Serac

User-Friendly Abstractions for MFEM-Based engineering applications

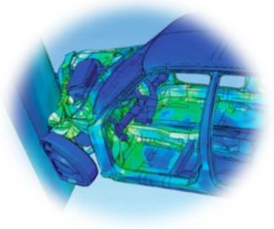
Jamie Bramwell

October 20, 2021

LLNL-PRES-828051

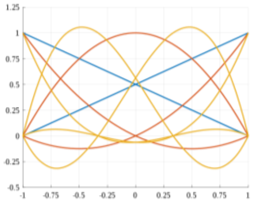
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Who uses high fidelity engineering simulation analysis codes?

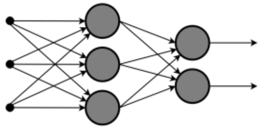


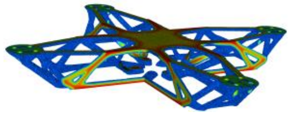
Engineering Analyst
Physics and material models, coupling schemes, analysis workflows

Software Engineer
Hardware portability, data management, code health, integration concerns



Computational Scientist
Discretization schemes, numerical linear algebra, high order and scalable algorithms





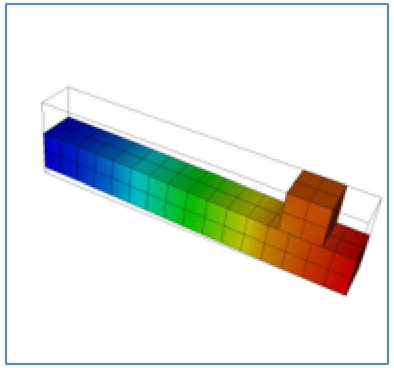
Data Scientist/Design Optimization
Access to raw data, well defined APIs, ensemble workflows

We need to address the needs of our wide user base!

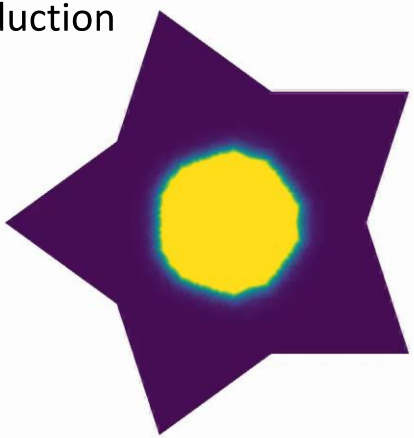
Serac: a modern tool kit for flexible engineering simulation code development at scale

Plug-and-play MFEM-based components for emerging applications

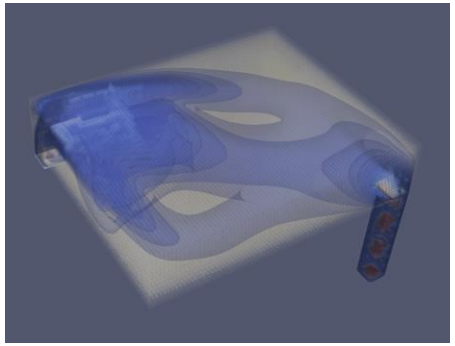
- *Modular physics solvers*: nonlinear heat conduction, finite deformation solid mechanics
- *Modern simulation workflows*: easy data access for optimization, reduced order modeling, machine learning, and UQ
- *Rapid development*: new multiphysics capabilities in weeks instead of years
- *Software quality:* modern software standards (using C++17) shortening time from research to production



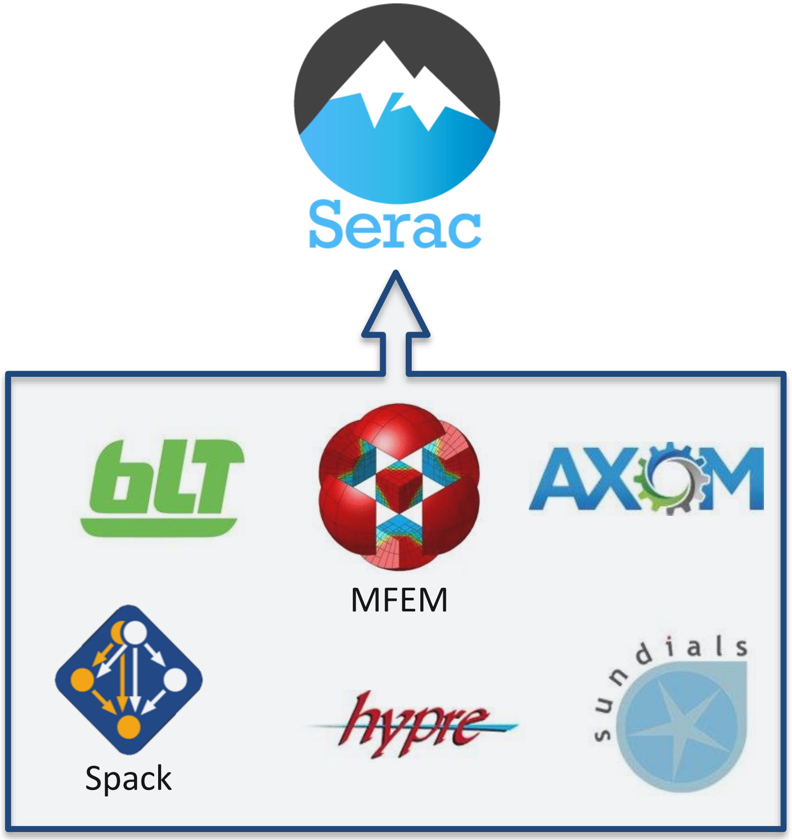
Finite deformation implicit mortar contact



Fully parallel high order implicit heat conduction in 10 simple lines of C++



Redox flow battery



Leverage and improve LLNL institutional HPC software

Don’t reinvent the wheel for each new code effort!

We are employing a two-repo strategy for agile and transparent code development

- Serac
  - https://github.com/LLNL/serac
  - Open source under BSD 3-clause
  - All computer science infrastructure for engineering-focused multiphysics simulations
  - Simple nonlinear thermal-structural mechanics development permitted
  - Source-generated documentation available at https://serac.readthedocs.io/en/latest/

- Smith
  - https://lc.llnl.gov/gitlab/smith/smith (All LLNL CZ users have read access)
  - Leverages infrastructure provided by Serac
  - Extra physics not allowed in our current open-source agreement
    - Mortar-based contact mechanics (serial statics only)
    - DG advection-diffusion-reaction
    - Steady state incompressible Navier-Stokes
    - Electromagnetics (quasi-electrostatics, magnetic diffusion, time domain full wave)
    - Helmholtz filter
  - Documentation on LC at https://lc.llnl.gov/smith/

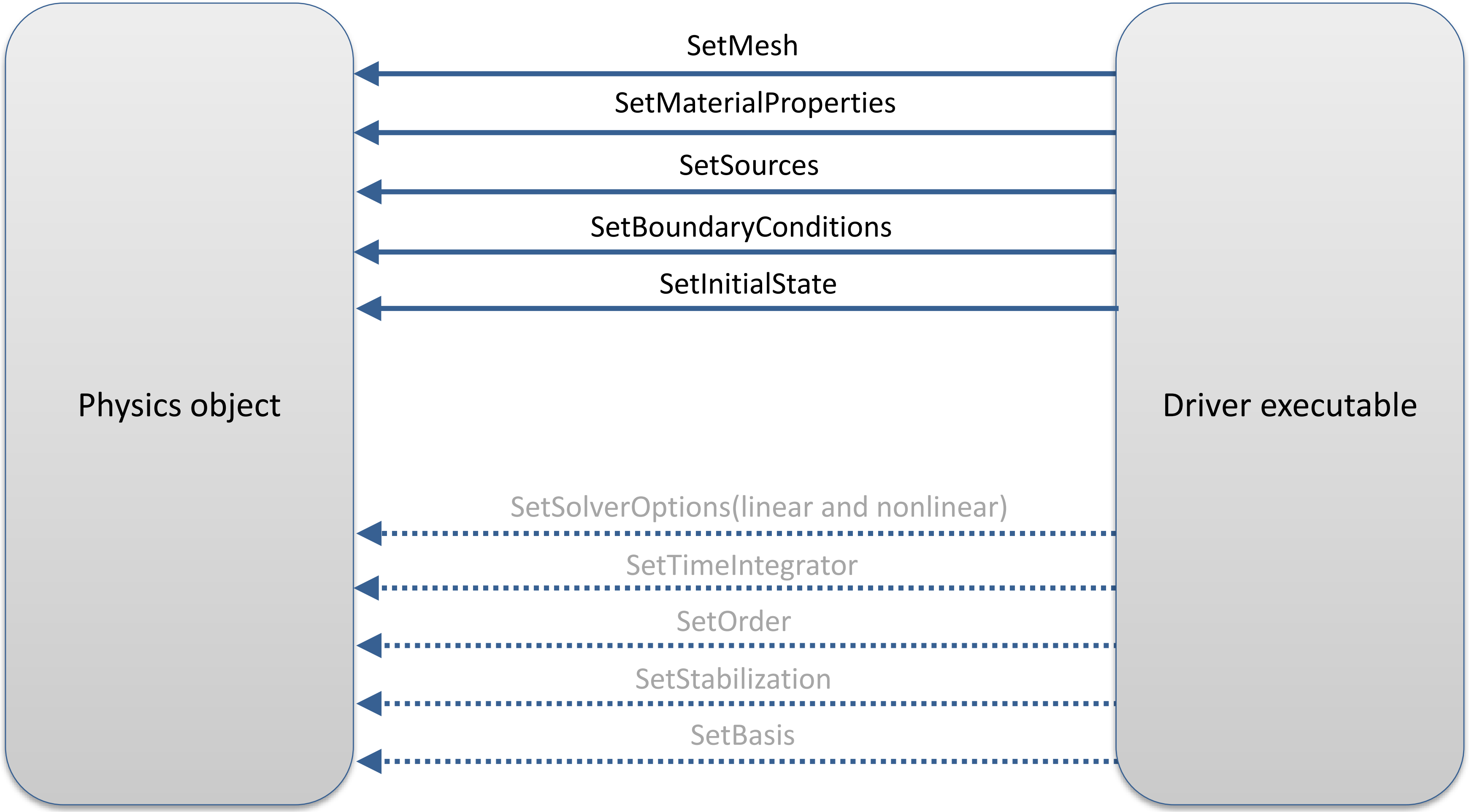
Serac

Smith

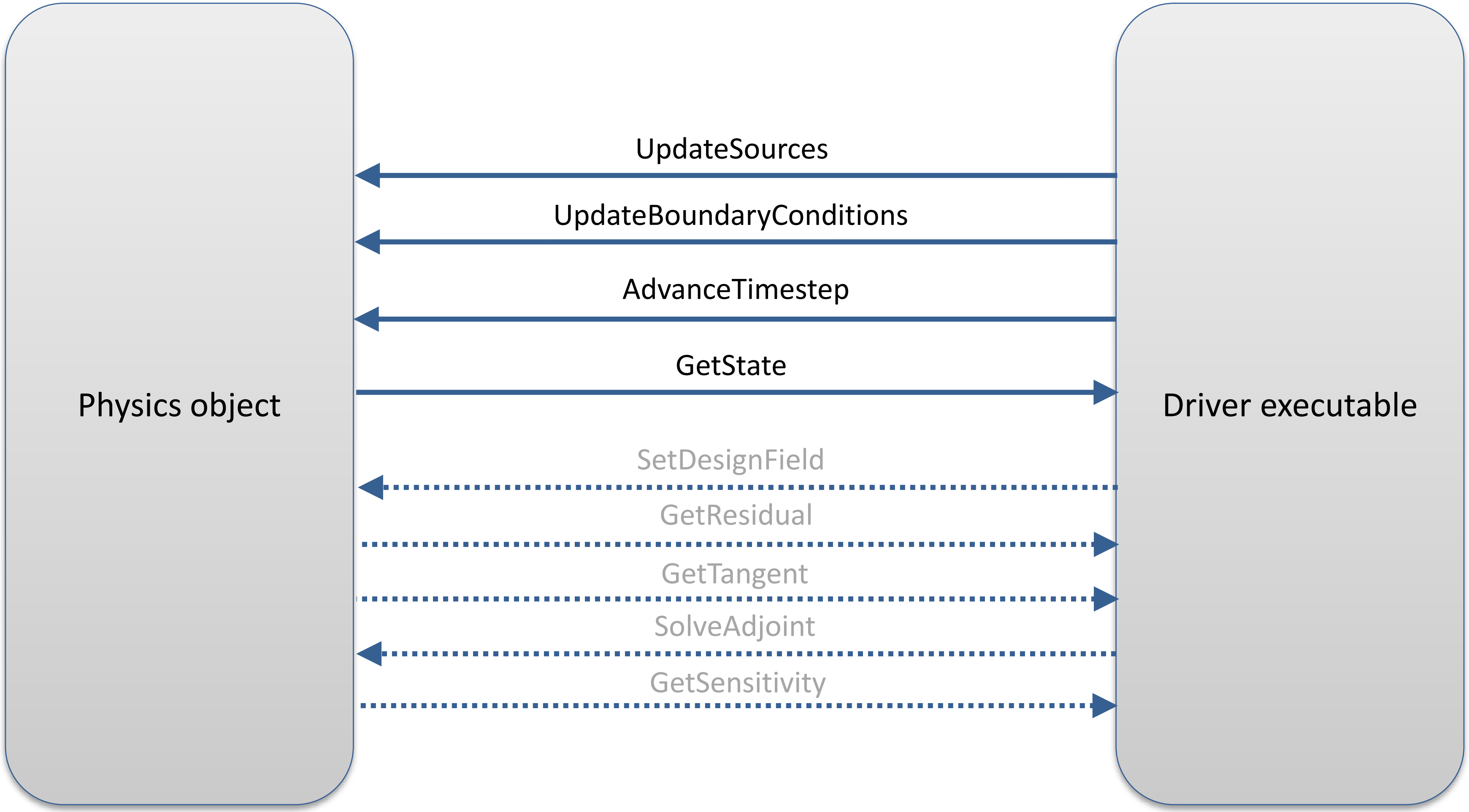
A key Serac component is the Physics object

- Defined interface for a generic forward PDE solver
  - Abstract away finite element spaces, stabilization issues, integrator definitions, boundary condition application, operator and memory management, etc.
- Can use physics-specific words in API
  - Make MFEM codes readable by engineers
- Can use common tools between physics modules
  - State manager for restart capability
  - Wrapper for primal and dual vectors
  - Residual-based operator definitions
  - Simple input file definitions
- Can be viewed as specialized single-physics simulation bundles
  - Think common language for MFEM examples
  - Not necessarily wrapping an MFEM-based code
  - Multiphysics by composition is possible

Physics setup phase overview



Run phase overview



A concrete example – transient thermal conduction in C++

https://github.com/LLNL/serac/blob/develop/examples/simple_conduction/without_input_file.cpp

Flexible nonlinear forms via Functional*

**name subject to change*

- Developed by Julian Andrej and Sam Mish
- Simple, intuitive interface for defining nonlinear finite element kernels
  - Uses CEED QFunction source and flux definition
- Forward mode auto differentiation for determining gradients
- Easy to implement complex material models
- Uses statically sized tensor class with dual numbers
- GPU-enabled functionality without the need for a partial assembly integrator
  - However, will not get the same performance

$$\mathbf{r}(\mathbf{u}) = \int_\Omega \nabla\psi \cdot \sigma\left(\nabla \mathbf{u}\right)\, dv$$

$$\Downarrow$$

```
Functional<test_space(trial_space)> residual(&fespace, &fespace);

residual.AddDomainIntegral(
    Dimension<dim>{},
    [=](auto /*x*/, auto displacement) {
      auto [u, du_dx] = displacement;
      auto strain     = 0.5 * (du_dx + transpose(du_dx));
      auto stress     = b * tr(strain) * I + 2.0 * b * strain;
      return serac::tuple{zero{}, stress};
    },
    mesh);

mfem::Vector res = residual(current_state);
mfem::Operator& grad = residual.GetGradient(current_state);
```

Linear elasticity via functional

MFEM ex1 via Functional

https://github.com/LLNL/serac/blob/feature/bramwell%2Ffunctional_ex1/src/serac/physics/utilities/functional/tests/functional_mfem_ex1.cpp

Lawrence Livermore National Laboratory