



Geant4 10.2.p01
<http://www.geant4.org>

Event Biasing

20-23 June, 2016
Geant4 tutorial @ MIT
Whitaker Building – Room 56-154

Slides from Marc Verderi (LLR, Ecole polytechnique)




Overview

- › Introduction
- › Existing Biasing Options before v10.0
 - Primary Particle Biasing
 - Options In Hadronic
 - Geometry based importance biasing
 - Weight Window Technique
 - Reverse Monte Carlo
 - User defined biasing
- › New approach, since v10.0



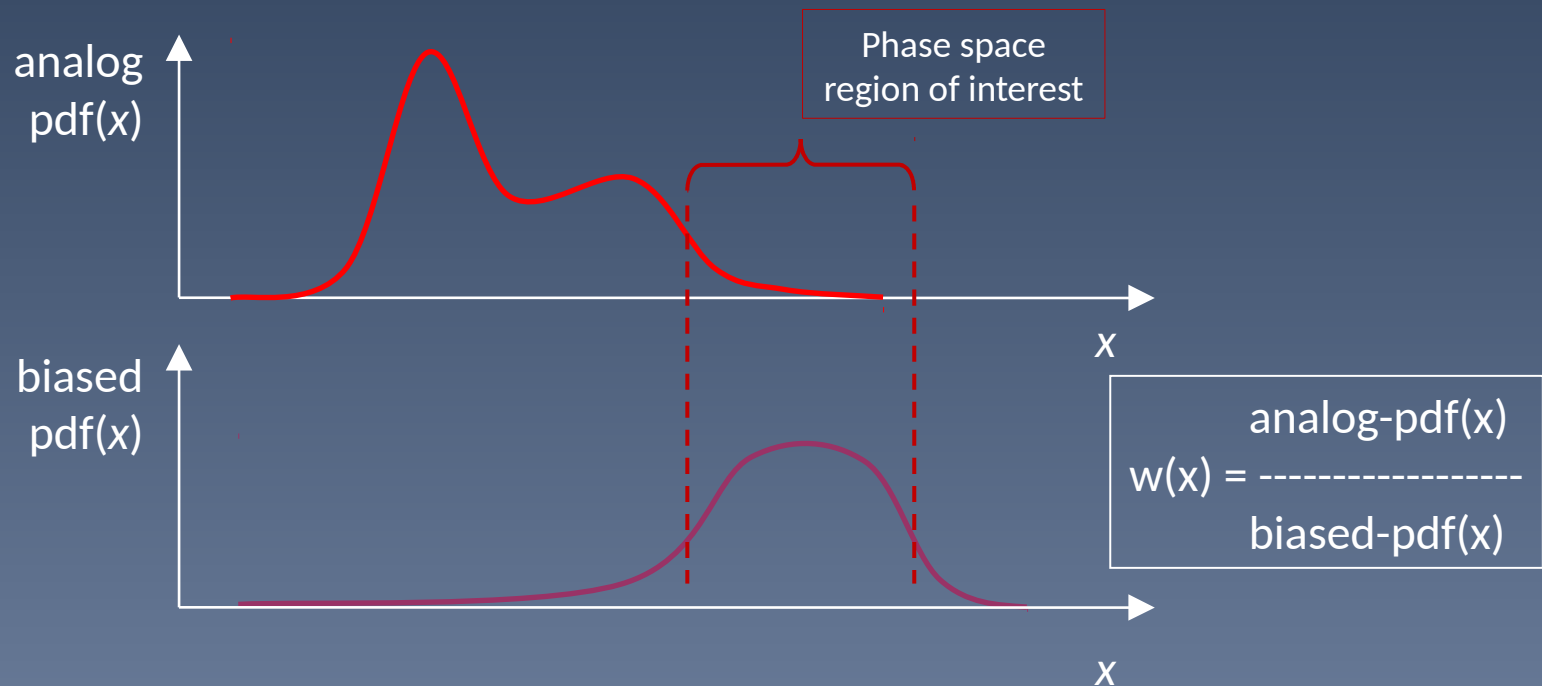
Introduction

What is “event biasing” ?

- Event biasing (variance reduction) is for simulating rare events efficiently
 - It is an accelerated simulation with respect to these events
 - ~~And to these events only : ie not everywhere~~
 - It “focuses” on what we want to tally
 - It can provide LARGE CPU improvements
 - Several orders of magnitude ! Depending on the problem.
- “Biasing” stands for using a “biased simulation”:
 - Simulation is “biased” because the region of the phase space that contributes to the tally is enhanced compared to the analog simulation.
 - This enhancement comes together with the computation of statistical “weights” that are used to go back to analog quantities, ie to “de-bias”
 - Eg : 
- There is a large variety of biasing technique implementations, with many different names, very “jargonified”, but two methods make the “backbone” of most of them:
 - Importance Sampling
 - Splitting

Importance Sampling

- › In an « importance sampling » technique, the analog pdfs (probability density function) are replaced by biased ones:

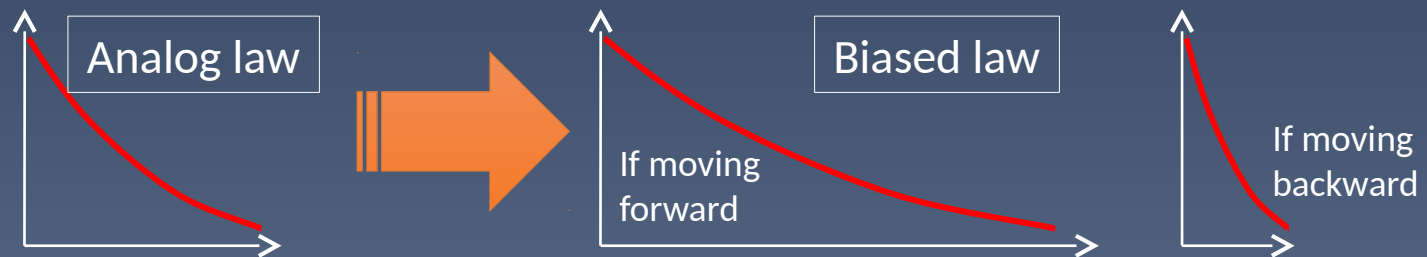


- › The weight, for a given value x , is then the ratio of the analog over the biased distribution values at x .

Examples of importance sampling techniques

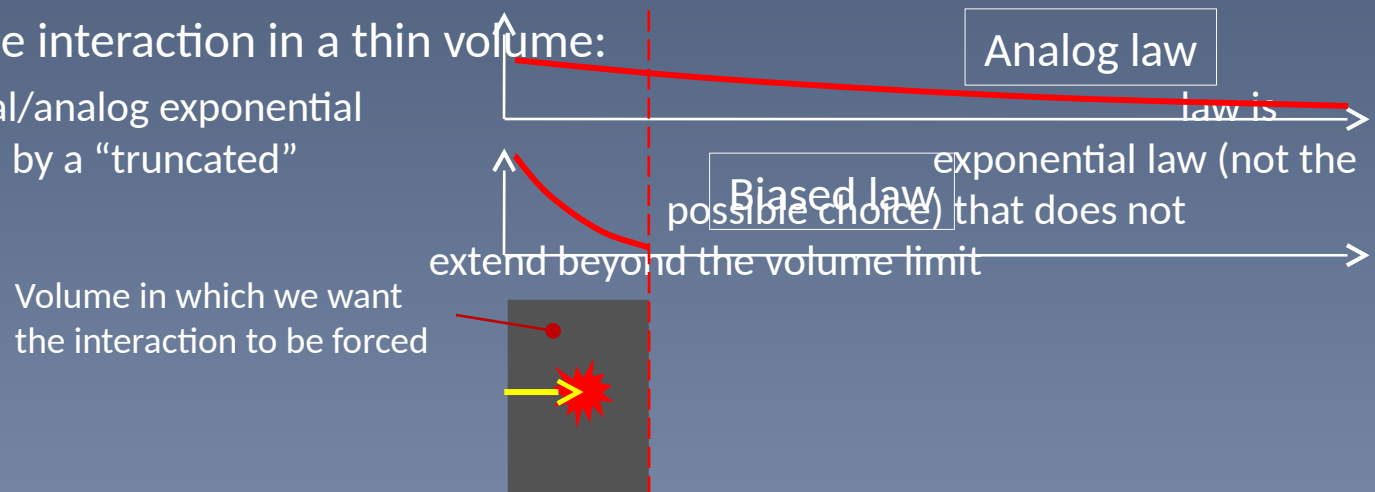
> The “exponential transform” technique

- It is a change of the total cross-section
- Often made direction dependent
- The usual/analog exponential law is replaced by an other law (still an exponential one here, but with different cross-section)



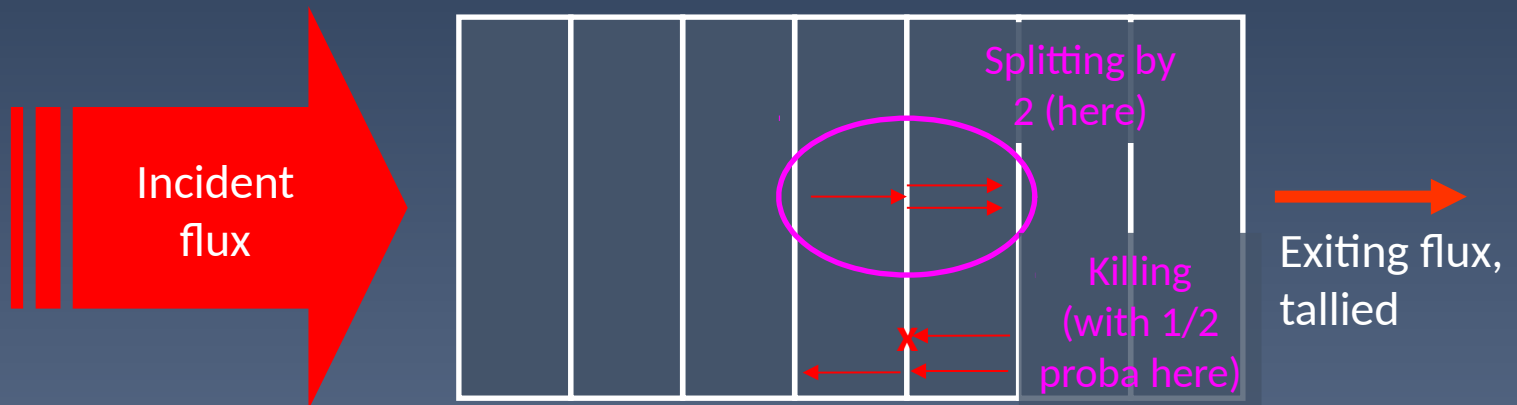
> Forcing the interaction in a thin volume:

- The usual/analog exponential replaced by a “truncated” only



Splitting

- › In splitting, the physical distributions are unchanged, but a “splitting” occurs at some places when particles go towards the tally is to be made



- In above example, particles moving toward exit are cloned
 - › cloning is for compensating the physical absorption in the shield material
- When cloning happens, clones receive a weight $\frac{1}{2}$ (here) of the initial track
- › Splitting comes together with « killing » / « Russian Roulette »
 - For particles going opposite to what we want
 - In above example, particles moving backward are killed with a (here) probability $\frac{1}{2}$ and, if it survives, its weight is multiplied by 2 (here).



Existing Biasing Options before v10.0



Primary Particle Biasing

- › The General Particle Source (GPS) allows to bias the particle source:
 - Position
 - Angular Distribution
 - Energy Distribution
- › This is an “importance sampling” biasing
 - The physical distributions are replaced by biased ones
- › Primary particles are given a weight that is the ratio of analog / biased probabilities
 - In the simulation, all daughter, gran-daughter,... particles from the primary inherit the weight of this primary

Options In Hadronic (1)

> Cross-section Biasing:

- Possibility to enhance a (low) cross-section
 - > This an “importance sampling” approach
- Available for photon inelastic, electron & positron nuclear processes

```
//init reaction model
G4ElectroNuclearReaction* theElectroReaction = new G4ElectroNuclearReaction();

//create process and register model
G4ElectronNuclearProcess theElectronNuclearProcess;
theElectronNuclearProcess.RegisterMe(theElectroReaction);

//apply biasing
theElectronNuclearProcess.BiasCrossSectionByFactor(100);

//register process
pManager->AddDiscreteProcess(&theElectronNuclearProcess);
```

- (no messenger for this ?)

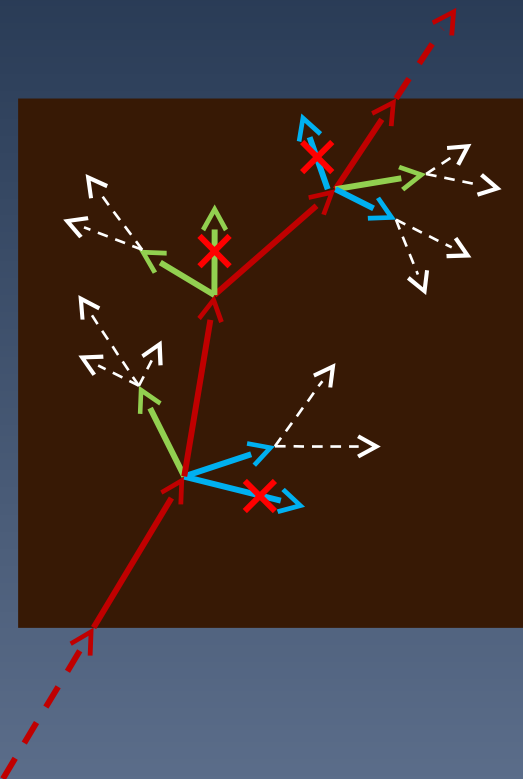
Options In Hadronic (2)

> Leading Particle Biasing:

- It is a technique used for estimating penetration of particles in a shield
- Instead of simulating the full shower, interaction, only keep:
 - > The most energetic particle
 - > + randomly one particle of each species
- This is a « killing » - based biasing

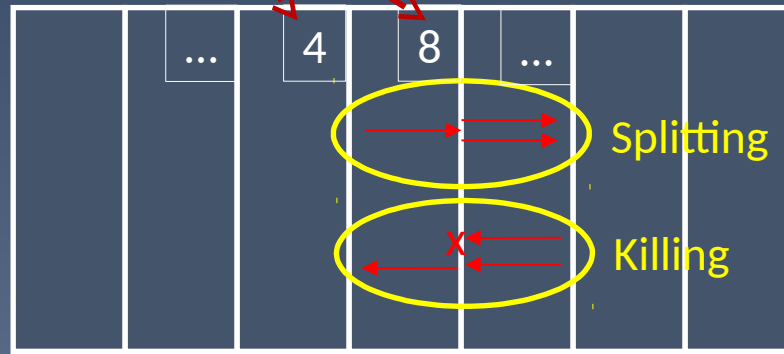
> Radioactive decay

- Enhance particular decay within a given time window
- Sample all decay modes with equal probabilities
- Split parent nuclide in a user defined number of copies, letting the decay go
- Enhance emission in a particular direction



Geometry based importance biasing

- > A direct application of what was presented earlier here
- > Attach “importance” to cells in geometry
 - Not to be confused with “importance sampling” : ie change of probability density functions of interaction laws



- Applies splitting if the track moves forward
 - > with splitting factor $8/4 = 2$, if track goes from « 4 » to « 8 » (eg, here)
 - > each copy having a weight = $\frac{1}{2}$ of the incoming track
- Applies killing if the track moves the other way
 - > it is killed with a probability $\frac{1}{2}$
 - > If particle survives, its weight is multiplied by 2 (eg, here)

Geometry based importance biasing

- › Geometry cells, meant to carry importance values, are created and associated to physical volumes in the detector construction:

```
G4IStore *istore = G4IStore::GetInstance();
```

Replica number, if any 

```
// Loop on physical volumes:
```

```
istore->AddImportanceGeometryCell(importanceValue, physicalVolumePointer [, nReplica]);
```

- All volumes should be given an importance value.

- › Biasing may be applied to some particle type only: eg neutron. In the main program, this is specified as:

```
G4GeometrySampler geometrySample(worldVolume,"neutron");
```

- (worldVolume specified as there is the ability to define cells in a parallel geometry: not discussed here)

- › The actual splitting and killing are handled by dedicated processes. These are to be inserted in the physics list. If using a modular physics, it is done as:

```
G4VModularPhysicsList* physicsList = new FTFP_BERT;
```

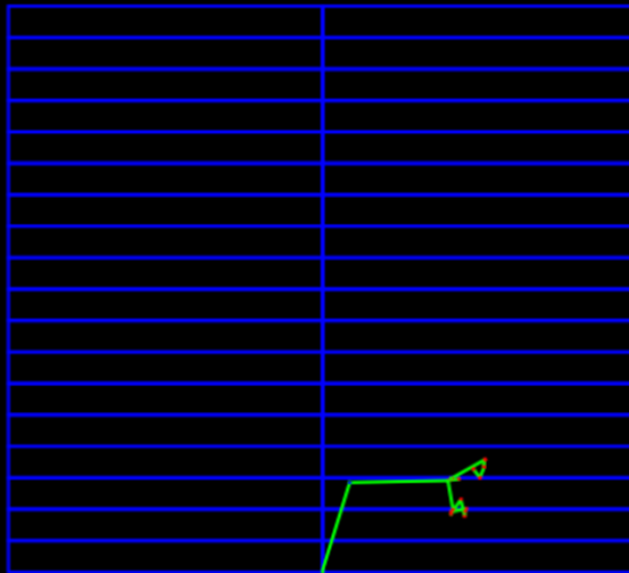
```
physicsList->RegisterPhysics(new G4ImportanceBiasing(&geometrySampler));
```

- › More details can be found in examples:

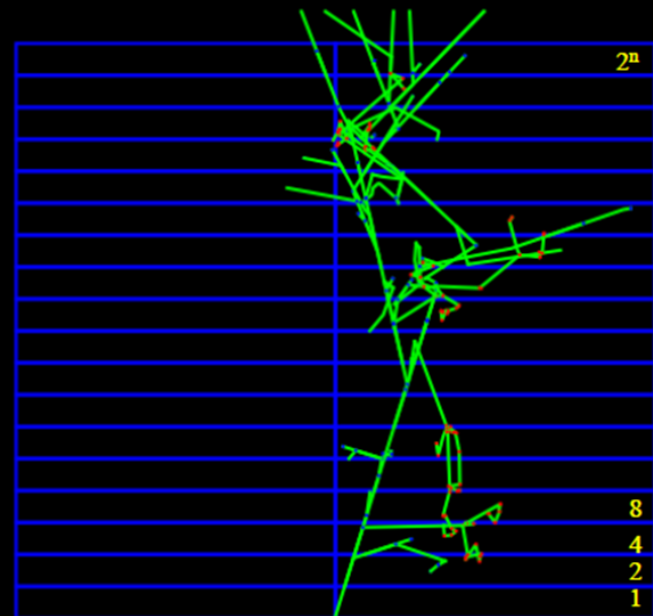
- examples/extended/biasing/B01 : geometry biasing with importance cells or weight window
- examples/extended/biasing/B02 : same, with cells in a parallel geometry
- examples/extended/biasing/B03 : same, for the case of a non modular physics list

Example B01 - 10 MeV neutrons, thick concrete cylinder

Analogue Simulation

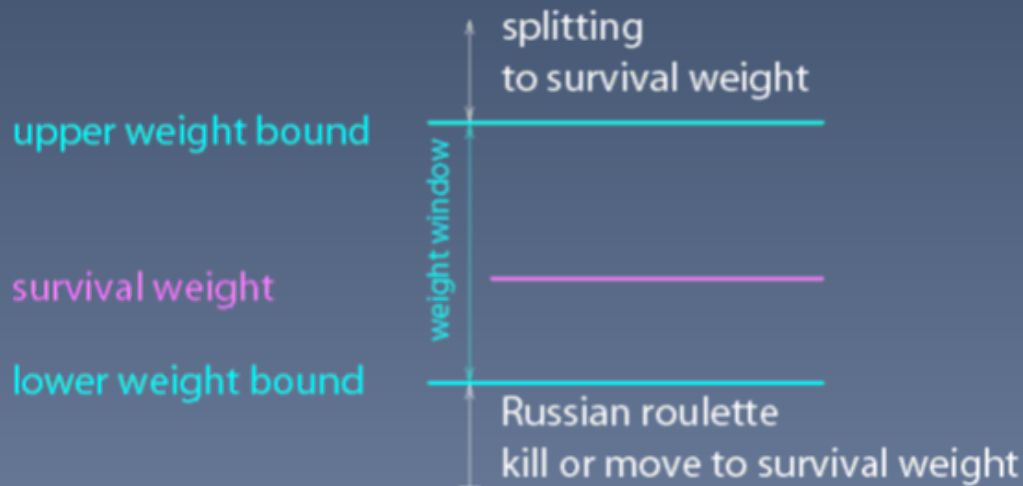


Importance Sampled



Weight Window Technique

- › This is a « splitting & killing » technique, to avoid having
 - Too high weight tracks at some point
 - › As this makes the convergence of the estimated quantities difficult
 - › Tracks with weight above some value are splitted
 - Too low weight tracks
 - › As these are essentially a waste of time
 - › Tracks below some value are “Russian roulette” - killed



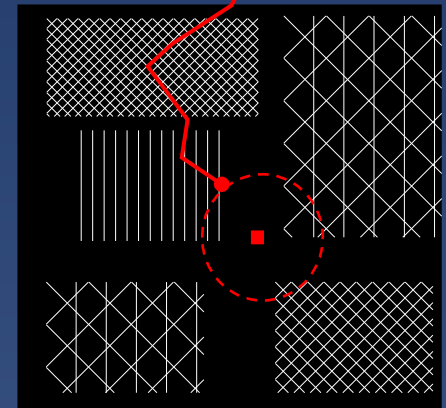
- › As with importance, this is configured per cell
 - And can be configured per energy
- › See: [geant4/examples/extended/biasing/B01](#)

Reverse Monte Carlo

- › Simulation in which particles go back in time !
- › Useful in case of small device to be simulated in a big environment
 - Requested by ESA, for dose study in electronic chips in satellites
- › Simulation starts with the red track
 - So-called “adjoint” track
 - Going back in time
 - Increasing in energy
 - Up to reaching the extended source

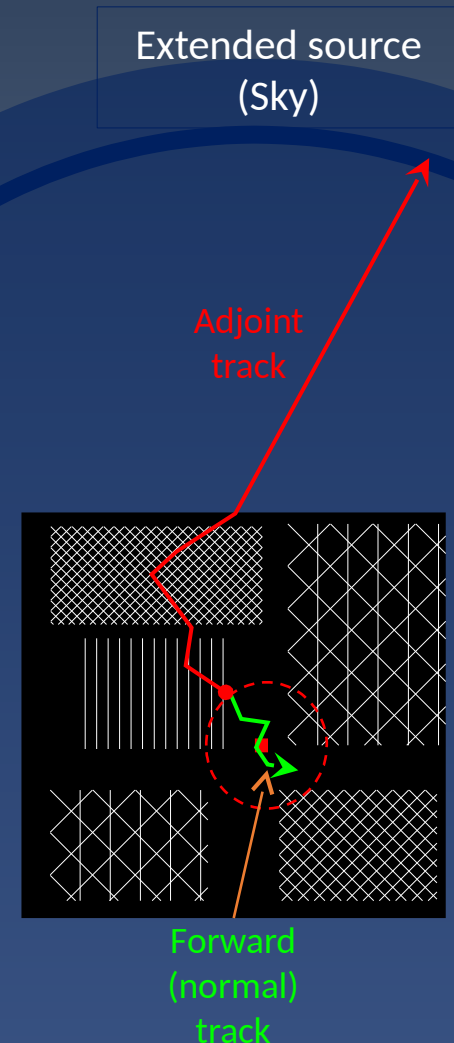
Extended source
(Sky)

Adjoint
track



Reverse Monte Carlo

- > Simulation in which particles go back in time !
- > Useful in case of small device to be simulated in a big environment
 - Requested by ESA, for dose study in electronic chips in satellites
- > Simulation starts with the red track
 - So-called “adjoint” track
 - Going back in time
 - Increasing in energy
 - Up to reaching the extended source
- > If track energy < high energy limit of the sky source spectrum
 - Proceed with a usual “forward” simulation (green track)
- > When many track simulated, adjust adjoint track weights so that energy spectrum of adjoint particle matches sky source spectrum
 - This provides the weight of the green tracks
- > Dose in chip can then be computed, accounting for the forward track contribution, with the proper weight.
- > See: [geant4/examples/extended/biasing/ReverseMC01](#)





User defined biasing

- › G4WrapperProcess can be used to implement user defined event biasing
- › G4WrapperProcess, which is a process itself, wraps an existing process
- › All function calls forwarded to wrapped process
- › Needed steps:
 1. Create derived class inheriting from G4WrapperProcess
 2. Override only the methods to be modified, e.g., PostStepDoIt()
 3. Register this class in place of the original
 4. Finally, register the original (wrapped) process with user class

Example with brem. splitting

- Bremstrahlung splitting is a technique used for example in medical applications:
 - Radio-therapy with photon beam generated by bremstrahlung
 - Interest is in generated photons, not in the electron → enhance photon production
- The bremstrahlung process is repeated N times, with the same initial electron
 - All bremstrahlung photons are given a weight $1/N$
 - The electron continues with one of the state among the N ones generated

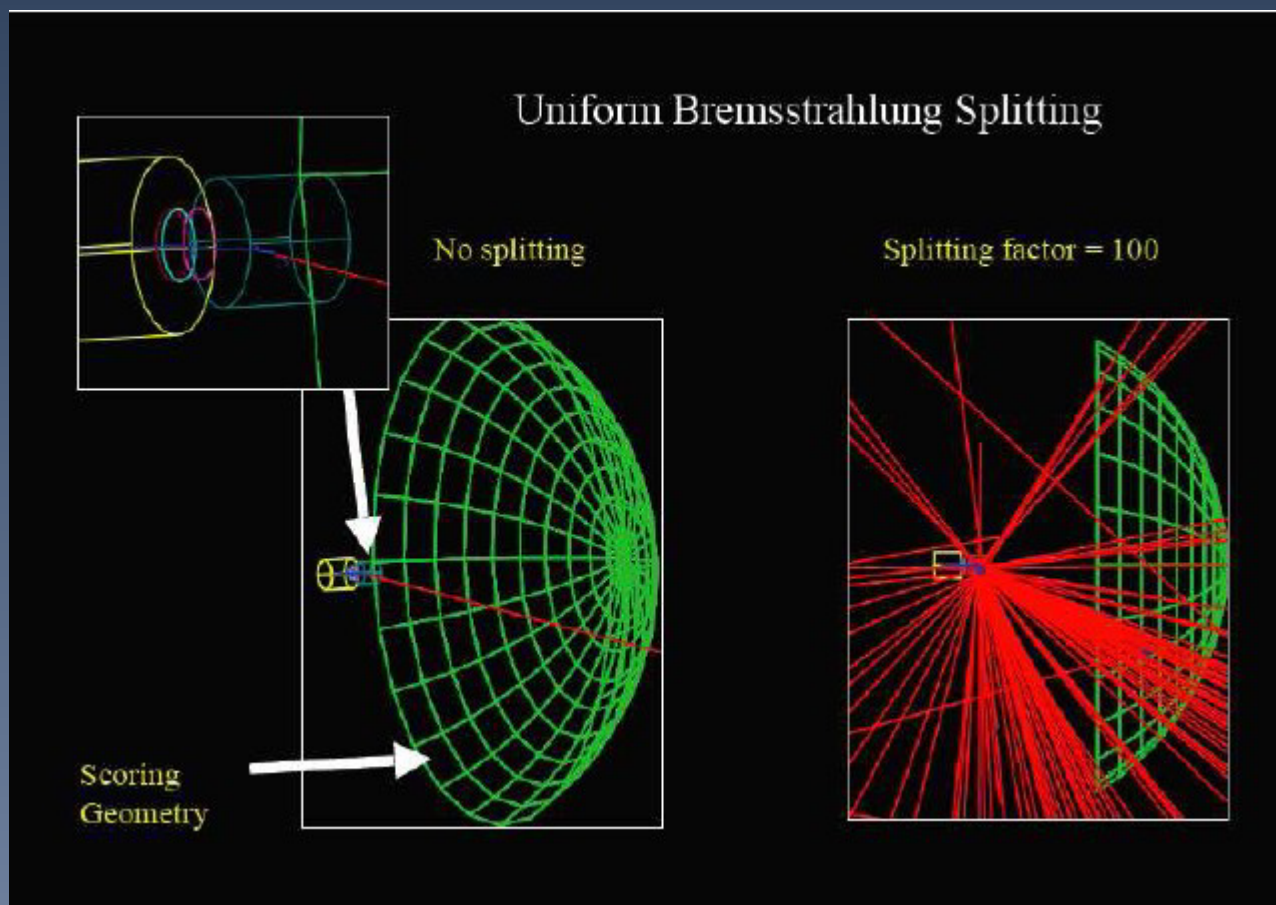
```
G4VParticleChange* myWrappedProc::PostStepDoIt(const G4Track& track, const G4Step& step)
{
    G4double weight = track.GetWeight()/fNSplit;

    std::vector<G4Track*> secondaries(fNSplit);

    // Secondary store
    // Loop over wrapped PSDI method to generate multiple secondaries
    for (G4int i=0; i<fNSplit; i++)
    {
        particleChange = pRegProcess->PostStepDoIt(track, step);
        assert (0 != particleChange);
        particleChange->SetVerboseLevel(0);
        // Save the secondaries generated on this cycle
        for (G4int j=0; j<particleChange->GetNumberOfSecondaries(); j++)
        {
            secondaries.push_back (new
                                   G4Track(*(particleChange->GetSecondary(j))));
        }
    }
    // -- and then pass these secondaries to the particle change...
}
```

← Brem. Process in this case

Example with a brem. splitting



> Note: Since 9.6, a built-in brem. splitting option is available.



**New approach,
since v10.0**

Introduction

- Many options already exist. But they
 - do not allow to change the probability distribution of processes
 - do not make “easy” to change the final state behavior of a process
 - do not make “easy” to mix biasing options
- Current development aims/hopes at improving this
 - to allow options like

- Exponential transform: $p(\ell) = \sigma \cdot e^{-\sigma\ell} \rightarrow p'(\ell) = \sigma' \cdot e^{-\sigma'\ell}$

- Change total cross-section
 - Make change direction dependent

- forced interaction:

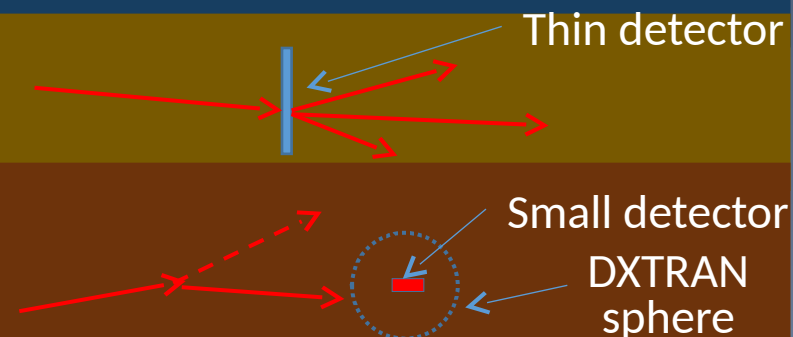
- Force interaction in thin volume

- forced flight (towards detector)

- So called DXTRAN
 - Force scattering towards detector

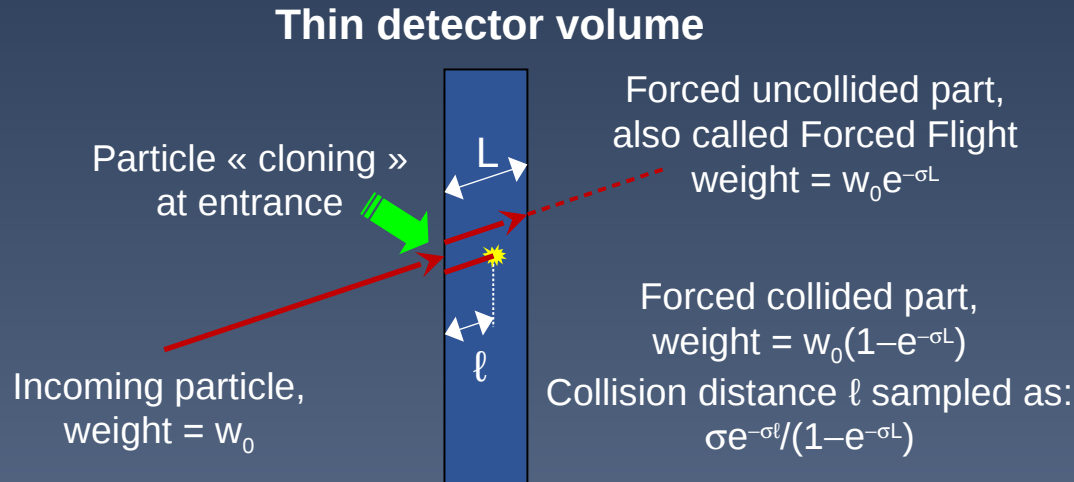
- etc.

- and to allow complex logic to be implemented by providing “building blocks” rather than “built-in” functionalities



Building blocks versus built-in

- › Example of “force collision” scheme à la MCNP



- › Approach considered:

- Instead of providing the whole scheme in one go
- Consider “atomic” “biasing operations” (G4VBiasingOperation)
 - › splitting, forced free flight, forced interaction, etc.
- Compose these with a “biasing operator” (G4VBiasingOperator)
 - › that selects the operations and builds the needed sequence
- This operator sends the operations to a dedicated process
 - › that controls the physics processes (G4BiasingProcessInterface)

Physics process behavior in Geant4 : where to bias ?

> Illustrate with point-like processes:

G4PhotoelectricEffect

75 cm

G4GammaConversion

5 cm

G4ComptonScattering

9 cm

1. `GetPostStepPhysicalInteractionLength()`
 - Returns the distance at which the process will make an interaction
 - Analog exponential law is at play
 - Above analog behavior superseded by biased behavior (if wished)
2. `PostStepDoIt()`
 - Called if the process has responded the shortest of the interaction distances
 - Generate final state, according to specific process analog physical law
 - Above analog behavior superseded by biased behavior (if wished)

> `G4BiasingProcessInterface` wraps the physics process. It is meant to intercept calls for distance of interaction and for final state production

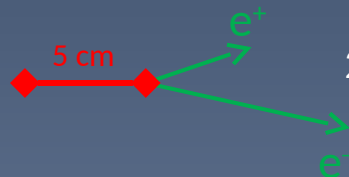
Physics process behavior in Geant4 : where to bias ?

> Illustrate with point-like processes:

G4PhotoelectricEffect



G4GammaConversion



G4ComptonScattering

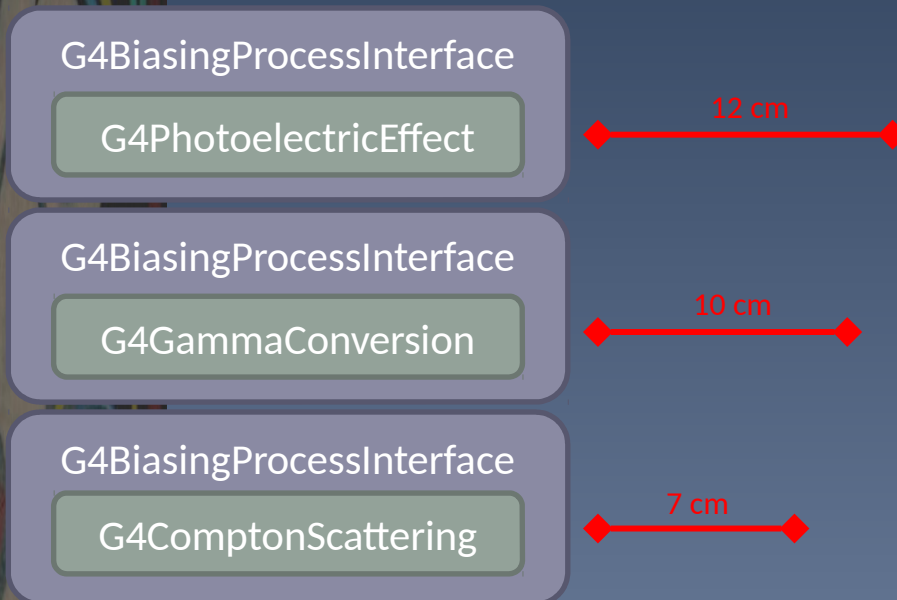


1. `GetPostStepPhysicalInteractionLength()`
 - Returns the distance at which the process will make an interaction
 - Analog exponential law is at play
 - Above analog behavior superseded by biased behavior (if wished)
2. `PostStepDoIt()`
 - Called if the process has responded the shortest of the interaction distances
 - Generate final state, according to specific process analog physical law
 - Above analog behavior superseded by biased behavior (if wished)

> `G4BiasingProcessInterface` wraps the physics process. It is meant to intercept calls for distance of interaction and final state production.

Physics process behavior in Geant4 : where to bias ?

> Illustrate with point-like processes:

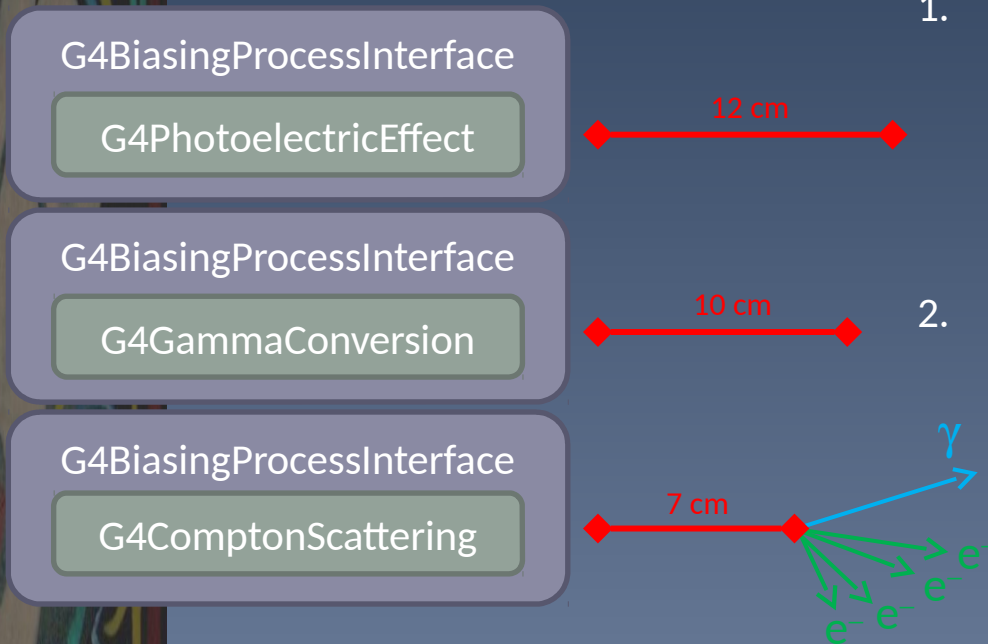


1. `GetPostStepPhysicalInteractionLength()`
 - Returns the distance at which the process will make an interaction
 - ~~Analog exponential law is at play~~
 - Above analog behavior superseded by biased behavior (if wished)
2. `PostStepDoIt()`
 - Called if the process has responded the shortest of the interaction distances
 - Generate final state, according to specific process analog physical law
 - ~~Above analog behavior superseded by biased behavior (if wished)~~

> `G4BiasingProcessInterface` wraps the physics process. It is meant to intercept calls for distance of interaction and for final state production.

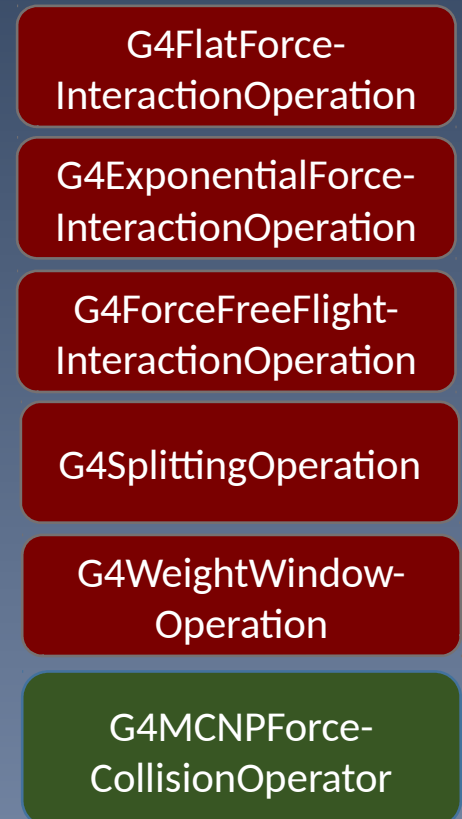
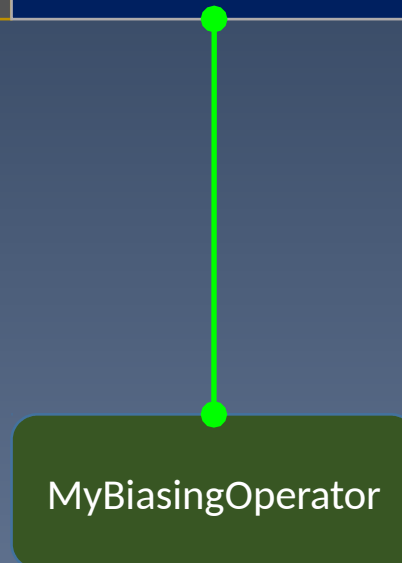
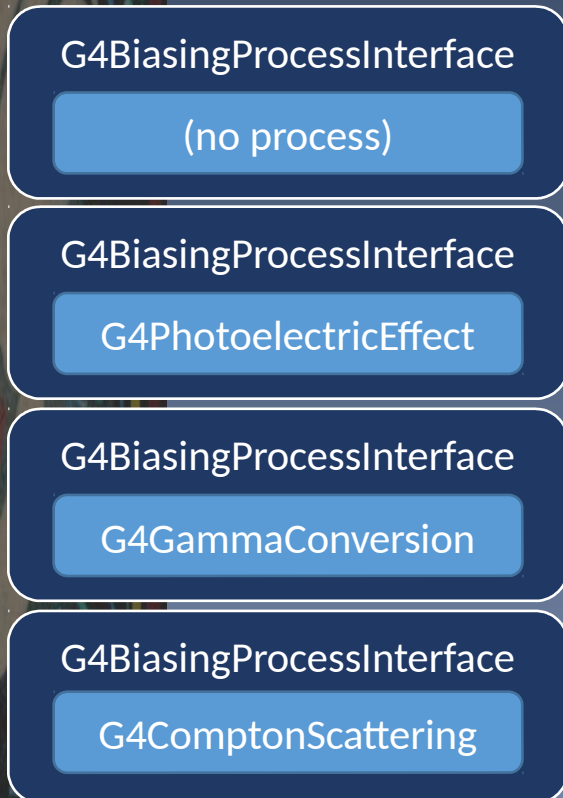
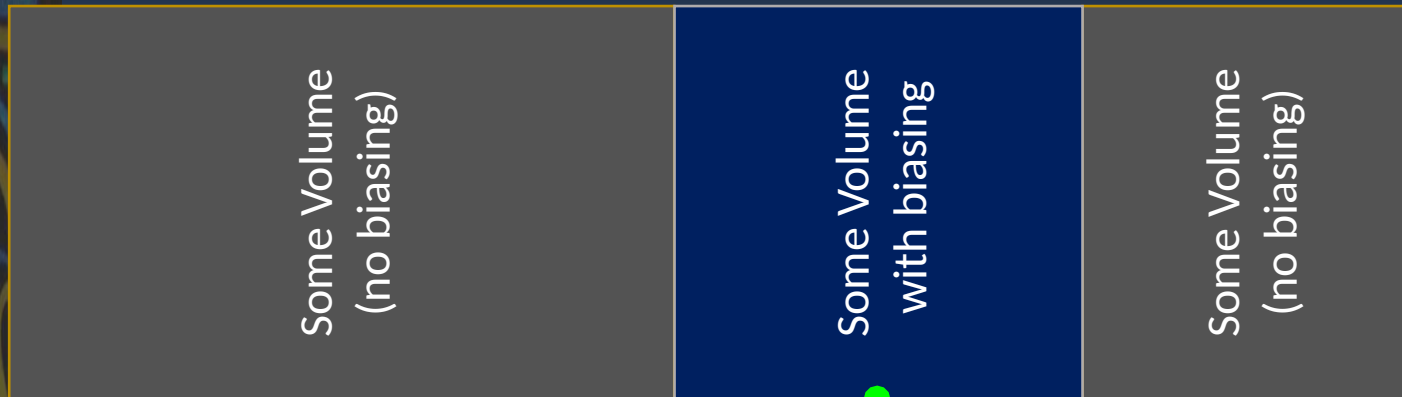
Physics process behavior in Geant4 : where to bias ?

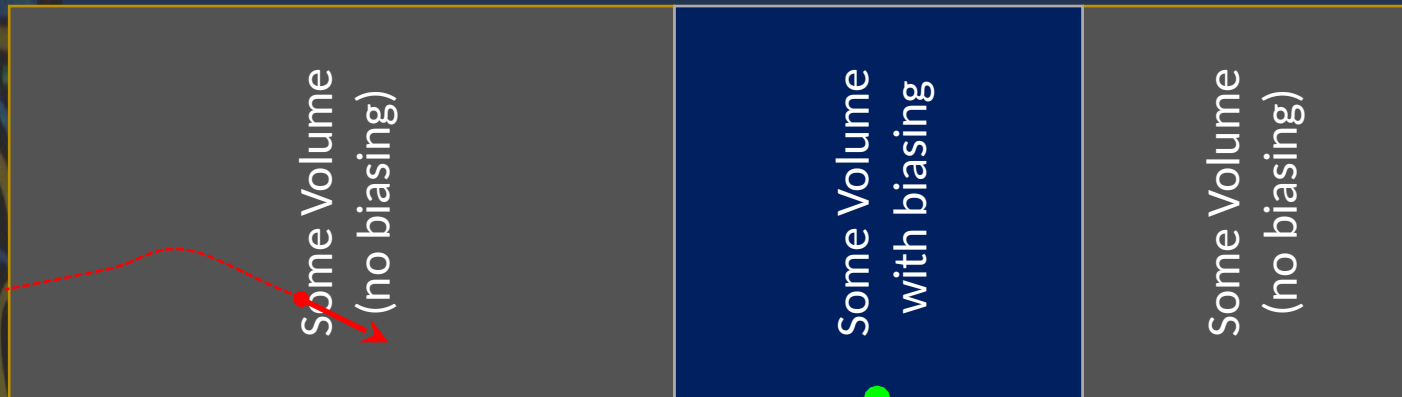
> Illustrate with point-like processes:



1. `GetPostStepPhysicalInteractionLength()`
 - Returns the distance at which the process will make an interaction
 - ~~Analog exponential law is at play~~
 - Above analog behavior superseded by biased behavior (if wished)
2. `PostStepDoIt()`
 - Called if the process has responded the shortest of the interaction distances
 - ~~Generate final state, according to specific process analog physical law~~
 - Above analog behavior superseded by biased behavior (if wished)

> `G4BiasingProcessInterface` wraps the physics process. It is meant to intercept calls for distance of interaction and for final state production.





G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

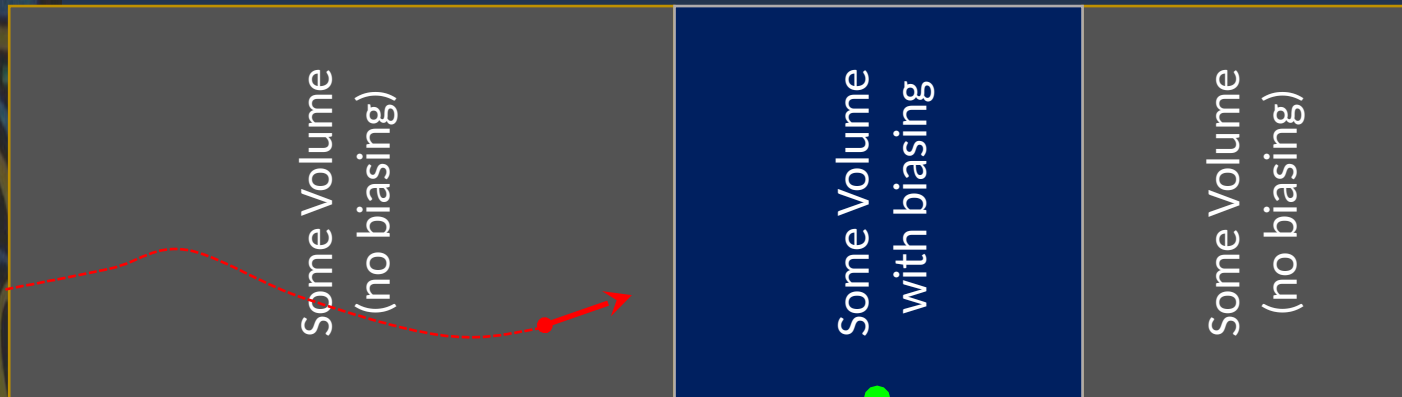
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

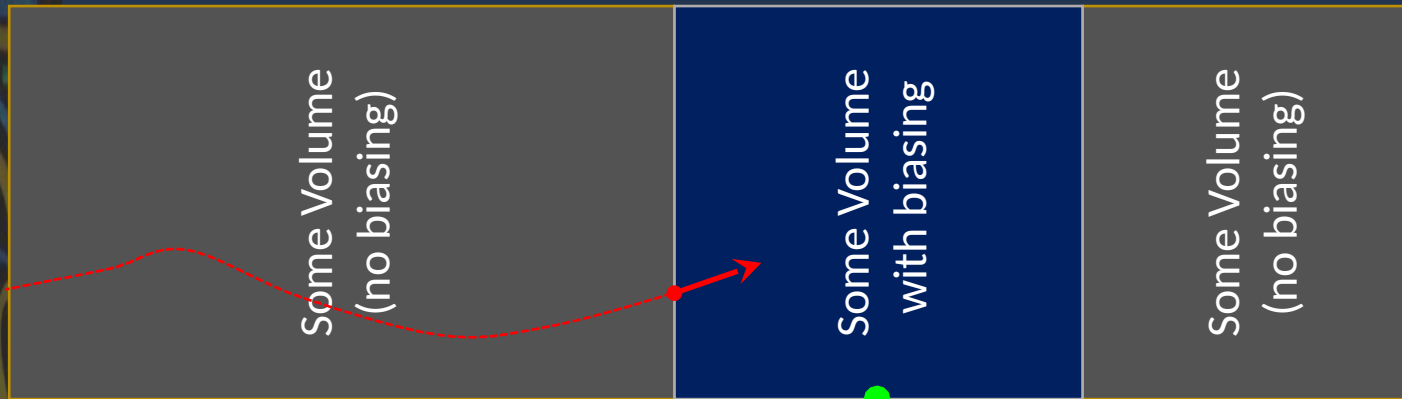
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

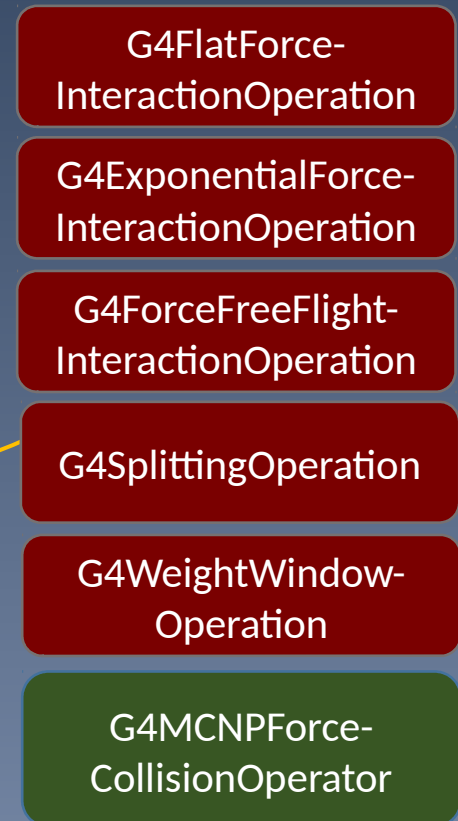
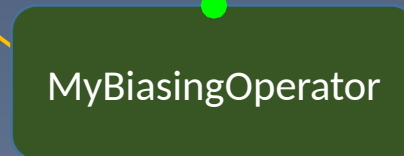
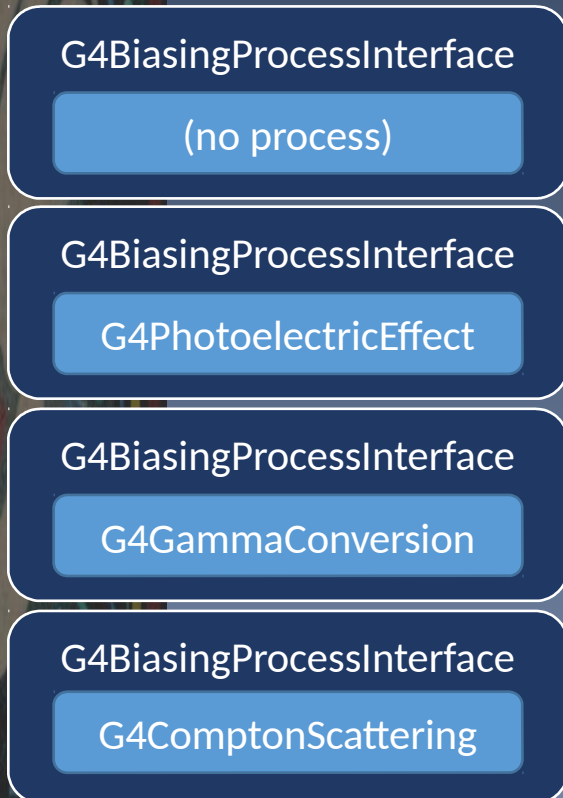
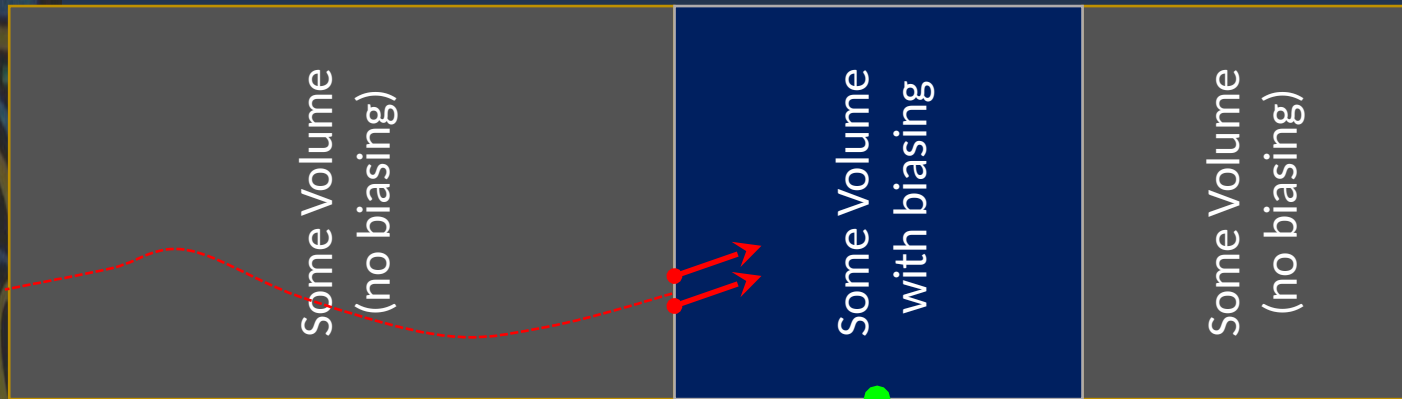
G4ExponentialForce-
InteractionOperation

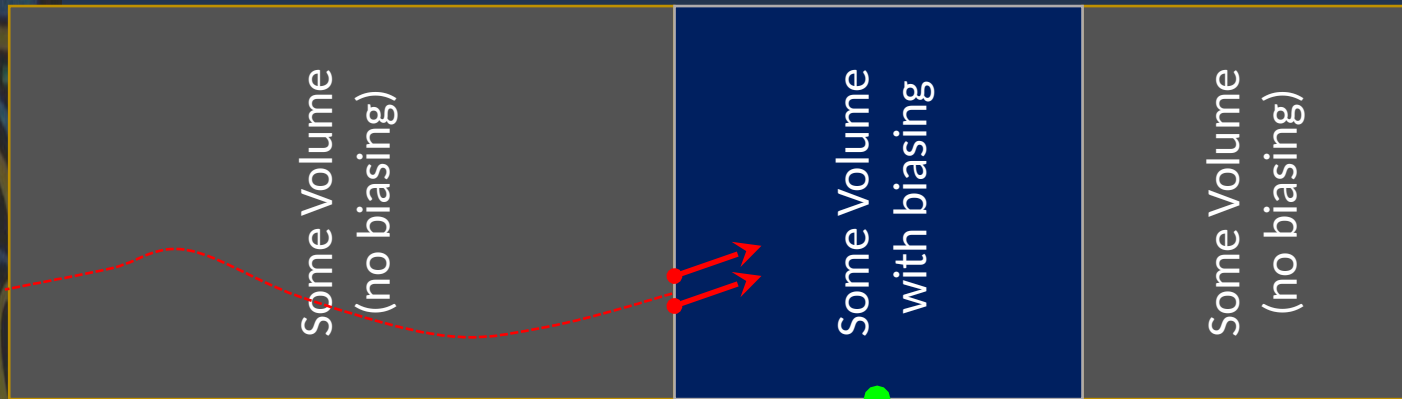
G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator





G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

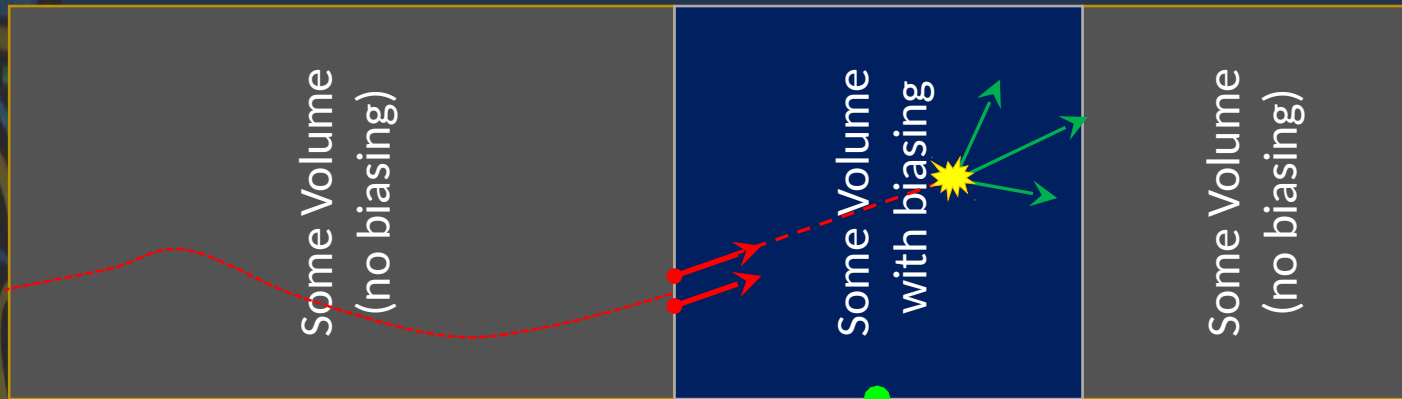
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

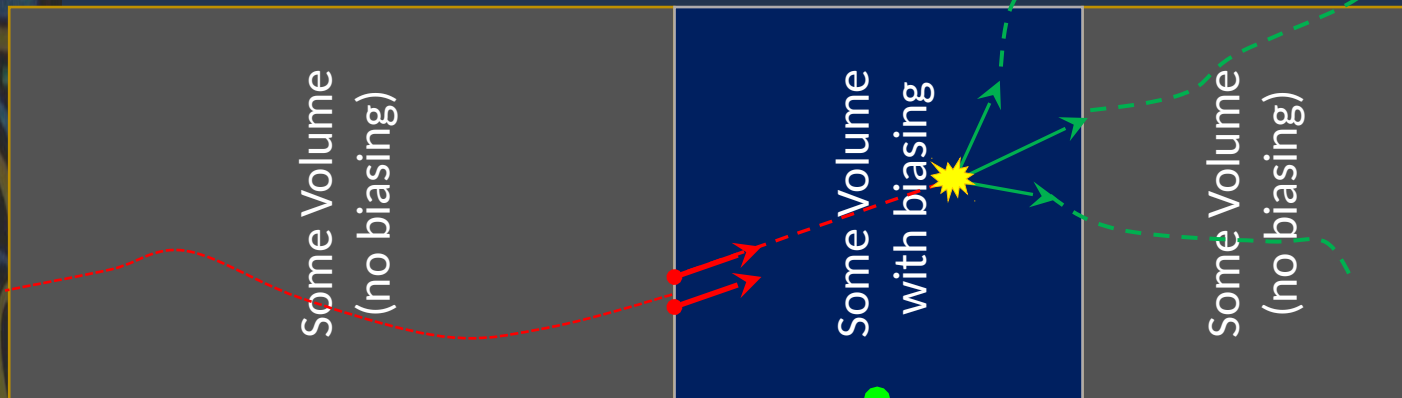
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



G4BiasingProcessInterface

(no process)

G4BiasingProcessInterface

G4PhotoelectricEffect

G4BiasingProcessInterface

G4GammaConversion

G4BiasingProcessInterface

G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

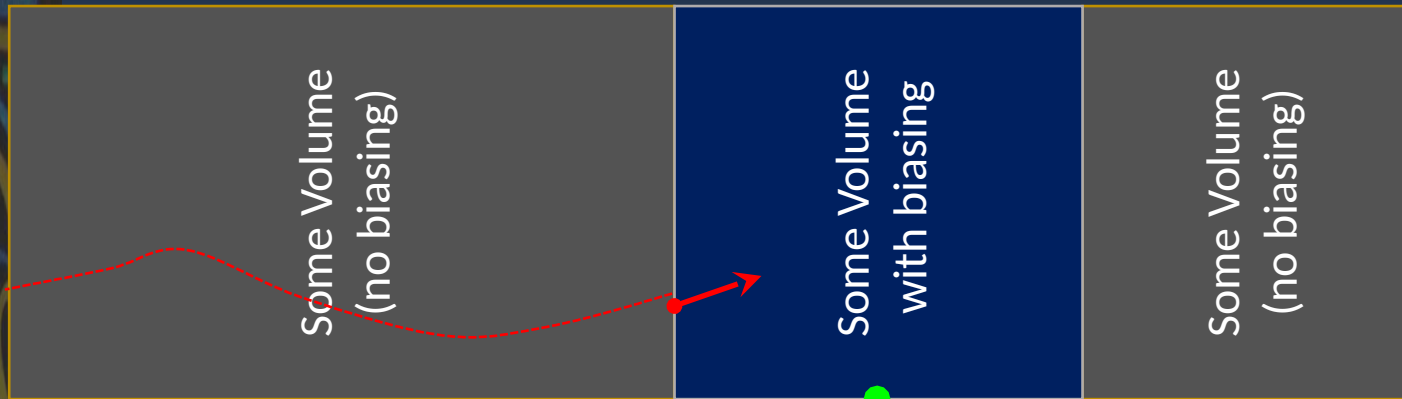
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

MyBiasingOperator

G4FlatForce-
InteractionOperation

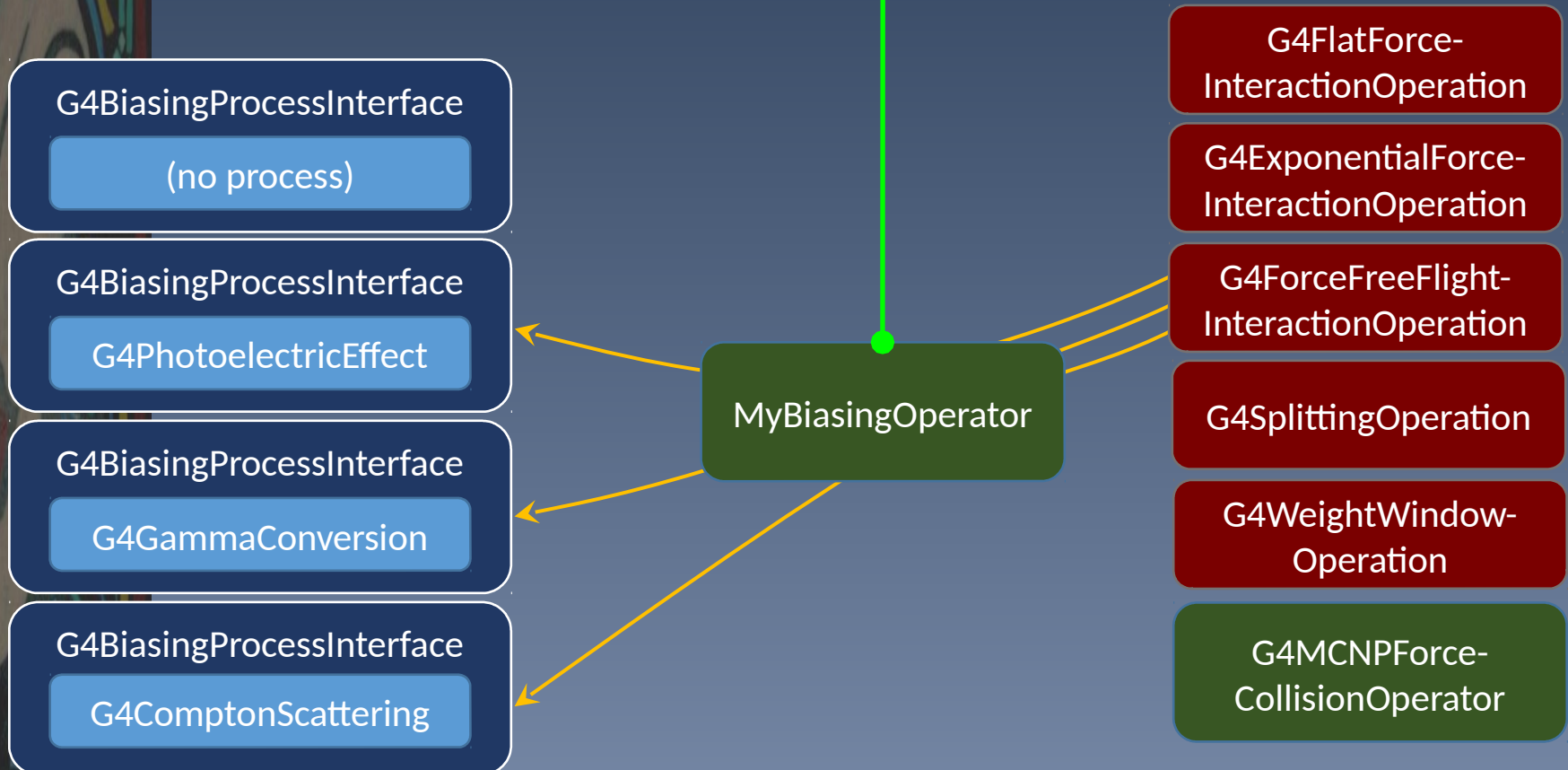
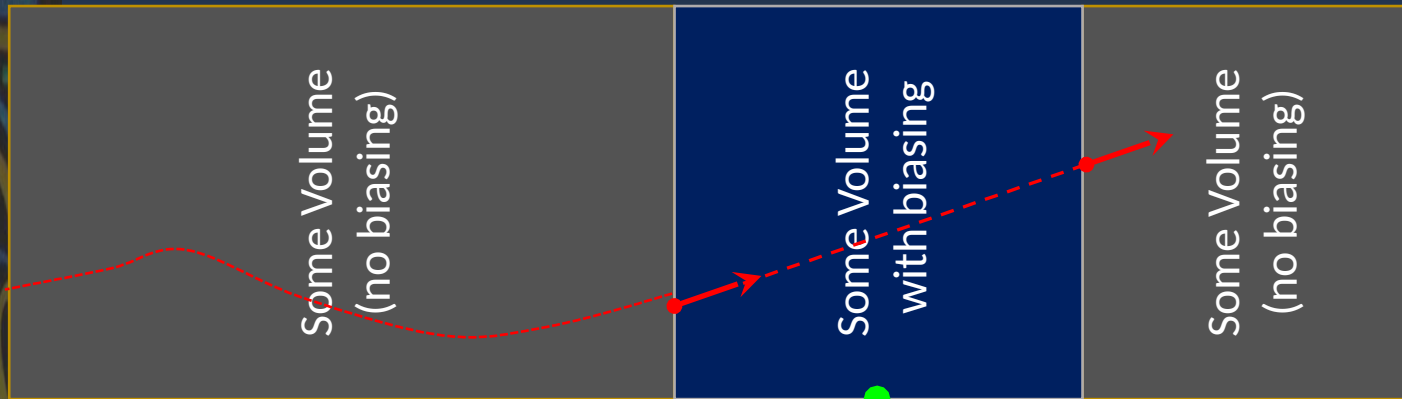
G4ExponentialForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

G4MCNPForce-
CollisionOperator



How to use

> In main program:

```
// -- Select a modular physics list
FTFP_BERT* physicsList = new FTFP_BERT;
// -- And augment it with biasing facilities:
G4GenericBiasingPhysics* biasingPhysics = new G4GenericBiasingPhysics();
biasingPhysics->Bias("gamma");
biasingPhysics->Bias("neutron");
physicsList->RegisterPhysics(biasingPhysics);
runManager->SetUserInitialization(physicsList);
```

> In ConstructSDandField() of detector construction:

```
MyBiasingOperator* biasingOperator = new MyBiasingOperator();
biasingOperator->AttachTo( logicalVolumeToBias );
```

> Examples:

- geant4/examples/extended/biasing/GB01 : individual process cross-section biasing
- geant4/examples/extended/biasing/GB02 : force collision à la MCNP
- More examples expected this year

> Above lines are for using existing biasing operations or operators

> G4VBiasingOperation and G4VBiasingOperator are base classes meant for extending functionalities (toolkit approach)

- But writing such classes is generally demanding

> This development is “young” and is expected to be enriched and evolved



Summary

- Biasing techniques can provide very large acceleration factors in problems in which we must tally rare events
- Geant4 proposes biasing options
 - Primary source (GPS) , leading particle, cross-section (had), radioactive decay, splitting with importance in geometry, weight window, user biasing with G4WrapperProcess, and bremstrahlung splitting
- New “toolkit-like” approach since v10.0
 - Allowing biasing of process interaction and process final state
 - Together with easy mix with killing, splitting, etc.
 - Introduction of base classes allows extension of functionalities et re-use of components
- Techniques delicate to handle, but nevertheless unique in addressing rare event simulation problems.



Backup



Some Words of Caution with Convergence



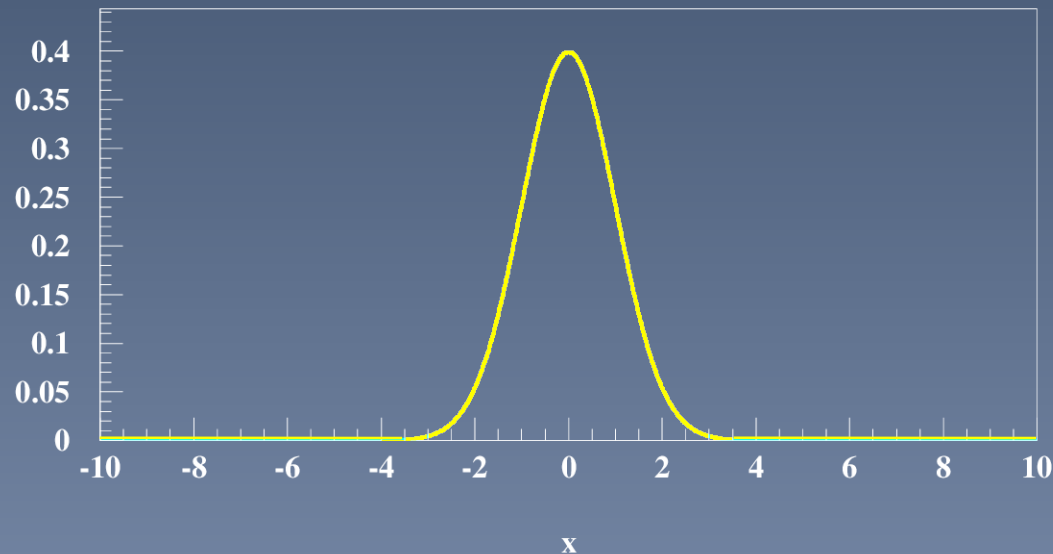
Introduction

- › Biasing techniques are powerful
 - If a technique is well suited to a problem, gains in simulation efficiency by several order of magnitudes are usual
- › In contrast to an analog simulation where all “events” have the same weight ($=1$), biasing is not “democratic”
 - Because of the weight, some events are more important than others
- › This weight disparity may cause convergence problems
 - That can be nasty !
 - What happens is that, from time to time, an event with big weight may come, and change drastically the –say- estimated current mean value
 - Always a fear that this may happen...
- › Proper convergence is the issue the user of biasing has to care about
 - And this is the price to pay for using biasing
- › Let’s illustrate a “bad convergence case” with a simple problem
 - But this allows to spot the difficulties

Convergence

> A « toy » example of a bad sampling:

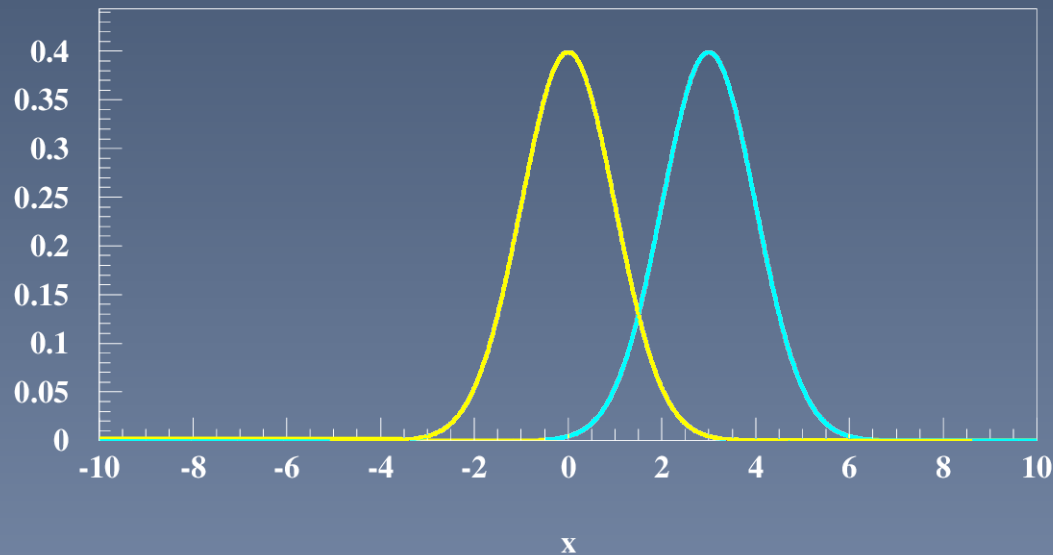
- We want to estimate the mean value of the unknown « yellow » distribution.
- As we don't know where it is, we try to sample it with the « blue » distribution
 - > And we know how to compute the weight

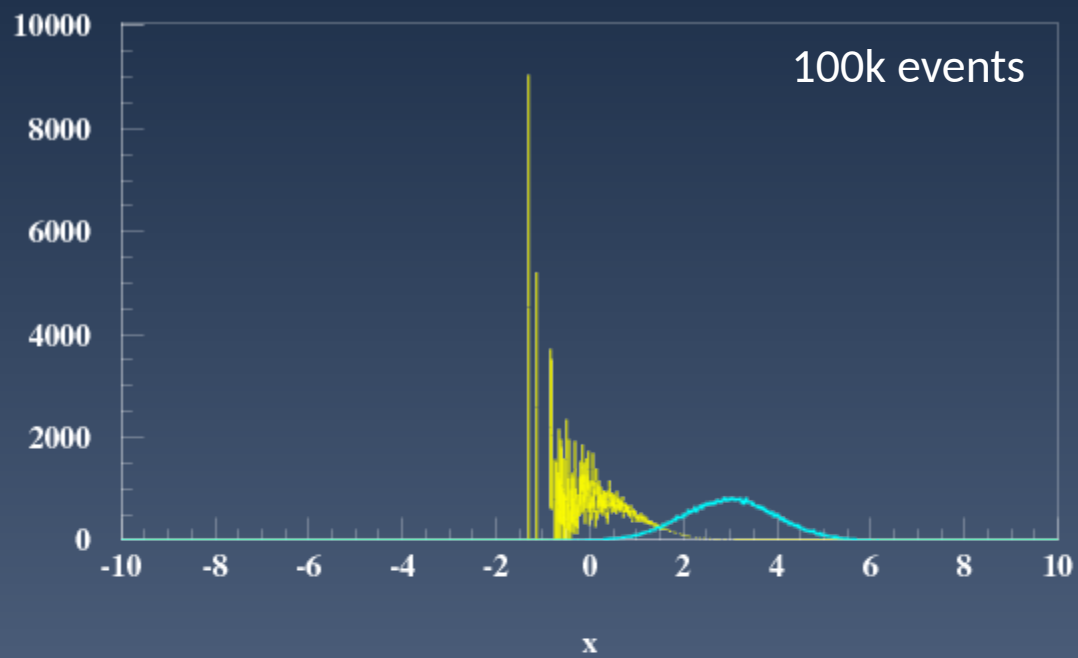


Convergence

> A « toy » example of a bad sampling:

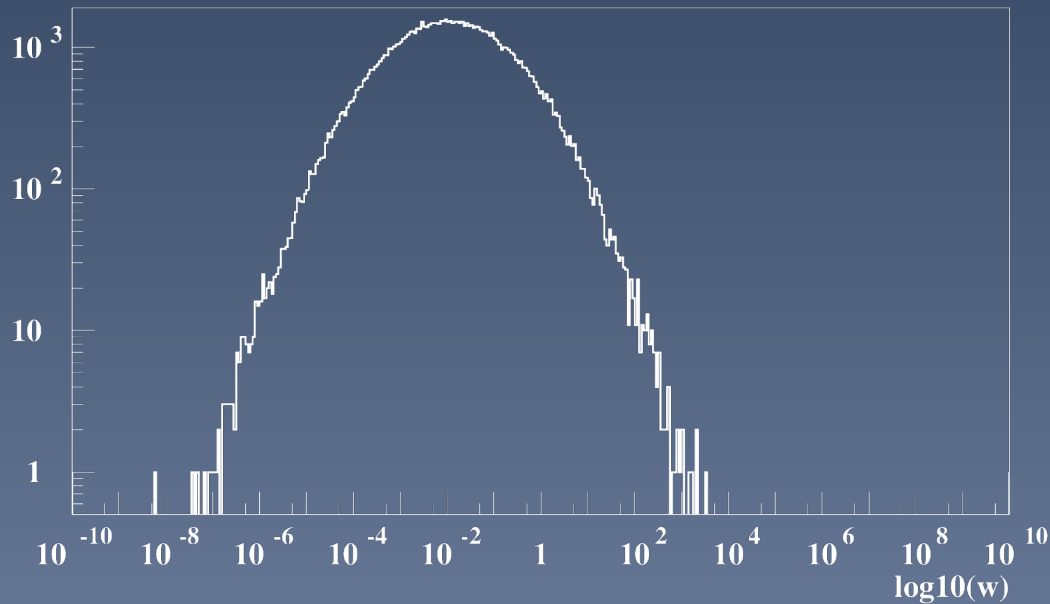
- We want to estimate the mean value of the unknown « yellow » distribution.
- As we don't know where it is, we try to sample it with the « blue » distribution
 - > And we know how to compute the weight



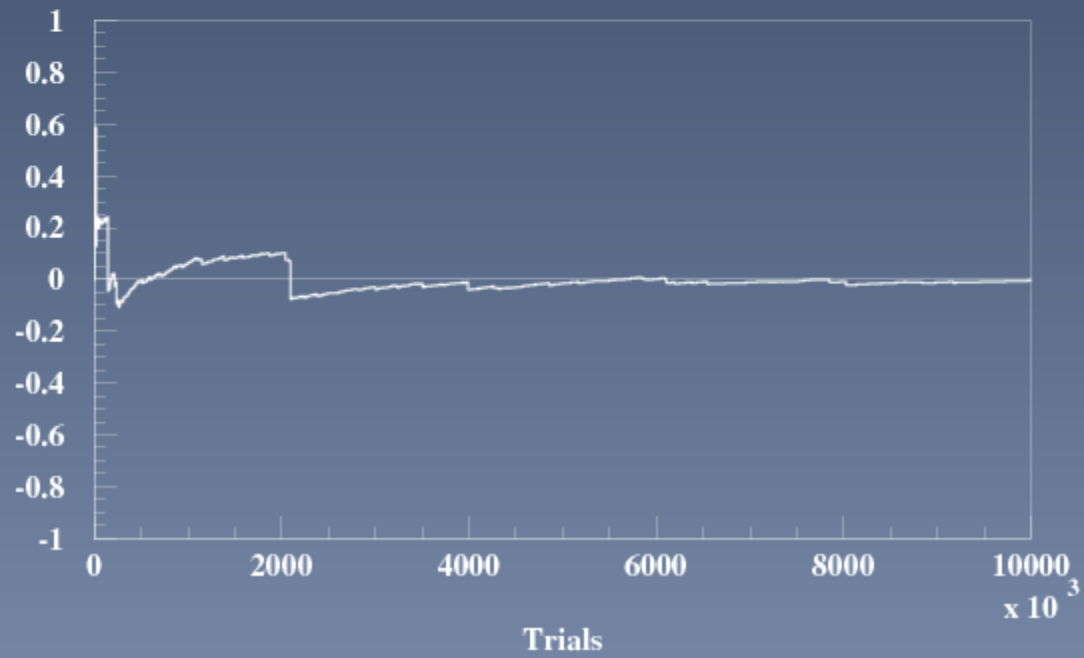
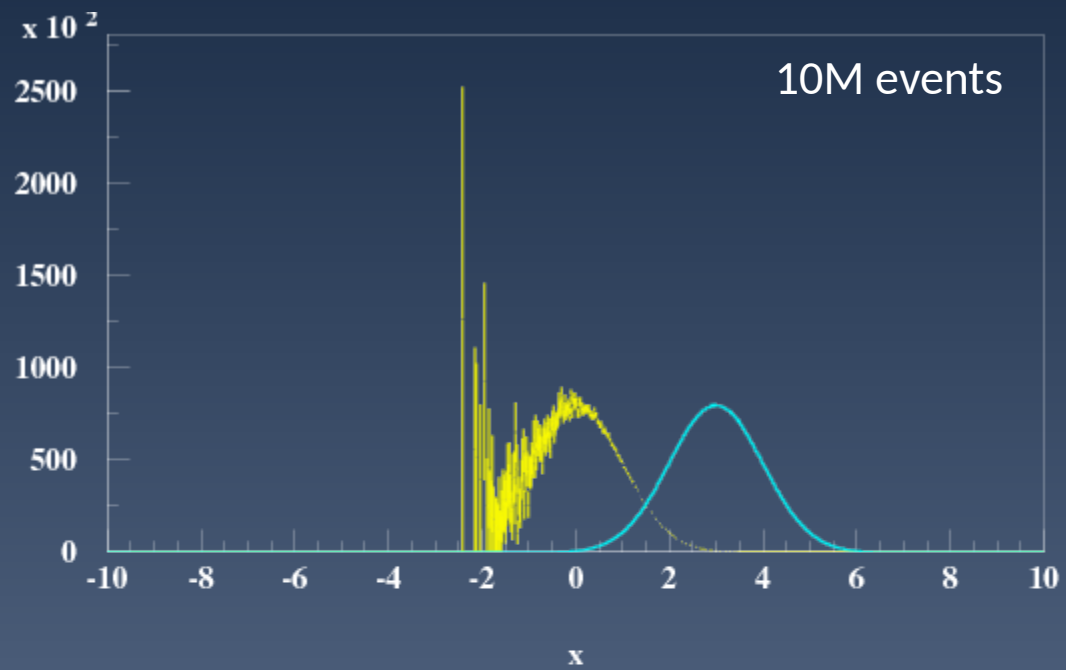


Weight distribution...

- › Lots of tiny weights...



- › Few huge ones...





Some qualitative observations

- › Observations that can help spotting the inappropriate sampling:
 - We have a wide variety of weights
 - › Many tiny ones
 - Waste of time to determine the mean value
 - › Few huge ones
 - Responsible for jumps
 - › Such problem –if can not be improved- could at least be alleviated with a weight window technique
 - In a real problem (useless here in our toy problem)
 - We have huge weights from time to time
 - › These are not wrong !
 - › Temptation would be to “dismiss” these events
 - › But in our case, these events bring down the mean value to the correct value
 - › These are not wrong, but ***their presence is a sign of a bad sampling***
 - We observe monotonic increase of the mean value
 - › We are sampling only “one side” of the problem
- › Convergence criteria have been established in other packages
 - Like MCNP
- › You are invited at staying critical when using such biasing techniques !