Software for Large Astroparticle Physics Projects

The Example of Radio Detection of Neutrinos

Anna Nelles

Particle Physics Colloquium Karlsruhe

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

ecap ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

DESY.

Software in Multi-Messenger Astronomy

There is no science without it

Software in Multi-Messenger Astronomy

There is no science without it

- All research in multi-messenger astronomy relies on software
    - Software to run the experiments
    - Software to transmit the data
    - Software to simulate the instruments performance
    - Software to reconstruct the data
    - …
- However, only a (small) fraction of the software is written and developed by people who have an explicit training for these tasks
- And that is where the 'fun' begins

Software in Science

What the realty looks like

Software in Science

What the realty looks like

- Conversations that you may have encountered:

Software in Science

What the realty looks like

- Conversations that you may have encountered:
    - Advisor to PhD student: “There is this software that I used during my PhD thesis, I think we should use this for this project.”

Software in Science

What the realty looks like

- Conversations that you may have encountered:
  - Advisor to PhD student: “There is this software that I used during my PhD thesis, I think we should use this for this project.”
  - Senior PhD student to young PhD student: “I did the analysis with this random set of scripts that I dumped without documentation on this cluster. You can probably understand what they do.”

Software in Science

What the realty looks like

- Conversations that you may have encountered:
  - Advisor to PhD student: “There is this software that I used during my PhD thesis, I think we should use this for this project.”
  - Senior PhD student to young PhD student: “I did the analysis with this random set of scripts that I dumped without documentation on this cluster. You can probably understand what they do.”
  - Collaboration member to new PhD student: “The software is super easy to install, you only need to follow this 10 page manual and run an obscurely old version of an operating system and it will work out of the box.”

Software in Science

What the realty looks like

- Conversations that you may have encountered:
  - Advisor to PhD student: “There is this software that I used during my PhD thesis, I think we should use this for this project.”
  - Senior PhD student to young PhD student: “I did the analysis with this random set of scripts that I dumped without documentation on this cluster. You can probably understand what they do.”
  - Collaboration member to new PhD student: “The software is super easy to install, you only need to follow this 10 page manual and run an obscurely old version of an operating system and it will work out of the box.”
  - Advisor to PhD student: “Fortran77 is not a problem for you, right?”

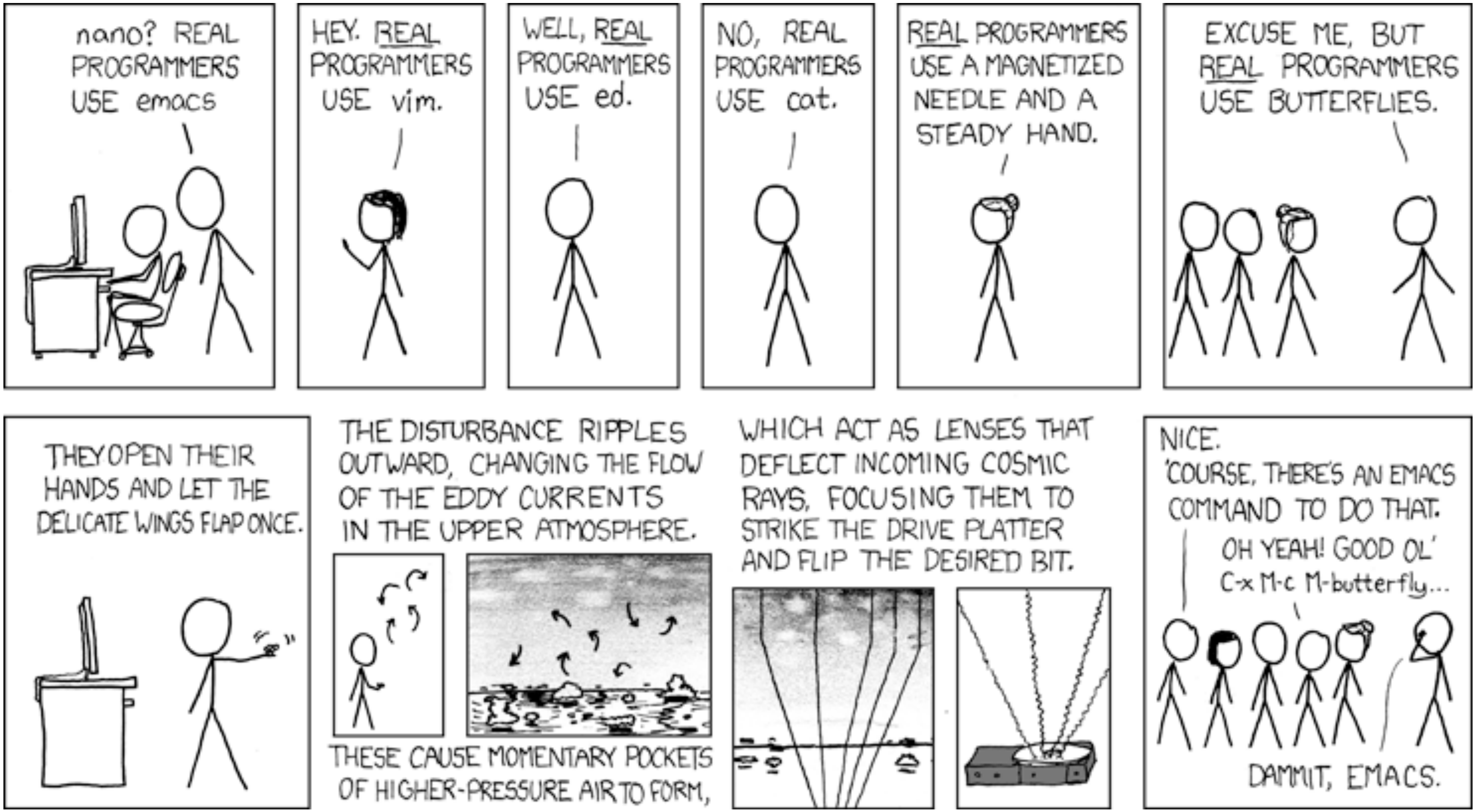
Software in Science

What the realty looks like

- Conversations that you may have encountered:
  - Advisor to PhD student: “There is this software that I used during my PhD thesis, I think we should use this for this project.”
  - Senior PhD student to young PhD student: “I did the analysis with this random set of scripts that I dumped without documentation on this cluster. You can probably understand what they do.”
  - Collaboration member to new PhD student: “The software is super easy to install, you only need to follow this 10 page manual and run an obscurely old version of an operating system and it will work out of the box.”
  - Advisor to PhD student: “Fortran77 is not a problem for you, right?”
  - Expert PhD student to young PhD student: “*Llga &%$CTRL gcpaiugrhg zrg (Lots of complicated computing words).* This will fix your problem, super efficiently.”

Software in Science

Some of you may find this funny … others not all



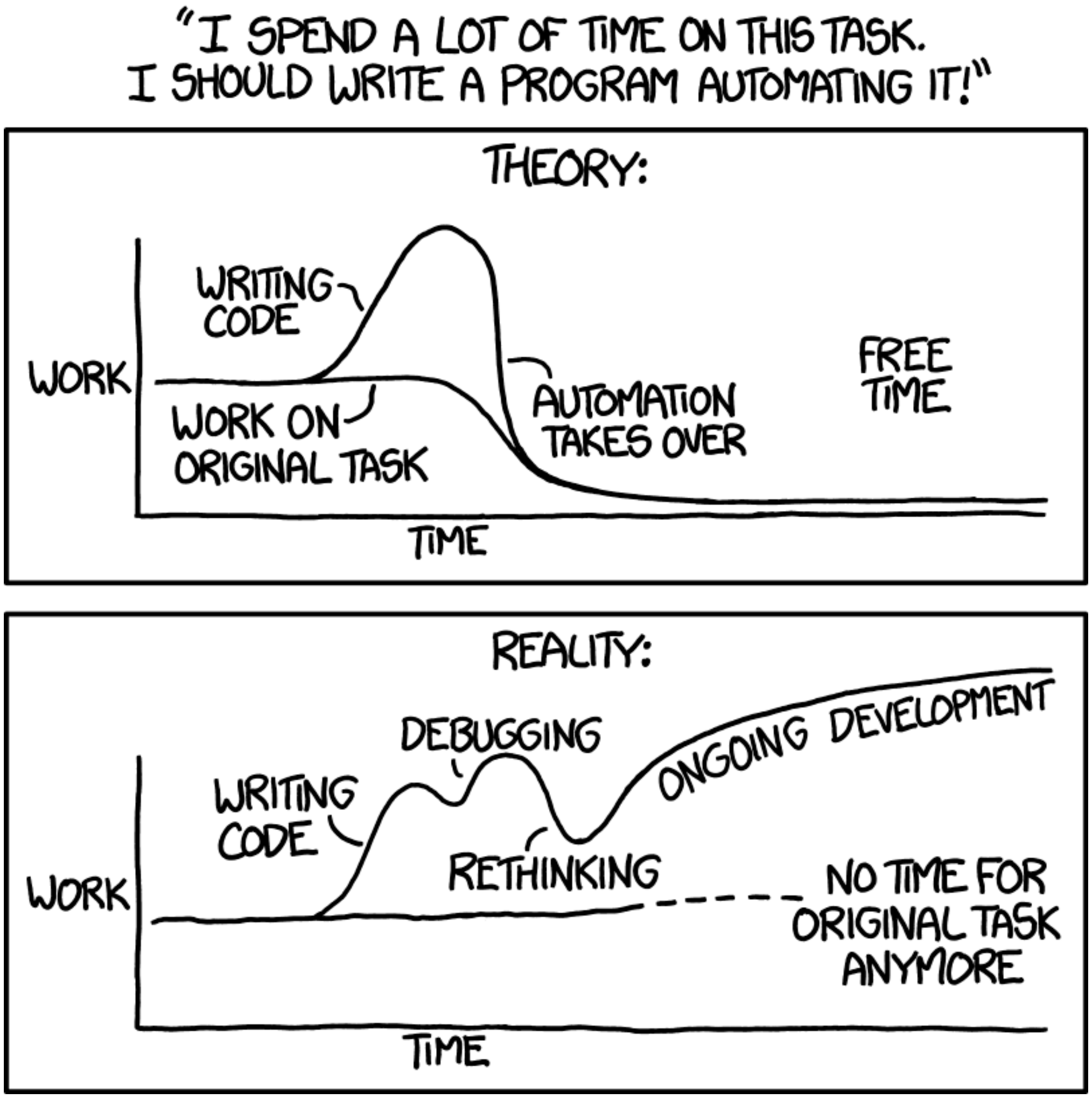
https://xkcd.com

Software in Science

What are the underlying problems?

- Having professional software, e.g. user-friendly, sustainable, flexible, well-documented, well-maintained, is not (yet) a high priority in many research projects
  - Priority: Software development is considered a ‘service task’ that doesn’t give you the papers you need for the scientific career, i.e. waste of time
  - Hierarchy: The PhD students and post-docs carry most of the load and their complaints tend to not reach the senior staff, i.e. “we also always complained during our PhD, how bad can it be? I don’t understand what this is about anyway.”
  - Money: Professional software developers ask for more money than TVL-13, so we simply cannot afford them. It is really hard to get dedicated money for such tasks in research grants, especially for a permanent role.
  - Science: When you start developing a project, you tend to not know what the problems will be ahead.

Software in Science

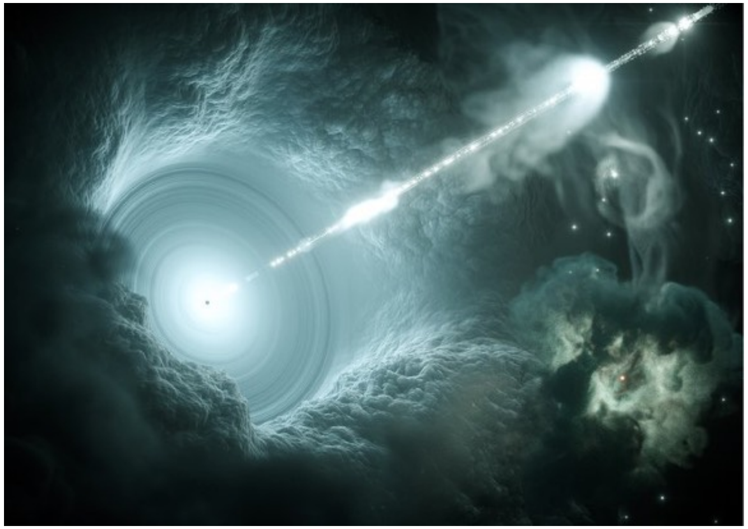


https://xkcd.com

The example of radio detection of neutrinos

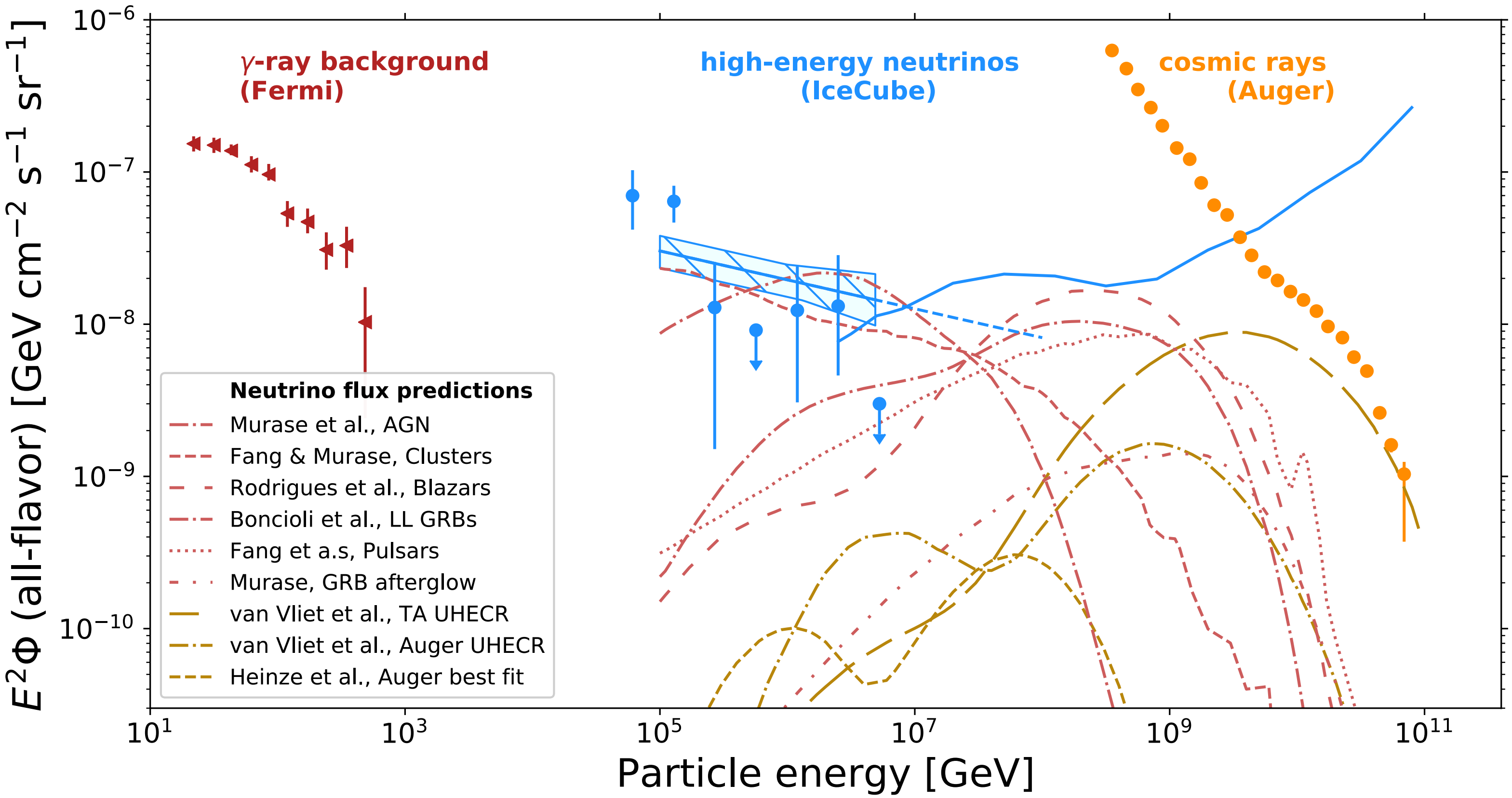
A story that really happened

- Outline
  - Scientific motivation of radio detection of particle showers
  - Underlying physics of radio emission
  - Different software to tackle the problem(s)
  - Current software
  - Outlook



Multi-messenger Universe

Where are we at, at the moment?



Radio detection of particle showers

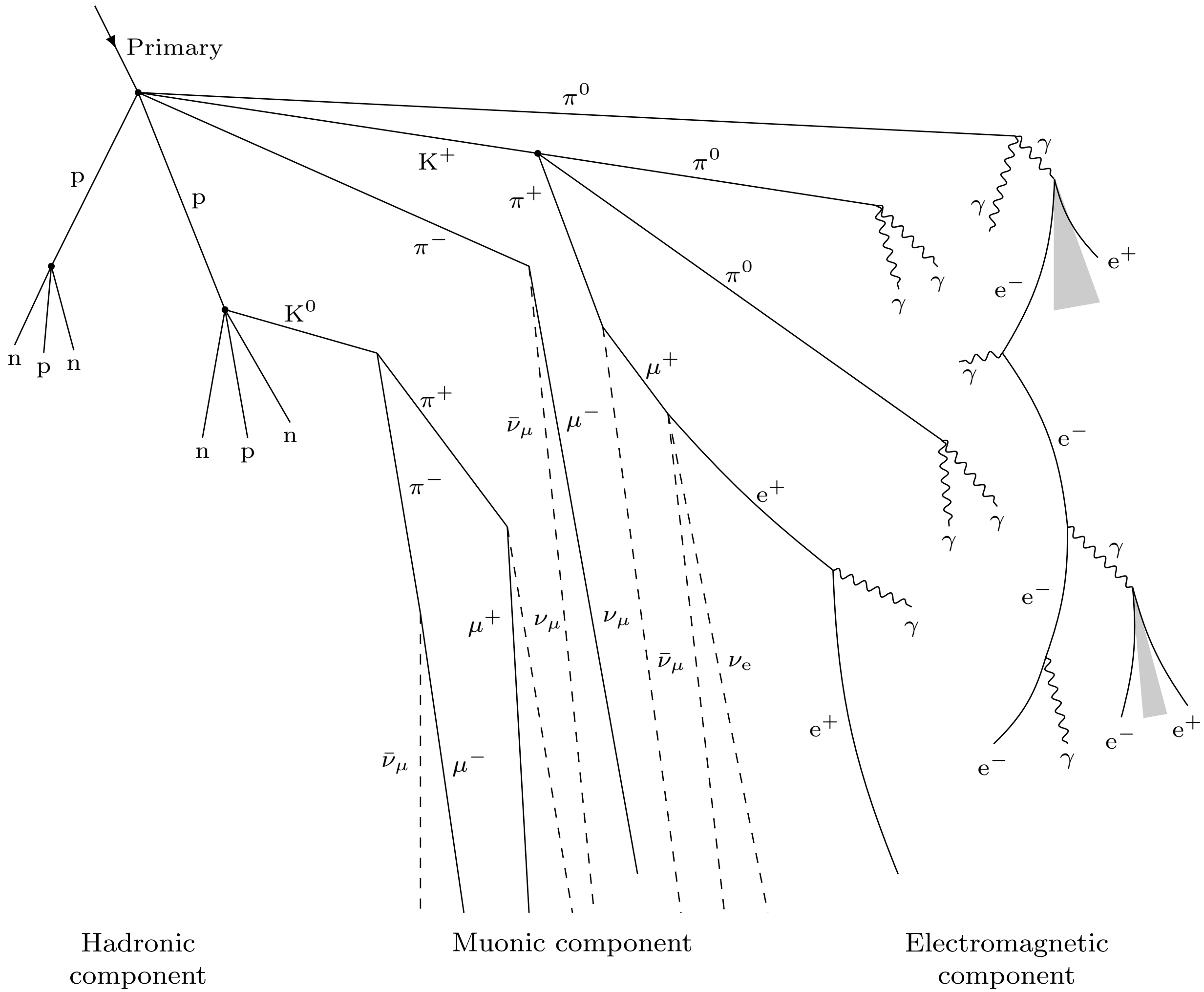
Basic idea, suggested already in the 1960s

- Particle showers create radio emission (see next slides)
- Radio waves are not attenuated in air/ice like light
- Radio antennas are cheap(er) than particle detectors
- One needs huge instrumented volumes to detect the low flux at the highest energies
- So measuring the radio emission of a shower sounds like a useful idea to instrument large volumes to detect air showers or neutrino induced showers

Radio signals

A theoretical introduction

- Highly energetic particles interact with medium and create shower of secondary particles
- Generally one distinguishes hadronic and electromagnetic showers
- Hadronic showers always have a electromagnetic component



Radio emission of particle showers

A theoretical introduction

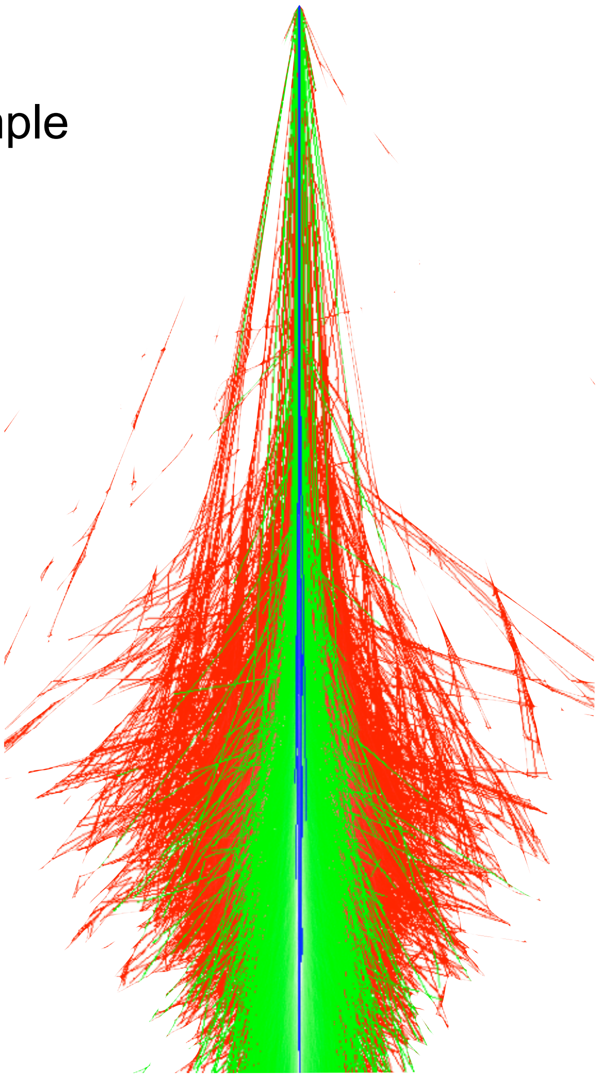
- Radio emission of showers can be explained from simple first principles

- Three ingredients:

    - **Magnetic field**
      (*Geomagnetic field, Lorentz-force*)

    - **Charge imbalance**
      (*Particle Physics processes*)

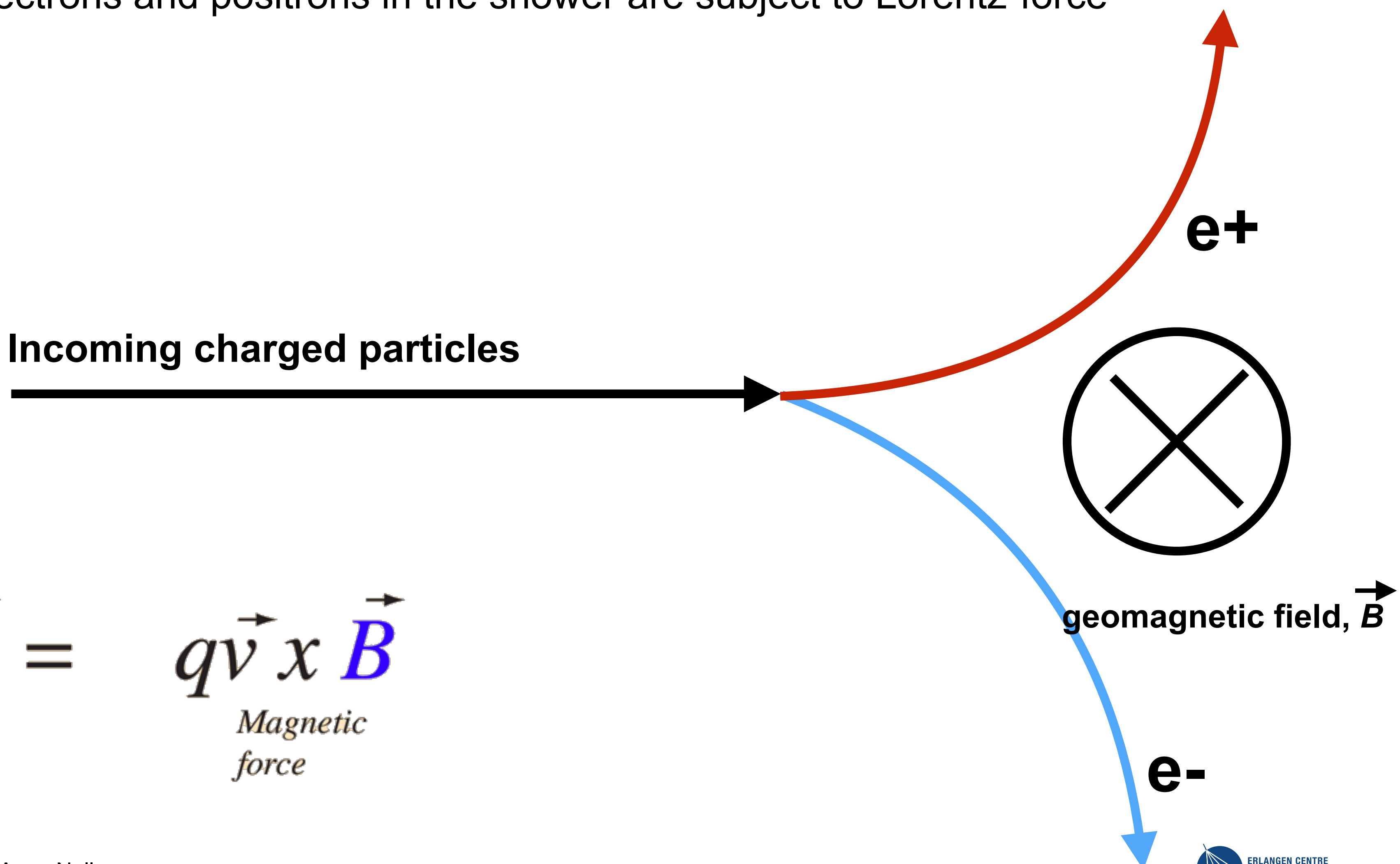
    - **Relativistic compression**
      (*Ray optics and relativity*)



Radio

Geomagnetic effect

- Electrons and positrons in the shower are subject to Lorentz-force



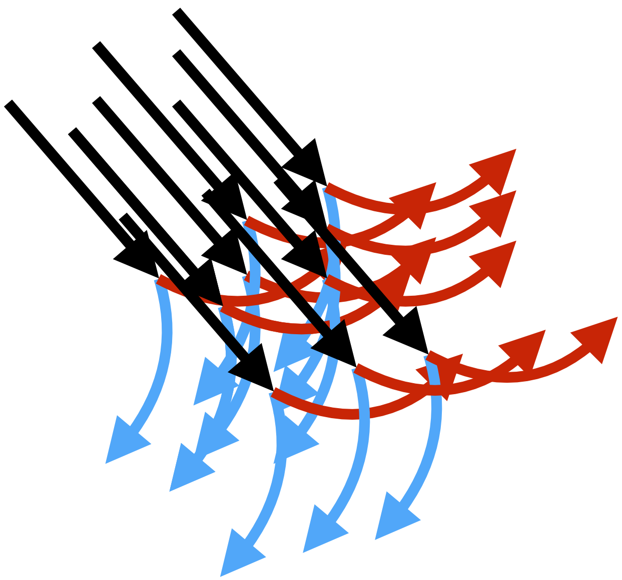
$$\vec{F} = q\vec{v} \times \vec{B}$$

Magnetic force

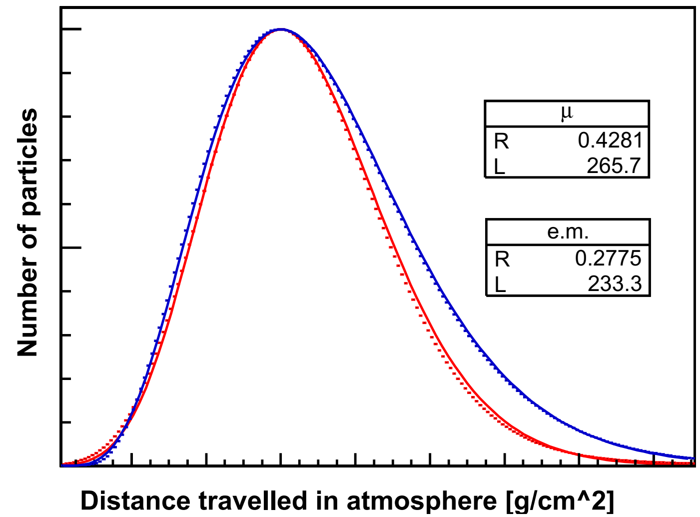
Radio

Geomagnetic effect

- In a shower: many particles
- Charge separation produces a current



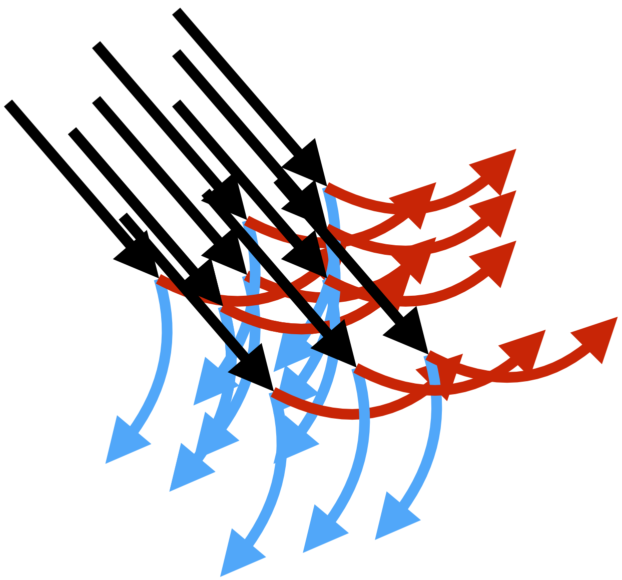
- Number of particles is a function of height above ground



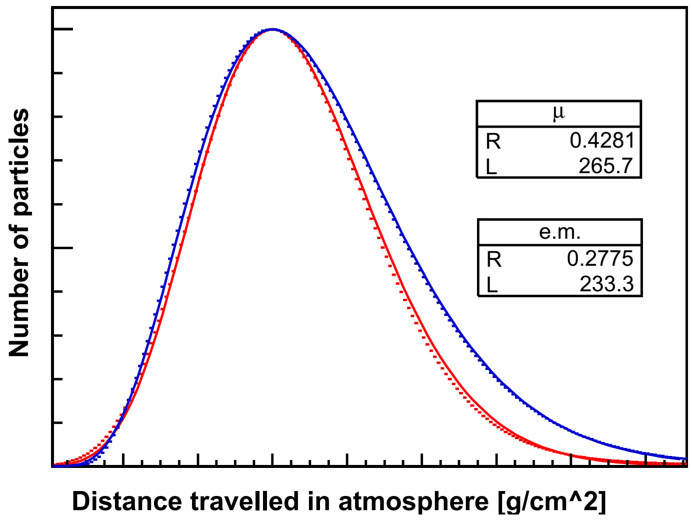
Radio

Geomagnetic effect

- In a shower: many particles
- Charge separation produces a current



- Number of particles is a function of height above ground

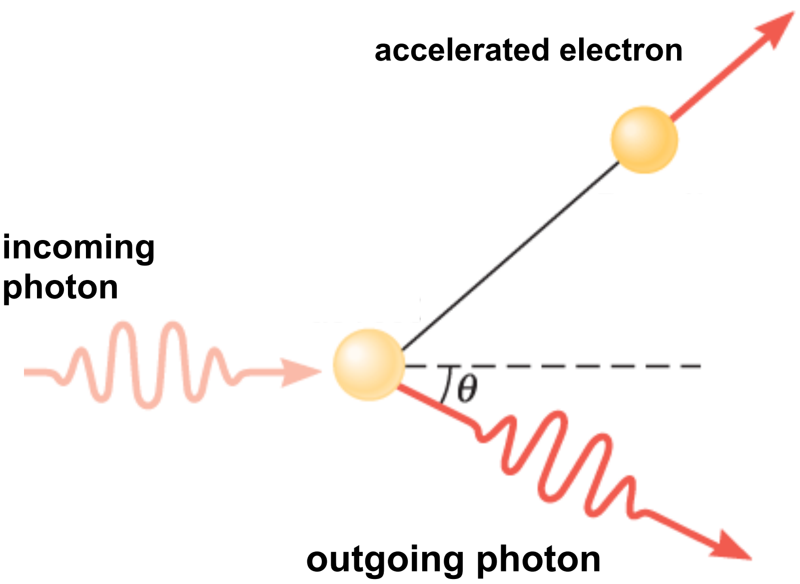


- The current changes as function of time/height
- A changing current causes electromagnetic emission

Radio emission of particle showers

Askaryan effect

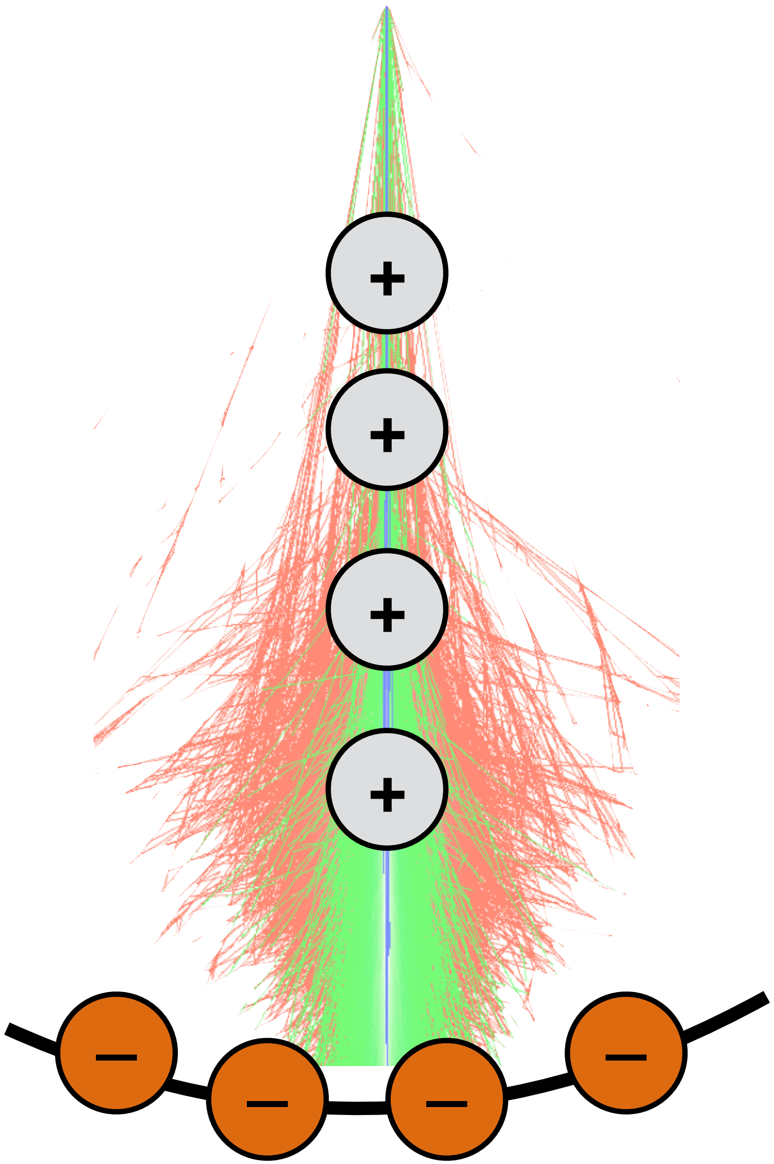
- Remember: numerous high energy photons, positrons electrons in shower
- In atmosphere: only electrons, no positrons
- Shower particles interact with particles in the atmosphere



Radio emission of particle showers

Askaryan effect

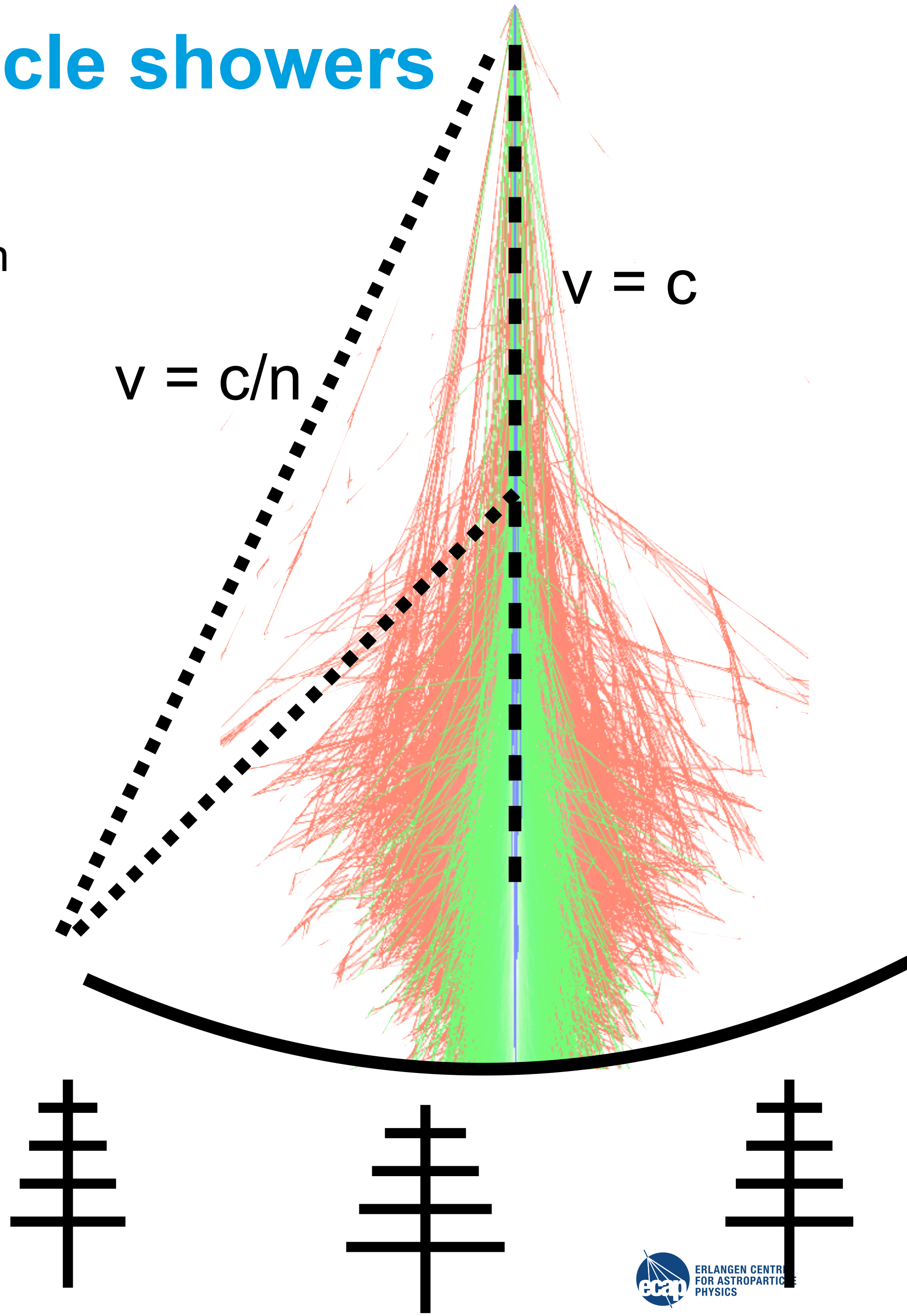
- Charge separation along axis
- Shower front is negative, axis positively charged
- Current along axis, changing as function of time/height
- Also here: changing current induces electric emission



Radio emission of particle showers

Cherenkov-like effects

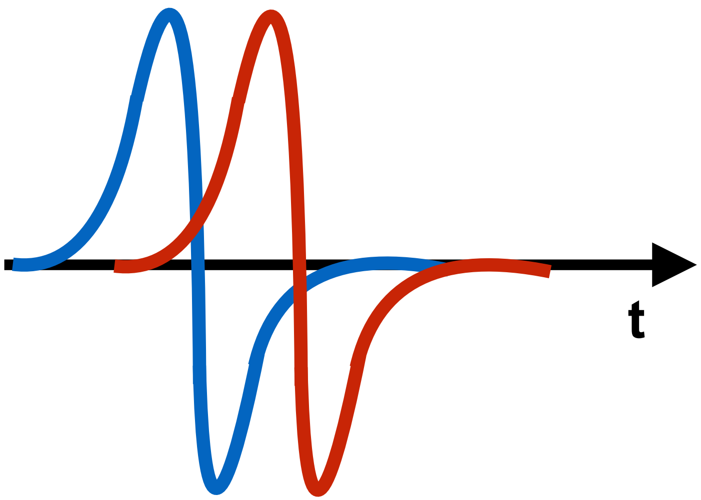
- Shower is faster than its emission at n = 1.003



Radio emission of particle showers

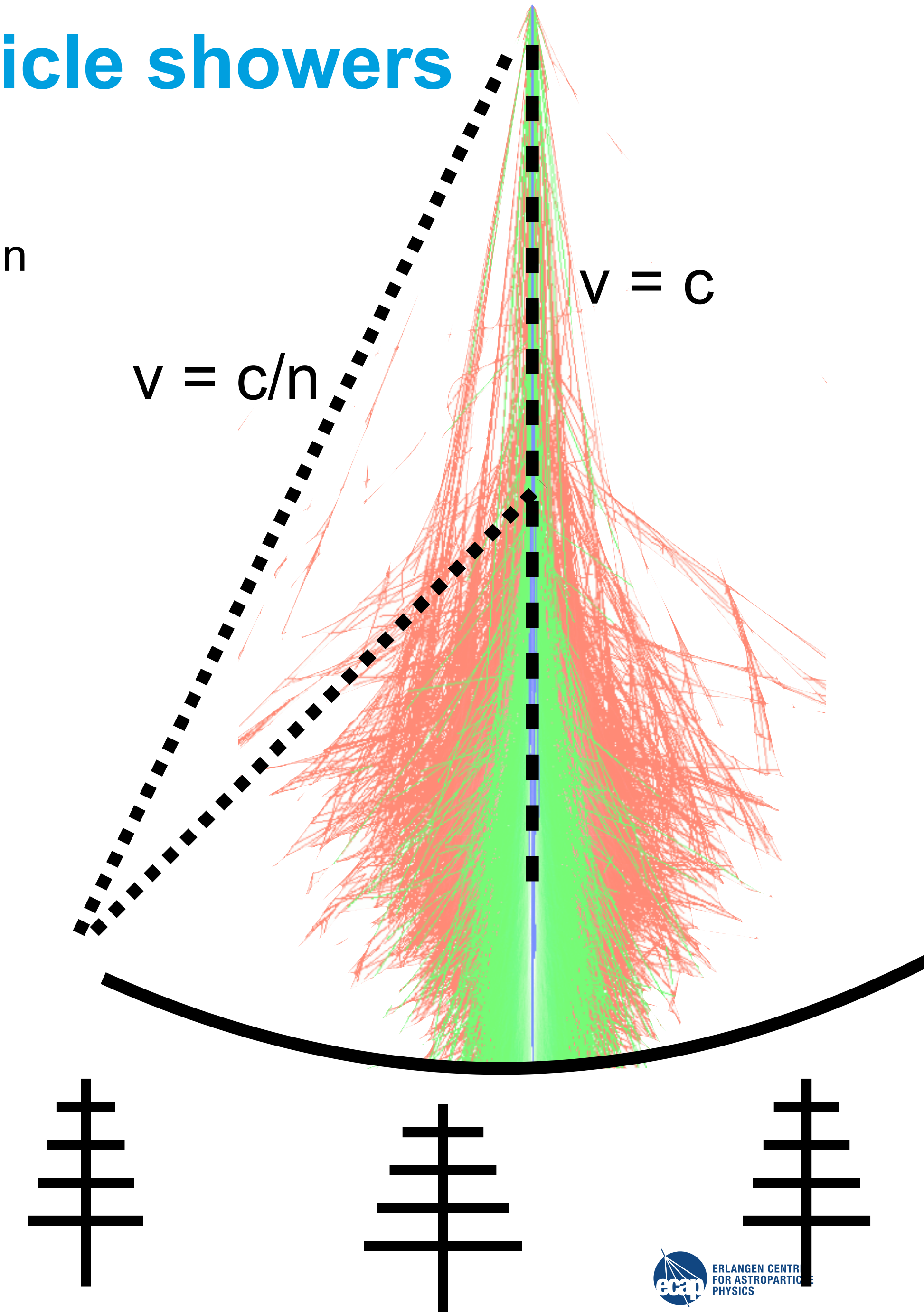
Cherenkov-like effects

- Shower is faster than its emission at n = 1.003



- Signal gets enhanced when it arrives in phase = coherence

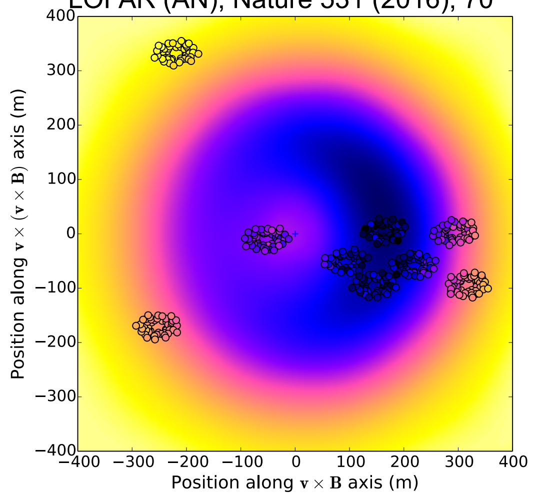
- Enhancement at the Cherenkov angle



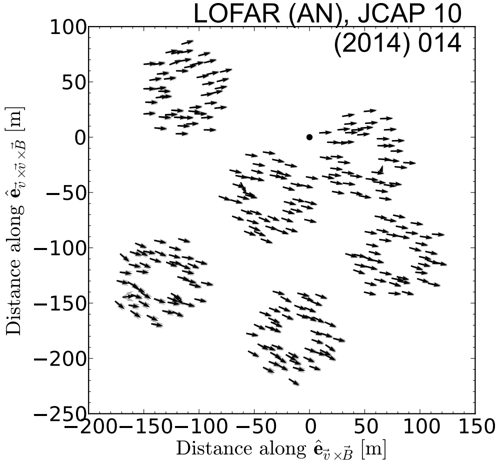
Radio emission of particle showers

We know this now, but not when we started

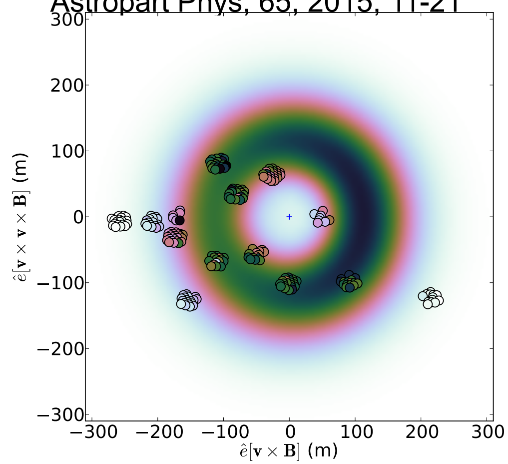
- Signal distribution ✅
- Signal amplitude ✅
- Signal polarization ✅
- Signal frequency spectrum ✅
- Dependence on magnetic field ✅
- Propagation effects ✅
- Detector characterization ✅



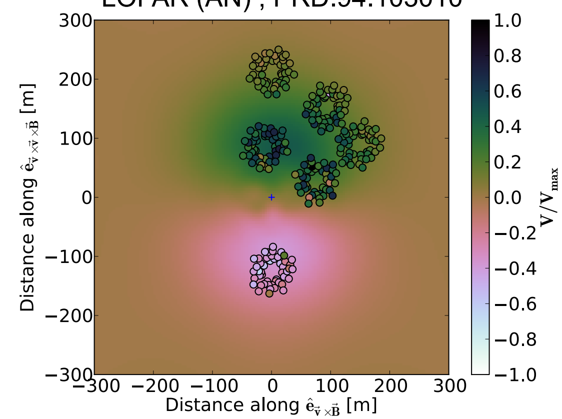
LOFAR (AN), Nature 531 (2016), 70



LOFAR (AN), JCAP 10 (2014) 014



AN et al. (LOFAR) Astropart Phys, 65, 2015, 11-21



LOFAR (AN) , PRD.94.103010

Science and the software

Status roughly 2010

- Radio detection was revisited for air shower detection
- Digital radio telescopes were coming up (like LOFAR) — additional use
- Calculations existed that indicated what an air shower signal would look like
- Software existed that was used for ‘regular’ air showers, i.e. CORSIKA or reconstruction framework of Pierre Auger Observatory
    - A couple of full-time maintainers and mostly user based software development — heritage of particle physics experiments
    - If the software doesn’t do what you want it to do, you have to write it
- Software of radio telescopes existed
    - Software maintained by observatories, users only used complete ‘pipelines’ for producing astronomical images — “the magic black box”
    - If the software doesn’t do what you want it to do, it is likely impossible

Science and the software

Structural decisions

- The format the data comes in, determines half your software set-up, for example:
- Pierre Auger Observatory:
  - originally some binary that the data acquisition spits out, it routinely converted to ROOT (CERN data and analysis framework) using a reader library (different for every detector system)
  - ROOT (used to be) fully C++ based = convenient programming language
  - main Auger software is ROOT and C++ based, with many external dependencies to read all data-formats
- LOFAR:
  - some custom version of HDF5, with a particular data access library embedded in a larger framework of analysis pipeline
  - C++ and Python in principle possible, but framework of analysis pipeline needs to be installed

Science and the software

Structural decisions

- Factors that weren’t thought about **when devising** software:
- Pierre Auger Observatory:
  - Calibration data is stored in XML files and is assumed to be (relatively) constant in time and the same for the whole array
  - Whole software was built around this idea of the detector being stable and identical — good for particle detectors
  - Poor choice for an R&D radio detector with calibration curves for every single electronics item including cables
- LOFAR:
  - LOFAR data was split in half, by polarization of antenna (same orientation of dipoles) and processed separately — good for astronomical images
  - Poor choice for air showers, where one needs to combine all data per antenna to obtain full electric field

Science and the software

Likely consequences

- "One cannot do certain analyses"
  - Not an option, physics needs to advance
- Students and post-docs hack existing software to make the analysis work
  - error-prone, lots of work, complicated (ugly) code, only works until the next problem arises
- Throw out the old software, start again from scratch to write software that actually suits the problem at hand
  - Significant time investment, risk of not finishing on time for a PhD
  - Without decent knowledge of software design, the new solution may just be as bad as the former one
  - Typically no time for documentation, user-friendliness
  - How can one be sure that the new results are correct?

Scientific software development

Some things that would be nice to consider

- “Refactor early and often” (Book: The Pragmatic Programmer)
    - for all software development, one should throw out the old stuff and start fresh — I think this is the most violated rule in scientific software development
- Requirement for refactoring: Testing the results
    - If you have an automated testing system in place, you can always ensure that the results are reproducible
- Requirement for refactoring: Modularized code
    - It is typically impossible for a small group of people to re-write the entire code, it is much more doable to refactor a smaller sub-module
- Requirement for refactoring: Patience of advisors
    - During refactoring, there will be a period in which no new science output can be shown

Scientific software development

Some things that would be nice to consider

INVOLVES RISK

- "Refactor early and often" (Book: The Pragmatic Programmer)
  - for all software development, one should throw out the old stuff and start fresh — I think this is the most violated rule in scientific software development
- Requirement for refactoring: Testing the results
  - If you have an automated testing system in place, you can always ensure that the results are reproducible
- Requirement for refactoring: Modularized code
  - It is typically impossible for a small group of people to re-write the entire code, it is much more doable to refactor a smaller sub-module
- Requirement for refactoring: Patience of advisors
  - During refactoring, there will be a period in which no new science output can be shown

Scientific software development

Some things that would be nice to consider

- Who do you need for the task in an ideal world?
  - The right group of people, sometimes you have to be lucky:
  - Someone with excellent programming and computing skills
  - Someone who is willing to look at existing solutions and finds the right industry standard for this project
  - Someone with the oversight of the whole problem and project management skills
  - Someone who is willing to make the hard decisions
  - Someone with attention to detail and cleaning up after others

Scientific software development

Some things that would be nice to consider

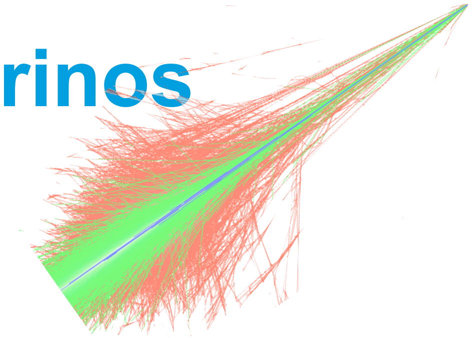
Very few projects have it all

- Who do you need for the task in an ideal world?
  - The right group of people, sometimes you have to be lucky:
  - Someone with excellent programming and computing skills
  - Someone who is willing to look at existing solutions and finds the right industry standard for this project
  - Someone with the oversight of the whole problem and project management skills
  - Someone who is willing to make the hard decisions
  - Someone with attention to detail and cleaning up after others

¯_(ツ)_/¯

The story of radio detection of neutrinos

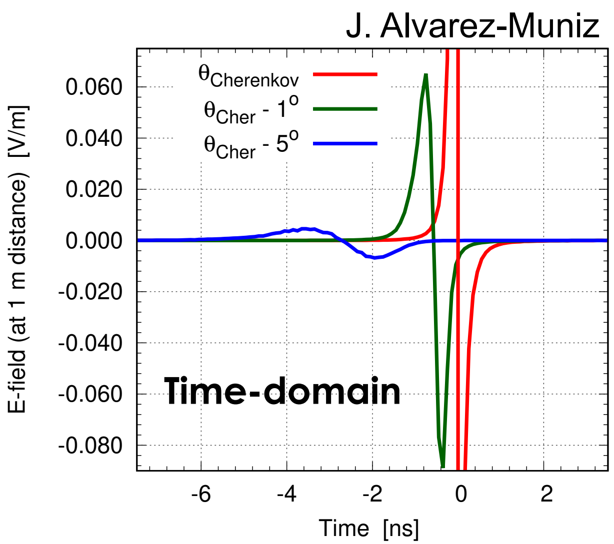
How a piece of code developed and still hindered progress

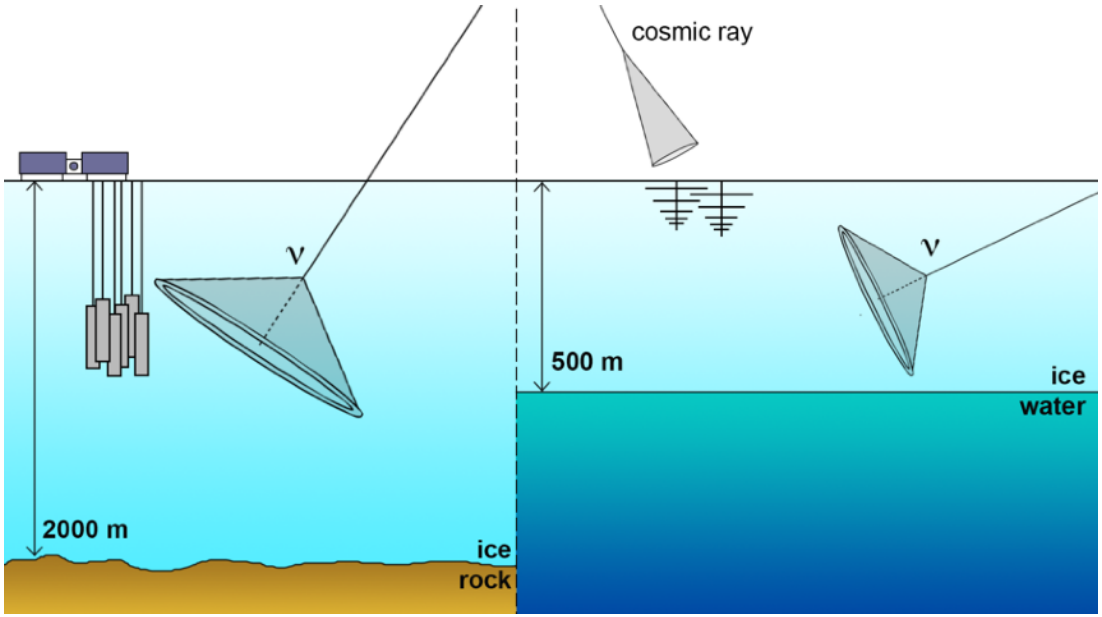


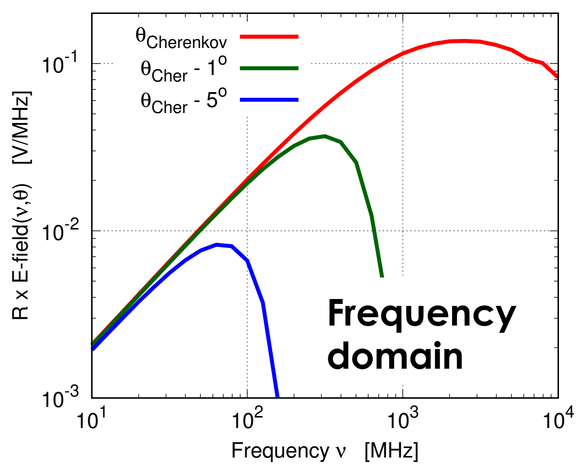
- **Starting conditions:**
  - A paper from 2000 described a theoretical parameterization of the amplitude of the radio signal of neutrinos
  - A student took this parameterization and developed a simple calculation
- **More concrete experimental ideas were developed:**
  - Piece of code was duplicated and extended
    - for a balloon to fly over Antarctica (IceMC) — assuming all emission in the far-field and a single point of detection
    - for a shallow-array of antennas on a reflective surface (ShelfMC) — assuming a regularly spaced grid of stations at the surface
    - for an array of antennas deep in the ice (ARAsim) — assuming deep stations with a regular symmetry

The story of radio detection of neutrinos

Different concepts for the same problem







- Short nano-second scale broad-band pulse
- Amplitude scales with energy of neutrino

The story of radio detection of neutrinos

The code developed (away from each other)

- **Open question: What is the most efficient detector to built?**
  - Some codes used more advanced emission models, some codes used further developed signal propagation, others better treatment of the antenna responses
  - no apples-to-apples comparison using the same code for all designs
- No code had s serious version control, with releases and automated testing (results changed with time)
- Most of the code was uselessly documented and very hard to read
- No code was fully public and installable for anyone outside the collaboration (dependencies of non-public libraries, non-public documentation)
- **Two people with really bad experiences concerning unsuitable software in the past, decided to take the risk and start from scratch**

NuRadioMC

Let's try to build software according to 'industry standards'

- Avoid custom-made solutions at any costs, use only **standard (python) libraries**
- Follow software development routines using **git**
  - Branching, pull-requests, review, commit to master, continous integration, code linting … (see next slide)
- Modularize the code
  - Every step should be exchangeable
- Make code public from Day 1
  - Including issue tracking, discussion and feature requests
- Test the results
  - Standard module sequences are automatically tested for consistency between versions

NuRadioMC

We want a community tool — but it needs rules, not anything goes

- As senior, I have the ‘best’ job now:
  - Review code
  - Complain about missing documentation
  - Ending discussions about which of the 100 options we should choose
  - Being allowed to be convinced to do things differently

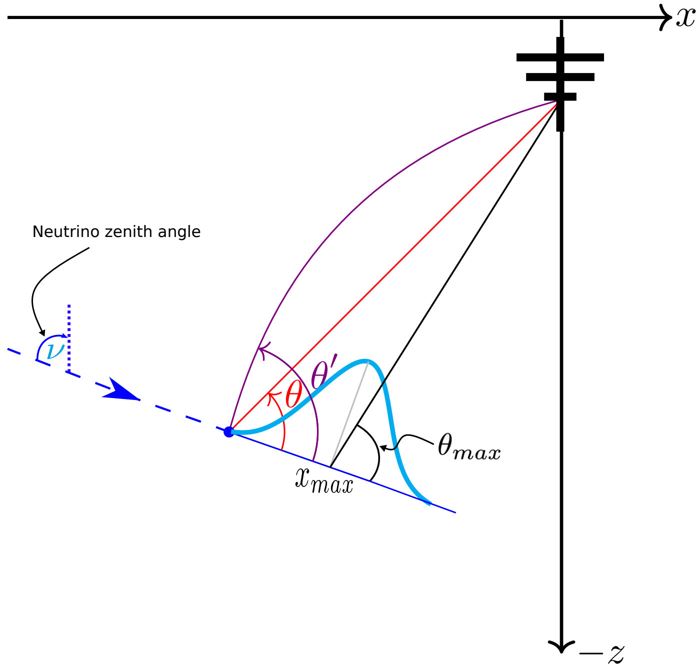
- https://nu-radio.github.io/NuRadioMC/Introduction/pages/contributing.html
- https://github.com/nu-radio/NuRadioMC

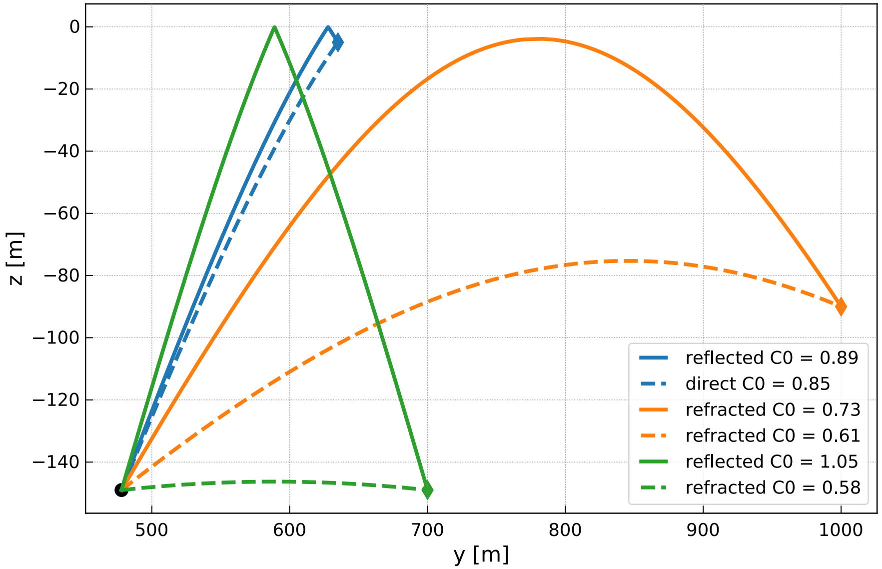
NuRadioMC

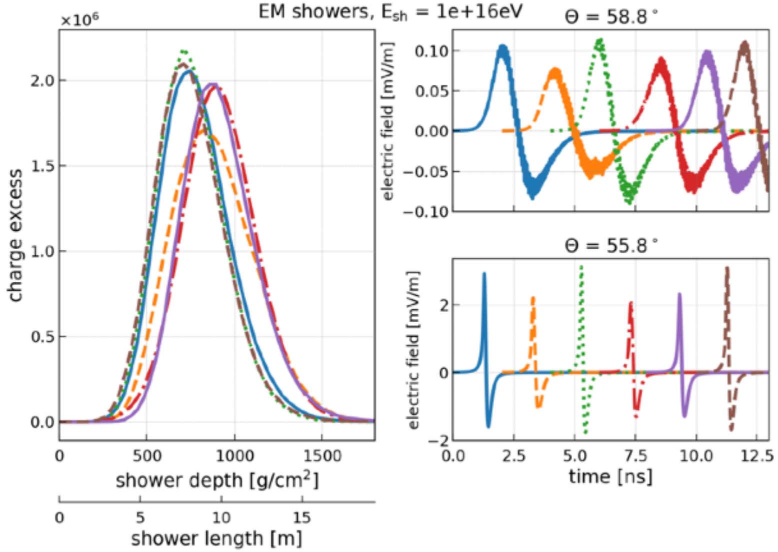
Simulation of neutrino signals

Glaser et al., European Physics Journal C 80, 77 (2020)

- https://github.com/nu-radio/NuRadioMC
- This is our current solution, my personal guess: In 5 years from now, we will start throwing it out







NuRadioMC

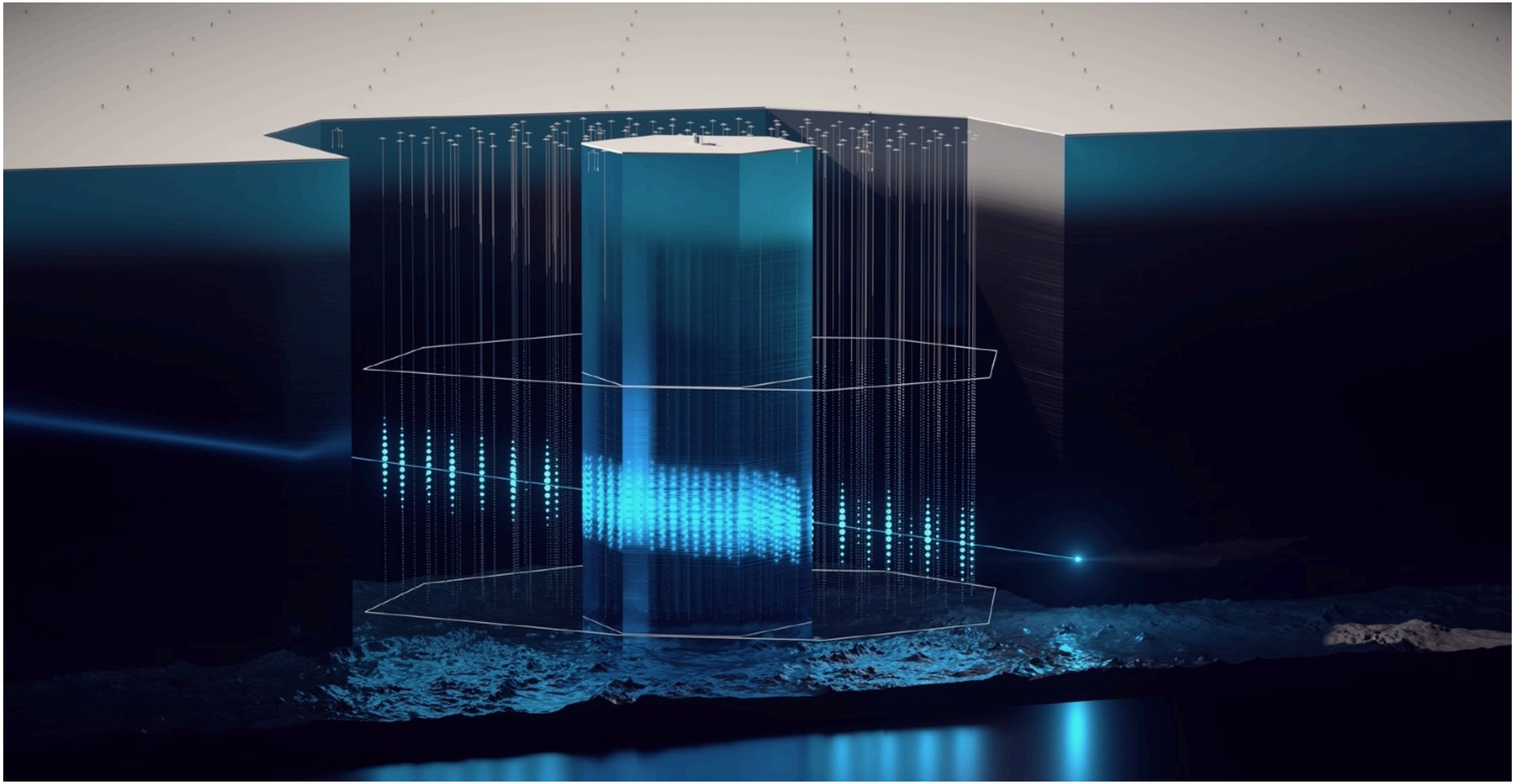
Let’s try to build software according to ‘industry standards’

- Side note:
  Core developers took a risk, but were ‘lucky’ to be in positions where freedom to do so was allowed
  - Positions funded on scholarships with scientific freedom
  - Choice paid off in the end — not necessarily the case
  - Always had ‘other projects’ on the side, i.e. a fall-back option for papers and results
- It is not like there were no hurdles to overcome
  - “Why should we trust the new software?”
  - Developers change positions in the course of writing, load fell temporarily on fewer people
  - Lucky to find students with good software development skills to support project

Why are we doing this?

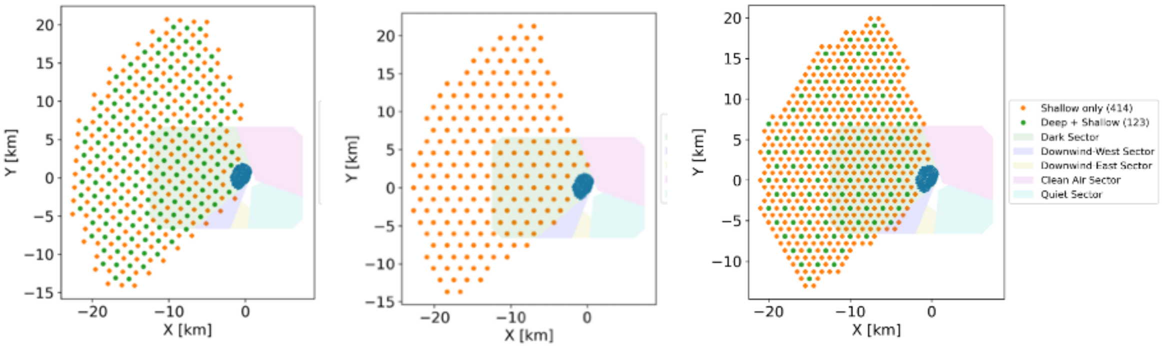
Software should follow a purpose

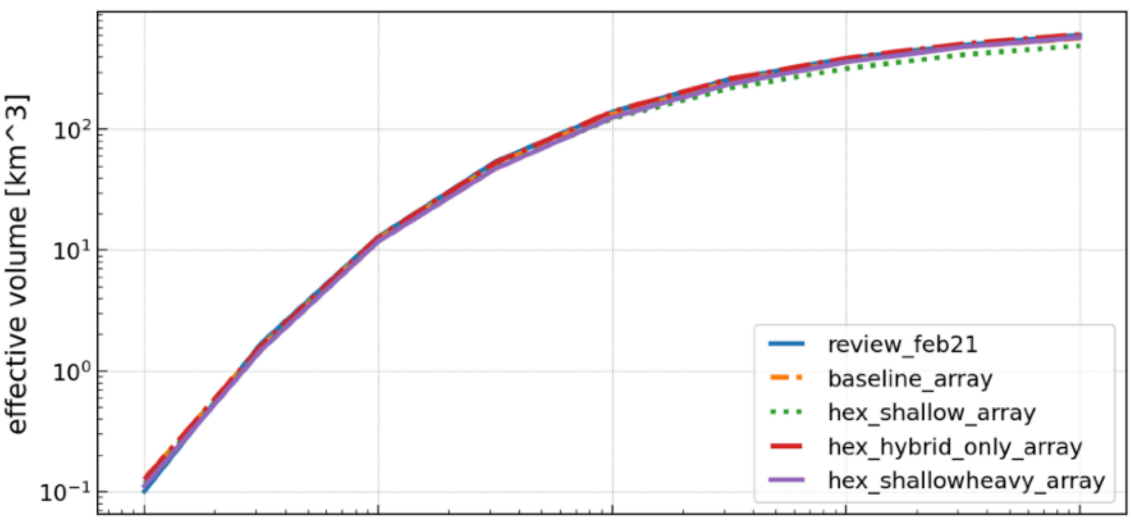
- We can now finally do all the comparisons that we had wanted to do

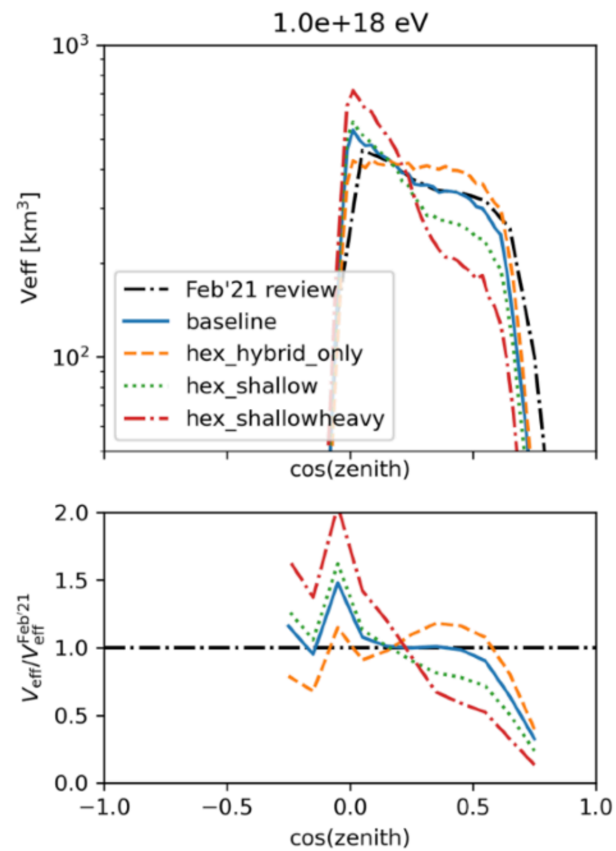


IceCube-Gen2

The reference design to move forward





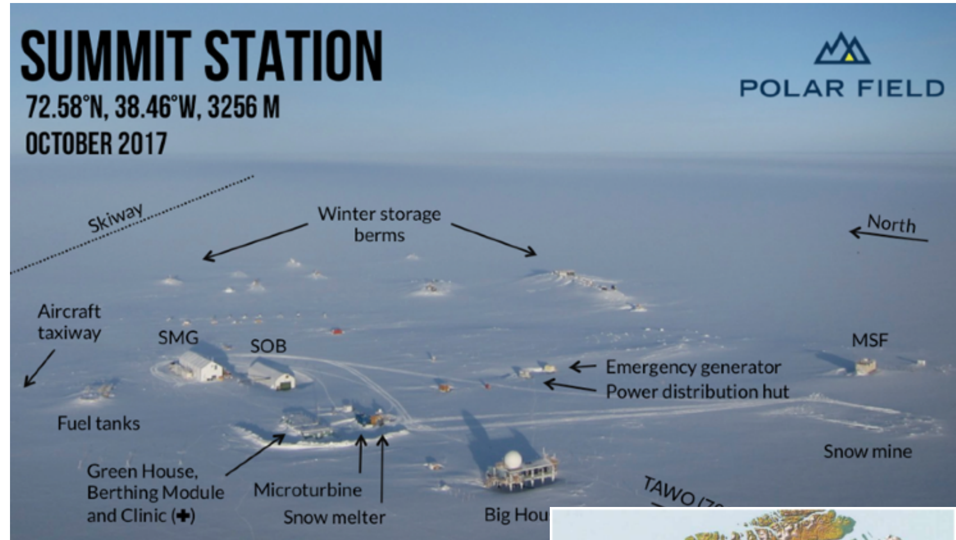


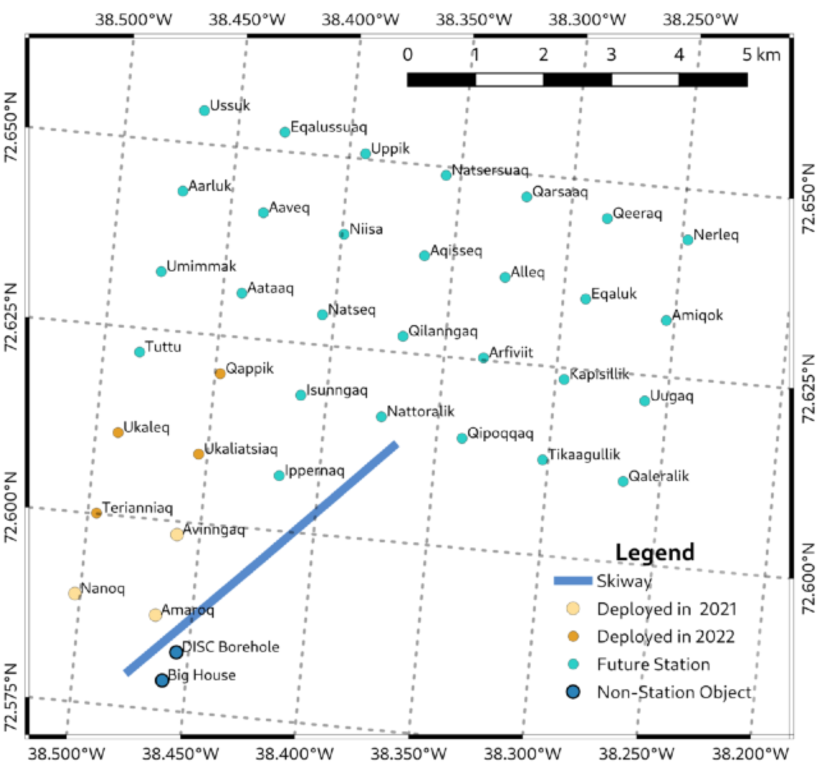
Punchline: After A LOT of CPU hours, it seems that our back-of-the-enevelope calculations weren't so bad after all — only now everyone feels better about them!

So what are we doing for now?

Getting data for all this software

Radio Neutrino Observatory

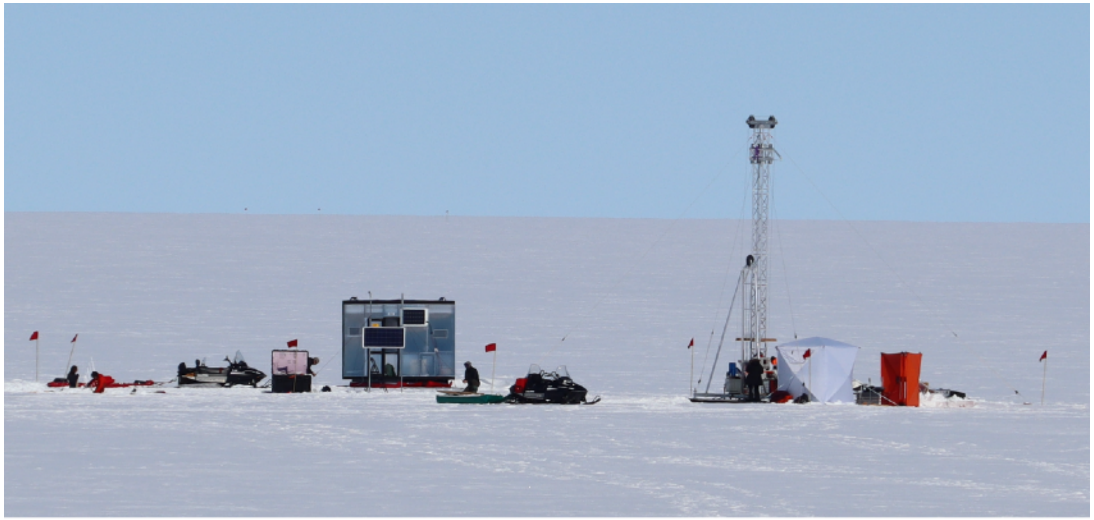


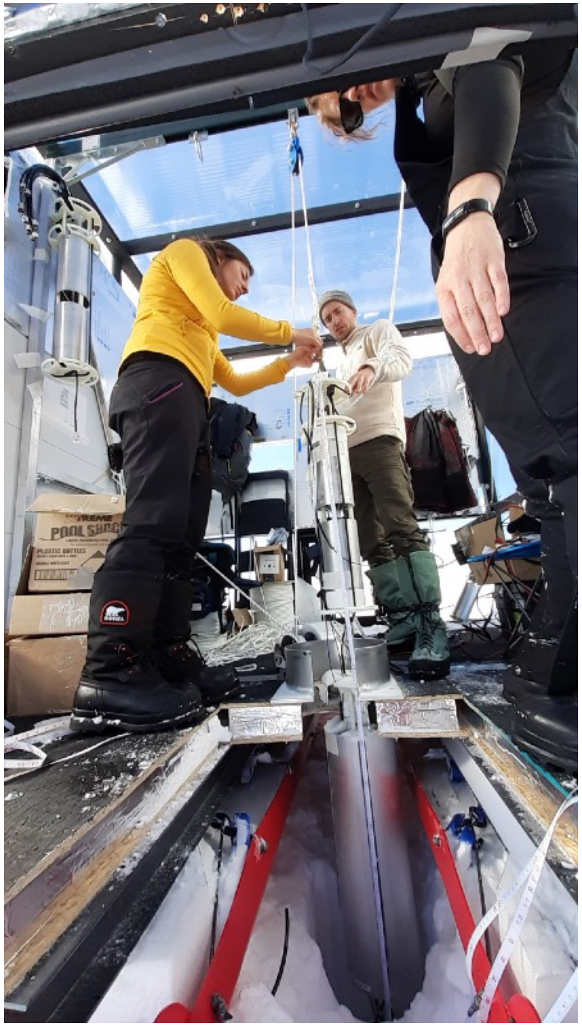


- After lots of proof-of principle experiments: first scale-up to large array

The RNO-G approach

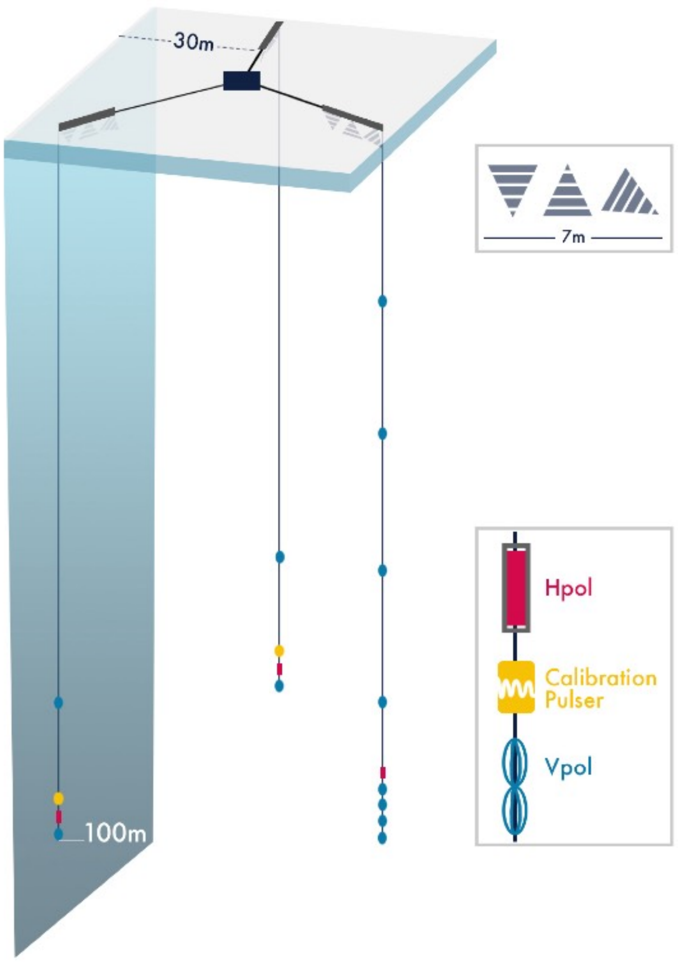
What we are building





The RNO-G approach

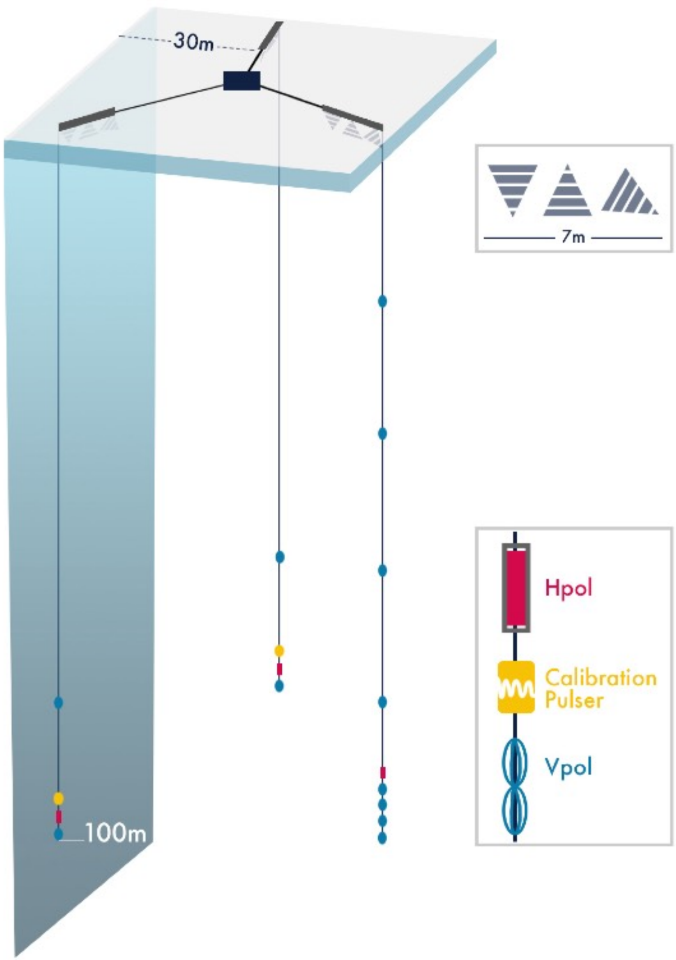
The single components



- Log-periodic dipole antennas (**LPDA**) at the surface:
- High-gain antennas with very good response to neutrino signals, but too big to fit in a hole
- At the surface subject to ray-bending = not all trajectories reach these antennas
- Antennas at the surface also act as cosmic ray veto
- 3x3 antennas to detect all arrival directions and polarizations

The RNO-G approach

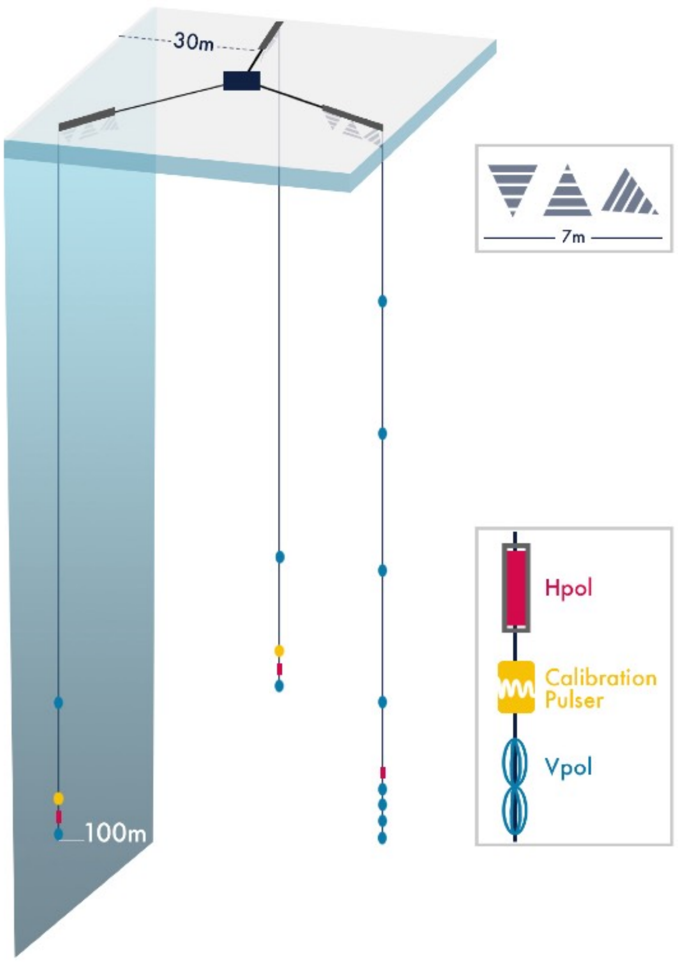
The single components



- Bicone antennas and quad-slot antennas in 100 meter deep holes
- the deeper the better (ray shadowing)
- 100 meters achievable with a fast mechanical drill (cheap)
- two different types of antennas to cover all polarizations
- small antennas have less gain and are typically less broad-band

The RNO-G approach

The single components

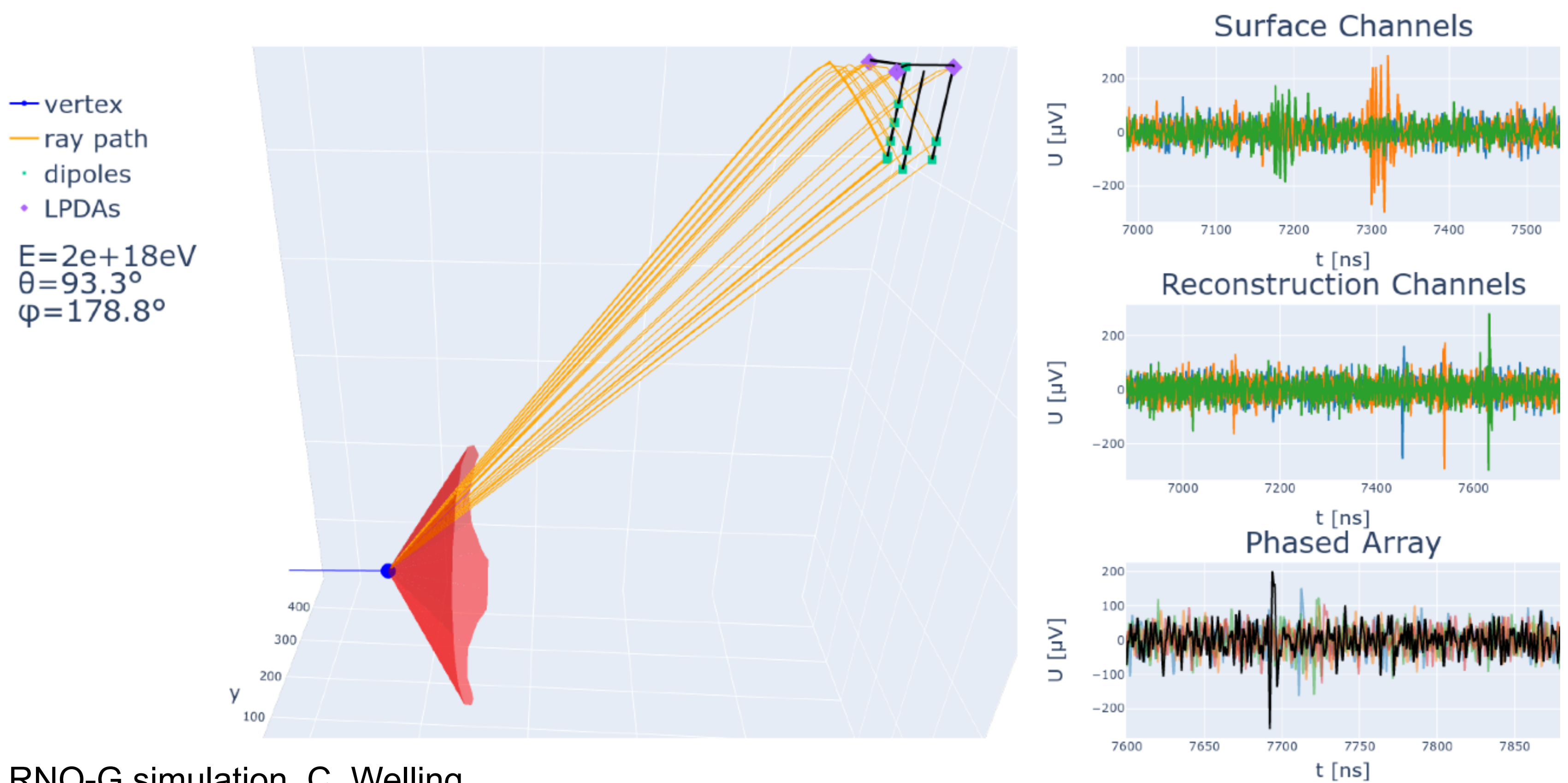


- Station geometry:
- Three strings to reconstruct arrival direction
- One string with many antennas to make the reconstruction of the vertex distance a one-dimensional problem
- String also hosts the phased array trigger
- The lower the threshold the better the sensitivity

The RNO-G approach

More on Friday!

What do the signals look like



Full RNO-G simulation, C. Welling

Software development:
Always make nice visualization tools, increases the number of users

Conclusions

Scientific Software development

- It is not that different from industry software development, but we are all not trained for that
- Additional hurdle: we don’t know what science will be important later and a first iteration always makes assumptions
- Bad software should no longer be acceptable: it is a time sink and precludes scientific results