Event Biasing and Fast Simulation

Dennis Wright

Geant4 Tutorial at Jefferson Lab

28 March 2024

using slides of

Marc Verderi (IN2P3/LLR), Alexei Sytov (INFN, KISTI) and
Anna Zaborowska (CERN),

Outline

- Speeding up the simulation
- Event biasing overview
- Event biasing methods
- Fast simulation in Geant4
- Fast simulation interfaces and applications

Speeding up the Simulation

- Nearly ubiquitous requirement for faster simulation
  - greater statistics
  - more efficient use of computing resources

- Can achieve this by biasing events
  - bias events which are rare or more interesting
  - many methods
  - big speed-up factors possible

- Or using fast simulation
  - approximate the physics
  - use parallel geometries and parallel processing
  - use hardware acceleration

Event Biasing

What is Event Biasing ?

(aka variance reduction) simulates rare events efficiently

Accelerated simulation of these events only - not all

Focuses on what we want to tally

Large CPU improvements (orders of magnitude depending on the problem)

"Biasing" refers to biased simulation

Biased because region of phase space contributing to tally is enhanced compared to analog

This enhancement comes with the computation of statistical "weights" that are connected back to analog quantities (e.g. track i, weight i)

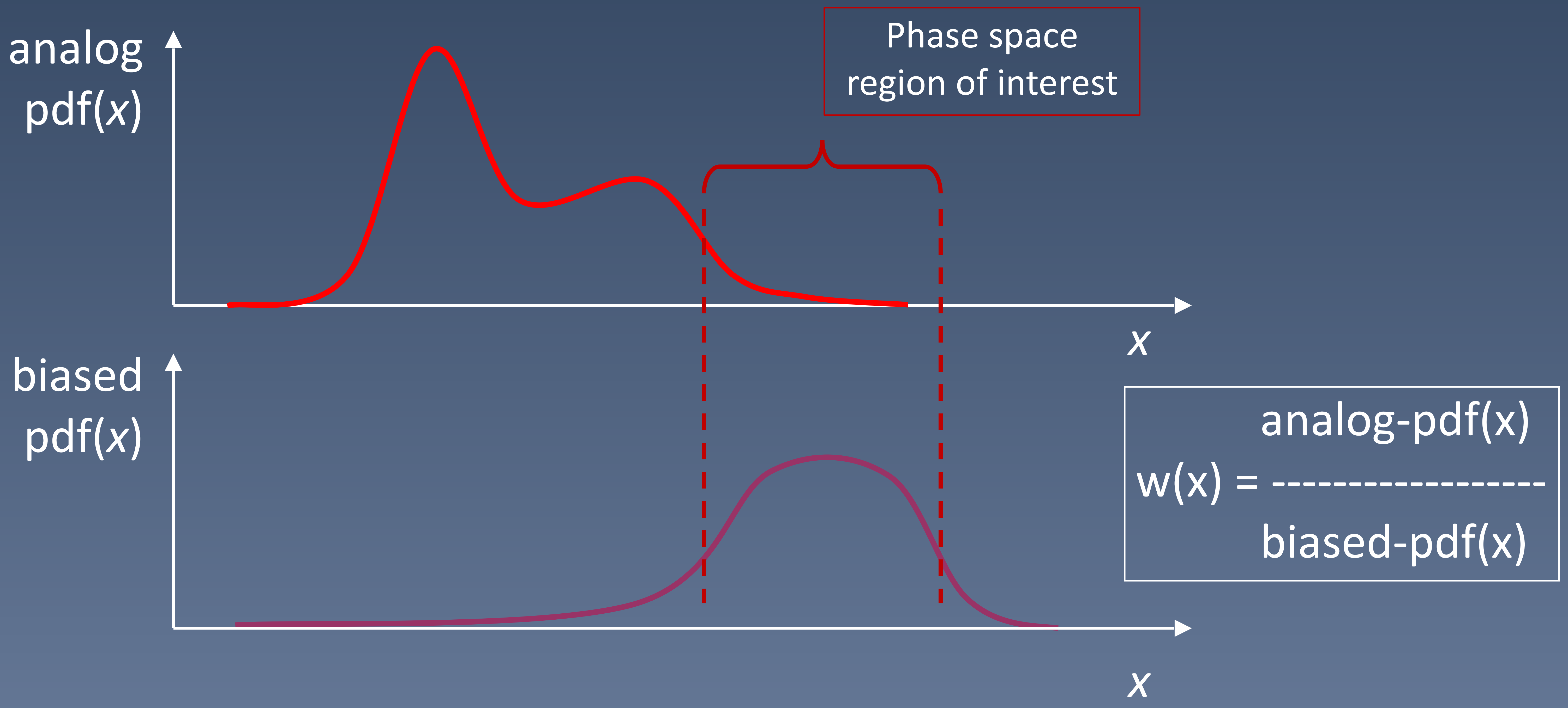
Many biasing techniques with different names, but two are the backbone of most of them

Importance sampling

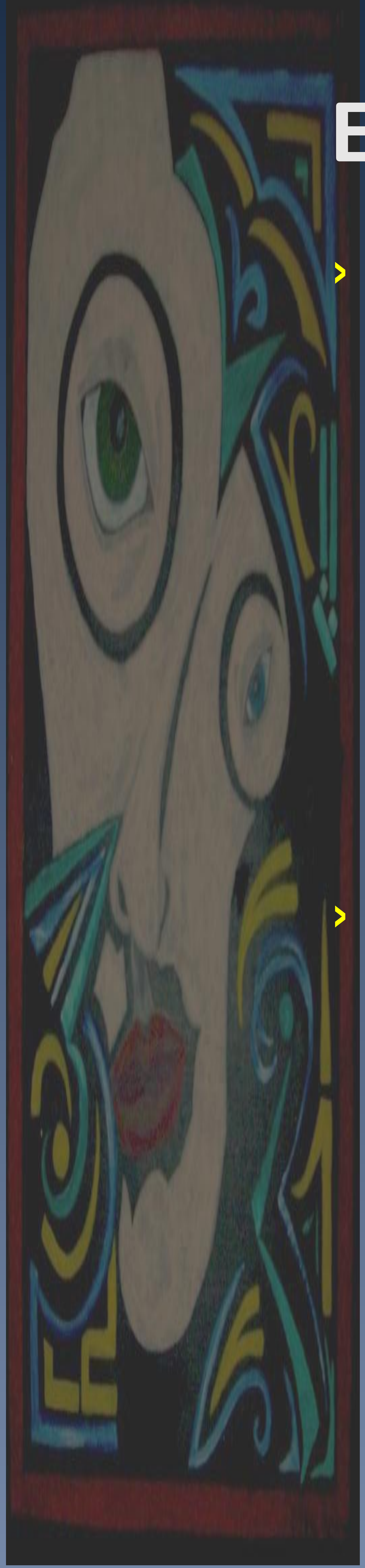
Splitting

Importance Sampling

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

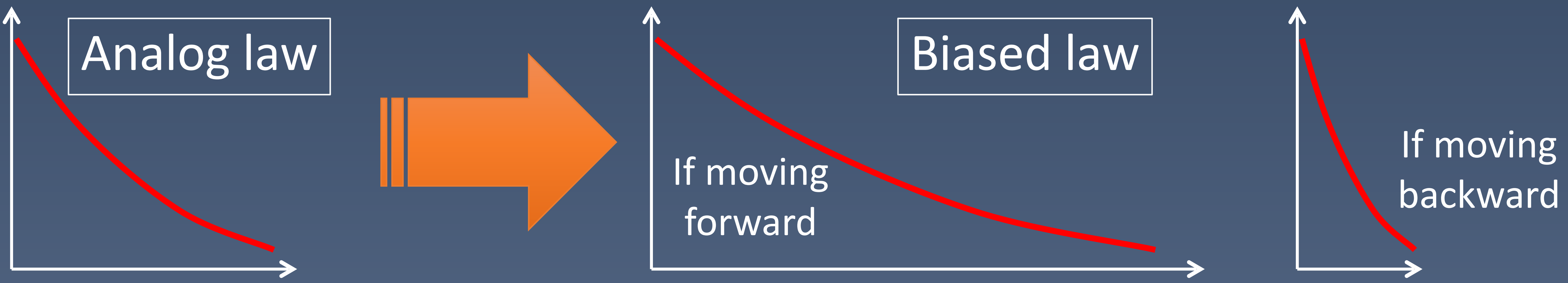


- The weight, for a given value x, is then the ratio of the analog over the biased distribution values at x.

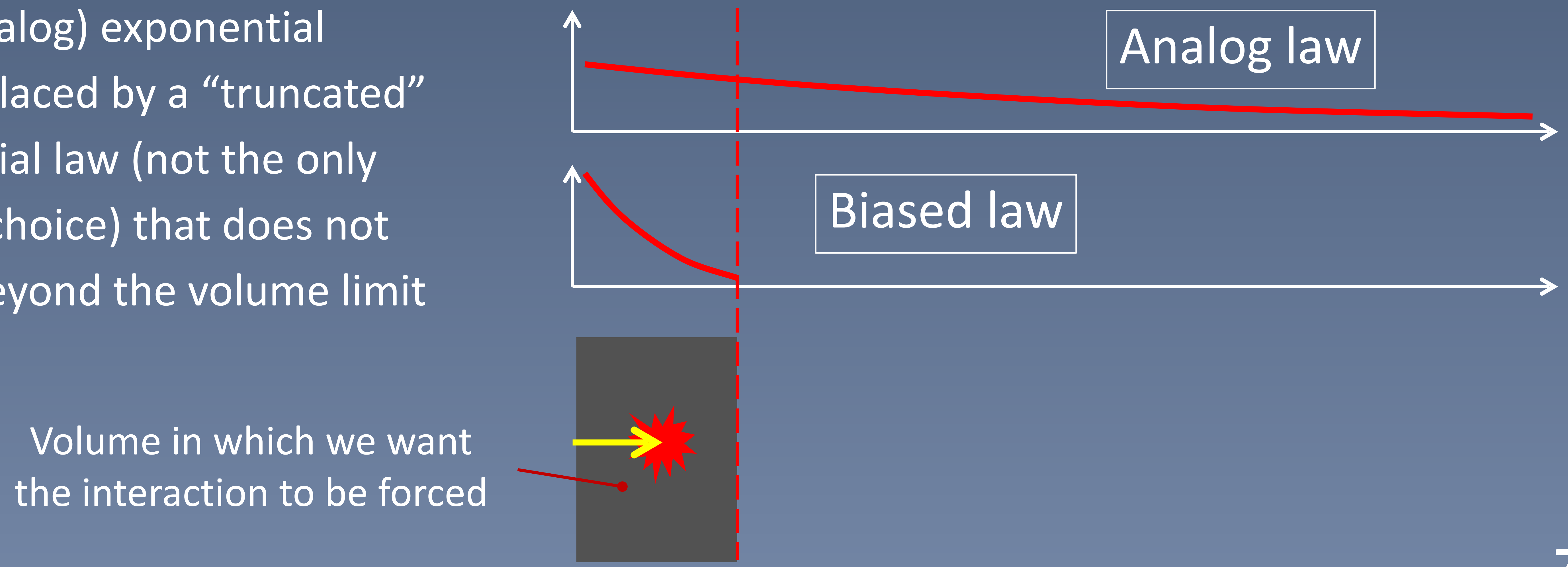


Examples of importance sampling techniques

- The “exponential transform” technique
  - changes the total cross section
  - often made direction dependent
  - usual (analog) exponential law is replaced by an other law (still an exponential one here, but with different cross-section)

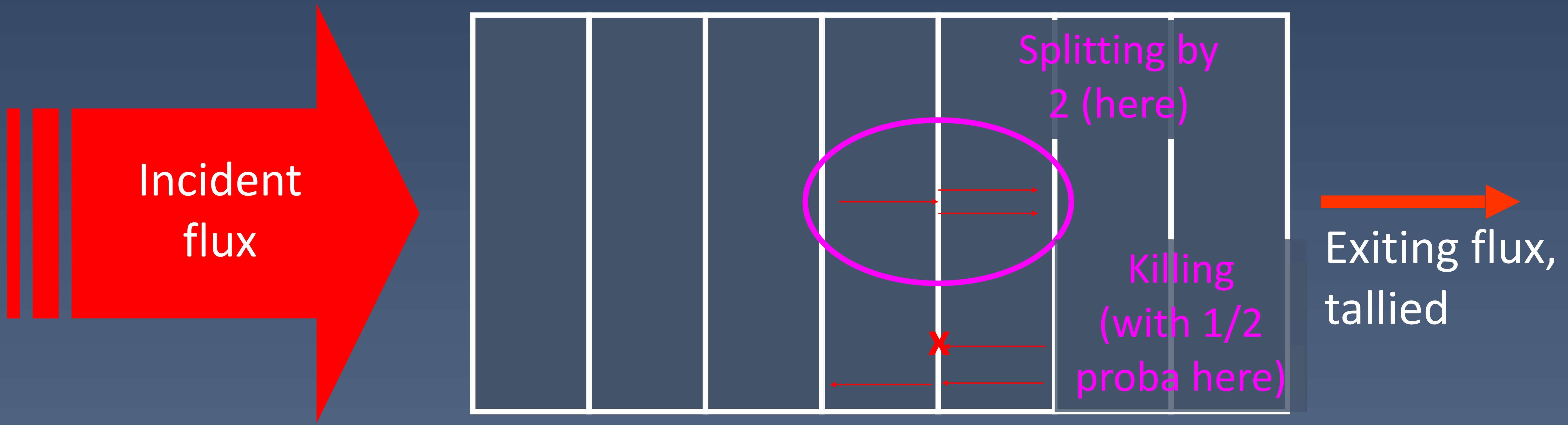


- Forcing interaction in thin volume
  - usual (analog) exponential law is replaced by a “truncated” exponential law (not the only possible choice) that does not extend beyond the volume limit



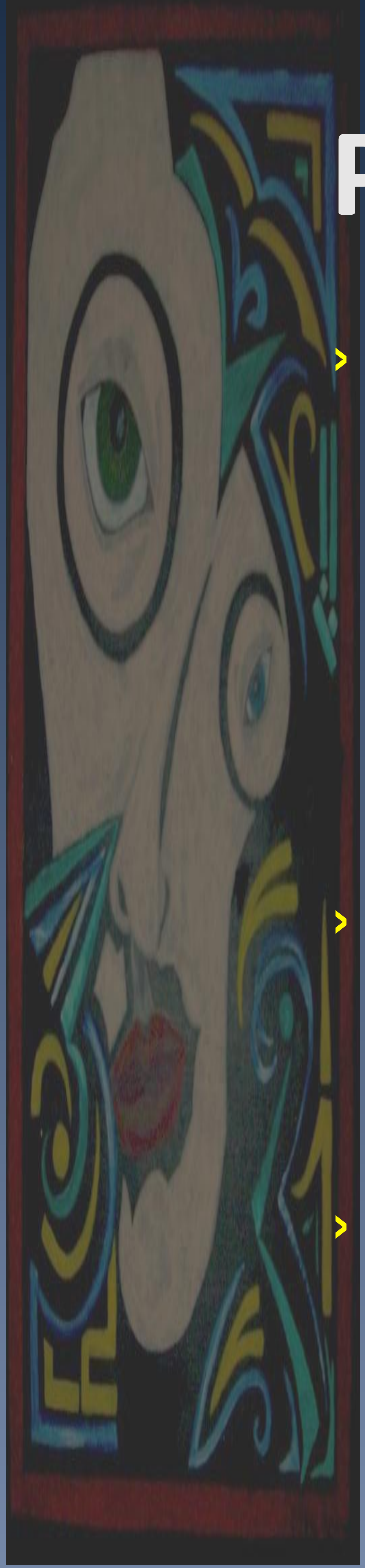
Splitting

- Physical distributions are unchanged, but a “splitting” occurs in some regions when particles go towards the tally



  - In above example, particles moving toward exit are cloned
    - cloning compensates for the physical absorption in the shield material
  - When cloning happens, clones receive a weight ½ (here) of the initial track
- Splitting goes along with killing / Russian Roulette
  - For particles going the opposite direction
  - In above example, particles moving backward are killed with a probability ½ (e.g.) and, if it survives, its weight is multiplied by 2 (e.g.).

Event Biasing Methods



Primary Particle Biasing

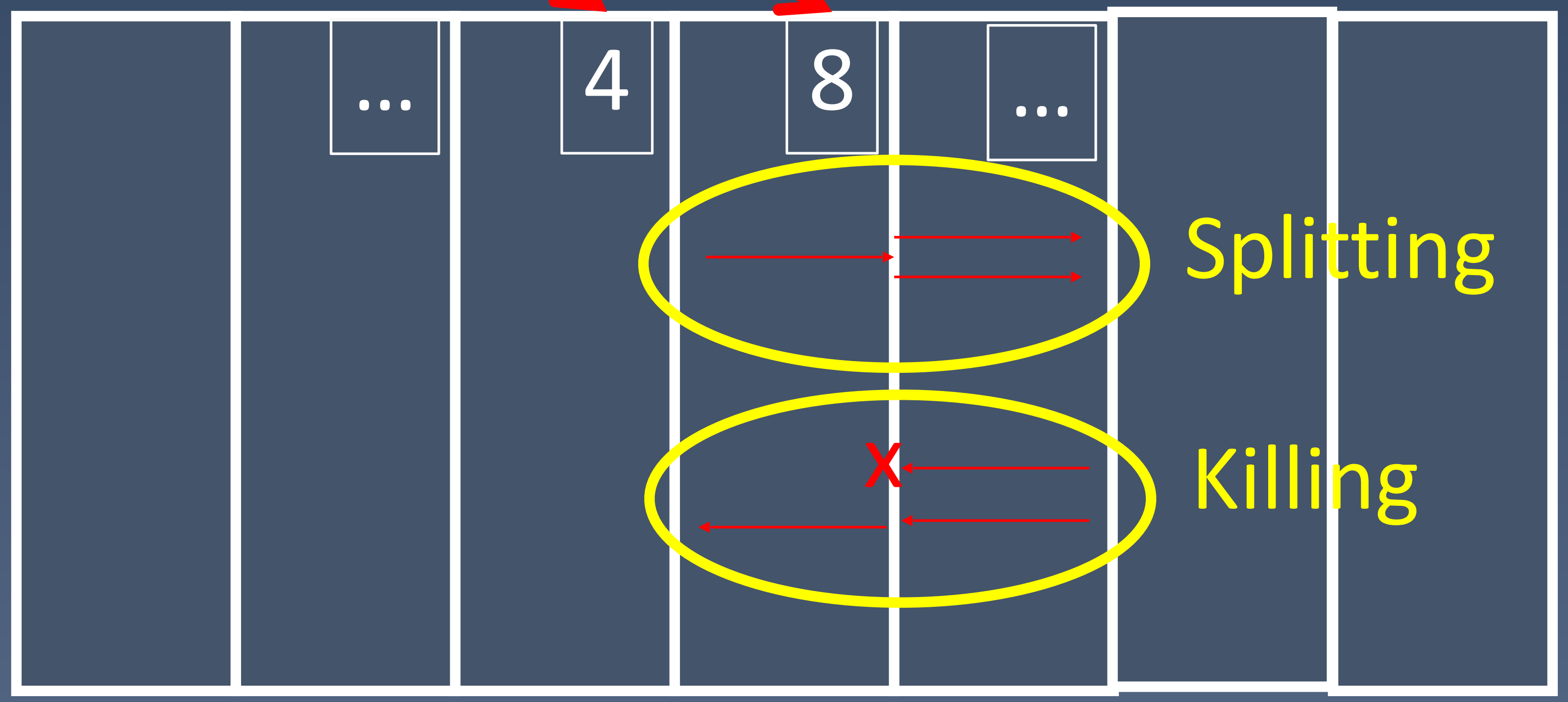
- The General Particle Source (GPS) allows particle source to be biased
  - position
  - angular distribution
  - energy distribution

- This is an “importance sampling” biasing
  - physical distributions are replaced by biased ones

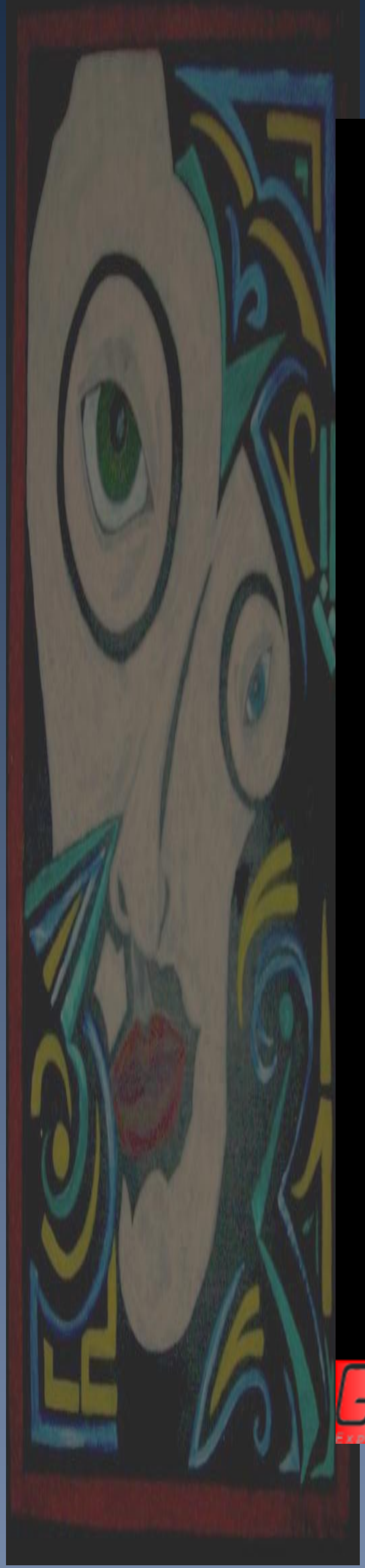
- Primary particles are given a weight that is the ratio of analog / biased probabilities
  - in simulation, all daughter, granddaughter,… particles from the primary inherit the weight of the primary

Geometry-based Importance Biasing

- Attach "importance" to cells in geometry
  - Not to be confused with "importance sampling" : ie change of probability density functions of interaction laws

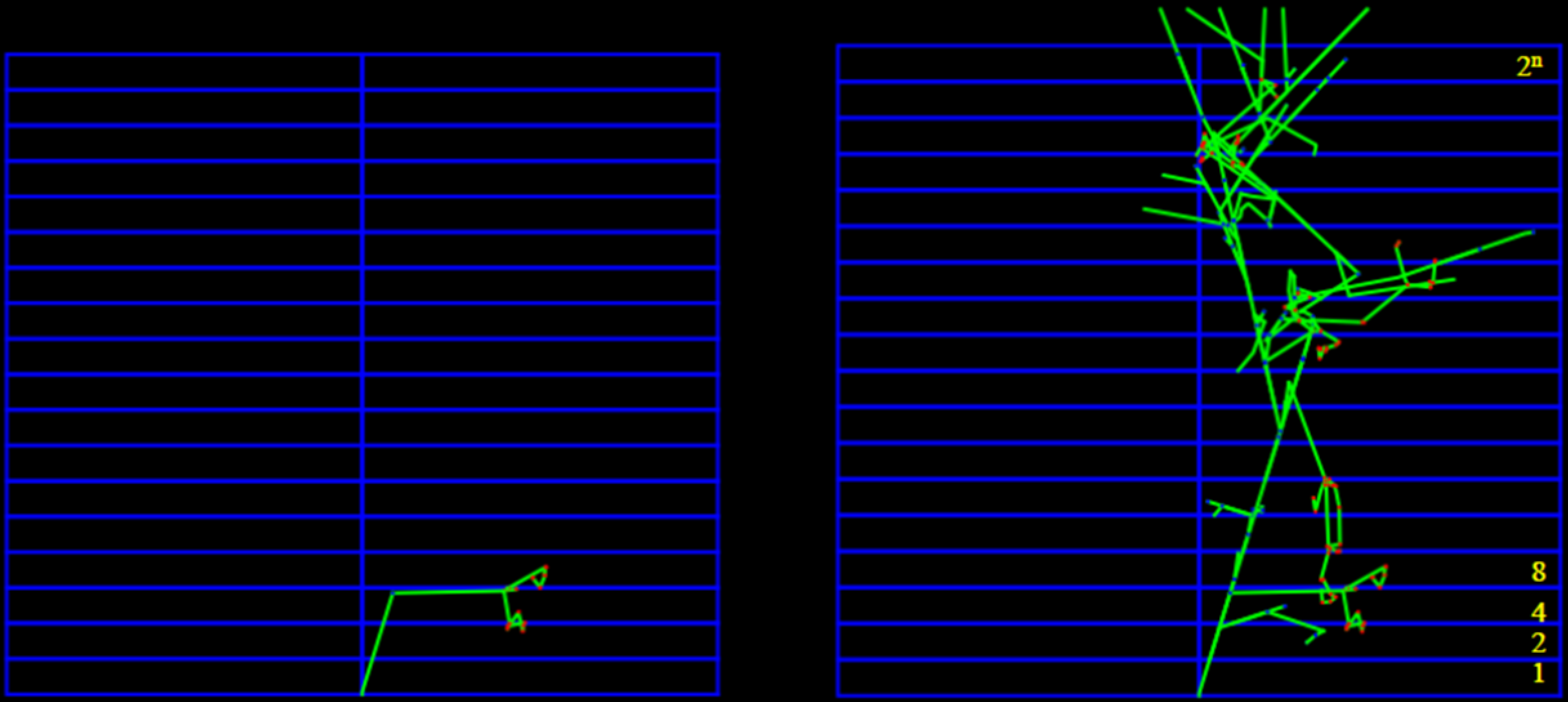


  - Applies splitting if the track moves forward
    - with splitting factor 8/4 = 2, if track goes from « 4 » to « 8 » (e.g.)
    - each copy having a weight = ½ of the incoming track
  - Applies killing if the track moves the other way
    - it is killed with a probability ½
    - If particle survives, its weight is multiplied by 2 (e.g.)



Example B01 - 10 MeV neutrons, thick concrete cylinder

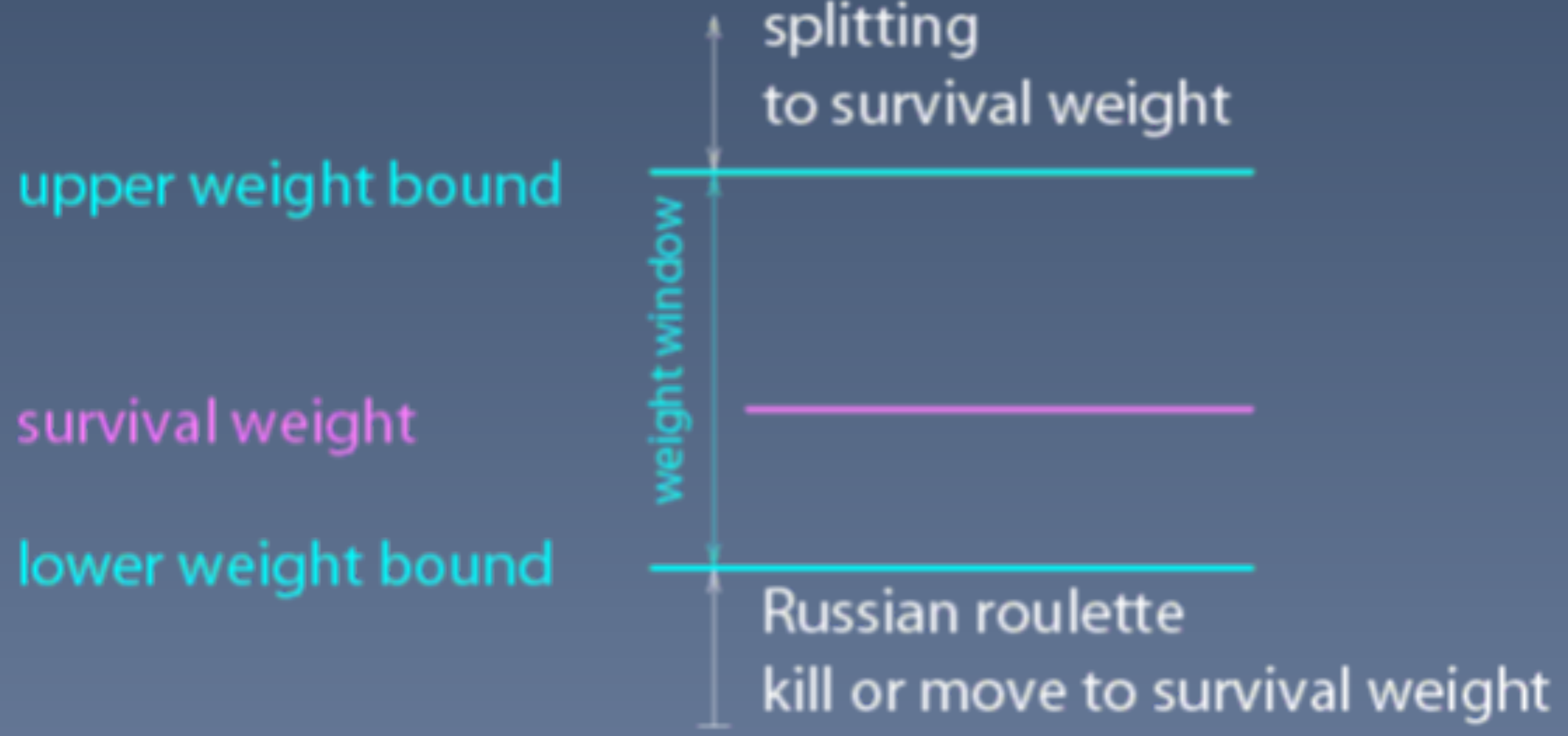
Analogue Simulation | Importance Sampled



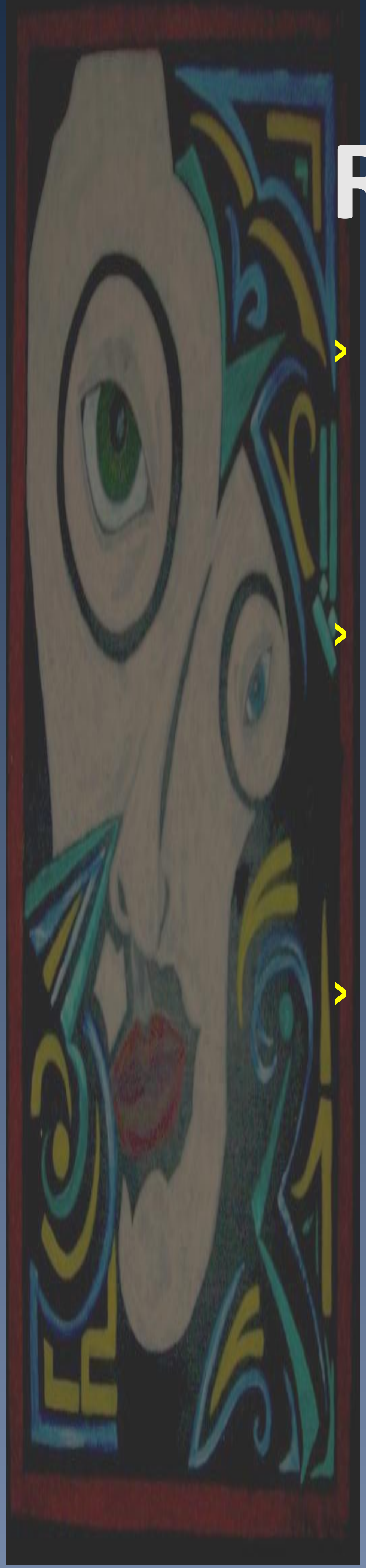
See: geant4/examples/extended/biasing/B01

Weight Window Technique

- This is a splitting & killing technique, to avoid having
  - excessively high weight tracks at some point
    - makes the convergence of the estimated quantities difficult
    - tracks with weight above some value are split
  - excessively low weight tracks
    - don’t waste time tracking them
    - tracks below some value are killed using Russian roulette

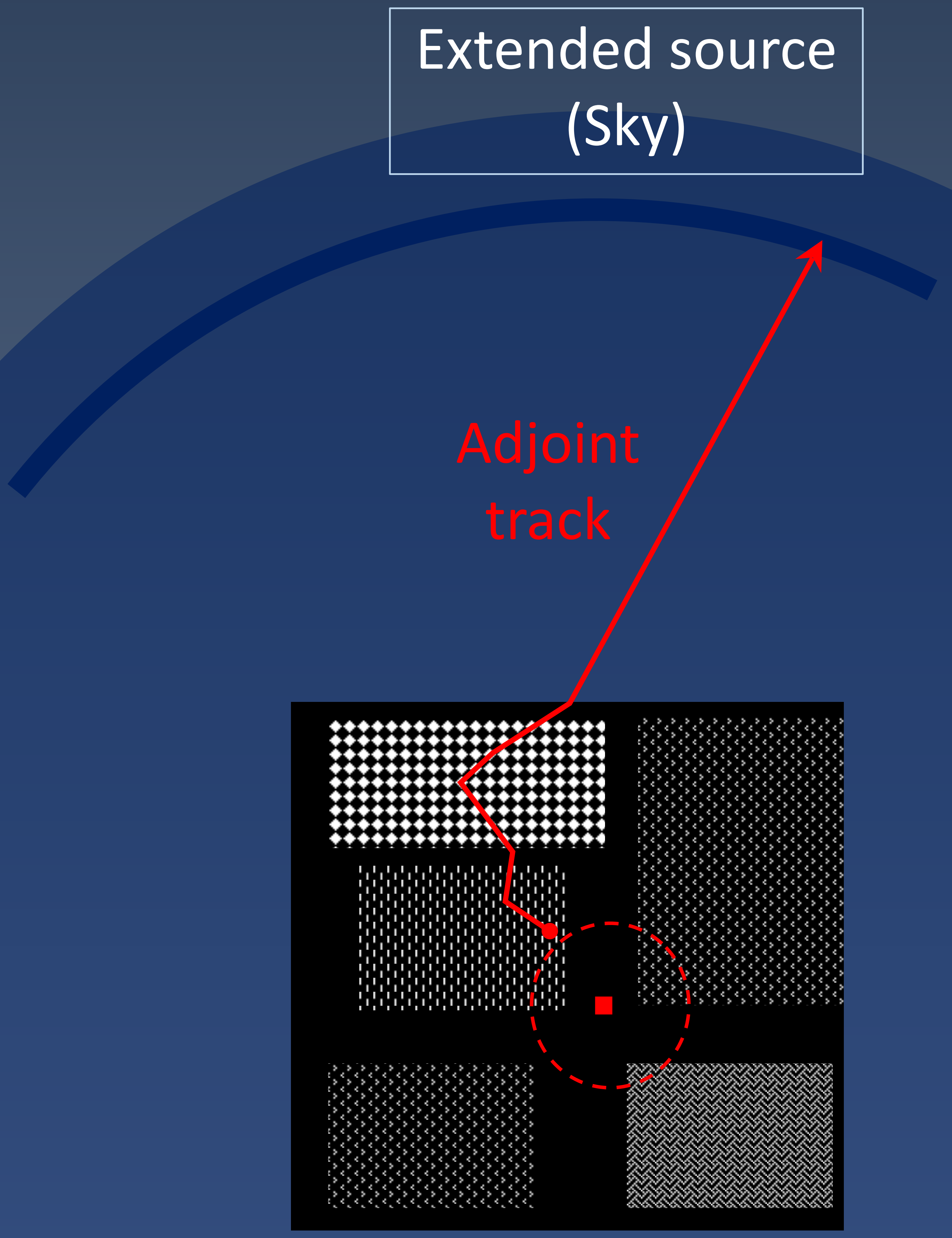


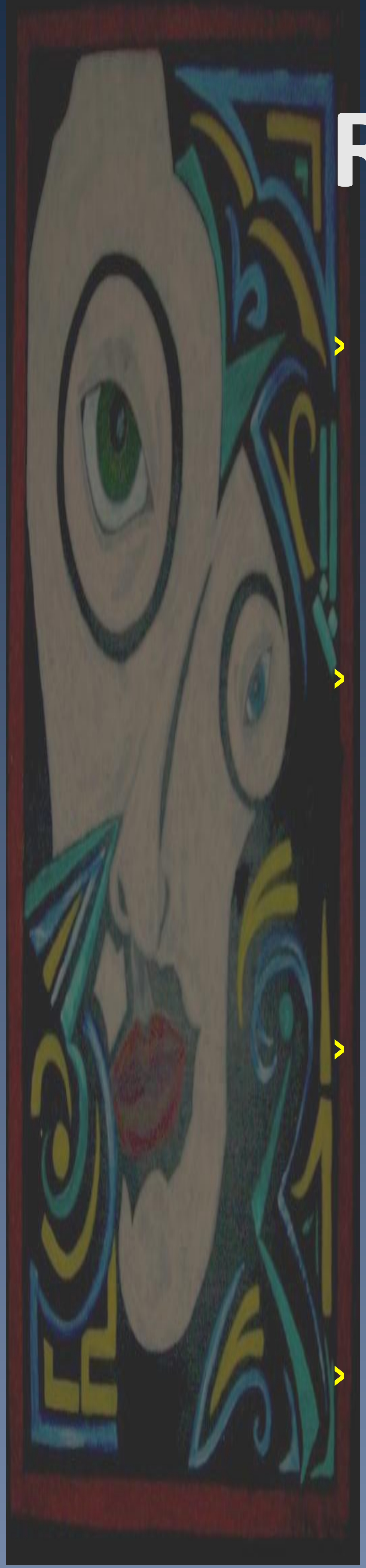
- As with importance, this is configured per cell
  - can also be configured per energy
- See: geant4/examples/extended/biasing/B01



Reverse Monte Carlo

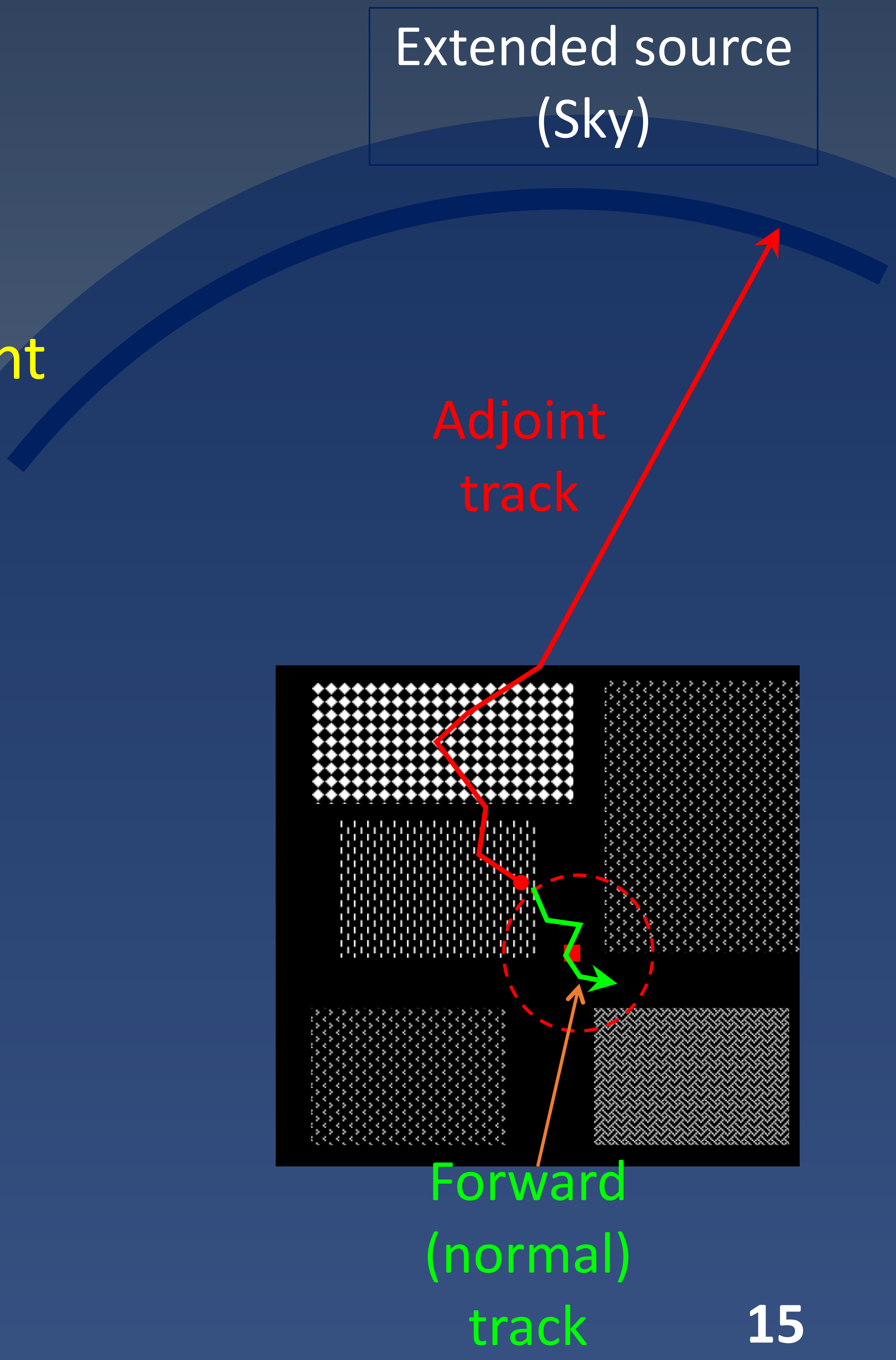
- Simulation in which particles go back in time !
- Useful in case of small device to be simulated in a big environment
  - requested by ESA, for dose study in electronic chips in satellites
- Simulation starts with the red track
  - so-called “adjoint” or backward track
  - goes back in time
  - increasing in energy
  - travels until reaching the extended source





Reverse Monte Carlo

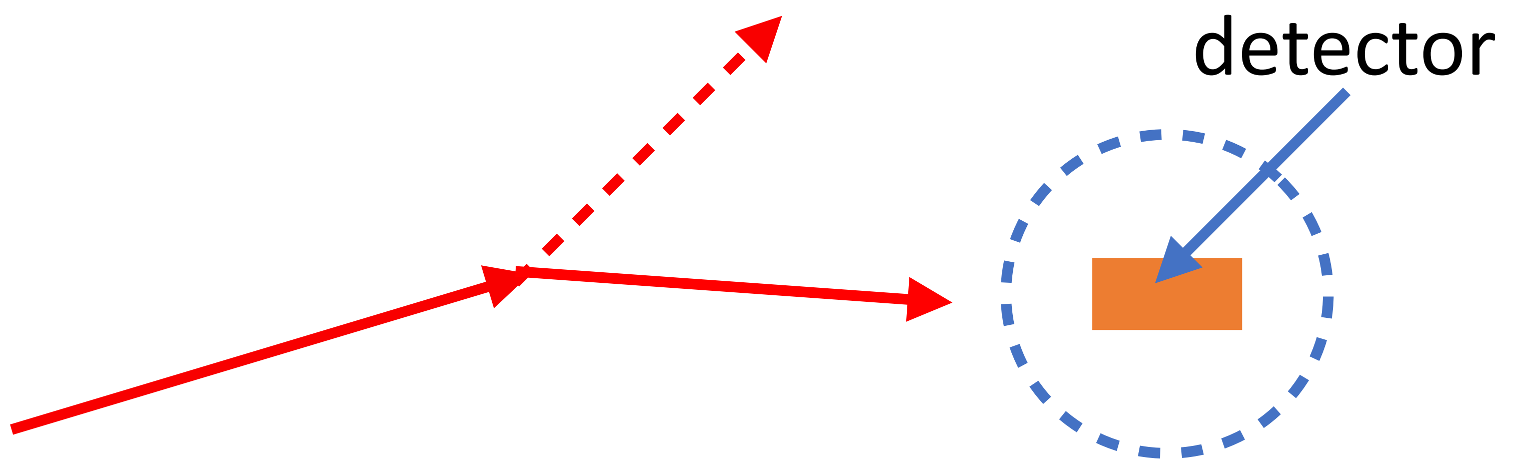
- If track energy < high energy limit of the source spectrum
  - proceed with usual “forward” simulation (green track)
- After simulating many tracks, adjust adjoint track weights so that energy spectrum of adjoint particle matches source spectrum
  - provides the weight of the green tracks
- Dose in volume can then be computed, accounting for the forward track contribution, with the proper weight
- See: geant4/examples/extended/biasing/ReverseMC01



Building Block Biasing

- Many biasing options available, but they
  - don't allow changing the PDF of processes
  - don't make it easy to change final state behavior of a process
  - don't make it easy to mix biasing options

- Want options like
  - exponential transform: $p(l) = \sigma e^{-\sigma l} \rightarrow p'(l) = \sigma' e^{-\sigma' l}$
    - change total cross section
    - add direction dependence
  - forced interaction
    - guarantee interaction in thin volumes
  - forced flight (towards detector)
    - also called DXTRAN
    - force scattering towards detector
  - and so on

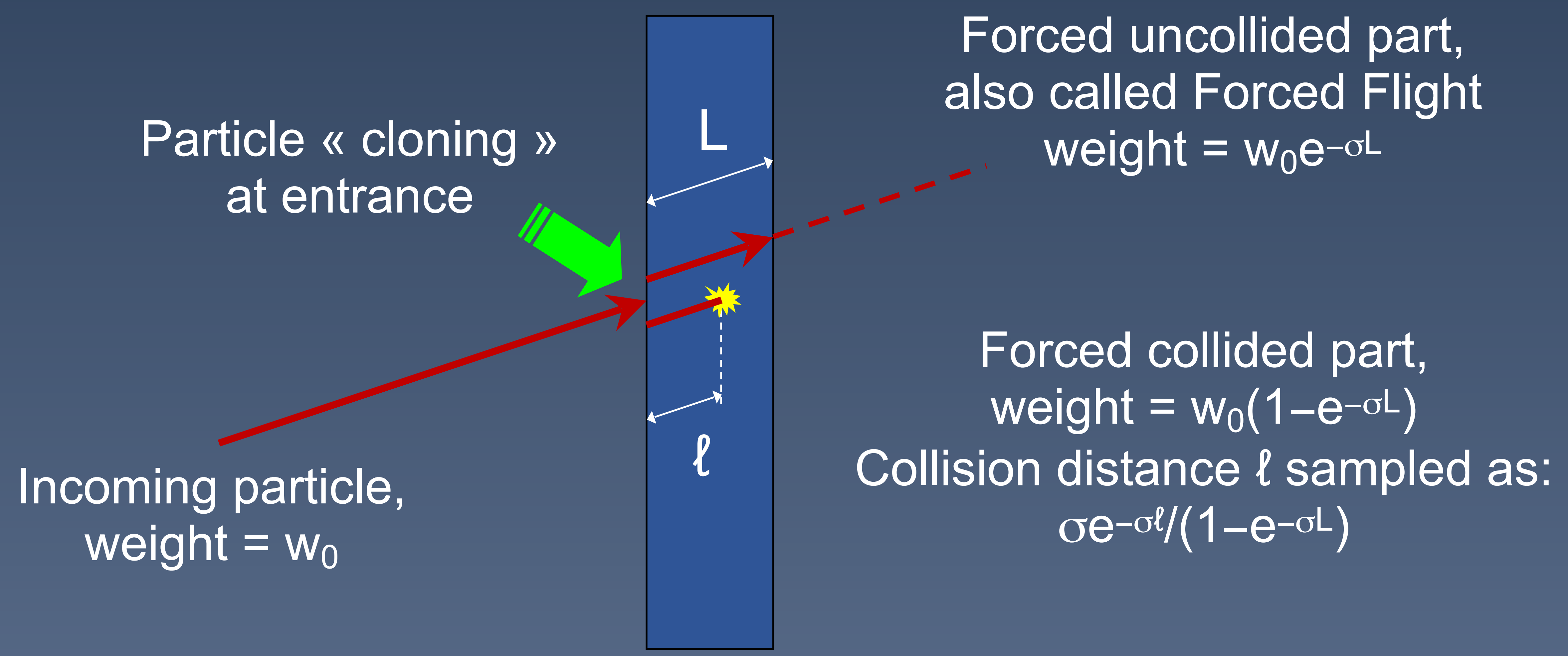


- Also to allow configurable logic rather than built-in functionalities

Building Blocks vs. Built-in

- Example of "forced collision" scheme à la MCNP

Thin detector volume



- 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)

Using Building Block Biasing

- Wrap the process you want to bias (e.g. G4GammaConversion, G4ComptonScattering) in G4BiasingProcessInterface
  - it intercepts the calls for distance to interaction and final state generation from the wrapped process
  - can be any number of processes
- Add a biased physics constructor to a physics list: in main()

```
// Select a physics list and augment it with biasing facilities
FTFP_BERT* physicsList = new FTFP_BERT;
auto biasingPhysics = new G4GenericBiasingPhysics();

// Select processes to be biased
std::vector<G4String> gammaProcessesToBias, electronProcessesToBias, positronProcessesToBias, .....  ;

gammaProcessesToBias.push_back("conv");
gammaProcessesToBias.push_back("photonNuclear");
...
...
// Assign particles
biasingPhysics->PhysicsBias("gamma", gammaProcessesToBias);
biasingPhysics->PhysicsBias("e-", electronProcessesToBias);
...
...

// Register physics constructor and initialize
physicsList->RegisterPhysics(biasingPhysics);
runManager->SetUserInitialization(physicsList);
```

- Then, in ConstructSDandField() method of detector construction:
-

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

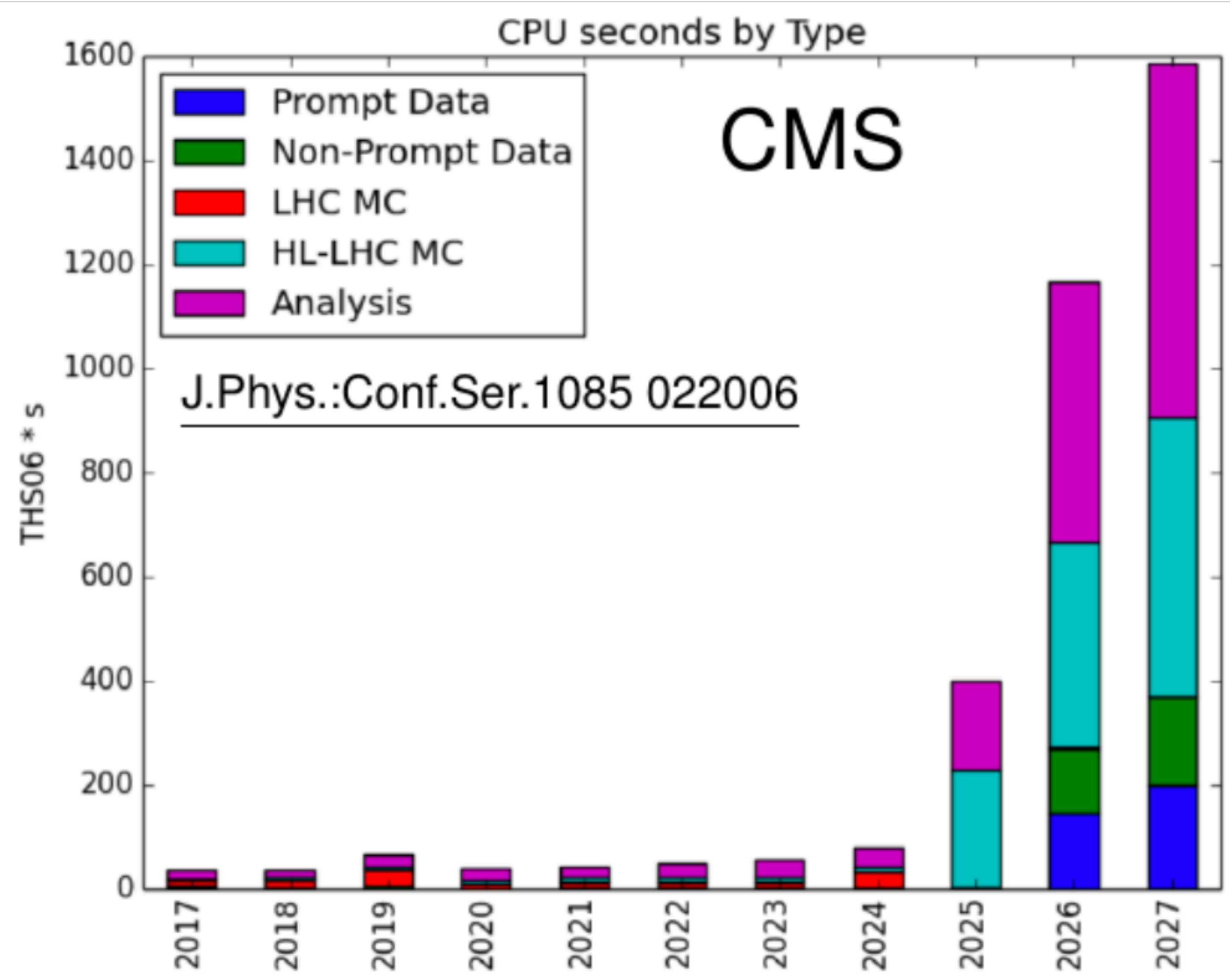
Using Building Block Biasing

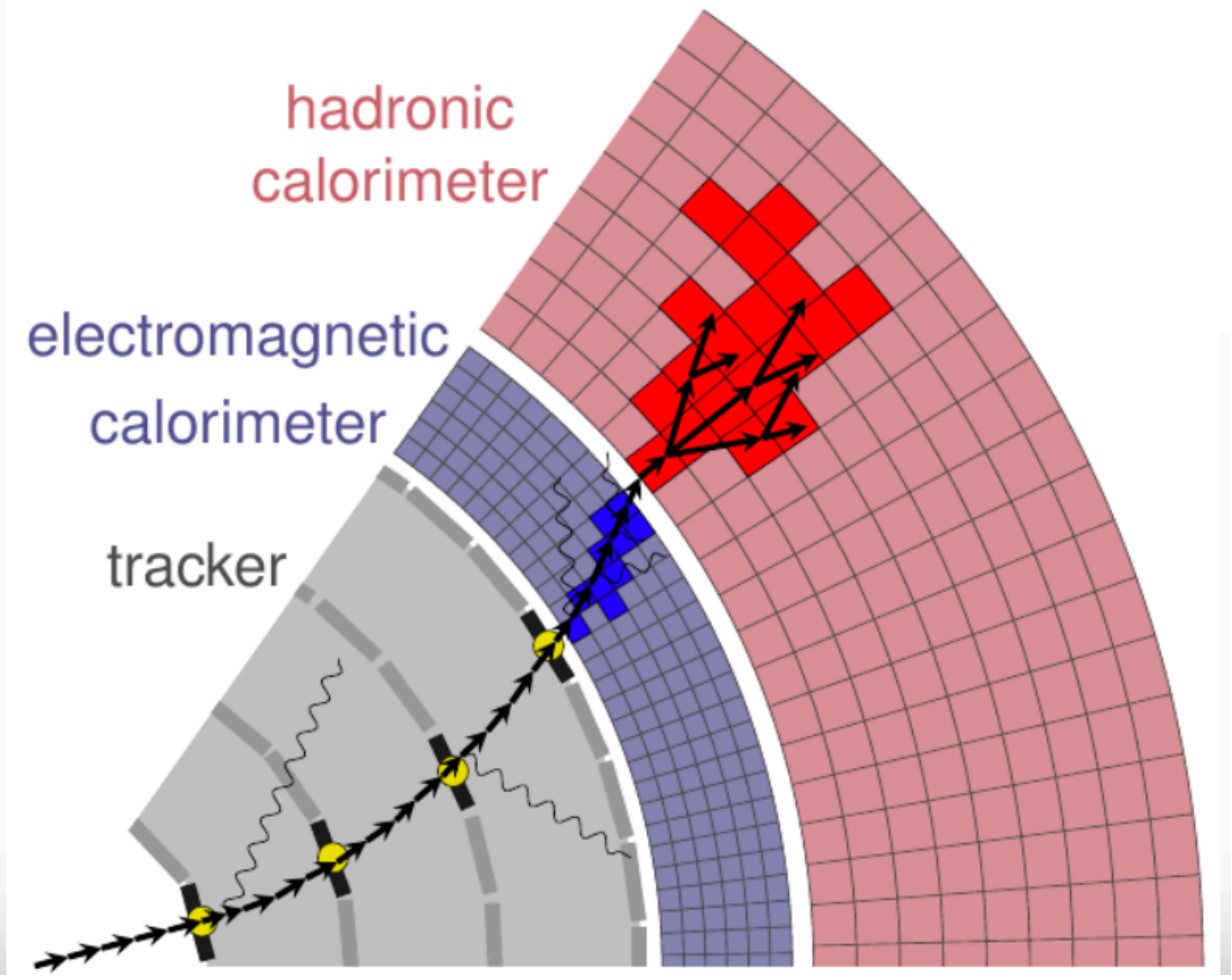
- G4VBiasingOperation and G4VBiasingOperator are base classes meant for extending functionality
  - derive MyBiasingOperation, MyBiasingOperator from these
  - writing such classes is generally demanding
- Relatively new development in Geant4
  - enrichment and evolution expected
- Examples
  - geant4/examples/extended/biasing/GB01 : individual process cross section biasing
  - geant4/examples/extended/biasing/GB02 : forced collision as in MCNP
  - GB03 : geometry-based biasing
  - GB04 : bremsstrahlung splitting
  - GB05 : splitting by cross section (for neutrons)
  - GB06 : biasing using parallel geometries
  - GB07 : leading particle biasing

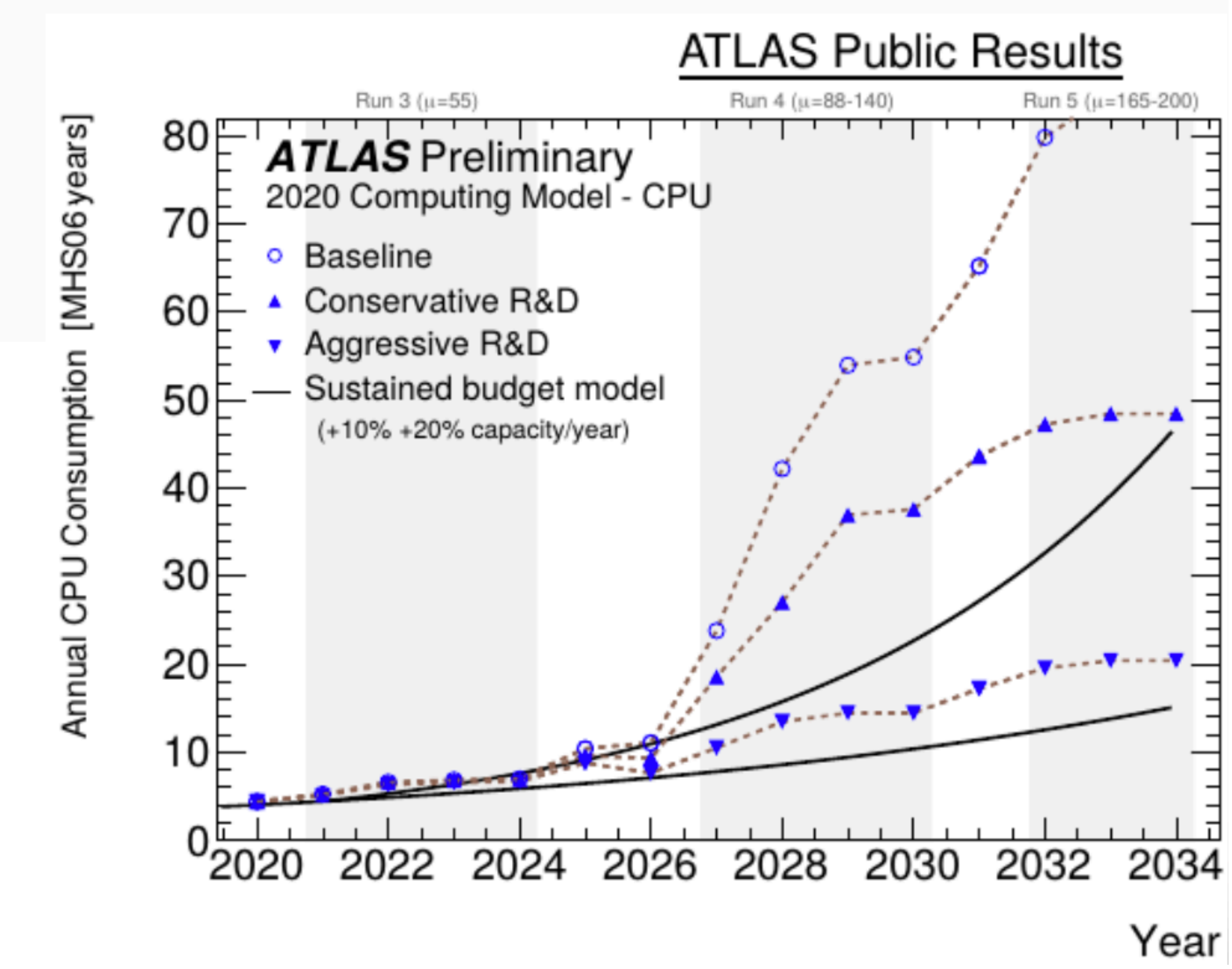
Fast Simulation in Geant4

Why Fast Simulation?

- More simulated data per CPU time
- More simulation (statistics) and data analysis will be required in future experiments
- Economy of simulation resources
- Limit power consumption







Fast Simulation in Geant4

- A trade-off between simulation time and accuracy
- Not always necessary or interesting to simulate every track, step and interaction

- In certain regions, for certain processes, simulate only
  - the summed or average behavior (as in EM showers) -> parameterization
  - the most important (or leading) particles (as in hadronic showers) -> biasing

- Once invoked, fast simulation
  - stops the standard Geant4 process
  - replaces it with and approximate, fast process
  - resumes the original process when fast process is complete

- Activated only
  - in a particular G4Region
  - under certain conditions
  - for certain particle types (G4VProcess::IsApplicable)

Parameterization

- Net effect of a large number of tracks may be described on average by a few parameters
  - EM showers:
    - depth and width of shower in a particular material vary predictably with incident particle energy -> describe them with a function
    - detector response may depend only on the energy deposit -> can ignore detailed geometry

- Could also parameterize the net effect of many steps in a single track
  - already a central part of some processes like multiple scattering
  - can generalize to other applications

- Which processes can be parameterized?
  - depends on physics, particle type and detector geometry
  - EM showers in calorimeters of sufficient depth : yes
  - hadronic showers : mostly no

- A parameterized shower process must:
  - determine how much energy is deposited and where
  - decide what to do with the incident particle (move it, kill it, …)
  - determine whether or not secondaries will be created

Biasing

- Some types of particles produced in processes can be ignored, deferred or de-emphasized
  - particles below an energy threshold or outside an angular range
  - long-lived particles (e.g,. neutrons) killed or queued for later processing

- Which processes can be biased?
  - those which can’t be parameterized
  - those which produce a large number of secondaries per interaction (hadronic)
  - those which produce many different types of particles (leptons + hadrons + gammas)
  - discrete processes

- A biased shower process must :
  - select which particles in the final state should be kept (e.g. leading particles), which removed or deferred
  - rebalance the 4-momenta of the surviving secondaries

Fast Simulation Interfaces

Define G4Region in DetectorConstruction

- Add to DetectorConstruction::Construct()

```
//my volume
G4Box* MySolid = new G4Box("MyBox",SizeX/2,SizeY/2,SizeZ/2.);
G4LogicalVolume* MyVolumeLogic = new G4LogicalVolume(MySolid,MyMaterial,"MyVolume");
new G4PVPlacement(Rotation,Position,MyVolumeLogic,"MyVolume",logicWorld,false,0);

//my region (necessary for the FastSim model)
fRegion = new G4Region("MyRegion");
fRegion->AddRootLogicalVolume(MyVolumeLogic);
```

- Add to DetectorConstruction::ConstructSDandField()

```
void DetectorConstruction::ConstructSDandField()
{
    // --------------- fast simulation -------------------------
    //extract the region of the crystal from the store
    G4RegionStore* regionStore = G4RegionStore::GetInstance();
    G4Region* MyRegion = regionStore->GetRegion("MyRegion");

    //create the channeling model for this region
    MyFastSimModel* MyModel = new MyFastSimModel("ChannelingModel",MyRegion);

    //some options of the model...
```

Create Your Own Fast Simulation Model

- MyFastSimModel.hh

```
class MyFastSimModel : public G4VFastSimulationModel
{
public:
  //-------------------------
  // Constructor, destructor
  //-------------------------
  MyFastSimModel (G4String, G4Region*);
  MyFastSimModel (G4String);
  ~MyFastSimModel ();

  //------------------------------
  // Virtual methods of the base
  // class to be coded by the user
  //------------------------------

  // -- IsApplicable
  virtual G4bool IsApplicable(const G4ParticleDefinition&);
  // -- ModelTrigger
  virtual G4bool ModelTrigger(const G4FastTrack &);
  // -- User method DoIt
  virtual void DoIt(const G4FastTrack&, G4FastStep&);
```

Create Your Own Fast Simulation Model

- MyFastSimModel.cc

```
MyFastSimModel::MyFastSimModel(G4String modelName, G4Region* envelope)
: G4VFastSimulationModel(modelName, envelope)
{
}

//....oooOO0OOooo........oooOO0OOooo........oo(                .....oooOO0OOooo

MyFastSimModel::MyFastSimModel(G4String modelName)
: G4VFastSimulationModel(modelName)
{
}

//....oooOO0OOooo........oooOO0OOooo......

MyFastSimModel::~ MyFastSimModel()
{
}
```

MyFastSimModel.cc : Particles and Conditions

```
G4bool MyFastSimModel::IsApplicable(const G4ParticleDefinition& particleType)
{
  return
    &particleType == G4Electron::ElectronDefinition() ||
    &particleType == G4Positron::PositronDefinition() ||
    &particleType == G4Gamma::GammaDefinition();
    //& my particles ...
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo...

G4bool MyFastSimModel::ModelTrigger(const G4FastTrack& fastTrack)
{
  //my code:
  //...
  return MyCondition;
}
```

MyFastSimModel.cc : Model Implementation

```
void MyFastSimModel::DoIt(const G4FastTrack& fastTrack,
                          G4FastStep& fastStep)
{

  //get some necessary information
  G4double Etotal = fastTrack.GetPrimaryTrack()->GetTotalEnergy();
  G4double mass = fastTrack.GetPrimaryTrack()->GetParticleDefinition()->GetPDGMass();
  G4double charge = fastTrack.GetPrimaryTrack()->GetParticleDefinition()->GetPDGCharge();
  G4ThreeVector MomentumDirection=fastTrack.GetPrimaryTrackLocalDirection();
  G4ThreeVector xyz = fastTrack.GetPrimaryTrackLocalPosition();
  G4double TGlobal = fastTrack.GetPrimaryTrack()->GetGlobalTime();
  //fastTrack.Get...

  //do very important simulations

  //my code ...

  //set new parameters:

  //set global time
  fastStep.ProposePrimaryTrackFinalTime(TGlobal);
  //set final position
  fastStep.ProposePrimaryTrackFinalPosition(xyz);
  //set final kinetic energy
  fastStep.ProposePrimaryTrackFinalKineticEnergy(Etotal-
          fastTrack.GetPrimaryTrack()->GetParticleDefinition()->GetPDGMass());
  //set final momentum direction
  fastStep.ProposePrimaryTrackFinalMomentumDirection(MomentumDirection);

  //kill a primary particle if necessary
  fastStep.KillPrimaryTrack();
}
```

MyFastSimModel.cc : Secondary Particle Production

```
void MyFastSimModel::DoIt(const G4FastTrack& fastTrack,
                          G4FastStep& fastStep)
{
    //some code ...

    //there is a default but it is better to do:
    fastStep.SetNumberOfSecondaryTracks(MaxParticlesProducedPerStep);

    //particle declaration
    const G4DynamicParticle theGamma =
            G4DynamicParticle(G4Gamma::Gamma(),PhotonMomentumDirection,Ephoton);

    //generation of a secondary photon
    fastStep.CreateSecondaryTrack(theGamma,PhotonCoordinateXYZ,PhotonGlobalTime,true);
}
```

MyFastSimModel.cc : Register Fast Sim Process

- Add to physics list:

```
G4FastSimulationPhysics* fastSimulationPhysics = new G4FastSimulationPhysics();
fastSimulationPhysics->BeVerbose();
// -- activation of fast simulation for particles having fast simulation models
// -- attached in the mass geometry:
fastSimulationPhysics->ActivateFastSimulation("e-");
fastSimulationPhysics->ActivateFastSimulation("e+");
// -- Attach the fast simulation physics constructor to the physics list:
physicsList->RegisterPhysics( fastSimulationPhysics );
```

- Important:
  - If any condition of the model is not fulfilled (isApplicable, ModelTrigger), the standard Geant4 process will be active as normal
  - If there are several fast simulation models, the first model whose conditions are fulfilled will be activated

Parallel Worlds for Different Particle Types

- For mass and parallel geometry
  - examples/extended/parameterisations/Par01/examplePar01.cc

```
FTFP_BERT* physicsList = new FTFP_BERT; // G4VModularPhysicsList
G4FastSimulationPhysics* fastSimulationPhysics = new G4FastSimulationPhysics(); // helper
fastSimulationPhysics->BeVerbose();
// - activation of fast simulation for particles having fast simulation models attached
↪  in the mass geometry:
fastSimulationPhysics->ActivateFastSimulation("e-");
fastSimulationPhysics->ActivateFastSimulation("e+");
fastSimulationPhysics->ActivateFastSimulation("gamma");
// - activation of fast simulation for particles having fast simulation models attached
↪  in the parallel geometry:
fastSimulationPhysics->ActivateFastSimulation("pi+","pionGhostWorld");
fastSimulationPhysics->ActivateFastSimulation("pi-","pionGhostWorld");
physicsList->RegisterPhysics( fastSimulationPhysics ); // attach to the physics list
```

- For parallel geometry
  - examples/extended/parameterisations/Par01/Par01ParallelWorldForPion.cc

```
  G4Region* ghostRegion = new G4Region("GhostCalorimeterRegion");
// ghostLogical is a G4LogicalVolume in parallel geometry, a box made of air encompassing
↪  both EM&H calorimeters
ghostRegion->AddRootLogicalVolume(ghostLogical);
```

Examples and Applications

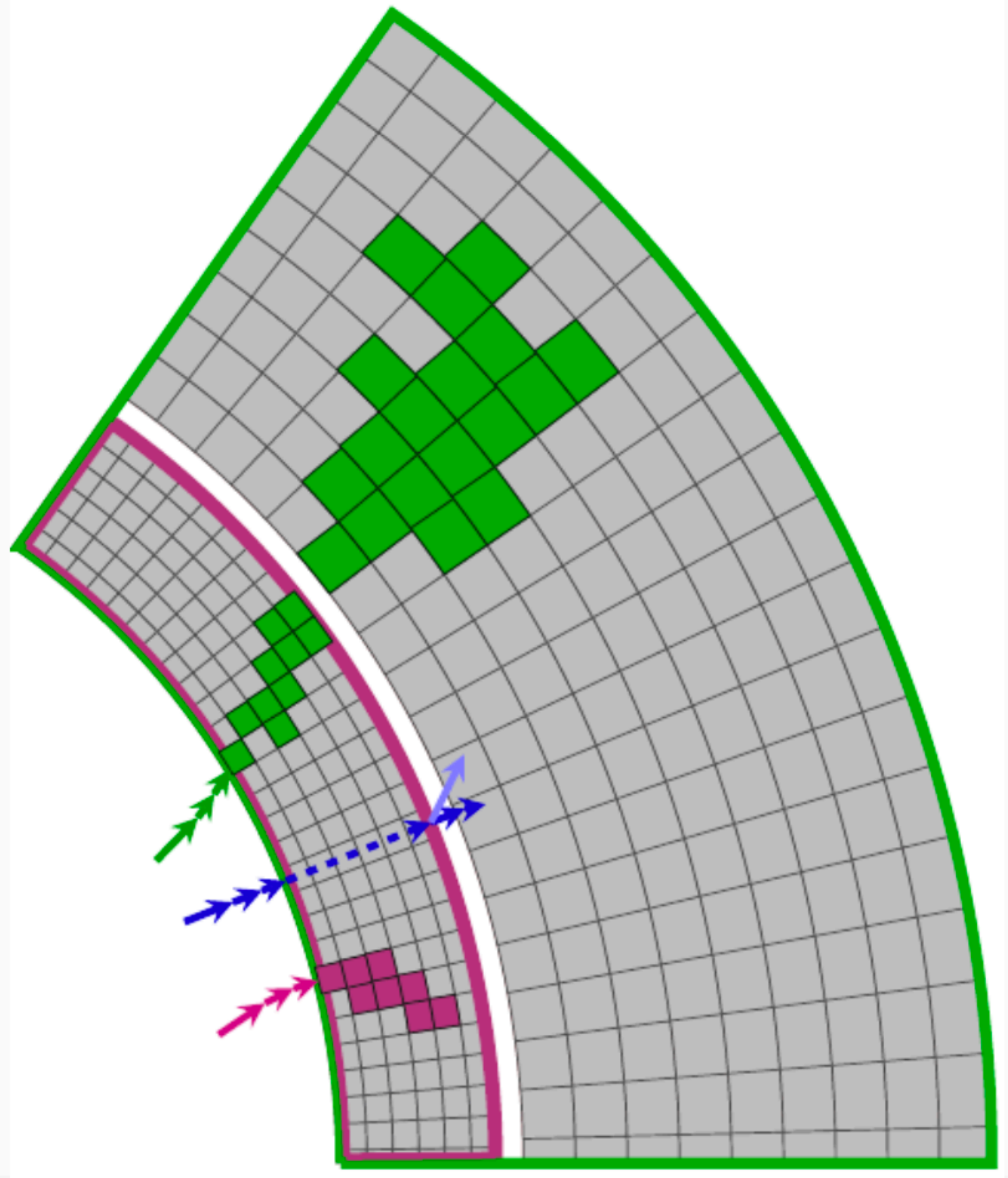
Extended Examples

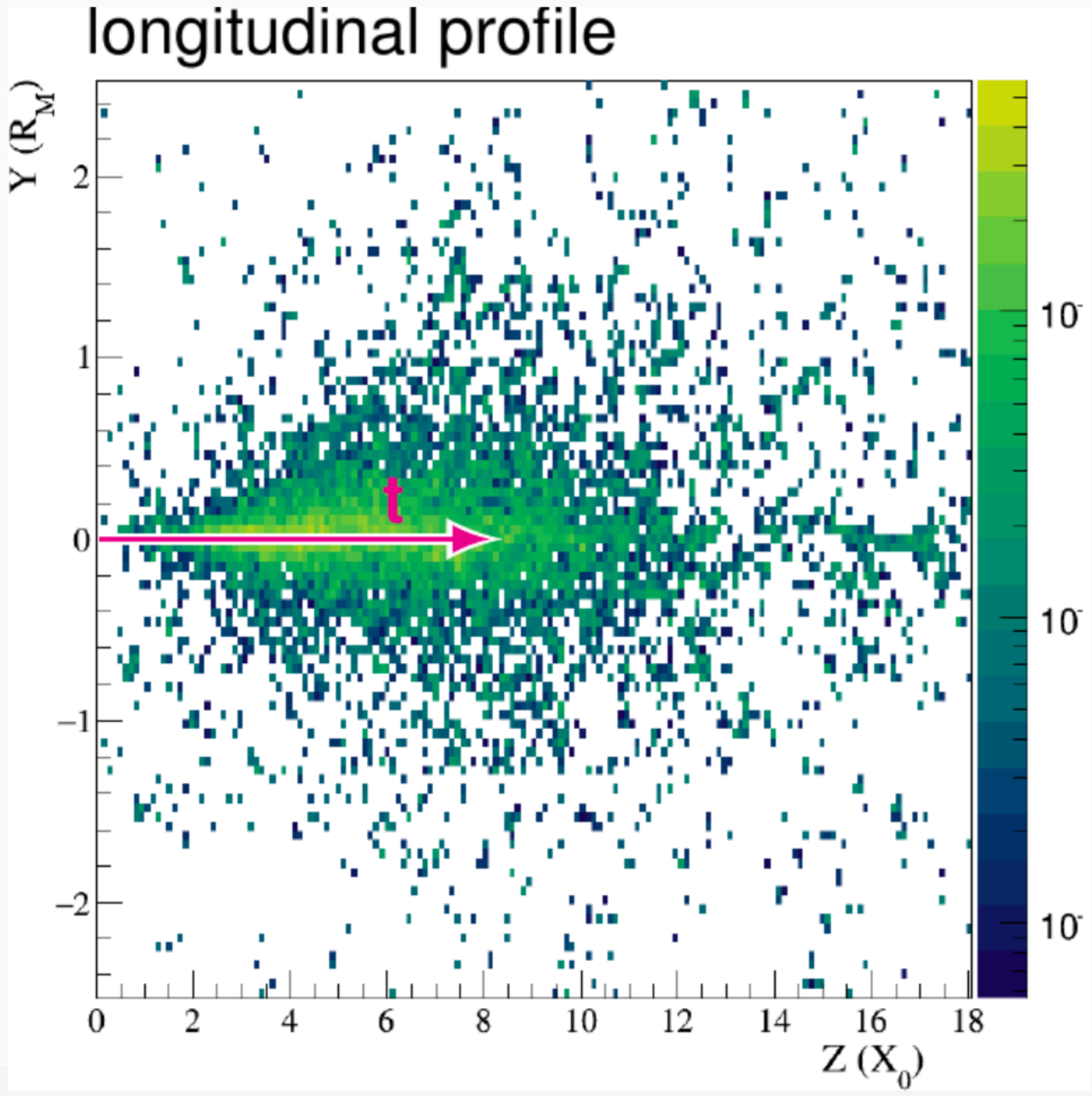
- In examples/extended/parameterisations

  - /Par01/src
    - Par01EMShowerModel.cc (crude e^-, e^+, γ shower parameterization)
    - Par01PionShowerModel.cc (crude π^+, π^- shower model in ghost volume)
    - Par01PiModel.cc (shows how a parameterization can create secondaries)
  - /Par02/src
    - Par02FastSimModelEMCal.cc (e^-, e^+, γ in EM calorimeter using energy smearing)
    - Par02FastSimModelHCal (hadrons in hadronic calorimeter using energy smearing)
    - Par02FastSimModelTracker.cc
  - /Par03/src
    - Par03EMShowerModel.cc (creates multiple energy deposits)
  - /Par04/src
    - Par04MLFastSimModel.cc (uses machine learning to create multiple energy deposits)
  - /gflash
    - GFlashShowerModel (uses GFLASH EM parameterization library)

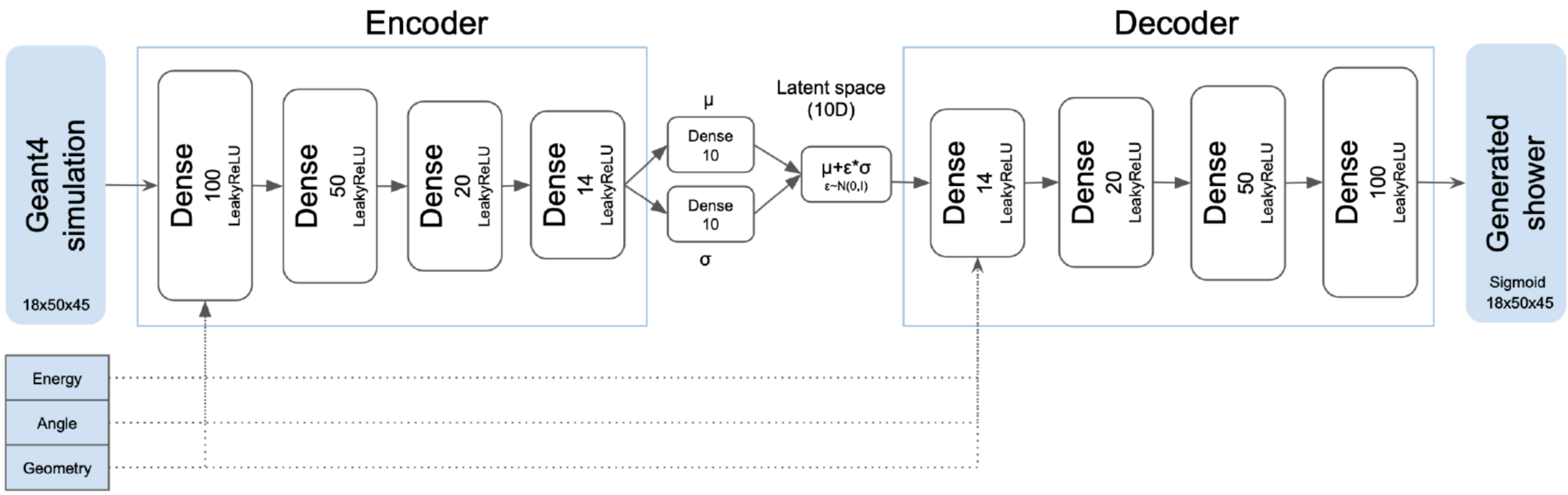
Applications

- Simulation of electromagnetic showers in matter
    - EM calorimeters
- Simulation of sampling calorimeters
- Machine learning
- Implementation of external codes in Geant4



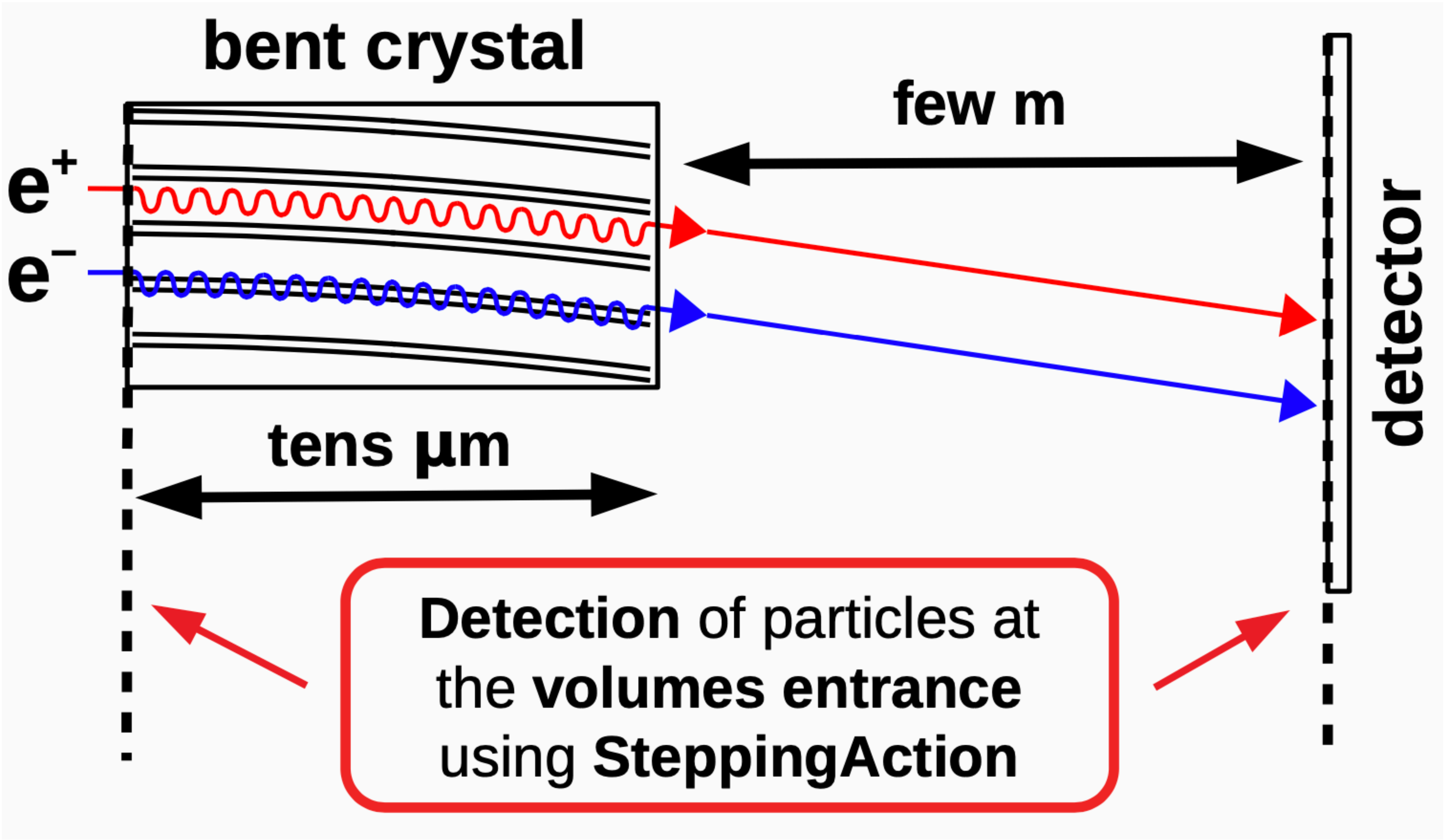


Machine Learning



- Variational auto-encoder: one of the best ways to randomly generate a distribution using initial parameters
  - such as energy, angle of track, geometry, …
- Fast simulation model can upload neural network parameters for inference using
  - Lightweight Trained Neural Network or
  - Open Neural Network Exchange (ONNX) libraries

Channeling in Crystals



- Can use fast simulation to speed up transport of e^- and e^+
- Geant4 simulation developed based on 855 MeV electrons from Mainz Mikrotron on ultra-short crystal
- Multithreaded version of this has run on NURION@KISTI supercomputer

Summary

- Biasing and fast simulation can result in big speed gains in simulation
- Many biasing tools and examples available in Geant4
- Building block biasing allows the combination of biasing elements
- Fast simulation replaces standard Geant4 processes with condensed or approximated processes which are much faster but less detailed
  - can be done using the fast simulation interface
  - activated only in a particular G4Region under certain conditions for certain particles
- Applications for fast simulation:
  - homogeneous and sampling calorimeters
  - electromagnetic shower
  - machine learning and more