Introduction to ROOT: part 2

Mirco Dorigo
mirco.dorigo@ts.infn.it

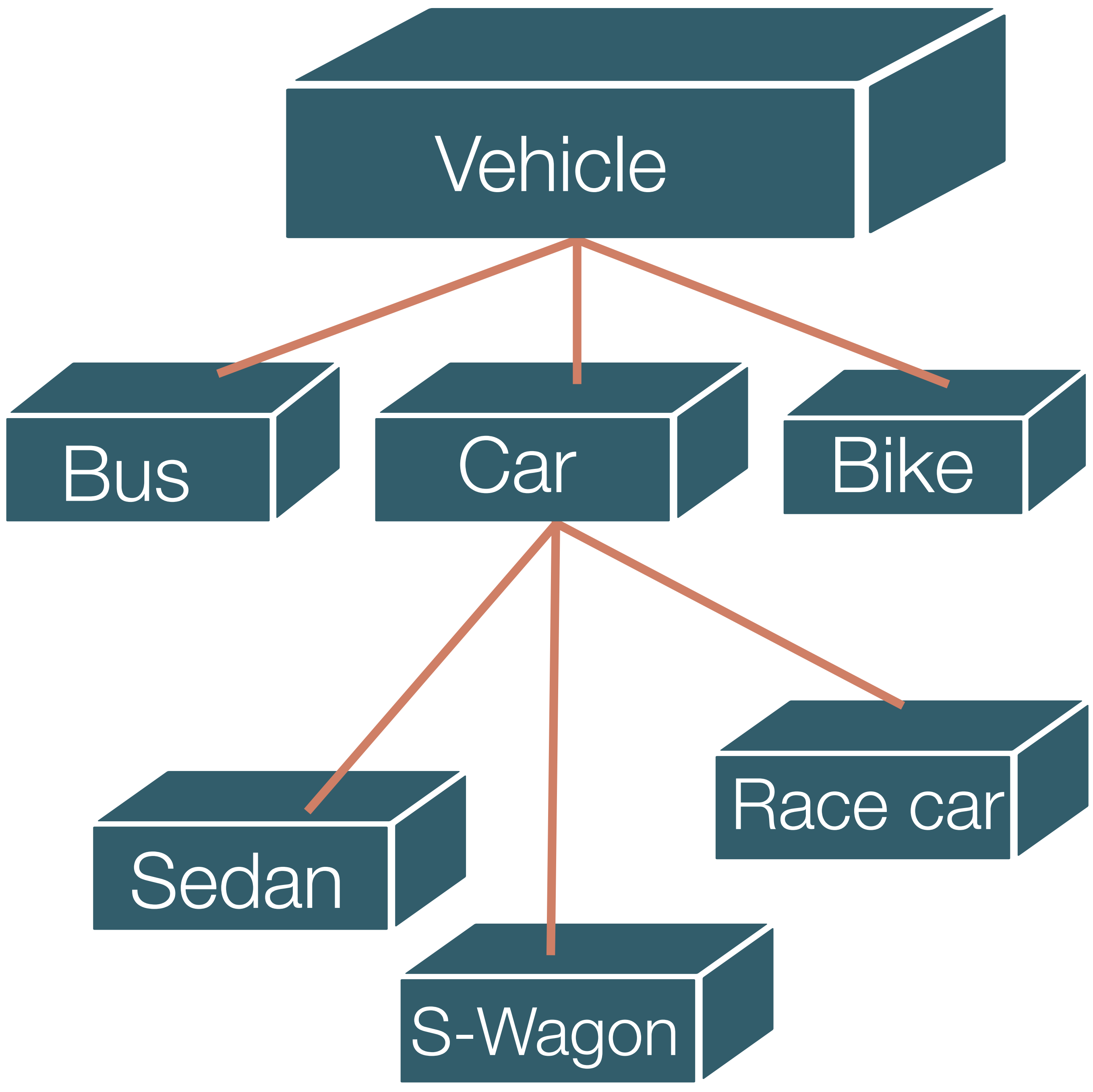
LACD 2022-2023
March 21st, 2023

Previous lesson

- Done a very quick (and incomplete) tour of `C++`.
  NOT sufficient `C++` for real-life.
- Sufficient to follow the course. We will do very simple coding (might not be really `C++` kosher…).
- Important to understand basic concepts, such that you are not lost when navigating the ROOT class reference
  (eg. https://root.cern.ch/doc/master/classTH1.html)
- Writing macros will come with examples…

Object oriented

- Classes have members (variables) and methods (functions)
- An instance of a class is an object, created by a special method, the constructor (can be overloaded).
- We can define very abstract classes, and then add derived classes that inherit from them to go more specific with what we need to do.



Inheritance

- The capability of a class to derive properties and characteristics from another class. one of the most important features of Object-Oriented Programming.

- With inheritance, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”.

- The derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class. The derived class is the specialized class for the base class.

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
        //body
}
```

Some exercises

- Start ROOT. From the prompt look at the content of your folder, and look at the content of the folder above.

- Write a macro to compute the integral of x^2 between —1 and 1.
  Don't use `TF1`, but compare your results with that of `TF1`.

- Compile the macro in ROOT (`.L macro.C+`) and run it.

- Explore the `TF1` class. Look at the type 2, expression using variable x with parameters. Using this, write a normal Gaussian function in the range —5 and 5, set the mean to 0 and the std deviation to 1, and draw it.
  Get the value of the 2nd derivative at x = 0. Put all in a macro and run it.

- From the ROOT prompt: draw the `Landau` function. Set some parameters of your preference. Draw some random value from this distribution.

Let’s make a data analysis together

- We will learn ROOT by doing an analysis using (simulated) data from a real experiment, Belle II.

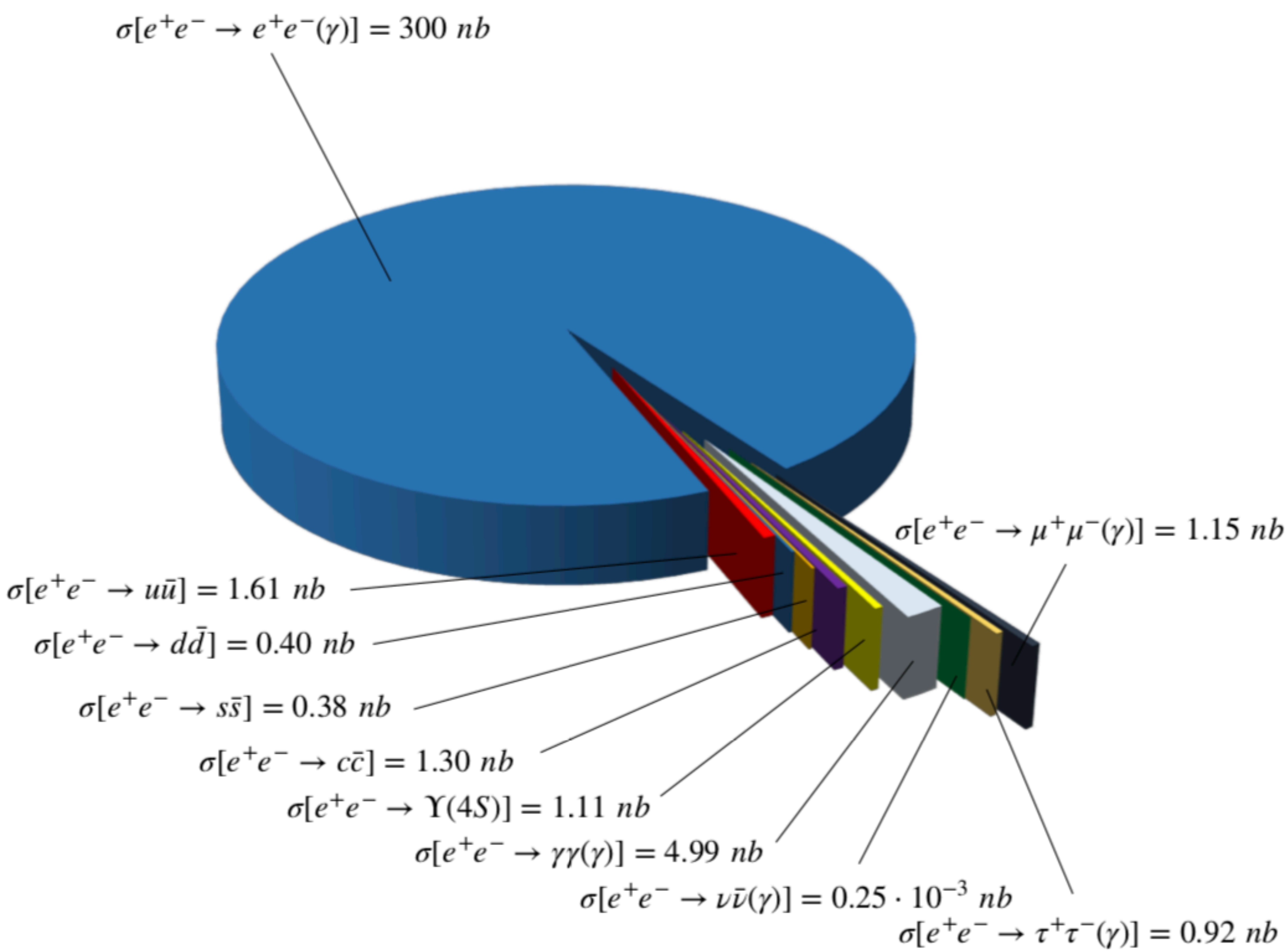
- Our goal is to see the signal peak of a rare *B* decay (branching fraction ~10^{-5}):

$$B^0 \rightarrow K^+\pi^-$$

- With ROOT we will optimise a selection to enhance our signal and measure its yield in our data.

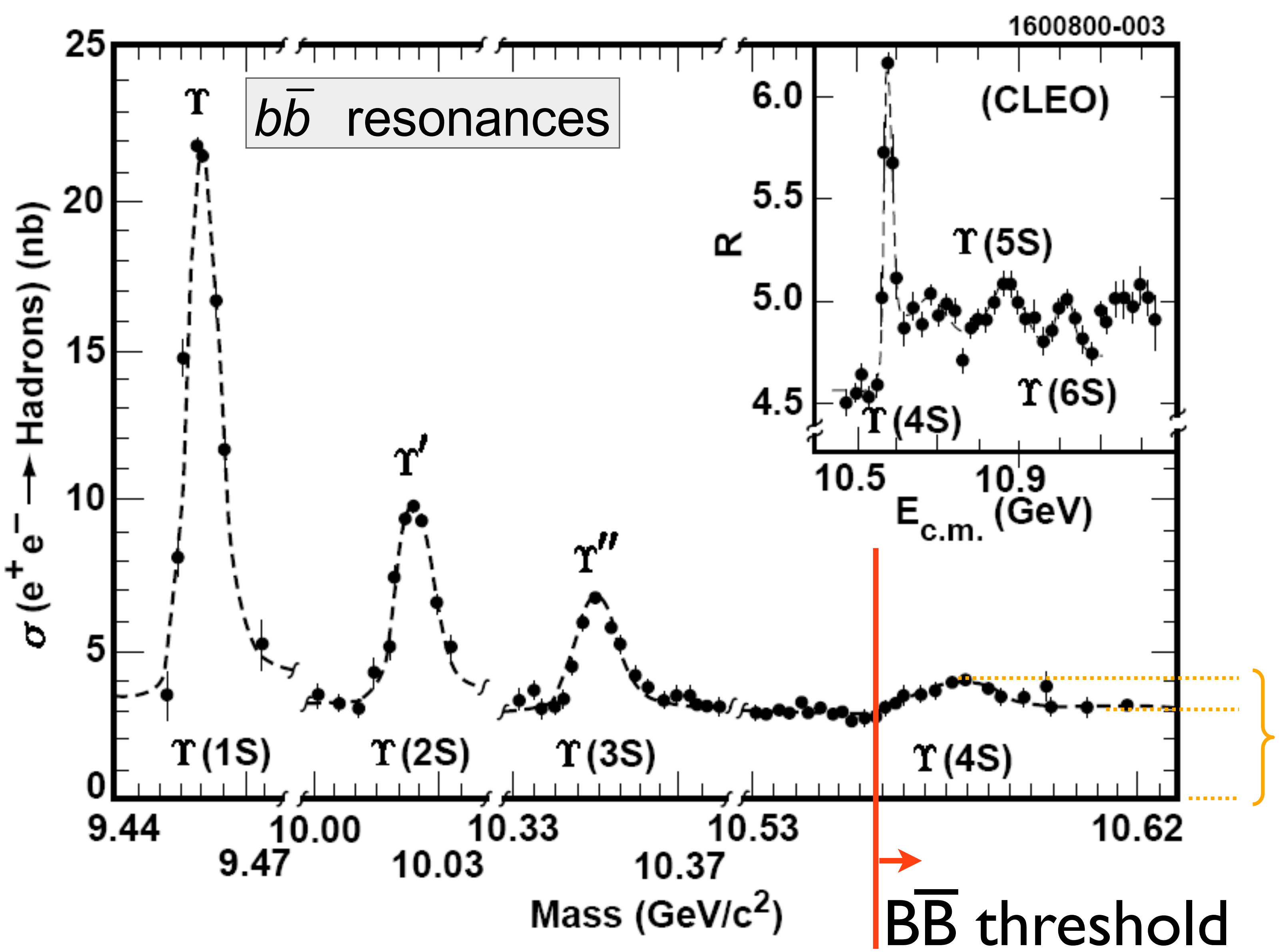
Data from Belle II

- Collisions of $(7+4)\,\mathrm{GeV}$ electron-positron beams at $\sqrt{s} \simeq 10.5794\,\mathrm{GeV}$



Data from Belle II

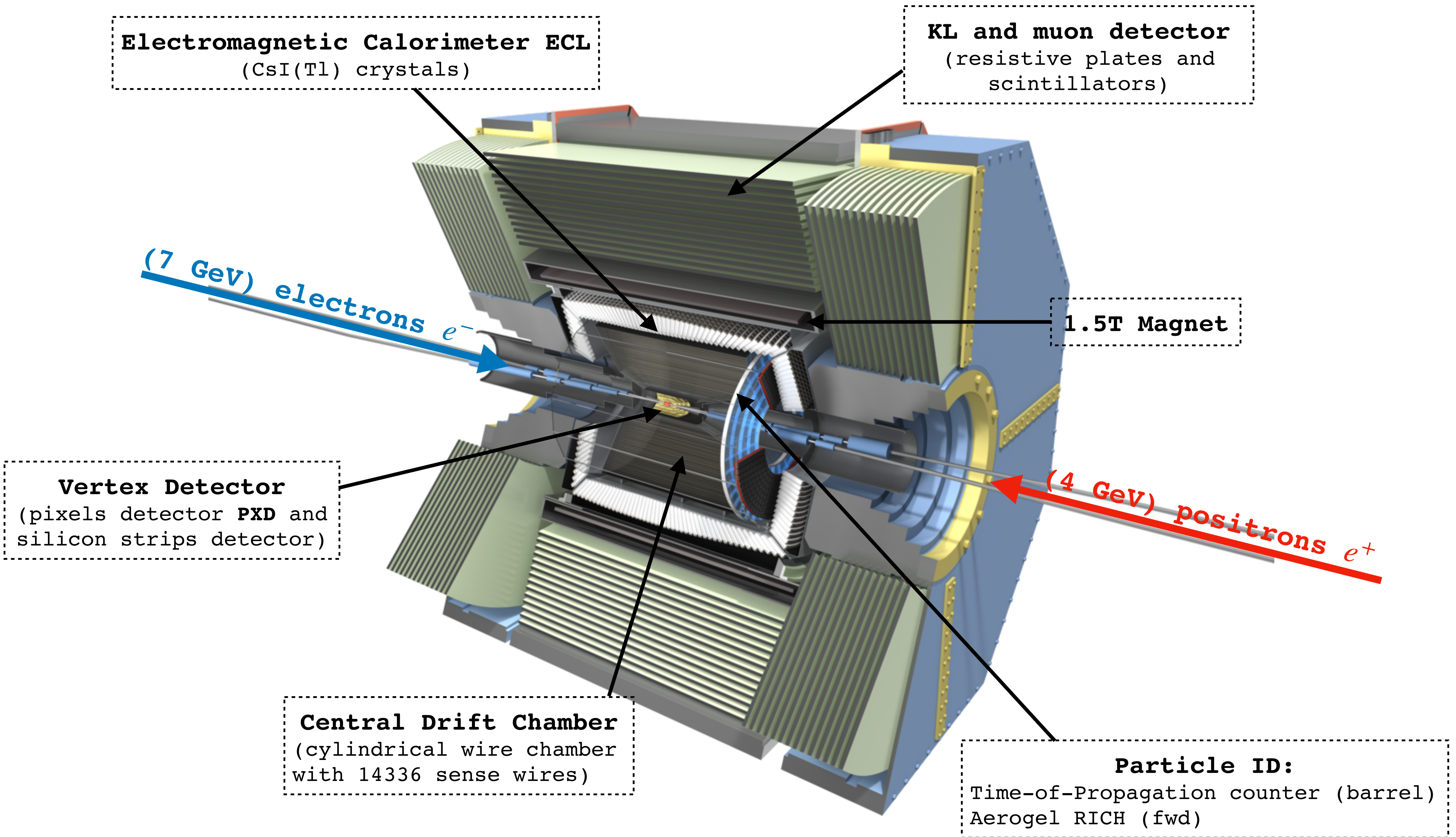
- Collisions of $(7+4)\,\mathrm{GeV}$ electron-positron beams at $\sqrt{s} \simeq 10.5794\,\mathrm{GeV}$
- $e^+e^- \rightarrow$ hadrons produce ~25% of the times a $\Upsilon(4S) \rightarrow B\overline{B}$



$$\frac{\sigma(b\bar{b})}{\sigma(hadrons)} = 0.28$$

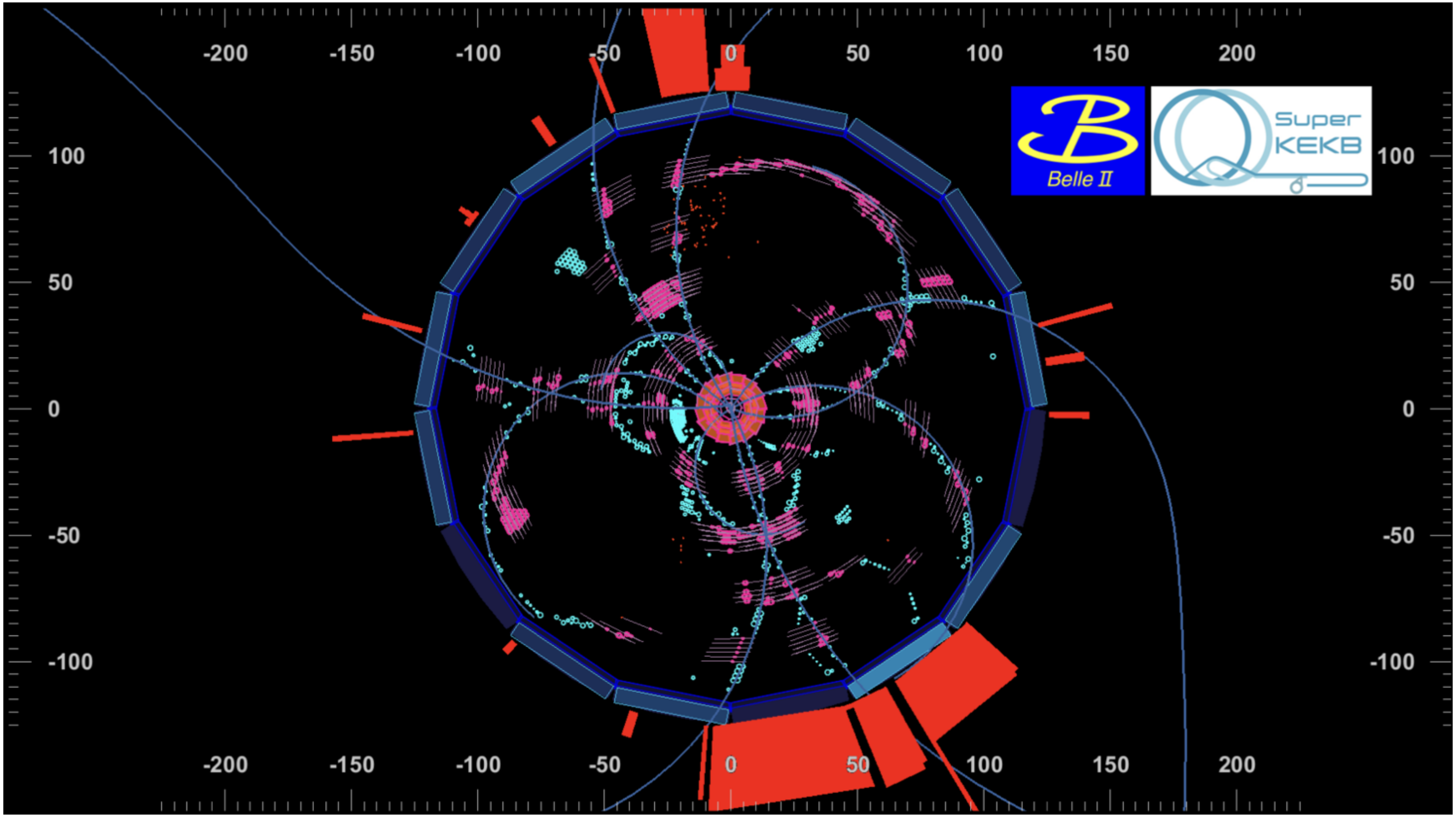
Data from Belle II

- *B* mesons have a lifetime of ~1.5 ps*: we detect the decay products.



**how much does it travel in the detector?*

Data from Belle II



Data from Belle II

- We start from a txt file which contains the momenta of candidates kaon and pion of selected events (as measured in the CMS of the $\Upsilon(4S)$ system)
- $B^0 \to K^+\pi^-$ candidates are searched for by computing the invariant mass of the kaon-pion system: the signal should peak at the expected B^0 mass.
- In $\Upsilon(4S) \to B\overline{B}$ decays, the B mesons in the CMS of $\Upsilon(4S)$, have both an energy $\sqrt{s}/2$. Since this energy is well known, let's exploit it in the mass calculation, to have a better mass resolution:

$$M = \sqrt{s/4 - |\vec{p}^{\,*}_B|^2}$$

Our data

K			π		
p_x^*	p_y^*	p_z^*	p_x^*	p_y^*	p_z^*
0.193687	-2.00117	-1.32568	0.060917	2.53833	1.11248
0.753111	2.46267	-0.931541	-0.891277	-2.01095	0.971655
-2.13514	1.34968	-0.693332	2.35859	-1.1042	0.79431
0.0305453	2.03618	1.59718	-0.204699	-1.69236	-1.92447
-2.31804	-0.747861	0.964579	2.18732	1.17039	-1.05308
-1.85069	0.557923	1.60877	1.53409	-0.841219	-1.93897
-1.0612	1.86811	1.67097	1.23339	-1.68585	-1.61119
0.936175	1.83101	-1.85589	-0.979103	-1.58107	1.75001
1.64432	0.0979302	-2.07102	-1.53113	-0.0245924	1.99033
-1.11439	1.51882	2.00723	1.00509	-1.0165	-1.89929
-0.459256	-2.09789	-1.69227	0.219982	1.67985	2.02722
-1.07834	-0.299723	-2.3634	0.91339	0.892909	2.4356
0.911784	0.89264	2.42608	-1.07651	-0.299878	-2.3489
0.233934	2.16318	1.48514	0.1323	-1.79962	-1.77417
-1.56779	-1.57578	1.15762	1.86769	1.40043	-1.14583
1.01496	1.44624	-1.68575	-0.957066	-1.99439	1.55816
0.797147	-2.48212	-0.0348265	-0.634267	2.43209	0.114049
0.597294	-2.25237	-1.35079	-1.04192	2.04739	0.929552
1.74844	1.65814	1.65216	-1.44217	-1.1239	-1.60946
-0.979769	2.00133	-1.02641	0.794014	-2.44144	1.35458
1.79225	-0.894297	1.01274	-2.25295	1.2013	-1.27669
1.54127	2.03435	-1.0719	-0.97392	-1.98406	0.777642

Reading the data

- Let's have a look at the macro `readData.C`

```
#include "Riostream.h"
#include <string>

using namespace std;

void readData(){

    //name of file for the stream of data
    string file_name ="data_file.txt";
    //initialise and open the input stream
    ifstream file_in(file_name);

    //check that the file is open
    if(!file_in.is_open()) {
        //if not, complain,
        cout << "Cannot open data file!" << endl;
        //exit and do nothing
        return;
    }
```

Riostream.h File Reference

```
#include <fstream>
#include <iostream>
#include <iomanip>
```

Avoid writing `std::` all the times when using objects from the standard `C++` library

`if(condition)` statement

Reading the data

- Nothing special …

```
//the variable in the file to read
//px, py, pz coordinates for K and pi
double k_px,  k_py,  k_pz;
double pi_px, pi_py, pi_pz;

//counter to check the total number of candidates
int icand = 0;

```

Reading the data

- The interesting part

```
    //loop till the end of the file, line-by-line
    while(file_in.is_open()){

        //read the data in a line
        file_in >> k_px  >> k_py  >> k_pz
                >> pi_px >> pi_py >> pi_pz;

        //when reach end-of-file, exit the loop
        if(file_in.eof()) break;

        //just make a check
        if(icand<10)
            printf("cand %i \t k_p(%f,%f,%f) \t pi_p(%f,%f,%f) \n",
                   icand,k_px,k_py,k_pz,pi_px,pi_py,pi_pz);

        //for each line read, increment the check counter
        ++icand;

    }
```

`while(condition)` statement

use `cin` operator

to exit the loop when reaching the end of the file

Reading the data

- Closing…

```
    //close input stream
    file_in.close();

    // just print the total number
    cout << "Total data is: " << icand << endl;

    return;

}

```

- The output is

```
mb-md-01:secondLesson dorigo$ root -l readData.C
root [0]
Processing readData.C...
cand 0   k_p(0.193687,-2.001170,-1.325680)      pi_p(0.060917,2.538330,1.112480)
cand 1   k_p(0.753111,2.462670,-0.931541)       pi_p(-0.891277,-2.010950,0.971655)
cand 2   k_p(-2.135140,1.349680,-0.693332)      pi_p(2.358590,-1.104200,0.794310)
cand 3   k_p(0.030545,2.036180,1.597180)        pi_p(-0.204699,-1.692360,-1.924470)
cand 4   k_p(-2.318040,-0.747861,0.964579)      pi_p(2.187320,1.170390,-1.053080)
cand 5   k_p(-1.850690,0.557923,1.608770)       pi_p(1.534090,-0.841219,-1.938970)
cand 6   k_p(-1.061200,1.868110,1.670970)       pi_p(1.233390,-1.685850,-1.611190)
cand 7   k_p(0.936175,1.831010,-1.855890)       pi_p(-0.979103,-1.581070,1.750010)
cand 8   k_p(1.644320,0.097930,-2.071020)       pi_p(-1.531130,-0.024592,1.990330)
cand 9   k_p(-1.114390,1.518820,2.007230)       pi_p(1.005090,-1.016500,-1.899290)
Total data is: 31523
root [1]
```

Reading the data

- Download the material.
- Try the macro yourself. Try also to compile it and run.

Compute a momentum

- Look at `computeP.C`

```
#include "Riostream.h"
#include <string>
#include "TVector3.h"
#include <vector>
#include <numeric>

using namespace std;

void computeP(){

    //File to read
    string file_name ="data_file.txt";
    ifstream file_in(file_name);

    if(!file_in.is_open()) {
        cout << "Cannot open data file!" << endl;
        return;
    }

    //the variable in the file to read
    double k_px,  k_py,  k_pz;
    double pi_px, pi_py, pi_pz;

    //counter to check the total number of candidates
    int icand = 0;

    vector<double> k_p_all;
```

A ROOT class to use 3D vector

C++ standard libraries:

- vector to store a collection of a types, a container that can change in size
- numeric to use some convenient algorithms

Compute a momentum

- Look at `computeP.C`

```
    while(file_in.is_open()){

        //read the data in a line
        file_in >> k_px  >> k_py  >> k_pz
                >> pi_px >> pi_py >> pi_pz;

        //when reach end-of-file, exit the loop
        if(file_in.eof()) break;

        //let's compute the momentum vector
        //using the class TVector3
        TVector3 k_3p(k_px,k_py,k_pz);
        //compute the magnitude of the vector
        double k_p = k_3p.Mag();

        k_p_all.push_back(k_p);

        //just have a look
        if(icand<10)
            printf("cand %i: \t k_p=%0.3f GeV/c \n",icand, k_p_all.at(icand));
                                                           //k_p_all[icand]
        ++icand;
    }
```

Construct the object and use a method

Append an element at the end, the size of the vector grows.

Easily access any element of the vector

Compute a momentum and an average

- Just std-library show-off

Number of elements in the vector

```
    //close input stream
    file_in.close();

    // just print the total number
    cout << "Total data is: " << k_p_all.size() << endl;

    //compute the mean of the K mometum
    //using the vector and the std library numeric
    double k_p_mean = accumulate(k_p_all.begin(), k_p_all.end(), 0.0) / k_p_all.size();

    //now goes the mean in the squares, again with std library numeric
    double k_p_meanSquares =
        inner_product(k_p_all.begin(), k_p_all.end(), k_p_all.begin(), 0.0) / k_p_all.size();
    //to get the standard deviation
    double k_p_stdDev = sqrt(k_p_meanSquares - k_p_mean * k_p_mean);

    cout << "Mean value of K_p (GeV/c): " << k_p_mean << endl;
    cout << "Std dev of K_p (GeV/c): " << k_p_stdDev << endl;

    return;

}
```

Compute a momentum and an average

…and what do you expect???

Notice: m(*B*) = 5280 MeV/c^2, m(*K*) = 494 MeV/c^2, m(π) = 140 MeV/c^2

Compute a momentum and an average

- The output

```
root [1] computeP()
cand 0:           k_p=2.408 GeV/c
cand 1:           k_p=2.739 GeV/c
cand 2:           k_p=2.619 GeV/c
cand 3:           k_p=2.588 GeV/c
cand 4:           k_p=2.620 GeV/c
cand 5:           k_p=2.515 GeV/c
cand 6:           k_p=2.722 GeV/c
cand 7:           k_p=2.770 GeV/c
cand 8:           k_p=2.646 GeV/c
cand 9:           k_p=2.753 GeV/c
Total data is: 31523
Mean value of K_p (GeV/c): 2.6146
Std dev of K_p (GeV/c): 0.150434
root [2]
```

Look at the distribution

- Take `histoP.C`

```
#include "Riostream.h"
#include "TVector3.h"
#include <vector>
#include <numeric>
#include "TString.h"
#include "TH1D.h"
#include "TCanvas.h"

using namespace std;

void histoP(){

    //Let's see the distribution of the K momentum
    //we will create an histogram to bin the data in the file
    //using the class TH1D: https://root.cern.ch/doc/master/classTH1.html
    int nbins = 40;  //number of bins
    double min = 1.5; //min value of the x axis
    double max = 3.5; //max value of the x axis
    TString h_name = "h_K_p"; //name of the object (eg. when saved in a root file)
    TString h_title = "K momentum distribution; p(K) [GeV/c]; "; //the histo title, can also leave blank; the x-axis title; could put also the y-axis title

    //The constructor
    TH1D* h_p = new TH1D(h_name,h_title,nbins,min,max);

    //Just set the axis title
    h_p->GetYaxis()->SetTitle(Form("Candidates per %.1f [Gev/c]",
                              h_p->GetXaxis()->GetBinWidth(1)));
    //the x axis was just set in the constructor, otherwise
    //h_p->GetXaxis()->SetTitle("p(K) [GeV/c]");
```

[ROOT class](https://root.cern.ch/doc/master/classTH1.html) for histograms of double variables, you will use it a lot

[An empty space, to draw object on it]()

Look at the distribution

- Take `histoP.C`

```
    while(file_in.is_open()){

        file_in >> k_px  >> k_py  >> k_pz
                >> pi_px >> pi_py >> pi_pz;

        if(file_in.eof()) break;

        TVector3 k_3p(k_px,k_py,k_pz);
        double k_p = k_3p.Mag();

        k_p_all.push_back(k_p);

        h_p->Fill(k_p);

        ++icand;

    }
```

Can be easier than this?

Look at the distribution

- Can take a lot of information from the histogram

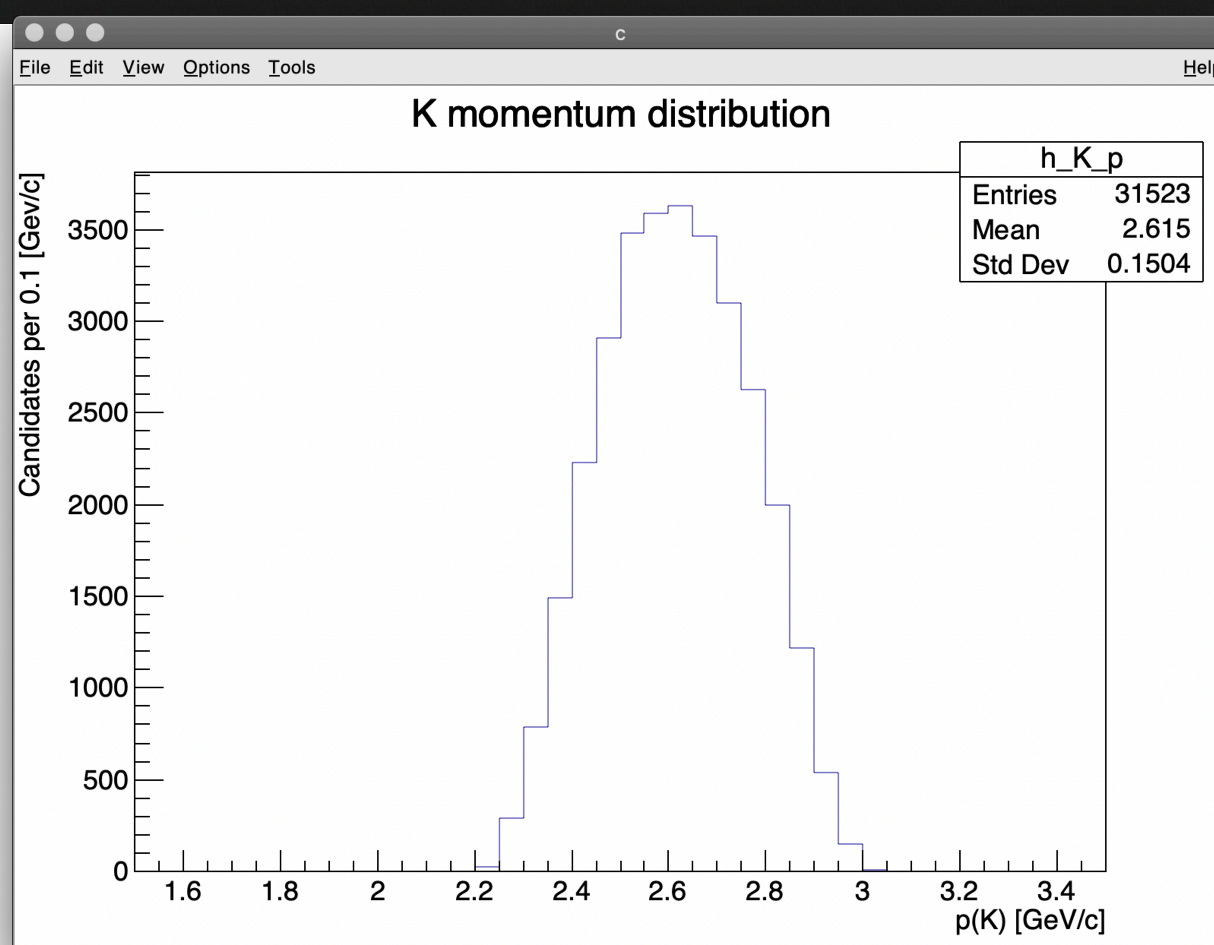
```
    cout << "Total data is:                 " << k_p_all.size() << endl;
    cout << "Mean value of K_p (GeV/c):     " << k_p_mean << endl;
    cout << "Std dev of K_p (GeV/c):        " << k_p_stdDev << endl;

    //let's print a few information from the histogram
    cout << "Total entries in the histogram: " << h_p->GetEntries() << endl;
    cout << "Integral of the histogram:     " << h_p->Integral() << endl;
    cout << "Mean of the distribution:      " << h_p->GetMean()  << " +- " << h_p->GetMeanError() << " GeV/c \n";
    cout << "Std. dev. of the distribution:  " << h_p->GetStdDev() << " +- " << h_p->GetStdDevError() << " GeV/c \n";

    //finally, draw it!
    TCanvas* c = new TCanvas("c","c",800,600);
    h_p->Draw();

    return;

}
```

Make a canvas and draw it

Look at the distribution

- The output

```
root [1] histoP()
Total data is:                   31523
Mean value of K_p (GeV/c):       2.6146
Std dev of K_p (GeV/c):          0.150434
Total entries in the histogram:  31523
Integral of the histogram:       31523
Mean of the distribution:        2.6146 +- 0.00084729 GeV/c
Std. dev. of the distribution:   0.150434 +- 0.000599125 GeV/c
root [2]
```



Look at the distribution

- Plot the histogram yourself.
- What happens if:
  - you use 40000 o 4 bins?
  - you change the range to be 0.0–2.0 or 2.6–4.0 GeV/c?
- Let’s explore the histogram “live”

Saving data in a ROOT format

- Can save data (and any `C++` object) in a compressed binary form in a ROOT file.
- ROOT provides a tree-like data structure, extremely powerful for fast access of huge amounts of data. ROOT files can have a sub-structure: they can contain directories.
- The file is in a machine-independent compressed binary format, including both data and their description

Data structures

- Simple model: many copies of the same linear data-structure (a “record”), ending up into a bidimensional data structure (a “table”).
- The tables are named “n-tuples”, as in mathematics,
  the records are called “events”, as in physics,
  and the column headers are called “variables”, as in computer science.
- ROOT provides more than n-tuples, “tree”: same data structure used in OS to save files into folders that may contain other folders.
- A tree have “branches”: simple variables or more complex objects
- A variable is the end point of a branch, a “leaf” in the ROOT jargon.

Make a Tree

- Take `makeTree.C`

```
#include "Riostream.h"
#include "TVector3.h"
#include "TString.h"
#include "TH1D.h"
#include "TCanvas.h"
#include "TTree.h"
#include "TFile.h"

using namespace std;

void makeTree(){

```

TTree.h → ROOT class for the TTree

TFile.h → Class for making a ROOT file

Make a Tree

- Here it is the structure of our tree:

```
    int icand = 0;
    double k_px,  k_py,  k_pz;
    double pi_px, pi_py, pi_pz;
    double k_p;

    //Will store all variables in a format
    //called a TTree, a root dataformat
    //very conveniet to aggregate data
    //in several dimensions
    //https://root.cern.ch/doc/master/classTTree.html
    TTree* dataTree = new TTree("dataTree","B0toKpi data");
    //define a branch of the tree for each variable
    //first the branch name, then the address of the variable,
    //then the leaf list, which is optional in case of one leaf only
    dataTree->Branch("icand",&icand,"icand/I");
    //the K momentum components
    dataTree->Branch("k_px",&k_px,"k_px/D");
    dataTree->Branch("k_py",&k_py,"k_py/D");
    dataTree->Branch("k_pz",&k_pz,"k_pz/D");
    //the pi momentum components
    dataTree->Branch("pi_px",&pi_px,"pi_px/D");
    dataTree->Branch("pi_py",&pi_py,"pi_py/D");
    dataTree->Branch("pi_pz",&pi_pz,"pi_pz/D");
    //the k momentum that we will calculate
    dataTree->Branch("k_p",&k_p,"k_p/D");
```

Variables I want to put in, to be referenced in the tree

Constructor

List of branches with their leaves: here we put a leaf for each branch, a very simple structure

Make a Tree

- Fill the tree

```
    while(file_in.is_open()){

        file_in >> k_px  >> k_py  >> k_pz
                >> pi_px >> pi_py >> pi_pz;

        if(file_in.eof()) break;

        TVector3 k_3p(k_px,k_py,k_pz);
        k_p = k_3p.Mag();

        h_p->Fill(k_p);

        dataTree->Fill();

        ++icand;
    }
```

Make a Tree

- Save in a ROOT file. We can also store the histogram.

```
    cout << "Total data is:          " << icand << endl;

    //make a trivial check...
    cout << "Candidates in the tree: " << dataTree->GetEntries() << endl;
    //look at the content
    dataTree->Print();

    //store now in a root file
    TFile* dataFile = new TFile("data_B0toKpi.root","RECREATE");
    dataTree->Write();
    h_p->Write();
    dataFile->Close();

    return;

}
```

Make a Tree

- The output
- Try it and then explore the tree from `TBrowser`

```
root [1] makeTree()
Total data is:          31523
Candidates in the tree: 31523
******************************************************************************
*Tree    :dataTree  : B0toKpi data                                           *
*Entries :    31523 : Total =         1905173 bytes  File  Size =          0 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :icand     : icand/I                                                *
*Entries :    31523 : Total  Size=     127225 bytes  All baskets in memory   *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :k_px      : k_px/D                                                 *
*Entries :    31523 : Total  Size=     253945 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    2 :k_py      : k_py/D                                                 *
*Entries :    31523 : Total  Size=     253945 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    3 :k_pz      : k_pz/D                                                 *
*Entries :    31523 : Total  Size=     253945 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    4 :pi_px     : pi_px/D                                                *
*Entries :    31523 : Total  Size=     253965 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    5 :pi_py     : pi_py/D                                                *
*Entries :    31523 : Total  Size=     253965 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    6 :pi_pz     : pi_pz/D                                                *
*Entries :    31523 : Total  Size=     253965 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    7 :k_p       : k_p/D                                                  *
*Entries :    31523 : Total  Size=     253925 bytes  All baskets in memory   *
*Baskets :        7 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
```

Exercises

- We still have to see a signal peak…
- Let's build the variables. Calculate the mass M defined in slide 10, by using the class TLorentzVector.
- Another useful variable is the difference between the B-candidate energy in the CMS and half of the collision energy, $\Delta E = E^* - \sqrt{s}/2$. Calculate the variable.
- Plot the distribution of M and that of ΔE into two canvas.
  Is this what you expected? Describe the distributions (mean, standard dev…).
- Add the variable to your tree, and save the tree in a file, adding also the two canvas showing the distributions.