CTAPIPE:

experimental framework for CTA event data processing

github.com/cta-observatory/ctapipe

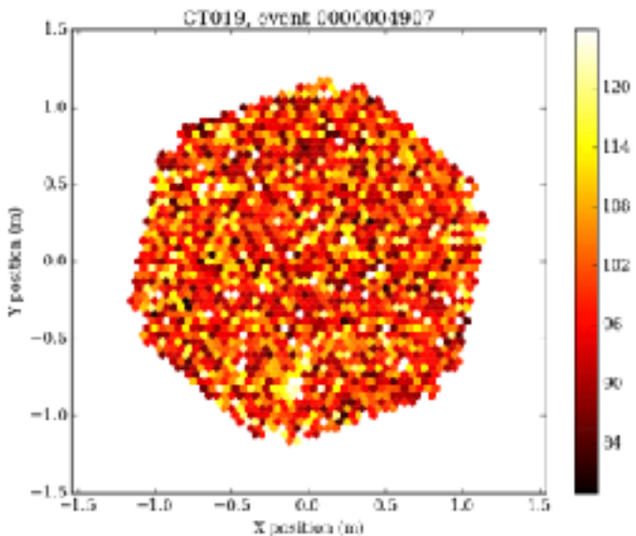
Goal

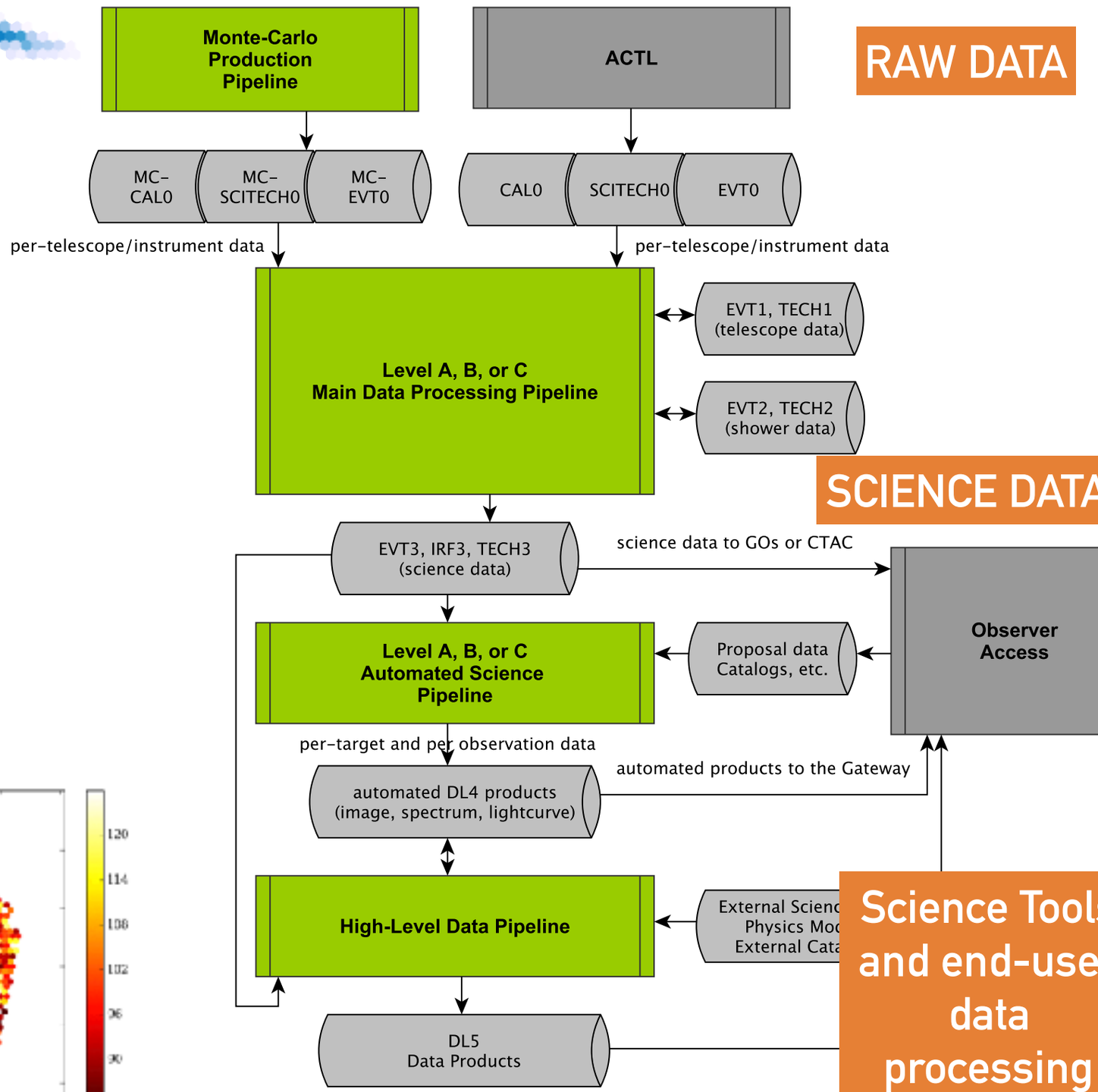
Process raw data :

- images of air-showers produced by gamma-rays or cosmic rays, seen by an array of telescopes
- Calibrate Camera data
- Process Camera Images
- Reconstruct showers
- Select gamma-like events

Produce science data products:

- event lists (photons + bg)
- instrument response tables
- These are then given to end-users to do science





[CTA Technical Design Report]

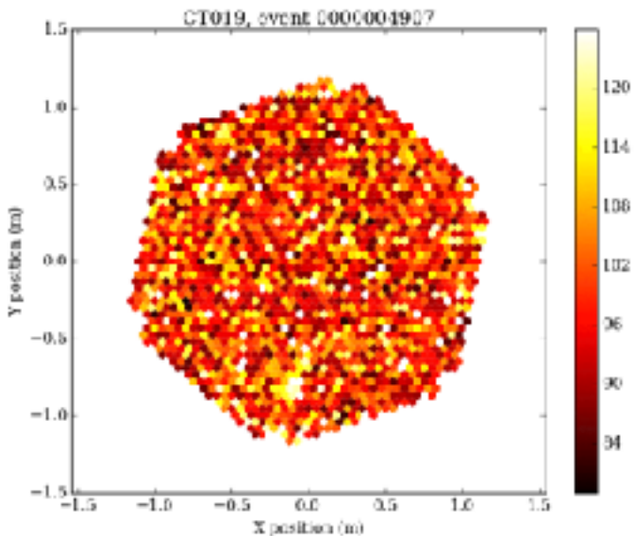
Goal

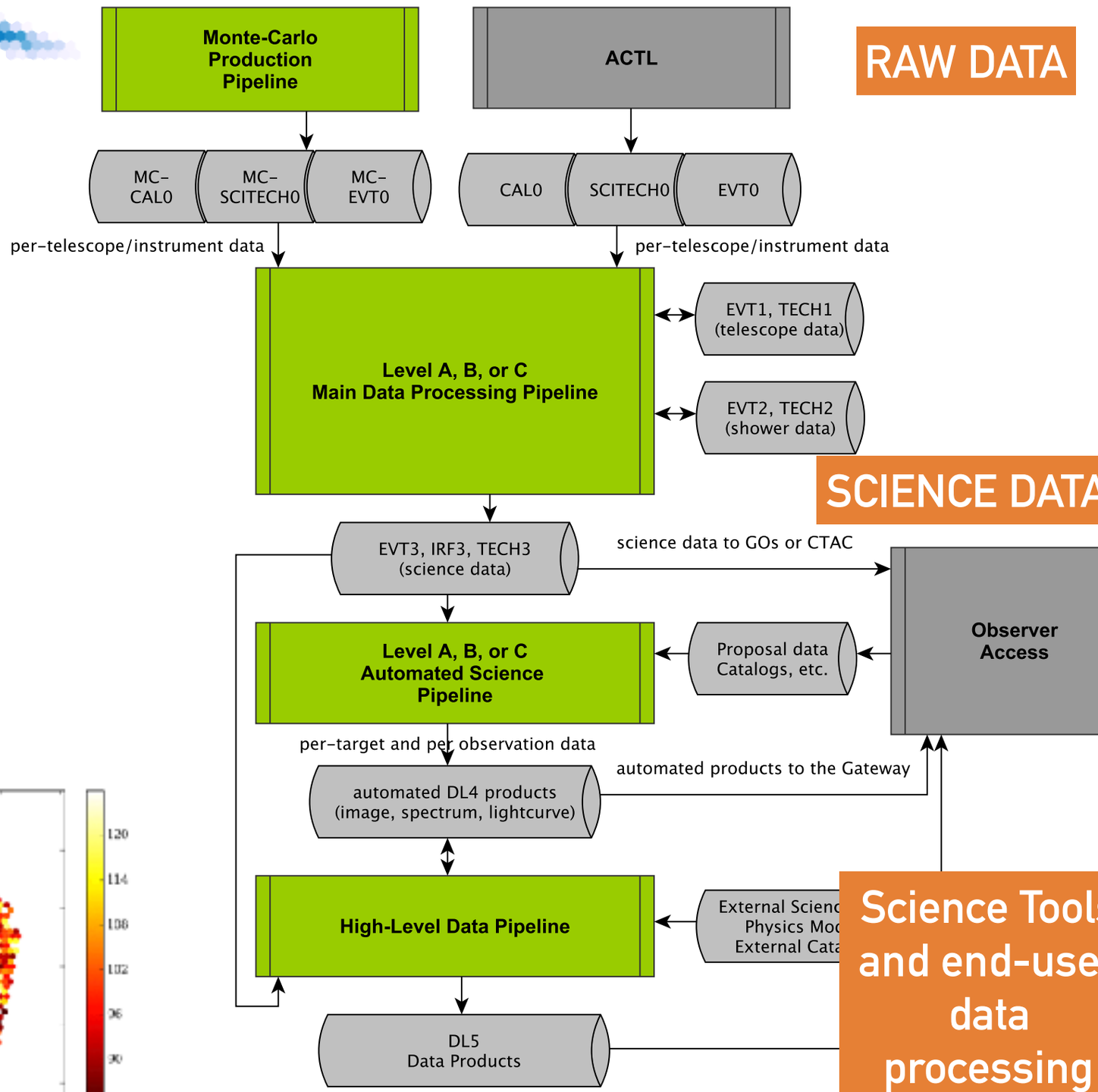
Process raw data :

- images of air-showers produced by gamma-rays or cosmic rays, seen by an array of telescopes
- Calibrate Camera data
- Process Camera Images
- Reconstruct showers
- Select gamma-like events

Produce science data products:

- event lists (photons + bg)
- instrument response tables
- These are then given to end-users to do science





[CTA Technical Design Report]

Planning

Lessons to incorporate:

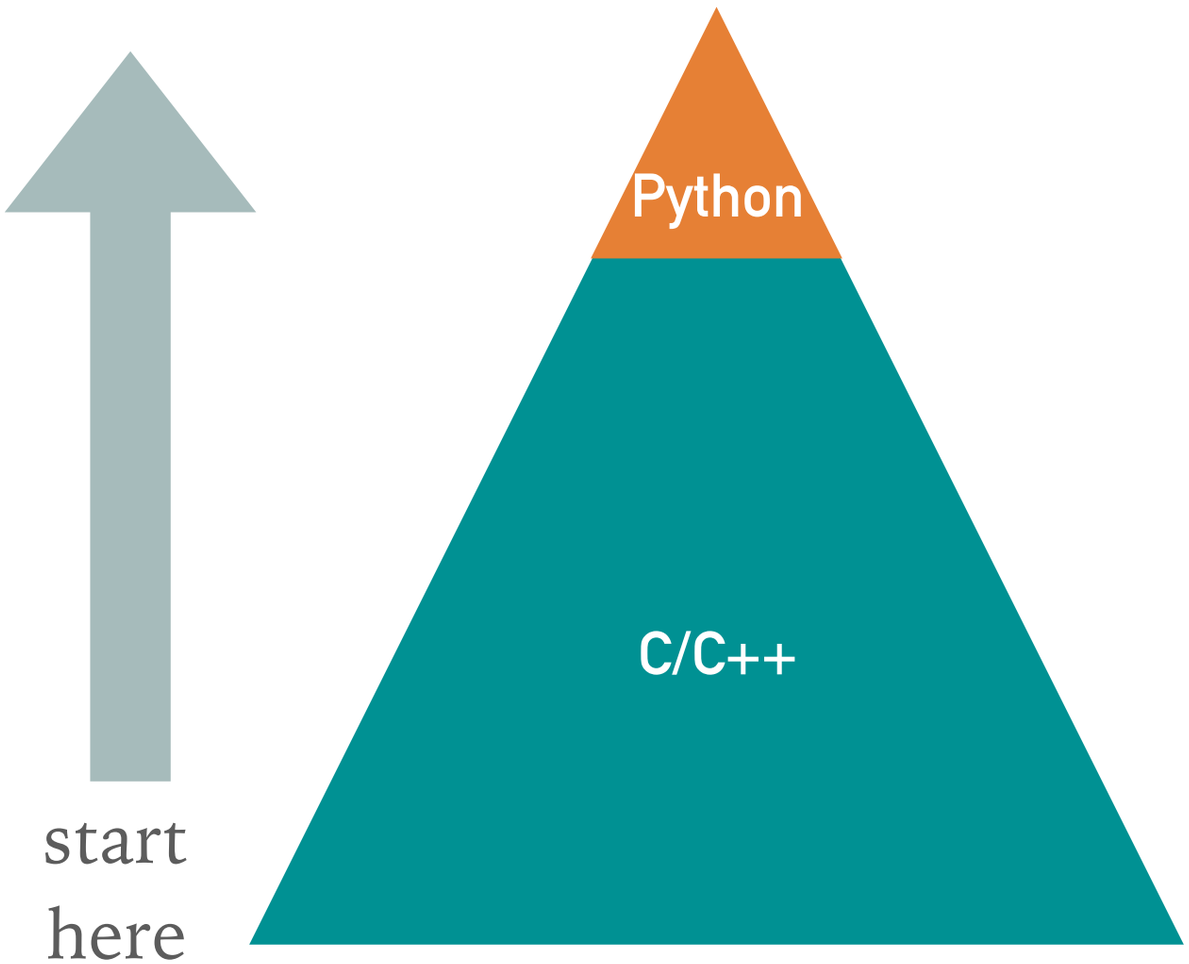
- Don't make too heavy a framework
- Make it easy to work with data interactively during development
- Make visualizations for algorithms

Python stack a good match!

But still a lot of questions…

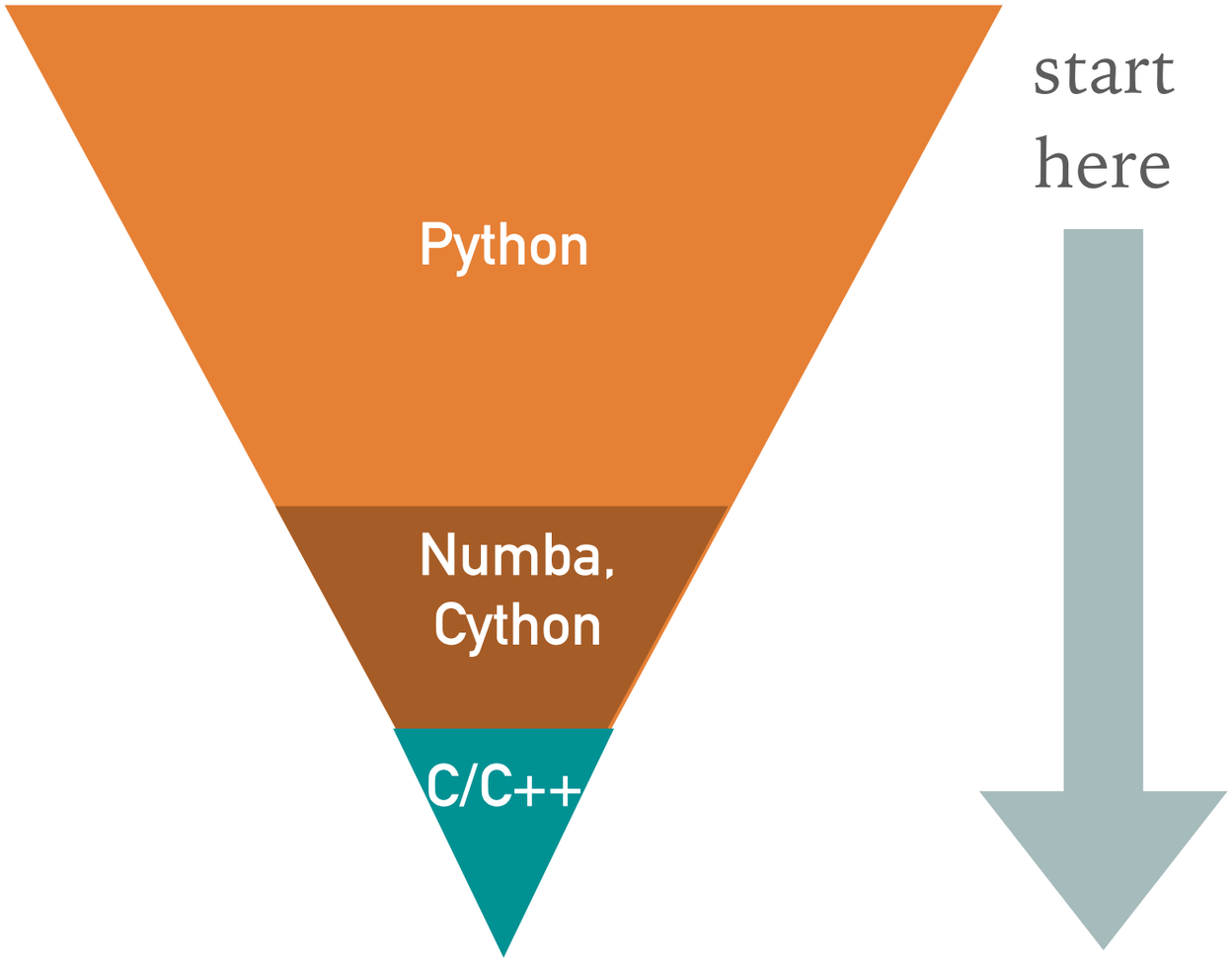
Building a Framework

Bottom-Up approach



Most current frameworks did it this way (if they use python at all)

Top-Down approach



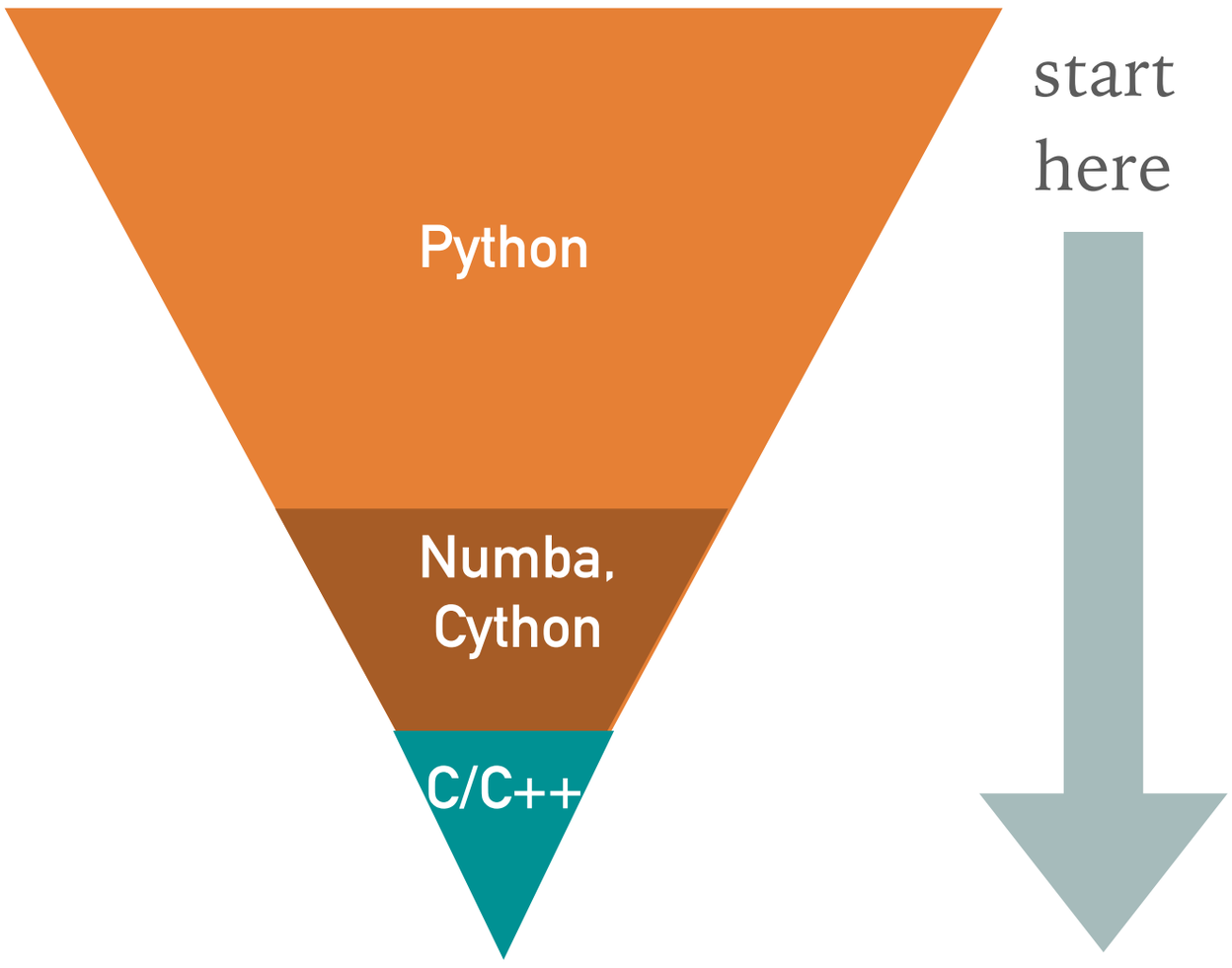
Our approach: start early with python and high-level API

Building a Framework

Bottom-Up approach

Most current frameworks did it this way (if they use python at all)

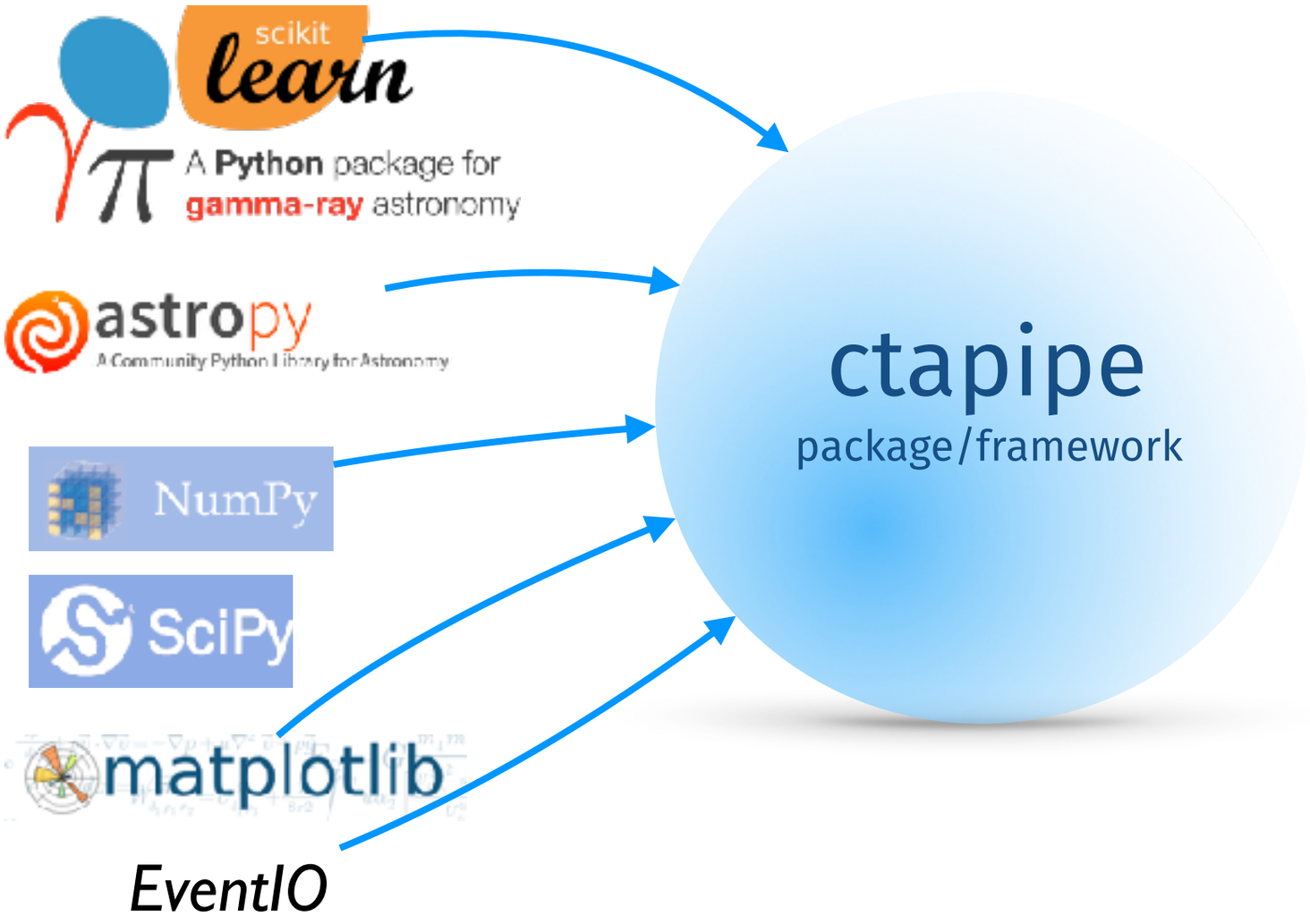
Top-Down approach



Our approach: start early with python and high-level API

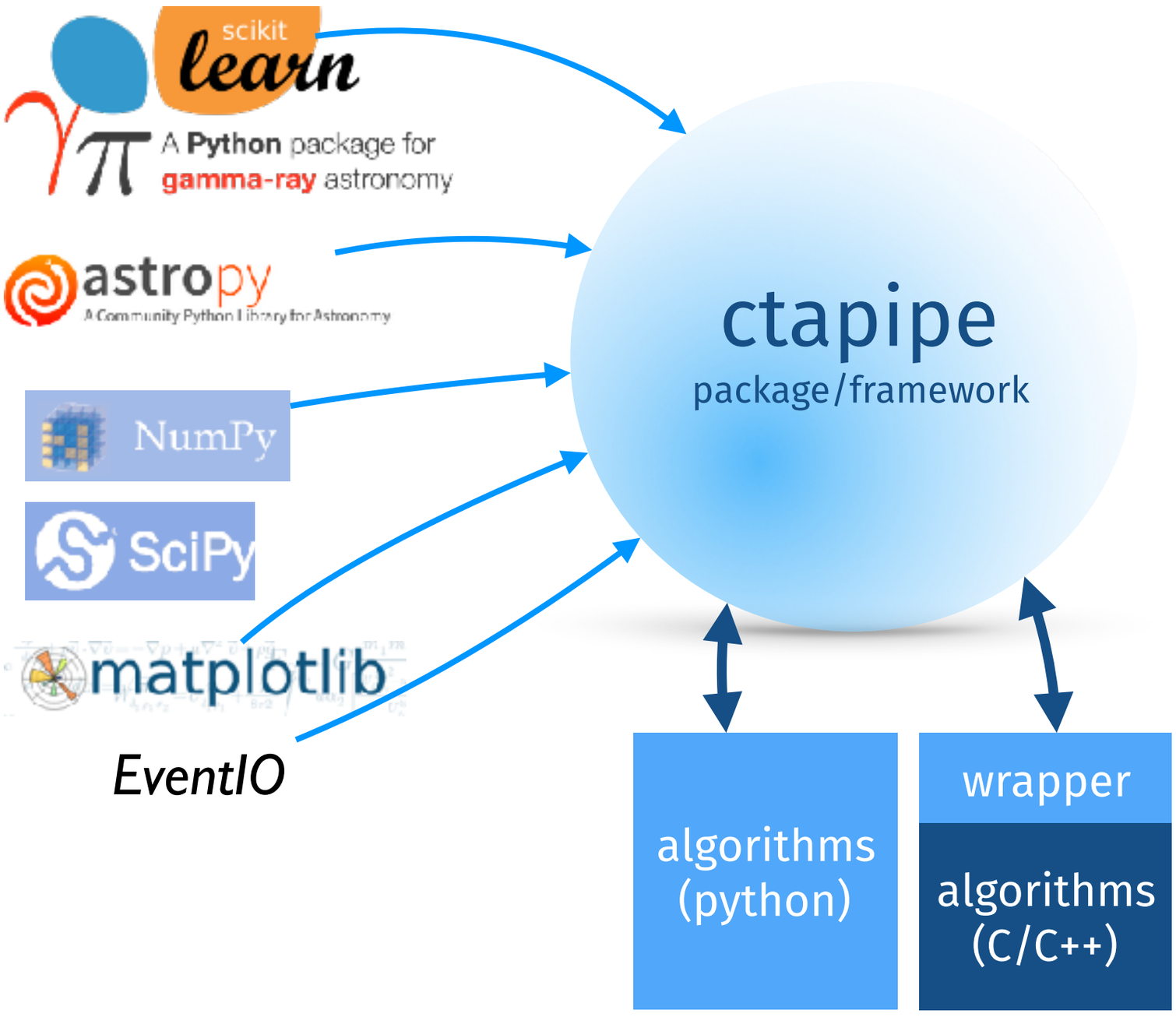
common “core” package

ctapipe will be **glue** between various components.
Provides common APIs and user interfaces
packaging, etc**.**



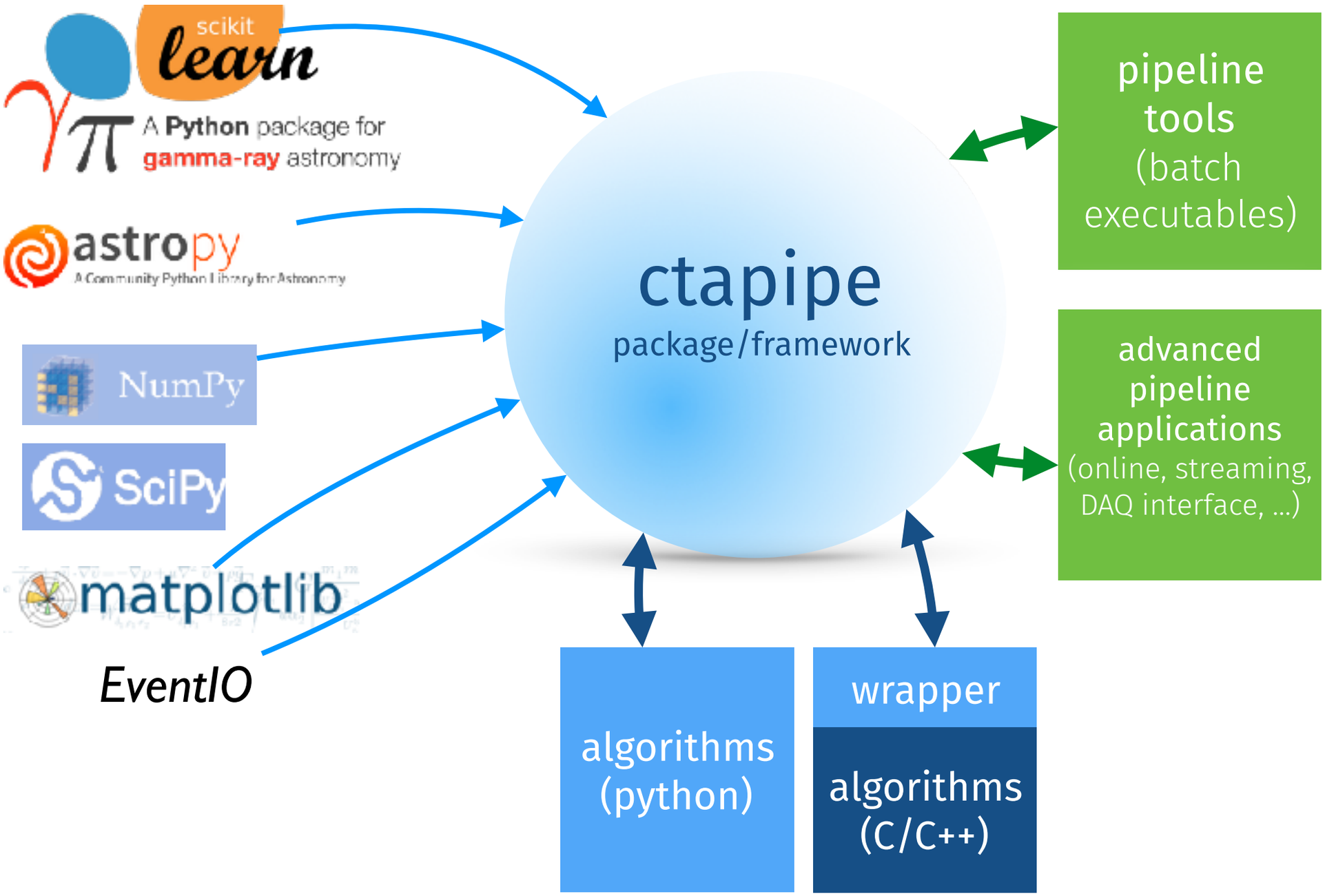
common “core” package

ctapipe will be **glue** between various components.
Provides common APIs and user interfaces
packaging, etc**.**



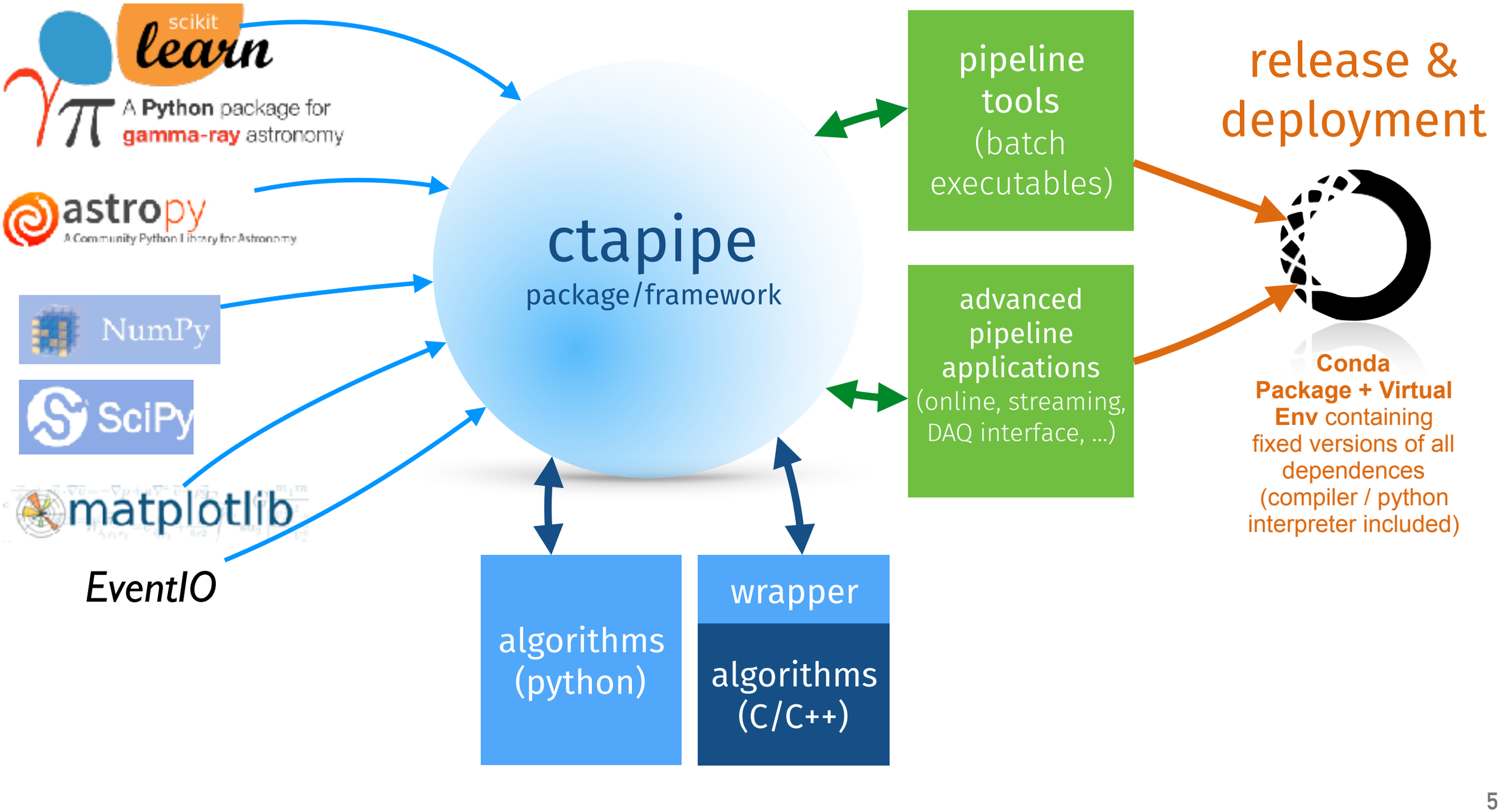
common “core” package

ctapipe will be **glue** between various components.
Provides common APIs and user interfaces
packaging, etc**.**

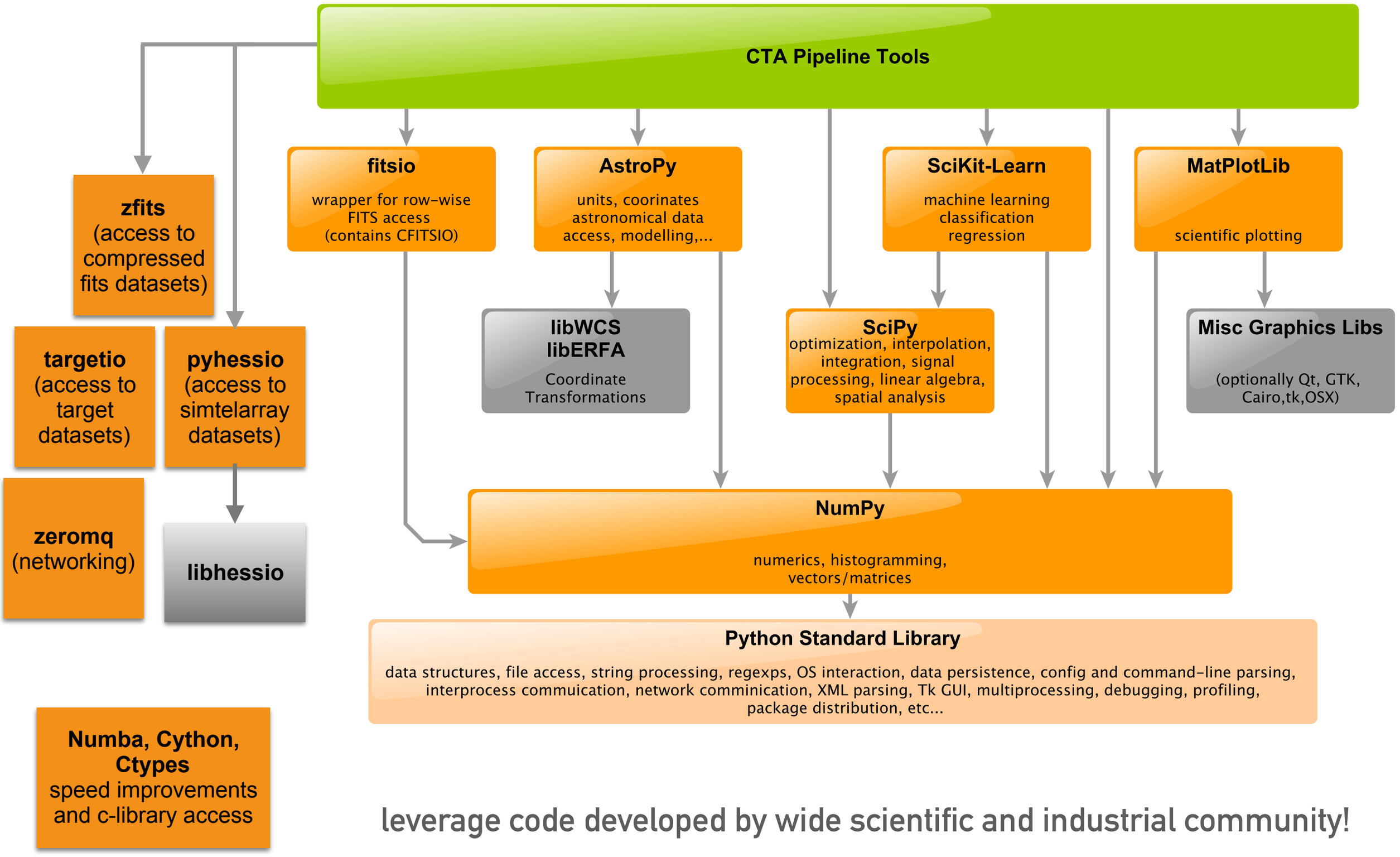


common “core” package

ctapipe will be **glue** between various components.
Provides common APIs and user interfaces
packaging, etc**.**

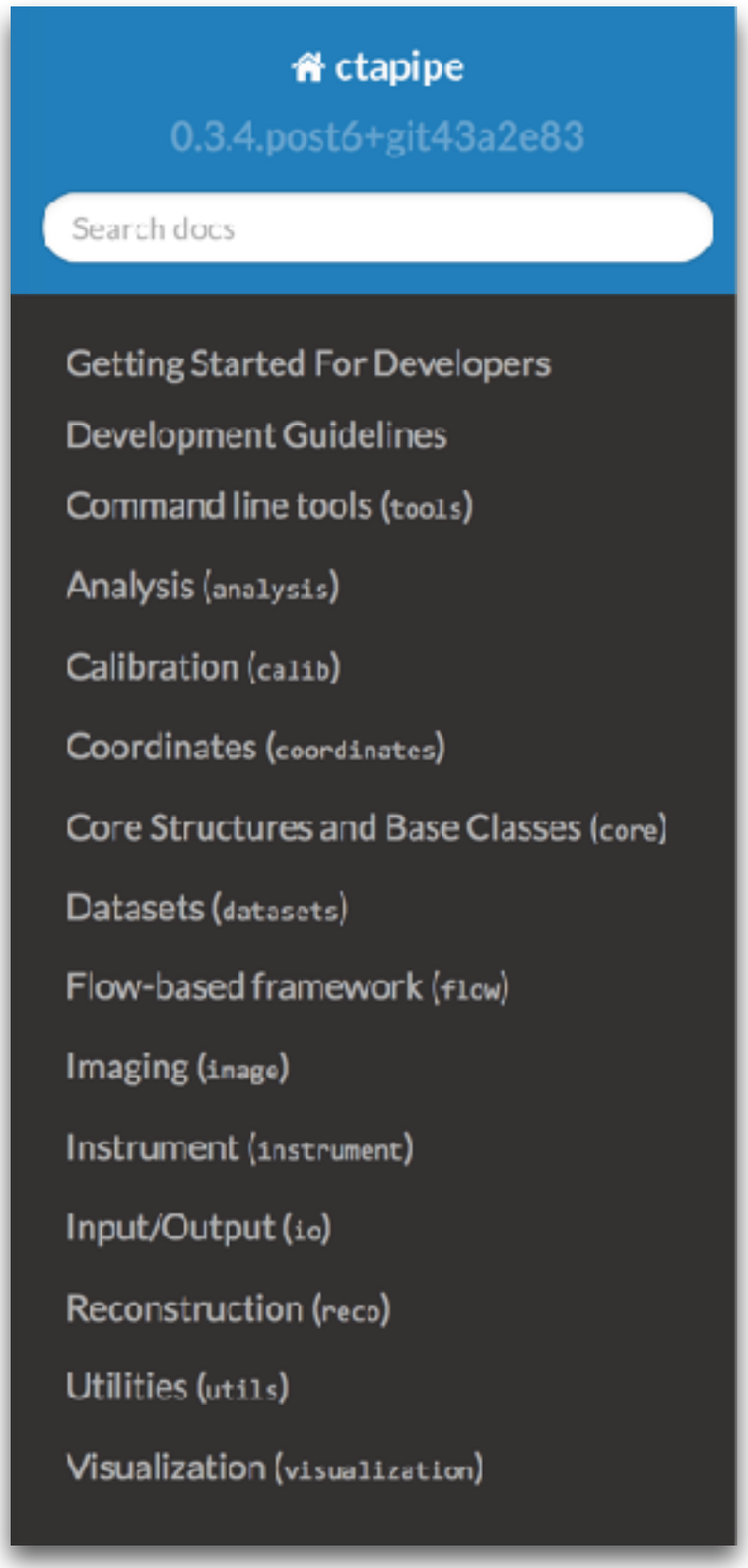


Core dependencies



leverage code developed by wide scientific and industrial community!

Modules



Main sub-packages: ctapipe.*

- **core**: base data structures
- **io**: event data access
- **tools**: command-line scripts
- **image**: Cherenkov image (cube) processing
- **reco**: shower reconstruction
- **utils**: histogramming and math
- **visualization**: camera and array displays

- **plotting**: higher-level plots (may merge with visualization)
- **instrument**: load instrument metadata

Core Data Structures

Container

- a class with *metadata* per element (name, type default, helpstring)
- nested hierarchies supported
- conversion to dict, flattened dict
- schema defined in class definition, attributes locked

Component

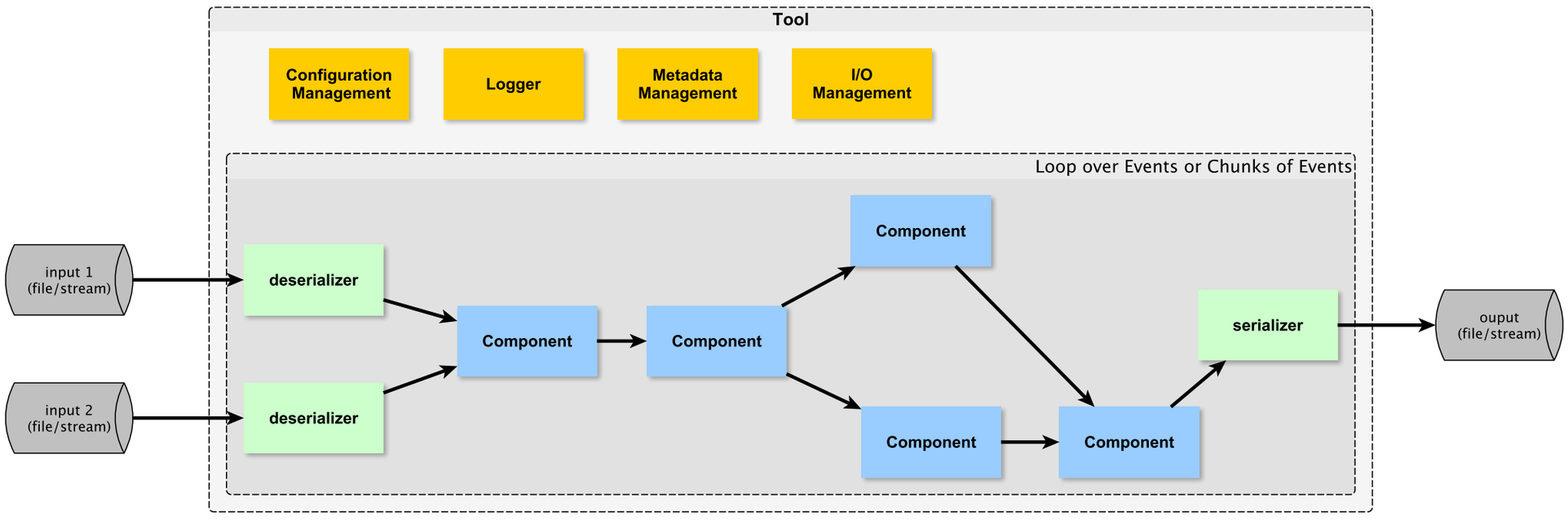
- Simple class that wraps an algorithm and handles algorithm configuration params
- currently based on **traitlets.config.Configurable**

Tool

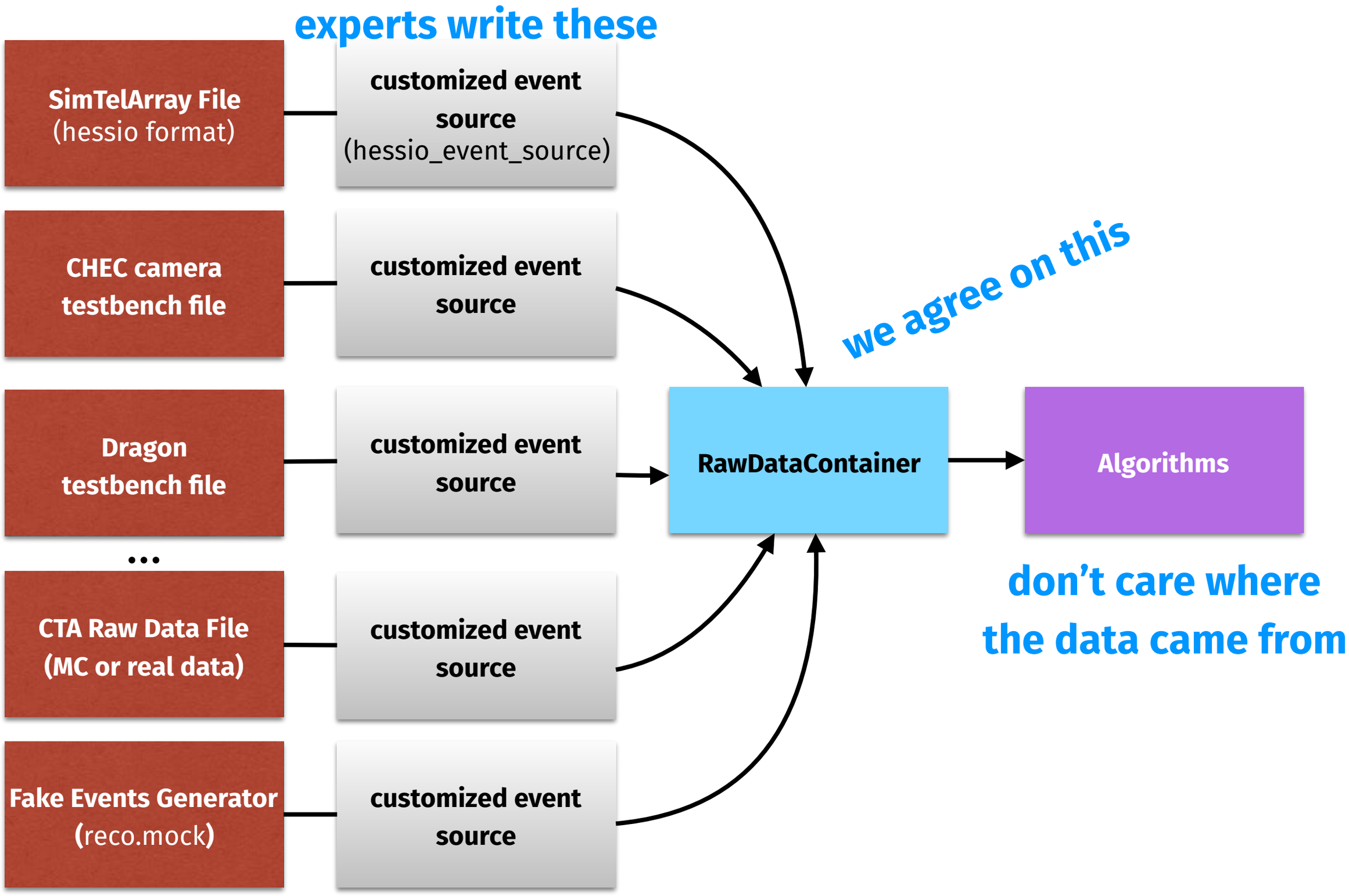
- command-line application base class
- currently based on **traitlets.config.Application** (may change)

```
: data.mc

: ctapipe.io.containers.MCEventContainer:
                    energy: Monte-Carlo Energy
                       alt: Monte-carlo altitude [deg]
                        az: Monte-Carlo azimuth [deg]
                    core_x: MC core position
                    core_y: MC core position
              h_first_int: Height of first interaction
                    tel[*]: map of tel_id to MCCameraEventContainer
```

```
data.mc.tel[1]

ctapipe.io.containers.MCCameraEventContainer:
        photo_electron_image: reference image in pure photoelectrons, with no
                              noise
    reference_pulse_shape: reference pulse shape for each channel
               time_slice: width of time slice [ns]
                  dc_to_pe: DC/PE calibration arrays from MC file
                  pedestal: pedestal calibration arrays from MC file
     azimuth_raw: Raw azimuth angle [radians from N->E] for the
                  telescope
  altitude_raw: Raw altitude angle [radians] for the telescope
   azimuth_cor: the tracking Azimuth corrected for pointing
                errors for the telescope
  altitude_cor: the tracking Altitude corrected for pointing
                errors for the telescope
```



Data access


experts write these
SimTelArray File
(hessio format)
customized event source
(hessio_event_source)
CHEC camera testbench file
customized event source
we agree on this
Dragon testbench file
customized event source
RawDataContainer
Algorithms
...
CTA Raw Data File
(MC or real data)
customized event source
don’t care where the data came from
Fake Events Generator
(reco.mock)
customized event source

io: Monte-Carlo Input

Working with data is supposed to be simple:

```
from ctapipe.io.hessio import hessio_event_source

source = hessio_event_source("gammas.simtel.gz")

for event in source:
    print(event.trig.tels_with_trigger)
```

complex I/O hidden in python generator

set of hierarchical containers for various data items

- attempt to keep the framework lightweight for algorithm designers (***lesson learned***), while supporting advanced processing techniques

Data Output

What do we need to write?

- **DL1 data**: complex (per-camera calibrated images with compression) + tabular data per camera and per shower
- **DL2 data**: easy: event list, just table of reconstructed parameters (for multiple algorithms) + classification features
- **DL3 data**: interface with **Science Tools** (GammaPy & CTools, etc) (reduced event list data)
- **IRFs:** interface with **Science Tools…** needs study

Implementation : Not yet well developed

- Generally, event-wise (at least at first few stages
- container classes can be pickled for short-term storage or for sending over network (not sufficient for storage)
- have a preliminary FITS table writer, not yet fully working or fast
- HDF5 would be better…

Coordinates

source: D. Parsons

Extension of astropy.coordinates **for our local frames:**

- CameraFrame
- TelescopeFrame
- NominalFrame
- GroundFrame
- TiltedGroundFrame

Automatically connected to Sky frames like Alt/Az, RA/Dec, Galactic

In [2]:
```
pix = [1,2,0] #z positions set to zero (as in most cameras)
```

Then we pass this array to CameraFrame object

In [3]:
```
camera_coord = CameraFrame(pix*u.m) # has to be in distance units
```

We then can directly convert this to the nominal telescope system. Need to know telescope focal length and camera rotation (if any).

In [4]:
```
telescope_coord = camera_coord.transform_to(TelescopeFrame(focal_length = 15*u.m,rotation=0*u.deg,
                                                           pointing_direction = [70*u.deg,180*u.deg]))
# pointing direction should maybe be changed to use astropy style AltAz object

print("Telescope Coordinate",telescope_coord)
```

```
Telescope Coordinate <TelescopeFrame Coordinate (focal_length=15.0 m, rotation=0.0 deg, pointing_direction=[<Quantity
70.0 deg>, <Quantity 180.0 deg>]): (x, y, z) in rad
    (0.06666667, 0.13333333, 0.0)>
```

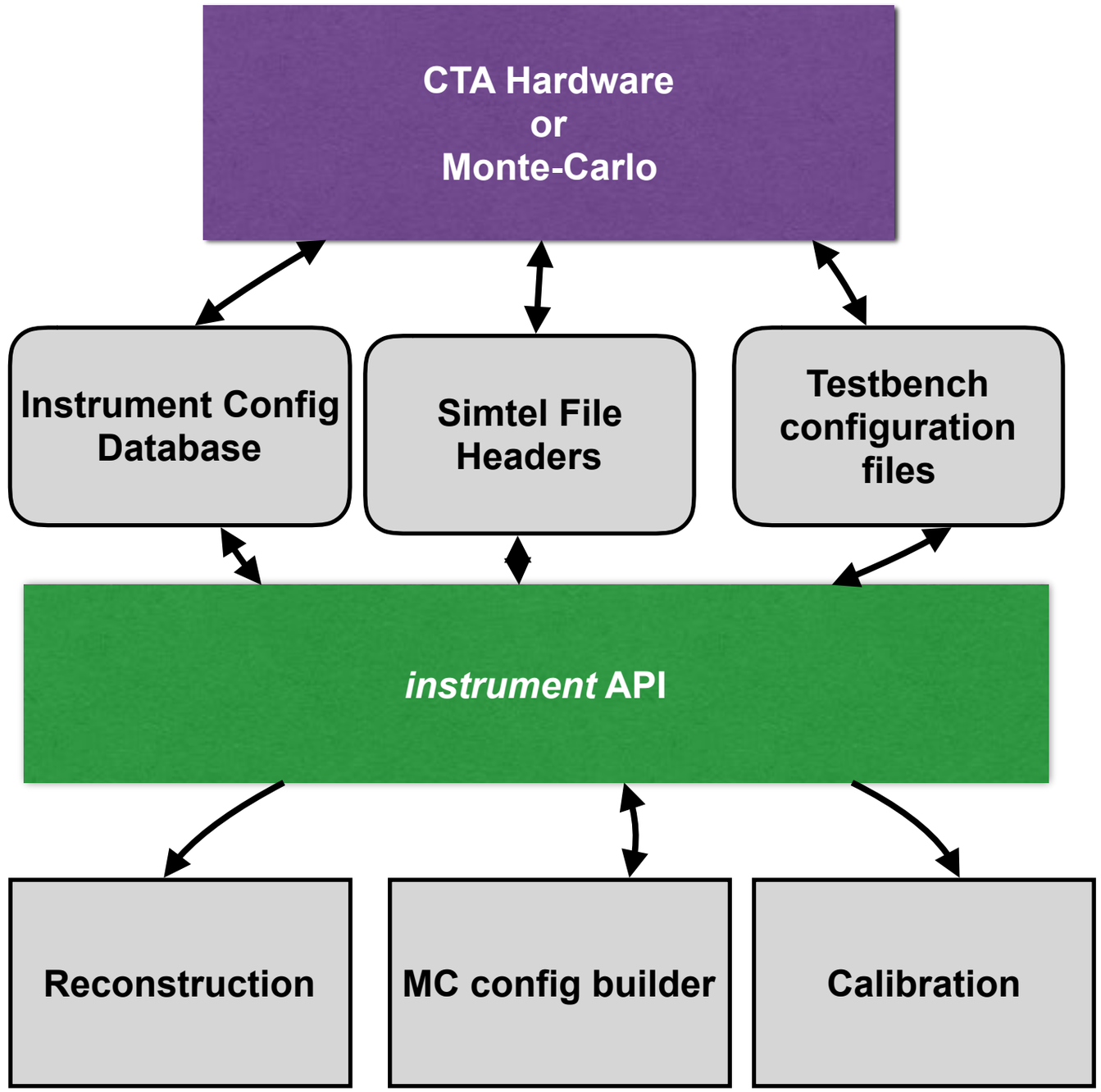
Instrument

Concept:

- provide single point of access to all instrumental configuration information
- User should not care where it comes from, just give a date, or ObsID, etc and get the correct info.

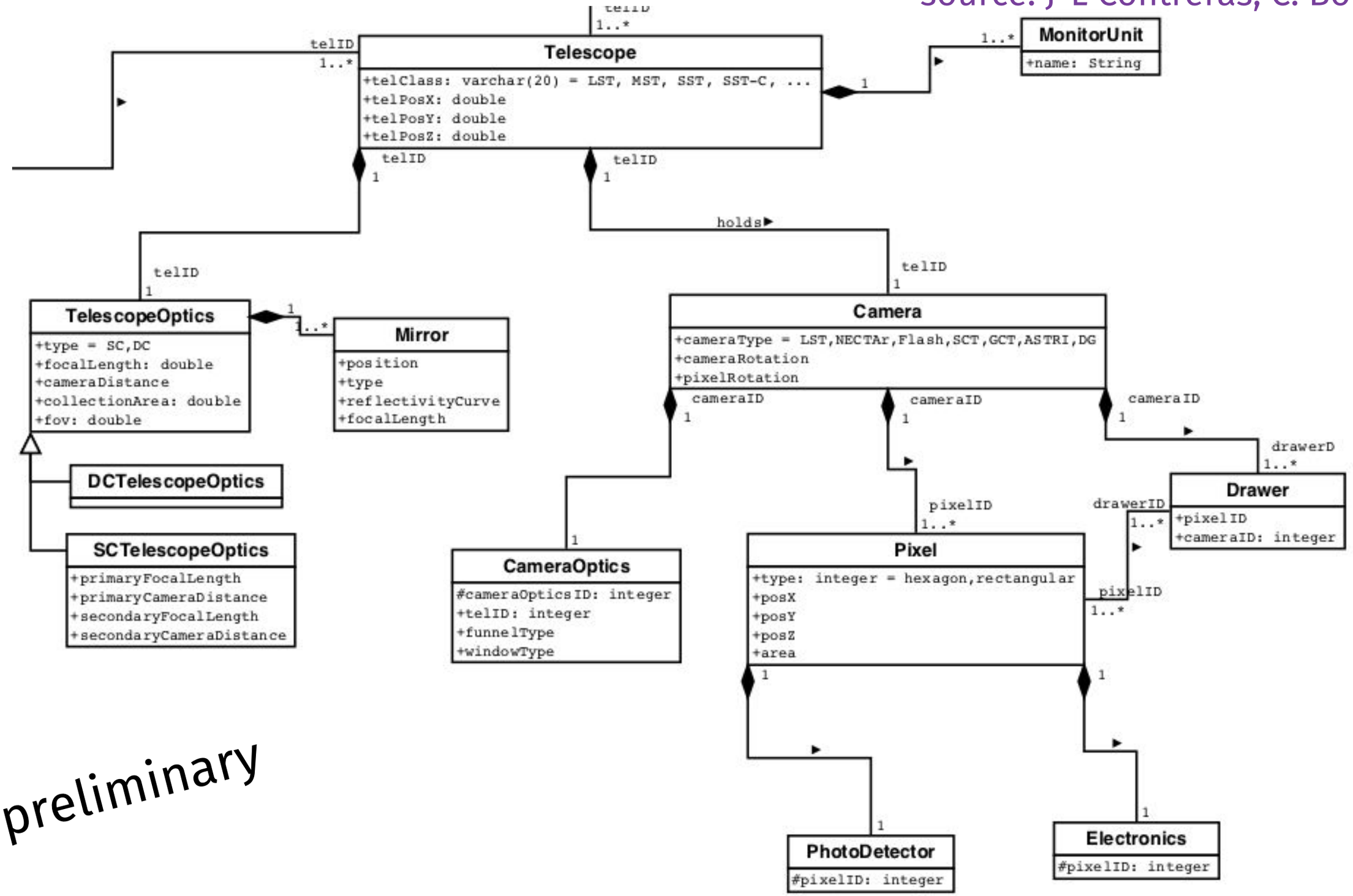
Status:

- Currently too tied to MC files
- Some prototyping done, but not ready for use….



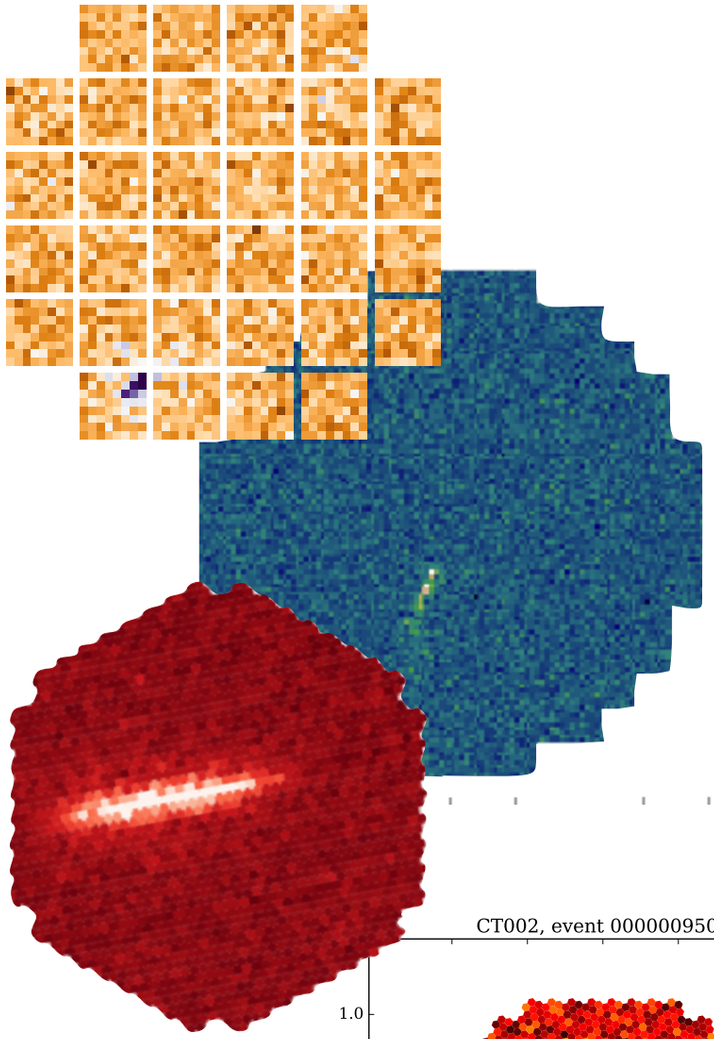
Instrument Description Model

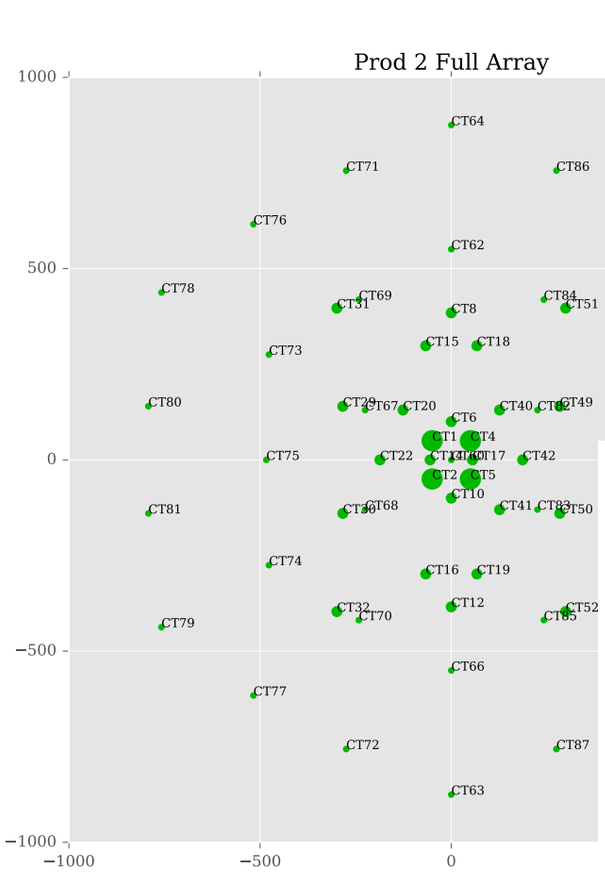
source: J-L Contreras, C. Boisson

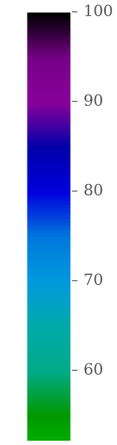


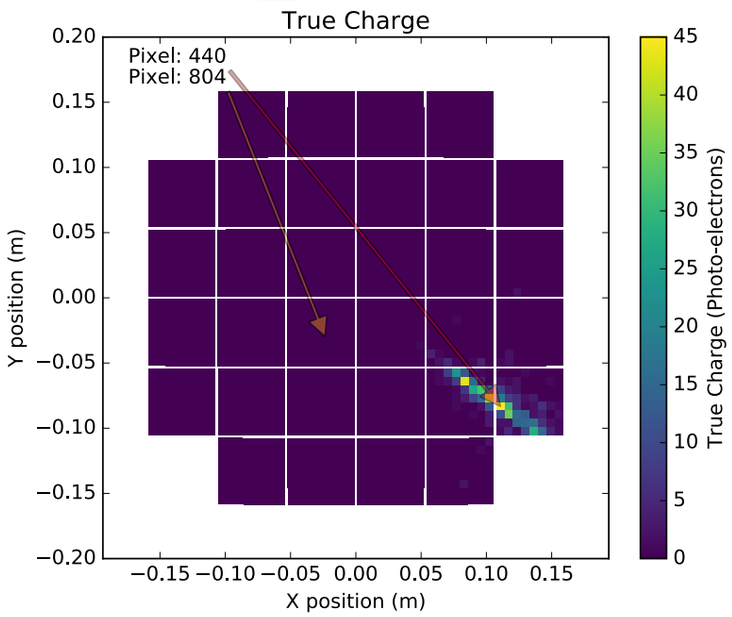
preliminary

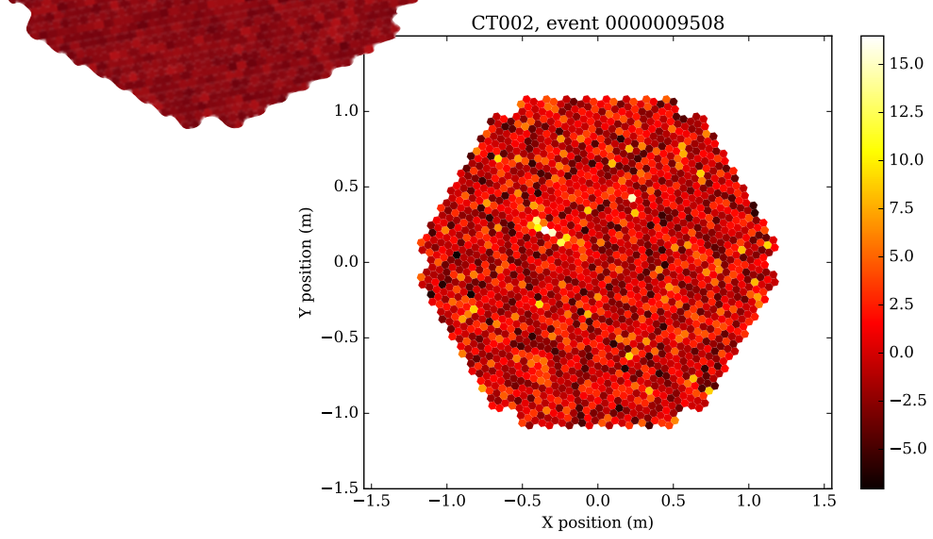
Visualization

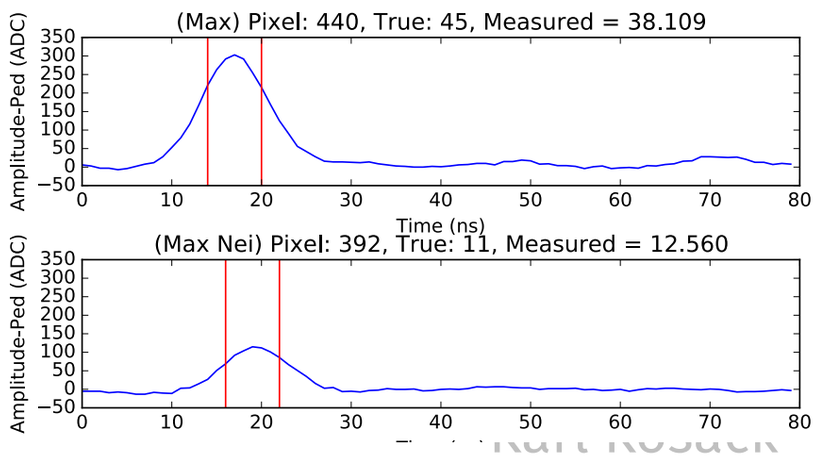








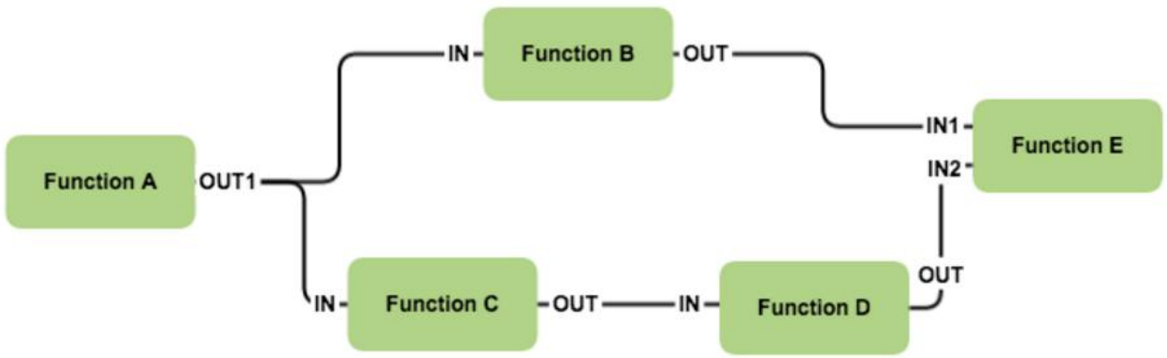




Flow-based Framework

source: J. Jacquemier

J. Jacquemier (LAPP)



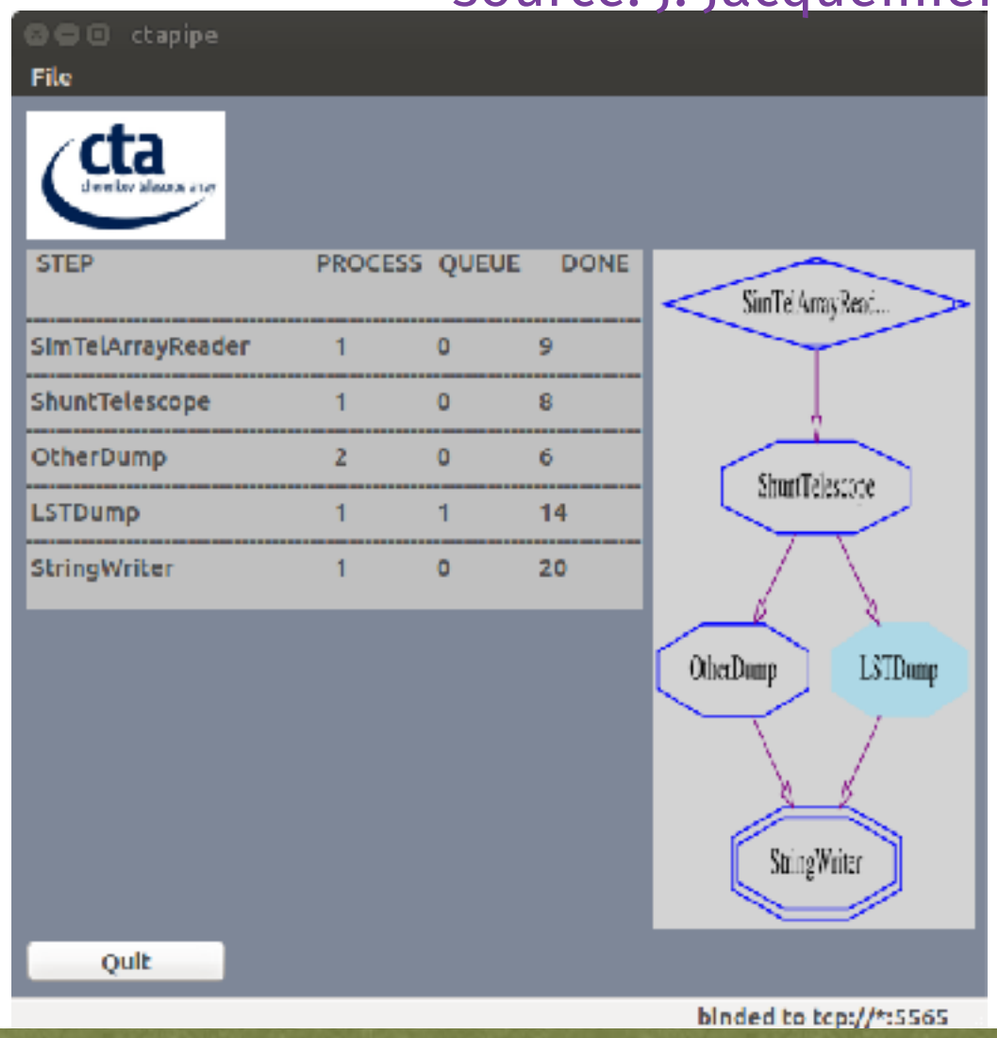
- Flow Based Programming (FBP) a "data factory" metaphor
- Networks of "black box" processes, connections externally to the processes.

- Session persistence
- Multiprocessing
- Distributed computing (Cluster , Grid)
- Memoization
- Report generation (text, HTML, ...)
  - Automatic time–space complexity monitoring
- Graphical interface for creating and exploring dia
- Robustness to exceptions
- Support for time–outs
- Real–time logging / monitoring

ctapipe.flow

- in-house (no large external dependencies)
- uses ZeroMQ as transport mechanism
- auto parallelization
- GUI for monitoring
- easy to wrap and run existing algorithms
- independent of algorithm design
  - devs don't need to worry about it
  - their algorithms can be wrapped by experts

source: J. Jacquemier



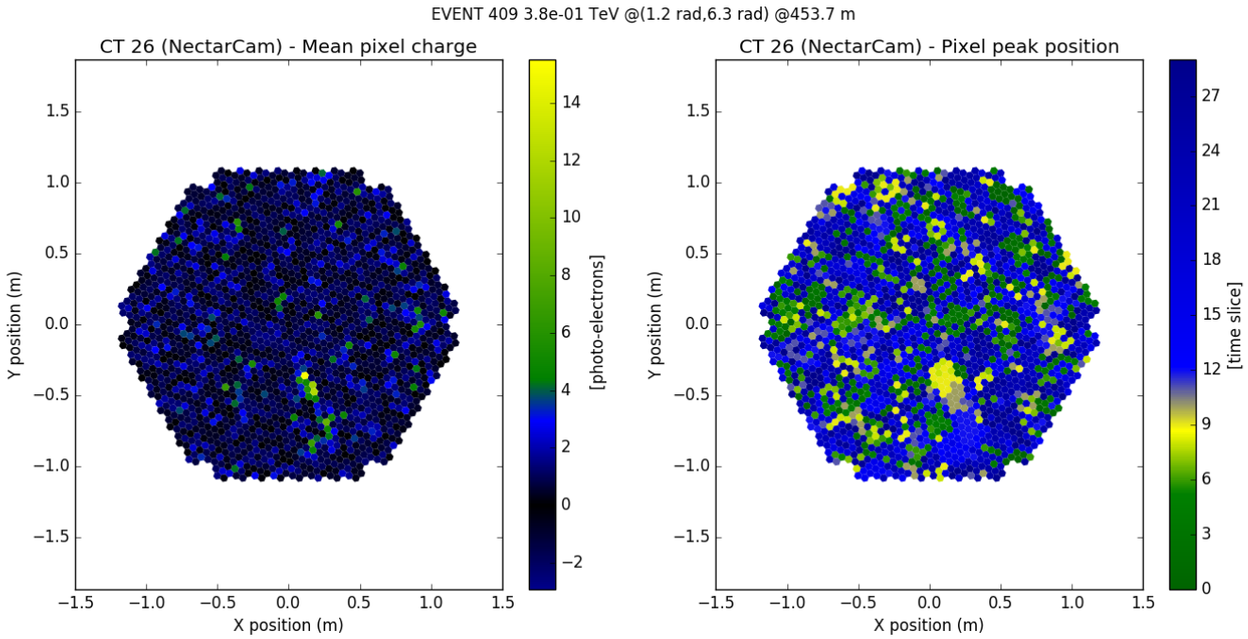
Prototype, also looking into similar external technologies (Apache Storm, Apache Spark, Dask, ...)

```
>>> def run(self):
>>>     for input_file in os.listdir(self.source_dir):
>>>         yield (self.source_dir + "/" + input_file)
```

https://cta-observatory.github.io/ctapipe/flow

Camera Calibration

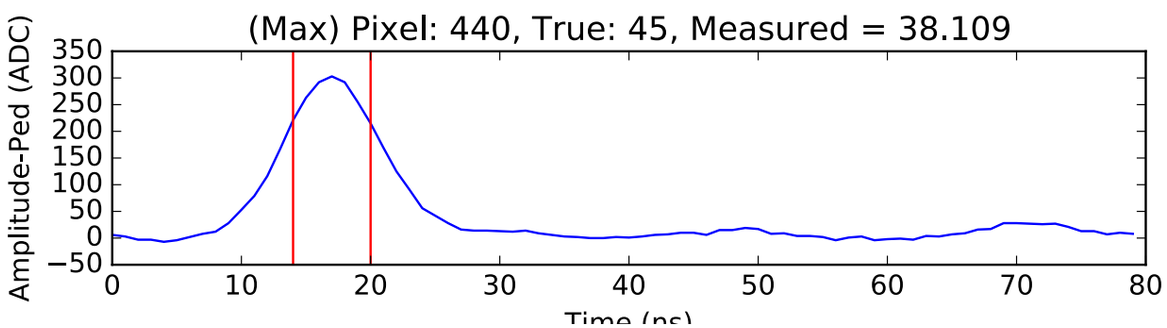
source: J. Watson



J.J. Watson

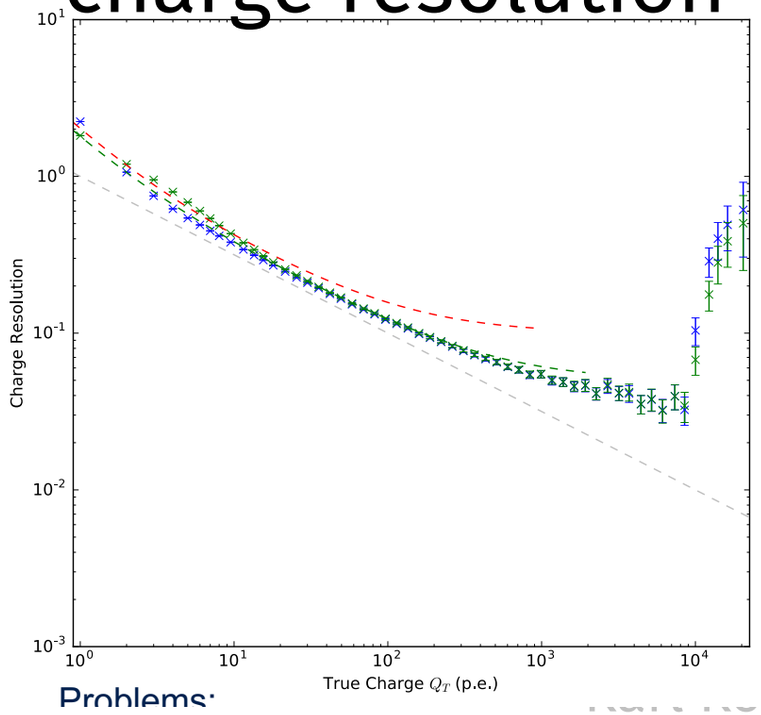
2nd CTA Pipeline Developer's Workshop
11th October 2016

7



- Trace integration (reconstruction)
- application of FF, NSB pedestals, (optional DC/PE conversion)

validation example: charge resolution



Problems:

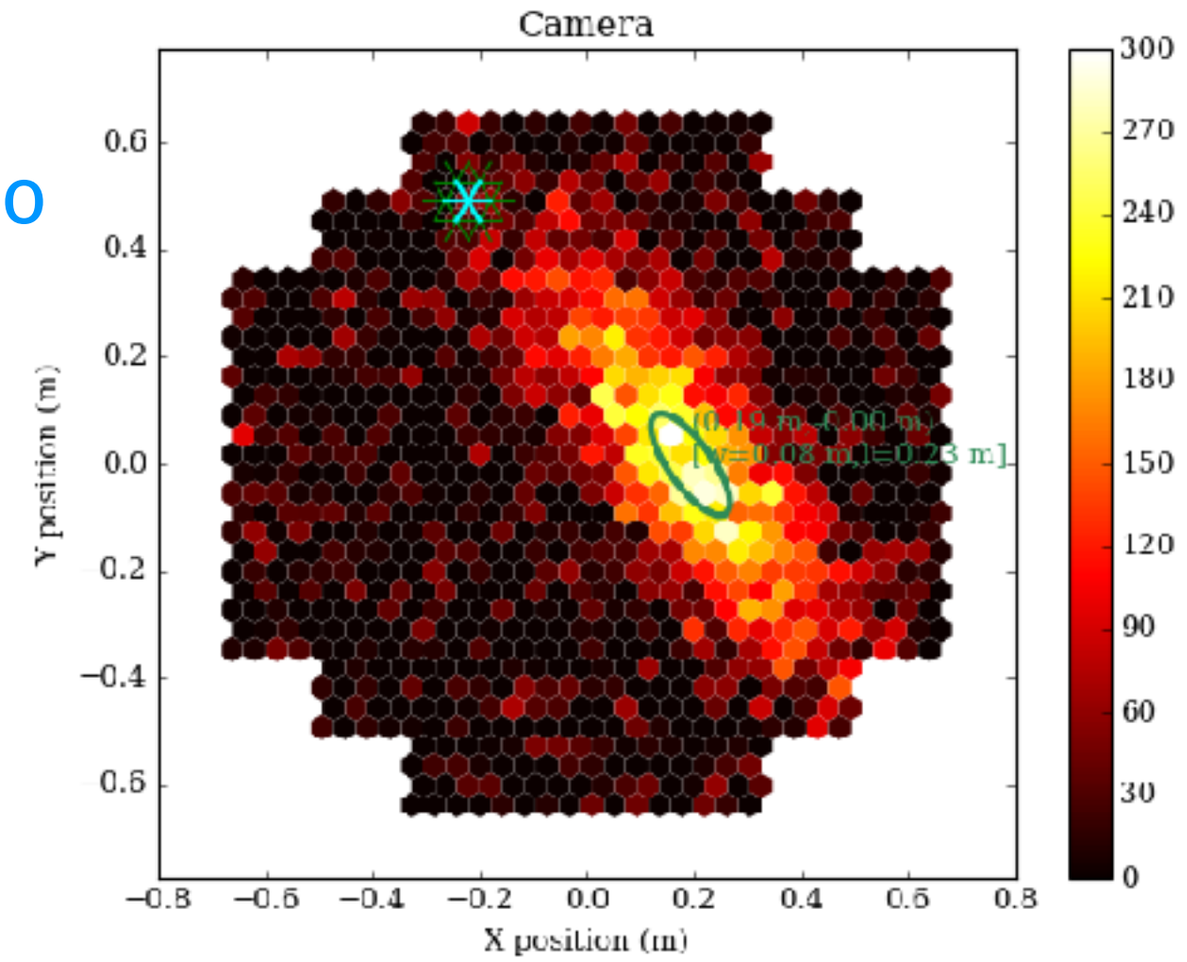
Hillas Image analysis

source: W. Bhattacharyya

Simple algorithms, but some options on how to do it.

- 4 variants implemented so far,
- Verification underway

Input to many of the reconstruction algorithms, so want to do it right



Muon Image Analysis

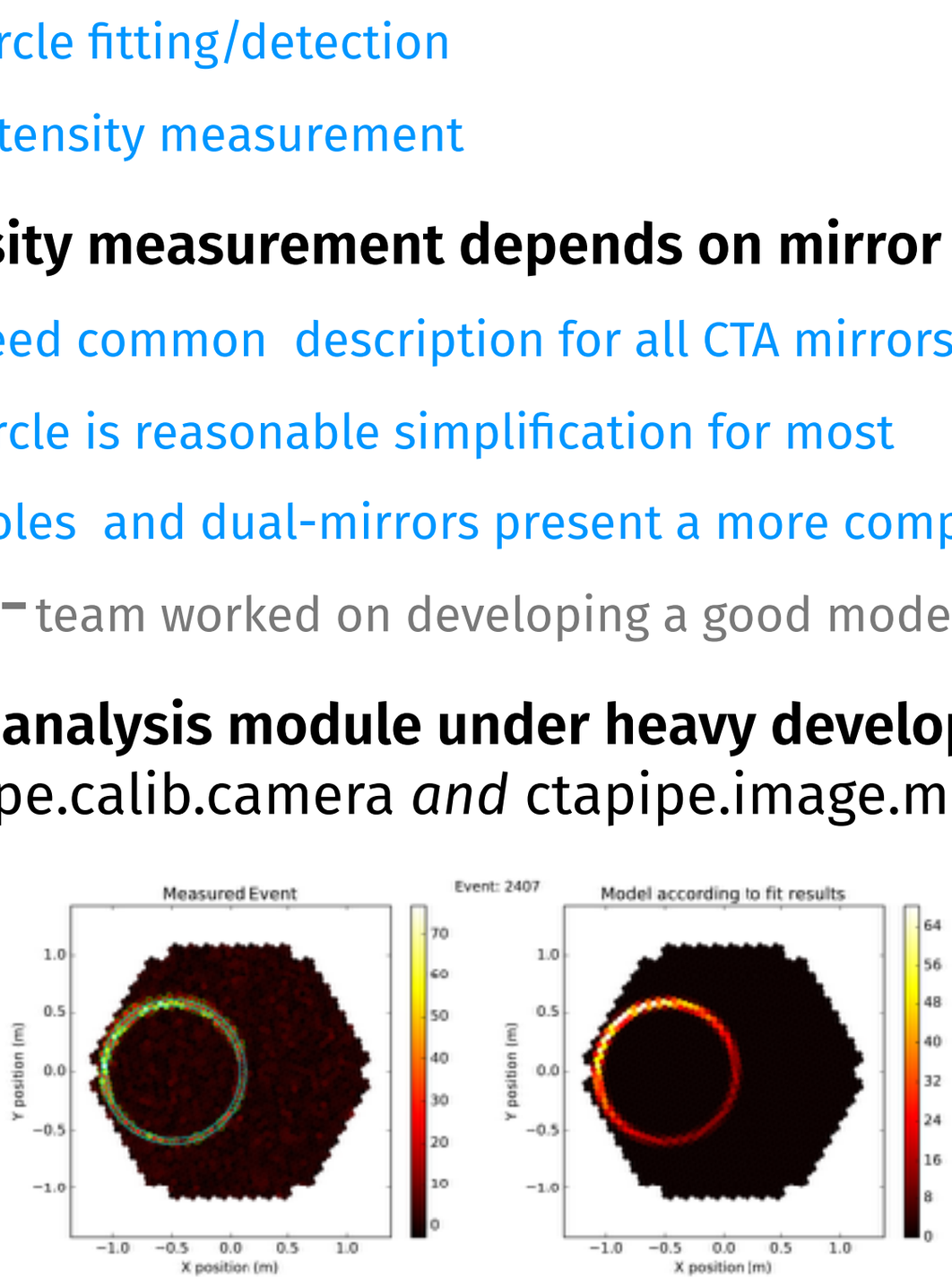
Two Steps, multiple algos for each:

- Circle fitting/detection
- Intensity measurement

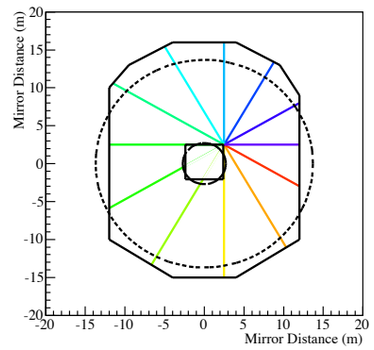
Intensity measurement depends on mirror geometry

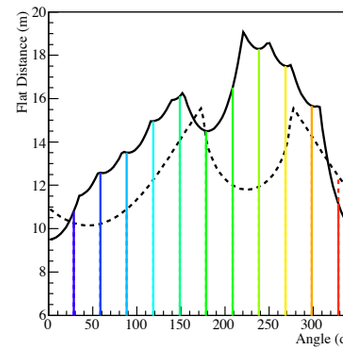
- need common description for all CTA mirrors
- circle is reasonable simplification for most
- Holes and dual-mirrors present a more complex problem
  - team worked on developing a good model for this

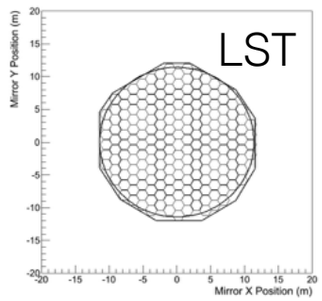
Muon analysis module under heavy development
(ctapipe.calib.camera *and* ctapipe.image.muon)

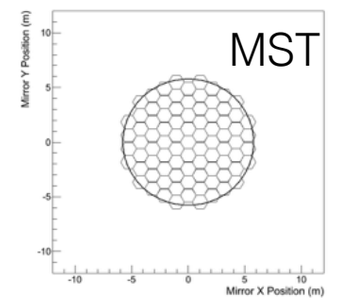


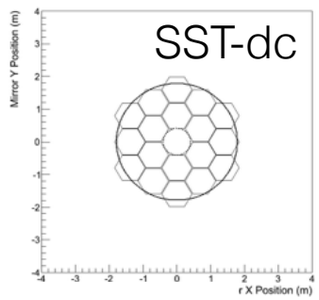
source: A. Mitchell

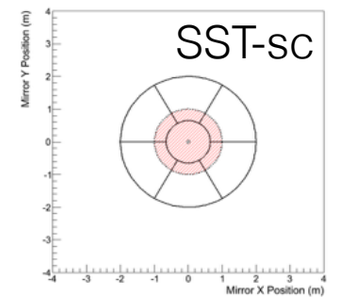


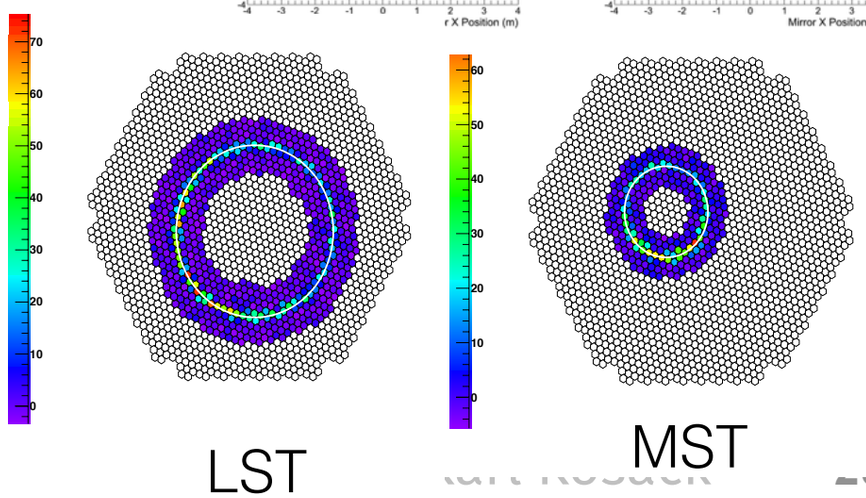




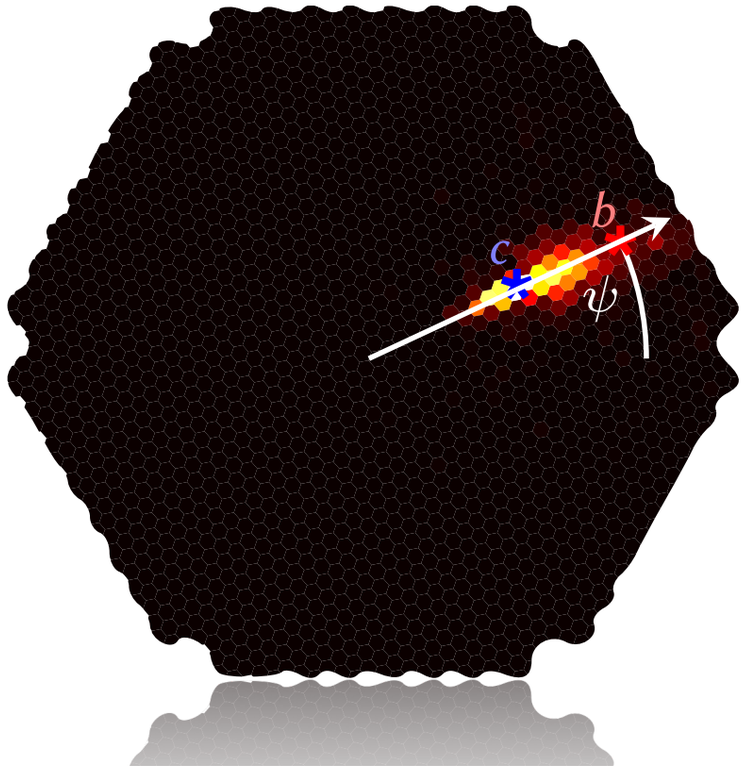








Reconstruction



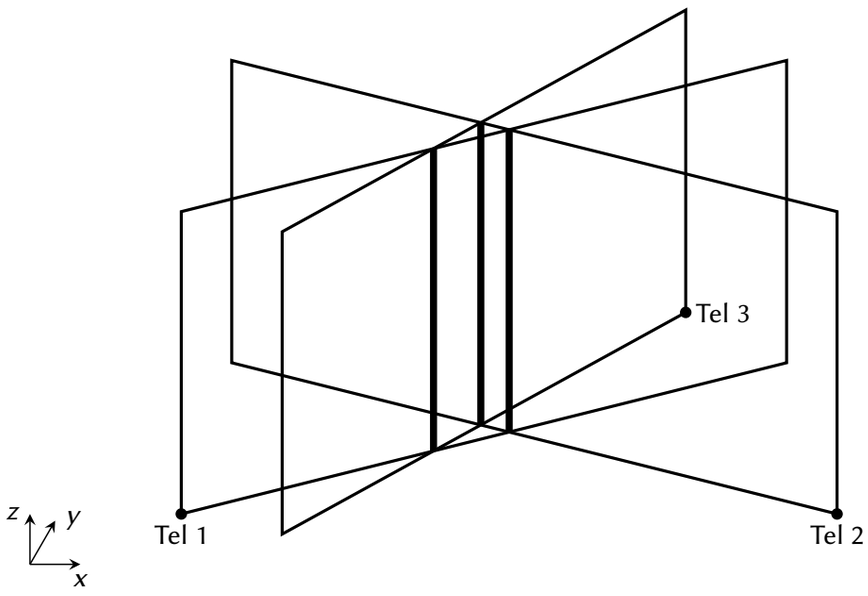
source: T. Michael

Hillas reconstruction

- this cross section is perpendicular to the normal direction of both intersecting planes ($\vec{n} = \vec{b} \times \vec{c}$)
- $\rightarrow$ shower direction is $\vec{n}_1 \times \vec{n}_2$
- add up all cross products for weighted mean direction:

$$\vec{d}_\gamma = \sum_{i=1}^{N_{\text{Tels}}} \sum_{j=i+1}^{N_{\text{Tels}}} \vec{n}_i \times \vec{n}_j$$

- note: $|\vec{n}_i \times \vec{n}_j| = |\vec{n}_i| \cdot |\vec{n}_j| \cdot \sin[\measuredangle(\vec{n}_i, \vec{n}_j)]$
- $\rightarrow$ automatically weights contributions according to the angle between intersecting planes



second step

- maximising the sum of the angular distances between the shower direction and all the normal vectors (actually $\sum_i \sin(\angle(\vec{n}_i, \vec{p}_{\text{sh}}))$
- again weighted with sine of the angles

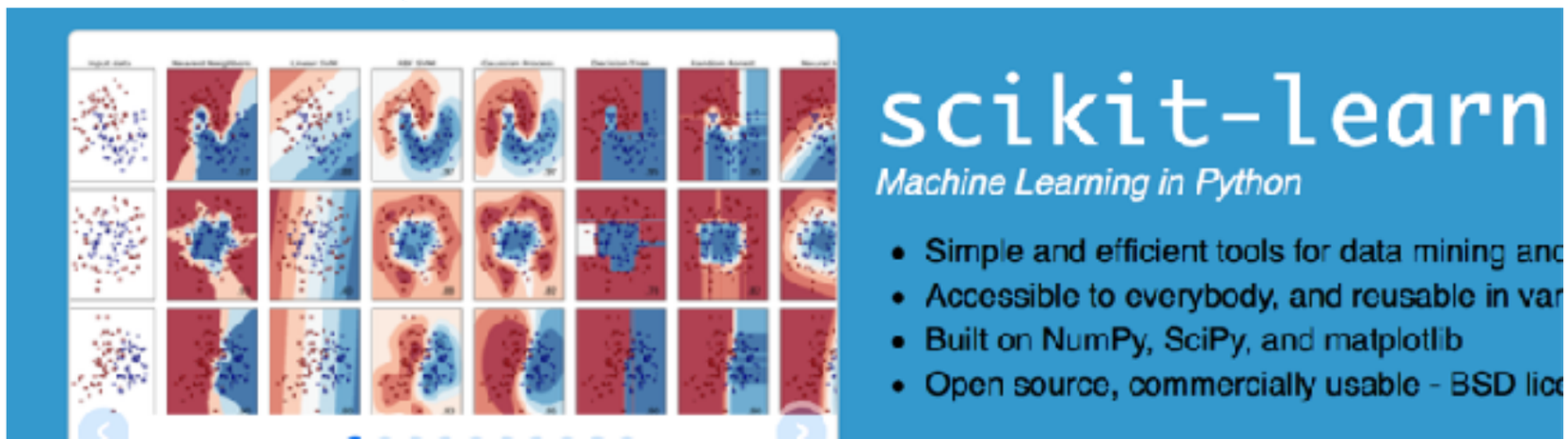
Event classification

Two goals:

- separate progenitor types (gamma, hadron, electron)
- divide gammas into classes/types [future work]

Implementation:

- very little work from us needed! All covered by scikit-learn!
  - wide variety of machine learning algorithms
  - BDTs, RandomForests, SVM, Perceptrons ,etc...
  - Huge user and develop base
- easy to use API: train() , predict()
- both classifiers and regressors for each type
- work for us: explore best methods



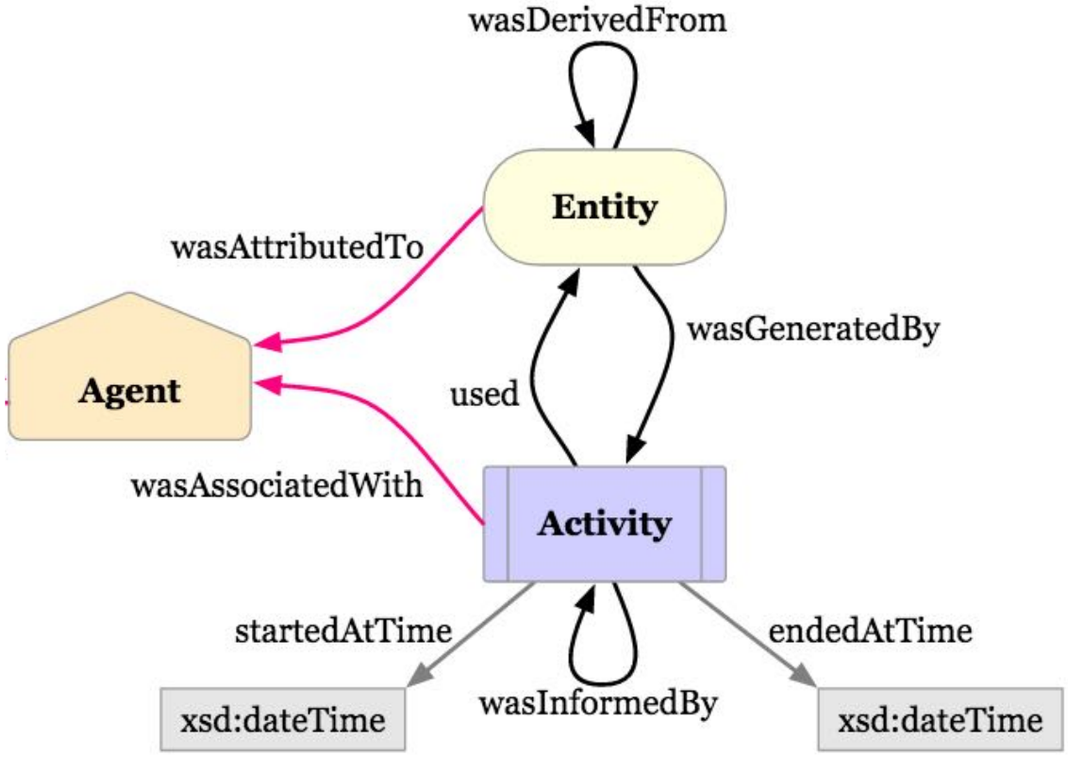
Provenance

source: M. Servillat

Provenance DM from W3C

http://www.w3.org/TR/prov-dm/, published 2013

- 3 core classes:
  - Activity
  - Entity
  - Agent
- core relations:
  - used
  - wasGeneratedBy
  - wasDerivedFrom
  - wasAttributedTo
  - wasAssociatedWith
- + many more classes and relations



Low-level python API provided by w3c (JSON or XML)

Want an *automatic* API in ctapipe

- prototype underway
- register all input and output files, runtime environment parameters, start/stop time, etc.

Instrument Description

source: J-L Contreras

Used by:

- **MC pipeline** (need full instrument description + obsconfig)
- **Main data processing** pipeline (need camera, mirror, array layout, etc)
- **ACTL** (need to configure observations)
- **PO and HW teams** (track movement/changes to hardware via serial numbers)

Goal 1: fill in and iterate data model:

- quite complete model already exists (UML)
- working on a YAML realization of it, with real data example filled in

Goal 2: provide API to work with the data model

- user should not care where it is read from (YAML file, database, FITS, MC header, etc)
- Possible simplification: always extract model from DB or wherever to YAML, and then ctapipe only needs to read yaml (though converter code is still necessary)

Issues:

Speed:

- nothing is optimized so far (focus on clarity first)
  - ➤ will need some numba, cython, or c code in critical algorithms like image cleaning, etc.

Data Structures and I/O:

- format of Containers - not quite sure how to store things correctly (evolving)
- data output: no good way to store output yet
  - ➤ looking into HDF5 and FITS table serialization
  - ➤ can use pickle now, but not efficient or appropriate for long-term storage
  - ➤ Haven't settled on DL1 format yet (so far quite camera-dependent)
  - ➤ no CTA DL0 format yet

Configuration: not so friendly yet

Instrument metadata: not so friendly to use

Still lots of API changes to make…