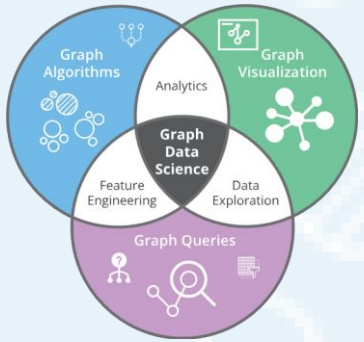


Habib University
shaping futures

CS343 Graph Data Science

Spring 2024

Introduction to Graph Algorithms, GDS Library and Projections

Chapter #4, Estelle, Chapter #6, Tomaz

Muhammad Qasim Pasta

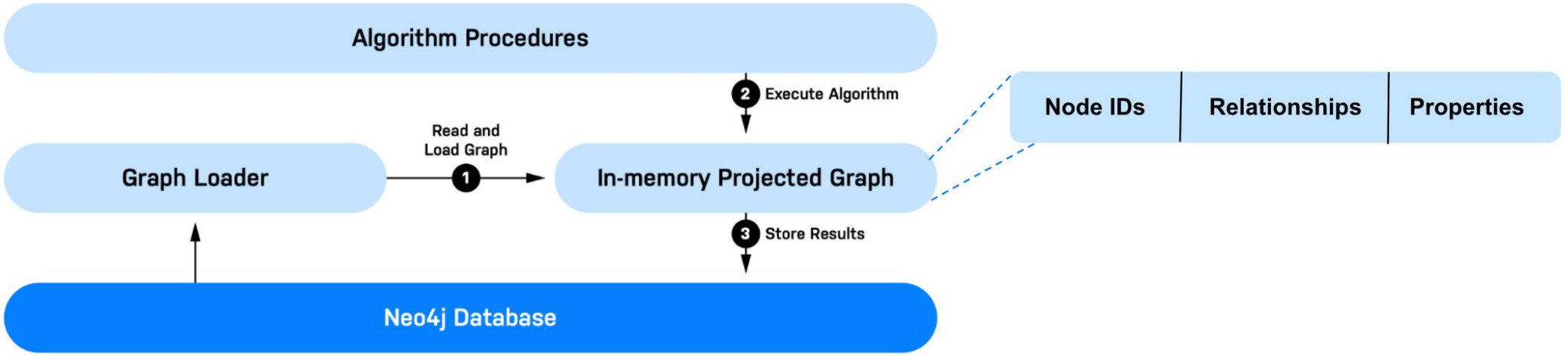
qasim.pasta@sse.habib.edu.pk

Slides are intended to be filled during the lectures. Certain details are intentionally omitted for in-class discussions. These slides are not meant to be used as reading material.

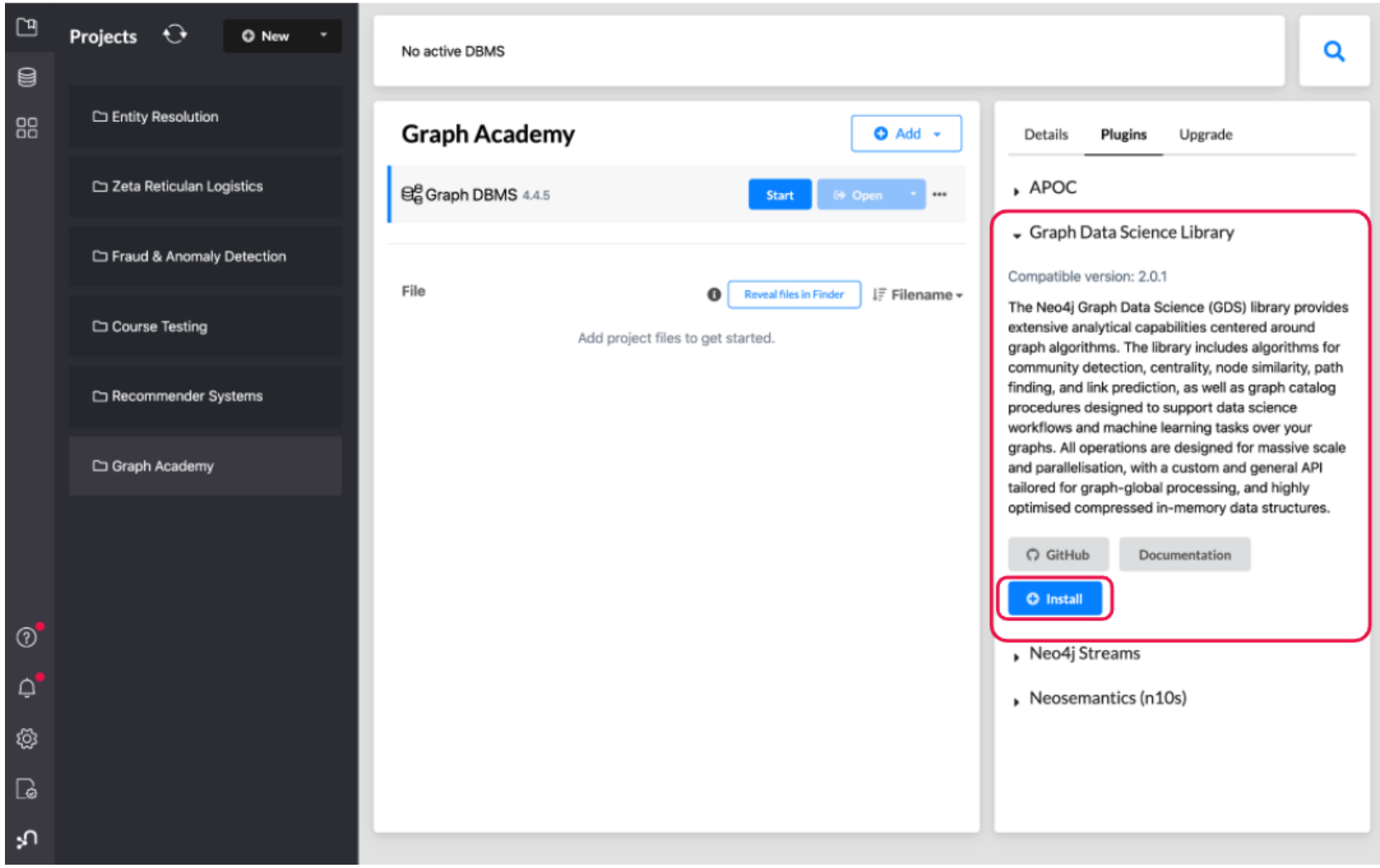
Graph Data Science Library (GDS)

- In 2020, release as GDS, earlier, released as Graph Algorithm plugin, 2019.
- Contains tools to be used in a data science project using data stored in Neo4j
- Path-related algorithms:
  - traversing a graph to find specific paths from one node to another
- Graph algorithms
  - extract some kind of information from the graph structure itself
  - Centrality, Community Detection, Similarity Algorithms
- Machine learning (ML) models and pipelines
  - Algorithms for Embedding: f transforming a high-dimensional object into a low-dimension vector; such as topology for the graph.
  - Node classification, link prediction

Working


Algorithm Procedures
2 Execute Algorithm
Read and
Load Graph
Graph Loader
1
In-memory Projected Graph
Node IDs
Relationships
Properties
3 Store Results
Neo4j Database

Installation



gds.version()

gds.list()

```
neo4j$ CALL gds.list
```

Table | Text | Code

	name	description
1	"gds.allShortestPaths.delta.mutate"	"The Delta Stepping shortest path algorithm computes the shortest (weighted) p
2	"gds.allShortestPaths.delta.mutate.estimate"	"Returns an estimation of the memory consumption for that procedure."
3	"gds.allShortestPaths.delta.stats"	"The Delta Stepping shortest path algorithm computes the shortest (weighted) p
4	"gds.allShortestPaths.delta.stats.estimate"	"Returns an estimation of the memory consumption for that procedure."
5	"gds.allShortestPaths.delta.stream"	"The Delta Stepping shortest path algorithm computes the shortest (weighted) p
6	"gds.allShortestPaths.delta.stream.estimate"	"Returns an estimation of the memory consumption for that procedure."

Algorithm Execution Mode

GDS algorithms have 4 executions modes which determine how the results of the algorithm are handled.

1. **stream:** Returns the result of the algorithm as a stream of records.
2. **stats:** Returns a single record of summary statistics, but does not write to the Neo4j database or modify any data.
3. **mutate:** Writes the results of the algorithm to the in-memory graph projection and returns a single record of summary statistics.
4. **write:** Writes the results of the algorithm back the Neo4j database and returns a single record of summary statistics.

Workflow

- Projected Graph: used to run all algorithms from the GDS
  - Only a certain node label(s) / relationship(s)
  - Only certain properties
  - New relationship/properties computed on the fly

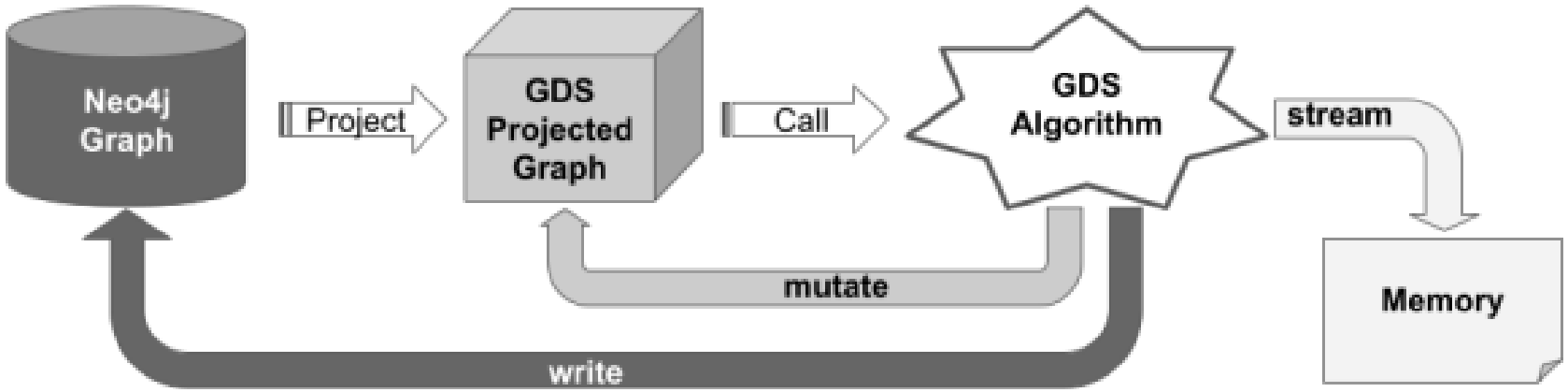


Figure 4.3 – GDS workflow

Two kind of projects

Native Projection

- Nodes, relationships, and properties are selected from the database with a standardized configuration

Cypher Projection

- Nodes, relationships, and properties are filtered or created on the fly using Cypher queries

Native Projection: gds.graph.project()

```
CALL gds.graph.project(
     <graphName>,
     <nodeProjectionConfig>,
     <relationshipProjectionConfig>
)
```

```
CALL gds.graph.project(
       'native-proj',
       ['User'],
       "*"
)
```

```
gds.graph.drop()
```

Name	Type	Optional	Description
graphName	String	no	The name under which the graph is stored in the catalog.
nodeProjection	String, List or Map	no	The configuration for projecting nodes.
relationshipProjection	String, List or Map	no	The configuration for projecting relationships.
configuration	Map	yes	Additional parameters to configure the native projection.

Changing Relationship Nature

```
CALL gds.graph.project(
      'native-proj',
      ['User', 'Movie'],
      {RATED_BY: {
            type: 'RATED',
             orientation: 'REVERSE'}
      }
);
```

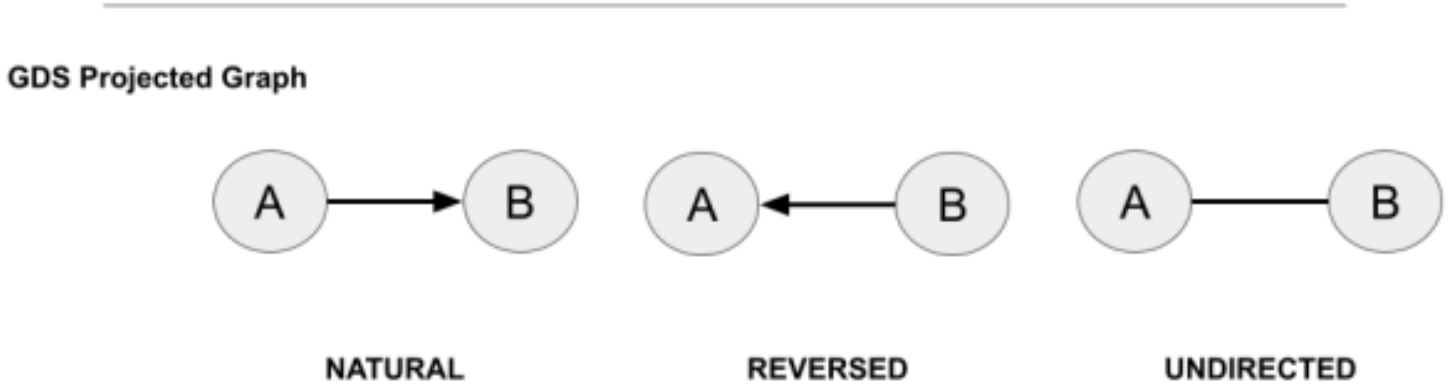
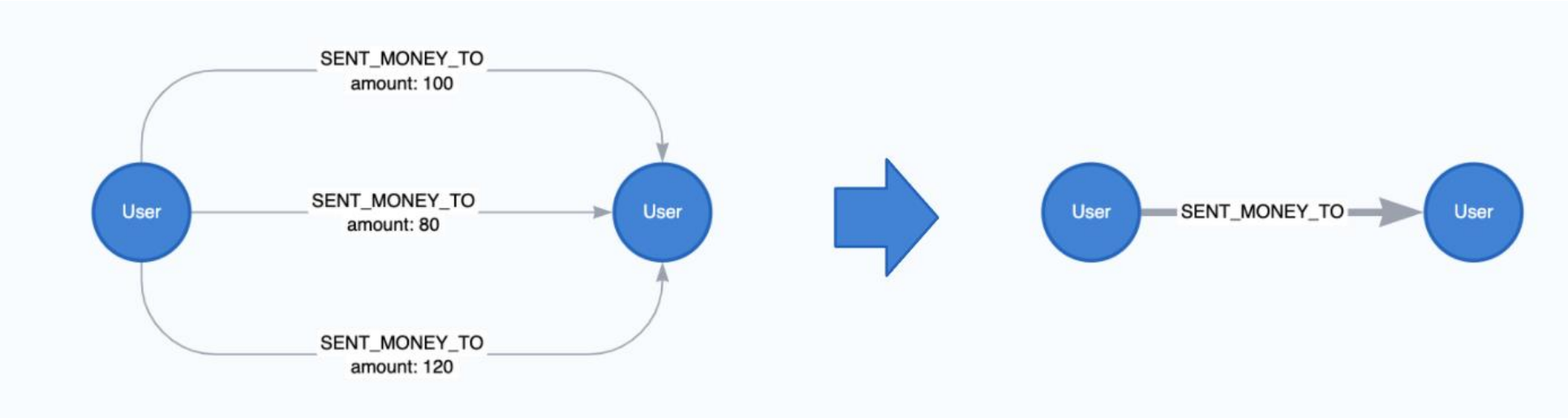


Figure 4.5 – Relationship configuration in a GDS projected graph

Parallel Relationship Aggregations

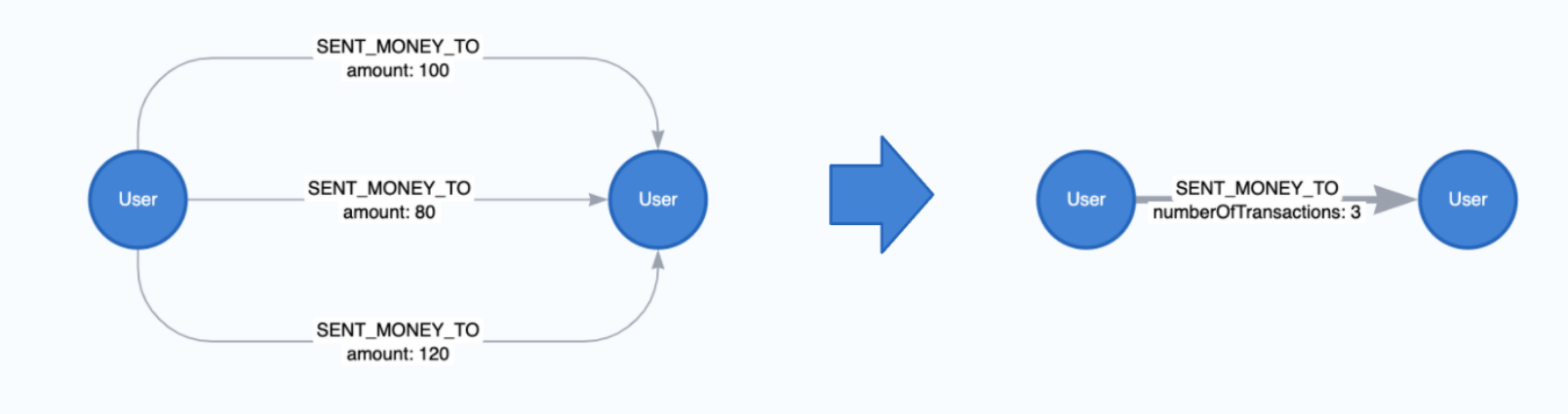
- merge the relationships and keep only one




```
CALL gds.graph.project(
        'user-proj',
        ['User'],
        {
                SENT_MONEY_TO: { aggregation: 'SINGLE' }
        }
);
```

Parallel Relationship Aggregations: Count

- merge the relationships and keep only one

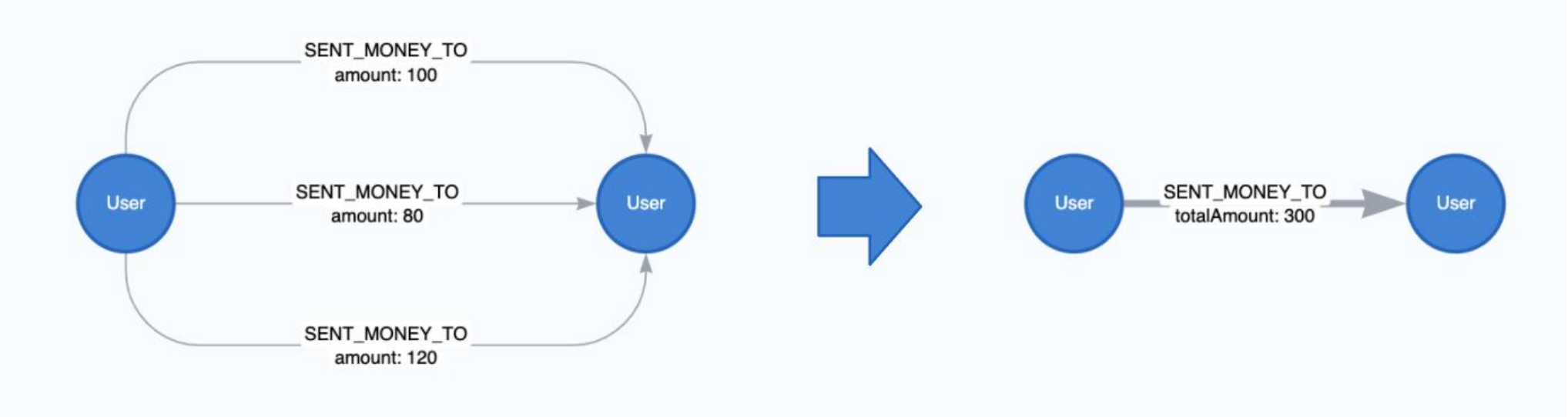


Parallel Relationship Aggregations: Count

- merge the relationships and keep only one

```
CALL gds.graph.project(
       'user-proj',
       ['User'],
       {
              SENT_MONEY_TO: {
                     properties: {
                            numberOfTransactions: {
                                   property: '*',
                                   aggregation: 'COUNT' }
                     }
              }
       }
);
```

Parallel Relationship Aggregations: Sum



Parallel Relationship Aggregations: Sum

```
CALL gds.graph.project(
       'user-proj',
       ['User'],
       {
              SENT_MONEY_TO: {
                     properties: {
                            numberOfTransactions: {
                                   property: 'amount',
                                   aggregation: 'SUM' }
                     }
              }
       }
);
```

Cypher Projection

```
CALL gds.graph.project.cypher(
       'proj-cypher’,
       'MATCH (a:Actor) RETURN id(a) AS id, labels(a) AS labels’,
       'MATCH (a1:Actor)-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(a2)
              WHERE m.year >= 1990 AND m.revenue >= 1000000
              RETURN id(a1) AS source , id(a2) AS target, count(*) AS
                     actedWithCount, "ACTED_WITH" AS type' );
```