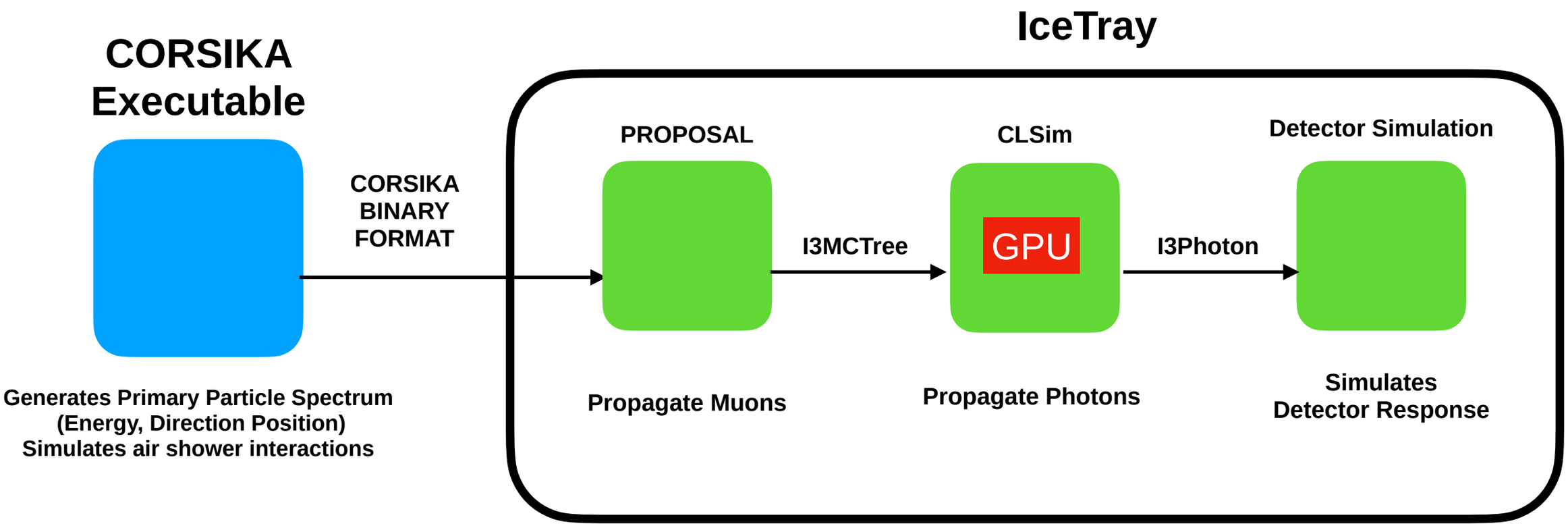
Triggered (Dynamic Stack) CORSIKA And Multiprocess Server

Kevin Meagher
IceCube Simulation Workshop
October 20, 2020

How CORSIKA currently works



I3Photon = Lots of Memory
I3MCPE = Uses less memory

CORSIKA files are generated by a separate CORSIKA binary
IceTray then processes in a linear fashion by each module
First Muons are propagated by by PROPOSAL,
Then Photons are processed by CLSim using the GPU
The entire I3Photon sequence is stored in memory before being converted to the binned I3MCPE format

Why is CORSIKA so hard to produce?

Low Energy:

- CORSIKA shower files are created separately and transferred at the start of the job which saturates IO

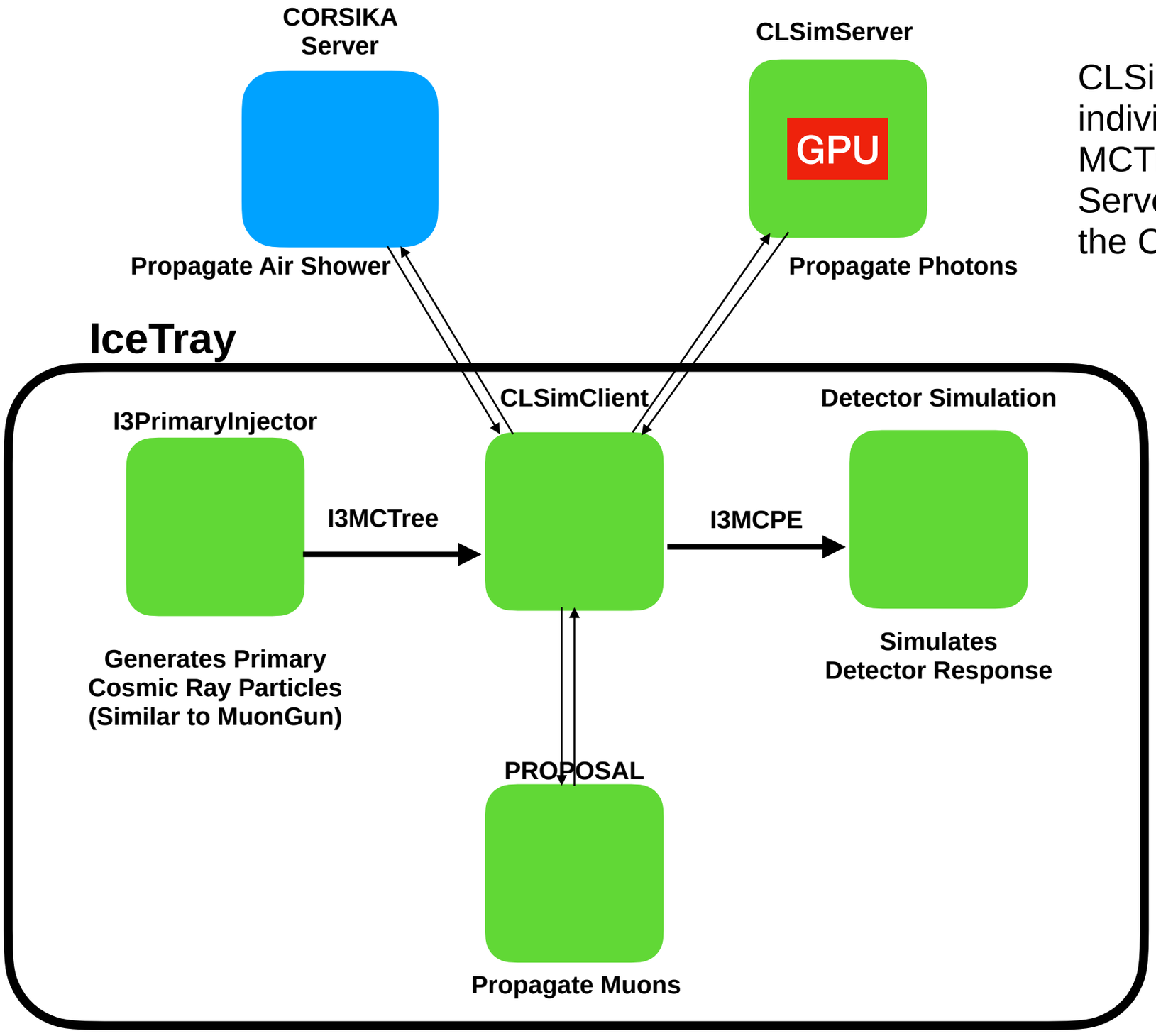
Medium Energy:

- CORSKIA showers need more CPU than GPU which is an inefficient use of our cluster resources (CPUs sit idle while waiting for GPUs to finish)

High Energy:

- Large showers push the memory limits on nodes. CLSim needs to store the entire event (both the MCTree and I3Photons) in memory. Power-law statistics require that we have to allocate memory for very rare events.

Review: Dynamic Stack CORSIKA and Multiprocess Server CLSim



CLSimClient passes individual particles from the MCTree to the CORSIKA Server, to PROPOSAL to the CLSimServer

I3MCPE are created directly from the output of each individual CLSim propagation Saving memory

What do we gain from this?

No Need to break up CORSIKA jobs by energy:

- Low Energy:
  - No need to generate CORSIKA files separately (prevents IO bottleneck)
- Medium Energy:
  - Multiple instances of IceTray can share a CLSimServer resulting in better CPU utilization
- High Energy:
  - Individual particles are passed from CORSIKA to PROPOSAL to CLSim and binned I3MCPE are made for each particle from CLSim rather than the entire event. This significantly reduces the memory footprint
  - Multiple instance of IceTray can run in the same cluster job

Event More Benefits: Oversampling and Other Tricks

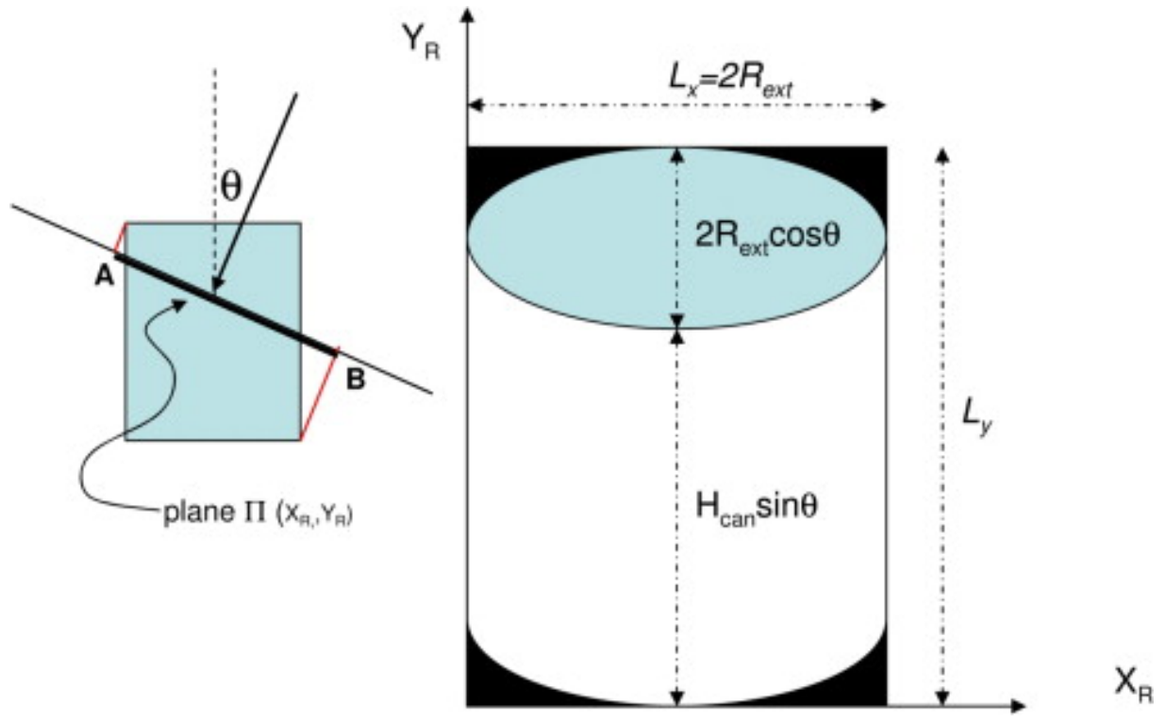
- Cosmic ray air shower primary can be generated according to arbitrary spectral and spatial distributions
  - Generate primaries directly on the detector cylinder before cosmic ray propagation (similar to MuonGun)
- Different CORSIKA configuration cards can be sent to different events
  - Set muon energy threshold based on the inclination of the shower
- CORSIKA propagation of a shower can be under-sampled based on shower development
  - Kill events with low leading energy muon

Other ideas which have been proposed to bias simulation

- Oversample coincident events: eg 1 for coincident showers, 10^{-4} for single showers
- Only populate EM component if the shower hits IceTop.
- Oversample IceCube muon “Lanes”
- Tau analysis: oversample charmed mesons
- ESTES: oversample veto hotspots
- MESE/cascades: oversample based on expected charge in the veto region
- Cosmic rays: oversample high P_T muons
- Force neutrino interaction in air shower

I3PrimaryInjector Module

- Utilizes S-Frame object to keep track of generation surface
- Uses `SampleImpactRay()` to sample on the surface of the cylinder
- Samples the energy from a different power-law for each primary type
- Creates I3MCTree with primary
- Creates an I3ShowerBiasMap and puts it in the frame (so CORSIKA knows how to bias showers)



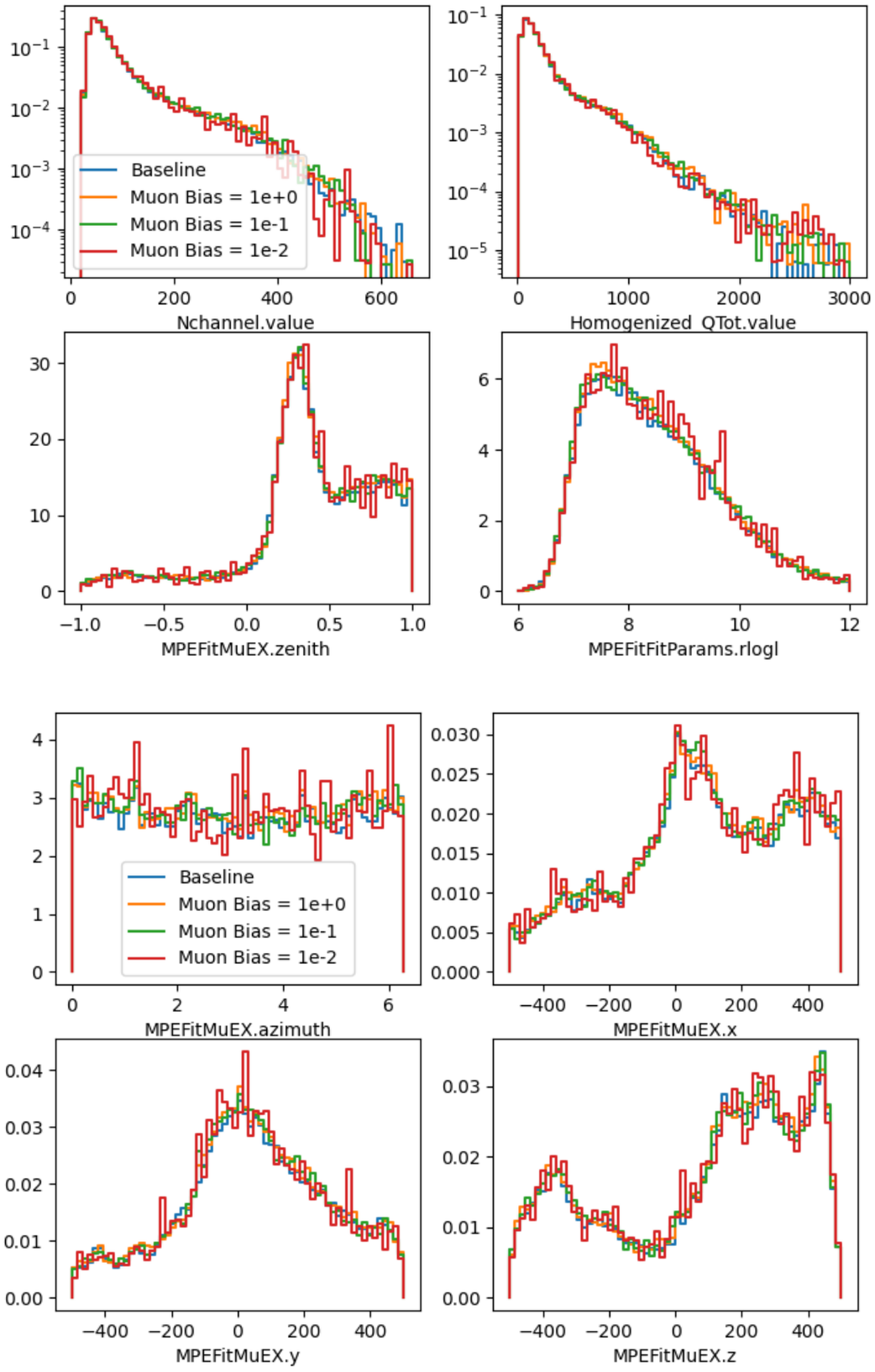
server_sim.py

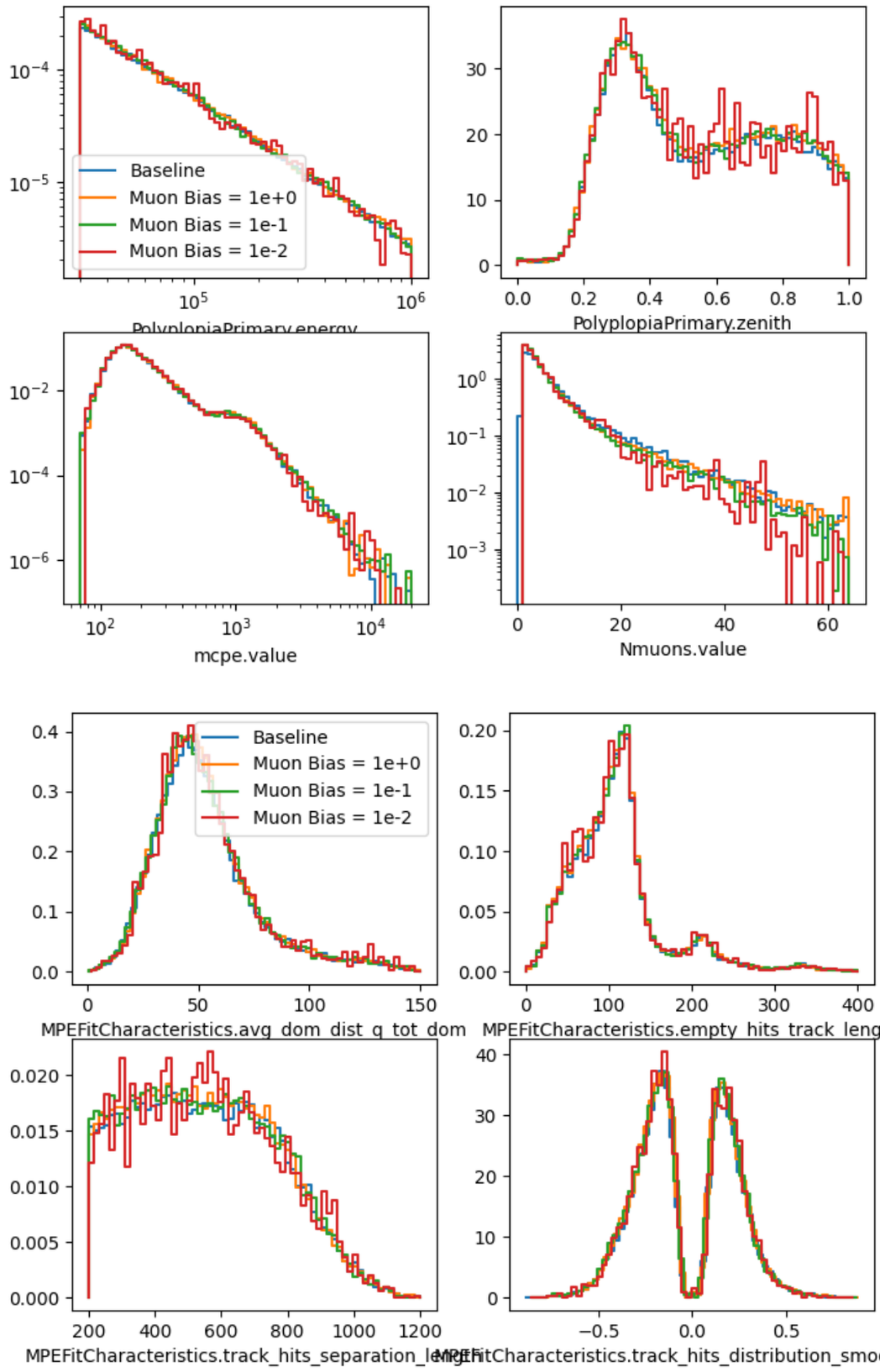
One Script to run the entire shish kabob

- Creates I3CLSimServer
- Runs tray with:
  - I3PrimaryInjector
  - PolyplopiaSegment
  - I3CLSimClientModule with the following propagators:
    - CorsikaService as a CosmicEventGenerator
    - PROPOSAL and I3CMC as propagators
    - I3CLSimLightSourceToStepConverterAsync

Facilities to run multiple trays which connect to the same I3CLSimServer existed in the past and will be reenabled soon

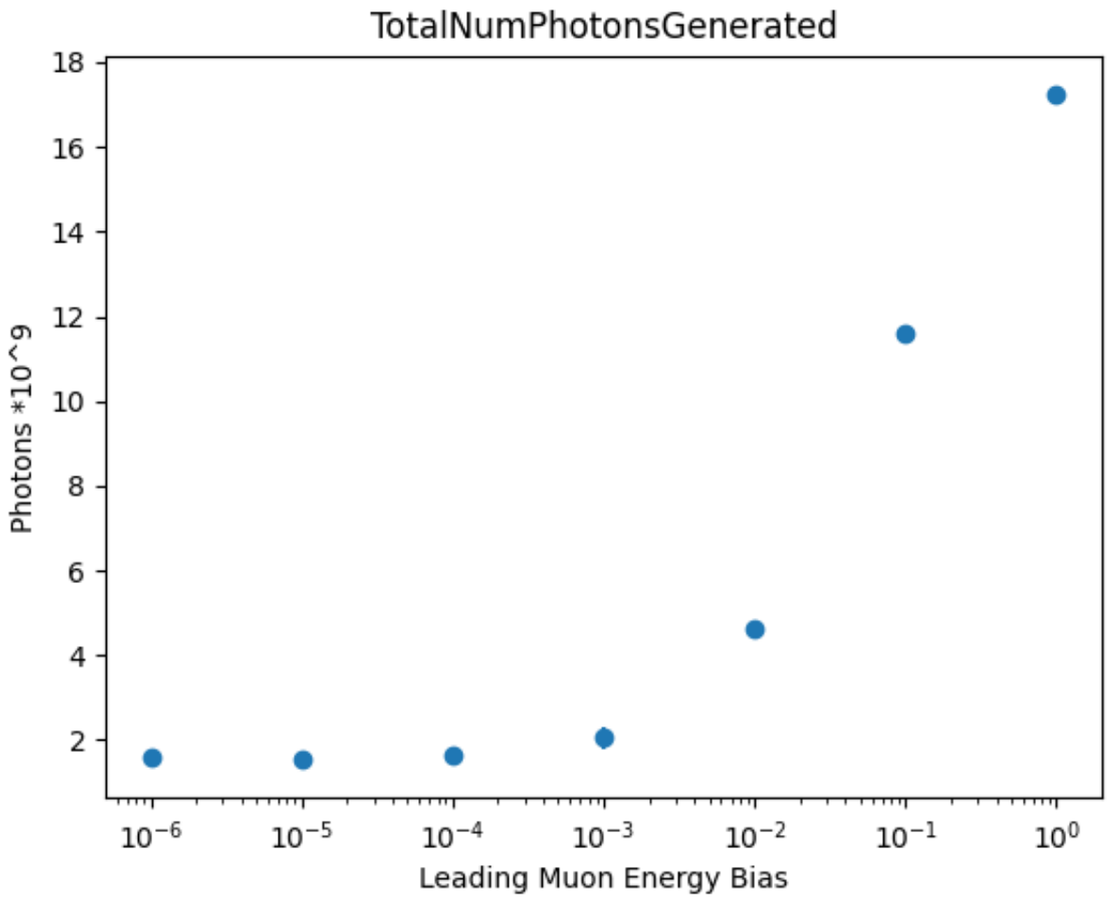
Very good agreement with baseline CORSIKA

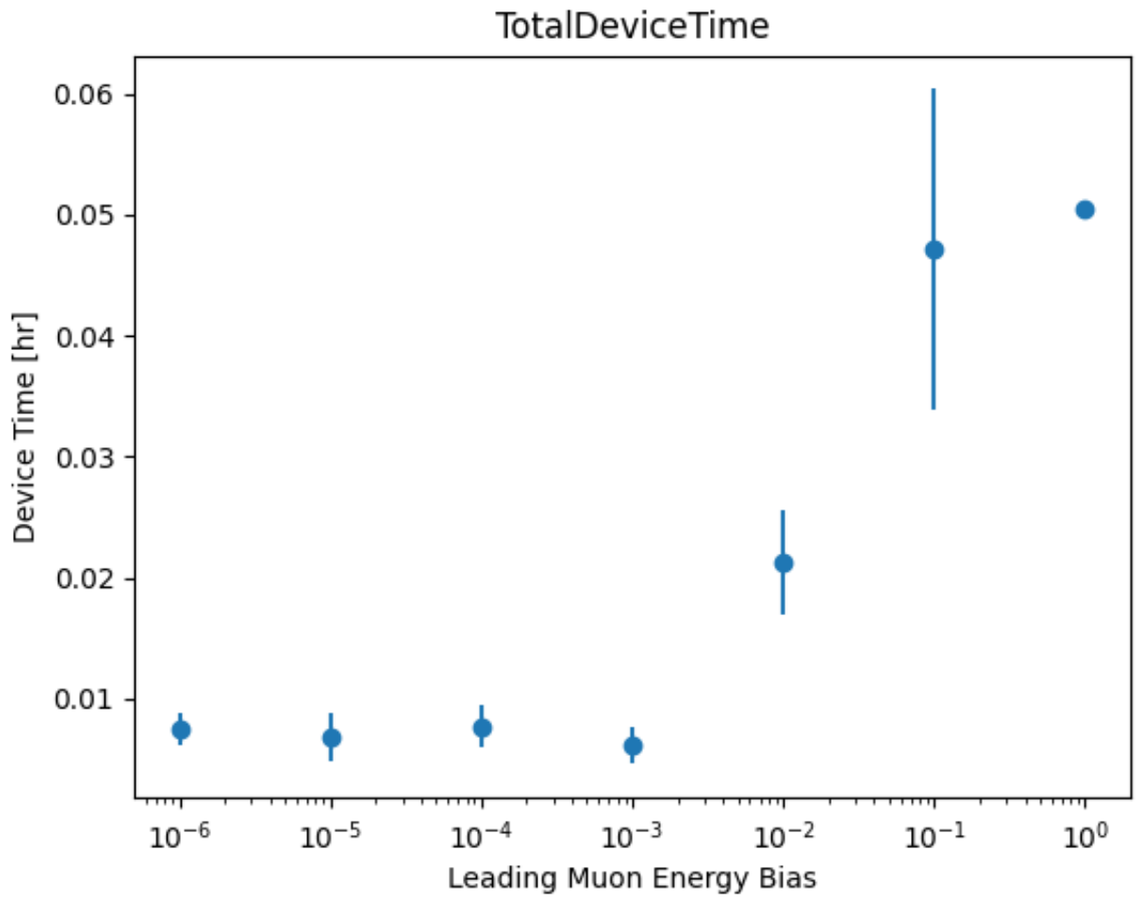




Performance

Biasing on the energy of the leading muon in the air shower results in a significant decrease of the number of photons simulated

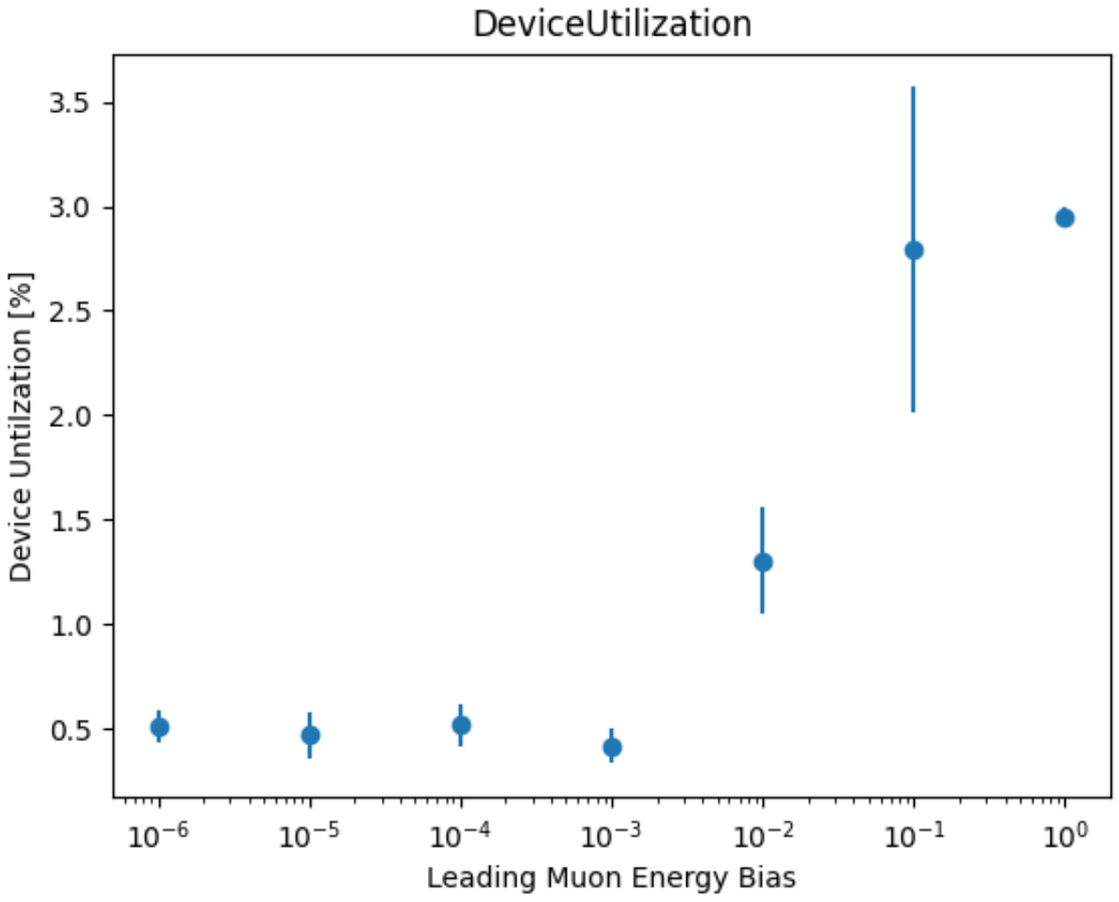


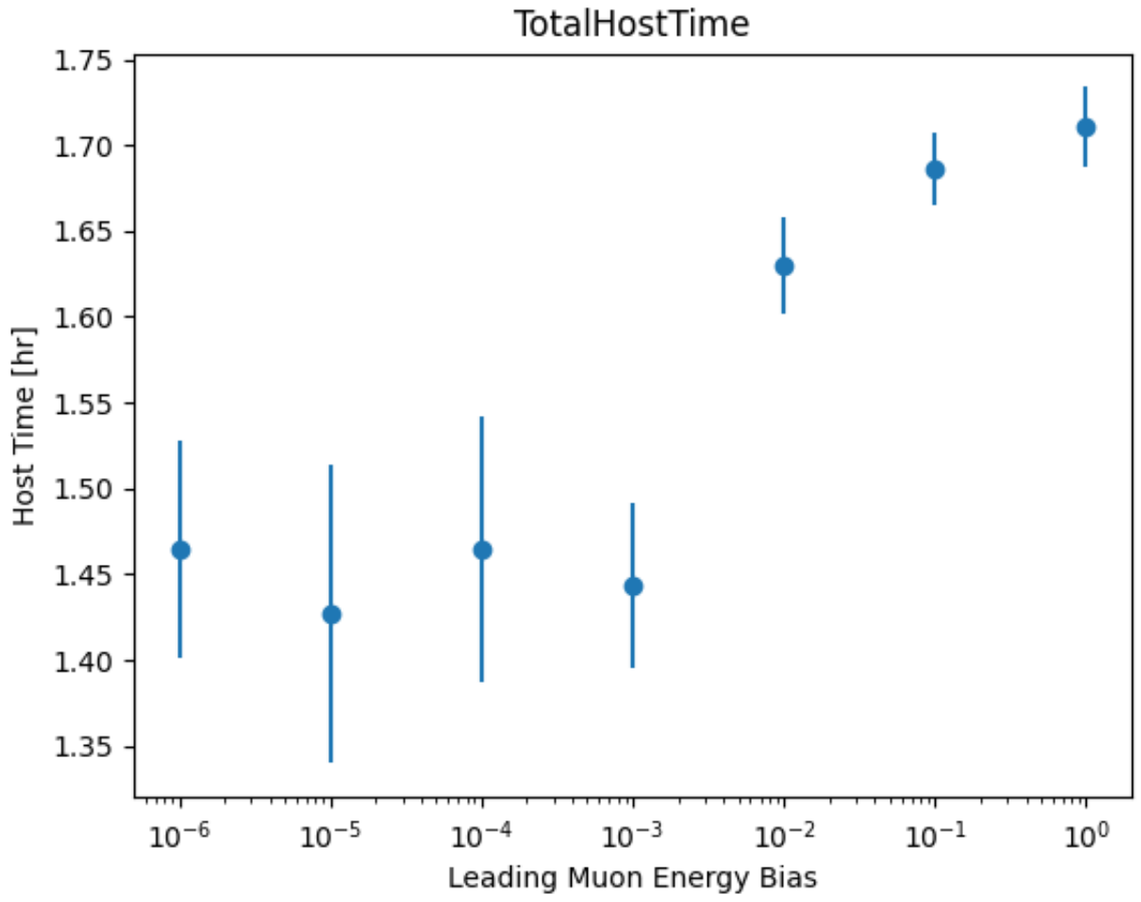


Error bars represent standard deviation of 100 jobs

Performance

Unfortunately, the GPU is severely underutilized so there is little gain from adding leading muon energy bias





Error bars represent standard deviation of 100 jobs

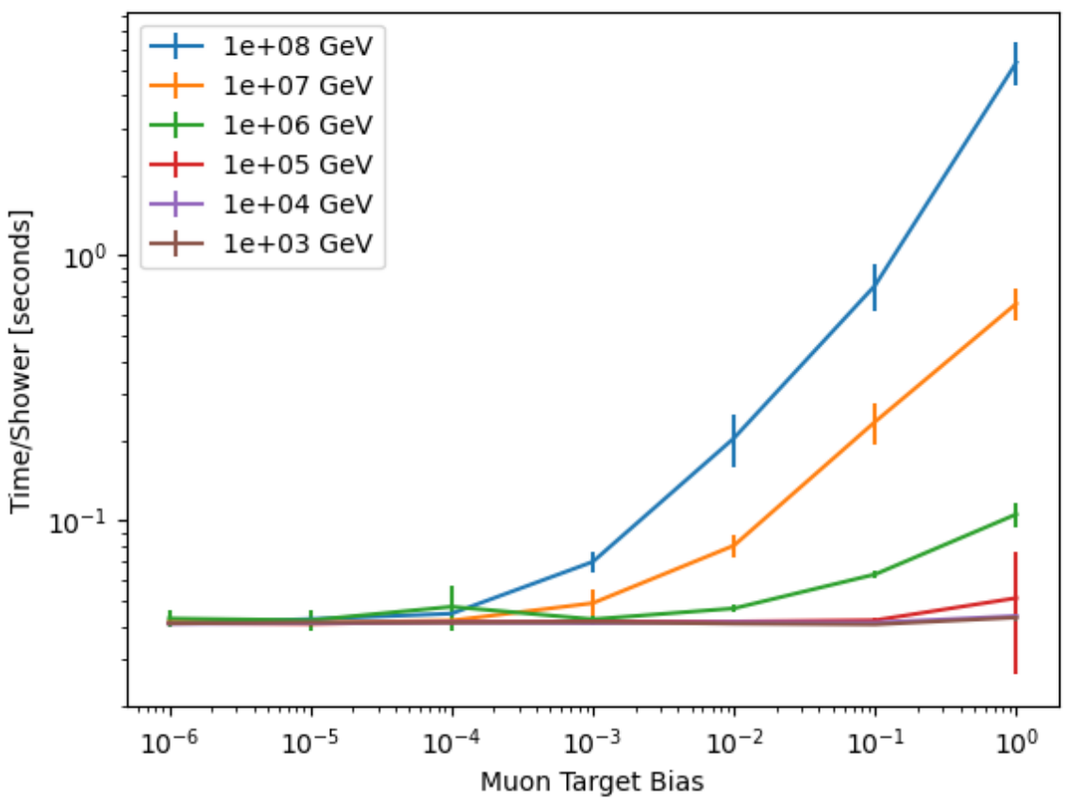
Execution Time

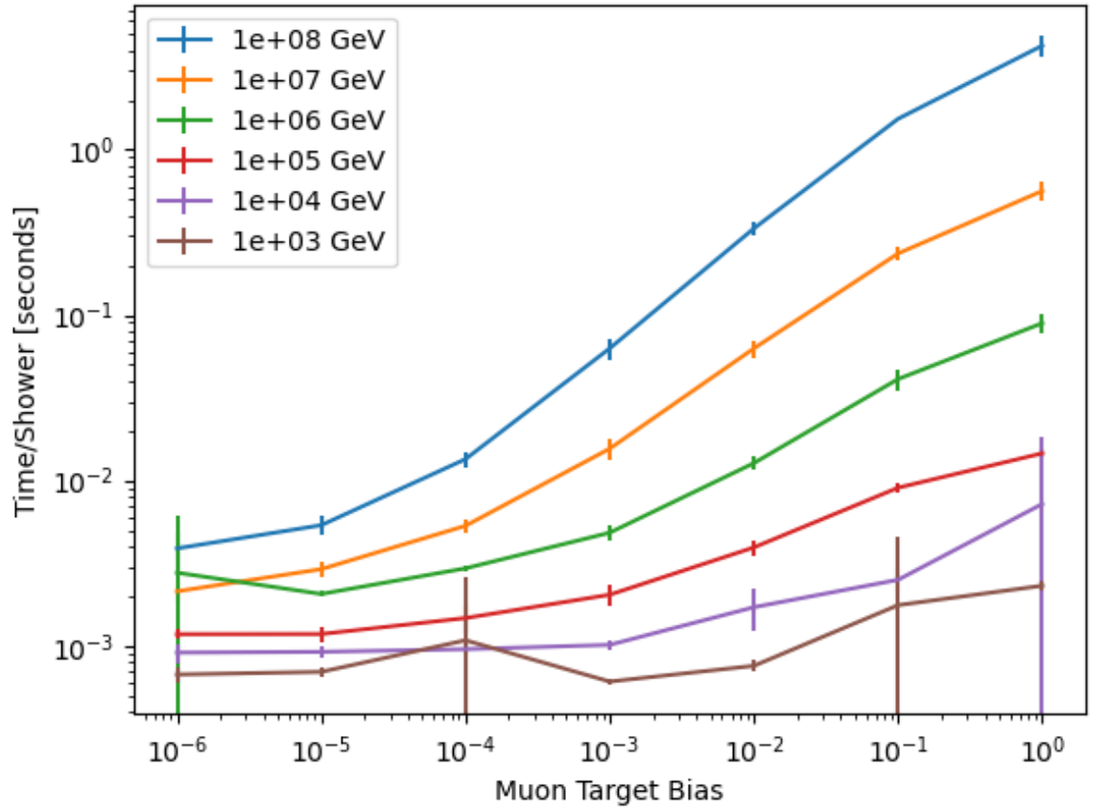
For medium energy CORSIKA primary = 3e4 to 1e6 GeV

CorsikaServer	105 min
PROPOSAL / CMC	6 min
CLSim	5 min

CORSIKA was taking much longer with CorsikaServer than when run standalone

CORSIKA Execution Time





- CorsikaService had a floor of ~40ms per shower.
- Socket was missing `TCP_NODELAY`
- With this fixed it should be possible to run the shish kabob on a single GPU node process (test are currently being run)

Weighting

- Weighting for dynamic-stack will get complicated
- `SampleImpactRay()` introduces a zenith term in the weight
- The shower biases introduce another factor that needs to be included in the weighting
- Planned future biases will need additional terms in the weight as well

Weights implementation

- Written in pure python so it can be installed on laptops without need for compiling combo
- Same basic structure as the objects in `icecube.weighting` project but I had to rewrite the insides
- Will use S-Frames so:
  - there will be no need to keep track of the number of jobs which ended up in hdf5 files
  - No need to access database to get weighting information
  - Will work with any combination of generation surfaces
- Interfaces for older CORSIKA and NuGen but since they don’t have S-Frames you need to know the number of files

Weighting Usage

```
import pandas as pd
from icecube.primary_injector import
from icecube.weighting.fluxes import GaisserH4a

pd.HDFStore(fname,'r')
wobj = PrimaryWeighter(f)

flux = GaisserH4a()
weights = wobj.get_weights(flux)

histogram(f[‘Reco’][‘zenith’],weight=weights)
```

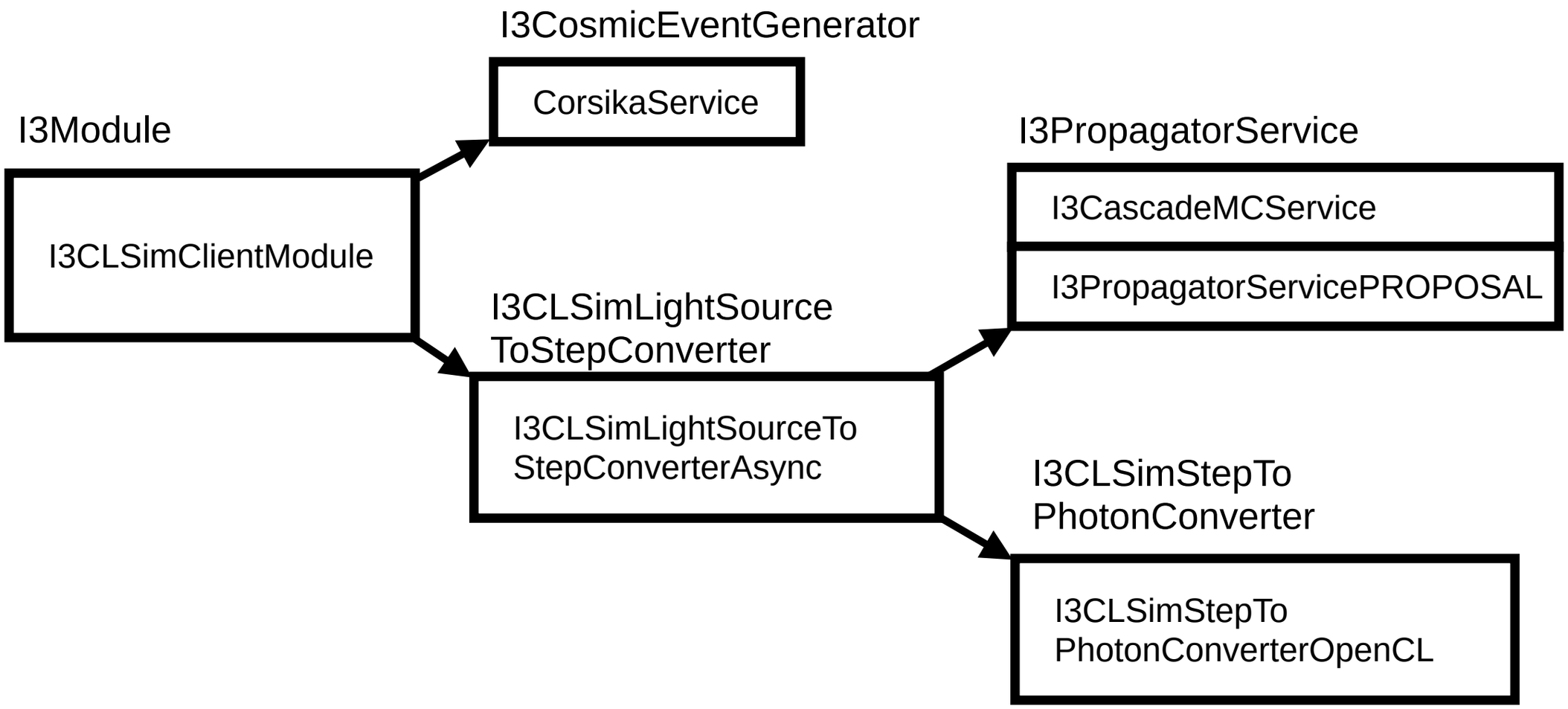
Flux Models in weighting

- The only thing needed to complete the weighting module is the flux models
- Flux models are based on numexpr and very difficult to understand
- Use the old CORSIKA particle ID’s
- I am not sure if all of the models are still in use

Issues with I3CLSim going forward

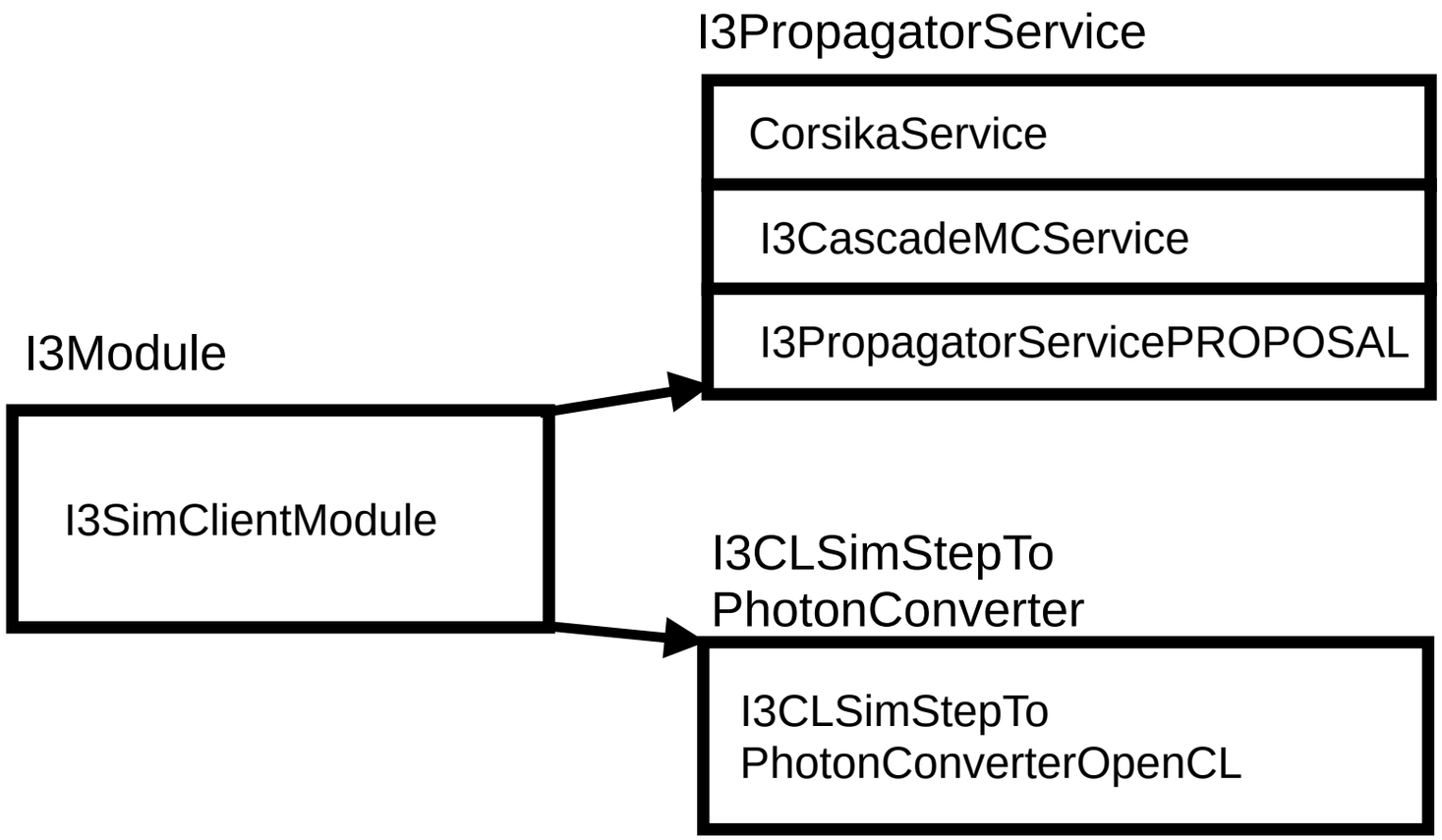
- The enums in I3Particle are poorly defined and seem to mean different things to different modules
  - For Example, what does InIce really mean?
- I3CLSimClientModule seems to have a lot of ad hoc code that is oddly specific in how it handles incoming I3MCTrees
- I anticipate problems going forward integrating other photon propagators, other detector components such as IceTop, or neutrino interactions

Currently the simulation client has A very complicate structure



There is a lot of code that is specific to propagating certain particles in both I3CLSimClientModule and I3CLSimLightSourceToStepConverterAsync which I anticipate will cause problems

I Suggest we simplify the structure of the server so that adding more components is conceptually more straight forward



Move all particle type specific code to the respective I3PropagatorServices

Current Status

- Triggered CORSIKA produces results which are identical to the reference dataset
- Performance Issues appear to be solved — runs to verify this will be performed soon
- New weighting code will be pure python and easy to use with S-Frames
- I anticipate refactoring the CLSim Server will need to be done to add IceTop/photon propagators/Neutrino interactions
- I am interested in a discussion on better definitions of enums in I3Particle/replacing with something better

If you are interested in testing triggered corsika or want to help implement any additional biasing please contact me

Also join #dynamic-stack on slack for updates