

Progress on generic biasing

Marc Verderi
LLR – Ecole polytechnique

Geant4 Collaboration Meeting
Seville
September 2013

INTRODUCTION

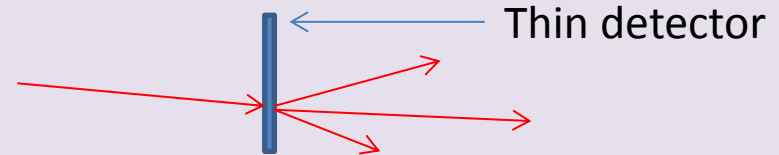
Introduction / reminder (1/2)

- Geant4 proposes biasing options
 - Geometrical importance sampling, Leading particle biasing, Radioactive decay biasing, G4WrapperProcess, Reverse MC

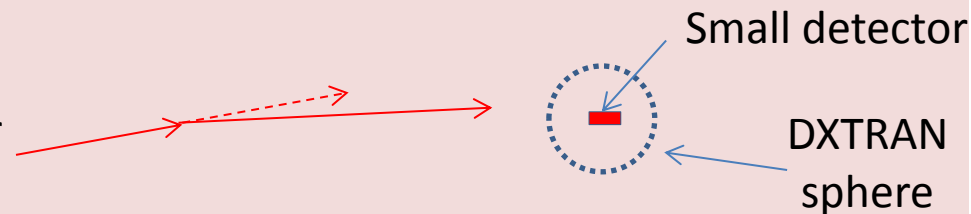
- But misses others

- 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.

- These options implies changing the behavior of either
 - the interaction probability (PostStep GPIL)
 - the generation of the final state (PostStep Dolt)

Reminder (2/2)

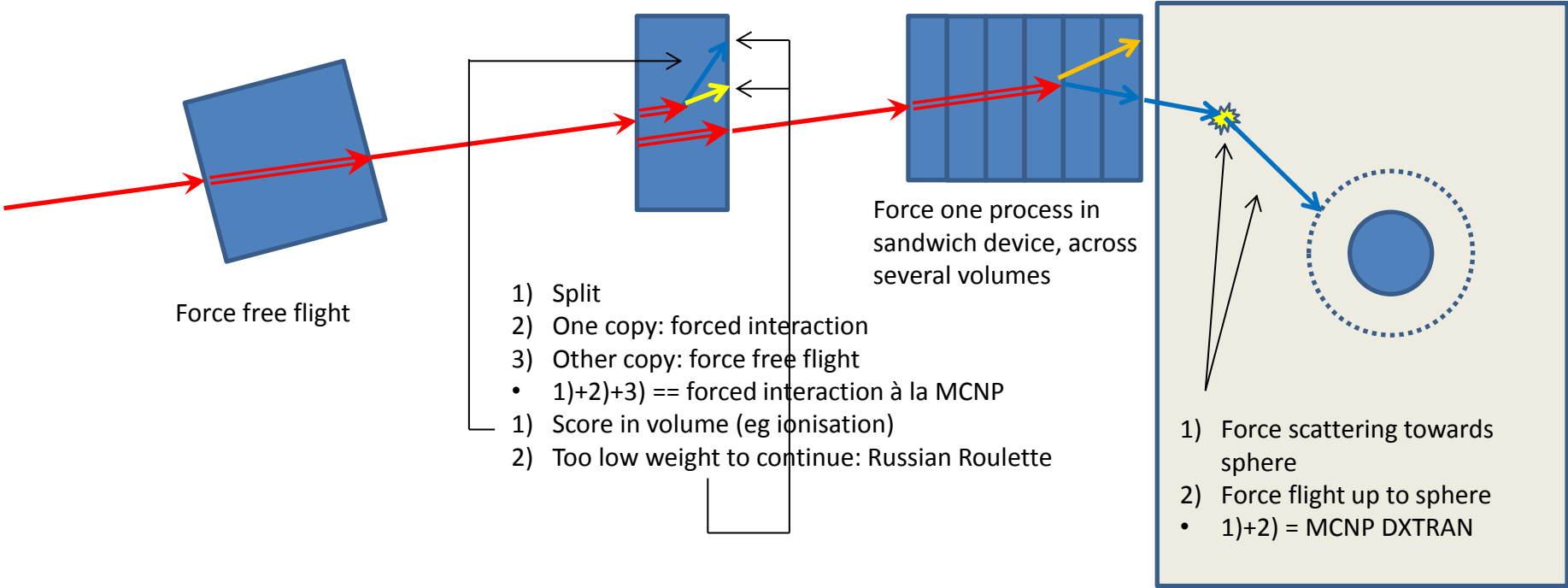
- We rely on the identification of the two main biasing techniques:
 - Importance sampling
 - Change of (analog) probabilities (ie : GPIL, Final State)
 - Splitting (killing)
 - Split(kill) particles when moving towards the (un)desired direction of phase space.
- For biasing of GPIL (importance sampling) we reviewed the formalism, inferred how to fit it in Geant4, and came to:
 - 1) The sampling of the interaction law should be made according to the biased law in the PostStepGPIL, while remembering the analog cross-section for further weight calculation
 - 2) At each step, and for each biased process, a non-interaction weight has to be applied.
 - This weight is $w_{NI}(l) = P_{NI}^{analog}(l)/P_{NI}^{biased}(l)$, where the P_{NI} are the non-interaction probabilities in the analog and biased schemes, over a step of length l .
 - 3) When a step ends with an interaction produced by one process, an interaction weight has to be applied too.
 - This weight is $w_I(l) = \sigma/\sigma_{eff}(l)$, where σ is the analog process cross-section and $\sigma_{eff}(l) = p(l)/P_{NI}^{biased}(l)$; $P_{NI}^{biased}(l) = \int_0^l p(x)dx$; where $p(l)$ is the biased interaction law.

DESIGN APPROACH

Approach of design

- FLUKA and MCNPX control biasing through “data cards”
 - Advantage of being robust
 - But a new functionality requires a new development of the software
- Try to follow a “toolkit” approach, taking advantage of OO technology
 - Try to model the problem in term of a few abstract classes
 - that provide the interfaces to the Geant4 kernel
 - And let the concrete cases be implemented inheriting from these abstract classes
 - Eg : splitting, killing, forced collision, brem. splitting, etc.
 - Then Geant4 can provide some usual implementations of concrete cases
 - That can be controlled with “command line”, in a “data cards”- style
 - But users are free to extend the toolkit with their own dedicated biasing techniques
 - If none of the options provided respond to their need
 - Without the need (hopefully) of modifying the Geant4 kernel
- Some thinking on cartoon-like and real cases...

A cartoon



A real example, from MCNP

Neutron shielding problem:

- 1) Neutron source
 - Force emission direction
- 2) Cross concrete slabs
 - Force penetration
- 3) Travel up to top
- 4) Make tally in cell flux (F4)
 - Force interaction for scoring
- 5) Tally in detector off-axis (F5)
 - Force scattering towards detector

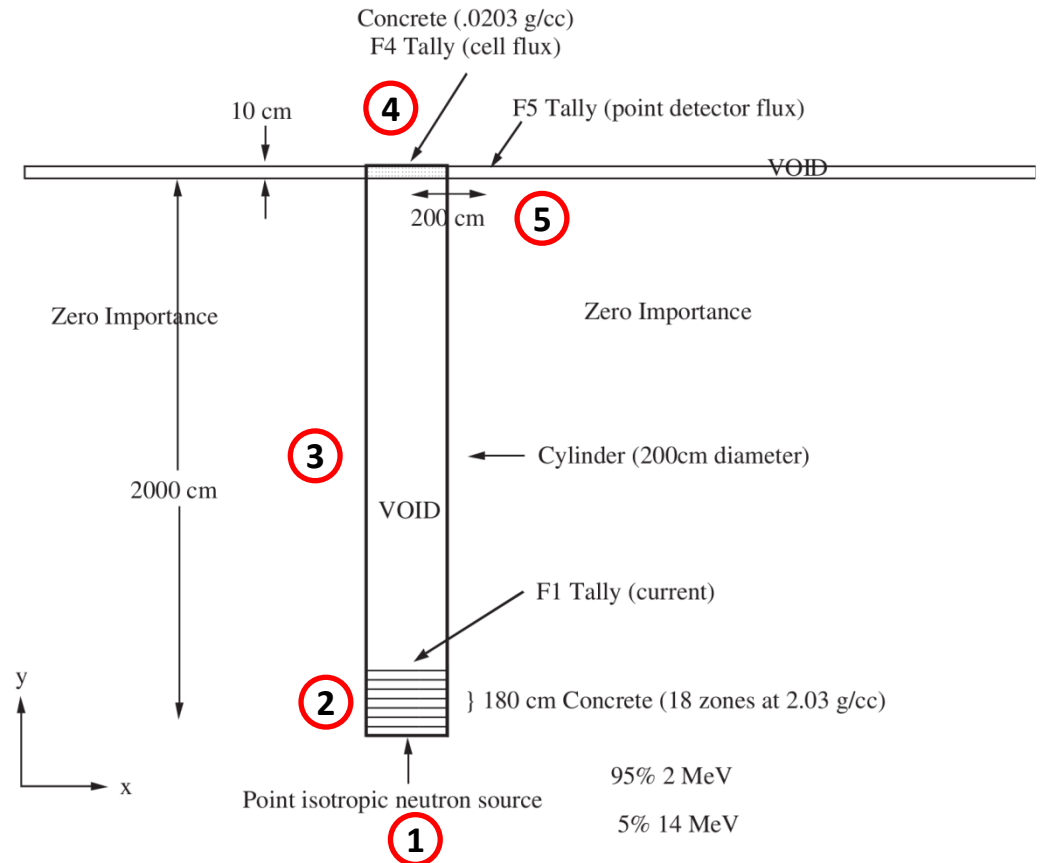


Figure 10.1 Transport problem.

Some observations

- Many techniques...
- Several techniques are used, and are used in non-trivial sequences
 - Decision making for applying these techniques needs lot of freedom
 - Decision to be made on volume, particle energy, position, etc. on a step by step basis
- Physics processes have to behave differently depending on volume or set of volumes
 - Change in their interaction probability
 - Change in their final state production
- Non-physics (splitting / killing) applies as well
- What do we identify ?
 - Biasing operations:
 - Splitting or killing
 - or change in interaction probability
 - or a change in final state production
 - A decision making entity, to “pilot” these biasing operations
 - A “biasing operator”, consulted step by step for decision
 - Need to control the behavior of physics processes
 - In their interaction probability
 - In their final state production

Components defined

- Classes that implement biasing code:
 - G4VBiasingOperation
 - Represent simple operations:
 - “non-physics” ones, eg : splitting, killing
 - Physics-modifying ones:
 - » Change in interaction probability : force collision, force free flight, exponential transform ...
 - » Change in final state production : change in angular/energy distribution, brem splitting, leading particle biasing, etc.
 - » Above two are independent and can be combined : eg, force e- collision with brem splitting
 - Operations can technically be delicate to be implemented (expose to the many tricks of the tracking...), but Geant4 can provide the common ones
 - G4VBiasingOperator
 - The decision making class.
 - It is meant for user’s code.
 - **It decides for the biasing operation to be applied.**
 - **And it gets reported about this application (particle change returned to tracking, etc.)**
 - It is messaged by the G4BiasingProcessInterface which enquires for the biasing operation to be applied for itself.
 - Again, some concrete implementation can be provided for common cases.
- Class that interfaces the biasing code with the tracking:
 - G4BiasingProcessInterface
 - Checks for presence of a biasing operator in current volume
 - Messages it for getting actions on:
 - For physics physics:
 1. Occurrence biasing : biasing of PostStepGPIL
 2. Shortening of AlongGPIL
 3. Final State biasing : biasing of PostStepDolt
 - It also remembers the physics process analog cross section (interaction length)
 - For non physics biasing

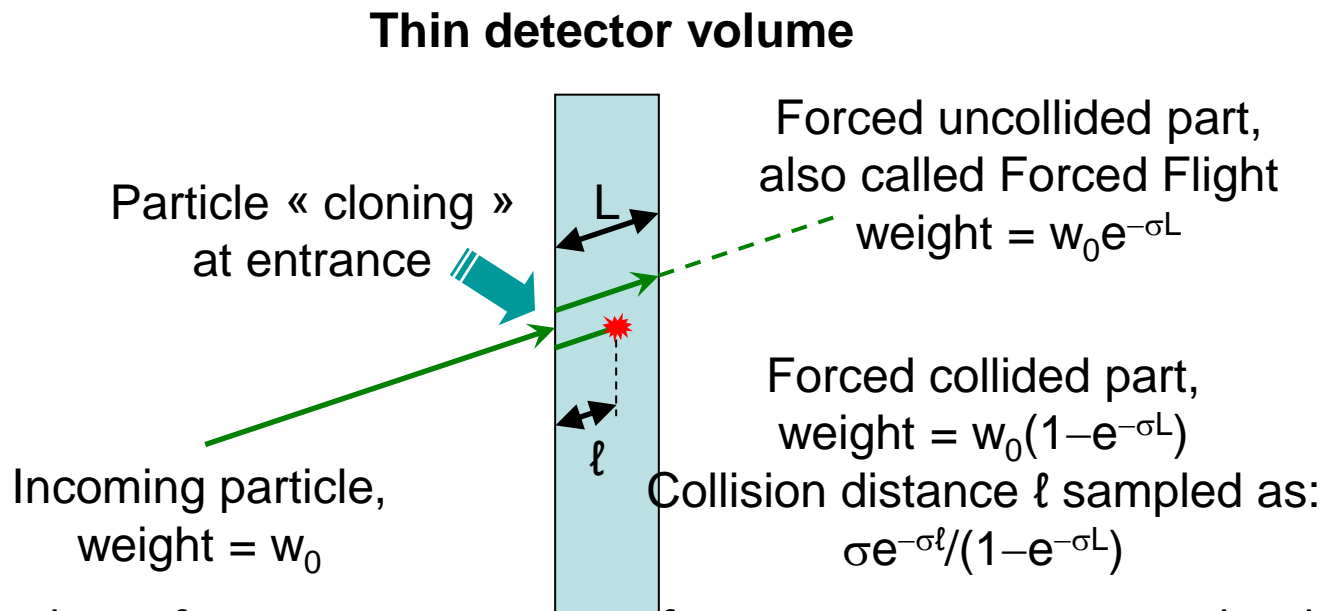
Examples of biasing operations

- **Physics-based biasing operations:**
- Occurrence biasing operations:
 - Forced interaction
 - With exponential of flat law
 - Exponential transform
 - Change of cross-section
 - Forced free flight
 - Non-interaction probability = 1 always
 - Effective cross-section = 0
 - Can imagine also:
 - Point-like force (eg: forcing interaction at birth place)
 - Non-interaction probability = 0
 - Effective cross-section = Infinity
 - Note BTW that:
 - Making an operation acting on individual process interaction law is easy
 - And fits easily in G4 tracking
 - Making an operation working on the “total cross-section” is more tricky
 - But useful
- Final state biasing actions:
 - Leading particle
 - Brem. splitting
 - Force scattering towards some direction
- **Non-physics-based operations:**
 - Splitting
 - Act immediately, with zero step
 - Geometry splitting:
 - Act on next boundary
 - Makes use of force flag
 - Killing / Russian roulette
 - May act immediately, with zero step
- **But also ?**
 - Exploratory track operation ?
 - Not really “biasing operation” as not leading to a physics scoring, but useful for biasing
 - Would collect on the track way:
 - Geometry information (volumes, distances, etc.)
 - Physics information (cross-sections, int. length,...)
 - Could be for neutral and charged particles, in magnetic field
 - Deactivation of MSC fluctuations ?
 - Should be careful not to call sensitive/scoring
 - Prevent sensitive calls : flag exists
 - Scoring : set weighth = 0
 - Clone a track (and suspends original) and follows its tracking up to some travelled distance (in cumulated length, or interaction length, etc.)
 - And terminate it with fStopAndKillSecondaries
 - Fictitious interaction/Woodcock tracking:
 - Acceleration tracking technique for neutral
 - Step limitation based on highest (total) σ in –say– region, ignoring volumes, and make fictitious (nothing) or real interaction at the point depending on actual σ .
 - Makes uses of fExclusivelyForce flag

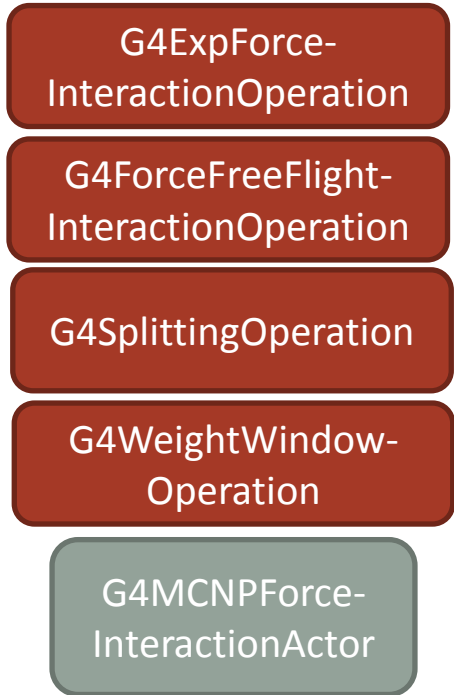
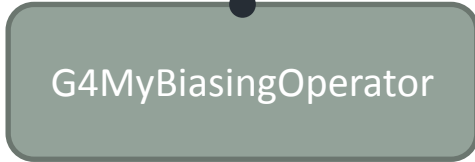
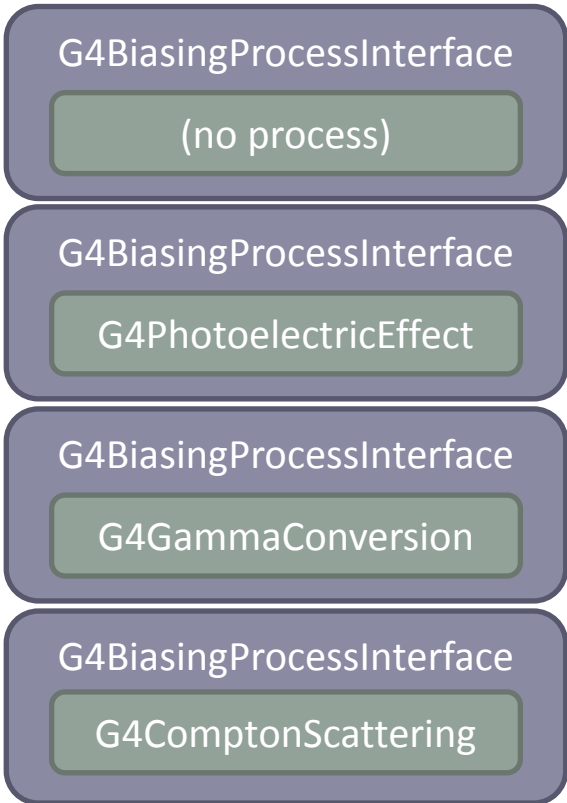
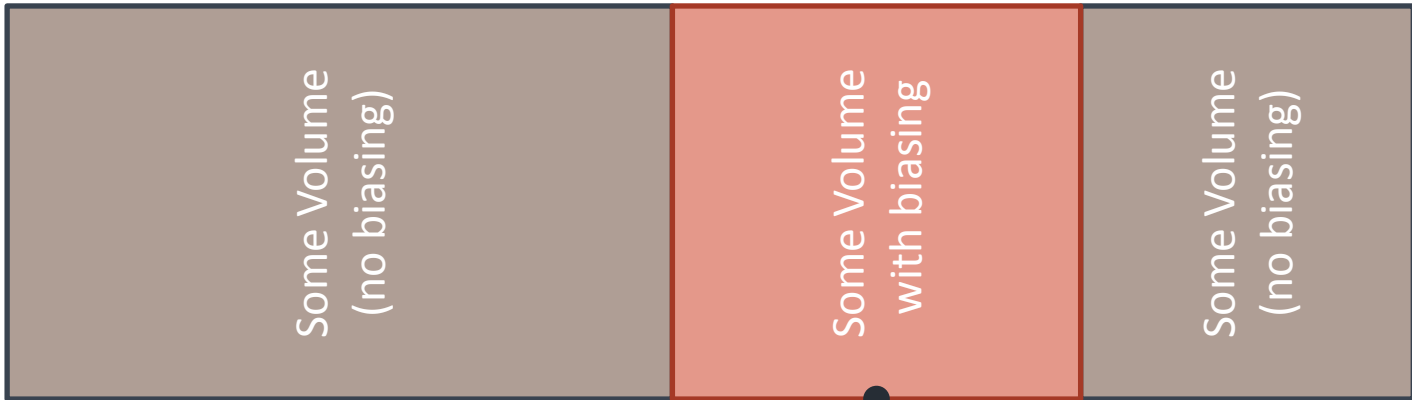
IMPLEMENTATION OF AN MCNP- LIKE FORCE COLLISION SCHEME

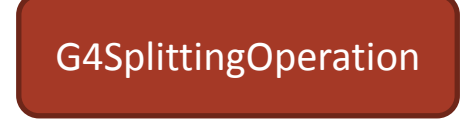
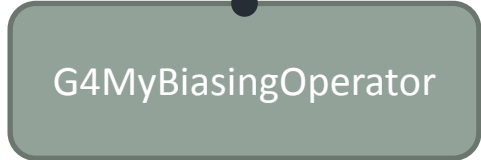
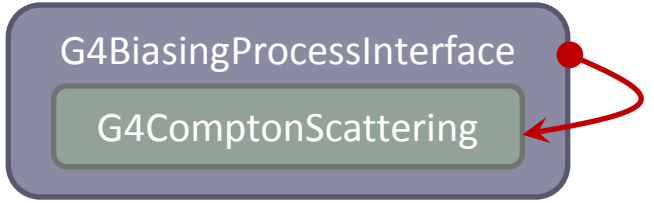
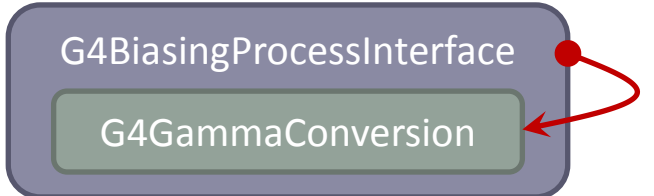
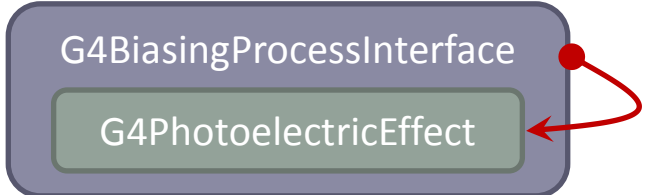
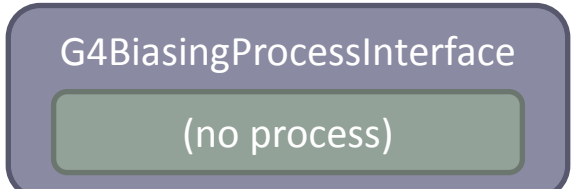
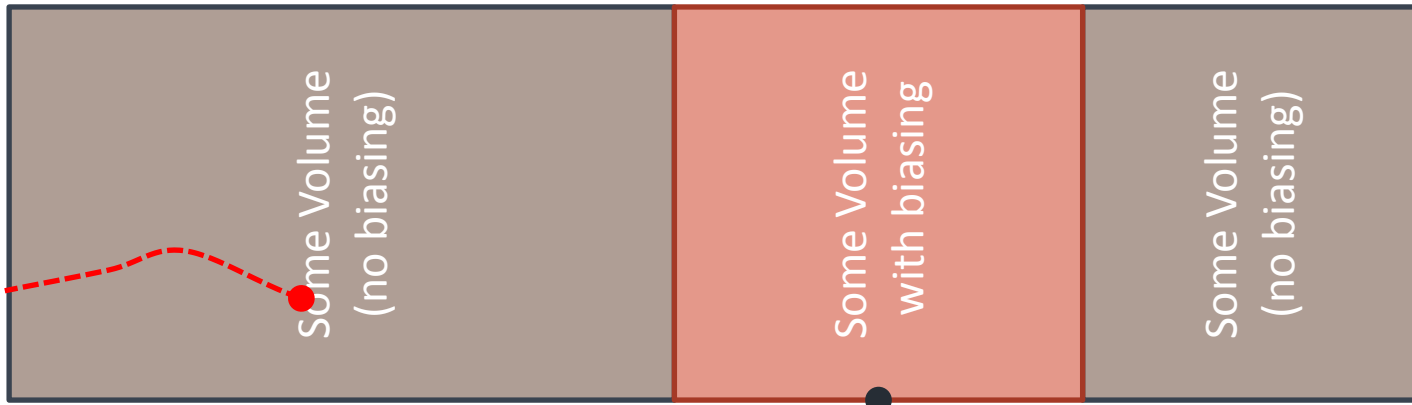
Emulation of an MCNP-like “force collision”

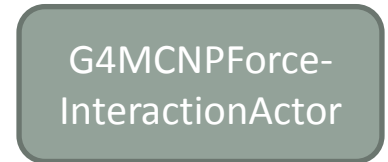
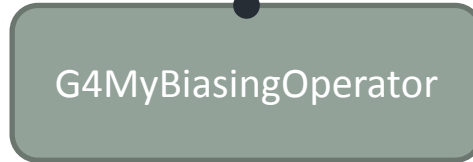
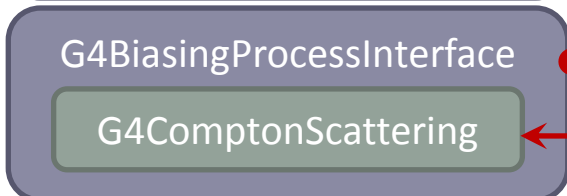
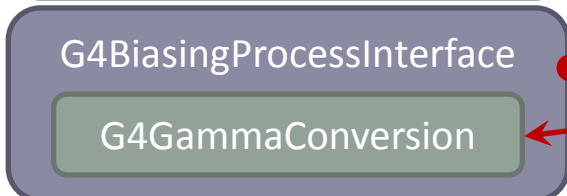
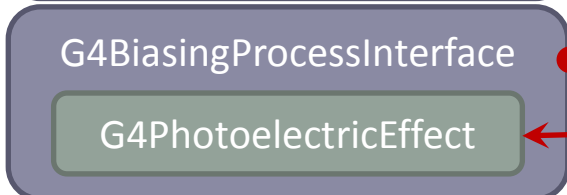
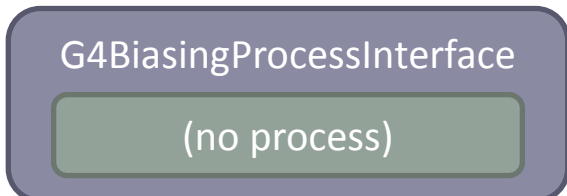
- “Exercise” to challenge proposed design
 - Can it satisfy this existing / popular option ?
- MCNP scheme:

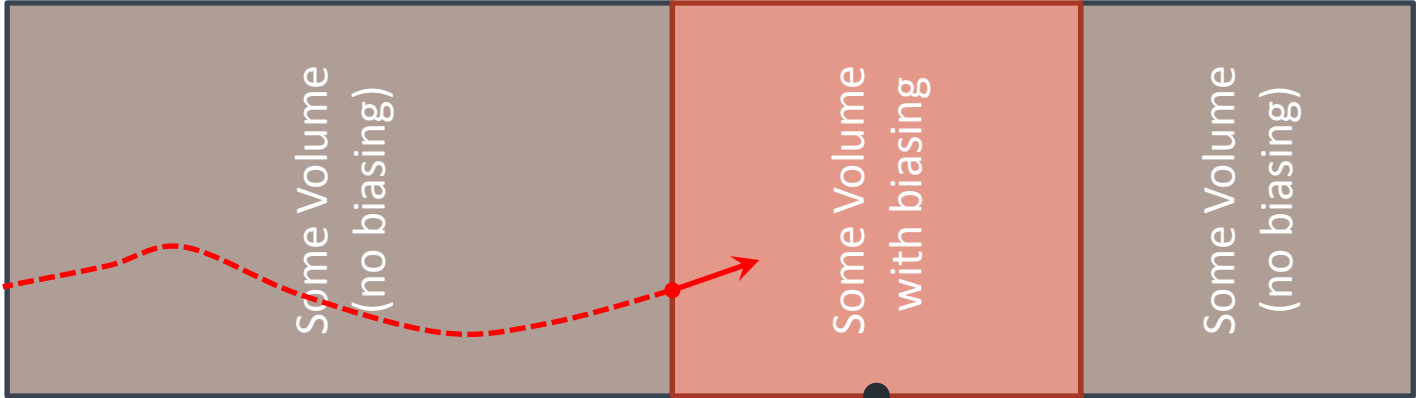


- Involves : forcing an interaction, forcing an interaction -with related weight computations, relying on *our* formalism- , do that using the *total* cross-section –making some “cooperation” of operation- : a pretty challenging exercise.
- Coming cartoon illustrates the way it works









G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

G4MyBiasingOperator

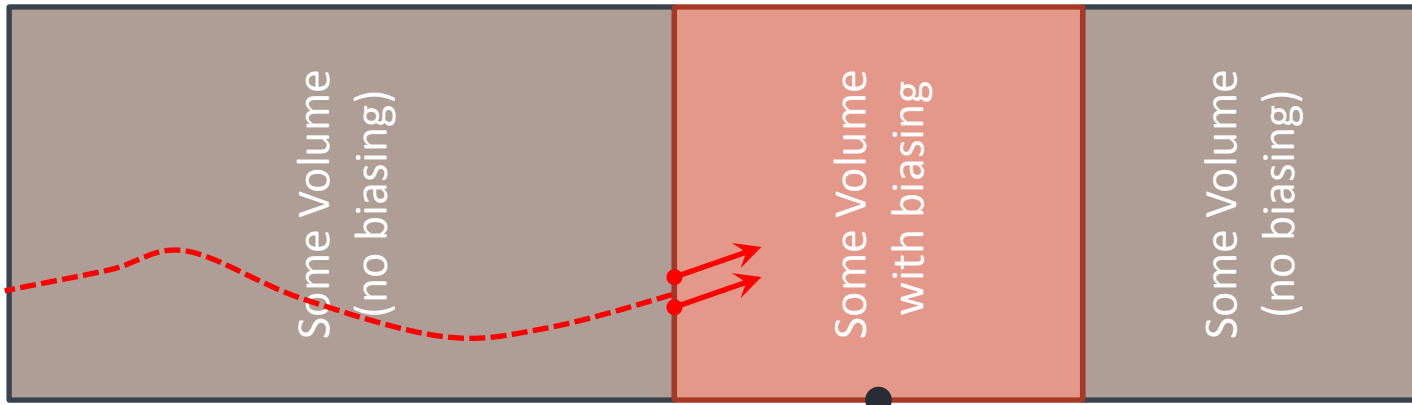
G4ExpForce-
InteractionOperation

G4ForceFreeFlight-
InteractionOperation

G4SplittingOperation

G4WeightWindow-
Operation

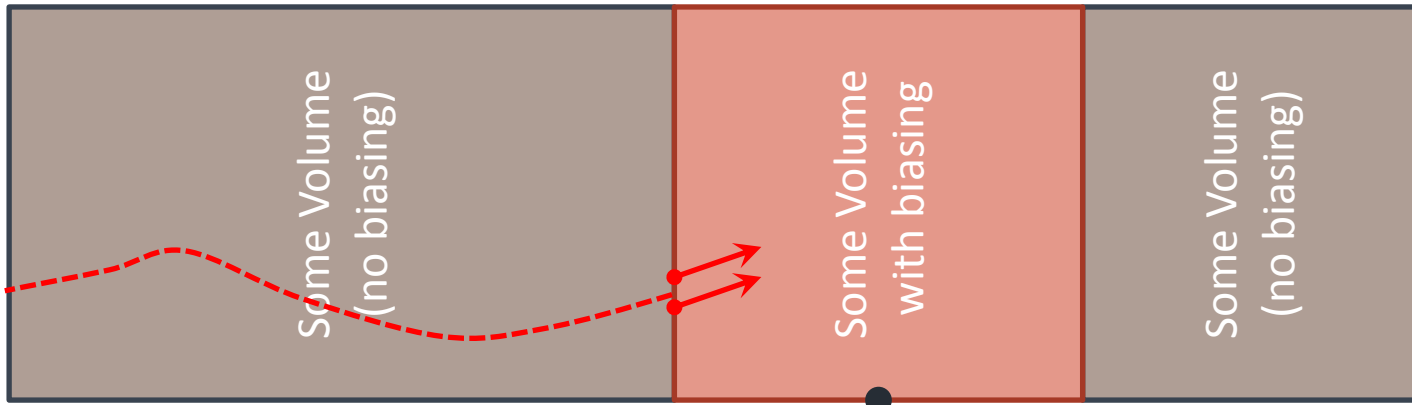
G4MCNPForce-
InteractionActor



- G4BiasingProcessInterface
(no process)
- G4BiasingProcessInterface
G4PhotoelectricEffect
- G4BiasingProcessInterface
G4GammaConversion
- G4BiasingProcessInterface
G4ComptonScattering

G4MyBiasingOperator

- G4ExpForce-InteractionOperation
- G4ForceFreeFlight-InteractionOperation
- G4SplittingOperation
- G4WeightWindow-Operation
- G4MCNPForce-InteractionActor



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

G4MyBiasingOperator

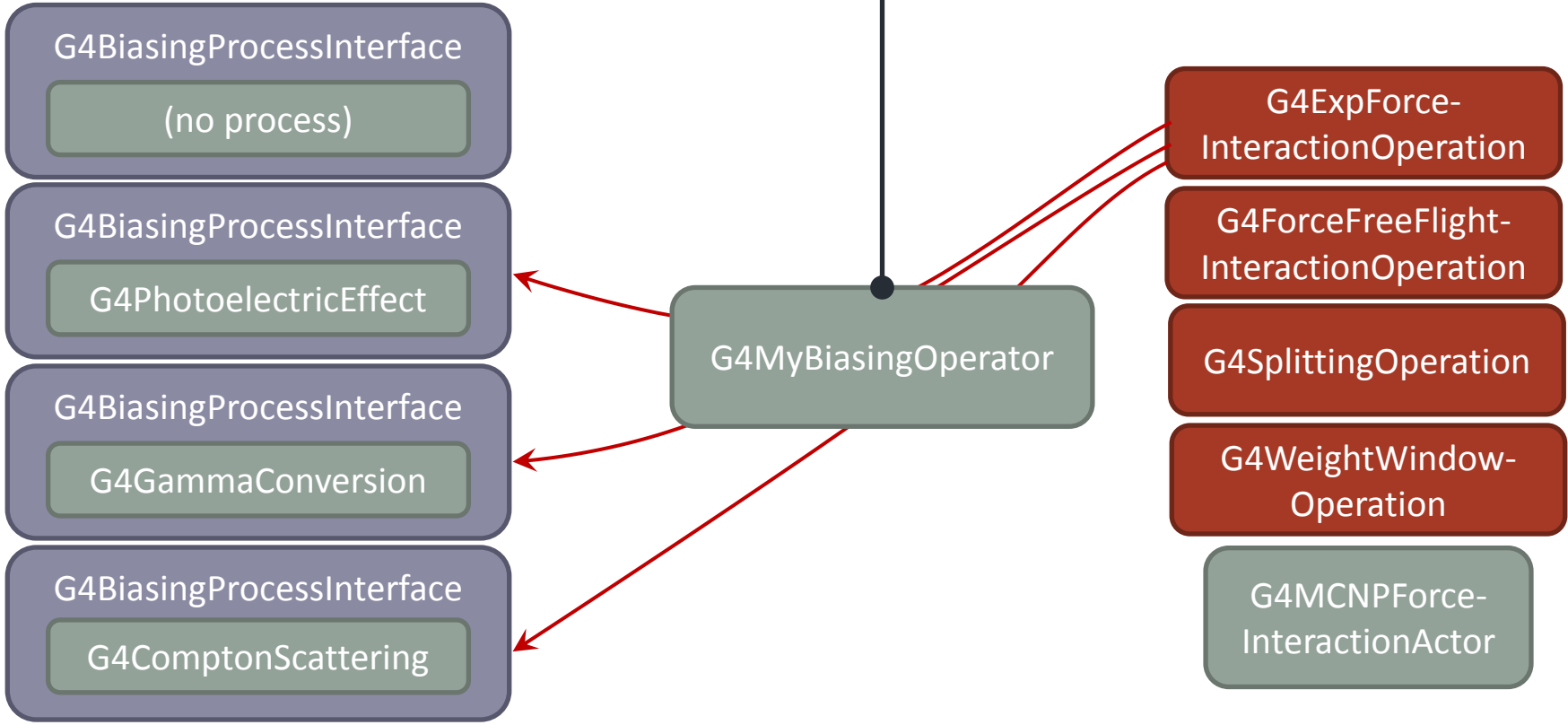
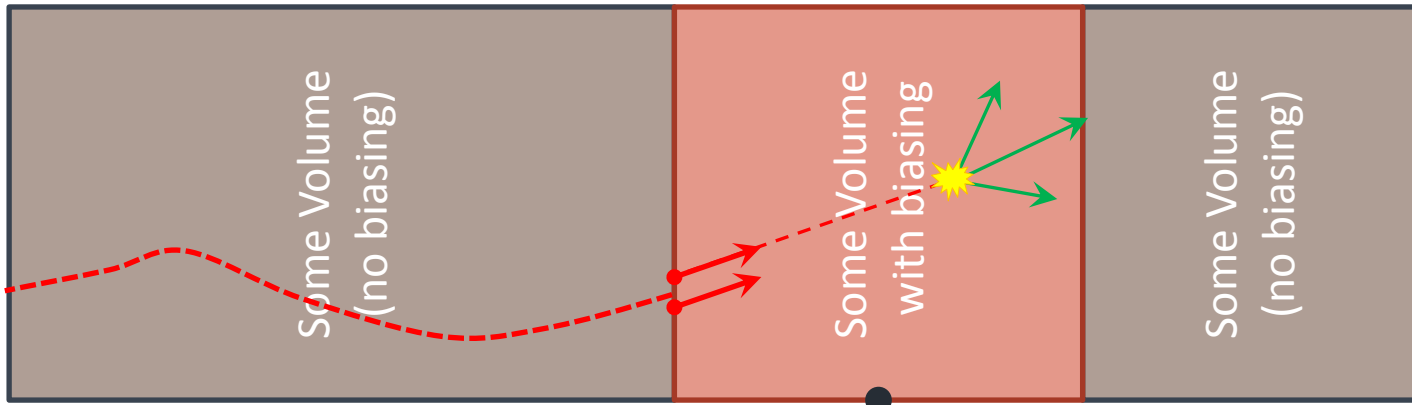
G4ExpForce-InteractionOperation

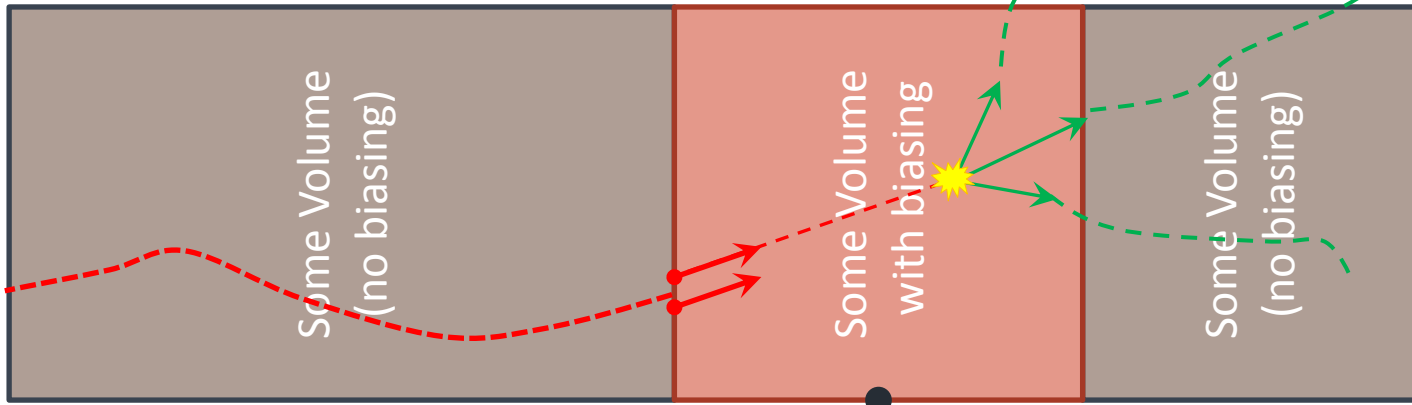
G4ForceFreeFlight-InteractionOperation

G4SplittingOperation

G4WeightWindow-Operation

G4MCNPForce-InteractionActor

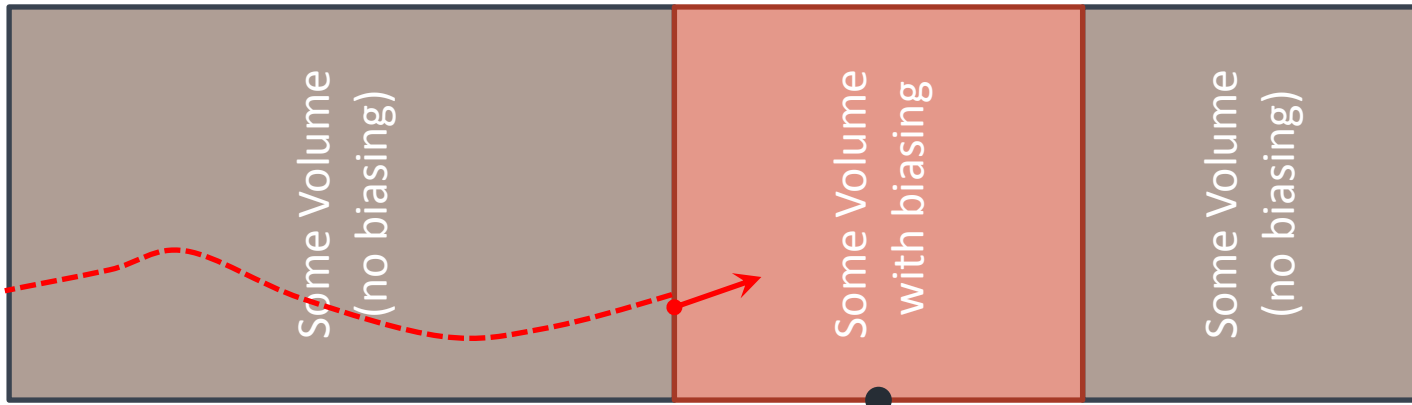




- G4BiasingProcessInterface
(no process)
- G4BiasingProcessInterface
G4PhotoelectricEffect
- G4BiasingProcessInterface
G4GammaConversion
- G4BiasingProcessInterface
G4ComptonScattering

G4MyBiasingOperator

- G4ExpForce-InteractionOperation
- G4ForceFreeFlight-InteractionOperation
- G4SplittingOperation
- G4WeightWindow-Operation
- G4MCNPForce-InteractionActor



G4BiasingProcessInterface
(no process)

G4BiasingProcessInterface
G4PhotoelectricEffect

G4BiasingProcessInterface
G4GammaConversion

G4BiasingProcessInterface
G4ComptonScattering

G4MyBiasingOperator

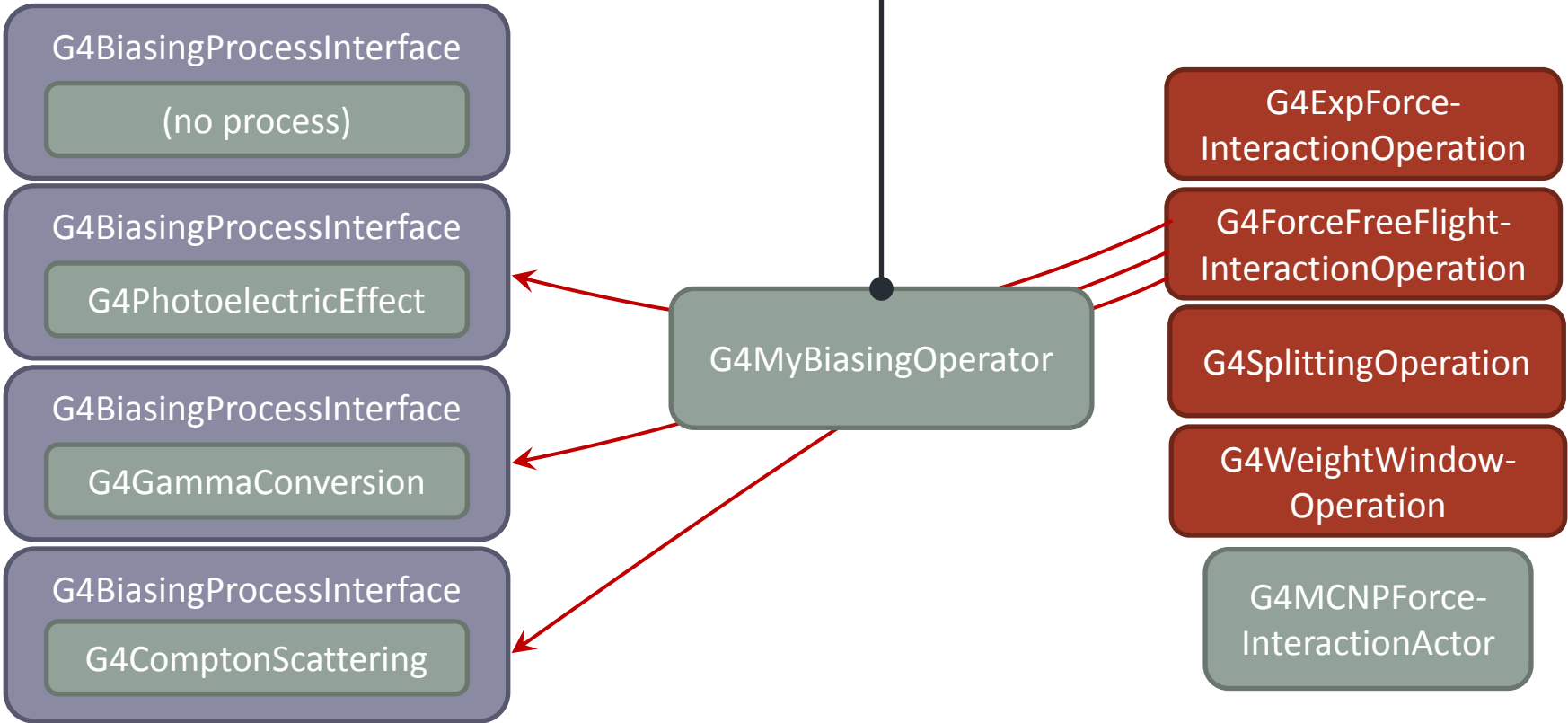
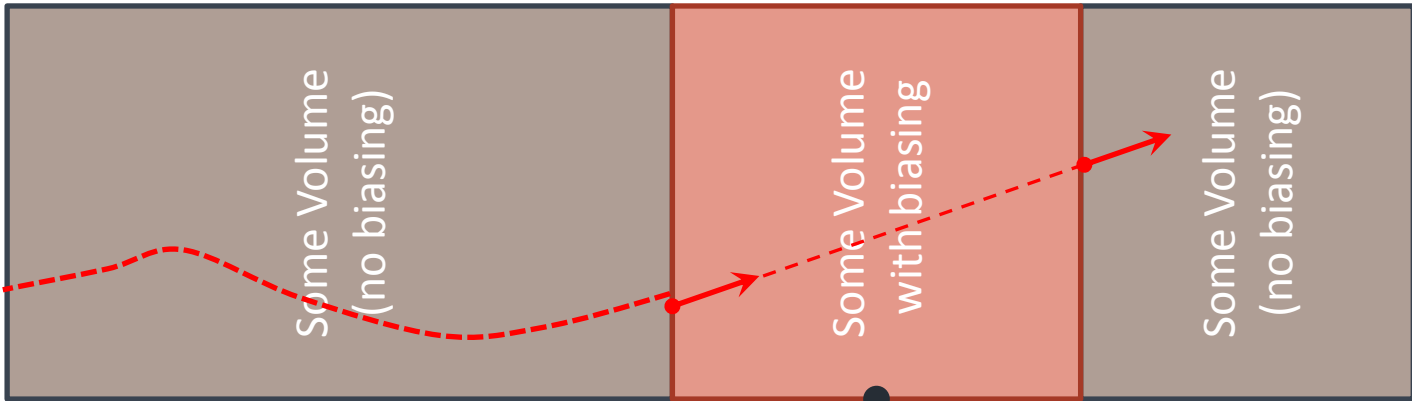
G4ExpForce-InteractionOperation

G4ForceFreeFlight-InteractionOperation

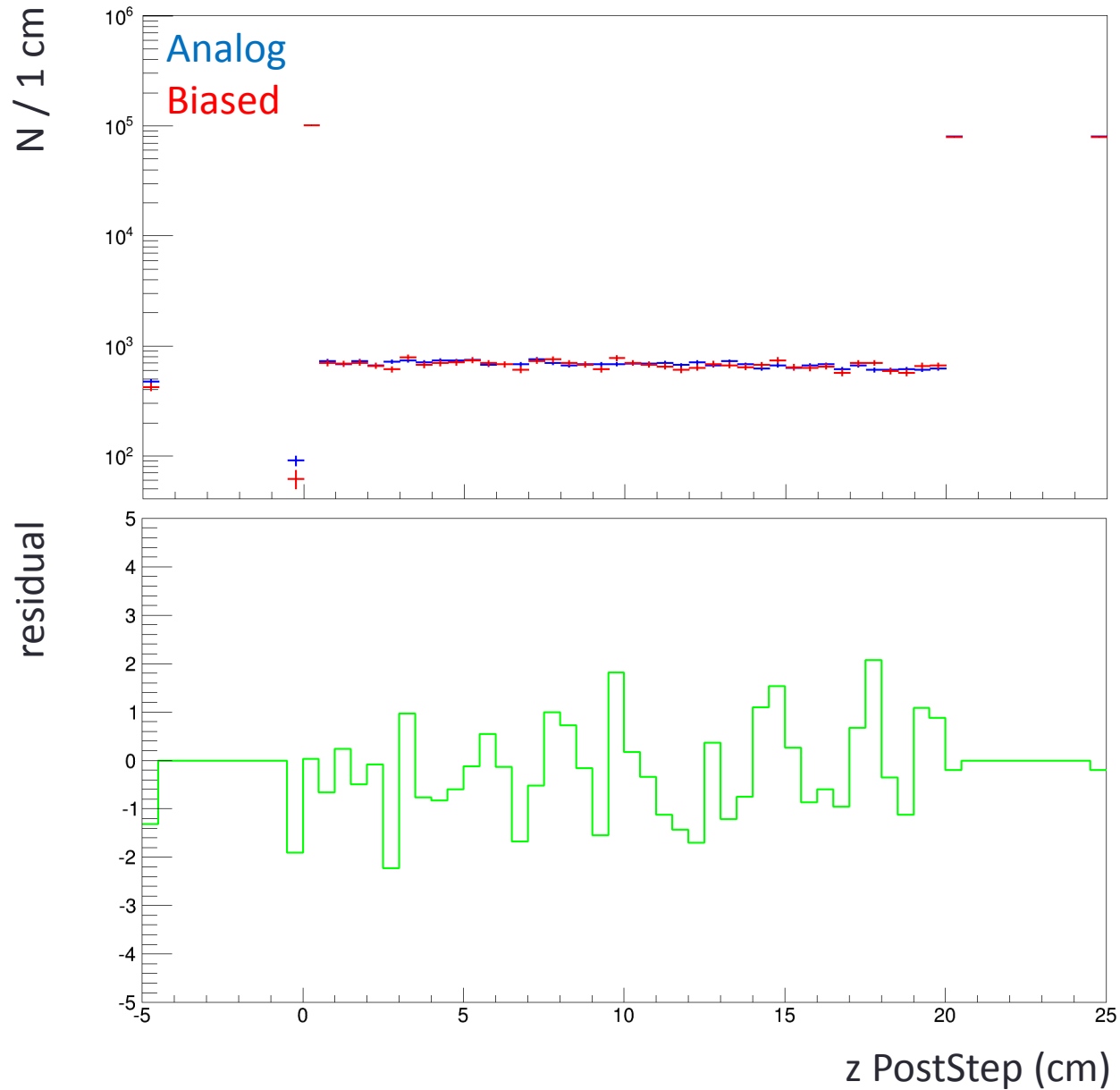
G4SplittingOperation

G4WeightWindow-Operation

G4MCNPForce-InteractionActor



z distribution of PostStep for primary



HELPER TOOLS

Tools to help configuring physics lists (1/2)

- Approach is intrusive to physics lists
 - Have to make simple the modifications needed
- **Granular level: G4BiasingHelper**
 - Insert needed processes in the G4ProcessManager of a particle
 - For a physics process: substitute the process with a wrapped version
 - Adding or wrapping the AlongStep methods
 - I.e. a PostStep process becomes a PostStep + AlongStep process
 - Caring about process ordering
 - Methods:

```
static G4bool   ActivatePhysicsBiasing(G4ProcessManager* pmanager, G4String processToBias,
                                       G4String wrappedName = "");
static void    ActivateNonPhysicsBiasing(G4ProcessManager* pmanager, G4String nonPhysProcessName = "");
```
 - Usage:
 - After a process manager has been setup, helper methods can be invoked:

```
G4BiasingHelper::ActivatePhysicsBiasing(pmanager, "compt");
G4BiasingHelper::ActivatePhysicsBiasing(pmanager, "conv");
G4BiasingHelper::ActivateNonPhysicsBiasing(pmanager);
```

Tools to help configuring physics lists (2/2)

- **Global level: G4BiasingPhysics**

- A physics constructor to be used to modify an existing physics list
 - Makes use of the G4BiasingHelper

- Example of methods:

```
void PhysicsBias(const G4String& particleName);  
void NonPhysicsBias(const G4String& particleName);  
void Bias(const G4String& particleName);
```

- Usage:

```
FTFP_BERT* physList = new FTFP_BERT;  
G4BiasingPhysics* biasPhys = new G4BiasingPhysics();  
biasPhys->Bias("gamma");  
biasPhys->Bias("neutron");  
biasPhys->Bias("kaon0L");  
physList->RegisterPhysics(biasPhys);  
runManager->SetUserInitialization(physList);
```

Analog Version

```
Idle> /particle/select e+
Idle> /particle/process/dump
G4ProcessManager: particle[e+]
[0]=== process[Transportation :Transportation] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      2:      0:      6:      0:
parameter           -1:      -1:      0:      0:      0:      0:
[1]=== process[msc :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      1:      1:      5:      1:
parameter           -1:      -1:      1:      1:      1:      1:
[2]=== process[eIoni :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      0:      2:      4:      2:
parameter           -1:      -1:      2:      2:      2:      2:
[3]=== process[eBrem :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      -1:     -1:      3:      3:
parameter           -1:      -1:      -1:     -1:      3:      3:
[4]=== process[annihil :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               0:      0:      -1:     -1:      2:      4:
parameter           5:      5:      -1:     -1:      5:      5:
[5]=== process[CoulombScat :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      -1:     -1:      1:      5:
parameter           -1:      -1:      -1:     -1:     1000:  1000:
[6]=== process[positronNuclear :Hadronic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      -1:     -1:      0:      6:
parameter           -1:      -1:      -1:     -1:     1000:  1000:
```

Wrapped Version


```
Idle> /particle/select e+
Idle> /particle/process/dump
G4ProcessManager: particle[e+]
[0]=== process[Transportation :Transportation] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      6:      0:      7:      0:
parameter           -1:      -1:      0:      0:      0:      0:
[1]=== process[biasWrapper(0) :NotDefined] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      -1:     -1:      2:      5:
parameter           -1:      -1:      -1:     -1:     1000:  1000:
[2]=== process[biasWrapper(msc) :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      5:      1:      6:      1:
parameter           -1:      -1:      1:      1:      1:      1:
[3]=== process[biasWrapper(eIoni) :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      4:      2:      5:      2:
parameter           -1:      -1:      2:      2:      2:      2:
[4]=== process[biasWrapper(eBrem) :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      3:      3:      4:      3:
parameter           -1:      -1:     1000:  1000:      3:      3:
[5]=== process[biasWrapper(annihil) :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               0:      0:      2:      4:      3:      4:
parameter           5:      5:     1000:  1000:      5:      5:
[6]=== process[biasWrapper(CoulombScat) :Electromagnetic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      1:      5:      1:      6:
parameter           -1:      -1:     1000:  1000:     1000:  1000:
[7]=== process[biasWrapper(positronNuclear) :Hadronic] Active
Ordering::          AtRest          AlongStep          PostStep
                   GetPIL/    DoIt    GetPIL/    DoIt    GetPIL/    DoIt
Ordering::
index               -1:      -1:      0:      6:      0:      7:
parameter           -1:      -1:     1000:  1000:     1000:  1000:
```

**COMPARISON WITH FLUKA AND
MCNP ?**


Tentative comparison with FLUKA and MCNP functionalities

- Compare the existing FLUKA and MCNPX functionalities with the existing and *planned* Geant4 ones, provided by previous presented development.
 - I.e. : comparison is not exactly fair with FLUKA and MCNPX
 - Need to say the above to be fair ;)
- But also, to repeat, as based on abstract classes, a non-implemented functionality can be provided by (well-advertised) user.
- Coming comparison is also an exercise to check if proposed design is able to cover FLUKA and MCNPX functionalities
 - Even if, in a first stage, all functionalities may not be concretely provided.

FLUKA / Geant4 biasing functionalities

Biasing options in FLUKA from http://www.fluka.org/content/manuals/fluka2011.manual	Options in Geant4, present or future
Leading particle biasing for electrons and photons: region dependent, below user-defined energy threshold and for <u>selected physical effects</u> . 	Will be done, and will be shared with other “leading”.
Russian Roulette and splitting at boundary crossing based on region relative importance.	Existing. Will be “re-provided” with new design, prototyped.
Region-dependent multiplicity tuning in high energy nuclear interactions.	Can be done and more general.
Region-dependent biased downscattering and non-analogue absorption of low-energy neutrons.	Can be done.
Biased decay length for increased daughter production.	Will be done. Prototyped.
Biased inelastic nuclear interaction length.	Will be done. Prototyped.
Biased interaction lengths for electron and photon electromagnetic interactions.	Will be done. Prototyped.
Biased angular distribution of decay secondary particles.	Can de done.
Region-dependent weight window in three energy ranges (and energy group dependent for low energy neutrons).	Existing. Can be “re-provided” and more general.
Bias setting according to a user-defined logics.	(need more info in FLUKA, but is actual purpose of this dev.)
User-defined neutrino direction biasing.	Can be done, easily.
User-defined step by step importance biasing.	Can be done, easily.

Simple for neutral. More difficult for charged but understood.



MCNPX / Geant4 biasing functionalities

Biasing options in MCNPX From LA-UR-03-1987, MCNP5 manual	Options in Geant4, present or future
Energy Cutoff & Time Cutoff	Existing (not considered as biasing)
Geometry Splitting with Russian Roulette	Existing. Will be “re-provided” with new design, prototyped.
Energy Splitting/Roulette and Time Splitting/Roulette	Can be done easily.
Weight Cutoff	Existing (in some way). Easy.
Weight Window	Existing. Will be “re-provided” with new design. Can be made more general.
Exponential Transform	Will be done. Prototyped.
Implicit Capture (or “Implicit capture,” “survival biasing,” and “absorption by weight reduction”)	Can be done.
Forced Collisions	Will be done. Prototyped.
Source Variable Biasing	Existing.
Point Detector Tally (?)	(not biasing ?)
DXTRAN	Planned, need more work. Doable.
Correlated Sampling	Not planned for now “à la MCNP”. But doable with user’s invest.

Conclusion

- What has been done:
 - Review of principles (importance sampling, splitting)
 - Review of formalism for GPIL biasing, and extension compared to existing packages
 - Per process modification interaction length looks doable and consistent
 - Prototype implementation of G4VBiasingOperation, G4VBiasingOperator, G4BiasingProcessInterface
 - Prototype implementation for:
 - Force interaction operation (exp. And flat laws)
 - Forced free flight operation
 - Exponential transform operation
 - (and related simple operators to allow activating these operations for the tracking)
 - Tools to help configuring the physics list simply
- What is missing:
 - Have to release this code somewhere in the distribution !
 - Discussion this afternoon
 - And related examples, documentation and *tests* (comparison biasing / analog)
 - Final state biasing not prototyped yet:
 - Brem splitting and leading particle should be “easy”
 - Can imagine a generic way to do that
 - Mode difficult is final state biasing in term of biasing distributions:
 - Not generic “differential cross-section” concept
 - Would need some explicit dependence of biasing onto physics packages or the reverse
 - » Has to be discussed
 - Facilities for “histories”:
 - I.e trajectory-like quantities, recoding weight evolution, much useful when chasing for big weights
 - Facilities to monitor convergence