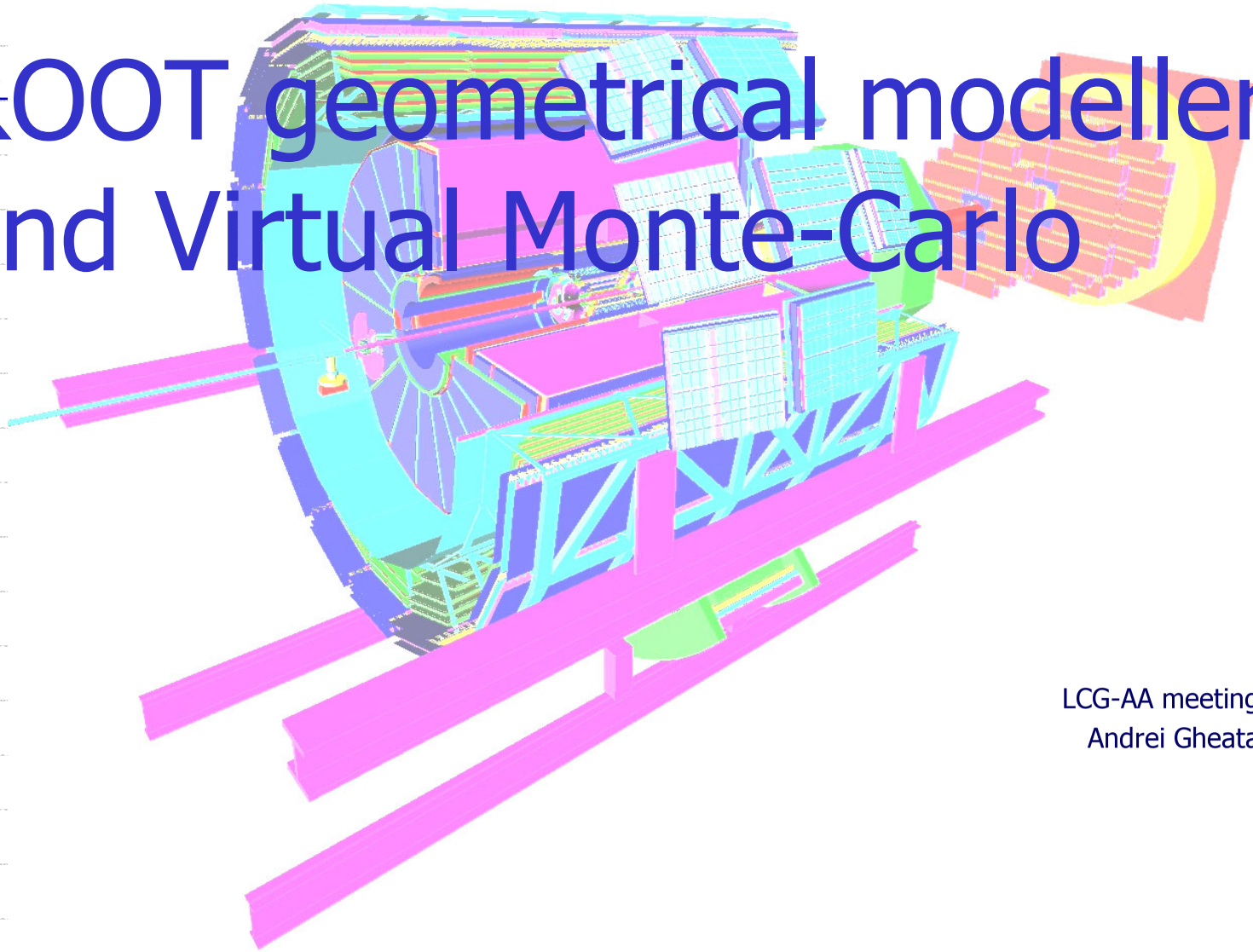
ROOT geometrical modeller and Virtual Monte-Carlo



LCG-AA meeting
Andrei Gheata

Topics

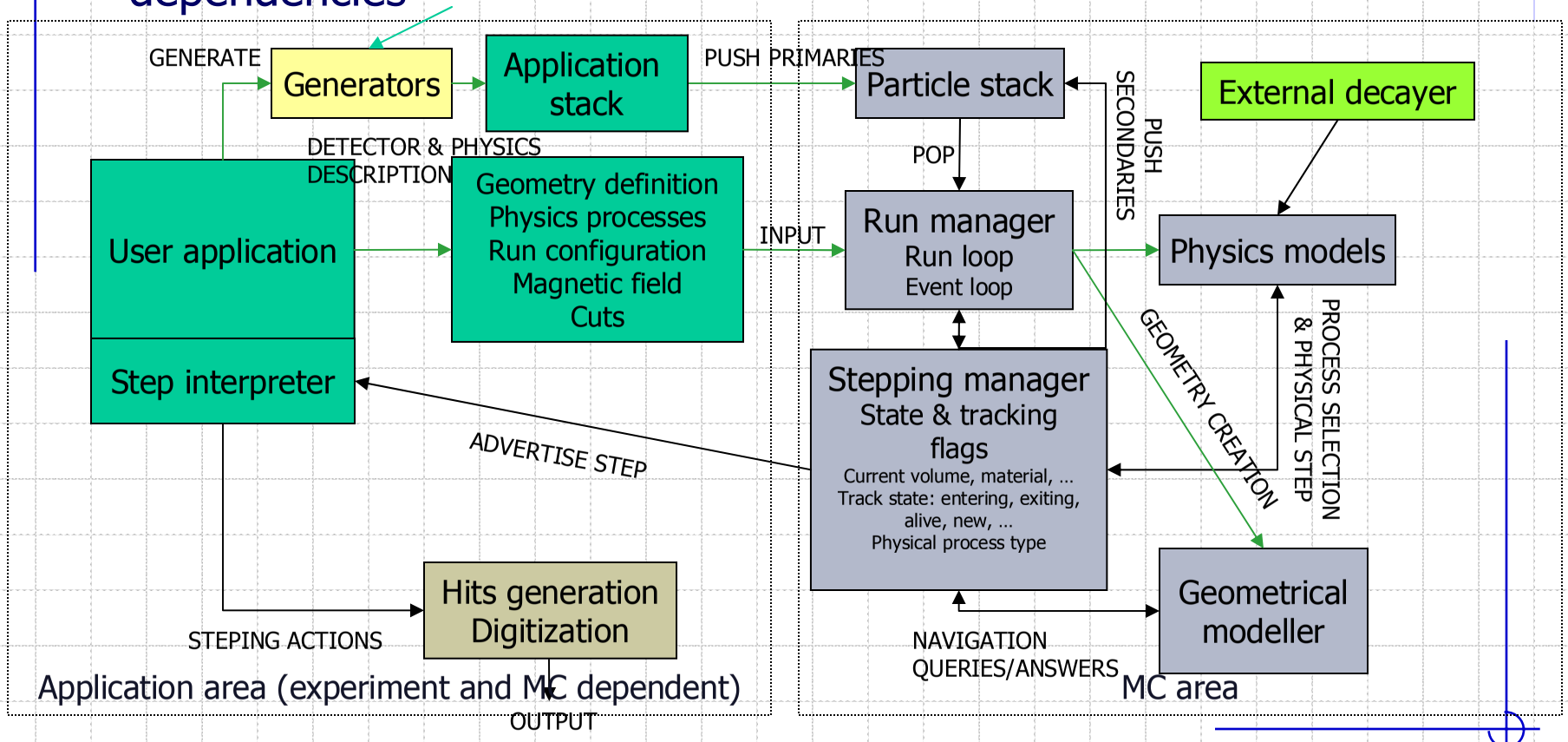
- Motivation
- The concept of VMC
- VMC evolution
- TGeo general description
- Navigation functionality and additional tools
- Performance issues
- VMC & GEANT3, GEANT4, FLUKA
- TGeo to-do's
- VMC to-do's
- Conclusions

Main initial VMC motivation

- There are several MC simulation engines on the market (GEANT3, GEANT4 and FLUKA – to quote the most well-known)
- These are all made for the same purpose (e.g. to try modeling our best knowledge in particle/detector physics), but certainly are using different ways to achieve that
- When far beyond existing experimental data (e.g LHC), we have poor means to cross-check MC predictions
  - The complexity of LHC experiments make things even worse
- One would surely wish to have at least a second opinion related to the "MC truth" …
  - BUT, can one big experiment afford a development effort proportional to the number of "opinions" ?
- … by running different MC's starting with the same code.

First look at a classical simulation framework

- Experiment/MC – tailored application with heavy inter-dependencies



Application/MC insulation

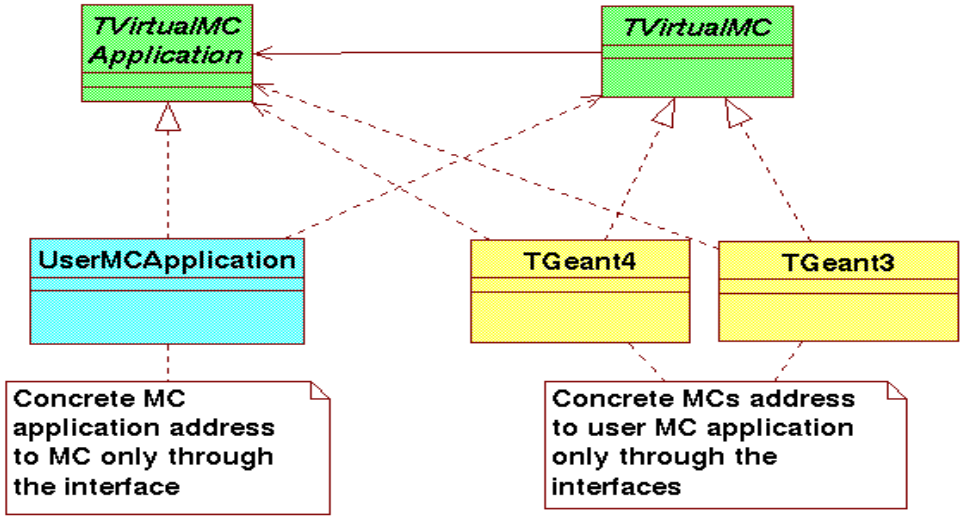
- The way application is designed highly depend on the MC used
  - To minimize/eliminate MC dependencies we surely need an insulation layer at MC level: **TVirtualMC**
    - Common denominator for G3, G4 and FLUKA for "generic functionality" concerning:
      - Geometry/material creation
      - Physics configuration, processes and cuts
      - Content for tracking information
      - Stack handling, stepping management
  - An insulation layer is needed also for the user application
    - All application-MC interoperability methods.
      - Stepping callbacks
      - Particle stack
      - Navigation info

VMC evolution

- Just a layer on top of GEANT3 at the beginning (1999)
  - *TGeant3* allowed only communication to G3
- Generalized to **TVirtualMC** to eliminate dependencies to MC (2000)
  - Still dependent on ALICE-specific code due to callbacks
  - API inspired by G3
- Start development of *TGeant4* (I. Hrivnacova) (2001)
  - Main problems related to geometry mapping: G3 *MANY* option and reflections – partially solved later by G4 team
- *TVirtualMCApplication* introduced to eliminate dependencies to experiment software (2002)
- Few abstract classes added in the VMC schema: (2003)
  - *TVirtualMCStack*, *TVirtualMCDecayer* – interfacing a user defined stack and decayer

The first VMC release (2003)

- Authors: R.Brun, F.Carminati, I.Hrivnacova, A.Morsch
- Abstract layers on top of both user MC application and MC simulation engine
- Interoperability done only via calls to abstract classes
- Creating and running with MC native geometries
- G3 and G4 interfaces
- 3 examples from G4



VMC interfaces

- TVirtualMCApplication – representing the application (mandatory)
  - User application must derive from it an implement some pure virtual methods
  - Only calls to TVirtualMC allowed
    - Primary particles generation
    - Geometry creation, run/event/step control
- TVirtualMC – interface to the MC used (provided)
  - API inspired by GEANT3
  - Provide methods for: building and accessing geometry & materials, setting physics, accessing tracking information and run control
  - Implementations to specific MC's are provided : GEANT3, GEANT4 and FLUKA

ConstructGeometry
InitGeometry
GeneratePrimaries
BeginEvent
BeginPrimary
PreTrack
Stepping
PostTrack
FinishPrimary
FinishEvent

Other interfaces

- TVirtualMCStack – stack for users particles (mandatory)
  - Must implement methods for pushing/popping (T)particles to/from the stack
  - Called by *TVirtualMCApplication::GeneratePrimaries()* to initially fill the stack
  - Communicates with MC specific stacks via *Pop()* method
  - Nothing fancy to implement (see VMC examples)
- TVirtualMCDecayer – interface to an external decayer (optional)
  - Provides a way to take control on some particle decays
    - E.g. Pythia6

Geometry problems

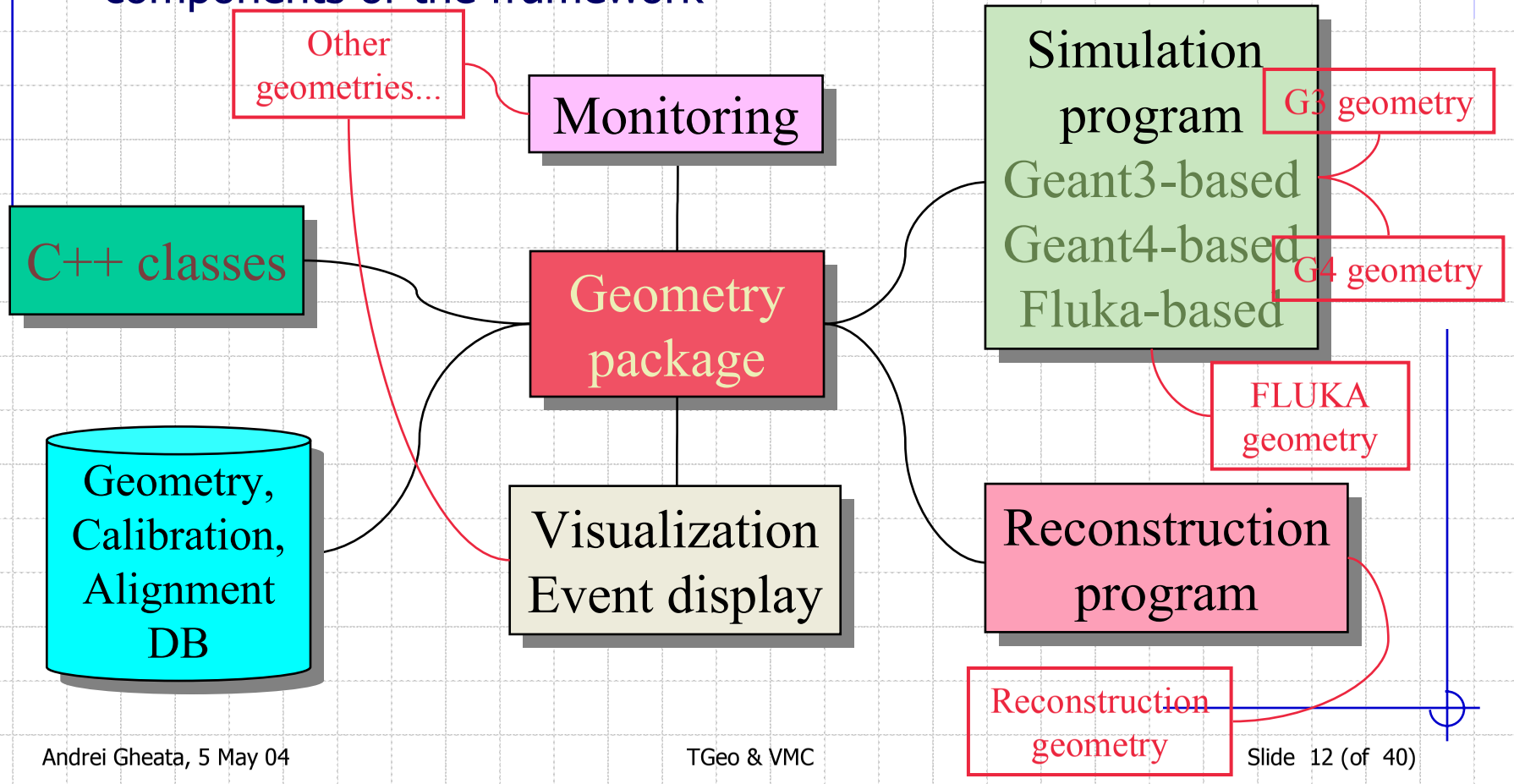
- Geometry – the weak point in the first VMC
  - The goal is to be able to do simulation comparisons between different MC's under the same conditions
    - Building native MC geometries starting from the same initial geometry description does not insure having identical geometrical answers for a given query
      - G3, G4 and FLUKA geometries can be just partially or not at all mapped one to each other – even building the native model becomes an unsolvable problem
    - Not having the same geometry at simulation time – forget about comparisons at detector scale, even if everything else works smoothly
  - Eventually one has to feed simulated data to reconstruction algorithms that require also a geometry – which one ?

Geometry external to MC ?

- Using in the VMC a single external geometrical modeller – the obvious solution to all problems
  - Not as obvious if possible – simulation MC's know about their OWN geometries to which they ask questions
    - This can be worked-around with some effort (wrappers/commons for FORTRAN, abstract layers for C++)
    - Still, the tool should provide full navigation functionality, be able to represent GEANT-like geometries, scale CPU resources and be reasonably fast
      - We looked around (CSG modelers, CAD systems) – nothing even close
  - Only alternative – developing a new modeller (fall 2001)
    - Quite big effort but worth making – other applications in the experiment framework would also benefit : reconstruction, event display

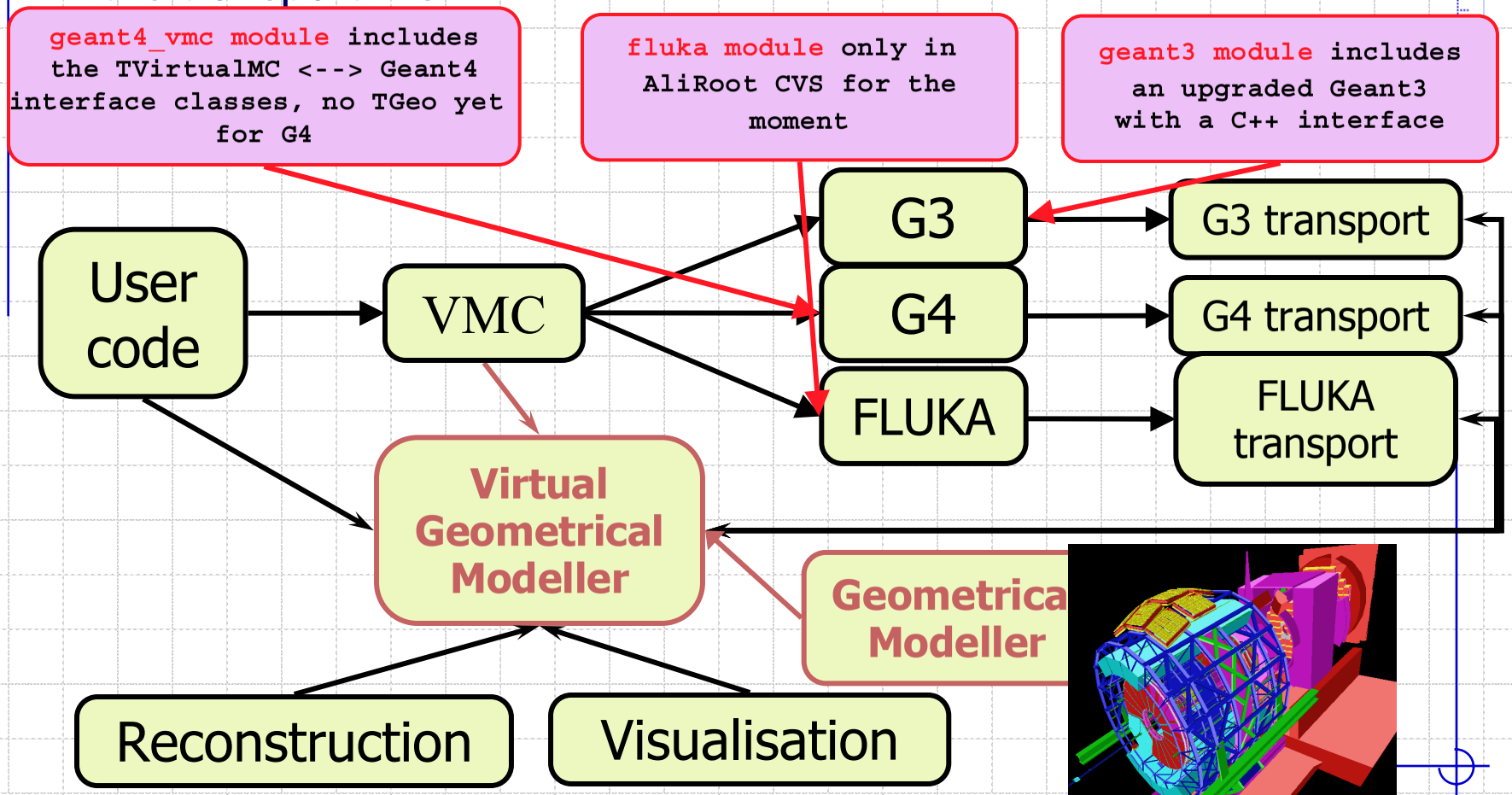
Some benefits

- Using a single geometry representation accessible by all components of the framework



TVirtualMCGeometry

- The idea is to intercept/answer geometrical queries posted by the transport MC



TGeo model description

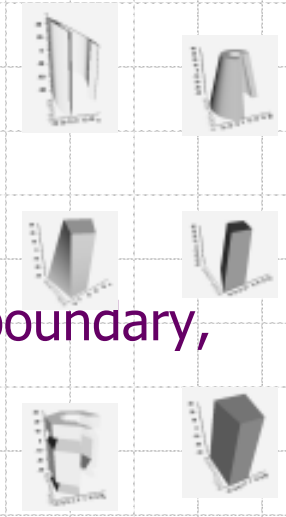
- GEANT-like CSG-based model with constraints.
  - For portability and efficiency reasons;
  - The model is build-up from elementary “bricks” (primitive shapes) put together via hierarchical structures (volumes and nodes);
  - Containment is the main constraint.
- Object position information stored in a relative rather than absolute way.
  - In this way information is quite compressed (logical hierarchy) and fits to memory even for huge geometries;
- Large number of primitive shapes (18) and possibility to make Boolean combinations of any ‘reasonable’ number of partners
  - This leads to a huge combinatorial – accurate model description

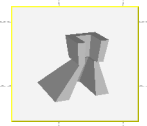
Main features

- Based on some initial requirements
  - Provide basic navigation features : *"Where am I?", "How far from next boundary ?", ...*
  - Map Geant3 geometries -> smooth transition
  - Scalability : we deal with big geometries
  - Performance : it rather be faster than existing modelers
  - Interactivity : should be at highest possible level - users should easily build, access and debug their geometry
- GEANT-like flavor (CSG based on container-contained concept)
  - main elements : volumes and nodes
- Extensible set of 18 primitives + composite (Boolean) + parameterized shapes + support for "MANY" concept
- Volumetric pixel-ized navigation
- Make use of symmetries – divisions
- Assemblies of volumes – to facilitate geometry building when some containers are hard to define
- Geometry checking interactive tool
- Support for alignment
- Visualization tools
- Ray-tracing

Shapes

- 18 primitive (basic) shapes supported
  - Last shapes: *TGeoTorus*, *TGeoXtru*
- Derived from an abstract class: *TGeoShape*
  - Extension always possible
  - Local navigation queries (point in/out, distance to boundary, safety, normal vector)
  - Visualization algorithms (mesh creation)
- Composite shapes – an easy way to make any combination
  - (A:t1+B:rot1)-(A:t2*C) : creates a binary CSG hierarchy
  - Not to be abused…

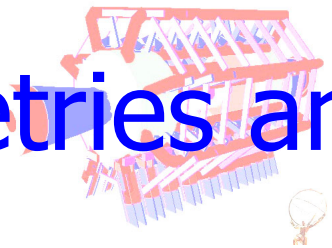
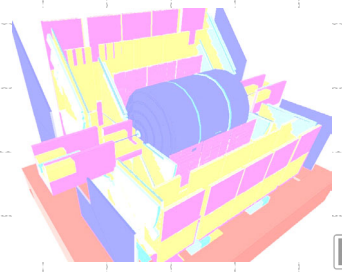
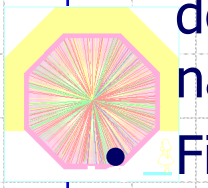
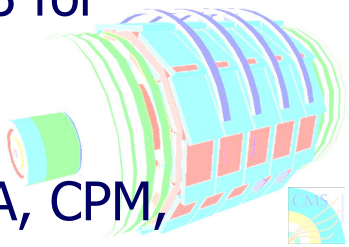
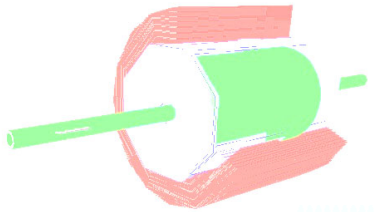
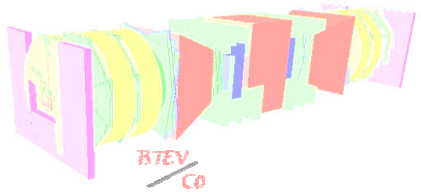
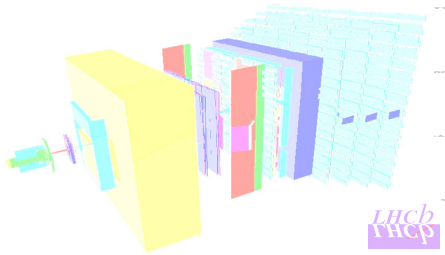
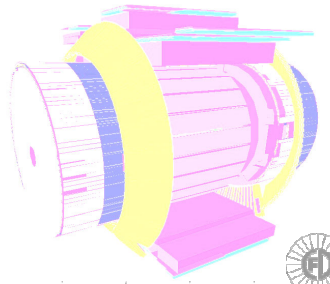




The geometrical hierarchy

- **Volumes and nodes** : logical objects used to arrange shapes in a hierarchical structure and to assign them physical properties (materials, tracking media) and a relative positioning.
- These DO NOT represent a "touchable" geometry object since they cannot store the object GLOBAL transformation in the master frame.
  - C inside B (6 times) + B inside A (10 times)
  - 3 volumes & shapes (A,B,C) / 16 nodes (6 + 10) BUT 60 different "physical nodes" (different paths/global transformations)
  - How many physical nodes in a real geometry? – up to more than 1 billion found so far…
- Physical node = volume + path in the logical hierarchy. These can be represented as objects on demand (visualization and alignment), but are NOT manipulated during navigation. Path information can be always retrieved on demand.

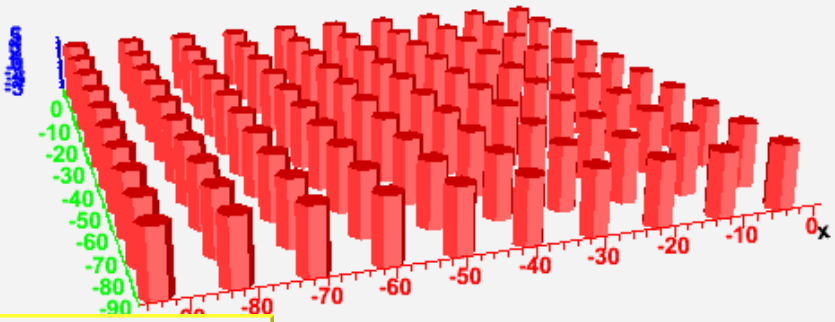
Geometries and TGeo

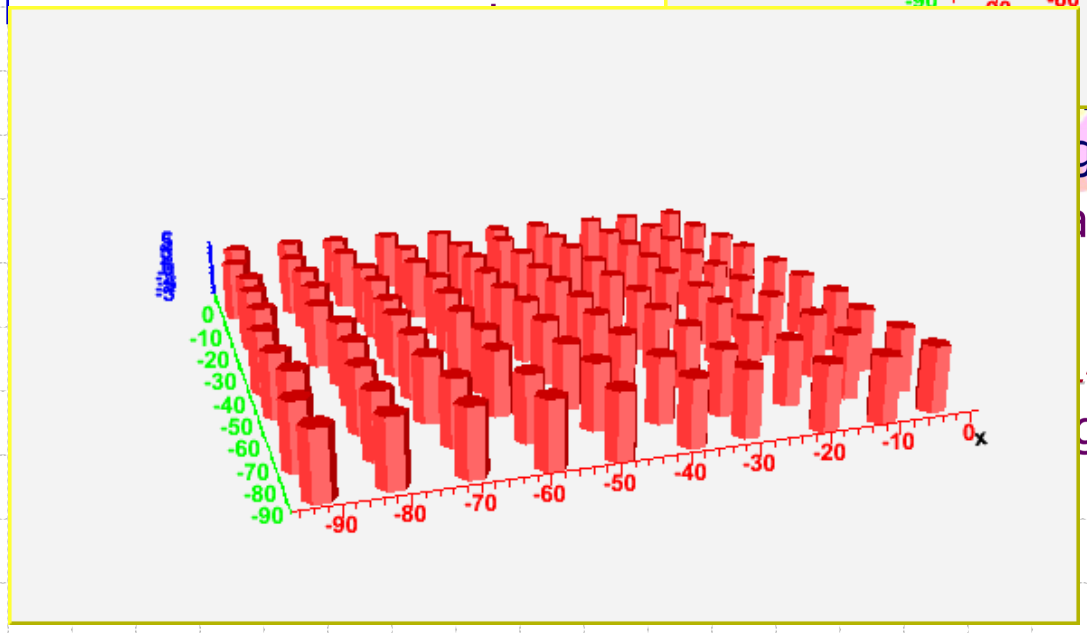










- The modeling features are a superset of those from G3 – G4
- A converter from G3 to TGeo was written in a very early development stage to make sure it can represent/perform navigation tasks for any "ideal" G3 geometry.
- Filling TGeo from external formats (e.g. XML) already requested
- Navigation accuracy and performance tested against G3 for several geometries, including all 4 LHC experiments.
- Few experiments besides ALICE started already making developments based on VMC and TGeo (MINOS, OPERA, CPM, PANDA,…)

Support for real geometries

- Usually during simulation
- The real geometry can be
  - Typically has to be ha
    alignment data
  - The way to deal with
  - Typical task – extrapo





geometry was created.

ted

shape, check);

gnment – optional check

Navigation features

- Allow computation of all geometry parameters needed by particle tracking codes.
- It has been a long process from implementing to validating/optimising/ tuning these features => took most of the time.
  - Luckily we had a gold mine of G3 geometries to test upon.
- *Finding the location in the geometry tree* : up to x20 performance gain compared to GEANT3
- *Computing the distance to next boundary* : up to x8 gain;
- *Computing the safe distance in current object* : when a maximum step limit is provided
- *Computation of the normal vector to crossed surfaces* : on demand
- *Computation of the distance along a helix to the next boundary* : feature currently under testing, to be released in the weeks to come

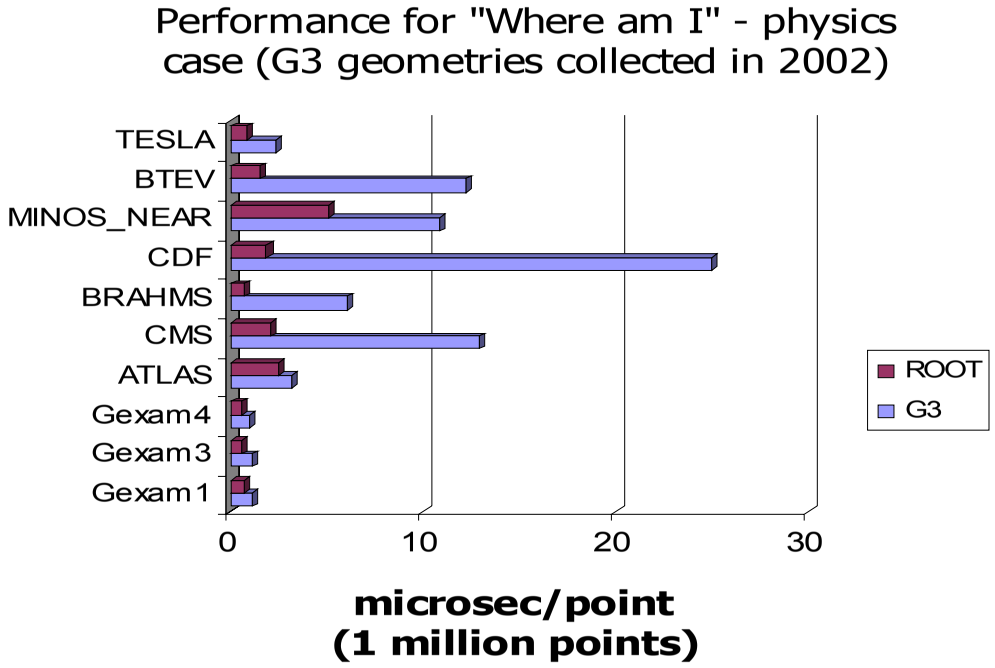
Performance issues

- How much time is spent during simulation simply for navigation?
  - Up to 60%, maybe more for badly designed geometries…

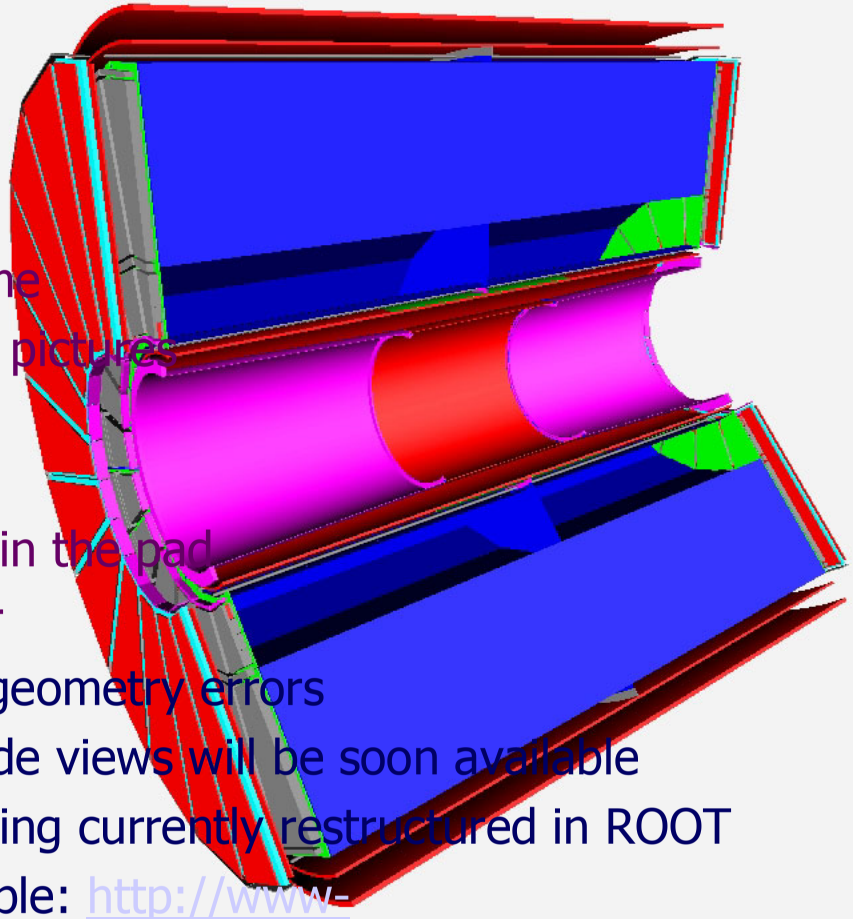
• Easy to badly design a geometry – much hard time to optimize it…

• TGeo has several levels to improve search performance:

  - Bounding boxes
  - Divisions
  - Volumetric pixels
  - Code optimization exercise done by Rene -> things can be improved !



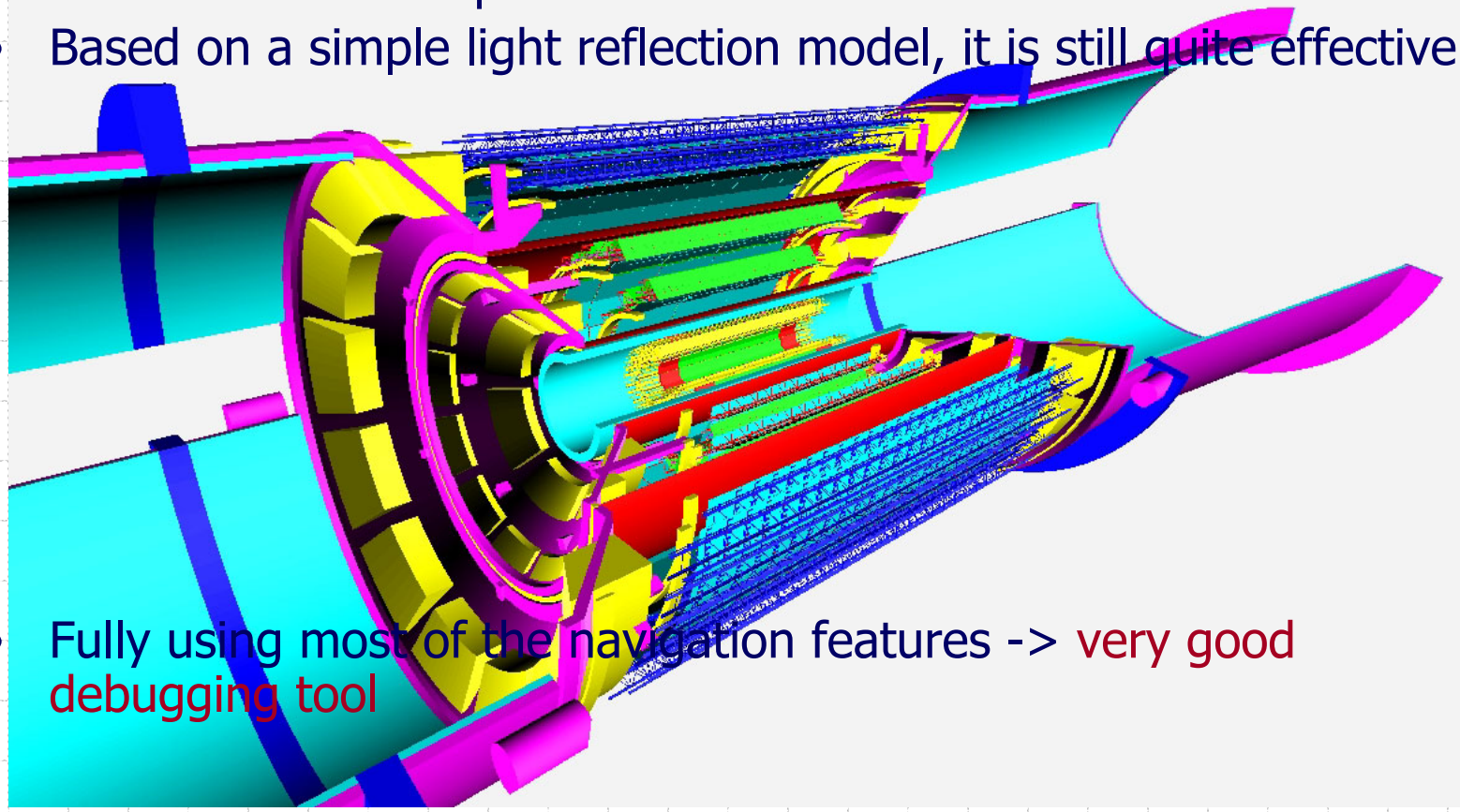
Visualization features



- Several options
  - Depth selection
  - Visibility selection per volume
  - Ray-tracing for high-quality pictures
    - Clipping regions
  - Object picking
  - Rotating/zooming/focusing in the pad
  - X3D-based solid view so far
- Fastest tool for understanding geometry errors
- OpenGL/COIN3D/X3D solid mode views will be soon available
- Geometry 3D visualization is being currently restructured in ROOT
- Interface to CAD systems possible: http://www-hades.gsi.de/~biryukov/CADD/

Ray-tracing

- More time consuming, but allows solid mode high-quality visualization in the pad.
- Based on a simple light reflection model, it is still quite effective



- Fully using most of the navigation features -> very good debugging tool

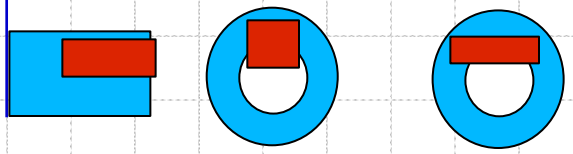
User-defined tracks

- Can be easily made out of (x,y,z) points along a tracked particle.
- Tracks can be visualized in the context of the geometry (drawn in wire-frame mode)
- When time information for each point is available, tracks can be animated.
- Not really support for real track objects, introduced mostly to visualize what was happening in VMC

Geometry checker

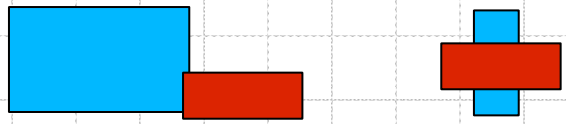
- It has to be able to check (in reasonable time) for illegal extrusions/overlaps in the geometry definition.
  - About 30 sec. Full ALICE
- Extrusions

Detected Not detected



- Overlaps

Detected Not detected



Geometry	Overlaps/Extrusions (~70 s detection time)		
	> 1 mm	>100 μ	>10 μ
ALICE	154	764	1460

No error-free geometry found…
(up to 20K > 1mm)

New VMC design (2004)

- *TVirtualMC* has a *TVirtualMCGeometry*
  - All geometry-building part and geometry state control methods handled now by this.
  - Geometry now completely decoupled from the MC part
  - Navigation control methods highly dependent on the MC – still handled inside the specific TVirtualMC implementations
- Implementation of *TVirtualMCGeometry* for **TGeo** : *TGeoMCGeometry*
  - In practice navigation interface methods and building methods are different for different MC's
  - Redesign to be applied once *TFluka* interface will be fully stable

Interface to GEANT3

- Implementation of TVirtualMC: TGeant3 class
- Available in ROOT
  - *cvs -d :pserver:cvs@root.cern.ch:/user/cvs co geant3*
  - Contains both GEANT3 modified to cope with VMC and TGeant3 interface
- Running either G3 with its own geometry, either G3 with TGeo navigation
  - Default: G3 geometry but the validation with TGeo practically completed
  - -DWITHROOT added to CXXOPTS enables TGeo
    - In this case, G3 geometry not built at all
    - Materials and tracking media have to be created in G3
- The next AliRoot data challenge will use G3+TGeo

TGeo and GEANT3

- All geometry-building calls redirected to TGeo
- Navigation queries posted by G3 (“Where am I?”, “How far from next boundary?”, …) intercepted and answered by TGeo
- Several fixes/improvements with respect to the first release
  - Communication between G3 and TGeo optimized
  - G3 “sees” a ONLY geometry (non overlapping).
- Validation of TGeo option is done by direct comparisons with G3 native
  - Same initial condition, same geometry setup, same output format

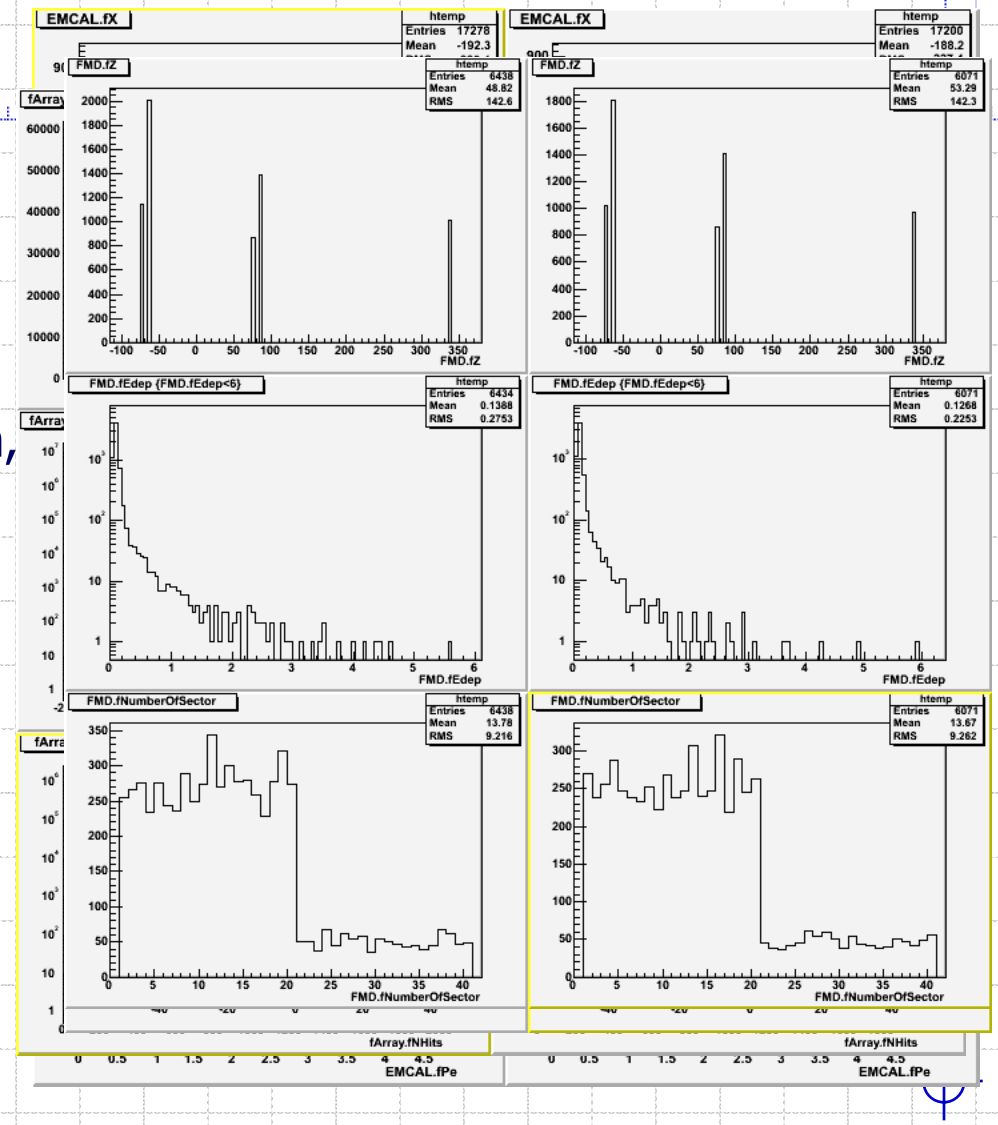
Validation tests of G3+TGeo

- Running same initial configuration macro.
  - *gAlice->Init("Config.C")*
  - Several detector modules configurations (we have the possibility to switch them ON/OFF).
  - Same particles put on the stack.
  - Same initial random seed – however this is not significant since it changes due to geometry differences during tracking (e.g. step-by-step comparisons not available except for extremely simple geometries)
- Comparisons at hit level.
  - Number of hits per module, simple distributions for hit parameters
- Performance comparisons – quite important issue.

Comparisons

- Per detector module in full ALICE setup, event-by-event
- Reasonable statistics (1000-10000) primaries
- Simple distributions: position, energy deposition, momentum, time of flight,…
- 2D plots

- Most plots matching
- Some differences observed
  - Bugs – solved
  - Statistics, low acceptance
  - Under investigation – almost none remaining



TGeo vs. G3 performance

- Quite simple to test within TGeant3: *gROOT->Time()*
- We already had some idea on performance against GEANT3 for several geometries
  - Geometries imported from G3 .rz format
  - "Physical" points collected during G3 stepping within simulation
  - Point injected both in G3 and TGeo to compare timing for pure geometrical tasks
  - G3 slightly faster for "toy" geometries (e.g. G3 examples) – hard to beat FORTRAN here
  - Gain ranging up to **x20** for real detector geometries compared to native G3

Performance of TGeo-G3 in ALICE

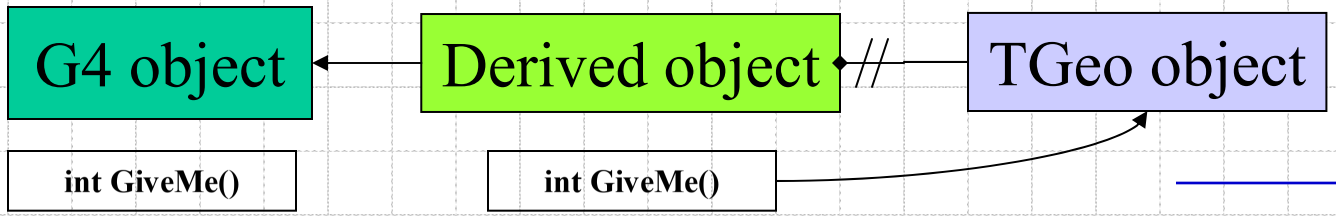
- gAlice->Init()
  - Geometry creation + internal initialization stuff
  - Full ALICE geometry: G3-120 sec. TGeo-40 sec. (x3 gain)
- gAlice-RunMC()
  - Shooting and transporting N primaries (1000 for this test) in the selected setup (full ALICE) using some generator (HIJING parameterization in this case) + hits creation; realistic physics settings
  - Time per fully-tracked primary: ***0.61 sec.*** G3 only, ***0.55 sec.*** G3+TGeo (latest TGeant3 version) - ~10% gain
    - Not negligible considering the simulation time for ALICE events
    - The gain related strictly to geometry is obviously bigger, since just a fraction of the total simulation time is spent in geometry calls.
    - ALICE G3 geometry description was already highly optimized

Interface to GEANT4

- Main author: I. Hrivnacova
- Available module in ROOT:
  - *cvs -d :pserver:cvs@root.cern.ch:/user/cvs co geant4_vmc*
  - Requires installation of *GEANT4 (currently v6)*
- Native GEANT4 geometry creation via g3tog4 module in G4
  - Some geometry limitations when running full ALICE setup due to incompatibilities G3-G4
  - Not a real limitation for running G4 via this interface in the general case
- 3 examples coming with geant4_vmc
  - Working with both G3 and G4
  - Good starting point for understanding how to use VMC
- More details on this interface can be found in Ivana's presentation at the previous ROOT workshop

GEANT4 + TGeo to-do's

- Needs an additional abstract layer in GEANT4 navigation
  - G4 team already did some work in this sense
  - We will have to provide them with more specific requirements so that such an interface can work and be efficient
  - G4 navigation interface provides pointers to its geometry objects – we need to find a way out without re-creating the full G4 geometry
- Certainly a lot of thinking to be done before starting real work
  - Thin navigation history layer with mixed objects and replaceable pointers – might be a possible approach



Interface to FLUKA

- Unlike GEANT3/4, FLUKA is more a program than a toolkit.
  - Less transparent, interfacing it is a difficult task and debugging this interface can become a real problem…
  - Specific input: ASCII file containing "cards" for any setting
    - API inexistent (or at least not documented)
- Implementation of TVirtualMC – TFluka class
- Main problems during development
  - Configuration has to be preprocessed (based on the calls to TVirtualMC) and converted to FLUKA cards
  - Not all configuration options and cuts available in the VMC can be directly mapped to FLUKA cards/ additional options in FLUKA -> core input + generated input
  - Geometry incompatible with GEANT architecture – ALICE geometry cannot be built in FLUKA native format ->Initial implementation based on FLUGG, now we have completely switched to TGeo

TFluka class

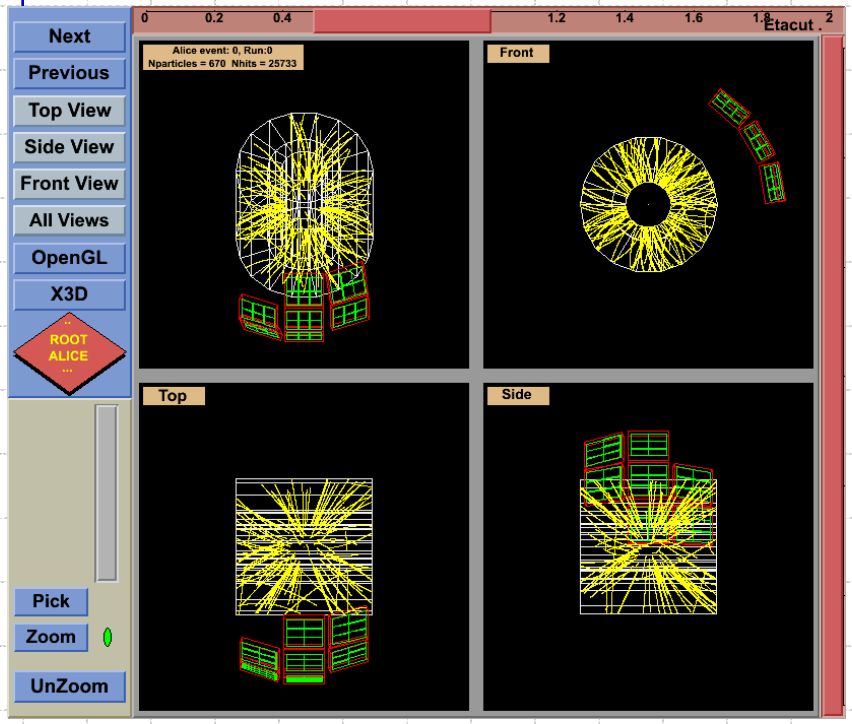
- Main subject of development/evaluation at the moment
  - Still inside AliRoot framework – to be moved to ROOT once this stage is completed
  - Running ALICE simulations with partial geometry and most physics settings switched on (see next slide)
  - 1 FTE for validation purpose only until done
- Longer development stage compared to G3 interface
  - Different navigation policy: FLUGG taken as example for navigation API but also for "not to-do's"
  - Several modifications done in TGeo to allow/optimize FLUKA-like navigation
- Few limitations still, to be addressed soon (see next slide…)

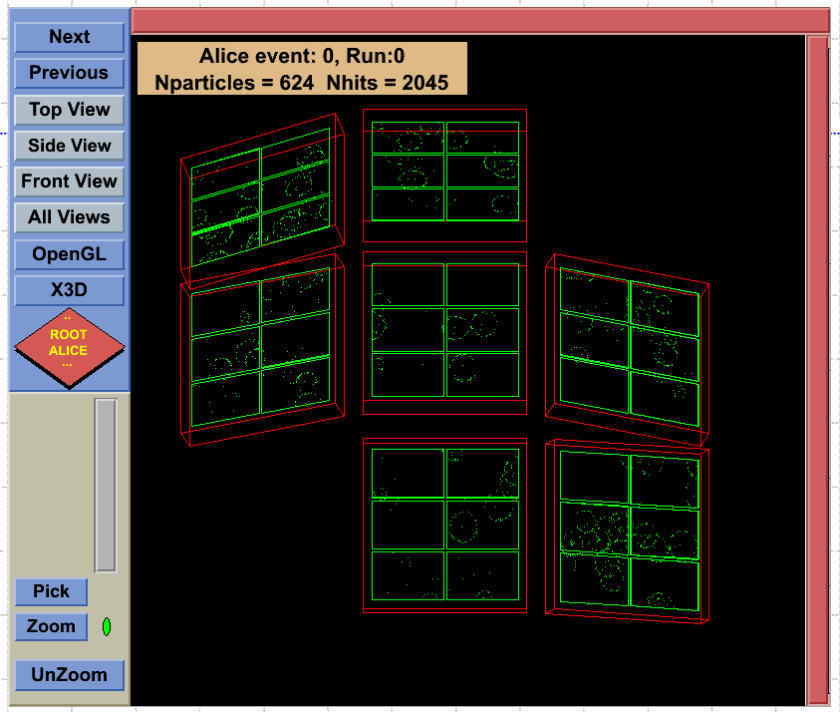
Features and limitations

- Most testing done in a limited geometry setup
  - Mostly FLUKA hadronics tested
  - Electromagnetic properties have to be registered per material – NOT automatic procedure in FLUKA (.EMF files)
    - Work done by hand for a couple of detectors – work to extend to full ALICE sizeable… we hope having this solved by FLUKA soon
  - Limitation related to navigation across replicated volumes close together (in the current version of the geometry interface) – solved by treating corresponding virtual boundaries as "special" FLUKA regions
- Implementation completely transparent – everything can be set/retrieved via VMC
  - FLUKA input generated automatically on the flight, mapping with FLUKA entities fully invisible

First tests

- First Cerenkov rings in ALICE HMPID detector





- Tracking ALICE TPC

TFluka to-do's

- This interface is not yet as mature as TGeant3/TGeant4 – some remaining problems to be fixed soon
  - Enabling features one-by-one – much easier now when running with TGeo
- FLUGG interface can be still run – switching between the 2 geometry interfaces is very easy
  - Maintenance is a pain, but we still needed it for a while as a reference for comparisons
  - To be dropped (really) soon
- Comparisons with native FLUKA for very simple setups to be done
- Comparisons with TGeant3+TGeo and native FLUKA to be done
  - Same geometry during comparison – main condition
- Currently TFluka class available only from AliRoot CVS (not yet in ROOT)

Conclusions

- The VMC framework provide mechanisms to decouple the user application from specific Monte-Carlo used
  - http://root.cern.ch/root/vmc/VirtualMC.html
- A geometrical modeller was developed within ROOT to have a single geometry engine at tracking time
  - Integrated in 2 out of 3 MC interfaces
  - Fully supporting GEANT-like geometries and providing support for alignment
  - Optimized navigation features
  - User guide available: ftp://root.cern.ch/root/doc/chapter17.pdf
- Few other experiments besides ALICE started developing/testing VMC+TGeo – based frameworks

Why not GEANT4 modeller ?

- At the time we have decided going for TGeo (2001):
  - We had problems representing ALICE geometry with G4
    - Missing features: no support for *MANY*, reflection matrices, divisions
  - All reports pointed out that G4 geometry was ~2 times slower than G3 one, and its size in memory ~4 times bigger
  - No geometry I/O, poor modularity
  - We learned from the bad experience FLUGG authors had with G4 support for doing such things
- Using directly a MC-native geometry for a Virtual MC was already a contradictory approach…
- … that would have not much helped in solving the rest of the problems, e.g. dealing better with reconstruction.