IceCube Analyses from a User Perspective

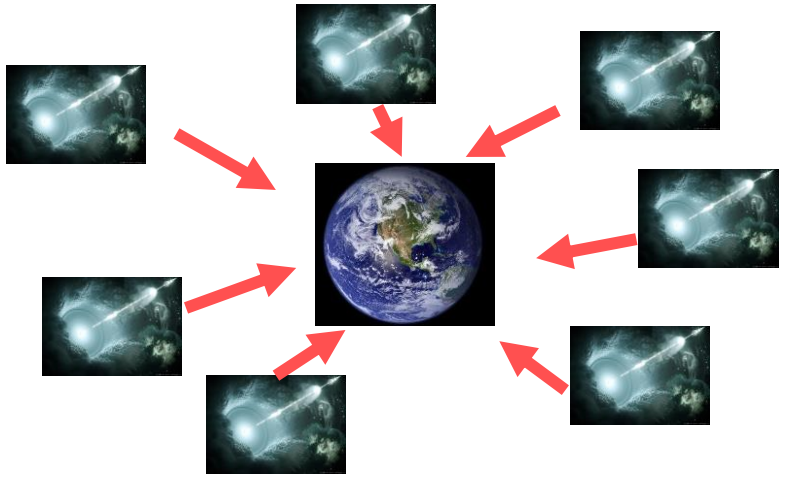
Christian Haack, TU Munich

Analyses in IceCube

- Lots of different science cases (see Benedikt‘s Talk) covering six+ orders of magnitude in neutrino energy
- Wide range of analysis techniques & statistical methods
- Data driven or heavily reliant on MC templates
- Various data processing pipelines & event selections
- Various analysis frameworks

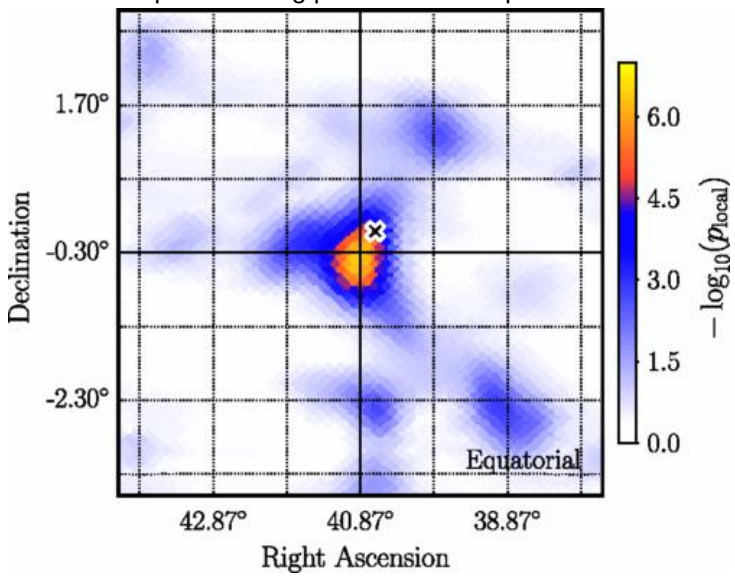
Two Examples

„Point source search“

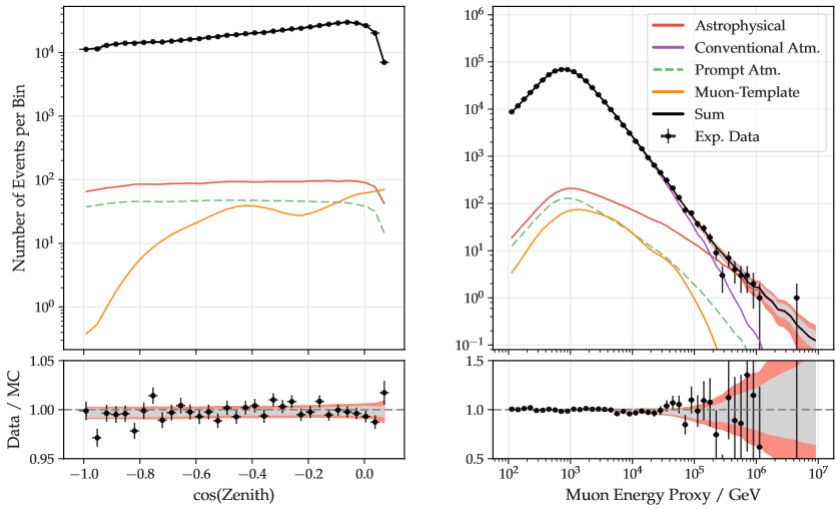


„Diffuse analysis“

https://arxiv.org/pdf/1910.08488.pdf

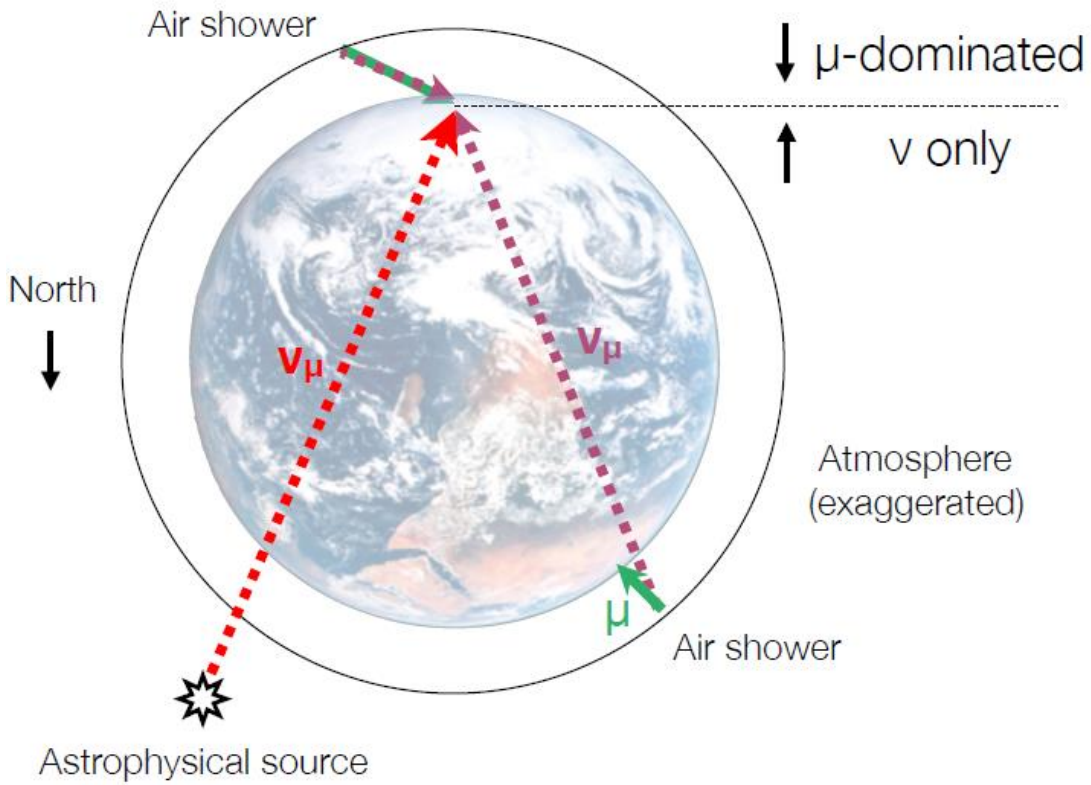


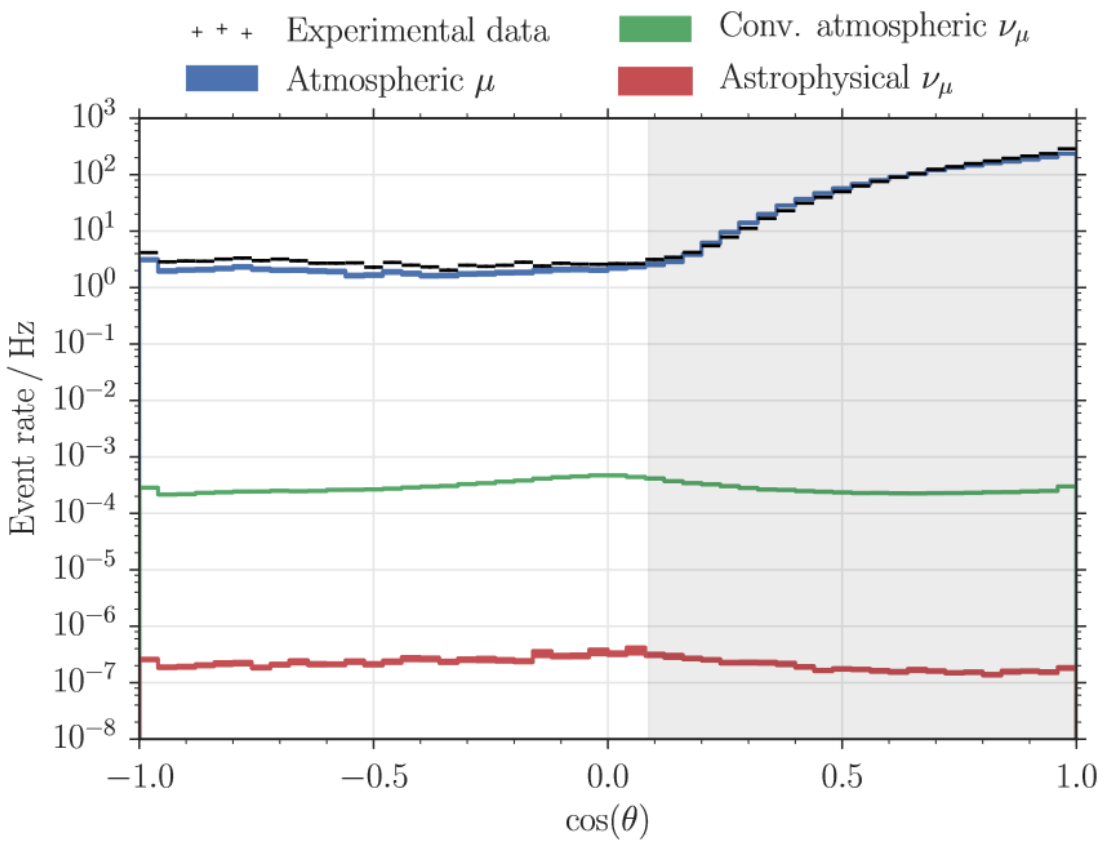
https://arxiv.org/abs/2111.10299



The Challenge

3kHz trigger rate (atmospheric BG), ~100s astrophysical neutrinos per year

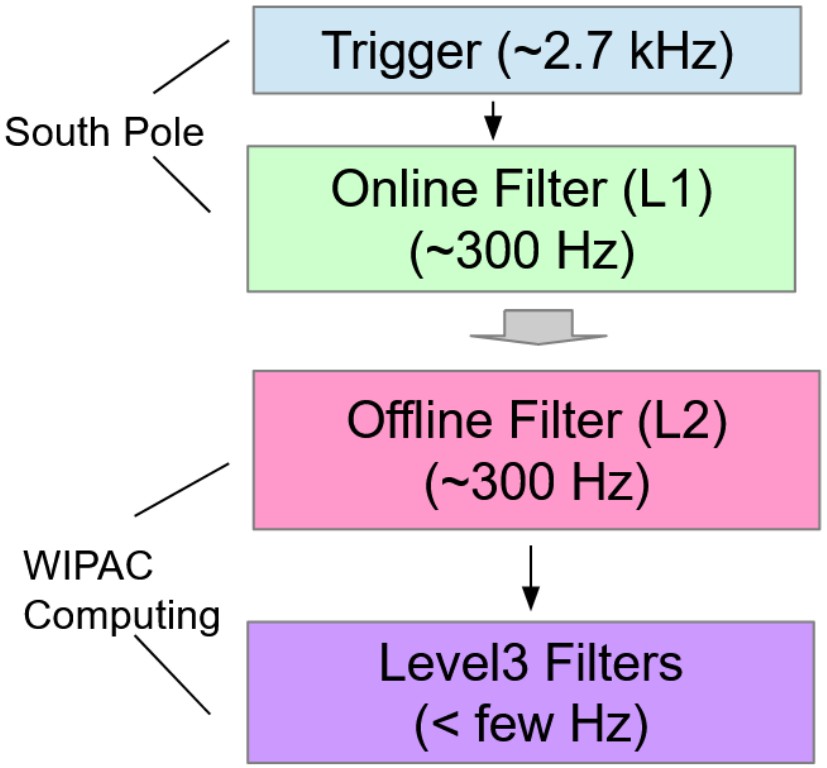




(b) Reconstructed zenith angle.

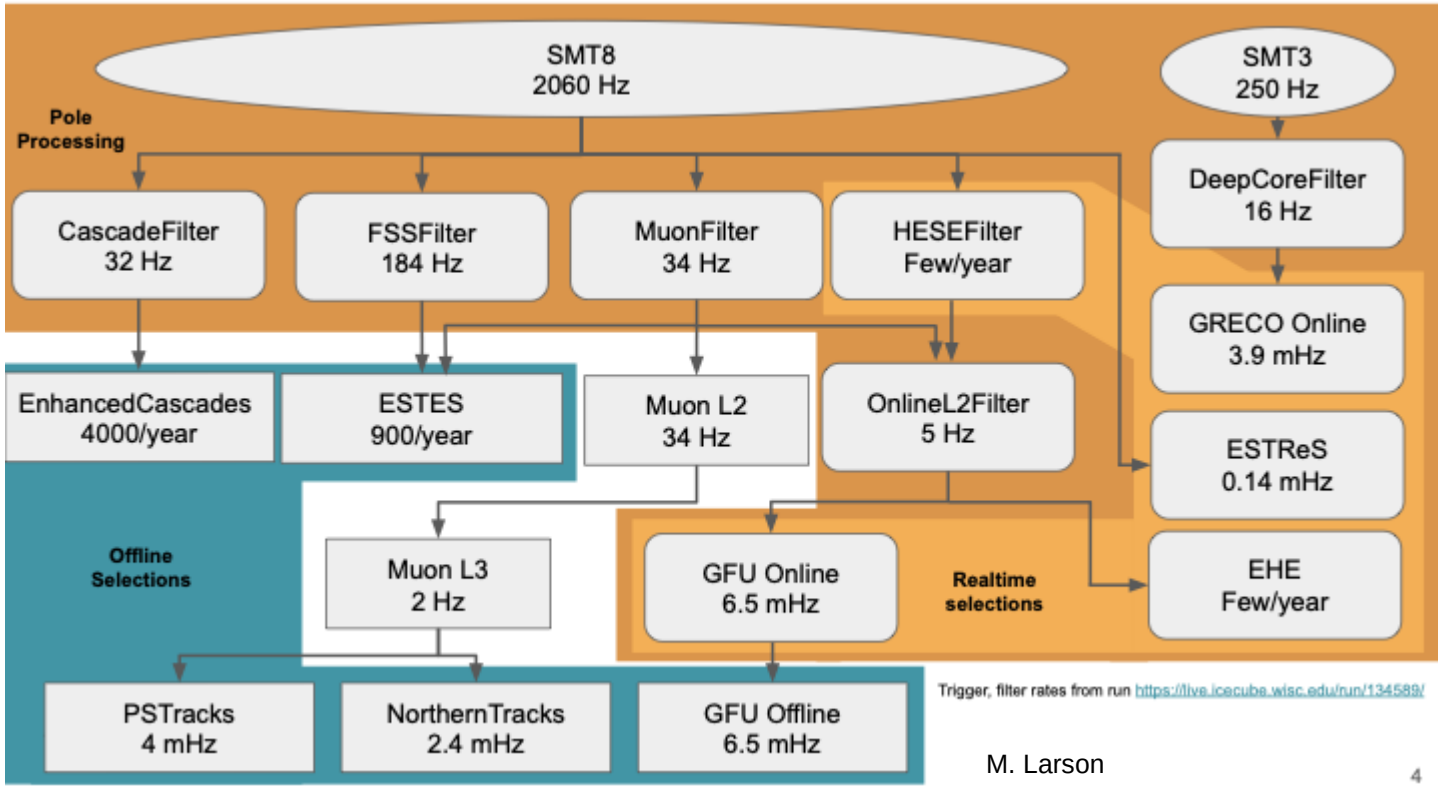
IceCube Filtering Chain

Existing Model



Naoko Kurahashi Neilson

Example: Full chain for point source analyses



M. Larson

4

IceCube Level2 Dependency Graph

- System has evolved over 10+ years of continuous operation
- Accumulation of technical debt: Difficult for new useres to get involved
- Unsolved Problem: How to convince PI‘s that it‘s worth investing into processing rather than focusing only on analysis?
- Currently trying to overhaul low-level processing: Use ML as early as possible. But: Needs a fairly large amount of GPUs (which we also need for simulation & processing of simulation)

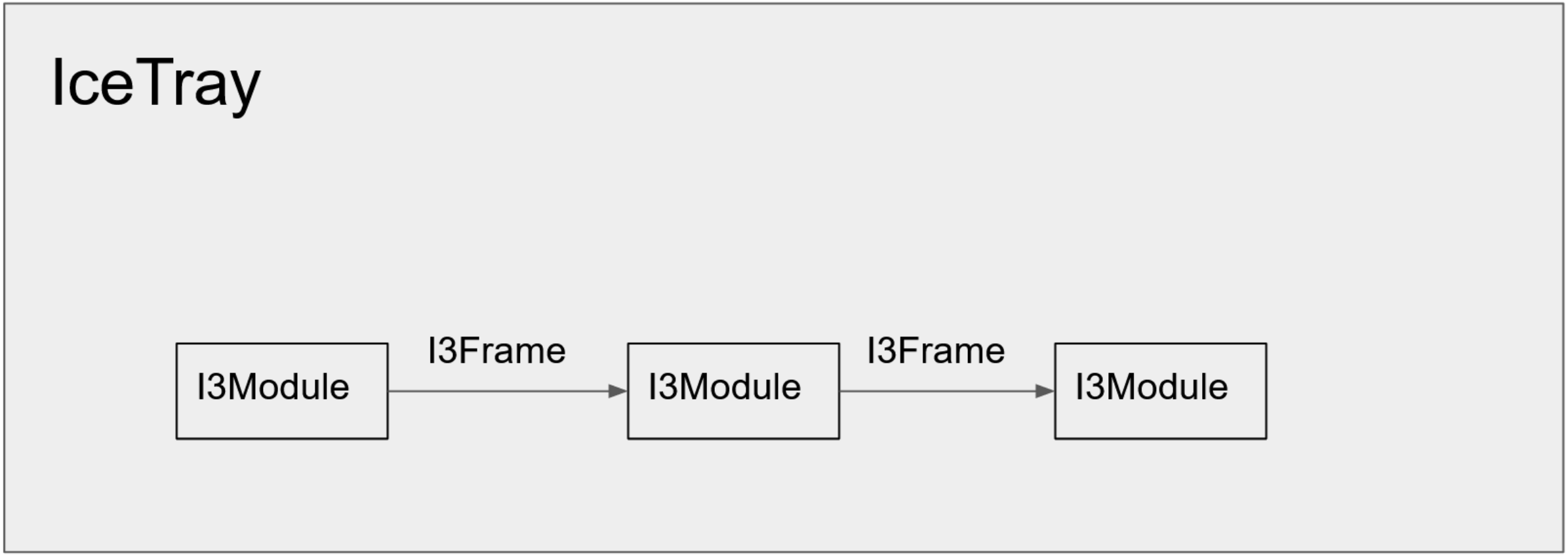
Event Selections

- Event selections largely mature (not much new development for standard channels)
- Essentially all event selection rely on ML to select signal: BDTs, CNNs
- Recently a bit of development in utilizing GNNs (better suited for IceCube‘s hetereogenous detector)
- Final level selections can be (and a few have been) implemented into centralized processing infrastructure – but a lot of selections are still processed by users

IceCube Processing Framework: IceTray

- Written in C++, pybindings (boost::python) for almost everything
- Core object: I3Frame
  - Base unit of the IceCube data stream (one event per Frame)
  - Essentially a container for Key – Value pairs, stores raw data, reconstructions, …
  - Processing on serial sequence of I3Frames
  - Each frame (and objects contained within) can have a „stream“ which allows control over lifetime of objects during processing: E.g. objects in the „Calibration“ stream will be accesible by all later frames until a new Calibration frame is encountered
  - Serialization (fork of boost::serialization) into „I3Files“
- Users build I3Trays (a collection of I3Modules) that manipulate I3Frames in series

IceCube Processing Framework: IceTray



Soon open source: https://github.com/icecube/icetray-public

IceTray Workflow

- JupyterHub: Containers with pre-compiled versions of IceTray & standard python tools (numpy, scipy, …)
- data, simulation available on various processing levels (l2, l3, final level) in central filesystem
  - Accessible from jupyterhub, (interactive-) compute nodes, and local condor cluster (4k cpus, 100s GPUs)

[18]:
```
from icecube import icetray, dataclasses, dataio
from I3Tray import *
def lambda_module(frame, arg1):
    frame["NewKey"] = dataclasses.I3Double(arg1)

tray = I3Tray()
tray.Add("I3Reader", Filename="/cvmfs/icecube.opensciencegrid.org/data/i3-test-data/nugen_numu_ic80_dc6.002488.000000.processed.i3.gz")
tray.Add(lambda_module, arg1=1, Streams=[icetray.I3Frame.Physics]) # apply module only to "Physics" frames
tray.Execute(1)
```

IceTray: Pros & Cons

Pros:

- Scripting in python allows for plug&play – setting up a processing pipeline is straight forward
  - E.g., probe parameter space of reconstruction algorithms with simple for loop in python
- Complex algorithms can be implemented in C++
- A large variety of datastructures implemented in C++
- Stable and mature (processing data for 15+ years)
- Converters for tabular dataformats (can e.g. export to HDF5 or root files)

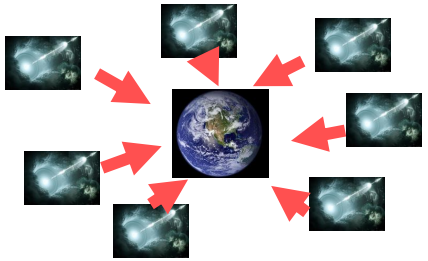
Cons:

- New datatypes require implementation in C++ (+ pybindings + converters)
- Serial processing paradigm imposes some hurdles when trying to efficiently use accelerators (GPUs)
  - We typically bake a client / server structure on top
- Datastructures & processing model were designed before the advent of ML -> many ML users break out of IceTray as soon as possible
- Stream model not always ideal (data can be hierarchical e.g. event splitting) but stream model enforces serial data stream
- Plug and play tends to accumulate technical debt

IceCube Analysis Workflow

- No centralized solution as for processing
- Users take care of extracting data from I3Files and converting to a file format of their choice
- Analyses typically use only a handful of observables -> small datasets (<GB)
- Users take care of computation (job submission etc)
  - For example: Split up a profile likelihood scan and compute one (or a handful) of points per batch job
- Can use similar infrastructure as processing (multiple sites via condor glideins)
- Cvmfs available for processing: Can store analysis scripts (and small datasets) in cvmfs user directory

Analysis Frameworks



Diffuse Analysis

- Forward folding poissonian template fit
- Bin-wise expectation from MC: Updating model / nuisance parameters changes event-weight
- Frequentist: profile likelihood to obtain confidence contours on astrophysical neutrino flux parameters
- More or less standard tool: NNMFit (C.H. et al.)
- Constructs computation graph with aesara
- https://github.com/icecube/NNMFit (soon public)
- For now check out docs

Point Source Analysis

- Unbinned likelihood: test fine grid on sky for deviation from isotropy
- Historically only signal expectation from MC. BG hypothesis realisations computed from scrambled data. Recently also analyses with BG from MC
- Frequentist: p-value for isotropy at each point in sky, find hot spot and correct for multiple testing. Confidence contours for flux parameters of sources
- Multiple tools: E.g. skyllh https://github.com/icecube/skyllh

Typically each tool reimplements statistics backend => Would be nice to use standardized implementations (pyhf?)

Analyses: The Good & The Bad

Good

- Mature analyses have standardized tools: Moving away from 1 code/1 analysis to 1 code/many analyses
- IceCube computing infrastructure allows convenient access to data & simulation
- JupyterHub, compute nodes & cvmfs make prototyping easy
- Users have large computing resources at their disposal
- python

Bad

- Generally (analysis) frameworks have poor test coverage (if any)
- Little no utilization of CI or containers
- Users typically write their own batch processing scripts (no integration of e.g. dask)
- No common backend for statistical analysis
- While technically having open source licenses, not all tools are public
- Little utilization of probabilistic programming
- We live in a bubble: Almost no collaboration on tools with other experiments
- python

Wishlist

- More communication outside of our bubble
- Collaboration on analysis frameworks
- Training in „modern“ coding tools / paradigms (unit tests, CI, containers)
- Investment into modern (ML-aided) statistics: Can we benefit from probabilistic and differentiable programming?

Systematic Uncertainties

- Icemodel & PMT calibration are main (detector-related) systematic uncertainties
- Recently started using the „Snowstorm“ technique to propagate uncertainties to final analysis level: https://arxiv.org/abs/1909.01530
- Idea:
  - In simulation, sample a set of parameters $\vec{\chi}$ that perturbe the icemodel & detector acceptance around baseline
  - Process a small batch of events with parameter set
  - In analysis compute gradient of event weight wrt. perturbation and calculate taylor expansion (first / second order) to include systematic nuisance parameter
- For each simulated event $\vec{x} \rightarrow \vec{x}(\vec{\chi})$: Have to recompute expectation (e.g.