CTLearn: Deep Learning for Gamma-ray Astronomy

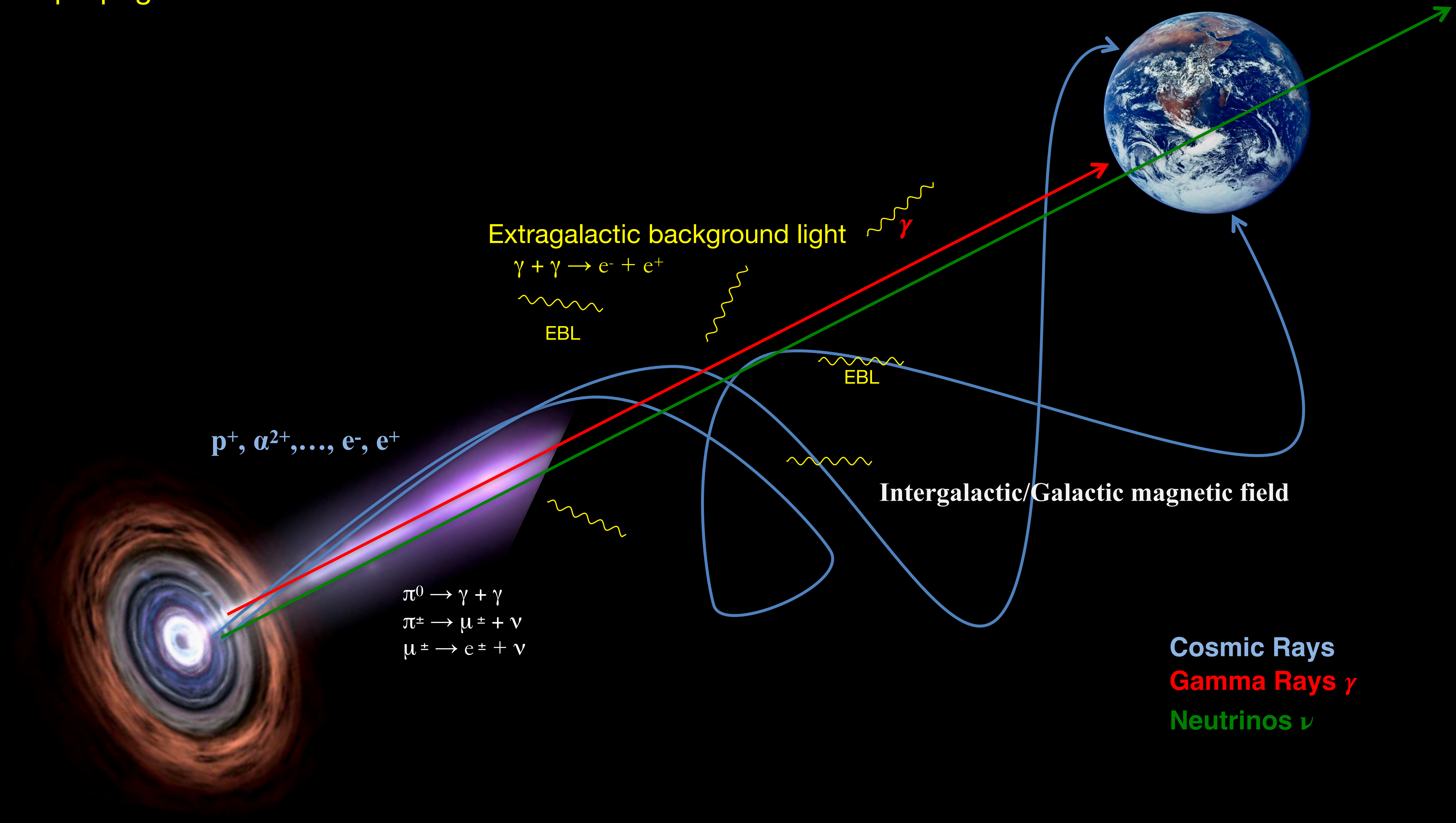
Qi Feng
for D. Nieto, A. Brill, Q. Feng, T. B. Humensky, B. Kim, T. Miener, R. Mukherjee, J. Sevilla, and T. Vuillaume

Barnard College / Columbia University

2019 Oct 6, Brooklyn, NY

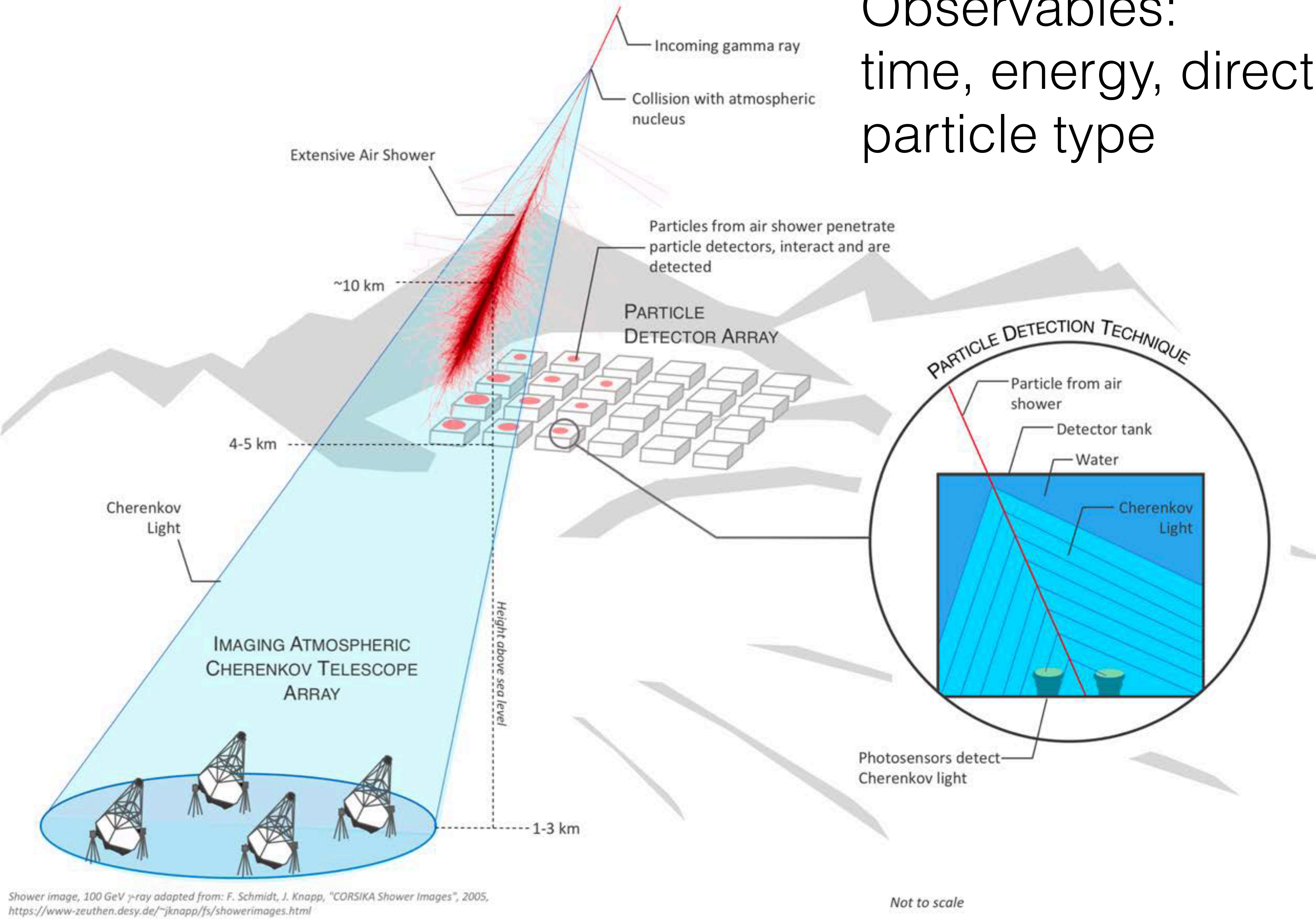
Gamma rays: cosmic-ray messengers

Study particle acceleration and emission mechanisms and propagation effects.



Gamma rays cannot penetrate the atmosphere

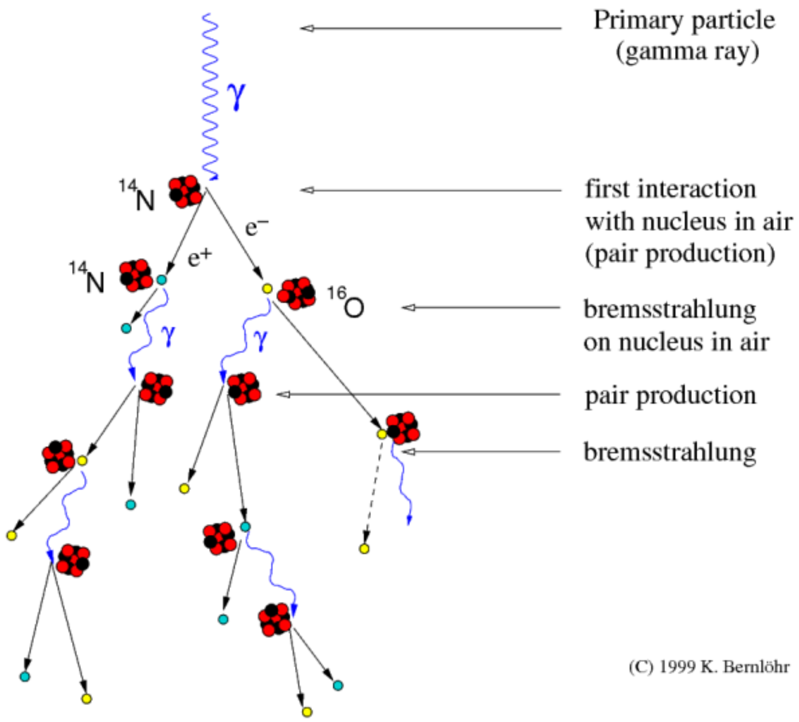
Observables:
time, energy, direction, particle type



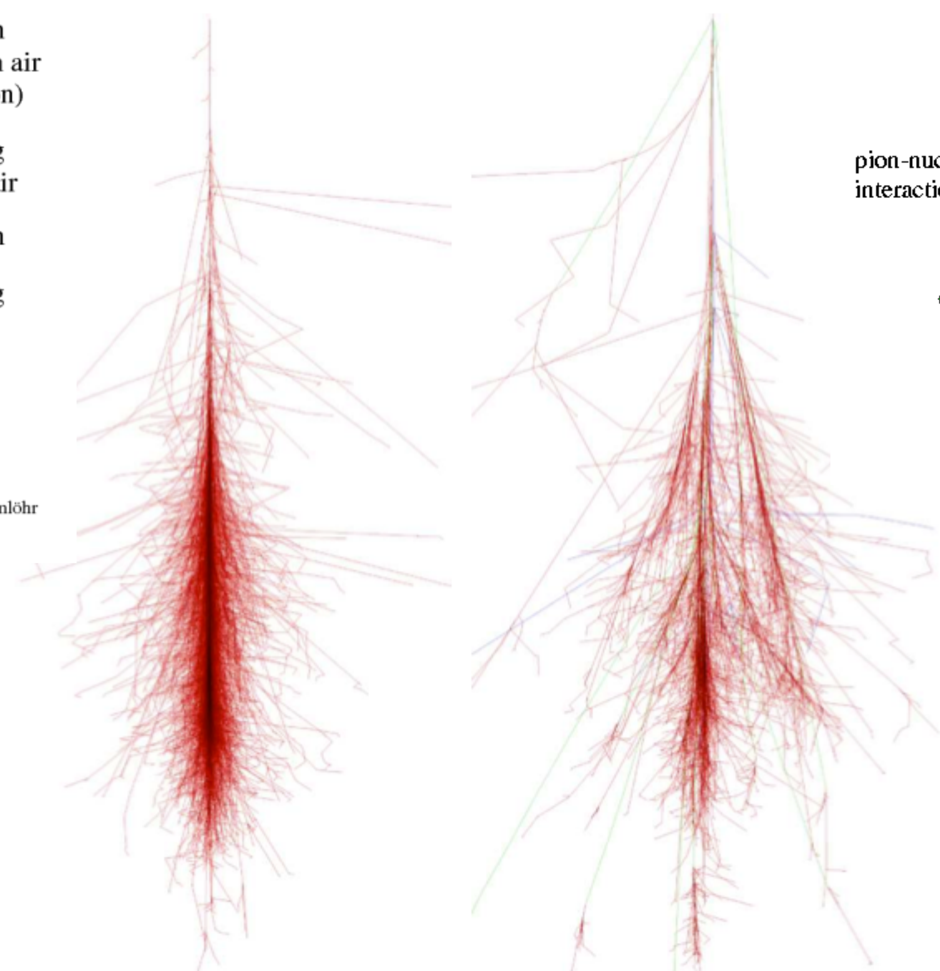
Extensive Air Shower (EAS) from VHE gamma rays and cosmic rays

Signal

Development of gamma–ray air showers



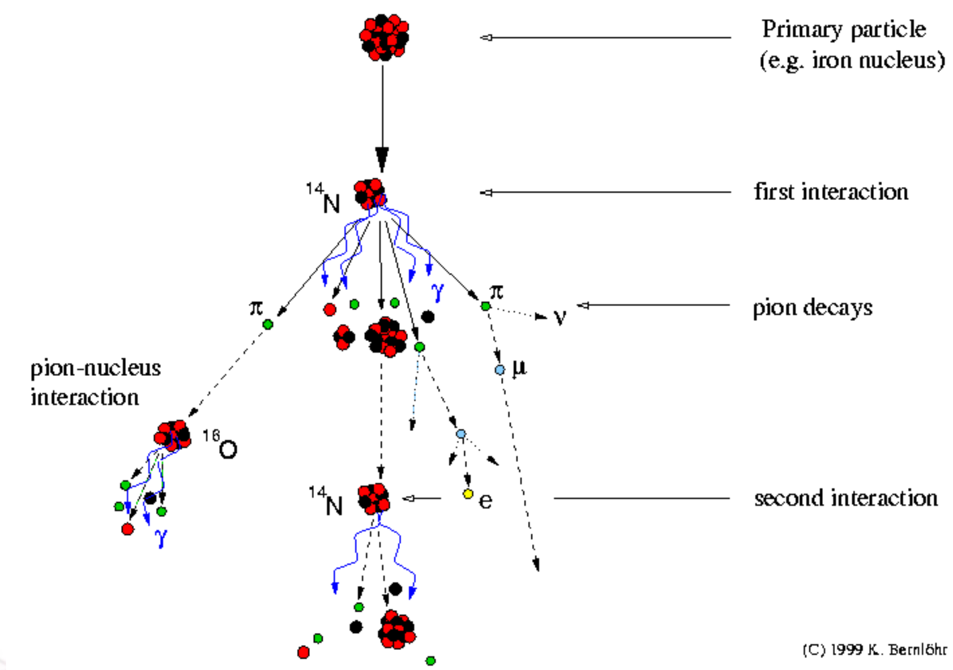
Background



(a) Longitudinal view of 100 GeV photon.

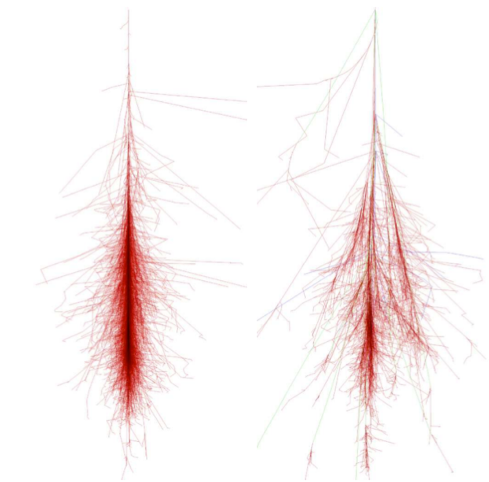
(b) Longitudinal view of 100 GeV proton.

Development of cosmic-ray air showers



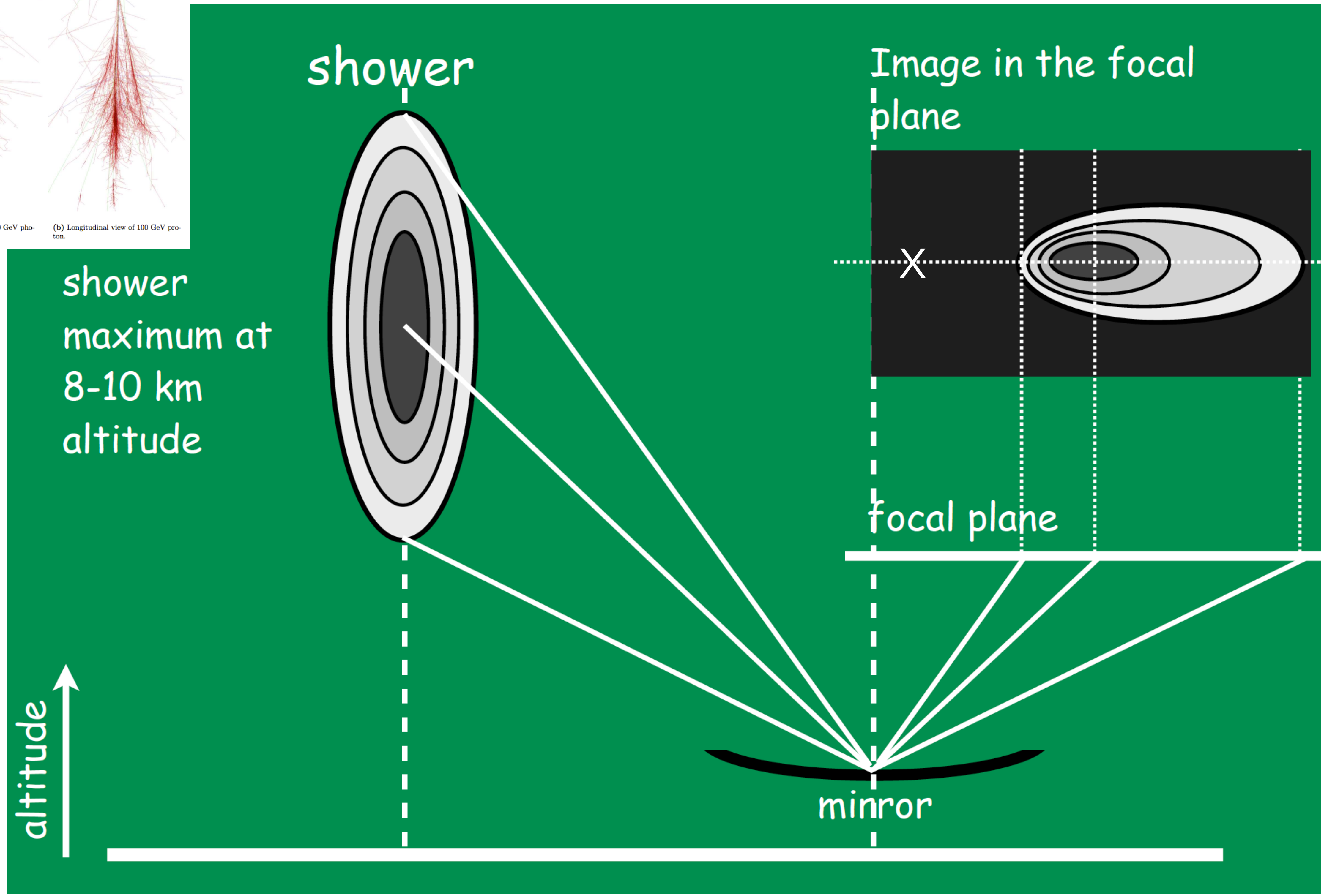
Imaging Atmospheric Cherenkov Arrays

Gamma/CR separation

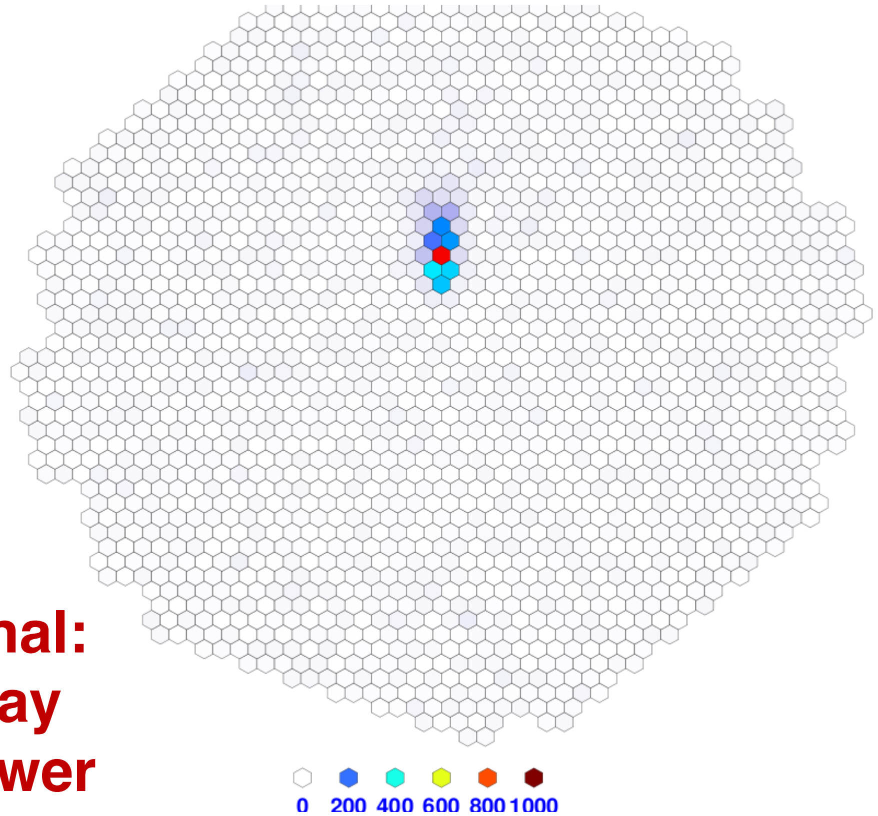


(a) Longitudinal view of 100 GeV photon.

(b) Longitudinal view of 100 GeV proton.

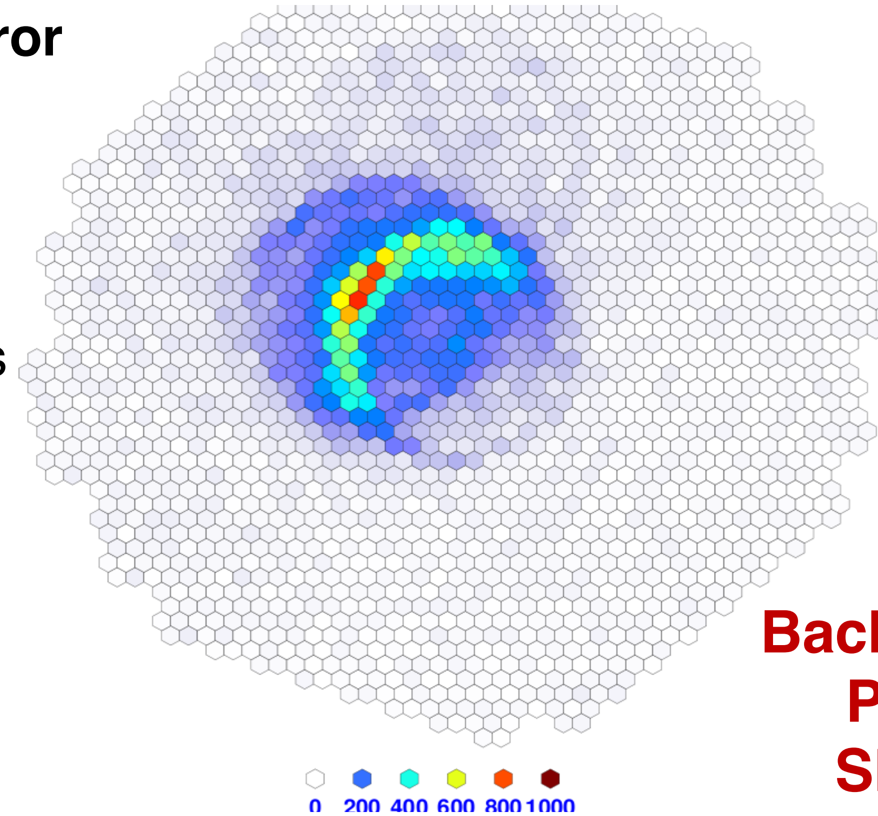


Shower image from gamma ray and cosmic ray



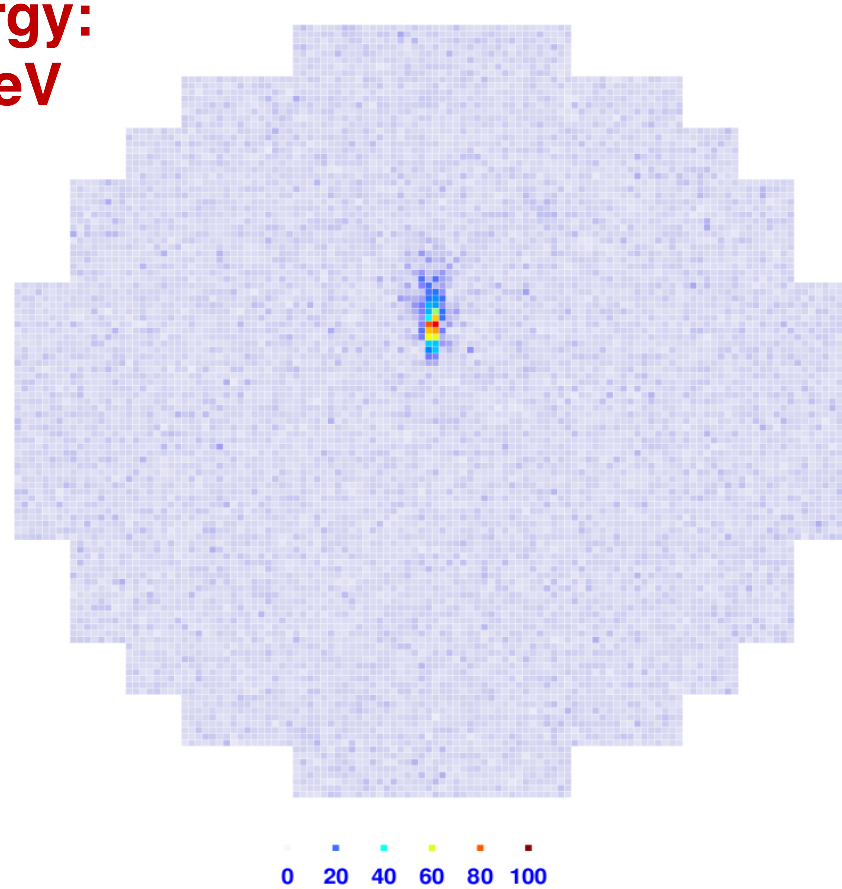
Single-Mirror MST

8° FoV
1,570
0.18° pixels



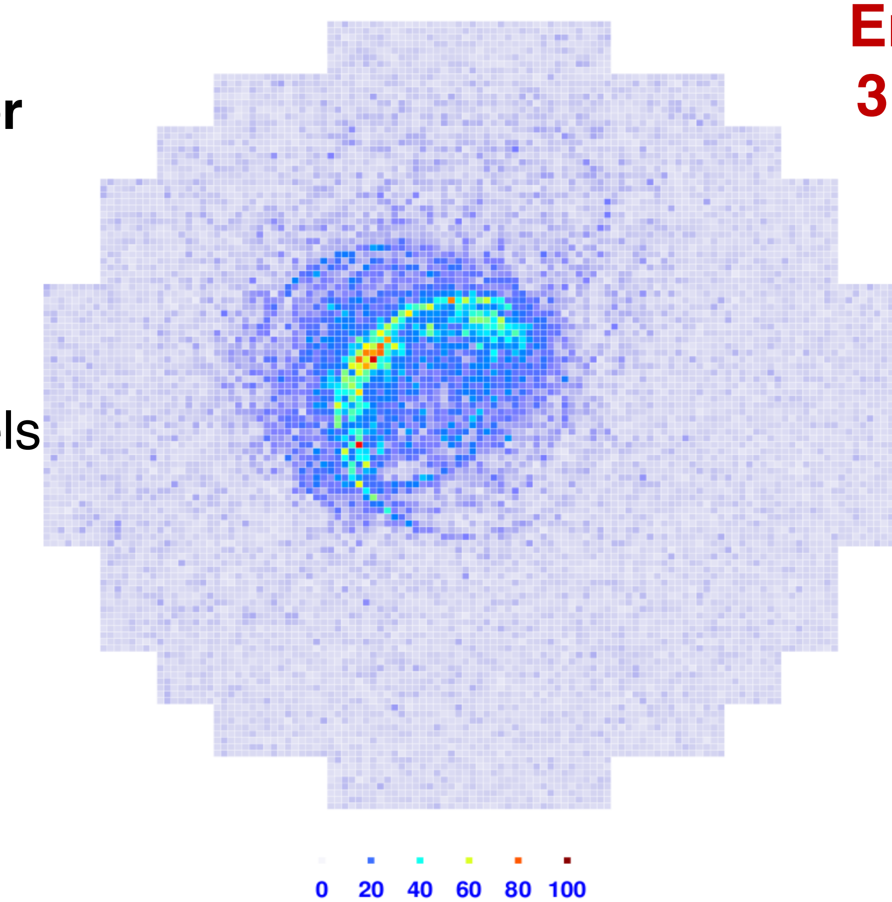
Signal:
γ-ray
Shower
Energy:
1 TeV

Background:
Proton
Shower
Energy:
3.2 TeV



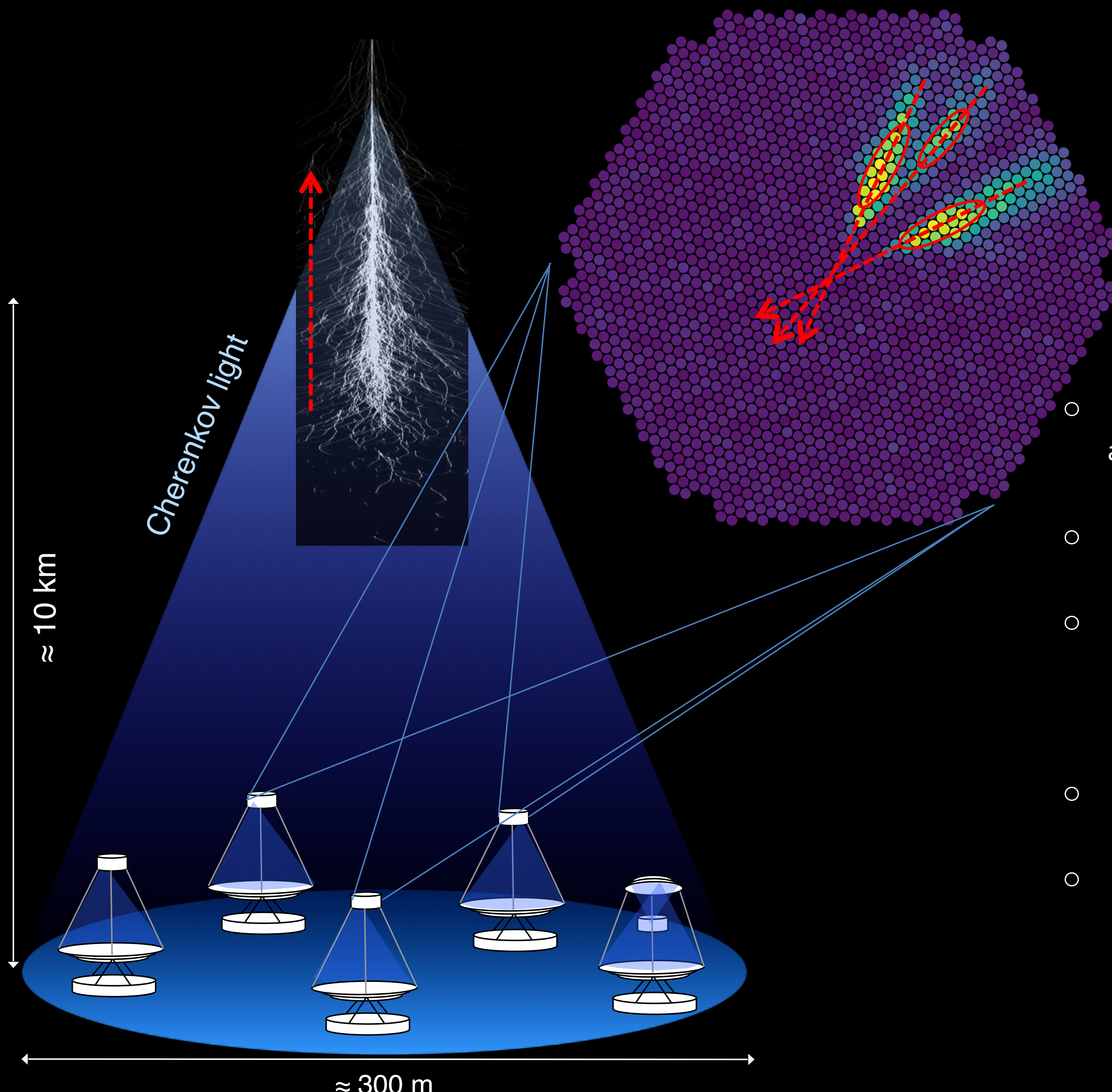
Dual-Mirror SCT

8° FoV
11,328
0.067° pixels



Imaging Atmospheric Cherenkov Arrays

Stereo Imaging



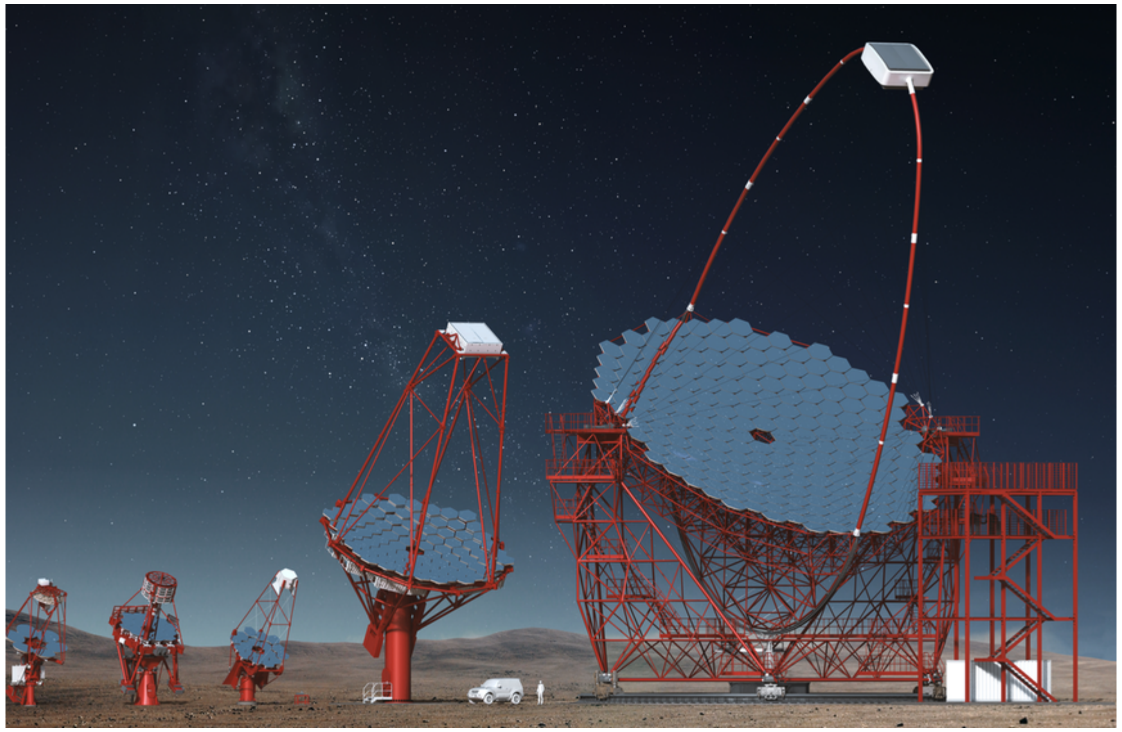
- Detection of extended air showers using the atmosphere as a calorimeter
- Huge γ–ray collection area (~10^5 m^2)
- Large background from charged CR
  - Partly irreducible (e^-/e^+ , single-EM, with current methods)
- Energy window: tens GeV - tens TeV
- Event reconstruction from image:
  - Type of primary event
  - Primary energy estimation
  - Primary arrival direction

Image from D. Nieto

Cherenkov Telescope Array (CTA)

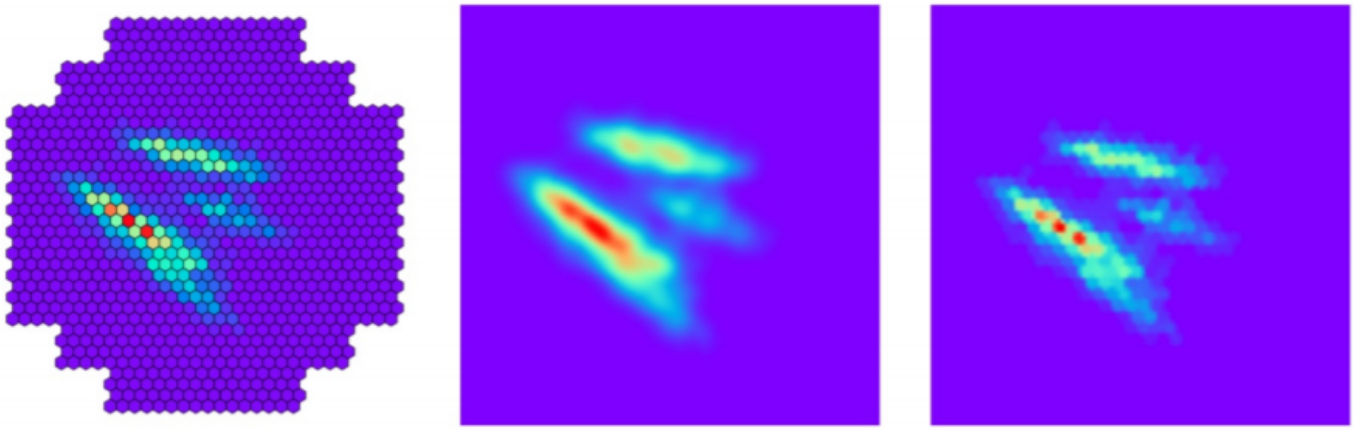
- 10 fold sensitivity of current instruments
- gy coverage: tens of GeV to >100 TeV
- resolution and energy resolution
- rrays (North/South)

IACT Deep Learning Challenges



Combine images from a telescope array containing many different telescopes which may or may not have triggered → best results so far achieved with recurrent neural networks (RNN), but still preliminary

Process **hexagonally spaced pixels** into a square matrix → many methods under study, no consensus yet



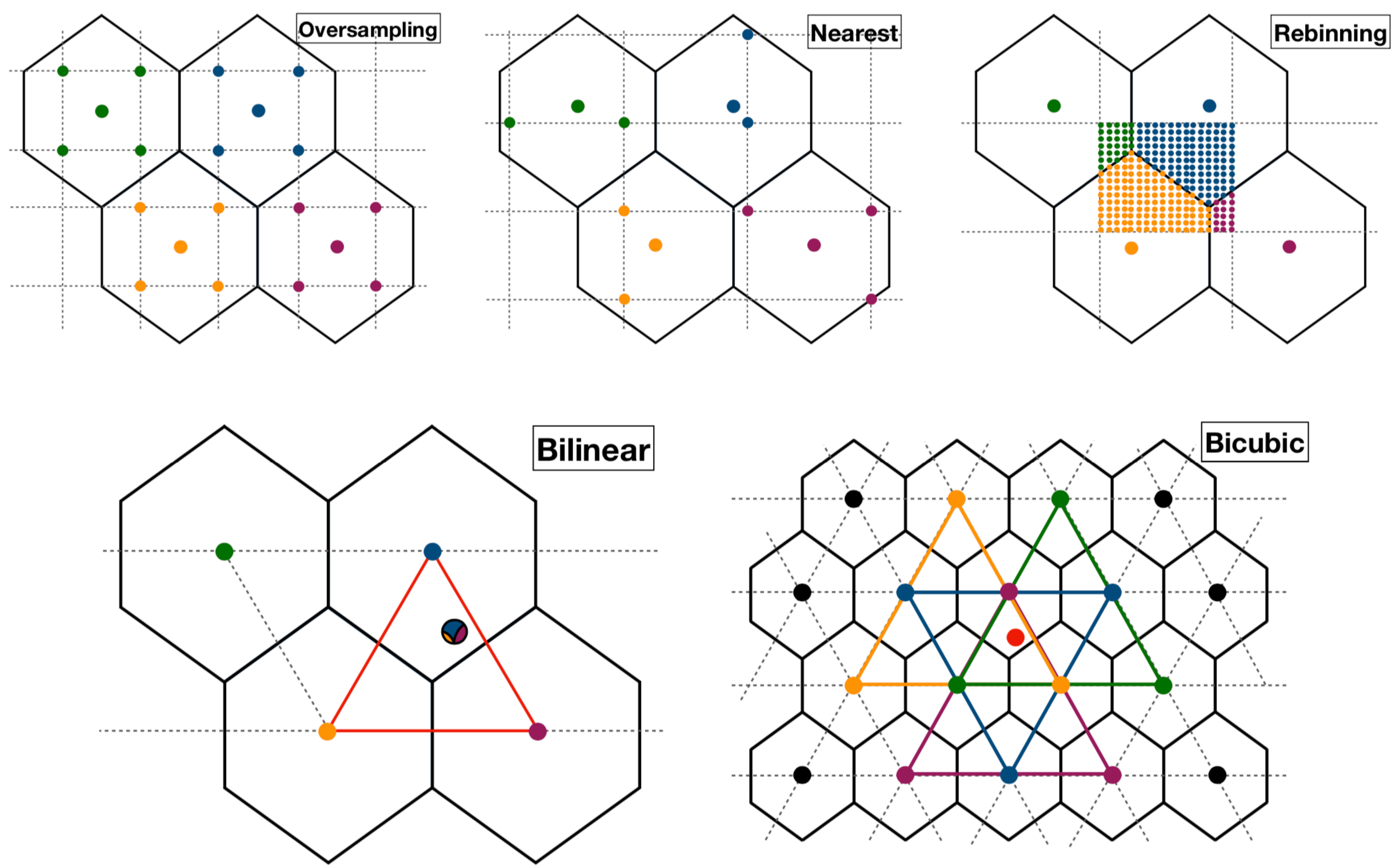
Shilon et al. 2018

Get lots of **labeled training data**:

1. Reprocess MC simulations
   → ImageExtractor
2. Understand effect of subtle differences between sims and data
   → Generative Adversarial Nets?

IACT Deep Learning Challenges

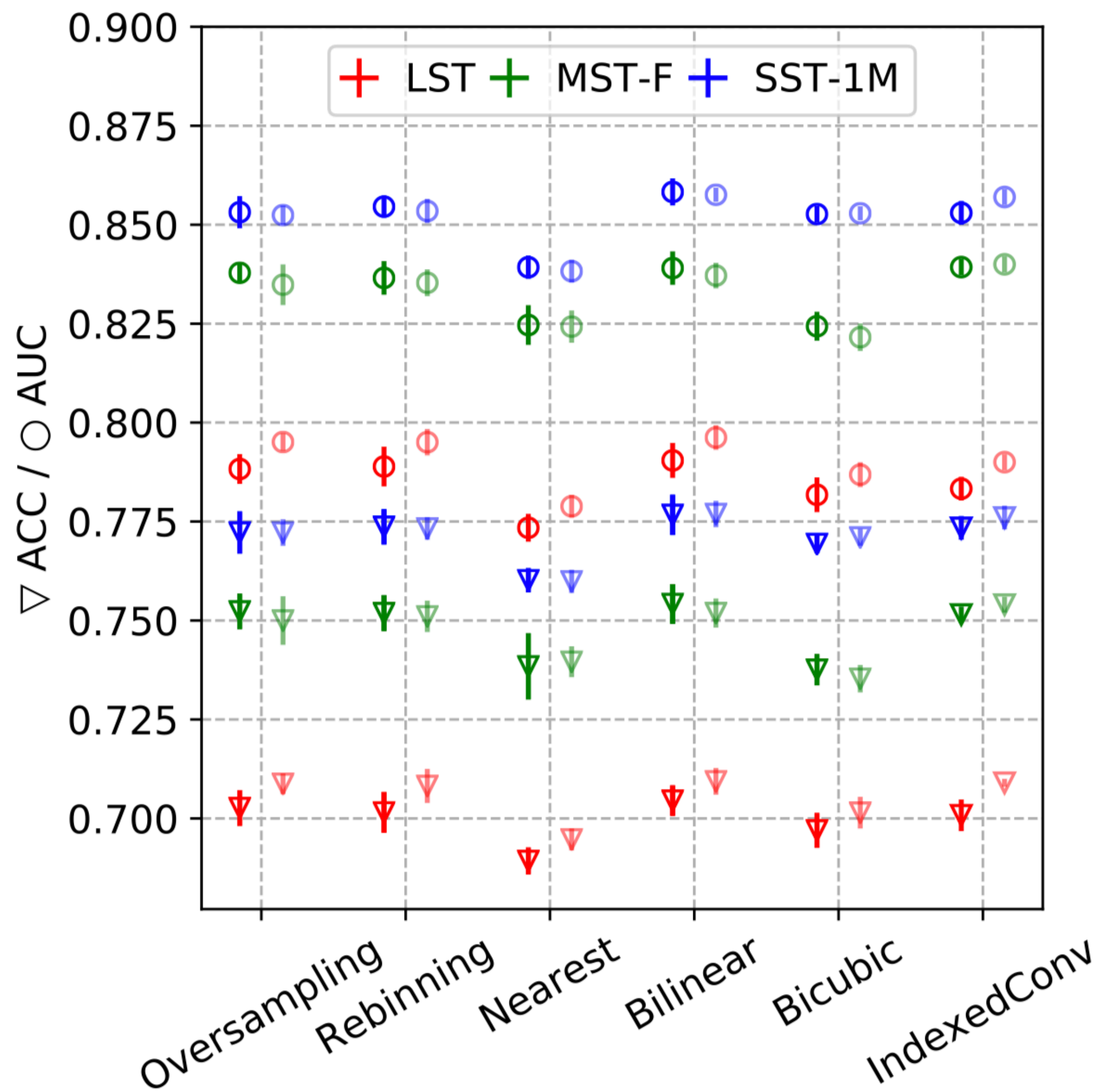
Process **hexagonally spaced pixels** into a square matrix
→ Different image mapping methods and hexagonal convolution (via IndexedConv) compared.
→ Image mapping + conventional 2D convolution was performing identical to hexagonal convolution (within errors).



Nieto et al. 2019
PoS(ICRC2019)753

IACT Deep Learning Challenges

Process **hexagonally spaced pixels** into a square matrix
→ Different image mapping methods and hexagonal convolution (via IndexedConv) compared.
→ Image mapping + conventional 2D convolution was performing identical to hexagonal convolution (within errors).



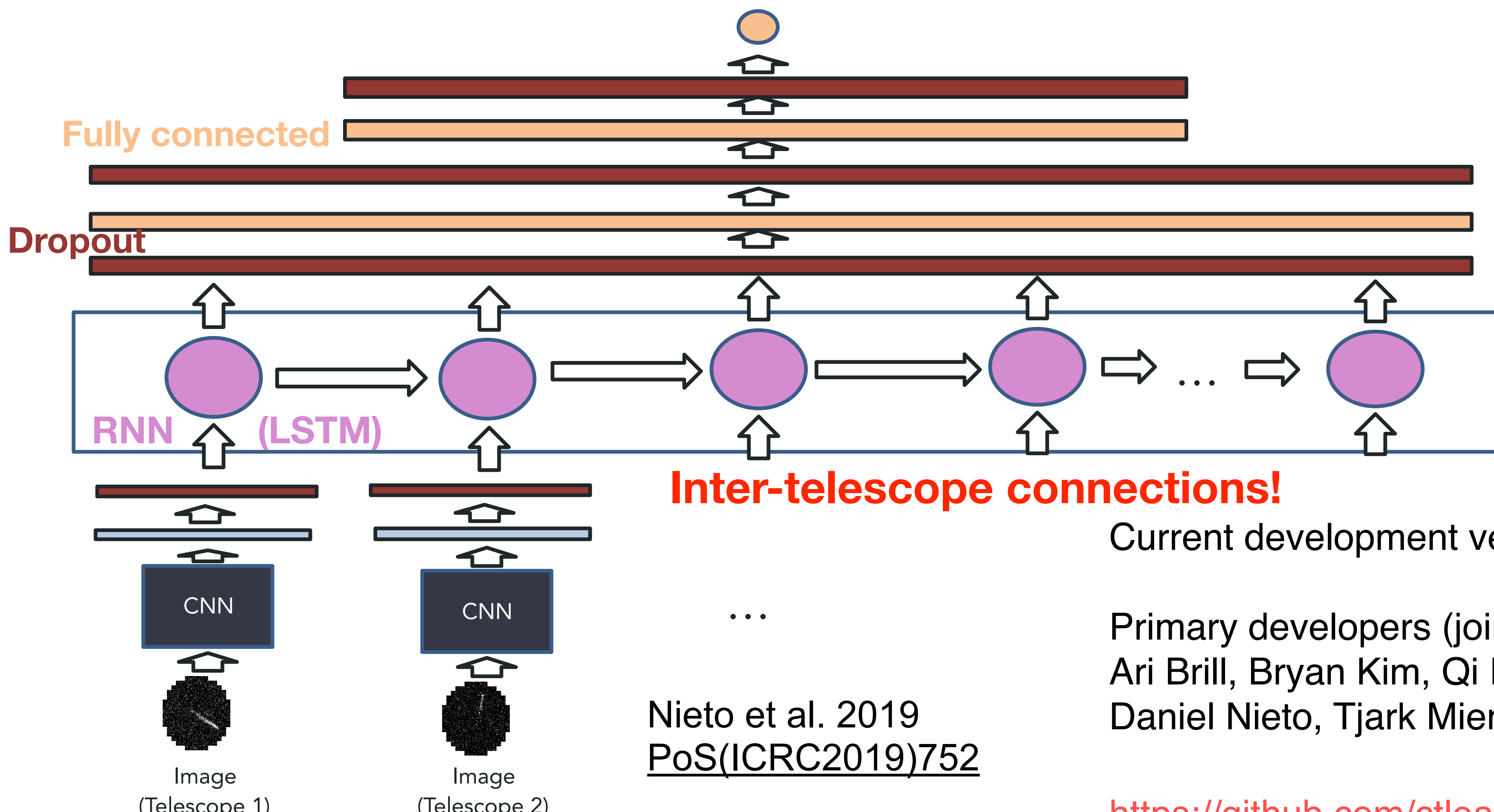
Nieto et al. 2019
PoS(ICRC2019)753

CTLearn

High-level Python library for doing deep learning with IACT image data

Includes modules for loading and manipulating ImageExtractor HDF5 data and for running machine learning models with TensorFlow

Configuration-file-driven workflow drives reproducible training and prediction



Inter-telescope connections!

Nieto et al. 2019
PoS(ICRC2019)752

TensorFlow

Current development version: v0.4.0

Primary developers (join us!):
Ari Brill, Bryan Kim, Qi Feng (Columbia)
Daniel Nieto, Tjark Miener, Jaime Sevilla (UCM)

https://github.com/ctlearn-project/ctlearn

Clone Repository with Git

Clone the CTLearn and DL1-Data-Handler repositories:

```
cd </ctlearn/installation/path>
git clone https://github.com/ctlearn-project/ctlearn.git

cd </dl1-data-handler/installation/path>
git clone https://github.com/cta-observatory/dl1-data-handler.git
```

Install Package with Anaconda

Next, download and install Anaconda, or, for a minimal installation, Miniconda. Create a new conda environment that includes all the dependencies for CTLearn:

```
conda env create -f </installation/path>/ctlearn/environment-<MODE>.yml
```

where `<MODE>` is either 'cpu' or 'gpu' (for linux systems) or 'macos' (for macOS systems), denoting the TensorFlow version to be installed. If installing the GPU version of TensorFlow, verify that your system fulfills all the requirements here. Note that there is no GPU-enabled TensorFlow version for macOS yet.

Finally, install DL1-Data-Handler and CTLearn into the new conda environment with pip:

```
source activate ctlearn

cd <dl1-data-handler/installation/path>/dl1-data-handler
pip install --upgrade .

cd <ctlearn/installation/path>/ctlearn
pip install --upgrade .
```

Configuration

CTLearn encourages reproducible training and prediction by keeping all run settings in a single YAML configuration file.

```
Training:

    # Optional float. Default: 0.1
    # Randomly chosen fraction of data to set aside for validation.
    validation_split: 0.1

    # Required integer.
    # Number of validations made before finishing training. If 0, run fo
    num_validations: 0

    # Required integer.
    # Number of training steps to run before each evaluation
    # on the validation set.
    num_training_steps_per_validation: 1000

    # Required string.
    # Valid options: ['Adadelta', 'Adam', 'RMSProp', 'SGD']
    # Optimizer function for training.
    optimizer: 'Adam'

    # Optional float. Required if optimizer is 'Adam', ignored otherwise
    # Epsilon parameter for the Adam optimizer.
    adam_epsilon: 1.0e-8

    # Required integer.
    # Base learning rate before scaling or annealing.
    base_learning_rate: 0.001
```

CTLearn can be configured to load any TensorFlow model obeying the signature:

logits = model(features, params, training)

```
# Settings for the TensorFlow model. The options in this and the
# Model Parameters section are passed to the Estimator model_fn
# and the user's model function.
Model:

    # Optional string or null. Default if missing or null is to load
    # CTLearn default models directory: 'ctlearn/ctlearn/default_models/'
    # Path to directory containing model module.
    model_directory: null  # '/my/model/path/'

    # Required dictionary containing a module/function pair.
    # Module in model directory and function in module implementing the model.
    # Module and function pairs included with CTLearn:
    #   - {module: 'single_tel', function: 'single_tel_model'}: single
    #       tel model using a basic convolutional neural network (CNN)
    #   - {module: 'cnn_rnn', function: 'cnn_rnn_model'}: array model
    #       feeding output of a basic CNN for each  telescope into a
    #       recurrent neural network (RNN)
    #   - {module: 'variable_input_model', function: 'variable_input_model'}:
    #       array model feeding combined outputs of a CNN for each
    #       telescope into a CNN network head
    model: {module: 'single_tel', function: 'single_tel_model'}
```

Run a Model

Run CTLearn from the command line:

```
CTLEARN_DIR=</installation/path>/ctlearn/ctlearn
python $CTLEARN_DIR/run_model.py myconfig.yml [--mode <MODE>]
[--debug] [-log_to_file]
```

View training progress in real time with TensorBoard:

```
tensorboard --logdir=/path/to/my/model_dir
```

Inspect Data

Print dataset statistics only, without running a model:

```
python $CTLEARN_DIR/run_model.py myconfig.yml --mode load_only
```

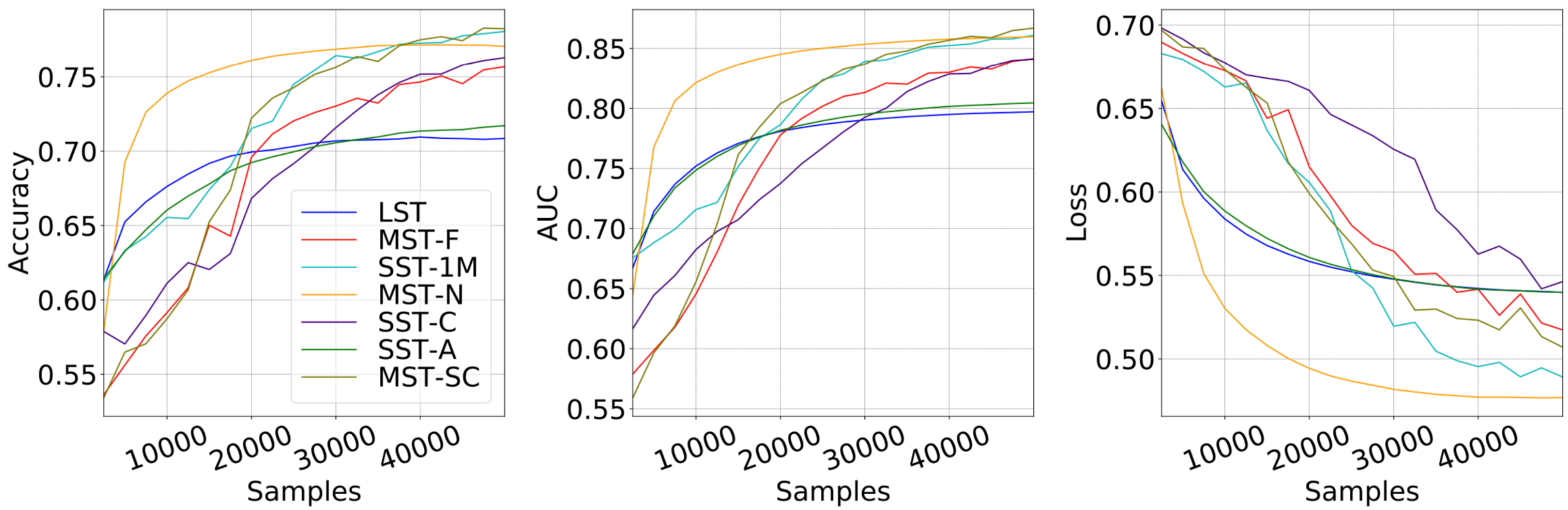
Download Data

CTLearn can load and process data in the HDF5 PyTables format produced from simtel files by DL1DataHandler (https://github.com/cta-observatory/dl1-data-handler).

CNN-RNN (Recurrent)

- Trained on 200k events each of protons and diffuse gammas from ImageExtractor "ProtoML" dataset
- No quality cuts, no pre-selection of data (except for multiplicity)
- Preliminary results look promising albeit no hyperparameter optimization

Single-tel:



Multi-tel CNN-RNN: