Introduction to PLUTO

PLUTO: Package for generation of particle emission.

Scope:

Source : free particle , 2-particle collision , particle decay , fireball from heavy-ion collision

Particles : hadrons, leptons, photon
stable and unstable (including: resonances with broad mass distribution)

Properties of particles and decay channels are taken from **Review of Particle Physics**.

Decays : into particles or through the virtual photon
According to the relativistic kinematics + possible simple physical models

Emission : one can trim down the detector acceptance by filtering out momenta, angles.

Possibility to encode own particle / decay / model.

Form: Library linked by ROOT session. Dedicated objects usable interactively or in macro, e.g.
`PParticle, PChannel, PReaction, ...`

Compatibility: PLUTO 5 ↔ ROOT 5, PLUTO 6 ↔ ROOT 6

Algorithm: Define the emission source + emitted particle(s) + activate specific decays
Account for beam energy (Lorentz transform CM → Lab)
Generate events in loop. Each step: pull angles (momenta) from some distributions
+ execute decays

Result: Set of events with particles and their momenta.

Output format: By default, `TTree` object in root file. Optionally: text , `TNtuple`, `THnF` .

krzysztof.piasecki@fuw.edu.pl

Modes of work:

- Building "from bricks", i.e objects are: sources, particles, decay channels and reaction
- Dedicated scripting language

www :

- https://hades.gsi.de/node/31 : Home page, GDPR problem → most of content is private
- https://plutouser.github.io/v6.01/ : Sources, class documentation, exemplary macros

Papers:

- I. Fröhlich et al., arXiv:0708.2382v2 "Pluto: A Monte Carlo Simulation Tool for Hadronic Physics"
- I. Fröhlich et al., arXiv:0905.2568v1 "Design of the Pluto Event Generator"

Talk:

- I. Fröhlich, "The Pluto++ Event Generator" (ACAT 07 Amsterdam)

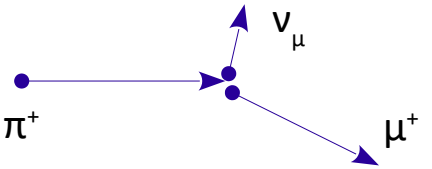
Installation

- Sources:
  ```
  wget https://plutouser.github.io/v6.02/pluto_v6.02.tar.gz
  ```
  or:
  ```
  git clone https://github.com/PlutoUser/pluto6.git
  ```
- Installation steps:
  ```
  cd pluto6 ; mkdir builddir ; cd builddir
  cmake ..
  make
  ```
- Result: Library `libPluto.so`
  I suggest: `export PLUTOLIBDIR={.....}/builddir`
- Activation of library:
  - ▷ in Root session: `gSystem->Load ("$PLUTOLIBDIR/libPluto.so");`
  - ▷ in Root macro: `R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)`
- NPD training computer:
  ```
  cp -p ~kpiasecki/soft/pluto/pluto_setup.sh .
  . ./pluto_setup.sh
  ```
  ↖ **each time you open a terminal**

Example 1

Decay $\pi^+ \longrightarrow \mu^+ \nu_\mu$

(E_{kin} = 1 GeV)



Preprocessor macro
to link `libPluto.so` library

We create 1 x π^+ .
Pluto identifies hadron by name.
We give E_{Kin} = 1 GeV.

We form the reaction:

① Substrate: π^+
② Products : μ^+, ν_μ
Format: "Descriptor string"
③ Name for output file

We execute the loop of
10000 events.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu ()
{
  PParticle pip ("pi+", 1.0 );

  PReaction* my_reaction = new PReaction
      ( &pip ,
        "mu+ nu",
        "pip_mupnu", 1);

  my_reaction->Print ();
  my_reaction->Loop (10000, 1);

  return 0;
}
```

- Let's launch the simulation
- and read the output file
in the root format

Sometimes you may get this:
You then need to link the library

```
$ nice root -b -q pip_mupnu.C

$ root -l pip_mupnu.root
{...}
Warning in <TClass::Init>: no dictionary {...}
Root [1] gSystem->Load ("$PLUTOLIBDIR/libPluto.so")
(int) 0
```

```
$ TTree* t = (TTree*) _file0->Get ("data")
$ t->Scan ("pid:GetParentId()")
$ t->Draw ("M()", "pid==8")
$ t->Draw ("E()-M()", "pid==8")
           "Rapidity()"

$ t->Draw ("Rapidity()", "pid==5")
           "Pt():Rapidity()", "pid==5")

$ t->Draw ("P():Theta()", "  Name() == \"mu+\"  ")

$ t->Draw ("Rapidity():Pt()", "pid==5")
            Beta()
            Pz()  Px()  Py()
```

{PID → see p. 6}

1 entry = collection of particles from 1 reaction (TClonesArray)

Kinematics calculated in Lab

PParticle : Class representing a particle with given 4-momentum and other physical properties. It inherits from TLorentzVector.

```
$ PParticle p ("pi0")
$ p.Print ()
$ cout << p.Name() << "\t" << p.ID() << endl;

$ listParticle ("pi0")   or   ( 7 )

$ p.Set... (...)
    SetE, Px, Py, Pz,
    SetTheta (...), SetPhi(...)

$ p.E() , Px Py Pz Pt Mt Theta Phi

$ p.Boost ( βx , βy, βz )
$ p.BoostVector() .Print()
```

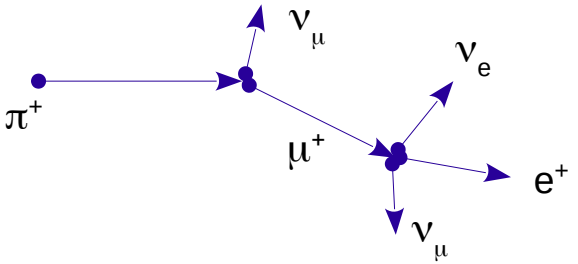
rotate the momentum vector

Apply Lorentz Transform by velocity vector β

Retrieve the velocity vector in terms of β

Example 2

Decay $\pi^+ \rightarrow \mu^+ \nu_\mu$, $\mu^+ \rightarrow \nu_\mu e^+ \nu_e$ (E_{kin} = 1 GeV)



- μ^+ is an unstable particle ($\tau \approx 10^{-6}$ s) and decays into $\nu_\mu e^+ \nu_e$. Let's activate this decay.

Descriptor string

```
"nu mu+ [e+ nu nu]"
```

(We open brackets after the mother particle and place the products therein)

Caution: *Pluto does not differentiate* ν.

Output file name

Flags

f0: Save (1–all particles / 0–only final ones)
f1: (unused)
f2: Generate production vertex (1/0 = yes/no)
f3: Generate text output file (1/0 = yes/no)

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_epnunu ()
{
  PParticle pip ("pi+", 1.0 );

  PReaction* my_reaction = new PReaction (
                    &pip ,
                    "nu mu+ [e+ nu nu]",
                    "pip_mupnu_epnunu" ,
                    1, 0, 0, 1
                  );
    my_reaction->Print ();

    my_reaction->Loop (10000, 1);
    return 0;
}
```

```
$ TTree* t = (TTree*) _file0->Get ("data")
$ t->Scan ("Name() : pid : GetParentId() : GetSiblingIndex()")
$ t->Draw ("Theta()", "pid==8")
                        pid==5
                        pid==4 && GetParentId()==8
                        pid==4 && GetParentId()==5
```

❖ Access to data on particles and decays

► Particle types included in Pluto :

```
$ listParticle ()
```
← Prints out particles contained in Pluto's database

- **Particles which are stable or have mass width Γ < 1 MeV**

PID	Symbol	Type	M[MeV]	PID	Symbol	Type	M[MeV]	PID	Symbol	Type	M[MeV]
1	g	γ	0	13	n	n	939.6	25	anti_n	$\overline{n}$	939.6
2	e+	e^+	0.511	14	p	p	938.3	26	anti_Lambda	$\overline{\Lambda}$	1115.7
3	e-	e^-	0.511	15	anti_p	$\overline{p}$	938.3	27	anti_Sigma-	$\overline{\Sigma}^-$	1189.4
4	nu	ν	0	16	K0S	K^0_s	497.7	28	anti_Sigma0	$\overline{\Sigma}^0$	1192.6
5	mu+	μ^+	105.7	17	eta	η	547.5	29	anti_Sigma+	$\overline{\Sigma}^+$	1197.4
6	mu-	μ^-	105.7	18	Lambda	Λ	1115.7	30	anti_Xi0	$\overline{\Xi}^0$	1314.9
7	pi0	π^0	135.0	19	Sigma+	Σ^+	1189.4	31	anti_Xi+	$\overline{\Xi}^+$	1321.3
8	pi+	π^+	139.6	20	Sigma0	Σ^0	1192.6	32	anti_Omega+	$\overline{\Omega}$	1672.5
9	pi-	π^-	139.6	21	Sigma-	Σ^-	1197.4	53	eta'	η'	957.7
10	K0L	K^0_L	497.7	22	Xi0	Ξ^0	1314.9	67	J/Psi	J/Ψ	3096.9
11	K+	K^+	493.7	23	Xi-	Ξ^-	1231.3	68	Psi'	Ψ'	3686.0
12	K-	K^-	493.7	24	Omega	Ω	1672.5				

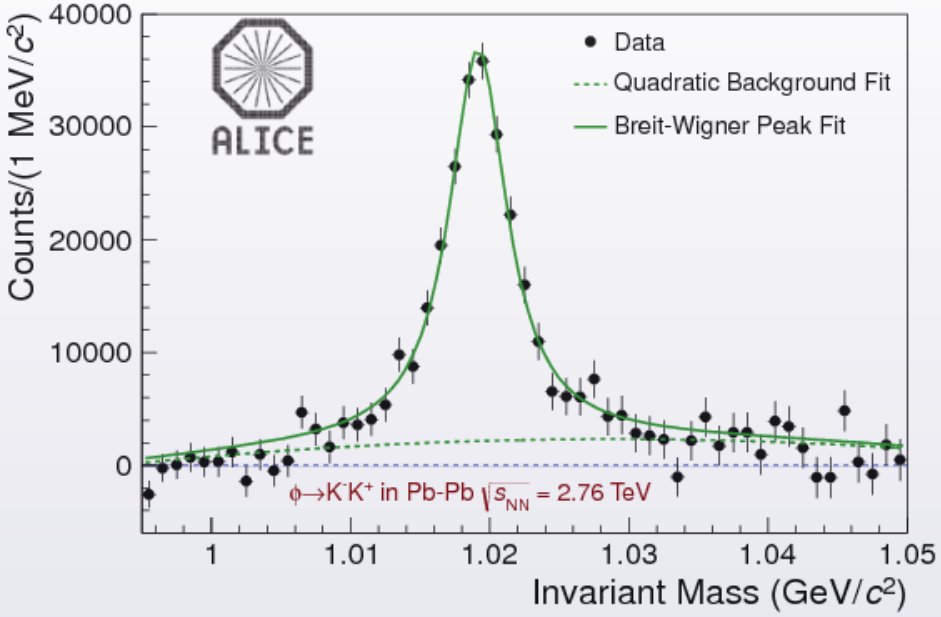
- **Nucleons and lightest nuclei:**

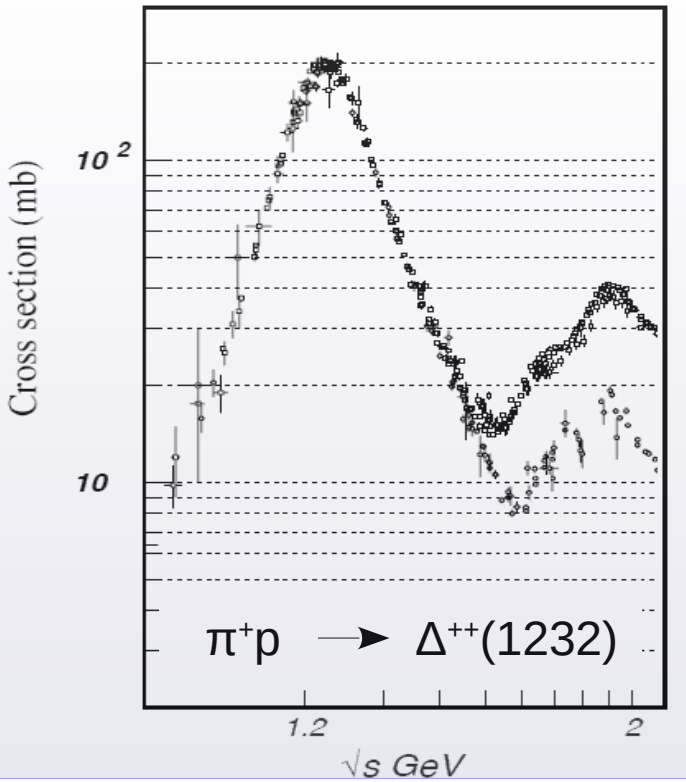
PID	Symbol	Type	M[MeV]	PID	Symbol	Type	M[MeV]
13	n	n	939.6	45	d	d	1875.6
14	p	p	938.3	46	t	t	2809.3
15	anti_p	$\overline{p}$	938.3	47	alpha	^{4}He	3727.4
25	anti_n	$\overline{n}$	939.6	49	He3	^{3}He	2809.2

- **Virtual photons:**

51	dilepton (→ e^+e^-)	50	dimuon (→ $\mu^+\mu^-$)

- **Resonances** (particles with broad mass distribution)
  Excited states of group of 2 or 3 valence quarks.





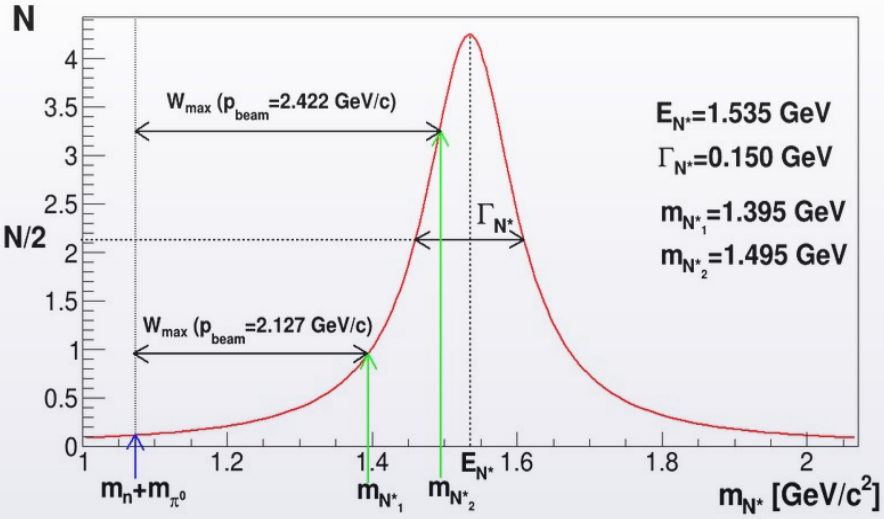
- **Model**: **relativistic Breit-Wigner** profile

$$g(m) = A\frac{m^2\Gamma^{\mathrm{tot}}(m)}{(M_{\mathrm{R}}^2 - m^2)^2 + m^2(\Gamma^{\mathrm{tot}}(m))^2}$$

$$\Gamma^{\mathrm{tot}}(m) = \sum_{\mathrm{k}}^{N}\Gamma^{\mathrm{k}}(m)$$

Γ^{tot} : total half-width,
sum over contributions to decay channels, Γ^{k}

Probing interval: $m \in [\, M_R - 2\Gamma\,,\, M_R + 12\,\Gamma\,]$



- **Resonances** in Pluto's database :

Mesons

PID	Symbol	Type	M[MeV]	Γ [MeV]
41	rho0	ρ^0	769.9	150.7
42	rho+	ρ^+	769.9	150.7
43	rho-	ρ^-	769.9	150.7
52	w	ω	781.9	8.43
54	sigma	σ	600.0	500.
55	phi	ϕ	1019.4	4.43

Baryons

PID	Symbol	Type	J[ħ]	M[MeV]	Γ[MeV]	PID	Symbol	Type	J[ħ]	M[MeV]	Γ[MeV]
34	D0	$\Delta(1232)^0$	3/2	1232	120	58	DP33+	$\Delta(1600)^+$	3/2	1600	350
35	D++	$\Delta(1232)^{++}$	3/2	1232	120	59	DP33-	$\Delta(1600)^-$	3/2	1600	350
36	D+	$\Delta(1232)^+$	3/2	1232	120	60	DS310	$\Delta(1620)^0$	1/2	1620	150
37	D-	$\Delta(1232)^-$	3/2	1232	120	61	DS31++	$\Delta(1620)^{++}$	1/2	1620	150
38	NP11+	$N(1440)^+$	1/2	1440	350	62	DS31+	$\Delta(1620)^+$	1/2	1620	150
39	ND13+	$N(1520)^-$	3/2	1520	120	63	DS31-	$\Delta(1620)^-$	1/2	1620	150
40	NS11+	$N(1535)^-$	1/2	1535	150	64	NP110	$N(1440)^0$	1/2	1440	350
56	DP330	$\Delta(1600)^0$	3/2	1600	350	65	ND130	$N(1520)^0$	3/2	1520	120
57	DP33++	$\Delta(1600)^{++}$	3/2	1600	350	66	NS110	$N(1535)^0$	1/2	1535	150

- Nature has many more resonances, in particular the ones with higher masses.
- One can add new particles to Pluto's data base ⇨ see page 24.

Access to data on particles and decays, cont.

```
$ listParticle (34)
```

Data on $\Delta(1232)^0$:
- centroid and Γ of mass distribution
- Decay channels + contributions (BR ≡ branching ratio)

- Direct access:

```
$ PStaticData* sd = makeStaticData() ;

$ sd->GetParticleMass (7)
      GetParticleID ("p")
      GetParticleName (7)
      GetParticleTotalWidth (34)
      GetParticleSpin (34)
      GetParticleParity (34)
      GetParticleIsospin (34)

$ sd->GetParticleNChannels (7)  ---> 2

$ sd->PrintParticle (7)  or ("pi0")

$ sd->PrintDecayByKey (143)

Database key=143
Database name=pi0 --> photon + photon
Decay index=3
Branching ratio=0.988000
Decay product 1->Database name=g
Decay product 2->Database name=g

$ sd->GetDecayBR (3)   ---> 0.988
```

"Manager" object containing the data base

```
(...)
Database key=143
Database name=pi0 --> photon + photon
Decay index=3
Branching ratio=0.988000
Decay product 1->Database name=g
Decay product 2->Database name=g

Database key=144
Database name=pi0 --> dilepton + photon
                                    (Dalitz)
Decay index=4
Branching ratio=0.012000
Decay product 1->Database name=g
Decay product 2->Database name=dilepton
```

Access to data on particles and decays, cont.

- Access to models of decays, distributions :

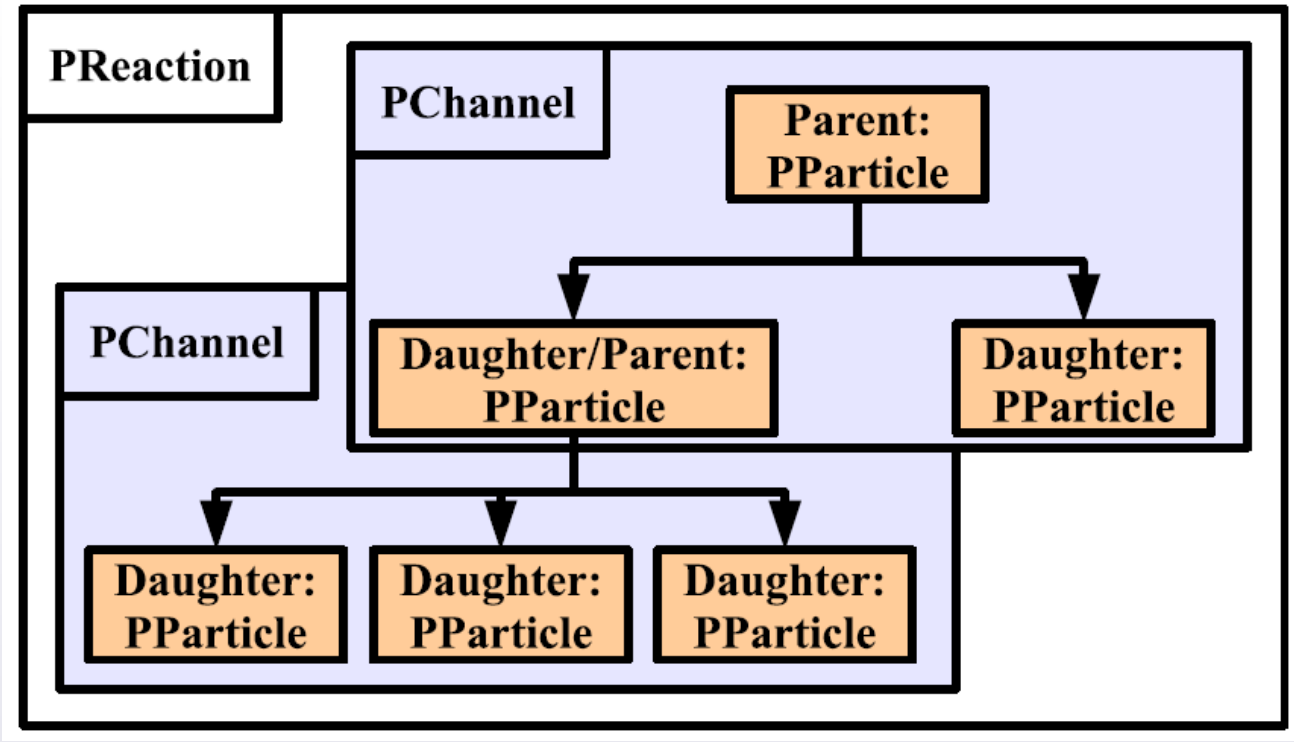
```
PDistributionManager* pdist = makeDistributionManager ()
pdist->Print ()
pdist->Print ("decay_models")
pdist->Print ("pi0_fixed_g_g")
```

nb. GenBod: cernlib.web.cern.ch/cernlib/mc/genbod.html

Example 3 Decay $\pi^+ \longrightarrow \mu^+ \nu_\mu$. This time, let's build the **reaction from bricks.**

(E_{kin} = 1 GeV)

Structural scheme:



- **Algorithm in our case:**

① Create 3 particles: π^+, μ^+ and ν , represented by the objects of `PParticle` class.

② Group these particles: create the array of addresses of PParticle objects (substrate first)

③ Create the $\pi^+ \rightarrow \mu^+ \nu$ decay channel, represented by the object of `PChannel` class.

④ Combine all the channels in the array of adresses of `PChannel` objects

(For us: 1 channel only ⇨ only 1 element in this array)

⑤ Define the reaction: create an object of `PReaction` class, giving the channel array and output file.

⑥ Execute the event loop.

- Let's encode the algorithm in the macro:

Substrate
Products

Array of particles

Decay channel

Array of all channels participating in the reactions

Define the reaction

Execute loop of 10000 events

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_long ()
{
    PParticle pip ("pi+", 1.0 );
    PParticle mup ("mu+") , nu ("nu");

    PParticle* parttable[] = { &pip , &mup , &nu };

    PChannel* pip_decay = new PChannel ( parttable , 2 );

                                // "2" = no. of decay particles

    PChannel* tab_of_chan[] = { pip_decay };

    PReaction* my_reaction = new PReaction
                ( tab_of_chan , "pip_mupnu_long",

                  1 ,           /* No of channels in table */
                  1 , 0 , 0 , 0 );

    my_reaction->Print ();
    my_reaction->Loop (10000, 1);

    return 0;
}
```

Example 4 Emission of particles from the two-particle collision.



Let's consider a reaction: p (E_{Kin} = 0.2797 GeV) + p ⟶ p + p + π^0

- Qualitatively it's new (two substrates). We'll use the dedicated constructor of `PReaction` class.

- Why do we use this "strange" value of 0.2797 GeV? It's a minimal E_{kin} that the beam nucleon should have to produce π^0 in NN collision (NN survives). ***"Threshold energy"***

  For substrates of masses m_1, m_2 and a product of mass m_X:

$$E_K^{Min} = \frac{m_X^2 + 2m_X(m_1 + m_2)}{2m_2}$$

  Verify that if $E_{Kin}(p^+)$ is lowered by 0.1 MeV, Pluto refuses to create π^0.

- However, the process does not end here: π^0 is unstable and after $\tau \approx 10^{-16}$ s it decays.
  Dominant decay channel: $\pi^0 \rightarrow \gamma\gamma$ (BR ≈ 99%)
  Let's activate it too.

Inside "" : E_{kin} of beam particle
Without "" : p of beam particle

Substrates (beam and target)

Reaction scheme (descriptor string)

Output file name

We run the loop

```
R__LOAD_LIBRARY (($PLUTOLIBDIR/libPluto.so)

int pp_pppi0_gg ()
{
    PReaction* my_reaction = new PReaction
          ("0.2797",
           "p", "p",
           "p p pi0 [g g]",
           "pp_pppi0_gg",
            1, 0, 0, 0 );

    my_reaction->Print ();
    my_reaction->Loop (10000, 1);
    return 0;
}
```

```
$ root -l pp_pppi0_gg.root
root [0] TTree* t = (TTree*) _file0->Get ("data")
root [1] t->Scan ("Name():pid:GetSiblingIndex()")

*        0 *         0 *            *      14014 *        -1 *
*        0 *         1 *            p *       14 *         2 *
*        0 *         2 *            p *       14 *         3 *
*        0 *         3 *         pi0 *        7 *         1 *
*        0 *         4 *            g *       1 *         5 *
*        0 *         5 *            g *       1 *         4 *
```

Unknown PID...?

- **Composite particle**

Substrates and products are bound by ***Available energy*** ($\sqrt{s} = \sqrt{(\sum E_i)^2 - (\sum \vec{p}_i)^2}$) ,
a Lorentz-invariant physical quantity.
In particular, in the CM frame, from its definition: $\sum \vec{p}_i = \vec{0} \quad \Rightarrow \quad \sqrt{s} = \sum E_i$
So it is the *amount of energy* available in CM.
From this resource, energy is spent to produce new particles. They can also move.

Composite Particle = "intermediate particle", for which: $\vec{p} = \sum \vec{p}_i \quad E = \sum E_i \quad M = \sqrt{(\sum E_i)^2 - (\sum \vec{p}_i)^2}$
From it, Pluto creates products, if the conservation rules allow for a decay.

$$PID_{CP} = 1000 \times PID_{2\ (Target)} + PID_{1\ (Beam)} .$$

```
$ t->Scan ("E() : P() : M()", " pid==14014 ")
*        0 *         0 * 2.1562446 * 0.7765961 * 2.0115390 *
*        1 *         0 * 2.1562446 * 0.7765961 * 2.0115390 *
```

As we see,
$E^2 - P^2 = M^2$

```
$ t->Draw ("Beta()", " pid==14014 {or 7} ")
$ t->Draw ("Rapidity()", " pid==7  {or 1} ")
$ t->Draw ("Pt() : Rapidity()", " pid==7  {or 1} ")
```

Distribution of β, *y for* pions concentrates around β, *y* of composite particle.

Example 5 Emission of particle from collision of two particles (construction "from bricks")
A *composite particle* is made of p + p , and this "particle" decays.

Substrates (beam + target)

Composite particle

Products

Array of participants of pp → ppπ⁰ channel

pp → ppπ⁰ channel

Number of products (3)

Array of participants in π⁰ → γγ decay

Array of all the channels

Definition of whole reaction

Number of channels (2)

Loop over events

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pp_pppi0_gg_long ()
{
  PParticle p1 ("p", 0.2797) , p2 ("p");
  PParticle comp = p1 + p2;
  PParticle p3 ("p"), p4 ("p"), pi0 ("pi0");

  PParticle* parttab_pi0[] =
             { &comp , &pi0 , &p3 , &p4 };

  PChannel* pi0_prod = new PChannel (
             parttab_pi0 ,
             3 );

  PParticle g1 ("g") , g2 ("g");
  PParticle* parttab_gg[] = {&pi0, &g1 , &g2};
  PChannel* pi0_ggdecay = new PChannel (
             parttab_gg, 2 );

  PChannel* tab_of_chan[] =
             {pi0_prod , pi0_ggdecay};

  PReaction* my_reaction = new PReaction
         ( tab_of_chan , "pp_pppi0_gg_long",
           2 ,
           1 , 0 , 0 , 0 );

  my_reaction->Print ();
  my_reaction->Loop (10000, 1);
  return 0;
}
```

pp → ppπ⁰

π⁰ → γγ

Complete reaction

Example 5 Analysis of output TTree within a macro

- Let's analyse the output tree from the simulation of $\pi^+ \rightarrow \mu^+ \nu_\mu$ decays.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int pip_mupnu_readtree ()
{
  TClonesArray* partArray = new TClonesArray ("PParticle", 10);
  PParticle* Part[10] ;
  TVector3 b ;

  TFile* fileIn = new TFile ("pip_mupnu.root") ;
  TTree* tree = (TTree*) fileIn->Get ("data") ;
  tree->SetBranchAddress ("Particles", &partArray ) ;

  Int_t nR = tree->GetEntries() ;
  cout << "\n * Analysing " << nR << " reactions.\n\n";

  for (Int_t iR = 0 ; iR < nR ; iR++ )                          <-- Loop over reactions
  {
    tree->GetEntry (iR);                                        <-- Reading i-th reaction
    for (Int_t iP = 0; iP < partArray->GetEntries() ; iP++ )    <-- Loop over particles
      Part[iP] = (PParticle*) partArray->At (iP);

    Part[1]-> Boost ( -Part[0]->BoostVector() );                <-- We transform the
    Part[2]-> Boost ( -Part[0]->BoostVector() );                    4-Momenta of products
                                                                    into the parent's frame
    for (Int_t iP = 1; iP <= 2; iP++)
      cout << Part[iP]->Name() <<'\t'<< Part[iP]->Px() <<'\t'   <-- Data
           << Part[iP]->Py()   <<'\t'<< Part[iP]->Pz() << endl;     printout

    cout << endl;
  }
  return 0;
}
```

Emission of particles from the collision zone of heavy ions

Subject : production of new particles
in collisions of heavy ions
at E_{kin} from 100 MeV/nucleon to LHC.

- It is found that the hadron multiplicities per nucleon are approximately governed by one quantity : **√s** .

- This finding is in line with the **statistical approach** to the collision zone ("Fireball") :

▻ A global thermodynamic equilibrium is assumed.

▻ The zone is filled with fermions and bosons.
Their distributions of energies (momenta)
should follow the quantum statistics (F-D or B-E).
PLUTO approximates them by ***Boltzmann distribution*** :

$$\frac{dN}{dE} \propto p\, E\, e^{-E/T}$$

- Boltzmann distribution:

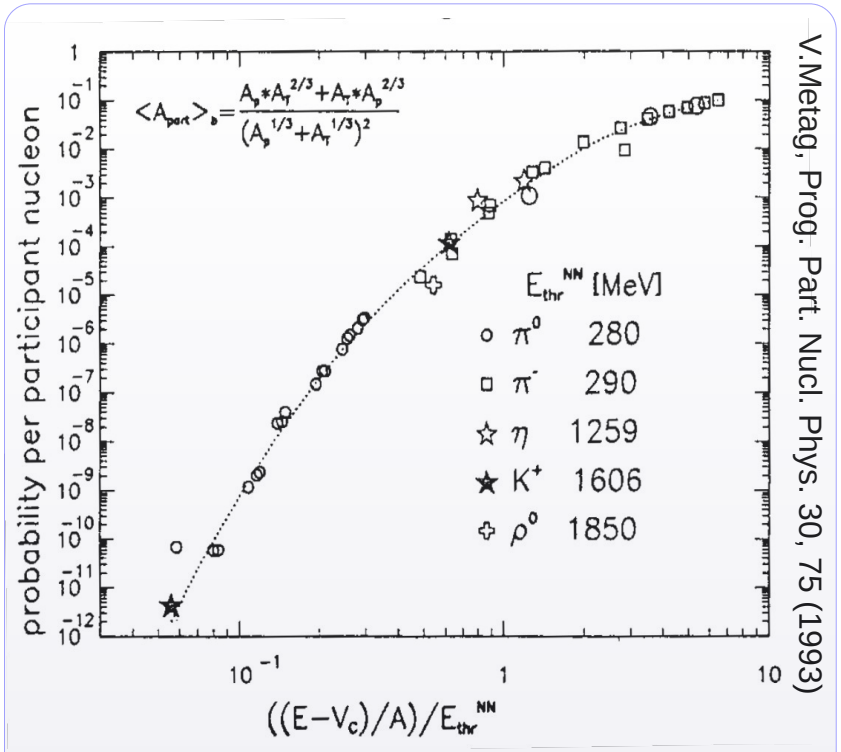
  ▻ is isotropic (equipartition of energy)
  ▻ $\langle E \rangle = {}^3/_2\, kT$ occurs (E is not a function of particle's mass).

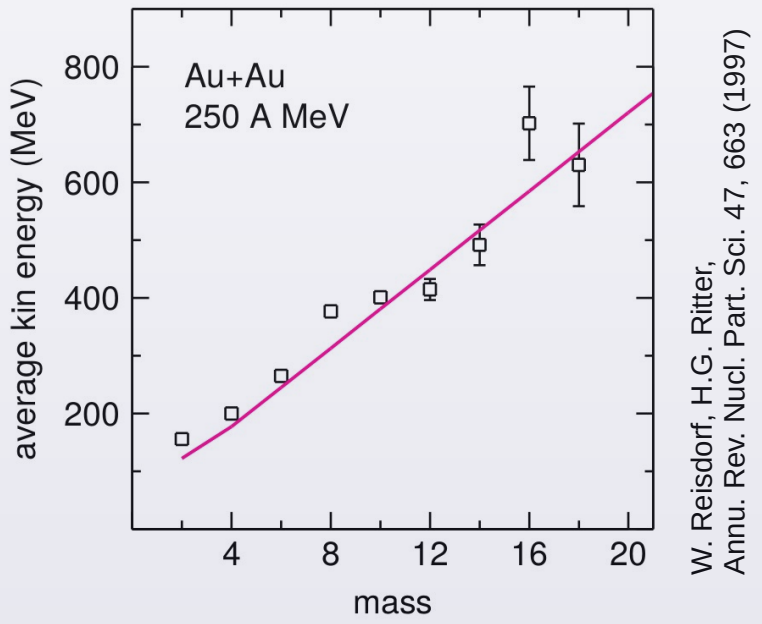
- Often the exp data contradict the above claims.

- Momentum distributions of some particles can be described only as superposition of 2 distributions (2 thermal sources)

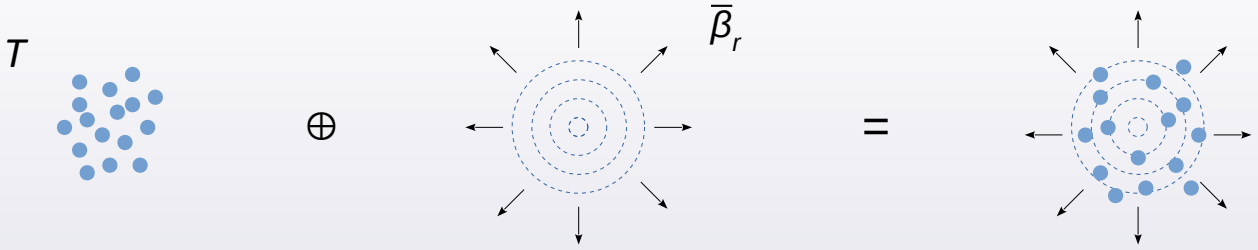
- Dependence $\langle E \rangle = f(mass)$ → prompts to postulate:

  in addition to a thermal motion, also
  a common **radial expansion** (collective motion).





Siemens-Rasmussen model: overlay of thermal motion (temperature *T*) on the radial expansion (velocity β).



Thermal motion (Boltzmann) | Collective motion (radial expansion) | Combined together

$$\left.\frac{d^3n}{dp^3}\right|_{CM} = \frac{N_i}{Z(T)} \cdot e^{-\frac{\gamma_r E}{T}} \cdot \left[\left(\gamma_r + \frac{T}{E}\right)\frac{sh\,\alpha}{\alpha} - \frac{T}{E}ch\,\alpha\right]$$

where: $\alpha = \gamma_r \beta_r p/T$

N_i, Z : normalization const.

P.J. Siemens, J.O. Rasmussen, Phys. Rev. Lett. 42, 880 (1979)

- Angular distributions (in ϑ and φ) are often not isotropic.

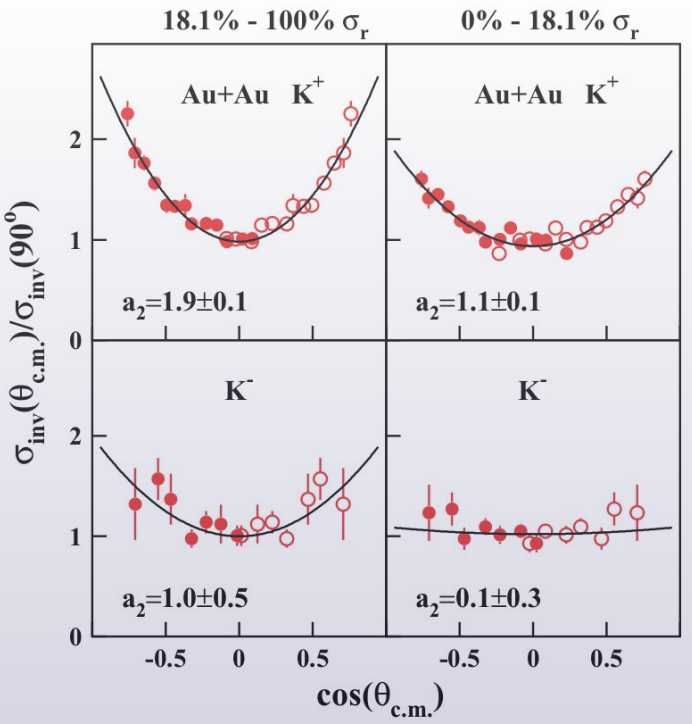
▷ Distribution in ϑ is phenomenologically parameterized by the Legendre polynomial, with weights of subsequent terms.

$$\frac{dN}{d\cos\theta_{NN}} \sim 1 + \sum_n a_n P_n(\cos\theta_{NN})$$

▷ Distribution in φ is phenomenologically parameterized by the Fourier series with weights of subsequent harmonics:

$$\frac{dN}{d\varphi} \sim \frac{1}{2\pi}\left(1 + 2\sum_{n\geq 1}^{\infty} v_n \cos n\varphi\right)$$

In Pluto we can set up v_n weights for a few first terms.



A. Förster et al., Phys. Rev. C 75, 024906 (2007)

Emission of particles from the collision zone of heavy ions

- Formula for the energy part of the distribution: sum of two Siemens-Rasmussen functions.

$$\frac{dN}{dE} \propto p\,E \left\{ f e^{-\gamma_r \frac{E}{T_1}} \left[\left(\gamma_r + \frac{T_1}{E} \right) \frac{\sinh \alpha_1}{\alpha_1} - \frac{T_1}{E} \cosh \alpha_1 \right] + (1-f) e^{-\gamma_r \frac{E}{T_2}} \left[\left(\gamma_r + \frac{T_2}{E} \right) \frac{\sinh \alpha_2}{\alpha_2} - \frac{T_2}{E} \cosh \alpha_2 \right] \right\}$$

Notice: ① for $\beta_r \to 0$ Siemens-Rasmussen function → Boltzmann function.
② for $f = 1$ only the 1st source is assumed.

- Forms of ϑ and φ distrib. : $\frac{dN}{d\cos\theta_{CM}} \sim 1 + a_2 \cos^2\theta_{CM} + a_4 \cos^4\theta_{CM}$ $\frac{dN}{d\varphi_{RP}} \sim 1 + 2\left[v_1 \cos\varphi_{RP} + v_2 \cos(2\varphi_{RP}) \right]$

`PFireball` – object representing the emission source of "our" particles.

Programming-wise, the `PFireball` object mimics a particle, that decays into "our" `PParticle`.
⇒ one should implement the `PChannel` that links the `PFireball` object and the emitted `PParticle`.

```
PFireball* source_pi0 = new PFireball (
  "pi0",   /* type of particle emitted from source                          */
  0.040,   /* Beam Ekin / nucleon  [GeV/A]                                  */
  0.015,   /* Temperature of thermal source of particles [GeV]              */
  0.,      /* Temperature of 2. thermal source if it exists [GeV]           */
  1.,      /* Weight of 1. source                                           */
  0.,      /* Beta param (v/c) of radial expansion of source if applicable  */
  0.,      /* a2 coefficient of theta angle distribution                    */
  0.,      /* a4 coefficient of theta angle distribution                    */
  0.,      /* v1 coefficient of  phi  angle distribution                    */
  0.       /* v2 coefficient of  phi  angle distribution                    */
);
```

Example 7 (**PFireball**) Emission of π^0 mesons from the collision zone. Temperature is 15 MeV. The beam ion moves in Lab with E_{kin} = 40 MeV / nucleon.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int thermal_pi0_gg ()
{
  PFireball* source_pi0 = new PFireball (
      "pi0", 0.040,  0.015,
      0., 1., 0.,
      0., 0.,
      0., 0.
  );
  source_pi0 ->Print ();

  PParticle* pi0 = new PParticle ("pi0");
  PParticle* parttable_fireball_pi0[] = {source_pi0, pi0};
  PChannel*  chan_fireball_pi0 = new PChannel (parttable_fireball_pi0, 1 );

  PParticle* g1  = new PParticle ("g");
  PParticle* g2  = new PParticle ("g");
  PParticle* parttable_pi0_gg [] = {pi0, g1, g2};
  PChannel*  chan_pi0_gg = new PChannel (parttable_pi0_gg, 2 );

  PChannel*  tab_of_chan[] = { chan_fireball_pi0 , chan_pi0_gg };

  PReaction* reaction_thermal_pi0 = new PReaction (
    tab_of_chan, "thermal_pi0_gg_Eb40MeV_T15MeV",
     2,
     1, 0, 0, 0
    );
  reaction_thermal_pi0 ->Print ();
  reaction_thermal_pi0 ->loop (10000, 1);
  return 0;
}
```

Definition of Fireball

Particle type, beam E_{kin} [GeV], T of source [GeV]
T of 2. source [GeV], weight of 1. source, radial β
a_2 and a_4 coefficients of ϑ angle distribution
v_1 and v_2 coefficients of φ angle distribution

Linking of Fireball and π^0 into channel

Linking of π^0 and $\gamma\gamma$ into channel

Table of channels

Definition of reaction

Number of channels in table

Example 8a ("Cocktail") : spectra originating from more than 1 source.

- You can lay side by side two or more independent sources contributing to your final spectrum.
- You can also weight them, using `SetW (weight)`. Pluto will compose each event with the contributions in the 1 : 1 proportions. But the weights will be remembered.
  At the stage of analysis of output data, you can fill histograms applying those weights.

Example: some mesons like ω or ϕ decay into e^+e^- pairs. Let's make a cocktail of ω (70%) and ϕ (30%).

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)
int dielectron_sources () {
  PFireball fireOmega ("w"   , 2.0, 0.100, 0., 1., 0., 0., 0., 0., 0. );
  PFireball firePhi   ("phi", 2.0, 0.100, 0., 1., 0., 0., 0., 0., 0. );
  fireOmega.SetW (0.7);
  firePhi  .SetW (0.3);
```

← These fireballs and their decay products will remember these weights

```
  PParticle omega ("w") , ep2 ("e+") , em2 ("e-") ;
  PParticle* arr_fireOmega_omega[] = { &fireOmega , &omega  };
  PParticle* arr_omega_ee        [] = { &omega , &ep1 , &em1 };
  PChannel * ch_fireOmega_omega    = new PChannel ( arr_fireOmega_omega , 1 );
  PChannel * ch_omega_ee           = new PChannel ( arr_omega_ee        , 2 );
```

```
  PParticle phi   ("phi") , ep1 ("e+") , em1 ("e-") ;
  PParticle* arr_firePhi_phi    [] = { &firePhi , &phi   };
  PParticle* arr_phi_ee         [] = { &phi, &ep2 , &em2 };
  PChannel * ch_firePhi_phi       = new PChannel ( arr_firePhi_phi     , 1 );
  PChannel * ch_phi_ee            = new PChannel ( arr_phi_ee          , 2 );
```

```
  PChannel* tab_chans[] = { ch_fireOmega_omega, ch_omega_ee, ch_firePhi_phi, ch_phi_ee };

  PReaction* r_TwoFBs = new PReaction ( tab_chans, "omegaphi_ee", 4, 1, 0, 0, 0);
  r_TwoFBs->Print();
  r_TwoFBs->loop (1000, 1);
  return 0;
}
```

Example 8b ("Cocktail") : let's now make the histogram of masses of parents, using the inscribed weights.

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)
int dielectron_readtree () {
  TClonesArray* partArray = new TClonesArray ("PParticle", 10);
  PParticle* parts[10] ;

  TFile* fileIn = new TFile ("omegaphi_ee.root");
  TTree* tree = (TTree*) fileIn->Get ("data");
  tree->SetBranchAddress ("Particles", &partArray );

  TH1F* hMass = new TH1F ("hmass", "", 150, 0., 1.5);

  for (int iR = 0 ; iR < tree->GetEntries() ; iR++ ) {
    tree->GetEntry (iR);
    PParticle* ep_om, * em_om , * ep_phi , * em_phi ;
    for (int iP = 0; iP < partArray->GetEntries() ; iP++ )
      parts[iP] = (PParticle*) partArray->At (iP);

    for (int iP = 0; iP < partArray->GetEntries() ; iP++ ) {
      int pid         = parts[iP] -> ID() ,
          parentIndex = parts[iP] -> GetParentIndex();
      if ( parentIndex < 0 ) continue;
      if       ( parts[parentIndex]->ID() == 52 ) {
        if ( pid == 2 ) ep_om  = parts[iP] ;
        if ( pid == 3 ) em_om  = parts[iP] ;
      } else if ( parts[parentIndex]->ID() == 55 ) {
        if ( pid == 2 ) ep_phi = parts[iP] ;
        if ( pid == 3 ) em_phi = parts[iP] ;
      }
    }
    PParticle omega = *ep_om  + *em_om  ,
              phi   = *ep_phi + *em_phi ;

    hMass->Fill ( omega.M() , ep_om ->W() );
    hMass->Fill ( phi  .M() , ep_phi->W() );
  }
  return 0;
}
```

- Matching $e^{\pm}$ with proper parent
- If a parent is a ω meson
- If a parent is a ϕ meson
- Reconstructing 4-vector of parents from children
- Reconstructing 4-vector of parents from children

Example 9 What if we want a particle to decay into more channels (some or all) ?

- `PDecayManager` is the master tool to use. Through it, we define the decays, initialize the reaction and loop.
  `PDecayChannel` object = list of particle's decay channels (with info on Branching Ratios = BR).

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int lambda_decays ()
{
  PFireball* source_lambda = new PFireball (
   "Lambda", 2.0, 0.100, 0., 1., 0., 0., 0., 0., 0.);

  PDecayManager* pdm = new PDecayManager;
  pdm->SetVerbose (1);

  PDecayChannel* ch_FBall  = new PDecayChannel;
  ch_FBall ->AddChannel ( 1. , "Lambda" );

  /* To activate all the available decays of Lambda */

  pdm->SetDefault ("Lambda");

  /* Instead, to create your own decay list with your BR */
/*
  PDecayChannel* ch_lam_decays = new PDecayChannel;
  ch_lam_decays->AddChannel ( 0.639, "p", "pi-" );
  ch_lam_decays->AddChannel ( 0.358, "n", "pi0" );

  pdm->AddChannel ("Lambda", ch_lam_decays );
*/
  pdm->InitReaction ( source_lambda , ch_FBall );
  pdm->PrintReactionList() ;

  pdm->loop (10000, 0, "lambda_decays", 1, 0, 0, 1, 1);
  return 0;
}
```

Initialization of Decay Manager

A decay channel. It represents "something" decaying into Λ .
A connection to the parent (here: Fireball) is done further, with `PDecayManger::InitReaction`

To activate **all** Λ's default decays (including info on BR)

Instead, to define decays **we want**, including info on BR.
① we say "something" decays this way
② we inform Manager that Λ decays in such ways

We initialize reaction:
① define the initial particle
② define, how it decays

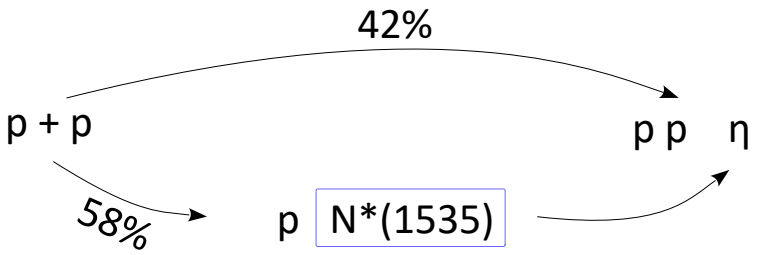
Loop command with name of output file containing TTree ⊕ flags

- **Creating "cocktails" of several channels originating from elementary reactions**

① By colliding two hadrons we create a composite particle.
② Then we employ the `PDecayManager/PDecayChannel` technique, as if that composite particle had several ways of decay, each with given weight.

Example 10 : production of η ("eta") meson both:

- ▻ directly (42%)
- ▻ via N*(1535) resonance (58%)




```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int eta_cocktail_simple ()
{
  PParticle p1 ("p", 2.2), p2 ("p"), comp = p1 + p2;

  PDecayManager* pdm = new PDecayManager;

  PDecayChannel* dc   = new PDecayChannel;
  dc->AddChannel (0.42, "p", "p", "eta"); // direct η production
  dc->AddChannel (0.58, "p", "NS11+");    // or: first N*(1535)

// Then, we take care for decay of N*(1535)

  PDecayChannel* nstar_decay = new PDecayChannel;
  nstar_decay->AddChannel (1.0 , "eta", "p");
  pdm->AddChannel ("NS11+", nstar_decay );

  pdm->InitReaction ( &comp , dc );
  pdm->Print();
  pdm->loop (100, 0, "eta_sample", 1, 0, 0, 1, 1);

  return 0;
}
```

`comp` will be the initial particle.

We will connect `comp` and its decays at the level of `InitReaction`.

▻ Nb. η itself should also decay, but we aim here to grasp the main idea.

Data output into TNtuple database and THnF histograms

Pluto allows to define the physical quantities ("variables") related to the involved particles, and to **preview them** in the **TNtuple data base**. Notice: 1 reaction = 1 database entry.

① We create the TNtuple data base and a separate output file, where this TNtuple will be stored:

```
TFile*   file_TNtuple = new TFile   ("ntufile.root", "RECREATE");
TNtuple* my_ntuple    = new TNtuple ("ntu", "Name", "thgam:phgam:..." );
```

② We define how the variables should be calculated in each event:

```
my_reaction->Do ("thgam = [g]->Theta()* TMath::RadToDeg()" );
my_reaction->Do ("phgam = [g]->Phi()  * TMath::RadToDeg()" );
```

③ We ask the PReaction to book the generation of TNtuple database. We can apply **filters** here:

```
my_reaction->Output (my_ntuple , "if (thgam>20 && thgam < 50) " );
```

④ We can also apply **filters** to the main tree (TTree data). Each variable, whose name starts from # is a filter. Entry in TTree is made only if none of filters is found to be 0.

```
my_reaction->Do ("#myaccept= 1; if (thgam<20 || thgam > 50) ; #myaccept= 0");
```

Preview of variables via 1, 2, 3 - dimensional histograms (THnF)

① We create a histogram:

```
TH1F* my_histo = new TH1F ("myhisto", "Photon azim. angle", 90, -180., 180.);
```

② In reaction we define special variables **_x (_y, _z)**, then apply **filters** (independent from previous ones):

```
my_reaction->Do (my_histo, "if thgam>20 && thgam < 50; _x=phi");
```

Notice: Writing the TNtuple and THnF to file should be ordered explicitly:

```
gDirectory->cd ("ntufile.root:/");
my_ntuple ->Write ();                    my_histo ->Write();
```

Example 11 Compton scattering: preview variables in TNtuple and THnF + applying filters to various outputs

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int compton ()
{
  PReaction* R = new PReaction ("0.010",
            "g", "e-", "g e-", "compton_ttree");
  R->Print ();

  TFile* file_TNtuple = new TFile ("compton_tntuple.root", "RECREATE");

  TNtuple* my_ntuple = new TNtuple ("ntu", "Monitor ntuple",
                       "thgam:phgam:thel:phel:opang" );

  R->Do ("RTD = TMath::RadToDeg()");
  R->Do ("thgam = [g]->Theta()    * RTD");
  R->Do ("phgam = [g]->Phi()      * RTD");
  R->Do ("thel  = [e-]->Theta()   * RTD");
  R->Do ("phel  = [e-]->Phi()     * RTD");
  R->Do ("opang = [g]->Angle([e-])* RTD");
  R->Output (my_ntuple, "if (thgam>20 && thgam < 50)" );

  R->Do ("#myAccept=1; if (thgam<20 || thgam > 50); #myAccept=0");

  TH1F* my_histo = new TH1F ("myhisto", "Opening angle", 90, 0., 90.);
  R->Do (my_histo , "if thgam > 20 && thgam < 50 ; _x = opang");

  R->Loop (10000, 1);
  gDirectory->cd ("compton_tntuple.root:/");
  my_ntuple->Write ();
  my_histo ->Write ();

  return 0;
}
```

Definitions of variables

apply filter to TNtuple

TTree filter

apply filter to TH1F

TNtuple

TH1F

Output

Scripting language

Dedicated manual by I. Froehlich:

https://www.fuw.edu.pl/~kpias/ctnp/pluto/ifroehlich_pluto_script_manual.pdf

Adding new particles to Pluto's data base

To add a new particle and possibly provide the required decays,

① We start from creating the "manager" object:

```
PStaticData* sd = makeStaticData() ;
```

② We give PID, name and particle's mass. PID = -1 will assign the first free number. We can also give Γ (width).

```
sd->AddParticle ( -1, "New", mass [GeV] );
sd->SetParticleTotalWidth ( "New", width [GeV] );
```

③ We assign to this particle all the necessary properties
Caution: missing important data may result in lack of the relevant decay model, and wrong distributions.

```
sd->SetParticleBaryon ("New", 1);
                Meson ( )     Charge ( )     Spin ( )      Isopin ( )    ....
```

④ For unstable particles: we define the decay channel(s). PID = -1 will assign the first free No. from the list.

```
sd->AddDecay (-1, "New -> b + c", "New", "b,c", 1.);
```

Example 12 Creation of the e^+e^- pair in the vicinity of the nucleus (represented as a new particle)

New particle

```
R__LOAD_LIBRARY ($PLUTOLIBDIR/libPluto.so)

int gamma_pairproduction ()
{
  double mN = 0.938, A = 40, mass_nucl = mN * A;
  PStaticData* sd = makeStaticData() ;
  sd->AddParticle (-1, "A", mass_nucl );

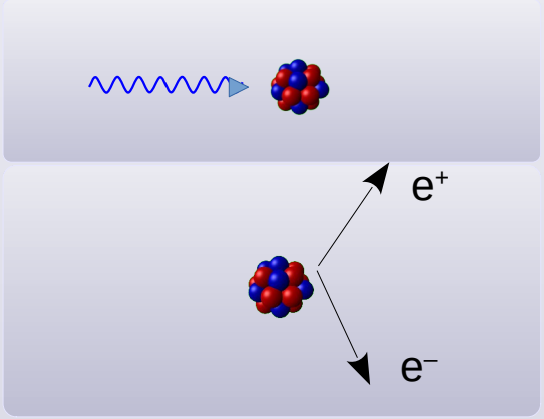
  PReaction* R = new PReaction ( "1.", "g", "A", "A e+ e-", "gamma_ee_ttree");
  R->Print ();

  TFile* file_TNtuple = new TFile ("gamma_pairprod_tntuple.root", "RECREATE");
  TNtuple* my_ntuple = new TNtuple
         ("ntu",  "My Ntuple", "pxep:pyep:pzep:pxem:pyem:pzem:pxA:pyA:pzA" );

  R->Do ("pxep = [e+]->Px() ");
  R->Do ("pyep = [e+]->Py() ");
  R->Do ("pzep = [e+]->Pz() ");
  R->Do ("pxem = [e-]->Px() ");
  R->Do ("pyem = [e-]->Py() ");
  R->Do ("pzem = [e-]->Pz() ");
  R->Do ("pxA  = [A]->Px() ");
  R->Do ("pyA  = [A]->Py() ");
  R->Do ("pzA  = [A]->Pz() ");

  R->Output (my_ntuple);
  R->Loop (10000, 1);

  file_TNtuple->Write ();
  file_TNtuple->Close ();
  return 0;
}
```



Example 13 Text files: writing kinematics + reading kinematics to process particles inside Pluto

- Let's say we want to store selected information in the text file .

  In this example we do so for a list of η's with given momenta.

```
int eta_write_ascii ()
{
  PReaction* R = new PReaction
                 ("3.", "p", "p", "p p eta") ;

  R->Do ("px = [eta]->Px() ; py = [eta]->Py() ;"
         "pz = [eta]->Pz() ; m  = [eta]->M () ") ;

  R->Output("eta_sample.txt", "echo $px $py $pz $m");
  R->Print();
  R->Loop (1000) ;
  return 0;
}
```

```
int eta_read_ascii ()
{
 PParticle eta ("eta"), pip ("pi+"), pim ("pi-"), pi0 ("pi0");
 PParticle* ar_eta_3pi[] = {&eta , &pip , &pim , &pi0} ;
 PChannel* ch_eta_3pi  = new PChannel ( ar_eta_3pi , 3);

 PReaction* R = new PReaction (&ch_eta_3pi, "eta_decays", 1);

 PProjector* input = new PProjector ();
 input->AddInputASCII ("eta_sample.txt");
 input->Do ("readline {@px @py @pz @m} ");
 input->Do ("myeta = P3M (px, py, pz, m) ");
 input->Do ("myeta->SetID (eta.pid) ");
 input->Do ("push (myeta) ");

 R->AddPrologueBulk (input);
 R->Print();
 R->Loop(1000);
 return 0;
}
```

- Pluto has a tool to read your data from an ASCII file, and assign the kinematics of given particle with these data.

  E.g. we have a text file with momenta vectors of η mesons.

  Here we program a specific channel: $\eta \to \pi^+ \pi^- \pi^0$, but the η's will have momenta from that ASCII file.