The Path Transparency Observatory

Brian Trammell, ETH Zürich

MAMI Summer School on Path Transparency Measurement
Aberdeen, Scotland, 11 June 2018

measurement and architecture for a middleboxed internet

measurement | architecture | experimentation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.The opinions expressed and arguments employed reflect only the authors' view. The European Commission is not responsible for any use that may be made of that information.

Scaling Path Transparency Measurement

- with PATHspider, we've seen...
  - measurements from a single machine
  - in a single run
- How to compare measurements...
  - from multiple vantage points
  - across longer time scales?

- Answer: centralize analysis in an ***observatory***.

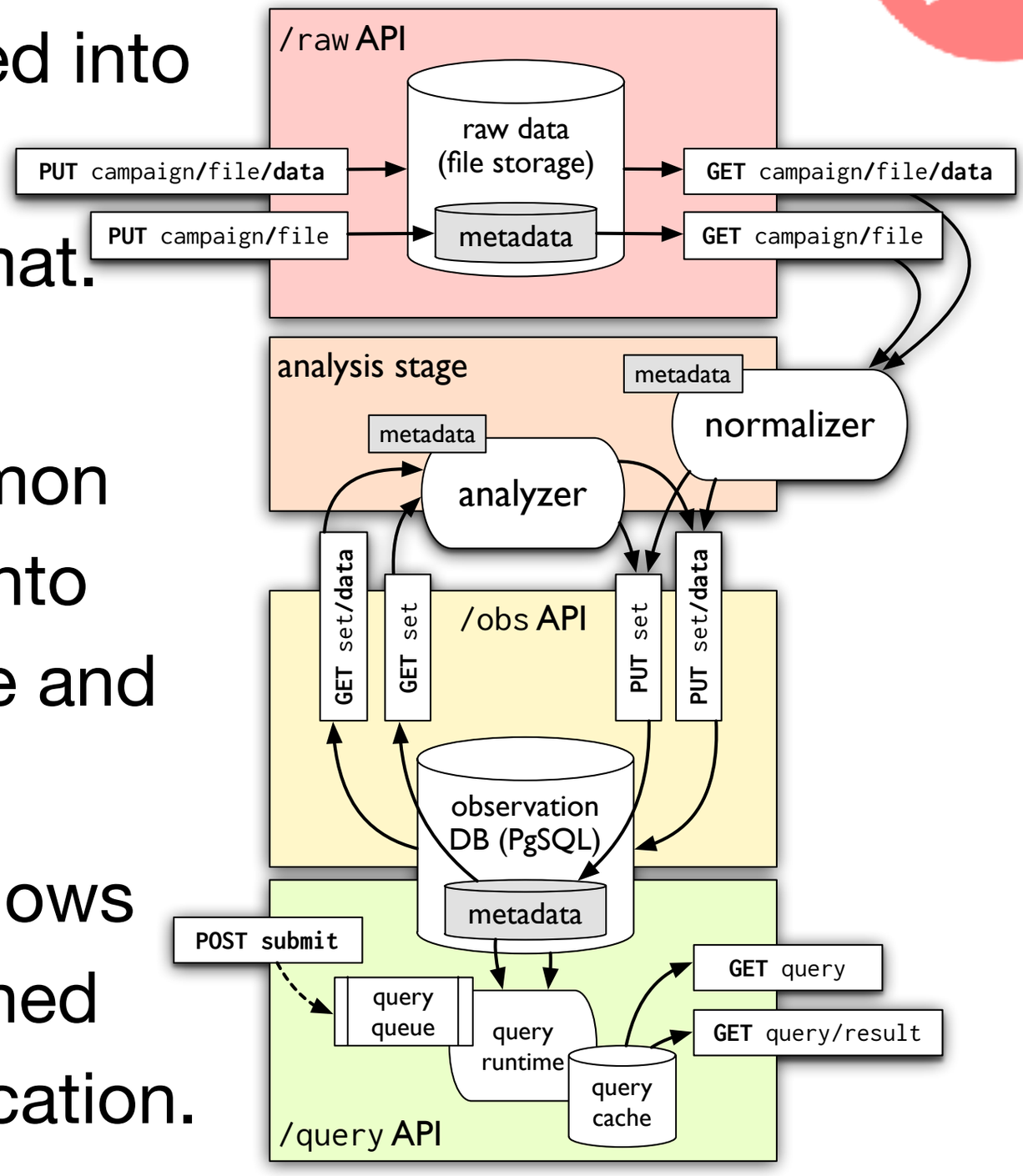
Observatory Design Goals

Measurement data observatories can support better science through the following design goals:

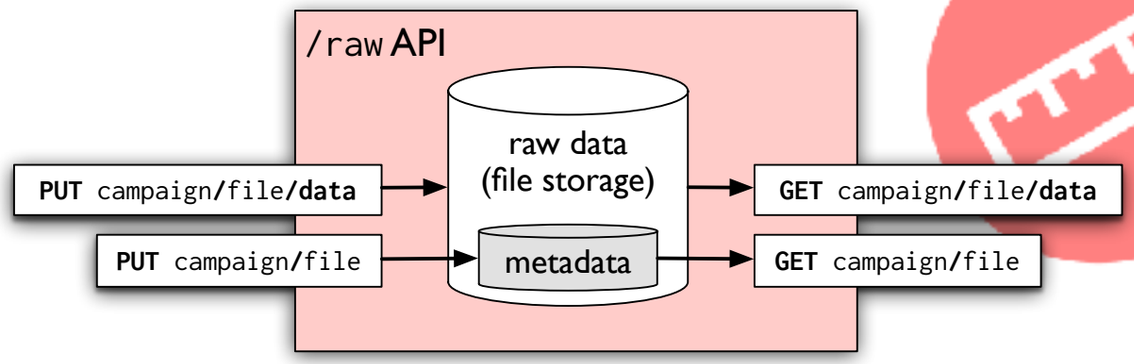
- Comparability: allow the results from diverse tools to be expressed in the same vocabulary so they can be compared.
- Repeatability: keep enough metadata around so that future users of the data know how to repeat the experiment.
- Protection: reduce information in raw data to only that needed for a particular analytical task.

PTO design

- Raw data store organized into ***campaigns*** containing ***raw files*** in original format.
- Analysis stage derives ***observations*** in a common data model, organized into sets sharing provenance and other metadata
- Flexible query engine allows ***query*** results to be cached and annotated for publication.



Raw Data and Metadata



- Every campaign and datafile is associated with metadata
  - Metadata must be created before data is uploaded
  - Metadata mutable, data is not
  - Raw files inherit metadata
- ***System metadata*** required for every file:
  - `_time_start, _time_end:` time interval covered by measurements
  - `_owner`: who owns the raw data (and should be contacted for questions about it), as email address or URI.
- ***User metadata*** is freeform

Upload: creating a campaign

- First, set up your environment: you'll need your API key and campaign name:

```
$ export TOKEN="your API key"
$ export CAMPAIGN="your campaign name"
```

- Then, create metadata for your campaign, as a JSON file with these keys:

```
{
    "_owner": "your email address",
    "_file_type": "pathspider-v2-ndjson-bz2"
}
```

Upload: creating a campaign

- ```
  $ curl -X PUT
         -H "Content-type: application/json"
         -H "Authorization: APIKEY $TOKEN"
         https://summer.pto.mami-project.eu/raw/$CAMPAIGN
         --data-binary @campaign-metadata.json
  ```

- Now we have a campaign. Next, let's upload some data into it...
  ```

Upload: Create and upload metadata

- Data files need at least a time range (`_time_start, _time_end`) in their metadata.
  - We can extract this from PATHspider output:

```
$ python3 extract_pathspider_metadata.py pathspider_files
```

- Now we can upload metadata for each file:

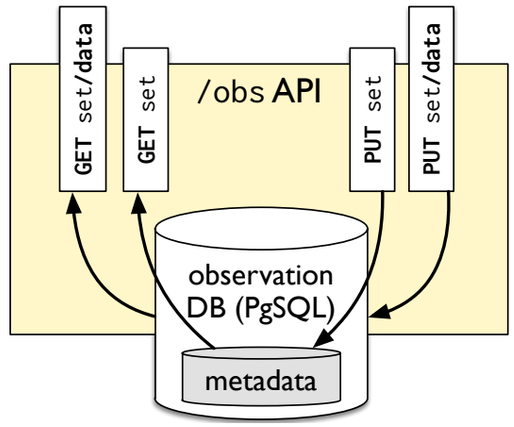
```
$ curl -X PUT
       -H "Content-type: application/json"
       -H "Authorization: APIKEY $TOKEN"
       https://summer.pto.mami-project.eu/raw/$CAMPAIGN/file
       --data-binary @file.meta.json
```

Upload: upload raw data

- And finally, data:

```
$ curl -X PUT
       -H "Content-type: application/bzip2"
       -H "Authorization: APIKEY $TOKEN"
       https://summer.pto.mami-project.eu/raw/$CAMPAIGN/file
       --data-binary @file
```

Observation Data Model



- Raw data is important for provenance and repeatability, but before we can query our data, we need to normalize it into observations.

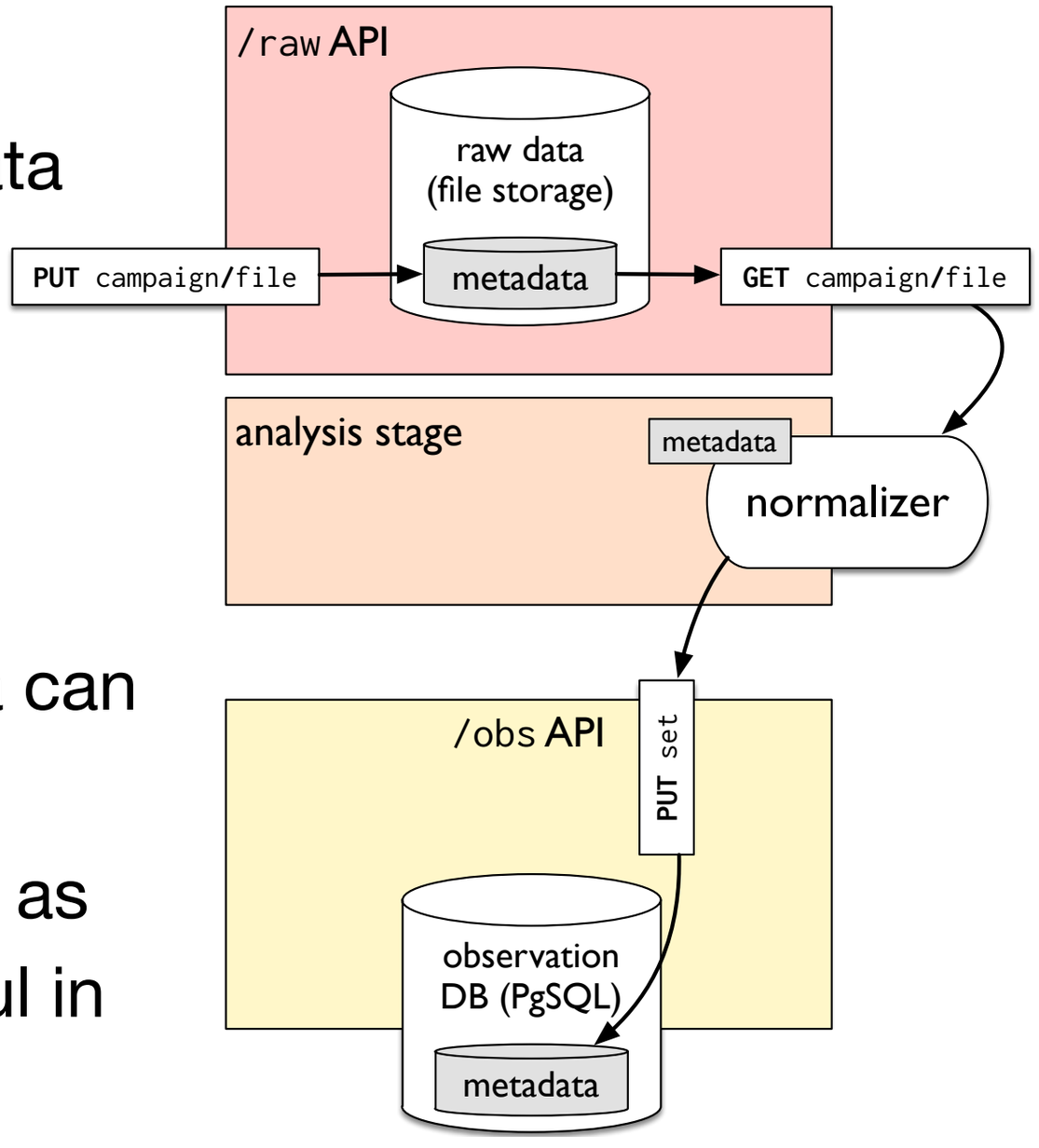
- An observation is an assertion that at a given **time**, a given **condition**, held on a given **path:**

```
["0","2014-08-28T22:41:02Z","2014-08-28T22:46:25Z",
  "* 82.192.86.197","ecn.multipoint.connectivity.works","3"]
```

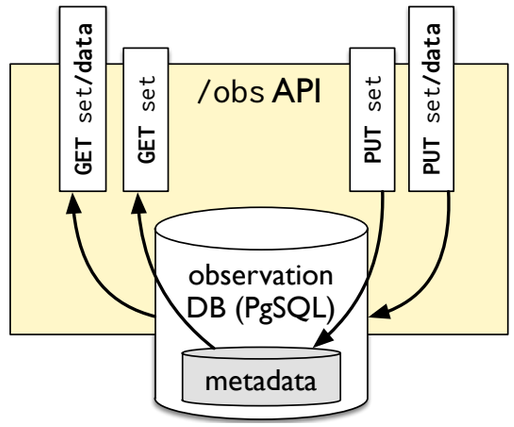
- Observations are organized into **sets**, which share metadata and provenance.

Metadata Flow

- PTO's design is **metadata-first:**
  - All additions consist of a metadata phase, then a data phase.
  - Normalizers and analyzers are controlled indirectly by raw and observation set metadata.
  - Data is immutable, but metadata can be updated.
- Arbitrary metadata: we don't know as well as the users what will be useful in the future.



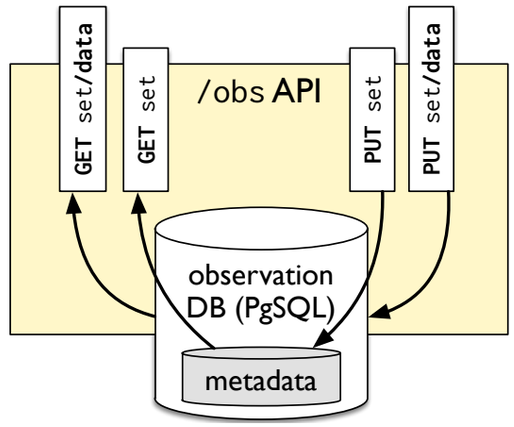
Conditions



- A condition is some **state** of an **aspect** of a **feature** observed on a network:
  - e.g. **ecn**.**connectivity**.**works** or **ecn**.**negotiation**.**reflected**

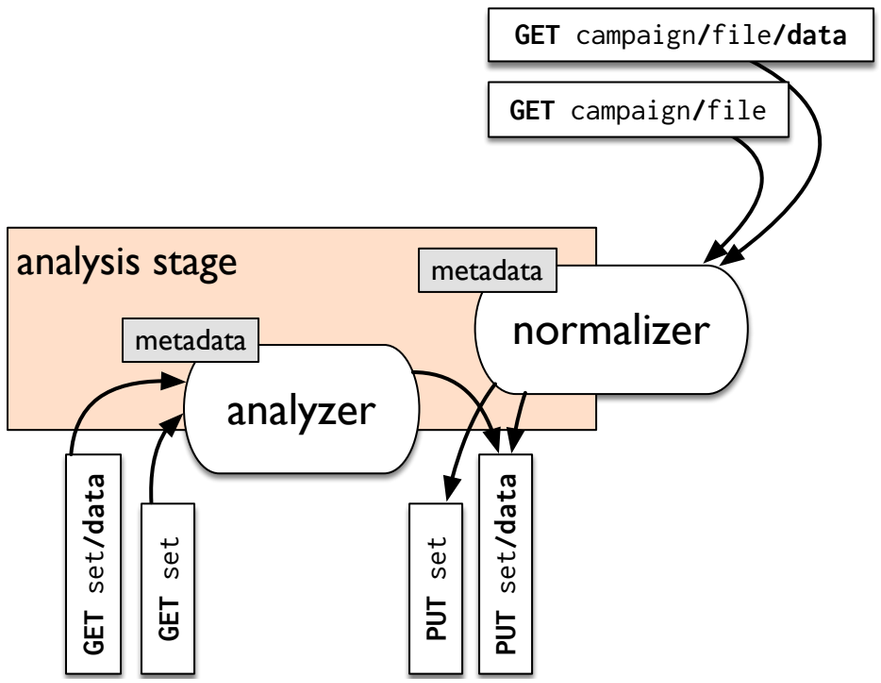
- Feature: what protocol or feature are we trying to use?
- Aspect: what question are we asking about the feature?
- State: what happens when we try to use it?
  - States mutually exclusive for a given aspect on a path at a given time.

Paths



- A path is a **sequence of path elements** (IP addresses, prefixes, AS numbers, pseudonyms, or wildcard) on which an observation was taken
- For active measurement: first path element is the **source** or device sending traffic, last path element is the **target** or device under test.
- e.g. `[digitalocean-ams3 * 104.24.127.228]` or `[128.10.18.52 * 209.200.170.230 204.106.55.245 * 159.45.6.20]`

Normalizers and Analyzers



- *Normalize* raw data and metadata into observations:
- **`ptonorm normalizer campaign file > obs.ndjson`**
  - takes data on stdin, metadata on fd 3
  - produces data + metadata on stdout
- **`ptoload obs.ndjson`**
  - loads resulting observation set

- *Analyze* observation sets into new observation sets:
- **`ptocat set₀ ... setₙ | analyzer > obs.ndjson`**

Step two: normalize raw data to observations

- Normalization and analysis take place on the PTO server itself.
- To normalize all the PATHspider files of a given type in a list of campaigns:

```
for c in `cat CAMPAIGNS`; do
    for f in `pushd -q $RAWSTORE/$c && ls *.bz2 && popd -q` do;
        ptonorm ecn_normalizer $c $f > $OBSCACHE/$c-$f.obs
        ptoload $OBSCACHE/$c-$f.obs
```

- (In a multi-owner environment, normalizers are provided by data owners, run by the observatory operator, and the resulting observation sets are vetted by the owner and operator before loading)

Step three: additional analysis

- Let's look for path dependency: different results for the same target from different sources. First, we need a list of sets to analyze:

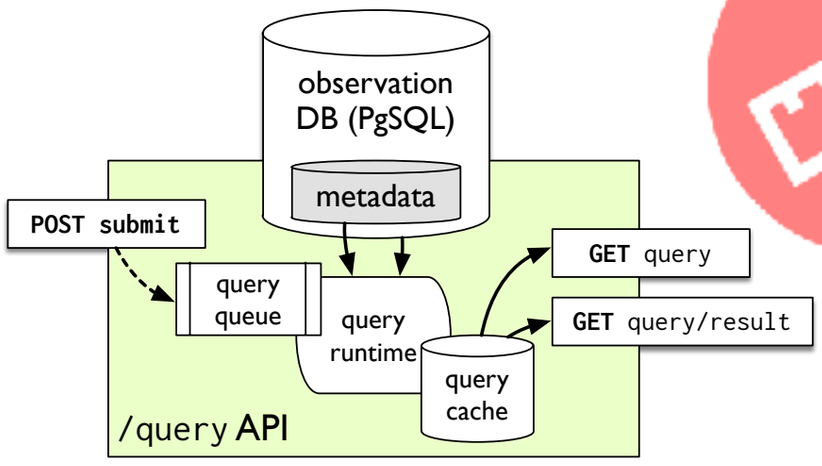
```
curl -H "Authorization: APIKEY $TOKEN"
      https://summer.pto.mami-project.eu/obs/by_metadata?\
      analyzer=https://raw.githubusercontent.com/\
      mami-project/pto3-ecn/master/ecn_normalizer/\
      ecn_normalizer.json | jq `.["sets"]`
```

Step three: additional analysis

- Now we can analyze all these sets, comparing results for the same target:

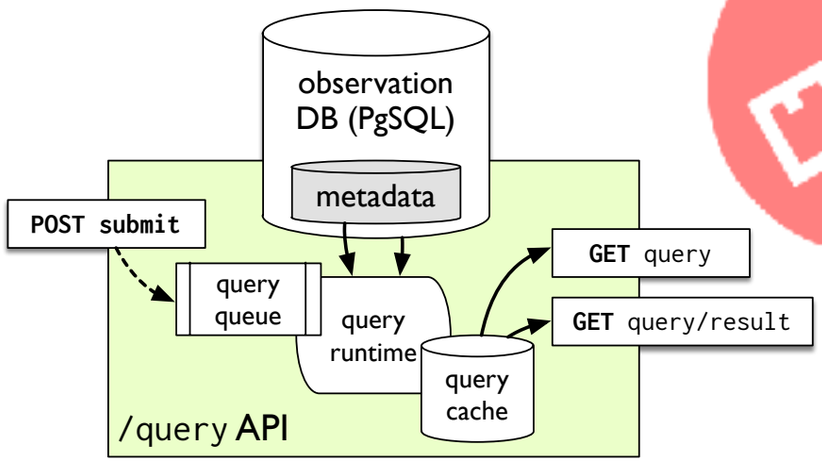
- `ptocat \`cat SETS\` | ecn_pathdep > pathdep.obs`
- `ptoload pathdep.obs`

Queries



- Queries select observations across observation sets based on predicates given in the query parameters.
- Queries are associated with metadata, and arbitrary metadata may be added to them.
- Query results are cached temporarily, and may be made permanent by adding an external reference to them.

Query Parameters



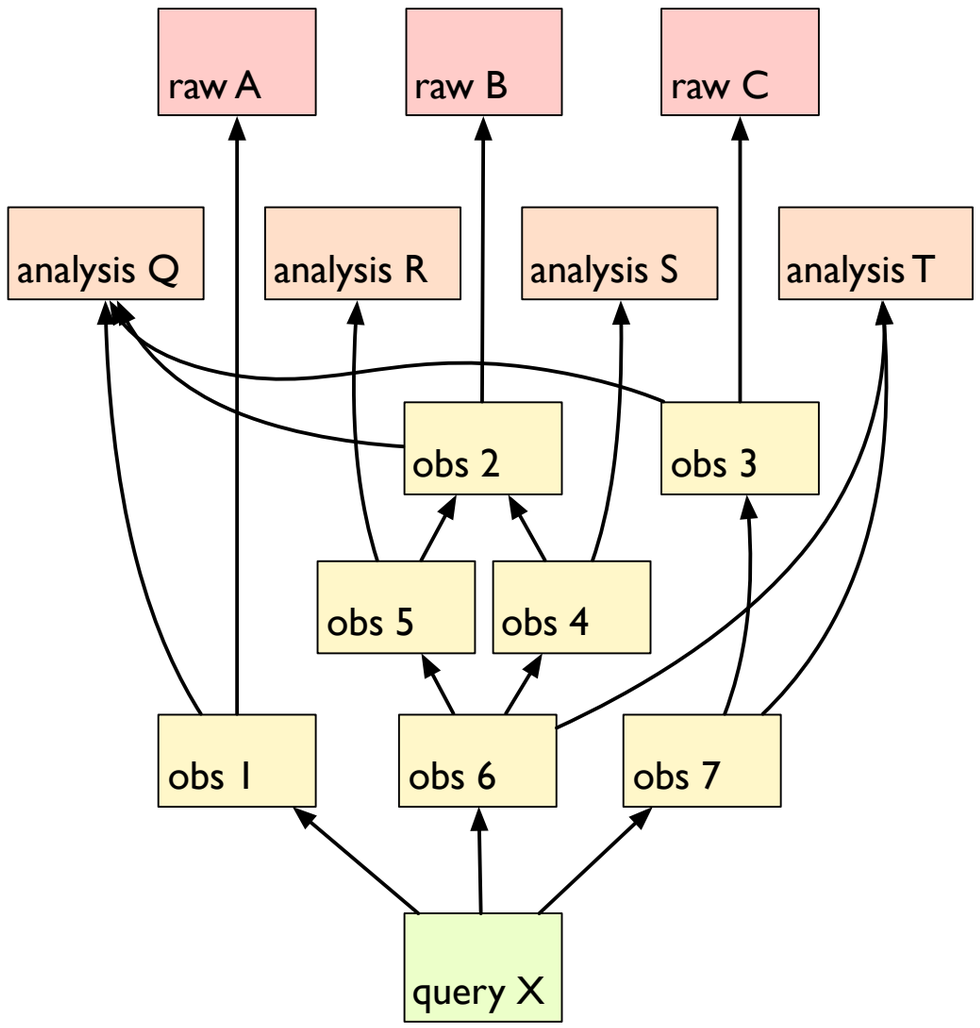
Parameter	Semantics	Meaning
`time_start`	temporal	Select observations starting at or after the given start time
`time_end`	temporal	Select observations ending at or before the given end time
`set`	select	Select observations with in the given set ID
`on_path`	select	Select observations with the given element in the path
`source`	select	Select observations with the given element at the start of the path
`target`	select	Select observations with the given element at the end of the path
`condition`	select	Select observations with the given condition, with wildcards
`group`	group	Group observations and return counts by group
`option`	options	Specify a query option

Step four: aggregate queries

- For working with queries, we use a Python client library in Jupyter notebooks:

```
$ cd summerschool/pto-notebook
$ jupyter notebook
```

Supporting Repeatability: Provenance



- Following `__sources`, `_sources`, and `_analyzer` links back from a query or observation set results in a **provenance tree.**
- This provenance tree is, in effect, a set of instructions for recreating a given observation set.

PTO: what we've seen

- a metadata-first observatory for supporting comparable and repeatable network measurements: this is a general design pattern you should consider in your future work.

- a data model and set of analysis and query tools for pulling path transparency measurements together into a single place, demonstrating the general design pattern.