BGPSTREAM

A framework for BGP data analysis

Alberto Dainotti, **Alistair King**, Chiara Orsini, Vasco Asturiano
alistair@caida.org

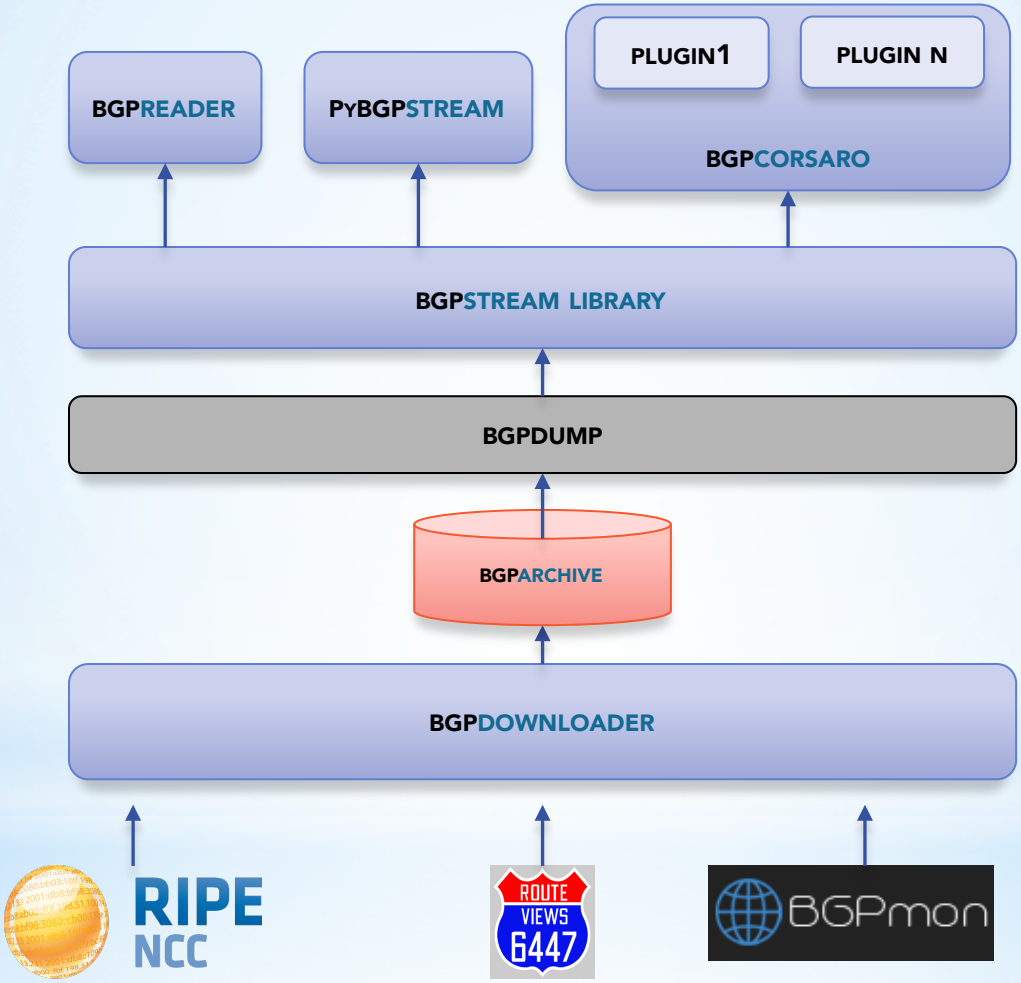
The Problem

* Lack of tools for efficient analysis of large volumes of BGP data
* BGPdump is the de-facto standard
  * Lightly-maintained; low-level deserialization of MRT data
* Processing historical data requires (semi-)manual download and curation of data
* Processing across time/collectors/types requires custom demux code
  * Identifying correct files, sorting of records/types
* No tools available for near-realtime/streaming analysis

BGPSTREAM

- Framework for historical analysis and real-time monitoring of BGP data
- Set of tools, libraries, and interfaces
  - C API
  - Python Bindings
  - ASCII-output command-line tool
  - Modular interval-driven processing tool
- Work in progress. Soon to be released as open-source
  - v1 release planned (summer 2015)
  - Beta code/access available upon request

BGPSTREAM framework



Data Feeds

Transparent access to different MRT data sources:

1. Previously-downloaded **local files**
2. Historical and continuous download
   * RIBs and updates from RouteViews and RIPE RIS projects
3. Real-time streams
   * Colorado State's BGPmon (RouteViews collectors) *[work-in-progress for release v1]*
   * *RIPE RIS*
     [discussion in progress]

Data Feeds

Transparent access to different MRT data sources:

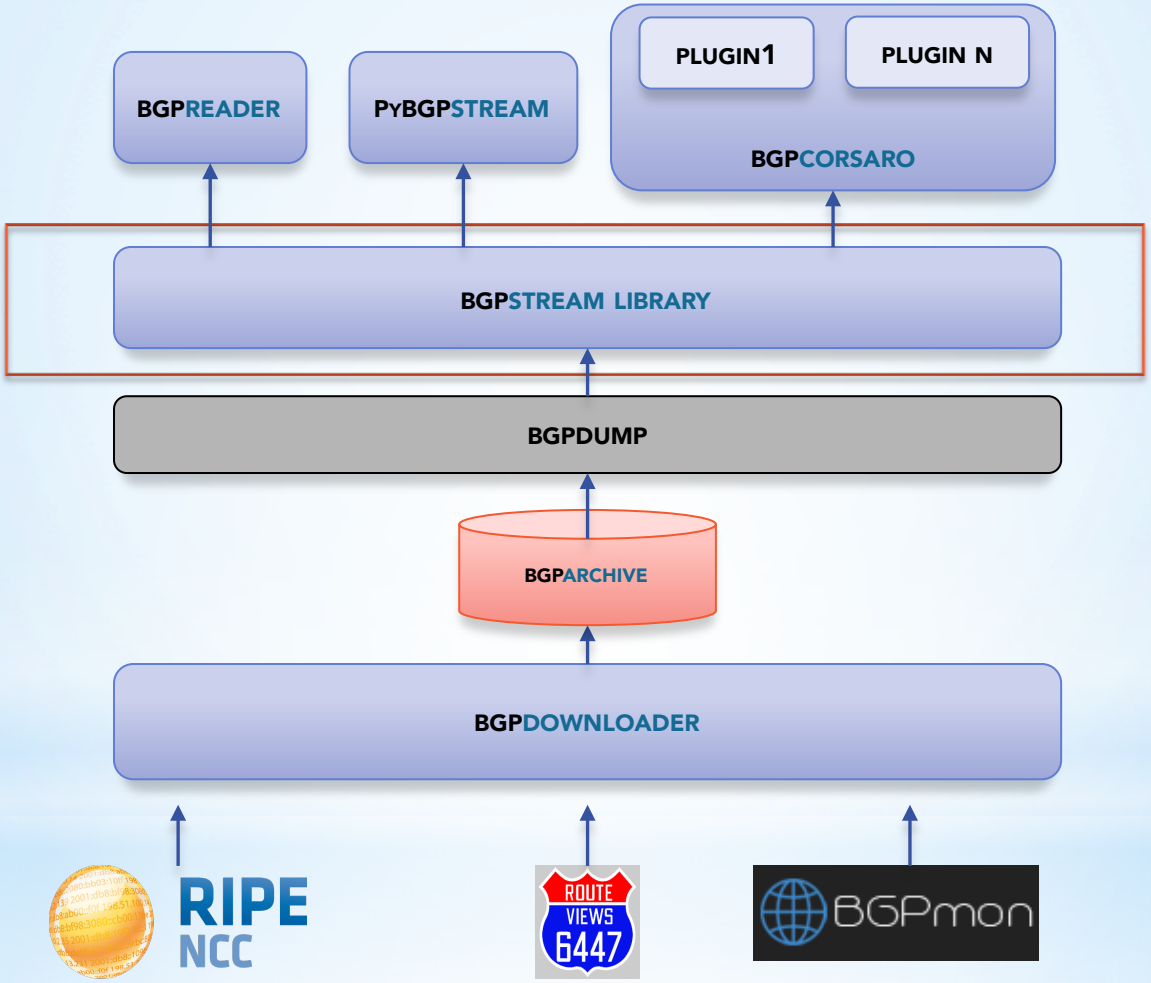
1. Previously-downloaded local files
2. **Historical and continuous download**
   * RIBs and updates from RouteViews and RIPE RIS projects
3. Real-time streams
   * Colorado State's BGPmon (RouteViews collectors) *[work-in-progress for release v1]*
   * *RIPE RIS*
     [discussion in progress]

Data Downloader

* *Polls RouteViews and RIS websites, downloads new data as it is published.*
* *'Normal' latency of <20mins from capture to usability, but:*
  * *RIS* and *Routeviews* have different delay profiles.
  * *RIB* and *update* delays are different
  * *RIS update* delays have some recurring phenomena
* On average we expect data availability after:

	RIBs	Updates
RIS	11.5 min	7.5 min
Routeviews	6.5 min	16.8 min

BGPSTREAM framework

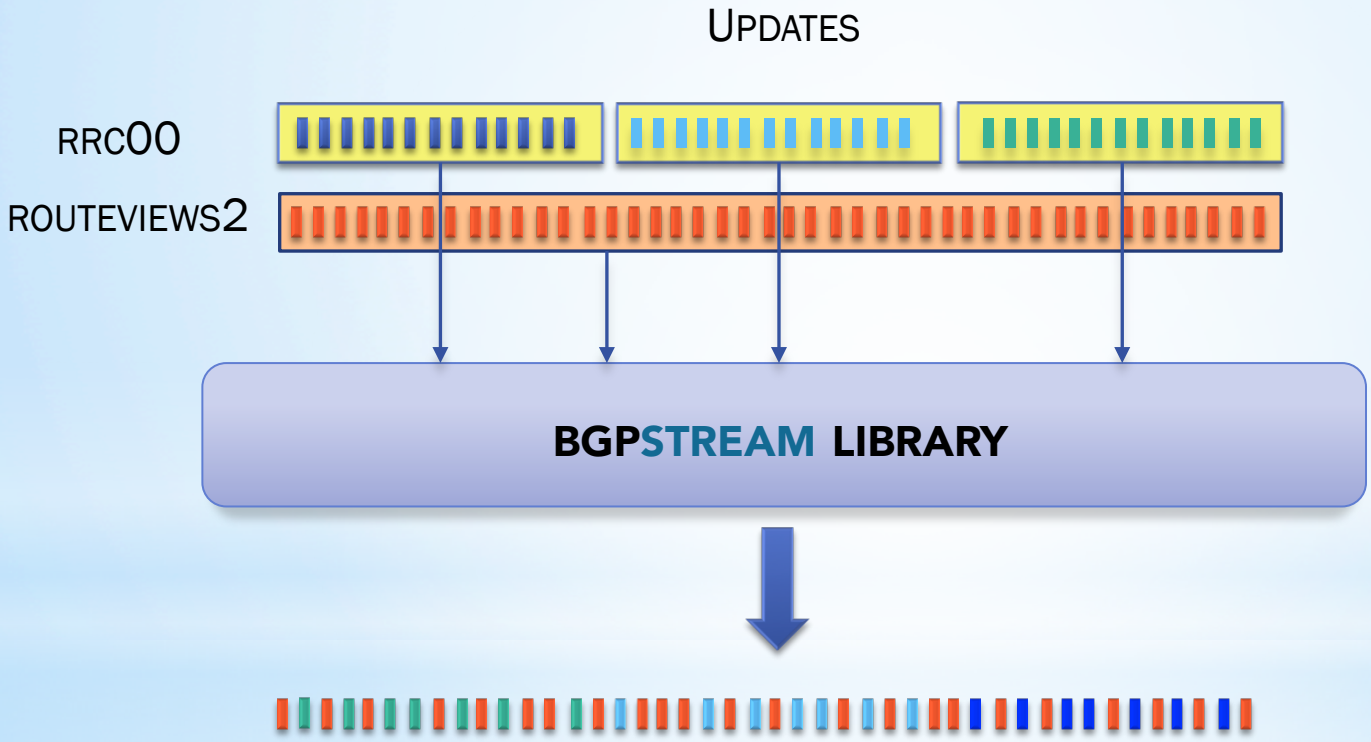


BGPSTREAM library

- C library providing a **sorted stream** of **BGPRECORDS**
- Transparently combines data sources from different projects/collectors/types
- Hides data source details/management from users
- Metadata **filters** to select subset of data (time/collector/type etc.)
- Identifies unreliable MRT data
- Supports **real-time processing**

BGPSTREAM library

* How does **BGPSTREAM** sort heterogeneous data?



* use metadata to decide how many dumps to open in parallel

* sort based on **BGPRECORD** time

BGPRECORD

* PROJECT
* BGP TYPE
* COLLECTOR
* DUMP TIME
* DUMP POSITION
* RECORD TIME
* RECORD STATUS
* **BGPDUMP ENTRY**

BGPARCHIVE metadata (common to entire dump)

position of entry in dump
- START
- MIDDLE
- END

time associated with the **BGPDUMP ENTRY**

status of **BGPRECORD**
- VALID
- CORRUPTED RECORD
- EMPTY SOURCE
- CORRUPTED SOURCE

set of MRT formatted entries

Hello **BGPSTREAM** World!

```
#include "bgpstream.h"

int main(int argc, char *argv[])
{
  bgpstream_t * bs = bgpstream_create();
  bgpstream_record_t *rec = bgpstream_create_record();
```

Allocate memory

```
  bgpstream_start(bs);
```

Start interface

```
  while(bgpstream_get_next_record(bs, rec) > 0)
  {
     // [[ USE BGPRECORD HERE ]]
  }
```

Pull bgprecords

```
  bgpstream_stop(bs);
```

Stop interface

```
  bgpstream_destroy_record(rec);
  bgpstream_destroy(bs);
```

Deallocate memory

```
  return 0;
}
```

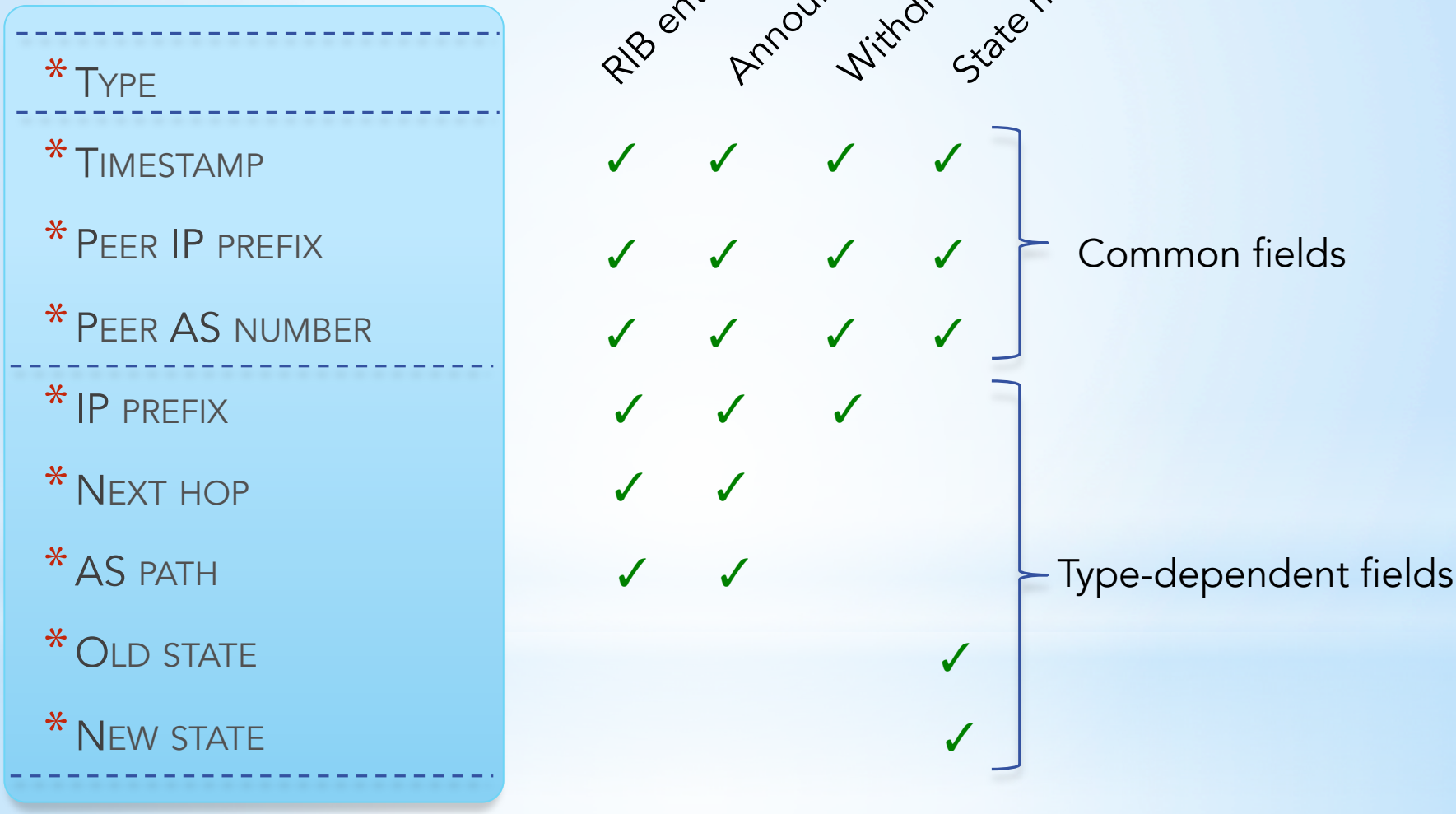
BGPSTREAM filters

```
...
  bgpstream_t * bs = bgpstream_create();

  bgpstream_add_filter(bs, BS_PROJECT, "routeviews");

  bgpstream_add_filter(bs, BS_COLLECTOR, "route-views2");

  bgpstream_add_filter(bs, BS_COLLECTOR, "route-views.linx");

  bgpstream_add_filter(bs, BS_BGP_TYPE, ”updates");

  bgpstream_add_interval_filter(bs, BS_TIME_INTERVAL,
                                "1410285571", ──→ Tue, 09 Sep 2014 17:59:31 UTC
                                "1412877600"); ──→ Thu, 11 Sep 2014 00:32:51 UTC

  bgpstream_init(bs);
...
```

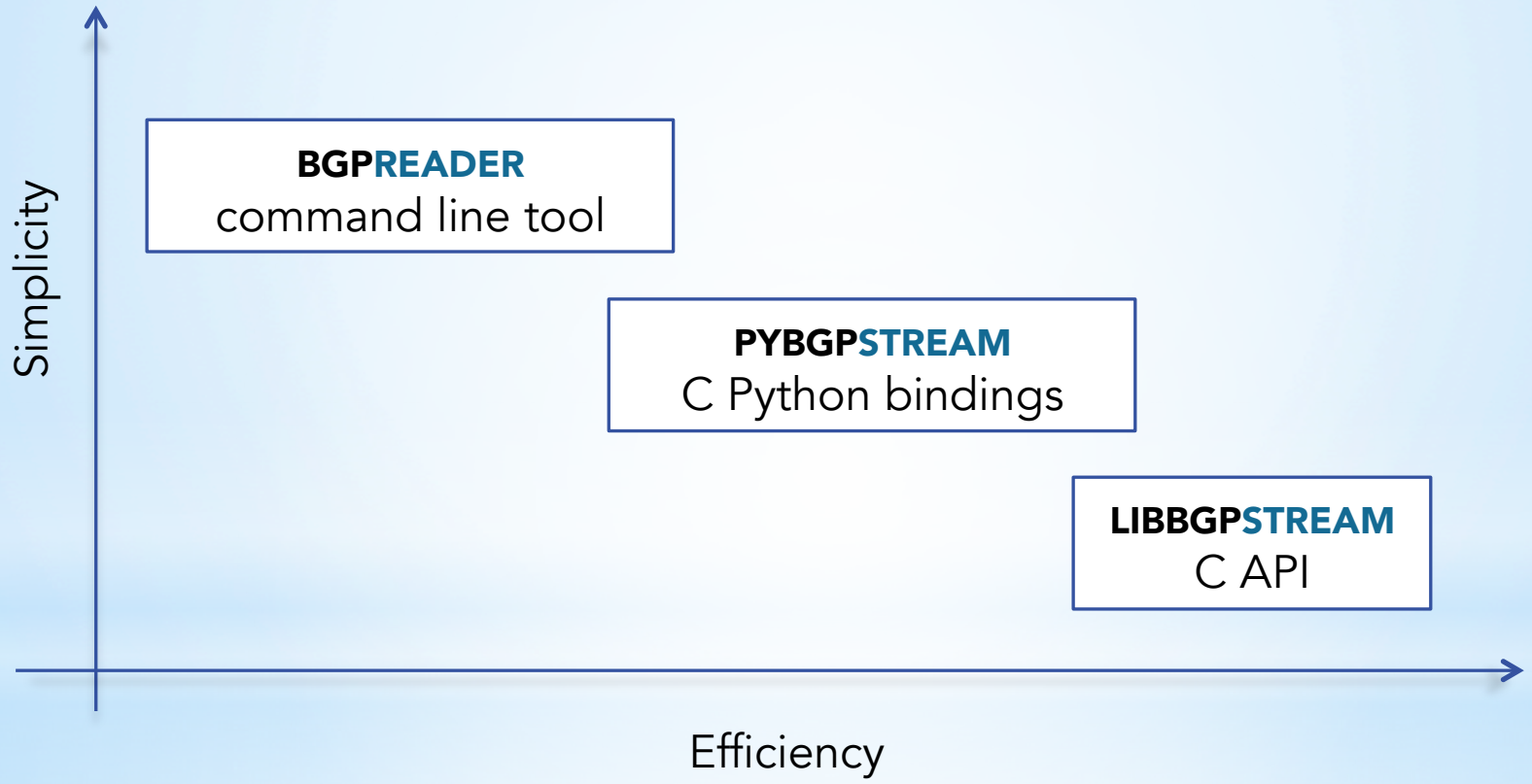
BGPRECORD → BGPELEM

* PROJECT
* BGP TYPE
* COLLECTOR
* DUMP TIME
* DUMP POSITION
* RECORD TIME
* RECORD STATUS
* **BGPDUMP ENTRY**

BGPELEM

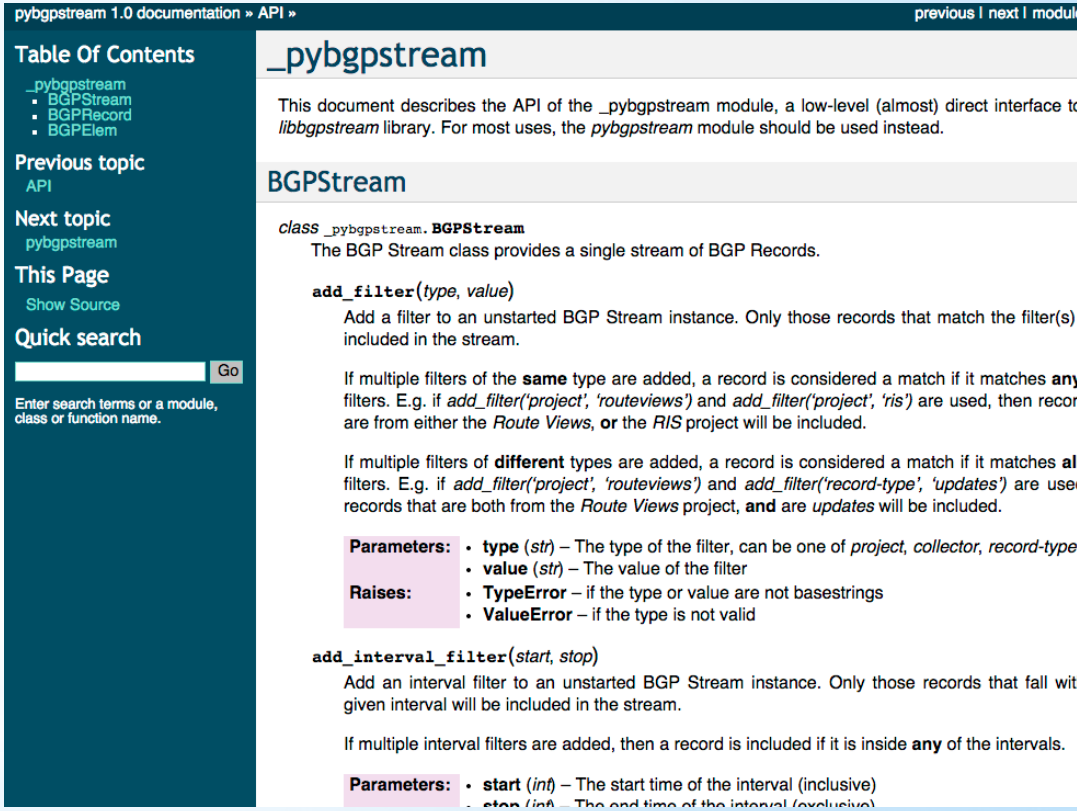

* TYPE
* TIMESTAMP
* PEER IP PREFIX
* PEER AS NUMBER
* IP PREFIX
* NEXT HOP
* AS PATH
* OLD STATE
* NEW STATE
RIB entry
Announcement
Withdrawal
State message
✓ ✓ ✓ ✓
✓ ✓ ✓ ✓
✓ ✓ ✓ ✓
✓ ✓ ✓
✓ ✓
✓ ✓
✓
✓
Common fields
Type-dependent fields

BGPSTREAM just C?



PYBGPSTREAM demo

- Python C bindings
- Same API as C (almost)
- No functionalities are lost
- Great for prototyping, experimental analysis



pybgpstream 1.0 documentation » API »

previous | next | modul

Table Of Contents

_pybgpstream
- BGPStream
- BGPRecord
- BGPElem

Previous topic
API

Next topic
pybgpstream

This Page
Show Source

Quick search
Go

Enter search terms or a module, class or function name.

_pybgpstream

This document describes the API of the _pybgpstream module, a low-level (almost) direct interface t
libbgpstream library. For most uses, the *pybgpstream* module should be used instead.

BGPStream

class _pybgpstream.**BGPStream**

The BGP Stream class provides a single stream of BGP Records.

add_filter(*type*, *value*)

Add a filter to an unstarted BGP Stream instance. Only those records that match the filter(s) included in the stream.

If multiple filters of the **same** type are added, a record is considered a match if it matches **any** filters. E.g. if *add_filter('project', 'routeviews')* and *add_filter('project', 'ris')* are used, then recor are from either the *Route Views*, **or** the *RIS* project will be included.

If multiple filters of **different** types are added, a record is considered a match if it matches **al** filters. E.g. if *add_filter('project', 'routeviews')* and *add_filter('record-type', 'updates')* are use records that are both from the *Route Views* project, **and** are *updates* will be included.

Parameters:
- **type** (*str*) – The type of the filter, can be one of *project*, *collector*, *record-type*
- **value** (*str*) – The value of the filter

Raises:
- **TypeError** – if the type or value are not basestrings
- **ValueError** – if the type is not valid

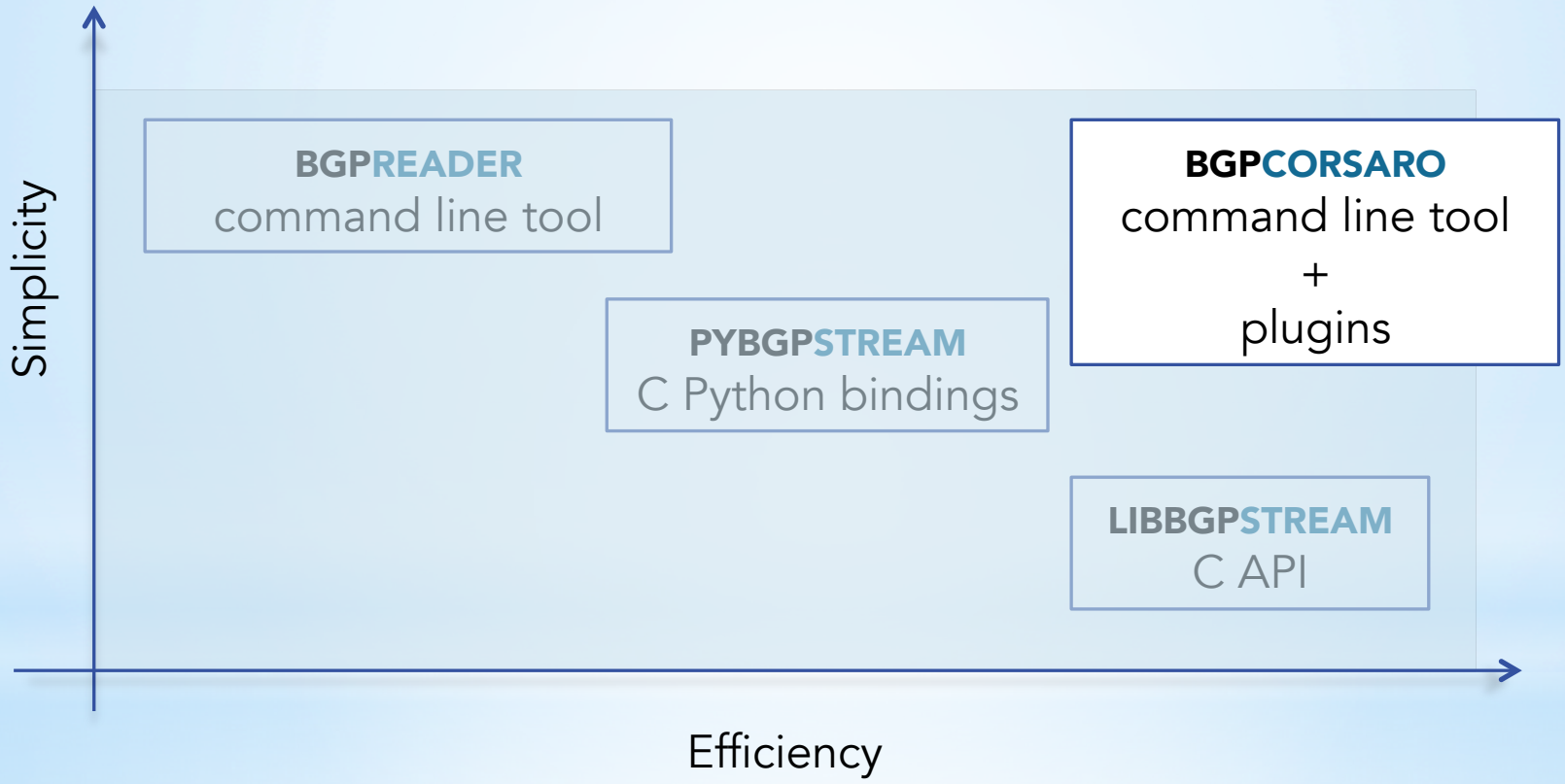
add_interval_filter(*start*, *stop*)

Add an interval filter to an unstarted BGP Stream instance. Only those records that fall wit given interval will be included in the stream.

If multiple interval filters are added, then a record is included if it is inside **any** of the intervals.

Parameters:
- **start** (*int*) – The start time of the interval (inclusive)
- **stop** (*int*) – The end time of the interval (exclusive)

BGPSTREAM just C?

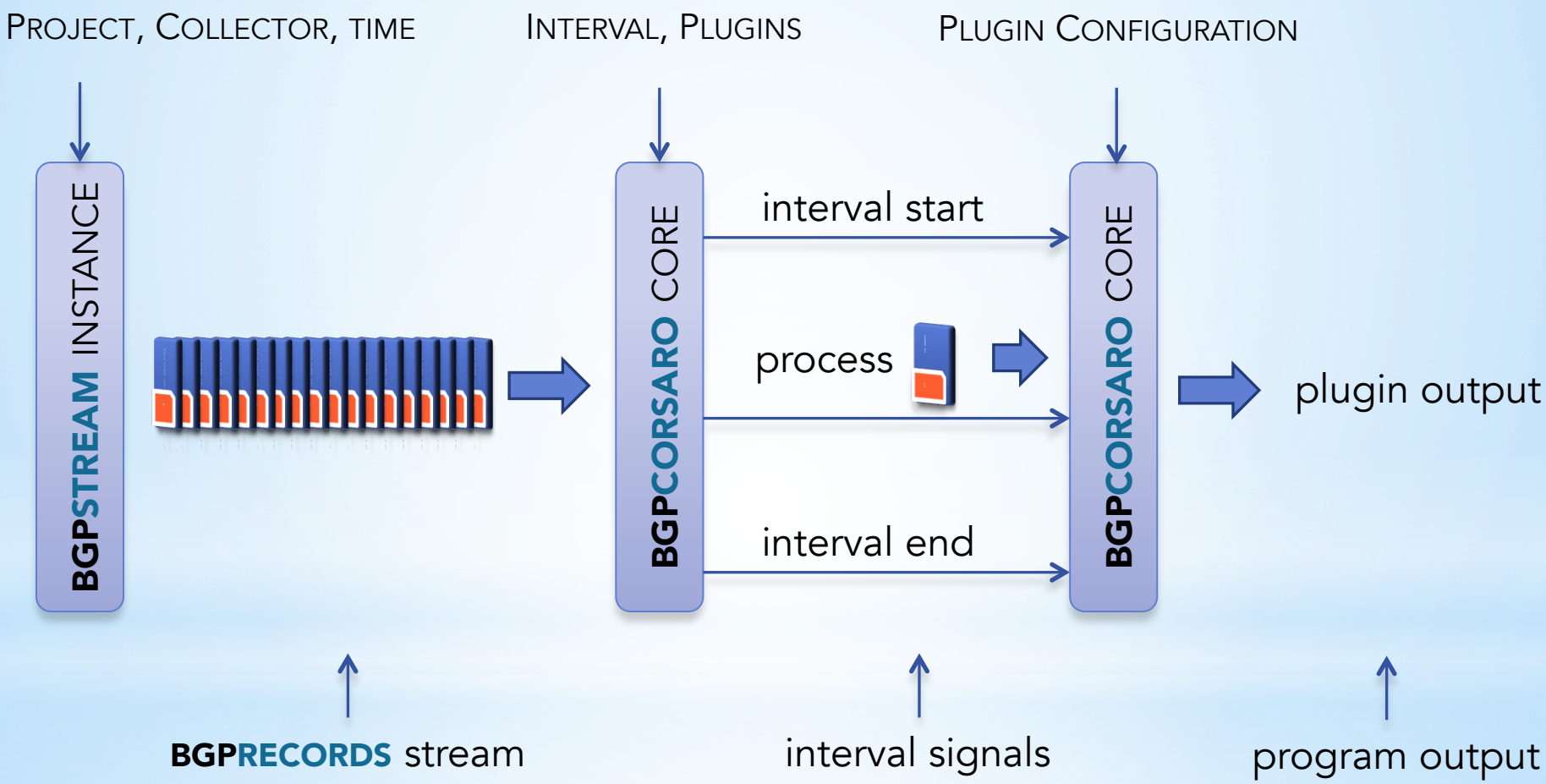


BGPCORSARO

* C tool that transforms a stream of **BGPRECORDS** into a set of structures and metrics representative of specific time intervals
  * interval driven tool
  * modular architecture based on plugins
* a fork of **CORSARO** [2] that operates on **BGPRECORDS** rather than **LIBTRACE** packets

[2] http://www.caida.org/tools/measurement/corsaro/

BGPCORSARO architecture



BGPCORSARO how to write a plugin?

```
# lib/plugins/bgpcorsaro_myplugin.c
```

process START of interval signal

```
int
bgpcorsaro_myplugin_start_interval(bgpcorsaro_t *bgpcorsaro,
                                   bgpcorsaro_interval_t *int_start)
```

process record

```
int
bgpcorsaro_myplugin_process_record(bgpcorsaro_t *bgpcorsaro,
                                   bgpcorsaro_record_t *record)
```

process END of interval signal

```
int
bgpcorsaro_myplugin_end_interval(bgpcorsaro_t *bgpcorsaro,
                                 bgpcorsaro_interval_t *int_end)
```

CAIDA framework for real-time outage detection

⑥ Compute global metrics

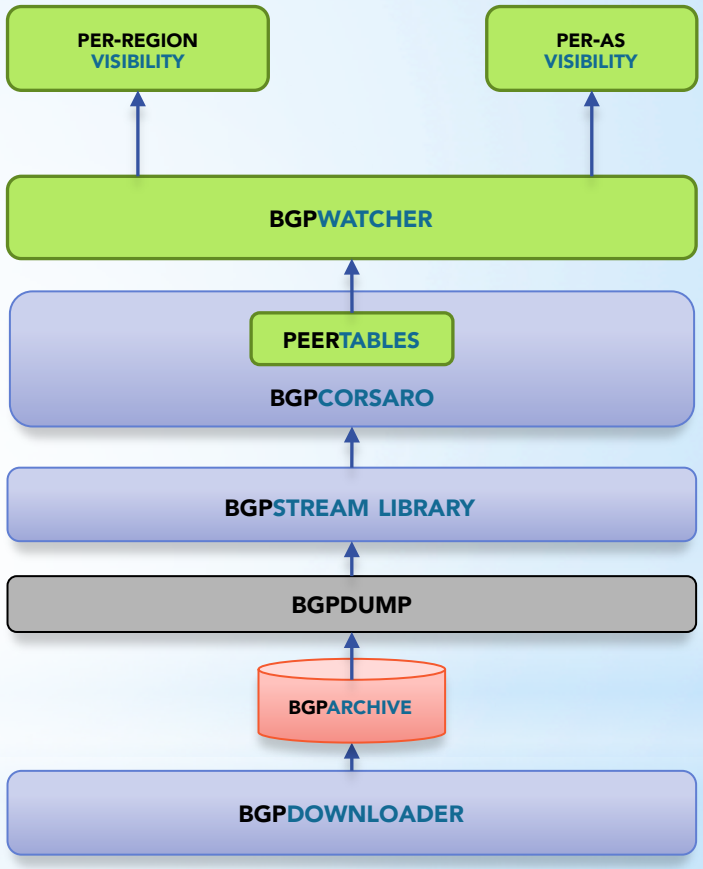
⑤ Combine routing tables as seen by different peers

④ Derive the status of each peer

③ Sample routing properties over time

② Manage data heterogeneity

① Get data

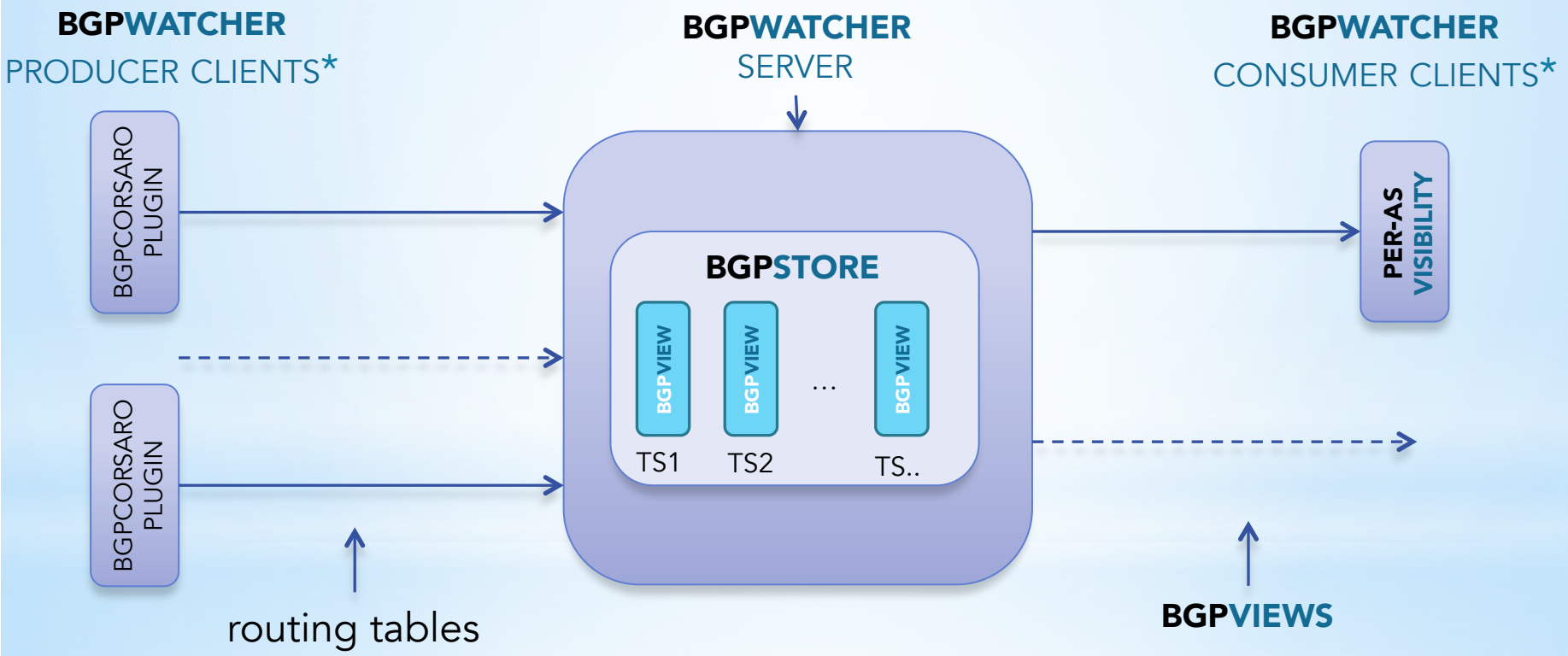


CAIDA framework for real-time outage detection

- Real-time challenges:
  - collectors delay varies a lot:
    - project constraints
    - per collector differences
  - the computational load of each collector varies too
- Processing challenges
  - we need to process BGP data faster than real time in order to keep up with the flow

NSF CNS-1228994 PIs: Claffy, Dainotti

CAIDA framework for real-time outage detection



NSF CNS-1228994 PIs: Claffy, Dainotti

Interested?

* Code is stable and pre-release version is available, but bootstrapping a deployment is not trivial
* Production deployment here at CAIDA
  * Talk to us about getting beta access
* First public release this summer, will support on-demand streaming from BGPmon archives
* Talk to me about hands-on tutorial (Friday morning?)

ANY BGPQUESTIONS?

Alistair King
alistair@caida.org