Other GEANT4 capabilities

Event biasing

Parameterisation (fast simulation)

Persistency

Parallelisation and integration in a distributed computing environment

Outline

- Event biasing
- Parameterization (fast simulation)
- Persistency
- Parallelization
- Conclusions

Biasing: introduction

☞ Analog simulation: the possible outcomes of measurements to the estimator of an observable occur with the same frequencies as they do in nature

☞ Biased simulation: important contributions to the estimator are sampled more often than the less important ones

☞ The variance reduction techniques (VRT) aim to reduce the computing time, being constant the mean value of an estimator and reducing its variance

☞ Four classes of VRT:

truncation methods (energy and time cutoff): like a Russian roulette game with zero survival probability

population control methods (geometry splitting and Russian roulette, energy splitting/roulette, weight cutoff, weight window): many samples of low weight are tracked in important regions, while few samples of high weight in unimportant regions

Modified sampling methods (exponential transform, implicit capture, forced collisions, source biasing): to sample from any arbitrary distribution rather than the physical probability as long as the particle weights are then adjusted to compensate

Partially deterministic methods (next event estimators, controlling the random number sequence): to control the normal random walk process through deterministic-like sequence

Event biasing in Geant4

☞ Event biasing technique is one of the most important requirements, which Geant4 collaboration is aware of.

This feature could be utilized by many application fields such as:

- radiation shielding
- dosimetry

Geant4 is a toolkit and all source code is open ➔ the user can implement his/her own method

- CMS, ESA, Alice and some other experiments have already their own implementations of event biasing options

☞ It's convenient for the user that Geant4 itself provides most commonly used event biasing techniques.

Partial MARS migration

- n, p, pi, K (< 5 MeV)
- Since Geant4 0.0

Primary particle biasing

- Since Geant4 3.0

Leading particle biasing

- Taking only the most energetic (or most important) secondary
- Since Geant4 3.0

Physics based biasing

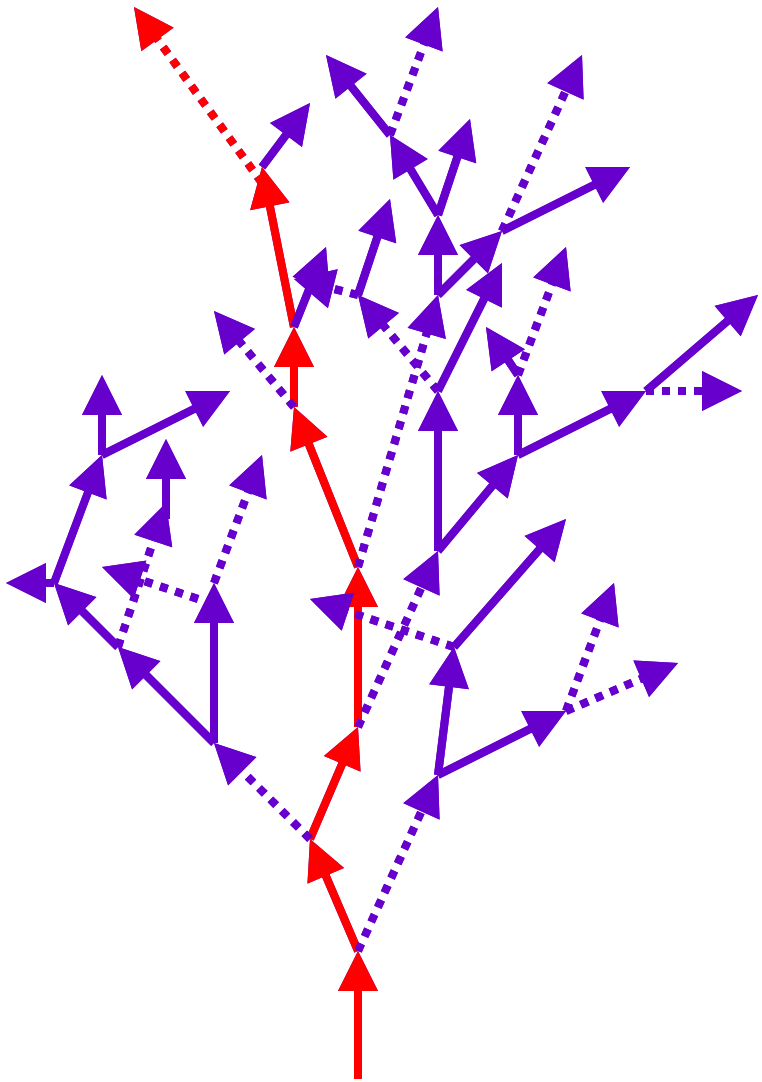
- Biasing secondary production in terms of decay products and momentum distribution
- Cross-section biasing (partial) for hadronic physics
- Since Geant4 3.0

Geometry based biasing:

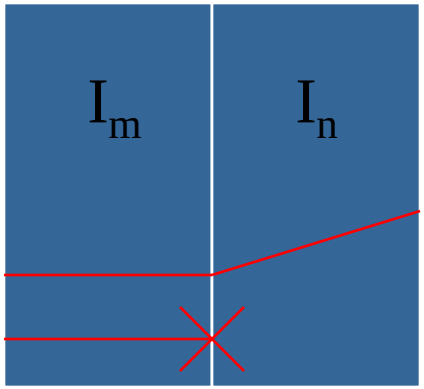
- geometric splitting and Russian roulette
- Weight roulette (or weight cutoff)
- Weight windows
- Since Geant4 5.0

Leading particle biasing

- Simulating a full shower is an expensive calculation.
- Instead of generating a full shower, trace only the most energetic secondary.
  - Other secondary particles are immediately killed before being stacked.
  - Convenient way to roughly estimate, e.g. the thickness of a shield.
  - Of course, physical quantities such as energy are not conserved for each event.



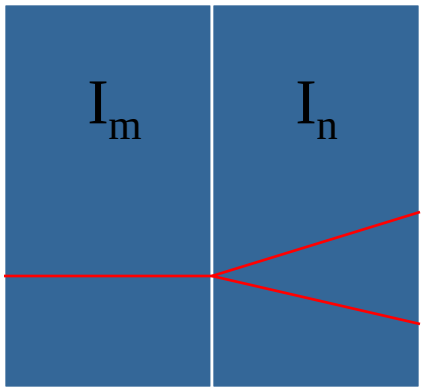
Geometrical importance biasing



Russian roulette

$r = I_n/I_m < 1$

r = survival probability



Splitting

$r = I_n/I_m > 1$

if r = 2

Geometric splitting/Russian roulette

Biasing cells are used. These can be the same volumes of the mass geometry or *ad hoc* volumes in a parallel geometry (the configuration is done in *G4MassGeometrySampler* and *G4ParallelGeometrySampler*)

- Each cell in the problem is assigned an importance *I* by the user in the G4DetectorConstruction.
- Number *I* should be proportional to the estimated value that particles have in the cell for the quantity being scored
- Splitting occurs on the boundaries between cells when a particle moves in the direction of increased importance
- Tracks crossing the cell surfaces in the direction of reduced importance are killed according to the survival probability. Necessity of the weight adjustment in order not to bias the result

Operatively, the user should activate the following classes for biasing:

G4VSampler(*G4MassGeometrySampler*, *G4ParallelGeometrySampler*): configurator of the biasing in terms of the user-defined geometry

G4GeometryCell (only simple replicas and no consideration of the hierarchical positions of physical volumes in the geometry tree)

G4VIStore for the creation of the importance store)

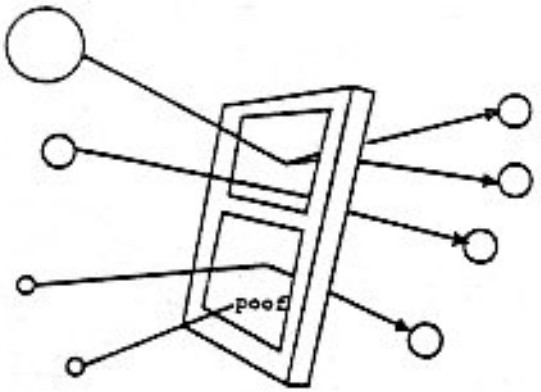
G4VImportanceAlgorithm (for customizing the importance algorithms)

Weight roulette or weight cutoff

- Russian roulette is played if a particle's weight drops below a user-specified weight cutoff.
- The source cell has an importance I. R_j is the ratio between I and I_j (j is the new cell). WC1 and WC2 are two weight cutoff values.
- The weight cutoff is applied when the particle's weight falls below R_j*WC2. With probability W/WC1*R_j the particle survives with new weight WC1*R_j, otherwise the particle is killed
- This technique is pretty useful in combination with implicit capture and geometry splitting. Weight cutoff is dependent on the importance ratio.

Weight Window (WW)

- It is a space-energy-dependent splitting and Russian roulette technique
- W_L is the lower weight bound
  $W_U = W_L * C_L$ is the upper weight bound (multiple of W_L)
  $W_S = W_L * C_S$, the survival weight for particles playing roulette



- Particles are split if their weight $W > W_U$
- Particles play Russian roulette if $W < W_L$
- Particles survive with a weight $W = W_s$
- WW is particularly useful in combination with other VRTs, which cause large weight fluctuations (such as the exponential transform)

Geometrical biasing (scoring)

➢ **SCORING**: *G4VScorer* (for the definition of the information to be scored). A default implementation is provided through *G4Scorer*, which provides scores based on the following quantities:

➔ D = step length between previous and post step point
➔ WD = weight of the particle at the previous step point times the step length
➔ WDT = WD divided by the velocity of the particle at the previous step point
➔ WDE = weight times energy (both from the previous step point) times the step length
➔ WTE = WDE divided by the velocity

Improving B01

☞ Changing the geometry:

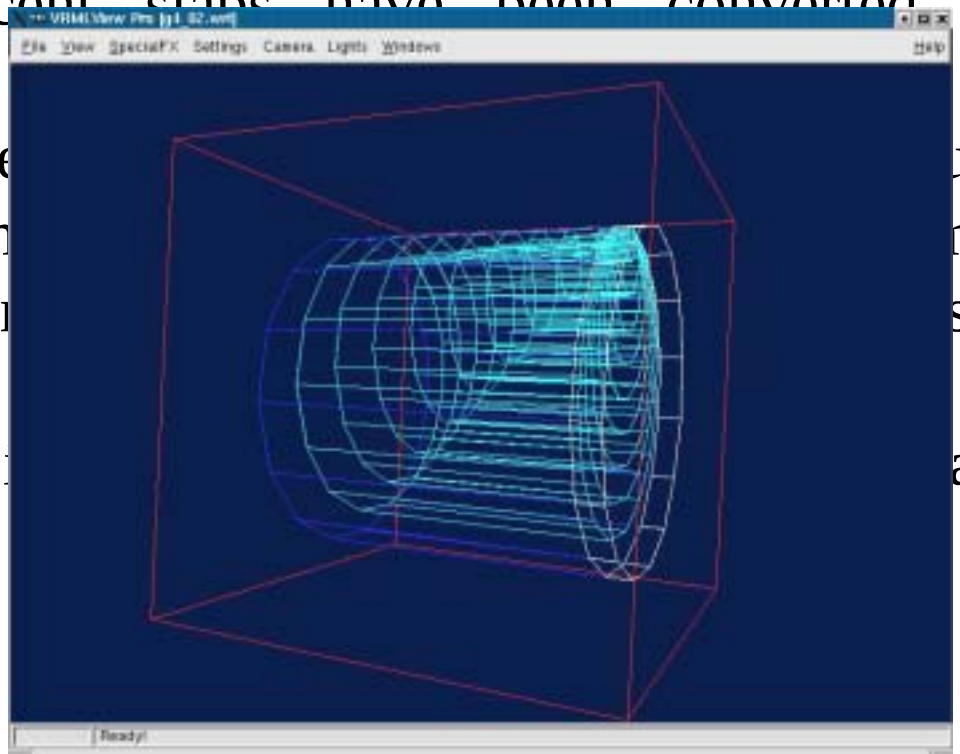
Original configuration: neutrons impinging on a thick concrete shield (18 slabs) – Tiara testbeam (examples/advanced).

➢ Obtaining results with another configuration.

➢ 10 adjacent slabs have been converted into 18 nested cylinders.

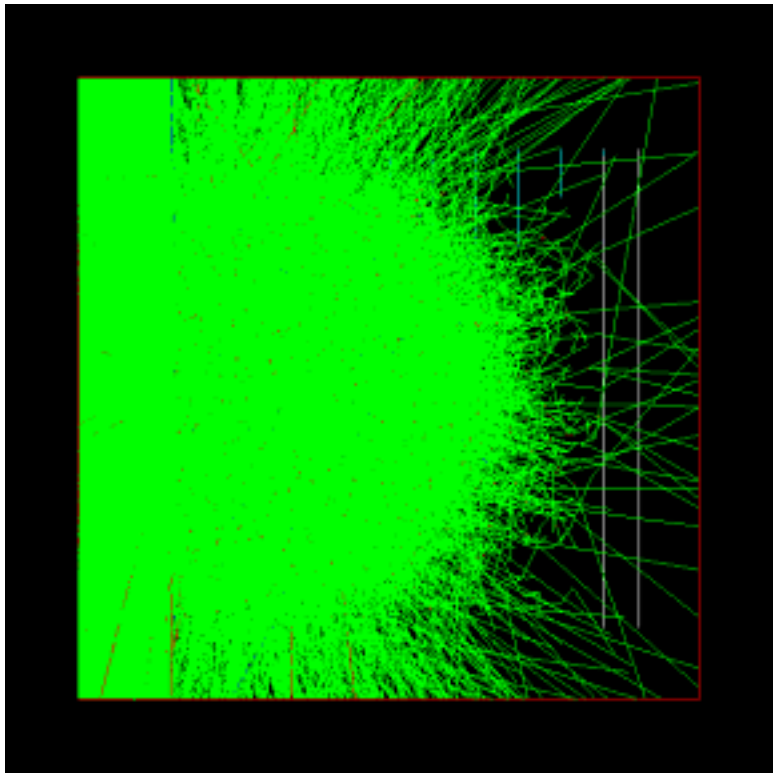
➢ Biasing te ulette with Mass Sampler. A n nands to take the scoring infor sed and unbiased problem.

➢ Geometry and VRML

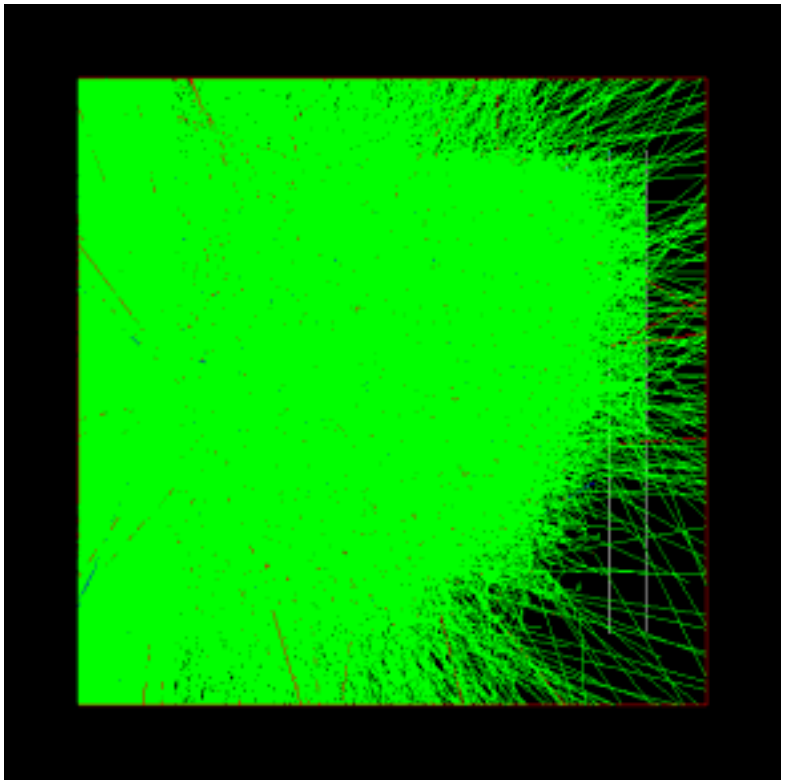


Testing with MassGeometrySampler

- Using the B01 modified, two runs with 1000000 primary particles (one with and the other without biasing)



Without biasing



With biasing

Testing with MassGeometrySampler

- Using the scoring table in the unbiased case

Cell name	Importance	Tracks entering	Population	Collisions	Coll*WGT	NumWGTedE	FluxWGTedE	Av.Tr.WGT
1	1	1042221	1368317	2047117	2047117	1.40629E-05	9.11009	1
2	1	405864	533549	900152	900152	1.13616E-05	8.52042	1
3	1	138784	181115	331956	331956	9.01605E-06	8.03625	1
4	1	43795	56200	110038	110038	1.07014E-05	7.63709	1
5	1	11489	14787	30225	30225	1.30999E-05	7.34783	1
6	1	1876	2422	5448	5448	0.00010015	6.71384	1
7	1	255	332	743	743	3.56539	6.14856	1
8	1	28	33	72	72	2.34861	5.21457	1
9	1	**1**	1	1	1	9.0678	9.0678	1
10	1	---	---	---	---	---	---	---
Rest	1	9	9	0	0	4.58625	5.97013	1
World	1	85216	1080615	0	0	10	10	1

Testing with MassGeometrySampler

- Using the scoring table in the biased case

Cell name	Importance	Tracks entering	Population	Collisions	Coll*WGT	NumWGTedE	FluxWGTedE	Av.Tr.WGT
1	1	1084374	1410448	2039367	2039367	1.58966E-05	9.11318	1
2	2	459214	1107809	1805789	902894	1.14833E-05	8.52461	1
3	4	323197	757628	1329572	332395	1.2539E-05	8.03993	1.00001
4	8	208226	473035	877449	109684	1.17692E-05	7.65557	1.00004
5	16	103657	242493	473884	29663.2	9.63787E-06	7.38201	1.00058
6	32	33839	80148	168062	5291.12	7.93479E-06	6.82078	1.00918
7	64	9091	21005	46951	737.781	1.07359E-05	6.16203	1.00595
8	128	2271	5153	11716	91.9844	5.41002E-06	5.73127	1.00551
9	256	**466**	1071	2364	9.27344	4.69943E-06	5.15814	1.00709
10	512	**52**	114	244	0.476562	2.46977	4.40858	1
Rest	512	321	4707	0	0	2.86472	5.34699	1.01301
World	1	89374	1085249	0	0	10	10	1

Plans of event biasing in Geant4

- Full interface to MARS
  - For fully biased mode

- Complete cross-section biasing fro physics processes

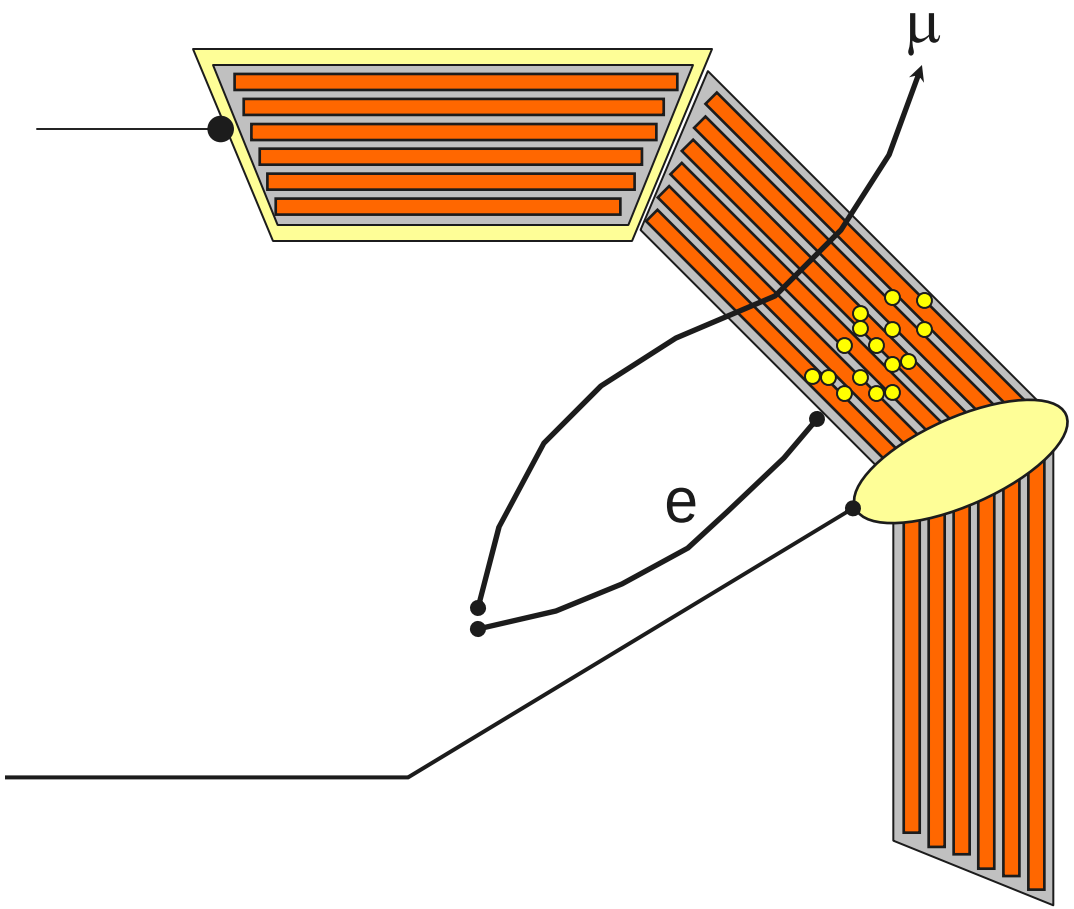
- Other scoring options rather than surface flux counting, which is currently supported, are under study
  - Tallies (doses and fluences in a volume)

Fast simulation - Generalities

- Fast Simulation, also called as shower parameterization, is a shortcut to the "ordinary" tracking.
- Fast Simulation allows you to take over the tracking and implement your own "fast" physics and detector response.
- The classical use case of fast simulation is the shower parameterization where the typical several thousand steps per GeV computed by the tracking are replaced by a few ten of energy deposits per GeV.
- Parameterizations are generally experiment dependent. Geant4 provides a convenient framework.

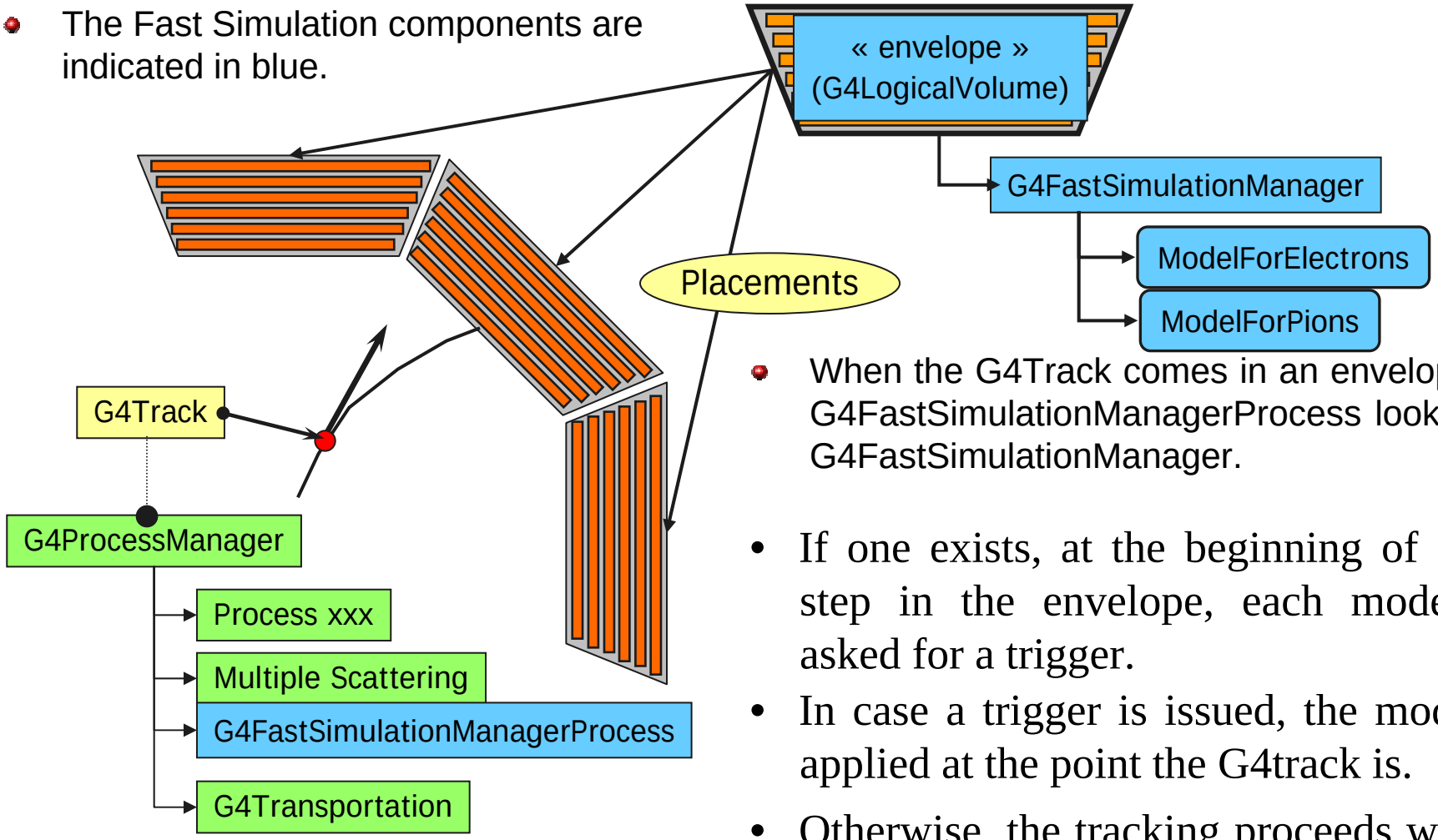
Parameterization features

- Parameterizations take place in an *envelope.* This is typically a mother volume of a sub-system or of a major module of such a sub-system.
- Parameterizations are often dependent and/or may be applied to only some kinds of particles.
- They are often not applied in complicated regions.



Fast Simulation

- The Fast Simulation components are indicated in blue.



- When the G4Track comes in an envelope, the G4FastSimulationManagerProcess looks for a G4FastSimulationManager.

- If one exists, at the beginning of each step in the envelope, each model is asked for a trigger.
- In case a trigger is issued, the model is applied at the point the G4track is.
- Otherwise, the tracking proceeds with a normal tracking.

G4FastSimulationManagerProcess

- The G4FastSimulationManagerProcess is a process providing the interface between the tracking and the fast simulation.
- It has to be set to the particles to be parameterized:
  - The process ordering must be the following:

    [n-3] ...

    [n-2] Multiple Scattering

    [n-1] G4FastSimulationManagerProcess

    [n] G4Transportation
  - It can be set as a discrete process or it must be set as a continuous & discrete process if using ghost volumes.

Ghost Volume

- Ghost volumes allow to define envelopes independent to the volumes of the tracking geometry.
  - For example, this allows to group together electromagnetic and hadronic calorimeters for hadron parameterization or to define envelopes for geometries imported from a CAD system which does not have a hierarchical structure.
- In addition, Ghost volumes can be sensitive to particle type, allowing to define envelopes individually to particle types.
- Ghost Volume of a given particle type is placed as a clone of the world volume for tracking.
  - This is done automatically by G4GlobalFastSimulationManager.
- The G4FastSimulationManagerProcess provides the additional navigation inside a ghost geometry. This special navigation is done transparently to the user.

Persistency

- Geant4 does not rely on any particular persistency solution.
  - User should provide his/her own solution
    - Exception : Cross-section tables
  - Geant4 provides various examples
- Event input
  - Sample : G4HEPEvtInterface
- Geometry
  - XML, GDML, STEP, GGE (Geant4 Geometry Editor), etc.
- Histograms
  - AIDA, ROOT
- Primaries, hits, trajectories, digits
  - G4VPersistencyManager abstract base class
  - Convert Geant4 objects to user persistency objects
    - ASCII file, ROOT, Objectivity/DB, etc.

Parallelisation

- By design, Geant4 can be executed in more than one processes/machines in parallel.
- Geant4 itself does not provide any mechanism of parallelisation but with some external utilities.
  - "Event parallelism"
    - Master process distributes events to slave processes.
    - Geometry, physics processes, user classes, parameters are sent to slave processes before start processing events.
    - Event output and histograms are sent back to the master process to be collected.
- Geant4 provides one example which requires TOP-C.
  - examples/extended/parallel
  - TOP-C : developed by G.Cooperman (Northeastern U.)

- Other possibilities of parallelisation and access to distributed computing resources
  - E.g. via DIANE