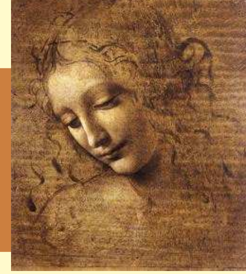
Introduction to DaVinci

- Introduction
  - essentially a reminder from **Gaudi** sessions
- My first `DVAlgorithm`
  - we will loop over muons and plots some quantities

June 2006 Bologna Software Course

Patrick Koppenburg
Imperial College
London

DaVinci Links



- **DaVinci** web page:
  http://lhcb-comp.web.cern.ch/lhcb-comp/Analysis/default.htm
  From there you'll find :
  - Some documentation
  - A "getting started" guide
  - A FAQ
  - The Tutorial page
    - I will add these slides next week.
- Any 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.

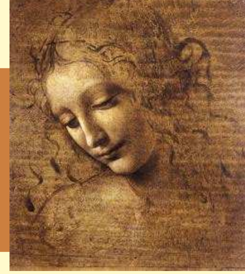
Applications

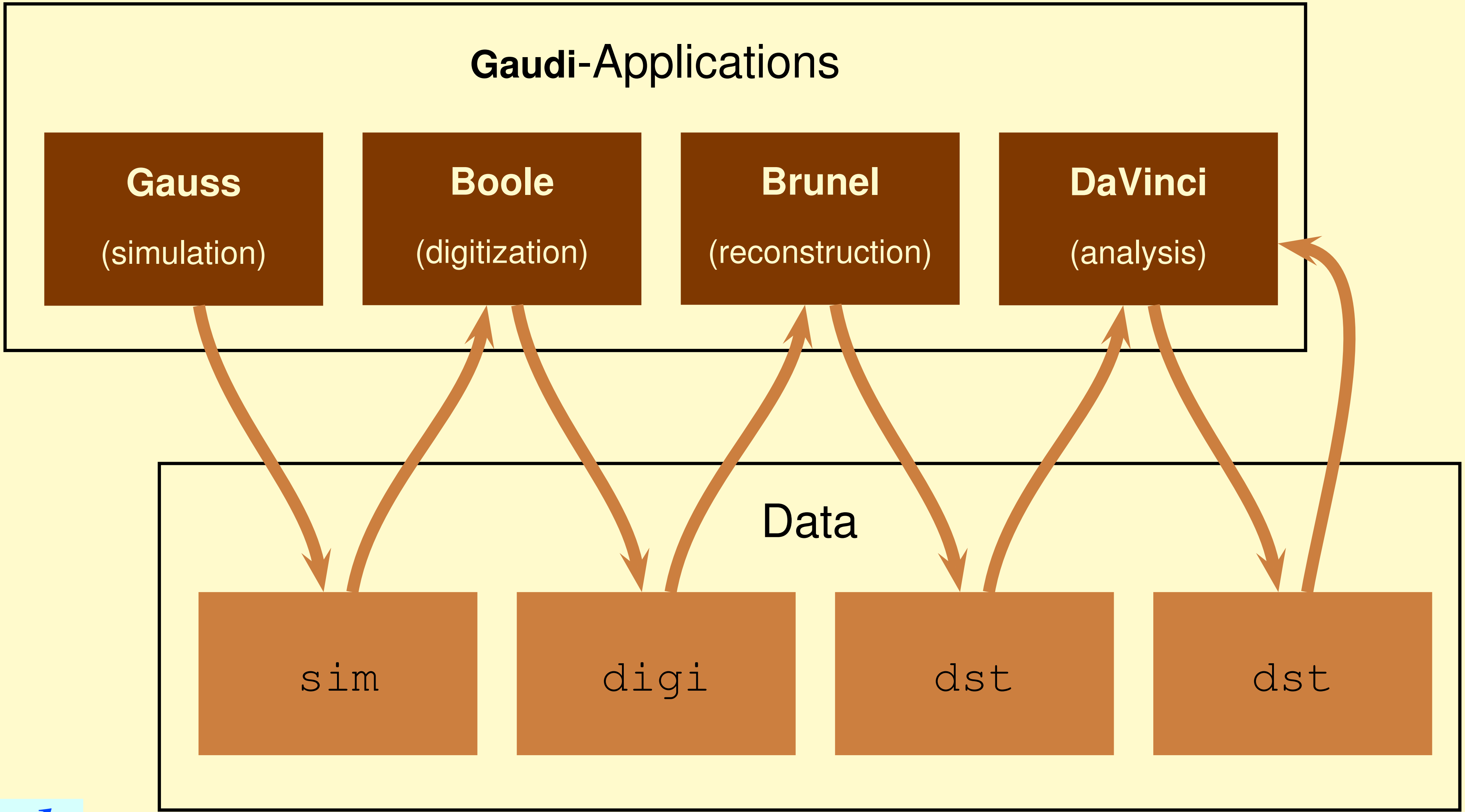
Gaudi-Applications

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

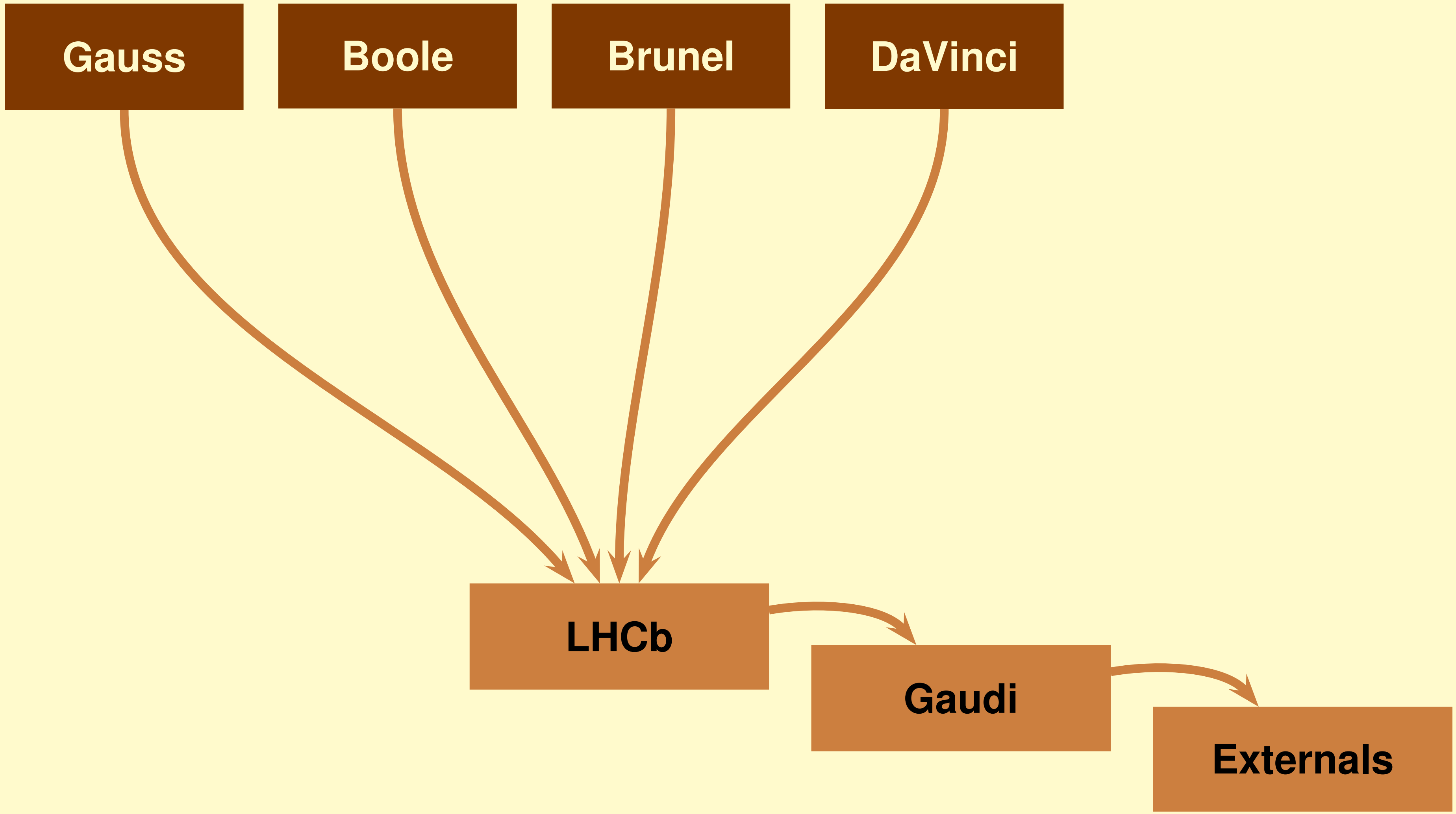
- There are four applications based on **Gaudi**
- They are actually all **Gaudi**-programs
- The only difference are the packages (shared libraries) included
- One could easily build an application that does it all (like in the old **SICB** days. . .)
- Somewhere here **Panoramix** and **Bender** are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

Applications



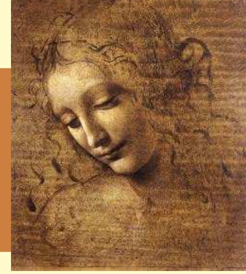


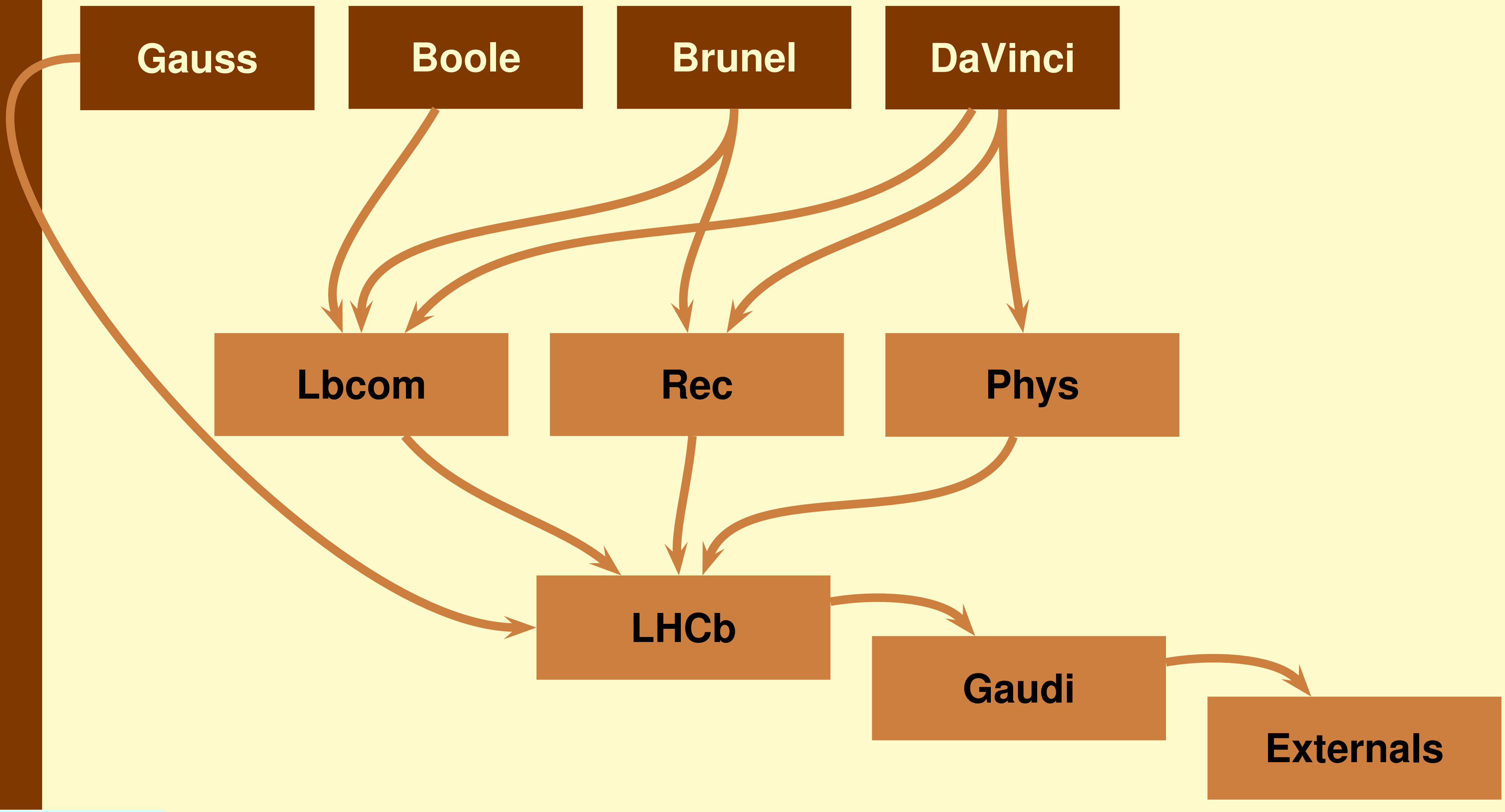
Projects



DC04: 4 applications with many overlaps

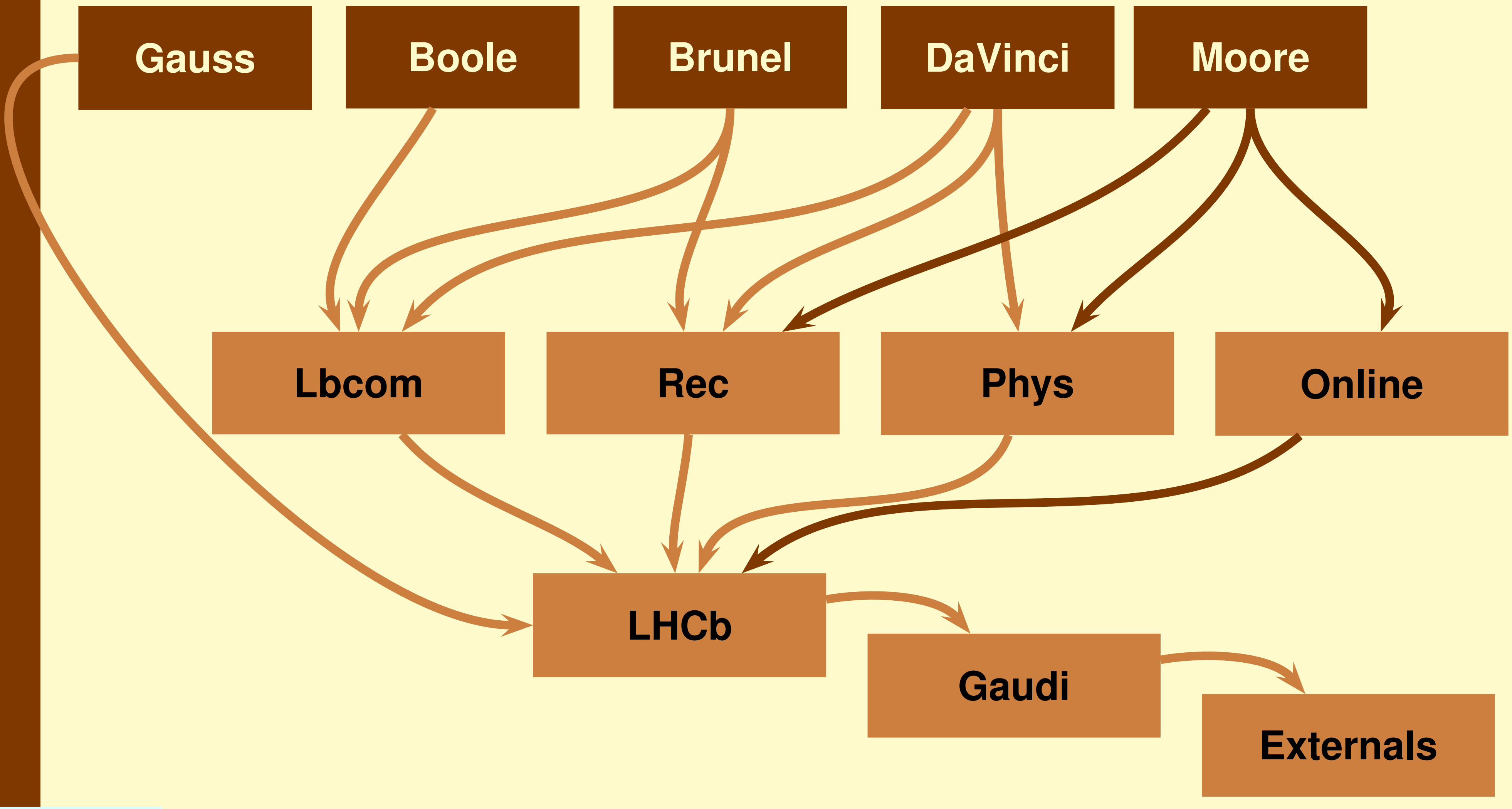
Projects





DC06: 4 applications with no overlaps

Projects



DC06: 5 applications with no overlaps

Packages

A project is a set of packages containing the code necessary to build a shared library and the relevant options.

They all have the subdirectories `cmt`, `doc`, `src` and `options` (sometimes `python`)

See the **Gaudi** tutorial for an explanation of the package structure.

- Packages in **DaVinci**:
  - **`Phys/DaVinci`:** The application and the main options
- Packages in **Phys**:
  - **`Phys/*`:** Physics algorithms and tools (18 packages)
  - **`Tools/*`:** Stripping, utilities
  - **`PhysSel/*`:** Specific decay channel selections
  - **`Tests/*`:** Some tests
- Packages in **LHCb**:
  - **`Event/*`:** Event Model
  - **`Kernel/*`:** Common basic stuff

Physics Packages (`v3r1`)

Basic components:

`Phys/DaVinciKernel/`: Base classes

`Phys/DaVinciFilter/`: Particle filters

`Phys/ParticleMaker/`: Particle makers

`Phys/VertexFit/`: Vertex fitters

`Phys/DaVinciTransporter/`: Transporters

`Phys/DaVinciTools/`: Anything else

`Tools/Utilities/`: Simple utilities

Physics analysis:

`Phys/CommonParticles/`: Standard Particles

`Phys/FlavourTagging/`: Flavour tagging (not yet back)

`Phys/Tampering/`: Tis Tos Tob (not yet back)

`Phys/LoKi*/`: **LoKi**

`Tools/Stripping/`: Stripping tools

MC-truth and test packages

`Phys/DaVinciMCTools/`: MC Tools

`Phys/DaVinciAssociators/`: Associators to MC truth (not yet)

`Phys/DaVinciEff/`: Efficiency algorithms (not yet)

`Phys/DaVinciUser/`: Tests

Disclaimer: Status of DC06

- We are presently rewriting everything . . . and we are not yet completley done.
- We show what we can show
- And try to hide what you don't need to know. . .
  - If you're curious you can check what's in `$ANALYSISROOT/options/BolognaOptions.opts`
- We could have shown you much more with DC04 software, but what's the point?
  - It is obsolete.
  - It's going to disappear by the end of the year.

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.

Get the `Tutorial` package

Get the latest version of the `Tutorial/Analysis` package. You should already have done that with Marco:

```
  > cd $HOME/cmtuser/
  > DaVinciEnv v16r1
  > getpack Tutorial/Analysis v6r0
  > getpack Phys/DaVinci v16r1
  > cd Phys/DaVinci/v16r1/cmt
  > emacs requirements
    ... add: use Analysis v6r0 Tutorial
  > cmt config
  > cmt br make
  > source setup.csh
  > echo $ANALYSISROOT
/afs/cern.ch/.../cmtuser/Tutorial/Analysis/v6r0
  > cd $ANALYSISROOT
```

Start to write the options

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

```
#include "$DAVINCIROOT/options/DaVinciCommon.opts"
#include "$ANALYSISROOT/options/BolognaOptions.opts"
ApplicationMgr.DLLs += { "Analysis" };// Don't forget the DLL
ApplicationMgr.TopAlg += { "GaudiSequencer/TutorialSeq" };
TutorialSeq.Members += { "TutorialAlgorithm" };
```

- `DaVinciCommon.opts` makes (should make) the `Particles` using the `ProtoParticles` available on the DST.
- `BolognaOptions.opts`: Since **DaVinci** is under construction we need some "special" options to ensure that everything works smoothly.
- Then let's start a sequence of algorithms with one algorithm inside.

ProtoParticles?

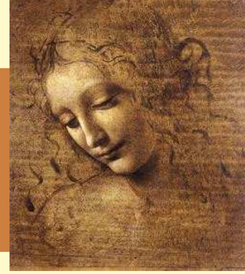
`ProtoParticles`

- 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
  - Track?
  - Calo cluster?
- have a list of PID hypothesis with a probability
- contain the *kinematic* information

You need to assign them a mass and a PID to get the full 4-vector.

$\Rightarrow$ `Particles`

Particles?



- `Particle` = `ProtoParticle`+ one PID *choice*

  $\longrightarrow$ 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 $\mathbf{B} \to \pi\pi$ 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`

Standard Particles

- The Particles are actually already done for you. To ensure that everybody agrees on what is a $\mathbf{K}^{+}$, a π or a $\mathbf{K}^{0}_{\mathbf{S}}$, we have a set of standard particles predefined.
- They are defined in DaVinciCommon.opts
- This is not yet ready for DC06, but you can have a look at the DC04 options here. In the meantime we make Particles from MCParticles.
- All you need to know are the names of the locations: Phys/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 are objects 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`

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)

You can as well re-use Marco’s example

Before you forget it, add the following line to `src/Analysis_load.cpp`:

```
 DECLARE_ALGORITHM(TutorialAlgorithm)
```

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

A look at the header file

```
#include "DaVinciTools/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 t
4. Get the Primary vertices
5. Plot the muons IP and IP significance

To get data from the TES we have a nice 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, just do

- ```
  const ParticleVector& parts =
      desktop()->particles();
  ```
- ```
  const VertexVector& verts =
      desktop()->primaryVertices();
  ```
- ```
  const VertexVector& pvs =
      desktop()->secondaryVertices();
  ```
  ```

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
  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 loop, add:

```
LHCb::PrimVertex::ConstVector pvs = desktop()->primaryVertices();
```

In the loop add another loop

```
    for ( LHCb::PrimVertex::ConstVector::const_iterator ipv =
          pvs.begin() ; ipv != pvs.end() ; ++ipv ){
      double IP, IPE;
      debug() << (*ipv)->position() << endmsg ;
      sc = geomDispCalculator()->calcImpactPar(*(*im), *(*ipv), IP, IPE);
      if (sc){
        plot(IP, "Muon IP", 0., 10.*Gaudi::Units::mm);
        if (IPE>0.) plot(IP/IPE, "Muon IP/error", 0., 10.);
      }
    }
```

- The `geomDispCalculator()` 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;
IGeomDispCalculator* geomDispCalculator() const;
IParticleFilter* particleFilter() const;
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
- We can run!
  - But we need some data. . .
  - We can get it from the Bookkeeping database

Bookkeeping!

Access it at http://lhcbdata.home.cern.ch/lhcbdata/bkk/

In this case we want the most recent DC06 data.

1. Click “Dataset Search”
2. Select “Configuration = "DC06 - v1-lumi2"”
3. Select “Event type = Incl_b”
4. Select “Datasets replicated at CERN”
5. Select “Datatype = SIM 1”
6. Select “Step 1 = Gauss v24r6” (the latest)
7. Submit
8. You get a new page. Click on the Gaudi logo
9. You get a new window. Paste the contents in your options

Run!

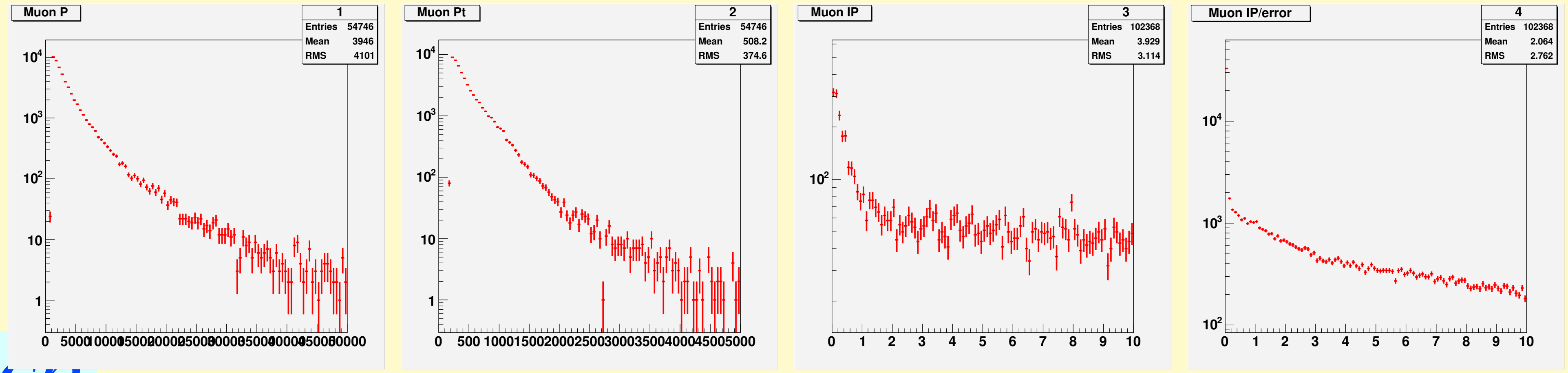
Before you run, add to your options the line

```
HistogramPersistencySvc.OutputFile = "DVHistos.root";
```

You can now run your job with the command

```
> DaVinci $ANAYSISROOT/options/MyOptions.opts
```

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!

- Let's go for the exercises

  Ex. 1 asks you to try by yourself what we just showed