Querying Model Oriented Data From TAP Services

Any TAP evolution must be seamless or even shy

The problem

What we have (ADQL)

```
SELECT something FROM table WHERE (constraints on columns)
```

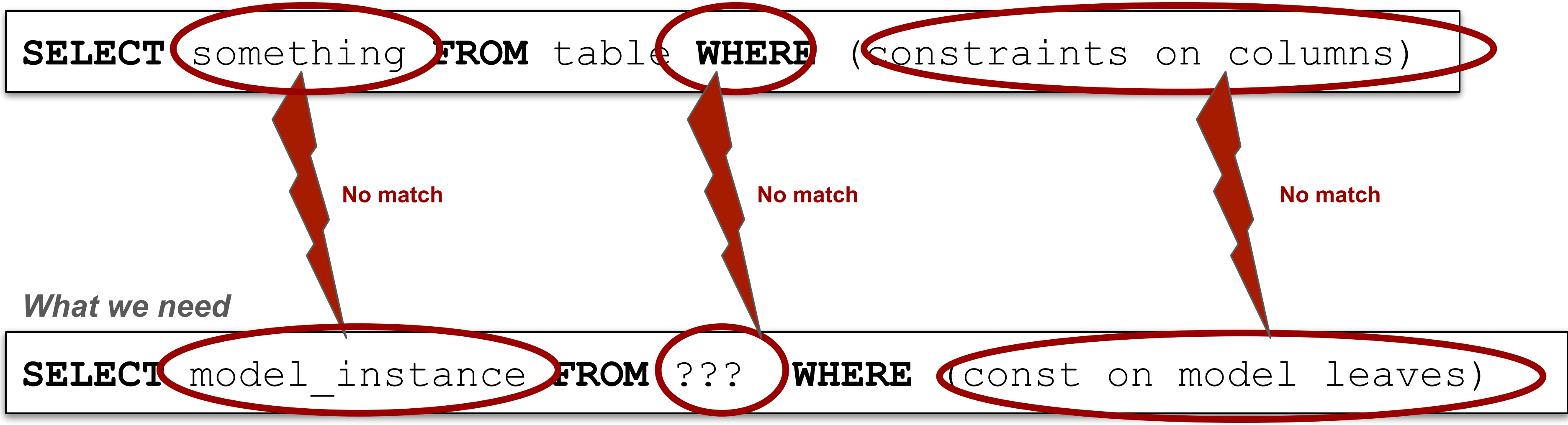
The problem

What we need

```
SELECT model_instance FROM ???  WHERE (const on model leaves)
```

The problem

What we have



```
SELECT something FROM table WHERE (constraints on columns)
```

What we need

```
SELECT model_instance FROM ??? WHERE (const on model leaves)
```

2 Situations

- **Searching legacy data**
  - users query data in the usual way
    - ADQL, VOTable
    - Data selection does not care about the model
  - Users expect the searched data to be mapped on a model
    - Data enhanced with a model view

- **Searching model instances (e.g. Provenance)**
  - Users do not know how the model is mapped on relational table
    - Just knows the model
    - ORM matter
  - Queries can only refer to model elements

Searching Legacy Data

- **No way to add model features to ADQL**
  - Neither seamless nor shy!
- **We can add a qualifier to the TAP query URL**

Searching Legacy Data

- **No way to add model features to ADQL**
  - Not seamless or shy either!

- **We can add a qualifier to the TAP query URL**

```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
FORMAT=Votable
QUERY=SELECT TOP 100 * FROM foo
```

Searching Legacy Data

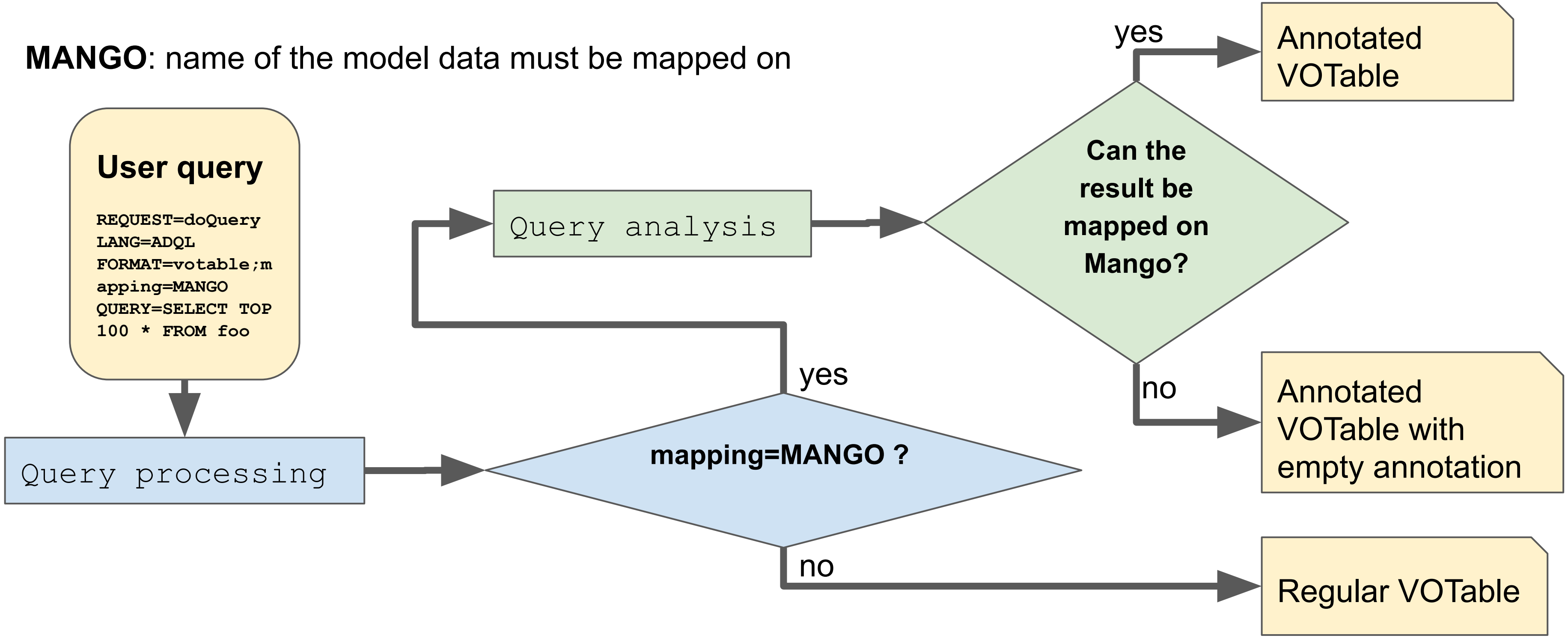
- **No way to add model features to ADQL**
  - Not seamless or shy either!
- **We can add a qualifier to the TAP query URL**
  - Just an example of what can be done

```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
FORMAT=Votable
QUERY=SELECT TOP 100 * FROM foo
```

```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
FORMAT=votable;mapping=MANGO
QUERY=SELECT TOP 100 * FROM foo
```

Model Annotation on Server Side

MANGO: name of the model data must be mapped on



- **The server processes queries in a regular way**
- **When annotations are requested, they are added by post-processing**

Searching Model Instance

- **What about an OO Query language**
  - Not enough demand to undertake something that never succeeded in others circumstances
  - Let’s focus on our use-cases

- **Typical use case**
  - Retrieving the provenance of a given dataset
  - Likely served by a Datalink
  - No need of a full featured query language

Searching Model Instance

- **What about an OO Query language**
  - No enough demand to undertake something that never succeeded in others circumstances
  - Let’s focus on our use-cases

- **Typical use case**
  - Retrieving the provenance of a given dataset
  - Likely served by a Datalink
  - No need of a full featured query language

```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
QUERY=SELECT TOP 100 * FROM foo
```

Searching Model Instance

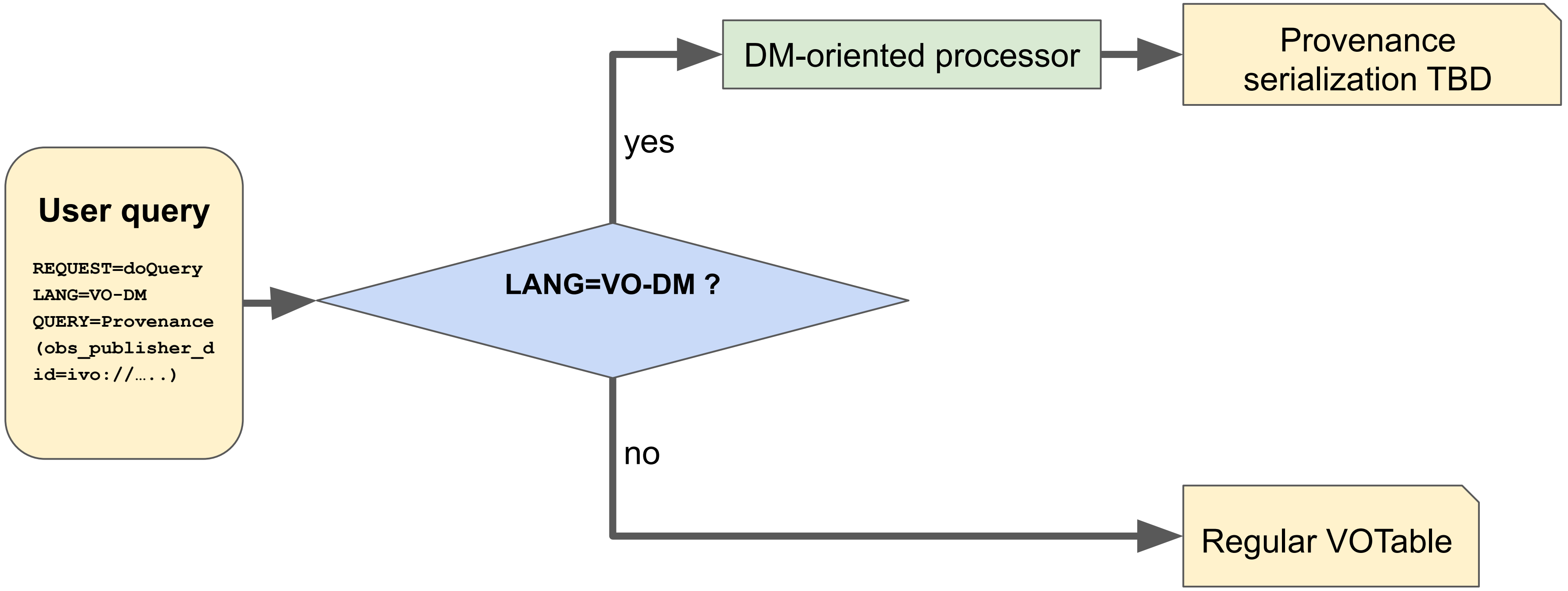
- **What about an OO Query language**
  - No enough demand to undertake something that never succeeded in others circumstances
  - Let’s focus on our use-cases

- **Can go ahead without OO query language**
  - Provenance user case:
    - Retrieving the provenance of a given dataset
    - Likely served by a Datalink
    - No need of a full featured query language

```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
QUERY=SELECT TOP 100 * FROM foo
```

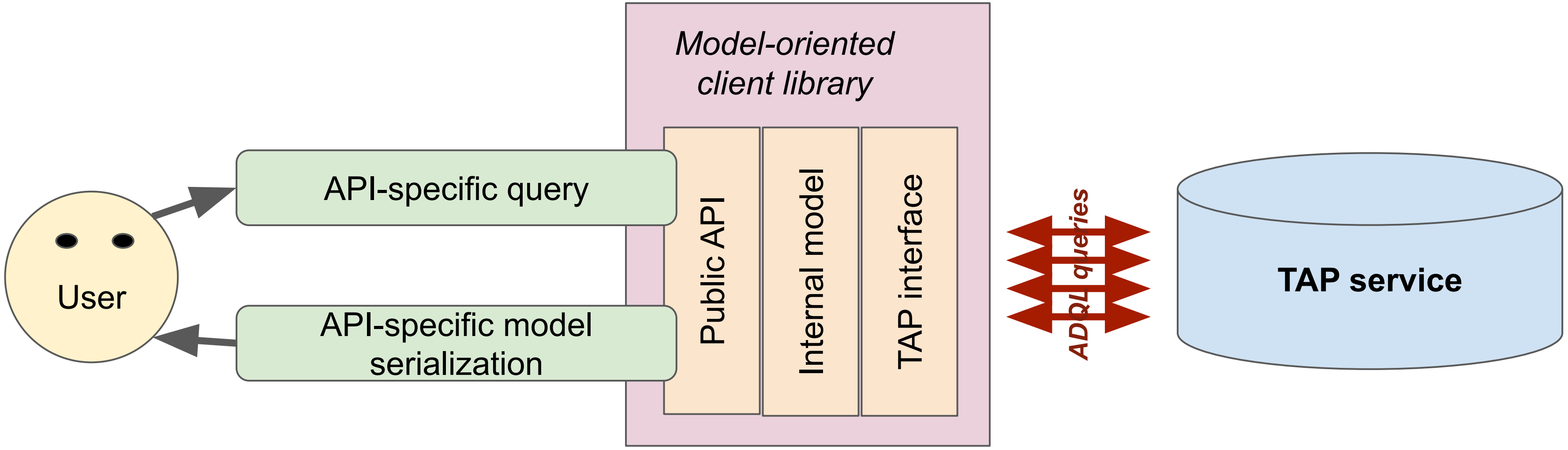
```
HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=VO-DM
QUERY=Provenance(obs_publisher_did=ivo://…..)
```

Searching Model Instances: Server side processing



- **The VO-DM query cannot be processed by a regular server**
  - A model-specific processor must be run

Searching Model Instances: Client side processing



- **The object layer can be delegated to the client**
    - The client module sent simple queries to the server to retrieve model components
    - It reconstructs searched model instances from those query results
    - It provide some instance serialization to the final user (human or software)