Schema Matching and Mapping

Jose Luis Ambite

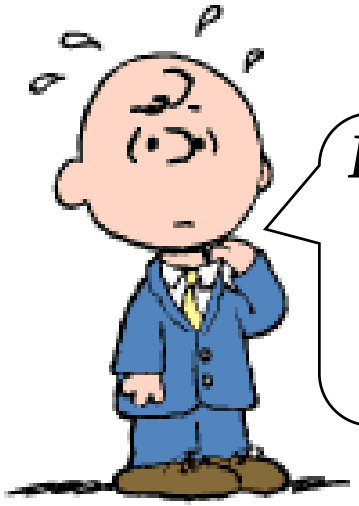
University of Southern California

[based on slides by Anhai Doan, by Doan, Halevy & Ives, and by Atzeni]

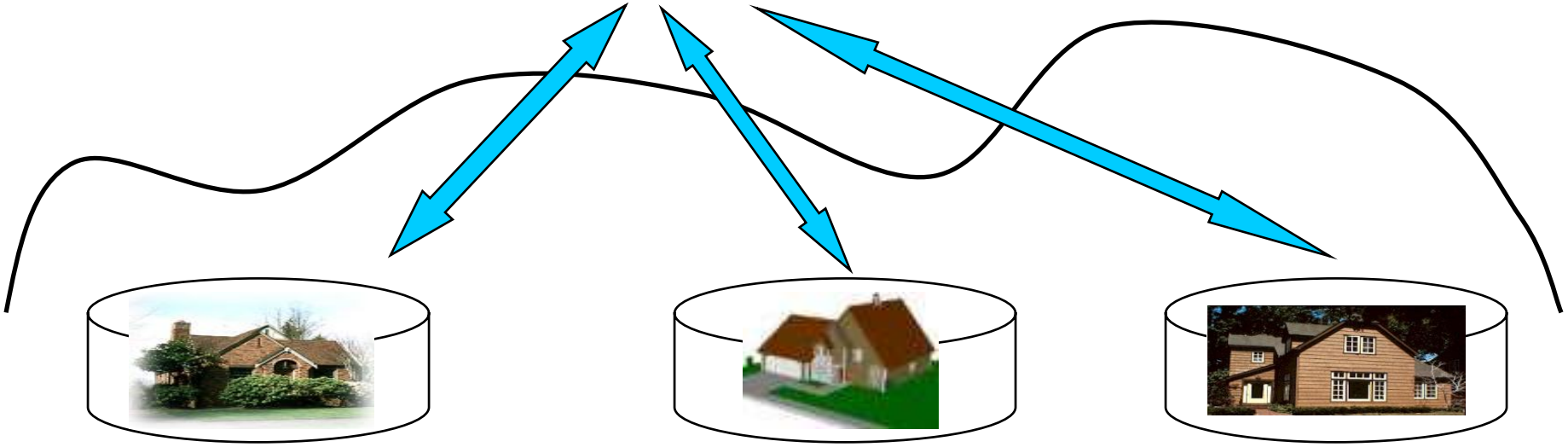
Outline

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

Motivation: Data Integration



Find houses with 2 bedrooms priced under 200K

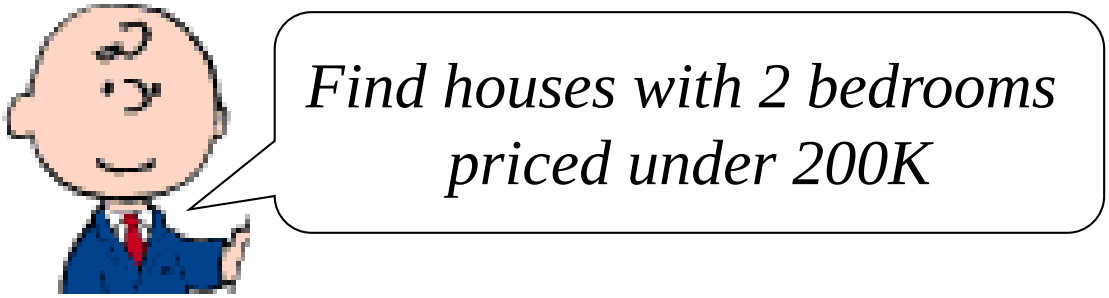


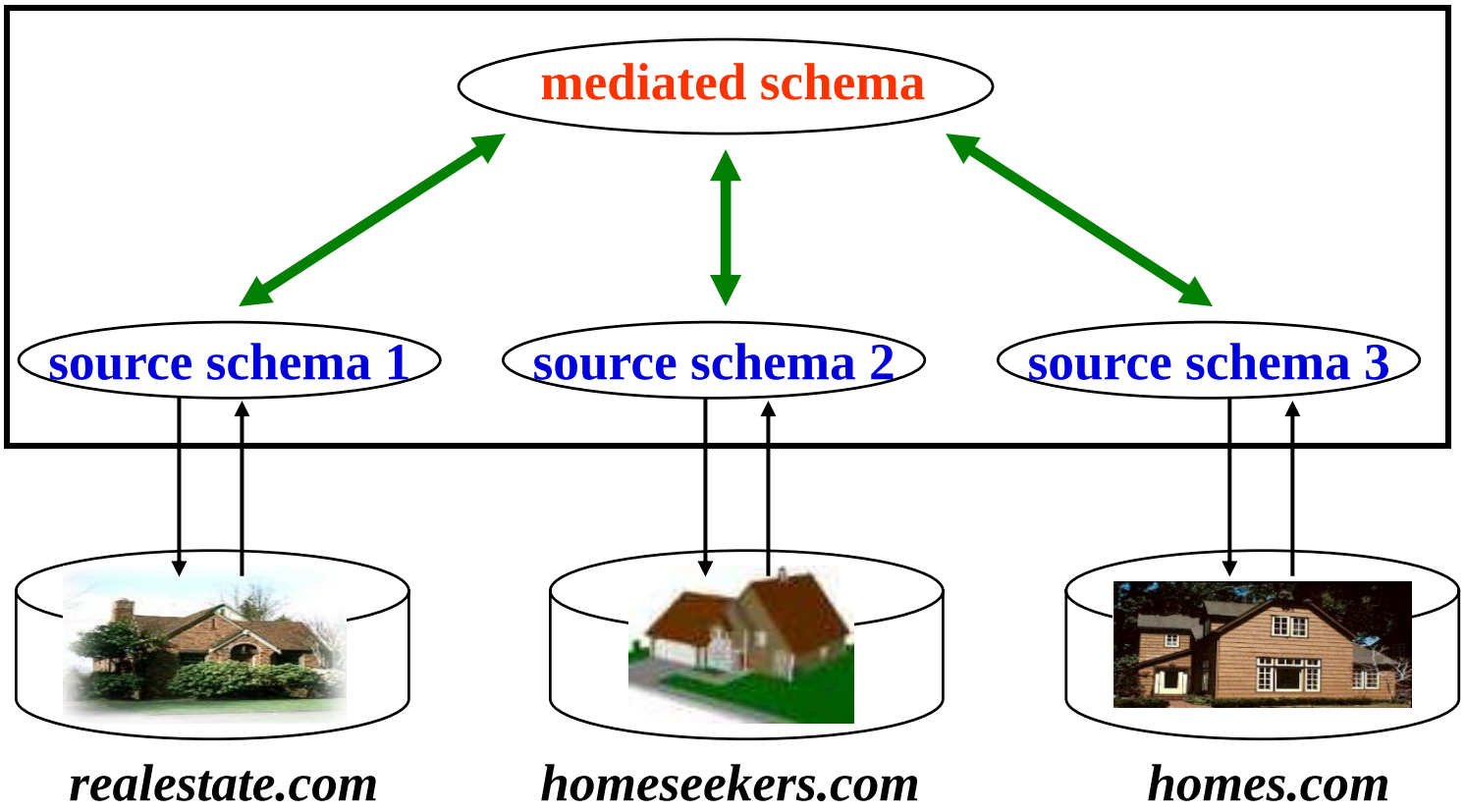
realestate.com

homeseekers.com

homes.com

Architecture of Data Integration System

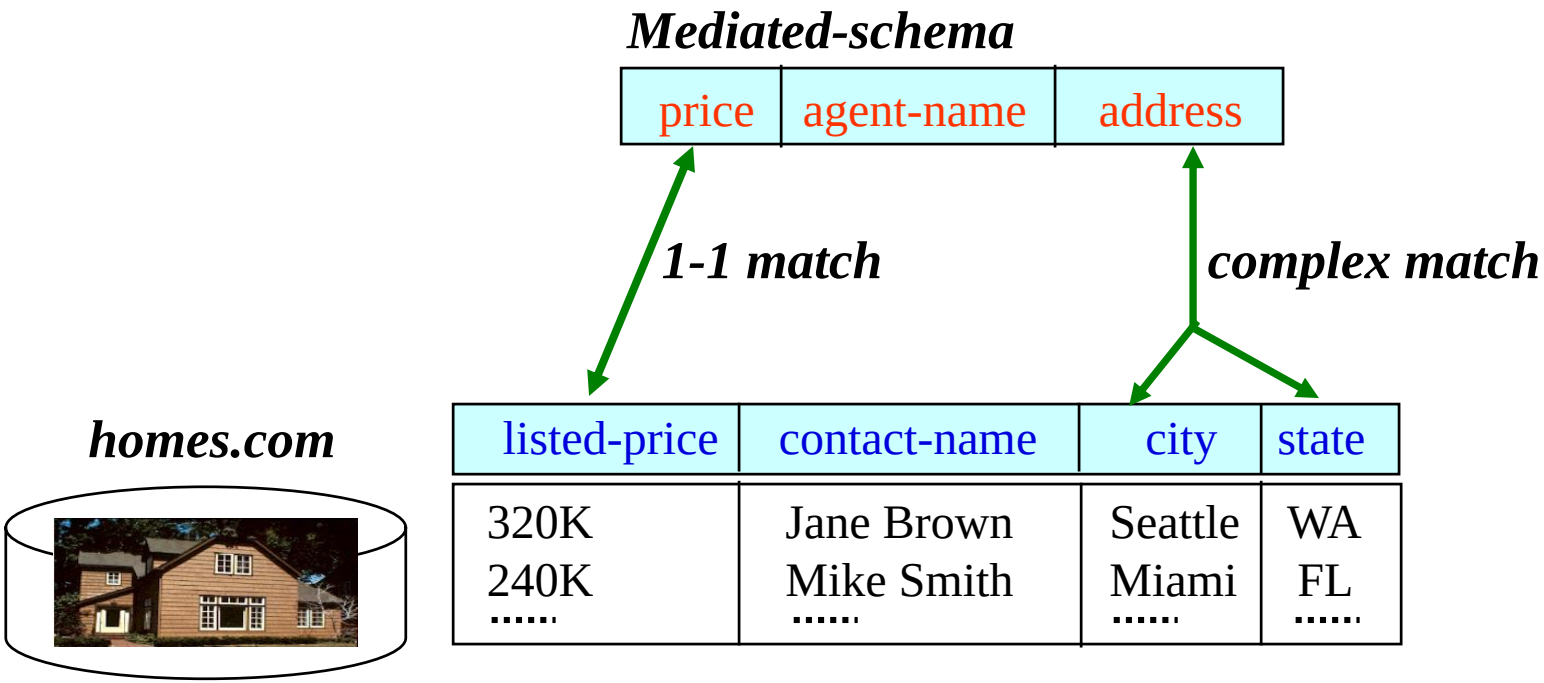




Schema Mappings

- Global-as-View (GAV): $\Phi s(X,Y) \rightarrow g(X)$
  - Mediator relation defined as a view over source relations
  - s1(ID,Title,Director,Year) ^ s3(ID,Review) → review(Title,Year,Review)
- Local-as-View (LAV): $s(X) \rightarrow \exists Y\ \Phi g(X,Y)$
  - Source relation defined as view over mediator relations
  - s1(id, title, year, director) → Movie(title,year,director,genre) ^ American(director) ^ year ≥1960 ^ genre = 'Comedy
- General (GLAV, st-tgd): $\Phi_S(X,Y) \rightarrow \exists Z\ \Psi_G(X,Z)$
  - s1(id, title, year, director) ^ s2(id, review) → Movie(title,year,director,genre) ^ American(director) ^ year ≥1990 ^ genre = 'Comedy' ^ hasReview(title, review)

Semantic Matches between Schemas



Exploit Multiple Types of Information

realestate.com

listed-price	contact-name	contact-phone	office	comments
$250K	James Smith	(305) 729 0831	(305) 616 1822	Fantastic house
$320K	Mike Doan	(617) 253 1429	(617) 112 2315	Great location

homes.com

sold-at	contact-agent	extra-info
$350K	(206) 634 9435	Beautiful yard
$230K	(617) 335 4243	Close to Seattle

- If use only names
  - contact-agent matches either contact-name or contact-phone
- If use only data values
  - contact-agent matches either contact-phone or office
- If use both names and data values
  - contact-agent matches contact-phone

Semantic Matches

- Semantic match: relates elements of schema S to elements of schema T without specifying in detail (SQL) the exact nature of relationship
- One-to-one matches
  - Movies.title = Items.name
  - Products.rating = Items.classification
- One-to-many matches
  - Items.price = Products.basePrice * (1 + Locations.taxRate)
  - m.address = concat(s1.city, s1.state)
- Many-to-many
  - s.address(s.num, s.name, s.citystate) ~~>
    m.address(concat(s.num, s.name) as s.street, split(citystate)$1 as city,
    split(citystate)$2 as state)

DVD-VENDOR
Movies(id, title, year)
Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
Locations(lid, name, taxRate)

AGGREGATOR
Items(name, releaseInfo, classification, price)

Schema Mappings

Schema matchings are not enough

→ schema mappings (GLAV rules)

Example:

DVD-VENDOR
Movies(id, title, year)
Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
Locations(lid, name, taxRate)

AGGREGATOR
Items(name, releaseInfo, classification, price)

```
CREATE VIEW AGGREGATOR AS
SELECT title AS name, releaseDate AS releaseInfo,
       rating AS classification,  basePrice * (1 + taxRate) AS price
FROM Movies m, Products p, Locations l
WHERE m.id = p.mid AND p.saleLocID = l.lid
```

Relationship between Schema Matching and Mapping

- To create source description: start by creating semantic matches, then elaborate matches into mappings
- Why start with semantic matches?
  - Easier to learn or to elicit from designers
    - price = basePrice * (1 + taxRate) based on domain knowledge, or function learning program
  - Break long process in the middle and allow designer to verify and correct the matches, reduce complexity of overall process
- Why the need to elaborate matches into mappings?
  - Matches specify functional relationships but they cannot be used to obtain data instances
  - Need schema mappings (SQL queries) to get data:
    - SELECT (basePrice * (1 + taxRate)) AS price
      FROM Product, Location
      WHERE Product.saleLocID = Location.lid

Schema Mapping is Ubiquitous

- Fundamental problem in numerous applications
- Databases
  - data integration
  - data exchange / warehousing
  - semantic query processing
  - model management
  - peer data management
- AI/Knowledge Representation
  - knowledge base/ontology merging, ontology alignment, ...
- Web
  - e-commerce: matching product catalogs
  - marking up web forms using (Semantic Web) ontologies
  - linking ontologies of linked data

Schema Matching/Mapping is Difficult

- Matching and mapping systems must reconcile semantic heterogeneity between the schemas
- Such semantic heterogeneity arise in many ways
  - same concept, but different names for tables and attributes
    - rating vs classification
  - multiple attributes in 1 schema relate to 1 attribute in the other
    - basePrice and taxRate relate to price
  - tabular organization of schemas can be quite different
    - one table in AGGREGATOR vs three tables in DVD-VENDOR
  - coverage and level of details can also differ significantly
    - DVD-VENDOR also models releaseDate and releaseCompany

DVD-VENDOR
Movies(id, title, year)
Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
Locations(lid, name, taxRate)

AGGREGATOR
Items(name, releaseInfo, classification, price)

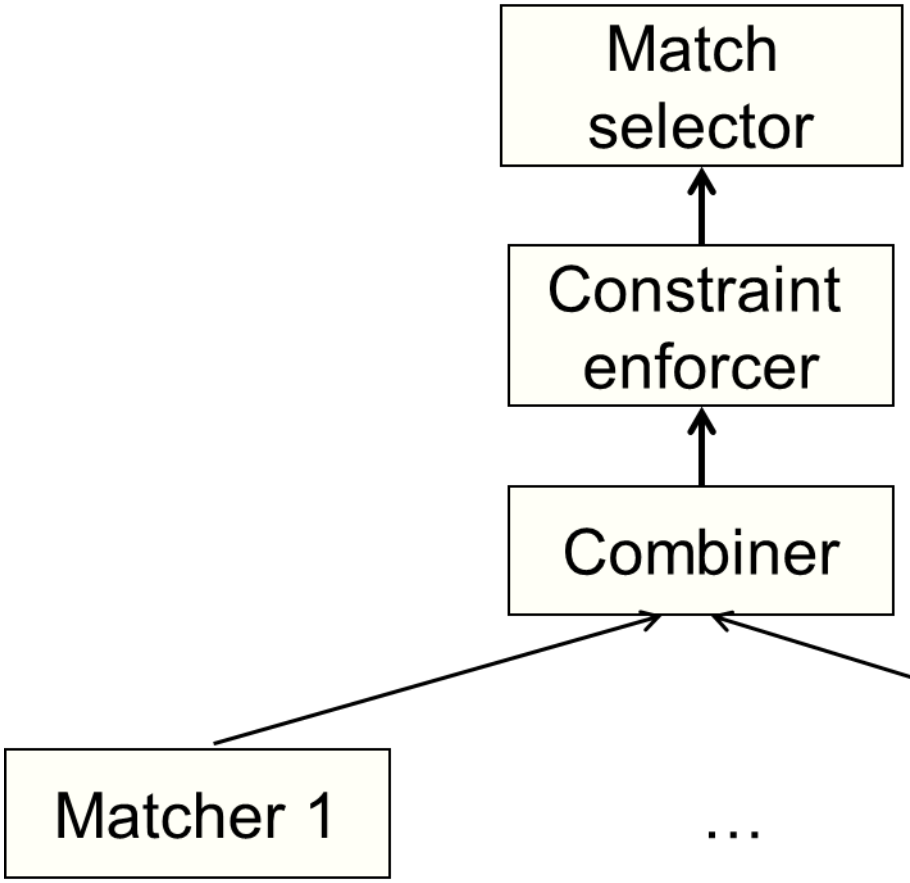
Schema Matching/Mapping is Difficult

- Why do we have semantic heterogeneity?
  - Schemas created by people with different knowledge and styles
  - Databases created for different purposes
- Why reconciling semantic heterogeneity is hard
  - the semantics is not fully captured in the schemas/data
    - not adequately documented
    - schema creator has retired to Florida!
  - schema clues can be unreliable
    - same names => different entities: area => location or square-feet
    - different names => same entity: area or address => location
  - intended semantics can be subjective
    - house-style = house-description?
  - correctly combining the data is difficult
    - Multiple attributes map to one: price = basePrice * (1 + taxRate)
- Standard is not a solution!
  - Small number of attributes, strong incentive to agree
- Cannot be *fully* automated, needs user feedback!

Outline

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

Matching System Architecture



sim → matches

filter(sim) → sim

c(sim_1, … sim_n) → sim

Schemas → similarity matrix

Matchers

- Schemas → Similarity Matrix
- Input:
  - two schemas S and T, plus any possibly helpful auxiliary information (e.g., data instances, text descriptions)
- Output:
  - similarity matrix that assigns to each element pair of S and T a number in [0,1] predicting whether the pair match
- Numerous matchers have been proposed. Ex:
  - name matchers
  - data matchers

Variety of Schema Matching Approaches

- Match algorithm can consider
  - *Instance* data – i.e., data contents
  - *Schema* information or metadata
- Match can be performed on
  - *Individual elements* – e.g., attributes
  - *Schema structure* – combination of elements
- Match algorithm can use
  - *Language* approaches – based on names or text descriptions
  - *Constraint-based* approaches – based on keys and relationships
  - *Machine learning* approaches – probabilistic description of data structure
- Match may relate 1 or n elements of one schema to 1 or n elements of another schema

Name-Based Matchers

- Use string matching techniques
  - Ex: edit distance, Jaccard, Soundex, etc.
- Linguistic pre-processing
  - split on delimiters
    - e.g., saleLocID → sale, Loc, ID
  - expand known abbreviations or acronyms
    - loc → location, cust → customer
  - expand a string with synonyms / hyponyms
    - Car ~ automobile, make ~ brand, cost ~ price
    - expand product into book, dvd, cd, …
  - remove stop words
    - in, at, and
  - sometimes stemming
    - products product*ion* ~> product

Instance-Based Matchers

- When schemas come with data instances, these can be extremely helpful in deciding matches
- Many instance-based matchers have been proposed
- Some of the most popular
  - recognizers
    - use dictionaries, regexes, or simple rules
  - overlap matchers
    - examine the overlap of values among attributes
  - Classifiers
    - use machine learning techniques

Schema Matching based on data values

Price	agent-name	City	State
$679K	Jane Fiedler	Los Angeles	CA
$905K	Jay Bellet	Los Angeles	CA
$850K	Chris Lee	Culver City	CA
$899K	Chris Lee	Culver City	CA

Listed-price	Contact-name	address
$755,000	J. Smith	Los Angeles, CA
$849,000	D. May	Venice, CA
$899,000	D. May	Venice, CA
$710,000	R. Chun	Los Angeles, CA
$935,000	J. Smith	Los Angeles, CA

Exploit similarities in the instances to learn relationship between schema elements

Recognizers

- Use dictionaries, regular expressions, or rules to recognize data values of certain kinds of attributes
- Example attributes for which recognizers are well suited
  - country names, city names, US states, zip codes
  - person names (can use dictionaries of last and first names)
  - colors, rating (e.g., G, PG, PG-13, etc.)
  - phone, fax, social security numbers
  - genes, protein names

Overlap Matchers

- Typically applies to attributes whose values are drawn from some finite domain
  - e.g., movie ratings, movie titles, book titles, country names
- Jaccard measure is commonly used to measure overlap of values
- Example:
  - use Jaccard measure to build a data-based matcher between DVD-VENDOR and AGGREGATOR
  - AGGREGATOR.name refers to DVD titles,
    DVD-VENDOR.name refers to sale locations,
    DVD-VENDOR.title refers to DVD titles
    → low score for (name, name), high score for (name, title)

Classifiers

- Build classifiers one schema and use them to classify elements of the other schema
  - Ex: Naïve Bayes, decision tree, rule learning, SVM
- Common strategy
  - for each element t_i of schema T, train classifier C_i to recognize instances of t_i
    - positive examples: instances of t_i
    - negative examples: instances of other elements of T
  - use classifier C_i to compute similarity between t_i and each element s_j of schema S
    - for each instance of s_j, Ci produces a number in [0,1] that is the confidence that the instance is indeed an instance of t_i
    - Aggregate confidence scores of instances of s_j (e.g., average) to return a single confidence score
      → similarity score between s_j and t_i

Using Classifiers: An Example

SCHEMA S

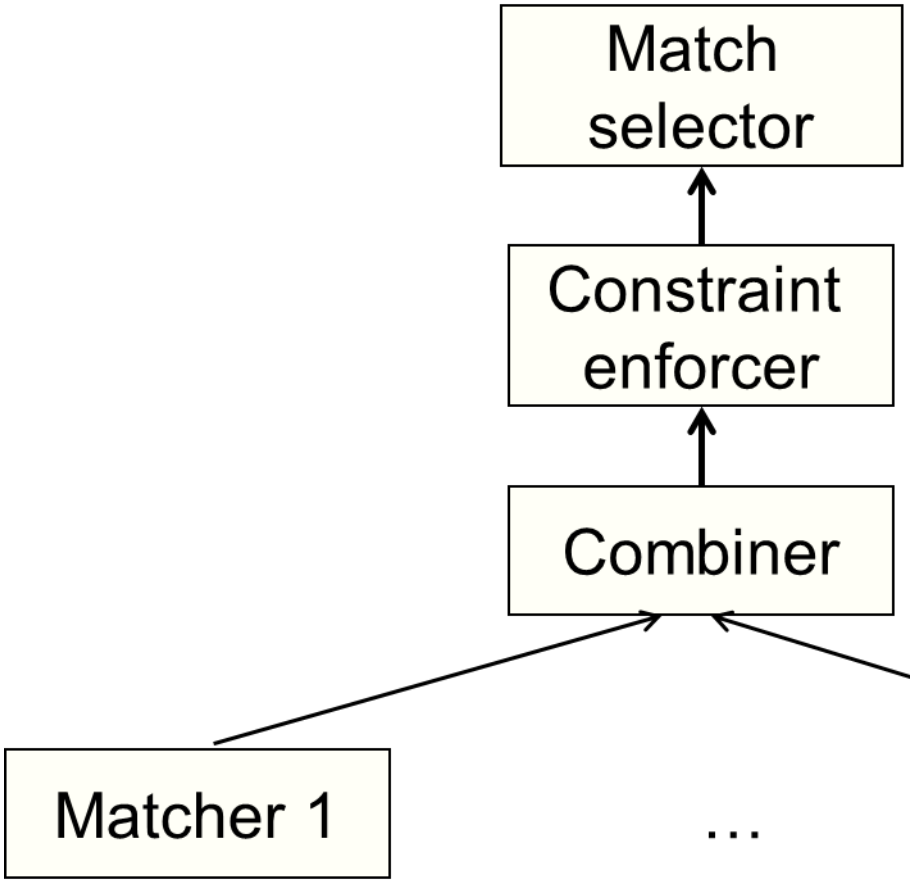
current-showing	address	phone
Lord of the Rings	Madison WI	(608) 695 2311
	Mountain View CA	(650) 277 1358

SCHEMA T

name	location	phone
…	Milwaukee WI	…
	Palo Alto CA	
	Philadelphia PA	

Assume we train on S.address, and we get probabilites of the 3 instances of T.location are 0.9, 0.7, and 0.5 ➔ return average score of 0.7 as similarity score between S.address and T.location

Matching System Architecture



sim → matches

filter(sim) → sim

c(sim_1, … sim_n) → sim

Schemas → similarity matrix

Combining Match Predictions

- Combination functions
  - average combiner : when we do not have any reason to trust one matcher over the others
  - maximum combiner: when we trust a strong signal from matchers
  - minimum combiner: when we want to be more conservative
- More complex types of combiners
  - use hand-crafted scripts
    - Ex: if s_i is address, then return score of the data-based matcher, otherwise return average score of all matchers
  - weighted-sum combiners
    - give weights to each matcher, according to its importance
  - combiner itself can be a learner
    - Learn weights from training data
    - Ex: decision tree, linear regression, logistic regression ...

Example of Average Combiner

DVD-VENDOR
Movies(id, title, year)
Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
Locations(lid, name, taxRate)

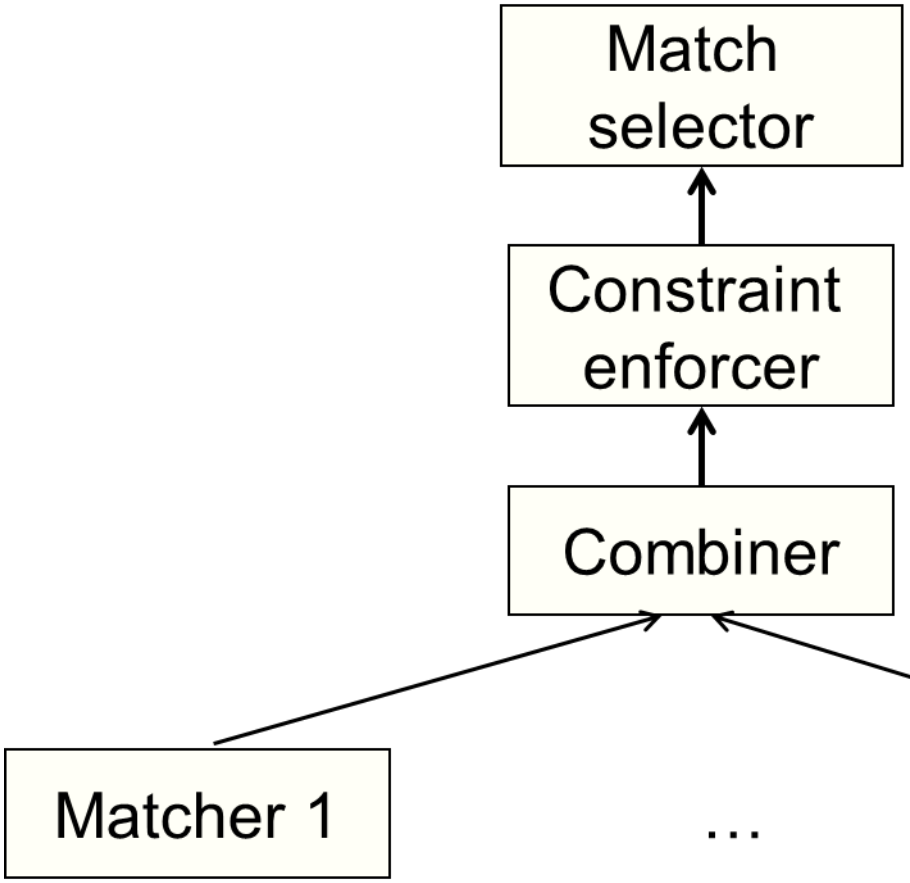
AGGREGATOR
Items(name, releaseInfo, classification, price)

name-base matcher:
name = <name: 1, title: 0.2>
releaseInfo = <releaseDate: 0.5, releaseCompany: 0.5>
price = <basePrice: 0.8>

data-based matcher:
name = <name: 0.2, title: 0.5>
releaseInfo = <releaseDate: 0.7>
classification = <rating: 0.6>
price = <basePrice: 0.2>

average-combiner:
name = <name: 0.6, title: 0.35>
releaseInfo = <releaseDate: 0.6, releaseCompany: 0.25>
classification = <rating: 0.3>
price = <basePrice: 0.5>

Matching System Architecture



sim → matches

filter(sim) → sim

c(sim_1, … sim_n) → sim

Schemas → similarity matrix

Enforcing Domain Constraints

- Designer often has knowledge that can be naturally expressed as domain integrity constraints
- Constraint enforcer exploits these to prune certain match combinations
  - searches through the space of all match combinations produced by the combiner
  - finds one combination with the highest aggregated confidence score that satisfies the constraints
- Two kinds of constraints:
  - Hard constraints: must be enforced
  - Soft constraints: more heuristic, may be violated
- Minimize cost of constraint violation
  - hard constraints: cost is infinite
  - soft constraints: cost can be any positive number

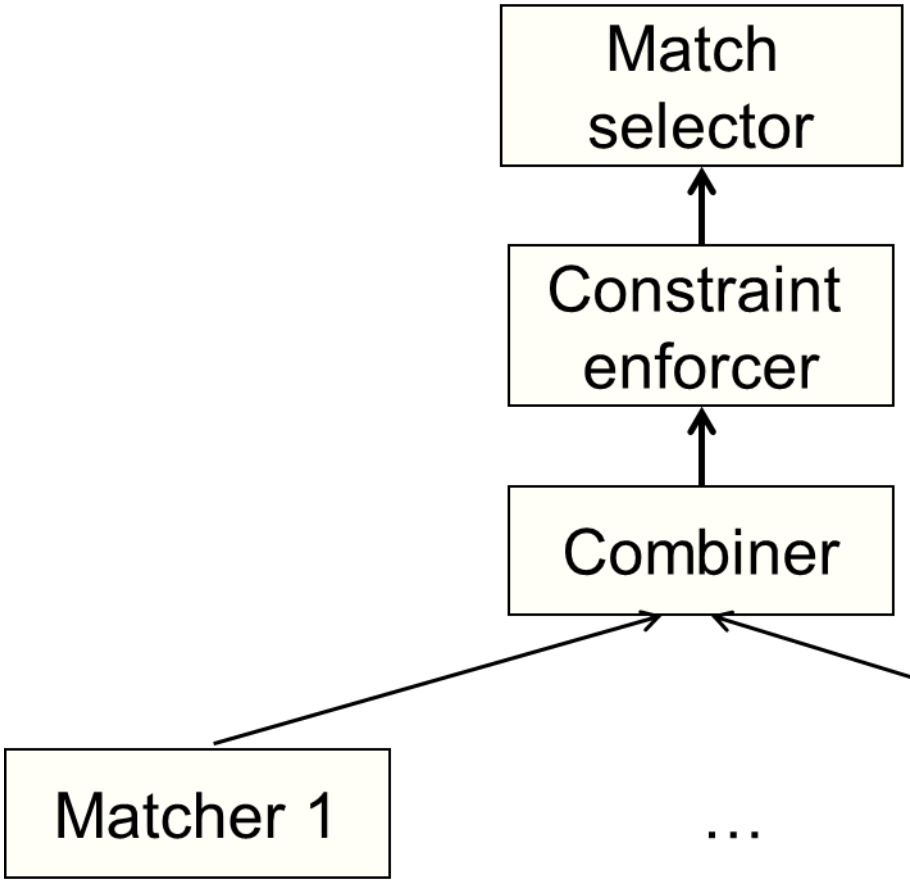
Example of Constraints

BOOK-VENDOR
Books(ISBN, publisher, pubCountry, title, review)
Inventory(ISBN, quantity, location)

DISTRIBUTOR
Items(code, name, brand, origin, desc)
InStore(code, availQuant)

	Constraint	Cost
c_1	If A = Items.code, then A is a key (all distinct values)	∞
c_2	If A = Items.desc, then any random sample of 100 data instances of A must have an average length of at least 20 words	1.5
c_3	If A_1=B_1, A_2=B_2, B_2 is next to B_1 in the schema, but A_2 is not next to A_1, then there is no A' next to A_1 such that $\|sim(A',B_2) - sim(A_2,B_2)\| \leq t$ for a small pre-specified t	2
c_4	If more than half of the attributes of Table U match those of Table V, then U = V	1

Matching System Architecture



sim → matches

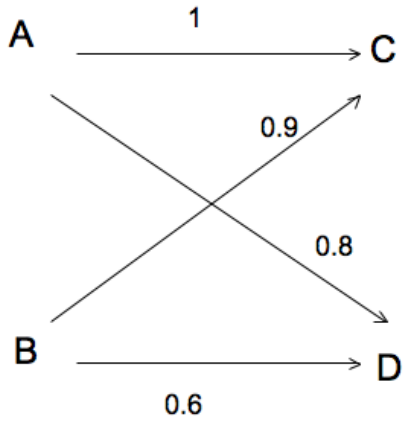
filter(sim) → sim

c(sim_1, … sim_n) → sim

Schemas → similarity matrix

Match Selector

- Selects matches from the similarity matrix
- Thresholding: return element pairs with similarity above threshold
  - Given similarity matrix: name=<title: 0.5>, releaseInfo = <releaseDate:0.6>, classification = <rating: 0.3>, price = <basePrice: 0.5> , and threshold 0.5, return matches name~title, releaseInfo~release, and Dateprice~basePrice
- Stable marriage:
  - Match is unstable if there are two couples $A_i = B_j$ and $A_k = B_l$ such that A_i and B_l want to be with each other [sim(i,l) > sim(i, j) and sim(i,l) > sim(k,l)]
    - A-C, B-D
- Maximize sum of scores
  - A-D, B-C
- More complex strategies return the top few match combinations



Outline

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

Reusing Previous Matches

- Schema matching tasks are often repetitive
  - Ex: keep matching new sources into the mediated schema
- Can a schema matching system improve over time? Can it learn from previous experience?
- Yes, use machine learning techniques
  - consider matching sources S_1, ..., S_n into a mediated schema G
  - manually match S_1, ..., S_m into G (where m << n)
  - system generalizes to predict matches for S_{m+1}, ..., S_n
    - use a technique called multi-strategy learning

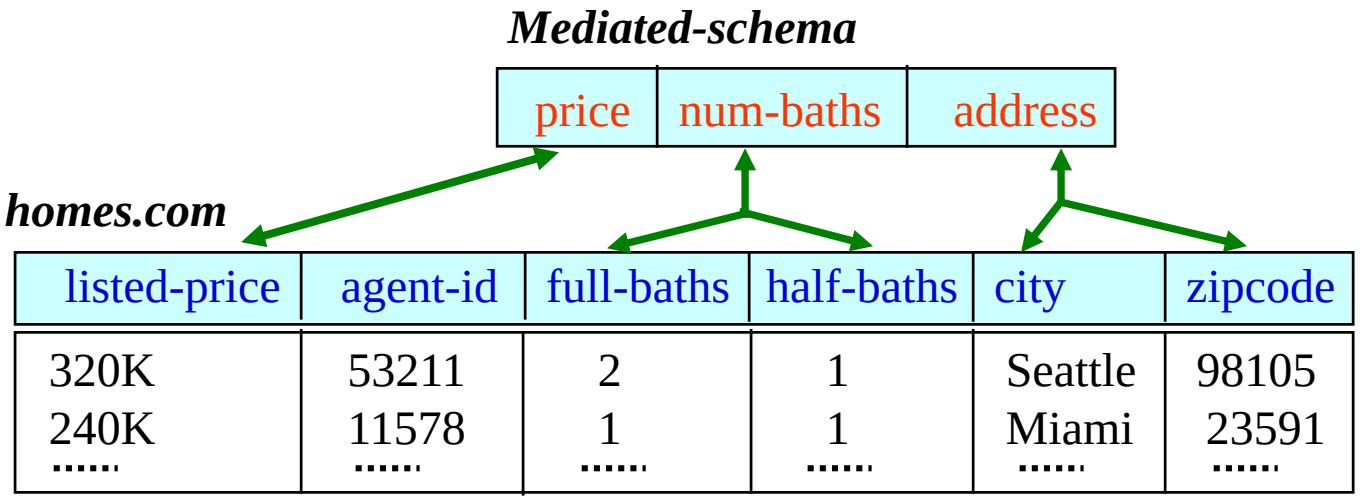
Multi-Strategy Learning: Training Phase

- Employ a set of learners $\mathbf{L}_1$, ..., $\mathbf{L}_k$
  - each learner creates a classifier for an element e of the mediated schema G, from training examples of e
  - these training examples are derived using semantic matches between the training sources $\mathbf{S}_1$, ..., $\mathbf{S}_m$ and G
- Use a meta-learner to learn a weight $\mathbf{w}_{e,Li}$ for each element e of the mediated schema and each learner $\mathbf{L}_i$
  - these weights will be used later in the matching phase to combine the predictions of the learners $\mathbf{L}_i$

Outline

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

iMap: Discovering Complex Semantic Matches between Database Schemas



- For each mediated-schema element
  - searches space of all matches
  - finds a small set of likely match candidates
- To search efficiently
  - employs a specialized searcher for each element type
  - Text Searcher, Numeric Searcher, Category Searcher, ...

[Dhamanka, Lee, Halevy, Domingos, SIGMOD 2004]

Candidate Match Generator

Given target (mediated) schema, generator discovers a small set of candidate matches

- E.g., for real estate schema
  - S2.list-price = S1.price
  - S2.list-price = S1.price * (1 + S1.monthly-fee-rate)

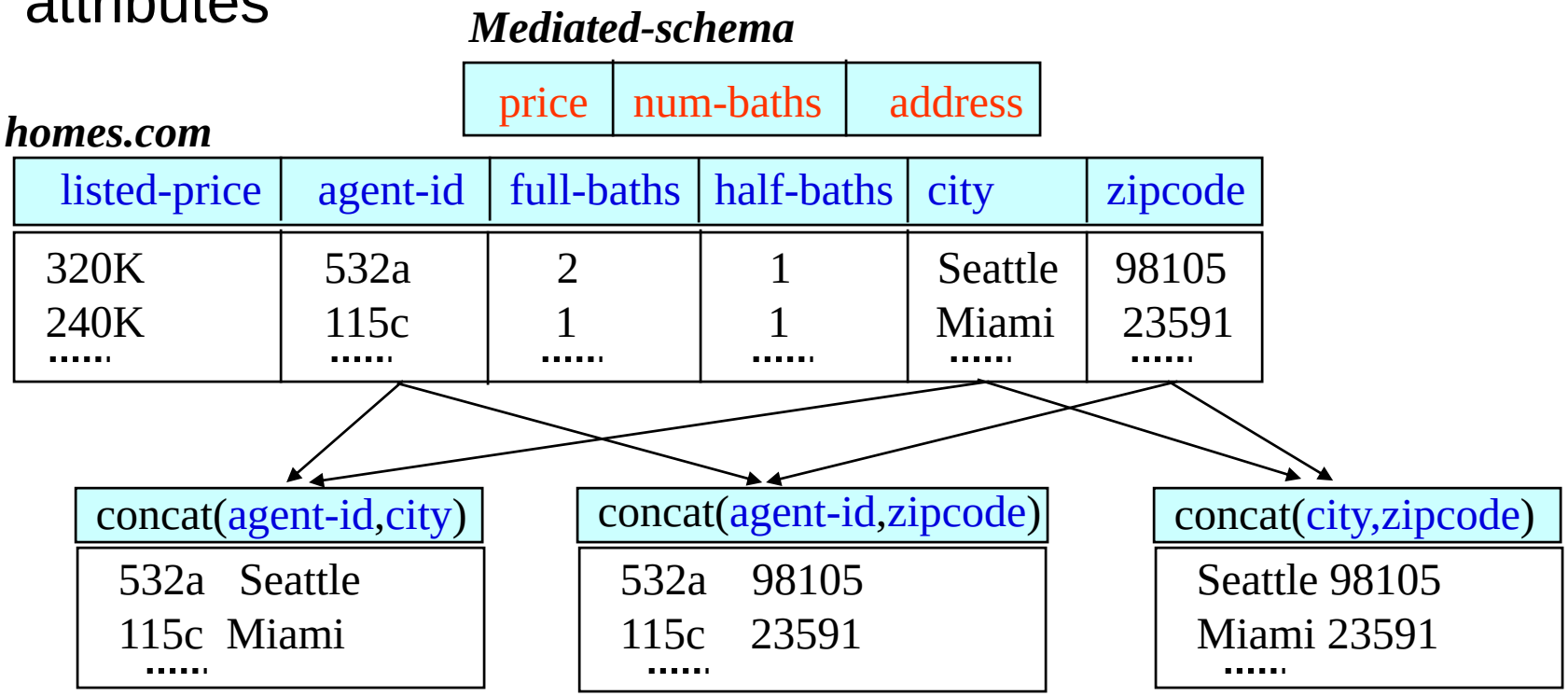
- Search through space of possible match candidates
  - Uses specialized searchers
    - Text searchers: concatenations of columns
    - Numeric searchers: arithmetic operations
    - Date searcher: combine month/year/date
  - Each searcher explores a small portion of search space based on background knowledge of operators and attribute types
- System is extensible with additional searchers
  - E.g., Later add searcher that knows how to operate on Address

Searcher

- Search strategy
  - Beam search to handle large search space
    - Uses a scoring function to evaluate match candidate
    - Keep only *k* highest-scoring match candidates
- Match evaluation
  - Score of match candidates approximates semantic distance between it and target attribute
    - E.g., concat(city, state) and agent-address
    - Uses machine-learning, statistics, heuristics
- Termination based on diminishing returns
  - Diminishing return
    - Highest scores of beam search do not grow as quickly

An Example: Text Searcher

- Find match candidates for address
- Search in space of all concatenation matches over all string attributes



- Best match candidates for address
  - (agent-id,0.7), (concat(agent-id,city),0.75), (concat(city,zipcode),0.9)

iMap Searchers

Searcher	Space of candidates	Examples	Evaluation technique
Text	Text attributes of source schema	name=concat(first-name,last-name)	Naïve Bayes and beam search
Numeric	User supplied matches of past complex matches	list-price= price*(1+tax-rate)	Binning, KL divergence
Category	Attributes w/less than *t* distinct values	product-categories= product-types	KL divergence
Schema mismatch	Source attribute containing target schema info	fireplace=1 if house-desc has "fireplace"	KL divergence
Unit conversion	Physical quantity attributes	weigh-kg=2.2*lbs	Properties of distributions
Dates	Columns recognized as ontology nodes	birth-date=b-day/ b-month / b-year	Mapping into ontology

Exploiting Domain Knowledge

- Domain knowledge can help reduce search space, direct search, and prune unlikely matches early
- Types of domain knowledge
  - Domain constraints
    - name and beds are unrelated→never generate match candidates that combine these attributes
  - Past complex matches in related domains
    - Reuse past matches: e.g., price = pr*(1+0.06) to produce a template VARIABLE*(1+CONSTANT) to guide search
  - Overlap data between databases
    - Source and target databases share some data
    - Re-evaluate matches based on overlap data
  - External data supplied by domain experts
    - Can be used to describe the properties of attributes

Empirical Evaluation

- iMAP system
  - 12 searchers
- Four real-world domains
  - real estate, product inventory, cricket, financial wizard
  - target schema: 19 -- 42 elements, source schema: 32 -- 44
- Accuracy: 43 -- 92%
- Sample discovered matches
  - agent-name = concat(first-name,last-name)
  - area = building-area / 43560
  - discount-cost = (unit-price * quantity) * (1 - discount)
- More detail in [Dhamanka *et. al.* SIGMOD-04]

Observations

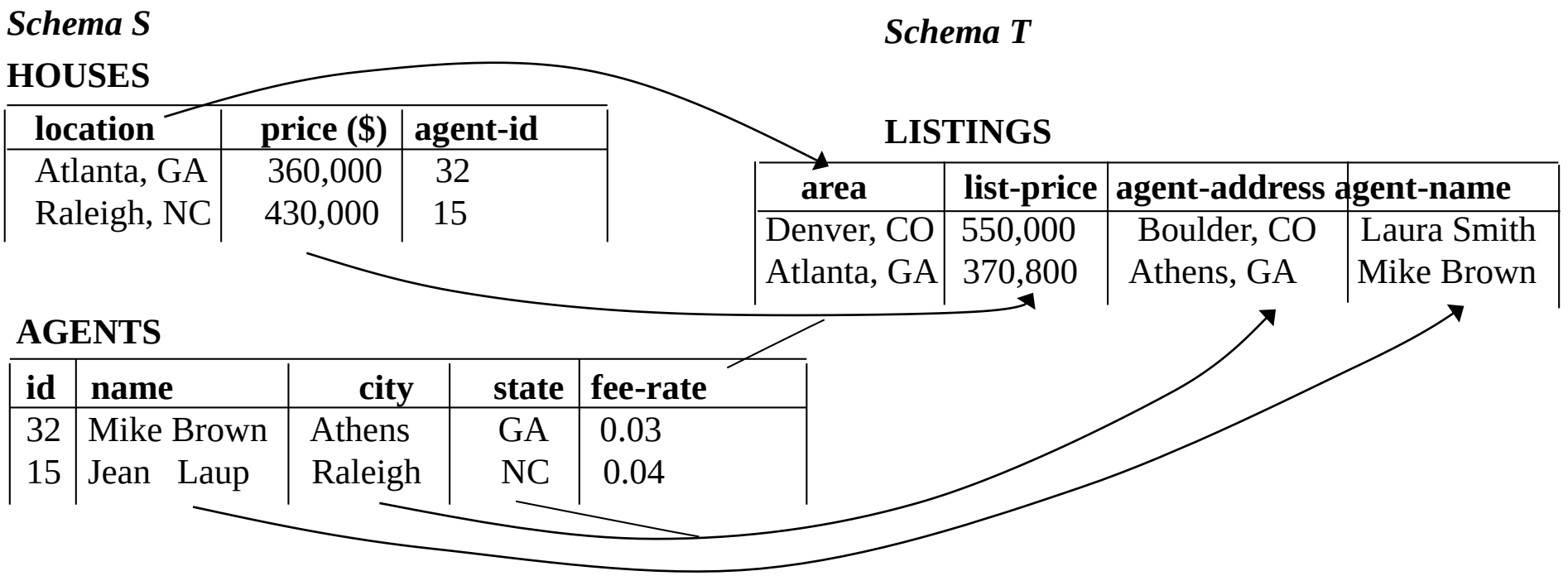
- Finding complex matches much harder than 1-1 matches!
  - require gluing together many components
  - e.g., num-rooms = bath-rooms + bed-rooms + dining-rooms + living-rooms
  - if missing one component => incorrect match
- However, even partial matches are already very useful!
  - so are top-k matches => need methods to handle partial/top-k matches
- Huge/infinite search spaces
  - domain knowledge plays a crucial role!
- Matches are fairly complex, hard to know if they are correct
  - must be able to explain matches
- Human must be fairly active in the loop
  - need strong user interaction facilities
- Break matching architecture into multiple "atomic" boxes!

Outline

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

Finding Matches is only Half of the Job!

- To translate data/queries, need mappings, not matches



- Mappings
  - area = SELECT location FROM HOUSES
  - agent-address = SELECT concat(city,state) FROM AGENTS
  - list-price = SELECT price * (1 + fee-rate)
    FROM HOUSES, AGENTS
    WHERE agent-id = id

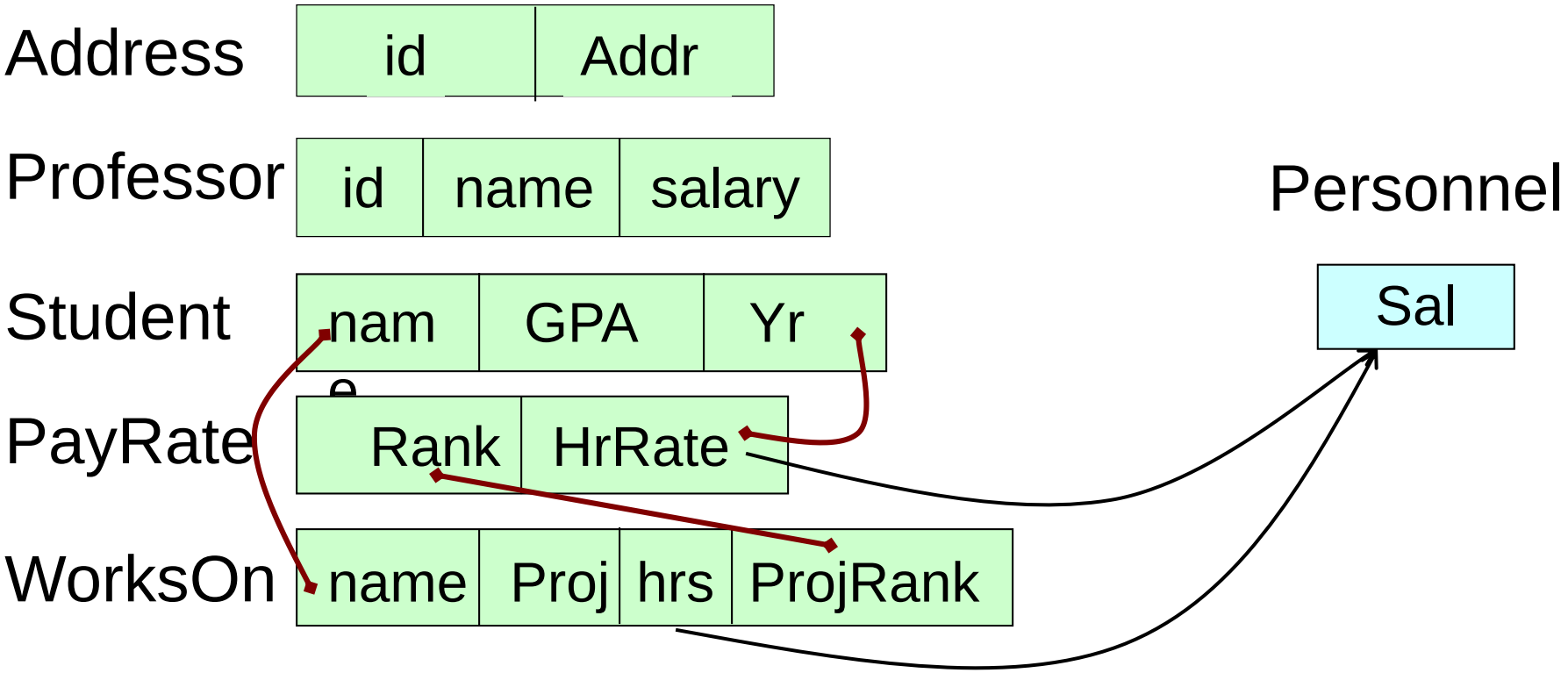
Clio: Elaborating Matches into Mappings

- Developed at U. of Toronto & IBM Almaden, 2000-2003

  R Fagin, L Haas, M Hernández, RJ Miller, L Popa, Y Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In Conceptual Modeling: Foundations and Applications, Springer 2009.

- Given a match
  - list-price = price * (1 + fee-rate)
- Refine it into a mapping
  - list-price = SELECT price * (1 + fee-rate)
    FROM houses FULL OUTER JOIN agents
    ON agent-id = id
- Need to discover
  - the correct join path among tables, e.g., agent-id = id
  - the correct join, e.g., full outer join? inner join?
- Use heuristics to decide
  - when in doubt, ask users
  - employ sophisticated user interaction methods [VLDB-00, SIGMOD-01]

Multiple Join Paths

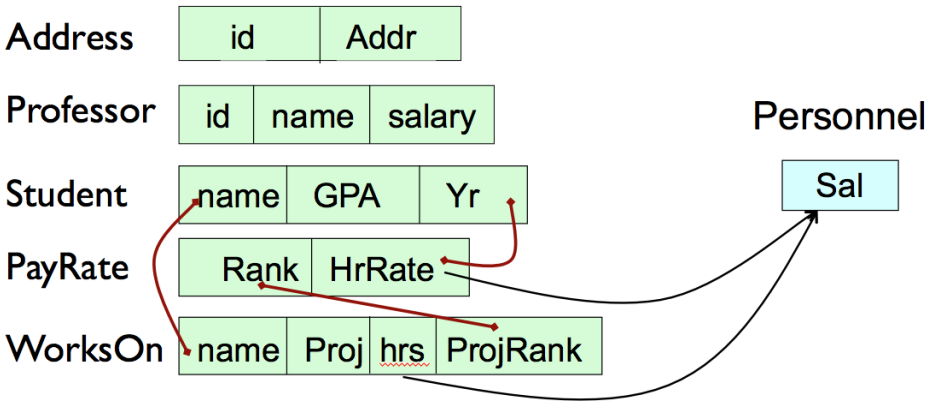


*f1: PayRate.HrRate * WorksOn.hrs = Personnel.Sal.*

Two Possible Queries

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W
where P.Rank = W.ProjRank
```

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W, Student S
where W.Name=S.Name and S.Yr = P.Rank
```

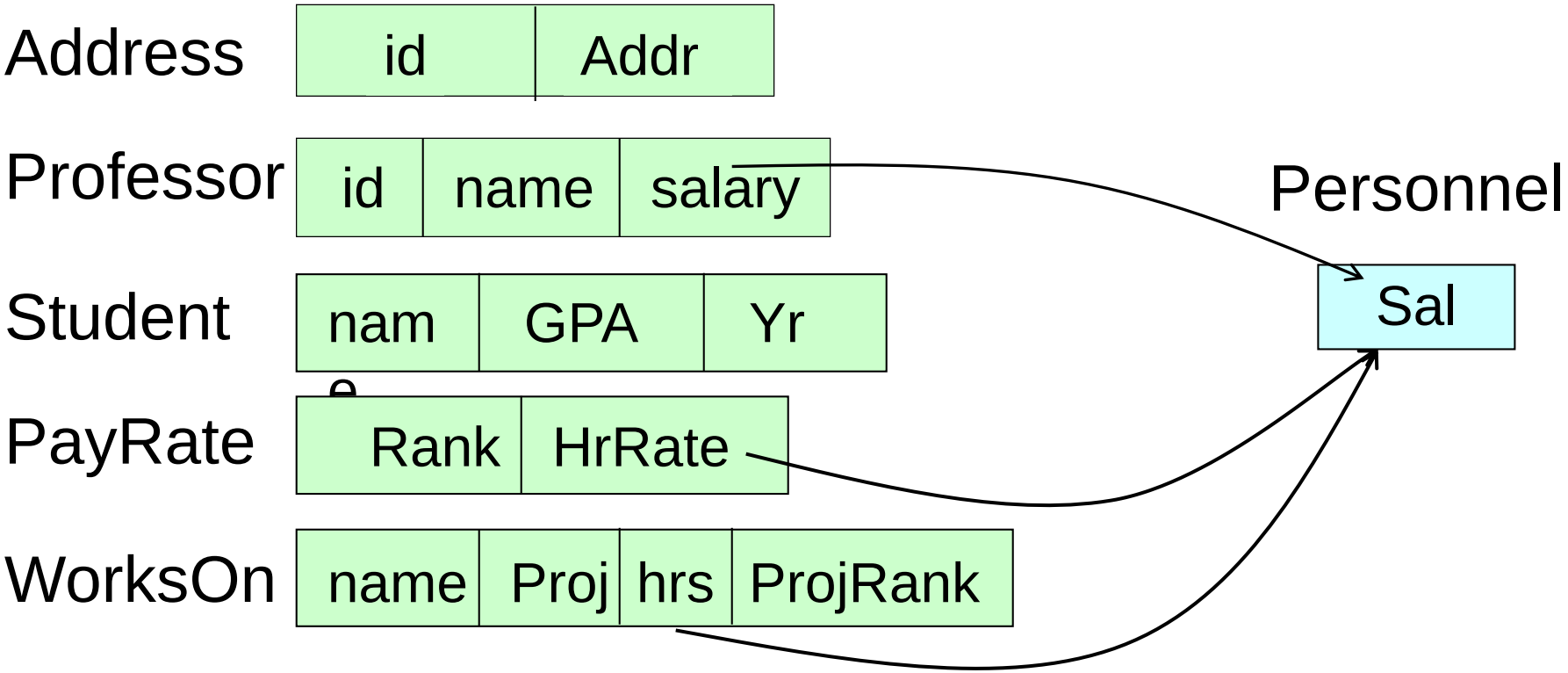


(We could also consider the Cartesian product, but that seems intuitively wrong)

Two Sets of Decisions

- What join paths to choose?
- How to combine the results of the joins?
- Underlying database-design principles:
  - Values in the source should appear in the target
  - They should only appear once
  - We should not lose information
- Discover candidate join paths by:
  - following foreign keys
  - look at paths used in queries
  - paths discovered by mining data for joinable columns.
- Select paths by:
  - Prefer foreign keys
  - Prefer ones that involve a constraint
  - Prefer smaller difference between inner and outer joins.

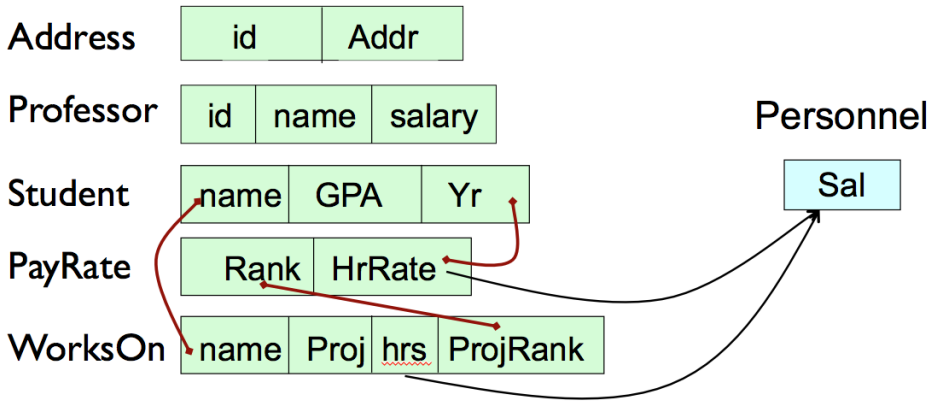
Horizontal partitioning



f2: Professor(Sal) → Personnel(Sal)

What Kind of Union?

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W
where P.Rank = W.ProjRank
UNION ALL
select Sal
from Professor
```

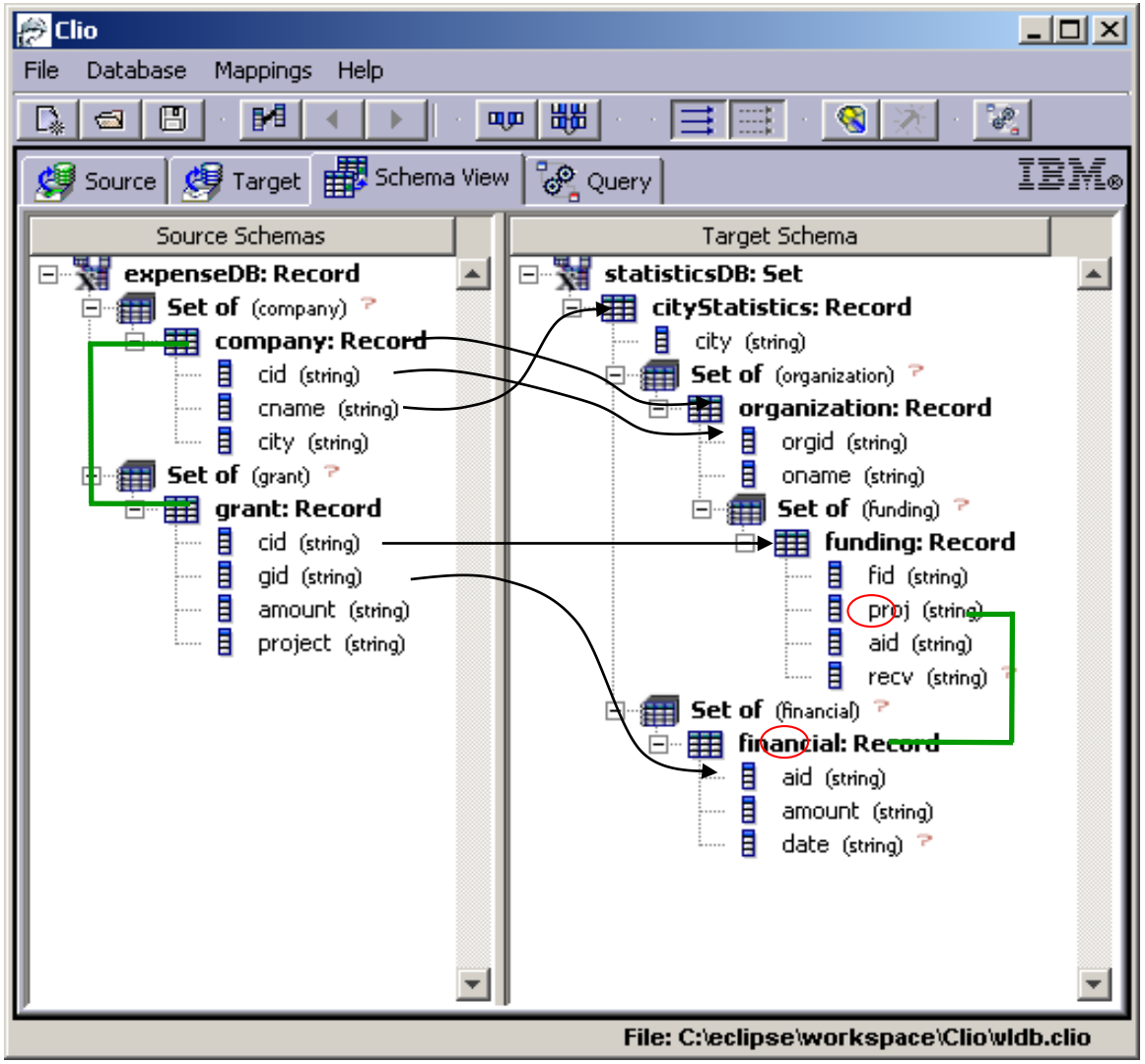


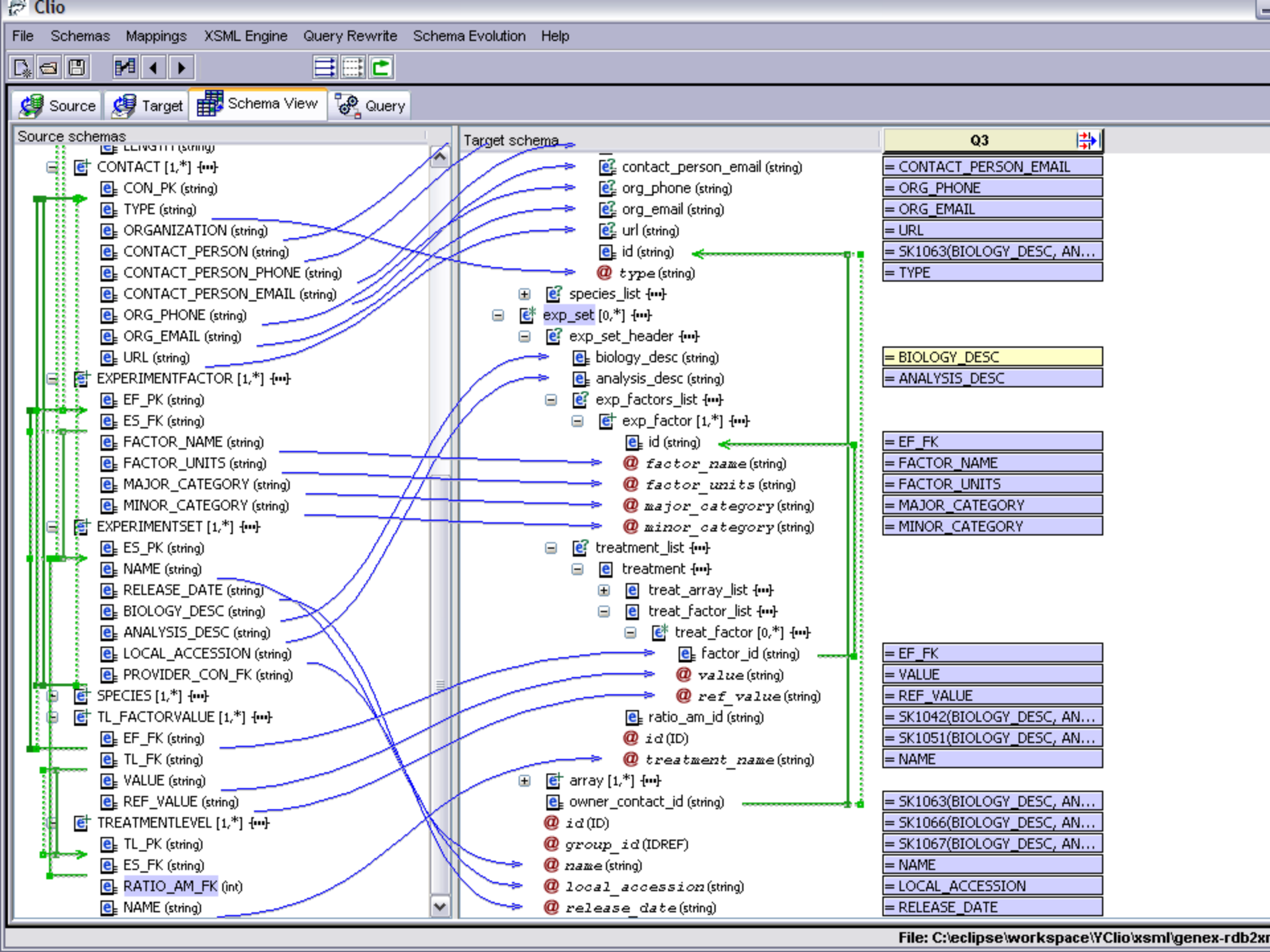
Selecting Covers

- Candidate cover: a minimal set of candidate sets (join paths) that covers all the input correspondences
- Select best cover:
  - Prefer with fewest candidate paths
  - Prefer one that covers more attributes of target
- Express mapping as union of candidate sets in selected cover.

Features of Clio

- Supports nested structures
  - Nested Relational Model & Constraints
- Interactive discovery of attribute correspondence.
- Interactive derivation of schema mappings.
- Performs data exchange
- Supports composition of schema mappings




Clio
File
Schemas
Mappings
XSML Engine
Query Rewrite
Schema Evolution
Help
Source
Target
Schema View
Query
Source schemas
CONTACT [1,*]
CON_PK (string)
TYPE (string)
ORGANIZATION (string)
CONTACT_PERSON (string)
CONTACT_PERSON_PHONE (string)
CONTACT_PERSON_EMAIL (string)
ORG_PHONE (string)
ORG_EMAIL (string)
URL (string)
EXPERIMENTFACTOR [1,*]
EF_PK (string)
ES_FK (string)
FACTOR_NAME (string)
FACTOR_UNITS (string)
MAJOR_CATEGORY (string)
MINOR_CATEGORY (string)
EXPERIMENTSET [1,*]
ES_PK (string)
NAME (string)
RELEASE_DATE (string)
BIOLOGY_DESC (string)
ANALYSIS_DESC (string)
LOCAL_ACCESSION (string)
PROVIDER_CON_FK (string)
SPECIES [1,*]
TL_FACTORVALUE [1,*]
EF_FK (string)
TL_FK (string)
VALUE (string)
REF_VALUE (string)
TREATMENTLEVEL [1,*]
TL_PK (string)
ES_FK (string)
RATIO_AM_FK (int)
NAME (string)
Target schema
contact_person_email (string)
org_phone (string)
org_email (string)
url (string)
id (string)
type (string)
species_list
exp_set [0,*]
exp_set_header
biology_desc (string)
analysis_desc (string)
exp_factors_list
exp_factor [1,*]
id (string)
factor_name (string)
factor_units (string)
major_category (string)
minor_category (string)
treatment_list
treatment
treat_array_list
treat_factor_list
treat_factor [0,*]
factor_id (string)
value (string)
ref_value (string)
ratio_am_id (string)
id (ID)
treatment_name (string)
array [1,*]
owner_contact_id (string)
id (ID)
group_id (IDREF)
name (string)
local_accession (string)
release_date (string)
Q3
= CONTACT_PERSON_EMAIL
= ORG_PHONE
= ORG_EMAIL
= URL
= SK1063(BIOLOGY_DESC, AN...
= TYPE
= BIOLOGY_DESC
= ANALYSIS_DESC
= EF_FK
= FACTOR_NAME
= FACTOR_UNITS
= MAJOR_CATEGORY
= MINOR_CATEGORY
= EF_FK
= VALUE
= REF_VALUE
= SK1042(BIOLOGY_DESC, AN...
= SK1051(BIOLOGY_DESC, AN...
= NAME
= SK1063(BIOLOGY_DESC, AN...
= SK1066(BIOLOGY_DESC, AN...
= SK1067(BIOLOGY_DESC, AN...
= NAME
= LOCAL_ACCESSION
= RELEASE_DATE
File: C:\eclipse\workspace\YClio\xsml\genex-rdb2xm

```
LET $doc0 := document("input XML file goes here")
RETURN
 <T>
   {distinct-values (
   FOR $x0 IN $doc0/companies, $x1 IN $doc0/grants,
       $x2 IN $doc0/contacts,  $x3 IN $doc0/contacts
   WHERE
       $x0/name/text() = $x1/recipient/text() AND $x1/supervisor/text() = $x2/cid/text() AND
       $x1/manager/text() = $x3/cid/text()
   RETURN
      <organization>
        <code>  {  $x0/name/text() }  </code>
        <year>  {  "Sk3(", $x0/name/text(), ")" }  </year>
        {distinct-values (
        FOR $x0L1 IN $doc0/companies, $x1L1 IN $doc0/grants,
            $x2L1 IN $doc0/contacts,  $x3L1 IN $doc0/contacts
        WHERE
            $x0L1/name/text() = $x1L1/recipient/text() AND $x1L1/supervisor/text() = $x2L1/cid/text()
        AND $x1L1/manager/text() = $x3L1/cid/text() AND $x0L1/name/text() = $x0/name/text()
        RETURN
           <funding>
              <fid> {$x0L1/gid/text()} </fid>
              <finId>{"Sk5(", $x0L1/name/text(), ", ", $x2L1/phone/text(), ", ",
                      $x1L1/amount/text(), ", ", $x1L1/gid/text(), ")"}</finId>
           </funding> ) }
     </organization> ) }
   {distinct-values (
   FOR $x0 IN $doc0/companies, $x1 IN $doc0/grants,
       $x2 IN $doc0/contacts,  $x3 IN $doc0/contacts
   WHERE
       $x0/name/text() = $x1/recipient/text() AND $x1/supervisor/text() = $x2/cid/text() AND
       $x1/manager/text() = $x3/cid/text()
   RETURN
       <financial>
          <finId>{"Sk5(", $x0/name/text(), ", ", $x2/phone/text(), ", ",
                       $x1/amount/text(), ", ", $x1/gid/text(), ")"}</finId>
          <budget>  {  $x1/amount/text() }  </budget>
          <phone>   {  $x2/phone/text() }  </budget>
      </financial> ) }
</T>
```

Summary

- Schema matching:
  - Use multiple matchers and combine results
  - Learn from the past
  - Incorporate constraints and user feedback
- From matching to mapping:
  - Search through possible queries
  - Principles from database design guide search
  - User interaction is key

Additional Slides (not for test)

Data integration and transformation

3. Data Exchange

Paolo Atzeni

Dipartimento di Informatica e Automazione

Università Roma Tre

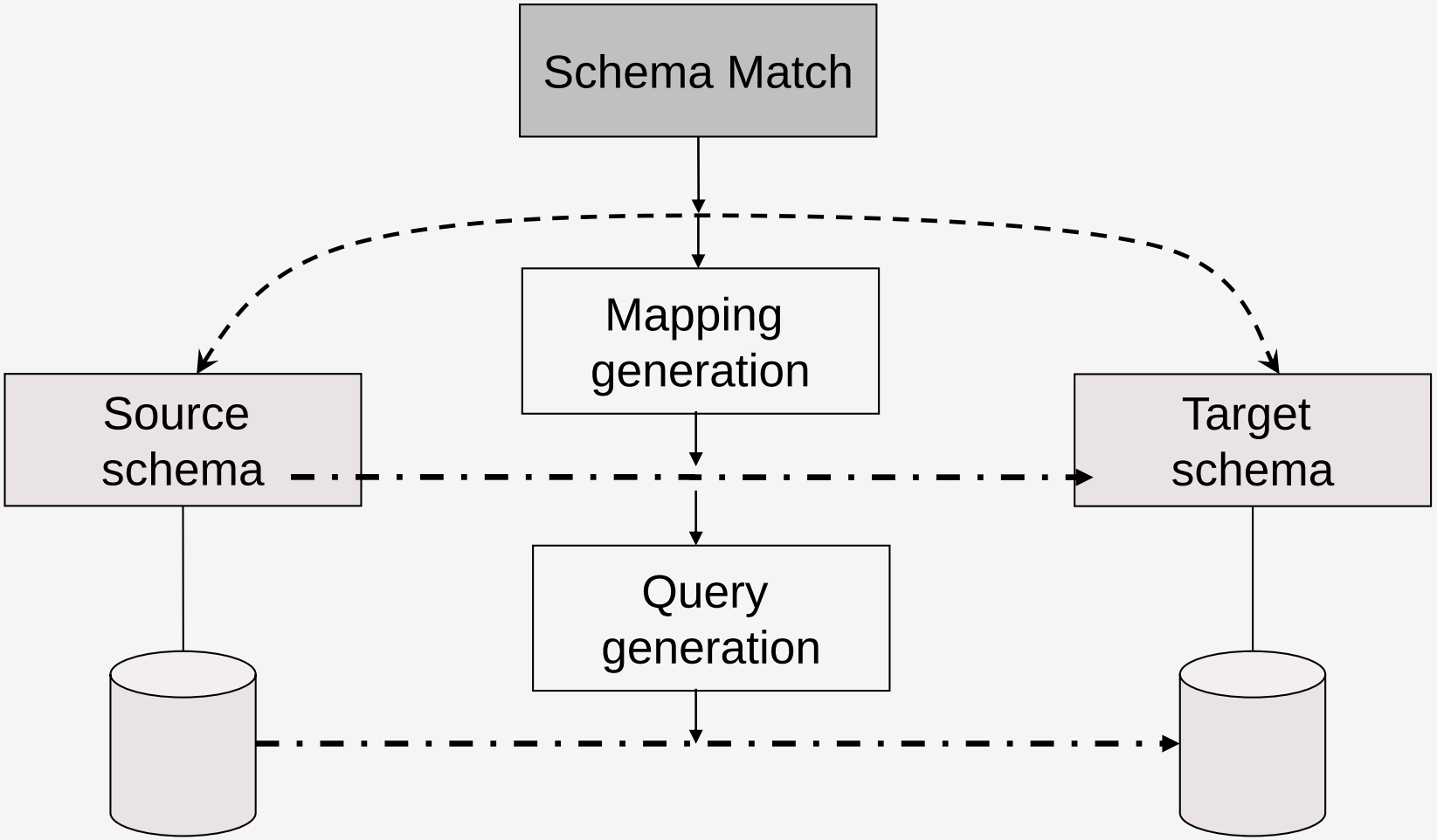
28/10-4/11/2009

References

- Ronald Fagin, Laura M. Haas, Mauricio Hernandez, Renee J. Miller, Lucian Popa, and Yannis Velegrakis "**Clio: Schema Mapping Creation and Data Exchange"** A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, LNCS 5600, Springer-Verlag Berlin Heidelberg, 2009, pp. 198–236.

  and other papers cited in it

Data exchange, a typical approach (the Clio project)



Simple example

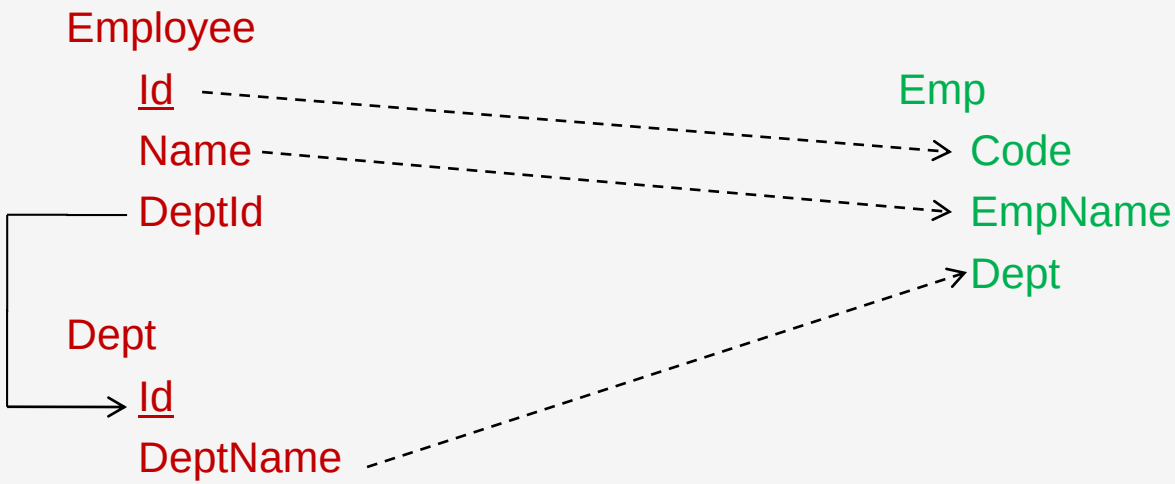
Dept(Id,DeptName) Emp(Code,EmpName,Dept)
Employee(Id,Name,DeptId)
(with FK from DeptId to Dept.Id)

Assume we know that
Employee.Id corresponds to Code
Name corresponds to EmpName
DeptName corresponds to Dept

We would like to obtain a query that populates Emp

```
SELECT Id as Code, Name AS EmpName, DeptName AS Dept
FROM Employee JOIN Dept ON DeptId = Dept.Id
```

Better visualization



We want to obtain

```
SELECT Id as Code, Name AS EmpName, DeptName AS Dept
FROM Employee JOIN Dept ON DeptId = Dept.Id
```

and not

```
SELECT Id as Code, Name AS EmpName, NULL AS Dept FROM Employee
UNION
SELECT NULL as Code, NULL AS EmpName, DeptName AS Dept FROM Dept
```

nor

```
SELECT Id as Code, NULL AS EmpName, NULL AS Dept FROM Employee
UNION
…
```

The main issue

- How do we discover we should use a join and not one or two unions?
- Attributes that appear together in a relation
  - Id,Name in the source and Code,EmpName in the target
- The foreign key

Data exchange, another example

Address (Id Addr)

Professor (Id Name Sal)

Student (Name GPA Yr)

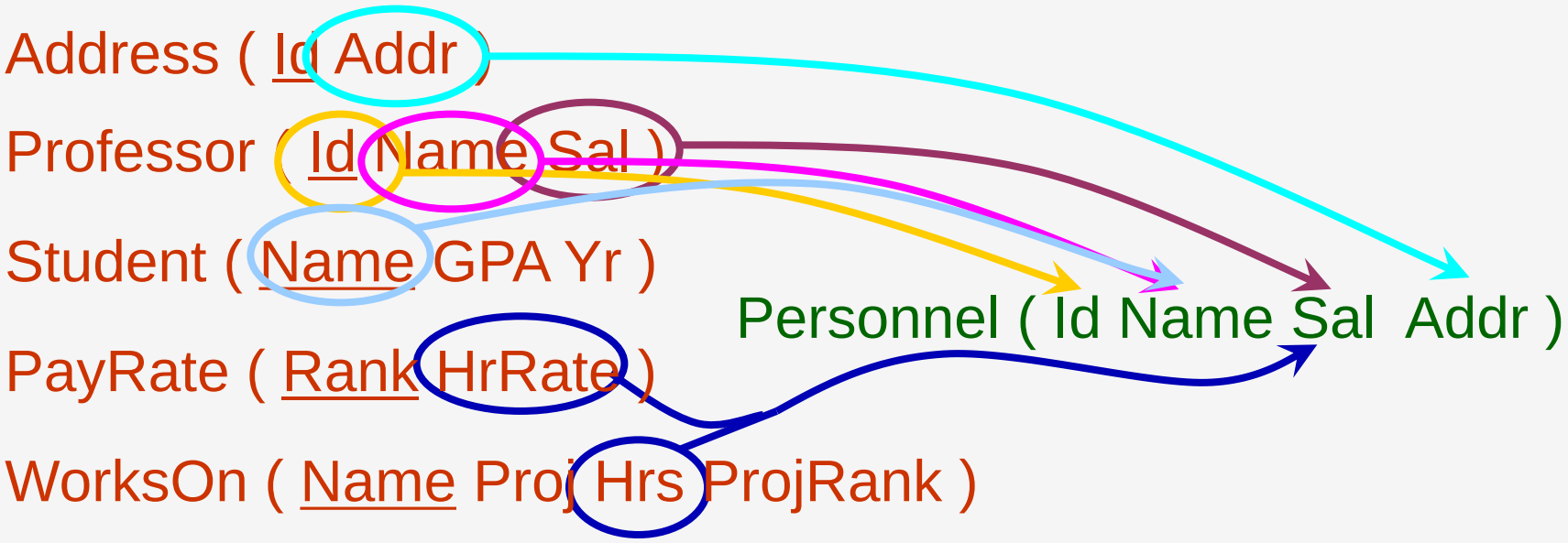
PayRate (Rank HrRate)

WorksOn (Name Proj Hrs ProjRank)

Personnel (Id Name Sal Addr)

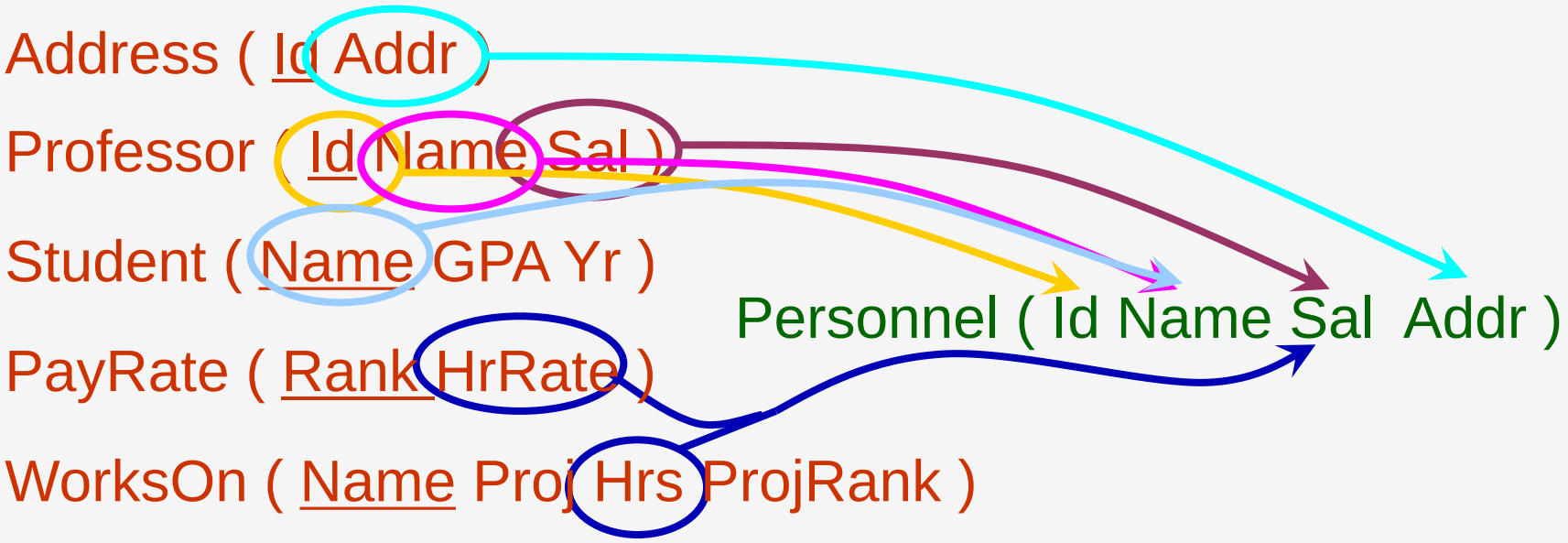
- Foreign keys
  - between the two Id
  - between ProjRank and Rank
  - between the two Name

Data exchange, example



- Assume we are given correspondences, which involve functions:
  - Usually identity
  - PayRate(HrRate)*WorksOn(Hrs) → Personnel(Sal)

Data exchange, example

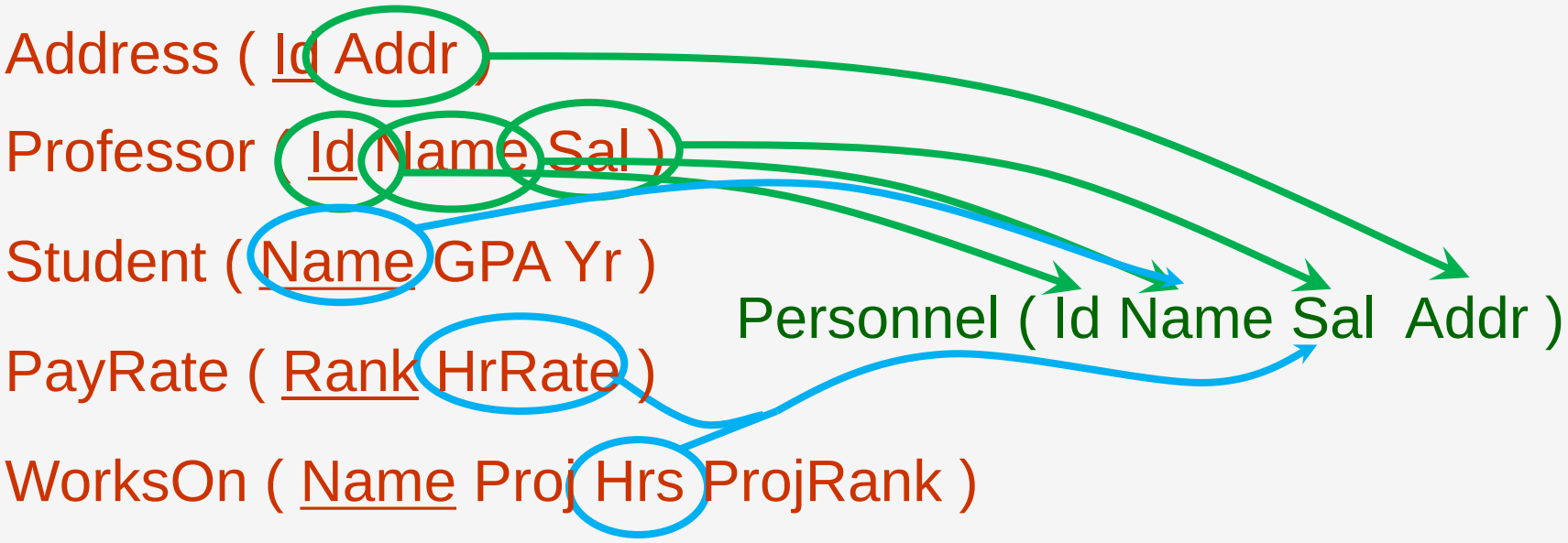


- How do we combine HrRate and Hrs?
  - Via a join suggested by foreign keys
- Foreign key between ProjRank and ProjRank suggests a join
- Foreign keys over Name and between Yr and Rank suggest another

Heuristic

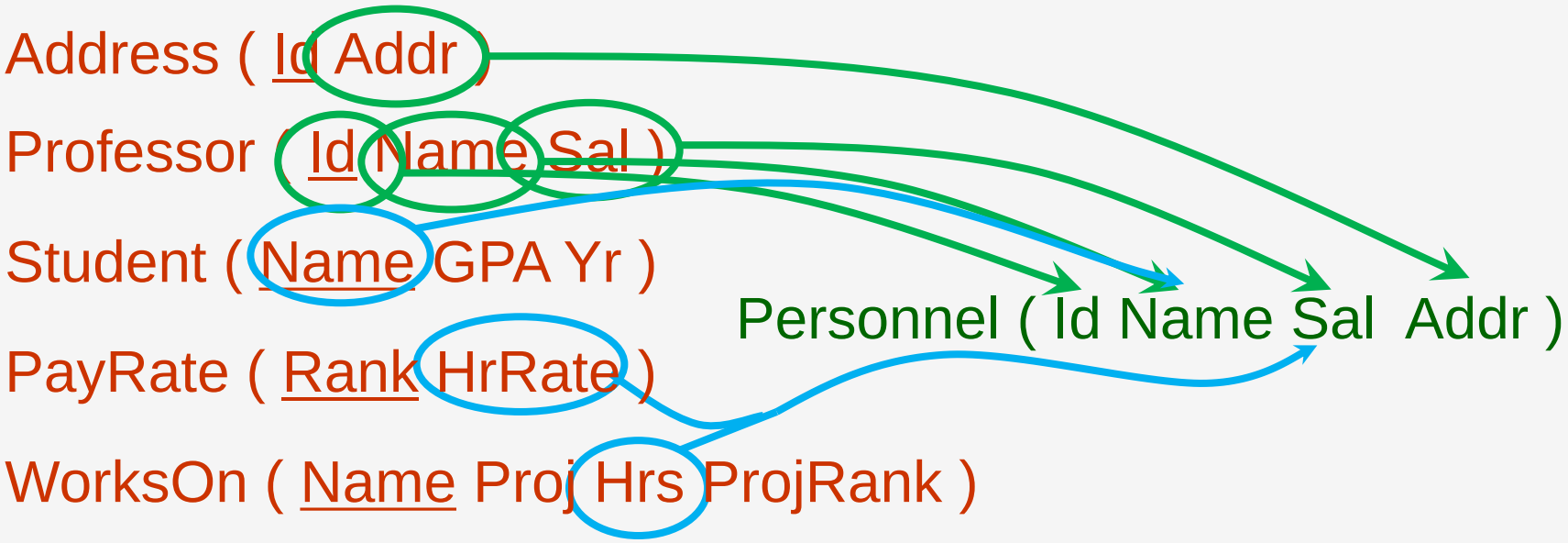
- We have many correspondences
- Group correspondences in such a way that each set contains at most one correspondence for each attribute in the target
- We are interested in sets where the source attribute are either in the same relations or in relations whose join is meaningful

Partition the correspondences



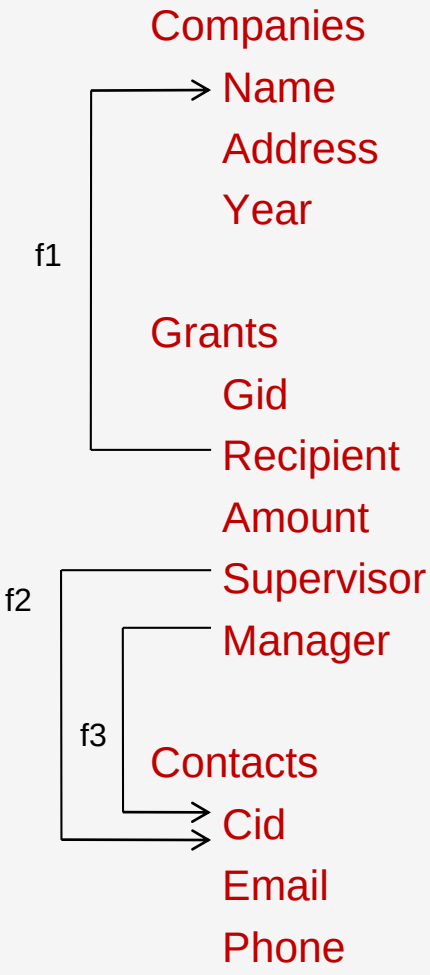
- … and for each partition the joins are meaningful

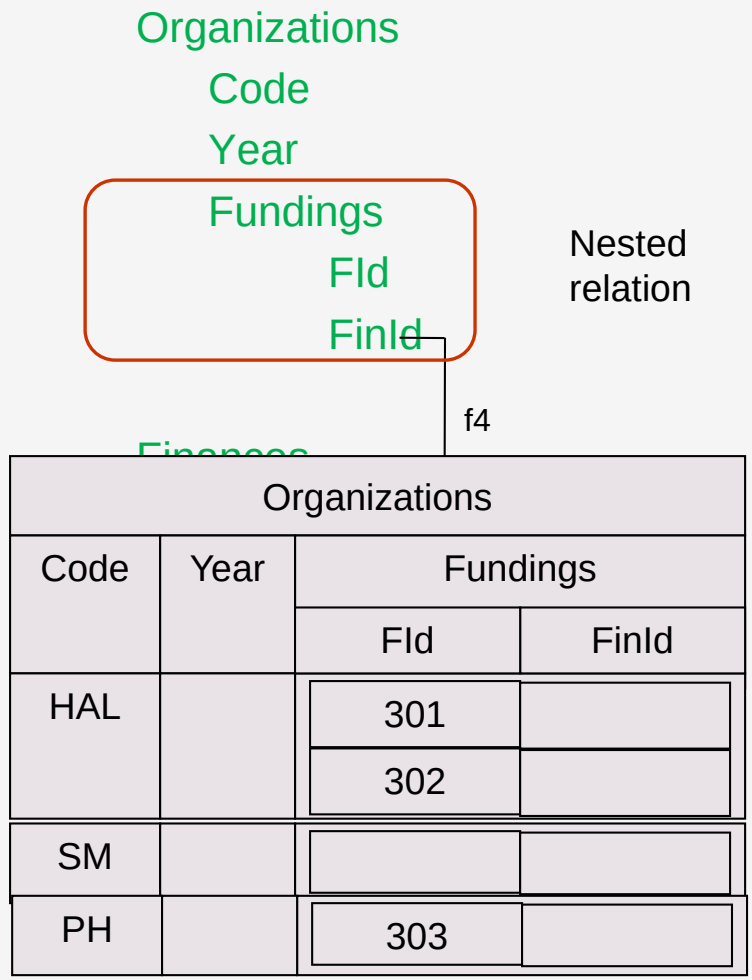
The process, example



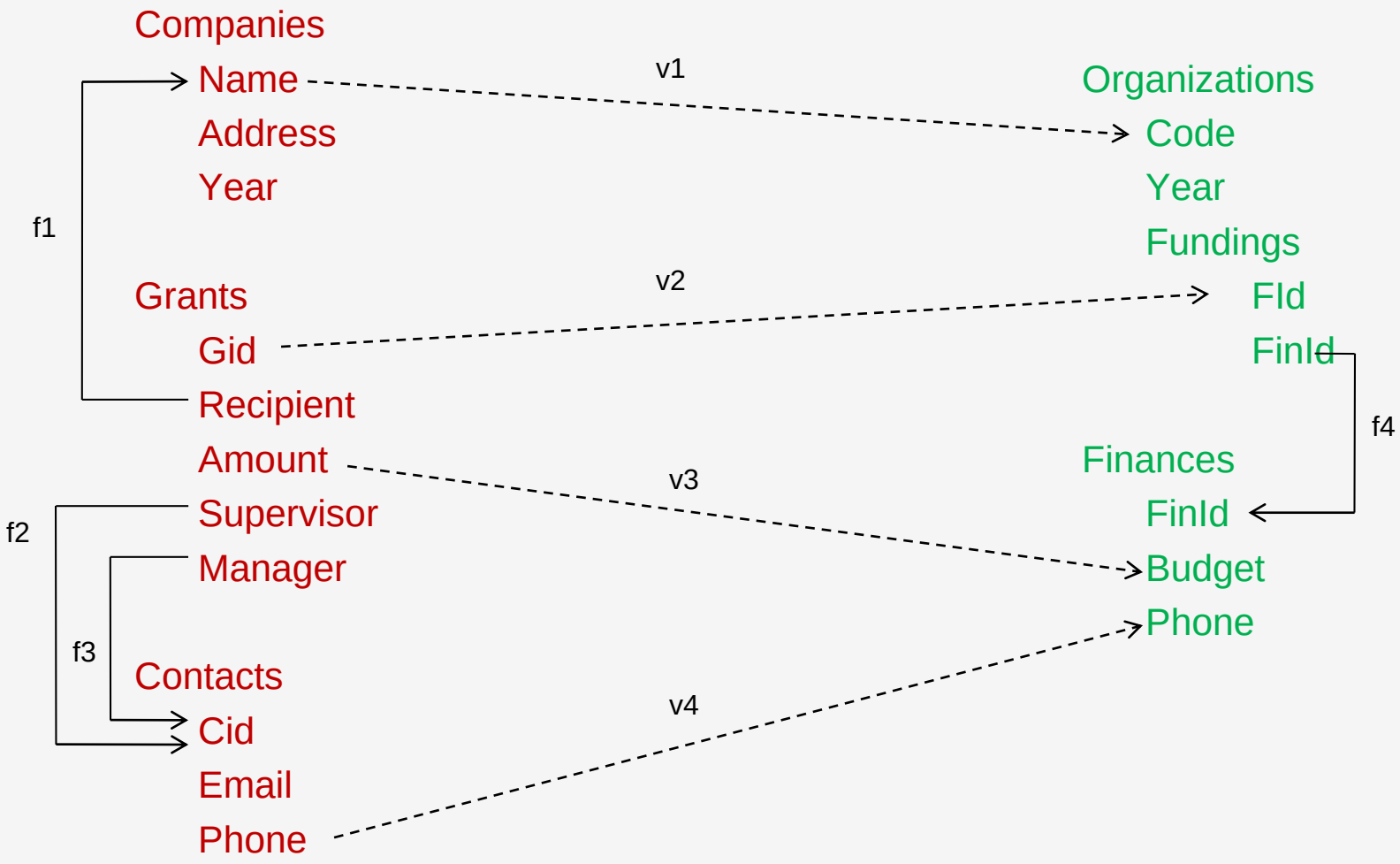

```
SELECT P.Id, P.Name, P.Sal, A.Addr
FROM Professor P, Address A
WHERE A.Id = P.Id
UNION ALL
SELECT NULL AS Id, S.Name, p.HrRate * W.Hrs, NULL AS Addr
FROM PayRate P, Student S, WorksOn W
WHERE W.Name = S.Name AND S.Yr = P.Rank
```

More complex example (with nesting)

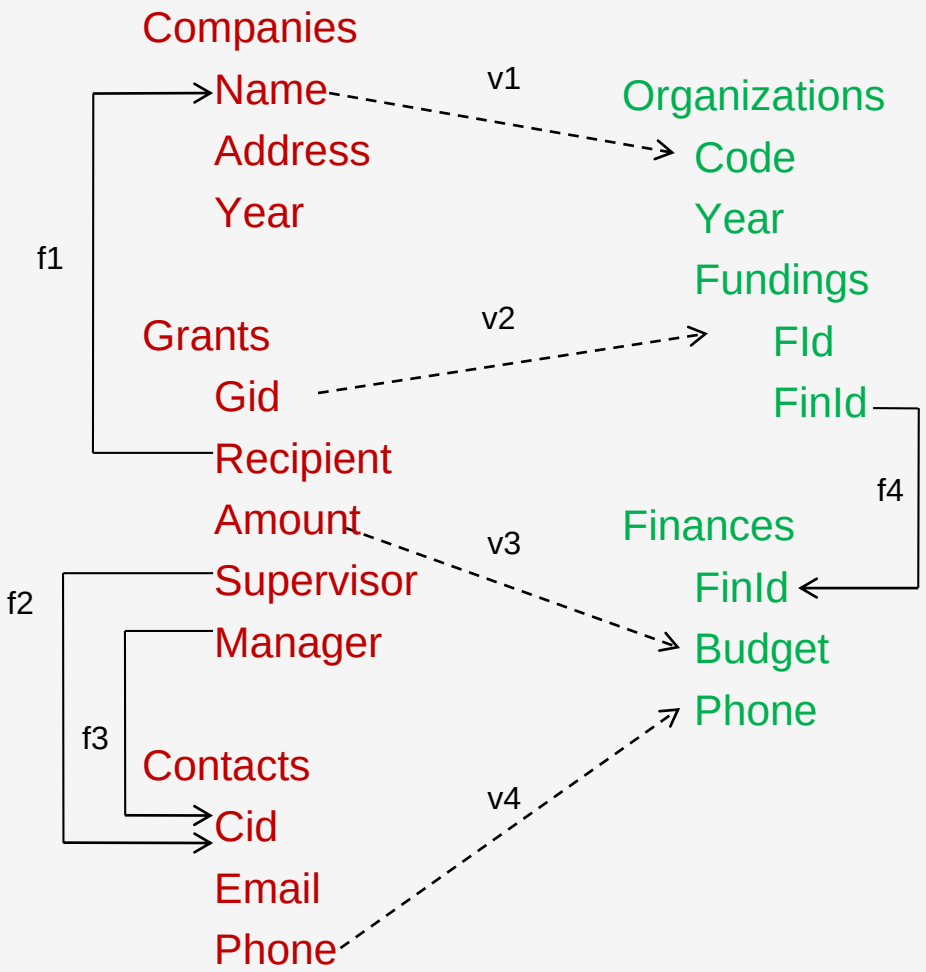




Correspondences (given by a "schema matcher")



Let us formalize correspondences



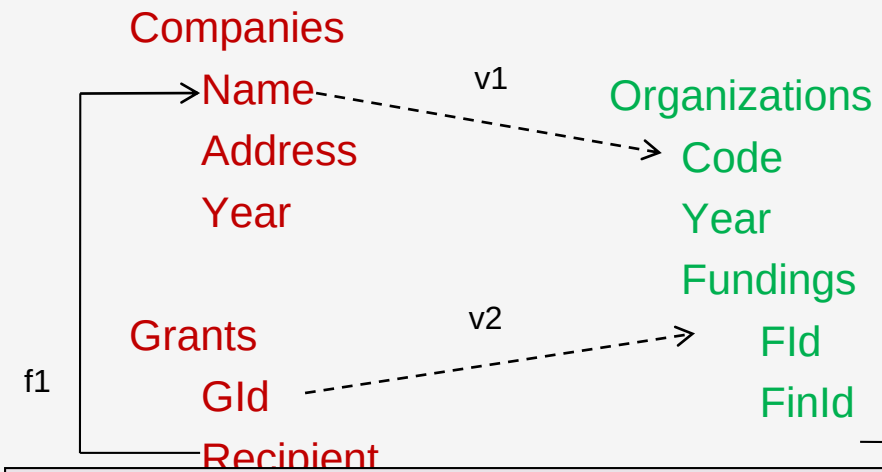
v1 $\forall n,d,y\ \text{Companies}(n,d,y) \rightarrow \exists y',F\ \text{Organizations}(n,y',F))$

v2 $\forall g,r,a,s,m\ \text{Grants}(g,r,a,s,m) \rightarrow \exists c,y,F,f\ \text{Organiz…}(c,y,F)), F(g,f)$

v3 $\forall g, r, a, s, m\ \text{Grants}(g,r,a,s,m) \rightarrow \exists f,p\ \text{Finances}(f,a,p)$

v4 $\forall c, e, p\ \text{Contacts}(c,e,p) \rightarrow \exists f,b\ \text{Finances}(f,b,p)$

Correspondences alone are not enough



v1 ∀n,d,y Companies(n,d,y) →
∃y',F Organizations(n,y',F))

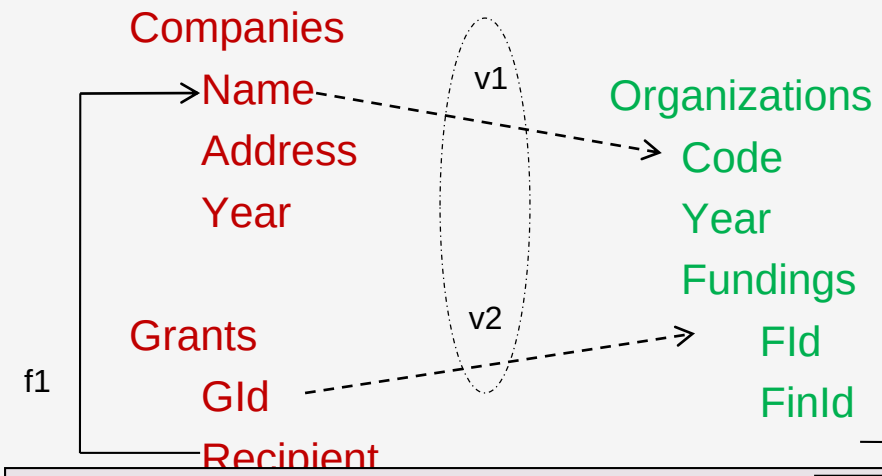
v2 ∀g,r,a,s,m Grants(g,r,a,s,m) →
∃c,y,F,f Organiz…(c,y,F)), F(g,f)

Companies		
Name	Address	Year
HAL	NY	1920
SM	Seattle	1984
PH	SF	1957

Grants		
GId	Rec.t	Amt
301	HAL	30
302	HAL	40
303	PH	30

Organizations			
Code	Year	Fundings	
		FId	FinId
HAL			
SM			
PH			
		301	
		302	

More complex mappings are needed, representing associations



$\forall n,d,y,g,a,s,m$ Companies(n,d,y), Grants(g,n,a,s,m) → $\exists y',F,f$ Organizations(n,y',F)), F(g,f)

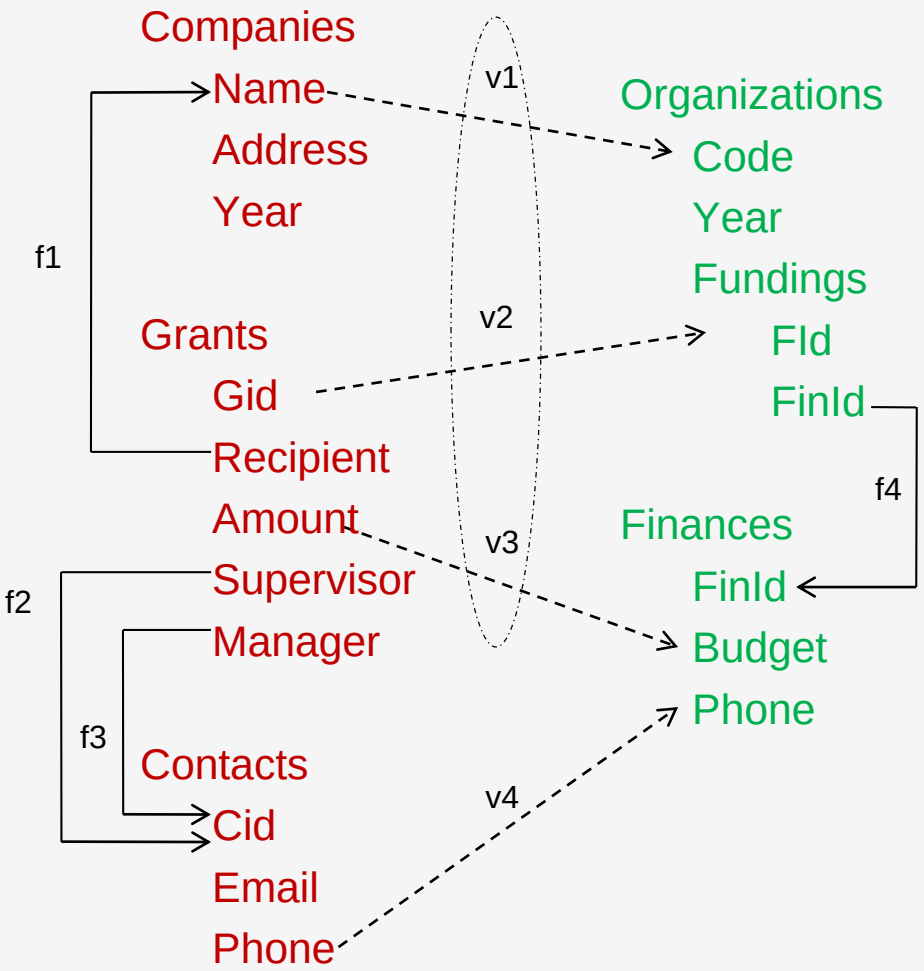
Note: The "association" between companies and grants in the source is suggested by f1 (a foreign key)

Companies		
Name	Address	Year
HAL	NY	1920
SM	Seattle	1984
PH	SF	1957

Grants		
GId	Rec.t	Amt
301	HAL	30
302	HAL	40
303	PH	30

Organizations			
Code	Year	Fundings	
		FId	FinId
HAL		301	
		302	
SM			
PH		303	

Yet more complex

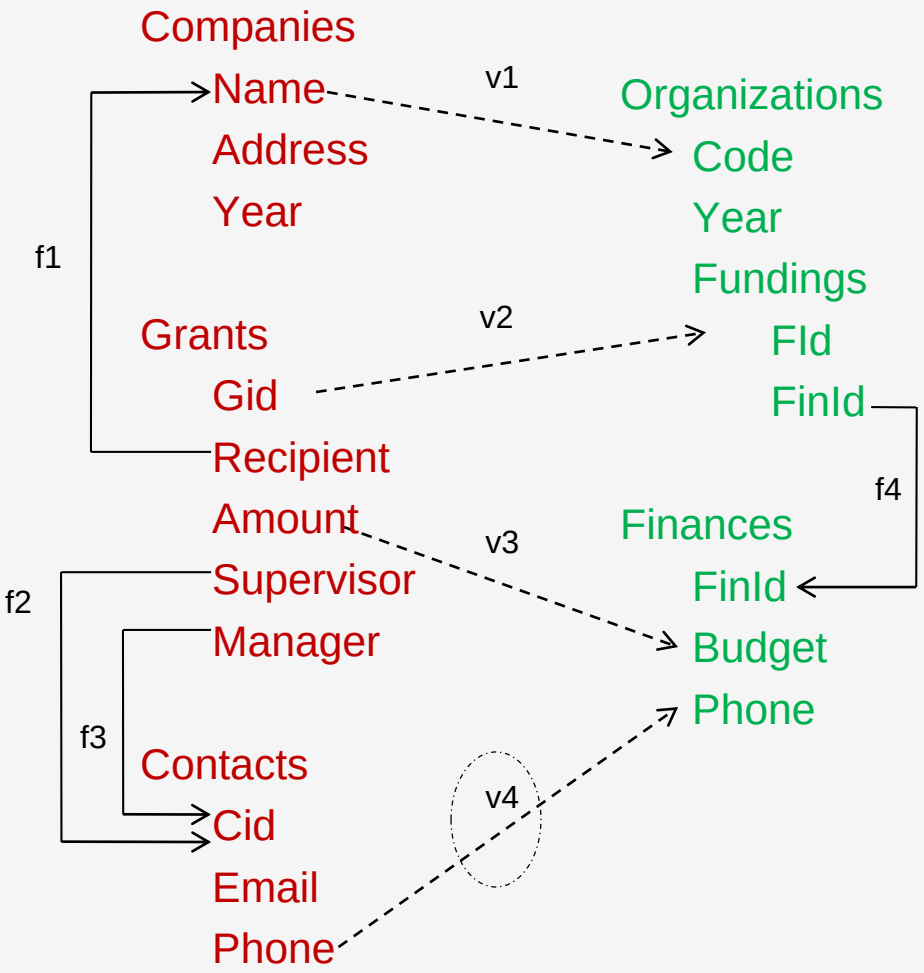


$\forall n,d,y,g,a,s,m$ Companies(n,d,y),
Grants(g,n,a,s,m) $\rightarrow$
$\exists y',F,f,p$
Organizations(n,y',F),
$F(g,f)$,
Finances(f,a,p)

Notes:

- Three tuples are generated for each pair of related companies and grants
- The mapping specifies that there exist an f, appearing in two places, without saying which its value should be

A final issue



- How do we obtain the phone to be put in finances?
  - Is it the supervisor's one or the manager's?
- FKs suggest either (or even both)
- Human intervention is needed to choose

Various solutions in nested cases with possibily undesirable features

Companies		
Name	Address	Year
HAL	NY	1920
SM	Seattle	1984
PH	SF	1957

Grants		
GId	Rec.t	Amt
301	HAL	30
302	HAL	40
303	PH	30

Organizations			
Code	Year	Fundings	
		FId	FinId
HAL		301	k1
		302	k1
SM			
PH		303	k1

Finances		
FinId	Budget	phone
k1	30	
k1	40	
k1	30	

A better solution

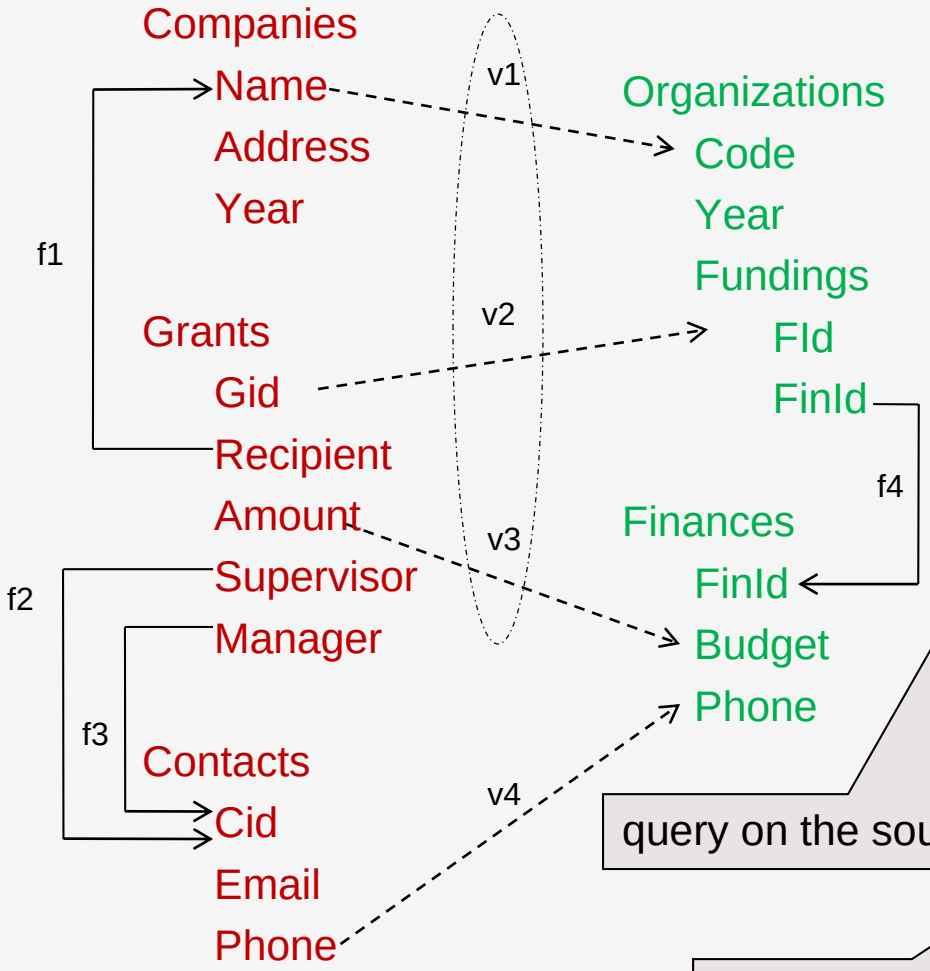
Companies		
Name	Address	Year
HAL	NY	1920
SM	Seattle	1984
PH	SF	1957

Grants		
GId	Rec.t	Amt
301	HAL	30
302	HAL	40
303	PH	30

Organizations			
Code	Year	Fundings	
		FId	FinId
HAL		301	k1
		302	k2
SM			
PH		303	k3

Finances		
FinId	Budget	phone
k1	30	
k2	40	
k3	30	

A more verbose notation for mappings



$$\forall n,d,y,g,a,s,m\ \text{Companies}(n,d,y), \text{Grants}(g,n,a,s,m) \rightarrow \exists y',F,f,p\ \text{Organizations}(n,y',F)), F(g,f), \text{Finances}(f,a,p)$$

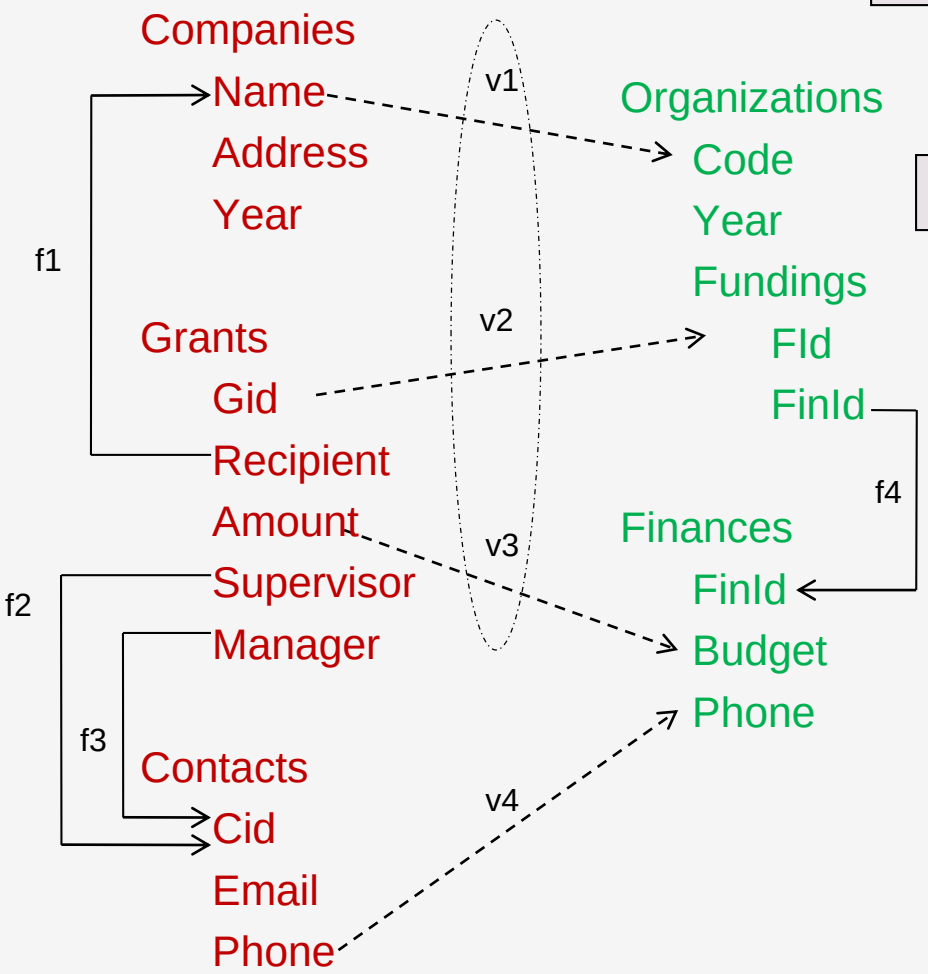
```
foreach c in companies, g in grants
        where c.name=g.recipient
exists  o in organizations,
        f in o.fundings,
        i in finances
        where f.finId = i.finId
with    o.code = c.name
  and   f.fId = g.gId
  and   i.budget = g.amount
```

query on the source

correspondences

query on the target

The mapping as a source-to-target constraint



Q^S

foreach c in companies, g in grants
where c.name=g.recipient

exists o in organizations,
f in o.fundings,
i in finances
where f.finId = i.finId

Q^T

with o.code = c.name
and f.fId = g.gId
and i.budget = g.amount

$$Q^S \rightrightarrows Q^T$$

"the result of Q^T (over the target, projected as in the with-clause) must contain the result of Q^S (over the source, projected as in the with-clause)"

Syntax and restrictions

foreach x_1 in g_1, . . . , x_n in g_n
    where B_1
exists y_1 in g'_1, . . . , y_m in g'_m
    where B_2
with $e_1 = e'_1$ and . . . and $e_k = e'_k$

foreach c in companies, g in grants
    where c.name=g.recipient
exists o in organizations,
        f in o.fundings,
        i in finances
    where f.finId = i.finId
with o.code = c.name
and f.fId = g.gId
and i.budget = g.amount

x_i in g_i (generator)
- x_i variable
- g_i set (either the root or a set nested within it)

B_1 conjunction of equalities over the x_i variables

y_i in g'_i
B_2
similar

$e_1 = e'_1$ … equalities between a source expression and a target expression

Restrictions: See paper, page 210, lines 5+: "The mapping is well formed …"

Schema constraints

- Referential integrity is essential in this approach as the basis for the discovery of "associations"
- Given the nested model, they need a rather complex definition
- So, two steps
  - Paths (primary paths and relative paths)
  - Nested referential integrity (NRI) constraints

Primary paths

- Primary path (given a schema root R, that is a first level element in the schema):
  - x_1 in g_1, x_2 in g_2, …, x_n in g_n
    - where g_1 is an expression on R (just R?), g_i (for $i \geq 2$) g_1 is an expression on x_{i-1}
- Examples
  - c in companies
  - o in organizations
  - o in organizations, f in o.fundings

Relative paths

- Primary path (given a schema root R, that is a first level element in the schema):
  - x_1 in g_1, x_2 in g_2, …, x_n in g_n
    - where g_1 is an expression on R (just R?), g_i (for $i \geq 2$) g_1 is an expression on x_{i-1}
- Relative path with respect to a variable x
  - x_1 in g_1, x_2 in g_2, …, x_n in g_n
    - where g_1 is an expression on x (just x?), g_i (for $i \geq 2$) g_1 is an expression on x_{i-1}
- Example
  - f in o.fundings

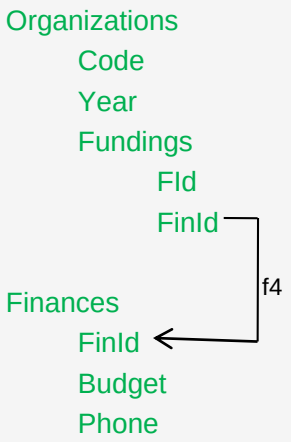
Nested referential integrity (NRI) constraints

- foreach P_1 exists P_2 where B
  - P_1 is a primary path
  - P_2 is either a primary path or a relative path with respect to a variable in P_1
  - B is a conjunction of equalities between an expression on a variable of P_1 and an expression on a variable of P_2
- Example

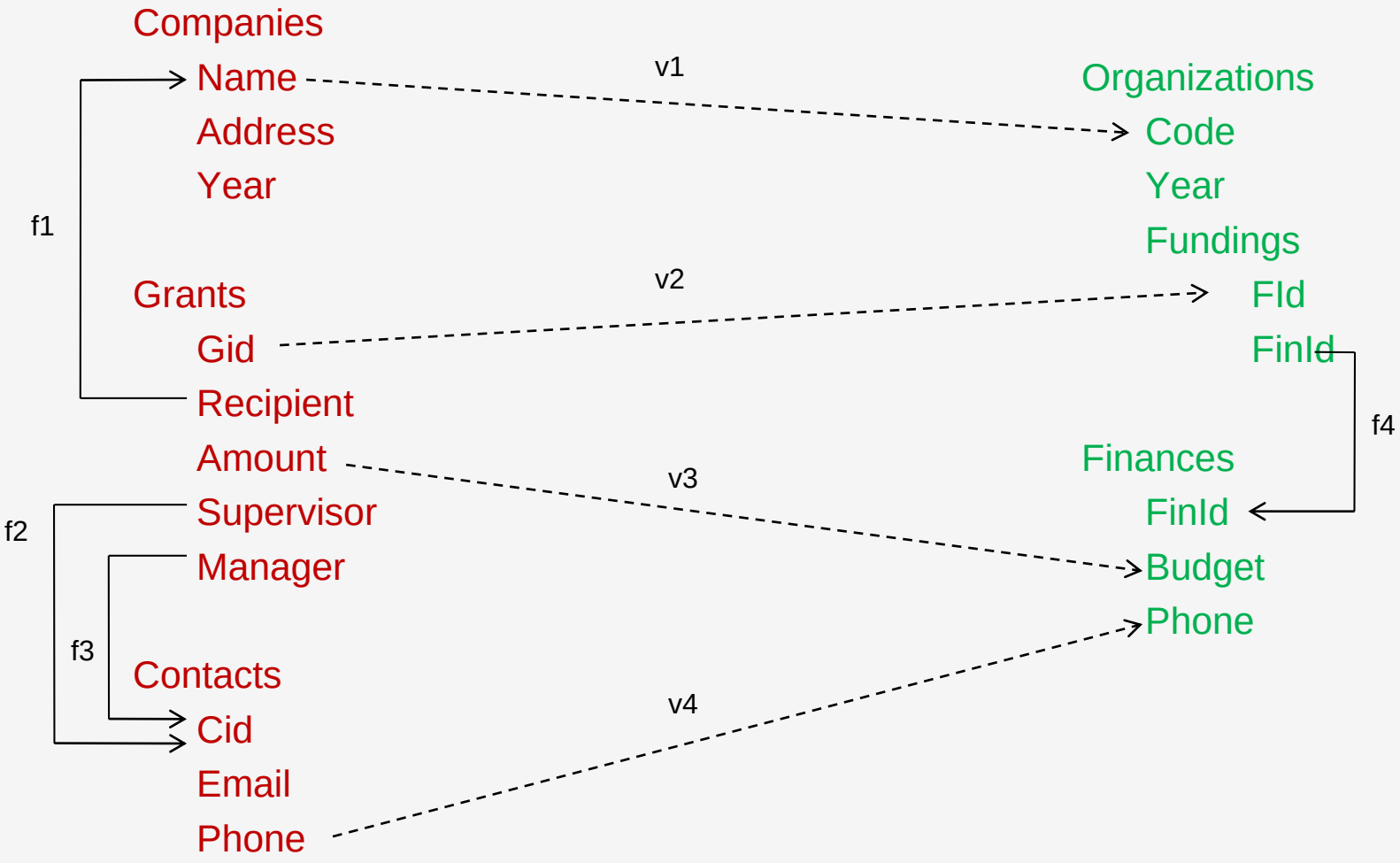
  foreach o in organizations, f in o.fundings

  exists i in finances

  where f.finId = i.finId



The context


Companies
Name
Address
Year
Grants
Gid
Recipient
Amount
Supervisor
Manager
Contacts
Cid
Email
Phone
Organizations
Code
Year
Fundings
FId
FinId
Finances
FinId
Budget
Phone
v1
v2
v3
v4
f1
f2
f3
f4

Associations

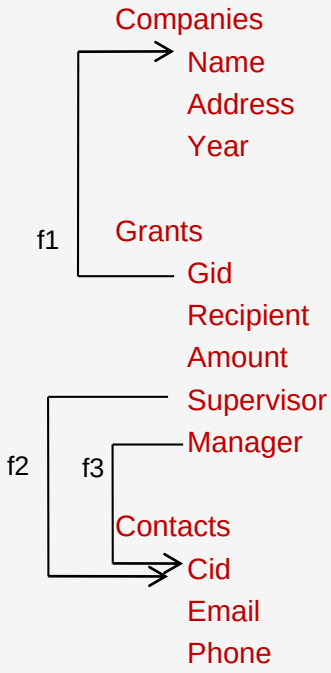
from x_1 in g_1, x_2 in g_2, …, x_n in g_n
[where B]

- x_i in g_i generator (each expression may include variables defined in a previous generator)
- B a conjunction of equalities (with variables and constants)

- Examples
  - from c in contacts
  - from g in grants, c in companies, s in contacts, m in contacts
    where g.recipient = c.name and
    g.supervisor = s.cid and
    g.manager = m.cid

Associations

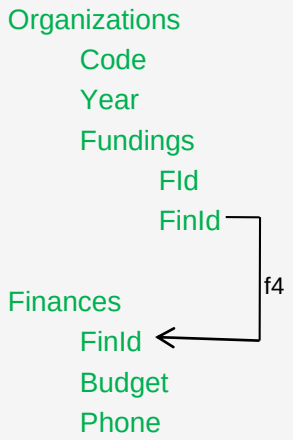
- In the (flat) relational model, an association is a join (possibly with a selection)
  - from c in contacts
  - from g in grants, c in companies, s in contacts, m in contacts
    where g.recipient = c.name and
    g.supervisor = s.cid and
    g.manager = m.cid



Dominance and union

- A_2 **dominates** A_1 ($A_1 \leq A_2$) if
  - the from and where clauses of A_1 are subsets of those of A_2 (after suitable renaming and with other technicalities)
- Example
  - A_2 : from g in grants, c in companies, s in contacts, m in contacts
    where g.recipient = c.name and
    g.supervisor = s.cid and
    g.manager = m.cid
  - A_1 : from g in grants, c in companies
    where g.recipient = c.name
- Union of associations:
  - Union of from and of where (with renamings if needed)

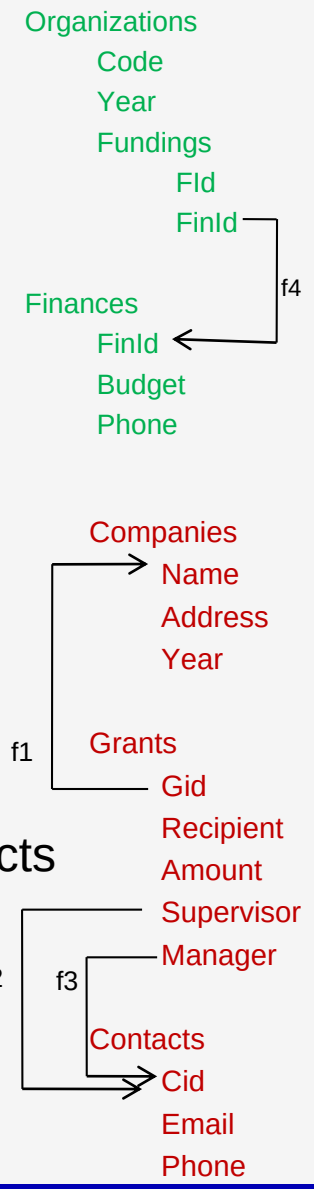
Useful associations



- **Structural association**:
  - from P with P primary path
  - from o in organizations, f in o.fundings
- **User association**
  - Any association (specified by the user)
- **Logical association**
  - An association obtained by "chasing" constraints (starting with a structural or a user association)
    - from o in organizations, f in o.fundings, i in finances where f.finId=i.finId

Logical associations

- from o in organizations, f in o.fundings NO
- from o in organizations, f in o.fundings, i in finances
  where f.finId=i.finId SÌ
- from c in companies SÌ
- from g in grants, c in companies
  where g.recipient = c.name NO
- from g in grants, c in companies, s in contacts, m in contacts
  where g.recipient = c.name and
  g.supervisor = s.cid and
  g.manager = m.cid SÌ



The chase

- Given as association, repeatedly applying a chase rule to the "current" association (initialed as the input one)
  - If there is a NRI constraint

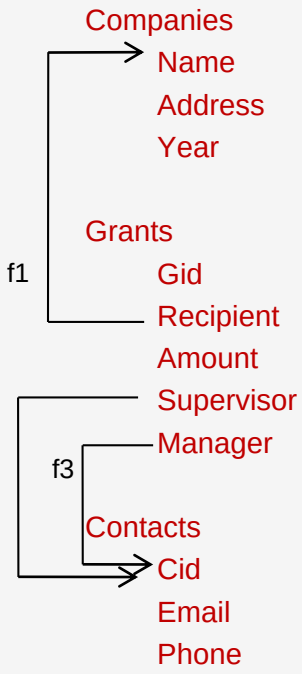
    foreach X exists Y where B

    such that (this is a bit informal but intuitive) the "current" association contains X and does not contain a Y that satisfies B then add Y to the generators and B to the where clause
- Example. If we start with

  from g in grants

  then we have to add various components and obtain

  from g in grants, c in companies,
  s in contacts, m in contacts
  where g.recipient = c.name and
  g.supervisor = s.cid and
  g.manager = m.cid
- If the NRIs are acyclic, then the chase terminates and the result does not depend on the order of application



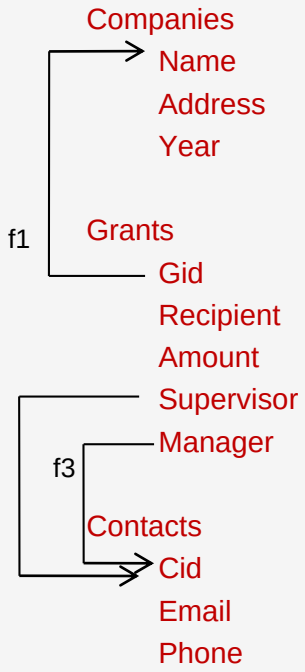
Mapping generation

- Logical associations are meaningful combinations of correspondences
- A set of correspondences can be interpreted together if there are two logical associations (one in the source and one in the target) that cover them
- The algorithm for generating schema mappings
  - Finds maximal sets of correspondences that can be interpreted together
  - Compares pairs of logical association (one in the source and the other in the target)
  - Select a suitable set of pairs

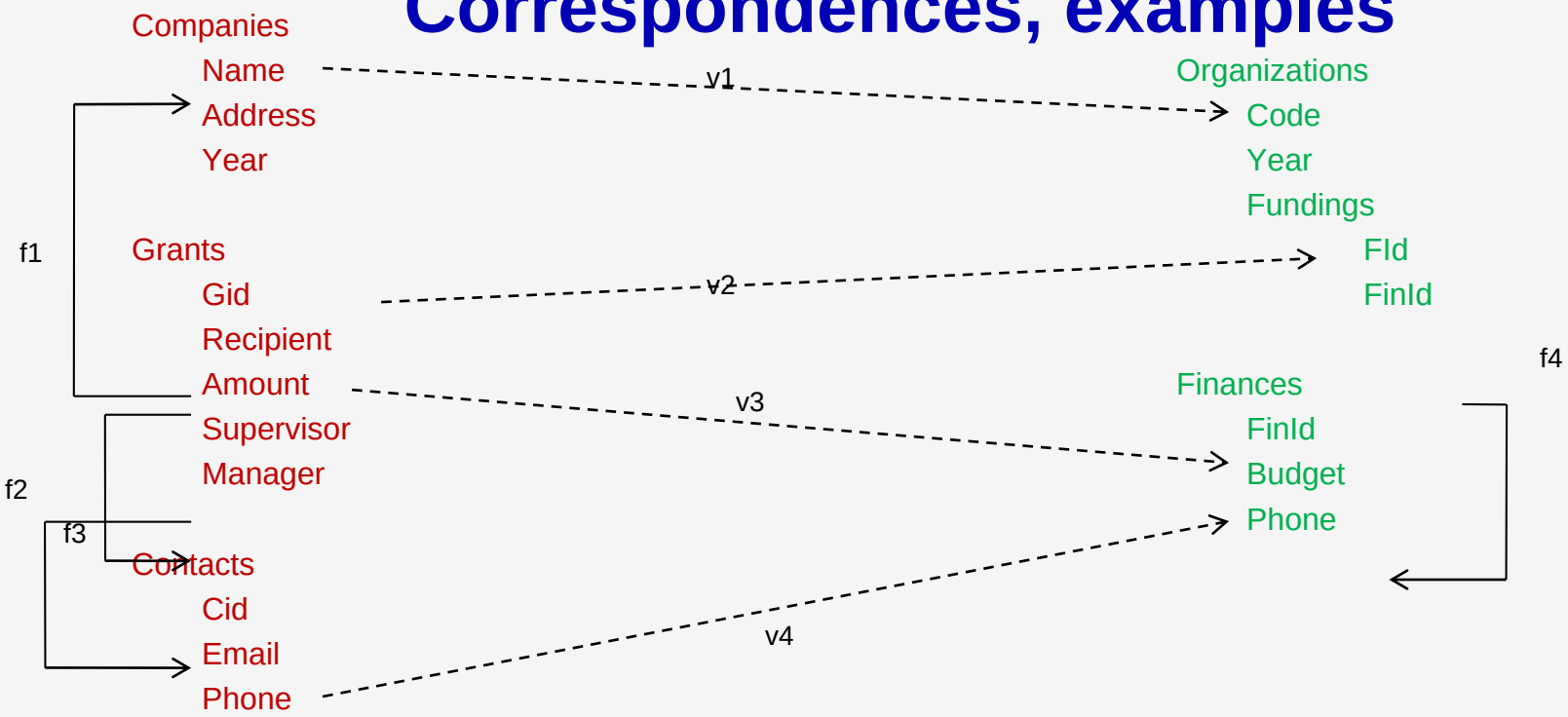
Correspondences

- <P;e> **schema element** (an attribute somewhere)
  - P primary path
  - e expression on the last variable of P
- Examples
  - <c in companies; c.name>
  - <o in organizations, f in o.fundings; c.name>
- **Correspondence**:

  for each P^S exists P^T with e_S=e_T
  - with <P^S; e_S > and <P^T; e_T> schema elements
- Example (v1)
  - for each c in companies exists o in organizations with c.name = o.code



Correspondences, examples



v1: for each o in companies
exists o in organizations
with c.name = o.code

v2: for each g in grants
exists o in organizations ,
f in o.fundings
with g.gId = f.fId

v3: for each g in grants
exists i in finances,
with g.amount= i.budget

v4: for each c in contacts
exists i in finances
with c.phone = i.phone

Correspondences and associations

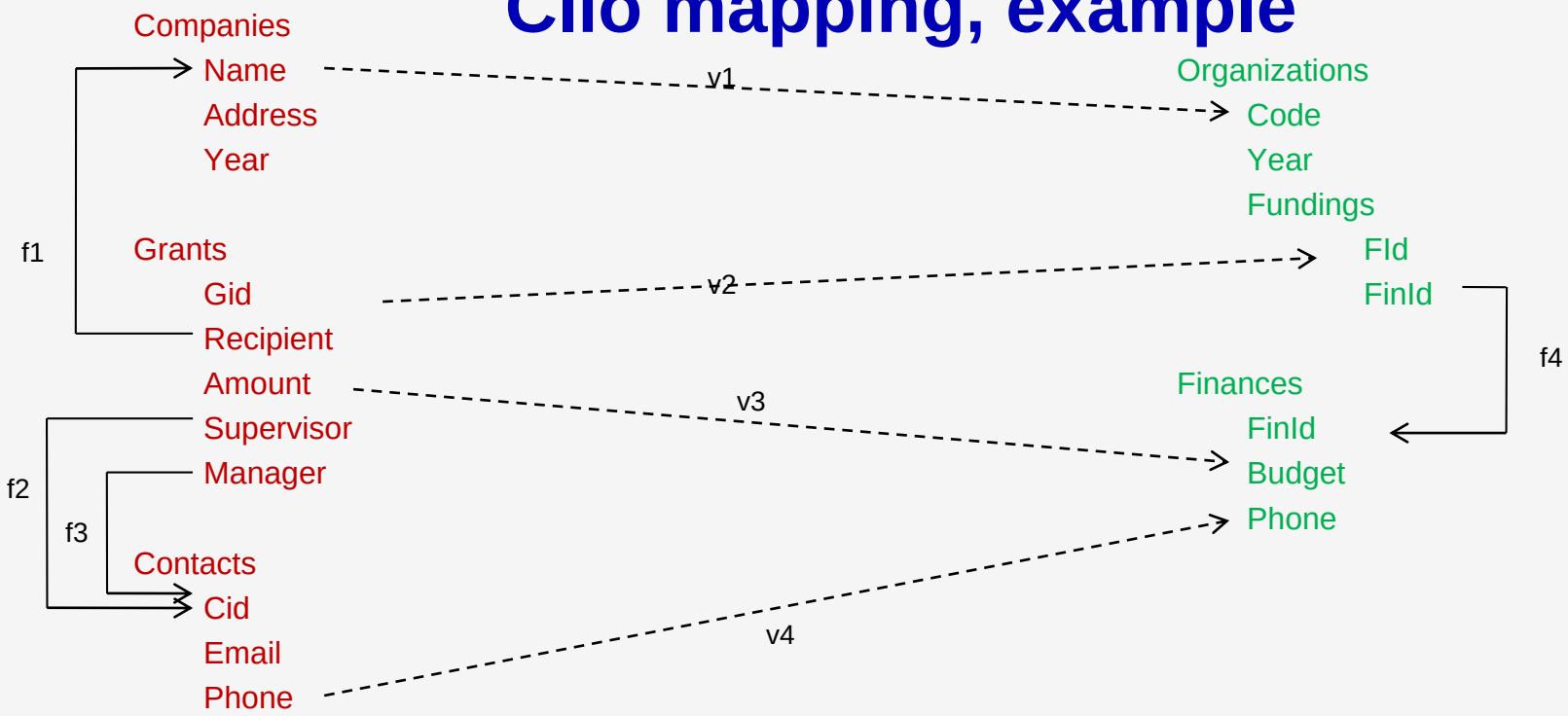
- A correspondence
  v : for each P^S exists P^T with $e_S=e_T$
  is **covered** by a pair of associations (on source and target, resp.) $<A^S , A^T>$ if $P^S \leq A^S$ and $P^T \leq A^T$ with some renaming h, h' (on source and target, resp.)
- We say that
  - there is a **coverage** of v by $<A^S , A^T>$ via <h,h'>
  - the **result** of the coverage is $h(e_S)=h'(e_T)$

Clio mapping

- Given
  - S, T source and target schemas
  - C set of correspndences
- A **Clio mapping**:

  for each A^S exists A^T with E
  - A^S A^T logical associations (on source and target, resp.)
  - E a conjunction of equalities:
    - for each correspondence v in C covered by $<A^S , A^T>$, E includes the equality $h(e_S)=h(e_T)$ which is the **result** of the coverage, for one of the coverages

Clio mapping, example




```
from g in grants, c in companies, s
     in contacts, m in contacts
where  g.recipient = c.name
     and g.supervisor = s.cid
     and g.manager = m.cid
```

```
from o in organizations,
     f in o.fundings, i in finances
where  f.finId = i.finId
```

v1, v2, v3 are covered

Clio mapping, example

```
from g in grants, c in companies,
     s in contacts, m in contacts
where  g.recipient = c.name
     and g.supervisor = s.cid
     and g.manager = m.cid
```

```
from o in organizations,
     f in o.fundings, i in finances
where  f.finId = i.finId
```

for each g in grants, c in companies, s in contacts, m in contacts
where g.recipient = c.name and g.supervisor = s.cid and g.manager = m.cid

exists o in organizations, f in o.fundings, i in finances
where f.finId = i.finId

with c.name = o.code and g.gId = f. fId and g.amount = i.budget

Clio mappings, more

v4: for each c in contacts
exists i in finances
with c.phone = i.phone

for each g in grants, c in companies, s in contacts, m in contacts
where g.recipient = c.name and g.supervisor = s.cid
and g.manager = m.cid
exists o in organizations, f in o.fundings, i in finances
where f.finId = i.finId
with c.name = o.code and g.gId = f. fId and g.amount = i.budget
and m.phone = i.phone

for each g in grants, c in companies, s in contacts, m in contacts
….
and s.phone = i.phone

Mapping generation algorithm

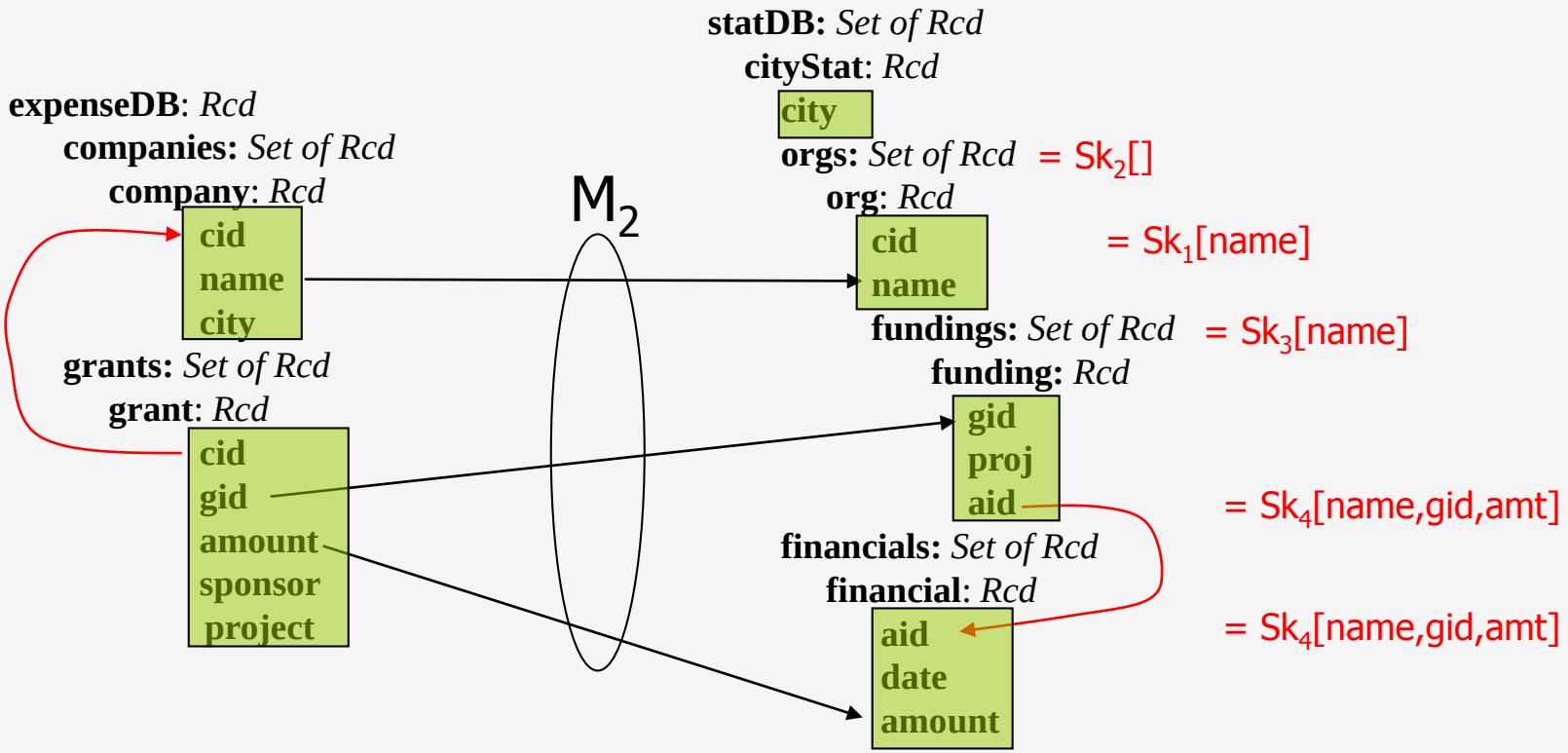
Input: A source schema $\mathcal{S}$
A target schema $\mathcal{T}$
A set of correspondences $\mathcal{C}$
Output: The set of all Clio mappings $\mathcal{M}$

GENERATEMAPPINGS($\mathcal{S}, \mathcal{T}, \mathcal{C}$)
(1) $\mathcal{M} \leftarrow \emptyset$
(2) $\mathcal{A}^{\mathcal{S}} \leftarrow$ Logical Associations of $\mathcal{S}$
(3) $\mathcal{A}^{\mathcal{T}} \leftarrow$ Logical Associations of $\mathcal{T}$
(4) **foreach** pair $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ of $\mathcal{A}^{\mathcal{S}} \times \mathcal{A}^{\mathcal{T}}$
(5) $V \leftarrow \{v \mid v \in \mathcal{V} \wedge v$ is covered by $<A^{\mathcal{S}}, A^{\mathcal{T}}> \}$
(6) *// If no correspondences are covered*
(7) **if** $V = \emptyset$
(8) continue;
(9) *// Check if subsumed*
(10) **if** $\exists$ <X,Y> with $X \dot{\preceq} A^{\mathcal{S}}$ or $Y \dot{\preceq} A^{\mathcal{T}}$, and at least one dominance is strict
(11) $V' \leftarrow \{v \mid v \in \mathcal{C} \wedge v$ covered by $<X, Y> \}$
(12) **if** $V' = V$
(13) continue;
(14) let V be $\{v_1, \ldots, v_m\}$
(15) for every v_i: let Δ_{v_i} be $\{<h, h'> \mid v_i$ covered by $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ via $<h, h'>\}$
(16) *// For every combination of correspondence coverages*
(17) **foreach** $(\delta_1, \ldots, \delta_m) \in \Delta_{v_1} \times \ldots \times \Delta_{v_m}$
(18) $W \leftarrow \emptyset$
(19) **foreach** $v_i \in V$
(20) let $e = e'$ be the equality in v_i
(21) let δ_i be $<h, h'>$
(22) add equality $h(e) = h'(e')$ to W
(23) form Clio mapping M: foreach $A^{\mathcal{S}}$ exists $A^{\mathcal{T}}$ with W
(24) $\mathcal{M} \leftarrow \mathcal{M} \cup \{M\}$
(25) **return** $\mathcal{M}$

Data Exchange

```
<statisticsDB>
  {   FOR  $x0 IN /expenseDB/grant, $x1 IN /expenseDB/project, $x2 IN /expenseDB/company
      WHERE
        $x2/cid/text() = $x0/cid/text()
        $x0/project/text() = $x1/name/text()
      RETURN
        <cityStatistics>
        {   FOR $x0L1 IN /expenseDB/grant, $x1L1 IN /expenseDB/project,  $x2L1 IN /expenseDB/company
            WHERE
              $x2L1/cid/text() = $x0L1/cid/text()
              $x0L1/project/text() = $x1L1/name/text()
              $x2/city/text() = $x2L1/city/text()
            RETURN
              <organization>
                  <cid>  { $x0L1/cid/text() }  </cid>
                 <cname>  { $x2L1/name/text() }  </cname>
                 {
                   FOR  $x0L2 IN /expenseDB/grant, $x1L2 IN /expenseDB/project, $x2L2 IN /expenseDB/company
                   WHERE
                       $x2L2/cid/text() = $x0L2/cid/text()
                       $x0L2/project/text() = $x1L2/name/text()
                       $x2L1/name/text() = $x2L2/name/text()
                       $x2L1/city/text() = $x2L2/city/text()
                       $x0L1/cid/text() = $x0L2/cid/text()
                  RETURN
                     <funding>
                           ……..
```

Query Generation



- Correspondences map only into some of the *atomic* attributes
- We use *Skolem functions* to control the creation of the other elements
  - sets (this controls how we group elements in the target)
  - atomic values (this enforces the integrity of the target)