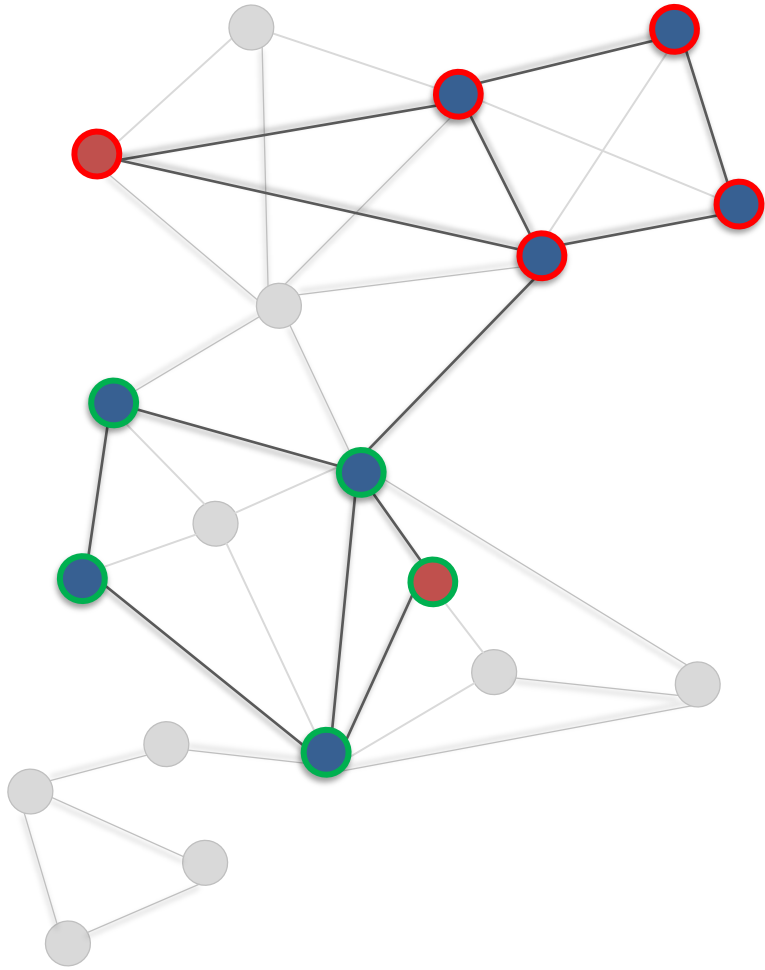
„Graphs can be analyzed“



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 algorithm

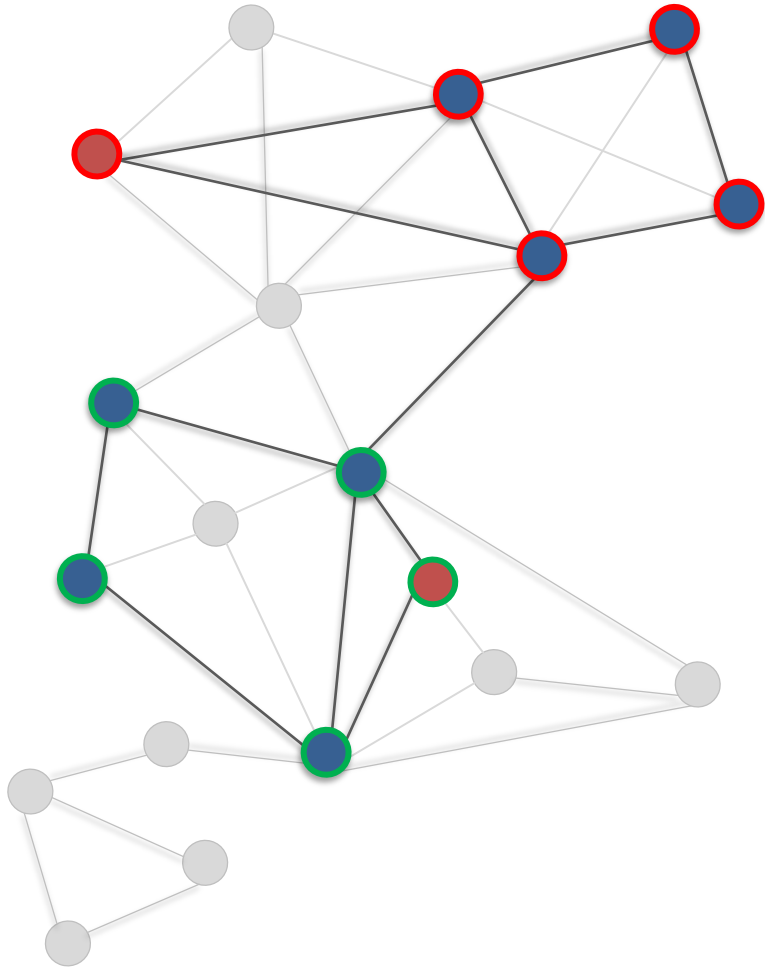
„Graphs can be analyzed“



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 algorithm

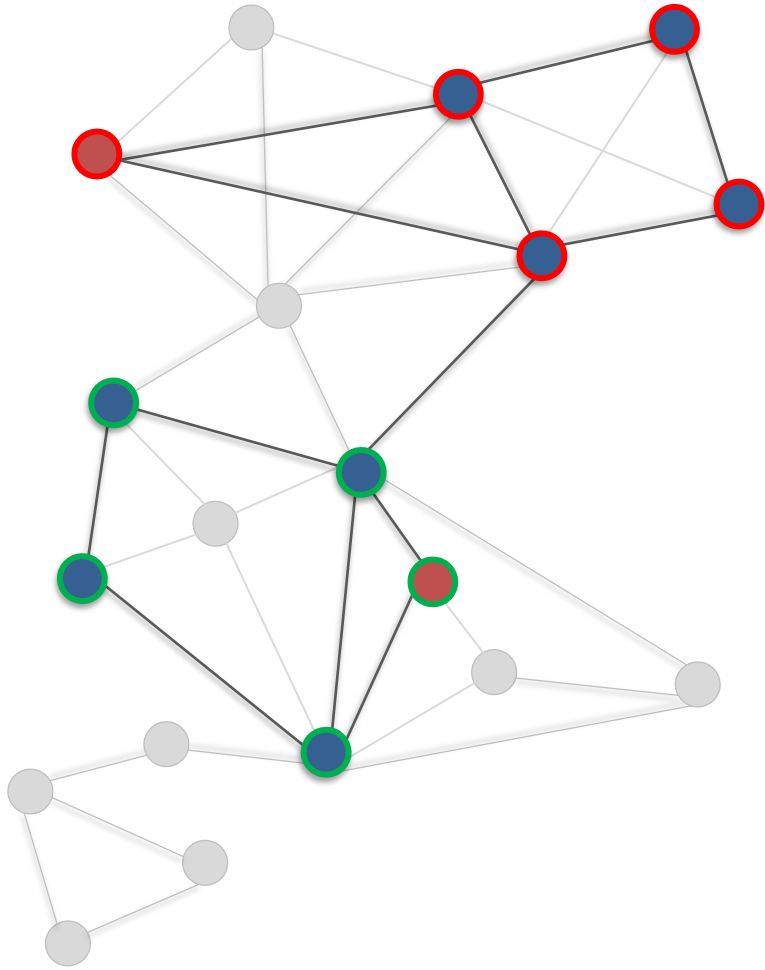
„Graphs can be analyzed“



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 algorithm

„Graphs can be analyzed“



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 algorithm

„And let's not forget ...“

“...Graphs are large”



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



High Level Architecture



Graph Analytical Language (GrALa)

👉 Java 8

Extended Property Graph Model (EPGM)

👉 ALv2

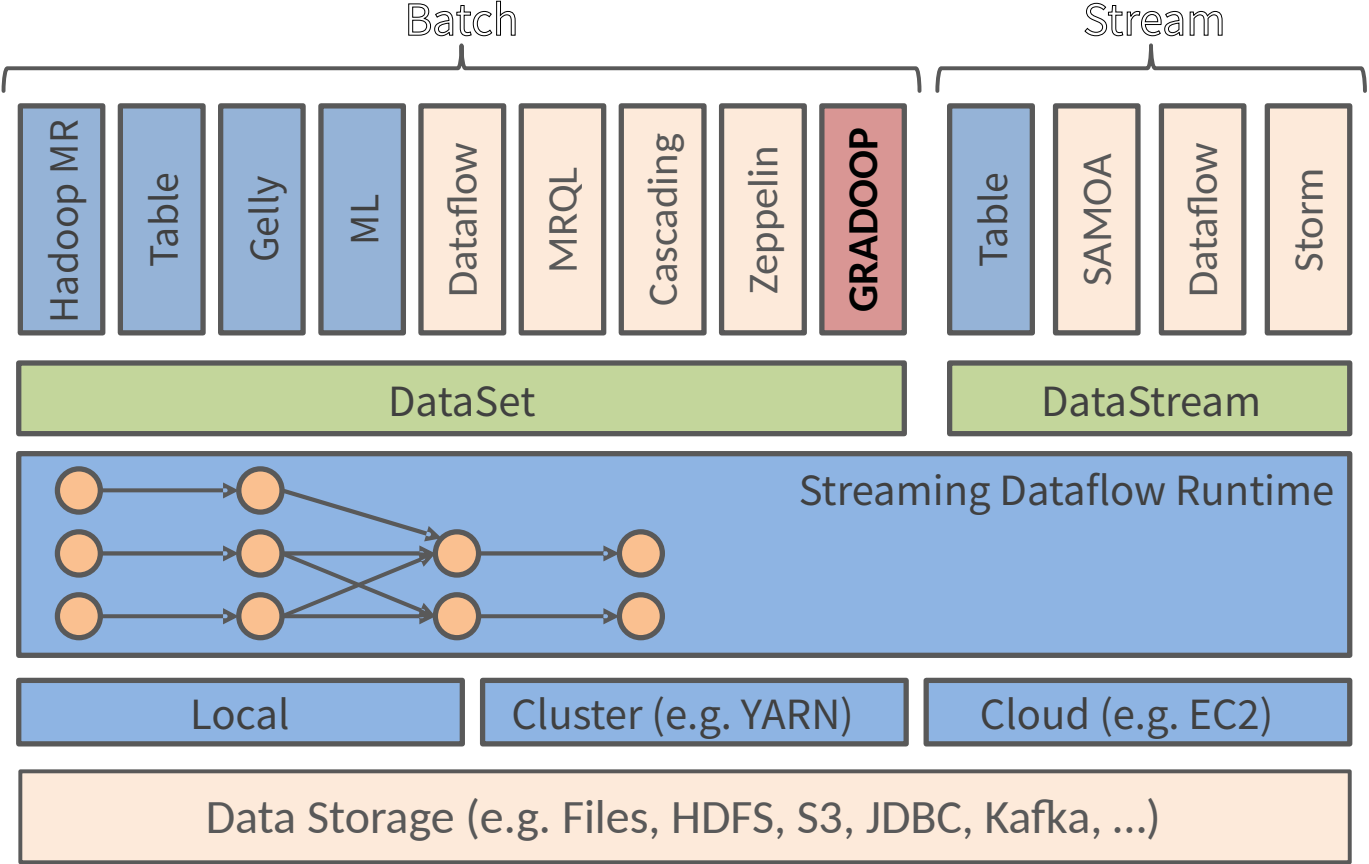
Apache Flink Operator Implementation

Apache Flink Distributed Operator Execution

Apache HBase Distributed Graph Store

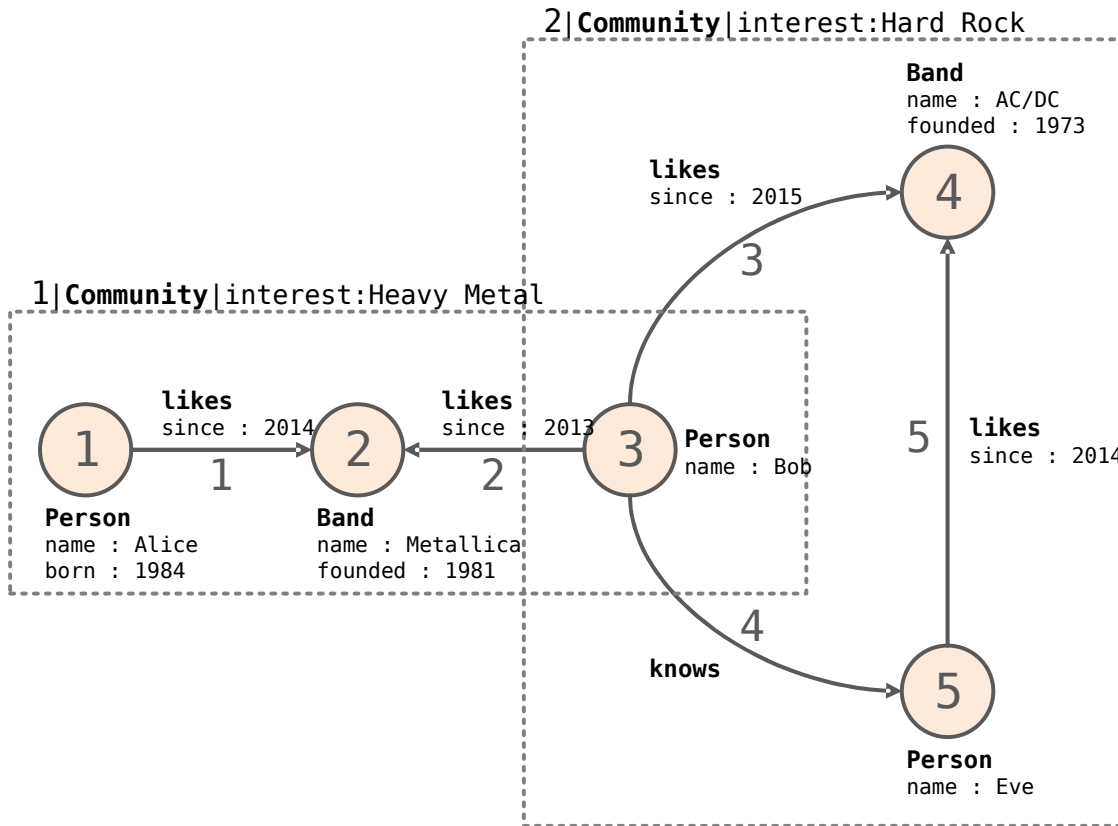


Apache Flink Third-party library



Extended Property Graph Model (EPGM)

Extended Property Graph Model



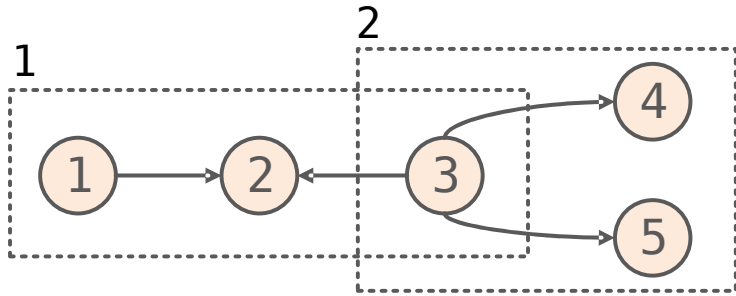
- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- Properties

Why multiple graph support?

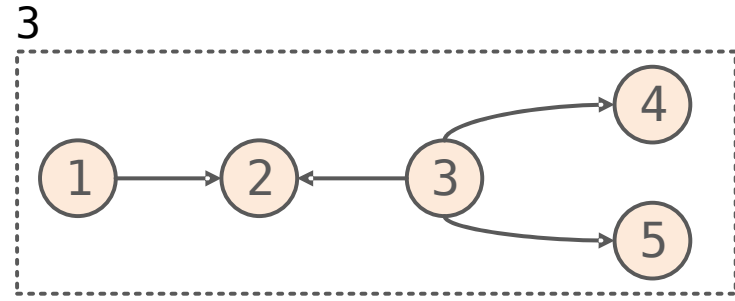
- It allows reasoning over multiple versions of the same graph (e.g. comparing daily snapshots)
- It provides an effective grouping mechanism for naturally-partitioned data (e.g. per continent graph)
- It is useful for combining data from disparate data sources in one system (e.g. data integration)
- It fits the paradigm of prominent analytical big-data processing systems (e.g. Apache Flink and Apache Spark)
- It mirrors mathematical graph theory where working with multiple graphs is common

EPGM Operators

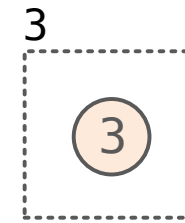
Basic Binary Operators



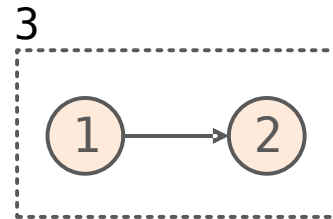
Combination



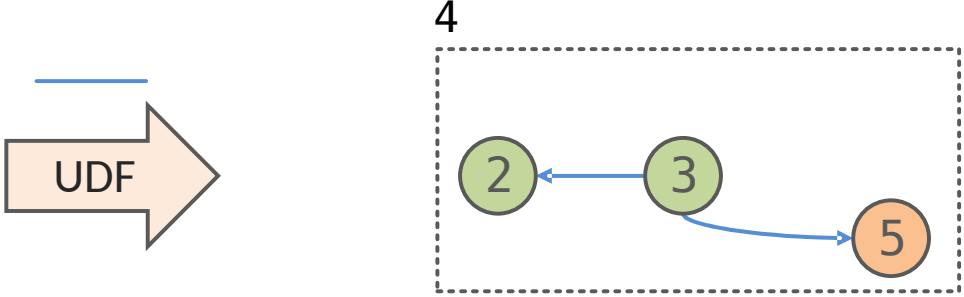
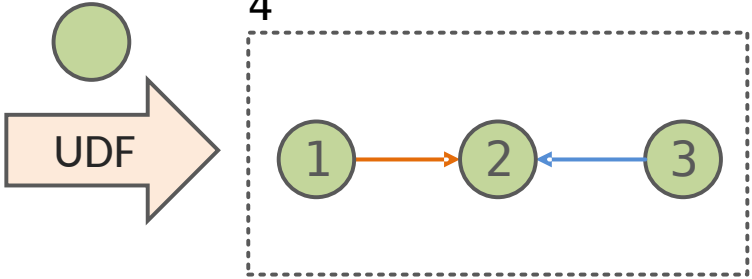
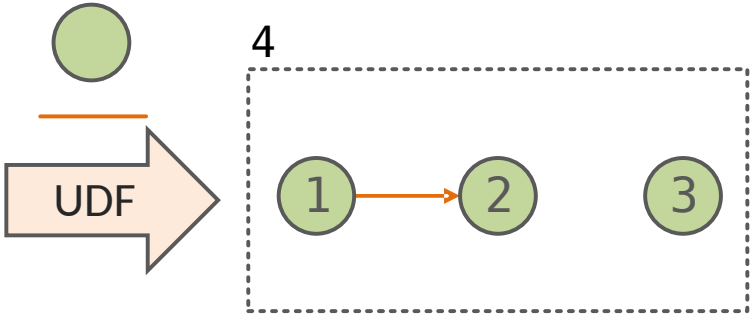
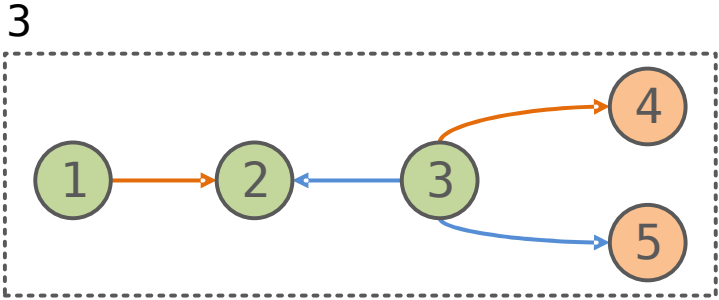
Overlap



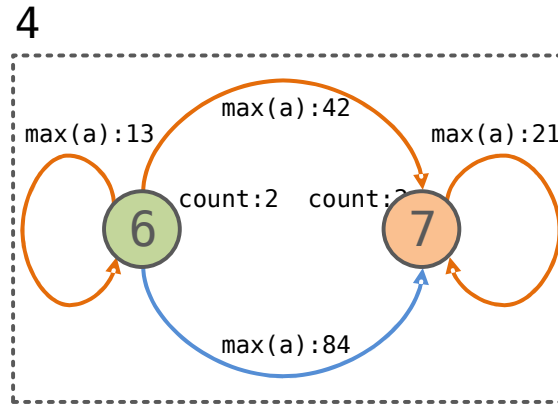
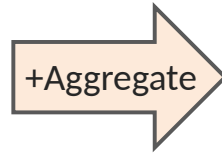
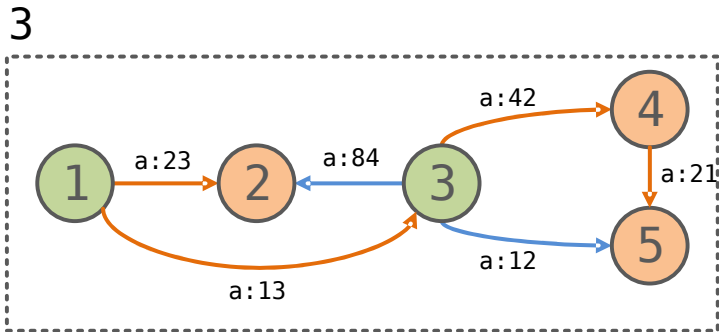
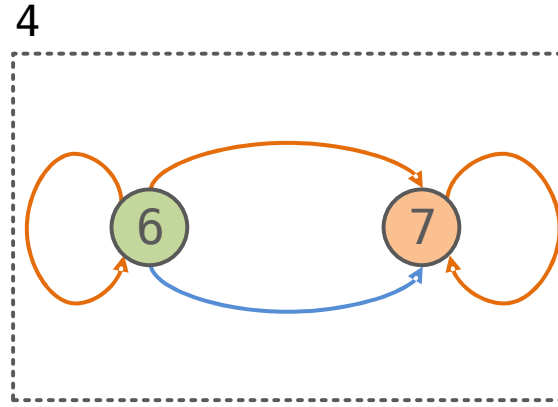
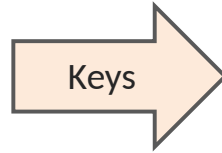
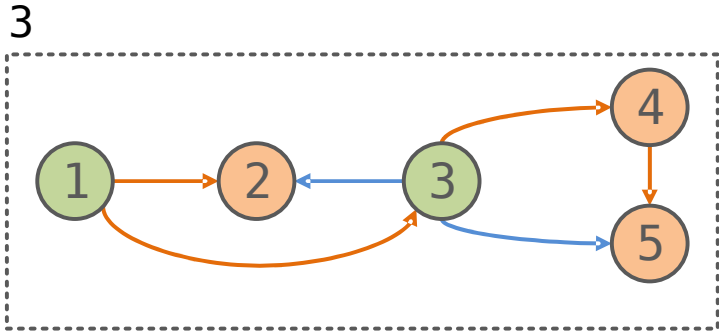
Exclusion



Subgraph Extraction

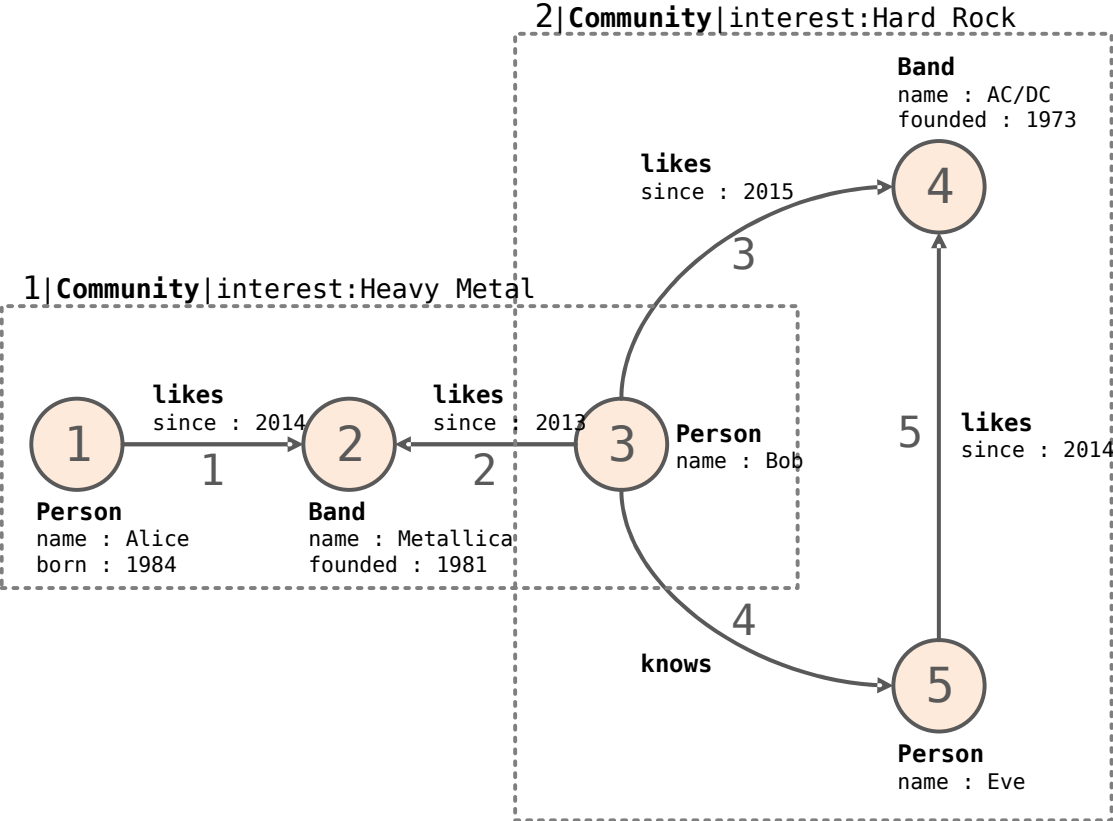


Graph Grouping



Cypher Pattern Matching

Which people like the same band that was founded after 1980?
Would they possibly become buddies?

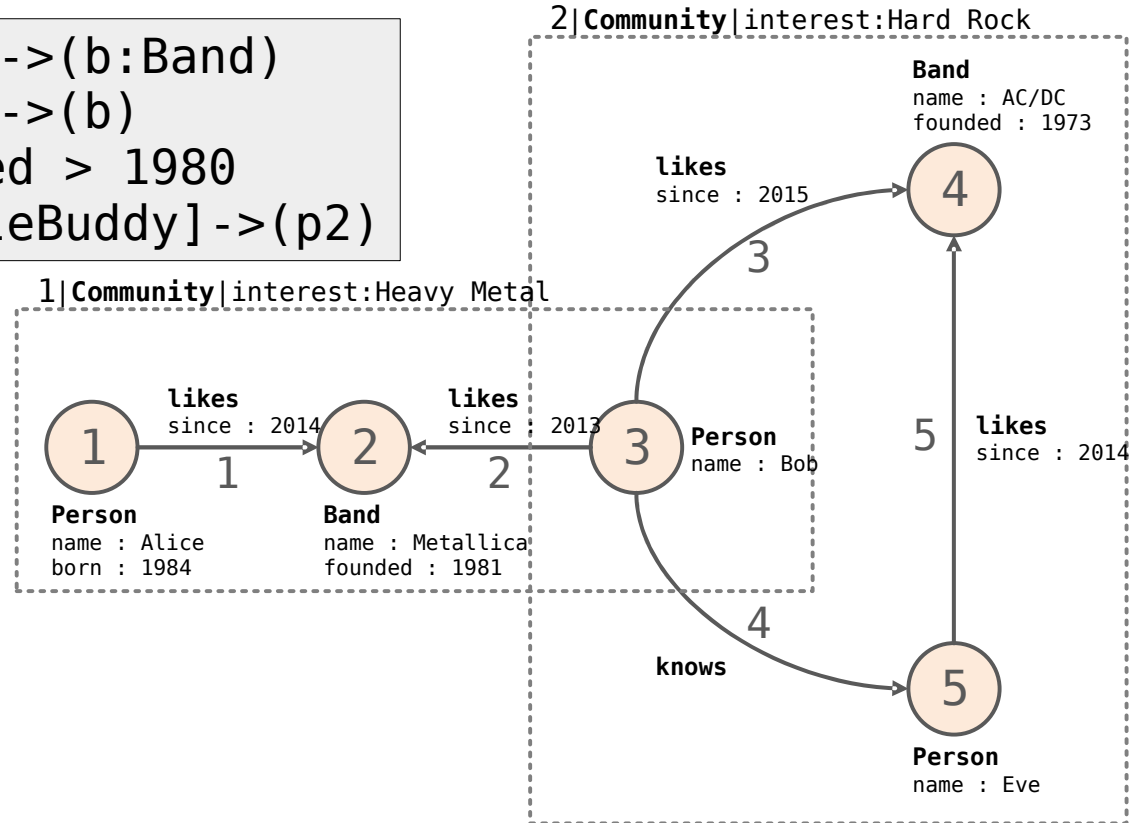


Cypher Pattern Matching

```
MATCH (p1:Person)-[:likes]->(b:Band)
      (p2:Person)-[:likes]->(b)
WHERE p1 != p2 AND b.founded > 1980
CONSTRUCT (p1)-[new:possibleBuddy]->(p2)
```

Which people like
the same band
that was founded after 1980?

Would they possibly
become buddies?

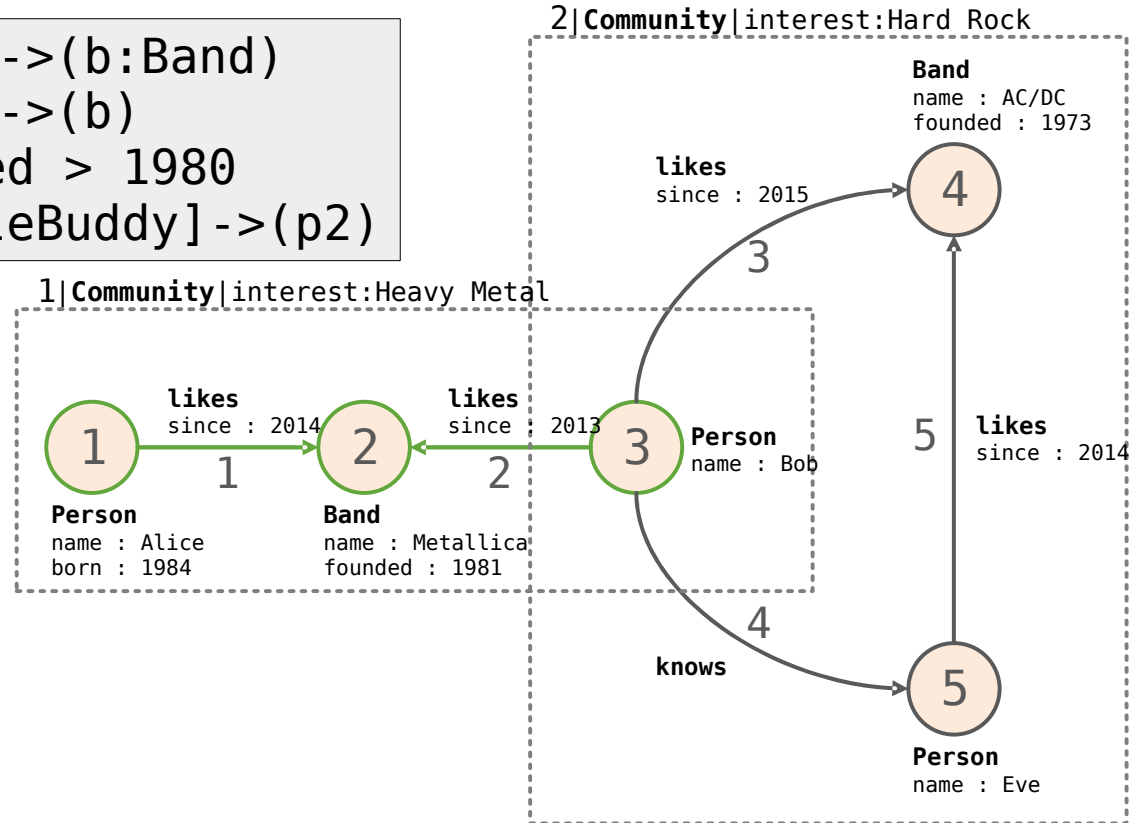


Cypher Pattern Matching

```
MATCH (p1:Person)-[:likes]->(b:Band)
      (p2:Person)-[:likes]->(b)
WHERE p1 != p2 AND b.founded > 1980
CONSTRUCT (p1)-[new:possibleBuddy]->(p2)
```

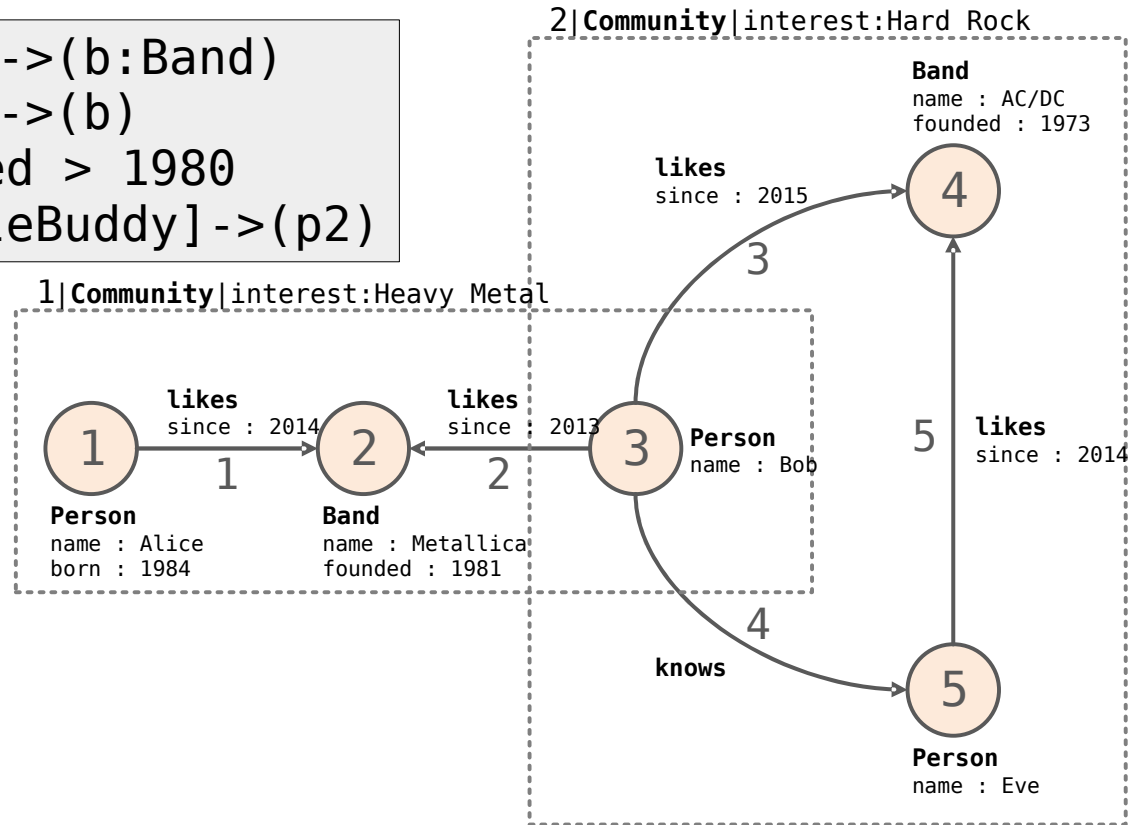
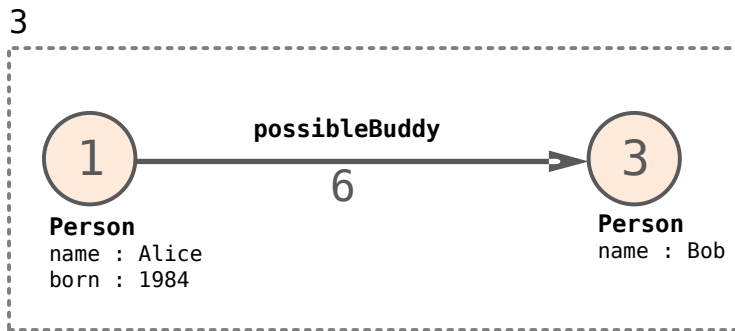
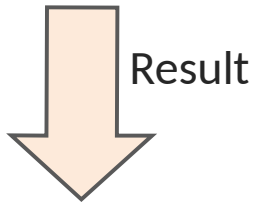
Which people like
the same band
that was founded after 1980?

Would they possibly
become buddies?

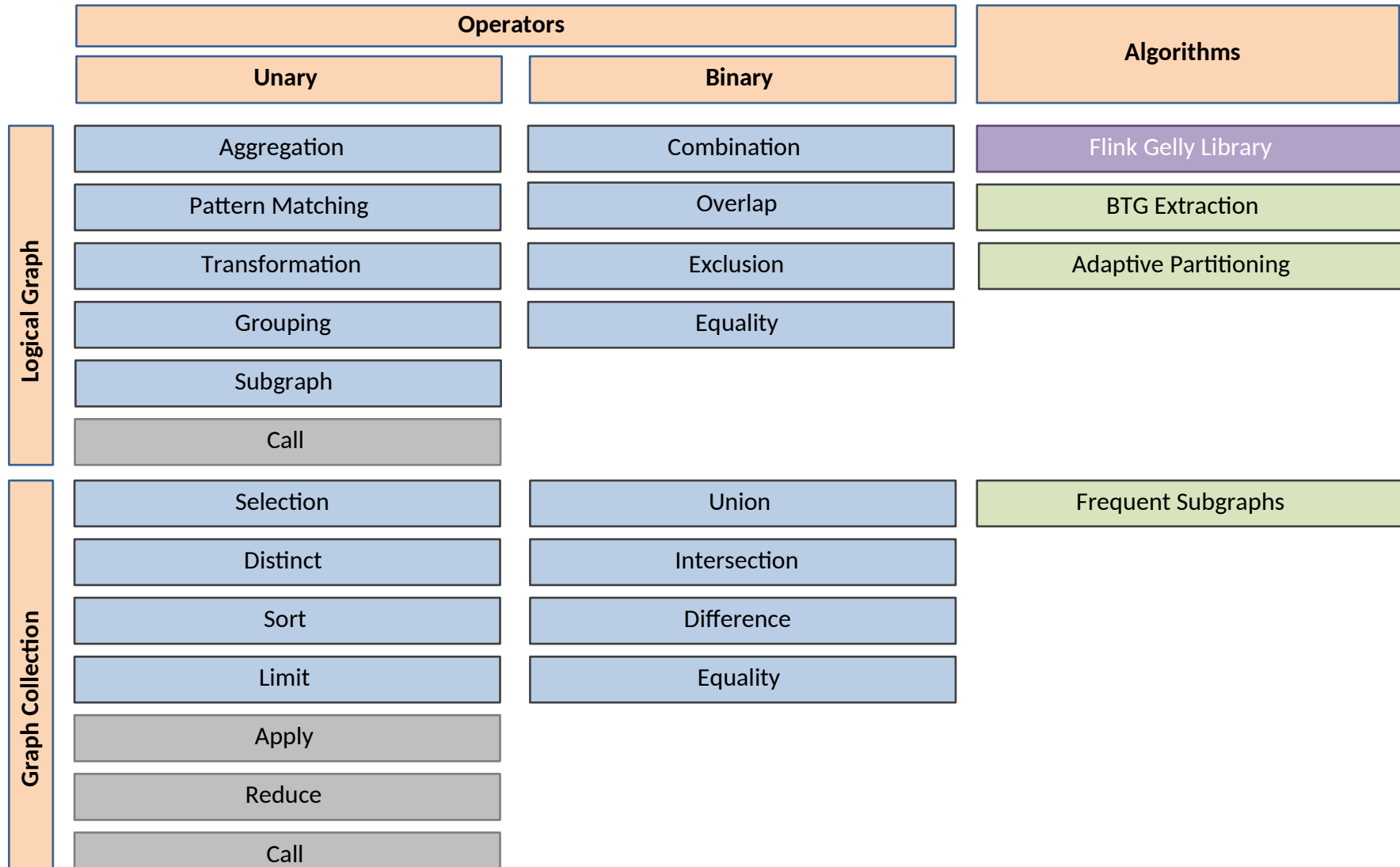


Cypher Pattern Matching

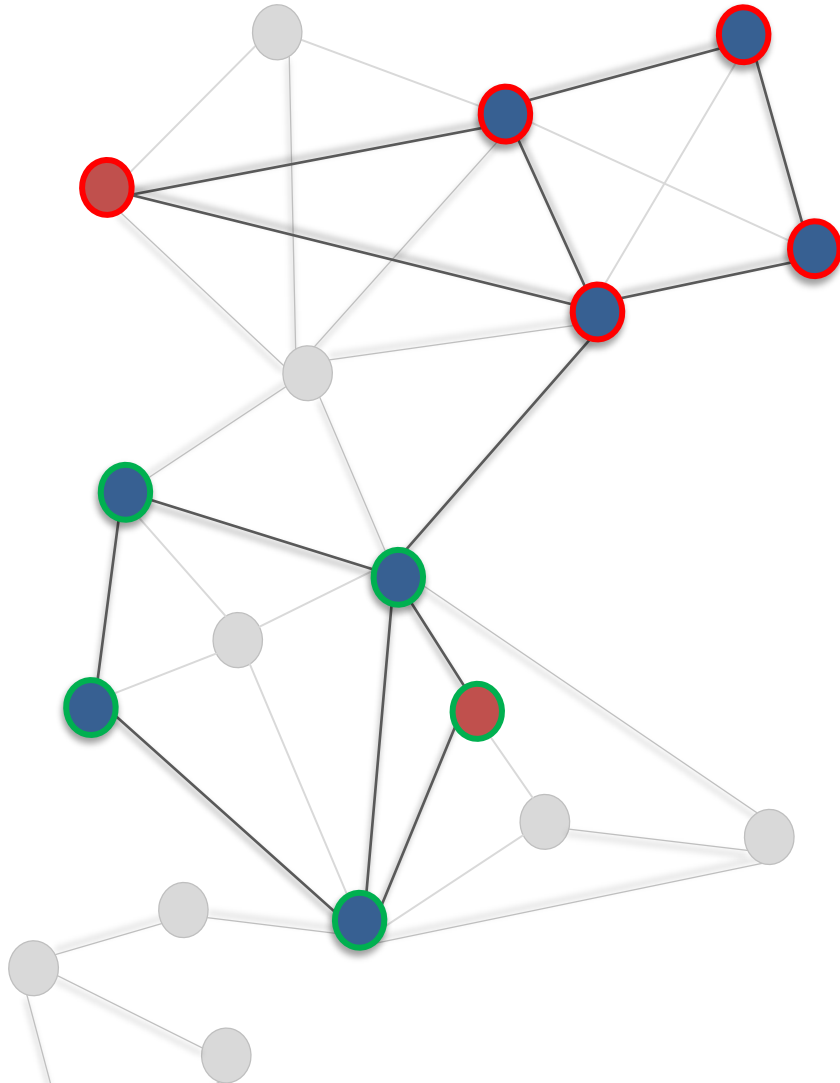
```
MATCH (p1:Person)-[:likes]->(b:Band)
      (p2:Person)-[:likes]->(b)
WHERE p1 != p2 AND b.founded > 1980
CONSTRUCT (p1)-[new:possibleBuddy]->(p2)
```



EPGM Operators Overview



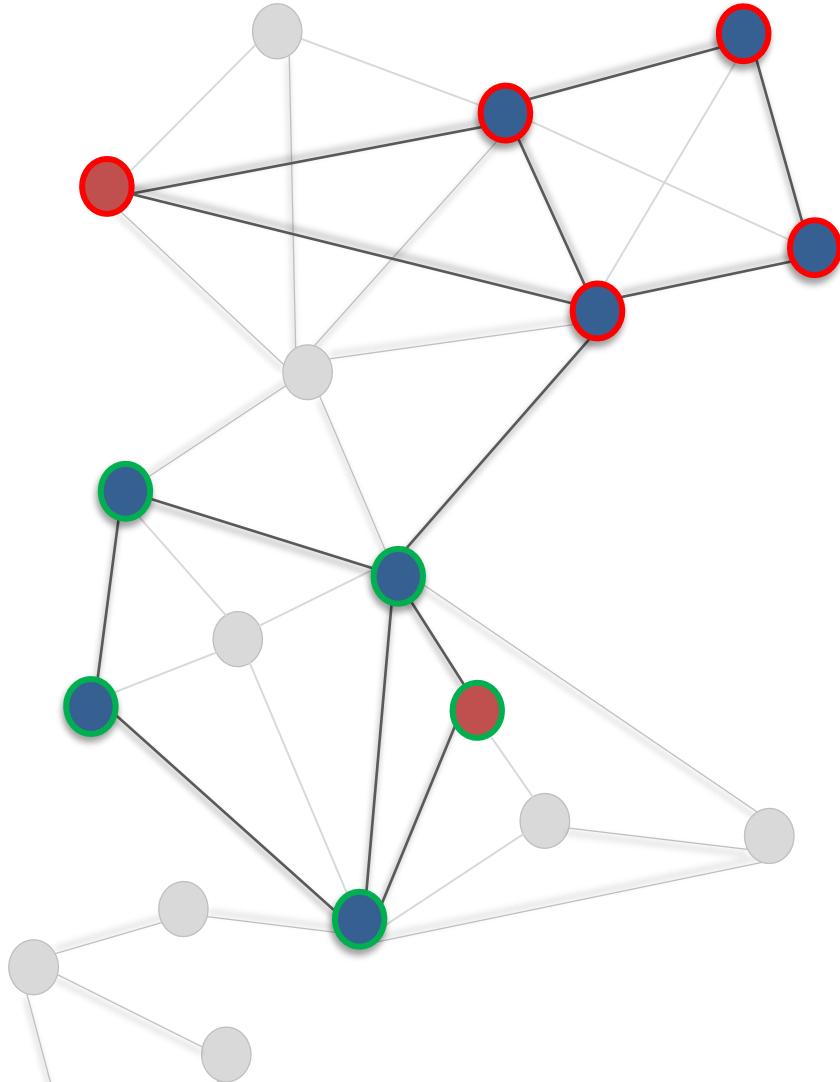
„Graphs can be analyzed“



Assuming a social network

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

„Graphs can be analyzed“

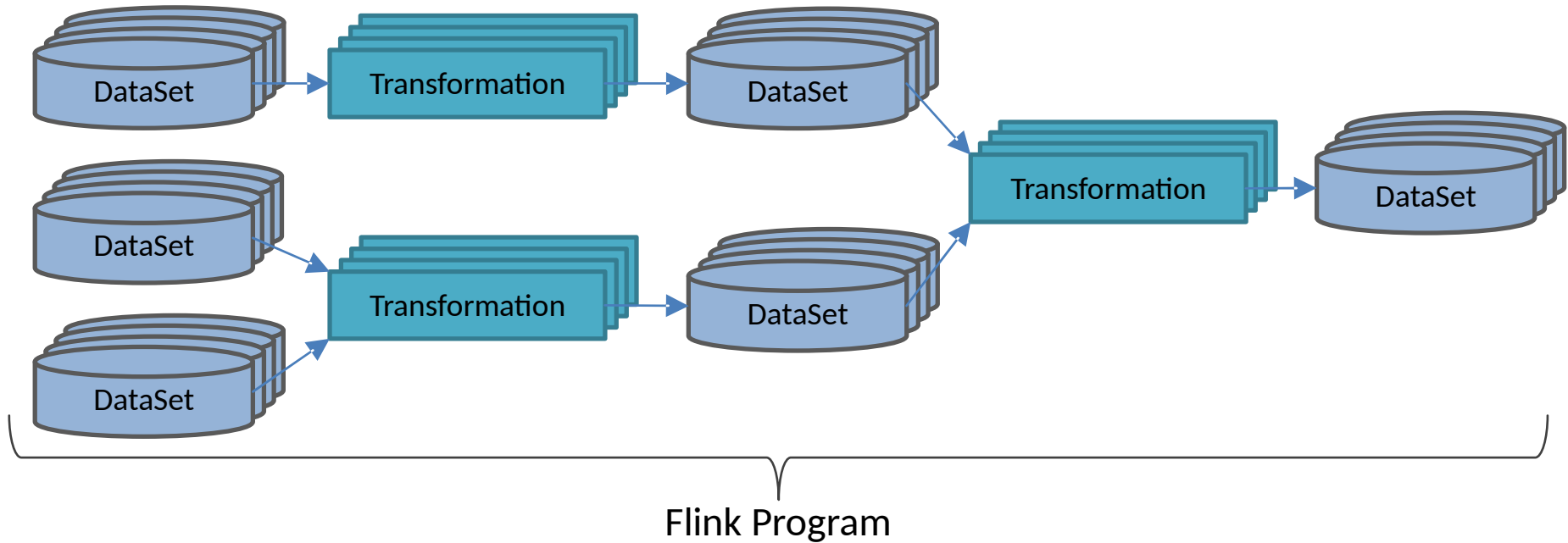


```
return socialNetwork
return socialNetwork
// 1) extract subgraph
.subgraph((vertex) -> {
    return vertex.getLabel().toLowerCase().equals(person);
}, (edge) -> { return edge.getLabel().toLowerCase().equals(knows); })
// project to necessary information
.transform((current, transformed) -> { return current; }, (current, transformed) -> {
    transformed.setLabel(current.getLabel());
    transformed.setProperty(city, current.getPropertyValue(city));
    transformed.setProperty(gender, current.getPropertyValue(gender));
    transformed.setProperty(label, current.getPropertyValue(birthday));
    return transformed;
}, (current, transformed) -> {
    transformed.setLabel(current.getLabel());
    return transformed;
})
// 3a) compute communities
.callForGraph(new GellyLabelPropagation<GraphHeadPojo, VertexPojo, EdgePojo>(maxIterations, label))
// 3b) separate communities
.splitBy(label)
// 4) compute vertex count per community
.apply(new ApplyAggregation<>(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>()))
// 5) select graphs with more than minClusterSize vertices
.select((g) -> { return g.getPropertyValue(vertexCount).getLong() > threshold; })
// 6) reduce filtered graphs to a single graph using combination
.reduce(new ReduceCombination<GraphHeadPojo, VertexPojo, EdgePojo>())
// 7) group that graph by vertex properties
.groupBy(Lists.newArrayList(city, gender))
// 8a) count vertices of grouped graph
.aggregate(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>())
// 8b) count edges of grouped graph
.aggregate(edgeCount, new EdgeCount<GraphHeadPojo, VertexPojo, EdgePojo>());
```

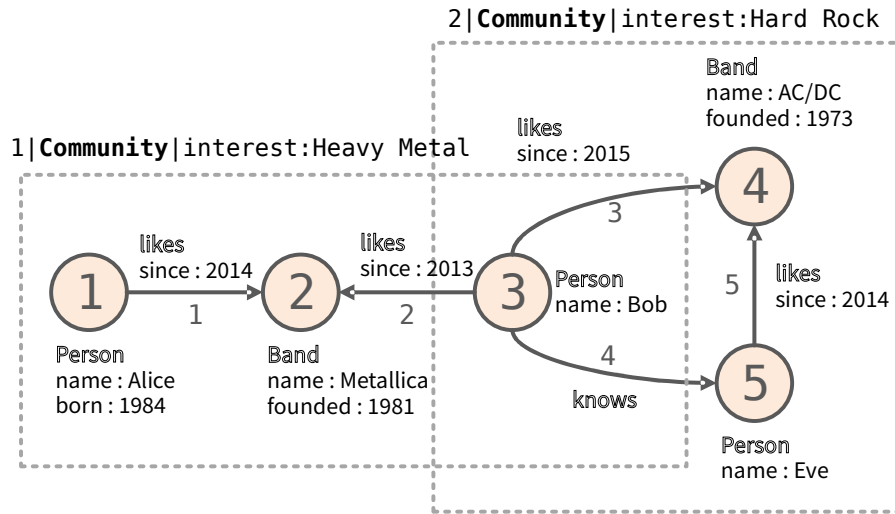
EPGM on Apache Flink

Flink DataSet API

- **DataSet** := Distributed Collection of Data Objects
- **Transformation** := Operation on DataSets
- **Flink Programm** := Composition of Transformations



Graph Representation



DataSet<EPGMVertex>

Id	Label	Properties	Graphs
1	Person	{name:Alice, born:1984}	{1}
2	Band	{name:Metallica, founded:1981}	{1}
3	Person	{name:Bob}	{1,2}
4	Band	{name:AC/DC, founded:1973}	{2}
5	Person	{name:Eve}	{2}

DataSet<EPGMGraphHead>

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

DataSet<EPGMEdge>

Id	Label	Source	Target	Properties	Graphs
1	likes	1	2	{since:2014}	{1}
2	likes	3	2	{since:2013}	{1}
3	likes	3	4	{since:2015}	{2}
4	knows	3	5	{}	{2}
5	likes	5	4	{since:2014}	{2}

Flink DataSet Transformations

SQL-like Transformations

- filter
- project
- cross
- union
- distinct
- first-N (limit)
- groupBy
- aggregate
- join
- leftOuterJoin
- rightOuterJoin
- fullOuterJoin

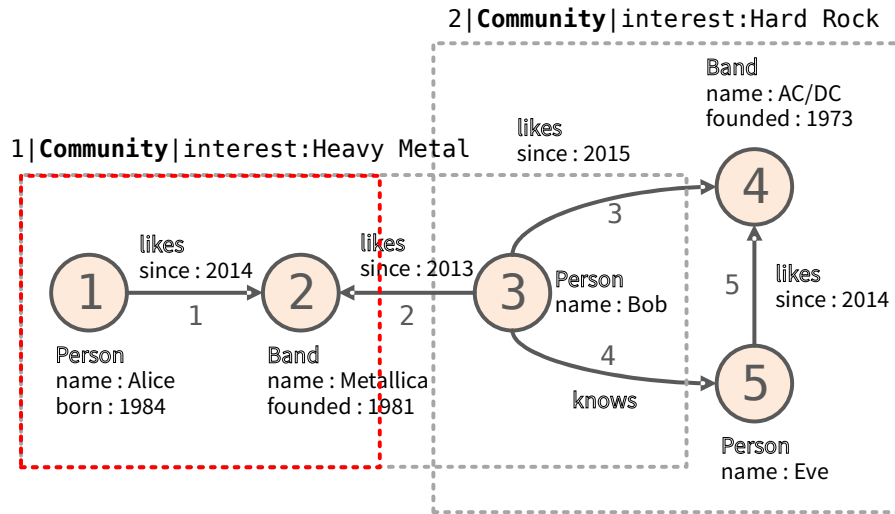
Hadoop-like Transformations

- map
- flatMap
- mapPartition
- reduce
- reduceGroup
- coGroup

Special Flink Operations

- iterate
- iterateDelta

Operator Implementation

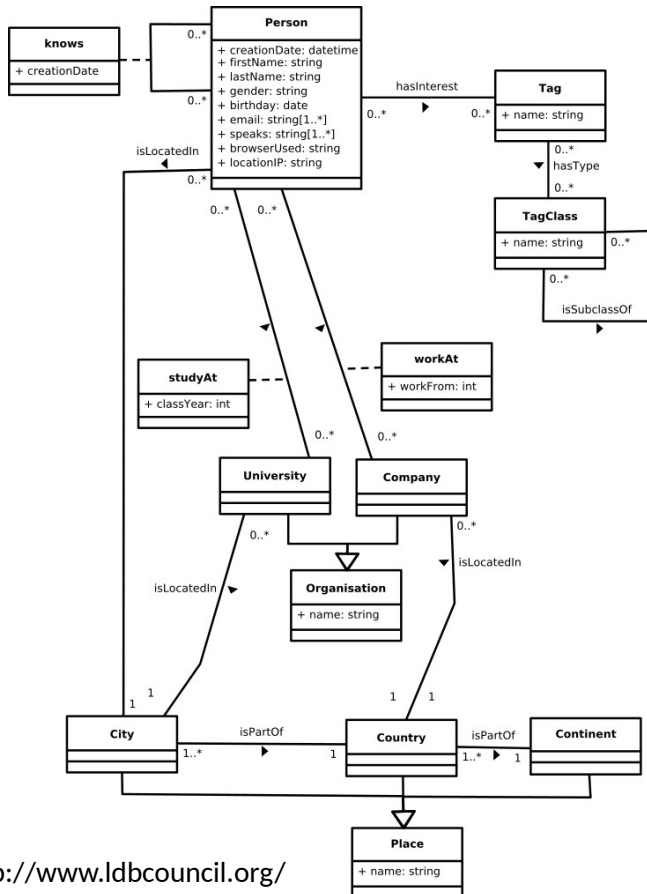


Exclusion

```
// input: firstGraph (G[1]), secondGraph (G[2])
1: DataSet<GradoopId> graphId = secondGraph.getGraphHead()
2:   .map(new Id<G>());
3:
4: DataSet<V> newVertices = firstGraph.getVertices()
5:   .filter(new NotInGraphBroadCast<V>())
6:   .withBroadcastSet(graphId, GRAPH_ID);
7:
8: DataSet<E> newEdges = firstGraph.getEdges()
9:   .filter(new NotInGraphBroadCast<E>())
10:  .withBroadcastSet(graphId, GRAPH_ID)
11:  .join(newVertices)
12:  .where(new SourceId<E>().equalTo(new Id<V>()))
13:  .with(new LeftSide<E, V>())
14:  .join(newVertices)
15:  .where(new TargetId<E>().equalTo(new Id<V>()))
16:  .with(new LeftSide<E, V>());
```

Performance

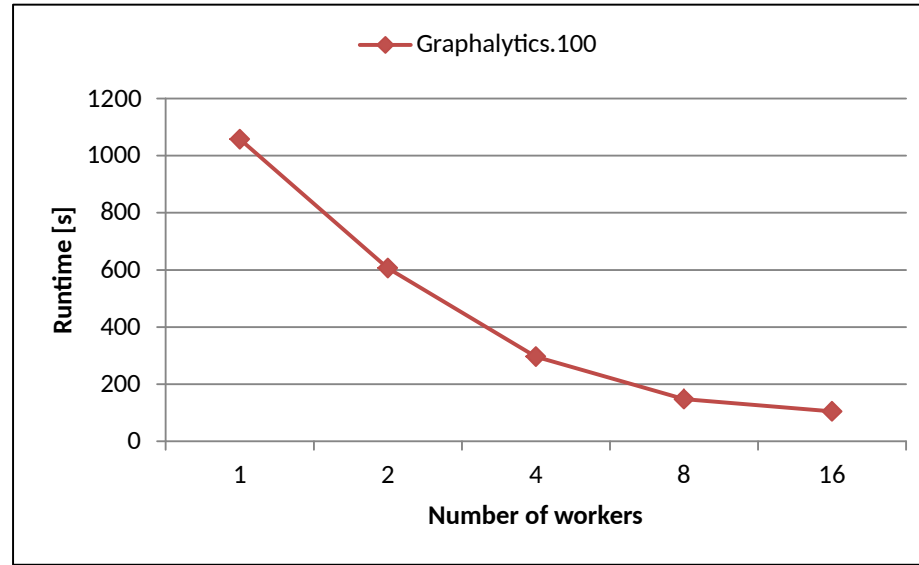
Social Network Benchmark



<http://www.ldbcouncil.org/>

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

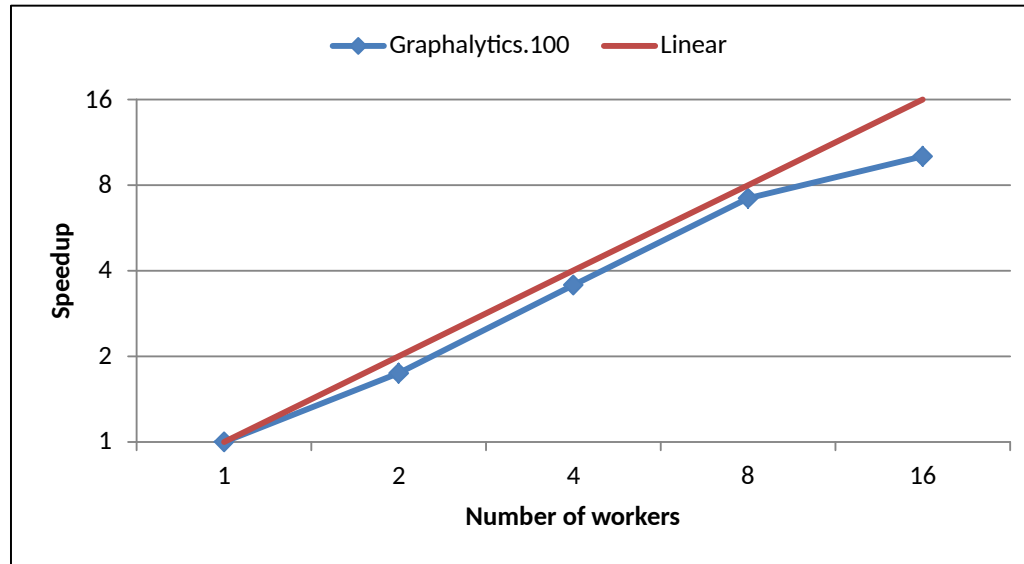
Social Network Benchmark – Runtime



Dataset	# Vertices	# Edges	Disk size
Graphalytics.1	61,613	2,026,082	570 MB
Graphalytics.10	260,613	16,600,778	4.5 GB
Graphalytics.100	1,695,613	147,437,275	40.2 GB
Graphalytics.1000	12,775,613	1,363,747,260	372 GB
Graphalytics.10000	90,025,613	10,872,109,028	2.9 TB

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 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

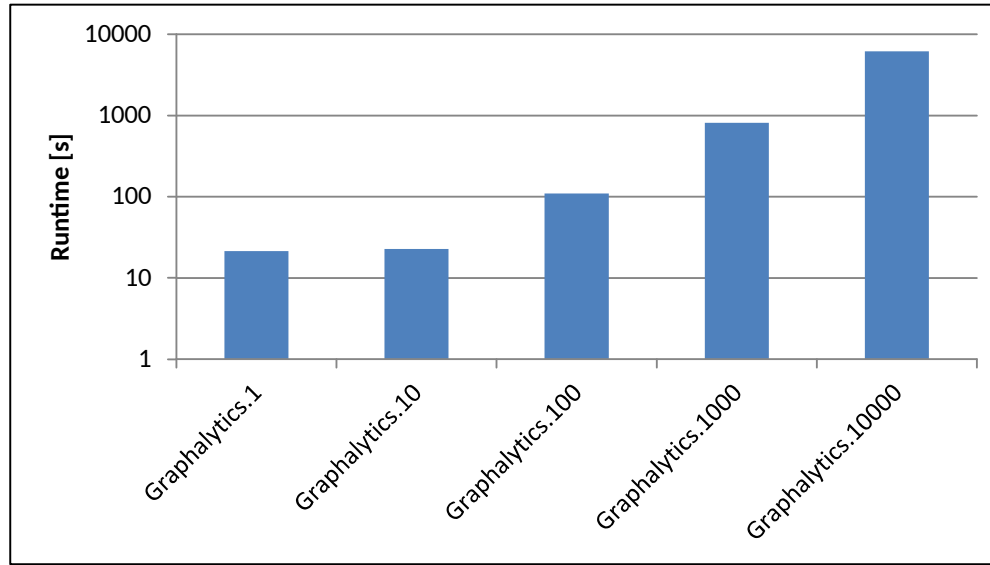
Social Network Benchmark – Speedup



Dataset	# Vertices	# Edges	Disk size
Graphalytics.1	61,613	2,026,082	570 MB
Graphalytics.10	260,613	16,600,778	4.5 GB
Graphalytics.100	1,695,613	147,437,275	40.2 GB
Graphalytics.1000	12,775,613	1,363,747,260	372 GB
Graphalytics.10000	90,025,613	10,872,109,028	2.9 TB

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 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

Social Network Benchmark – Datasets



Dataset	# Vertices	# Edges	Disk size
Graphalytics.1	61,613	2,026,082	570 MB
Graphalytics.10	260,613	16,600,778	4.5 GB
Graphalytics.100	1,695,613	147,437,275	40.2 GB
Graphalytics.1000	12,775,613	1,363,747,260	372 GB
Graphalytics.10000	90,025,613	10,872,109,028	2.9 TB

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 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

Whats next?

Future Work

- Support Temporal Graph Data and Temporal Analysis
- Support Graph Streams and Online analysis
- Inclusion of Apache Flink's Table API
 - Additional optimizer (Calcite)
 - Projection- and Filter-Pushdown to data sources
 - Base for Graph Stream support
 - Support for SQL based operators
 - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting

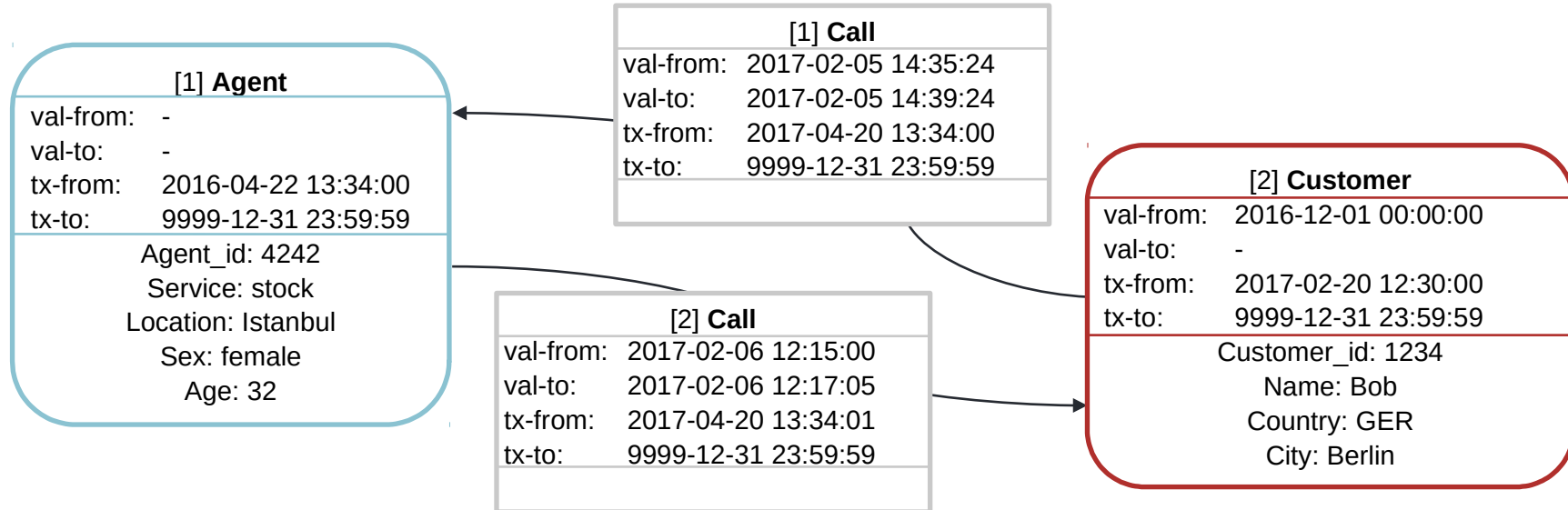
Future Work

- **Support Temporal Graph Data and Temporal Analysis**
- Support Graph Streams and Online analysis
- Inclusion of Apache Flink's Table API
 - Additional optimizer (Calcite)
 - Projection- and Filter-Pushdown to data sources
 - Base for Graph Stream support
 - Support for SQL based operators
 - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting

Temporal Property Graph Model (TPGM)

- Extends the EPGM (downwards compatible)
- **Bitemporal** time representation: valid- and transaction time
- Times can be (1) empty, (2) a timestamp or (3) a time-interval
- Valid times are specified by the application
- Transaction times are maintained by the system
- Whole graph with rollback and historical information
- Chaining of temporal operators → analytical workflow

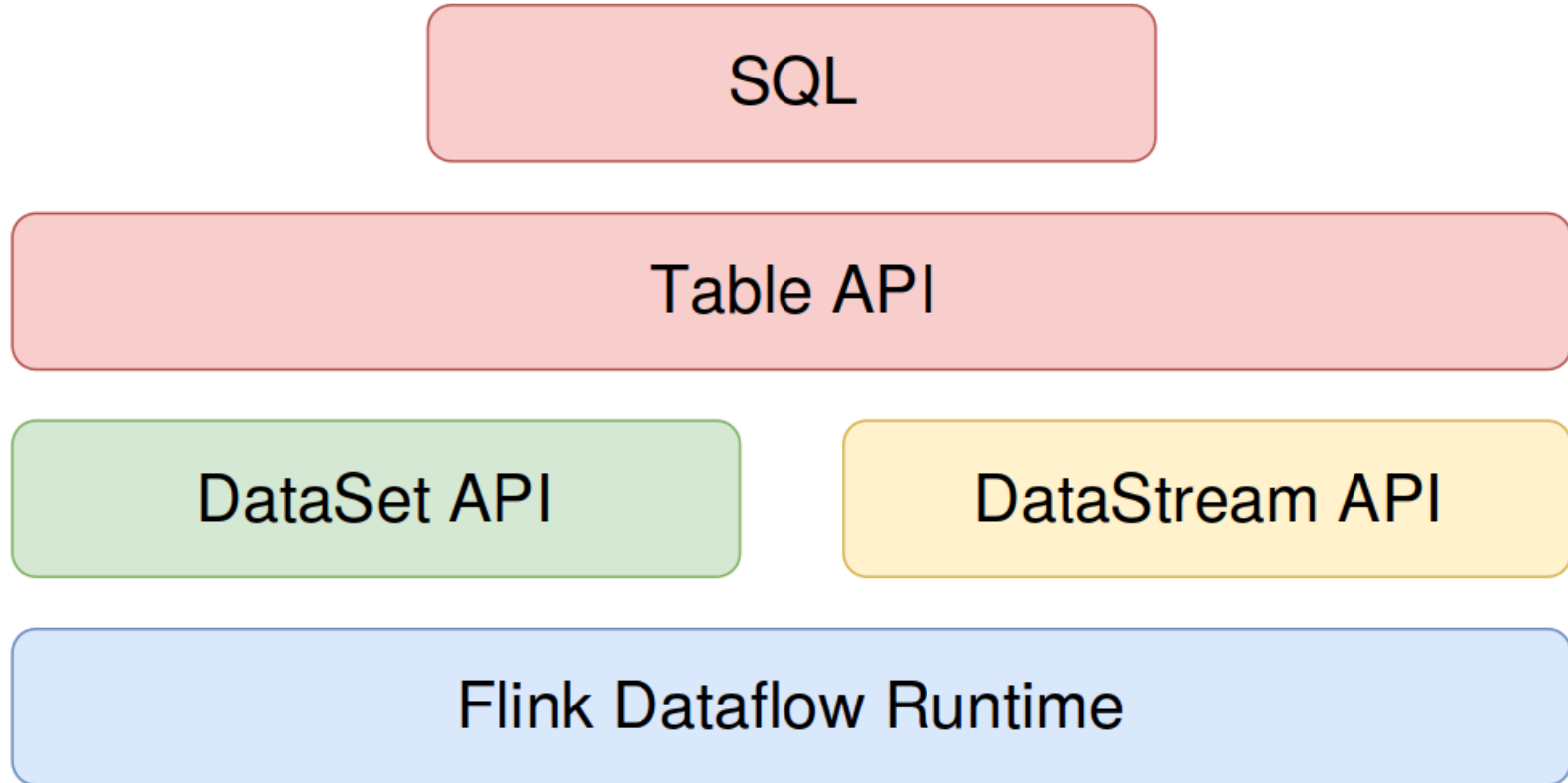
Future Work



Future Work

- Support Temporal Graph Data and Temporal Analysis
- Support Graph Streams and Online analysis
- **Inclusion of Apache Flink's Table API**
 - Additional optimizer (Calcite)
 - Projection- and Filter-Pushdown to data sources
 - Base for Graph Stream support
 - Support for SQL based operators
 - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting

Gradoop on Flink Table API



Gradoop on Flink Table API

- Relational API is declarative
 - Knowledge of dataflow processing concepts is not needed
 - Everyone knows Sequel (default language for data analytics)
 - Table-centric: Just think about tables
- Efficient query optimization
 - Less logic in User Defined Functions (UDFs)
 - System optimizes queries more efficiently (Calcite)

COMMING SOON!

Thank you!

www.gradoop.com

<http://flink.apache.org>

<http://ldbouncil.org>

Visit our Wiki!

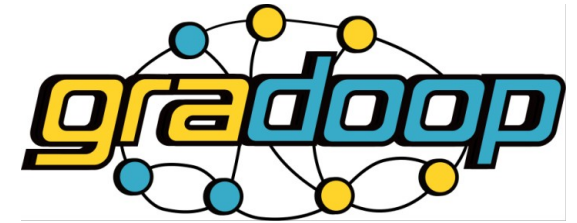
<https://github.com/dbs-leipzig/gradoop/wiki>

Try the Gradoop-Demo!

https://github.com/dbs-leipzig/gradoop_demo

Questions? Mail to: gomez@informatik.uni-leipzig.de

UNIVERSITÄT LEIPZIG



Europäische Union
Europäischer Fonds für
regionale Entwicklung
Europäischer
Sozialfonds

Europa fördert Sachsen.



- [1] Junghanns, M.; Petermann, A.; Teichmann, N.; Gomez, K.; Rahm, E.,
"Analyzing Extended Property Graphs with Apache Flink",
Int. Workshop on Network Data Analytics (NDA), SIGMOD 2016.
- [2] Petermann, A.; Junghanns, M.,
"Scalable Business Intelligence with Graph Collections",
It – Special Issue on Big Data Analytics, 2016.
- [3] Junghanns, M.; Kießling, M.; Teichmann, N.; Gomez, K.; Petermann, A.; Rahm, E.,
"Declarative and distributed Graph analytics with GRADOOP",
PVLDB 2018
- [4] Rost, C.; Thor, A.; Fritzsche, P.; Gomez, K.; Rahm, E.,
"Evolution Analysis of Large Graphs with GRADOOP",
Proc. Of Intl. Workshop on Advances in managing and mining large evolving graphs,
(LEG@ECML-PKDD)