LA-UR-08-2330

Approved for public release;
distribution is unlimited.

Title: Implementation of Pulse Height Tally Variance Reduction in MCNP5

Author(s): Jeffrey S. Bull
Thomas E. Booth
Avneet Sood

Intended for:

Form 836 (7/06)

LA-UR-08-2330

Implementation of Pulse Height Tally Variance Reduction in MCNP5

Jeffrey S. Bull & Thomas E. Booth

MCNP5 1.50 Release – Supporting Document

July 9, 2008

Los Alamos NATIONAL LABORATORY EST. 1943

Operated by the Los Alamos National Security, LLC for the DOE/NNSA

NNSA

Abstract

One of the new features in MCNP5 1.50 is pulse height tally variance reduction (PHTVR). These viewgraphs summarize how PHTVR was implemented in MCNP and the modifications to the code needed to support PHTVR.

Outline

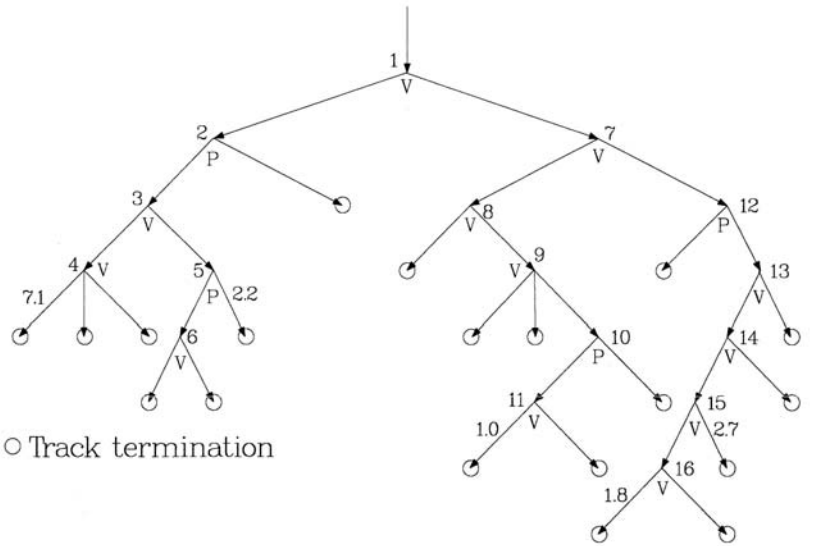
- Introduction
- Building the pulse height variance reduction trees
- Scoring the pulse height tally (debranching)
- Implementation with OpenMP

Pulse Height Tallies and Variance Reduction

- History
  - Theory developed by Tom Booth
    - *Monte Carlo Variance Reduction Approaches for Non-Boltzmann Tallies*, LA-12433 (1992)
    - *Pulse Height Tally Variance Reduction in MCNP*, LA-13955 (2004)
  - Partial implementation previously in MCNPX
- Pulse height (F8) tallies depend on collections of particles (for example, the entire particle history)
- Example: Two 1 MeV photons are deposited in one cell
  - F1 tally scores two 1 MeV events
  - F8 tally scores one 2 MeV event
    - Assumes both photons have weight one.
      - **Not** valid if using variance reduction

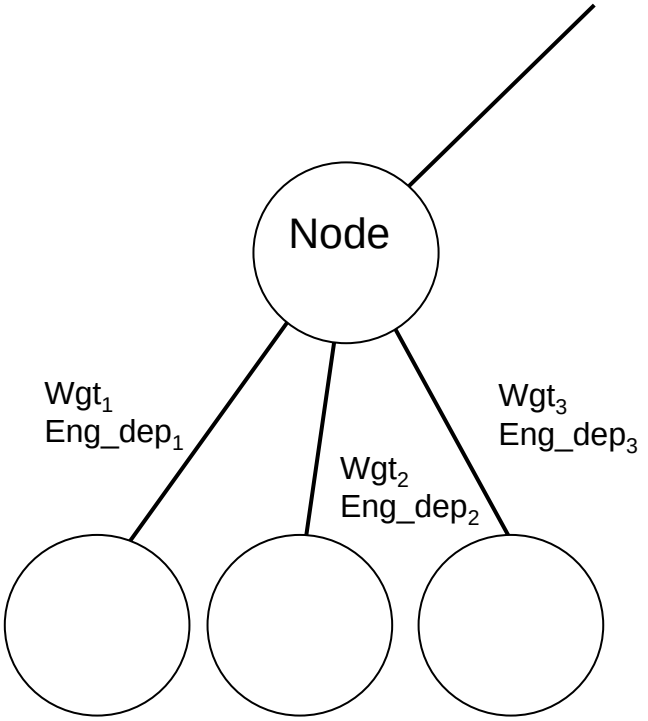
Track History Trees

- Need to store all path weights, particle production, and energy deposition events for the whole track history.
- Create “trees” which recreate the track history
- Tree nodes represent events (importance splitting, particle creation, etc)
- Path weights and energy deposition are associated with the tree branches. They record changes between nodes (events).
- At the end of the history, the tree is “debranched” into possible physically occurring subtrees, and energy deposition is calculated for each possible physically occurring subtree



Tree Nodes Types

- Physical Node
  - Represent particle creation and other physical (real) events
  - All branches are followed as part of the history
  - When debranching, subtrack must include all branches of the node
- Variance Reduction Node
  - Represents importance splitting and other variance reduction events
  - Each branch represents an alternative path the particle could take
  - When debranching, each branch of the node creates a new subtrack
- Terminate Node
  - Represents particle termination event
  - There are no branches under these nodes.



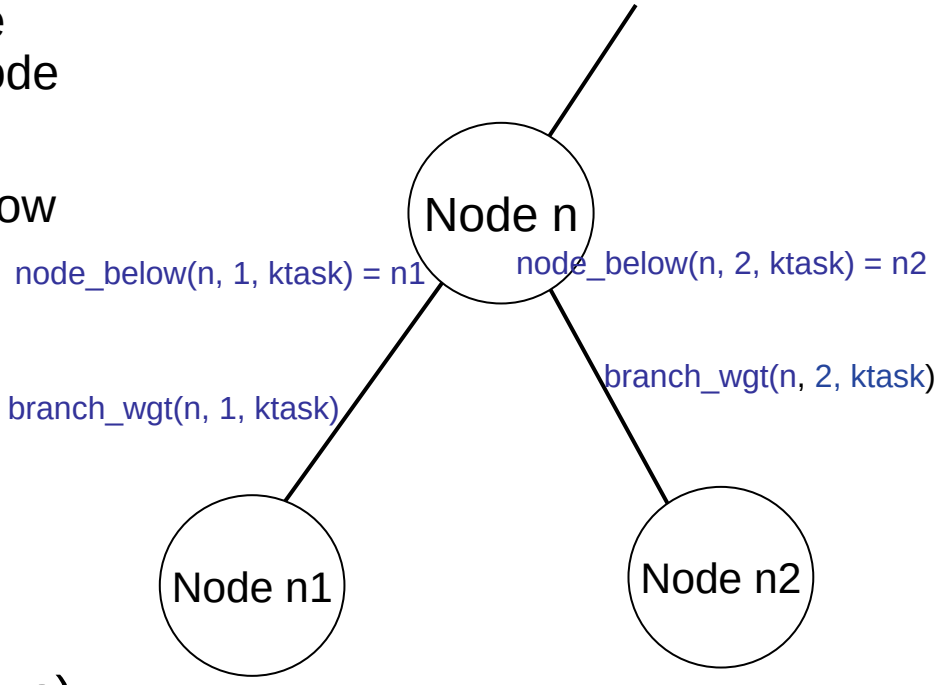
Code development

- Development History
  - Tom Booth did the original coding
  - Avneet Sood updated it to MCNP5 1.22
  - Jeff Bull updated it to current version of MCNP5

- Code Development Goals
  - Produce same results (within statistics) as analog case
  - Minimize the memory requirements
  - Execute efficiently
  - Conform to modern coding style

Building the Trees, Node by Node

- To better manage the array sizes, the maximum number of branches per node is 2.
- During transport, the particle must know where it is in relation to the tree.
  - Added 2 variables to PBLCOM
    - node_above
    - branch
- Variable n_nodes
  - Number of nodes in the history



- Tree Arrays (all are dynamic arrays)
  - n_branches(n_nodes,ktask) (i1_knd) - number of branchs, & node type (+ = var redu, - = phys, 0 = term.)
  - branch_wgt(n_nodes,branch,ktask) - branch weight
  - node_below(n_nodes,branch,ktask) - node below node and branch

Building the Trees, Creating the nodes

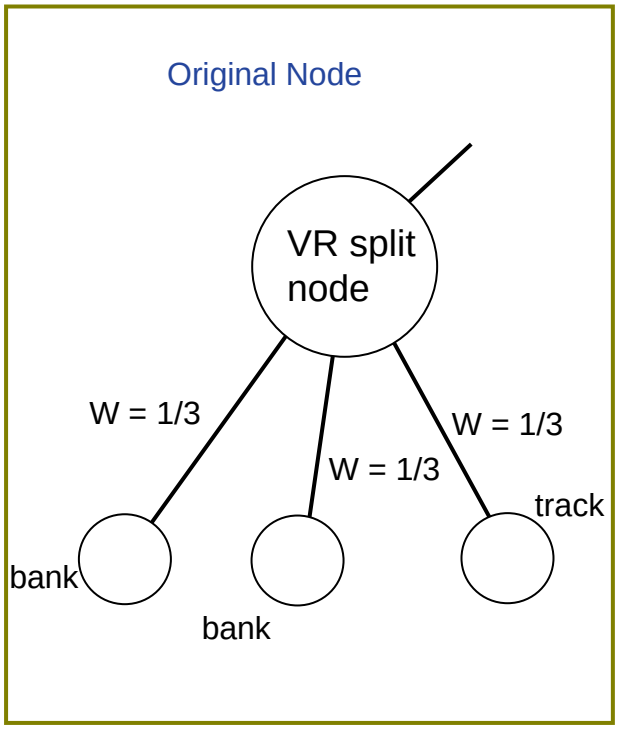
- Most physical and variance reduction nodes are created in the new subroutine phtvr_bankit_particle, called from bankit.
  - Exceptions:
    - Dxtran
    - Implicit Capture
    - Forced collisions
    - Pair production
    - Annihilation photons
- Call to subroutine phtvr_bankit_particle controlled by flag phtvr_bankit_flag

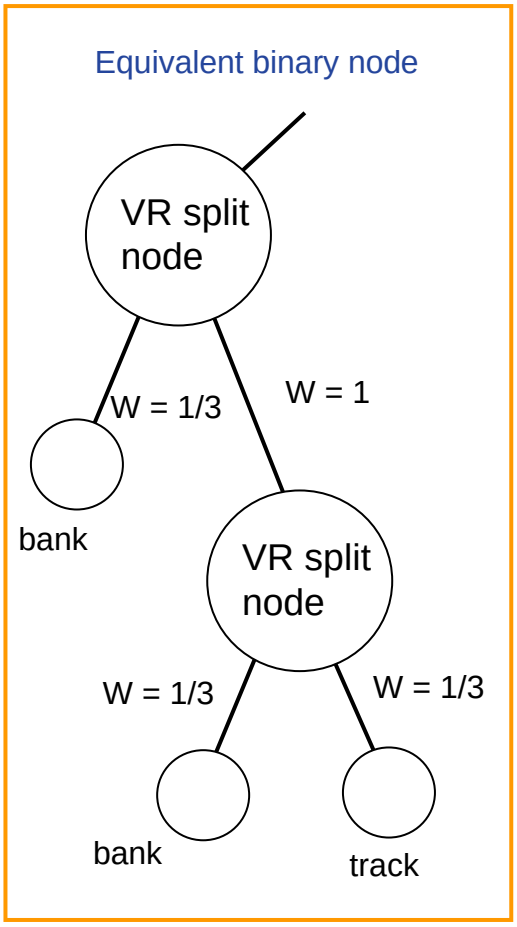
Creating Physical Nodes

- New single particle nodes are created in phtvr_bankit_particle
- Particles are not always banked in the correct order for the PHTVR trees
  - Particles created by weight window splitting and esplt/tsplt are banked before the physical particle.
  - The flag make_physical_node is used to signal if the physical node needs to be created before the variance reduction node
- Particle may be created but not banked
  - May be rouletted instead
  - Instead of creating a new node, zero the branch weight. This is equivalent to creating a new physical node and then rouletting the new particle

Building the Trees – Importance Splitting

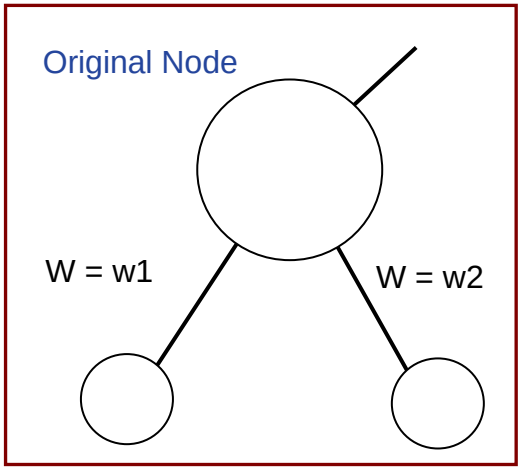
- Particles can split into several new tracks, each with the same weight
- All but one of the new tracks are banked
- Need to convert to a binary tree
  - Create additional VR nodes
  - Set branch weight between nodes to one
- Used for cell, time, and energy importances, and weight windows
- For noninteger splitting, the expected value is used as the branch weight.

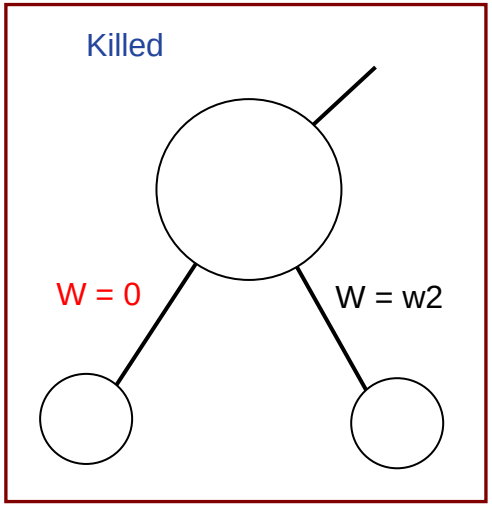


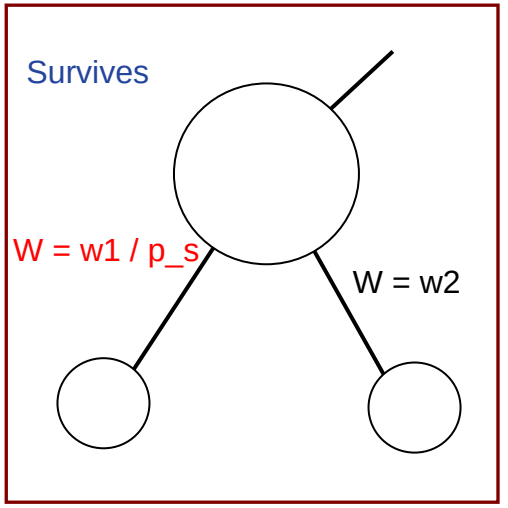


Building the Trees – Roulette

- In roulette games with survival probability p_s, particles are either killed with probability 1-p_s (rouletted), or survive with the weight increased by a factor of 1/p_s.
- Set the branch weight to 0 for rouletted particles, or increase by R
- Roulette can be turned off for pulse height tallies in MCNP using the VAR card



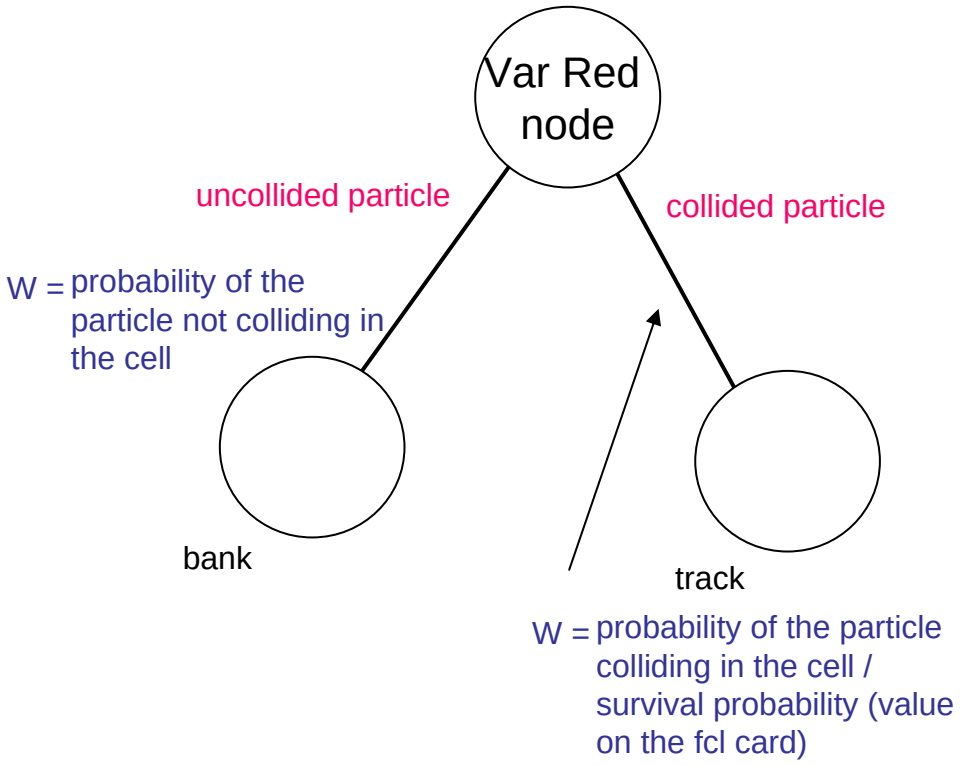




$R = imp_{in}/imp_{out}$

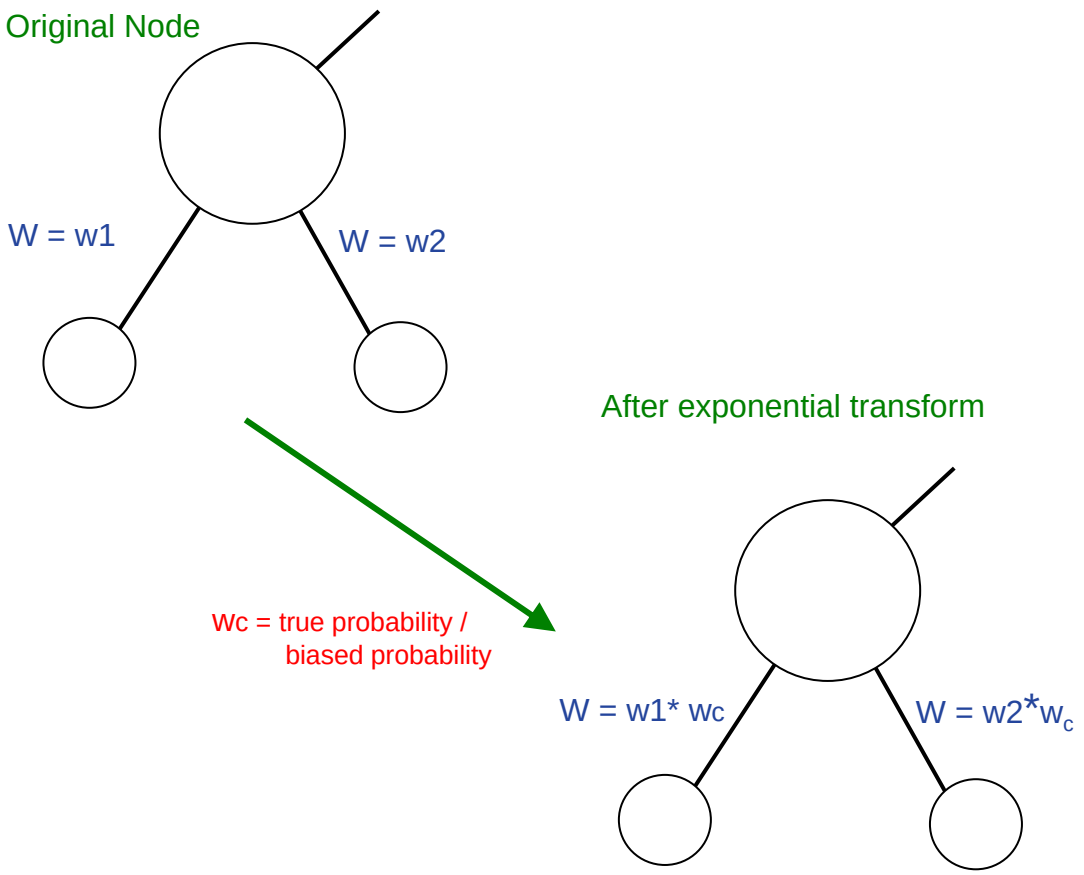
Building the Trees – Forced Collisions

- For forced collisions, the track is split into collided and uncollided parts.
- Create a Variance Reduction node with branch weights corresponding to the probability of a collision in the cell



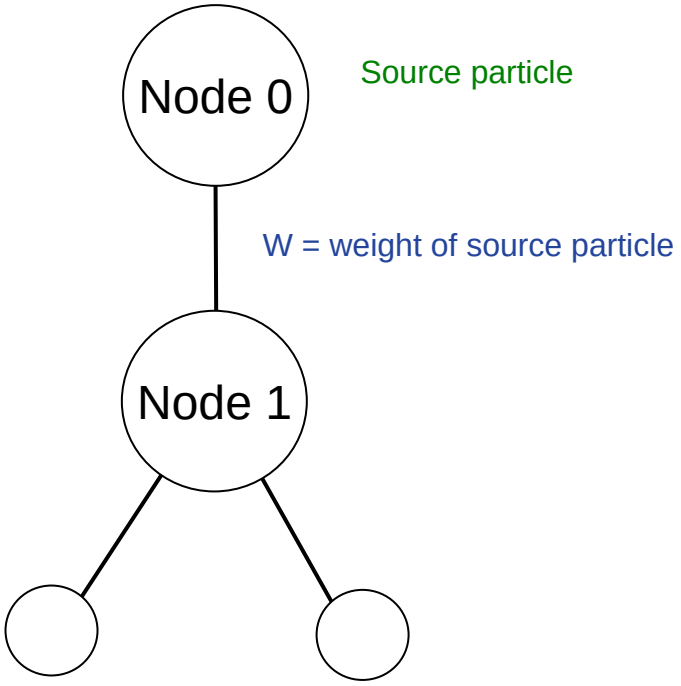
Building the Trees – Exponential Transform

- The exponential transform biases the distance to collision by sampling from a fictitious density and adjusts for this by multiplying by the ratio of the true probability to the biased probability
- For PHTVR trees, adjust the branch weight of the particle by w_c



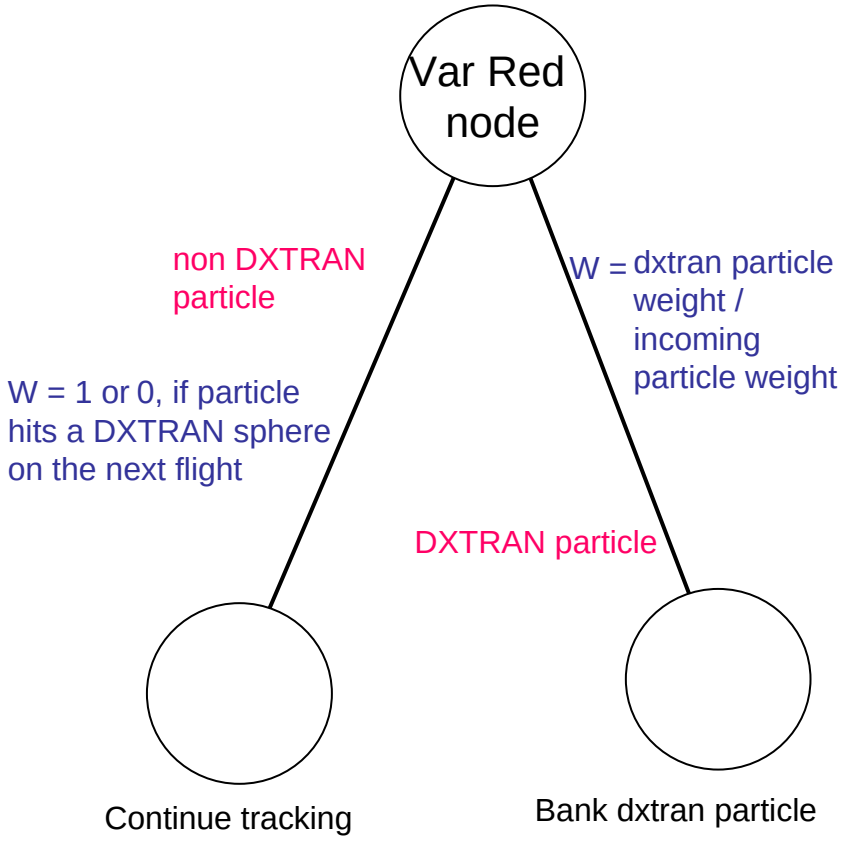
Building the Trees – Source Biasing

- Source biasing only changes the weight of the source particle
- For PHTVR trees, adjust the branch weight source particle (node zero)



DXTRAN

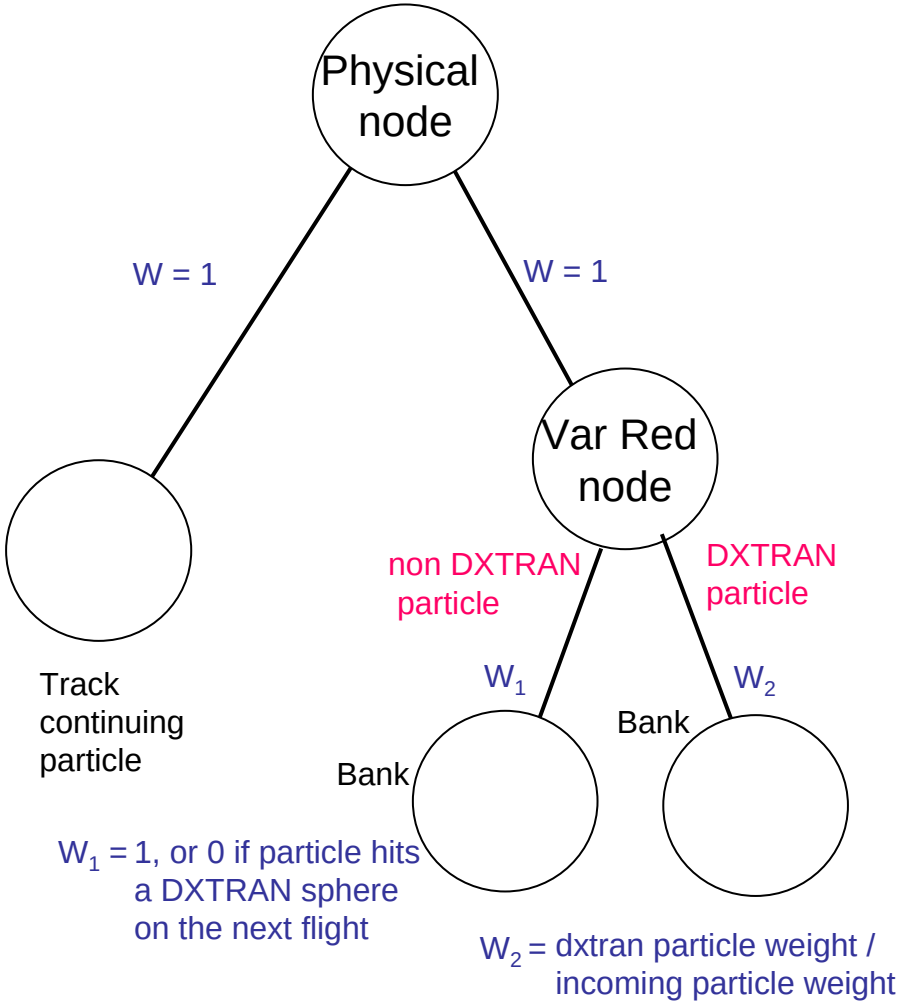
- DXTRAN splits the particle history into two parts:
  1) the part that enters the DXTRAN sphere on the next flight (the DXTRAN particle)
  2) the part that does not enters the DXTRAN sphere on the next flight
- Create Variance Reduction node with DXTRAN particle branch weight equal to the ratio of the new DXTRAN particle weight to the original particle weight
- If the non DXTRAN particle hits the dxtran sphere on the next flight, its branch weight is set to zero (killed)



DXTRAN: Single particle created

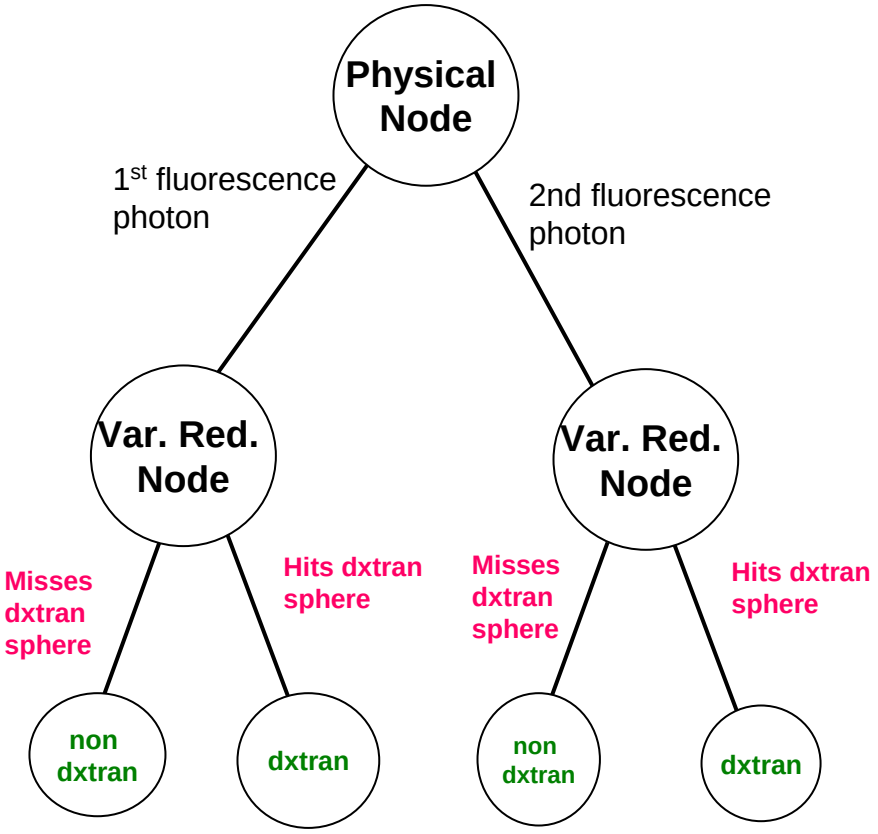
- Create physical node for the new particle, followed by a variance reduction node for the dxtran particle

Coding Note: Since MCNP creates and banks the dxtran particle before producing any of the physical particles, special coding is required to create the physical node before the variance reduction node.



DXTRAN: Double Fluorescence

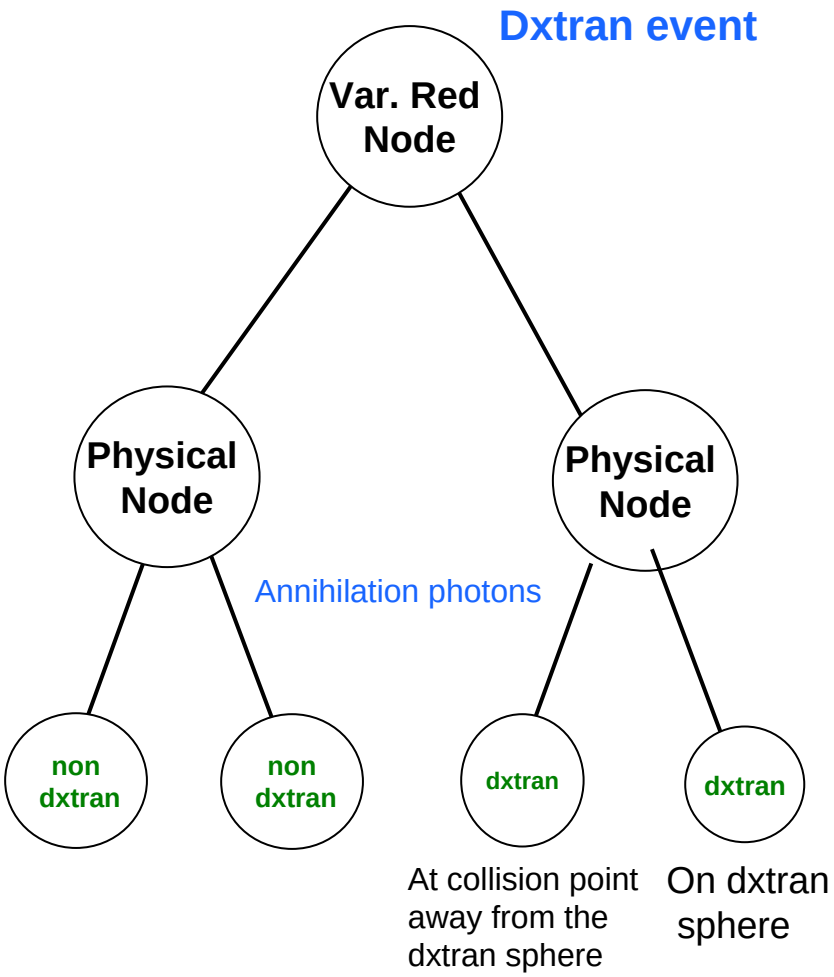
- Assumes no correlation between photons
- Create DXTRAN particle for both fluorescence photons
- 4 possible outcomes
  - Neither particle hits dxtran sphere
  - 1st photon hits the sphere and 2nd misses
  - 2nd photon hits the sphere and the 1st misses
  - Both photons hit the sphere
- The tree can be reduced to that shown on the right



See *Pulse Height Tally Variance Reduction in MCNP*, LA-13955 (2004) for details

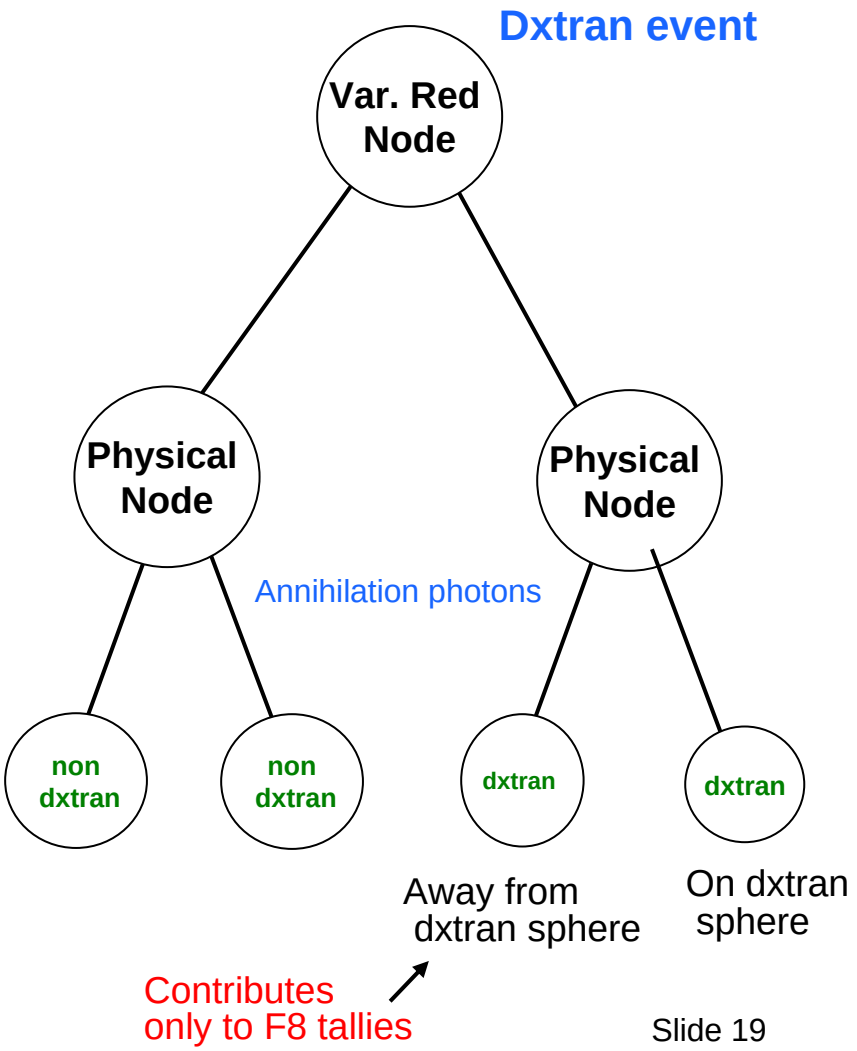
DXTRAN: Annihilation Photons

- There is a correlation between the annihilation photons
  - Both photons cannot hit the dxtran sphere
- For non-F8 tallies, only one annihilation photon is created.
  - The weight balance is preserved by killing the non-dxtran particle if it hits the dxtran sphere.
- F8 tallies require that both photons be tracked
  - To preserve the weight balance, if one non-dxtran photon hits the dxtran sphere, both non-dxtran photons must killed, and not contribute to any of the tallies



DXTRAN: Annihilation Photons (2)

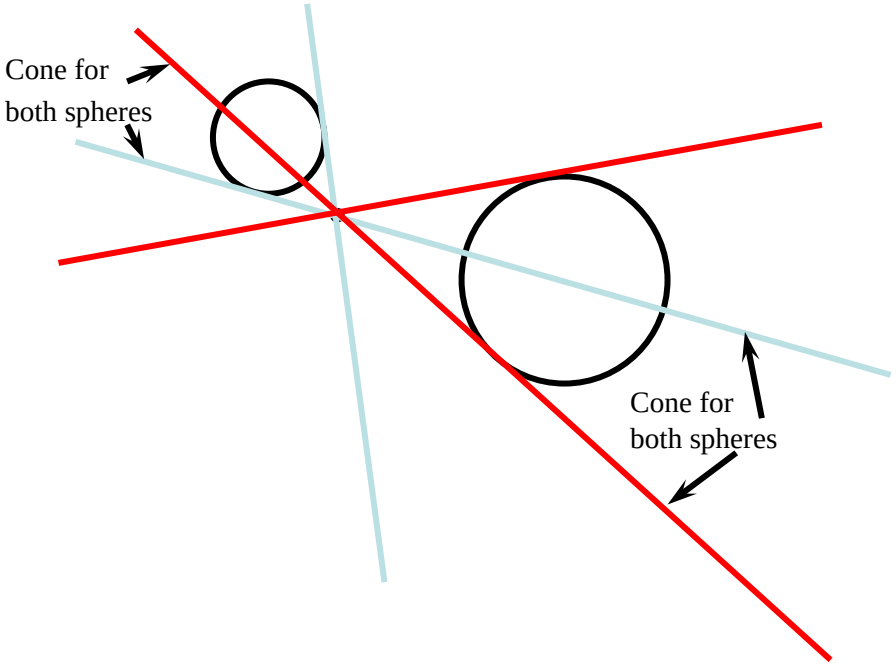
- Problems with creating both annihilation photons
  - The photon traveling towards the sphere could have split before it gets there, with some particles hitting the sphere and others missing it.
  - Other photons could have contributed to several tallies, which would need to be undone
- Solution
  - score nonF8 tallies only with the dxtran photon on the sphere (traditional method)
  - Create flag scoring_particle in PBLCOM to track scoring status



Annihilation Photons with Multiple DXTRAN Spheres

PHTVR and positron decay requires creating both 0.511 dxtran photons

- There are 4 possible outcomes:
  - neither particle hits the spheres (non-dxtran photons)
  - one particle only hits sphere 1
  - one particle only hits sphere 2
  - both particles hit a sphere

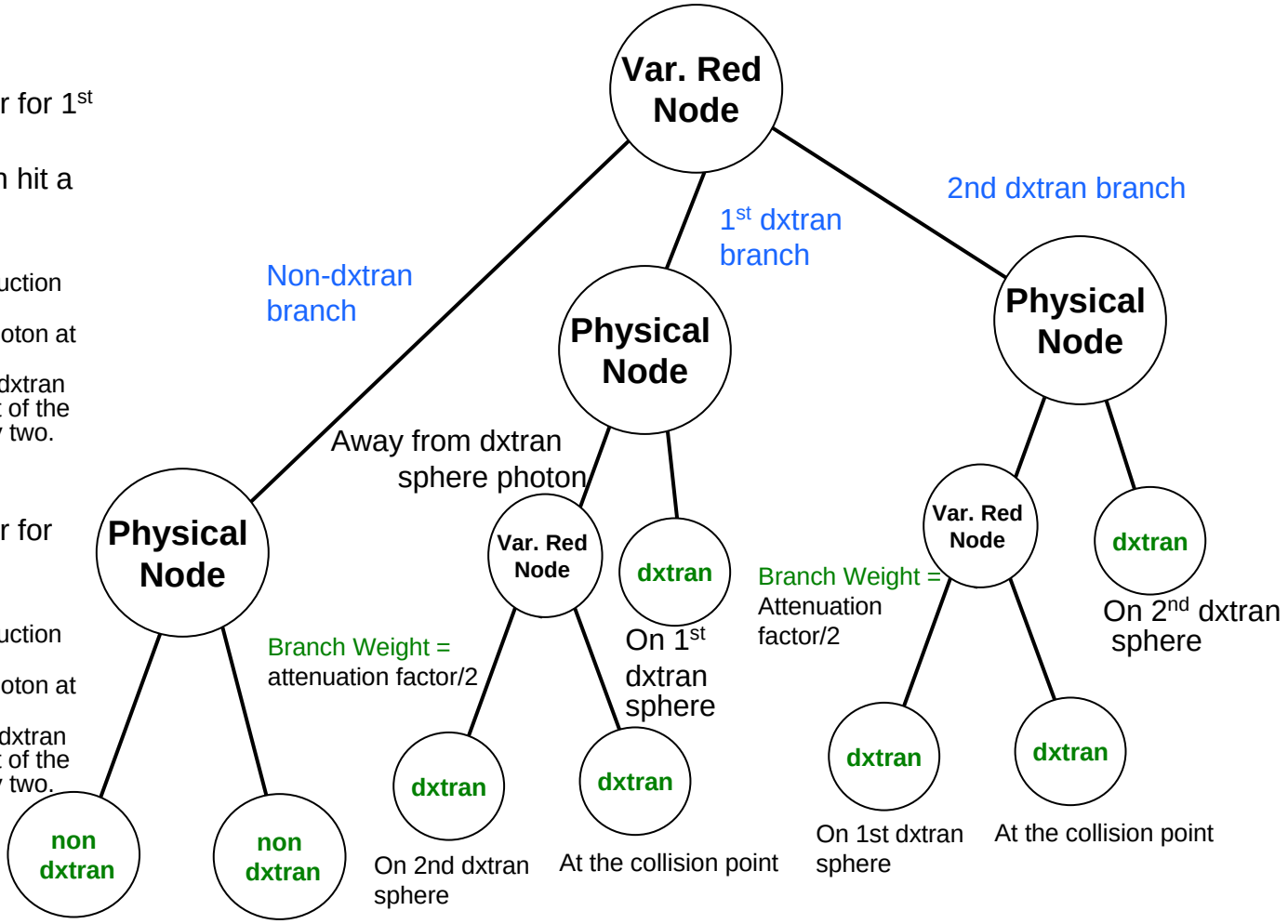


Annihilation Photons with Multiple DXTRAN Spheres (2)

Solution

- create nondxtran pair
- create dxtran annihilation pair for 1st dxtran sphere
- if the away dxtran photon can hit a 2nd dxtran sphere, then
  - create another variance reduction node
  - under this node start one photon at the collision point
  - 2nd photon is put on the 2nd dxtran sphere with a branch weight of the attenuation factor divided by two.
- create dxtran annihilation pair for 2nd dxtran sphere
  - create another variance reduction node
  - under this node start one photon at the collision point
  - 2nd photon is put on the 1st dxtran sphere with a branch weight of the attenuation factor divided by two.

The tree is shown on the right



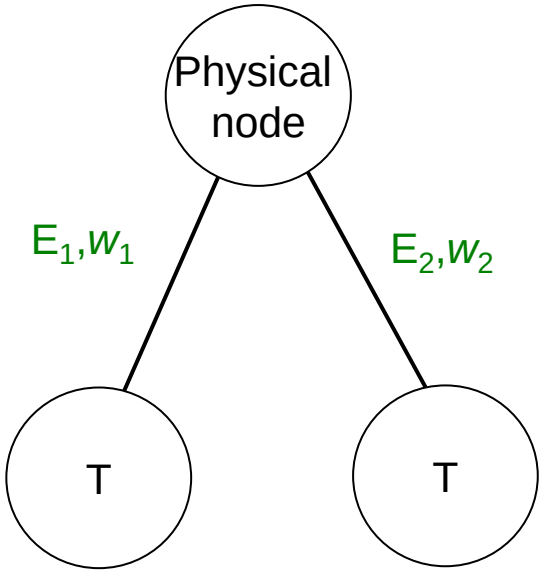
Energy Deposition Storage

- The same branch weights are used for all F8 tallies
- The energy deposited on each branch is unique for each F8 tally cell/user bin combination.
- Need to store the energy deposited on each branch for each F8 tally/cell/user bin.
- To store energy depostion, use a derived structure with array elements for the different tally bins: Eng_dep(F8_tally, cell, user_bin)
  - Structure array elements (all dynamic):
    - counter(ktask) - Number of node/branch combos which has eng dep
    - node_branch(counter, ktask) - Encoded values of node/branch combination
    - erg_deposited(counter,ktask) – Energy deposited under corresponding node/branch combination

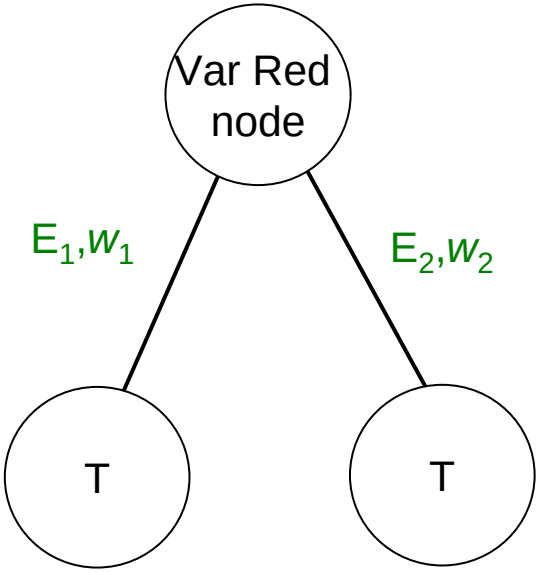
Debranching

- To score the tally at the end of the history, need to debranch the tree into individual subtracks (possible choices)
- The result is several energy_deposition/weight pairs (choices)
  - These choices are scored individually in the pulse height tally
- Variance Reduction nodes
  - Max number of new choices is sum of choices under node
- Physical node
  - Max number of new choices is product of choices under node
- Number of choices has an upper limit (currently 200)

Debranching process – start from terminal node



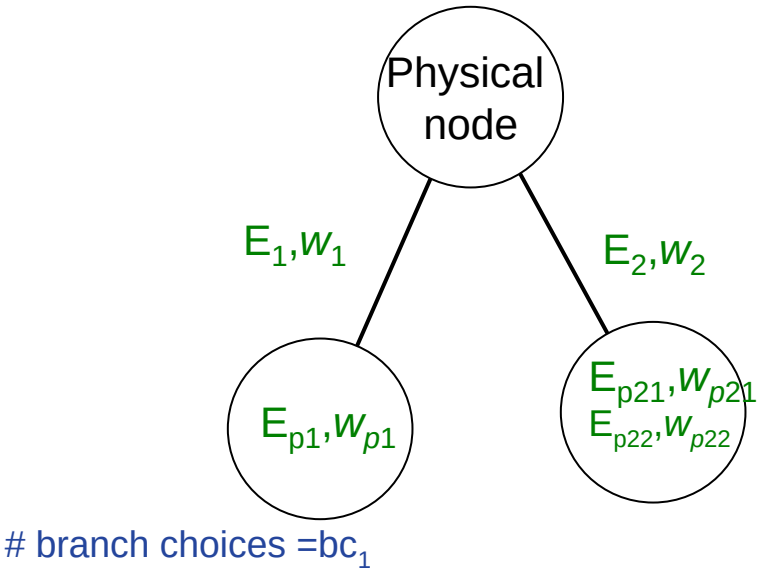
Choice 1: deposit $E_1 + E_2$ with weight $w_1 * w_2$

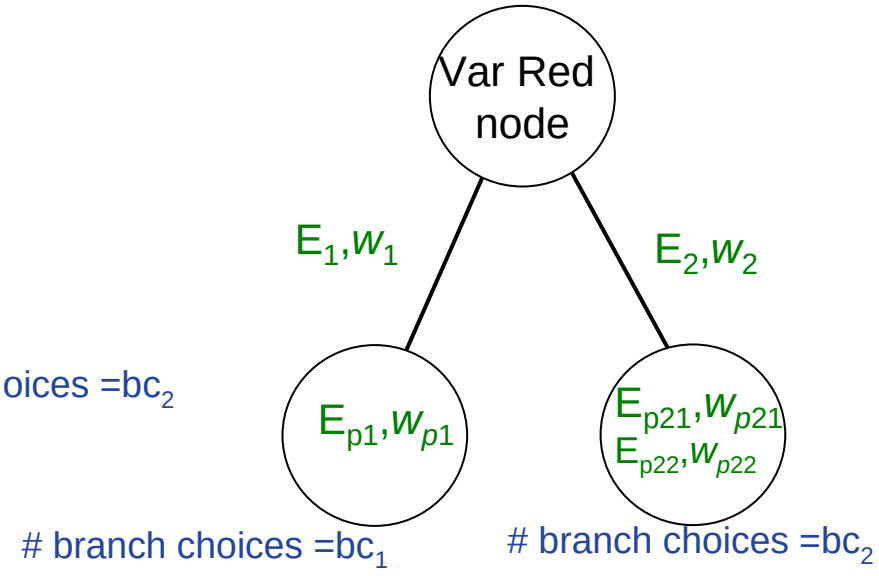


Choice 1: deposit E_1 with weight w_1

Choice 2: deposit E_2 with weight w_2

Debranching process – choices below nodes





Choice 1: deposit $E_1 + E_2 + E_{p1} + E_{p21}$ with weight $w_1 * w_2 * w_{p1} * w_{p21}$

Choice 2: deposit $E_1 + E_2 + E_{p1} + E_{p22}$ with weight $w_1 * w_2 * w_{p1} * w_{p22}$

Total # choices = bc1 * bc2

Choice 1: deposit $E_1 + E_{p1}$ with weight $w_1 * w_{p1}$

Choice 2: deposit $E_2 + E_{p21}$ with weight $w_2 * w_{p21}$

Choice 3: deposit $E_2 + E_{p22}$ with weight $w_2 * w_{p22}$

Total # choices = bc1 + bc2

Debranching variables

- Module variables
  - Energy, weight pairs are stored in two indexed arrays:

- de_br_eng_dep(:,:) ! Debranched energy deposition for each node
- de_br_wgt(:,:) ! Debranched weights for each node

- de_br_start(:,:) ! Starting index for each node in the de_br_eng_dep and de_br_wgt arrays
- de_br_end(:,:) ! Ending index for each node in the de_br_eng_dep and de_br_wgt arrays
- n_choices(:,:) ! Number of choices for each node

Limiting the Number of Choices

- Combine all zero and low energy depositions (particles that just pass through the cell) into two choices
- Cull choices
  - Different culling methods are used depending on the type of node
  - Physical nodes
    - Reduce 40,000 possible choices to 200
    - use a roulette “combing method”
  - Variance reduction nodes
    - Reduce 400 possible choices to 200
    - Remove the lowest weight choices so that 200 choices are left after culling.
    - Keep one of the lowest choices and adjust weight to compensate removing the others

Dynamic Arrays and OpenMP

- Several arrays need to be increased during particle transport
- OpenMP Retrictions
  - Only the master task can increase array sizes
  - No task can be accessing the array when size is increased.
- Need to coordinate between tasks during transport and pause the subtasks when memory needs to be increased

OpenMP coding

- Set up arrays and flags to signal subtasks
- Use !OMP FLUSH (variable) directive
  - Immediately updates/reads current value of variable
- All arrays are increased in one subroutine
- Array sizes can be increased only if subtasks are
  - Waiting for memory to be increase
  - Is not in the transport or debranching sections
  - Subtask has exited trnspt for tally updates, write to runtpe, etc

When an array size needs to be increased

- Master task
  - Calls increase_phtvr_arrays
    - Waits until other subtasks have paused or are not in the transport loop
    - Sets lock by calling sm_lon
    - Increases array sizes
    - Removes lock by calling sm_loff
- Subtasks
  - Signal other tasks that it needs to increase memory
  - Pause until master task signals that array size has been adjusted
- All tasks check whether memory needs to be increased
  - Before particle history begins (call to RN_init_particle) in trnspt
  - Before exiting trnspt
    - The master task is not allowed to exit trnspt until all the subtasks have finished

Increasing the dynamic arrays

- 3 variables store the current array sizes
  1. Max_n_nodes – tree building and some debranching arrays
  2. leng_de_br_arrays – debraching arrays
  3. eng_dep(i_tal,i_cell,i_usr_bin)%max_counter – energy deposition arrays for talph.
- Array increase_memory is used to signal which arrays need to be increased.
  - If sum(increase_memory) > 0, signals tasks that memory needs to be increased
- Subroutine increase_phtvr_arrays increases the arrays

Coordination of subtasks (OpenMP)

- Coordination variables
  - threads_stopped(ntasks-1) Flag for which threads are not transporting particles either they are paused or not in particle trans. loop
  - threads_running(ntask-1) Flag for which threads are currently running; set in trnspt.
  - master_thread_locked Flag for when the master thread has placed a lock on the other threads; subtasks exit from do loop and call sm_lon
- When sum(threads_stopped) = sum(threads_running) the master task can increase the arrays sizes
- increase_memory is checked
  1. Before call to startp
  2. Before exiting trnspt
- Also, the master task will call increase_phtvr_arrays if it needs more memory

Restrictions and Defaults

- No phys:e card variance reduction techniques
  - Enum, rnok, bnum, xnum,
    - Set to 1 if not equal to 1 or 0
- Russian Roulette
  - On by default
  - Can be turned off with the VAR card
- Weight cutoffs
  - Default to zeros unless
    - explicit values included on cut:p or cut:e card.
    - forced collisions are used (use default values if no values are provided)