Introduction to `TAMBOSim` Software

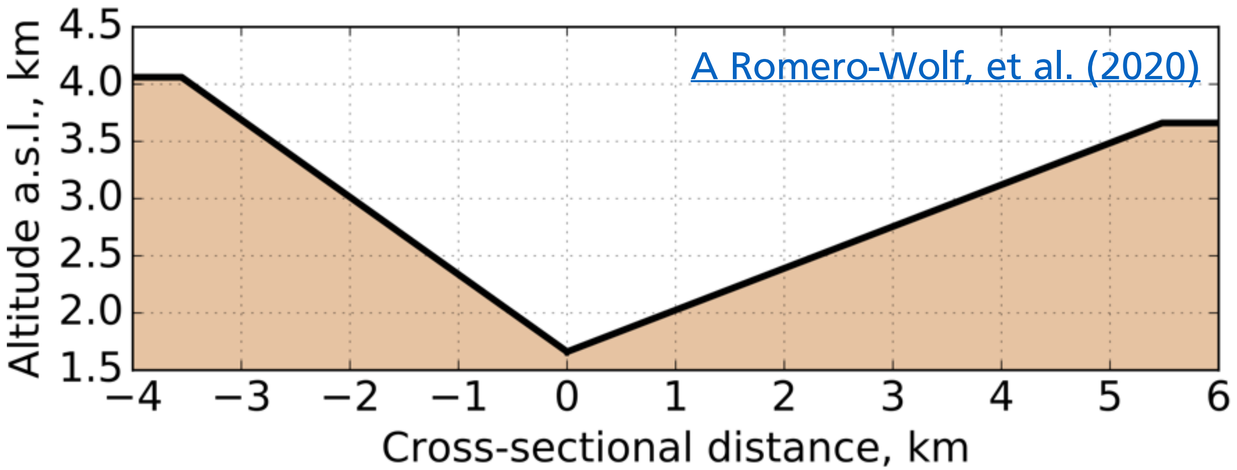
Jeffrey Lazar
TAMBO Workshop
Lima, Peru
20 Oct., 2025

Outline

- Simulation overview
- `TAMBOSim` main results
- `TAMBOSim`, a practical introduction

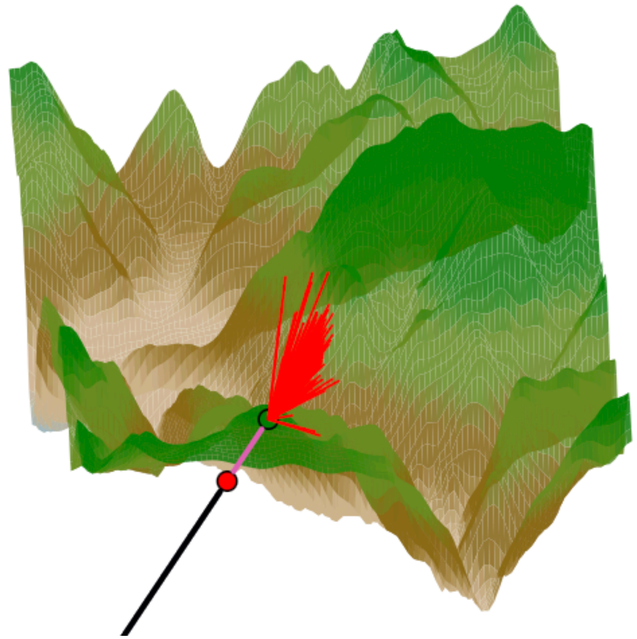
Moving Past Initial Estimation

Initial Calculation



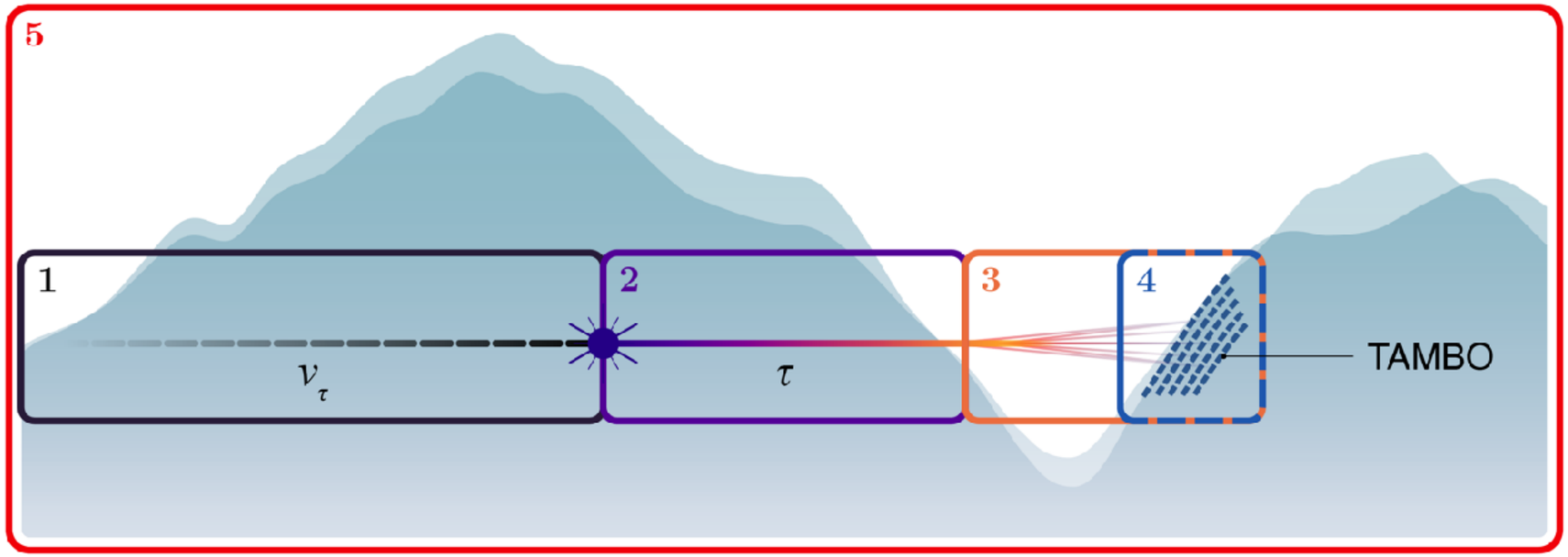
- Simplified geometry
- No treatment of $\tau^{\pm}$ energy losses
- Approximation of air-shower physics

Updated Simulation



- Flexible geometry
- Full treatment of $\tau^{\pm}$ energy losses
- Air-shower simulation with CORSIKA 8

Simulation Overview



1. **Initial neutrino injection**
   Select initial neutrino properties, i.e. energy, direction, interaction vertex, etc...

2. **Charged lepton propagation**
   Propagate outgoing charged lepton, accounting for energy losses and decay, to find decay point.

3. **Air-shower simulation**
   Model shower development from lepton decay.

4. **Detector response**
   Simulate internal hardware to model what we will see.

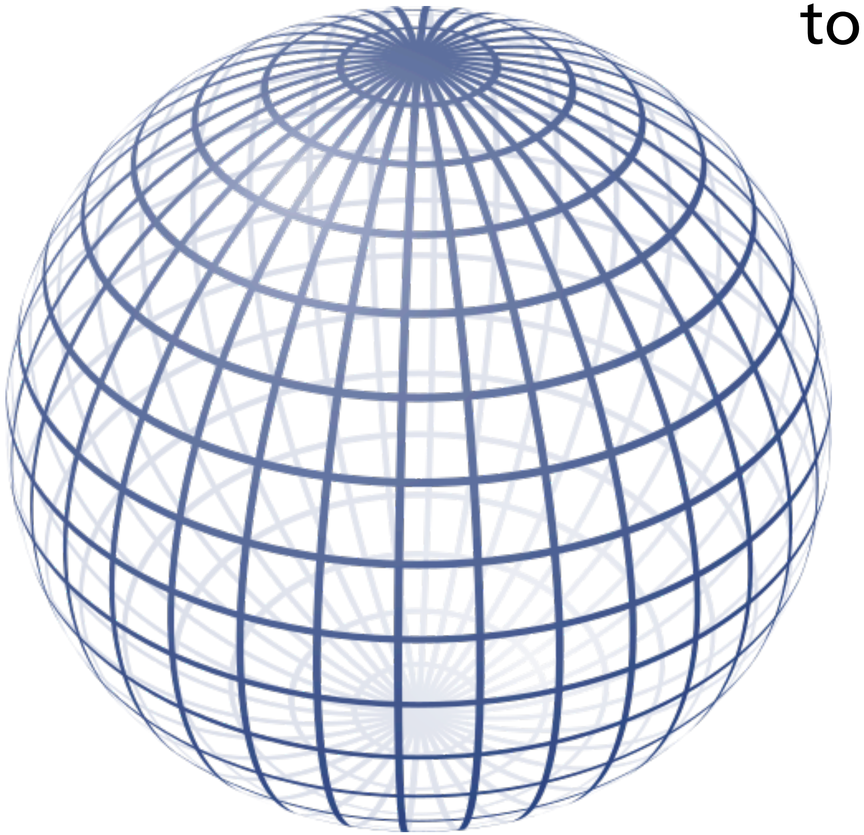
5. **Event weighting**
   Remove unphysical remnants from selection of initial neutrino properties.

Simulation regions

- Configurable simulation region is a box that sits on top of the Earth
  - Box length and width defined by input spline and extends 5km below and 10 km above the center of the detector
  - The Earth composition is taken from the PREM

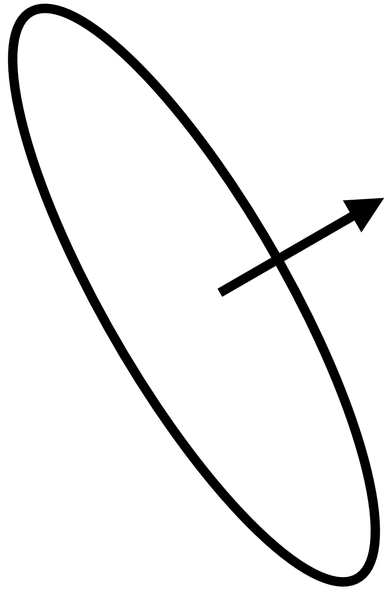
Neutrino Injection

Select initial direction uniformly on sphere

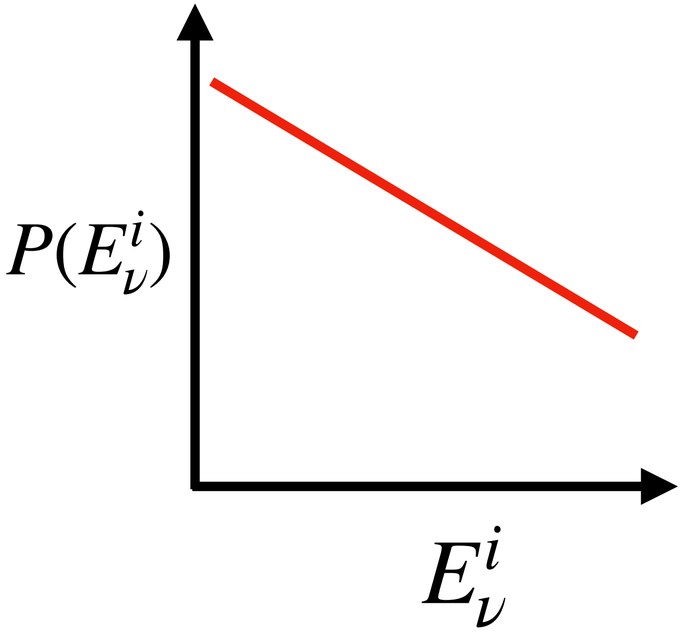


Wikimedia commons

Sample point of closest approach uniformly on disk perpendicular to direction



Sample initial neutrino energy from power law

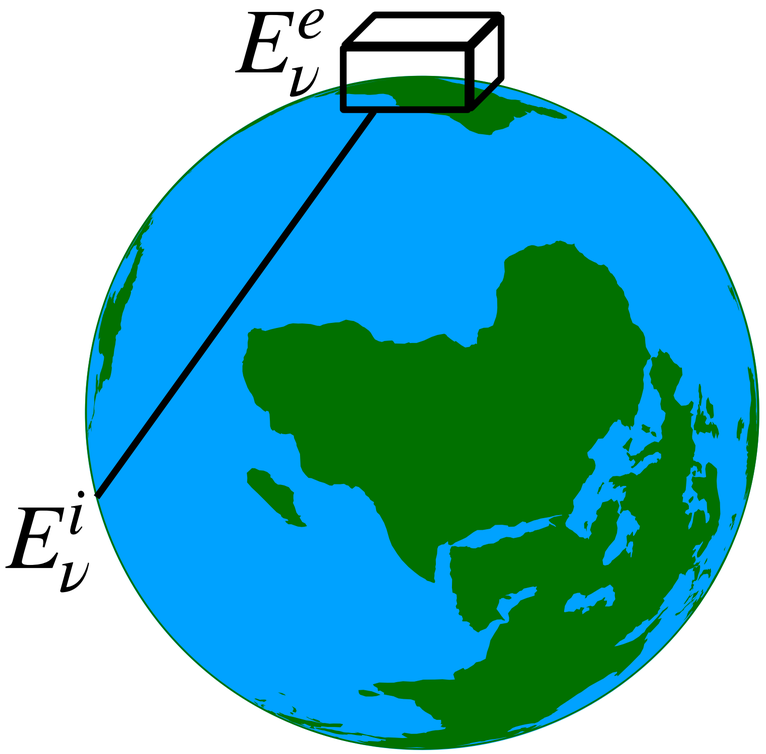


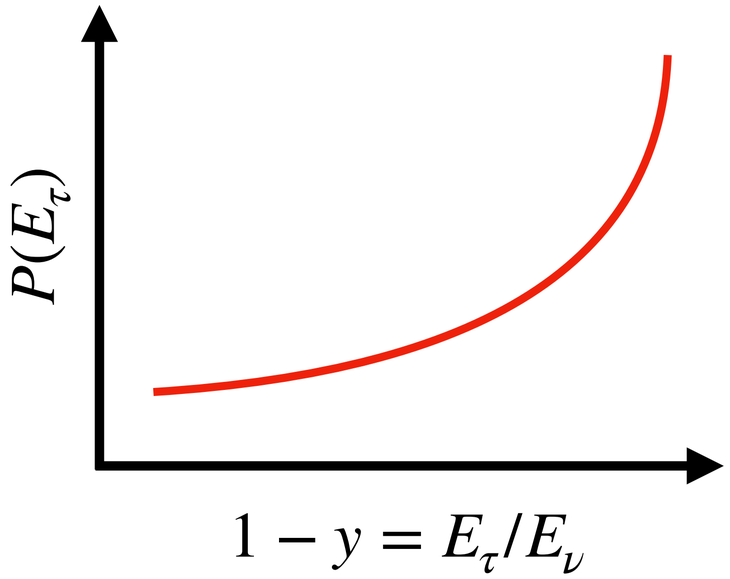
Neutrino Injection

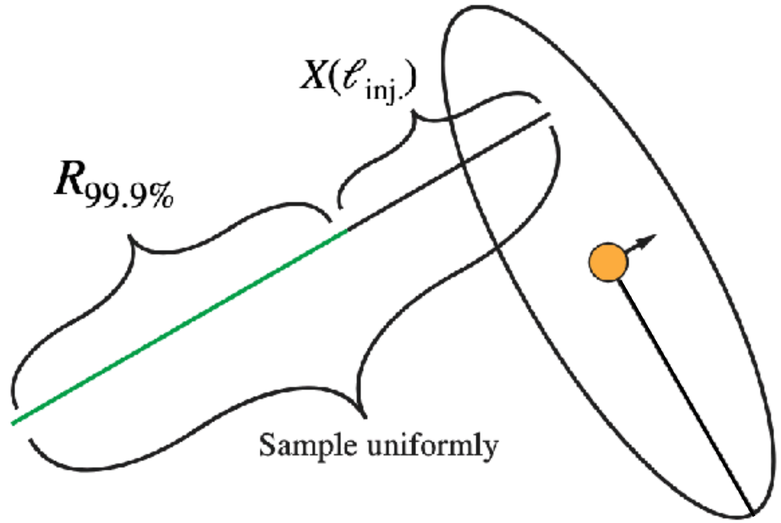
Use `TauRunner` to compute neutrino energy at TAMBO simulation region

Sample outgoing tau lepton energy using inelasticity distributions

Find interaction vertex by sampling uniformly in column depth







Neutrino Injection

- Many parameters can be adjusted in this, including:
  - angular ranges;
  - radius of the disk;
  - minimum energy, maximum energy, and spectral index;
  - size of simulation region; and
  - cross section model
- **Italicized parameters are critical; incorrect values will lead to undercounting events or inefficient simulation**
- N.B.: A cross section model is baked into the simulation at this point and cannot be changed via reweighting*

Neutrino Injection

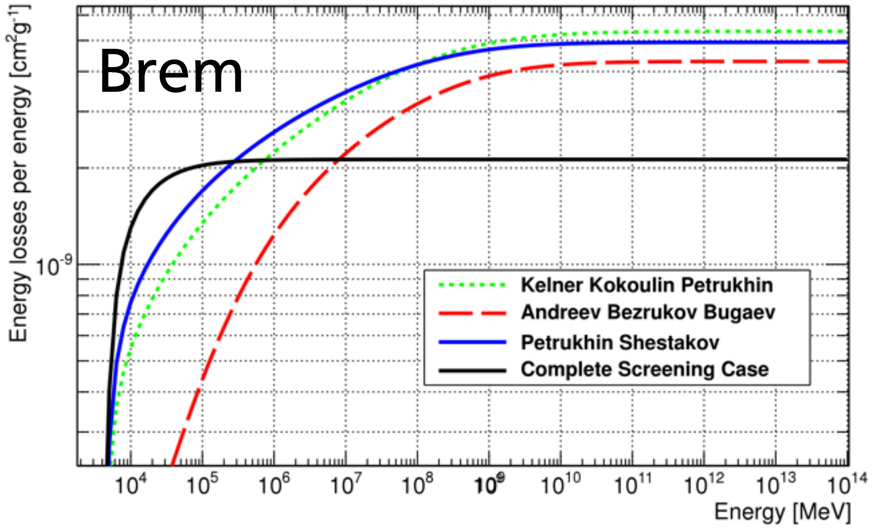
- Oooof!

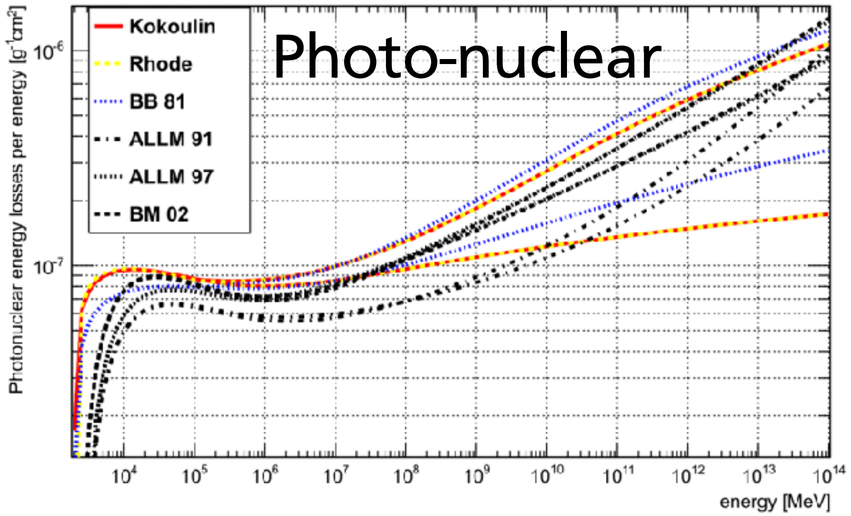
Neutrino Injection

- Oooof!
- Now we have the energy and direction of the initial neutrino and charged tau lepton and the interaction vertex so we can move on!

Charged Lepton Propagation

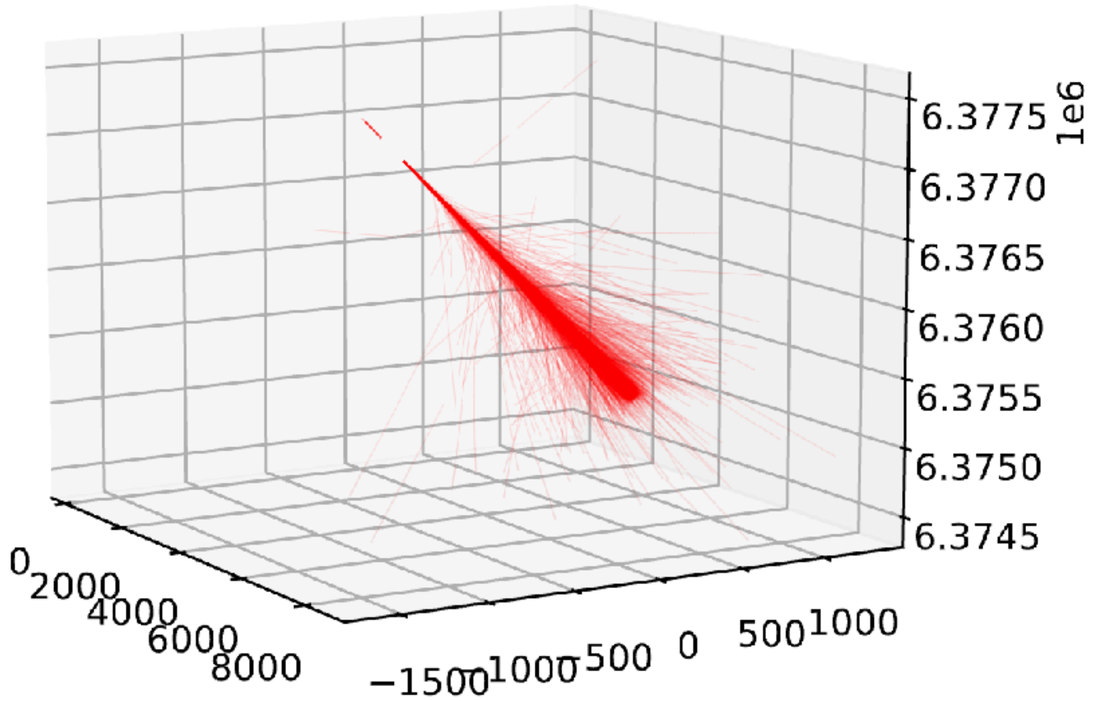
- We use the PROPOSAL package to propagate charged leptons from the interaction vertex to the decay
- This follows physical rates and so no reweighting is possible





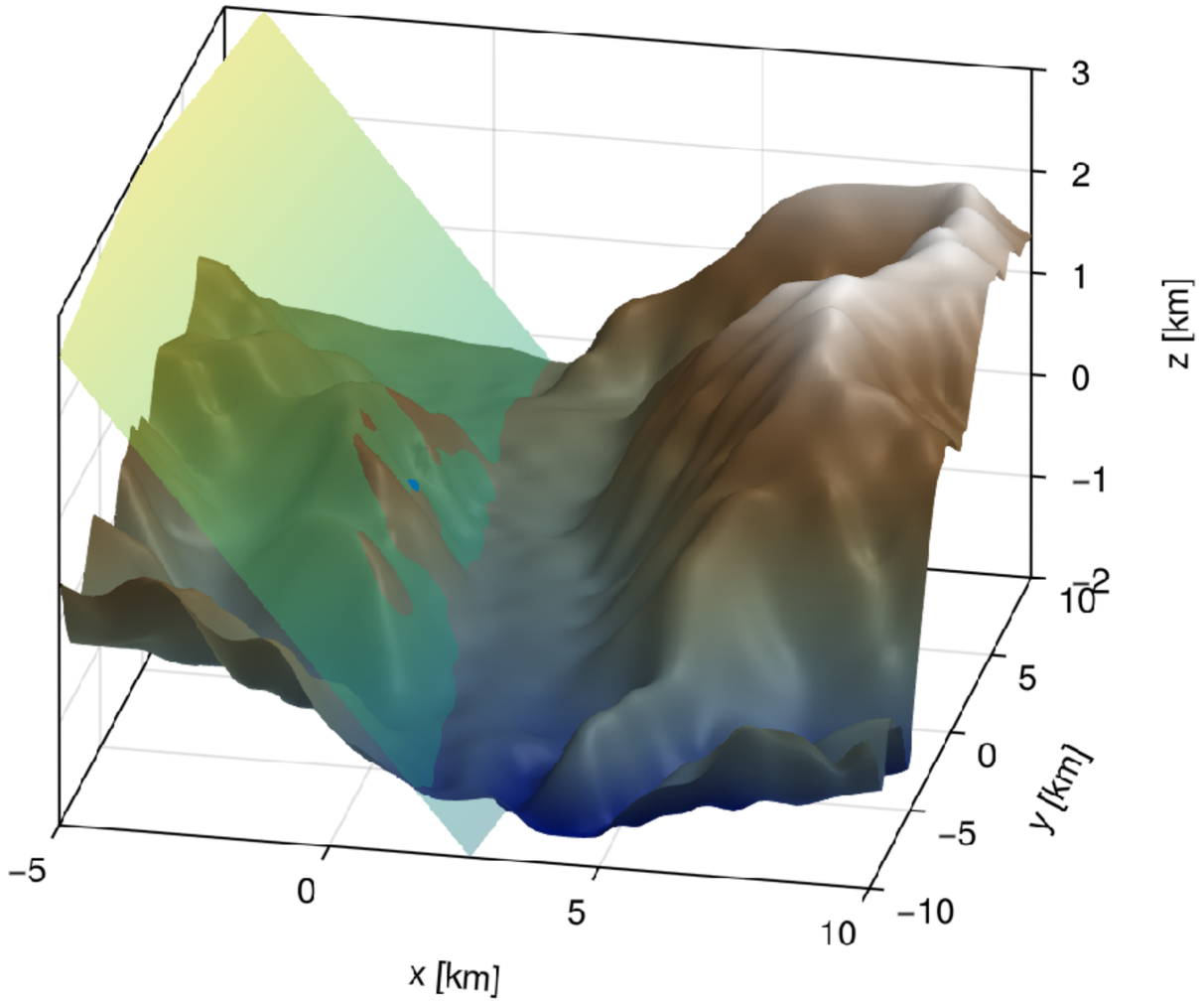
Extensive Air-Shower Development

- We use the CORSIKA8 package to simulate the extensive air showers from tau lepton decay products
- Once again, this follows physical rates
- Energy cut settings play a critical role in getting our rates correct. This is an ongoing area of study



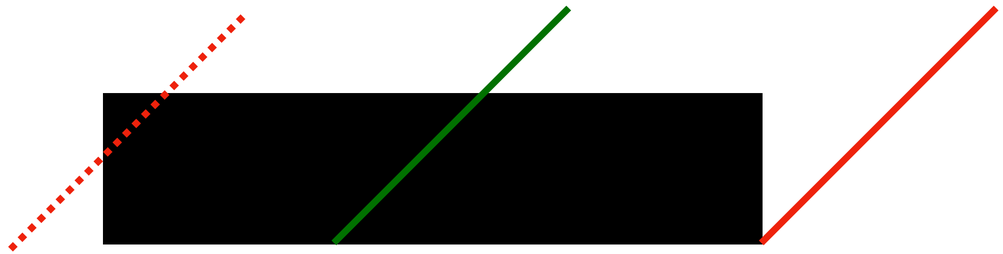
Extensive Air-Shower Development

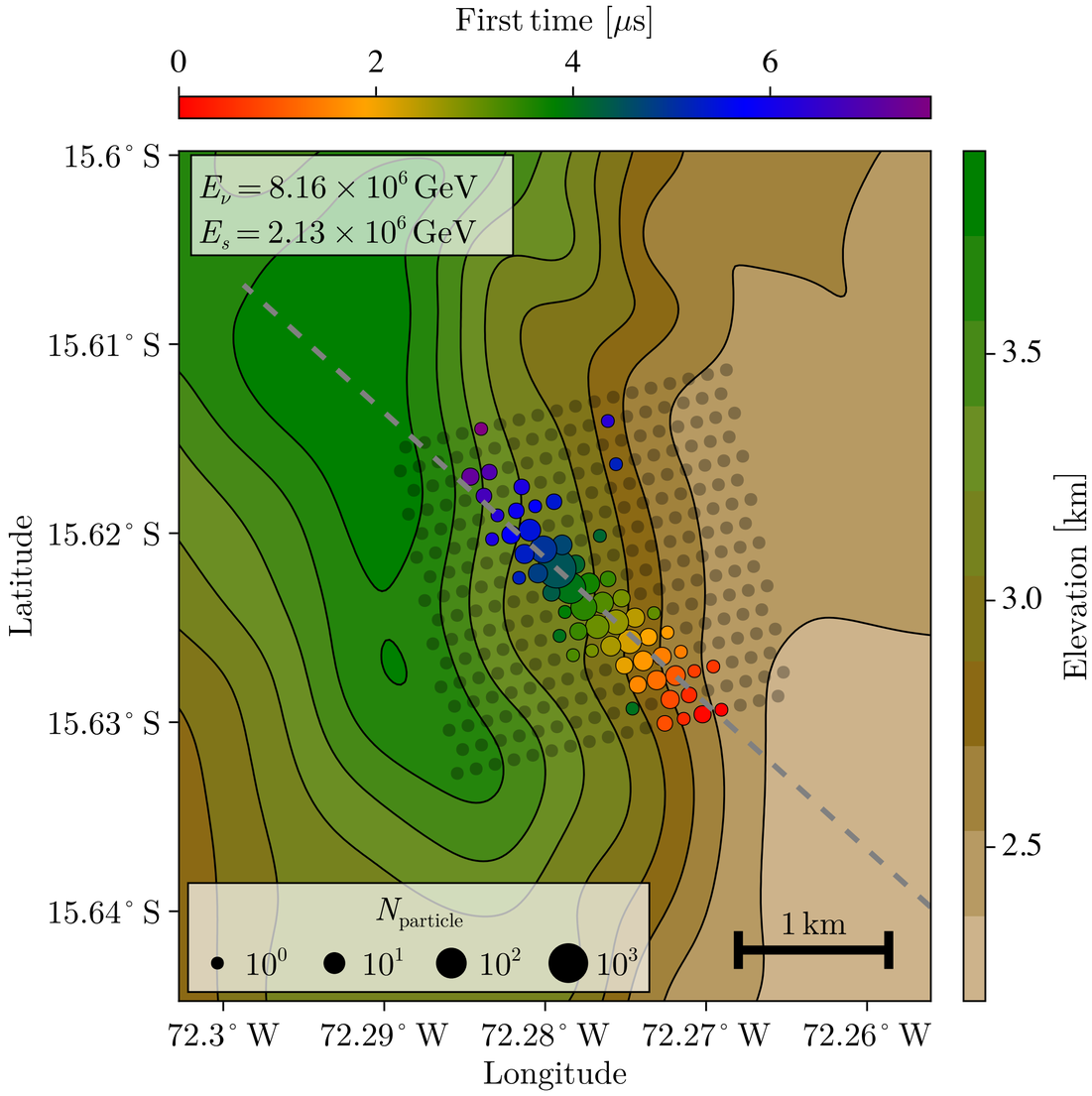
- `CORSIKA` output is read out onto an inclined plane that approximates the mountain face
- This approximation only holds in limited region around center of detector
- More sophisticated treatments are being considered, but are unlikely to be included in simulation for awhile



Detector Response

- Particle are considered to have hit a detection module if the point at which they intersect the plane lies within module footprint
- This neglects 3D nature of module, which may slightly undercount
- Realistic module triggering is currently being fully implemented





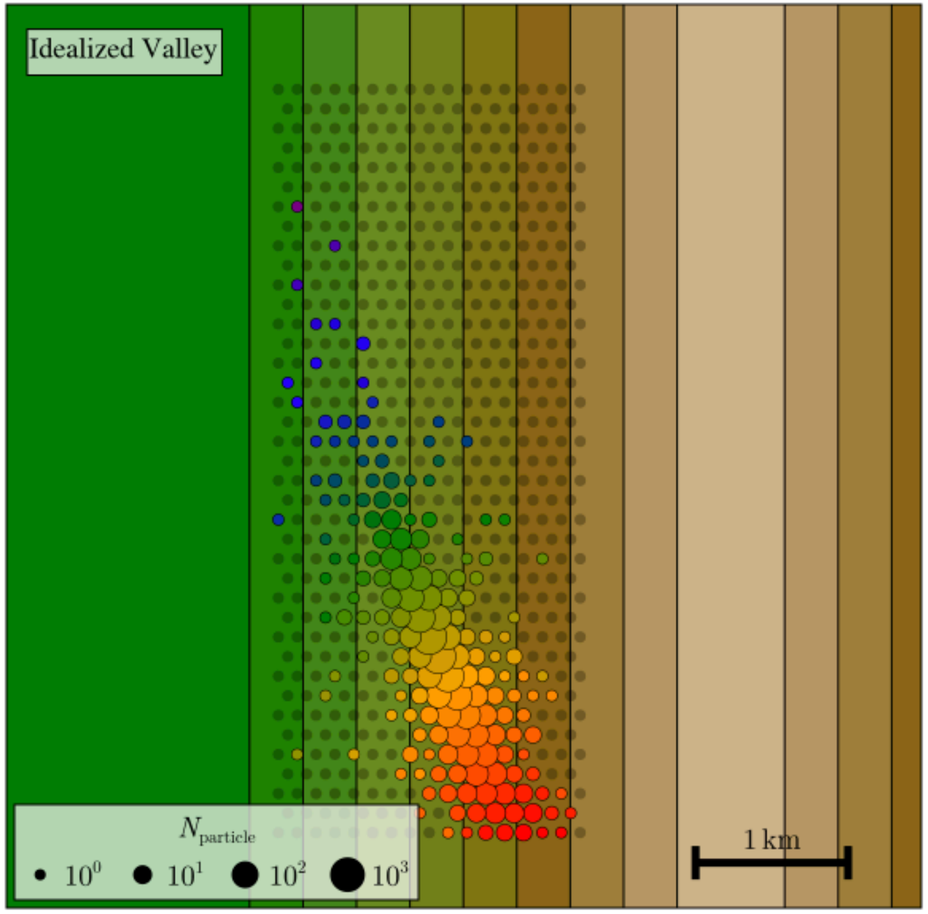
Event Weighting

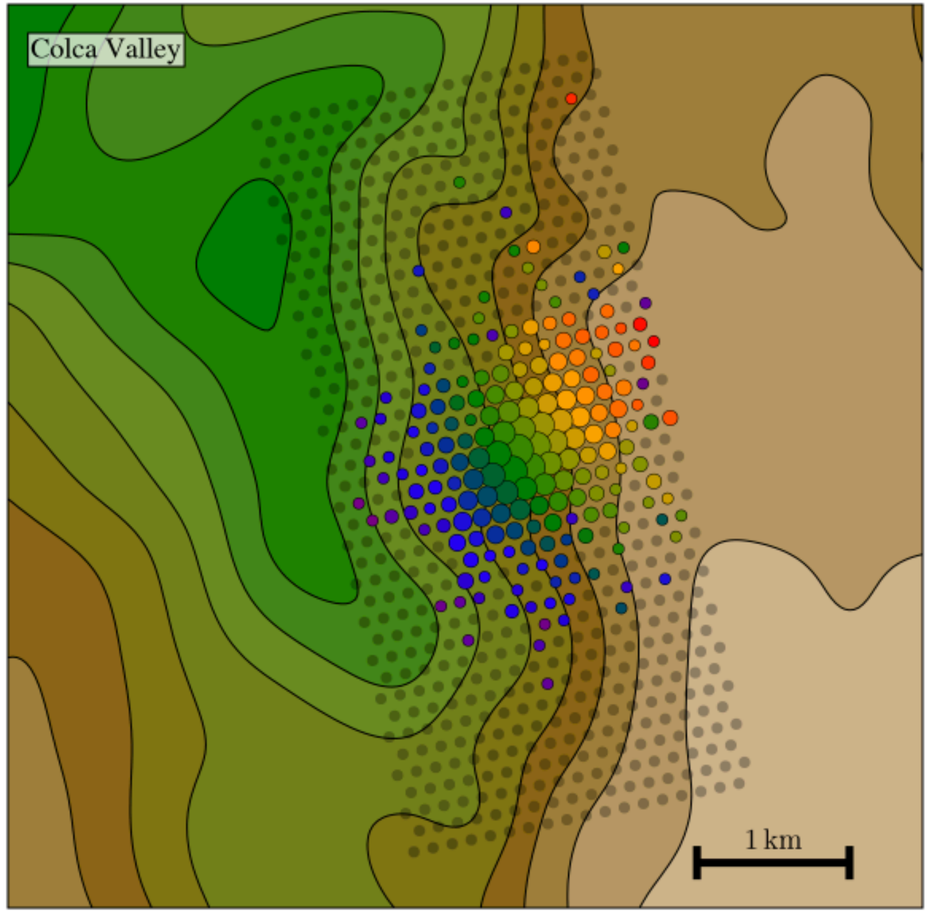
- We often want to know the event rate for a physics case than the injected case
  - E.g. a different energy spectrum or cross section model
- Event weighting allows us to convert injected events to a physically relevant rate
- In principle, this is easy, $w = p_{\text{phys}}/p_{\text{gen}}$, but, as always, the devil is in the details

Outline

- Simulation overview
- `TAMBOSim` main results
- `TAMBOSim`, a practical introduction

Flexible Simulation Framework

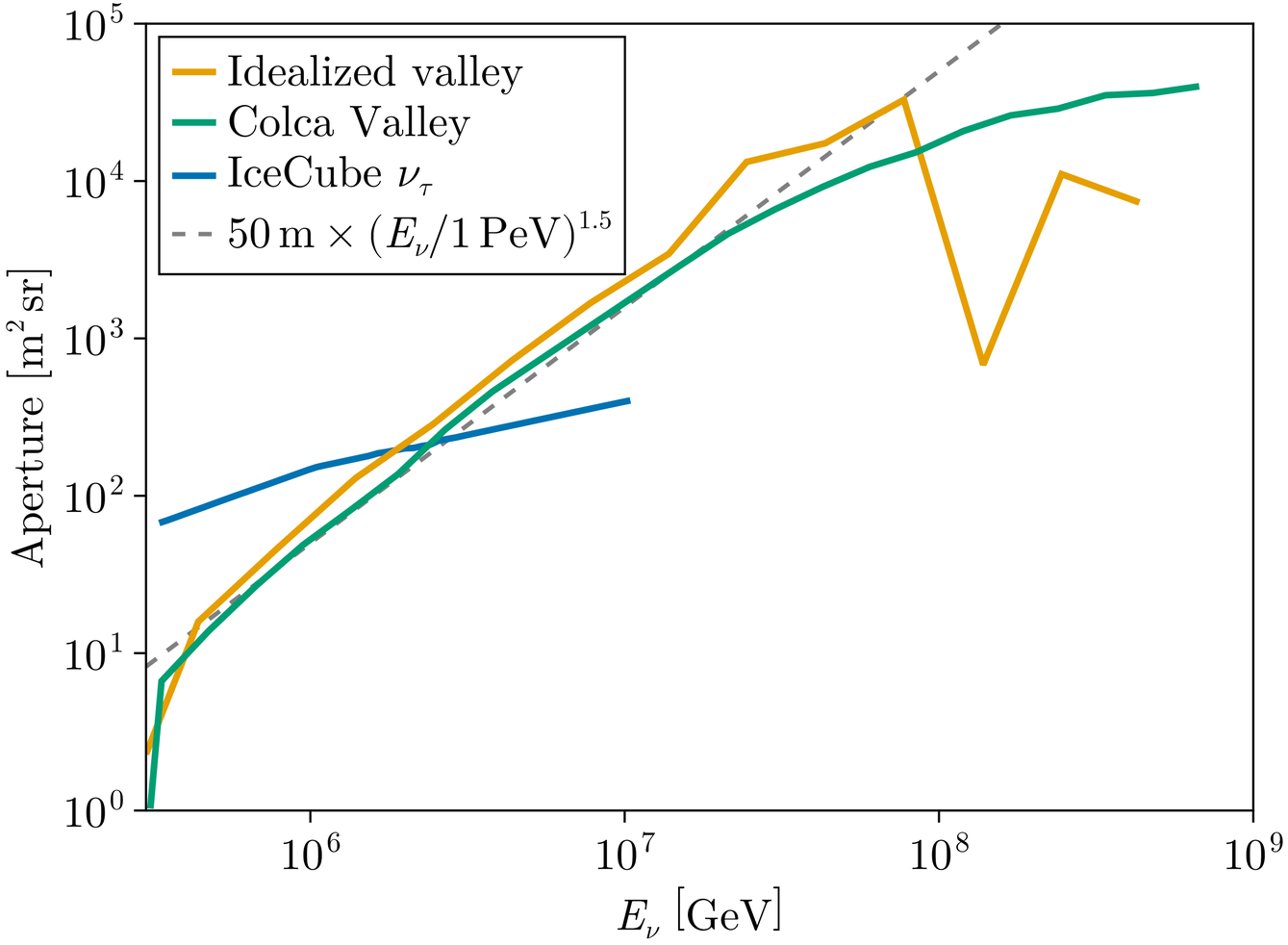




- Simulation carried out in both realistic and idealized value

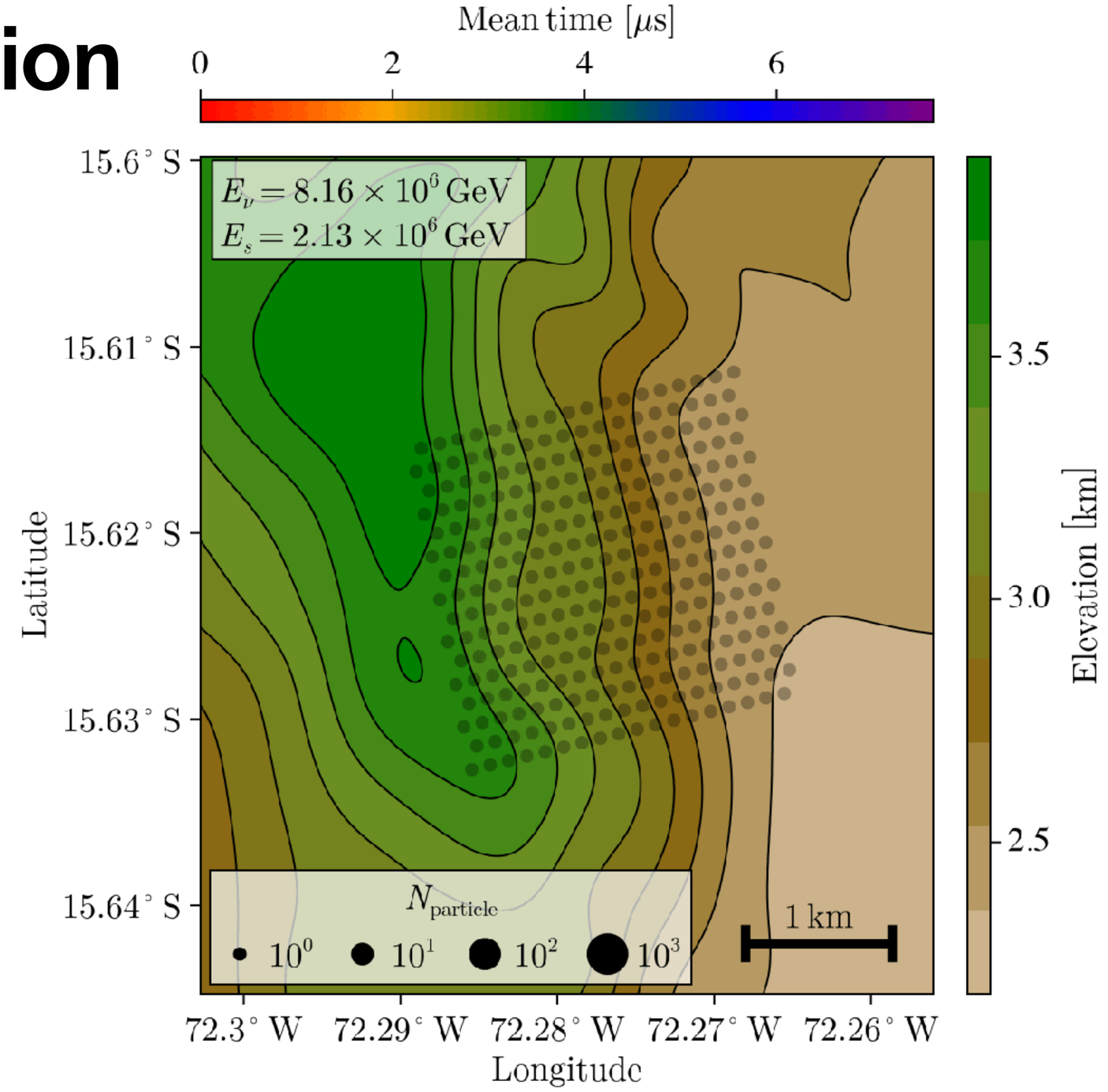
Effective Area

- Aperture has correct energy scaling
- Simulation in comparable valley to initial calculation gives comparable rate (60%–70% difference)
  - Ongoing efforts to track down differences
- Aperture supersedes IceCube's around 3 PeV and is 5x larger at 10 PeV



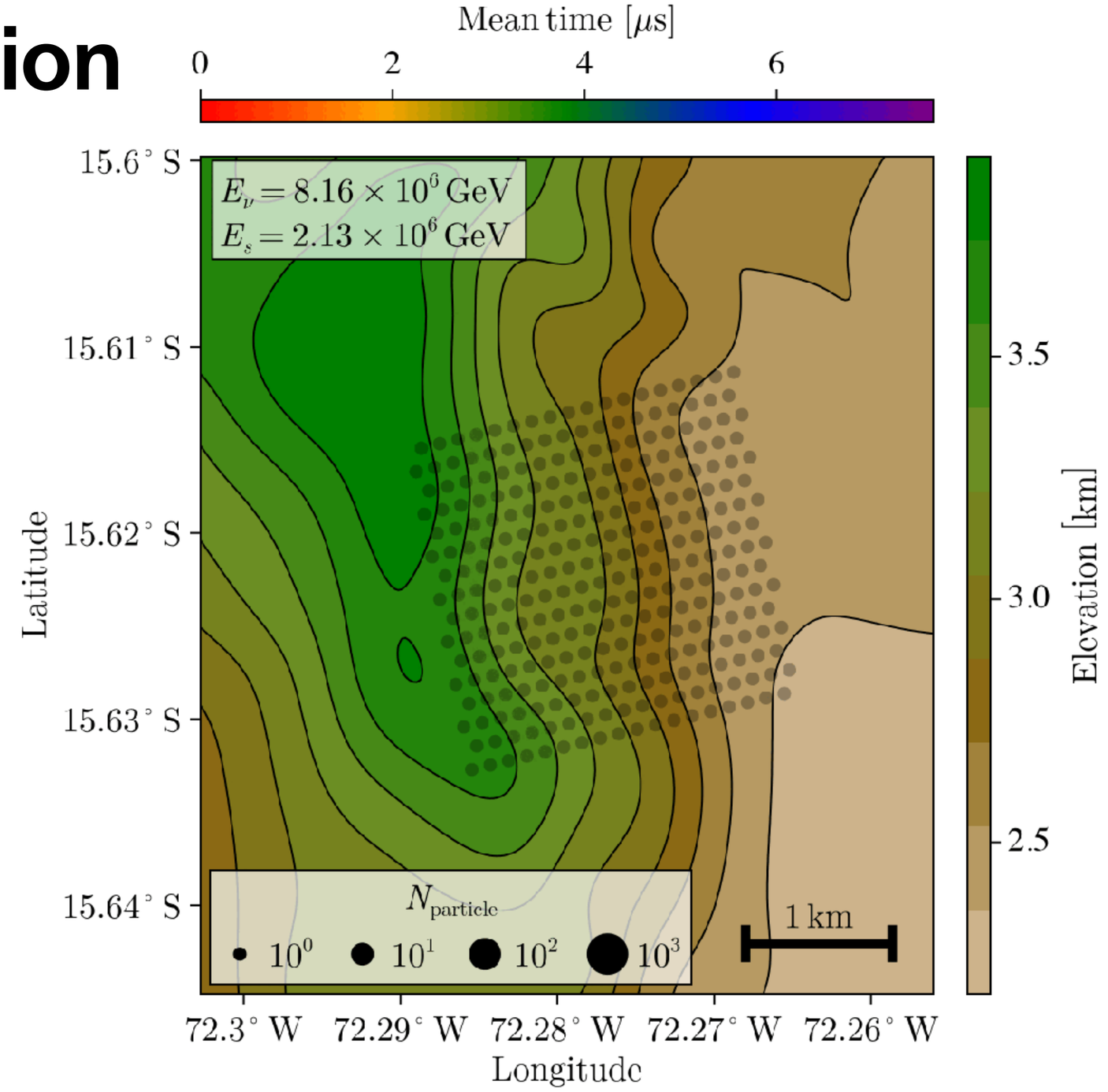
Full Timing Information

- Event from 8 PeV neutrino which gives 2 PeV to shower
- Detector has ~300 modules spaced by 150 m
  - Larger than TAMBITO, but closer to that than full array



Full Timing Information

- Event from 8 PeV neutrino which gives 2 PeV to shower
- Detector has ~300 modules spaced by 150 m
  - Larger than TAMBITO, but closer to that than full array

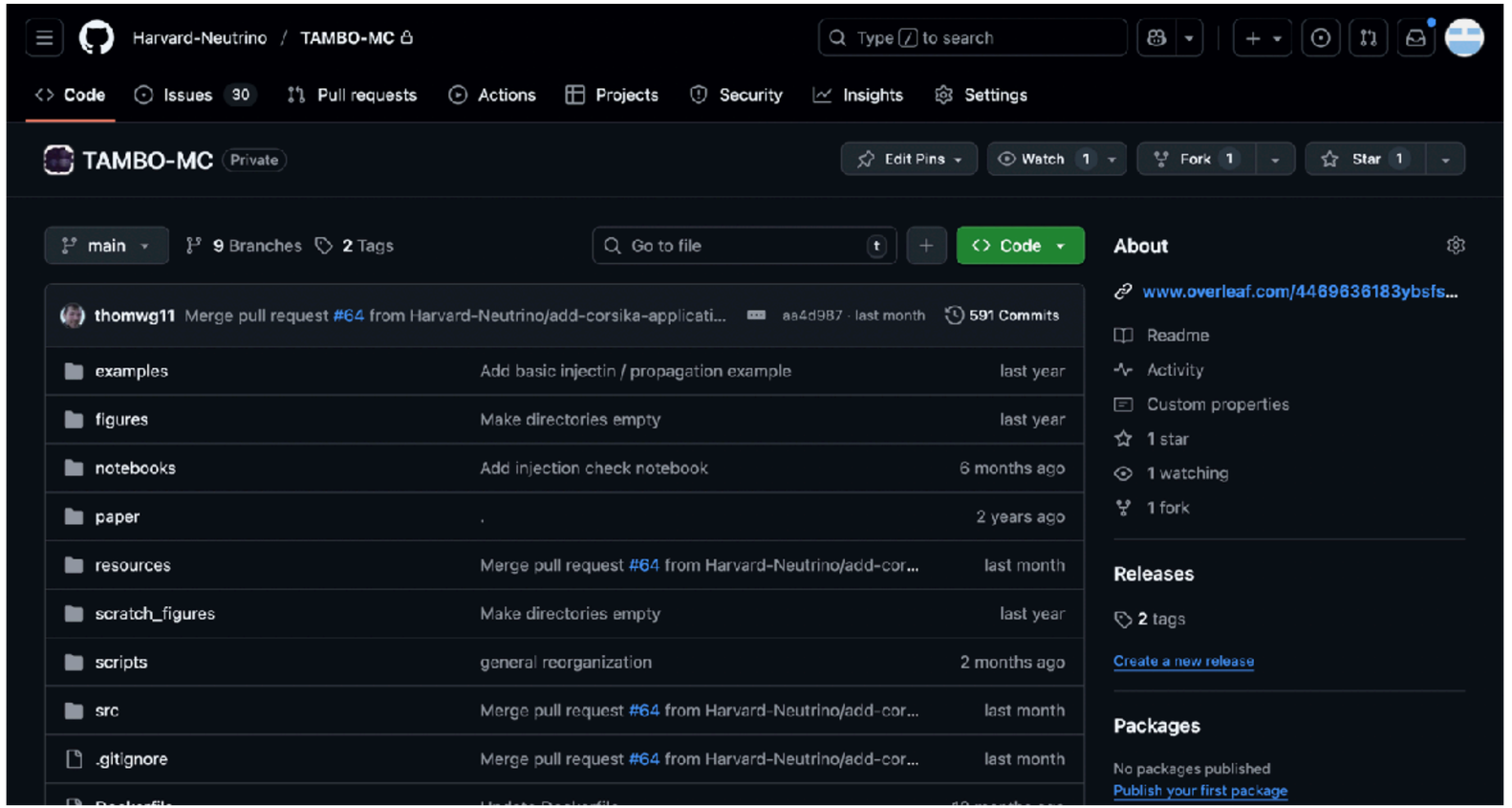


Outline

- Simulation overview
- `TAMBOSim` main results
- `TAMBOSim`, a practical introduction

TAMBOSim

- All of these steps have been combined in the TAMBOSim package which is available through GitHub
- The repository is currently public and can be accessed at



TAMBOSim

Capabilities and limitations

- TAMBOSim was designed to simulate the main TAMBO physics case, and as such that is the most ergonomic process
- Down-going cosmic-ray air showers can be trivially included by running CORSIKA8 independently and feeding the output into the detector response portion of the code
- More horizontal air showers that may skim the mountain require more care. This is actively being developed

TAMBOSim Installation

FLUKA Dependency

- First you must install FLUKA, which is used for hadronic interactions in CORSIKA8
- This can be downloaded from the FLUKA website, but you must make an account
- Installation directions for MacOS and Linux can be found here. Instructions for installation on Windows can be found here, but beware

`TAMBOSim` Installation

Python Dependencies

- Python version requirements have not been rigorously tested, but there is anecdotal success using 3.11.X and 3.12.X
- Set up a virtual environment by running `` `python -m venv </path/to/tambo_venv>` `` and activate it by running `` `source </path/to/tambo_venv>/bin/activate` ``
- Install required packages from PyPI with `` `pip install conan particle==0.25.1 numpy proposal` ``
- Install TauRunner by running `` `git clone git@github.com:icecube/TauRunner.git <path/to>/TauRunner; pip install <path/to>/TauRunner` ``

TAMBOSim Installation

CORSIKA8 Dependency

- You must have a working version of CORSIKA8 installed
- The source code can be cloned from this repository
  - N.B. You must clone using the `—recursive` flag!!!
- Compilation directions can be found here, and have been kind of reliable
  - Anecdotally, there have been problems resolved by editing `corsika/tests/framework/CMakeLists.txt` and commenting line 11, which pertains to testProcessSequence

TAMBOSim Installation

CORSIKA8 Application

- We have a custom CORSIKA8 application that ships with TAMBOSim
- Navigate to `</path/to/TAMBOSim>/src/corsika/` and run

```
mkdir build

export CORSIKA_PREFIX=/path/to/corsika8/top/level/directory
export CONAN_DEPENDENCIES=${CORSIKA_PREFIX}/corsika-install/lib/cmake/dependencies
export FLUPRO=/path/to/fluka/top/level/directory
export FLUFOR=<name of FORTRAN you built CORSIKA against, like gfortran>
export WITH_FLUKA=ON

cmake -DCMAKE_TOOLCHAIN_FILE=${CONAN_DEPENDENCIES}/conan_toolchain.cmake \
    -DCMAKE_PREFIX_PATH=${CONAN_DEPENDENCIES} \
    -DCMAKE_POLICY_DEFAULT_CMP0091=NEW \
    -DCMAKE_BUILD_TYPE=RelWithDebInfo \
    -Dcorsika_DIR=${CORSIKA_PREFIX}/corsika-build \
    -DWITH_FLUKA=ON \
    -S $PWD/source \
    -B $PWD/build

cd build

make
```

`TAMBOSim` Installation

Jump into Julia

- `TAMBOSim` is written in the Julia language, and as such, one must download the Julia executable.
- We recommend Julia1.11 with installation via `juliaup`, see here for instructions

`TAMBOSim` Installation

Jump into Julia

- Now you need to set a few environmental variables
  ```
  export TAMBOSIM_PATH=</path/to/TAMBOSim>
  export TAMBO_DATA_PATH=</path/to/data>
  ```
- And don't forget to set `FLUPRO` and `FLUFOR` if you are in a new session
- Now you can run `julia` to launch an interactive session
- Then you can run `using Pkg; Pkg.develop(path=ENV["TAMBOSIM_PATH"]); Pkg.resolve(); Pkg.instantiate()` and `TAMBOSim` will be installed

`TAMBOSim` Installation

Jump into Julia

- Now you need to set a few environmental variables
  ```
  export TAMBOSIM_PATH=</path/to/TAMBOSim>
  export TAMBO_DATA_PATH=</path/to/data>
  ```
- And don't forget to set `FLUPRO` and `FLUFOR` if you are in a new session
- Now you can run `julia` to launch an interactive session
- Then you can run `using Pkg; Pkg.develop(path=ENV["TAMBOSIM_PATH"]); Pkg.resolve(); Pkg.instantiate()` and `TAMBOSim` will be installed
- Phew!!!

Basic Usage

Injection

- The basic injection can be run by doing

```
using Pkg
Pkg.develop(path="$(ENV["TAMBOSIM_PATH"])")
using Tambo
sim = Simulation("$(ENV["TAMBOSIM_PATH"])/resources/
configuration_examples/larger_valley.toml")
inject_ν!(sim, sim.config["injection"], 100, 925)
```

Basic Usage

Injection

- The results are then saved in `sim.results["injected_events"]`
- This has particle information at three points in time:
  - `initial_state`: particle information at the surface of the Earth
  - `entry_state`: particle information when the neutrino enters the simulation box
  - `final_state`: particle information for the final state charged lepton
- The `oneweight` is also stored so that it doesn't need to be computed

Play with the output!

Injection

- Make a plot to confirm that the injected energy spectrum matches expectation
- Find out how that energy spectrum is shifted for when the neutrinos arrive at the simulation region. Is there an angular dependence? Should we expect one?

Basic Usage

Tau Propagation

- Building off this, we can then propagate the resulting charged leptons by running `propagate_τ!(sim, sim.config["proposal"], 925)`
  - Warning! This will take a very long time the first time you run it. Maybe grab a coffee
- This will populate `sim.results[“proposal_events”]`
- This will have:
  - `propped_state`: state of the tau lepton before decay
  - `decay_products`: state of all decay products

Play with the Output

- What fraction of the tau lepton energy was lost in propagation?
- What fraction of the initial neutrino energy went to visible decay products?

Basic Usage

CORSIKA

- The individual decay products can then be propagated by running `` `run_subshower!(sim, sim.config["corsika"], proposal_id, decay_id, seed` ``, where `proposal_id` and `decay_id` are the indices of the desired tau lepton and decay product respectively
- This can take awhile so run at your discretion

Advanced Usage

Snakemake

- Will has kindly written Snakelike interface that allows you to run all of these steps automatically in sequence
- This is very useful for large simulation batches on distributed clusters
- There will be a session on this in the afternoon

Questions??