StandardRecord overhaul proposal

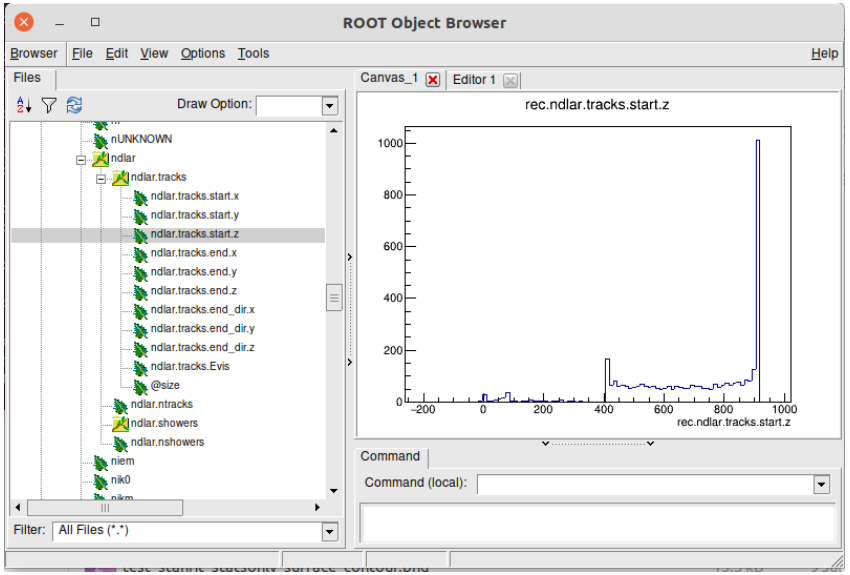
Jeremy Wolcott
Tufts University

Long-baseline working group
Apr. 3, 2023

Pre-introduction

- It's possible for this discussion to swell to fill any available volume
- These slides contain a walk-through of the *whole* set of changes
  - Not everything is relevant to LBL, I think
  - I'm currently on a “speaking tour” to shop this proposal around, so some slides are definitely here for other groups to digest carefully
- I'll try to keep the discussion from wandering too far off of LBL-relevance to keep the time under control

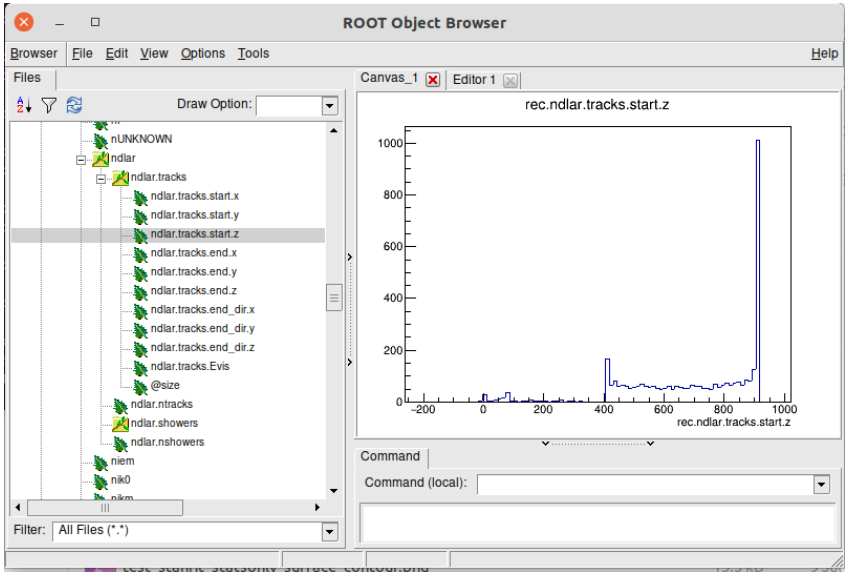
Intro: CAFs & StandardRecord



CAFs are our ROOT ntuple format intended for high-level analysis

[They are the input file type for the FD TDR LBL analysis, which used CAFAna, and will be used by LBL for ND TDR studies also, whether CAFAna, Mach3, ...]

Intro: CAFs & StandardRecord



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

They're currently a mess:

- No documentation
- Everything is at top level
- Cryptic branch names
- Duplicated quantities

Only the people who put the branches in know what they are...

caf::StandardRecord Class Reference

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

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	mvanumu
float	cvnnue
float	cvnnumu
float	cvnnutau
float	cvnnc
int	reco_q
float	Elep_reco
float	theta_reco
int	reco_lepton_pdg
float	RecoLepEnNue
float	RecoHadEnNue
float	RecoLepEnNumu
float	RecoHadEnNumu
double	pileup_energy
SRNDBranch	nd
int	RecoMethodNue
int	RecoMethodNumu
int	TrackMomMethodNumu
int	reco_numu
int	reco_nue
int	reco_nc
int	muon_contained
int	muon_tracker
int	muon_ecal
int	muon_exit
float	Ehad_veto
float	nue_pid

float	numu_pid
int	LongestTrackContNumu
float	Ev
float	Elep
int	isCC
int	nuPDG
int	nuPDGunosc
int	LepPDG
int	mode
int	GENIE_ScatteringMode
int	nP
int	nN
int	nipi0
int	nipip
int	nipim
int	niem
int	nikp
int	nikm
int	nik0
int	niother
int	nNucleus
int	nUNKNOWN
float	Q2
float	W
float	Y
float	X
float	vtx_x
float	vtx_y
float	vtx_z
float	det_x
float	eP
float	eN
float	ePip
float	ePim
float	ePi0
float	eOther
float	eRecoP
float	eRecoN
float	eRecoPip

float	eRecoPim
float	eRecoPi0
float	eRecoOther
float	eDepP
float	eDepN
float	eDepPip
float	eDepPim
float	eDepPi0
float	eDepOther
float	NuMomX
float	NuMomY
float	NuMomZ
float	LepMomX
float	LepMomY
float	LepMomZ
float	LepE
float	LepNuAngle
ctor3D	LepEndpoint
int	run
int	subrun
int	event
int	isFD
int	isFHC
float	CVNResultIsAntineutrino
float	CVNResultNue
float	CVNResultNumu
float	CVNResultNutau
float	CVNResultNC
float	CVNResult0Protons
float	CVNResult1Protons
float	CVNResult2Protons
float	CVNResultNProtons
float	CVNResult0Pions
float	CVNResult1Pions
float	CVNResult2Pions
float	CVNResultNPions
float	CVNResult0Pizeros
float	CVNResult1Pizeros
float	CVNResult2Pizeros

⋮

A better design

caf::StandardRecord Class Reference

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

Public Attributes

SRHeader	**hdr** Header branch: run, subrun, etc. More...
SRSpill	**spill** Beam spill branch: pot, beam current, etc. More...
SRSlice	**slc** Slice branch: nhit, extents, time, etc. More...
SRTrackBranch	**trk** Track branch: nhit, len, etc. More...
SRVertexBranch	**vtx** Vertex branch: location, time, etc. More...
SRMichelE	**me** Michel electron branch. More...
SREnergyBranch	**energy** Energy estimator branch. More...
SRIDBranch	**sel** Selector (PID) branch. More...
SRTruthBranch	**mc** Truth branch for MC: energy, flavor, etc. More...
SRParentBranch	**parent** True parent branch for matching, e.g. MRCC. More...
SRTrainingBranch	**training** Extra training information for prototyping PIDs etc. More...
SRTestBeam	**tb** Test Beam branch. More...

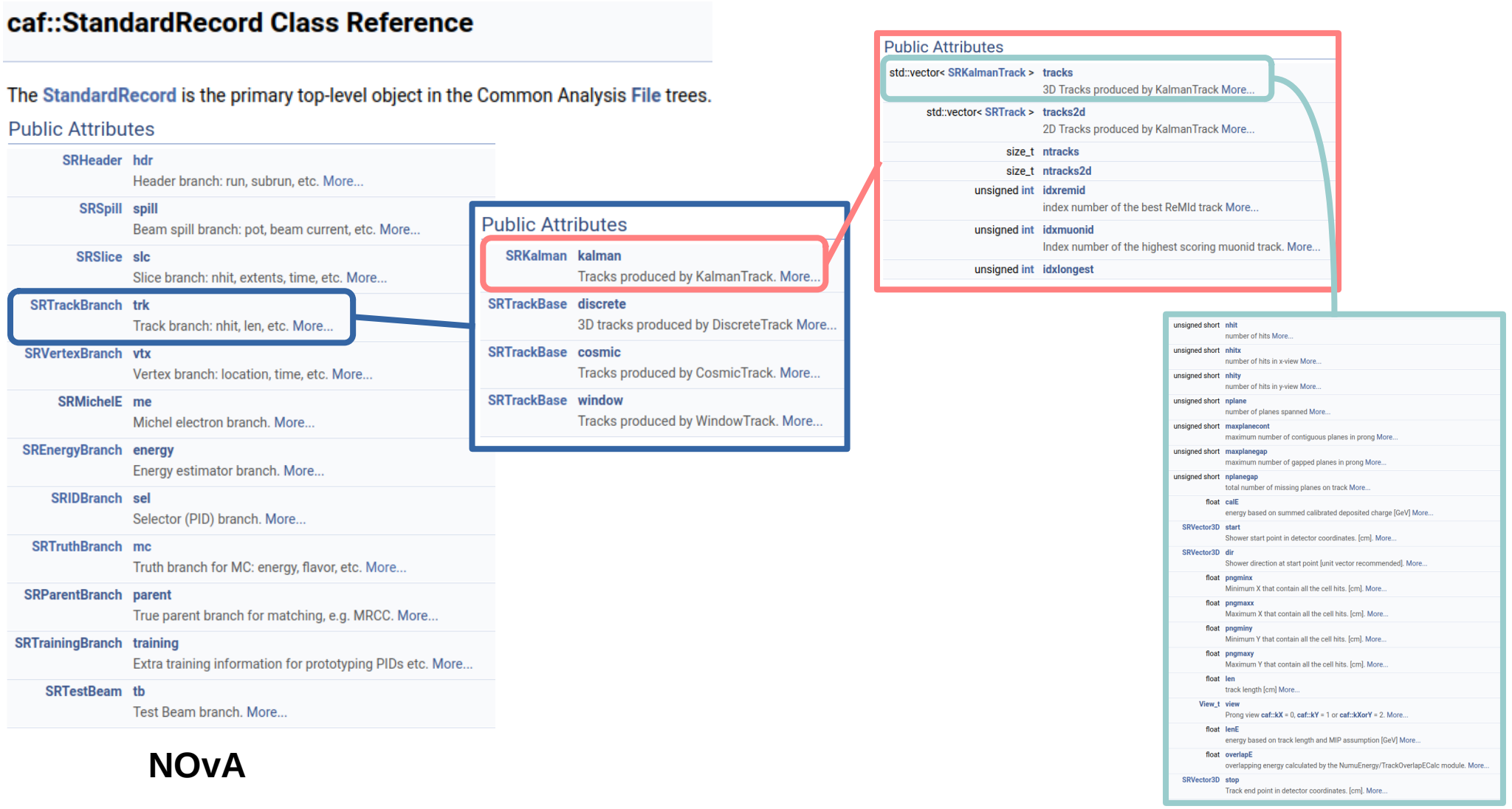
NOvA

```
class StandardRecord
{

public:
  StandardRecord();
  ~StandardRecord();

  SRHeader          hdr;    ///< Header branch: run, subrun, etc.
  //    SRSpill          spill;  ///< Beam spill branch: pot, beam current, etc.
  SRSliceRecoBranch reco;   ///< Slice reco branch: tracks, showers, etc.
  SRTruthBranch     mc;     ///< Truth branch for all interactions

  int                       nslc;    ///< Number of slices in list
  std::vector<SRSlice>       slc;    ///< Slice branch.
  int                       nfake_reco; ///< Number of Fake-Reco's in list
  std::vector<SRFakeReco>    fake_reco; ///< List of fake-reco slices
  int                       ntrue_particles; ///< Number of true particles in list
  std::vector<SRTrueParticle> true_particles; ///< True particles in spill
  int                       ncrt_hits; ///< Number of CRT hits in event
  std::vector<SRCRTHit>      crt_hits; ///< CRT hits in event
  int                       ncrt_tracks; ///< Number of CRT tracks in event
  std::vector<SRCRTTrack>    crt_tracks; ///< CRT tracks in event
  int                       nopflashes; ///< Number of OpFlashes in spill
  std::vector<SROpFlash>     opflashes; ///< List of OpFlashes in spill

  bool pass_flashtrig;     ///< Whether this Record passed the Flash Trigger requirement

};
```

SBN

CAF makers in other experiments have found that **a “hierarchical” structure is both more easily maintained and easier for beginners to understand**
[also notice the documentation of what each branch is!]

A better design



NOvA

CAF makers in other experiments have found that **a "hierarchical" structure is both more easily maintained and easier for beginners to understand**

A better design

caf::StandardRecord Class Reference

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

Public Attributes

SRHeader	**hdr** Header branch: run, subrun, etc. More...
SRSpill	**spill** Beam spill branch: pot, beam current, etc. More...
SRSlice	**slc** Slice branch: nhit, extents, time, etc. More...
SRTrackBranch	**trk** Track branch: nhit, len, etc. More...
SRVertexBranch	**vtx** Vertex branch: location, time, etc. More...
SRMichelE	**me** Michel electron branch. More...
SREnergyBranch	**energy** Energy estimator branch. More...
SRIDBranch	**sel** Selector (PID) branch. More...
SRTruthBranch	**mc** Truth branch for MC: energy, flavor, etc. More...
SRParentBranch	**parent** True parent branch for matching, e.g. MRCC. More...
SRTrainingBranch	**training** Extra training information for prototyping PIDs etc. More...
SRTestBeam	**tb** Test Beam branch. More...

NOvA

Public Attributes

std::vector< SRNeutrino >	**nu** implemented as a vector to maintain mc.nu structure, i.e. not a pointer, but with 0 or 1 entries. More.
std::vector< SRCosmic >	**cosmic** implemented as a vector to maintain mc.cosmic structure More...
std::vector< SRNeutrino >	**allnus** vector holding all Neutrinos More...
std::vector< SRCosmic >	**allcosmics** vector holding all Cosmics More...

short	**pdg** pdg code More...
float	**E** True energy [GeV]. More...
float	**visE** Sum of FLS hits that made CellHits from this neutrino [GeV]. More...
SRLorentzVector	**p** True momentum [GeV]. More...
SRVector3D	**vtx** Vertex position in detector coordinates [cm]. More...
std::vector< SRTrueMichelE >	**michel** Vector of true Michel electrons. More...
short	**pdgorig** Unoscillated (unswapped) pdg code. More...
int	**hitnuc**
SRLorentzVector	**hitnucp** Initial state 4-momentum of the struck nucleon. More...
float	**woscdumb** Simplest possible oscillation weight. More...
int	**mode** interaction mode from enum mode_type::[QE, RES, COH, ...] More...
bool	**iscc** true if charged-current interaction, false if not. More...

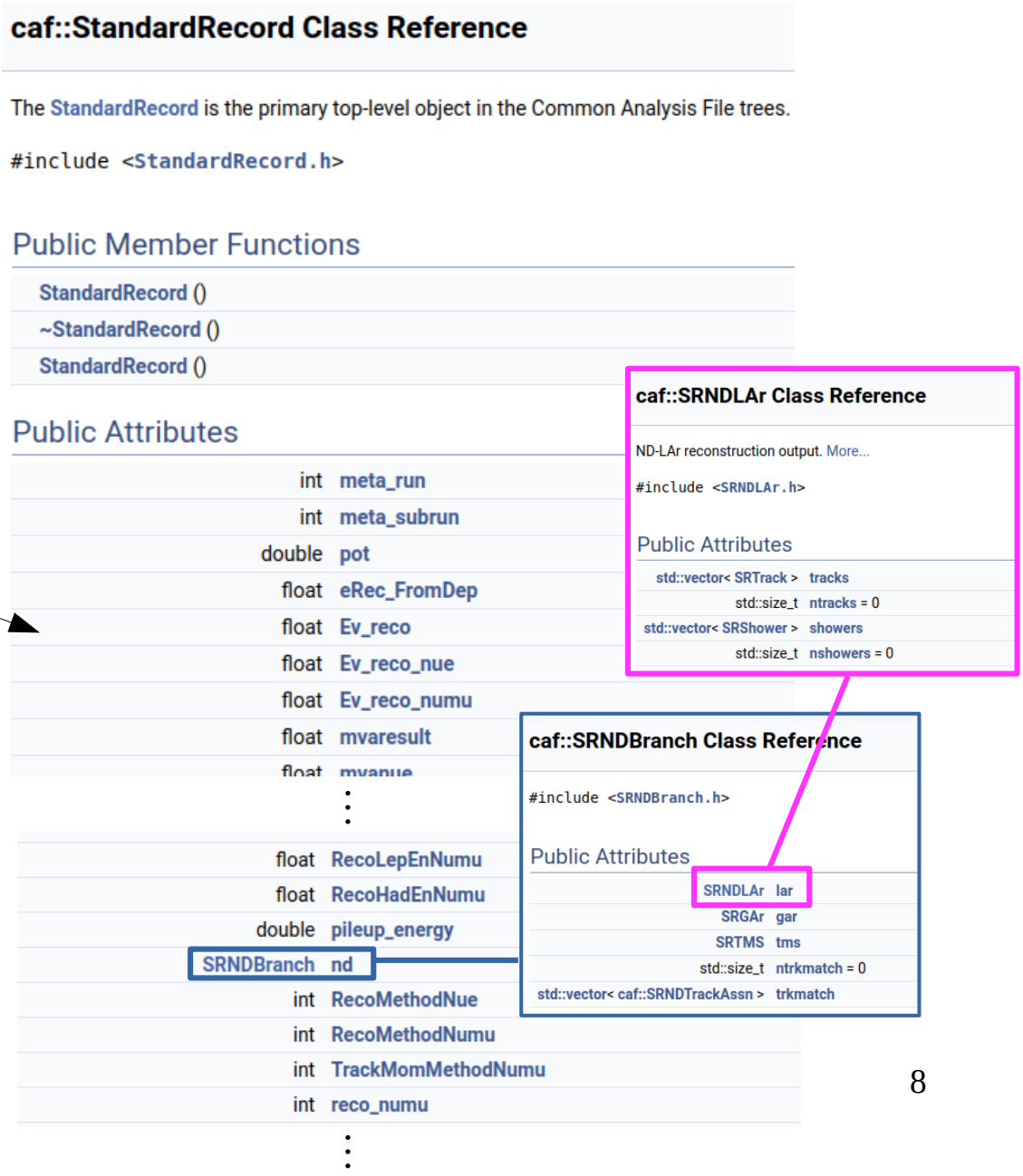
⋮

Other CAF users have found that **a "hierarchical" structure is both more easily maintained and easier for beginners to understand**

A better design

We took a baby step in this direction with the introduction of ND reco branches last year...

... but it's time to fix the rest of the StandardRecord, or we probably will never get to it.



Considerations

- Our problem is not isomorphic to NOvA or SBN
  - We have detectors of *fundamentally different types*
  - We have *many* detectors (7 in Phase II!)
  - We have *independent* detectors that sometimes *need to be matched* on an event-by-event basis (exhibit A: ND+LAr + spectrometer)
- We have multiple use cases in mind for CAFs
  - LBL – (likely) highest-level info only
  - Detector-specific studies (e.g.: xsec measurements)
  - Prototyping data analysis

New design: base elements

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

Design choice:
Broad categories
at top level

(we'll drill down into
these in a moment)

New design: base elements

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

Design choice:
Separate “common” from “detector-specific” reco

Highest-level reco info: particles, interactions, etc. Stuff that you can infer from *any* detector.
[LBL analysis should be able to work from this?]

Detector-specific reco info: cluster, tracks, showers, whatever
[everything you need for a detailed analysis]

Drilling down: metadata

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam conf
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to b
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the
      SRFDBranch fd;

      /// Reconstructed info unique to the
      SRNDBranch nd;

  };
```

```
class SRDetectorMetaBranch
{
  public:
    // Full NDs (Phase I or Phase II)
    SRDetectorMeta nd_lar;    ///< 35-module liquid argon TPC  (forms part of movable PRISM detector concept)
    SRDetectorMeta nd_gar;    ///< high-pressure gaseous argon TPC (forms part of movable PRISM detector concept in Phase II)
    SRDetectorMeta tms;       ///< magnetized spectrometer/calorimeter (forms part of movable PRISM detector concept in Phase I)
    SRDetectorMeta sand;      ///< scintillator tracker and calorimeter, fixed on-axis in beam

    // ND prototypes (more to add?)
    SRDetectorMeta lar2x2;    ///< ND-LAr prototype in NuMI beam  ('lar' prefix b/c you can't start a name with a digit in C++)
    SRDetectorMeta minerva;   ///< tracker & muon veto for 2x2; repurposed former MINERvA detector components

    // full FDs (Phase II modules TBD ... )
    SRDetectorMeta fd_hd;     /// Horizontal drift (a.k.a. module 1)
    SRDetectorMeta fd_vd;     /// Vertical drift (a.k.a. module 2)

    // FD prototypes  (add VD ProtoDUNE if/when we CAF it?)
    SRDetectorMeta pd_hd;     /// Horizontal drift prototype
```

One metadata element for each detector...

Drilling down: metadata

```
//todo: do we need branches for “post-merge” metadata too?  (shared post-merge run/subrun numbering?...)
```

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam conf
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be
  };
```

```
class SRDetectorMetaBranch
{
  public:
    // Full NDs (Phase I or Phase II)
    SRDetectorMeta nd_lar;    ///< 35-module liquid argon TPC  (forms part of movable PRISM detector concept)
    SRDetectorMeta nd_gar;    ///< high-pressure gaseous argon TPC (forms part of movable PRISM detector concept in Phase II)
    SRDetectorMeta tms;       ///< magnetized spectrometer/calorimeter (forms part of movable PRISM detector concept in Phase I)
                              d on-axis in beam

                              ix b/c you can't start a name with a digit in C++)
                              rmer MINERvA detector components
```

```
class SRDetectorMeta
{
  public:
    bool enabled = false;     ///< Does this detector have data present in this event?

    unsigned int run    = 0;
    unsigned int subrun = 0;
    unsigned int event  = 0;
    unsigned int subevt = 0;

    /// detector-dependent trigger type for the relevant readout window
    int triggertype = -1;

    unsigned long int readoutstart_s   = 0;   ///< GPS time of trigger readout start, seconds part
    unsigned int      readoutstart_ns  = 0;   ///< GPS time of trigger readout start, nanoseconds part
    unsigned long int readoutend_s     = 0;   ///< GPS time of trigger readout end, seconds part
    unsigned int      readoutend_ns    = 0;   ///< GPS time of trigger readout end, nanoseconds part

    /// For NDs that are part of the PRISM system,
    /// where (in meters relative to the beam center)
    /// was the detector center located for this event?
    double prism_offset = std::numeric_limits<double>::signaling_NaN();

};
```

Drilling down: beam

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch

      /// Reconstru
      SRCommonRecoB

      /// Reconstru
      SRFDBranch fd

      /// Reconstru
      SRNDBranch nd

  };
```

```
class SRBeamBranch
{
  private:
    // save on typing below
    constexpr static float NaN = std::numeric_limits<float>::signaling_NaN();

  public:
    bool                ismc;      ///< data or simulated beam pulse?

    bool                isgoodpulse         = true;      ///< Was the pot for a pulse good? (only applicable to data)
    unsigned long int   pulsetimesec        = 0;         ///< pulse time in seconds [s]
    unsigned long int   pulsetimensec       = 0;         ///< pulse time in nanoseconds [ns]
    unsigned long int   gpspulsetimesec     = 0;         ///< pulse time from GPS [s]
    unsigned long int   gpspulsetimensec    = 0;         ///< pulse time from GPS [ns]
    signed long long int deltapulsetimensec = -9999999;  ///< Delta time [ns]
    float               pulsepot            = NaN;       ///< POT in pulse including factor of 1e12 so that a user does not have to apply this correction
    float               hornI               = NaN;       ///< Horn current [kA]

    bool isFHC() const  { return hornI > 0; };                              ///< Checks #hornI to see if the polarity is positive ⟶ this is FHC
    bool is0HC() const  { return std::abs(hornI) < 1; };                    ///< Checks #hornI to see if the polarity is zero
    bool isRHC() const  { return hornI < 0; }                               ///< Checks #hornI to see if the polarity is negative ⟶ this is RHC

    // someday when we have real beam we'll have beam parameters here to include.
    // this is just a reminder/placeholder for now.

};
```


Not much to see here. Can easily be expanded as we go

Drilling down: truth

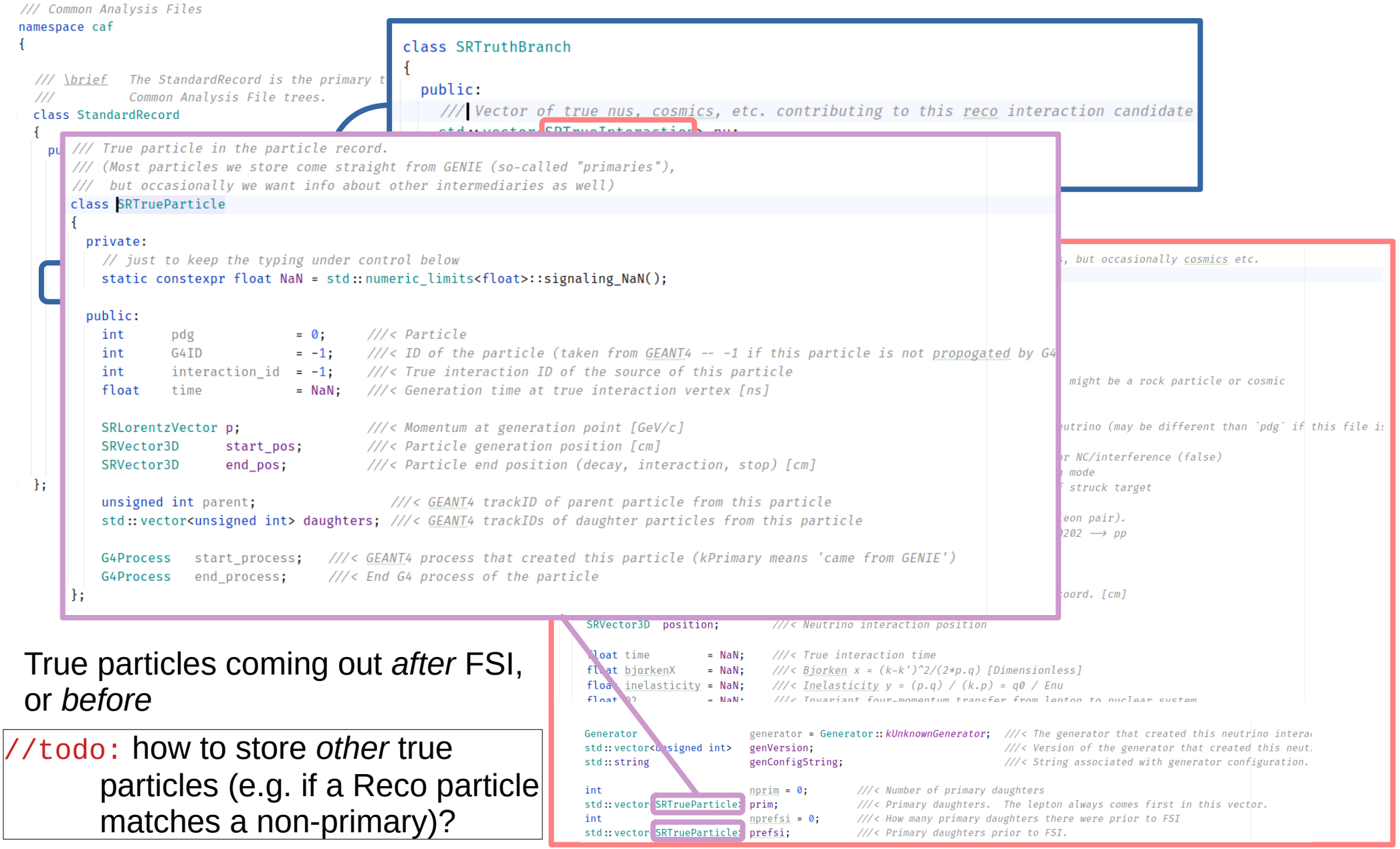
```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary t
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

```
class SRTruthBranch
{
  public:
    ///Vector of true nus, cosmics, etc. contributing to this reco interaction candidate
    std::vector<SRTrueInteraction> nu;
    std::size_t nnu = 0;
};
```

```
/// \brief True interaction of probe particle with detector.  Usually neutrinos, but occasionally cosmics etc.
class SRTrueInteraction
{
  private:
    // just to keep the typing under control below
    static constexpr float NaN = std::numeric_limits<float>::signaling_NaN();

  public:
    bool  isvtxcont = false;   ///< Is true vertex is within detector?  If not, might be a rock particle or cosmic

    int   pdg     = 0;         ///< PDG code of probe particle
    int   pdgorig = 0;         ///< Initial (unoscillated) PDG code of probe neutrino (may be different than `pdg` if this file is

    bool           iscc      = false;                          ///< CC (true) or NC/interference (false)
    ScatteringMode mode      = ScatteringMode::kUnknownMode;   ///< Interaction mode
    int            targetPDG = 0;                              ///< PDG code of struck target

    /// PDG code of struck nucleon (or, in the case of MEC, struck nucleon-nucleon pair).
    /// For MEC, the codes are: 2000000200 ⟶ nn,  2000000201 ⟶ np,  2000000202 ⟶ pp
    int   hitnuc = 0;

    float       E         = NaN; ///< True energy [GeV]
    SRVector3D  vtx;             ///< Interaction vertex position in detector coord. [cm]
    SRVector3D  momentum;        ///< Neutrino three-momentum
    SRVector3D  position;        ///< Neutrino interaction position

    float time         = NaN;    ///< True interaction time
    float bjorkenX     = NaN;    ///< Bjorken x = (k-k')^2/(2*p.q) [Dimensionless]
    float inelasticity = NaN;    ///< Inelasticity y = (p.q) / (k.p) = q0 / Enu
    float Q2           = NaN;    ///< Invariant four-momentum transfer from lepton to nuclear system

    Generator                  generator = Generator::kUnknownGenerator;  ///< The generator that created this neutrino intera
    std::vector<unsigned int>  genVersion;                                ///< Version of the generator that created this neut
    std::string                genConfigString;                           ///< String associated with generator configuration.

    int                        nprim = 0;        ///< Number of primary daughters
    std::vector<SRTrueParticle> prim;            ///< Primary daughters.  The lepton always comes first in this vector.
    int                        nprefsi = 0;      ///< How many primary daughters there were prior to FSI
    std::vector<SRTrueParticle> prefsi;          ///< Primary daughters prior to FSI.
```

All the truth stuff you'd expect.
Not the full GENIE record
(that's stored separately),
but enough to make many useful plots

Drilling down: truth



True particles coming out *after* FSI, or *before*

`//todo:` how to store *other* true particles (e.g. if a Reco particle matches a non-primary)?

Drilling down: common reco

```cpp
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

```cpp
/// Shared reconstructed info across all (?) detectors
class SRCommonRecoBranch
{
  public:
    /// Hypotheses for this neutrino interaction's identity
    SRNeutrinoHypothesisBranch nuhyp;

    /// Hypotheses for this neutrino interaction's energy
    SRNeutrinoEnergyBranch Enu;

    /// Reconstructed vertex location
    SRVertexBranch vtx;

    /// Collections of reconstructed particles
    SRRecoParticlesBranch part;
};
```

Shared reco stuff.
Each of these is a branch because there are multiple ways they can be reconstructed...

Drilling down: neutrino hypotheses

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

```
/// Shared reconstructed info across all (?) detectors
class SRCommonRecoBranch
{
  public:
    /// Hypotheses for this neutrino interaction's identity
    SRNeutrinoHypothesisBranch nuhyp;

    /// Hypotheses for this neutrino interaction's energy
    SRNeutrinoEnergyBranch Enu;

    /// Reconstructed vertex location
    SRVertexBranch vtx;

    /// Collections of reconstructed particles
    SRRecoParticlesBranch part;
};
```

```
class SRNeutrinoHypothesisBranch
{
  public:
    SRCVNScoreBranch cvn;

    // other reconstructions can go here: Pandora, DeepLearnPhysics, etc. once we have stuff to fill for them
};
```


Neutrino classification. FD has CVN; we'll add others as they are available

Drilling down: neutrino energy

Design choice:
Different reconstruction pathways get different CAF branches, rather than different collections of CAFs with the same branches (that have to be distinguished by metadata)

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

```
/// Shared reconstructed info across all (?) detectors
class SRCommonRecoBranch
{
  public:
    /// Hypotheses for this neutrino interaction's identity
    SRNeutrinoHypothesisBranch nuhyp;

    /// Hypotheses for this neutrino interaction's energy
    SRNeutrinoEnergyBranch Enu;

    /// Reconstructed vertex location
    SRVertexBranch vtx;

    /// Collections of reconstructed particles
    SRRecoParticlesBranch part;
};
```

```
class SRNeutrinoEnergyBranch
{
  private:
    static constexpr float NaN = std::numeric_limits<float>::signaling_NaN();

  public:
    float calo     = NaN;  ///< Calorimetric estimate using all hits
    float lep_calo = NaN;  ///< Lepton (longest track or largest shower) + calorimetric estimate from remaining hits
    float regcnn   = NaN;  ///< Regression CNN (assumes nue hypothesis)
};
```

Neutrino energy. One entry per style of reconstruction

Drilling down: neutrino vertex

Design choice:
Different reconstruction pathways get different CAF branches, rather than different collections of CAFs with the same branches (that have to be distinguished by metadata)

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBranch nd;

  };
```

```
/// Shared reconstructed info across all (?) detectors
class SRCommonRecoBranch
{
  public:
    /// Hypotheses for this neutrino interaction's identity
    SRNeutrinoHypothesisBranch nuhyp;

    /// Hypotheses for this neutrino interaction's energy
    SRNeutrinoEnergyBranch Enu;

    /// Reconstructed vertex location
    SRVertexBranch vtx;

    /// Collections of reconstructed particles
    SRRecoParticlesBranch part;
};
```

```
class SRVertexBranch
{
  public:
    // these are just guesses, we'll need to fill them in with the actual reco tools we're using ASAP
    SRVector3D dlp;      ///< Vertex location estimated by DeepLearnPhysics machine learning reco stack
    SRVector3D pandora;  ///< Vertex location estimated by Pandora reco stack
};
```

Vertex location. One per reconstruction pathway

Drilling down: reco particles

```
//todo: need to add linkage to true particle here
//todo: is it possible to preserve linkage to
        "detector-specific" reco info?
```

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;
```

```
/// Shared reconstructed info across all (?) detectors
                                                        's identity
                                                        's energy
```

```
/// \brief Reconstructed particle candidate
class SRRecoParticle
{
  private:
    // make the uses of it below more readable
    static constexpr float NaN = std::numeric_limits<float>::signaling_NaN();

  public:
    static constexpr int kPdgHadronicBlob = 2000000002;   ///< Special PDG code used for a "hadronic blob" (usu. calorimetrically reconstructed), borrowed from GENIE

    int         pdg      = 0;                    ///< PDG code inferred for this particle.

    float       score    = NaN;                  ///< PID score for this particle, if relevant

    float       E        = NaN;                  ///< Reconstructed energy for this particle
    PartEMethod E_method = PartEMethod::kUnknownMethod;   ///< Method used to determine energy for the particle
    SRVector3D  p;                               ///< Reconstructed momentum for this particle

    SRVector3D  start;                           ///< Reconstructed start point of this particle
    SRVector3D  end;                             ///< Reconstructed end point of this particle, if that makes sense

    // todo: would we prefer some kind of "extents" thing so that we can make a decision about containment later?
    //       or should this be the responsibility of the reco module?  (what about stuff that crosses detector boundaries? ... )
    bool        contained = false;
};
```

```
{
  public:
    int ndlp = 0;  // need these counters for SRProxy
    std::vector<SRRecoParticle> dlp;       ///< Particles reconstructed by DeepLearnPhysics machine learning stack

    int npandora = 0;
    std::vector<SRRecoParticle> pandora;   ///< Particles reconstructed by Pandora
};
```

Design choice:
Containment flag lives here. (Requires stuffing a particle corresponding to “ungrouped energy” in here)

Drilling down: FD

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info unique to the ND complex
      SRNDBran
  };
```

Design choice:
Each FD gets its own sub-branch.
May not be necessary.

Do we need separate branches for ProtoDUNEs, or would the same "main detector" branches work ok?

```
class SRFDBranch
{
  public:
    SRFD hd;   ///< Horizontal drift, a.k.a. Module 1
    SRFD vd;   ///< Vertical drift, a.k.a. Module 2

    // deal with modules 3 & 4 when we have a better idea what they are?

    SRFD pd_hd;   ///< Horizontal drift ProtoDUNE.  todo: do we really need a separate branch for it?
    SRFD pd_vd;   ///< Vertical drift ProtoDUNE.  todo: do we really need a separate branch for it?

};
```


One entry per FD. Won't drill into SRFD, it's just a placeholder for the moment.

Drilling down: ND

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info u
      SRNDBranch nd;

  };
```

```
class SRNDBranch
{
  public:
    SRNDLAr   lar;
    SRGAr     gar;
    SRTMS     tms;
    SRSAND    sand;

    /// MINERvA detector pieces used in conjunction
    /// with 2×2 prototype in NuMI beam
    SRMINERvA minerva;

    std::size_t ntrkmatch = 0;
    std::vector<caf::SRNDTrackAssn> trkmatch;
};
```

Design choices:

- Cross-detector track matches live outside the individual detector reco
- The “full ND-LAr” branch can be repurposed for 2x2

Drilling down: ND detectors

```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info u
      SRNDBranch nd;

  };
```

```
class SRNDBranch
{
  public:
    SRNDLAr    lar;
    SRGAr      gar;
    SRTMS      tms;
    SRSAND     sand;

    /// MINERvA detector pieces used in conjunction
    /// with 2×2 prototype in NuMI beam
    SRMINERvA minerva;

    std::size_t ntrkmatch = 0;
    std::vector<caf::SRNDTrackAssn> trkmatch;
};
```

```
/// ND-LAr reconstruction output
class SRNDLAr
{
  public:
    std::vector<SRTrack> tracks;
    std::size_t          ntracks  = 0;

    std::vector<SRShower> showers;
    std::size_t           nshowers = 0;
};
```

The “detector” elements here are just placeholders, for the moment, but are ready to be extended at need

Drilling down: ND track matches

```
/// Common Analysis Files
namespace caf
{
  /// \brief
  ///
  class Sta
  {
    public:
      /// M
      SRDet

      /// I
      SRBea

      /// T
      SRTru

      /// Reconstructed info expected to be common to all (?) detectors
      SRCommonRecoBranch common;

      /// Reconstructed info unique to the FDs
      SRFDBranch fd;

      /// Reconstructed info u
      SRNDBranch nd;
  };
```

```
class SRNDTrackAssn
{
  public:
    int larid          = -1;        ///< index of ND-LAr track
    int tmsid          = -1;        ///< index of TMS track
    int minervaid      = -1;        ///< index of MINERvA track
    float transdispl   = -999.;     ///< perpendicular distance between the two tracks at longitudinal position of matching point
    float angdispl     = -999.;     ///< angular difference between the two tracks at longitudinal position of matching point
};
```

```
class SRNDBranch
{
  public:
    SRNDLAr   lar;
    SRGAr     gar;
    SRTMS     tms;
    SRSAND    sand;

    /// MINERvA detector pieces used in conjunction
    /// with 2×2 prototype in NuMI beam
    SRMINERvA minerva;

    std::size_t ntrkmatch = 0;
    std::vector<caf::SRNDTrackAssn> trkmatch;
};
```

Currently storing track indices (into the relevant track vectors within the detector objects shown on previous slide), as well as distance and angular difference.
(These came from ND-LAr+TMS matching studies.)

```
//todo: this should be a branch with multiple vectors (multiple styles of track matching)
//todo: duplicate for showers?
//todo: add a SRTrack branch here too
```

Things to ponder

- We assume each “event” is somehow overlapping in time across detectors.

  (Not StandardRecord's problem to solve, I don't *think*, but important to note.)

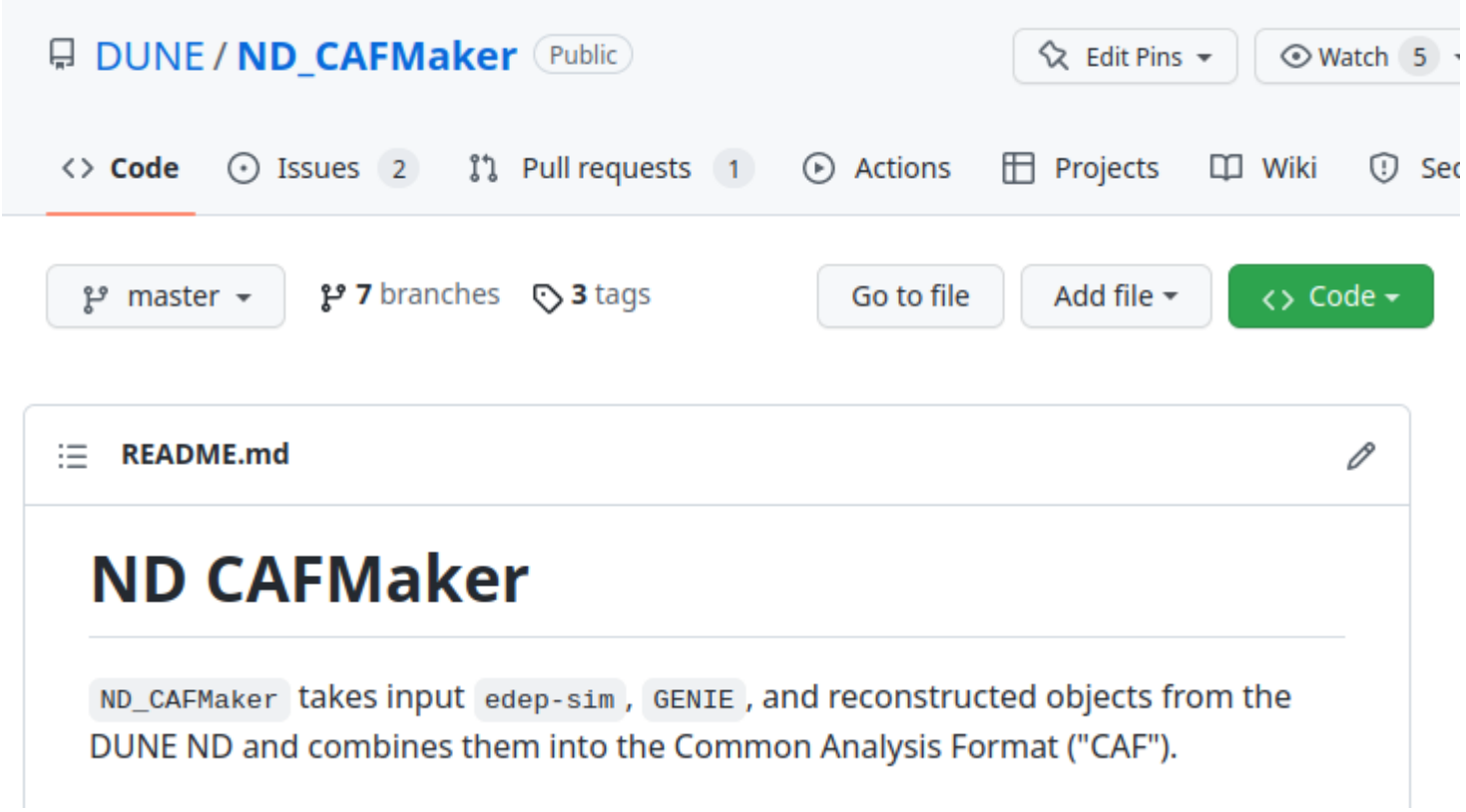
- How do we record “containment” in the common reco for particles that cross detector boundaries?
- I've dropped the PRISM-specific branches (since everything should be in the new layout)

What's next?

- I'm currently “on tour” with this proposal. Goal is to get feedback from:
  - ND-prototypes “CAF task force”
  - ND sim/reco
  - LBL
  - others? [your suggestion here]
- I have a draft pull request against `duneanaobj` containing my proposal as of right now
  - I encourage detailed feedback there, esp. line-by-line comments on the classes/files, if you have any
- There's a pending SAND pull request adding SAND-specific info, which needs to get merged before this proposal so that we don't stomp on each other
- The CAFMakers (ND & FD) will need updates to fill the updated structures
- Need to ensure CAFAna will build & run with updated StandardRecord

backup slides follow

How is ~~babby~~ CAF formed?

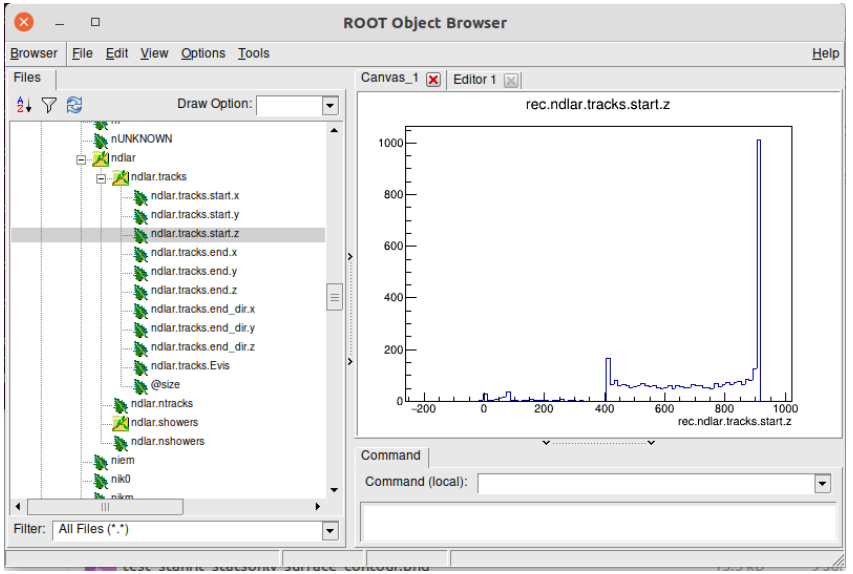


The ND CAFMaker writes out CAFs.

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