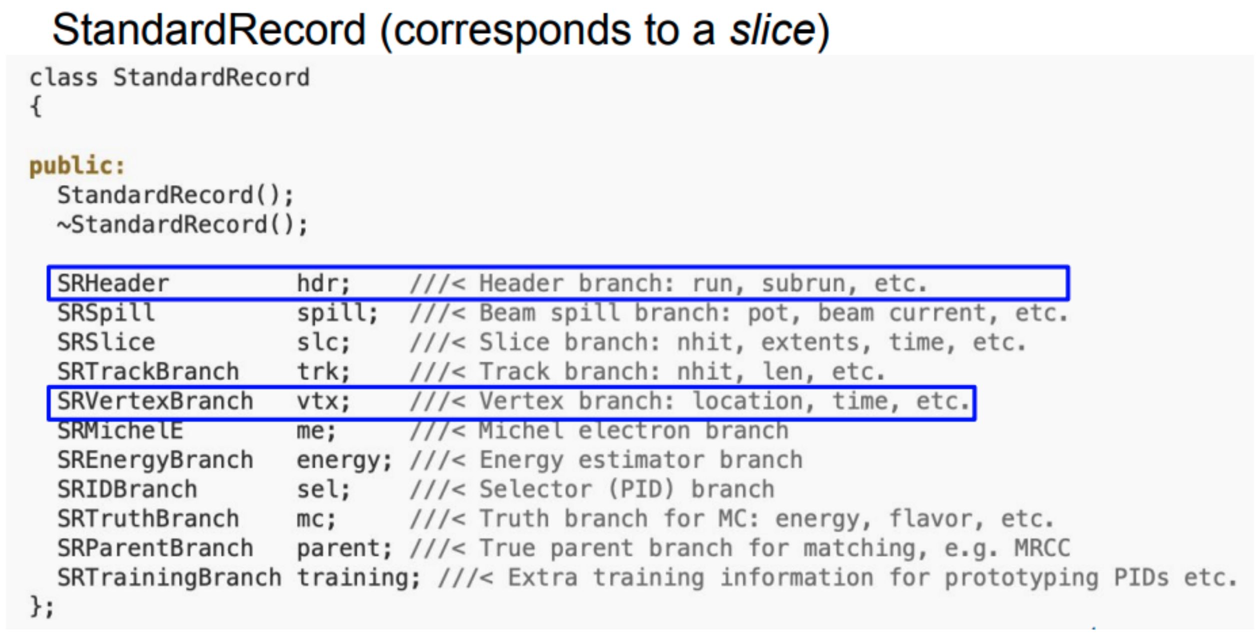
NOvA*

While DUNE,ATLAS and CMS talks were focused on the data model of their data products during reconstruction, NOvA talk is about the remodeling of the final analysis ntuples to support parallel (and GPU friendly) I/O. Here each “event” is the data collected during the neutrino beam window during which the detector DAQ is collecting the data. The data is stored as a reconstructed variable where each “event” contains many “slices” (A slice is roughly a collection of physics activities recorded by the detector that corresponds to an offline sub trigger) or “subevents”. Each “slice” stores the information that is defined the “StandardRecord” (SR) class.


StandardRecord (corresponds to a *slice*)

```
class StandardRecord
{

public:
  StandardRecord();
  ~StandardRecord();

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

Figure: The variables inside the StandardRecord class. Each “slice” is a StandardRecord (SR) object. A collection of slices creates an event. An event is the data collected during the beam window by the detector. The SR object consists of both fundamental type members as well as class objects (highlighted by blue rectangles above).

Each event consists of many “slices” or SR objects and one SR header which contains metadata information like date and time stamp, run information, detector status etc.

This format is called CAF (Common Analysis Format) which is used by NOvA to perform its physics analysis.

Transformation to make Data GPU Friendly

CAF is an object oriented data model format with many levels of hierarchies and segmentations. The data model goes through reorganization to make it scalable and support parallel I/O. The data is written into HDF5 to utilize its MPI based parallel I/O.

Data Model:

The data is organized in a columnar table format where each variable (members of SR class) is a column and each slice/subevent is a row. Each class object (like a (SR)Vertex object made from SR object members) is written in separate tables with relevant attributes. The first 4 columns of all tables store the metadata which helps to identify the object.

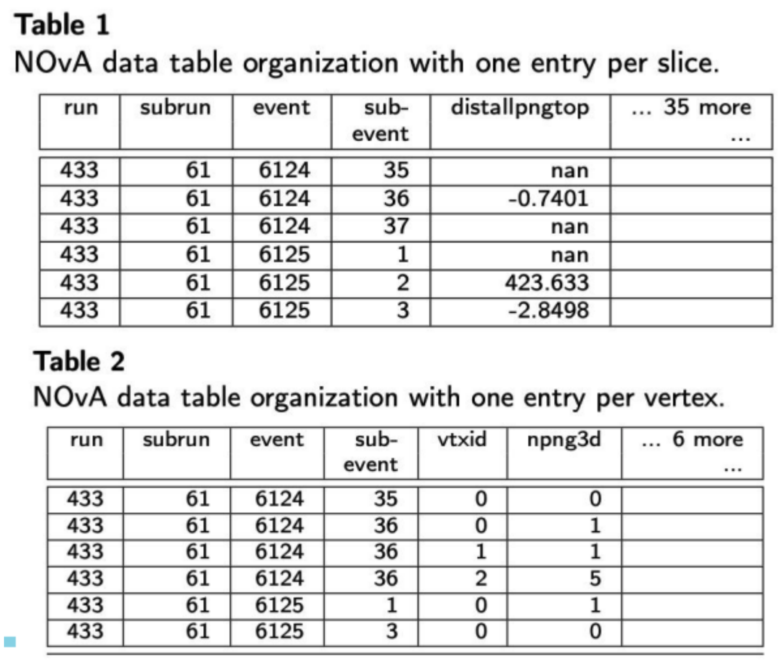


Table 1
NOvA data table organization with one entry per slice.

run	subrun	event	sub-event	distallpngtop	... 35 more ...
433	61	6124	35	nan	
433	61	6124	36	-0.7401	
433	61	6124	37	nan	
433	61	6125	1	nan	
433	61	6125	2	423.633	
433	61	6125	3	-2.8498	

Table 2
NOvA data table organization with one entry per vertex.

run	subrun	event	sub-event	vtxid	npng3d	... 6 more ...
433	61	6124	35	0	0	
433	61	6124	36	0	1	
433	61	6124	36	1	1	
433	61	6124	36	2	5	
433	61	6125	1	0	1	
433	61	6125	3	0	0	

Figure: Tabular representation of the data in NOvA. Each complex variable (SRvertex object in this example (Table 2)) is written in separate tables with the relevant attributes and metadata. Since the indexing or the metadata (the first 4 columns in the tables above)are part of each separate table, it is stored redundantly in the memory to allow faster I/O and is highly compressed in the disk.

(Need to write the tabular format when stored in disk vs. memory)

(Support of parallelism from MPI/IO based on HDF5)

Representation of the data in HDF5 format*

Memory Management

Tabular data is optimized for vectorized computation. The tables of data (and related metadata) are written in HDF5 that can utilize the inherent HDF5 libraries for faster access of data and metadata. Parameters like chunk-size and storage layout are optimized for efficient memory management and I/O.

(Also Marc mentioned about compression of their data)

Compatibility with the ROOT

The tabular CAF is written in HDF5 format but the data itself is the reorganization of hierarchical object oriented CAF data written in ROOT.

Persistency

Files are stored in HDF5.