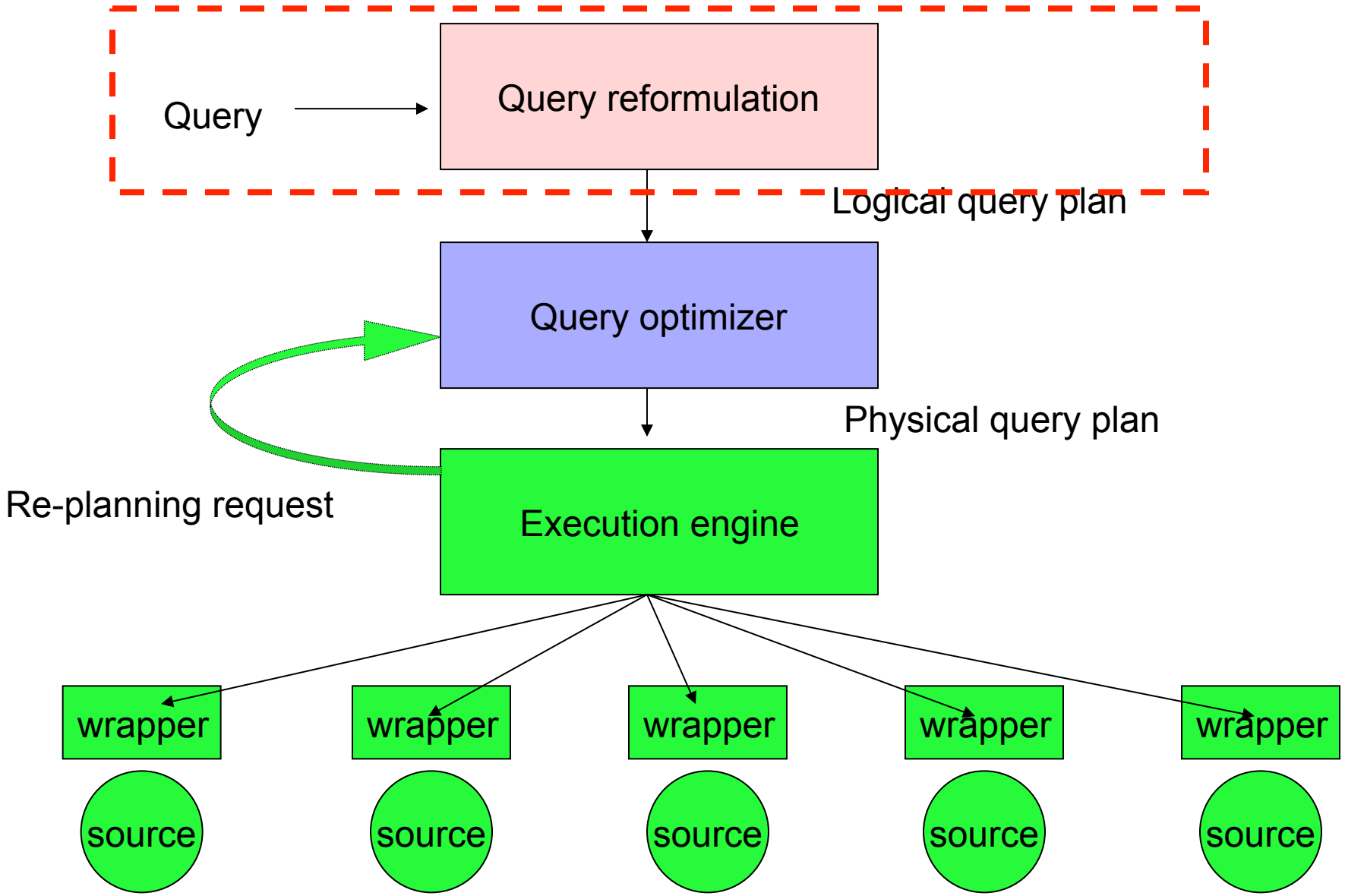
Querying Heterogeneous Data Sources

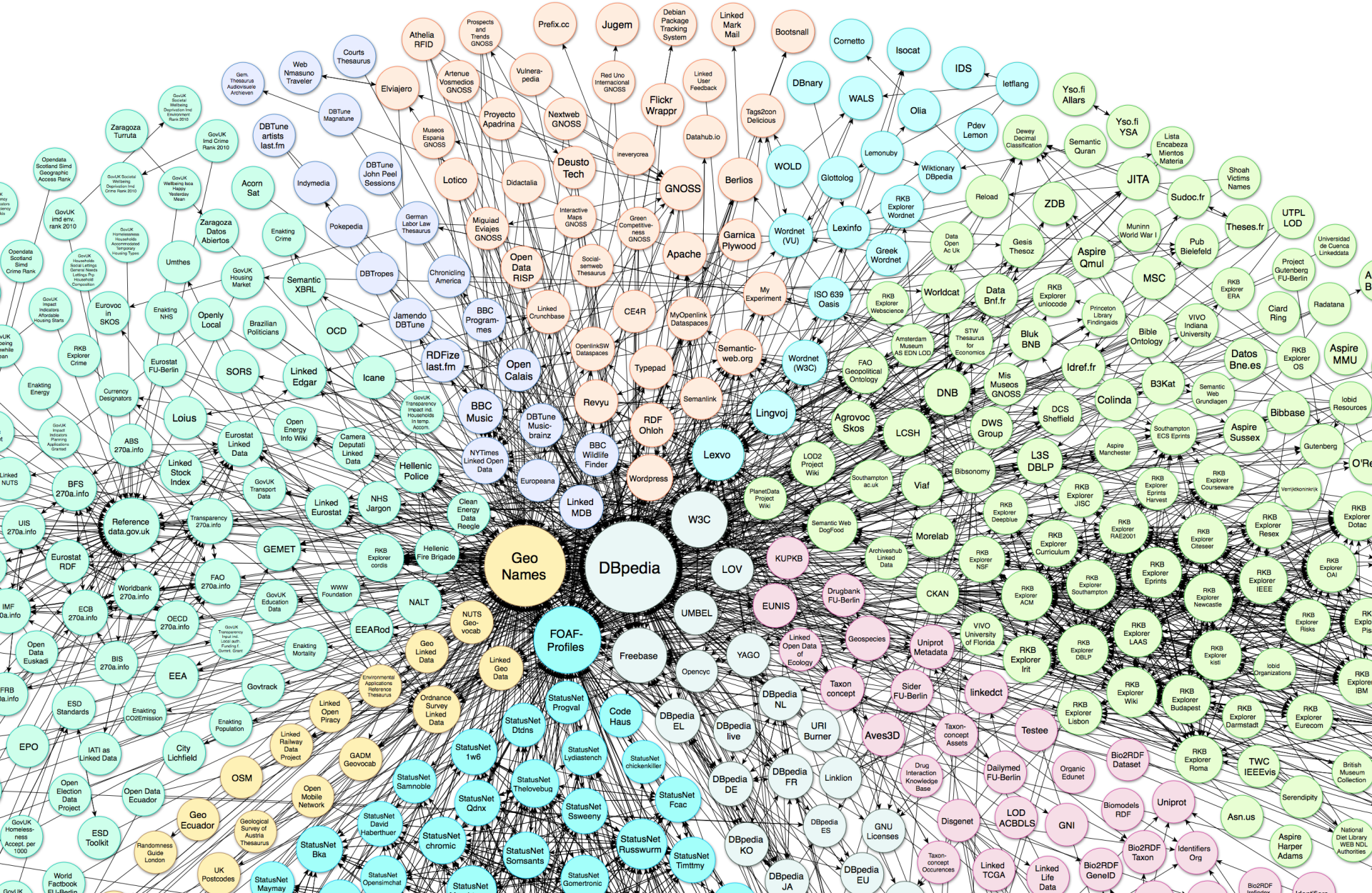
Outline

- Federated Databases
- Source Descriptions
- Answering Queries Using Views

Federated Databases



A Possible Application: Linked Data



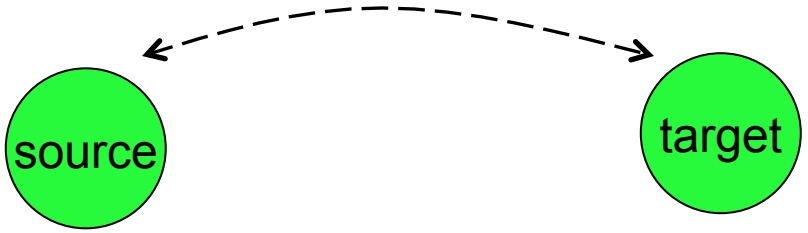
Issues: Semantic Heterogeneity

- Differences in:
  - Naming of schema elements
  - Organization of tables
  - Coverage and detail of schema
  - Languages (XML vs. Relational)
  - Data-level representation (sculpture description in YAGO vs. Musée Rodin DB)

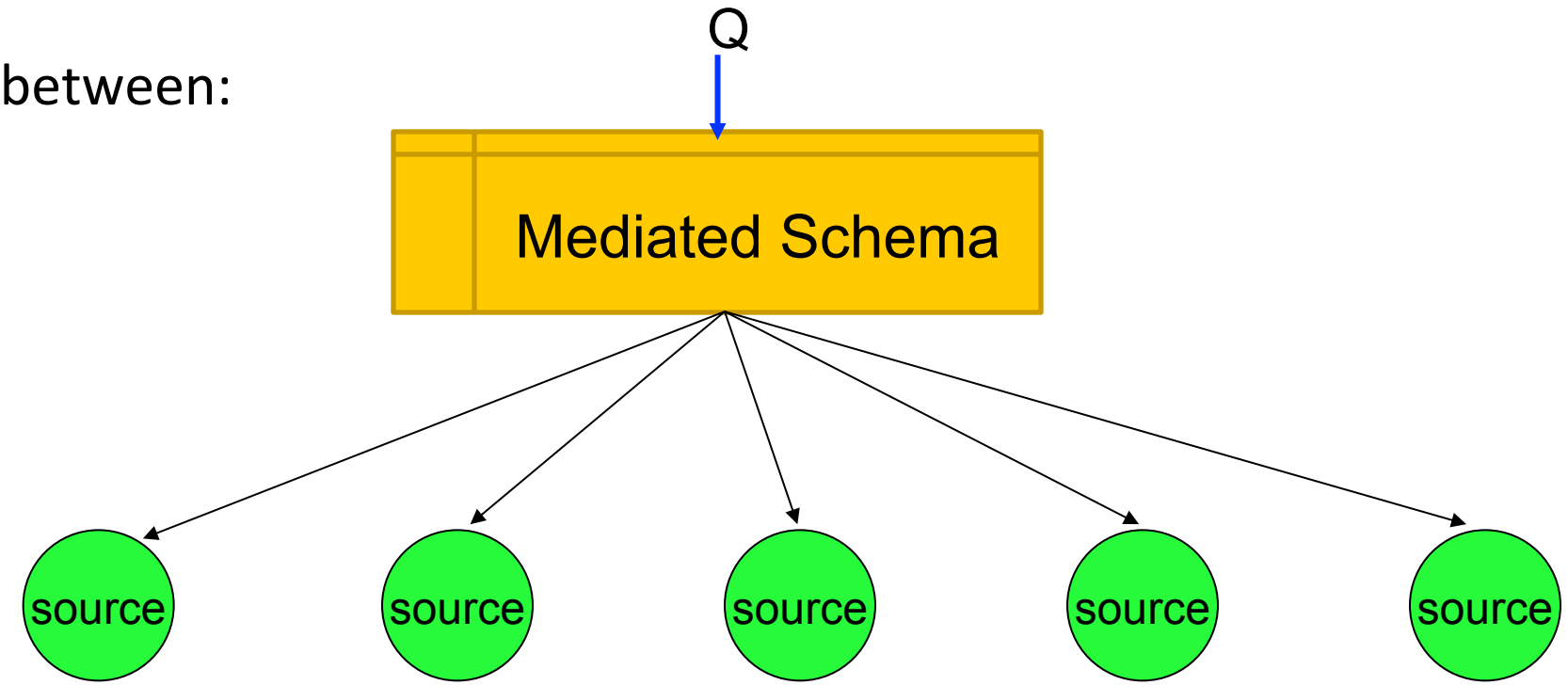
Schema Mappings

Principles of Schema Mappings

Schema mappings describe the relationship between:



Or between:



Question

How does the system know:

- Which source is relevant to the query?
- How to access the source appropriately?

Source Description

Descriptions of data sources should enable the system to:

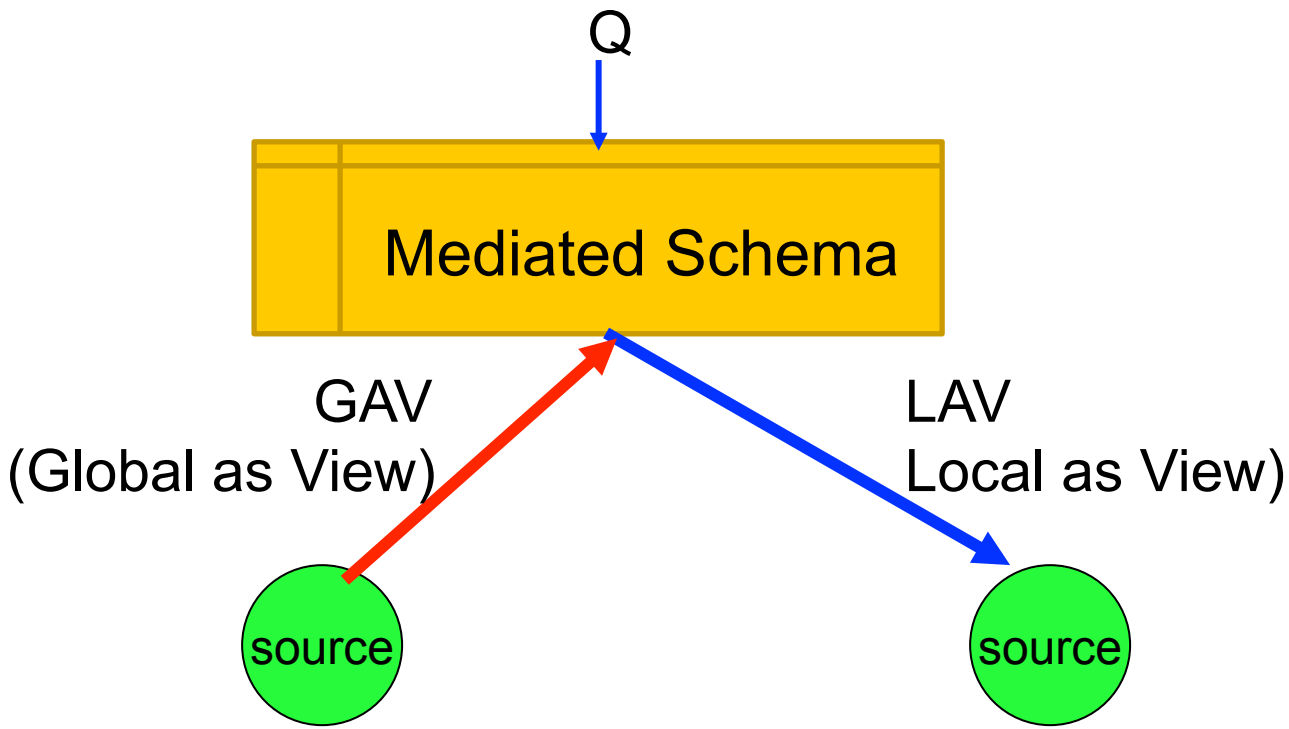
- Determine which sources are relevant to a query
- Access the sources appropriately
- Combine data from multiple sources
- Overcome limitations that specific sources may have
- Identify opportunities for more efficient query evaluation
- Possible access patterns to the data source:
  - Are there required inputs? (e.g., web forms, web services)
  - Can the source process database queries?
- Source completeness:
  - Is the source complete, or partially complete?
- Load restrictions, Quality Of the Service, Quality Of the Data ..

Source Description Languages

Desiderata:

- Effective: it should be able to express relationships between real schemata
- Enable efficient reformulation: computational complexity of reformulation and finding answers
- Easy update: it should be easy to add and delete sources

Languages for Schema Mapping

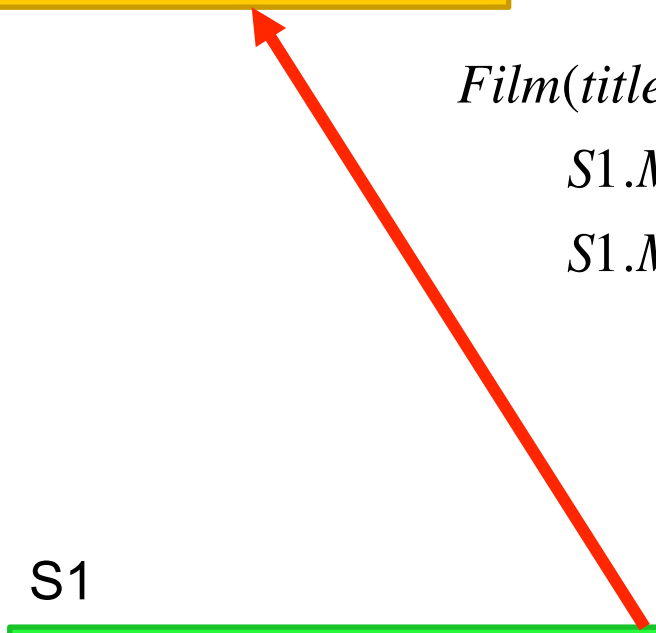


Global-as-View (GAV)

Mediated schema defined as a set of views over the data sources

Film(title, director, year, genre)

$$Film(title, director, year, genre) \supseteq$$
$$S1.Movie(MID, title),$$
$$S1.MovieDetail(MID, director, genre, year)$$



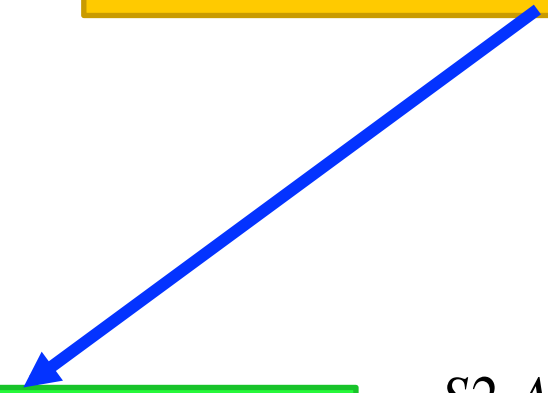
S1

Movie(MID,title)
Actor(AID, firstName, lastName, nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

Local-as-View (LAV)

Data sources defined as views over mediated schema!

Movie(title, director, year, genre)
Actors(title, name)
Plays(movie, location, startTime)
Reviews(title, rating, description)



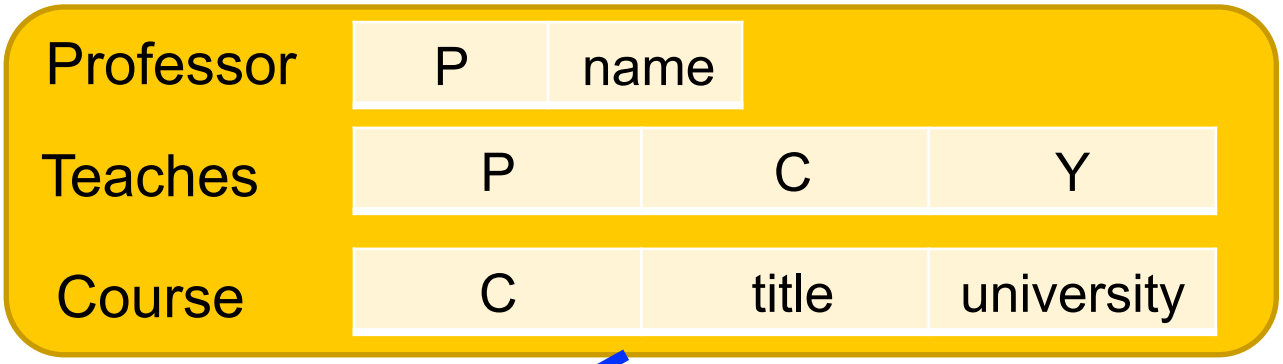
S2:

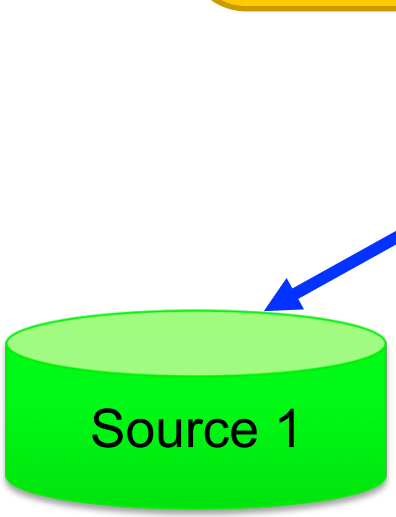
ActorDirectors(actor,dir)

$$S2.ActorDirectors(actor, dir) \subseteq$$
$$Movie(title, dir, year, genre), Actor(title, actor)$$
$$year \geq 1980$$

Local-As-View (LAV)

Data sources defined as views over mediated schema!





UVSQProfs

```
Create View    UVSQProfs  As
Select         Course.title, Profesor.name
From           Course, Teaches, Professor
Where          Course.C = Teaches.C and
               Teaches.P = Professor.P and
               Course.university = “UVSQ”
```

Specification of Source Description

GAV: Global as view

- Each mediator relation defined as a view over the sources
- Explains how to compute the extensions of mediator relations using the data of sources

LAV: Local as view

- Each source defined as a view over the mediated schema
- Answer a query Q over global DB is answer Q using the views

Answering Queries

Answering Queries over Possible Instances of Mediated Schema:

- Certain Answer: An answer is certain if it is true in every instance of the mediated schema that is consistent with: (1) the instances of the sources and (2) the mapping *M*.

Examples: Certain Answers (1)

$$S.ActorDirectors(actor, dir) \subseteq$$
$$Movie(title, dir, year, genre), Actor(title, actor), year \geq 1980$$

S: Keaton Allen

$$Q(actor, dir) :-$$
$$Movie(title, dir, year, genre), Actor(title, actor)$$

Only one certain answer: (Keaton, Allen)

Examples: Certain Answers (2)

$V_1(dir) = DirectorActor(ID, dir, actor)$ Keaton

$V_2(actor) = DirectorActor(ID, dir, actor)$ Allen

$Q(dir, actor) :- Director(ID, dir), Actor(ID, actor)$

Under closed-world assumption:
single DB possible $\Rightarrow$ (Allen,Keaton)

Under open-world assumption:
no certain answers.

Comparison

- Query Reformulation:
  - GAV: Query/View Unfolding (easy)
  - LAV: the problem of answering queries using views
- Incompleteness:
  - GAV: does not express incomplete information. Only a single instance of the mediated schema is consistent with sources.
  - LAV: expresses incomplete information

Answering Queries using Views

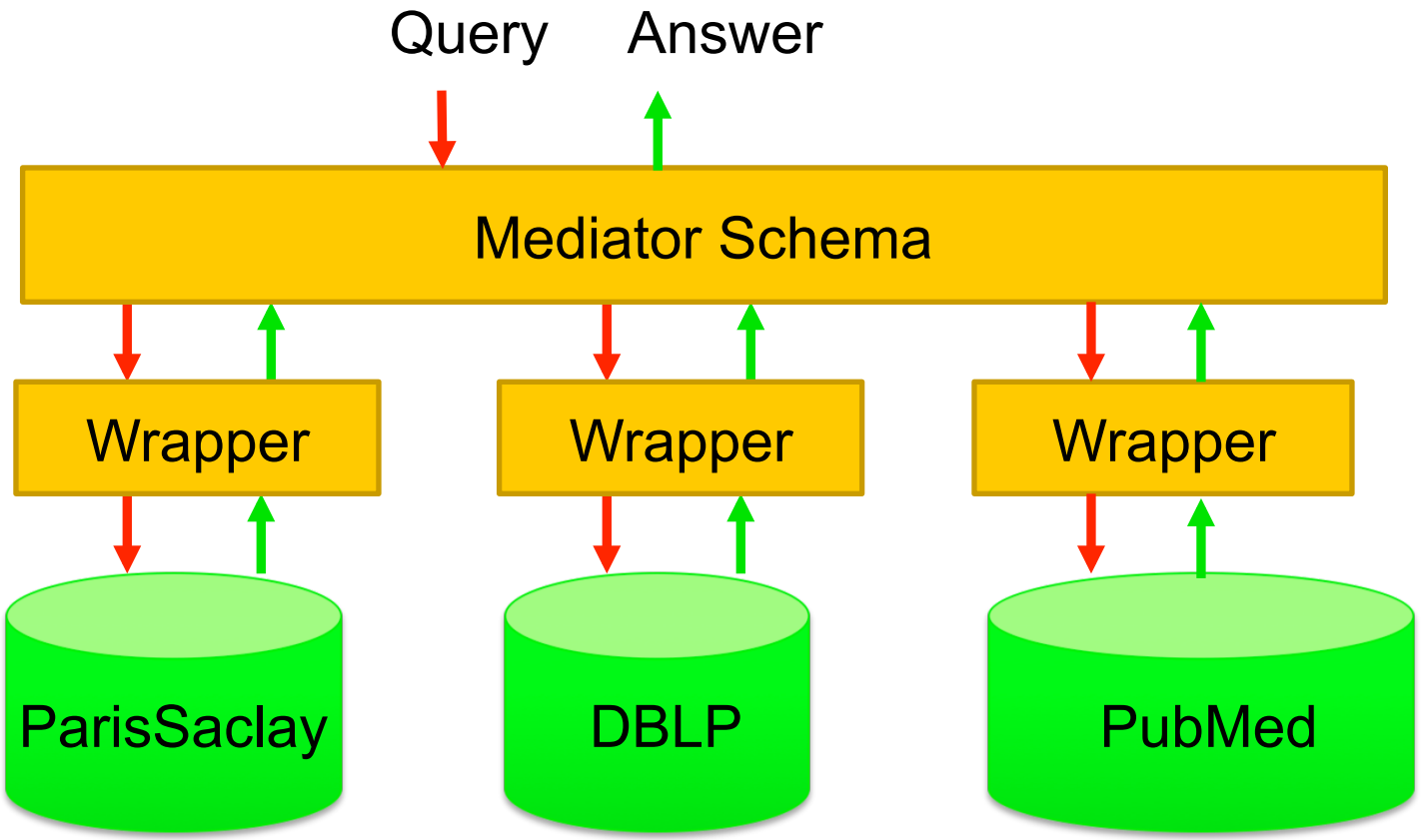
after the survey Alon Y. Levy. *Answering Queries Using Views: A Survey*.

Answering Queries Using Views

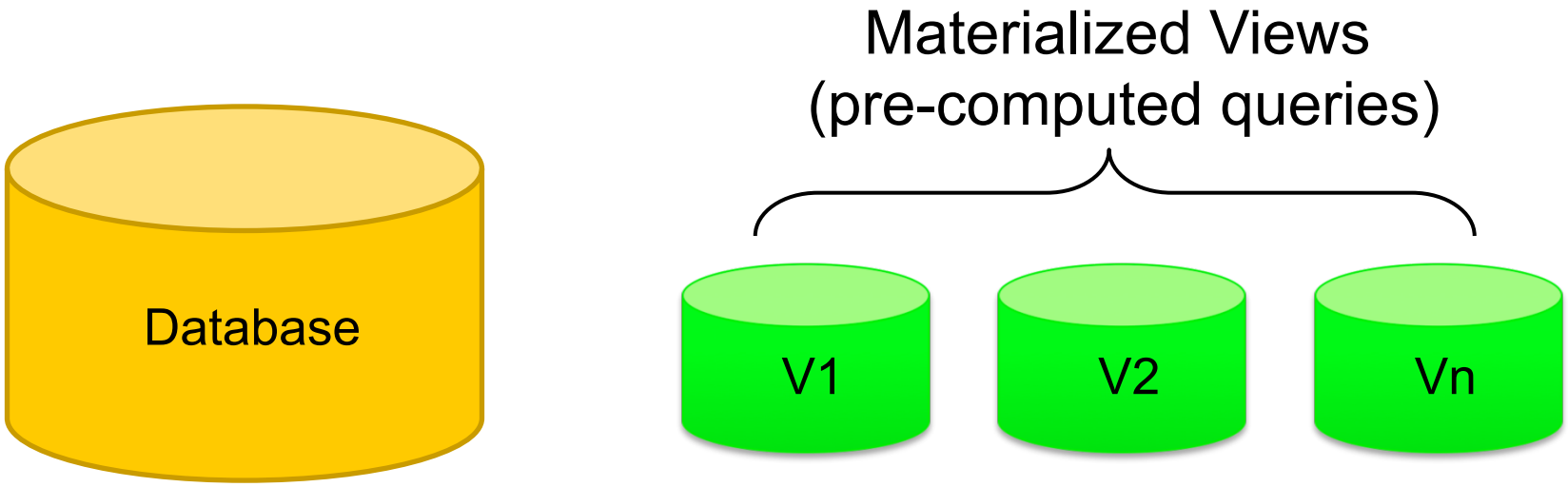
- Motivating Examples
- Problem Definition
- Two different settings:
  - Data Integration
  - Query Optimization & Physical Data Independence

Scenario 1: Data Integration

Goal: Uniform access to data sources e.g. Amazon, IMDB



Scenario 2: Query Optimization



Given a query Q against the Database use the pre-computed results of V1, V2 …Vn to reduce the evaluation time!

Answering Queries Using Views

- Motivating Examples
- Problem Definition
- Two different settings:
  - Data Integration
  - Query Optimization & Physical Data Independence

Query Equivalence and Containment

- Query Containment:

Q1 is contained in Q2 (Q1⊆Q2), if for all database instances D,

$Q1(D) \subseteq Q2(D)$

- Query Equivalence:

Q1 is equivalent to Q2 if Q1⊆Q2 and $Q2 \subseteq Q1$

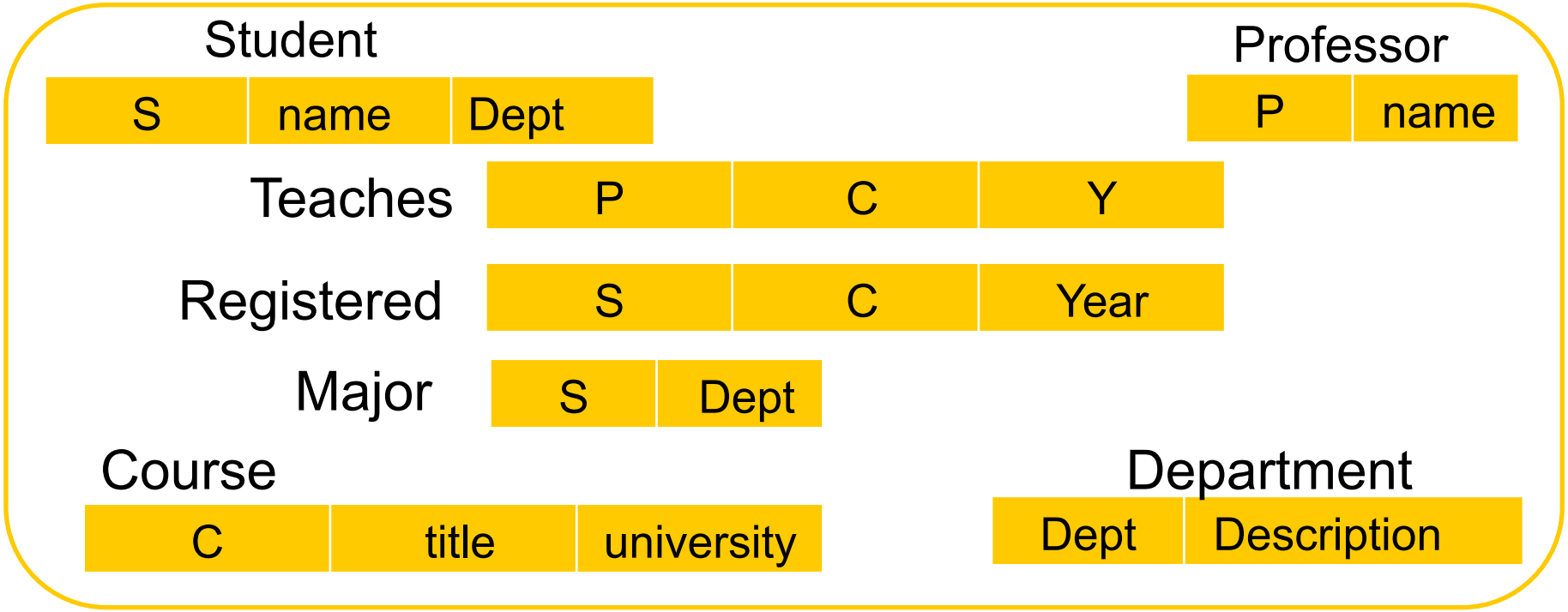
Equivalent and Containment Rewritings

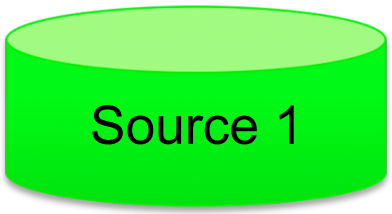
Let A be a schema and V={V1, V2, …, Vm} a set of views over A and Q be a query over A

- Q' is a maximally contained rewriting of Q, if
  - Q' is a query expression in a given language L
  - Q' refers only to views in V
  - Q' $\subseteq$ Q
  - There is no other rewriting Q1, such as Q' $\subseteq$ Q1 $\subseteq$ Q

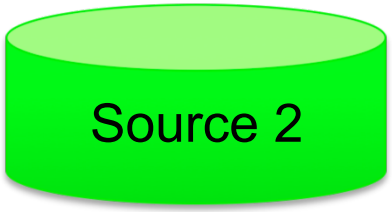
- Q' is a equivalent rewriting of Q, if
  - Q' refers only to views in V
  - Q' is equivalent to Q

Example





DBProfs



UVSQProfs

Create View UVSQProfs As
Select Course.title, Profesor.name
From Course, Teaches, Professor
Where Course.C = Teaches.C and
Teaches.P = Professor.P and
Course.university = “UVSQ”

Example: Query Containment

- Consider the query Q:

  "Select name from Professor"
- Let Q' be the query:

  "Select name from DBProfs union UVSQProfs"
- Q' is maximally-contained in Q
  - Q' ⊆ Q but Not(Q ⊆ Q')
  - Let D be a instance of the global schema consisting (also) of "Professors of Algorithms from Paris XI"…

Example: Query Containment (2)

- Consider the query Q:

```
Select  Profesor.name
From    Course, Teaches, Professor
Where   Course.C = Teaches.C and
        Teaches.P = Professor.P and
        Course.university = "UVSQ"  and
        Course.title = "Databases"
```

- Let Q' be the query:

```
Select name
from DBProfs, UVSQProfs
where DBProfs.name= UVSQProfs.name
```

- Q' is a equivalent to Q (…if the sources are indeed complete)

The variations of the problem

	Data integration	Query optimization	Database design
Equivalent rewriting	Only if the sources are complete	✓	✓
Maximally-contained rewriting	✓		
Views are materialized	✓	✓	✓
The db schema is virtual	✓		✓

Usual Settings & Problems

Data Integration:

Problem: Compute the maximally contained rewriting

Approach: Logical Rewriting (compute a query expression)

- Rewriting algorithms
- Query answering algorithms

Query Optimization & Physical data independence:

Problem: Compute the equivalent rewriting

Approach: Cost Based Rewriting (compute an evaluation plan)

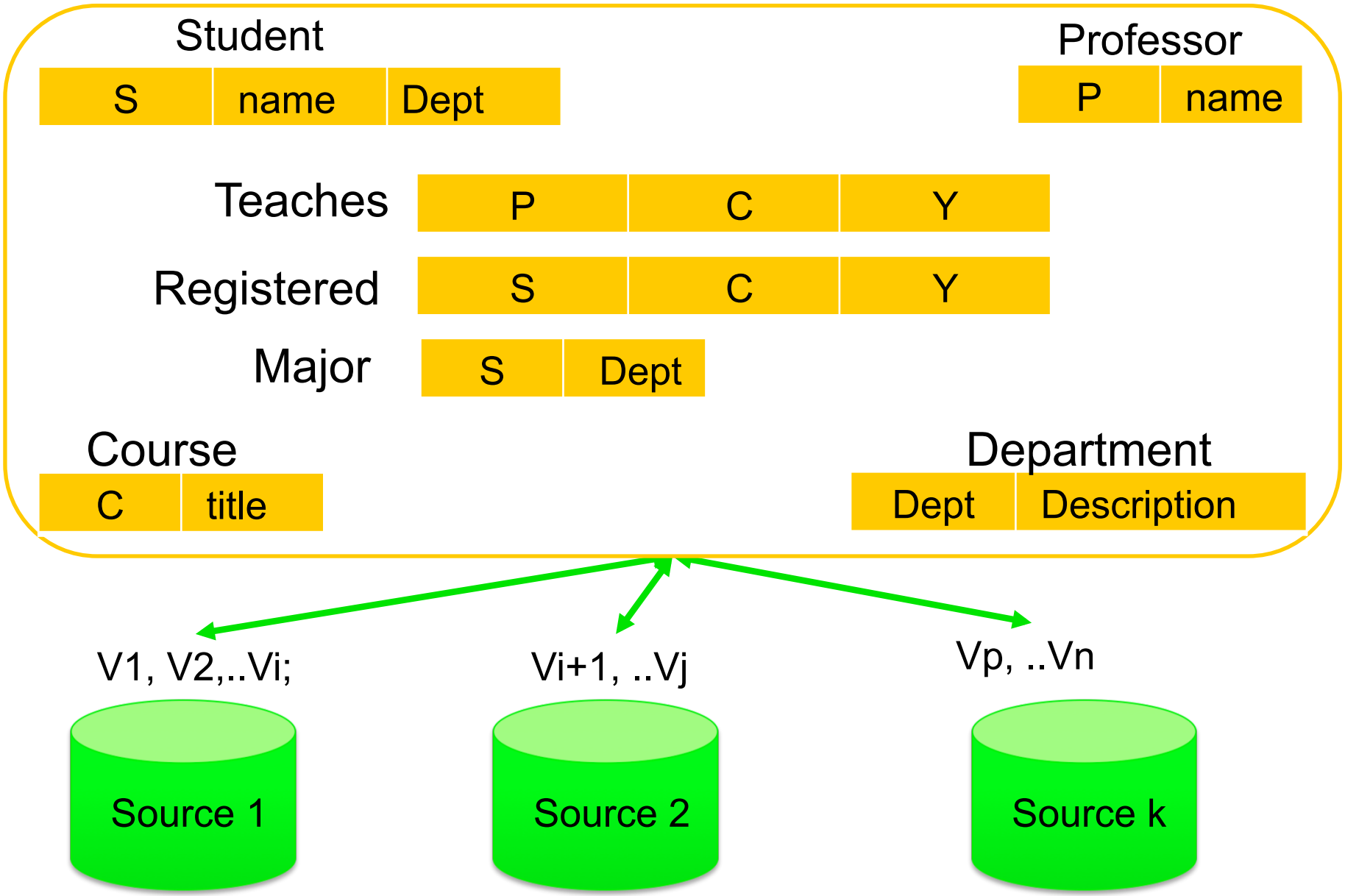
- System-R Style
- Transformational approaches

Answering Queries Using Views

- Motivating Examples
- Problem Definition
- Two different settings:
  - Data Integration
  - Query Optimization & Physical Data Independence

Answering Queries Using Views In Data Integration

Example: A Mediated Schema



Conjunctive Queries

A conjunctive query has the form:

$$q(X,\ldots) :- r_1(Y,\ldots),\ldots\ r_n(Z,\ldots)$$

where:

- q, r_1, r_n are predicate names (database relations names or arithmetic comparison e.g. ≤, =, ≥)
- X, Y, Z are variables or constants
- q(X,…) is the head of the query
- $r_1(Y,\ldots)$, $r_n(Z,\ldots)$ are subgoals
- the variables in the head should appear also in the body

✓ Conjunctive queries can express select-project-join queries

✓ Union is expressed as sets of conjunctive queries with the same head predicate

Example

A Project-Select-Join SQL query as a conjunctive query:

```
Select    Advises.P, Advises.S
From      Registered, Teaches, Advises
Where     Registered.C = Teaches.C and
          Registered.year=Teaches.year  and
          Advises.P = Teaches.P and
          Advises.S = Registered.S
          Registered.year > “98”

q(P, S)      :-    Registered(S, C, year),
                   Teaches(P, C, year),
                   Advises(P, S),
                   Year > “98”
```

The Problem

Answering queries using views expressed as conjunctive queries

Given:

- A set of Database Relations (from the Mediated Schema)
- A set of views V1,…Vn declared as
  conjunctive queries over the Database Relations
- A conjunctive query q over the Database Relations

Compute a new query expression that is

- A disjunction of conjunctive queries over the view relations
- Is maximally contained rewriting of q

Algorithms

- Algorithms for conjunctive queries
  - The Bucket Algorithm
  - The Inverse-Rules Algorithm
  - The MiniCon algorithm

Example

- The query:

  q(S, C, P) :- Teaches(P, C, year), Registered(S, C, year), Course(C, T), C ≥ 300, year≥1995

- The mediated schema (Database Relations):

  Registered(S, C, Y) Course(C, title) Teaches(P,C,Y)

- The views exported by the data sources:
  - V1(S, C, Y, title) :- Registered(S, C, Y), Course(C, title), C ≥ 500, Y≥2000
  - V2(S, P, C, Y) :- Registered(S, C, Y), Teaches(P, C, Y)
  - V3 (S, C) :- Registered(S, C, Y), Year<=1990
  - V4(P, C, title, Y) :- Registered(S, C, Y), Course(C, title), Teaches(P, C, Y), Y≤1998

Complexity

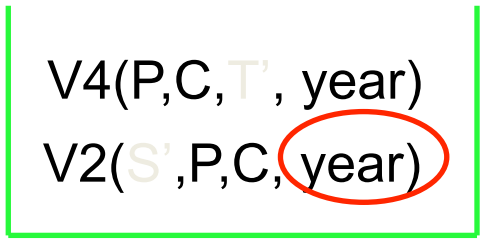
- The number of possible rewritings is exponential in the size of the query

- Bucket Algorithm:
  - It is sufficient to consider for each subgoal the views that might be relevant

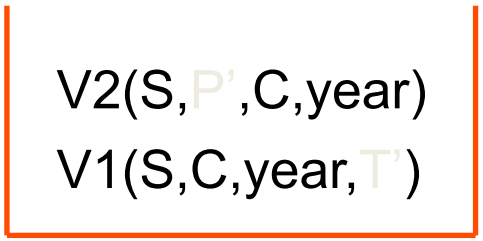
The Bucket Algorithm

- V1(S, C, Y, title) :- Registered(S, C, Y), Course(C, title), C ≥ 500, Y≥2000
- V2(S, P, C, Y) :- Registered(S, C, Y), Teaches(P, C, Y)
- V3 (S, C) :- Registered(S, C, Y), Y≤1990
- V4(P, C, title, Y) :- Registered(S, C, Y), Course(C, title), Teaches(P, C, Y), Y≤1998

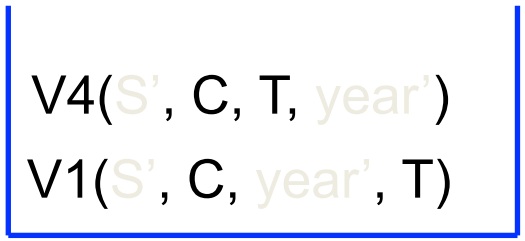
q(S, C, P) :- Teaches(P, C, year), Registered(S, C, year), Course(C, T), C ≥ 300, year≥1995



Teaches



Registered



Course

The Bucket Algorithm

I. For each subgoal of q, which is not a comparison predicate, create a 'bucket'
II. Fill the bucket, with the unifiers of relevant views
III. A solution must include a view from each bucket

Consider a query subgoal p(X1,..Xi,..Xn).

A subgoal p(Y1,..Yi,..Yn) of V is relevant to p(X1,..Xi,..Xn). :

- There is function θ that unifies the variables & constants
  θ(p(X1, X2,..)) = θ (p(Y1, Y2,..))
- The comparison predicates in V and q are compatible
- If Xi appears in the head of q then Yi must be in the head of V

The Bucket Algorithm

- There are 8 combinations that have to be checked
- For each combination:
  - Make equal appropriate variables to enforce joins
  - If conflicting conditions, abandon
  - Eliminate duplicated goals

Teaches	Registered	Course
V4(P,C,T', year)	V2(S,P',C,year)	V4(S', C, T, year')
V2(S',P,C, year)	V1(S,C,year,T')	V1(S', C, year', T)

q'(S,C,P) :- V2(S,P,C,year), V1(S,C,year,T'), V1(S',C,year,T)

q'(S,C,P) :- V2(S',P,C, year), V1(S, C, year, T') The first solution!

Algorithms

- Algorithms for conjunctive queries
  - The Bucket Algorithm
  - The Inverse-Rules Algorithm
  - The MiniCon Algorithm

The Inverse-Rules

Idea: Express the predicates of the mediated schema in terms of views: Invert the view definitions

View Example: V3(D, C) :- Major(S, D), Registered(S, C)

Intuition: A tuple V3(D, C) in the view V3 witnesses the existence of a value $Z=f_S(D,C)$ such that Major(Z, D) and, Registered(Z, C)

Skolem functions:

- f_S is called Skolem function
- f_S was introduced for the existential variable S of V3

Create Inverse Rules:

- Major($f_S(D,C)$, D) :- V3(D, C)
- Registered($f_S(D,C)$, C) :- V3(D, C)

Generate the Inverse-Rules

Algorithm:

For each view V:

I. For each existential variable in V, create a Skolem function

II. For each subgoal of a view create a rule:

- The head is the subgoal
  - ! Use the Skolem functions for existential variables in V
- The body is the head of the view

The Inverse Rule Algorithm

Algorithm: The rewriting of the query Q using a set of views V1, V2, .. Vn is the Datalog program consisting of

- The inverse rules for V1, V2, .. Vn
- The query Q

Observation: Datalog is a language similar to Prolog

Inverse Rule Algorithm (2)

Example:

Consider the query: q(D) :- Major(S,D), Registered(S, 444)

and the view: V3(D, C) :- Major(S, D), Registered(S, C)

Inputs:

The rules:

- Major(f_S(D,C), D) :- V3(D, C)
- Registered(f_S(D,C), C) :- V3(D, C)
- q(D) :- Major(S,D), Registered(S, 444)

Suppose that V3 has the following tuples:

D	C
CS	444
EE	444
CS	333

Inverse Rule Algorithm (3)$_{V3}$

Execute

I. The inverse rules produce the tuples:

Major

$f_S(D,C)$	C
$f_S(CS,444)$	444
$f_S(EE,444)$	444
$f_S(CS,333)$	333

Registered

D	$f_S(D,C)$
CS	$f_S(CS,444)$
EE	$f_S(EE,444)$
CS	$f_S(EE,444)$

D	C
CS	444
EE	444
CS	333

II. Apply the query q to the extensions Major & Registered

Bucket versus Inverse-rules

- Both algorithms produce the maximal contained rewritings
- The same intuition:

  Compute the views that are relevant to DB relations
- The inverse-rules are computed once and used anytime
- The bucket considers only the views relevant to the query

The MiniCon Algorithm

An improvement of Bucket

- Excludes much more views from participating in buckets
- There are fewer combination to check at the end

New constraint:

- If a view V contains a ‘target’ subgoal of Q and
- The subgoal participates in a join in Q

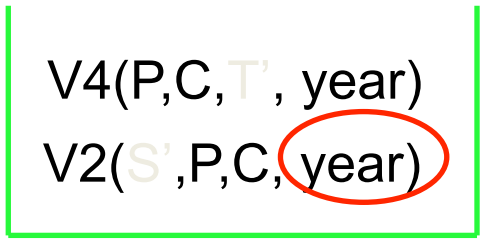
then V is relevant to Q if

- Either the ‘join’ variables appear in the head of V
- Or the join takes place also within V

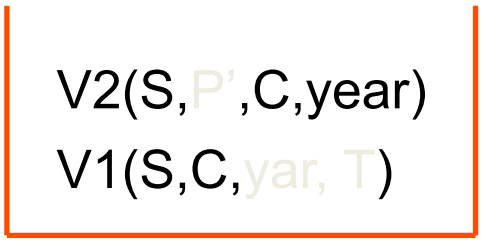
Bucket vs Minico Algorithm

- V1(S, C, title) :- Registered(S, C, Y), Course(C, title), C ≥ 500, Y≥2000
- V2(S, P, C) :- Registered(S, C, Y), Teaches(P, C,Y)
- V3 (S, C) :- Registered(S, C, Y), Y≤1990
- V4(P, C, title, Y) :- Registered(S, C, Y), Course(C, title), Teaches(P, C, Y), Y≤1998

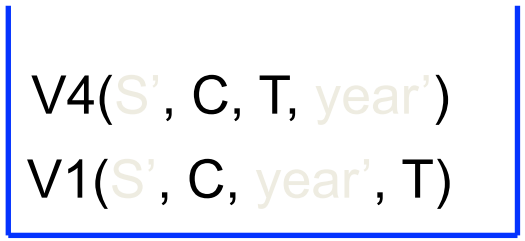
q(S, C, P) :- Teaches(P, C, year), Registered(S, C, year), Course(C, T), C ≥ 300, year≥1995



Teaches



Registered



Course

Further Reading

- Bucket Algorithm [Levy & Rajaraman & Ullman 1996]
- Inverse Rules Algorithm [Duscka & Genesereth 1997]
- MiniCom Algorithm [Pottinger & Halevy 2001]
- MDCSAT [Arvelo & Bonet & Vidal 2006]
- SSSAT [Izquierdo & Vidal & Bonet 2011]
- GQR [Konstantinidis & Ambite, 2011]

AQUV for other Languages

Rewriting language plays a key role:

- The maximal contained expression depends of it

Works considering other rewriting languages

- Object query languages (OQL) (Florescu 96)

- Views with binding patterns (Rajaraman 95)
  - Describe sources such as Web services, Web Forms

Access-Pattern Limitations

- Often we can only access the data sources in specific ways:
  - Web forms: require certain inputs
  - Web services: interface definitions
  - Controlling load on systems: allow only limited types of queries
- Model limitations using adornments:
  - $V^{bf}(X,Y)$: first argument must be bound

Example Binding Patterns

Cites(X,Y)
DBPapers(X)
AwardPaper(X, P)

$$S1 : CitationDB^{bf}(X,Y) \subseteq Cites(X,Y)$$

$$S2 : CitingPapers^{f}(X) \subseteq Cites(X,Y)$$

$$S3 : DBSource^{f}(X) \subseteq DBpapers(X)$$

$$S4 : AwardDB^{bf}(X,P) \subseteq AwardPaper(X,P)$$

Executable Plans

Conjunctive plan $q_1(\overline{X_1}),...,q_n(\overline{X_n})$

And let BF_i be the adornments for q_i

We say that the plan is *executable* if there are $bf_i \in BF_i$ *s.t.*

If a variable *X* is in position *k* of q_i, and bf_i has a '*b*' in *k*'th position, then *X* occurs earlier in the plan i.e., every variable that must be bound has a value.

Example (1)

Cites(X,Y)
DBPapers(X)
AwardPaper(X, P)

$S1: CitationDB^{bf}(X,Y) \subseteq Cites(X,Y)$

$S2: CitingPapers^{f}(X) \subseteq Cites(X,Y)$

$S3: DBSource^{f}(X) \subseteq DBpapers(X)$

$S4: AwardDB^{bf}(X,P) \subseteq AwardPaper(X,P)$

Q(X) :- Cites(X,001)

X *Q'(X) :- CitationDB(X,001)*

Q'(X) :- CitingPapers(X),CitationDB(X,001)

Rewritings of unbound length!

Cites(X,Y)
DBPapers(X)
AwardPaper(X, P)

$$S1: CitationDB^{bf}(X,Y) \subseteq Cites(X,Y)$$

$$S3: DBSource^{f}(X) \subseteq DBpapers(X)$$

$$S4: AwardDB^{bf}(X,P) \subseteq AwardPaper(X,P)$$

Q(X) :- AwardPaper(X,"VLDB Best Paper")

Q'(X) :- DBSource(X),AwardDB(X, "VLDB Best Paper")

Q'(X) :- DBSource(Y),CitationDB(Y,X),
AwardDB(X, "VLDB Best Paper")

Q'(X) :- DBSource(Y),CitationDB(Y,X_1),…,
CitationDB(X_n,X), AwardDB(X, "VLDB Best Paper")

Recursive Plans to the Rescue

$$papers(X) :- DBSource(X)$$

$$papers(X) :- papers(Y), CitationDB(Y, X)$$

$$Q'(X) :- papers(X), AwardDB(X, "VLDBBestPaper")$$

Theorem: this can be done in general

[Duschka, Genesereth & Levy, 1997]

Recursive Query Plans
[Duschka & al 2000]

- For every view with binding pattern

  k times

  $V^{b,\ \ldots\ b,f,..f}(x_1,\ldots x_k..x_n)$:- $r_1(\ldots),\ldots\ r_m(\ldots)$

  – Add an inverse rule for each of the terms $r_1,..\ r_m$

    $r_i(\ldots)$:- $dom(x_1),dom(x_2),..dom(x_k),V^{b,\ \ldots\ b,\ f,\ ..f}(x_1,..x_k..x_n)$

  – Add a dom rule for every free variable $x_{k+1},\ldots x_n$

    $dom(x_j)$:- $dom(x_1),\ dom(x_2),..dom(x_k),V^{b,\ \ldots\ b,\ f,\ldots f}(x_1,..x_k..x_n)$

- For every constant c in the query add a rule

  dom(c) :-

The query is evaluated on the resulted Datalog program

A Graph Perspective

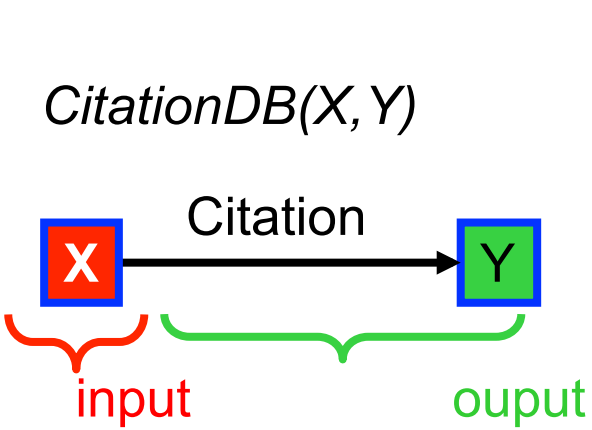
- If the mediated schema consists of binary relations (as in RDF), we can represent queries and plans as graphs

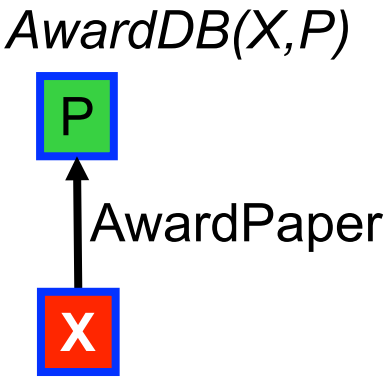
Cites(X,Y)
DBPapers(X,T)
AwardPaper(X, P)

$$S1 : CitationDB^{bf}(X,Y) \subseteq Cites(X,Y)$$

$$S2 : DBSource^{f}(X) \subseteq DBpapers(X,"Databases")$$

$$S4 : AwardDB^{bf}(X,P) \subseteq AwardPaper(X,P)$$

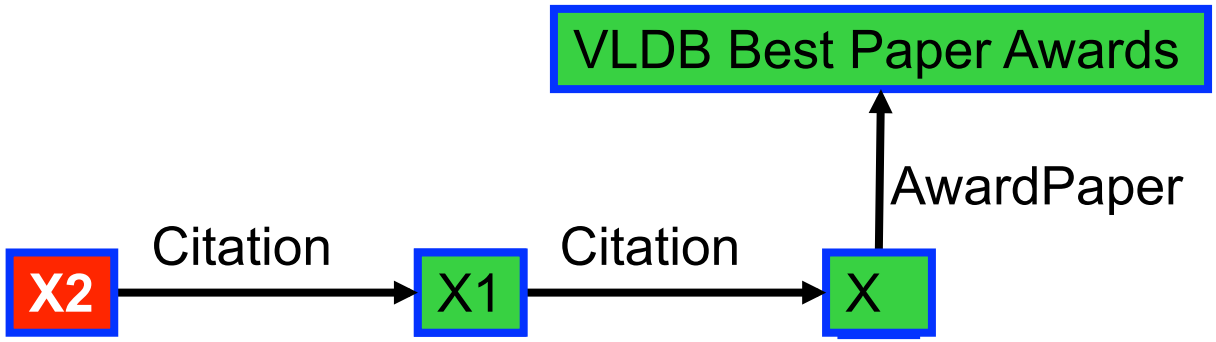




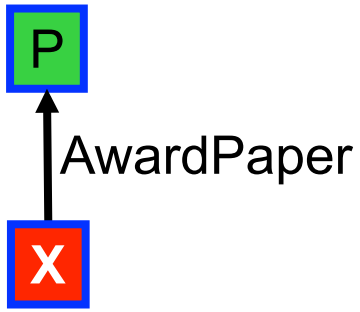
Query Evaluation Example

- A query is a graph
- A query plan is also a graph

Q(X) :- AwardPaper(X,"VLDB Best Paper")



AwardDB(X,P)



DBPapers(X)

CitationDB(X,Y)

Question

- Are the all the enumerated rewritings relevant (can they return new results) ?

- Answer: Unfortunately, yes as the sources are incomplete.

Query Answering Problem

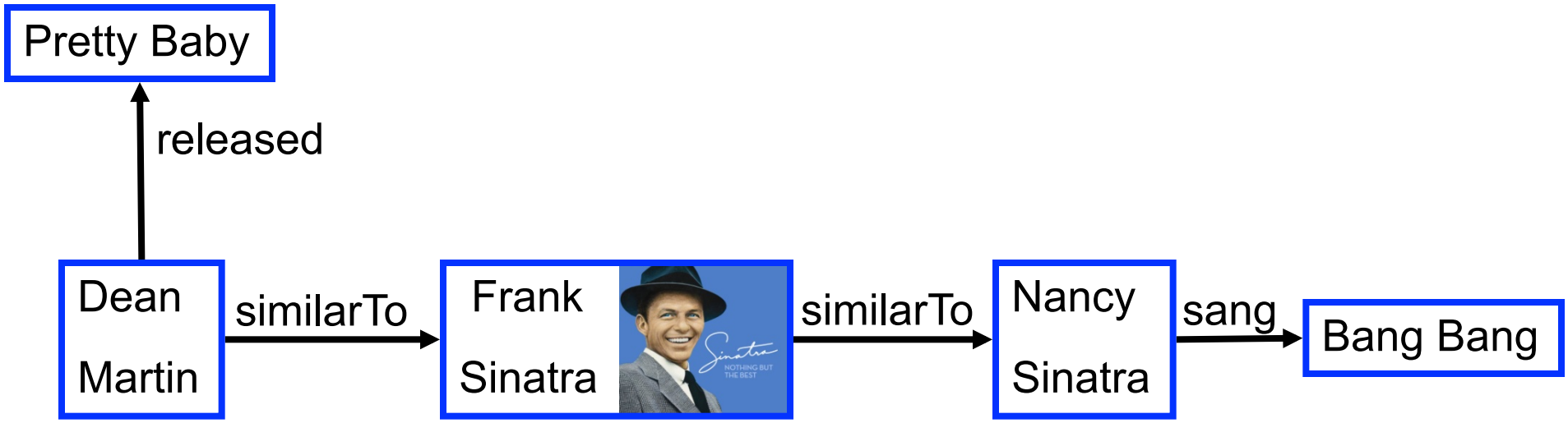
Given a query Q against

- a RDF base G
- a set of functions F
- A limit Max for the number of Web calls

compute the larger number of answers using Max calls

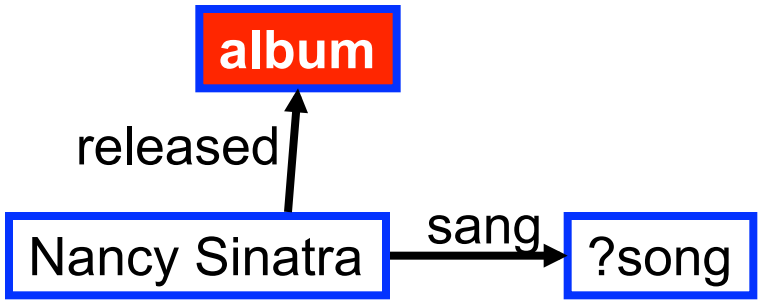
→ We need to prioritize the calls

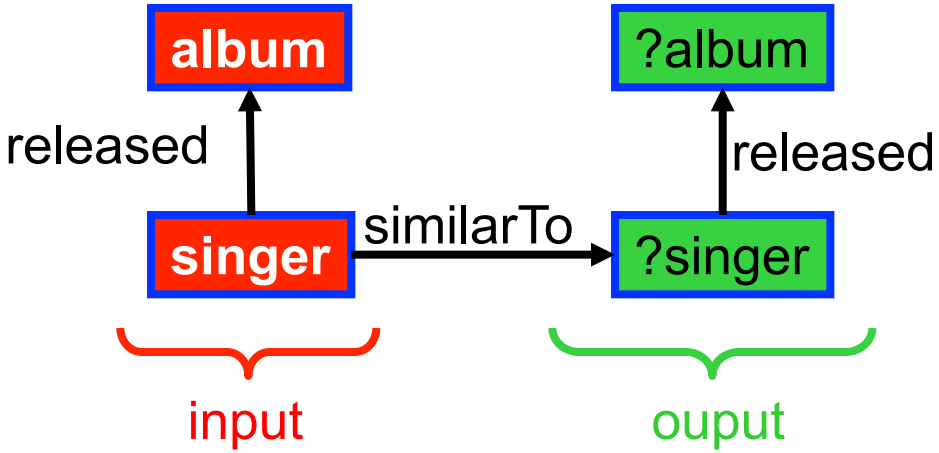
RDF Fragment



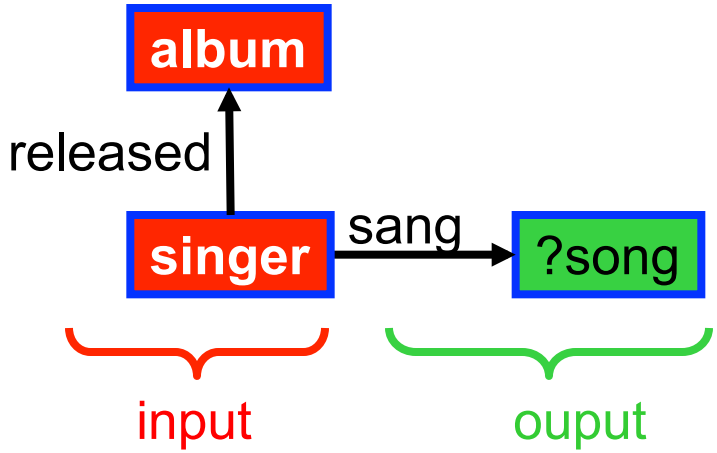
The RDF fragment itself can be described as LAV views!

Query example



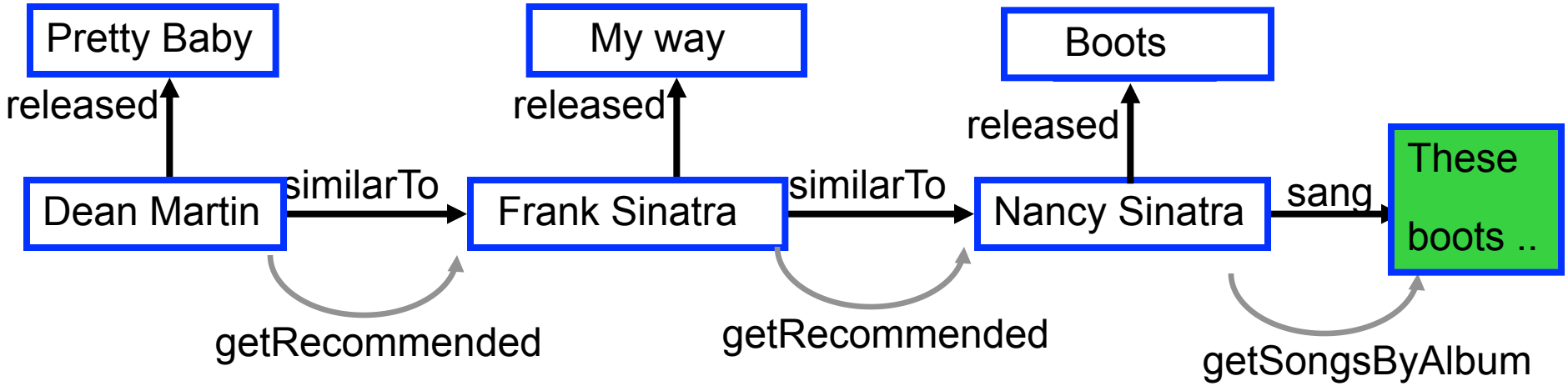


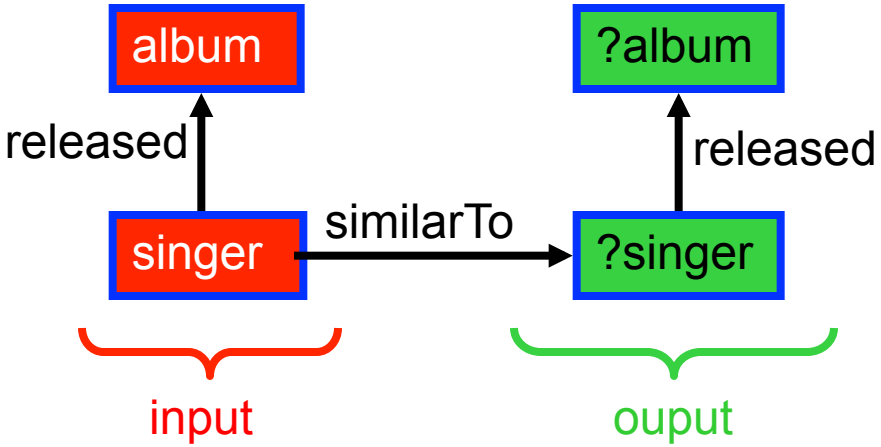
get Recommended

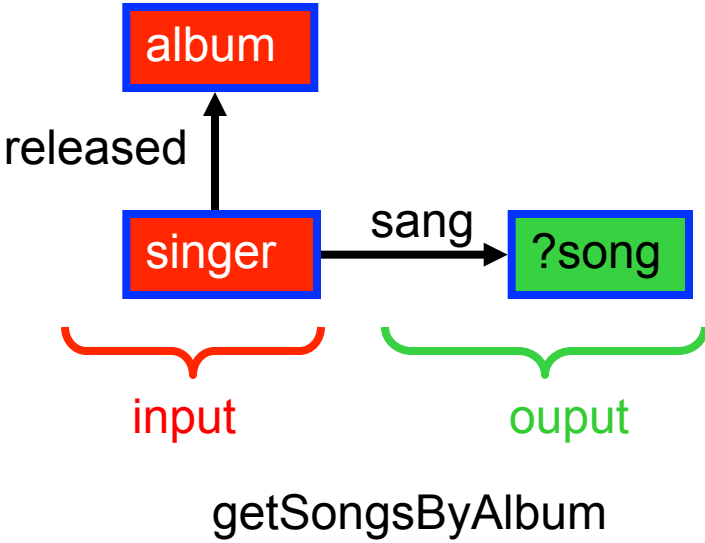


getSongsByAlbum

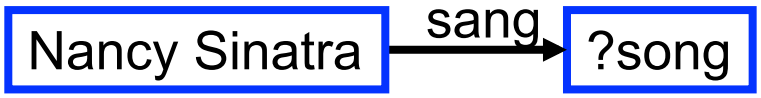
ANGIE Algorithm



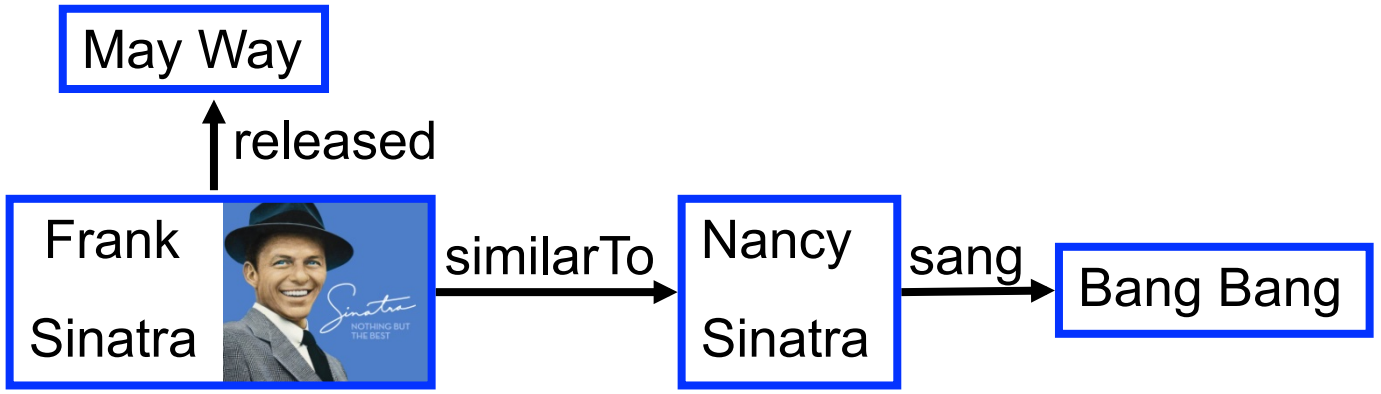


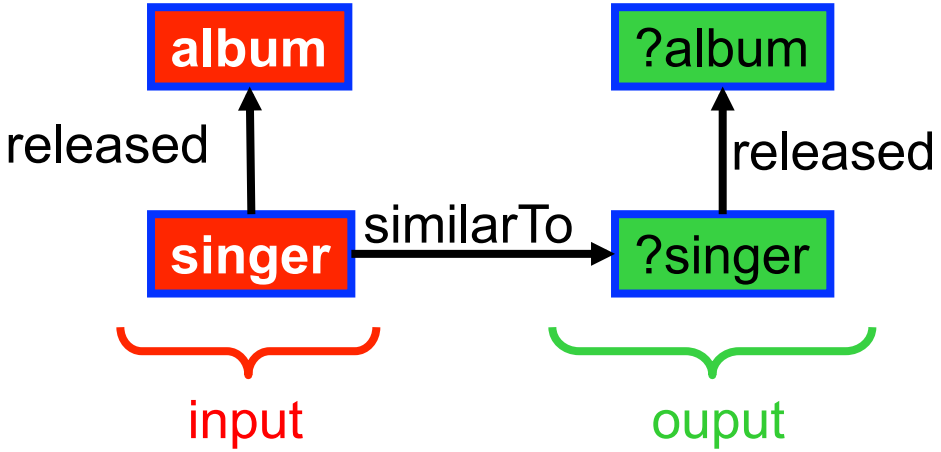


Exercise:

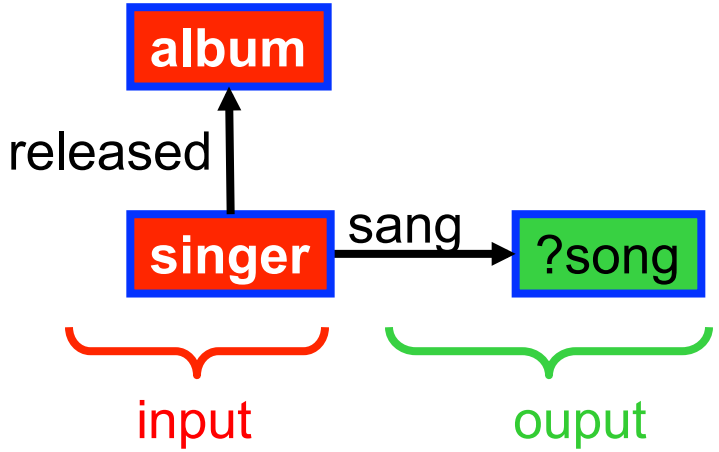


- Input:



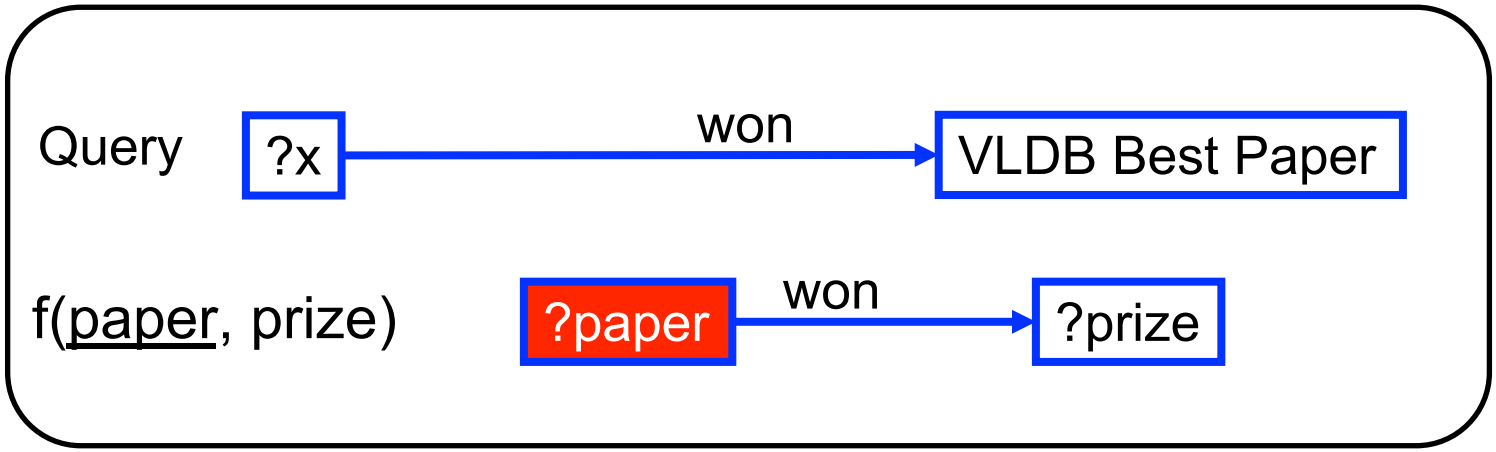


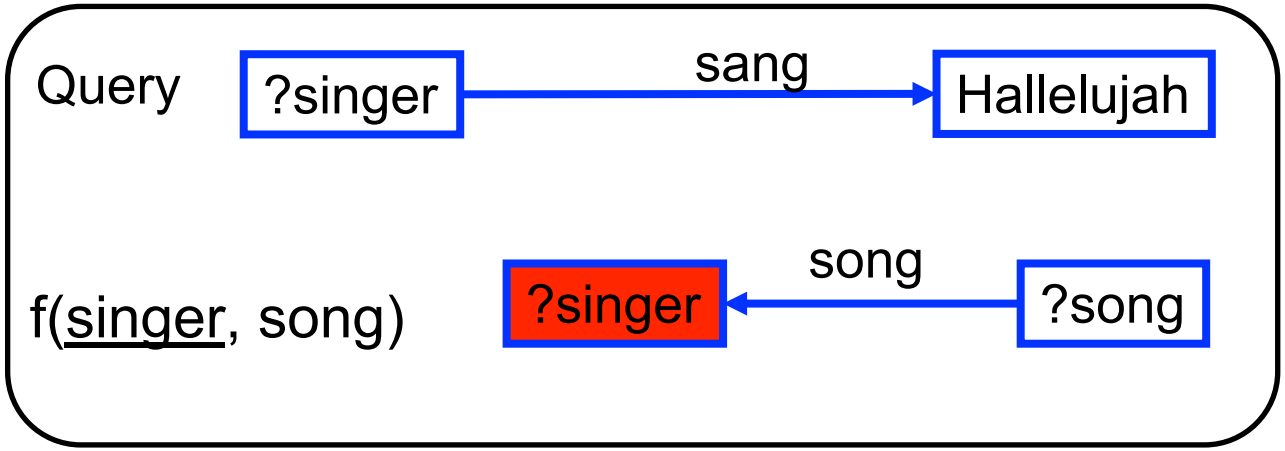
get Recommended



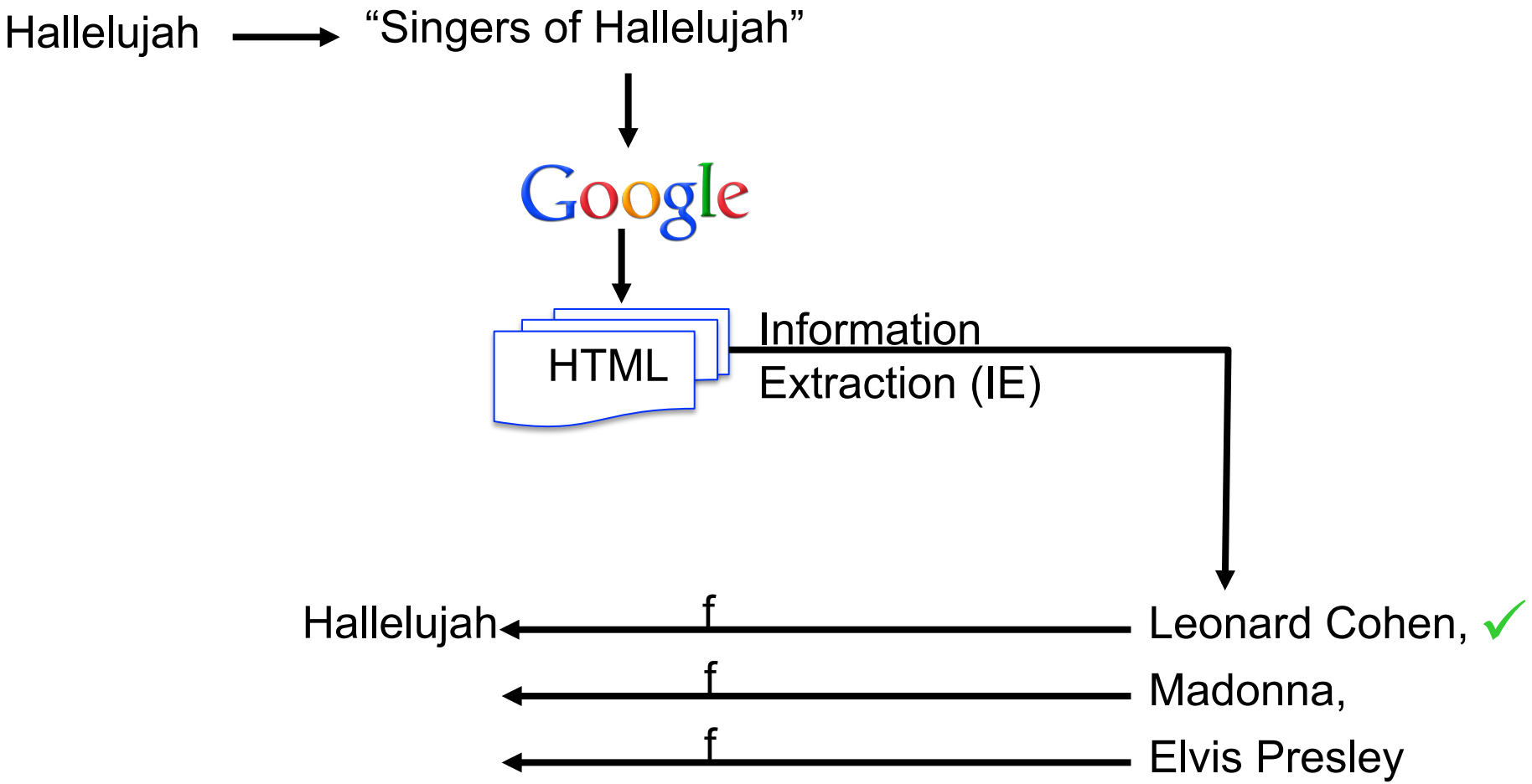
getSongsByAlbum

Asymmetric Accesses

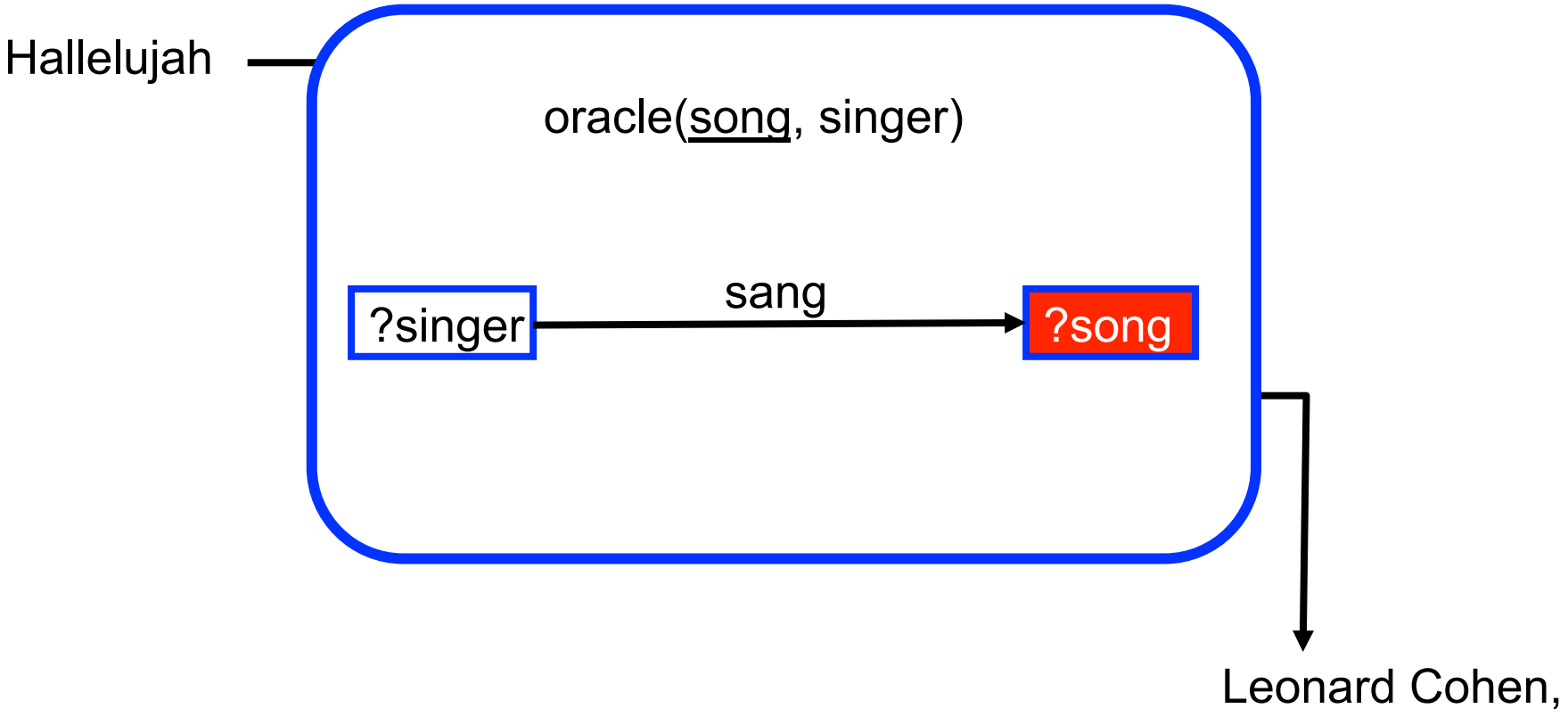




Idea: Use the Web as an Oracle



Model Oracles as Web services



AQUV for other Languages

Rewriting language plays a key role:

- The maximal contained expression depends of it

Works considering other rewriting languages

- Object query languages (OQL) (Florescu 96)
- Views with binding patters (Rajaraman 95)
  - Describe sources such as Web services, Web Forms
- Conjunctive Queries with integrity constraints

Integrity Constraints with LAV

Schedule(airline, flightNum, date, pilot, aircraft)

Functional dependencies:

Pilot → Airline and Aircraft → Airline

S(date,pilot,aircraft) ⊆ schedule(a,fN,date,pilot,aircraft)

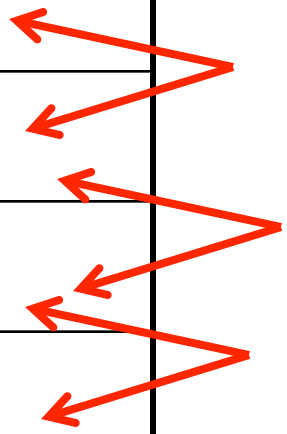
Query: (pilots in same airline as Mike)

q(p) :- schedule(a,fN,date,"mike",aircraft)
schedule(a,f,d,p,a)

Without functional dependencies, no answers!

But We Do Have Answers…

Date	Pilot	Aircraft
1/1	Mike	111
5/2	Ann	111
1/3	Ann	222
4/3	John	222



Mike and Ann work for the same airline

111 and 222 belong to the same airline

John and Ann work for the same airline, …

No Limit to Rewriting Size

$$q_n'(p) :- S(D_1, \text{"}mike\text{"}, C_1), S(D_2, p_2, C_1),$$
$$S(D_3, p_2, C_2), S(D_4, p_3, C_2), \ldots,$$
$$S(D_{2n-2}, p_n, C_{n-1}), S(D_{2n-1}, p_n, C_n)$$
$$S(D_{2n}, p, C_n)$$

Does this sound familiar?

Recursive plans to the rescue!

Recursive Plan

- Start with inverse rules and get:

$$schedule(f_1(1/1, Mike, \#111), f_2(1/1, Mike, \#111), 1/1, Mike, \#111)$$
$$schedule(f_1(5/2, Ann, \#111), f_2(5/2, Ann, \#111), 5/2, Ann, \#111)$$
$$schedule(f_1(1/3, Ann, \#222), f_2(1/3, Ann, \#222), 1/3, Ann, \#222)$$
$$schedule(f_1(4/3, John, \#222), f_2(4/3, John, \#222), 4/3, John, \#222)$$

- But we know that:

$$f_1(1/1, Mike, \#111) = f_1(5/2, Ann, \#111)$$
$$f_1(5/2, Ann, \#111) = f_1(1/3, Ann, \#222)$$
$$f_1(1/3, Ann, \#222) = f_1(4/3, John, \#222)$$

The e relation

- Let's define a relation *e(X,Y)* that stores all pairs that are actually equal.
- *e(x,x)* – holds for every *x*.
- The other rules for inferring *e(X,Y)* are derived from the functional dependencies.
- Add: *e(X,Y)* :- *e(X,Z)*, *e(Z,Y)*

The Rules

Aircraft → Airline: So…

$$e(X,Y) :- schedule(X,F,P,D,A),$$
$$schedule(Y,F',P',D',A'),$$
$$e(A,A')$$

Pilot → Airline: So…

$$e(X,Y) :- schedule(X,F,P,D,A),$$
$$schedule(Y,F',P',D',A'),$$
$$e(P,P')$$

One More Step

- Reformulate the query to use *e(X,Y):*

$$q(P) :- schedule(AL, F', D', Mike, A),$$
$$schedule(AL, F, D, P, A')$$

$$q(P) :- schedule(AL, F', D', P', A),$$
$$schedule(AL', F, D, P, A'),$$
$$e(AL, AL'), e(P', Mike)$$

Answering Queries Using Views

- Motivating Examples
- Problem Definition
- Two different settings:
  - Data Integration
  - Query Optimization & Physical Data Independence

Query answering: Optimization Versus Data Integration

- In the query optimization context
  - A variation of the Selinger Algorithm
  - Cost-based equivalent rewritings

  } Equivalent rewritings

- In the data integration context
  - The bucket algorithm
  - The inverse-rules algorithm
  - The MiniCon algorithm

  } Maximally-contained rewritings

Materialized Views

Suppose that we have the following views:

- create view *JoinStudent&Registered* as

  select Name, C

  from Student, Registered

  where Student.S = Registered.S

- create view *JoinProfessor&Teaches as*

  select Name, P

  from Professor, Teaches

  where Professor.P = Teaches.P

Observation:

- The views are evaluated and the result is materialized (stored)
- The materialized result can be used in the evaluation of queries

Optimization Goal

Reduce the computation by using pre-computed joins stored as materialized views

Query Optimization Example

- Consider the query:

```
Select    Student.Name, Professor. Name
From      Student, Registered, Teaches, Professor
Where     Student.S=Registered.C and Teaches.P=Professor.P
          and Registered.C=Teaches.C
```

- A possible rewriting to reduces the query execution time:

```
Select    Student.Name, Professor.Name
From      JoinStudent&Registered, JoinProfessor&Teaches
Where     JoinStudent&Registered.C= JoinProfessor&Teaches.C
```

2 Joins eliminated by using materialized views instead

Query Optimization Problem

Given

- A query Q
- A set of views V1, V2, …Vn
- A cost model C

Compute the equivalent query plan Q' (Q'=Q)

that has the cheapest cost

Difficulties

- Large number of many possible combinations

  Large search space to find the solution
- In some cases, views may not help due to loss of indexes

Relevant Views

A view V is useful in the evaluation of a query Q if:

- There is a mapping of relations from V to Q and
- V must project at least all attributes projected in Q and
- Each relation R1 that appears in both V and Q satisfies:
  the join and the selection conditions

Relevant Views: Join Conditions

Consider a relation R1 appearing in both Q and V and let R2 be the other relation in the join in Q.

R1 satisfies the join conditions if:

- In V, R1 and R2 are joined in the same attributes and conditions
- OR In V, R1 and R2 are joined in the same attributes and for weaker conditions and the conditions are projected in V
- OR In V, R1 an R2 are not joined, but the attributes participating in the join in Q are projected in V

Relevant Views: Selection Conditions

Consider a relation R1 appearing in both Q and V and let A be an attribute of R1 for which Q applies a selection predicate:

R1 satisfies the selection conditions if :

- Either V applies the same selection predicate for the attribute A
- OR V applies a weaker selection predicate and A is projected in V
- OR V applies no selection predicate for A and A is projected in V

Examples

Q: Select Student.Name, Professor.Name
From Student, Registered, Teaches, Professor
Where Student.S=Registered.S and Registered.C = Teaches.C and Teaches.P = Professor.P and Student.Year> 2000

V1:
```
Select  name
from Student
where Student.year>2000
```


V2:
```
Select name, S
from Student
where  Year> 2000
```


V3:
```
Select name, S
from Student
where Year >1996
```


V4:
```
Select name, S, Year
from Student
where Year >1996
```


V5:
```
Select Student.name, Course.name
from Student, Registered, Course
where Student.S = Registered.S
and Registered.C = Course.C
```


V6:
```
Select  Student.name,Student.year,
                    Course.name,
Course.C
from Student, Registered, Cours
where Student.S= Registered.S
and Registered.C = Course.C
```


Query Optimization – the Selinger Algorithm

- The Selinger Algorithm chooses the best join order
- A bottom-up dynamic programming algorithm
- Each iteration explores all join combinations of certain length
- For each combination of joins, it keeps only the cheapest plan, taking into account costs estimated in the previous iteration

A variation of the Selinger Algorithm

- The Selinger Algorithm can be modified in order to choose the cheapest equivalent rewriting
- Start with views that seem useful for a certain query
- Continue in a bottom-up fashion exploring all join combinations
- For each combination, prune plans estimated to be more expensive or less contributing than others

A variation of the Selinger Algorithm

- First Iteration:
  - Find all views that are relevant (useful) to the query
  - Choose the best access path for each view and evaluate its cost
- Second Iteration:
  - Find all pair combinations
  - For each pair find all possible Joins in all possible attributes that lead to useful plans
  - Plans that are complete (i.e. are sufficient to answer the query) are distinguished as final candidates
  - For the rest plans do the following:
    - Compare pairs of plans
    - For each pair (p, p'), if p' is cheaper and has greater or equal contribution (covers more relations), then p is pruned
- Third Iteration:
  - Find all possible pairs of plans derived from the previous iteration and the plans derived from the first iteration
  - …

A variation of the Selinger Algorithm

Query: R1⋈R2 ⋈R3 ⋈R4

Views: V1=R1⋈R2 , V2=R2 ⋈R3 ⋈ R4

V3=R2 ⋈ R3, V4=R3 ⋈ R4,