

# Kraken workshop

Python level introduction to coupled calculations with Kraken using Cerberus

Ville Valtavirta

24/05/2022 VTT – beyond the obvious

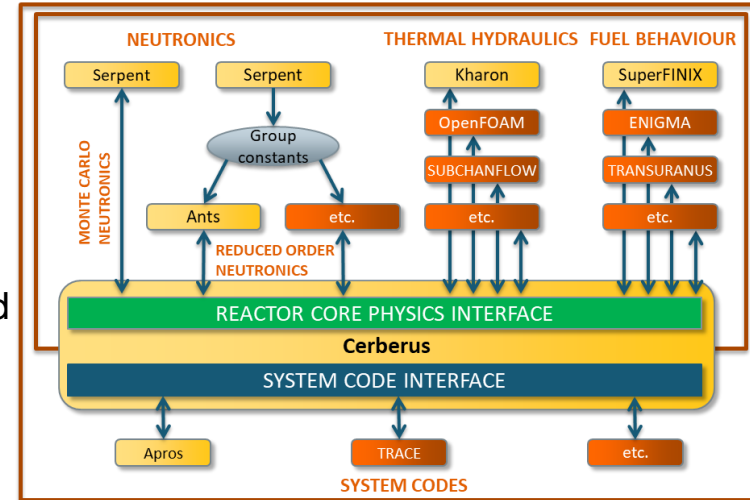
# Outline

- Cerberus Python package:
  - Idea
  - Capabilities:
    - Solver()
    - Transferrable()
    - About Field()s and Mesh()es
    - Interpolator()
- Setting up coupled calculations with Cerberus:
  - Critical boron iteration.
  - Control rod iteration.
  - Transient simulations.
  - Burnup calculations.

# Cerberus Python package

# Cerberus Python package

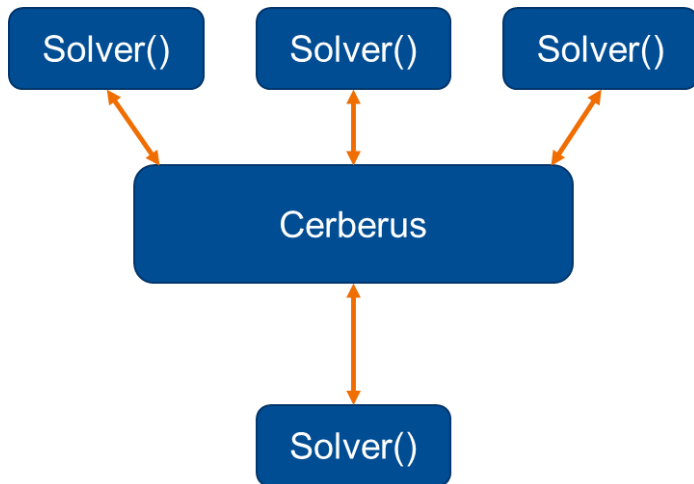
- **Code agnostic multi-physics driver** of the Kraken framework.
- Provides high-level API for solvers, fields and variables on Python side.
- Python makes building coupled calculation schemes simple and fun.
- Cerberus aims to hide most of the boring and technical stuff from the user.
  - E.g. actual communications between processes, copying files and creating folders.
- 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 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.

# 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 most of the boring and technical stuff from the user.
  - E.g. actual communications between processes, copying files and creating folders.
- 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.

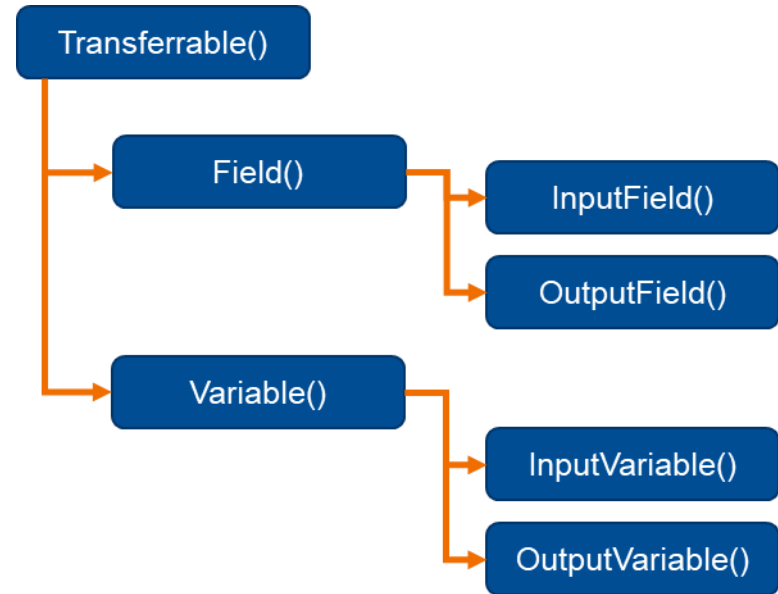
All solvers provide the same functionality to Python even if actual implementation in the solver module may differ.

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

The expert user, of course, does care.

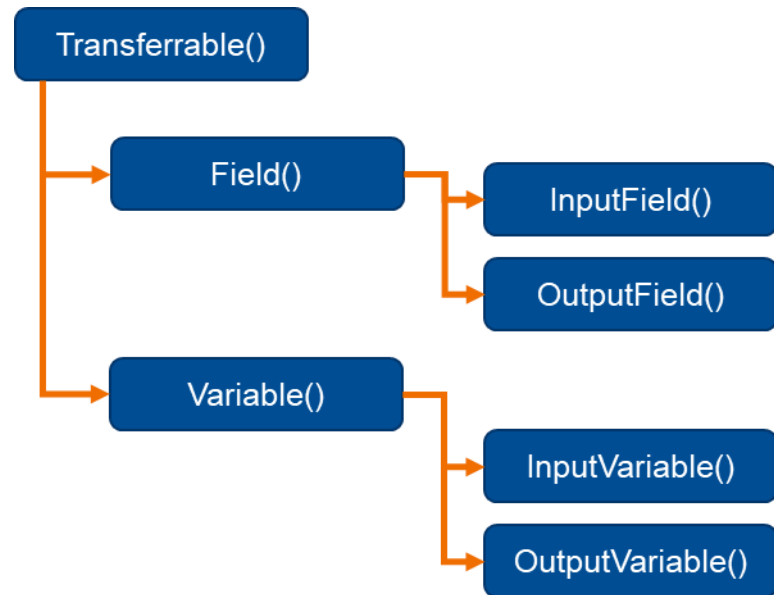
# Transferrables in Cerberus

- 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

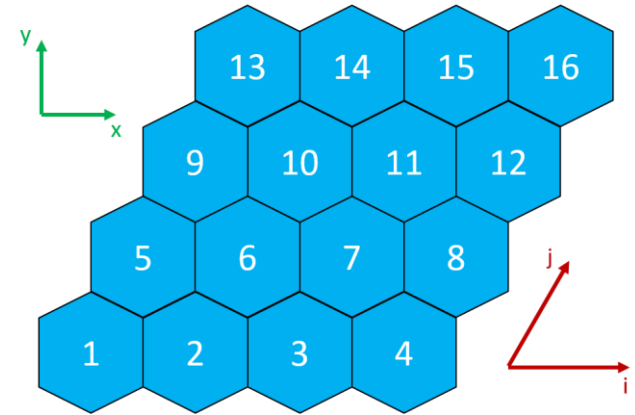
- `Field()` and `Variable()` are sub-classes of `Transferrable()`.
- `Field` is a (physical) dataset with a spatial representation (mesh).
- `Variable` is a more general set/piece of data. Often single valued.





# On Field()s and Mesh()es

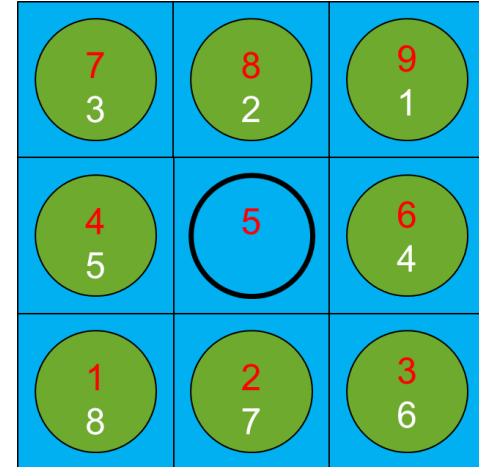
- Field data on Python side (**as seen by the user**) is in SI-units and uses a default (global) indexing for the mesh.
- Conversions between solver (local) and Cerberus (global) units and indexings are handled automatically by Cerberus based on data Cerberus obtains from the solver.
- Cerberus output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



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

# On Field()s and Mesh()es

- Meshes may be used for automatic generation of interpolations between fields in the future. At the moment, 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.kraken.Mesh` class, which offers some useful functionalities for plotting etc.
- Automated output to FoamFiles is partially supported with additional support added in the future.



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

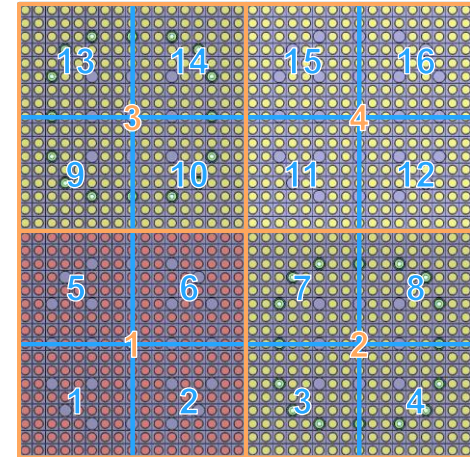
- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix  $\bar{\bar{A}}$  that describes the production of the destination data field  $\bar{d}$  from the source data field  $\bar{s}$ .

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

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

Mesh 1

Mesh 2

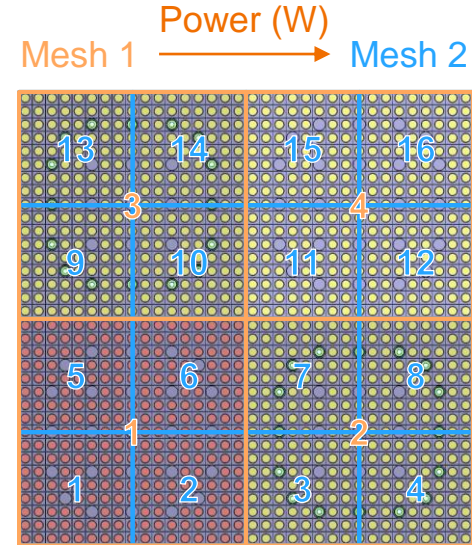


# The Interpolator() class

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

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

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} a_{1 \rightarrow 1} & a_{2 \rightarrow 1} & a_{3 \rightarrow 1} & a_{4 \rightarrow 1} \\ a_{1 \rightarrow 2} & a_{2 \rightarrow 2} & a_{3 \rightarrow 2} & a_{4 \rightarrow 2} \\ a_{1 \rightarrow 3} & a_{2 \rightarrow 3} & a_{3 \rightarrow 3} & a_{4 \rightarrow 3} \\ a_{1 \rightarrow 4} & a_{2 \rightarrow 4} & a_{3 \rightarrow 4} & a_{4 \rightarrow 4} \\ a_{1 \rightarrow 5} & a_{2 \rightarrow 5} & a_{3 \rightarrow 5} & a_{4 \rightarrow 5} \\ a_{1 \rightarrow 6} & a_{2 \rightarrow 6} & a_{3 \rightarrow 6} & a_{4 \rightarrow 6} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1 \rightarrow 16} & a_{2 \rightarrow 16} & a_{3 \rightarrow 16} & a_{4 \rightarrow 16} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

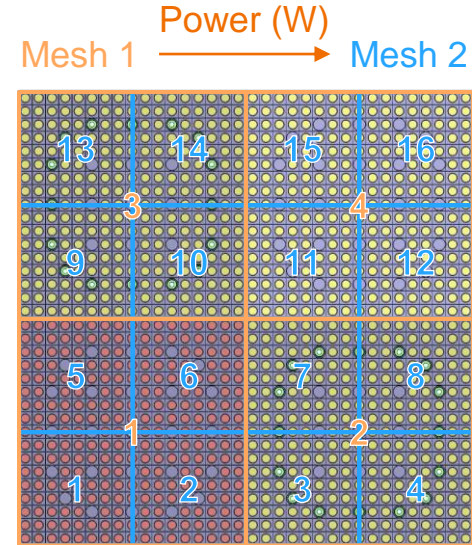


# The Interpolator() class

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

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

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



# The Interpolator() class

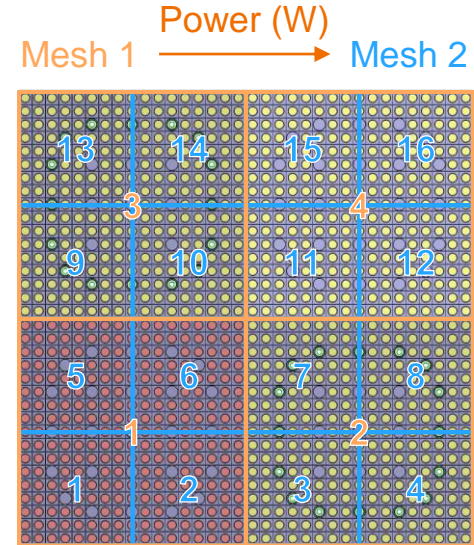
- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix  $\bar{\bar{A}}$  that describes the production of the destination data field  $\bar{d}$  from the source data field  $\bar{s}$ .

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

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

mapping.txt:

```
1 0.25
1 0.25
2 0.25
2 0.25
1 0.25
1 0.25
...
4 0.25
```



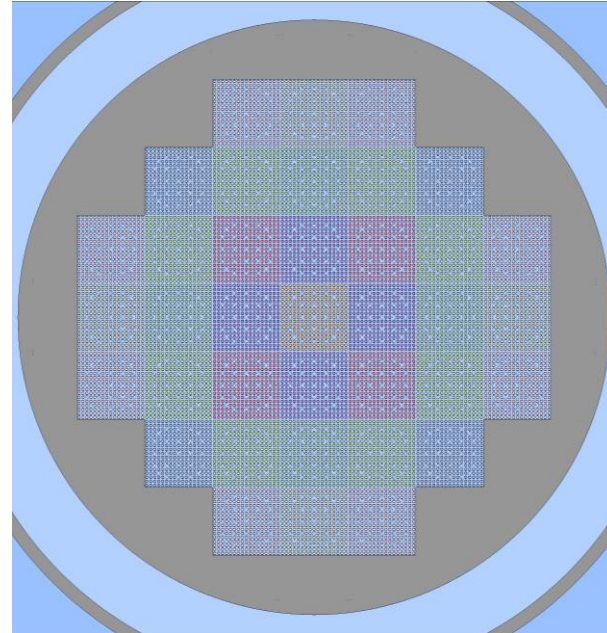
# Setting up coupled calculations with Cerberus

# Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

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



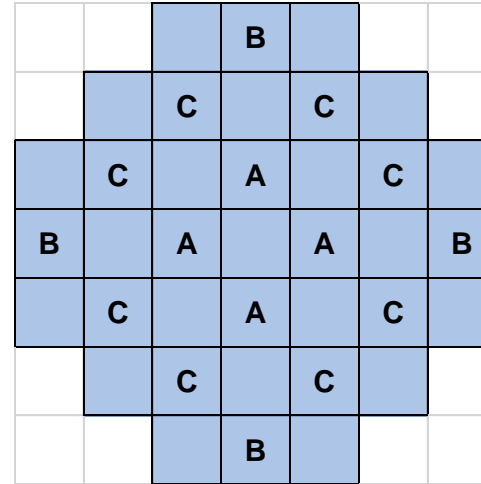


# Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

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



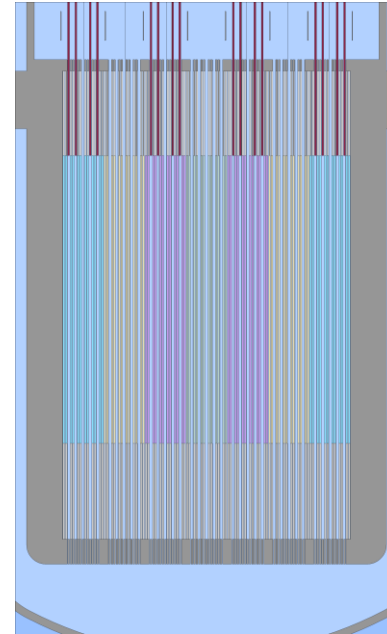
Control rod groups in the reactor core.

# Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

Valtavirta, V., Tuominen, R.

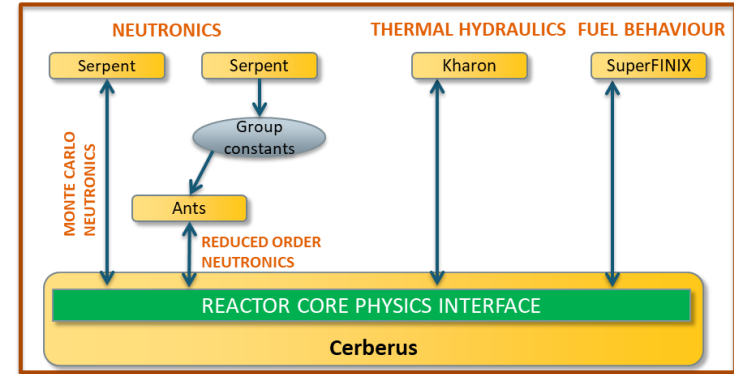
“A simple reactor core simulator based on VTT's Cerberus Python package”  
ANS M&C 2021, April 11-15, 2021, Raleigh, NC



# Applied calculation chains

## Calculation setup:

- Neutronics (**Serpent** OR **Ants**):
  - Continuous energy Monte Carlo **OR**
  - Eight group nodal diffusion with  $\frac{1}{4}$  assembly subnodalization.
- Thermal hydraulics (**Kharon**):
  - Porous medium closed channel single phase.
  - One channel per assembly.
- Fuel behaviour (**SuperFINIX**):
  - Traditional 1.5 dimensional approach.
  - One representative fuel rod per assembly.
- Coupled calculation:
  - Coupled fields exchanged at assembly level.

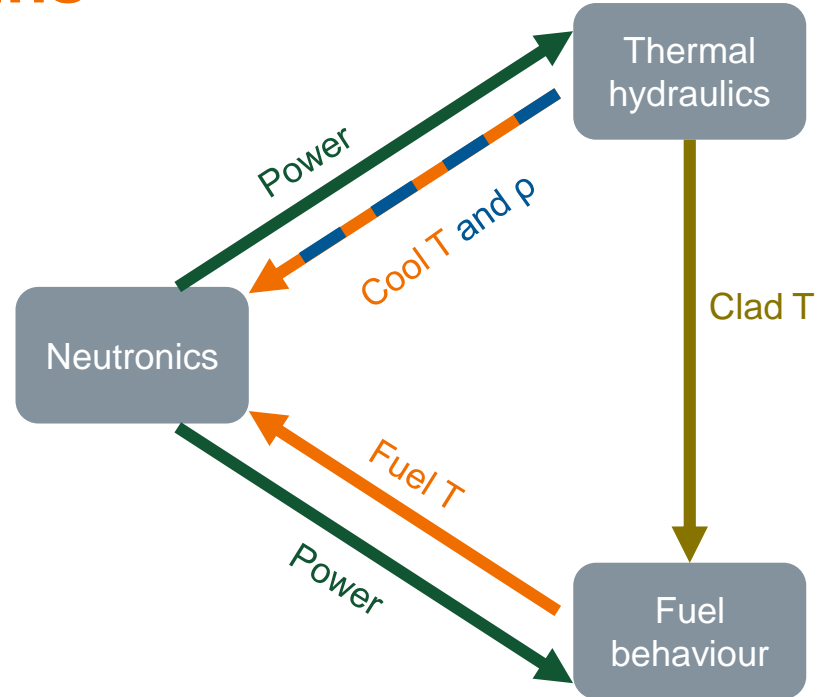


The Kraken based core simulator in this work used SuperFINIX for fuel behaviour, Kharon for thermal hydraulics and either Serpent or Ants for the neutronics.

# Applied calculation chains

## Calculation setup:

- Neutronics (**Serpent** OR **Ants**):
  - Continuous energy Monte Carlo **OR**
  - Eight group nodal diffusion with  $\frac{1}{4}$  assembly subnodalization.
- Thermal hydraulics (**Kharon**):
  - Porous medium closed channel single phase.
  - One channel per assembly.
- Fuel behaviour (**SuperFINIX**):
  - Traditional 1.5 dimensional approach.
  - One representative fuel rod per assembly.
- Coupled calculation:
  - Coupled fields exchanged at assembly level.



# Available inputs

- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
  - mappings/

# Available inputs

- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
    - Ants8g.inp
    - includes/
      - control\_rods.inc
      - core.inc
      - fuel\_assemblies.inc
      - radial\_reflector.inc
  - kharon/
  - serpent/
  - superfinix/
  - mappings/

# Available inputs

- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
  - kharon/
    - Kharon.inp
  - serpent/
  - superfinix/
  - mappings/

# Available inputs

- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
    - Serpent.inp
    - includes/
      - ...
  - superfinix/
  - mappings/



# Available inputs

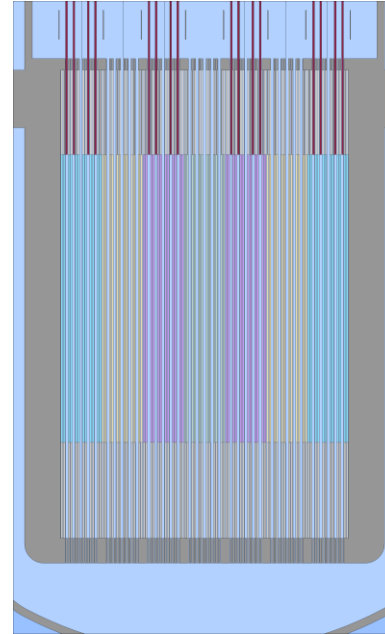
- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
    - SuperFINIX.inp
    - finixfiles/
      - finix.options
      - finix.rods
      - finix.scenario
  - mappings/

# Available inputs

- group\_constants/new8g\_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
  - mappings/
    - ants\_to\_kharon.txt
    - ants\_to\_sf.txt
    - kharon\_to\_ants.txt
    - kharon\_to\_sf.txt
    - sf\_to\_ants.txt
    - sf\_to\_neutronics.txt

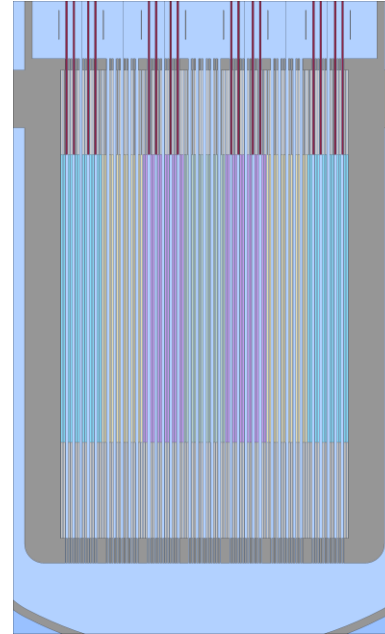
# First step: ARO HZP CBC

- All rods out (ARO) hot zero power (HZP) critical boron concentration.
- Initialize neutronics solver and set boron iteration on.
- Set HZP state.
- Converge neutronics solution.
- Get and store results.



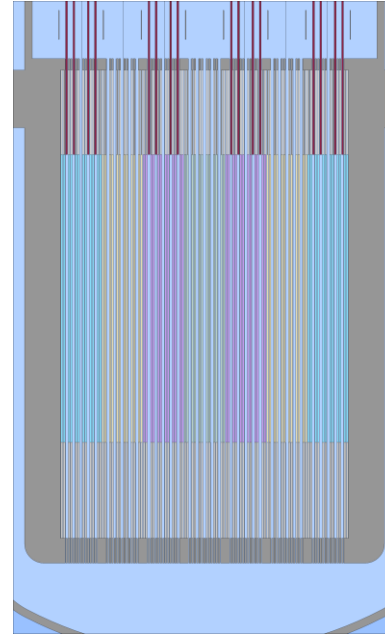
## Second step: ARO HFP CBC

- All rods out (ARO) hot full power (HFP) critical boron concentration.
- Initialize required solvers.
- Supply initial guess for fields.
- Converge coupled solution.
- Get and store results.



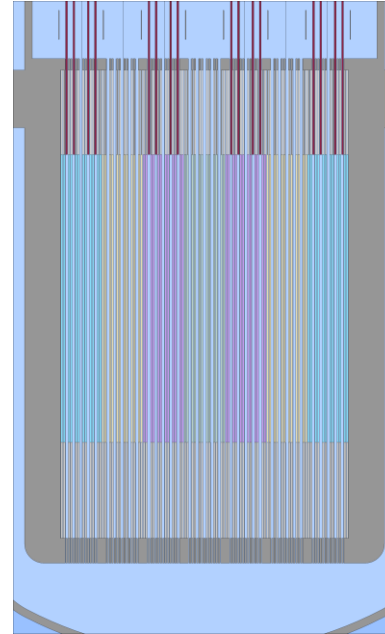
## Third step: ARO HFP CBC + control rod worths

- Initialize required solvers.
- Supply initial guess for fields.
- Converge coupled solution.
- Evaluate control rod worths.



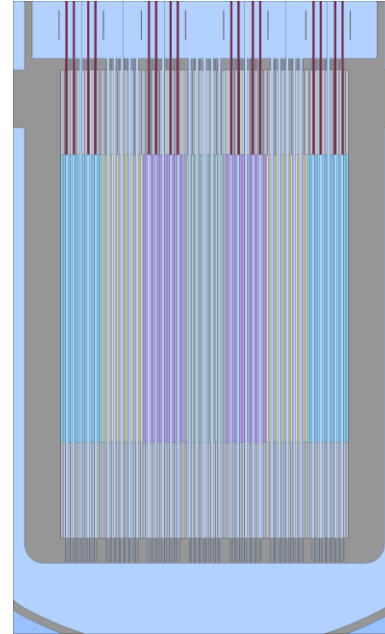
# Final step: Coupled burnup calculation with full power (constant extrapolation)

- Initialize required solvers.
- Supply initial guess for fields.
- For each time point:
  - Converge coupled solution.
  - Time integrate solution to next time point.



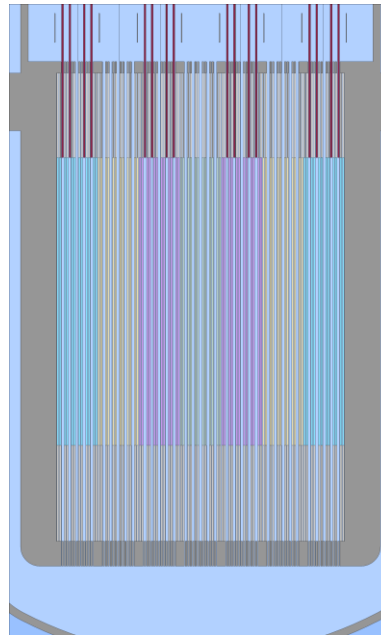
# What about Serpent?

- Serpent is still a Solver.
- Field and variable names are different:
  - Ants\_iv\_xenon\_state
  - sss\_iv\_fixed\_xenon
  - Ants\_ov\_boron
  - sss\_ov\_critical\_boron
- Internal workings of solvers are different.
- Meshing of fields may be different:
  - Ants meshing based on node structure.
  - Serpent meshing more freely chosen (multi-physics interface based).
- The same calculation can be repeated with Serpent (including the same variations).
  - See also next presentation.



# Summary

- Cerberus offers a high level Python API for solvers and their fields and variables in Kraken.
- Coupled calculation chains can be easily created in Python.
- More complex calculation chains quickly get tedious to write from scratch.
  - Common calculation sequences and sub-sequences can be packaged into their own modules.
- Next presentation on the **cerberus.simulator** module is one example.





# bey<sup>0</sup>nd

## the obvious

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

@VTTFinland

[www.vtt.fi](http://www.vtt.fi)