Flux and Geometry Drivers

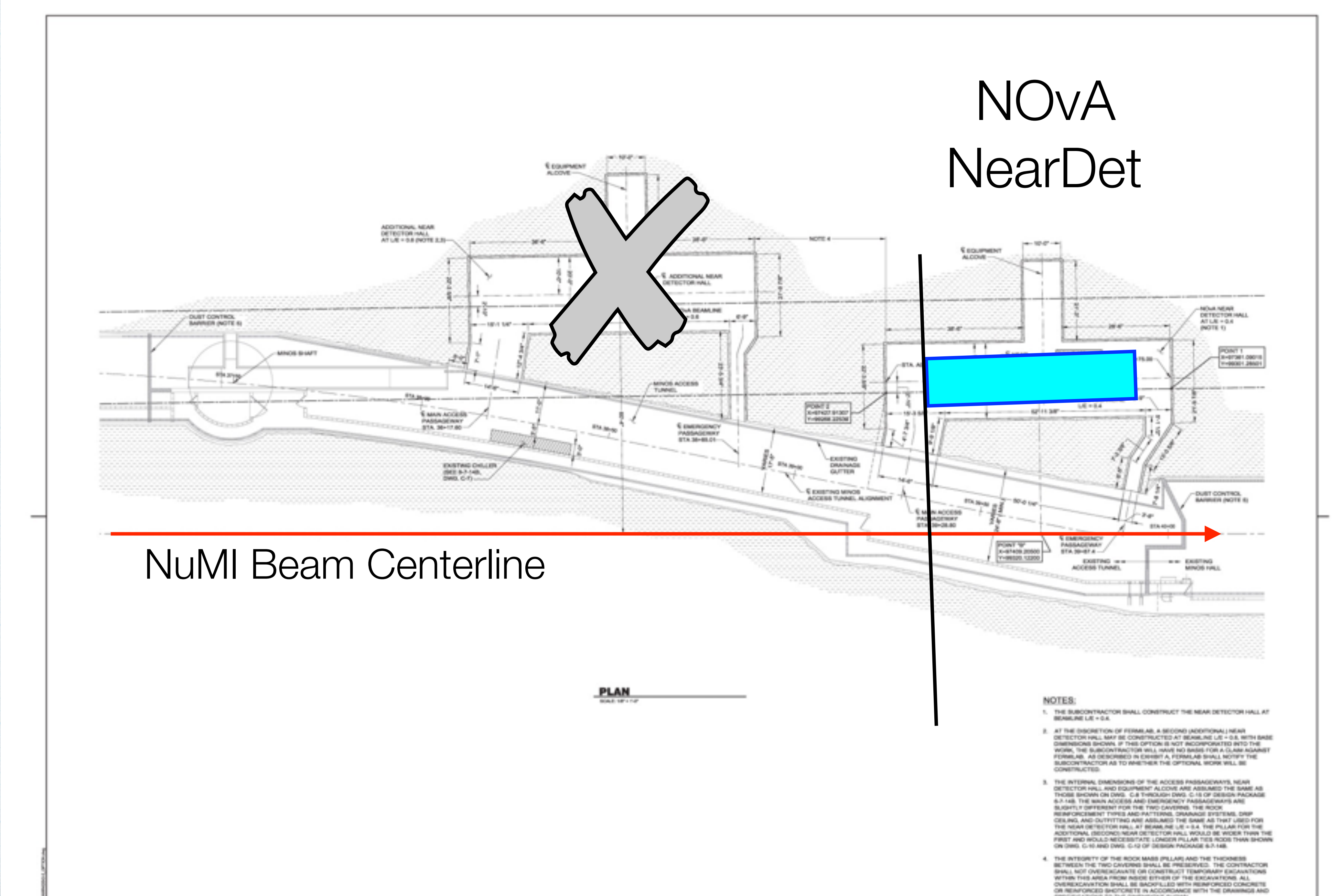
Robert Hatcher
Fermilab Computing Division

Generator Workshop 2020-01-08

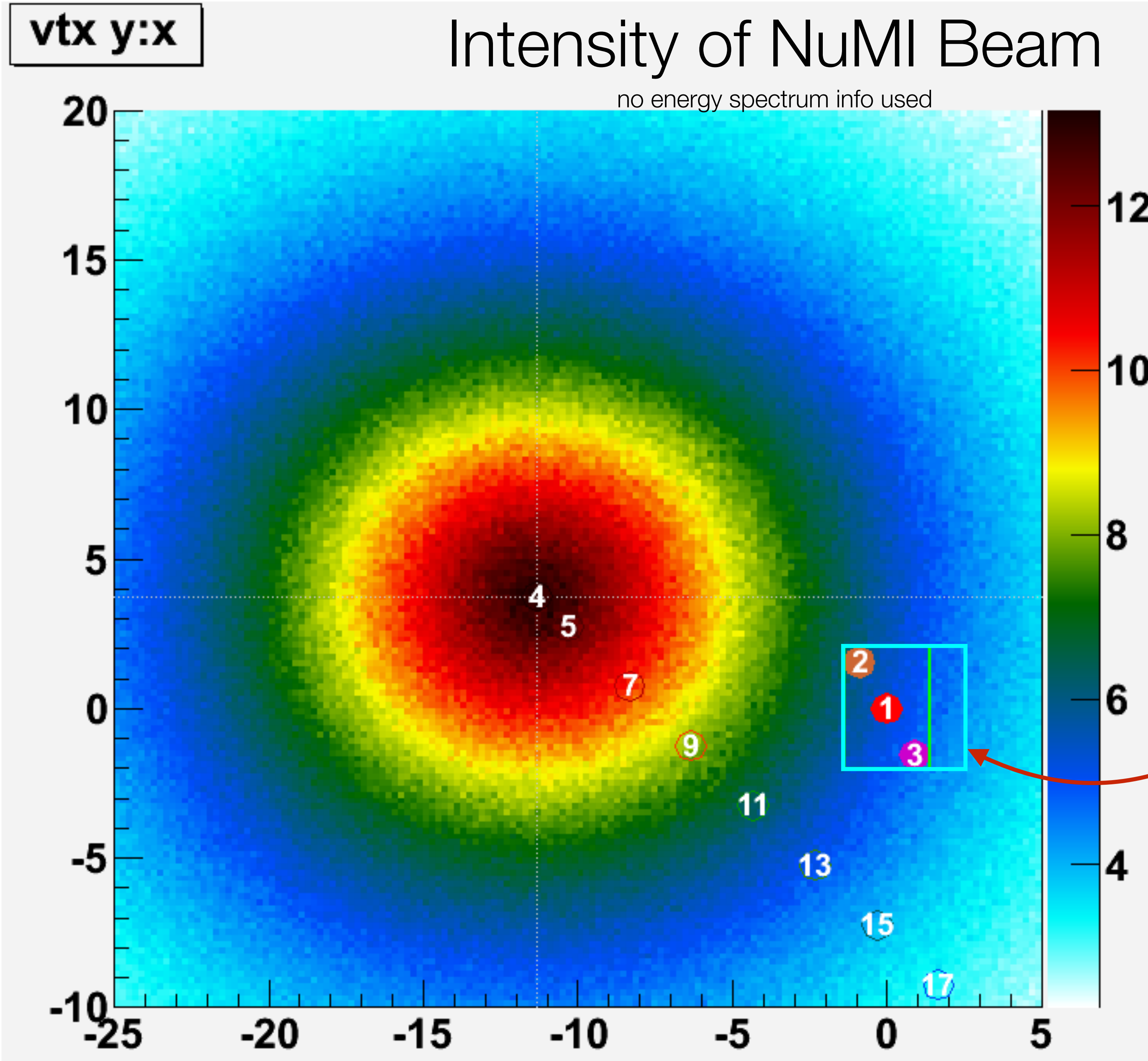
Motivation

- Theoretical and early detector design simulations often make simplifying assumptions about the neutrino flux and the detector.
  - flux spectrum without positional dependence
  - flux without divergence (i.e. as a plane wave)
  - uniform detector material
  - "infinite" detector (or extremely limited fiducial volume)
- But for a complete publishable analysis of actual data in as-built detectors, generally one must move beyond that to fully understand effects from detector acceptance and mischaracterizations of the beam

Motivation

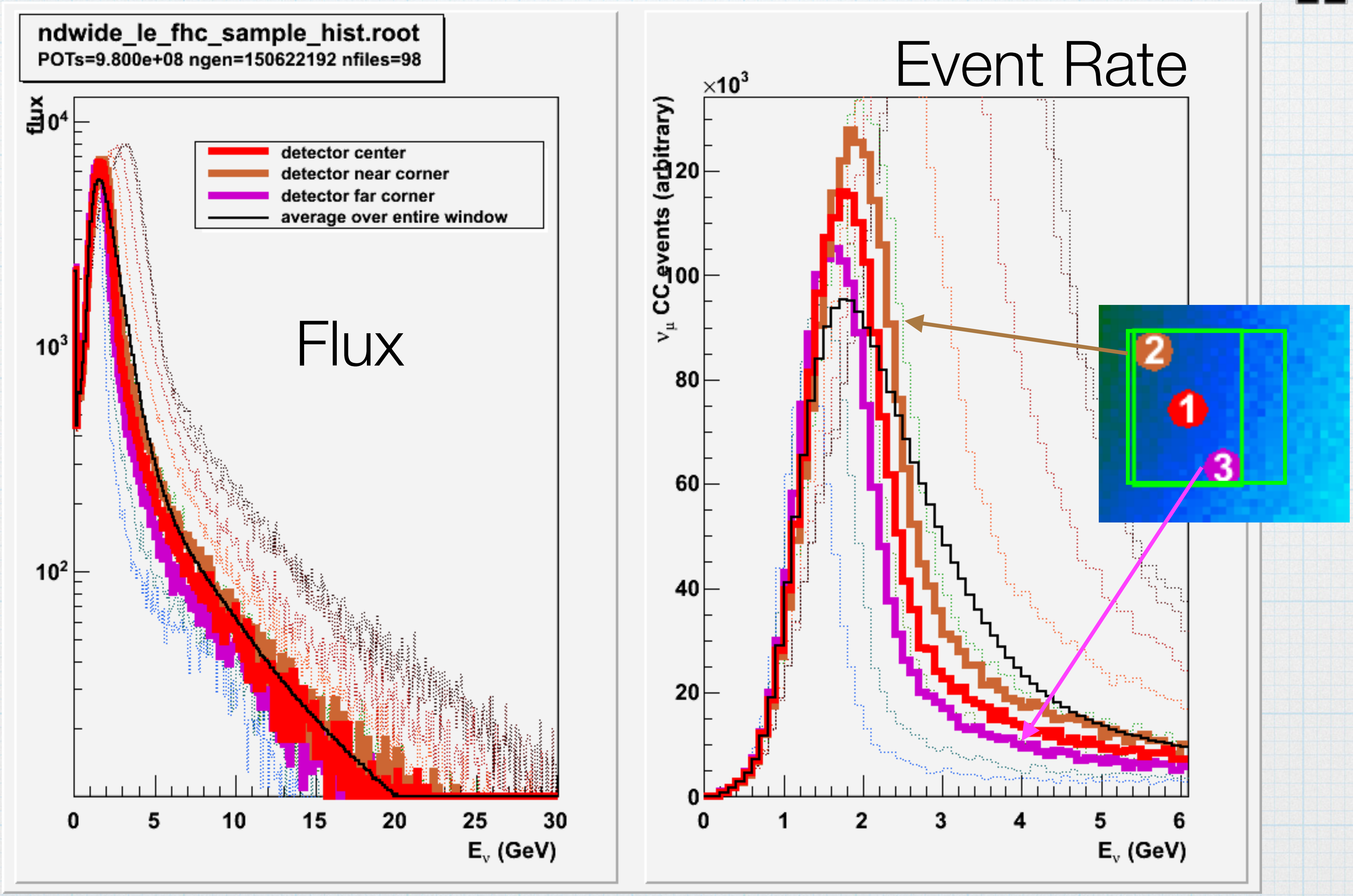


Motivation



NOvA NearDet

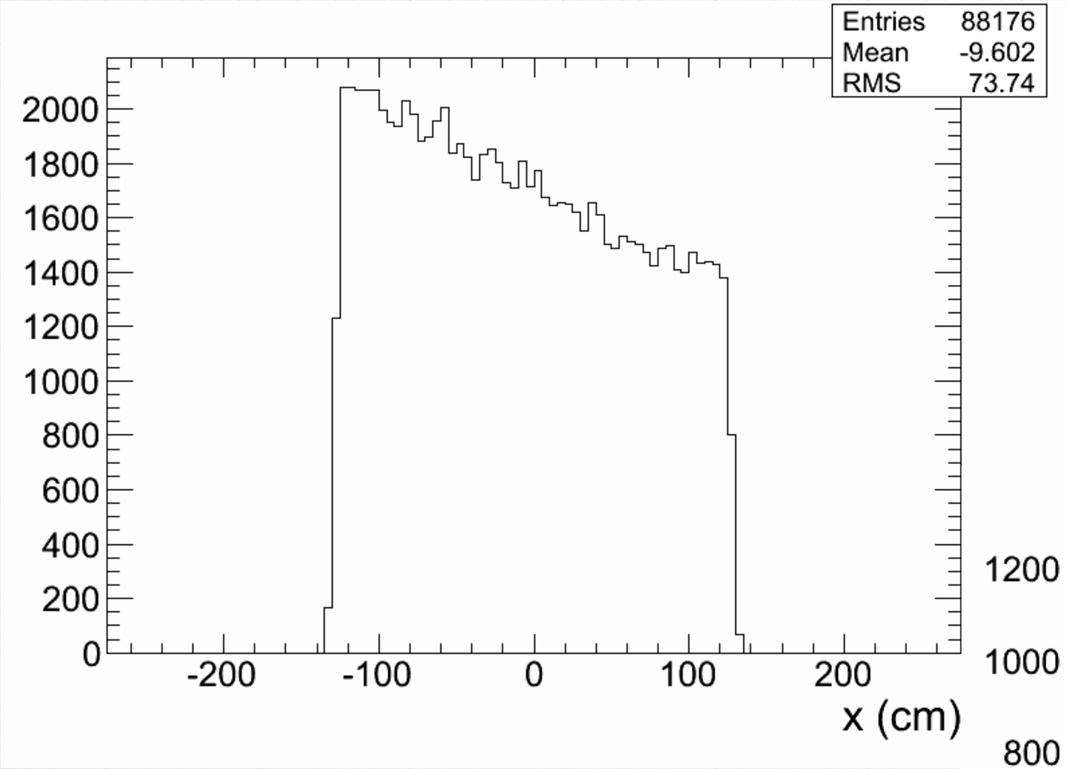
Motivation

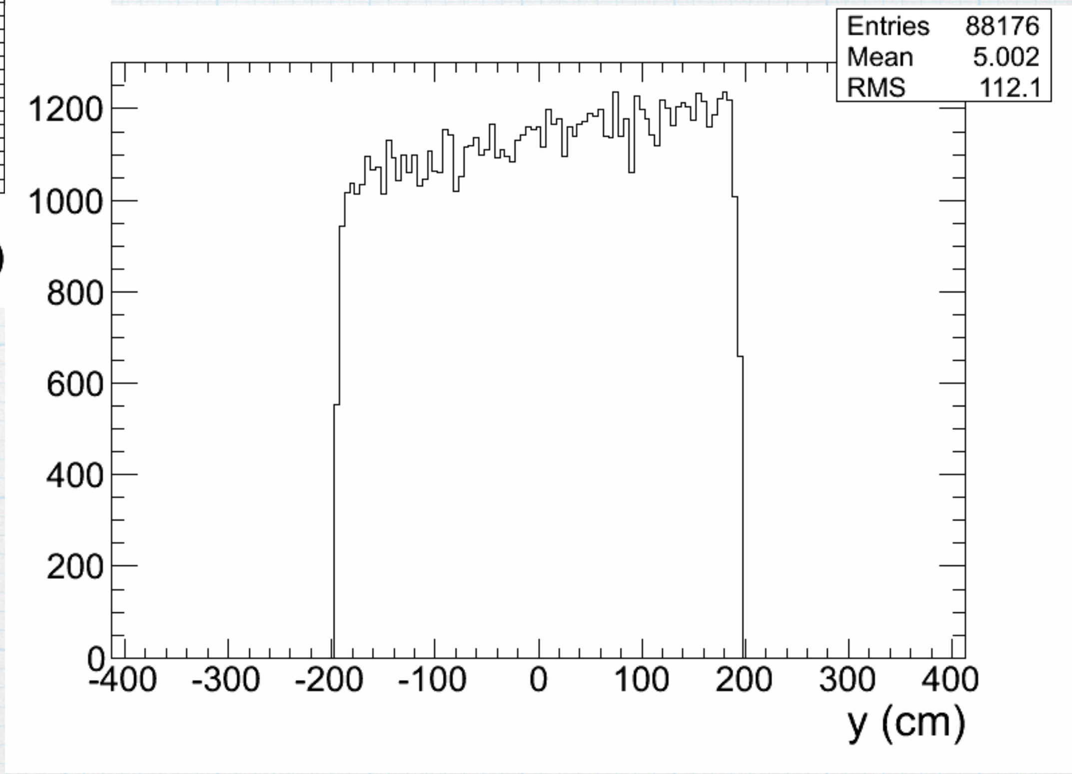


ndwide_le_fhc_sample_hist.root
POTs=9.800e+08 ngen=150622192 nfiles=98
flux
detector center
detector near corner
detector far corner
average over entire window
Flux
E_ν (GeV)
Event Rate
$\times 10^3$
ν_μ CC events (arbitrary)
E_ν (GeV)
1
2
3

Motivation

Vertex Positions

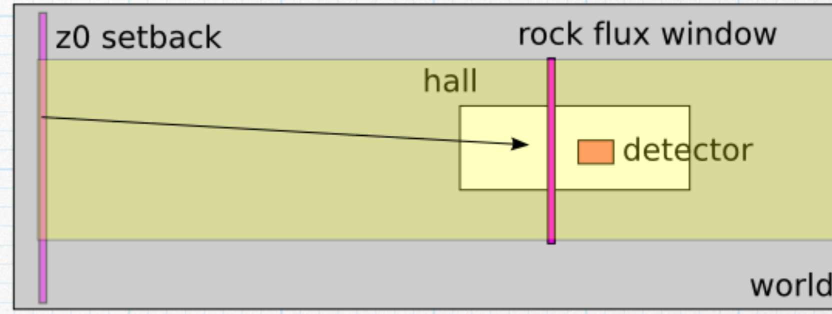




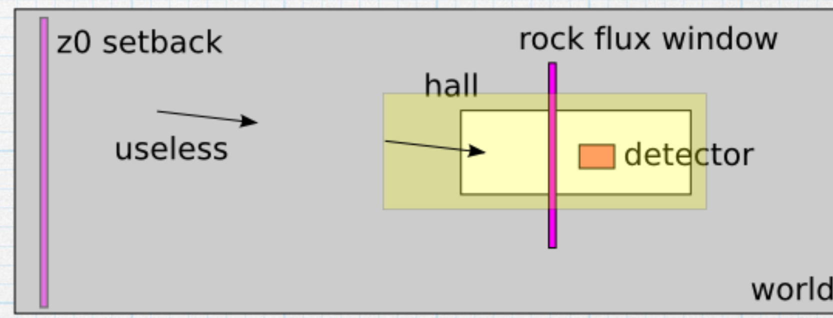
Motivation

- "Rock Box" optimization
- ICARUS - 2 volumes
- sub-volume selection
- arbitrary fiducial cuts

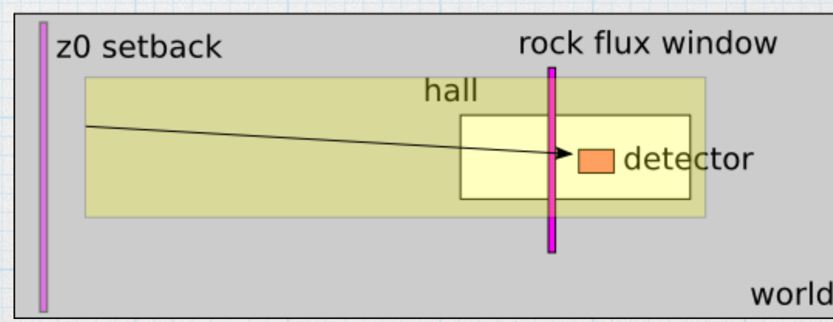
No GeomSelector



RockBox: 2 GeV



RockBox: 80 GeV



Some Definitions

- driver - an complete class that implements a API (application programming interface) specifically to tie the generator to some external source of information, to wit: flux & geometry drivers
- neutrino ray
  - neutrino flavor as PDG code
  - 4-momentum (in some defined space)
  - 4-position from which to start (in some defined space)
  - (possibly a weight)
- flux window - a surface upon which the ν ray starts (generally pointing towards the detector volume of interest).

Generator Requirements for Flux

- Upon query (i.e. call `generateRay()`) return a ν ray
  - {pdg, p4, x4, wgt}
  - possibly signal "out of entries"; otherwise cycle around
- During initialization the generator need to know:
  - maximum energy ever returned (as configured)
    - need for scaling normalization
  - list of potential ν flavors
    - to ensure all necessary cross-sections are available
- At the end of generation it should return:
  - some measure of "exposure"
    - POTs for accelerator beam fluxes
    - time for atmospheric / cosmic fluxes

Flux Notes

- Weights are "inconvenient" generally if carried through to event generation
  - especially for "near" detectors that might have event pileup in space & time
  - events "contaminated" by others are difficult to interpret if either are weighted
- Re-using flux rays -- usually not an issue because other issues contribute "entropy" via: non-uniform material, vertex choice along ray direction, interaction choice, event kinematics making events "unique" enough.
  - most cases just return `false` for `exhausted()`
  - still need sufficient entries to be representative

Example Fluxes

- Sampling from histograms of energy for various flavors, but no position dependence and no divergence (i.e. plane wave)
- Atmospheric expands on this by having bins of energy, $\cos\theta$ (relative to zenith), perhaps φ bins.
- Dk2Nu - flux beam line simulation recording decay of particles (hadrons, muons) that yield neutrinos. Decays can be reweighed for different placement of the detector and apparent size.
  - rays fully correlate flavor, position, energy, direction

Minimal Interface

```
class GenericFluxInterface {

  public :

    typedef enum EExposureUnits {
       kUnknown = 0,
       kPOTs    = 1,  // particles (protons) on target
       kSeconds = 2   // exposure duration
    } ExposureUnits_t;

    virtual ~GenericFluxInterface();

    // basic generator-facing interface for event generation

    int                     pdg    (void);
    const TLorentzVector& p4     (void);
    const TLorentzVector& x4     (void);
    double                  weight (void);

    // every derived class must implement these

    virtual bool  generateRay (void) = 0;
    virtual bool  exhausted   (void) = 0;

    // generator initialization information

    virtual const std::set<int>&  returnedFlavors   (void) = 0;
    virtual double                maxReturnedEnergy (void) = 0;

    void    setGenerateWeighted (bool genWgt);
    bool    getGenerateWeighted (void);
```

Minimal Interface

```
    // for accounting purposes

    virtual double                totalExposure (void) = 0;
    virtual ExposureUnits_t       exposureUnits (void) = 0;
    virtual void                  clearExposure (void) = 0;

  protected:
    // no one can construct one of these base class objects
    GenericFluxInterface();

    // common info for all fluxes
    int             _pdg;
    TLorentzVector  _p4;
    TLorentzVector  _x4;
    double          _weight;
    bool            _genAsWeighted;

};


inline GenericFluxInterface::GenericFluxInterface()
  : _pdg(0), _genWeighted(false)
{
  //
}

inline GenericFluxInterface::~GenericFluxInterface() { }

inline int                   GenericFluxInterface::pdg(void)        { return _pdg; }
inline const TLorentzVector& GenericFluxInterface::p4(void)         { return _p4; }
inline const TLorentzVector& GenericFluxInterface::x4(void)         { return _x4; }
inline int                   GenericFluxInterface::weight(void)     { return _weight; }
inline void GenericFluxInterface::setGenerateWeighted(bool genWgt) { _genAsWeighted = genWgt; }
inline bool GenericFluxInterface::getGenerateWeighted(void)        { return _genAsWeighted; }
```

Additional Flux Driver Requirements

- Generator needs a means to configure driver
  - list of input files (histograms, binned data, dk2nu files)
  - "fake" fluxes (generally no exposure accounting)
    - mono-energetic or "functional" form
  - atmospheric/cosmic flux
    - set min/max energies to generate
    - generation sphere radius, spot size
    - coordinate transform (z = zenith vs. detector z)
  - beam fluxes
    - beam to detector coordinate transformation
    - detector specific items: flux "window" definition; set backs, other limits; exception handing config

Other Flux Driver Concerns

- Does input file ordering matter or need to be randomized
- For drivers like Dk2Nu one needs access to underlying TChain entries so they can be recorded with the generated events and the flux can be reweighted for corrections of beam line simulation
- Some of these issues can be further standardized by use of "mix-in" interfaces for general subclasses of flux drivers
    - common scheme for coordinate transformations
    - common scheme for those needing sets of files
    - ...

Geometry Drivers

- Basically two types
  - material admixture with no spatial extent
  - full representation of the detector geometry
    - common to use GDML (geometry description markup language) file as input, which can be translated into a ROOT representation (with associated utilities) and Geant4's internal representation.

Generator Requirements for Geometry

- During generator initialization it needs:
  - list of potential target isotopes
  - maximum path length through for each target isotope
- During initialization the generator sets:
  - to volume specification (GDML file or admixture)
  - the "top" volume to use, if not the "world"
    - additional volume sub-selection and/or fiducial cuts
  - a means for scanning the geometry for maximum path L
  - setting units (geometry in "cm"? vs. generator units)ilable

Generator Requirements for Geometry

- Upon given a ν ray:
  - compute the path length sum for seen for each target isotope along that ray's path
  - ? make a decision whether the ray interacts and which target was struck ?
  - generate a vertex position based on chosen target isotope and position of such isotopes along the ray path
- At the end of generation
  - ? a weighting factor for applying to the flux exposure ?
- The light gray is where the interface has some ambiguity about where the separation with the generator lies.

Generic Geometry

- More difficult to specify
- Choices, choices ...