Path Transparency Observatory (PTO) Version 3: *Once More, With Feeling*

Brian Trammell (ETH)

MAMI Plenary Cambridge, 31 January 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.

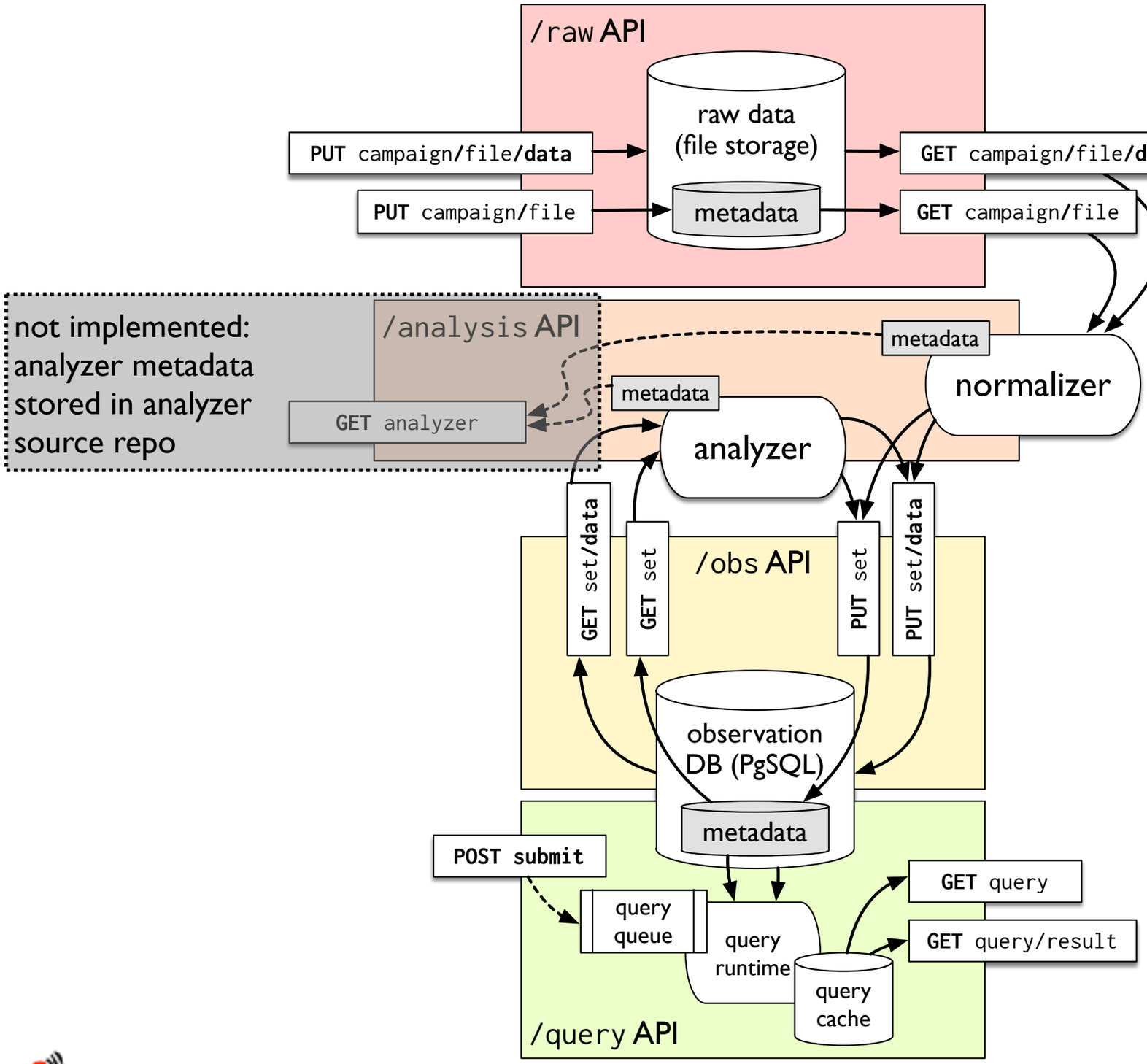
Current Status

- PTOv3: reimplement pilot PTO to a RESTful architecture
  - Less BigData™ overhead, web concepts for provenance
- Timeline in Oslo (Python 3 + flask + SQLalchemy impl)
  - raw data by August
  - analysis up in September
- Timeline in reality (almost bare-metal Golang impl)
  - alpha nowish, doing first real analysis
  - beta by IMC '18 deadline

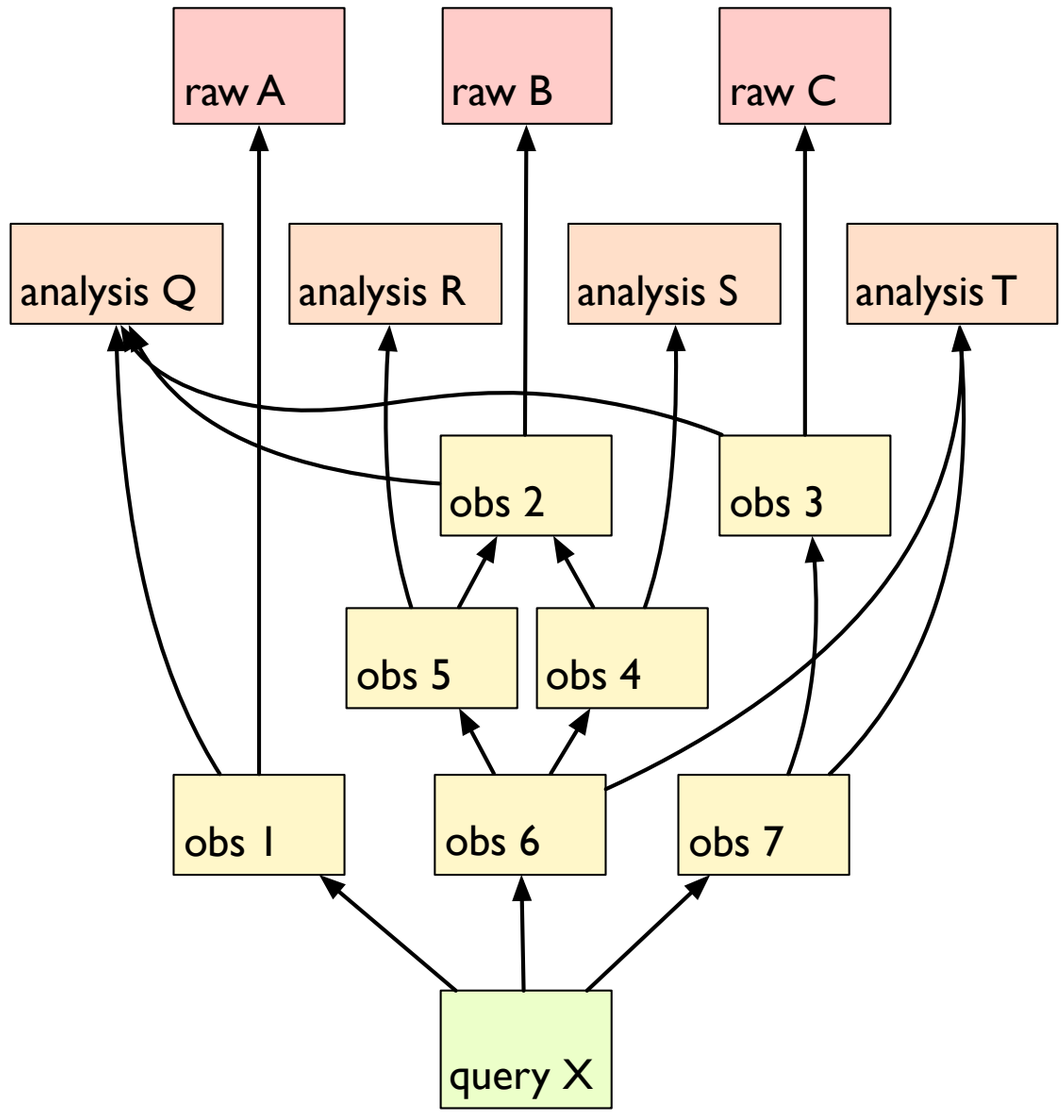
What changed?

- Migrate (again) from Python to Go
  - Python web frameworks get annoying quickly unless all you want to implement is the example social network app.
  - Go support for web API development in standard library is easier to deal with, higher performance, less prone to concurrency errors.
- Raw access to observation data through /obs paths
  - Allows easy creation of remote analyzers.
- Replace IQL with a (less capable) HTTP parameter representation
  - Easier analysis of queries to generate provenance information
  - Restriction of queries to reduce database running time

PTO v3 Architecture



PTO v3 Provenance



/raw

- Each campaign is an resource, representing its metadata
  - e.g. `/raw/my_campaign`
- Each file is a resource representing its metadata
  - e.g. `/raw/my_campaign/my_file.ext`
  - Files inherit, override metadata from their campaigns
- File contents are a separate resource under the file
  - e.g. `/raw/my_campaign/my_file.ext/data`
- All content is immutable once uploaded
  - Deletion requires manual intervention (and isn't implemented yet)

/obs

- Each observation set is an resource, representing its metadata, identified by its serial number
  - e.g. `/obs/0000000000003a`
- Observation contents are a separate resource under the observation set, represented as NDJSON
  - e.g. `/obs/0000000000003a/data`
- Observation content is immutable once uploaded

Obervation file format

- each observation is a JSON string array:
- [set_id, time_start, time_end, path, condition, value]
  - set_id in hex (just like IDs in links)
  - time_start, time_end in RFC 3339 format UTC
  - path is a space-separated element list
  - condition is a condition name
  - value is ~~an integer~~ a string ***(todo)***

Local analyzer interface

- Analyzers are simply UNIX executables
  - Normalizer: raw to obs
    - raw data on stdin
    - raw metadata on filehandle 3
    - observations and metadata on stdout
  - Analyzer: obs to obs
    - observations and metadata on stdin
    - observations and metadata on stdout

ptocat, ptoload, ptonorm

- `ptocat`: dump observation set +metadata on stdout
- `ptoload`: load observation set + metadata from stdin
- running an analyzer: `$ ptocat` *set-ids* | *analyzer* | `ptoload`

- `ptonorm`: read raw data and metadata, set up on stdin + fh3, read observation data into db.
- running a normalizer: `$ ptonorm` *analyzer campaign/file*

- `ptopass`: simple normalizer to pass raw observation files into the database

/query

- Each query is identified by a hash of its normalized parameters
- `/query`: list queries in the cache
  - `pending`: queries still waiting for execution, or executing
  - `completed`: queries that are done, not yet purged
  - `permanent`: queries that are externally referenced and will not be purged
- `/query/submit`: submit a query given params
  - idempotent: a given set of params will only run once
- `/query/`*query-id*: read, update query metadata
- `/query/`*query-id*`/results`: read/paginate results

/query/submit parameters

- Select parameters use and-of-or semantics
  - `time_start, time_end`: temporal scope, required
  - `set`: observation set scope
  - `conditions`: select by condition, with wildcards
  - `source, on_path, target`: select by path element
    - ***missing***: query by observation set metadata
- `group` parameters change query to count from selected observations by group
  - `year, month, week, day, hour, week_day, day_hour:` time series grouping
  - `condition`
  - `source, target`
- `options` change how queries are interpreted
  - `sets_only`: only return links to sets having observations matching query; used for finding sets to feed into an analysis
  - ***missing***: option to count targets, not observations

Status

- Everything in this presentation *should* work,
  in `master` branch of https://github.com/mami-project/pto3-go
  - (caveat: test coverage less than 60%)

- New observatory API running at https://v3.pto.mami-project.eu by 8 Feb

- Now: loading 2017 ANRW and various ETH Pathspider measurements.
  - pto is metadata agnostic, but we need conventions for MAMI data
- Next: migration of the rest of the raw data, analysis into obsets
- Later: update the front-end, beta release.
  - Issues list: https://github.com/mami-project/pto3-go/milestone/3