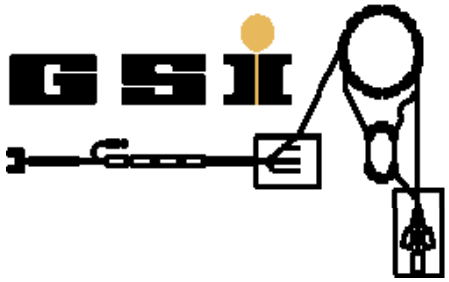


Analysis in FairRoot

Klaus Götzen
GSI Darmstadt

28. Apr. 2008

Prospects

In general an analysis toolset should...

- offer elaborate functionality to
  - handle particle candidates
  - handle lists of particles
  - do combinatorics
  - provide kinematic/vertex fitting
  - compute event related quantities ...etc

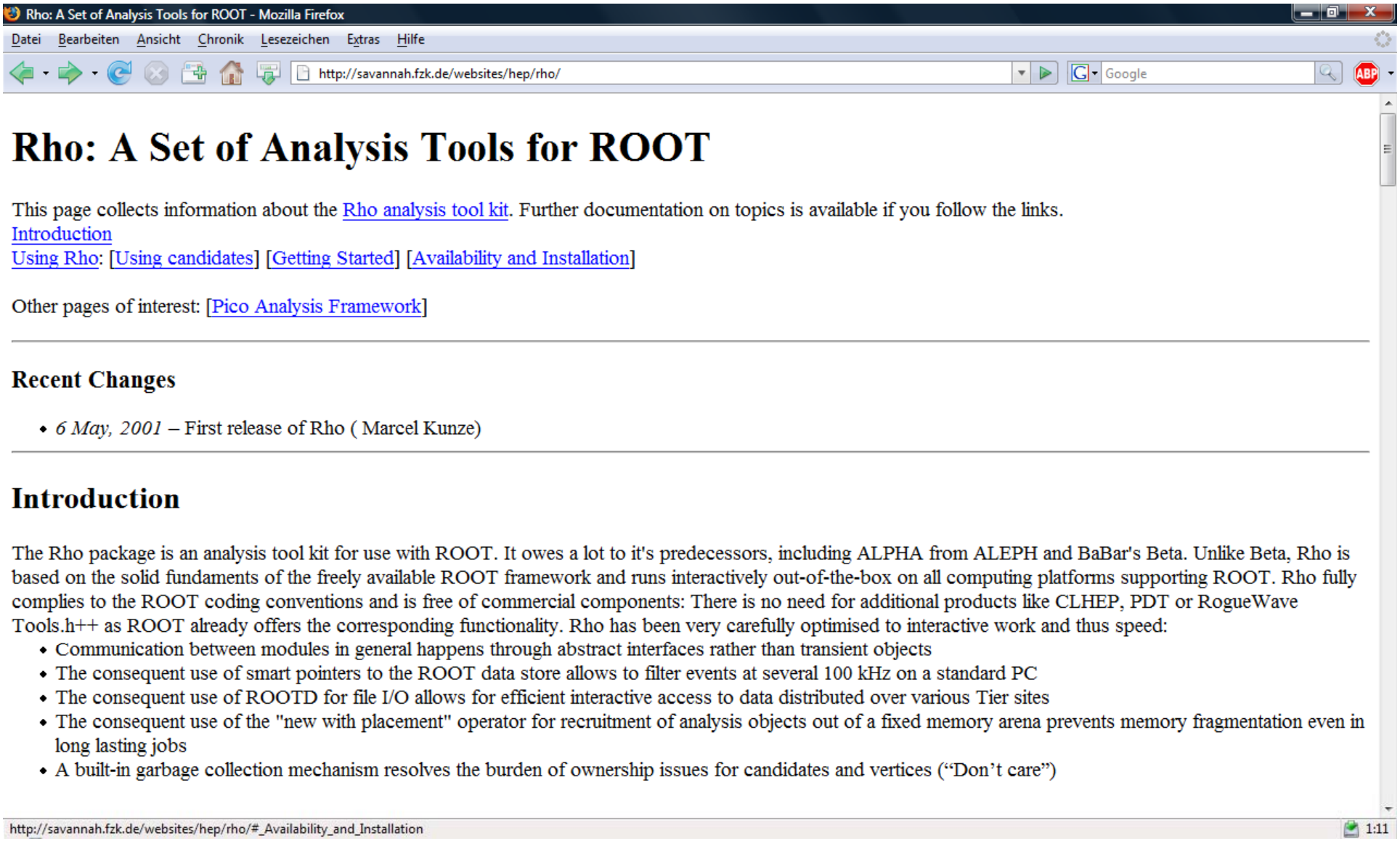
- be clear and easy to use for analysts
- have common handling for simulation (fast, full) and data
- be robust, fail-safe and performant

- **Rho analysis toolset is a good candidate**

Motivation and Objective

- Brief historical overview
  - Rho (aka PAF) was written for Babar
  - It was programmed when Beta was already existing
  - **Idea:**
    Replace slow Objectivity data access with
    lightweight and fast ROOT I/O handling
  - Users should switch from one to other without notice
  - Therefore:
    Keep interfaces and object hierarchy exactly the same as in Beta
  - Behind interfaces:
    Everything is ROOT compliant

Rho webpage



http://savannah.fzk.de/websites/hep/rho/

Rho packages

Contents (...web page snippet)

The basic Rho release consists of five packages:

RhoMath : Astatistics package to enhance ROOT with error classes, error functions, consistency calc.
RhoBase : Candidate classes, base classes, list processing etc.
RhoTools : Analysis-level tools, boosters, event shape variables, etc.
RhoExamples : Analysis-level examples of techniques you might want to use in your program
RhoParameters : Particle data tables, neural network files and schema definitions

Independent optional tools packages

RhoHistogram : A Heptuple compliant interface to support THistogram and TTuple classes
RhoNNO : The ROOT implementation of the Neural Network Objects package
RhoGA : MIT's library for implementation of genetic algorithms

Additional packages for event management, readers and writers for various applications

RhoManager : Event manager classes for modules, I/O, object persistence etc.
PAFSchema and KangaSchema : Readers and writers for data of the BABAR experiment
RhoSelector : A sample of event selectors, PID selectors and vertex selectors
RhoConditions : A standalone sample implementation of a ROOT based conditions database for beam parameters etc. Databases are transparently accessible over WAN using ROOTD.

used in PandaRoot at the moment

The Basis - RhoBase

- **TFitParams**: basic candidate parameters
  - Position, LorentzVector, Helix parameters, Covariance matrices
  - Algorithms for calculation of error matrices

```
class TFitParams : public TObject
{
protected:
    // the params
    Char_t  fCharge;              // The electrical charge
    Float_t fXposition,           // The origin in x
            fYposition,           // The origin in y
            fZposition,           // The origin in z

            fXmomentum,           // The momentum in x
            fYmomentum,           // The momentum in y
            fZmomentum,           // The momentum in z
            fEnergy,              // The total energy

            fErrP7[MATRIXSIZE],   // The symmetric 7*7 error matrix
            fParams[5],           // The helix fit parameters
            fCov[15];             // The helix error matrix

    ...
```

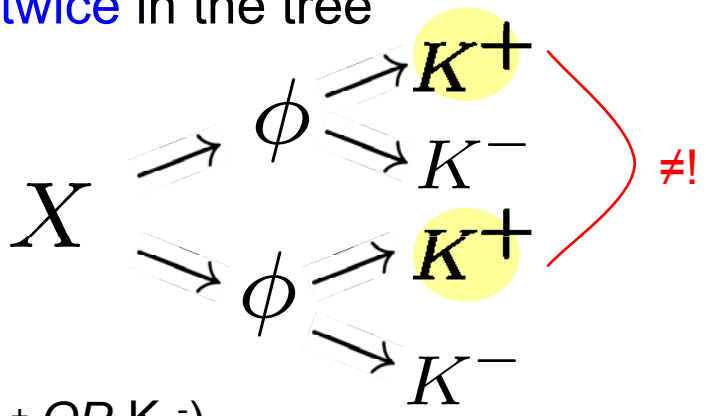
The Basis - RhoBase

- **TCandidate**: basic particle candidate
  - Mother/Daughter links, fit constraints, PDG type
  - Interfaces to MicroCandidate, MC truth, PID info, ...
  - Methods to combine w/ other TCandidate

```
class TCandidate : public TFitParams
{
TCandidate* fTheMother;                 //! The mother

protected:
    VAbsVertex* fDecayVtx;              //! Vertex
    TCandList *fDaugList;               //! List of daughters
    TCandidate* fDaughters[5];          //! Array of daughters
    Short_t nDaug;                      //! Number of daughters

    TConstraint* fConstraints[5];       //! Array of constraints
    Short_t nCons;                      //! Number of constraints

    VAbsTruth* fTruth;                  //!  Pointer to MCTruth info
    const TParticlePDG* fPdtEntry;      //! Pointer to particle database

    VAbsMicroCandidate* fMicroCand;     //!Pointer to micro data
...
```

Details: Overlap mechanism

- Problem:
  - reconstruct a complicated decay tree
  - make sure, that no final state appears twice in the tree
  - check should be fast!



- Idea:
  - Make use of bit markers
  - Example
    - K^+_1: 10000000
    - K^+_2: 01000000
    - K^-_1: 00100000
    - K^-_2: 00010000

ϕ_1: 10100000 (K_1^+ *OR* K_1^-)
ϕ_2: 10010000 (K_1^+ *OR* K_2^-)
ϕ_3: 01100000 (K_2^+ *OR* K_1^-)
ϕ_4: 01010000 (K_2^+ *OR* K_2^-)

$X_1 = \phi_1 + \phi_2$: overlap ϕ_1 *AND* ϕ_2 = 10000000 ≠ 0 → invalid!
$X_2 = \phi_1 + \phi_4$: overlap ϕ_1 *AND* ϕ_4 = 00000000 → valid!

  - Very fast with binary & and |, even in complicated decay trees!

The Basis - RhoBase

- **TCandList**: handle lists of TCandidates
  - Combine lists, apply selectors on lists
  - cares about multiple counting, overlap

```
class TCandList : public TNamed {
private:
    TObjArray*   fOwnList;        // This holds the candidates
    Bool_t        fFast;          // Fast mode = no error matrices

public:

void Combine(TCandList& l1, TCandList& l2,VAbsVertexSelector* s=0);
void CombineAndAppend( TCandList& l1,TCandList& l2,VAbsVertexSelector* s=0);

void Select(VAbsPidSelector* pidmgr);
void Select(TCandList& l, VAbsPidSelector* pidmgr);
void Select(TCandList& l, Bool_t (*selfunc)(TCandidate&) );

Int_t OccurrencesOf(TCandidate& );
Int_t Remove(TCandidate& );         // Returns #removed cands
Int_t RemoveFamily(TCandidate& );   // Returns #removed cands
Int_t RemoveClones();               // Returns #removed cands

...
```

Details: TCandidate creation

- Creating new objects and deleting is time consuming
  - Try to use always the same memory for allocation, aka new with placement
- Rho realizes this with the TFactory:

```
void TCandList::Put(const TCandidate& c, Int_t i )
{
    TCandidate *newCand = TFactory::Instance()->NewCandidate(c);

    if (i<0) fOwnList->Add(newCand);
    else (*fOwnList)[i] = newCand;
}
```

```
TCandidate* TFactory::NewCandidate(const TCandidate &c)
{
    if (fgCandBuffer==0) fgCandBuffer = new TClonesArray("TCandidate");
    int current = fgCandPointer++;
    if (current>fgCandWatermark) fgCandWatermark = current;
    ...
    new ((*fgCandBuffer)[current]) TCandidate(c);
    return GetCandidate(current);
}
```

Rho and PandaRoot

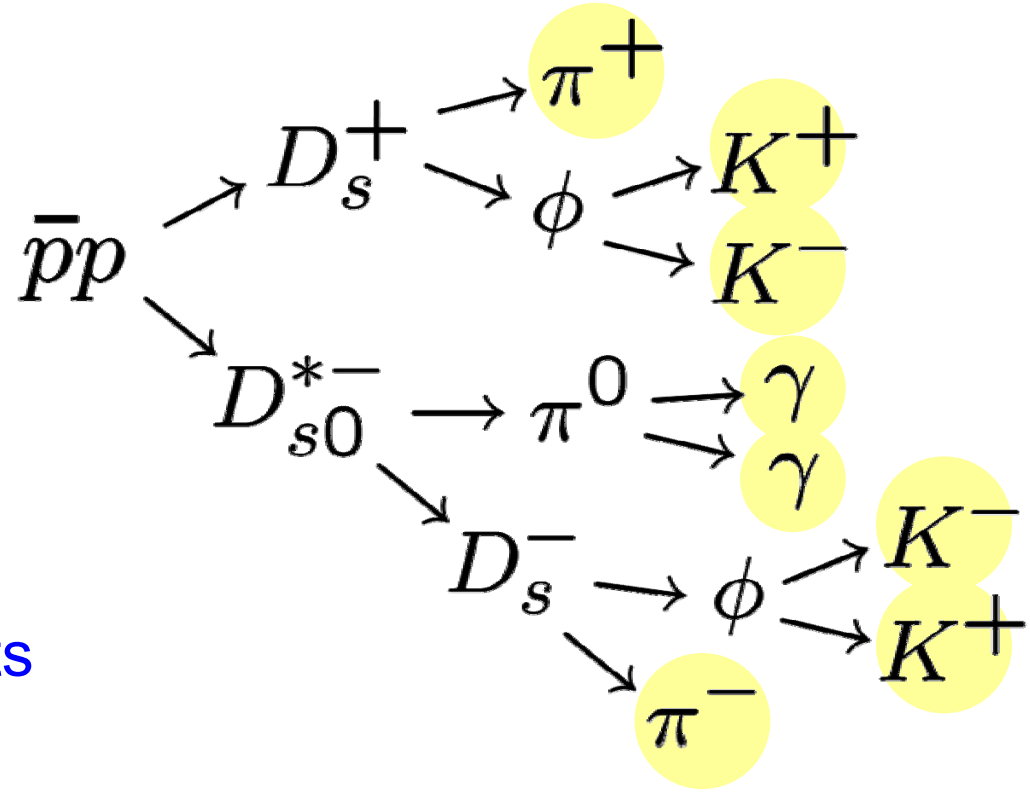
- **What are we using so far**
  - TCandidate, TCandList in combination with Fast Simulation (preliminary also with Full Simulation)
  - In CbmTask pick up simulated tracks
  - Store them in a TClonesArray("TCandidate")

- **Three different styles to perform user analysis**
  - directly in a ROOT macro in CINT (almost interactive...)
  - in a CbmTask
  - by using PndSimpleAnalysis *(new → preliminary!)*
    - inspired by SimpleCompositions in BaBars beta framework
    - configure your combinatorics in a .cfg file
    - combinatorics are done automatically
    - charged conjugates are produced automatically

  - Speed (≈ 1-2 kHz) is almost independent of method!!

Simulation example: $\bar{p}p \rightarrow D_s D_{s0}$

- Decay Tree @ $\sqrt{s}$ = 4.306 GeV (p_{pbar} = 8.8931 GeV/c)



- 8 final states
- Data
  - 20k signal events
- Reconstruction
  - full exclusive

Analysis in CINT macro

```
for (Int_t j=0; j< num;j++)
{
  t->GetEntry(j);
  TFactory::Instance()->Reset();
  chargedCands.Cleanup();
  neutralCands.Cleanup();

 for (i1=0; i1<fChrgCands->GetEntriesFast(); i1++){
    tc = (TCandidate *)fChrgCands->At(i1);
    chargedCands.Add(*tc);}

 for (i1=0; i1<fNeutCands->GetEntriesFast(); i1++){
    tc = (TCandidate *)fNeutCands->At(i1);
    neutralCands.Add(*tc);}

  plusCands.Select(chargedCands  ,plusSel);
  minusCands.Select(chargedCands ,minusSel);

  kpCands.Select(plusCands    ,kSel);
  kmCands.Select(minusCands   ,kSel);
  piCands.Select(chargedCands ,piSel);

  phiCands.Combine(kpCands,kmCands);
  phiCands.Select(phiMSel);

  dsCands.Combine(phiCands,piCands);
  dsCands.Select(dsMSel);

  pi0Cands.Combine(neutralCands,neutralCands);
  pi0Cands.Select(pi0MSel);

  ds0Cands.Combine(dsCands,pi0Cands);

  ppCands.Combine(ds0Cands,dsCands);
  ppCands.Select(neutralSel);
```

Loop over all events

Do some cleanup

Fill TCandList with TCandidates

... positive and negative charged ...

...kaon candidates and pion candidates

form ϕ from K^+ and K^-

D_s from ϕ and $\pi^{\pm}$

π^0 from 2 γs

D_{s0} from D_s and π^0

and finally the pp candidates
... only neutral combinations...

Definition of Lists + Selectors

```
TFile* f = new TFile(fname.Data());
TTree *t = f->Get("cbmsim") ;
TClonesArray *fCands = new TClonesArray("TCandidate");
t->SetBranchAddress("PndCandidates",&fCands);

TCandidate *tc;

TCandList allCands,neutralCands,chargedCands, plusCands,minusCands;
TCandList kpCands,kmCands,piCands;
TCandList phiCands,pi0Cands,dsCands,ds0Cands,ppCands;

TPidChargedSelector *chargedSel = new TPidChargedSelector;
TPidNeutralSelector *neutralSel = new TPidNeutralSelector;
TPidPlusSelector    *plusSel    = new TPidPlusSelector;
TPidMinusSelector   *minusSel   = new TPidMinusSelector;
TPidNeutralSelector *gammaSel   = new TPidNeutralSelector("gammaSel","gamma");

TPidMassSelector *phiMassSel    = new TPidMassSelector("phiMassSel" , 1.0195 , 0.010);
TPidMassSelector *pi0MassSel    = new TPidMassSelector("pi0MassSel" , 0.1350 , 0.005);
TPidMassSelector *dsMassSel     = new TPidMassSelector("dsMassSel"  , 1.9685 , 0.010);

TPidSimpleKaonSelector *kSel    = new TPidSimpleKaonSelector();
kSel->SetCriterion("loose");
TPidSimplePionSelector *piSel   = new TPidSimplePionSe
piSel->SetCriterion("loose");
```

TCandList: define all the lists we need

TSelectors: various selectors
- neutral/charged
- pid, vertex

Analysis in a CbmTask

- The analysis code is the same
  - copy/paste to the Task::Exec function

- Macro snippet to run then looks like

```
...
CbmRunAna *fRunA= new CbmRunAna();
fRunA->SetInputFile(infile);
fRunA->SetOutputFile(outfile);

//append the analysis task
PndAnalysis *anaTask=new PndAnalysis();
fRunA->AddTask(anaTask);

fRunA->Init();
if (nevts==0) nevts=10;
fRunA->Run(0,nevts);
...
```

Analysis with PndSimpleAnalysis

- Config file 'analysis.cfg' looks like:

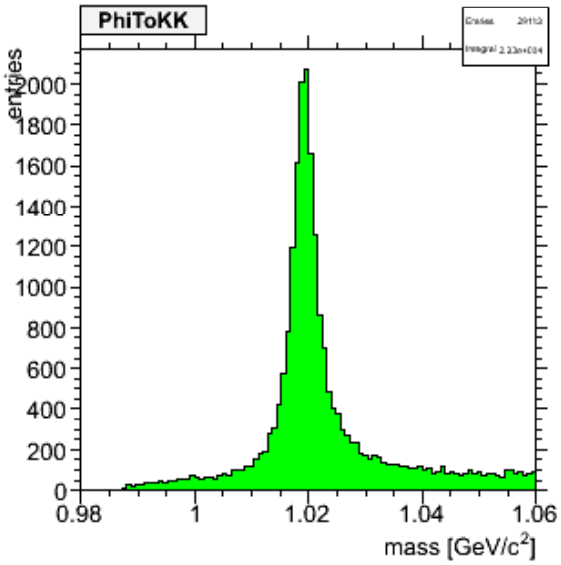
```
DefineList PhiToKK
  decayMode phi -> K+ K-
  daughterList KaonLoose
  daughterList KaonLoose
  histogram 0.98 1.06
  select Mass 1.0 1.06
End

DefineList DsToPhiPi
  decayMode D_s+ -> phi pi+
  daughterList PhiToKK
  daughterList PionVeryLoose
  histogram 0.03
  select Mass 0.03
End

DefineList Pi0
  decayMode pi0 -> gamma gamma
  daughterList Neutral
  daughterList Neutral
  histogram 0.11 0.16
  select Mass 0.03
End
```

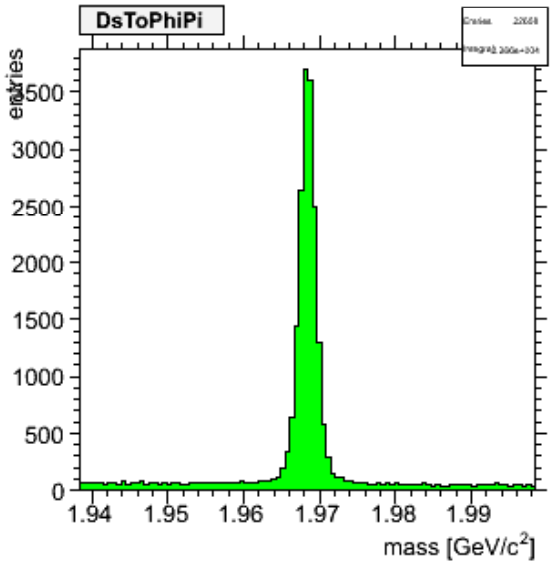
```
DefineList Ds0ToDspi0
  decayMode D*_0s+ -> D_s+ pi0
  daughterList DsToPhiPi
  daughterList Pi0
  histogram 2.217 2.417
  select Mass 0.2
End

DefineList pbarpToDs0Ds
  decayMode pbarp -> D*_0s+ D_s-
  daughterList Ds0ToDspi0
  daughterList DsToPhiPi
  dumpList
  histogram 4.185 4.385
End
```

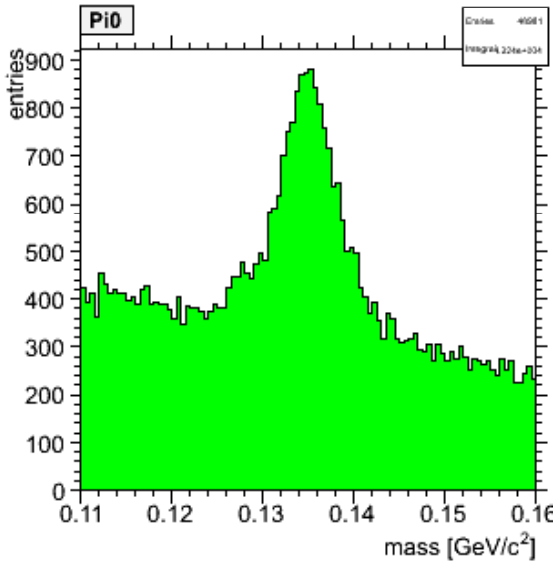
- Code in macro

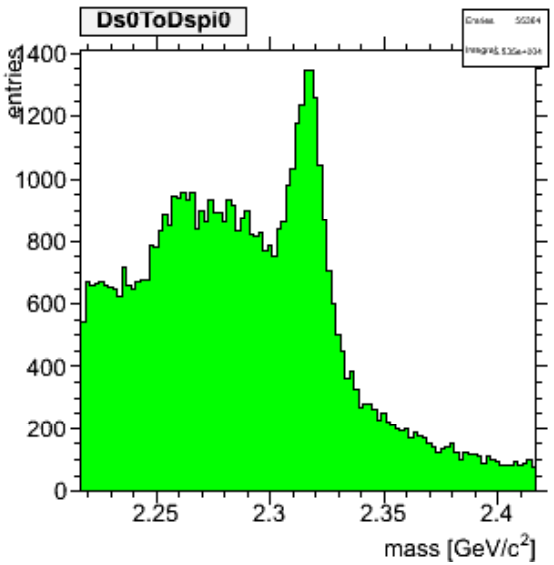
```
//append the analysis task
  PndSimpleAnalysis *anaTask=newPndSimpleAnalysis("analysis.cfg");
  fRunA->AddTask(anaTask);
```

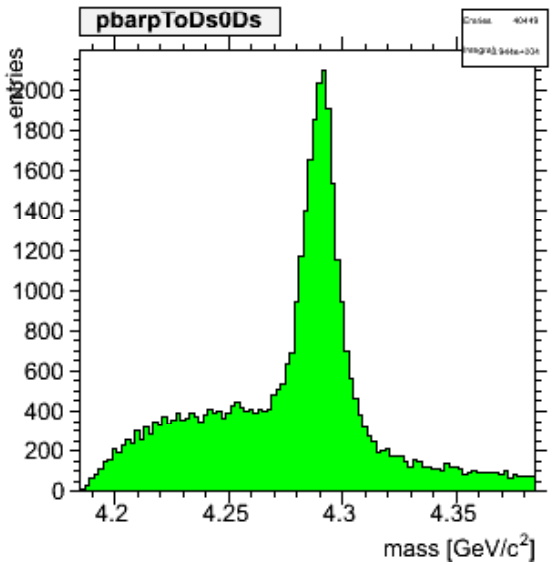
Analysis example: $\bar{p}p \rightarrow D_s D_{s0}$











... and the results...

Plans for PndSimpleAnalysis

- Provide possibility to perform fitting
  (at the moment PandaRoot generally lacks of ready-to-use Vtx/Kin Fitters ... work in progress)
- Implement more selection options like
  - energy
  - momentum
  - fit probability
  - requirements for event variables
- Perform MC truth match
- Dump out an NTuple conserving the structure of the decay tree (see BtaTupleMaker in beta)

Conclusion + Outlook

- Rho Analysis Toolset is the basis of PandaRoot analysis
- Easy and intuitive to use
- Analysis in PandaRoot is not completely mature but already usable (at the moment for Fast Simulations)
- Due to clear interfaces it can easily applied to Full Simulation
- Speed seems reasonable around 1 kHZ (without fitting or other complicated stuff)

⇨ **Things looks quite promising... :-)**

BACKUP

Fast Simulations – Concept

Full Simulation

Fast Simulation

Event Generation

Particle Transport
Digitization
Calibration
Reconstruction

Effective parametrization
- acceptance cuts
- resolution smearing
- PID info

Physics Analysis

Fast Simulations - Structure

- Implementation of the subdetectors as individual classes
  - Calorimeters: EmcBarrel, EmcFwCap, EmcBwCap, EmcFS
  - Tracking/Vtxing: Mvd, Stt, Tpc, MdcTS, MdcFS
    (+ EffTracker → next slide!)
  - PID: DrcBarrel, DrcDisc, Rich, Tof (+ dE/dx from trackers)
- Detectors can be added individually to detector setup and configured with parameters remotely
- Each detector provides acceptance info and accuracy of detection of various quantities for each particle
  - EMCs: dE, $d\phi$, $d\theta$
  - Trackers/Vertexer: dp, $d\phi$, $d\theta$, dx, dy, dz
  - PID: $d\theta_C$, dt, d(dE/dx)
- When detected, the particles 4-vector + position is altered according to resolution, and PID info is attached
- New: neutral hadronic split offs can be added! (see later!)

Feature – hadronic split offs

- According to full simulation (and reality!):
  finite probability, that charged particles (hadrons)
  produce fake neutrals in EMCs aka hadronic split offs

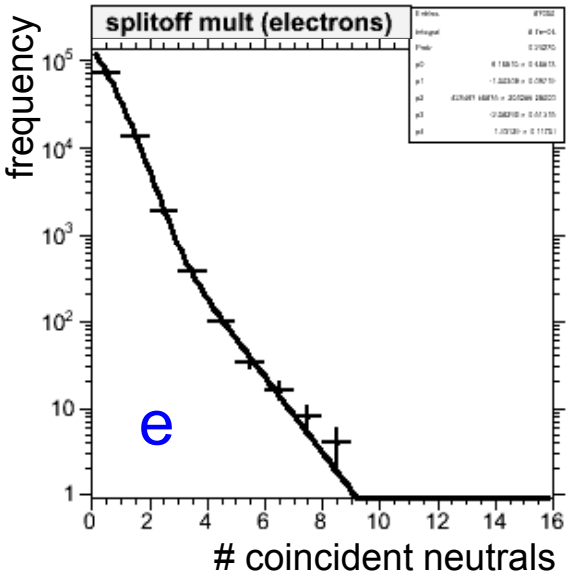
- Signal channels with charged and neutral final states
  might considerably suffer from that

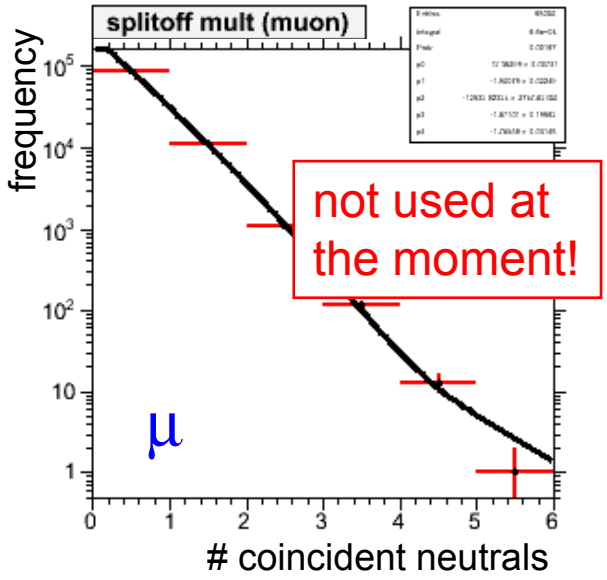
- To make Fast Sim more realistic:

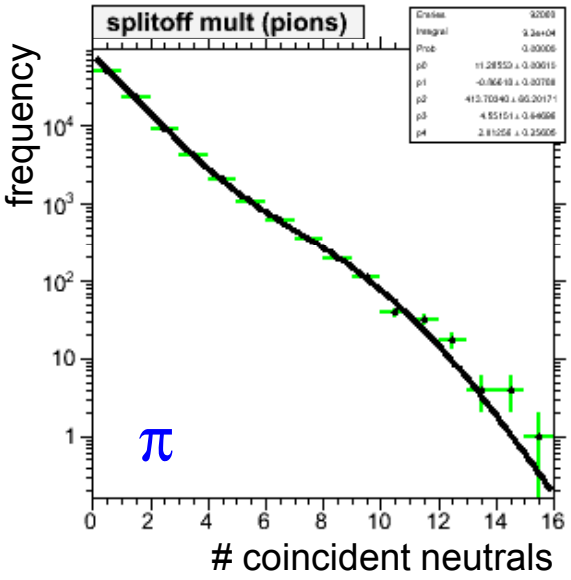
  Add parametrized neutrals to event!

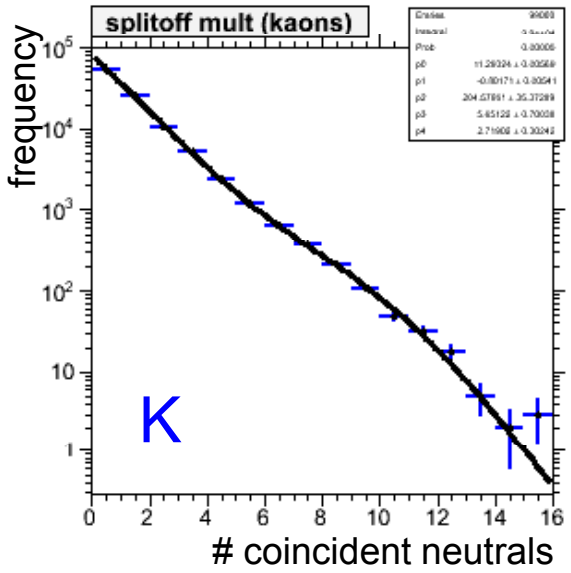
- Determine distributions relative to incident track from full simulation single track events

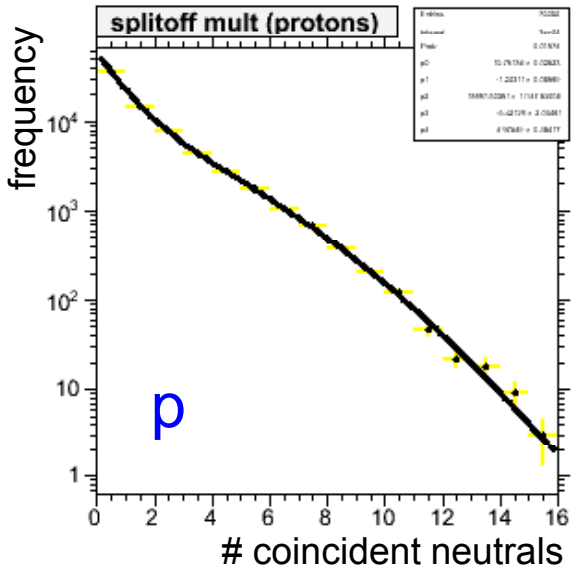
Split off distributions - multiplicity





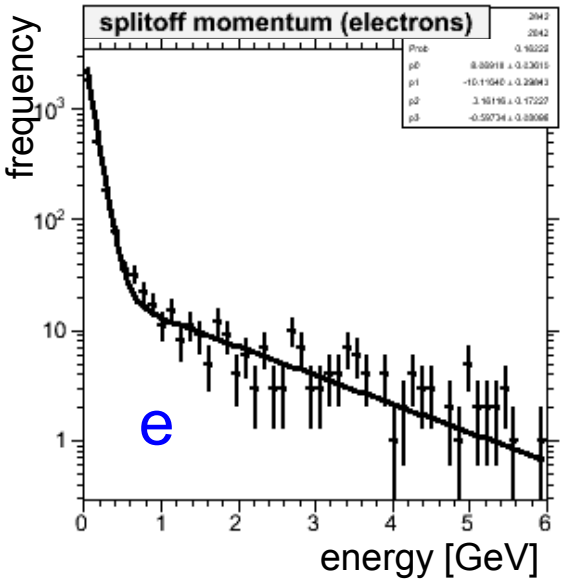


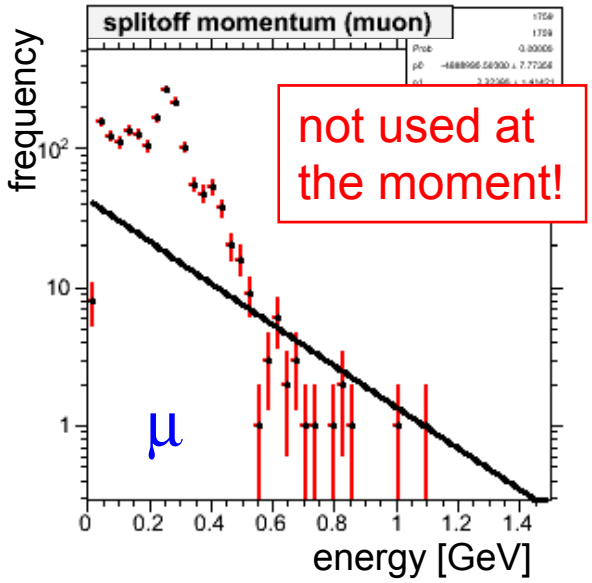




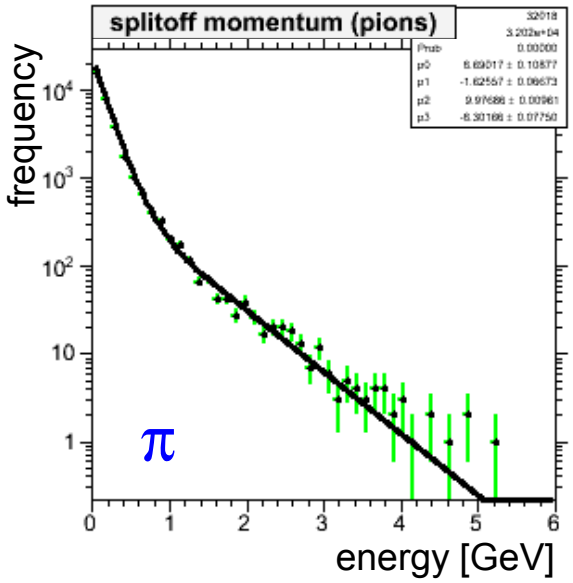
Multiplicities for e, μ, π, K, p

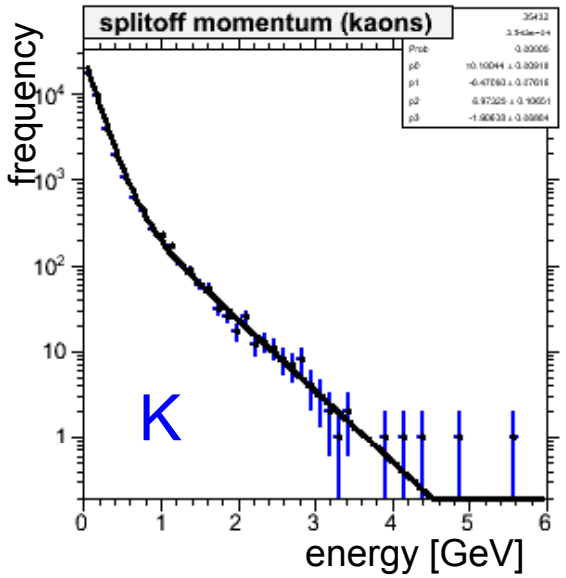
Split off distributions - energy

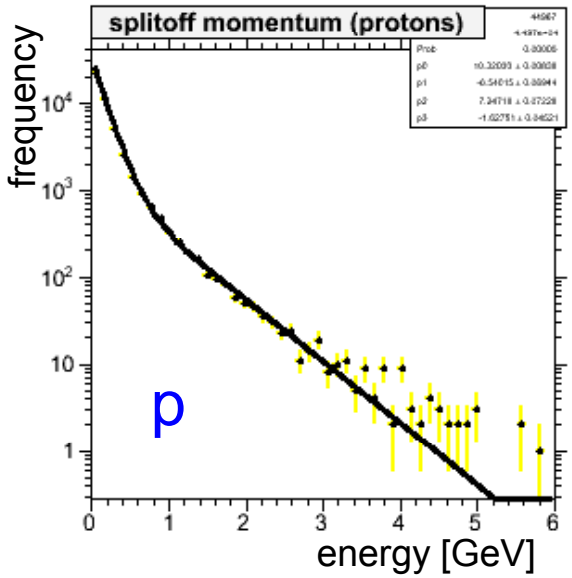




not used at the moment!



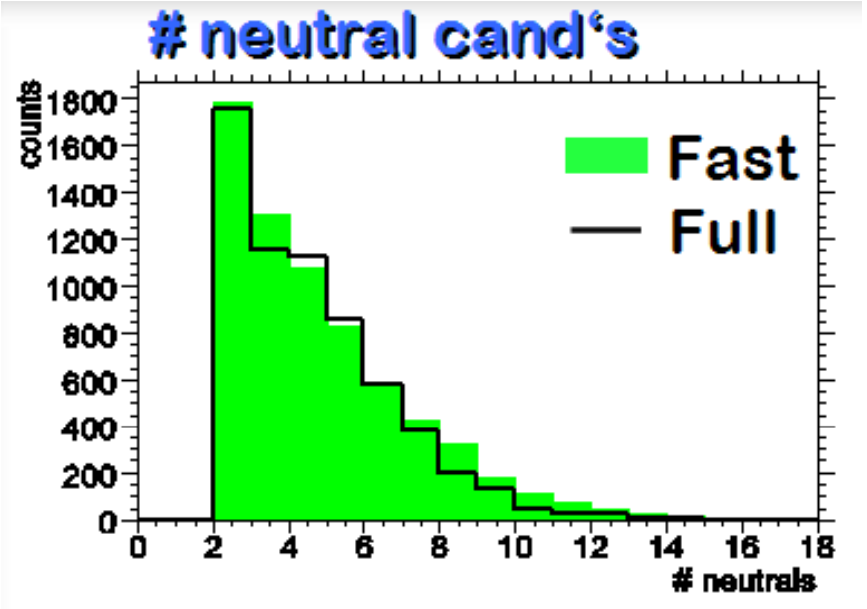


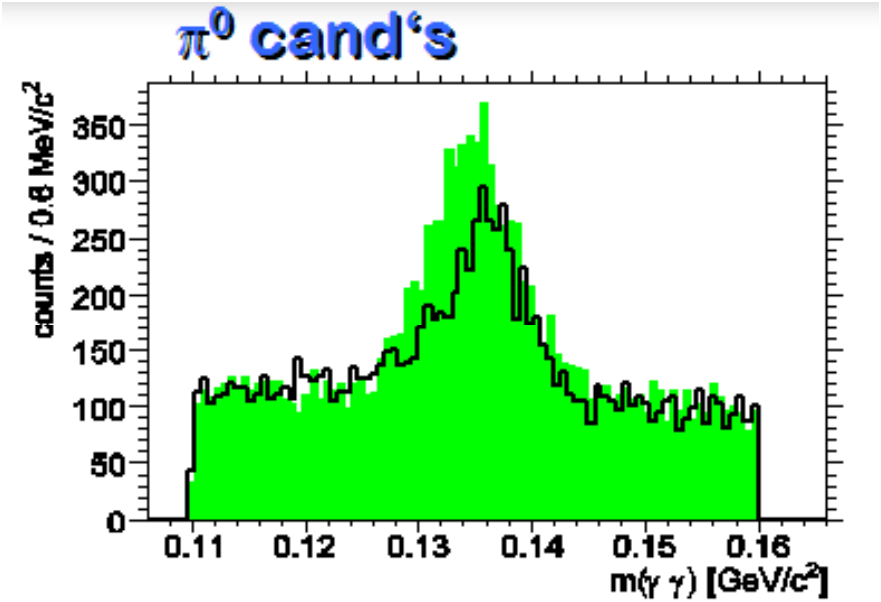


energies
e, μ, π, K, p

Split offs – Compared to Full Sim

- Overall multiplicity per $\bar{p}p$ candidate are quite reasonable (channel: $\bar{p}p \rightarrow D_s D_{s0}$)





- π^0 mass shape and background also looks quite ok

Performance

- Last but not least: How fast is Fast Sim?
- Example: signal events, DPM events on lxi010.gsi.de

- Generator:
  - EvtGen w/ output to file : ≈ 150 Hz → seems to be I/O limited
  - EvtGen w/o output to file : ≈ 2000 Hz
  - DPMEvtGen : ≈ 1600 Hz
- Simulation:
  - Signal channel D_sD_{sJ} : ≈ 1200 Hz (w/ split offs)
  - Signal channel $\eta_c \to \eta\ \pi^0\ \pi^0$: ≈ 2100 Hz
  - DPM : ≈ 1800 Hz (w/ split offs)
- Analysis (obviously strongly depends on what you do...):
  - Signal channel D_sD_{sJ} : ≈ 400 Hz
  - DPM : ≈ 600 Hz