Reasoning on Data:

An introduction to Ontology-Based Data Access

Marie-Laure Mugnier

University of Montpellier

École EGC 2019 – Metz

KNOWLEDGE BASED SYSTEMS

Knowledge Base (KB) ⇔ Reasoning Services

- **General knowledge on the application domain**
  « Cats are Mammals »

  Ontology

- **Factual knowledge**
  Description of specific individuals, situations, ...

  Félix is a Cat

  Factbase, Database(s)

Knowledge expressed in a knowledge representation (KR) language

Fundamental tasks

- **Checking the consistency** of the KB
- **Computing answers** to a query over the KB

...

Reasoning algorithms associated with the KR language

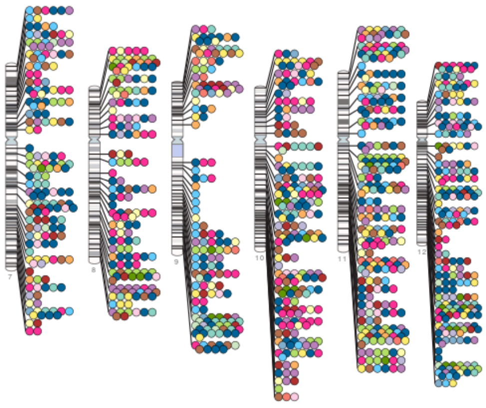
At the Heart of KB-Systems: Ontologies

What is an ontology?

A **formal specification of the knowledge of a particular domain**

- which allows for machine processing
- that relies on the semantics of knowledge

  > automated reasoning

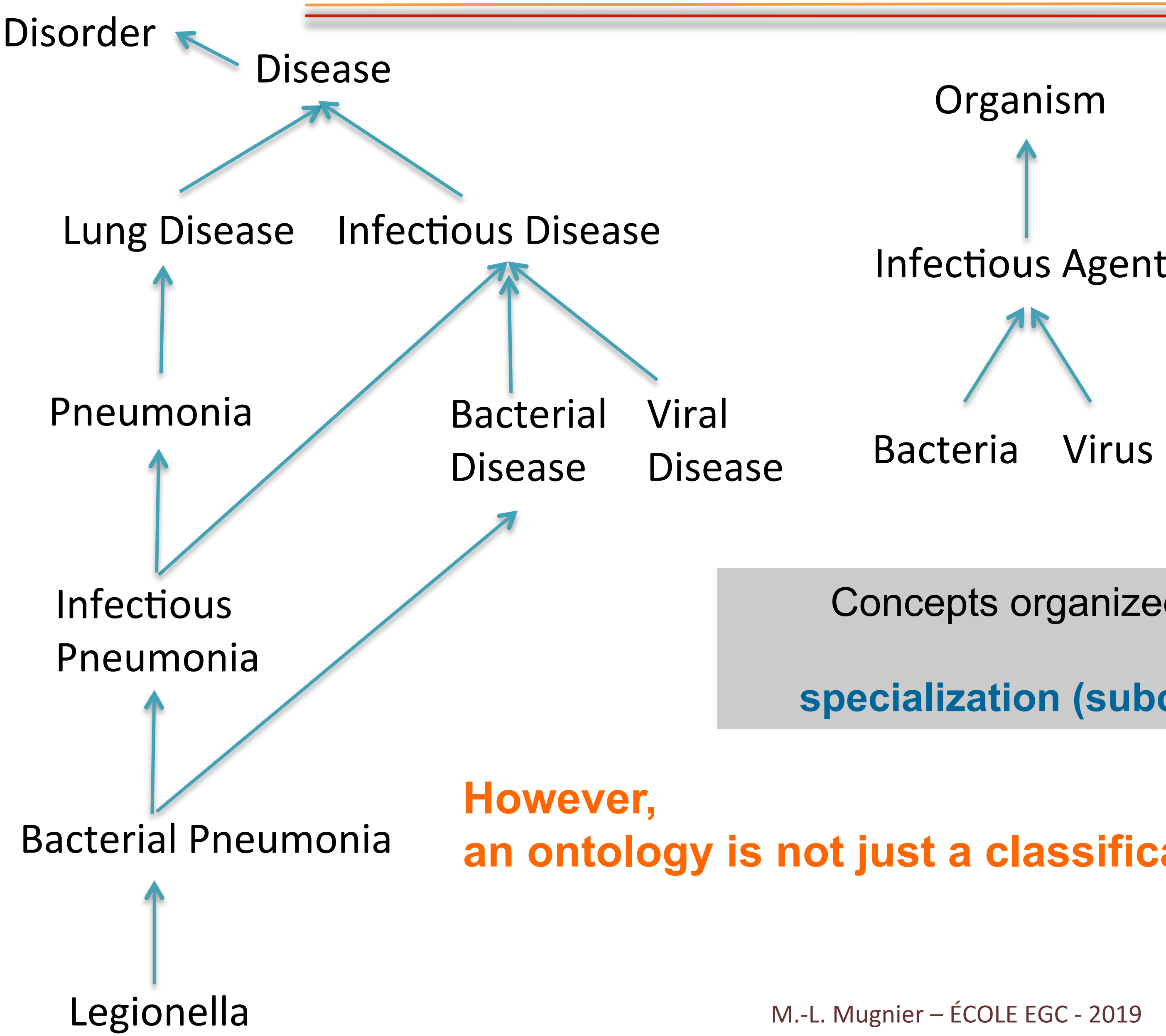


- **Medecine** and **life sciences**: hundreds of available ontologies

  SNOMED CT (400 000 terms), GALEN (> 30 000 terms), FMA (anatomy), ...

- Information systems of **large organizations and corporations**

At the Heart of Ontologies: Concepts / Classes



(SNOMED CT)

Concepts organized by

specialization (subclass)

However,
an ontology is not just a classification!

ONTOLOGIES ARE MUCH MORE THAN CLASSIFICATIONS

Formal specification of the knowledge of a particular domain

- **Vocabulary**

1. concepts / classes
2. relations

- **Semantic relationships between the terms of the vocabulary**

specialization on concepts

specialization on relations

+ properties of concepts

+ properties of relations

+ other axioms

What can be expressed depends on the KR language

→ different subsets of first-order logic

RELATIONS (THAT CAN HOLD BETWEEN CONCEPT INSTANCES)

Often these are binary relations (also called « roles » or « properties »)

dueTo

Argument 1 : a Disorder
Argument 2 : an Organism or [...]

subrelation

hasCausativeAgent

Argument 1 : a Disease
Argument 2 : an Organism

$$\forall x \forall y\ (hasCausativeAgent(x,y) \rightarrow dueTo(x,y))$$

Signature of a relation : assigns a maximum concept to each argument (« domain » and « range » in RDFS and OWL)

$$\forall x \forall y\ (hasCausativeAgent(x,y) \rightarrow Disease(x) \wedge Organism(y))$$

OTHER FREQUENT TYPES OF AXIOMS

- Negative constraints (disjointness between concepts, relations, ...)

Bacteria ∩ Virus = ∅

$\forall x\ (Bacteria(x) \wedge Virus(x) \rightarrow \bot)$
$\forall x\ (Bacteria(x) \rightarrow \neg Virus(x))$

- Necessary and/or sufficient properties of concepts (ex: BacterialDisease)

A bacterial disease is caused by a bacteria

$\forall x\ (BacterialDisease(x) \rightarrow \exists y\ (Bacteria(y) \wedge hasCausativeAgent(x,y)))$

- Properties of relations

inverse relations: $\forall x \forall y\ (hasPart(x,y) \leftrightarrow isPartOf(y,x))$

symmetry, transitivity, ...

functional relation:

$\forall x \forall y \forall z\ (isDirectlyPartOf(x,y) \wedge isDirectlyPartOf(x,z) \rightarrow y = z)$

What kinds of Languages to Express Ontologies?

Very light languages

Hierarchies of classes
Hierarchies of binary relations (called « properties »)
Signatures of these relations (« domain » and « range »)

→ **RDF Schema**

More expressive languages

Description Logics	
Rule-based languages	Datalog, existential rules, RDF deductive rules ...

From a more abstract viewpoint: set of logical sentences of the form (roughly)

$\forall X \forall Y$ **(condition[X,Y] → conclusion[X,..])** **« rules »**

WHAT ARE ONTOLOGIES GOOD FOR?

- **provide a common vocabulary**

  → it is easier to share information
  (typically between experts of several domains)

- **constrain the meaning of terms**
  → forces to explicit not-said things and to remove ambiguities
  hence less misunderstandings

- to do **automated reasoning**, basis of high-level services

  → find implicit links between pieces of knowledge
  → check the consistency of the KB, find errors in modeling
  → **enrich data query answering**

Ontology-Mediated Query Answering

Ex: Medical Records

Query (SQL, SPARQL, MongoDB ...)

« find all patients affected by a lung disease due to a bacteria »

??

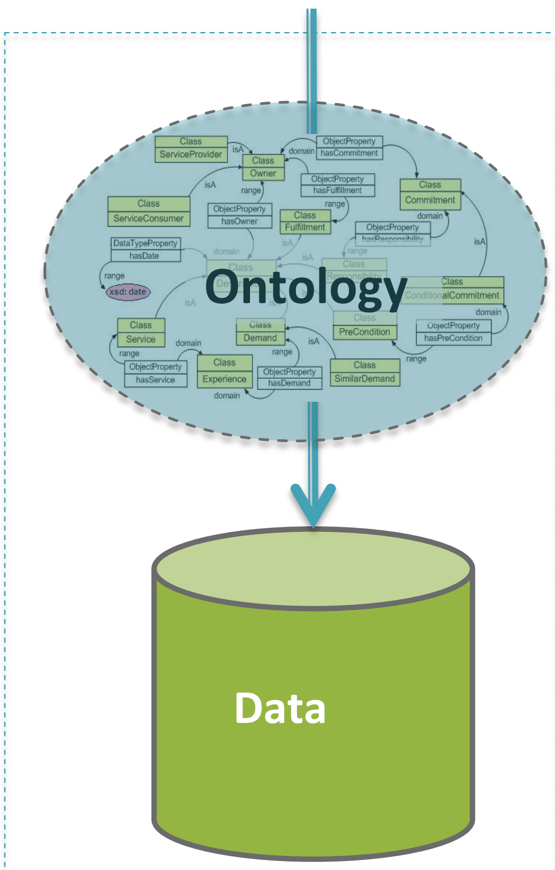
Patient P : Diagnosis = « legionella »

Database (relational, RDF, NoSQL, ...)

Ontology-Mediated Query Answering

Query

« find all patients affected by a lung disease due to a bacteria »



A legionella is bacterial pneumonia
A bacterial pneumonia is a pneumonia
A pneumonia is a **lung disease**
A bacterial pneumonia is caused by a **bacteria**
If *x* is caused by *y* then *x* **is due to** *y*
If the diagnosis of a patient *x* contains a disease y then *x* is **affected by** *y*

Patient P : Diagnosis = « legionella »

Knowledge Base

CONJUNCTIVE QUERIES (CQ)

« find all patients affected by a lung disease due to a bacteria »

q(x) = ∃y ∃z (Patient(x) ∧ isAffBy(x,y) ∧ LungDisease(y) ∧ dueTo(y,z) ∧ Bacteria(z))

A **CQ** is an **existentially quantified conjunction of atoms**

The **free variables** are the **answer variables**

Datalog notation

q(x) ← Patient(x), isAffBy(x,y), LungDisease(y), dueTo(y,z), Bacteria(z)

Select-Join-Project queries in relational algebra (SQL)

SELECT ... FROM … WHERE *<join conditions>*

SPARQL (semantic web queries)

SELECT … WHERE *<basic graph pattern>*

OMQA Example: Ontological Knowledge

A legionella is bacterial pneumonia

$$\forall x\ (Legionella(x) \rightarrow BacterialPneumonia(x))$$

A bacterial pneumonia is a pneumonia
A pneumonia is a lung disease
A bacterial pneumonia is caused by a bacteria

$$\forall x\ (BacterialPneumonia(x) \rightarrow \exists y\ (hasCausativeAgent(x,y) \wedge Bacteria(y)))$$

If *x* is caused by *y* then *x* is due to *y*

$$\forall x \forall y\ (hasCausativeAgent(x,y) \rightarrow dueTo(x,y))$$

If the diagnosis of a patient *x* contains a disease y then *x* is affected by *y*

$$\forall x \forall y\ ((Diagnosis(x,y) \wedge Disease(y)) \rightarrow isAffectedBy(x,y))$$

FACTBASE

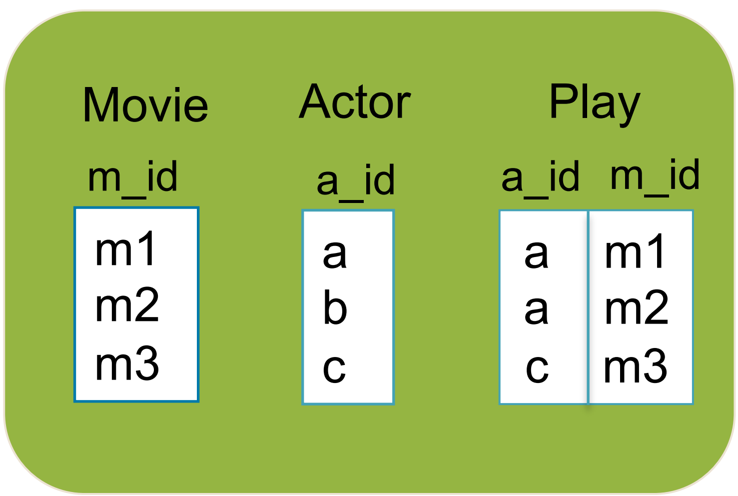
Factbase : a set of **ground logical atoms** (built on the ontological vocabulary)

{ movie(m1), movie(m2), movie(m3)
actor(a), actor(b), actor(c)
play(a,m1), play(a,m2), play(c,m3) }

seen as the conjunction of these atoms

Provides an abstract view of many data formats (relational, RDF, ...)

Relational database



ANSWERS TO A CONJUNCTIVE QUERY

$q(x) = \exists y\ (movie(y) \wedge play(x, y))$

movie(y)
play(x, y)

F

movie(m1)
movie(m2)
movie(m3)
actor(a)
actor(b)
actor(c)
play(a,m1)
play(a,m2)
play(c,m3)

Homomorphism *h* from *q* to *F*:
substitution of variables(*q*) *by* constants*(F)*
such that h(q) $\subseteq$ *F*

h1 : x → a
y → m1

$h1(q) = movie(m1) \wedge play(a, m1)$

h2 : x → a
y → m2

$h2(q) = movie(m2) \wedge play(a, m2)$

h3 : x → c
y → m3

$h3(q) = movie(m3) \wedge play(c, m3)$

Answers: obtained by restricting the domains of homomorphisms to the answer variables

x = a
x = c

On the Medical Example

$q(x) = \exists y \exists z\ (Patient(x) \wedge isAffectedBy(x,y) \wedge LungDisease(y) \wedge dueTo(y,z) \wedge Bacteria(z))$

« find all patients affected by a lung disease due to a bacteria »

Factbase = { Patient(P), Diagnosis(P,M), Legionella(M) }

« The diagnosis for the patient P is legionella »

No answer to *q* on the Factbase

Legionella *specialisation of* LungDisease *and* BacterialDisease (*and* Disease)

hence LungDisease(M)

hence BacterialDisease(M), Disease(M)

$\forall x\ (BacterianDisease(x) \rightarrow \exists y\ (hasCausativeAgent(x,y) \wedge Bacteria(y)))$

hence hasCausativeAgent(M,b) and **Bacteria(b)**

$\forall x \forall y\ (hasCausativeAgent(x,y) \rightarrow dueTo(x,y))$

hence dueTo(M,b)

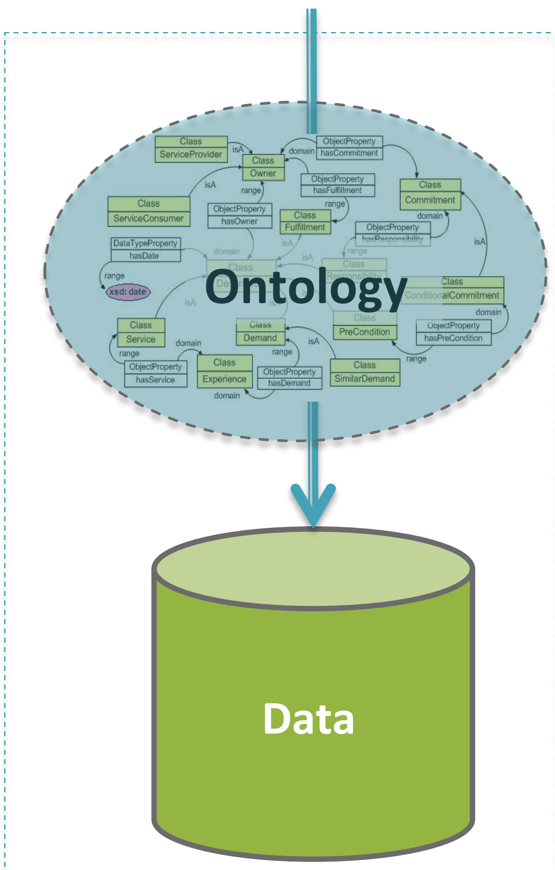
$\forall x \forall y\ ((Diagnosis(x,y) \wedge Disease(y)) \rightarrow isAffectedBy(x,y))$

hence isAffectedBy(P,M)

Answer : x = P

ONTOLOGY-MEDIATED QUERY ANSWERING

Query



Knowledge Base

Compute answers to queries

while taking into account

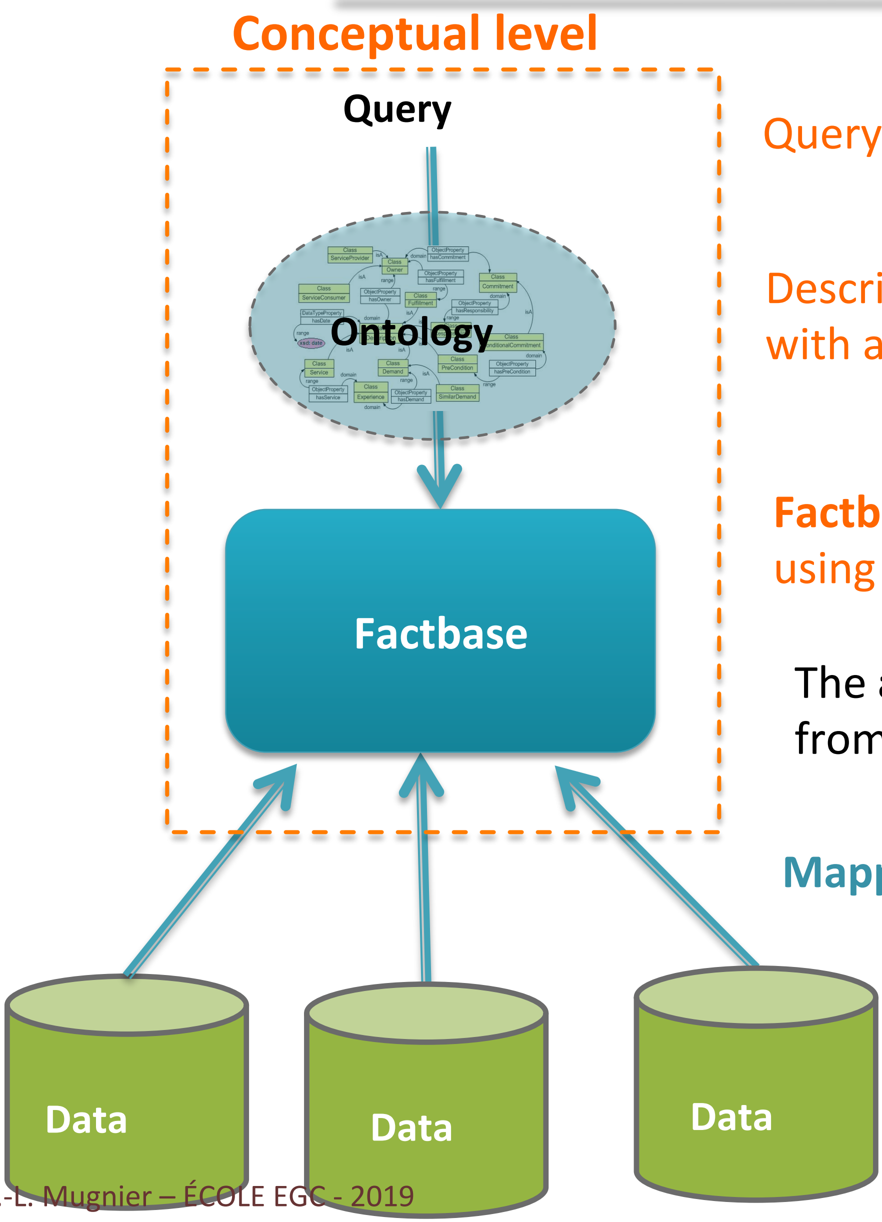
inferences enabled by an ontology

Limitation up to now:

ontology and data expressed on the same vocabulary

A MORE GENERAL SCHEMA: OBDA

« Ontology-Based **Data Access** »
[Poggi et al., JoDS, 2008]



Query using the vocabulary of the ontology

Description of the application domain with a high abstraction level

Factbase (possibly virtual)
using the vocabulary of the ontology

The **answers** to the query are **inferred** from the knowledge base

Mappings from data to facts

{ Database query ⇝ Facts }

Independent and heterogeneous data sources

Mappings

Patient_T [ID_PATIENT, NAME,SSN]

Diagnosis_T[ID_PATIENT, DISORDER]

Patient /1
Diagnosis / 2
Legionella /1

Mapping: database query(X) $\rightsquigarrow$ factual pattern using X

$q_1(x)$: $\exists n \exists s$ Patient_T (x,n,s) $\rightsquigarrow$ Patient(x)

$q_2(x)$: $\exists n \exists s$ Patient_T $(x,n,s) \wedge$ Diagnostic_T$(x,y) \wedge y =$ « Legionella »
$\rightsquigarrow$ $\exists z$ (diagnosis(x,z) $\wedge$ legionella(z))

Patient_T

id	name	ssn
P	..	..
..	..	..
..	..	..

Diagnosis_T

id	dis
P	« Leg. »
..	..
..	..

$\rightsquigarrow$ *triggering the mappings*

Patient(P)
Diagnosis(P,M)
Legionella(M)

BACK TO ONTOLOGICAL LANGUAGES

- **Description Logics** (base of OWL 2)

  « Classical » DLs lead to high complexity of query answering

  → development of lighter DLs (« **Horn DLs** »)

  in particular, core of **OWL2 tractable profiles**

DL-Lite$_R$	**OWL 2 QL**
EL	**OWL 2 EL**

- **RDFS**
- **Rule-based languages**

Datalog	**∀X∀Y (conjunction[X,Y] → p(X))**
RDF rules	**{ triples } → { triples }**

LOGICAL TRANSLATION OF DL-LITE$_R$

Base of OWL 2 QL

$$B_1 \sqsubseteq B_2 \quad B_1 \sqsubseteq \neg B_2 \quad S_1 \sqsubseteq S_2 \quad S_1 \sqsubseteq \neg S_2$$

$$B := A \mid \exists S \qquad S := R \mid R^-$$

∀x (A(x) → B(x))	A subclass of B
∀x ∀y (r(x,y) → A(x))	A domain of r
∀x ∀y (r(x,y) → B(y))	B range of r
∀x ∀y (r(x,y) → s(x,y))	r subproperty of s
∀x ∀y (r(x,y) → s(y,x))	role inversion
∀x ∀y (r(x,y) → ∃ z s(x,z))	s mandatory role
∀x (A(x) → ∃ z s(x,z))	s mandatory role
∀x (A(x) ∧ B(x) → ⊥)	A and B disjoint
∀x ∀y (r(x,y) ∧ s(x,y) → ⊥)	r and s disjoint

Ontological Languages

- **Description Logics** (base of OWL 2)

  « Classical » DLs lead to high complexity of query answering

  → development of lighter DLs (« **Horn DLs** »)

  in particular, core of **OWL2 tractable profiles**

DL-Lite$_R$	**OWL 2 QL**
EL	**OWL 2 EL**

- **RDFS**
- **Rule-based languages**

Datalog	**∀X∀Y (conjunction[X,Y] → p(X))**
RDF rules	**{ triples } → { triples }**

LOGICAL TRANSLATION OF RDFS (1)

Translation 1 : follows the classical KR approach

Strict separation between classes / properties / instances

class $\rightsquigarrow$ unary predicate

property $\rightsquigarrow$ binary predicate

The translation of ontological triples yields a subset of DL-Lite$_R$

$\forall x\, (A(x) \rightarrow B(x))$	<A subclassOf B>
$\forall x \forall y\, (r(x,y) \rightarrow s(x,y))$	<r subpropertyOf s>
$\forall x \forall y\, (r(x,y) \rightarrow A(x))$	<A domain of r>
$\forall x \forall y\, (r(x,y) \rightarrow B(y))$	<B range of r>

Logical translation of RDFS (2)

Translation 2: follows the RDF approach

Everything is triple

URI ⇝ constant, blank node ⇝ variable
a single ternary predicate « triple »
<s p o> ⇝ triple(s,p,o)

Reasoning is based on RDF(S) entailment rules

Ex: <s type C1>, <C1 subclass C2> → <s type C2>

$\forall s \forall C1 \forall C2\ (triple(s,type,C1) \wedge triple(C1,subclass,C2) \rightarrow triple(s,type,C2))$

The translation of RDF(S) entailment rules yields a subset of Datalog

ONTOLOGICAL LANGUAGES

- **Description Logics** (base of OWL 2)

  « Classical » DLs lead to high complexity of query answering

  → development of lighter DLs (« **Horn DLs** »)

  in particular, core of **OWL2 tractable profiles**

  DL-Lite$_R$ OWL 2 QL
  EL **OWL 2 EL**

All these languages can be translated into logical rules called existential rules

- **RDFS**
- **Rule-based languages**

  Datalog **∀X∀Y (conjunction[X,Y] → p(X))**
  RDF rules **{ triples } → { triples }**

EXISTENTIAL RULES

$\forall X\, \forall Y\, (\, \text{Body}\, [X,Y] \rightarrow \exists\, Z\, \text{Head}\, [X,Z]\,)$

X, Y, Z :
sets of variables

any **positive conjunction** (without functional symbols except constants)

$\forall x\, (\, \text{actor}(x) \rightarrow \exists\, z\, \text{play}(x,z)\,)$

$\forall x\, \forall y\, (\, \text{siblingOf}(x,y) \rightarrow \exists\, z\, (\text{parentOf}(z,x) \wedge \text{parentOf}(z,y))\,)$

we often simplify by omitting universal quantifiers

Key point: ability to assert **the existence of unknown entities**

Important for representing ontological knowledge in **open domains**

RULE APPLICATION

R = ∀x ∀y (siblingOf(x,y) → ∃ z (parentOf(z,x) ∧ parentOf(z,y)))

F = siblingOf(a,b)

R is **applicable** to *F* if there is a **homomorphism** *h* from *body(R)* to *F*

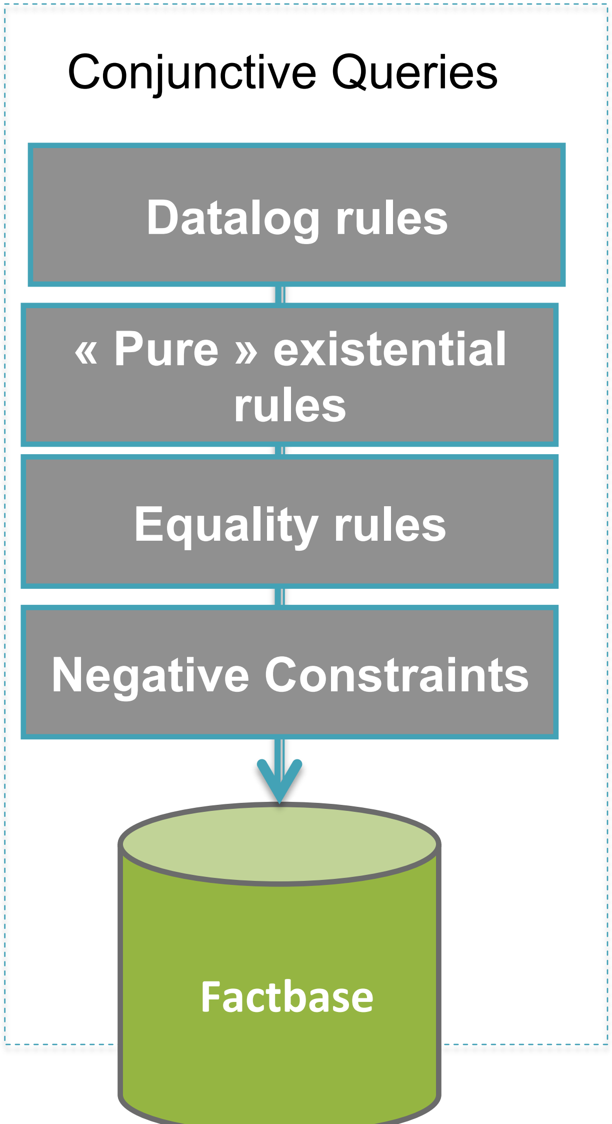
x → a
y → b

Applying *R* to *F* w.r.t. *h* produces *F* ∪ *h(head(R))*

where a new value (variable, null) is created for each existential *variable in R*

F' = **∃ z0** (siblingOf(a,b) ∧ parentOf(z0,a) ∧ parentOf(z0,b))

UNIFYING FRAMEWORK (EXISTENTIAL RULES)



$q(x) = \exists y\, (movie(y) \wedge play(x, y))$

$\forall x\, \forall y\, (movie(y) \wedge play(x,y) \rightarrow actor(x))$

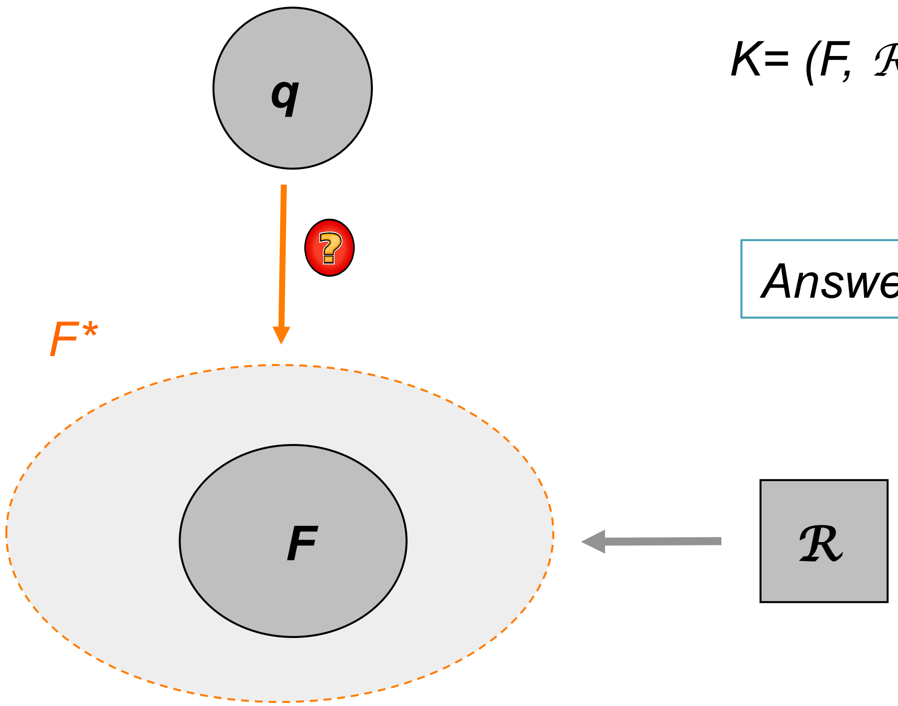
$\forall x\, (\, actor(x) \rightarrow \exists z\, (movie(z) \wedge play(x,z))\,)$

$\forall x\, \forall y\, \forall z\, (\, movie(y) \wedge director(x,y) \wedge director(z,y) \rightarrow x = z\,)$

$\forall x\, (\, movie(x) \wedge person(x) \rightarrow \bot\,)$

movie(m1)
play(a,m1)
...

APPROACH 1 TO RULES : FORWARD CHAINING / MATERIALISATION



$K = (F, \mathcal{R})$

« bottom-up »
« chase »

Answers(q,K) = Answers(q, F)*

Pros: materialisation offline, then online query answering is fast

Cons: volume of the materialisation
not feasible if data is distributed among several databases
not adapted if data change frequently

EXAMPLE (MATERIALIZATION)

$\forall x\ (\text{movieActor}(x) \rightarrow \exists z\ (\text{movie}(z) \wedge \text{play}(x,z)))$

movie(m1)
movie(m2)
movie(m3)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,m3)

movie(*z0*)
play(b,*z0*)

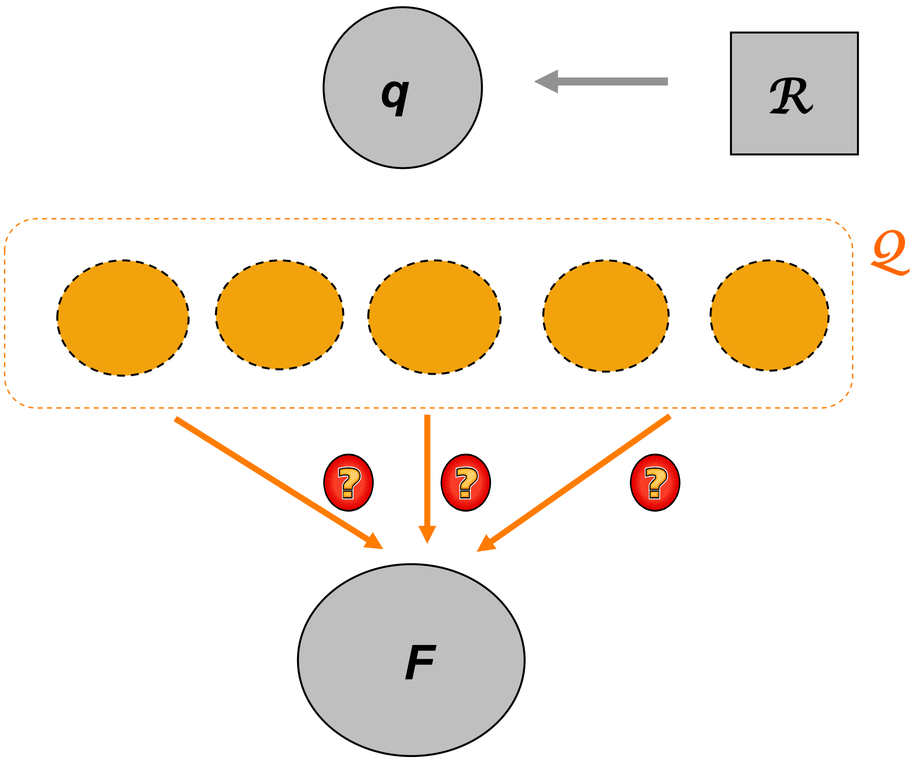
Saturation

$q(x) = \exists y\ (\text{movie}(y) \wedge \text{play}(x, y))$

« find those who play in a movie »

x = a	y = m1
x = a	y = m2
x = b	y = z0
x = c	y = m3

Approach 2 to Rules : Backward Chaining / Query Rewriting



$K = (F, \mathcal{R})$

« top-down » decomposition into 2 steps [DL-Lite]

Rewriting into a set of CQs, seen as a **union of conjunctive queries (UCQ)**

and more generally into a « first-order » query (core SQL query)

Query rewriting is independent from any factbase

For **any** *F*, *Answers(q,(F,* $\mathcal{R}$*)) = Answers(* $\mathcal{Q}$*, F)*

Pros: independent from the data
Cons: rewriting done at query time, easily leads to huge and unusual queries

EXAMPLE

$\forall x\ (\text{movieActor}(x) \rightarrow \exists z\ (\text{movie}(z) \wedge \text{play}(x,z)))$

movie(m1)
movie(m2)
movie(m3)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,m3)

$q(x) = \exists y\ (\text{movie}(y) \wedge \text{play}(x, y))$

« find those who play in a movie »

$\text{Rew}_q(x) = \exists y\ (\text{movie}(y) \wedge \text{play}(x, y)) \vee \text{movieActor}(x)$

Query rewriting

x = a	y = m1
x = a	y = m2
x = c	y = m3

x = a
x = b

Bottleneck of Query Rewriting

Query rewriting into a UCQ easily produces huge and unsual queries

A
|
B_1
|
B_2
⋮
B_n

$B_1(x) \rightarrow A(x)$

$B_2(x) \rightarrow B_1(x)$

…

$B_n(x) \rightarrow B_{n-1}(x)$

$q(x_1 \dots x_k) : A(x_1) \wedge \dots \wedge A(x_k)$

output UCQ: $(n+1)^k$ CQ

It is not a « theoretical worse-case » because class hierarchies are everywhere

→ Rewriting into more compact forms of queries trying to minimize the number of unions

→ Partition the rules: hierarchical rules used to saturate the factbase and the other rules used to rewrite the query

SATURATION MAY NOT HALT (EVEN IN DL-LITE)

R = person(x) → ∃ y hasParent(x,y) ∧ person(y)

F = person(a)

∧ person(y0) ∧ hasParent(a, y0)

∧ person(y1) ∧ hasParent(y0, y1)

(F,{R}) encodes an infinite factbase

However, here: query rewriting with *R* is finite for any *q*

QUERY REWRITING MAY NOT HALT (EVEN FOR DATALOG)

R = friend(u,v) ∧ friend(v,w) → friend(u,w)

q = friend(Giorgos,Maria)

q_1 = friend(Giorgos, v0) ∧ friend (v0,Maria)

q_2 = friend(Giorgos, v1) ∧ friend(v1, v0) ∧ friend (v0,Maria)

*$q_{2'}$ = friend(*Giorgos*, v0) ∧ friend(v0, v1) ∧ friend (v1,Maria)*

q_2 and $q_{2'}$ are equivalent

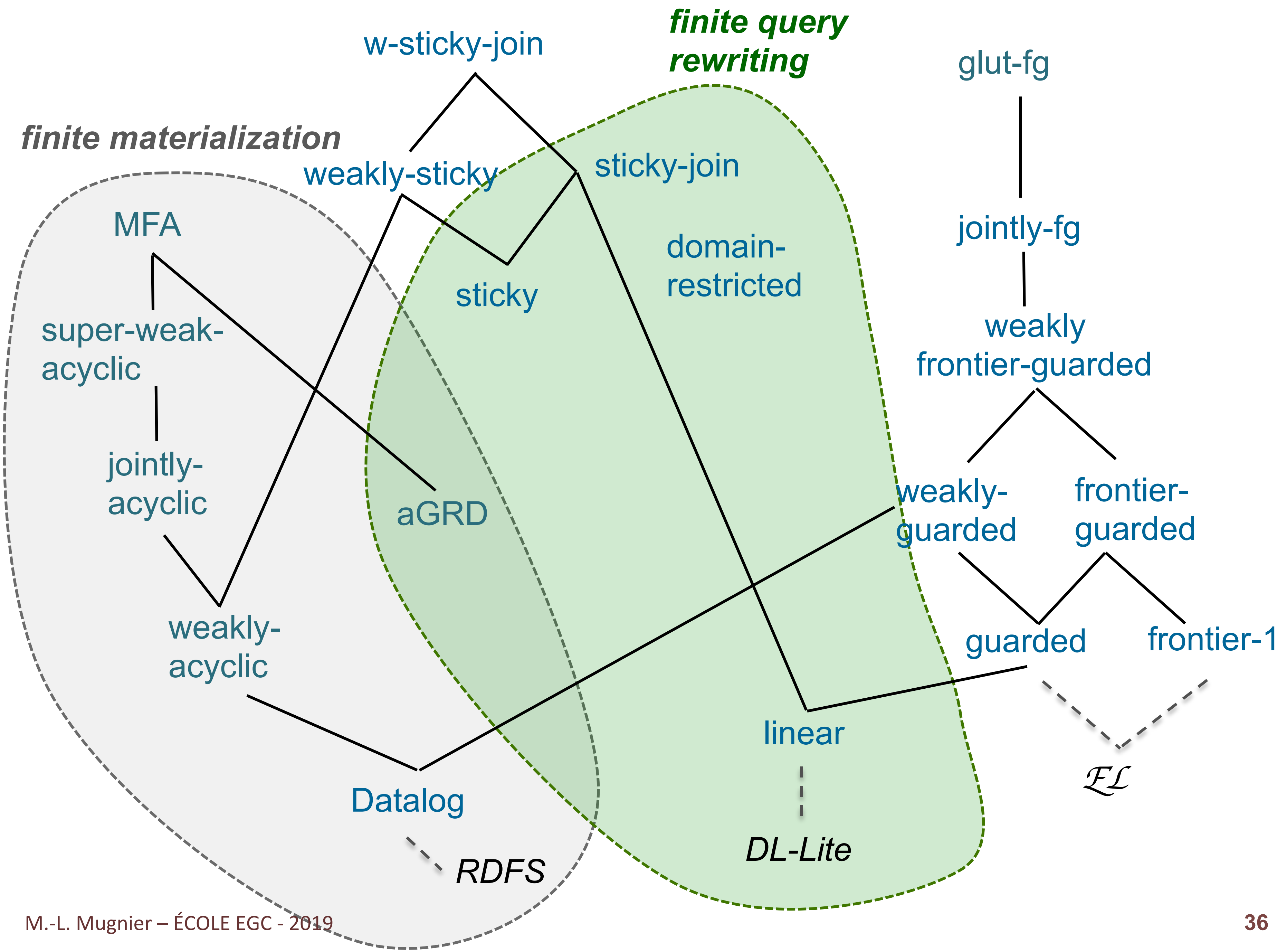
q_3 = friend(Giorgos, v2) ∧ friend(v2, v1) ∧ friend(v1, v0) ∧ friend (v1,Maria)

Etc.

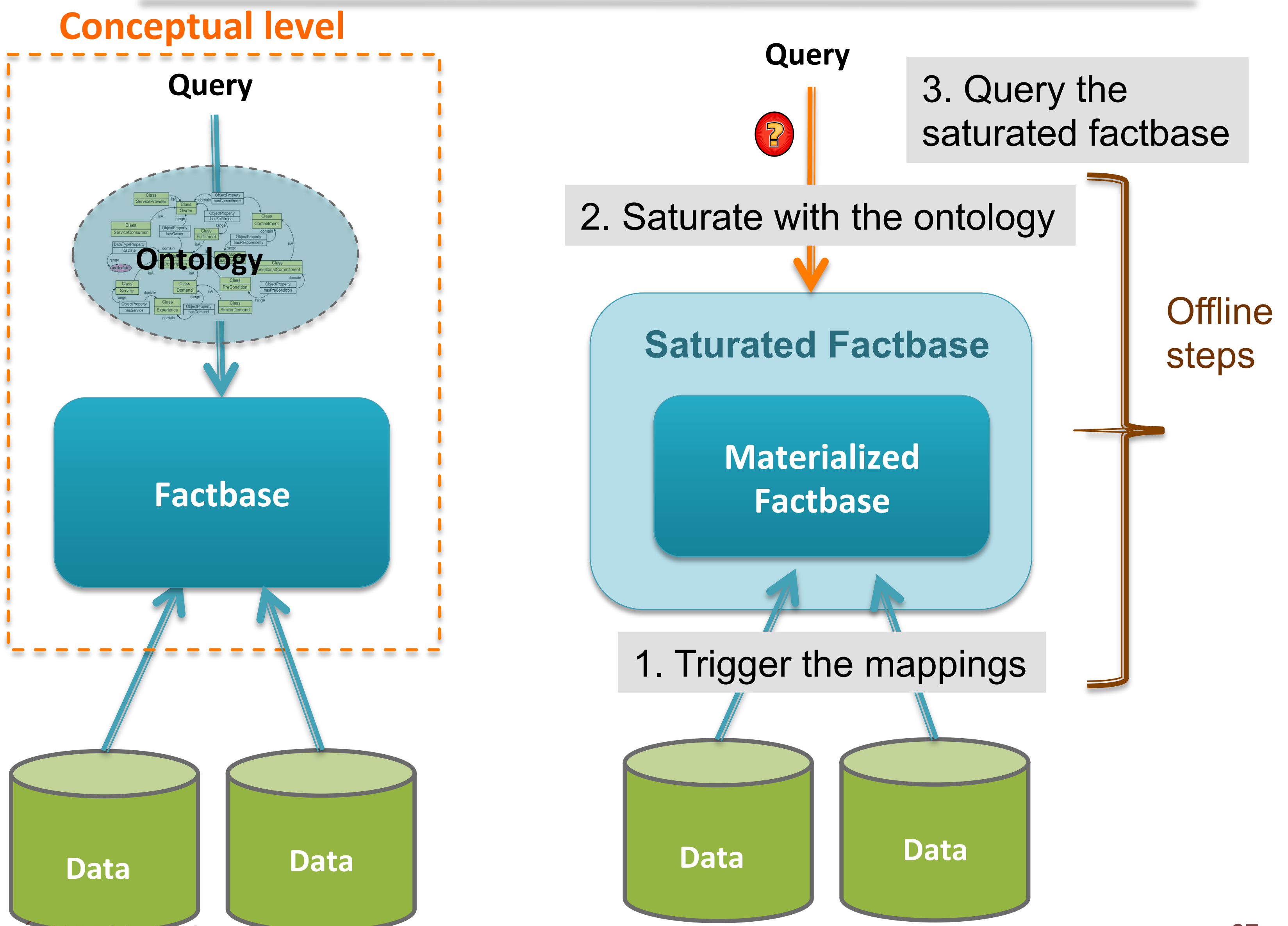
There is an infinite number of non-redundant rewritings

However, here: saturation with *R* is finite for any *F*

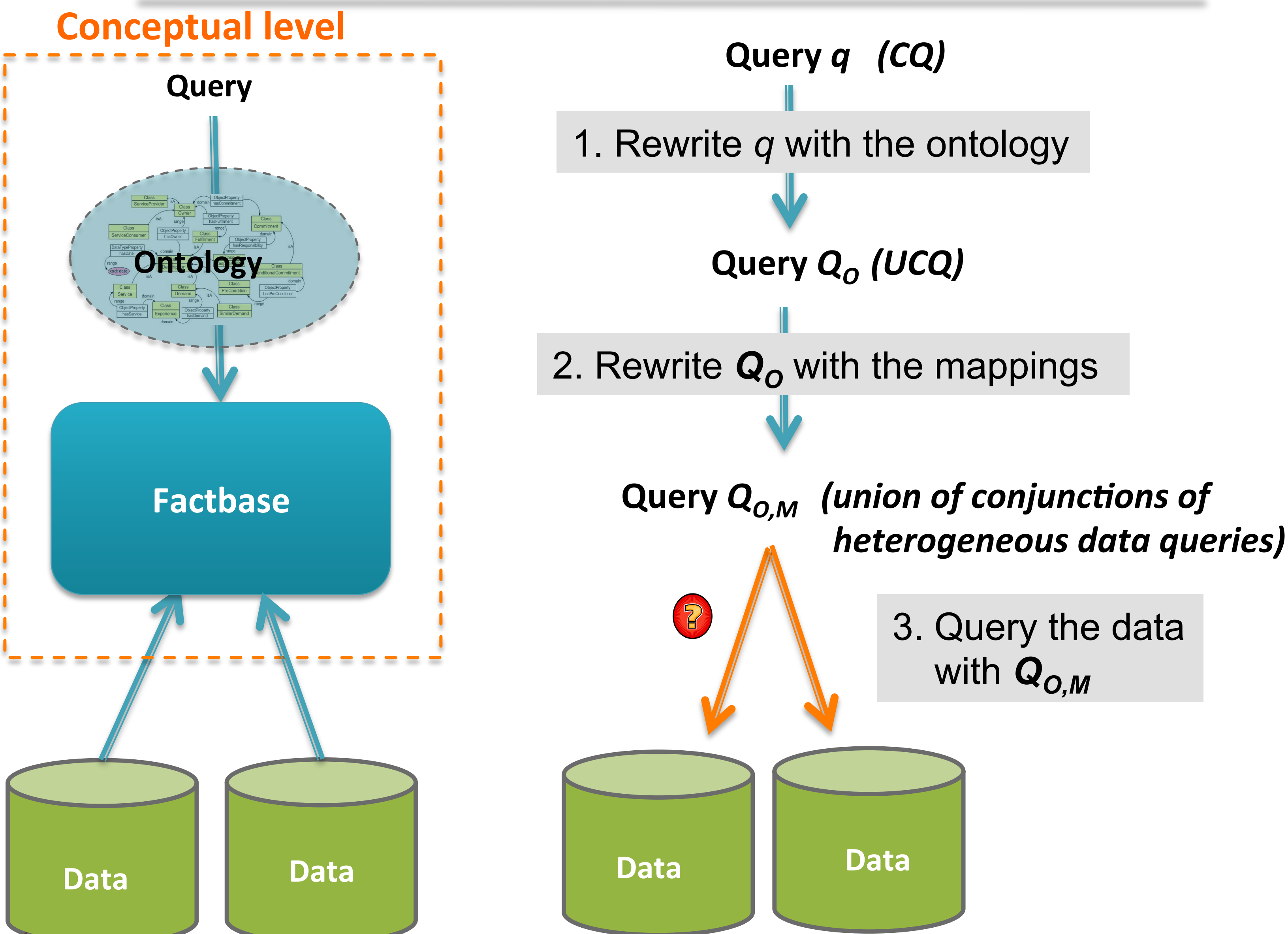
Actually, query answering with existential rules is an **undecidable** problem


finite query rewriting
finite materialization
w-sticky-join
weakly-sticky
sticky-join
sticky
domain-restricted
glut-fg
jointly-fg
weakly frontier-guarded
weakly-guarded
frontier-guarded
guarded
frontier-1
MFA
super-weak-acyclic
jointly-acyclic
weakly-acyclic
aGRD
linear
Datalog
RDFS
DL-Lite
$\mathcal{EL}$

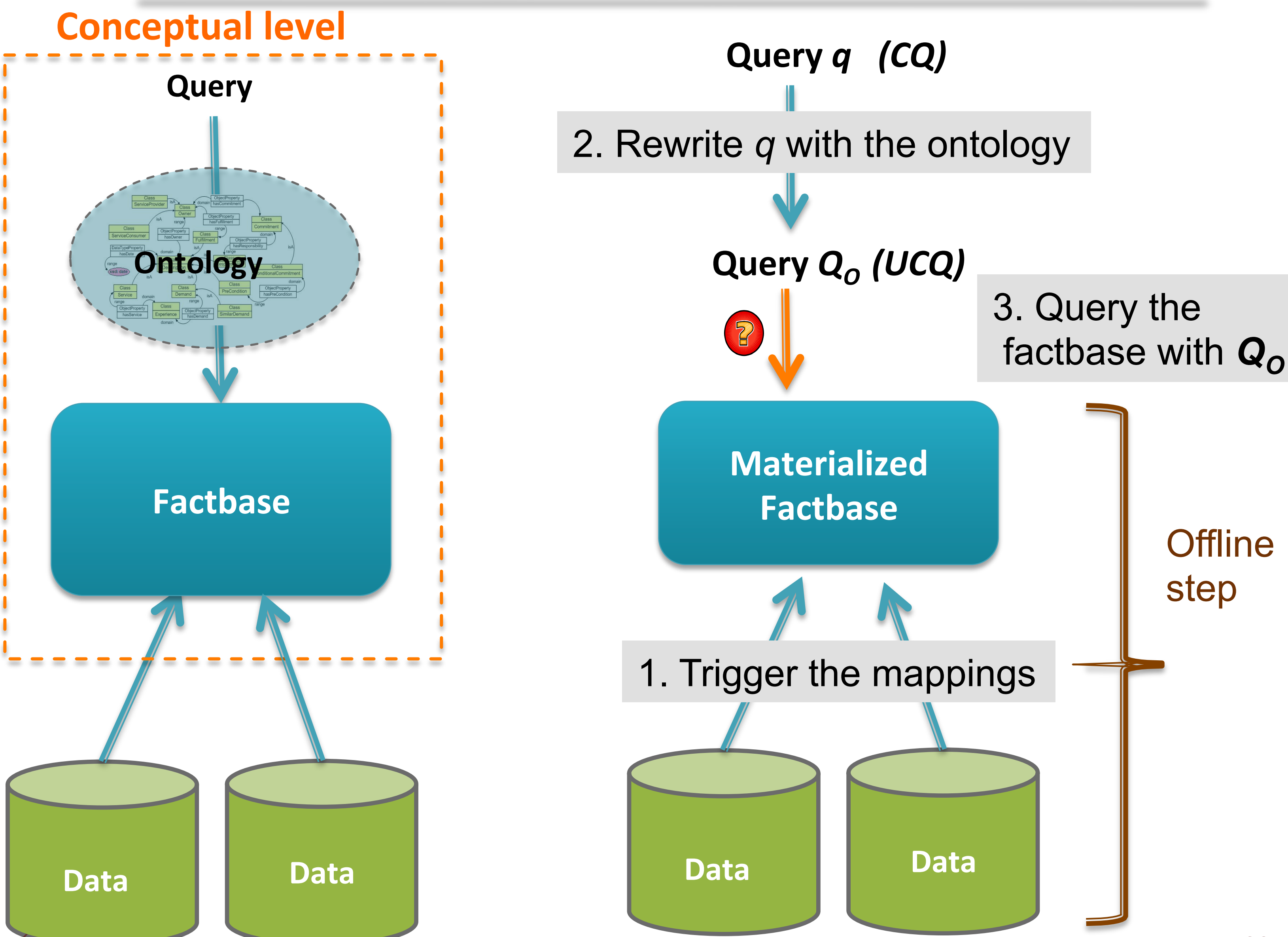
OBDA : TOTAL MATERIALIZATION



OBDA : TOTAL REWRITING



OBDA : Example of Mixed Approach



MAIN AVAILABLE SYSTEMS

- Query rewriting engines for DL-Lite (CQ → UCQ)
  OnTop (tw-rewriting), Rapid, Iqaros, Presto (Mastro), Requiem [...]

- Query rewriting engines for more expressive Horn DLs (CQ → Datalog)
  Rapid, Requiem, Kyrie, Clipper [...]

- Saturation engines for Datalog (extended to existential rules)

  RDFox http://www.cs.ox.ac.uk/isg/tools/RDFox/
  VLog https://github.com/jrbn/vlog

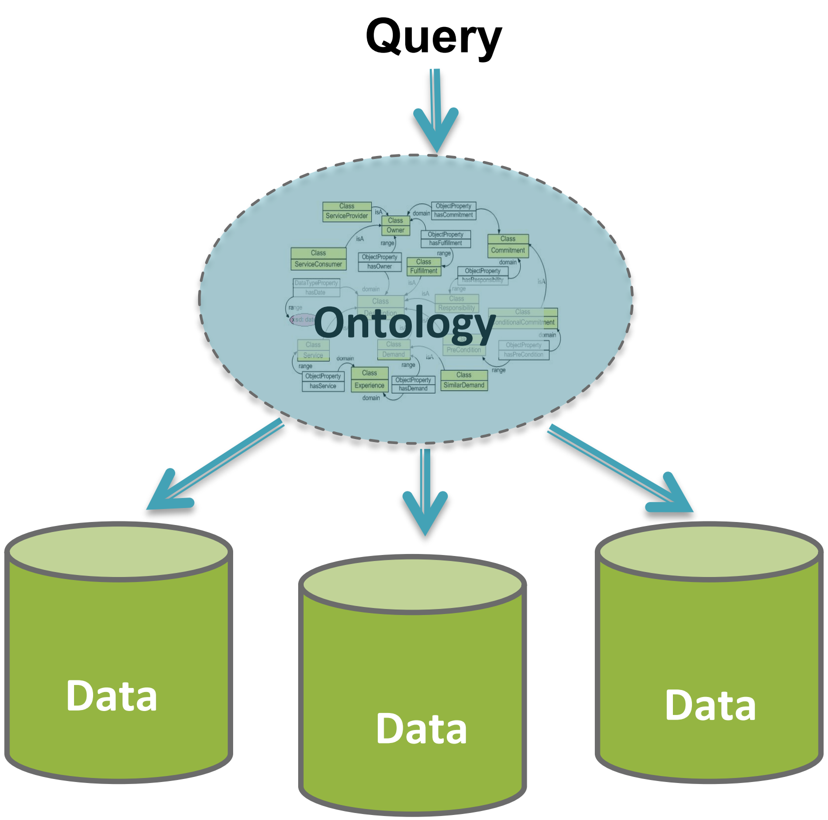
- Saturation / query rewriting engine for existential rules

  Graal http://graphik-team.github.io/graal/

- OBDA Systems (SPARQL + OWL 2 QL (DL-Lite) + GAV mappings + 1 RDBMS)
  OnTop https://ontop.inf.unibz.it/
  Mastro http://www.obdasystems.com/mastro

ONTOLOGY-BASED DATA ACCESS

Adding an ontological layer on top of data



1- **Enrich** the user vocabulary

→ **abstract** from a specific data storage

2 - **Infer** new facts, not explicitely stored,

→ **incomplete data** representation

3 – provide a **unified view** of heterogeneous sources

Making systems efficient requires to consider all the components in **interaction**