Generatori di numeri casuali e metodi Montecarlo (parte3)

Laboratorio Trattamento Numerico dei Dati Sperimentali

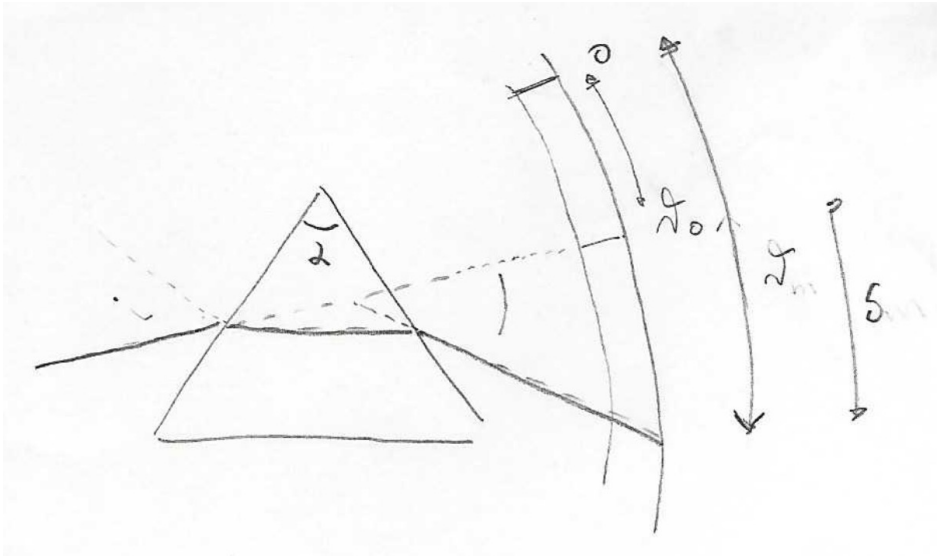
L. Carminati
Universita' degli studi di Milano

Simulazione di misure sperimentali

- Consideriamo un esperimento di misura reale che si effettua nel laboratorio di fisica e cerchiamo di utilizzare I numeri casuali per costruire una simulazione della misura:
  - Comprendere il comportamento dell'apparato di misura
  - Ottenere indicazioni su come migliorare l'apparato di misura

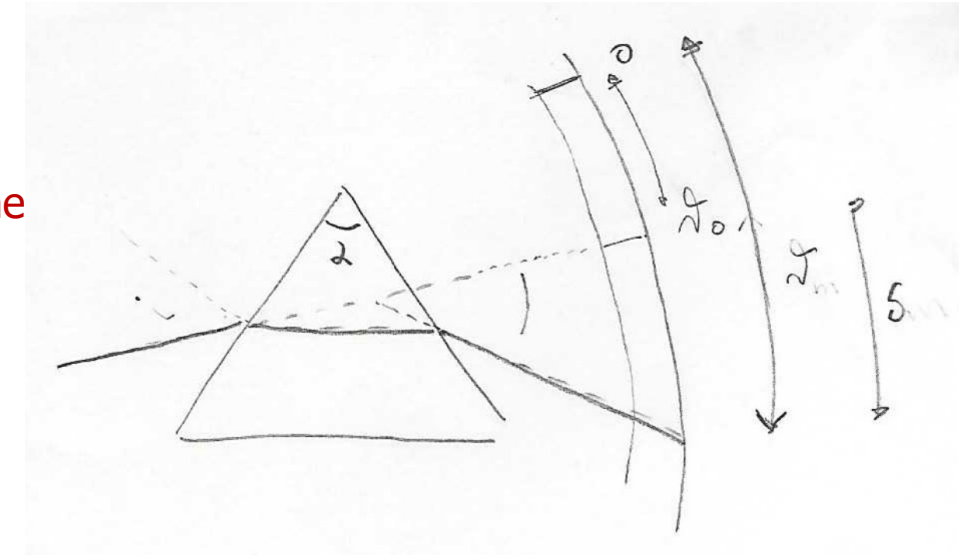
- misurazione dell'indice di rifrazione del prisma in funzione della lunghezza d'onda della luce, e la verifica sperimentale della legge di dispersione secondo la formula di Cauchy

$$n(\lambda) = \sqrt{A + \frac{B}{\lambda^2}}.$$



Simulazione di misure sperimentali : la procedura sperimentale

- Tolgo il prisma dall'apparato, posiziono la lampada e misuro θ_0 ovvero l'angolo del raggio non deflesso
- Inserisco il prisma e misuro $\theta_m(\lambda_1)$ di deflessione minimia relativo ad una certo colore (λ_1)
- Determino poi $\delta_m(\lambda_1) = \theta_m(\lambda_1) - \theta_0$



- Ora posso quindi calcolare l'indice di rifrazione come $n(\lambda_1) = \frac{\sin\left(\frac{\delta_m(\lambda_1)+\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)}$
- A questo punto posso cambiare riga e misurare $n(\lambda_2) = \frac{\sin\left(\frac{\delta_m(\lambda_2)+\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)}$
- Con due misure e due incognite posso invertire la relazione di Cauchy per determinare i coefficienti A e B secondo le relazioni

$$A = \frac{\lambda_2^2\, n(\lambda_2)^2 - \lambda_1^2\, n(\lambda_1)^2}{\lambda_2^2 - \lambda_1^2} \qquad B = \frac{n(\lambda_2)^2 - n(\lambda_1)^2}{\frac{1}{\lambda_2^2} - \frac{1}{\lambda_1^2}}$$

- Calcolo gli errori su A (σ_A) e B (σ_B): propagazione degli errori con correlazione θ_0 correla i $\delta_m(\lambda_1)$!

Costruzione di una simulazione

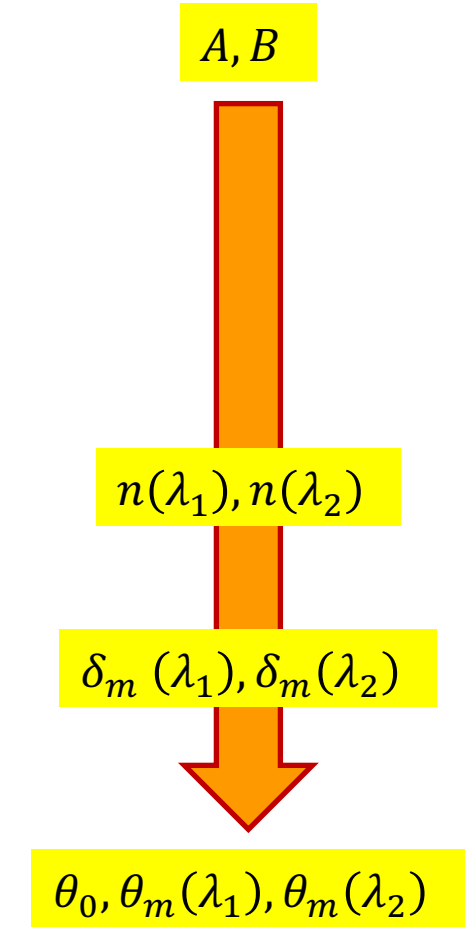
- Si procede a rovescio : si assume di conoscere il risultato della misura (nel nostro caso A e B)
  - La cosa non ci deve stupire : la simulazione non e' predittiva, va utilizzata per la comprensione dell'apparato sperimentale (esempio, stima degli errori che ci aspettiamo)
- Conosciamo i valori "veri" di A e B
- Calcoliamo i valori "veri" degli indici di rifrazione per le lunghezze d'onda di test λ_1 e λ_2

$$n(\lambda_1) = \sqrt{A + \frac{B}{\lambda_1}} \quad \text{e} \quad n(\lambda_2) = \sqrt{A + \frac{B}{\lambda_2}}$$

- Calcoliamo

$$\delta_m(\lambda_1) = 2\text{asin}(n(\lambda_1)\sin\frac{\alpha}{2} - \alpha) \quad \text{e} \quad \delta_m(\lambda_2) = 2\text{asin}(n(\lambda_2)\sin\frac{\alpha}{2} - \alpha)$$

- Scelgo un θ_0 arbitrario
- Calcolo $\theta_m(\lambda_1) = \theta_0 + \delta_m(\lambda_1)$ e $\theta_m(\lambda_2) = \theta_0 + \delta_m(\lambda_2)$



Esecuzione della simulazione

L'esecuzione della simulazione procede esattamente ripercorrendo i passi che lo sperimentatore avrebbe fatto in laboratorio

$\theta_0, \theta_m(\lambda_1), \theta_m(\lambda_2)$

- Genero le pseudo-misure : a partire dalle misure "vere" della slide precedente simulo il processo di misura con una smearing gaussiano di larghezza pari all'incertezza σ_θ sulla misura degli angoli

$$\theta_0^{mis} = Rand.Gaus(\theta_0, \sigma_\theta)$$
$$\theta_m^{mis}(\lambda_1) = Rand.Gaus(\theta_m(\lambda_1), \sigma_\theta)$$
$$\theta_m^{mis}(\lambda_2) = Rand.Gaus(\theta_m(\lambda_2), \sigma_\theta)$$

$\theta_0^{mis}, \theta_m^{mis}(\lambda_1), \theta_m^{mis}(\lambda_2)$

- Determino poi $\delta_m^{mis}(\lambda_1) = \theta_m^{mis}(\lambda_1) - \theta_0^{mis}$ e $\delta_m^{mis}(\lambda_2) = \theta_m^{mis}(\lambda_2) - \theta_0^{mis}$

$\delta_m^{mis}(\lambda_1), \delta_m^{mis}(\lambda_2)$

- Determiniamo $n^{mis}(\lambda_1) = \frac{\sin\left(\frac{\delta_m^{mis}(\lambda_1)+\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)}$ e $n^{mis}(\lambda_2) = \frac{\sin\left(\frac{\delta_m^{mis}(\lambda_2)+\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)}$

$n^{mis}(\lambda_1), n^{mis}(\lambda_2)$

- A questo punto con due misure e due incognite posso invertire la relazione di Cauchy per determinare i coefficienti

$$A^{mis} = \frac{\lambda_2^2\, n^{mis}(\lambda_2)^2 - \lambda_1^2\, n^{mis}(\lambda_1)^2}{\lambda_2^2 - \lambda_1^2} \qquad B^{mis} = \frac{n^{mis}(\lambda_2)^2 - n^{mis}(\lambda_1)^2}{\frac{1}{\lambda_2^2} - \frac{1}{\lambda_1^2}}$$

A^{mis}, B^{mis}

Mettiamo tutto insieme

Posso ripetere la simulazione un gran numero di volte

A, B

$n(\lambda_1), n(\lambda_2)$

$\delta_m\ (\lambda_1), \delta_m(\lambda_2)$

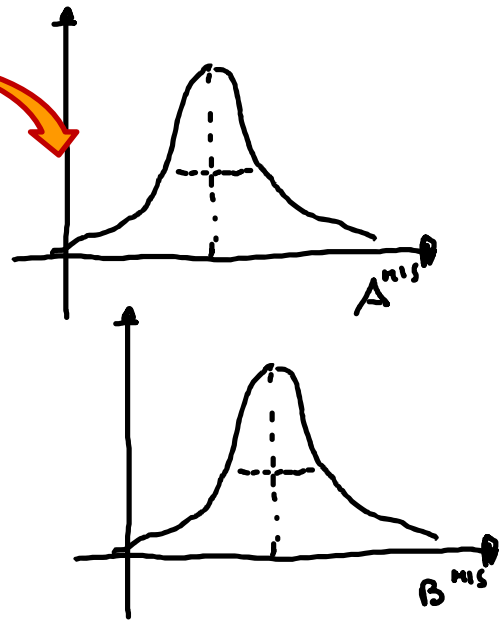
$\theta_0, \theta_m(\lambda_1), \theta_m(\lambda_2)$

$\theta_0^{mis}, \theta_m^{mis}(\lambda_1), \theta_m^{mis}(\lambda_2)$

$\delta_m^{mis}\ (\lambda_1), \delta_m^{mis}(\lambda_2)$

$n^{mis}(\lambda_1), n^{mis}(\lambda_2)$

A^{mis}, B^{mis}



Posso costruire la distribuzione degli A^{mis} e B^{mis} e usarle per stimare gli errori su A (σ_A) e B (σ_B)

Implementazione

Costruttore : inizializza tutte le variabili "vere" della simulazione

Esegui : genera la pseudomisura

Analizza : a partire dalla pseudomisura calcola A e B

Generatore di numeri casuali, il vero motore della simulazione

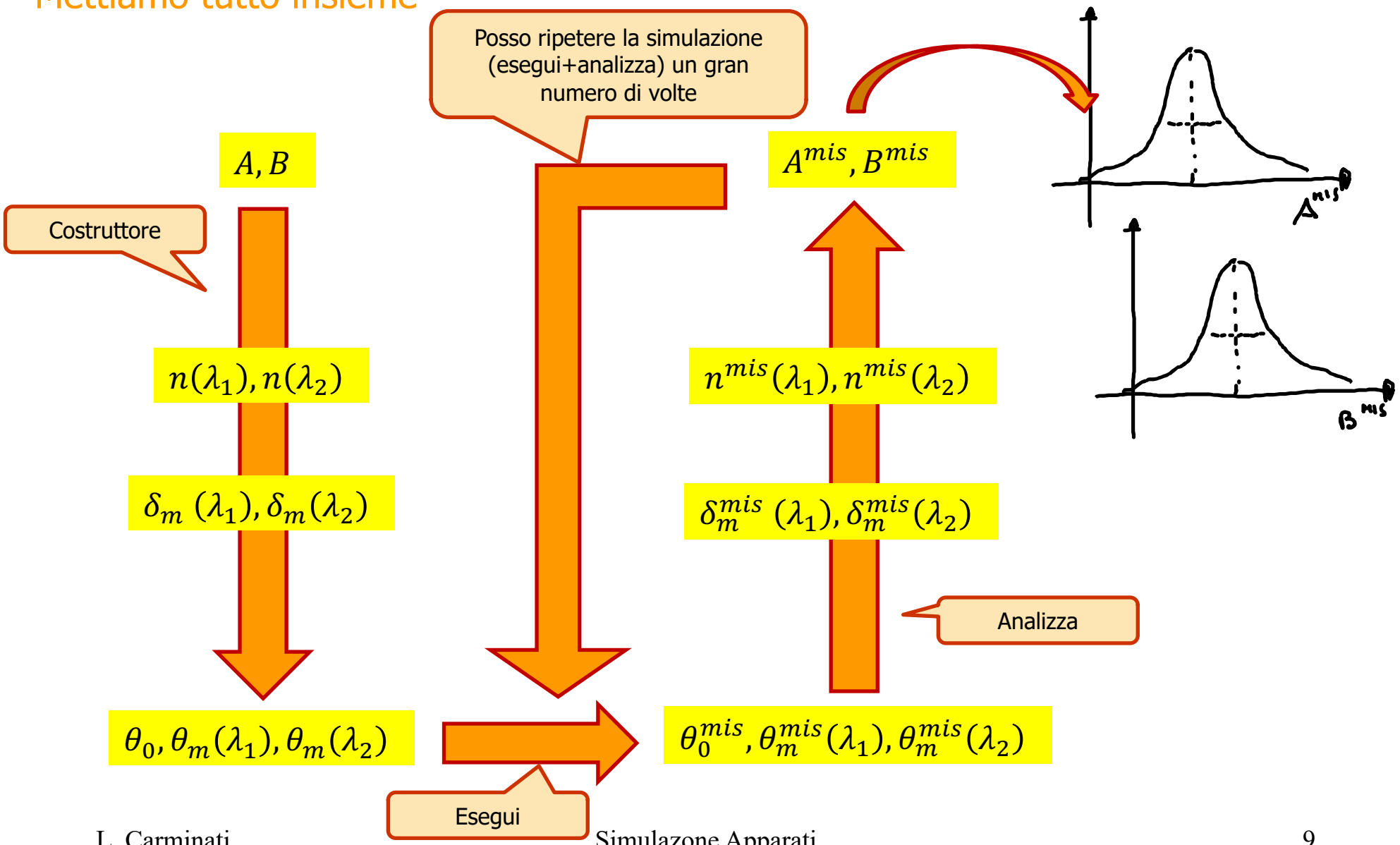
```
#ifndef _Esperimento_Prisma_h_
#define _Esperimento_Prisma_h_

#include "RandomGen.h"

class EsperimentoPrisma {

 public :
  EsperimentoPrisma();
  ~EsperimentoPrisma();

  void Esegui();
  void Analizza();

 private:

  // generatore di numeri casuali

  RandomGen m_rgen;

  // parametri dell'apparato sperimentale

  double m_lambda1, m_lambda2, m_alpha, m_sigmat;

  // valori delle quantita' misurabili :
  // _input    : valori assunti come ipotesi nella simulazione
  // _misurato : valore dopo la simulazione di misura

  double m_A_input, m_A_misurato;
  double m_B_input, m_B_misurato;
  double m_n1_input, m_n1_misurato;
  double m_n2_input, m_n2_misurato;
  double m_th0_input, m_th0_misurato;
  double m_th1_input, m_th1_misurato;
  double m_th2_input, m_th2_misurato;

};

#endif
```

Implementazione

Costruttore : lista di inizializzazione per inizializzare le variabili (indispensabile per m_rgen)

```
#include "EsperimentoPrisma.h"

EsperimentoPrisma::EsperimentoPrisma() :
  m_rgen(1),
  m_lambda1(579.1E-9),
  m_lambda2(404.7E-9),
  m_alpha(60.*M_PI/180.),
  m_sigmat(0.3E-3),
  m_A_input(2.7),
  m_B_input(60000E-18)
{

  // calcolo degli indici di rifrazione attesi

  m_n1_input = sqrt( m_A_input + m_B_input / (m_lambda1*m_lambda1) ) ;
  m_n2_input = sqrt( m_A_input + m_B_input / (m_lambda2*m_lambda2) ) ;

  // theta0 e' arbitrario, scelgo M_PI/2.

  m_th0_input = M_PI/2. ;

  // determino theta1 e theta2

  double dm ;
  dm = 2.*asin( m_n1_input * sin (0.5 * m_alpha) ) - m_alpha ;
  m_th1_input = m_th0_input + dm ;
  dm = 2.*asin( m_n2_input * sin (0.5 * m_alpha) ) - m_alpha ;
  m_th2_input = m_th0_input + dm ;

}
```

Mettiamo tutto insieme



Implementazione

```
#include "EsperimentoPrisma.h"
#include "TApplication.h"
#include "TH1F.h"

int main() {

  TApplication app("app",0,0);

  unsigned int nsimul = 1000;

  EsperimentoPrisma p ;

  TH1F hA("A","A",100, 2.68,2.72 ) ;

  for ( int k=0 ; k < nsimul ; k++ ) {

    p.Esegui();
    p.Analizza();
    hA.Fill(p.getAmis() ) ;

  }

  hA.Draw();
  app.Run();

}
```

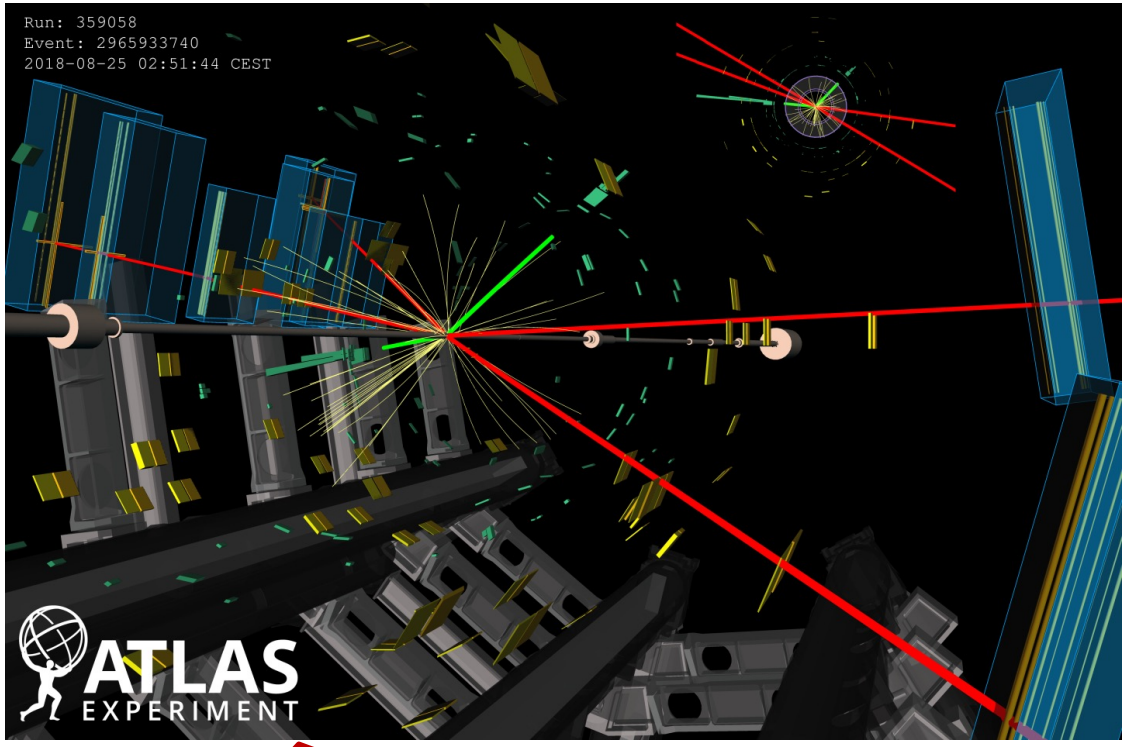
Costruisco un oggetto di tipo EsperimentoPrisma

Ciclo sulle simulazione

Due steps di simulazione: Esegui() e Analizza()

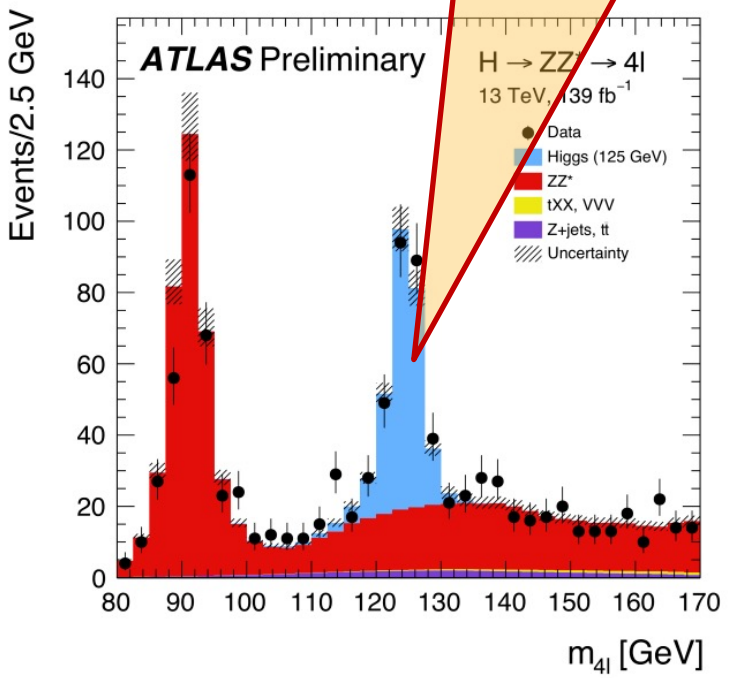
Dopo ogni simulazione accedo al valore di A "misurato"

Generatori di numeri casuali



Un evento reale misurato nel rivelatore ATLAS di possible decadimento del bosone di Higgs in 2e2μ in associazione ad un bosone Z che decade in due μ

In blu : una simulazione di cosa avrei dovuto vedere dall'analisi dei dati raccolti dal rivelatore ATLAS se esistesse un bosone di Higgs come predetto dalla teoria con una massa di 125 GeV.



Conclusive remarks

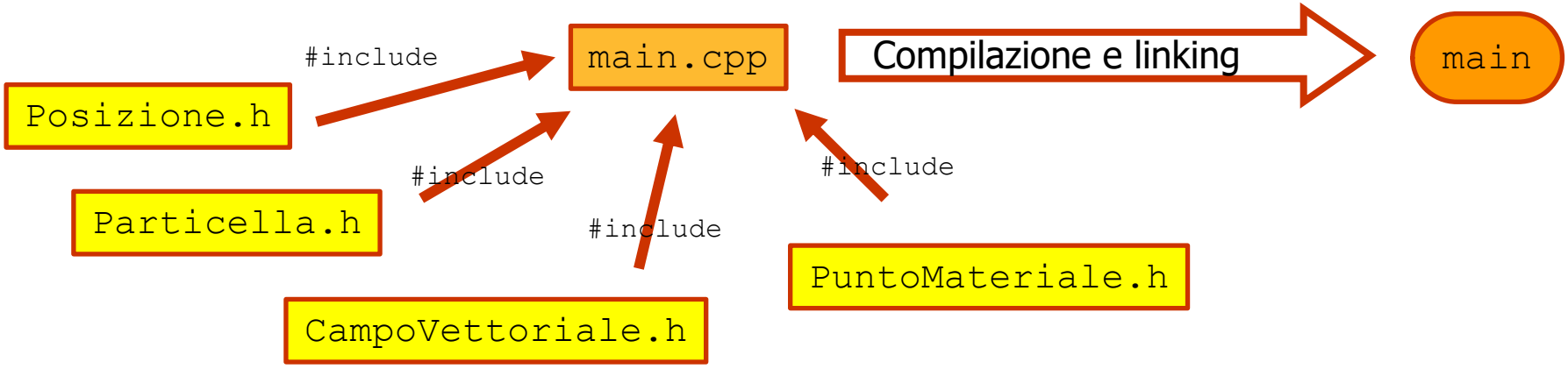
- Abbiamo cercato di mettere in evidenza l'importanza dell'approccio numerico nell'affrontare alcuni argomenti di fisica generale
- Abbiamo studiato il C++ (e ROOT per la visualizzazione): non e' l'unico linguaggio disponibile ma probabilmente e' quello piu' complesso (quindi didatticamente funziona bene). I tools di visualizzazione sono vari.
- Esame : cercate di farlo il prima possible !
  - Consegna esercizi (preferibilemente via replit oppure con consegna diretta vedi pagina web corso)
  - Esame scritto : presumibilmente in presenza, caricate tutto il codice anche su replit. La situazione e' molto fluida, attenzione ai messaggi su ARIEL
  - Esame orale : segnalatemi tutte le inesattezze che avete riscontrato nelle trasparenze vorrei rilasciare una versione revisionata il prima possible.
- Speriamo di aver gettato le basi di un vostro futuro coinvolgimento in argomenti di calcolo e simulazione numerica che verranno approfonditi e resi piu' specifici nei corsi successivi.


That's all Folks!

Compilazione : option 1

- Codice composto da : main.cpp, Particella.h, PuntoMateriale.h, Posizione.h, CampoVettoriale.h

```
LIBS = $(shell root-config --libs)
INCS = $(shell root-config --cflags)

main: main.cpp Posizione.h Particella.h CampoVettoriale.h PuntoMateriale.h
	g++ -o main main.cpp ${INCS} ${LIBS}
```



- Vantaggi : un solo file (.h), una sola istruzione di compilazione
- Svantaggi : una modifica ad una qualsiasi delle classi o funzioni causa una ricompilazione totale. Leggibilita' del codice .h puo' diventare complicata

Compilazione : option 2

- Codice composto da : main.cpp, Particella.cxx, PuntoMateriale.cxx, Posizione.cpp, CampoVettoriale.cpp (e i relativi relative files .h)

```
LIBS = $(shell root-config --libs)
INCS = $(shell root-config --cflags)

main: main.cpp Posizione.h Particella.h CampoVettoriale.h PuntoMateriale.h Posizione.cxx Particella.cxx CampoVettoriale.cxx PuntoMateriale.cxx
	g++ -o main main.cpp Posizione.cxx Particella.cxx CampoVettoriale.cxx PuntoMateriale.cxx${INCS} ${LIBS}
```

- Vantaggi : una sola istruzione di compilazione. Leggibilita' del codice migliore (?)
- Svantaggi : una modifica ad una qualsiasi delle classi o funzioni causa una ricompilazione totale.

Compilazione : option 3

- Codice composto da : main.cpp, Particella.cxx, PuntoMateriale.cxx, Posizione.cpp, CampoVettoriale.cpp (e relativi files .h)

${LIBS} is needed in the linking phase (link root pre-compiled libraries)

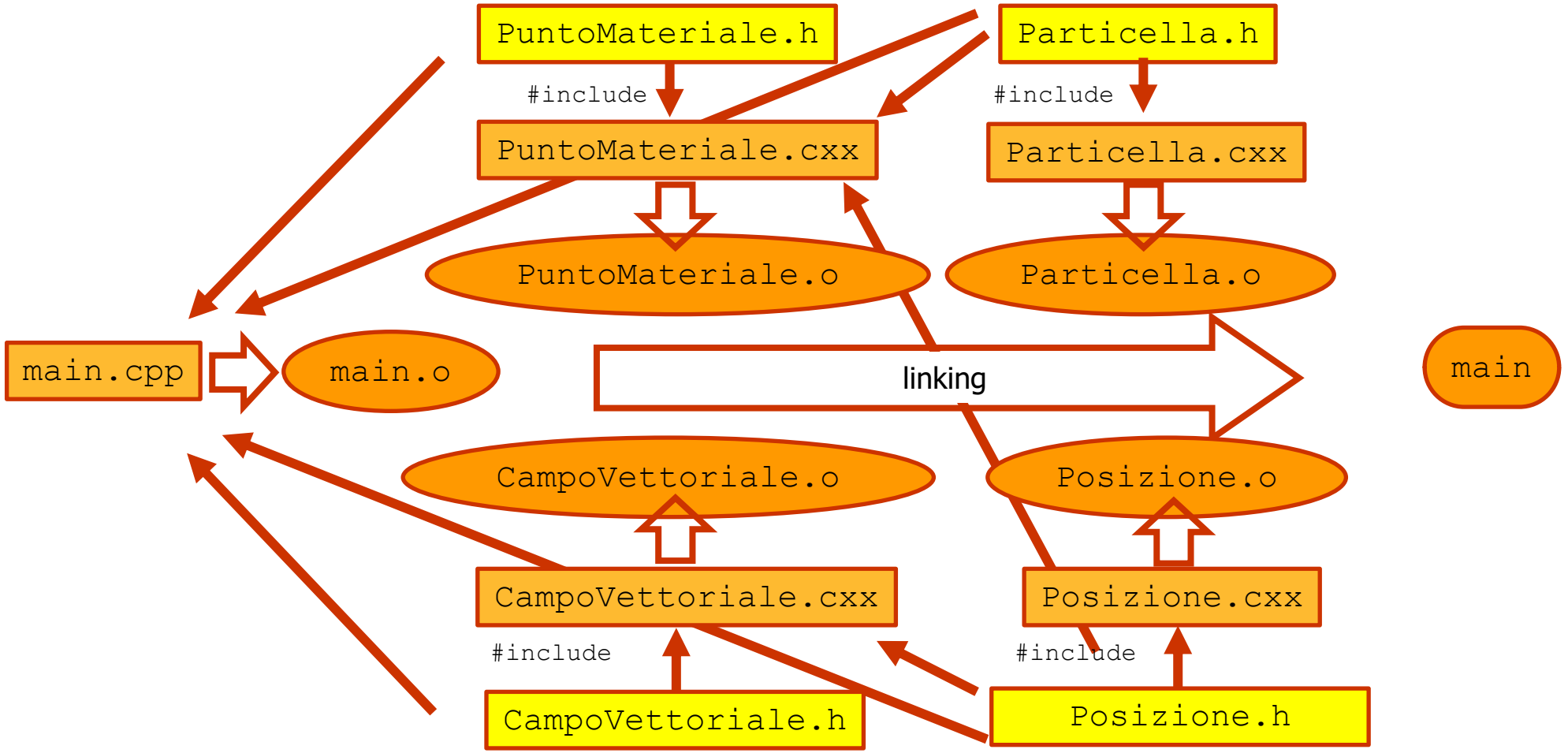
```
LIBS = $(shell root-config --libs)
INCS = $(shell root-config --cflags)

main: main.o Posizione.o Particella.o CampoVettoriale.o PuntoMateriale.o
        g++ -o main main.o Posizione.o Particella.o CampoVettoriale.o PuntoMateriale.o ${LIBS}

main.o: main.cpp Posizione.h Particella.h
        g++ -c -o main.o main.cpp ${INCS}

Posizione.o: Posizione.cxx Posizione.h
        g++ -c -o Posizione.o Posizione.cxx  ${INCS}

Particella.o: Particella.cxx Particella.h Posizione.h
        g++ -c -o Particella.o Particella.cxx  ${INCS}

CampoVettoriale.o: CampoVettoriale.cxx CampoVettoriale.h Posizione.h
        g++ -c -o CampoVettoriale.o CampoVettoriale.cxx ${INCS}

PuntoMateriale.o: PuntoMateriale.cxx PuntoMateriale.h Posizione.h Particella.h CampoVettoriale.h
        g++ -c -o PuntoMateriale.o PuntoMateriale.cxx ${INCS}

clean:
        rm main
        rm *.o
```

Option –c means no linking

${INCS} is needed when compiling user code which makes use of ROOT objects (specify where the .h are)

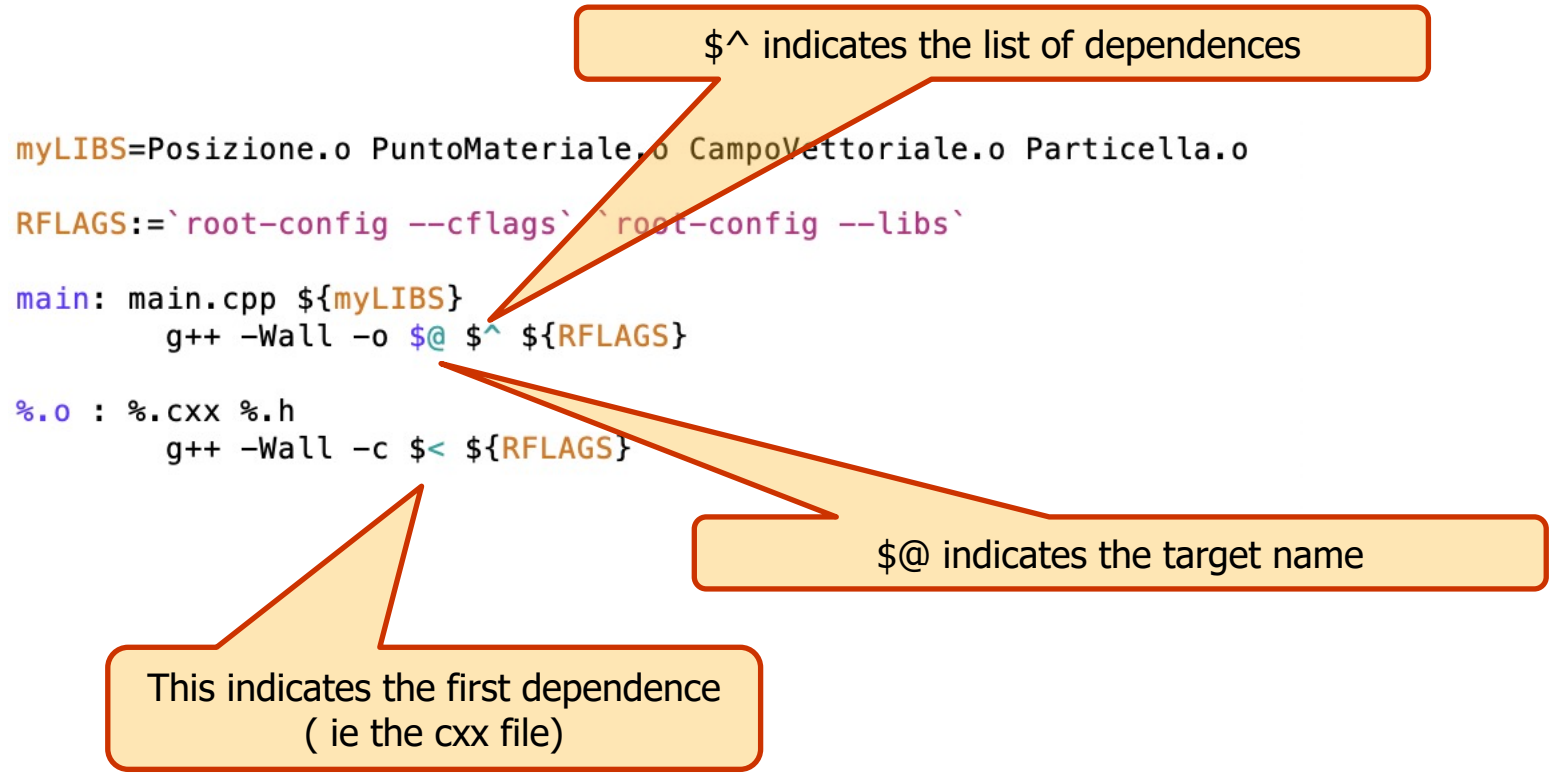
- Vantaggi : una modifica ad una qualsiasi delle classi o funzioni causa una ricompilazione parziale
- Svantaggi : makefile piu' complicato

Compilazione (II)



Tired about your Makefile getting longer and longer ? Try this !

Define a sort of "general" rule to compile the required .o



If written in this way no multiple dependencies on .h are considered !

Tired about your Makefile getting longer and longer ? Try this !

Define a sort of "general" rule to compile the required .o

```
myLIBS=Posizione.o PuntoMateriale.o CampoVettoriale.o Particella.o
myDEPS=Posizione.h PuntoMateriale.h CampoVettoriale.h Particella.h

RFLAGS:=`root-config --cflags` `root-config --libs`

main: main.cpp ${myLIBS}
        g++ -Wall -o $@ $^ ${RFLAGS}

%.o : %.cxx %.h ${myDEPS}
        g++ -Wall -c $< ${RFLAGS}
```

Specify the list of dependencies : not really efficient, a .cxx will be re-compiled each time any .h is modified

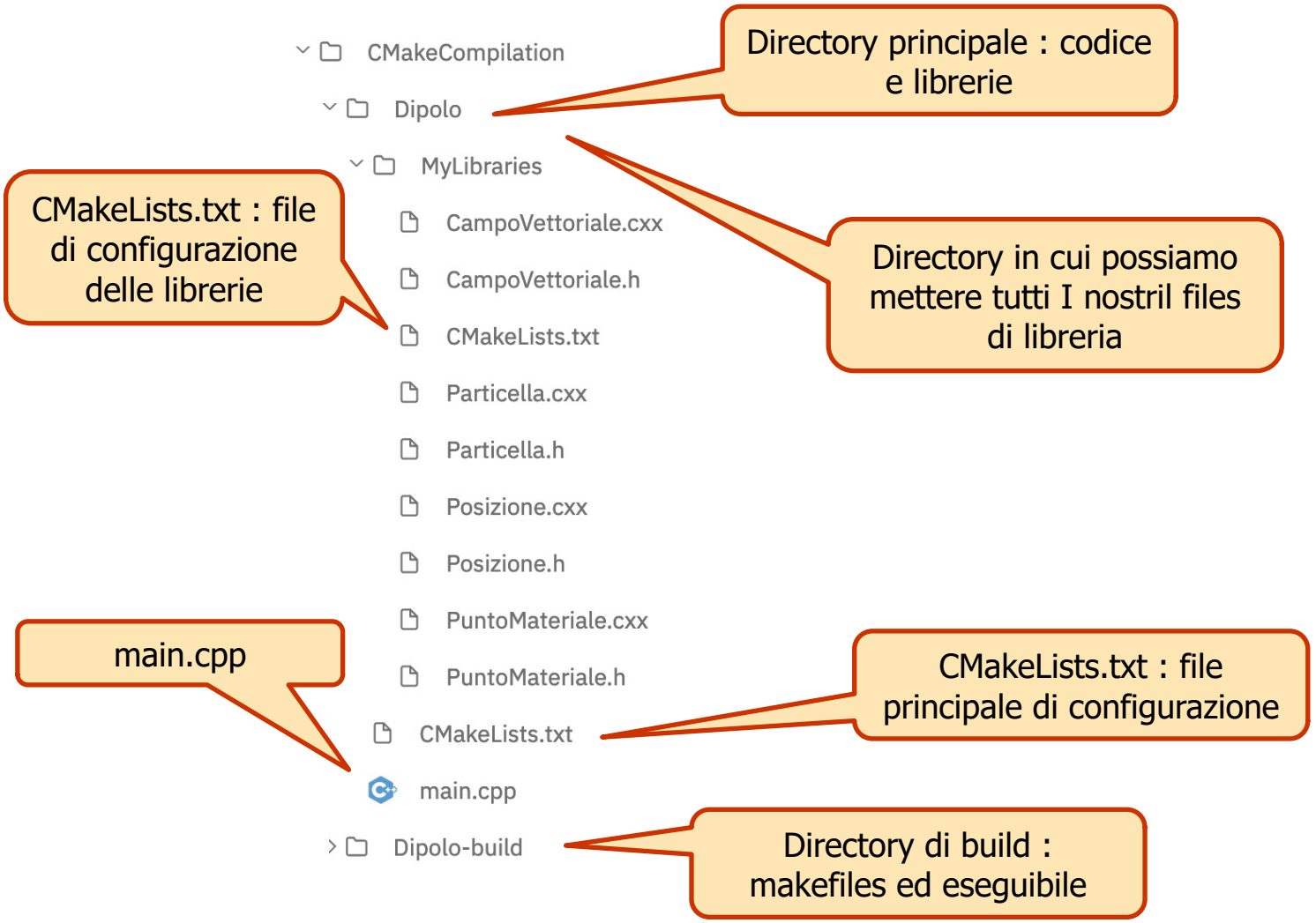
Not very elegant but should work at first order

More complex strategies : cmake

"CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice"

- Automatizza la creazione dei makefile : in generale non sono portabili, occorre controllare che certi compilatori, librerie etc siano istallati e quali sono i path. CMake permette di costruire il makefile giusto su ogni piattaforma

More complex strategies : cmake



More complex strategies : cmake

```
1   cmake_minimum_required(VERSION 3.10)
2
3   # set the project name
4   project(Esercizio53)
5
6   # specify the C++ standard
7   set(CMAKE_CXX_STANDARD 11)
8   set(CMAKE_CXX_STANDARD_REQUIRED True)
9
10  add_subdirectory(MyLibraries)
11
12  list(APPEND CMAKE_PREFIX_PATH $ENV{ROOTSYS})
13  find_package(ROOT REQUIRED)
14  include(${ROOT_USE_FILE})
15
16  # add the executable
17  add_executable(main main.cpp)
18
19  target_link_libraries(main ${ROOT_LIBRARIES} Particella CampoPosizione)
20
21  target_include_directories(main PUBLIC "${PROJECT_BINARY_DIR}" "$
    {PROJECT_SOURCE_DIR}/MyLibraries")
```

Specifichiamo che vogliamo un compilatore che supporti C++11

Specifichiamo che ci saranno delle librerie da caricare da questa directoy

Verifica che ROOT sia installato e genera le configurazioi necessarie

Creare l'eseguibile

Quali librerie vanno linkate

Dove cercare gli include files (.h)