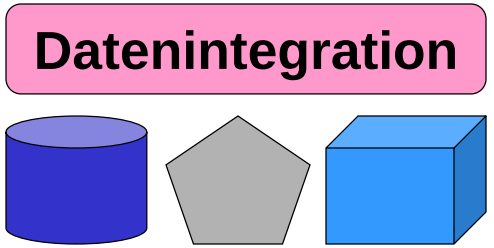
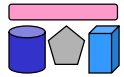
Datenintegration



Kapitel 6: Schemamanagement

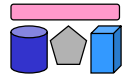
Dr. Anika Groß
Sommersemester 2016

Universität Leipzig
Institut für Informatik
http://dbs.uni-leipzig.de



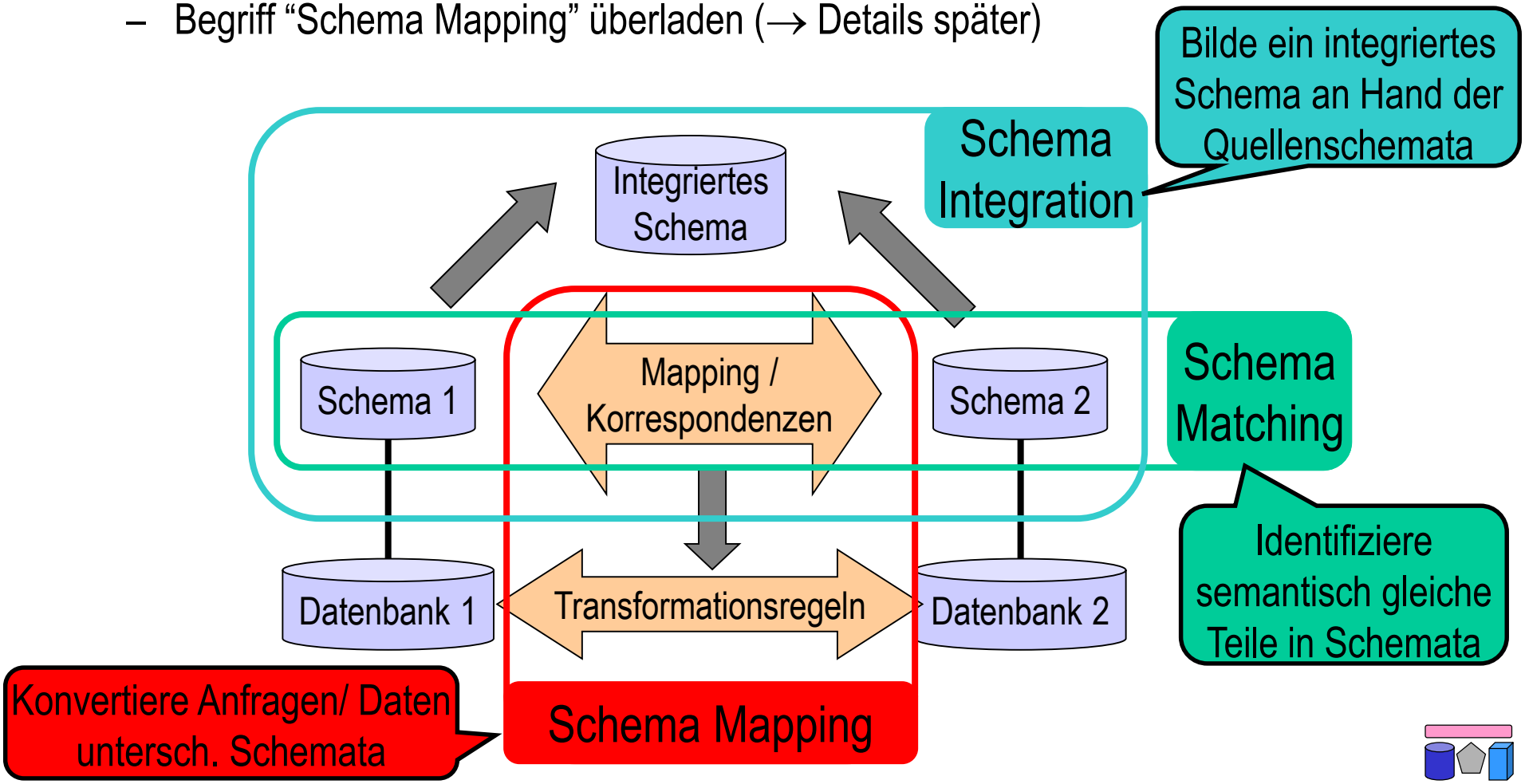
Inhalt

- Schemaintegration
  - Erzeugung eines globalen / integrierten Schemas
- Schema Matching
  - Identifikation semantischer Korrespondenzen zwischen Schemata
- Schema Mapping
  - Erstellung von Transformationsanfragen auf Basis von Schema-Matching-Ergebnis
- Model Management
  - Generische Manipulation von Metadaten



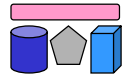
Schema-Integration, -Matching, -Mapping

- Integration der Metadaten (Schemata) ist Voraussetzung für Integration der Instanzdaten
- Schema-Integration, -Matching und -Mapping wichtige Teilaspekte
  - Begriff “Schema Mapping” überladen (→ Details später)



Schemaintegration

- Erzeugung eines integrierten (globalen) Schemas aus den Exportschemata einer Menge von Datenquellen
- Zentrales Problem bei der Integration strukturierter Informationen
- Forderungen
  - Vollständigkeit: Alle Daten der Quellsysteme sind repräsentiert
    - Eigentlich nicht immer notwendig
  - Korrektheit: Alle Daten werden semantisch korrekt repräsentiert
    - Integritätsconstraints, Kardinalitäten, …
  - Minimal: Minimales oder wenigstens übersichtliches Schema
    - Insbesondere redundanzfrei (Normalformen?)
  - Verständlich
- Viel erforscht, insb. im Umfeld föderierter Datenbanken



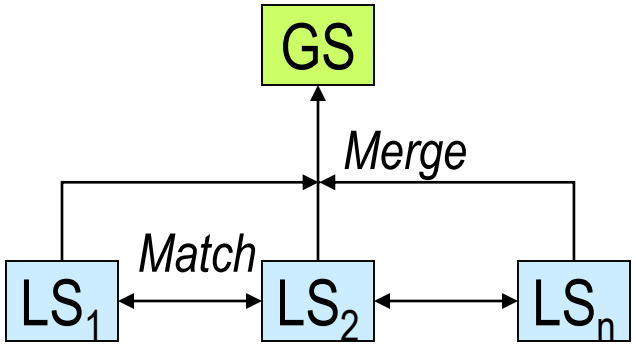
Bottom-Up- vs. Top-Down-Integration

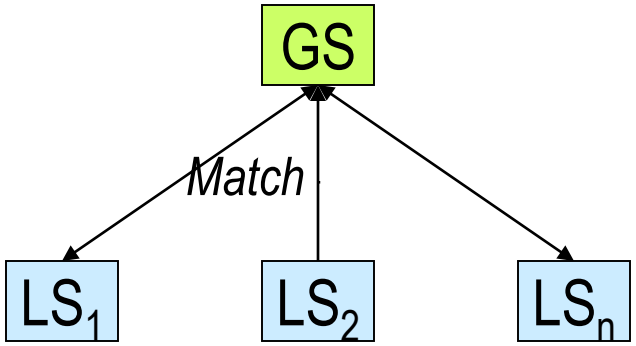
Bottom-Up (Global-as-View)

- globales Schema GS entspricht gemeinsamer Sicht (View) auf die zugrundeliegenden Quellen
  - vollständiges Mischen aller Source-Schemata in globales Schema
  - alle Daten sollen integriert werden
- setzt Abgleich zwischen Source-Schemas voraus, insbesondere Bestimmung von Korrespondenzen / Konflikten
- neue Quelle ändert globales Schema

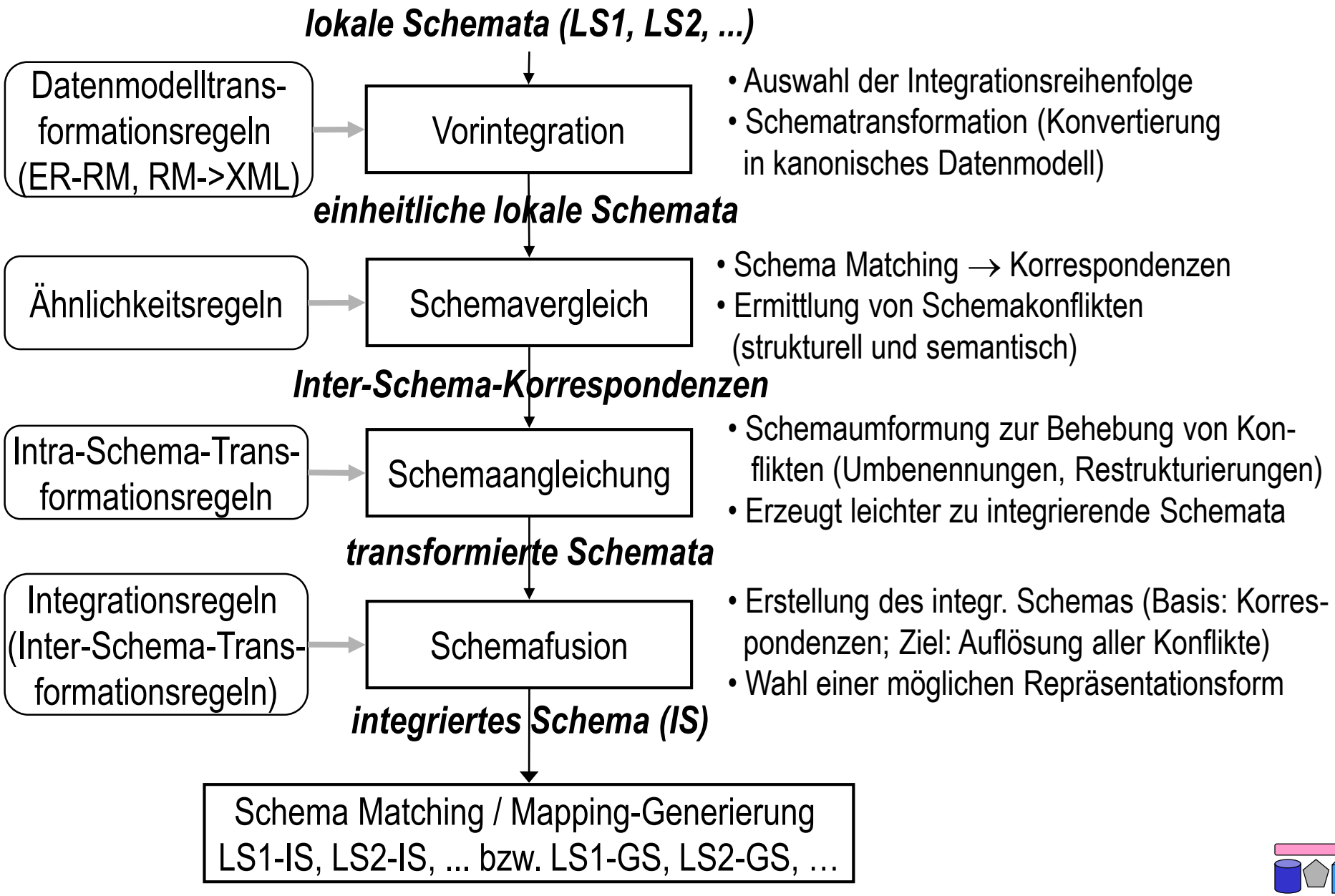
Top-Down (Local-as-View)

- globales Schema GS ist vorgegeben
  - globaler Informationsbedarf
- jede Source S wird unabhängig von anderen Sources mit globalem Schema abgeglichen, d.h. ein Mapping GS - LS erstellt
- aufwändige Query-Verarbeitung bei virtueller Integration (Kap. 4)
- GS berücksichtigt i.a. nur Teile der lokalen Schemata

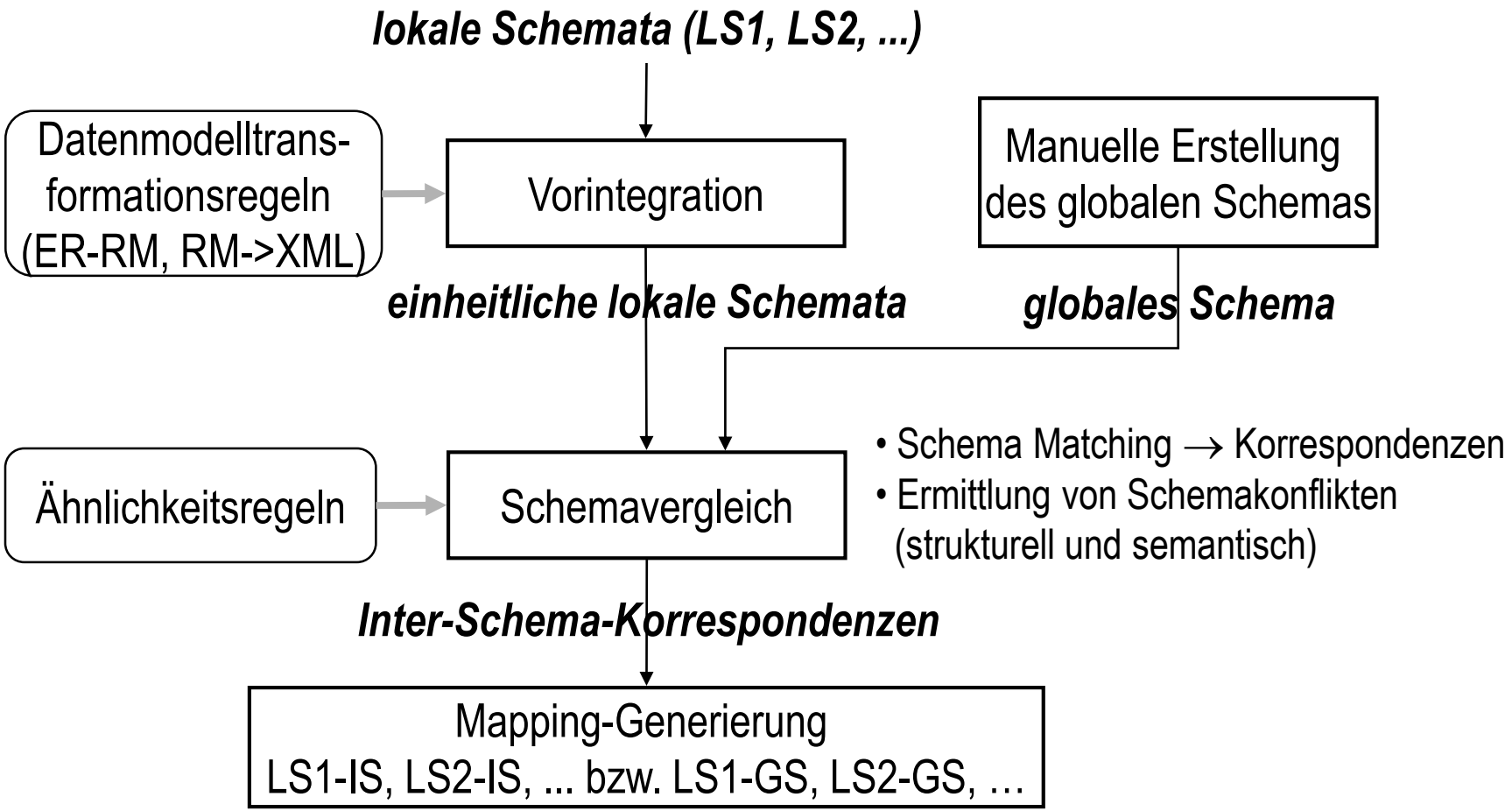


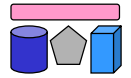


Bottom-Up-Schema-Integration



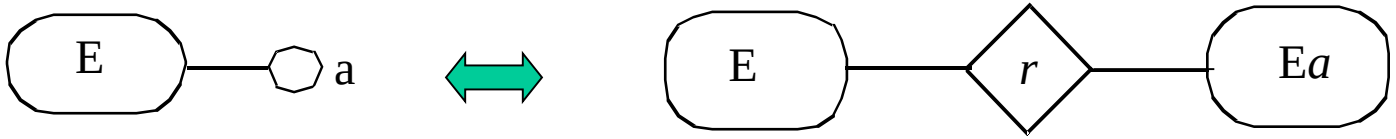
Top-Down-Schema-Integration



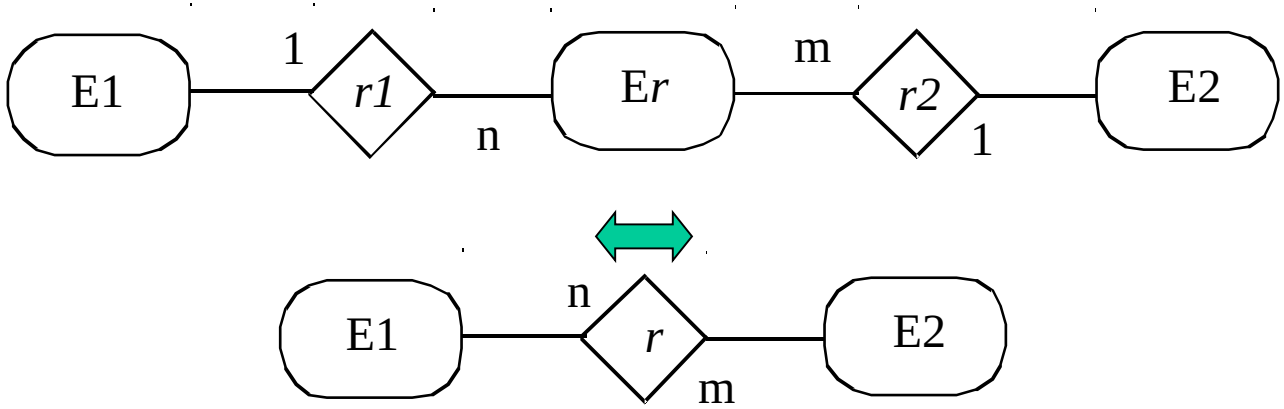


Schema-Konformationstransformationen

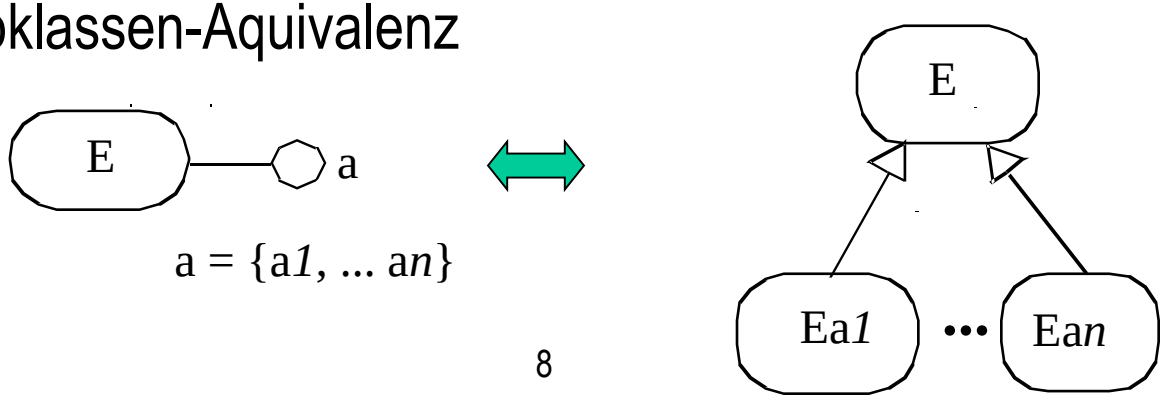
- Entity-/Attribut-Äquivalenz



- Entity-/Relationship-Äquivalenz

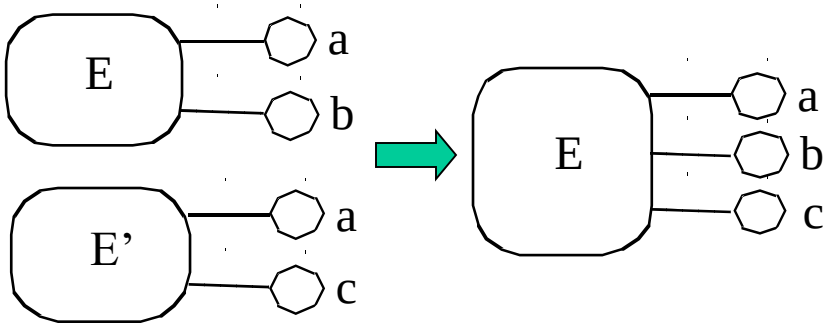


- Attribut-/Subklassen-Äquivalenz

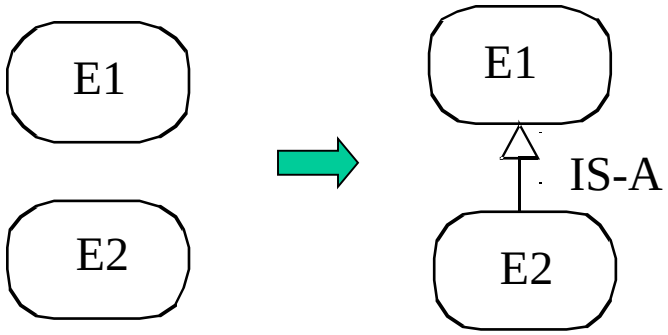


Schema-Merging-Transformationen

- Vereinigung der Attribute

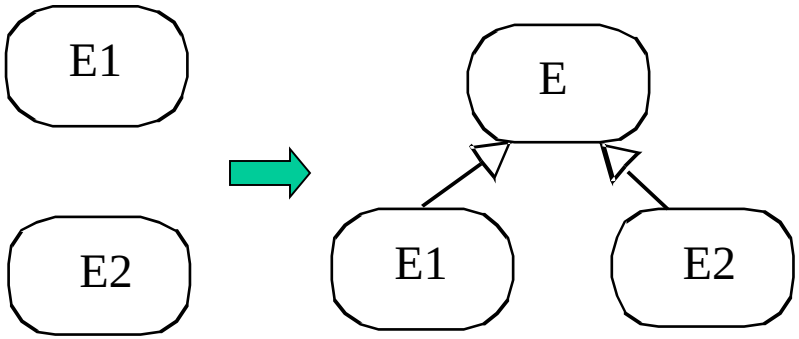


- Einführung einer IS-A-Beziehung



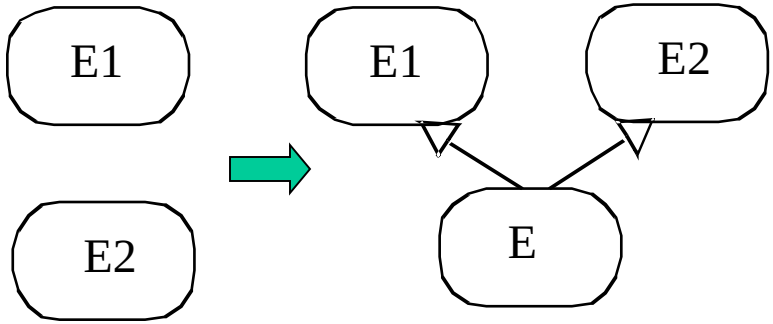
RWS(E1) contains RWS(E2)

- Einführung einer Superklasse



RWS(E1) disjoint RWS(E2)

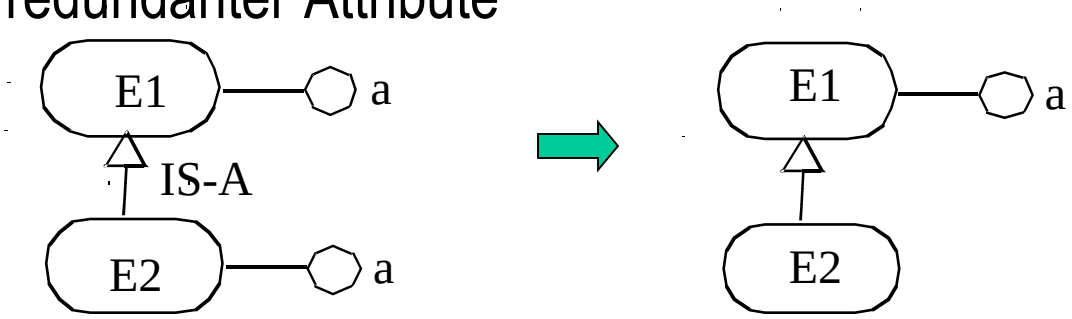
- Einführung einer Subklasse



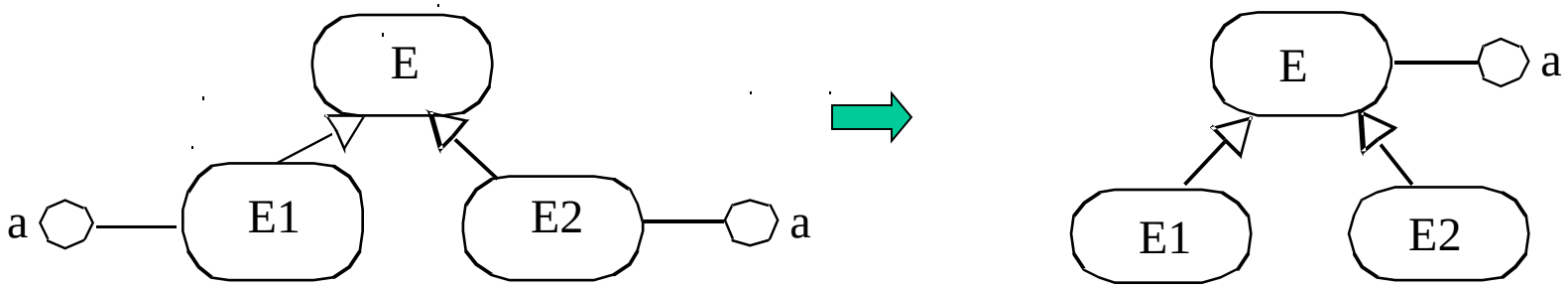
RWS(E1) overlaps RWS(E2)

Schema-Restrukturierungstransformationen

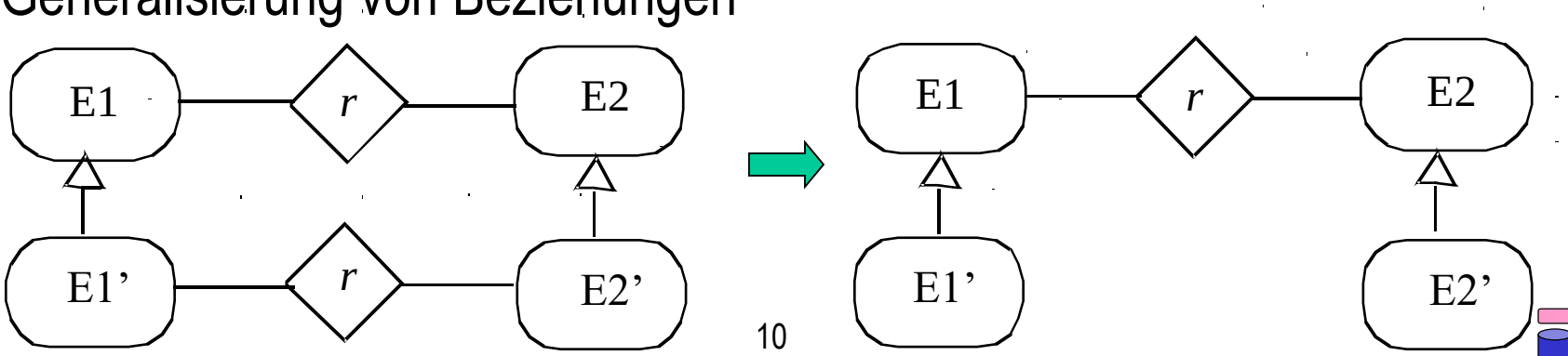
- Entfernung redundanter Attribute



- Generalisierung von Attributen

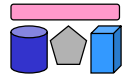


- Generalisierung von Beziehungen



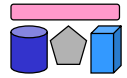
Schemaintegration: Zusammenfassung

- Generelles Vorgehen: Korrespondenzen finden → Schema ableiten
- bisherige Schemaintegrationsansätze weitgehend manuell
  - Vorgehensbeschreibungen (Regeln) statt Algorithmen
  - nutzerdefinierte Korrespondenzen und Konfliktbehandlung
  - Nutzung spezifischen Schema-/Domain-Wissens
  - aufwändig / fehleranfällig vor allem für größere Schemata
  - nicht skalierbar auf viele Schemata
  - Hoher Anpassungsaufwand bei Schemaänderungen
- Skalierbarkeit erfordert semi-automatische Lösungen / Tools
  - Vollautomatische Lösungen aufgrund semantischer Heterogenität nicht möglich
  - Namensproblematik (Synonyme, Homonyme)
  - begrenzte Mächtigkeit von Metadaten / Schemasprachen
  - Neigung zu sehr großen, komplexen Schemata (Forderung: Vollständigkeit)
- Schemaintegration nur ein Zwischenschritt bei Informationsintegration
- Wichtiger Aspekt: Schema Matching



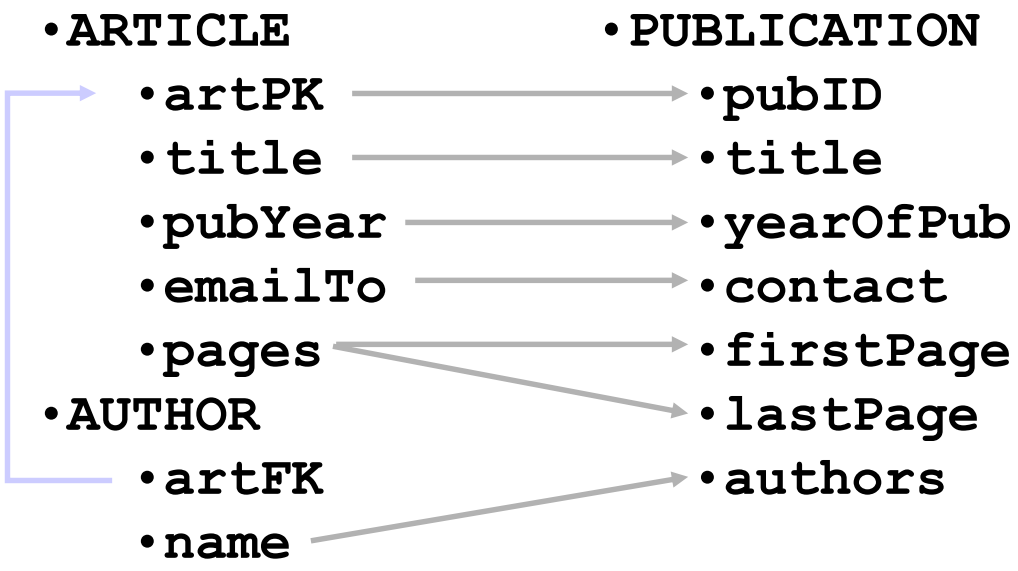
Schema Matching: Definition

- Identifikation semantischer Korrespondenzen zwischen Schemata
- Eingabe
  - zwei Schemata S1 und S2
  - evtl. Dateninstanzen zu S1 und S2
  - evtl. weiteres Hintergrundwissen
    - Beschreibungen der Schemata
    - Domänenwissen, z.B. gebräuchliche Abkürzungen, Synonyme, etc
- Ausgabe
  - Mapping zwischen S1 und S2, d.h. Korrespondenzen zwischen semantisch gleichen Schemaelementen
- Anwendungsgebiete
  - DB-Schemas, XML-Schemas, Ontologien, …
- Kritischer Schritt in zahlreichen Applikationen
  - Datenintegration: Data Warehouses, Mediatoren, P2P
  - E-Business: XML Message Mapping, Katalogintegration
  - Semantic Web: Ontology Matching



Schema Matching: Beispiel

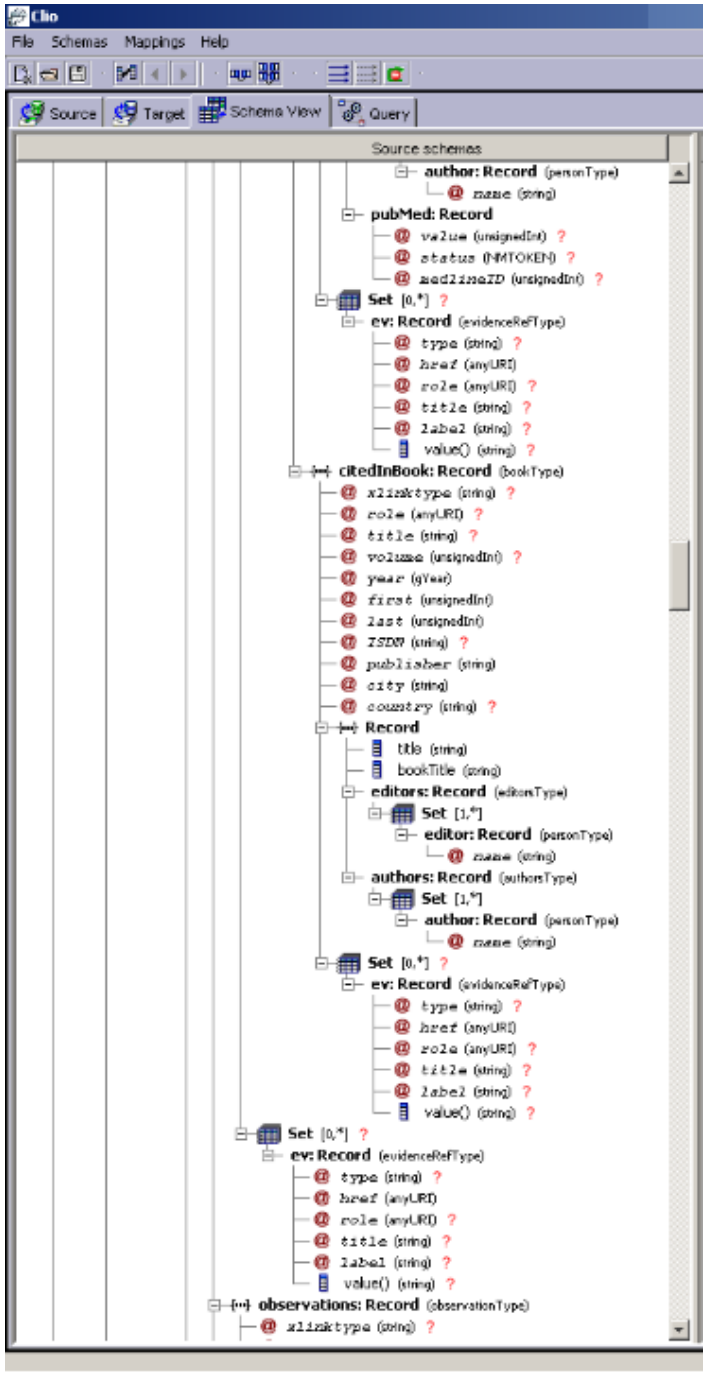
- Zwei Datenbankschemata für wissenschaftliche Publikationen



- Woran erkennt man “korrespondierende Elemente”?

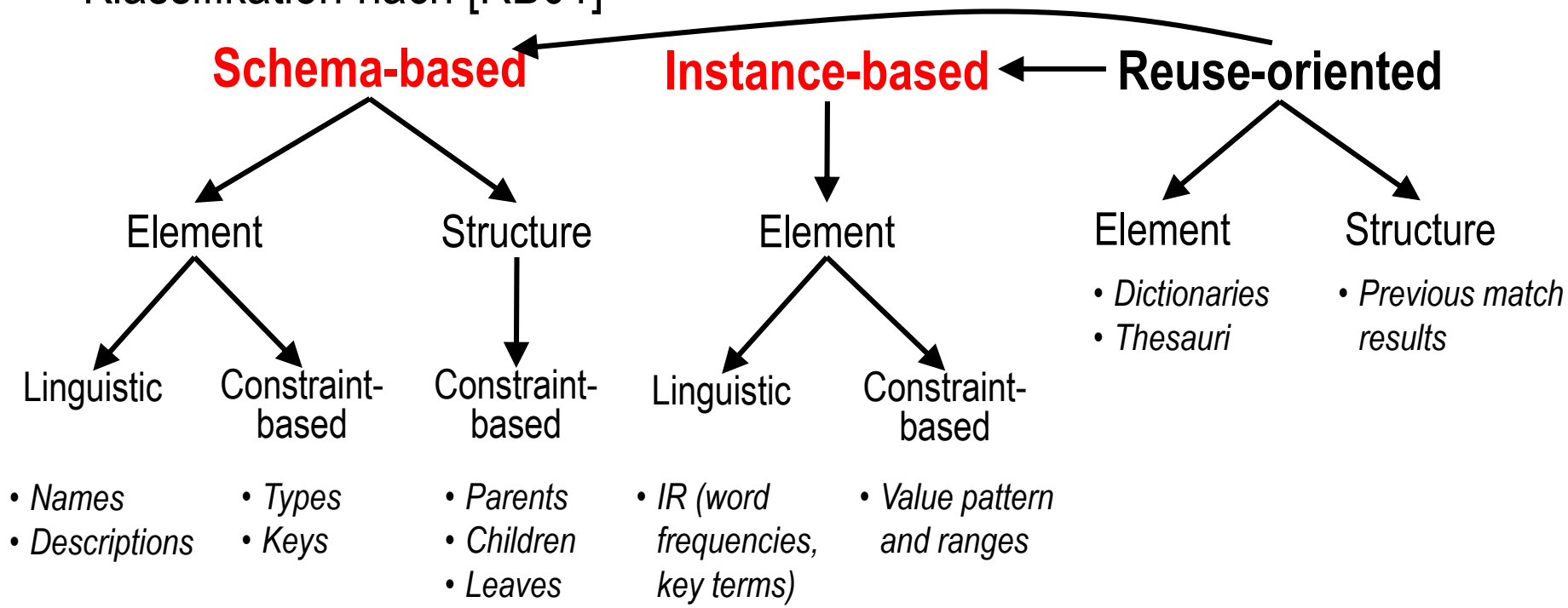
Probleme

- Große, unübersichtliche Schemas
  - > 100 Tabellen, viele Attribute
  - Tiefe Schachtelungen
  - Fremdschlüssel
- Fremde Schemas
  - Unbekannte Synonyme, Homonyme
  - Fremdsprachliche Schemas
- Kryptische Schemas
  - |Attributnamen| ≤ 8 Zeichen
  - |Tabellennamen| ≤ 8 Zeichen
- Schwierig
  - alle Korrespondenzen zu finden
  - falsche Korrespondenzen zu vermeiden



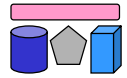
Automatische Ansätze zum Schema Matching

- Klassifikation nach [RB01]



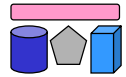
- Kombinierende Ansätze: Composite vs. Hybrid
  - Hybrid = ein Algorithmus wendet mehrere Verfahren an
  - Composite = Kombination der Ergebnisse mehrerer Verfahren

[RB01] Rahm, Bernstein: *A Survey of Approaches to Automatic Schema Matching.* VLDB Journal, 2001



Schema Matching: Schema-based / Element

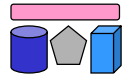
- Gegeben zwei Schemata mit Attributmengen A und B
- Kernidee:
  - Bilde Kreuzprodukt aller Attribute aus A und B.
  - Für jedes Paar vergleiche Ähnlichkeit, z.B. Attributnamen (Label) oder Datentyp
    - Verwendung von Ähnlichkeitsfunktion, z.B. Edit-distance (siehe Kap. 7)
  - Ähnlichste Paare sind Matches
- Probleme:
  - Effizienz
  - Auswahl der besten Matches (globales Matching)
    - Iterativ?
    - Stable Marriage?
  - Synonyme und Homonyme werden nicht erkannt



Schema Matching: Schema-based / Structure

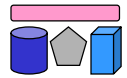
- Gegeben zwei Schemata mit Elementmengen A und B
- Kernidee: Nutze (komplexe) Struktur des Schemas aus
  - Hierarchieebene
  - Elementtyp (Attribut, Relation, ...)
  - Nachbarschaftsbeziehungen
- Beispiel: Similarity Flooding nach [MGMR02]
  - Gegeben initiale Ähnlichkeit zwischen Schemaelementen (z.B. durch edit-distance oder durch Analyse der darunterliegenden Daten)
  - Lasse Ähnlichkeiten „abfärben“ auf die Nachbarn (durch Struktur definiert)
  - Intuition: „Sind alle / viele Nachbarn von x und y ähnlich zueinander, sind (vielleicht) auch x und y ein match.“
  - Analogie: Man „flutet“ das Netzwerk der Ähnlichkeiten bis ein Gleichgewicht erreicht ist.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, Erhard Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. ICDE 2002: 117-128



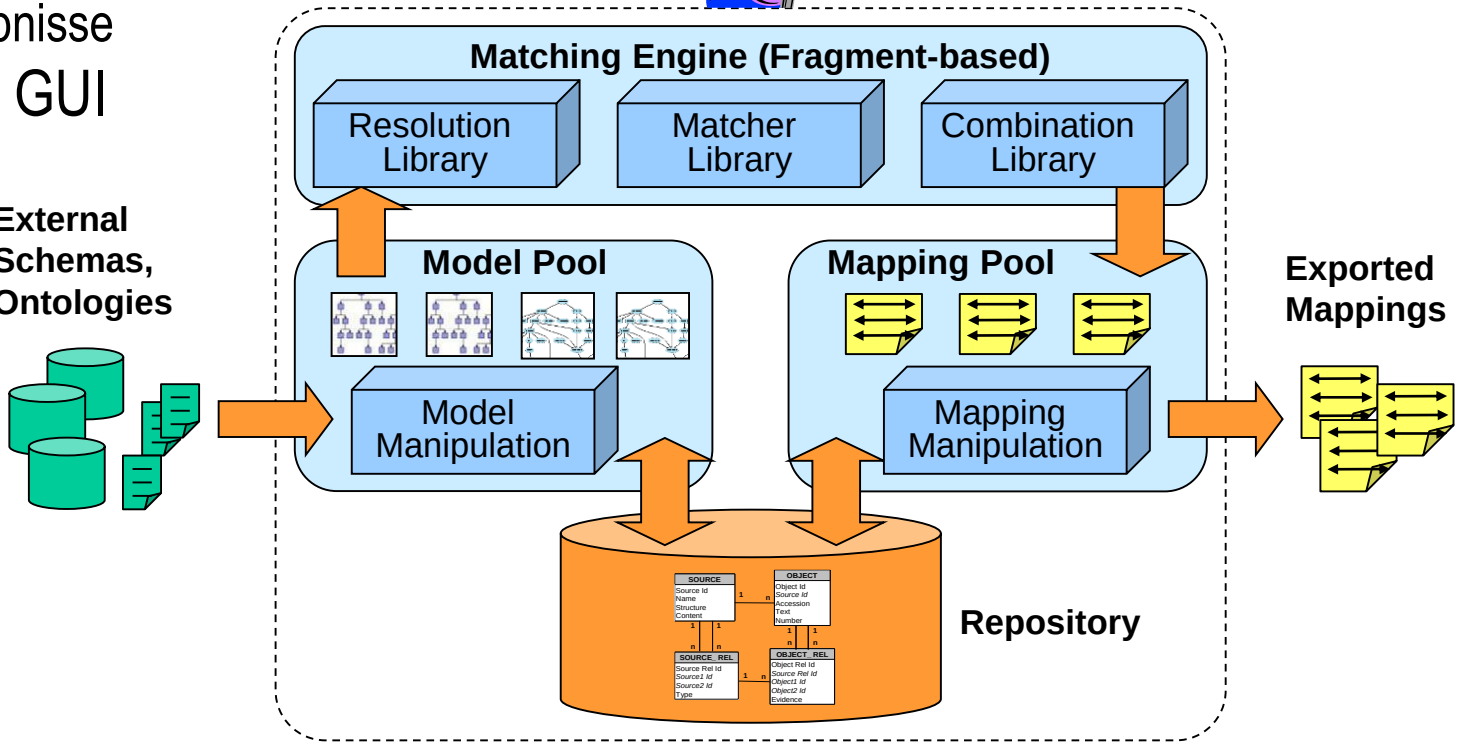
Schema Matching: Instance-based

- Gegeben zwei Schemata mit Attributmengen A und B, jeweils mit darunterliegenden Daten
- Kernidee
  - Für jedes Attribut extrahiere interessante Eigenschaften der Daten
    - Buchstabenverteilung, Länge, etc.
  - Bilde Kreuzprodukt aller Attribute aus A und B.
  - Für jedes Paar vergleiche Ähnlichkeit bzgl. der Eigenschaften
- Probleme
  - Auswahl der Eigenschaften
  - Datenmenge: Sampling
  - Vergleichsmethode, z.B. Naive Bayes
  - Gewichtung (Maschinelles Lernen)



Die Generische Match-Plattform COMA++

- Unterstützt XML, relationale Schemas und OWL-Ontologien
- Composite-Ansatz mit flexibler Konfiguration von Matchern
- Match-Strategien für große Schemas
  - Fragment-basiert
  - Wiederverwendung vorhandener Match-Ergebnisse
- Umfassendes GUI

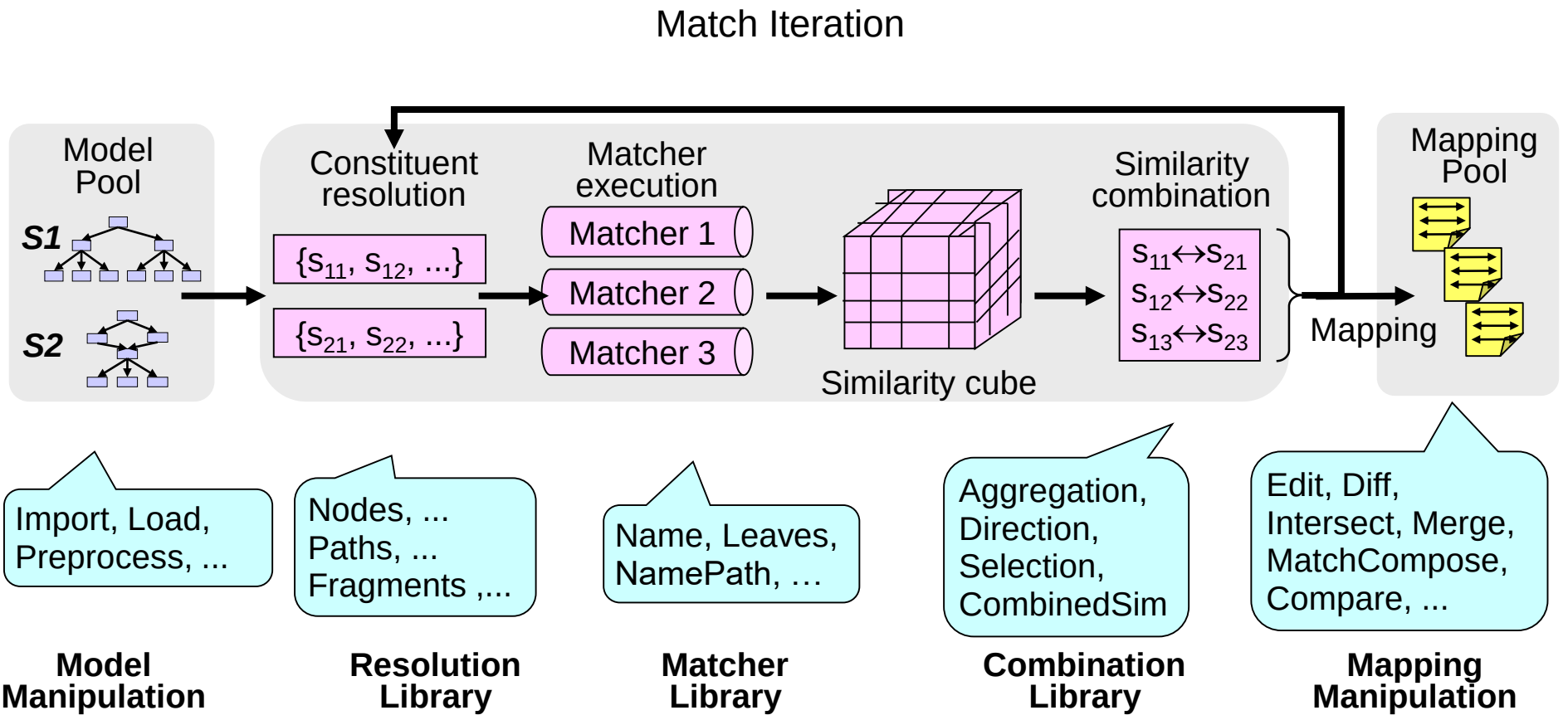


Do, Rahm: *COMA - A System for Flexible Combination of Schema Matching Approaches*. VLDB 2002

Aumüller, Do, Massmann, Rahm: *Schema and Ontology Matching with COMA++*. Sigmod 2005

COMA++: Match-Verarbeitung

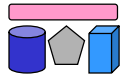
- Ausführung verschiedener Match-Algorithmen
- Manipulation/Kombination der erzeugten Mappings



COMA++: Matcher Bibliothek

- Basis-Matcher:
  - String-Matcher: Synonym, Type, Trigram, Affix, EditDistance
  - Type-Matcher
  - Taxonomie-Matcher
  - Reuse-Matcher: Wiederverwendung von Mappings
- Hybrid-Matcher: feste Kombination anderer Matcher

Name	*Constituents*	*Matchers/Sim measures*	*Combination*
Name	Name tokens	Synonym/Taxonomy, Trigram	Avg, Both, Max1, Avg
NameType	Node	Name, Type	Wgt(0.7,03), Both, Max1, Avg
NameStat	Node	Name, Statistics	
Children	Children	NameType	Avg, Both, Max1, Avg
Leaves	Leaves	NameType	
Parents	Parents	Leaves	
Siblings	Siblings	Leaves	
NamePath	Ascendants	Name	

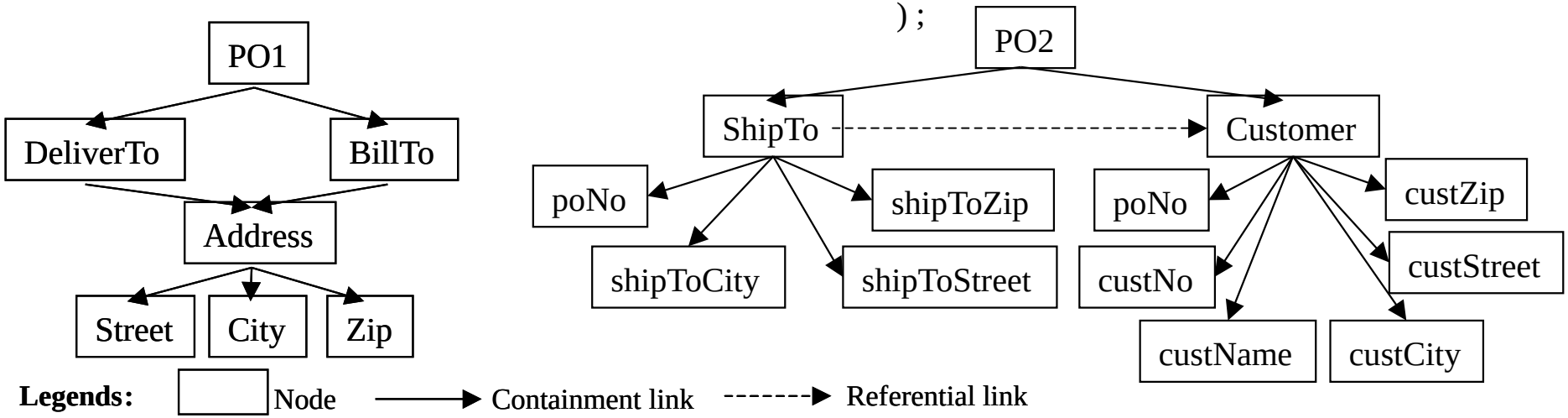


Schema-Beispiel

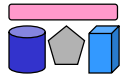
```
<xsd: schema xmlns:xsd="http://www.w3.org/2001/ XMLSchema">
<xsd: complexType name =“PO1" >
  <xsd: sequence>
        <xsd: element name=“ DeliverTo" type="Address"/>
        <xsd: element name=“BillTo" type="Address"/>
  </xsd: sequence>
</xsd: complexType>
<xsd: complexType name="Address" >
   <xsd: sequence>
       <xsd: element name=“Street" type="xsd:string"/>
       <xsd: element name =“City" type="xsd:string"/>
       <xsd: element name=“Zip" type="xsd: decimal"/>
  </xsd: sequence>
</xsd: complexType>
</xsd: schema>
```

```
CREATE TABLE PO2.ShipTo (
 poNo          INT,
 custNo        INT REFERENCES PO2.Customer,
 shipToStreet  VARCHAR(200),
 shipToCity    VARCHAR(200),
 shipToZip     VARCHAR(20),
 PRIMARY KEY (poNo)
) ;
CREATE TABLE PO2.Customer (
 custNo        INT,
 custName      VARCHAR(200),
 custStreet    VARCHAR(200),
 custCity      VARCHAR(200),
 custZip       VARCHAR(20),
 PRIMARY KEY (custNo)
) ;
```

a) XML-Schema (links)
Relationales Schema (rechts)

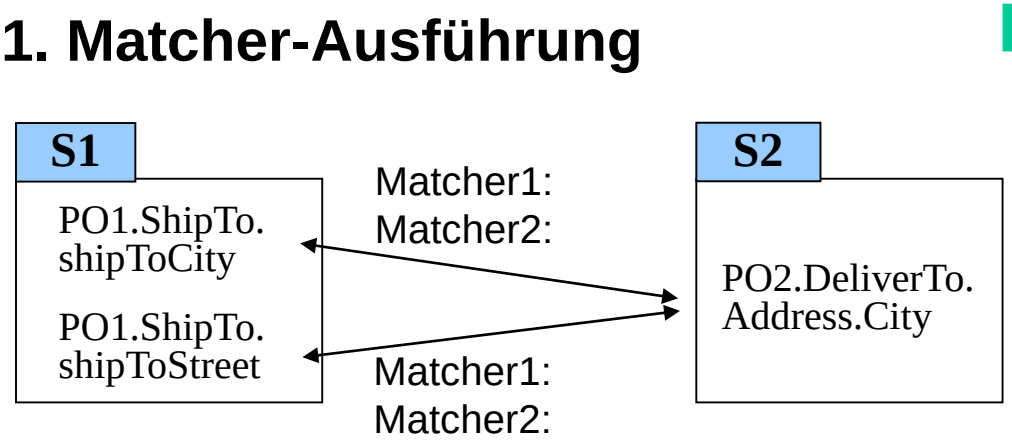


b) Zugehörige (interne) Graphrepräsentation

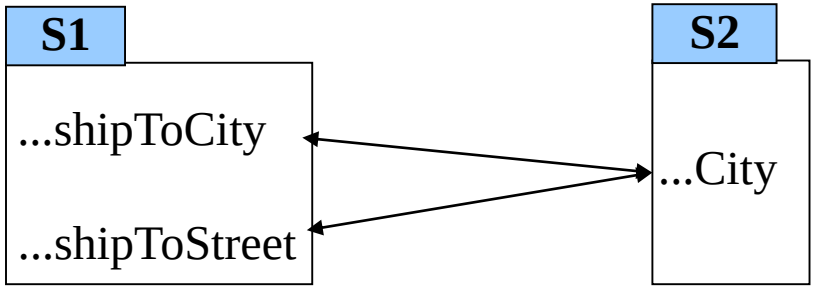


Kombination von Match-Ergebnissen (Beispiel)

1. Matcher-Ausführung



2. Aggregation



3. Selektion

Max1

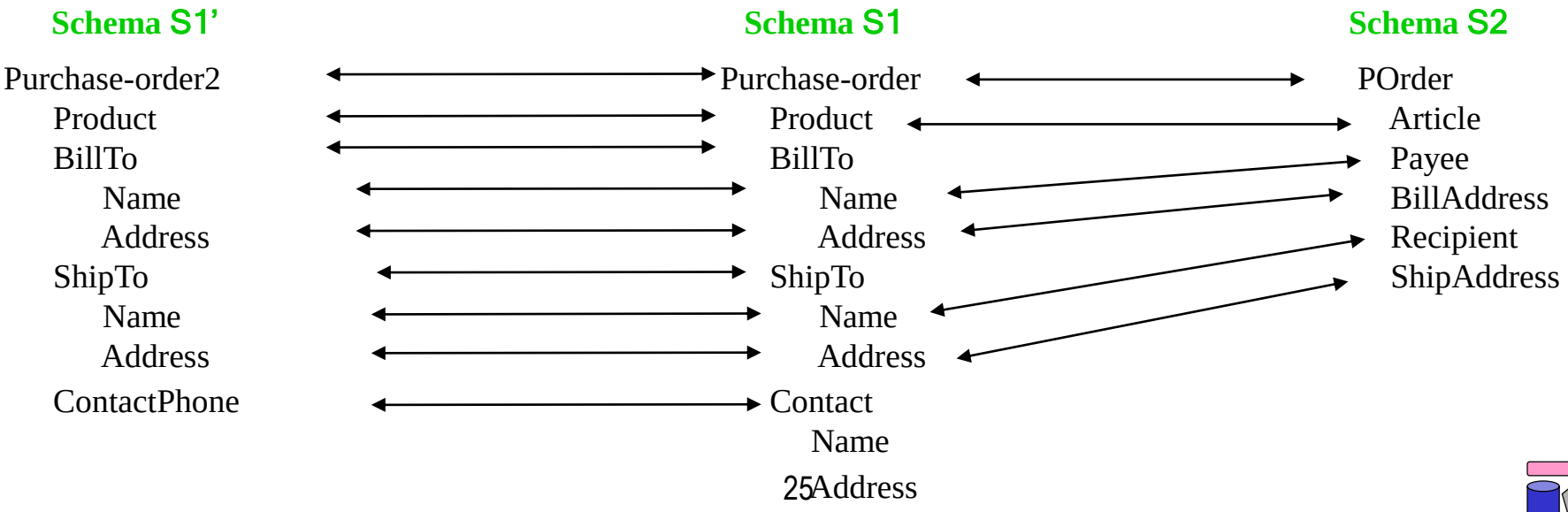
S2 elements	*S1 elements*	*Sim*
...City	...shipToCity	

Threshold(0.5)

S2 elements	*S1 elements*	*Sim*
...City	...shipToCity	
...City	...shipToStreet	

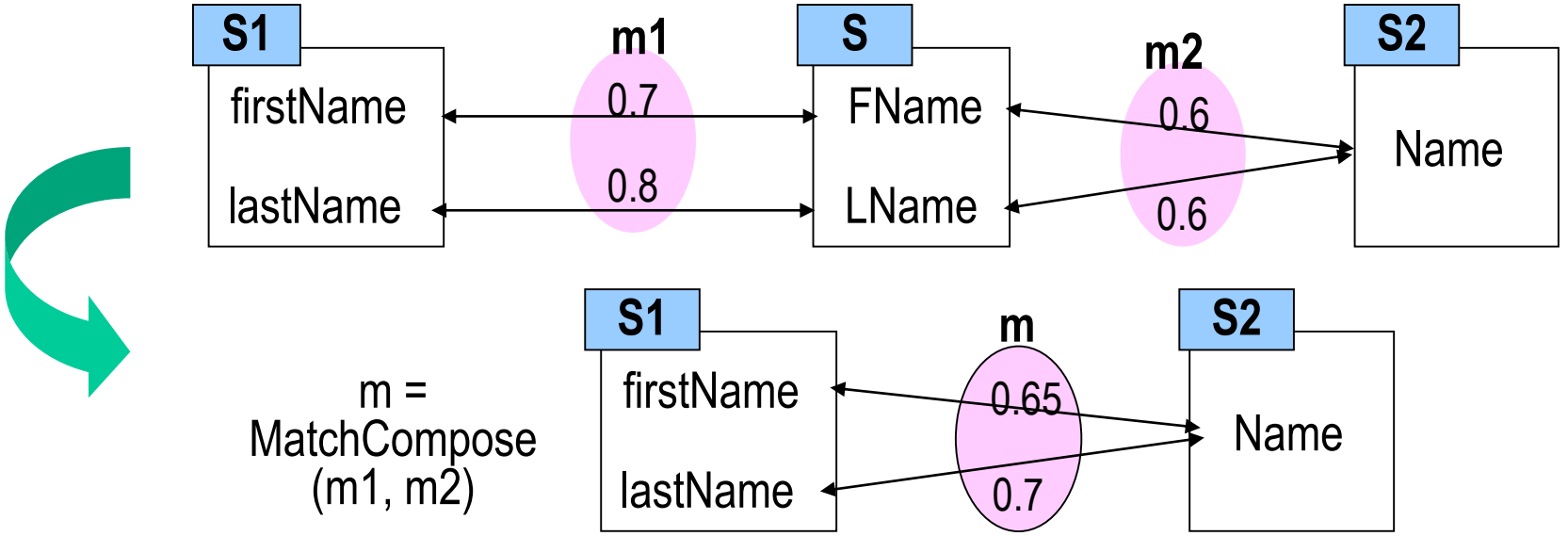
Wiederverwendung (Reuse)

- Nutzung von Hilfsquellen
  - Nutzerspezifizierte Synonymtabellen, allgemeine/domänenspezische Vokabulare, Wörterbücher, gemeinsame Ontologien
- Nutzung bereits bestätigter Match-Ergebnisse für ähnliche Match-Probleme
  - Speichern von Schemas und Mappings in Repository
  - Besonders vorteilhaft für Abgleich neuer Schema-Versionen (Schema-Evolution)
- Beispiel: Wiederverwendung des vorhandenen (bestätigten) Mappings S1—S2 zur Lösung des neuen Match-Problems S1'—S2

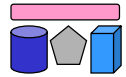


Wiederverwendung von Mappings

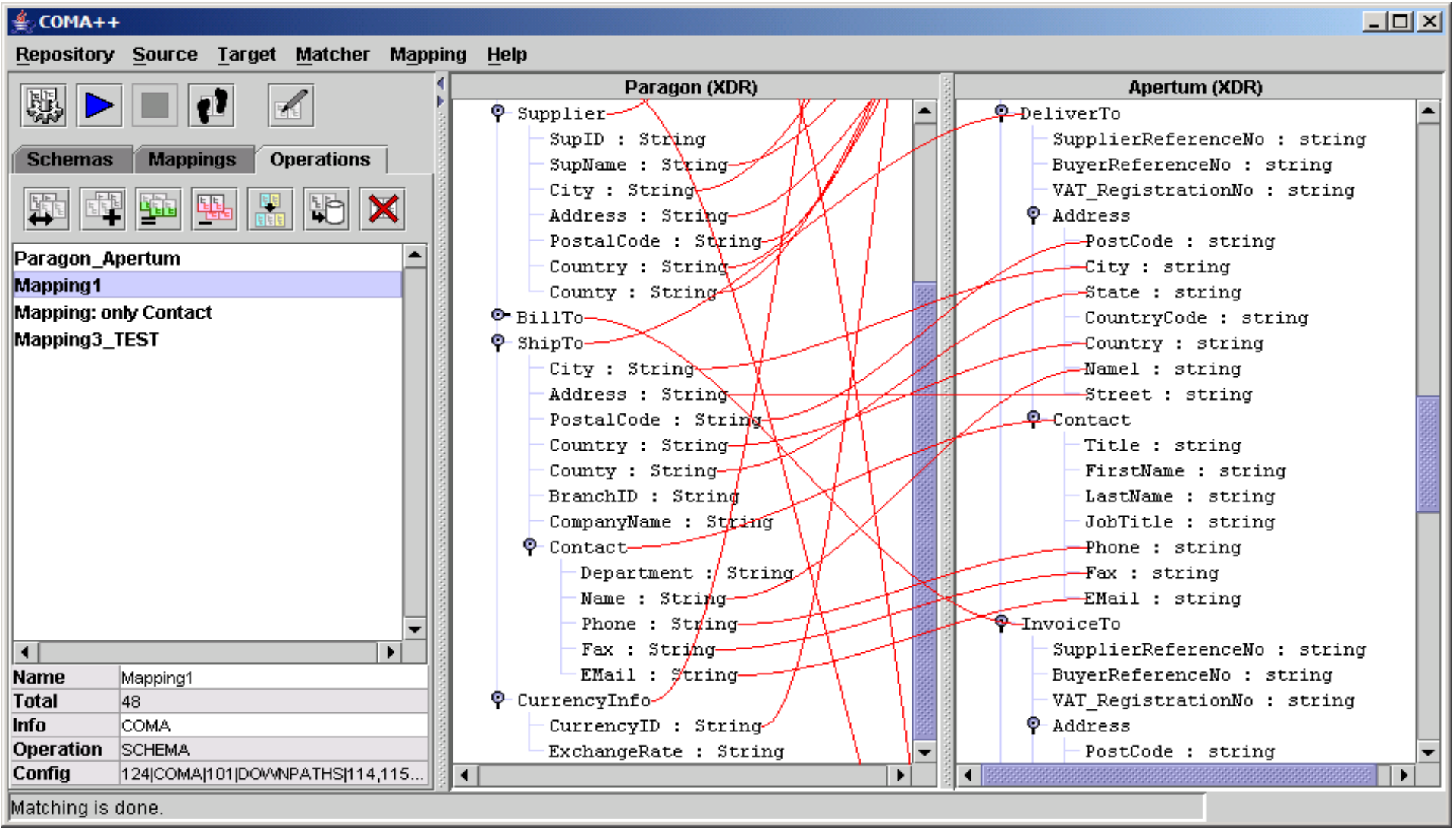
- MatchCompose-Operation: Ähnlichkeit/Match als transitive Beziehung

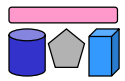


- Wiederverwendungsmöglichkeiten für neues Match-Problem S1-S2
  - Direkte Mappings S1-S2
  - Mapping-Pfade (S1-S3-S2, S2-S4-S5-S1, …)
  - Nutzung ähnlicher Mappings, z.B. mit unterschiedlichen Schemaversionen



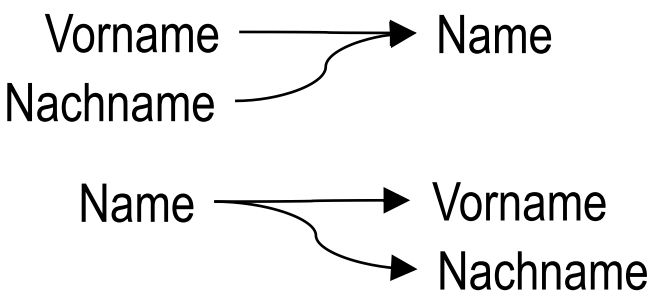
COMA++: Nutzerschnittstelle (GUI)





Schema Matching: Erweiterungen

- 1:n und n:1 Matches
  - Viele Kombinationsmöglichkeiten
  - Viele Funktionen denkbar
- Globales Matching
  - Matche nicht nur einzelne Attribute (oder Attributmengen)
  - Sondern komplette Tabellen oder komplette Schemata
  - Beispiele:
    - Stable Marriage Problem (nächste Folie)
    - Maximum Weighted Matching



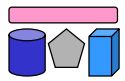
Stable Marriage: Beispiel

- Gegeben: n Frauen und m Männer
  - Frauen = Attribute in Schema A, Männer = Attribute in Schema B
- Monogamie: Je eine Frau kann nur mit je einem Mann verheiratet sein
  - Nur 1:1 Matches
- Jede Frau hat eine Rangliste der Männer und umgekehrt
  - Attribut-Ähnlichkeit gemäß eines der vorigen Verfahren
- Gesucht: Paarung (globales Matching), so dass niemals gilt
  - f1 heiratet m1, f2 heiratet m2, aber f1 bevorzugt m2 und m2 bevorzugt f1 (Instabil!)

Männer (1-4)	Frauen (A-D)
1: B, D, A, C	A: 2, 1, 4, 3
2: C, A, D, B	B: 4, 3, 1, 2
3: B, C, A, D	C: 1, 4, 3, 2
4: D, A, C, B	D: 2, 1, 4, 3

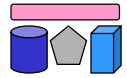
Antrag			Ergebnis

David Toth: "The Stable Marriage Problem: More Marital Happiness than Reality TV“, 2003

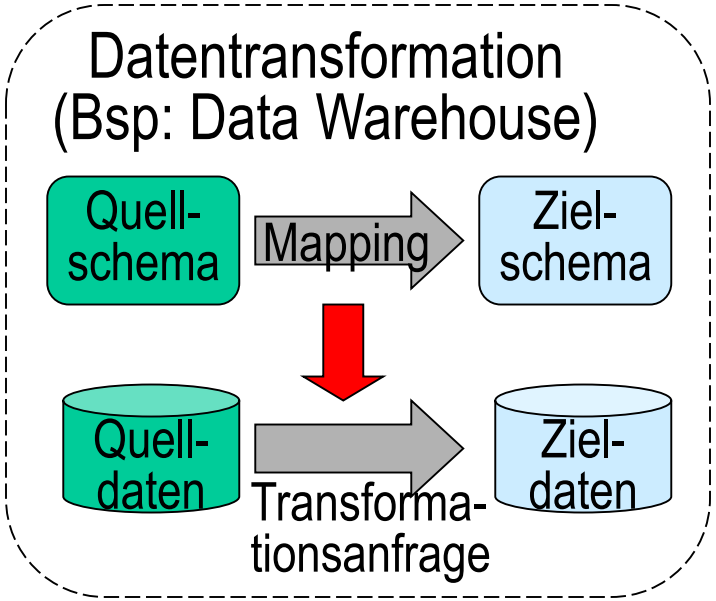


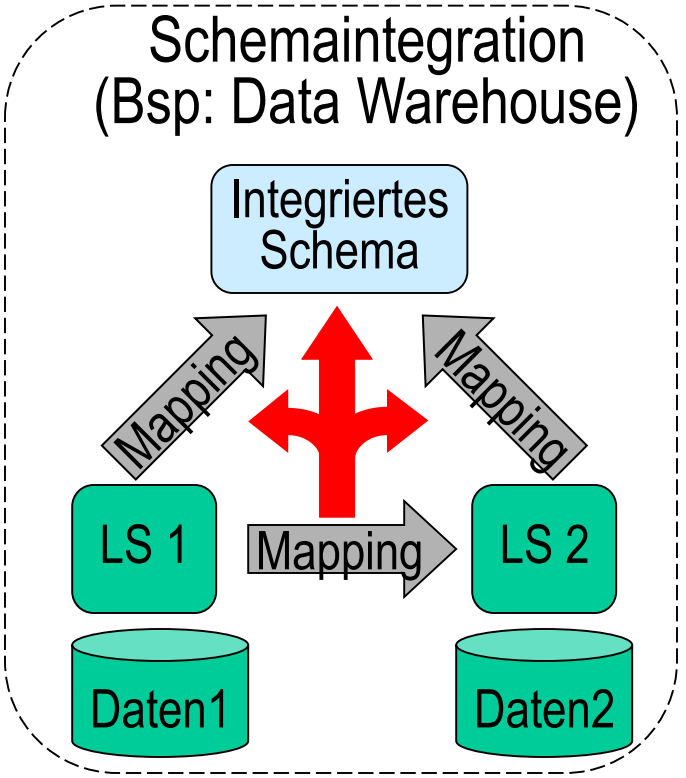
Schema Mapping: Definition(en)

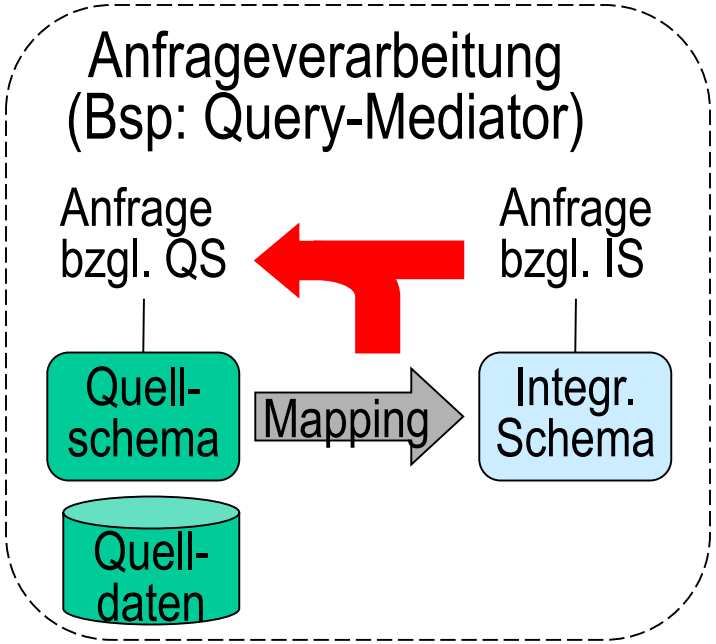
- (Inter-Schema) Korrespondenz
  - Eine Zuordnung eines oder mehrerer Elemente eines (Quell-) Schemas zu einem oder mehreren Elementen eines anderen (Ziel-) Schemas
  - Auch: Value-correspondence
- (High-level) Mapping (= Schema-Matching-Ergebnis)
  - Eine Menge von Korrespondenzen zwischen zwei Schemas.
- (Low-Level) Logisches Mapping
  - Logische Übersetzung eines oder mehrerer Mappings, die
    - den Integritätsbedingungen beider Schemas gehorcht und
    - die Intention des Nutzers wiederspiegelt.
- Übersetzung (bzw. Interpretation)
  - Übersetzung eines Mappings in ein oder mehrere logische Mappings
  - Übersetzung eines logischen Mappings in eine Transformationsanfrage
    - Anfrage in einer Anfragesprache (z.B. SQL), die Daten des Quellschemas in die Struktur des Zielschemas überführt

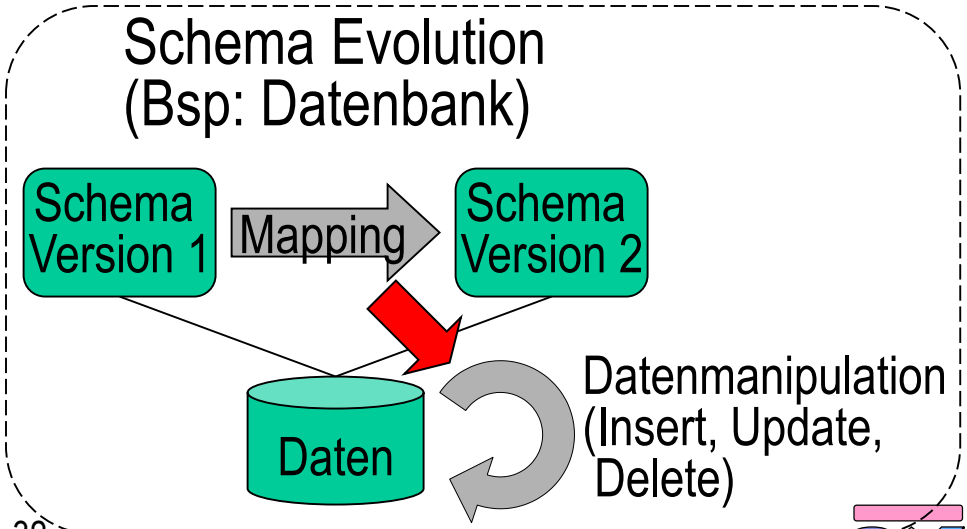


Anwendungen



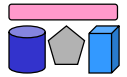






Problemstellung

- Gegeben 1: Zwei Schemata S1 und S2
  - Unabhängig voneinander erzeugt (relational, geschachtelt, ...)
  - Mit Integritätsbedingungen (Schlüssel/Fremdschlüssel)
  - Stellen teilweise unterschiedliche Daten dar
- Gegeben 2: Eine Menge von Korrespondenzen zwischen den Schemata
  - Ergebnis des Schema Matching
- Gesucht: Anfrage oder Sequenz von Anfragen, die Daten des einen in Daten des anderen Schemas transformiert, wobei
  - Semantik des Quellschemas erhalten bleibt,
  - Integritätsbedingungen des Zielschemas berücksichtigt werden,
  - und möglichst alle Korrespondenzen berücksichtigt werden



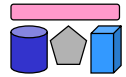
Überwindung der schematischen Heterogenität

- Modellierung
  - Relation vs. Attribut
  - Attribut vs. Wert
  - Relation vs. Wert

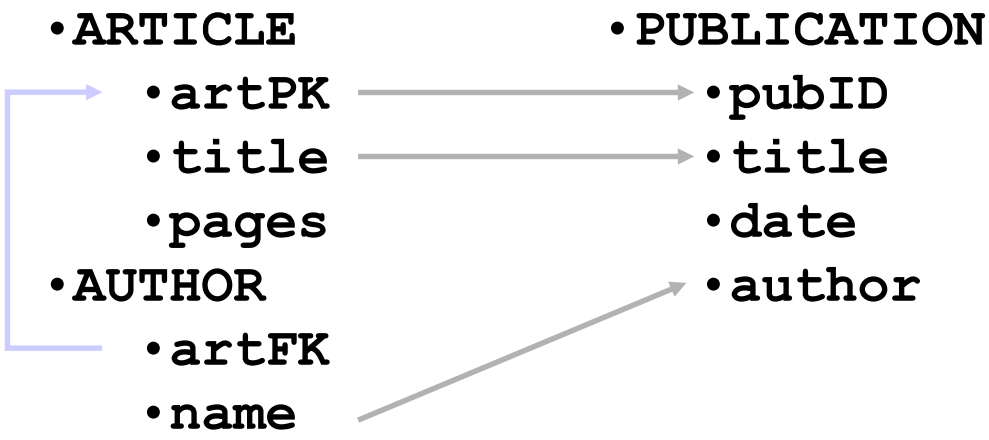
High-order Mappings
- aktuelles Forschungsgebiet
- viele Fragen offen, z.B. Transformationssprache

- Benennung
  - Relationen
  - Attribute
- Normalisiert vs. Denormalisiert
- Geschachtelt vs. Fremdschlüssel
- Geschachtelt vs. Flach

Beispiel: CLIO-Algorithmus
- IBM Forschungsprojekt



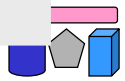
Schema Mapping am Beispiel



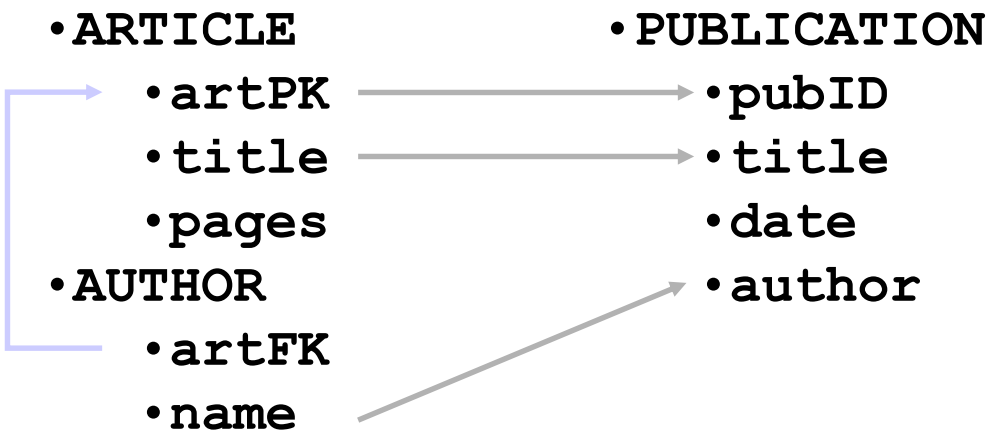

```
CREATE VIEW publication1 (…) AS
SELECT artPK AS pubID
       title AS title
       null AS date
       null AS author
FROM   ARTICLE
UNION
SELECT null AS pubID
       null AS title
       null AS date
       name AS author
FROM   AUTHOR
```

```
CREATE VIEW publication2 (…) AS
SELECT  artPK AS pubID
        title AS title
        null AS date
        name AS author
FROM    ARTICLE JOIN  AUTHOR
ON      ARTICLE.artPK = AUTHOR.artFK
```

```
CREATE VIEW publication2b (…) AS
...
FROM    ARTICLE FULL OUTER JOIN  AUTHOR
ON      ARTICLE.artPK = AUTHOR.artFK
```



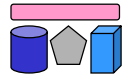
Schema Mapping am Beispiel (2)



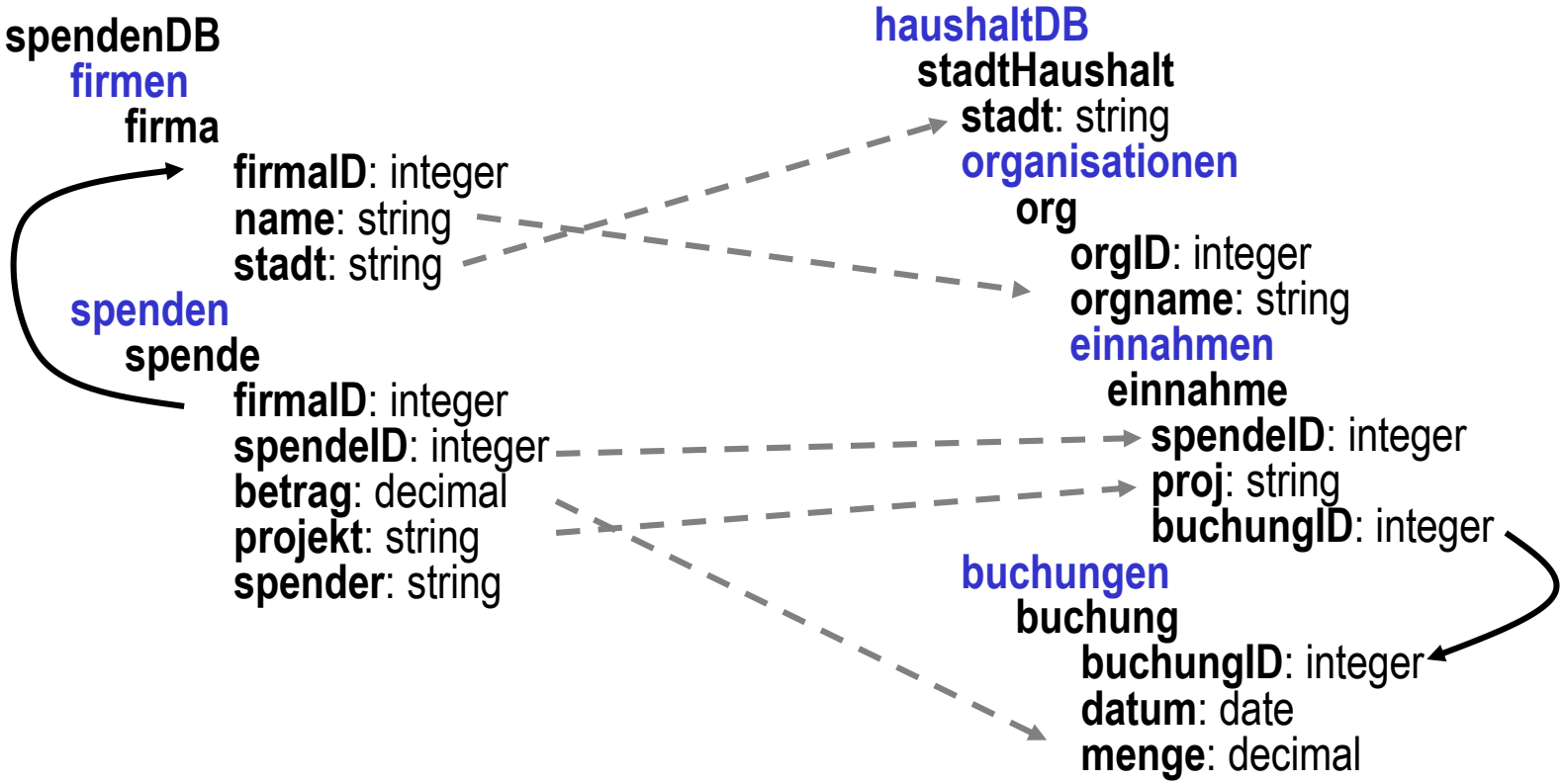

```
CREATE VIEW publication3 (…) AS
SELECT  artPK AS pubID
        title AS title
        null AS date
        name AS author
FROM    ARTICLE LEFT OUTER JOIN  AUTHOR
ON      ARTICLE.artPK = AUTHOR.artFK
UNION
SELECT  skolem(name) AS pubID,
        null AS title,
        null AS date,
        name AS author
FROM   AUTHOR
WHERE artFK IS NULL
```

Skolemfunktion

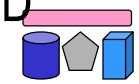
- Funktion zum “Erfinden von Werten”
- Input: n Werte
- Output: bezgl. Input eindeutiger Wert
- Beispiel: Konkatenation aller Inputwerte als String



Schema Mapping an komplexerem Beispiel

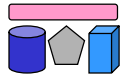


- Erzeuge für jede firma in spendenDB eine org in haushaltDB
- Erzeuge für jede spende in spendenDB eine einnahme in haushaltDB
- Erzeuge für jede spendenDB.firma.stadt ein haushaltDB.stadthaushalt mit gleichem Namen
- Gruppiere jede firma unter den entsprechenden stadtHaushalt
- Erzeuge für jede spende in spendenDB eine buchung in haushaltDB
- Konstruiere für jede einnahme einen Verweis zu buchung mittels künstlicher buchungID



Schema Mapping: Probleme

- Erfinden von Schlüsseln, Skolemfunktionen
- Attributkorrespondenzen verraten nicht, wie Relationen innerhalb eines Schemas zusammenhängen
  - “Nicht nur Join-Werte raten, auch die Joins selber raten”
  - Bei XML: Schachtelung bestimmt den „Join“
- Beachtung von Integritätsconstraints
- Gruppierungen
- Datentypen
- Kardinalitäten, z.B. [0,1] vs. [1,1]
- “Unmapped”-Attribute
- n:1-Korrespondenzen, z.B. concat(Vorname, Nachname) = Name
- ...



Clio-Projekt

IBM Almaden Research Center

IBM Research

Schema Mapping Management System

http://www.almaden.ibm.com/cs/projects/criollo/

Overview

Enterprise databases cover hundreds of tables with thousands of attributes in complex and disparate structures. As many cover the same domain, the need to integrate them for more insight and a broader scope is apparent. To overcome structural heterogeneities, users must define mappings from one or more source schemas to a target schema.

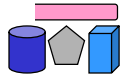
Our schema mapping management system, Clio, is a semiautomatic tool that helps users define such mappings. Clio then also interprets these mappings to construct a set of database queries that transform and integrate source data to conform to the target schema. Such queries can be used to populate data warehouses or to define views and virtual tables in federated database environments. Source and target can be any combination of relational databases and XML data.

Highlights:

- XML -> XML mapping (XML views on Web data, ...)
- Relational -> XML mapping (Web-publishing of legacy da
- XML -> Relational mapping (XML shredding into RDBMS, .
- Relational -> Relational mapping (Data warehousing, ...)
- Full usage of source and target data constraints
- Discovery of data constraints
- User friendly interface
- Dynamic interpretation of user input
- Incremental generation of transformation queries
- Intelligent suggestions of likely correspondences
- Merging source collections with automatic join-path selec
- Splitting source collections with ID invention

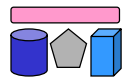
Allgemeines Vorgehen

1. Entdeckung von Intra-Schema-Assoziationen
2. Entdeckung von logischen Inter-Schema-Mappings
3. Erzeugung der Trans-formationsanfragen

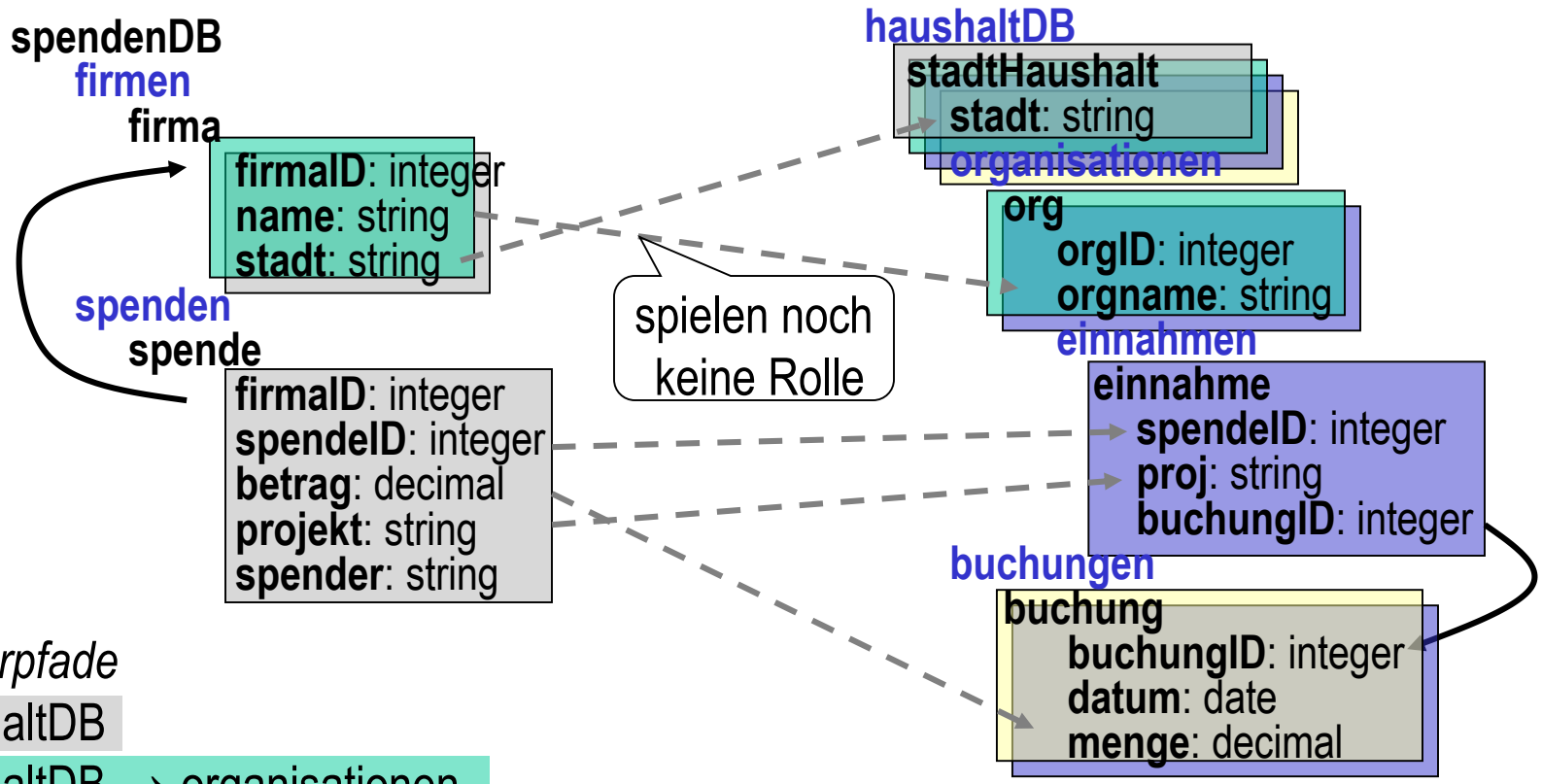


1. Intra-Schema Assoziationen

- Zunächst: Ignorieren der Attributkorrespondenzen
- Intra-Schema Assoziationen ergeben sich aus
  - Attribute gehören zur selben Relation
  - Attribute haben dasselbe unmittelbare Elternelement (XML)
  - Attribute haben einen gemeinsamen Vorfahren (XML)
  - Tabellen oder Elemente sind durch PK/FK verbunden (RM)
- Konstruktion von Blöcken zusammenhängender Daten
  - Zunächst alle Relationen und alle komplexen XML Elemente mit deren komplexen Eltern
  - Erweiterung mit allen PK/FK Beziehungen
  - Genannt: Logische Relationen (LR)
- Repräsentation als Pfade
  - Primärpfade
  - Erweiterte Pfade



1. Intra-Schema Assoziationen (2)



Primärpfade

haushaltDB

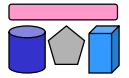
haushaltDB → organisationen

haushaltDB → organisationen → einnahmen

haushaltDB → buchungen

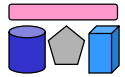
Erweiterte Pfade

haushaltDB → organisationen → einnahmen → buchungen

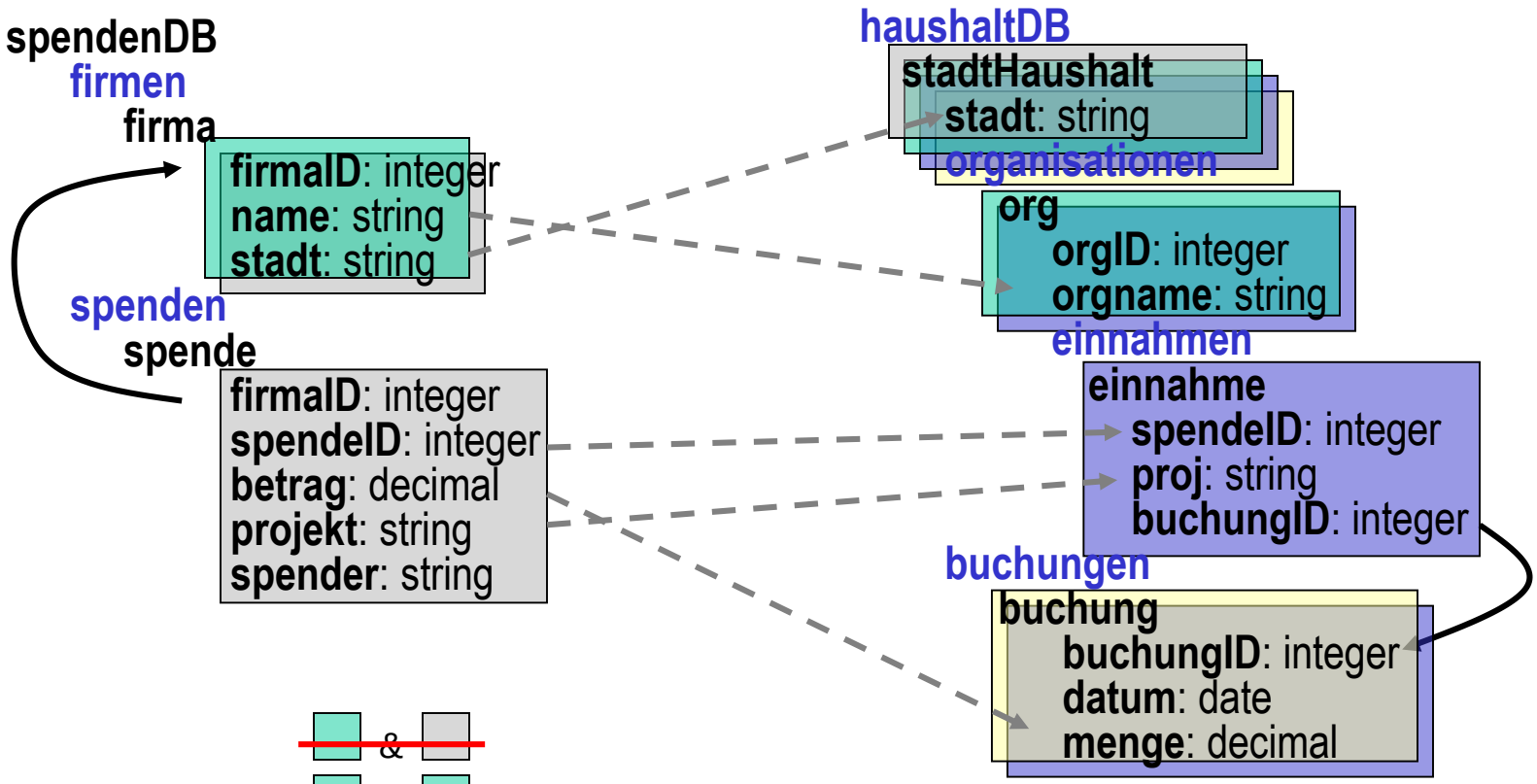


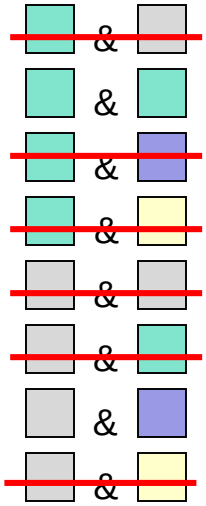
2. Logische Inter-Schema-Mappings

- Finde sinnvolle Paare von logischen Relationen in beiden Schemata
  - Nutzung der Attribut-Korrespondenzen
- Forderungen
  - Die logischen Relationen (LR) eines Paares dürfen keine ausgehenden oder eingehenden Mappings zu anderen LR haben
  - Intuition: Daten dürfen nicht auf unzusammenhängende LR‘s verteilt bzw. aus solchen gebildet werden, da sonst die Semantik des Quellschemas verloren gehen würde. „Damit zusammen bleibt, was zusammen gehört“
  - Starke Einschränkung der Menge möglicher Interpretationen
  - (Sinnvolle) Heuristik
- Vorgehen
  - Bilde alle Paare von LR
  - Streiche alle Paare, für die die Forderung nicht erfüllt ist



2. Logische Inter-Schema-Mappings (2)

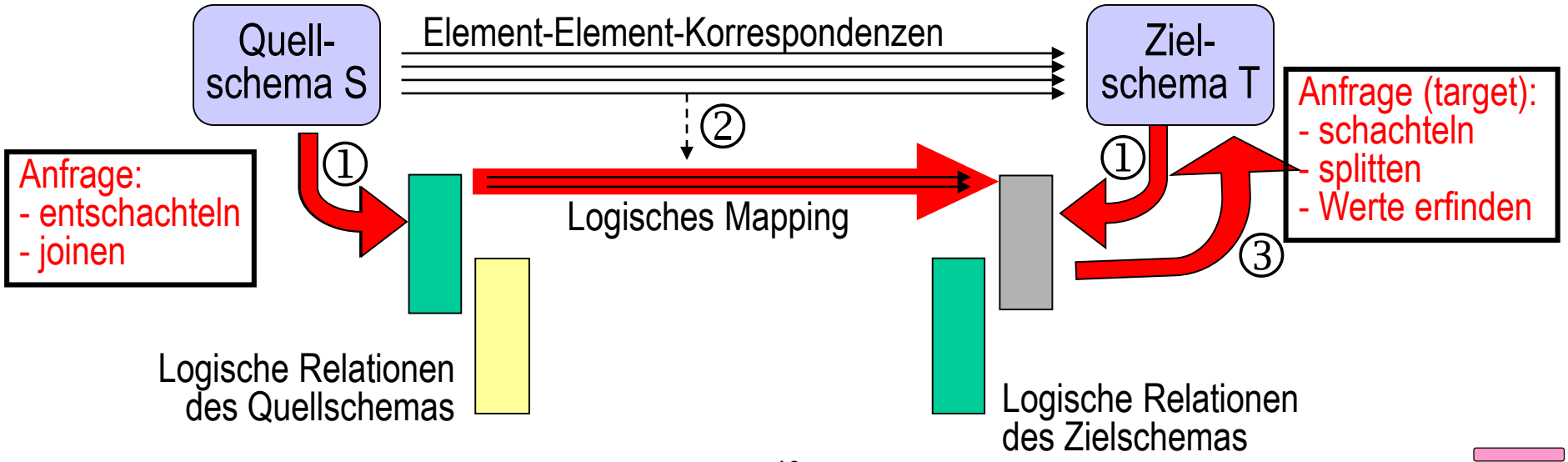




1. Nutzer “malt” Korrespondenzen
2. Prüfung der ausgehenden Korrespondenzen:
   “Finden alle ein Ziel in der betreffenden LR?”
3. Prüfung der eingehenden Korrespondenzen:
   “Stammen alle aus der betreffenden LR?”
4. Nutzer wählt logische Mappings (Interpretationen) aus

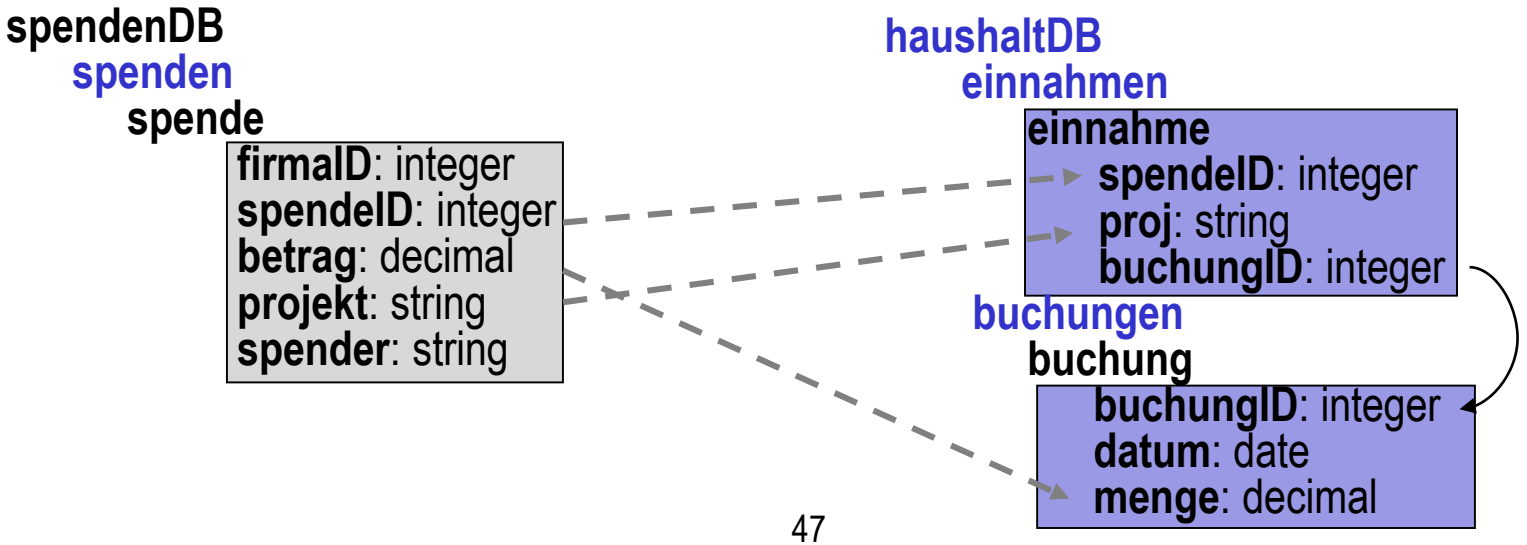
3. Erzeugung der Transformationsanfragen

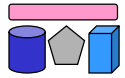
- Paare werden separat behandelt
  - pro Paar eine Transformationsanfrage
- Werte müssen erfunden werden, um
  - NOT NULL constraints einzuhalten
  - PK/FK Beziehungen zu bewahren
- Schachtelungen bewahren
  - Umgruppierung von Werten



3. Transf.-anfragen (2): Erfinden neuer Werte

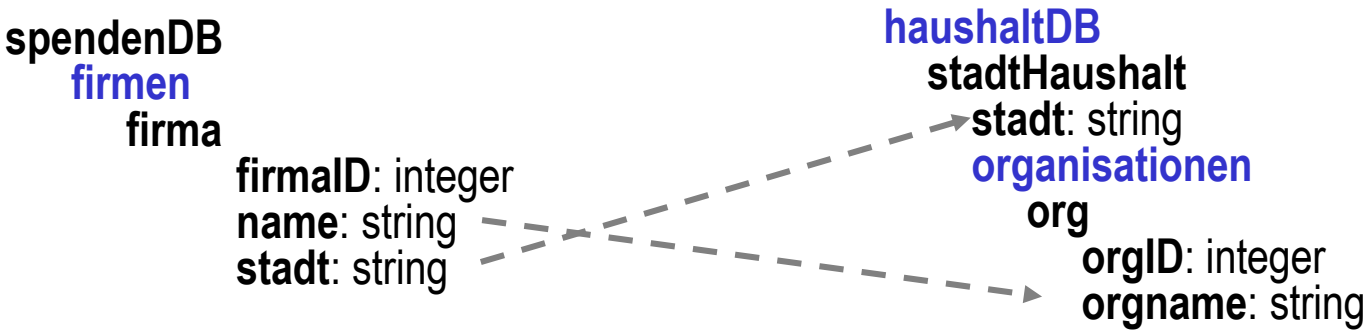
- LR-Paar: spenden – einnahmen+ buchungen
- buchungID hat keine Korrespondenz
  - Assoziationen würden verloren gehen → neue ID erfinden
  - Skolemfunktion basierend auf allen Werten des Mappings dieser logischen Relation
  - buchungID = Skolem(spendeID, projekt, betrag)
  - Gleiche Funktion für Schlüssel und Fremdschlüssel verwenden





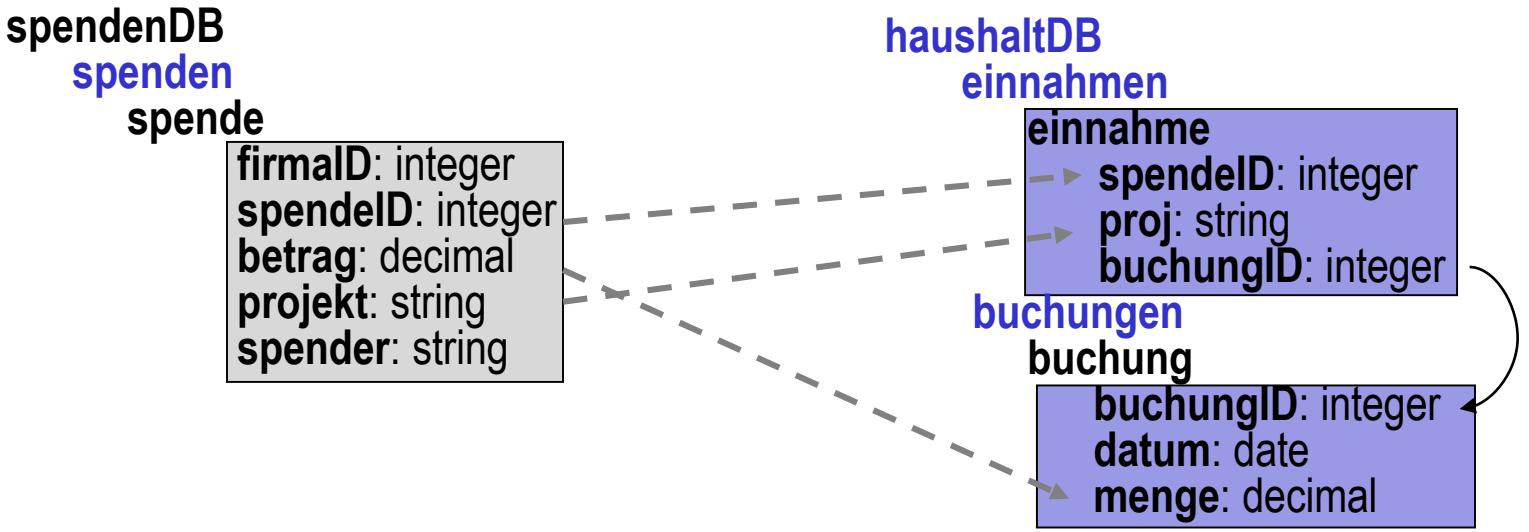
3. Transf.-anfragen (3): Gruppierung

- Schachtelung verlangt Gruppierung der Werte aus firmen nach stadt
- Virtuelles Gruppierungsattribut G durch Skolemfunktion generieren
  - Basierend auf allen Werten hierarchisch über der aktuellen LR
  - Jedes Tupel haushaltDB erhält Gruppenwert Sk(stadt)
  - Jedes weitere Tupel aus firmen mit gleichen Werten für stadt errechnet gleichen Gruppenwert und wird gleich geschachtelt
- Evtl. mehrfache Gruppierung nötig
  - Beispiel: firma hat zusätzliches Attribut land; haushaltDB gruppiert stadtHaushalte nach zusätzlicher Stufe land

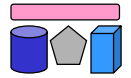


3. Transf.-anfragen (4): Regeldarstellung

for x in spenden
let a = x.spende.spendeID, b = x.spende.projekt, c = x.spende.betrag
return <einnahme = <spendeID = a, proj = b, buchungID = Sk(a,b,c)>> **in** einnahmen,
<buchung = <buchingID = Sk(a,b,c), datum = null, menge = c>> **in** buchungen



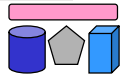
- Übersetzung der Regeln in Datenmodell-abhängige Anfragesprache
  - Relational → Relational: SQL
  - Relational → XML: SQLX (Schachtelung und tagging des Ergebnisses)
  - XML → Relational: XQuery oder XSLT (Tags weglassen)
  - XML → XML: XQuery oder XSLT



Beispiel: XQuery

for x in spenden
let a = x.spende.spendeID, b = x.spende.projekt, c = x.spende.betrag
return <einnahme = <spendeID = a, proj = b, buchungID = Sk(a,b,c)>> **in** einnahmen,
<buchung = <buchingID = Sk(a,b,c), datum = null, menge = c>> **in** buchungen

```
LET $doc0 := document("input XML file")
RETURN <haushaltDB>   {
   distinct-values (
   FOR   $x0 IN $doc0/spendenDB/spende
   RETURN      <einnahme>
         <spendeID>  {  $x0/spendeID/text() }  </spendeID>
         <proj>  {  $x0/projekt/text() }  </proj>
         <buchungID>  {  "Sk32(", $x0/betrag/text(), ", ", $x0/spendeID/text(), ", ", $x0/projekt/text(), ")" }
         </buchungID>
      </einnahme> )   }
   {    distinct-values (
   FOR     $x0 IN $doc0/spendenDB/spende
   RETURN      <buchung>
         <buchungID>  {  "Sk32(", $x0/betrag/text(), ", ", $x0/spendeID/text(), ", ", $x0/projekt/text(), ")" }
         </buchungID>
         <menge>  {  $x0/betrag/text() }  </menge>
      </buchung> )   }
</haushaltDB>
```

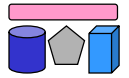



Beispiel: SQL

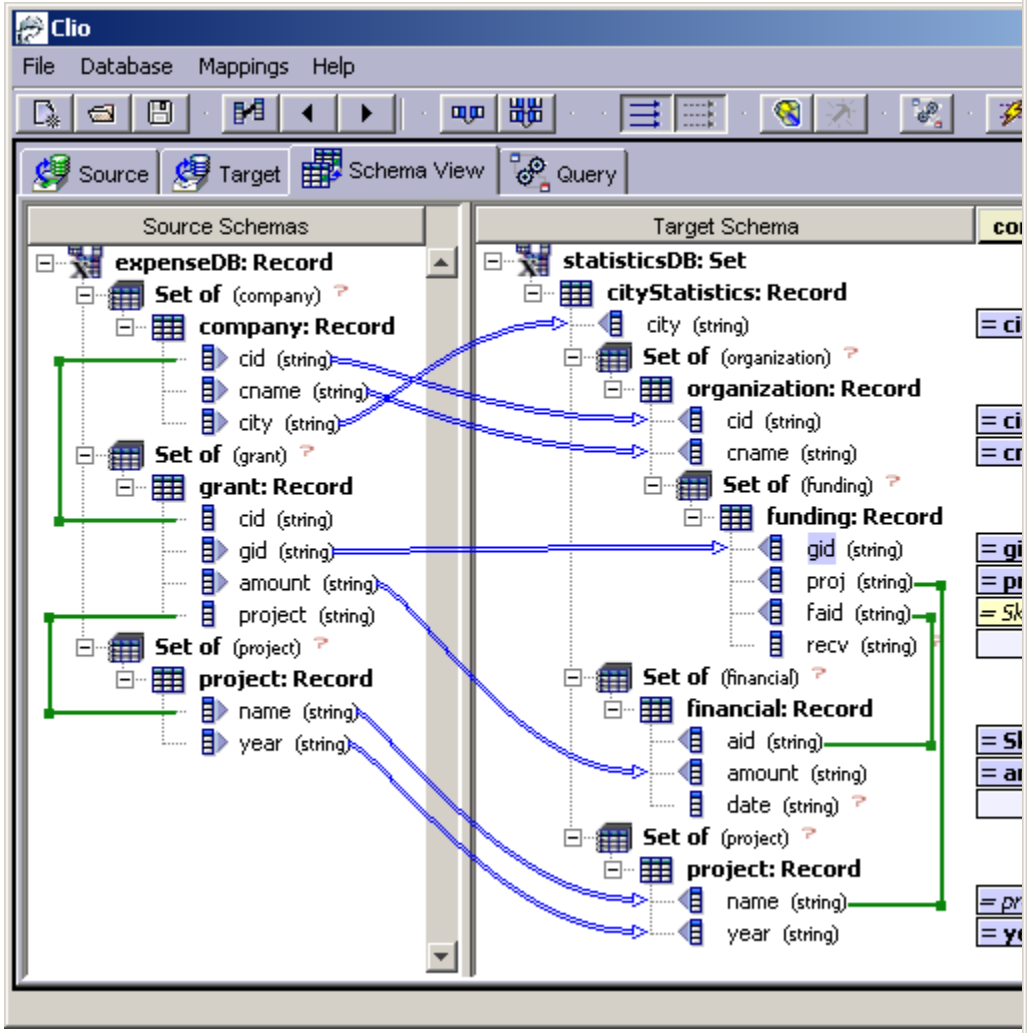
for x in spenden
let a = x.spende.spendeID, b = x.spende.projekt, c = x.spende.betrag
return <einnahme = <spendeID = a, proj = b, buchungID = Sk(a,b,c)>> **in** einnahmen,
<buchung = <buchingID = Sk(a,b,c), datum = null, menge = c>> **in** buchungen

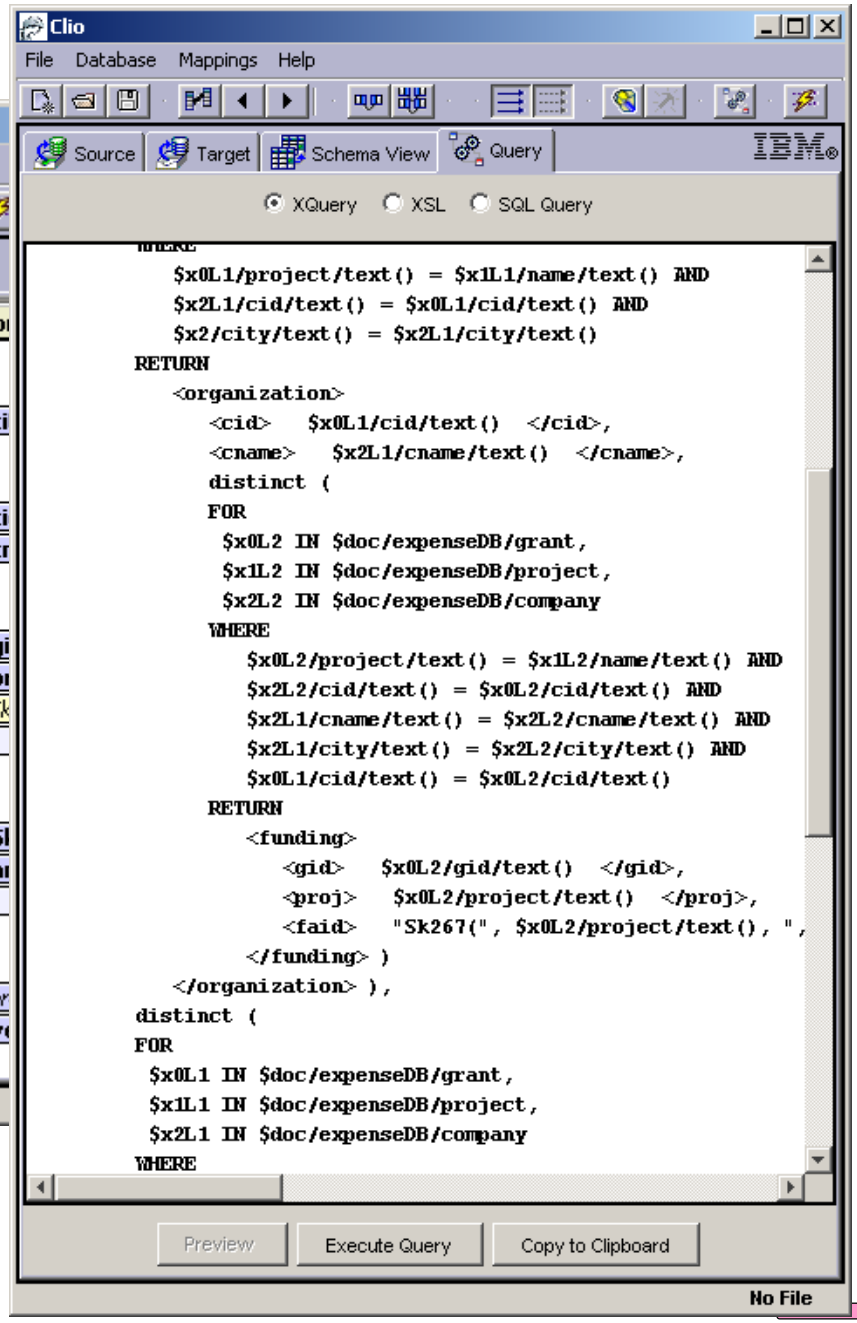
```
CREATE VIEW einnahme AS
SELECT
        x0.spendeID AS spendeID,
        x0.projekt AS proj,
        RTRIM('Sk32(` CHAR(x0.betrag) || ','  || (CHAR(x0.spendeID) || ','  || CHAR(x0.projekt) || ')')
        AS buchungID
FROM   spendenDB.spende x0


CREATE VIEW buchung AS
SELECT
        RTRIM('Sk32(' || CHAR(x0.betrag) || ','  || CHAR(x0.spendeID) || ','  || CHAR(x0.projekt) || ')')
        AS buchungID,
        x0.betrag AS menge
FROM   spendenDB.spende x0
```

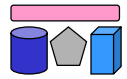
Clio: Ergebnis





Model Management

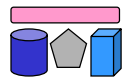
- Generischer Ansatz für Verarbeitung von Schemata und Mappings
- Infrastruktur zur Manipulation von Metadaten (Schemata und Mappings)
  - Generische (d.h. datenmodellunabhängige) Repräsentation von Metadatenstrukturen (Schemata) und ihren Beziehungen (Mappings)
  - Algebra mit Operatoren zur Datenmanipulation
  - High-Level-Definition für Metadaten-intensive Aufgaben
- Anwendungsgebiete
  - Schemaintegration
  - Schema Evolution
  - Datenmigration
  - ...
- Nicht nur für “klassischen Datenbankbereich”, sondern u.a. auch für Software Engineering
  - Kopplung von Web-Services (auch hier: Matching, Evolution, ...)



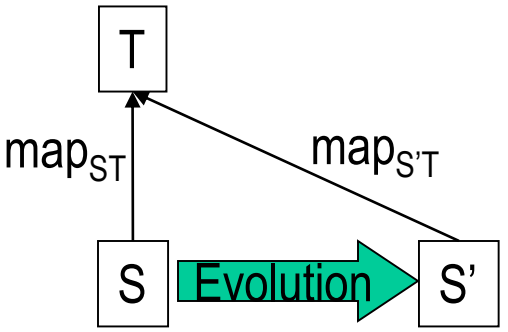
Model Management: Strukturen & Operatoren

- Model: Komplexe Informationsstruktur
  - DB-Schema, XML-Schema, OO-Klassenschnittstelle, UML-Diagramm, etc.
- Mapping: Repräsentation einer Transformation von Model A in Model B
  - zw. XML-Schemas, zw. ER-Diagramm+DB-Schema, Workflow-Skript

- Match: $M_A \times M_B \rightarrow map_{AB}$
  - Schema Matching
- Compose: $map_{AB} \times map_{BC} \rightarrow map_{AC}$
  - Komposition, d.h. Verkettung der Mappings
- Diff (Differenz): $M_A \times map_{AB} \rightarrow M'_A$
  - Ergebnis sind alle Teile von A, die nicht im Mapping erfasst sind
- ModelGen: $M_A \rightarrow M_B \times map_{AB}$
  - Generierung eines neuen Models
- Merge: $M_A \times M_B \times map_{AB} \rightarrow M_C \times map_{AC} \times map_{BC}$
  - Schema Integration



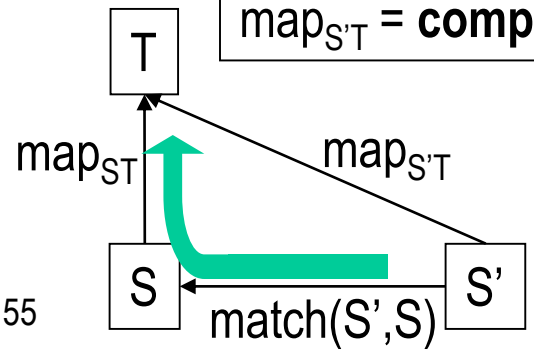
MM-Szenario: Schema Evolution

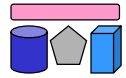


- Existierendes Mapping zwischen S und T
- Evolution (Veränderung) von Schema S zu S'
  - mögliche Gründe: neue Attribute, Umstrukturierung, ...
- Aufgabe: Anpassung des Mappings, d.h. Mapping zwischen S' und T

- Lösung 1 (ohne Model Management): Schema Matching S'-T
  - ggf. aufwändig für große Schemata, Nutzerüberprüfung nötig
  - keine Wiederverwendung der Informationen im Mapping zwischen S-T
- Lösung 2 (mit Model Management): Mapping-Komposition
  - Wiederverwendung bestehender Mappings (map_{ST})
  - Matching zwischen evolvierten Schemata (S'-S) i.Allg. viel einfacher als zwischen unabhängigen Schemata (S-T)

$map_{S'S}$ = **match** (S', S);
$map_{S'T}$ = **compose** ($map_{S'S}$, map_{ST})





Zusammenfassung

- Datenintegration erfordert Integration der Metadaten
  - Erzeugung eines globalen / integrierten Schemas → Schemaintegration
  - Identifikation semantischer Korrespondenzen zw. Schemata → Schema Matching
  - Erstellung von Transformationsanfragen → Schema Mapping
- Meist semi-automatische Lösungen, nicht vollautomatisch
  - Schemaintegration: strukturierte Vorgehensweise, aufbauend auf Schema-Matching-Ergebnissen
  - Schema Matching: Ähnlichkeitsberechnungen (u.a. Parameter/Schwellwerte setzen), manuelle Überprüfung der Ergebnisse
  - Schema Mapping: Tool-Unterstützung (u.a. Auswahl relevanter Korrespondenzen), nur einfache Mappings
- Model Management
  - Generische Manipulation von Metadaten