# Data(base) Integration
## Multi-Database Systems (MDBS)

Vera Goebel
Department of Informatics, University of Oslo

# Interoperability Problem

- Interoperability at the application level
  Data Integration from various data sources including non-database sources ->
  Data Stream Processing Systems, Data Lakes
- Interoperability at the database level
  Bottom-up design process ->
  Multi-Database Systems, Data Warehouses

- Data exchange → Web Data Management

# Integration Alternatives

(complementary) for different applications/usage

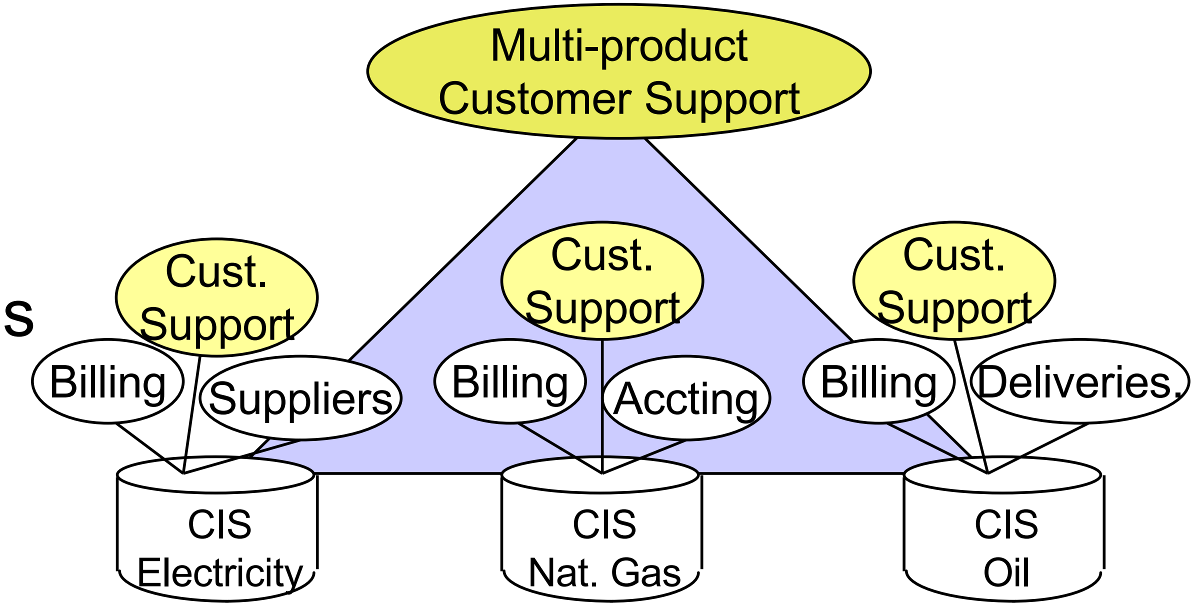
- Physical integration (OLAP) ->
  Data Warehouses, Data Lakes
  - Source databases integrated and integrated database is materialized (ETL - extract-transform-load)
  - Multi-dimensional DB for complex data analysis (ML)
- Logical integration (OLTP) ->
  Multi-DBS (MDBS)
  - Global conceptual schema is virtual and not materialized
  - Data control and availability, multi-user, high throughput

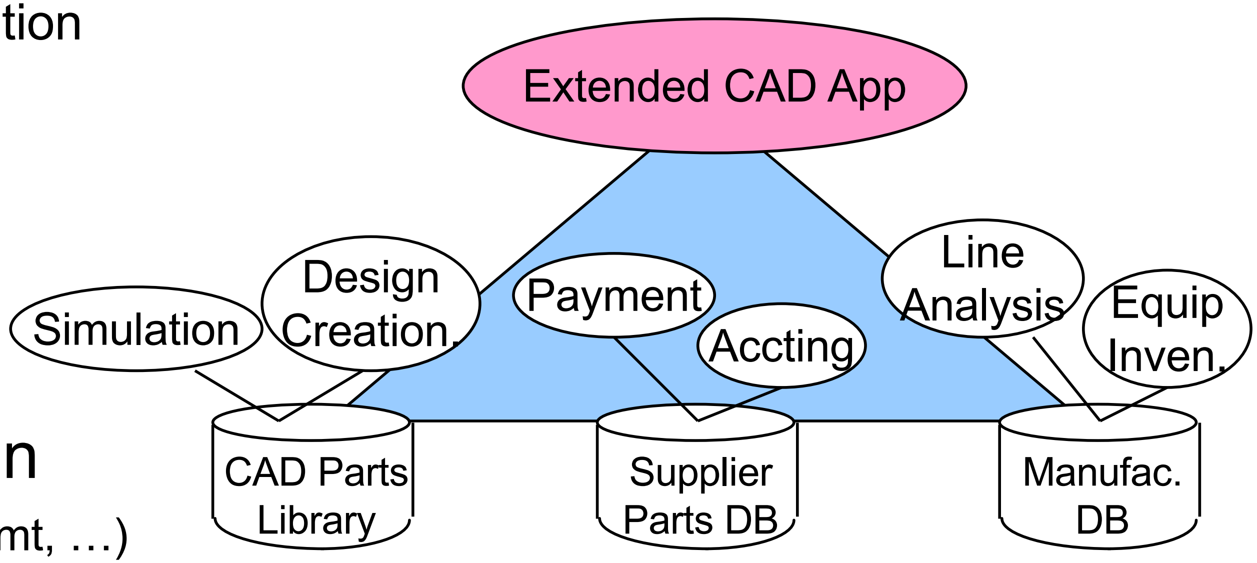
# Heterogeneous / Federated / Multi-Database Systems (MDBS)

- Why Heterogeneous MDBS?
- Applications
- Architectures for MDBS
- Main Problems:
  - Global Data Model
  - Query Processing
  - Query Optimization?
  - Transaction Management?

# Applications

- Multitude of extensive, isolated data agglomerations managed by different DBS
  - Similar data
    - Ex: 3 Customer Info Systems
  - Dissimiliar data
    - Ex: Extended CAD Application
- Extension of data and management software because of new and/or extended applications
- Heterogeneous application domains (e.g., CIM, CAD, Biz-mgmt, …)



# Requirements for MDBS

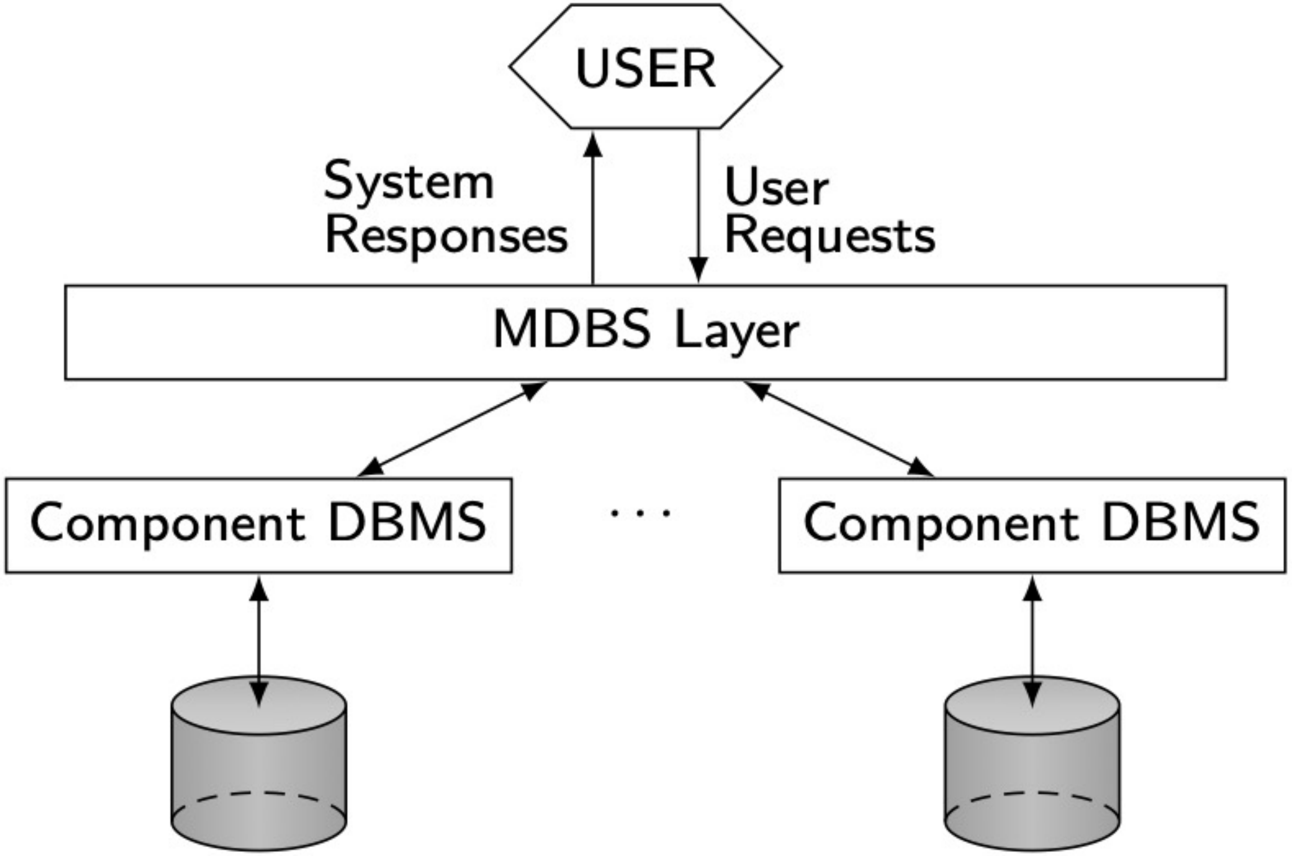
**Integration of Heterogeneous DBSs**

-> queries across MDBS (combine heterogeneous data)
-> heterogeneous information structures
-> avoid redundancy
-> access (query) language transparency

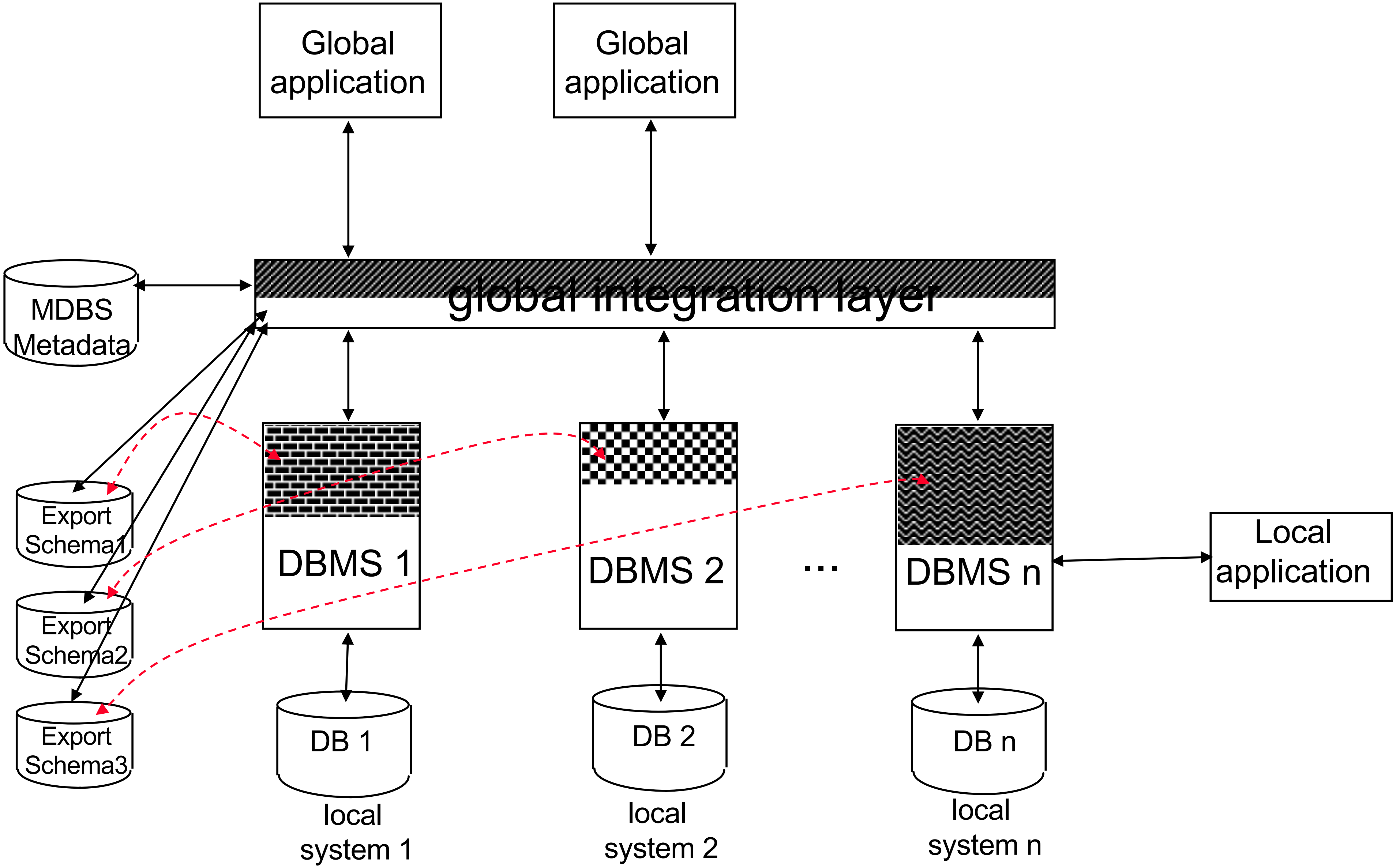
- **"Open" system**
  support for integration of existing data models and DBSs, as well as their schemata and databases

- **Constraints**
  -> retain autonomy of DBSs to be integrated
  -> avoid modifications of existing local applications
  -> define a viable global data model for global applications

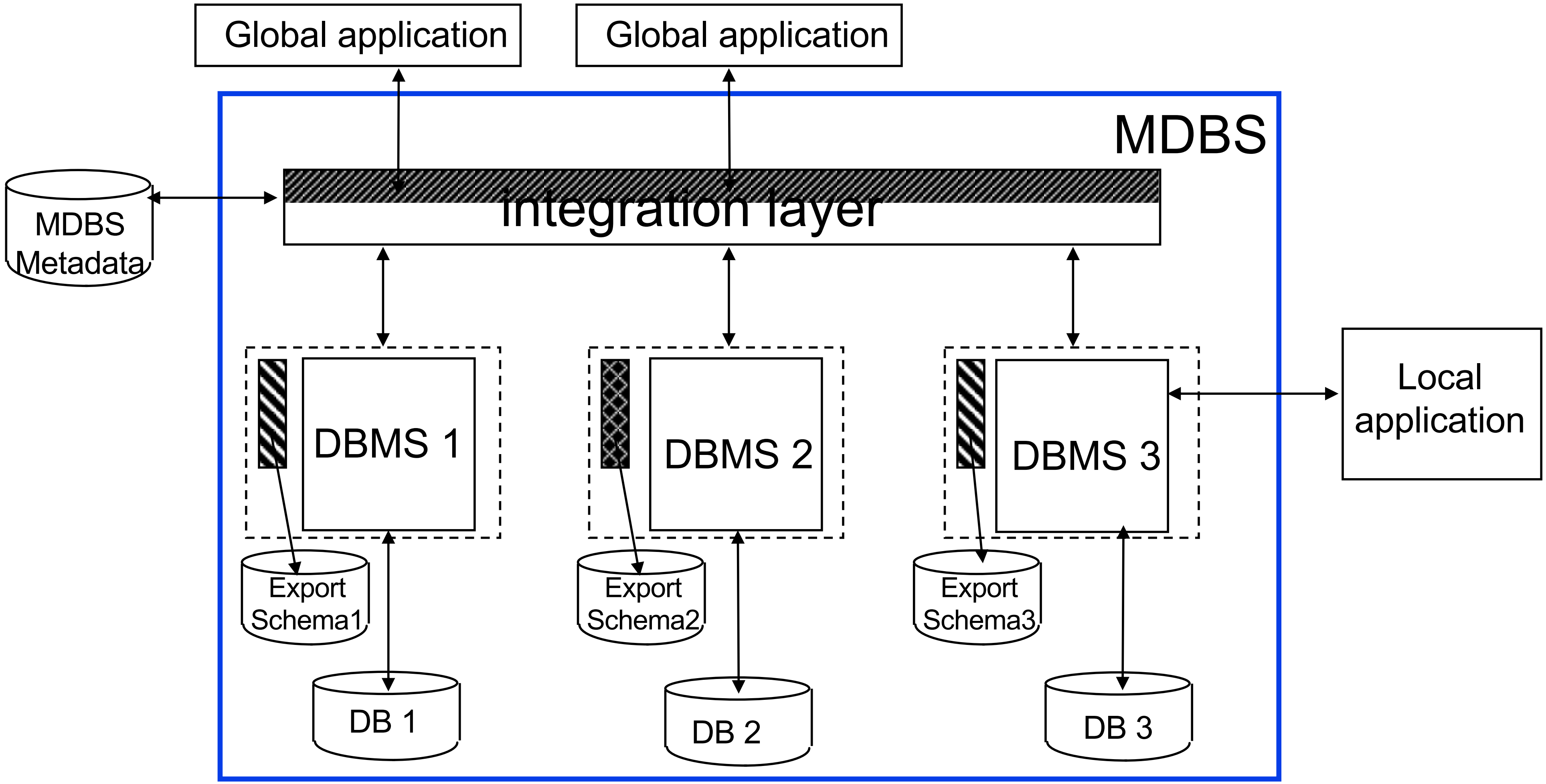
# Database Integration – Multi-DBS Architecture



# MDBS (federation, partially autonomous)



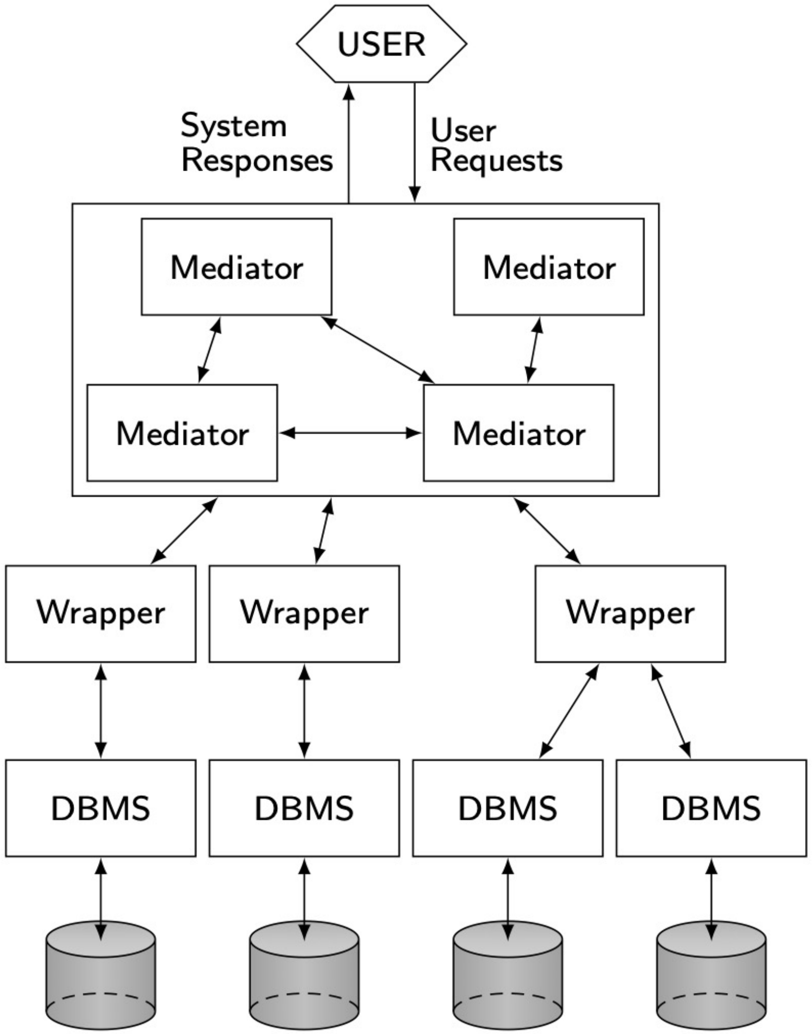
# MDBS (fully autonomous)




MDBS Server or MDBS Proxy

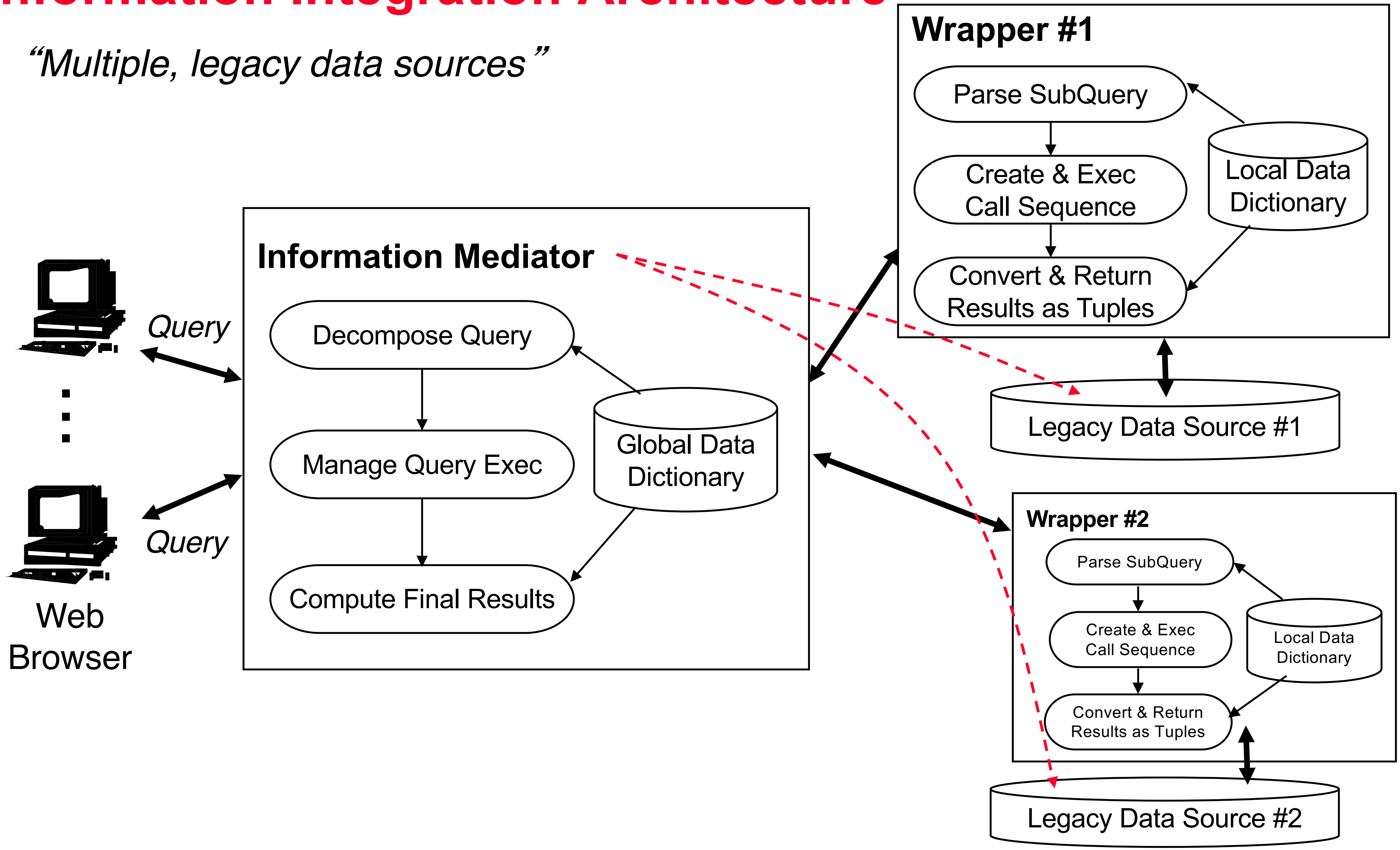
- Runs on the local DB site
- Typically includes some code that is specific to the local DB type

# Mediator/Wrapper Architecture



# Information Integration Architecture

*"Multiple, legacy data sources"*



# Integration Layer

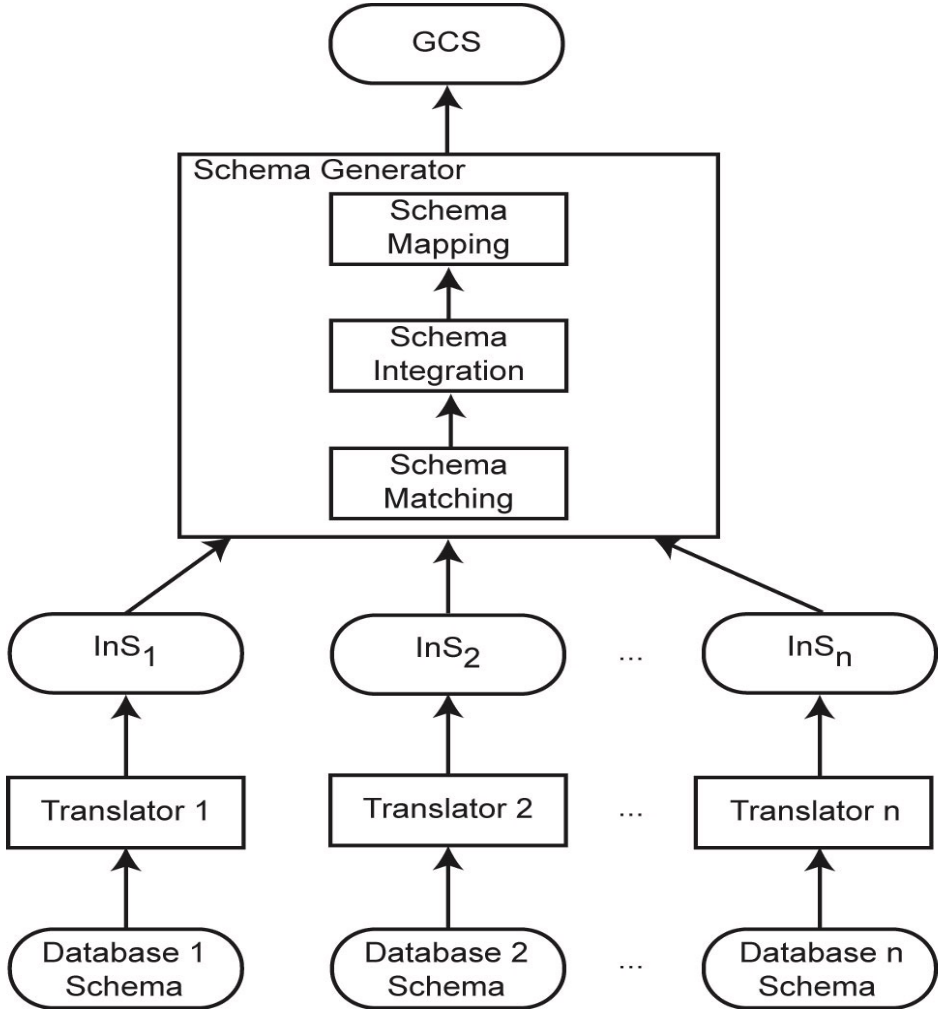
- **Global data model**:
  - Local data models: any kind of data model possible, e.g., object-oriented, relational, entity-relationship, hierarchical, network-oriented, ...
  - Global data model: must comprise modeling concepts and mechanisms to express the features of the local data models
- Global schema and metadata management
- Distributed query processing and optimization

- Primarily read-only queries/transactions at the global level
  - Autonomous local DBS can run all kinds of queries/transactions
    -> unknown at the global integration layer/level

# Database Integration Process

- Schema translation: Component database schemas translated to a common intermediate canonical representation
- Schema generation: Intermediate schemas are used to create a global conceptual schema
  - Schema Matching
  - Schema Integration
  - Schema Mapping
- Query rewriting and processing
- Query optimization

# Database Integration Process



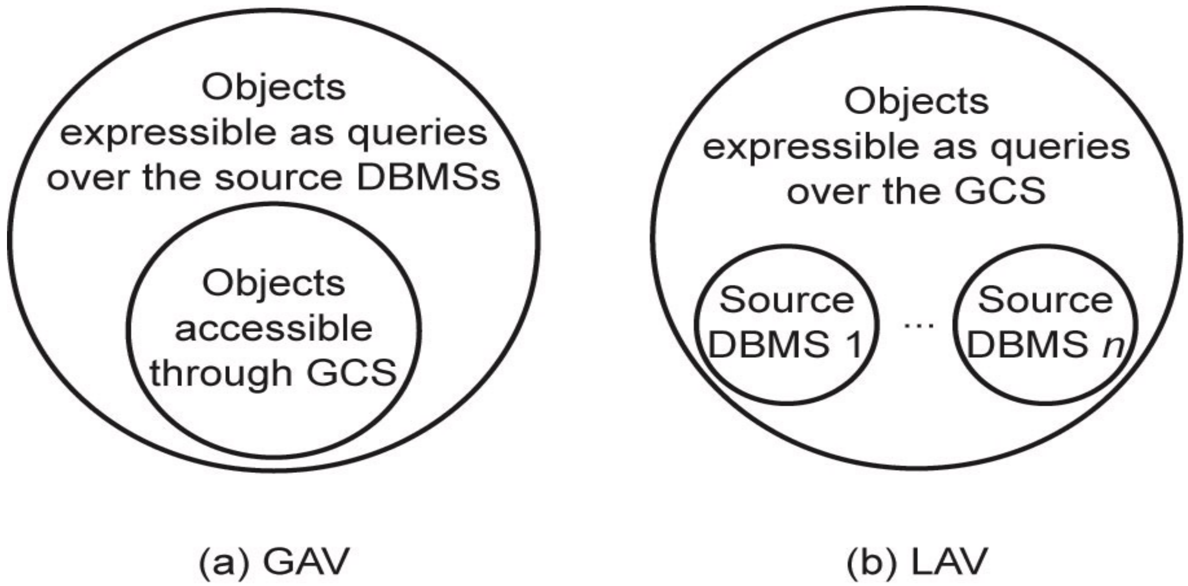
# Bottom-up Design

- Problem: Given existing databases with their Local Conceptual Schemas (LCSs), how to integrate the LCSs into a Global Conceptual Schema (GCS)

- GCS (mediated schema) is defined first
  - Map LCSs to this schema
  - As in data warehouses
- GCS is defined as an integration of parts of LCSs
  - Generate GCS and map LCSs to this GCS

# GCS/LCS Relationship

- Local-as-view
  – The GCS definition is assumed to exist, and each LCS is treated as a view definition over it
- Global-as-view
  – The GCS is defined as a set of views over the LCSs



# Schema Generation

- Schema matching
  - Finding the correspondences between multiple schemas
- Schema integration
  - Creation of the GCS (or mediated schema) using the correspondences
- Schema mapping
  - How to map data from local databases to the GCS
- Important: sometimes the GCS is defined first and schema matching and schema mapping is done against this target GCS

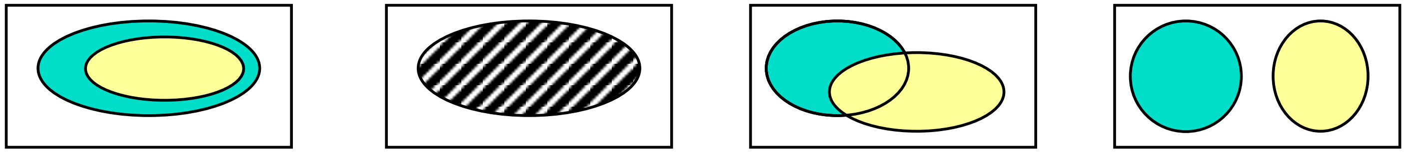
# Schema Homogenization

- Schema Translation
  - Map each local schema to the language of the global data model
    - Ex: a Relational schema to an Object-oriented schema

- Schema Integration
  - For *N* translated, local schemas
    - Pairwise integration, X-at-a-time integration, One-step integration

Adequate design tools are not available

  - Determine ”common semantics” of the schemas



  - Make the ”same things” be ”one thing” in the integrated schema
  - Resolve conflicts
    - structural and semantic

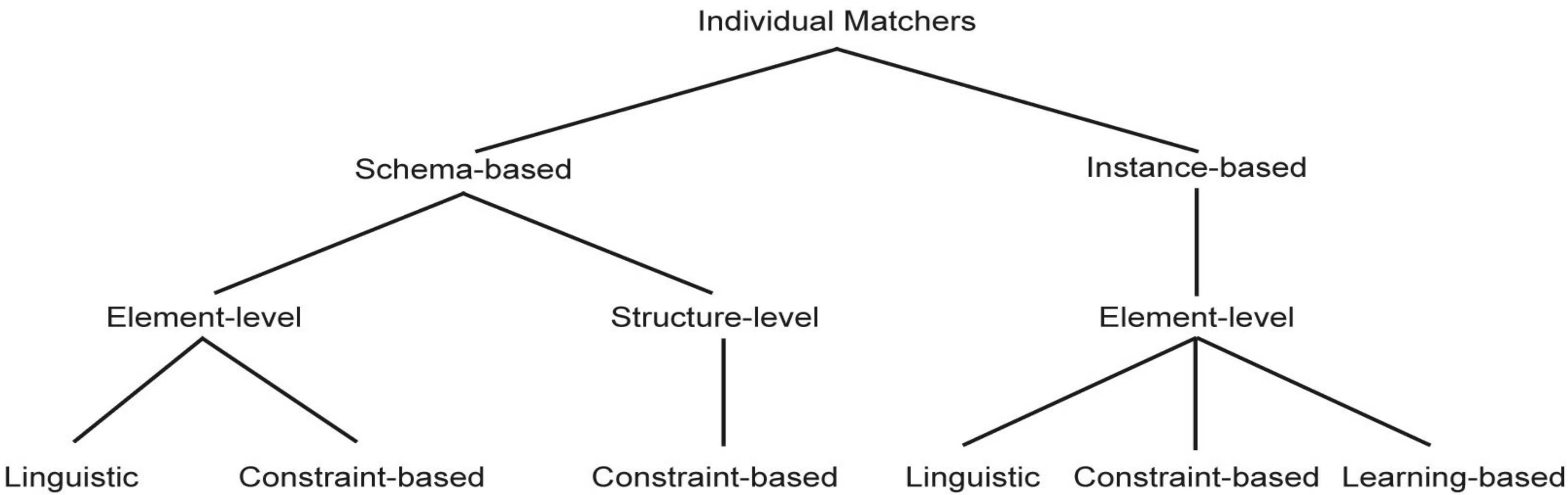
# Schema Matching

- Schema heterogeneity
  - Structural heterogeneity
    - Type conflicts
    - Dependency conflicts
    - Key conflicts
    - Behavioral conflicts
  - Semantic heterogeneity
    - More important and harder to deal with
    - Synonyms, homonyms, hypernyms
    - Different ontology
    - Imprecise wording

# Schema Matching (cont.)

- Other complications
  - Insufficient schema and instance information
  - Unavailability of schema documentation
  - Subjectivity of matching
- Issues that affect schema matching
  - Schema versus instance matching
  - Element versus structure level matching
  - Matching cardinality

# Schema Matching Approaches



# Schema Conflicts

- Name
  - Different names for equivalent entities, attributes, relationships, etc.
  - Same name for different entities, attributes, …
- Structure
  - Missing attributes
  - Missing but implicit attributes
- Relationship
  - One-to-many, many-to-many
- Entity versus Attribute (inclusion)
  - One attribute or several attributes
- Behavior
  - Different integrity constraints

# Data Representation Conflicts

- Different representation for equivalent data
  - Different units
    - Celsius ↔ Farenheit;  Kilograms ↔ Pounds; Liters ↔ Gallons;
  - Different levels of precision
    - 4 decimal digits versus 2 decimal digits
    - Floating point versus integer
  - Different expression denoting same information
    - Enumerated Value sets that are not one-to-one
      - {good, ok, bad} versus {one, two, three, four, five}

# Conflict Resolution

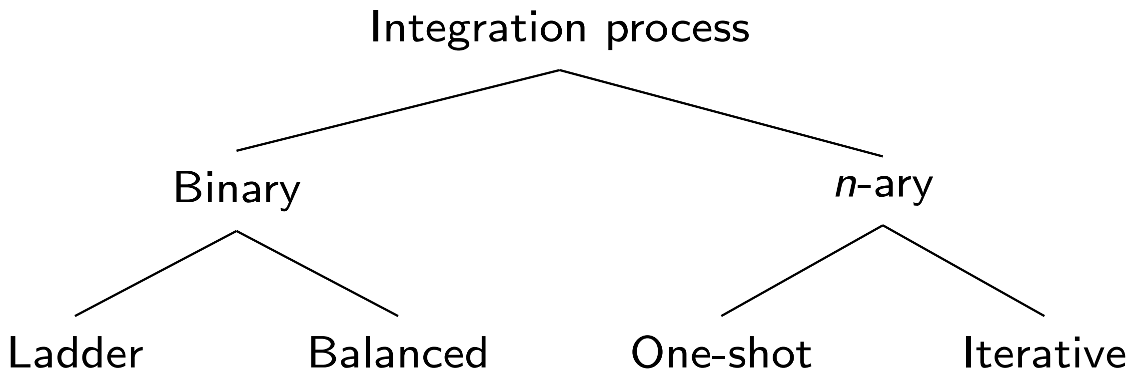
- Renaming entities and attributes
  - Pick one name for the same things
  - Use unique prefixes for different things
- Homogenizing representations
  - Use conversions and mappings
    - stored programs in relational systems
    - methods in OO systems
    - auxiliary schemas to store conversion rules/code
- Homogenizing attributes
  - Use type coercion (e.g., integer to float)
  - Attribute concatenation (e.g., first name || last name)
  - For missing attributes, assign default values
- Homogenizing an attribute and an entity
  - Extract an attribute from the entity
  - Create an entity from the attribute

# Conflict Resolution

- <u>Horizontal joins</u>
  - Union compatible
    - For missing attributes, assign default values or compute implicit values
  - Extended union compatible
    - Use generalization
      - Define a virtual class containing common attributes
    - Subclasses of the generalization
      - Provide specialized values and compute attribute values for generalized attributes
    - See earlier example
      - class Person generalizes class Student and class Employee
- <u>Vertical joins</u>
  - Many and many to one
- <u>Mixed Joins</u>
  - Vertical and horizontal joins in combination

# Schema Integration

- Use the correspondences to create a GCS
- Mainly a manual process, rules can help



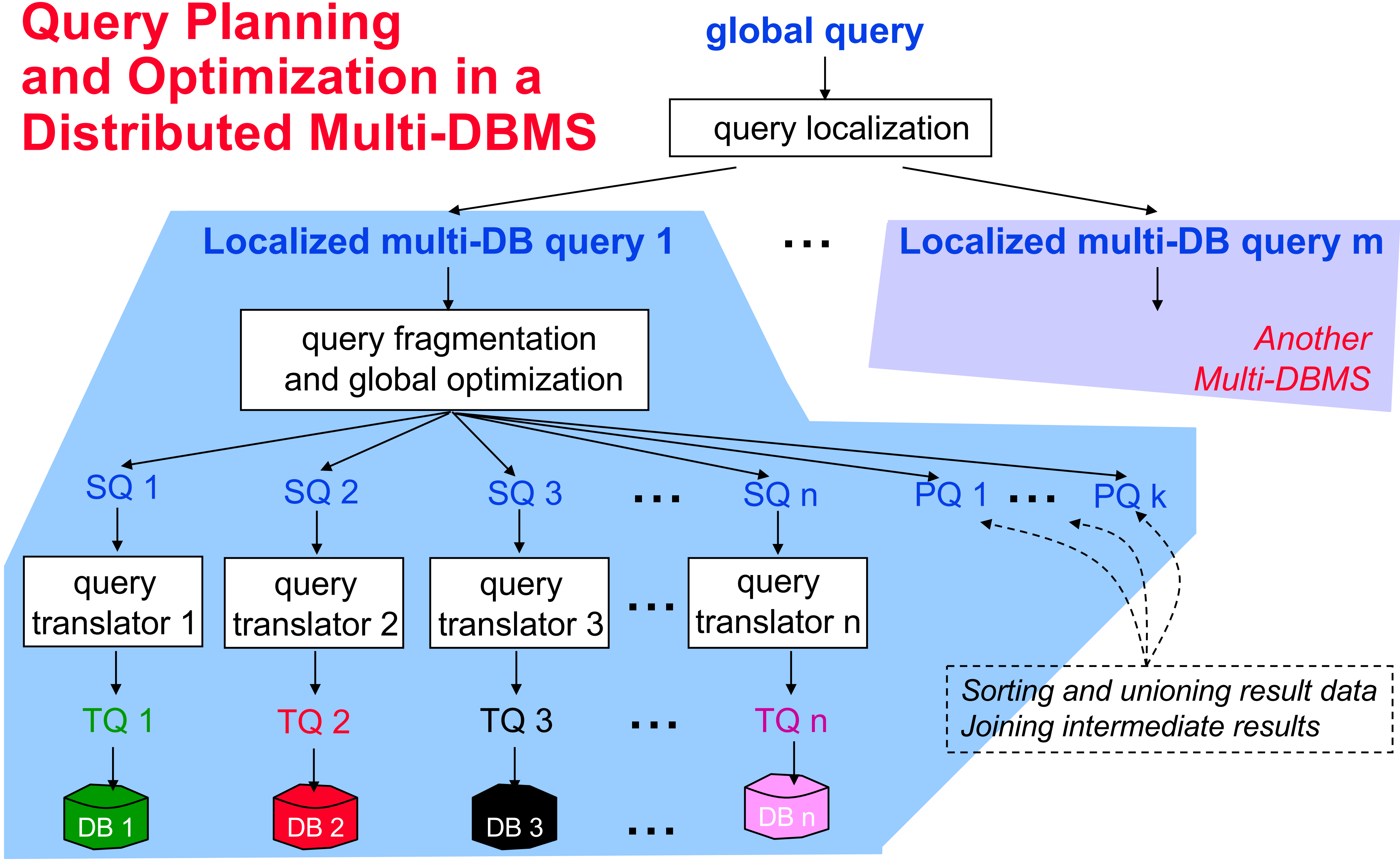
# Schema Mapping

- Mapping data from each local database (source) to GCS (target) while preserving semantic consistency as defined in both source and target.
- Data warehouses ⇒ actual translation
- Data integration systems ⇒ discover mappings that can be used in the query processing phase
- Mapping creation
- Mapping maintenance

# Global Schema Management

- Global schema = sum of all local exported schemata
- Global schema definition facilities provide mechanisms for handling the full spectrum of schematic differences that may exist among the heterogeneous local schemata.
- Data is stored in the local component DBS.
- Global dictionary information is used to query and manipulate the data. The
- Global queries are translated into equivalent queries of the local languages supported by the local DBS

# Query Planning and Optimization in a Distributed Multi-DBMS



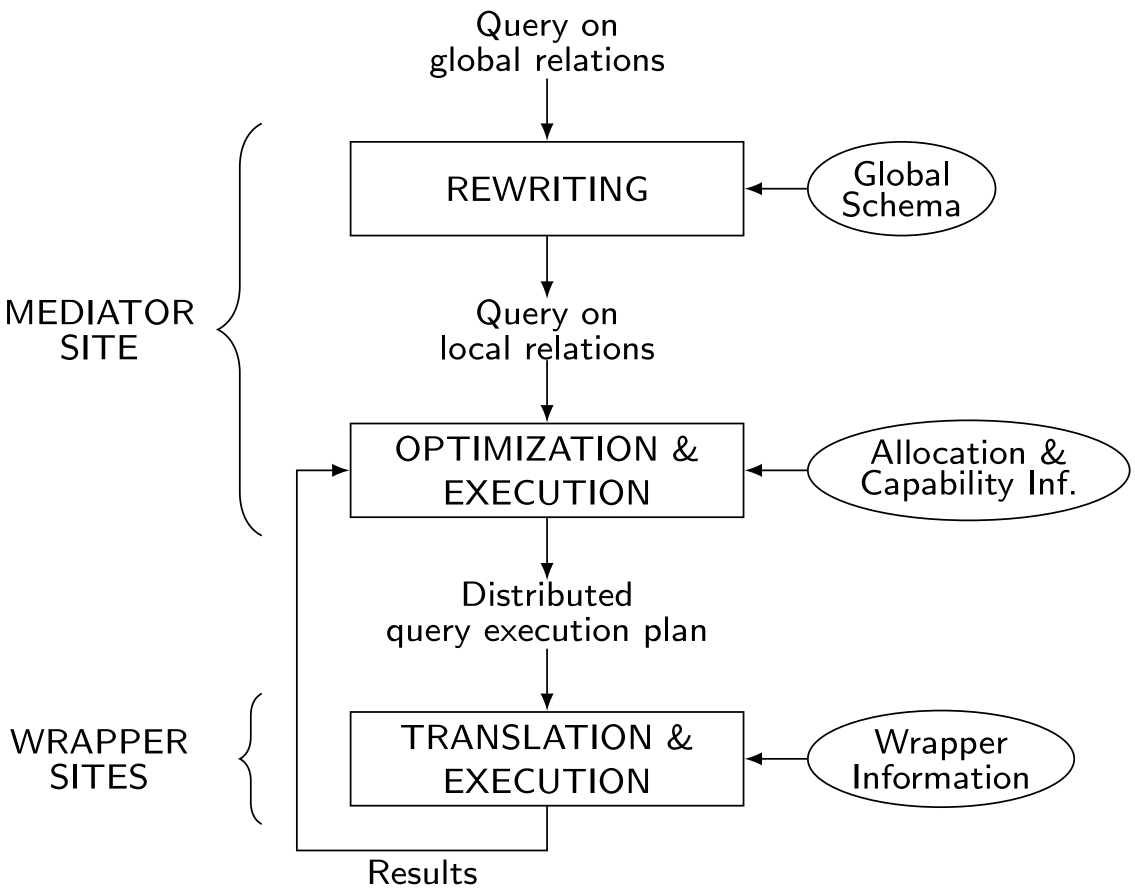
# Issues in MDBS Query Processing

- Component DBMSs are autonomous and may range from full-fledge relational DBMS to flat file systems
  - Different computing capabilities
    - Prevents uniform treatment of queries across DBMSs
  - Different processing cost and optimization capabilities
    - Makes cost modeling difficult
  - Different data models and query languages
    - Makes query translation and result integration difficult
  - Different runtime performance and unpredictable behavior
    - Makes query execution difficult

# Component DBMS Autonomy

- Communication autonomy
  - The ability to terminate services at any time
  - How to answer queries completely?
- Design autonomy
  - The ability to restrict the availability and accuracy of information needed for query optimization
  - How to obtain cost information?
- Execution autonomy
  - The ability to execute queries in unpredictable ways
  - How to adapt to this?

# MDBS Query Processing Steps



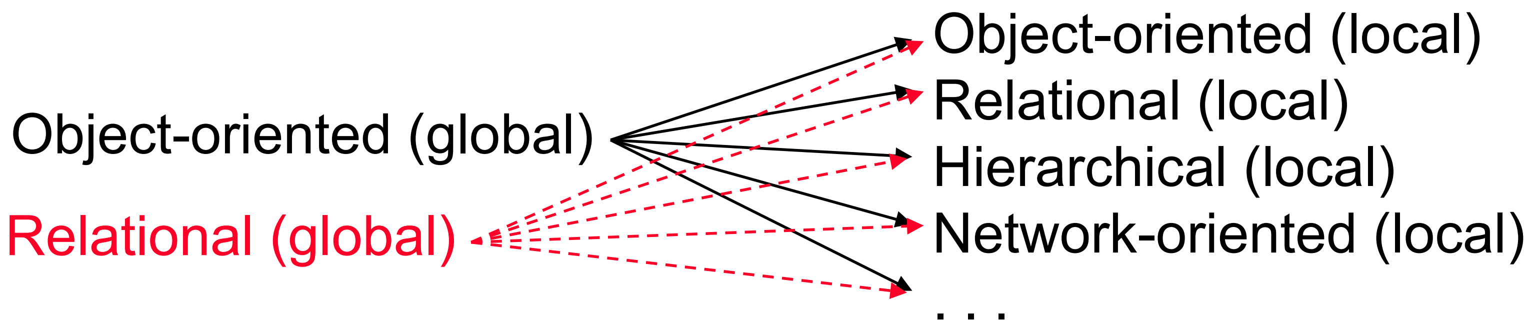

Global/local correspondences

Allocation and capabilities

Local/DBMS mappings

# Query Translation

When a query language of a local DBS is different from the global query language, each export schema subquery for the local DB needs to be translated from the global language to the target language.




Weaker target languages do not support the same operations, emulate required operations in post-processing

- retrieve more data than requested by the query
- then post-process data to compute correct query result

# Query Fragmentation

- Similar to query fragmentation problem for homogeneous distributed DBSs
- Complicating factors:
  - Autonomy
    - Little information about "how" the subquery will be executed by the Local DBS
  - Heterogeneous Data Definition Languages
    - Weaker modeling languages do not support the same manipulation "features"
    - Must use multiple techniques in order to define a consistent global data model
    - Query fragmentation must produce a set of subqueries that reverse the operations used to create/define the global schema

- Processing Steps:

  (1) Replace names from the global schema with "fullnames" from the export schemas

  (2) If a subquery involves multiple export schemas, then break the query into queries that operate on one export schema and insert data communication operators to exchange intermediate results between local database systems

# Global Query Optimization

- Similar to global query optimization for homogeneous distributed DBSs (many algorithms can be used directly)
- But only possible under the following assumptions:
  - No data inconsistency (the global schema correctly represents the semantics of disjoint, overlapping, and conflicting data)
  - Know the characteristics of local DBSs
    - e.g., statistical info on data cardinalities and selectivities are available
  - Can transfer partial data results between different local DBSs
    - Major impact on post-processing plans
- Primary Considerations:
  - Post-processing Strategy
  - Parallel Execution Possibilities
  - Global Cost Function/Estimation

# Post-Processing Strategies

- Three Strategies:

  1) Control site performs all intermediate and post-processing operations (I&PP-ops)
     - Heavy workload; minimal parallelism

  2) Control site performs I&PP-ops for multi-DB results; Multi-DB managers, and HDBMS agents on the local database sites perform I&PP-ops for DBSs within one multi-DB environment
     - Better work load balance; more parallelism

  3) Use strategy #2 and use “pushdown” to get the local database systems to perform I&PP-ops
     - Possible if local DBMS can read intermediate results from external sources, and sort, join, etc. can be directly invoked

# Global Cost Estimation

- Differs from cost estimation in homogeneous distributed DBSs
  - Little (or no) info on QP algorithms and data statistics in local DBS
- Cost Estimation Function
  - Cost to execute each subquery on the local DBMSs
  - Cost to execute all I&PP-ops
    - via pushdown or by any HDBMS agent/service
- Use a simplified cost function

Cost = Initialization cost
+ cost to retrieve a set of objects
+ cost to process a set of objects

- Run test queries on the local DBSs to get time estimates for ops
  - Selection, with and without an index
  - Join (testing for different algorithms: sort, hash, or indexed based algorithms)

# Query Optimization and Execution

- Takes a query expressed on local relations and produces a distributed QEP to be executed by the wrappers and mediator
- Three main problems
  - Heterogeneous cost modeling
    - To produce a global cost model from component DBMS
  - Heterogeneous query optimization
    - To deal with different query computing capabilities
  - Adaptive query processing
    - To deal with strong variations in the execution environment

# Heterogeneous Cost Modeling

- Goal: determine the cost of executing the subqueries at component DBMS
- Three approaches
  - Black-box: treats each component DBMS as a black-box and determines costs by running test queries
  - Customized: customizes an initial cost model
  - Dynamic: monitors the run-time behavior of the component DBMS and dynamically collect cost information

# Query Translation and Execution

- Performed by wrappers using the component DBMS
  - Conversion between common interface of mediator and DBMS-dependent interface
    - Query translation from wrapper to DBMS
    - Result format translation from DBMS to wrapper
  - Wrapper has the local schema exported to the mediator (in common interface) and the mapping to the DBMS schema
  - Common interface can be query-based (e.g. ODBC or SQL/MED) or operator-based
- In addition, wrappers can implement operators not supported by the component DBMS, e.g. join

# Wrapper Management Issues

- Wrappers mostly used for read-only queries
  - Makes query translation and wrapper construction easy
  - DBMS vendors provide standard wrappers
    - ODBC, JDBC, ADO, etc.
- Updating makes wrapper construction harder
  - Problem: heterogeneity of integrity constraints
    - Implicit in some legacy DB
  - Solution: reverse engineering of legacy DB to identify implicit constraints and translate in validation code in the wrapper
- Wrapper maintenance
  - schema mappings can become invalid as a result of changes in component DB schemas
    - Use detection and correction, using mapping maintenance techniques

# Transaction Management?

- Local transactions: access data at a single site outside of the global MDBS control.
- Global transactions: are executed under the MDBS control -> mostly read-only

Local DBMSs have three types of autonomy:

| Autonomy Type | Definition | Resulting Problem |
|---|---|---|
| Design | No changes can be made to the local DBMS software to support the HDBMS | Non-serializable schedule for global transactions |
| Execution | Each local DBMS controls execution of global subtransactions and local transactions ( the commit/abort decision) | Non-atomic & non-durable global transactions |
| Communication | Local DBMS do not communicate with each other and they do not exchange execution control information | Distributed deadlock can not be detected |