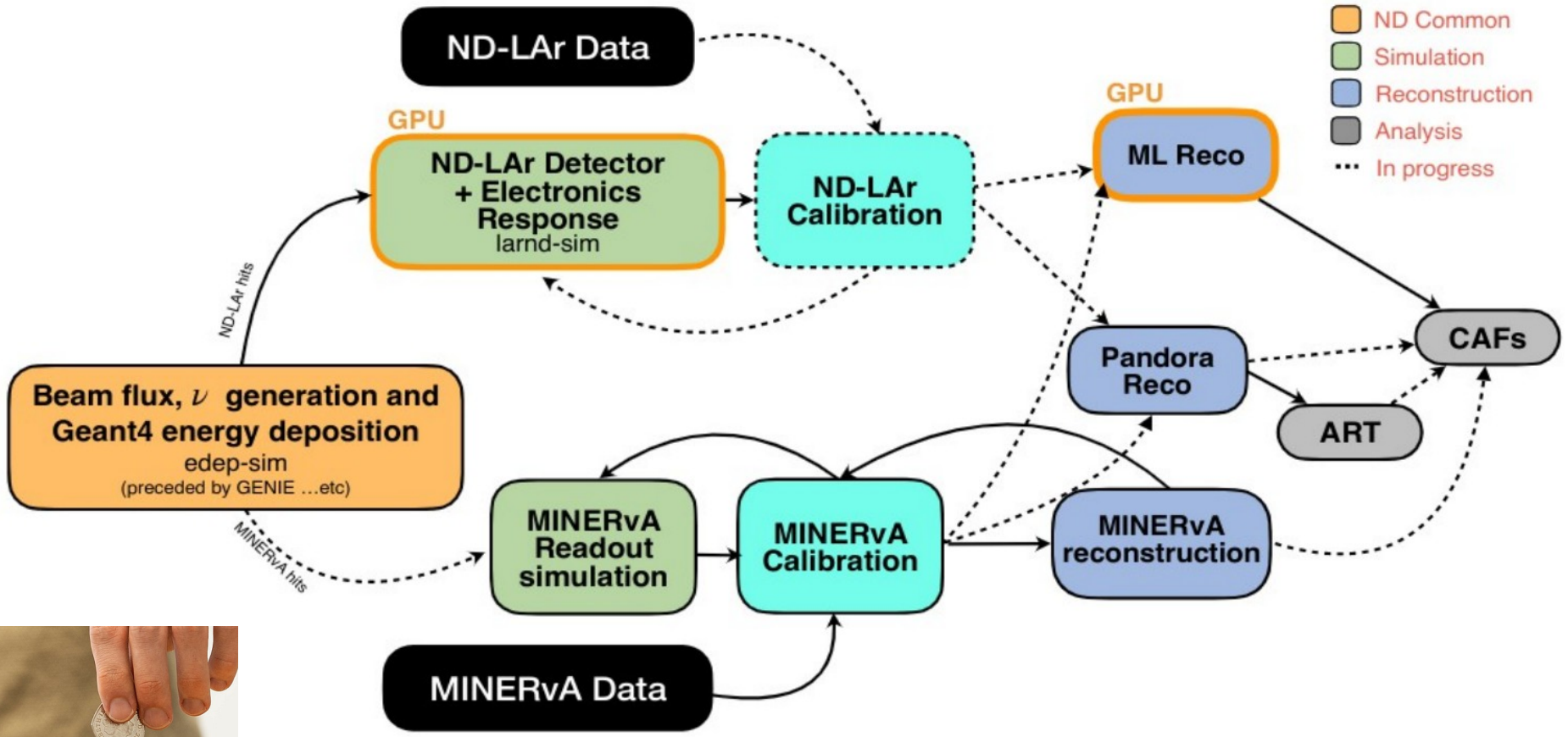
CAFs & analysis

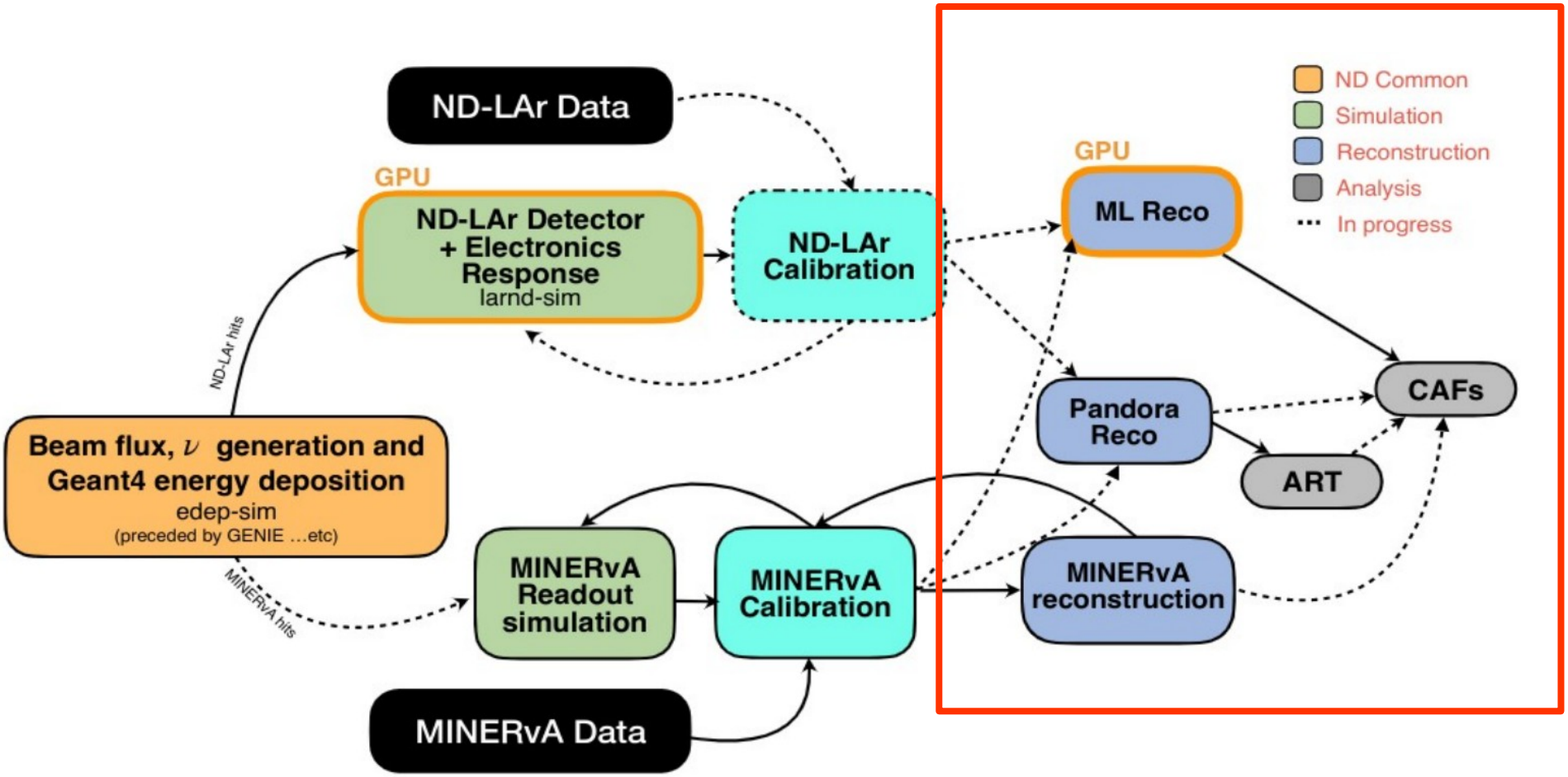
Jeremy Wolcott
Tufts University

2x2 workshop
University of Bern
Jan. 21, 2023

A knotty path

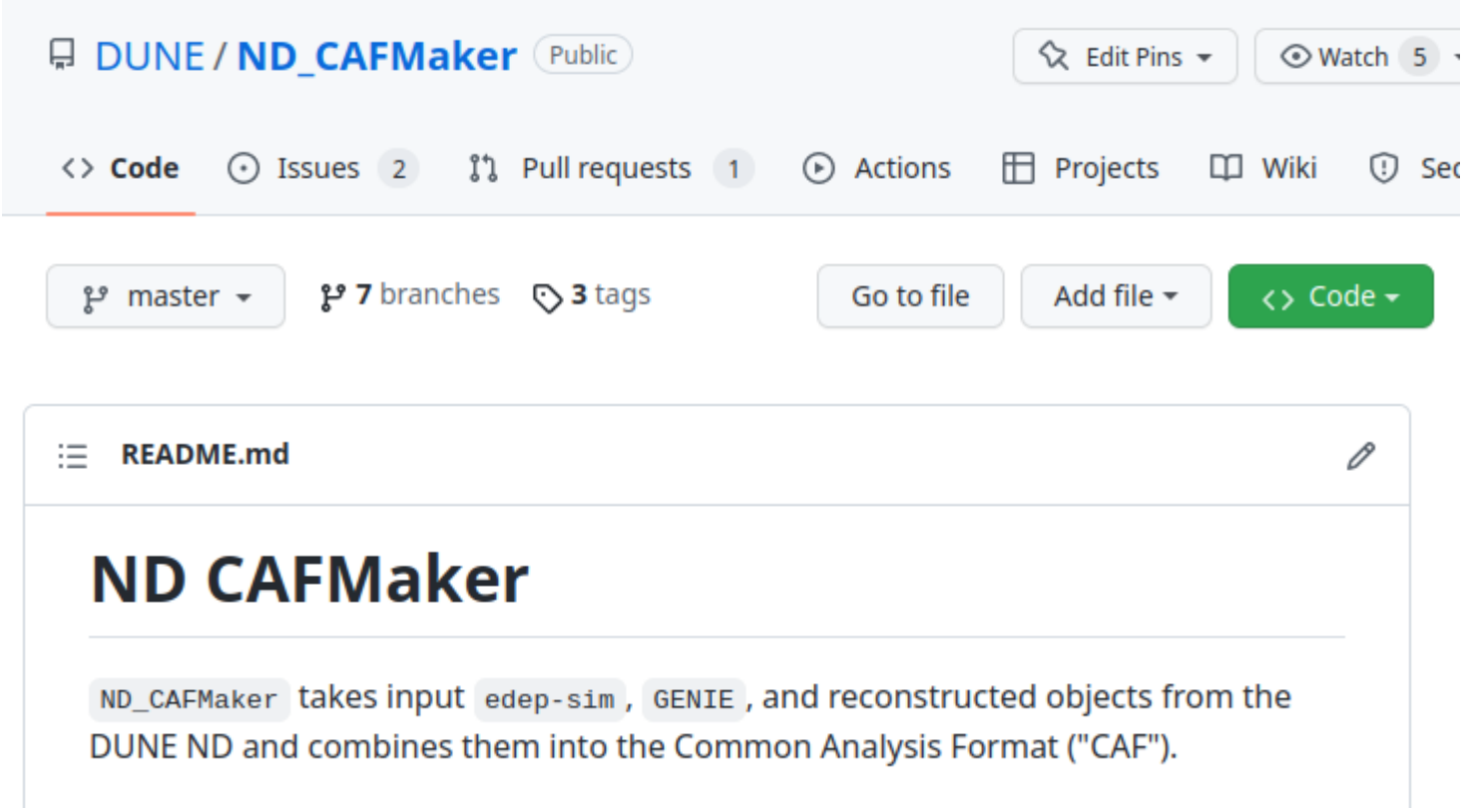


CAF-making



In this talk I'm focusing on the CAF-making pathway.

How is ~~babby~~ CAF formed?

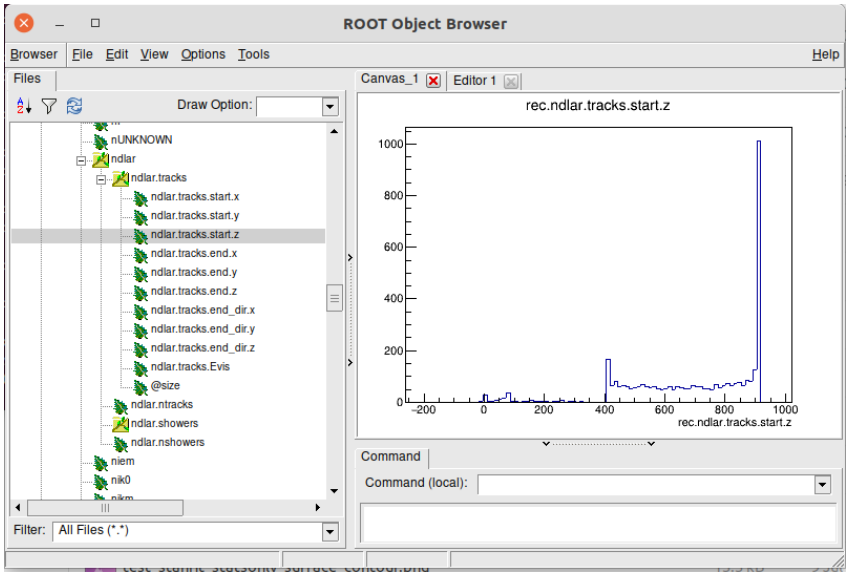


The ND CAFMaker writes out CAFs.

This is a *shared* tool amongst all ND groups and LBL
(originally built by LBL for FD TDR,
and has somehow become my ~~problem~~ responsibility?)

What is a “CAF”?

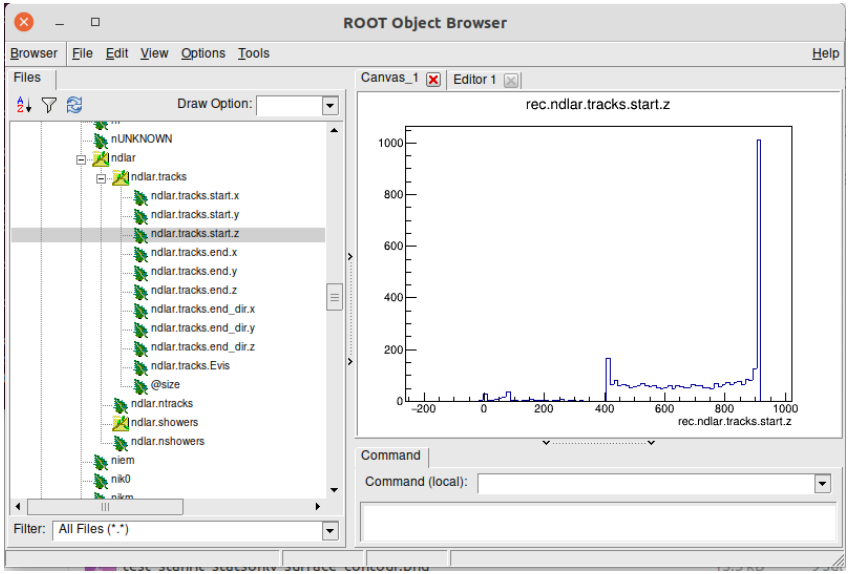
- CAFs are input to LBL analysis
  - “Common Analysis Files,” which are ROOT format trees based on custom StandardRecord object (more on that shortly)



  - Contain *summaries* of events: higher-level reconstructed objects & truth information
    - Goal: fast iteration in analysis. (More propaganda at arXiv:2203.13768)
- CAFs are intended to have low barrier-to-entry and be easy to use
  - I showed an example ν_μ CC energy estimator based on the ML reco reconstruction, with accompanying “howto”, in Dec. 2021
  - The TMS group has demonstrated matching ND-LAr to TMS with CAFs as well

What is a “CAF”?

- CAFs are input to LBL analysis
  - “Common Analysis Files,” which are ROOT format trees based on custom StandardRecord object

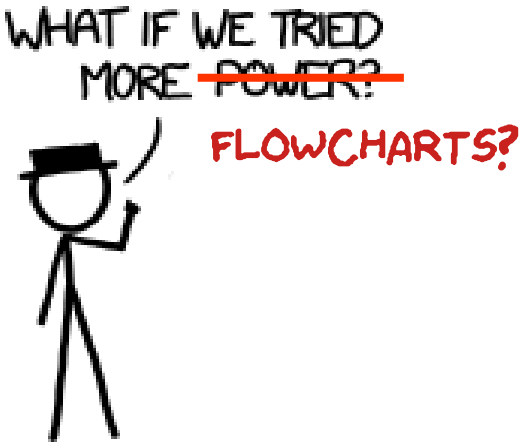


where is this utopia??

  - Contain *summaries* of events: higher-level reconstructed objects & truth information
    - Goal: fast iteration in analysis. (More propaganda at arXiv:2203.13768)
- CAFs are intended to have low barrier-to-entry and be easy to use
  - I showed an example ν_μ CC energy estimator based on the ML reco reconstruction, with accompanying “howto”, in Dec. 2021
  - The TMS group has demonstrated matching ND-LAr to TMS with CAFs as well

Problem #1:
file format handshaking

To explain problem #1:

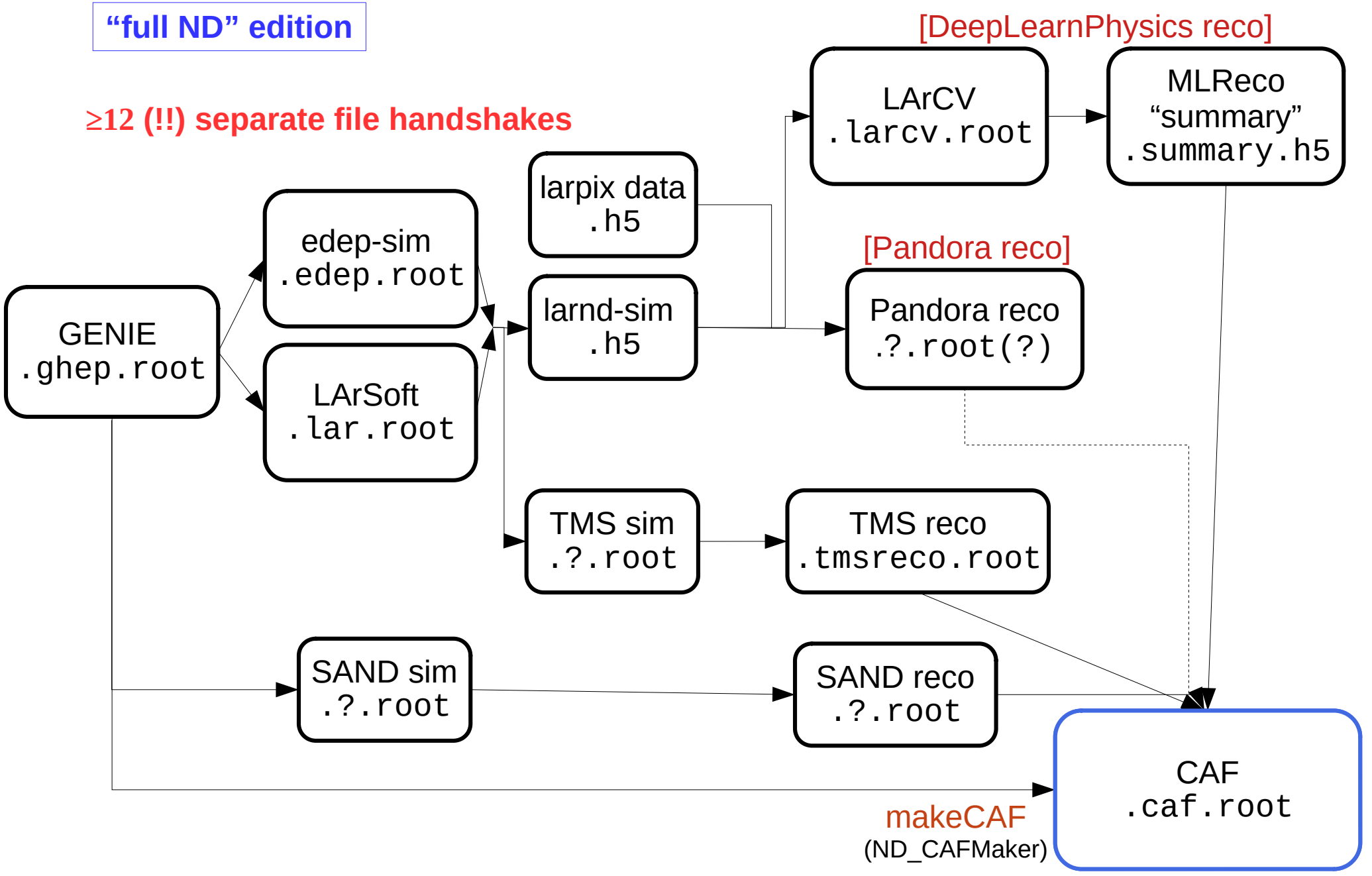


[https://what-if.xkcd.com/13/]

Problem #1: file format handshaking

“full ND” edition

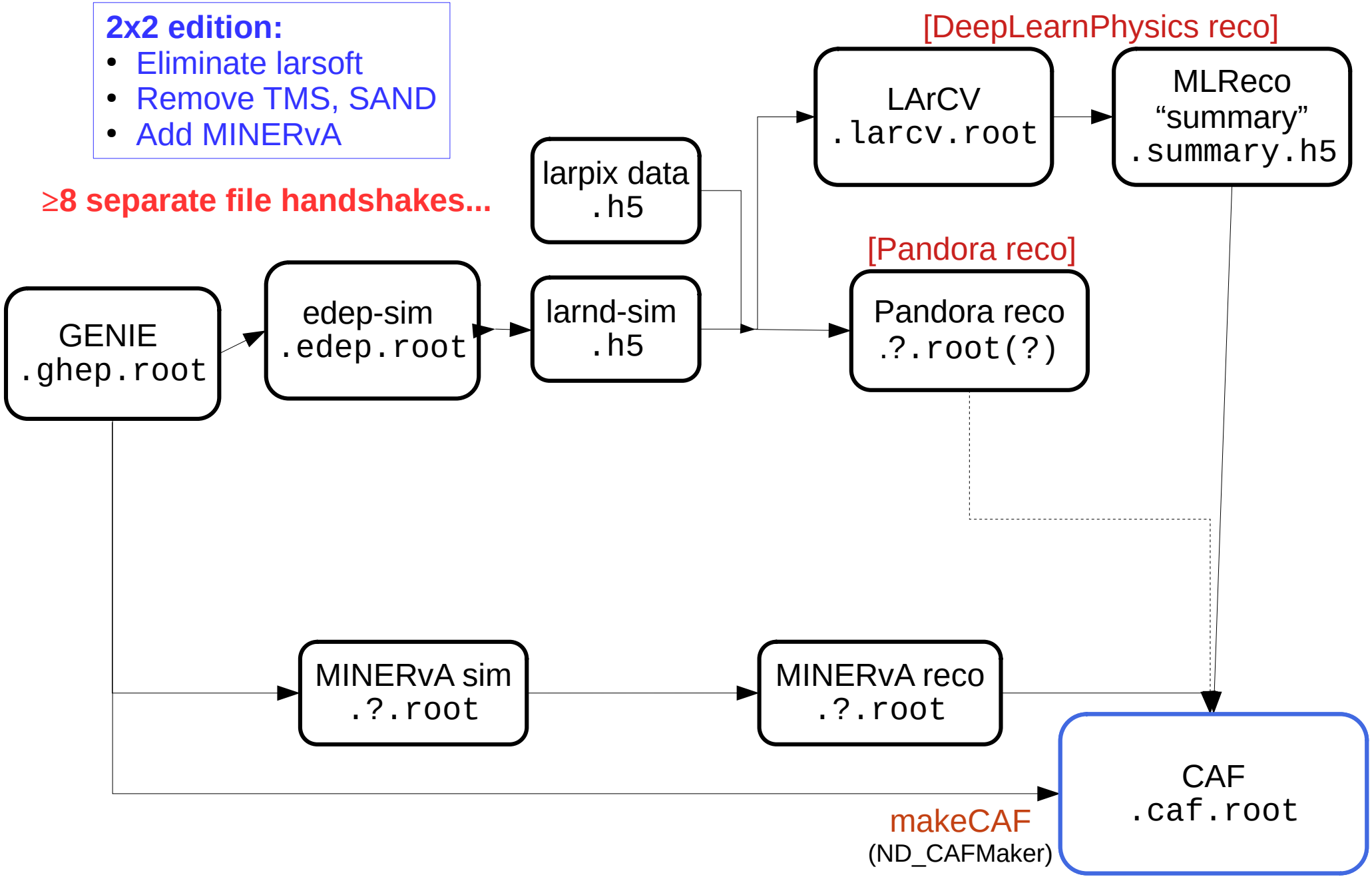
≥12 (!!) separate file handshakes



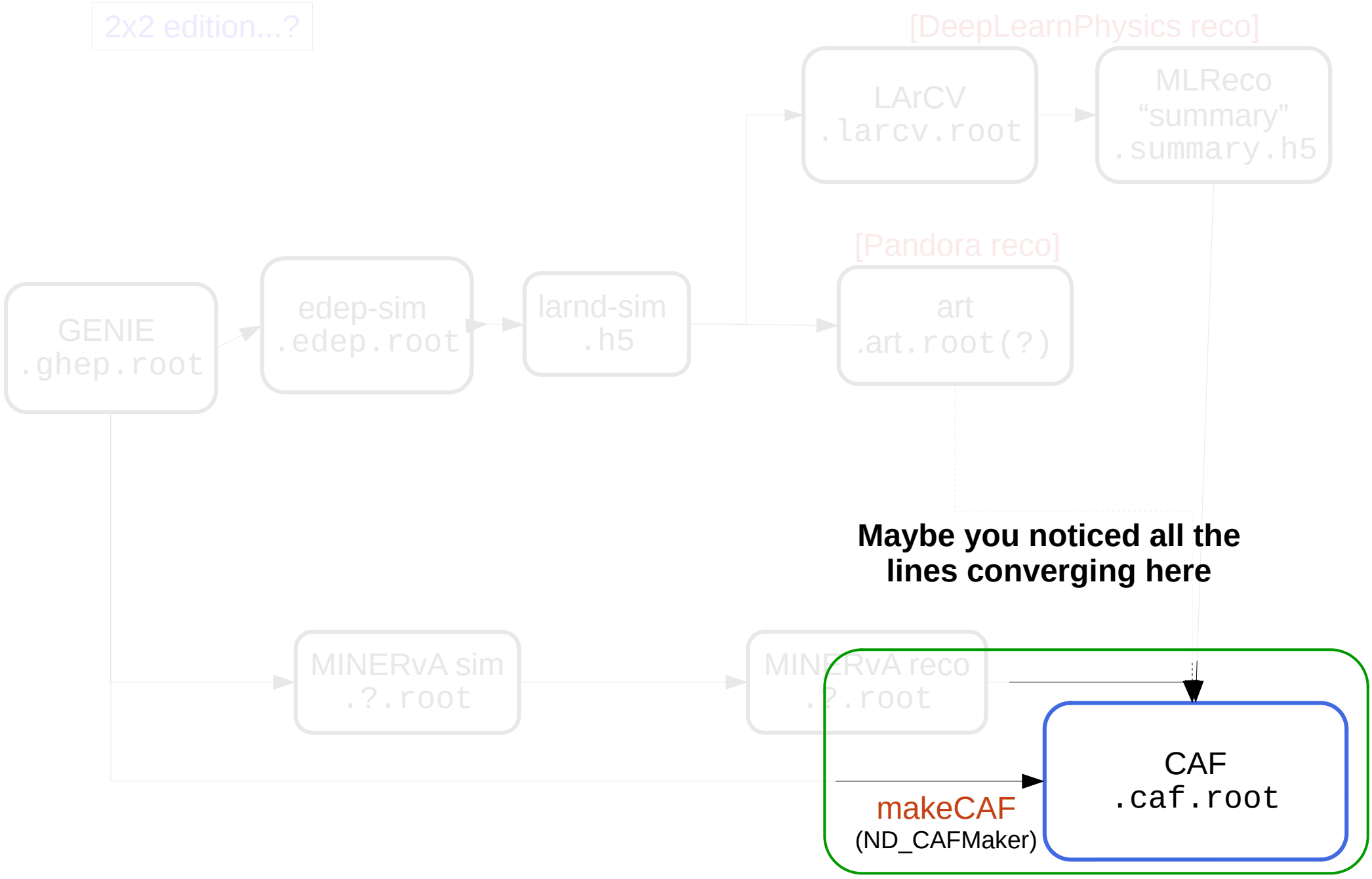
Problem #1: file format handshaking

2x2 edition:
- Eliminate larsoft
- Remove TMS, SAND
- Add MINERvA

≥8 separate file handshakes...



Problem #1: file format handshaking



Problem #1: file format handshaking

DUNE / **ND_CAFMaker** Public

Edit Pins | Watch 5

Code | Issues 2 | Pull requests 1 | Actions | Projects | Wiki | Sec

master | 7 branches | 3 tags | Go to file | Add file | Code

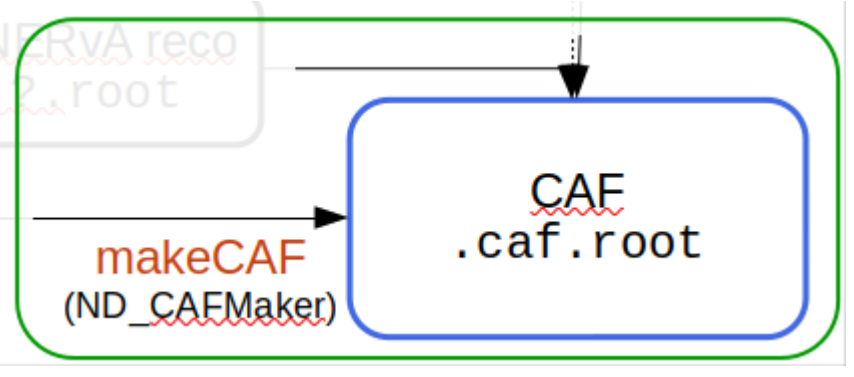
README.md

ND CAFMaker

`ND_CAFMaker` takes input `edep-sim`, `GENIE`, and reconstructed objects from the DUNE ND and combines them into the Common Analysis Format ("CAF").

The ND CAFMaker synthesizes all these inputs together and writes out CAFs.

It's a “framework-less” (standalone) C++ tool that only “natively” depends on `fhicl-cpp` (configuration format) and `duneanaobj` (output format – more momentarily)



Problem #1: file format handshaking

```
std::vector<std::unique_ptr<cafmaker::IRecoBranchFiller>> recoFillers;

// first: we do SAND or ND-LAr reco
std::string ndlarFile;
std::string sandFile;
if(par().cafmaker().ndlarRecoFile(ndlarFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::MLNDLArRecoBranchFiller>(ndlarFile));
else if (par().cafmaker().sandRecoFile(sandFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::SANDRecoBranchFiller>(sandFile));

// next: did we do TMS reco?
std::string tmsFile;
if (par().cafmaker().tmsRecoFile(tmsFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::TMSRecoBranchFiller>(tmsFile));

// if we did both ND-LAr and TMS, we should try to match them, too
if (!ndlarFile.empty() && !tmsFile.empty())
  recoFillers.emplace_back(std::make_unique<cafmaker::NDLArTMSMatchRecoFiller>());
```

Inputs

The package is controlled by `fhicl` config files, found in the `cfg` directory. The `cfg/ndcafmakerjob.fcl` shows the basic setup.

For the minimal test setup:

- Provide an `InputGHEPFile`, which contains the GENIE truth information in `GHEP` format
- Provide an `OutputFile`, where your file will be saved

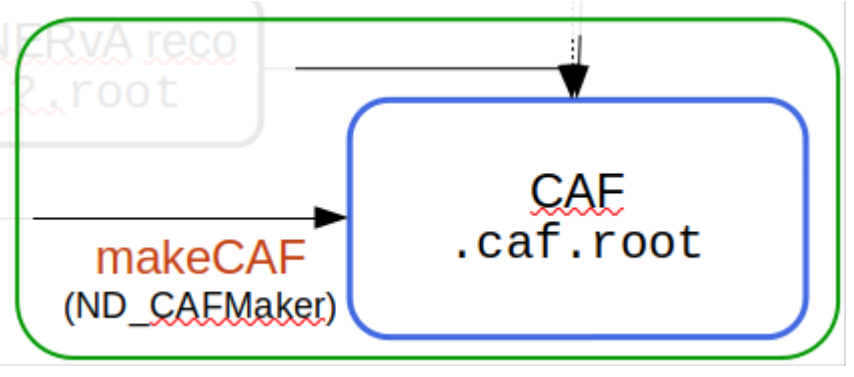
Extending upon the minimal test setup you can:

- Provide a `NDLArRecoFile`, which contains the output of the ND LAr reconstruction
- Provide a `TMSRecoFile`, which contains the output of the TMS reconstruction
- Provide a `SANDRecoFile`, which contains the output of the SAND reconstruction

To add variables and inspect what is set and how, check `src/Params.h`.

ND CAFMaker has a “pluggable” architecture that simplifies adding new reco “branch fillers”.

It actually works ok!



Problem #1: file format handshaking

```
std::vector<std::unique_ptr<cafmaker::IRecoBranchFiller>> recoFillers;

// first: we do SAND or ND-LAr reco
std::string ndlarFile;
std::string sandFile;
if(par().cafmaker().ndlarRecoFile(ndlarFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::MLNDLArRecoBranchFiller>(ndlarFile));
else if (par().cafmaker().sandRecoFile(sandFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::SANDRecoBranchFiller>(sandFile));

// next: did we do TMS reco?
std::string tmsFile;
if (par().cafmaker().tmsRecoFile(tmsFile))
  recoFillers.emplace_back(std::make_unique<cafmaker::TMSRecoBranchFiller>(tmsFile));

// if we did both ND-LAr and TMS, we should try to match them, too
if (!ndlarFile.empty() && !tmsFile.empty())
  recoFillers.emplace_back(std::make_unique<cafmaker::NDLArTMSMatchRecoFiller>());
```

Inputs

The package is controlled by `fhicl` config files, found in the `cfg` directory. The `cfg/ndcafmakerjob.fcl` shows the basic setup.

For the minimal test setup:

- Provide an `InputGHEPFile`, which contains the GENIE truth information in `GHEP` format
- Provide an `OutputFile`, where your file will be saved

Extending upon the minimal test setup you can:

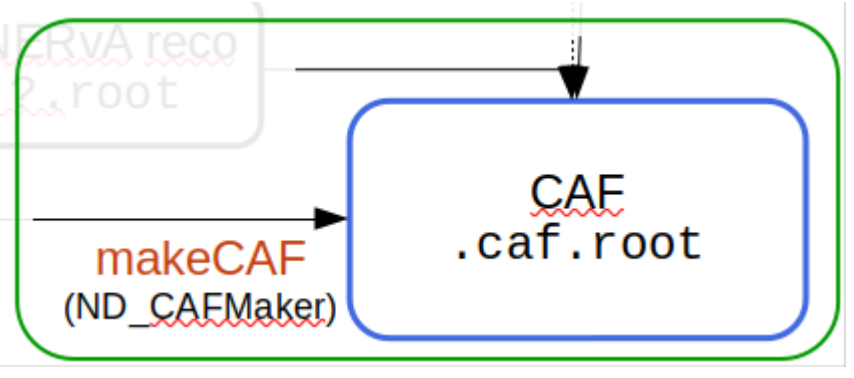
- Provide a `NDLArRecoFile`, which contains the output of the ND LAr reconstruction
- Provide a `TMSRecoFile`, which contains the output of the TMS reconstruction
- Provide a `SANDRecoFile`, which contains the output of the SAND reconstruction

To add variables and inspect what is set and how, check `src/Params.h`.

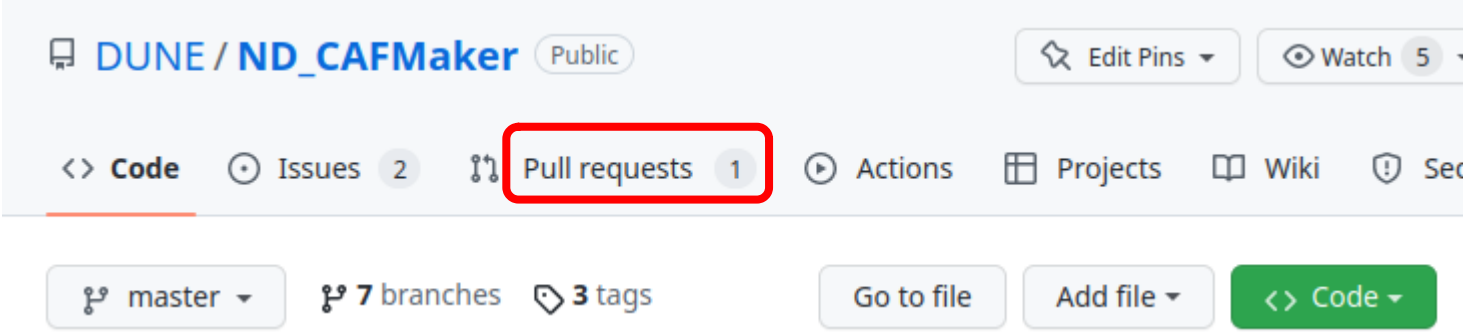
However...

Library dependencies required to read all the input formats (ROOT, hdf5, GENIE, ~~edep-sim~~...) **make building the CAFMaker a chore**

[Problem #1a]



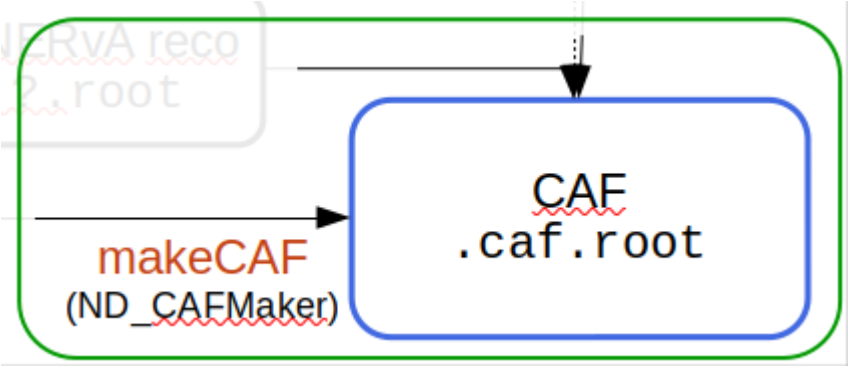
Problem #1: file format handshaking



Another big problem (Problem #1b) here is "ownership":
who is supposed to sign off on changes
proposed by any of the providers of inputs
(ND-LAr, 2x2/MINERvA, SAND, TMS)**?**
(and tag releases, coordinate w/ production, ...)

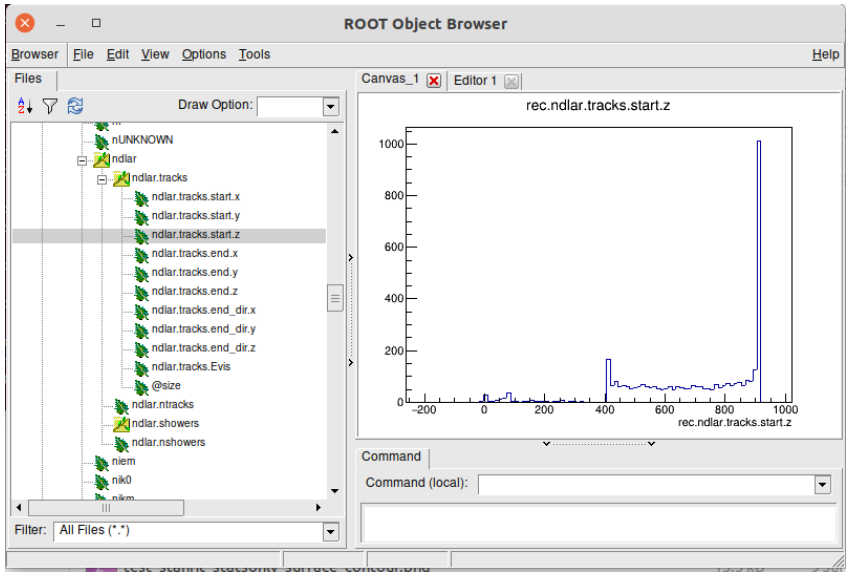
Right now, it's just ... me, plus whoever else I can get to pay attention 😭

→ **needs well-delineated responsibilities/succession plan**
(depends on sufficient expertise!)



Problem #2:
output & usage

Problem #2: output & usage



CAFs contain a series of StandardRecord objects (one per event)

Though they're currently a mess (problem #2.5, I guess), the vision is for them to be organized hierarchically so info is easy for the uninitiated to find & understand. (ND branches are!)

caf::StandardRecord Class Reference

The StandardRecord is the primary top-level object in the Common Analysis File trees.

```
#include <StandardRecord.h>
```

Public Member Functions

StandardRecord ()
~StandardRecord ()
StandardRecord ()

Public Attributes

int meta_run
int meta_subrun
double pot
float eRec_FromDep
float Ev_reco
float Ev_reco_nue
float Ev_reco_numu
float mvaresult
float mvapue
⋮
float RecoLepEnNumu
float RecoHadEnNumu
double pileup_energy
SRNDBranch nd
int RecoMethodNue
int RecoMethodNumu
int TrackMomMethodNumu
int reco_numu
⋮

caf::SRNDLAr Class Reference

ND-LAr reconstruction output. More...

```
#include <SRNDLAr.h>
```

Public Attributes

std::vector< SRTrack > tracks
std::size_t ntracks = 0
std::vector< SRShower > showers
std::size_t nshowers = 0

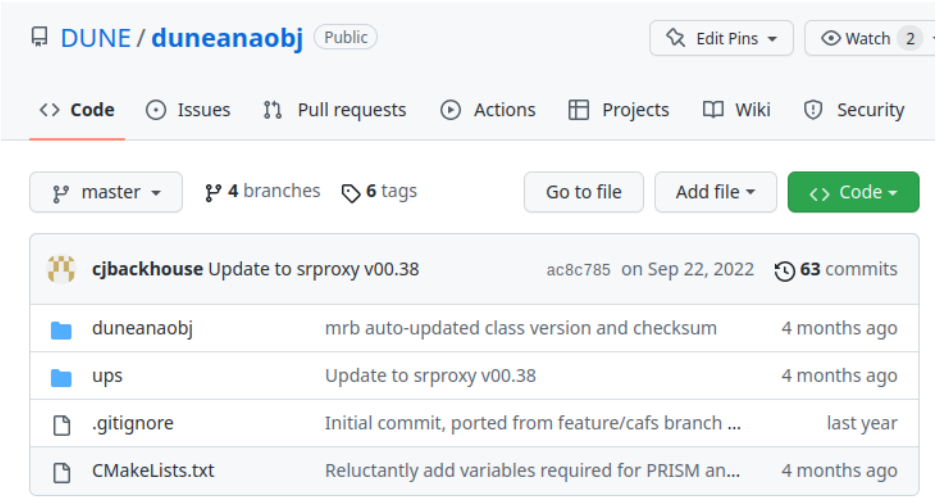
caf::SRNDBranch Class Reference

```
#include <SRNDBranch.h>
```

Public Attributes

SRNDLAr lar
SRGAr gar
SRTMS tms
std::size_t ntrkmatch = 0
std::vector< caf::SRNDTrackAssn > trkmatch

Problem #2: output & usage



The StandardRecord format is defined in (yet another) DUNE GitHub repository called duneanaobj

caf::StandardRecord Class Reference

The StandardRecord is the primary top-level object in the Common Analysis File trees.

```
#include <StandardRecord.h>
```

Public Member Functions

- StandardRecord ()
- ~StandardRecord ()
- StandardRecord ()

Public Attributes

- int meta_run
- int meta_subrun
- double pot
- float eRec_FromDep
- float Ev_reco
- float Ev_reco_nue
- float Ev_reco_numu
- float mvaresult
- float mvanue
- ⋮
- float RecoLepEnNumu
- float RecoHadEnNumu
- double pileup_energy
- SRNDBranch nd
- int RecoMethodNue
- int RecoMethodNumu
- int TrackMomMethodNumu
- int reco_numu
- ⋮

caf::SRNDLAr Class Reference

ND-LAr reconstruction output. More...

```
#include <SRNDLAr.h>
```

Public Attributes

- std::vector< SRTrack > tracks
- std::size_t ntracks = 0
- std::vector< SRShower > showers
- std::size_t nshowers = 0

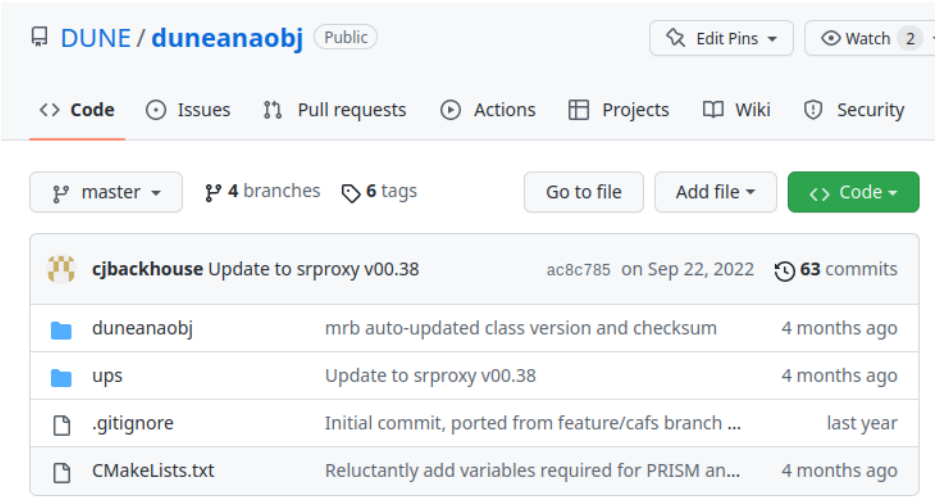
caf::SRNDBranch Class Reference

```
#include <SRNDBranch.h>
```

Public Attributes

- SRNDLAr lar
- SRGAr gar
- SRTMS tms
- std::size_t ntrkmatch = 0
- std::vector< caf::SRNDTrackAssn > trkmatch

Problem #2: output & usage



Problem #2:

Modifying duneanaobj is absurdly complicated!

(and JW is single point of failure in several places 🙈)

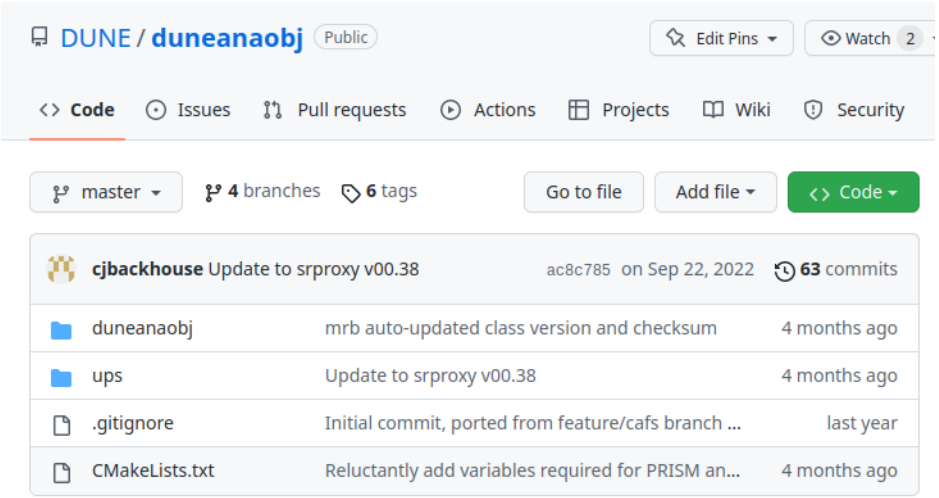
[the complexity of the workflow + single-point-of-failure has inhibited development in TMS, SAND as well as Pandora CAF integration...]

Approximate workflow for adding / modifying variables:

1. Check out repository, make new branch
2. Make desired changes
3. Discover you can't build it, random walk through people & Slack channels you know until somebody says "**ask Jeremy**"
   → JW tells you you need to build on a dunegpvm with UPS set up because ... reasons

4. Check out `ND_CAFMaker`
5. Make desired changes (to fill new vars)
6. Discover that `ND_CAFMaker` build system is very fragile. **Ask Jeremy** why it doesn't work
   → JW tells you that your default gpvm setup probably sets up something incompatible
7. Discover that no matter how hard you try `ND_CAFMaker` always gets `duneanaobj` from UPS instead of your edits. **Ask Jeremy** again
   → JW tells you you need an elaborate setup where you make a UPS package in your local `/dune/app` area and use that
8. Finally make a test CAF, fix your bugs, etc.
9. Make pull requests to `ND_CAFMaker`, `duneanaobj` with your changes
10. Try to get people to review your PRs. Eventually **JW** gets sufficiently fed up that he **unilaterally rubber-stamps them** (see Problem #1b)
11. Wait until Tom Junk and Lynne Garren have time to build & install a new release of duneanaobj
12. Profit ...?

Problem #2: output & usage



Problem #2:

Modifying duneanaobj is absurdly complicated!

(and JW is single point of failure in several places 🙈)

[the complexity of the workflow + single-point-of-failure has inhibited development in TMS, SAND as well as Pandora CAF integration...]

The major reason for all this is because UPS is the main dependency manager for ND_CAFMaker right now.

Maybe we need a proper ND_CAFMaker build system that *can* get dependencies from UPS for its “batteries-included” Production version**, but doesn't *have* to**
(and can operate without all the dependencies for input formats if they're not available, for testing)

Problem #2: output & usage

- Possible ingredients for addressing problem #2:
  - Document the steps on previous page better
    - I'm going to try to do this during working time today
  - Clean up ND_CAFMaker dependency & build system. Add limited-functionality, “UPS-free” build paths for duneanaobj and ND_CAFMaker for testing purposes?
    - Requires some CMake know-how
    - Dependencies of ND_CAFMaker are nontrivial
  - Designate responsible parties for approving PRs
  - Build more expertise (will be needed to get production workflow fully integrated anyways)

[more of the same]

... four consecutive walls of text

What does all this buy us? (I)

- Reading CAFs **requires only ROOT and duneanaobj**, no “frameworks”
  - Straightforward HEP analysis without extensive art, h5py, or other “domain-specific” experience
- ND_CAFMaker is **already set up to integrate multiple datastreams**
  - Should be straightforward to adapt for “2x2 + MINERvA” use case
- Some CAFMakers (there's a draft PR for ND_CAFMaker) can also emit **“flat” CAFs**:
  - Flat ROOT trees that have hierarchical organization but **don't *require* duneanaobj to read them**
  - CAFAna (see next slide) can rebuild StandardRecords from FlatCAFs on-the-fly, completely invisibly, using SRProxy – so you still access them via the hierarchical organization
  - **Read times are** (anecdotally) **competitive with HDF5**
  - Could also use Pythonized tools like Uproot to read into numpy arrays

What does all this buy us? (II)

- **CAFs** (whether “flat” or not) **work naturally with CAFAna** (LBL's toolkit from TDR era), clean-simple-and-fast C++ analysis framework
  - e.g.: make ~30 plots, fit, and evaluate an entire ν_μ CC energy estimator in 4 pretty-easy-to-read C++ files of ~200 lines each (see the `.C` files here) → the entire macro set for the ND-LAr reco APS talk I gave last year
    - n.b.: Not trying to “code golf” here. Just an example of what can be done with ~minimal effort.
  - Simplicity means fewer mistakes, quick analysis turnaround, etc.
  - If ROOT fits your comfort level, learning curve is pretty shallow
  - There's also a Python interface if that's more your thing, though it's not super “pythonic”
- LBL folks **need CAFs to do oscillation analysis** studies for ND TDR timeline, so exercising the pathway here helps us all anyway

What do we need for 2x2?

- Pathway using **ML-reco** is already established (most mature)
  - Uses unofficial code I wrote to run the reco & “summarize” tracks & showers... needs to be re-homed into an official DUNE repo
  - May want to replace the above with more official tools from DeepLearnPhysics, but probably not on (beginning of) 2x2 timescale
- Need to work with **Pandora** folks to make sure Pandora reco output is integrated into CAFs
  - They encountered some of the roadblocks I mentioned previously...
- **Track matching** code for TMS can probably be adapted for MINERvA matching at first, but needs a champion
  - Long-term plan is to integrate MINERvA reco natively into reco rather than happening *post facto*...
- Current **CAFs only have charge info**. We could include a “Flash” data product/summary, but needs a champion
  - Introduces an extra interconnection, too, if light info isn't in the same (.h5) file as the charge

What do we need for the future?

- The StandardRecord needs a thorough cleanup
  - Want to fully “hierarchalize” the arrangement and eliminate the unreadable dumping-ground that the top level currently is
  - Needs to be heavily coordinated w/ LBL
- Consider rewriting ND_CAFMaker using some supported framework that streamlines Production workflows (*art*?)
  - Probably requires a month or two from someone who likes frameworky projects [not JW]
  - c.f. discussion with M. Kirby yesterday
- ND Sim/Reco should manage CAF format and CAF-maker since coordination among all ND users, LBL will be important
  - I expect that proposed additions to the CAF format (both from ND and FD!) will be frequent over the next few years
  - Need to develop streamlined process for reviewing/accepting/versioning