A Declarative Approach to Information Extraction using Web Services API

Slides before 1st Section Divider

Studied issue

Locks and keys

Declarative data extraction

Data integration with WS

Main scenarios

Results and discussion

Additional slides

John SAMUEL

Christophe REY

Tutorial ICWE 2016

June, 7th

A Declarative Approach to Information Extraction using Web Services API

ICWE 2016 Tutorial
June, 7th

John SAMUEL
Université de Lyon - FRANCE
Christophe REY
LIMOS, Université Clermont Auvergne - FRANCE

Outline

- Studied issue → Integrating with WS
- Locks and Keys → SME, few developers, numerous WS
  → be declarative !
- Declarative data extraction → A proposal
- Scientific basis → LAV mediation with access patterns
- Main scenarios
- Results and discussion

[Studied issue]

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

Small and Medium Scale Enterprises dependent on Numerous Web Services

Small and Medium Scale Enterprises (SME)

- Less human resources
- Dependent on numerous web services for their day to day activities
- Data spread across different data centers
- No direct control over their own data
- Need to access historical data
- Easier mechanism to compute performance indicators

Web Services (WS)

- Services accessible on internet
- Primary audience: users with web browsers
- Cater to needs of numerous clients
- Manage user resources
- Provide ability to access and manipulate user resources
- Heterogeneous and autonomous
- For machine to machine access, service providers provide API
  (Application Programming Interface)

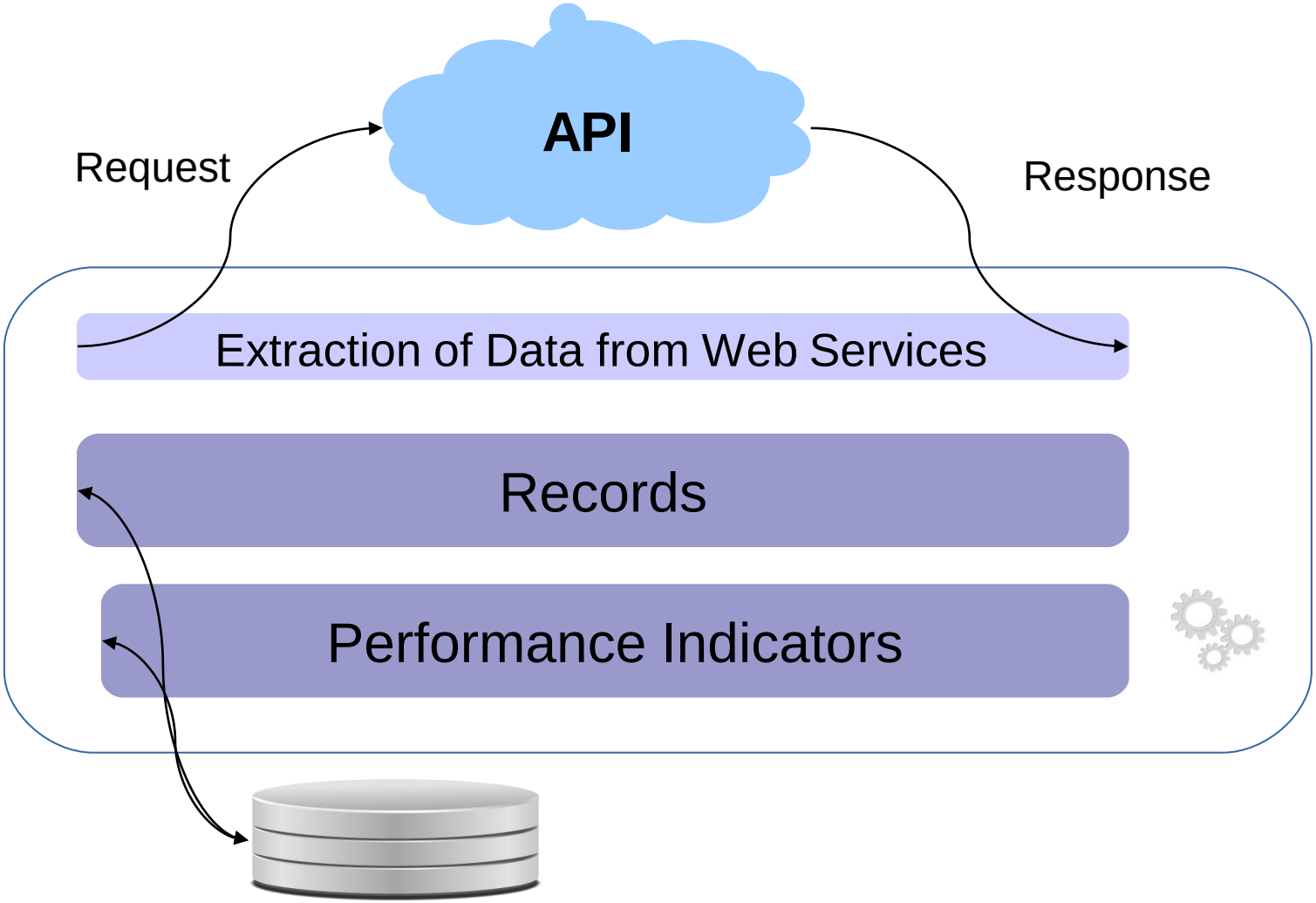
Integration with WS

- Problem:
  - Integrate with **many** web services using their APIs
  - More precisely:
    "integration" = "data extraction"
- Constraint
  - **Very few** developers
- Initially proposed by Rootsystem, Vichy, France

Integration with WS

- Goal 1
  Extract and transform data from WS using APIs
  Historical data
- Goal 2
  Compute performance indicators using historical data

Integration with WS



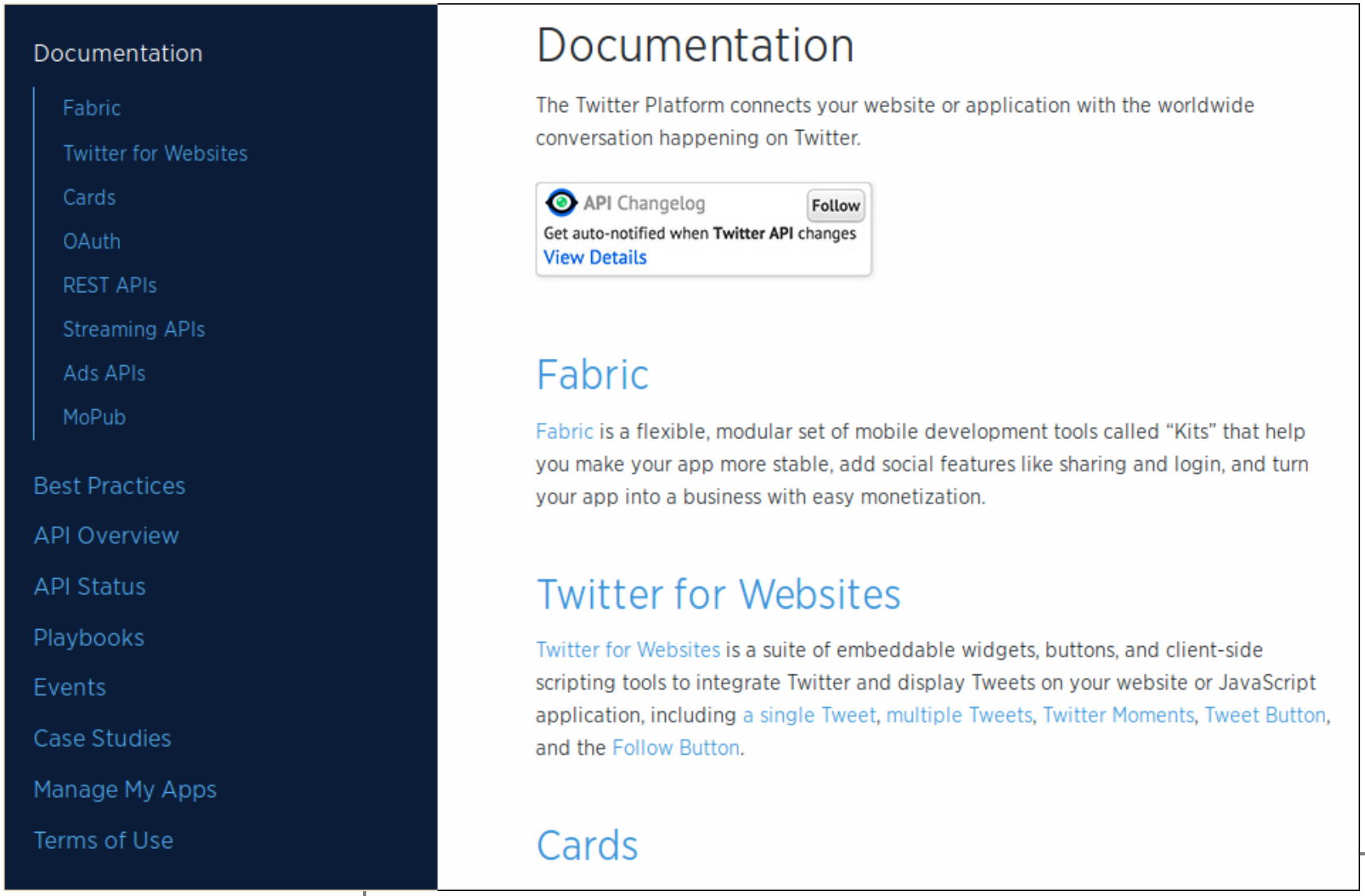
Real Life Example

- Social networking service: Twitter API
  - Resources: user, status, followers, lists,
  - Analytics: favorites, retweets
- REST API
  - Operations: Read, Write, Update and Delete
  - Simple and widely used
- Message formats
- Authentication mechanism

Performance Indicators

- Performance Indicators like profit, sales etc.
- Historical data for comparing the evolution of enterprise performance
- Use of various web services like social networking sites. E.g. Twitter
- Example: Number of followers on a social media site

Twitter Documentation



Twitter REST API

REST APIs

The REST APIs provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are available in JSON.

If your intention is to monitor or process Tweets in real-time, consider using the Streaming API instead.

Overview

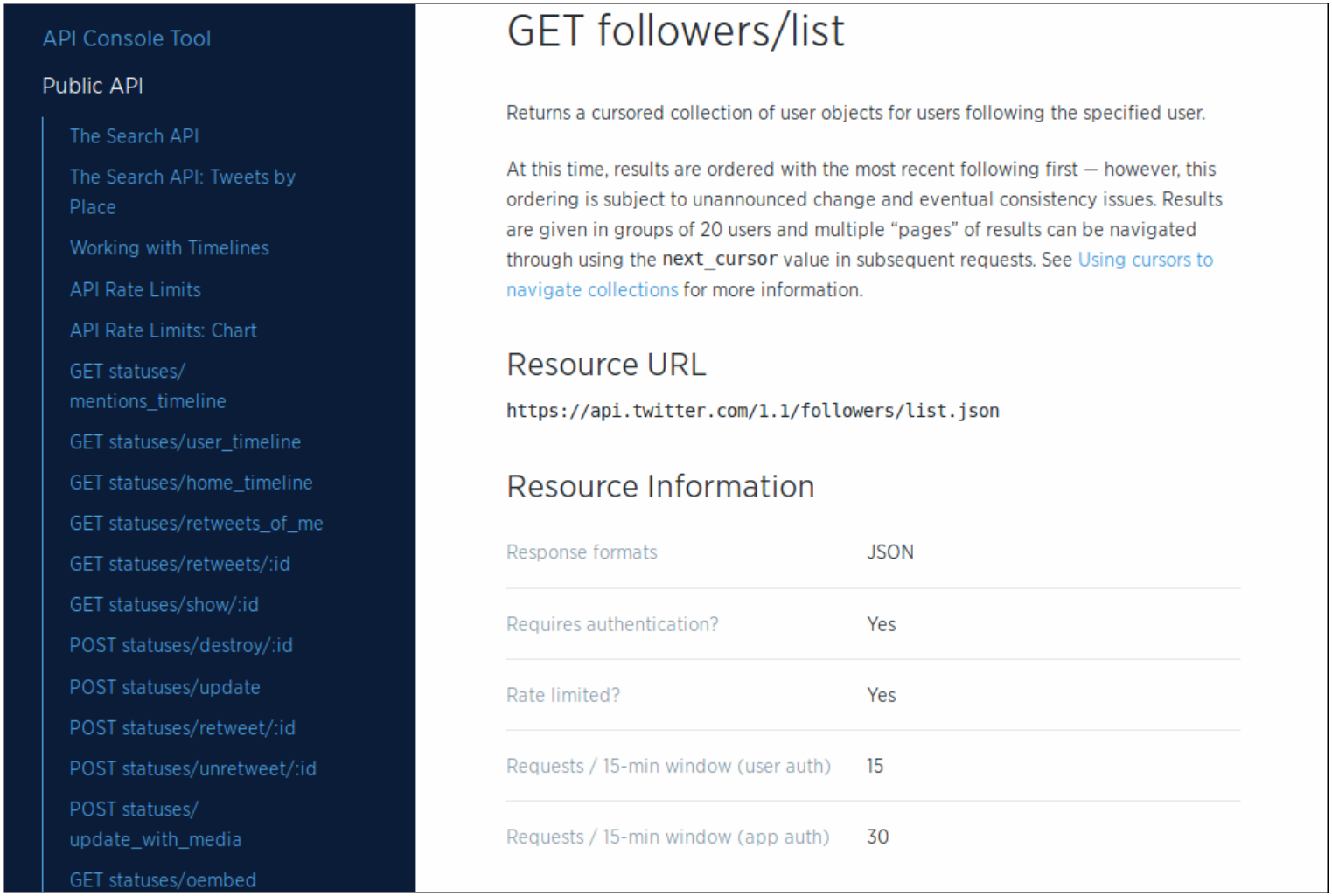
Below are the documents that will help you get going with the REST APIs as quickly as possible

- API Rate Limiting
- API Rate Limits
- Working with Timelines
- Using the Twitter Search API
- Uploading Media
- Multiple Media Entities in Statuses
- Finding Tweets about Places

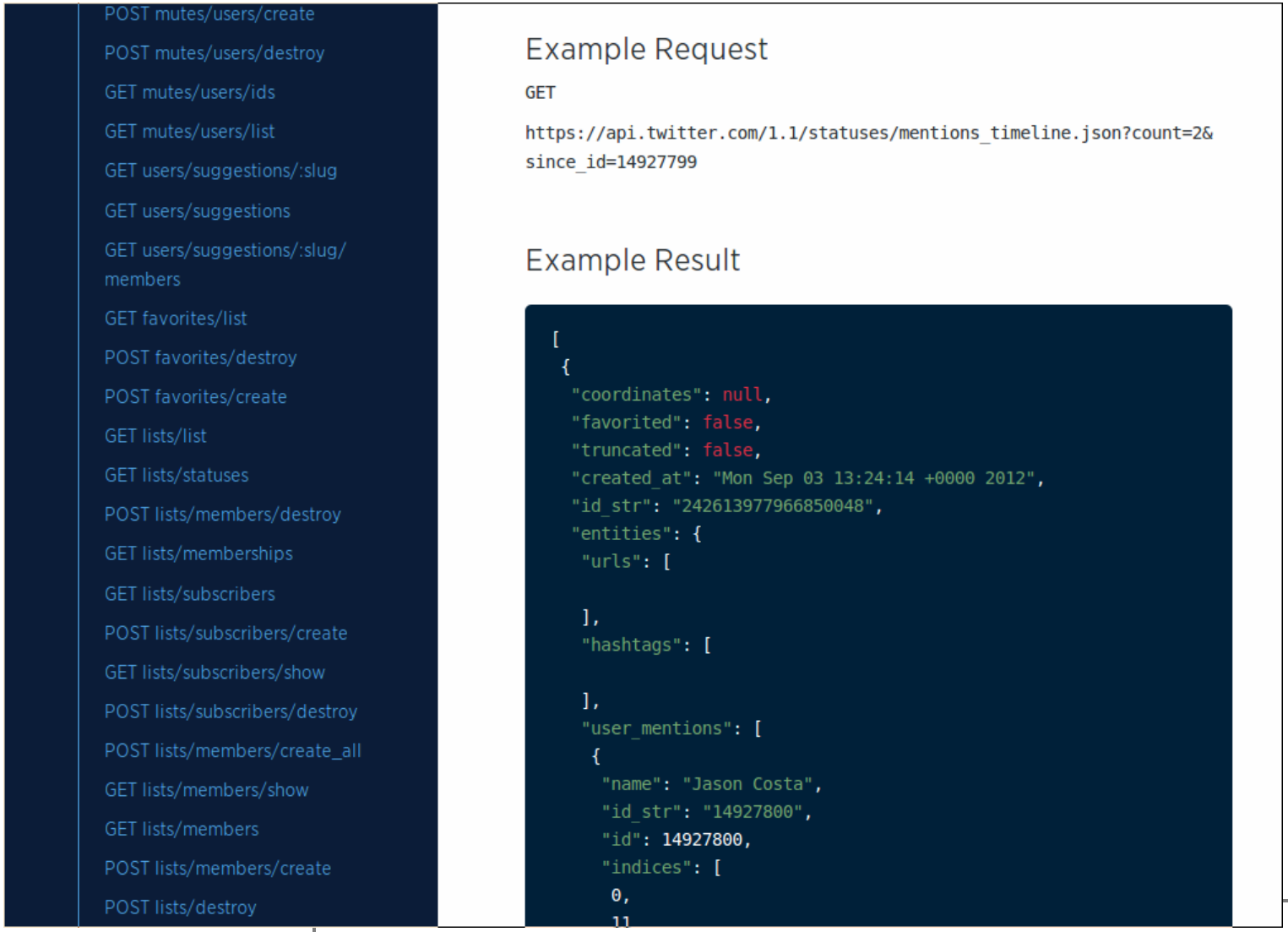
Latest Updates

As of version 1.1 of the Twitter API, the more recent updates to our API are highlighted below. We're excited about what it means for developers and we've captured all the meaningful changes here so you don't miss a thing.

Twitter Message Format: JSON



Twitter Status



Twitter Authentication: OAuth

- Overview
- Application-Only Authentication
- 3-Legged OAuth
- PIN-Based OAuth
- xAuth
- OAuth Echo

OAuth

Send secure authorized requests to the Twitter API

Twitter uses OAuth to provide authorized access to its API.

Running (toy) example

- Two social networking services:
  - SN_1
  - SN_2
- SN_1 uses XML data format
- SN_2 uses JSON data format
- Both provide API operations to access
  - user status
  - view counts

Example: SN_1 operations

- sn1myid^{o}(userid)
  - Output: the user identifier of the respective owner of the social networking service
- sn1user^{io}(userid, name)
  - Input: the user identifier
  - Ouput: the name of the user
- sn1status^{ioo}(userid, statusid, creationdate)
  - Input: the identifier of a user
  - Ouput: the details of statuses like the status identifier and the creation date
- sn1viewcount^{io}(statusid, count)
  - Input: the status identifier
  - Ouput: returns the view count

sn1myid

userid
1
…

sn1user

userid	name
1	John
…	…

sn1status

userid	statusid	creationdate
1	12	05-12-02
…	…	…

sn1viewcount

statusid	count
12	5
…	…

Example: SN_2 operations

- sn2myid^{oo}(userid, name)
  - Ouput: the user identifier and name of the respective owner of the social networking service.
- sn2update^{iooo}(userid, statusid, creationdate, count)
  - Input: the identifier of a user
  - Output: the details of statuses like the status identifier, creation date and view count.

sn2myid

userid	name
54	Chris
…	…

sn2update

userid	statusid	creationdate	count
54	12	05-12-02	4
…	…	…	…

[Locks and keys]

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

Commonly Used Solution

1. Read the documentation
2. Coding a specific wrapper (with a programming language)
3. Coding business specific code (with a programming language)

Example of programming languages

JAVA, C++, C#, …

➔ procedural languages

Commonly Used Solution

1. Read the web service API documentation
   - Determine interesting operations
   - Determine message formats (XML, JSON etc.)
   - Determine API authentication mechanisms
   - For every operation, determine the input and output parameters, required HTTP URL, header, method and body contents

➔ In real life, only example inputs and outputs are given.

Commonly Used Solution

2. Coding a specific wrapper with a (procedural) progr. language
   - Translate API input/output parameters to programming language related data structures (int, string, struct, class…)
   - Code to make web service API operation call (framing HTTP method, URL, body etc. on a per-operation basis)
   - Code to receive API operation responses and extract and transform this response to chosen internal format (data structures)

Commonly Used Solution

3. Coding business specific code with a (procedural) progr. language
   - Determining what information is really interesting for the enterprises
   - Determining the sequence of operations to obtain this information from the web service API operations using the code previously written

Estimated cost ? (in time)

- Relative to developer experience
- Relative to complexity of web service API operations
- Approximation:
  it may take hours/days to develop a program to integrate with one web service API.

Twitter Libraries: Java

Twitter Libraries

These libraries, while not necessarily tested by Twitter, should support Twitter API v1.1.

Want your library to be included in this index or need to update the details we have? Submit your library for inclusion!

Libraries built and maintained by Twitter

Java

- hbc — A Java HTTP client for consuming Twitter's Streaming API
- twitter-kit-android — Twitter Kit is a multi-module gradle project containing several Twitter SDKs including TweetComposer, TwitterCore, and TweetUi. It is built on the Fabric platform and uses many shared components.

Libraries built for the Twitter Platform

Multi-platform

- Temboo — *by @temboo* — Framework for working with Twitter via many platforms including iOS, Android, Java, PHP, Python, Ruby, and Node.js

Twitter Library: HBC

```
120   public void run(String consumerKey, String consumerSecret, String token, String tokenSecret)
121           throws InterruptedException, ControlStreamException, IOException {
122     // Create an appropriately sized blocking queue
123     BlockingQueue<String> queue = new LinkedBlockingQueue<String>(10000);
124
125     // Define our endpoint: By default, delimited=length is set (we need this for our processor)
126     // and stall warnings are on.
127     List<Long> followings = new ArrayList<Long>();
128     followings.add(111111111L);
129     followings.add(222222222L);
130
131     SitestreamEndpoint endpoint = new SitestreamEndpoint(followings);
132     Authentication auth = new OAuth1(consumerKey, consumerSecret, token, tokenSecret);
133
134     // Create a new BasicClient. By default gzip is enabled.
135     BasicClient client = new ClientBuilder()
136             .hosts(Constants.SITESTREAM_HOST)
137             .endpoint(endpoint)
138             .authentication(auth)
139             .processor(new StringDelimitedProcessor(queue))
140             .build();
141
142     // Create an executor service which will spawn threads to do the actual work of parsing the incoming messages and
143     // calling the listeners on each message
144     int numProcessingThreads = 4;
145     ExecutorService service = Executors.newFixedThreadPool(numProcessingThreads);
146
147     // Wrap our BasicClient with the twitter4j client
148     Twitter4jSitestreamClient t4jClient = new Twitter4jSitestreamClient(
149             client, queue, Lists.newArrayList(listener), service);
150
151     // Establish a connection
152     t4jClient.connect();
153     for (int threads = 0; threads < numProcessingThreads; threads++) {
154       // This must be called once per processing thread
155       t4jClient.process();
156     }
```

Problem 1: API modifications

- Change in message formats (XML ⇔ JSON...)
- Change in authentication mechanisms
- Change in input/output parameters
- Change in sequence of operations required to extract desired information

Problem 2: WS lifetime

- What if the user decides to change WS or if WS shuts down ?
- Loss of historical data (unless some data export option is provided)
- Rewrite previous code or make use of some adapter to integrate with new web service

Proposal

- Constraints (recalls)
  - Many WS APIs to integrate
  - Very few developers
- Idea
  - Replacing procedural programming languages by declarative ones
  - View integration as an administration task (not a coding one anymore)

Procedural vs declarative

- Procedural program: « HOW »
  - Developers describe the steps (computations) the system has to follow to reach the intended result.
- Declarative program: « WHAT »
  - Developers: describe the intended result.
  - System: finds the steps to compute it.
  - Same outputs for same inputs
  - No side-effect

Procedural vs declarative

- Procedural ➔ declarative
  - Loops ➔ recursive calls
  - Mutable variables ➔ immutable variables
- Examples of more or less declarative languages
  - SQL (database querying)
  - Prolog (logic programming)
  - Lisp (functional programming)
  - CHR (constraint programming)
  - XML, JSON (content and presentation description)

Recalls about declarative lang.

- Any programming language can be used declaratively when
  - It has a sufficient number of libraries, or
  - It is based on a generic reasoning
    - Prolog and SLD-resolution
    - SQL and associated processing engines (query optimizer and memory management handler).
- Declarative programming
  - an additional abstraction layer
  - often a new syntaxe

Advantages of declarativeness

- Smaller code ➔ safer code
  - less syntax errors
  - conception errors: easier to locate and understand
- Closer to natural language ➔ easier to understand
- ➔ improved development time (e.g. prototyping)
- ➔ improved scalabilty

Drawbacks of declarativeness

- Dependence on the used libraries, reasoning or engine
  ➔ declarative languages often are specialized (associated to a specific usage)
- The recursive way of thinking has to be practised.
- Handling complex data structures may be tricky.

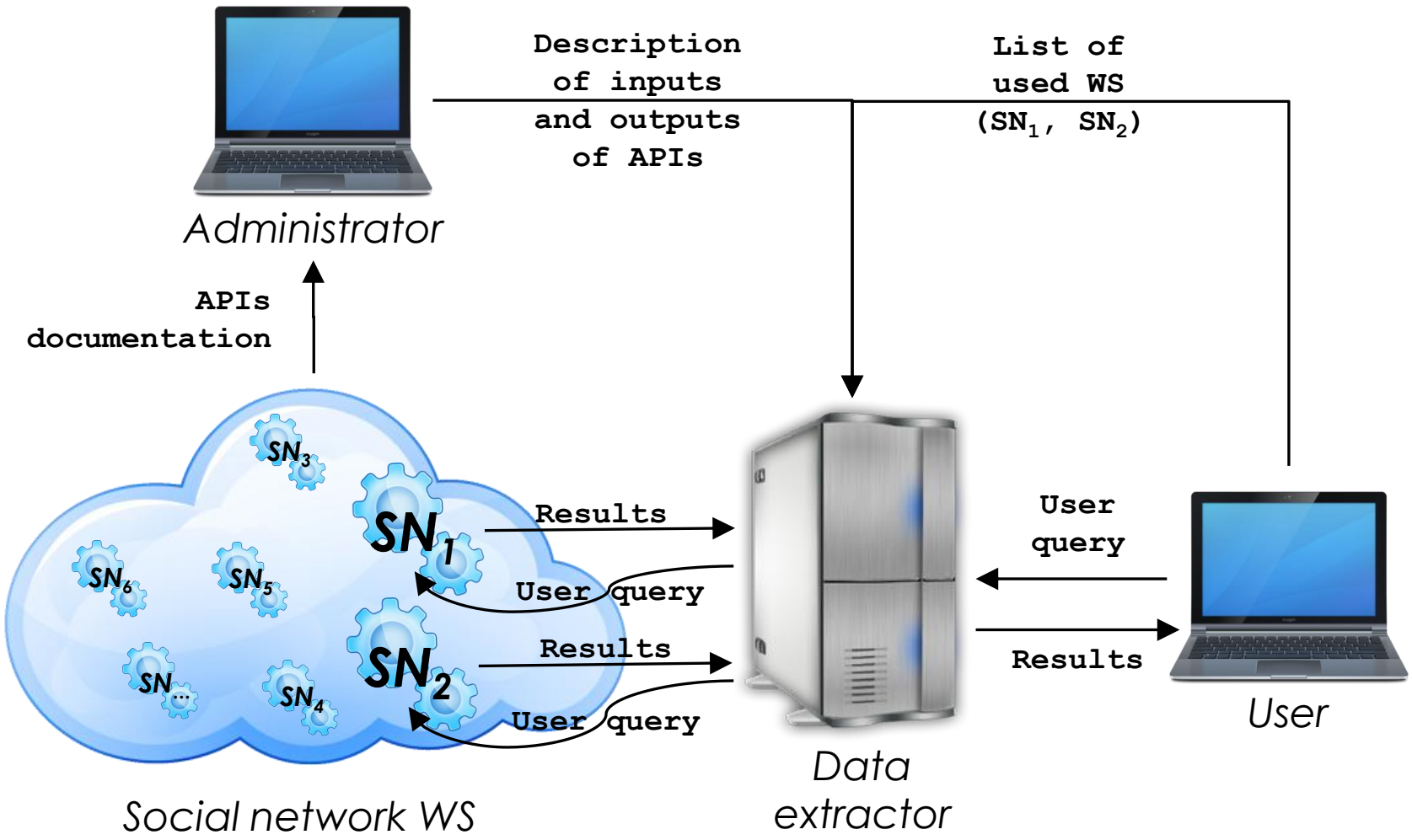
Two main usages

1. Building big programs
   ➔ declarative software development
   ➔ needs developers
2. Configuring an existing program or reasoning
   - Modifying parameters
   - Adding components, data
   - Expressing queries
   - Specifying presentation rules

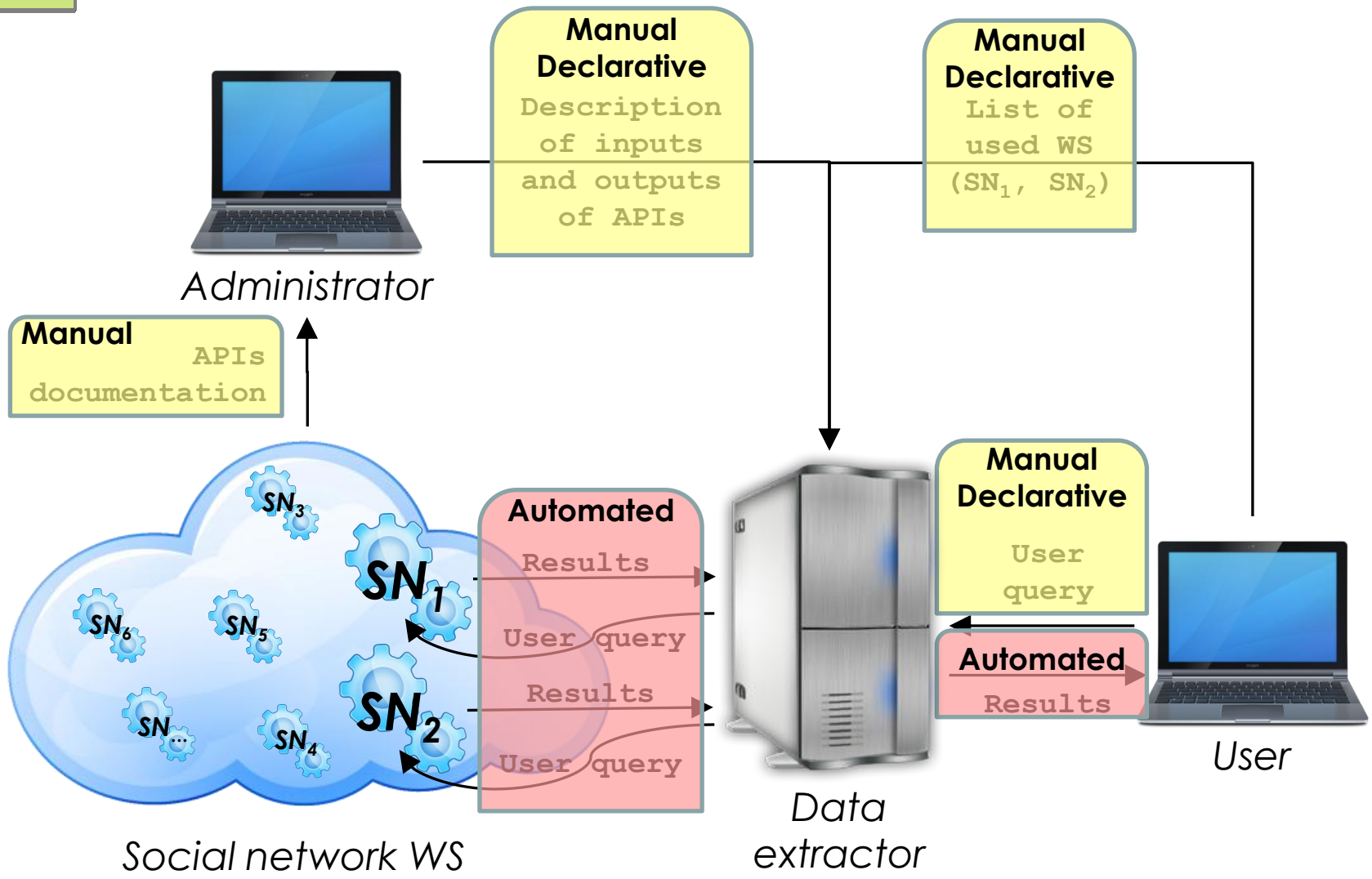
   ➔ declarative software administration
   ➔ needs administrators

➔ We choose the second option.

Declarative data extractor



Declarative data extractor



Proposed workflow

1. Read the web service documentation
2. Declaratively set up the data extractor
3. Consuming data by querying
    1. Asking for data by querying
    2. Dealing with results via a database (DB)

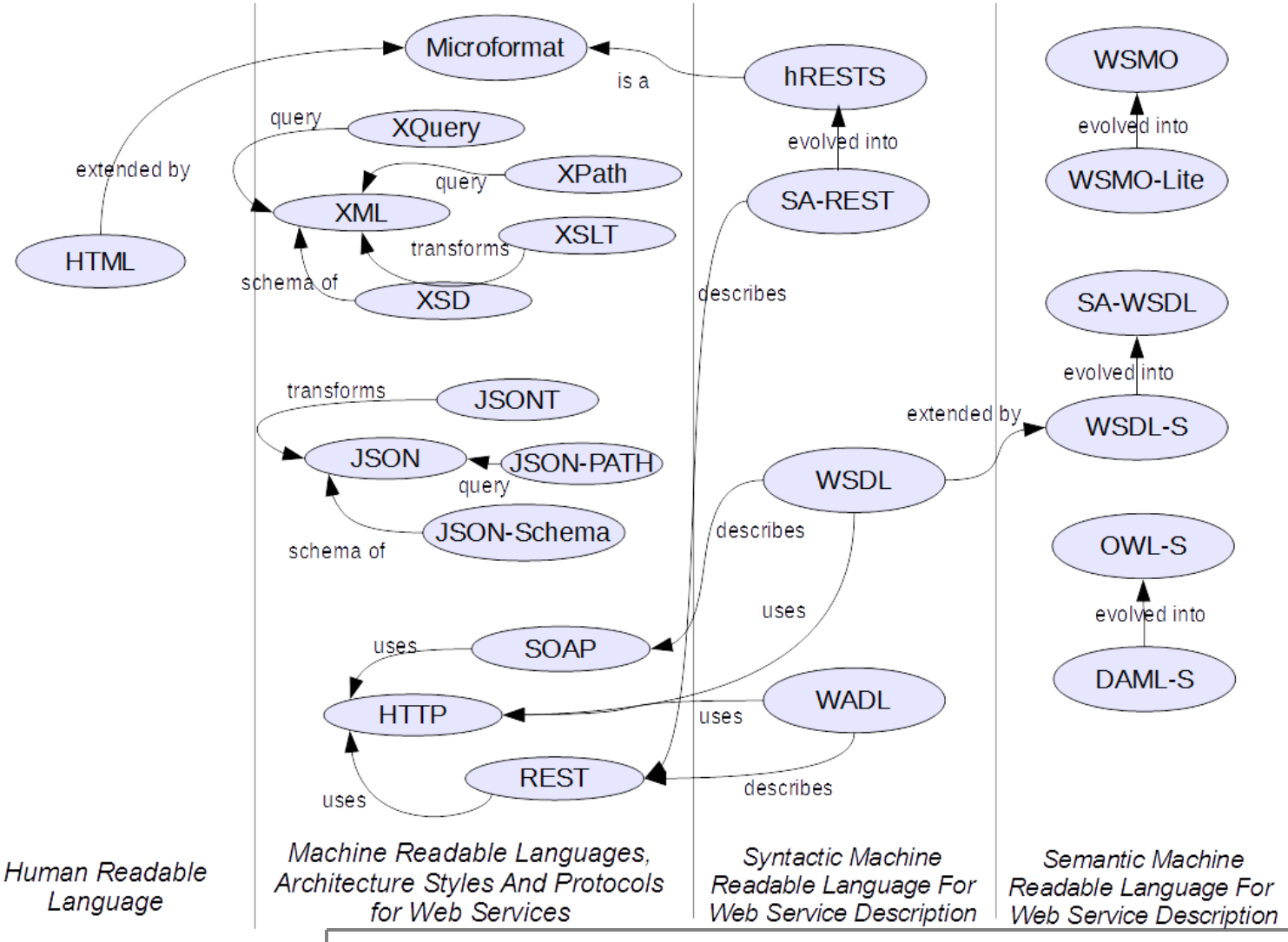
Envisioned benefits

- Reduced administration time
  - A new WS = a new DB entry
  - A new WS version = a new DB entry
  - Api modifications = DB updates
    (e.g. change of input/output parameters)
    ➔ no need to code any more
- Estimated cost in time ?
  - Shifting to minutes/hours administration tasks (and not hours/days coding tasks)

Which declarative languages ?

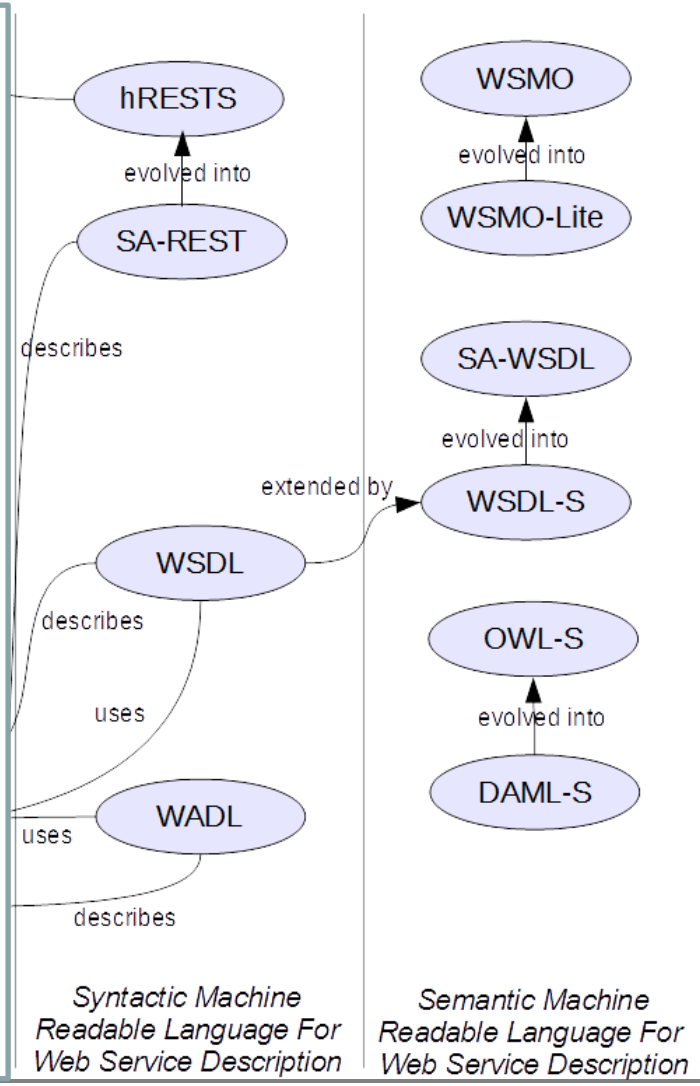
- Assume the idea is feasible
- Which declarative language ?
- What to do exactly ?
- Chosen declarative languages must be as much as possible
  - Simple
  - Well-known
  - Widespread (already used by many people)

WS languages overview



WS languages overview

Languages ➔ are not widely used (yet).



About WSDL

- Definition
  - Web Service Description Language
  - ➔ possible to automatically generate WS wrappers
- Problems
  - WSDL-generated wrappers are pieces of (procedural) code to be inserted in a larger program ➔ need a developer
  - Chaining API calls
    - One output becomes an input ➔ API calls plans
    - Not easy to handle such plans
    - Solution: manage as query plans ➔ declarative approach
  - WSDL is not widely used
    ➔ not declarative enough ?

Which declarative language ?

- Previous candidates
  - not widely used
- What a pity !
  - ~~WSDL → automatically generated wrapper~~
  - ~~Semantic web languages → automatic discovery of web services~~
  - ~~…~~

In the real world

- Documentation
  - Natural language
  - Examples with code snippets
- Message format
  - JSON
  - XML

➔ now, we must deal with this !

Challenge

- Limit the manual work to
  - Reading APIs documentation
  - Declaratively describing APIs to the system (system set up)
  - Declaratively express user queries
- Automate
  - Calls to WS APIs
  - User query processing
  - Results management
- Allow a quite expressive user query language

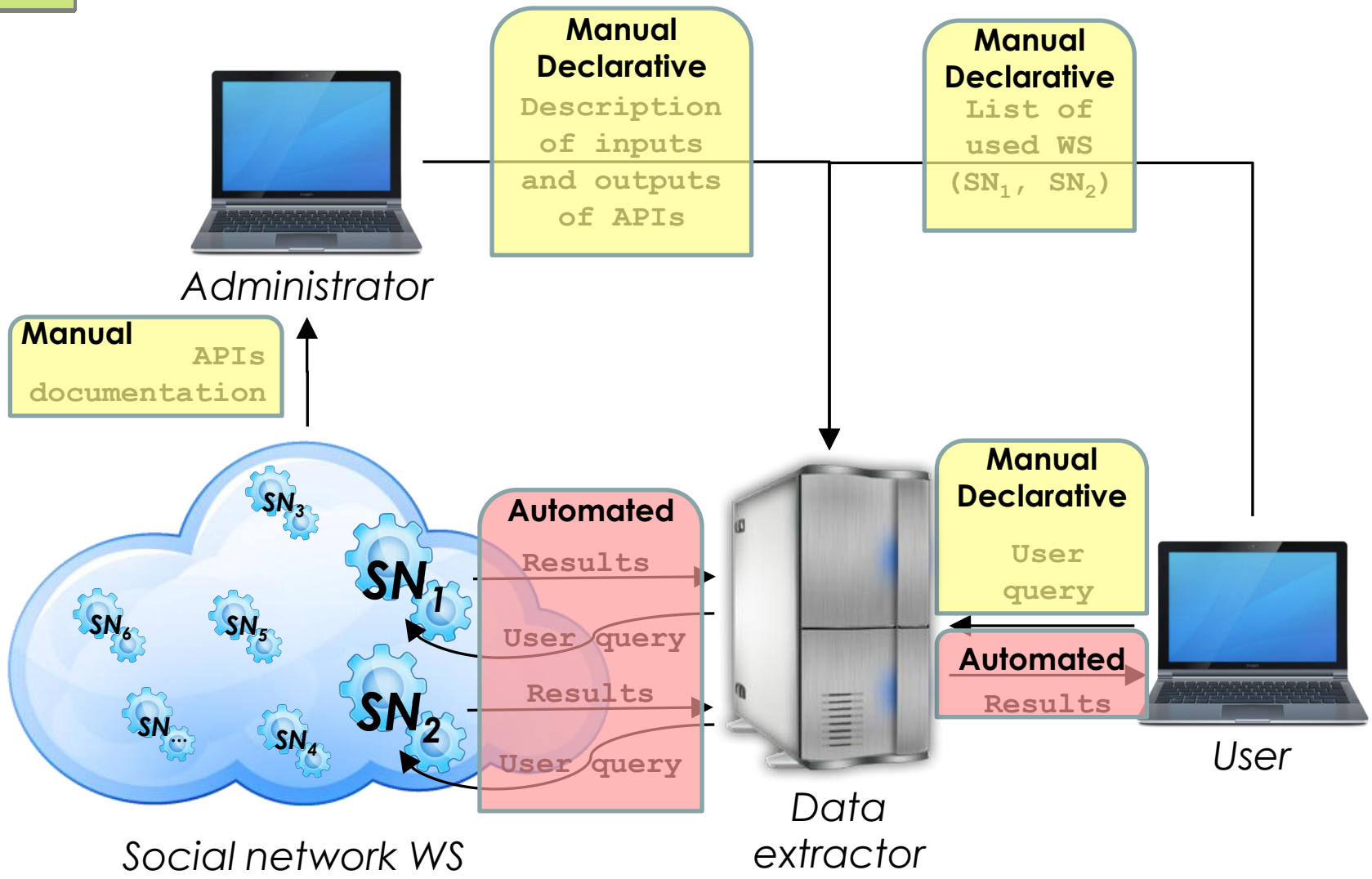
➔ possible with widely used declarative languages ?

[Declarative data extraction]

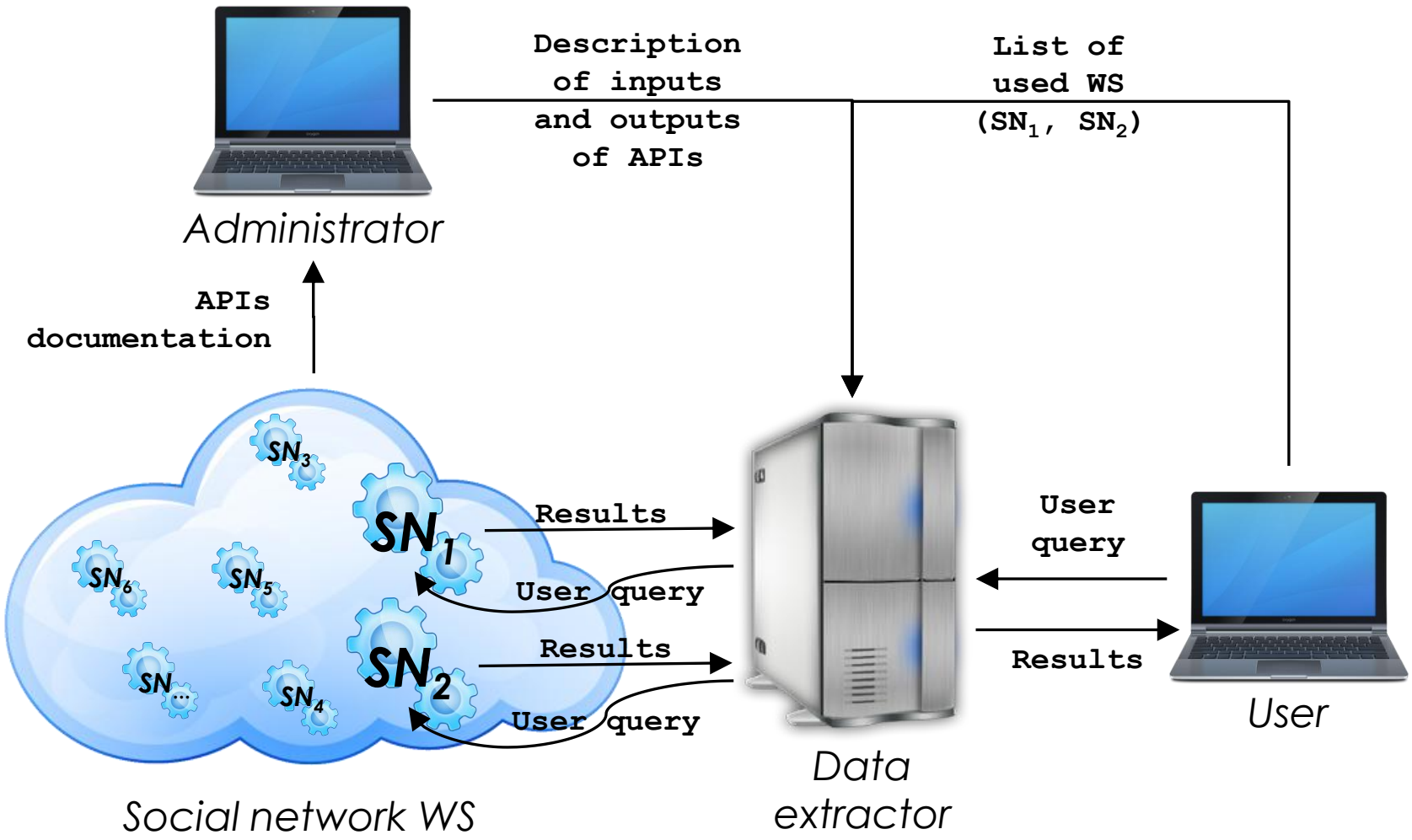
Getting into the details…

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

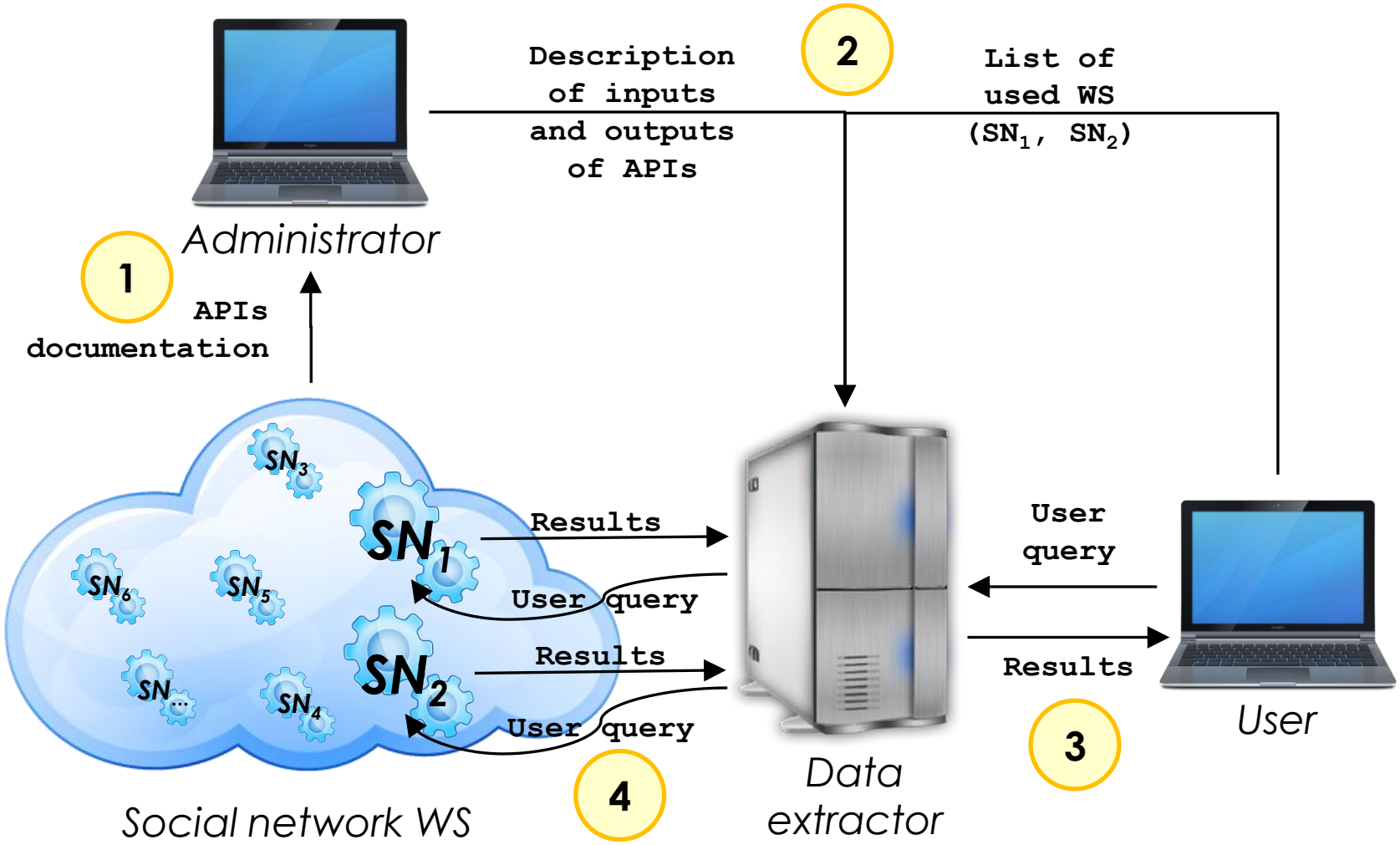
Declarative data extractor



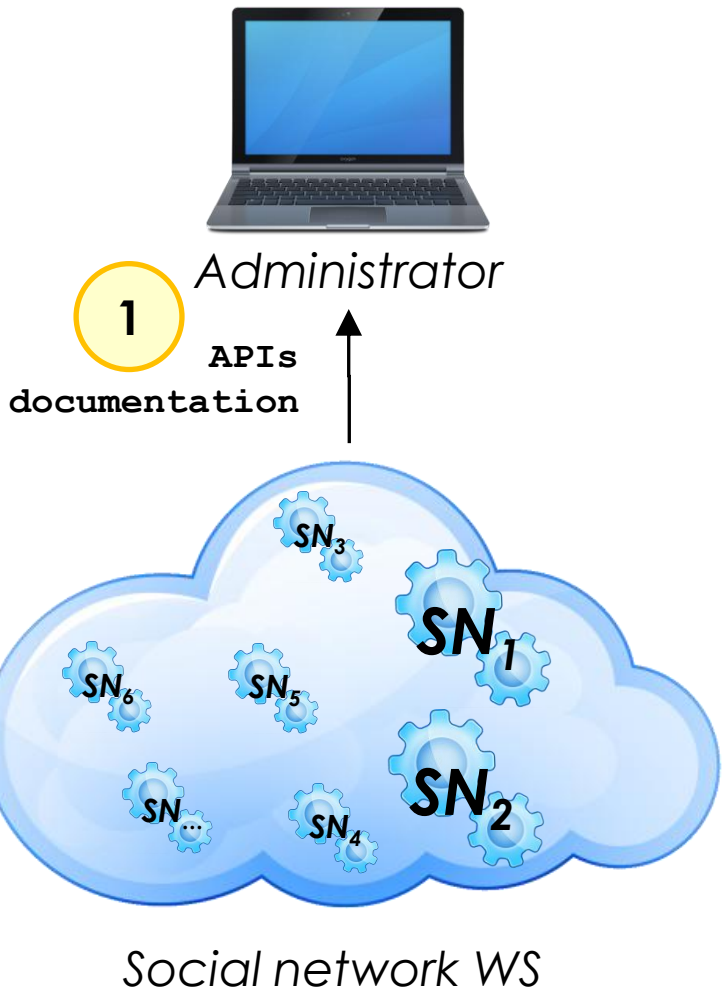
Declarative data extractor



4 use cases

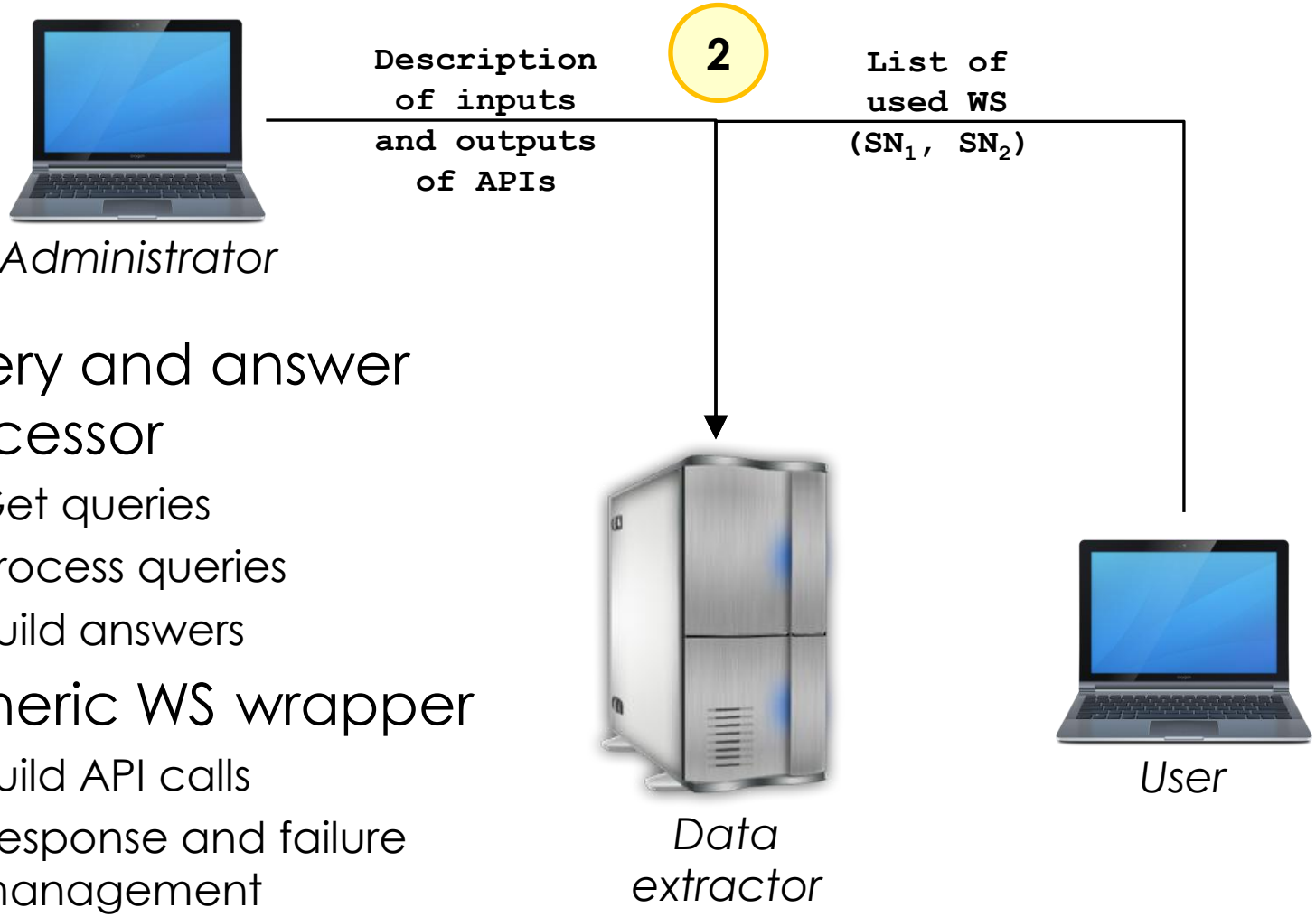

Description of inputs and outputs of APIs
2
List of used WS (SN_1, SN_2)
Administrator
1
APIs documentation
SN_3
SN_1
SN_6
SN_5
SN_2
SN
SN_4
Results
User query
Results
User query
4
Social network WS
Data extractor
User query
Results
3
User

Read the WS API doc.



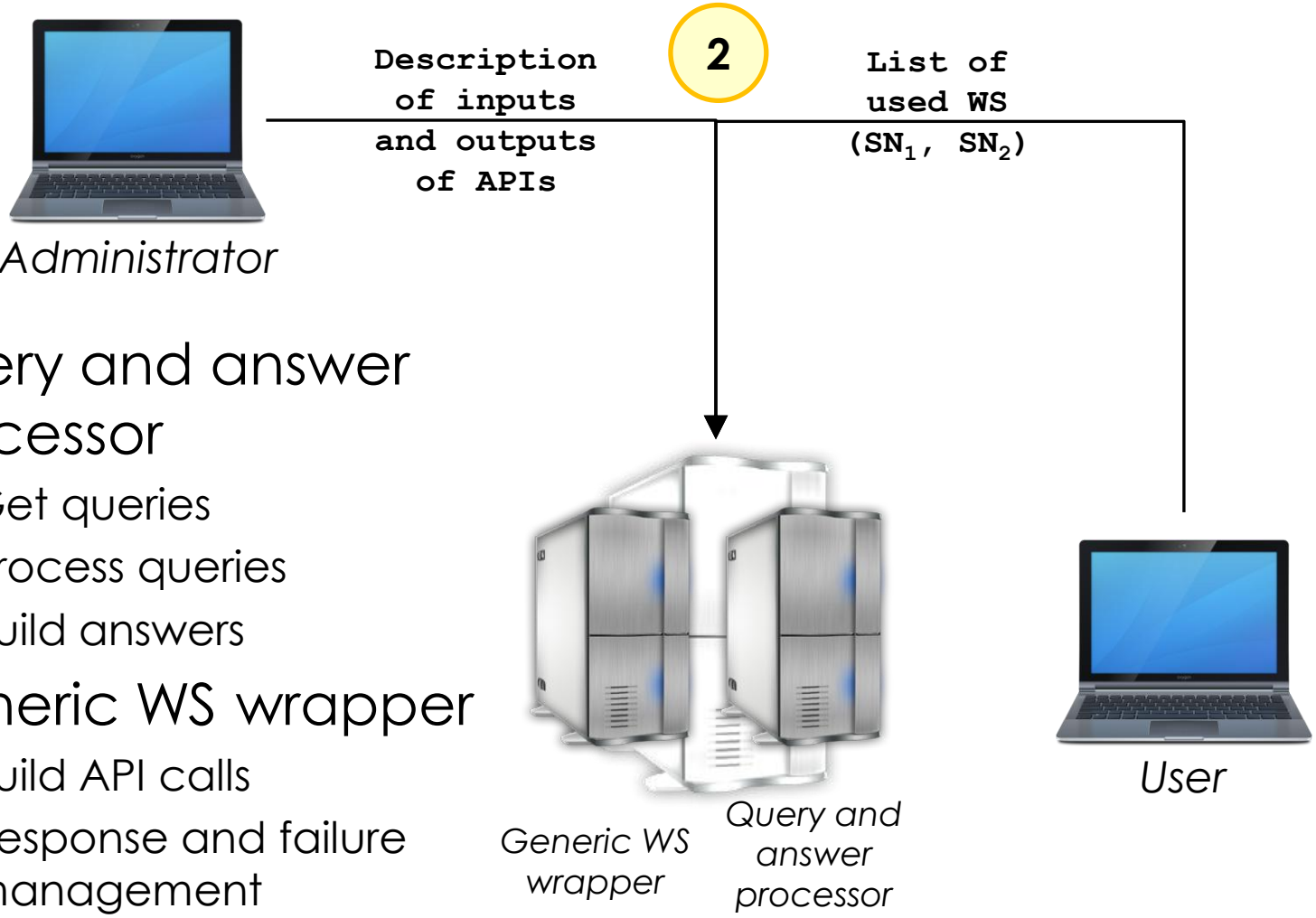
- Determine operations
- Determine message and response formats
- Determine API authentication mechanisms
- For every operation, determine
  - the input and output parameters, and
  - required HTTP URL, header, method and body contents

Inside the data extractor



- Query and answer processor
  - Get queries
  - Process queries
  - Build answers
- Generic WS wrapper
  - Build API calls
  - Response and failure management

Inside the data extractor



- Query and answer processor
  - Get queries
  - Process queries
  - Build answers
- Generic WS wrapper
  - Build API calls
  - Response and failure management

Declarative set up

Administrator

Description of inputs and outputs of APIs

2

List of used WS (SN_1, SN_2)

- Describe WS API operation call (HTTP method, URL template, …) ➔ language ?
- Set up the Generic Wrapper
  - Building of operation calls
  - Processing of responses into chosen internal formats
- Set up HTTP body template
- Set up validation, extraction and transformation details required to build responses

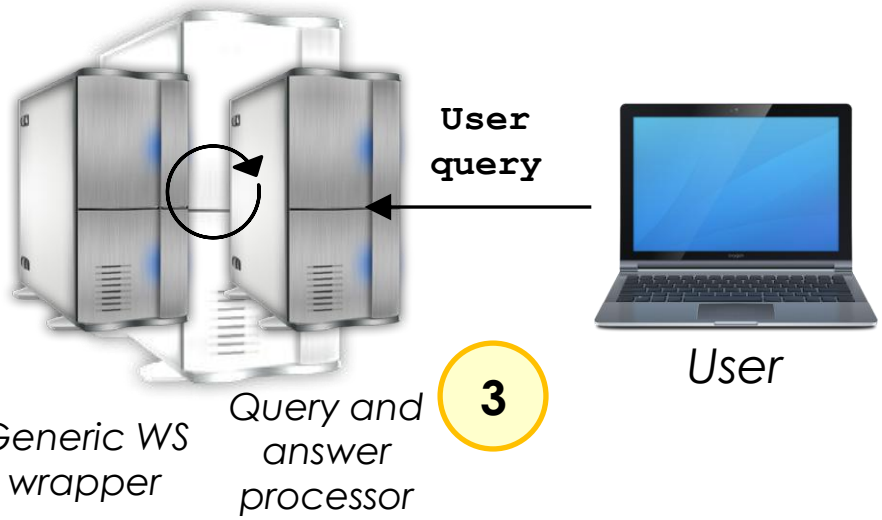
Generic WS wrapper

Query and answer processor

User

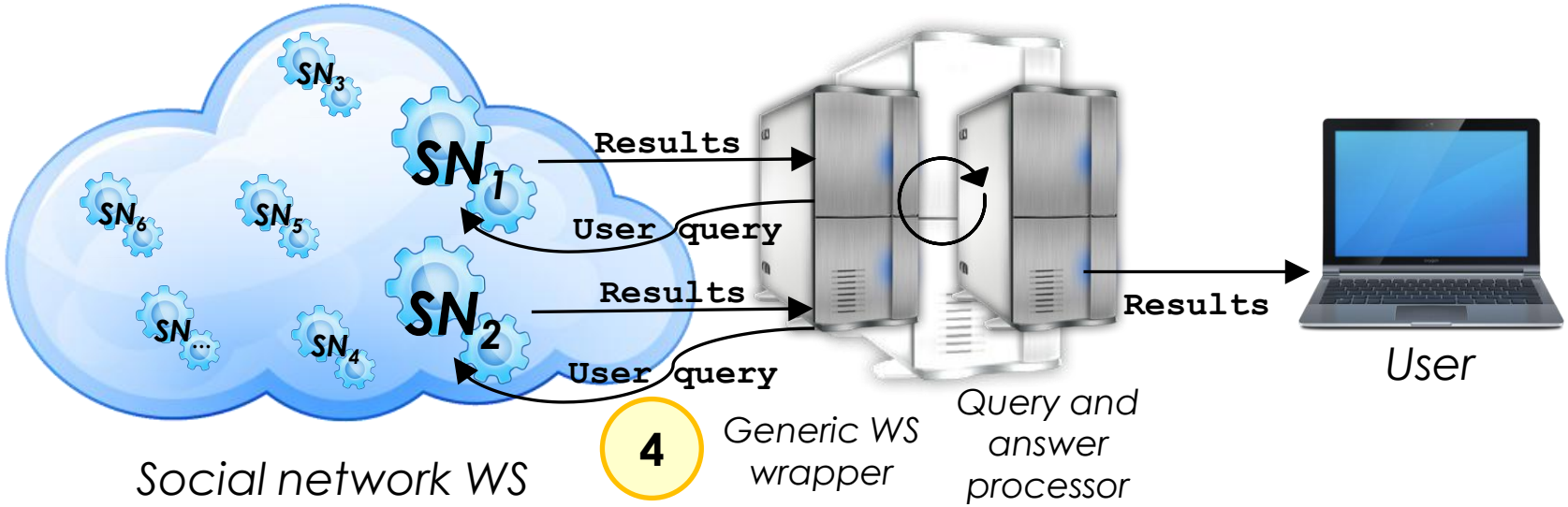
Consuming data by querying

- Querying is declarative !
- Query language : SQL ? Other ?
- Internal formats : ?

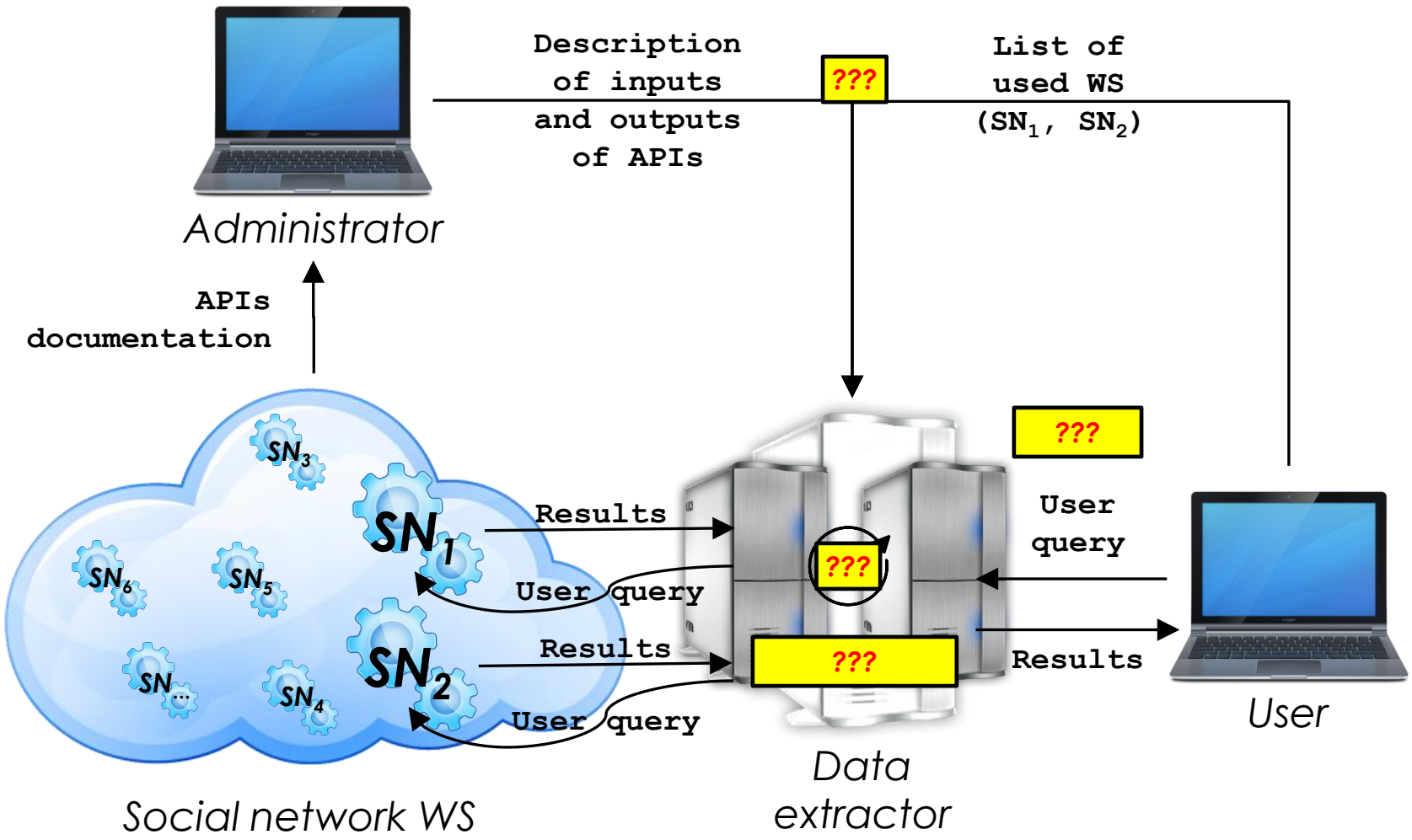


Automated querying processing

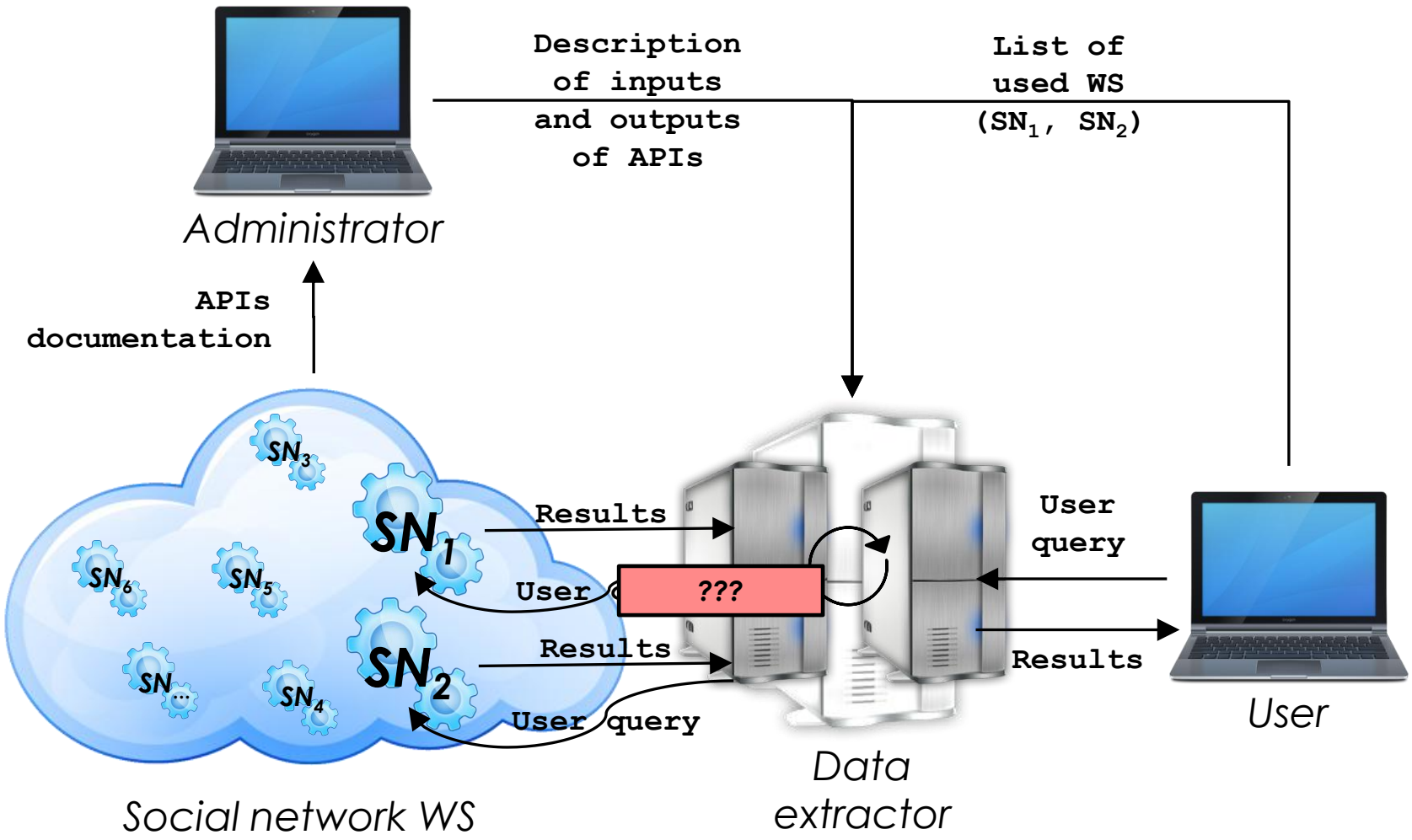
- How to automatically build the API calls ?
- How to deal with responses and potential failures ?
- Response processing formats ?



Languages ?


Description of inputs and outputs of APIs
???
List of used WS (SN_1, SN_2)
Administrator
APIs documentation
SN_3
SN_1
SN_6
SN_5
SN_2
SN
SN_4
Results
User query
Results
User query
???
???
???
User query
Results
User
Social network WS
Data extractor

Automated processing ?



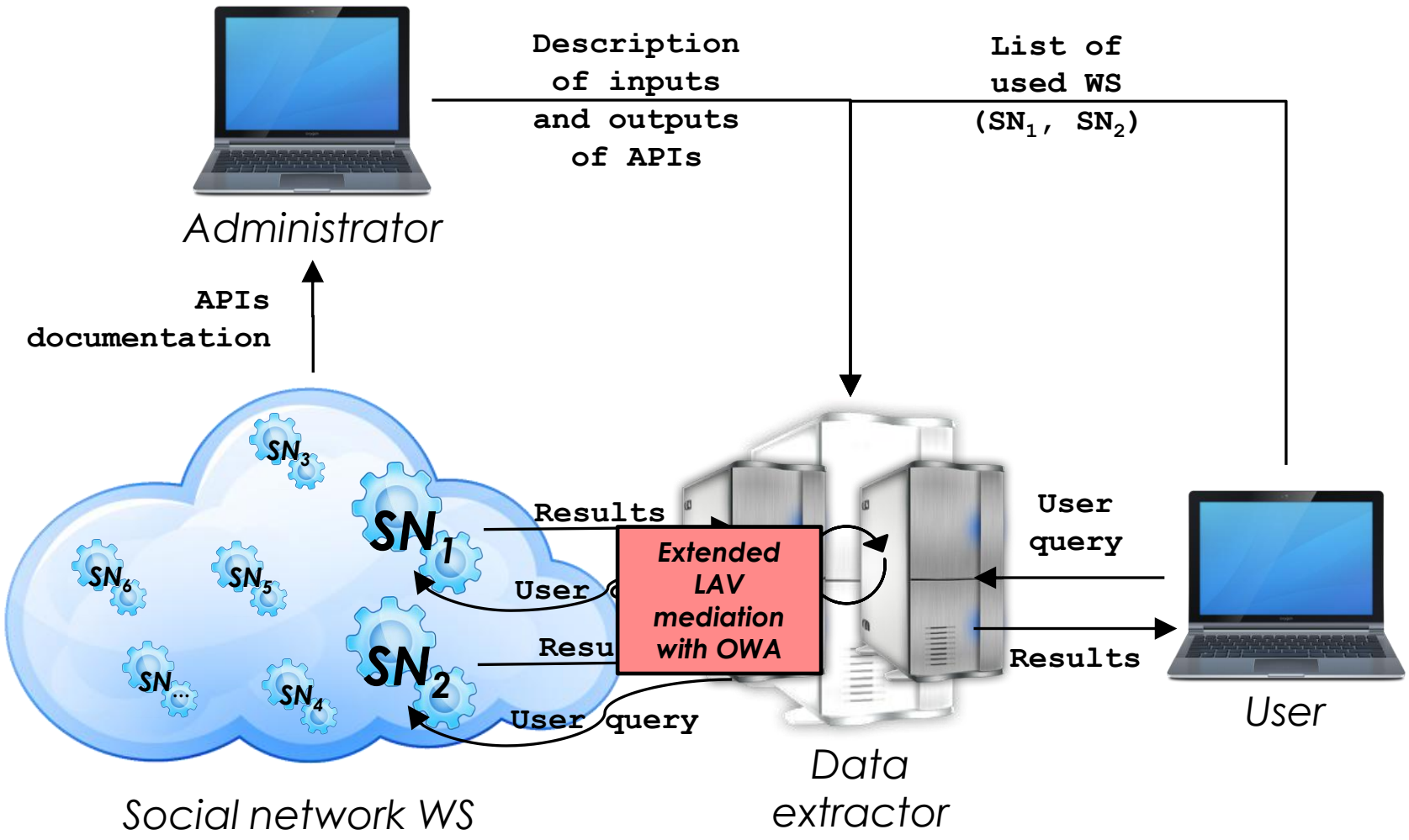
A data integration problem

- WS = data sources
- WS operations = relations with access patterns
  (some attributes are inputs, some ar outputs)
- One user query to be posed to many WS operations via WS APIs.
- Constraints:
  - Declarative links between WS and the data extractor
  - Use of widely spread declarative languages

Followed approach

- Basis : mediation with LAV mappings and OWA.
- Use of the well-known Inverse Rules algo
  - Access patterns
  - Full and functional dependencies
- Mappings are CQ with access patterns
- User queries are datalog queries.
- Extension
  - To deal with incomplete information (heuristics)
  - To handle pagination

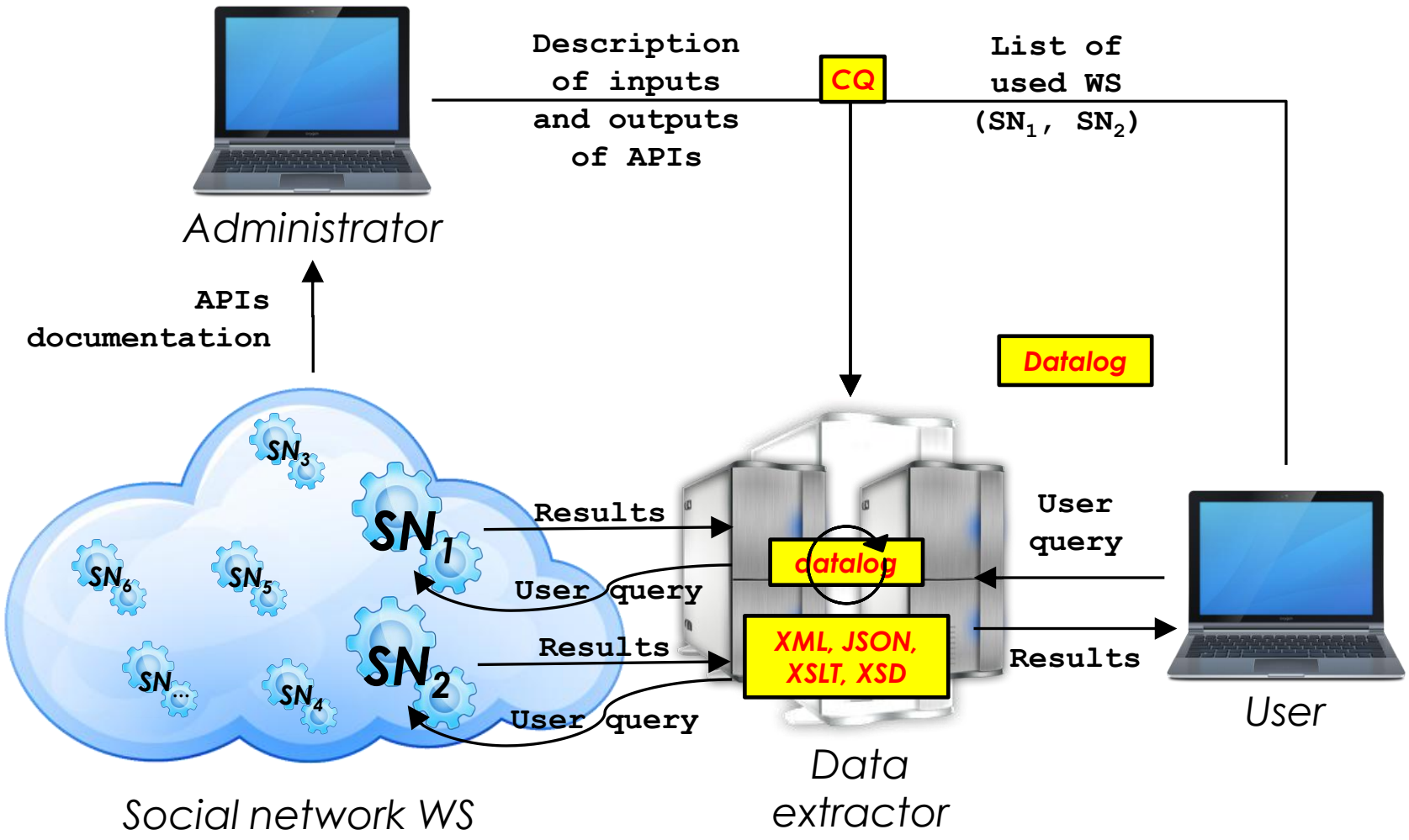
Followed approach



Declarative languages used in this work

- Presentation languages/ message formats ➔ those which are used
  - JSON / XML
  - XSD
  - XSLT
- Query languages
  - Conjunctive queries
  - Datalog

Back to the extractor



Conjunctive queries CQ

- Equivalent to
  - select-project-join queries
  - i.e. SELECT… FROM … WHERE … ; in SQL
- Example
  - Relational schema: {r1(A,B), r2(C,D), r3(E,F,G)}
  - ans(X) ← r1(X,Y),r2(Y,Z),r3(Z,X,W).
  - SELECT r1.A
    FROM r1, r2, r3
    WHERE (r1.B = r2.C) and (r2.D = r3.E) and (r1.A=r3.F);

Datalog queries

- Equivalent to
  - Recursive unions of conjunctive queries
- Example
  - Transitive closure of a graph
  - Relational schema: {edge(X,Y)}
  - path(X,Y) ← edge(X,Y).
    path(X,Y) ← edge(X,Z),path(Z,Y).

JSON

```
{
    "updates": [{
        "identifier": 1245,
        "update": "I hope the weather is good",
        "date": "2016-03-04T10:20:18+02:00",
        "count": 14
    }, {
        "identifier": 1246,
        "update": "It's sunny here",
        "date": "2016-03-04T14:25:16+02:00",
        "count": 14
    }]
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
    <viewcount>
        <count>123</count>
    </viewcount>
</xml>
```

XSD

```
<xs:schema attributeFormDefault="unqualified"
 elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="xml">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="viewcount">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:byte" name="count"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <xsl:for-each select="statuses/status">
         <xsl:value-of select="identifier" />
         ,
         <xsl:value-of select="text" />
         <xsl:text />
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

[Data integration with WS]

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

Data integration approaches

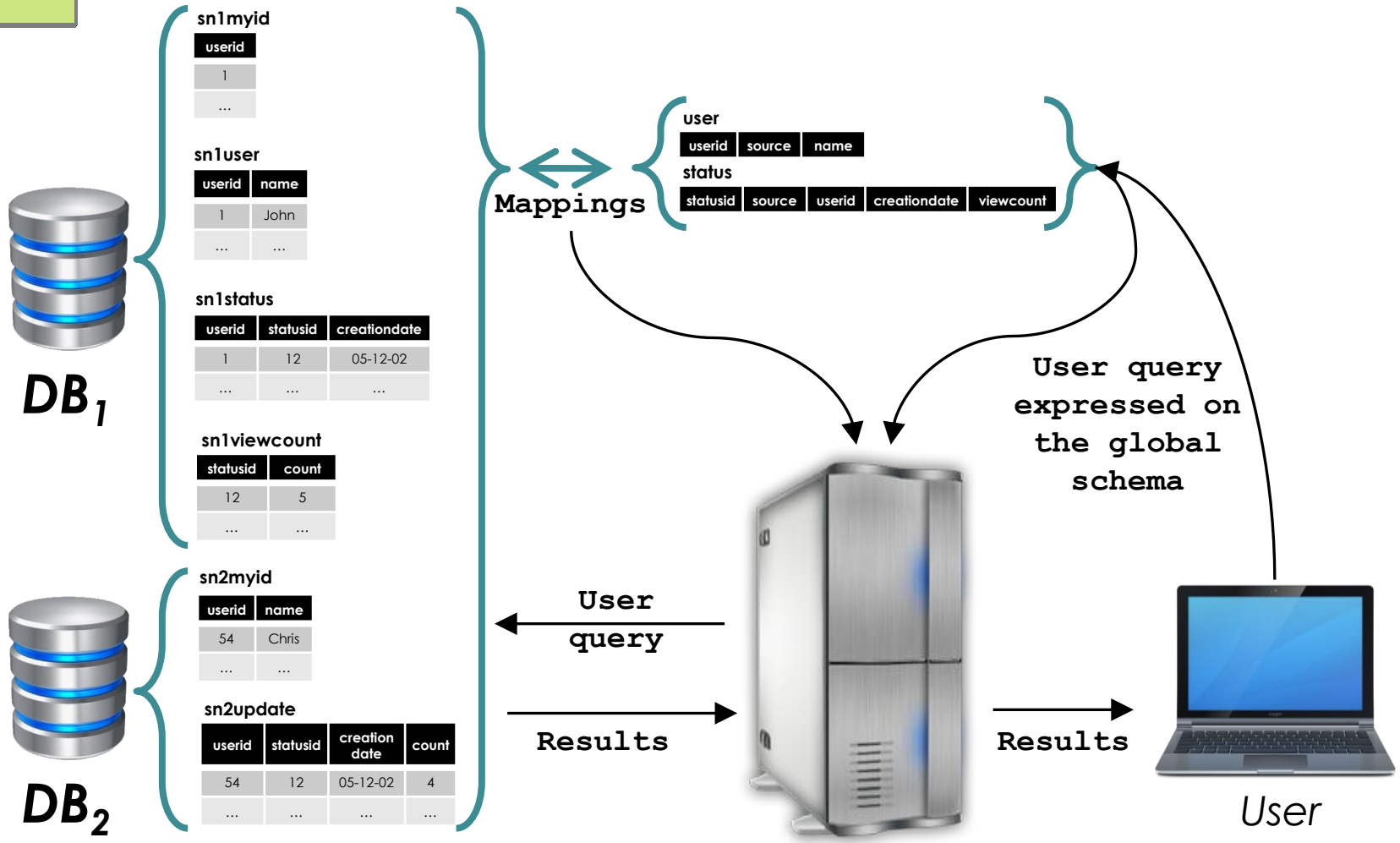
- Materialized ➔ data warehousing
  - Source data are stored locally and maintained over time
  - Main purposes: statistics, decision support
  - Analytical queries
- Virtual ➔ mediation
  - Source descriptions only (no data) are stored locally
  - Main purpose: « lightweight » heterogeneous sources querying
  - Transactional queries (no modification)

Dealing with WS

- Access patterns
  ➔ complete data unavailable
- Great number of WS
- High versatility
  - Data keep on changing
  - WS keep on evolving
- Networking costs
- Transactional « get » queries only

➔ The mediation data integration approach fits well.

Mediation in a nutshell



Private data sources with public views

Mediator with a virtual global schema

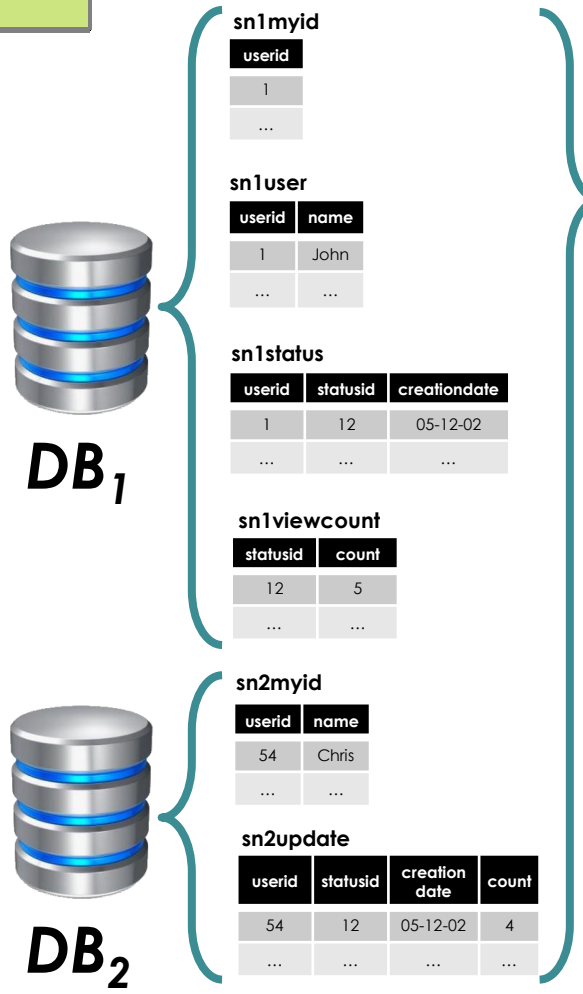
Mediation in a nutshell

- Main problem:
  (User) query answering using (data from the) views
- Certain answers semantics
- Depend on the mappings:
  - GAV mappings: « Global As View »
  - LAV mappings: « Local As view »
  - GLAV mappings: « Global and Local As View »

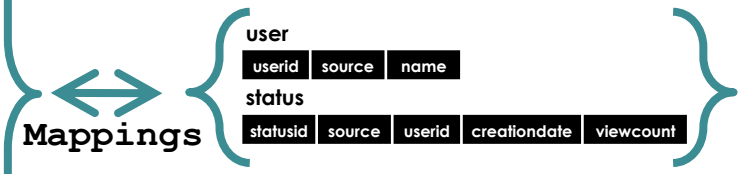
Mediation in a nutshell

- GAV mappings:
  - Each global schema predicate = a query on the schema made up with all views (i.e. the source schema).
  - Query answering by unfolding the user query: replace global schema predicates by their definition
- LAV mappings:
  - Each view = a query on the global schema.
  - Query answering by query rewriting:
    - The user query is rewritten (if possible) into one or more query rewritings.
    - Each rewriting is a query plan that is answered by calling one or many views.
- GLAV mappings:
  - Sort of generalized LAV case
  - Link (inclusion) between a query on the global schema and a query on the views schema
  - Query answering by query rewriting

Examples of mappings



Private data sources with public views



- LAV mapping

sn1user(userid,name**)** ← **user(**userid,'SN1',name**).**

- GAV mapping

user(userid,'SN1',name**)** ←	**sn1user(**userid,name**),** **sn1myid(**userid**).**
user(userid,'SN2',name**)** ←	**sn2myid(**userid,name**).**

- GLAV mapping

user(userid,source,name**),** ⊇ **status(**statusid,source,userid,creationdate,viewcount**)**	**sn1user(**userid,name**),** **sn1status(**userid,statusid,creationdate**),** **sn1viewcount(**statusid,count**).**

Mediation in a nutshell

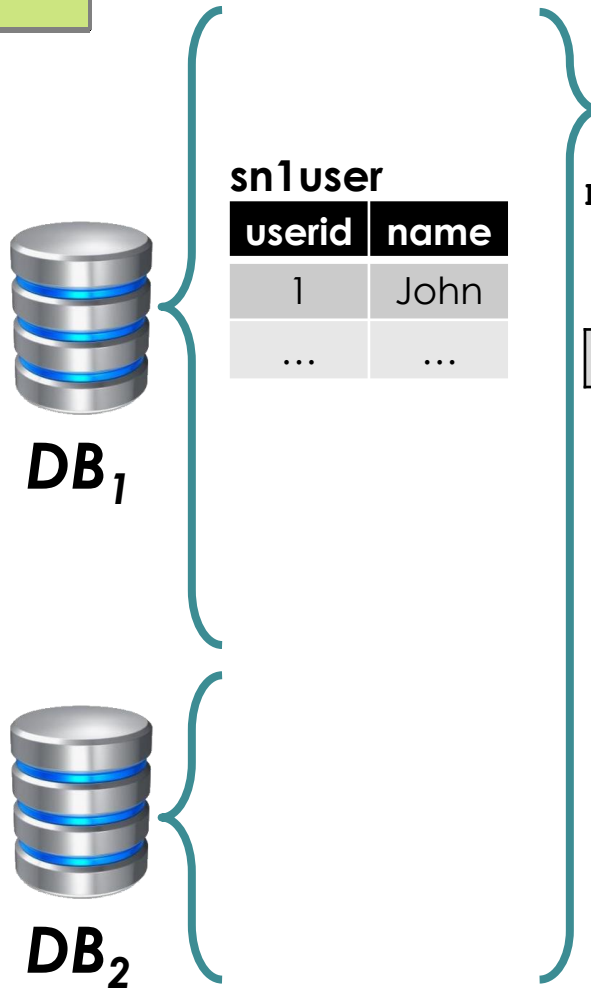
- Pros and cons GAV
  - (+) Query processing expected to be easier
  - (-) Defining global schema not always an easy task
  - (-) Adding new sources impacts the global schema
- Pros and cons LAV
  - (+) Modularity/Scalability (e.g., adding new sources do not impact the global schema)
  - (-) Query processing is more complex

Mediation in a nutshell

- Pros and cons GLAV
  - (+) Modularity/Scalability (e.g., adding new sources do not impact the global schema)
  - (+) Highly expressive mappings
  - (-) Query processing is more complex
  - (-) Handle recursive queries ?

- Conclusion
  - ➔ LAV best suited for WS integration

LAV mappings



Private data sources with public views

- Example

sn1user(userid,name**)** ← **user(**userid,'SN1',name**).**

- If « GS was materialized », then:
  couples (userid,name) of sn1user would be in triples (userid,'SN1',name) from user
- A conjunctive query
- Declarative !

LAV mappings

DB_1

sn1user

userid	name
1	John
…	…

DB_2

Private data sources with public views

LAV Mappings

user

userid	source	name

Mediator's virtual global schema GS

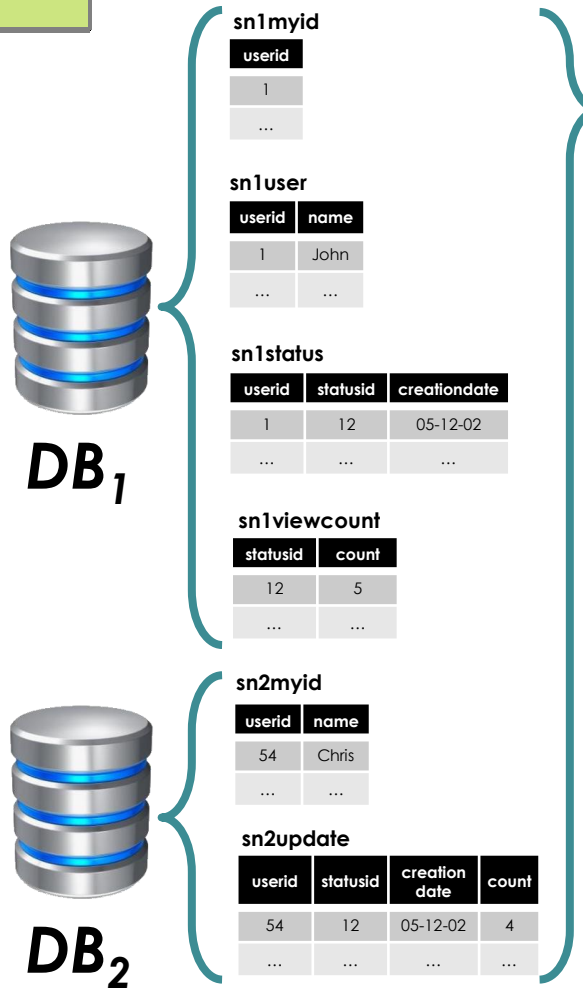
- Example

sn1user(userid,name**)** ← **user(**userid,'SN1',name**).**

- Assumptions
  - Sound views
    « all couples from sn1user are included in triples of user »
  - Incomplete views
    « a couple that is not in sn1user may be true or false (we don't know) »
    ➔ a.k.a. the open world assumption (OWA)

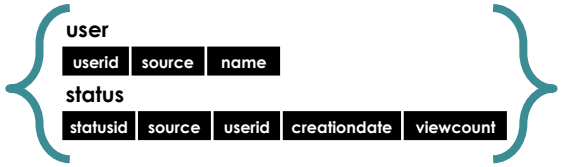
➔ LAV with OWA

LAV mappings



Private data sources with public views

LAV Mappings



Mediator's virtual global schema GS

sn1user(userid,name) ← **user**(userid,'SN1',name).

sn1myid(userid) ← **user**(userid,'SN1',name), **status**(statusid,'SN1',userid,creationdate,viewcount).

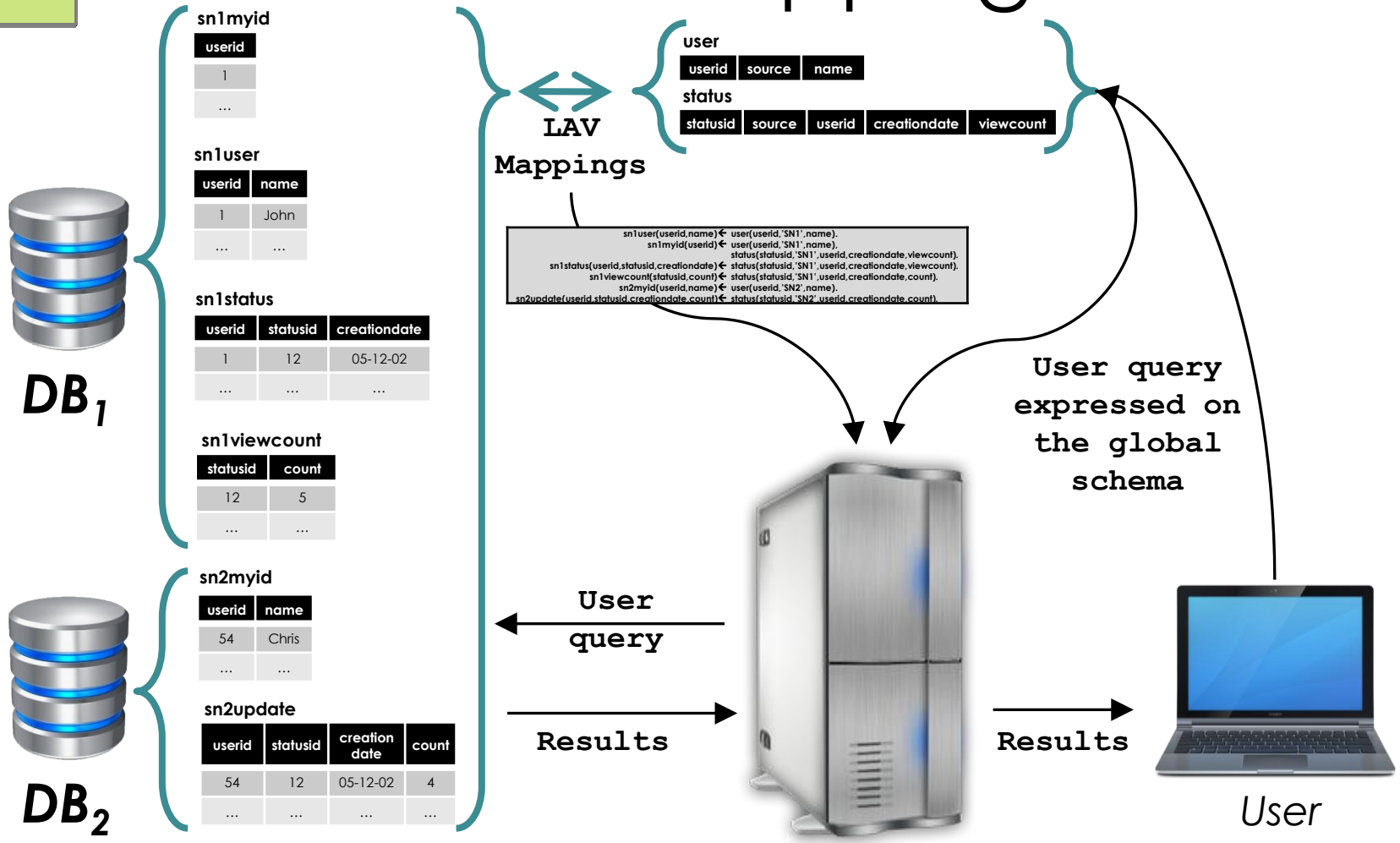
sn1status(userid,statusid,creationdate) ← **status**(statusid,'SN1',userid,creationdate,viewcount).

sn1viewcount(statusid,count) ← **status**(statusid,'SN1',userid,creationdate,count).

sn2myid(userid,name) ← **user**(userid,'SN2',name).

sn2update(userid,statusid,creationdate,count) ← **status**(statusid,'SN2',userid,creationdate,count).

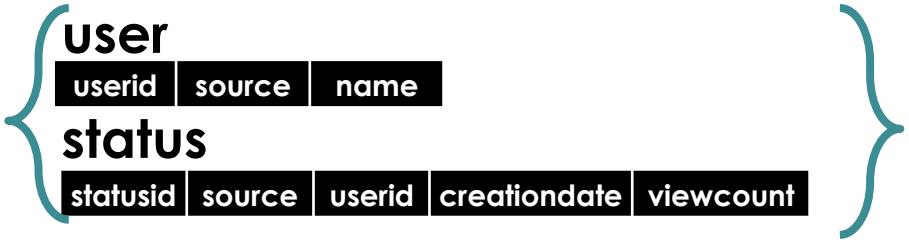
LAV mappings



Private data sources with public views

Mediator with a virtual global schema

User query



- Example 1

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Give me all triples (userid,statusid,viewcount) which creation date is 2016-03-03 and source is 'SN1'.

➔Conjunctive query (subset of datalog)

User query

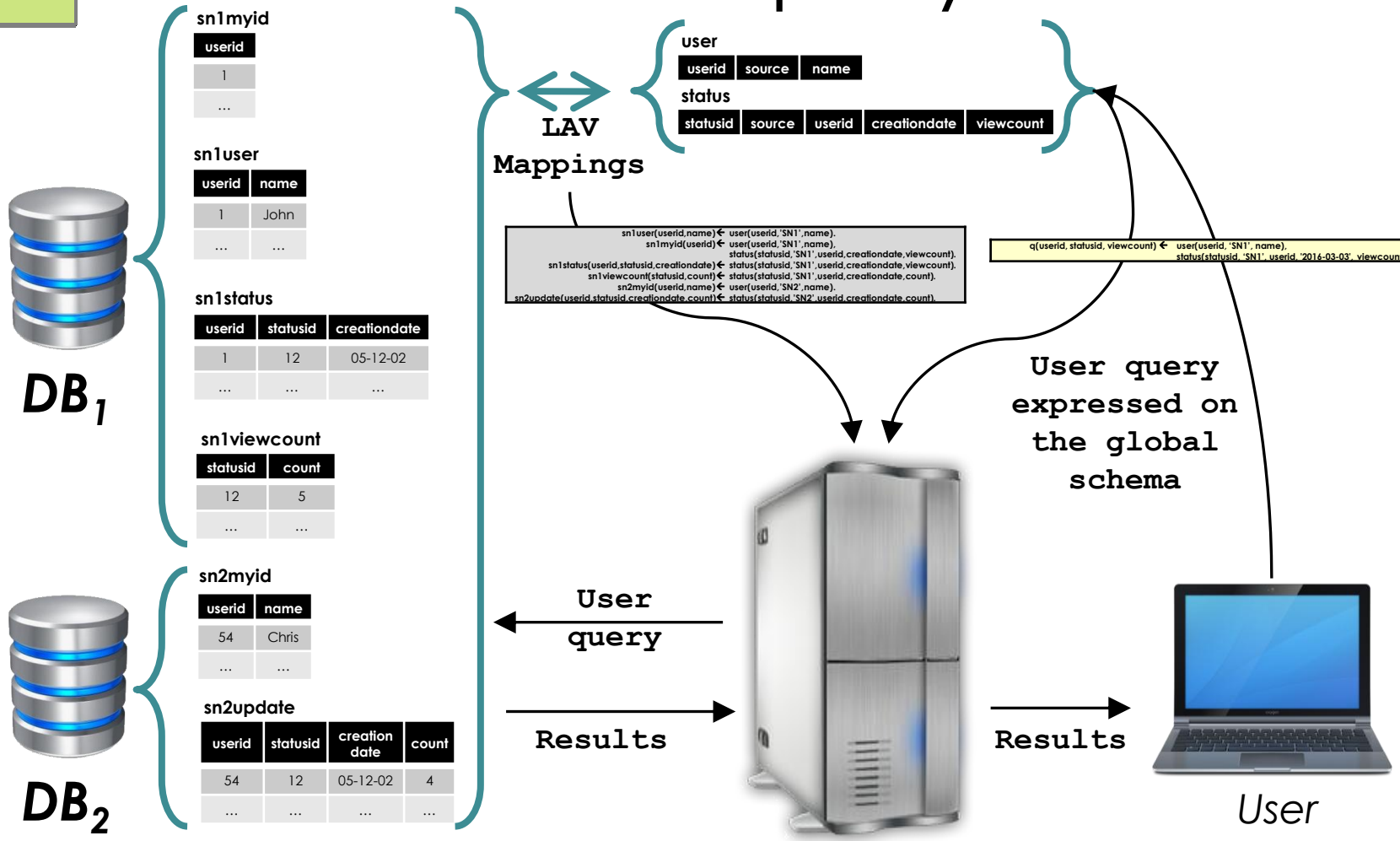
- Example 2

q(statusid, source, viewcount) ←status(statusid, source, userid, '2016-03-03', viewcount)
q(statusid, source, viewcount) ←status(statusid, source, userid, '2016-03-04', viewcount)
q(statusid, source, viewcount) ←status(statusid, source, userid, '2016-03-05', viewcount)

- Give me all triples (statusid,source,viewcount) which creation date is 2016-03-03, or 2016-03-04 or 2016-03-05.

➔Union of conjunctive queries (subset of datalog)

User query



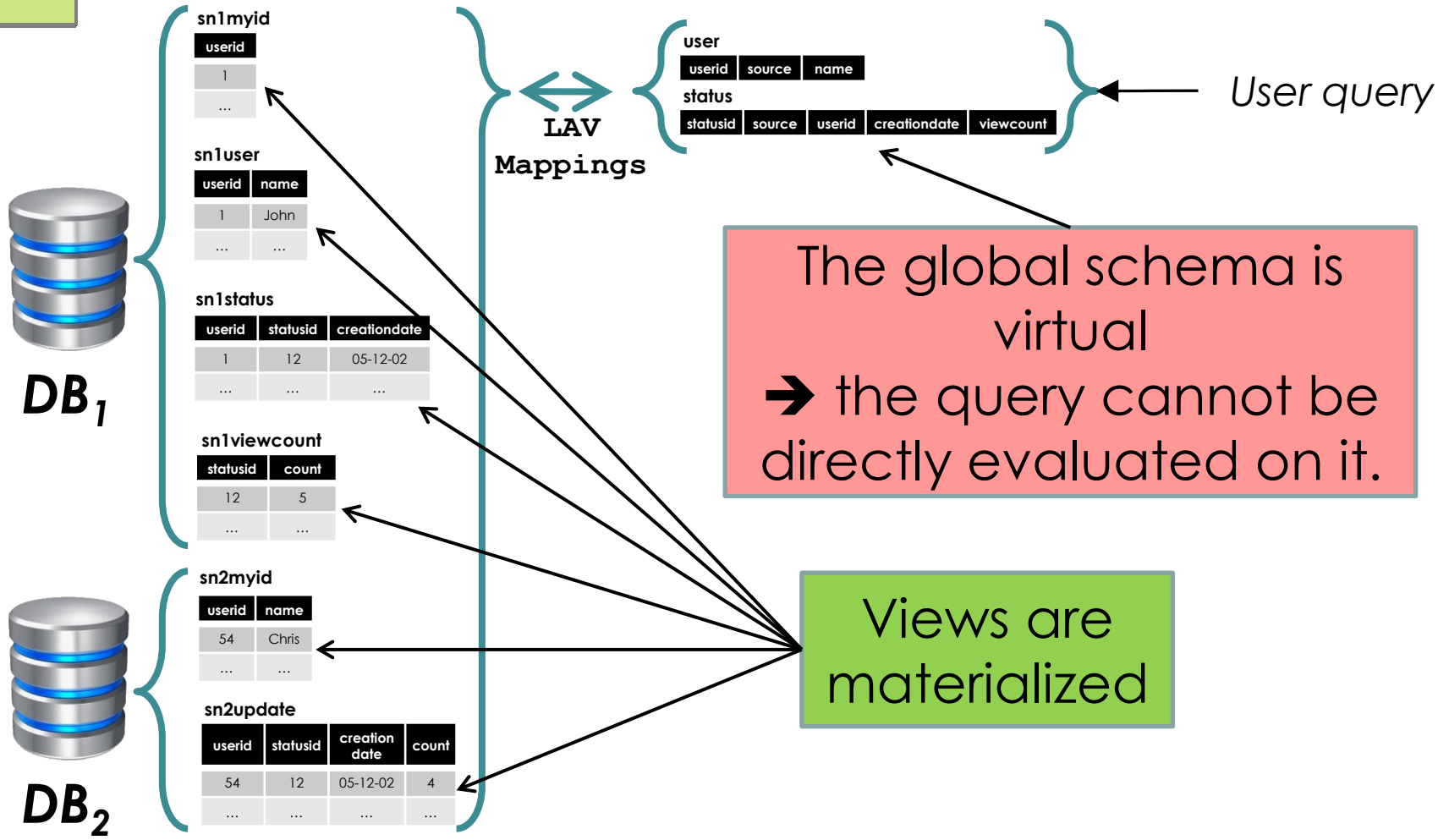
Private data sources with public views

Mediator with a virtual global schema

Query answering using views

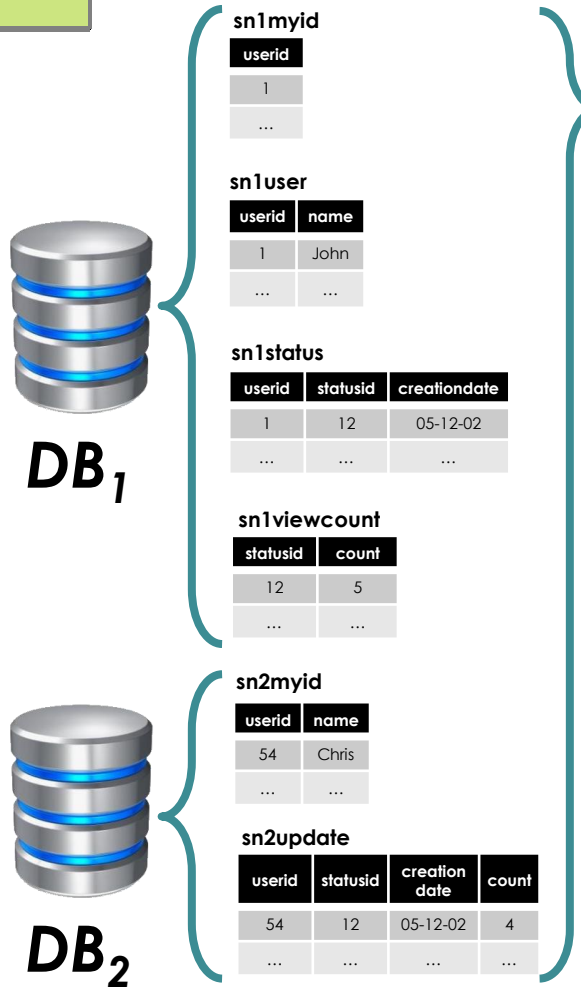
- Assume
  - LAV mappings with OWA
  - A user query posed to the global schema
- Certain answers semantics [AD98]
  - Finding « certain » answers
  - i.e. answers that would be obtained, whatever materialization of the global schema consistent with the LAV mappings with OWA, if the user query was directly evaluated on the global schema

Certain answers semantics

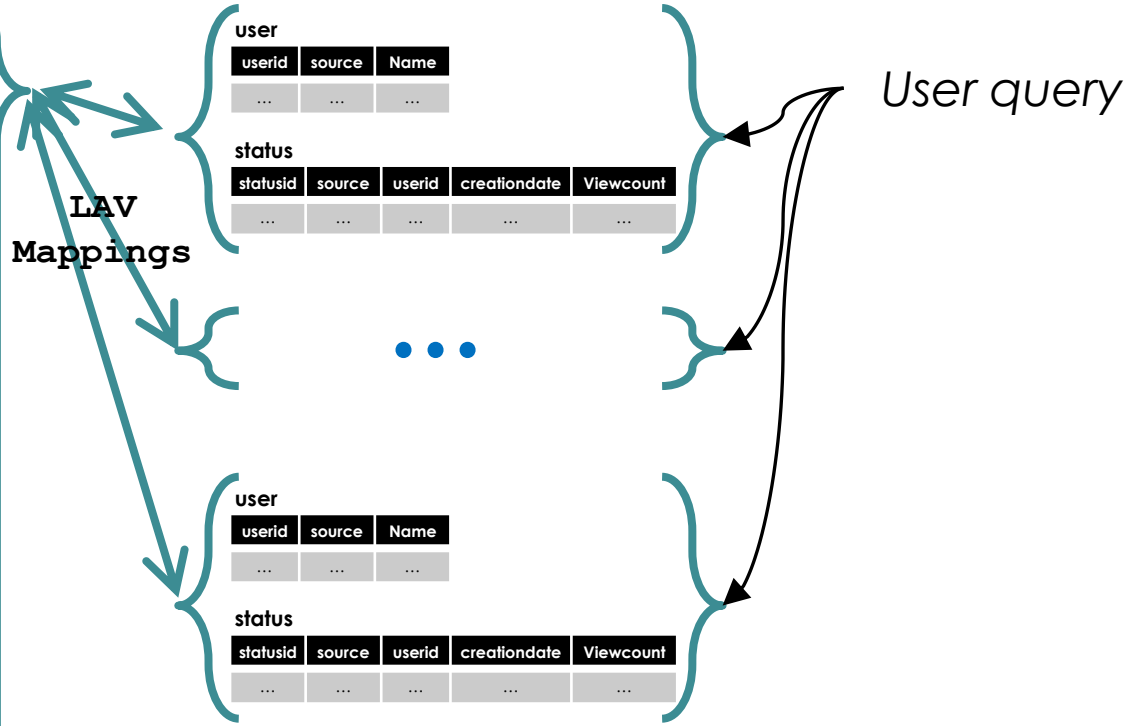


Private data sources with public views

Certain answers semantics

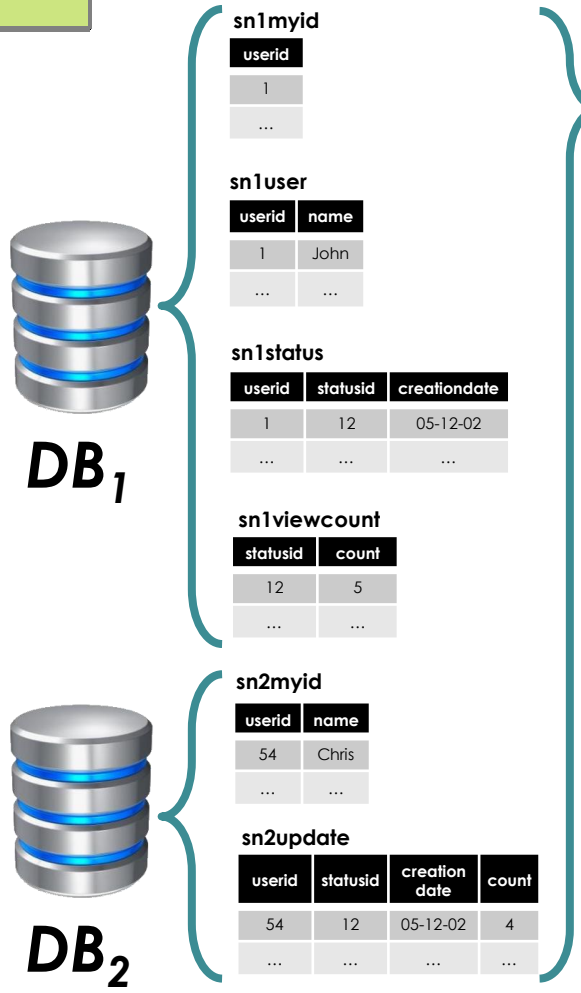


Private data sources with public views

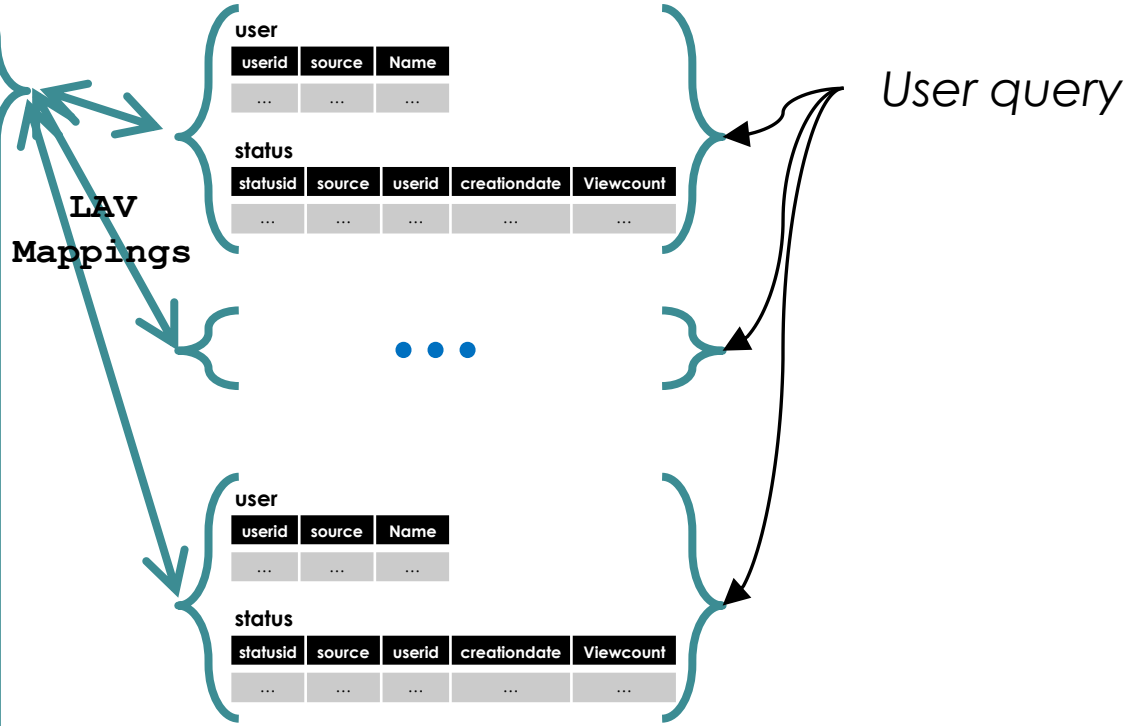


All possible materializations of the global schema consistent with LAV mappings with OWA

Certain answers semantics

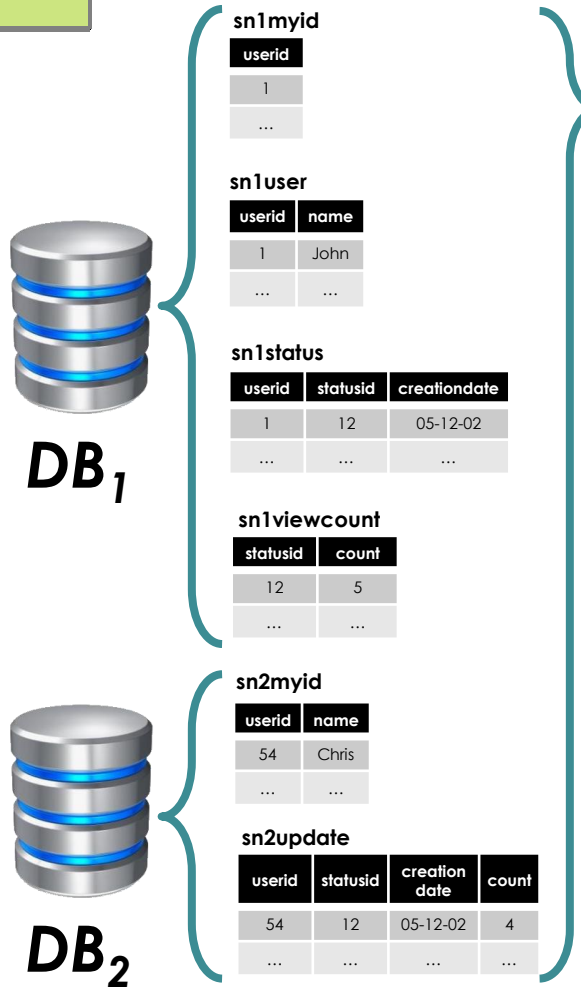


Private data sources with public views

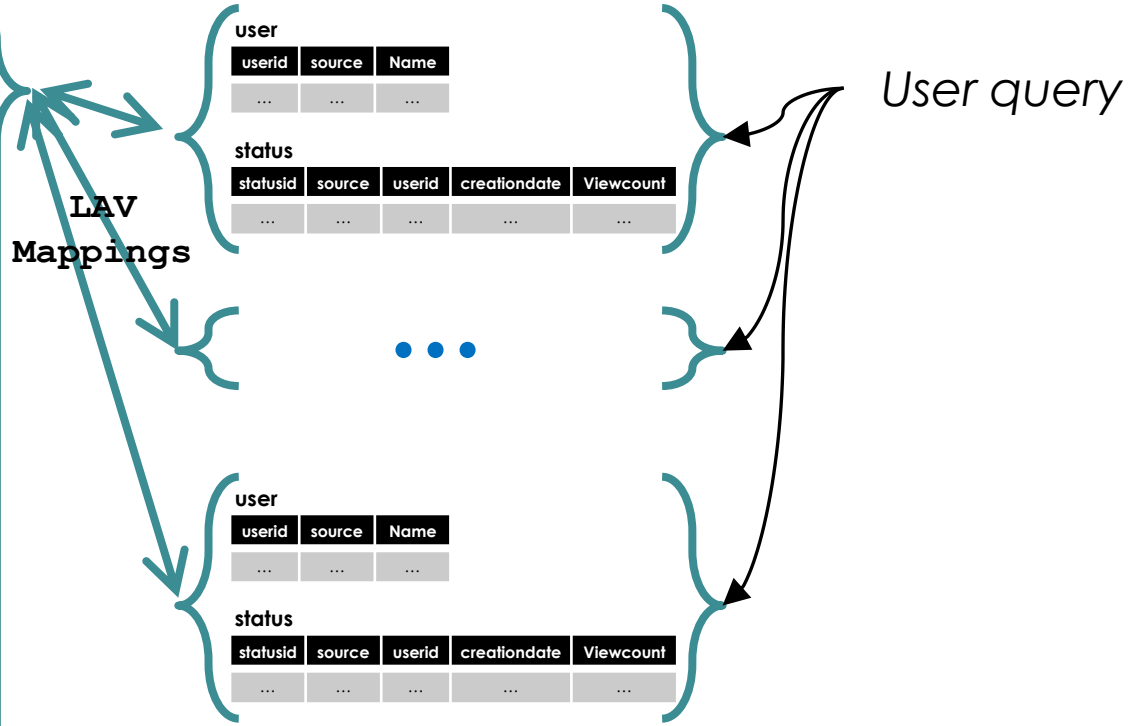


These materializations
➔ depend on the source data
➔ the query can be evaluated on them.

Certain answers semantics



Private data sources with public views



Certain answers *= the intersection of all answers when the query is evaluated on all materializations.*

Computing certain answers

- Assume
  - A set of views with their extensions Is (i.e. their data)
  - A set S of LAV mappings as conjunctive queries, with OWA
  - A datalog user query Q posed on the global schema
- Then [AD98]
  - Certain answers of Q wrt S and Is are computed by any query plan Q' that is maximally contained in Q wrt S.
  - A query plan = a datalog query with only view predicates

Max. contained query plans

- A query plan P
  - a datalog query with only view predicates
  - Can be « expanded » or « unfolded » predicates are replaced by their definitions (from the LAV mappings)
  - P^{exp} is the expanded query plan
- P is a maximally contained query plan in a datalog query Q iff
  - P^{exp} contained in Q
  - For each query plan P' s.t. $(P')^{exp}$ contained in Q there is $(P')^{exp}$ contained in P^{exp}

Query containment

- Classical result
  - For CQ [CM77]
  - For datalog [S93]
- CQ containment is NP-complete.
  - P $\subseteq$ Q iff there exists a substitution θ on variables(Q) s.t.
    - atoms(θ(Q)) $\subseteq$ atoms(P), and
    - var(head(θ(Q)) = var(head(P))
- Datalog containment is undecidable.

Complexity results [AD98]

- Data complexity of the problem of computing certain answers under OWA

	query Q				
views s	CQ	$CQ^{\neq}$	PQ	*datalog*	FO
CQ	*PTIME*	*Co-NP*	*PTIME*	*PTIME*	und
$CQ^{\neq}$	*PTIME*	*Co-NP*	*PTIME*	*PTIME*	und
PQ	*Co-NP*	*Co-NP*	*Co-NP*	*Co-NP*	und
datalog	*Co-NP*	und	*Co-NP*	und	und
FO	und	und	und	und	und

"und" means undecidable

- Depends on query containment complexity

Running example

- LAV mappings

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Example of query plans

q(userid, statusid, viewcount) ← sn1myid(userid)

q(userid, statusid, viewcount) ← sn1user(userid, name), sn1status(userid,statusid, '2016-03-03'), sn1viewcount(statusid, viewcount).

q(userid, statusid, viewcount) ← sn2update(userid,statusid, '2016-03-03', viewcount).

Running example

- LAV mappings

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Example of query plans
  - P1 = q(userid, statusid, viewcount) ←sn1myid(userid)
  - $P1^{exp}$ = q(userid, statusid, viewcount) ←user(userid,'SN1', _1), status(_2,'SN1',userid, _3, _4).

Query plan P1

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Query plan P1
  - P1 = q(userid, statusid, viewcount) ← sn1myid(userid)
  - $P1^{exp}$ = q(userid, statusid, viewcount) ← user(userid,'SN1',_1), status(_2,'SN1',userid,_3,_4).

- Problem 1
  - Head variables statusid and viewcount are not in the body.
  - When calling the view, we get only userid !

- Problem 2 : $P1^{exp}$ is not contained in Q
  there cannot be any substitution θ s.t.
  θ('2016-03-03') = _3

Query plan P2

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Query plan P2
  - P2 = q(userid, statusid, viewcount) ← sn1user(userid, name), sn1status(userid,statusid, '2016-03-03'), sn1viewcount(statusid, viewcount).
  - $P2^{exp}$ = q(userid, statusid, viewcount) ← user(userid, 'SN1',name), status(statusid,'SN1',userid, '2016-03-03',_1), status(statusid,'SN1', _3, _2, viewcount),
- $P2^{exp}$ is NOT contained in Q

θ: userid → userid 'SN1' → 'SN1'
name → name '2016-03-03' → '2016-03-03'
statusid → statusid
viewcount → _1

We should have viewcount → viewcount !

Query plan P3

- User query Q

q(userid, statusid, viewcount) ← user(userid, ‘SN1’, name), status(statusid, ‘SN1’, userid, '2016-03-03', viewcount)

- Query plan P3
  - P3 =

q(userid, statusid, viewcount) ← sn2update(userid,statusid, '2016-03-03’, viewcount),

  - $P3^{exp}$ =

q(userid, statusid, viewcount) ← status(statusid,’SN2’,userid, '2016-03-03’, viewcount),

- Problem: $P3^{exp}$ is not contained in Q
  there cannot be any substitution θ s.t.
  θ(‘SN1’) = ‘SN2’
  moreover
  there is no atom user(…) in $P3^{exp}$.

Running example

- LAV mappings

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

There is no (max.) contained rewiting for Q.

Updated running example

- LAV mappings

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,userid,creationdate,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

- User query Q

q(userid, statusid, viewcount) ← user(userid, 'SN1', name),
status(statusid, 'SN1', userid, '2016-03-03', viewcount)

- Maximally contained rewriting (only one)

q(userid, statusid, viewcount) ←sn1user(userid,name),
sn1viewcount(statusid,userid,'2016-03-03',viewcount)

Max contained query plans computation

- Naive algorithm (for CQ, not datalog) [LMSS95,RSU95]
- Bucket Algorithm [LRO96]
- Unification-join algorithm [Q96]
- Inverse rules [DG97,DGL00]
- Minicon [PH01]
- Inverse Minicon [K04]

Max contained query plans computation

- Naive algorithm (for CQ, not datalog) [LMSS95,RSU95]
- Bucket Algorithm [LRO96]
- Unification-join algorithm [Q96]
- Inverse rules [DG97,DGL00]
- Minicon [PH01]
- Inverse Minicon [K04]

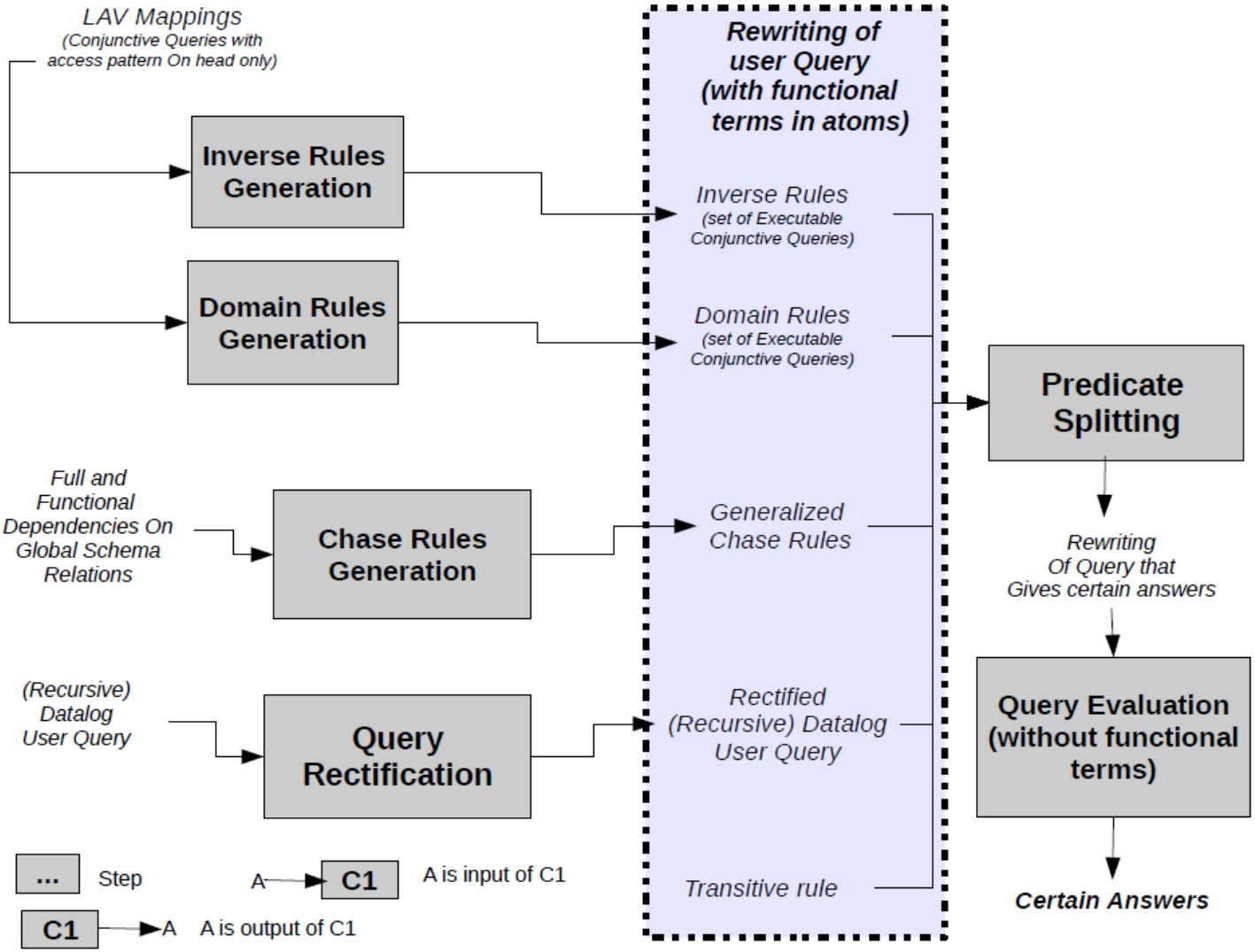
The only algorithm that handles at the same time

- **recursive queries**
- **access patterns (on head of mappings)**
- **full and functional dependencies**
- **PTIME complexity**

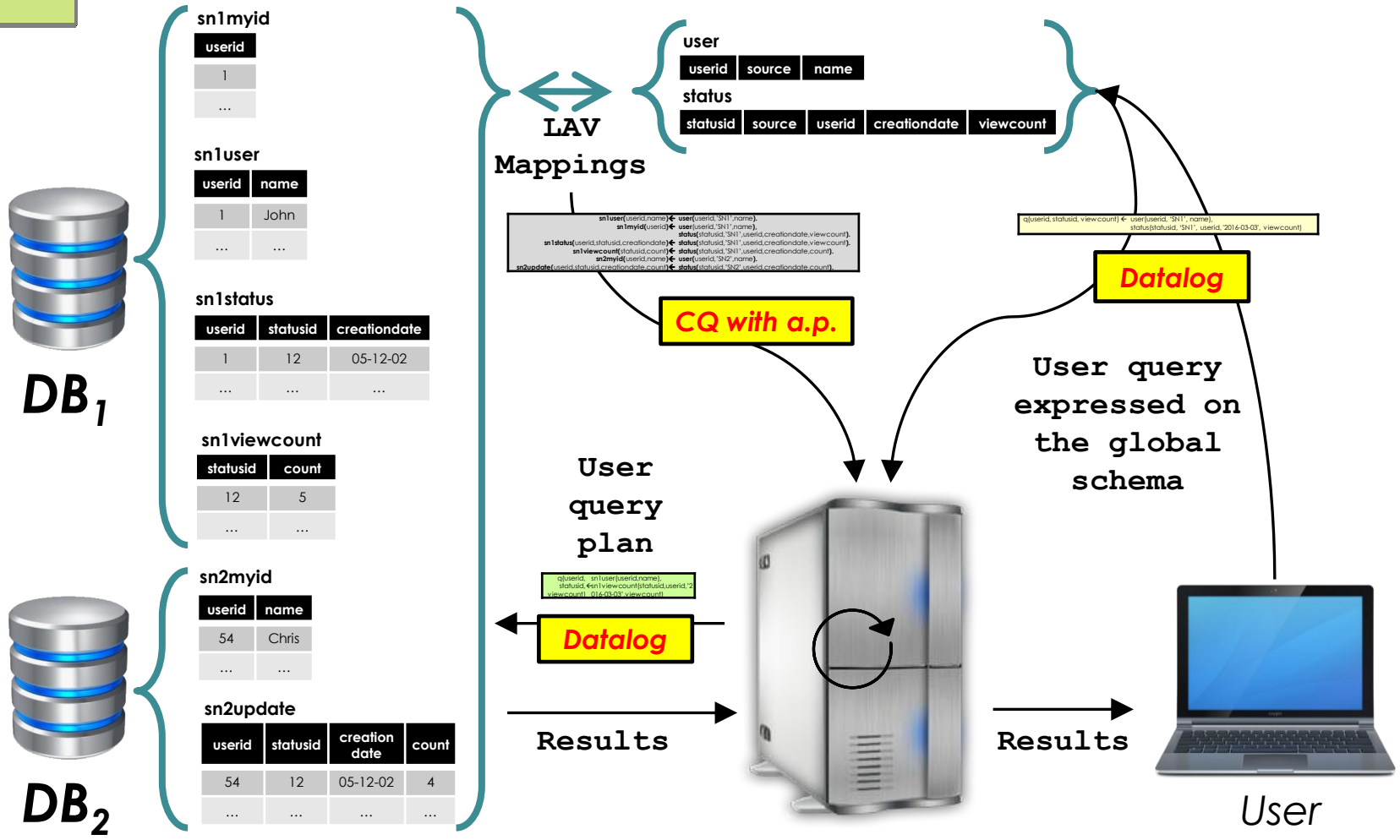
Inverse rules [DG97,DGL00]

- LAV mappings: CQ with access patterns on heads
- User query: datalog
- Maximally contained rewritings: datalog queries
- Generate maximally contained rewritings by adding to (rectified) query rules
  - Inverse rules: handle LAV mapping
  - Domain rules: handle access patterns
  - Generalized chase rules: handle dependencies
  - Transitive rules: handle the equality between variables
- Get rid of functional terms by predicate splitting
- Compute certain answers by query evaluation

Inverse rules [DG97,DGL00]



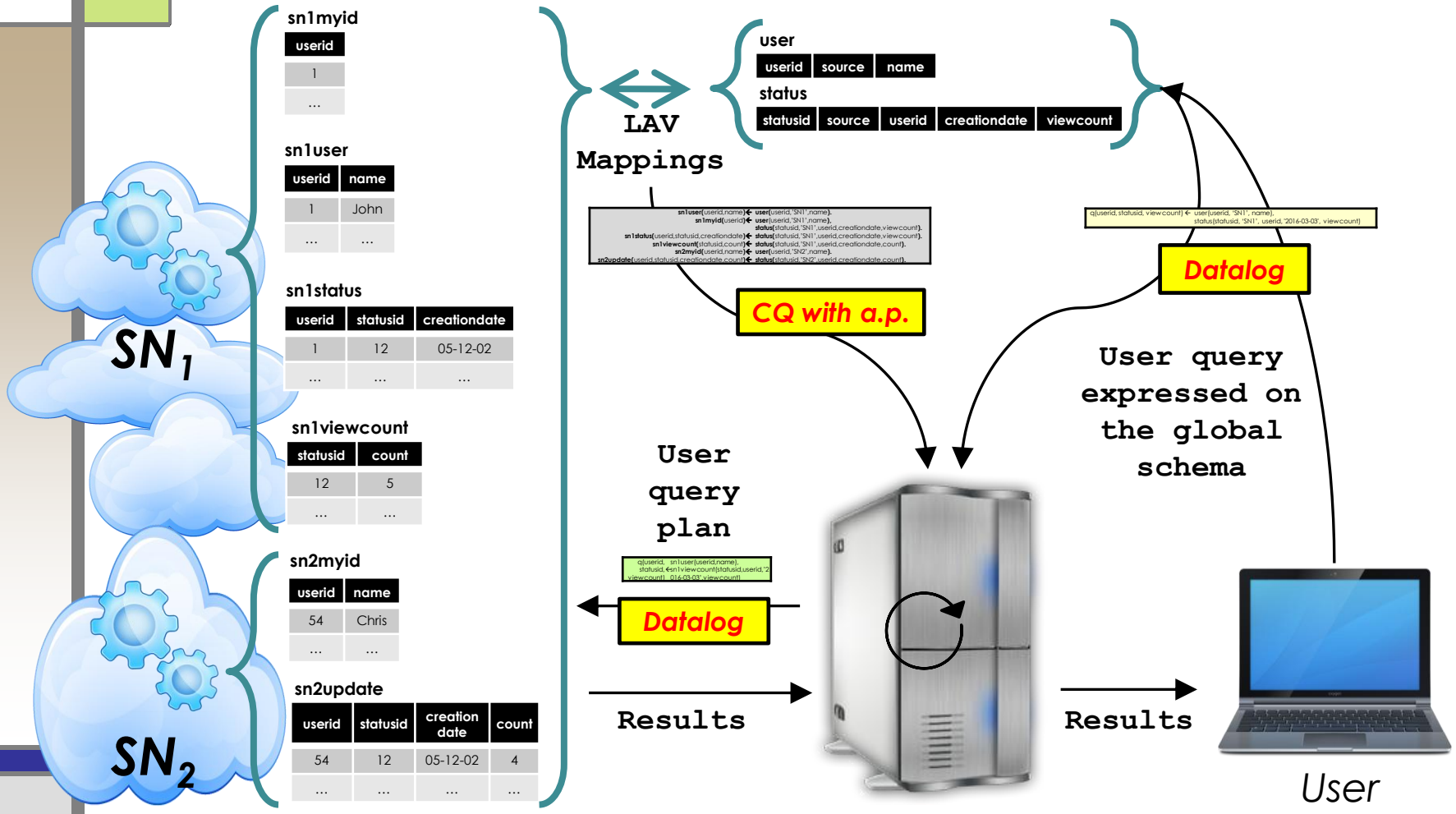
From DI to WS data extraction



Private data sources with public views

Mediator with a virtual global schema powered by the IR algo

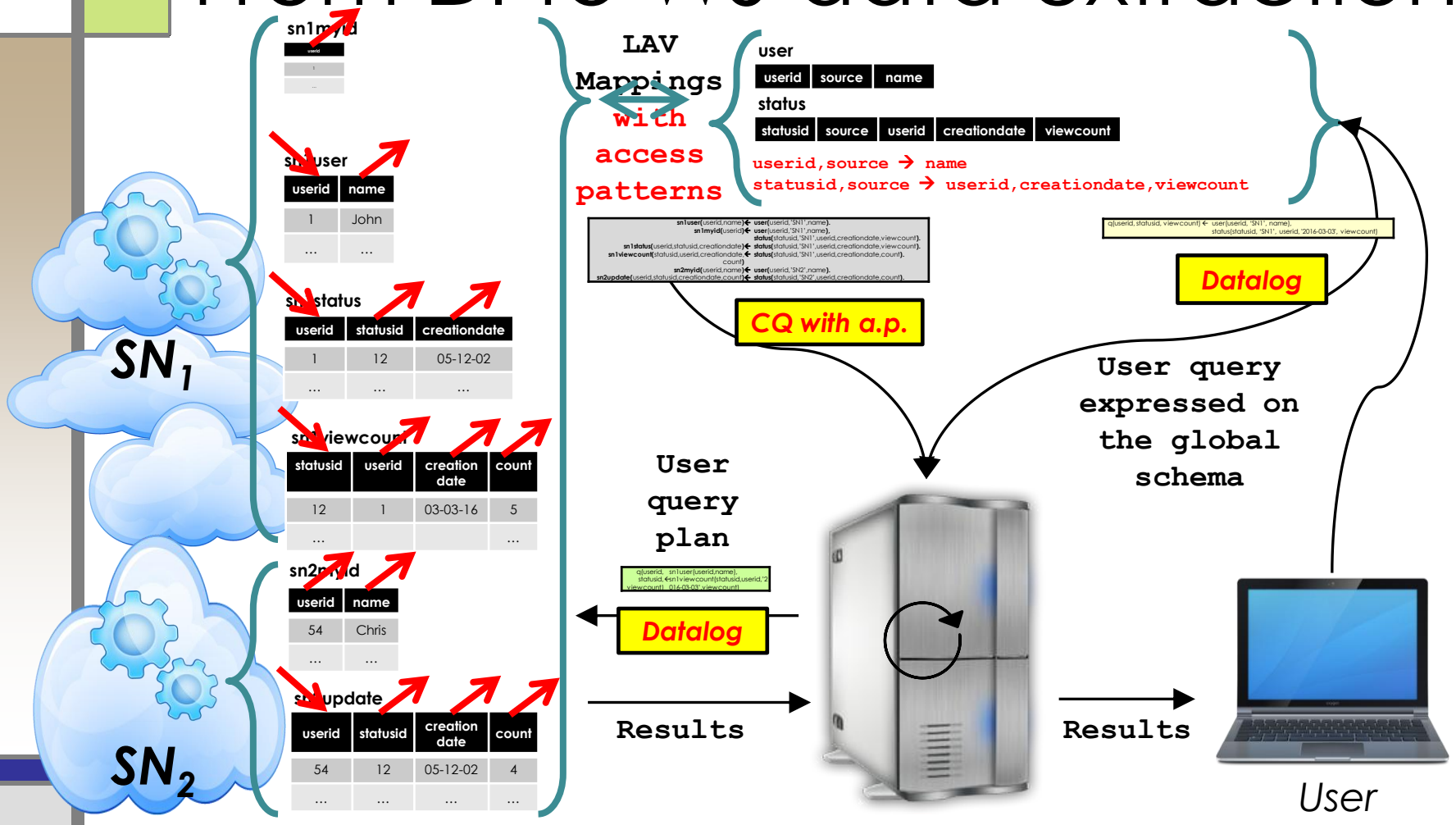
From DI to WS data extraction



Private data sources with public views

Mediator with a virtual global schema powered by the IR algo

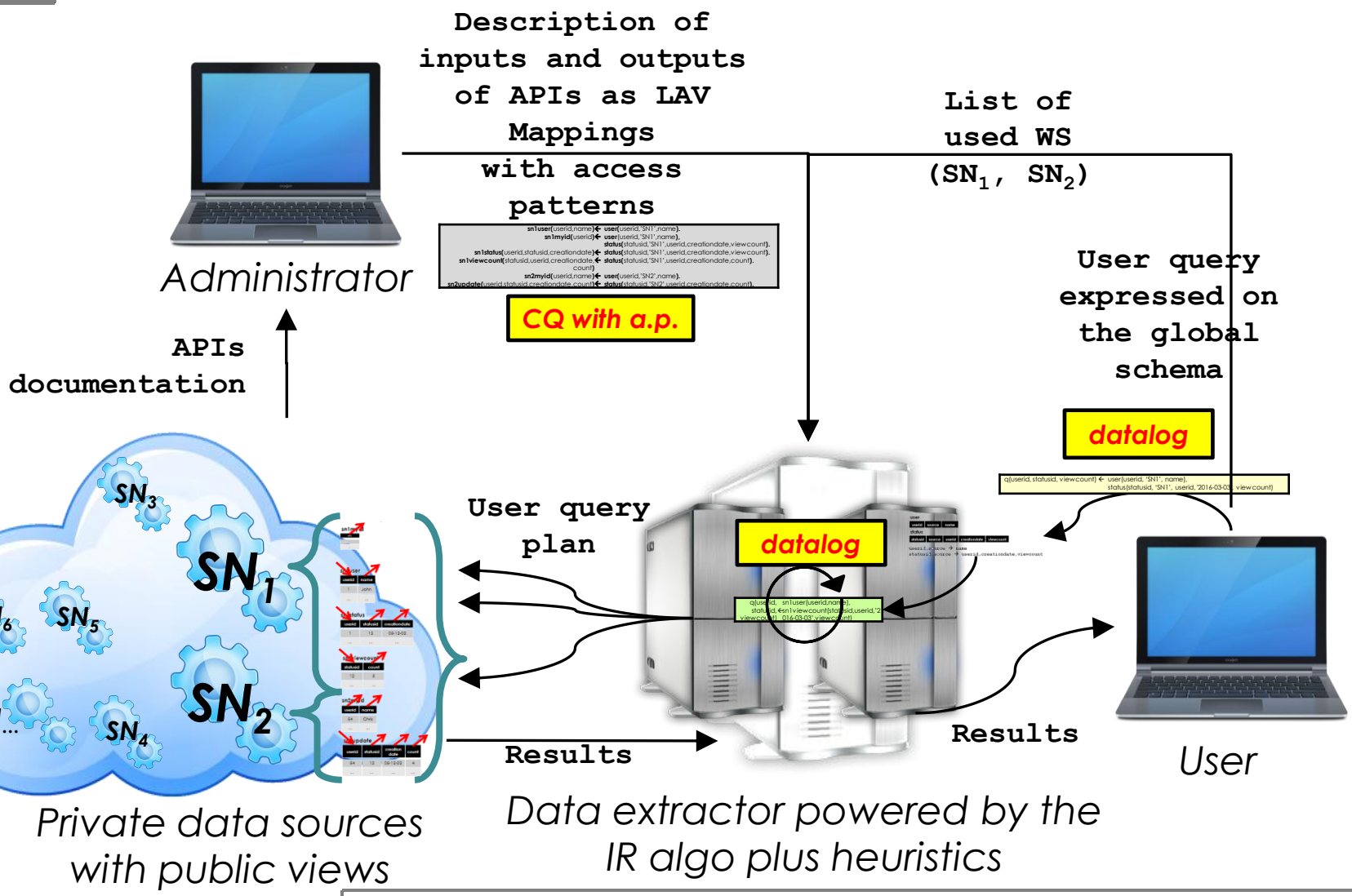
From DI to WS data extraction



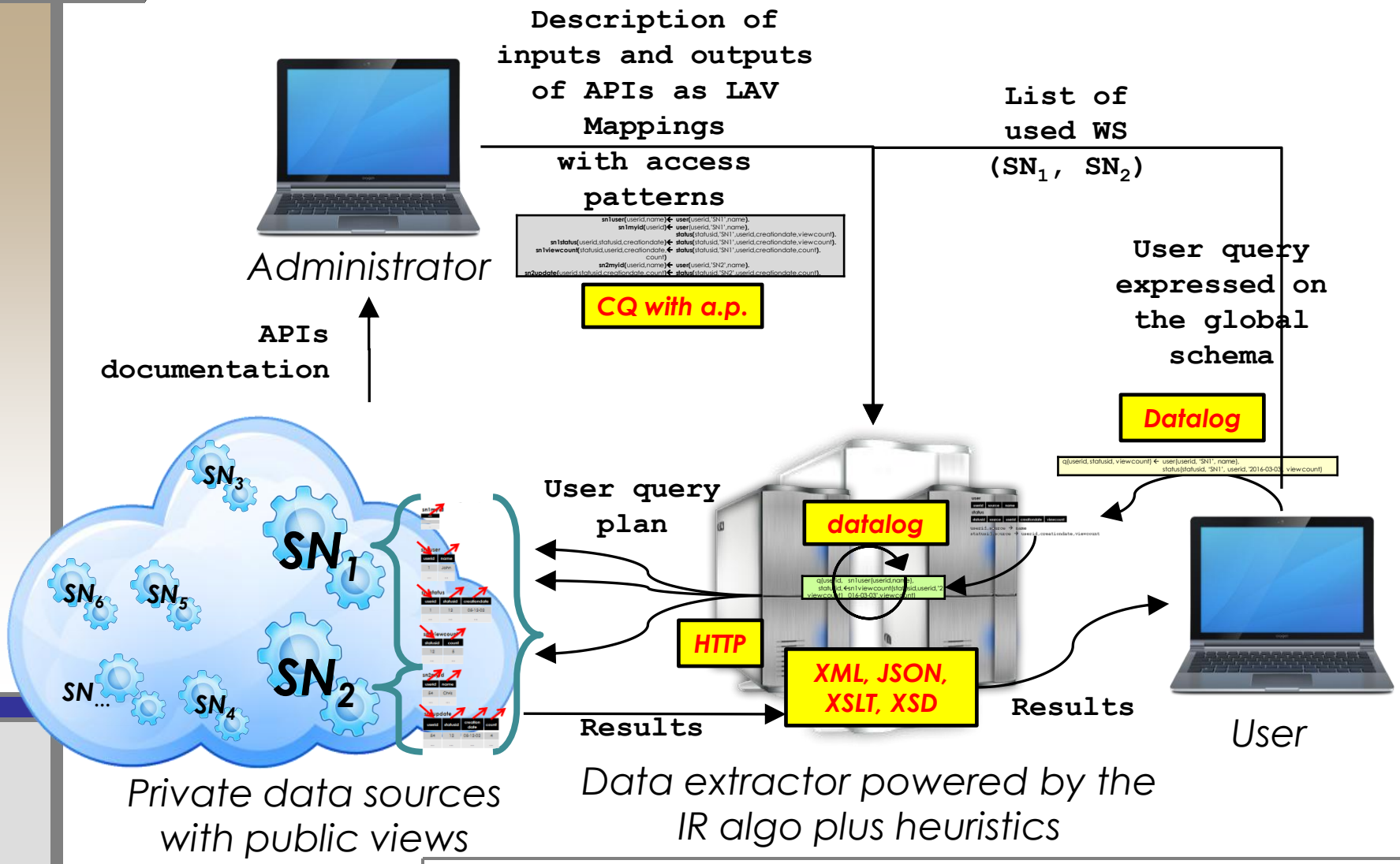
Private data sources with public views

Mediator with a virtual global schema powered by the IR algo

From DI to WS data extraction



From DI to WS data extraction



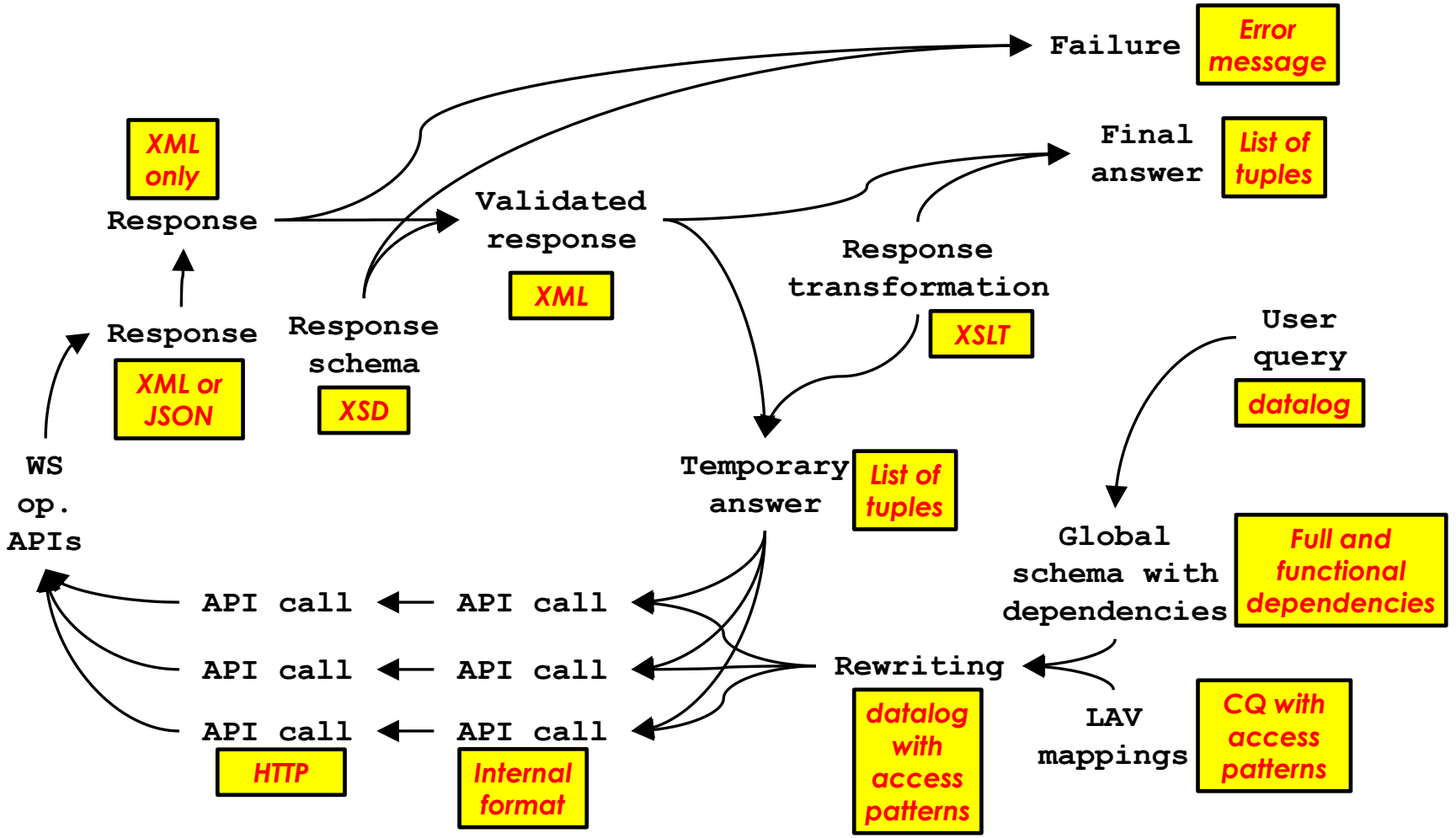
Declarative languages

- **CQ with access patterns**: express the LAV mappings that define WS operations in terms of the global schema
- **Datalog**:
  - express the user queries posed to the global schema
  - express the max. cont. rewritings, i.e. the way the user query is distributed over WS operations
- **XML / JSON**:
  - Send requests to WS operations
  - Get back the response of WS operations
- **XSD**: validate operation responses and detecting API changes
- **XSLT**: extract and transform desired information from operation response
- **HTTP**: call WS operations APIs

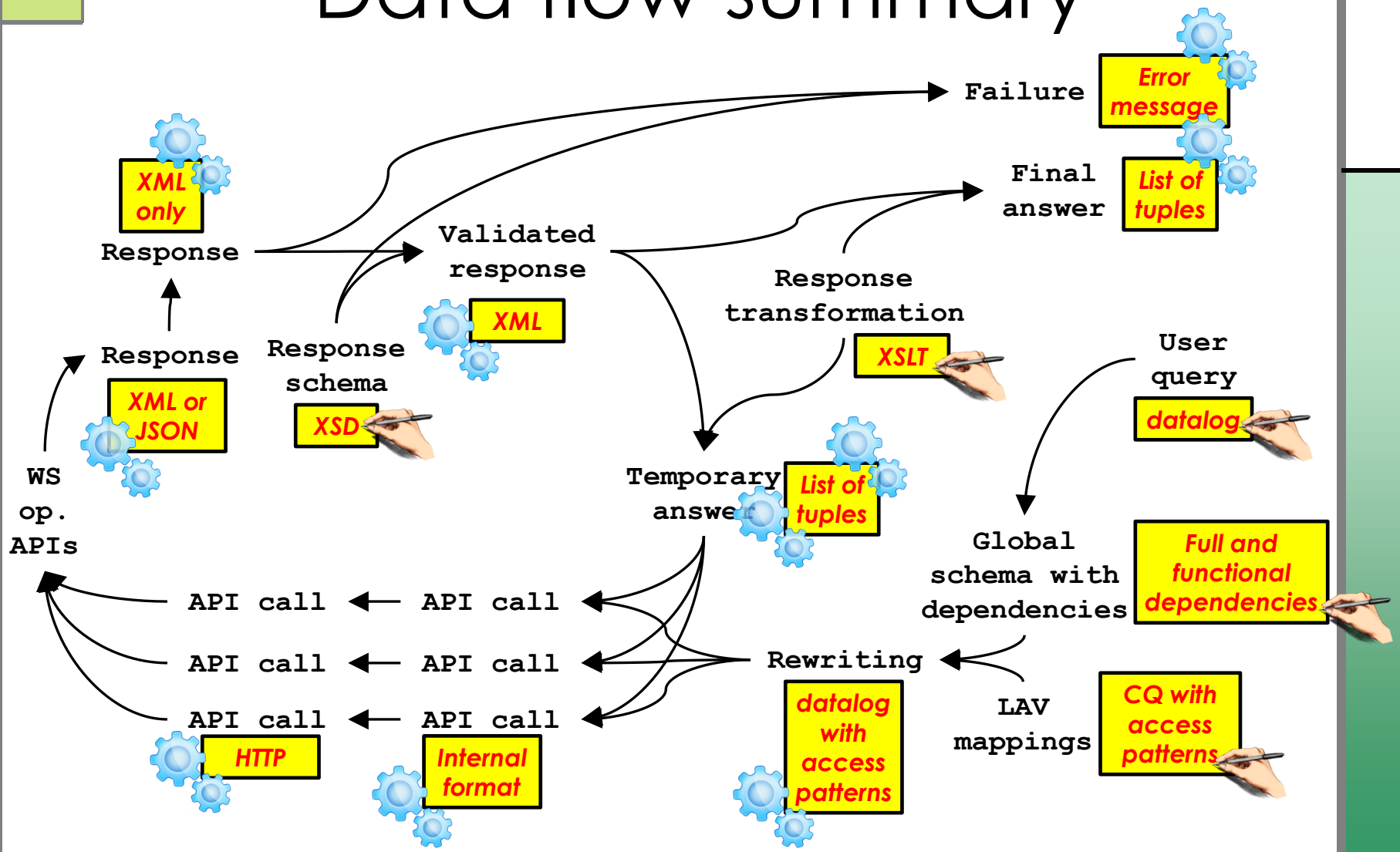
Handling incomplete information

- Certain answers semantics:
  - A bit restricted (cf examples) for real use
  - Incomplete query answers may still be interesting.
- Relaxation:
  - Handle functional terms in the IR algo
  - Allows to return incomplete but additional answers
- Proposal of a heuristics
- Proposal of a semantics for datalog with access patterns
- Work in progress

Data flow summary



Data flow summary



[Main scenarios]

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

Scenarios outline

- 1. Introducing the WS and WADL (sn1, sn2)
- 2. Defining the Global Schema Relation (user, status)
- 3. Defining the WS (Category, Provider, API, Operations)
- 4. Defining the Local Schema Relations (LAV Mapping)
- 5. Defining Records
- 6. Defining Performance Indicators
- 7. Defining the user (User's choice of WS, Records, Performance Indicators)
- 8. Obtaining the user record values, performance indicators

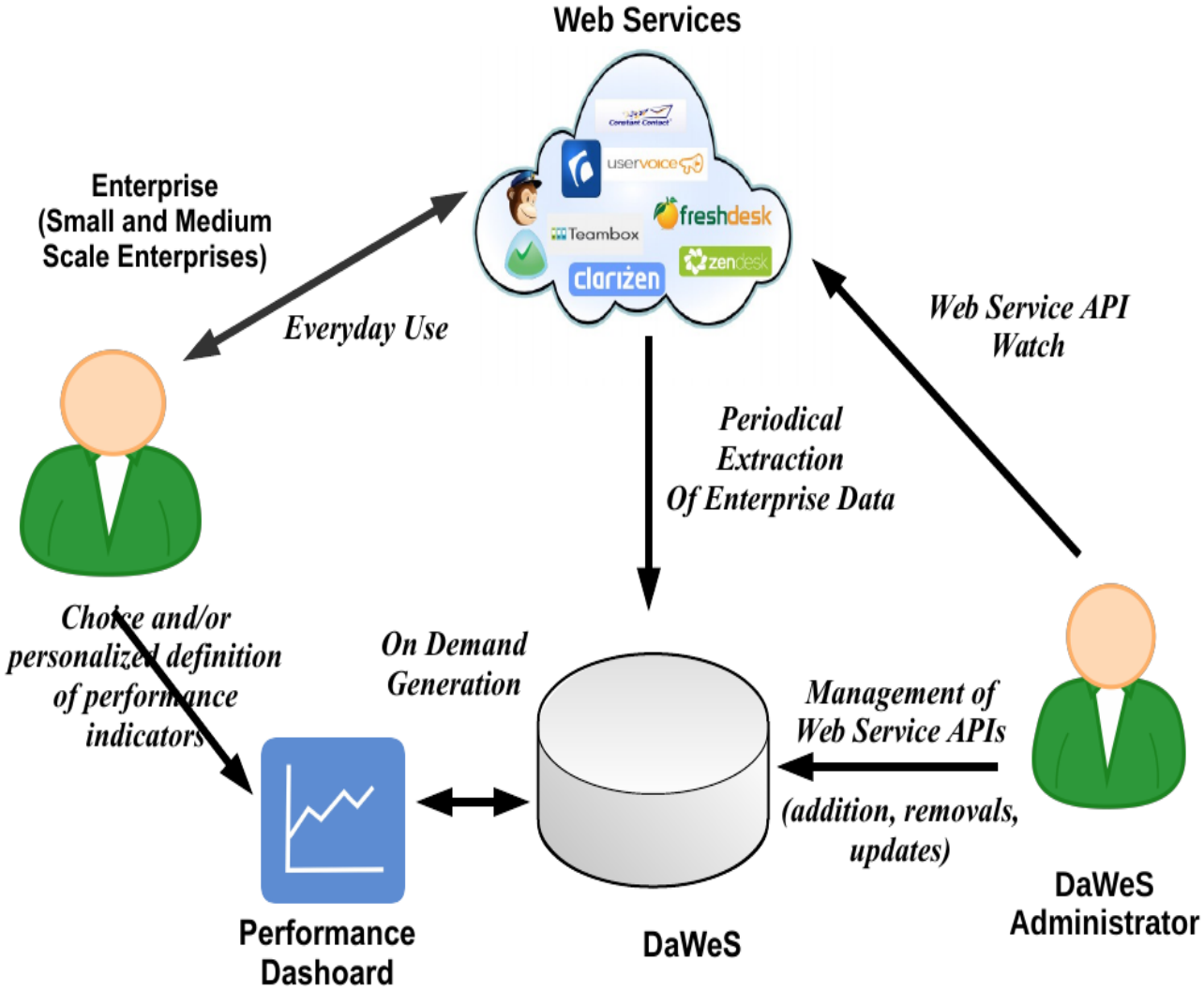
[Results and discussion]

Studied issue
Locks and Keys
Declarative data extraction
Data integration with WS
Main scenarios
Results and discussion

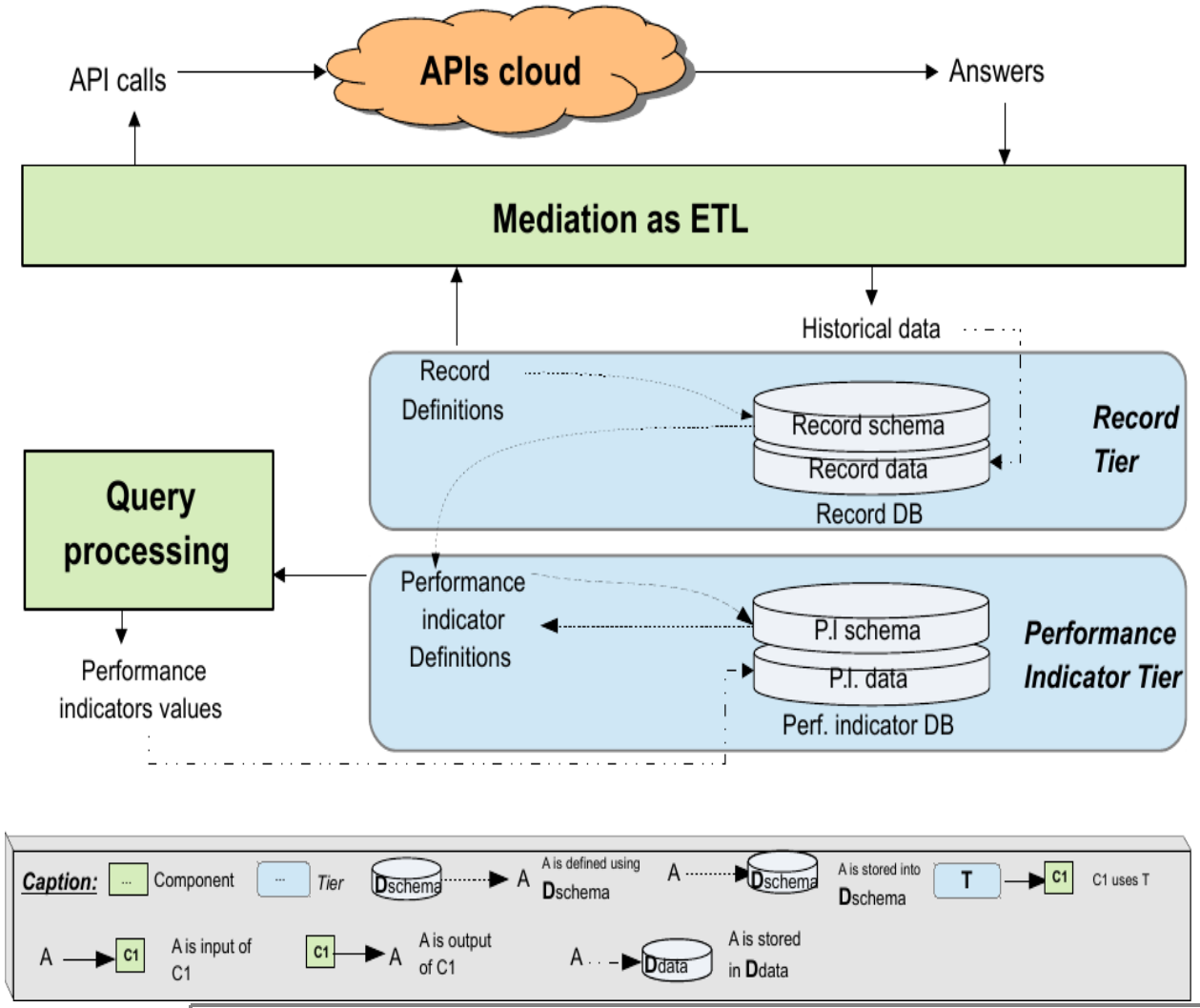
Application to DW

- Use of this declarative approach to WS data extraction to feed a datawarehouse (DW)
- « Mediation as ETL »
- Prototype : DaWeS
  « Data Warehouse
  fed with Web Services »

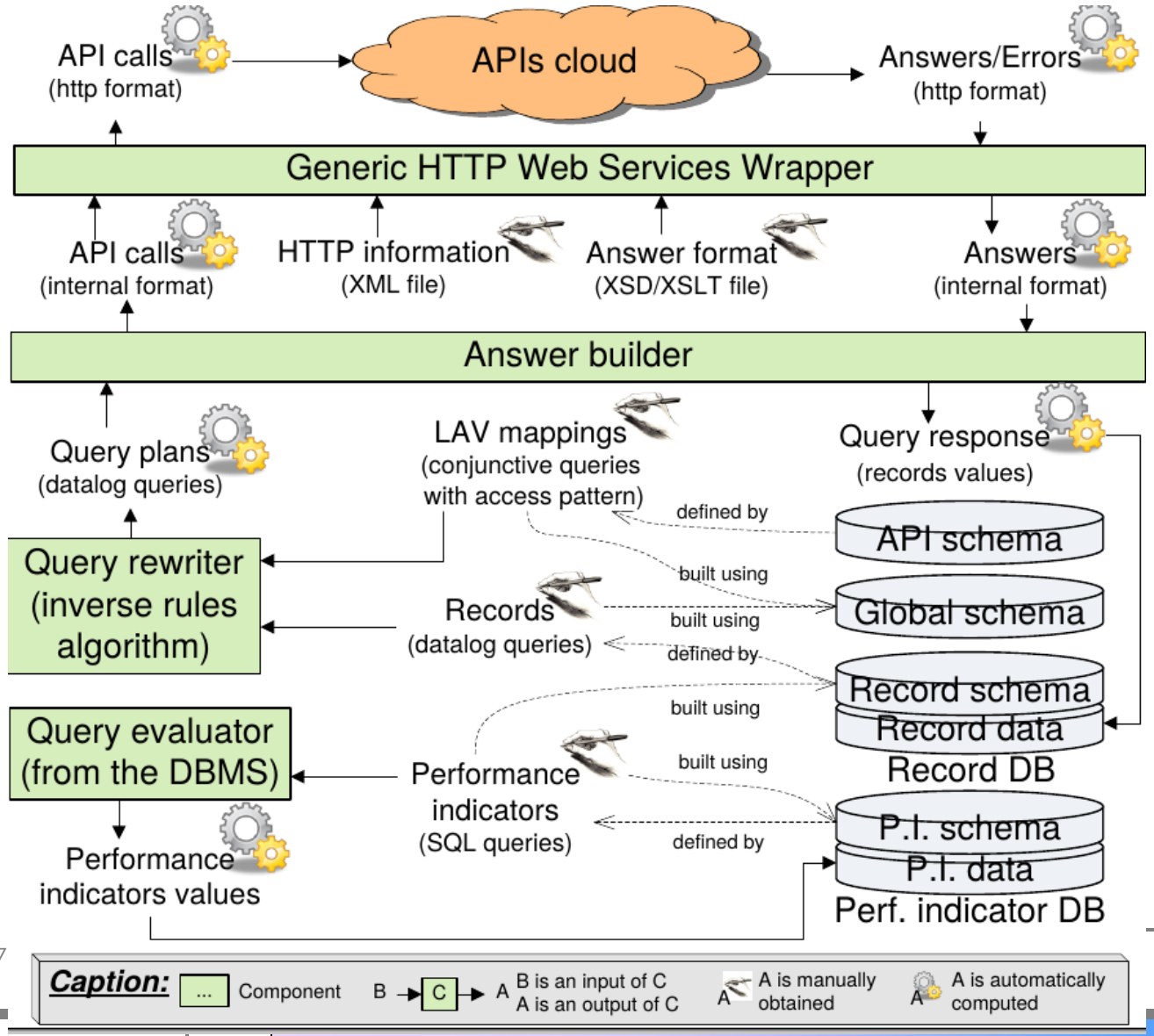
DaWeS overview



DaWeS architecture



DaWeS declarativeness



DaWeS experiments

- Number of domain of Web Services: 3
- Number of Web services considered : 12
- Number of API operations considered: 35
- Number of Global Schema Relations: 12
- Number of Test Organizations: 100
- Message Formats: 2
  XML, JSON
- Authentication Mechanisms: 2
  HTTP basic authentication, OAuth
- Number of Record Definitions: 17
- Number of Performance Indicator Queries: 20
- Operation details:
  - None, one or more input parameters
  - Pagination

Discussion

- Mediation: not recent domain
  About 20 years old
- Industrial development of mediators ?
  No one, up to our knowledge
- Main reasons (we suppose)
  - Computationally complex algorithms
  - Not easy to implement
  - Need to add extra features to be really useful
    - Access patterns
    - Incomplete information
    - Dependencies
    - Aggregation functions
    - More expressive view language
    - GAV, GLAV mappings

Perspectives

- Extra features management
  - Adding features ?
  - Already done:
    - One heuristics to handle incomplete info
    - Definition of the semantics of datalog with access patterns
  - To do:
    - Going on with the state-of-the-art [DLN07, YKC06, CM08, CCM09, …]
    - Complexity study of this heuristics (bound on the number of accesses)
    - Enhancing our heuristics to algorithm
    - Complexity study of this algorithm
- Concerning DaWeS
  - Optimizing the number of operation calls
  - Access patterns and errors
  - Automated error handling

Perspectives

- Industrial scale experimentations
  - Already done
    - 3 domains: project management, email marketing, support/helpdesk
    - 12 WS
    - 35 operations
  - To do: other domains, far more operations
- Broaden the approach
  - To ontological query answering of WS APIs
  - To other applications (than feeding a DW)

DaWeS future

- Looking for industrial partners
  - Other domains
  - Other applications
- Looking for academic partners
  - Towards a test platform for rewriting algorithms
    - Data integration extensions
    - Ontological query answering algorithms
    - …

Thank you.

[Additional slides]

Bibliography

[AD98] Abiteboul S. and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *Proc. 17th Annual ACM Symp. Principles of Databases (PODS '98)*, Seattle, Washington 1998, pp. 254–263.

[S93] Oded Shmueli. Equivalence of DATALOG Queries is Undecidable. J. Log. Program. 15(3) 231-241 (1993)

[CM77] A. K. Chandra, P. M. Merlin. Optimal implementation of conjunctive queries. In *Proc. ACM SIGACT Symp. on the Theory of Computing (STOC '77)*, pp. 77–90, 1977.

[FGMP05] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: semantics and query answering. Theor. Comput. Sci., 336(1):89–124.

[ABFL13] Arenas, M., Barceló, P., Fagin, R., and Libkin, L. (2013). Solutions and query rewriting in data exchange. Inf. Comput., 228:28–61.

[DLN07] Deutsch, A., Ludäscher, B., and Nash, A. (2007). Rewriting queries using views with access patterns under integrity constraints. Theor. Comput. Sci., 371(3):200–226.

[YKC06] Yang, G., Kifer, M., and Chaudhri, V. K. (2006). Efficiently ordering subgoals with access constraints. In PODS, pages 183–192.

[CM08] Calì, A. and Martinenghi, D. (2008). Querying data under access limitations. In ICDE, pages 50–59.

[CCM09] Calì, A., Calvanese, D., and Martinenghi, D. (2009a). Dynamic query optimization under access limitations and dependencies. J. UCS, 15(1):33–62.

Bibliography

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, Divesh Srivastava: Answering Queries Using Views. PODS 1995: 95-104

[RSU95] Anand Rajaraman, Yehoshua Sagiv, Jerey D. Ullman: Answering Queries Using Templates with Binding Patterns. PODS 1995: 105-112

[LRO96] Alon Y. Levy, Anand Rajaraman, Joann J. Ordille: Querying Heterogeneous Information Sources Using Source Descriptions VLDB 1996: 251-262

[LRO96] Alon Y. Levy, Anand Rajaraman, Joann J. Ordille: Query-Answering Algorithms for Information Agents. AAAI/IAAI, Vol. 1 1996: 40-47

[Q96] Xiaolei Qian: Query Folding. ICDE 1996:48-55

[DR97] Oliver M. Duschka, Michael R. Genesereth: Answering Recursive Queries Using Views. PODS 1997:109-116

[DRL00] Oliver M. Duschka, Michael R. Genesereth, Alon Y. Levy: Recursive query plans for data integration. The Journal of Logic Programming 43 (2000):49-73

[PH01] Rachel Pottinger, Alon Y. Halevy: MiniCon: A scalable algorithm for answering queries using views. VLDB J. (VLDB) 10(2-3):182-198 (2001)

[K04] Christoph Koch: Query rewriting with symmetric constraints. AI Commun. (AICOM) 17(2):41-55 (2004)

Characteristics of some real WS

A study of some real WS in the field of project management

Web Service	Basecamp	Liquid Planner	Teamwork	Zoho Projects	Wrike	Teambox
1. API Description	HTML page	HTML page	HTML page	HTML page	HTML page	HTML page
2. Conformance to REST	REST like	REST like	REST like	Not REST	Not REST	REST like
3. Version	v1	3.0.0	N.A.	N.A.	v2	1
4. Authentication	Basic HTTP, OAuth 2	Basic HTTP	Basic HTTP	Basic HTTP	OAuth 1.0	OAuth 2.0
5. Resources Involved	Project,Todo List, Todo	Project, Task	Project, Task List, Task	Project, Task List, Task	Task	Project, Task
6. Message Formats	JSON	JSON	XML, JSON	XML, JSON	XML, JSON	JSON
7. Service Level Agreement	Max 500 requests /10s from same IP address for same account	Max 30 requests /15s for same account	Max 120 requests /1min	Error code:6403 on exceeding the limit	N.A	N.A.
8. HTTP Resource Access	GET	GET	GET	POST	POST	GET
9. Data Types (dt)	Enumerated dt (Project and Todo Status), Date	Enumerated dt (Project and Task Status), Date	Enumerated dt (Project and Task Status), Date	Enumerated dt (Project and Task Status), Date	Enumerated dt (Task Status), Date	Enumerated dt (Project and Task Status), Date
10. Dynamic nature of the resources	Yes (Project and Todo Status)	Yes (Project and Task Status)	Yes (Project and Task Status)	Yes(Project and Task Status)	Yes (Task Status)	Yes (Project and Task Status)
11. Operation Invocation Sequence Required	Yes	No	Yes	Yes	Yes	Yes
12. Pagination	No	No	No	Yes	Yes	No

A study of some real WS in the field of email marketing

Web Service	Mailchimp	Campaign Monitor	iContact	Constant Contact	AWeber
1. API Description	HTML page	HTML page	HTML page	HTML page	HTML page
2. Conformance to REST	Not REST	REST like	REST like	REST	REST
3. Version	1.3	v3	2.2	N.A.	1.0
4. Authentication	Basic HTTP	Basic HTTP, OAuth 2	Basic HTTP (with Sandbox)	OAuth 2.0	OAuth 1.0
5. Resources Involved	Campaign, Campaign Statistics	Campaign, Campaign Statistics	Campaign, Campaign Statistics	Campaign, Campaign Statistics	Campaign, Campaign Statistics
6. Message Formats	XML, JSON, PHP, Lolcode	XML, JSON	XML, JSON	XML	JSON
7. Service Level Agreement	N.A.	N.A.	75,000 requests /24h, with a max of 10,000 requests /1h	N.A	60 requests per minute
8. HTTP Resource Access	GET	GET	GET	GET	GET
9. Data Types (dt)	Enumerated Data types (Campaign Status), Date	Enumerated Data types (Campaign Status), Date	Enumerated Data types (Campaign Status), Date	Enumerated Data types (Campaign Status), Date	Enumerated Data types (Campaign Status), Date
10. Dynamic nature of the resources	Yes (Campaign Status)	Yes (Campaign Status)	Yes (Campaign Status)	Yes (Campaign Status)	Yes (Campaign Status)
11. Operation Invocation Sequence Required	Yes	Yes	No	Yes	Yes
12. Pagination	Yes	No	No	Yes	Yes

A study of some real WS in the field of support/ helpdesk

Web Service	Zendesk	Desk	Zoho Support	Uservoice	Freshdesk	Kayako
1. API Description	HTML page	HTML page	HTML page	HTML page	HTML page	HTML page
2. Conformance to REST	REST like	REST	Not REST	REST like	REST like	REST
3. Version	v1	v2	N.A.	v1	N.A.	N.A.
4. Authentication	Basic HTTP	Basic HTTP, OAuth 1.0a	Basic HTTP	OAuth 1.0	Basic HTTP	Basic HTTP
5. Resources Involved	Forum, Topic, Ticket	Case	Task	Forum, Topic, Ticket	Forum, Topic, Ticket	Ticket, Topic
6. Message Formats	XML, JSON	JSON	XML, JSON	XML, JSON	JSON	XML
7. Service Level Agreement	Limit exists (but unknown)	60 requests per minute	250 calls /day /org (Free)	N.A.	N.A.	N.A.
8. HTTP Resource Access	GET	GET	GET	GET	GET	GET
9. Data Types (dt)	Enumerated dt (Ticket Status), Date	Enumerated dt (Case Status), Date	Enumerated dt (Task Status), Date	Enumerated dt (Ticket Status), Date	Enumerated dt (Ticket Status), Date	Enumerated dt (Ticket Status), Date
10. Dynamic resources	Yes (Ticket Status)	Yes (Case Status)	Yes (Task Status)	Yes (Ticket Status)	Yes (Ticket Status)	Yes (Ticket Status)
11. Operation Invocation Sequence Required	Yes	Yes	Yes	Yes	Yes	Yes
12. Pagination	Yes	Yes	Yes	Yes	No	Yes

JSON schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "http://jsonschema.net",
    "type": "object",
    "properties": {
        "updates": {
            "id": "http://jsonschema.net/updates",
            "type": "array",
            "items": {
                "id": "http://jsonschema.net/updates/1",
                "type": "object",
                "properties": {
                    "identifier": {
                        "id": "http://jsonschema.net/updates/1/identifier",
                        "type": "integer"
                    },
                    "update": {
                        "id": "http://jsonschema.net/updates/1/update",
                        "type": "string"
                    },
                    "date": {
                        "id": "http://jsonschema.net/updates/1/date",
```

Computing certain answers complexity

Complexity [AD98]

Let $\mathbf{IS} = ((G, I_G), (S, I_S), M = (s : I_G \rightarrow s(I_G)))$ be a LAV integration system under OWA. Let Q be a query on G. Let's call "views" the predicates defined in S by s.

- Languages for queries and views may be:
  - CQ and CQ with inequality ($CQ^{\neq}$)
  - Positive query (PQ): nr-datalog program with a particular query predicate
  - Positive query + inequality ($PQ^{\neq}$)
  - Datalog queries ($datalog$)
  - First order queries (FO)
- Complexity may be:
  - Data complexity: function of the size of I_S
  - Query complexity: function of the sizes of the view definition s and the query Q
  - Combined complexity: function of all (both the sizes of the view definition s and the query Q, and the size of I_S

Complexity [AD98]

- For $s \in PQ^{\neq}$ and $Q \in datalog^{\neq}$, the problem of determining, given I_S, whether a tuple is a certain answer under OWA or CWA is in co-NP (combined complexity).
- Data complexity of the problem of computing certain answers under OWA.

	query Q				
views s	CQ	$CQ^{\neq}$	PQ	$datalog$	FO
CQ	PTIME	Co-NP	PTIME	PTIME	und
$CQ^{\neq}$	PTIME	Co-NP	PTIME	PTIME	und
PQ	Co-NP	Co-NP	Co-NP	Co-NP	und
$datalog$	Co-NP	und	Co-NP	und	und
FO	und	und	und	und	und

"und" means undecidable

Complexity [AD98]

Let

- **IS** $= ((G, I_G), (S, I_S), M = (s : I_G \rightarrow s(I_G)))$ be a LAV integration system under OWA
- $\mathcal{L}_1, \mathcal{L}_2 \in \{CQ, CQ^{\neq}, PQ, Datalog, FO\}$
- $s \in \mathcal{L}_1$ and $Q \in \mathcal{L}_2$

Then

1. (View pb) Computing $cert(Q, s, I_S)$ is decidable **iff** (Containment pb) the containment problem of a query in $\mathcal{L}_1$ in a query in $\mathcal{L}_2$ is decidable

2. If the two problems are decidable: the combined complexity of the view pb = the query complexity of the containment pb

   ($\Rightarrow$ The data complexity of the problem of computing certain answers under OWA is **at most** the query complexity of the query containment problem)

The bucket algo example 1

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

user(userid, 'SN1', name)	**status(statusid, 'SN1', userid, '2016-03-03', viewcount)**
sn1user(userid,name**)**	**sn1myid(**userid**)**
sn1myid(userid**)**	**sn1status(**userid,statusid,'2016-03-03'**)**
	sn1viewcount(statusid,viewcount**)**

Candidate query plans
q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)**
q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid, '2016-03-03'**)**
q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,viewcount**)**
q(userid, statusid, viewcount**)** ← **sn1myid(**userid**)**~~**, sn1myid(**userid**)**~~
q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid, '2016-03-03'**)**
q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,viewcount**)**

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

Candidate query plans

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)**

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid, '2016-03-03'**)**

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,viewcount**)**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), ~~sn1myid(userid)~~**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid, '2016-03-03'**)**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,viewcount**)**

Candidate expanded query plans (bodys only)

user(userid,'SN1',name**), user**(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid2,creationdate2,viewcount**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid2,creationdate3,viewcount**)**

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

Candidate query plans

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid, '2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,viewcount**)**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid, '2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,viewcount**)**

Candidate expanded query plans (bodys only)

user(userid,'SN1',name**), user**(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid2,creationdate2,viewcount**)** — Not contained in Q

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid2,creationdate3,viewcount**)** — Not contained in Q

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

Candidate query plans

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)** Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid,'2016-03-03'**)** Not safe

q(userid, statusi nt**)**

q(userid, statusi Not safe

q(userid, statusi Not safe

q(userid, statusi

There is no contained rewriting for this query.

Candidate expa

user(userid,'SN1', date2,viewcount2**)**

user(userid,'SN1',

user(userid,'SN1',name**), status(**statusid,'SN1',userid2,creationdate2,viewcount**)** Not contained in Q

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid,'2016-03-03',viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid2,creationdate3,viewcount**)** Not contained in Q

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

user(userid, 'SN1', name)	**status(statusid, 'SN1', userid, '2016-03-03', viewcount)**
sn1user(userid,name**)**	**sn1myid(**userid**)**
sn1myid(userid**)**	**sn1status(**userid,statusid,'2016-03-03'**)**
	sn1viewcount(statusid,viewcount**)**

Extra candidate query plan

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**),**
sn1status(userid,statusid, '2016-03-03'**),**
sn1viewcount(statusid,viewcount**)**

Expanded extra candidate query plan

q(userid, statusid, viewcount**)** ← **user(**userid,'SN1',name**),**
status(statusid,'SN1',userid, '2016-03-03',viewcount2**)**
status(statusid,'SN1',userid2,creationdate2,viewcount**)**

Even if we had the FD statusid → userid, the extra candidate is not contained in Q.

The bucket algo example 2

Same mappings as example 1 but simplified query.

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(statusid, viewcount) ← user(userid, 'SN1', name),
status(statusid, 'SN1', userid, creationdate, viewcount)

user(userid, 'SN1', name)	**status(statusid, 'SN1', userid, creationdate, viewcount)**
sn1user(userid,name**)**	**sn1myid(**userid**)**
sn1myid(userid**)**	**sn1status(**userid,statusid, creationdate**)**
	sn1viewcount(statusid,viewcount**)**

Candidate query plans
q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)**
q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid, creationdate**)**
q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,viewcount**)**
q(statusid, viewcount**)** ← **sn1myid(**userid**), ~~sn1myid(userid)~~**
q(statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid, creationdate**)**
q(statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,viewcount**)**

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(statusid, viewcount) ← user(userid, 'SN1', name),
status(statusid, 'SN1', userid, creationdate, viewcount)

Candidate query plans

q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)** — Not safe

q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid, creationdate**)** — Not safe

q(statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,viewcount**)**

q(statusid, viewcount**)** ← **sn1myid(**userid**), sn1myid(**userid**)** — Not safe

q(statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid, creationdate**)** — Not safe

q(statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,viewcount**)**

Candidate expanded query plans (bodys only)

user(userid,'SN1',name**), user**(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, creationdate,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid2,creationdate2,viewcount**)** — Not contained in Q

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid, creationdate,viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid2,creationdate3,viewcount**)** — Not contained in Q

The bucket algo example 3

One different mapping from example 1 but same query (as in example 1).

sn1user(userid,name**)**←	**user(**userid,'SN1',name**).**
sn1myid(userid**)**←	**user**(userid,'SN1',name**),** **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**←	**status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,userid,creationdate,count**)**←	**status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**←	**user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**←	**status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

user(userid, 'SN1', name)	**status(statusid, 'SN1', userid, '2016-03-03', viewcount)**
sn1user(userid,name**)**	**sn1myid(**userid**)**
sn1myid(userid**)**	**sn1status(**userid,statusid,'2016-03-03'**)**
	sn1viewcount(statusid,userid, '2016-03-03',viewcount**)**

Candidate query plans

- **q(**userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)**
- **q(**userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid,'2016-03-03'**)**
- **q(**userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**
- **q(**userid, statusid, viewcount**)** ← **sn1myid(**userid**), ~~sn1myid(~~~~userid~~**)**~~
- **q(**userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid,'2016-03-03'**)**
- **q(**userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,userid,creationdate,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

Candidate query plans

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid,'2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid,'2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**

Candidate expanded query plans (bodys only)

user(userid,'SN1',name**), user**(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount**)** — Contained in Q

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1', userid, '2016-03-03',viewcount**)** — Contained in Q

sn1user(userid,name**)**← **user(**userid,'SN1',name**).**
sn1myid(userid**)**← **user**(userid,'SN1',name**),**
status(statusid,'SN1',userid,creationdate,viewcount**).**
sn1status(userid,statusid,creationdate**)**← **status(**statusid,'SN1',userid,creationdate,viewcount**).**
sn1viewcount(statusid,userid,creationdate,count**)**← **status(**statusid,'SN1',userid,creationdate,count**).**
sn2myid(userid,name**)**← **user(**userid,'SN2',name**).**
sn2update(userid,statusid,creationdate,count**)**← **status(**statusid,'SN2',userid,creationdate,count**).**

q(userid, statusid, viewcount) ← user(userid, 'SN1', name), status(statusid, 'SN1', userid, '2016-03-03', viewcount)

Candidate query plans

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1status(**userid,statusid,'2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1user(**userid,name**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1status(**userid,statusid,'2016-03-03'**)** — Not safe

q(userid, statusid, viewcount**)** ← **sn1myid(**userid**), sn1viewcount(**statusid,userid,'2016-03-03',viewcount**)**

Candidate expanded query plans (bodys only)

user(userid,'SN1',name**), user**(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name**), status(**statusid,'SN1',userid, '2016-03-03',viewcount**)** — Max. contained in Q

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**)**

user(userid,'SN1',name2**), status(**statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1',userid, '2016-03-03',viewcount2**)**

user(userid,'SN1',name2**),**
status(statusid2,'SN1',userid,creationdate2,viewcount2**), status(**statusid,'SN1', userid, '2016-03-03',viewcount**)** — Not maximally contained

Inverse rules algo.

Inverse rules algo. overview

Algorithm to build maximally contained rewritings (i.e. maximally contained datalog query plans)
Inputs: $Q \in datalog$ and $s \in \mathcal{CQ}$
Output: the maximally contained query plan $\mathcal{P}$ (i.e. $\mathcal{P} \in$ datalog) in Q wrt s

1. Building a logic program (datalog program with functions) with views as only EDB predicates
   a. remove from Q rules containing EDB predicates that do not appear in any view definition
   b. add inverse view definitions
2. Transforming the logic program into a datalog program (without function)
3. Optimizing
   a. remove unnecessary parts
   b. reduce the size of the obtained program

Inverse rules algo. Step 1a

1a. Building a logic program (datalog program with functions) with views as only EDB predicates

Running example

- Query to rewrite $\mathcal{P}$:
  $q(X, Y) :- edge(X, Y)$
  $q(X, Y) :- edge(X, Z), q(Z, Y)$
- Views s:
  $v(X, Y) :- edge(X, Z), edge(Z, Y)$
- Logic program $(\mathcal{P}^-, s^{-1})$:
  $q(X, Y) :- edge(X, Y)$
  $q(X, Y) :- edge(X, Z), q(Z, Y)$
  $edge(f(X, Y), Y) :- v(X, Y)$
  $edge(X, f(X, Y)) :- v(X, Y)$

Inverse rules algo. Step 1a

The obtained logic program is $(\mathcal{P}^-, s^{-1})$:

$q(X, Y) :- edge(X, Y)$

$q(X, Y) :- edge(X, Z), q(Z, Y)$

$edge(f(X, Y), Y) :- v(X, Y)$

$edge(X, f(X, Y)) :- v(X, Y)$

$(\mathcal{P}^-, s^{-1})$ produces answers with and without function symbols (f for the running example).

Let's suppose that $(\mathcal{P}^-, s^{-1})\downarrow$ is a datalog program that computes the answers of $(\mathcal{P}^-, s^{-1})$ with no function symbol.

Inverse rules algo. Step 1b

1b. Inversing view definition $v(X_1, \ldots, X_m)$:

- distinguished variables $X_1, \ldots, X_m$ remain unchanged
- each existential variable must be replaced by a term $f(X_1, \ldots, X_m)$ where f is a new function symbol for each existential var.

Another example

- Views s:

  $v_1(X, Y) :- edge(X, Z), edge(Z, W), edge(W, Y)$

  $v_2(X) :- edge(X, Z)$

- Inversed views s^{-1}

  $edge(X, f_1(X, Y)) :- v_1(X, Y)$

  $edge(f_1(X, Y), f_2(X, Y)) :- v_1(X, Y)$

  $edge(f_2(X, Y), Y) :- v_1(X, Y)$

  $edge(X, f_3(X)) :- v_2(X)$

Inverse rules algo. Step 2

2. Eliminating function symbols

In a bottom-up evaluation like process:

2.a functions terms in s^{-1} are replaced by the list of their variables and the associated predicate is replaced by an annotated one

Running example:

$edge(X, f(X, Y)) :- v(X, Y)$

is replaced by

$edge^{<*,f(*,*)>}(X, X, Y) :- v(X, Y)$

2.b new annotated predicates replace original ones only in the rules of $(\mathcal{P}^-, s^{-1})$

Inverse rules algo. Example

Beginning $(\mathcal{P}^-, s^{-1})$:

(1) $q(X, Y) : -edge(X, Y)$

(2) $q(X, Y) : -edge(X, Z), q(Z, Y)$

(3) $edge(f(X, Y), Y) : -v(X, Y)$

(4) $edge(X, f(X, Y)) : -v(X, Y)$

Result of 2.a 2 rules have been generated:

(3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$

(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$

after elimination of function symbol f.

Inverse rules algo. Example

$(\mathcal{P}^-, s^{-1})$ (1) $q(X, Y) : -edge(X, Y)$

(2) $q(X, Y) : -edge(X, Z), q(Z, Y)$

(3) $edge(f(X, Y), Y) : -v(X, Y)$

(4) $edge(X, f(X, Y)) : -v(X, Y)$

Result of 2.a (3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$

(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$

Result of 2.b Step 1

(5) $q^{<*,f(*,*)>}(X, Y_1, Y_2) : -edge^{<*,f(*,*)>}(X, Y_1, Y_2)$

- by replacing $edge(X, Y)$ in (1) by $edge^{<*,f(*,*)>}(\ldots, \ldots, \ldots)$ from (4')

(6) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Y)$

- by replacing $edge(X, Y)$ in (1) by $edge^{<f(*,*),*>}(\ldots, \ldots, \ldots)$ from (3')

Inverse rules algo. Example

$(\mathcal{P}^-, s^{-1})$ (1) $q(X, Y) : -edge(X, Y)$

(2) $q(X, Y) : -edge(X, Z), q(Z, Y)$

(3) $edge(f(X, Y), Y) : -v(X, Y)$

(4) $edge(X, f(X, Y)) : -v(X, Y)$

2.a (3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$

(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$

2.b Step 1 (5) $q^{<*,f(*,*)>}(X, Y_1, Y_2) : -edge^{<*,f(*,*)>}(X, Y_1, Y_2)$

(6) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Y)$

2.b Step 2 (7) $q^{<*,*>}(X, Y) : -$
$edge^{<*,f(*,*)>}(X, Z_1, Z_2), q^{<f(*,*),*>}(Z_1, Z_2, Y)$

- by replacing $edge(X, Z)$ in (2) by $edge^{<*,f(*,*)>}(\ldots, \ldots, \ldots)$ from (4')
- and by replacing $q(Z, Y)$ in (2) by $q^{<f(*,*),*>}(\ldots, \ldots, \ldots)$ from (6)

(8) $q^{<f(*,*),f(*,*)>}(X_1, X_2, Y_1, Y_2) : -$
$edge^{<f(*,*),*>}(X_1, X_2, Z), q^{<*,f(*,*)>}(Z, Y_1, Y_2)$

- by replacing $edge(X, Z)$ in (2) by $edge^{<f(*,*),*>}(\ldots, \ldots, \ldots)$ from (3')
- and by replacing $q(Z, Y)$ in (2) by $q^{<*,f(*,*)>}(\ldots, \ldots, \ldots)$ from (5)

Inverse rules algo. Example

$(\mathcal{P}^-, s^{-1})$ (1) $q(X,Y) :- edge(X,Y)$
(2) $q(X,Y) :- edge(X,Z), q(Z,Y)$
(3) $edge(f(X,Y),Y) :- v(X,Y)$
(4) $edge(X,f(X,Y)) :- v(X,Y)$

2.a (3') $edge^{<f(*,*),*>}(X,Y,Y) :- v(X,Y)$
(4') $edge^{<*,f(*,*)>}(X,X,Y) :- v(X,Y)$

2.b Step 1 (5) $q^{<*,f(*,*)>}(X,Y_1,Y_2) :- edge^{<*,f(*,*)>}(X,Y_1,Y_2)$
(6) $q^{<f(*,*),*>}(X_1,X_2,Y) :- edge^{<f(*,*),*>}(X_1,X_2,Y)$

2.b Step 2 (7) $q^{<*,*>}(X,Y) :-$
$edge^{<*,f(*,*)>}(X,Z_1,Z_2), q^{<f(*,*),*>}(Z_1,Z_2,Y)$
(8) $q^{<f(*,*),f(*,*)>}(X_1,X_2,Y_1,Y_2) :-$
$edge^{<f(*,*),*>}(X_1,X_2,Z), q^{<*,f(*,*)>}(Z,Y_1,Y_2)$

2.b Step 3 (9) $q^{<f(*,*),*>}(X_1,X_2,Y) :-$
$edge^{<f(*,*),*>}(X_1,X_2,Z), q^{<*,*>}(Z,Y)$

- by replacing $edge(X,Z)$ in (2) by $edge^{<f(*,*),*>}(\ldots,\ldots,\ldots)$ from (3')
- and by replacing $q(Z,Y)$ in (2) by $q^{<*,*)>}(\ldots,\ldots)$ from (7)

(10) $q^{<*,f(*,*)>}(X,Y_1,Y_2) :-$
$edge^{<*,f(*,*)>}(X,Z_1,Z_2), q^{<f(*,*),f(*,*)>}(Z_1,Z_2,Y_1,Y_2)$

- by replacing $edge(X,Z)$ in (2) by $edge^{<*,f(*,*)>}(\ldots,\ldots,\ldots)$ from (4')
- and by replacing $q(Z,Y)$ in (2) by $q^{<f(*,*),f(*,*)>}(\ldots,\ldots,\ldots,\ldots)$ from (8)

2.b End

Inverse rules algo. Example

The result of part 2 of the algorithm is the following datalog program, called $(\mathcal{P}^-, s^{-1})^{datalog}$ or $(\mathcal{P}^-, s^{-1})^{split}$:

(3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$

(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$

(5) $q^{<*,f(*,*)>}(X, Y_1, Y_2) : -edge^{<*,f(*,*)>}(X, Y_1, Y_2)$

(6) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Y)$

(7) $q^{<*,*>}(X, Y) : - \ edge^{<*,f(*,*)>}(X, Z_1, Z_2), q^{<f(*,*),*>}(Z_1, Z_2, Y)$

(8) $q^{<f(*,*),f(*,*)>}(X_1, X_2, Y_1, Y_2) : -$
$edge^{<f(*,*),*>}(X_1, X_2, Z), q^{<*,f(*,*)>}(Z, Y_1, Y_2)$

(9) $q^{<f(*,*),*>}(X_1, X_2, Y) : -$
$edge^{<f(*,*),*>}(X_1, X_2, Z), q^{<*,*>}(Z, Y)$

(10) $q^{<*,f(*,*)>}(X, Y_1, Y_2) : -$
$edge^{<*,f(*,*)>}(X, Z_1, Z_2), q^{<f(*,*),f(*,*)>}(Z_1, Z_2, Y_1, Y_2)$

Part 2 always terminates since function symbols are introduced into inverse rules that are not recursive.

Inverse rules algo. Optim.

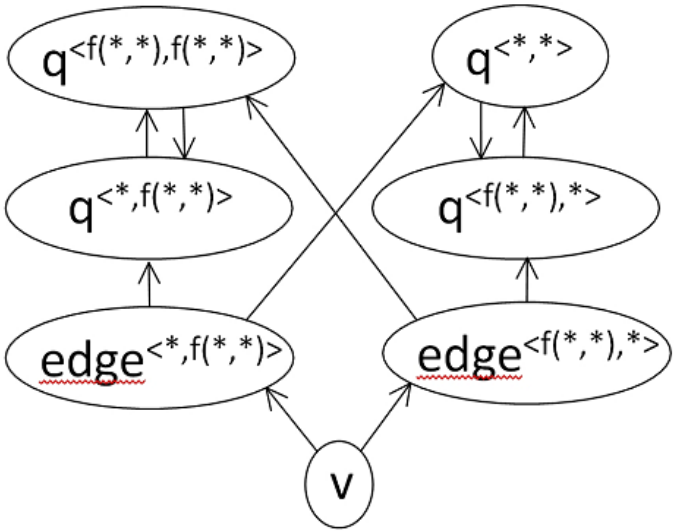
Optimizations can be applied to $(\mathcal{P}^-, s^{-1})^{datalog}$:

3.a If a predicate p cannot contribute answers to the query (because there is no path from p to the query predicate in the dependency graph) then rules defining p can be deleted

3.b IDB predicates defined by one rule with one atom body can be replaced by their body atom (be careful of variables).

Inverse rules algo. Example

Here is the dependency graph of the program:



So rules defining predicates $q^{<*,f(*,*)>}$ and $q^{<f(*,*),f(*,*)>}$ can be deleted. We obtain then:

(3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$
(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$
(6) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Y)$
(7) $q^{<*,*>}(X, Y) : - edge^{<*,f(*,*)>}(X, Z_1, Z_2), q^{<f(*,*),*>}(Z_1, Z_2, Y)$
(9) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Z), q^{<*,*>}(Z, Y)$

Inverse rules algo. Example

In the current program, rule 3.b cannot be applied beacause of variables. So the result is not changed wrt 3.a.

$(\mathcal{P}^{-}, s^{-1})^{datalog}$:

(3') $edge^{<f(*,*),*>}(X, Y, Y) : -v(X, Y)$

(4') $edge^{<*,f(*,*)>}(X, X, Y) : -v(X, Y)$

(6) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Y)$

(7) $q^{<*,*>}(X, Y) : - \ edge^{<*,f(*,*)>}(X, Z_1, Z_2), q^{<f(*,*),*>}(Z_1, Z_2, Y)$

(9) $q^{<f(*,*),*>}(X_1, X_2, Y) : -edge^{<f(*,*),*>}(X_1, X_2, Z), q^{<*,*>}(Z, Y)$

is the maximally contained query plan for our problem.

Inverse rules algo. Results

For a set of source description $\mathcal{D}$, let $\mathcal{P}$ be a query plan. Then we note $\mathcal{P}\downarrow$ the plan that computes the same answers than $\mathcal{P}$ on $\mathcal{D}$ without tuples that use a function symbol.

About $(\mathcal{P}^-, s^{-1})$ and $(\mathcal{P}^-, s^{-1})\downarrow$

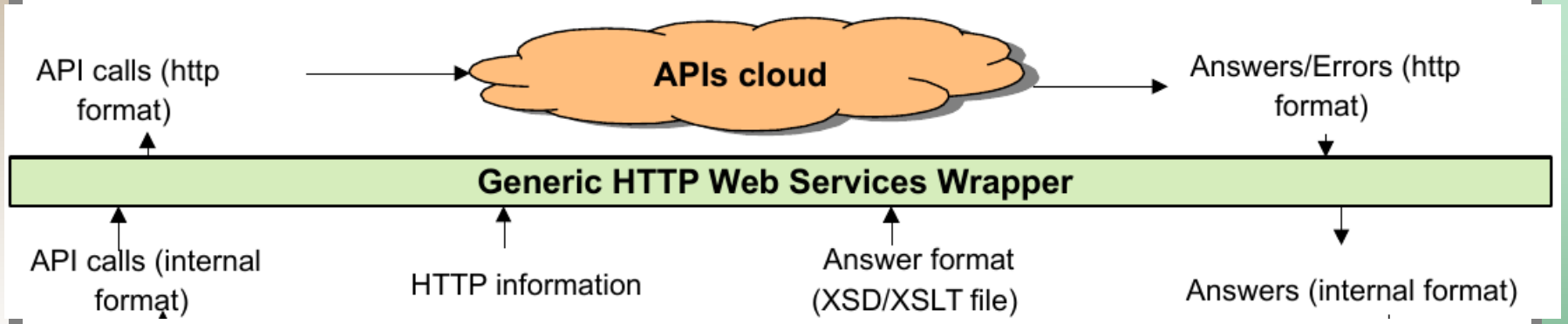
- $\forall \mathcal{P}, \forall s, \forall I_s(\mathit{finite})$, $(\mathcal{P}^-, s^{-1})$ has a unique minimal fixpoint (cf semantics of a logic program) and naive and semi naive evaluation always terminate and produce this fixpoint.
- $(\mathcal{P}^-, s^{-1})$ can be built in time polynomial in the sizes of $\mathcal{P}$ and s.
- $\forall \mathcal{P}, \forall s$, $(\mathcal{P}^-, s^{-1})\downarrow$ is maximally contained in $\mathcal{P}$

About $(\mathcal{P}^-, s^{-1})^{datalog}$

- $\forall \mathcal{P}, \forall s$, $(\mathcal{P}^-, s^{-1})\downarrow$ is equivalent to $(\mathcal{P}^-, s^{-1})^{datalog}$
- $\forall \mathcal{P}, \forall s$, $(\mathcal{P}^-, s^{-1})^{datalog}$ is maximally contained in $\mathcal{P}$.
- If there exists a query plan equivalent to $\mathcal{P}$ then $(\mathcal{P}^-, s^{-1})^{datalog}$ is equivalent to $\mathcal{P}$.

Architecture of the generic wrapper

Wrapper overview



Inputs and output

- Inputs
  - Enterprise Information (Authentication parameters, subdomains, profile related information)
  - HTTP URL template, method
  - HTTP body template (XML)
  - XML Schema (XSD)
  - XML Transformation (XSLT)
- Output
  - Transformed (desired) information from WS
  - Or Error

More detailed architecture