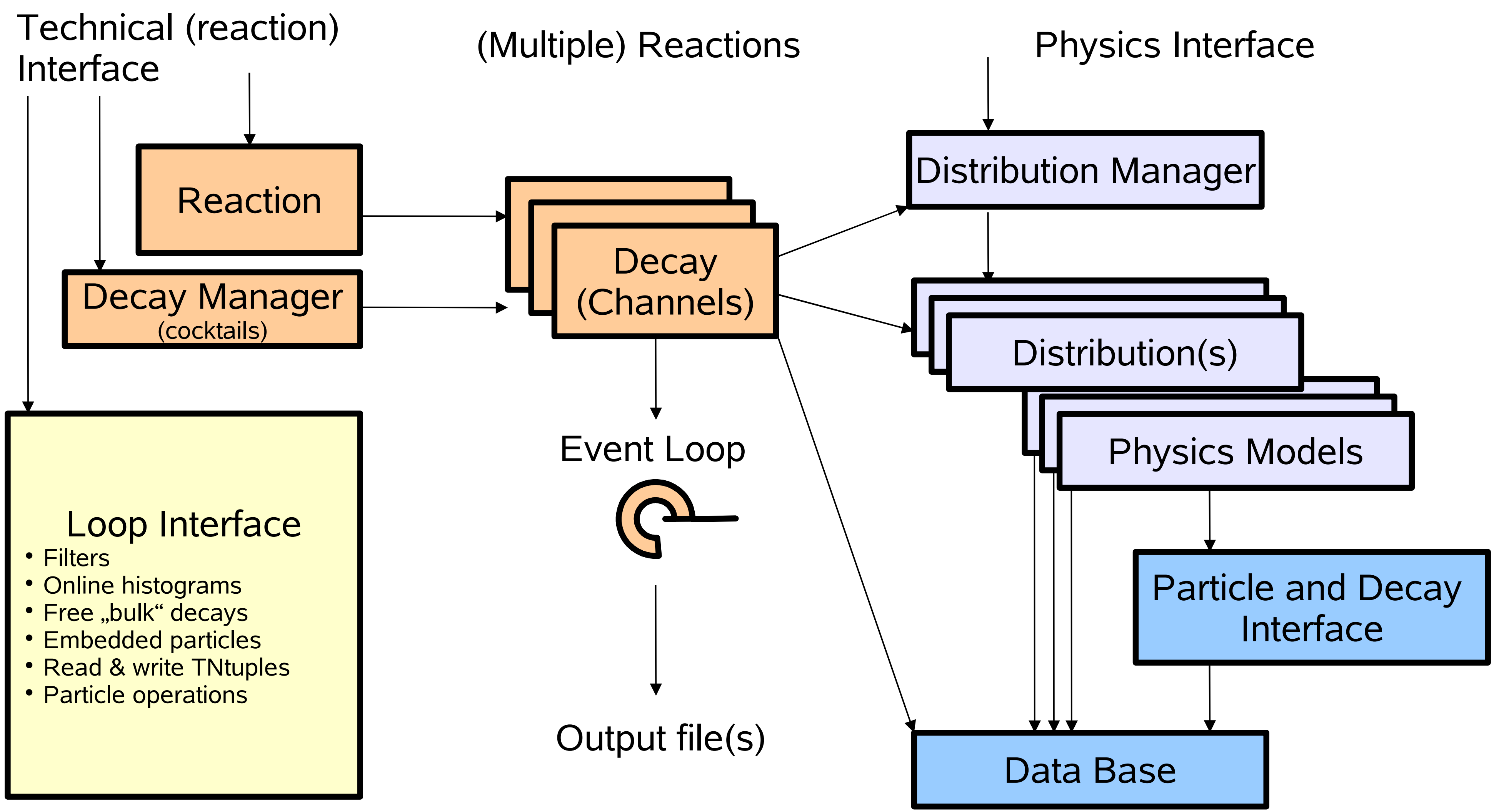
Pluto: Update

- New features (finalized with v5.37)
- Advanced scripting
  - ntuple, histogram, filters
  - Applications
- How to implement a new decay
  - Template for possible other solutions: rare η decays
  - $\eta \rightarrow \gamma^* \gamma^* \rightarrow$ e+e-e+e-

Distributions

- Interface to PChannel and PReaction



Particle List

- Particle & decays defined in data base
- Aliases

```
root [3] listParticle("D+");
Database key=73
Database name=D+
Pluto particle ID=36
Particle static width [GeV]=0.120000
Particle pole mass [GeV]=1.232000

Alias: Delta+
Alias: Delta(1232)+

This particle decays via the following
modes:
Database key=186
Database name=Delta+ --> p + pi0
Decay index=46
Branching ratio=0.662957
Decay product 1->Database name=pi0
Decay product 2->Database name=p

Database key=187
Database name=Delta+ --> n + pi+
Decay index=47
Branching ratio=0.331478
Decay product 1->Database name=pi+
Decay product 2->Database name=n

Database key=188
Database name=Delta+ --> p + photon
Decay index=48
Branching ratio=0.005525
Decay product 1->Database name=g
Decay product 2->Database name=p
..........
```

```
.........
Database key=189
Database name=Delta+ --> dilepton + p (Dalitz)
Decay index=49
Branching ratio=0.000040
Decay product 1->Database name=p
Decay product 2->Database name=dilepton
```

Interfaces to Database

- makeStaticData() -> variables

```
useDummyModel.C

    //Add a bunch of dummy particles
    makeStaticData()->AddParticle(-1,"A", 1.2);
    makeStaticData()->SetParticleTotalWidth("A",0.3);
    makeStaticData()->SetParticleBaryon("A",1);
    makeStaticData()->AddParticle(-1,"b", 0.5);
    makeStaticData()->AddParticle(-1,"c", 0.3);


    makeStaticData()->AddDecay(-1,"A -> b + c", "A", "b,c", 1.);
    makeStaticData()->AddDecay(-1,"A -> b + b", "A", "b,b", 1.);
```

- makeDynamicData() -> models

```
Double_t Gamma = makeDynamicData()->GetParticleTotalWidth(mass[0],pid);
```

- Conclusion: never touch PStdData!

Scripting

Pluto Script Manual

January 4, 2011, Pluto version 5.37

- One single platform for communication and configuration
- Combination of histograms and script commands
  - Filtering
  - Smearing
  - „Virtual detectors“
- Script commands: sufficient to do the „job“ but not high-level

„Hello World“

- Standalone:

```
makeGlobalBatch()->Execute("echo Hello world");
```

- or:

```
PBatch * batch = new PBatch();
batch->AddCommand("....");
batch->AddCommand("....");
batch->Execute();
```

„compiled“ during run-time

Very fast (like compiled C++ code)

- Inside the event loop:

```
r->Do("echo Yet another event....");
```

Variables

Variables are always Double_t's and assigned automatically without any definition or constructor:

```
"myvar = 0.2;"
```

Connection to ROOT-macro:

makeflag

```
makeStaticData()->GetBatchValue("myvar",0);
```

Print:

```
"echo The value of myvar is $myvar"
```

-> Event-by-event debugging

Operations

- Based on TFormula but extended

The batch script can use all operations which are provided by TFormula. This means, all normal operations like “+”, “-”, “*”, “/”, but also boolean, are accessible. In particular, all functions of TMath can be used.

```
"echo $myvar; myvar = myvar + 0.1; echo $myvar;"
```

```
<PBatch> 0
<PBatch> 0.1
```

Tests

- Normal tests:

```
"if condition; commands;"
"if myvar < 0.3; echo myvar too small;"
```

- Tests on equality:

```
"myvar = 1;   if(myvar ~ 1); echo myvar is one"
"myvar = ...; if(myvar); echo myvar is not zero";
"myvar = ...; if(!myvar); echo myvar is zero"
```

Jumps

- Goto:

```
"label: ..."
"goto label;"
```

- Example:

```
"myvar = 0.0; loop: myvar = myvar + 0.1;
   if myvar < 0.7; echo $myvar; goto loop;"
```

- Gosub (subroutine):

```
"gosub label;"
"..."
"label: ..."
"..."
"return;"
```

- Return to event:

```
"exit;"
```

PParticle c'tors

The P3M constructor

This constructor uses the 3-momentum (p_x, p_y, p_z) and the invariant mass as arguments. Syntax:

```
"mypar = P3M(px,py,pz,mass);"
```

The following example creates a particle with eta mass and 2 GeV momentum in z-direction:

```
"mypar = P3M(0,0,2.,0.547);"
```

The P3E constructor

This constructor uses the 3-momentum (p_x, p_y, p_z) and the energy as arguments. Syntax:

```
"mypar = P3E(px,py,pz,e);"
```

Copy constructor

New particle objects are created automatically. Syntax:

```
"newpar = mypar;"
```

Methods

Script objects can use all public methods of `PParticle` (and therefore also of the `TLorentzVector`) which uses only `void`'s, `Int_t`'s, or `Double_t`'s as arguments/return values.

```
"mypar->SetID(17);"

"mass  = mypar->M();"
"angle = mypar->Theta();"
```

Access to particles of event

- Via [ ], the particle name, and the position:

```
"mypar = [id,num]"
"myvar = [id,num]->...()"
```

- Example:

```
"[eta,1]"
"[eta]"
```

The script can access only particles which are stored in the file. If the "tracked only" option of `PReaction` is used, the unstable particles can not be read by the script.

- Dummy:

Any particle

```
"mypar = [*,num]"
```

- With variable:

```
"num=1; mypar = [id,$num]"
```

Similar to syntax in „echo“

Particle Loops

Replaces „current position“

```
"foreach(id); ... [id] ..."

"foreach(p); mom = [p]->P(); echo proton momentum $mom"
"foreach(*); id = [*]->ID(); echo found particle with $id"

"loop: ..."
"...[id]..."
"formore(id); goto loop;"
```

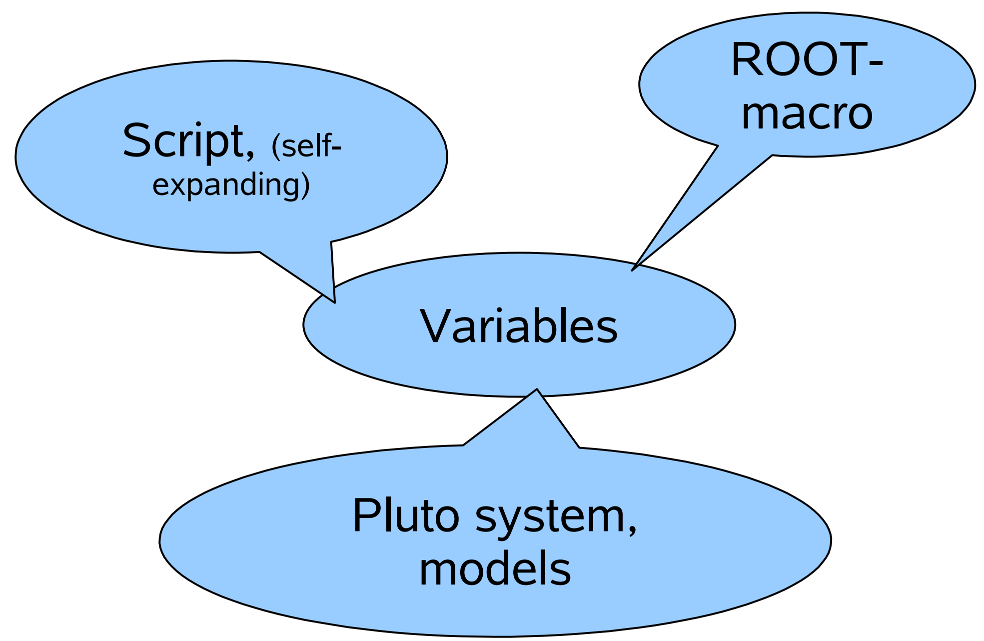
System values

- Idea: Move all system variables to batch variable space
  - Readable by script (conditions on versions)
  - Modifications by script (filters, smearings)

Change Pluto system variables only when you know what you are doing

- Examples:

```
_system_version
_system_weight_version
_system_unstable_width
_event_vertex_x / ..._y / ..._z
```



Data base values

(read-only)

- Reading of the (particle) data base values

```
"... = id.varname;"
```

- Mass:

```
"mass = eta.mass; echo $mass;"
"[rho0]->SetM(rho0.mass);"
```

- Pid:

```
"mypar->SetID(eta.pid);"
```

- N(pid) in current event:

```
"cur = 0; myloop: if !(cur ~ p.npar); cur = cur + 1;
    echo proton $cur; [p,$cur]->Print(); goto myloop"
```

PUtils

The script language offers to call all methods which are available in the `PUtilsREngine` class (which is a wrapper to `PUtils`). This can be used, e.g., to access the random number generator:

```
"var = sampleFlat();"
"echo The number is $var;"

"var = sampleGaus(10.,0.1);"

"mean = 10; sigma=0.1;"
"var = sampleGaus(mean,sigma);"
```

Projecting

```
r->Do(histo,"command");
```

- _x, _y, _z, _w as input for histogram filling:

```
//Missing mass of the p2 and pi0 pair:
r->Do(histo1,"miss= [p + p]- ( [p,2]+ [pi0] );_x=miss->M()");

//Theta of the first proton
r->Do(histo3,"_x= ([p,1]->Theta() * 180.)/TMath::Pi()");
```

NTuple in- and output

Syntax:

```
r->Output(ntuple,"var1 = ...; var2 = ...; ...");
```

where `var1, var2, ...` are the variables of the ntuple.

Example(s):

```
TNtuple *ntuple = new TNtuple("ntuple","eta events",
                      "eta_px:eta_py:eta_pz:eta_m");
PReaction r("3.5","p","p","p p eta");
r.Output(ntuple,"eta_px = [eta]->Px(); eta_py = [eta]->Py();
                 eta_pz = [eta]->Pz(); eta_m  = [eta]->M()");
```

Syntax:

```
r->Input(ntuple);
r->Do("... = var1; ... = var2; ....");
```

```
PReaction r;
r.Input(ntuple);
r.Do("myeta = P3M(eta_px,eta_py,eta_pz,eta.mass)");
r.Do("cm = P3E(0.000000,0.000000,4.337961,5.376545) ;
      myeta->Boost(cm);");
r.Do(histo,"_x= myeta->CosTheta();");
r.Loop()
```

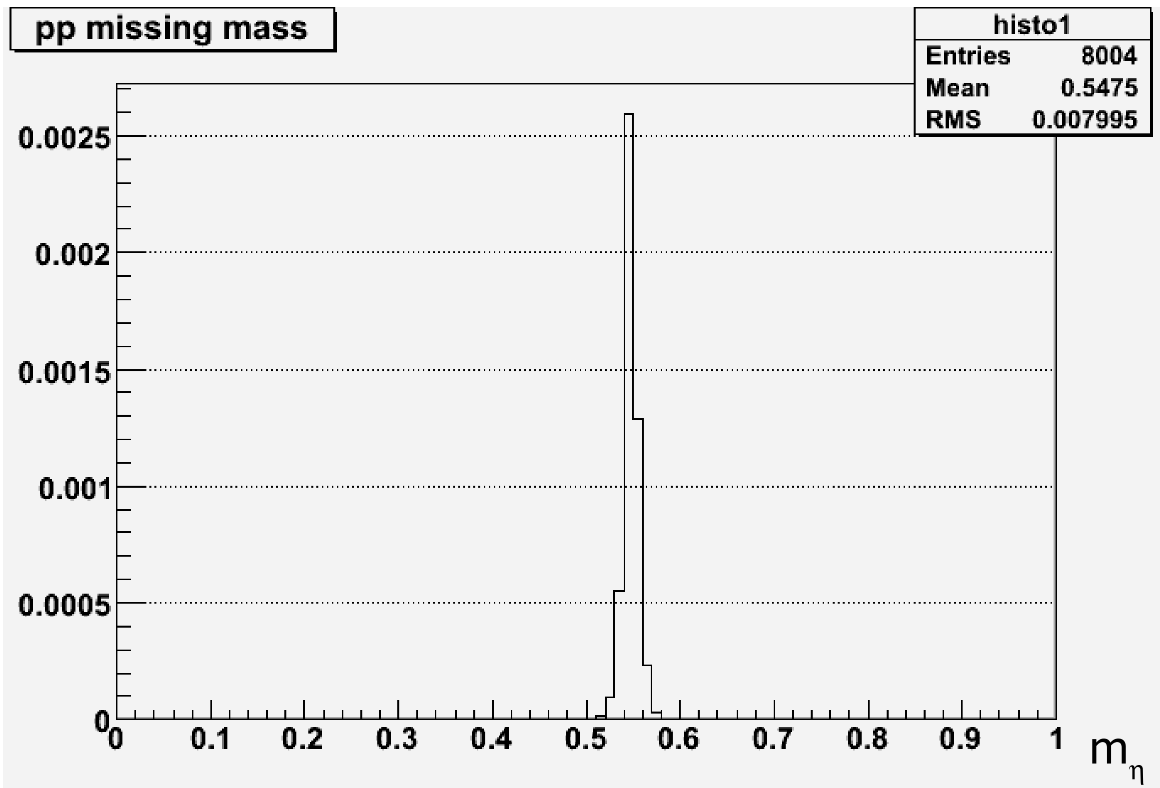
```
makeDistributionManager()->Unpack("pluto_demo_filter.root");
```

```
root use_demo_pfilter.C
<PBatch> **** This is our demo filter, v1
<PBatch> ********************************************************************************
<PBatch> Usage:      This filter works in 2 modi:
<PBatch> Usage:      1.) inclusive:
<PBatch> Usage:          default
<PBatch> Usage:      2.) exclusive measurement:
<PBatch> Usage:          Add in your macro after filter attachment:
<PBatch> Usage:          makeDistributionManager()->Startup("_filter_exclusive=1")
<PBatch> ********************************************************************************
<PBatch> Usage:      Momentum smearing is applied, you can change:
<PBatch> Usage:          makeDistributionManager()->Startup("_filter_smear_factor=....")
<PBatch> ********************************************************************************
<PBatch> Debug mode:     makeDistributionManager()->Startup("_filter_debug=1")
<PBatch> ********************************************************************************
Info in <PDistributionManager::Unpack>: Recovered TH3 <eff_elec>
Info in <PDistributionManager::Unpack>: Recovered TH3 <eff_elec>
Info in <PDistributionManager::Unpack>: Recovered TH3 <eff_protons>
...............
```

Debug Mode

```
<PBatch> **** New event called
<PBatch> e+: 69.8261, 0.715592, 245.617 eff: 0.555556 <removed>
<PBatch> e-: 137.257, -0.253324, 280.738 eff: 0 <removed>
<PBatch> p: 595.693, 0.955668, 353.261 eff: 0.88
<PBatch> p: 1580.33, 0.96885, 88.5876 eff: 0.898054
<PBatch> **** New event called
<PBatch> e+: 260.844, 0.987175, 205.963 eff: 0.347826 <removed>
<PBatch> e-: 510.485, 0.985685, 191.594 eff: 0.369565 <removed>
<PBatch> p: 514.151, 0.877485, 245.36 eff: 0.878049
<PBatch> p: 1916.66, 0.981379, 48.3063 eff: 0.901349 <removed>
<PBatch> **** New event called
<PBatch> e+: 566.346, 0.903716, 124.156 eff: 0.230769 <removed>
<PBatch> e-: 176.562, 0.901836, 125.033 eff: 0.45 <removed>
<PBatch> p: 1213.43, 0.921156, 321.4 eff: 0.926829
<PBatch> p: 1175.81, 0.994925, 29.614 eff: 0.891892
<PBatch> **** New event called
<PBatch> e+: 194.403, 0.715122, 112.443 eff: 0.785714
<PBatch> e-: 135.961, 0.36051, 148.823 eff: 0.85
<PBatch> p: 1630.92, 0.984752, 45.5283 eff: 0.901349
<PBatch> p: 1292.88, 0.938574, 228.09 eff: 0.890909
<PBatch> **** New event called
<PBatch> e+: 607.752, 0.853266, 239.26 eff: 0.954545
<PBatch> e-: 29.9976, 0.78329, 243.631 eff: 0 <removed>
<PBatch> p: 1540.69, 0.964513, 354.837 eff: 0.916805
<PBatch> p: 546.474, 0.989674, 84.4436 eff: 0.903846
```

Result of the Filter



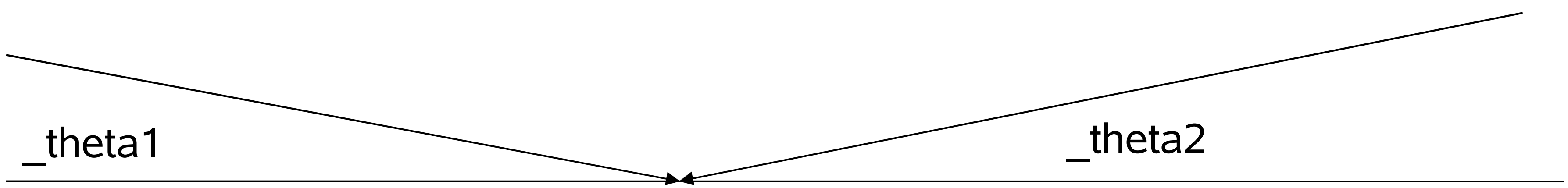
Something more applications for scripting

Beam parameters

```
PReaction my_reaction("_T1=0.4; _T2=0.4; ","p","p","p p");

PReaction my_reaction("_T1=0.4; _T2=0.4; _theta1=90*TMath::DegToRad();
                                          _theta2=-90*TMath::DegToRad();","p","p","p p");
```

- Keywords: _T1, _T2, (or _P1, _P2), _theta1, _theta2, _phi
- Same syntax as script



Very flexible models

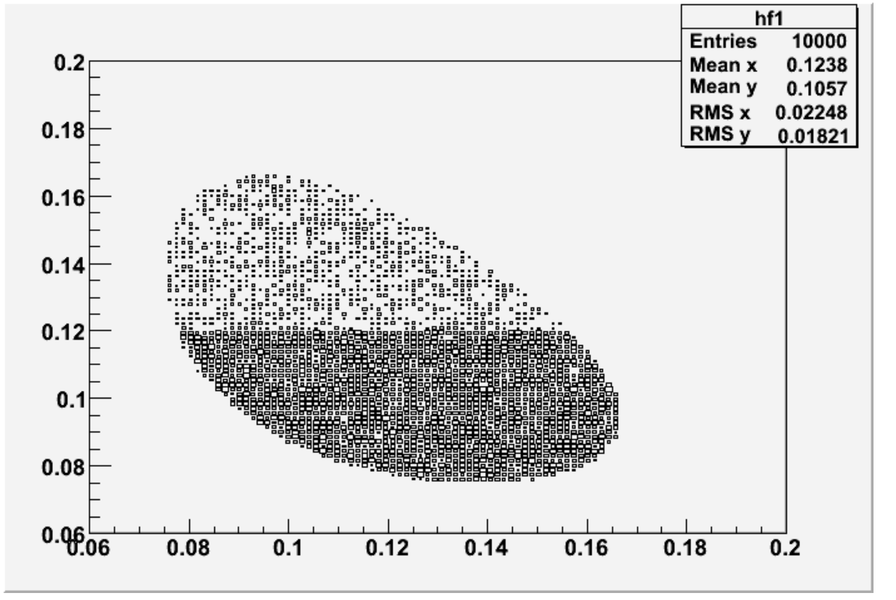
- The VMD function in the class PSimpleVMD can be replaced:

```
AddEquation("_ff2 = 0.17918225/( (0.4225- _q2)*(0.4225- _q2) + 0.000676)");
```

  - „Landsberg formfactor“

one can use a self-defined equation using the
PBatch syntax
in this case the batch variables "_q" and "_q2"
are the dilepton mass (resp. squared)
The equation has to calculate _ff2, the form factor
squared.

3-body correlation




```
makeDistributionManager()->Disable
          ("eta_hadronic_decay");


PDalitzDistribution* decay =
    new PDalitzDistribution("my_hadronic_decay",
                            "Eta matrix element for decay into charged pions");
decay->Add("eta,    parent");
decay->Add("pi0,    daughter,    primary");
decay->Add("pi+,    daughter,    s1");
decay->Add("pi-,    daughter,    s2");

//A "step function"
decay->AddEquation("_f = 1.; m = (_s1 + _primary)->M2(); if m > 0.12; _f = 0.2");
decay->SetMax(1);

makeDistributionManager()->Add(decay);

PReaction my_reaction("2.2","p","p","p p eta [pi+ pi- pi0]");

TH2F *hf1= new TH2F("hf1","",100,0.06,.2,100,0.06,.2);
my_reaction.Do(hf1,"_x = ([pi-] + [pi0])->M2() ; _y = ([pi+] + [pi0])->M2()");
my_reaction.Print();
my_reaction.Loop(10000);
hf1->Draw("box");
```

Summary Part I

- Script:
  - Based on TFormula, but:
    - Access to data base values
    - Can handle PParticle & all methods
    - Access to event loop
    - Flow control (goto, loops)
  - Filters, projections
  - Makes future model extensions very simple
- Manual available

The „famous“ example

All particles

```
PReaction my_reaction("2.2","p","p",
       "p p eta [dilepton [e+ e-] g]", "eta_dalitz",1,0,0,0);
```

Distributions

Models

```
Reaction Channels:
   1. p + p --> p + p + eta
      Interaction model(s):
      [pp_eta_prod_angle] Eta polar angles in pp reactions for direct production
      [pp_eta_pp_align] pp alignment in pp reactions for direct eta production
      [p + p_fixed_p_p_eta] Fixed product masses {/}
      [p + p_genbod_p_p_eta] Pluto build-in genbod {/genbod}
   2. eta --> dilepton + photon (Dalitz)
      Interaction model(s):
      [eta_dalitz] Dalitz decay {/}
      [eta_genbod_g_dilepton] Pluto build-in genbod {/genbod}
   3. dilepton --> e+ + e-
      Interaction model(s):
      [dilepton_fixed_e-_e+] Fixed product masses {/}
      [dilepton_genbod_e-_e+] Pluto build-in genbod {/genbod}
      [eta_dilepton_helicity] Helicity angle of the dilepton decay of eta
 Output Files:
   Root : eta_dalitz.root, all particles on file.
```

Models are mandatory (at least one per decay), distributions are optional

Distribution Methods

`Bool_t Init(void);` --> Sets pointer to PParticle

`Bool_t Prepare(void);` --> First step

`Bool_t SampleMass(void);` --> Required once per decay

`Bool_t SampleMomentum(void);` --> Required once per decay

`Bool_t SampleAngle(void);`

`Bool_t IsValid(void);` --> Rejection method

`Bool_t CheckAbort(void);` --> Abort event and re-sample it

`Bool_t Finalize(void);` --> Return kFALSE if still not ready (see below)

`Bool_t EndOfChain(void);` --> After full event sampling in PReaction
if !Finalized EndOfChain is called
if kFALSE, the event will be re-done

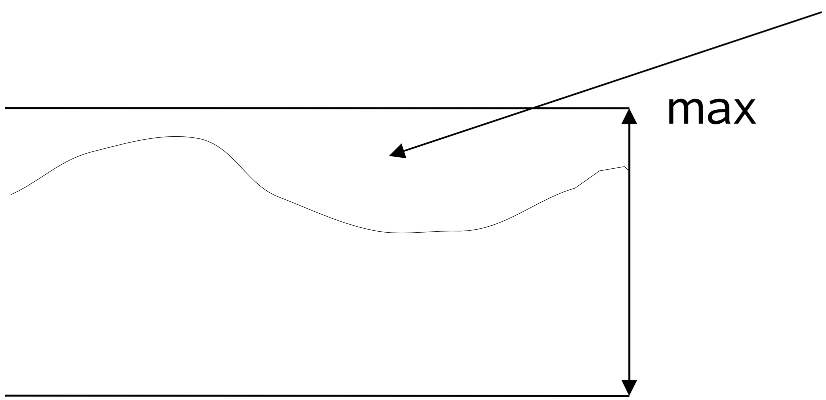
`Double_t GetWeight(void);` --> weight, alternative to sampling

Rejection methods
(IsValid, EndOfChain)

- Works in the case of small modifications to p.s.

„lost“ events

Random number

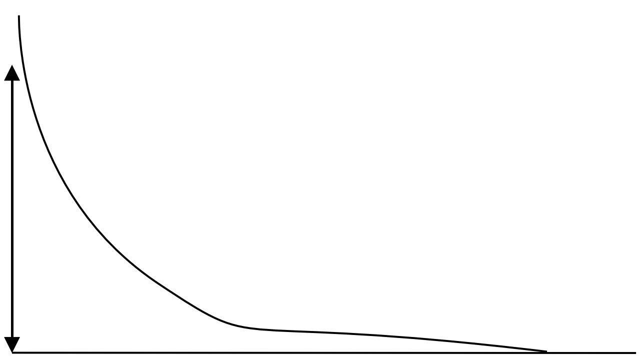


max

Example(s):
- Angular distributions
- 3body hadronic decays

- Does not work with singularities/strong deviations

Random number ?



Sampling
- Dalitz decays („dilepton“)

Models

- Inherited from PDistribution (and TF1)
- Standalone-methods (PParticles not required)
- Link to data base (i.e. they require a valid decay key)
- Therefore they can be accessed by „the Pluto world“
  - Connect the „PParticle“ world with data base (=function)
- Eval() wrapped to GetWeight()
  - Draw + this->GetRandom()

Models: Standard Constructor

PDistribution name

parent „anything“ daughters

```
PEtaDoubleDalitz("eta_double_dalitz_simple@eta_to_dilepton_dilepton",
                 "Dilepton generator for eta -> dilepton + dilepton",
                  -1);
```

Description (PReaction print)

- C'tor parsed and data base key attached
- Only one primary model (“/”) allowed
  - This is by default the mass sampling PDistribution

PEtaDoubleDalitz

```
Bool_t PEtaDoubleDalitz::Init(void) {

    parent = GetParticle("parent");
    if (!parent) {
     Warning("Init","Parent not found");
     return kFALSE;
    }

    dil1 = GetParticle("dilepton");
    dil2 = GetParticle("dilepton");

    if (!dil1 || !dil2) {
     Warning("Init","Dileptons not found");
     return kFALSE;
    }
```

Called once during setup phase

```
Bool_t PEtaDoubleDalitz::SampleMass(void) {

    Double_t MEta = parent->M();

    Double_t weight = 1.;
    Double_t mVV[2];
    do {
     mVV[0]=Gen2lepton1(MEta);//IM of the 1-st pair e+e-
     mVV[1]=Gen2lepton1(MEta);//IM of the 2-nd pair e+e-
        ......

    } while((mVV[0]+mVV[1]>MEta) || .......  );


    dil1->SetM(mVV[0]);
    dil2->SetM(mVV[1]);

    return kTRUE;
};
```

Called in event loop

Secondary Models

- In the macro:

```
ff = new PSimpleVMDFF("vmd_ff_dd@eta_to_dilepton_dilepton/formfactor",
                      "VMD form factor",-1);
ff->SetVectorMesonMass(0.77);
ff->SetWeightMax(5.);

makeDistributionManager()->Add(ff);
```

- in PEtaDoubleDalitz:

```
Bool_t PEtaDoubleDalitz::Init(void) {
  .......
formfactor_model=
    GetSecondaryModel("formfactor");
if (formfactor_model)
    ff_w_max = formfactor_model->GetWeightMax();
if (!formfactor_model) ff_w_max = 1.;

return kTRUE;
}
```

VMD-model

```
Double_t PSimpleVMDFF::GetWeight(void) {

    Double_t q     = dilepton->M();
    Double_t pmass = parent->M();

    return GetWeight(q,pmass);
};

Double_t PSimpleVMDFF::GetWeight(Double_t *mass, Int_t * ) {

    Double_t q2=mass[0]*mass[0];

    Double_t ff= vector_meson_mass2/(vector_meson_mass2 - q2);
    return ff*ff;

}
```

- Can be replaced by a customized class
  - or PBatch syntax, as described above

Calling a Secondary Model

(we are back in PEtaDoubleDalitz)

```
do {
    mVV[0]=Gen2lepton1(MEta);//IM of the 1-st pair e+e-
    mVV[1]=Gen2lepton1(MEta);//IM of the 2-nd pair e+e-

    if (formfactor_model
          && formfactor_model->GetVersionFlag(VERSION_SAMPLING) ) {

        weight *= formfactor_model->GetWeight(mVV[0]);
        weight *= formfactor_model->GetWeight(mVV[1]);

        }

    } while((mVV[0]+mVV[1]>MEta) ||
           ((weight/ff_w_max) < PUtils::sampleFlat())  );
```

Macro Output

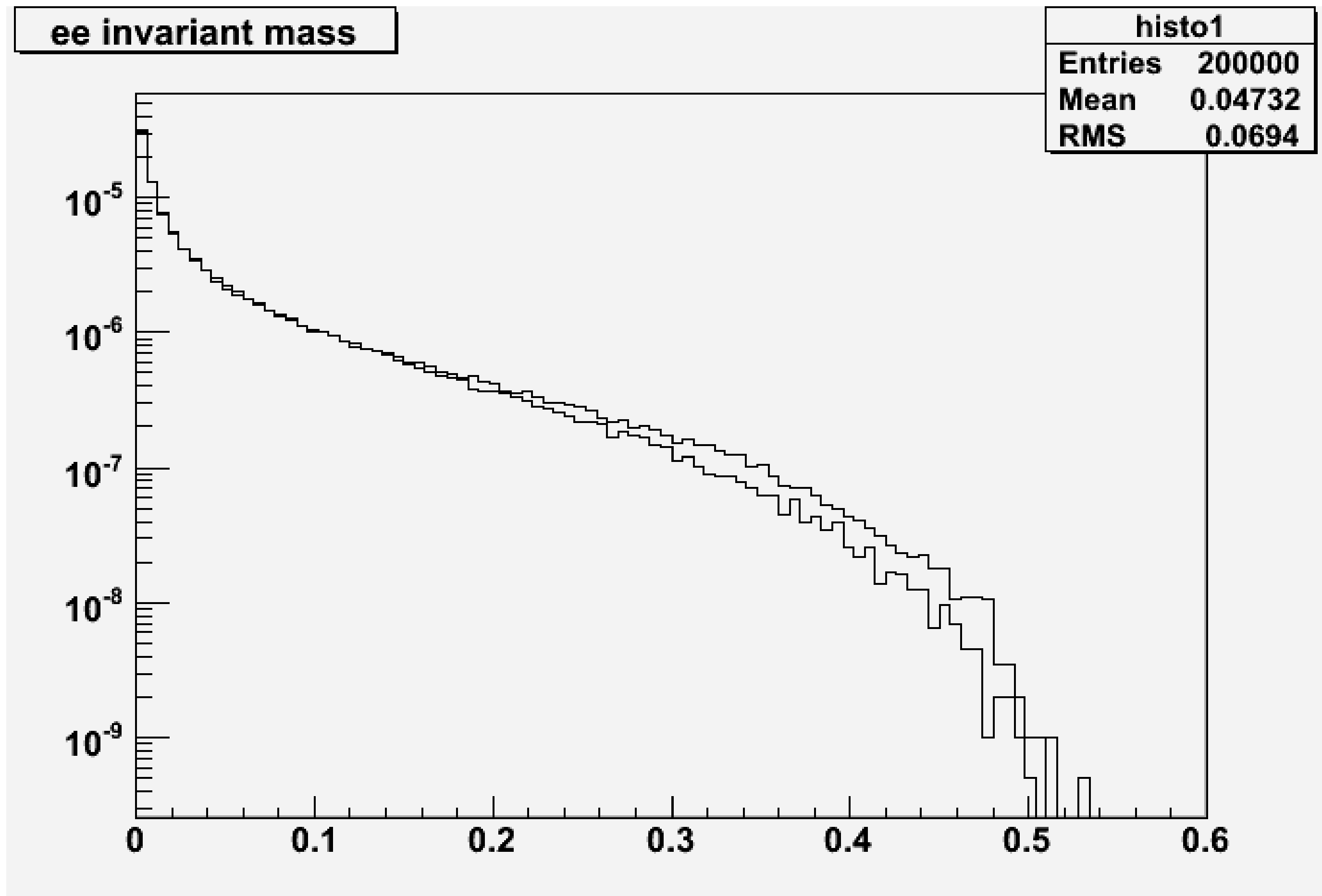
```
> makeDistributionManager()->Print("rare_eta_decays");
--------------------
PDistributionManager
--------------------
    user                                 User defined distributions : 1 objects (of 1)
[X] vmd_ff_dd                            VMD form factor
    root                                 Root group : 499 objects (of 507), subgroups: 8
      eta_physics                        Physics about eta production, and decay :
                                             11 objects (of 13), subgroups: 1
        rare_eta_decays                  Rare eta decays : 5 objects (of 6)
[X]     eta_double_dalitz_simple         Dilepton generator for eta -> dilepton + dilepton
[ ]     eta_double_dalitz_complex        Full dilepton generator for eta -> e+ e- e+ e-

.......



> my_reaction.Print();
.........
     2. eta -> dilepton + dilepton
        Interaction model(s):
        [eta_double_dalitz_simple] Dilepton generator for eta -> dilepton + dilepton {/}
        [vmd_ff_dd] VMD form factor {/formfactor}
        [eta_genbod_dilepton_dilepton] Pluto build-in genbod {/genbod}
     3. dilepton --> e+ + e-
        Interaction model(s):
        [dilepton_fixed_e-_e+] Fixed product masses {/}
        [dilepton_genbod_e-_e+] Pluto build-in genbod {/genbod}
```

Results for $\eta \rightarrow \gamma^* \ \gamma^*$



Complex Version

- „Afterburner“ using EndOfChain + rejection
  - Angular distributions calculated by T. Petri and A. Wirzba, Jülich
  - (all higher-order angles taken into account)

```
2. eta -> dilepton + dilepton
   Interaction model(s):
   [eta_double_dalitz_simple] Dilepton generator for eta -> dilepton + dilepton {/}
   [eta_double_dalitz_complex] Full dilepton generator for eta -> e+ e- e+ e- {/full}
   [vmd_ff_dd] VMD form factor {/formfactor}
   [eta_genbod_dilepton_dilepton] Pluto build-in genbod {/genbod}
```

```
17638 aborted events were repeated, error codes:
10=5964 82=11674 <-- ca 10%
```

Auto-executing plugins

```
makeDistributionManager()->Print("plugins");

    plugins                      Plugins : 4 objects (of 10)
[ ] nucleus_fermi                Fermi motion for some targets
[ ] pion_beam                    Plugin for physics with pion beams
[ ] brems                        Plugin for Bremsstrahlung models
[ ] dalitz_mod                   Plugin to modify Dalitz decays
[ ] elementary                   Plugin for elementary collisions/
                                           total cross section
[X] eta_decays                   Plugin for (rare) eta decays
( ) low_energy_pp_elastic        Low energy scattering <scatter_mod>
 (X)pp_elastic                   PP elastic scattering with SAID
[X] pn_elastic                   p+n scattering <scatter_mod>
[X] strangeness                  Strangeness extension
[X] w_to_pi+_pi-_pi0_matrix      Omega to 3pi Distribution
```

- This „new physics“ is enabled by default

Summary Part II

- Pluto: ready to include (also sophisticated) models
- Models can use „secondary“ sub-models (form factors, matrix elements, cross sections, ...)
- Sub-models can be replaced
- Rare η decay plugin is ready to be used