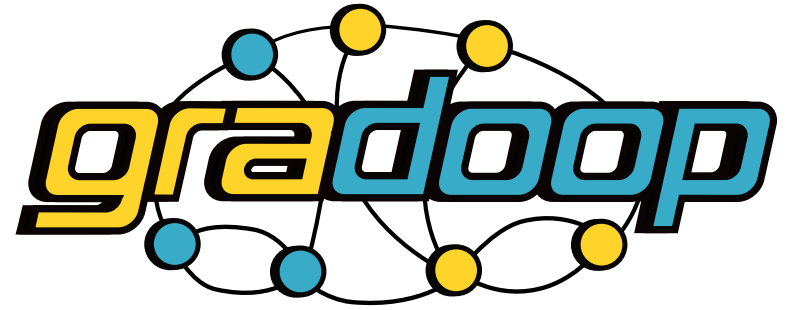




UNIVERSITÄT
LEIPZIG

ScaDS.AI
DRESDEN LEIPZIG



Gradoop Tutorial

Scalable Graph Analytics on Apache Flink

BOSS Tutorial @ VLDB2020

Kevin Gomez

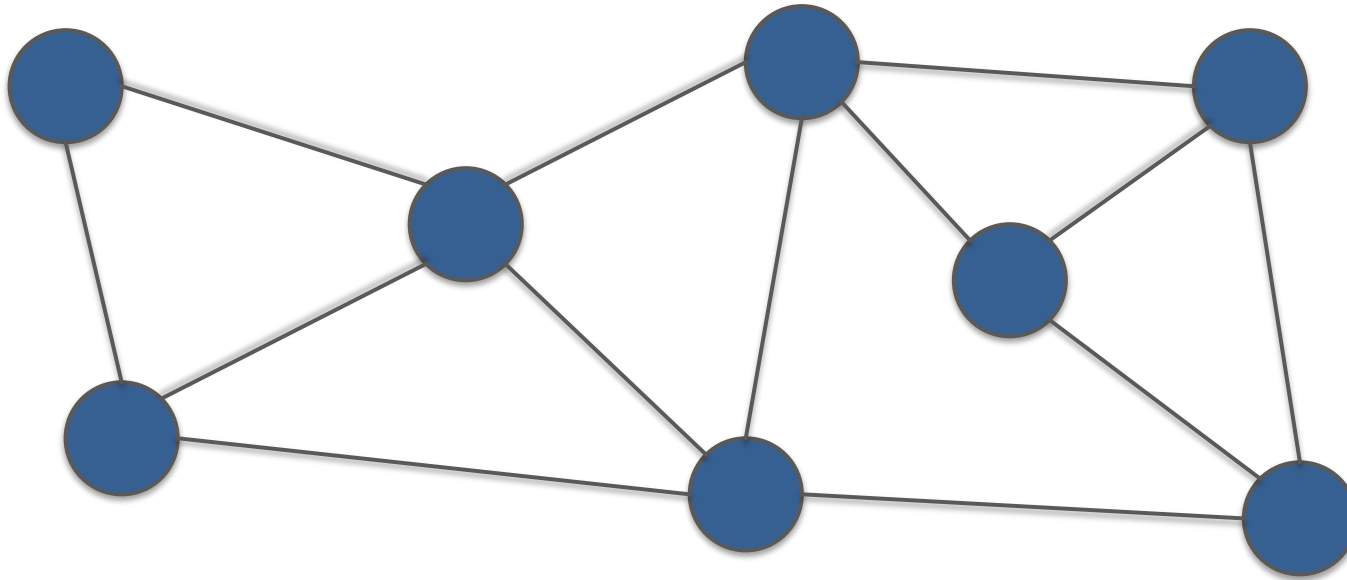
University of Leipzig

{lastname}@informatik.uni-leipzig.de

Christopher Rost

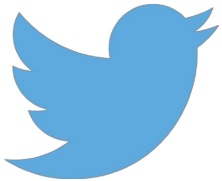
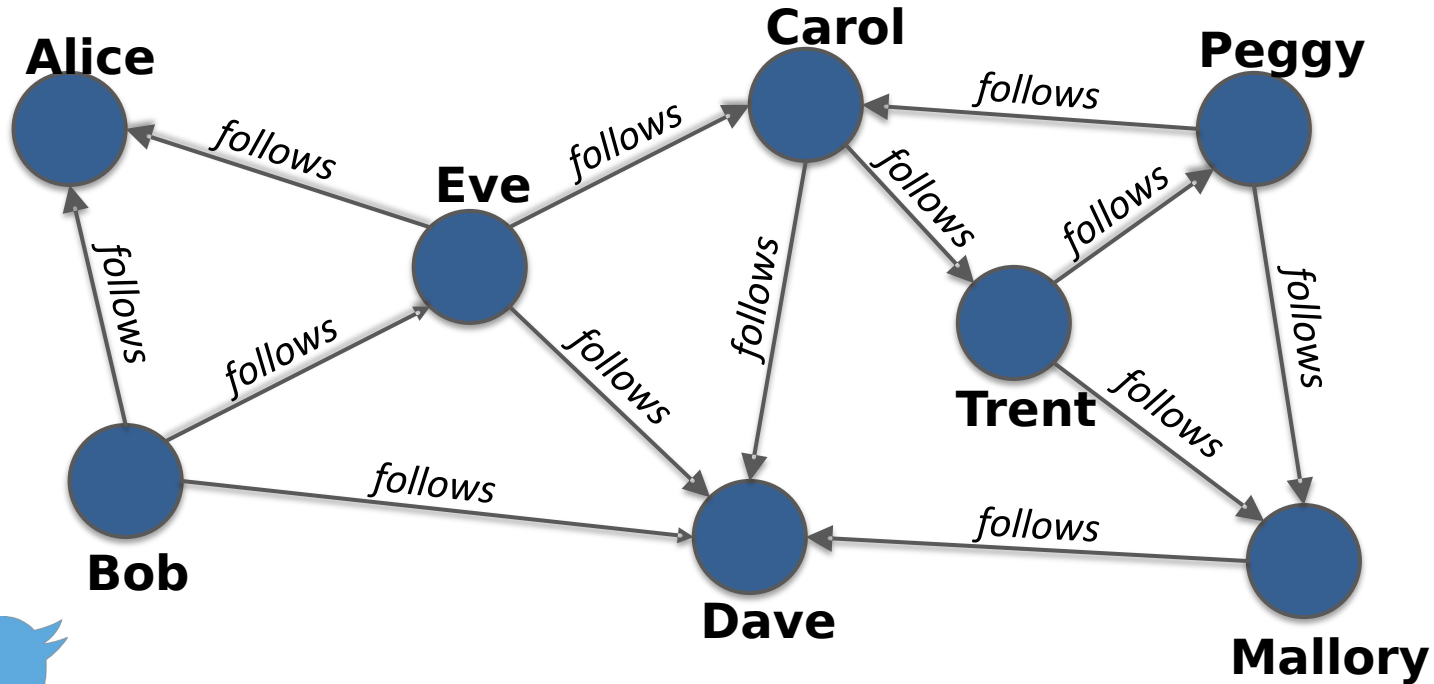
University of Leipzig

„Graphs are everywhere“



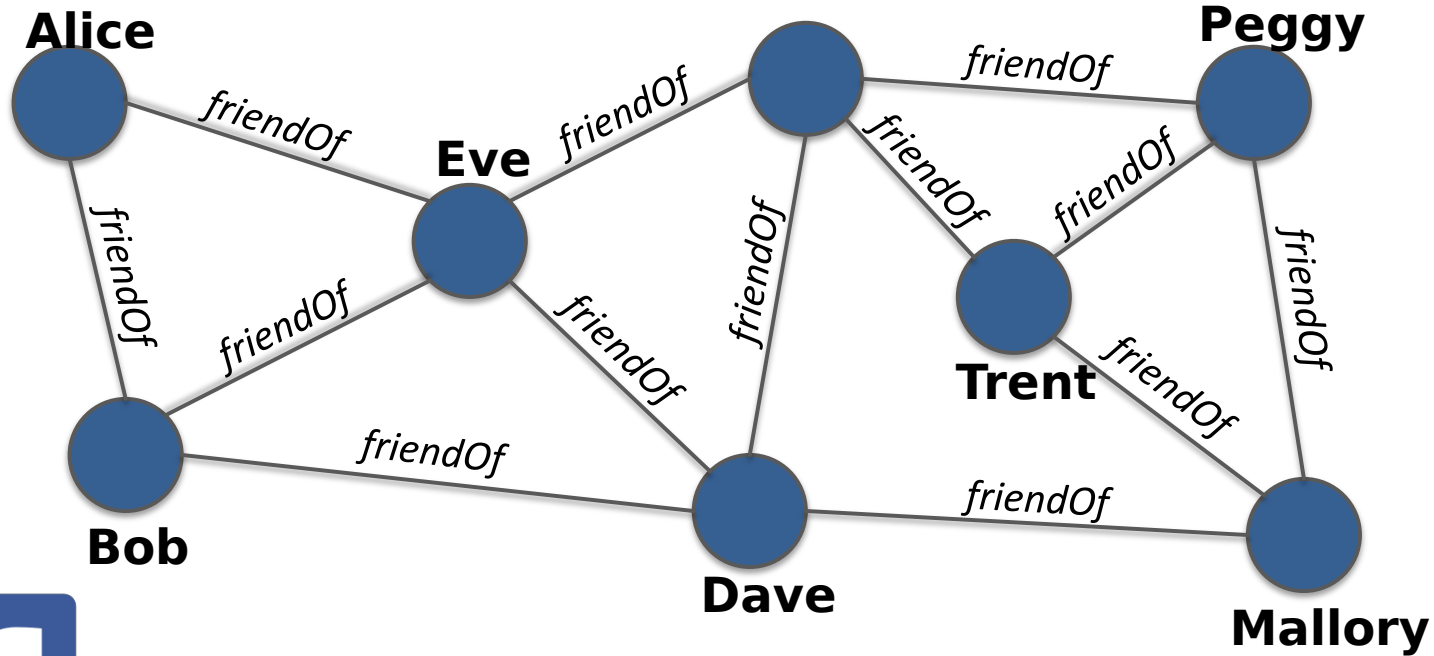
$Graph = (Vertices, Edges)$

„Graphs are everywhere“



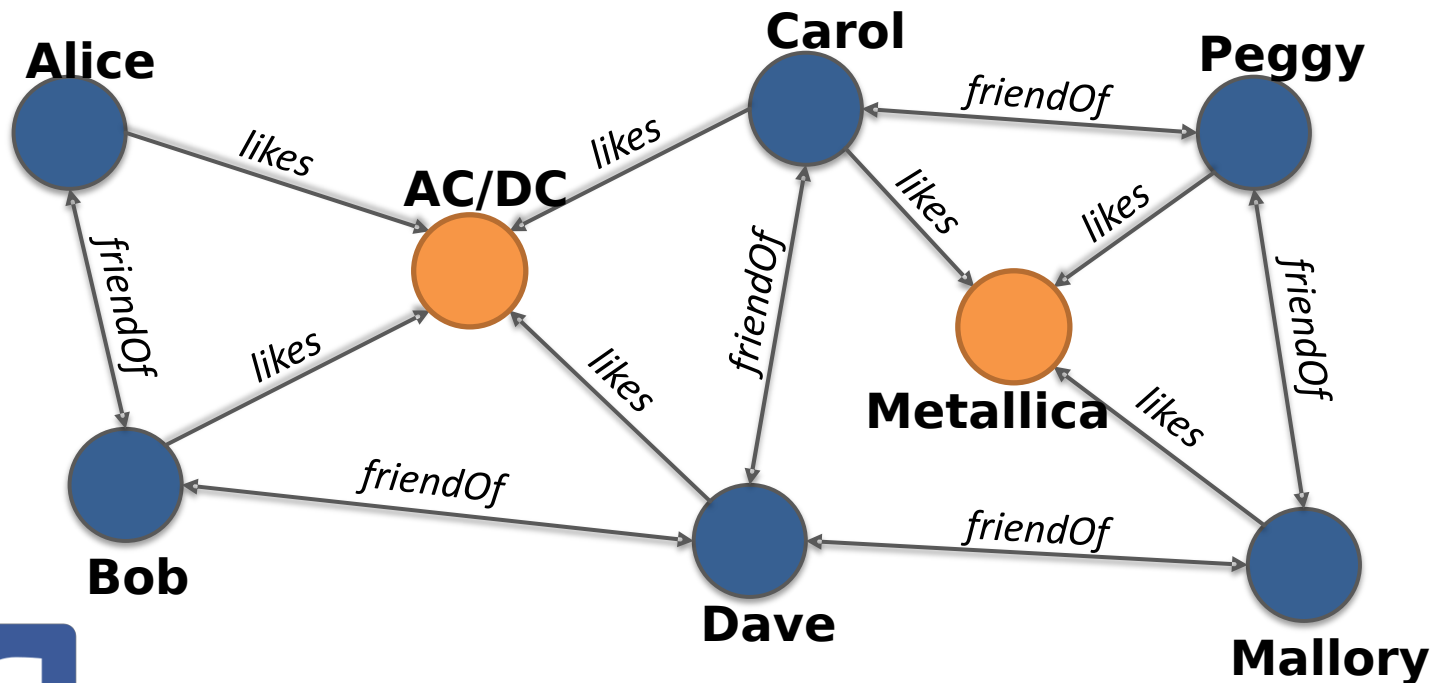
$Graph = (Users, Followers)$

„Graphs are everywhere“



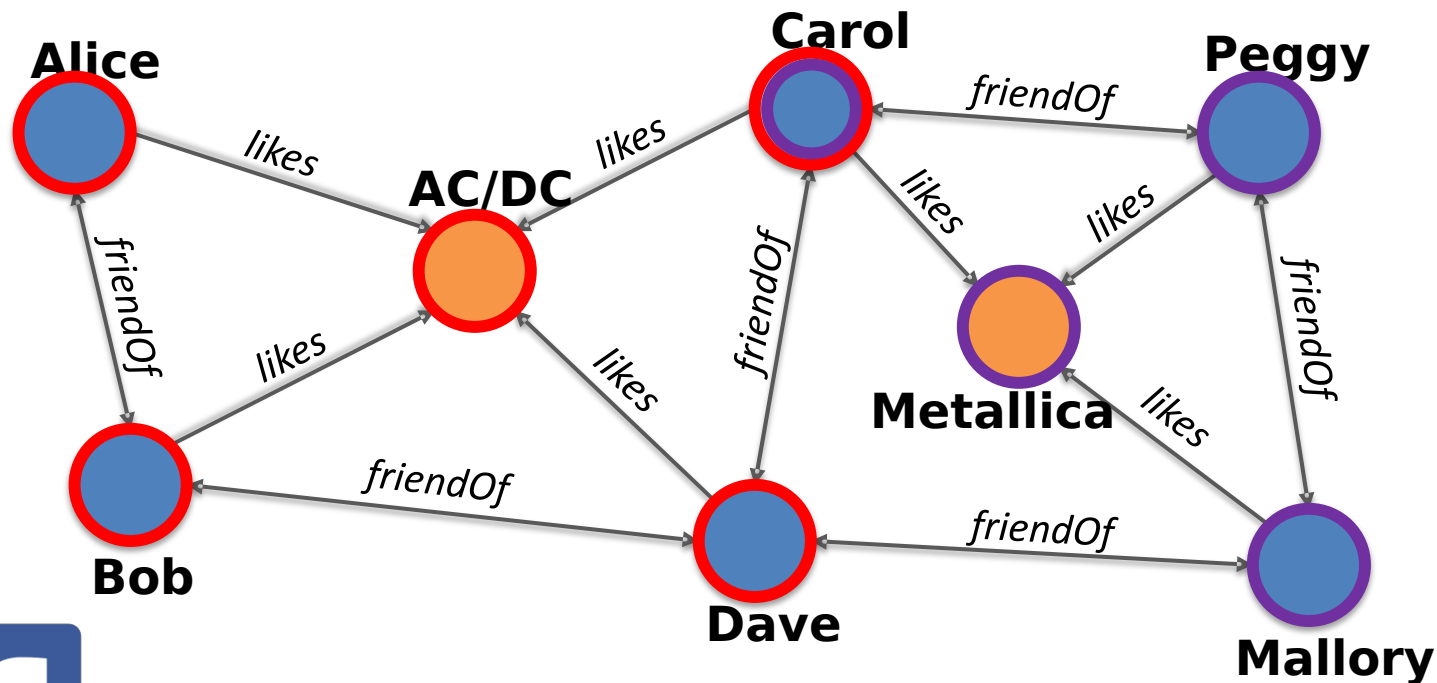
$Graph = (Users, Friendships)$

„Graphs are heterogeneous“



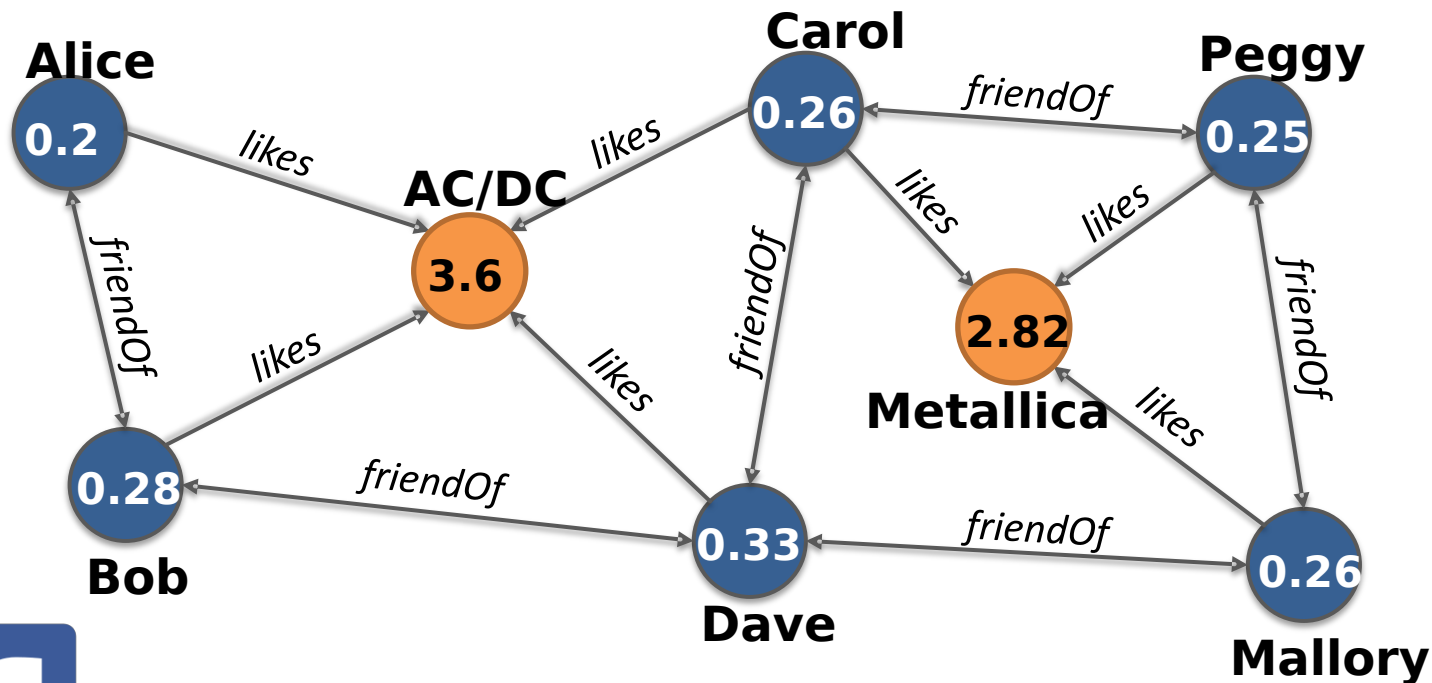
$Graph = (Users \cup Bands, Friendships \cup Likes)$

„Graphs can be analyzed“



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

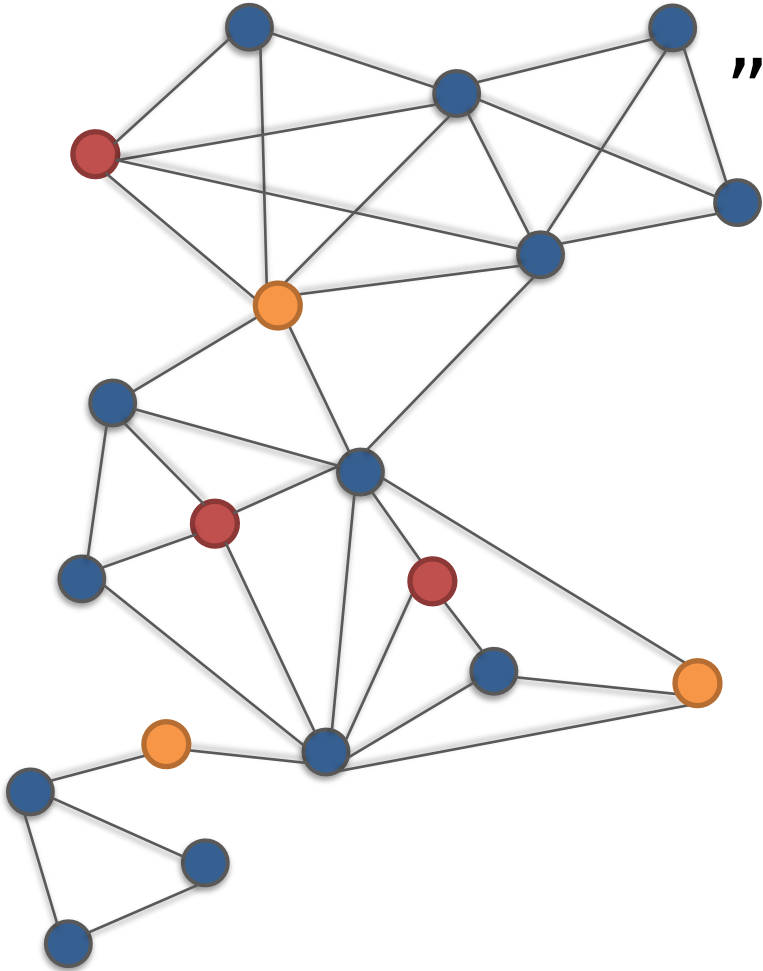
„Graphs can be analyzed“



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

„Graphs can be analyzed“

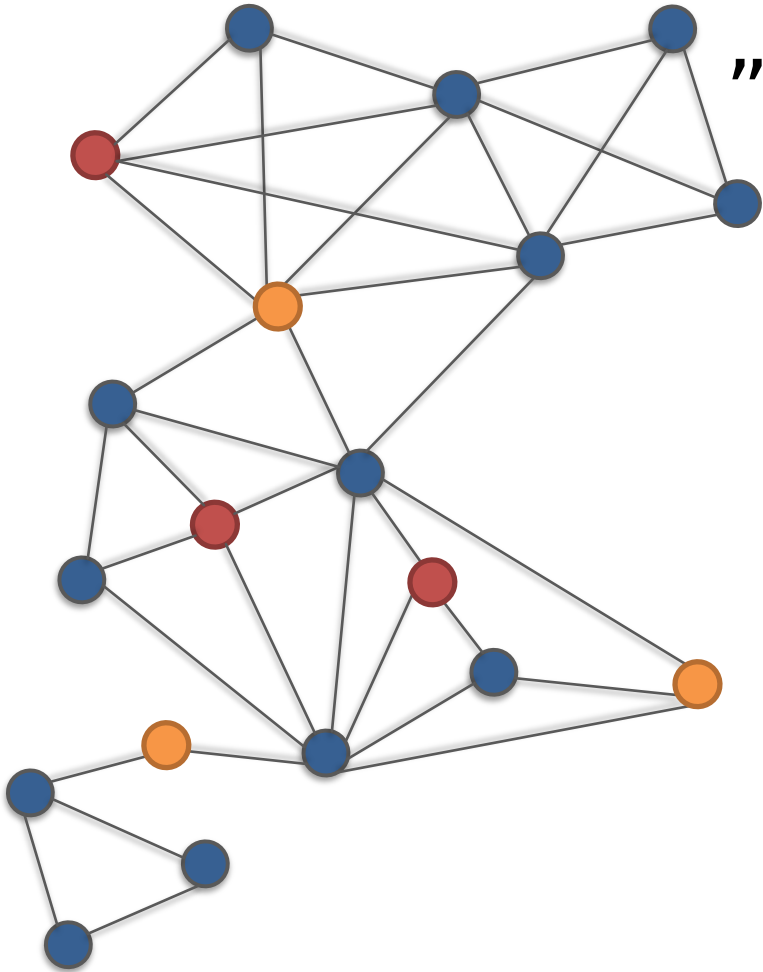
Assuming a social network



„Graphs can be analyzed“

Assuming a social network

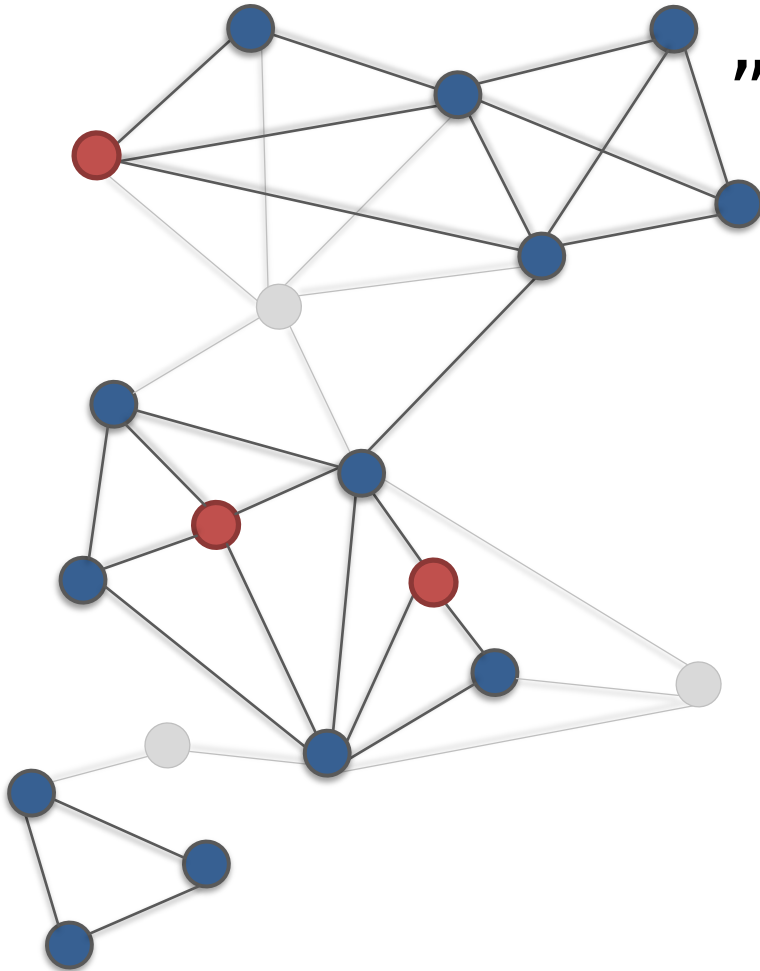
1. Determine subgraph



„Graphs can be analyzed“

Assuming a social network

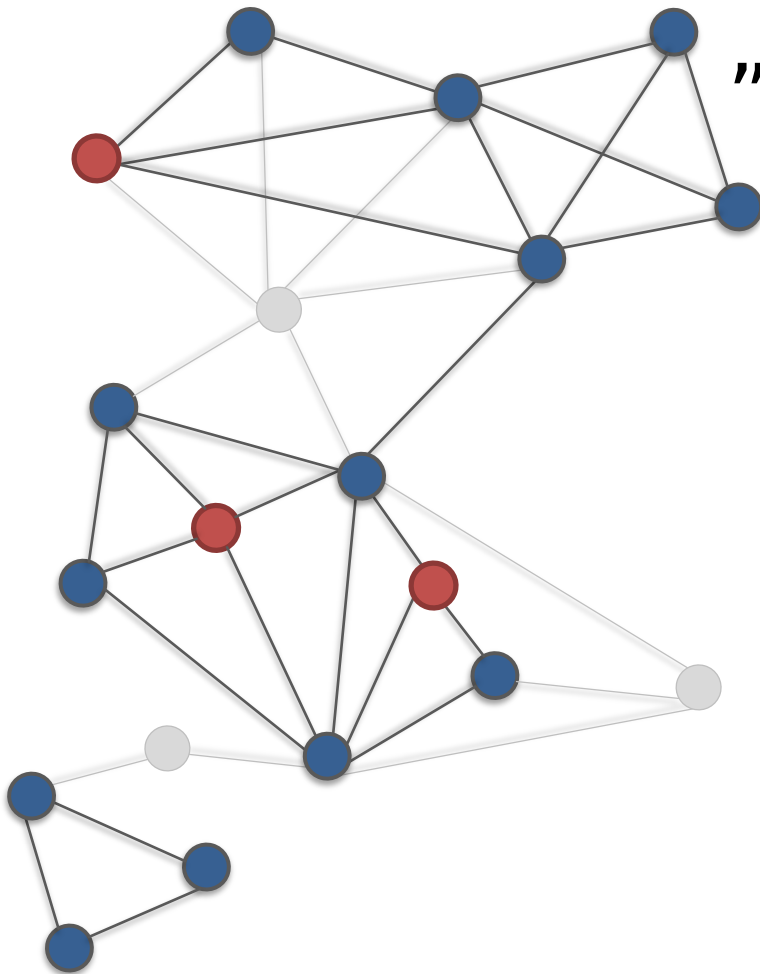
1. Determine subgraph



„Graphs can be analyzed“

Assuming a social network

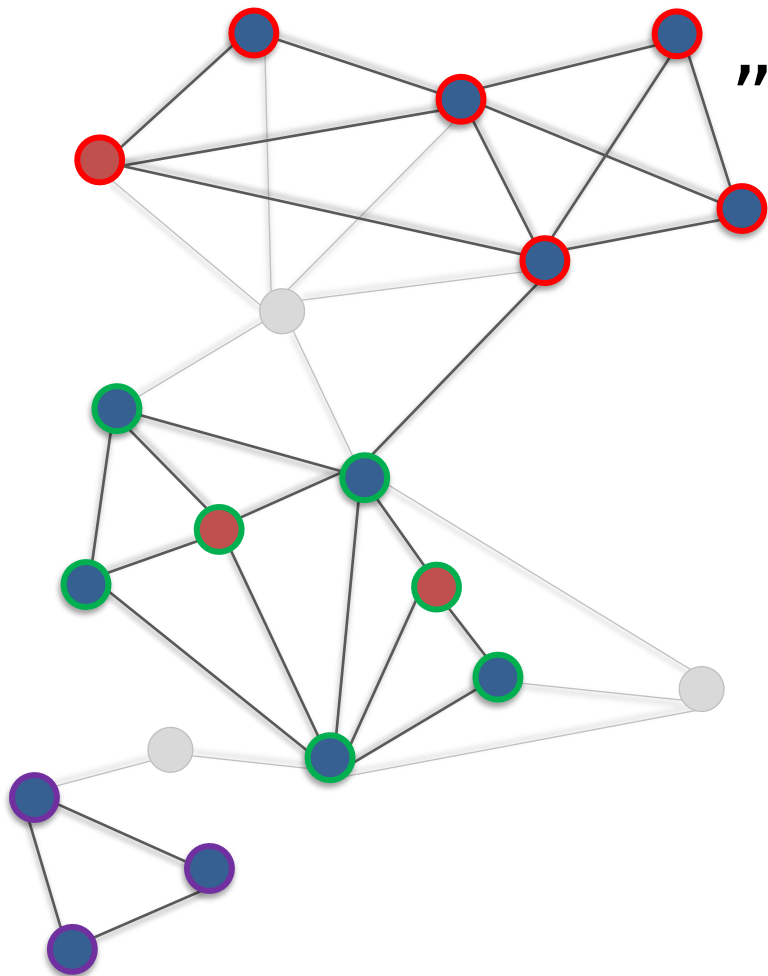
1. Determine subgraph
2. Find communities



„Graphs can be analyzed“

Assuming a social network

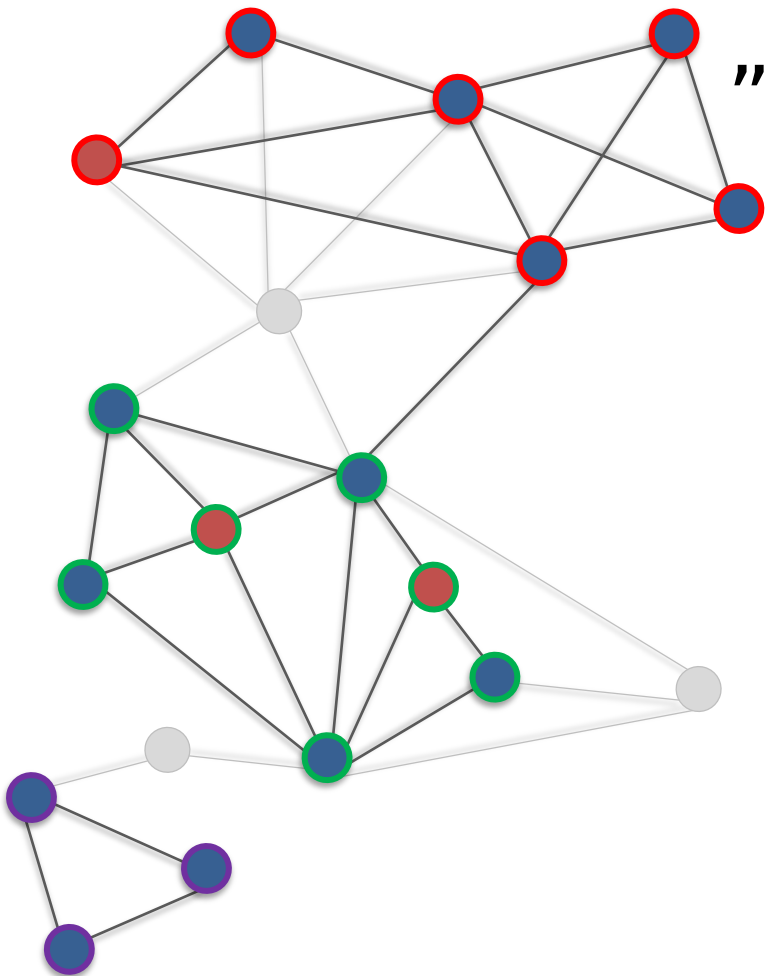
1. Determine subgraph
2. Find communities

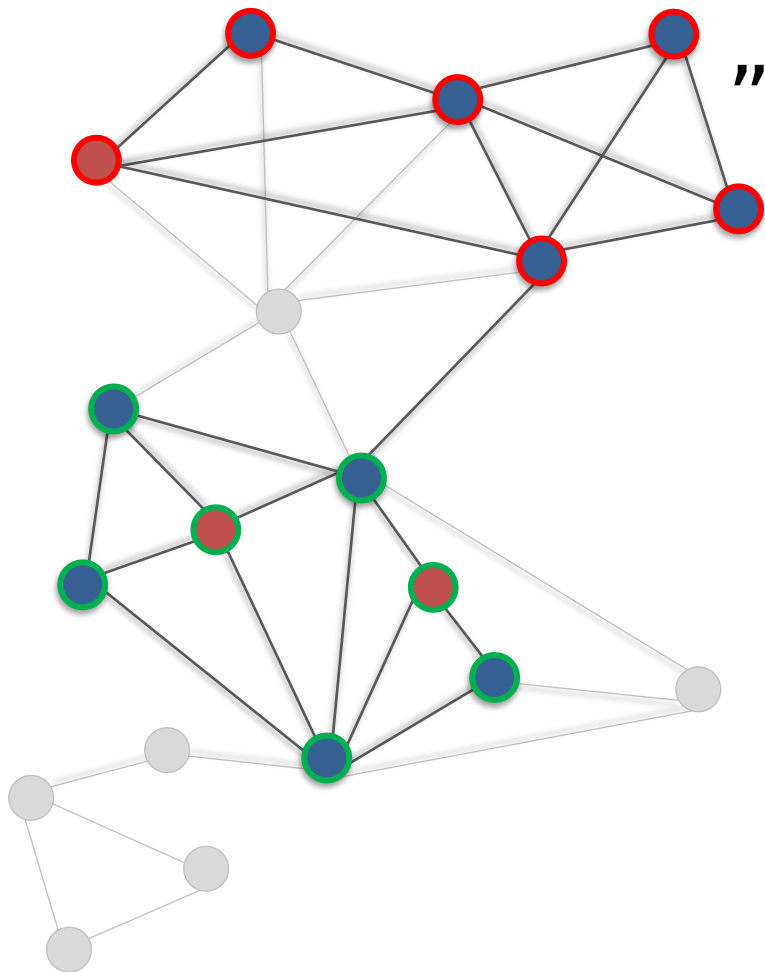


„Graphs can be analyzed“

Assuming a social network

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

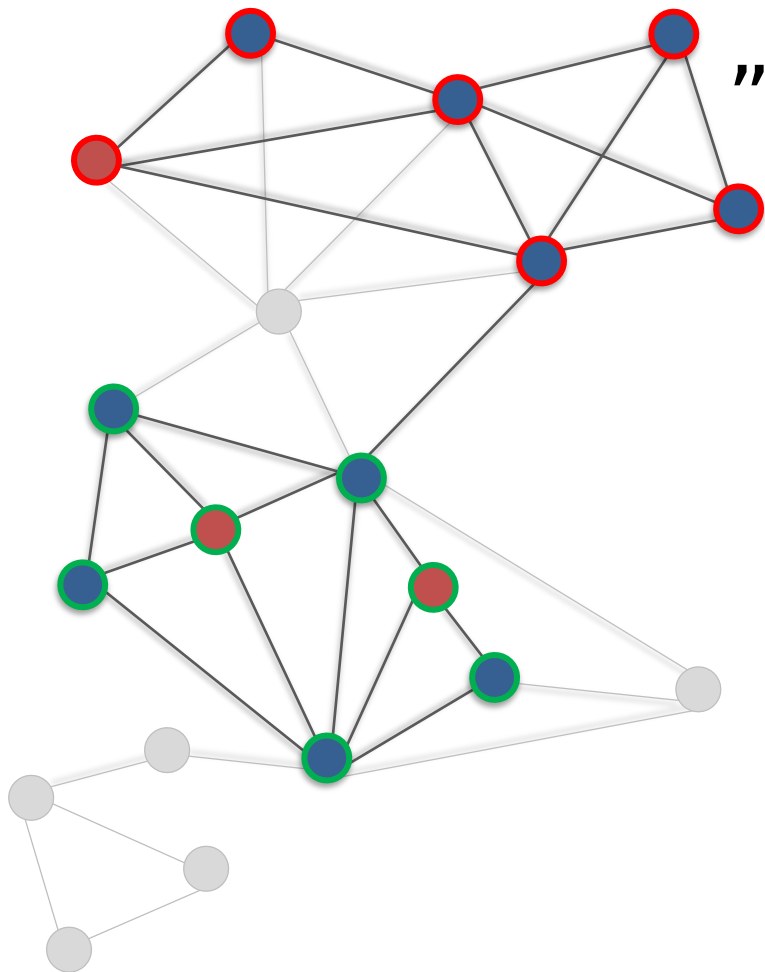




„Graphs can be analyzed“

Assuming a social network

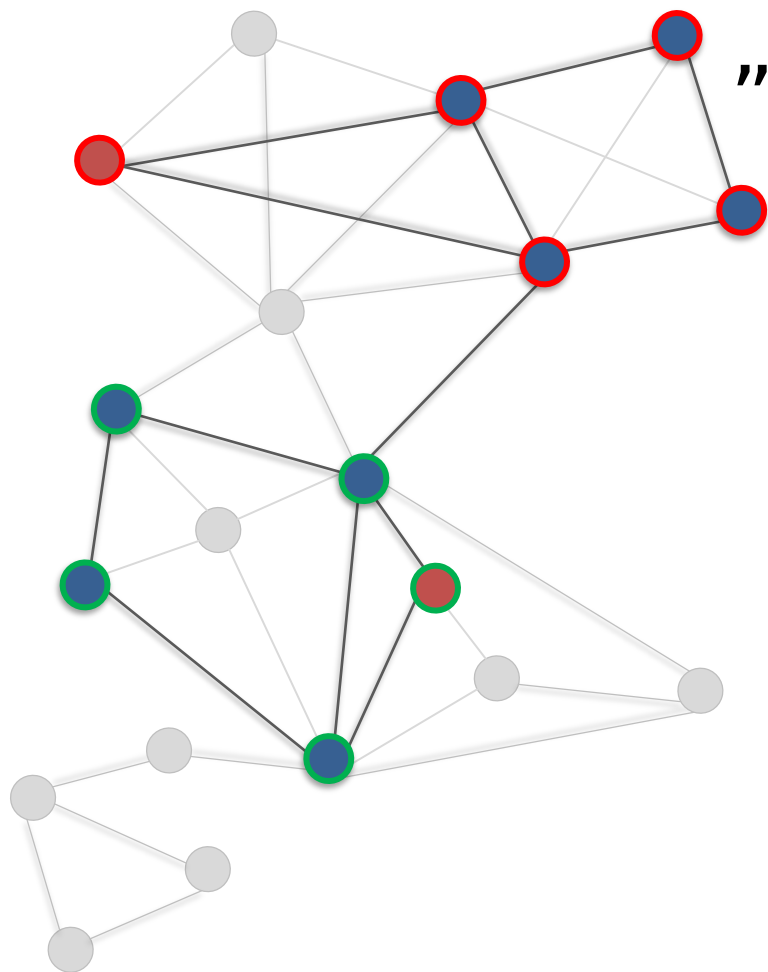
1. Determine subgraph
2. Find communities
3. Filter communities



„Graphs can be analyzed“

Assuming a social network

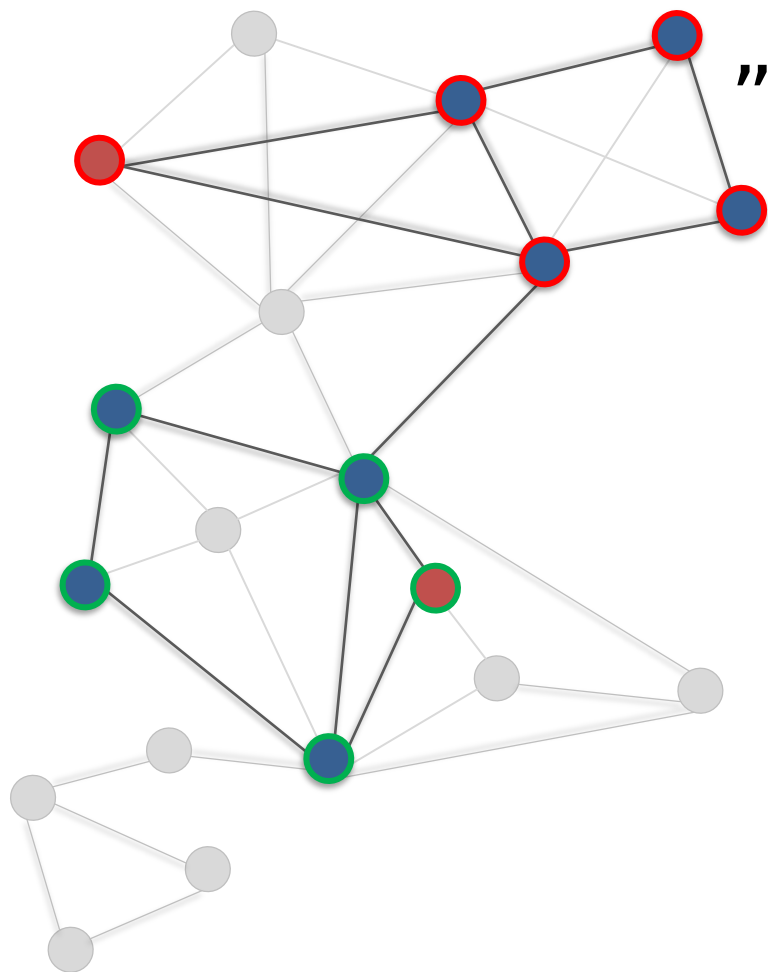
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph



„Graphs can be analyzed“

Assuming a social network

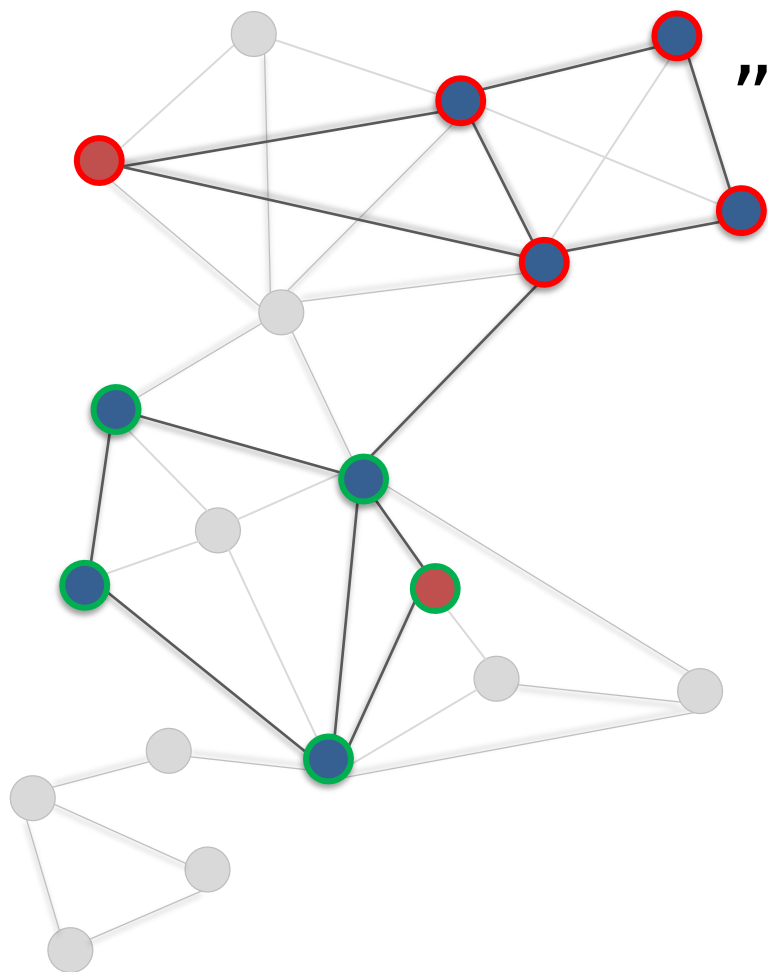
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph



„Graphs can be analyzed“

Assuming a social network

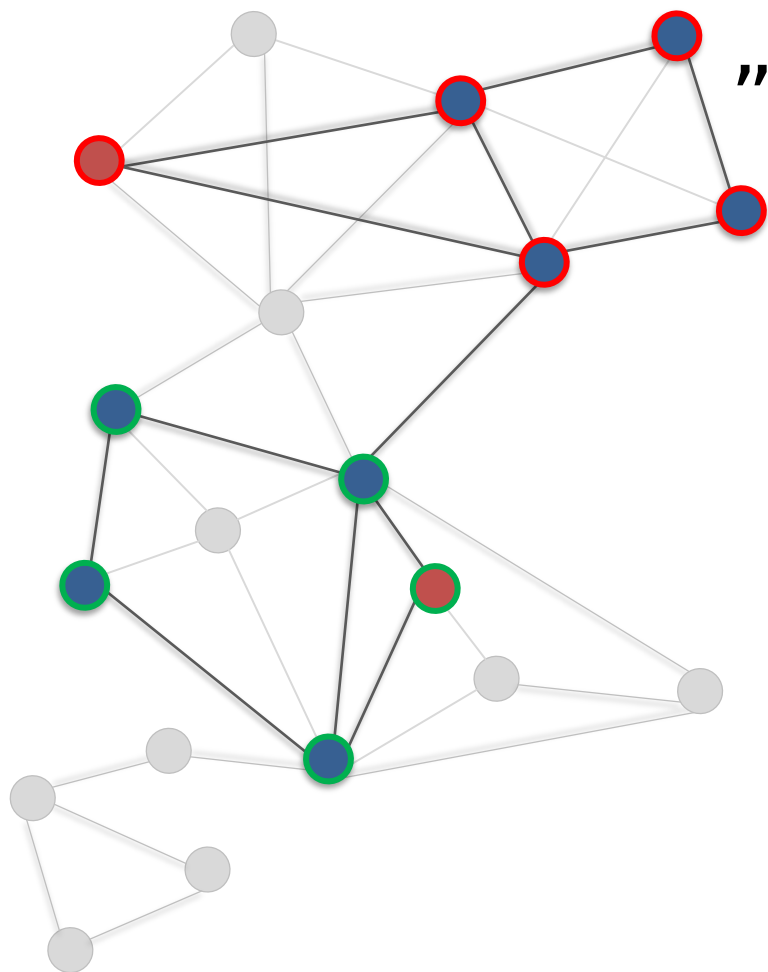
- **Heterogeneous data**
 1. Determine subgraph
- **Apply graph transformation**
 2. Find communities
- **Handle collections of graphs**
 3. Filter communities
- **Aggregation, Selection**
 4. Find common subgraph
- **Apply dedicated algorithm**



„Graphs can be analyzed“

Assuming a social network

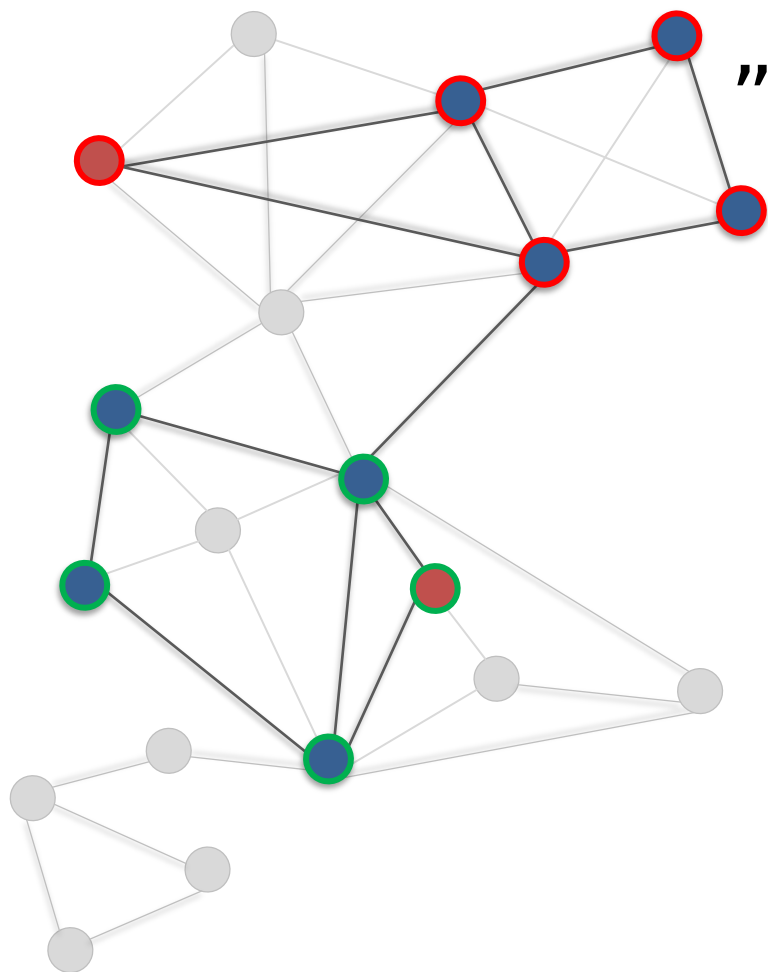
- Heterogeneous data
1. **Determine subgraph**
 2. Find communities
 - **Apply graph transformation**
 3. Filter communities
 - Handle collections of graphs
 4. Find common subgraph
 - Aggregation, Selection
 4. Find common subgraph
 - Apply dedicated algorithm



„Graphs can be analyzed“

Assuming a social network

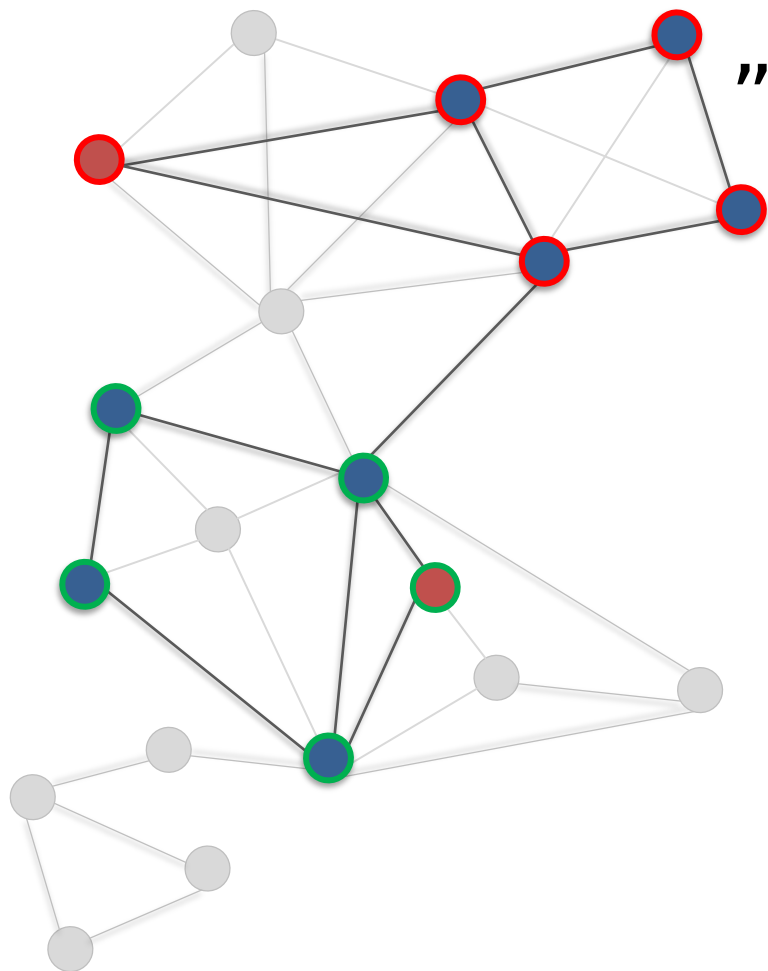
- Heterogeneous data
1. Determine subgraph
- Apply graph transformation
- 2. Find communities**
- **Handle collections of graphs**
3. Filter communities
- Aggregation, Selection
4. Find common subgraph
- Apply dedicated algorithm



„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...“

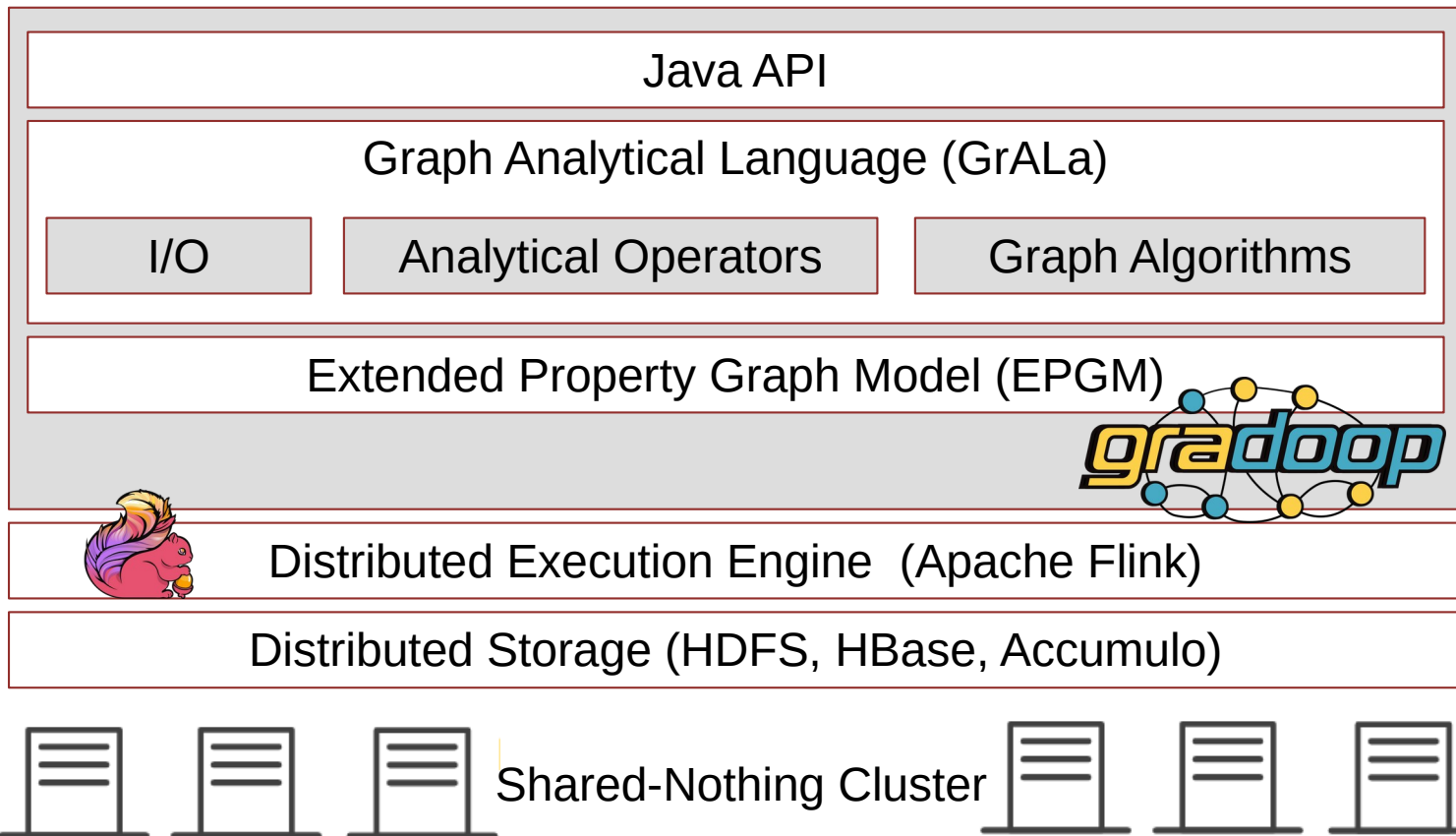
„...Graphs are large“



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

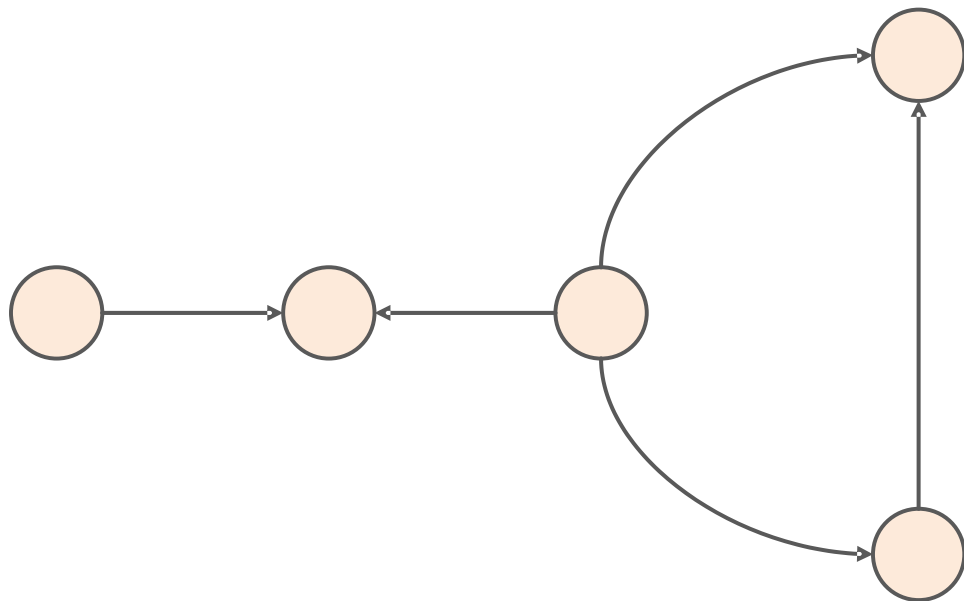


Architecture

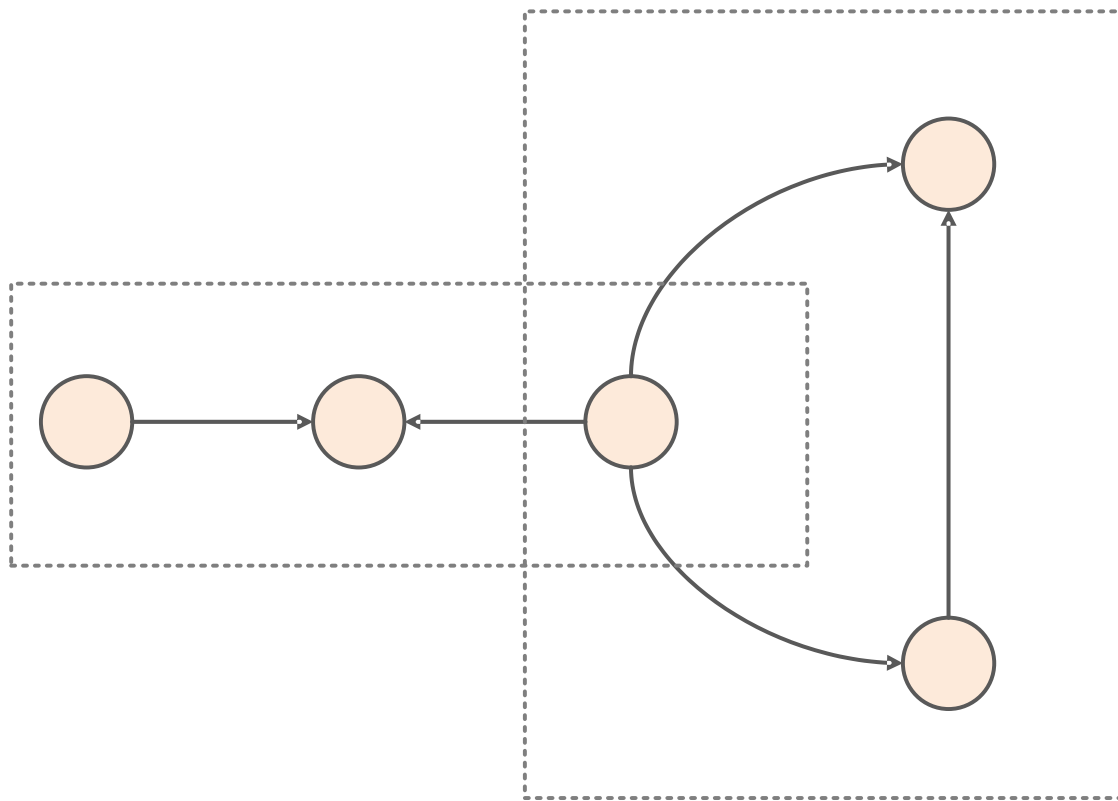


Extended Property Graph Model (EPGM)

- **Vertices and directed Edges**

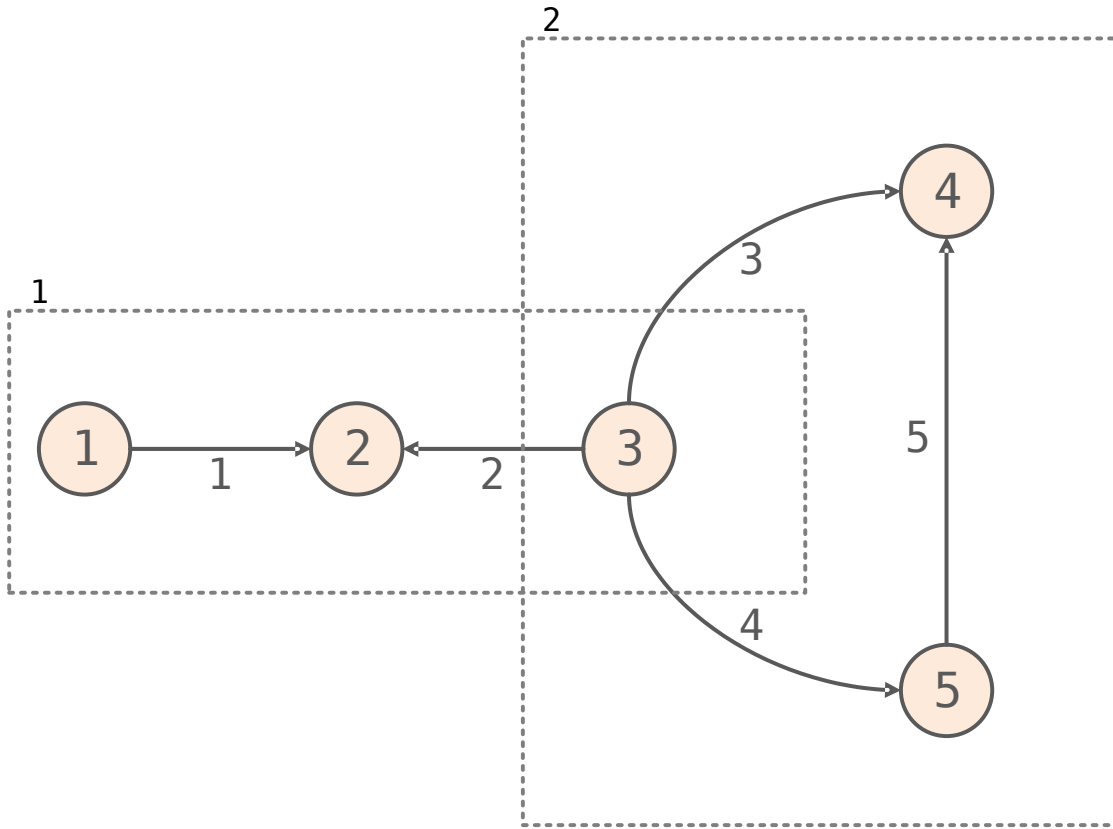


Extended Property Graph Model (EPGM)



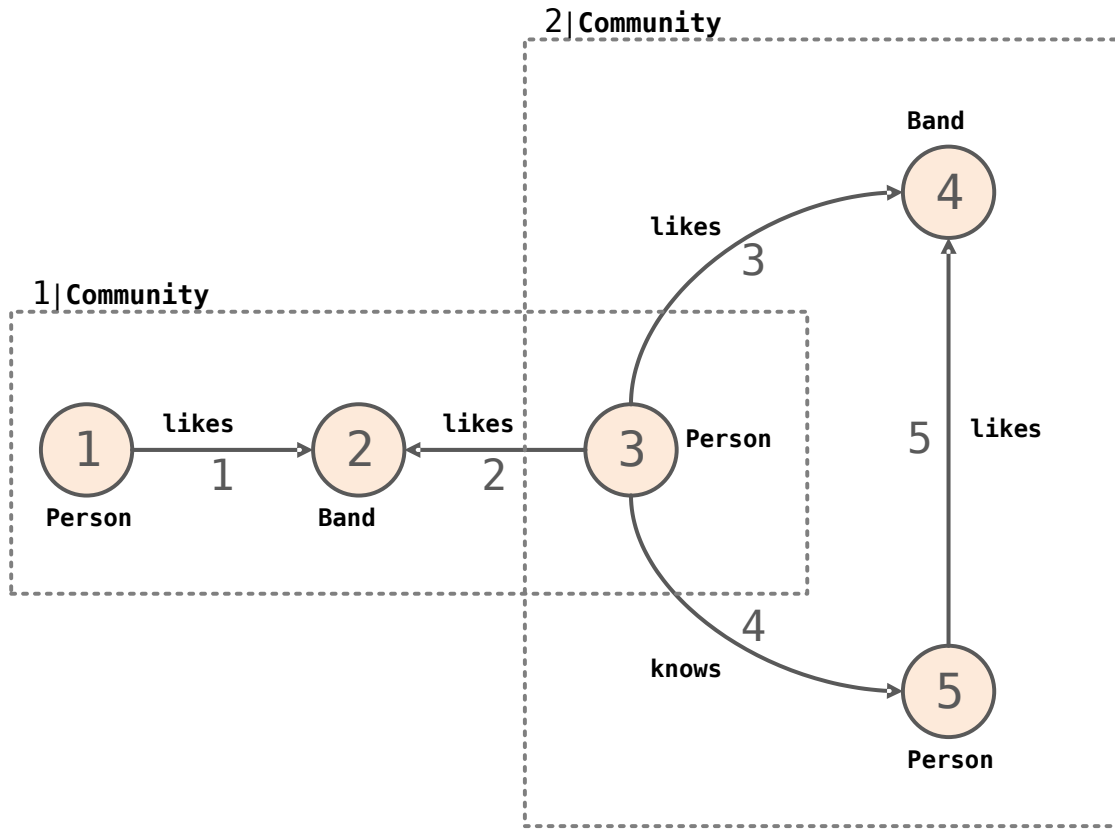
- Vertices and directed Edges
- **Logical Graphs**

Extended Property Graph Model (EPGM)



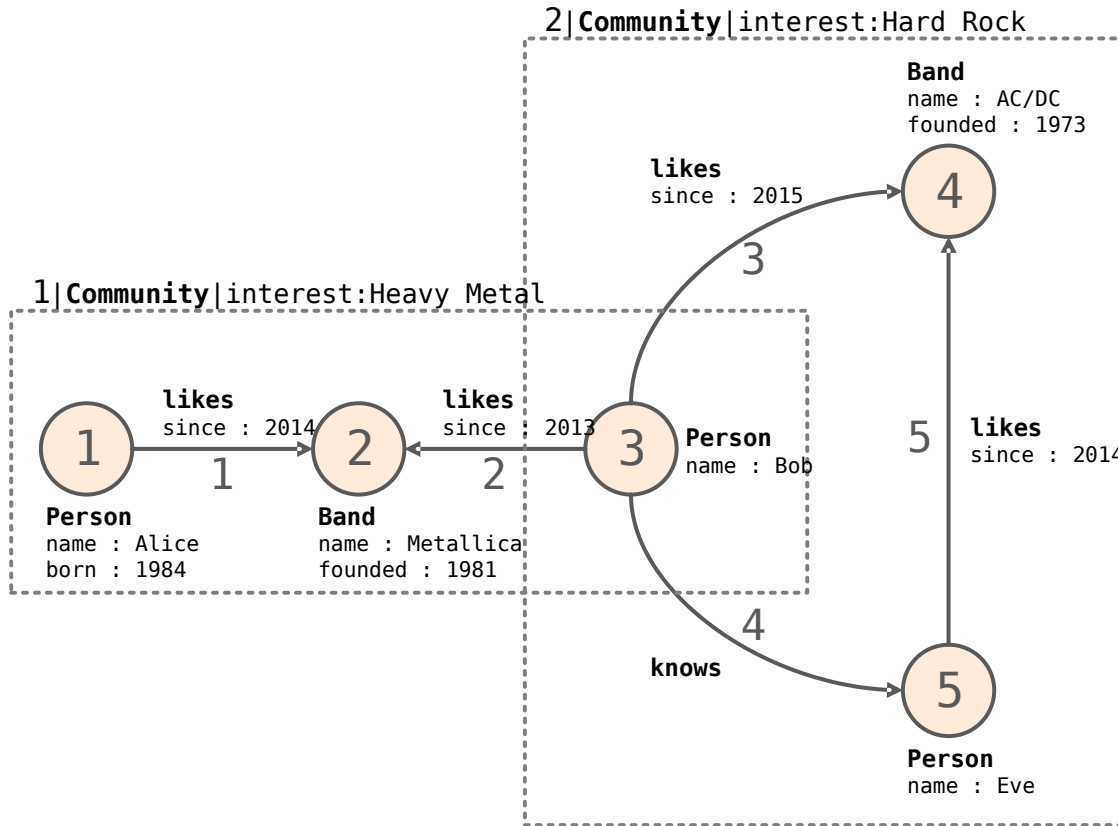
- Vertices and directed Edges
- Logical Graphs
- **Identifiers**

Extended Property Graph Model (EPGM)



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

Extended Property Graph Model (EPGM)



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

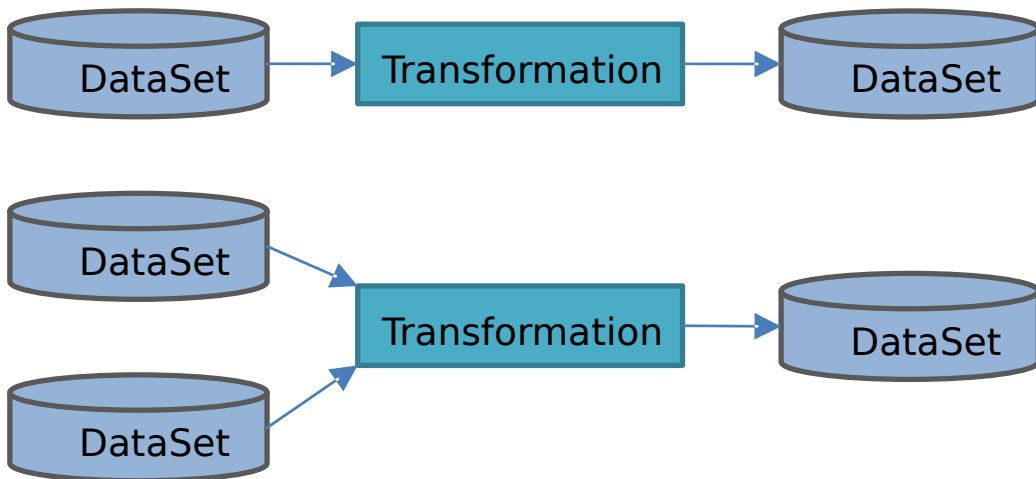
EPGM on Apache Flink

- **DataSet** := Distributed Collection of Data Objects



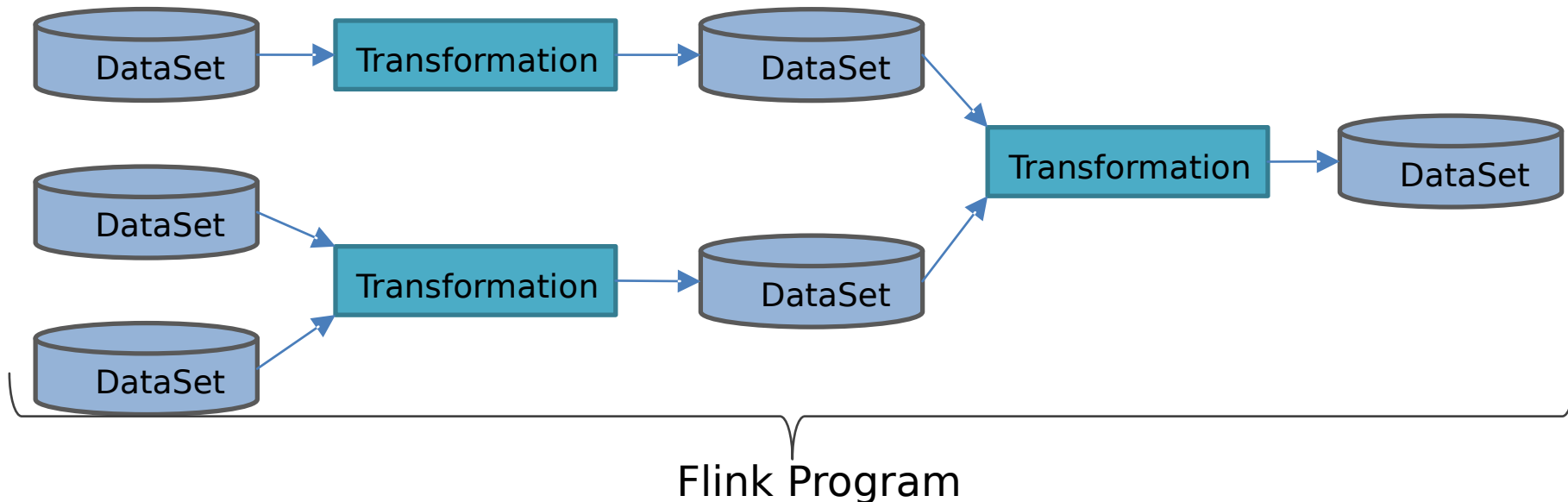
EPGM on Apache Flink

- **DataSet** := Distributed Collection of Data Objects
- **Transformation** := Operation on DataSets



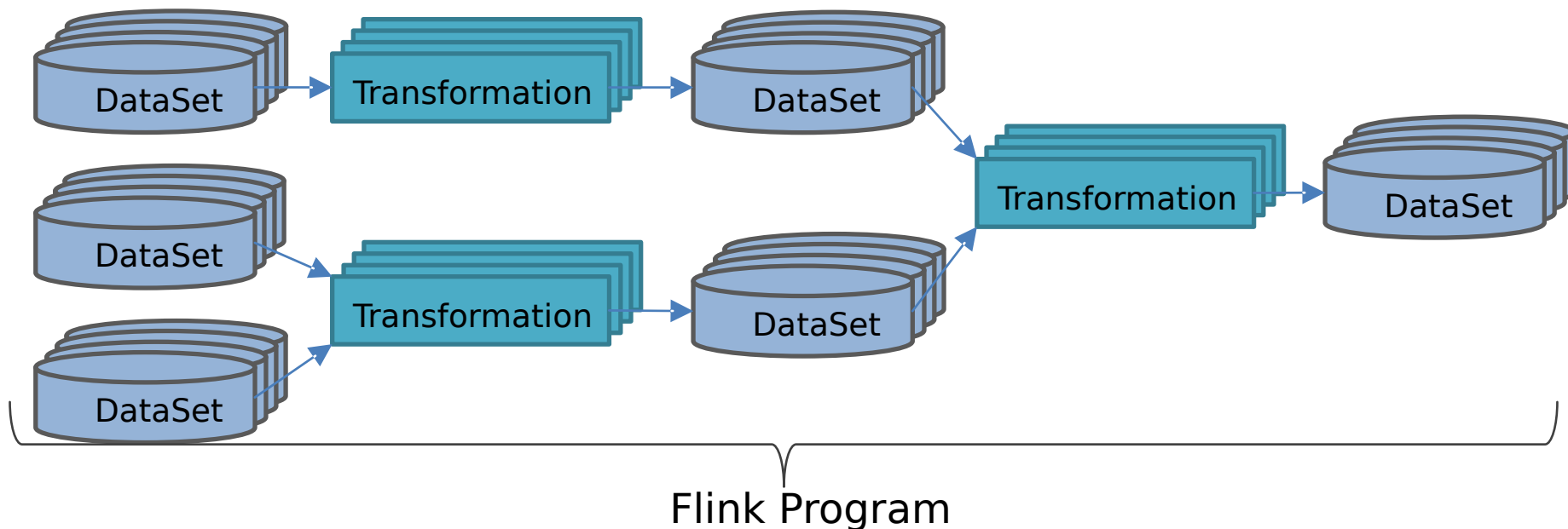
EPGM on Apache Flink

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

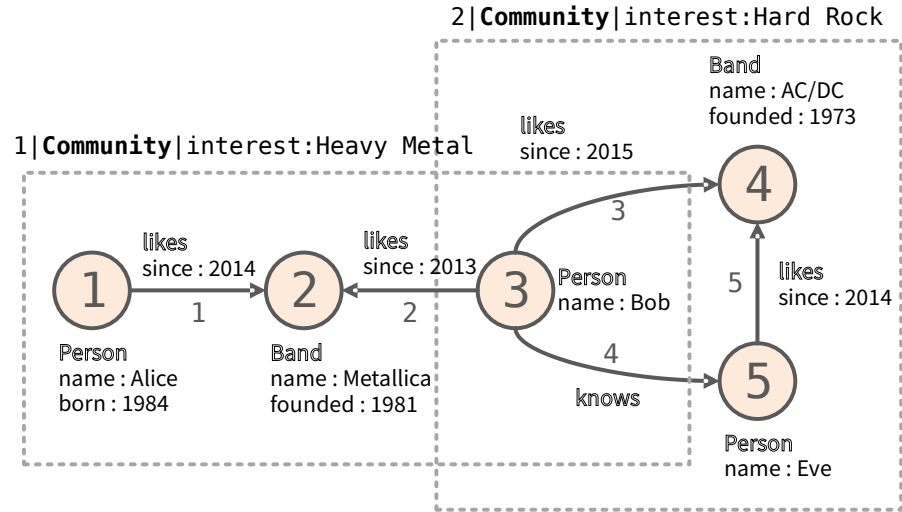


EPGM on Apache Flink

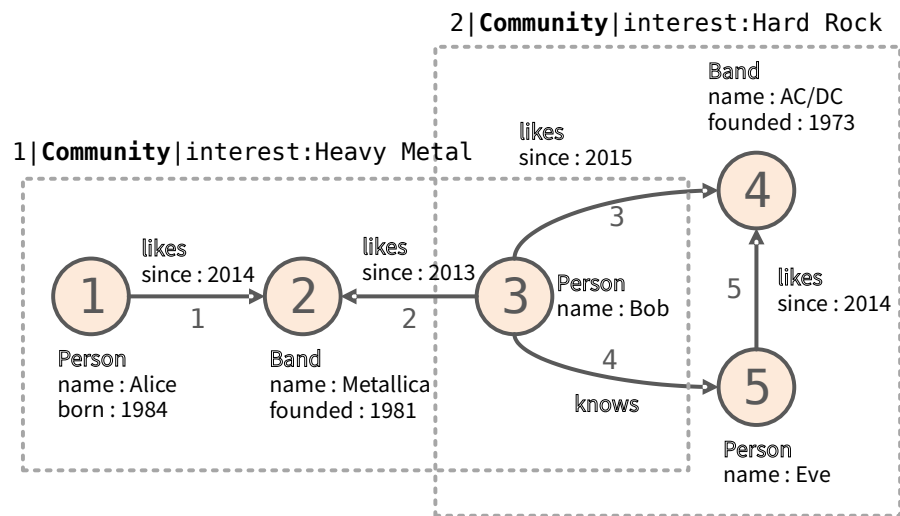
- **DataSet** := Distributed Collection of Data Objects
- **Transformation** := Operation on DataSets
- **Flink Programm** := Composition of Transformations



Graph Representation



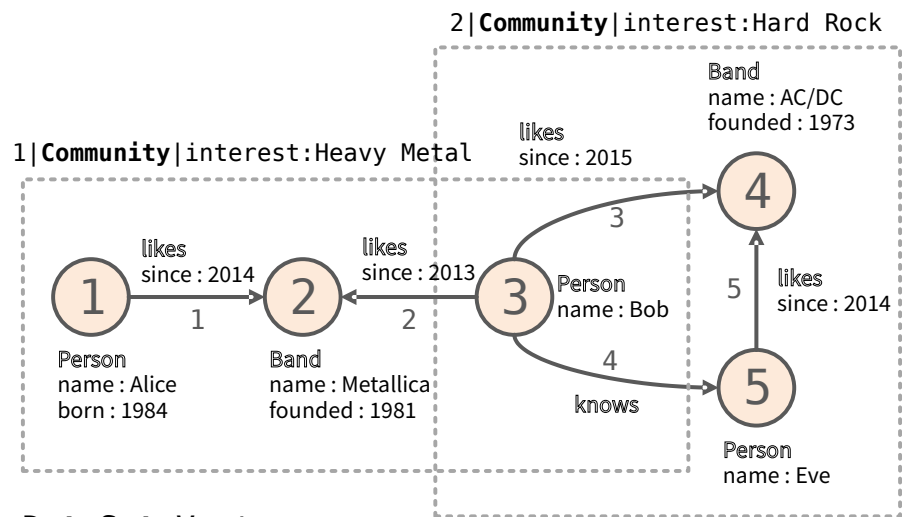
Graph Representation



DataSet<GraphHead>

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

Graph Representation



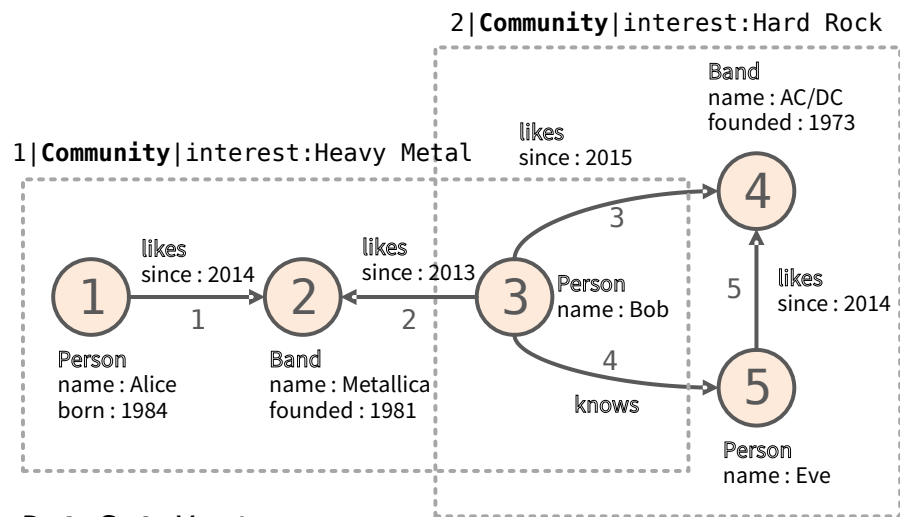
DataSet<Vertex>

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<GraphHead>

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

Graph Representation



DataSet<Vertex>

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<GraphHead>

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

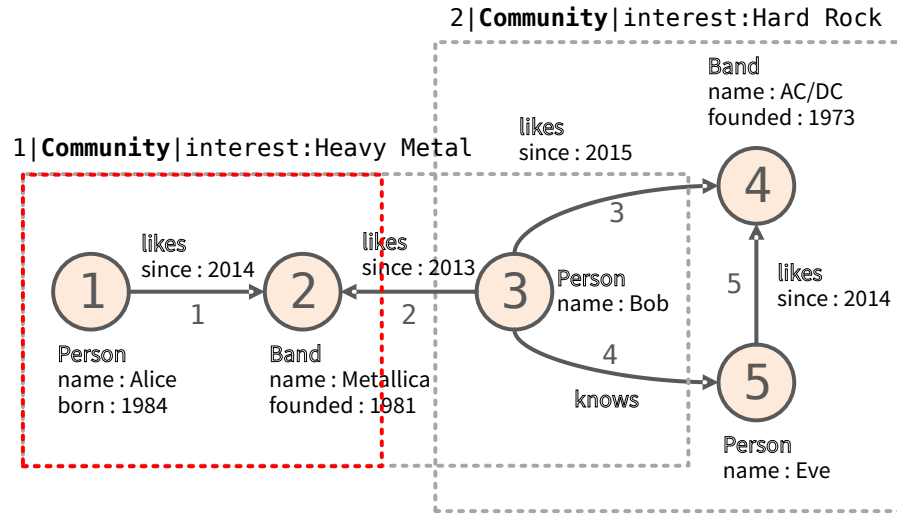
DataSet<Edge>

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}

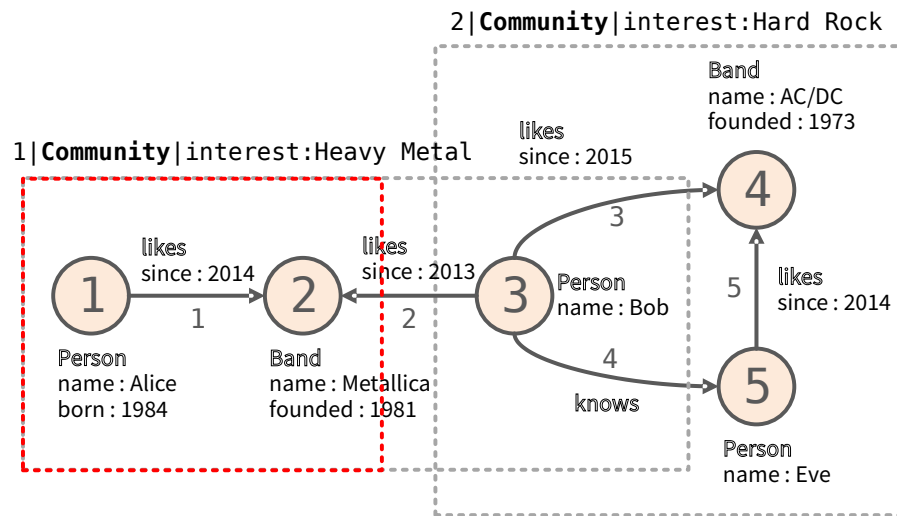
EPGM Operators

		Operators		Algorithms
		Unary	Binary	
Logical Graph		Aggregation	Combination	Flink 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		

Operator Implementation



Operator Implementation



Exclusion

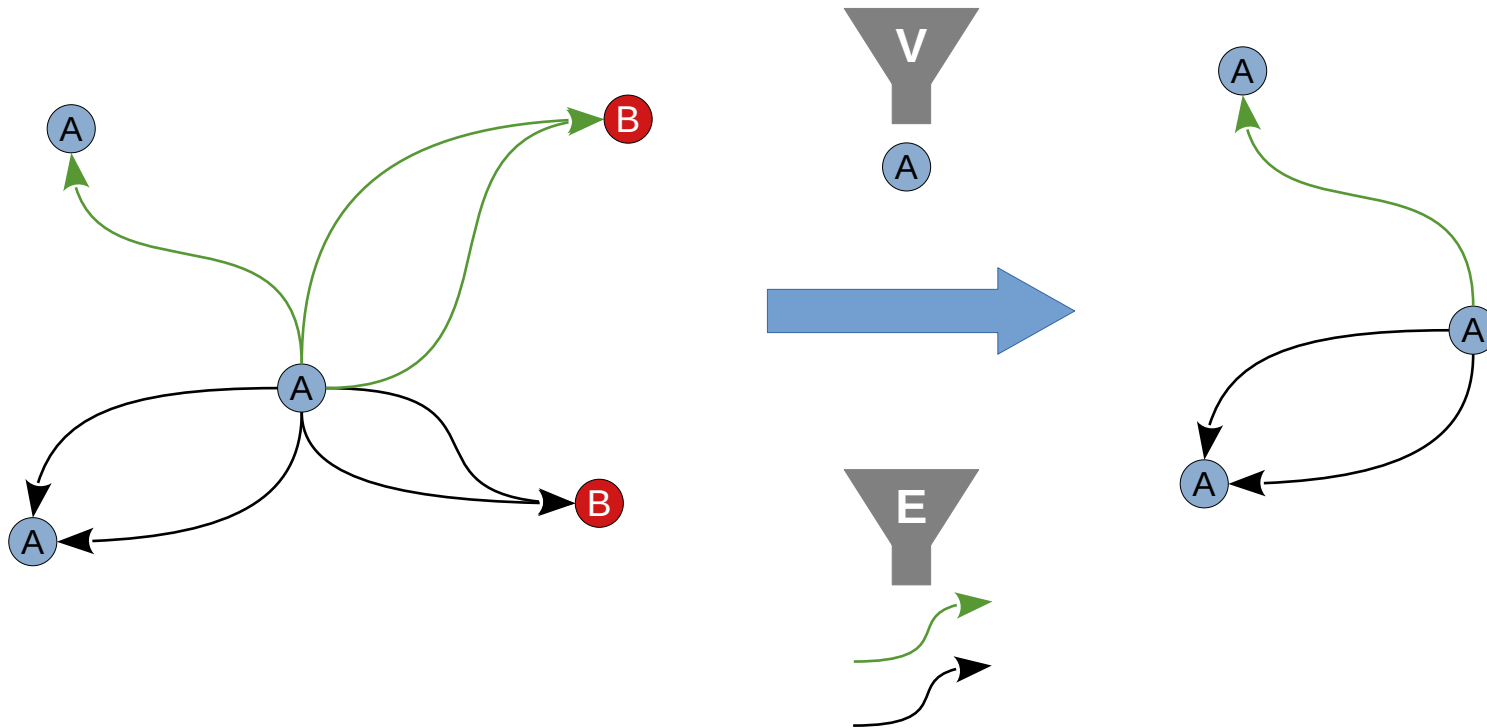
```
// input: firstGraph (G[1]), secondGraph (G[2])
1: DataSet<GradoopId> graphId = secondGraph.getGraphHead()
2:   .map(new Id<G>());
3:
4: DataSet<V> newVertices = firstGraph.getVertices()
5:   .filter(new NotInGraphBroadCast<V>())
6:   .withBroadcastSet(graphId, GRAPH_ID);
7:
8: DataSet<E> newEdges = 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>());
```



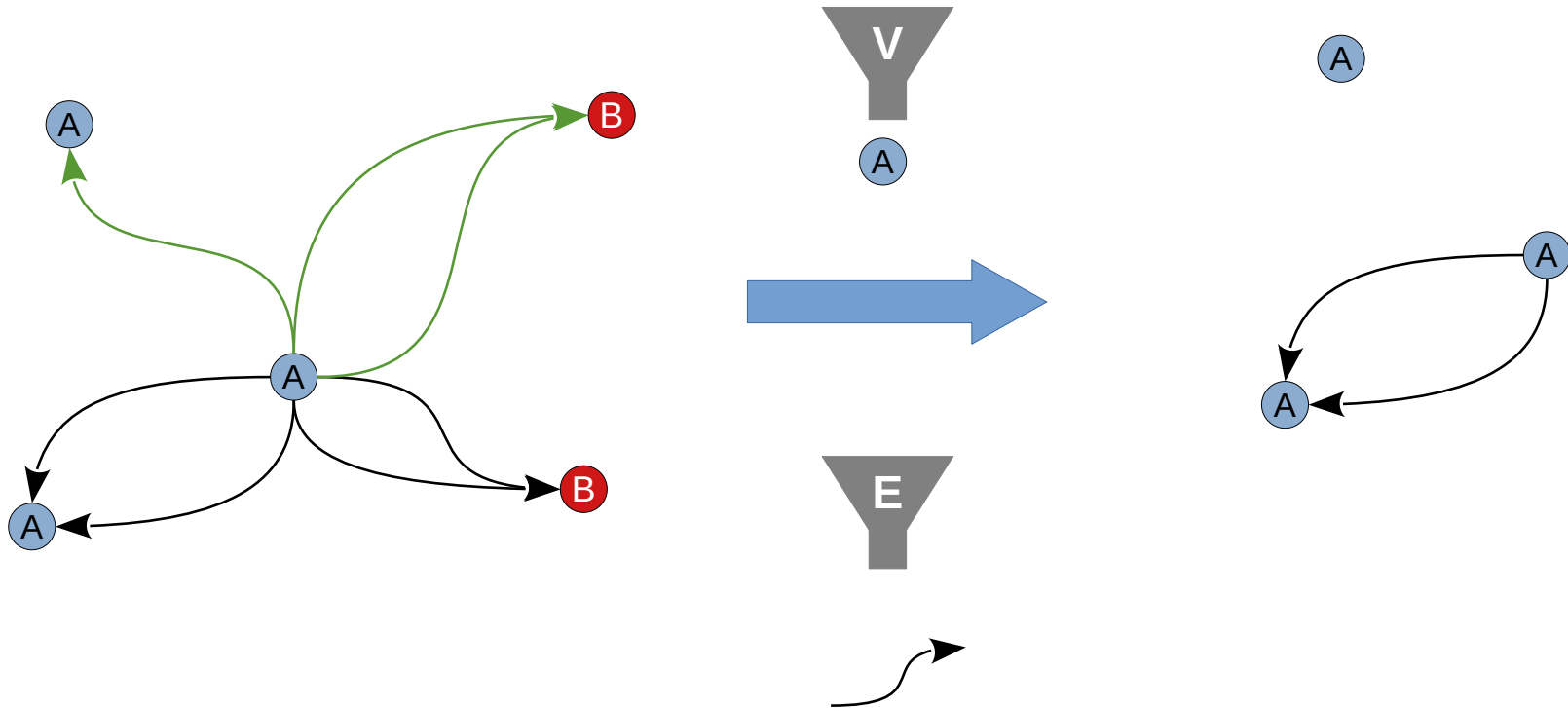
UNIVERSITÄT
LEIPZIG

Practical Tasks

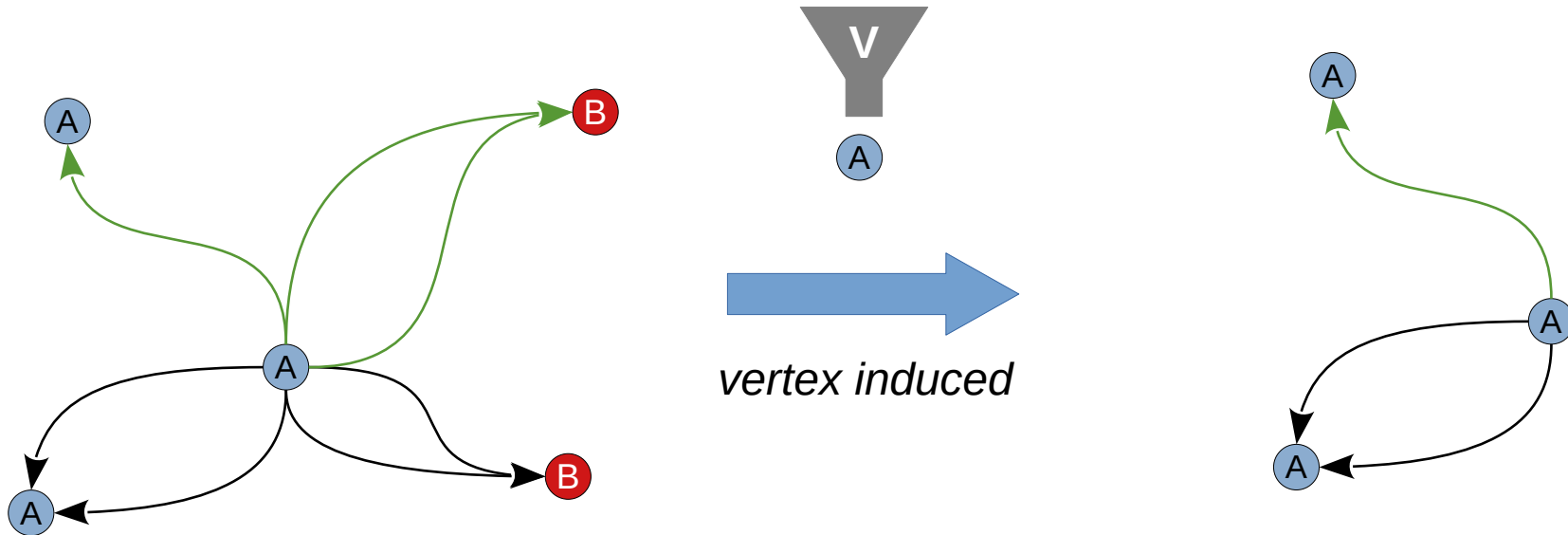
Lesson 1: Subgraph (I)



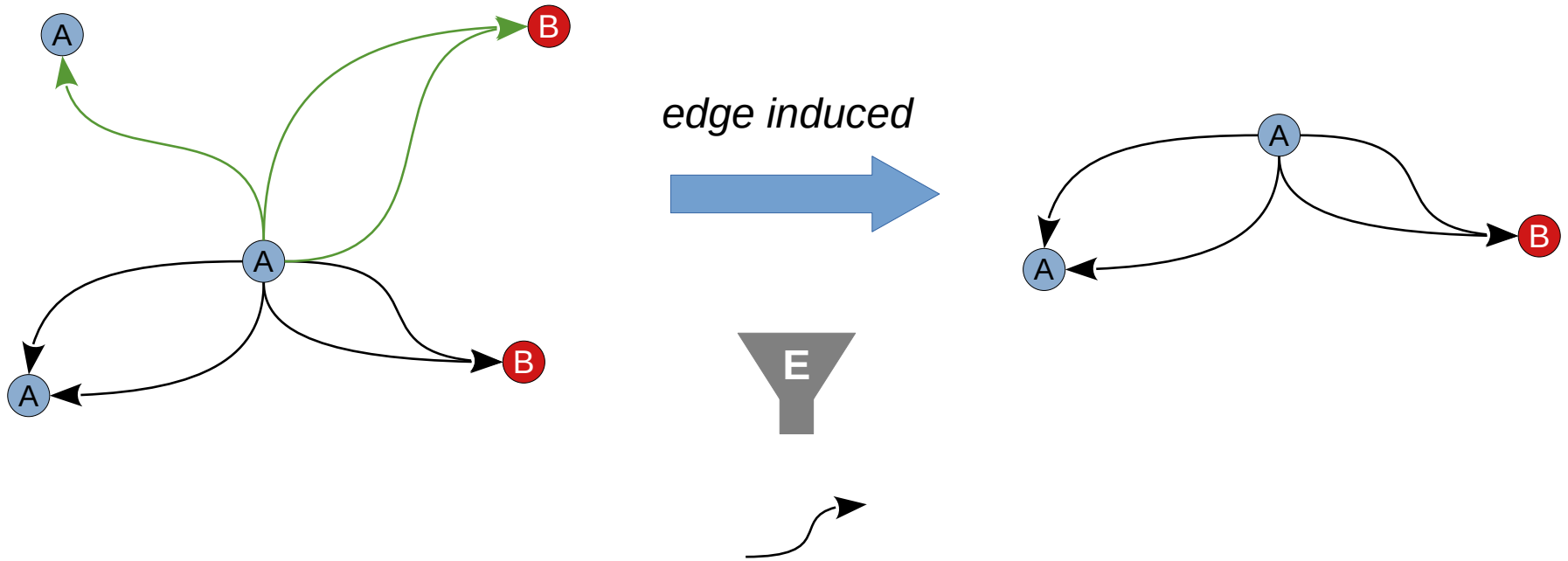
Lesson 1: Subgraph (II)



Lesson 1: Subgraph (III)



Lesson 1: Subgraph (IV)



Lesson 1: Subgraph (V)

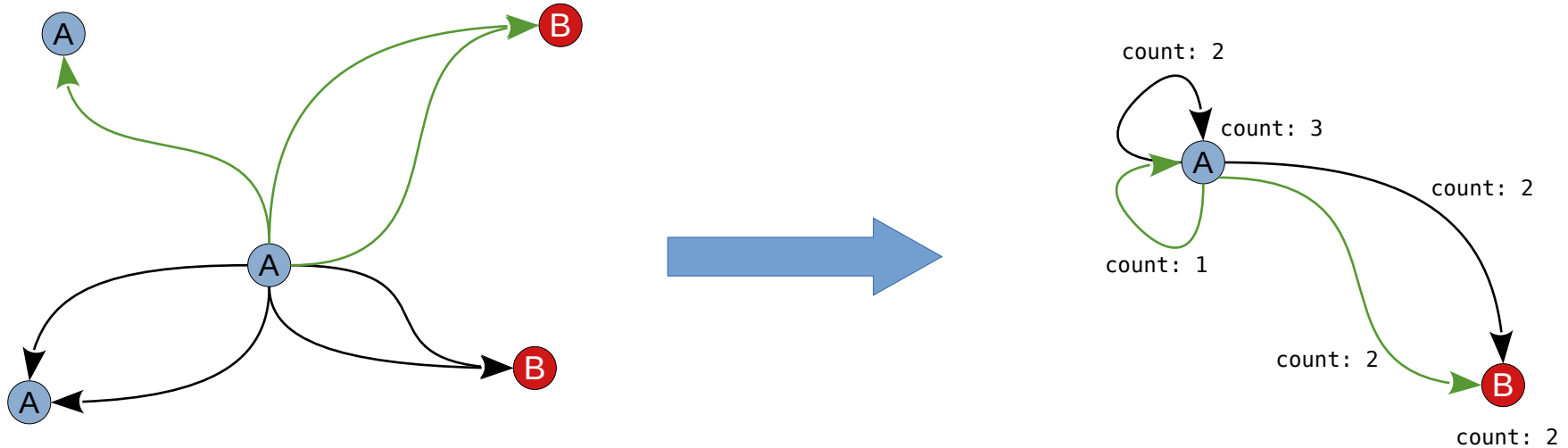
- Extracts a subgraph by applying filter conditions to vertices and edges
 - `myGraph.subgraph(vertexFilter, edgeFilter)`
- Vertex- and Edge-induced filtering supported
 - `myGraph.vertexInducedSubgraph(vertexFilter)`
- Pre-defined filters including logical operators AND, OR and NOT
 - `new And<>(new ByProperty<>("name"), new LabelIsIn("person", "post"))`
- User-defined filters including lambda expressions
 - `myGraph.subgraph(v -> true, e -> e.getLabel().equals("knows"))`

Lesson 1: Subgraph (VI)

Now it's your turn!

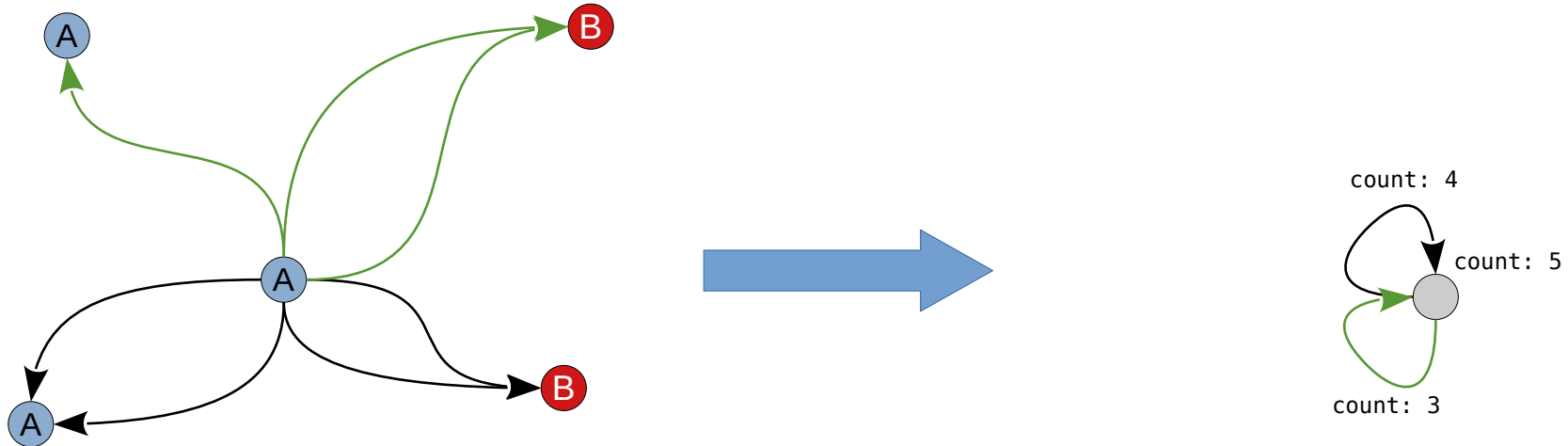
- **Task 1** – Filter by label
 - Create a subgraph that contains ‘person’ vertices and ‘knows’ edges only.
- **Task 2** – Logical operators
 - Get universities and persons that used the browser Firefox.
 - Include all relationships between these vertices.
- **Task 3** – User-defined filter
 - Some edge types have a ‘creationDate’ property. Get all edges that contain this property and are crated in the period [2012-01-01 00:00 , 2012-01-05 00:00) by using a user-defined edge filter function.
 - Only the source and target vertices of these edges should be included.
 - Optional: Use parameters (by adding a constructor) to define the lower and upper bound for the condition.

Lesson 2: Grouping (I)



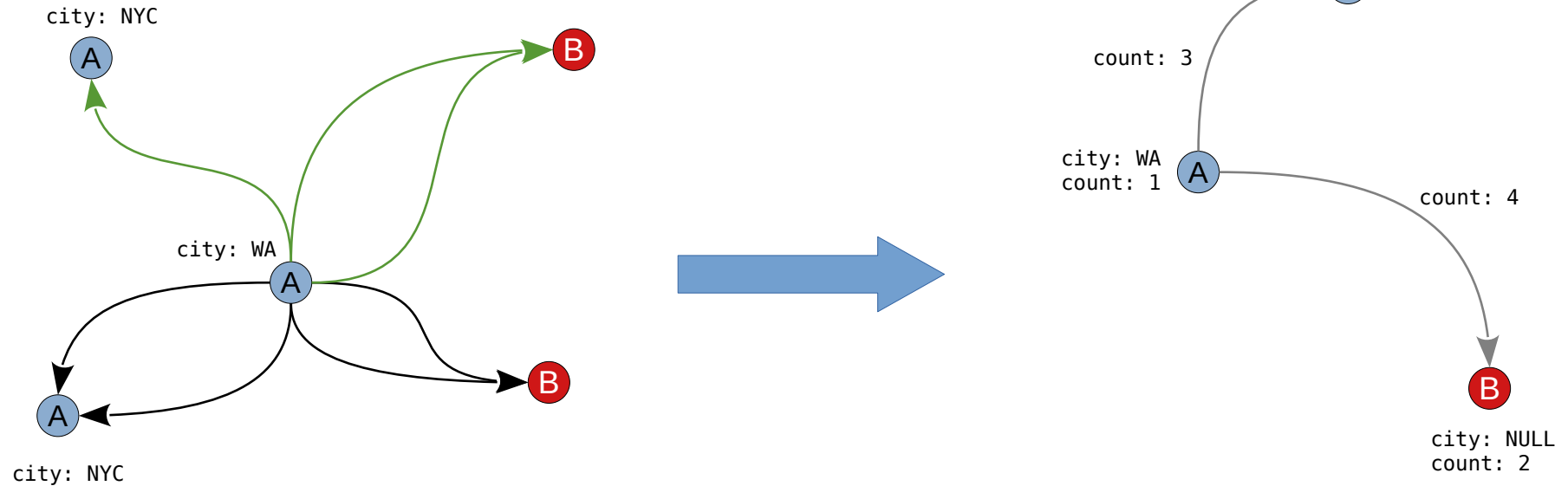
Config:
 Group vertices by label.
 Group edges by label.
 Count vertices.
 Count edges.

Lesson 2: Grouping (II)



Config:
 Group all vertices.
 Group edges by label.
 Count vertices.
 Count edges.

Lesson 2: Grouping (III)



Config:

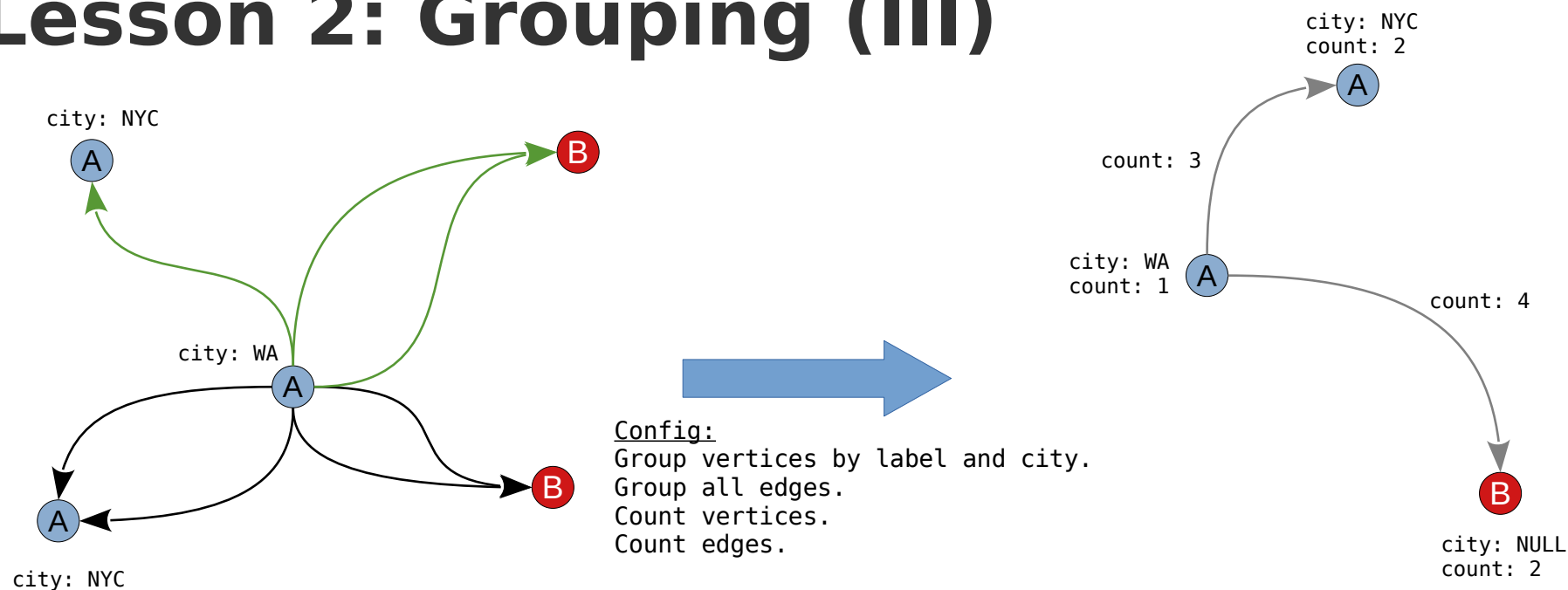
Group vertices by label and city.

Group all edges.

Count vertices.

Count edges.

Lesson 2: Grouping (III)



```
myGraph.callForGraph(
  new KeyedGrouping<>(
    Arrays.asList(GroupingKeys.label(), GroupingKeys.property("city")), // V grouping keys
    Arrays.asList(new Count()), // V aggregates
    null, // E grouping keys
    Arrays.asList(new Count()) // E aggregates
  ));
```

Lesson 2: Grouping (IV)

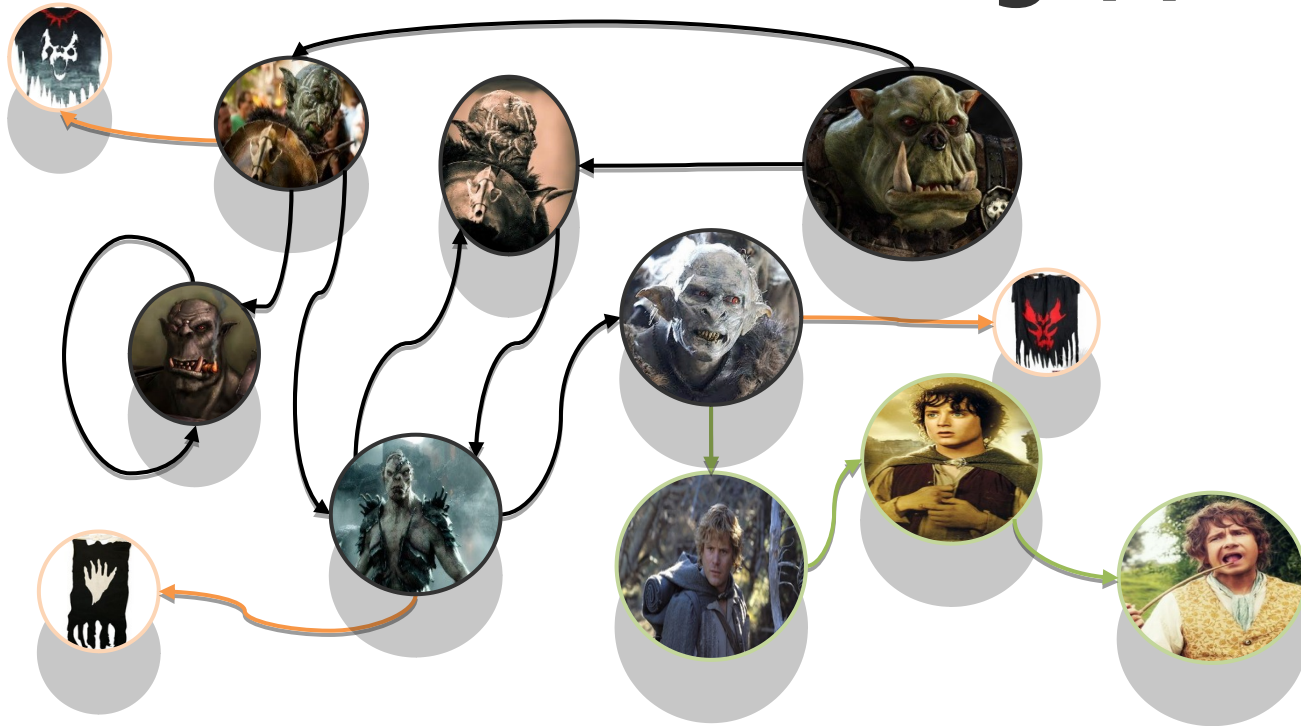
- Structural grouping based on *key functions*
- Aggregations per group by *aggregate functions*
- *Key function*: Mapping from Vertex/Edge to grouping value
 - `GroupingKeys.label()` returns the label of the element
 - `GroupingKeys.property('name')` returns the value of the property 'name' or NULL, if there is no property 'name'
- *Aggregate functions*: compute aggregated property values
 - e.g., `new Count('cnt')` counts the number of grouped elements, creates a property 'cnt' on the grouped element

Lesson 2: Grouping (V)

Now it's your turn!

- **Task 1** - Schema graph
 - Create a schema graph by grouping vertices and edges by the label.
 - How many vertices and edges exist per label?
- **Task 2** - Attributed grouping
 - How many males and females are there? What is the average age per gender?
 - How many males know women and vice versa?
 - How many male and female people study at universities per class year?
- **Task 3** - User-defined key function
 - How many people are born at the same weekday (Mon - Sun)?
 - You have to create a UDF that extracts the day from the property 'birthday'.
 - How old is the youngest and oldest person in the group?
 - How many know each other from these groups?

Lesson 3: Pattern Matching (I)



„Which two clan leaders hate each other and one of them knows Frodo over one to ten hops?“

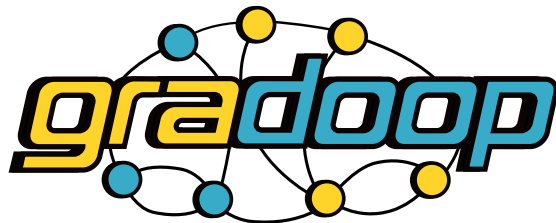
Lesson 3: Pattern Matching (II)



Cypher

```
1 MATCH (c1:Clan)<-[:leaderOf]-(o1:Orc),
2      (o1)-[:hates]->(o2:Orc),
3      (o2)-[:leaderOf]->(c2:Clan),
4      (o2)-[:knows*1..10]->(h:Hobbit)
5 WHERE NOT (c1 = c2 OR o1 = o2)
6          AND h.name = "Frodo Baggins"
7 RETURN o1.name, o2.name;
```


Lesson 3: Pattern Matching (III)








Graph Definition Language (GDL)

```
GraphCollection collection = graph.query(' MATCH (c1:Clan)-[:leaderOf]-(o1:Orc),  
      (o1)-[:hates]->(o2:orc),  
      (o2)-[:leaderOf]->(c2:Clan),  
      (o2)-[:knows*1..10]->(h:Hobbit),  
      WHERE NOT (c1 = c2 OR o1 = o2)  
      AND h.name = "Frodo Baggins"',  
      construct, statistics);
```

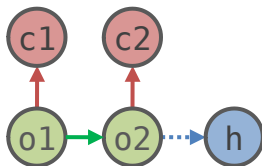
Lesson 3: Pattern Matching (IV)

```
graph.query('MATCH (c1:Clan)-[:LeaderOf]-(o1:Orc),
            (o1)-[:hates]->(o2:orc),
            (o2)-[:LeaderOf]->(c2:Clan),
            (o2)-[:knows*1..10]->(h:Hobbit),
            Where NOT (c1 = c2 OR o1 = o2)
            AND h.name = "Frodo Baggins"',
            construct, statistics);
```

 => 23
 => 42
 => 84
 => 123
 => 456
 => 789

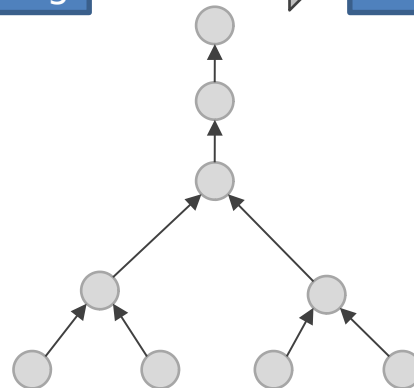
ANTLR

Parsing

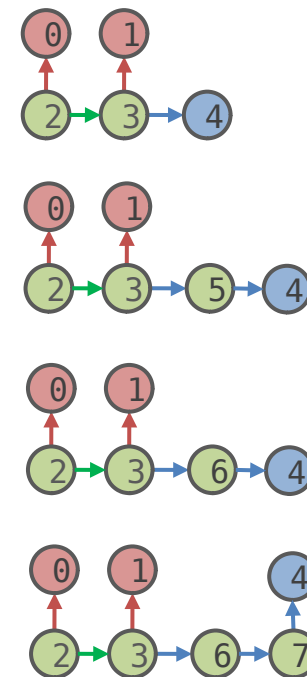


$(c1 \neq c2) \text{ AND } (o1 \neq o2)$
 $\text{AND } (h.name = \text{Frodo Baggins})$

Planning



Execution

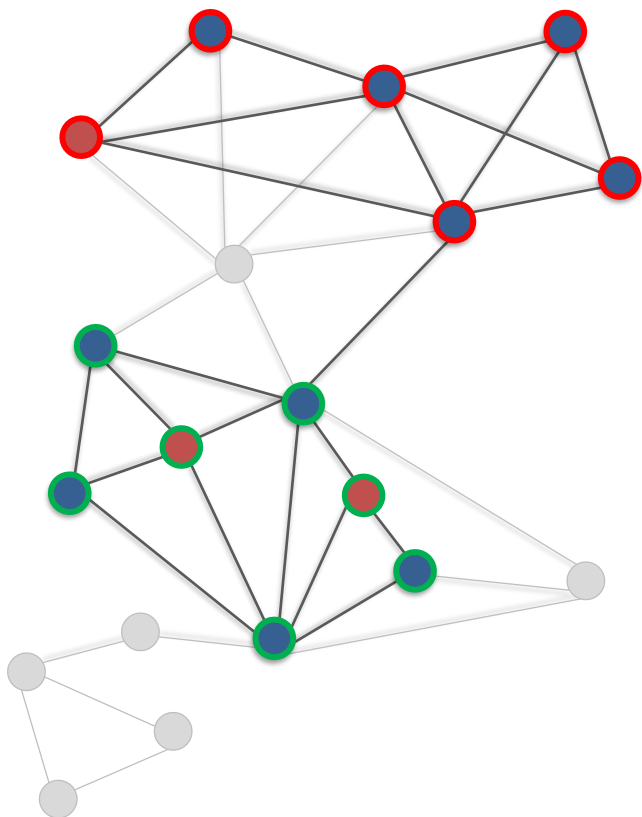


Lesson 3: Pattern Matching (V)

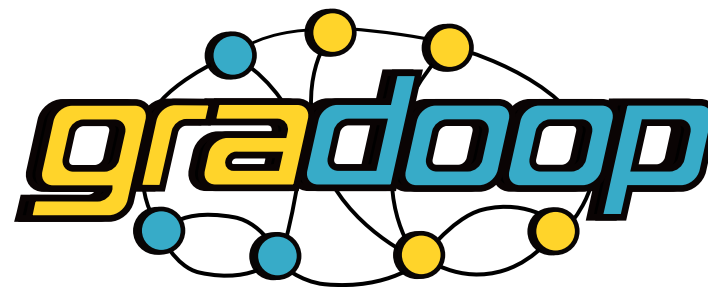
Now it's your turn!

- **Task 1** - Friendship Graph
 - Find all persons that know "John Jones".
- **Task 2** - Locations of Persons and Universities
 - Find all entities that are located in Bangkok
- **Task 3** - Construct Pattern
 - Find all Persons that are creators of Posts located in "Senegal"
 - Posts should not be in the result set (use construct pattern)
 - See the documentation about construct pattern in the Gradoop Wiki!

Lesson 4: Complex Analysis (I)



Let's do a more complex analysis using the operators of **GRADOOP** :)



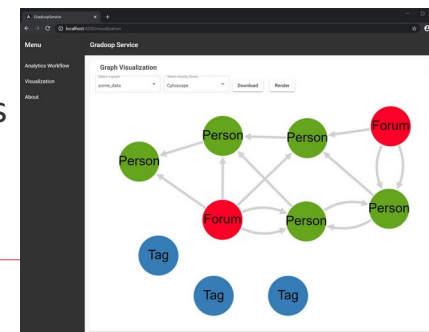
Lesson 3: Pattern Matching (V)

Now it's your turn!

- **Task** - Distribution of male and females students at the given universities.
 - Create a graph that show how many males and females are studying at the universities contained in the test data.
- **Bonus** - Time left? Playground!
 - Play around with the operators Gradoop offers to learn more about the possibilities of distributed graph analytics.
 - Execute different Operators and Algorithms on Logical Graphs or Graph Collections.
 - Take a look at our publications about Gradoop

Ongoing research

- Real-world graphs evolve over time
 - E.g., social relationships were added, messages were sent, transactions were made
 - Analyzing a graph with respect to it's evolution requires a suitable data model and operators
 - We therefore extended the EPGM to the **T**emporal **P**roperty **G**raph **M**odel (TPGM)
- Graph-Stream analysis
 - Analyze a stream of relationships for online analysis
 - Graph-Stream grouping, pattern matching and visualization are already prototypes
- Entity resolution
 - Integrating graph data from different sources to a knowledge graph requires complex ER tasks
 - **F**ast **M**ulti-source **E**ntity **R**esolution system (FAMER) is part of current research and based on Gradoop
- Gradoop-Service
 - A graphical user interface to use Gradoop without programming skills
 - Operator configuration and result visualization

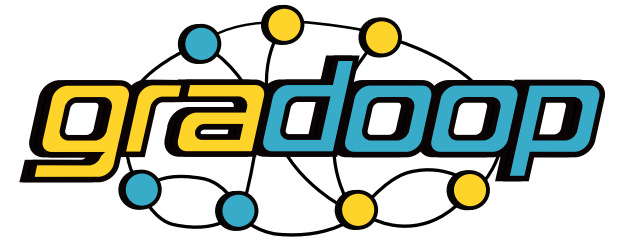


Literature

- K. Gomez, M. Taschner, M. A. Rostami, C. Rost, and E. Rahm. Graph sampling with distributed in-memory dataflow systems. arXiv preprint arXiv:1910.04493, **2019**.
- M. Junghanns et al. GRADOOP: scalable graph data management and analytics with hadoop. CoRR, **2015**.
- M. Junghanns et al. Analyzing Extended Property Graphs with Apache Flink. In Proc. SIGMOD NDA Workshop, **2016**.
- M. Junghanns, M. Kießling, A. Averbuch, A. Petermann, and E. Rahm. Cypher-based graph pattern matching in gradoop. In Proc. GRADES, **2017**.
- M. Junghanns, A. Petermann, M. Neumann, and E. Rahm. Management and Analysis of Big Graph Data: Current Systems and Open Challenges. In Handbook of Big Data Technologies. Springer, **2017**.
- M. Junghanns, A. Petermann, and E. Rahm. Distributed Grouping of Property Graphs with GRADOOP. In Proc. BTW, **2017**.
- A. Petermann et al. BIIG: Enabling Business Intelligence with Integrated Instance Graphs. In Proc. ICDEW, **2014**.
- A. Petermann et al. Dimspan - transactional frequent subgraph mining with distributed in-memory dataflow systems. Proc. BDCAT, **2017**.
- C. Rost, A. Thor, P. Fritzsche, K. Gomez, and E. Rahm. Evolution analysis of large graphs with gradoop. In PKD-D/ECML Workshops (1), volume 1167 of Communications in Computer and Information Science, pages 402–408. Springer, **2019**.
- C. Rost, A. Thor, and E. Rahm. Analyzing temporal graphs with gradoop. Datenbank-Spektrum, 19(3):199–208, **2019**.
- M. A. Rostami, M. Kricke, E. Peukert, S. Kühne, M. Wilke, S. Dienst, and E. Rahm. BIGGR: Bringing gradoop to applications. Datenbank-Spektrum, 19:51–60, **2019**.
- M. A. Rostami, E. Peukert, M. Wilke, and E. Rahm. Biggraph analysis by visually created workflows. In BTW 2019, pages 559–563. Gesellschaft für Informatik, Bonn, **2019**.



UNIVERSITÄT
LEIPZIG



Thank you for participating

You learned:

- What Gradoop is and how it works
- How single operators work and how they can be configured
- How operators can be composed to build a more complex program
- How a Gradoop program can be executed on a cluster