Using Serpent with the nodal neutronics program Ants

V. Valtavirta, A. Rintala, U. Lauranto and others

Contents

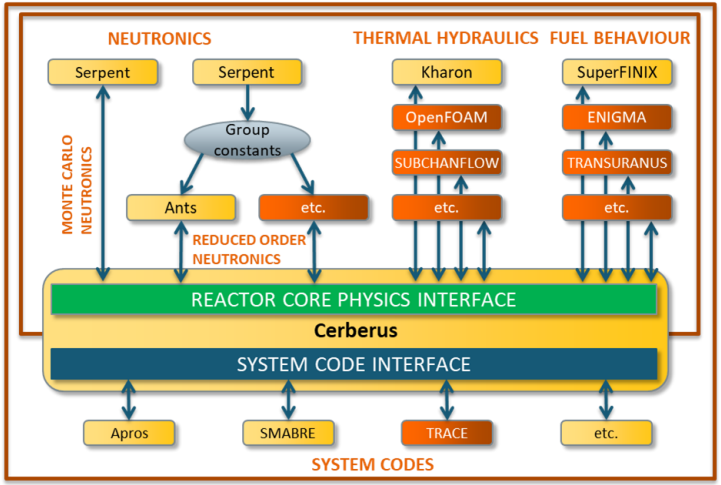
- Context
  - The Kraken framework
  - The Serpent-Ants calculation chain
- Group constant generation with Serpent for Ants.
- Current work on VVER benchmarks.
- Summary and next steps

Context:

The Kraken framework
The Serpent-Ants calculation chain

The Kraken framework

- VTT's new computational reactor analysis framework[1].
- Built for:
  - Independent deterministic safety analyses.
  - Evaluation of new reactor concepts.
  - Generation of input data for system codes, e.g. Apros.
- Intended to replace the current tools (e.g. HEXBU-3D and HEXTRAN based calculation chains) in some years.
- Neutronics solution is based on **either**
  - Direct Serpent continuous energy Monte Carlo solution **or**
  - Serpent-Ants two step calculation chain.
    - Serpent can provide both the homogenized group constants and the best possible reference solution even for 3D full core.



A schematic representation of the plans for the completed Kraken framework. Finnish solver modules developed at VTT are shown in yellow, while potential state-of-the-art third party solvers to be coupled are shown in orange.

[1] V. Valtavirta *et al*. "Kraken – an Upcoming Finnish Reactor Analysis Framework". *Proc. ANS MC2019*. Portland, OR, USA, Aug. 2019.

The Serpent-Ants calculation chain

Serpent[2]

- Continuous energy Monte Carlo multi-purpose particle transport code.
- Initially designed for group constant generation.
- Flexible geometry, neutron and photon transport.
- Steady state, burnup and transient.
- Developed at VTT since 2004

Ants[3,4]

- Multi-group nodal neutronics code.
- Currently uses nodal diffusion.
- Combines AFEN and FENM approaches for flux solution.
- Rectangular, hexagonal and triangular nodal models.
- Steady state, burnup and transient.
- Developed at VTT since 2017

- Serpent is the one and only tool for group constant generation in the Kraken framework.
- The aim is to leverage the advanced capabilities of Serpent in the two step calculation chain.

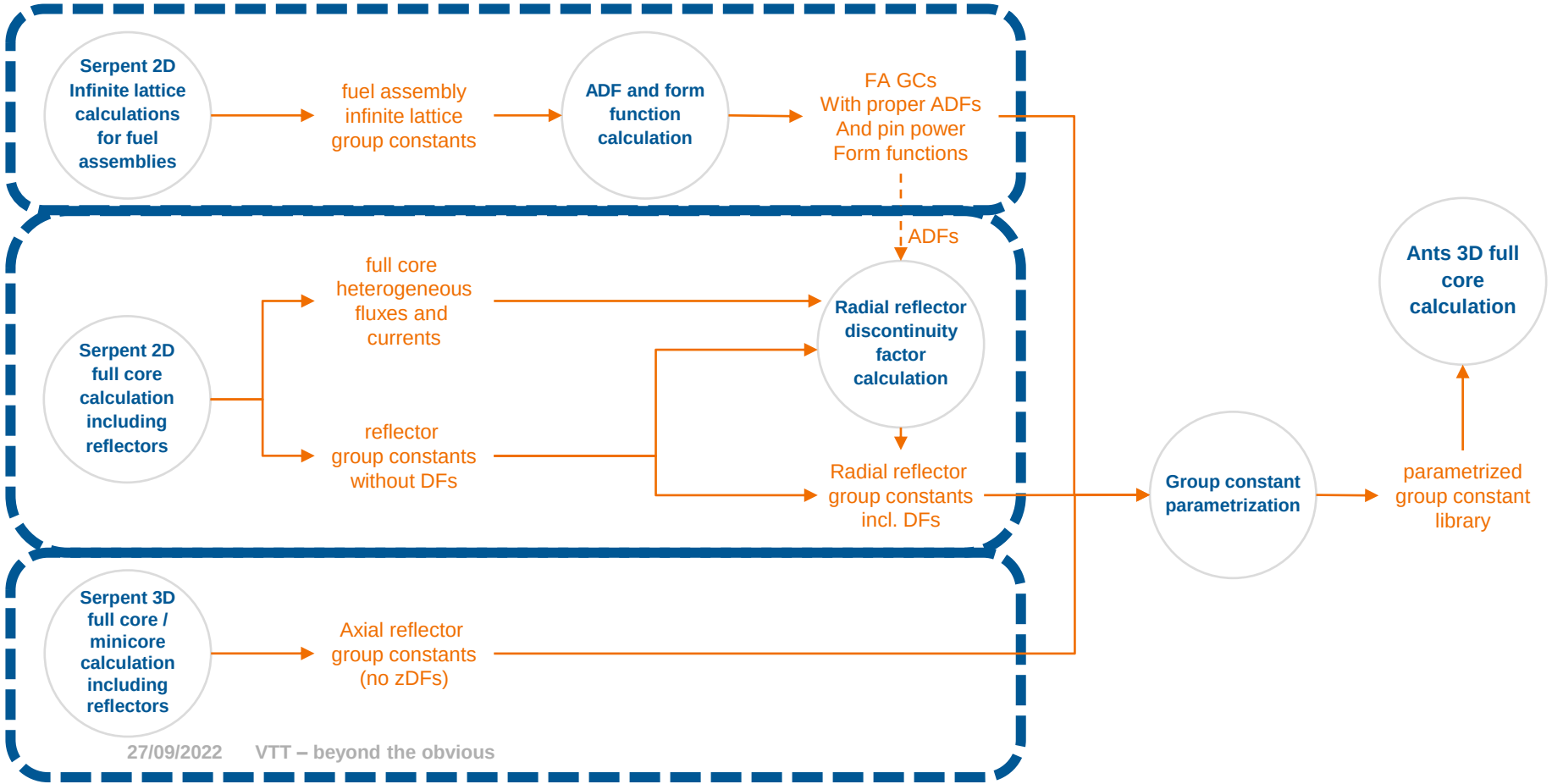
[2] J. Leppänen *et al.* “The Serpent Monte Carlo code: Status, development and applications in 2013”. Annals of Nuclear Energy 82 (2015), pp. 142–150.

[3] V. Sahlberg and A. Rintala. “Development and first results of a new rectangular nodal diffusion solver of Ants”. Proc. PHYSOR 2018. Cancun, Mexico, Apr. 2018

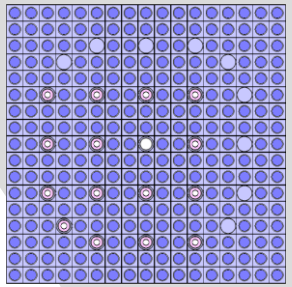
[4] A. Rintala and V. Sahlberg. “Extension of nodal diffusion solver of Ants to hexagonal geometry”. Kerntechnik 84 (2019), pp. 252–261.

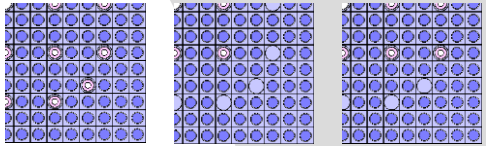
Using Serpent to generate group constants for Ants in the Kraken framework

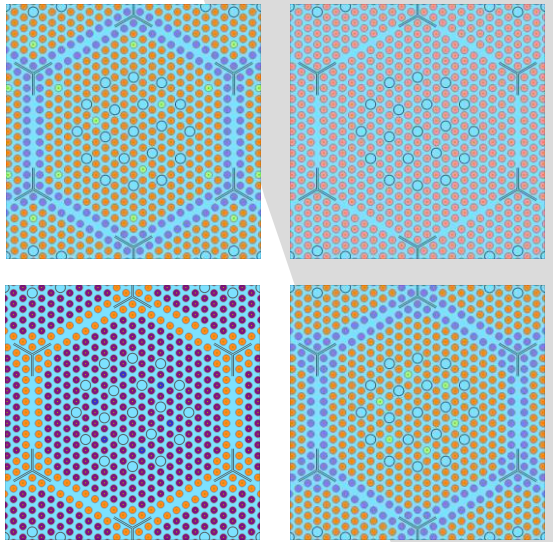
Group constant generation



Best practices calculation chain fuel GCs







- Full assembly in infinite lattice.
- Depletion calculations with nominal and off-nominal conditions.
- Branch calculations with momentary variations:
  - Different (T_{fuel} , T_{cool} , ρ_{cool} , C_B) variations.
  - Control rod variations.
  - Spacer grid variations.
  - Instrument tube variations.
- Can use an intermediate multigroup structure and apply leakage correction / critical spectrum in condensation to a few group structure.
- Typically produce CMM[7] or transport corrected diffusion coefficients.

[7] Z. Liu et al. “Cumulative migration method for computing rigorous diffusion coefficients and transport cross sections from Monte Carlo”. *Annals of Nuclear Energy*, 112 (2018), pp. 507–516.

Practical things about fuel GCs

- Full assembly in infinite lattice (set bc) (input example)
  - ADF setup
  - Pin power setup
  - Poison constants, microdepletion setup.
- Depletion calculations with nominal and off-nominal conditions.
- Branch calculations with momentary variations:
  - Different (T_{fuel} , T_{cool} , ρ_{cool} , C_B) variations.
  - Control rod variations.
  - Spacer grid variations.
  - Instrument tube variations.
- Can use an intermediate multigroup structure and apply leakage correction / critical spectrum in condensation to a few group structure.
- Typically produce CMM[7] or transport corrected diffusion coefficients.

```
set cmm 1
set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010
```

Use of:
branch-card
casematrix-card

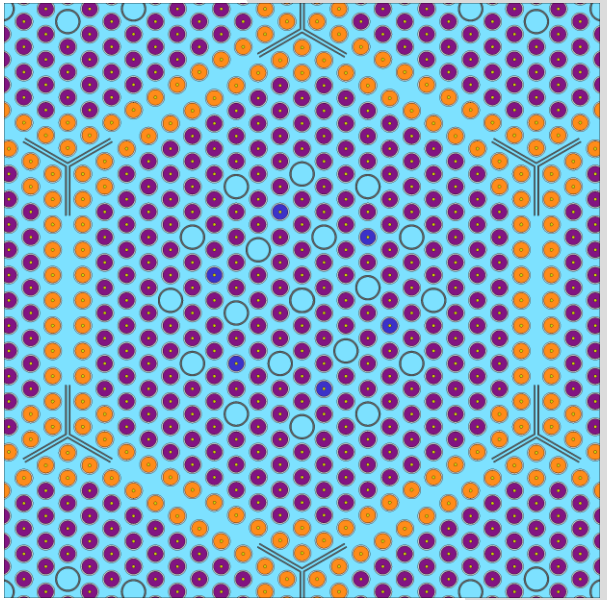
Running Serpent from command line

his, coe, ln -s

```
set fum cas70_ext 2 f 3

set micro cas70_ext

set nfg cas2_ext

set repro 0

set shbuf 0 0
```

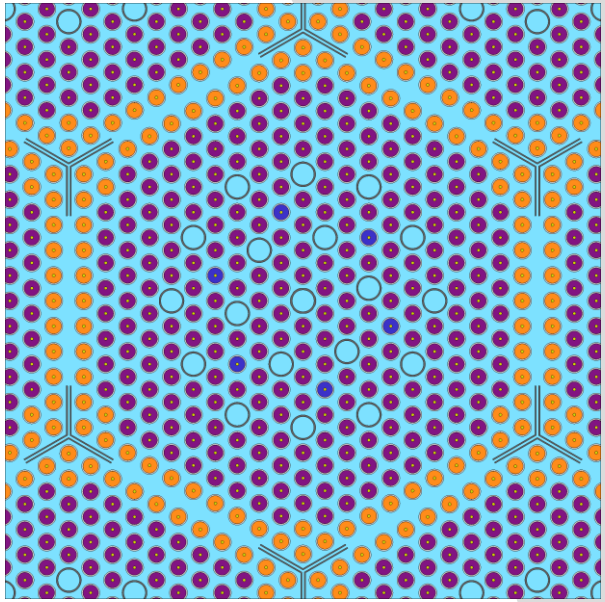
Setting up ADF and pin power evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Input file for group constant generation for fuel assembly 390GO ---%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

include "../includes/materials.inc"
include "../includes/pins.inc"
include "../includes/lattices.inc"
include "../includes/C_run_options_GC.inc"
include "../includes/390GO.mvol"
include "../includes/C_branches.inc"
include "../includes/C_histories.inc"
include "../includes/C_watercomp.inc"

plot 3 2000 2000 0 -15 15 -15 15

% --- Fuel assembly boundary surface (lattice pitch 23.6 cm, Bilodid 2020, Tbl 2.1)

surf hSurf hexxc 0 0 11.8
cell c1 0 fill u390GOs -hSurf
cell c2 0 outside      hSurf

% Sets parameters for the calculation of assembly discontinuity factors.

%set adf 0 hSurf 1 %  30 degree reflective
set adf 0 hSurf 8 %  60 degree periodic
%set adf 0 hSurf 9 % 120 degree periodic

% Turns on the calculation of pin powers and pin power form factors

set ppw 0 l390GOs

% Periodic boundary conditions needed for hexagonal assemblies

set bc 3
```



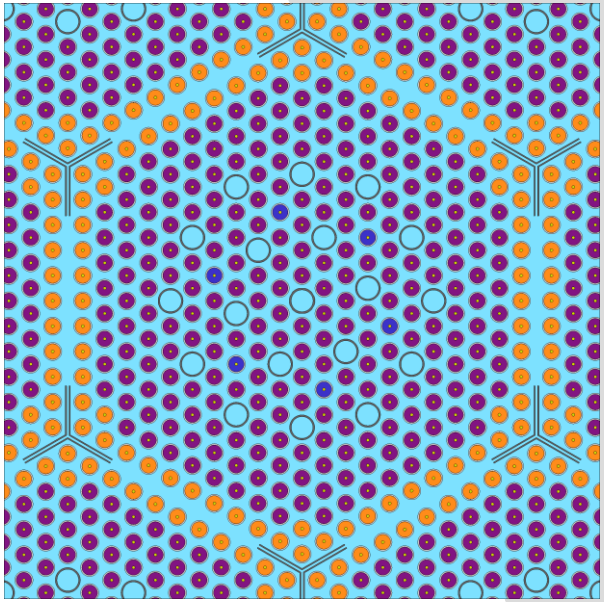
Poison constants and microdepletion data

```
% --- Fission poison and microdepletion data generation

% set poi OPT VOL [ XE135M ]

set poi 1 482.3415

% set mdep UNI VOL N MAT1 MAT2 ... MATN
%          ZAI1 MT1
%          ZAI2 MT2
%          ...

set mdep 0 482.3415 0
         922380 16
         922380 18
         922380 102
         932390 16
         932390 18
         932390 102
         942390 16
         942390 18
         942390 102
```



Using `branch` and `casematrix` to set up history and branch calculations

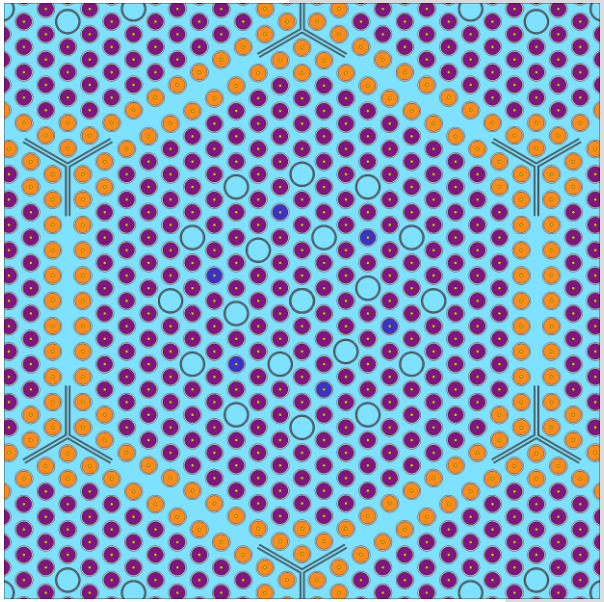
```
branch case0
stp fuel13 -10.2605 1005.0
stp fuel22 -10.2605 1005.0
stp fuel30 -10.2605 1005.0
stp fuel36 -10.2605 1005.0
stp fuel40 -10.2605 1005.0
stp fuel44 -10.2605 1005.0
stp fuel24Gd -10.2279 1005.0
stp fuel33Gd -10.2279 1005.0
stp fuel36Gd -10.2279 1005.0
stp E110 -6.54516 1005.0
stp E635 -6.55 1005.0
stp steel -7.9 1005.0
stp DyTi -5.1 1005.0
stp B4C -1.8 1005.0
stp helium -0.0015981 1005.0
repm cool cool_1207B_0554T_0762D
var BOR 1207
var TFU 1005.0
var TMO 554.0
var DMO 0.7621
```



KrakenTools/tests/*GC_generator*

Using `branch` and `casematrix` to set up history and branch calculations

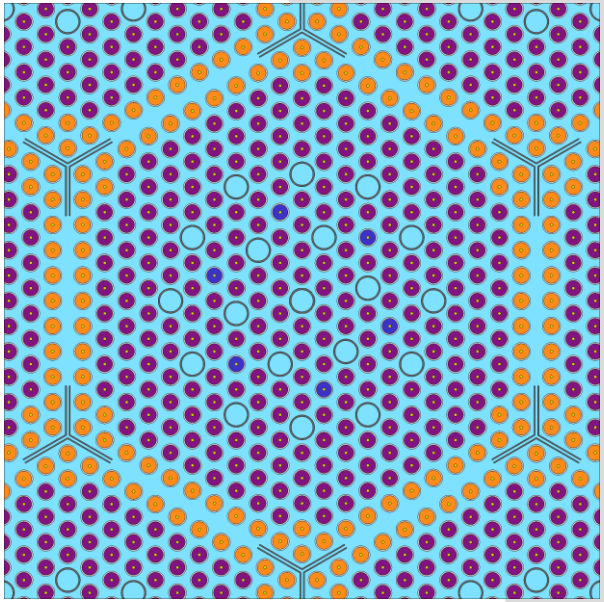
```
branch no_cr
repu pGT pGTEmpty
var CR 0

branch dyti
repu pGT pGTDT
var CR 1

branch boc
repu pGT pGTBC
var CR 2

branch no_spa
repu uAxiWater uWater
var SPA 0

branch spa
repu uAxiWater uGrid
var SPA 1
```



KrakenTools/tests/*GC_generator*

Using `branch` and `casematrix` to set up h ons

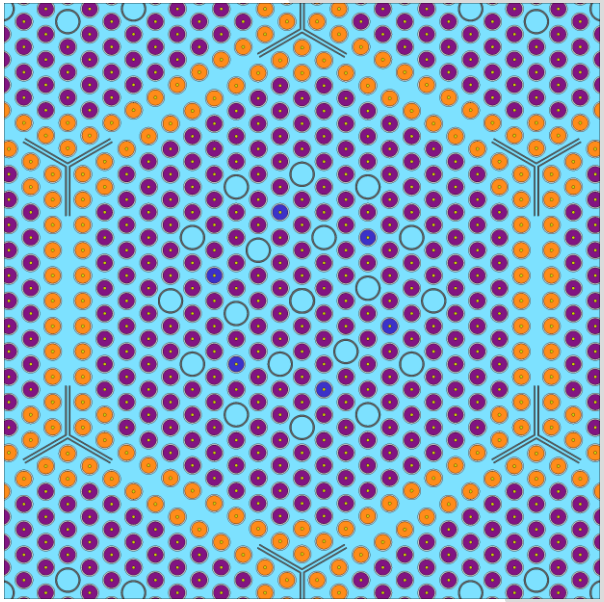
```
branch nom_his
stp fuel13 -10.2605 1005.0
stp fuel22 -10.2605 1005.0
stp fuel30 -10.2605 1005.0
stp fuel36 -10.2605 1005.0
stp fuel40 -10.2605 1005.0
stp fuel44 -10.2605 1005.0
stp fuel24Gd -10.2279 1005.0
stp fuel33Gd -10.2279 1005.0
stp fuel36Gd -10.2279 1005.0
stp E110 -6.54516 578.0
stp E635 -6.55 578.0
stp steel -7.9 578.0
stp DyTi -5.1 578.0
stp B4C -1.8 578.0
stp helium -0.0015981 578.0
repm cool cool_0525B_0578T_0716D
repu pGT pGTEmpty
repu uAxiWater uWater
var hTFU 1005.0
var hBOR 525.0
var hTMO 578.0
var hDMO 0.7167
var hCR 0
var hSPA 0
```



KrakenTools/tests/*GC_generator*

Using `branch` and `casematrix` to set up history and branch calculations

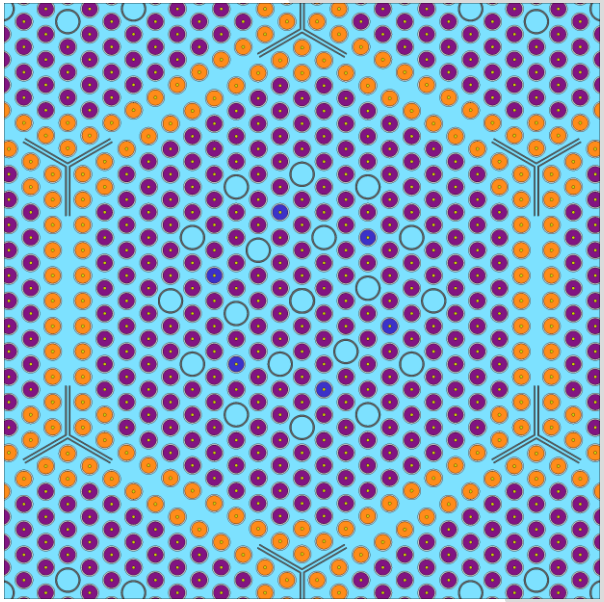
```
% Nominal state point for all spacer/cr combinations
% at many burnup points

casematrix nominals
2 nom_his off_nom_his
25 0 0.1 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12.5 15
17.5 20 22.5 25 27.5 30 34.0
1 case0
2 no_spa spa
3 no_cr dyti boc

% Off nominal state points for polynomial fitting
% at a coarser burnup grid

casematrix coefficients
2 nom_his off_nom_his
11 0 1 3 5 7 10 15 20 25 30 34
13 case1 case2 case3 case4 case5 case6 case7 case8
case9 case10 case11 case12 case13
2 no_spa spa
3 no_cr dyti boc
```



KrakenTools/tests/*GC_generator*

Using `branch` and `casematrix` to set up history and branch calculations

```
# First run histories (burnup calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix nominals 1 -1 390GO
# ^Produces 390GO_nominals_h1.wrk binary restart
# (nominal history)

sss2 -omp 20 -casematrix nominals 2 -1 390GO
# ^Produces 390GO_nominals_h2.wrk binary restart
# (off-nominal history)

# We can use the same restarts for coefficient calculations

ln -s 390GO_nominals_h1.wrk 390GO_coefficients_h1.wrk
ln -s 390GO_nominals_h2.wrk 390GO_coefficients_h2.wrk
```



Using `branch` and `casematrix` to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 –coe –casematrix nominals 1 0 390GO
# Runs all branches for nominal history based on
# 390GO_nominals_h1.wrk binary restart
# Has 1x2x3=6 branches
# (x25 burnups = 150 transport solutions)

sss2 -omp 20 –coe –casematrix nominals 2 0 390GO

sss2 -omp 20 –coe –casematrix coefficients 1 0 390GO
# Has 13x2x3=78 branches
# (x11 burnups = 858 transport solutions)

sss2 -omp 20 –coe –casematrix coefficients 2 0 390GO

# These 4 calculations could be distributed across 4
# calculation nodes on a cluster
```

```
% Nominal state point for all spacer/cr combinations
% at many burnup points

casematrix nominals
2 nom_his off_nom_his
25 0 0.1 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12.5 15 17.5 20
22.5 25 27.5 30 34.0
1 case0
2 no_spa spa
3 no_cr dyti boc

% Off nominal state points for polynomial fitting
% at a coarser burnup grid

casematrix coefficients
2 nom_his off_nom_his
11 0 1 3 5 7 10 15 20 25 30 34
13 case1 case2 case3 case4 case5 case6 case7 case8 case9 case10
case11 case12 case13
2 no_spa spa
3 no_cr dyti boc
```

Using branch and casematrix to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix nominals 1 1 390GO
sss2 -omp 20 -casematrix nominals 1 2 390GO
sss2 -omp 20 -casematrix nominals 1 3 390GO
sss2 -omp 20 -casematrix nominals 1 4 390GO
sss2 -omp 20 -casematrix nominals 1 5 390GO
sss2 -omp 20 -casematrix nominals 1 6 390GO

# Runs single branches for nominal history based on
# 390GO_nominals_h1.wrk binary restart
# Has 1x2x3=6 branches
# (x25 burnups = 150 transport solutions)
```

```
% Nominal state point for all spacer/cr combinations
% at many burnup points

casematrix nominals
2 nom_his off_nom_his
25 0 0.1 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12.5 15 17.5 20
22.5 25 27.5 30 34.0
1 case0
2 no_spa spa
3 no_cr dyti boc

% Off nominal state points for polynomial fitting
% at a coarser burnup grid

casematrix coefficients
2 nom_his off_nom_his
11 0 1 3 5 7 10 15 20 25 30 34
13 case1 case2 case3 case4 case5 case6 case7 case8 case9 case10
case11 case12 case13
2 no_spa spa
3 no_cr dyti boc
```

Using `branch` and `casematrix` to set up history and branch calculations

```
# Then run branches (coefficient calculations)
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>

sss2 -omp 20 -casematrix coefficients 2 1 390GO
sss2 -omp 20 -casematrix coefficients 2 2 390GO
sss2 -omp 20 -casematrix coefficients 2 3 390GO
...
sss2 -omp 20 -casematrix coefficients 2 77 390GO
sss2 -omp 20 -casematrix coefficients 2 78 390GO

# Runs single branches for off-nominal history based on
# 390GO_coefficients_h2.wrk binary restart
# Has 13x2x3=78 branches
# (x11 burnups = 858 transport solutions)

# Running branches separately yields 6*2+78*2 = 168
# separate Serpent runs which can be distributed across
# a computational cluster
```

```
% Nominal state point for all spacer/cr combinations
% at many burnup points

casematrix nominals
2 nom_his off_nom_his
25 0 0.1 0.3 0.6 1 1.5 2 3 4 5 6 7 8 9 10 11 12.5 15 17.5 20
22.5 25 27.5 30 34.0
1 case0
2 no_spa spa
3 no_cr dyti boc

% Off nominal state points for polynomial fitting
% at a coarser burnup grid

casematrix coefficients
2 nom_his off_nom_his
11 0 1 3 5 7 10 15 20 25 30 34
13 case1 case2 case3 case4 case5 case6 case7 case8 case9 case10
case11 case12 case13
2 no_spa spa
3 no_cr dyti boc
```

Output data from fuel GC calculations

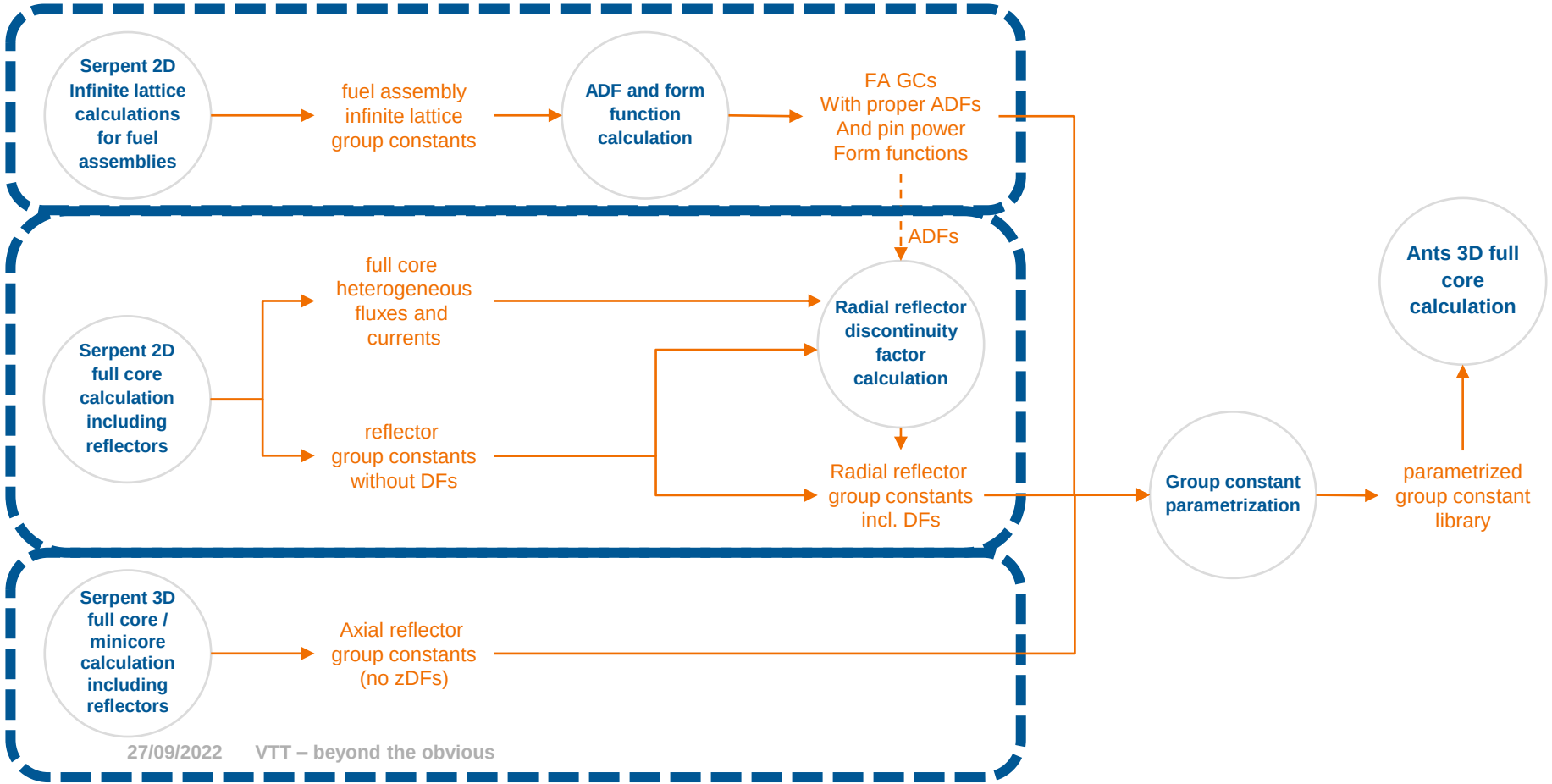
```
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>
```

- `390GO_<case_name>_h<his_idx>_r<coe_idx>.coe` files
  - Contain homogenized few group constants for homogenized universes.
    - Includes cross sections, discontinuity factor data, pin power form function data, poison constants, basic time constants, microdepletion data etc.
    - var definitions from branch cards show up in .coe files to help identify, which file contains which data.
- `390GO_<case_name>_h<his_idx>_r<coe_idx>_res.m` files
  - Contain some other important data not directly bound to homogenized universes.
- `390GO_<case_name>_h<his_idx>_r<coe_idx>_mdxb<coe_idx>.m` files
  - Contain important data for microdepletion:
    - Fission spectra.
    - Decay reactions (decay constants, targets, branching ratios).
    - Neutron induced reactions (MTs, reaction products, Q-values).

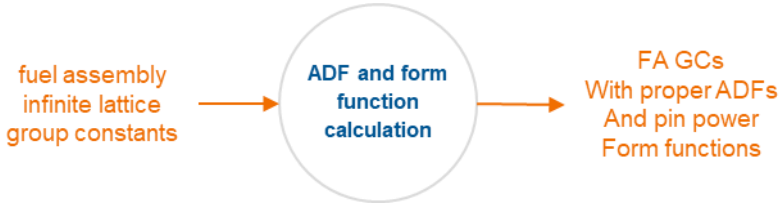
```
branch case0
stp fuel13 -10.2605 1005.0
stp fuel22 -10.2605 1005.0
stp fuel30 -10.2605 1005.0
stp fuel36 -10.2605 1005.0
stp fuel40 -10.2605 1005.0
stp fuel44 -10.2605 1005.0
stp fuel24Gd -10.2279 1005.0
stp fuel33Gd -10.2279 1005.0
stp fuel36Gd -10.2279 1005.0
stp E110 -6.54516 1005.0
stp E635 -6.55 1005.0
stp steel -7.9 1005.0
stp DyTi -5.1 1005.0
stp B4C -1.8 1005.0
stp helium -0.0015981 1005.0
repm cool cool_1207B_0554T_0762D
var BOR 1207
var TFU 1005.0
var TMO 1005.0
var DMO 0.7621
```

Can be read into Python objects with `serpentTools` and `KrakenTools`

Group constant generation



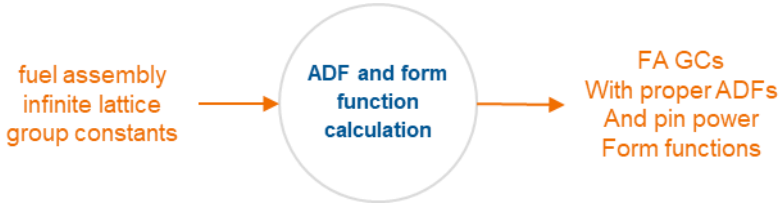
Fuel ADFs and pin power form functions



- Assembly discontinuity factors and pin power form functions (FFs) are by definition dependent on the homogeneous flux solution.
  - In some simple cases, the homogeneous flux is constant inside the assembly and equal to the mean heterogeneous flux.
  - In general, an actual solution to the homogeneous problem is required.
  - Serpent has an internal diffusion flux solver, but as the homogeneous solution is dependent on the nodal model, using the Serpent calculated ADFs and form functions is **wrong** in general.

- Instead, Ants single node 2D simulations are executed using each set of generated group constants (and boundary conditions) to provide the corresponding homogeneous surface fluxes and homogeneous pin-cell fluxes.
  - The process is heavily automated: `krakentools.ants.evaluate_ffs_and_adfs_with_ants()`
  - ADFs and FFs can be evaluated based on known heterogeneous and homogeneous data.

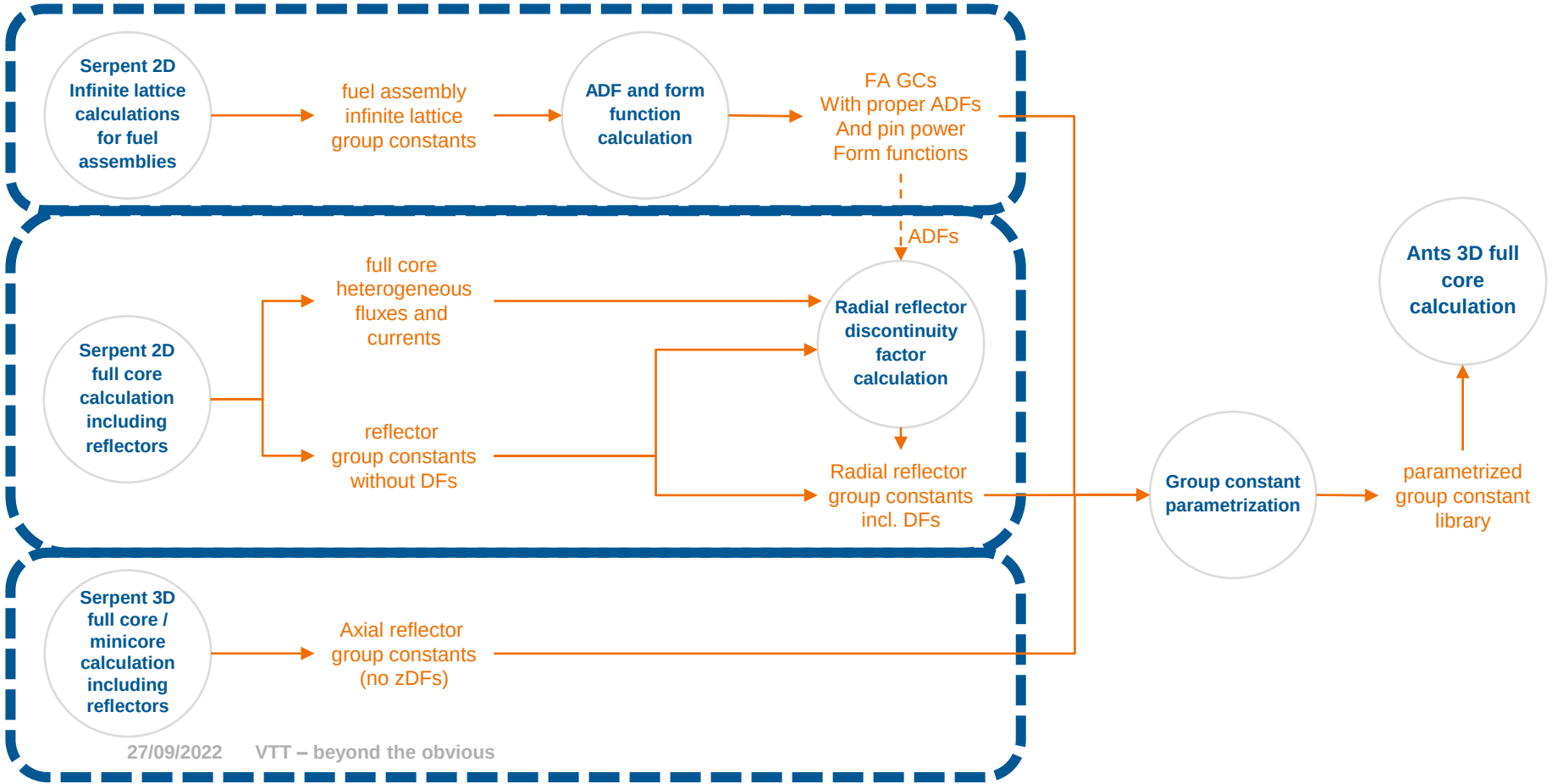
Fuel ADFs and pin power form functions



- Instead, Ants single node 2D simulations are executed using each set of generated group constants (and boundary conditions) to provide the corresponding homogeneous surface fluxes and homogeneous pin-cell fluxes.
  - The process is heavily automated: `krakentools.ants.evaluate_ffs_and_adfs_with_ants()`
  - ADFs and FFs can be evaluated based on known heterogeneous and homogeneous data.

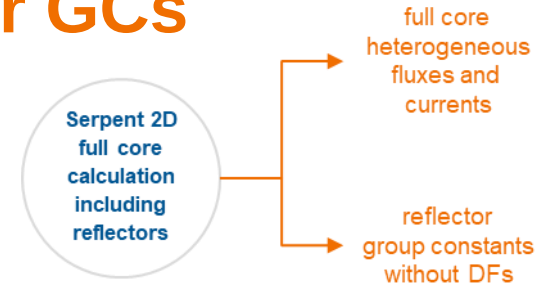
Heterogeneous data utilized from .coe files:
DF_HET_SURF_FLUX
PPW_POW

Group constant generation

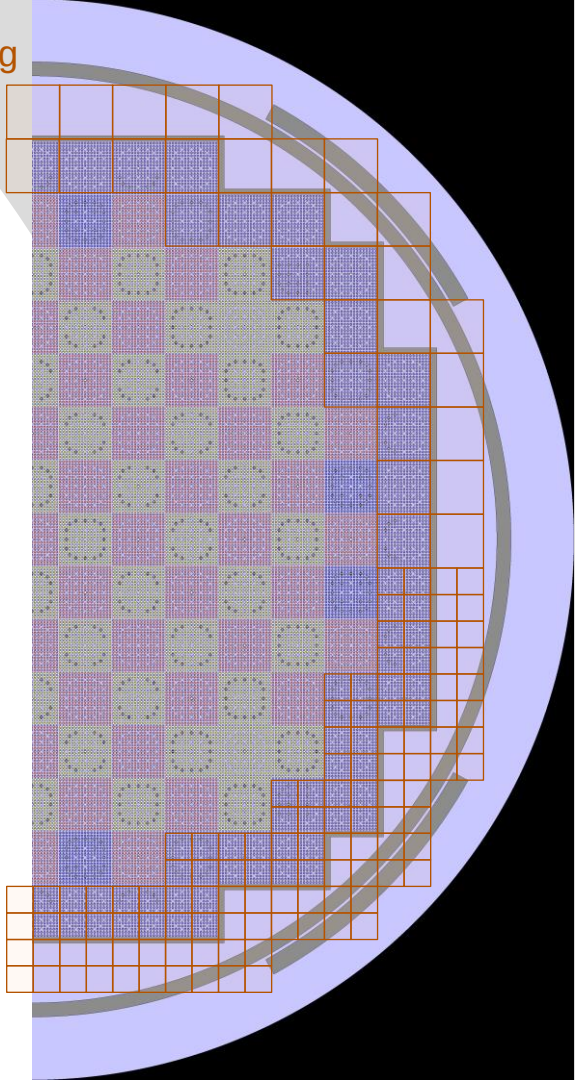


1x1 reflector meshing

Best practices calculation chain reflector GCs

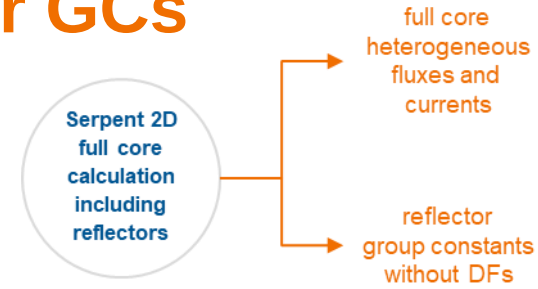


- Radial reflector constants are generated using a 2D full core geometry.
- Multiple transport solutions at zero burnup:
  - Cover different (T_{cool} , ρ_{cool} , C_B) variations.
- Diffusion coefficients transport corrected for H-1 in water.
  - set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010

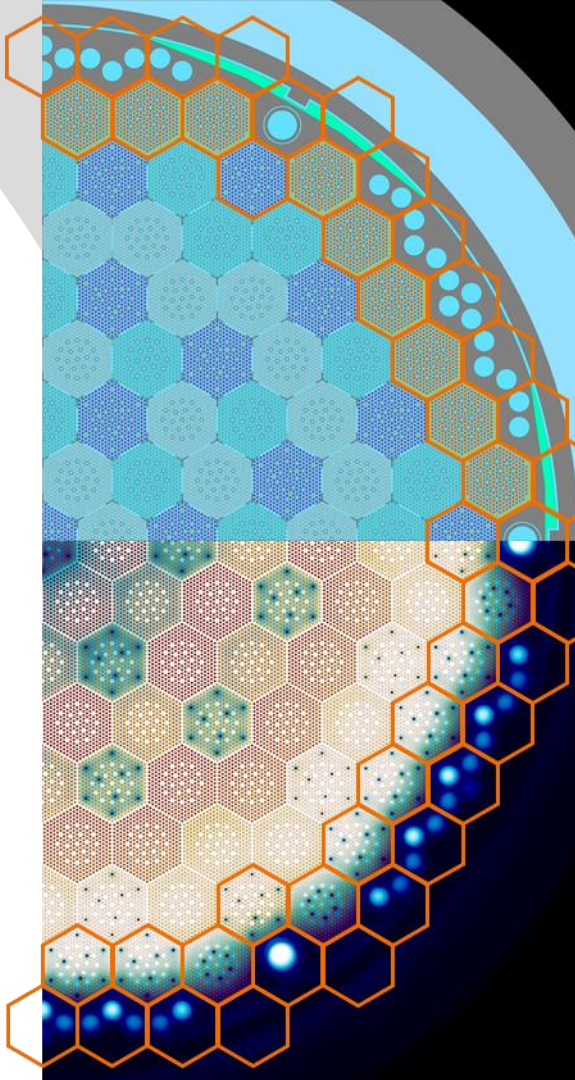


2x2 reflector meshing

Best practices calculation chain reflector GCs

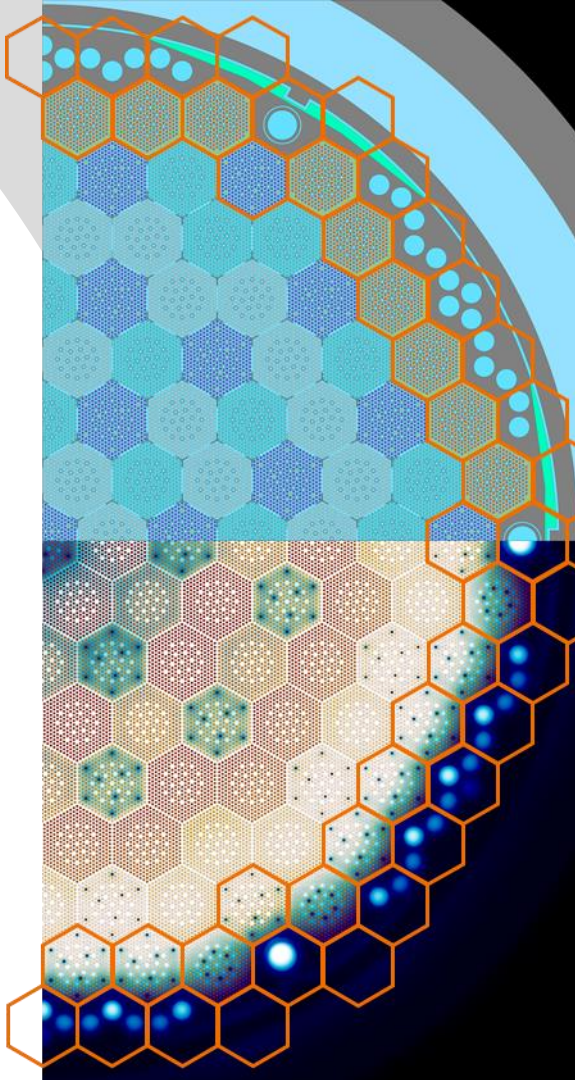


- Radial reflector constants are generated using a 2D full core geometry.
- Multiple transport solutions at zero burnup:
  - Cover different (T_{cool} , ρ_{cool} , C_B) variations.
- Diffusion coefficients transport corrected for H-1 in water.
  - set trc cool "s2v0_endfb71.h_in_h2o.trcdata" 1.000000E-11 10010
- Hexagonal lattice radial reflector currently homogenized using hexagonal nodes. In the future, also with triangular nodes.



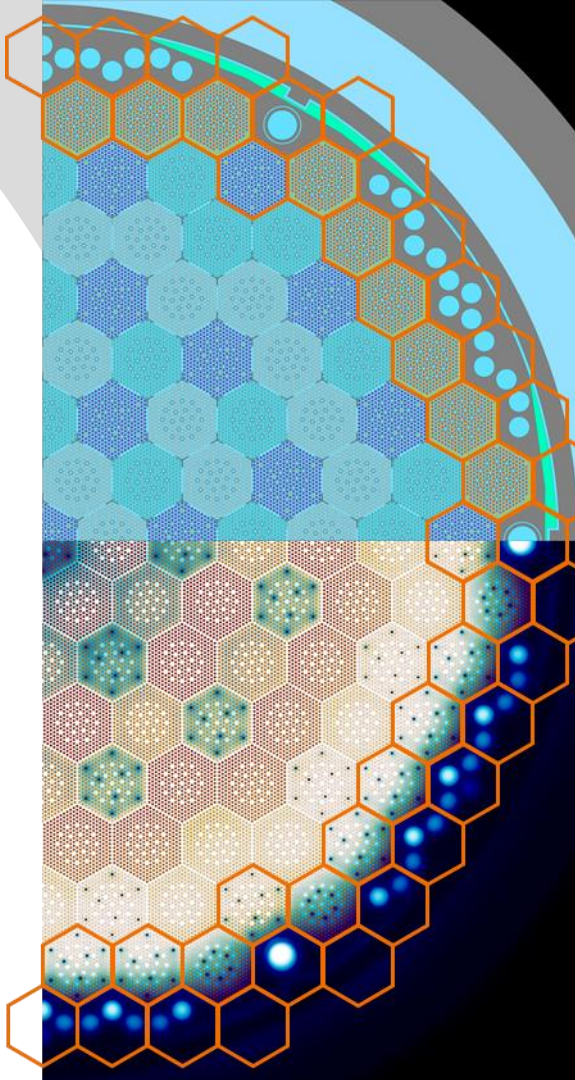
Superimposed universes for reflector group constants

```
% --- First define bounding surfaces for superimposed universes
%     (must not overlap)

% --- Surface bounding node RR01

surf s_bound_RR01 hexxprism 165.20000000000002 0.0 11.8 30 50

% --- Surface bounding node RR02

surf s_bound_RR02 hexxprism 177.0 20.43819952931275 11.8 30 50

% --- Surface bounding node RR03

surf s_bound_RR03 hexxprism 165.20000000000002 40.8763990586255 11.8 30 50

...
```



Superimposed universes for reflector group constants

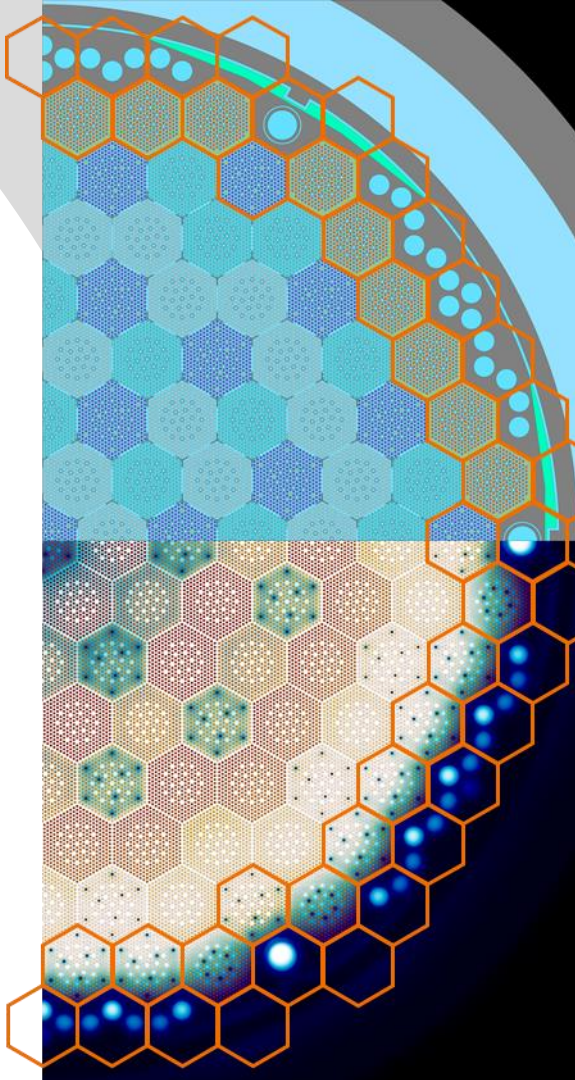
```
% --- The define (superimposed) universes based on the surfaces

% --- Superimposed universe for node RR01

cell  c_SI_RR01  -u_SI_RR01  void  -s_bound_RR01

% --- Superimposed universe for node RR02

cell  c_SI_RR02  -u_SI_RR02  void  -s_bound_RR02

% --- Superimposed universe for node RR03

cell  c_SI_RR03  -u_SI_RR03  void  -s_bound_RR03

...
```



Superimposed universes for reflector group constants

```
% --- Finally setup gcu and adf cards for the superimposed universes

set gcu  -u_SI_RR01
set adf  -u_SI_RR01 s_bound_RR01 0

set gcu  -u_SI_RR02
set adf  -u_SI_RR02 s_bound_RR02 0

set gcu  -u_SI_RR03
set adf  -u_SI_RR03 s_bound_RR03 0

...
```

- Universes linked to gcu or adf cards, but that are not part of the geometry are treated by Serpent as superimposed on top of the geometry.
- Some slowdown to simulations due to (additional) checking if collision is in a superimposed universe or crosses the boundary of one at each interaction site.

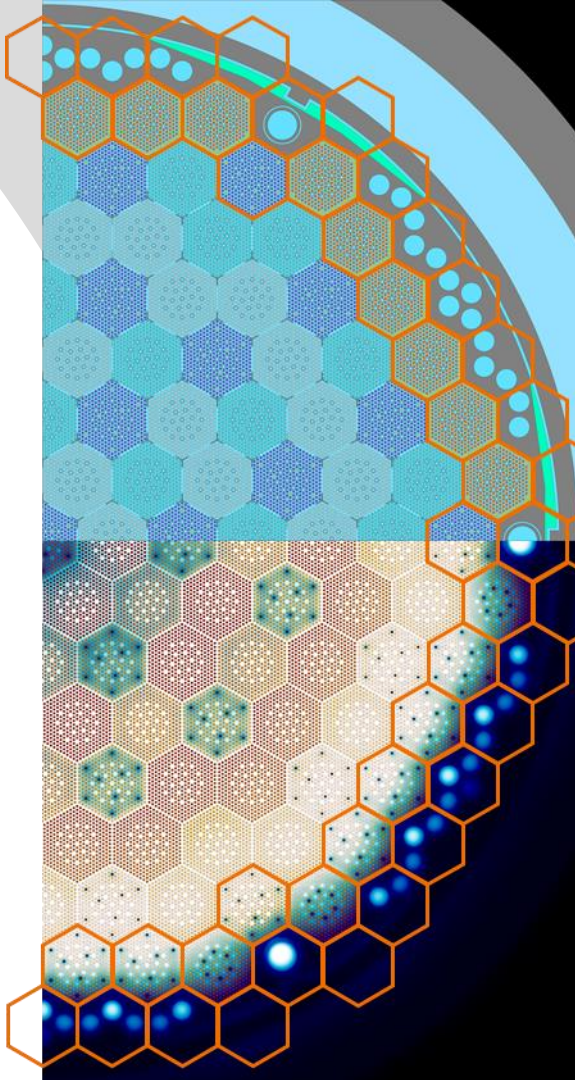


Output data

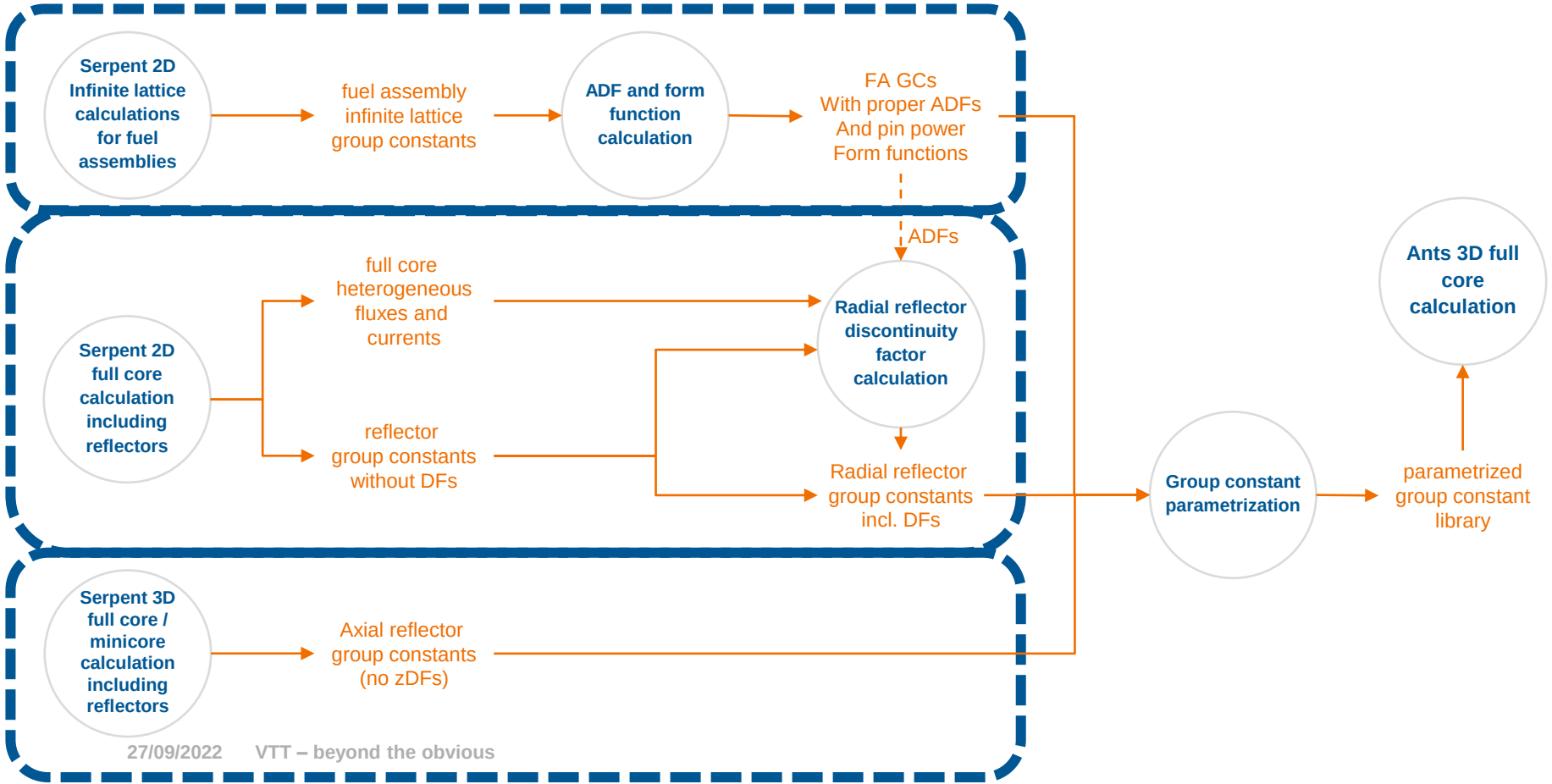
```
# sss2 -casematrix <case_name> <his_idx> <coe_idx> <input>
# Can run with

sss2 –omp 20 -casematrix reflector -1 <coe_idx> fullcore
```

- Reflector casematrix may not need > 0 burnups or fuel temperature branches.
- `fullcore_<case_name>_h<his_idx>_r<coe_idx>.coe` files
  - Contain homogenized few group constants for homogenized universes.
    - Includes group constants and heterogeneous node boundary fluxes and currents.
    - var definitions from branch cards show up in .coe files to help identify, which file contains which data.
- `fullcore_<case_name>_h<his_idx>_r<coe_idx>_res.m` files
  - Contain some other important data not directly bound to homogenized universes.

KrakenTools collects results from 360 degree core and averages results over symmetric positions

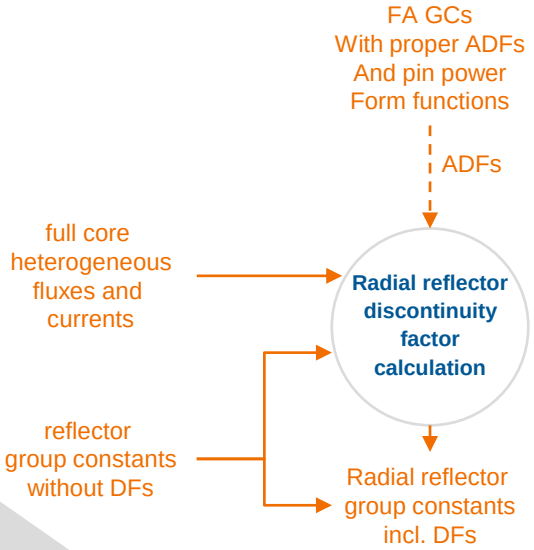


Group constant generation



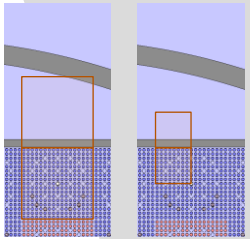
Best practices calculation chain reflector discontinuity factors

`krakentools.reflectorhg.solve_ants_2d_nodes()`



1. The reflector side DF is first evaluated simply as the ratio of the heterogeneous surface flux from the Serpent 3D solution and the homogeneous surface flux from a single node Ants calculation using group constants and boundary condition currents from the Serpent3D solution:

$$f_{\text{refl.}}^{\text{Ants}} = \frac{\phi_{\text{refl.}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{refl.}}^{\text{Ants}}}$$



2. The fuel side DF is similarly evaluated

$$f_{\text{fuel}}^{\text{Ants}} = \frac{\phi_{\text{fuel}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{fuel}}^{\text{Ants}}}$$

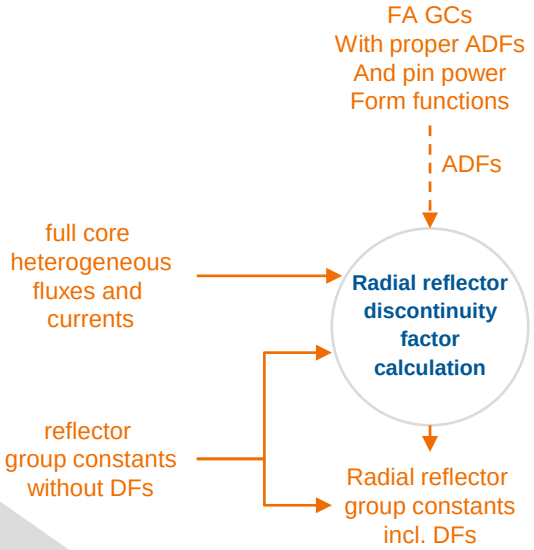
3. This DF is then corrected[8] by the ratio of the assembly discontinuity factor $f_{\text{fuel}}^{\text{ADF}}$ evaluated for the fuel assembly in the infinite lattice 2D Serpent calculation and $f_{\text{fuel}}^{\text{Ants}}$:

$$f_{\text{refl.}} = f_{\text{refl.}}^{\text{Ants}} \times \frac{f_{\text{fuel}}^{\text{ADF}}}{f_{\text{fuel}}^{\text{Ants}}}$$

[8] K. S. Smith. “Nodal diffusion methods and lattice physics data in LWR analyses: Understanding numerous subtle details”. *Progress in Nuclear Energy* 101 (2017), pp. 360–369

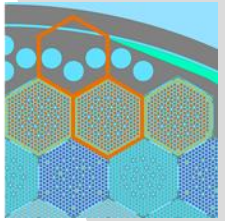
Best practices calculation chain reflector discontinuity factors

`krakentools.reflectorhg.solve_ants_2d_nodes()`



1. The reflector side DF is first evaluated simply as the ratio of the heterogeneous surface flux from the Serpent 3D solution and the homogeneous surface flux from a single node Ants calculation using group constants and boundary condition currents from the Serpent3D solution:

$$f_{\text{refl.}}^{\text{Ants}} = \frac{\phi_{\text{refl.}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{refl.}}^{\text{Ants}}}$$



2. The fuel side DF is similarly evaluated

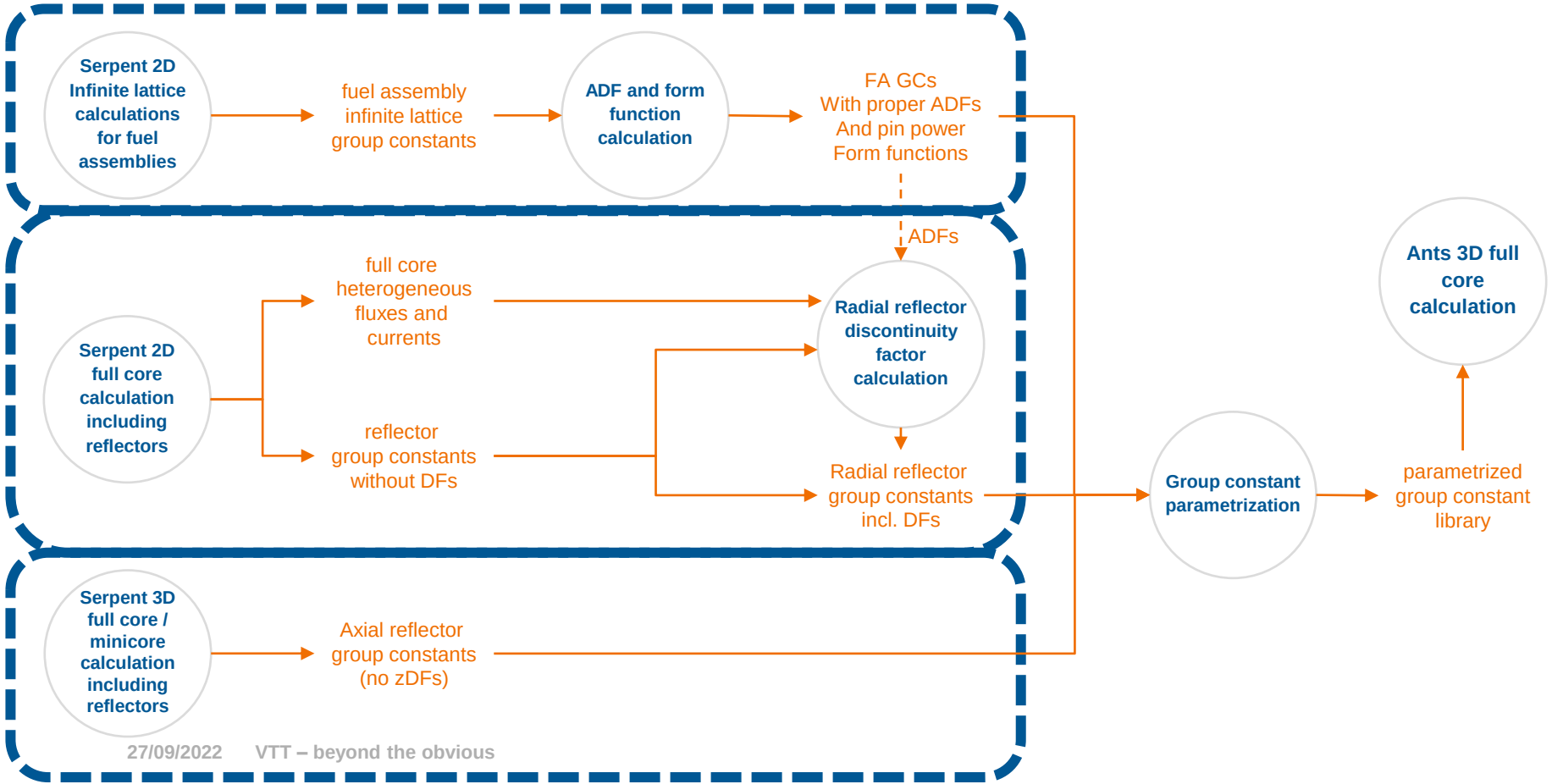
$$f_{\text{fuel}}^{\text{Ants}} = \frac{\phi_{\text{fuel}}^{\text{Serpent3D}}}{\boldsymbol{\Phi}_{\text{fuel}}^{\text{Ants}}}$$

3. This DF is then corrected[8] by the ratio of the assembly discontinuity factor $f_{\text{fuel}}^{\text{ADF}}$ evaluated for the fuel assembly in the infinite lattice 2D Serpent calculation and $f_{\text{fuel}}^{\text{Ants}}$:

$$f_{\text{refl.}} = f_{\text{refl.}}^{\text{Ants}} \times \frac{f_{\text{fuel}}^{\text{ADF}}}{f_{\text{fuel}}^{\text{Ants}}}$$

[8] K. S. Smith. “Nodal diffusion methods and lattice physics data in LWR analyses: Understanding numerous subtle details”. *Progress in Nuclear Energy* 101 (2017), pp. 360–369

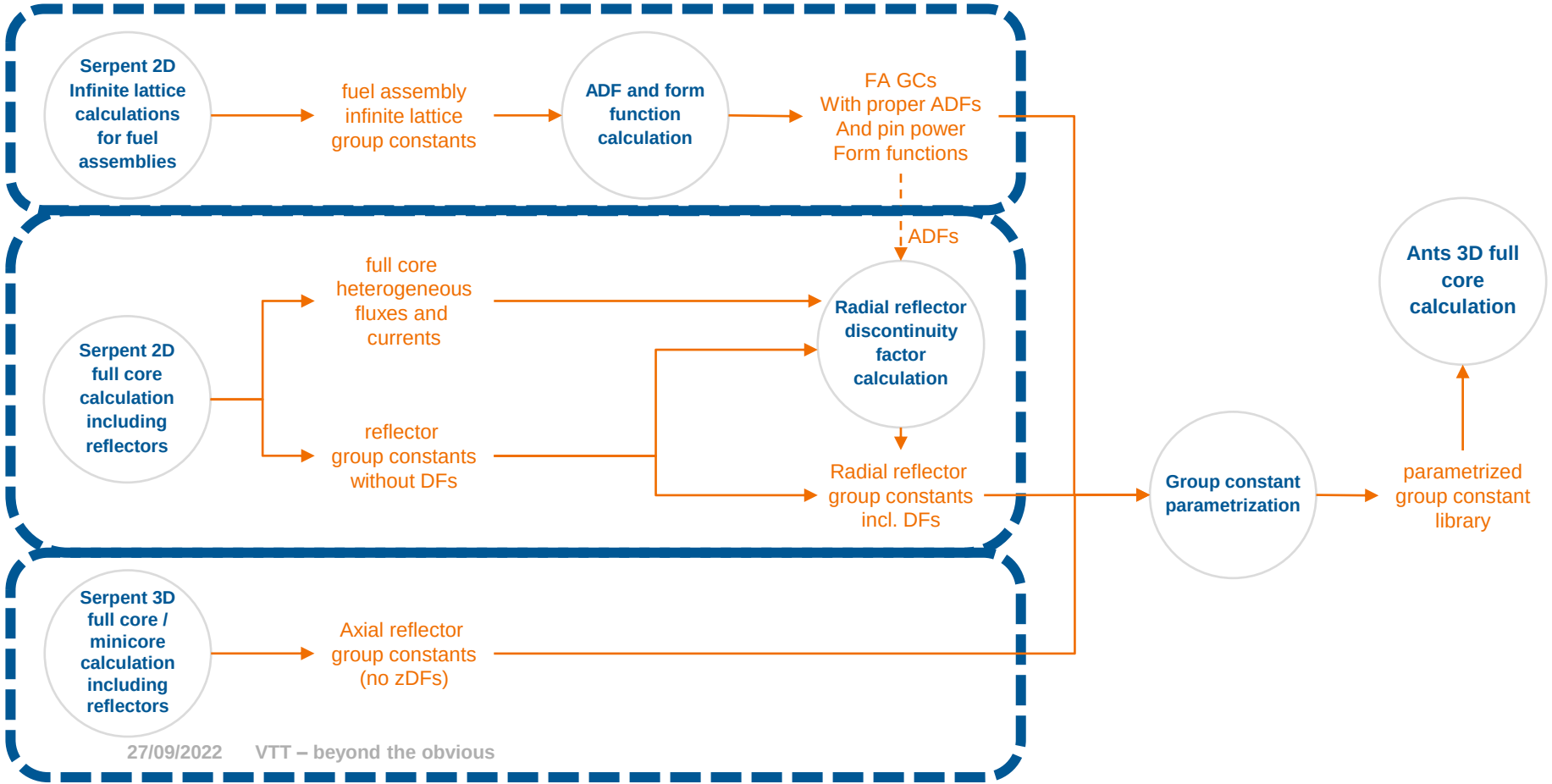
Group constant generation



Best practices calculation chain
Axial reflector homogenization

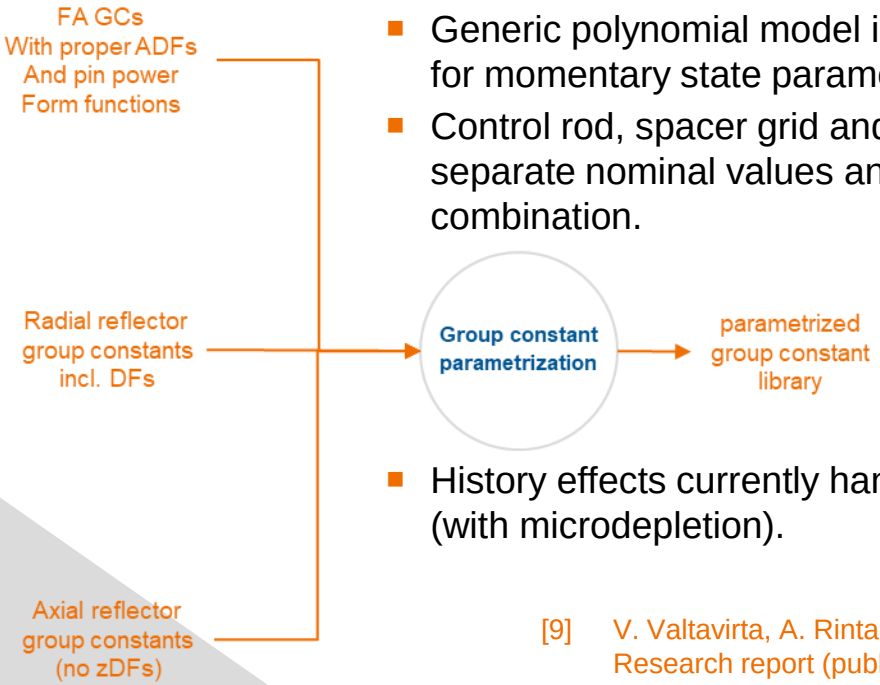
- Rather similar to radial reflector homogenization, but need a 3D model:
  - Single assembly.
  - Colorset.
  - Full core.
- May need control rod branches?
- Superimposed universes set up similar to radial reflector.
- Axial discontinuity factors could be calculated similar to radial ones.

Group constant generation



Group constant parametrization

krakentools.groupconstants.genpoly



- Generic polynomial model implemented in Ants[9] with a polynomial fit for momentary state parameters. (T_{fuel} , T_{cool} , ρ_{cool} , C_B).
- Control rod, spacer grid and instrumentation tube are treated as select variables with separate nominal values and polynomial coefficients tabulated for each possible combination.
- History effects currently handled using a plutonium history approach[10] (with microdepletion).

[9] V. Valtavirta, A. Rintala. “Specifications for the generic polynomial group constant model of Ants”, Research report (public), VTT-R-00154-21, 2021.

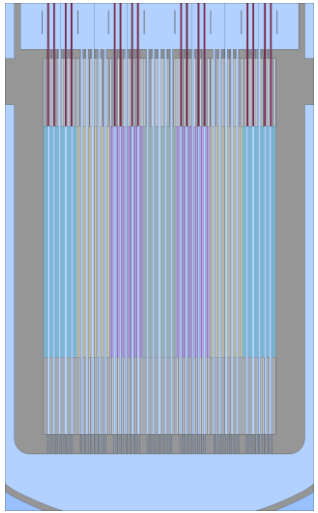
[10] Y. Bilodid. “Spectral history modelling in the reactor dynamics code DYN3D”, PhD thesis, Technical University of Dresden, 2014 (HZDR-051).

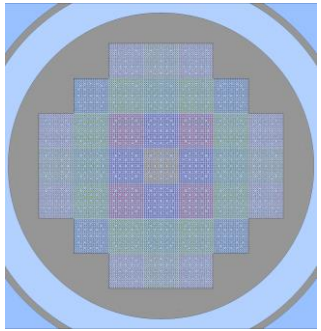
Run some Ants calculations

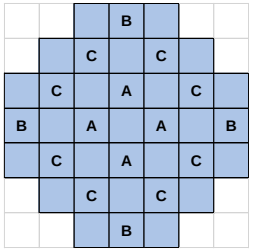
Example of Ants fuel cycle simulations

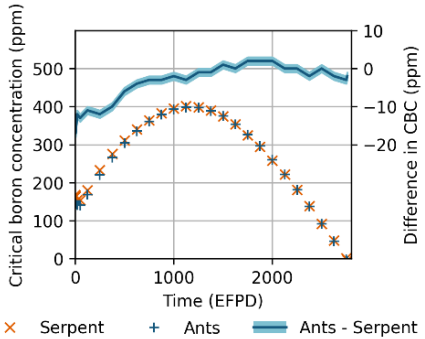
Valtavirta, V., Tuominen, R. “A simple reactor core simulator based on VTT's Cerberus Python package” ANS M&C 2021, April 11-15, 2021, Raleigh, NC

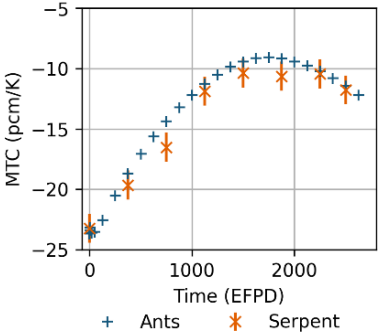
Automatic evaluation of licensing relevant data during the simulation of an SMR operating cycle. Verification by switching one physics from reduced order solver (Ants) to a high-fidelity one (Serpent), while Kharon and SuperFINIX models are kept constant

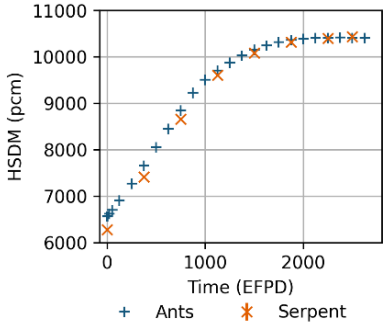


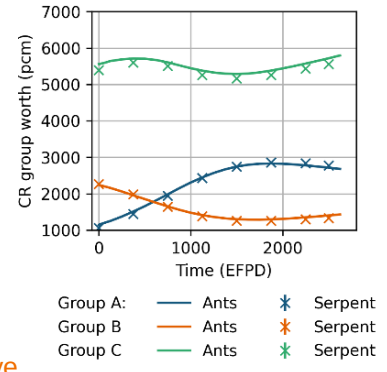












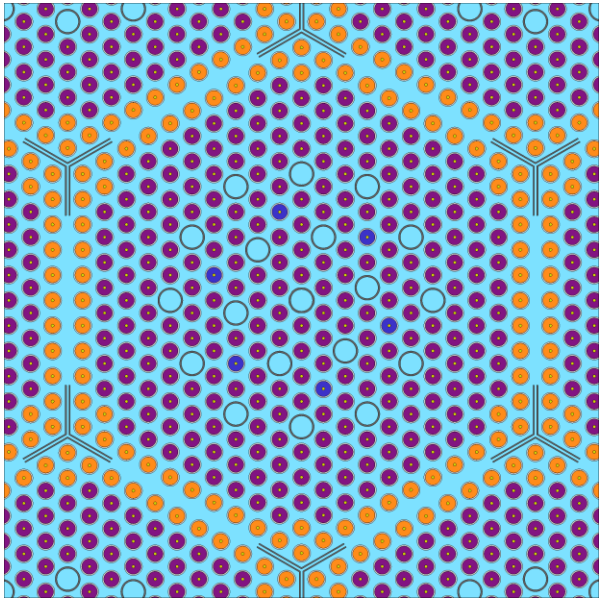
Top left: Boron letdown curve.
Top right: Moderator temperature reactivity coefficient.
Bottom left: Instantaneous hot shutdown margin.
Bottom right: Control rod group worths.

Bring back nuclide data from Ants to Serpent

Serpent model based on Ants results

- We might have nuclide densities for each node in the Ants simulation available through microdepletion.
- How to bring these back to Serpent?

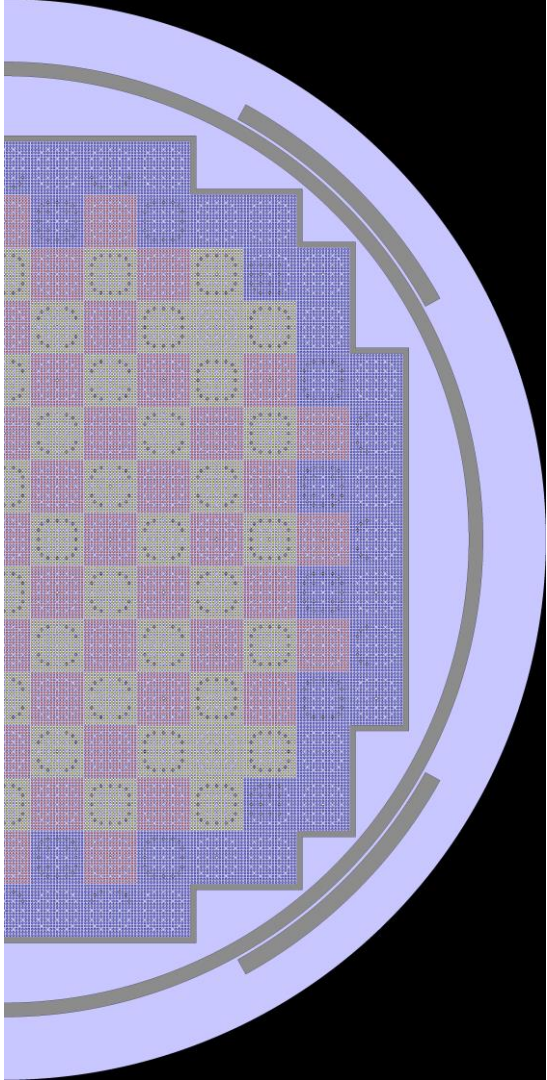
1. Serpent model with depletion zone division.
2. Setup correct material volumes for Serpent model.
3. Run a short decay calculation with “set rfw” to get binary restart file with atomic densities.
4. Figure out where each material zone is in Serpent by running “sss2 -matpos <coordinates_file> <input_file>”
5. Use KrakenTools to read in binary restart file and write the atomic densities you want for each material zone.
6. Run Serpent with “set rfr” to read atomic densities from binary restart.



Serpent model based on Ants results

- We might have nuclide densities for each node in the Ants simulation available through microdepletion.
- How to bring these back to Serpent?

1. Serpent model with depletion zone division.
2. Setup correct material volumes for Serpent model.
3. Run a short decay calculation with “set rfw” to get binary restart file with atomic densities.
4. Figure out where each material zone is in Serpent by running “sss2 -matpos <coordinates_file> <input_file>”
5. Use KrakenTools to read in binary restart file and write the atomic densities you want for each material zone.
6. Run Serpent with “set rfr” to read atomic densities from binary restart.



Summary

Summary

- Serpent has been developed for group constant generation from the start.
- In the recent years, the application of Serpent for such tasks at VTT has started in earnest.
- The process of generating group constants for fuel cycle simulations is starting to be pretty clear:
  - Fuel assemblies, reflector regions, proper DFs and form functions.
  - Use of `branch` cards for setting up history and branch conditions.
  - Use of `casematrix` to set up the calculation matrix and run it efficiently.
- Still a good amount of work in the future:
  - Effects of statistics.
  - Time constants (lambdas and betas with ENDF/B have issues).
  - Work thus far on PWRs and VVERs. BWRs need own parametrization.
    - Not to mention non-LWR applications.

beyond
the obvious

Ville Valtavirta
ville.valtavirta@vtt.fi