Bruno: present status and future perspectives

Andrea Di Simone
INFN Roma2

Outline

- Short introduction to G4
- SuperB G4 Simulation:
  - Present status
  - Main missing functionalities
    - MC truth machinery
    - Physics lists
    - Physics regions
    - User interface
  - Proposals

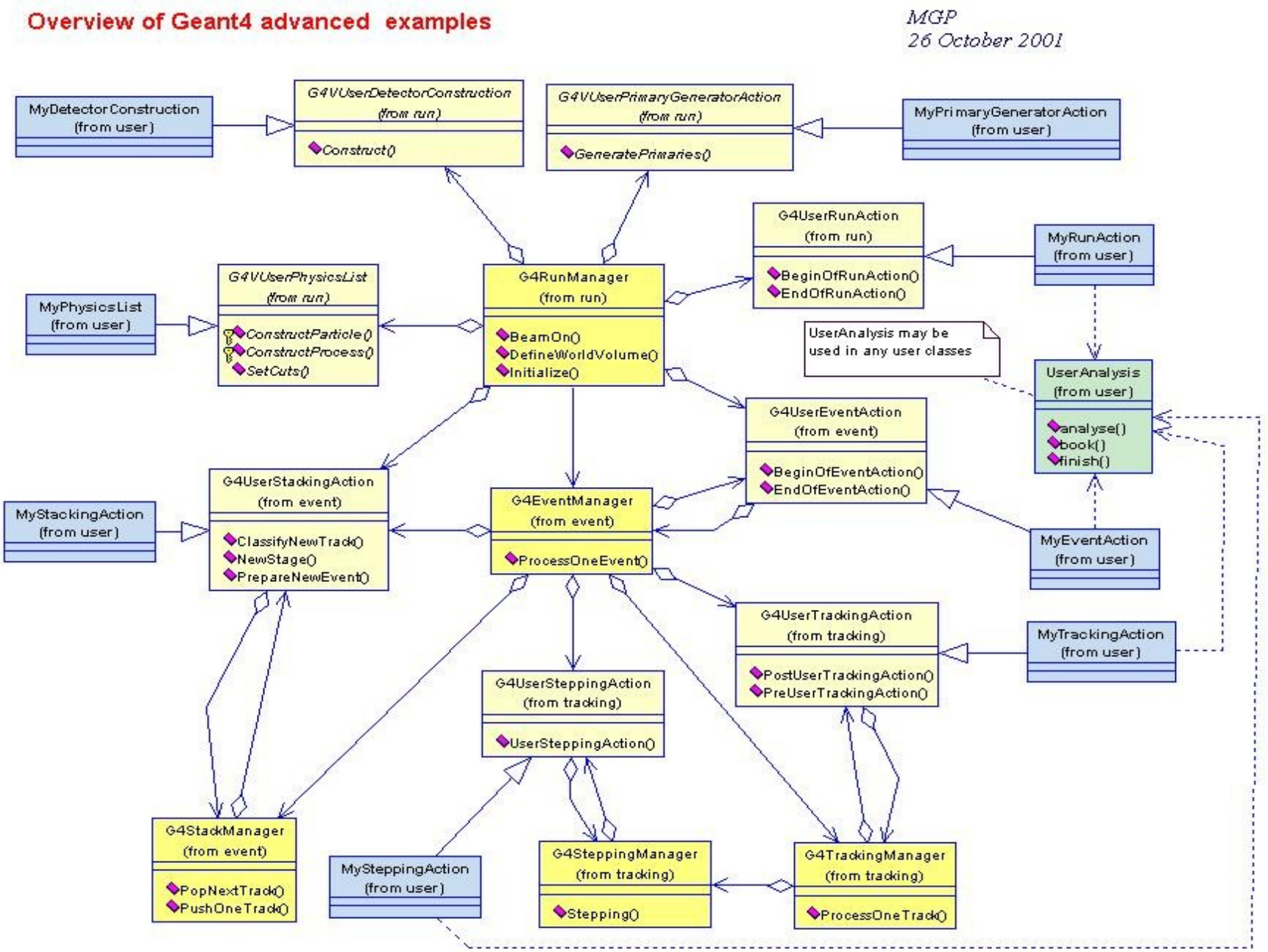
Why not using G3?

- Geant3 was a detector simulation program developed for the LEP era
  - Fortran, ZEBRA
  - Electromagnetic physics directly from EGS
  - Hadronic physics added as an afterthought (and always by interfacing with external packages)
  - Powerful but simplistic geometry model
  - Physics processes very often limited to LEP energy range (100 GeV)
  - (Painfully) debugged with the help and the collaboration of 100s of physicist from all over the world
- LHC detectors need powerful simulation tools for the next 20 years
  - reliability, extensibility, maintainability, openness
  - good physics, with the possibility of extending

What's new

- **Geant3**
  - The geometry model is limited to a pre-defined set of basic shapes. Adding a new shape requires changing the code of several (~20) routines. Interface to CAD systems in not possible
  - Tracking is the result of several iterations (by different people), resulting in difficult to maintain (and to understand) code.
  - EM physics is built in, but several processes are missing and their implementation would be very hard
  - Hadronic physics is implemented via external (and obsolete) packages. Modifications require the author's intervention
- **Geant4**
  - The geometry has been based since the beginning on a CAD-oriented model. The introduction of a new shape does not influence tracking
  - Tracking has been made independent from geometrical navigation, tracking in electromagnetic fields (or any field) has been improved
  - EM and hadronic physics implemented in terms of processes. A process can be easily added or modified by the user and assigned to the relevant particles with no change in the tracking. The cut philosophy has been changed so as to make result less dependent on the cuts used for the simulation. Framework for physics parameterisation in place

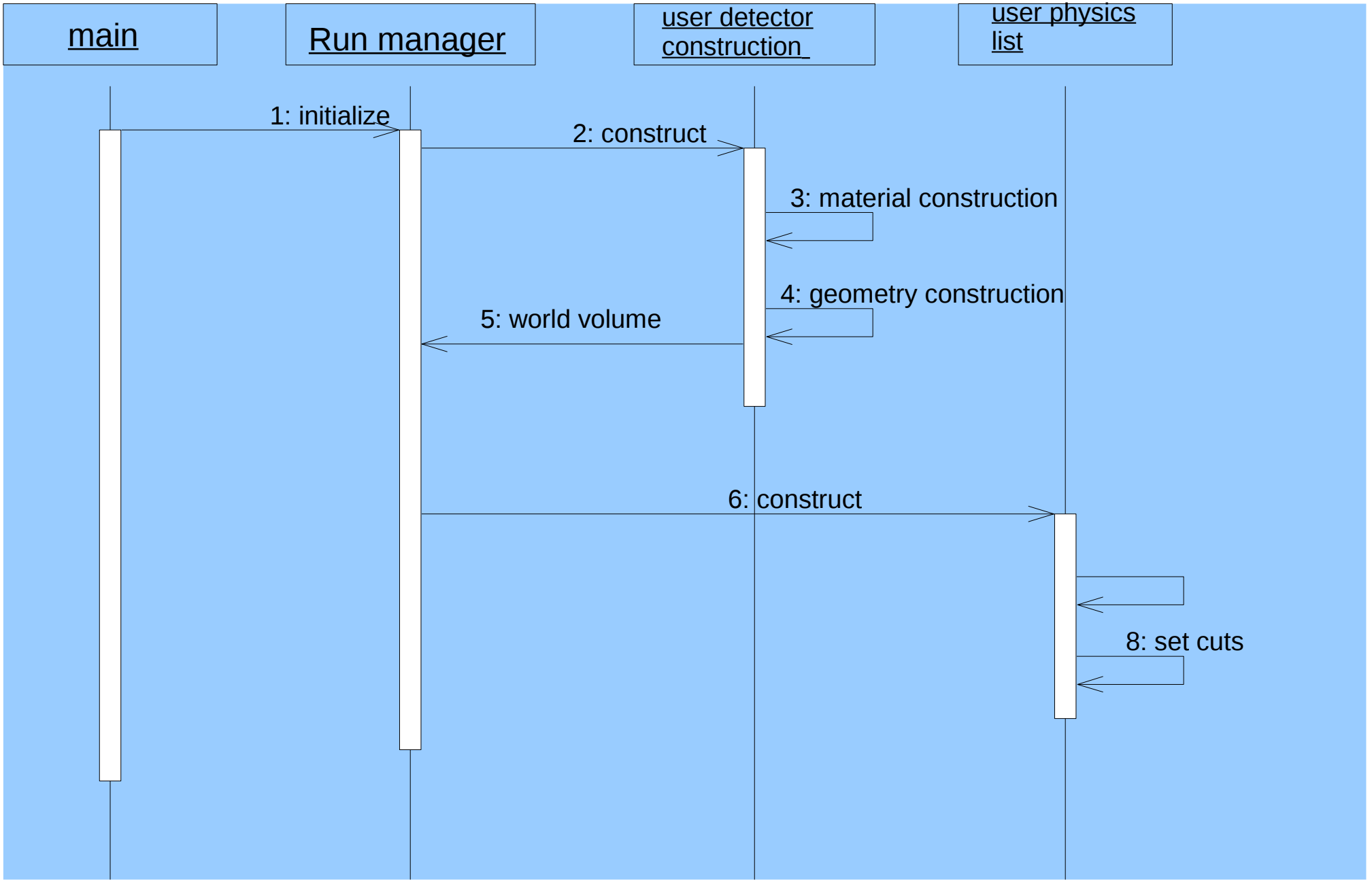
Architectural overview



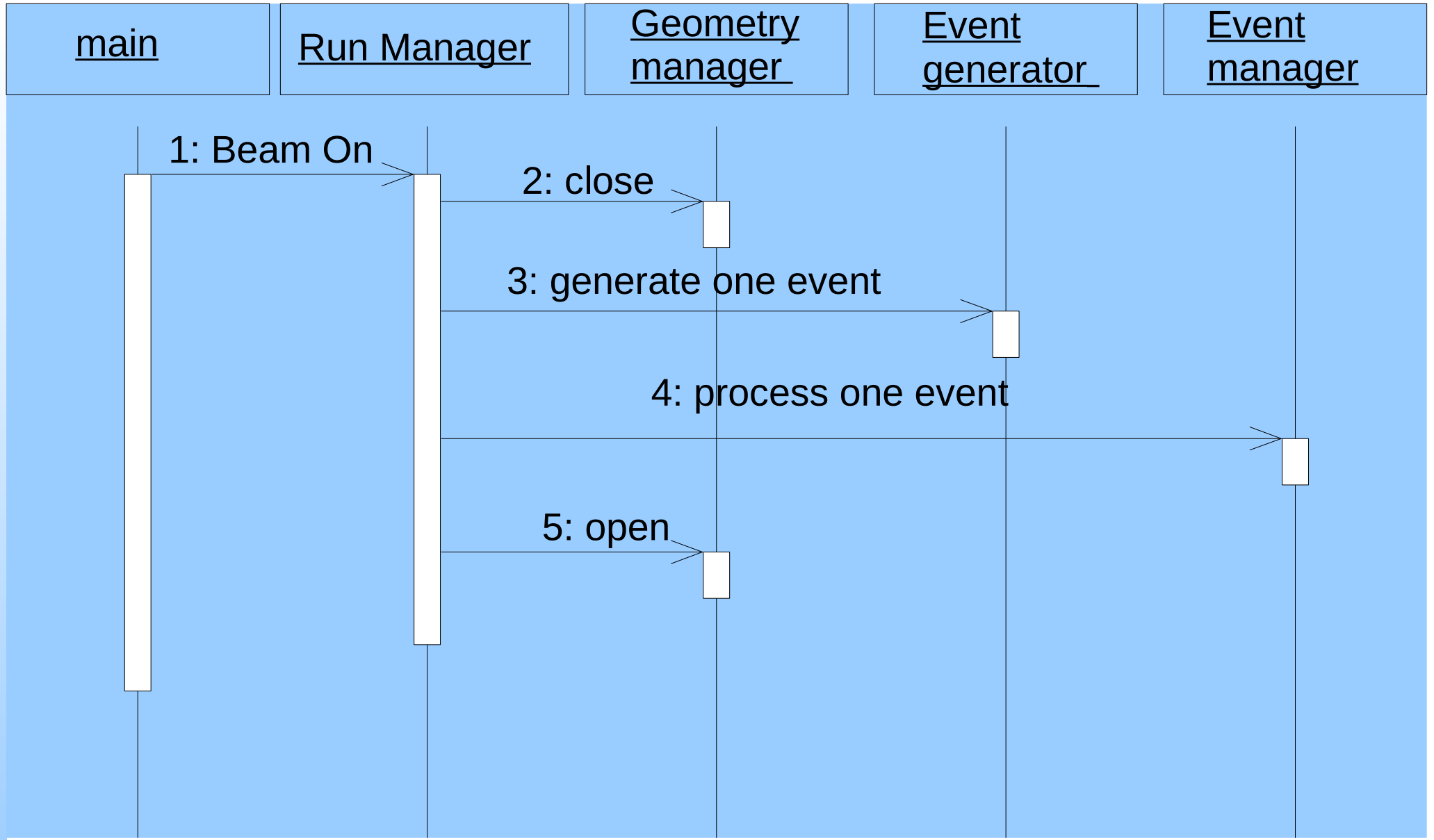
G4RunManager

- G4RunManager is the root class of the Geant4 hierarchy
- It controls the main flow of the program:
  - Constructs the manager classes of Geant4 (in its constructor)
  - Manages initialization procedures including methods in the user initialization classes (in its method Initialize())
  - Manages event loops (in its method BeamOn())
  - Terminates manager classes in Geant4 (in its destructor)
- The method Initialize() takes care of building the detector geometry (as specified by the user), the physics processes and of setting all parameters needed for G4 to run
- The detector setup and the physics processes and cuts cannot be modified during a run. The G4RunManager must be notified if one of these were to change, before a new run

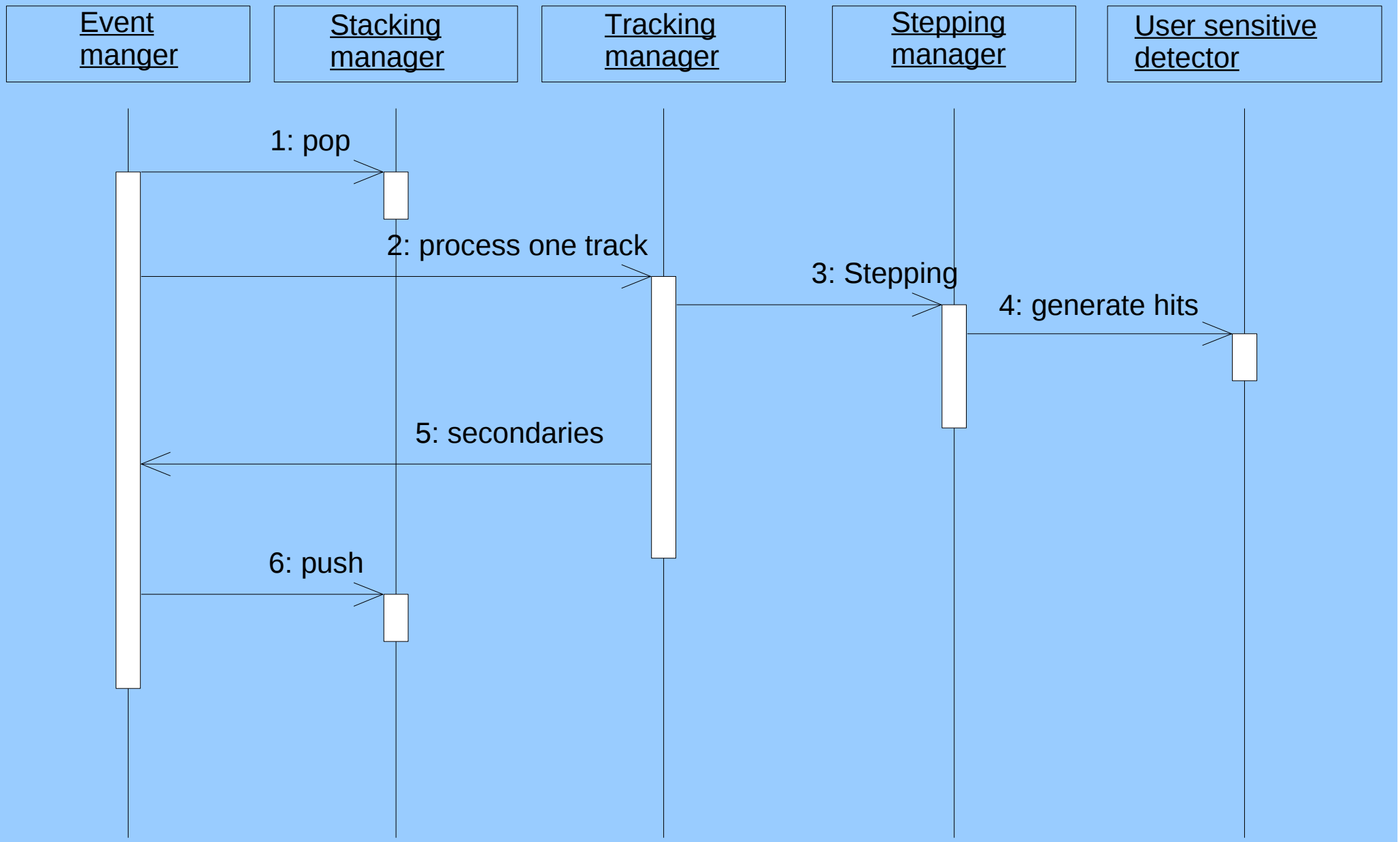
G4RunManager: init



G4RunManager: beamOn



Event loop


Event manger
Stacking manager
Tracking manager
Stepping manager
User sensitive detector
1: pop
2: process one track
3: Stepping
4: generate hits
5: secondaries
6: push

Interaction with user code

- User can instruct G4 to run some specific code at given points of the simulation
- Code must be wrapped into a UserAction
- Several types of user actions, depending on what you want do to with your code
  - RunAction allows you to specify code to be run at beginning and end of a run
  - EventAction: executes user code at beginning and end of event
  - TrackingAction: each time a track is created or killed, your code will be run
  - SteppingAction: runs user code at EACH STEP. Critical for CPU performance

Production threshold

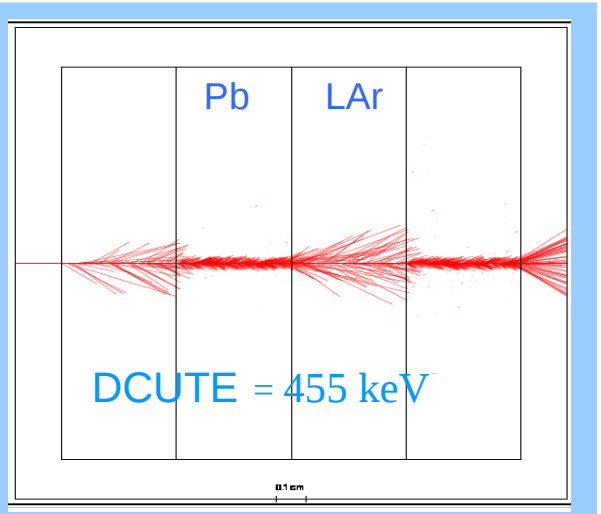
- One of the most critical parameters of any G4 simulation.
- Secondaries below threshold will not be tracked by G4.
- Instead, they will be killed, and their energy given back to the simulation as a local deposit.
- It has a big impact on simulation time per event
  - Should be as big as possible
- On the other hand, a very big value would lead to incorrect energy response by detectors (i.e. calorimeters)
- Also a very small value could give unphysical results
  - A balance must be found.
  - One typically tries several cut values, looking for a region where physical observables (like visible energy in calorimeters) do not depend too much on the actual value of the cut.

Production threshold (2)

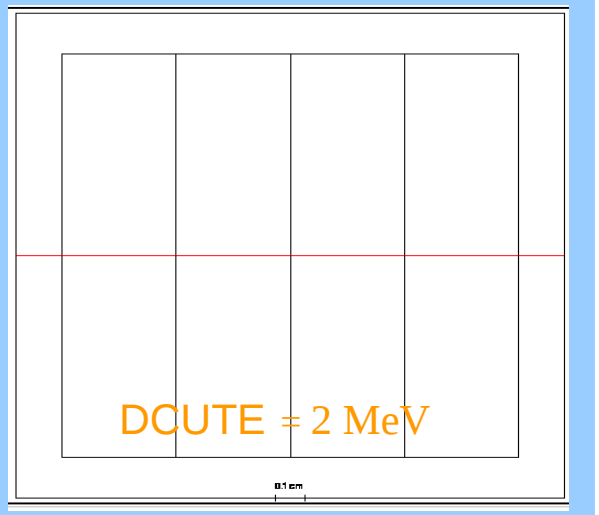
- In G4, all production thresholds for secondaries are given in range, not in energy

Geant3

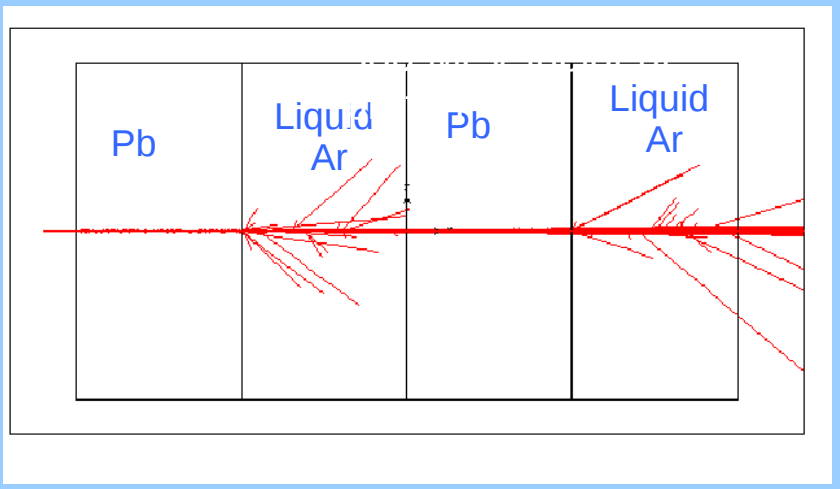
Set energy cut at Ar value;



Set energy cut at Pb value, killing also secondaries in Ar...



Geant4



(e-) range cut: 1.5 mm

455 keV in liquid Ar

2 MeV in Pb

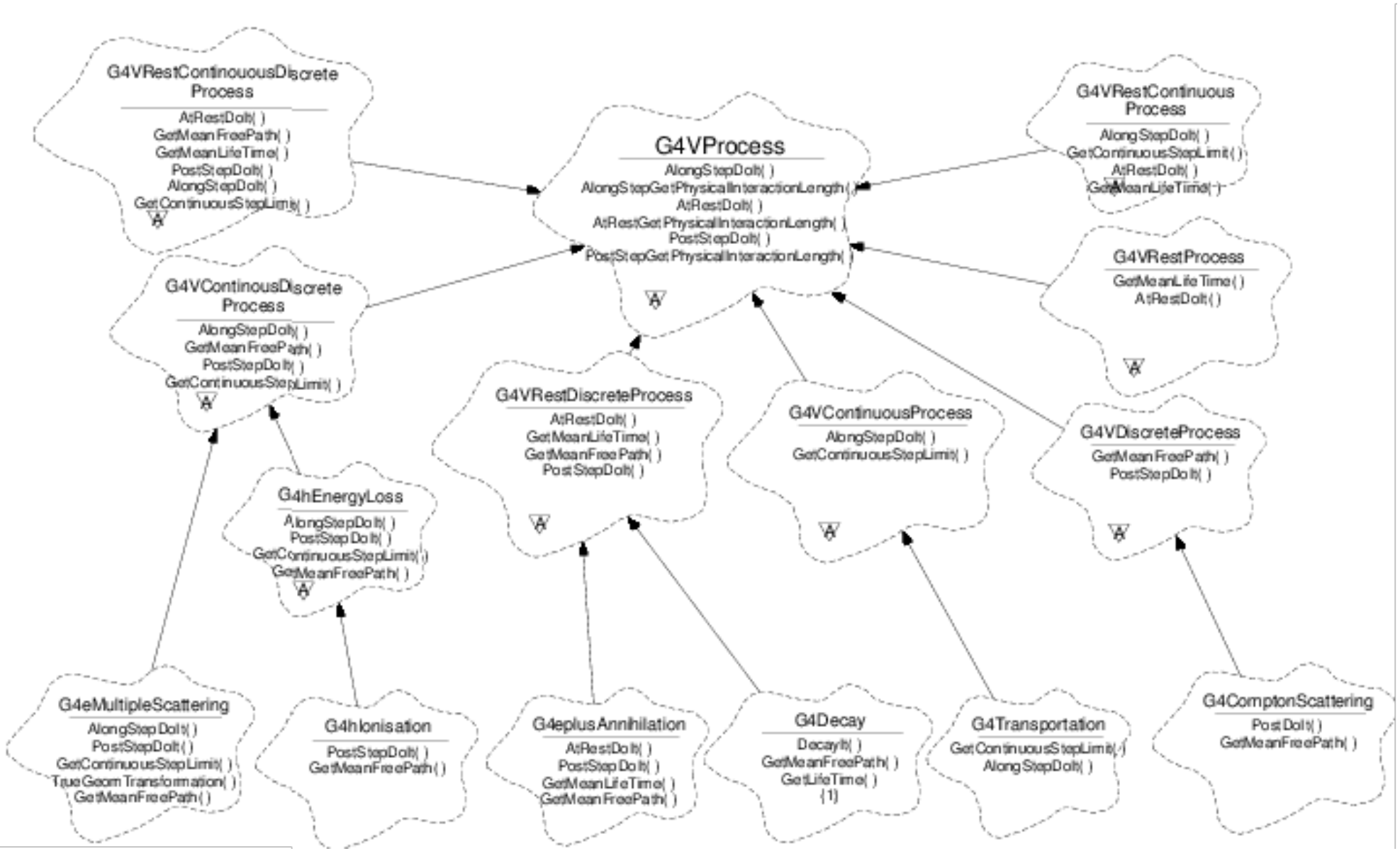
Production threshold (3)

- One in principle can set different production cuts for individual volumes
- More often, a default cut is used for all the simulation, and special cuts are applied just to specific groups of volumes (called *physics regions*)
- A process can still decide to produce secondary particles even below the recommended production threshold
  - if, by checking the range of the secondary produced against quantities like safety (~the distance to the next boundary), it turns out that the particle, even below threshold, might reach a sensitive part of the detector
  - when mass-to-energy conversion can occur, to conserve the energy. For instance, in gamma conversion, the positron is always produced, even at 0 energy, for further annihilation

Physics processes

- A class G4VProcess is provided as the base class for all physics processes
- All physics processes are described by using three (virtual) methods:
  - AtRestDoIt()
  - AlongStepDoIt()
  - PostStepDoIt()
- The following classes are then used as base classes for simple processes
  - G4VAtRestProcess Process with AtRestDoIt() only
  - G4VContinuousProcess Process with AlongStepDoIt only
  - G4VDiscreteProcess Process with PostStepDoIt
- 4 additional classes (such as G4VContinuousDiscreteProcess) are provided for complex processes

Physics processes



Hits/digits

- Sensitive volumes in the simulation must be associated to a SensitiveDetector
- At each step, G4 checks whether the volume the particle is in is sensitive, and calls the ProcessHits method of the corresponding SensitiveDetector.
- Any Sensitive Detector has three major methods:
  - Initialize(): it is invoked at the beginning of each event.
  - ProcessHits(): it is **invoked by G4SteppingManager when a step takes place** in the G4LogicalVolume which point to this sensitive detector. The first argument is a G4Step object for the current step. Here, one or more G4VHit objects should be constructed if the current step has to be registered
  - EndOfEvent(): This method is invoked at the end of each event. For example, here associate hits collections to the corresponding hit collection

Hits/digits

- Each time ProcessHits() is called, the SD will generate a Hit
- A Hit is a snapshot of a physical interaction of a track in a sensitive region of the detector
  - One can store various informations associated with a G4Step object, like:
    - Position and time of the step
    - momentum of the track
    - energy deposition of the step
    - geometrical information
- In general, *hits* represent the *physics* of the *detection* mechanism, while *electronics* simulation is taken into account when creating *digits*
- Example: muon crossing scintillator slab. Will do many steps, hence produce many hits. The PM however would produce only one signal of a given shape
  - Digitization should group together the hits and create the digit
    - This is where detector experts are really needed

Bruno simulation: present status

- Simulation steered by C++ code (Bruno.cc)
- Possibility to use gdml as source of geometry
- Possibility to choose between two custom physics lists
  - Needs recompilation
- Hit recording in place, by means of a UserAction-based mechanism (AnalysisManager)
- No digits yet

What is missing: MCTruth

- Hits (digits) take into account detector response
  - They are the input for reconstruction
  - Their representation in memory could in principle be identical to the one used for real data
- Of course, when running simulation, you know many more things
  - Particle type, name of the process which originated it, exact position of the vertex where it was created, etc.
  - A HUGE amount of information, which needs to be somehow selected and stored on disk
    - Could include it in hits. Bad for many reasons. For example: you can have many hits from the same true particle and don't want to replicate info. Or, you can have a true particle not giving any hit and still want to record it
    - Better to use a separate class

MCTruth: proposal

- It can be handled using a stepping action
  - A list of interesting processes, particles, energy thresholds is provided for concerned volumes
  - At each step, check whether we are in one of the “truthable” volumes, check the particle type, the process which generated it, its energy
  - If all OK, store info
- It should not be too difficult to implement
  - Change in configuration must not need recompilation
    - Use external config files
  - Trick is to optimize the code as much as possible
    - it will be executed AT EACH STEP

Physics lists

- In G4, a process manager is associated to each particle
  - “knows” which processes the particle can undergo (and in which order)
- The code which “fills” the particle managers for all the particles is called Physics List
  - Writing a physics list is a painful exercise
  - And dangerous too, unless you are a real expert of the underlying G4 physics models
- G4 provides a set of recommended physics lists, which are used as the baseline for most experiments
  - Validated by experiments
  - Maintained by G4 developers
- Unless one wants to do some very specific tests, it is in general a good idea to try those before writing one's own list
- Default physics lists distributed with G4 release.
  - We already have them, it's just a matter of telling Bruno to use them

Physics regions

- Aim is not to waste time in tracking very low range particles
- Production cuts can be defined up to a volume-by-volume basis by using regions
  - They associate to a volume (and all its children) a set of production cuts
- Presently, regions are created during geometry definition (if not using gdml)
  - Cuts are associated to regions in the physics list
- Could be a good idea to do region creation and cut definition in one single step, decoupled (from the point of view of the code) from both geometry and physics
  - Implemented a working prototype: see following slides

User Interface

- Classical way of interacting with simulation program is by means of macro files
- Doesn't scale well with growing size and complexity of the program
- Geant4 implements, in latest versions, a new python interface
  - Uses boost for python binding generation
    - Not G4 depending: can be used by any C++ code
  - Exposing to python a subset of core classes and methods
  - Easy to add user classes too
- Advantages:
  - No need to recompile each time you change an option
  - Fully-fledged scripting language
  - Flexibility, maintainability
  - User friendliness

Prototype of "pythonized" Bruno

- As a proof of concept, tried to reimplement Bruno.cc in python
- Design can be optimized, but for the time being it splits the program in two parts
  - BrunoEng.py: actual manipulation of high level G4 classes (like the run manager) which must be protected from the user
  - BrunoConf.py: container for user options, to be passed to the Engine
  - User writes his/her own python script, using a BrunoConf to wrap options and feeding it to a BrunoEng
- We exposed AnalysisManager and all the UserActions to python too
- When doing this excercise, we found that the physics list selection comes for free, as a bonus...
  - In any case, Bruno.cc can now choose between 3 standard physics lists at run time, using a command line switch

pyBruno

```
#!/usr/bin/python

from Geant4 import *

from BrunoEng import *
from BrunoConf import *

theConf.PhysicsList="QGSP"
theConf.GeoName="SuperB.gdml"
theConf.setVerbosity(1,1,1)
theConf.OutputRootFile="myOutput.root"

theEng.setConfig(theConf)

theEng.prepareForRun()
```

testRun.py

```
[superb@localhost pyBruno]$
[superb@localhost pyBruno]$ ./testRun.py

*************************************************************
 Geant4 version Name: geant4-09-01-patch-03    (12-September-2008)
                      Copyright : Geant4 Collaboration
                      Reference : NIM A 506 (2003), 250-303
                            WWW : http://cern.ch/geant4
*************************************************************

Visualization Manager instantiating...

<<< Geant4 Physics List engine packaging library: PACK 5.4
<<< Geant4 Physics List simulation engine: QGSP 3.3
```

- Example run
  - User edits a py file and executes it
  - Under the hood, pyBruno does all the work
  - See next slide...

```
[superb@localhost pyBruno]$ ./testRun.py

*************************************************************
 Geant4 version Name: geant4-09-01-patch-03    (12-September-2008)
                      Copyright : Geant4 Collaboration
                      Reference : NIM A 506 (2003), 250-303
                            WWW : http://cern.ch/geant4
*************************************************************

Visualization Manager instantiating...

<<< Geant4 Physics List engine packaging library: PACK 5.4
<<< Geant4 Physics List simulation engine: LHEP 4.2
```

pyBruno: under the hood

```python
class BrunoConf(object):

    def __init__(self):
        #not really needed, but sort of more clear this way
        self.PhisicsList=str()
        self.GeoName=str()
        self.Verbosity=list()
        self.OutputRootFile=str()
    def setVerbosity(self,run,event,track):
        self.Verbosity=[run,event,track]

theConf=BrunoConf()
```

BrunoConf.py

```python
    def setGeo(self):
        if(self.config.GeoName[-4:]=='gdml'):
            detconstr=pyBruno.gogdmlDetectorConstruction()
            detconstr.setFile(self.config.GeoName)
            gRunManager.SetUserInitialization(detconstr)

    def setRegions(self):
        pass

    def setPhysics(self):
        if(self.config.PhysicsList=="QGSP"):
            gRunManager.SetUserInitialization(QGSP())
        elif(self.config.PhysicsList=="LHEP"):
            gRunManager.SetUserInitialization(LHEP())
        elif(self.config.PhysicsList=="QGSP_BERT"):
            gRunManager.SetUserInitialization(QGSP_BERT())
```

BrunoEng.py
(geo + phyis list)

- Code snippets from BrunoConf and BrunoEng
- Presently, BrunoEng is doing the same things as Bruno.cc
  - Apart from minor things (such as command line parsing, not needed anyway)
  - Primary generator is there, but a way to configure it still to be implemented
  - Note the hook for regions...

```python
    def setVerbosity(self):
        gApplyUICommand('/run/verbose '+str(self.config.Verbosity[0]))
        gApplyUICommand('/event/verbose '+str(self.config.Verbosity[1]))
        gApplyUICommand('/tracking/verbose '+str(self.config.Verbosity[2]))

    def prepareForRun(self):
        self.setGeo()
        self.setPhysics()
        self.setActions()
        self.setGenerator()
        self.setVerbosity()
        self.setAnalysis()
        # put here the field dumper
         myDumper=pyBruno.getBFieldDumper()
         myDumper.book()
         myDumper.BeginOfRunAnalysis(None)
         myDumper.finish()
        gRunManager.Initialize()
        gRunManager.BeamOn(1)
```

BrunoEng.py
(verbosity + steering)

pyBruno: Physics Regions

- BrunoRegionManager C++ class manages region definition, cut setting, association with logical volumes
  - This can be used also by Bruno.cc
- Relevant methods exposed to python for usage by BrunoEng
- A “buffer” class in python gathers all user info about regions
  - BrunoEng loops over these python regions and creates the “real” ones using the BrunoRegionManager
  - User is protected from the G4 kernel (and viceversa)
  - Freedom to add regions or modify existing ones until the very last minute from top python script
- Things look pretty good, and prototype works (see next slide)
  - A python module BrunoRegions.py was created to store regions for default productions
  - Only issue now is understand to which volumes we want to associate regions

pyBruno: Physics Regions (2)

BrunoRegions.py

```
from Geant4 import *
from BrunoConf import theConf, Region

# start by defining default region for the world

worldRegion=Region("worldRegion")
worldRegion.addVolume("World")
worldRegion.addCut("gamma",0.001*mm)
worldRegion.addCut("e-",0.001*mm)
worldRegion.addCut("e+",0.001*mm)

theConf.addRegion(worldRegion)


# now suppress secondaries in vacuum

solenoidRegion=Region("solenoidRegion")
solenoidRegion.addVolume("The_solenoidal_field_1")
solenoidRegion.addCut("default",1000*km)

theConf.addRegion(solenoidRegion)

QD0_HER_VacRegion=Region("QD0_HER_VacRegion")
QD0_HER_VacRegion.addVolume("qd0_inner_bmp_logHER_IN")
QD0_HER_VacRegion.addCut("default",1000*km)

theConf.addRegion(QD0_HER_VacRegion)
```

BrunoEng.py

```
from Geant4 import *
import pyBruno

class BrunoEng(object):

    def __init__(self):
        pass
    def setConfig(self, config):
        self.config=config

    def setGeo(self):
        if(self.config.GeoName[-4:]=='gdml'):
            detconstr=pyBruno.gogdmlDetectorConstruction()
            detconstr.setFile(self.config.GeoName)
            gRunManager.SetUserInitialization(detconstr)

    def setRegions(self):
        # loop over configured regions
        for region in self.config.Regions:
            rm=pyBruno.getRegionManager()
            for volume in region.Volumes:
                rm.createRegion(region.Name,volume)
            for particle in region.Cuts.keys():
                rm.addCut(region.Name,particle,region.Cuts[particle])

    def setPhysics(self):
        if(self.config.PhysicsList=="QGSP"):
            gRunManager.SetUserInitialization(QGSP())
        elif(self.config.PhysicsList=="LHEP"):
            gRunManager.SetUserInitialization(LHEP())
        elif(self.config.PhysicsList=="QGSP_BERT"):
            gRunManager.SetUserInitialization(QGSP_BERT())
```

- Cut definition is done in the python module
  - Only “official” cuts should go here
    - No user interaction
  - User can add/modify things from his/her top script (see next slide)

PyBruno: Physics Regions (3)

testRun.py

```
#!/usr/bin/python

from Geant4 import *

from BrunoEng import *
from BrunoConf import *

import BrunoRegions

# modify one of the existing cuts

myRegion=theConf.getRegion("worldRegion")
myRegion.Cuts["gamma"]=0.002*mm

theConf.PhysicsList="LHEP"
theConf.GeoName="SuperB.gdml"
theConf.setVerbosity(1,0,0)
theConf.OutputRootFile="myOutput.root"

myGenerator=Generator()
myGenerator.Seed=1234
#myGenerator.Name="Bbb"
#myGenerator.BbbPrescaling=10.0
#myGenerator.BbbMinDeltaE=0.001

#myGenerator.Name="ReadFile"
#myGenerator.InputFile="filename.dat"


theConf.Generator=myGenerator

theEng.setConfig(theConf)

theEng.prepareForRun()

theEng.beamOn(1)
```

```
========= Table of registered couples ==============================

Index : 0     used in the geometry : Yes     recalculation needed : No
 Material : Vacuum
 Range cuts        :  gamma 700 um     e- 700 um     e+ 700 um
 Energy thresholds :  gamma 990 eV     e- 990 eV     e+ 990 eV
 Region(s) which use this couple :
    DefaultRegionForTheWorld

Index : 1     used in the geometry : Yes     recalculation needed : No
 Material : Air
 Range cuts        :  gamma 700 um     e- 700 um     e+ 700 um
 Energy thresholds :  gamma 990 eV     e- 990 eV     e+ 990 eV
 Region(s) which use this couple :
    DefaultRegionForTheWorld
```

```
Index : 28     used in the geometry : Yes     recalculation needed : No
 Material : emcAir
 Range cuts        :  gamma 1 um     e- 1 um     e+ 1 um
 Energy thresholds :  gamma 990 eV     e- 16.8647 keV     e+ 16.8647 keV
 Region(s) which use this couple :
    worldRegion
```

```
Index : 52     used in the geometry : Yes     recalculation needed : No
 Material : Air
 Range cuts        :  gamma 1000 km     e- 1000 km     e+ 1000 km
 Energy thresholds :  gamma 990 eV     e- 990 eV     e+ 990 eV
 Region(s) which use this couple :
    solenoidRegion
```

- Example of region customization from the user's side
- Log showing that the regions are actually being used by G4

pyBruno: open issues

- Prototype is working
  - Geometry is build
  - Physics list is chosen at run time
  - Physics regions can be defined and modified at runtime
    - A default list is provided
  - Event loop is carried on as expected
- Some features missing, but it shouldn't take too long to be able to replace Bruno.cc completely
  - All you have seen here was done in ~2 days of work
- My proposal is that we work in this direction
- In the meantime, take care of backporting all new functionalities to Bruno.cc

pyBruno: open issues

- Python is not the solution to everything:
  - Good for code to be executed once per run (i.e. configuration)
  - Critical code (i.e. executed more often) should still be coded in C++ for performance reasons, but some parts can be exposed to python
    - For example a SteppingAction MUST be written in C++ (note: it IS possible to implement it in python, with no C++ at all)
    - However, the configuration part can be exposed to the python layer
    - Immediate application for example to the MCTruth problem

Conclusions

- Three main missing functionlities
  - MCTruth
  - Handling of regions
  - User interface
- At least one (the physics list setting) is solved almost for free if we switch to python interface
- The rest needs some coding, but it should be possible to implement in a fairly short timescale
- Longer term targets may include:
  - use of parallel geometries for scoring volumes
  - Implementation of proper digitization