Introduction to DaVinci 2

- Start to write a full selection sequence
- Write an algorithm that selects $\mathbf{J}/\psi \rightarrow \mu\mu$
- DC04 to DC06 translation table

June 2006 Bologna Software Course

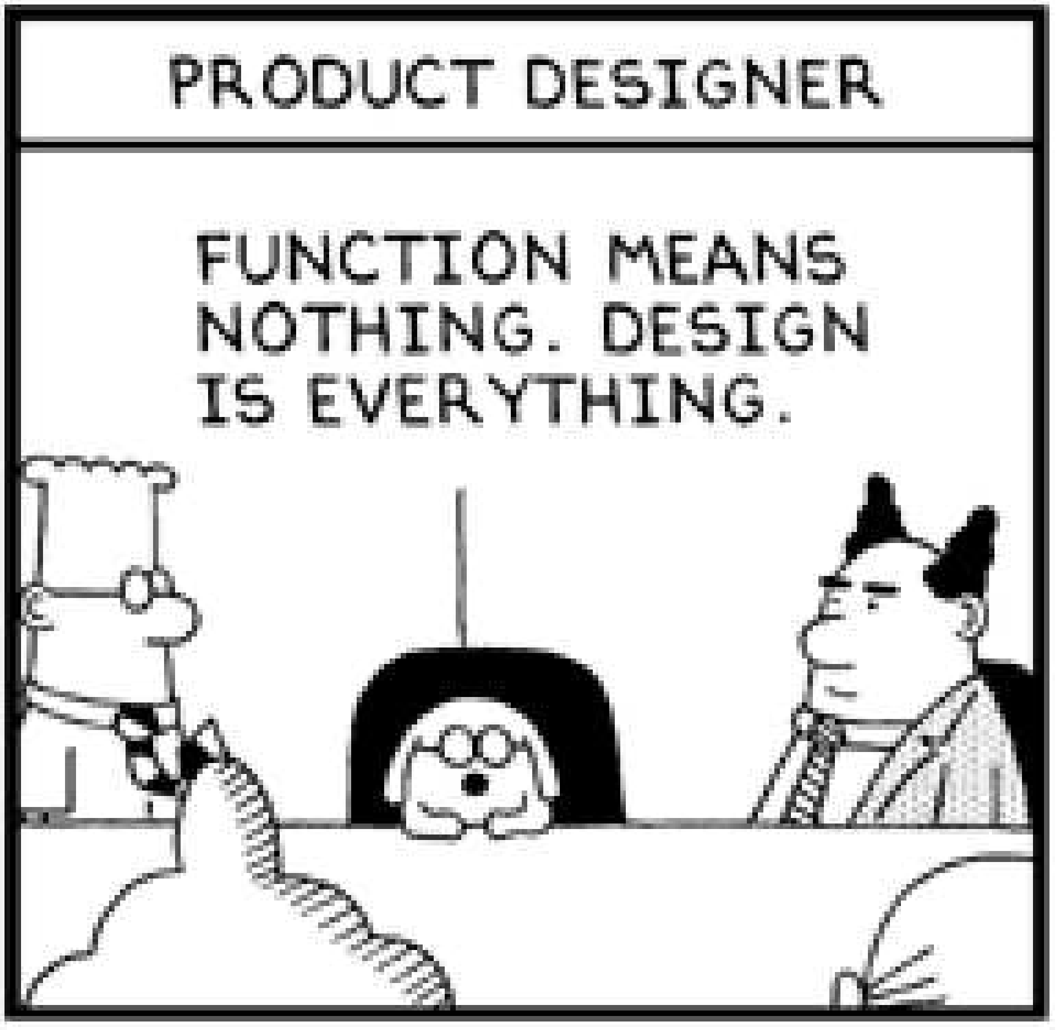
Patrick Koppenburg
Imperial College
London

Select $\mathrm{B_s} \to \mathrm{J}/\psi\phi$:

- Design it
- Make particles
- Make J/ψ's

This part is based on the `Tutorial/Analysis` package. All can be found there.

Design it

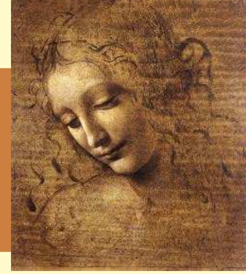


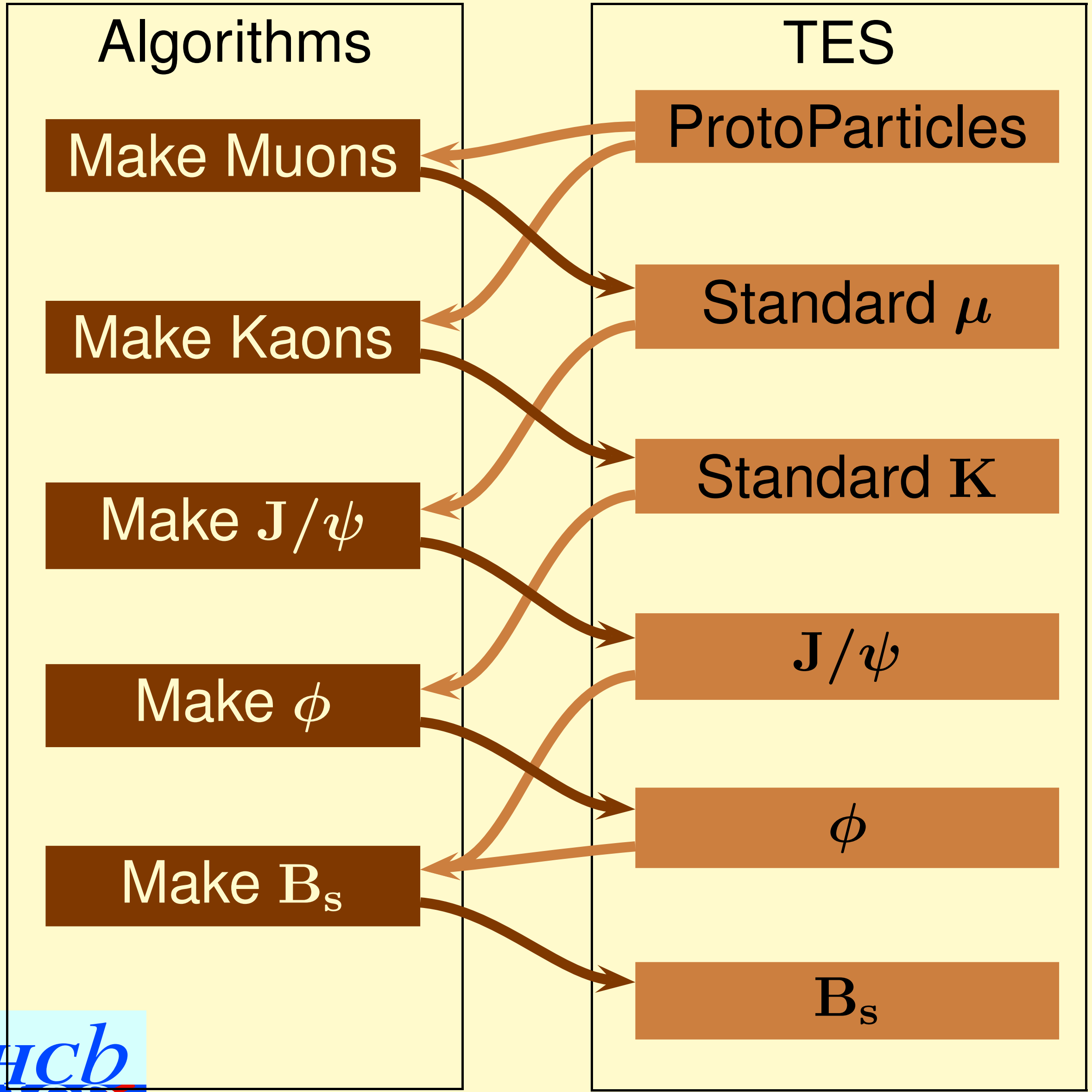
One could write a single algorithm that makes particles, combines μ into $\mathbf{J}/\psi$ and $\mathbf{K}$ into ϕ and then makes the $\mathbf{B_s}$.

This is not a good idea!

It is much better to write a simple algorithm for each task and to save the intermediate data in the transient event store (TES).

Design it



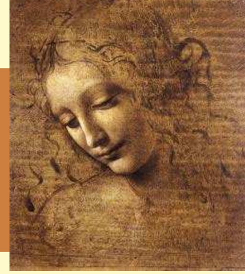


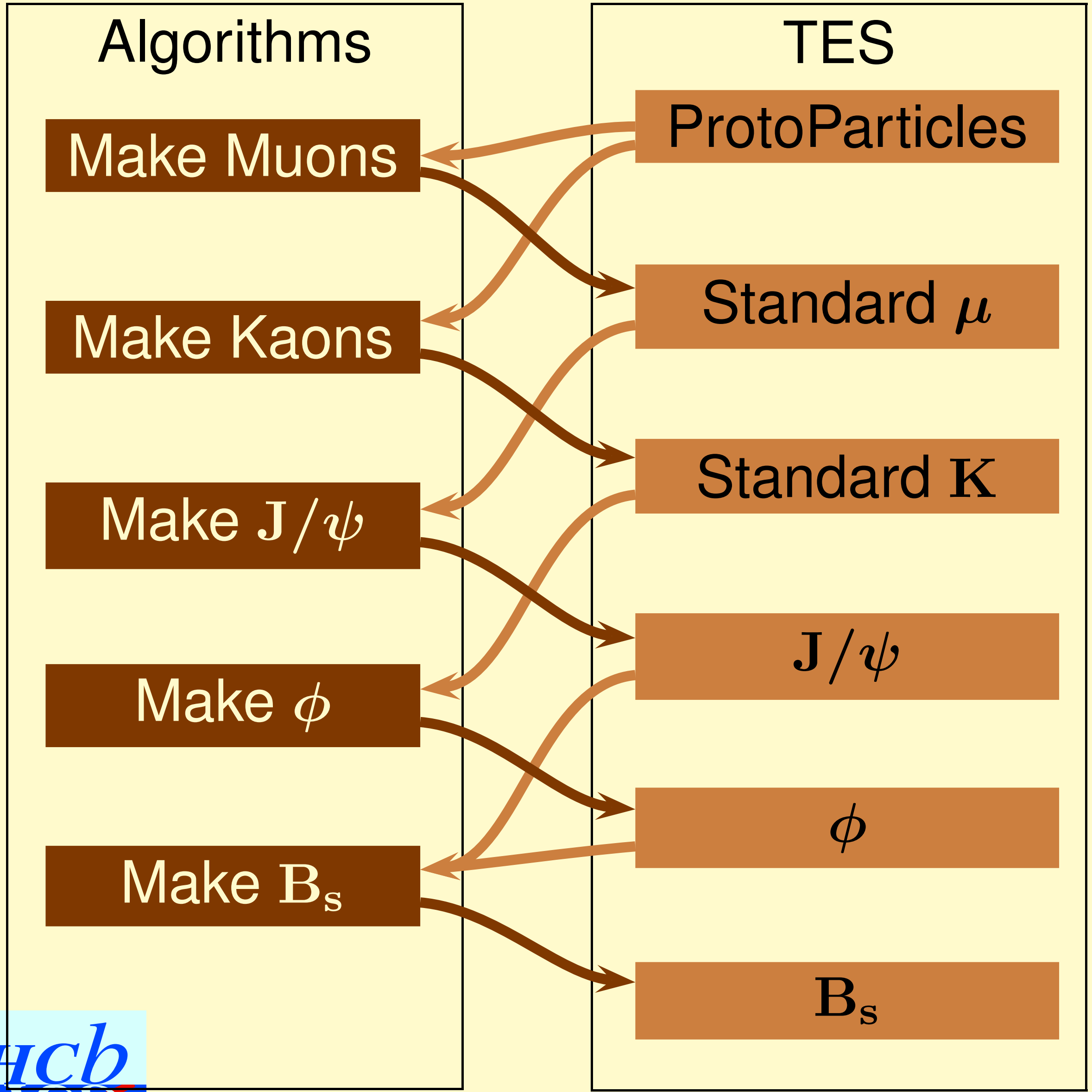
One could write a single algorithm that makes particles, combines μ into $\mathbf{J}/\psi$ and $\mathbf{K}$ into ϕ and then makes the $\mathbf{B_s}$.

This is not a good idea!

It is much better to write a simple algorithm for each task and to save the intermediate data in the transient event store (TES).

Design it





- Algorithms have as many inputs as needed, but only one output
- TES locations can be read by any algorithm, but only one can write to them

Let's start to write the chain!

Locations in the TES

The output of a `DVAlgorithm` called `"MyAlgo"` is saved in

- `/Event/Phys/MyAlgo/Particles` and
- `/Event/Phys/MyAlgo/Vertices`

Algorithm instance names have to be unique $\rightarrow$ particles will be stored in different locations. (This is ensured by the `PhysDesktop` tool).

This becomes important if you want to test the correlation of your $\mathbf{B_s} \rightarrow \mathbf{J}/\psi\phi$ selection with the DC06 selection of $\mathbf{B} \rightarrow \mathbf{J}/\psi\mathbf{K^0_S}$, or test the efficiency of the HLT $\mathbf{J}/\psi$ selection.

Make sure all algorithm names are unique!
It is mandatory for the stripping.

Some cuts and counters

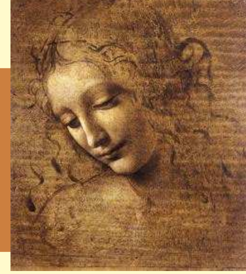
Cuts should be defined by options, we hence need them to be data members of the algorithm. In this example we suggest to cut on

- A mass cut ($\pm 100\,\mathrm{MeV}$ is fine)
- A χ^2 cut (< 100 for instance)

Don't forget to declare these cuts as properties of the algorithm. We can also add a few counters

- Number of $\mathbf{J}/\psi$ found
- Number of events processed

Particle properties



We will need the PDG mass and PID of the $\mathbf{J}/\psi$. We will to get them once from the `ParticlePropertySvc` in the initialisation and will use them in each events. They hence also have to be member variables.

```
ParticleProperty* mother = ppSvc()->find( "J/psi(1S)" );
if ( !mother ) { //
  err() << "Cannot find particle property for J/psi(1S)" << endmsg ;
  return StatusCode::FAILURE;
}
m_jPsiID = mother->pdgID();
m_jPsiMass = mother->mass();
```

- Use the pointer to the `ParticlePropertySvc` `ppSvc()`.
- The name of the $\mathbf{J}/\psi$ can be found in `$PARAMFILESROOT/data/ParticleTable.txt`.
- It returns a `ParticleProperty`

Filtering the muons

- The first thing we need to do in `execute()` is to get positive and negative muons.
- We know from the previous session how to get muons.
- The `ParticleFilter` allows to filter muons by charge using the `filterNegative` and `filterPositive` methods
- But is also allows to apply cuts by options.

```
Jpsi2MuMu.ParticleFilter.CriteriaNames = {"KinFilterCriterion" } ;
Jpsi2MuMu.ParticleFilter.KinFilterCriterion.MinPt = 1*GeV ;
```

  which allows to cut on the p_T of the muons.
- You can put as many filter criteria as you like
- We will see much more on that in the next session.

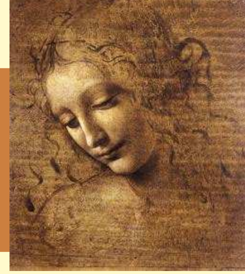
Vertex fit

The `IVertexFit` interface is common to all vertex fitters. The fit method always takes references to `Particles` as input and returns a `Vertex` and the created `Particle`.

```
LHCb::Vertex MuMuVertex;
LHCb::Particle Jpsi(m_jPsiID);
StatusCode scFit = vertexFitter()->fit(*(*imp),*(*imm),Jpsi,MuMuVertex);
if (!scFit) {
  Warning("Fit error");
  continue;
}
```

Presently the fitter returns a failure if the fit failed. This is not a good reason to stop the execution. So catch this error and handle it properly.

Use the PhysDesktop



To save the $\mathbf{J}/\psi$ candidate you need to

1. Declare each $\mathbf{J}/\psi$ to the `PhysDesktop`

   ```
   desktop()->save(&Jpsi);
   ```

2. Save all declared `Particles` at the end

   ```
   return desktop()->saveDestop();
   ```

The `PhysDesktop` has also methods to save a given list of particles

```
LHCb::Particle::Vector myPsis ;
sc = desktop()->saveTrees( myPsis );
sc = desktop()->saveTrees( m_jPsiID );
```

- All particles and vertices will be saved to `/Event/Phys/Jpsi2MuMu/Particles` and `/Event/Phys/Jpsi2MuMu/Vertices`

Particles and Vertices

The `Particle` and `Vertex` classes depend on each other

```
LHCb::Vertex* LHCb::Particle::endvertex() ;
SmartRefVector<LHCb::Particle> &
                LHCb::Vertex::products() ;
```

To navigate from a `Particle` to its daughters do:

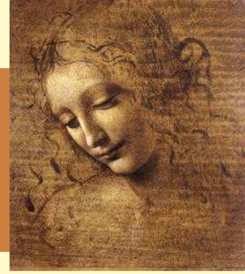
```
SmartRefVector<LHCb::Particle> themus =
                Jpsi.daughters() ;
```

And to navigate to the `Particles` used to fit the vertex do

```
SmartRefVector<LHCb::Particle> themus
              = Jpsi.endVertex()->products() ;
```

In the case of the $\mathbf{J}/\psi$ they are the same, but not in the case of the $\mathbf{B_s} \longrightarrow \mathbf{J}/\psi\phi$, where the vertex is made with the two muons and the two kaons.

Options

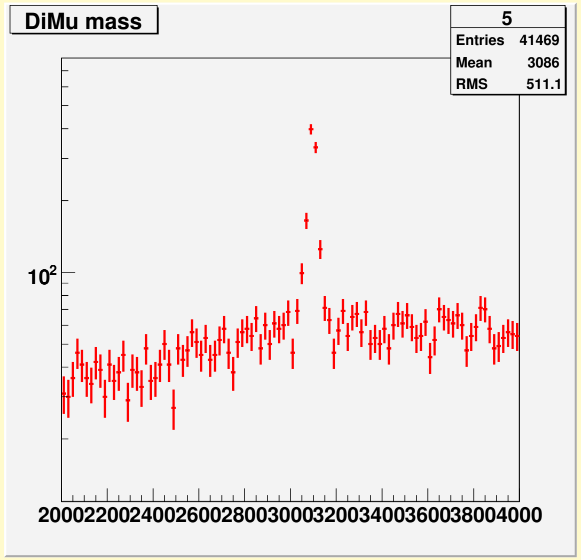


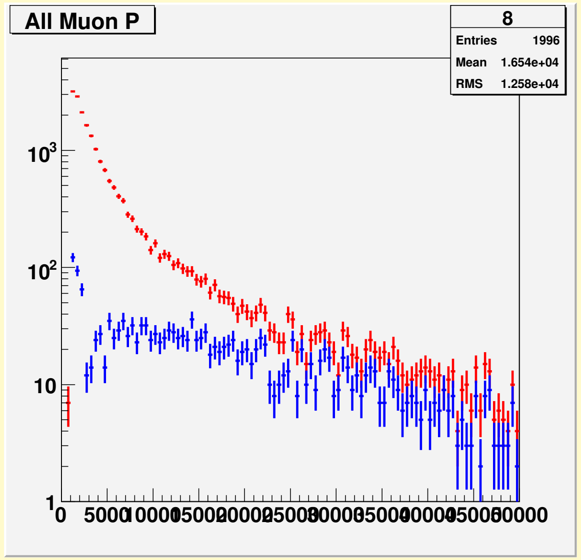
```
TutorialSeq.Members += { "TutorialAlgorithm/Jpsi2MuMu" };
Jpsi2MuMu.PhysDesktop.InputLocations = { "Phys/StdLooseMuons" } ;
Jpsi2MuMu.MassWindow = 50*MeV ;
Jpsi2MuMu.MaxChi2 = 100 ;
Jpsi2MuMu.OutputLevel = 3 ;
```

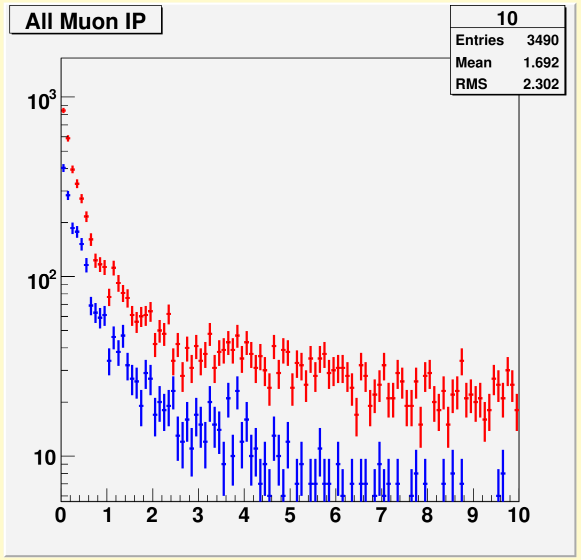
- Here we name the instance of `TutorialAlgorithm` `Jpsi2MuMu`
- Configure the cuts and the verbosity level.
- Tell the `PhysDesktop` from where to take the particles.
  - It automatically adds `"/Event/"` to the location if necessary.

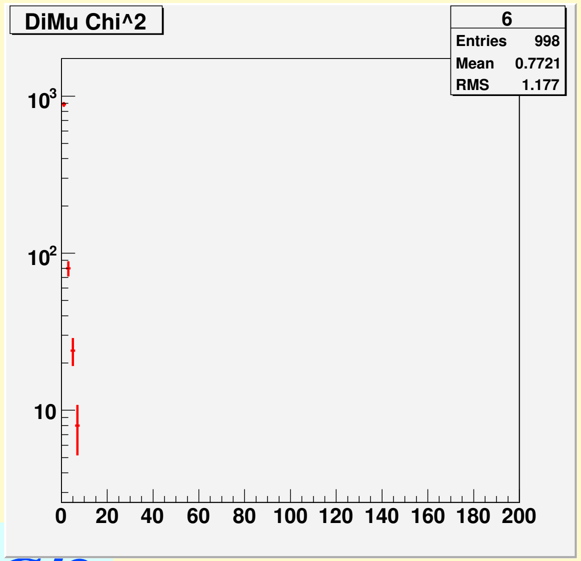
Now you can run it, but make sure you use signal data if you want to select something.

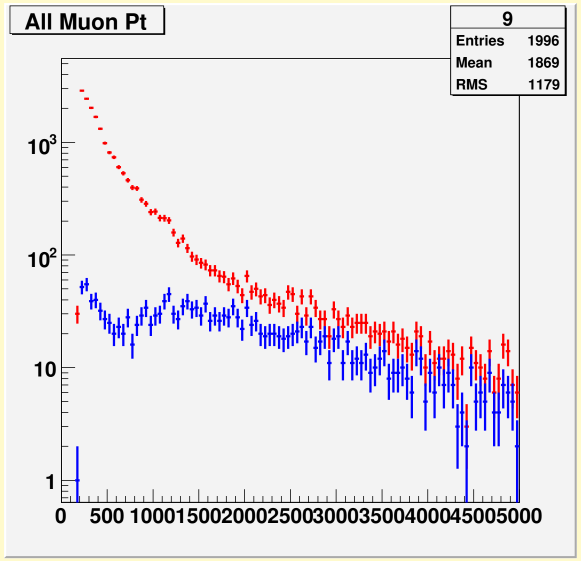
A Few Histograms

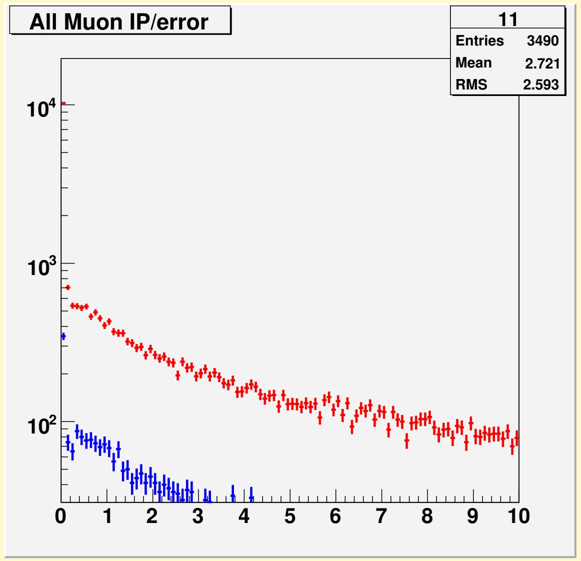












DC04 to DC04 translation table

- We have revised the Event Model in 2005
  - Track event model in Spring:
    (`TrStoredTrack` $\longrightarrow$ `Track`)
  - Physics event model in Summer:
    (`ParticleVector` $\longrightarrow$ `Particle::Vector`)
  - All the rest in Autumn:
    (`Particle` $\longrightarrow$ `LHCb::Particle`)
- Based on this we have started rewriting all the code and took the opportunity to make a few backward-incompatible changes
  - All vertex fitters share the same interface (one can switch them by options!)
  - `DVAlgorithm` behaves as `GaudiAlgorithm`
  - LoKi has been split into MC and non-MC parts (and more)

DC04 to DC04 translation table

DC04 (**DaVinci** v12rX)	DC06 (**DaVinci** v16rX)
Don't call DVAlgorithm::initialize()	Must call it
Don't call DVAlgorithm::finalize()	Must call it
AnyEventClass	LHCb::AnyEventClass
ParticleVector	LHCb::Particle::Vector (deprecated)
	LHCb::Particle::ConstVector
myB->endVertex()->products()	myB->daughters()
	myB->daughtersVector()
	myB->endVertex()->outgoingParticles()
dynamic_cast<ProtoParticle*> (Kaon->origin())	Kaon->proto()
There are any other event model changes	
vertexFitter()->fit(Dau1,dau2,vertex); particleStuffer()->fillParticle(vertex, mother,pid);	LHCb::Particle mother(pid); vertexFitter()->fit(dau1,dau2, mother,vertex);
desktop()->desktop()->createParticle(&B);	dektop()->save(&B);
Many other signatures changed	

Exercises!

- Let's go for the exercises

Ex. 2 Asks you to reconstruct the $\mathbf{J}/\psi \rightarrow \mu\mu$ as shown

Ex. 3 Lets you already think about the ϕ. We can make a single algorithm that we can call twice, once for the $\mathbf{J}/\psi$, once for the ϕ

- You'll have to give `StdLoseKaons` as input to the second one
- And you'll need a variable that tells which mother to reconstruct.