Observatories:

Collection, Preservation, Metadata, and Provenance for Active Measurement

Brian Trammell, ETH Zürich

Repeatability and Comparability in Measurement (RCM) Tutorial

ACM SIGCOMM 2018, Budapest, 20 August 2018

mami

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.

Supported by the Swiss State Secretariat for Education, Research and Innovation under contract number 15.0268. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

Scaling Up Active Measurement

- Remembering what you did and how is almost as important as trying to do the right thing in the first place,
- Remembering what you did a thousand times requires good tooling.

- Here, we'll present a general model for scaling up, supporting repeatability and comparability.
- We call these *observatories.*

Role of an observatory

- Provides a central data repository to store data together with metadata, tracking provenance.
- Makes it easier to share interemediate and final results.

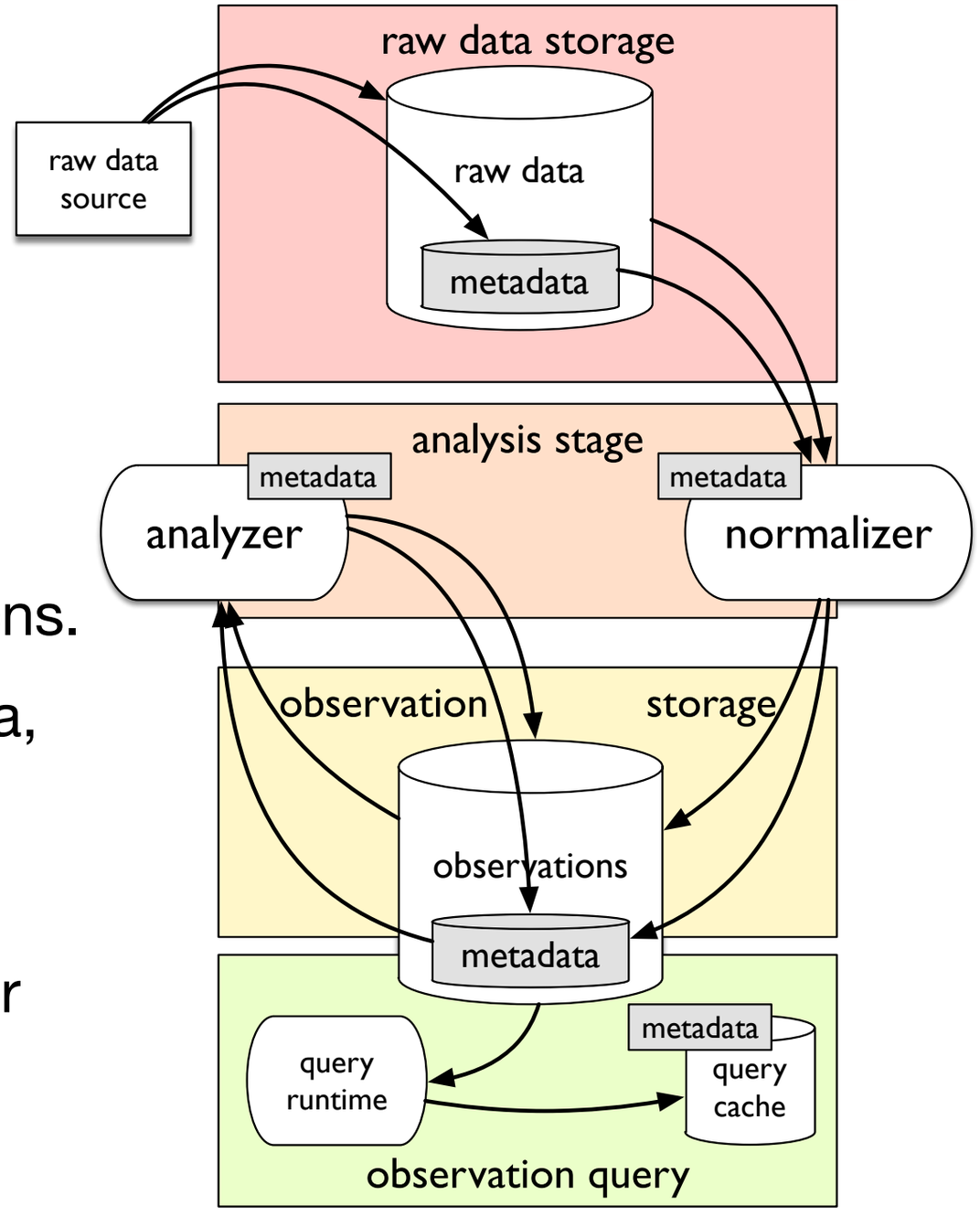
Design Goals

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.

Generalized Observatory Workflow

- Raw data submitted to observatory
  - ...by value or reference
- Normalizers convert raw data to observations, controlled by raw metadata
- Analyzers create derived/combined observations from (lower-level) observations.
- Observations stored in a common schema, grouped into sets that share metadata
- Queries over observations are always temporally scoped, and can be cached for permanent reference.



What does an observatory do?

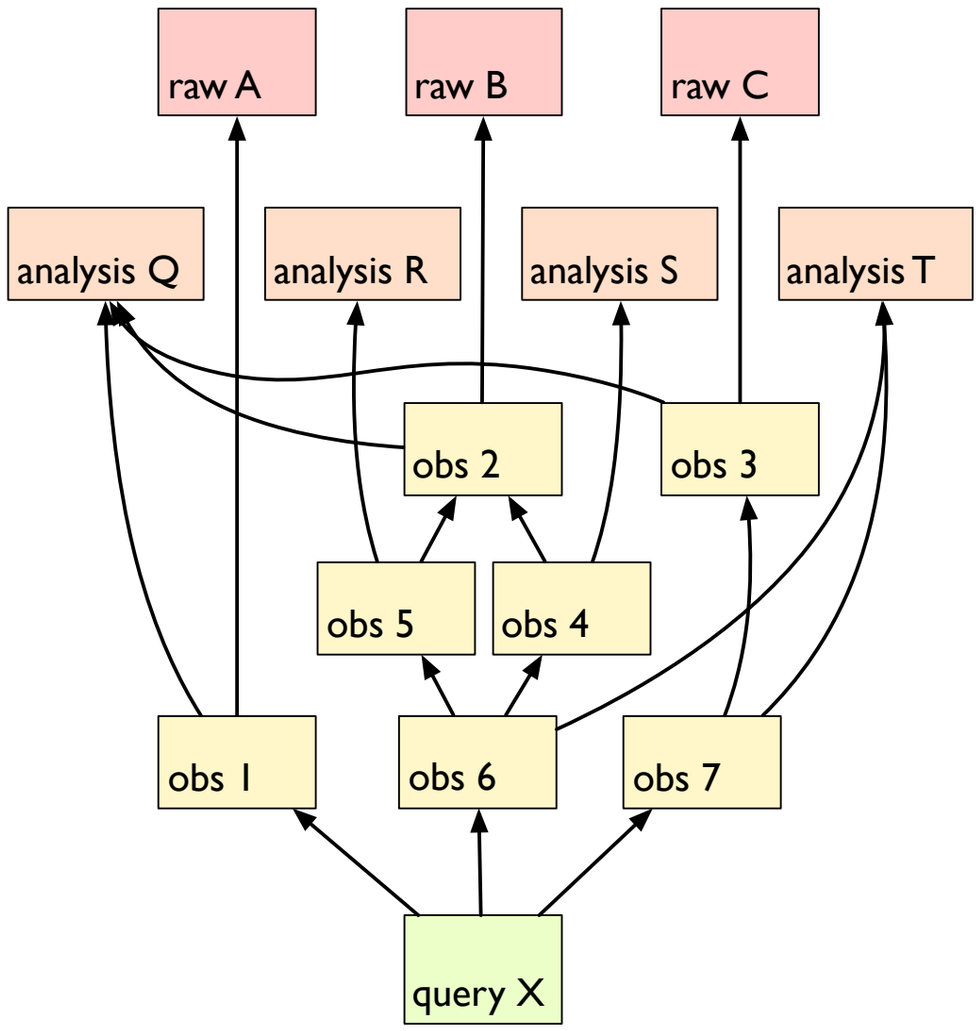
- ***Collect***: take data from multiple tools at multiple vantage points in multiple formats, centralizing it for analysis.
- ***Preserve***: keep raw data in its original form and control access to it.
- ***Normalize***: produce ***observations*** from raw measurement data in a simple schema designed for a specific measurement purpose.
- ***Analyze***: combine observations for higher-level insight.
- ***Query***: allow arbitrary queries over observations.
- Track ***Metadata***: Annotate raw data and observations with arbitrary user-defined information.
- Track ***Provenance***: Provide information about raw and intermediate data sources for all objects/queries in the observatory.

Supporting Comparability: Observations

- Diversity of tools/methods can make comparison difficult.
- Normalizing raw data into *observations* with simple semantics can push this difficulty to the edge.

- An observation schema is specific to the measurement goal...
  - ...should consist of a small set of classes, each containing information elements derived from input data and useful for analysis.
  - e.g. a topology-centered observatory's core object could be a "hop"
  - e.g. a performance observatory could store latency and bandwidth samples

Supporting Repeatability: Provenance



- Every object refers to its antecedents
- Following these 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.

Our instance: The Path Transparency Observatory

Brian Trammell, ETH Zürich

Repeatability and Comparability in Measurement (RCM) Tutorial

ACM SIGCOMM 2018, Budapest, 20 August 2018

mami

measurement and architecture for a middleboxed internet

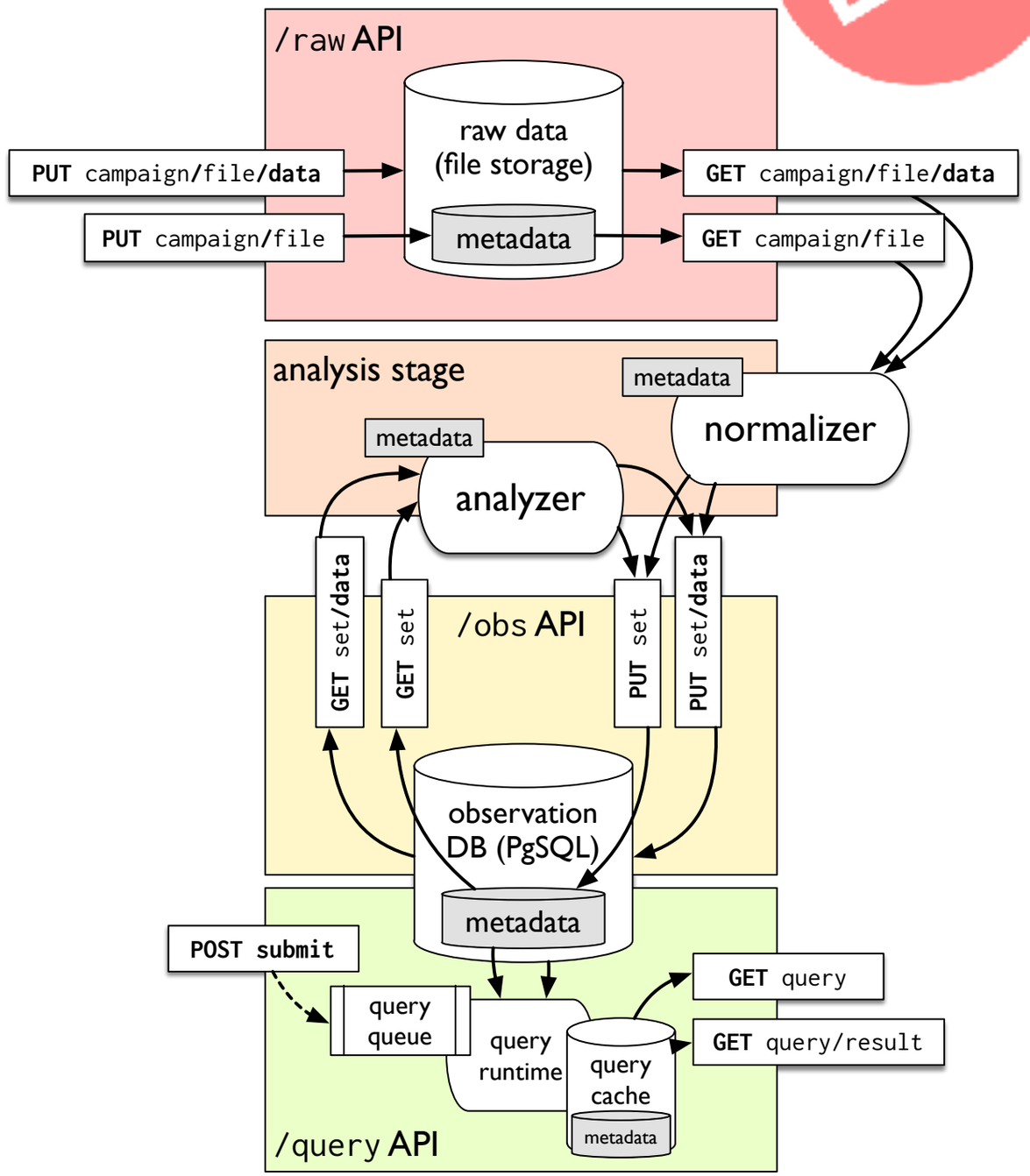
measurement | architecture | experimentation

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.

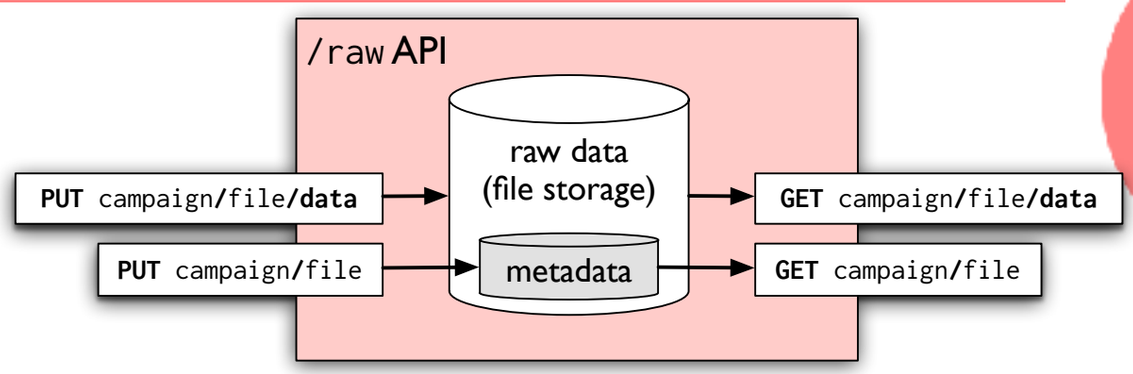
Supported by the Swiss State Secretariat for Education, Research and Innovation under contract number 15.0268. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

PTO design

- Built around a RESTful API
- ***Raw data*** is stored as *files* in original output format of each tool/ source
  - "metadata first"
- ***Analysis*** stage derives ***observations*** in a common schema
  - organized into sets sharing provenance and other metadata
- Flexible ***query*** engine to access observation storage
  - allows query results to be cached and annotated for publication



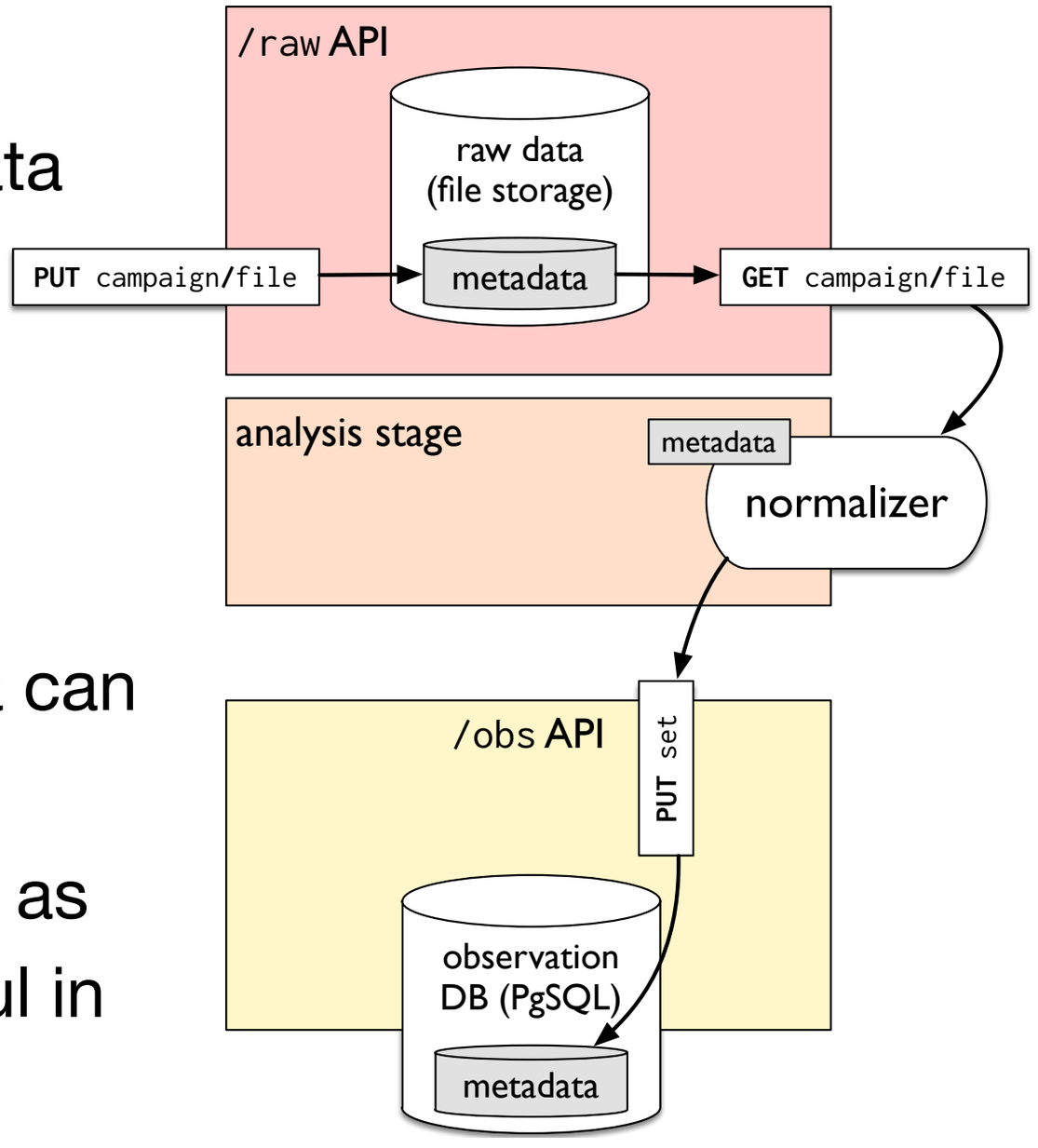
Raw Data Files and Campaigns



- Raw data files are grouped into ***campaigns***
- 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 from campaigns
- ***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.
  - `_file_type`: format of the raw data file
- ***User metadata*** is freeform

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.



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 (edit) your campaign metadata file (`campaign.json`):

```
{
    "_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 $TOKEN"
         https://rcm.pto.mami-project.eu/raw/$CAMPAIGN
         --data-binary @campaign.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 result.ndjson
$ cat result.ndjson.meta.json
```

- Now we can upload metadata for each file:

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

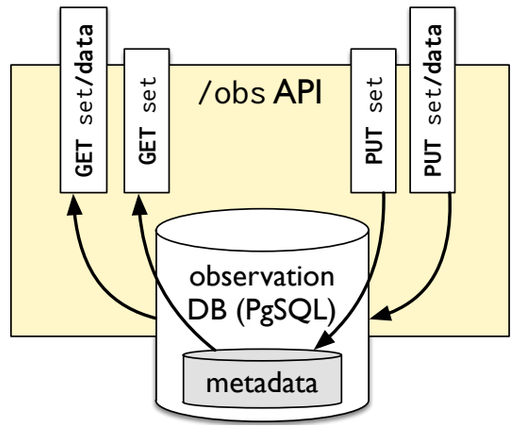
- ***yes, you're uploading metadata at the url identifying the datafile: metadata first!***

Upload: upload raw data

- And finally, data. Compress it first:

```
$ bzip2 result.ndjson
$ curl -X PUT
-H "Content-type: application/bzip2"
-H "Authorization: APIKEY $TOKEN"
https://rcm.pto.mami-project.eu/raw/$CAMPAIGN/result.ndjson/data
--data-binary @result.ndjson.bz2
```

Observation Data Model

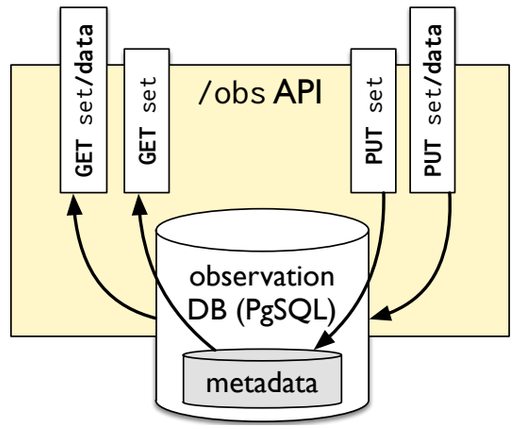


- Raw data is important for provenance and repeatability, but before we can query our data, we need to normalize it into observations.
- A PTO observation is an assertion that at a given **time**, a given **condition**, held on a given **path:**

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

  - ***This schema is what makes our observatory a PTO.***
- Observations are organized into **sets**, which share metadata and provenance.

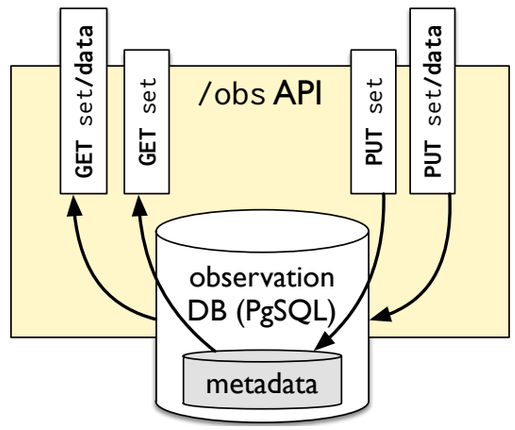
Conditions for Path Transparency



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

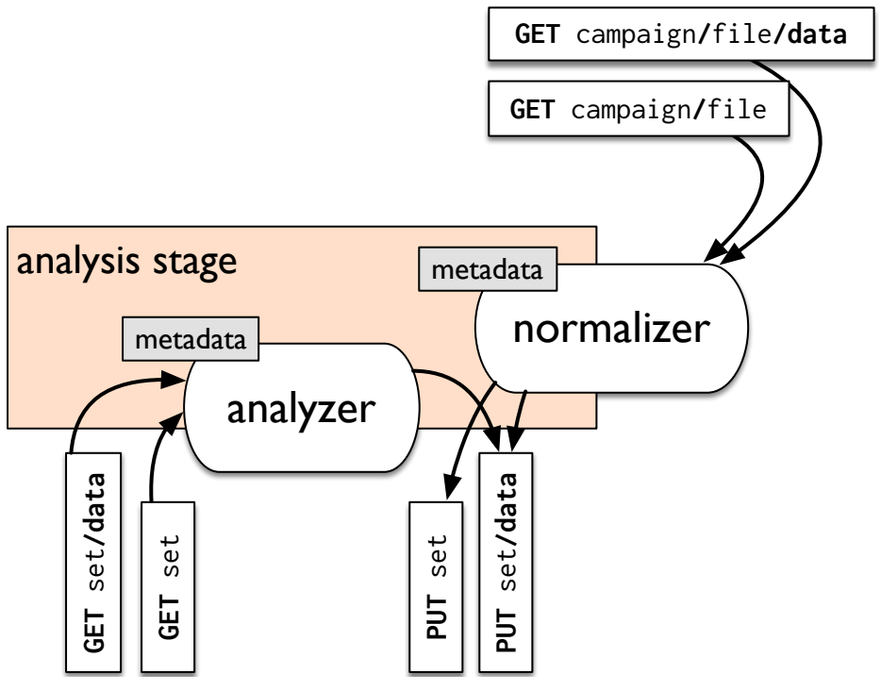
- Feature: what protocol or feature are we trying to use?
- Aspect: what question are we asking about the feature?
- State: what happens when we try to use it?
  - States mutually exclusive for a given aspect on a path at a given time.

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
- **`autonorm`**
  - normalizes new files in a campaign according to simple rules

- *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.
- We'll run **`autonorm`** to kick off the pathspider normalizer.

- This is usually semi-manual:
  - Normalizers are provided by data owners, run by the observatory operator
  - 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:

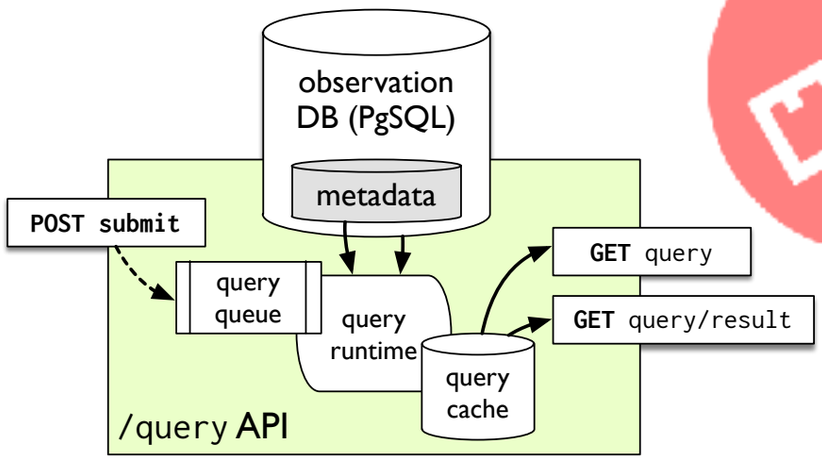
```
$ curl https://rcm.pto.mami-project.eu/obs/by_metadata?\
      analyzer=https://raw.githubusercontent.com/\
      mami-project/pto3-ecn/master/normalize_pathspider/\
      normalize_pathspider.json | jq `.["sets"]`
```

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



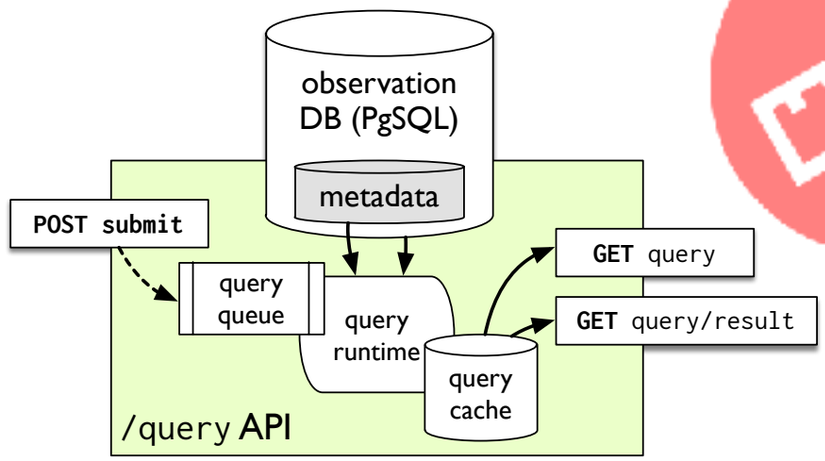
- Queries select or aggregate observations across observation sets based on predicates given in the query parameters.
- Queries are associated with metadata, and arbitrary metadata may be added to them.
- Queries might take a while, so result retrieval is separate from metadata management.
  - Query results are cached temporarily, and may be made permanent by adding an external reference to them.

Step four: aggregate queries

- Queries can also be issued from curl...
- ...but we also supply a Python client library meant for working with queries and query results in Pandas

- We'll switch to a notebook running on this for illustrating queries...

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

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.
  - See `https://github.com/mami-project/pto3-go`.

- 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.

PTO: lessons learned from our implementation

- Simple interfaces for analysis are way more flexible
  - Prior versions of the PTO were big-data-heavy, required all analysis to be written to specific Apache frameworks.
- Thinking hard about your observatory data model pays off
  - Few changes in the core schema since project start.
  - Structured data allows us to dispense with NoSQL.
- For many tasks, medium data suffices
  - Our PTO instance stores ~1T of data, built on flat files and PostgreSQL on one large host (40-core, 256G/40T).