Kraken workshop

Python level introduction to coupled calculations with Kraken using Cerberus

Ville Valtavirta

Outline

- Cerberus Python package:
  - Idea
  - Some important Cerberus classes:
    - Solver()
    - Transferrable()
    - About Field()s and Mesh()es
    - Interpolator()

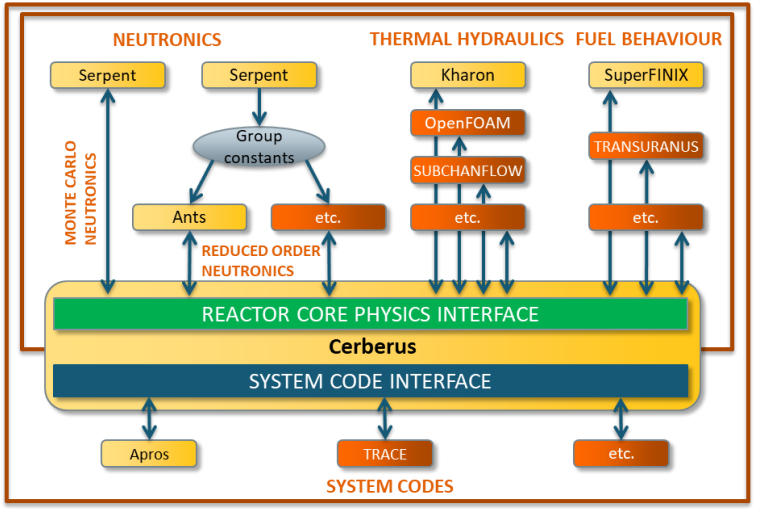
- Serpent and Cerberus
  - Three hands on tutorials.

Cerberus Python package

Cerberus Python package

- Code agnostic multi-physics driver of the Kraken framework.
- Provides high-level API for solvers, fields and variables on Python side.
- Python makes building coupled calculation schemes simple and fun.

- Cerberus aims to hide the most boring and technical stuff from the user.
- Strikes a balance between simplicity and flexibility.
- Aimed for expert users, who can package common calculation sequences into further Python packages/modules for non-expert users.

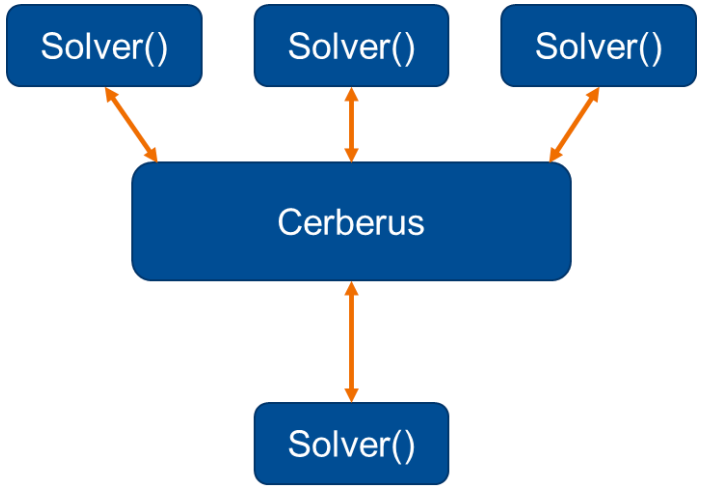


A schematic representation of the Kraken framework. Finnish solver modules developed at VTT are shown in yellow, while potential state-of-the-art third-party solvers to be coupled are shown in orange.

Cerberus Python package

- Code agnostic multi-physics driver of the Kraken framework.
- Provides high-level API for solvers, fields and variables on Python side.
- Python makes building coupled calculation schemes simple and fun.

- Cerberus aims to hide the most boring and technical stuff from the user.
- Strikes a balance between simplicity and flexibility.
- Aimed for expert users, who can package common calculation sequences into further Python packages/modules for non-expert users.



The Solver class of Cerberus

- All solvers participating in the calculation are based on the Solver class that provides methods such as
  - Solver.initialize()
  - Solver.get_transferrable()
  - Solver.solve()
  - Solver.set_current_time()
  - Solver.suggest_next_time()
  - Solver.move_to_time()
  - Solver.write_restart()
  - Solver.read_restart()
  - Etc.

The user does not need to know what happens “under the hood” when calling one of these methods from Python.

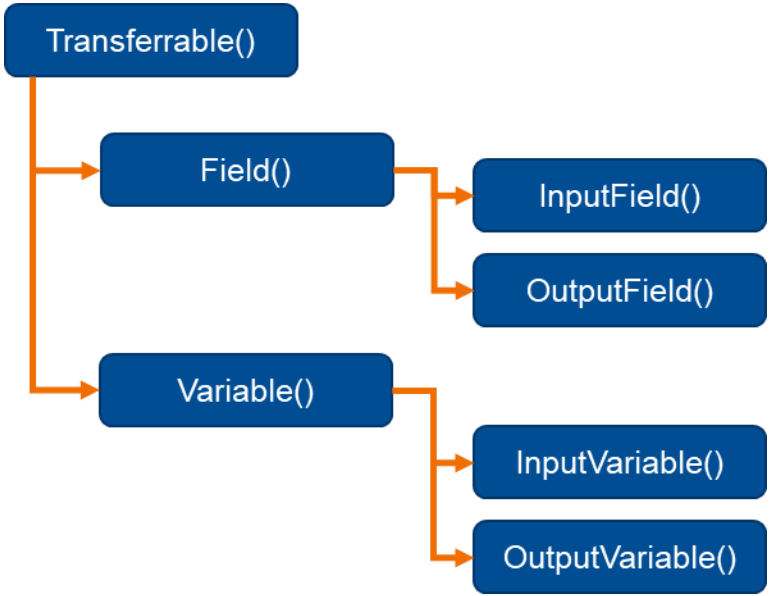
All solvers provide the same functionality to Python even if actual implementation in the solver module may differ.

Cerberus does not know (or care) which solver handles neutronics and which thermal hydraulics.

The expert user, of course, does care.

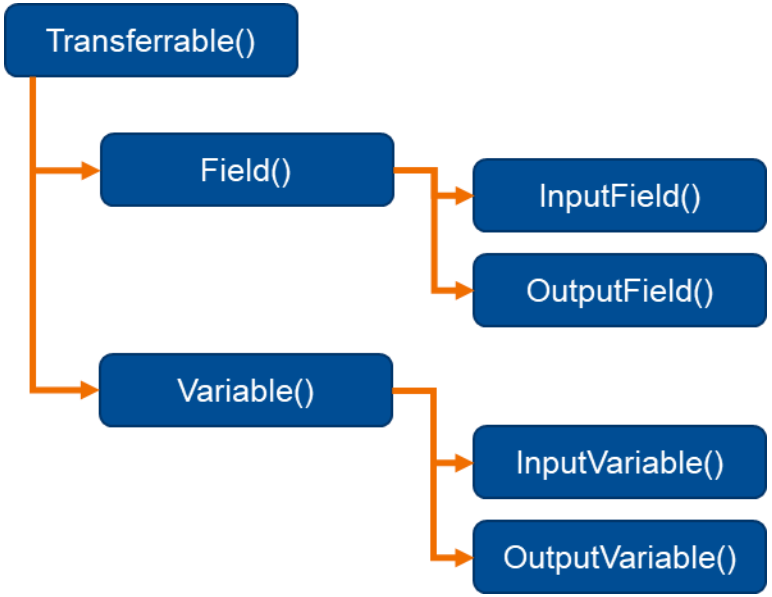
Transferrables in Cerberus

- The transferrable superclass covers all input and output data needs of the solver that can be transferred between the solver and Cerberus.
- Transferrable.communicate(): Exchange data with solver.
- Transferrable.write_simple(): Write data to file.
- Transferrable.write_foam(): Write data to Foam files.
- Transferrable.value_vec: Current values of data.
- Transferrable.get_conv_crit(): Evaluate convergence criterion between current and previous values.
- …



Transferrables in Cerberus

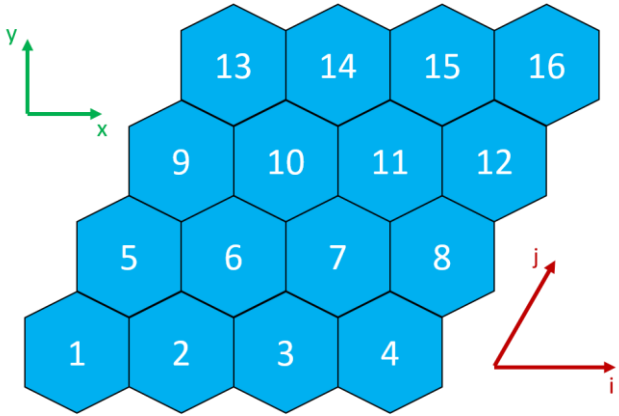
- Field() and Variable() are sub-classes of Transferrable().
- Field is a (physical) dataset with a spatial representation (mesh).
- Variable is a more general set/piece of data. Often single valued.



On Field()s and Mesh()es

- Field data on Python side (as seen by the user) is in SI-units and uses a default (global) indexing for the mesh.
- Conversions between solver (local) and Cerberus (global) units and indexings are handled automatically by Cerberus based on data Cerberus obtains from the solver.
- Cerberus output can be written to files separately using the global and local indexings (typically only global indexing is interesting).

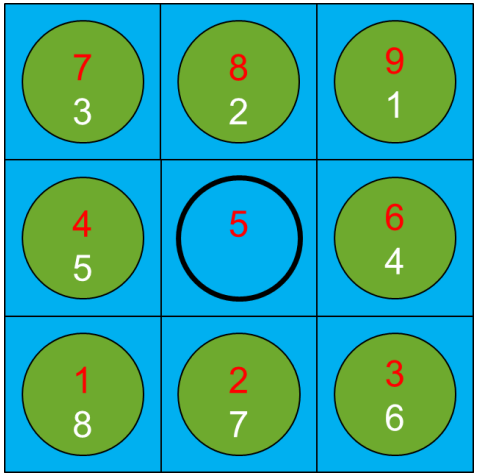
- https://serpent.vtt.fi/kraken/index.php/Mesh_types



Global indexing for mesh type 3: One axial layer of structured x-type 60 degree hexagonal mesh.

On Field()s and Mesh()es

- Meshes are used for automatic generation of interpolations between fields in simple cases.
- More complex interpolations need to be pre-generated by the user.
- Mesh information is also written in files and can be re-created in postprocessing using krakentools.meshes.Mesh class, which offers some useful functionalities for plotting etc.
- Automated output to FoamFiles is supported for some structured mesh types.



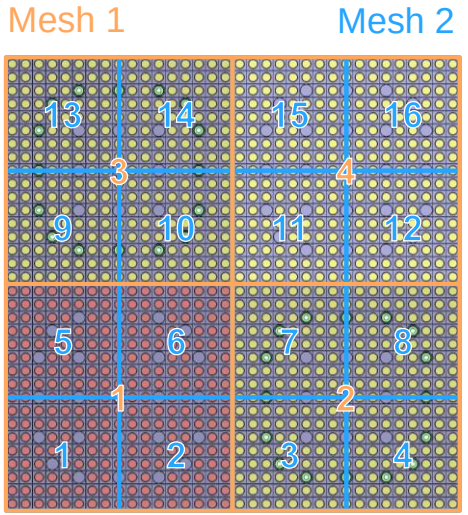
Cerberus will automatically handle conversions between solver indexing to global indexing to provide one uniform indexing scheme across all solvers and fields on the Python side.

The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ \vdots \\ d_{N_d} \end{pmatrix} = \begin{pmatrix} a_{1\to1} & \cdots & a_{N_s\to1} \\ \vdots & \ddots & \vdots \\ a_{1\to N_d} & \cdots & a_{N_s\to N_d} \end{pmatrix} \begin{pmatrix} s_1 \\ \vdots \\ s_{N_s} \end{pmatrix}$$

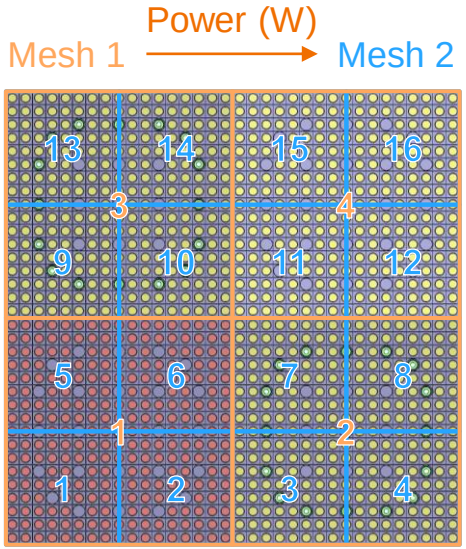


The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} a_{1\to1} & a_{2\to1} & a_{3\to1} & a_{4\to1} \\ a_{1\to2} & a_{2\to2} & a_{3\to2} & a_{4\to2} \\ a_{1\to3} & a_{2\to3} & a_{3\to3} & a_{4\to3} \\ a_{1\to4} & a_{2\to4} & a_{3\to4} & a_{4\to4} \\ a_{1\to5} & a_{2\to5} & a_{3\to5} & a_{4\to5} \\ a_{1\to6} & a_{2\to6} & a_{3\to6} & a_{4\to6} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1\to16} & a_{2\to16} & a_{3\to16} & a_{4\to16} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

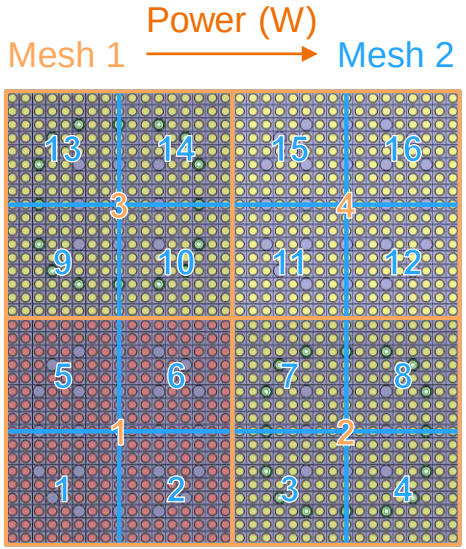


The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0.25 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$



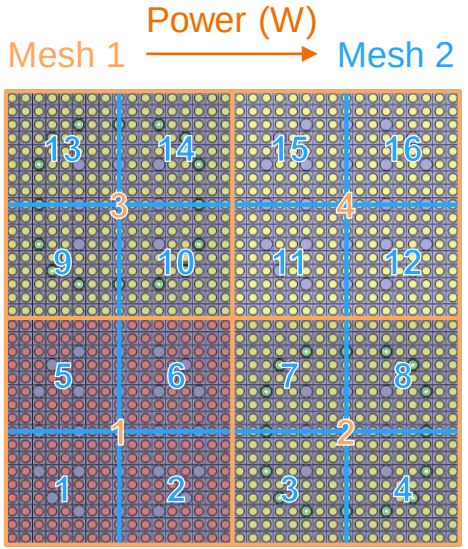
The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0.25 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

mapping.txt:
1 0.25
1 0.25
2 0.25
2 0.25
1 0.25
1 0.25
…
4 0.25



Using Serpent through Cerberus

Running solvers with Cerberus

- Cerberus copies the solver inputs to a work folder (./wrk_<solver_name>/)

- Cerberus spawns the solver processes in the work folders.
  - Solver (terminal) output is directed to ./wrk_<solver_name>/out.txt
  - Solver system errors are directed to ./wrk_<solver_name>/err.txt
    - Note that solvers can print out warnings / errors to their out.txt.

- Cerberus connects to solvers using sockets.
  - Adding -port or --port to solver command line arguments Solver(add_params=[...]) will have Cerberus pass [“-port“, “<port_number>”] or [“--port“, “<port_number>”] as command line arguments when spawning the process.
  - First port used by Cerberus can be set with cerberus.PORT_NUMBER: int = <port-number>.
    - Default starting port is 2211.
    - The port for connection is incremented after each solver is spawned.

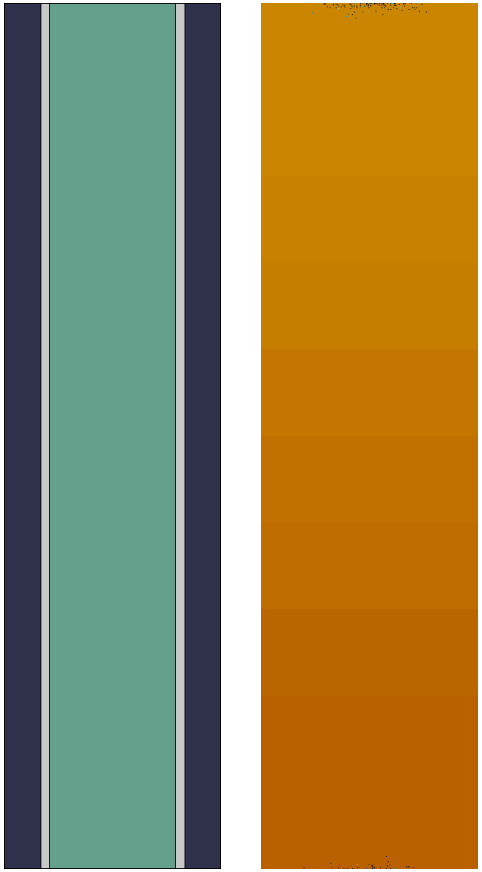
- Cerberus output is printed to terminal output.
  - verbosity can be controlled with cerberus.LOG.set_verbosity(level: int)

Multi-physics with Cerberus

- Redoing the familiar old Minimal Serpent Coupling Script (https://serpent.vtt.fi/mediawiki/index.php/Minimal_Serpent_Coupling_Script) with Cerberus.
  - Radially reflected pin cell with a very simple thermal hydraulics solution coupled to Serpent neutronics.
  - Water temperature and density updated via a Serpent multi-physics interface.

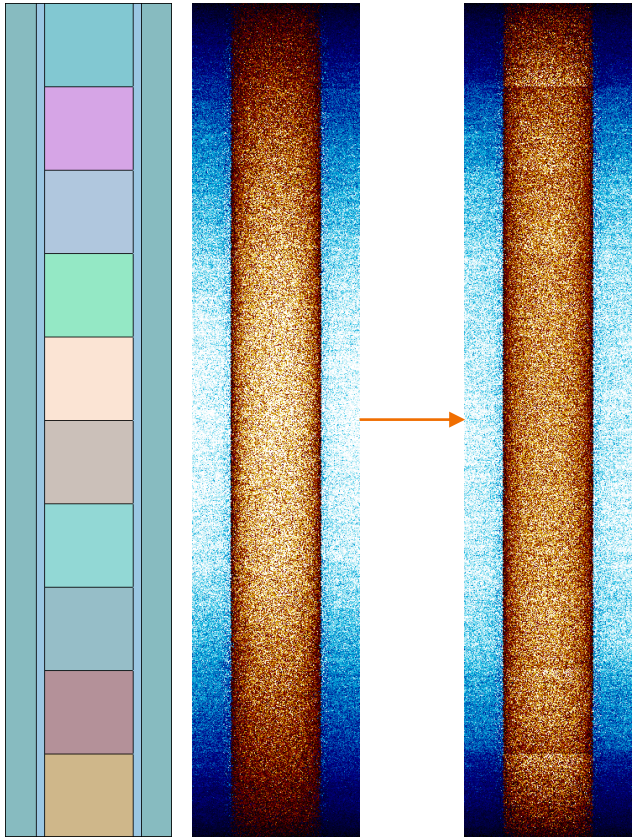
- No need for handling of files or signals with Cerberus.
  - From 287 to 103 lines.

- Setting up cool.ifc for Serpent input means that we get:
  - **sss_if_cool_temperature** input field (SI unit K)
  - **sss_if_cool_density** input field (SI unit kg/m3)
  - **sss_of_cool_power** output field (SI unit W)
    - The name of this field is actually formed based on the output file name specified in cool.ifc



Adjusting a running Serpent calculation

- Axially homogeneous and axially black pin cell.
  - Should produce a cosine shaped power distribution.
- Here we adjust ^{235}U content to achieve a flat axial power profile for fun.
  - Simply showcases modifying material compositions during runtime / between neutronics iterations.
- Having **det fission dr -6 void dz 0.0 100.0 10** gives us:
  - **sss_ov_DET_fission** output variable
  - **sss_ov_DET_fission_rel_unc** output variable
- Using a div-card in Serpent input to divide fuel into 10 axial zones gives us:
  - **sss_ov_composition_fuelz<index>_nuclides** output variables
  - **sss_ov_composition_fuelz<index>_zai** output variables
  - **sss_ov_composition_fuelz<index>_adens** output variables
  - **sss_ov_composition_fuelz<index>_original** output variables
  - **sss_iv_composition_fuelz<index>_adens** input variables



Adjusting a running Serpent calculation

- Axially homogeneous and axially black pin cell.
  - Should produce a cosine shaped power distribution.
- Here we adjust ^{235}U content to achieve a flat axial power profile for fun.
  - Simply showcases modifying material compositions during runtime / between neutronics iterations.
- Having **det fission dr -6 void dz 0.0 100.0 10** gives us:
  - **sss_ov_DET_fission** output variable
  - **sss_ov_DET_fission_rel_unc** output variable
- Using a div-card in Serpent input to divide fuel into 10 axial zones gives us:
  - **sss_ov_composition_fuelz<index>_nuclides** output variables
  - **sss_ov_composition_fuelz<index>_zai** output variables
  - **sss_ov_composition_fuelz<index>_adens** output variables
  - **sss_ov_composition_fuelz<index>_original** output variables
  - **sss_iv_composition_fuelz<index>_adens** input variables

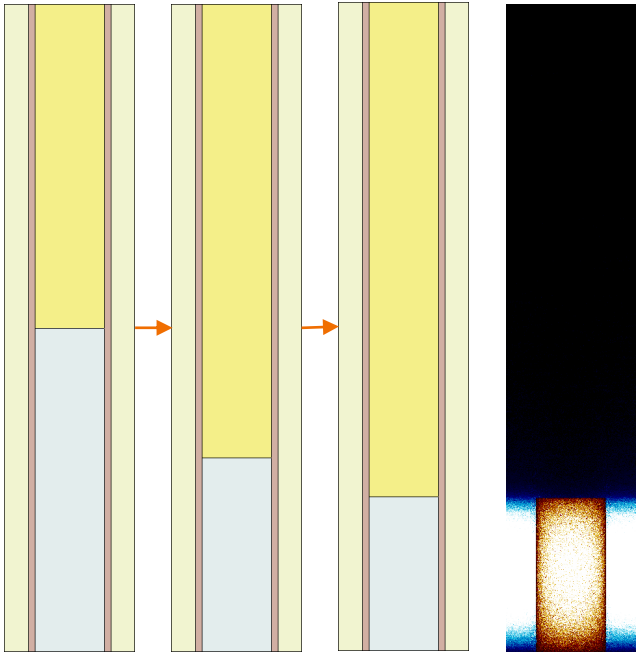
```
Iteration 1
axial zone  relative power   U-235 content
        10            0.34     6.84319E-04
         9            0.80     6.84319E-04
         8            1.11     6.84319E-04
         7            1.31     6.84319E-04
         6            1.44     6.84319E-04
         5            1.42     6.84319E-04
         4            1.36     6.84319E-04
         3            1.11     6.84319E-04
         2            0.77     6.84319E-04
         1            0.33     6.84319E-04

...

Iteration 20
axial zone  relative power   U-235 content
        10            0.59     1.59851E-03
         9            0.94     7.88999E-04
         8            1.08     6.01906E-04
         7            1.13     5.38819E-04
         6            1.13     5.19114E-04
         5            1.15     5.17418E-04
         4            1.17     5.36125E-04
         3            1.15     5.99989E-04
         2            1.01     7.82835E-04
         1            0.64     1.58008E-03
```

Adjusting a running Serpent calculation geometrical adjustment

- Axially black pin cell, with 50 % fuel and 50 % boron carbide
  - Supercritical to start with.
  - Iterate boundary of fuel and boron carbide to get critical system.

- Adding a named transformation card `trans s s2 name my_trans 0.0 0.0 50.0` in Serpent gives us:
  - **sss_iv_trans_my_trans** input variable
    - Sending e.g. 0 0 10 as values for the variable sets the z-part of the transformation to 10.

- We'll send 1 to **sss_iv_plot_geometry** to have Serpent plot the geometry after each transformation.



Adjusting a running Serpent calculation geometrical adjustment

- Axially black pin cell, with 50 % fuel and 50 % boron carbide
  - Supercritical to start with.
  - Iterate boundary of fuel and boron carbide to get critical system.

- Adding a named transformation card `trans s s2 name my_trans 0.0 0.0 50.0` in Serpent gives us:
  - **sss_iv_trans_my_trans** input variable
    - Sending e.g. 0 0 10 as values for the variable sets the z-part of the transformation to 10.

- We'll send 1 to **sss_iv_plot_geometry** to have Serpent plot the geometry after each transformation.

```
Iteration 1:
Keff is 1.26489, relative uncertainty is
0.001 with boundary at 50.0 cm.

Iteration 2:
Keff is 1.10747, relative uncertainty is
0.001 with boundary at 30.0 cm.

...

Iteration 10:
Keff is 0.99928, relative uncertainty is
0.002 with boundary at 23.8 cm.
```

Future ideas for Serpent and Cerberus

- More output data can be exposed to be accessed from Cerberus.
- Additional input data can also be exposed with some limitations:
  - Updating some input data will need (re)processing.
  - The more complex the processing required, the more work it is to implement.
  - Straightforward (evaluated during transport):
    - Modifying surface parameters, weight window mesh weights.
  - More complex (needs some processing):
    - Adjusting unstructured geometries (needs rebuilding of search mesh).
    - Bringing in source neutrons/photons from Python (new source sampling routine).
  - Problematic (needs memory allocation):
    - Adding new detectors, materials, surfaces etc. (can maybe be circumvented by adding some dummies in initial input)
- Some new “commands” will probably be included (e.g. write restart)
- Could write a Serpent Class, that is much more straightforward to use
  - serpent.detectors[<some_detectors>]
  - serpent.source_particles[<particle N>]
  - serpent.weight_window_mesh[<cell N>]

VTT

beyond

the obvious

Ville Valtavirta
Ville.Valtavirta@vtt.fi
Kraken@vtt.fi

@VTTFinland

www.vtt.fi