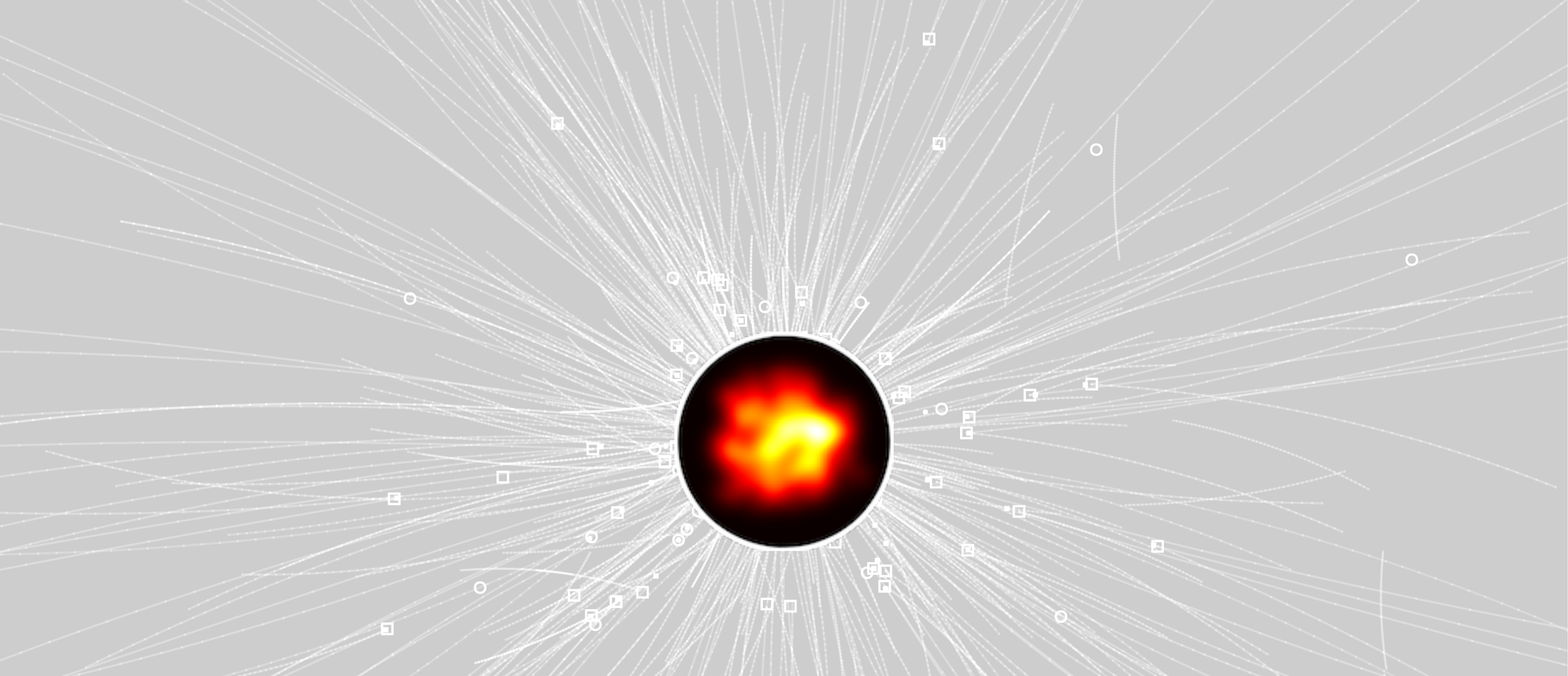
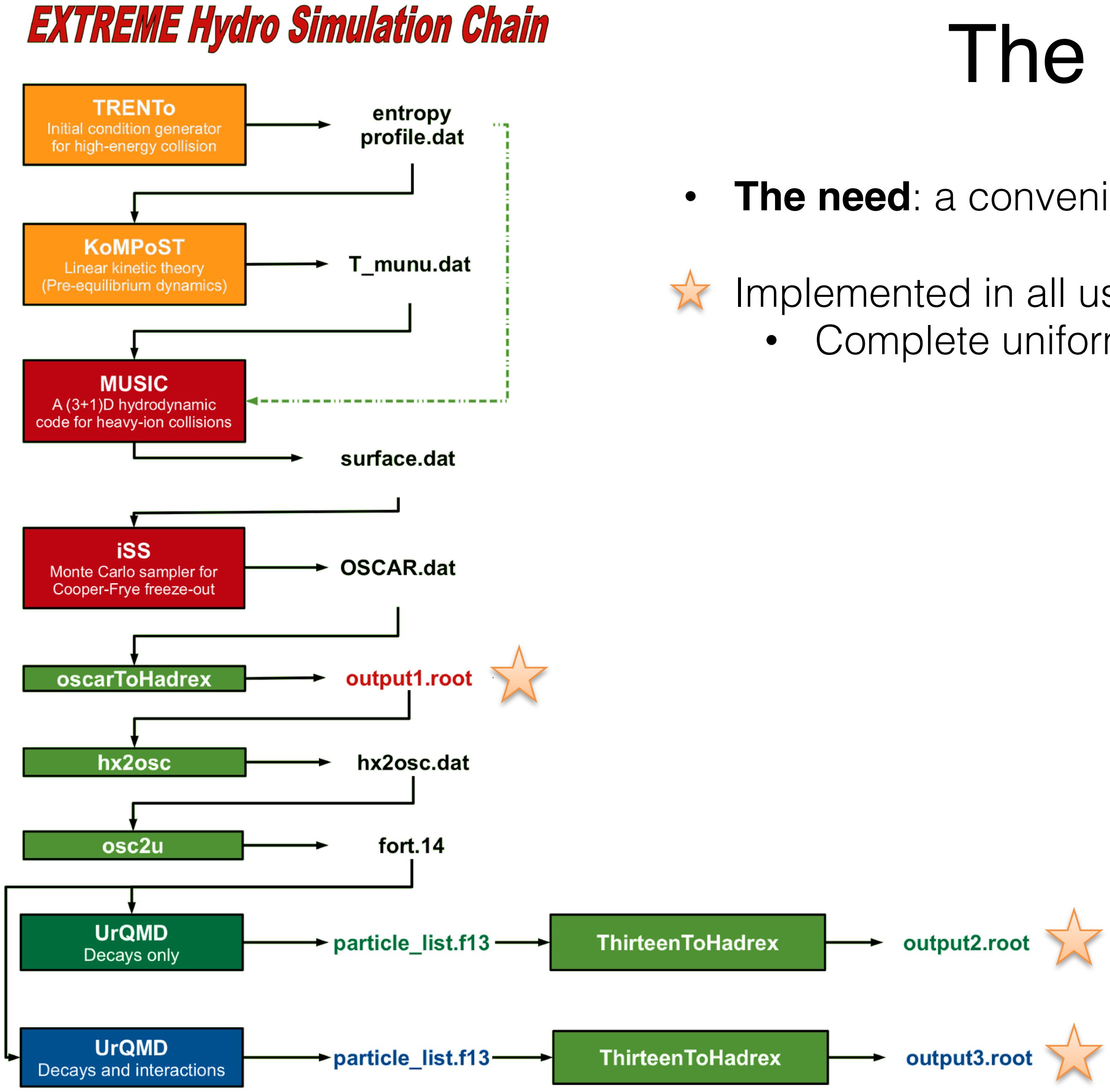


The EXTREME chain:

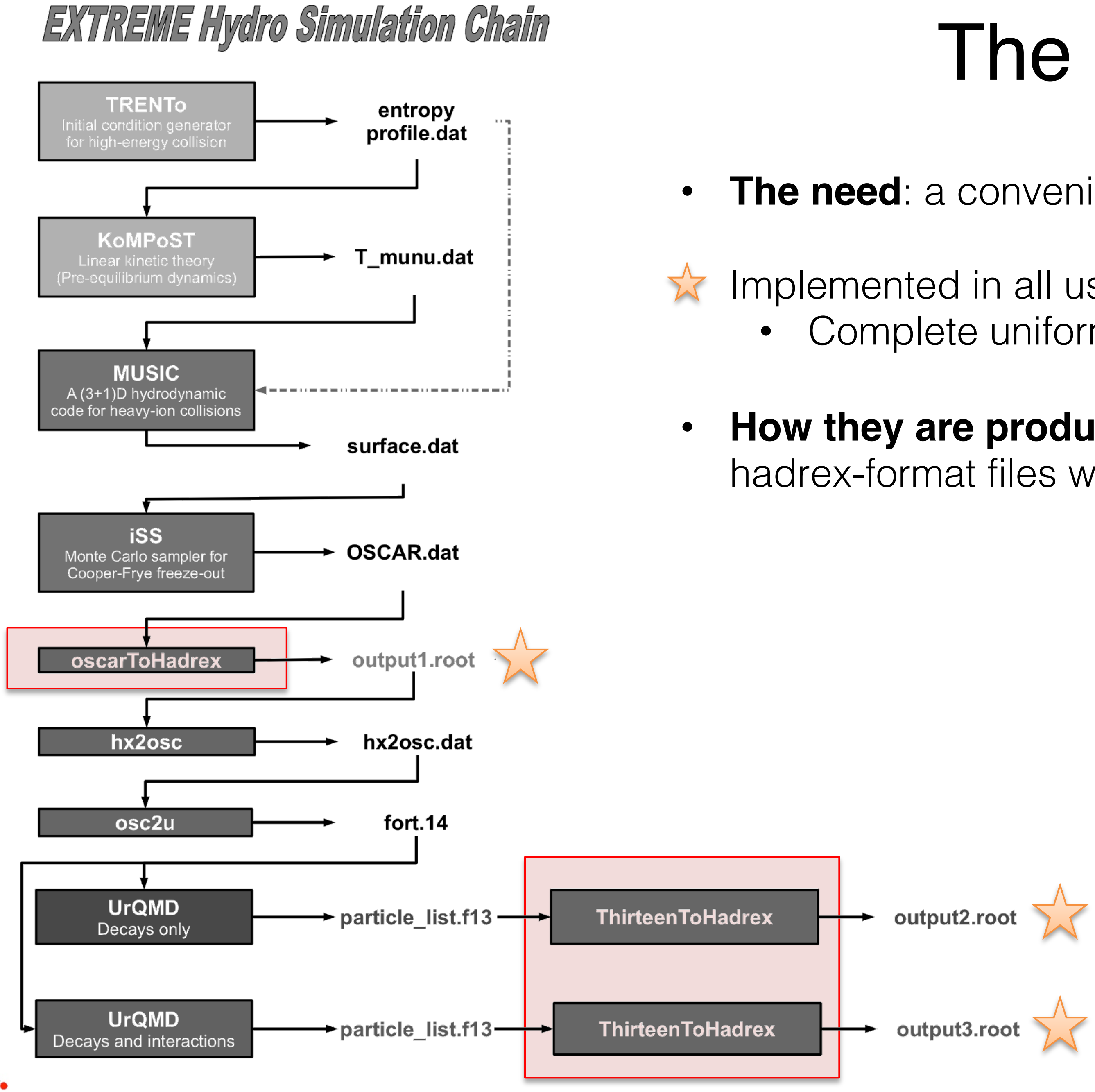
The HadrEx analysis framework

The HadrEx format



- **The need**: a convenient ROOT-based data storage format

☆ Implemented in all user-analysable formats: outputs 1, 2 and 3
  - Complete uniformity for analysis!



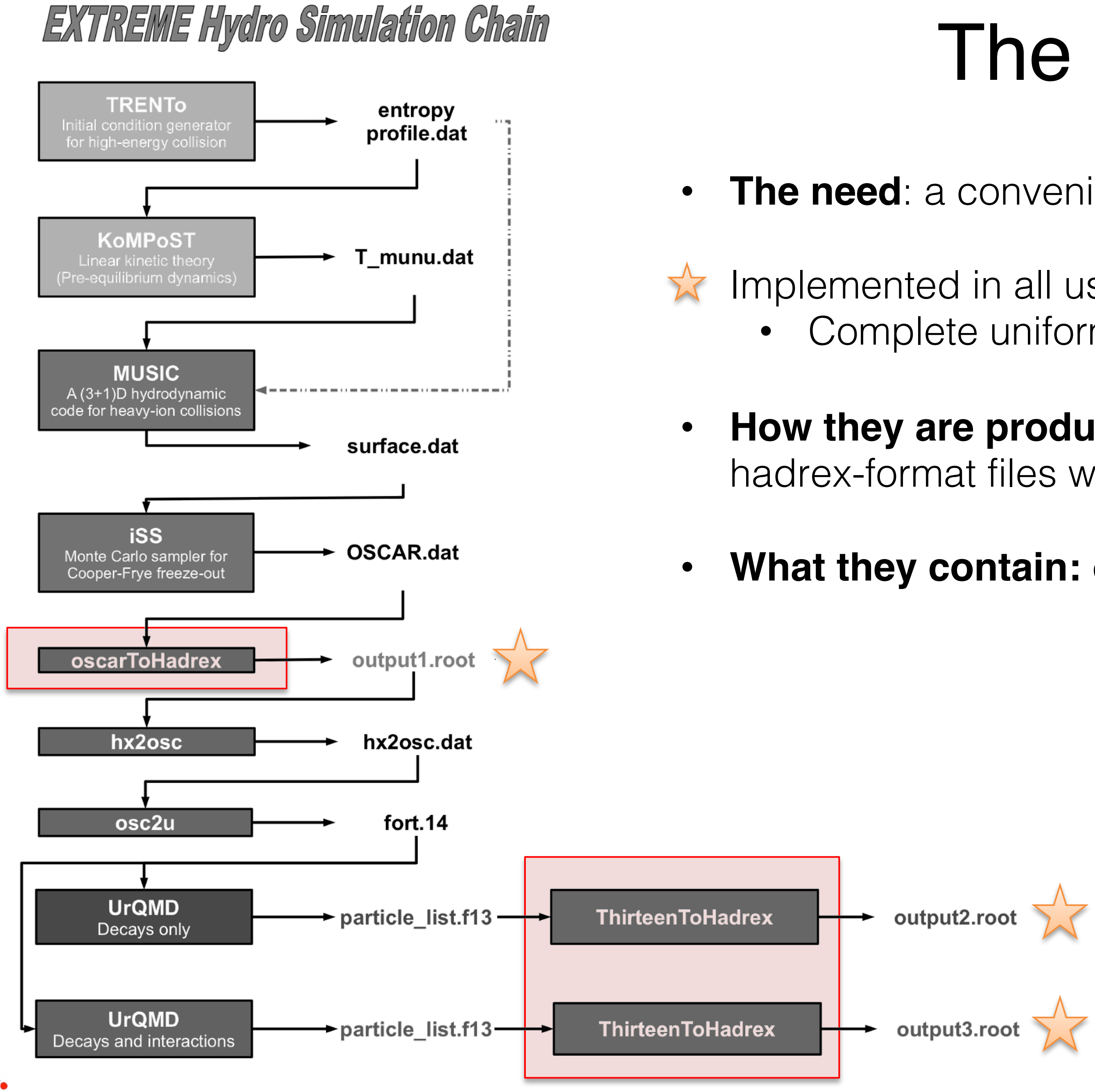
The HadrEx format

- **The need**: a convenient ROOT-based data storage format

☆ Implemented in all user-analysable formats: outputs 1, 2 and 3
  - Complete uniformity for analysis!

- **How they are produced**: converters at various stages produce hadrex-format files with various types of information

The HadrEx format

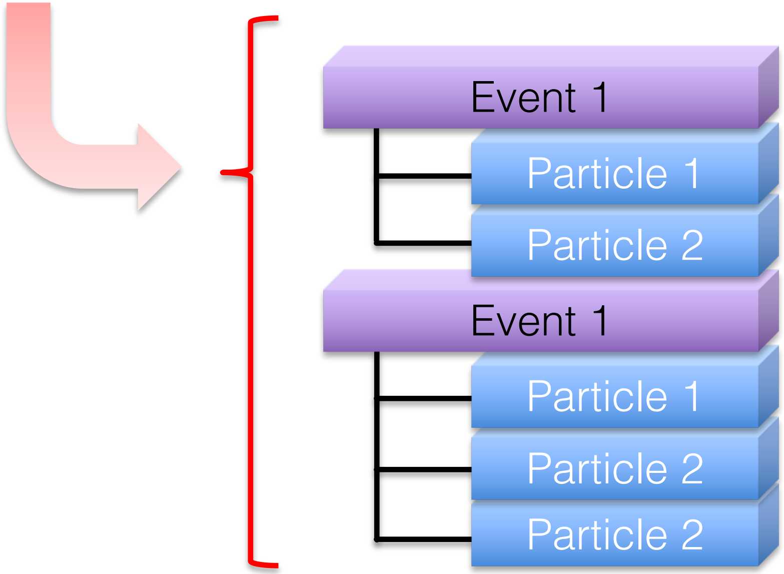


- **The need**: a convenient ROOT-based data storage format

☆ Implemented in all user-analysable formats: outputs 1, 2 and 3
  - Complete uniformity for analysis!

- **How they are produced**: converters at various stages produce hadrex-format files with various types of information

- **What they contain:** event information + a particle list



The HadrEx event and particle classes

Event

- **HxSimEvent**: a generic event class
  - TClonesArray of HxSimParticles,
  - total multiplicity at mid-rapidity + V0M acceptance (ALICE centrality selection)
  - hadronic collision history (optional)
- **HxHydroEvent**: a hydro-specific event class. Derives from HxSimEvent
  - Ncoll
  - Entropy
  - Derived centrality already there (optional)

Particle

- **HxSimParticle**: a generic particle class
  - Momentum 3-vector
  - Energy
  - PDG code
  - Mother / daughter indices (optional)
- **HxSpaceTimeParticle**: particle class with spacetime info
  - Creation position and time
  - Hadronic interaction information

C++ object-oriented paradigm:
All classes have convenient setters/getters for analysis!

In blue: links to the corresponding headers in our repository.

UNICAMP

A practical example:
Calculate p_T spectra

```
int main ( int argc, char** argv )
{
    cout<<"****************************************"<<endl;
    cout<<"      Example analysis module "<<endl;
    cout<<"****************************************"<<endl;
    
    // Check that correct number of command-line arguments
    if (argc < 2) {
        cout<<"  Improper number of arguments! (received "<<argc<<")  \n"
        <<"  Correct call: ./dopTSpectra [inputfile] [outfile]"<<endl;
        return -1;
    }
    
    TString lInputFile = argv[1];
    TString lOutputFile = argv[2];
    
    cout<<"Input .......: "<<lInputFile.Data()<<endl;
    cout<<"Output ......: "<<lOutputFile.Data()<<endl;
    
    //Loop over events, compute baryon-to-meson ratio
    HxSimEvent *event = new HxSimEvent();
    TFile *f = new TFile(lInputFile.Data(),"read");
    TTree *T = (TTree*)f->Get("T");
    
    //Loop over events, compute baryon-to-meson ratio
    TLeaf *leafEvent = T->GetLeaf("simEvent") ; //necessary for HxSimEvent typecast from any other
    
    TFile *foutput = new TFile(lOutputFile.Data(), "RECREATE");
    
    //Keep track of event counts
    TH1D *hEventCounter = new TH1D("hEventCounter","",1,-0.5,0.5);
    
    //Don't do variable binning: do very finely binned histogram instead
    Long_t lNPtBins_Uniform = 500; //1MeV/c^2 bins
    Double_t lPtMax_Uniform = 50; //GeV/c (should be plenty for anyone...)
    
    TH1D *hPtCharged       = new TH1D("hPtCharged", "", lNPtBins_Uniform, 0, lPtMax_Uniform);
    TH1D *hPtChargedYCut   = new TH1D("hPtChargedYCut", "", lNPtBins_Uniform, 0, lPtMax_Uniform);
    TH1D *hEtaCharged      = new TH1D("hEtaCharged", "", 400, -20, +20);
    TH1D *hYCharged        = new TH1D("hYCharged", "", 400, -20, +20);
```

Program interface

Setup I/O

Output Histograms

- Let's go through an example code to show how to write a simple analysis: **fill a p_T histogram for charged particles at midrapidity!**

No need to touch this, at least when getting started

This is where you should add extra output histograms in case you write an analysis

UNICAMP

A practical example:

The actual analysis

```
    cout<<"Will loop over "<<T->GetEntries()<<" events."<<endl;
    int nevents = T->GetEntries();
    for(int i=0; i<nevents; i++) {
        T->GetEntry(i);
        event = (HxSimEvent*) leafEvent->GetValuePointer();
        int nparticles = event->GetNparticles();

        //Number of Events
        hEventCounter -> Fill(0);

        //=================================================================================
        //(1) Acquire basic particle spectra
        //=================================================================================
        for(Long_t j=0; j<nparticles; j++) {
            HxSimParticle *particle = event->GetParticle(j);
            //Charged particle spectra
            if( TMath::Abs( particle->Charge() )>1e-4 ) {
                hEtaCharged -> Fill( particle->Eta() ) ;
                hYCharged -> Fill( particle->Y() ) ;
                if( TMath::Abs( particle->Eta() ) < 0.5 ) hPtCharged -> Fill ( particle->Pt() );
                if( TMath::Abs( particle->Y() ) < 0.5 )   hPtChargedYCut -> Fill ( particle->Pt() );
            }
        }


        //--- ETA Calculation ------------------
        //Only do this if I increased the counter...
        if ( i % 100 == 0 ) {
            Double_t complete = 100. * ( double ) ( i ) / ( double ) ( nevents );
            cout << "Event # " << i << "/" << nevents << " (" << complete << "%, Time Left: ";
            timer->Stop();
            Double_t time = timer->RealTime();

            //events per hour:
            lEventsPerSecond = ( ( Double_t ) ( i ) ) /time;

            timer->Start ( kFALSE );
            Double_t secondsperstep = time / ( Double_t ) ( i+1 );
            Double_t secondsleft = ( Double_t ) ( nevents-i-1 ) * secondsperstep;
            Long_t minutesleft = ( Long_t ) ( secondsleft / 60. );
            secondsleft = ( Double_t ) ( ( Long_t ) ( secondsleft ) % 60 );
            cout << minutesleft << "min " << secondsleft << "s, working at "<<lEventsPerSecond<<" Events/s..." << endl;
        }
        //--- end ETA calculation ---------------

    }
```

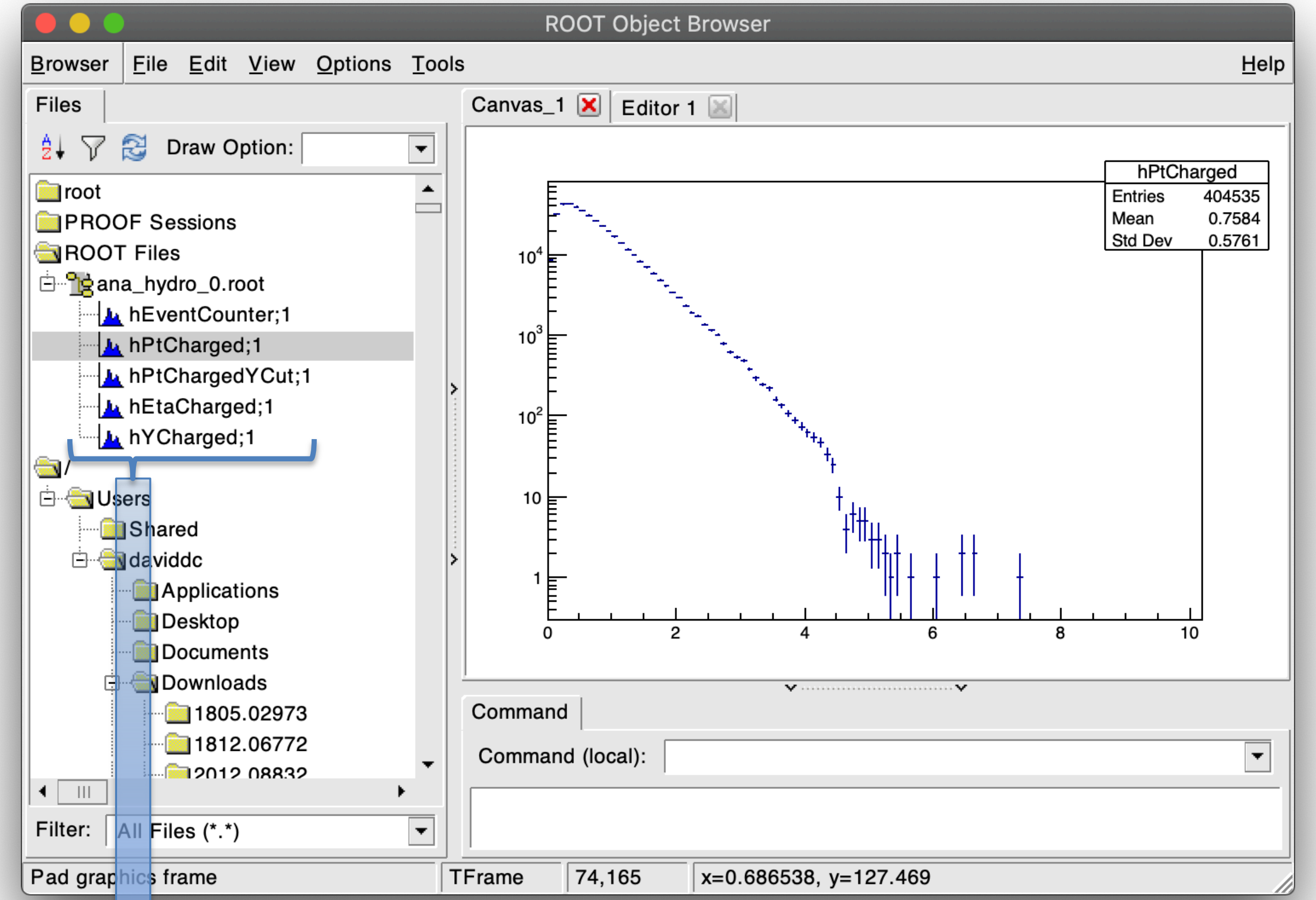
Count your events!

The analysis: fill histograms

Customize to your needs!

Calculate time to finish

Helpful to know: just a handy printout



A practical example:

The output

- **hEventCounter**: counts events
- **hPtCharged**: charged particle p_T spectra in $|\eta| < 0.5$
- **hPtChargedYCut**: charged particle p_T spectra in $|y| < 0.5$
- **hEtaCharged**: charged particle η distribution
- **hYCharged**: charged particle y distribution

UNICAMP

Processing the output

```
void GetNchAndMeanPt(){
  TFile *file = new TFile("AnalysisResults.root", "READ");
  if(!file) {
    cout<<"File 'AnalysisResults.root' not found! Please check"<<endl;
    return;
  }
  TH1D* hEventCounter = (TH1D*) file->Get("hEventCounter");
  TH1D* hPtCharged = (TH1D*) file->Get("hPtCharged");
  
  Double_t lNch = hPtCharged->Integral(1,hPtCharged->GetNbinsX()) / hEventCounter->GetBinContent(1);
  
  cout<<"Nch = "<<lNch<<endl;
  cout<<"Mean pT = "<<hPtCharged->GetMean()<<endl;
}
```

Multiplicity per event: divide by the number of events stored!

Average p_T: a simple "GetMean" call will do it

To run: `root.exe -q -b GetNchAndMeanPt.C`

```
root [0]
Processing GetNchAndMeanPt.C...
Nch = 853.449
Mean pT = 0.758421
```

You will run this later today!

WE NEED YOU!

Moving forward…

- **The HadrEx framework is there to help!**
    - But it is still constantly evolving according to our needs!
    - Custom classes can also be done for PYTHIA events, etc
- Tiago will now walk us through the details of how to get your first event generation and analysis going.
    - Ab initio simulations: from beginning to the end!

Thank you for your attention!

BACKUP