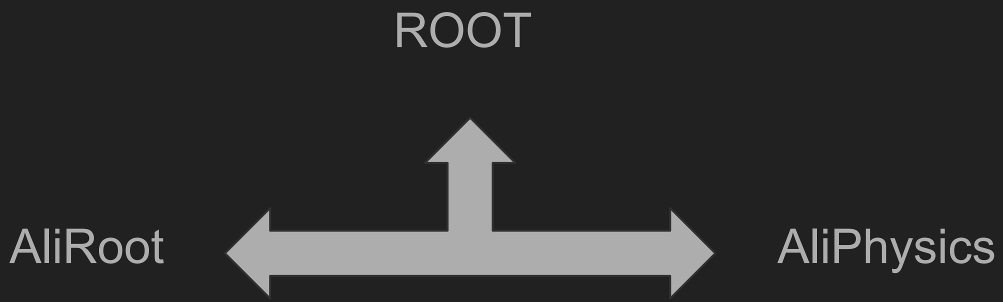
ALICE
analysis tutorial

How to write your analysis task

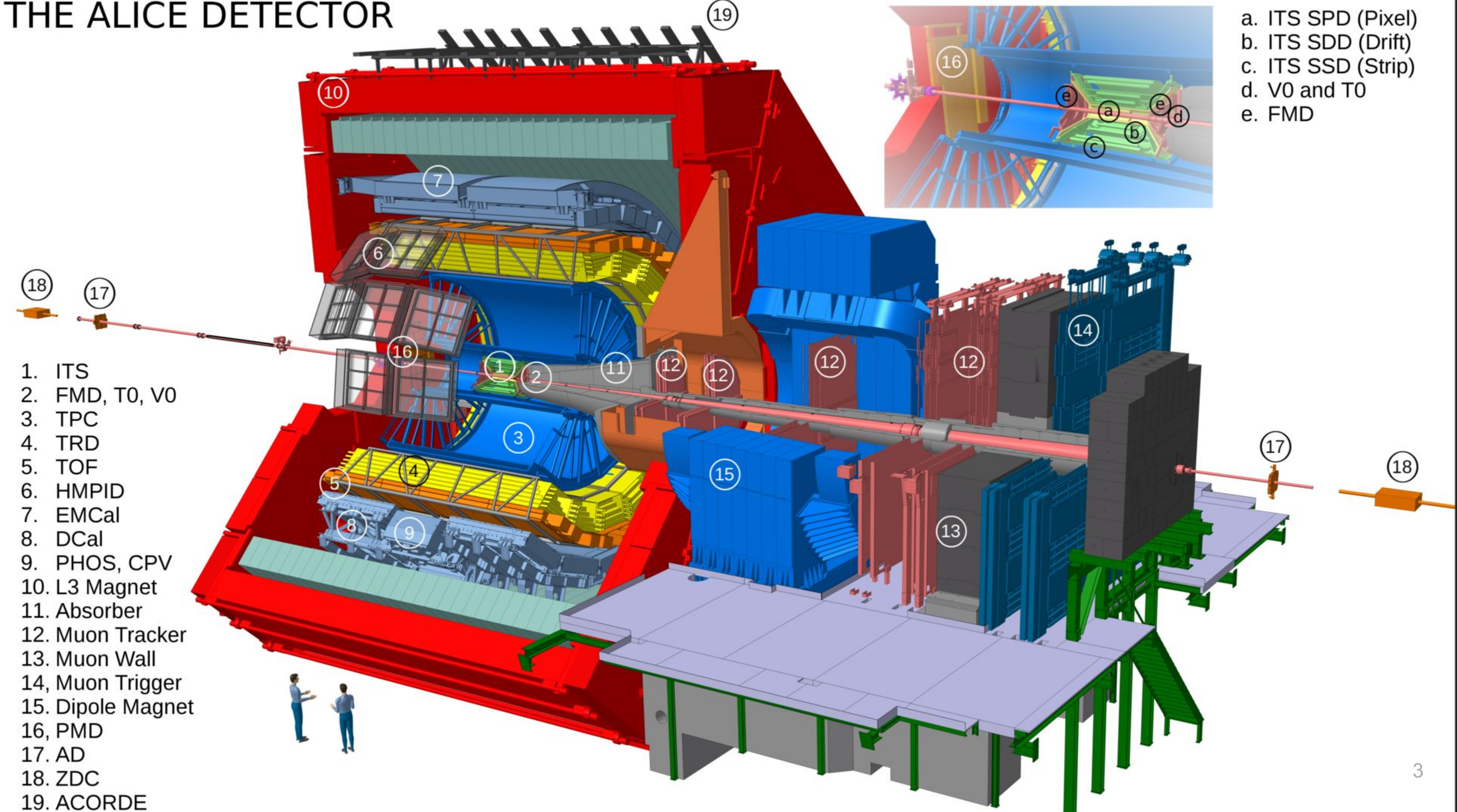
Mike Sas (mike.sas@cern.ch)

The ALICE framework

Main website for information: alice-doc.github.io

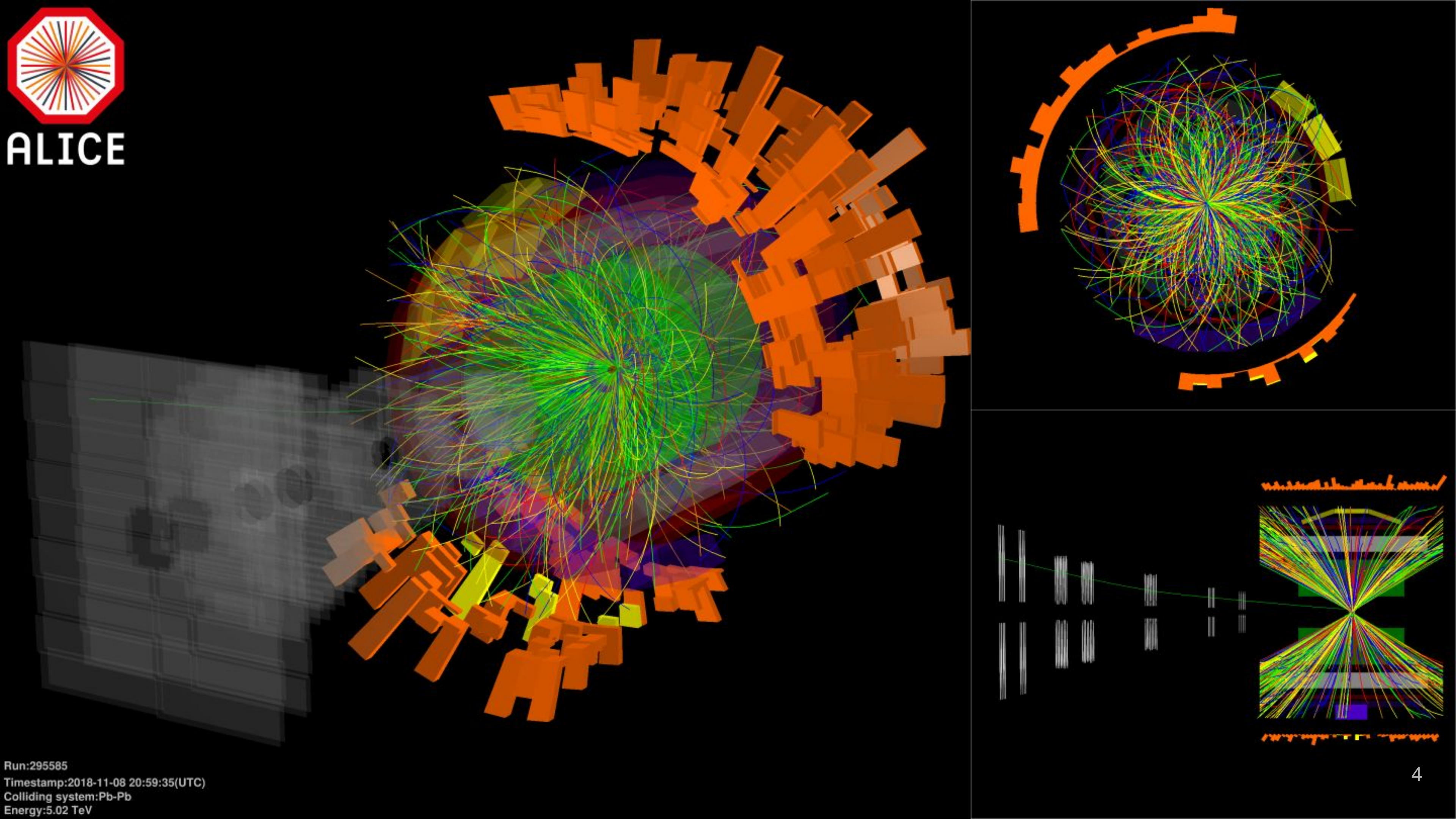


THE ALICE DETECTOR



1. ITS
2. FMD, T0, V0
3. TPC
4. TRD
5. TOF
6. HMPID
7. EMCal
8. DCal
9. PHOS, CPV
10. L3 Magnet
11. Absorber
12. Muon Tracker
13. Muon Wall
14, Muon Trigger
15. Dipole Magnet
16, PMD
17. AD
18. ZDC
19. ACORDE

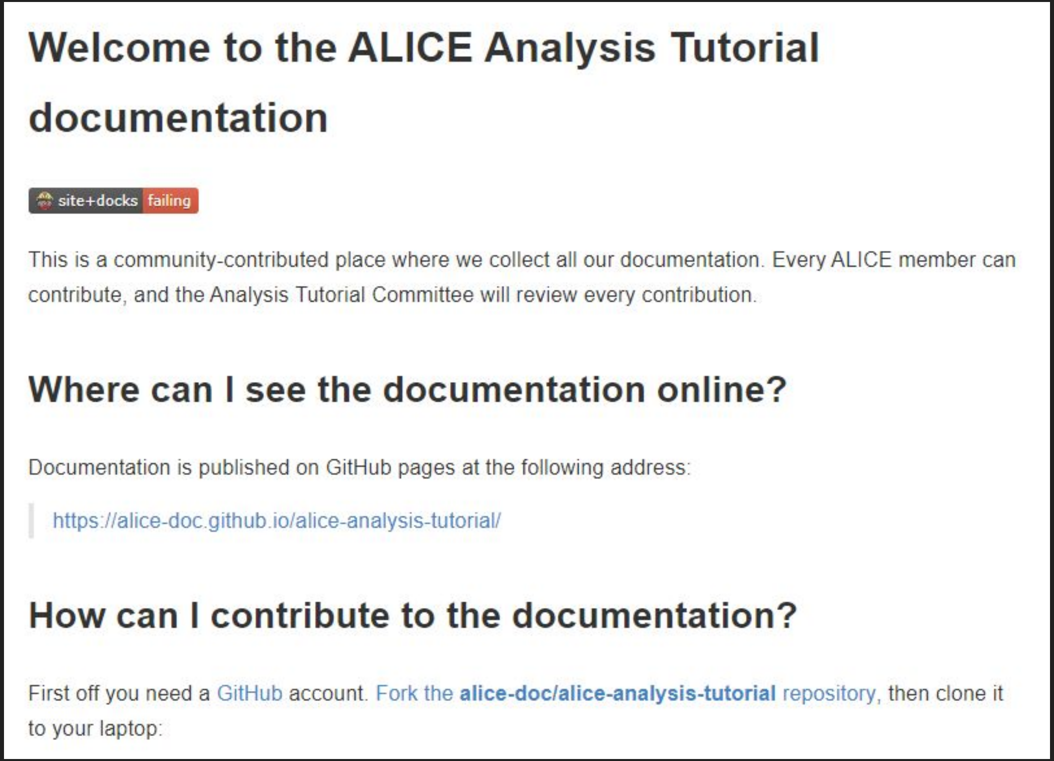
a. ITS SPD (Pixel)
b. ITS SDD (Drift)
c. ITS SSD (Strip)
d. V0 and T0
e. FMD


ALICE
Run:295585
Timestamp:2018-11-08 20:59:35(UTC)
Colliding system:Pb-Pb
Energy:5.02 TeV

How to do analysis in ALICE - Quick reference

Main website for information: alice-doc.github.io

1. **Setup**
   a. Register as ALICE member
   b. Get a Grid certificate
   c. Build the ALICE software
2. **Write your analysis task**
   a. Define and design your measurement
   b. Develop locally
   c. Test locally
3. **Acquire results**
   a. Run your task on (MC) data
   b. Make conclusions
4. **Present in your PAG / PWG** $\longrightarrow$ **iterate** $\longrightarrow$ **profit!**



Welcome to the ALICE Analysis Tutorial documentation

site+docks failing

This is a community-contributed place where we collect all our documentation. Every ALICE member can contribute, and the Analysis Tutorial Committee will review every contribution.

Where can I see the documentation online?

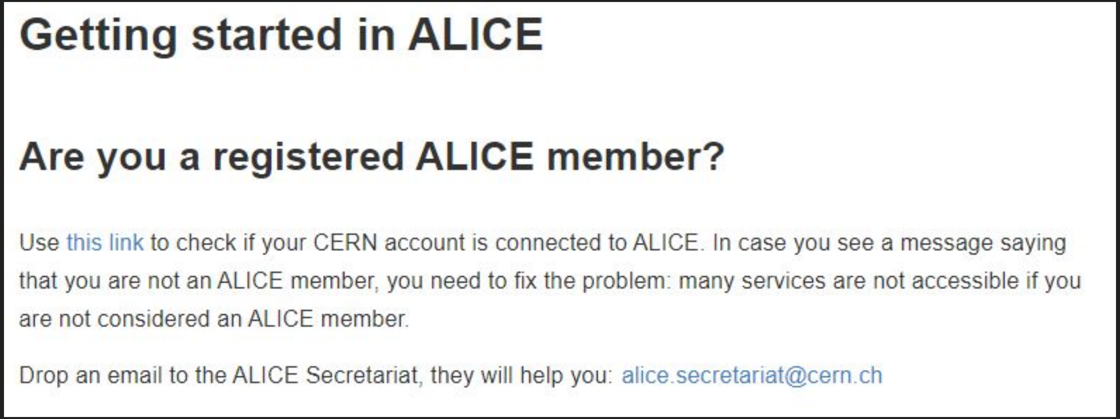
Documentation is published on GitHub pages at the following address:

> https://alice-doc.github.io/alice-analysis-tutorial/

How can I contribute to the documentation?

First off you need a GitHub account. Fork the **alice-doc/alice-analysis-tutorial** repository, then clone it to your laptop:

Setup

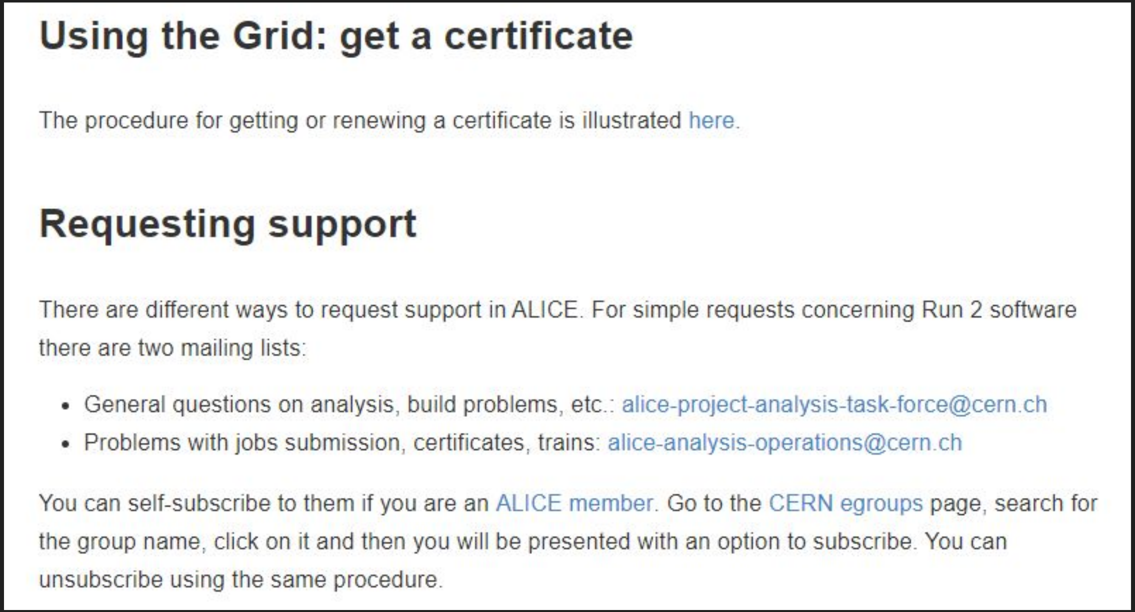

Getting started in ALICE

Are you a registered ALICE member?

Use this link to check if your CERN account is connected to ALICE. In case you see a message saying that you are not an ALICE member, you need to fix the problem: many services are not accessible if you are not considered an ALICE member.

Drop an email to the ALICE Secretariat, they will help you: alice.secretariat@cern.ch

Setup



Using the Grid: get a certificate

The procedure for getting or renewing a certificate is illustrated here.

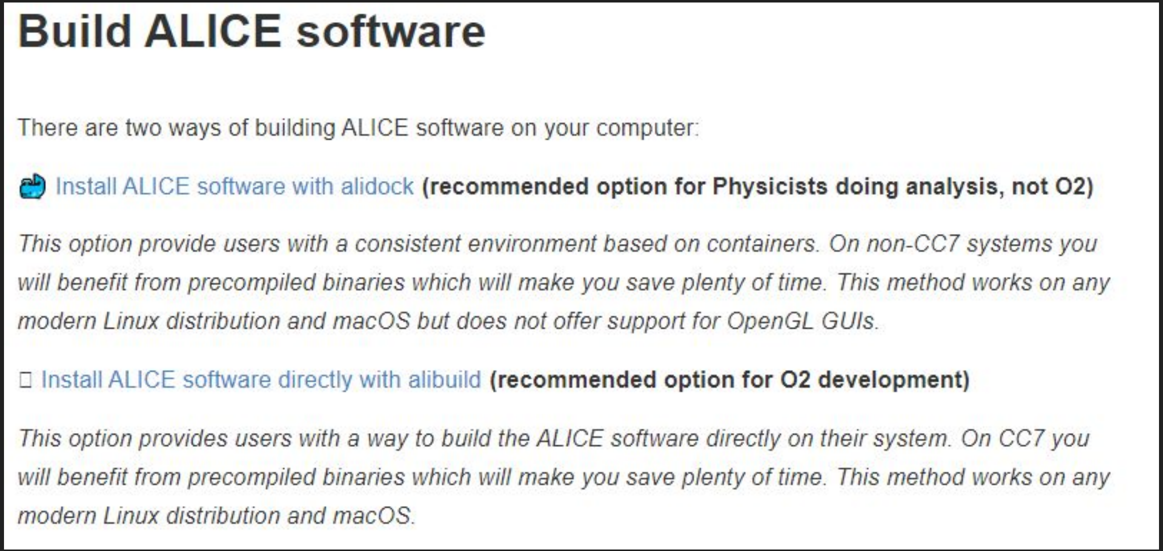
Requesting support

There are different ways to request support in ALICE. For simple requests concerning Run 2 software there are two mailing lists:

- General questions on analysis, build problems, etc.: alice-project-analysis-task-force@cern.ch
- Problems with jobs submission, certificates, trains: alice-analysis-operations@cern.ch

You can self-subscribe to them if you are an ALICE member. Go to the CERN egroups page, search for the group name, click on it and then you will be presented with an option to subscribe. You can unsubscribe using the same procedure.

Setup



Build ALICE software

There are two ways of building ALICE software on your computer:

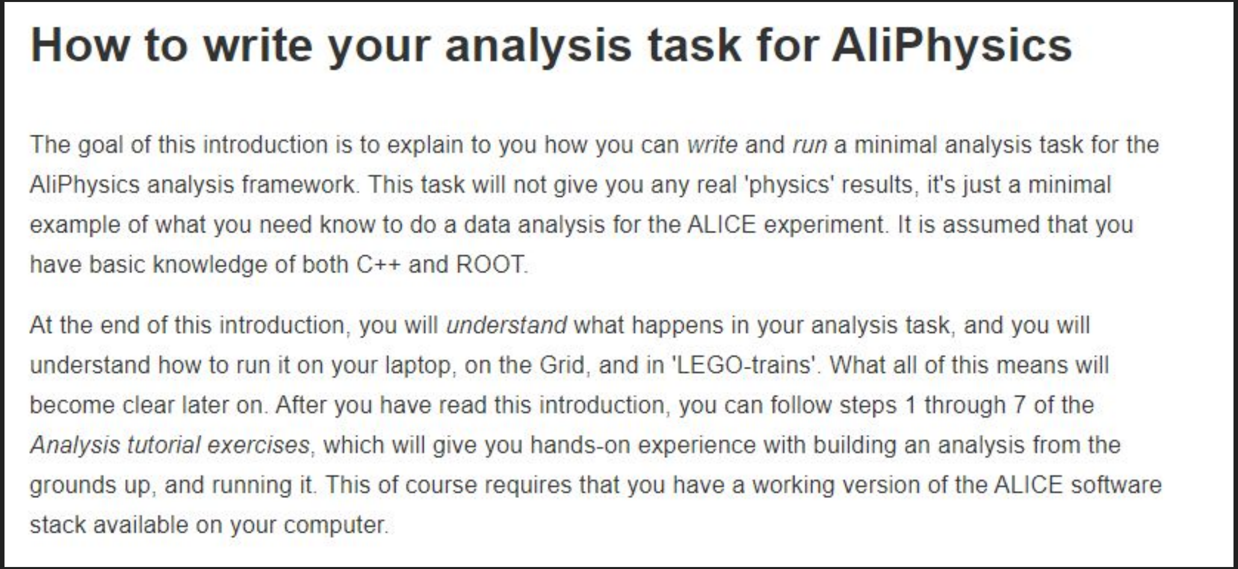
Install ALICE software with alidock **(recommended option for Physicists doing analysis, not O2)**

This option provide users with a consistent environment based on containers. On non-CC7 systems you will benefit from precompiled binaries which will make you save plenty of time. This method works on any modern Linux distribution and macOS but does not offer support for OpenGL GUIs.

Install ALICE software directly with alibuild **(recommended option for O2 development)**

This option provides users with a way to build the ALICE software directly on their system. On CC7 you will benefit from precompiled binaries which will make you save plenty of time. This method works on any modern Linux distribution and macOS.

Write your analysis task

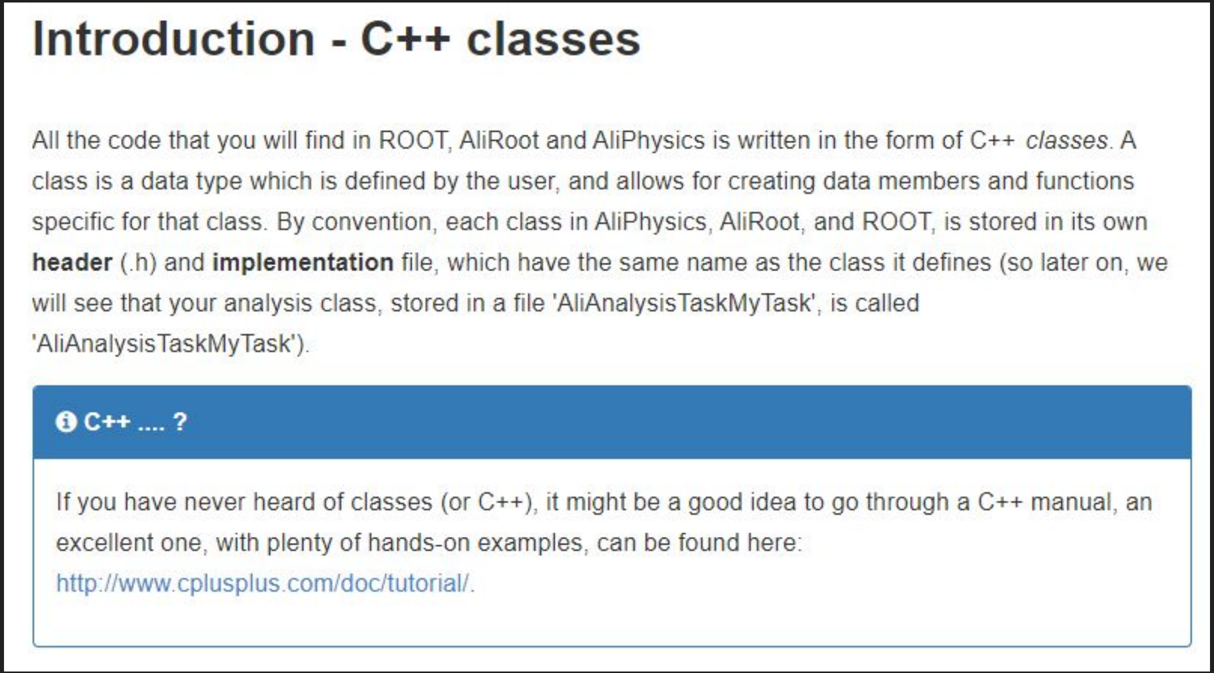


How to write your analysis task for AliPhysics

The goal of this introduction is to explain to you how you can *write* and *run* a minimal analysis task for the AliPhysics analysis framework. This task will not give you any real 'physics' results, it's just a minimal example of what you need know to do a data analysis for the ALICE experiment. It is assumed that you have basic knowledge of both C++ and ROOT.

At the end of this introduction, you will *understand* what happens in your analysis task, and you will understand how to run it on your laptop, on the Grid, and in 'LEGO-trains'. What all of this means will become clear later on. After you have read this introduction, you can follow steps 1 through 7 of the *Analysis tutorial exercises*, which will give you hands-on experience with building an analysis from the grounds up, and running it. This of course requires that you have a working version of the ALICE software stack available on your computer.

Write your analysis task



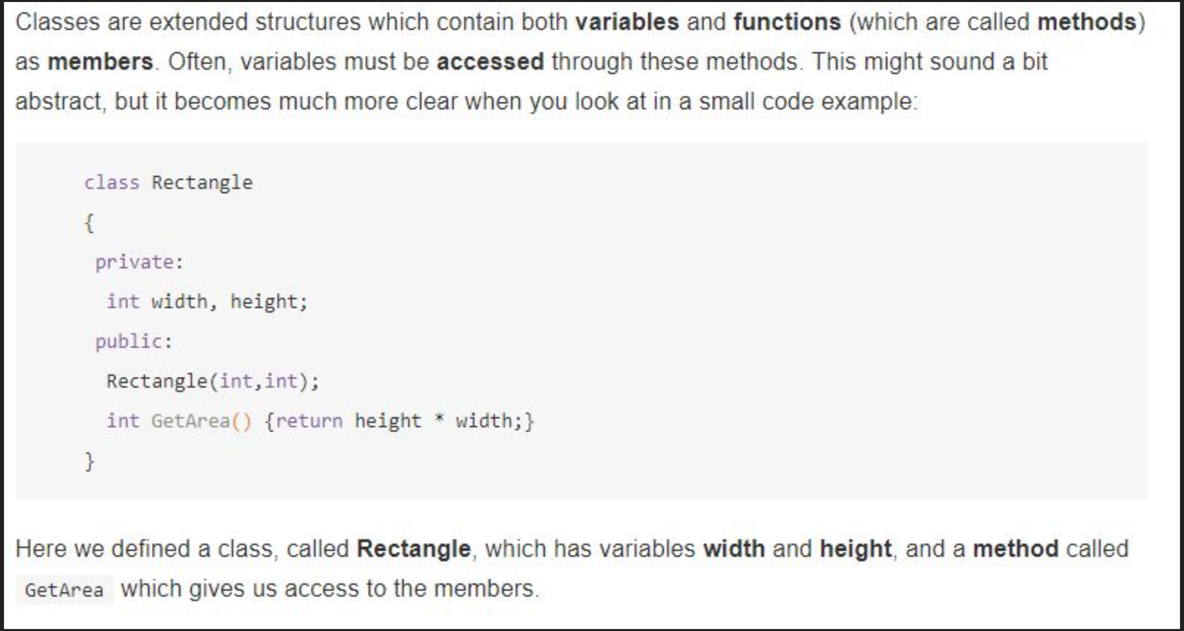
Introduction - C++ classes

All the code that you will find in ROOT, AliRoot and AliPhysics is written in the form of C++ *classes*. A class is a data type which is defined by the user, and allows for creating data members and functions specific for that class. By convention, each class in AliPhysics, AliRoot, and ROOT, is stored in its own **header** (.h) and **implementation** file, which have the same name as the class it defines (so later on, we will see that your analysis class, stored in a file 'AliAnalysisTaskMyTask', is called 'AliAnalysisTaskMyTask').

C++ .... ?

If you have never heard of classes (or C++), it might be a good idea to go through a C++ manual, an excellent one, with plenty of hands-on examples, can be found here: http://www.cplusplus.com/doc/tutorial/.

Write your analysis task

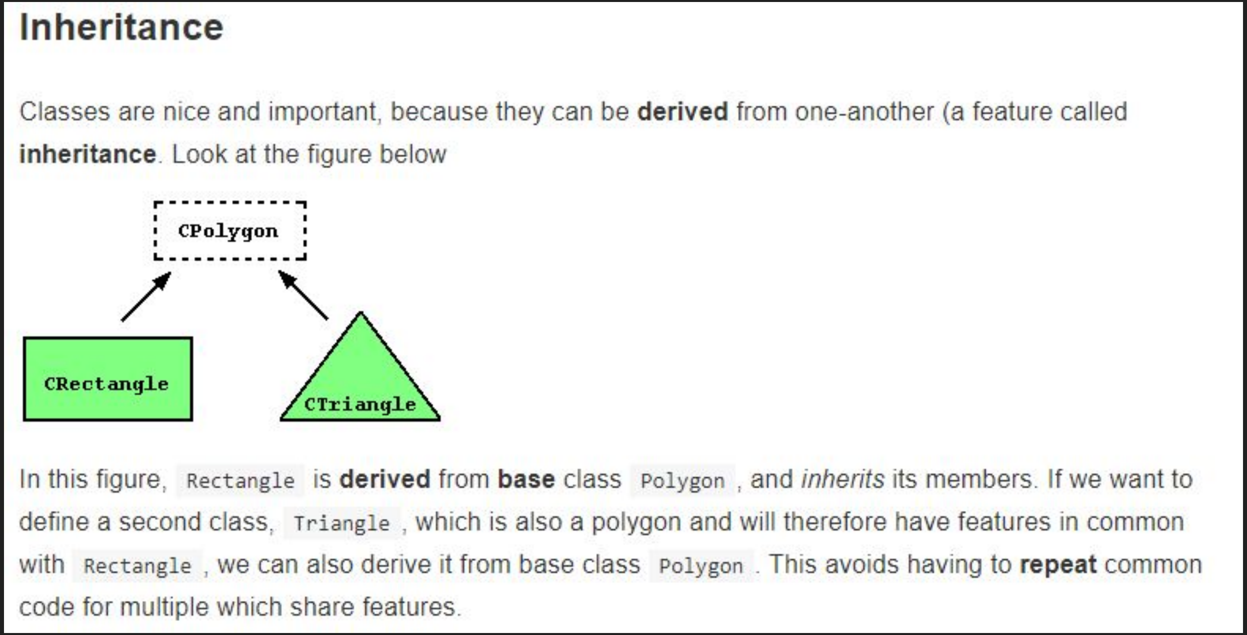


Classes are extended structures which contain both **variables** and **functions** (which are called **methods**) as **members**. Often, variables must be **accessed** through these methods. This might sound a bit abstract, but it becomes much more clear when you look at in a small code example:

```
class Rectangle
{
 private:
  int width, height;
 public:
  Rectangle(int,int);
  int GetArea() {return height * width;}
}
```

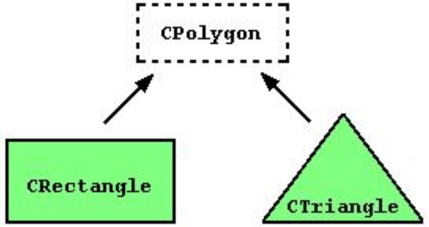
Here we defined a class, called **Rectangle**, which has variables **width** and **height**, and a **method** called `GetArea` which gives us access to the members.

Write your analysis task



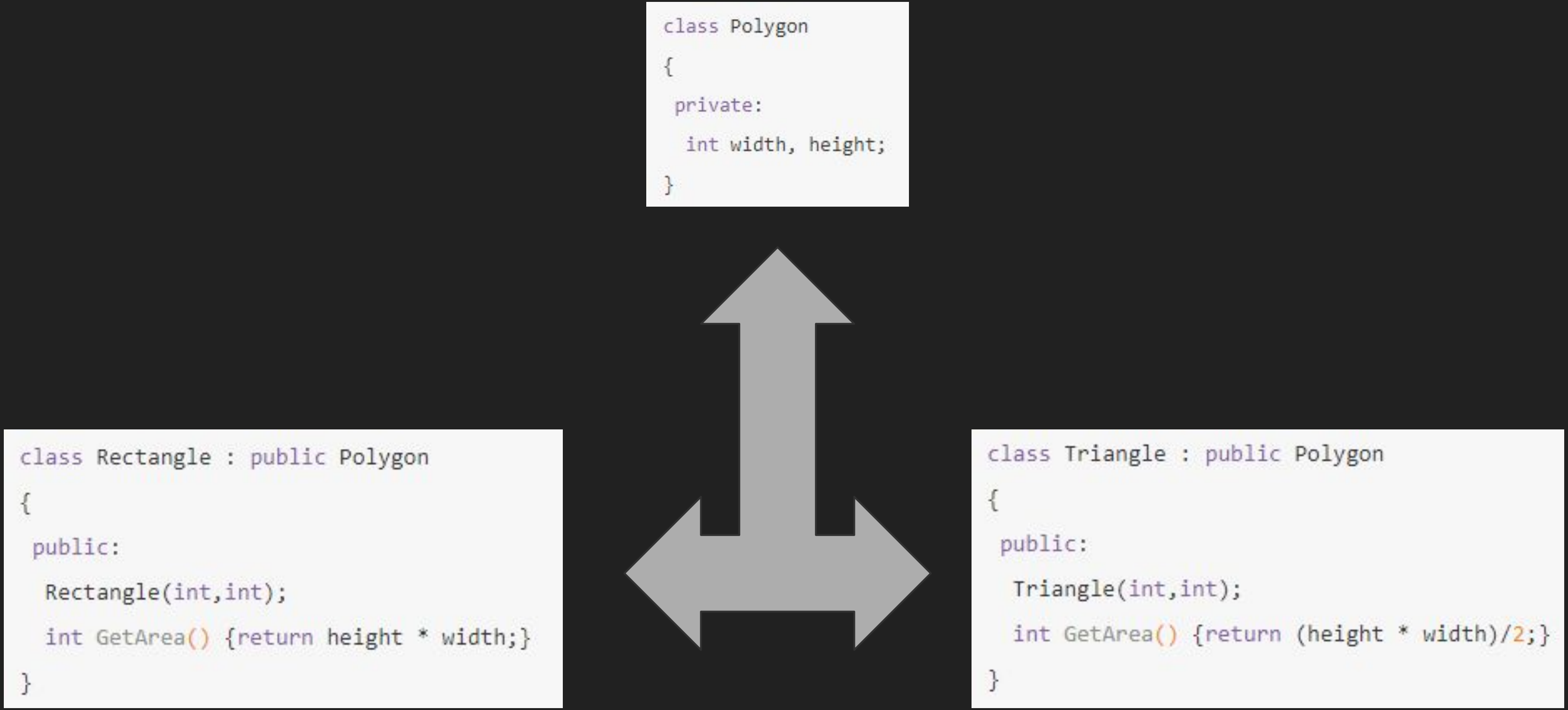
Inheritance

Classes are nice and important, because they can be **derived** from one-another (a feature called **inheritance**. Look at the figure below



In this figure, `Rectangle` is **derived** from **base** class `Polygon` , and *inherits* its members. If we want to define a second class, `Triangle` , which is also a polygon and will therefore have features in common with `Rectangle` , we can also derive it from base class `Polygon` . This avoids having to **repeat** common code for multiple which share features.

Write your analysis task

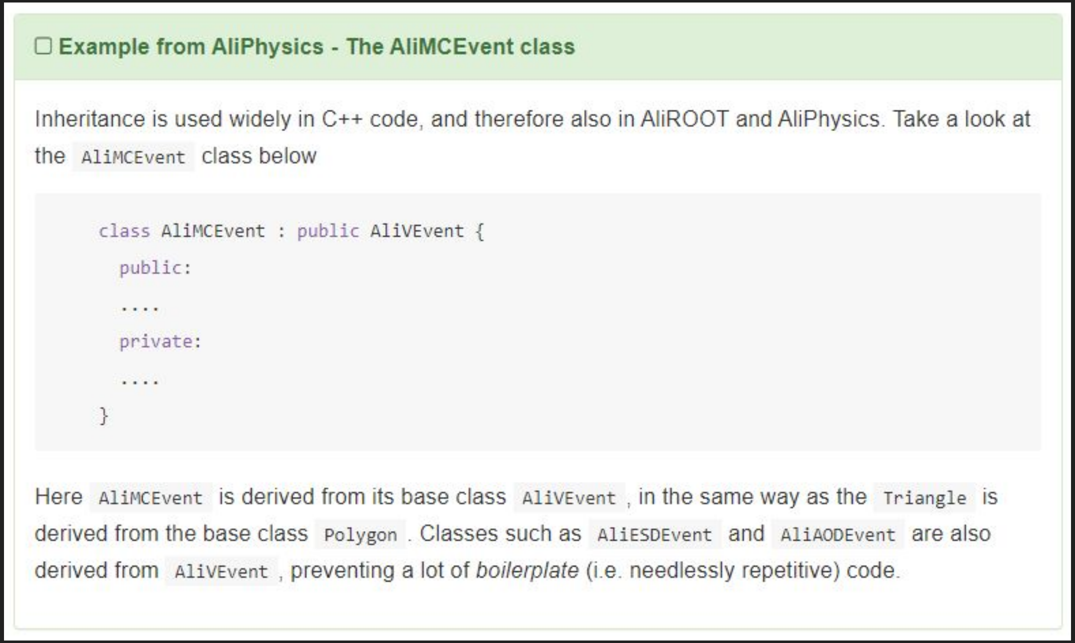


```
class Polygon
{
 private:
  int width, height;
}
```

```
class Rectangle : public Polygon
{
 public:
  Rectangle(int,int);
  int GetArea() {return height * width;}
}
```

```
class Triangle : public Polygon
{
 public:
  Triangle(int,int);
  int GetArea() {return (height * width)/2;}
}
```

Write your analysis task



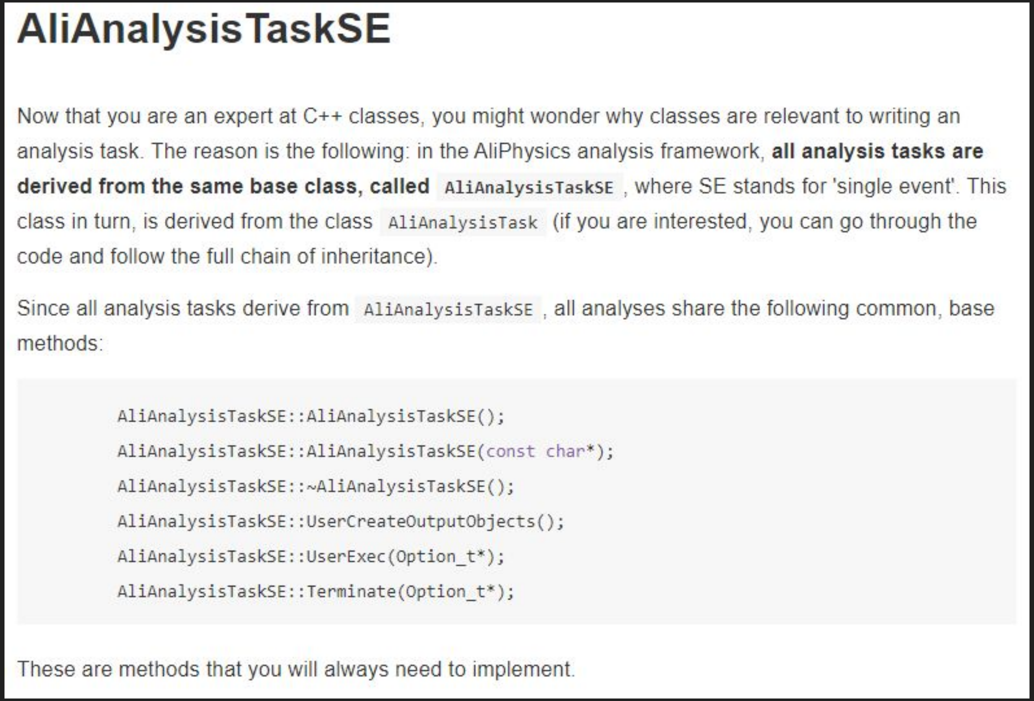
☐ **Example from AliPhysics - The AliMCEvent class**

Inheritance is used widely in C++ code, and therefore also in AliROOT and AliPhysics. Take a look at the `AliMCEvent` class below

```
class AliMCEvent : public AliVEvent {
  public:
  ....
  private:
  ....
}
```

Here `AliMCEvent` is derived from its base class `AliVEvent`, in the same way as the `Triangle` is derived from the base class `Polygon`. Classes such as `AliESDEvent` and `AliAODEvent` are also derived from `AliVEvent`, preventing a lot of *boilerplate* (i.e. needlessly repetitive) code.

Write your analysis task



AliAnalysisTaskSE

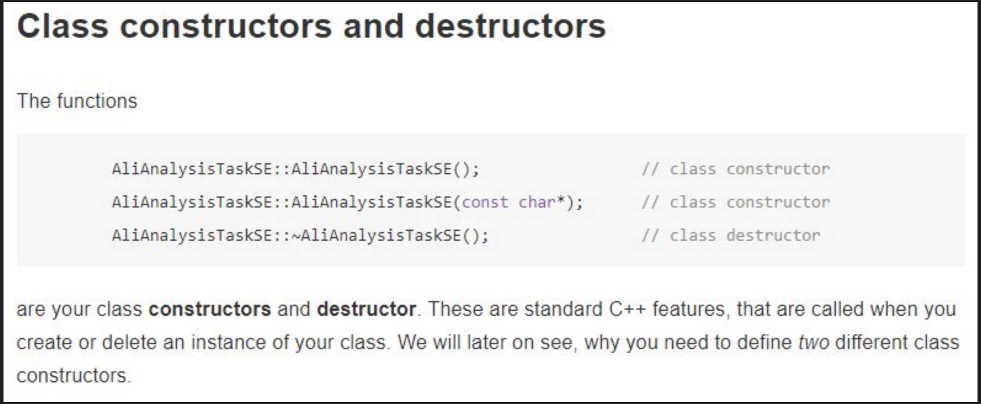
Now that you are an expert at C++ classes, you might wonder why classes are relevant to writing an analysis task. The reason is the following: in the AliPhysics analysis framework, **all analysis tasks are derived from the same base class, called** `AliAnalysisTaskSE` , where SE stands for 'single event'. This class in turn, is derived from the class `AliAnalysisTask` (if you are interested, you can go through the code and follow the full chain of inheritance).

Since all analysis tasks derive from `AliAnalysisTaskSE` , all analyses share the following common, base methods:

```
        AliAnalysisTaskSE::AliAnalysisTaskSE();
        AliAnalysisTaskSE::AliAnalysisTaskSE(const char*);
        AliAnalysisTaskSE::~AliAnalysisTaskSE();
        AliAnalysisTaskSE::UserCreateOutputObjects();
        AliAnalysisTaskSE::UserExec(Option_t*);
        AliAnalysisTaskSE::Terminate(Option_t*);
```

These are methods that you will always need to implement.

Write your analysis task



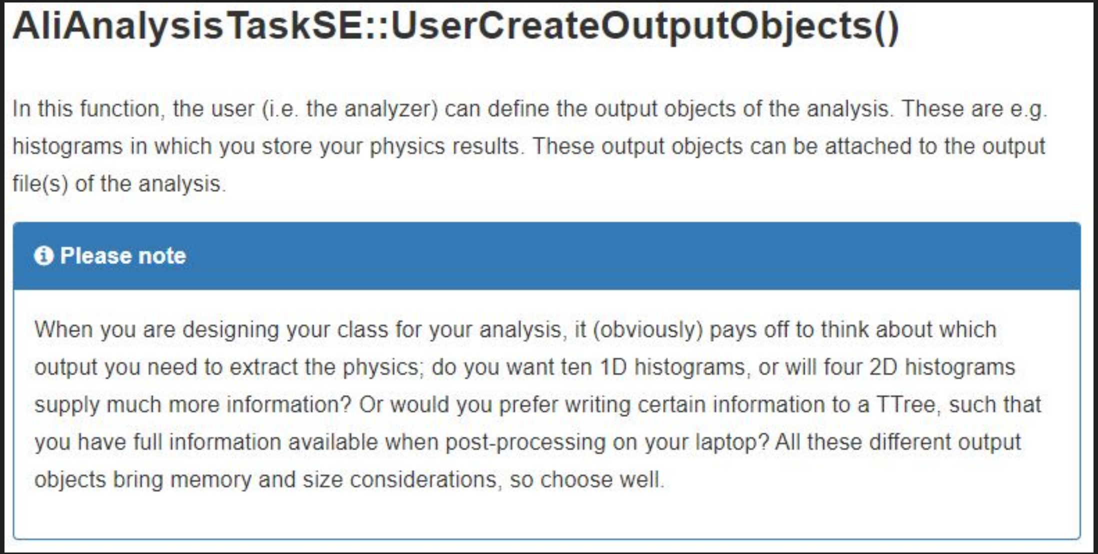
Class constructors and destructors

The functions

```
        AliAnalysisTaskSE::AliAnalysisTaskSE();                 // class constructor
        AliAnalysisTaskSE::AliAnalysisTaskSE(const char*);      // class constructor
        AliAnalysisTaskSE::~AliAnalysisTaskSE();                // class destructor
```

are your class **constructors** and **destructor**. These are standard C++ features, that are called when you create or delete an instance of your class. We will later on see, why you need to define *two* different class constructors.

Write your analysis task



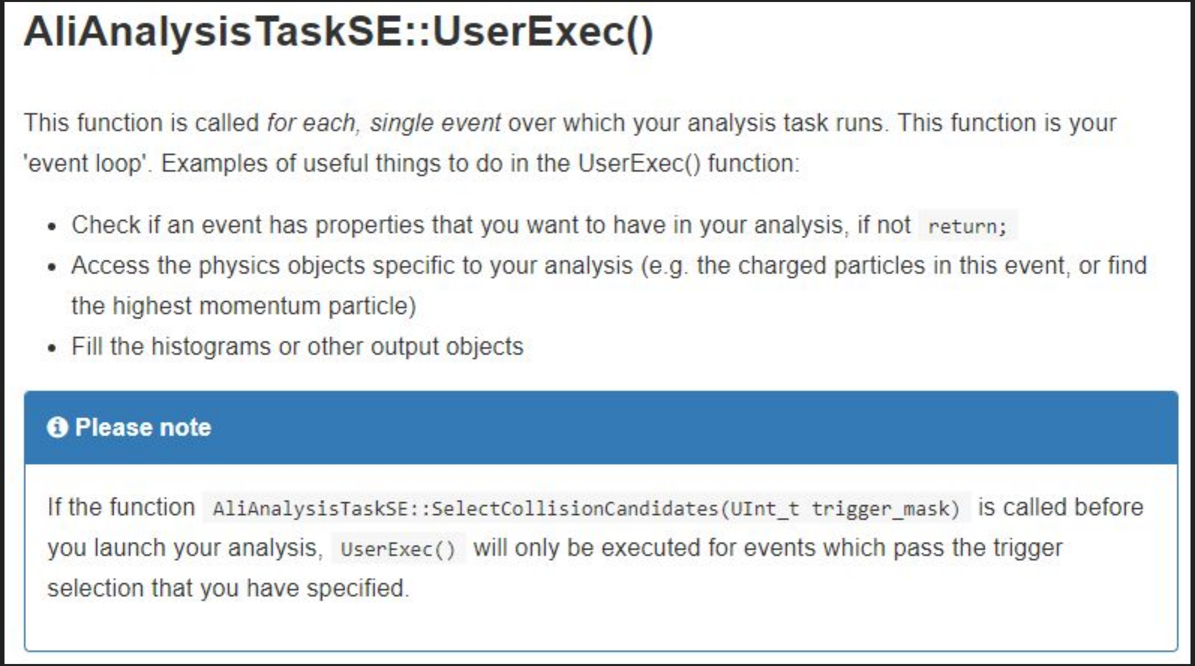
AliAnalysisTaskSE::UserCreateOutputObjects()

In this function, the user (i.e. the analyzer) can define the output objects of the analysis. These are e.g. histograms in which you store your physics results. These output objects can be attached to the output file(s) of the analysis.

Please note

When you are designing your class for your analysis, it (obviously) pays off to think about which output you need to extract the physics; do you want ten 1D histograms, or will four 2D histograms supply much more information? Or would you prefer writing certain information to a TTree, such that you have full information available when post-processing on your laptop? All these different output objects bring memory and size considerations, so choose well.

Write your analysis task



AliAnalysisTaskSE::UserExec()

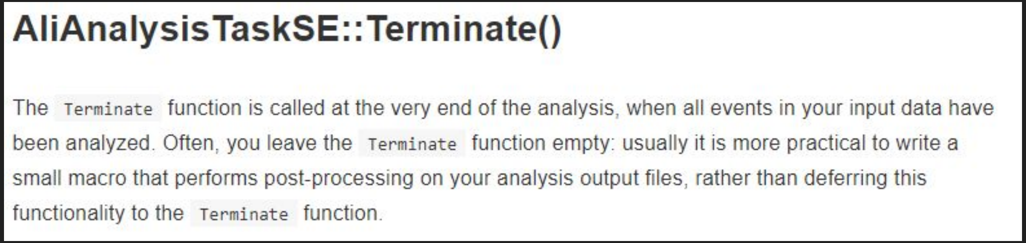
This function is called *for each, single event* over which your analysis task runs. This function is your 'event loop'. Examples of useful things to do in the UserExec() function:

- Check if an event has properties that you want to have in your analysis, if not `return;`
- Access the physics objects specific to your analysis (e.g. the charged particles in this event, or find the highest momentum particle)
- Fill the histograms or other output objects

Please note

If the function `AliAnalysisTaskSE::SelectCollisionCandidates(UInt_t trigger_mask)` is called before you launch your analysis, `UserExec()` will only be executed for events which pass the trigger selection that you have specified.

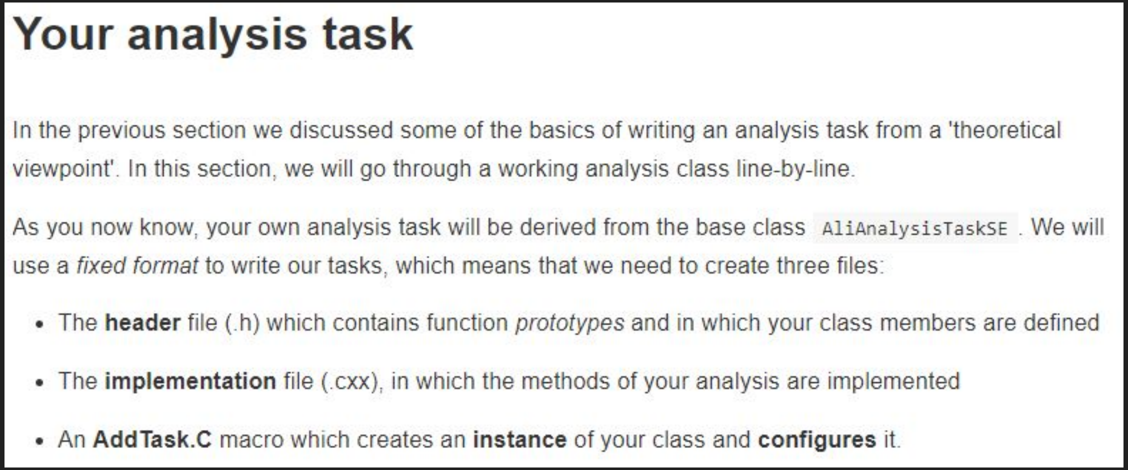
Write your analysis task



AliAnalysisTaskSE::Terminate()

The `Terminate` function is called at the very end of the analysis, when all events in your input data have been analyzed. Often, you leave the `Terminate` function empty: usually it is more practical to write a small macro that performs post-processing on your analysis output files, rather than deferring this functionality to the `Terminate` function.

Write your analysis task



Your analysis task

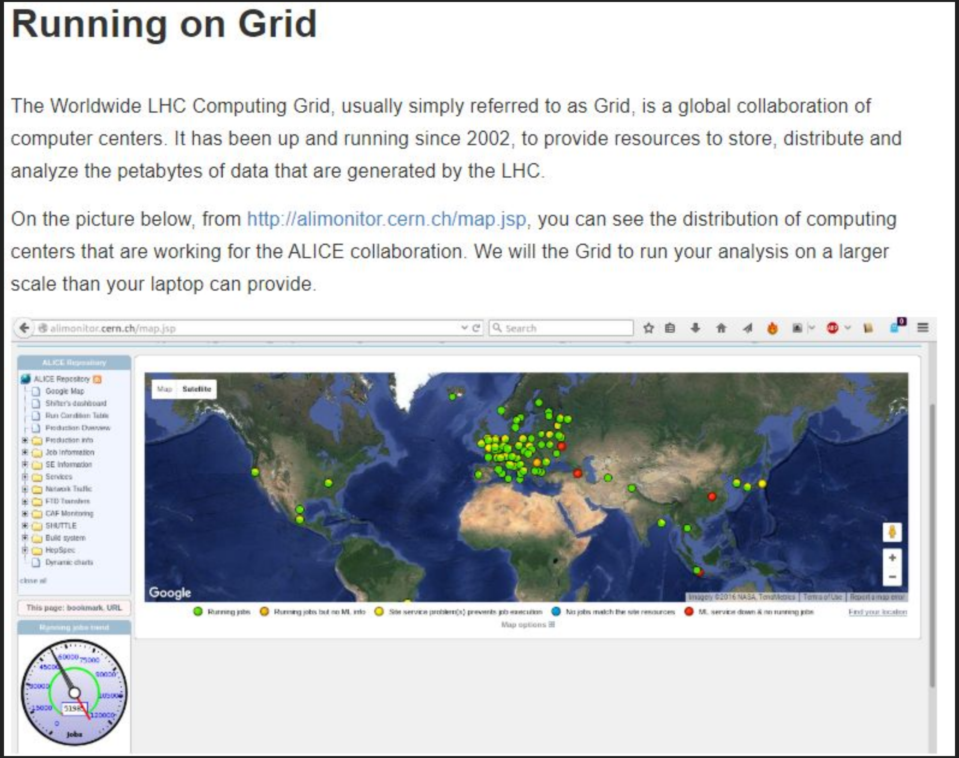
In the previous section we discussed some of the basics of writing an analysis task from a 'theoretical viewpoint'. In this section, we will go through a working analysis class line-by-line.

As you now know, your own analysis task will be derived from the base class `AliAnalysisTaskSE`. We will use a *fixed format* to write our tasks, which means that we need to create three files:

- The **header** file (.h) which contains function *prototypes* and in which your class members are defined
- The **implementation** file (.cxx), in which the methods of your analysis are implemented
- An **AddTask.C** macro which creates an **instance** of your class and **configures** it.

Switch to ‘live’ broadcast

AliMonitor



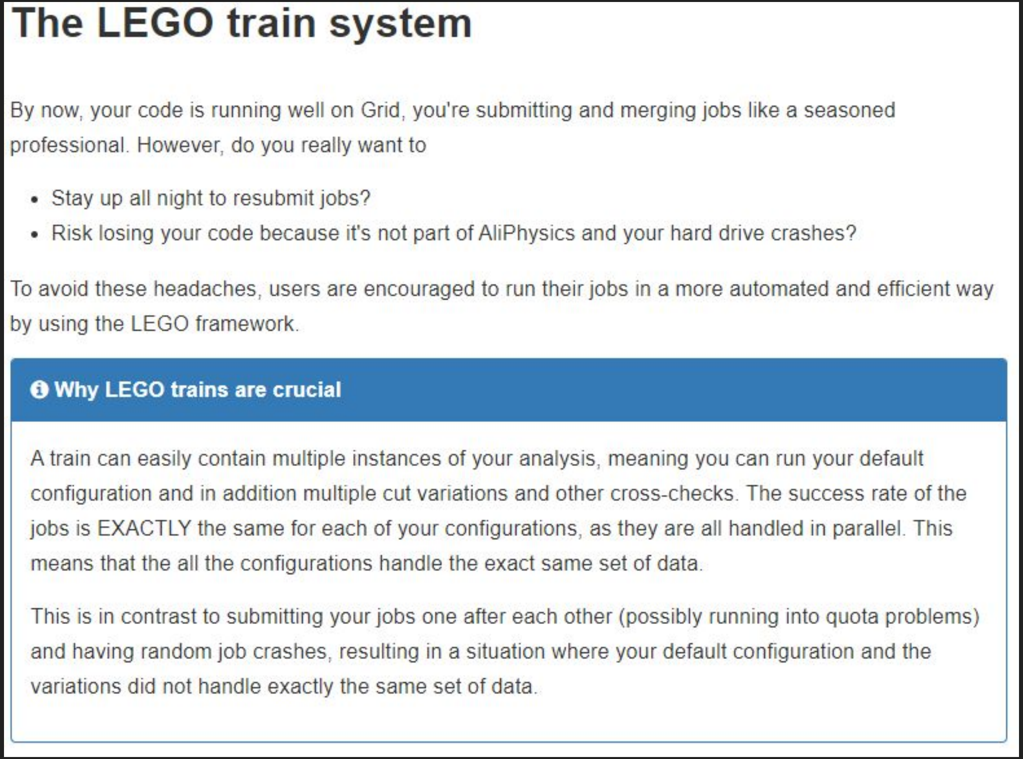
Running on Grid

The Worldwide LHC Computing Grid, usually simply referred to as Grid, is a global collaboration of computer centers. It has been up and running since 2002, to provide resources to store, distribute and analyze the petabytes of data that are generated by the LHC.

On the picture below, from http://alimonitor.cern.ch/map.jsp, you can see the distribution of computing centers that are working for the ALICE collaboration. We will the Grid to run your analysis on a larger scale than your laptop can provide.

Switch to ‘live’ broadcast

AliMonitor



The LEGO train system

By now, your code is running well on Grid, you're submitting and merging jobs like a seasoned professional. However, do you really want to

- Stay up all night to resubmit jobs?
- Risk losing your code because it's not part of AliPhysics and your hard drive crashes?

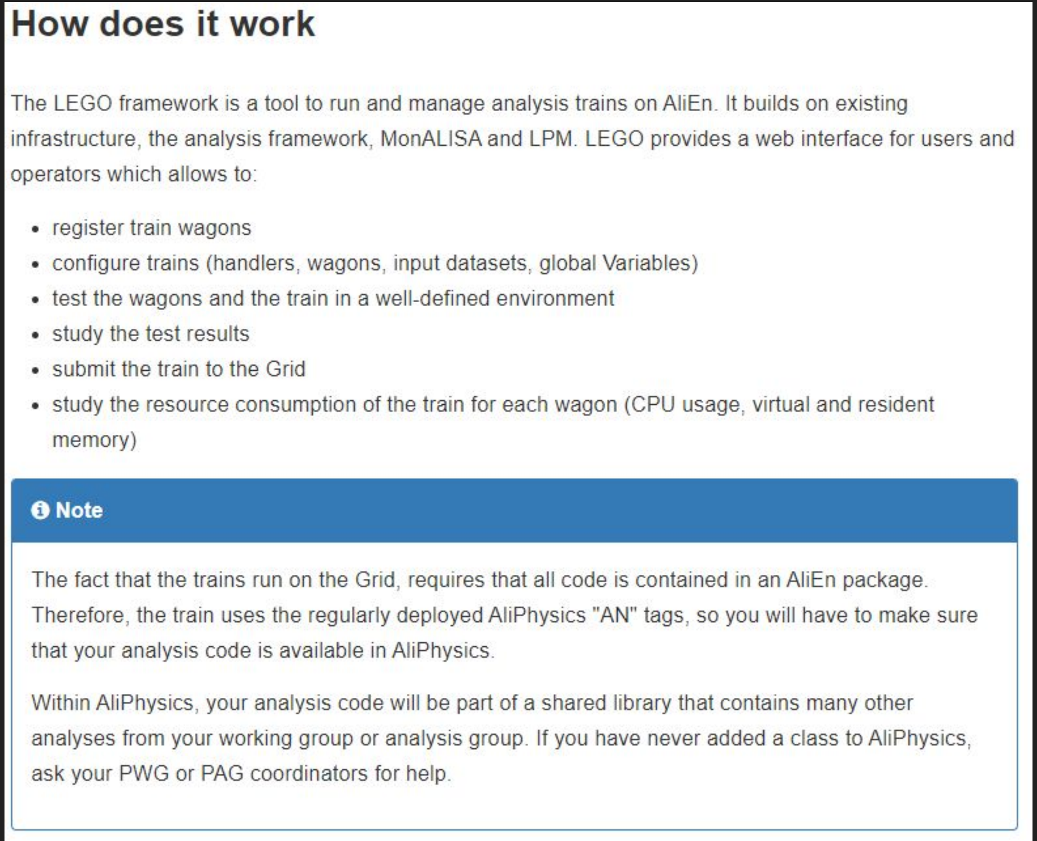
To avoid these headaches, users are encouraged to run their jobs in a more automated and efficient way by using the LEGO framework.

Why LEGO trains are crucial

A train can easily contain multiple instances of your analysis, meaning you can run your default configuration and in addition multiple cut variations and other cross-checks. The success rate of the jobs is EXACTLY the same for each of your configurations, as they are all handled in parallel. This means that the all the configurations handle the exact same set of data.

This is in contrast to submitting your jobs one after each other (possibly running into quota problems) and having random job crashes, resulting in a situation where your default configuration and the variations did not handle exactly the same set of data.

AliMonitor



How does it work

The LEGO framework is a tool to run and manage analysis trains on AliEn. It builds on existing infrastructure, the analysis framework, MonALISA and LPM. LEGO provides a web interface for users and operators which allows to:

- register train wagons
- configure trains (handlers, wagons, input datasets, global Variables)
- test the wagons and the train in a well-defined environment
- study the test results
- submit the train to the Grid
- study the resource consumption of the train for each wagon (CPU usage, virtual and resident memory)

Note

The fact that the trains run on the Grid, requires that all code is contained in an AliEn package. Therefore, the train uses the regularly deployed AliPhysics "AN" tags, so you will have to make sure that your analysis code is available in AliPhysics.

Within AliPhysics, your analysis code will be part of a shared library that contains many other analyses from your working group or analysis group. If you have never added a class to AliPhysics, ask your PWG or PAG coordinators for help.

Switch to ‘live’ broadcast

How to do analysis in ALICE - Quick reference

Main website for information: alice-doc.github.io

1. **Setup**
   a. Register as ALICE member
   b. Get a Grid certificate
   c. Build the ALICE software
2. **Write your analysis task**
   a. Define and design your measurement
   b. Develop locally
   c. Test locally
3. **Acquire results**
   a. Run your task on (MC) data
   b. Make conclusions
4. **Present in your PAG / PWG** $\longrightarrow$ **iterate** $\longrightarrow$ **profit!**

If time permits...

Switch to ‘live’ broadcast

Further development of your analysis code