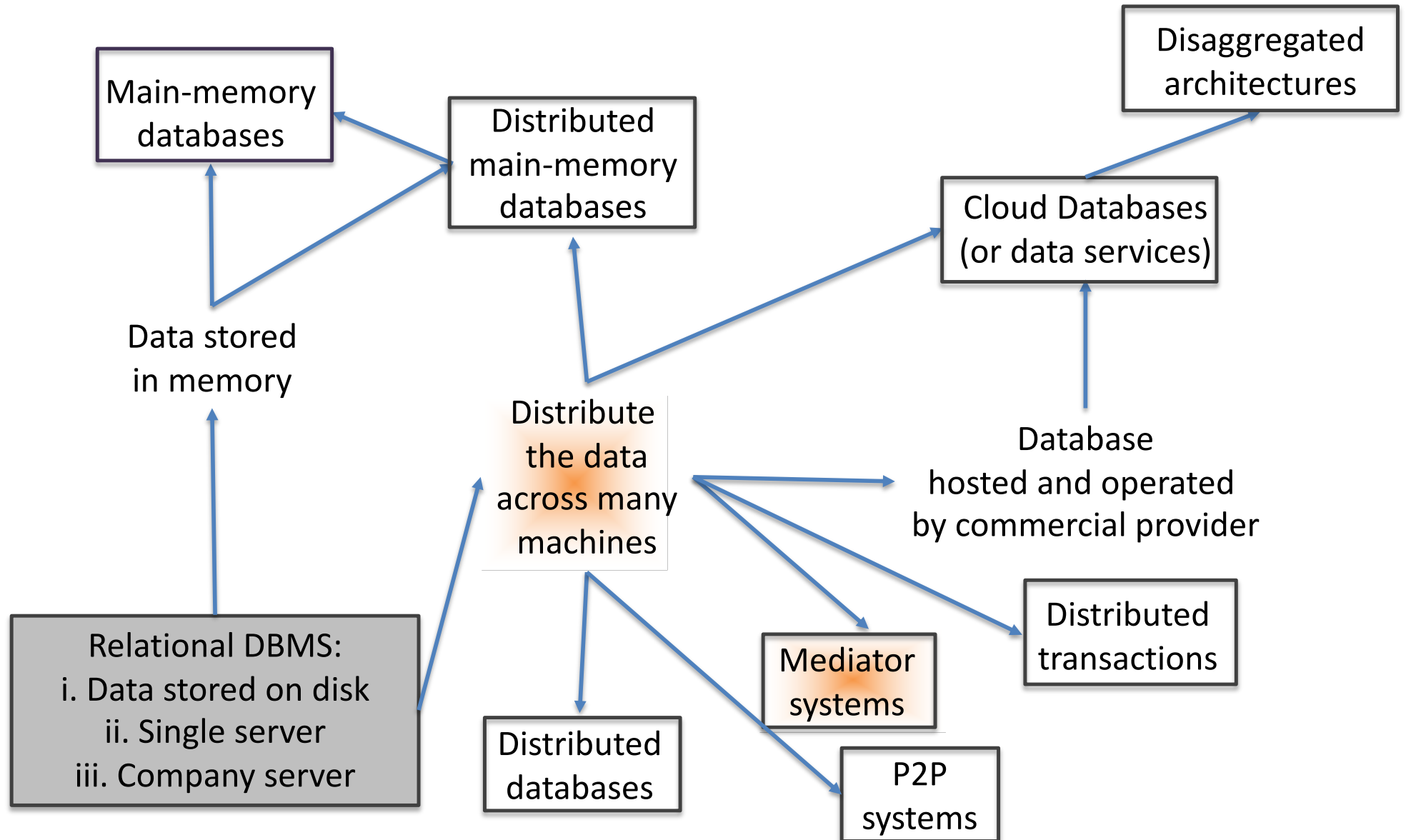
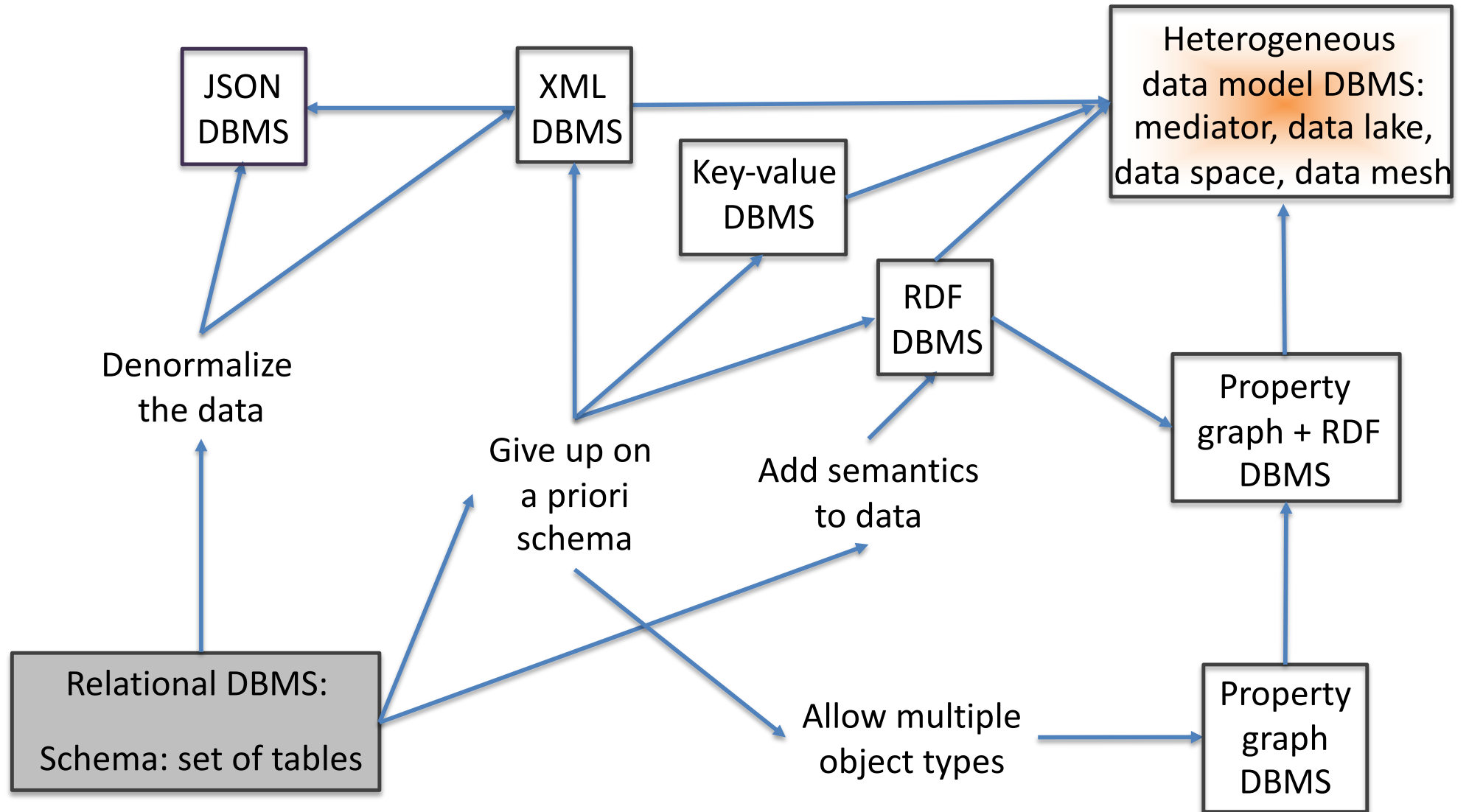


ARCHITECTURES FOR BIG DATA: HANDLING HETEROGENEITY

From databases to Big Data



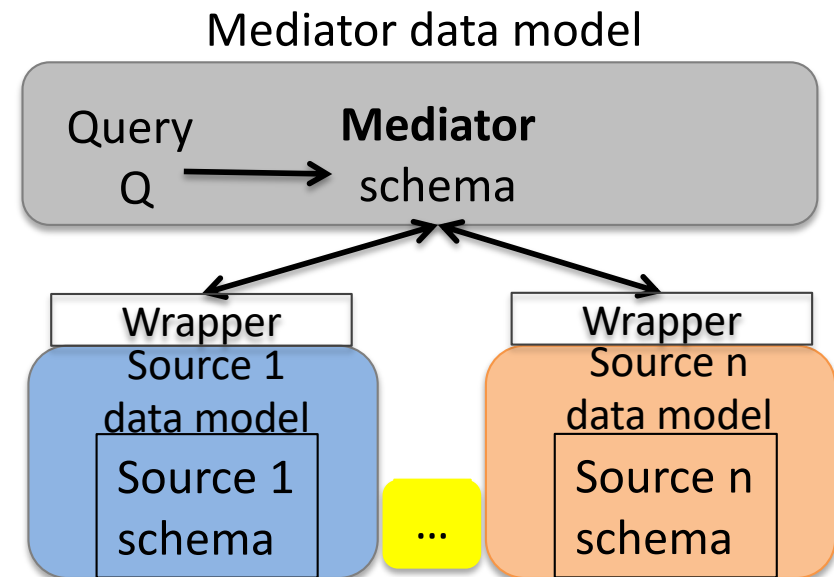
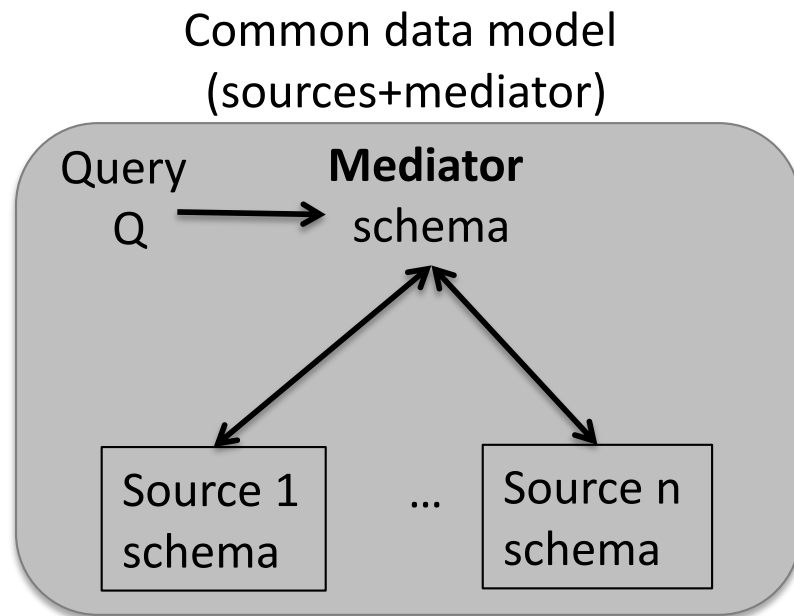
From databases to Big Data



HETEROGENEOUS DATA INTEGRATION: MEDIATOR SYSTEMS

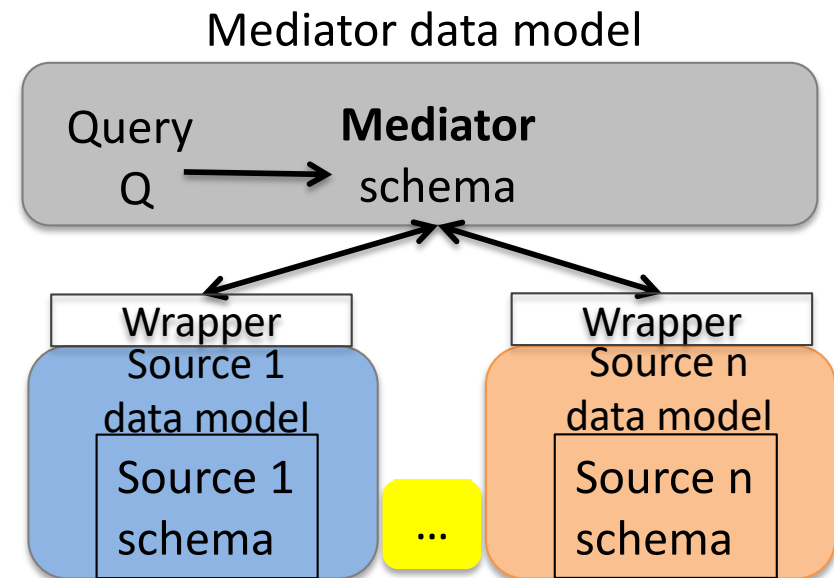
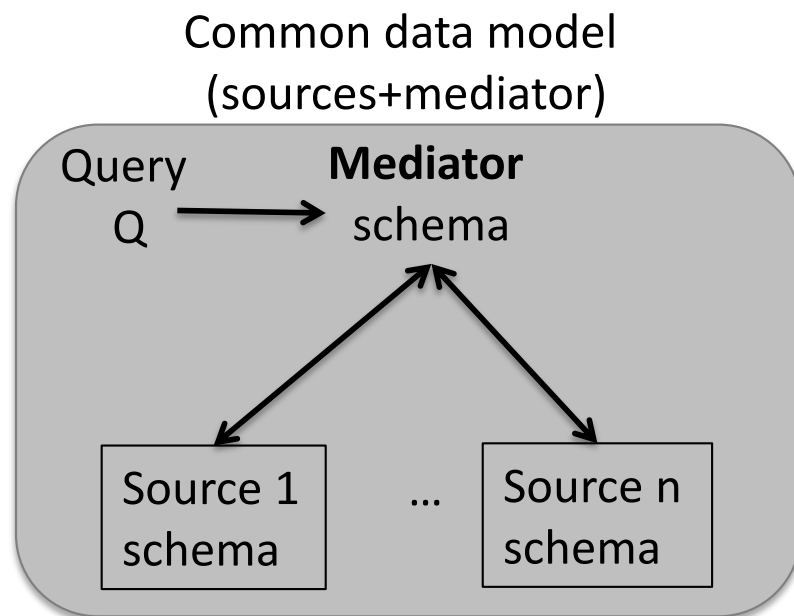
Mediator systems

- A set of **data sources**, each with: data model, query language, and schema (also called source schemas).
 - DM and QL may or may not differ across sources
- A **mediator** with its own DM, QL and mediator schema
 - Queries are asked against the mediator schema
- **Wrappers** interface the sources to the mediator's model



Mediator systems

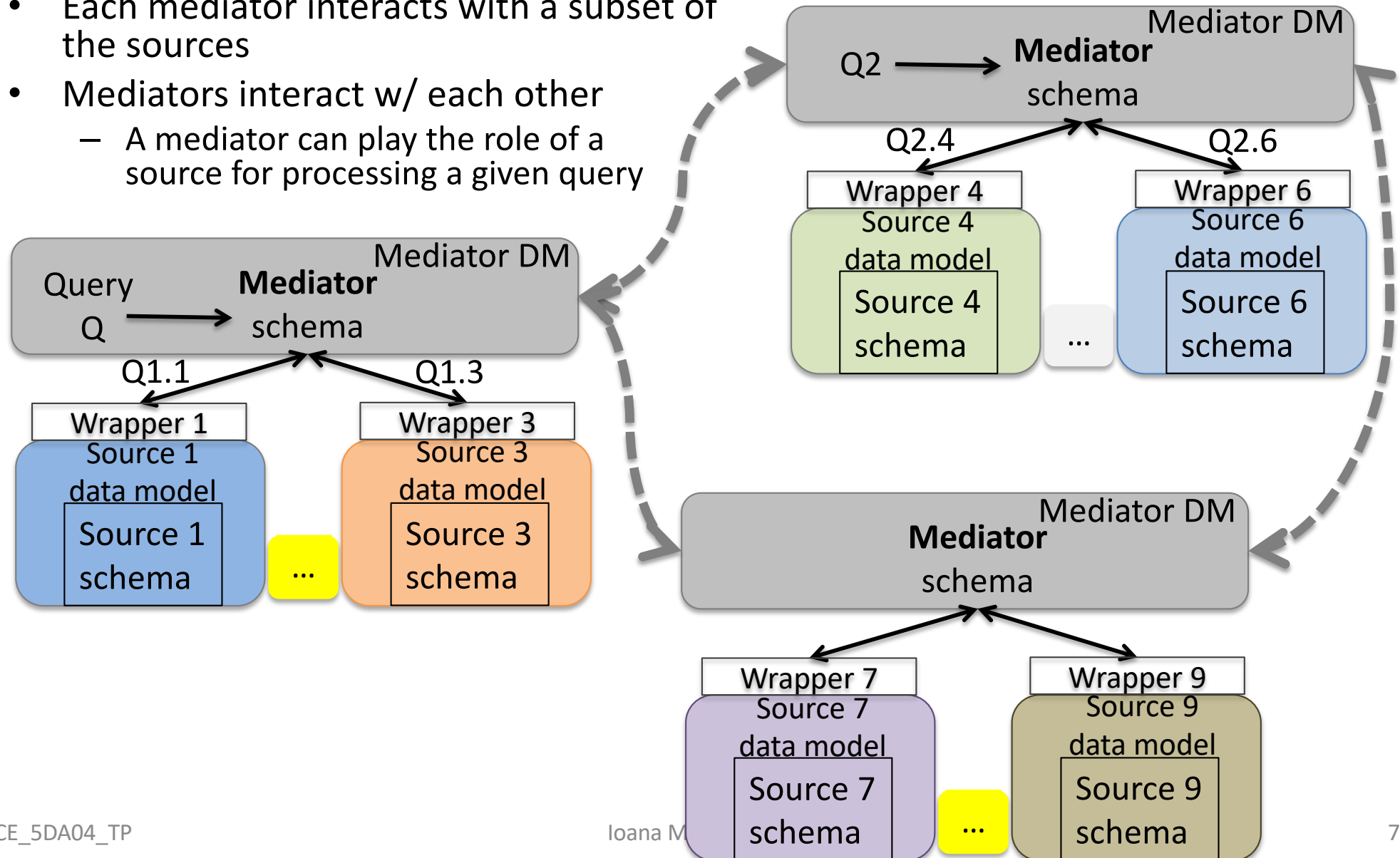
- A set of **data sources**, each with: data model, query language, and schema (also called source schemas).
 - DM and QL may differ across sources
- A **mediator** with its own DM, QL and mediator schema



- **ACID**: mostly read-only; **size**: small
- **Control**: Independent publishing; mediator-driven integration

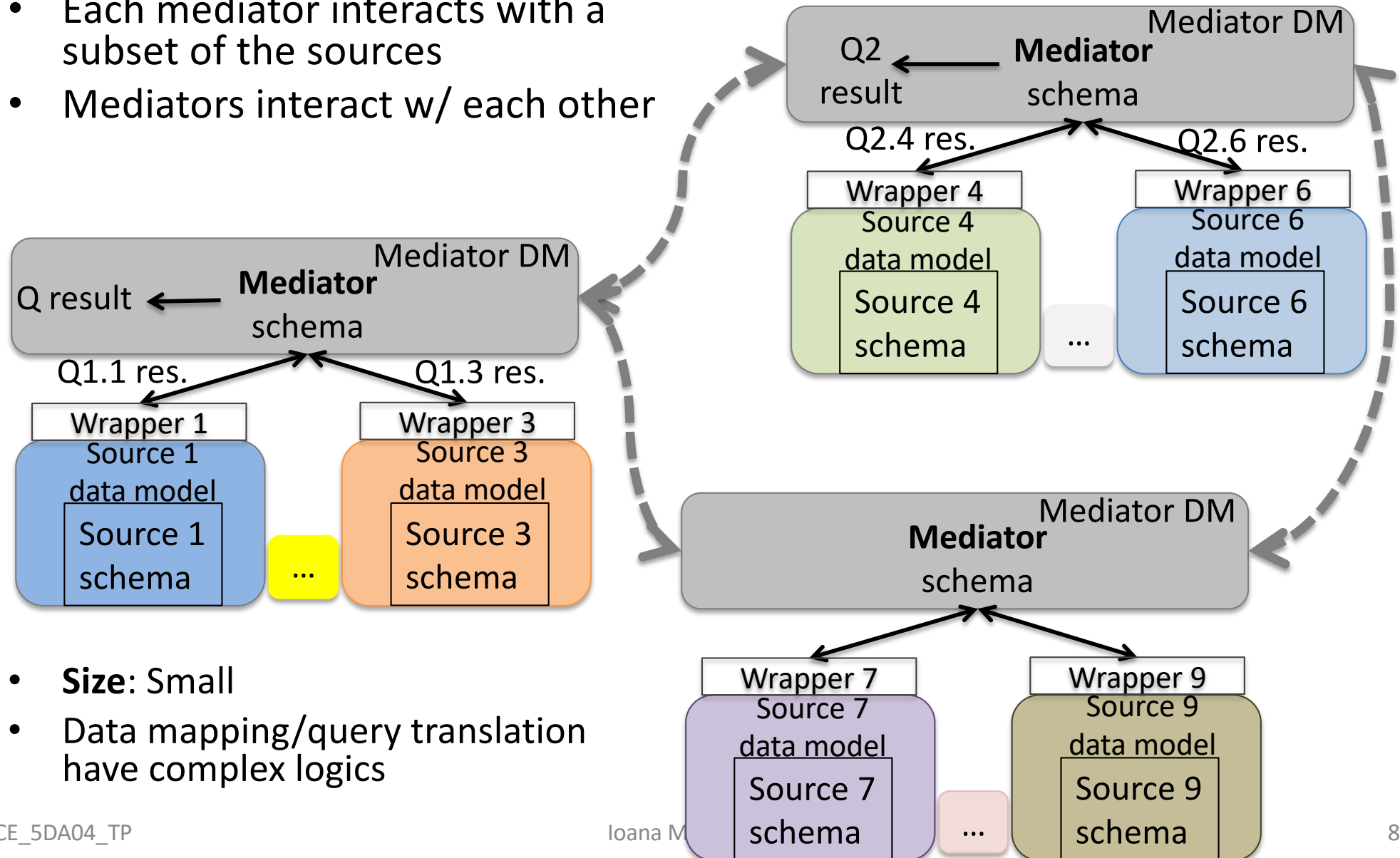
Many-mediator systems

- Each mediator interacts with a subset of the sources
- Mediators interact w/ each other
 - A mediator can play the role of a source for processing a given query



Many-mediator systems

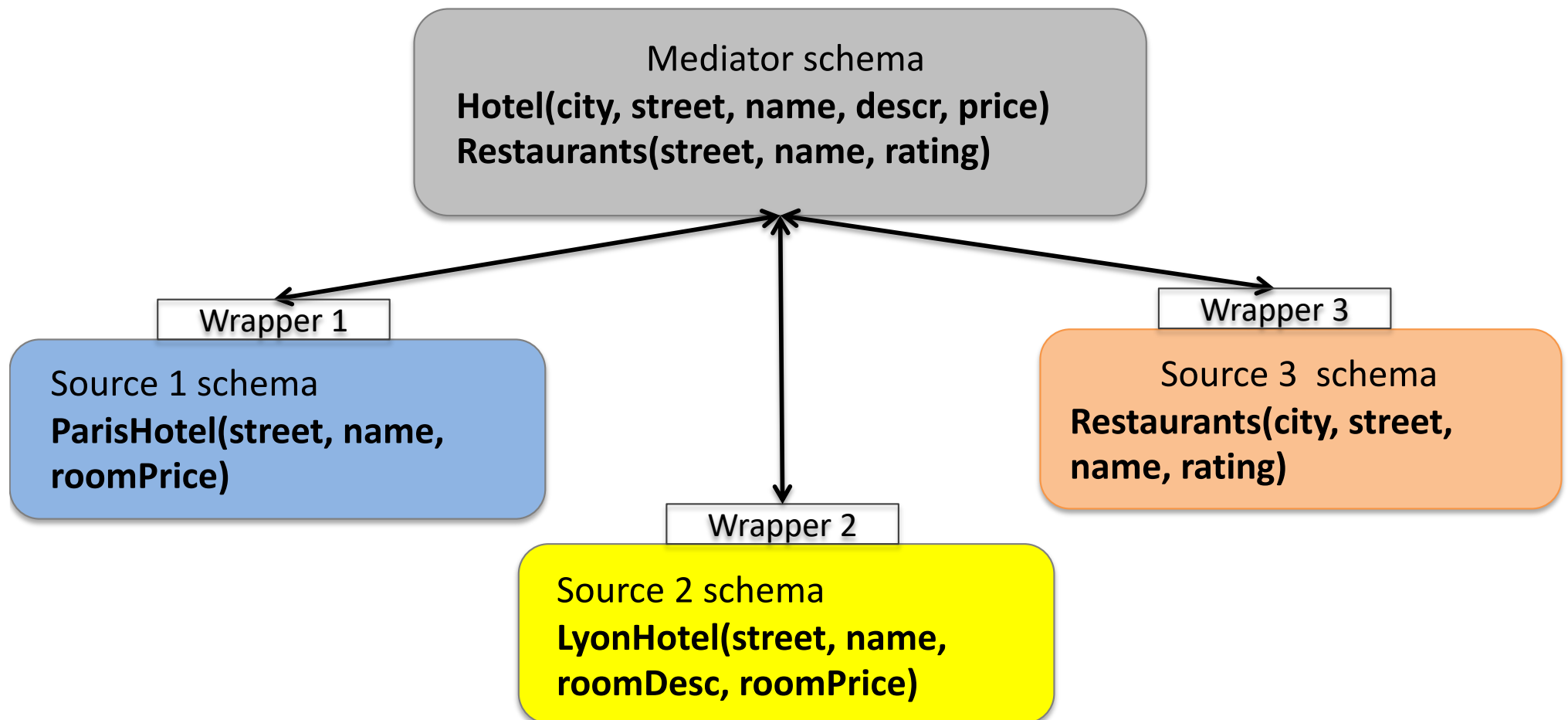
- Each mediator interacts with a subset of the sources
- Mediators interact w/ each other



- **Size:** Small
- Data mapping/query translation have complex logics

Connecting the source schemas to the global schema

- Sample scenario:



Connecting the source schemas to the global schema

- Data only exists in the sources.
- Applications only have access to, and only query, the mediator schema.
- How to **express the relation** between
 - the **mediator schema** accessible to applications, and
 - the **source schemas** reflecting the real data
 - so that a query over the mediator schema can be **automatically translated** into a query over the source schemas **?**
- Three approaches exist (see next)

Connecting the source schemas to the global schema: Global-as-view (GAV)

s1:ParisHotels(street, name, roomPrice)

s2:LyonHotel(street, name, roomDesc, roomPrice)

s3:Restaurant(city, street, name, rating)

Global: Hotel(city, street, name, descr, price),
Restaurant(city, street, name, rating)

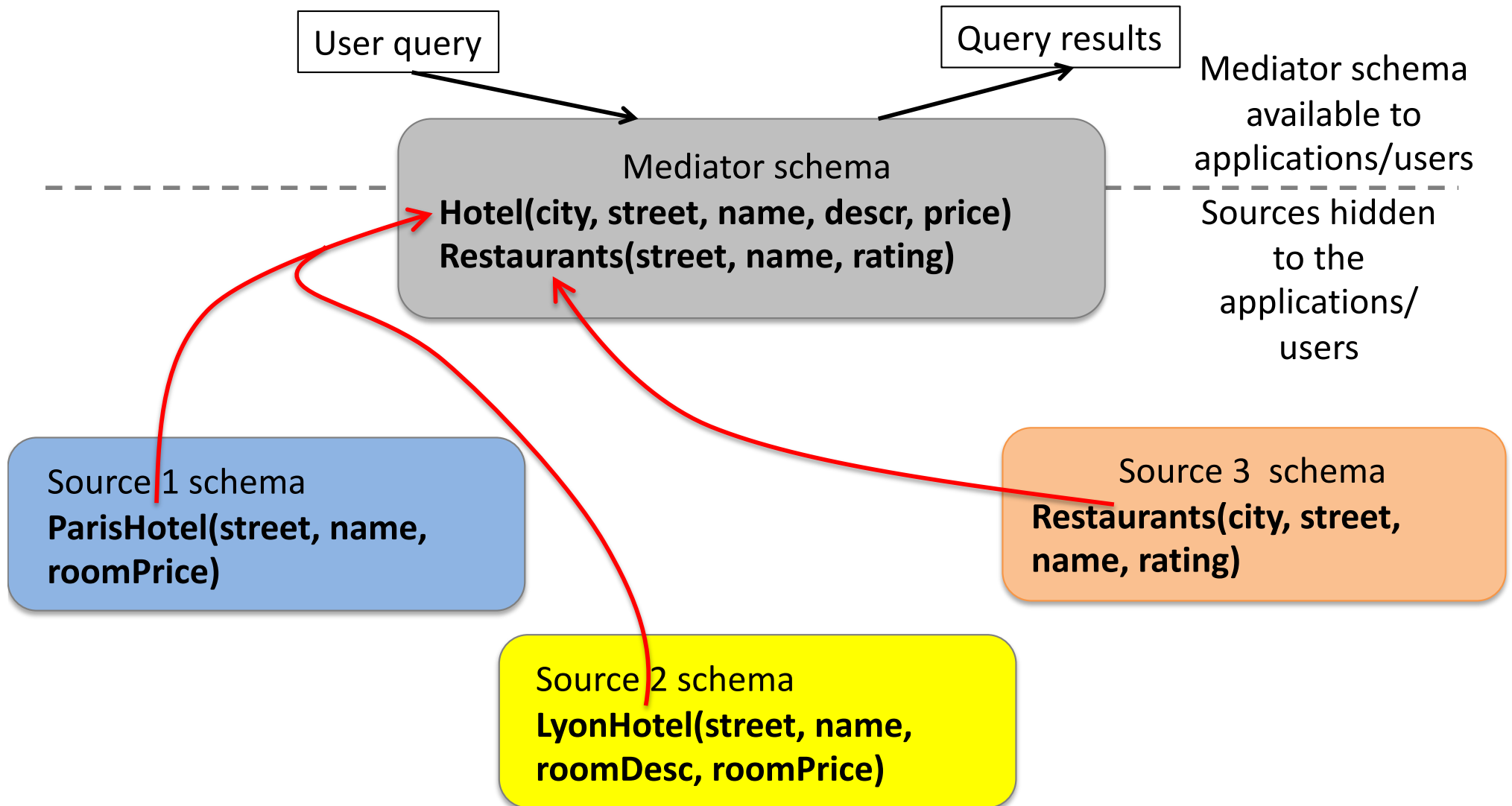
Defining **Hotel** as a view over the source schemas:

```
define view Hotel as
select 'Paris' as city, street, name, null as descr, roomPrice as price
from s1:ParisHotels
union all
select 'Lyon' as city, street, name, roomDesc as descr, price
from s2:LyonHotel
```

Defining **Restaurant** as a view over the source schemas:

```
define view Restaurant as select * from s3:Restaurant
```

Connecting the source schemas to the global schema: Global-as-View



Query processing in global-as-view (GAV)

```
define view Hotel as  
select 'Paris' as city, street, name, null as descr, roomPrice as price  
from s1:ParisHotels  
union all  
select 'Lyon' as city, street, name, roomDesc as descr, price  
from s2:LyonHotel
```

Query:

```
select * from Hotel where city='Paris' and price<200    becomes:
```

```
select * from (select 'Paris' as city... union... select 'Lyon' as city...)  
              where city='Paris' and price < 200    which becomes:
```

```
select * from (select 'Paris' as city...)  
              where city='Paris' and price < 200    which becomes:
```

```
select * from s1:ParisHotels where price < 200
```

Query processing in global-as-view (GAV)

define view **Hotel** as

```
select 'Paris' as city, street, name, null as roomDesc, roomPrice as price  
from s1:ParisHotels
```

union all

```
select 'Lyon' as city, street, name, descr as roomDesc, price from s2:LyonHotel
```

```
define view Restaurant as select * from s3:Restaurant
```

Query:

```
select h.street, r.rating from Hotels h, Restaurant r where h.city=r.city and  
r.city='Lyon' and and h.street=r.street and h.price<200 becomes:
```

```
select h.street, r.rating from (select 'Paris' as city... from s1:ParisHotels  
union all select 'Lyon' as city... from s2:LyonHotel) h, (select * from s3:Restaurant) r  
where h.city=r.city and r.city='Lyon' and h.street=r.street and h.price<200  
which becomes:
```

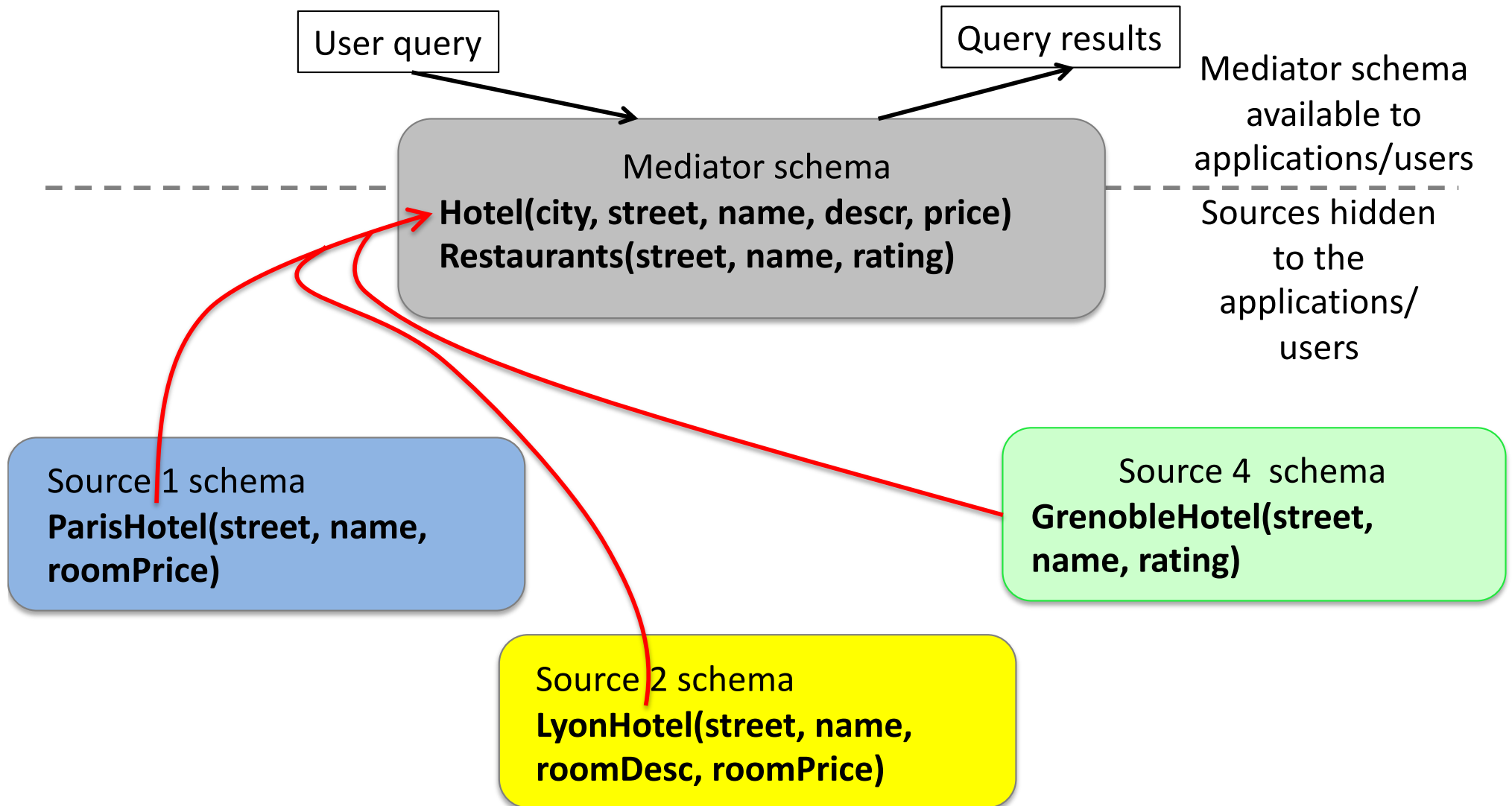
```
select h.street,r.rating from (select ... from s2:LyonHotel) h, s3:Restaurant r where  
r.city='Lyon' and h.street=r.street and h.price<200 which becomes:
```

```
select h.street, r.rating from s2:LyonHotel h, s3:Restaurant r where r.city='Lyon' and  
h.price<200 and h.street=r.street
```

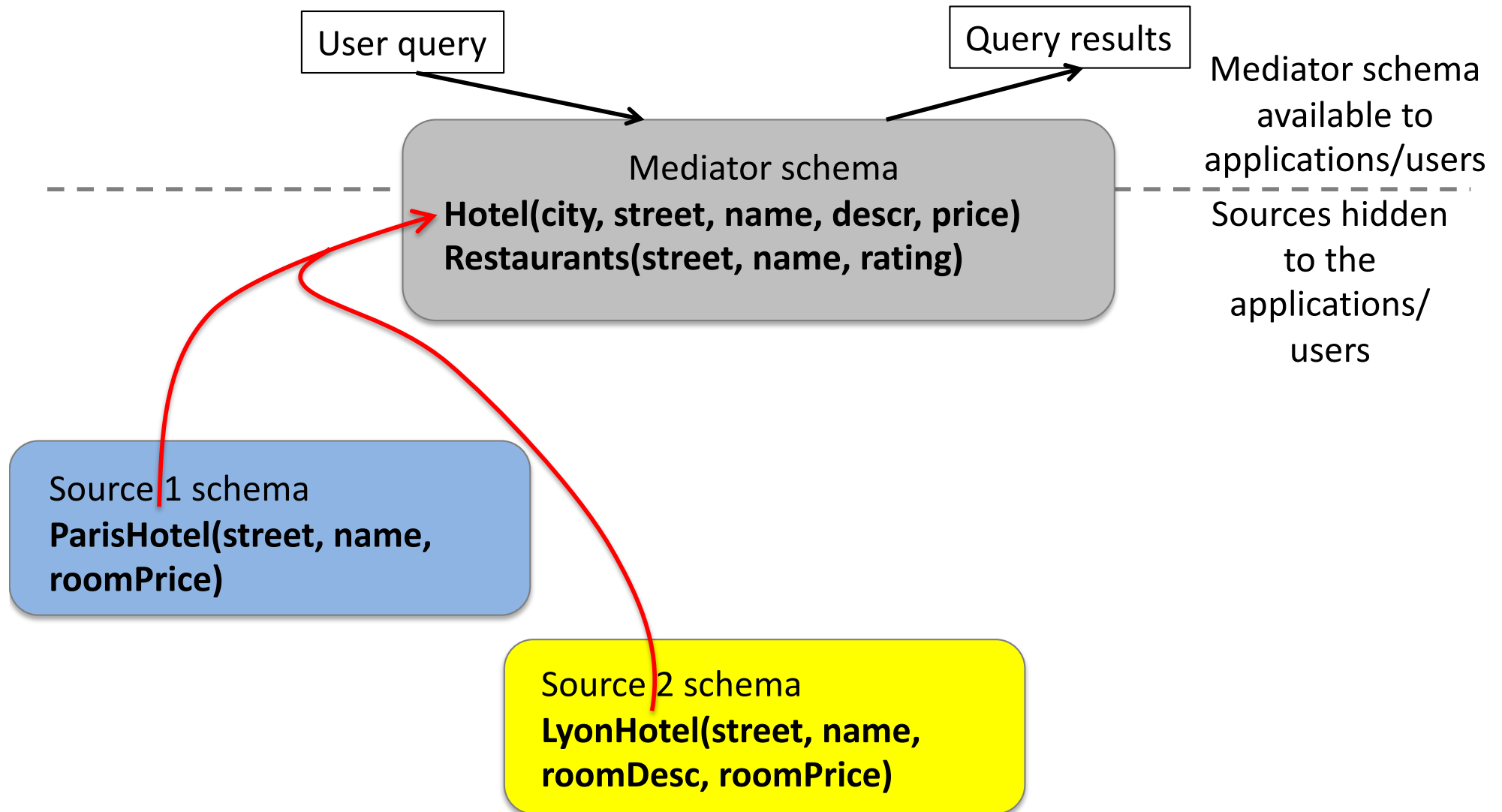
Concluding remarks on global-as-view (GAV)

- Query processing = **view unfolding**: replacing the view name with its definition
 - Just like queries over views in a centralized database
 - Heuristic: push as many operators (select, project, join; navigate...) on the sources as possible
- **Weakness**: changes in the data sources require changes of the global schema
 - In the worst case, all applications written based on this global schema need to be updated
 - Hard to maintain

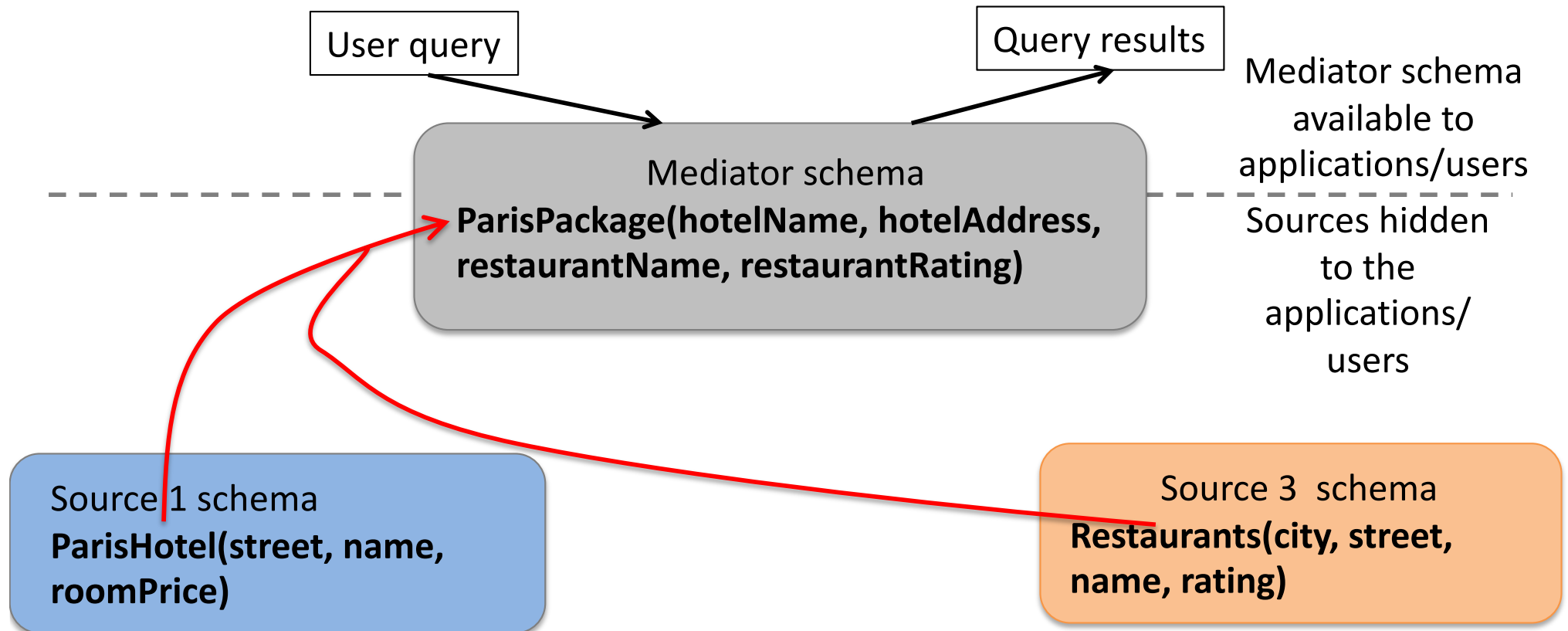
Global-as-View: Adding a new source



Global-as-View: Removing a source (1)



Global-as-View: Removing a source (2)



If **Source3.Restaurant** withdraws, the **ParisPackage** relation in the global schema becomes empty; applications cannot even access **Source1.ParisHotels**, even though they are still available.

Connecting the source schemas to the global schema: Local-as-view (LAV)

s1:ParisHotel(street, name, roomPrice)

s2:LyonHotel(street, name, roomDesc, roomPrice)

s3:Restaurant(city, street, name, rating)

Global: Hotel(city, street, name, descr, price), **Restaurant**(city, street, name, rating)

Defining **s1:ParisHotels** as a view over the global schema:

```
define view s1:ParisHotels as  
select street, name, price as roomPrice  
from Hotel where city='Paris'
```

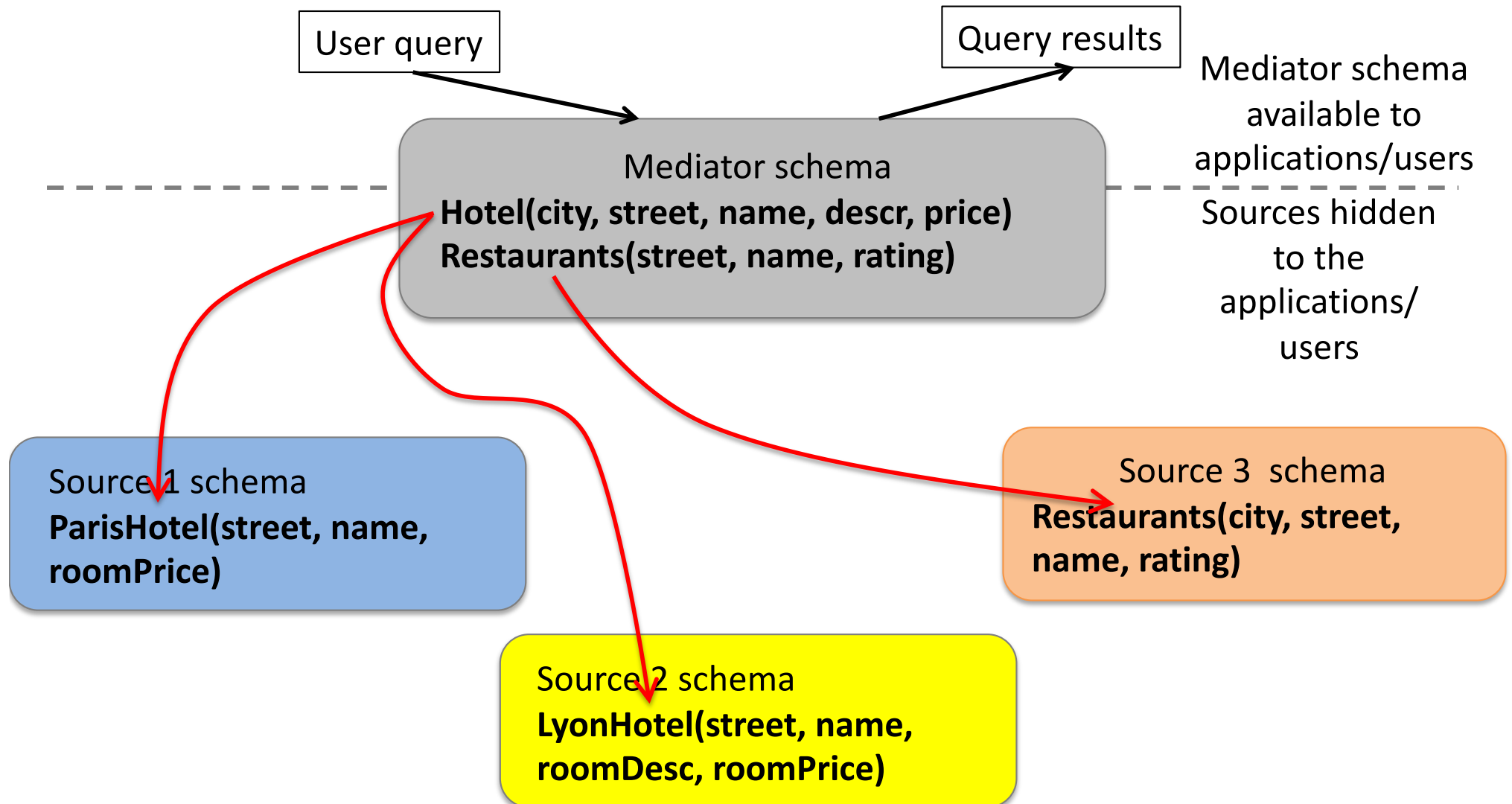
Defining **s2:LyonHotel** as a view over the global schema:

```
define view s2:LyonHotel as  
select street, name, descr as roomDesc, price as roomPrice  
from Hotel where city='Lyon'
```

Defining **s3:Restaurant** as a view over the global schema:

```
define view s3:Restaurant as  
select * from Restaurant
```

Connecting the source schemas to the global schema: Local-as-View



GAV and LAV have different expressive power

- **Some GAV scenarios cannot be expressed in LAV**
- Example:

create view **ParisPackage** as
select ph.name as hotelName, ph.street as hotelAddress,
r.name as restaurantName, r.rating as restaurantRating
from s1:ParisHotel ph, s3:Restaurants r
where **r.city='Paris' and r.street=ph.street**

- The view only contains (hotel, restaurant) pairs that are *on the same street* in Paris
- Not possible to express this with LAV mappings
 - LAV describes each source *individually* w.r.t. the global schema
 - Not in correlation with data available in *other sources* !

GAV and LAV have different expressive power

- There exist **LAV scenarios that cannot be expressed in GAV**
- Example: s3:MHotels(city, street, name, price) only has data about Marseille hotels, s4:WHotels(city, street, name price) has only data about Wien hotels
 - Assume Hotels is defined as:
`select * from Mhotels union all select * from WHotels`
 - A query about hotels in Rome will also be sent to s3 and s4, although it will bring no results
 - LAV query processing avoids this (see next)

GAV and LAV have different expressive power

- There exist **GAV** scenarios that cannot be expressed in LAV
- Example:

```
create view ParisPackage as
select ph.name as hotelName, ph.street as hotelAddress, r.name as
restaurantName, r.rating as restaurantRating
from s1:ParisHotel ph, s3:Restaurants r
where r.city='Paris' and r.street=ph.street
```

- The closest we can do is define s1.ParisHotel and s3.Restaurants *each* as a projection over ParisPackage
- But this changes the semantics of ParisPackage:
 - It does not express that *only Paris restaurants* are in ParisPackage
 - Not possible to express that only (hotel, restaurants) *on the same street* are available through the integration system
 - ParisPackage becomes the cartesian product of ParisHotel with all restaurants...

Query processing in Local-as-View (LAV)

Simple scenario 1

```
define view s1:ParisHotels as  
select street, name, price as roomPrice  
from Hotel where city='Paris'
```

Query:

```
select street, name from Hotel
```

No *equivalent* rewriting exists.

- We should not use a rewriting that has some **wrong** answers (not contained in those of the query)!
- We should aim for *contained rewritings*
- Better: *Maximally contained rewritings, such as:*
Select street, name from s1:ParisHotels

Query processing in Local-as-View (LAV)

Simple scenario 2

```
define view s1:ParisHotels as  
select street, name, price as roomPrice  
from Hotel where city='Paris'
```

```
define view s7:CheapHotels as  
select street, name  
from Hotel where price<80
```

Query:

```
select street, name from Hotel
```

S1:ParisHotels has some useful answers. So does s7:CheapHotels.
They may overlap (or not); no way of knowing.

Maximally contained rewriting:

```
select street, name from s1:ParisHotels union  
select street, name from s7:CheapHotels
```

Query processing in Local-as-View (LAV)

Simple scenario 3

```
define view s7:CheapHotels as  
select street, name  
from Hotel where price<80
```

Query:

```
select street, name from Hotel where city='Paris'
```

Maximally contained rewriting:

None exists! **X**

Query processing in Local-as-View (LAV)

Simple scenario 4

```
define view s7:CheapHotels as  
select street, name, city  
from Hotel where price<80
```

Query:

```
select street, name from Hotel where city='Paris'
```

Maximally contained rewriting:

```
select street, name from s7:CheapHotels where city='Paris'
```

Query processing in Local-as-View (LAV)

Simple scenario 5

```
define view s7:CheapHotels as  
select street, name  
from Hotel where price<80
```

Query:

```
select street, name, price from Hotel where city='Paris'
```

Maximally contained rewriting:

None exists! **X**

Query processing in Local-as-View (LAV)

Observation

A view may fail to be usable to answer a query if :

- The query applies a restriction the view does not apply and the rewriting cannot apply it either
- And/or, the view fails to project (store) an attribute that the query needs

define view **s7:CheapHotels** as
select street, name
from Hotel where price<80

Query:

select street, name, price
from Hotel where city='Paris'

define view **s7:CheapHotels** as
select street, name
from Hotel where price<80

Query:

select street, name
from Hotel where city='Paris'

Query processing in Local-as-View (LAV)

```
define view s1:ParisHotels as  
select street, name, price as roomPrice  
from Hotel where city='Paris'
```

```
define view s2:LyonHotel as  
select street, name, descr as roomDesc, price as roomPrice from Hotel  
where city='Lyon'
```

```
define view s3:Restaurant as  
select * from Restaurant
```

Query:

```
select h.street, h.price, r.rating  
from Hotel h, Restaurant r  
where r.city=h.city and h.street=r.street
```

Query processing in Local-as-View (LAV)

```
define view s1:ParisHotels as  
select street, name, price as roomPrice  
from Hotel where city='Paris'
```

Step 1: identify
potentially useful
views

```
define view s2:LyonHotel as  
select street, name, descr as roomDesc, price as roomPrice from Hotel  
where city='Lyon'
```

```
define view s3:Restaurant as  
select * from Restaurant
```

Query:

```
select h.street, h.price, r.rating from Hotel h, Restaurant r where  
r.city=h.city and h.street=r.street
```

Query processing in Local-as-View (LAV)

Query:

```
select h.street, h.price, r.rating from Hotel h, Restaurant r
where r.city=h.city and h.street=r.street
```

Step 2: generate **view combinations** that may be used to answer the query (one view per table):

s1:ParisHotels and s3:Restaurant

s2:LyonHotels and s3:Restaurant

Step 3: for each view combination and each view, check:

- If the view returns the attributes we need:
 - Those returned by the query, *and*
 - Those on which possible query joins are based
- If the view selections (if any) are compatible with those of the query

If one condition is not met, discard the view combination.

```
define view s1:ParisHotels as
select street, name, price as roomPrice
from Hotel where city='Paris'
```

The query needs:

- street, price, rating (returned): the view provides them
- city and street for the join: street is provided, city is not (but it is a constant, thus known)

The view has a selection on the city which the query does not have → The view provides *part* of the data needed by the query. The view selection is compatible with the query.

The view s1:ParisHotels is OK.

```
define view s3:Restaurant as select * from
Restaurant
```

The view s3:Restaurants is OK.

The view combination s1:ParisHotels, s3:Restaurants is OK provided that Restaurant.city is set to Paris.

Query processing in Local-as-View (LAV)

Query:

```
select h.street, h.price, r.rating from Hotel h, Restaurant r
where r.city=h.city and h.street=r.street
```

Step 2: generate **view combinations** that may be used to answer the query (one view per query table):

s1:ParisHotels and s3:Restaurant
s2:LyonHotels and s3:Restaurant

Step 3: for each view combination and each view, check:

[...]

If one condition is not met, discard the view combination.

Step 4: for each view combination, add the necessary joins among the views, possibly selections and projections → rewriting

Query rewriting using s1:ParisHotels and s3:Restaurant:

```
select h.street, h.price, r.rating
from s1:ParisHotels h and s3:Restaurant r
where r.city='Paris' and h.street=r.street
```

This is a *partial* rewriting, and so is:

Query rewriting using s2:LyonHotel and s3:Restaurant:

```
select h.street, h.price, r.rating
from s2:LyonHotels h and s3:Restaurant r
where r.city='Lyon' and h.street=r.street
```

Query processing in Local-as-View (LAV)

Query:

```
select h.street, h.price, r.rating from Hotel h, Restaurant r
where r.city=h.city and h.street=r.street
```

Step 2: generate **view combinations** that may be used to answer the query (one view per query table):

s1:ParisHotels and s3:Restaurant

s2:LyonHotels and s3:Restaurant

Step 3: for each view combination and each view, check:

[...]

If one condition is not met, discard the view combination.

Step 4: for each view combination, add the necessary joins among the views, possibly selections and projections → rewriting

Step 5: return the union of the rewritings thus obtained

Full query rewriting:

```
select h.street, h.price, r.rating
from s1:ParisHotels h and s3:Restaurant r
where r.city='Paris' and h.street=r.street
union all
select h.street, h.price, r.rating
from s2:LyonHotel h and s3:Restaurant r
where r.city='Lyon' and h.street=r.street
```

Query processing in Local-as-View (LAV)

```
define view s1:ParisHotels as... from Hotel where city='Paris'  
define view s2:LyonHotel as... from Hotel where city='Lyon'  
define view s3:Restaurant as select * from Restaurant
```

Query:

```
select h.street, h.price, r.rating  
from Hotel h, Restaurant r  
where r.city=h.city and h.street=r.street
```

Rewriting of the query using the views:

```
select h1.street, h1.price, r3.rating  
from s1:ParisHotels h1, s3:Restaurant r3  
where h1.city=r3.city and h1.street=r3.street
```

union all

```
select h2.street, h2.price, r3.rating  
from s2:LyonHotels h2, s3:Restaurant r3  
where h2.city=r3.city and h2.street=r3.street
```

Concluding remarks on Local-as-View (LAV)

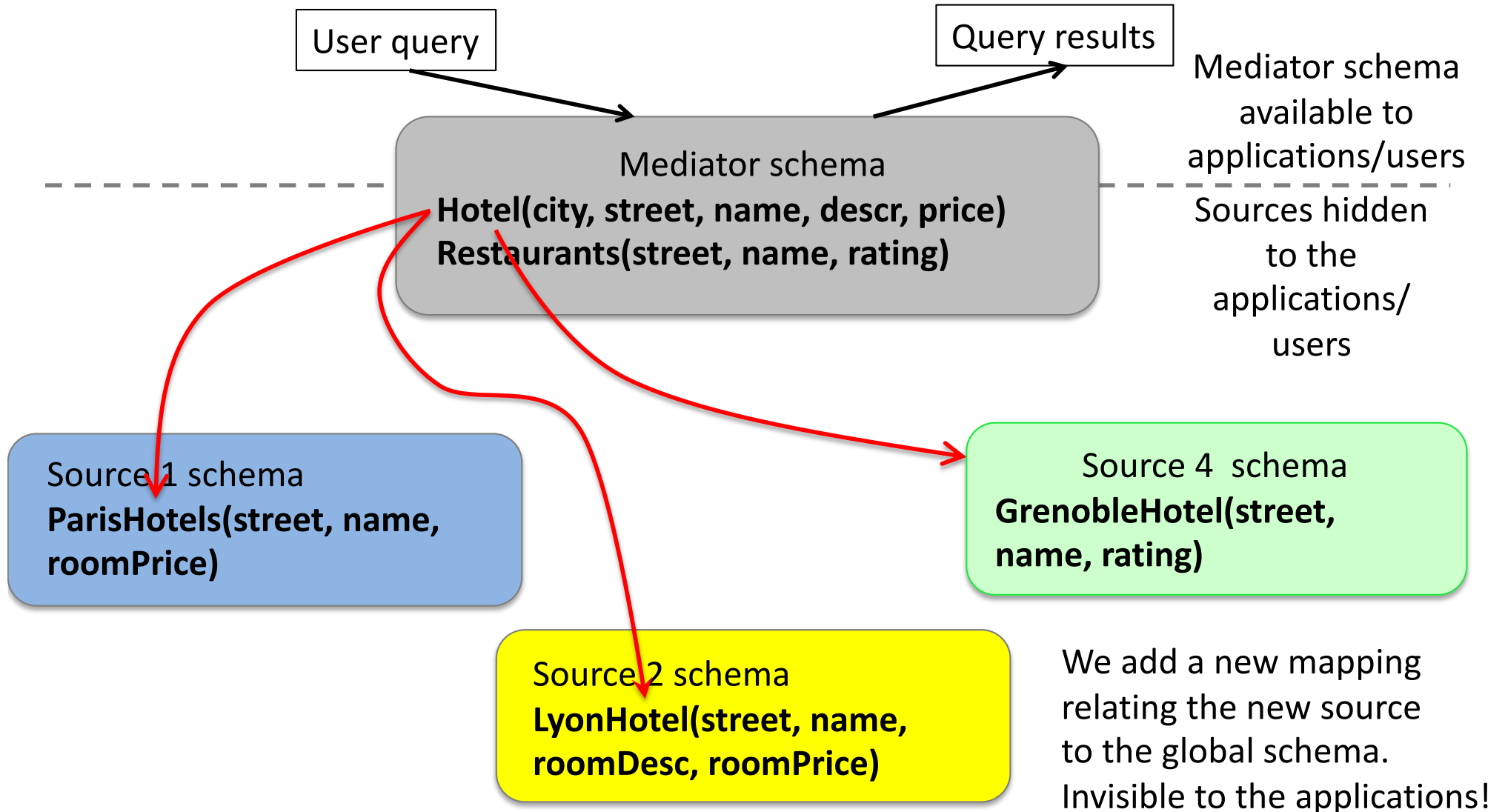
Query processing

- The problem of finding all rewritings given the source and global schemas and the view definitions = **view-based query rewriting**, NP-hard in the size of the (schema+view definitions).
 - These are often much smaller than the data

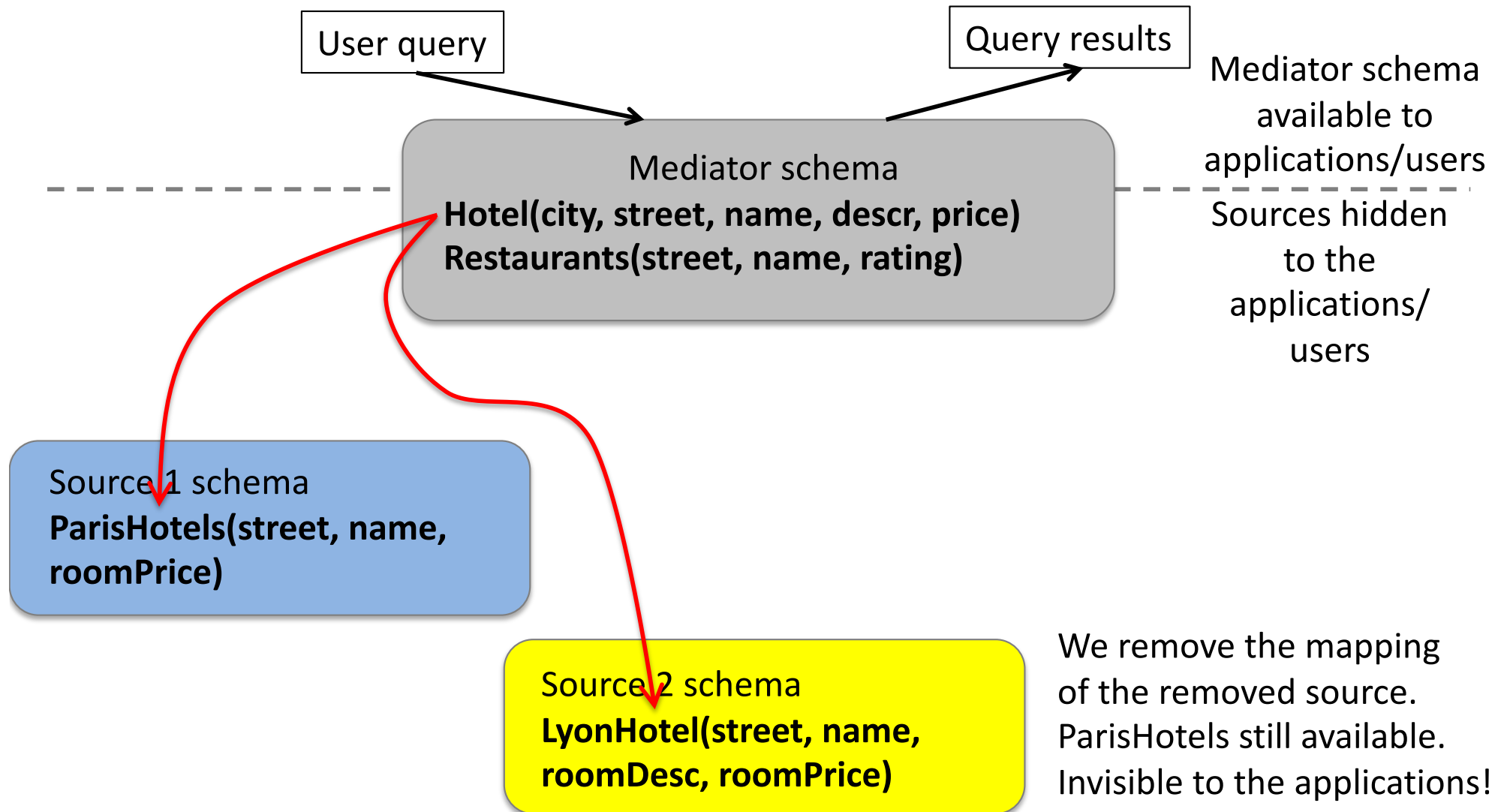
The schema definition is **more robust**:

- One can independently add/remove sources from the system without the global schema being affected at all (see next)
- Thus, no application needs to be aware of the changes in the schema

Local-as-View: adding a new source



Local-as-View: Removing a source



Connecting the source schemas to the global schema: Global-Local-as-View (GLAV)

Generalizes both GAV and LAV

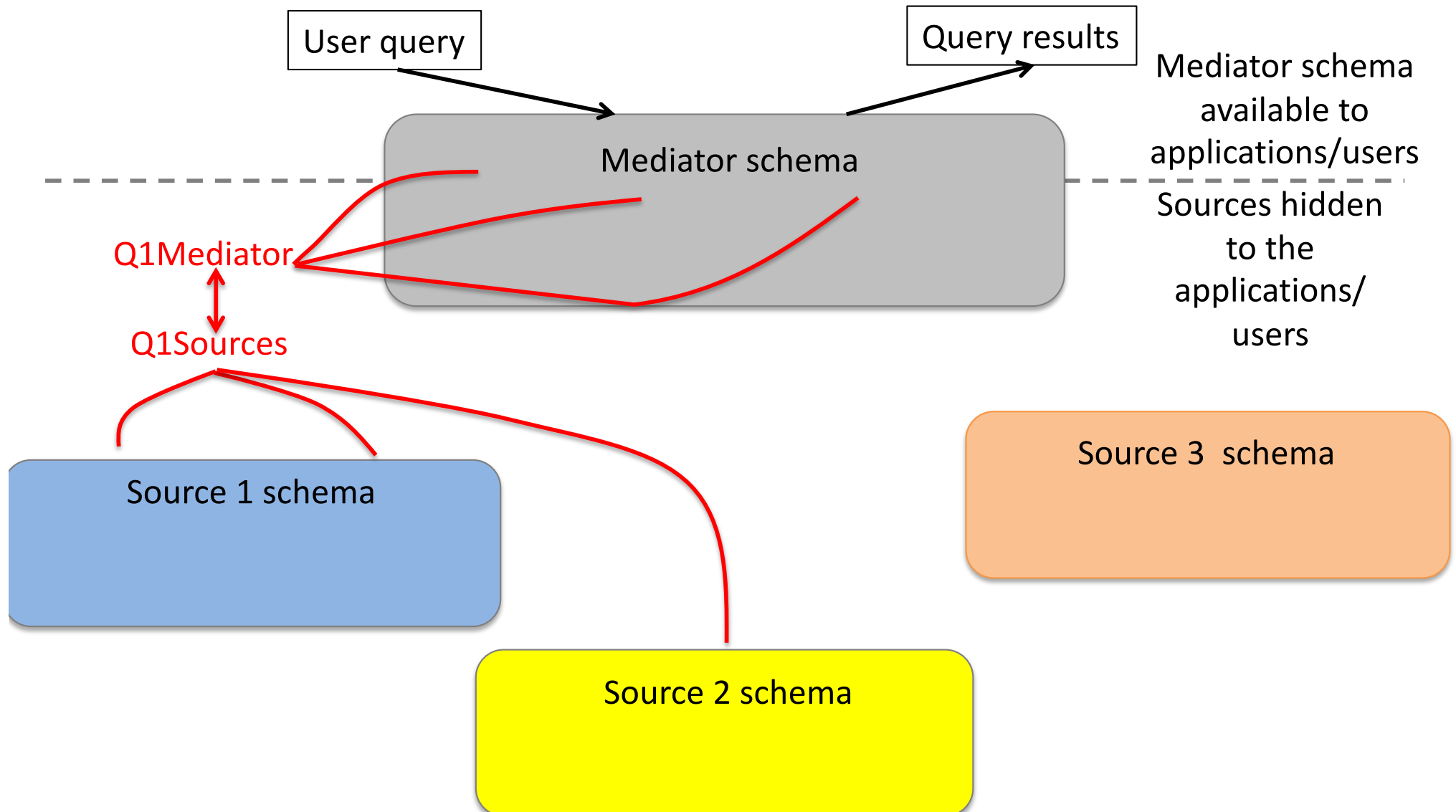
1 mapping = 1 pair (query over 1 or several sources schemas,
query over the mediator schema)

$$\begin{aligned} Q1\text{Mediator}(m:r1, m:r2, m:r3, \dots) &\leftrightarrow Q1\text{Sources}(s1:t1, s2:t1, \dots) \\ Q2\text{Mediator}(m:r1, m:r2, m:r3, \dots) &\leftrightarrow Q2\text{Sources}(s1:t1, s2:t1, \dots) \\ Q3\text{Mediator}(m:r1, m:r2, m:r3, \dots) &\leftrightarrow Q3\text{Sources}(s1:t1, s2:t1, \dots) \end{aligned}$$

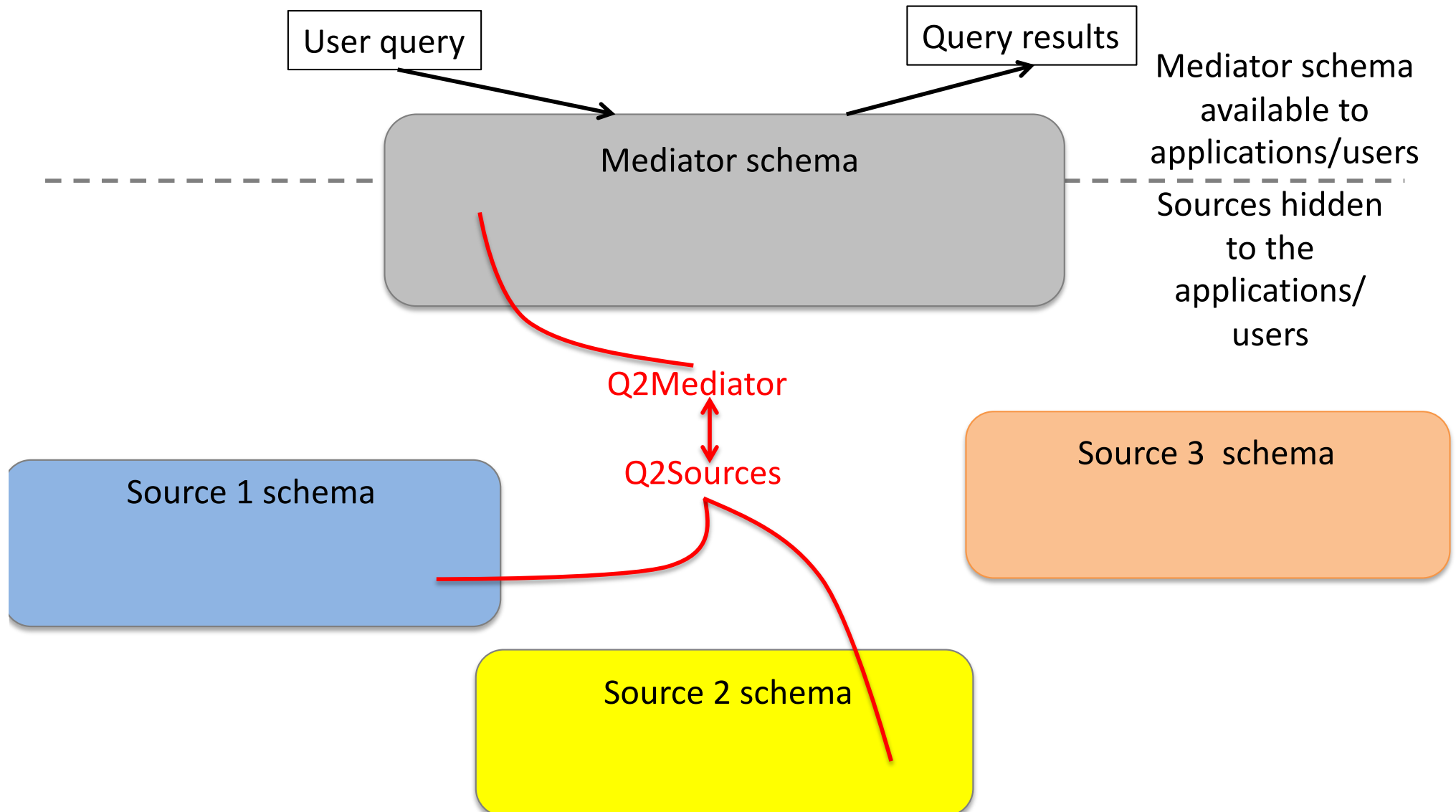
Semantics: *there is a tuple in $Q_i\text{Mediator}(\dots)$ for each result of $Q_i\text{Sources}(\dots)$*

- A GAV mapping is a particular case of GLAV mapping where QMediator is exactly one mediator relation
- A LAV mapping is a particular case of GLAV mapping where QSources is exactly one source relation

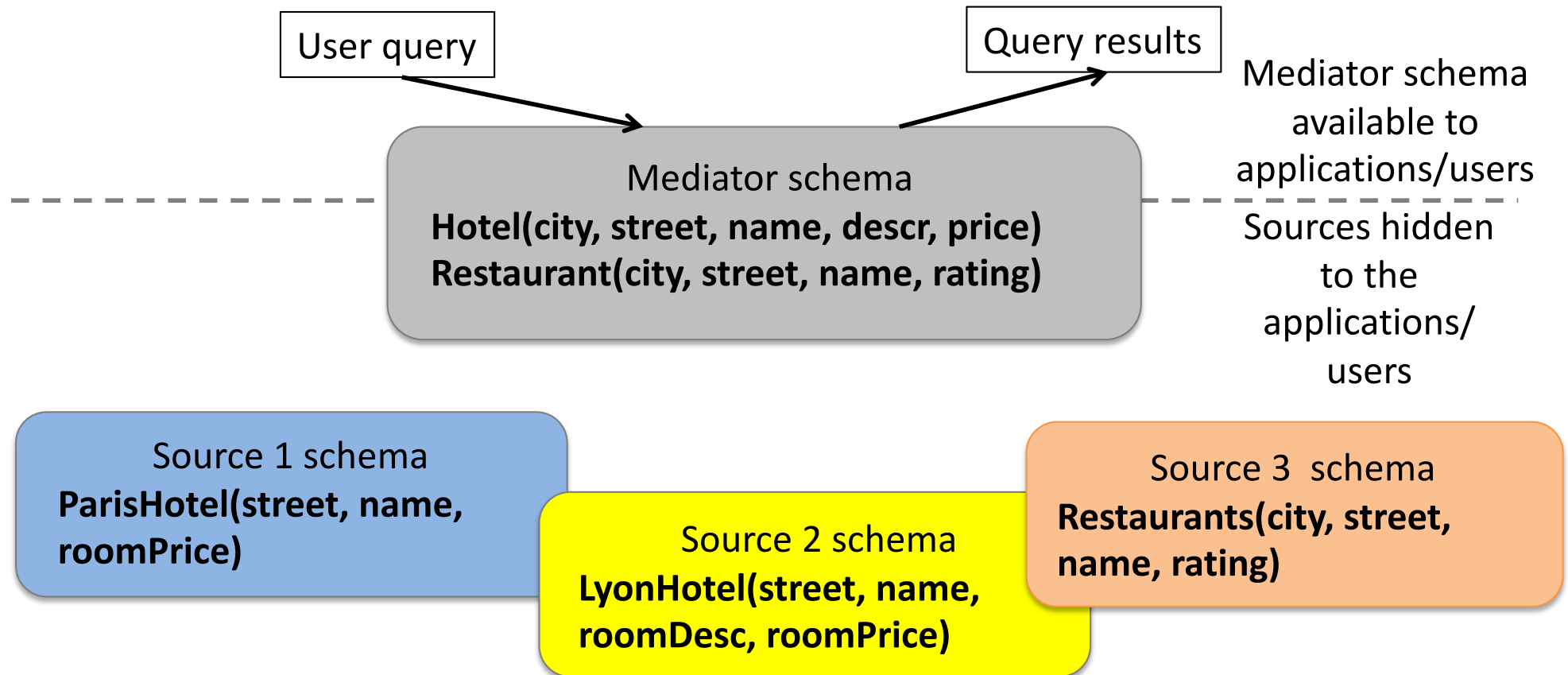
Connecting the source schemas to the global schema: Global-Local-as-View (GLAV)



Connecting the source schemas to the global schema: Global-Local-as-View (GLAV)



Global-Local-as-View: example

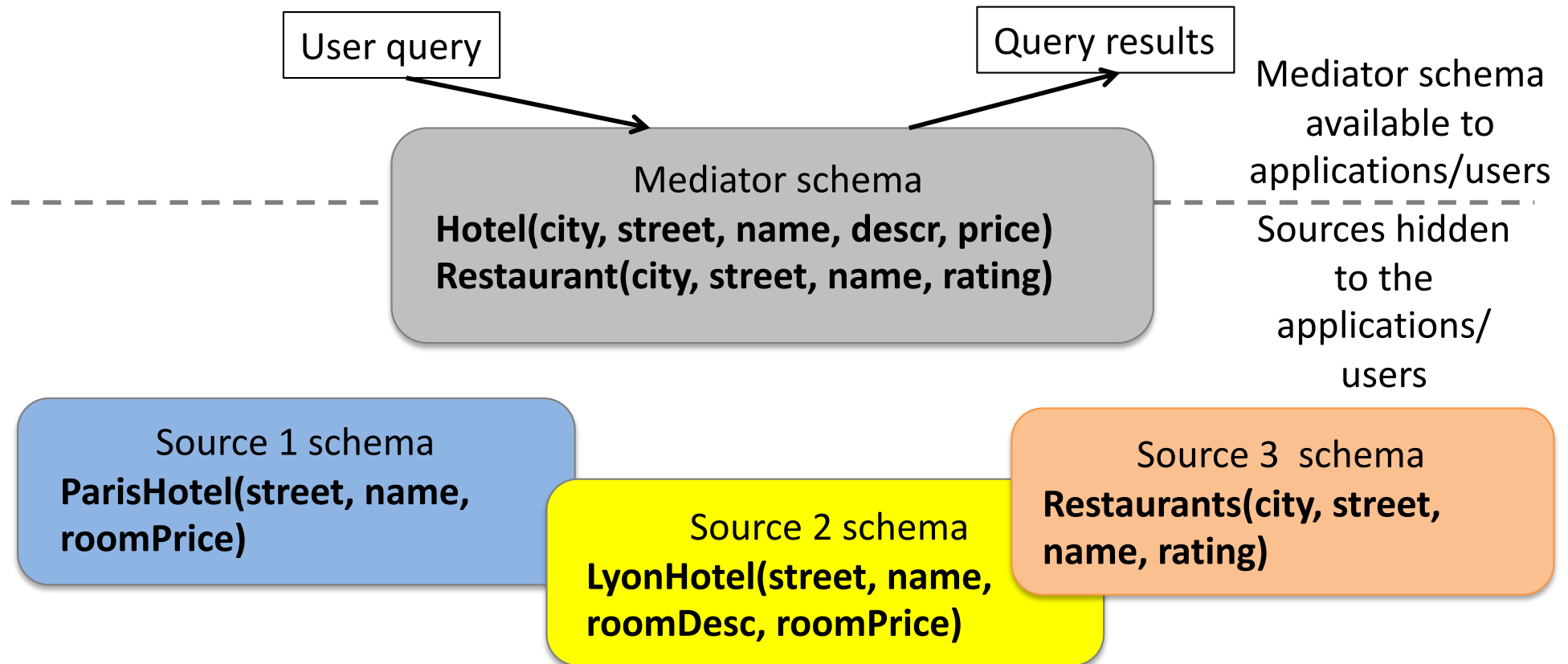


Previous LAV mapping of Source 1:

Q1Mediator: `select street, name, price as roomPrice from Hotel where city='Paris'`

Q1Sources: `select * from ParisHotel`

Global-Local-as-View: example



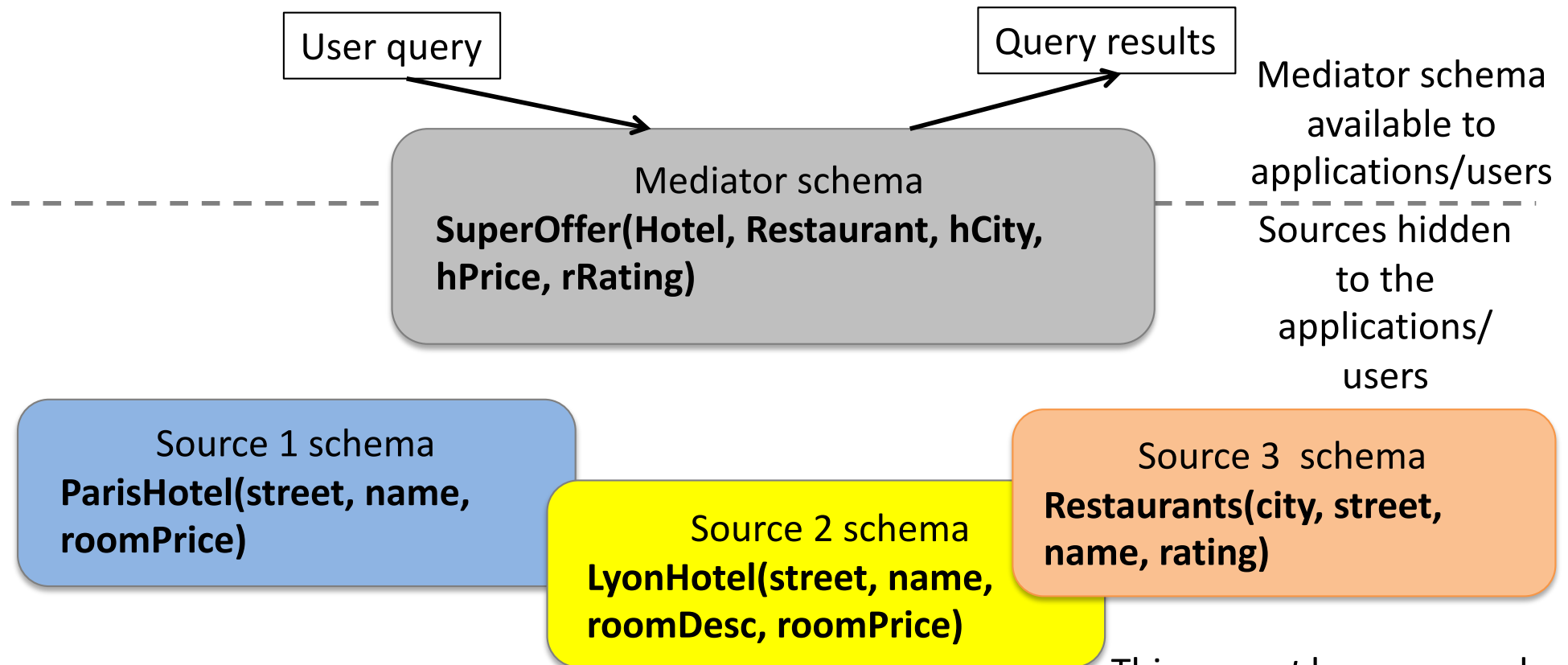
Previous GAV mapping of Hotel:

Q2Mediator: `select * from Hotel`

Q2Sources: `select 'Paris' as city, street, name, null as descr, roomPrice as price from ParisHotel
union`

`select 'Lyon' as city, street, name, roomDesc as descr, roomPrice as price from LyonHotel`

Global-Local-as-View: example



New GLAV mapping:

Q3Mediator: select * from SuperOffer where hCity='Lyon'

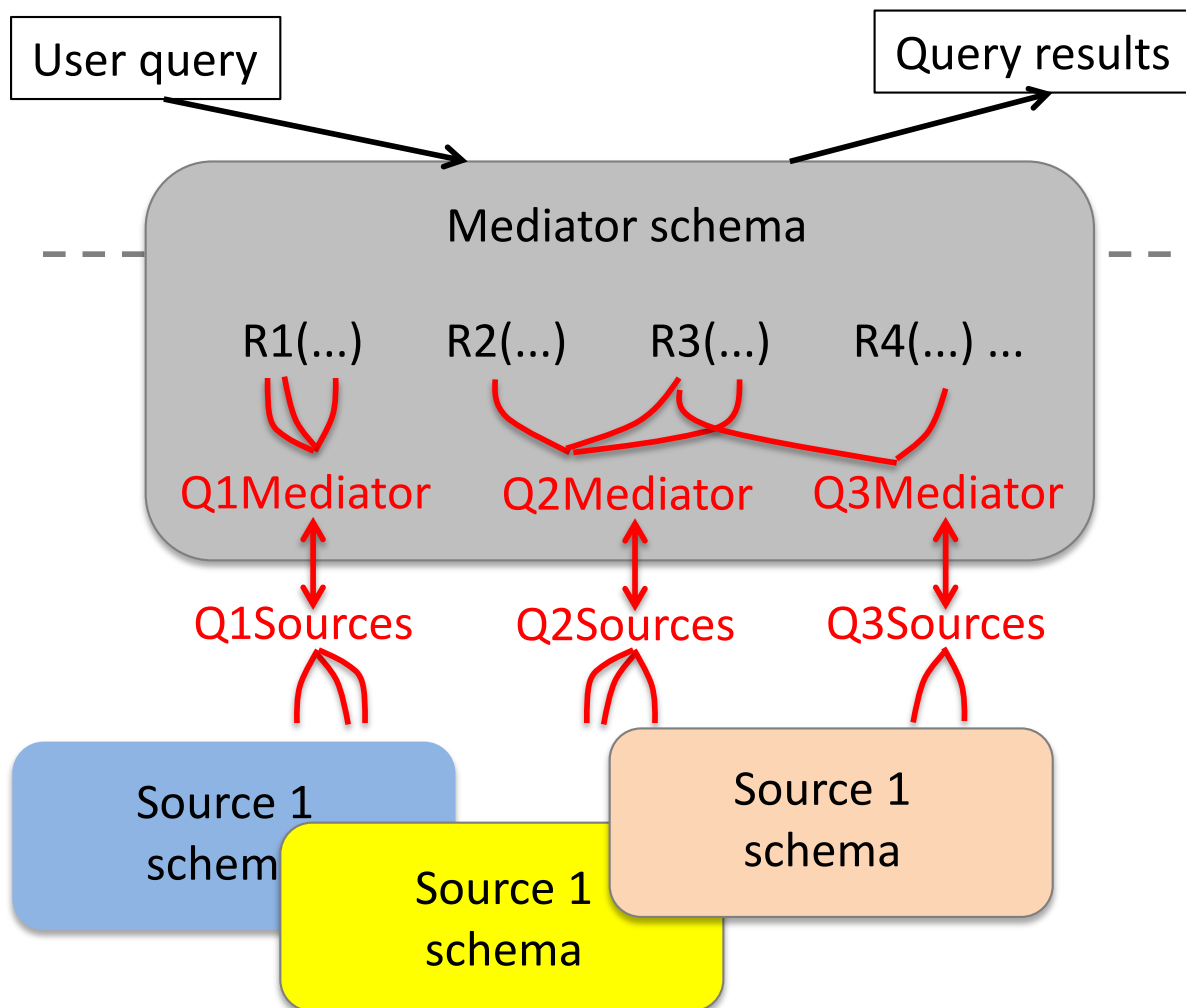
Q3Sources: select lh.name, r.name, h.roomPrice * 0.5 as hPrice, r.rating as rRating
from LyonHotel lh, Restaurants r
where r.city='Lyon' and r.name='Lion d'Or' and r.street=lh.street

This *cannot* be expressed either in LAV or GAV.

This mapping says: "each result of Q3Sources leads to a SuperOffer in Lyon".

Other mappings could define more SuperOffers in Lyon, or in other cities, or with rRating=3...

Query Processing in GLAV



User queries asked on the mediator schema.

Q1Mediator, Q2Mediator, ... are queries over this schema

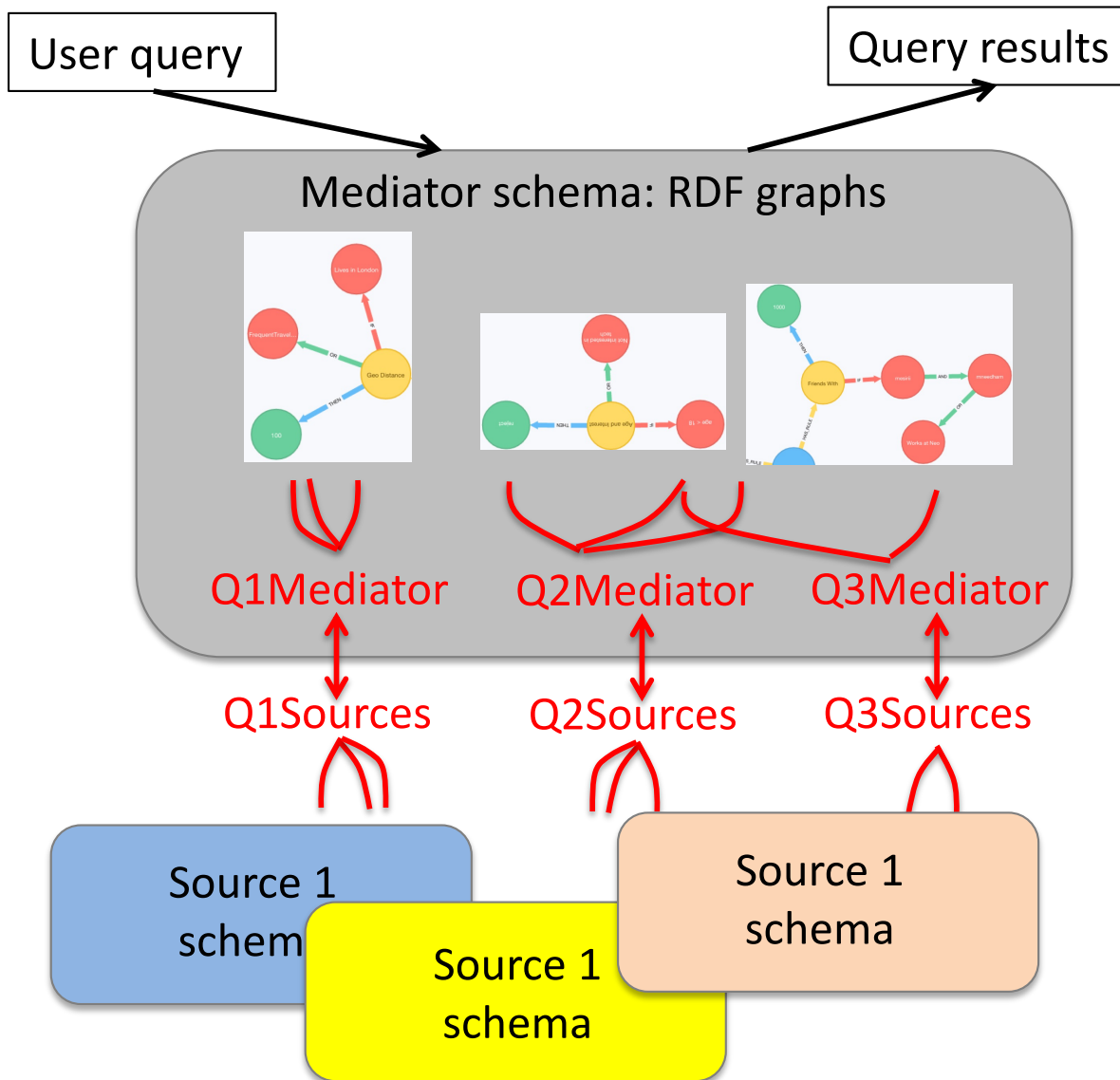
1. Apply **LAV**-style rewriting considering each QiMediator as a view over the mediator schema.
 - This leads to rewritings of Q over QiMediator relations (Q1Mediator, Q2Mediator, ...)
2. For each such rewriting, in **GAV** style, replace the symbol QiMediator by the query QiSources.
 - Then unfold → query over the sources themselves.

Examples: find all super offers in Paris? in Lyon?

Concluding remarks on GLAV

- The most flexible approach
 - Can express LAV, GAV, and more
- If a source changes or sources are added, as long as Q1Sources can be rewritten, applications will not be impacted
 - Only the "invisible" part of the system (the mappings) may have to be adapted
- Query rewriting remains expensive because it includes view-based query rewriting (NP-hard) as well as query unfolding (simple)

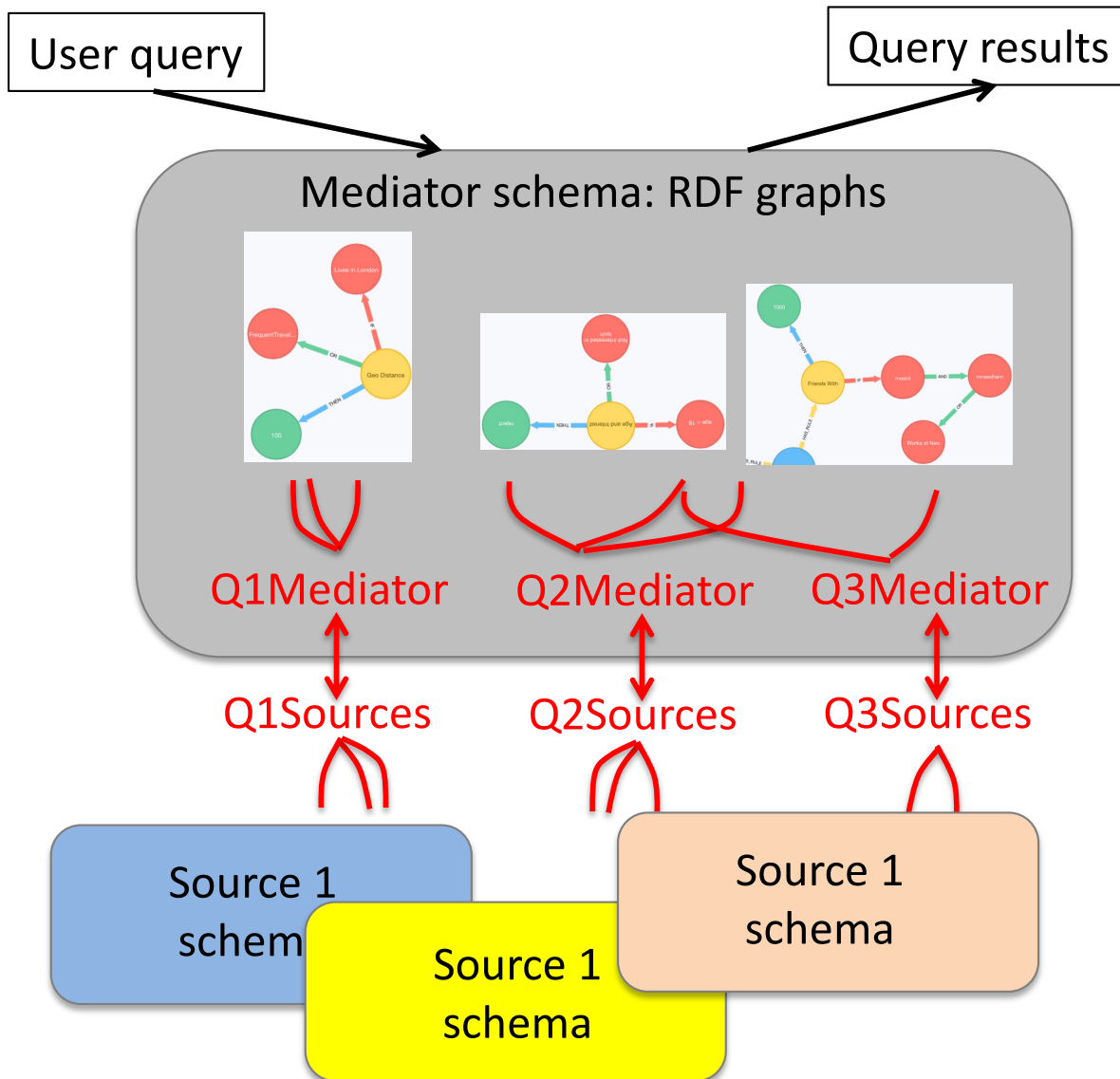
Modern mediators: GLAV with RDF global schema



Advantages:

- ❑ Heterogeneity of RDF allows a wide variety of data sources to be mapped
- ❑ Ontology at the level of the mediated (virtual) graph to describe application constraints
- ❑ E.g., subclass (specialization) hierarchies of classes; property typing (domain, range...)

Modern mediators: GLAV with RDF global schema



Sample GLAV mappings:

1. **Q1Sources:** an SQL query returning (x, y, z) tuples

Q1Mediator:

$(x, \text{'friend'}, y), (y, \text{'worksfor'} z)$
Q1Mediator "creates RDF out of relational data »

2. **Q2Sources:** a JSON query returning (z) nodes

Q2Mediator:

$(z, \text{'type'}, \text{Company})$

If common z value, the graphs built by Q1,2Mediator **connect!**

Final lab next week.