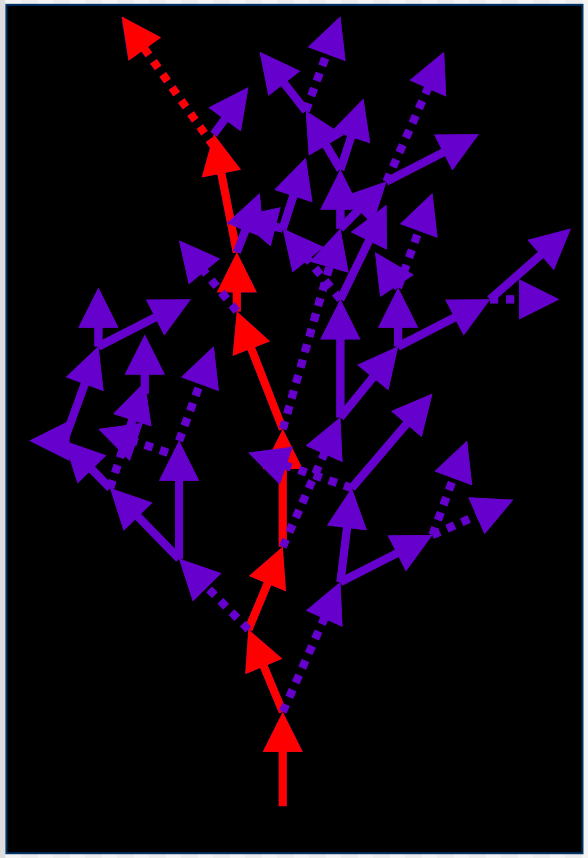
Geant 4

Biasing and scoring

http://geant4.cern.ch



PART I

Geant4 biasing

Event biasing (1/2)

- What is **analogue** simulation ?
  - Sample using natural probability distribution, N(x)
  - Predicts mean with correct fluctuations
  - Can be inefficient for certain applications
- What is non-analogue/**event biased** simulation ?
  - **Cheat** - apply artificial biasing probability distribution, B(x) in place of natural one, N(x)
    - B(x) enhances production of whatever it is that is interesting
    - To get meaningful results, must apply a **weight correction**
  - Predicts same analogue mean with smaller variance
    - Increases efficiency of the Monte Carlo
  - Does **not** predict correct fluctuations
  - Should be used **with care**

Event biasing (2/2)

- Geant4 provides **built-in** general use **biasing techniques**
- The effect consists in producing a small number of secondaries, which are artificially recognized as a huge number of particles by their statistical weights → **reduce CPU time**
- Event biasing can be used, for instance, for the transportation of particles through a thick shielding
- An utility class `G4WrapperProcess` support user-defined biasing

Event biasing techniques (1)

- Production cuts / threshold
  - This is a biasing technique – most popular for many applications: set high cuts to reduce secondary production
- Geometry based biasing
  - Importance weighting for volume/region
  - Duplication or sudden death of tracks
- Primary event biasing
  - Biasing primary events and/or primary particles in terms of type of event, momentum distribution → generate *only primaries* that can produce *events that are interesting for you*

Event biasing techniques (2)

- Forced interaction
  - Force a particular interaction, e.g. within a volume
- Enhanced process or channel and physics-based biasing
  - Increasing cross section for a given process (e.g.bremsstrahlung)
  - Biasing secondary production in terms of particle type, momentum distribution, cross-section, etc.
- Leading particle biasing
  - Take into account only the most energetic (or most important) secondary
  - Currently NOT supported in Geant4

Variance Reduction

- Use variance reduction techniques to **reduce computing time** taken to calculate a result with a given variance (= statistic error)
- Want to **increase efficiency** of the Monte Carlo
- Measure of efficiency is given by

$$\varepsilon = \frac{1}{s^2 T}$$

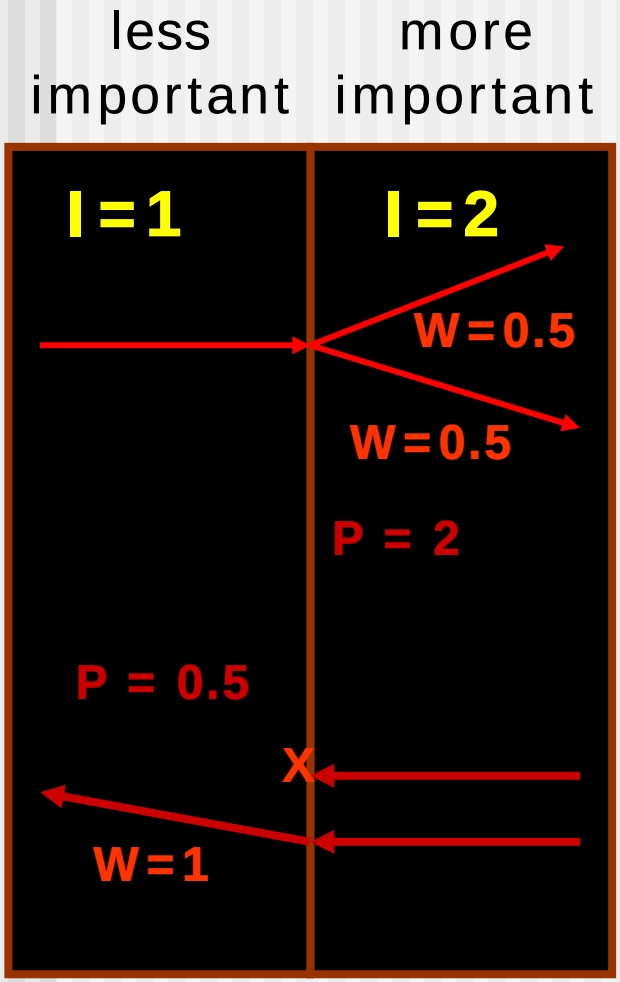
s = variance on calculated quantity
T = computing time

Geometric Biasing

The purpose of geometry-based event biasing is to save computing time by sampling less often the particle histories entering “less important” geometry regions, and **more often in more “important” regions**.

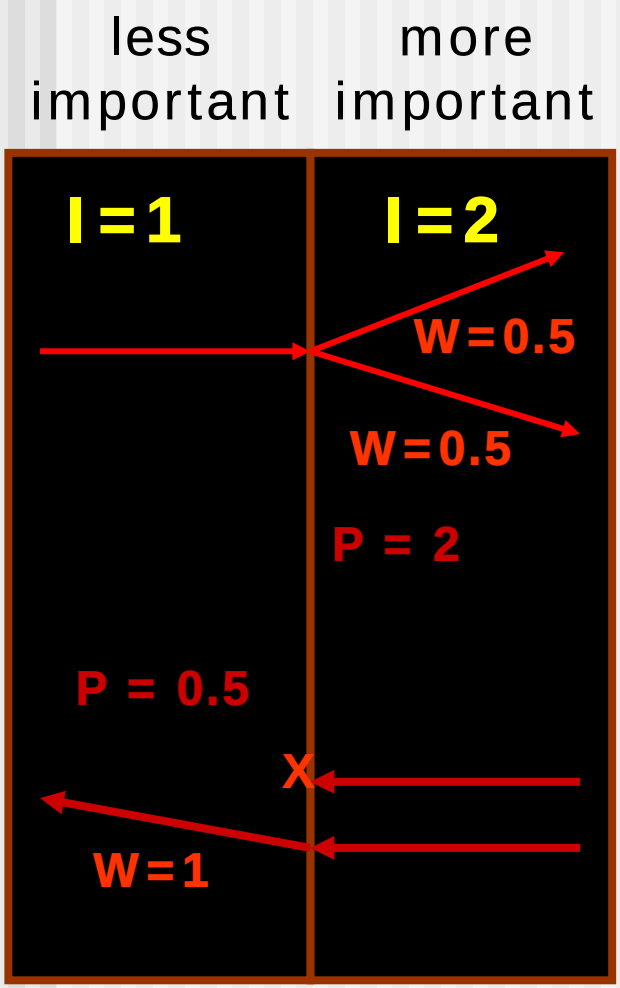
* Importance sampling technique
* Weight window technique

Importance sampling technique (1)



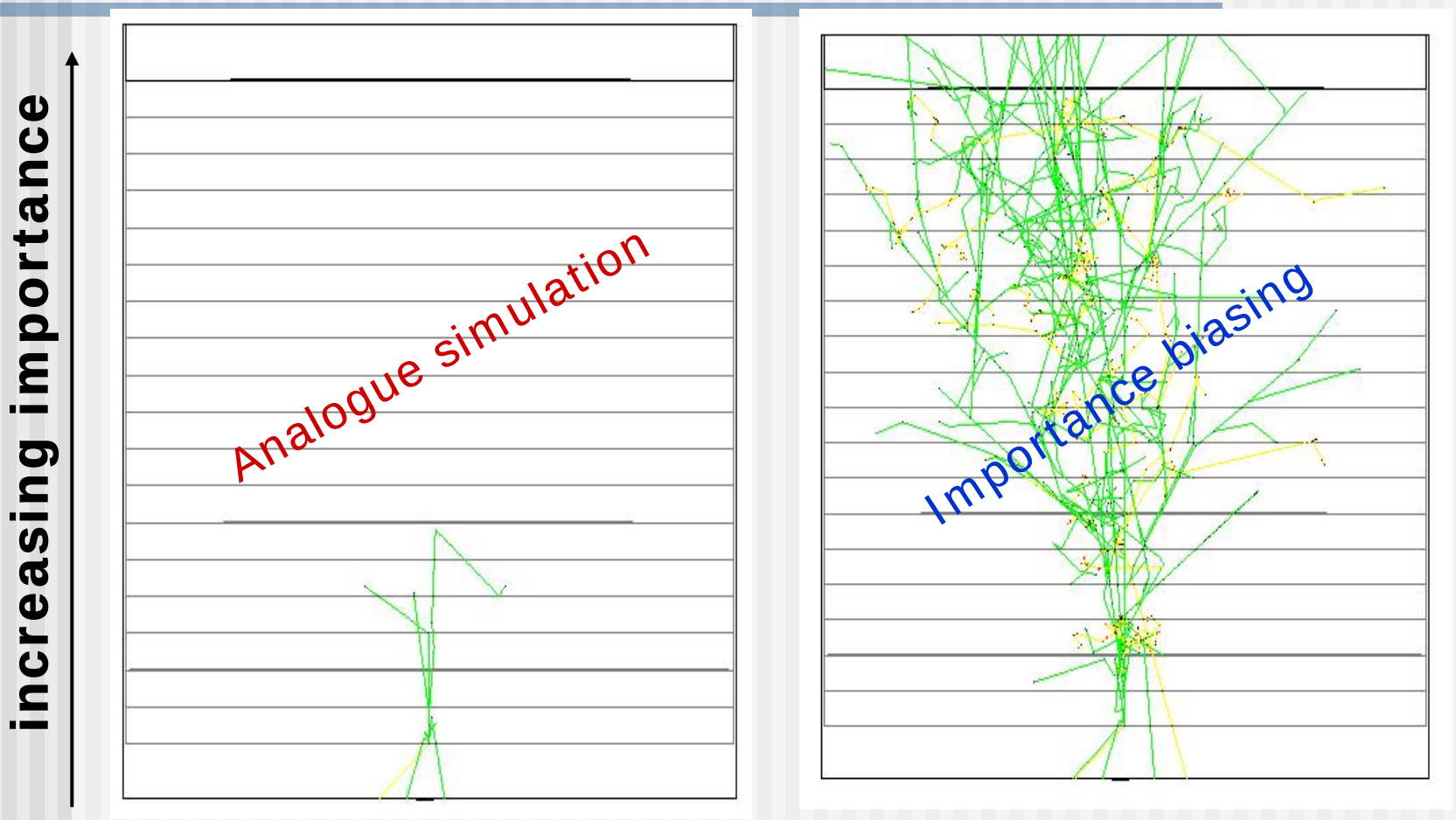
- Importance sampling acts on particles **crossing boundaries** between “importance cells”.
- The action taken depends on the importance value (I) assigned to the cell.
- In general, a track is played either **split** or **Russian roulette** at the geometrical boundary depending on the importance value assigned to the cell.

Importance sampling technique (2)



- Survival probability (P) is defined by the ratio of importance value
  $P = I_{post} / I_{pre}$
- The **track weight** is changed to W/P (weight necessary to get correct results at the end!)
- If **P>1**: splitting a track
  - E.g. creating two particles with half the 'weight' if it moves into volume with double importance value.
- If **P<1**: Russian-roulette in opposite direction
  - E.g. Kill particles according to the survival probability (1 - P).

Importance biasing



10 MeV neutron in thick concrete cylinder

Physics biasing

- Built-in cross section biasing for PhotoInelastic, ElectronNuclear and PositronNuclear processes

```
G4ElectroNuclearReaction * theeReaction = new G4ElectroNuclearReaction;
G4ElectronNuclearProcess theElectronNuclearProcess;
theElectronNuclearProcess.RegisterMe(theeReaction);
theElectronNuclearProcess.BiasCrossSectionByFactor(100);
```

- Similar tool for rare EM processes (e^+e^- annihilation to μ pair or hadrons, γ conversion to $\mu^+\mu^-$)

```
G4AnnihiToMuPair* theProcess = new G4AnnihiToMuPair();
theProcess->SetCrossSecFactor(100);
```

- It is possible to introduce these factors for all EM processes, with a definition of customized processes that inherit from the "normal" ones (→ extended example)

- **Artificially** enhance/reduce cross section of a process (useful for thin layer interactions or thick layer shielding)

General implementation under development

How to learn more about biasing

There are **examples** in Geant4, to show how to use the most common biasing techniques:

examples/extended/biasing

examples/advanced/Tiara

geometry-based biasing

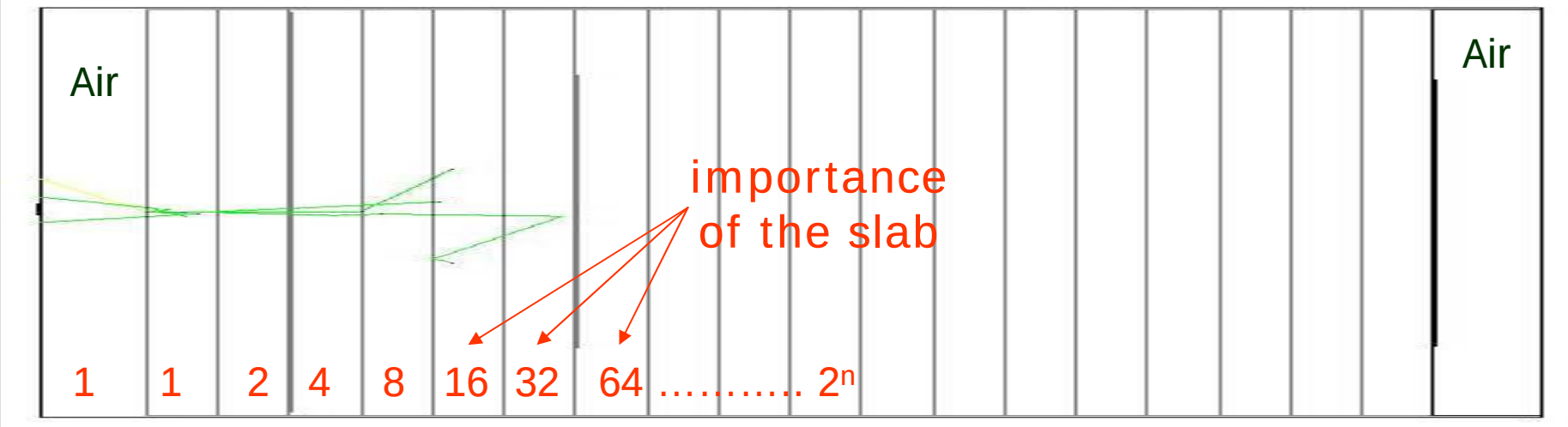
examples/extended/medical/fanoCavity

cross-section biasing (Compton scattering)

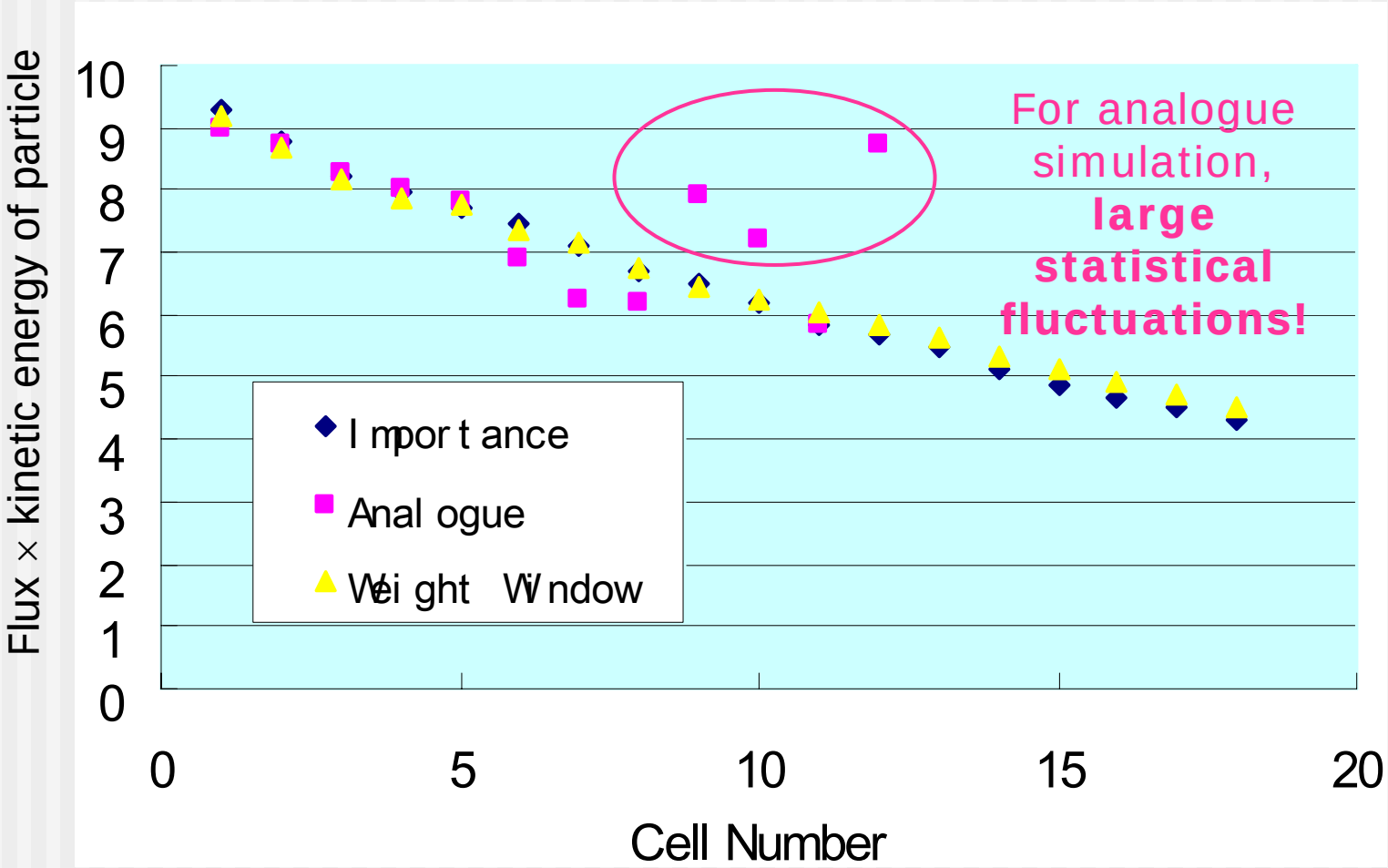
Additional documentation about biasing techniques available in the Geant4 User Guide, section 3.7

Biasing example B01

- Shows the **importance sampling** in the mass (tracking) geometry
- 10 MeV neutron shielding by cylindrical thick concrete
- 80 cm high concrete cylinder divided into 18 slabs (importance values assigned in the DetectorConstruction for simplicity)



Results of example B01



PART II

Geant4 scoring

Extract useful information (1/2)

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
  - You have to **add a bit of code** to extract information useful to you
- One way is to use the **available user hooks** described yesterday (G4SensitiveDetector, G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have full access to almost all information
    - Straight-forward, but do-it-yourself

Extract useful information (2/2)

- Alternatively to user-defined sensitive detectors, primitive scorers provided by Geant4 can be used
- Geant4 provides a number of **primitive scorers**, each one accumulating one physics quantity (e.g. total dose) for an event
- It is convenient to use primitive scorers instead of user-defined sensitive detectors when:
  - you are not interested in recording each individual step, but accumulating physical quantities for an event or a run
  - you have not too many scorers

G4MultiFunctionalDetector

- **`G4MultiFunctionalDetector`** is a concrete class derived from **`G4VSensitiveDetector`**
- It should be **assigned to a logical volume** as a **kind of** (ready-for-the-use) **sensitive detector**
- It takes an arbitrary number of `G4VPrimitiveSensitivity` classes, to define the scoring quantities that you need
  - Each `G4VPrimitiveSensitivity` accumulates one physics quantity for each physical volume
  - E.g. `G4PSDoseScorer` (a concrete class of `G4VPrimitiveSensitivity` provided by Geant4) accumulates dose for each cell
- By using this approach, **no need to implement sensitive detector** and **hit classes**!

G4VPrimitiveSensitivity

- Primitive scorers (classes derived from G4VPrimitiveSensitivity) have to be registered to the `G4MultiFunctionalDetector`
- They are designed to **score one kind of quantity** (surface flux, total dose) and to **generate one hit collection** per event
  - automatically named as

    `<MultiFunctionalDetectorName>/<PrimitiveScorerName>`
  - hit collections can be retrieved in the EventAction or RunAction (as those generated by sensitive detectors)
  - do **not** share the same primitive score object among multiple G4MultiFunctionalDetector objects (results may mix up!)

For example…

```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
   G4MultiFunctionalDetector* myScorer = new
     G4MultiFunctionalDetector("myCellScorer");
   G4SDManager::GetSDMpointer()->
     AddNewDetector(myScorer);
   myCellLog->SetSensitiveDetector(myScorer);
   G4VPrimitiveSensitivity* totalSurfFlux = new
     G4PSFlatSurfaceFlux("TotalSurfFlux");
   myScorer->Register(totalSurfFlux);
   G4VPrimitiveSensitivity* totalDose = new
     G4PSDoseDeposit("TotalDose");
   myScorer->Register(totalDose);
}
```

instantiate multi-functional detector and register in the SD manager

attach to volume

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

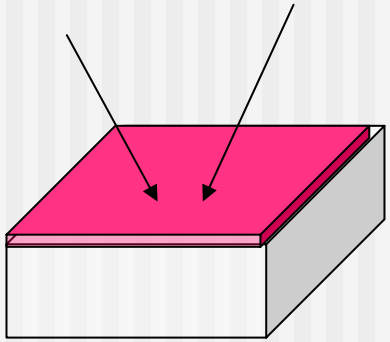
Some primitive scorers you may find useful

- Concrete Primitive Scorers (→ Application Developers Guide 4.4.6)
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDepsit, G4PSDoseDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent,G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep, G4PSCellCharge

A closer look at some scorers...

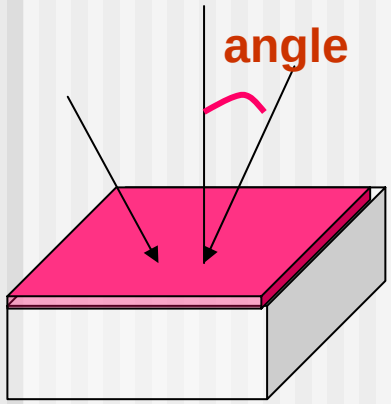
SurfaceCurrent :

Count number of injecting particles at defined surface.



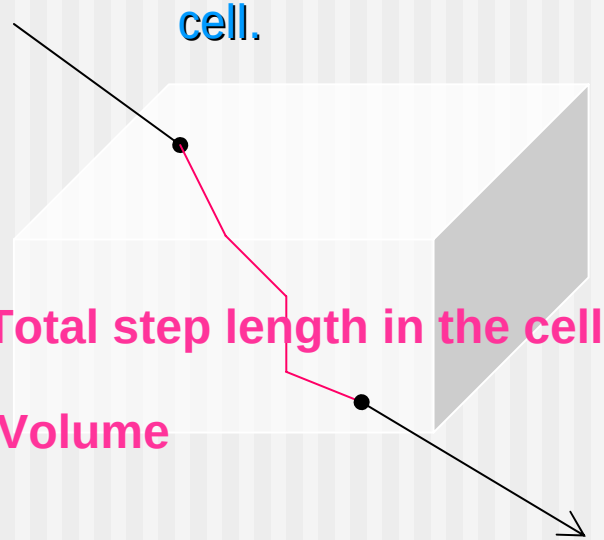
CellFlux :

Sum of L / V of injecting particles in the geometrical cell.



SurfaceFlux :

Sum up 1/cos(angle) of injecting particles at defined surface



L : Total step length in the cell

V : Volume

G4VSDFilter

- A **`G4VSDFilter`** can be attached to `G4VPrimitiveSensitivity` to define **which kind of tracks** have to be scored (e.g. one wants to know surface flux of protons only)
  - `G4SDChargeFilter` (accepts only charged particles)
  - `G4SDNeutralFilter` (accepts only neutral particles)
  - `G4SDKineticEnergyFilter` (accepts tracks in a defined range of kinetic energy)
  - `G4SDParticleFilter` (accepts tracks of a given particle type)
  - `G4VSDFilter` (base class to create user-customized filters)

For example…

```
MyDetectorConstruction::Construct()
{
   G4VPrimitiveSensitivity* protonSurfFlux
   = new G4PSFlatSurfaceFlux("pSurfFlux");
    G4VSDFilter* protonFilter = new
      G4SDParticleFilter("protonFilter");
    protonFilter->Add("proton");

    protonSurfFlux->SetFilter(protonFilter);

    myScorer->Register(protonSurfFlux);
}
```

create a primitive scorer (surface flux), as before

create a particle filter and add protons to it

register the filter to the primitive scorer

register the scorer to the multifunc detector (as shown before)

Command-based scoring (β release)

Under development!

Thanks to the newly developed parallel navigation, an **arbitrary scoring mesh geometry** can be defined which is independent to the volumes in the mass geometry.
Also, G4MultiFunctionalDetector and primitive scorer classes now offer the built-in scoring of most-common quantities

UI commands for scoring → no C++ required, apart from instantiating `G4ScoringManager` in `main()`

- Define a scoring mesh
  - /score/create/boxMesh <mesh_name>
  - /score/open, /score/close
- Define mesh parameters
  - /score/mesh/boxsize <dx> <dy> <dz>
  - /score/mesh/nbin <nx> <ny> <nz>
  - /score/mesh/translate,
- Define primitive scorers
  - /score/quantity/eDep <scorer_name>
  - /score/quantity/cellFlux <scorer_name>
  - currently **20 scorers** are available
- Define filters
  - /score/filter/particle <filter_name> <particle_list>
  - /score/filter/kinE <filter_name> <Emin> <Emax> <unit>
  - currently **5 filters** are available
- Output
  - /score/draw <mesh_name> <scorer_name>
  - /score/dump, /score/list

How to learn more about scoring

Have a look at the **dedicated extended examples** released with Geant4:

examples/extended/runAndEvent/RE02
(use of primitive scorers)

examples/extended/runAndEvent/RE03
(use of UI-based scoring)

PART III

Summary

Summary

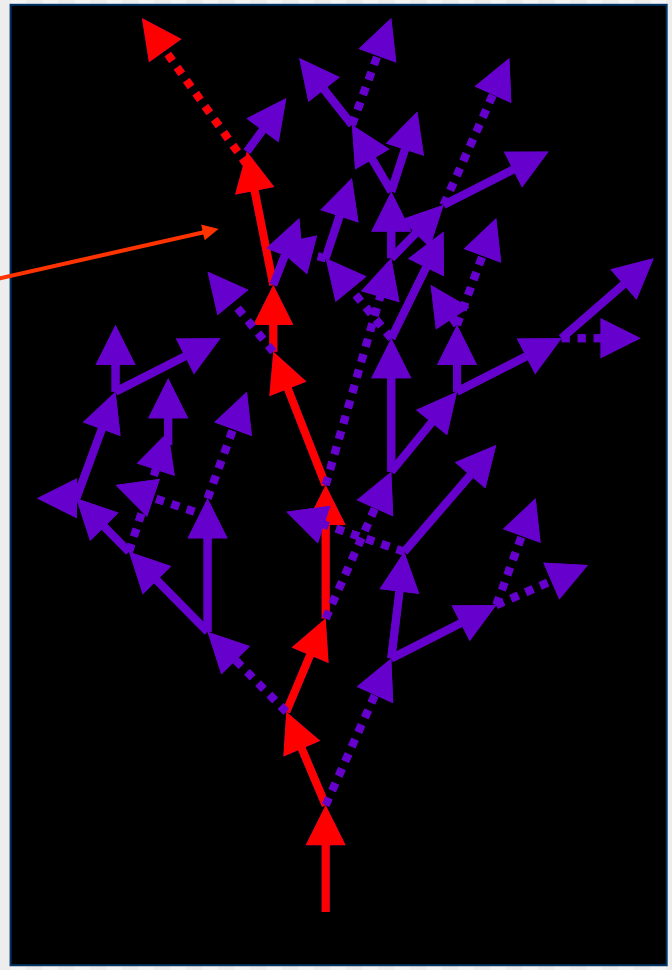
- Geant4 offers the possibility to improve computing performance and save CPU time via fast simulation and biasing
- A number of biasing techniques are available but are the *user's responsibility* to use the results correctly (e.g. set proper weights)
- Scoring is implemented with a degree of flexibility, offering convenience of keeping tallies of common quantities (doses, fluxes, etc.) without the user-defined sensitive detector
- A number of examples available in
  `$G4INSTALL/examples/extended`

PART IV

Backup slides

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
  - Physical quantities such as energy are not conserved for each event



A tip for scoring

- For scoring purposes, you need to accumulate a physical quantity (e.g. energy deposition of a step) for entire run of many events. In such a case, do NOT sum up individual energy deposition of each step directly to a variable for entire run.
  - Total energy deposition of 10^6 events of 1 GeV incident particle ends up to 1 PeV (10^{15} eV), while energy deposition of each single step is O(1 keV) or even smaller → possible rounding problems
- Possible work-around: create your own Run class derived from G4Run, and implement RecordEvent(const G4Event*) virtual method. Here you can get all output of the event so that you can accumulate the sum of an event to a variable for entire run.