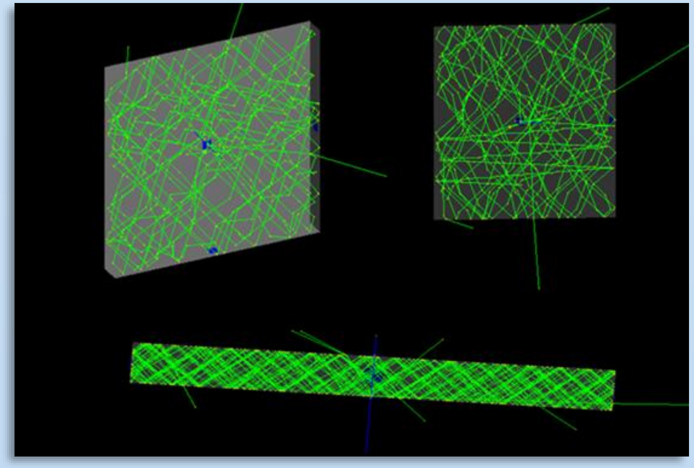
Simulation of optical photon propagation for generic scintillator-based detectors

Lecture 4

Physics List



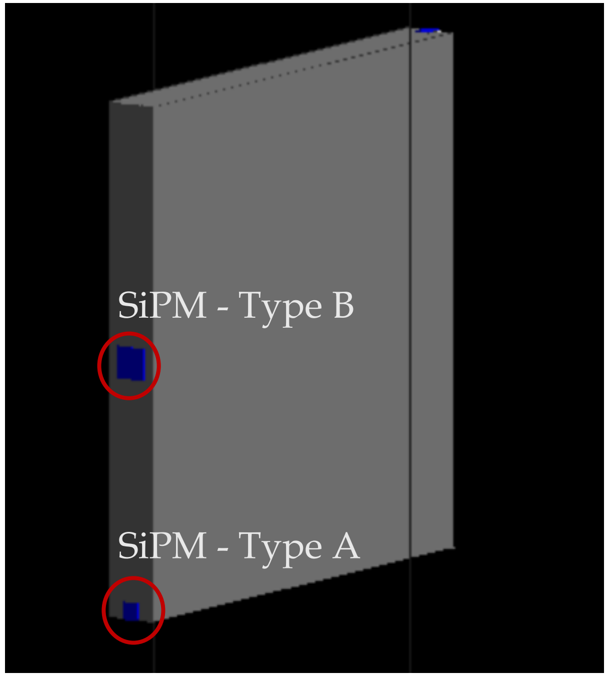
Davide Serini
davide.serini@ba.infn.it

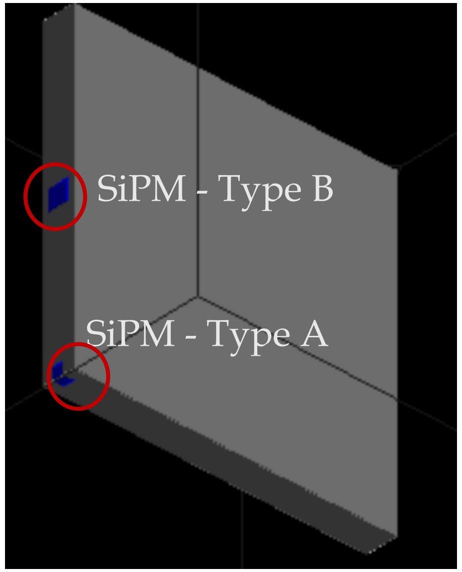
Exercise – Example0

- Starting from the Blank project create your plastic scintillator tile of $10x10x1cm^3$ equipped with:
  - 4 SiPMs (type A) of 3mm size on two edges of your tile
  - 2 SiPMs (type B) of 6mm size on the center of two small faces
- Wrap your tile with a 300 µm of Teflon
  - Suggestion: you need to «create the window» for your SiPMs (using multiple times G4Subtraction)

```
// World
G4double world_sizeX  = 20*cm;
G4double world_sizeY  = 20*cm;
G4double world_sizeZ  = 20*cm;

G4double ScintillatorX      = 10*cm;
G4double ScintillatorY      = 10*cm;
G4double ScintillatorZ      = 1*cm;
G4double SiPMThickness      = 0.4*mm;
G4double WrappingThickness = 300*um;
G4double SiPMSize           = 3*mm;
G4double SiPMSize2          = 6*mm;
```





Introduction

- This part of the course provides an overview of Geant4:
  - its capabilities
  - how they can be used in an experimental simulation application (we will focus on the optical simulations)
- Geant4 is a complex and powerful toolkit
  - Impossible to teach (and learn) everything in few days and also our own expertise is not infinite
- This course would provide you with a global vision of Geant4 and a method for "how to use it"
  - Finding **your way** in the complexity of Geant4 is not easy
  - Use the Geant4 User Documentation and the "**BookForApplicationDevelopers**"

Geant4 web site: http://cern.ch/geant4

General G4 introduction

- What is necessary to built a complete simulation:
  - Detector Construction:
    - *Build your geometry, define your materials and eventually assign optical bulk and surface properties*
  - Physics List:
    - *"Activate" the physical processes that can occur during your simulation*
  - Hit collection:
    - *Define your "hit container"; i.e. which are the information that you want store during your simulation*
  - Sensitive Detector:
    - *Define your definition of hit; i.e. when you want to store the information*
  - Action classes:
    - *G4UserRunAction:*
      - Define what happen at the begin and at the end of the **complete simulation**
        - Create an output file with its structure (Beginning)
        - Close the output file (end)
    - *G4UserEventAction:*
      - Define what happen at the begin and at the end of **each event**
        - Retrieve the information of each hit
        - Fill the output

General G4 introduction

- Composing a G4 application:
  - General Classes:
    - *Initialization classes*
      - G4VUserDetectorConstrucion
      - G4VUserPhysicsList
    - *Action classes*
      - G4VUserPrimaryGenerationAction
        - G4UserRunAction
        - G4UserTrackingAction
        - G4UserStackingAction
        - G4UserSteppingAction

Main.cc()

RunManager

INITIALIZATION
MyDetector.cc
MyPhisicsList.cc

ACTION
MyPrimaryGen.cc

Defines:
geometry
and magnetic fields

Defines:
Particles and physical processes

Defines:
Particle sources and user actions

Classes Hierarchy

Main() → **RunManager**

- *G4VUserDetectorConstruction()* → Defines Geometry and magnetic fields
- *G4VUserPhysicsList()* → Defines Particles and physical processes
- *G4VUserActionInitialization()* → Defines Particles sources and user actions
  - *G4VUserPrimaryGenerationAction()*
  - *G4UserRunAction()*
  - *G4UserEventAction()*
    - *G4VSensitiveDetector()*
      - *G4VHit()*

How to define a «Physics List»

- GEANT4 doesn't have any default particles or processes, but there are a set of "ready-for-use" physics lists released with G4.

  - PhysicsList() class inherited from G4VUserPhysicsList()
    - *ConstructParticle()*
      - Defines all necessary particles.
      - **G4ParticleDefinition**: name, mass, spin, PDG number, etc.
      - **G4DynamicParticle**: energy, momentum, polarization, etc.
    - *ConstructProcess()*
      - Defines all necessary processes and assign them to proper particles.
    - *SetCuts()*
      - Defines particles production threshold (in terms of range).

Particle	Process	G4Process
Photons	Gamma Conversion in $e^{\pm}$	G4GammaConversion
	Compton scattering	G4ComptonScattering
	Photoelectric effect	G4PhotoElectricEffect
	Rayleigh scattering	G4RayleighScattering
$e^{\pm}$	Ionisation	G4eIonisation
	Bremsstrahlung	G4eBremsstrahlung
	Multiple scattering	G4eMultipleScattering
e^{+}	Annihilation	G4eplusAnnihilation

A complete list and explanation of all the physical models implemented in Geant4 is available in the Application Developers Guide, Chapter 5

Your physics List (G4VUserPhysicsList)

- **ConstructParticle()**
  - Choose the particles you need in your simulation, define all of them here
- **ConstructProcess()**
  - For each particle, assign all the physics processes relevant to your simulation
- **SetCuts()**
  - Set the range cuts for the secondary production for processes

```
class MyPhysicsList : public G4VModularPhysicsList {
public:
    MyPhysicsList();            // define physics constructors
    void ConstructParticle(); // optional
    void ConstructProcess();  // optional
    void SetCuts();           // optional
}
```

Make your physics list

- ***Reference to physics lists***: Take one/more pre-defined physics list *(EASIEST used case)*
- ***Physics constructors***: Combine pre-defined sets of particles and processes *(INTERMEDIATE & for specific applications)*
- ***Manually***: Specify all particles & process that may occur in the simulation *(ADVANCED & for specific application)*

Particle categories

- Elementary particles which should be tracked in GEANT4 volumes
  - All particles that can fly a finite length and interact with materials in detectors:
    1. *stable particles*
       - Stable means that the particle can not decay, or has a very small possibility to decay in detectors, e.g., gamma, electron, proton, and neutron.
    2. *long life (>10^{-14} sec) particles*
       - Particles which may travel a finite length, e.g., muon, charged pions.
    3. *short life particles that decay immediately in GEANT4 (e.g. π^0)*
    4. *optical photon*
       - Gamma and optical photon are distinguished in the simulation view, though both are the same particle (photons with different energies). ***Optical photon is used for Cerenkov light and scintillation light.***
    5. *geantino*
       - Geantino is a virtual particles for simulation which do not interact with materials and undertake transportation processes only.

- Nuclei
  - Any kinds of nucleus can be used in GEANT4, such as alpha (He-4), uranium-238 and excited states of carbon-14.

- Short-lived particles
  - Particles with very short life time decay immediately and are never tracked in the detector geometry.
    - *These particles are usually used only inside physics processes to implement some models of interactions.*
      - quarks…

Particle list

- G4ParticleDefinition:
  - Define and see properties of particles:
    - *PDG encoding*
    - *mass and width*
    - *electric charge*
    - *spin, isospin and parity*
    - *magnetic moment*
    - *quark contents*
    - *life time and decay modes*
- Particle list:
  - This list is generated automatically by using GEANT4 functionality, so listed values are same as those in your GEANT4 application.
  - Categories
    - *gluon / quarks*
    - *leptons*
    - *mesons*
    - *baryons*
    - *ions*
    - *others*

`G4ParticleDefinition()` get methods

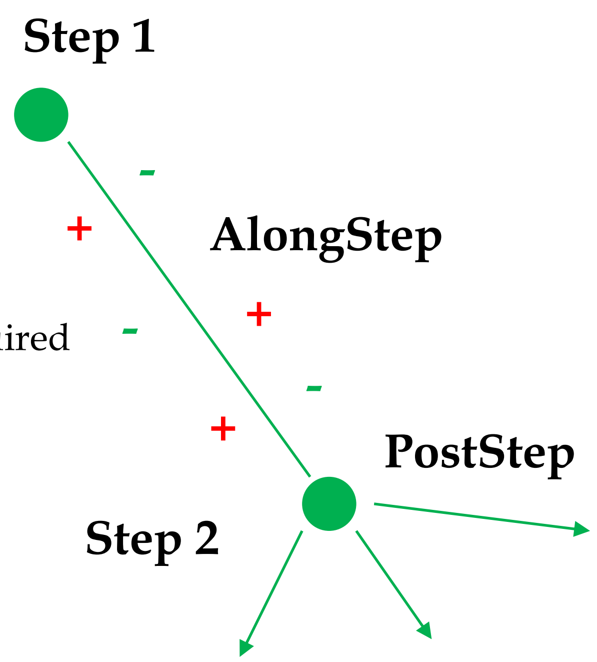
`G4String GetParticleName()`	particle name
`G4double GetPDGMass()`	mass
`G4double GetPDGWidth()`	decay width
`G4double GetPDGCharge()`	electric charge
`G4double GetPDGSpin()`	spin
`G4double GetPDGMagneticMoment()`	magnetic moment (0: not defined or no magnetic moment)
`G4int GetPDGiParity()`	parity (0:not defined)
`G4int GetPDGiConjugation()`	charge conjugation (0:not defined)
`G4double GetPDGIsospin()`	iso-spin
`G4double GetPDGIsospin3()`	3^{rd}-component of iso-spin
`G4int GetPDGiGParity()`	G-parity (0:not defined)
`G4String GetParticleType()`	particle type
`G4String GetParticleSubType()`	particle sub-type
`G4int GetLeptonNumber()`	lepton number
`G4int GetBaryonNumber()`	baryon number
`G4int GetPDGEncoding()`	particle encoding number by PDG
`G4int GetAntiPDGEncoding()`	encoding for anti-particle of this particle

`G4DynamicParticle()` get methods

`G4double theDynamicalMass`	dynamical mass
`G4ThreeVector theMomentumDirection`	normalized momentum vector
`G4ParticleDefinition* theParticleDefinition`	definition of particle
`G4double theDynamicalSpin`	dynamical spin (i.e. total angular momentum as a ion/atom)
`G4ThreeVector thePolarization`	polarization vector
`G4double theMagneticMoment`	dynamical magnetic moment (i.e. total magnetic moment as a ion/atom)
`G4double theKineticEnergy`	kinetic energy
`G4double theProperTime`	proper time
`G4double theDynamicalCharge`	dynamical electric charge (i.e. total electric charge as a ion/atom)
`G4ElectronOccupancy* theElectronOccupancy`	electron orbits for ions

Processes in Geant4

- All GEANT4 processes are treated generically.
  - A process does two things:
    - *decides when and where an interaction will occur*
      - method: GetPhysicalInteractionLength()
        - this requires a cross section or a decay lifetime
        - for the transportation process, the distance to the nearest object along the track is required
    - *generates the final state of the interaction (changes momentum, generates secondaries, etc.)*
      - method: DoIt()
        - this requires a model of the physics



- Physics processes describe HOW particles interact with material
  - G4 apply defined physics on the defined particles using three kinds of actions:
    - *AtRest actions:*
      - Decay, e+ annihilations,…
    - *AlongStep actions:*
      - To describe continuous actions, occurring along the path of the particle, like ionisation.
    - *PostStep actions:*
      - For describing point-like actions, like decay in flight, hadronic interactions,…

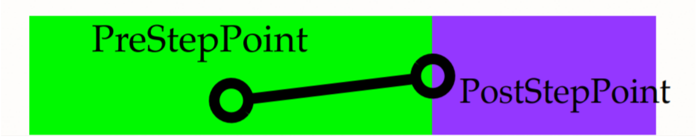
G4Track and G4Step

G4Track concept

- G4Track keeps 'current' information of the particle
  - *energy, momentum, position ,time ('dynamic' information)*
  - *mass, charge ('static' information)*
- GEANT4 tracks particles step by step inside the defined geometry.
  - *Only processes can change information of G4Track and add secondary tracks*
  - *G4Track is updated after each invocation of a PostStepDoIt.*

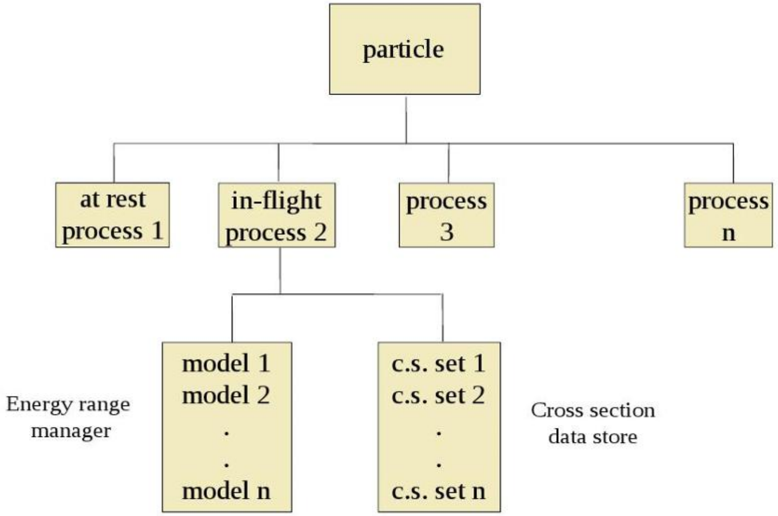
G4Step concept

- A G4Step can be seen as a "segment" delimited by two points
  - *It contains "delta" information (energy loss along the step, time-offlight, etc)*



G4VProcess concepts

- There are three kinds of basic processes:
  - well-located in space ➔ PostStep
  - distributed in space ➔AlongStep
  - well-located in time ➔AtRest



- A generic process
  - G4VProcess is a base class of all processes and it has 3 kinds of DoIt methods and a GetPhysicalInteraction method in order to describe generically the interations.
  - "Shortcut" processes are defined which invoke only one "physics" (DoIt action)
    - *Discrete process (has only PostStep physics)*
    - *Continuous process (has only AlongStep physics)*
    - *AtRest process (has only AtRest physics)*
- In the generic case six methods must be implemented, one GetPhysicalInteractionLength() and one DoIt() for each type of simple process
  - After each DoIt, ParticleChange updates PostStepPoint based on proposed changes.
    - *eg: decay = AtRest + PostStep*

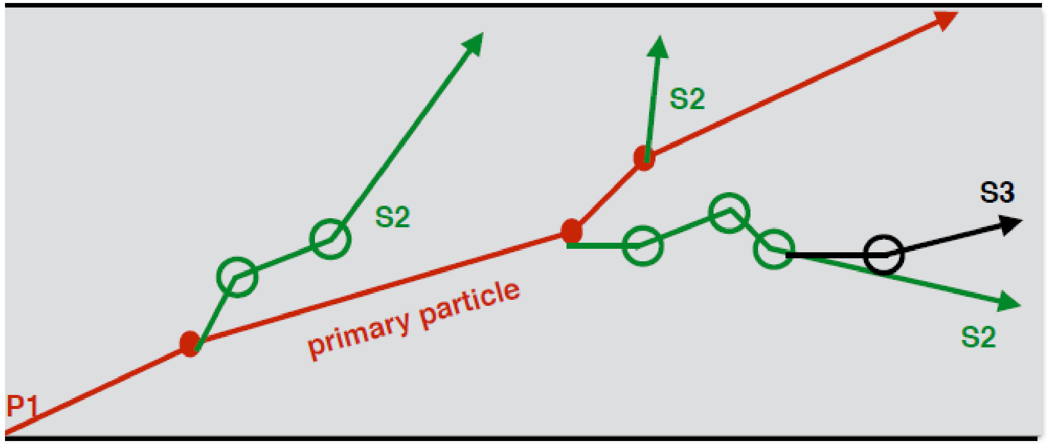
Handling multiple processes

1. A particle is shot and "transported".
2. All processes associated to the particle propose a geometrical step length (depends on process cross-section).
3. The process proposing the shortest step "wins" and the particle is moved to destination.
4. All processes along the step are executed (e.g. ionization)
5. Post step phase of the process that limited the step is executed. New tracks are "pushed" to the stack
6. If E_{kin}=0 all at rest processes are executed:

   if particle is stable the track is killed

   … else…

   New step starts and sequences repeats,…



Retrieve your information (Stepping Action example)

- Track and Step information may be accessed by invoking various Get methods provided in the G4Track and G4Step class

Example: Processing a single step (G4Step *aStep) and verify if an optical photon is absorbed (detected)

```
G4Track *aTrack = aStep->GetTrack();
const G4ParticleDefinition* aDef = aTrack->GetParticleDefinition();
G4String particleName = aDef->GetParticleName();
if(particleName=="opticalphoton"){
  G4String procName= aStep->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName();
  if(G4StrUtil::contains(procName,"Absorption")){
    G4String absVolume = aStep->GetPostStepPoint()->GetPhysicalVolume()->GetName();
    G4cout<<"[INFO]: A Photon was absorbed in "<<absVolume<<G4endl;
  }
 }
```

Cuts in Geant4

- In Geant4 there are no tracking cuts
  - particles are tracked down to a zero range/kinetic energy
  - **Only production cuts exist** i.e. *cuts allowing a particle to be born or not*
- Why are production cuts needed ?
  - Some processes could involve divergences
  - this leads to an infinity [huge number] of smaller and smaller energy
    - *e.g. photons/electrons for Bremsstrahlung, δ-ray production…*
  - production cuts limit this production to particles above the threshold
  - the remaining, "divergent part" is treated as a continuous effect (i.e. AlongStep action)
- A range cut allows to easily define such visibility:

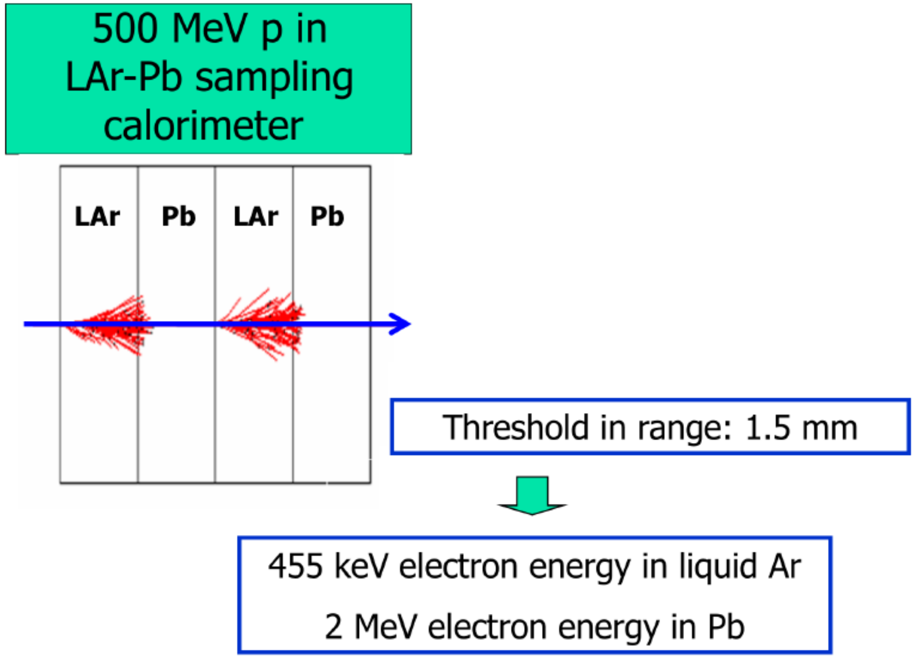
"I want to produce particles able to travel at least 1 mm"

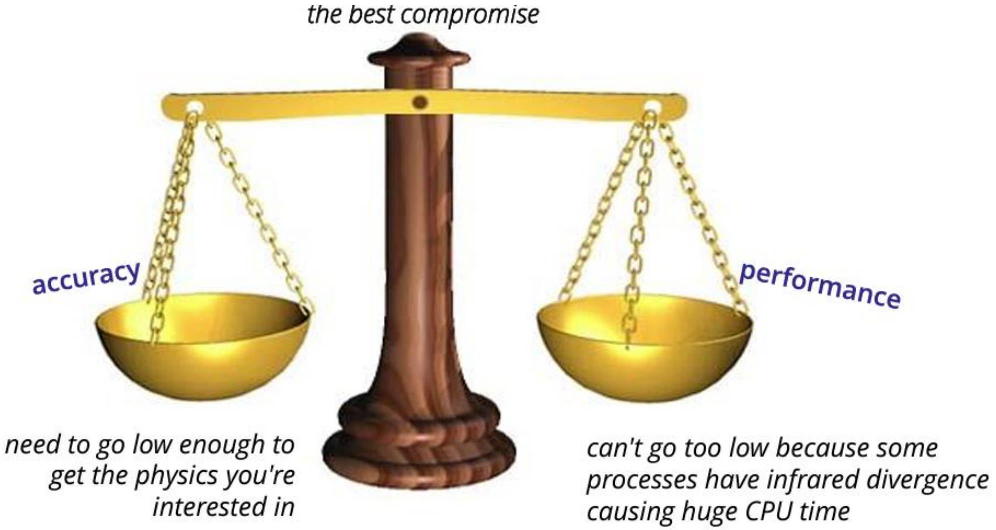
No secondary particles will be produced in the detector inside this distance

- The same energy cut leads to very different ranges
  - for the same particle type, depending on the material
  - for the same material, depending on particle type
- The user specifies a range cut in the PhysicsList
  - this range cut is converted into energy cuts
  - each particle (G4ParticleWithCut) converts the range cut into an energy cut, for each material
- Processes then compute the cross-sections ***based on the energy cut***

Range vs. energy production cuts

- You can set a "range" production threshold
  - This threshold is a distance, **not an energy**.
  - Particles unable to travel at least the range cut are not produced
  - Energy is conserved
  - Production threshold is internally concerted to the energy thresholds depending on particle type and material:
  - Effective energy threshold is different in each material
- You can also create sub-regions in your detector with their own cuts
  - In the geometry an instance of G4Region must be created which corresponds to the volume where the cuts are to be changed





Pre-defined Physics Lists

- Pre-packaged physics lists:
  - Geant4-toolkit provides pre-packaged physics lists *according to some reference use cases*
    - *These are "ready-to-use", complete physics lists provided by the toolkit and constructed by the* ***expert developers***
    - *each pre-packaged physics list includes different combinations of EM and hadronic physics*
  - The list of these pre-packaged physics lists can be found in

    `$G4DIR/src/geant4/source/physics_lists/lists/include`
- A guide is available at:

  https://geant4-userdoc.web.cern.ch/UsersGuides/PhysicsListGuide/html/index.html

```
FTFP_BERT.hh       FTF_BIC.hh                    G4PhysListStamper.hh          QBBC_ABLA.hh       QGSP_FTFP_BERT.hh
FTFP_BERT_ATL.hh   G4GenericPhysicsList.hh       G4RegisterPhysLists.icc       QGSP_BERT.hh       QGSP_INCLXX.hh
FTFP_BERT_HP.hh    G4GenericPhysicsList.icc      INCLXXPhysicsListHelper.hh    QGSP_BERT_HP.hh    QGSP_INCLXX_HP.hh
FTFP_BERT_TRV.hh   G4PhysListFactory.hh          INCLXXPhysicsListHelper.icc   QGSP_BIC.hh        QGS_BIC.hh
FTFP_INCLXX.hh     G4PhysListFactoryAlt.hh       LBE.hh                        QGSP_BIC_AllHP.hh  Shielding.hh
FTFP_INCLXX_HP.hh  G4PhysListFactoryMessenger.hh NuBeam.hh                     QGSP_BIC_HP.hh     ShieldingLEND.hh
FTFQGSP_BERT.hh    G4PhysListRegistry.hh         QBBC.hh                       QGSP_BIC_HPT.hh
```

Docs » Reference Physics Lists » FTFP_BERT

FTFP_BERT

It is the current GEANT4 default [eal16].

Physics Lists

- “QGS” Quark gluon string model (>~20 GeV)
- “FTF” Fritiof Model (>~ 10 GeV)
- “LHEP” Low and High energy parameterization model
- “BIC” Binary Cascade Model (<~10 GeV)
- “BERT” Bertini Cascade Model (<~10 GeV)
- “HP” High Precision Neutron Model (<~20 MeV)
- “PRECO” Pre compound Model (<~150 MeV)
- “EMV(X)” Variation od Standard EM package

Production physics list

- "Production physics lists":
  - these physics lists are used by large user groups like ATLAS, CMS, etc.
  - they are well-maintained and tested physics lists they are changed, updated less frequently
  - very stable physics lists they are extensively validated by the developers and the user communities FTFP_BERT, QGSP_BERT, QGSP_FTFP_BERT_EMV, FTFP_BERT_HP, QGSP_BIC_EMY, QGSP_BIC_HP, QBBC, Shielding
- FTFP_BERT: Recommended by Geant4 developers for HEP applications
  - Add your "standard" EM physics constructors:

    G4EmStandardPhysics - standard, for "higher energy physics"

    G4EmStandardPhysics_option1 - for HEP, fast but not precise settings

    G4EmStandardPhysics_option2 - for HEP, experimental

    G4EmStandardPhysics_option3 - for medical and space science applications

    G4EmStandardPhysics_option4 **- most accurate EM models and settings**
  - Pre-packaged physics lists are provided as a "best guess" of the physics needed in some given use cases
    - ***when a user decide to use them, the user is responsible for "validating" the physics for that given application and adding (or removing) the appropriate physics***

Reference physics lists: myPhysicsList Class

- **Pre-defined physics lists**
  - Already containing a complete set of particle & processes (that work together)
  - Targered at specific area of interest
  - Constructed as modular physics lists built on top of physics constructor
  - Customizable (adding/calling appropiate methods **before initialization**)

```
PhysicsList::PhysicsList()
  :G4VUserPhysicsList()
{
  //Same Physics of FTFP_BERT
  //EM Physics
  emPhysicsList = new G4EmStandardPhysics();
  //Synchroton Radiation & GN Physics
  emExtraPhysics = new G4EmExtraPhysics();
  //Decays
  DecayPhysics = new G4DecayPhysics();
  // Hadron Elastic scattering
  HadronElastic = new G4HadronElasticPhysics();
  // Hadron Physics
  HadronPhysics = new G4HadronPhysicsFTFP_BERT();
  // Stoping Physics
  StoppingPhysics = new G4StoppingPhysics();
  // Ion Physics
  IonPysics = new G4IonPhysics();
  //Neutron Pysics
  NeutronPysics = new G4NeutronTrackingCut();

}
```

See myPhysicsList example to create your complete PhysicsList class and add it to your simulation

[myPhysicsList.hh]()
[myPhysicsList.cc]()

Exercise – Example1A

- Add your physics list to your simulation (Example0) and activate the optical processes
  - Try to run a simulation

Reference physics lists

-- In your main() --

```
G4RunManager* runManager = new G4RunManager;
...

// Physics list
G4VModularPhysicsList* physicsList = new FTFP_BERT;
physicsList->ReplacePhysics(new G4EmStandardPhysics_option4());
auto opticalPhysics = new G4OpticalPhysics();
physicsList->RegisterPhysics(opticalPhysics);
runManager->SetUserInitialization(physicsList);
```

-- In your macro file --

```
/process/optical/processActivation Cerenkov     true
/process/optical/processActivation OpAbsorption  true
/process/optical/processActivation OpBoundary    true
/process/optical/processActivation Scintillation true

/process/optical/scintillation/setByParticleType false
/process/optical/scintillation/setFiniteRiseTime false
/process/optical/scintillation/setTrackSecondariesFirst false

/process/optical/cerenkov/setMaxPhotons 50

/run/initialize
```

Reference

`$G4INSTALL_DIR/share/Geant4/examples/extended/optical/OpNovice2/`

Remember to initialize the kernel after adding all the optical properties