Graph Data Models

Oscar Romero
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Graph Data Model in a Nutshell

- Occurrence-oriented
  - It is a schemaless data model
    - There is no explicit schema
    - Data (and its relationships) may quickly vary
  - Objects and relationships as first-class citizens
    - *An object o relates (through a relationship r) to another object o'*
      - *Such relationship is often known as a triple (o r o')*
    - Both objects and relationships may contain properties
  - Built on top of the graph theory
    - Euler (18th century)
    - More natural and intuitive than the relational model to deal with relationships

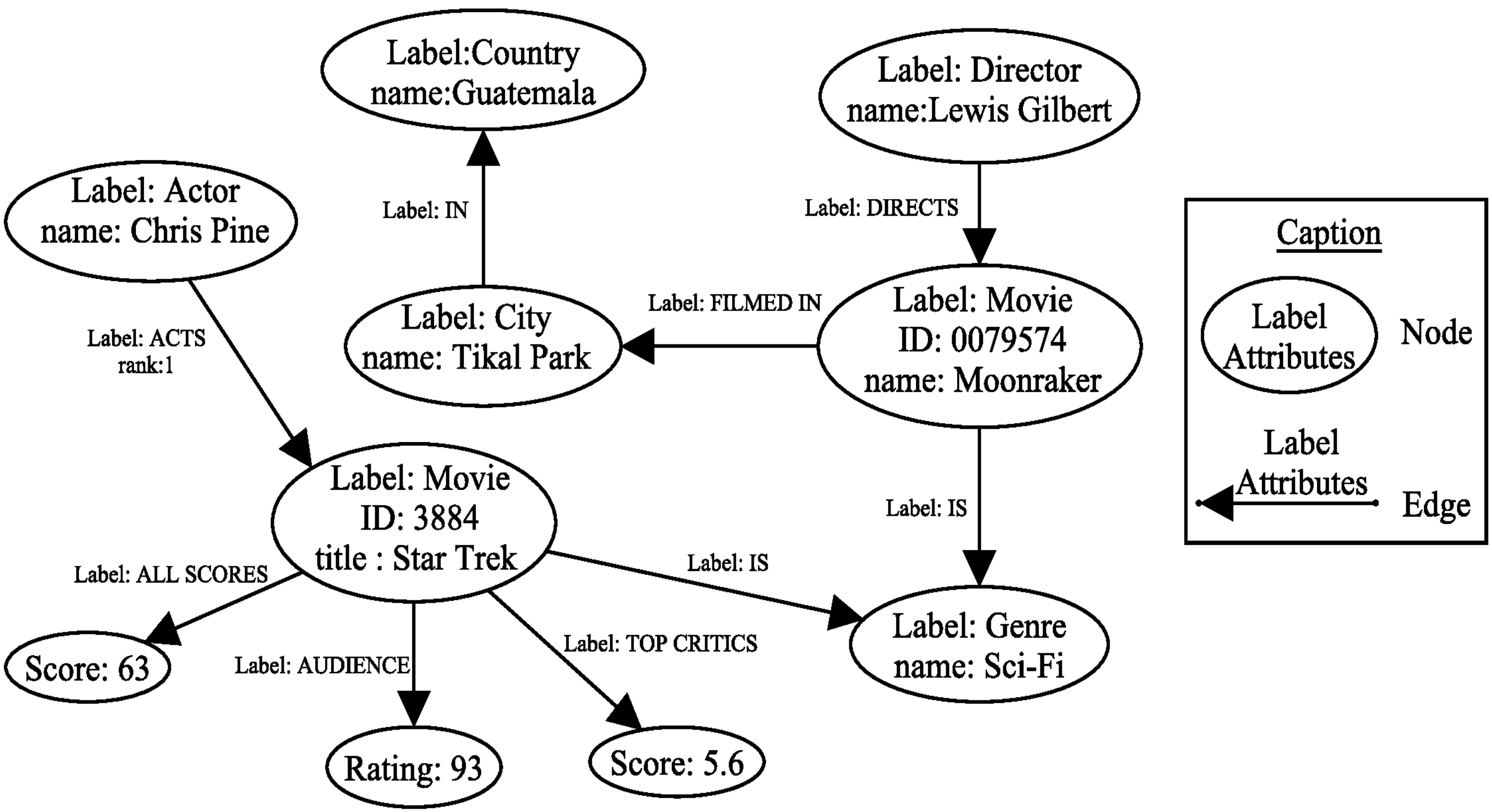
Notation (I)

- A **graph** *G* is a set of nodes and edges: *G (N, E)*
- *N* - **Nodes** (or vertices): n_1, n_2, … N_m
- *E* - **Edges** are represented as pairs of nodes: (n_1, n_2)
  - An edge is said to be **incident** to n_1 and n_2
  - Also, n_1 and n_2 are said to be **adjacent**
  - An edge is drawn as a line between n_1 *and* n_2
  - **Directed edges** entail direction: *from* n_1 *to* n_2
  - An edge is said to be **multiple** if there is another edge exactly relating the same nodes
  - An **hyperedge** is an edge inciding in more than 2 nodes.
- **Multigraph**: If it contains at least one multiple edge.
- **Simple graph**: If it does not contain multiple edges.
- **Hypergraph**: A graph allowing hyperedges.

Notation (II)

- **Size** (of a graph): #edges
- **Degree** (of a node): #(incident edges)
  - The degree of a node denotes the node adjacency
  - The neighbourhood of a node are all its adjacent nodes
- **Out-degree** (of a node): #(edges leaving the node)
  - Sink node: A node with 0 out-degree
- **In-degree** (of a node): #(incoming edges reaching the node)
  - Source node: A node with 0 in-degree
- Cliques and trees are specific kinds of graphs
  - **Clique**: Every node is adjacent to every other node
  - **Tree**: A connected acyclic simple graph

Example

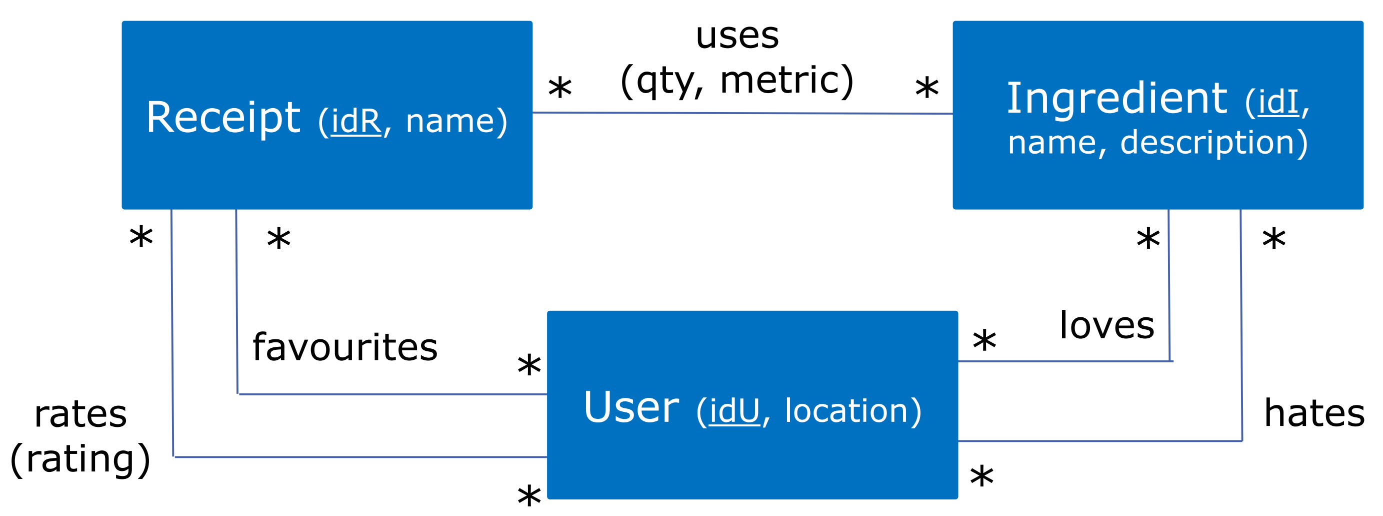

Label:Country
name:Guatemala
Label: Director
name:Lewis Gilbert
Label: Actor
name: Chris Pine
Label: IN
Label: DIRECTS
Caption
Label: ACTS
rank:1
Label: City
name: Tikal Park
Label: FILMED IN
Label: Movie
ID: 0079574
name: Moonraker
Label
Attributes
Node
Label
Attributes
Edge
Label: Movie
ID: 3884
title : Star Trek
Label: IS
Label: ALL SCORES
Label: IS
Label: TOP CRITICS
Label: Genre
name: Sci-Fi
Score: 63
Label: AUDIENCE
Rating: 93
Score: 5.6

Pros and Cons of Graphs

COMPARISON WITH OTHER DATA MODELS

Activity: Comparison with other Data Models

- (5’) Refresh the main data models
  - (10’) Consider the UML class diagram below:
    - Propose a sound relational **and** a graph schema capturing as much semantics as possible. Also, optimise your design based on the following queries:
      - For a given receipt, give me all the users that favorited it
      - For a given receipt, the list of ingredients it contains with their quantity and metric
      - For a given ingredient, how many users do hate it
      - For a given ingredient, all the receipts each participate
      - For a given user, all the ingredients he loves
      - For a given user, all the rates he gave
    - If you are familiar with key-value and document-stores, propose sound schemata capturing as much semantics as possible
    - What are the pros and cons of each data model?



Pros and Cons

Graphs

- They are occurrence-oriented
- **Occurrences** are **pointed by / point to** related occurrences
  - Query operators do not rely on schema
  - Naturally facilitate data linking
- The schema information is embedded together with data
  - The concept of stand-alone catalog does not exit
- Purely schemaless
  - Semantics are fixed by the edge / node labels
- Difficult to benefit from sequential access. Typically, it relies on random accesses
- By definition, it follows an Open-World assumption (i.e., assumes incomplete data)

Key-oriented Models

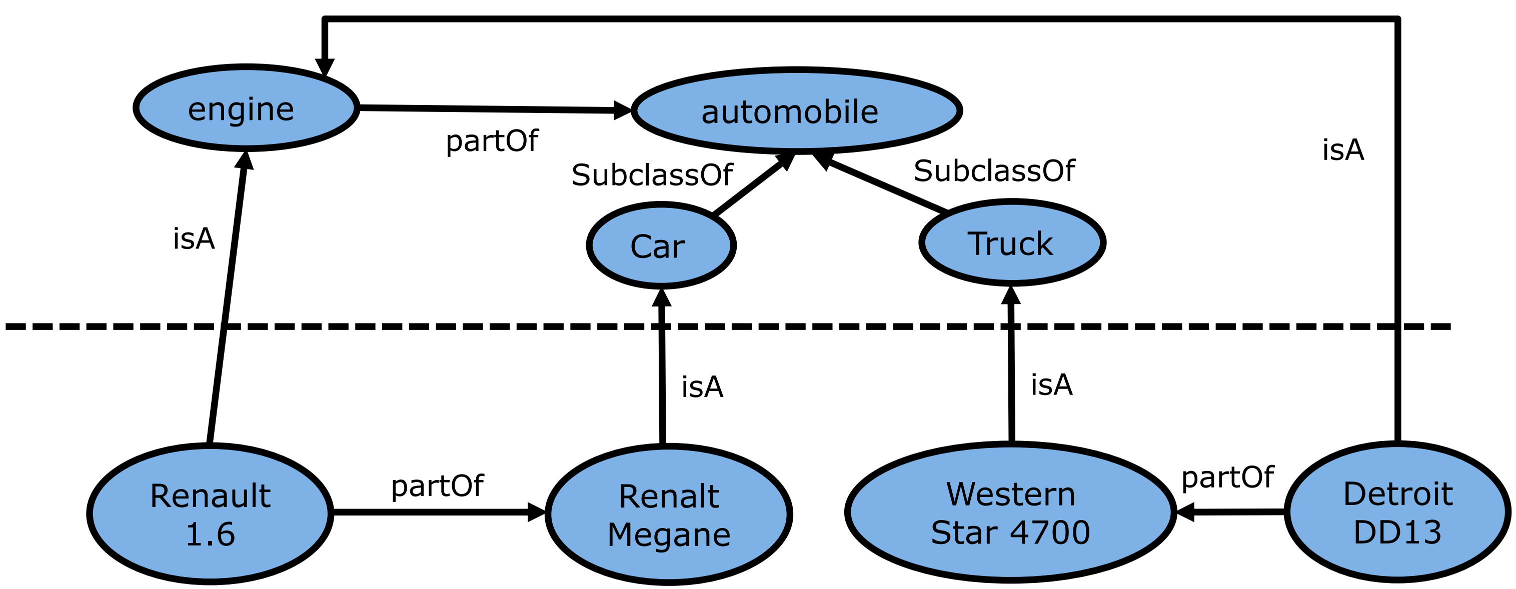
- The relational model is schema-oriented. Document-stores and key-values are schemaless databases but still rely on the concept of key
- Key-oriented models need to make a strong modeling call, which unbalances the logical / physical model
  - As consequence, the degree of (de)normalisation has a big impact in queries
- Can naturally benefit from sequencial reads
- Views are either virtual definitions or, if materialised, additional stand-alone constructs
- Poor relationship semantics: the relational model only deals with FK, document-stores / key-values do not support relationships
- Relational model, and most key-value / document-stores, follow a Closed-World assumption (i.e., complete data)

Graphs As Canonical Data Model (I)

- Expressiveness
  - Structural expressiveness
    - Relational data model: concepts / instances, referential integrity constraint
    - Graph data model: being a purely schemaless database, labels might embed any desirable semantics (e.g., subclass, aggregation, etc.)
  - Behavioural expressiveness
    - Relational model: checks, triggers, procedures an assertions (provided by the RDBMS)
    - Graph model: checks, triggers, procedures an assertions (provided by the GDBMS)

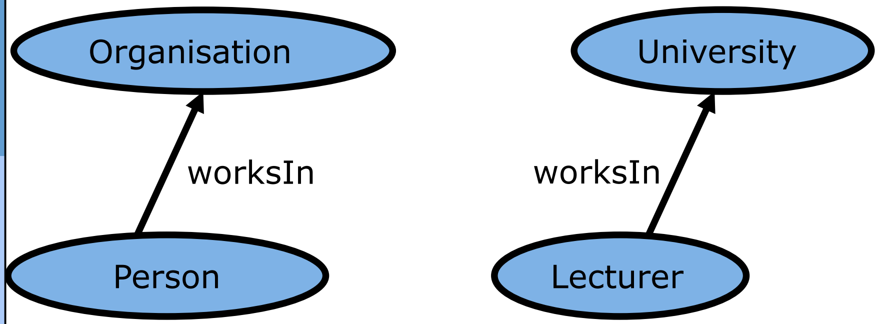
Graphs As Canonical Data Model (I)

- Expressiveness
  - Structural expressiveness
    - Relational data model: concepts / instances, referential integrity constraint
    - Graph data model: being a purely schemaless database, labels might embed any desirable semantics (e.g., subclass, aggregation, etc.)
  - Behavioural expressiveness
    - <u>Relational model</u>: checks, triggers, procedures an assertions (provided by the RDBMS)
    - <u>Graph model</u>: checks, triggers, procedures an assertions (provided by the GDBMS)



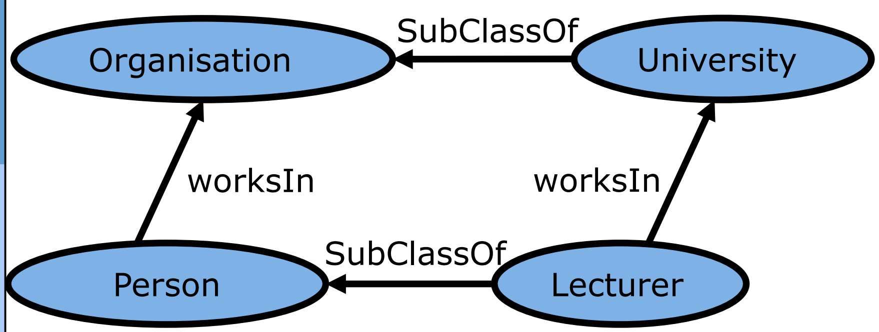
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



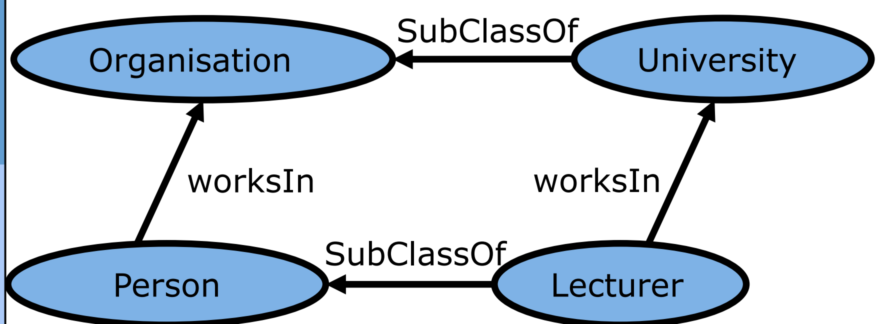
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)

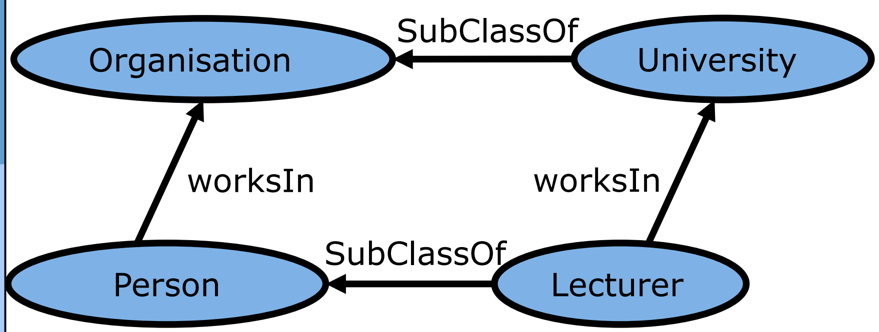


How to do this in relational?

Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)

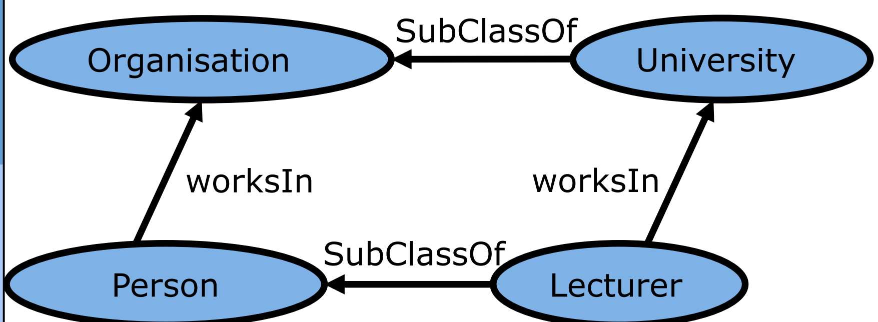
But graphs are even more flexible than that



How to do this in relational?

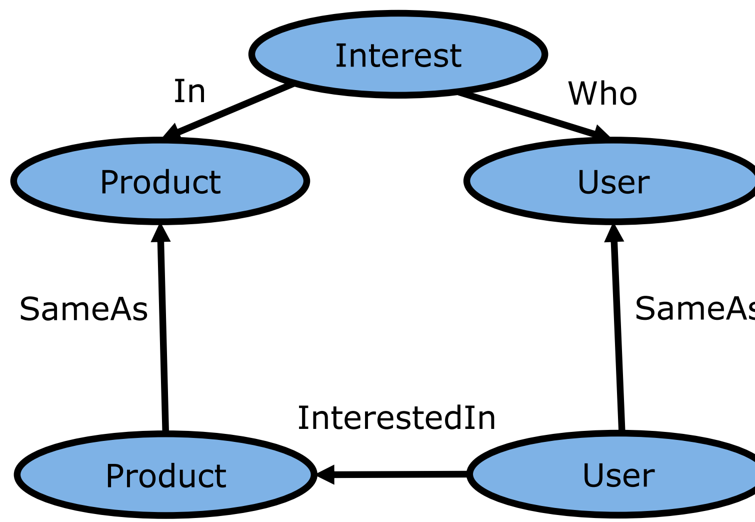
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



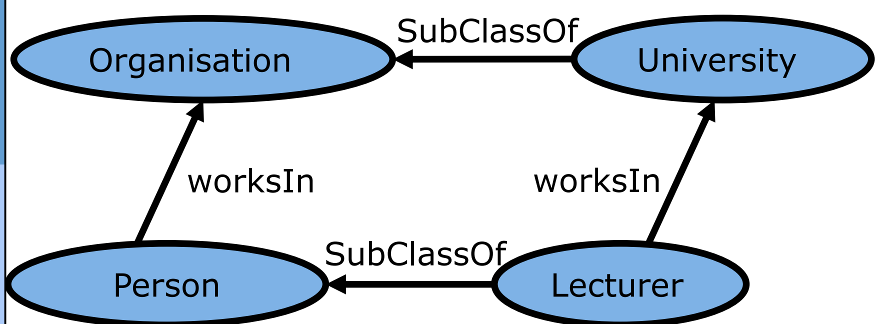
How to do this in relational?

But graphs are even more flexible than that



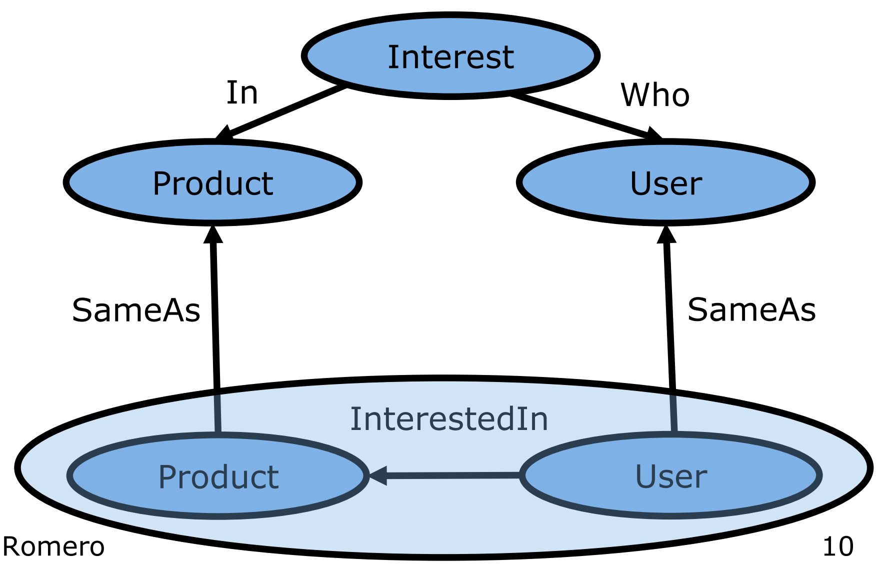
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



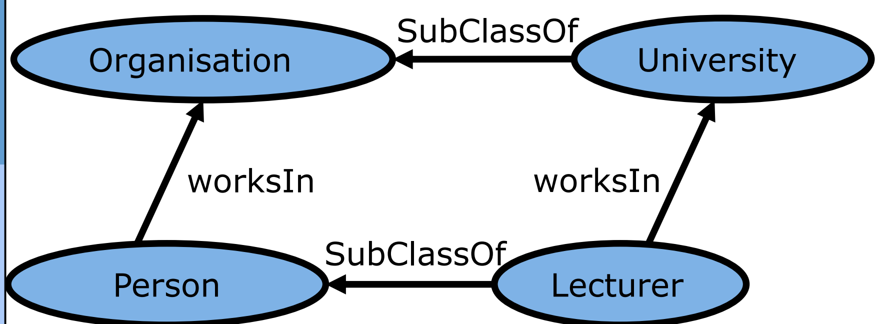
How to do this in relational?

But graphs are even more flexible than that



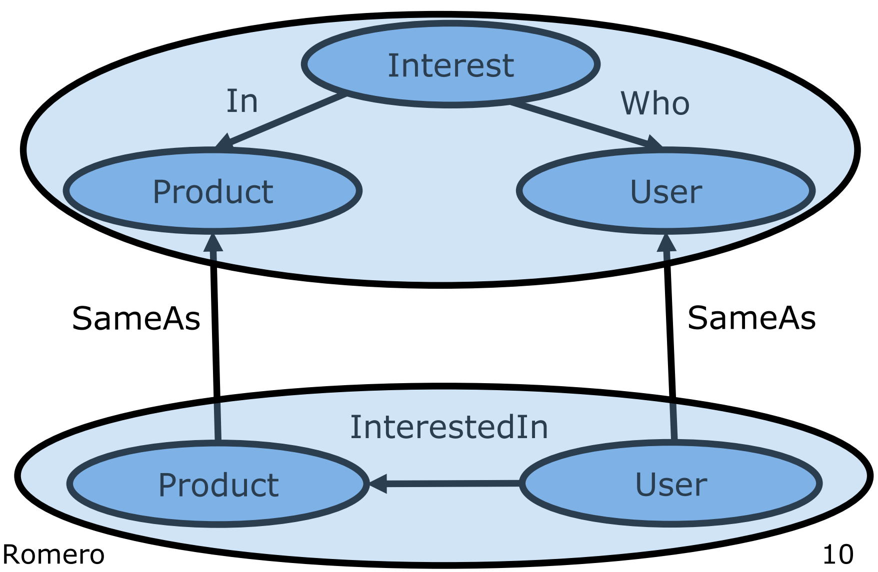
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - <u>Relational Model</u>: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - <u>Graph Model</u>: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



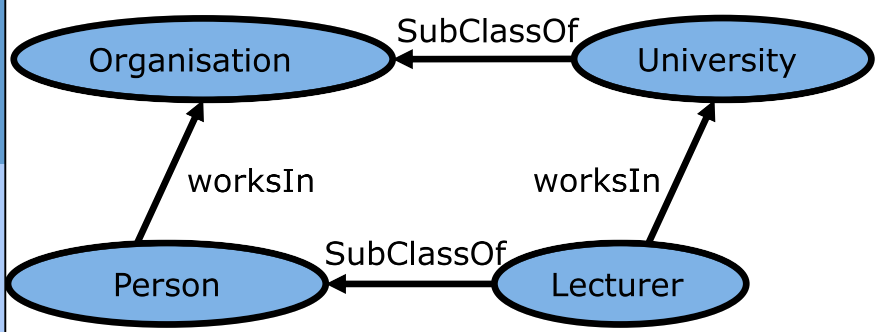
How to do this in relational?

But graphs are even more flexible than that



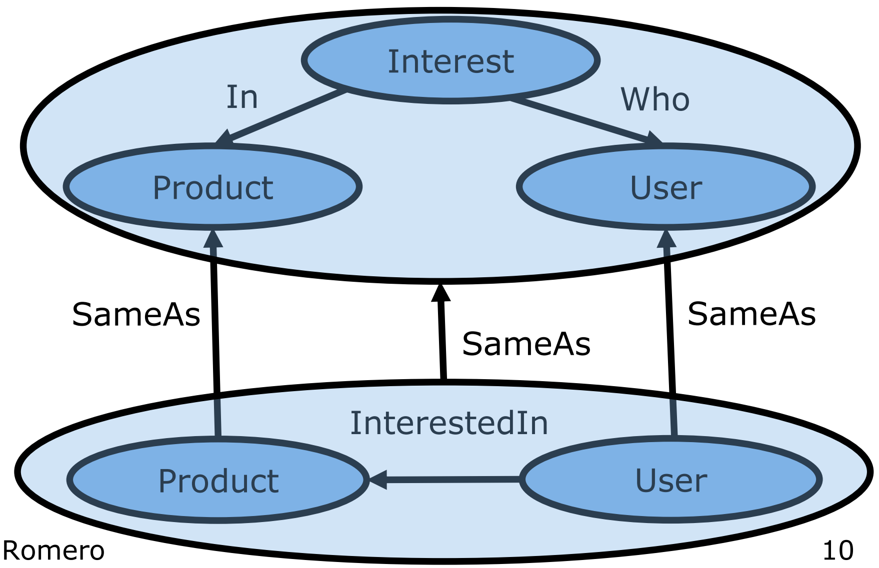
Graphs As Canonical Data Model (II)

- Semantic Relativeness
  - Relational Model: Monolithic
    - The model semantics are fixed. Table, columns and datatypes pre-defined at design time
    - Evolution not well-handled. Adding / deleting a column or changing a datatype may have a huge impact at the physical level
    - A powerful algebra available: the relational algebra
  - Graph Model: Flexible
    - New concepts / semantics can be added at any moment without drastically impacting the current data structures
    - Its flexibility allows to deal with evolution as first-class citizen
    - Powerful algebras available: For example, GraphQL is reducible to the relational algebra. In addition, it provides other relevant operations not naturally expressable on top of the relational model (e.g., pattern matching)



How to do this in relational?

But graphs are even more flexible than that



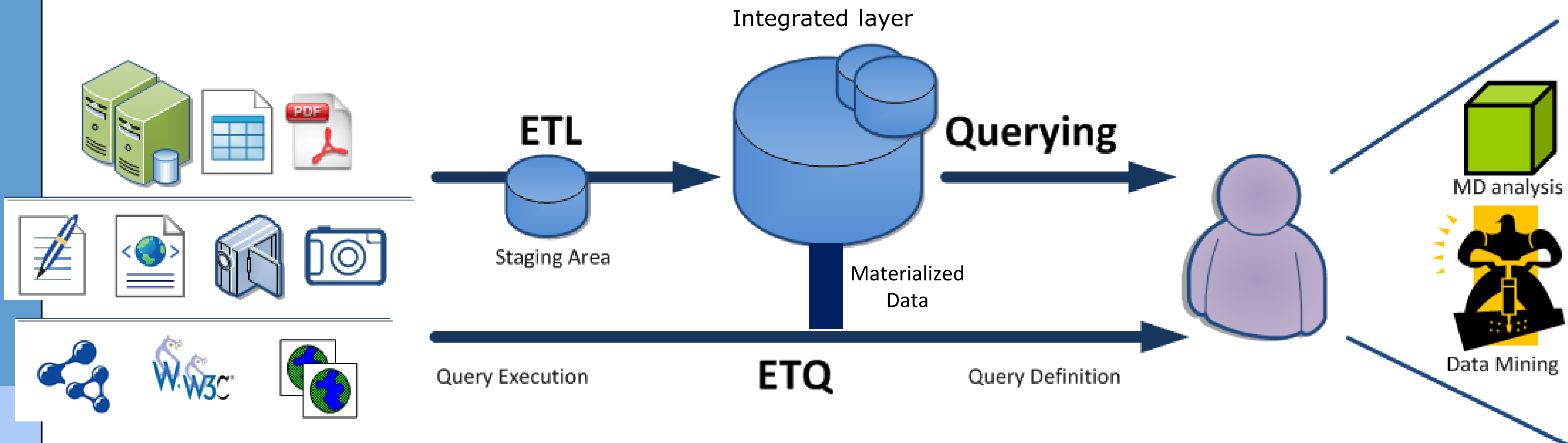
Properties of a Canonical Model

- Its semantic relativeness allow to represent any other data model
  - Highly expressive data structure
    - Nodes and edges enough to represent any modeling construct
      - Two basic structures
    - Allows to deal with semantic conflicts
      - Arbitrary semantics embedded in the edges
      - N-ary relationships can be represented by hypergraphs
  - Rich algebra
    - Mappable to the relational algebra plus topology-oriented operations to manipulate graph structures
- Arbitrary semantic annotations
  - Its structural and behavioural expressiveness allow a wide range of annotations
    - Distinguish classes / instances
    - Express rich relationships
    - Arbitrary constraints

GRAPHS FOR DATA INTEGRATION

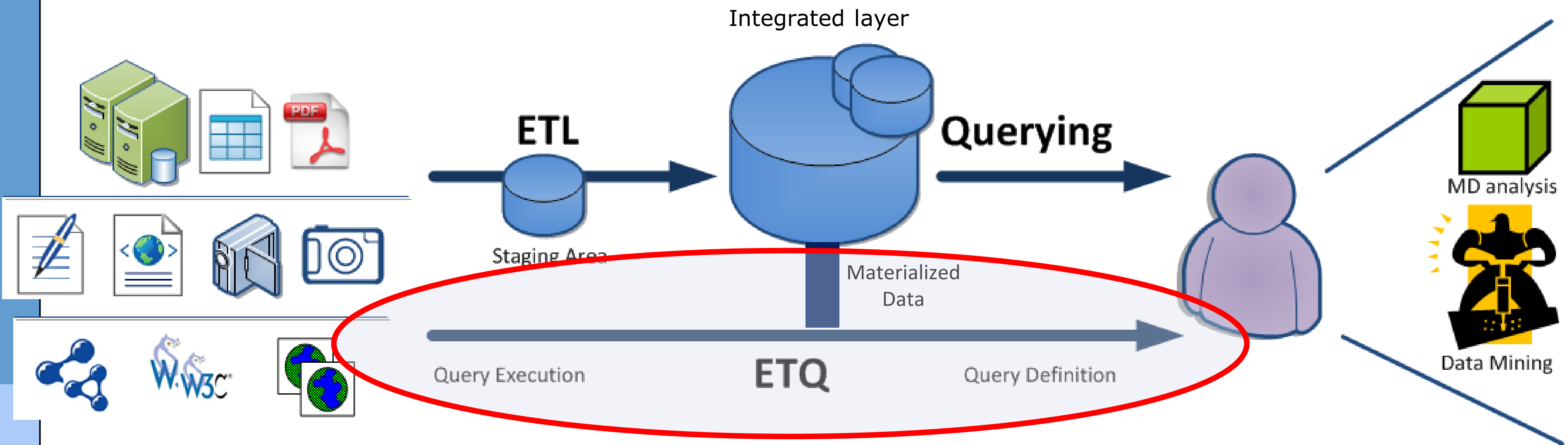
Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



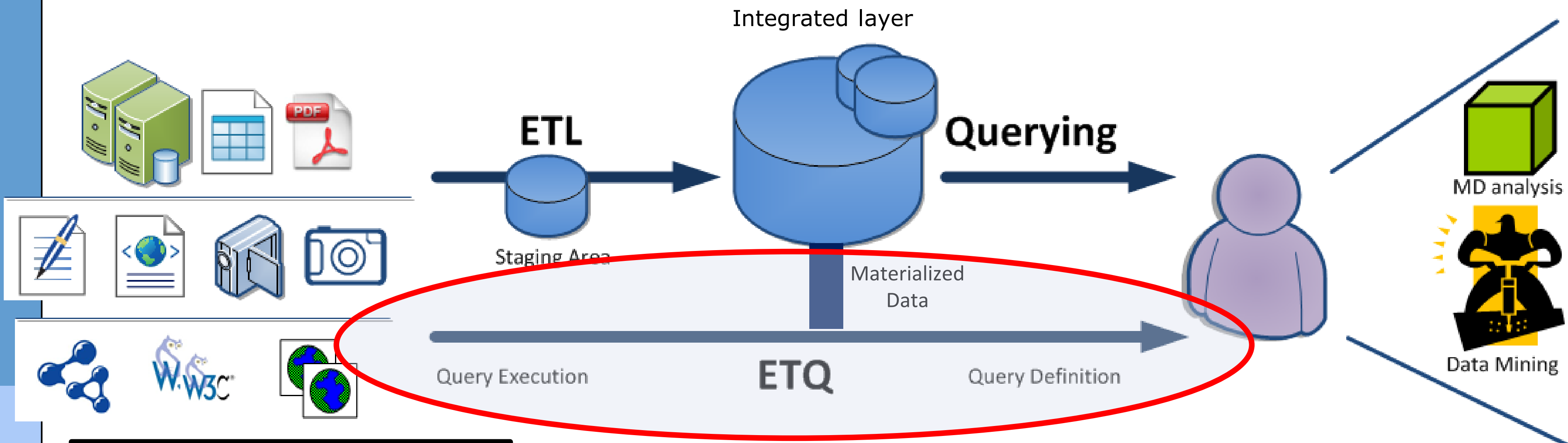
Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



Automating Data Integration

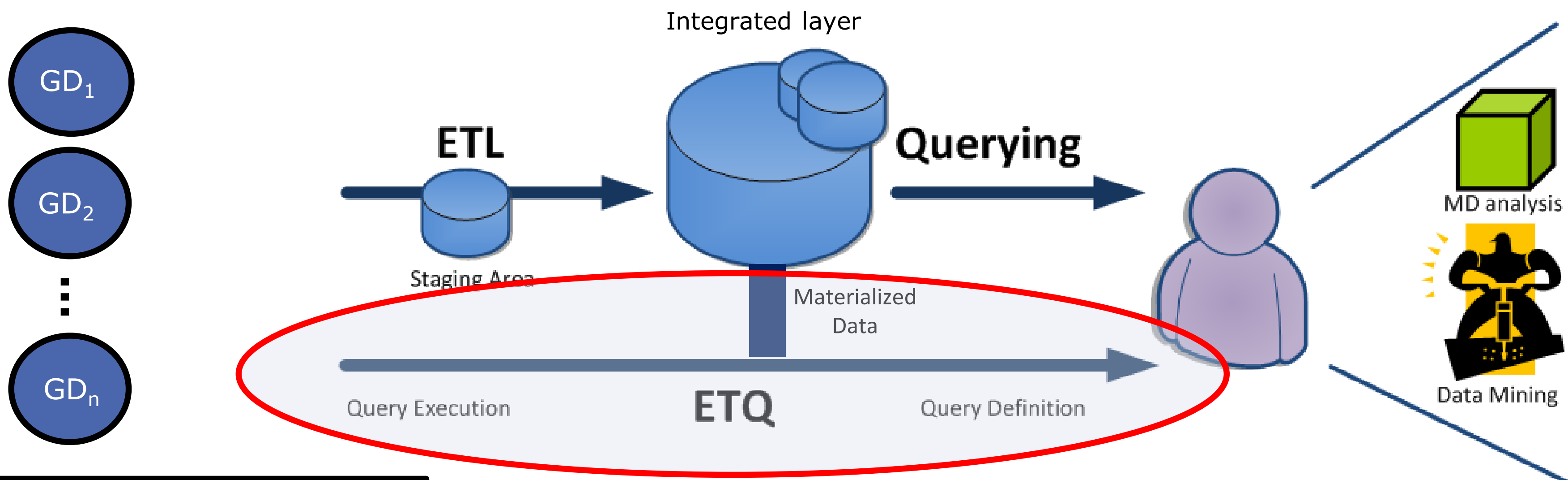
- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS_i)

Automating Data Integration

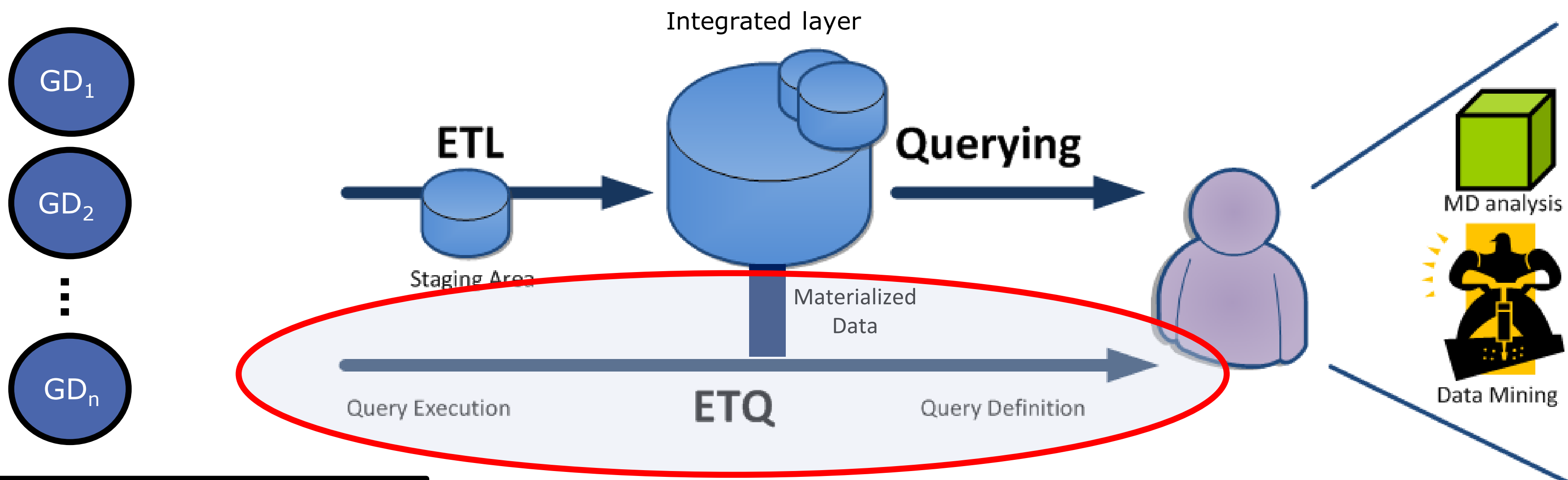
- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS$_i$)

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model

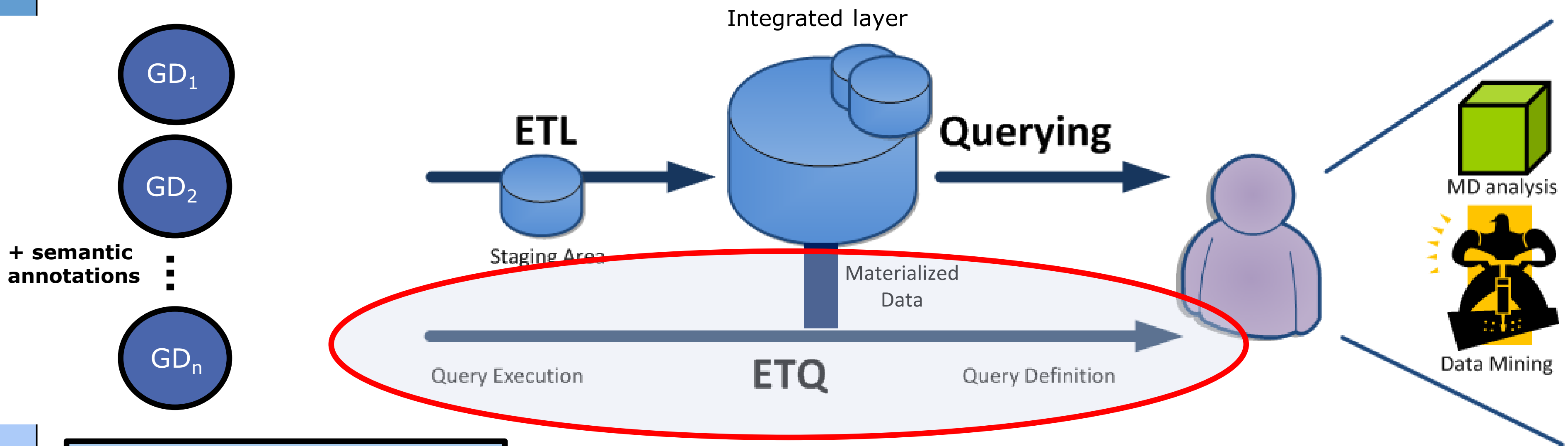


Represent each source as a graph (graph-based data source representation GDS$_i$)

Annotate the graph concepts with mappings

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model

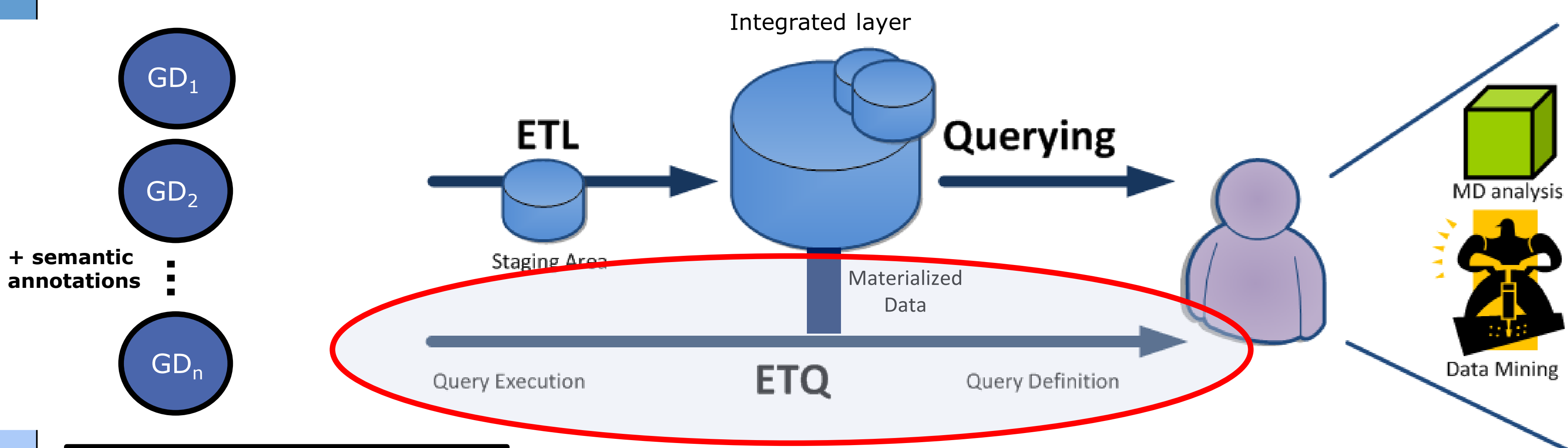


Represent each source as a graph (graph-based data source representation GDS_i)

Annotate the graph concepts with mappings

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



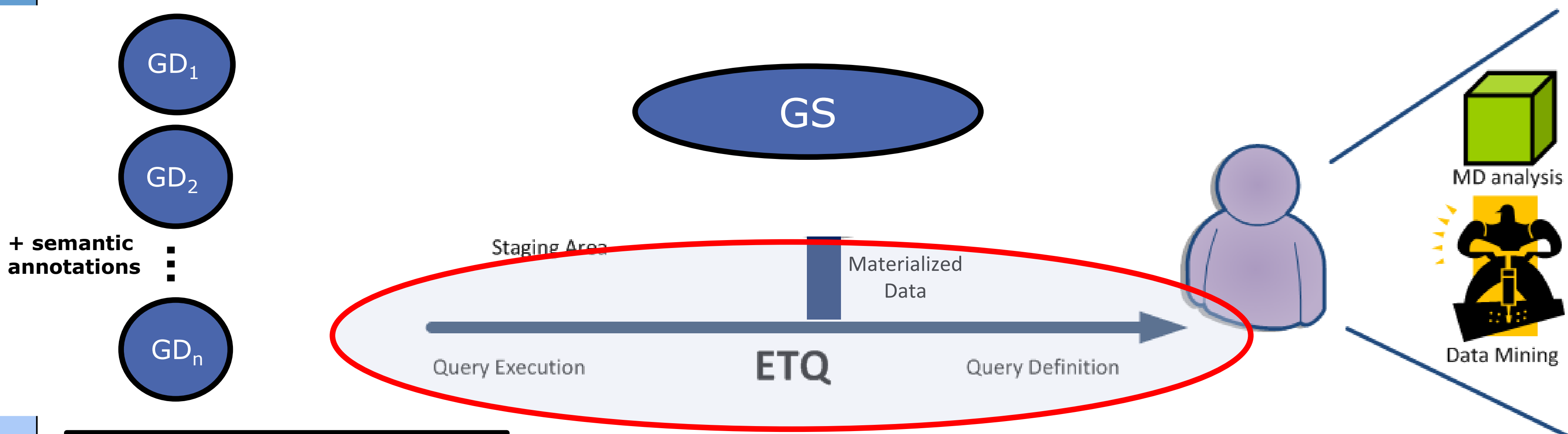
Represent each source as a graph (graph-based data source representation GDS_i)

Integrate all GDS_i into a single unified view (the Global Schema, GS)

Annotate the graph concepts with mappings

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



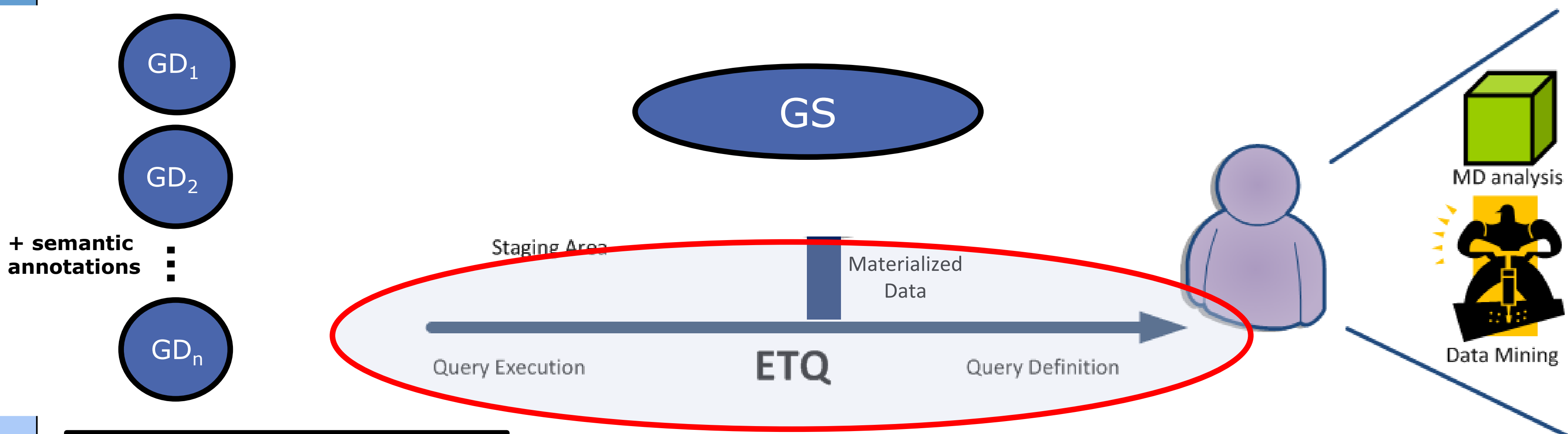
Represent each source as a graph (graph-based data source representation GDS_i)

Integrate all GDS_i into a single unified view (the Global Schema, GS)

Annotate the graph concepts with mappings

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS_i)

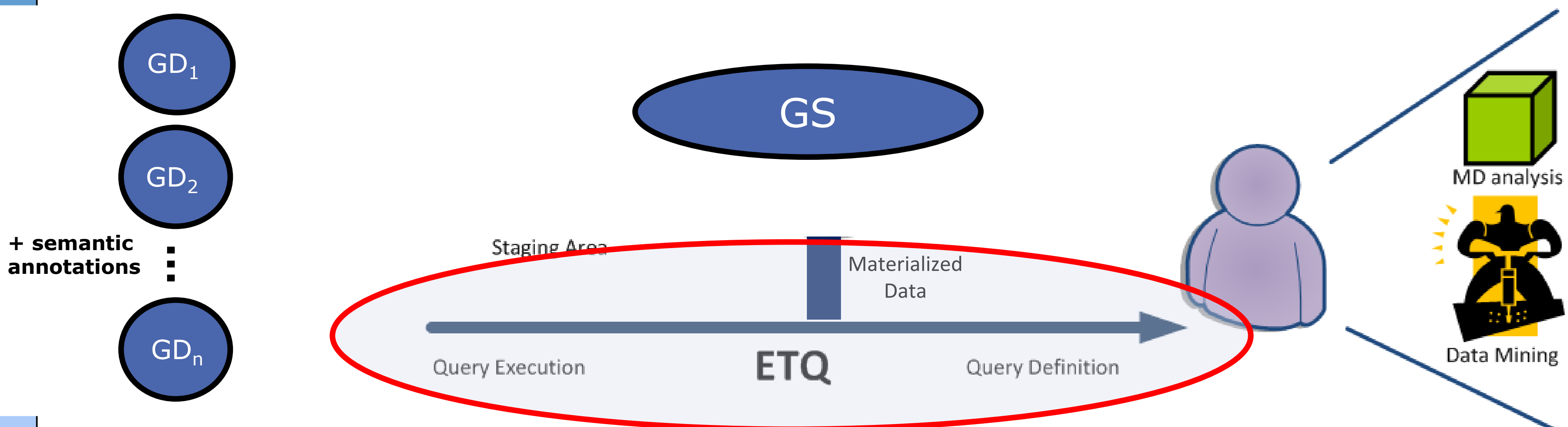
Annotate the graph concepts with mappings

Integrate all GDS_i into a single unified view (the Global Schema, GS)

For each GS concept generate a mapping by combining the existing mappings in GDS_i

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS_i)

Annotate the graph concepts with mappings

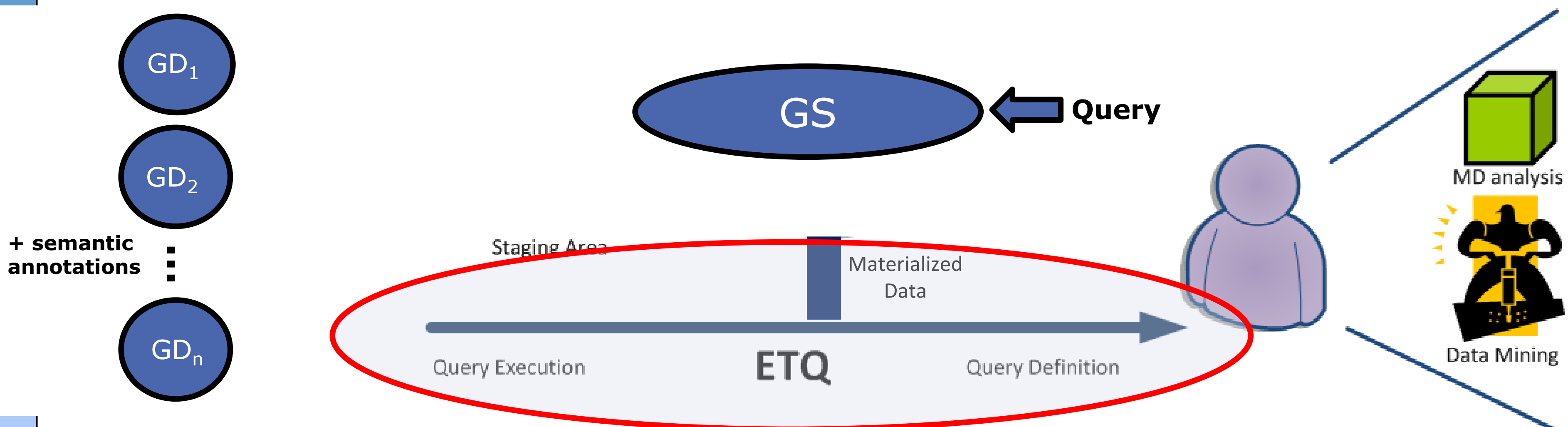
Integrate all GDS_i into a single unified view (the Global Schema, GS)

For each GS concept generate a mapping by combining the existing mappings in GDS_i

Devise a query rewriting algorithm that univocally rewrites queries in GS into an algebraic expression of queries over GDS_i

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS$_i$)

Annotate the graph concepts with mappings

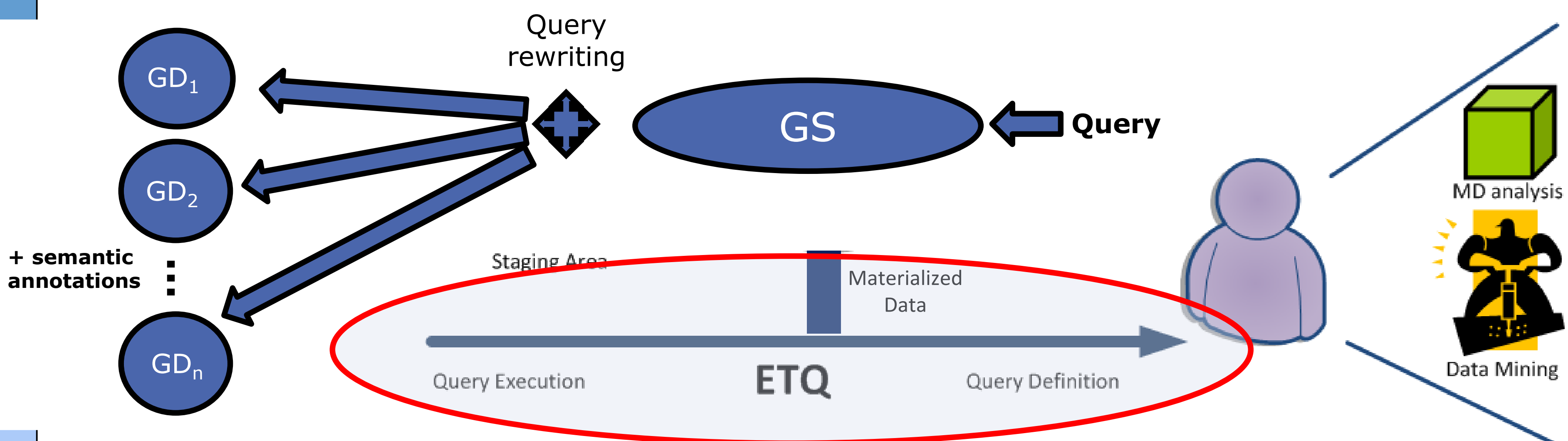
Integrate all GDS$_i$ into a single unified view (the Global Schema, GS)

For each GS concept generate a mapping by combining the existing mappings in GDS$_i$

Devise a query rewriting algorithm that univocally rewrites queries in GS into an algebraic expression of queries over GDS$_i$

Automating Data Integration

- A data-driven approach
  - Graphs as canonical data model



Represent each source as a graph (graph-based data source representation GDS$_i$)

Annotate the graph concepts with mappings

Integrate all GDS$_i$ into a single unified view (the Global Schema, GS)

For each GS concept generate a mapping by combining the existing mappings in GDS$_i$

Devise a query rewriting algorithm that univocally rewrites queries in GS into an algebraic expression of queries over GDS$_i$

Activity: Graph-Based Data Integration

- Objective: Understand graph-based data integration
- Tasks:
  1. (10') With a teammate think of the following:
     I. Assume graphs as canonical data model
     II. First, model as graphs each source (separately):

Source 1	Source 2	Source 3
- User - Tweet - Date - Location	- Product - Product features - User	- Customer - Product - Landing time - rating

     III. Now, think what elements from each graph can be related during the integration process
        I. Look for similar or identical concepts
        II. Think of the semantic relationship you would use
     IV. How would the global schema look like?

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle
  - Schemata,
  - Mappings,

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle
  - Schemata,
  - Mappings,
  - Queries (logs, sessions, etc.),

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle
  - Schemata,
  - Mappings,
  - Queries (logs, sessions, etc.),
  - User profiling,
    - Preferences
    - Characteristics

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle
  - Schemata,
  - Mappings,
  - Queries (logs, sessions, etc.),
  - User profiling,
    - Preferences
    - Characteristics
  - Data profiling,

Metadata

- Graphs allow arbitrary semantic annotations
  - For nodes or edges
  - For subgraphs or the whole graph
- **Data and metadata are stored together**
- Semantic annotations are used to express metadata artefacts needed to automate the whole data integration life-cycle
  - Schemata,
  - Mappings,
  - Queries (logs, sessions, etc.),
  - User profiling,
    - Preferences
    - Characteristics
  - Data profiling,
  - Traceability,
  - …

GRAPH DATA MODELS

Graph Data Models

- There is not a single graph data model
- Two main families of graphs
  - **<u>Property Graphs</u>**
    - Born in the database field
    - Not predefined semantics
    - Follow a Closed-World assumption
    - Generate data silos
    - Algebraic operations based on graph structures
  - **<u>Knowledge Graphs</u>**
    - Born in the knowledge representation field
    - Assume the Open-World assumption
    - Facilitate data sharing and linking
    - Two main families
      - RDF and RDF(S)
        - Born in the semantic web field
        - Vocabulary-based pre-defined semantics
        - Combine algebraic operations with simple reasoning operations
      - Description Logics (DL)-based
        - Representation of (subsets of) first-order logic
        - Pre-defined semantics based on logics
        - Reasoning operations founded in their logics nature

Summary

- Graphs are the perfect canonical data model given their:
  - Semantic expressiveness,
  - Semantic relativeness
- As result, data and metadata (semantic annotations on data) are stored together
  - Machine-readable metadata opens the door to automatic transformations
  - Covering the right metadata artefacts, graphs help to automate the whole data integration lifecycle
- Main graph families
  - Property graphs
  - Knowledge graphs