Outline

- Introduction
- Background
- Distributed Database Design
- Database Integration
  - Schema Matching
  - Schema Mapping
- Semantic Data Control
- Distributed Query Processing
- Multimedia Query Processing
- Distributed Transaction Management
- Data Replication
- Parallel Database Systems
- Distributed Object DBMS
- Peer-to-Peer Data Management
- Web Data Management
- Current Issues

Problem Definition

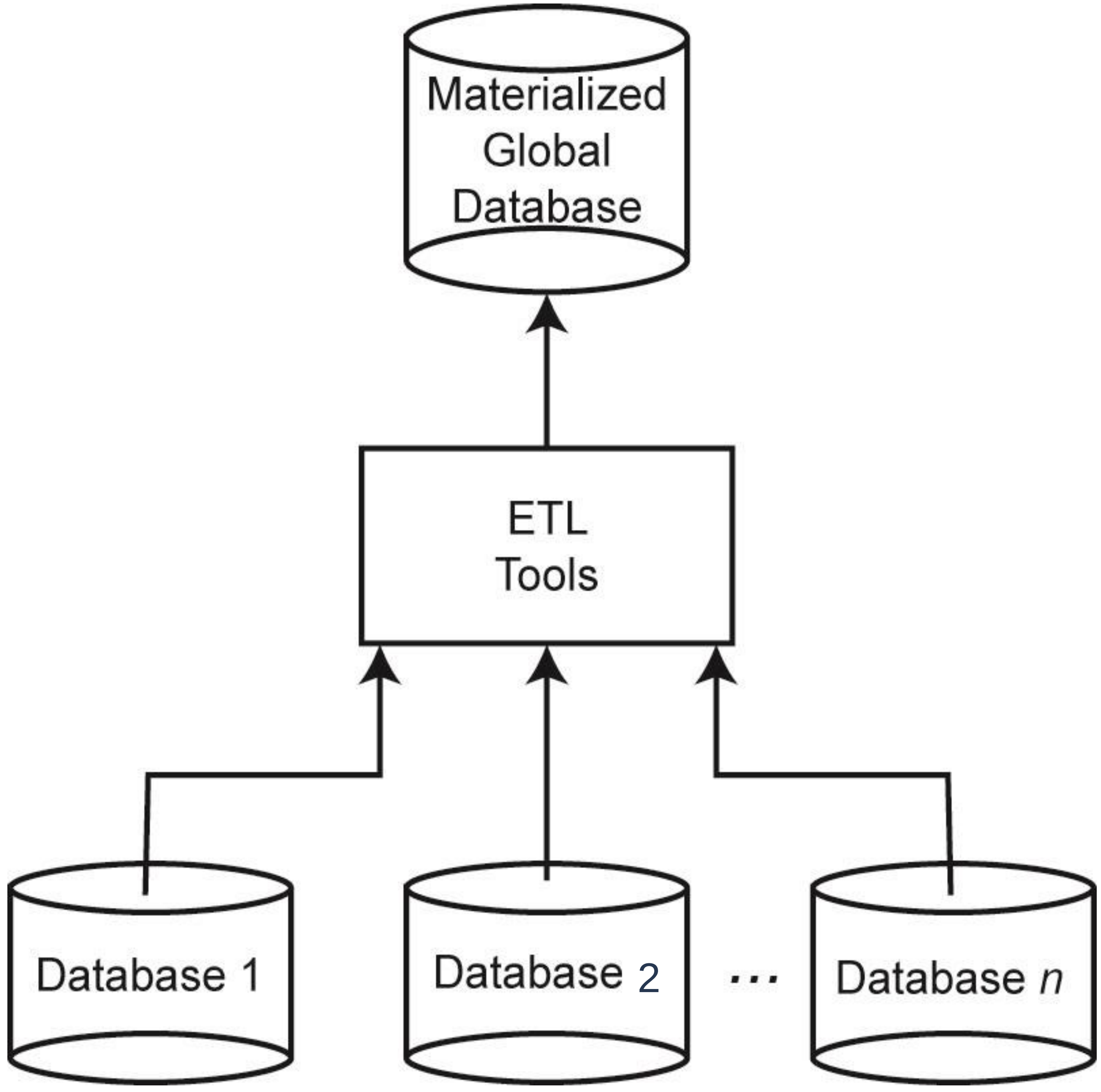
- Given existing databases with their Local Conceptual Schemas (LCSs), how to integrate the LCSs into a Global Conceptual Schema (GCS)
  - GCS is also called *mediated schema*

- Bottom-up design process

Integration Alternatives

- Physical integration
  - Source databases integrated and the integrated database is materialized
  - Data warehouses
- Logical integration
  - Global conceptual schema is virtual and not materialized
  - Enterprise Information Integration (EII)

Data Warehouse Approach



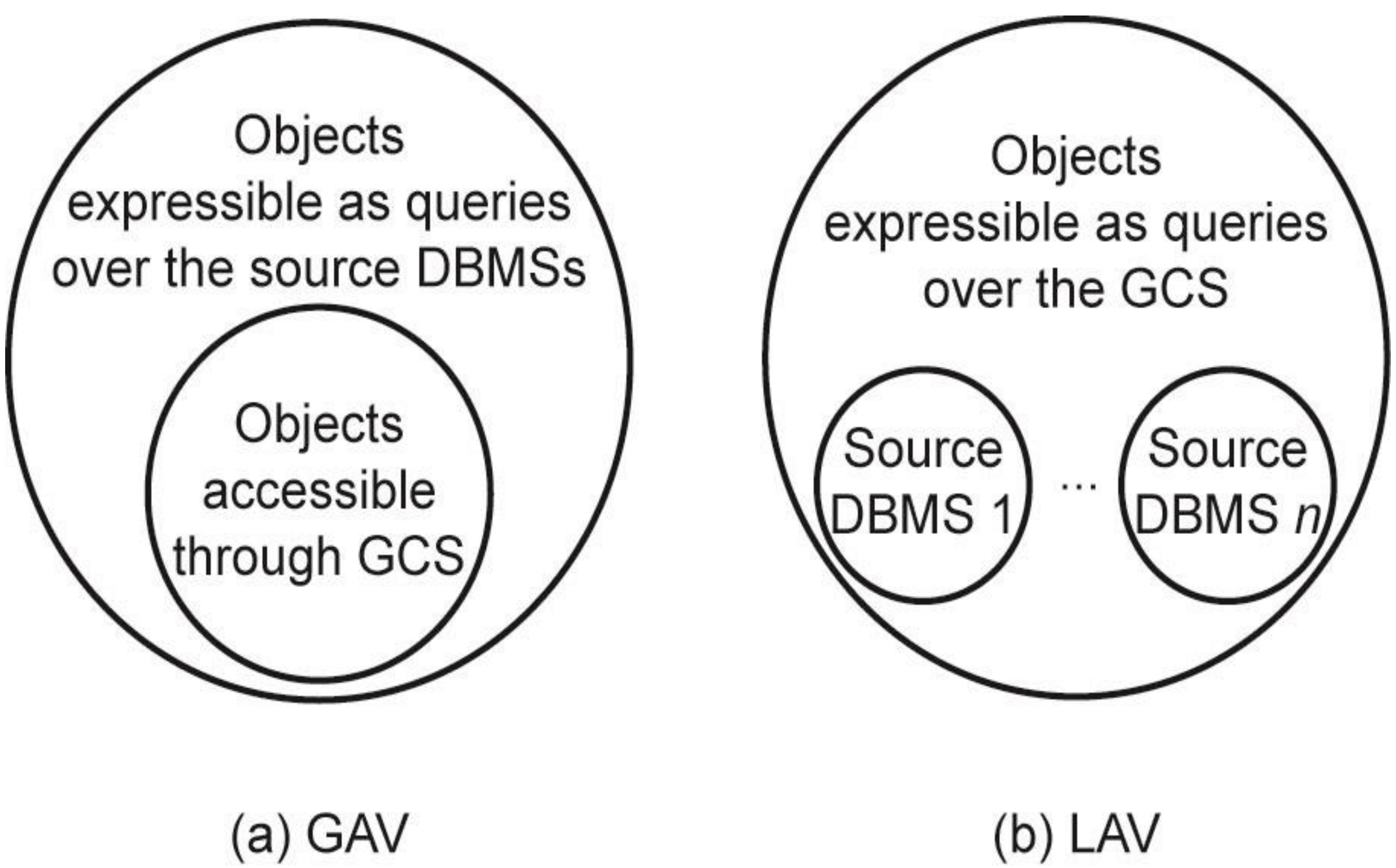
Bottom-up Design

- GCS (also called mediated schema) is defined first
  - Map LCSs to this schema
  - As in data warehouses

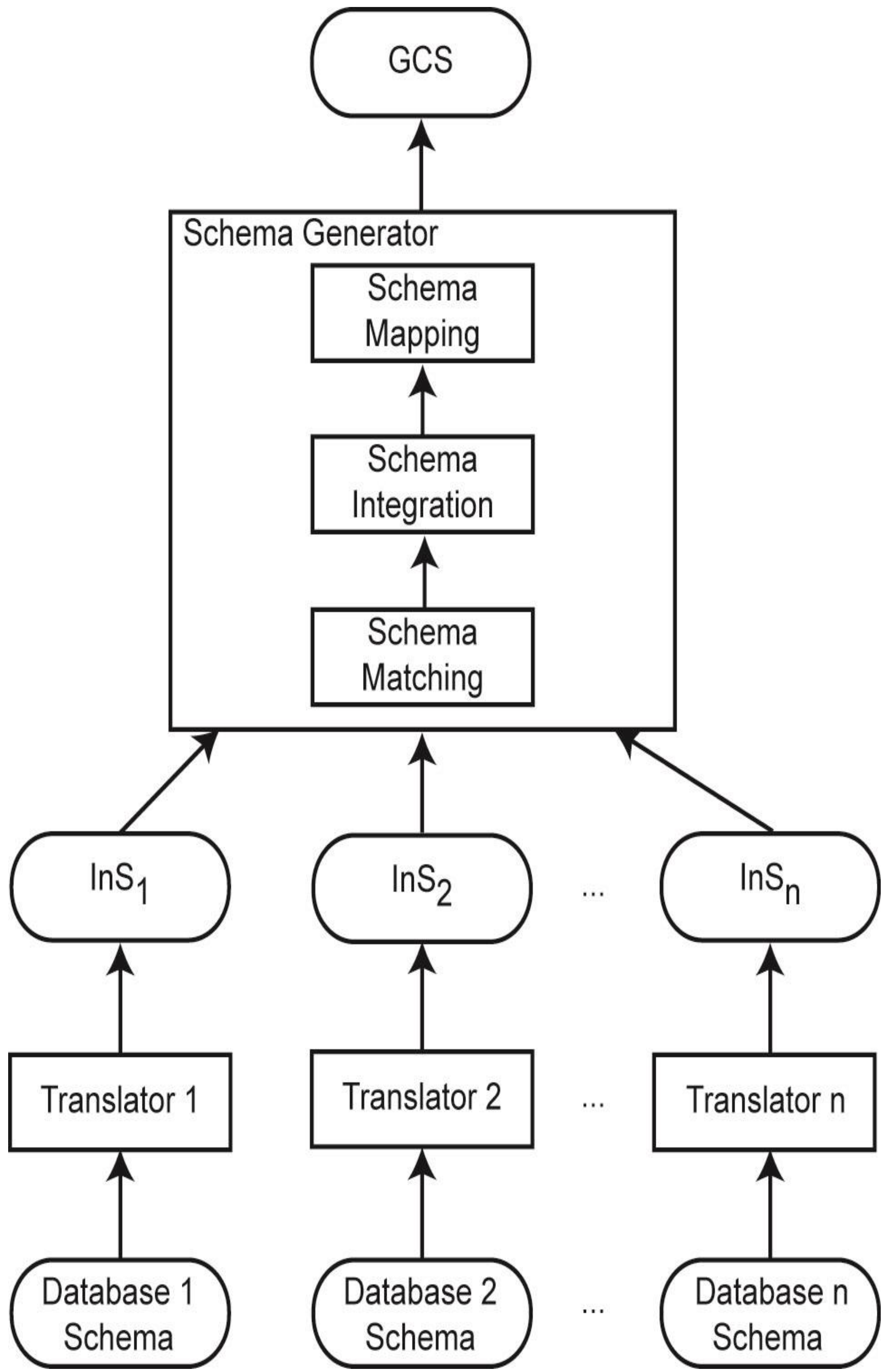
- GCS is defined as an integration of parts of LCSs
  - Generate GCS and map LCSs to this GCS

GCS/LCS Relationship

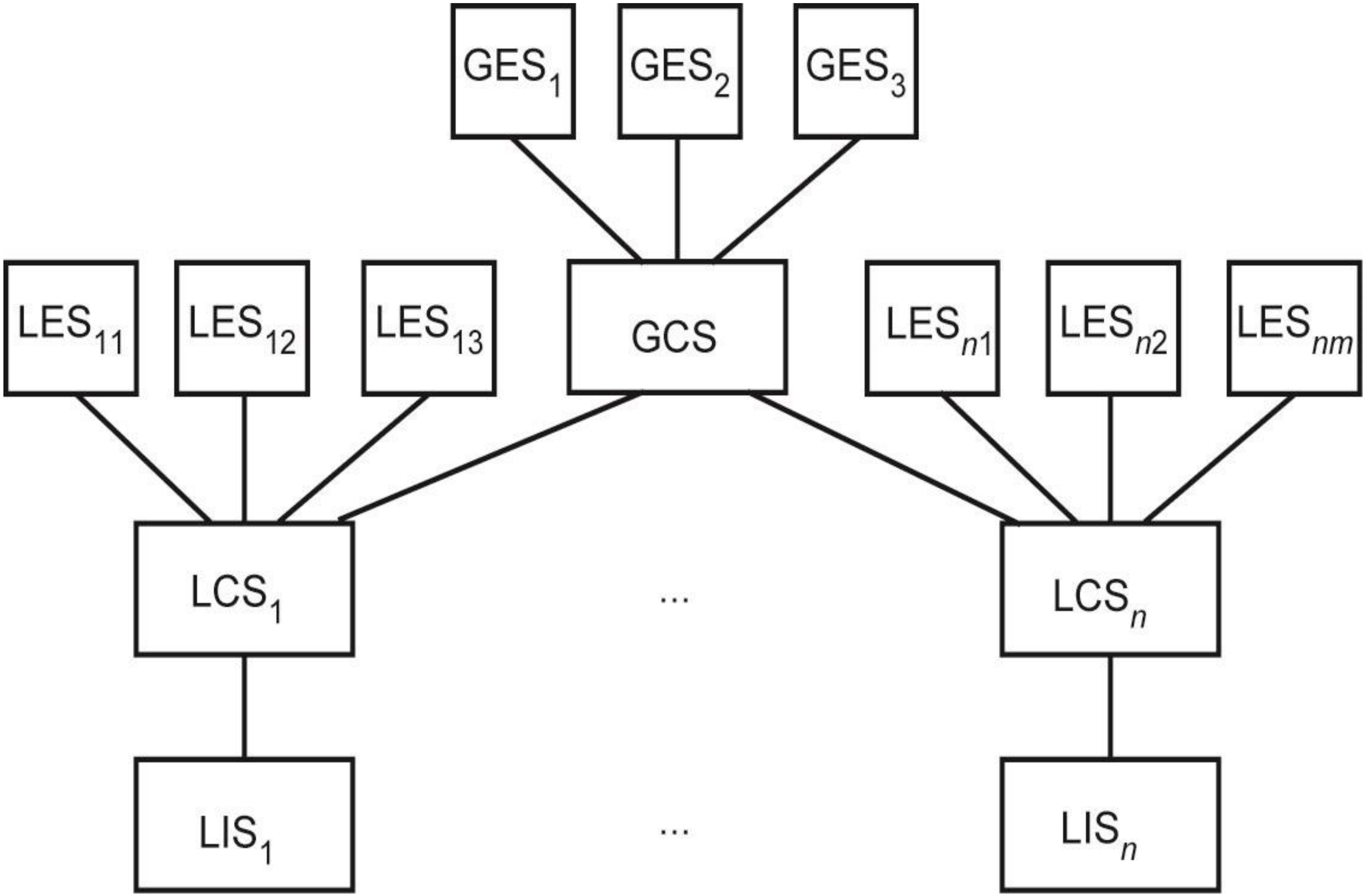
- Local-as-view
  - The GCS definition is assumed to exist, and each LCS is treated as a view definition over it
- Global-as-view
  - The GCS is defined as a set of views over the LCSs



Database Integration Process



Recall Access Architecture



Database Integration Issues

- 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 Translation

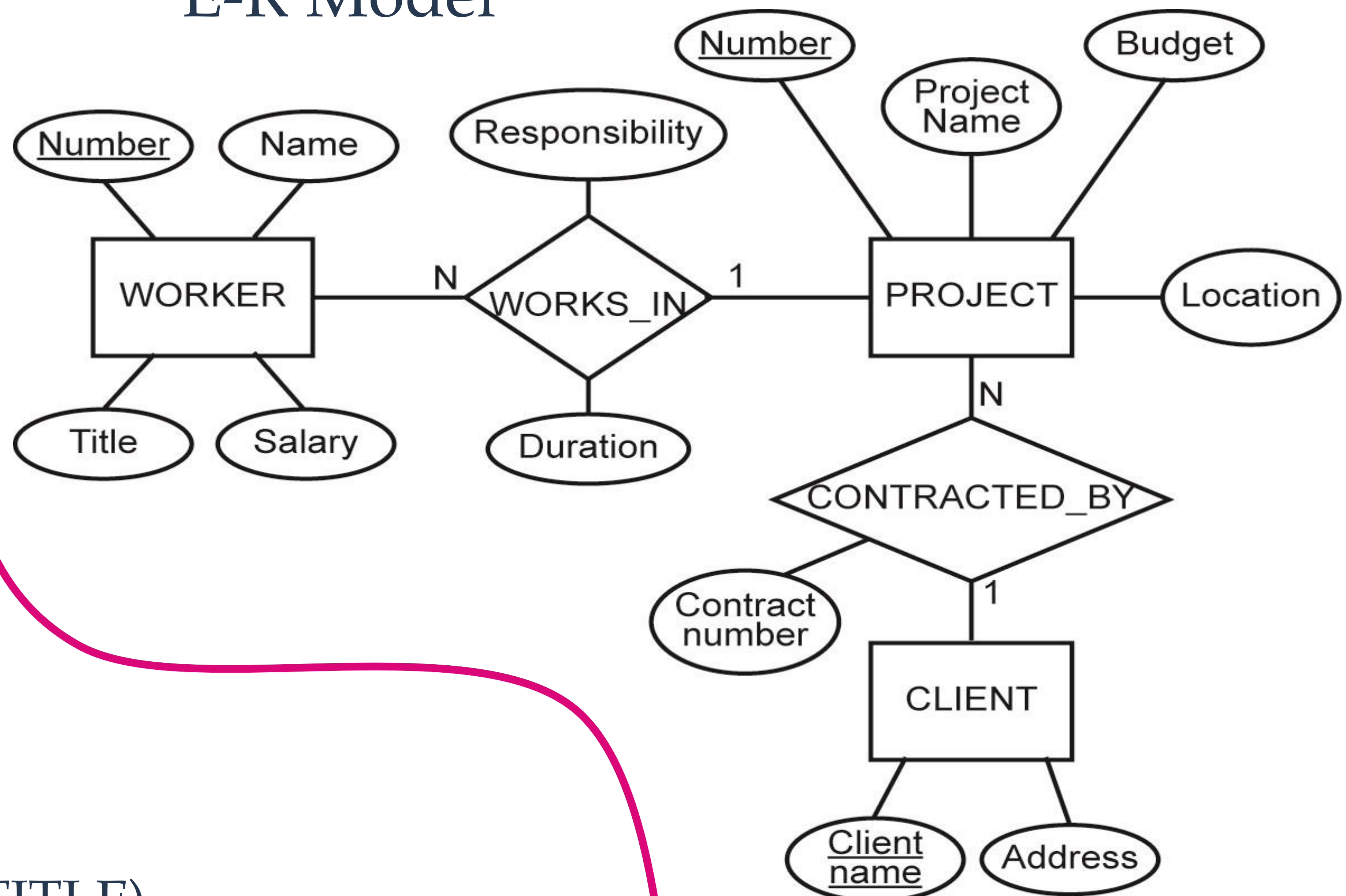
- What is the canonical data model?
  - Relational
  - Entity-relationship
    - DIKE (Database Intensional Knowledge Extractor)
  - Object-oriented
    - ARTEMIS(Analysis of Requirements: Tool Environment for Multiple Information Sources) : A Process Modeling and Analysis Tool Environment
  - Graph-oriented
    - DIPE(A Distributed Environment for Medical Image Processing), TranScm, COMA, Cupid
    - Preferable with emergence of XML
    - No common graph formalism
- Mapping algorithms
  - These are well-known

Schema Generation

- Schema matching
  - Finding the correspondences between multiple schemas
  - Determine the syntactic and semantic correspondences among the translated LCS elements or between individual LCS elements and the pre-defined GCS elements
- Schema integration
  - Creation of the GCS (or mediated schema) using the correspondences
  - Integration of the common schema elements into a global conceptual (mediated) schema
- 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

Running Example

E-R Model



Relational

EMP(ENO, ENAME, TITLE)
PROJ(PNO, PNAME, BUDGET, LOC, CNAME)
ASG(ENO, PNO, RESP, DUR)
PAY(TITLE, SAL)

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

Structural heterogeneity

- Type conflicts occur when the same object is represented by an attribute in one schema and by an entity (relation) in another.
- Dependency conflicts occur when different relationship modes (e.g., one-to-one versus many-to-many) are used to represent the same thing in different schemas.
- Key conflicts occur when different candidate keys are available and different primary keys are selected in different schemas.
- Behavioral conflicts are implied by the modeling mechanism. For example, deleting the last item from one database may cause the deletion of the containing entity (i.e., deletion of the last employee causes the dissolution of the department).

Semantic Heterogeneity

- semantic heterogeneity, which is a fairly loaded term without a clear definition.
- It basically refers to the differences among the databases that relate to the meaning, interpretation, and intended use of data
- Synonyms, homonyms, hypernyms. Synonyms are multiple terms that all refer to the same concept. In our database example, PROJ and PROJECT refer to the same concept.
- Homonyms, on the other hand, occur when the same term is used to mean different things in different contexts. Again, in our example, BUDGET may refer to the gross budget in one database and it may refer to the net budget (after some overhead deduction) in another, making their simple comparisondifficult.

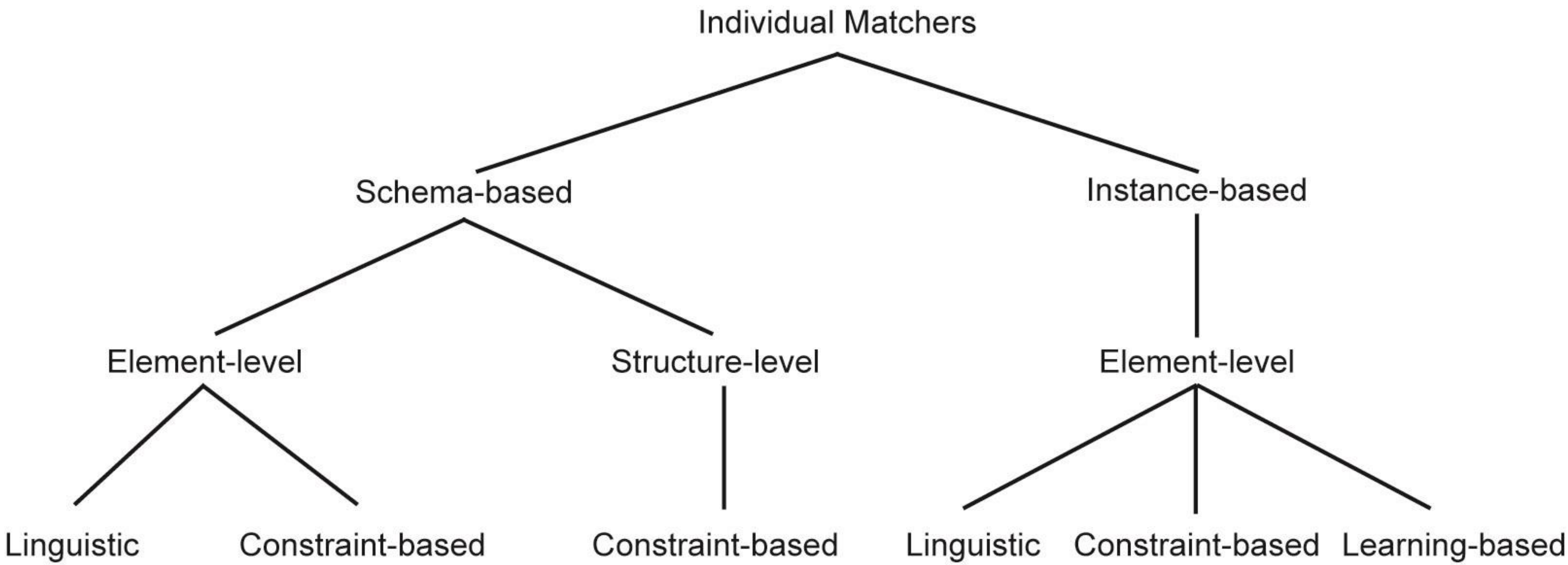
- Hypernym is a term that is more generic than a similar word. Although there is no direct example of it in the databases we are considering, the concept of a Vehicle in one database is a hypernym for the concept of a Car in another (incidentally, in this case, Car is a hyponym of Vehicle).
- These problems can be addressed by the use of domain ontologies that define the organization of concepts and terms in a particular domain.

Schema Matching (contd…)

- 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



Linguistic Schema Matching

- Use element names and other textual information (textual descriptions, annotations)
- May use external sources (e.g., Thesauri)
- ⟨SC1.element-1 ≈ SC2.element-2, *p*,*s*⟩
  - Element-1 in schema SC1 is similar to element-2 in schema SC2 if predicate *p* holds with a similarity value of *s*
- Schema level
  - Deal with names of schema elements
  - Handle cases such as synonyms, homonyms, hypernyms, data type similarities
- Instance level
  - Focus on information retrieval techniques (e.g., word frequencies, key terms)
  - "Deduce" similarities from these

Linguistic Matchers

- Use a set of linguistic (terminological) rules
- Basic rules can be hand-crafted or may be discovered from outside sources (e.g., WordNet)
- Predicate *p* and similarity value *s*
  - hand-crafted ⇒ specified,
  - discovered ⇒ may be computed or specified by an expert after discovery
- Examples
  - ⟨uppercase names ≈ lower case names, *true*, 1.0⟩
  - ⟨uppercase names ≈ capitalized names, *true*, 1.0⟩
  - ⟨capitalized names ≈ lower case names, *true*, 1.0⟩
  - ⟨DB1.ASG ≈ DB2.WORKS_IN, *true*, 0.8⟩

Automatic Discovery of Name Similarities

- Affixes
  - Common prefixes and suffixes between two element name strings
- N-grams
  - Comparing how many substrings of length *n* are common between the two name strings
- Edit distance
  - Number of character modifications (additions, deletions, insertions) that needs to be performed to convert one string into the other
- Soundex code
  - Phonetic similarity between names based on their soundex codes
- Also look at data types
  - Data type similarity may suggest stronger relationship than the computed similarity using these methods or to differentiate between multiple strings with same value

N-gram Example

- 3-grams of string “Responsibility” are the following:
  - Res
  - ibi
  - bip
  - ili
  - lit
  - ity
  - sib
  - esp
  - spo
  - pon
  - ons
  - nsi
- 3-grams of string “Resp” are
  - Res
  - esp
- 3-gram similarity: 2/12 = 0.17

Edit Distance Example

- Again consider “Responsibility” and “Resp”
- To convert “Responsibility” to “Resp”
  - Delete characters “o”, “n”, “s”, “i”, “b”, “i”, “l”, “i”, “t”, “y”
- To convert “Resp” to “Responsibility”
  - Add characters “o”, “n”, “s”, “i”, “b”, “i”, “l”, “i”, “t”, “y”
- The number of edit operations required is 10
- Similarity is 1 – (10/14) = 0.29

Constraint-based Matchers

- Data always have constraints – use them
  - Data type information
  - Value ranges
  - …
- Examples
  - RESP and RESPONSIBILITY: n-gram similarity = 0.17, edit distance similarity = 0.19 (low)
  - If they come from the same domain, this may increase their similarity value
  - ENO in relational, WORKER.NUMBER and PROJECT.NUMBER in E-R
  - ENO and WORKER.NUMBER may have type INTEGER while PROJECT.NUMBER may have STRING

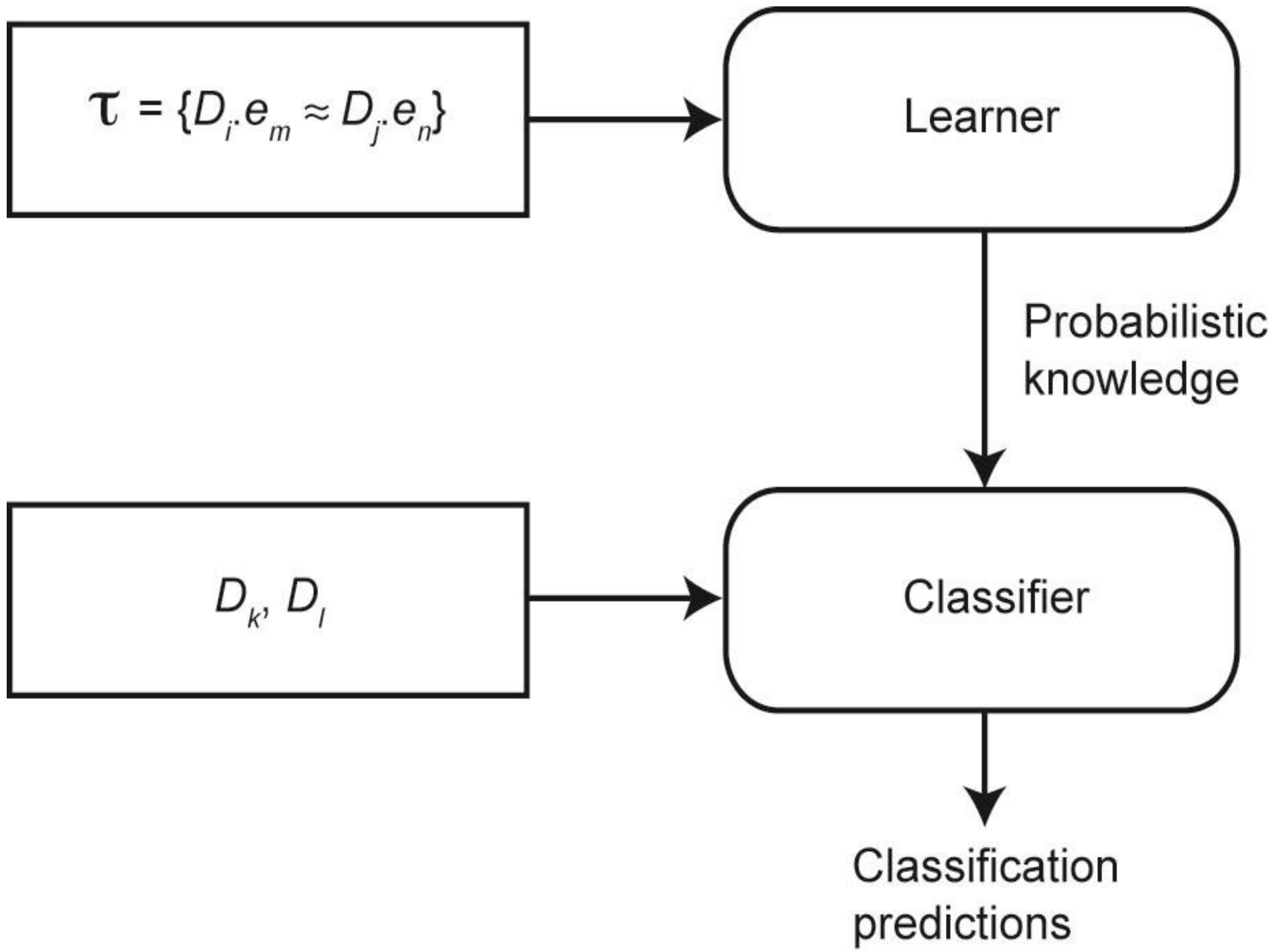
Constraint-based Structural Matching

- If two schema elements are structurally similar, then there is a higher likelihood that they represent the same concept
- Structural similarity:
  - Same properties (attributes)
  - “Neighborhood” similarity
    - Using graph representation
    - The set of nodes that can be reached within a particular path length from a node are the neighbors of that node
    - If two concepts (nodes) have similar set of neighbors, they are likely to represent the same concept

Learning-based Schema Matching

- Use machine learning techniques to determine schema matches
- Classification problem: classify concepts from various schemas into classes according to their similarity. Those that fall into the same class represent similar concepts
- Similarity is defined according to features of *data instances*
- Classification is "learned" from a training set

Learning-based Schema Matching

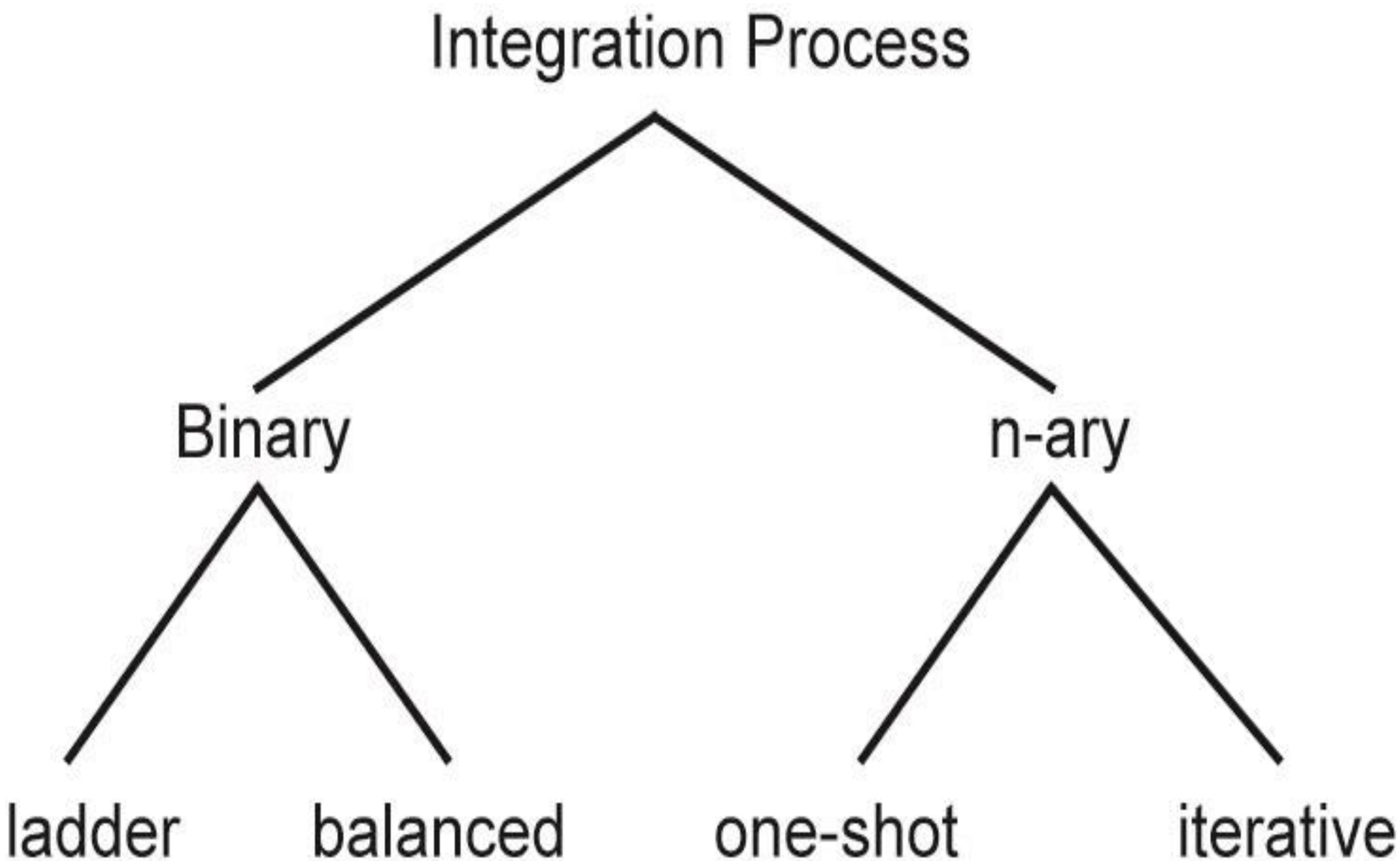


Combined Schema Matching Approaches

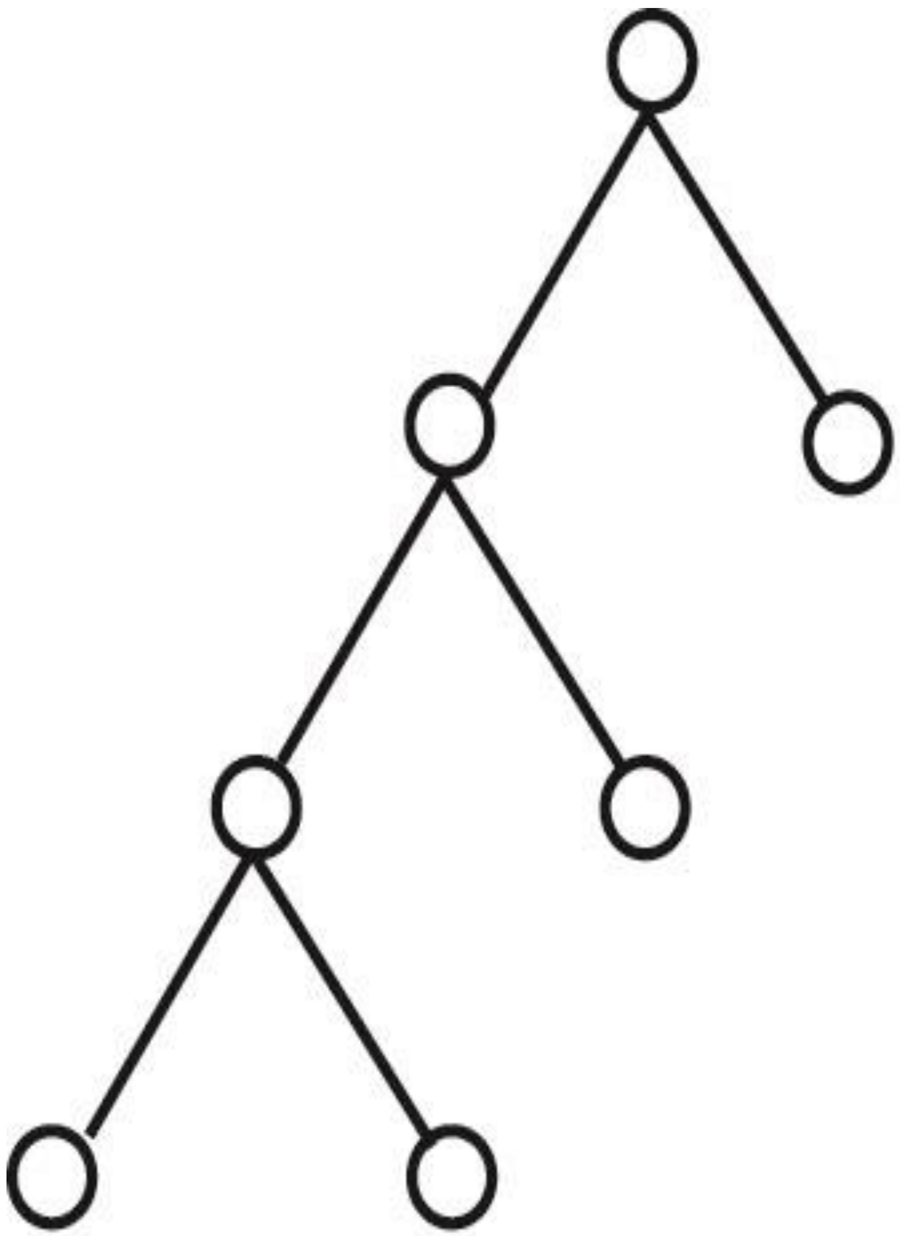
- Use multiple matchers
  - Each matcher focuses on one area (name, etc)
- Meta-matcher integrates these into one prediction
- Integration may be simple (take average of similarity values) or more complex (see Fagin's work)

Schema Integration

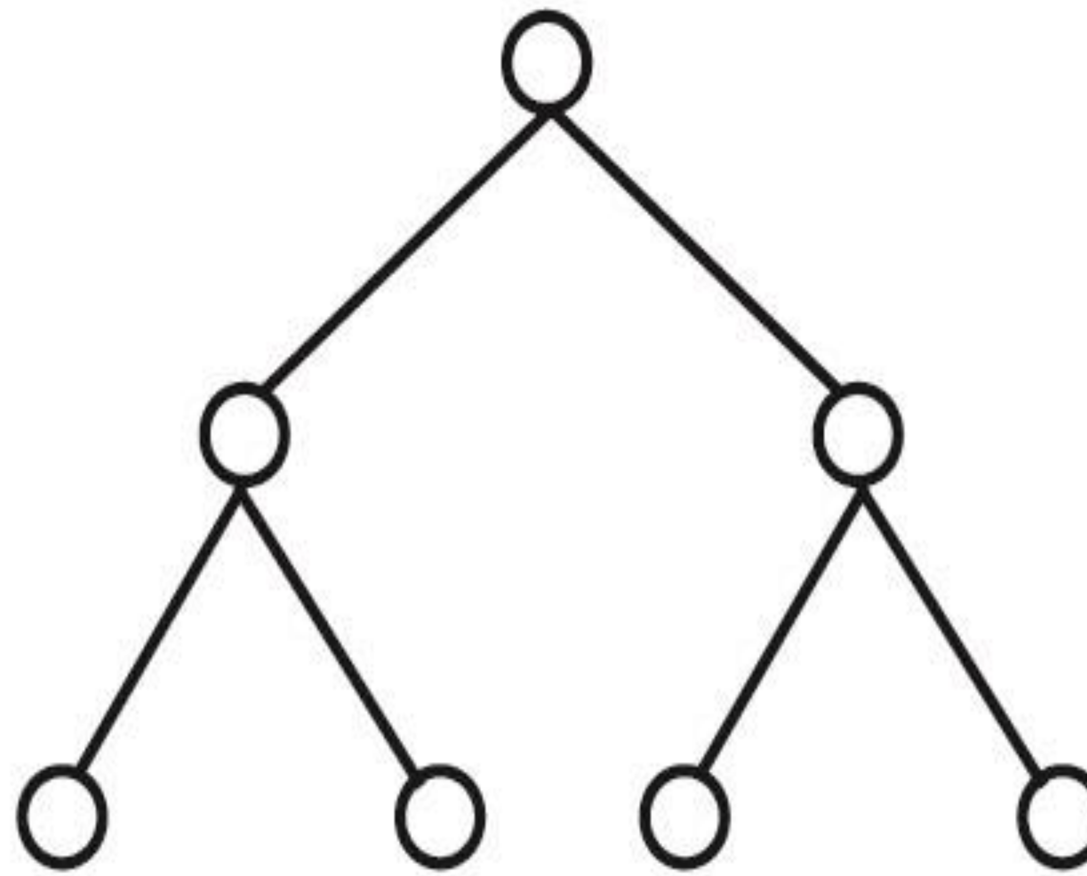
- Use the correspondences to create a GCS
- Mainly a manual process, although rules can help



Binary Integration Methods

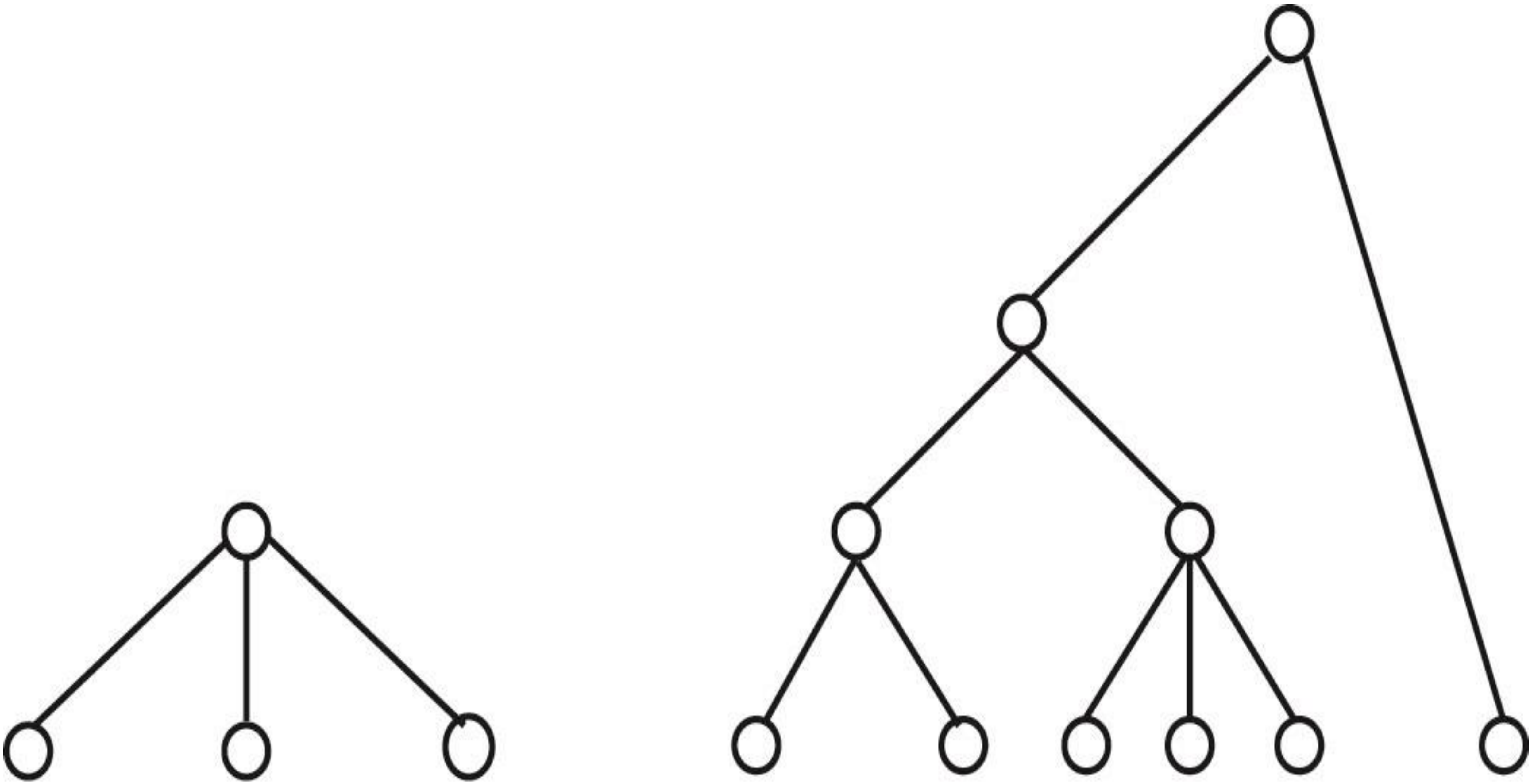


(a) Stepwise



(b) Pure binary

N-ary Integration Methods



(a) One-pass (b) Iterative

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

Mapping Creation

Given

- A source LCS $[\mathcal{S} = \{S_i\}]$
- A target GCS $[\mathcal{T} = \{T_i\}]$
- A set of value correspondences discovered during schema matching phase $[\mathcal{V} = \{V_i\}]$

Produce a set of queries that, when executed, will create GCS data instances from the source data.

We are looking, for each T_k, a query Q_k that is defined on a (possibly proper) subset of the relations in $\mathcal{S}$ such that, when executed, will generate data for T_i from the source relations

Mapping Creation Algorithm

General idea:

- Consider each T_k in turn. Divide V_k into subsets $\{V_k^1, \ldots, V_k^n\}$ such that each V_k^j specifies one possible way that values of T_k can be computed.
- Each V_k^j can be mapped to a query q_k^j that, when executed, would generate *some* of T_k's data.
- Union of these queries gives

$$Q_k (= \cup_j q_k^j)$$