ctapipe

Developing a *Python* framework for low-level data processing for CTA

Karl Kosack
CEA Paris-Saclay / CTA Observatory

PyGamma19, Heidelberg

What is the goal of low-level CTA data processing?

Generate data that science users want to start with!

What is the goal of low-level CTA data processing?

Generate data that science users want to start with!

Produce Event Lists

- Reconstruct single “events” (gamma ray or background), select gamma-ray candidates, provide Energy and Position information
  - ➤ For real data
  - ➤ For simulated data

What is the goal of low-level CTA data processing?

Generate data that science users want to start with!

Produce Event Lists

- Reconstruct single “events” (gamma ray or background), select gamma-ray candidates, provide Energy and Position information
  - ➤ For real data
  - ➤ For simulated data

Produce Instrument Response Functions

- Input is large library of reconstructed event lists
  - ➤ From Simulations (e.g. PSF, A_{eff}, E_{mig})
  - ➤ From Real Data (e.g. residual background rate)

What is the goal of low-level CTA data processing?

Generate data that science users want to start with!

Produce Event Lists

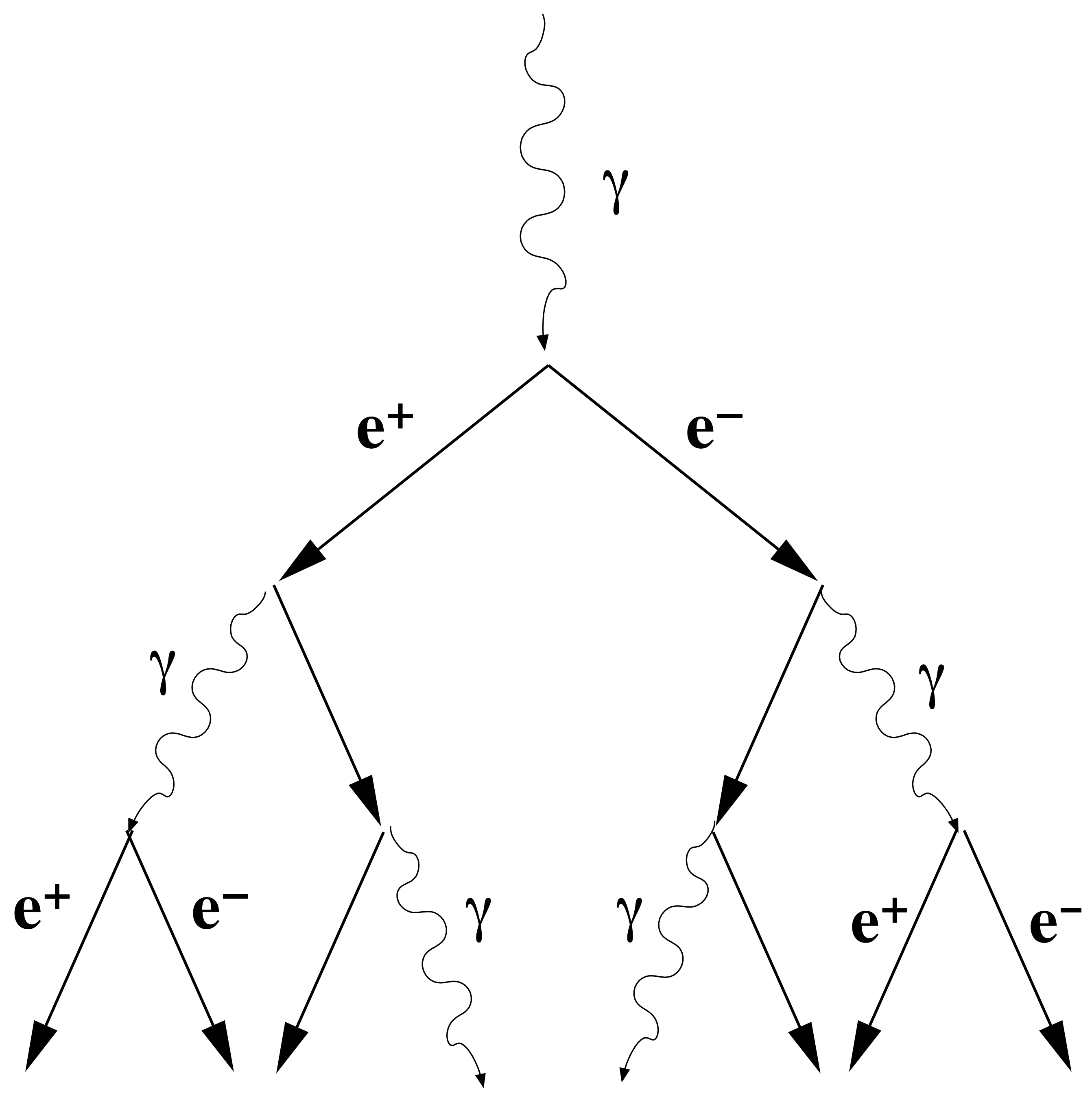
- Reconstruct single “events” (gamma ray or background), select gamma-ray candidates, provide Energy and Position information
  - ➤ For real data
  - ➤ For simulated data

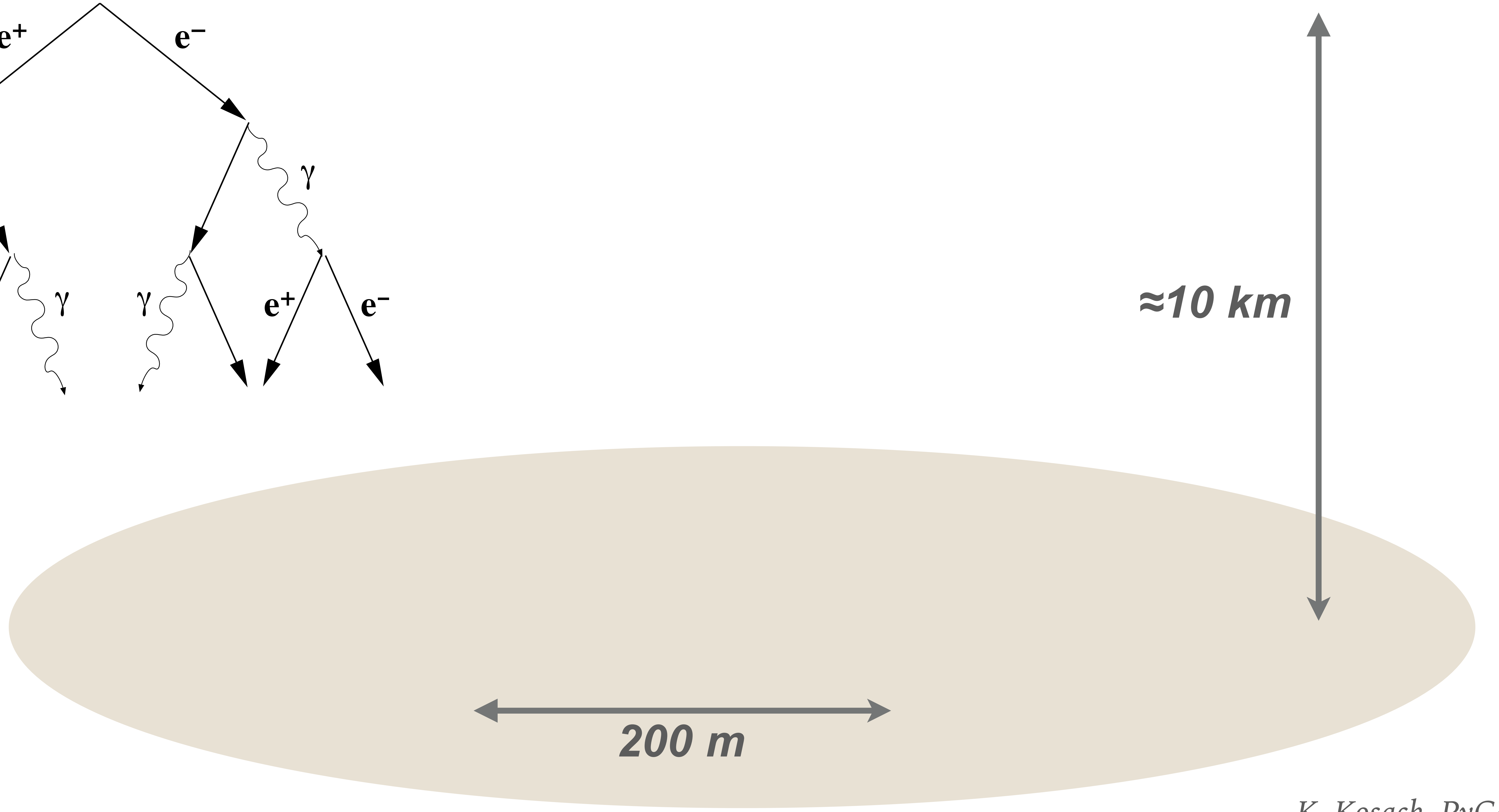
Produce Instrument Response Functions

- Input is large library of reconstructed event lists
  - ➤ From Simulations (e.g. PSF, A_{eff}, E_{mig})
  - ➤ From Real Data (e.g. residual background rate)

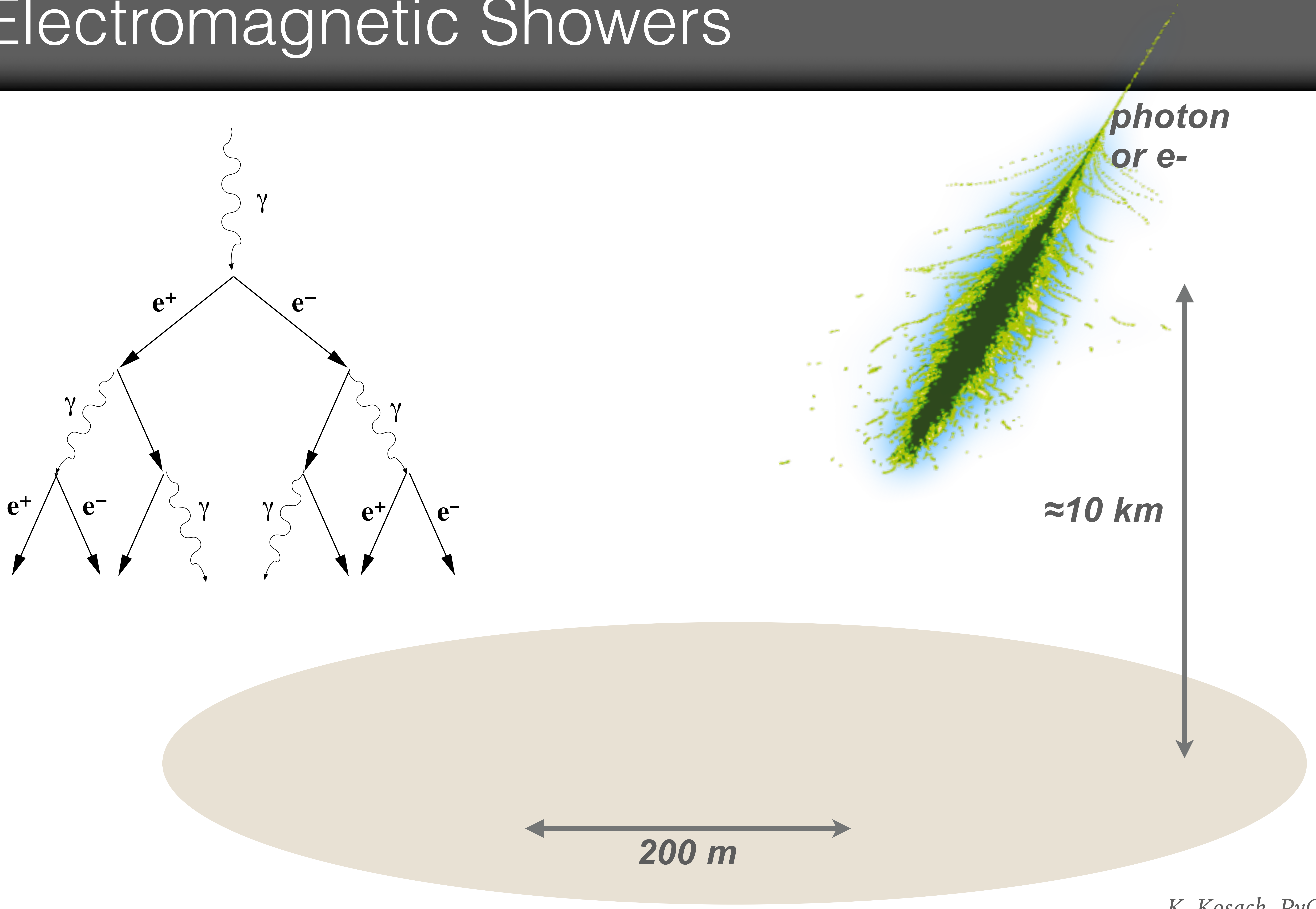
Support Development and Verification of Prototype Telescopes

Electromagnetic Showers

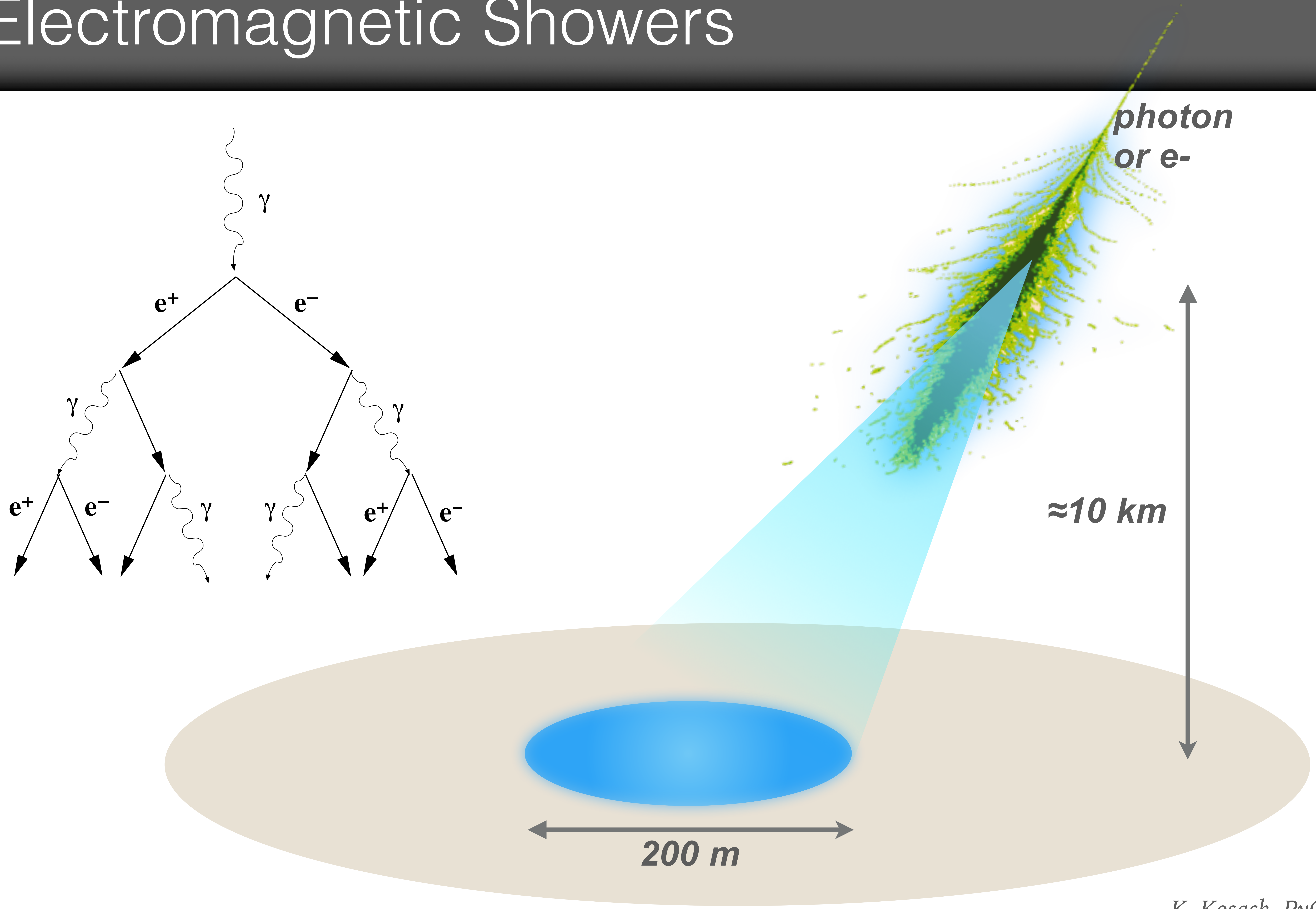




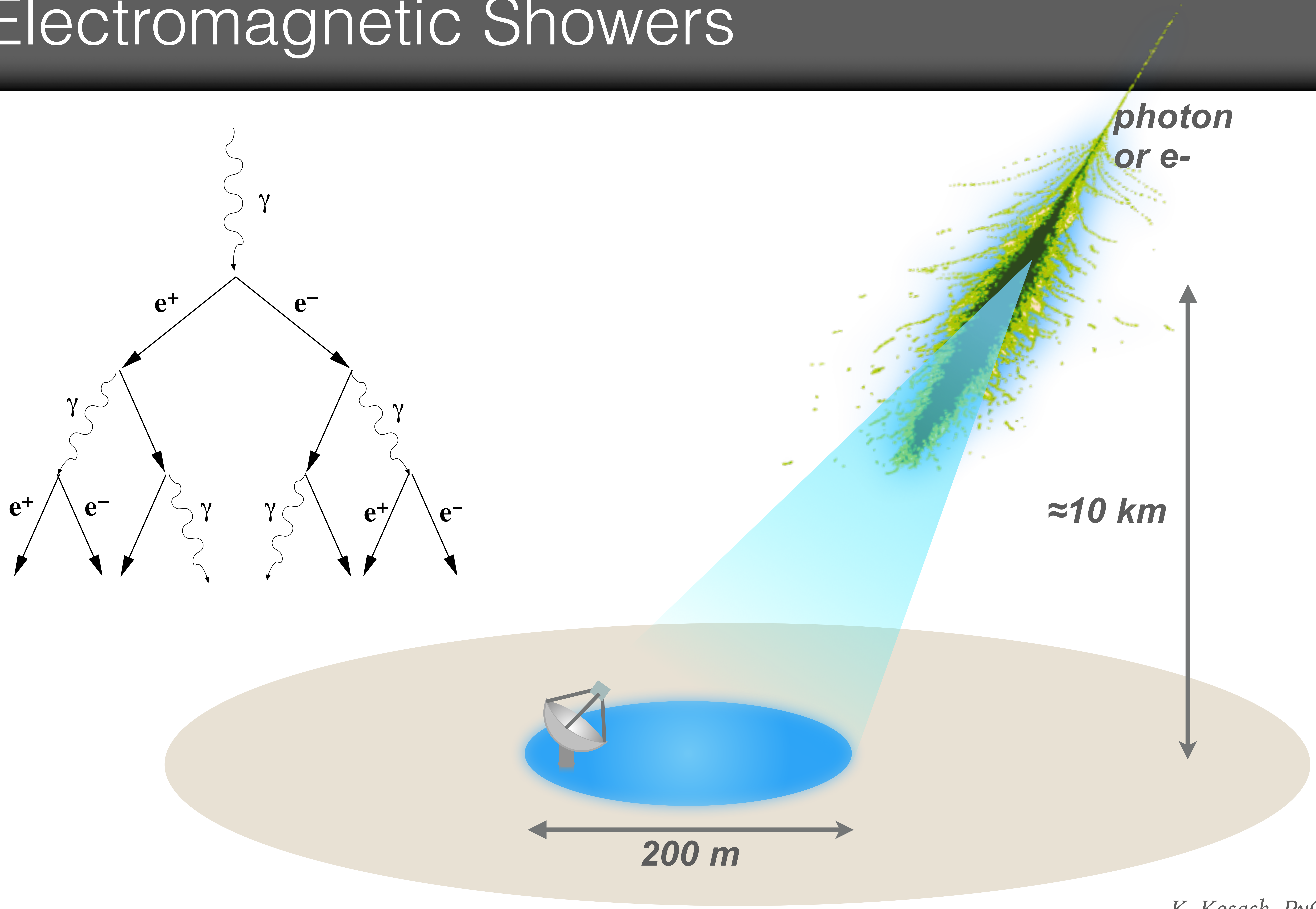
Electromagnetic Showers



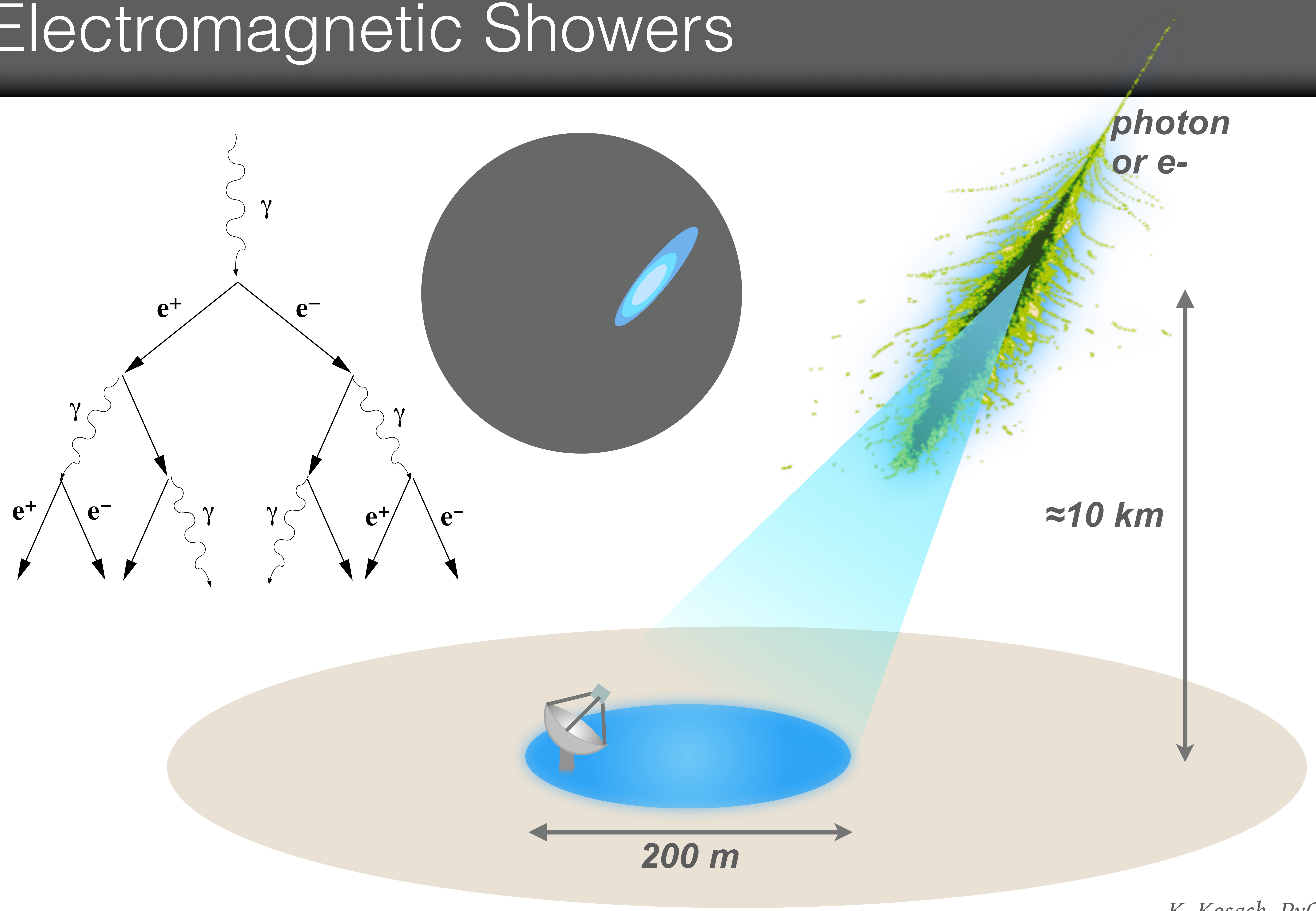
Electromagnetic Showers



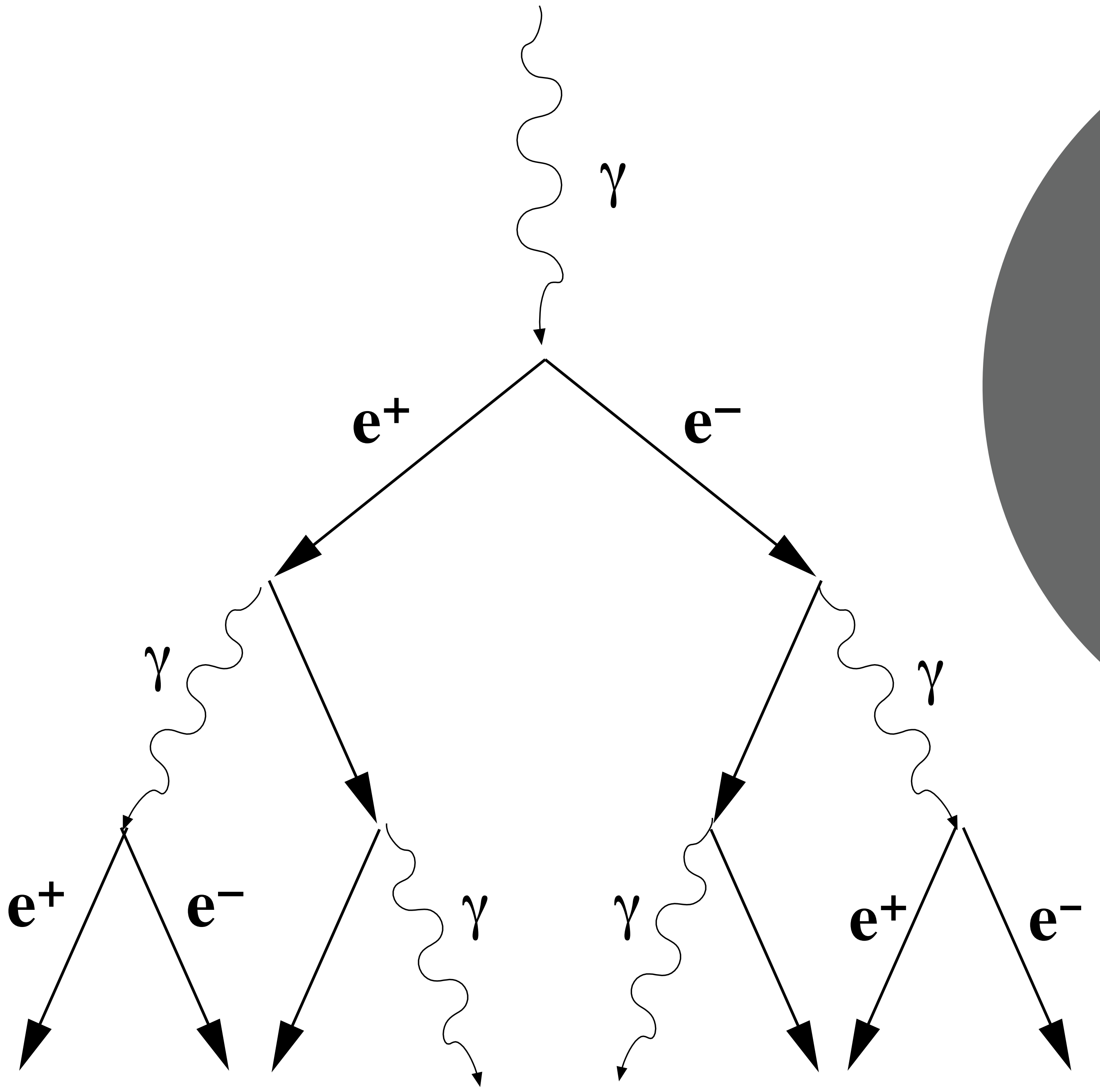
Electromagnetic Showers

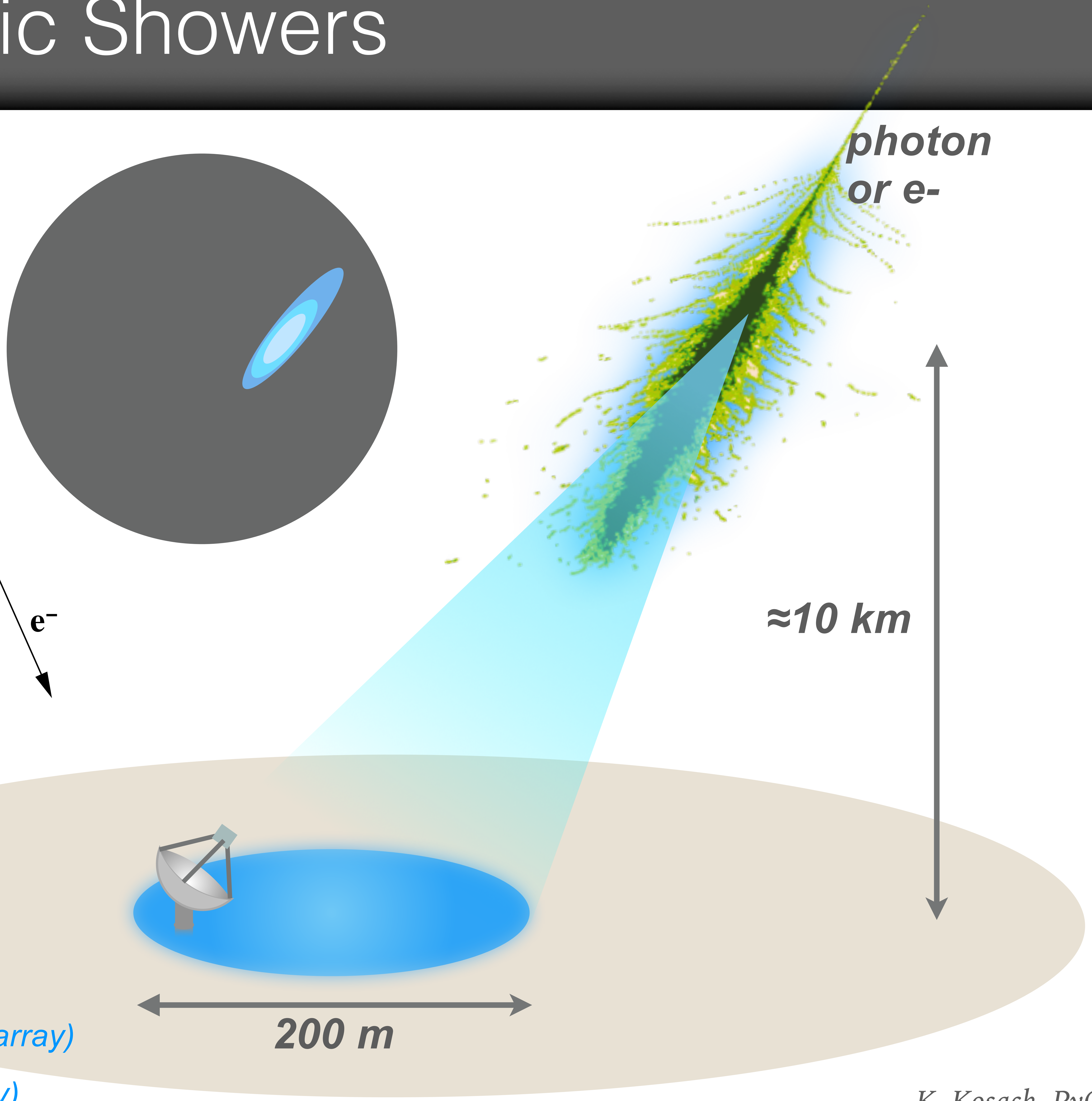


Electromagnetic Showers



Electromagnetic Showers

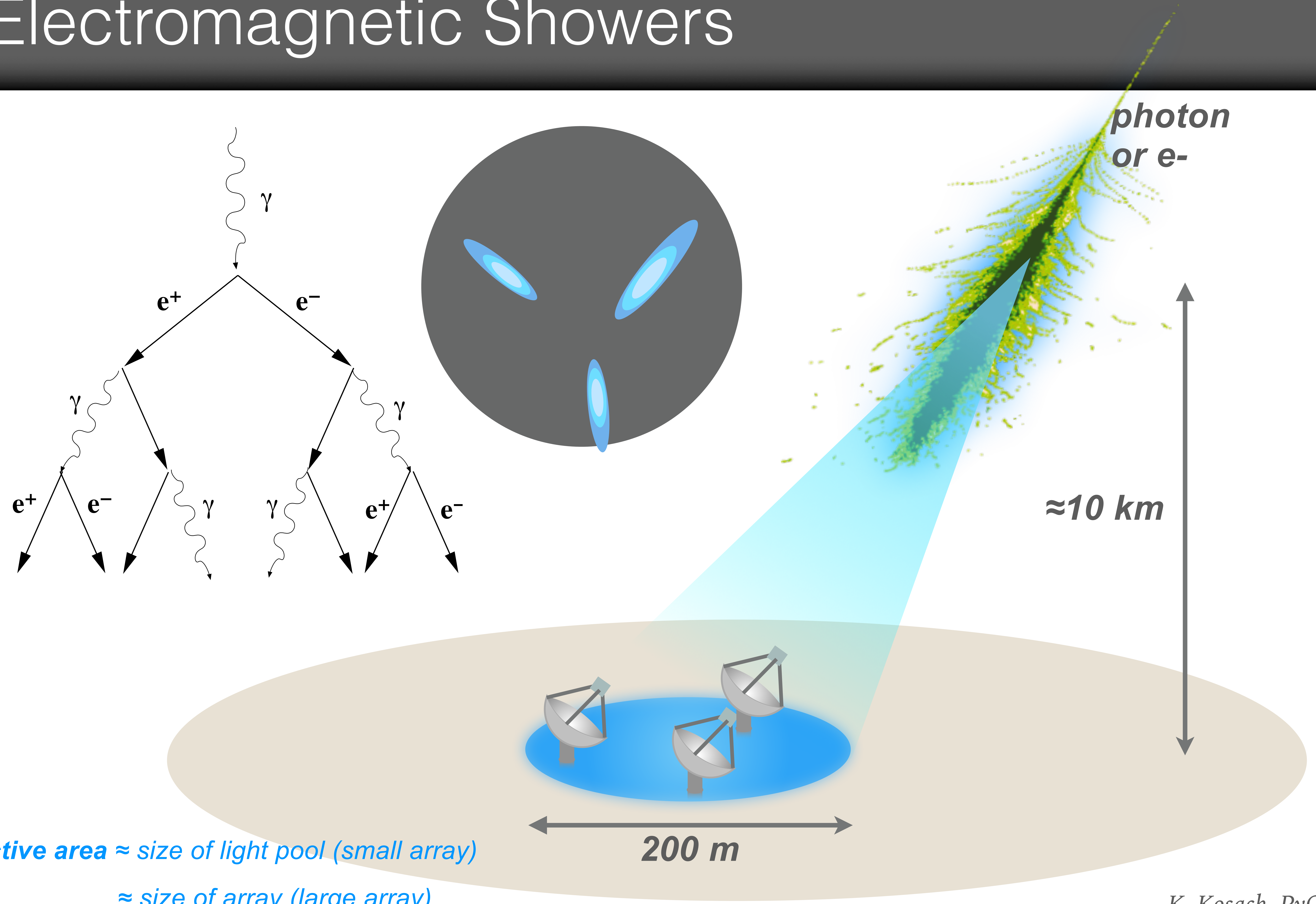




effective area *≈ size of light pool (small array)*

≈ size of array (large array)

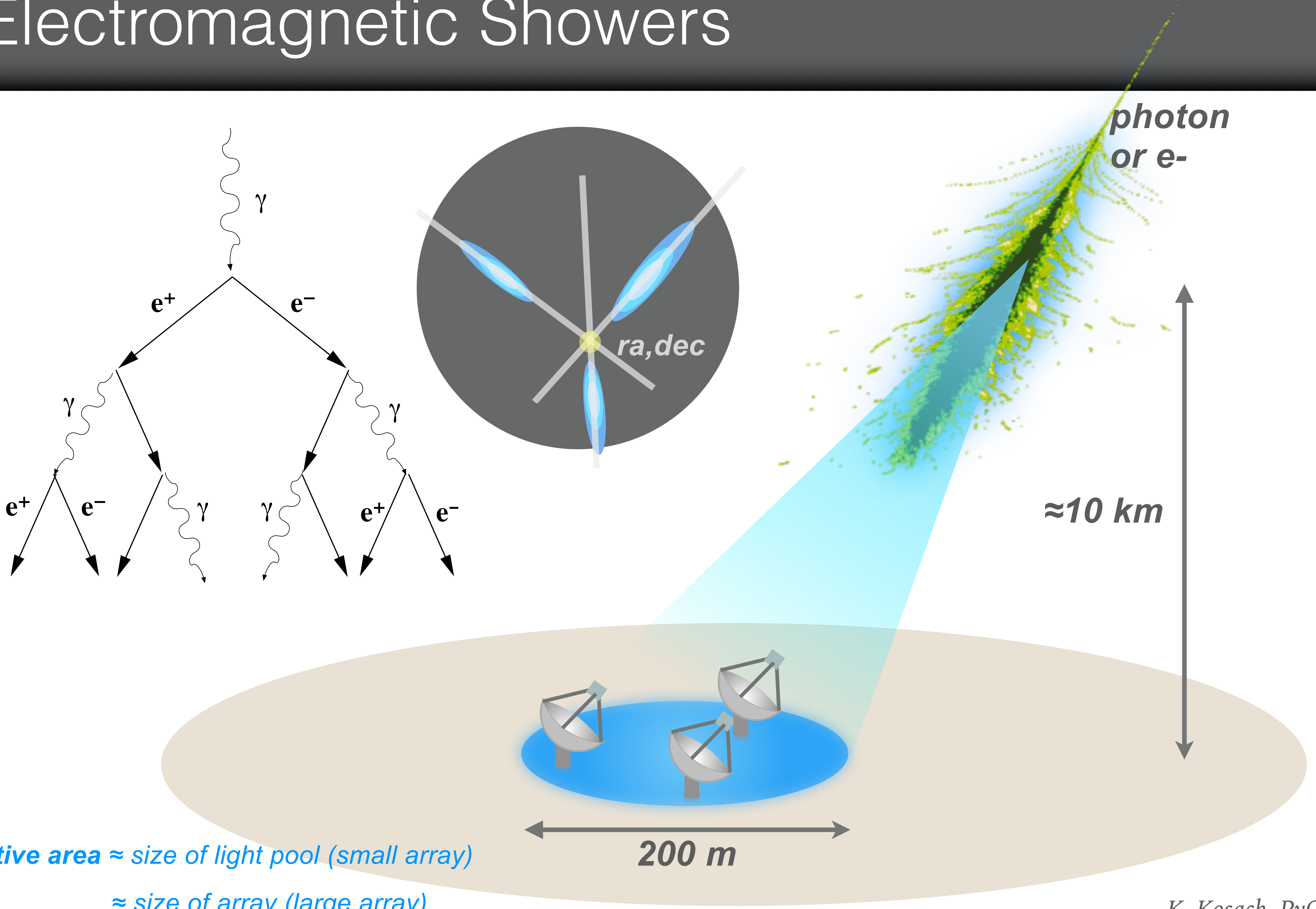
Electromagnetic Showers



effective area ≈ *size of light pool (small array)*

≈ *size of array (large array)*

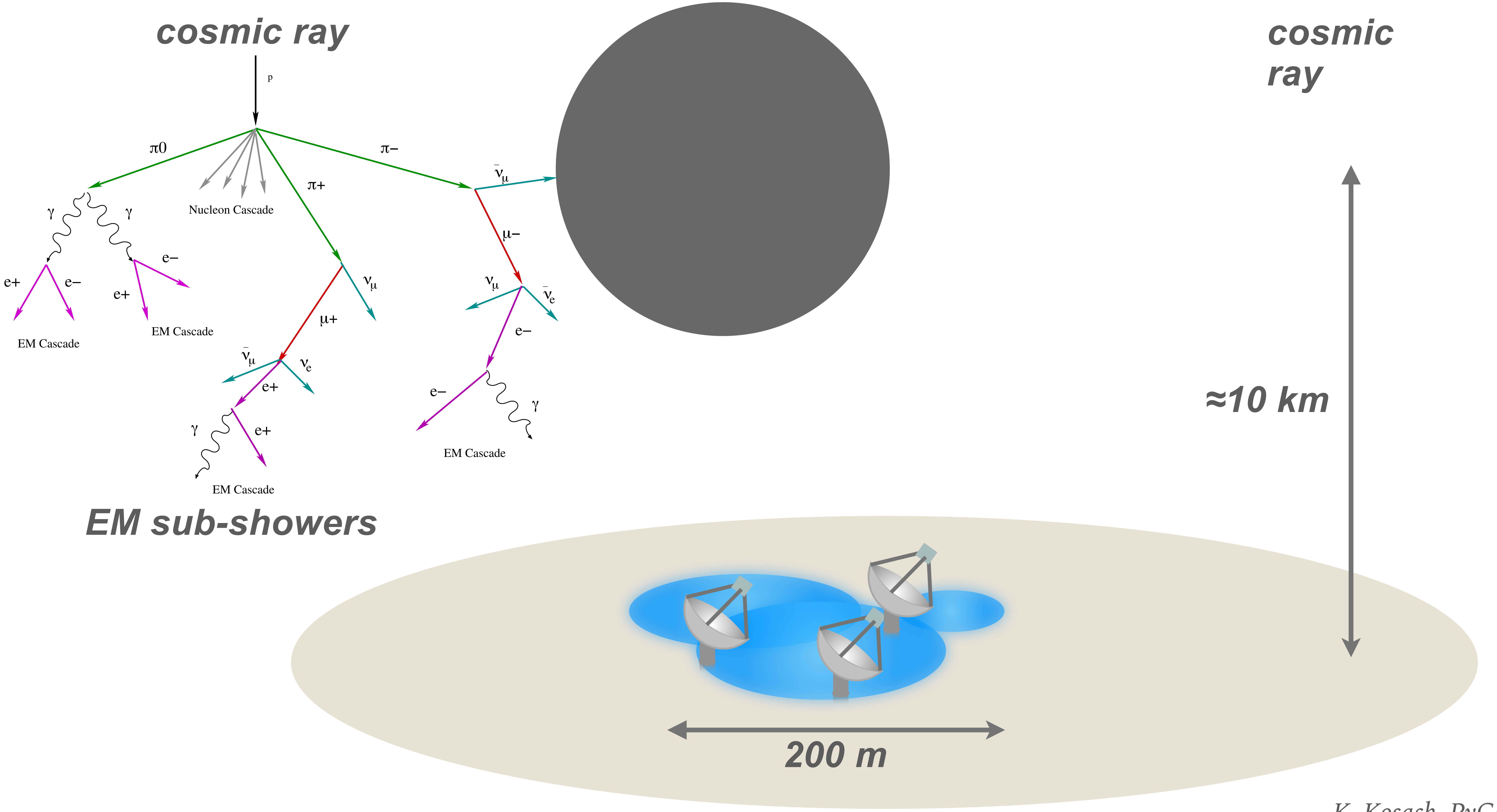
Electromagnetic Showers



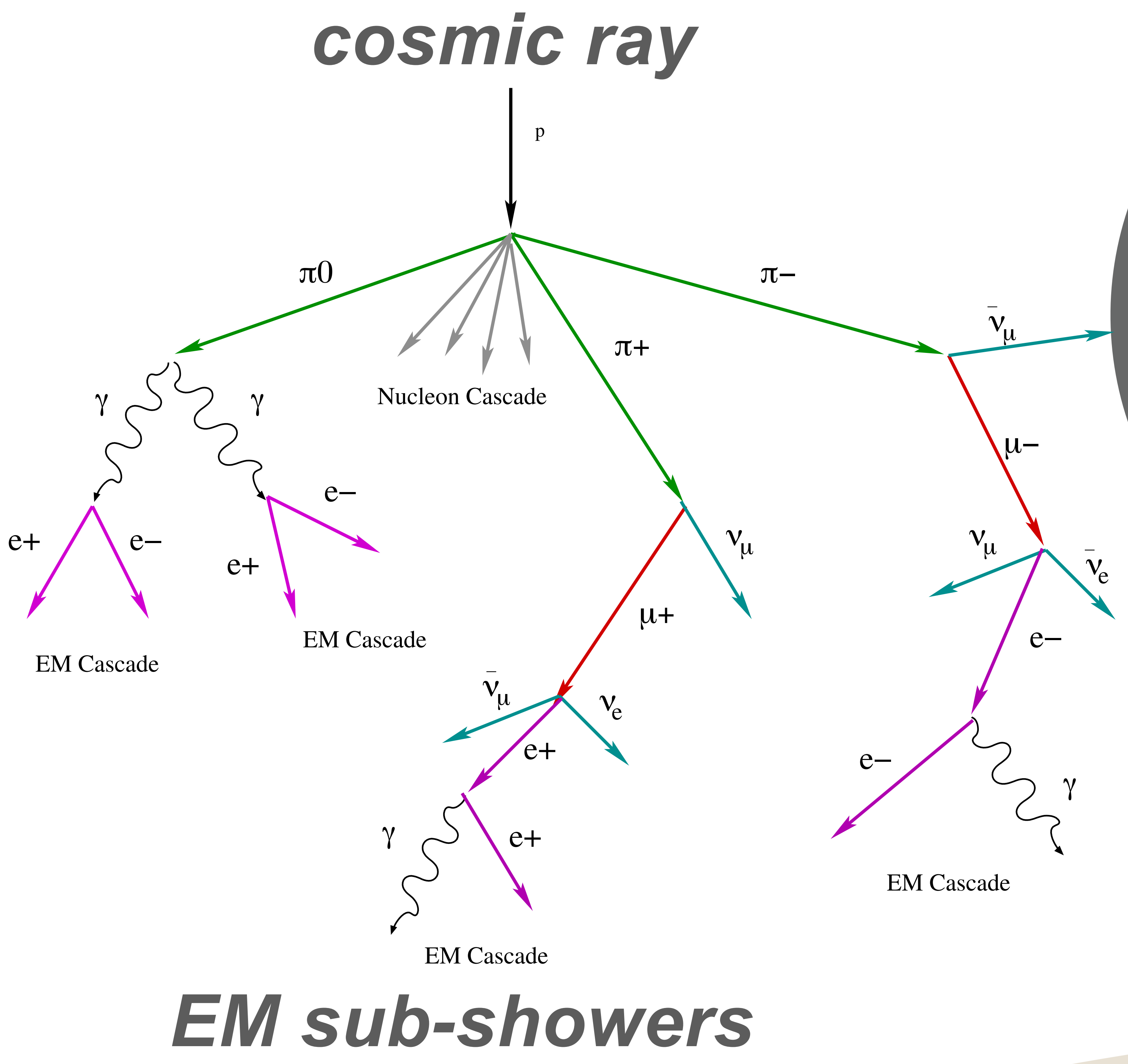
effective area *≈ size of light pool (small array)*

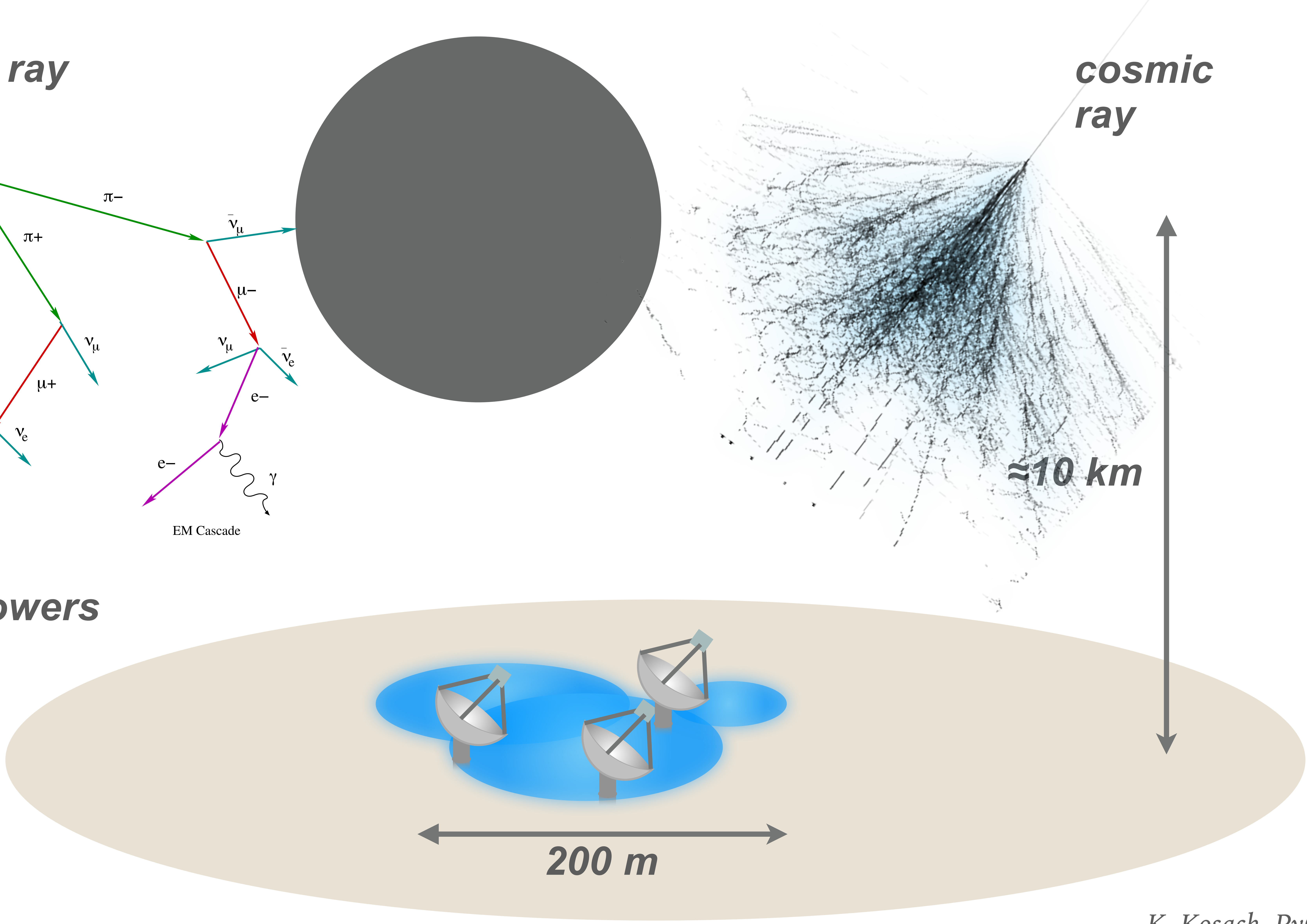
≈ size of array (large array)

Hadronic Showers

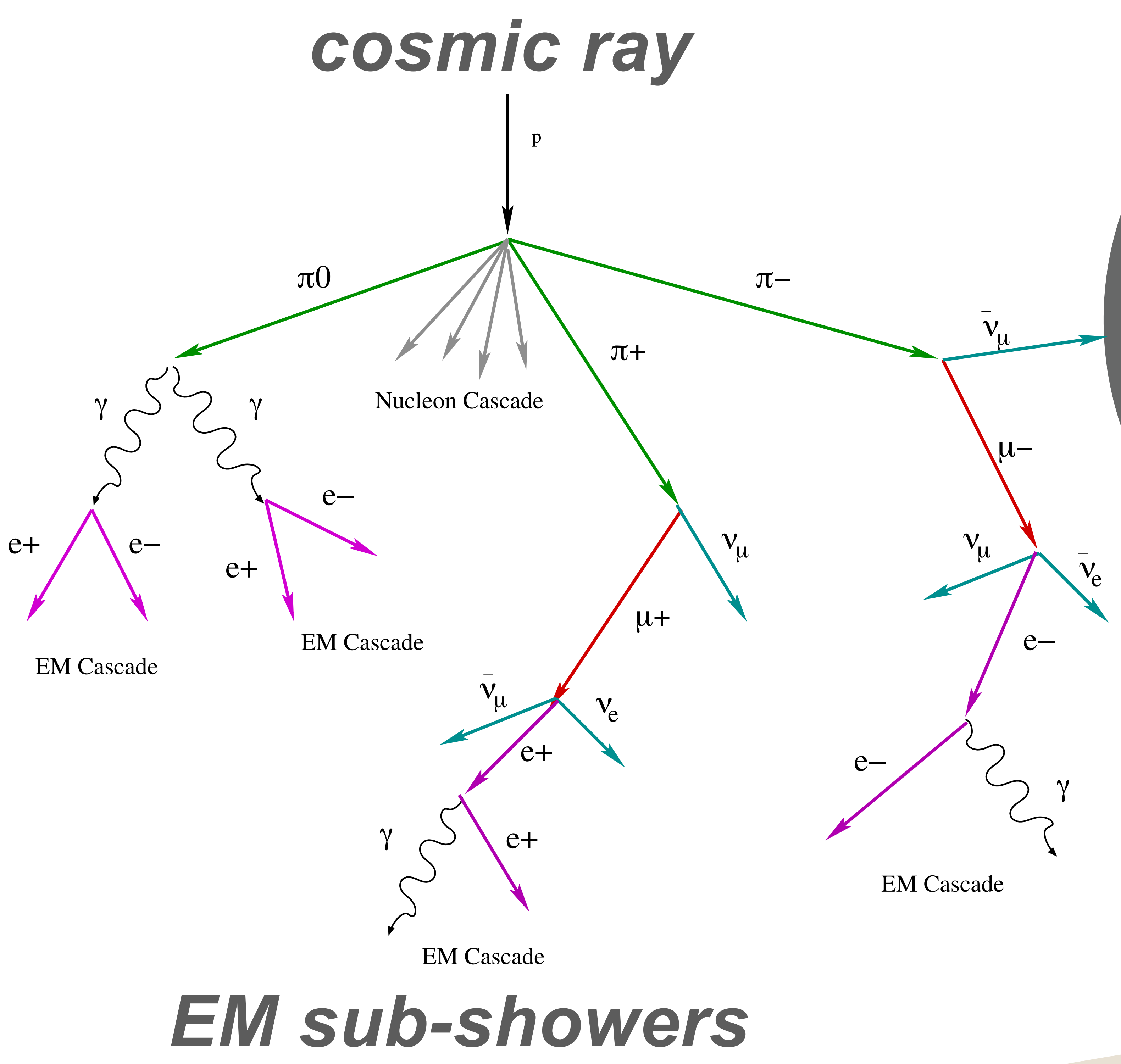


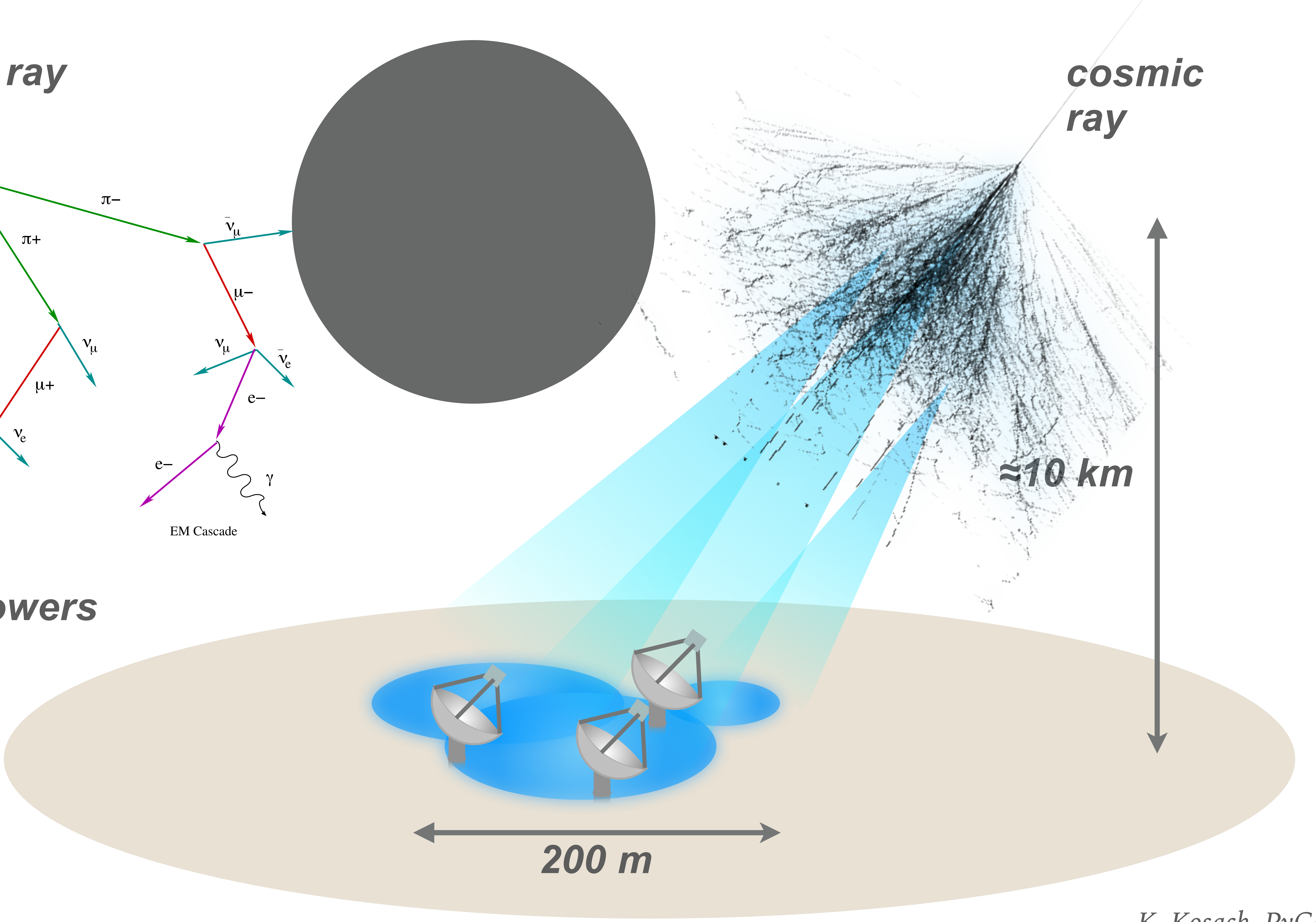
Hadronic Showers



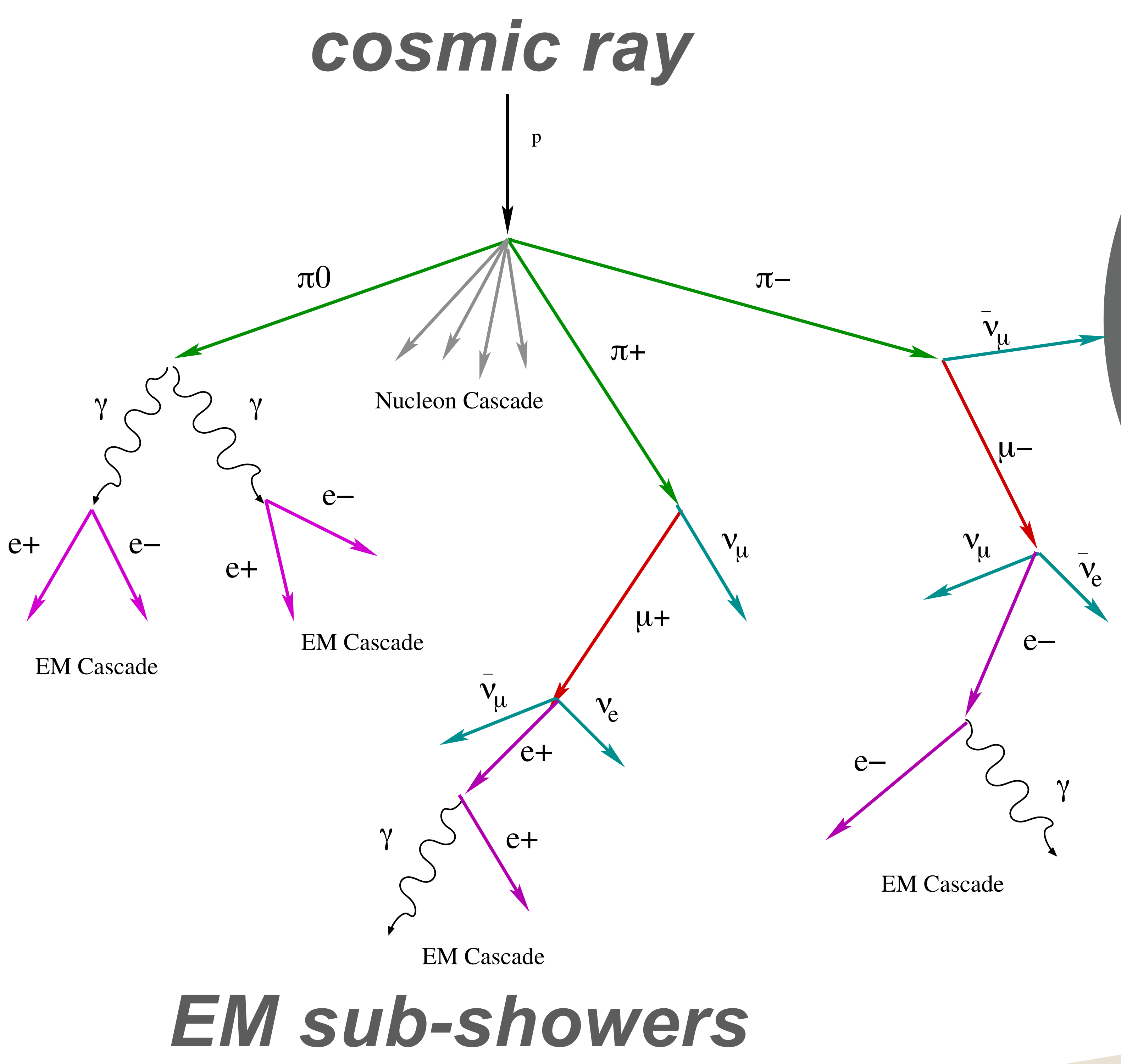


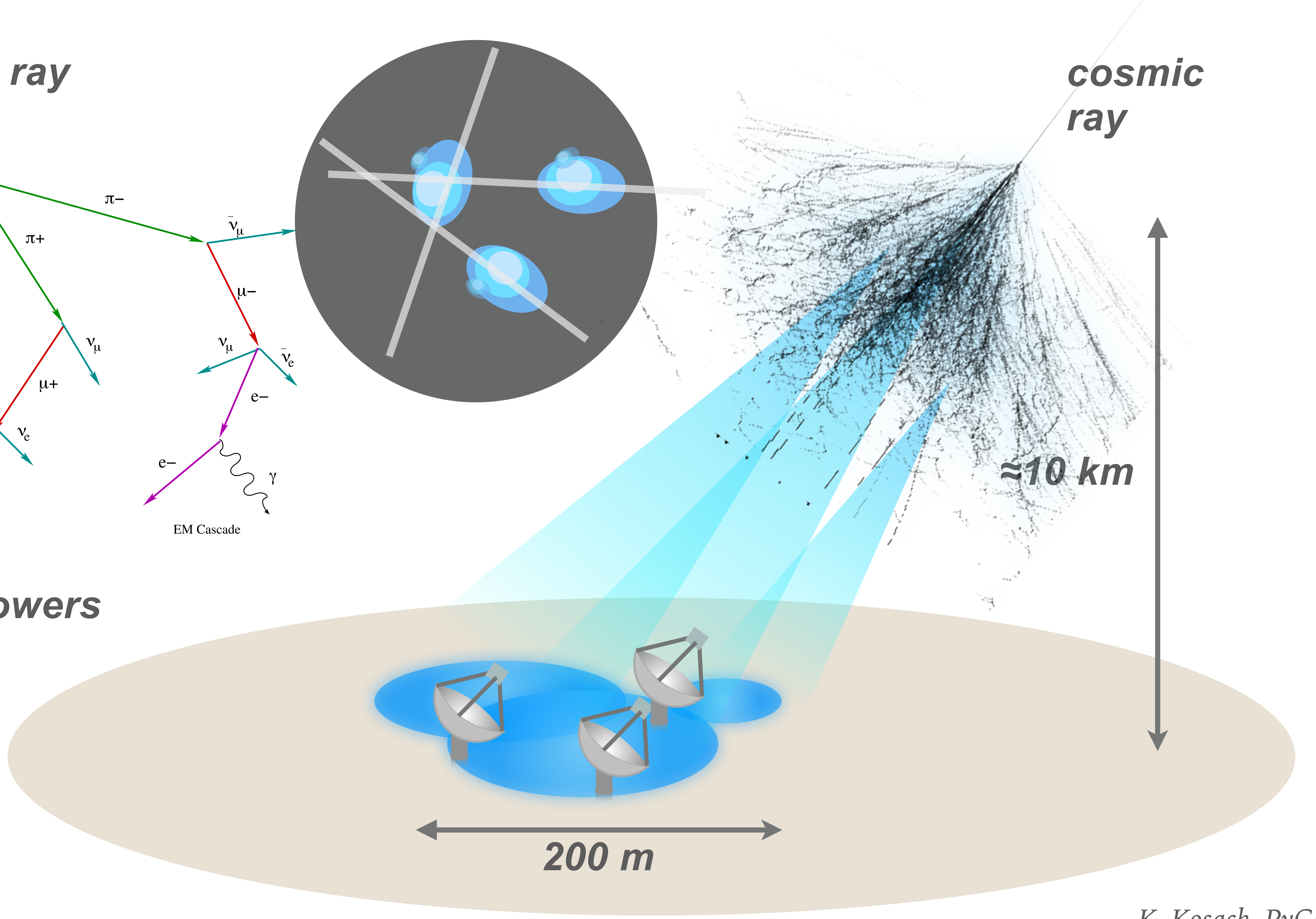
Hadronic Showers



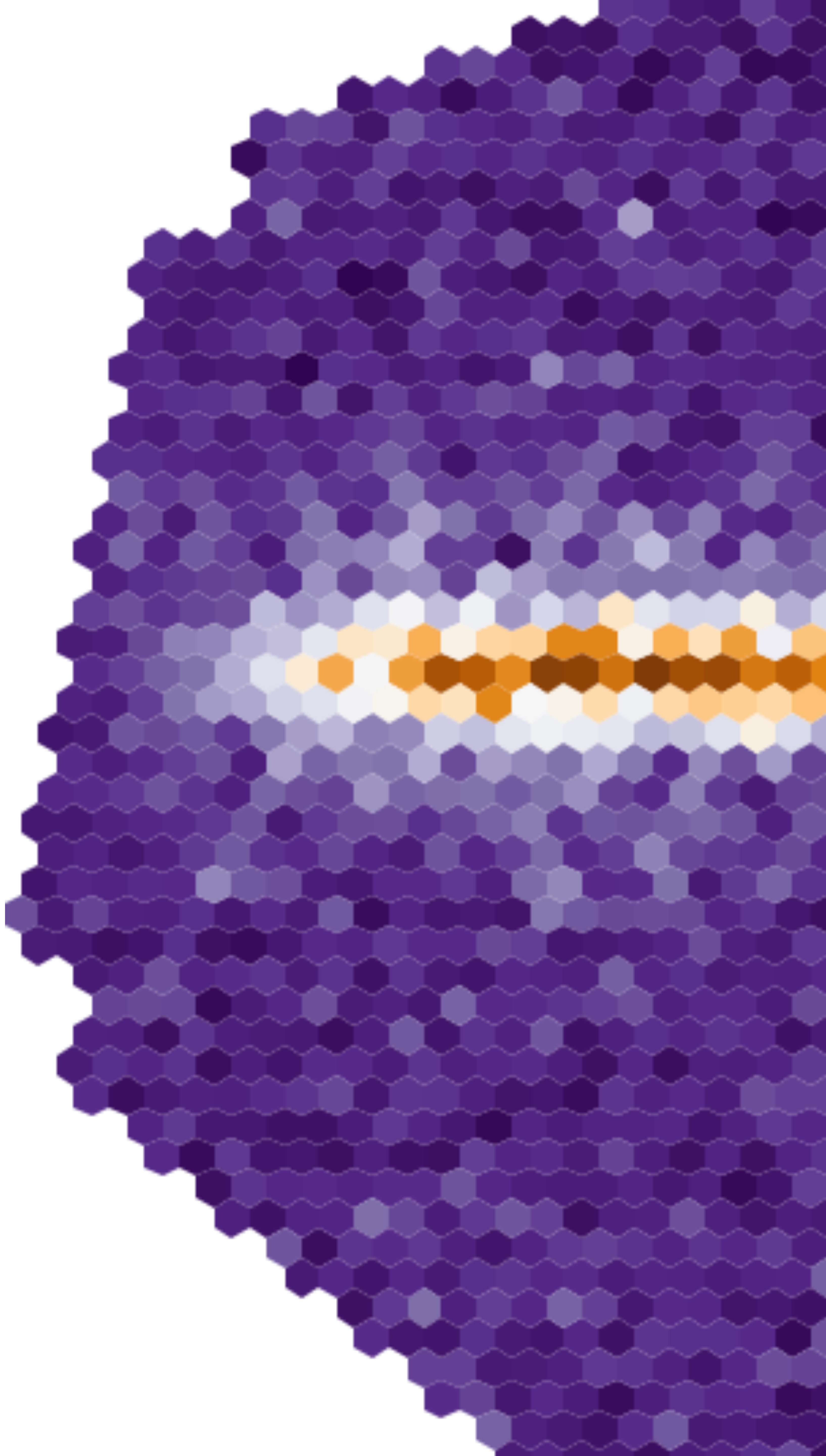


Hadronic Showers

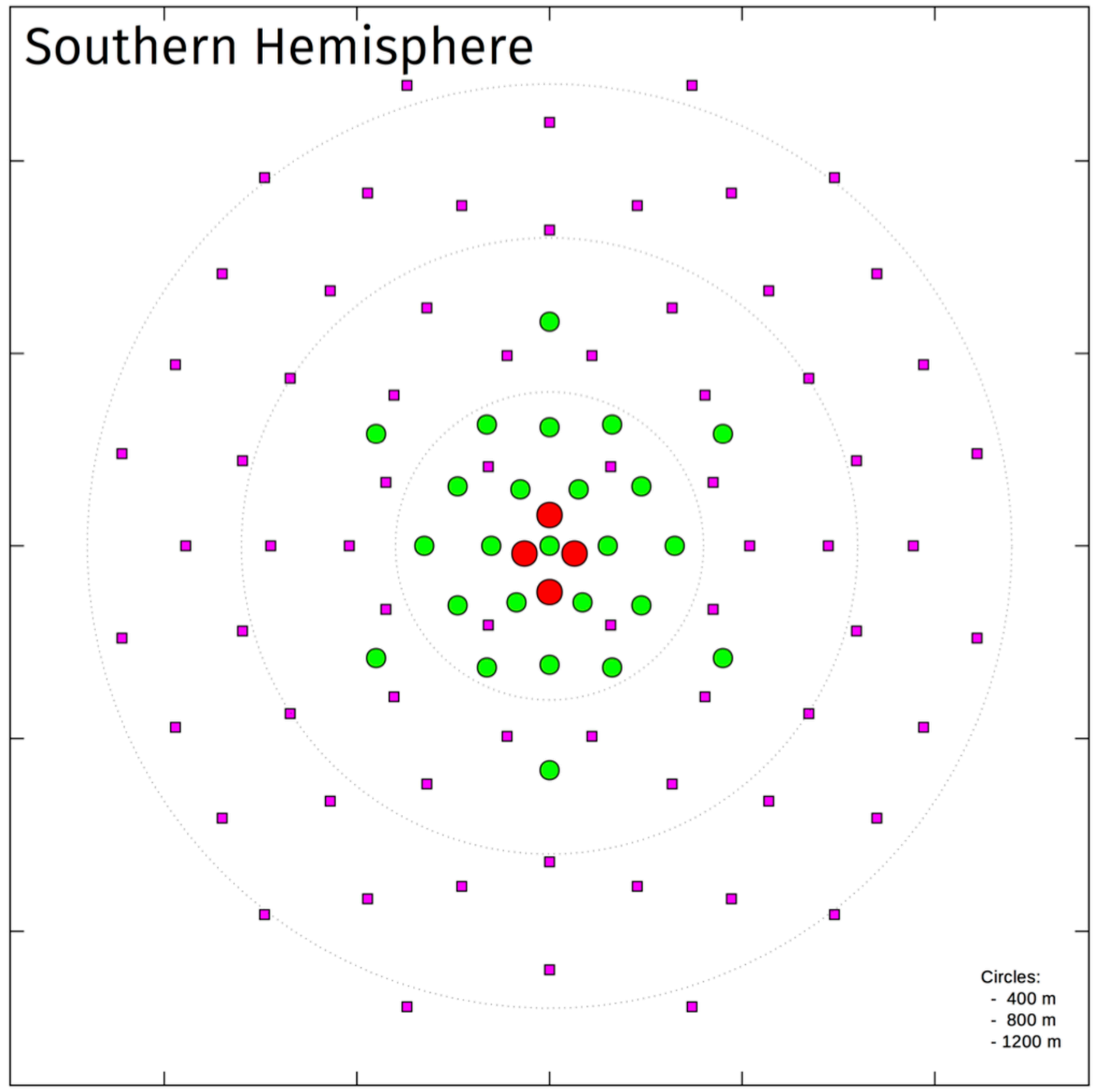




Challenges and Lessons



Challenge: Data Volume



4 LSTs, 25 MSTs, 70 SSTs

CTA data size (as powers of 10 only)**:**

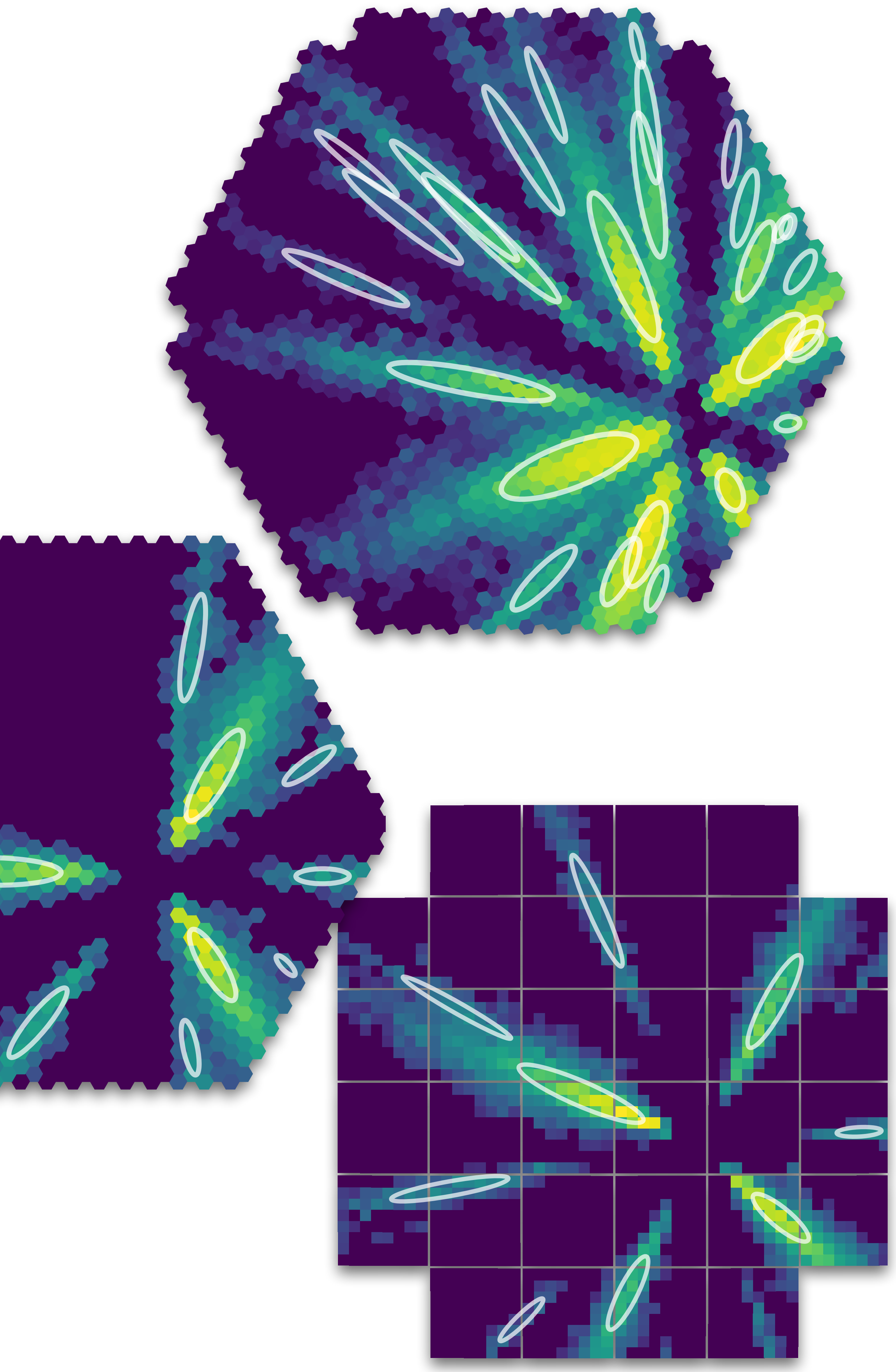
- 100 telescopes (CTA-south)
- 10,000 array triggers per second
- 10 telescopes on average per trigger
- 10-100 image frames per telescope camera
- 1,000 to 10,000 pixels per camera
- **÷ 100** lossy and lossless compression

→ *O(10) PB/yr of raw data*

Monte-Carlo Simulations
(basically continuously, similarly data volume)

Yearly reprocessing of *all* data with new calibration and reconstruction (30 year lifetime of CTA…)

Challenge: Complex Instrument (1)



Camera and optics complexity

- 7 cameras (for now)
  - ➤ Hexagonal and square pixels, Pixel gaps
  - ➤ Time-series readouts vs peak times, multiple sampling frequencies
- 6 telescope optics: 1 and 2-mirror systems, various mirror geometries
- 4+ raw data formats

CAVEAT: Much of this will simplify before the final construction phase… but still at least 3 cameras and 3 optics types, and likely many generations/variations in each.

Challenge: Complex Instrument (2)

Atmosphere and Observation Condition Complexity:

Instrument's response changes with:

- Gamma-ray **Energy**
- **Position** in Field-of-View
- **Zenith Angle** (elevation): atmosphere thickness
- **Azimuth**: Earth magnetic field orientation
- **Ground position** of shower in the array / Number of telescopes of each type that trigger / exactly which telescopes trigger!
- **Subarray Choice**
- **Atmosphere Density** profile
- Optical **Night-Sky-Background** light level (Moon, Zodiacal light, Light pollution)
- Atmosphere **Aerosol** content profile
- Detector Configuration (high voltage gain, etc)
- Analysis Configuration (reconstruction algorithm, discrimination strength, …)

Challenge: Complex Instrument (2)

Atmosphere and Observation Condition Complexity:

Instrument's response changes with:

Change during an observation

- Gamma-ray **Energy**
- **Position** in Field-of-View
- **Zenith Angle** (elevation): atmosphere thickness
- **Azimuth**: Earth magnetic field orientation
- **Ground position** of shower in the array / Number of telescopes of each type that trigger / exactly which telescopes trigger!
- **Subarray Choice**

Change between observations

- **Atmosphere Density** profile
- Optical **Night-Sky-Background** light level (Moon, Zodiacal light, Light pollution)
- Atmosphere **Aerosol** content profile
- Detector Configuration (high voltage gain, etc)
- Analysis Configuration (reconstruction algorithm, discrimination strength, …)

Challenge: Complex Instrument (2)

Atmosphere and Observation Condition Complexity:

Instrument's response changes with:

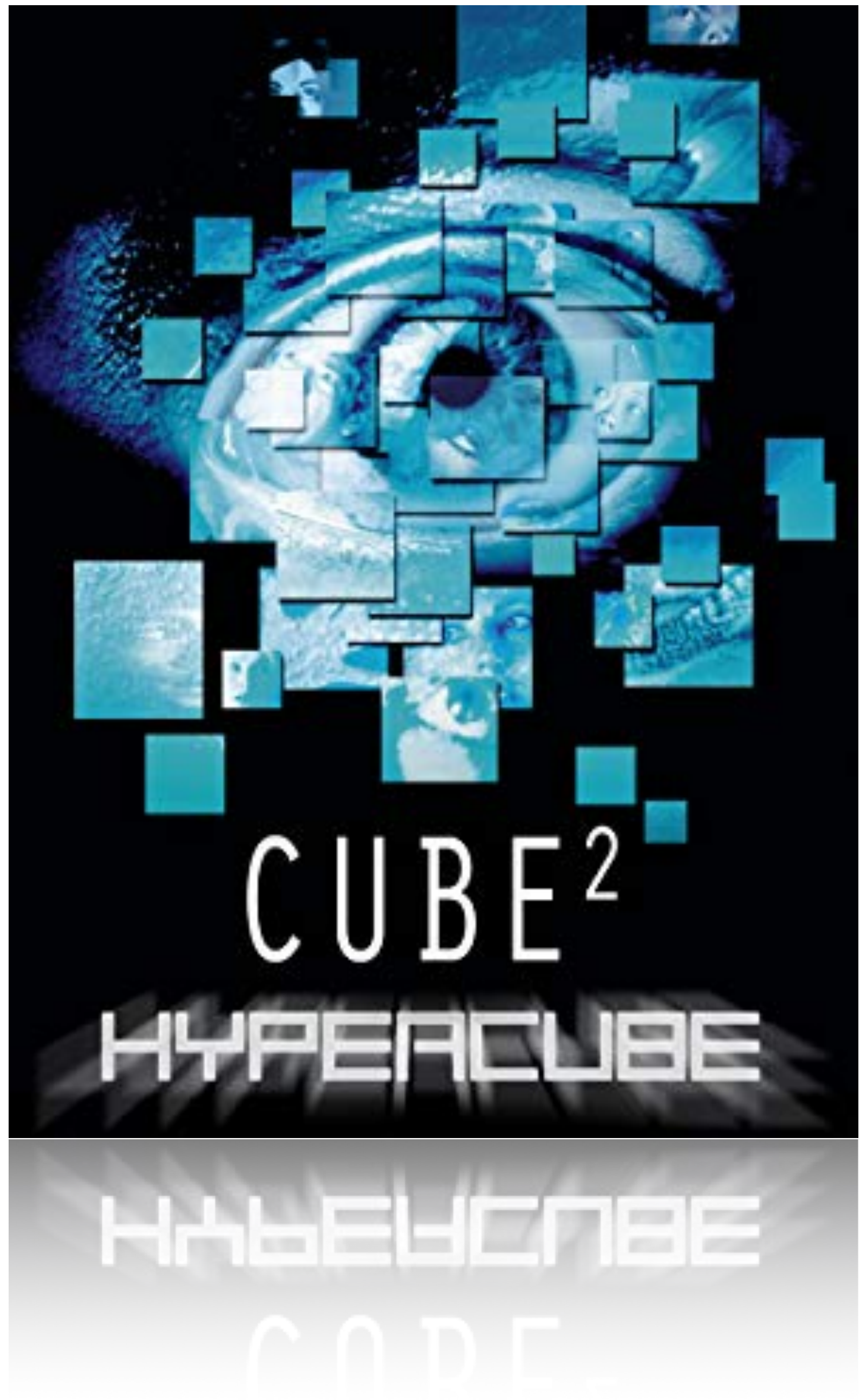
Change during an observation

- Gamma-ray **Energy**
- **Position** in Field-of-View
- **Zenith Angle** (elevation): atmosphere thickness
- **Azimuth**: Earth magnetic field orientation
- **Ground position** of shower in the array / Number of telescopes of each type that trigger / exactly which telescopes trigger!
- **Subarray Choice**

Change between observations

- **Atmosphere Density** profile
- Optical **Night-Sky-Background** light level (Moon, Zodiacal light, Light pollution)
- Atmosphere **Aerosol** content profile
- Detector Configuration (high voltage gain, etc)
- Analysis Configuration (reconstruction algorithm, discrimination strength, …)

Potentially *very high-dimensional* **Instrumental Response Functions**!



Or lots of custom simulations…

K. Kosack, PyGamma19

Challenge: diverse developer needs

What Physicists Want:

- Small learning curve for unexperienced developers
- Easy to play with data and explore , interactivity
- Ability to *quickly* implement a new algorithms and cross-check
- Simple deterministic loops over events and sequences of algorithm steps

Challenge: diverse developer needs

What Physicists Want:

- Small learning curve for unexperienced developers
- Easy to play with data and explore , interactivity
- Ability to *quickly* implement a new algorithms and cross-check
- Simple deterministic loops over events and sequences of algorithm steps

What we eventually need:

- High-performance processing or PBs of data
  - ➤ Big-Data-style Parallelisation (map-reduce, streaming, etc.)
  - ➤ High-Performance Computing: efficient use of CPU / GPU
- Well-maintainable code (CTA = 30 years!)
- Involvement of computer scientists / engineers

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

- Leverage the Astronomy community! (vs Particle Physics)

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

- Leverage the Astronomy community! (vs Particle Physics)
- Make it lightweight

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Lessons Learned…

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

- Leverage the Astronomy community! (vs Particle Physics)
- Make it lightweight
- Make it friendly : Rich visualisations , tutorials, notebooks, easy to discover and explore

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Lessons Learned…

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

- Leverage the Astronomy community! (vs Particle Physics)
- Make it lightweight
- Make it friendly : Rich visualisations , tutorials, notebooks, easy to discover and explore
- Use standards and open tools (minimize custom code)

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Lessons Learned…

From Whipple 10m, HESS, MAGIC, VERTIAS, Fermi-LAT, IceCube, Antares, …

Greater astronomical community

Nearly all High Energy Experiments / Telescopes based their low-level analysis on C++ and CERN's ROOT framework

- Nearly all complained about it.
- Much of the code was bookkeeping and custom built astronomy algorithms (coordinates, regions, etc)

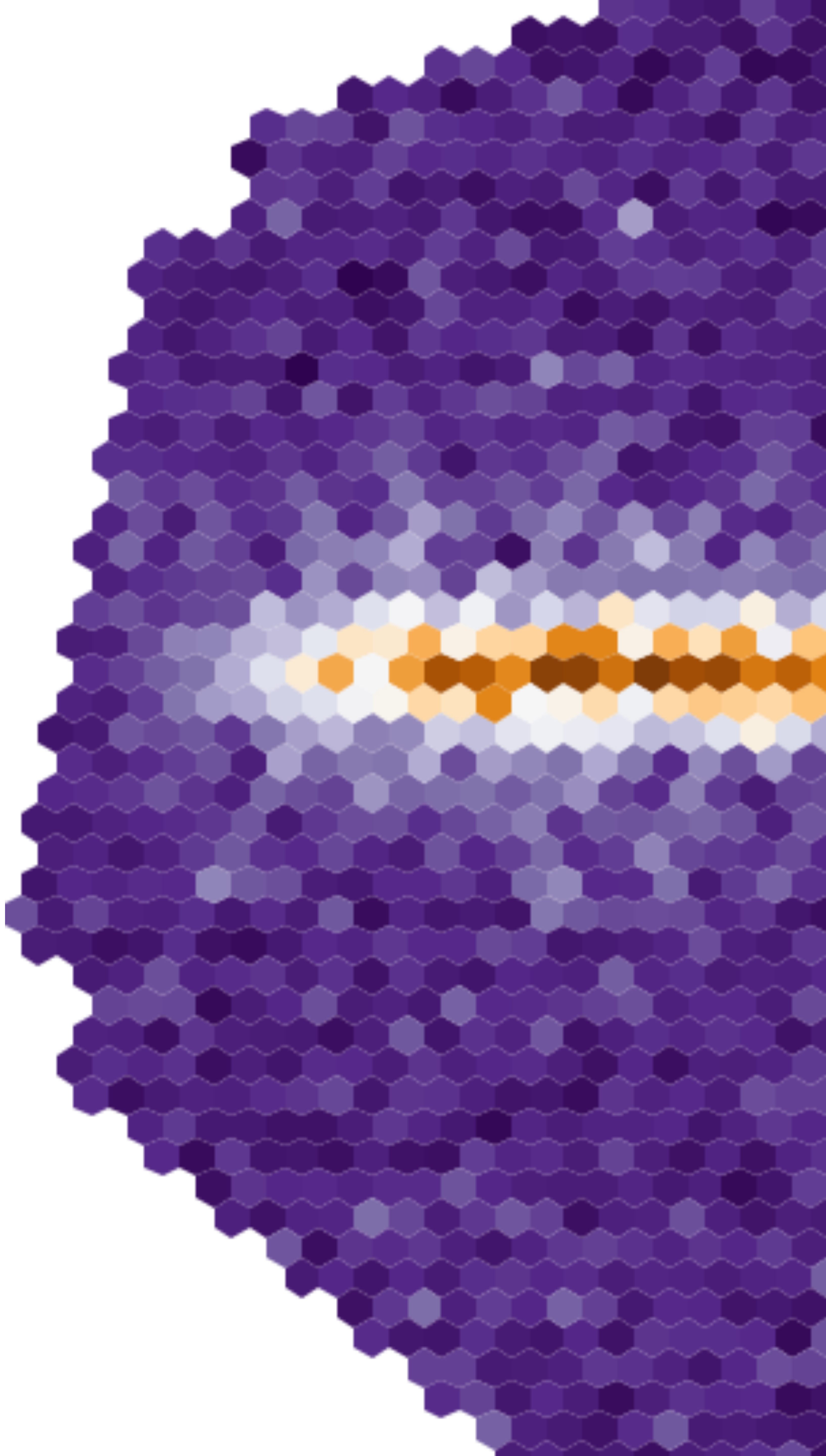
Few developers are really C++ experts and code in C++ can take many forms

Too complex / heavy a framework is a burden to developers and users.

Recommendations:

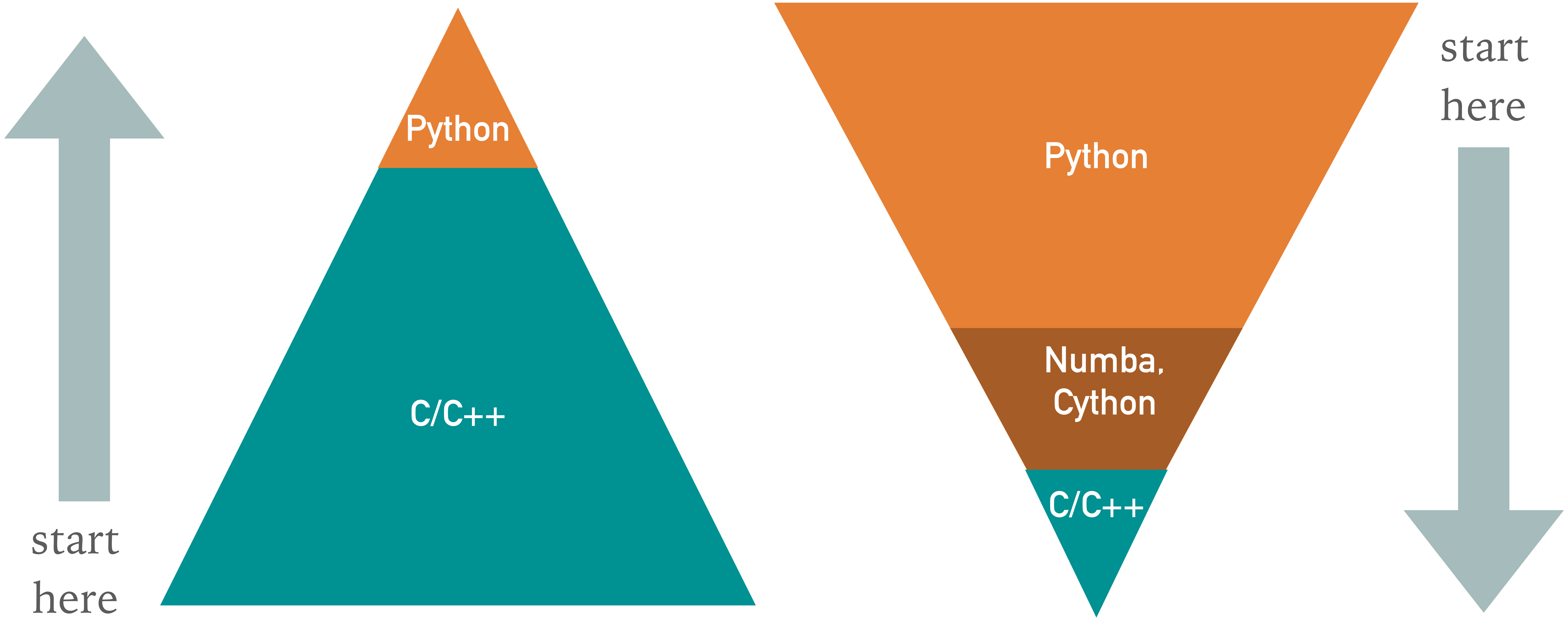
- Leverage the Astronomy community! (vs Particle Physics)
- Make it lightweight
- Make it friendly : Rich visualisations , tutorials, notebooks, easy to discover and explore
- Use standards and open tools (minimize custom code)
- Don't be too clever with how algorithms are chained together: can be confusing to users, difficult to debug, and you can achieve the same thing later by wrapping in a big-data framework (spark, celery, etc)

Implementation



Building a Framework

Bottom-Up approach | **Top-Down approach**



Most previous frameworks did it this way

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

How do we get to a final product? (Implementation Choices)

How do we get to a final product? (Implementation Choices)

Core library in Python:
a controversial choice at the time!
(a distant 3-4 years ago)

- Existence of **AstroPy** and early **GammaPy** was a major motivation, but both still <1.0 release at the time
- Momentum in **astronomy community**, but not well known in our community (astroparticle physics)
- **Bad experiences** (reportedly…) in past with python: (numeric / numarray mess, slowness, etc)
- Proof of concept was needed.

How do we get to a final product? (Implementation Choices)

Core library in Python:
a controversial choice at the time!
(a distant 3-4 years ago)

- Existence of **AstroPy** and early **GammaPy** was a major motivation, but both still <1.0 release at the time
- Momentum in **astronomy community**, but not well known in our community (astroparticle physics)
- **Bad experiences** (reportedly…) in past with python: (numeric / numarray mess, slowness, etc)
- Proof of concept was needed.

Open Source!

- Builds trust! (and better code if you worry about others seeing it…)
- Unforeseen science cases
  - low-level data will be accessible upon proposal to GOs (expect very few, but who knows?)
- Cross-over with other instruments (HESS, MAGIC, VERITAS in particular)

How do we get to a final product? (Implementation Choices)

Core library in Python:
a controversial choice at the time!
(a distant 3-4 years ago)

- Existence of **AstroPy** and early **GammaPy** was a major motivation, but both still <1.0 release at the time
- Momentum in **astronomy community**, but not well known in our community (astroparticle physics)
- **Bad experiences** (reportedly…) in past with python: (numeric / numarray mess, slowness, etc)
- Proof of concept was needed.

Open Source!

- Builds trust! (and better code if you worry about others seeing it…)
- Unforeseen science cases
  - low-level data will be accessible upon proposal to GOs (expect very few, but who knows?)
- Cross-over with other instruments (HESS, MAGIC, VERITAS in particular)

How do we get to a final product? (Implementation Choices)

Modern Collaborative Development Practices!

- GitHub, TravisCI, Codacy, coverage.io, Slack
- Require 2 code reviews before merging a PR (and no commits to master!)
- 35 committers so far
  - ≈10 with large contributions),
  - many just helping write good code and docs!

Future Path to Higher-Performance!

See talk by Florian Gaté on Thursday

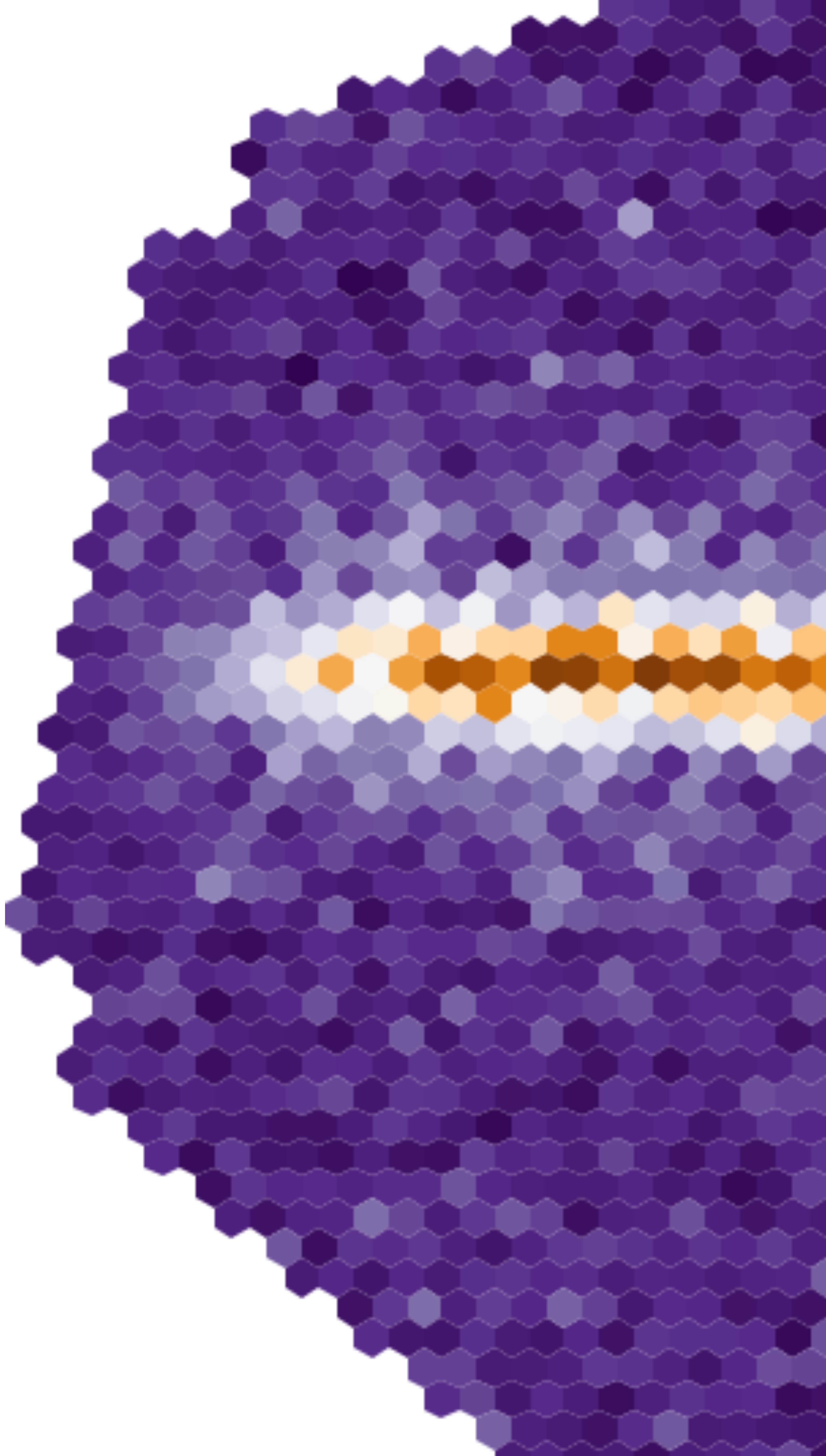
- HPC re-implementations of algorithms, cross-checked with “standard” python implementations via automated tests
  - Physicists → write python
  - Computer Scientists → Adapt it to HPC or wrap it in Big-Data frameworks
- to fancier parallelization systems:
  - Physicists → write algorithms
  - Experts → Wrap them to run in “Big Data” frameworks

common “core” package → full prototype

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

github.com/cta-observatory/ctapipe

Algorithms and Workflow



Data Processing Pipeline (simplified)

K. Kosack, PyGamma19

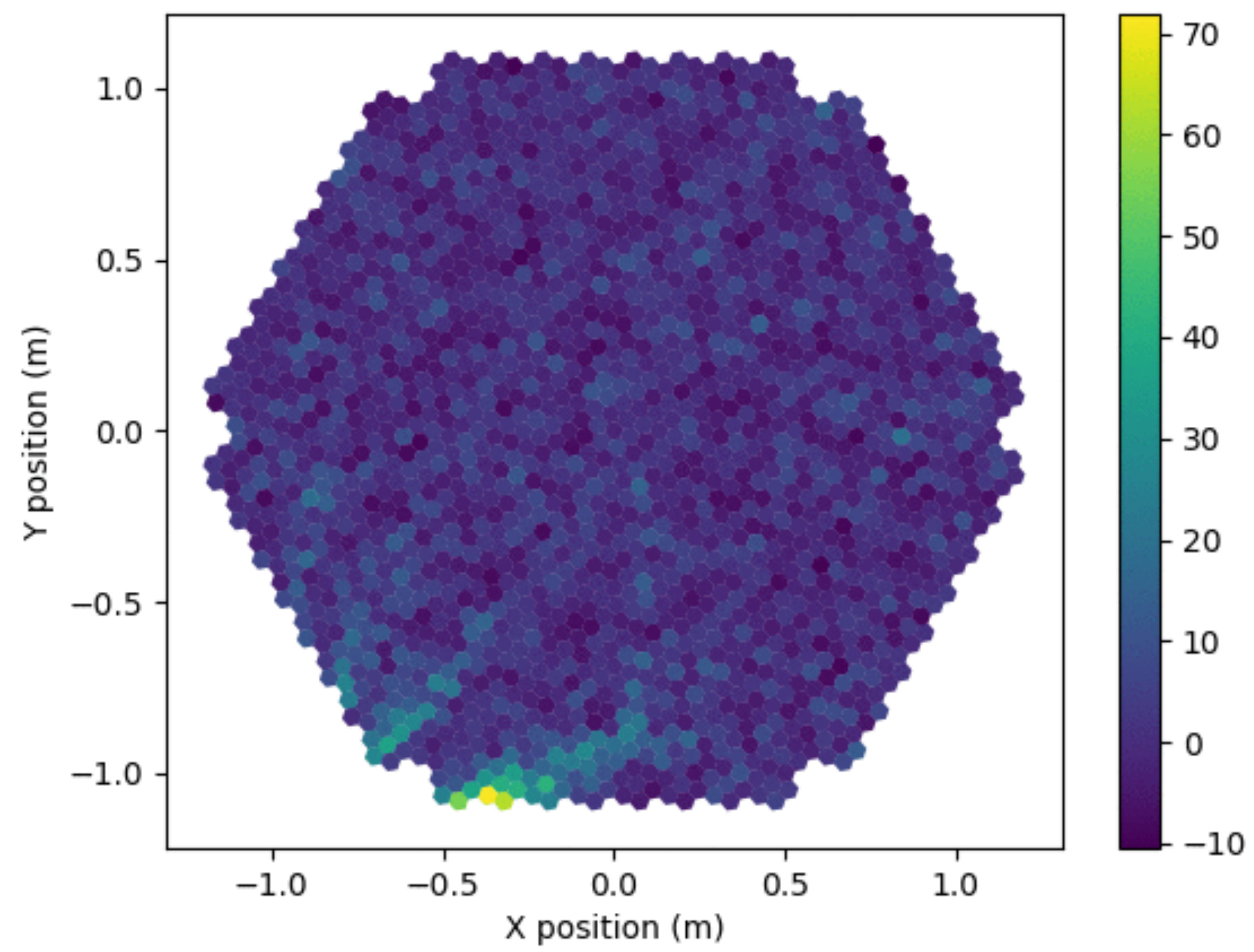
Data Processing Pipeline (simplified)

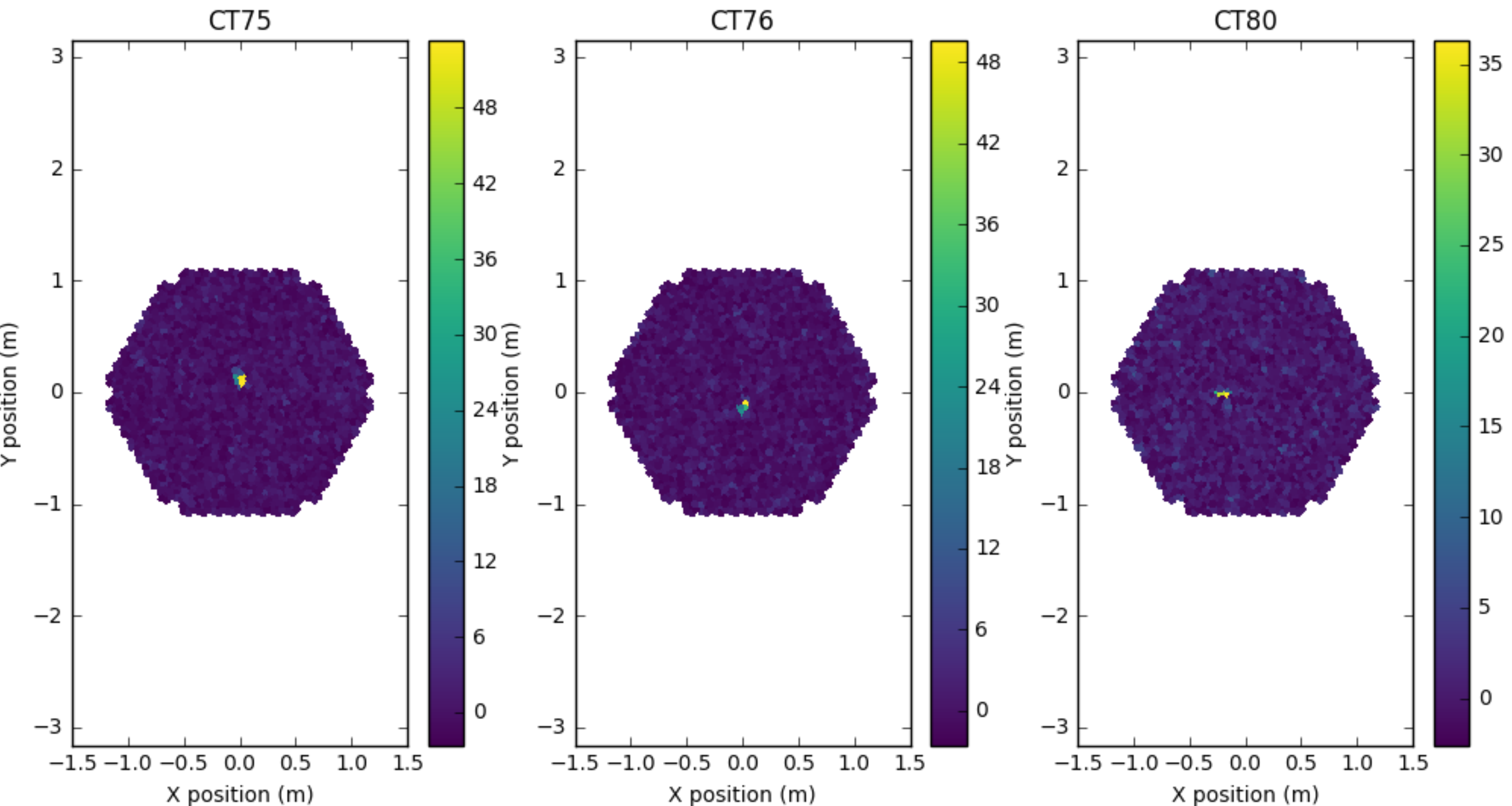
Stage 1: Per-telescope image processing

Stage 1: Per-telescope image processing

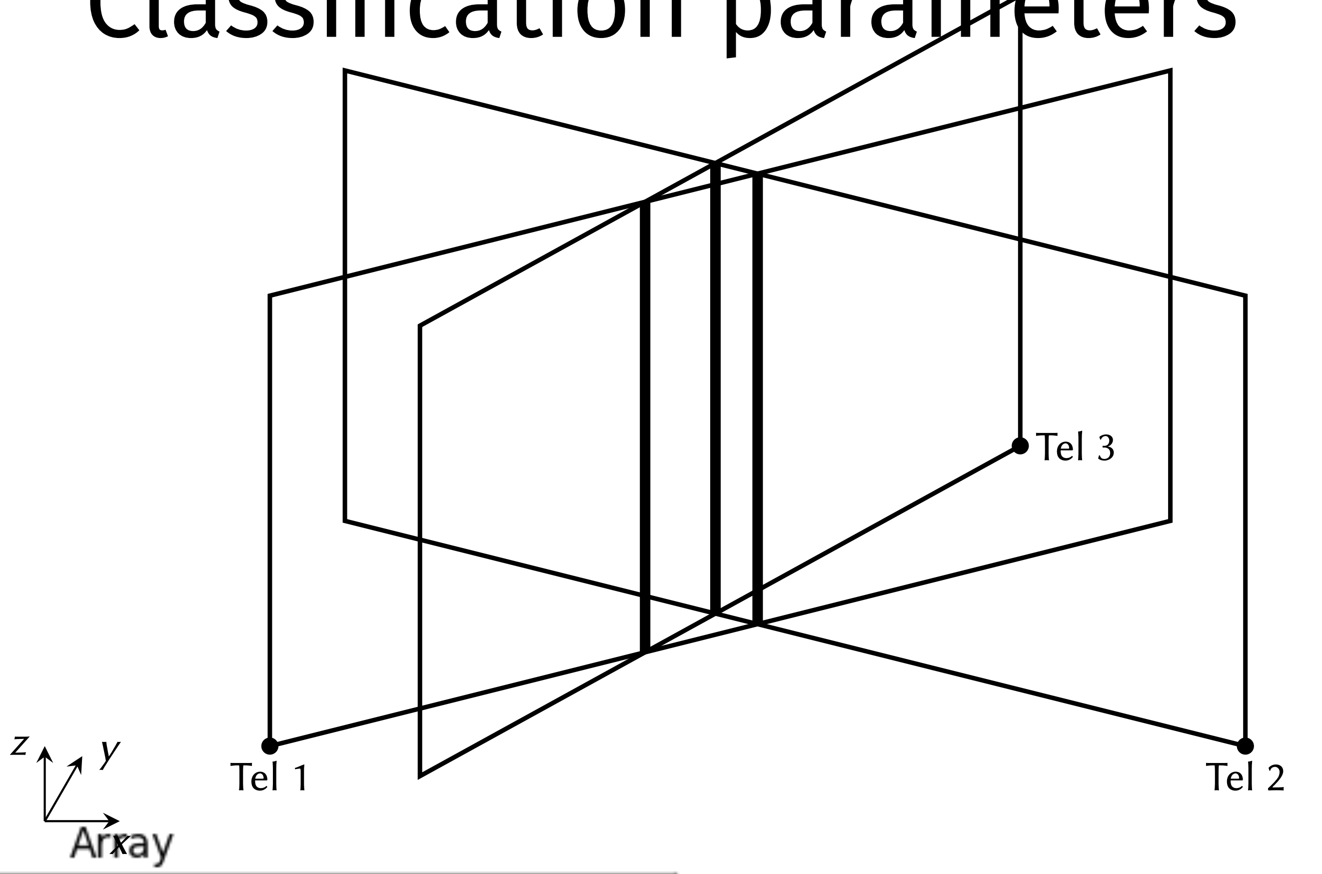
Stage 2: Reconstruction

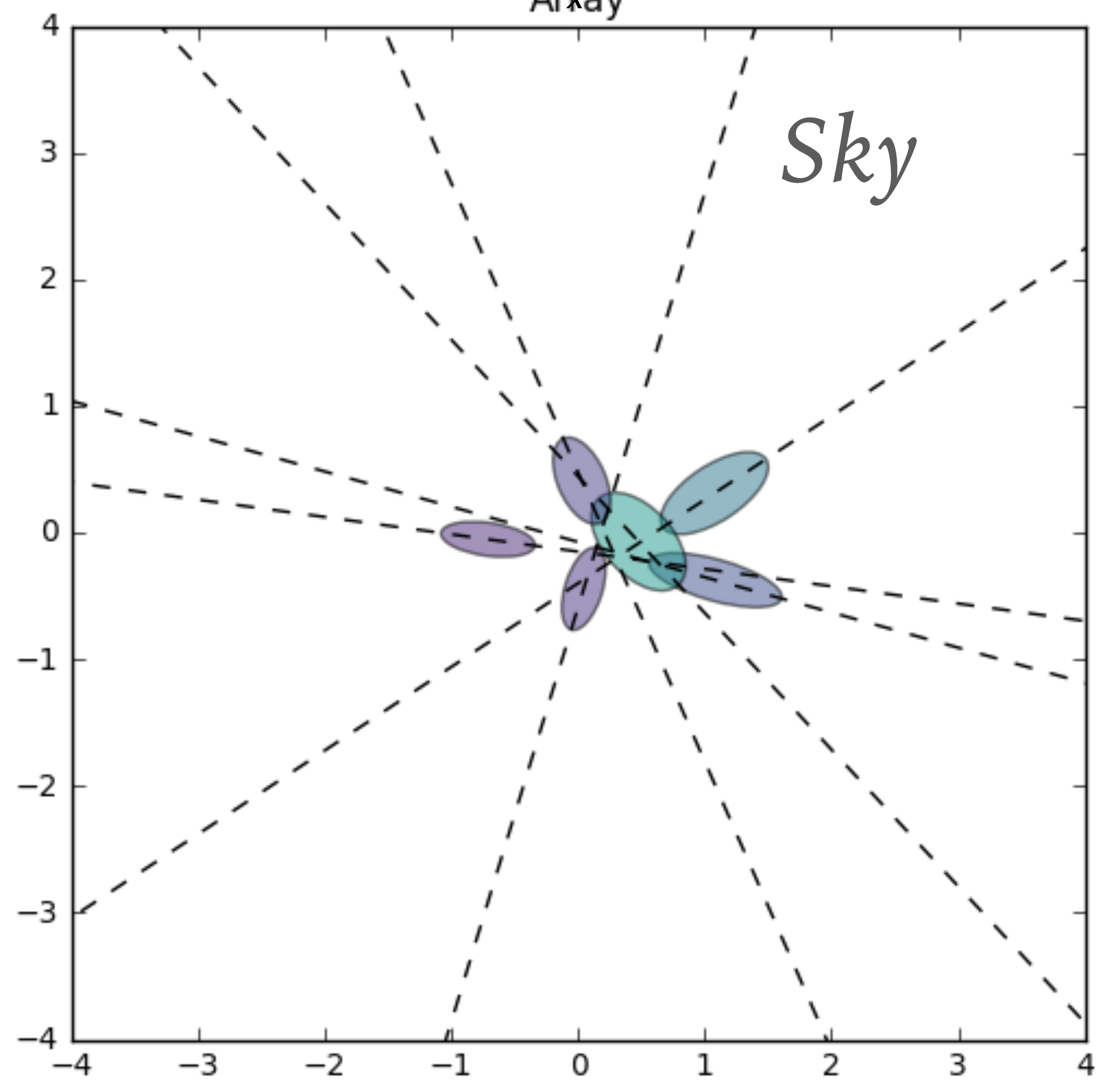
(All tels overlaid)

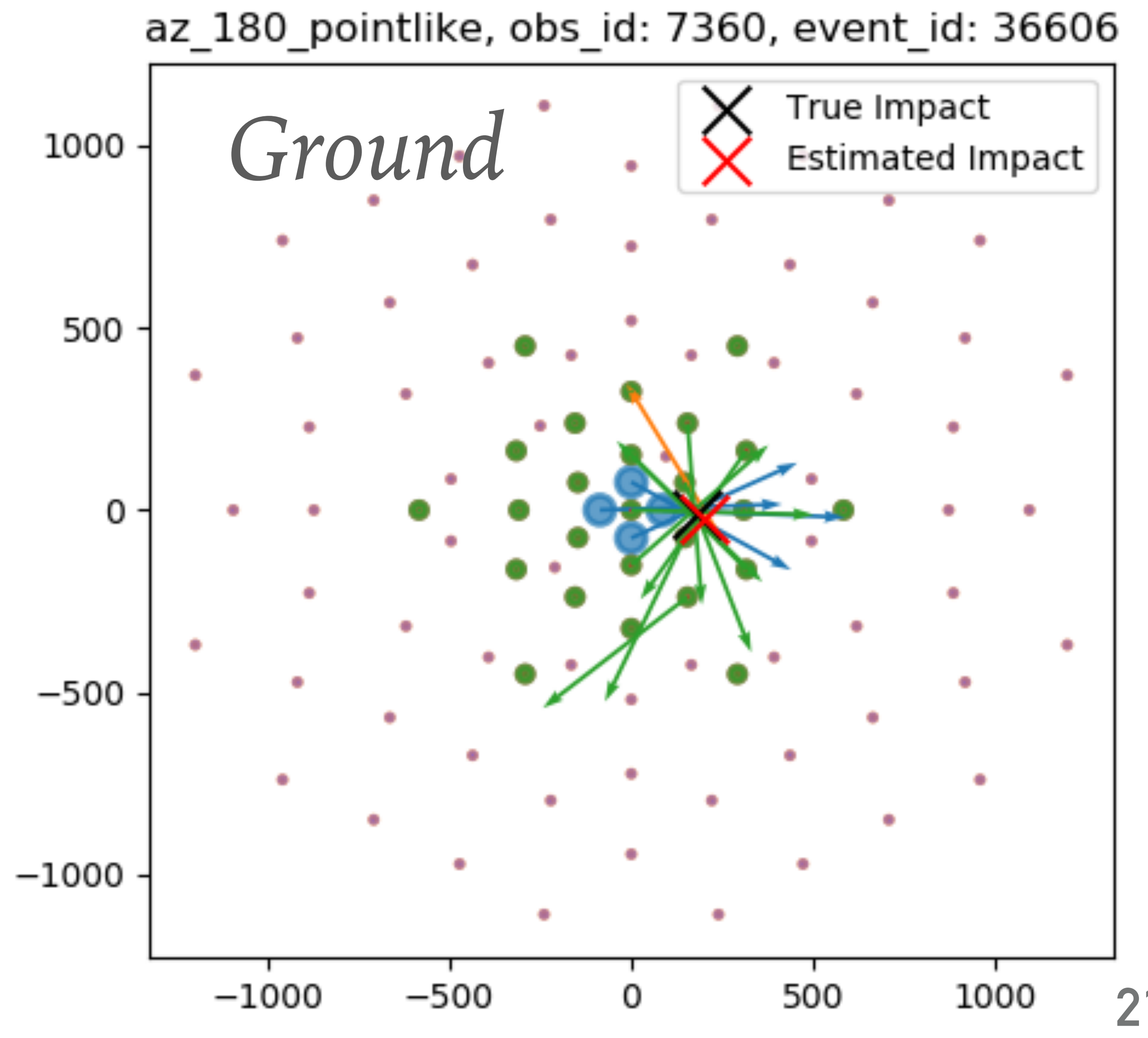




Outputs are: **Point-of-Origin** on **sky** and **ground**+ **Energy** + Classification parameters

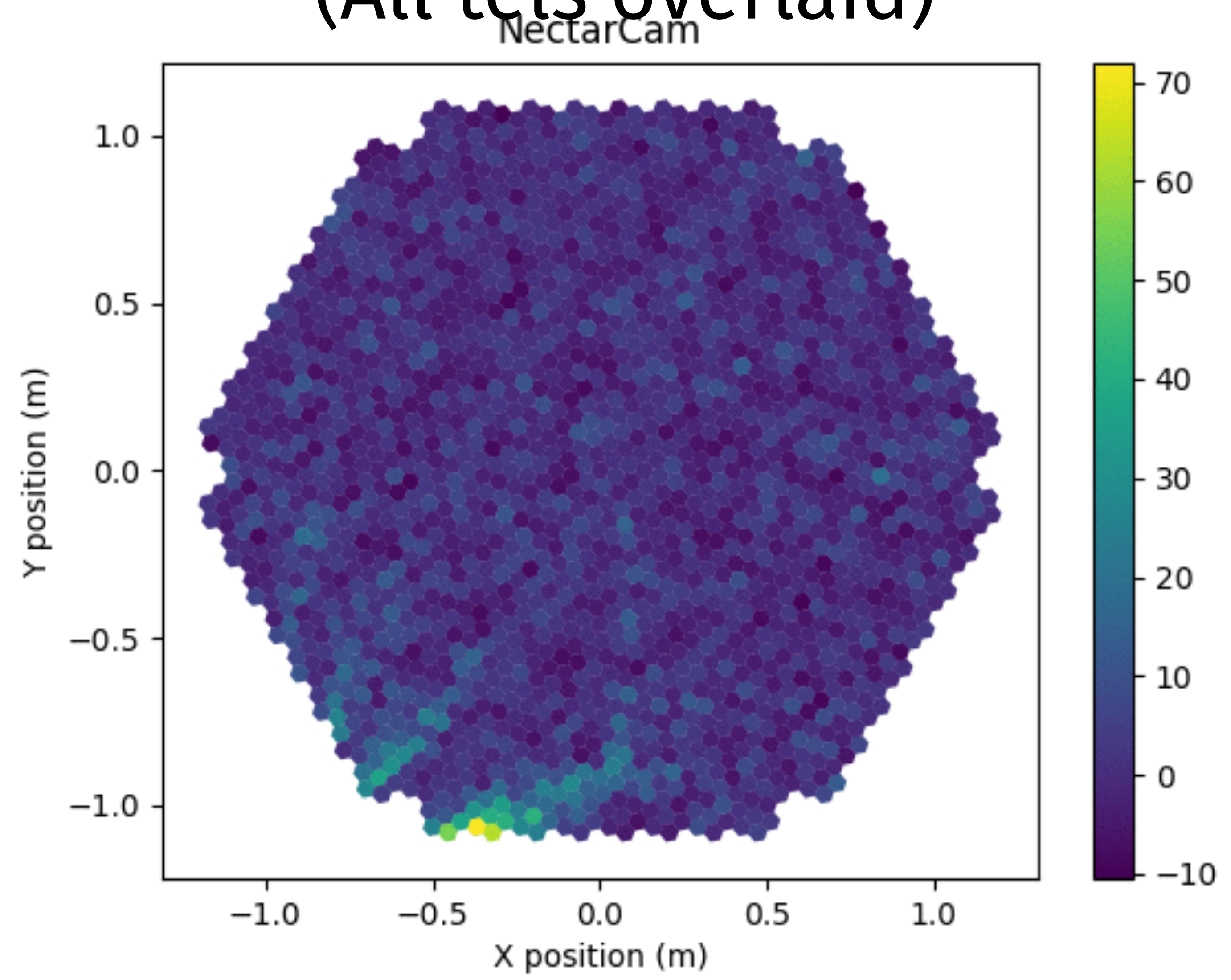


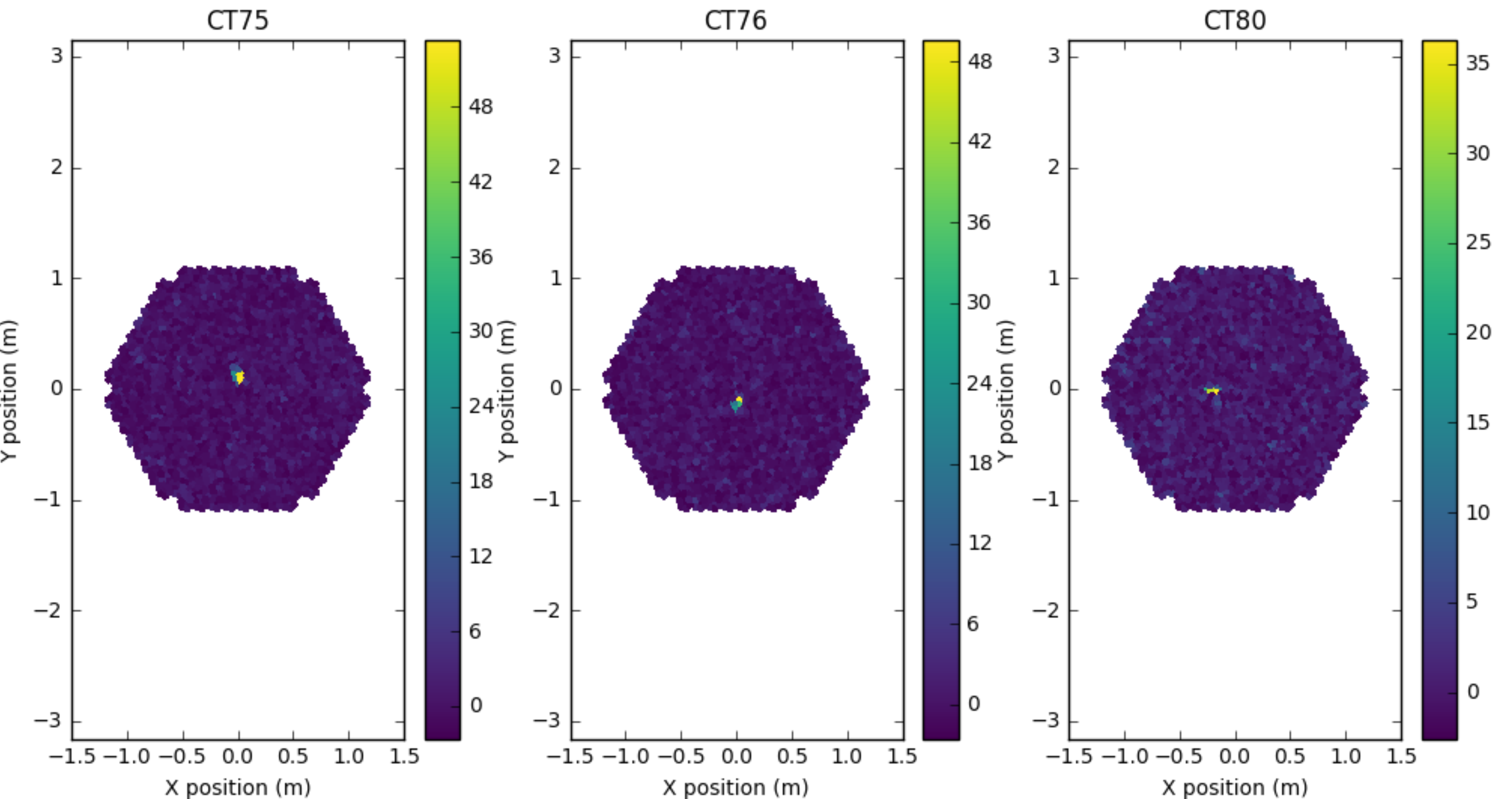




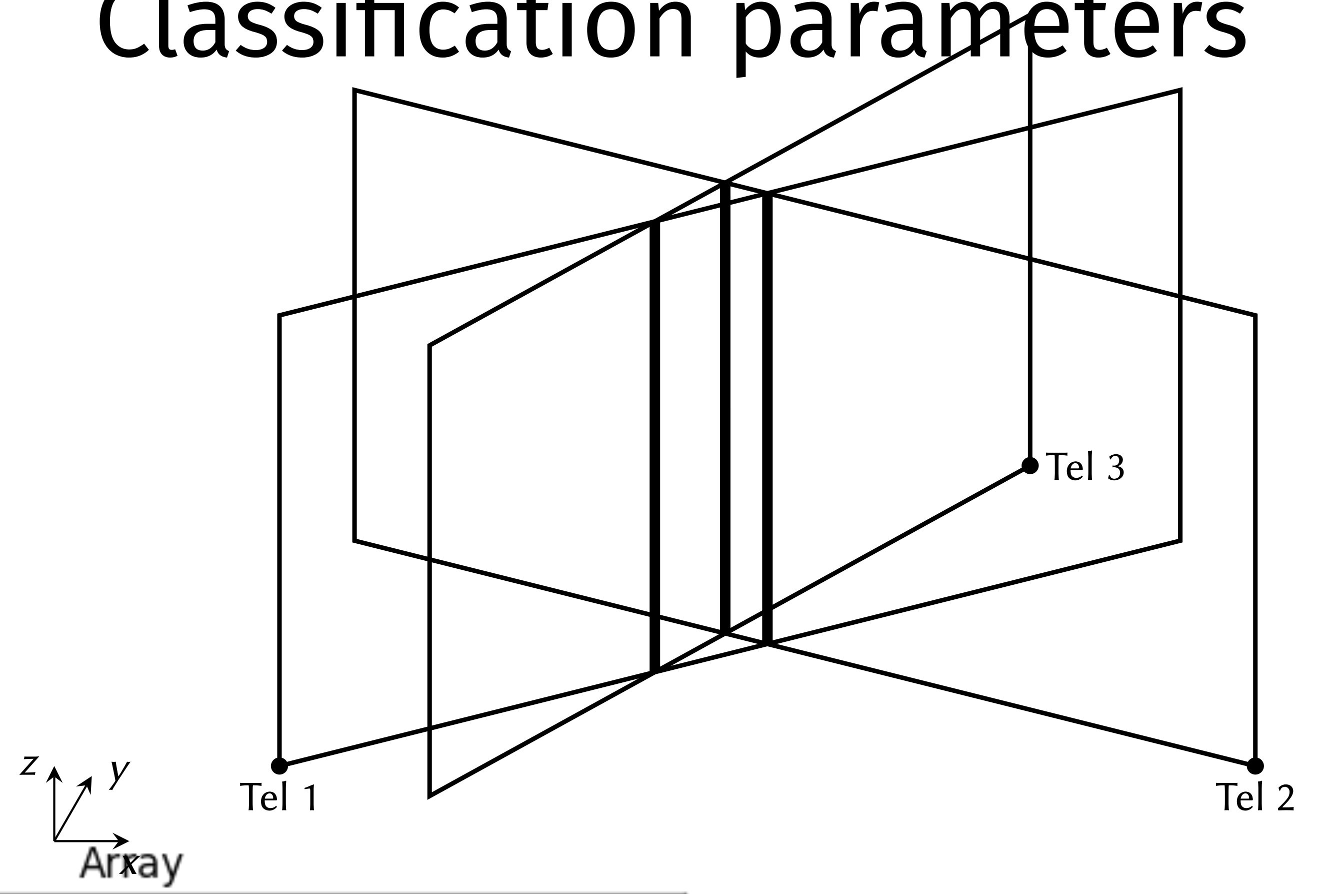
Stage 2: Reconstruction

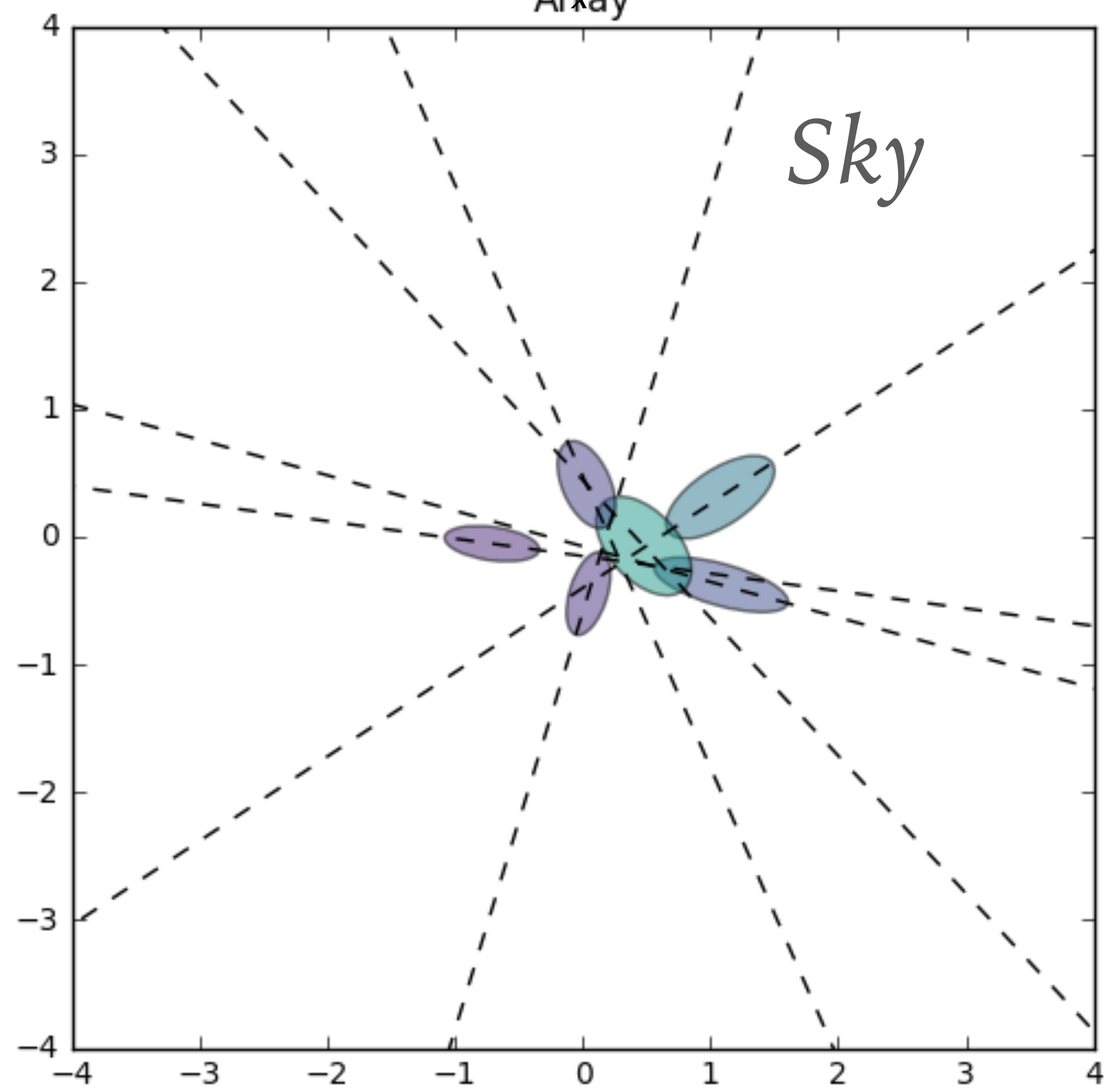
(All tels overlaid)

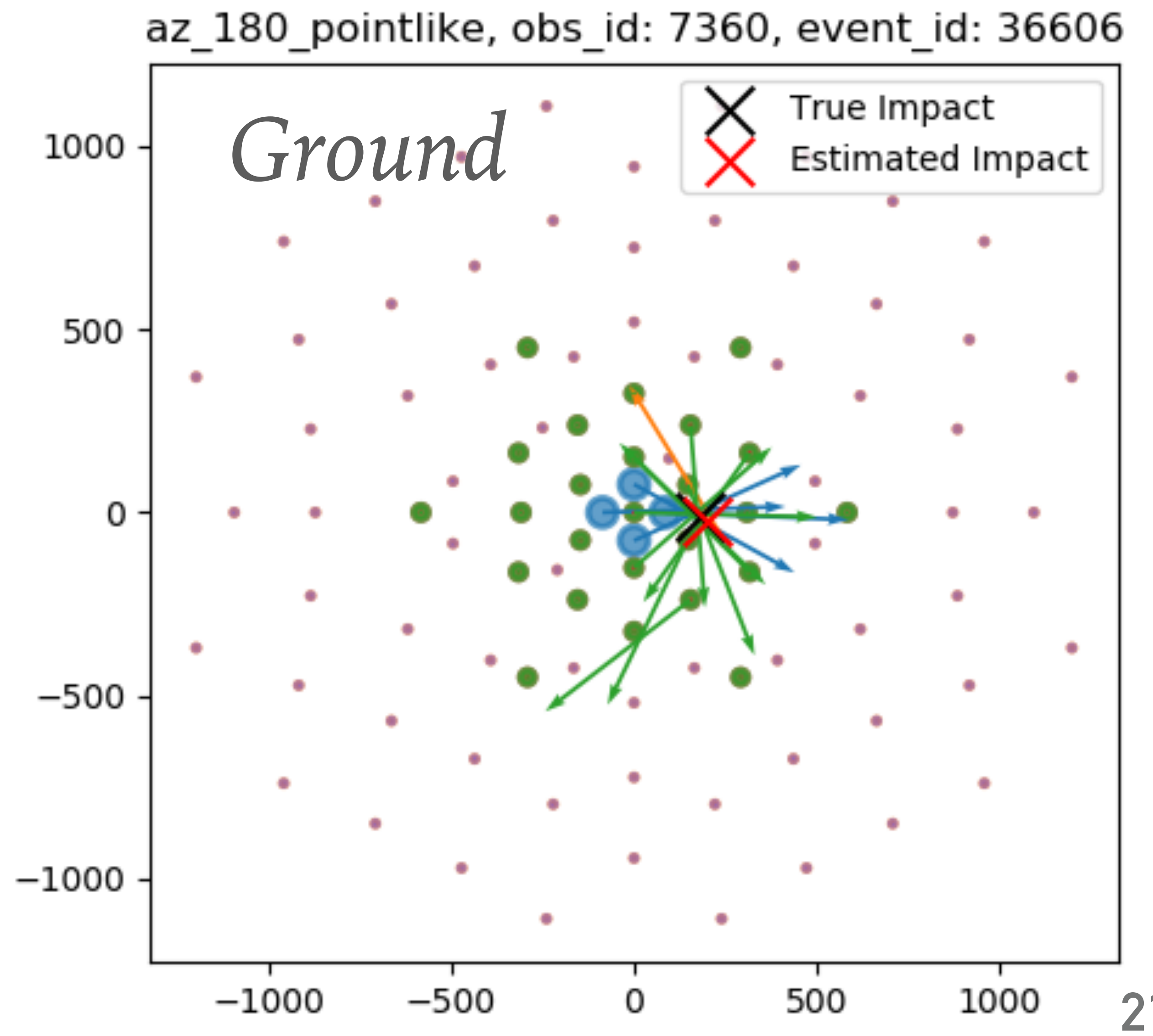




Outputs are: **Point-of-Origin** on **sky** and **ground**+ **Energy** + Classification parameters

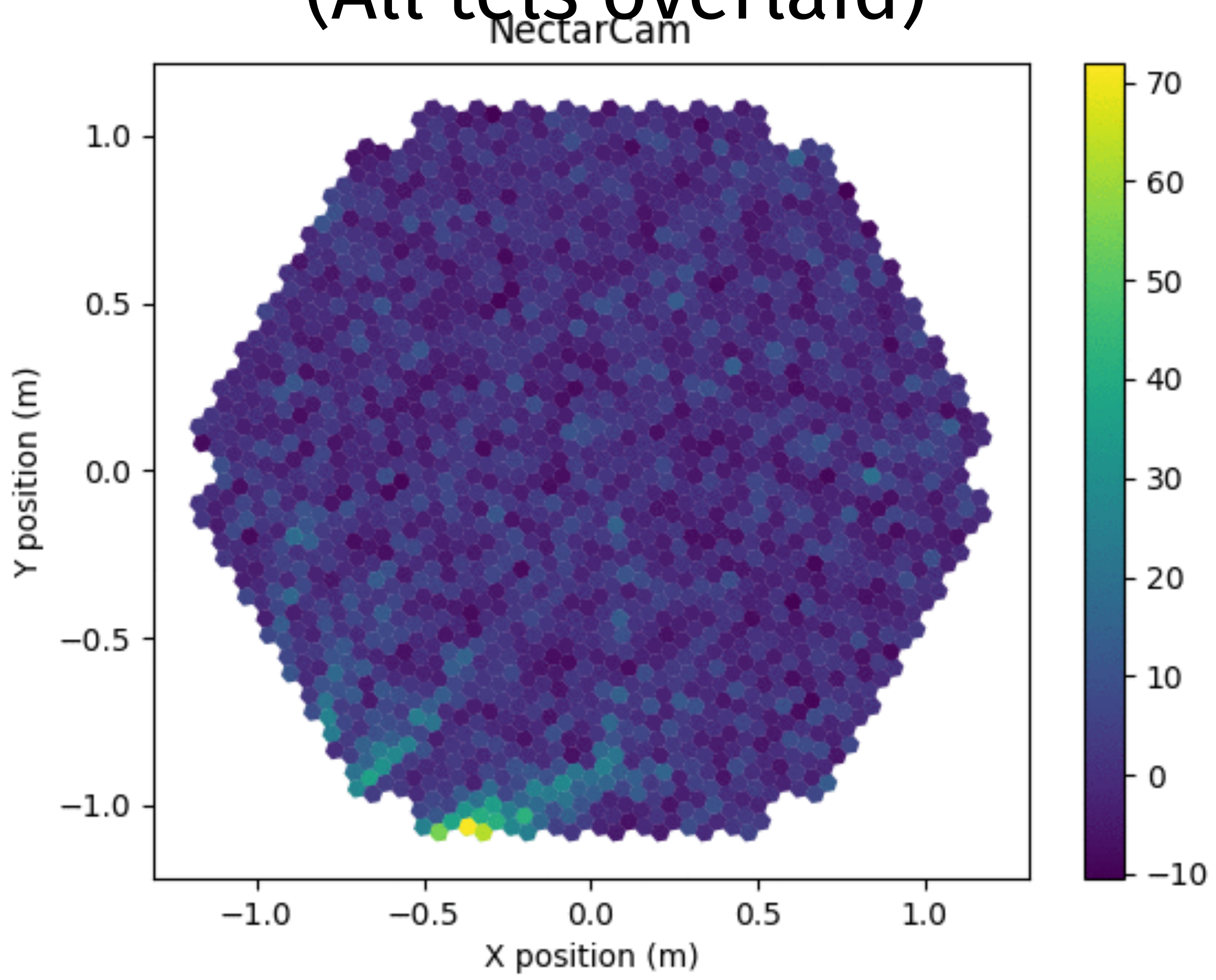






Stage 2: Reconstruction

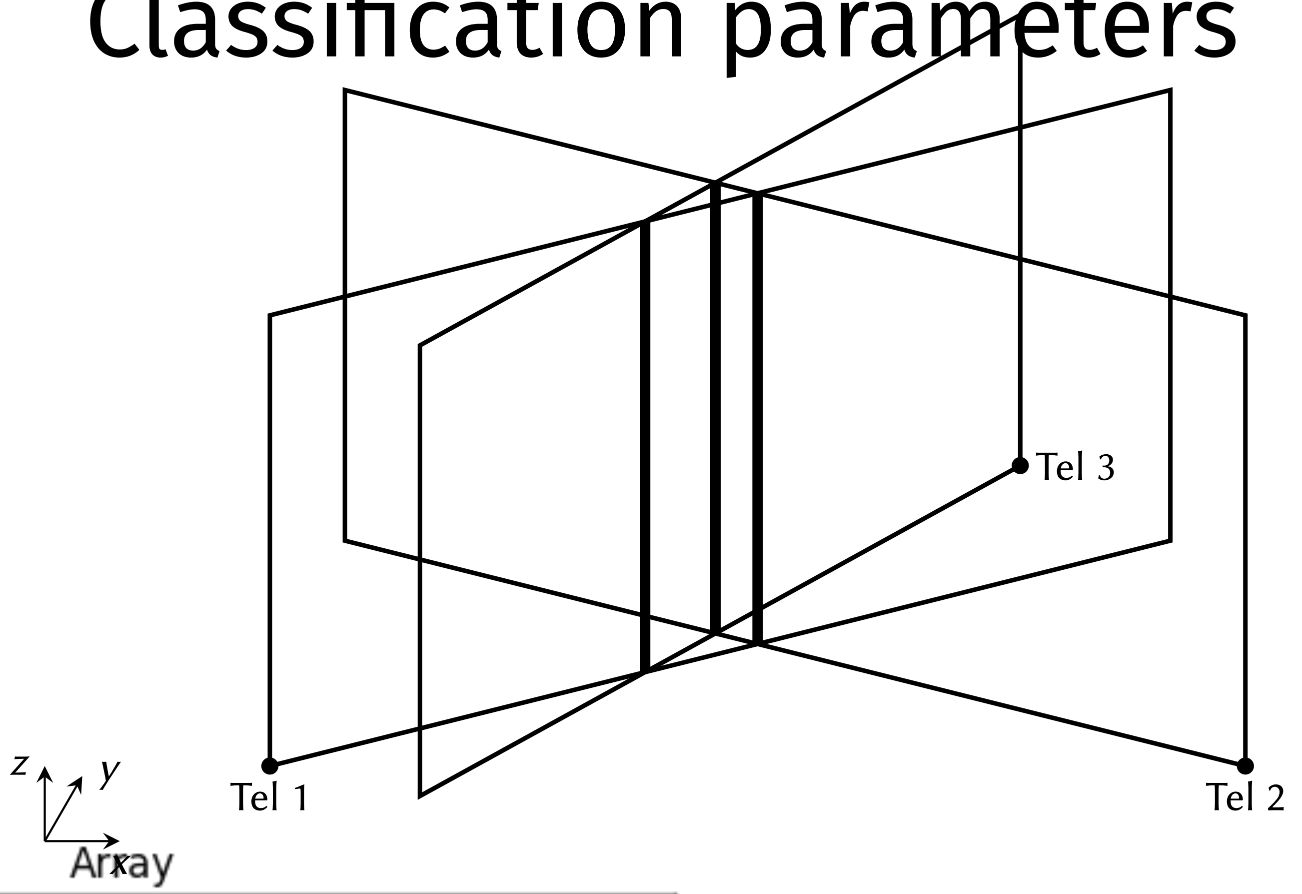
(All tels overlaid)

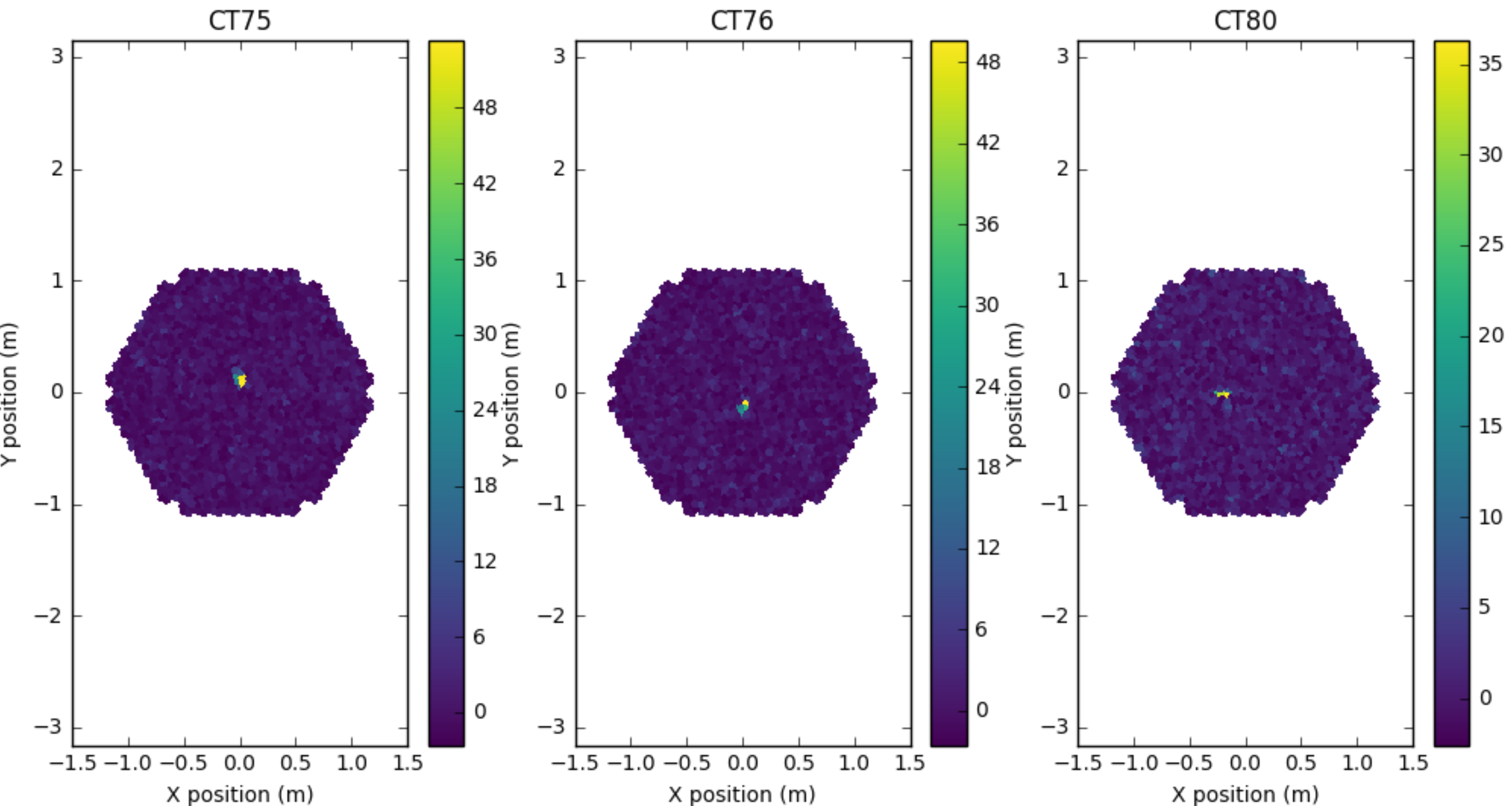


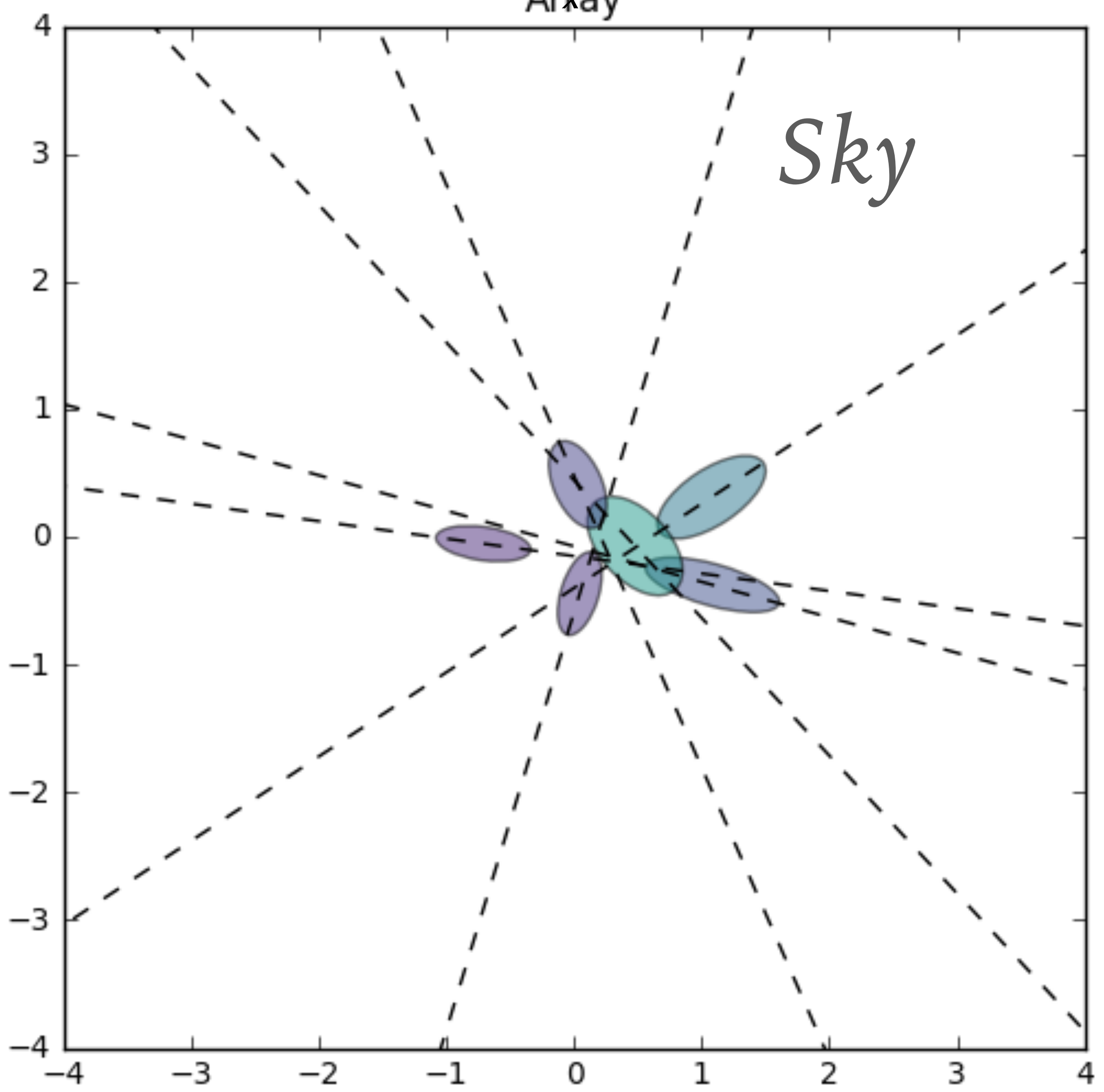
Note:
More advanced techniques exist and are being implemented

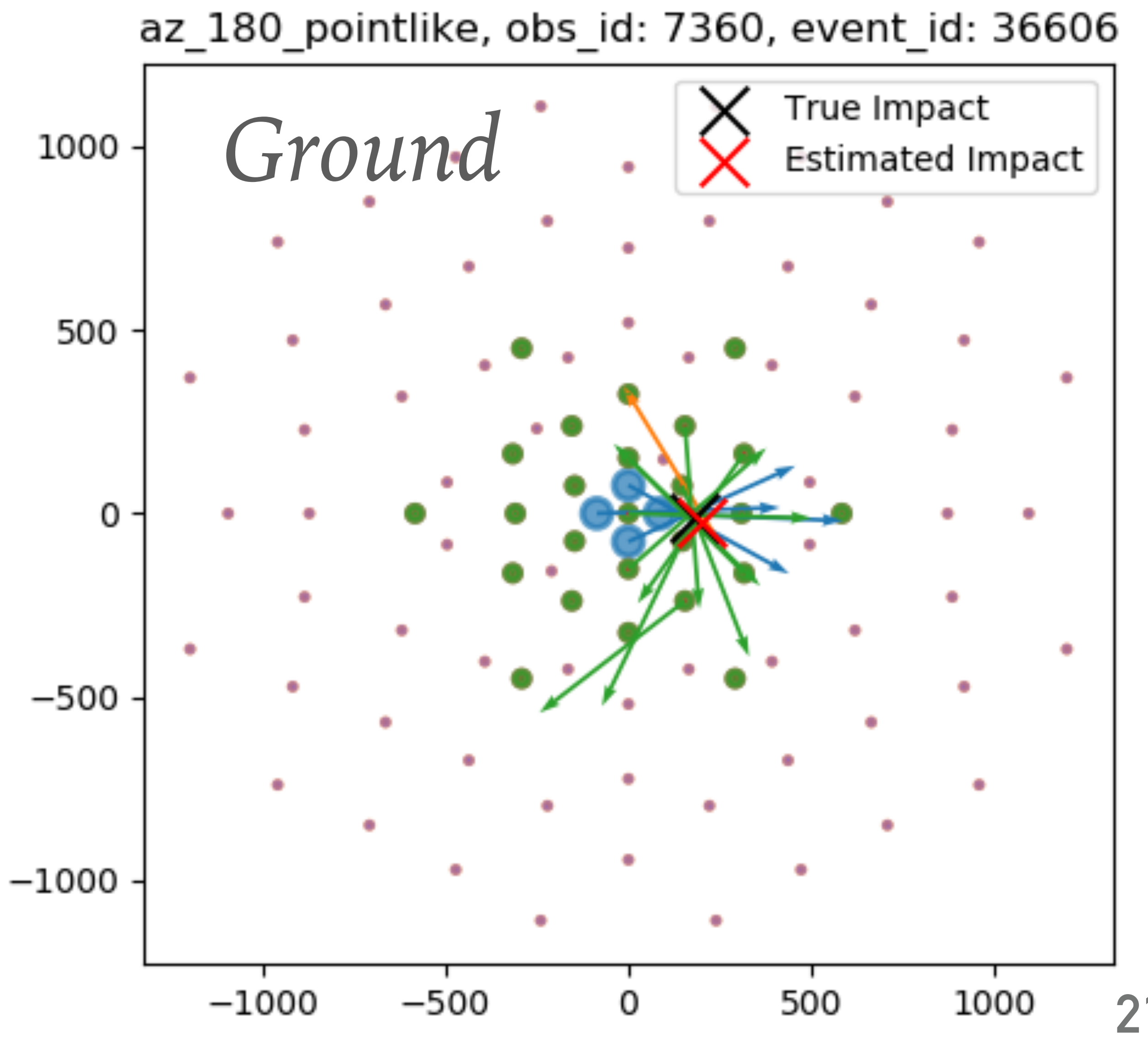
(generally with higher CPU requirements and data needs)

Outputs are: **Point-of-Origin** on **sky** and **ground**+ **Energy** + Classification parameters



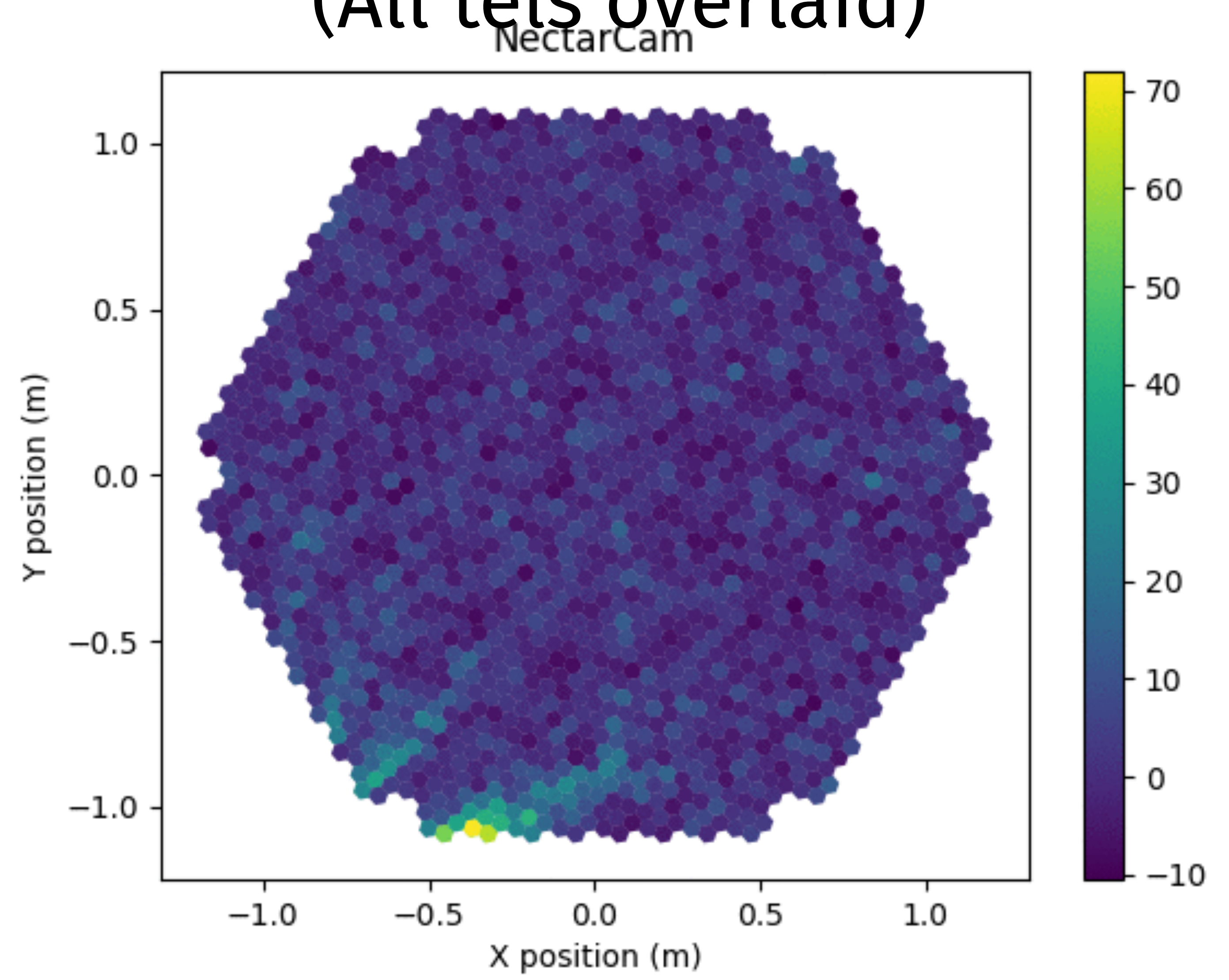






Stage 2: Reconstruction

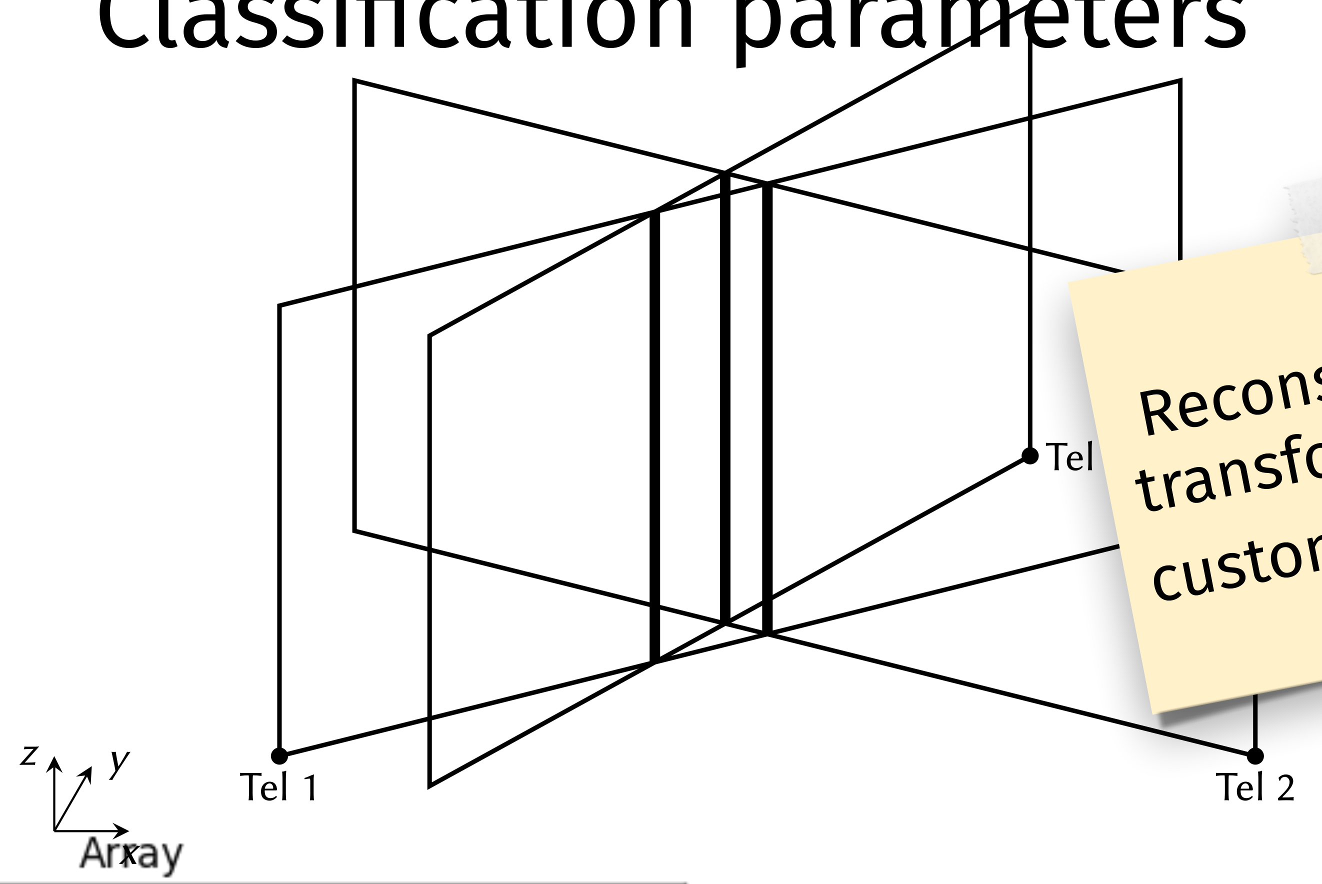
(All tels overlaid)



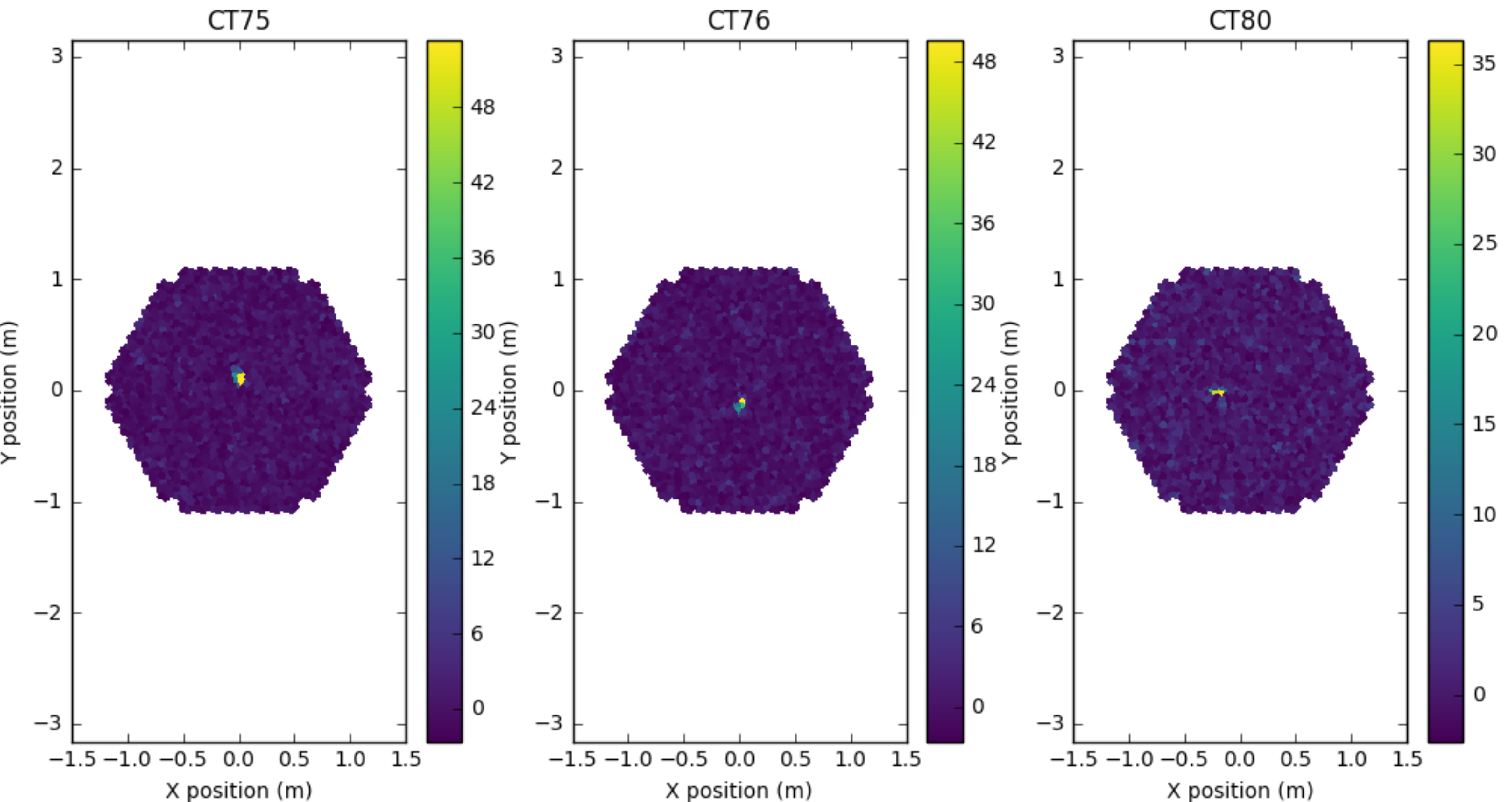
Note:
More advanced techniques exist and are being implemented

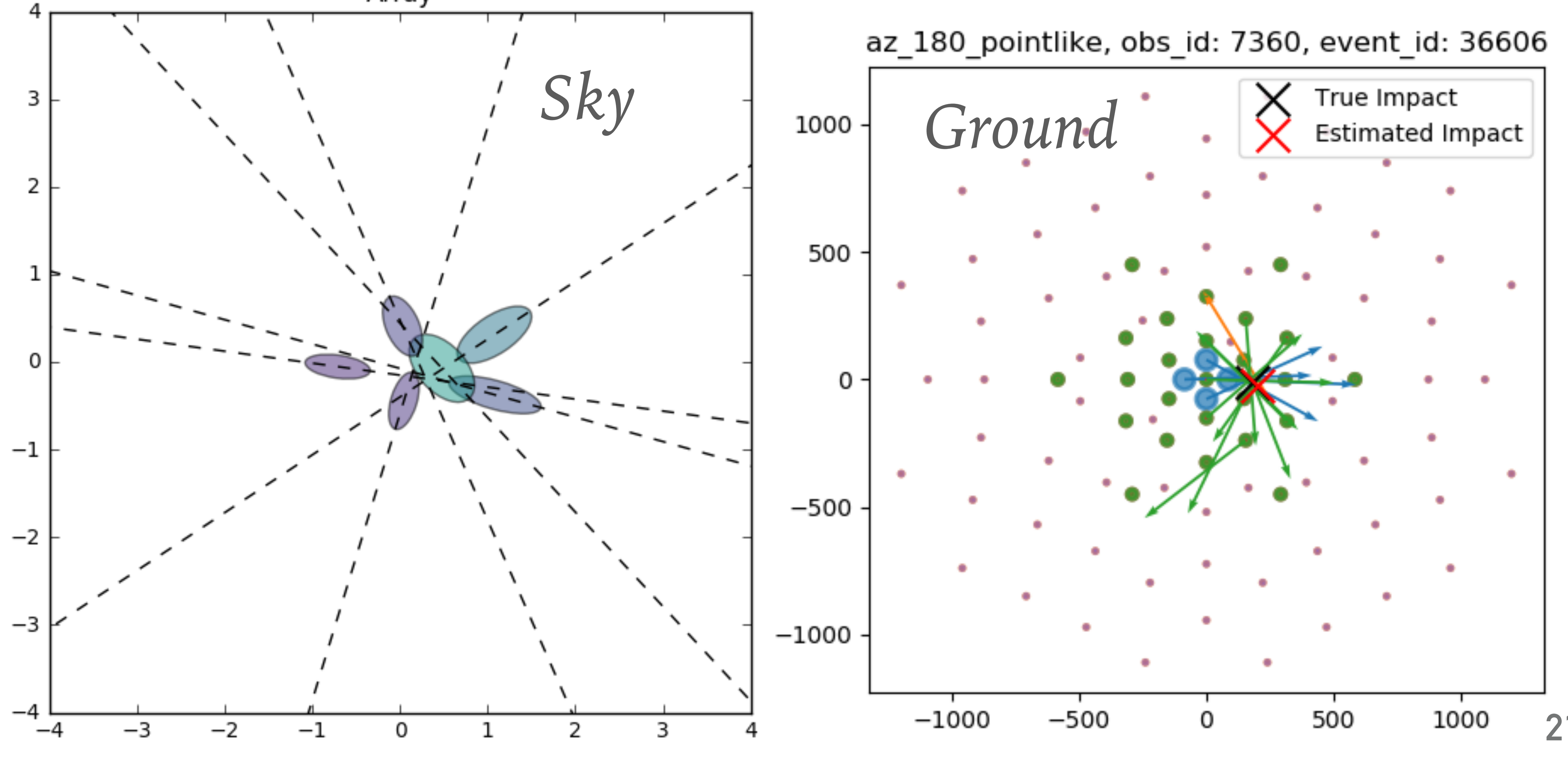
(generally with higher CPU requirements and data needs)

Outputs are: **Point-of-Origin** on **sky** and **ground**+ **Energy** + Classification parameters



Note:
Reconstruction coordinate transforms implemented as custom **astropy.coordinates** Frames





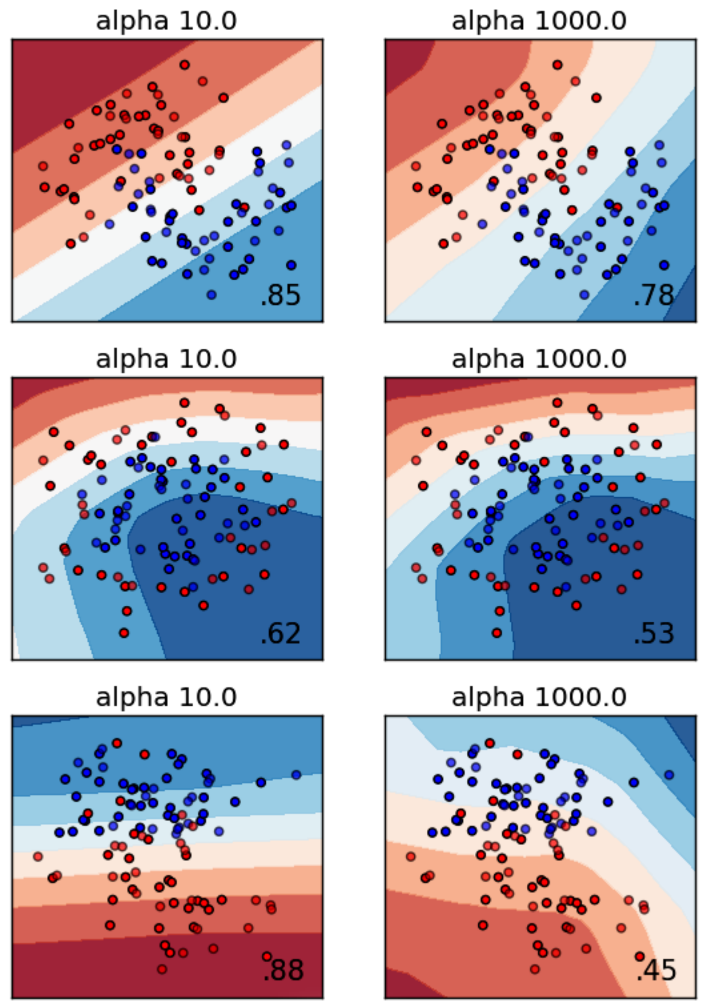
Stage 3: Discrimination

Note: Same technique for Energy reconstruction and Event Type Classification

per-image parameter sets (width, length, ...)

per-shower parameters (impact distance, energy, number of tels, ...)

parameters



scikits learn machine learning in Python

particle class

gamma-like

hadron-like

electron-like

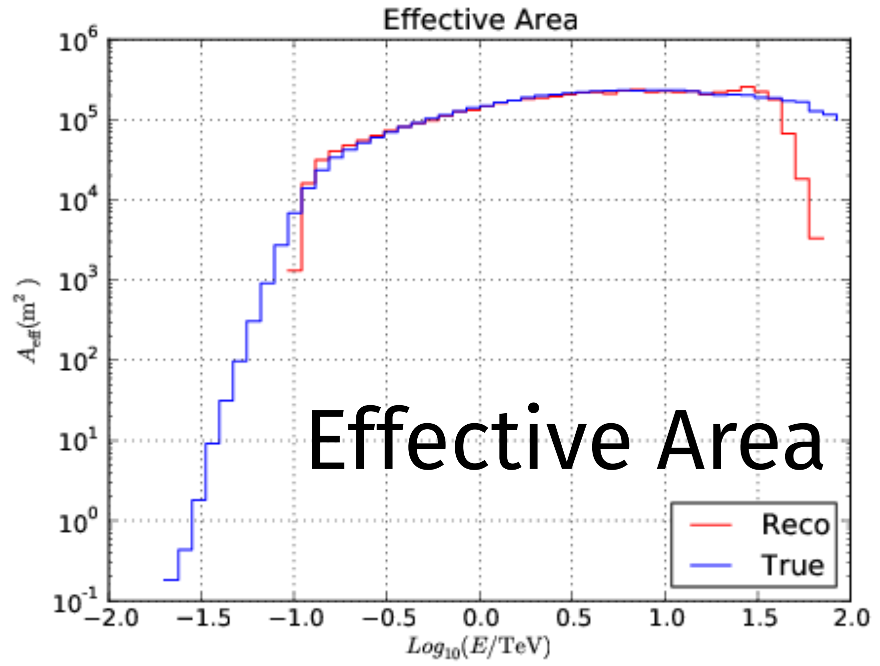
Note: Also event quality classification, e.g. good PSF, good spectral resolution, sensitivity to unknown sources, ...

Output: Science Data

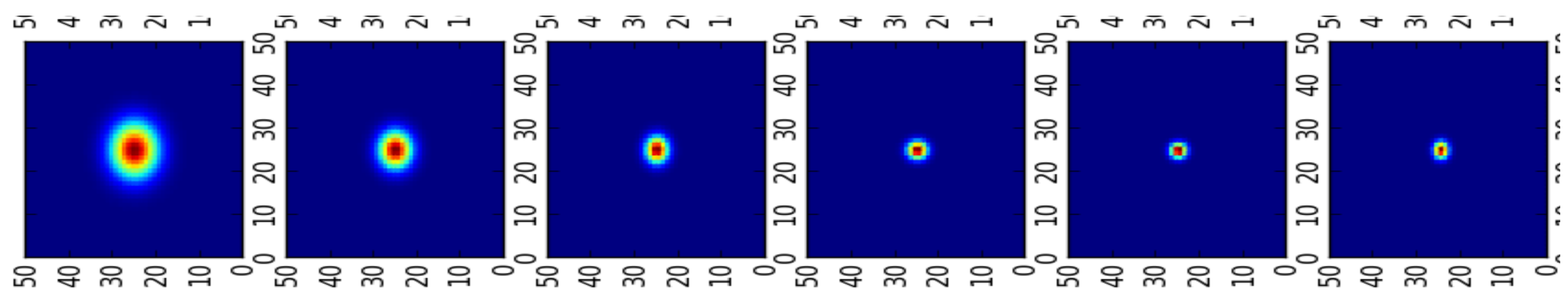
Event-List

event_id	RA	DEC	E	class	type	n_tels	...
1	23,3	-40,1	0,01			5	
2	24,6	-40,5	20,0			34	
3	23,5	-41,12	0,45			3	
4	21,3	-38,2	1,03			4	

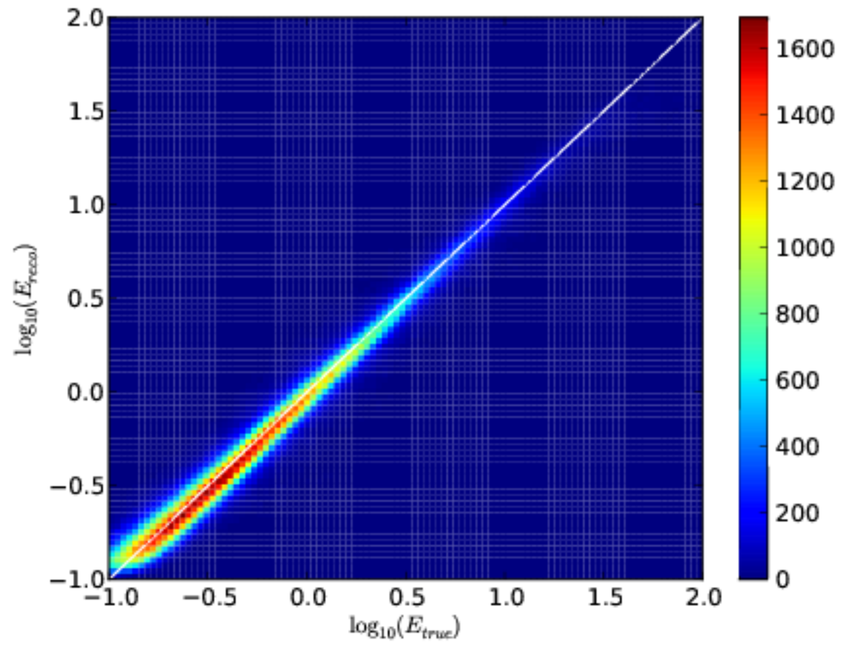
Instrumental Responses:

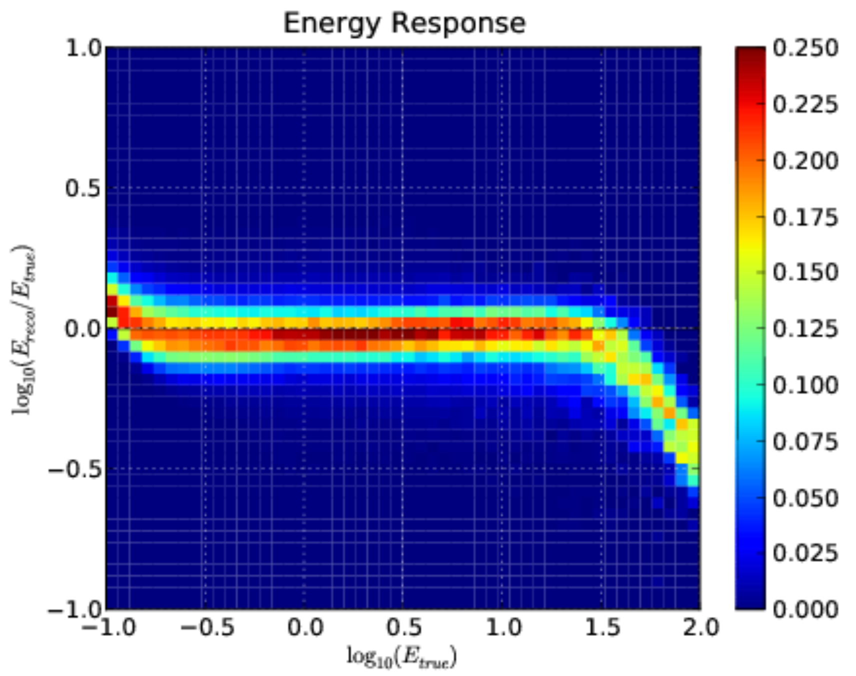


PSF



Energy Migration



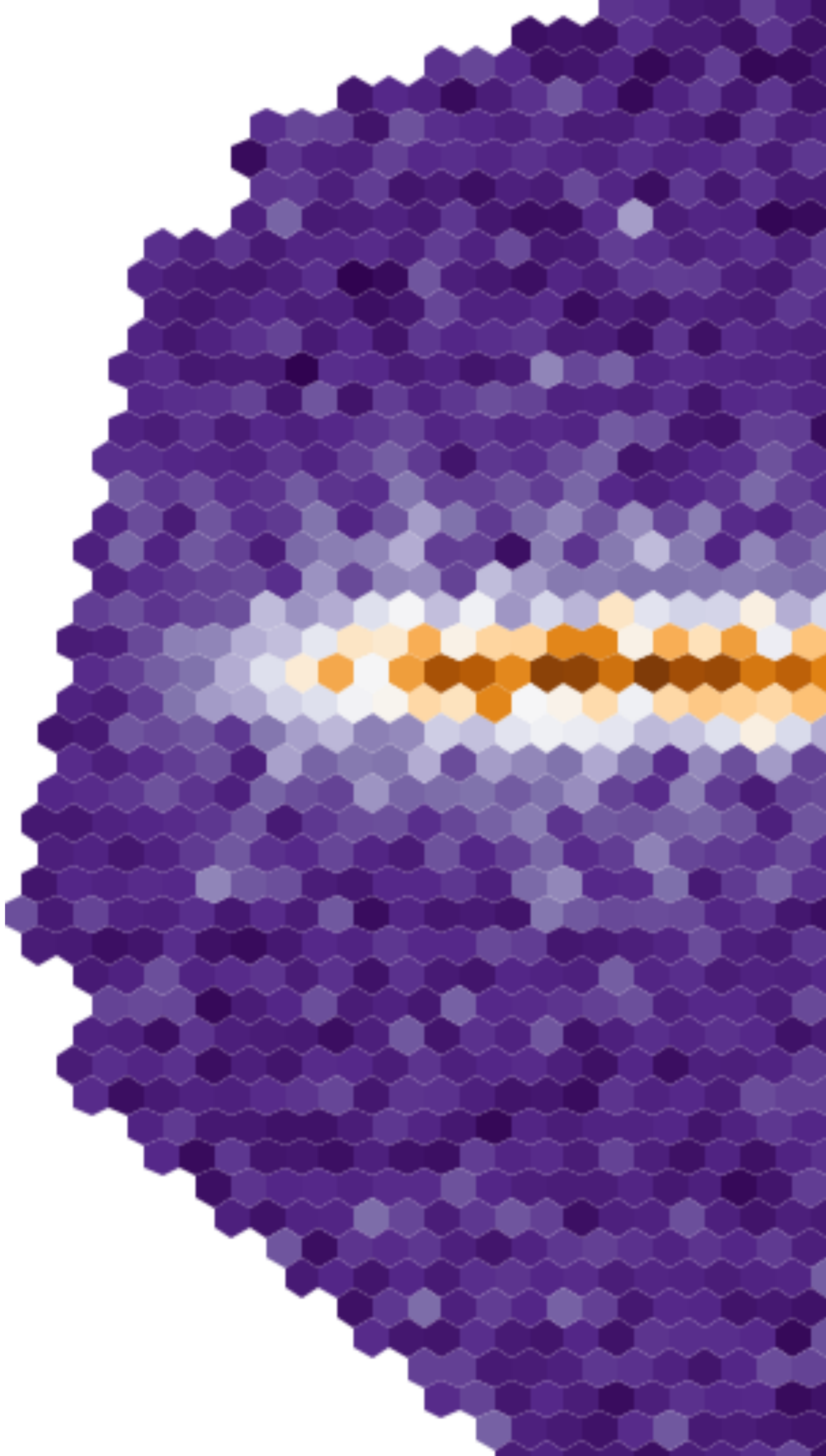


(note these are not CTA responses, just examples form HESS)

Technical Tables (for sub-GTIs)

TIME	Transparency	Temperature	Trigger Rate
580234.34	0.8	32	12034
580234.35	0.94	32	13023
580234.36	0.70	33	12532

A few framework features



Raw (event) Data Access Layer

Factory pattern used to choose implementation based on input

In the future:

Standard CTA raw data format (TBD)

Simple Event-wise Data Access

Working with data is supposed to be simple:

```
from ctapipe.io import event_source

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

for event in source:
   print(event.trig.tels_with_trigger)
   print(event.trig.gps_time.iso)
   print(event.trig.gps_time.mjd)
   print(event.mc.energy.to('GeV'))
   print(event.r0.tel[4].waveform.mean())
```

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

Simple Event-wise Data Access

Working with data is supposed to be simple:

an instance of **EventSource** based on file contents

```
from ctapipe.io import event_source

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

for event in source:
   print(event.trig.tels_with_trigger)
   print(event.trig.gps_time.iso)
   print(event.trig.gps_time.mjd)
   print(event.mc.energy.to('GeV'))
   print(event.r0.tel[4].waveform.mean())
```

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

Simple Event-wise Data Access

Working with data is supposed to be simple:

```
from ctapipe.io import event_source

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

for event in source:
   print(event.trig.tels_with_trigger)
   print(event.trig.gps_time.iso)
   print(event.trig.gps_time.mjd)
   print(event.mc.energy.to('GeV'))
   print(event.r0.tel[4].waveform.mean())
```

an instance of **EventSource** based on file contents

set of hierarchical **Containers** for various data items (also stores "column" metdata like units and descriptions)

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

Simple Event-wise Data Access

Working with data is supposed to be simple:

```
from ctapipe.io import event_source

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

for event in source:
   print(event.trig.tels_with_trigger)
   print(event.trig.gps_time.iso)
   print(event.trig.gps_time.mjd)
   print(event.mc.energy.to('GeV'))
   print(event.r0.tel[4].waveform.mean())
```

an instance of **EventSource** based on file contents

set of hierarchical **Containers** for various data items (also stores “column” metdata like units and descriptions)

rich conversions based on ***astropy*** time, units, angles

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

Simple Event-wise Data Access

Working with data is supposed to be simple:

```
from ctapipe.io import event_source

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

for event in source:
   print(event.trig.tels_with_trigger)
   print(event.trig.gps_time.iso)
   print(event.trig.gps_time.mjd)
   print(event.mc.energy.to('GeV'))
   print(event.r0.tel[4].waveform.mean())
```

an instance of **EventSource** based on file contents

set of hierarchical **Containers** for various data items (also stores “column” metdata like units and descriptions)

rich conversions based on ***astropy*** time, units, angles

All images and waveform cubes are NumPy **NDArrays**

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

Containers (fancy dict-like classes)

Used for data interchabnge between algoriuthms

Works as an object-relational mapper (ORM) for I/O

```
class MyContainer(Container):
    energy = Field(0.0, "reconstructed energy", unit=u.TeV)
    ra = Field(0.0, "right ascension", unit=u.deg, ucd='pos.eq.ra')

c = MyContainer(energy=12*u.TeV, ra=15.0*u.deg)
c.ra = 17*u.deg
c
```

```
MyContainer:
                energy: reconstructed energy [TeV]
                    ra: right ascension [deg]

In [10]: c.as_dict()
Out[10]: {'energy': 12, 'ra': 0.0}
```

Row-wise Data Output and further processing

TableWriter (serialize Containers to Tables, without keeping whole table in memory)**:**

- For writing data efficiently when you don't have the whole column at once
- Most Common Usage Pattern (not planned):

ctapipe event loop → HDF5TableWriter → HDF5 files → **pandas.read_hdf**()

- ➤ Pandas breaks if any column has array data (astropy.table supports that though)
- ➤ Pandas tends to strip off all useful metadata…
- ➤ Still need a better solution that users like

[48]:
```
data = utils.get_dataset_path("gamma_test_large.simtel.gz")
source = event_source(data, allowed_tels=[1,2,3,4],  max_events=10) # re
```

[50]:
```
from ctapipe.io import HDF5TableWriter
```

[51]:
```
with HDF5TableWriter(filename='hillas.h5', group_name='dl1', overwrite=

    for event in source:
        calib.calibrate(event)

        for tel_id, tel_data in event.dl1.tel.items():
            tel = event.inst.subarray.tel[tel_id]
            mask = tailcuts_clean(tel.camera, tel_data.image[0])
            params = hillas_parameters(tel.camera[mask], tel_data.image
            writer.write("hillas", params)
```

[53]:
```
import pandas as pd

hillas = pd.read_hdf("hillas.h5", key='/dl1/hillas')
hillas
```

[53]:

	intensity	kurtosis	length	phi	psi	r	skewness	width	
0	516.973074	2.002860	0.241504	158.337072	63.038146	1.044776	0.306567	0.047223	-0.970
1	12351.140514	3.896901	0.257008	131.028596	-77.249721	0.832273	1.061792	0.114391	-0.546
2	104.850078	5.350268	0.321235	46.050519	88.237061	0.812167	-2.056825	0.024754	0.563
3	111.341088	2.111895	0.050280	69.051947	79.696708	0.960495	0.138985	0.022882	0.343
4	92.755466	5.135449	0.217182	121.504415	80.614031	0.900817	-1.966423	0.021410	-0.470

Instrument Description (ctapipe.instrument)

SubarrayDescription

tel[i]

+ pos_x[i]
+ pos_y[i]
+ footprint

Astropy Table conversion: + to_table()

TelescopeDescription

str() → "MST:FlashCam"
→ "SST-ASTRI:ASTRICam"

optics

camera

str() → "MST"
→ "SST-ASTRI"

OpticsDescription

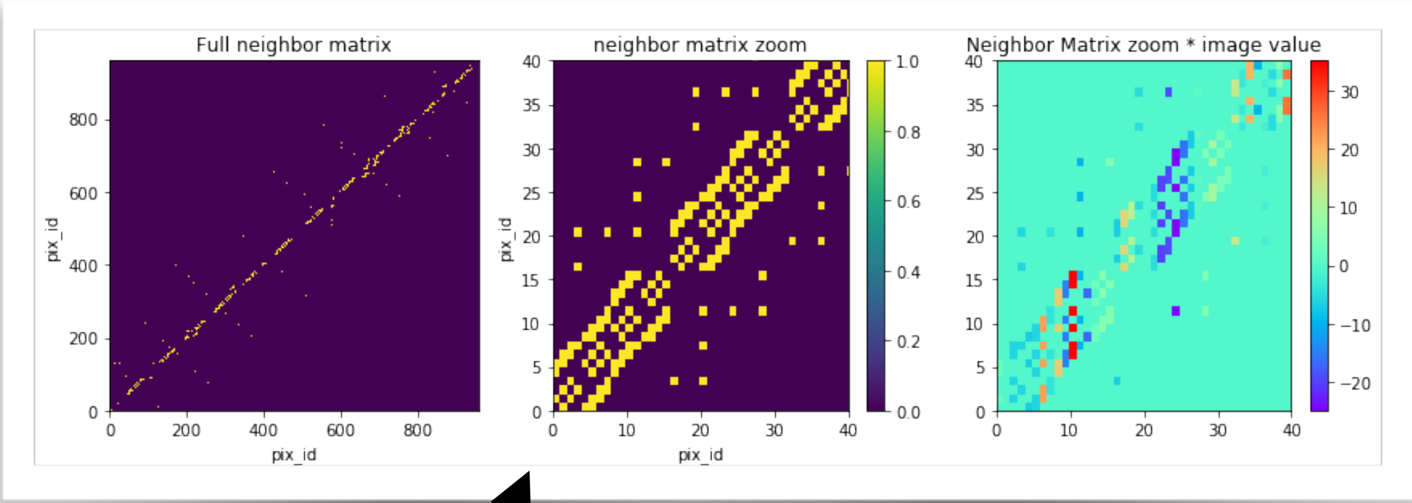
CameraGeometry

str() → "FlashCam"
→ "ASTRICam

+ effective_focal_length
+ mirror_area
+ mirror_type
+ num_mirror_tiles

+ pix_x *[shape: npix]*
+ pix_y *[shape: npix]*
+ pix_id *[shape: npix]*
+ pix_area *[shape: npix]*
+ pix_type
+ neighbors (list)
+ neighbor_matrix (shape: npix,npix)

+ to_table()



Configuration System: based on traitlets

Based on Traitlets library

Component (traitlets.config.Configurable)

- wrapper for complex algorithms that need to have user-level configuration parameters
- Parameters are defined in class Traitlets subclass

Tool (traitlets.config.Application)

- a UI, currently command-line application
- handles user configuration (command line or config file parameters) for a set of Components
- manages the Provenance system
- manages signals, etc.
- set up logging

https://traitlets.readthedocs.io

```
ctapipe-display-dl1 --help-all
Calibrate dl0 data to dl1, and plot the photoelectron images.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

-D
    Display the photoelectron images on-screen as they are produced.
--max_events=<Int> (EventSource.max_events)
    Default: None
    Maximum number of events that will be read from the file
--extractor=<CaselessStrEnum> (DisplayDL1Calib.extractor_product)
    Default: 'NeighbourPeakIntegrator'
    Choices: ['FullIntegrator', 'SimpleIntegrator', 'GlobalPeakIntegrator', 'LocalPeakIntegrator',
'NeighbourPeakIntegrator', 'AverageWfPeakIntegrator']
    ChargeExtractor to use.
--t0=<Int> (SimpleIntegrator.t0)
    Default: 0
    Define the peak position for all pixels
--window_width=<Int> (WindowIntegrator.window_width)
    Default: 7
    Define the width of the integration window
--window_shift=<Int> (WindowIntegrator.window_shift)
    Default: 3
    Define the shift of the integration window from the peakpos (peakpos -
    shift)
--sig_amp_cut_HG=<Float> (PeakFindingIntegrator.sig_amp_cut_HG)
    Default: None
    Define the cut above which a sample is considered as significant for
    PeakFinding in the HG channel
--sig_amp_cut_LG=<Float> (PeakFindingIntegrator.sig_amp_cut_LG)
    Default: None
    Define the cut above which a sample is considered as significant for
    PeakFinding in the LG channel
--lwt=<Int> (NeighbourPeakIntegrator.lwt)
    Default: 0
    Weight of the local pixel (0: peak from neighbours only, 1: local pixel
    counts as much as any neighbour
--clip_amplitude=<Float> (CameraDL1Calibrator.clip_amplitude)
    Default: None
    Amplitude in p.e. above which the signal is clipped. Set to None for no
    clipping.
-T <Int> (DisplayDL1Calib.telescope)
    Default: None
    Telescope to view. Set to None to display all telescopes.
-O <Unicode> (ImagePlotter.output_path)
    Default: None
    Output path for the pdf containing all the images. Set to None for no saved
    output.
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
    Set the log level by value or name.
--config=<Unicode> (Tool.config_file)
    Default: ''
    name of a configuration file with parameters to load in addition to command-
    line parameters

Class parameters
----------------

Parameters are set from command-line arguments of the form:
`--Class.trait=value`. This line is evaluated in Python, so simple expressions
are allowed, e.g.:: `--C.a='range(3)'` For setting C.a=[0,1,2].

DisplayDL1Calib options
-----------------------
--DisplayDL1Calib.config_file=<Unicode>
    Default: ''
    name of a configuration file with parameters to load in addition to command-
    line parameters
--DisplayDL1Calib.extractor_product=<CaselessStrEnum>
    Default: 'NeighbourPeakIntegrator'
    Choices: ['FullIntegrator', 'SimpleIntegrator', 'GlobalPeakIntegrator', 'LocalPeakIntegrator',
'NeighbourPeakIntegrator', 'AverageWfPeakIntegrator']
    ChargeExtractor to use.
--DisplayDL1Calib.log_datefmt=<Unicode>
    Default: '%Y-%m-%d %H:%M:%S'
    The date format used by logging formatters for %(asctime)s
--DisplayDL1Calib.log_format=<Unicode>
    Default: '[%(name)s]%(highlevel)s %(message)s'
    The Logging format template
--DisplayDL1Calib.log_level=<Enum>
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
    Set the log level by value or name.
--DisplayDL1Calib.telescope=<Int>
    Default: None
    Telescope to view. Set to None to display all telescopes.

EventSource options
-------------------
--EventSource.allowed_tels=<Set>
    Default: set()
    list of allowed tel_ids, others will be ignored. If left empty, all
    telescopes in the input stream will be included
--EventSource.input_url=<Unicode>
    Default: ''
    Path to the input file containing events.
--EventSource.max_events=<Int>
    Default: None
    Maximum number of events that will be read from the file

CameraDL1Calibrator options
---------------------------
--CameraDL1Calibrator.clip_amplitude=<Float>
```

Metadata and Provenance

Requirement that CTA data products are *reproducible*

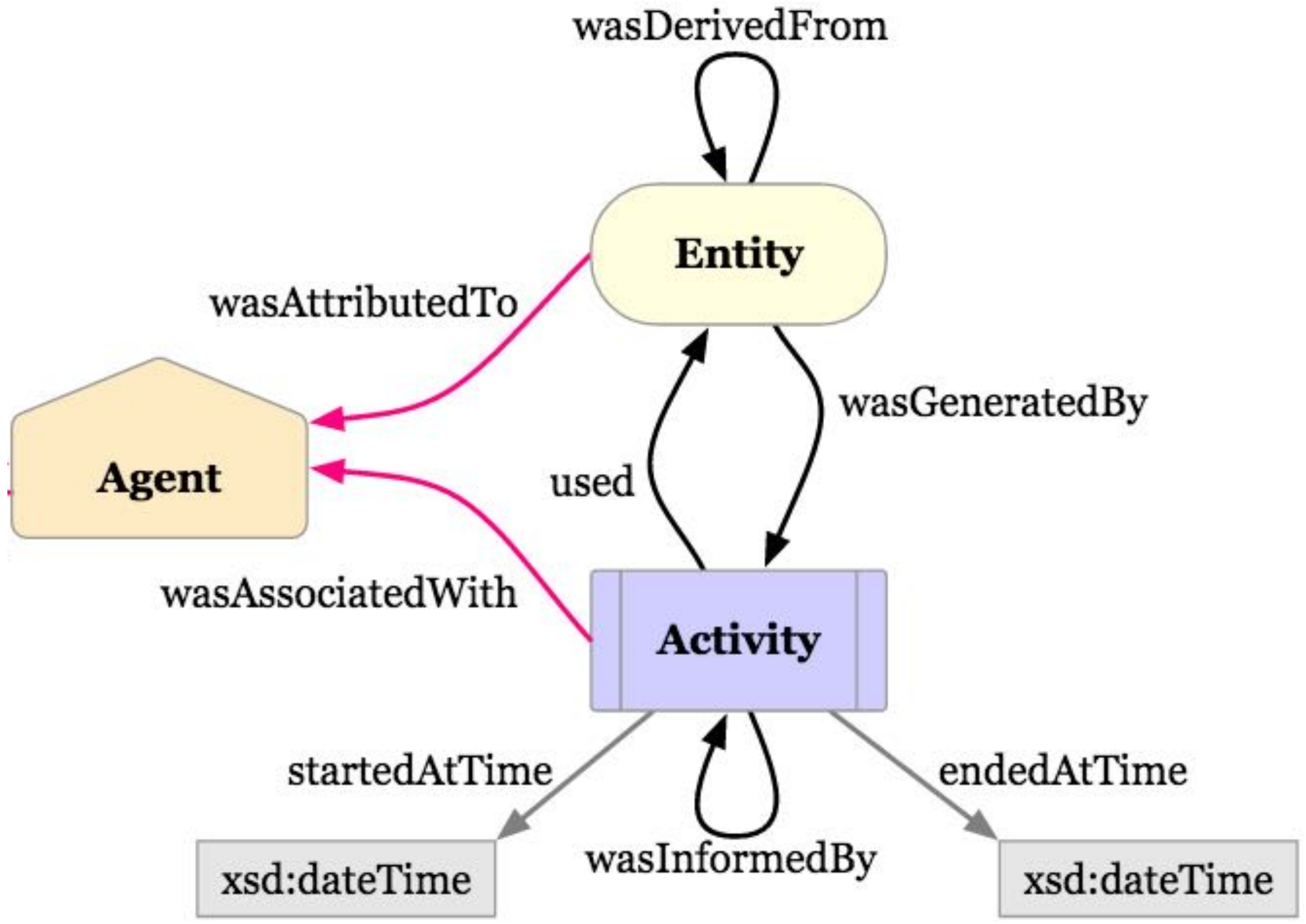
- *software version*
- *configurations*
- *inputs*
  - ➤ *1 IRF might have 1000s of input files, tables, calibration coefficients, lab measurements*

Inside ctapipe "Tools" automatically keep track of at least the "local provenance" metadata

- *Any file opened (input or output) is automatically tracked*
- *The "activity" details are also recorded (local machine name, running time, and other info)*

Local provenance can be put into a database to derive the full chain of processing history for any output file

See talk by Matthieu Servillat



Code Example

ctapipe.reco.HillasReconstructor

```
def estimate_core_position(self, hillas_dict, telescope_pointing):
        psi = u.Quantity([h.psi for h in hillas_dict.values()])
        z = np.zeros(len(psi))
        uvw_vectors = np.column_stack([np.cos(psi).value, np.sin(psi).value, z])

        tilted_frame = TiltedGroundFrame(pointing_direction=telescope_pointing)
        ground_frame = GroundFrame()

        positions = [
            (
                SkyCoord(*plane.pos, frame=ground_frame)
                .transform_to(tilted_frame)
                .cartesian.xyz
            )
            for plane in self.hillas_planes.values()
        ]
        core_position = line_line_intersection_3d(uvw_vectors, positions)

        core_pos_tilted = SkyCoord(
            x=core_position[0] * u.m,
            y=core_position[1] * u.m,
            frame=tilted_frame
        )

        core_pos = project_to_ground(core_pos_tilted)

        return core_pos.x, core_pos.y
```

0.6.2.post202+git4ee0824

arch docs

tting Started For Developers
velopment Guidelines
torials
Getting Started with ctapipe
Part 1: load and loop over data
Part 2: Explore the instrument description
Part 3: Apply some calibration and trace integration
Part 4: Let's put it all together:
xploring Raw Data
xplore Calibrated Data
Make a theta-square plot
018 LST Bootcamp walkthrough
amples
quently Asked Questions
l Docs
ferences
ange Log

Docs » Tutorials » Getting Started with ctapipe

Getting Started with ctapipe

This hands-on was presented at the Paris CTA Consoritum meeting (K

Part 1: load and loop over data

[1]:
```
from ctapipe.io import event_source
from ctapipe import utils
from matplotlib import pyplot as plt
%matplotlib inline
```

[2]:
```
path = utils.get_dataset_path("gamma_test_large.simtel.gz")
source = event_source(path, max_events=4)
```

[3]:
```
for event in source:
    print(event.count, event.r0.event_id, event.mc.energy)
```

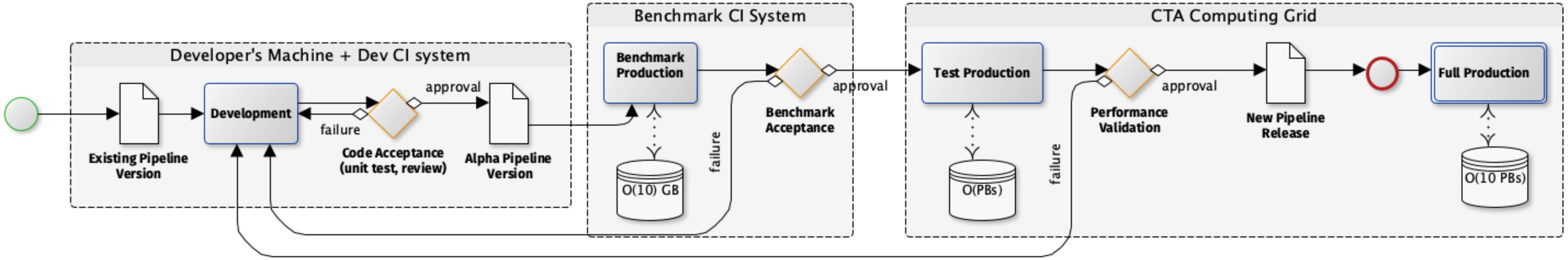
```
0 23703 0.5707105398178101 TeV
1 31007 1.8637498617172241 TeV
2 31010 1.8637498617172241 TeV
3 31012 1.8637498617172241 TeV
```

[4]:
```
event
```

[4]:
```
ctapipe.io.containers.DataContainer:
                        event_type: Event type
                              r0.*: Raw Data
                              r1.*: R1 Calibrated Data
                             dl0.*: DL0 Data Volume Reduced Data
                             dl1.*: DL1 Calibrated image
                             dl2.*: Reconstructed Shower Informat
                              mc.*: Monte-Carlo data
```

*Tutorials and examples in documentation using **nbsphinx** plugin*

Benchmarking



Current plan (partially realized):

- Collection of Jupyter notebooks
  - ➤ data preparation
  - ➤ low-level benchmarks
  - ➤ high-level summaries
- Papermill:
  - ➤ parameterization of notebooks
  - ➤ notebook output data access

Open Questions

Can we use ctapipe python algorithms in our RTA?

- preliminary studies say maybe
- tests using dask, spark and others found some bottlenecks (not related to algorithms themselves), but more work to do

What should the output data format be?

- so far we like HDF5, but some problems
- FITS for DL3…. still some things to define
-