Introduction to the bookkeeping and to DaVinci Software for Run (1+2)

Iván Cambón Bouzas

IGFAE, Universidade de Santiago de Compostela, Spain
LHCb Collaboration

Who am I?

- My name is Iván and I am a 2nd year PhD student from Santiago de Compostela.

- I'm working at the spectroscopy of the D_{sJ} resonances and their radiative decays to the D_s^+ meson.

- The main goal is to study their production as function of multiplicity

- My analysis is part of the Ions and Fixed Target (IFT) working group

- When I started I had my nTuples made for the analysis. However, a lot of changes were needed so I made all of them from scratch.

- I am not an expert, but I will share you all my experience as far as I know

- And finally, do not hesitate to ask any question!

- My Gitlab is at the bottom left of the slides if you are interested

What is the main goal of this lecture?

↓

Running a minimal DaVinci job locally for Run (1+2)

↓

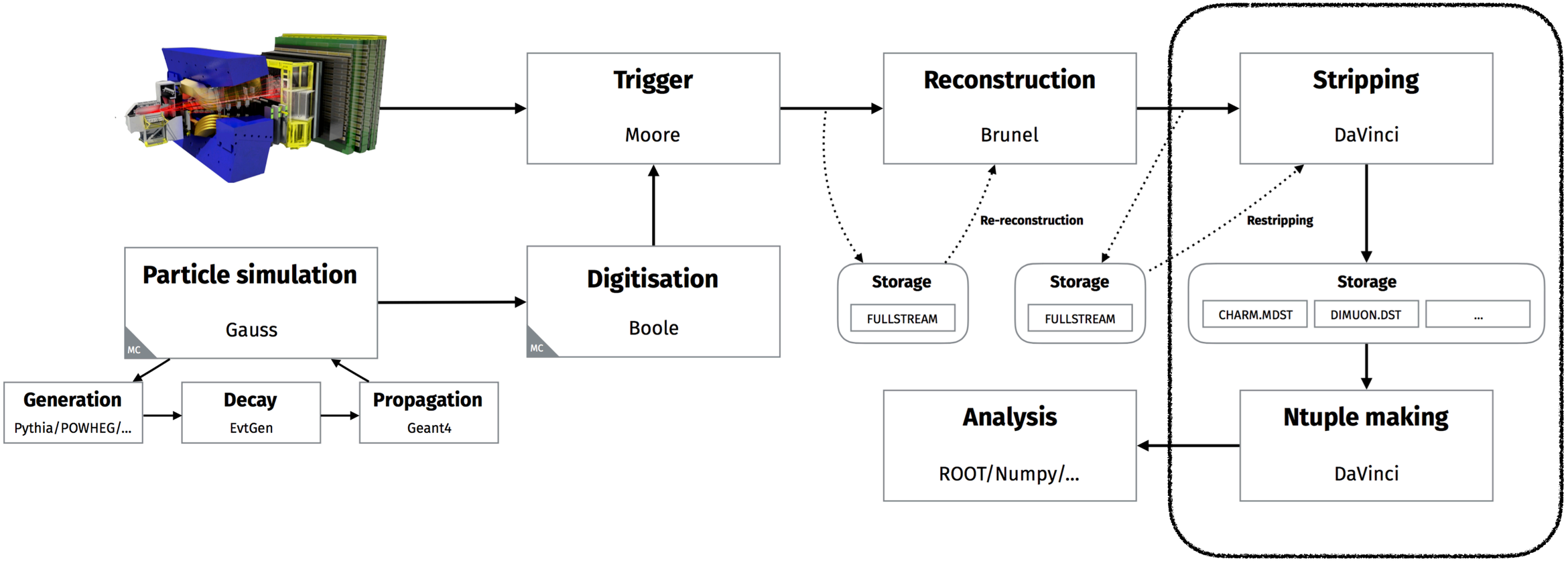
In other words

↓

Doing our first Run (1+2) nTuple

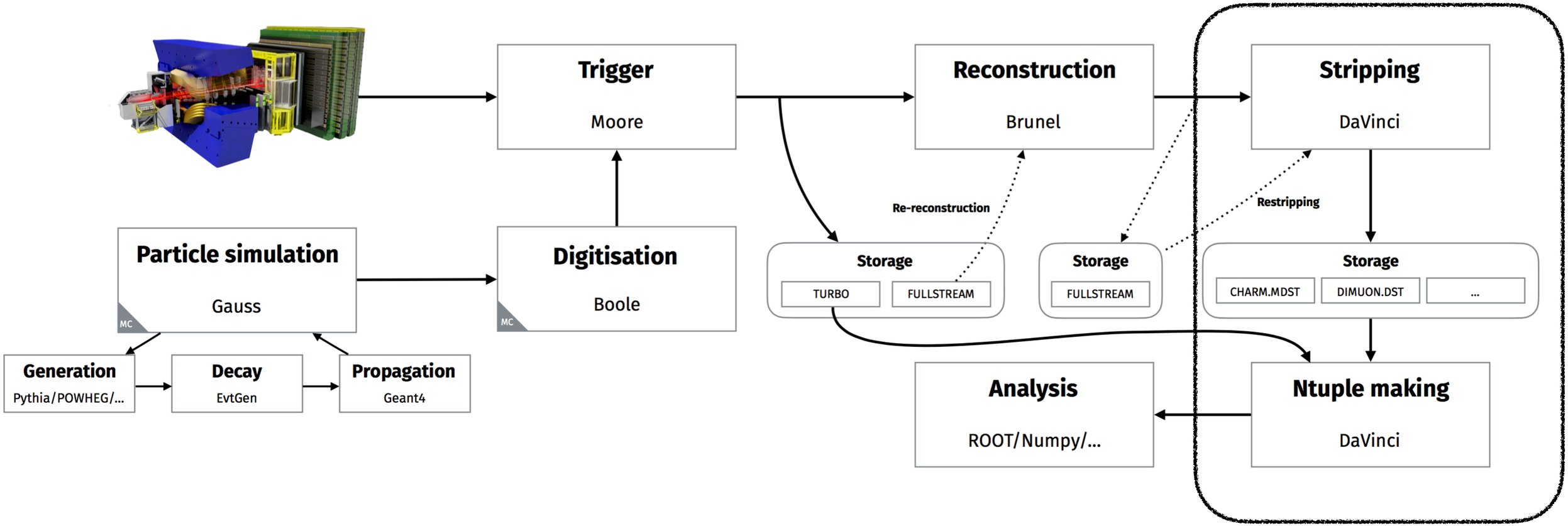
Firstly, let's start with some concepts

LHCb data flow for Run 1



Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

LHCb data flow for Run 2



The key point in this lecture → **The Stripping**

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

What is the Stripping? (In a nutshell)

Fully reconstructed event

(Particles, PVs, etc

→ **Offline Selection**

Decay candidates reconstruction

$B^+ \to J/\psi(\to \mu^+\mu^-)K^+$

This offline selection is known as the **stripping line**

The stripping lines can be found in the STRIPPING project

And it is made by the DaVinci framework 

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Data storage (In a nutshell)

Stripped particles and events

Note
More formats are available

.DST files
Everything is stored
150 kB/event

✓ Underlying event

✗ Too heavy

MC preferred

Less disk space ✓

Less info ✗

.mDST files
Only candidates are stored
50 kB/event

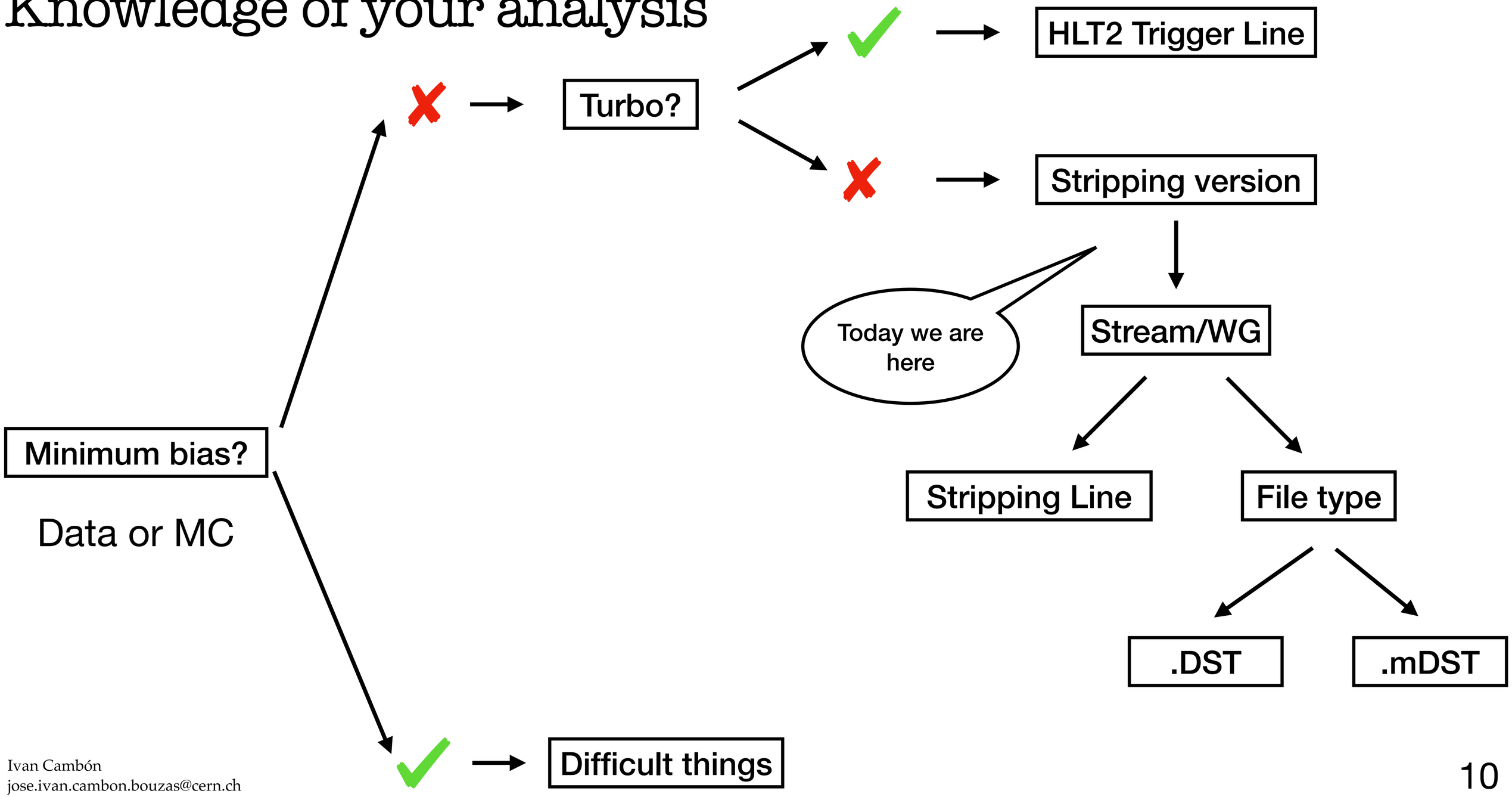
Data preferred

For Run 2 there is also **Turbo data** → No stripping line, only HLT2 line

Moreover, we have minimum bias data → No stripping line, no HLT2 line.
1 reconstructed track requirement

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Knowledge of your analysis



Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Now, we are ready to start our task

Do, locally and from scratch, a nTuple which has the information for analysing the following decay

$$D^{*+} \to D^0(\to K^+K^-)\pi^+$$

For that, use the MC sample with EventType 27163002 and the stripping line `D2hhPromptDst2KKLine`

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Step 1

Finding data in the Bookkeeping

Data managing in LHCb

DIRAC
THE INTERWARE

Computer Resources
GRID

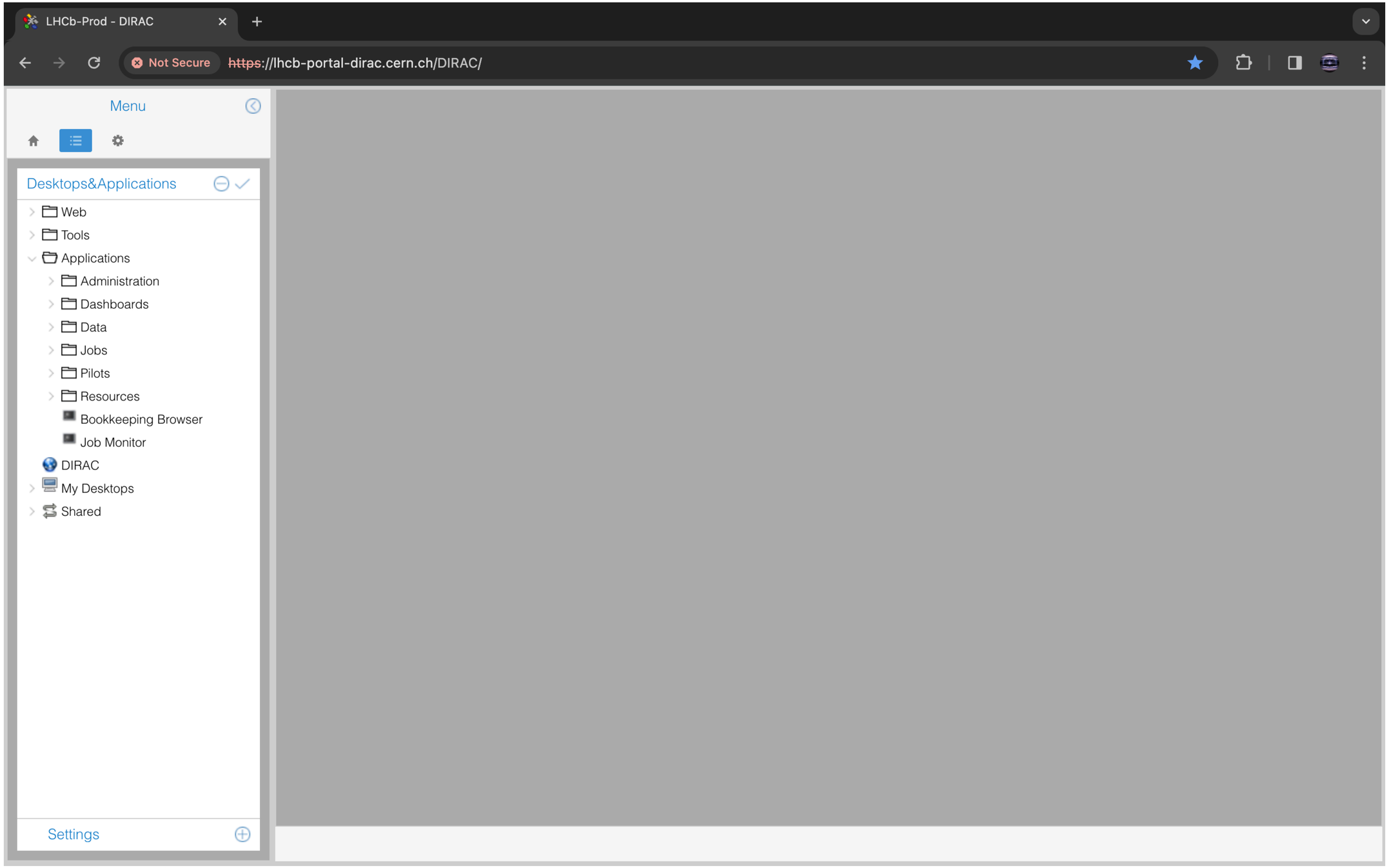
Storage Resources
EOS

Catalog Resources
Bookkeeping

Internet Portal

DIRAC

Grid Certificate required



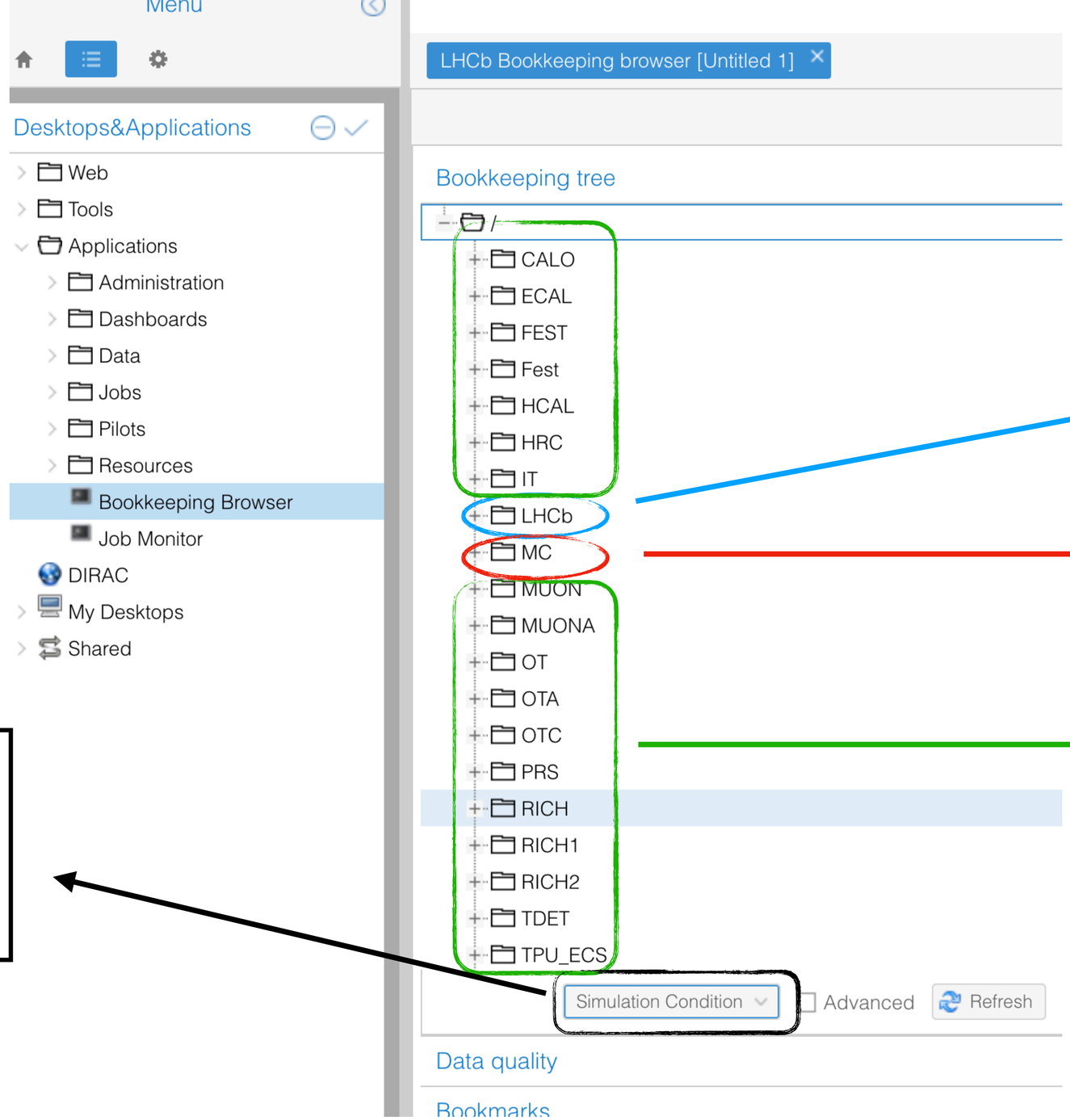
Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Remarkable applications in DIRAC

- **Job Monitor:** Allows us to check the status of any job that we are running on the Grid
- **Production Request:** Allows us to check any production that yourself or your WG is running (MC simulations or data processing)
- **Transformation Monitor:** Gives us information about a given production
- **Step Manager:** Gives us information about the steps that a given production followed
- **Bookkeeping Browser** *aka* **bkk:** Allows us to explore the data and MC produced as well as its location on the Grid

Today we will use this

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch



LHCb Data

MC Simulations

Subdetector data

Bookkeeping Tree

Two important options:

- **Simulation Condition**
- **EventType**

Some hands-on

Let’s find our MC simulation DSTs

Bkk Cheat Sheet for MC simulations

1. Change the Bookkeeping Tree from **Simulation Condition** to **EventType**
2. Click the **MC** folder
3. Click on the folder that corresponds to the year of your simulation (In our case 2016)
4. Look for the folder that has your decay. All folders are named as a number, the EventType. In our case it is **27163002** (We can use control+F or command+F to search it directly) and click on it
5. Click on the polarity and energy that you want (for this lecture **6500 GeV MagUp**)
6. Click on the folder of your simulation version (for this lecture the **Sim09c)**
7. Select the Trigger conditions (for this lecture there is only one)
8. Select the reconstruction version (for this lecture **Reco16** and after **Turbo03**).
9. Choose the stripping version (for this lecture **Stripping28r1NoPrescalingFlagged**)
10. Click on **ALLSTREAMS.DST.** The whole set of DST files will be displayed. To save the list of names, click on **Save** (at right corner) and save it as python file (*.py)

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Bkk Cheat Sheet for Official Run (1+2) Data

My method for CHARM .MDST files

1. Change the Bookkeeping Tree from **Simulation Condition** to **EventType**
2. Click the **LHCb** folder
3. Loads of folders will appear. Each of them correspond to several types of collisions and calibration samples. For pp collisions, the folders are name **CollisionXX** being XX the year. For this tutorial, click on **Collision18**
4. Folders for each stream will be shown. In this example, we are interested on **Full stream** so we click on it
5. Click on the polarity, energy and detector condition that you want (for this example **MagUp 6500 GeV VeloClosed**)
6. Now go to **Real Data** and **Reco18** (for other years)
7. Click on the stripping version that you need (for us **Stripping34)**
8. Tons of folders will appear, most of them with *AnaProd* prefix. We have to scroll until we see something like *BHadron.MDST.* Here we click on the stream that we want, in our case **CHARM.MDST**
9. The whole set of files will be displayed. To save the list of names, click on **Save** (at right corner) and save it as python file (*.py)

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Downloading a file

- We have found the MC sample in the bkk and we got the .py with the locations
- In order to download one file we have to

1. Init the proxy in lxplus → `lhcb-proxy-init`
2. Copy the file name with its LFN location (`DST_file`) from the .py file
3. Run the following command → `lb-dirac dirac-dms-get-file DST_file`

DO NOT RUN THIS IN THE LECTURE!!!!
WE CAN LAG LXPLUS FOR EVERYBODY

The file can be found here: "`/afs/cern.ch/user/j/jcambonb/public/DaVinci_Run12_Lessons_2023/DST_files/`"

Step 2

Running a minimal DaVinci job for Run (1+2)

But first, some concepts again 😭

LHCb Software Run (1+2) overview

Transient Event Store (TES)

- Per event file system
- Organices the information of the event in the DST files

Event Loop

- Events are almost completely independent
- We process them one by one
- We do not hold them all in memory at once

Gaudi Framework

Algorithms and Tools

C++ functions for common tasks

- Filters
- Vertex fitters
- Particle containers

Several projects are based on Gaudi.
One of the most important → **DaVinci**

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Why DaVinci?

Firstly, because this thing is in charge of the Stripping

And at user level?

Allows us to pass from DST to ROOT files which are

- *Smaller → only the information that we need is stored*
- *Are nicely to work with → python cannot read DSTs*

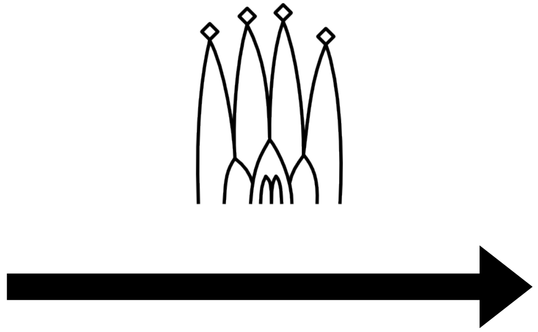
But some comments are needed

We can do this job with Gaudi

DST files

All event information stored

Particles, hits, tracks, etc.



C++ Gaudi or GaudiPython

ROOT files (nTuples)

Only particle decay information is stored i.e

$$B^+ \to J/\psi(\to \mu^+\mu^-)K^+$$

PROS

- ✓ More control of what we are doing
- ✓ Work with minimum bias data
- ✓ We can break the Event Loop → Event mixing

CONS

- ✗ Really complex
- ✗ More knowledge of the software is required
- ✗ Computational time

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

But, if we have stripping line or Turbo, DaVinci will be preferred

DST files

All event information stored

Particles, hits, tracks, etc.

C++ or python DaVinci

ROOT files (nTuples)

Only particle decay information is stored i.e

$B^+ \to J/\psi(\to \mu^+\mu^-)K^+$

PROS

- ✓ Easier and intuitive to work with
- ✓ Most of the job is done internally by its classes
- ✓ You do not complicate your life

CONS

- ✗ Some control is partially lost
- ✗ It only can run over stripped or turbo data
- ✗ We cannot break the Event Loop

The thing is that
almost everything comes from stripping or Turbo
So, we are going to learn how to use DaVinci

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

How do we run DaVinci?

- To run any LHCb environment we have to use (on terminal)

```
lb-run
```

- For DaVinci we have to add the following

```
lb-run DaVinci/v46r4
```

- Where v46r4 is the version (the latest one). For Run (1+2) we have to use versions previous to **v50**
- **What version we use?** Some experts recommend the latest while others the one used for the stripping process of your data. Whatever we use, we have to be <u>self-consistent</u>
- The next step is initialise the Event Loop. For that we add the following

```
lb-run DaVinci/v46r4 gaudirun.py
```

- Finally, we add the *option file* which is a python script where we specify the algorithms that we want to run

```
lb-run DaVinci/v46r4 gaudirun.py options.py
```

- After typing this on terminal, DaVinci will start to run whatever we put in the options file

Now, it’s time to hands-on staff

Let’s make our first nTuple

First steps: The `DecayTreeTuple` class

- The key class for creating nTuples for stripped data is the **`DecayTreeTuple`**
- What it does?
  - ✤ Runs algorithms to get information from the particles of a given TES location
  - ✤ Stores this information in a ROOT `Tree` object called *DecayTree*
- Let's take a view of how we define it in the option file

Decay Descriptor

This can be checked in the stripping line documentation

The notation is important

```
# Important imports
from Configurables import DecayTreeTuple

# Stream and stripping line where our particles are
stream = "AllStreams"
line = "D2hhPromptDst2D2KKLine"

# DecayTreeTuple definition
dtt = DecayTreeTuple("DstDKKPiTuple")
dtt.Inputs = ["/Event/{0}/Phys/{1}/Particles".format(stream, line)]
dtt.Decay  = "[D*(2010)+ -> (D0 -> K- K+) pi+]CC"
```

TES Location

Where the particles are

We need to now the stream and the stripping line

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

First steps: Configuring DaVinci

- After the previous lines, we have to set up DaVinci with some attributes

- In our case, we have to use the following

```
from Configurables import DaVinci

# DaVinci options
DaVinci().UserAlgorithms += [dtt]                # Add algorithms to the nTuple
DaVinci().InputType  = "DST"                     # DST or mDST
DaVinci().TupleFile  = "basic_ntuple.root"       # Name of the ntuple
DaVinci().PrintFreq  = 1000                      # Pritting options
DaVinci().DataType   = "2016"                    # Year
DaVinci().Simulation = True                      # MC or Data
DaVinci().Lumi       = not DaVinci().Simulation # Only True for Data
DaVinci().EvtMax     = -1                        # -1 means all events inside the DST file
# Magnet Conditions
DaVinci().CondDBtag  = "sim-20170721-2-vc-md100"
DaVinci().DDDBtag    = "dddb-20170721-3"
```

Detector conditions (i.e Magnet polarity)
REALLY IMPORTANT TO KNOW !!!

First steps: Read a local DST file and running the code

- Finally, we have to specify the DST file that we are going to run DaVinci over
- In my case, I have the file in a directory above the option file called `DST_files`

```
from GaudiConf import IOHelper

# Read local nTuples
dst_path = "../DST_files/"
dst_name = "00070793_00000040_7.AllStreams.dst"

IOHelper().inputFiles([dst_path+dst_name],
                      clear=True)
```

- Now, we are ready to run the option file

```
lb-run DaVinci/v46r4 gaudirun.py options.py
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

First steps: The output

- While running the option file, loads of messages will appear on the terminal
- We have to be careful about what warnings and errors can raise

```
TransportSvc          INFO Reset the static pointer to DetDesc::IGeometyrErrorSvc
DataOnDemandSvc       INFO Handled "DataFault" incidents: 34846/20747/233775(Alg/Node/Total).
ToolSvc               INFO Removing all tools created by ToolSvc
DaVinciInitAlg....SUCCESS Booked 4 Histogram(s) : 1D=4
DaVinciInitAlg....SUCCESS Exceptions/Errors/Warnings/Infos Statistics : 0/0/2/0
DaVinciInitAlg....SUCCESS  #WARNINGS   = 1        Message = 'Delta Memory for the event exceeds 3*sigma'
DaVinciInitAlg....SUCCESS  #WARNINGS   = 8        Message = 'Total Memory for the event exceeds 3*sigma'
ToolSvc.L0DUCon...    INFO --------------- TCK = 0x160F------------------
ToolSvc.L0DUCon...    INFO The configuration 0x160F matches the hardware limitations
ToolSvc.L0DUCon...    INFO - Usage : #Channels   : 17 [53.1% ]
ToolSvc.L0DUCon...    INFO - Usage : #Conditions : 32 [25.0% ]; order : OK ; reported  : 32/32
ToolSvc.L0DUCon...    INFO - Usage : #Conditions/type (max)  :[Spd(Mult) : 7/8]
ToolSvc.L0DUCon...    INFO - Info    : the condition 'SumEtPrev<250' relies on BX=[-1]
ToolSvc.L0DUCon...    INFO                 TCK = 0x10000
TimingAuditor.T...    INFO ----------------------------------------------------------------------------------------------------------------
TimingAuditor.T...    INFO This machine has a speed about   2.94 times the speed of a 2.8 GHz Xeon.
TimingAuditor.T...    INFO Algorithm                                (millisec) |    <user> |   <clock> |      min       max    sigma | entries | total (s) |
TimingAuditor.T...    INFO ----------------------------------------------------------------------------------------------------------------
TimingAuditor.T...    INFO EVENT LOOP                                          |     3.344 |     4.499 |    0.665    4870.9    92.54 |    2769 |    12.458 |
TimingAuditor.T...    INFO  DaVinciEventSeq                                    |     2.726 |     3.937 |    0.400    4870.6    92.54 |    2769 |    10.903 |
TimingAuditor.T...    INFO   DaVinciInitAlg                                    |     0.043 |     0.060 |    0.050       1.3     0.03 |    2769 |     0.169 |
TimingAuditor.T...    INFO   FilteredEventSeq                                  |     2.654 |     3.861 |    0.330    4870.5    92.54 |    2769 |    10.692 |
TimingAuditor.T...    INFO    DaVinciEventInitSeq                              |     0.014 |     0.008 |    0.008       0.0     0.00 |    2769 |     0.025 |
TimingAuditor.T...    INFO     PhysInitSeq                                     |     0.000 |     0.002 |    0.002       0.0     0.00 |    2769 |     0.007 |
TimingAuditor.T...    INFO     AnalysisInitSeq                                 |     0.000 |     0.002 |    0.002       0.0     0.00 |    2769 |     0.007 |
TimingAuditor.T...    INFO    DaVinciAnalysisSeq                               |     2.636 |     3.847 |    0.317    4870.5    92.54 |    2769 |    10.654 |
TimingAuditor.T...    INFO     DaVinciUserSequence                             |     2.632 |     3.840 |    0.311    4870.5    92.54 |    2769 |    10.634 |
TimingAuditor.T...    INFO      DstDKKPiTuple                                  |     0.711 |     1.784 |    0.003    4863.5    92.42 |    2769 |     4.942 |
TimingAuditor.T...    INFO     MonitoringSequence                              |     0.000 |     0.002 |    0.002       0.1     0.00 |    2769 |     0.007 |
TimingAuditor.T...    INFO   LumiSeq                                           |     0.014 |     0.006 |    0.006       0.1     0.00 |    2769 |     0.019 |
TimingAuditor.T...    INFO    EventAccount                                     |     0.003 |     0.002 |    0.002       0.1     0.00 |    2769 |     0.007 |
TimingAuditor.T...    INFO * createODIN                                        |     0.444 |     0.433 |    0.042       4.4     0.52 |    2769 |     1.200 |
TimingAuditor.T...    INFO *     AllStreams_PsAndVsUnpack                      |     0.136 |     0.153 |    0.003       6.5     0.43 |   28754 |     4.422 |
TimingAuditor.T...    INFO *       UnpackRecVertex                             |     1.114 |     1.108 |    0.068       9.2     0.96 |    1543 |     1.710 |
TimingAuditor.T...    INFO *         unpackFittedVeloTracks                    |     1.023 |     1.062 |    0.051       8.9     0.95 |    1543 |     1.640 |
TimingAuditor.T...    INFO *      L0DUFromRaw                                  |     0.506 |     0.348 |    0.237       4.1     0.43 |      79 |     0.028 |
TimingAuditor.T...    INFO *      Hlt1DecReportsDecoder                        |     0.379 |    10.089 |    0.030     793.8    89.30 |      79 |     0.797 |
TimingAuditor.T...    INFO *      Hlt2DecReportsDecoder                        |     9.367 |    10.189 |    0.196     787.0    88.52 |      79 |     0.805 |
TimingAuditor.T...    INFO ----------------------------------------------------------------------------------------------------------------
NTupleSvc             INFO NTuples saved successfully
ApplicationMgr        INFO Application Manager Finalized successfully
ApplicationMgr        INFO Application Manager Terminated successfully
```

Sequence of algorithms used and entries that passed them

IF THIS APPEARED, IT WENT FINE

But, how can we know the ConDBtags and DDDBtags?

Let’s do some hands-on with the Dirac Portal

Cheat Sheet for getting the CondDBtags

The old way (for data and MC)

1. Let's come back to the bkk path where our DSTs are. The names that appear are like `/lhcb/MC/2016/ALLSTREAMS.DST/00070793/0000/00070793_00000002_7.AllStreams.dst`. The production ID is `00070793`

2. We go to the **Transformation Monitor** app and insert this number in the field **ProductionID(s)** and click on **submit**

3. The production will appear on the screen. Let's do right click on it and we select **Show Request**

4. A **Production Request manager** window will spawn. We do right click on it and we select **View.** Now all the steps will appear. The tags are labelled as **DDDB** and **Condition DB**

The cooler way (for MC)

1. On the terminal, init the lhcb proxy: `lhcb-proxy-init`

2. Run the following command:

```
lb-dirac dirac-bookkeeping-decays-path <eventtype>
```

Where `<eventtype>` is the number that we used in the bkk to get the MC files, in our case `27163002`

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Step 3

Tunning our DaVinci option file

Adding branches to your `DecayTreeTuple`

- By default, the `DecayTreeTuple` only puts information about the particle which is the head of the decay

- Usually it is important to have info of the daughters and we do it with the `addBranches` method. Let's add this to the options file

```
# DecayTreeTuple definition
dtt = DecayTreeTuple("DstDKKPiTuple")
dtt.Inputs = ["/Event/{0}/Phys/{1}/Particles".format(stream, line)]
# Now we have to put ^ in each daughter particle
dtt.Decay  = "[D*(2010)+ -> ^(D0 -> ^K- ^K+) ^pi+]CC"
# We put the branches as a dictionary. Each key will be the new branches for each daugther particle
dtt.addBranches({"Dst": "[D*(2010)+ -> (D0 -> K- K+) pi+]CC",
                 "Dz" : "[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC", # For intermidiate resonances, we put the ^ before the parenthesis
                 "Kmi": "[D*(2010)+ -> (D0 -> ^K- K+) pi+]CC",
                 "Kpl": "[D*(2010)+ -> (D0 -> K- ^K+) pi+]CC",
                 "pi" : "[D*(2010)+ -> (D0 -> K- K+) ^pi+]CC"})
```

- After running we will have more branches in the TTree object, and its names will be the keys of the defined python dictionary

More information for our nTuples: `TupleTools` I

- The key aspect of nTupling is how can we make our data files more complete
- For that we have `TupleTools` which are algorithms that we can add to the `DecayTreeTuple`.
- This algorithms will compute fancy features for our particles (kinematics, PID or even more complex ones) and they will be included in the nTuples
- The list of `TupleTools` can be found on this gitlab and they are divided in several packs

  - `DecayTreeTuple`: General tools
  - `DecayTreeTupleANNPID`: NeuralNet-based PID
  - `DecayTreeTupleDalitz`: Dalitz analysis
  - `DecayTreeTupleJets`: Jets analysis
  - `DecayTreeTupleMC`: MC level information
  - `DecayTreeTupleMuonCalib`: Muon calibration
  - `DecayTreeTupleReco`: Reconstruction-level info
  - `DecayTreeTupleTracking`: Tracking info
  - `DecayTreeTupleTrigger`: Trigger information

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

More information for our nTuples: TupleTools II

- To add TupleTools to our option file we have several methods

1. Through a list and the method ToolList

```
tupletools_list =  ["TupleToolEventInfo",
                    "TupleToolANNPID",
                    "TupleToolGeometry",
                    "TupleToolKinematic"]

dtt.ToolList = tupletools_list
```

2. Through addTupleTool method

```
# 2nd method -> adding tupletools one by one: addtupl
track_tool = dtt.addTupleTool("TupleToolTrackInfo")
track_tool.Verbose = True  # Some tupletools have spe
```

Some attributes of the TupleTool can be changed

3. Through addTupleTool over a particle

```
Dz_ct  = dtt.Dz.addTupleTool("TupleToolPropertime")
```

DO NOT ADD THIS LINE YET
Some comments are need

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Fun with LoKi functors

- Sometimes, `TupleTools` are not enough to get some new features. For that we have **LoKi functors**

- These are C++ classes that can compute properties of the current decay. They are used in the **Stripping Lines** to specify the selection that it will do

- There are plenty of LoKi functors and a short list of them can be found on this two links

LoKi functors I and LoKi functors II

- Obviously we can add some of them to our option file. Here is an example

```
# LoKi functors addition
dtt_Dst_LoKi = dtt.Dst.addTupleTool("LoKi::Hybrid::TupleTool/dtt_Dst_LoKi")
dtt_Dst_LoKi.Variables = {"ETA": "ETA",
                          "PHI": "PHI"}
```

- Where we add to the nTuples the pseudorapidity add the azimutal angle of the D^{*+}

How to use a GaudiSequencer

- Some TupleTools need specific requirements to work properly
- For example, the TupleToolPropertime needs the mother particle to come from the PV
- For that we have to check that the events that we want to process have a PV, and for that we have to use a LoKi_VoidFilter, which is not a DecayTreeTuple algorithm
- To run this two algorithms consecutively and dependently, we use a class called GaudiSequencer. To apply it, some changes are needed to the option file

```
Dz_ct  = dtt.Dz.addTupleTool("TupleToolPropertime")

pv = LoKi__VoidFilter("hasPV", Code="CONTAINS('Rec/Vertex/Primary')>0")
gs = GaudiSequencer("myseq")
gs.Members += [pv, dtt]  # The VoidFilter must be made first. The DecayTreeTuple has to be in the end

# DaVinci options
DaVinci().UserAlgorithms += [gs]                # Instead of the DecayTreeTuple, we put the sequencer
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Step 4

Some advanced features for nTupling

Catalogs

- For working with DST files there is no need to download them from the grid. We can access them directly with `Catalogs`

- For using them, firstly we have to create a catalog file. We have to copy the *.py (we call the copy `testcatalog.py`) that we got from the bkk, remove all LFNs except for a couple and run the following command.

```
lb-dirac dirac-bookkeeping-genXMLCatalog --Options=testcatalog.py --
Catalog=myCatalog.xml
```

- Now, in the option file we have to substitute the `IOHelper` class by the following

```
from Gaudi.Configuration import FileCatalog
FileCatalog().Catalogs = ["xmlcatalog_file:/catalogs/myCatalog.xml"]
```

The `DecayTreeFitter` tool I

- While analysing, you can acquire new knowledge of your decay that can be useful to add in the reconstruction process to improve it. This knowledge is known as **constraints**
- In order to add this constraints, a (re)fit of the tracks should be made. And to do it ee have the `DecayTreeFitter` algorithm
- **Note**: I am not an expert on using this since in my analysis
  - The reconstruction of $D_s^+ \to K^+K^-\pi^+$ decay made by the stripping is almost excellent
  - Photons are complicated to work with in this cases (there is no tracking for photons)
- Therefore, I will follow entirely the steps explained in the <u>First Analysis Steps</u> section
- We have two ways of applying the `DecayTreeFitter` algorithm

The DecayTreeFitter tool II

- Through the TupleToolDecayTreeFitter

```
"""
We apply the vertex refit through the TupleToolDecayTreeFitter
"""

dtt.Dst.addTupleTool('TupleToolDecayTreeFitter/ConsD')
dtt.Dst.ConsD.constrainToOriginVertex = True            # Vertex contrain
dtt.Dst.ConsD.Verbose                 = True            # All the information
dtt.Dst.ConsD.daughtersToConstrain    = ["D0"]          # Do a constraint to D0 -> K+K- fit
dtt.Dst.ConsD.UpdateDaughters         = True            # Update the tracks for all daughters
```

✔ Easier

✘ Few info

- Through the LoKi functors

```
"""
We apply the vertex refit through the LoKi Functors
"""
from Configurables import LoKi__Hybrid__Dict2Tuple
from Configurables import LoKi__Hybrid__DTFDict as DTFDict
from Configurables import LoKi__H  (import) LoKi__Hybrid__Dict2Tuple: Any

DictTuple = dtt.Dst.addTupleTool(LoKi__Hybrid__Dict2Tuple, "DTFTuple")
DictTuple.addTool(DTFDict, "DTF")
DictTuple.Source = "LoKi::Hybrid::DTFDict/DTF"
DictTuple.NumVar = 10
DictTuple.DTF.constrainToOriginVertex = True
DictTuple.DTF.daughtersToConstrain    = ["D0"]

DictTuple.DTF.addTool(LoKi__Hybrid__DictOfFunctors, "dict")
DictTuple.DTF.Source = "LoKi::Hybrid::DictOfFunctors/dict"
DictTuple.DTF.dict.Variables = {
    "DTFDict_Dstar_PT" : "PT",
    "DTFDict_Dstar_M"  : "M",
    "DTFDict_Dz_PT": "CHILD(PT,1)",
    "DTFDict_Dz_M" : "CHILD(M, '[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC')",
    "DTFDict_Dz_PX": "CHILD(PX, '[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC')",
    "DTFDict_Dz_PY": "CHILD(PY, '[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC')",
    "DTFDict_Dz_PZ": "CHILD(PZ, '[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC')"
    }
```

✔ More complete

✘ More difficult

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

The `MCDecayTreeTuple` class

- We have seen the basics of how to access and treat data that passes the stripping
- But for MC it is useful to know the total number of events that we have before the reconstruction chain, which allows us to compute efficiencies
- In order to get that information `MCDecayTreeTuple` algorithm where we can access the truth-level variables (i.e Pythia8 variables that are not used in the reconstruction)

```
# MCDecayTreeTuple for D2hhPromptDst2D2KKLine stripping line
#==============================================================================
from Configurables import MCDecayTreeTuple

mcdtt = MCDecayTreeTuple("DstDKKPiMCTuple")
mcdtt.Inputs = ["/Event/{0}/Phys/{1}/Particles".format(stream, line)]
mcdtt.Decay  = "[D*(2010)+ -> ^(D0 -> ^K- ^K+) ^pi+]CC"
mcdtt.addBranches({"Dst": "[D*(2010)+ -> (D0 -> K- K+) pi+]CC",
                   "Dz" : "[D*(2010)+ -> ^(D0 -> K- K+) pi+]CC",
                   "Kmi": "[D*(2010)+ -> (D0 -> ^K- K+) pi+]CC",
                   "Kpl": "[D*(2010)+ -> (D0 -> K- ^K+) pi+]CC",
                   "pi" : "[D*(2010)+ -> (D0 -> K- K+) ^pi+]CC"})

# TupleTools addition
# Only DecayTreeTupleMC TupleTools are valid
mctupletools_list =  ["MCTupleToolKinematic",
                      "MCTupleToolHierarchy",
                      "MCTupleToolEventType",        # EventTypes runned
                      "MCTupleToolInteractions",     # Number of interactions of the MC
                      "TupleToolGeneration",         # Generation information
                      "MCTupleToolAngles"]           # Angular information

mcdtt.ToolList = mctupletools_list
```

CLARIFICATIONS

- We can only add TupleTools from the DecayTreeTupleMC package
- The created ROOT TTree object will be called *MCDecayTree*

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Setting up our option file for running over mDST data

- For mDST, getting the information from the TES locations becomes quite complicated
- Basically, the main problem is that the stream is not `AllStreams`. In order to adapt our script for running MC as well as CHARM mDST we have to do the following

1. Changes to the `DecayTreeTuple` inputs

```
# These bool will allow us to chose if we run the script o Data files or MC files
Simulation = False

# Stream and stripping line where our particles are
stream = "AllStreams"
line = "D2hhPromptDst2D2KKLine"    # We have to check if this stripping line is in the 2018 charm stream. Clearly it is
wg = "Charm"

# DecayTreeTuple for D2hhPromptDst2D2KKLine stripping line
#==========================================================================================================

# DecayTreeTuple definitions
dtt = DecayTreeTuple("DstDKKPiTuple")

# For MC DST we keep the same Inputs
if Simulation:
    dtt.Inputs = ["/Event/{0}/Phys/{1}/Particles".format(stream, line)]

# For data mDST the root is different. We use the RootInTES method to change it
# These RootInTES depends of the stream of each WG. Has to be checked
else:
    dtt.Inputs = ["/Phys/{0}/Particles".format(line)]
    dtt.RootInTES = "/Event/{0}".format(wg)
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Setting up our option file for running over mDST data

- For mDST getting the information from the TES locations becomes quite complicated
- Basically, the main problem is that the stream is not `AllStreams`. In order to adapt our script for running MC as well as CHARM mDST we have to do the following

2. Changes to DaVinci inputs

```
# DaVinci options
if Simulation:
    DaVinci().UserAlgorithms += [gs]                # Instead of the DecayTreeTuple, we put the sequencer
    DaVinci().InputType  = "DST"                    # DST MC
    DaVinci().DataType   = "2016"                   # Year
    DaVinci().EvtMax     = -1
    # Magnet Conditions
    DaVinci().CondDBtag  = "sim-20170721-2-vc-md100"
    DaVinci().DDDBtag    = "dddb-20170721-3"
else:
    DaVinci().UserAlgorithms += [dtt]
    DaVinci().InputType  = "MDST"                   # mDST data
    DaVinci().DataType   = "2018"                   # Year
    DaVinci().EvtMax     = -1                       # For data, -1 is to much for an easy check
    # Magnet Conditions
    DaVinci().CondDBtag  = "cond-20180202"          # The magnet properties changed because of 2018 year and data condition
    DaVinci().DDDBtag    = "dddb-20171030-3"

DaVinci().TupleFile  = "advanced_ntuple.root"       # Name of the ntuple
DaVinci().PrintFreq  = 1000                         # Pritting options
DaVinci().Simulation = Simulation                   # MC or Data
DaVinci().Lumi       = not DaVinci().Simulation     # Only True for Data
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Add Trigger Information to your nTuple I

- The trigger is one the most important parts of the data taking process in LHCb.
- Without going into detail, the trigger works with a set of decisions **(trigger lines)** which selects if the events can be reconstructed or not.
- It is important to know which trigger lines our data passed (to compute the trigger efficiency for instance)

  - So, in order to do that, we have the TupleTool `TupleToolTrigger`
  - We have to specify the trigger lines

```
"""
*Note*: There are a lot of trigger lines for each trigger level
        Not all of them are applied to our data or MC
        We have to check which ones are applied through
"""
trigger_list = [#L0
                "L0HadronDecision",
                "L0ElectronDecision",
                "L0PhotonDecision",
                #Hlt1
                "Hlt1TrackMVADecision",
                "Hlt1TwoTrackMVADecision",
                "Hlt1TrackMVATightDecision",
                "Hlt1TwoTrackMVATightDecision",
                #Hlt2 lines TCKsh getHlt2(0x21751801)
                "Hlt2Topo2BodyDecision",
                "Hlt2Topo3BodyDecision",
                "Hlt2Topo4BodyDecision",]

dtt_trigger = dtt.addTupleTool("TupleToolTrigger")
dtt_trigger.Verbose = True
dtt_trigger.TriggerList = trigger_list
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

Add Trigger Information to your nTuple II

- Know which trigger lines are the good ones it is not an easy task. You have to check which lines the data passed for each trigger layer, which can be a nasty job
- In order to do that, there is a method in the Second Analysis Steps tutorial
- To end with the trigger chapter, we will talk about the **TISTOS** method, which is the technique used to compute the trigger efficiencies
- The method is explained in this publication LHCb-PUB-2014-039, but basically it tells if the a trigger line was triggered by your candidate (TOS) or by the rest of the event (TIS)
- To add this information to our `DecayTreeTuple` we use the TupleTool `TupleToolTISTOS`

```
dtt_TISTOS = dtt.addTupleTool("TupleToolTISTOS")
dtt_TISTOS.Verbose = True
dtt_TISTOS.TriggerList = trigger_list
```

The Selection Framework in a nutshell I

- As we explained, the stripping lines are designed to get the particles from the DST and compute selections to obtain a specific decay
- There are tons of strippings lines, but we cannot cover all the possible decays. Moreover, maybe we are interested in a specific mass region of our decay that stripping cuts
- For that we have the **Selection Framework**, which basically consist of doing the job of a stripping line, but in your DaVinci option file.
- To explain that, we will reconstruct the $D^{*+} \to D^0(\to K^+K^-)\pi^+$ decay in our MC DST file without using the stripping line

The Selection Framework in a nutshell II

1. We have to get our final state particles from their TES locations, in our case kaons and pions.

   For that, we will use the particle containers `StdAllNoPIDsPions` and `StdAllLooseKaons` and the data reading class `AutomaticData`

```
from StandardParticles import StdAllNoPIDsPions as Pions
from StandardParticles import StdAllLooseKaons as Kaons

from PhysConf.Selections import AutomaticData

Pions = AutomaticData('Phys/StdAllNoPIDsPions/Particles')
Kaons = AutomaticData('Phys/StdAllLooseKaons/Particles')
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

The Selection Framework in a nutshell III

2. Now we want to combine the kaons to create a D^0 candidate.

- It's not mandatory, but it is really recommended to add cuts to our combination. For that, we have to use LoKi functors. We can add cuts to the daughters, the mother and the vertex

- Once the cuts are defined, we create the combination with `CombineParticles`

```
# First we define the D0->K+K- combination
# We can apply cut to the daughters, to the combination and to the vertex
# For that we use LoKi functors
d0_decay_products = {'K-': '(PT > 750*MeV) & (P > 4000*MeV) & (MIPCHI2DV(PRIMARY) > 4)',
                     'K+': '(PT > 750*MeV) & (P > 4000*MeV) & (MIPCHI2DV(PRIMARY) > 4)'}

d0_comb = "(AMAXDOCA('') < 0.2*mm) & (ADAMASS('D0') < 100*MeV)"

d0_vertex = ('(VFASPF(VCHI2/VDOF)< 9)'
             '& (BPVDIRA > 0.9997)'
             "& (ADMASS('D0') < 70*MeV)")

from Configurables import CombineParticles

# We do the combination
d0 = CombineParticles('Combine_D0',                          # Name
                      DecayDescriptor='[D0 -> K- K+]cc',     # new DecayDescriptor
                      DaughtersCuts=d0_decay_products,
                      CombinationCut=d0_comb,
                      MotherCut=d0_vertex
                     )   # Combine selection does vertex fit. For neutral, we have to add ParticleCombiners = {"" : "MomentumCombiner:PUBLI
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

The Selection Framework in a nutshell IV

3. The `CombineParticles` is an algorithm which is capable of apply track fits. But, as it was defined it is empty. We have to applied it over the particles, and for that we use the `Selection` class

```
from PhysConf.Selections import Selection

# We do the selection
d0_sel = Selection('Sel_D0',
                   Algorithm=d0,                # Here we put the combiner
                   RequiredSelections=[Kaons]   # Here we put the particles.
                   )
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

The Selection Framework in a nutshell V

4. The D^0 particle has been created. Now we have to combine it with the pions to create the D^{*+} candidate. For that, we have to just repeat the same steps as before. The only difference is that the D^0 `Selection` will be an input of the D^{*+} `Selection`

```
# Now we do the some procedure for the Dst -> D0 pi+ combination
dstar_decay_products = {'pi+': '(TRCHI2DOF < 3) & (PT > 100*MeV)'}

dstar_comb = "(ADAMASS('D*(2010)+') < 400*MeV)"

dstar_vertex = ("(abs(M-MAXTREE('D0'==ABSID,M)-145.42) < 10*MeV)"
                '& (VFASPF(VCHI2/VDOF)< 9)')

dstar = CombineParticles('Combine_Dstar',
                         DecayDescriptor='[D*(2010)+ -> D0 pi+]cc',
                         DaughtersCuts=dstar_decay_products,
                         CombinationCut=dstar_comb,
                         MotherCut=dstar_vertex
                        )

dstar_sel = Selection('Sel_Dstar',
                      Algorithm=dstar,
                      RequiredSelections=[d0_sel, Pions] # Here we have two particles, pions a
                     )
```

Ivan Cambón
jose.ivan.cambon.bouzas@cern.ch

The Selection Framework in a nutshell VI

5. Now we are almost ready to create our `DecayTreeTuple`. Firstly, we have to define a `SelectionSequence`

```
# Sequention of selection definition
from PhysConf.Selections import SelectionSequence
dstar_seq = SelectionSequence('Dstar_Seq', TopSelection=dstar_sel)
```

6. After that, we have to define the `DecayTreeTuple` algorithm. Now we do not have stripping line, so the input has to change and the new one has to be the previous sequence

```
# Now there is not stripping line. For inputs, we have use the following
dtt.Inputs = dstar_seq.outputLocations()
```

The Selection Framework in a nutshell VII

7. Finally, we have to create a `GaudiSequencer` to specify that, firstly we want to run all the Selection Framework chain of algorithms, and after the `DecayTreeTuple`

```
# Now we have to define the following GaudiSequencer
# DaVinci needs run the selection framework before the DecayTreeTuple
gs1 = GaudiSequencer()
gs1.Members += [dstar_seq.sequence(), dtt]
```

The rest of the script is equal to the normal options files. We can run this in both DST and mDST files, but for mDST we have to be careful because the particles that are stored are only stripped ones

The Selection Framework has other interesting features, such us getting particles from a given stripping line. For more detail, you can check the Second Analysis Steps gitlab