# Nuclear Fuel Performance

NE-533

Spring 2025

# MOOSE– Massively Object-Oriented Simulation Environment

the multiphysics framework

https://mooseframework.inl.gov/

# Access to MOOSE

- Everyone will have access to the RDFMG cluster
- Can utilize moose on your local machine, or on cluster
- If off campus, need to get the NCSU VPN
  - https://oit.ncsu.edu/campus-it/campus-data-network/vpn/
- Need some kind of terminal
  - linux/mac is native, for windows need something like PuTTY, Cygwin, MobaXterm, etc.
- ssh uname@rdfmg.ne.ncsu.edu

# Building MOOSE on RDFMG cluster

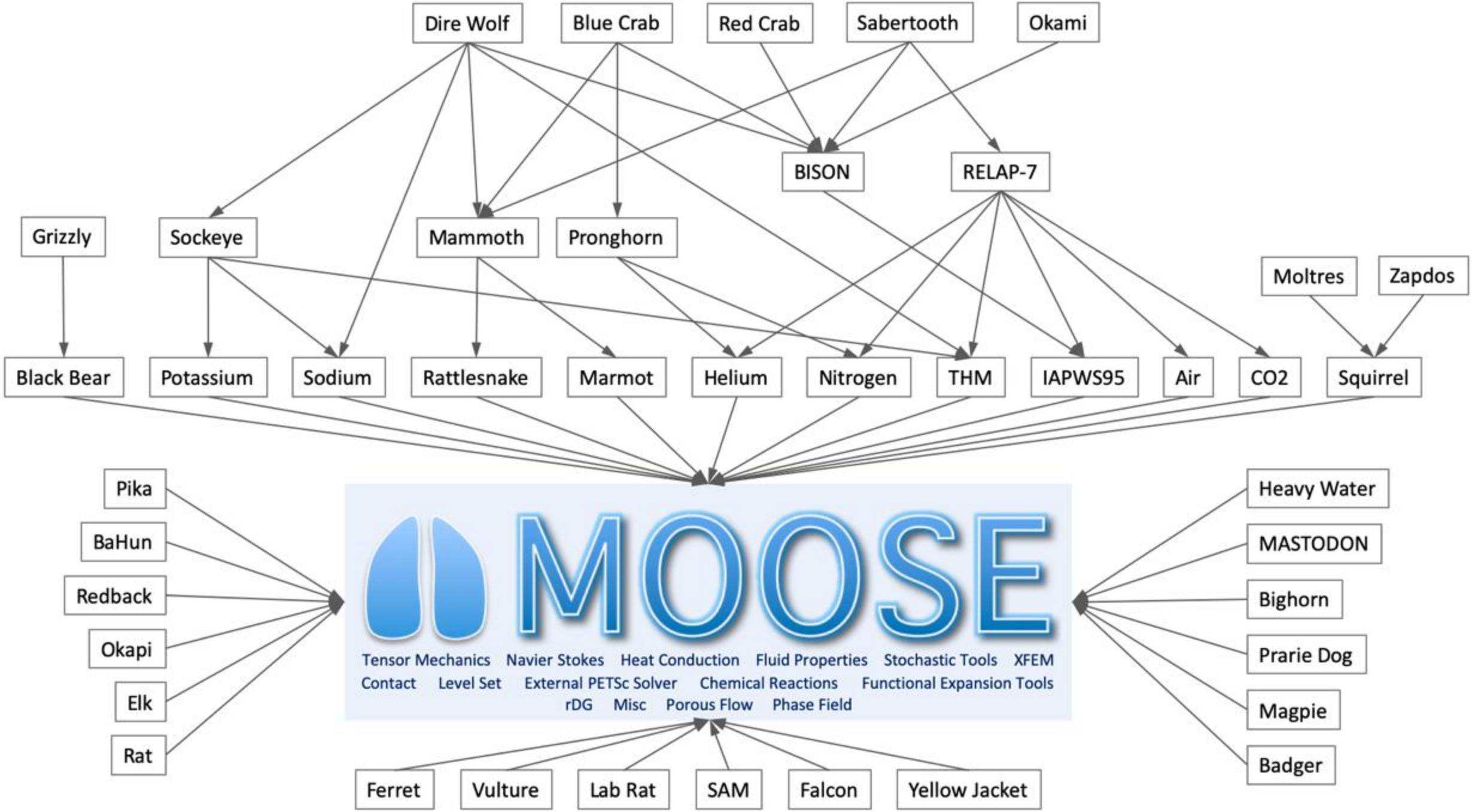
- Follow the instructions on the MOOSE website
  - https://mooseframework.inl.gov/getting_started/installation/conda.html
- Will install conda, create moose environment, git pull moose, build and test
- Then you will create your own moose app:
  - https://mooseframework.inl.gov/getting_started/new_users.html
- I recommend making a directory with your input files inside your new app folder
- Edit makefile in new app for modules you will need
  - heat transfer and solid mechanics should be all you will need for the project

# Handy bash commands

- ssh username@rdfmg.ne.ncsu.edu : allows you to get on the cluster
- sftp username@rdfmg.ne.ncsu.edu : allows you to transfer files to/from cluster
- ls : list files in a directory; pwd: print working directory; cd : change directory; cp : copy a file
- file editors: vi, vim, emacs, nano, atom
- You should not need a submission script to run on rdfmg (don't need that many cores), but I recommend running on an interactive node
- module load codes/workbench
- ***isqp -w node026***
- https://ne.ncsu.edu/rdfmg/rdfmg-guide/

# MOOSE Resources

- https://mooseframework.inl.gov/
- Under getting started tab, have workshop slides, workshop video, and other tutorials
- There is a lot here, work through examples, understand what the examples are doing
- The sooner you dive deep into MOOSE, the easier all of this will be


Dire Wolf
Blue Crab
Red Crab
Sabertooth
Okami
BISON
RELAP-7
Grizzly
Sockeye
Mammoth
Pronghorn
Moltres
Zapdos
Black Bear
Potassium
Sodium
Rattlesnake
Marmot
Helium
Nitrogen
THM
IAPWS95
Air
CO2
Squirrel
Pika
BaHun
Redback
Okapi
Elk
Rat
MOOSE
Tensor Mechanics
Navier Stokes
Heat Conduction
Fluid Properties
Stochastic Tools
XFEM
Contact
Level Set
External PETSc Solver
Chemical Reactions
Functional Expansion Tools
rDG
Misc
Porous Flow
Phase Field
Heavy Water
MASTODON
Bighorn
Prarie Dog
Magpie
Badger
Ferret
Vulture
Lab Rat
SAM
Falcon
Yellow Jacket

# MOOSE Physics Modules

Chemical Reactions
Contact
External PETSc Solver
Fluid Properties
Function Expansion Tools
Heat Conduction
Level Set
Navier Stokes
Phase Field
Porous Flow
rDG
Stochastic Tools
Tensor (solid) Mechanics
XFEM
Shallow Water (work in progress)
Ray Tracing (work in progress)

# Governing Equations

Conservation of Mass:

$$\nabla \cdot \bar{u} = 0 \tag{1}$$

Conservation of Energy:

$$C\left(\frac{\partial T}{\partial t} + \epsilon \bar{u} \cdot \nabla T\right) - \nabla \cdot k \nabla T = 0 \tag{2}$$

Darcy's Law:

$$\bar{u} = -\frac{\mathbf{K}}{\mu}(\nabla p - \rho \bar{g}) \tag{3}$$

where $\bar{u}$ is the fluid velocity, $\epsilon$ is porosity, $\mathbf{K}$ is the permeability tensor, $\mu$ is fluid viscosity, $p$ is the pressure, $\rho$ is the density, $\bar{g}$ is the gravity vector, and $T$ is the temperature.

# Input Files

All capabilities of MOOSE, modules, and your application are compiled into a single executable. An input file is used define which capabilities are used to perform a simulation.

MOOSE uses the "hierarchical input text" (hit) format.

```
[Kernels]
  [diffusion]
    type = ADDiffusion # Laplacian operator using automatic differentiation
    variable = pressure # Operate on the "pressure" variable from above
  []
[]
```

# Input File

A basic MOOSE input file requires six parts, each of which will be covered in greater detail later.

- `[Mesh]`: Define the geometry of the domain
- `[Variables]`: Define the unknown(s) of the problem
- `[Kernels]`: Define the equation(s) to solve
- `[BCs]`: Define the boundary condition(s) of the problem
- `[Executioner]`: Define how the problem will solve
- `[Outputs]`: Define how the solution will be written

```
[Mesh]
  type = GeneratedMesh # Can generate simple lines, rectangles and rectangular prisms
  dim = 2              # Dimension of the mesh
  nx = 100             # Number of elements in the x direction
  ny = 10              # Number of elements in the y direction
  xmax = 0.304         # Length of test chamber
  ymax = 0.0257        # Test chamber radius
[]

[Variables]
  [pressure]
    # Adds a Linear Lagrange variable by default
  []
[]

[Kernels]
  [diffusion]
    type = ADDiffusion  # Laplacian operator using automatic differentiation
    variable = pressure # Operate on the "pressure" variable from above
  []
[]

[BCs]
  [inlet]
    type = DirichletBC  # Simple u=value BC
    variable = pressure # Variable to be set
```

# Run Example via Command Line

An executable is produced by compiling an application or a MOOSE module. It can be used to run input files.

```
cd ~/projects/moose/tutorials/darcy-thermo_mech/step01_diffusion
make -j 12 # use number of processors for you system
cd problems
../darcy_thermo_mech-opt -i step1.i
```

# Mesh

- Meshes in MOOSE are built or loaded using MeshGenerators
- To only generate the mesh without running the simulation, you can pass --mesh-only on the command line
- FileMeshGenerator is the MeshGenerator to load external meshes
- Built-in mesh generation is implemented for lines, rectangles, rectangular prisms or extruded reactor geometries
- The sides are named in a logical way and are numbered :
  - 1D: left=0, right=1
  - 2D: bottom=0, right=1, top=2, left=3
  - wont need more than 2D here
- Human-readable names can be assigned to blocks, sidesets, and nodesets that can be used throughout an input file.
- A parameter that requires an ID will accept either numbers or "names".
- Names can be assigned to IDs for existing meshes to ease input file maintenance.

```
[Mesh]
  [fmg]
    type = FileMeshGenerator
    file = square.e
  []
[]
```

```
[Mesh]
  [generated]
    type = GeneratedMeshGenerator
    dim = 2
    xmin = 0
    xmax = 1
    nx = 2
    ymin = -2
    ymax = 3
    ny = 3
    elem_type = 'TRI3'
  []
```

# Variables

The `Variables` block within an input file is utilized to define the unknowns within a system of partial differential equations. These unknowns are often referred to as nonlinear variables within documentation. The nonlinear variables defined within this block are used by Kernel objects to define the equations for a simulation.

```
[Variables]
  [convected]
    order = FIRST
    family = LAGRANGE
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = convected
  []

  [conv]
    type = ExampleConvection
    variable = convected
    velocity = '0.0 0.0 1.0'
  []
[]
```

# Kernel

A "Kernel" is a piece of physics. It can represent one or more operators or terms in the weak form of a partial differential equation. With all terms on the left-hand-side, their sum is referred to as the "residual". The residual is evaluated at several integration quadrature points over the problem domain. To implement your own physics in MOOSE, you create your own kernel by subclassing the MOOSE `Kernel` class.

`ADHeatConduction` is the implementation of the heat diffusion equation in HeatConduction within the framework of automatic differentiation. The `ADHeatConduction` kernel implements the heat equation given by Fourier's Law where The heat flux is given as

$$\mathbf{q} = -k\nabla T,$$

where $k$ denotes the thermal conductivity of the material. $k$ can either be an `ADMaterial` or traditional `Material`.

# Materials

The material system operates by creating a producer/consumer relationship among objects

- `Material` objects **produce** properties.
- Other MOOSE objects (including materials) **consume** these properties.

```
[Materials]
  [block_1]
    type = OutputTestMaterial
    block = 1
    output_properties = 'real_property tensor_property'
    outputs = exodus
    variable = u
  []
  [block_2]
    type = OutputTestMaterial
    block = 2
    output_properties = 'vector_property tensor_property'
    outputs = exodus
    variable = u
  []
[]

[Outputs]
  exodus = true
[]
```

# GenericConstantMaterials

GenericConstantMaterial is a simple way to define constant material properties.

Two input parameters are provided using "list" syntax common to MOOSE:

```
prop_names  = 'conductivity density'
prop_values = '0.01         200'
```

# Boundary Conditions

$$
\begin{aligned}
-\nabla^2 u &= f && \in \Omega \\
u &= g && \in \partial\Omega_D \\
\frac{\partial u}{\partial n} &= h && \in \partial\Omega_N,
\end{aligned}
$$

## DirichletBC

Imposes the essential boundary condition $u = g$, where $g$ is a constant, controllable value.

## NeumannBC

Imposes the integrated boundary condition $\frac{\partial u}{\partial n} = h$, where $h$ is a constant, controllable value.

# Functions

- Many objects exist in MOOSE that utilize a function, such as:
  - FunctionDirichletBC, FunctionNeumannBC, FunctionIC, BodyForce
- Each of these objects has a "function" parameter which is set in the input file, and controls which Function object is used
- A ParsedFunction allows functions to be defined by strings directly in the input file

```
[Functions]
  [sin_fn]
    type = ParsedFunction
    expression = sin(x)
  []
  [cos_fn]
    type = ParsedFunction
    expression = cos(x)
  []
```

# MOOSE Executioner

Steady-state executioners generally solve the nonlinear system just once.

```
[Executioner]
  type = Steady
[]
```

(test/tests/executioners/steady_time/steady_time.i)

The Steady executioner can solve the nonlinear system multiple times while adaptively refining the mesh to improve the solution.

There are a number of options that appear in the executioner block and are used to control the solver. Here are a few common options:

| Option | Definition |
|---|---|
| `l_tol` | Linear Tolerance (default: 1e-5) |
| `l_max_its` | Max Linear Iterations (default: 10000) |
| `nl_rel_tol` | Nonlinear Relative Tolerance (default: 1e-8) |
| `nl_abs_tol` | Nonlinear Absolute Tolerance (default: 1e-50) |
| `nl_max_its` | Max Nonlinear Iterations (default: 50) |

Transient executioners solve the nonlinear system at least once per time step.

| Option | Definition |
|---|---|
| `dt` | Starting time step size |
| `num_steps` | Number of time steps |
| `start_time` | The start time of the simulation |
| `end_time` | The end time of the simulation |
| `scheme` | Time integration scheme (discussed next) |

```
[Executioner]
  type = Transient
  scheme = 'implicit-euler'
  solve_type = 'PJFNK'
  start_time = 0.0
  num_steps = 5
  dt = 0.1
[]
```

| Option | Definition |
|---|---|
| `steady_state_detection` | Whether to try and detect achievement of steady-state (Default = `false`) |
| `steady_state_tolerance` | Used for determining a steady-state; Compared against the difference in solution vectors between current and old time steps (Default = 1e−8) |

# Outputs

The output system is designed to be just like any other system in MOOSE: modular and expandable.

It is possible to create multiple output objects for outputting:

- at specific time or timestep intervals,
- custom subsets of variables, and
- to various file types.

There exists a short-cut syntax for common output types as well as common parameters.

```
[Outputs]
  interval = 10
  exodus = true
  [all]
    type = Exodus
    interval = 1 # overrides interval from top-level
  []
[]
```

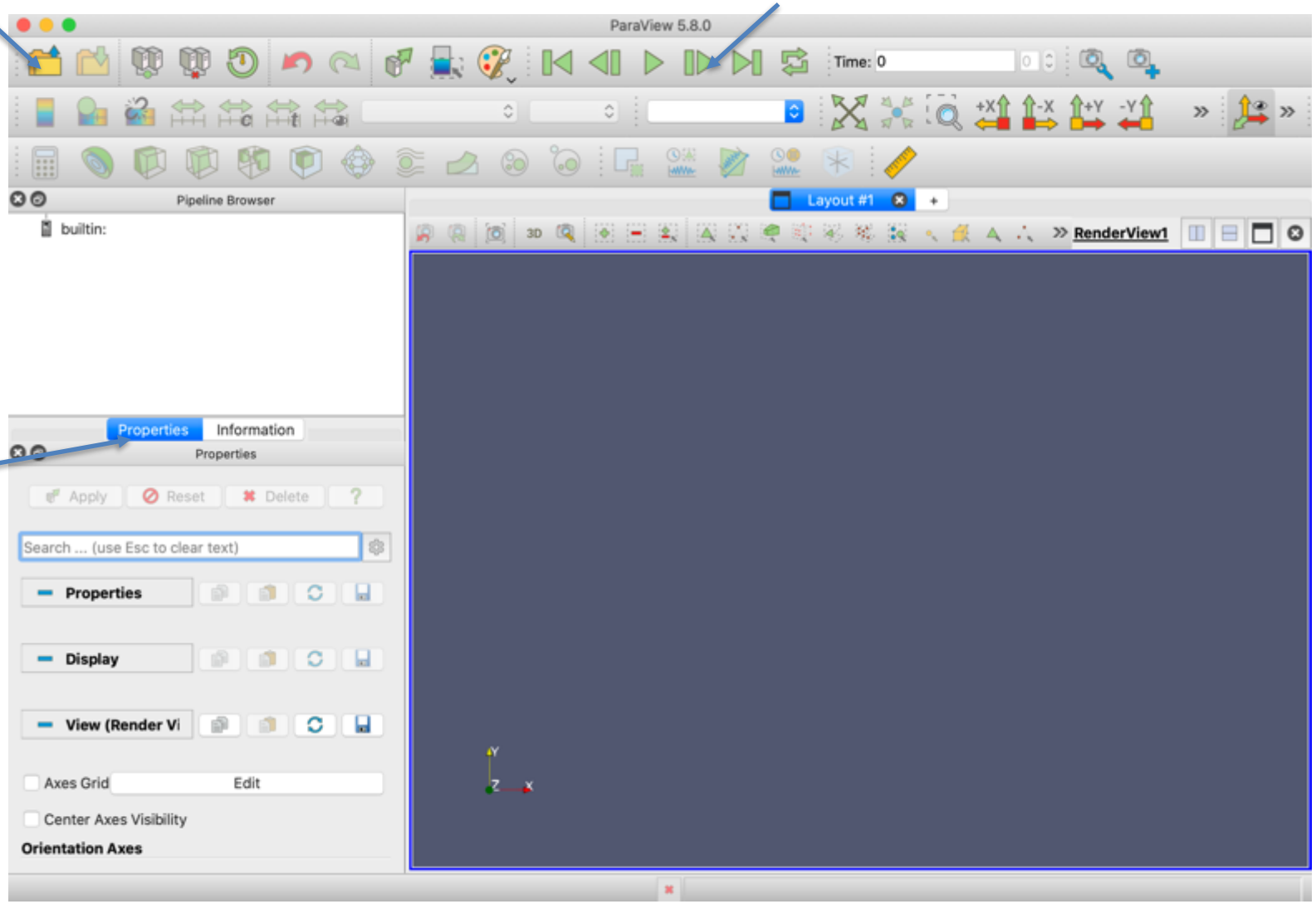
# Visualization

- Two ways to visualize
  - Peacock: MOOSE GUI
  - Paraview: External software program that can read exodus files
- I recommend downloading paraview
  - It is free and open source
  - Very flexible
  - Less buggy than peacock
- But, if you try peacock, and it works for you, great!

# Paraview

Open new

Move forward in time




Select Properties

# MOOSE C++ Fundamentals

- Will not cover here, if you feel like you need some background, go to below site, where they have some basics

- https://mooseframework.org/workshop/#/10

# Tutorials/Examples

- A stepped example/tutorial lives in:
  - moose/tutorials/darcy_thermo_mech
- This tutorial has 11 steps where they add complexity to a moose input file/physics problem
- This is a VERY valuable exercise to work through
- Tip : ./mooseapp-opt --dump > syntax.out creates a file “syntax.out” with all of the moose syntax and usage
- Better tip: mooseframework.org is a great resource with everything documented