Introduction to DaVinci

- Introduction
  - overview of **DaVinci** structure
- My first `DVAlgorithm`
  - we will loop over muons and plots some quantities

October 2009 Tutorial

Patrick Koppenburg

NIKHEF

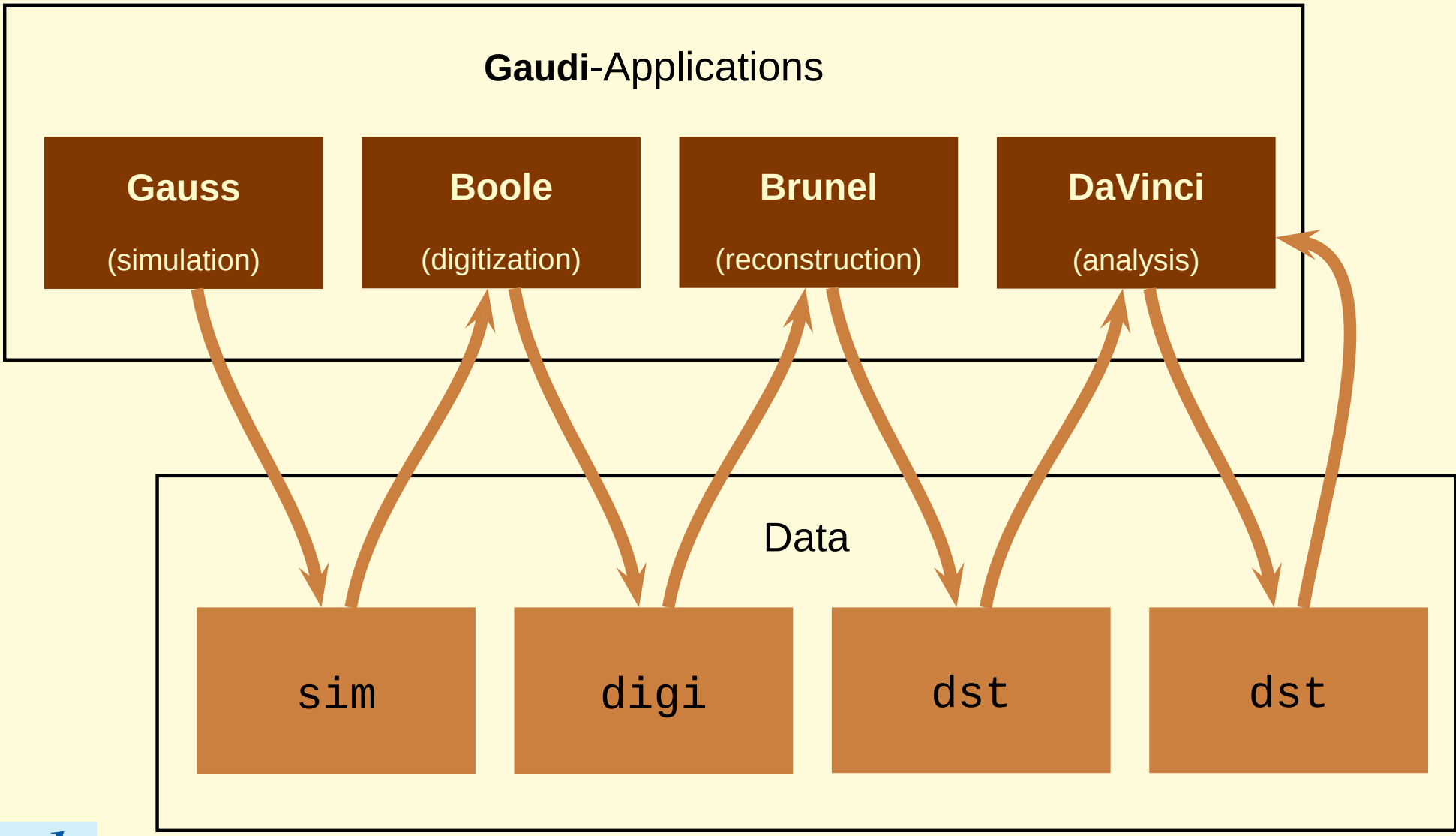
Applications

Gaudi-Applications

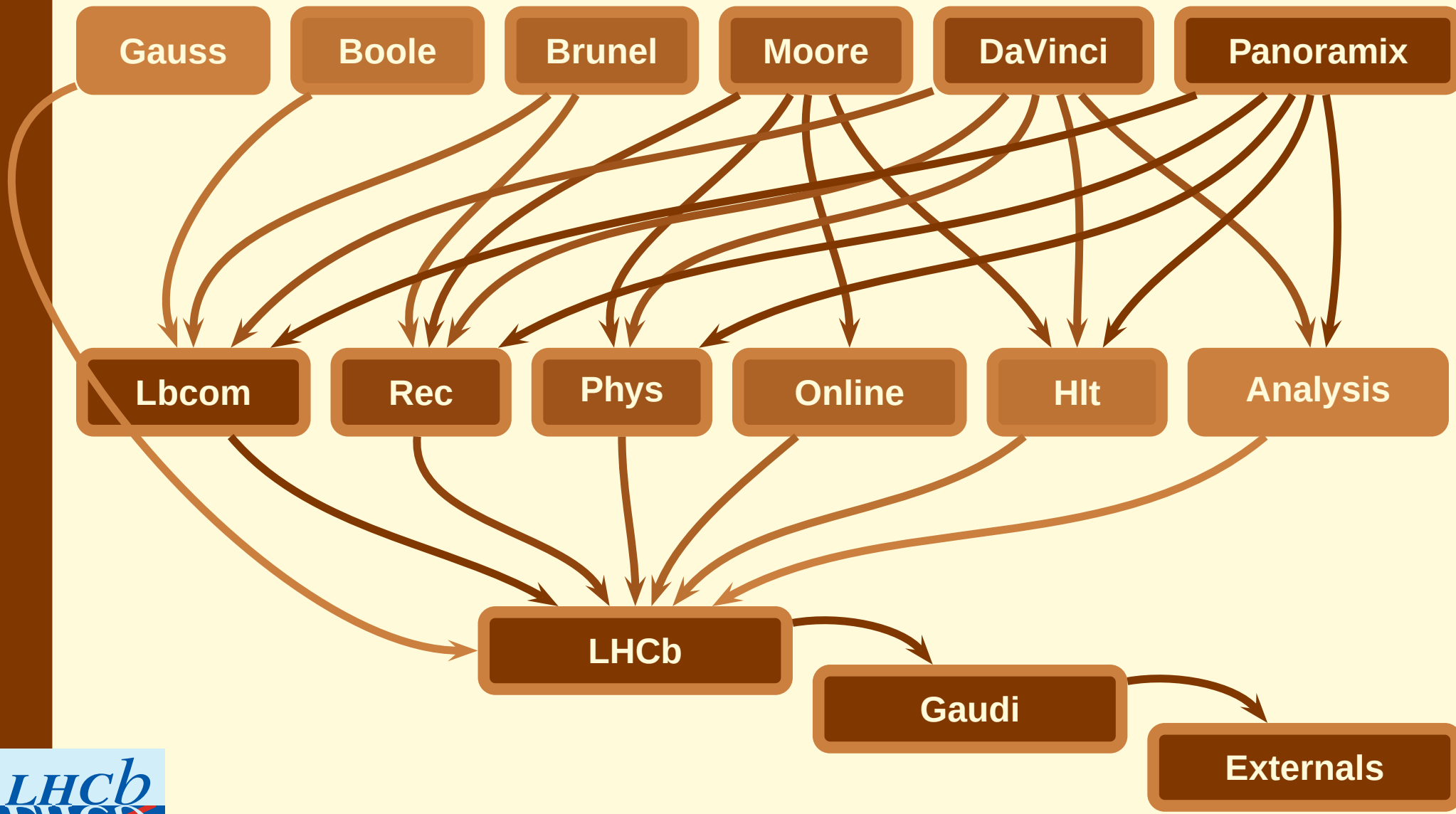
Gauss	Boole	Brunel	DaVinci
(simulation)	(digitization)	(reconstruction)	(analysis)

- There are four applications based on **Gaudi**
- They are actually all **Gaudi**-programs
- The only difference are the packages (shared libraries) included
- One could easily build an application that does it all (like in the old **SICB** days. . .)
- Somewhere here **Panoramix** and **Bender** are missing

Applications



Projects


Gauss
Boole
Brunel
Moore
DaVinci
Panoramix
Lbcom
Rec
Phys
Online
Hlt
Analysis
LHCb
Gaudi
Externals

DaVinci Links

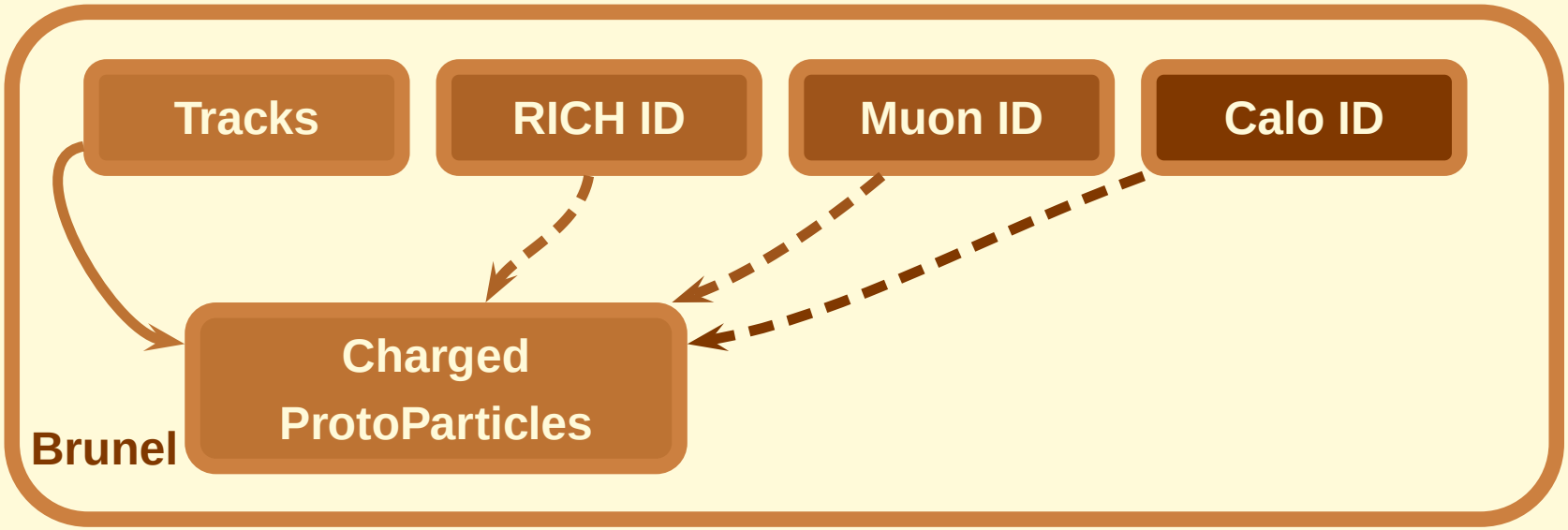
- **DaVinci** web page:
  `http://cern.ch/LHCb-release-area/DOC/davinci/`
  From there you'll find :
  - Some documentation. Links to `doxygen`.
  - The Tutorial page
- Any **DaVinci** question can be asked at the **DaVinci** mailing list:
  lhcb-davinci@cern.ch .
  - That's also the forum to propose improvements of **DaVinci**
  - You need to be registered to use it. You can do that online.
- Distributed analysis question should be asked at
  lhcb-distributed-analysis@cern.ch .
- General software questions should go to lhcb-soft-talk@cern.ch

DaVinci

Warning

- We will use **DaVinci** `v24r3`
  - It is a version all based on **python** configurables
  - There are (almost) no text options (`.opts`) files anymore
    - ➜ You don't really need to know about them
    - ✗ Avoid using them. Even if your supervisor tells you so.

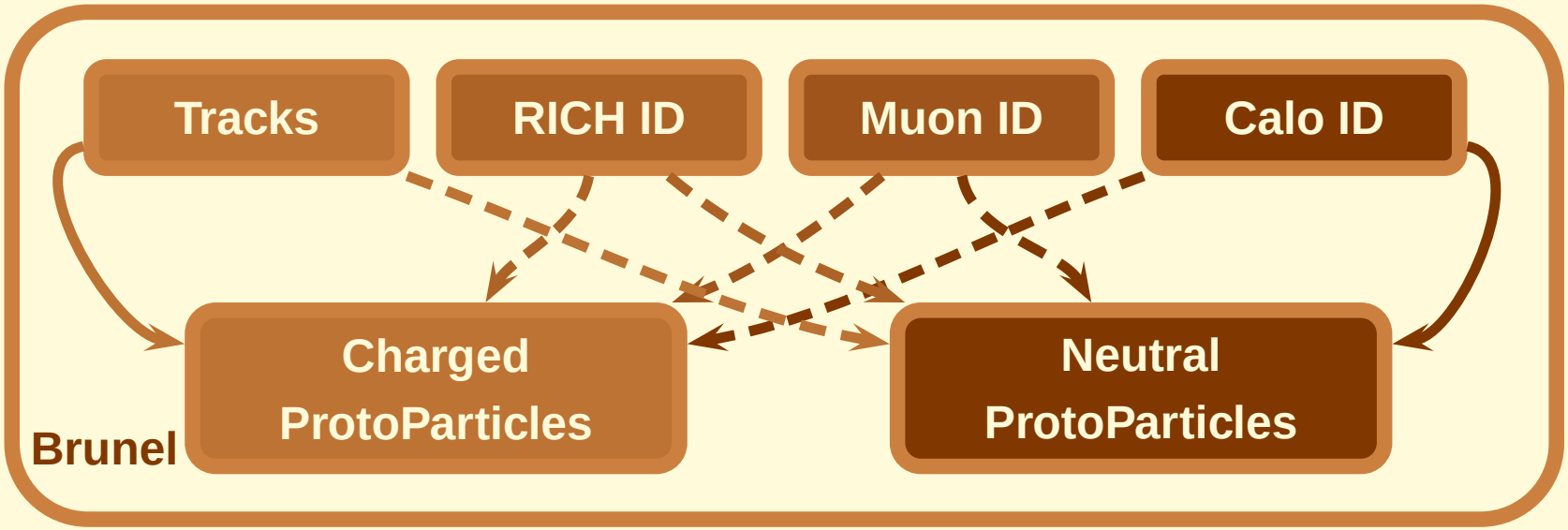
ProtoParticles



`ProtoParticles`

- are the end of the reconstruction stage
- are the starting point of the physics analysis
- have all the links about how they have been reconstructed
- have a list of PID hypothesis with a probability
- contain the *kinematic* information

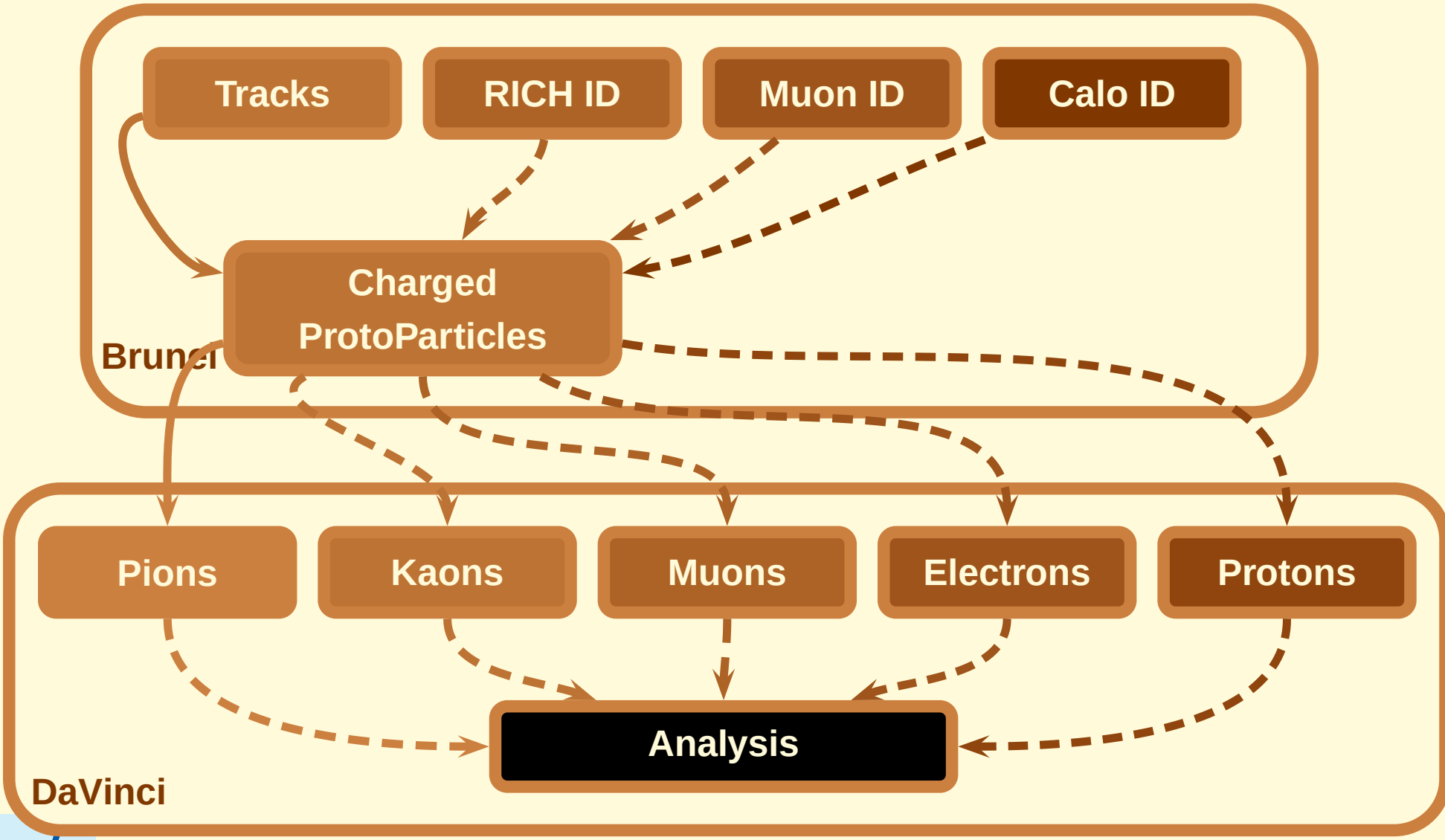
ProtoParticles



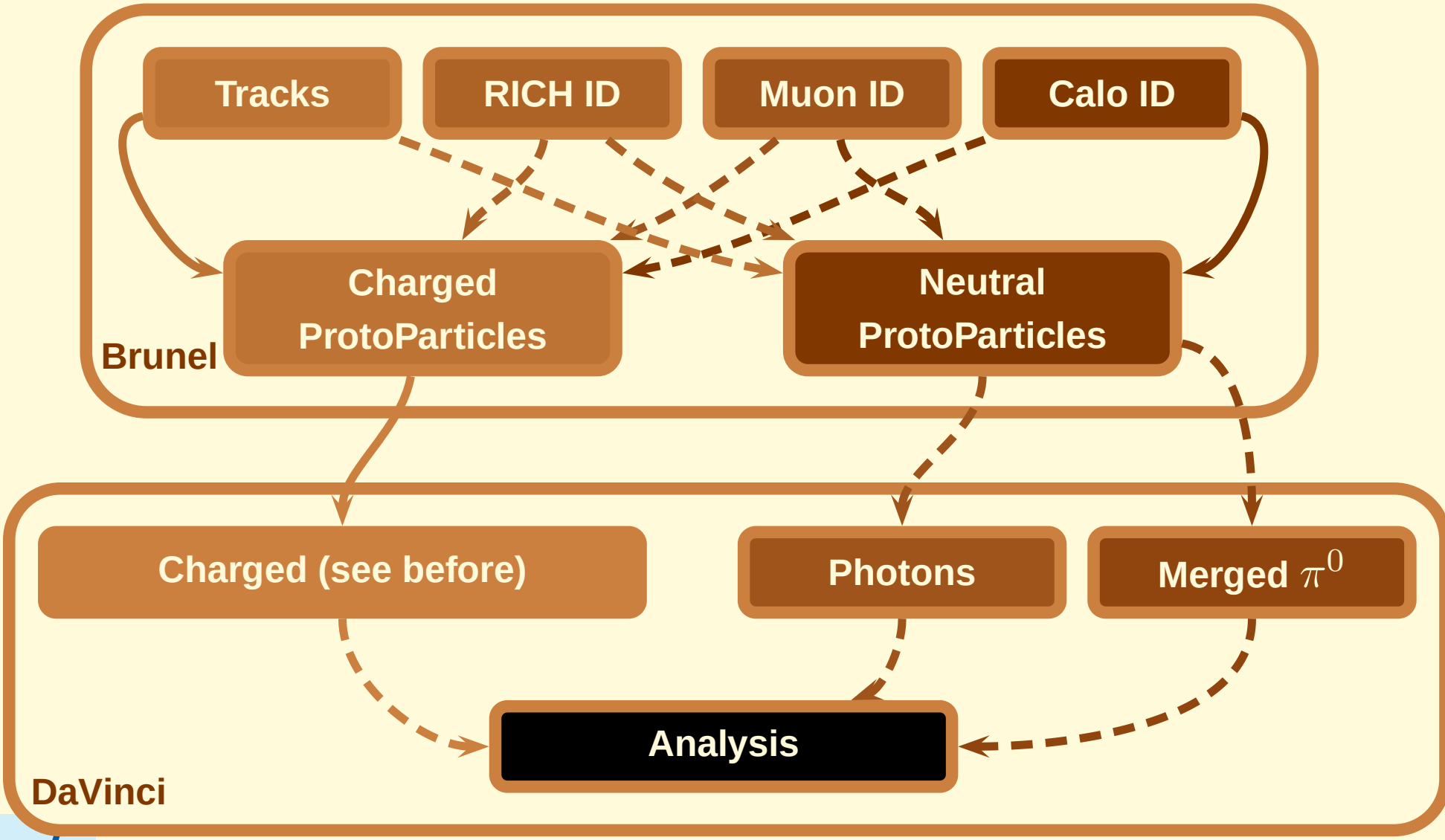
`ProtoParticles`

- are the end of the reconstruction stage
- are the starting point of the physics analysis
- have all the links about how they have been reconstructed
- have a list of PID hypothesis with a probability
- contain the *kinematic* information

Particles


Tracks
RICH ID
Muon ID
Calo ID
Charged ProtoParticles
Brunel
Pions
Kaons
Muons
Electrons
Protons
Analysis
DaVinci

Particles


Tracks
RICH ID
Muon ID
Calo ID
Charged ProtoParticles
Neutral ProtoParticles
Brunel
Charged (see before)
Photons
Merged π^0
Analysis
DaVinci

Particles

- `Particle` = `ProtoParticle` + one PID *choice*
  - ➜ one defined mass
- Physics analyses deal with `Particles`
  - You need to know the 4-vectors to compute the mass of a resonance
- The PID is your choice
  - The same `ProtoParticle` can be made as a π and as a μ...
    - This makes sense. Think of a pion from $B \to \pi\pi$ decaying in flight. Does it stop being a signal pion because it decayed before the Muon detector?
  - Some `ProtoParticles` can be ignored
  - All this is done by configuring a `ParticleMaker` algorithm
    - You don't need to worry about the configuration.
    - Many standards are pre-defined
    - But you need to choose which to use
    - ➜ Next slide

Standard `Particles`

- The `Particles` are actually already done for you. To ensure that everybody agrees on what is a K^+, a π or a K^0_S, we have a set of standard particles predefined.
- They are defined in `python/CommonParticles/*.py` in the `Phys/CommonParticles` package.
- All you need to know are the names of the algorithm that created them : `StdLooseKaons`, `StdTightProtons` ...

  StdNoPIDsXxxx: All tracks are made to `Xxxx`

  StdLooseXxxx: Loose PID cuts for hypothesis `Xxxx` (no cuts for pions)

  StdTightXxxx: Tight PID cuts for hypothesis `Xxxx`

DVAlgorithm

Algorithms contain the algorithmic part to be executed at each event

What **DaVinci** does is defined by the algorithms that are called. An algorithm is any class inheriting from `Algorithm`, which contains

- an `initialize()` method called at begin of job
- an `execute()` method called at each event
- a `finalize()` method called at end of job

To make life easier **DaVinci** contains a base-class `DVAlgorithm` that provides many useful features.

- `DVAlgorithm` inherits from the base-class `GaudiTupleAlg`,
- That inherits from `GaudiHistoAlg`,
- That inherits from `GaudiAlgorithm`,
- That inherits from `Algorithm`.

My first `DVAlgorithm`:

- Create it
- Get some `Particles`
- Loop over them
- Make some histograms

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

Start to write the options

It's a good idea to start with the options:

```
from Gaudi.Configuration import *
#
# 1) Let's define a sequence
#
from Configurables import GaudiSequencer
tutorialseq = GaudiSequencer("TutorialSeq")
#
# 2) Create the Tutorial Algorithm
#
from Configurables import TutorialAlgorithm
tutalg = TutorialAlgorithm()
tutorialseq.Members += [ tutalg ]
```

- Then let's start a sequence of algorithms with one algorithm inside.

Let's write a new algorithm

In `$ANALYSISROOT` type

```
> emacs src/TutorialAlgorithm.{cpp,h}
```

`Emacs` will ask you what you want to create. Answer (D) for `DVAlgorithm` (twice) and you will get a template for a new algorithm that compiles nicely but does nothing at all. (you actually need to modify the file to force `Emacs` to save it)

Now go to `cmt/` and recompile the package.

A look at the header file

```
#include "Kernel/DVAlgorithm.h"
class TutorialAlgorithm : public DVAlgorithm {
public:
  /// Standard constructor
  TutorialAlgorithm( const std::string& name, ISvcLocator* pSvcLocator );
  virtual ~TutorialAlgorithm();       ///< Destructor
  virtual StatusCode initialize();    ///< Algorithm initialization
  virtual StatusCode execute   ();    ///< Algorithm execution
  virtual StatusCode finalize  ();    ///< Algorithm finalization
protected:
private:
};
```

- It inherits from `DVAlgorithm` , which provides the most frequently used tasks in a convenient way.
- The constructor allows to initialise global variables (mandatory!) and to declare options.
- The three methods `initialize()`, `execute()`, `finalize()` control your algorithm. Feel free to add more!

Execute

Let's start with something easy

1. Take muons from the TES location where the particle maker algorithm has put them
2. Loop on them
3. Plot their momentum and p_T
4. Get the primary vertices
5. Plot the muons IP and IP significance

To get data from the TES we have a tool called the `PhysDesktop`

The PhysDesktop

The PhysDesktop is a tool that controls the loading and saving of the particles that are currently used.

- It collects previously made particles
- It produces particles and saves them to the TES when needed

➜ It hides the interaction with the TES

To get the particles and vertices, do

```
const Particle::ConstVector& parts = desktop()->particles();
const LHCb::RecVertex::Container* PVs = desktop()->primaryVertices();
const Vertex::ConstVector& verts = desktop()->secondaryVertices();
```

Practically no-one ever does the latter as one gets the Vertices from the Particles.

Our `execute()` method

```
StatusCode TutorialAlgorithm::execute() {
  debug() << "==> Execute" << endmsg;
  StatusCode sc = StatusCode::SUCCESS ;

  // code goes here
  LHCb::Particle::ConstVector muons = desktop()->particles();
  sc = loopOnMuons(muons);
  if (!sc) return sc;

  setFilterPassed(true);   // Set to true if event is accepted.
  return StatusCode::SUCCESS;
}
```

- We get the particles from the `PhysDesktop` tool
- Then we pass them to a method that we have to write

Our new method

In the header file add:

```
private:
  StatusCode loopOnMuons(const LHCb::Particle::ConstVector&)const ;
```

In the cpp file add:

```
//=============================================================
// loop on muons
//=============================================================
StatusCode TutorialAlgorithm::loopOnMuons(
  const LHCb::Particle::ConstVector& muons)const {

  StatusCode sc = StatusCode::SUCCESS ;

  // code goes here

  return sc ;
}
```

Our new method

In the method add:

```
for ( LHCb::Particle::ConstVector::const_iterator im = muons.begin() ;
      im != muons.end() ; ++im ){
  plot((*im)->p(),  "Muon P",  0., 50.*Gaudi::Units::GeV);  // momentum
  plot((*im)->pt(), "Muon Pt", 0., 5.*Gaudi::Units::GeV );  // Pt
  if (msgLevel(MSG::DEBUG)) debug() << "Mu Momentum: "
                                    << (*im)->momentum() << endmsg ;
}
```

- `LHCb::Particle::ConstVector` is a `typedef std::vector<LHCb::Particle*>`
  - ➜ Hence the non-intuitive `(*im)->momentum()` syntax
- The `plot` method allows to book histograms on demand.
  - It returns a pointer to the histogram that you could also use directly
- There are many units defined in `Gaudi::Units`
- Look at the `Particle` class doxygen

Let's get the primaries

In the method, before the muons loop, add:

```
const LHCb::RecVertex::Container* pvs = desktop()->primaryVertices();
```

In the muons loop add another loop

```
  for ( LHCb::RecVertex::Container::const_iterator ipv = pvs->begin() ;
        ipv != pvs->end() ; ++ipv ){
    double IP, IPchi2;
    if (msgLevel(MSG::DEBUG)) debug() << (*ipv)->position() << endmsg ;
    sc = distanceCalculator()->distance((*im), (*ipv), IP, IPchi2);
    if (sc){
      plot(IP, "IP", "Muon IP", 0., 10.*Gaudi::Units::mm);
      plot(IPchi2, "IPchi2", "Muon chi2 IP", 0., 10.);
      if ( (*im)->pt()>2*Gaudi::Units::GeV)
        plot(IP, "IP_2", "Muon IP for PT>2GeV", 0., 10.*Gaudi::Units::mm
    }
  }
```

- The `distanceCalculator()` is a tool owned by `DVAlgorithm` that allows to make geometry calculations.

Tools!

A look at the Doxygen web page shows that `DVAlgorithm` provides a lot of functionality (not all listed here):

```
IPhysDesktop* desktop() const;
IVertexFit* vertexFitter() const;
IDistanceCalculator* distanceCalculator() const;
IParticleFilter* particleFilter() const;
ILifetimeFitter* lifetimeFitter() const
LHCb::IParticlePropertySvc* ppSvc() const;
ICheckOverlap* checkOverlap() const;
IParticleDescendants* descendants() const;
IBTaggingTool* flavourTagging() const;
StatusCode setFilterPassed(bool);
std::string getDecayDescriptor();
```

We will use some of them.

Done!

- We have our algorithm
  - Don’t forget to compile it
- We have our options
  - Just need to tell the algorithm from where to get the muons
  - They are made behind your back (by the `DataOnDemandSvc`)

```
tutalg.InputLocations = [ "StdLooseMuons" ]
```

This used to be

```
tutalg.addTool( PhysDesktop )}
tutalg.PhysDektop.InputLocation = [" Phys/StdLooseMuons"]
```

This is deprecated. Correct it if you see it!

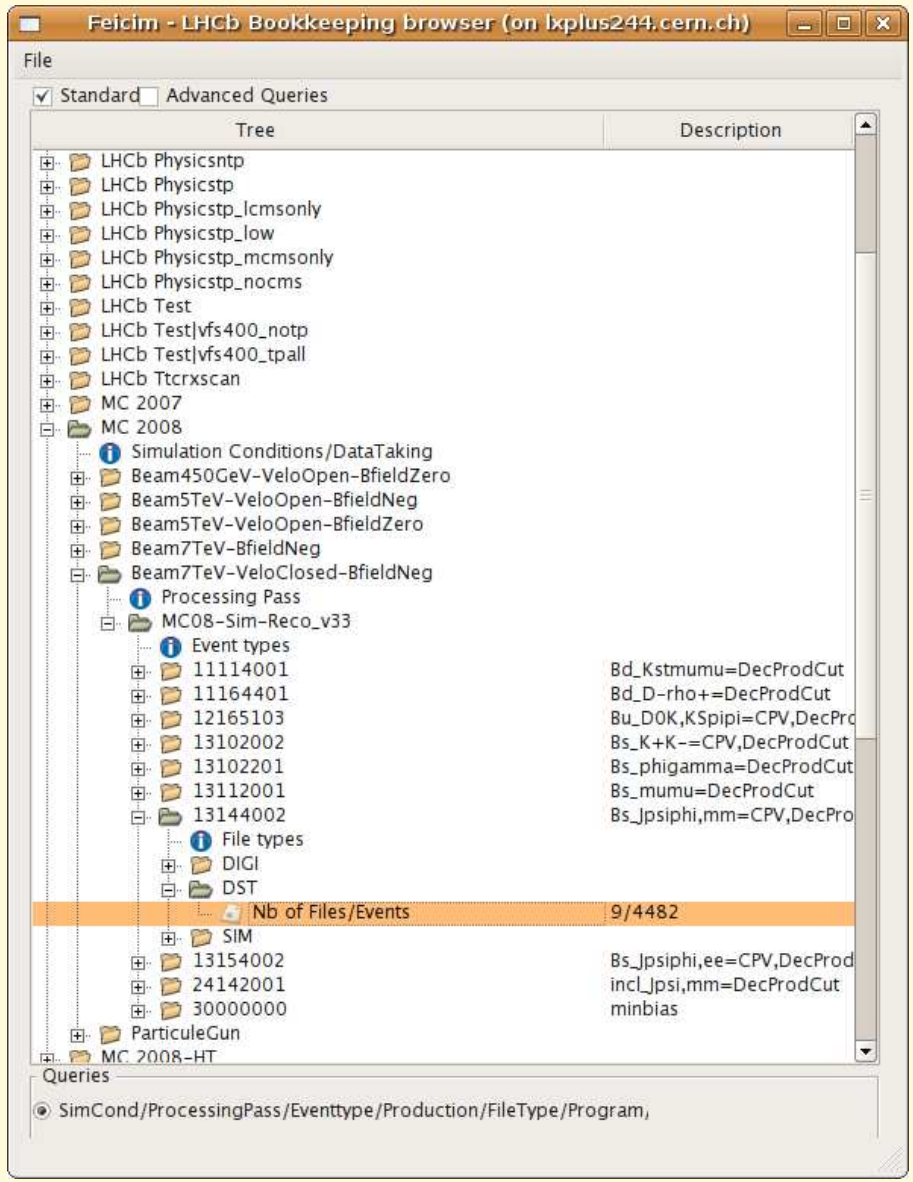
- We can run!
  - We still need to configure our application
  - And we need some data. . .
  - ➜ We can get it from the Bookkeeping database

Feicim

1. In ganga do
   `data = browseBK()`
2. Navigate to MC09, B field on, velo closed, latest version of everything
3. Select some data type with muons
4. Save as python file

➜ This gives you a list of LFNs that you can use to define an `LHCbDataset` in **ganga**.

5. Or you could translate to PFNs



Configure `DaVinci()`

```
from Configurables import DaVinci
DaVinci().HistogramFile = "DVHistos_1.root"    # Histogram file
DaVinci().EvtMax = 1000                        # Number of events
DaVinci().DataType = "MC09"                    # Default anyway
DaVinci().Simulation = True                    # It's MC
DaVinci().UserAlgorithms = [ tutorialseq ]     # our sequence
```

- `DaVinci()` takes care of all the initialisations you don't want to know about.
- It makes different things depending on the type of input data (MC, 2006, 2008 . . .)
- It's **your job** to tell **DaVinci** what you want to do
- It has a lot of other options . . .

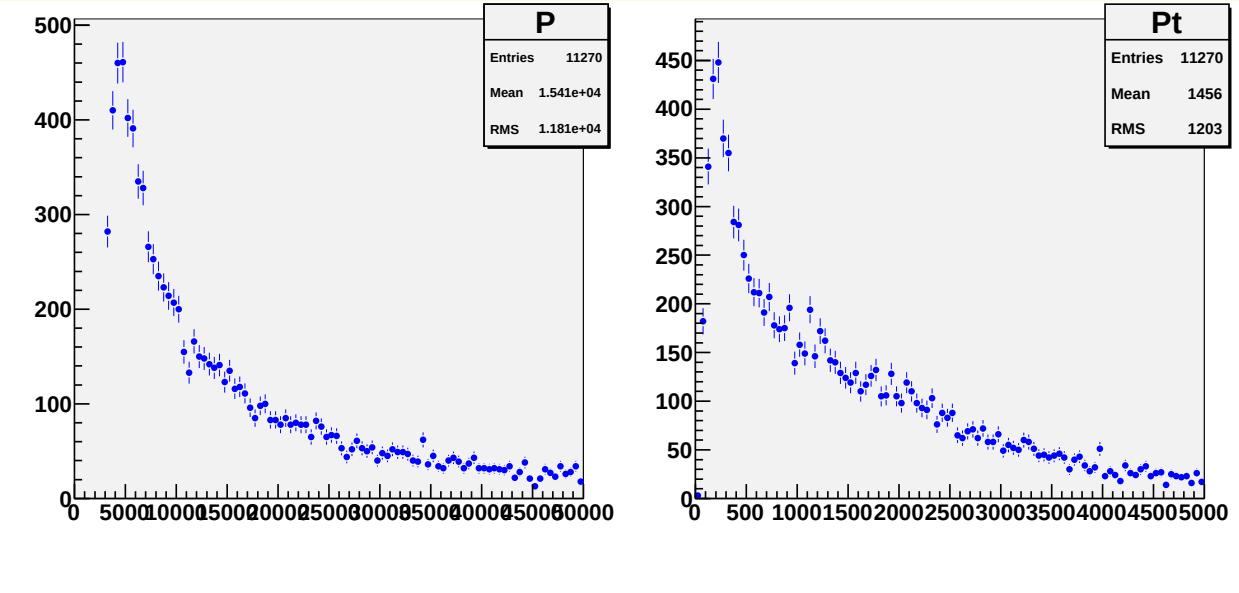
DaVinci()

```
# Application Configuration : sent to LHCbApp and Gaudi
DaVinci().EvtMax              =  -1           # Number of events to analyse
DaVinci().SkipEvents          =   0           # Number of events to skip at beginning for file
DaVinci().PrintFreq           = 100           # The frequency at which to print event numbers
DaVinci().DataType            = ’MC09’        # Data type, can be [’DC06’,’2008’,’2009’, ’MC09’] Forwarded
DaVinci().Simulation          = True          # set to True to use SimCond. Forwarded to PhysConf
DaVinci().DDDBtag             = "default"     # Tag for DDDB. Default as set in DDDBConf for DataType
DaVinci().CondDBtag           = "default"     # Tag for CondDB. Default as set in DDDBConf for DataType
# Input
DaVinci().Input               = []            # Input data. Can also be passed as a second option file.
# Output
DaVinci().HistogramFile       = ""            # Write name of output Histogram file
DaVinci().TupleFile           = ""            # Write name of output Tuple file
DaVinci().ETCFile             = ""            # Name of ETC file
# Monitoring
DaVinci().MoniSequence        = []            # Add your monitors here
# DaVinci Options
DaVinci().MainOptions         = ""            # Main option file to execute
DaVinci().UserAlgorithms      = []            # User algorithms to run.
DaVinci().RedoMCLinks         = False         # On some stripped DST one needs to redo the Track<->MC link
DaVinci().InputType           = "DST"         # or "DIGI" or "ETC" or "RDST" or "DST". Nothing means the in
# Trigger running
DaVinci().L0                  = False         # Run L0.
DaVinci().ReplaceL0BanksWithEmulated = False # Re-run L0
DaVinci().HltType             = ’’            # HltType = No Hlt. Use Hlt1+Hlt2 to run Hlt
DaVinci().HltUserAlgorithms   = [ ]           # put here user algorithms to add
DaVinci().Hlt2Requires        = ’L0+Hlt1’     # Say what Hlt2 requires
DaVinci().HltThresholdSettings = ’’           # Use some special threshold settings, eg. ’Miriam_20090430’
```

Run!

You can now run your job

This will produce a file `DVHistos.root` that you can inspect with root. It contains the four histograms we have created in the algorithm.



Exercises!

Ex. 1: asks you to loop over muons and make some plots

- Everything you need is on the wiki page
- The main difficulty is to figure out what to copy-paste where.
- Don't be afraid to ask if you are unsure

Ex. 2: Extend the algorithm to make a J/ψ (if you have time)

Ex. 3: Make your algorithm more generic: select also a ϕ (if you have time)

➜ Do Ex. 2 and 3 if you plan to develop C++ in **DaVinci**.

Ex. 4: The recommended way of writing a selection

➜ Talk by Juan on Wednesday

Ex. 5: Debugging

Ex. 6: MC truth, Trigger, Tagging, and much more

Ex. 7: More Tuples