CBM Simulation&Analysis Framework
Cbmroot

Mohammad Al-Turany

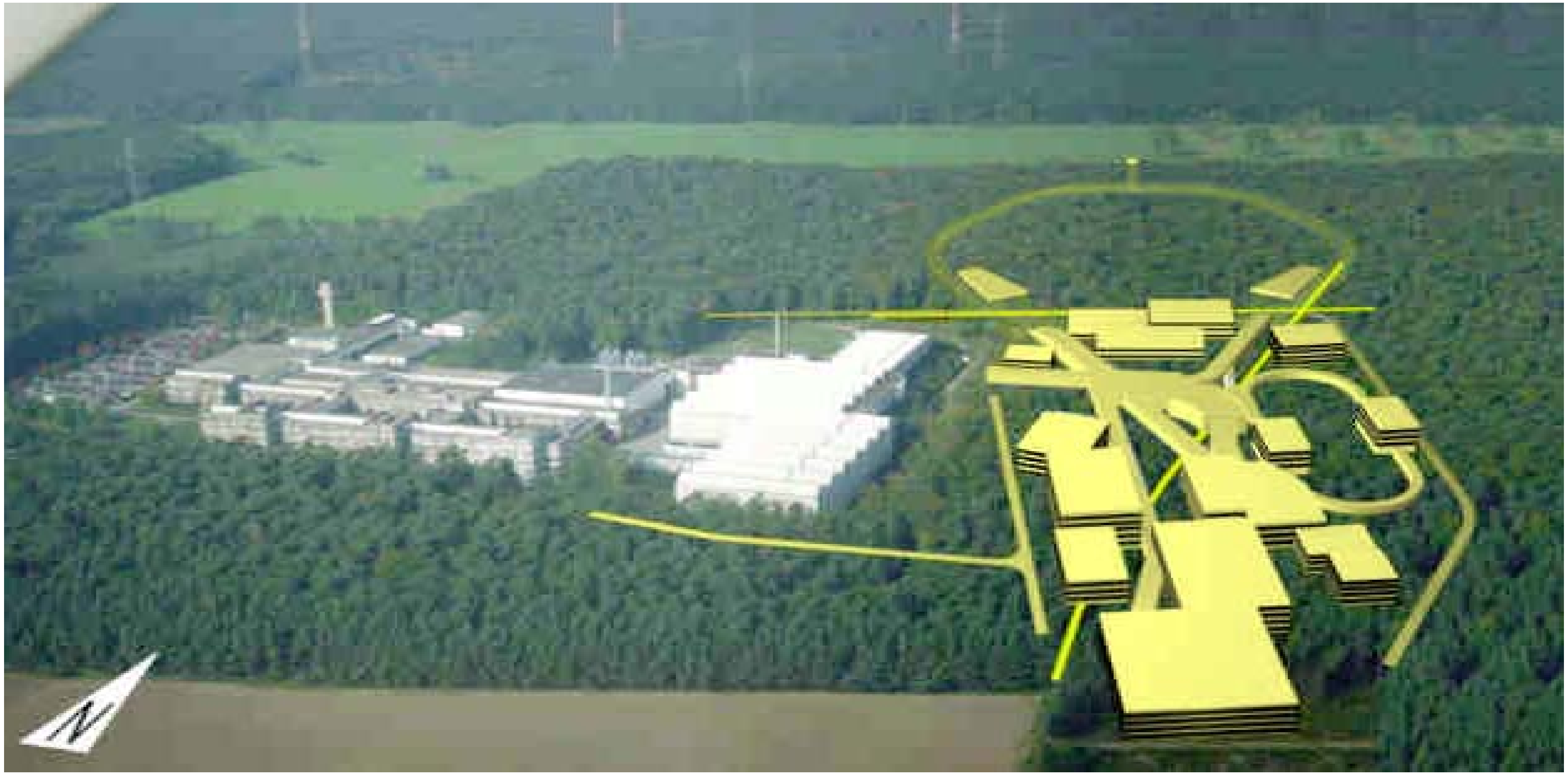
Denis Bertini

Ilse König

Overview

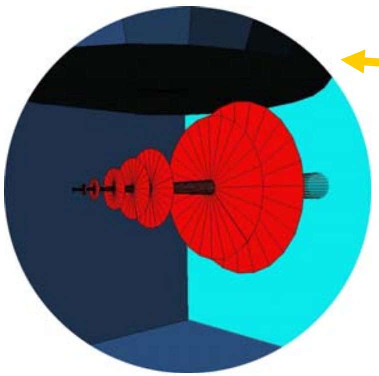
- CbmRoot Features
- Geometry Interface
- Runtime Database and Parameter Handling
- Examples (Simulation and Analysis)
- Summary

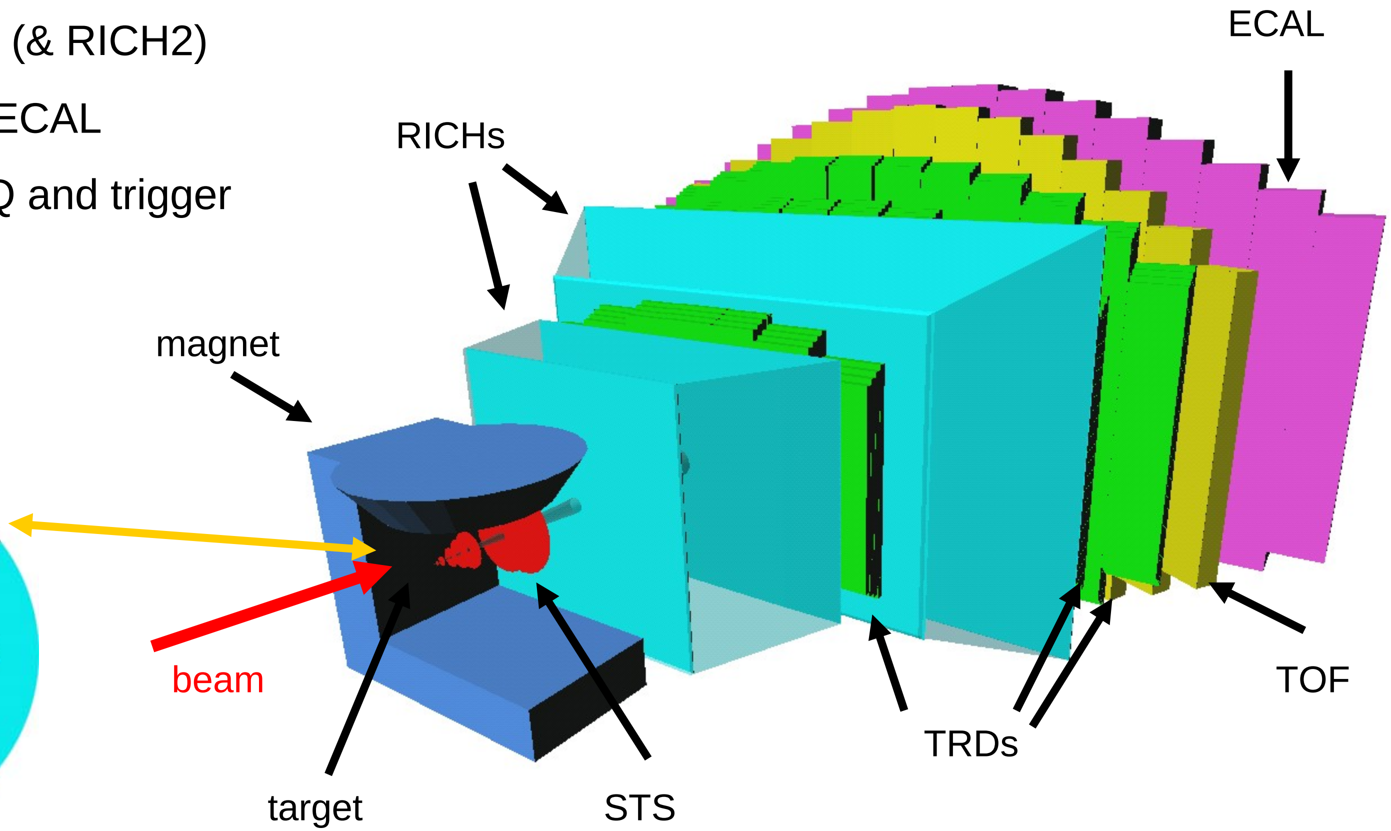
The FAIR project



CBM experiment

- tracking, vertex reconstruction: radiation hard silicon pixel/strip detectors (STS) in a magnetic dipole field
- electron ID: RICH1 & TRD (& ECAL) $\rightarrow \pi$ suppression $\geq 10^4$
- hadron ID: TOF (& RICH2)
- photons, π^0, μ: ECAL
- high speed DAQ and trigger





Features

- **The same framework** can be used for Simulation and Analysis
- **Fully ROOT based:**
  - **VMC** for simulation
  - **IO scheme** (TChain, friend TTrees, TFolders) for persistency
  - **TTask** to organize the analysis data flow

- **Completely configurable** via ROOT macros
- **Easy to maintain** (only ROOT standard services are used)

- **Reuse of HADES Geometry Interface.**
  - **G3 Native geometry**
  - **Geometry Modeller (TGeoManager)**

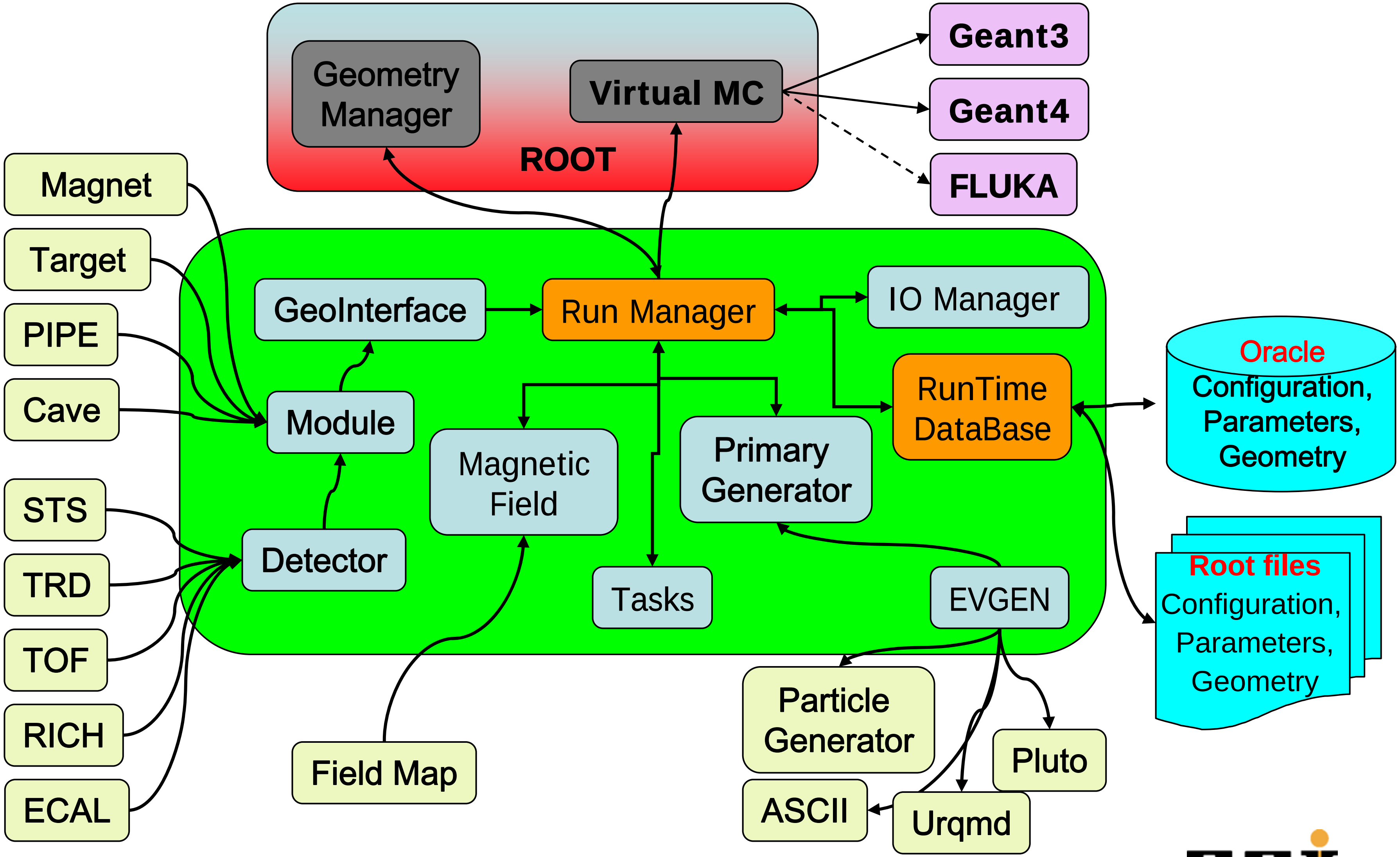
Configuration and building

Use of Autoconf/Automake

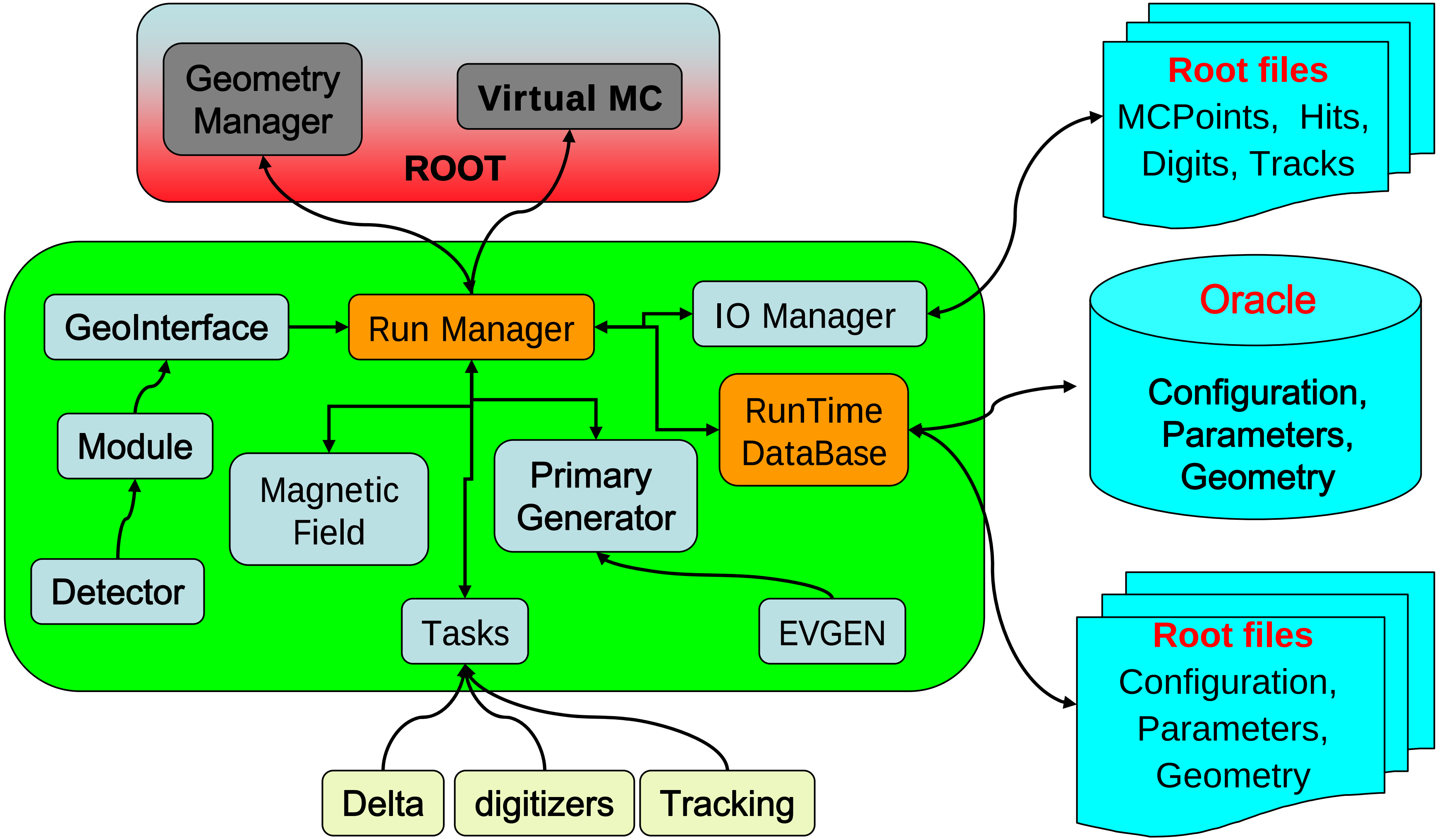
- Configure script generating Makefiles
  - ./configure –prefix=$PWD –enable-geant3 –enable-geant4
  - Generating corresponding config.sh

- Possiblity to configure the package with/without geant3/geant4
  - Automatic checking of correct packages
    - Gcc , Root, Geant3, Geant4, VMC ect …
  - Automation of additional checks can be done easily.

CBM Simulation



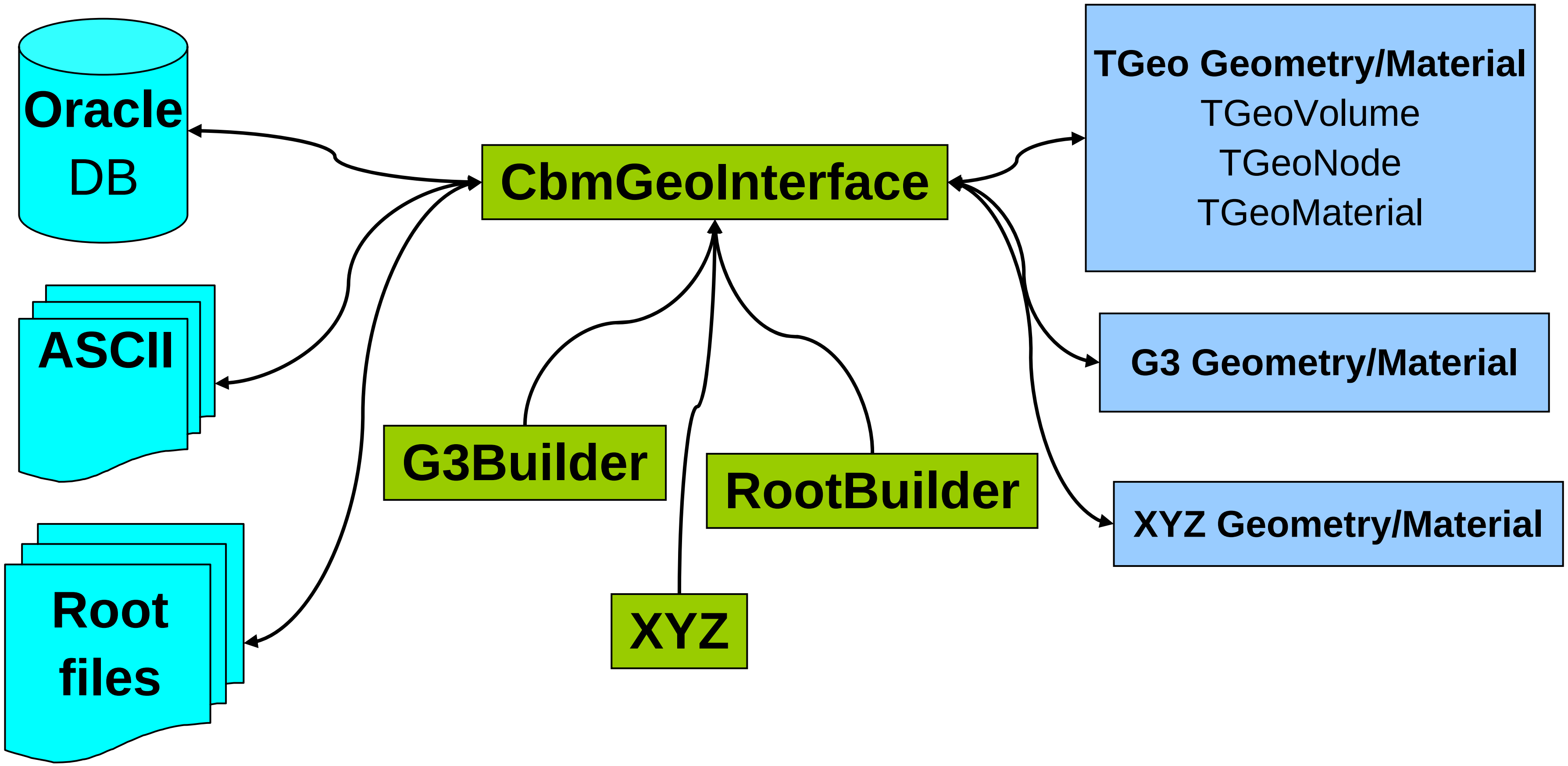
CBM Analysis



CBM Geometry Interface

- Use the copy mechanism
  - reduce the size of ASCII files
  - Reduce the number of Volumes in Geant
  - Improve Geant tracking performance
- Oracle interface
  - Hades geometry table design reusable
- Advantage:
  - more flexibility : different inputs can be used.
  - closer to technical drawings and analysis coordinate systems

Material & Geometry Interface



Runtime Database

- The runtime database is not a database in the classical sense, but a parameter manager.
- It knows the I/Os defined by the user in the macro and all parameter containers needed for the actual analysis and/or Simulation.
- It manages the automatic initialization and saving of the parameter containers
- After all initialization the complete list of runs and related parameter versions are saved either to Oracle or to ROOT files.

Runtime Database

The Runtime Database is the manager class for all Parameter containers:

Creation, Initialization, Output

Runtime Database

Container Factories

List of Parameter Containers

List of Runs

2 Inputs

1 Output

Created in the init function of the tasks via the container factories or in the macro

Filled during initialization

ASCII File

ROOT File

Oracle

Defined in the macro

Parameter handling

- **Reuse of HADES Parameter containers.**
  - Parameters can be stored and retrieved from:
    - Root files
    - Oracle
    - ASCII files
  - Parameters are connected to a runId
  - CbmBaseParSet: used to store relevant info from a simulation run.
  - CbmDetGeoPar geometry parameters
    - Stored for each detectors in CbmDetector::ConstructGeometry()

Runtime Database: Input/Output

The class design for I/O was developed according to the following user requests:

- Provide interfaces to Oracle, ROOT files and ASCII files
- Each detector has its own classes implementing the concrete I/O
- The actually used I/O is selected directly on ROOT interpreter level
- The interface to Oracle is a shared library completely separated from all other libraries in the framework to avoid the oracle C* precompiler dependency.

Storage in Oracle

- Oracle does not know the parameter containers needed by the analysis.
- The data are stored not as 'objects' but in different tables related to each other (referential constraints)
- The tables were designed to minimize the storage of redundant information to guarantee data consistency and to save storage space.

Version management in Oracle

- For time dependent information a version management is needed which fulfills the following requirements:
  - It must be possible to get a consistent set of information for any date (e.g.the start time of a certain run).
  - To preserve the history, no information - even if wrong - should be overwritten without trace, which means that only inserts should be made, no deletes nor updates.
  - It must be possible to get an answer to the question: 'Which parameters were used when analyzing this run X years ago?' (The calibration might have been optimized several times since this date. Maybe some bugs have been detected and corrected in the mean time.)

Version management in Oracle

Time dependend entries have a time stamp (date + time with the precision of one second) in form of three columns (Format: DATE):

- **valid_since :**First date when the entry is valid.
- **valid_until :**Last date when the entry is still valid
- **invalid_since :**Date when the entry is replaced by a correct entry or a better version in case of e.g. calibration parameters and therefore gets invalid.

Example: CbmGenericParSet

Base class for most parameter containers

Advantage:

only a few lines of code to be implemented
all I/O interfaces exist already
Allows to store various types of parameters (handled by CbmParamList):
- int, float, double, strings,…
- arrays
- TObjects (classes, histograms, …)

in derived class implemented

```
class CbmParGenericSet : public CbmParSet {
public:
  CbmParGenericSet(const char* name,const char* title,const char*
      context)
        : CbmParSet(name,title,context) {}
  virtual ~CbmParGenericSet() {}
  virtual Bool_t init(CbmParIo*);
  virtual Int_t write(CbmParIo*);
  virtual void putParams(CbmParamList*)=0;
  virtual Bool_t getParams(CbmParamList*)=0;
  virtual void printParams();
```

Example: Parameter Class definition

```
class CbmParTest : public CbmParGenericSet {
public:
  Float_t p1;
  Int_t ai[5000];
  TH1F* histo1;

  CbmParTest(const char* name="CbmParTest",
             const char* title="Test class for parameter io",
             const char* context="TestDefaultContext");
  ~CbmParTest(void);
  void clear(void);
  void putParams(CbmParamList*);
  Bool_t getParams(CbmParamList*);
  ClassDef(CbmParTest,1)
};
```

Example: Parameter Class Implementation

```
CbmParTest::CbmParTest(const char* name,const char* title,const char*
    context)
        : CbmParGenericSet(name,title,context) {
  clear();
  histo1=new TH1F("h1","test histogram",100,-3,3);
  histo1->SetDirectory(0);
}

void CbmParTest::putParams(CbmParamList* l) {
  if (!l) return;
  l->add("p1",p1);
  l->addBinary("ai",ai,5000);
  l->addBinary("histo1",histo1);
}

Bool_t CbmParTest::getParams(CbmParamList* l) {
  if (!l) return kFALSE;
  if (!l->fill("p1",&p1)) return kFALSE;
  if (!l->fillBinary("ai",ai,5000)) return kFALSE;
  if (!l->fillBinary("histo1",histo1)) return kFALSE;
  histo1->SetDirectory(0);
  return kTRUE;
}
```

Simulation Macro – loading Libs

```
// Load basic libraries
gROOT->LoadMacro("$VMCWORKDIR/gconfig/basiclibs.C");
basiclibs();

// Load Cbmroot libraries
gSystem->Load("libGeoBase");
gSystem->Load("libCbm");
gSystem->Load("libPassive");
gSystem->Load("libGen");
gSystem->Load("libSts");
gSystem->Load("libTrd");
gSystem->Load("libTof");
gSystem->Load("libRich");
```

Simulation Macro

```
//create the Run Class
CbmRunSim *fRun = new CbmRunSim();

// set the MC version used
fRun->SetName("TGeant3");    //for G4 use "TGeant4"

//Choose the Geant 3 Navigation System
fRun->SetGeoModel("G3Native");

// choose an output file name
fRun->SetOutputFile("test.root");
```

Simulation Macro- Create Modules

```
CbmModule *Cave= new CbmCave("WORLD");
Cave->SetGeometryFileName("PASSIVE/CAVE", "v03a");
fRun->AddModule(Cave);

CbmModule *Target= new CbmTarget("Target");
Target->SetGeometryFileName("PASSIVE/TARGET", "v03a");
fRun->AddModule(Target);

CbmModule *Pipe= new CbmPIPE("PIPE");
Pipe->SetGeometryFileName("PASSIVE/PIPE", "v03a");
fRun->AddModule(Pipe);

CbmModule *Magnet= new CbmMagnet("MAGNET");
Magnet->SetGeometryFileName("PASSIVE/MAGNET", "v03a");
fRun->AddModule(Magnet);
```

Simulation Macro- Create Detectors

```
CbmDetector *STS= new CbmSts("STS", kTRUE);
STS->SetGeometryFileName("STS/STS", "v03c");
fRun->AddModule(STS);

CbmDetector *TOF= new CbmTof("TOF", kTRUE );
TOF->SetGeometryFileName("TOF/TOF", "v03_v10");
fRun->AddModule(TOF);

CbmDetector *TRD= new CbmTRD("TRD",kFALSE );
TRD->SetGeometryFileName("TRD/TRD", "v04b_9" );
fRun->AddModule(TRD);
```

Simulation Macro-Event Generators

```
CbmPrimaryGenerator *priGen= new CbmPrimaryGenerator();
fRun->SetGenerator(priGen);

CbmUrqmdGenerator *fGen1= new CbmUrqmdGenerator("00-03fm.100ev.f14");

CbmPlutoGenerator *fGen2= new CbmPlutoGenerator("jpsi.root");

CbmParticleGenerator *fGen3= new CbmParticleGenerator();

fRun->AddGenerator(fGen1);
fRun->AddGenerator(fGen2);
fRun->AddGenerator(fGen3);
```

Simulation Macro-Magnetic Field

```
// setting a field map
CbmField *fMagField= new CbmField("Dipole Field");

fMagField->readAsciifile("FieldIron.map"); // read ASCII file

fMagField->readRootfile("FieldIron.root"); // read Root file

// setting a constant field
CbmConstField *fMagField=new CbmConstField();
fMagField->SetFieldXYZ(0, 30 ,0 );        // values are in kG
     // MinX=-75, MinY=-40,MinZ=-12 ,MaxX=75, MaxY=40 ,MaxZ=124 );

fMagField->SetFieldRegions(-74, -39 ,-22 , 74, 39 , 160 );   // values are in cm

fRun->SetField(fMagField);
```

Simulation Macro- Run Simulation

fRun->Init(); // Initialize the simulation

Simulation:

1. Initialize the VMC (Simulation)
2. Initialize Tasks (if they are used in Simulation)

fRun->Run(NoOfEvent); //Run the Simulation

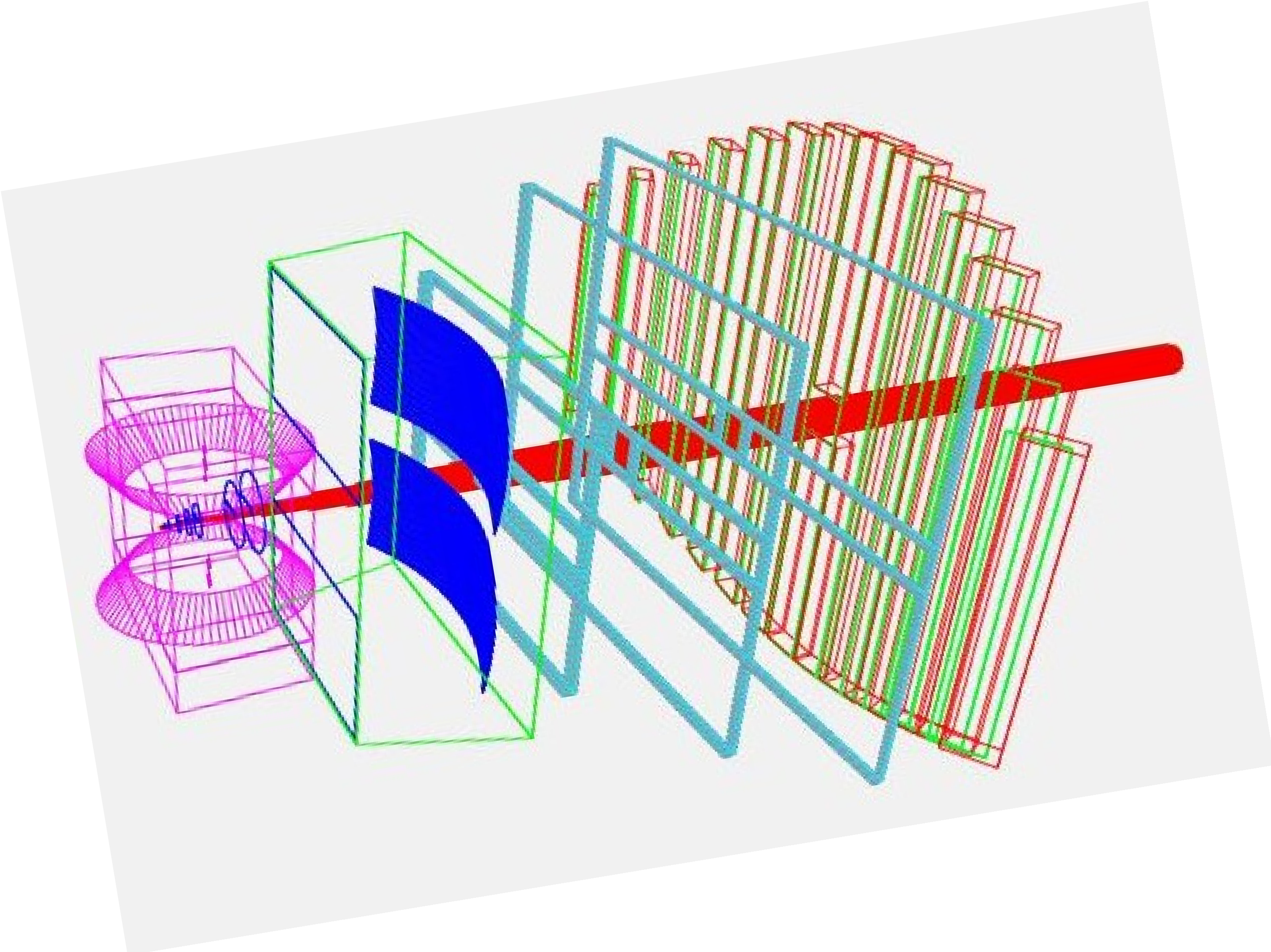
Writing Parameters to Oracle

```
gSystem->Load ( "libOra" );

CbmRuntimeDb* rtdb=fRun->GetRuntimeDb();
CbmParOraIo* ora=new CbmParOraIo;
ora->open("cbm_sts_oper");
rtdb->setOutput(ora);

CbmParTest* par=(CbmParTest*)(rtdb->getContainer("CbmParTest"));
par->setAuthor("M. Al-Turany");
par->setDescription("Analysis interface test");

par->write(ora);
```

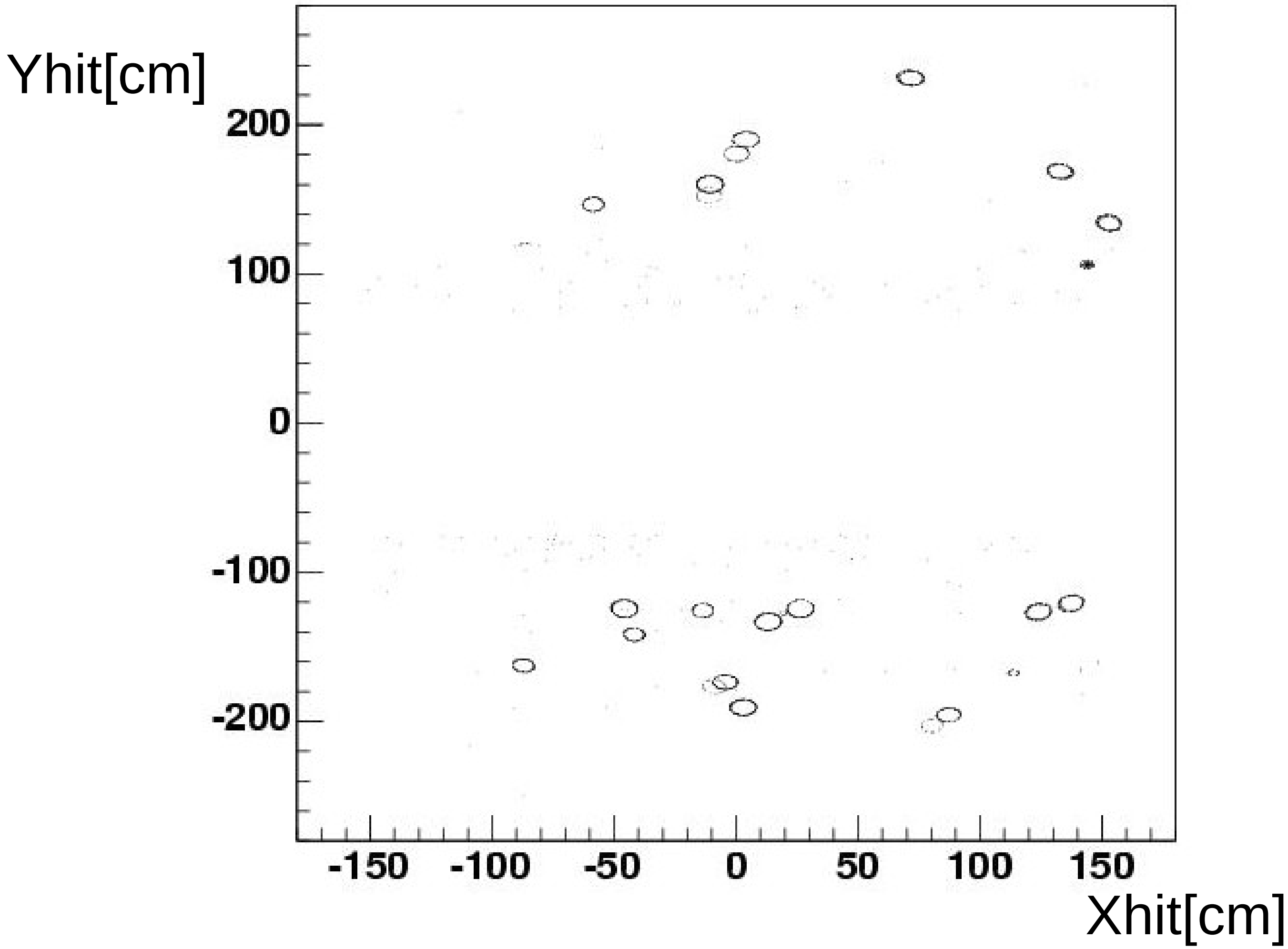
Accessing Parameters from Root file

```
CbmRunSim * fRun = new CbmRunSim;

CbmRuntimeDb* rtdb=fRun->GetRuntimeDb();

CbmParRootFileIo* input=new CbmParRootFileIo();
input->open("test.root");
rtdb->setFirstInput(input);

CbmParTest* par=(CbmParTest*)
                              (rtdb->getContainer("CbmParTest"));

rtdb->initContainers(fRunId);
```

CBM Detector Geometry



Example: Rich Detector



Analysis Macro

```
{
   gROOT->LoadMacro("$VMCWORKDIR/gconfig/basiclibs.C");
   basiclibs();
   gSystem->Load("libCbm");
   gSystem->Load("libITrack");
   CbmRunAna *fRun= new CbmRunAna();

   fRun->SetInputFile("/d/STS_AuAu25Gev_Urqmd.root");
   fRun->SetOutputFile("trackOutput.root");
```

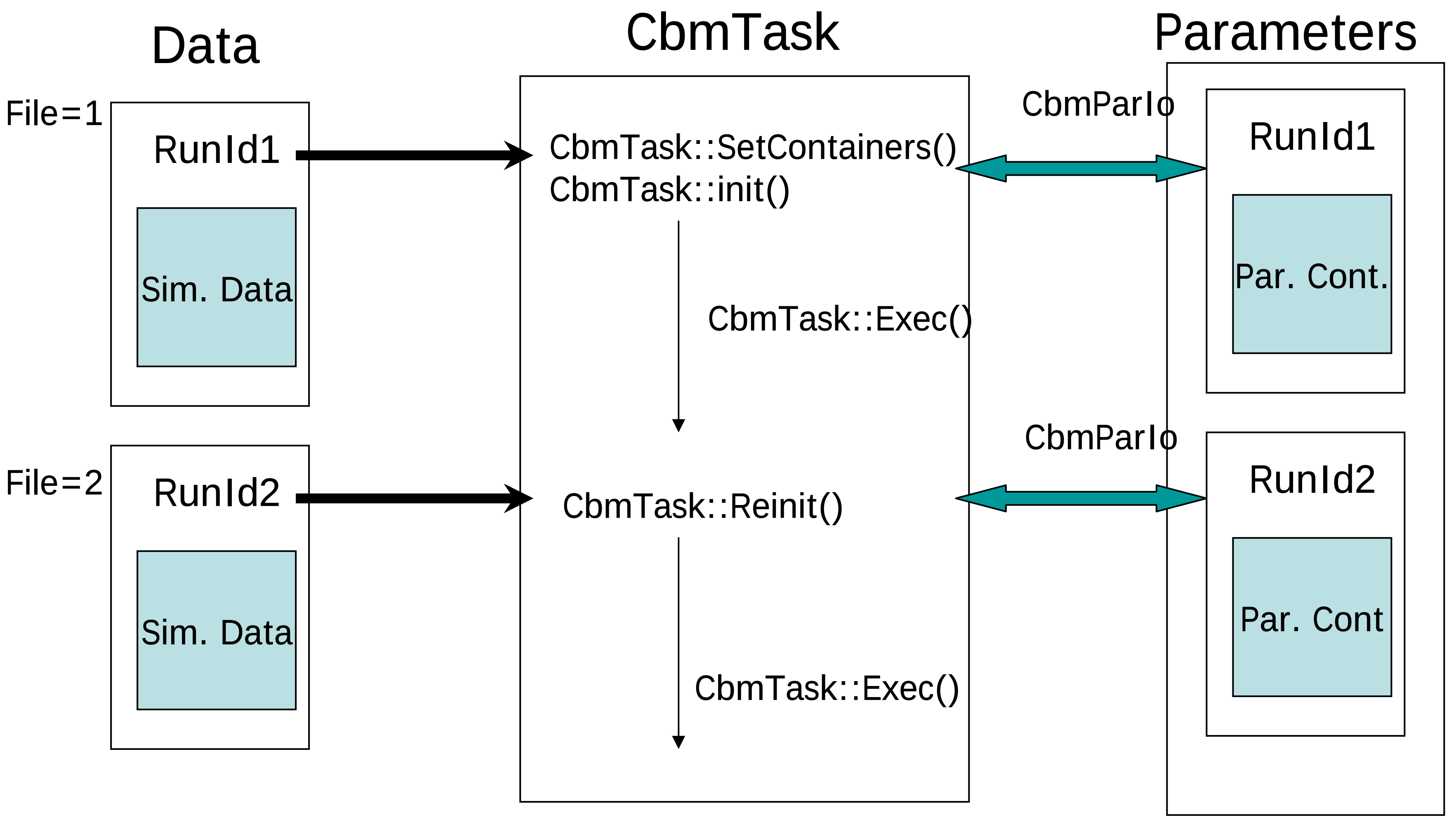
Analysis Macro: Parameters

```
// Init Simulation Parameters from Root File
CbmRuntimeDb* rtdb=fRun->GetRuntimeDb();
CbmParRootFileIo* input=new CbmParRootFileIo();

input->open("parfiles/testparams.root");

// Init Digitization Parameters from Ascii File
CbmParAsciiFileIo* input2 = new CbmParAsciiFileIo();

input2->open("sts_digi.par");

rtdb->setFirstInput(input);
rtdb->setSecondInput(input2);
```

Reading Parameters: Oracle

```
gSystem->Load ( "libOra" );
CbmRunAna * fRun = new CbmRunAna();
CbmRuntimeDb* rtdb=fRun->GetRuntimeDb();
CbmParOraIo* ora=new CbmParOraIo();
ora->open();
rtdb->setFirstInput(ora);
CbmGenericParOraIo* genio=
                    (CbmGenericParOraIo*)(ora->getDetParIo("CbmGenericParIo"));
CbmParTest* par=(CbmParTest*)(rtdb->getContainer("CbmParTest"));

genio->readFromLoadingTable(par,RunId);
par->print();
par->printParams();
```

Initialisation scheme (Analysis)



Track Visualization

- In the Simulation macro add:

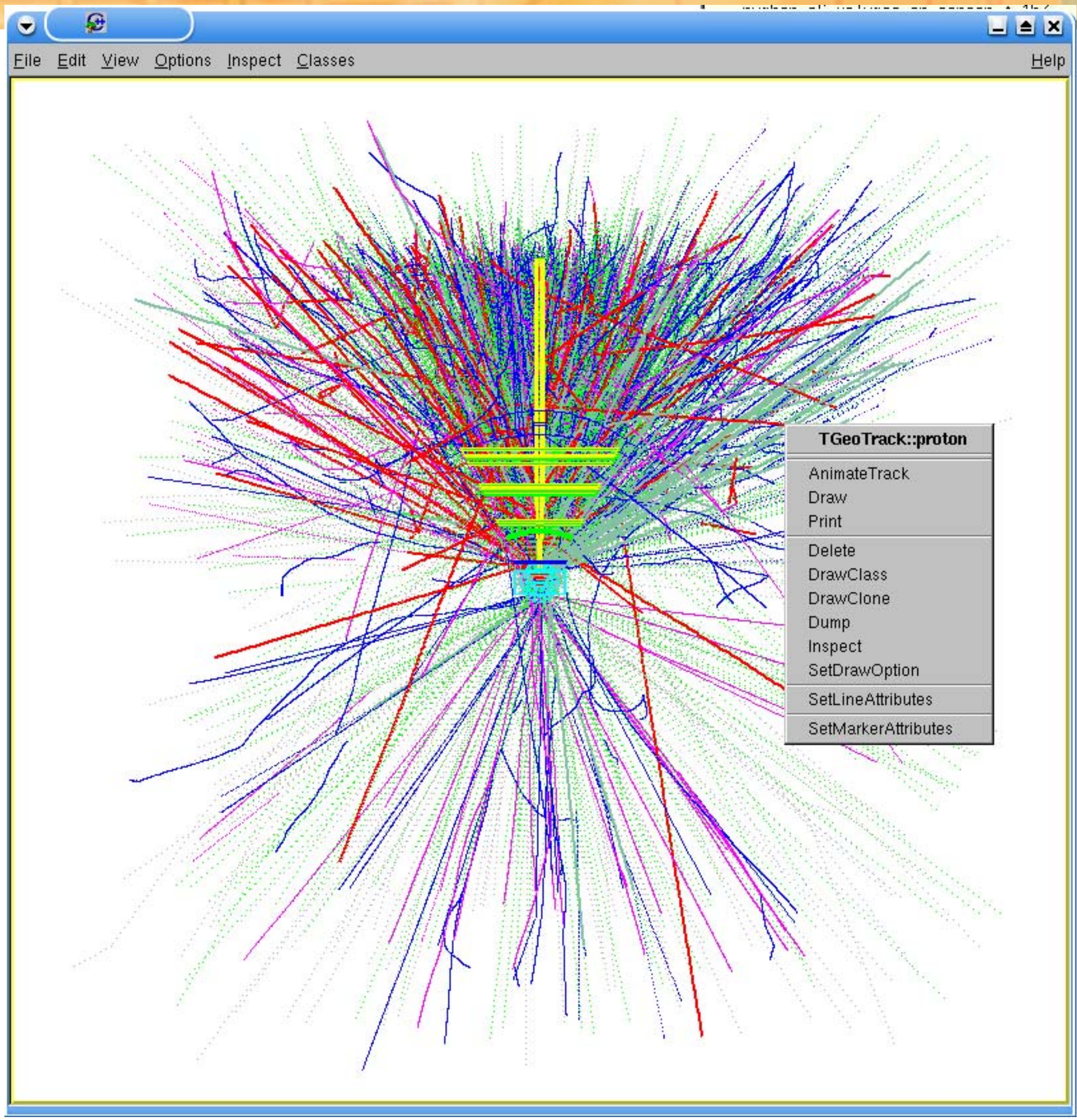
```
.........
fRun->SetStoreTraj(kTRUE);
fRun->Init();
```

```
// Set cuts for storing the trajectories
  CbmTrajFilter* trajFilter = CbmTrajFilter::Instance();
  trajFilter->SetStepSizeCut(1); // 1 cm
  trajFilter->SetEnergyCut(0., 1.04); // 0 < Etot < 1.04 GeV
  trajFilter->SetStorePrimaries(kFALSE);
  .........
```

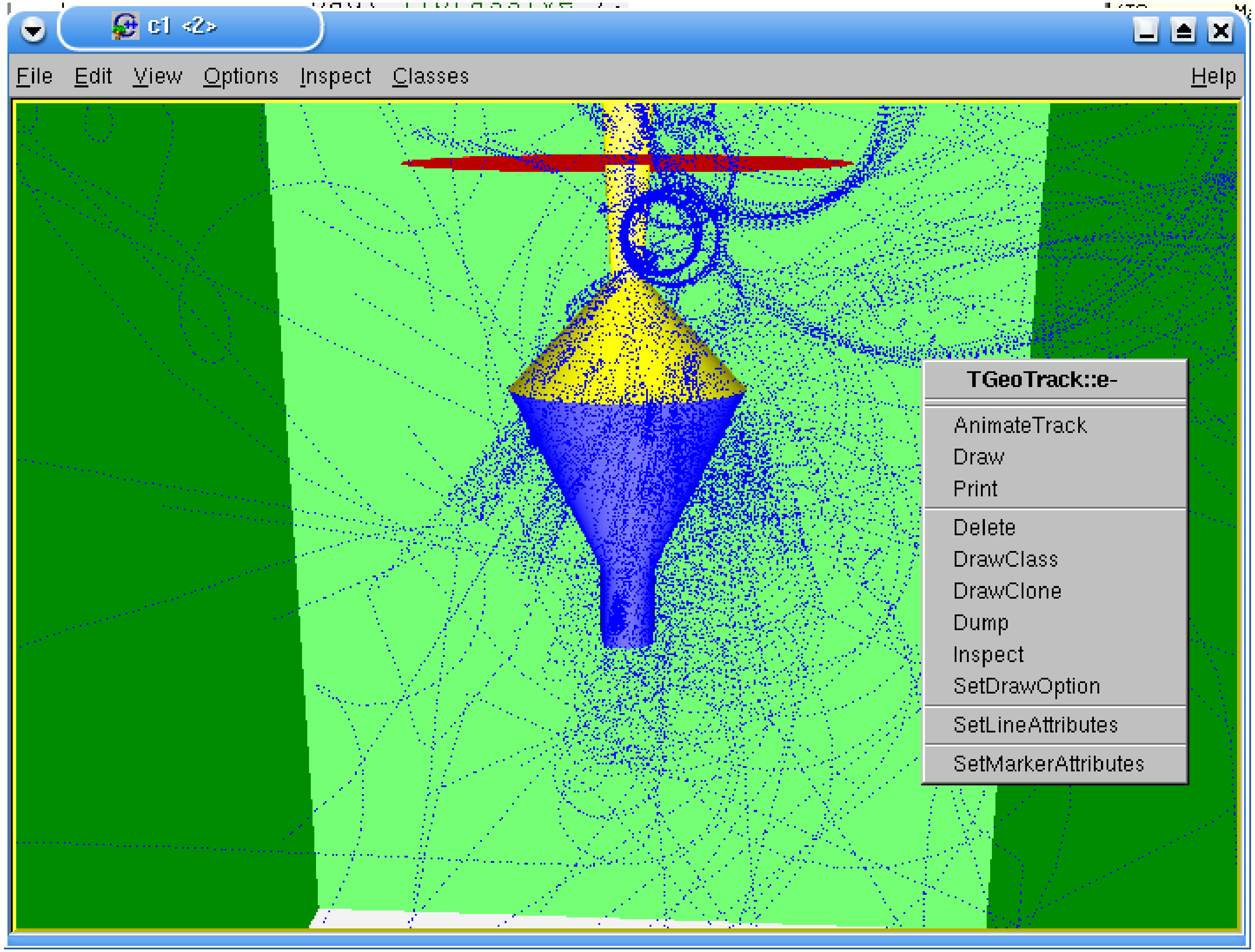
Example: Visualization macro

```
{
gROOT->LoadMacro("$VMCWORKDIR/gconfig/basiclibs.C");
basiclibs();
gSystem->Load("libCbm");
......

TFile* file = new TFile("test.root");
TGeoManager *geoMan = (TGeoManager*) file->Get("CBMGeom");
TCanvas* c1 = new TCanvas("c1", "", 100, 100, 800, 800);
c1->SetFillColor(10);
geoMan->DrawTracks("same/Nneutron");
geoMan->SetVisLevel(3);
geoMan->GetMasterVolume()->Draw("same");
}
```

Track Visualization



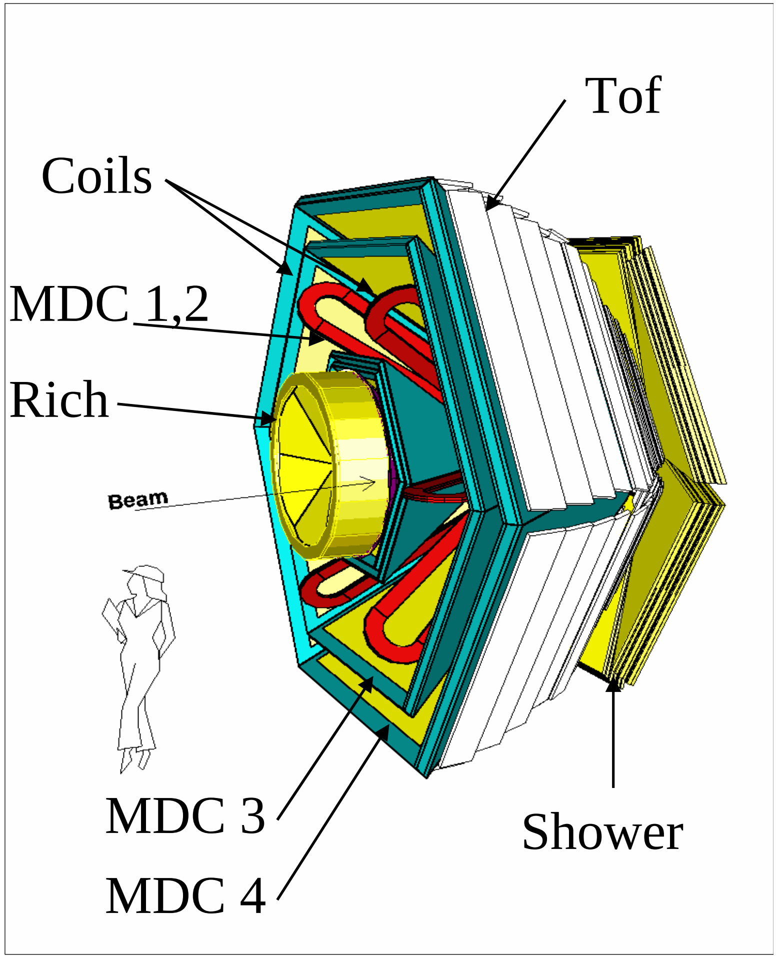
Track Visualization



Hades@GSI

- **Goal:** Study of in-medium modifications (ρ, ω, φ) properties
  - Produced in A+A, p+A, π+A collisions
  - Di-electrons are used as probes: $V \to e^+e^-$
- Hexagonal symmetry around the beam axis
- Geometrical acceptance of 40%
- Invariant mass resolution of 1%
- Operates at reaction rates up to 10^6 /s

  ⇒ 0.5 - 1 Tbyte/year
- ~ 70.000 readout channels



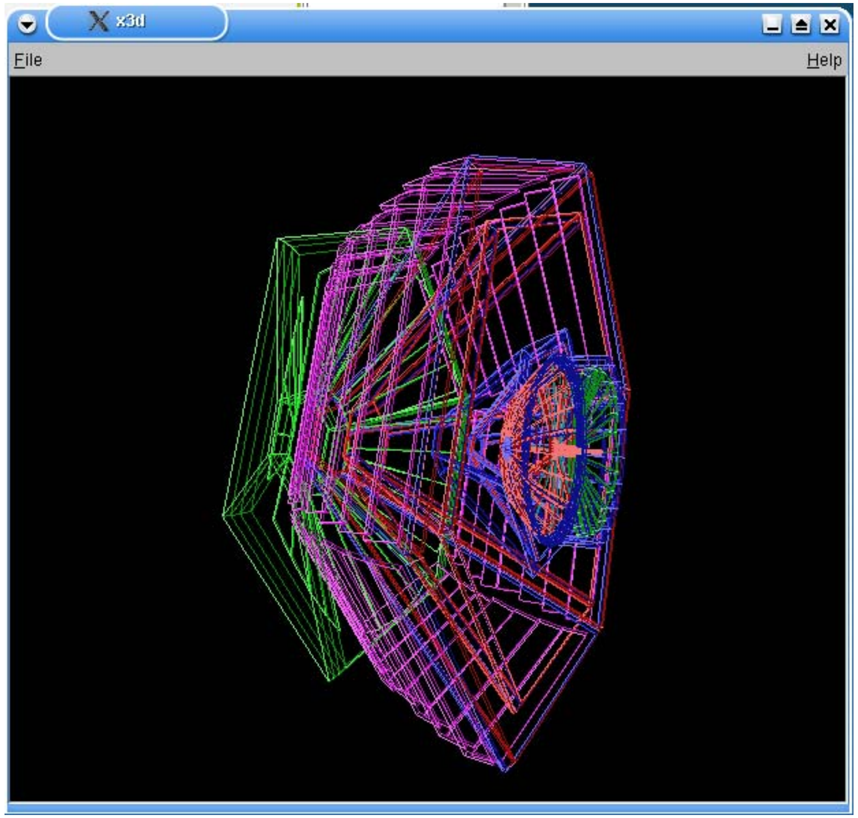
Hades Simulation using Cbmroot, why?

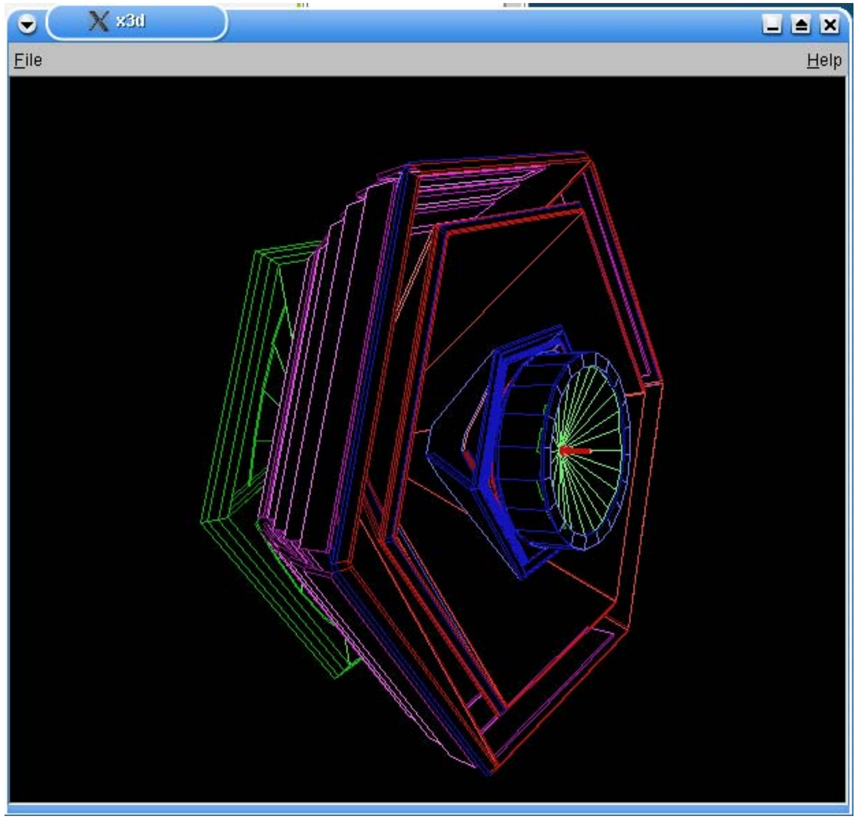
- **Need to simulate heavy system at High energy**
  - **Need external stack for Geant3: internal stack capacity reached**
  - **Check data with geant4**

- **Easy to use CBM framework services**

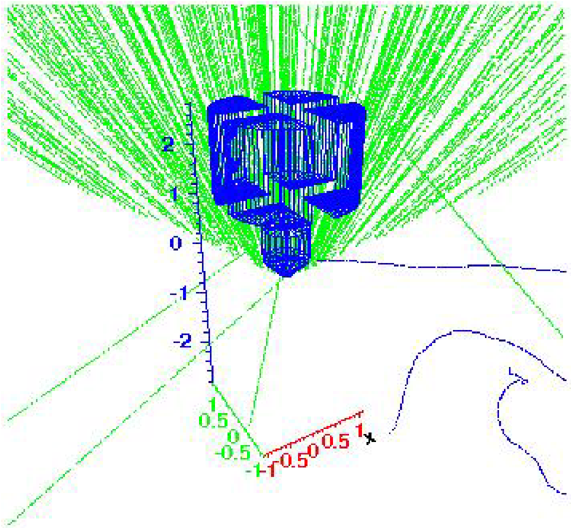
- **The Only efforts:**
  - Definition of Detector MC point container
  - Field map reader
  - Conversion from Lab. MC point definition to points defined in the local ref. Frame of the sensitive volume
  - Modification of some particles physical characteristics:
    - Use of TVirtualMC::Gspart()

Hades Simulation





CbmRoot@GF+E



Summary

- A VMC based framework for CBM has been implemented
  - First released in March 2004
  - Work on digitizers and full tracking is going on.

- Oct 04 release was used to produce data for the CBM technical report
  - Packages (ROOT 4.01/02 , GEANT3)

- June 05 release (initialization scheme added)
  - Packages (ROOT 5.02, GEANT3)

Availability

- Tested on
  - Red Hat 9.0 (gcc 3.2.2 and icc 8.1)
  - Suse 9.0 (gcc 3.3.1)
  - Debian (gcc 3.2.3)
  - Fedora Core 2 (gcc 3.3.3)
  - Fedora Core 4 (gcc 4.0)
  - AMD Opteron (64 bit architecture)
- Binaries are also available for these platforms

Other Applications

- Hades spectrometer has been fully integrated
  - Gives us the opportunity to tune Geant4 (cuts/physics list …)

  and compare with real Data !
  - Realistic test of the framework.

- Gesellschaft für Forschungs- und Entwicklungsservice mbH
  - Gamma detector development for cancer therapy project at the GSI

Ongoing work

- Complete the Hades simulation implementation using Cbmroot.
- Design and development of a Geometry builder for Simulation:
  - First proto type already available
  - Based on TGeo Classes
  - Undo/Redo Manager Implemented

(Mahmood Al-Turani and Wojciech Zietek)