CTAPIPE:

experimental framework for CTA event data processing

github.com/cta-observatory/ctapipe

Goal

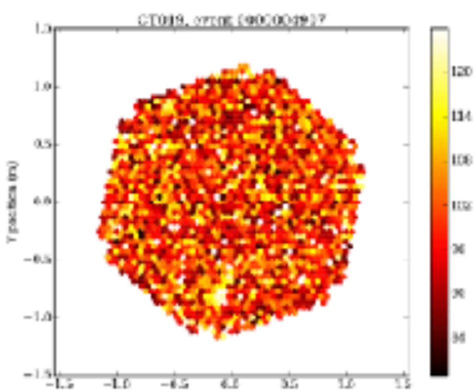
Process raw data :

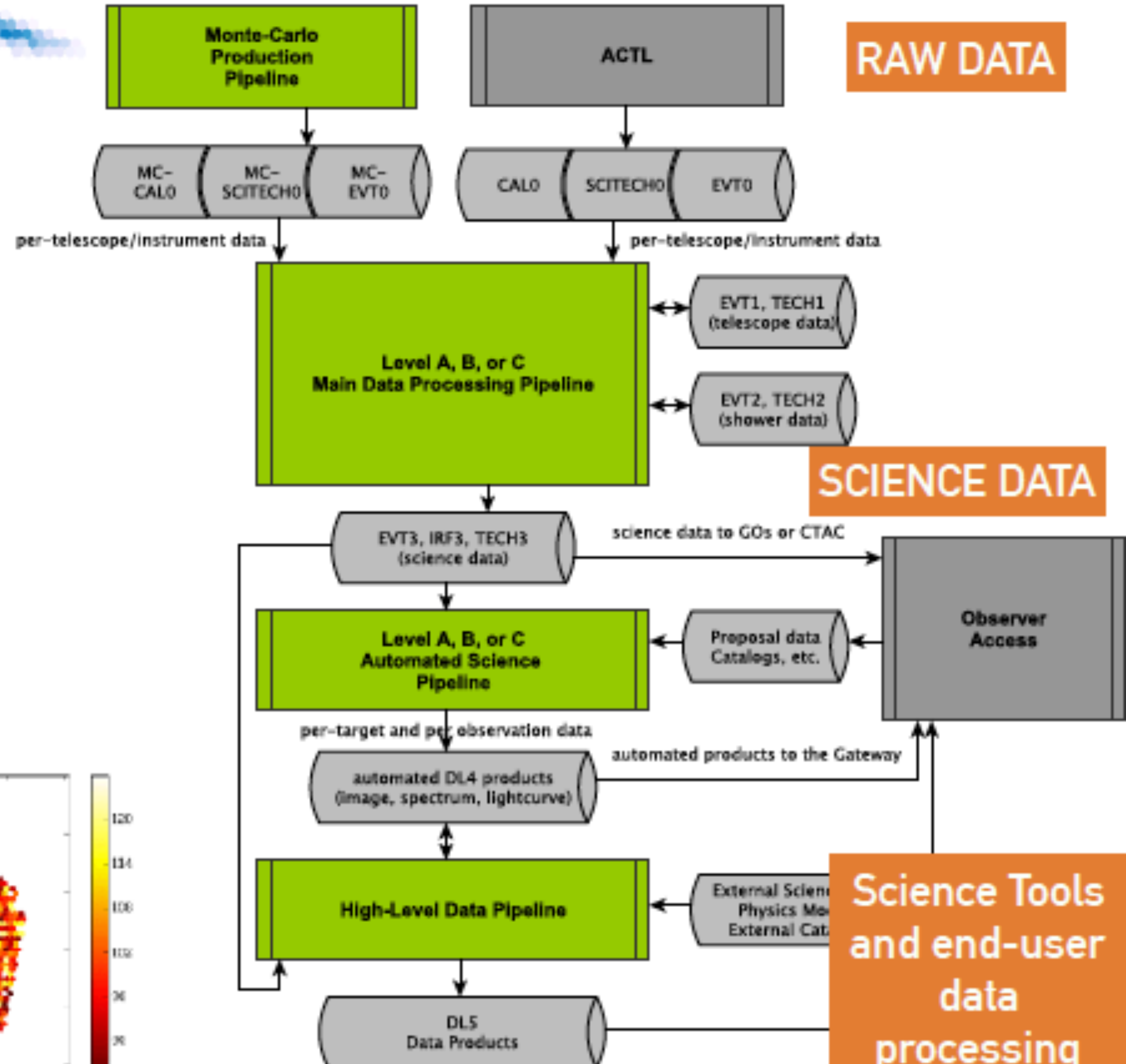
- images of air-showers produ
  by gamma-rays or cosmic rays,
  seen by an array of telescopes

- 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

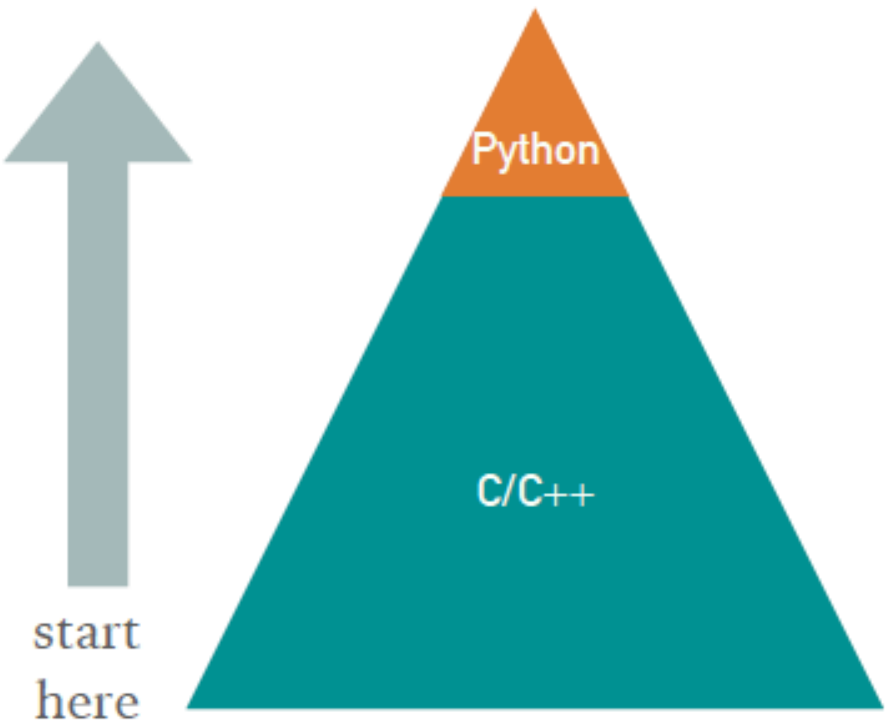




[CTA Technical Design Report]

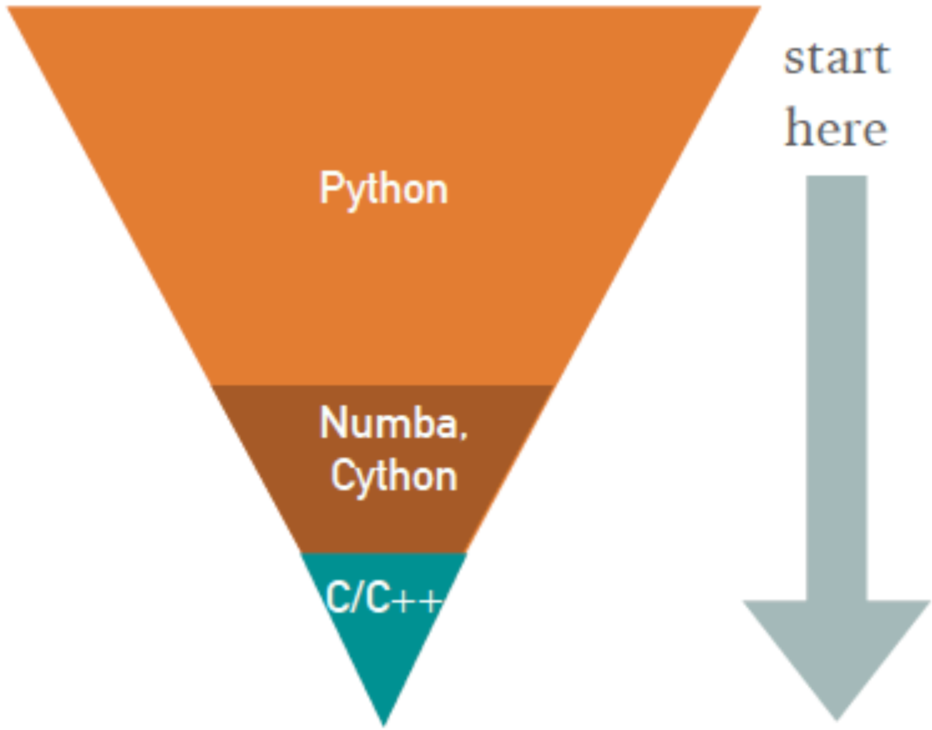
Building a Framework

Bottom-Up approach



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

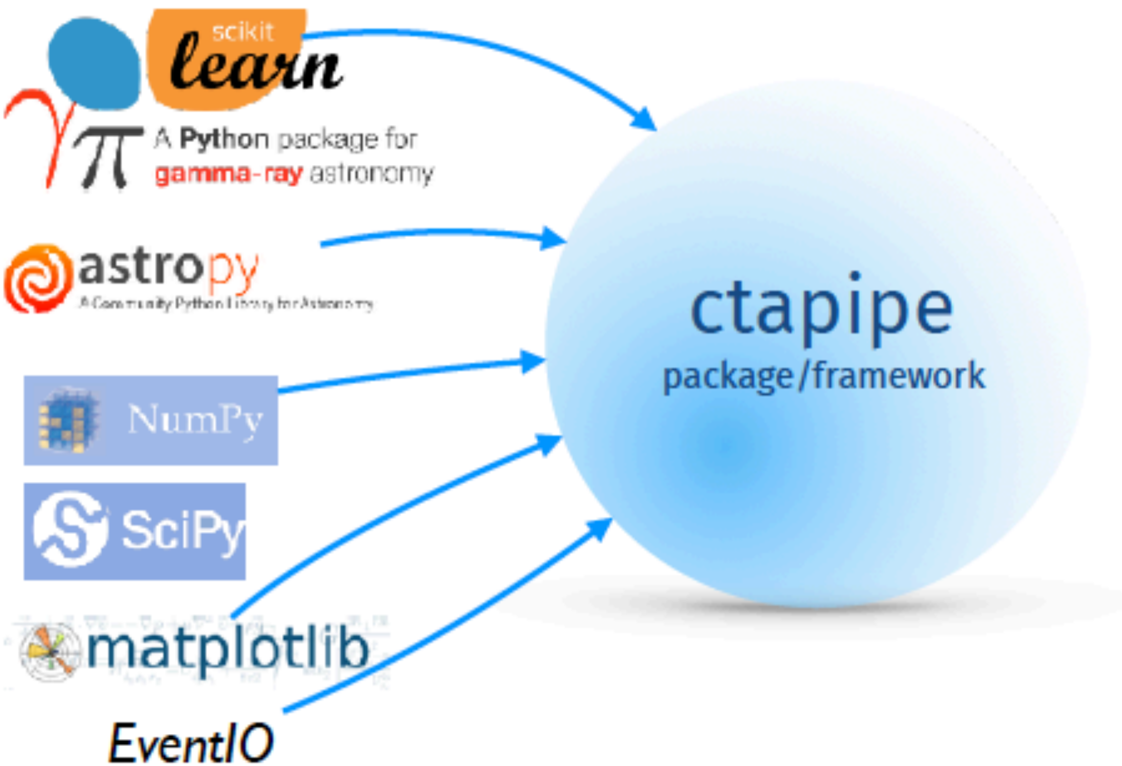
Top-Down approach



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

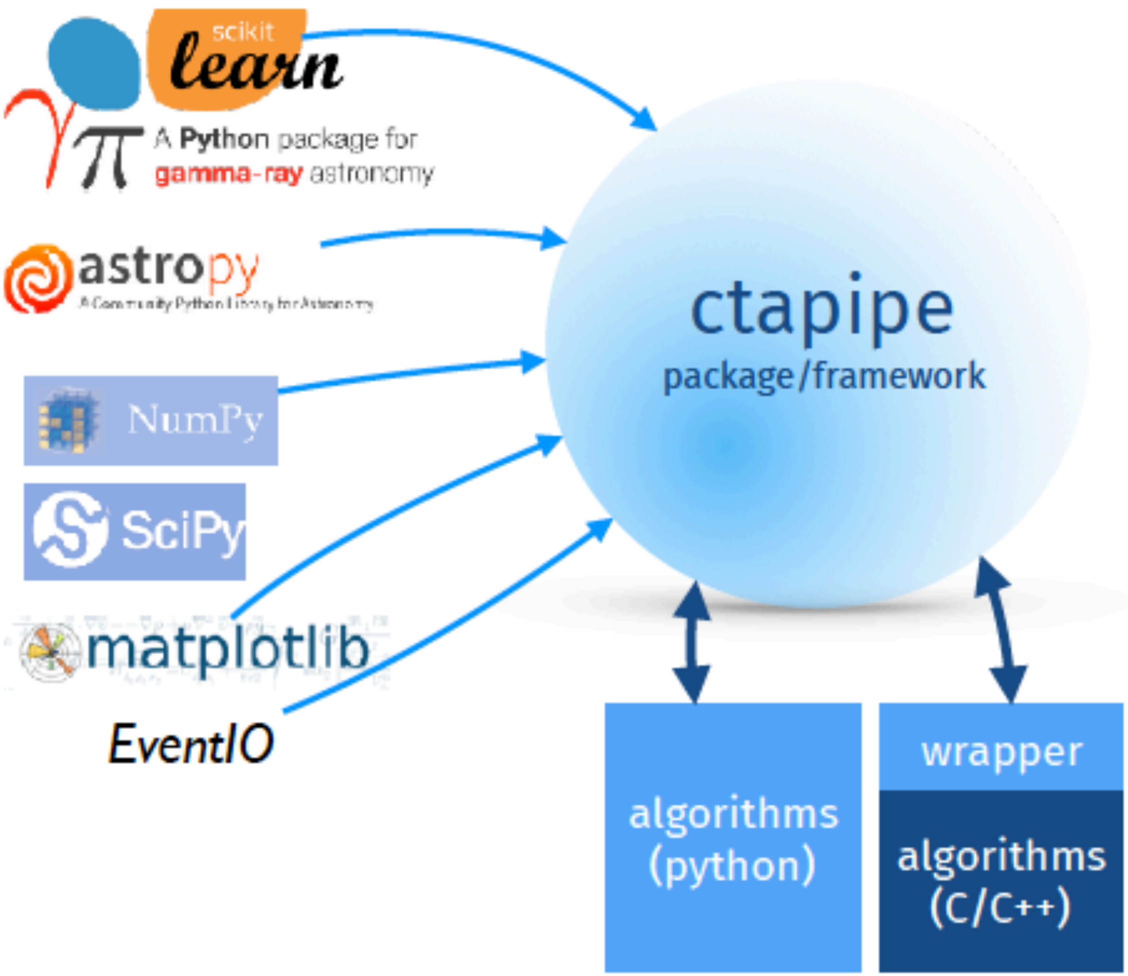
common “core” package

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



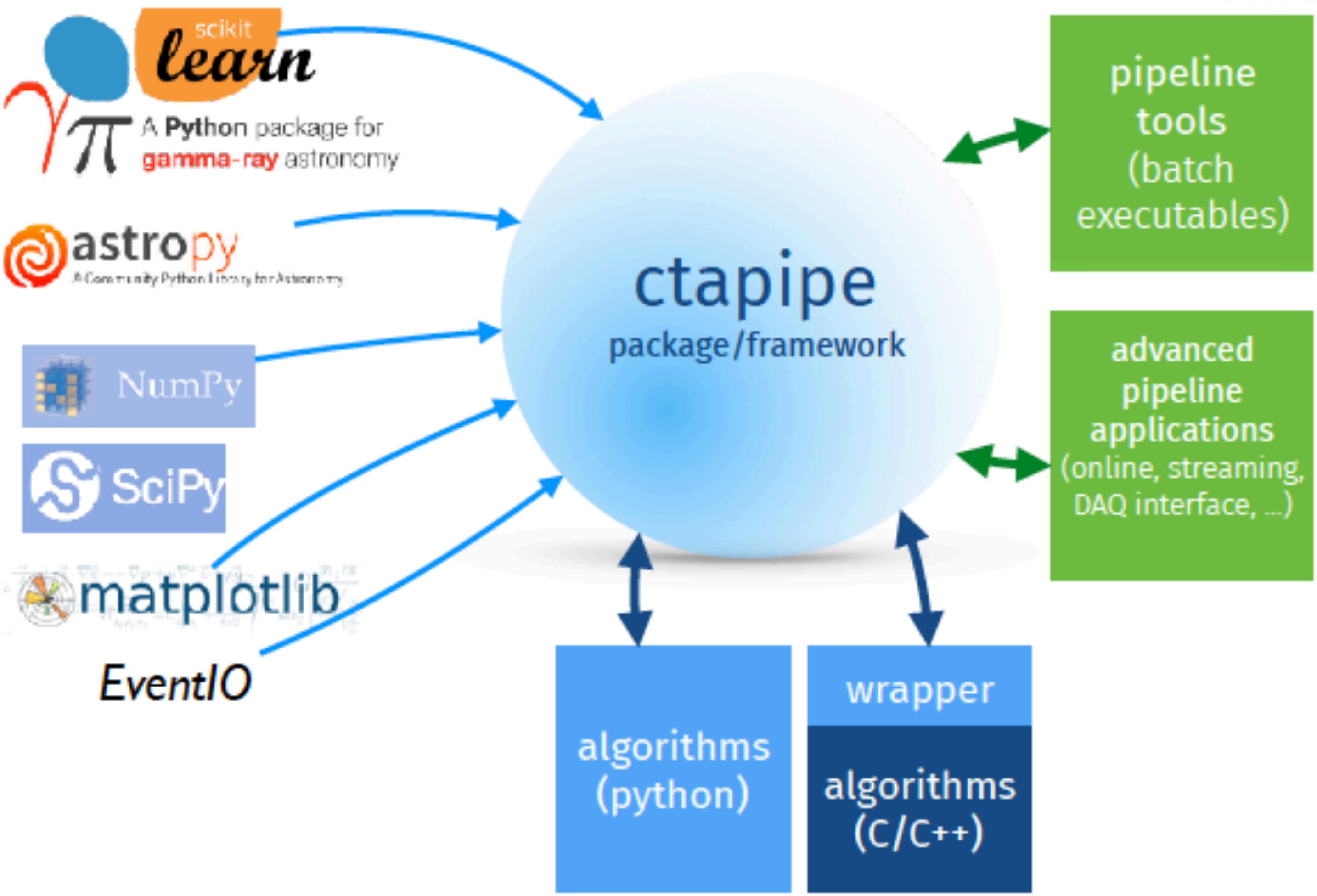
common “core” package

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



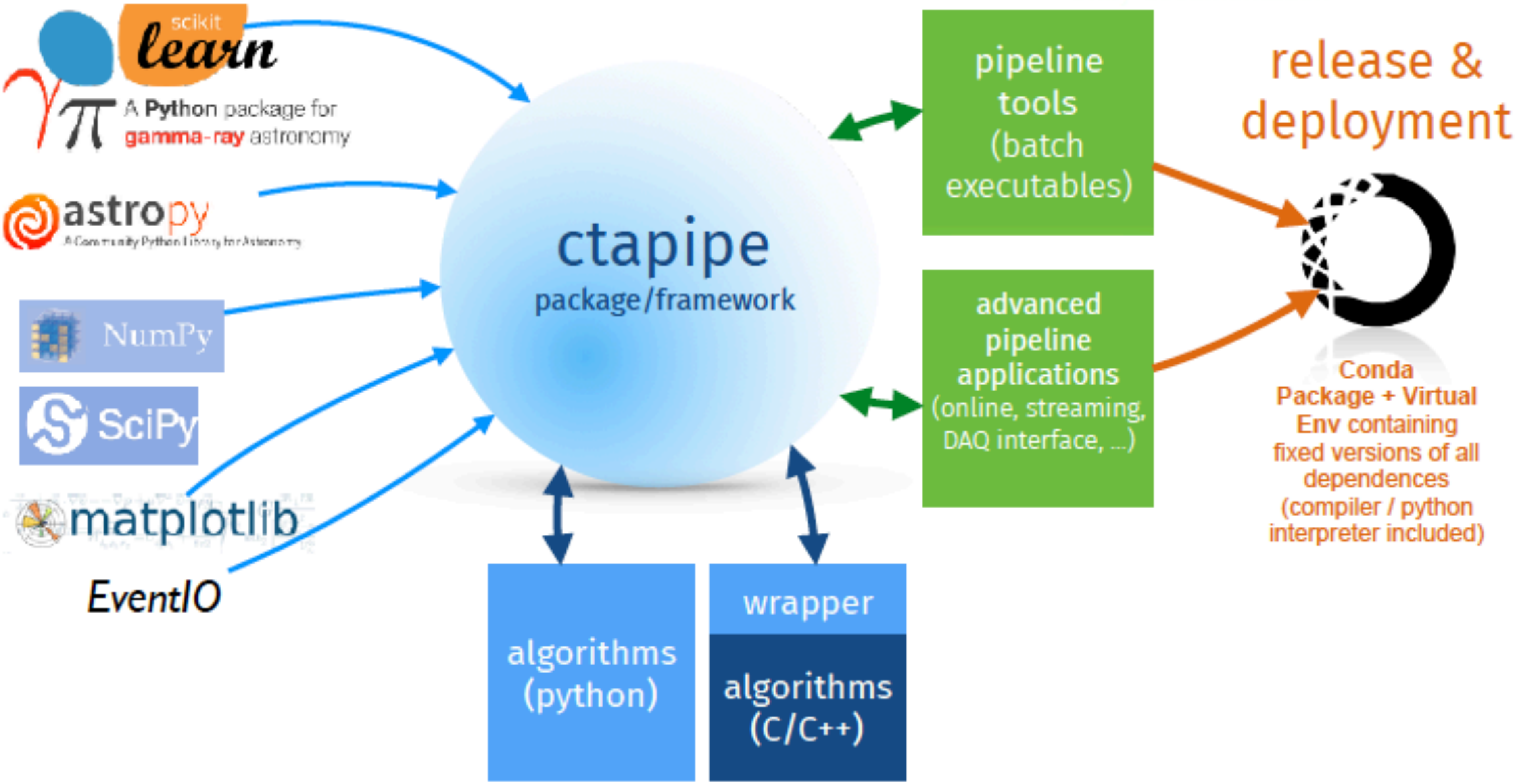
common “core” package

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

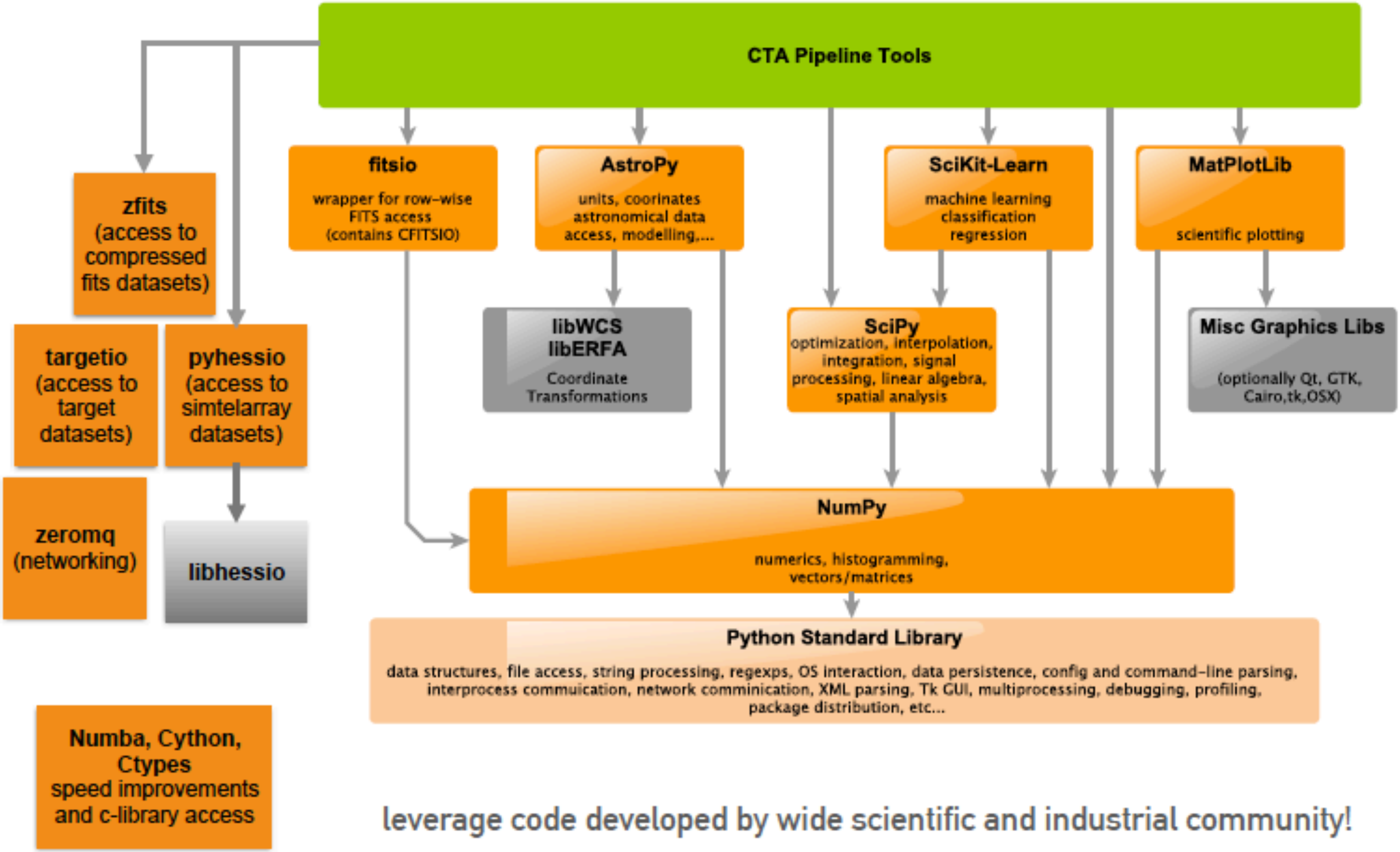


common “core” package

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!

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

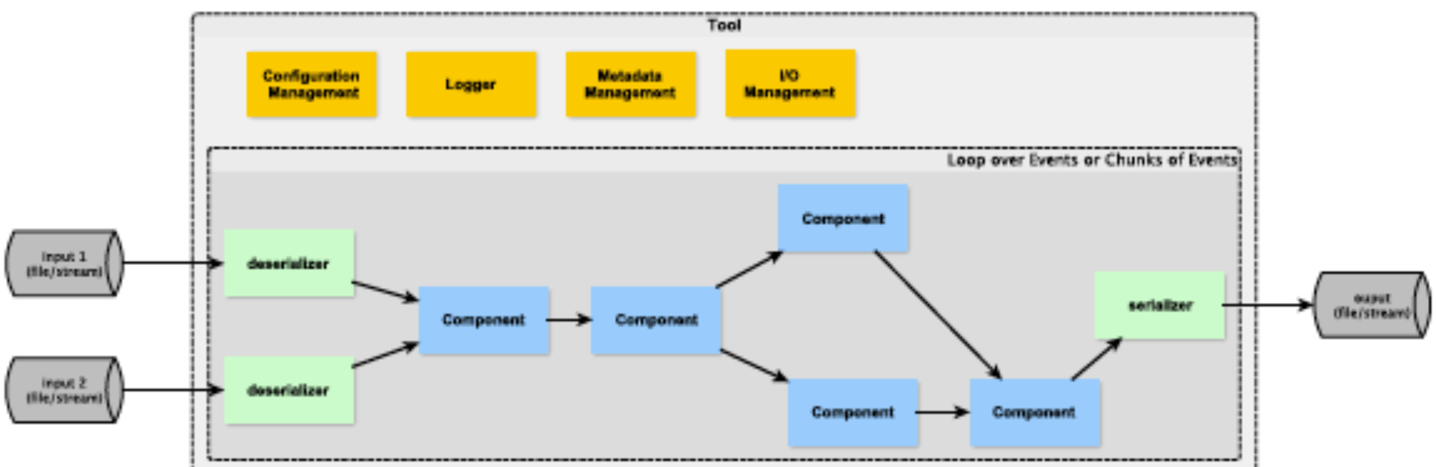
- command-line application base class
- currently based on **traitlets.config.Application** (may change)

```
data.mc
```

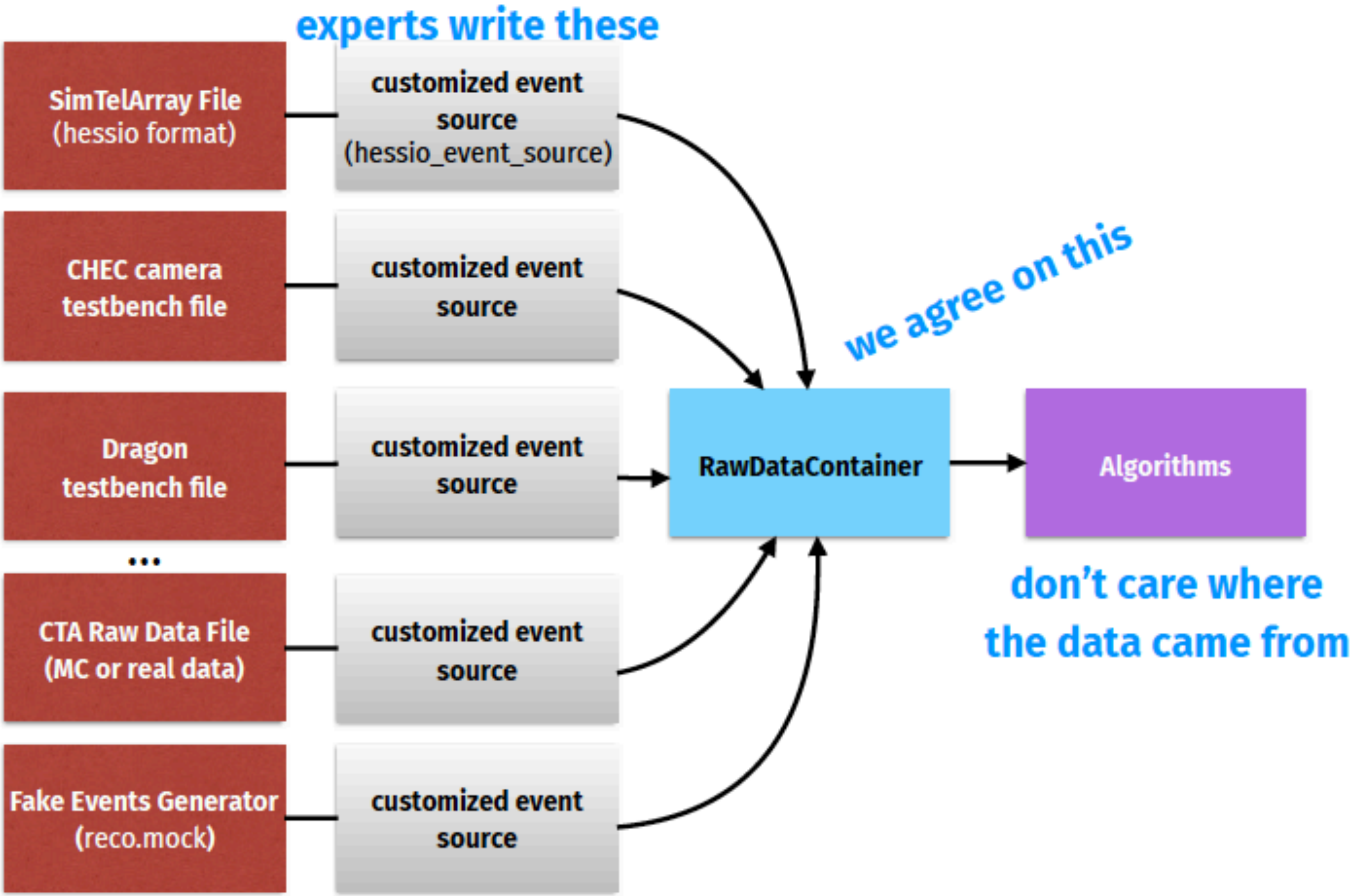
```
ctapipe.io.containers.MCEventContainer:
                         energy: Monte-Carlo Energy
                            alt: Monte-carlo altitude [deg]
                             az: Monte-Carlo azimuth [deg]
                         core_x: MC core position
                         core_y: MC 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
Dragon testbench file
customized event source
...
CTA Raw Data File (MC or real data)
customized event source
Fake Events Generator (reco.mock)
customized event source
we agree on this
RawDataContainer
Algorithms
don't care where the data came from

io: Monte-Carlo Input

Working with data is supposed to be simple:

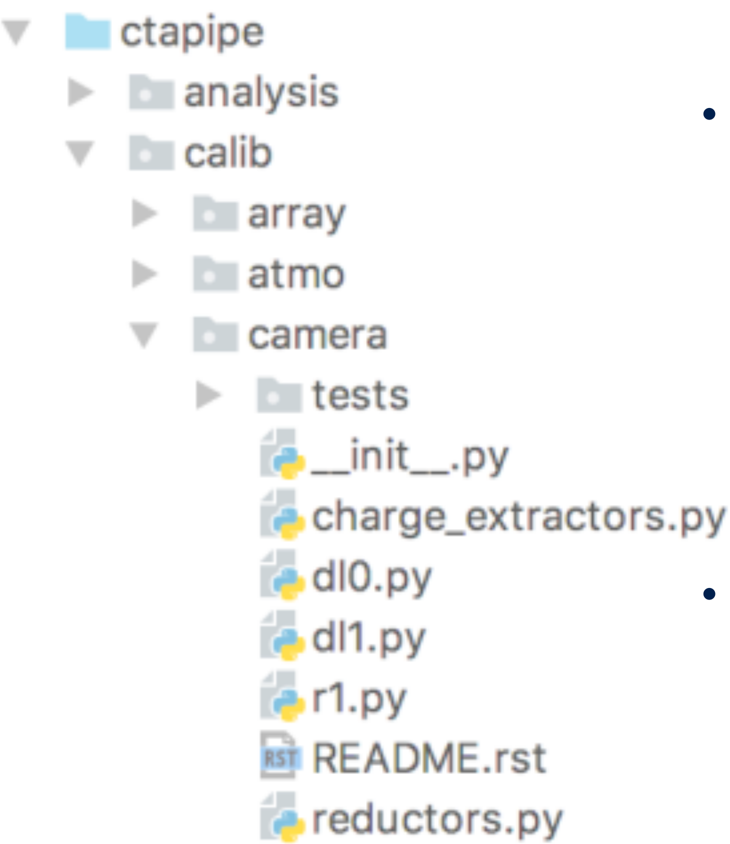
```python
from ctapipe.io.hessio import hessio_event_source

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

Camera Calibration

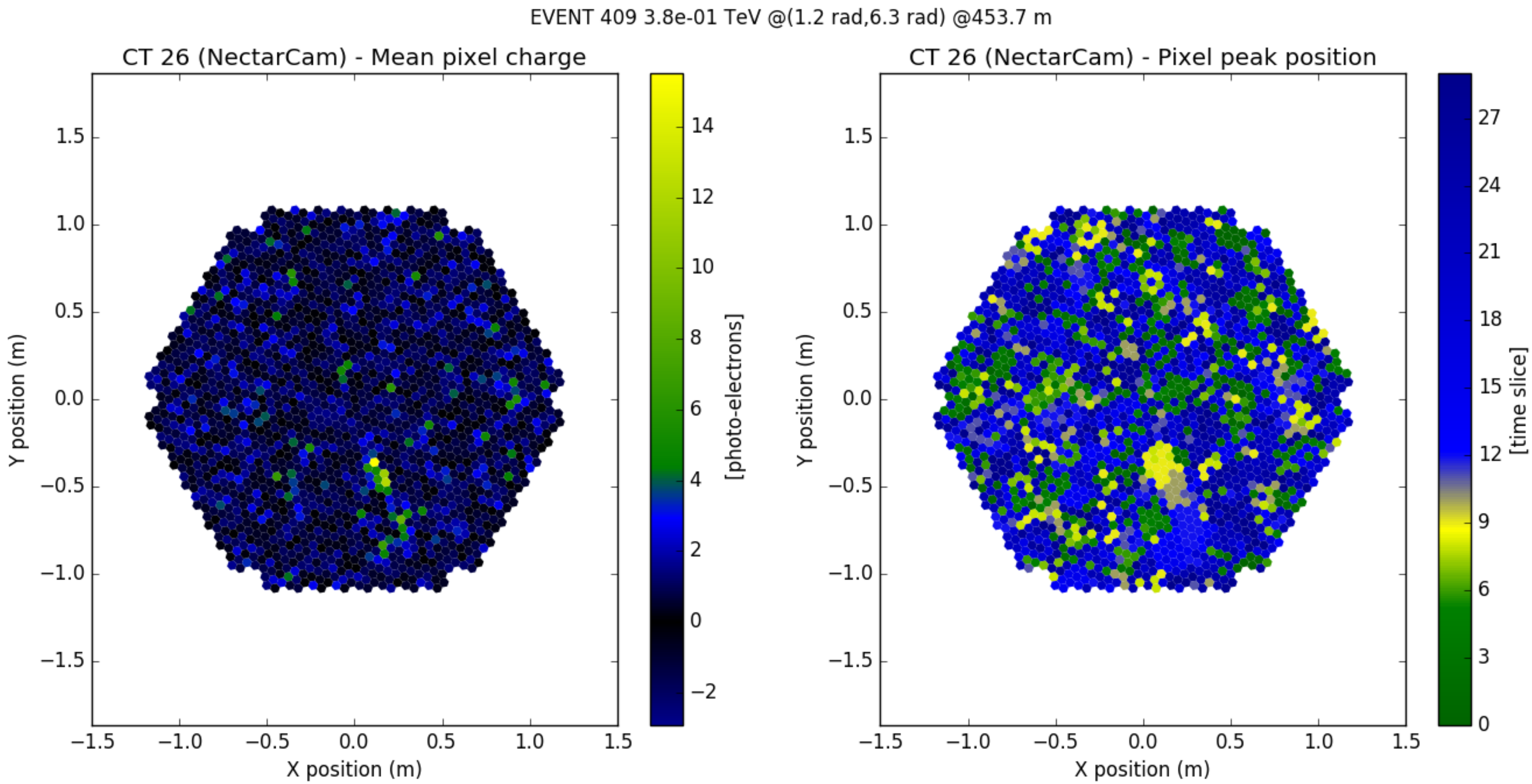


- Split into the 3 data levels
  - R1 – Camera/MC pre-calibration into photoelectron (or ADC) samples
  - DL0 – Data reduction algorithms before storage
  - DL1 – Charge extraction and conversion into photoelectrons, one value per pixel.

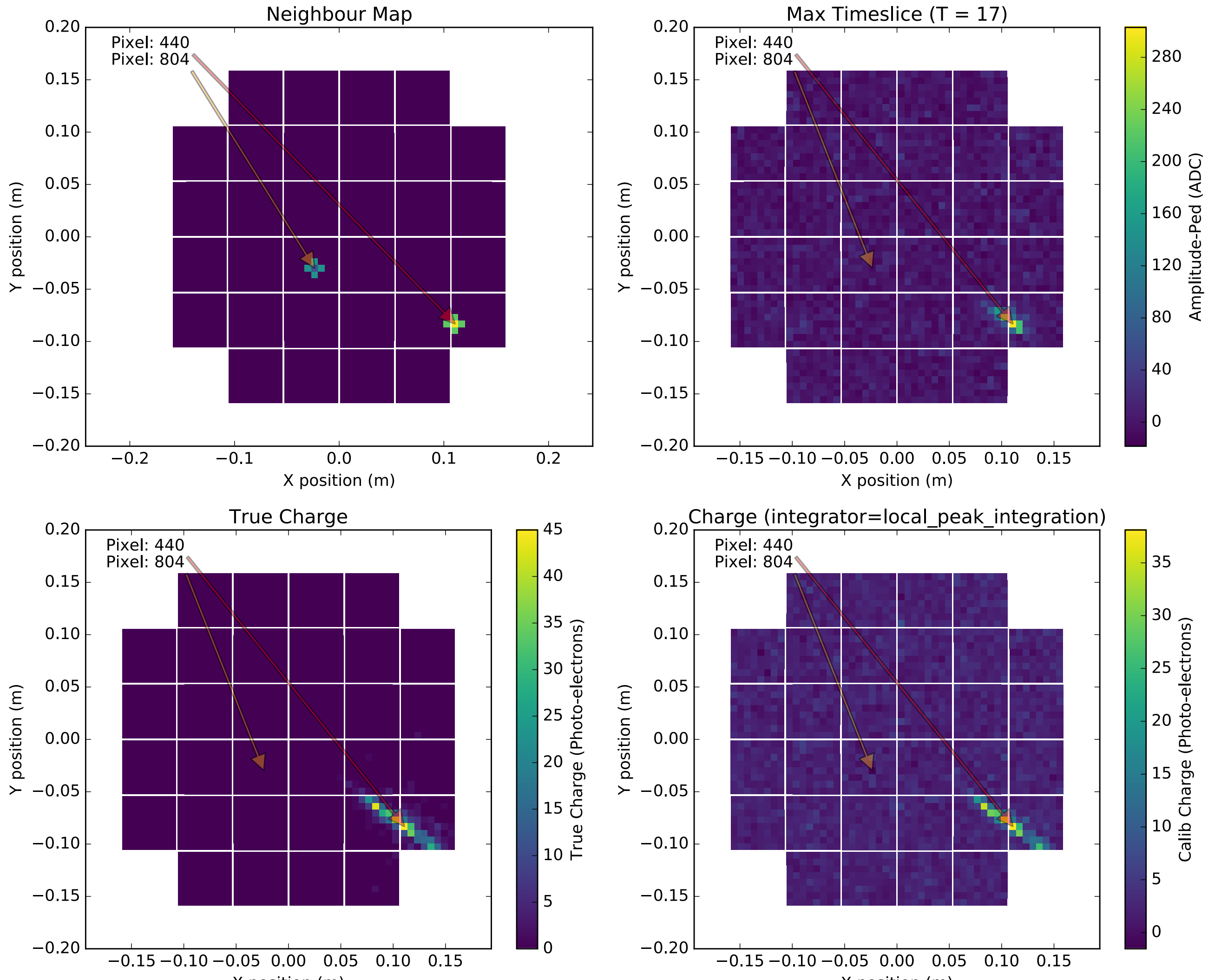
- First two calibration stages are not usually done by the offline calibration, except for MC and prototyping. However it is good that the same methods may be available to the offline analysis.

examples/calibration_pipeline.py

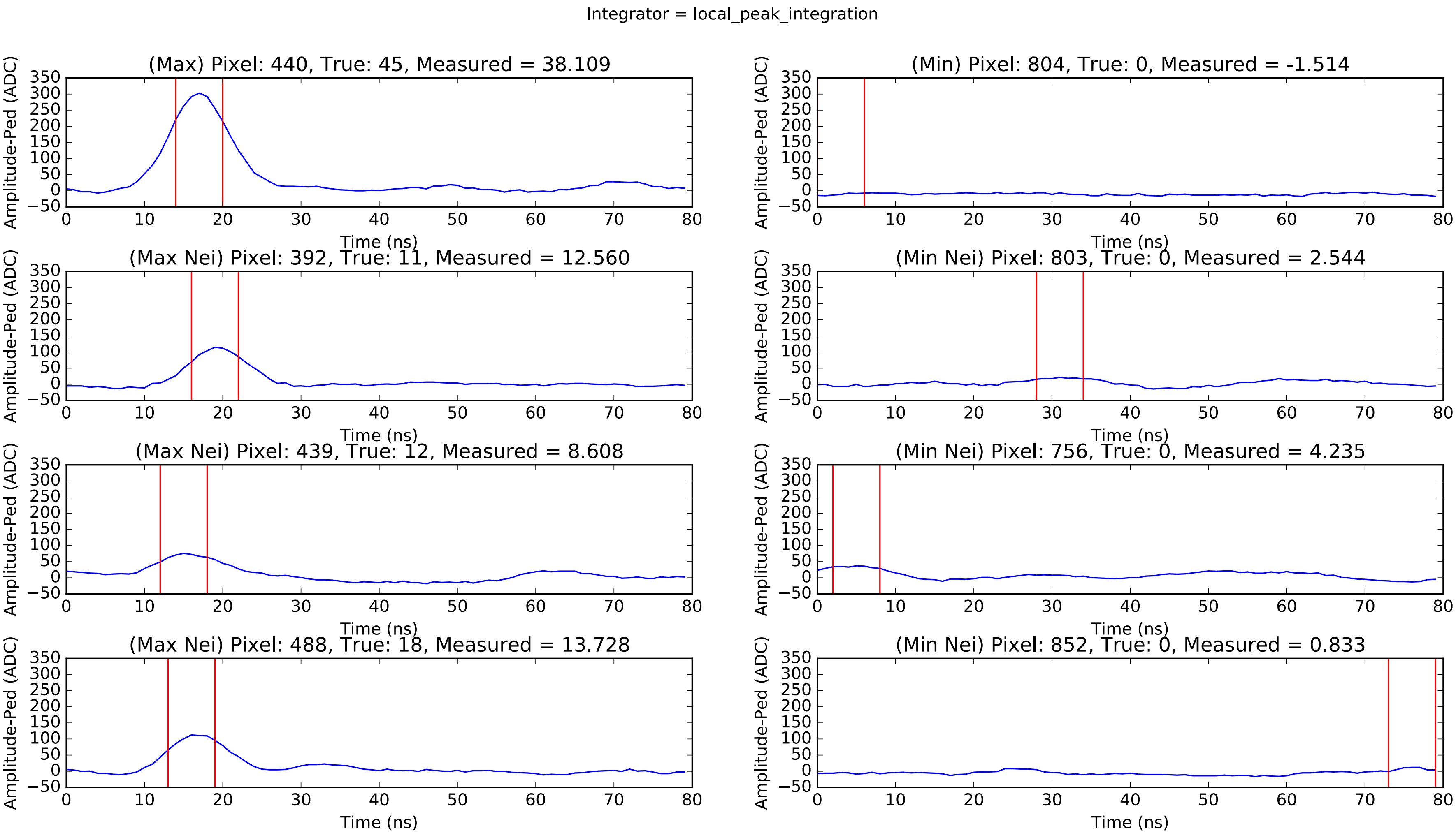
p examples/calibration_pipeline.py -D



examples/display_integrator.py



examples/display_integrator.py



Verification: Charge Resolution