Common Analysis Format in RNTuple

Amit Bashyal, Peter van Gemmeren

On behalf of HEP-CCE/SOP

Argonne National Laboratory

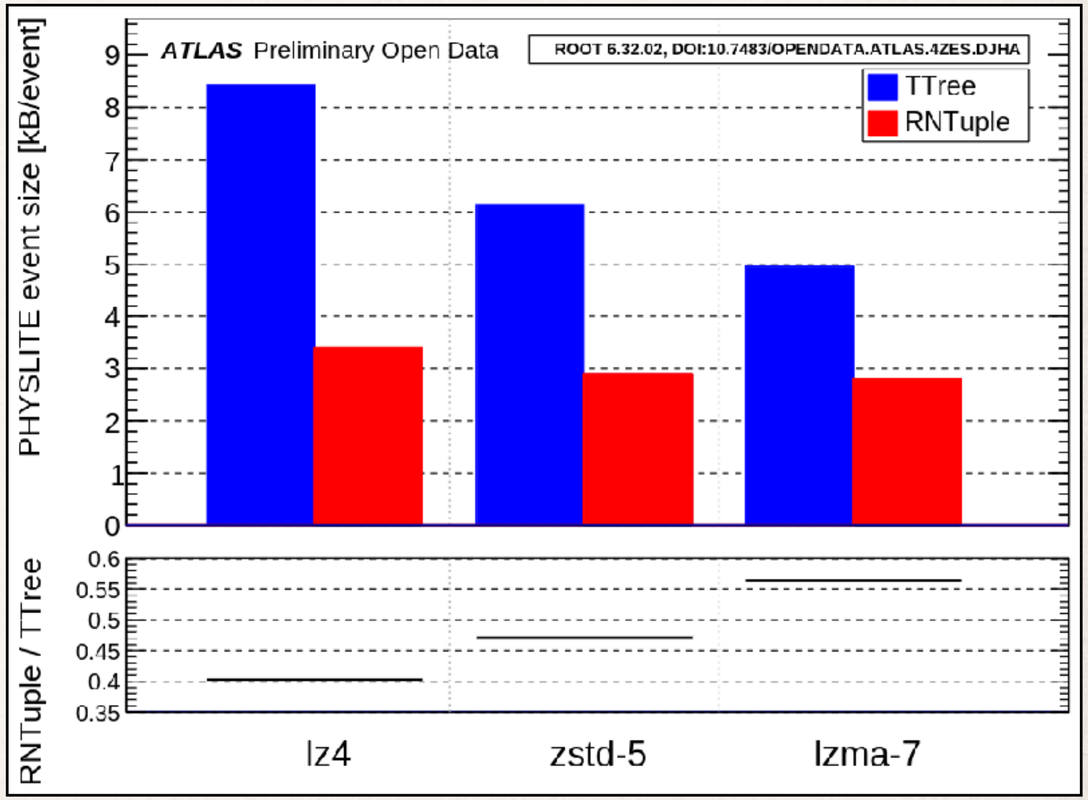
anl.gov/hep-cce

ROOT::TTree (~1995 →~2025)

- Storage backend of ROOT that has enabled HEP experiments to use ROOT ecosystem tools
  - Primary storage backend and I/O subroutine until 2024
  - >2 Exabytes of HEP data stored in TTree Format in over 25 years
  - Evolved to address HEP experimental needs
    - Supports the persistence and I/O of complex HEP experimental data
    - TTree evolution to support HEP experimental needs limits it from adopting modern computing and data analysis standards

ROOT::RNTuple (2025 →)

- Builds on top of 25+ years of TTree experience
- **Will store >10 EB of data by the end of HL-LHC**
  - Optimized I/O peformance with modern storage technologies to enable faster read/write, selective queries in large datasets
  - Success of RNTuple will be huge storage saving compared to TTree
- Adoption of modern C++ gudelines, use of smart pointers with the ability to pass raw ones, separation of read and write APIs for improved I/O
- Forward looking design to handle large sized events and files



ATLAS analysis data in TTree and RNTuple. Taken from CHEP 2024.

ROOT::RNTuple (continued..)

- RNTuple API review conducted by HEP-CCE
  - Review Report
  - RNTuple API is sufficient to address the requirements of large experiments like ATLAS and CMS with few upcoming (minor) improvements
  - **First On-Disk Binary Format release of RNTuple in November 2024**
    - **TTree will be in legacy when DUNE starts taking data**
    - Upcoming experiments (including DUNE) **MUST use RNTuple** to keep current with the ROOT ecosystem

DUNE Computing Requirements

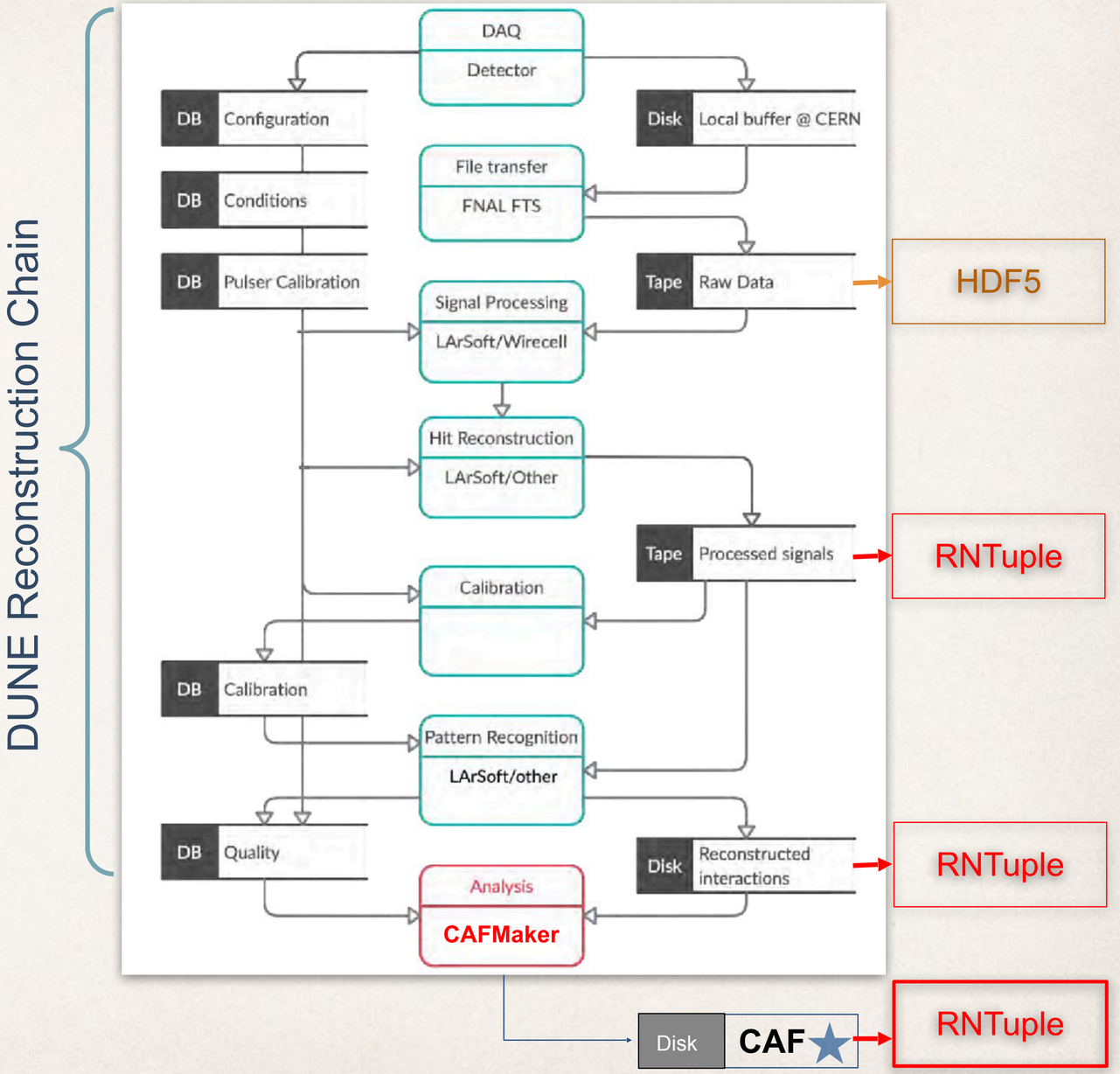
- DUNE →Long Baseline Neutrino Oscillation Experiment with
  - Complex and heterogeneous detector system near neutrino source (ND)
  - Large and homogeneous detector in South Dakota (~800 miles from neutrino source) (FD)
  - Different computing and I/O requirements for detectors at two diferent location
    - FD is homogenous and fine grained → Simple data model but ~GB size raw data from beam trigger
    - ND is small but heterogenous → Complex data model with ~MB size raw data from beam trigger
    - ~TB size readouts for supernova trigger

DUNE and RNTuple

- **RNTuple is the primary storage backend and I/O subsystem during DUNE/HL-LHC era**
  - ATLAS and CMS are ahead of looking at RNTuple (Both can store all their data in RNTuple)
- **DUNE will have to adopt RNTuple to address its computing requirements**
- DUNE is currently designing its Data Processing Framework and Persistence layer
  - RNTuple (and HDF5) for persistence and I/O
  - Opportunity to develop the framework that works with RNTuple which is easier than adapting RNTuple into an existing framework (like ATLAS, CMS)
- Adopt existing data models (including CAF) and I/O infrastructure that will appear in the DUNE experiment

DUNE Reconstruction Chain

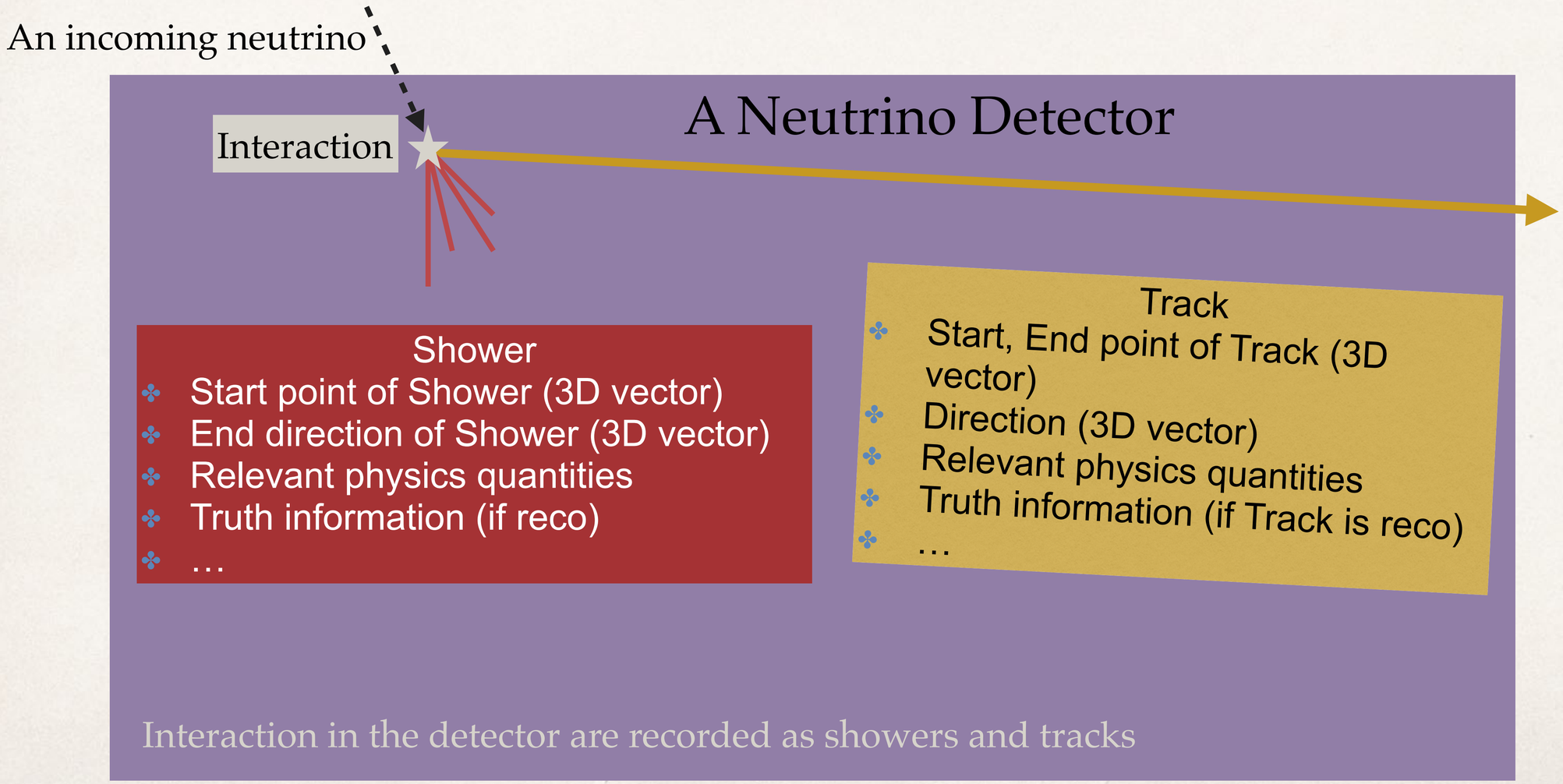
- Raw Data in HDF5
- Reconstructed data will be stored in ROOT and requires RNTuple support for persistency and I/O
- HEP-CCE looked at the persistence and the I/O of CAF Data model (This Talk)



Common Analysis Format (CAF) Data Model

- Neutrino Oscillation experiments record neutrino events in two detectors (one near neutrino source and another at some distance) to measure neutrino oscillations
- CAF takes the fine grained reconstructed data and only saves relevant information needed for oscillation analysis as light weight ntuple
  - Data model is common for both near and far detectors
  - CAF objects as tracks and showers
  - Energy, momentum, relevant generator level information (for simulated events)

A neutrino interaction record in CAF: A (simplified) visual explanation



StandardRecord

caf::StandardRecord Class Reference

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

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

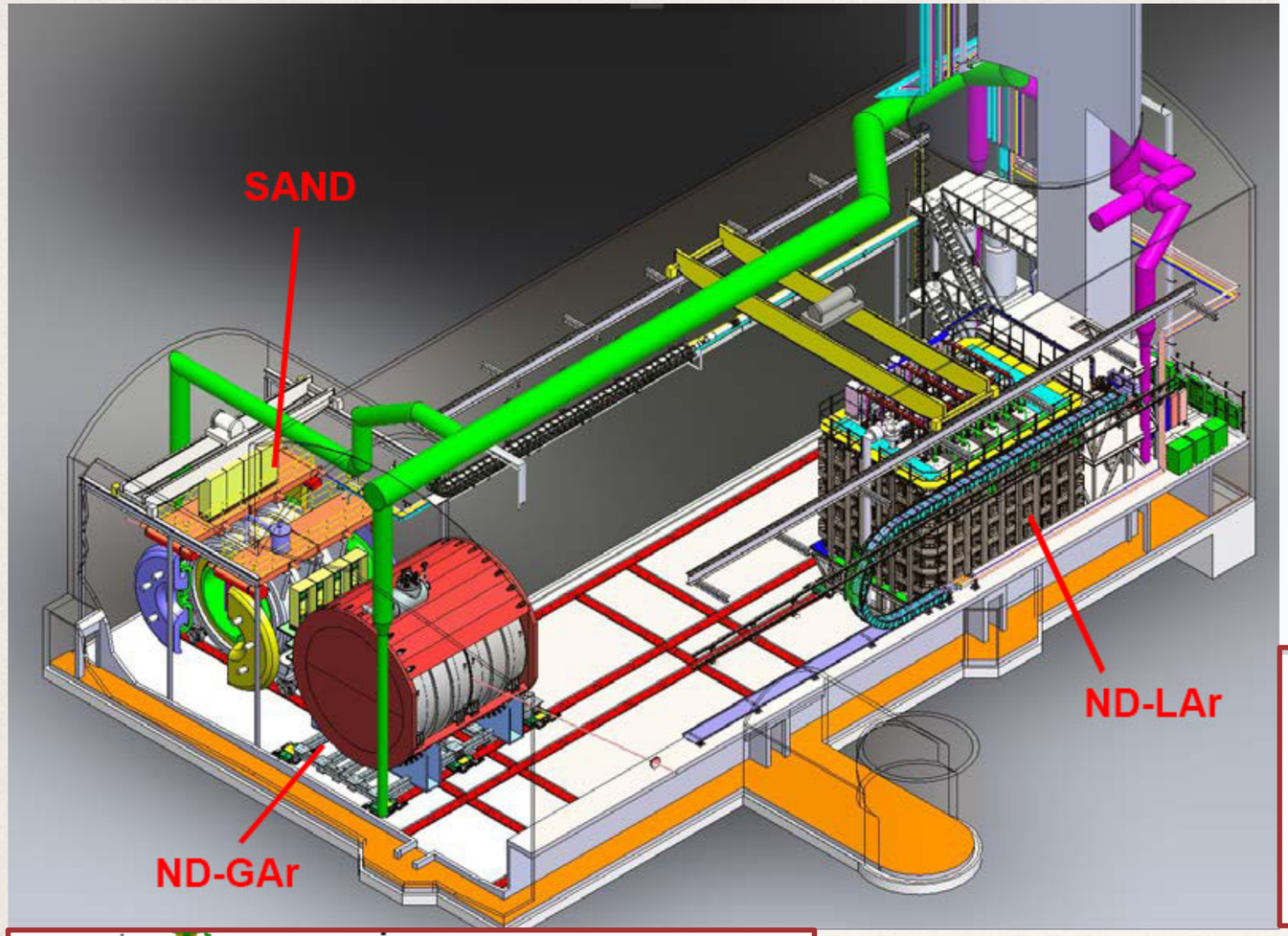
Public Attributes

SRDetectorMetaBranch	**meta** Metadata about the detectors.
SRBeamBranch	**beam** Information about the beam configuration and beam pulse for this event.
SRTruthBranch	**mc** Truth information.
SRCommonRecoBranch	**common** Reconstructed info expected to be common to all (?) detectors.
SRFDBranch	**fd** Reconstructed info unique to the FDs.
SRNDBranch	**nd** Reconstructed info unique to the ND complex.

StandardRecord: Top level CAF object that records neutrino interaction and information related to the interaction including beam, detector condition and generator level information

information related to the neutrino interaction recorded as meta, beam, mc objects.

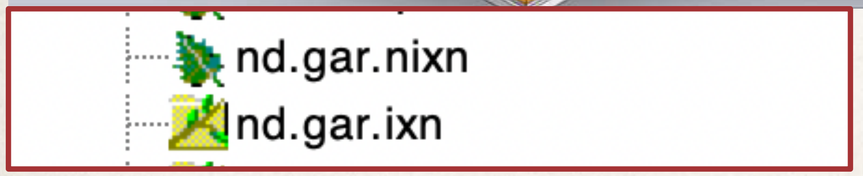
SRFD and SRND branches record the reconstructed interactions in ND and FD detectors as tracks, showers and interactions



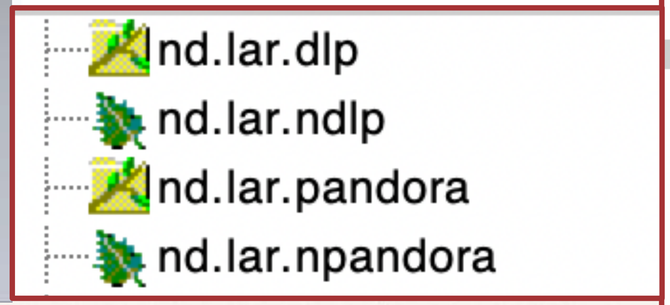
caf::SRNDBranch Class Reference

Public Attributes

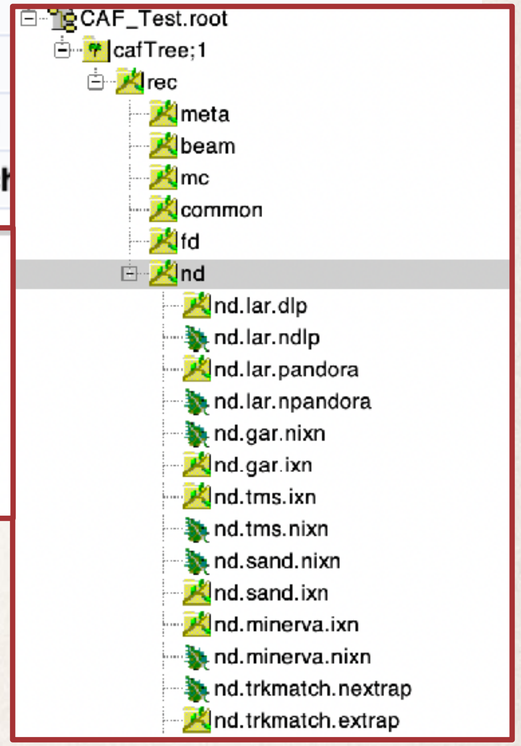
SRNDLAr	lar
SRGAr	gar
SRTMS	tms
SRSAND	sand
SRMINERvA	minerva
caf::SRNDTrkAssnBranch	trkmatch
caf::SRNDShwAssnBranch	shwmatch



Interaction in ND-GAr



Interaction in ND-LAr



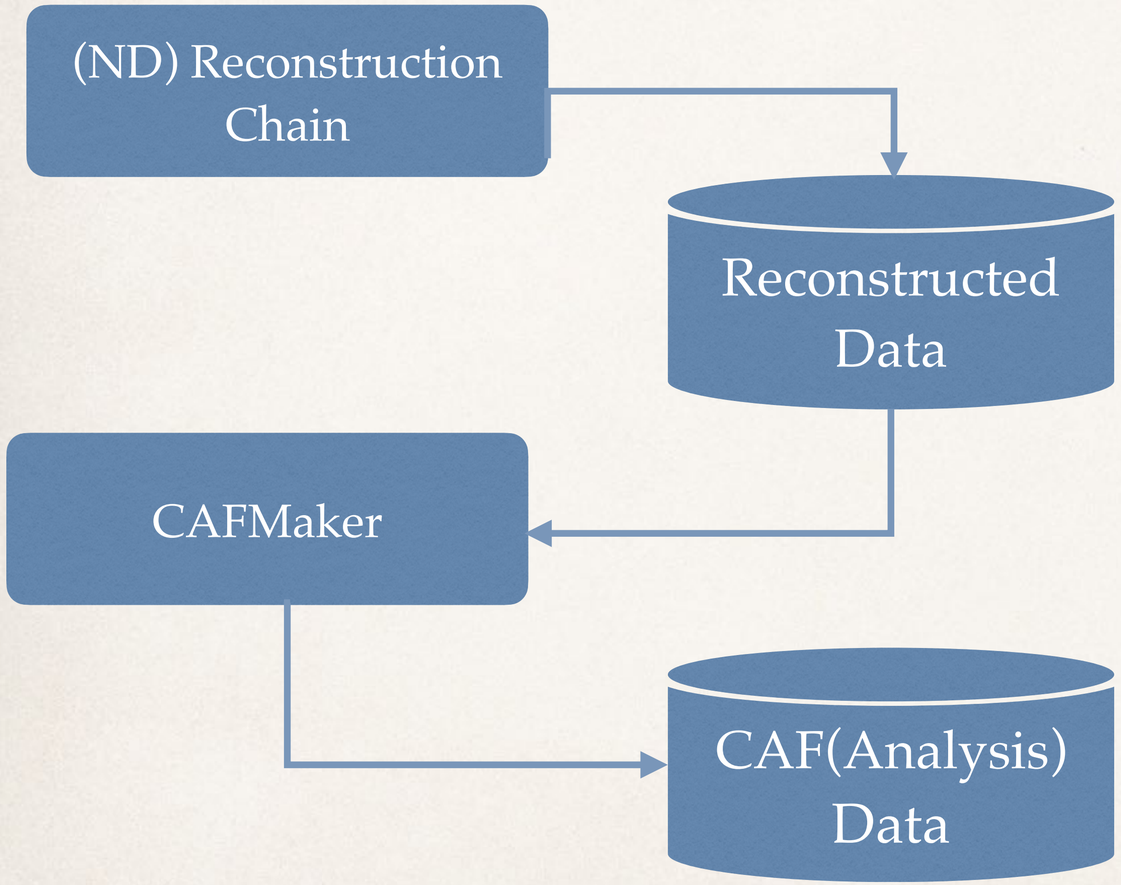
CAF is designed to record interactions in each detectors (illustrated here are detectors of DUNE ND Complex)

Argonne NATIONAL LABORATORY

CAF Objects in RNTuple (Framework)

- As a **HEP-CCE Effort**, a light weight test framework to look at the **persistence and I/O** of CAF Objects in RNTuple
- Some design features of the framework
  - Requires **ROOT** (latest) and **duneanaobj** (CAFObjects)
  - Use gitmodules to download duneanaobj alongside this package
  - **Standalone build of duneanaobj** different from original build which depends on DUNE specific packages
- **Standalone Framework** (support unix and various linux flavor builds)
  - **Tools and examples of I/O of structured CAF** in standalone environment
    - Examples of doing same I/O using TTree and RNTuple for better comparison
  - Reading and writing of CAF object using TTree and RNTuple
  - **Github workflow for CI** to facilitate automated checks on pull requests

CAF Data and I/O Framework



- **CAFMaker to create** CAF Objects in two kinds
  - **Structured**: One StandardRecord object per entry
  - **Flat**: StandardRecord object is "flattened" into basic ROOT types during serialization, structure information maintained in the branch names
- **Separate Reader/Writer helper** for CAF Objects

Framework mimics this workflow in a simplified way in a standalone environment.

Example Test Code Snippets

```
  TFile *cafFile = new TFile(fname, "recreate");
  std::shared_ptr<caf::StandardRecord> sr =
std::make_shared<caf::StandardRecord>();
  TTree *cafSR = new TTree(container_name,"Tree Container to
write StandardRecord Object");
  cafSR->Branch(obj_name,"caf::StandardRecord",sr.get());
```

Example of creating **TTree** container to write CAF Data

```
  auto model = RNTupleModel::Create();
  std::shared_ptr<caf::StandardRecord> field_sr = model-
>MakeField<caf::StandardRecord>(obj_name);

  auto ntuple =
RNTupleWriter::Recreate(std::move(model),container_name,fname);
```

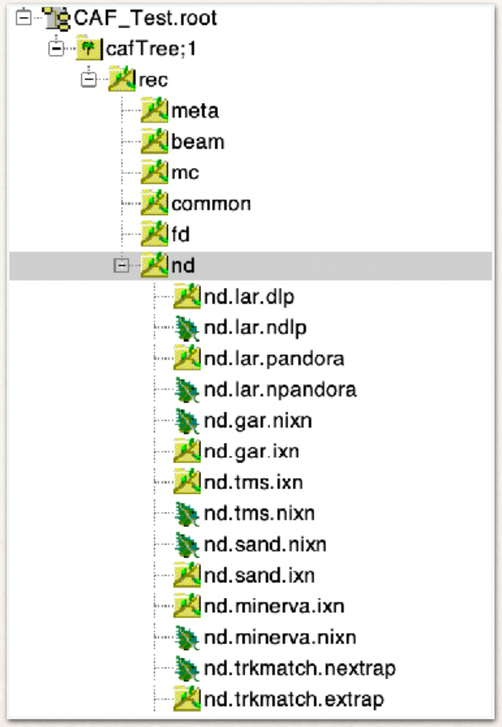
Example of creating **RNTuple** container to write CAF Data

(This Work)

Persistence of Structured CAF

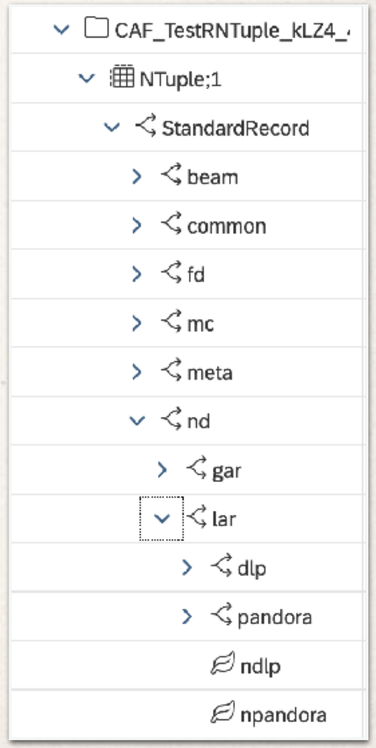
- StandardRecord as top level branch
  - Underlying attributes as sub-branches and leaves
  - I/O using CAF dictionary and library
- TTree for I/O

TTree



- StandaredRecord as top level field
  - Underlying attributes as subfields
  - I/O using CAF dictionary and library
- RNTupleWriter and Reader for I/O

RNTuple



This Work

Persistence of Flattened CAF

- Structured StandaredRecord is flattened and written into simple C++ types and persisted as ROOT::Tbranch
- Flattening done through layers of specialized template classes (each corresponding to a CAF Object)
  - Structure of the CAF Object is maintained via naming scheme
- RNTuple Implementation (Simplified example done)

```
template<> class flat::Flat
<StandardRecord> rec{
 SRBeam beam;
 SRNDBranch nd;
 SRFD fd;
 //other attributes
};
```

```
template<> class flat::Flat
<SRND> nd{
  SRNDLAr lar;
 SRNGGAr gar;
 SRSand sand;
  //other attributes
};
```

```
template <> class flat::Flat
std::vector<SRNDL
arInt> pandora;
  //other attributes
};
```

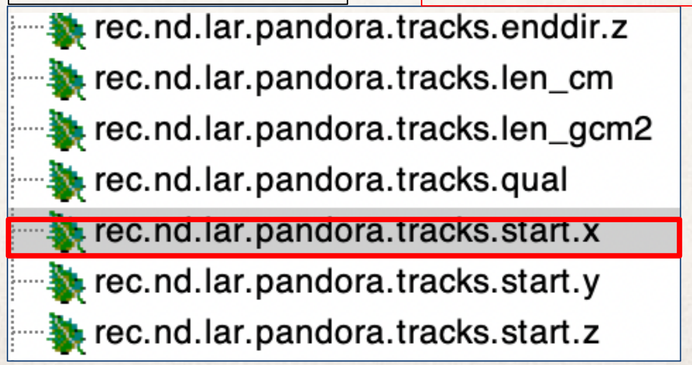
. . . .

. . .

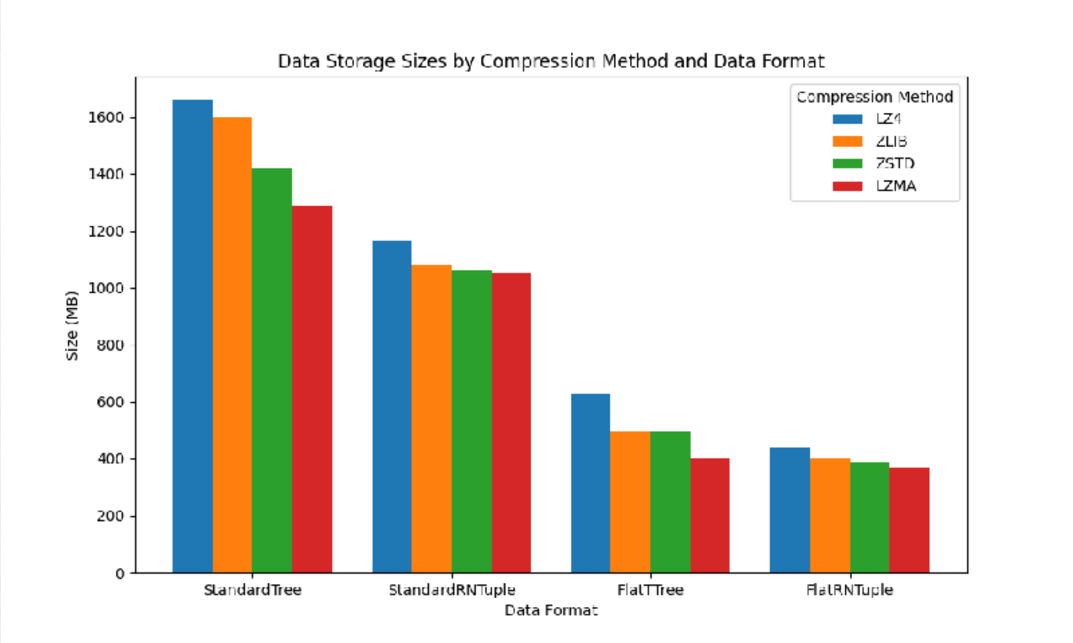
```
template <> class flat::Flat
<SRVector3D> start{
 float x;
 float y;
 float z;
};
```

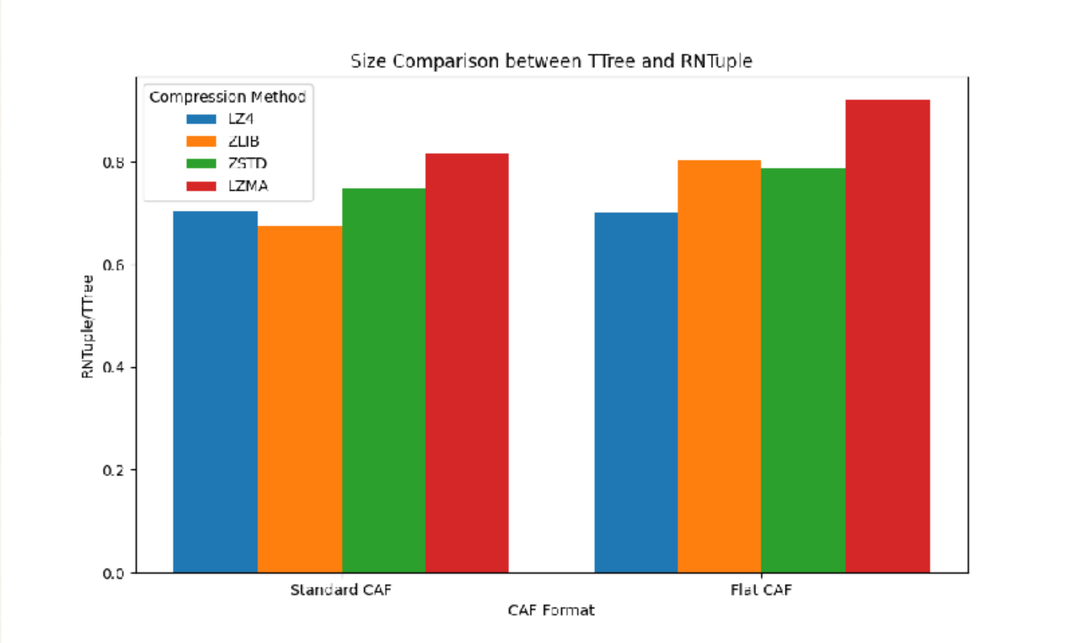
```
template < typename T>
flat:: Flat {
  //Create Branches
  //Write Data
 //Reset
};
```

- **Template class for Complex CAF objects**
  - Naming scheme
  - Pass Data to the CAFObject attributes
- **Template class for Simple C++ CAF Members**
  - Create Branches based on inherited name
  - Fill Data



File Size Comparisons with Different Compression Settings





Significant storage savings in all compression algorithms using RNTuple for both structured and Flat CAF Objects.

Summary and Outlook

- We showed structured **CAF Objects can be persisted** in RNTuple
  - Framework **provides test codes t**o write CAF Objects in both TTree and RNTuple
  - **Guideline to change/modify** existing DUNE software stack that reads and writes structured CAF
- Flat CAF I/O tool needs some code refactoring
  - Flat feature branch gives preliminary guidance on flattening CAFObjects using RNTuple API
  - Exetension of this work with more DUNE effort (**Future work**)
- Framework's standalone build
  - CAF is shared among many experiments (DUNE, Short Baseline Experiments, NOvA)
  - **Standalone build → Common code** and build tool regardless of experimental software environment