

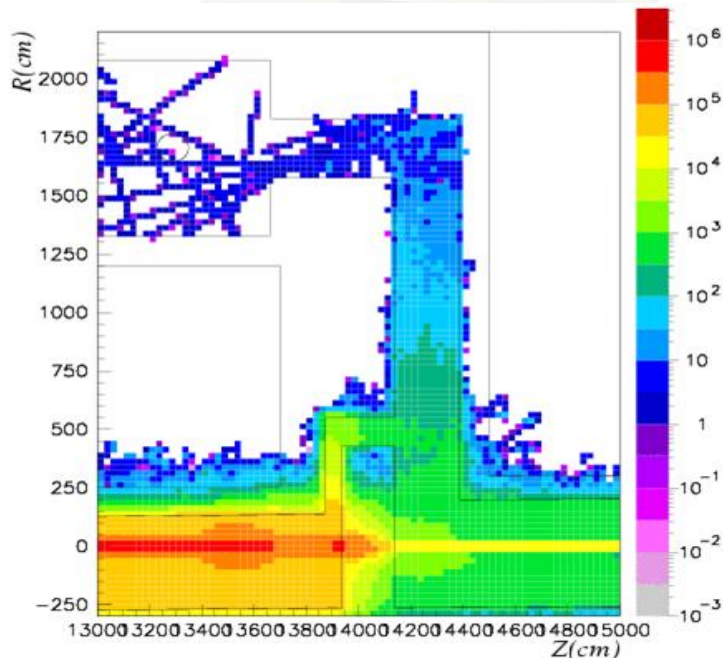


Simulation Optimization

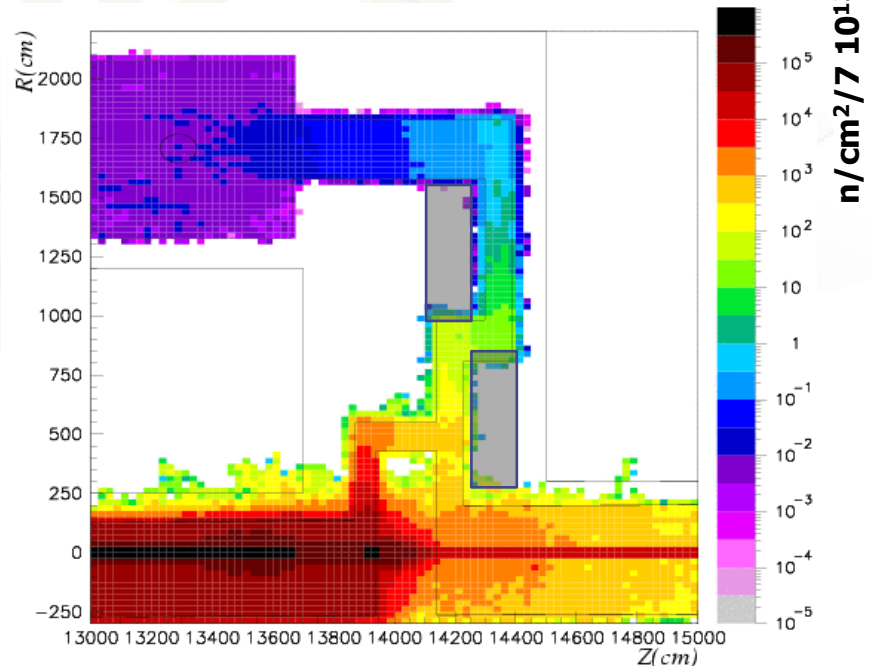
Advanced FLUKA Course

Concept

- Variance reduction techniques in Monte Carlo calculations **reduce the computer time** *or the opposite* to obtain results of sufficient precision in the phase-space region of interest.
- **Remember:** that precision is not the only requirement for a Good Monte Carlo calculation. Even a zero variance calculation cannot accurately predict natural behavior if **other sources of error** are not minimized.



No Bias and no maze



Region Biasing + maze

Is optimization necessary?

- **Visual inspection** of histograms (specially 2D maps) is a powerful way to determine if simulations have **NOT** converged in the area of interest. Big fluctuation in colors between neighboring voxels warn of potential systematic errors.
- However, even in smooth 2D histograms lack of convergence may be present, e.g.:
 - If the **scoring scale** spans over many orders of magnitude significant differences may be hard to spot
 - 2-step simulations may give a **false sense of convergence** if first step has not converged but 2nd step has. Plot also maps for the 1st step!
 - Relatively smooth maps may have some **'hard'-tracks** overlaid typically indicating strong energy dependent weight dispersion

Monte Carlo Flavors

	Microscopic Analog	Microscopic Biased	Macroscopic Analog
Physics Models	Theoretical	Theoretical	Parameterizations
PDF sampling	Physical processes	Artificial distributions	Fits & Data
Predict Average	Yes	Yes	Yes
Predict Higher Moments	Yes	-	-
Preserves Correlations	Yes	-	-
Reproduces Fluctuations	Yes	-	-
Rare events	-	Yes	-
Predictability	Yes	Yes	-
Convergence	Slow	Fast privileged regions	Fast
Safe	Yes	Almost	-

Analog vs. Biased - 1

Analog Monte Carlo

- samples from **actual phase space distributions**
- predicts average quantities and **all statistical moments** of any order
- preserves **correlations** and reproduces **fluctuations** (provided the physics is correct...)
- is (*almost*) safe and can (*sometimes*) be used as “black box”

BUT

- is **inefficient** and converges very slowly
- fails to predict important contributions due to **rare events**

Analog vs. Biased - 2

Biased Monte Carlo

- samples from **artificial distributions** and applies a **weight** to the particles to correct for the bias
- predicts **average quantities**, but not the higher moments
(on the contrary, its goal is to minimize the second moment)
- same mean with smaller variance, *i.e.*, **faster convergence**

BUT

- **cannot** reproduce correlations and fluctuations
- requires physical judgment, experience and a good understanding of the problem (**it is not a "black box"!**)
- in general, a user does not get the definitive result after the first run, but needs to do a **series of test runs** in order to **optimize the biasing parameters**

→ **balance between user's time and CPU time**

Figure of Merit

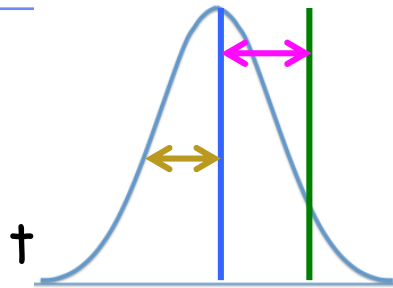
- Computer cost of an estimator

$$FOM = \sigma^2 \times t$$

$$\sigma^2 = \text{Variance} \propto 1/N, \quad t = \text{CPU time} \propto N$$

- some biasing techniques are aiming at reducing the σ^2 , others at reducing t
- often reducing σ^2 increase t , and vice versa
- therefore, minimizing $\sigma^2 \times t$ means to reduce σ at a faster rate than t increases, or vice versa
- \Rightarrow the choice depends on the problem, and sometimes a combination of several techniques is most effective
- **bad judgment, or excessive "forcing"** on one of the two variables, can have catastrophic consequences on the other one, making computer cost "explode"

Optimization strategies and cost



Optimization consists in focusing CPU use towards the area of interest so that **statistical error** is reduced with minimal no increase of **systematic errors**

Several types of optimization...

No biasing: no/insignificant systematic error, preserves correlations

Code & optimization: e.g. less use of memory: use 2D USRBIN in stead of 3D, read data at initialization, ...

Geometry optimization: Choice of body types, definition of geometry logics, number of neighboring regions (avoid NAS>>), avoid unnecessary/excessive transformation directives

Scoring/transport choice: energy deposition or fluence, *single scattering vs. multiple scattering, run-time processing

Biasing M1: Energy thresholds, blackhole outside of region of interest, detail of geometry, number of processes considered (e.g. neglect photonuc for hadron beams), granularity of scoring histograms...

Biasing Mn ($n > 1$): Importance biasing, LAM-BIAS, EMF-BIAS

Biasing Techniques

FLUKA offers the following possibilities for biasing

- Importance Biasing (**BIASING**)
- Weight window (**WW-FACTOR**, **WW-THRESH**, **WW-PROFILE**)
- Leading Particle Biasing (**EMF-BIAS**)
- Multiplicity Tuning (**BIASING**)
- Biased down scattering for neutrons, *only for experts* (**LOW-DOWN**)
- Non analogue absorption (**LOW-BIAS**)
- Biasing Mean free paths (**LAM-BIAS**)
- User defined biasing (**usbset.f**, **usimbs.f**)

Other optimization checks

- CPU intensive consuming physics options and uses

User written biasing

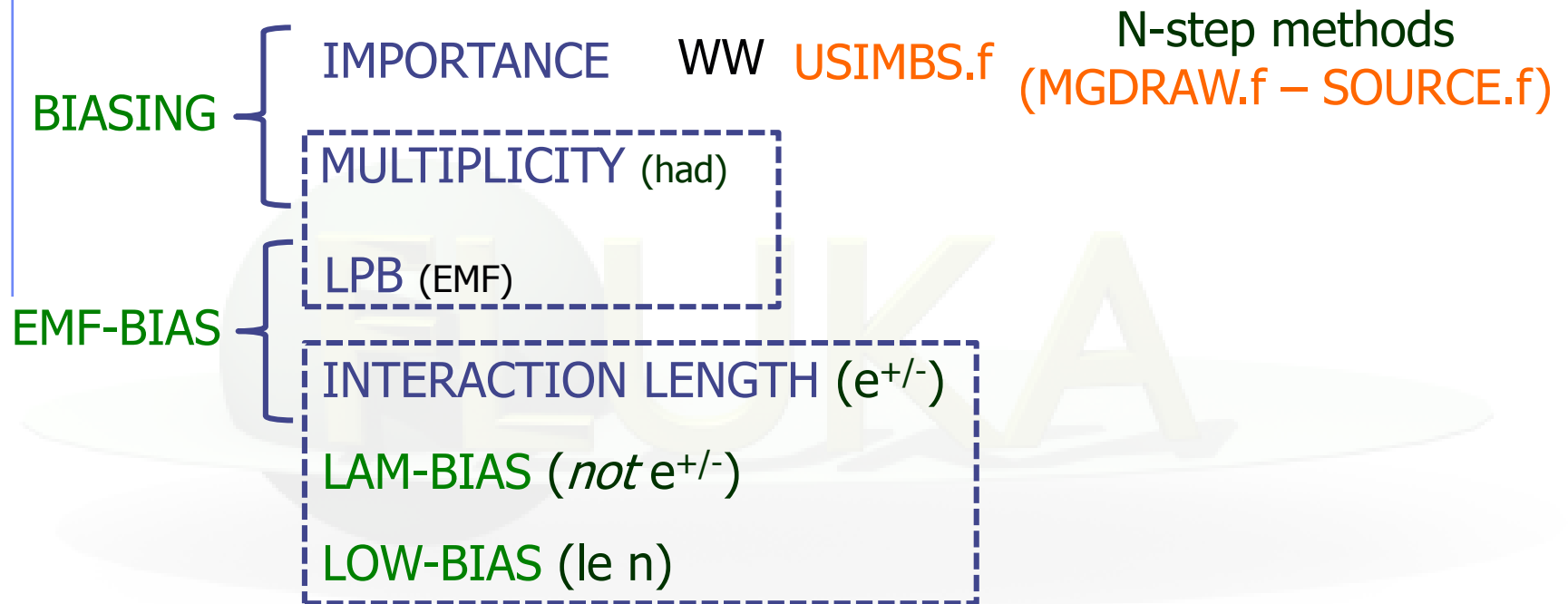
FLUKA offers the following routines for **user-written biasing**

- **ubsset.f: User Biasing SETting**
 - called after reading in the input file and before the first event
 - allows to alter almost any biasing weight on a region-dependent basis
- **usimbs.f: USer defined IMportance Biasing**
 - if activated, called at every particle step
 - allows to implement any importance biasing scheme based on region number and/or phase space coordinates
- **udcdrl.f: User defined DeCay DiRection biasing and Lambda**
 - only for neutrinos emitted in decays: bias the direction of emitted neutrino

Not biasing by itself, but it could be used for generating biased runs

- **source.f: User written source**
 - to sample primary particle properties from distribution in space, energy, time...

Summary Biasing Techniques



Reduce memory / check requirements

Use *physics process* and *thresholds* relevant to the problem

Importance Biasing

- Importance biasing combines two techniques:
 - **Surface splitting**: Reduces σ but increases t
 - **Russian roulette**: which does the opposite
- It is the **simplest, most safe** and easiest to use of all biasing
- The user assigns a **relative importance** to each geometry region (the absolute value doesn't matter) based on:
 - **expected fluence attenuation** with respect to other regions
 - **probability of contribution** to score by particles entering the region
- Importance biasing is commonly used to **maintain a constant particle population**, compensating for attenuation due to absorption or distance.
- In FLUKA it can be **tuned per type of particle**

Importance biasing - 2

Surface Splitting

A particle crosses a region boundary, coming from a region of importance I_1 and entering a region of **higher** importance $I_2 > I_1$:

- the particle is replaced on average by $n=I_2/I_1$ identical particles with the same characteristics
- the **weight** of each “daughter” is multiplied by I_1/I_2

If I_2/I_1 is too large, **excessive splitting** may occur with codes which do not provide an appropriate protection .

An **internal limit** in FLUKA prevents excessive splitting if I_2/I_1 is too large (> 5), a problem found in many biased codes.

Importance biasing - 3

Russian Roulette

A particle crosses a region boundary, coming from a region of importance I_1 and entering a region of **lower** importance $I_2 < I_1$:

- the particle is submitted to a random **survival test**: with a chance I_2/I_1 the particle survives with its **weight increased by a factor I_1/I_2**
- with a chance $(1 - I_2/I_1)$ the particle is killed

Importance biasing is commonly used to **maintain a uniform particle population**, compensating for attenuation due to absorption or distance. In FLUKA it can be **tuned per particle type**.

Importance biasing - 4

Note:


In FLUKA, for technical reasons, importances are internally stored as integers. Therefore, importances can only take values between 0.0001 and 100000. An input values 0.00015 is read as 0.0001, 0.00234 is read as 0.0023, *etc.*

There is also a user routine **USIMBS** which allows to assign importances not only at boundaries, but **at each step**, according to any logic desired by the user (as a function of position, direction, energy,...).

Very powerful, but ***time-consuming*** (it is called at each step!). The user must balance the time gained by biasing with that wasted by calls.

Importance biasing – 5

BIASING	0.0	0.0	4.64	8.	18.	2.PRINT
---------	-----	-----	------	----	-----	---------

 BIASING Opt: ▼	Type: ▼ Reg: ▼	RR: to Reg: ▼	Imp: Step:
--	-------------------	------------------	---------------

The meaning of WHAT(1)...WHAT(6) and SDUM is different depending on the sign of WHAT(1):

If WHAT(1) >= 0.0 :

WHAT(1) specifies the particles to be biased

- = 0.0 : all particles
- = 1.0 : hadrons and muons
- = 2.0 : electrons, positrons and photons
- = 3.0 : low energy neutrons


WHAT(2) = *(see multiplicity tuning)*

WHAT(3) = region importance (Default = 1.0)

Allowed values range from 0.0001 to 100000.

Importance biasing - 6

BIASING	0.0	0.0	4.64	8.	18.	2.PRINT
---------	-----	-----	------	----	-----	---------

 BIASING	Type: ▼	RR:	Imp:
Opt: ▼	Reg: ▼	to Reg: ▼	Step:

If WHAT(1) >= 0.0 :

WHAT(4) = lower bound of the region indices/names (Default = 2.0)

WHAT(5) = upper bound of the region indices/names (Default = WHAT(4))

WHAT(6) = step length in assigning indices (Default = 1.0)

SDUM = PRINT : importance biasing counters are printed
 = NOPRINT: counters are not printed
 (cancels any previous PRINT request)
 = USER: importance biasing
 according to the user defined routine USIMBS
 = NOUSER: reset to default (cancels any previous USER request)
 = RRPRONLY: multiplicity biasing for primary particles only
 = (blank): ignored

(Default: NOPRINT, NOUSER, multiplicity biasing for all generations)

Importance biasing - 7

BIASING	2.0	0.0	10.0	7.0	11.0	2.0
BIASING	2.0	0.0	15.0	8.0	9.0	0.0
BIASING	-1.0	0.0	3.0	4.0	0.0	0.0

If WHAT(1) < 0.0 :

WHAT(1) : flag indicating that all region importances shall be modified by a particle-dependent factor

WHAT(2) \geq 0.0 : modifying factor M

< 0.0 : M is reset to the default value 1.0

WHAT(3) = lower bound of the particle indices/names (Default: = 1.0)

WHAT(4) = upper bound of the particle indices/names

(Default: = WHAT(3) if WHAT(3) > 0, all particles otherwise)

WHAT(5) = step length in assigning particle indices (Default: 1.0)

WHAT(6) = not used

SDUM = PRIMARY : importance biasing is applied also to primaries

NOPRIMARY : importance biasing is applied only to secondaries

Default = PRIMARY

Importance Biasing – 8 : Problems 1

- In complex geometries, it may be hard to keep track of all importances in every region

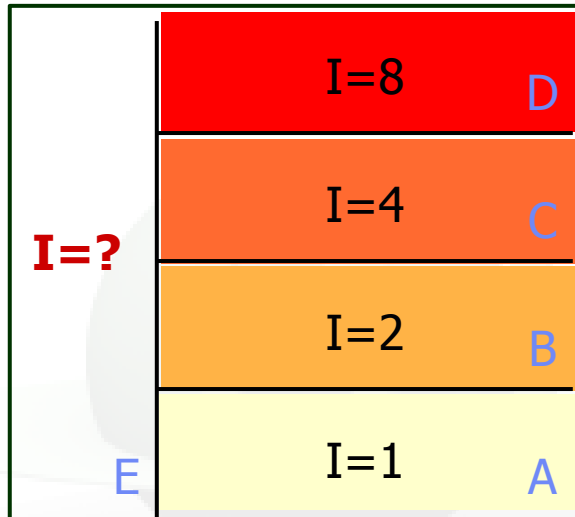
I=16	E
I=1	D
I=4	C
I=2	B
I=1	A

When going from **C** to **D** 75 % of the particles are killed!! – waste of previous tracking effort (then the remaining particles entering E are split in an average of $\max(16/1,5) = 5$ copies)
 Backscattered particles going from D to C will be split into 4 copies! – unnecessary!

- To avoid such 'cold spots', it is helpful to plot importances. This can be performed with Flair>Geoviewer

Importance Biasing – 8 : Problems 2

- Although important biasing is relatively easy and safe to use, there are a few cases where caution is recommended



Which importance shall we give to region E? Whatever value we choose we will get an inefficient splitting/RR at some of the boundaries.

- Another case is that of **splitting in vacuum** (or air). Splitting daughters are **strongly correlated**. It must be made sure that their further histories are differentiated enough to forget their correlation.
- The above applies in part also to **muons** (the differentiation provided by **multiple scattering** and by **Landau dE/dx** fluctuations is not always sufficient).

usimbs.f: User Defined Importance Biasing

- Typical problem: Spend a lot of time to write the problem input, geometry and on the first runs you realize that statistics are not good
- First method (and safest) is to introduce region importance biasing. In FLUKA you can introduce it with two ways:
 - 1st Manually slice the geometry and increase the number of regions. Modifying an existing geometry to introduce biasing can be a very cumbersome process
 - 2nd Introduce the “importance biasing” information with a user fortran routine **independent of the regions defined in the geometry**
- Routine: **usimbs.f**

USer defined IMportance BiaSing

Allows to implement any importance biasing scheme based on region number and/or phase space coordinates

usimbs.f: User Defined Importance Biasing

- Enable the call to **USIMBS** routine with the **BIASING** card:
 - WHAT(1) Particles to be biased
 - **WHAT(2) and WHAT(3) \neq 1.0** (Any value \neq 1.0)
 - WHAT(4) Lower bound of region
 - WHAT(5) Upper bound of region
 - WHAT(6) Step
 - SDUM = **USER**



Remember:

- If WHAT(3)=1 for a region, the routine will not be called during tracking of particles inside that region
- Cannot have both normal importance BIASING with cards and the routine at the same time

usimbs.f – Routine

The routine is called on every particle step!

WARNING: can slow down the execution speed! Use with caution.

Input:

- Region information at the **beginning** and **end** of the step
- X,Y,Z coordinates through the **TRACKR** common.
Beginning: X, Y, Ztrack(0)
End: X, Y, Ztrack(Ntrack)
- **Particle** type and **Energy** could be used for even more advanced biasing schemes

Output:

The routine must return the importance ratio to the new position (end/beginning) in the variable **FIMP** < 5.0

Entry: USIMST

Split the particles step between interactions in half (or any other user defined value)

```
SUBROUTINE USIMBS ( MREG, NEWREG, FIMP )

      INCLUDE ' (DBLPRC) '
      INCLUDE ' (DIMPAR) '
      INCLUDE ' (IOUNIT) '

      *
      *-----
      *   User defined IMportance BiaSing:
      *   Created on 02 july 2001   by   Alfredo Ferrari & Paola Sala
      *                                           Infn - Milan
      *
      *   Last change on 09-jul-01   by   Alfredo Ferrari
      *
      *   Input variables:
      *           Mreg = region at the beginning of the step
      *           Newreg = region at the end of the step
      *   (thru common TRACKR) :
      *           Jtrack = particle id. (Paprop numbering)
      *           Etrack = particle total energy (GeV)
      *           X,Y,Ztrack(0) = position at the beginning of the step
      *           X,Y,Ztrack(Ntrack) = position at the end of the step
      *
      *   Output variable:
      *           Fimp = importance ratio (new position/original one)
      *-----
      *
      *   INCLUDE ' (TRACKR) '
      *   FIMP = ONEONE
      *   RETURN
      *   END

      *=====
      *
      *   Entry USIMST:
      *   Input variables:
      *           Mreg = region at the beginning of the step
      *           Step = length of the particle next step
      *
      *   Output variable:
      *           Step = possibly reduced step suggested by the user
      *=====
      *
      *   ENTRY USIMST ( MREG, STEP )
      *   IF ( STEP .GT. ONEONE ) STEP = HLFHLF * STEP
      *   RETURN

      *=== End of subroutine Usimbs =====
      END
```

usimbs.f – Important Notes

- The routine has **only a relative effect** on the weight of the particle
⇒ Beam particles can have any weight.
- Importance ratio will be limited by **WW-THRESH** card
- The Russian Roulette / Splitting will take place at the middle (defined by the ENTRY) of the step, **and not on a fixed region boundary.**
- The biasing position will be a **little fuzzy** depending on the particle step. This has a visible effect when it is applied to **low density materials** (i.e. air)
- Results will **similar but not the same** as with the manual region biasing.
- Is a great time saver for complex geometries, as well different biasing schemes
- Combined with the particle type and energy could become very powerful

usimbs.f: Simple Example

- Biasing a factor of 2 every 50 cm on the Z direction from 100cm to 500cm

```
ZSTART = ZTRACK(0)
IF (ZSTART .LT. 100.0D0) THEN
    FSTART = ONEONE
ELSE IF (ZSTART .GT. 500.0D0) THEN
    FSTART = TWOTWO ** NINT((500.0D0-100.0D0)/50.0D0)
ELSE
    FSTART = TWOTWO ** NINT((ZSTART-100.0D0)/50.0D0)
ENDIF
```

Initial position

```
ZEND = ZTRACK(NTRACK)
```

Final position

* Similarly calculate the FEND from ZEND

...

```
FIMP = FEND / FSTART
```

Importance Ratio

usimbs.f: Function example

- Introduce an importance biasing assuming an exponential law of attenuation in the R direction $\exp(-\lambda \times R)$, for $R > 1\text{cm}$

```
RSTART = SQRT(XTRACK(0)**2 + YTRACK(0)**2 + ZTRACK(0)**2)
REND = SQRT(XTRACK(NTRACK)**2 + YTRACK(NTRACK)**2 +
            ZTRACK(NTRACK)**2)
IF (RSTART .LT. ONEONE) THEN
    FSTART = ONEONE
ELSE
    FSTART = EXP(-ALAMBDA * RSTART)
ENDIF
IF (REND .LT. ONEONE) THEN
    FEND = ONEONE
ELSE
    FEND = EXP(-ALAMBDA * REND)
ENDIF
FIMP = FSTART / FEND
```

N-step method


- 2-step method explained in sources lecture
- Events from one run are dumped in a file (mgdraw.f) and used as starting particles through source.f routine
- This method is similar to importance biasing, as (selected) particles crossing a given interphase are sampled many times in the next step, which is equivalent to splitting them

Leading Particle Biasing - 1

- Leading Particle Biasing is available only for e^+ , e^- and photons
- It is generally used to avoid the geometrical increase with energy of the number of particles in the electromagnetic shower
- It is a characteristic of EM interactions that 2 particles are present in the final state
- if LPB is activated only one is randomly retained and its weight is adjusted so as to conserve weight \times probability
- The most energetic of the two particles is kept with higher probability (as it is the one which is more efficient in propagating the shower)
- LPB is reducing t , but increases σ by introducing large weight fluctuations. Therefore it should nearly always be backed up by WW
- Very useful for shielding calculations at both electron and proton accelerators

Leading Particle Biasing - 2

EMF-BIAS	1022.	0.	5.E-4	16.	20.	2.LPBEMF
----------	-------	----	-------	-----	-----	----------

 EMF-BIAS	Type: LPBEMF ▼	Ethr e-e+:	Ethr γ:
Old brems.: off ▼	Bremsstrahlung: off ▼	Pair Prod.: off ▼	e+ ann @rest: off ▼
Compton: off ▼	Bhabha&Moller: off ▼	Photo-electric: off ▼	e+ ann @flight: off ▼
	Reg: ▼	to Reg: ▼	Step:

For SDUM = LPBEMF (default):

WHAT(1) > 0.0: leading particle biasing (LPB) is activated

$$\text{WHAT}(1) = 2^0 \times b_0 + 2^1 \times b_1 + 2^2 \times b_2 + 2^3 \times b_3 + 2^4 \times b_4 + 2^5 \times b_5 + 2^6 \times b_6 + 2^7 \times b_7$$

(b0 = 1 : LPB for bremsstrahlung and pair production)

b1 = 1 : LPB for bremsstrahlung

b2 = 1 : LPB for pair production

b3 = 1 : LPB for positron annihilation at rest

b4 = 1 : LPB for Compton scattering

b5 = 1 : LPB for Bhabha & Moller scattering

b6 = 1 : LPB for photoelectric effect

b7 = 1 : LPB for positron annihilation in flight

Note: WHAT(1) = 1022 activates LPB for all physical effects

(values larger than 1022 are converted to 1022)


(FLAIR uses 254)

< 0.0: leading particle biasing is switched off

= 0.0: ignored

Leading Particle Biasing - 3

EMF-BIAS	1022.	0.	5.E-4	16.	20.	2.LPBEMF
----------	-------	----	-------	-----	-----	----------

 EMF-BIAS	Type: LPBEMF ▼	Ethr e-e+:	Ethr γ:
Old brems.: off ▼	Bremsstrahlung: off ▼	Pair Prod.: off ▼	e+ ann @rest: off ▼
Compton: off ▼	Bhabha&Moller: off ▼	Photo-electric: off ▼	e+ ann @flight: off ▼
	Reg: ▼	to Reg: ▼	Step:

WHAT(2) > 0.0: energy threshold below which LPB is played for electrons and positrons

electrons: kinetic energy

positrons: total energy plus rest mass energy

< 0.0: resets any previously defined threshold to infinity (i.e., LPB is played at all energies, Default)

= 0.0: ignored

WHAT(3) > 0.0: energy threshold below which LPB is played for photons

< 0.0: resets any previously defined threshold to infinity (i.e., LPB is played at all energies, Default)

= 0.0: ignored

WHAT(4) = lower bound of the region indices/names (Default = 2.0)

WHAT(5) = upper bound of the region indices/names (Default = WHAT(4))

WHAT(6) = step length in assigning indices/names (Default = 1.0)

Multiplicity tuning - 1

- **Multiplicity tuning** is meant to be for hadrons what Leading Particle Biasing is for electrons and photons.
- A hadronic nuclear interaction at LHC energies can end in hundreds of secondaries. Thus, to simulate a whole hadronic cascade in bulk matter may take a lot of CPU time.
- Except for the leading particle, many secondaries are of the same type and have similar energies and other characteristics.
- Therefore, it is possible to **discard a predetermined average fraction** of them, provided the weight of those which are kept and transported be adjusted so that the total weight is conserved (*but the leading particle is never discarded*).
- The user can tune the average multiplicity in different regions of space by setting a region-dependent reduction factor (*in fact, it can even be > 1 ! But this possibility is seldom used*).

Multiplicity tuning - 2

BIASING	1.0	0.7	1.0	8.	18.	1.
---------	-----	-----	-----	----	-----	----

 BIASING	Type: All particles ▼	RR: 1.0	Imp: 0.000189
Opt: ▼	Reg: Box ▼	to Reg: Sphere ▼	Step: 1.0

WHAT(1) specifies the particles to be biased with **WHAT(3)** (*see before*)

- = 0.0 : all particles
- = 1.0 : hadrons and muons
- = 2.0 : electrons, positrons and photons

WHAT(2) = RR (or splitting) factor by which the average number of secondaries produced in a collision should be reduced (or increased). (Default = 1.0) – No RR if less than 3 secondaries

WHAT(3) = (*see importance biasing*)

WHAT(4) = lower bound of the region indices/names (Default = 2.0)

WHAT(5) = upper bound of the region indices/names (Default = WHAT(4))

WHAT(6) = step length in assigning indices/names (Default = 1.0)

SDUM = **RRPRONLY**: multiplicity biasing for primary particles only
 = **(blank)**: ignored (Default)

Biassing mean free paths - 1

Decay lengths:

- The **mean life/average decay length** of unstable particles can be artificially shortened.
- It is also possible to **increase the generation rate** of decay products without the parent particle actually disappearing.
- Typically used to enhance statistics of **muon or neutrino production**.
- The kinematics of the decay can also be biased (decay angle).

Biassing mean free paths - 2

LAM-BIAS	-3.E+3	1.	1.	13.	16.	0.GDECAY
----------	--------	----	----	-----	-----	----------

 LAM-BIAS	Type: GDECAY ▼	x $\langle\lambda\rangle$:	x λ inelastic:
Mat: ▼	Part: ▼	to Part: ▼	Step:


for SDUM = GDECAY:

WHAT(1) : mean decay length (cm) of the particle in the laboratory frame is set = |WHAT(1)| if smaller than the physical decay length (otherwise it is left unchanged).

- < 0.0 : At the decay point **Russian Roulette (i.e. random choice)** **decides whether the particle actually will survive or not** after creation of the decay products. The latter are created in any case and their weight adjusted taking into account the ratio between biased and physical survival probability.
- > 0.0 : Let P_u = unbiased probability and P_b = biased probability: at the decay point **the particle always survives with a reduced weight $W=(1-P_u/P_b)$** . Its daughters are given a weight **$W=P_u/P_b$** (as in case $WHAT(1) < 0.0$).

Biasing mean free paths - 3

LAM-BIAS	-0.5	1.	1.	13.	16.	0.
----------	------	----	----	-----	-----	----

 LAM-BIAS	Type: ▼	x mean life:	x λ inelastic:
Mat: ▼	Part: ▼	to Part: ▼	Step:

for SDUM = blank:

WHAT(1) : the mean life of the particle in its rest frame is reduced by a factor |WHAT(1)| (must be ≤ 1.0)

< 0.0 : (as for SDUM=GDECAY) **Russian Roulette**

> 0.0 : (as for SDUM=GDECAY) **the particle always survives with a reduced weight**

for SDUM = blank or GDECAY

WHAT(2) and **WHAT(3)** : (see interaction length biasing)

WHAT(4) = lower bound of the particle index (Default = 1.0)

WHAT(5) = upper bound of the particle index
(Default = WHAT(4) if WHAT(4) > 0, 46 otherwise)

WHAT(6) = step length in assigning indices (Default = 1.0)


Biasing mean free paths - 4

Interaction lengths:

- In a similar way, the hadron or photon **mean free path for non-elastic nuclear interactions** can be artificially decreased by a predefined particle or material-dependent factor.
- This option is useful for instance to increase the probability for beam interaction in a **very thin target** or in a material of **very low density**.
- It is also **necessary to simulate photonuclear reactions** with acceptable statistics, the photonuclear cross section being much smaller than that for EM processes.

Biassing mean free paths - 5

LAM-BIAS	0.0	0.02	11.	7.	0.	0.
----------	-----	------	-----	----	----	----

 LAM-BIAS	Type: ▼	x mean life:	x λ inelastic:
Mat: ▼	Part: ▼	to Part: ▼	Step:

WHAT(1) : *(see decay length biasing)*


WHAT(2) : **biassing factor for hadronic inelastic interactions**

The hadronic inelastic interaction length of the particle is reduced by a factor |WHAT(2)| (must be ≤ 1.0)

- < 0. : At the interaction point **Russian Roulette (i.e. random choice)** **decides whether the particle actually will survive or not** after creation of the secondaries products. The latter are created in any case and their weight adjusted taking into account the ratio between biased and physical survival probability.
- > 0. : At the interaction point **the particle always survives with a reduced weight.** The secondaries are created in any case and their weight adjusted taking into account the ratio between biased and physical survival probability.

Biassing mean free paths - 6

LAM-BIAS	0.0	0.02	11.	7.	0.	0.
----------	-----	------	-----	----	----	----

 LAM-BIAS	Type: ▼	x mean life:	x λ inelastic:
Mat: ▼	Part: ▼	to Part: ▼	Step:

WHAT (3) : If > 2.0 : number of the material to which the inelastic biasing factor has to be applied.
 < 0.0 : resets to the default a prev. assigned value
 $= 0.0$: ignored if a value has been previously assigned to a specific material, otherwise all materials
 $0.0 < \text{WHAT}(3) \leq 2.0$: all materials
(Default = 0.0)

WHAT (4) = lower bound of the particle indices/names (Default = 1.0)

WHAT (5) = upper bound of the particle indices/names
(Default = WHAT(4) if WHAT(4) > 0 , 46 otherwise)

WHAT (6) = step length in assigning indices (Default = 1.0)

Biassing can be applied to photons and muons if **PHOTONUC** and **MUPHOTON** cards are included.

Weight Windows - 1

- The WW technique is a combination of splitting and RR, but it is based on the **absolute value** of the weight of each **individual particle**, rather than on relative region importance
- The user sets **an upper and a lower weight limit**, generally as a function of region, energy and particle
- Particles having a **weight larger than the upper limit are split**, those with weight **smaller than the lower limit are submitted to RR** ⇒ killed or **put back** "inside the window"
- WW is a more **powerful** biasing tool than Importance Biasing, but it requires also more experience and patience to set it up correctly
"It is more an art than a science" (From MCNP Manual)
- Use of the WW is essential whenever other biasing techniques generate large weight fluctuations in a given phase space region.
- For **Low Energy neutrons** the energy range should be defined through **WW-PROFILE** (instead of **WW-THRESH**)

Weight Windows - 2

Killing a particle with a very low weight (with respect to the average for a given phase space region) decreases t but has very little effect on the score (and therefore on σ)

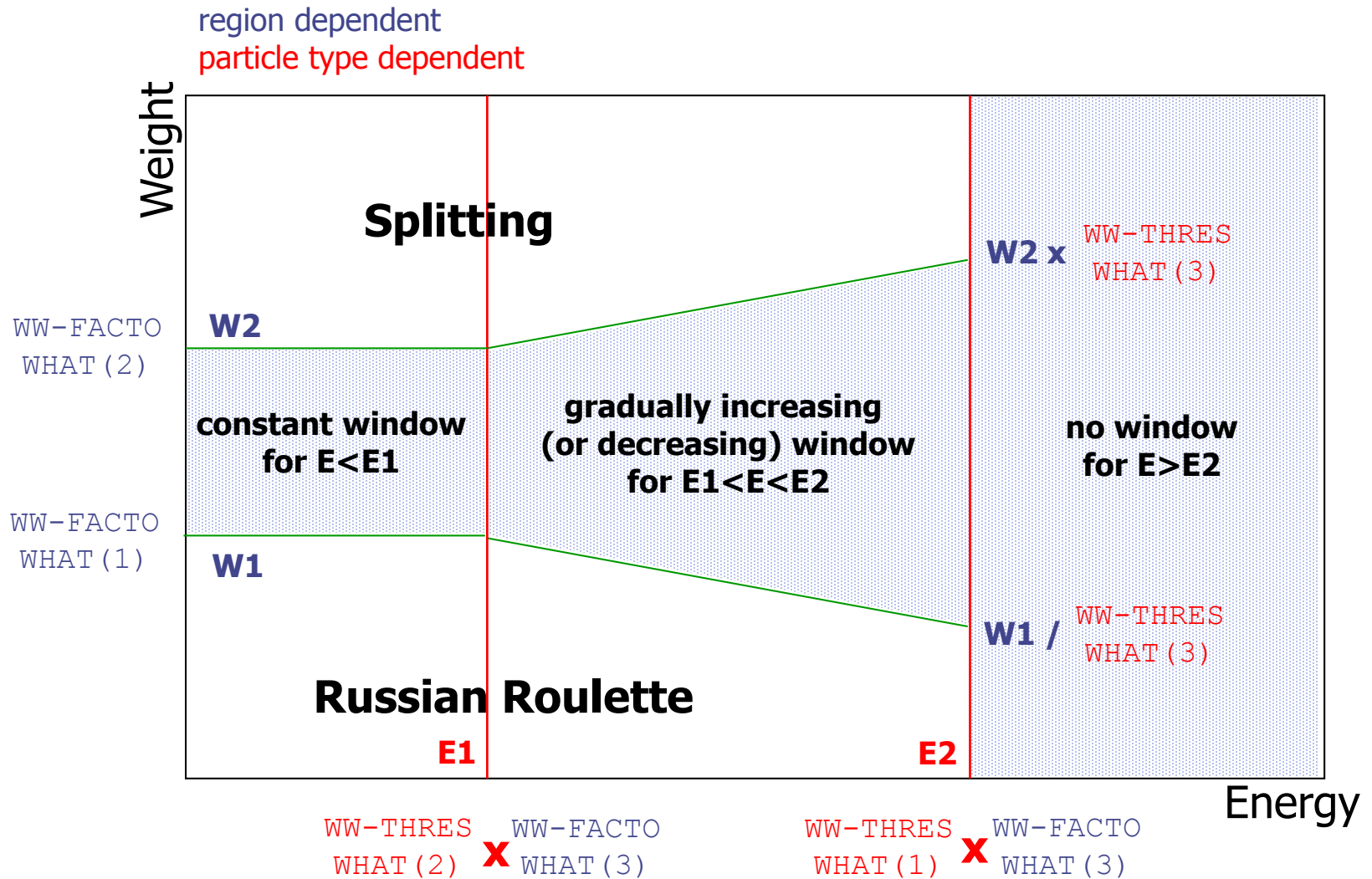
Splitting a particle with a large weight *increases* t (in proportion to the number of additional particles to be followed) but at the same time *reduces* σ by avoiding large fluctuations in the contributions to scoring.




The global effect is to reduce $\sigma^2 t$

A too wide window is of course ineffective, but also too narrow windows should be avoided. Otherwise, too much CPU time would be spent in repeated splitting / Russian Roulette. *A typical ratio between the upper and the lower edge of the window is about 10.* It is also possible to do Russian Roulette without splitting (setting the upper window edge to infinity) or splitting without Russian Roulette (setting the lower edge to zero)

Weight Windows - 3



Weight Windows - 4

WW-FACTO	13.0	120.0	1.5	27.0	31.0	2.0
 WW-FACTO LowE n: ▼	RR: Reg: ▼	Split: to Reg: ▼	mult f: Step:			

Defines Weight Windows in selected regions

WHAT(1) ≥ 0.0 : Window "bottom" weight
 < 0.0 : resets to -1.0 (no Russian Roulette, Default)

Weight below which *Russian Roulette* is played at the lower energy threshold (set by WW-THRES).


WHAT(2) $> 1.7 * \text{WHAT}(1)$: Window "top" weight
 $= 0.0$: ignored
 $\leq 1.7 * \text{WHAT}(1)$: resets to infinity (no Splitting, Default)

Weight above which *Splitting* is applied at the lower energy threshold (set by WW-THRES).

WHAT(3) > 0.0 : Multiplicative factor (Default: 1.0)
 $= 0.0$: ignored
 < 0.0 : resets to 1.0

Factor to be applied to the two energy thresholds for Russian Roulette / Splitting (set by WW-THRES)

Weight Windows - 5

WW-FACTO	13.0	120.0	1.5	27.0	31.0	2.0
 WW-FACTO	RR:	Split:	mult f:			
LowE n: ▼	Reg: ▼	to Reg: ▼	Step:			

WHAT (4) = lower bound of the region indices/names (Default = 2.0)

WHAT (5) = upper bound of the region indices/names (Default = WHAT(4))

WHAT (6) = step length in assigning indices/names (Default = 1.0)

SDUM : a number from 1.0 to 5.0 in any position, indicating the **low-energy neutron weight-window profile** to be applied in the regions selected


(see WW-PROFI). (Default = 1.0)

= blank, zero or non numerical: ignored

< 0.0 : resets to 1.0

Attention: Option WW-FACTO alone is not sufficient to define a weight window.
One or more WW-THRES cards are also necessary in order to activate the window.

Weight Windows - 6


WW-THRES	2.0	0.05	2.4	3.0	7.0	0.0
 WW-THRES Opt: ▼	E upper: Part: ▼	E lower: to Part: ▼	amp f: Step:			

- WHAT(1) > 0.0: **upper kinetic energy threshold (GeV)**
 Low-energy neutrons: lower group number (included)
 = 0.0: **ignored**
 < 0.0: **any previously selected threshold is cancelled**
- WHAT(2) >= 0.0 and < WHAT(1): **lower kinetic energy threshold (GeV)**
 Low-energy neutrons: upper group number (included)
 < 0.0 or > WHAT(1): **WHAT(2) is set = WHAT(1)**
- WHAT(3) > 0.0: **amplification factor** to define the weight window width
 at the higher energy threshold represented by WHAT(1).

The weight window at the higher energy threshold is obtained by multiplying by WHAT(3) the upper weight limit and by dividing by the same factor the lower weight limit. (Default = 10.0)

- < 0.0: **|WHAT(3)| multiplication factor** for the lower and upper weight limits for the particles selected by WHAT(4-6)
 (Default = 1.0)

Weight Windows - 7

WW-THRES	2.0	0.05	2.4	3.0	7.0	0.0
 WW-THRES Opt: ▼	E upper: Part: ▼	E lower: to Part: ▼	amp f: Step:			

WHAT(4) = lower bound of the particle indices (Default = 1.0)

Note that particle index 40 indicates low-energy neutrons (for this purpose only!). Particle index 8 indicates neutrons with energy > 19.6 MeV.

WHAT(5) = upper bound of the particle indices
(Default = WHAT(4) if WHAT(4) > 0, all particles otherwise)

WHAT(6) = step length in assigning indices (Default = 1.0)

SDUM = PRIMARY: the weight window applies also to primary particles (default)
= NOPRIMARY: the weight window doesn't apply to primaries

Selecting Weight Windows - 1

BIASING	0.0	0.0	4.64	8.	18.	2. PRINT
---------	-----	-----	------	----	-----	----------

BIASING	Type: ▼	RR:	Imp: 0.000189
Opt: PRINT ▼	Reg: ▼	to Reg: ▼	Step:

FLUKA output file:

```
Hadron importance RR/Splitting counters
```

Reg. #	N. of RR	<Wt> in	<Wt> kil	Reg. #	N. of RR	<Wt> in	<Wt> kil	Reg. #	N. of RR	<Wt> in	<Wt> kil
1	0.00E+00	0.00E+00	0.00E+00	2	0.00E+00	0.00E+00	0.00E+00	3	1.15E+05	9.31E-01	4.70E-02
Reg. #	N. of Sp	<Wt> in	<Wt> out	Reg. #	N. of Sp	<Wt> in	<Wt> out	Reg. #	N. of Sp	<Wt> in	<Wt> out
1	0.00E+00	0.00E+00	0.00E+00	2	0.00E+00	0.00E+00	0.00E+00	3	0.00E+00	0.00E+00	0.00E+00
Reg. #	N. of RR	<Wt> in	<Wt> kil	Reg. #	N. of RR	<Wt> in	<Wt> kil	Reg. #	N. of RR	<Wt> in	<Wt> kil
4	1.36E+04	4.66E-01	1.47E-01	5	8.97E+03	3.22E-01	1.06E-01	6	6.03E+03	2.16E-01	7.10E-02
Reg. #	N. of Sp	<Wt> in	<Wt> out	Reg. #	N. of Sp	<Wt> in	<Wt> out	Reg. #	N. of Sp	<Wt> in	<Wt> out
4	1.01E+05	9.99E-01	7.64E-01	5	9.25E+04	6.80E-01	5.23E-01	6	8.24E+04	4.65E-01	3.55E-01

"N. of RR" --> Number of FLUKA particles entering a region and which are not split (i.e., particles undergoing Russian Roulette as well as neither Russian Roulette nor splitting)

"<Wt> in" --> Average weight of these particles

"<Wt> kil" --> Average weight of particles killed after being submitted to Russian Roulette

"N. of Sp" --> Number of FLUKA particles entering the region and which are split

"<Wt> in" --> Average weight of these particles

"<Wt> out" --> Average weight of particles after being submitted to splitting

Selecting Weight Windows - 2

where

$$A = \text{"N. of RR"} + \text{"N. of Sp"} \\ = \textit{total number of particles entering the region}$$

$$B = (\text{"<Wt> in"}_{\text{RR}} * \text{"N. of RR"}) + (\text{"<Wt> in"}_{\text{Sp}} * \text{"N. of Sp"}) \\ = \textit{total weight of the particles entering the region}$$

$$B/A = \textit{average weight of the particles entering the region}$$

Note -1: RR and splitting arising from Weight-Window biasing (options WW-FACTOR, WW-THRESH, WW-PROFI) or from multiplicity biasing (WHAT(2) in option BIASING) are not accounted for in the counters.

Note - 2: Separate counters are printed for hadrons/muons, electrons/photons and low-energy neutrons (referring to importance biasing requested by BIASING, respectively, with WHAT(1) = 1.0, 2.0 and 3.0, or = 0.0 for all).

Selecting Weight Windows - 3

Strategy:

1. run without any biasing and print counter, *e.g.*,

```
BIASING      0.0      1.0      1.0      1.      9.      PRINT
```

2. analyze counter and adjust region importance biasing, *e.g.*, according to the inverse of the attenuation in shielding, add other biasing, *e.g.*, leading particle biasing run and print counter again

```
BIASING      0.0      1.0      1.0      1.      9.      PRINT
BIASING      0.0      1.0      1.47     4.
BIASING      0.0      1.0      2.15     5.
BIASING      0.0      1.0      3.16     6.
BIASING      0.0      1.0      4.64     7.
BIASING      0.0      1.0      4.64     8.
```

3. analyze counter, select Weight Windows (WW-THRES, WW-FACTO) around average weights and perform final (high-statistics) run

Selecting Weight Windows - 4

Alternative Strategy:

1. Include USERDUMP card in input file to activate call to **mgdraw.f**

```
USERDUMP      100.0      25.0      4.0      1.
```

2. Edit mgdraw.f (bxdraw) to print energy and weight of particles (filter by particle if wanted) entering specific areas of the geometry, e.g.

```
IF ( .NOT. LFCOPE) THEN
  LFCOPE = .TRUE.
  OPEN (UNIT = 86, FILE = "tuneWW.dat", STATUS = "UNKNOWN")
END IF
. . .
IF (MRGNAM.eq."OLDCELL".and.NRGNAM.eq."NEWCELL".and.jtrack.eq.3) THEN
  write(86,*)wtrack, etrack
END IF
. . .
```

3. Produce scatter plot weight(E) by loading file "tuneWW.dat" in **gnuplot**
gnuplot> plot 'tuneWW.dat' using 2:1

3. Tune WW accordingly

Non analogue absorption

- Implemented in most **low-energy** neutron transport codes, where at each neutron collision the neutron always survives with its weight multiplied by the physical survival probability σ_s/σ_t
- In FLUKA there is an additional choice: the user **can force** the neutron absorption probability to take an **arbitrary value**, pre-assigned on a region-by-region basis as a function of energy. The neutron weight is properly normalized by the code accordingly.

When and How

- A **smaller survival probability** is often assigned to **thermal neutrons** to limit the number of scatterings in non-absorbing media
- Also very useful in materials with **unusual scattering properties** (e.g. Iron)
- Survival probabilities **too small** with respect to the physical one σ_s/σ_t may introduce large weight fluctuations due to the very different number of collisions suffered by individual neutrons. In these cases a WW should be applied.
- *Also called **survival biasing***

Card: RADDECAY [1/2]

* 1) request radioactive decays

```
RADDECAY      1.0      0      3.0      0000099999      0
```

RADDECAY	Decays: Active ▼	Patch Isom: ▼	Replicas: 3.0
h/μ Int: ignore ▼	h/μ LPB: ignore ▼	h/μ WW: ignore ▼	e-e+ Int: ignore ▼
e-e+ LPB: ignore ▼	e-e+ WW: ignore ▼	Low-n Bias: ignore ▼	Low-n WW: ignore ▼
	decay cut: 0.0	prompt cut: 99999.0	Coulomb corr: ▼

- WHAT(1)** = 1 **radioactive decays activated for requested cooling times**
 Decays: Active "activation study case": time evolution calculated analytically for *fixed* (cooling) times. Daughter nuclei as well as associated radiation is considered at these (fixed) times
- > 1 **radioactive decays activated in semi-analogue mode**
 Semi-Analogue each radioactive nucleus is treated like all other unstable particles (random decay time, daughters and radiation), all secondary particles/nuclei carry time stamp ("age")
- WHAT(2)** > 0 **isomer "production" activated**
 Patch Isom: On
- WHAT(3)** **number of "replicas" of the decay of each individual nucleus**
 Replicas: #

Card: RADDECAY [2/2]

RADDECAY

h/μ Int: ignore ▼
e-e+ LPB: ignore ▼

Decays: Active ▼
h/μ LPB: ignore ▼
e-e+ WW: ignore ▼
decay cut: 0.0

Patch Isom: ▼
h/μ WW: ignore ▼
Low-n Bias: ignore ▼
prompt cut: 99999.0

Replicas: 3.0
e-e+ Int: ignore ▼
Low-n WW: ignore ▼
Coulomb corr: ▼

WHAT(4)

h/m Int .. Low-n WW

switch for applying various biasing features only to prompt radiation or only to particles from radioactive decays

9 digits, each responsible for a different biasing

Example:

5th digit, e+/e-/gamma leading particle biasing applied
000010000 to prompt radiation only
000020000 to decay radiation only
000030000 to both

Default: 11111111 (or blank as above)

WHAT(5)

decay cut: #
prompt cut: #

multiplication factors to be applied to transport cutoffs
10 digits, first five for decay radiation, second five for prompt radiation (see manual)

Special cases:

0000099999 kill EM cascade for prompt radiation
9999900000 kill EM cascade for residual radiation

Source.f: Biasing distribution of primaries

- Low occurrence primaries may have important consequences on the final response, e.g. high energy tail of synchrotron radiation can generate neutrons (i.e. activation)
- Actual distributions, e.g. $f(E)$, can be biased (i.e. distorted) by a function, e.g. $b(E) = c * E$
→ $g(E) = f(E) * b(E)$
- Whether $g(E)$ is sampled through inverse transform or through rejection, starting weights of particles **WTFLK(NPFLKA)** should then be adjusted by $*1/fb$
- Use with moderation/care...this can introduce strong weight dispersion → weigh-windows to split high-weight particles...

udcdrl.f: Neutrino Decay Biasing

- There is a special routine `udcdrl.f` where one can bias the direction of the emitted neutrino in decays.

DOUBLE PRECISION **FUNCTION** `UDCDRL`(IJ, KPB, NDCY, UDCDRB, VDCDRB, WDCDRB)

Input variables:

IJ	decaying particle
KPB	outgoing neutrino

Output variables:

U,V,W DCDRB	preferential outgoing direction for the neutrino
UDCDRL	Lambda for direction biasing ($1-\cos(\theta)$)

The biasing expression is of the form:

$$e^{-(1-\cos\theta/\lambda)}$$

- Useful for neutrino applications like CNGS, Beta Beams...
- For a fixed direction the `LAM-BIAS` card with `SDUM=DCY-DIRE` could be used instead
- **NOTE: angular biasing is only available (and mostly only makes sense) for neutrinos due to their low interaction cross-sections**

udcdrl.f: Example (n_TOF)

- Bias the direction of the neutrino in the pion decay so the daughter muon to be directed to the exp. area @(-120,0,18500). The direction is given in the lab-frame, and in this example the energies we are dealing are small, so safely we can assume that also in the lab-frame, the neutrino and muon go in opposite directions. The lambda is 1/4 wide enough to cover the whole exp area.

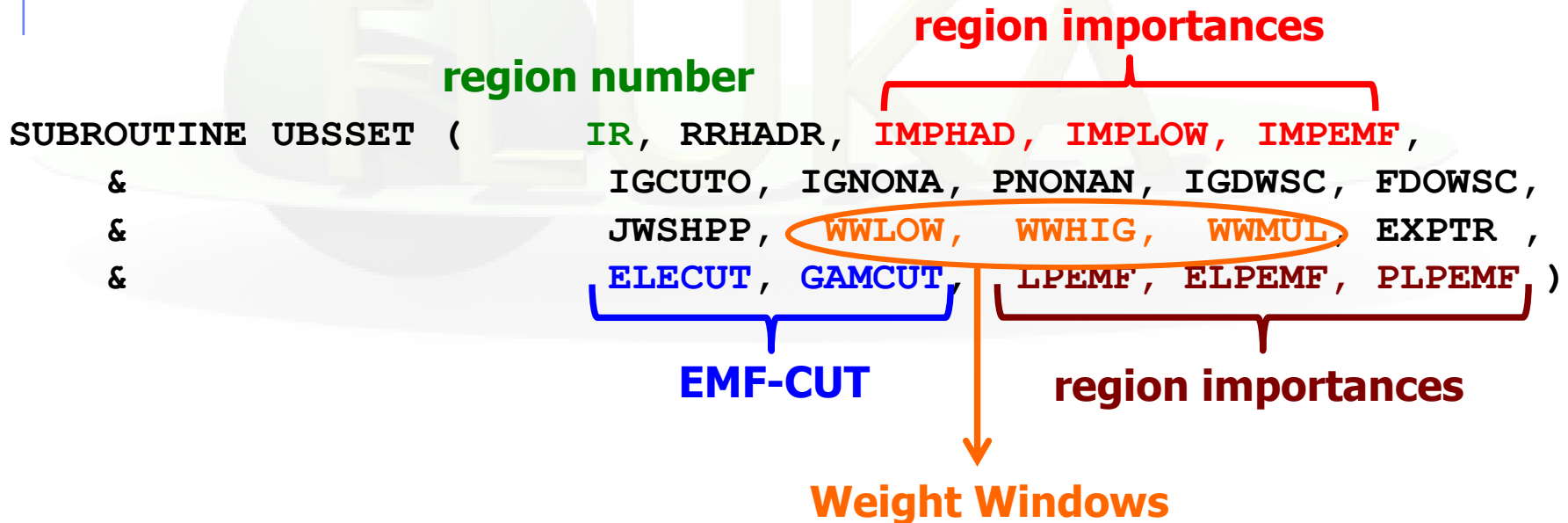
```
INCLUDE '(TRACKR)'  
*  
UDCDRB = XTRACK (NTRACK) + 120.D+00  
VDCDRB = YTRACK (NTRACK) - ZERZER  
WDCDRB = ZTRACK (NTRACK) - 18500.D+00  
HLPHLP = SQRT ( UDCDRB**2 + VDCDRB**2 + WDCDRB**2 )  
UDCDRB = UDCDRB / HLPHLP  
VDCDRB = VDCDRB / HLPHLP  
WDCDRB = WDCDRB / HLPHLP  
UDCDRL = ONEFOU  
*
```

Direction in the lab frame

Width [1-cos(θ)]

usbset.f: User Biasing SETting

- called after reading in the input file and before the first event
- allows to alter almost any biasing weight on a region-dependent basis



Other optimizations - 1

- **Raising EMF thresholds (EMF-CUT)...**

- in regions far from the scoring area or deep into the shielding
- in heavy materials (at least 100-200 keV, unless in single scattering mode)
- to GDR threshold (5-15 MeV) for neutron-dominated problems
- to muon production threshold ($> \sim 100$ MeV) for muon-dominated problems

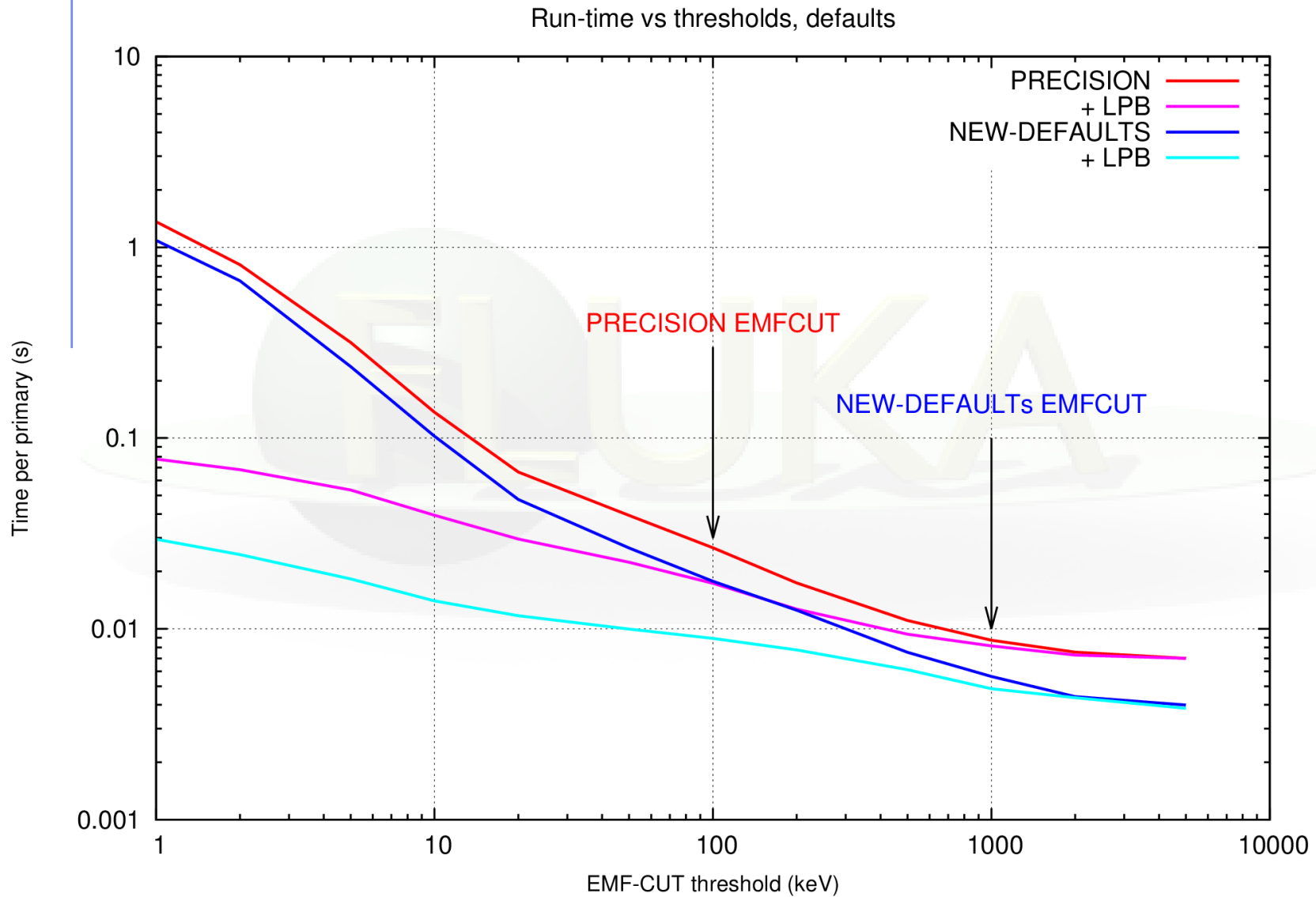
Notes:

- These techniques will help get a faster estimate with relatively good accuracy
- The sensitivity of results to EMF-CUT changes should be tested to assess biasing effect on results
- In many cases further simulations with lower thresholds are not needed

WARNING:

- Rather than optimization, raising thresholds is **OFTEN** a user mistake as done irrespectively of the relevant physical processes

Example: CPU vs Thresholds and LPB



Other optimizations - 2

- **Data initialization in user routines:** Avoid unnecessary repetition of global initialization steps at each call of a user routine
 - Use one-time check with logical variable, e.g.

```
IF (LFIRST) THEN
...
LFIRST = .FALSE.
END IF
```
 - Example 1: number to name conversion: generate conversion table, save and use thereafter
 - Example 2: preliminary mathematical transformations before sampling in source routine
- **Move instructions that are not updated in loops out of those**

Other optimizations - 3

- **Using symmetries for mesh scoring**, e.g. $|x|$, $|y|$, $|z|$, $|r|$, r - ϕ - z , r - ϕ - $|z|$, r - z , r - $|z|$

Notes:

- For example a symmetry over one plane saves 40% CPU time
- Symmetry can also be forced through `usrmed.f` (activated by option `MAT-PROP` with `SDUM USERDIRE`).
- **Rebinning** 1-D and 2-D to coarser mesh
- **Data writing**: dumping or writing large amount of data can considerably slow down calculations.
 - In large input files, and inherited setups, deactivate unnecessary legacy scoring, specially fine grid 3D histograms (e.g. USRBIN)
 - **Avoid using 3D histograms** (large size scoring), use instead 2D histograms (scoring in a plane) or 3D histograms with a coarse grid in one of the axis
 - Limit tape writing to necessary. Optimize the mathematical expressions in `mgdraw.f`
 - **Tune-up scoring cards** with short runs (good practice for several aspects) to optimize scoring scales

Other optimizations - 4

- **Geometry optimization**

- Adjust **NAZ** in region definition to make it equal or slightly larger than the number of neighboring regions. Check for for "GEOMETRY SEARCH ARRAY" warnings in output
- If a region has too many neighboring cells, subdivide it in zones or regions to lower number of neighboring cells per zone
- If there are components (with sub-structure) that are repeated many times in the geometry use **LATTICE** capabilities to reduce number of regions

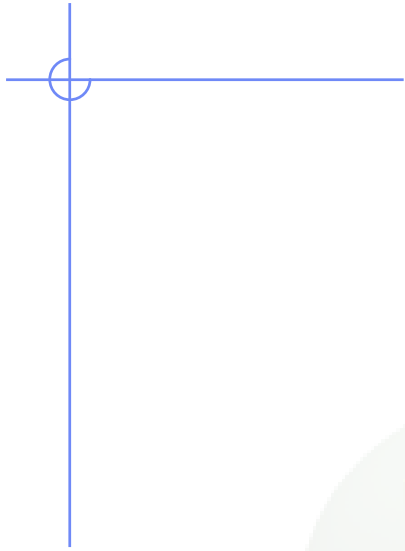
- **More:** Biasing impurities in compounds, number of replica in activation/decay...

- **Other checks:** these are *not* ON by default, so check only if you are recycling an input file or you are using it for a different application *and* CPU is an issue:

- Make sure single EMF scattering (**MULSOPT**) is not ON unless needed (low energy, thin geometry)
- If not looking at activation, heavy ion evaporation does *not* need to be switched ON (*WHAT(2)* of **PHYSICS** card with SDUM EVAPORAT)

Warnings...

- **BIASING** focuses the CPU onto a phase-space area of the problem, sacrificing other areas
- Reducing simulation time does not necessarily mean the simulation has been optimized
- Similarly, reducing variance does not necessarily mean the simulation has been optimized
- Small statistical error does not mean the error is actually low. This is specially true if splitting / two step methods, etc. are used. Correlations may be hidden.
- Biasing may reduce the error in the estimation of the *average* of a variable, but the higher momenta of that variable will not be obtained (e.g. the variance of the PDF distribution for that variable cannot be estimated).
- **Simulation optimization may save CPU time, but it can cost considerable manpower**





Biasing

Beginners' FLUKA Course

usimbs.f: Sampling from array [1/4]

- Create a concentric cylindrical biasing with the weights sample from an array with various radii

- Define the variables:

```
PARAMETER (NBIAS=5)
```

```
PARAMETER (Xcenter=ZERZER)
```

```
PARAMETER (Ycenter=ZERZER)
```

```
DOUBLE PRECISION BIASR(NBIAS), BIASF(NBIAS)
```

* Radius

```
DATA BIASR / 200.0, 250.0, 300.0, 400.0, 500.0 /
```

* Biasing factor

```
DATA BIASF / NBIAS * 2.0 /
```

```
LOGICAL LFIRST
```

```
DATA LFIRST / .TRUE. /
```

```
SAVE LFIRST, BIASR, BIASF
```

usimbs.f: Sampling from array [2/4]

- Initialization – build the cumulative importance factor

```
IF (LFIRST) THEN
```

```
  WRITE(LUNOUT,*) "*** User defined biasing ***"
```

```
  PREVBIAS = 1.0
```

```
  DO N=1,NBIAS
```

```
    PREVBIAS = PREVBIAS * BIASF(N)
```

```
    BIASF(N) = PREVBIAS
```

```
    WRITE(LUNOUT,*) "Bias cylinder: ",N, BIASR(N), BIASF(N)
```

```
*      convert the radius to square to avoid sqrt
```

```
    BIASR(N) = BIASR(N)**2
```

```
  ENDDO
```

```
  LFIRST = .FALSE.
```

```
ENDIF
```

usimbs.f: Sampling from array [3/4]

- Calculate the importance biasing

- * Find square of radius for starting/ending position

$$Rold = (Xtrack(0)-Xcenter)**2 + (Ytrack(0)-Ycenter)**2$$

$$Rnew = (Xtrack(Ntrack)-Xcenter)**2 + (Ytrack(Ntrack)-Ycenter)**2$$

- * Search index of the starting position

$$Nold = NBinSearch(Rold, NBIAS, BIASR)$$

IF (Nold.EQ.0) THEN

$$BIASOLD = 1.0$$

ELSE

$$BIASOLD = BIASF(Nold)$$

ENDIF

- * Search index of the new position

$$Nnew = NBinSearch(Rnew, NBIAS, BIASR)$$

IF (Nnew.EQ.0) THEN

$$BIASNEW = 1.0$$

ELSE

$$BIASNEW = BIASF(Nnew)$$

ENDIF

$$FIMP = BIASNEW / BIASOLD$$

usimbs.f: Sampling from array [4/4]

Perform a binary search
Converge in $\log_2(N)$ steps

```
INTEGER FUNCTION NBinSearch(x, N, VEC)
INCLUDE '(DBLPRC)'
DOUBLE PRECISION VEC(N)
NLOW = 1
NHIGH = N
NBinSearch = 0
IF (X.GE.VEC(NLOW) .AND. X.LE.VEC(NHIGH)) THEN
10    CONTINUE
      MID = (NLOW+NHIGH)/2
      IF (MID.EQ.NLOW) THEN
          NBinSearch = MID
          RETURN
      ELSEIF (X .GT. VEC(MID)) THEN
          NLOW = MID
      ELSEIF (X .LT. VEC(MID)) THEN
          HIGH = MID
      ELSE
          NBinSearch = MID
          RETURN
      ENDIF
      GOTO 10
END IF
END
```