SAND2021-11779C

Exceptional service in the national interest

Sandia National Laboratories

Projection-based reduced order models for large-scale multiphysics applications

September 28, 2021

John Tencer

Patrick Blonigan, Eric Parish, Francesco Rizzi

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

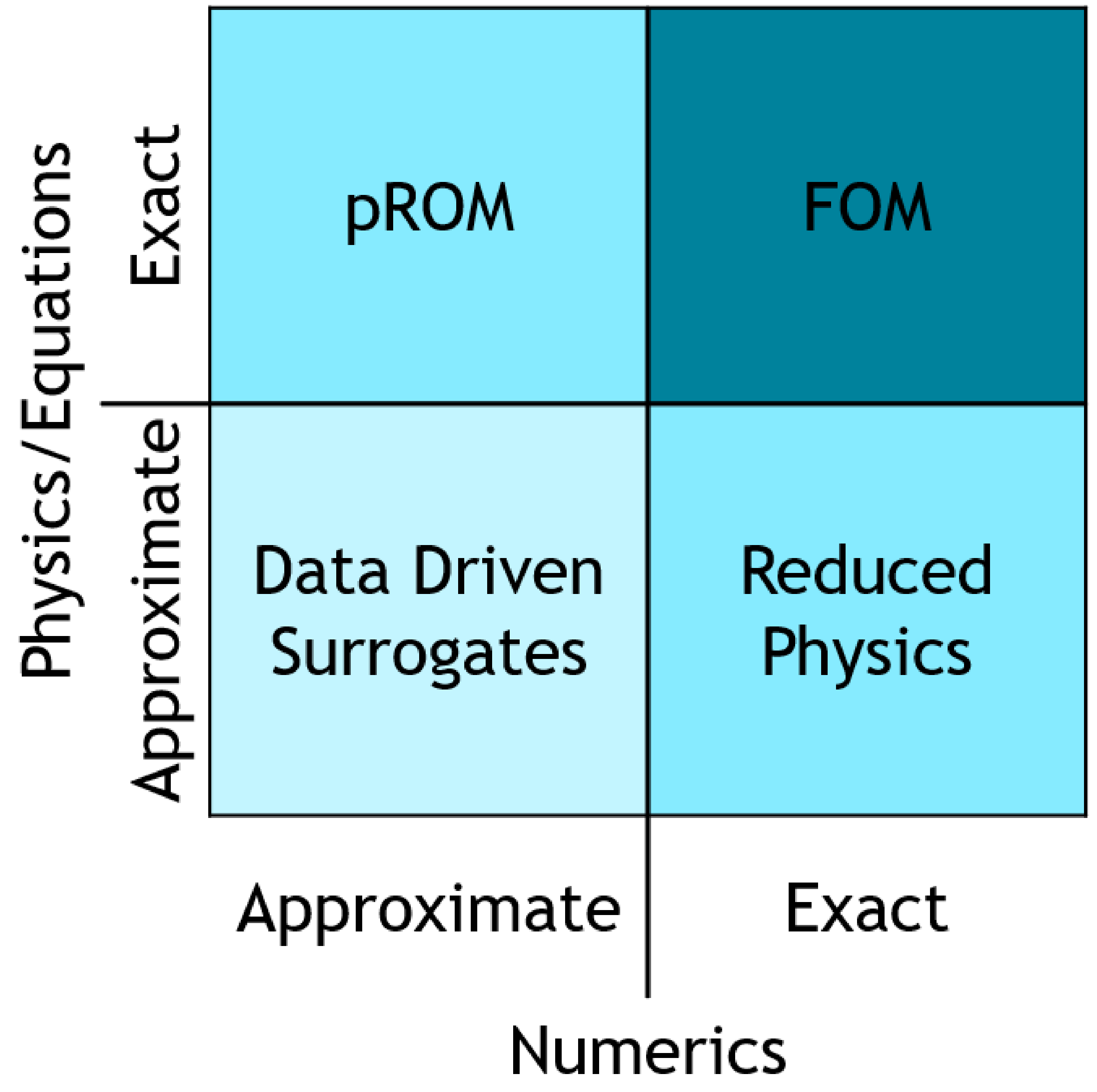
What are projection-based reduced order models?

(pROMs)

- Fast-running surrogate models which achieve their speed-ups by providing approximate solutions to the exact governing equations.

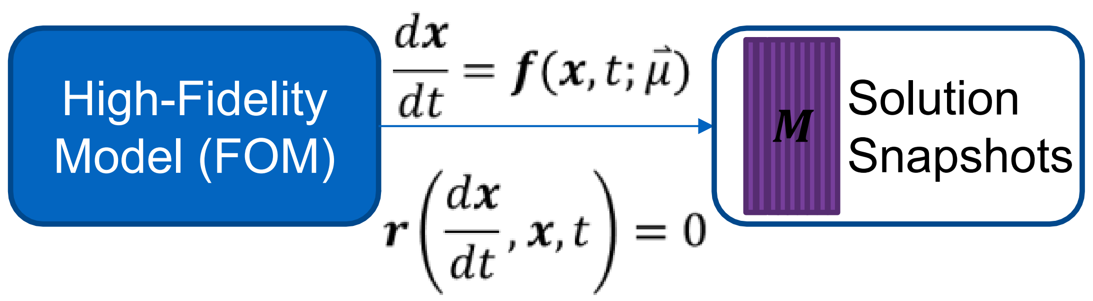
+ Leverages existing simulation software
+ Compatible with a priori and a posteriori error bounds
+ Full-field predictions
+ Less training data required

— Intrusive
— Slower than “black box” methods



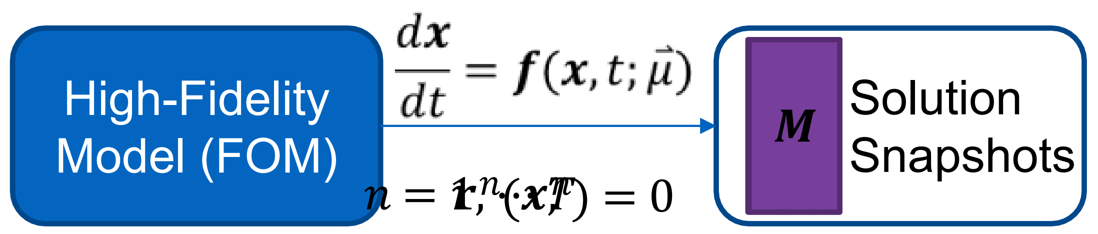
pROM Workflow Overview

Generate Solution Snapshots

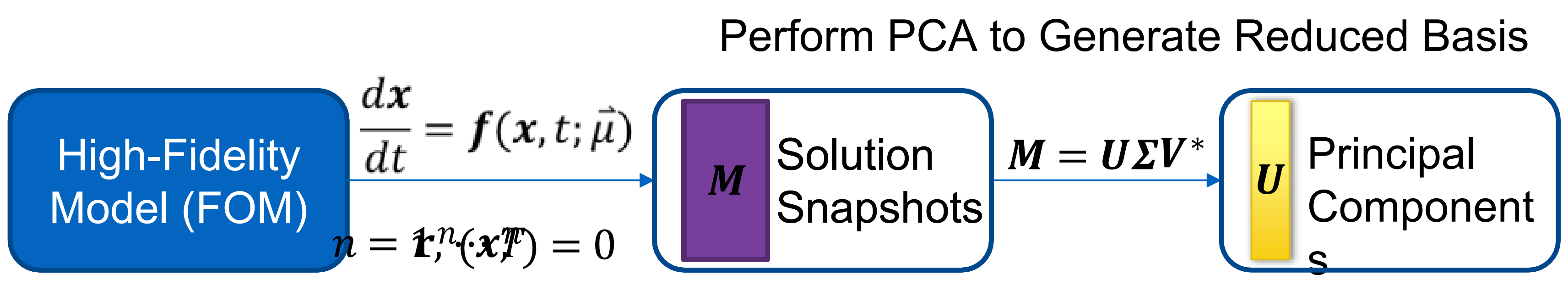

High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$\boldsymbol{r}\left(\frac{d\boldsymbol{x}}{dt}, \boldsymbol{x}, t\right) = 0$
$\boldsymbol{M}$
Solution Snapshots

pROM Workflow Overview

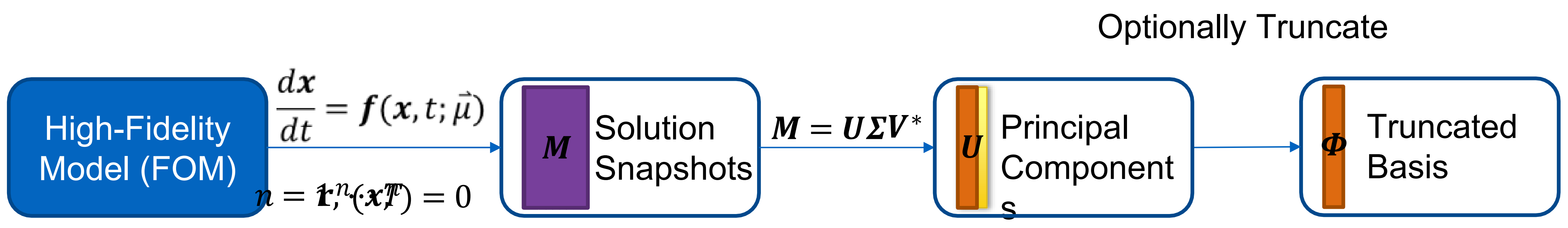
Generate Solution Snapshots


High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$n = \mathbf{r}^n(\boldsymbol{x}^n) = 0, \; 1, \cdots, T$
$\boldsymbol{M}$
Solution Snapshots

pROM Workflow Overview

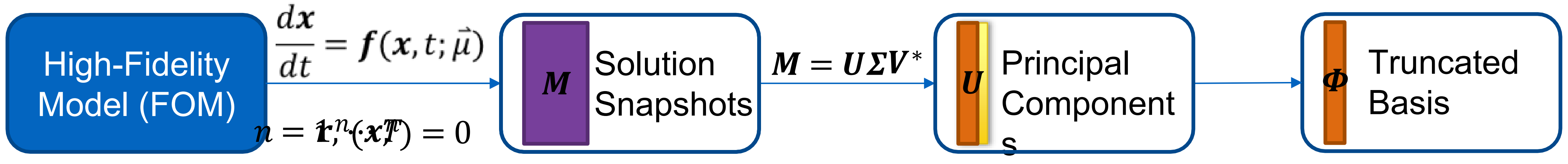

Perform PCA to Generate Reduced Basis
High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$n = 1, \cdots, T$
$\boldsymbol{r}^n(\boldsymbol{x}^n) = 0$
$\boldsymbol{M}$
Solution Snapshots
$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$
$\boldsymbol{U}$
Principal Components

pROM Workflow Overview



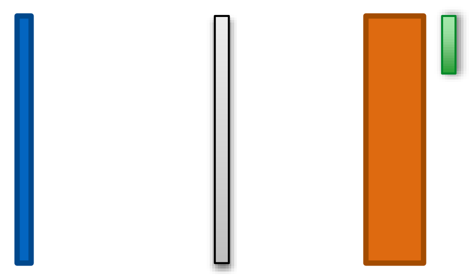
Optionally Truncate
High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$n = 1, \cdots, T$ $\boldsymbol{r}^n(\boldsymbol{x}) = 0$
$\boldsymbol{M}$
Solution Snapshots
$\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$
$\boldsymbol{U}$
Principal Components
$\boldsymbol{\Phi}$
Truncated Basis

pROM Workflow Overview

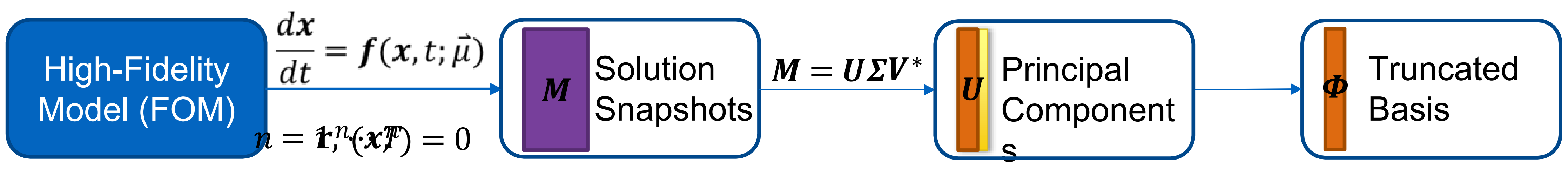

High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$n = \mathbf{r}^{n}(\boldsymbol{x}^{n}) = 0$
$\boldsymbol{M}$
Solution Snapshots
$\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$
$\boldsymbol{U}$
Principal Components
$\boldsymbol{\Phi}$
Truncated Basis

Approximate FOM State

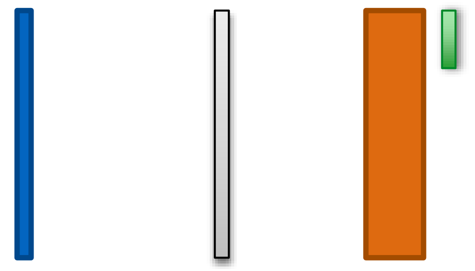
$\boldsymbol{x}(t; \vec{\mu}) \approx \tilde{\boldsymbol{x}}(t; \vec{\mu}) = \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t; \vec{\mu})$



pROM Workflow Overview


High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$\boldsymbol{r}^n(\boldsymbol{x}^n) = 0$
$n = 1, \cdots, T$
$\boldsymbol{M}$
Solution Snapshots
$\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$
$\boldsymbol{U}$
Principal Components
$\boldsymbol{\Phi}$
Truncated Basis

$\boldsymbol{x}(t;\vec{\mu}) \approx \tilde{\boldsymbol{x}}(t;\vec{\mu}) = \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t;\vec{\mu})$



Project System Dynamics

Galerkin Projection

$$\boldsymbol{\Phi}^T\boldsymbol{r}^n(\boldsymbol{\Phi}\hat{\boldsymbol{x}}^n) = 0$$
$$n = 1, \cdots, T$$

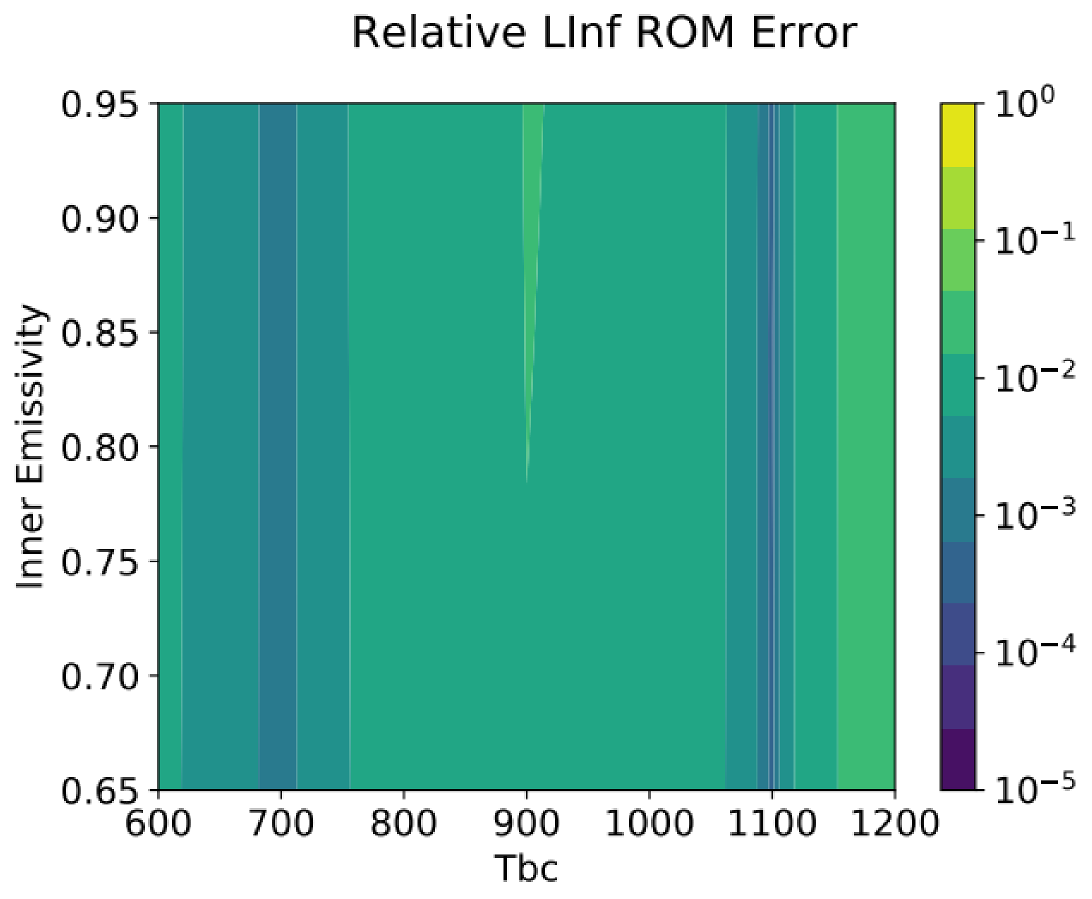
LSPG Projection

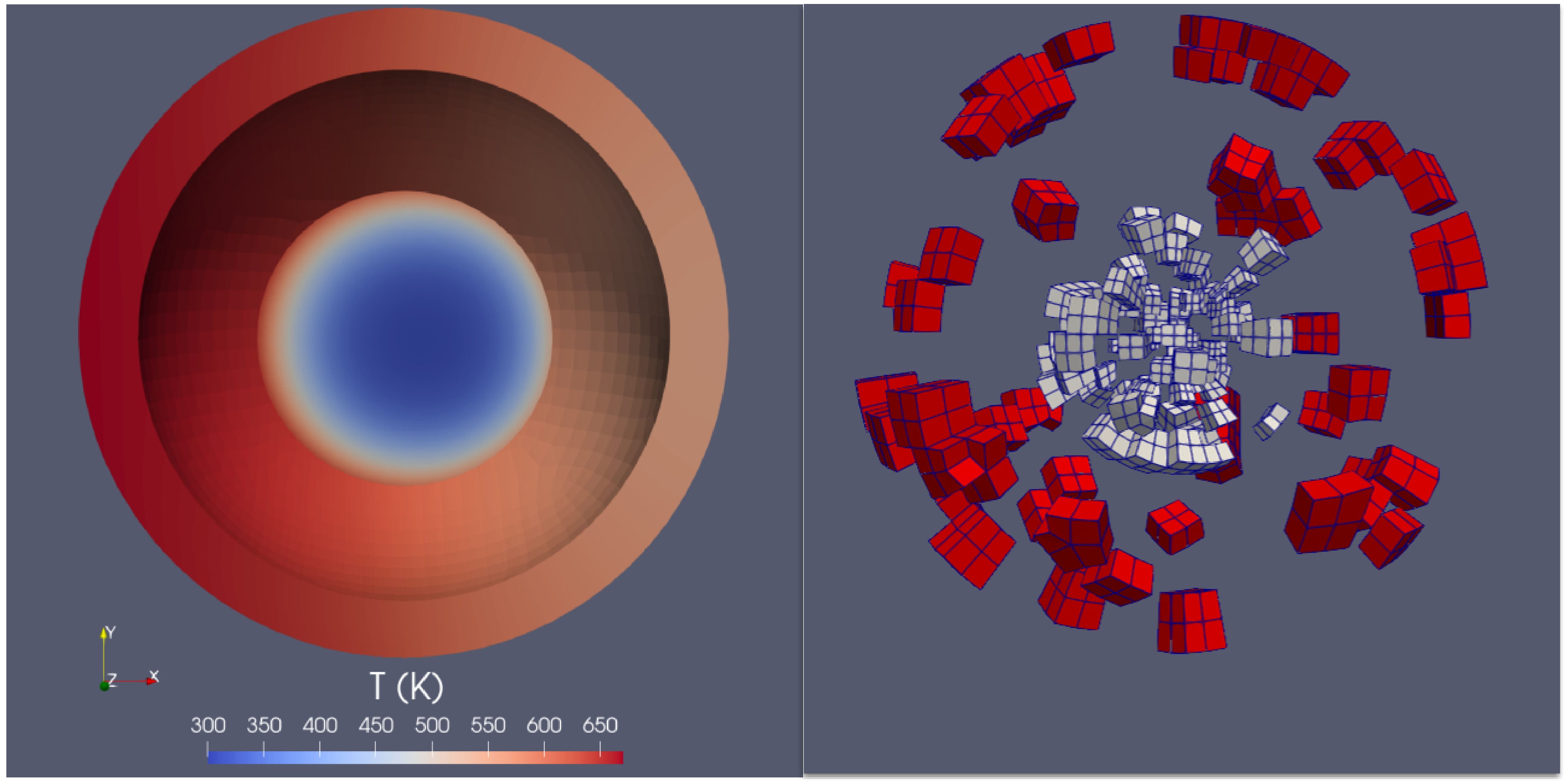
$$\boldsymbol{\Phi}\hat{\boldsymbol{x}}^n = \underset{v \in \text{range}(\boldsymbol{\Phi})}{\text{argmin}} \|\boldsymbol{r}^n(v)\|_2$$
$$n = 1, \cdots, T$$

$\boldsymbol{\Phi}$ is a *linear* trial subspace

This works well for many problems

- Coupled transient nonlinear conduction/radiation heat transfer
- LSPG for energy equation
- Galerkin for radiosity

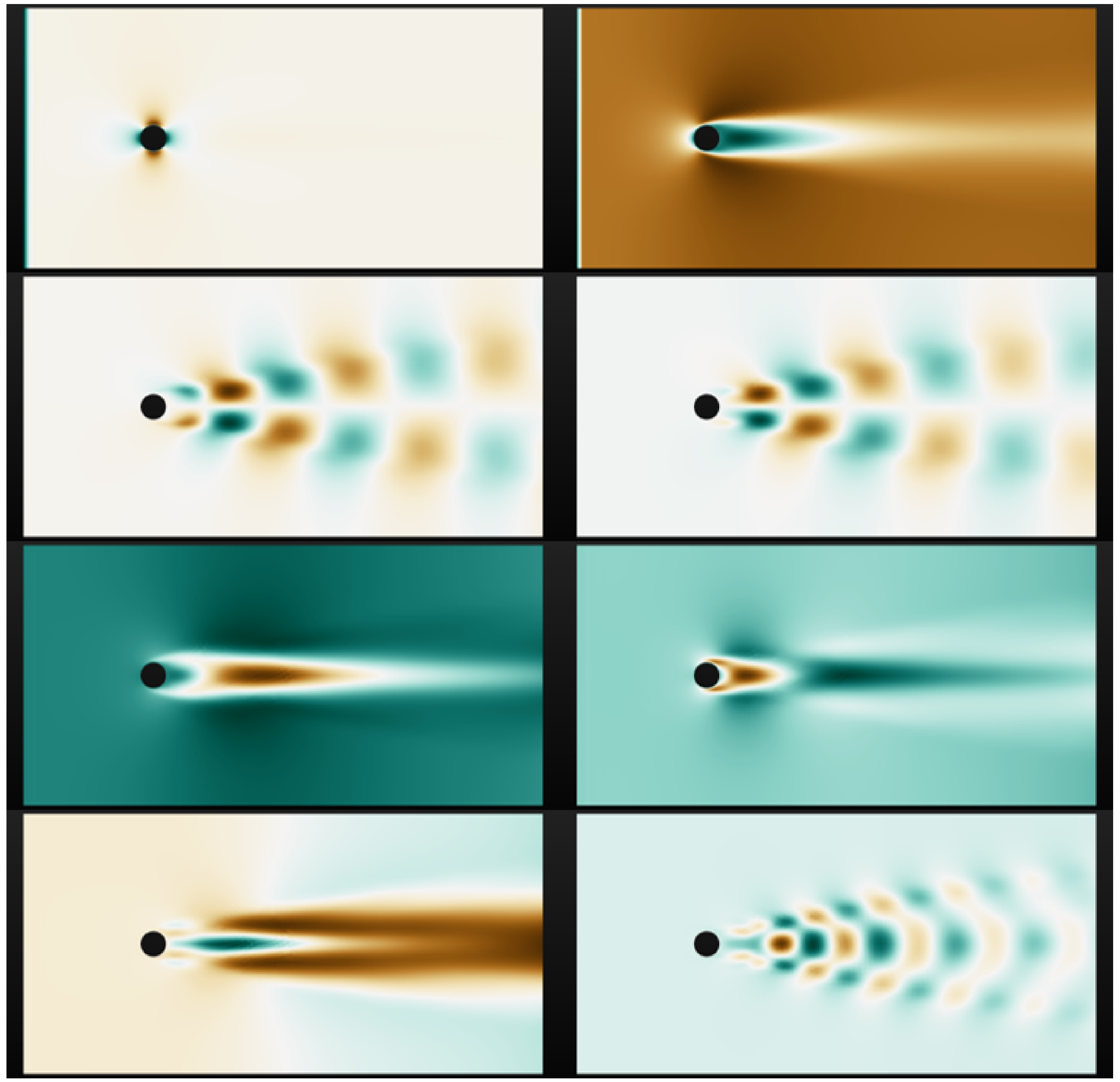




What's the catch?

Kolmogorov n-width limitation

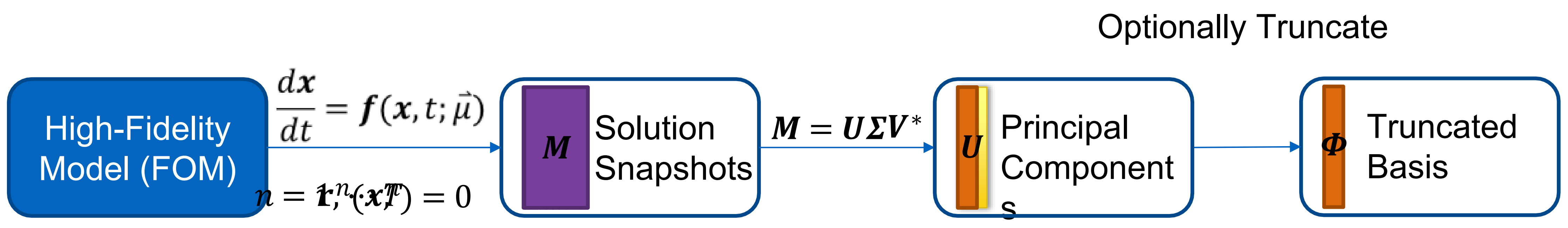
- Linear trial subspace sufficient for problems where the singular values of $\boldsymbol{M}$ decay rapidly (i.e. diffusion dominated problems).
- For other problems the singular values decay slowly and many of the columns from $\boldsymbol{U}$ are required to be retained in $\boldsymbol{\Phi}$ to achieve accurate solutions.
- ROM computational cost is closely tied to the number of modes retained
  - Newton-Raphson iteration costs scale quadratically with the trial subspace dimension



Solution is Nonlinear trial subspaces

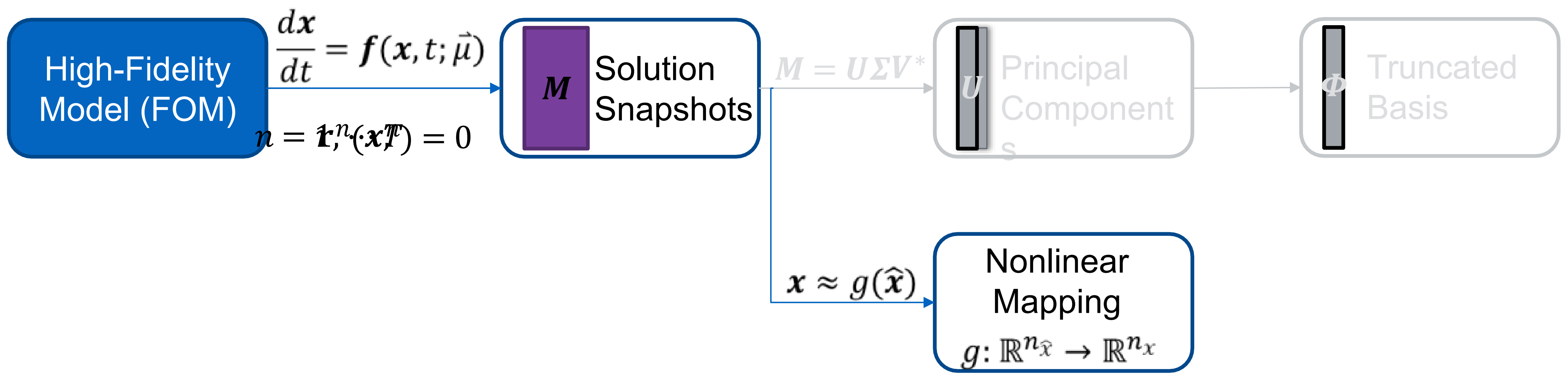
pROM Workflow Overview

Nonlinear trial subspace


Optionally Truncate
High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$n = \mathbf{r}^n(\boldsymbol{x}^n) = 0$
$\boldsymbol{M}$
Solution Snapshots
$\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$
$\boldsymbol{U}$
Principal Components
$\boldsymbol{\Phi}$
Truncated Basis

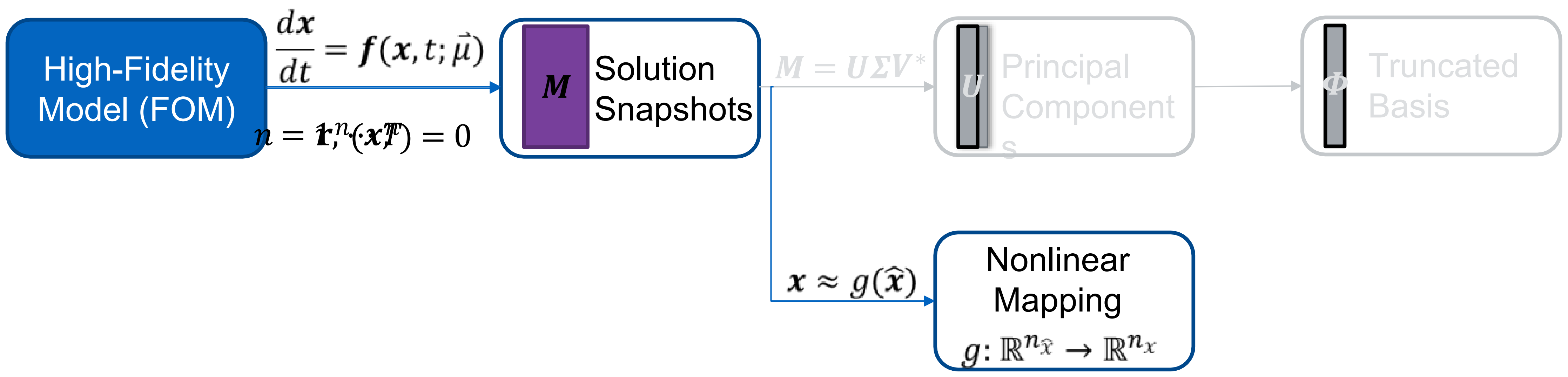
pROM Workflow Overview

Nonlinear trial subspace



pROM Workflow Overview

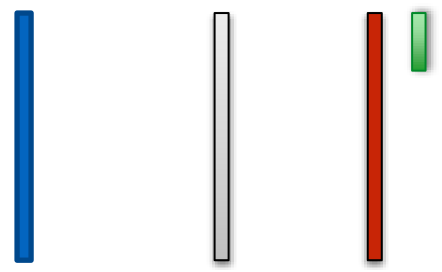
Nonlinear trial subspace



High-Fidelity Model (FOM)
$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \vec{\mu})$
$\boldsymbol{r}^n(\boldsymbol{x}^n) = 0$
$\boldsymbol{M}$
Solution Snapshots
$M = U\Sigma V^*$
U
Principal Components
Φ
Truncated Basis
$\boldsymbol{x} \approx g(\hat{\boldsymbol{x}})$
Nonlinear Mapping
$g: \mathbb{R}^{n_{\hat{x}}} \to \mathbb{R}^{n_x}$

Approximate FOM State

$$\boldsymbol{x}(t;\vec{\mu}) \approx \tilde{\boldsymbol{x}}(t;\vec{\mu}) = \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\vec{\mu}), t; \vec{\mu})$$



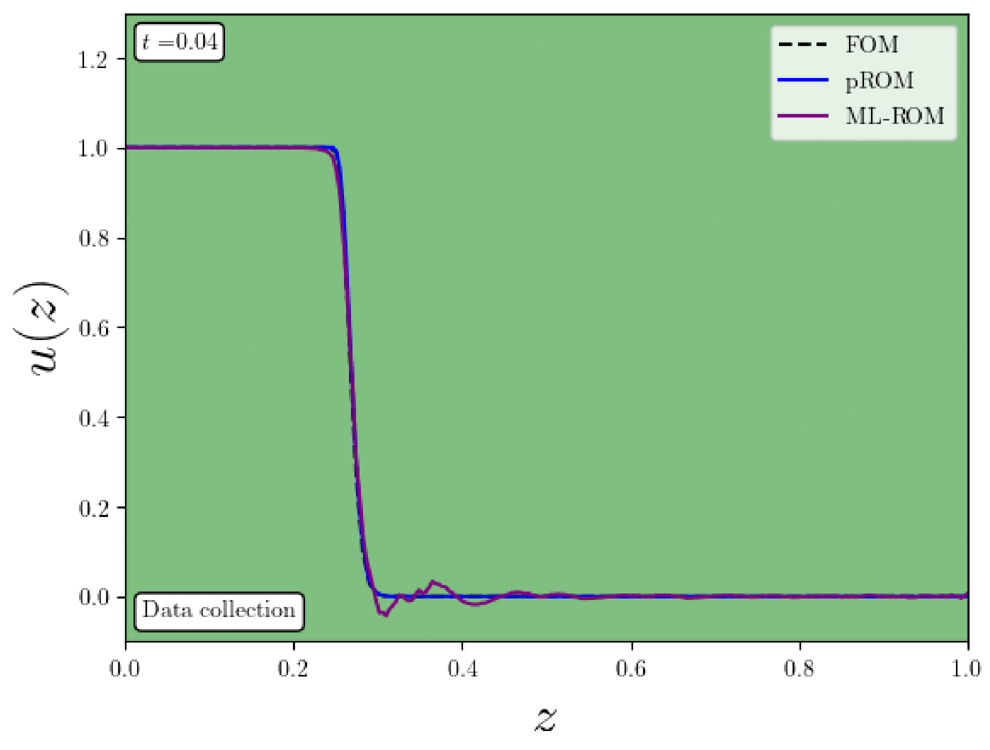
Project System Dynamics

Galerkin Projection

$$\frac{\partial \boldsymbol{x}}{\partial t} = \underset{v \in \mathbb{R}^{n_{\hat{x}}}}{\operatorname{argmin}} \|\boldsymbol{r}(\boldsymbol{J}(\hat{\boldsymbol{x}})v, \boldsymbol{g}(\hat{\boldsymbol{x}}), t; \vec{\mu})\|_2$$

LSPG Projection

$$g(\hat{\boldsymbol{x}}^n) = \underset{v \in \mathbb{R}^{n_{\hat{x}}}}{\operatorname{argmin}} \|\boldsymbol{r}^n(g(v))\|_2$$
$$n = 1, \cdots, T$$



$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial z} = 0$$

Build a pROM using data for $t \in [0, 0.5]$

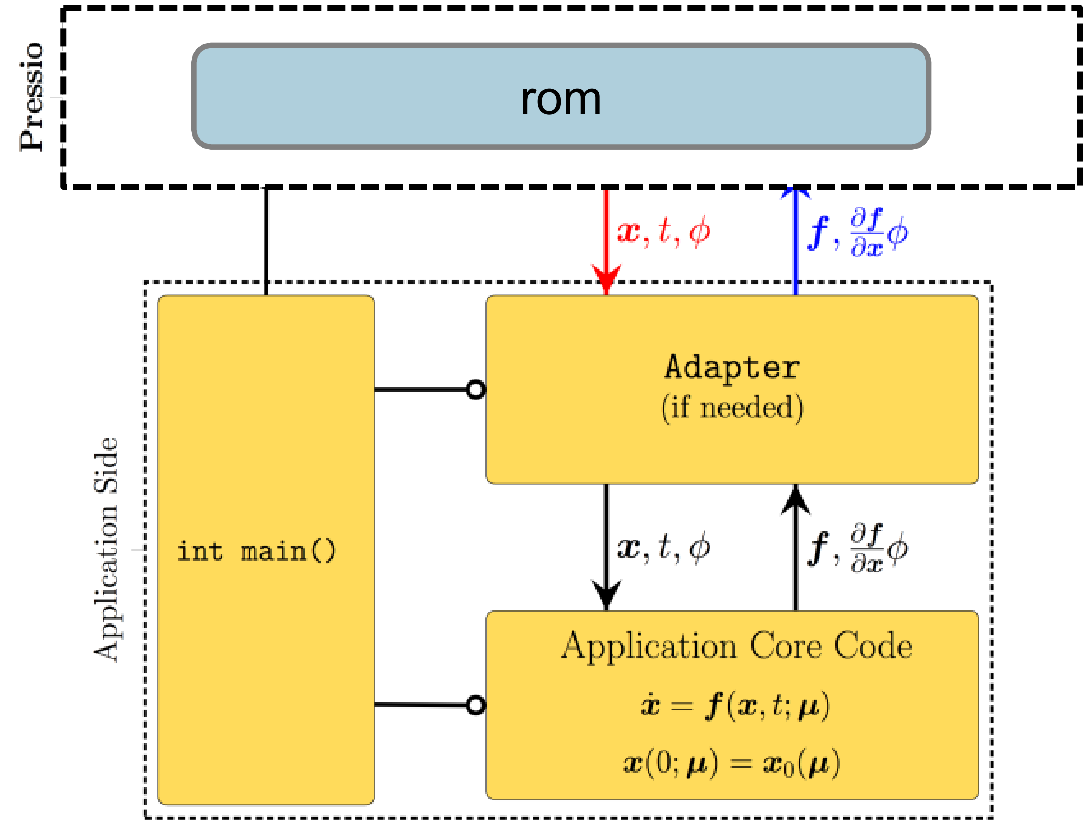
Use the pROM to predict for $t \in [0, 1]$

Pressio

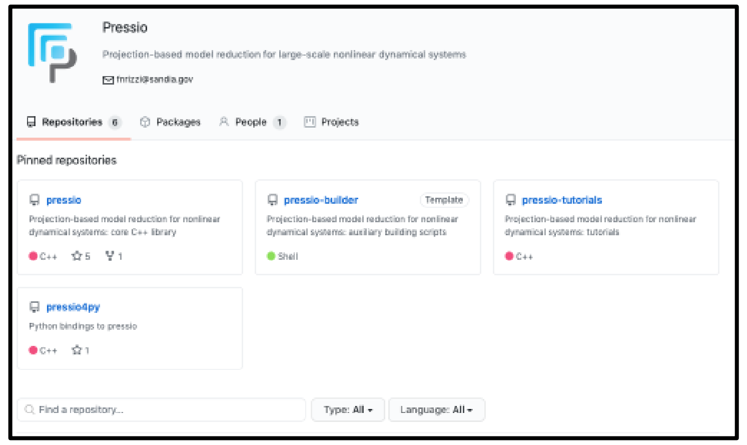
- A computational framework aimed at providing performant pROMs to **generic** application codes

- **Open source code developed at Sandia:**
  - Lead developer: Francesco Rizzi
  - Team includes: Patrick Blonigan, Eric Parish, Kenny Chowdhary, John Tencer, Victor Brunini, Flint Pierce, and more
  - Former developers: Kevin Carlberg and Mark Hoemmen

- **Main idea:**
  - Separate the "application" and the ROM
  - ROM methods are contained in the Pressio framework
  - Pressio "plugs in" to an application code



Pressio
rom
$\boldsymbol{x}, t, \phi$
$\boldsymbol{f}, \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\phi$
Application Side
int main()
Adapter
(if needed)
$\boldsymbol{x}, t, \phi$
$\boldsymbol{f}, \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\phi$
Application Core Code
$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu})$
$\boldsymbol{x}(0; \boldsymbol{\mu}) = \boldsymbol{x}_0(\boldsymbol{\mu})$


Pressio

Projection-based model reduction for large-scale nonlinear dynamical systems

fnrizzi@sandia.gov

Repositories 6 | Packages | People 1 | Projects

Pinned repositories

pressio
Projection-based model reduction for nonlinear dynamical systems: core C++ library
C++ 5 1

pressio-builder Template
Projection-based model reduction for nonlinear dynamical systems: auxiliary building scripts
Shell

pressio-tutorials
Projection-based model reduction for nonlinear dynamical systems: tutorials
C++

pressio4py
Python bindings to pressio
C++ 1

Find a repository... | Type: All | Language: All

https://github.com/Pressio

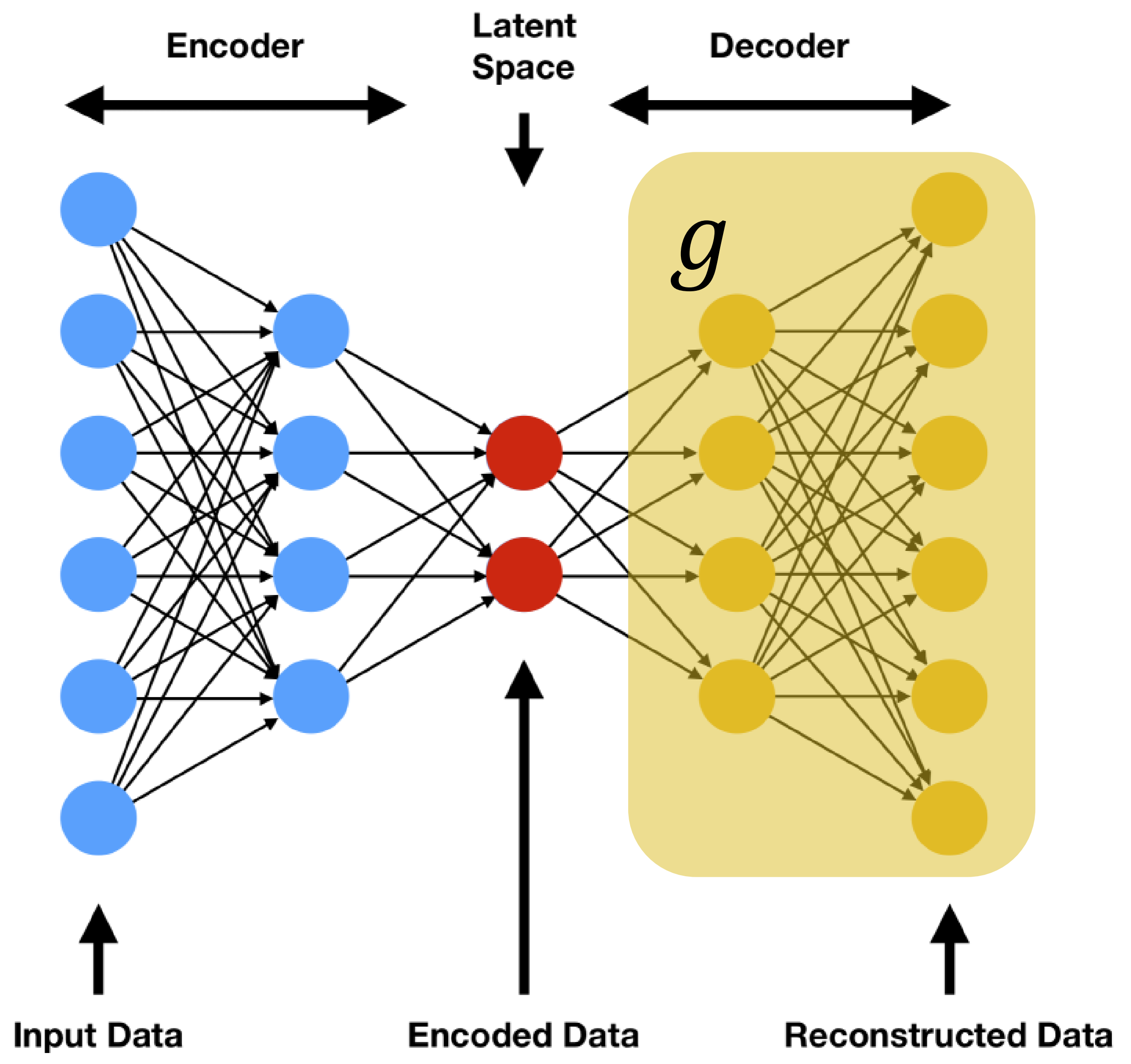
Pressio High-Level Features

- Header-only C++11 library
  - Benefits portability
  - Leverages C++11 and metaprogramming for type detection and compile-time dispatching
- Supports HPC performance portability (Kokkos)
- Natively support data structures from Trillinos
  - tPetra
  - tPetraBlock
  - ePetra
- Supports a Python API
  - Enables Python users to use the C++ Pressio functionalities from Python
- Supports Galerkin, LSPG, and WLS ROMs (w/ hyperreduction)
- **Supports arbitrary nonlinear mappings for state reduction**

How to define the mapping $g(\hat{x})$

Autoencoders, but what kind?

- Dense autoencoders
  - — Restricted to small states
  - — Parameter inefficient
- Convolutional autoencoders [1]
  - + Parameter efficient
  - — Not very fast (no hyper-reduction)
  - — Limited to structured data
- Shallow masked autoencoders [2]
  - + Faster evaluation (supports hyper-reduction)
  - + Supports unstructured data
  - — Accuracy very sensitive to network width
- Graph convolutional autoencoders [3]
  - + Parameter efficient (like traditional convolutional autoencoders)
  - + Supports unstructured data
  - + Robust accuracy
  - — Not very fast (no hyper-reduction)



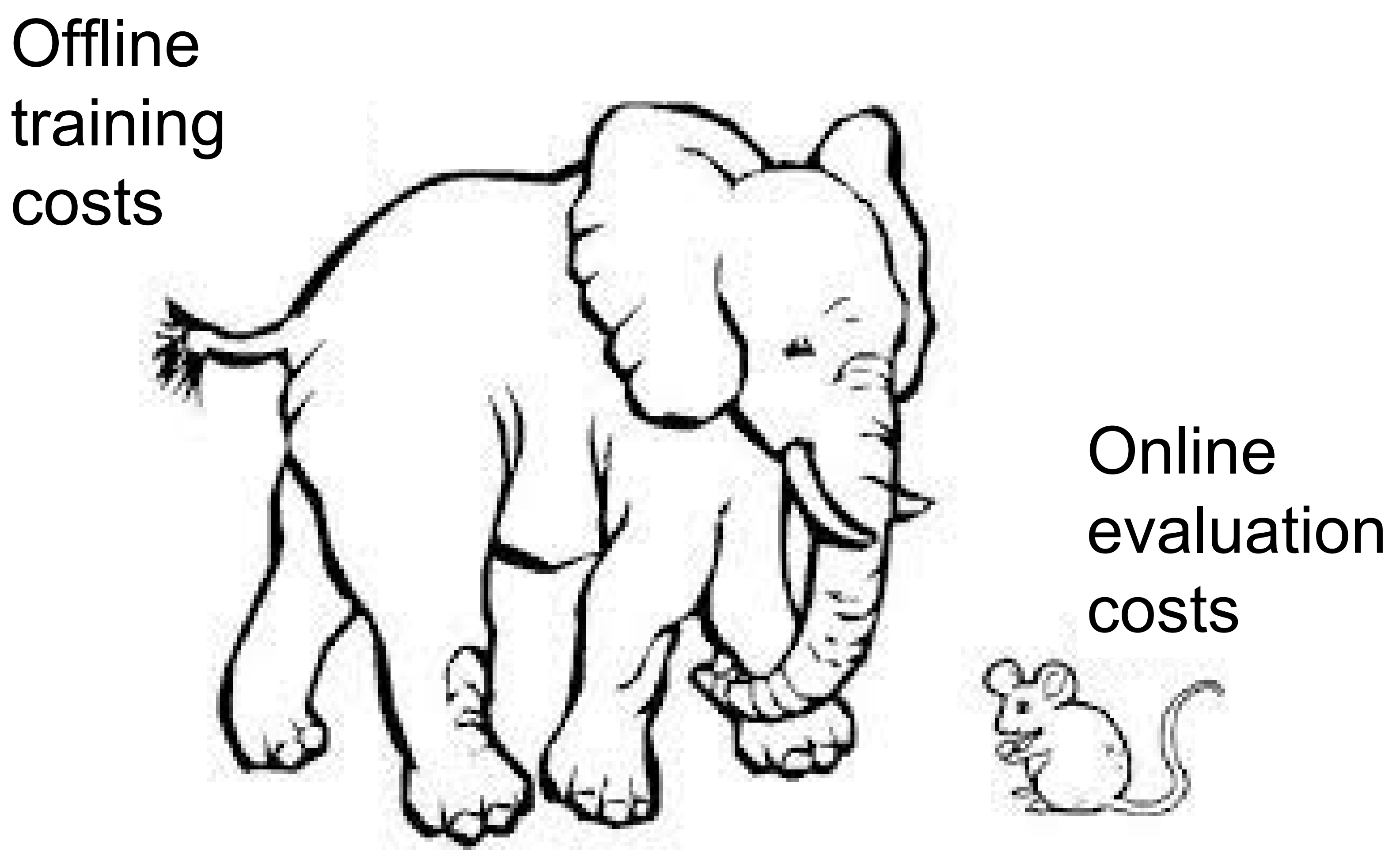
1. Lee, Kookjin, and Kevin T. Carlberg. "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders." *Journal of Computational Physics* 404 (2020): 108973.
2. Kim, Youngkyu, et al. "A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder." *arXiv preprint arXiv:2009.11990* (2020).
3. Tencer, John, and Kevin Potter. "A Tailored Convolutional Neural Network for Nonlinear Manifold Learning of Computational Physics Data Using Unstructured Spatial Discretizations." *SIAM Journal on Scientific Computing* 43.4 (2021): A2581-A2613.

There's an elephant in the room

Training time and scalability

- It's currently very difficult for non-pROM experts to assess if using a pROM is worthwhile for a new application.
  - Absent this information, people are opting to not adopt the technology and instead rely on existing simulation tools.

- We shouldn't sweep training costs under the rug. We should be reporting training costs for our models alongside evaluation costs.
- We should be evaluating our methods on realistically large problems.
- We should be investing research effort into speeding up and automating training in the same way we invest in improving accuracy or decreasing online costs.



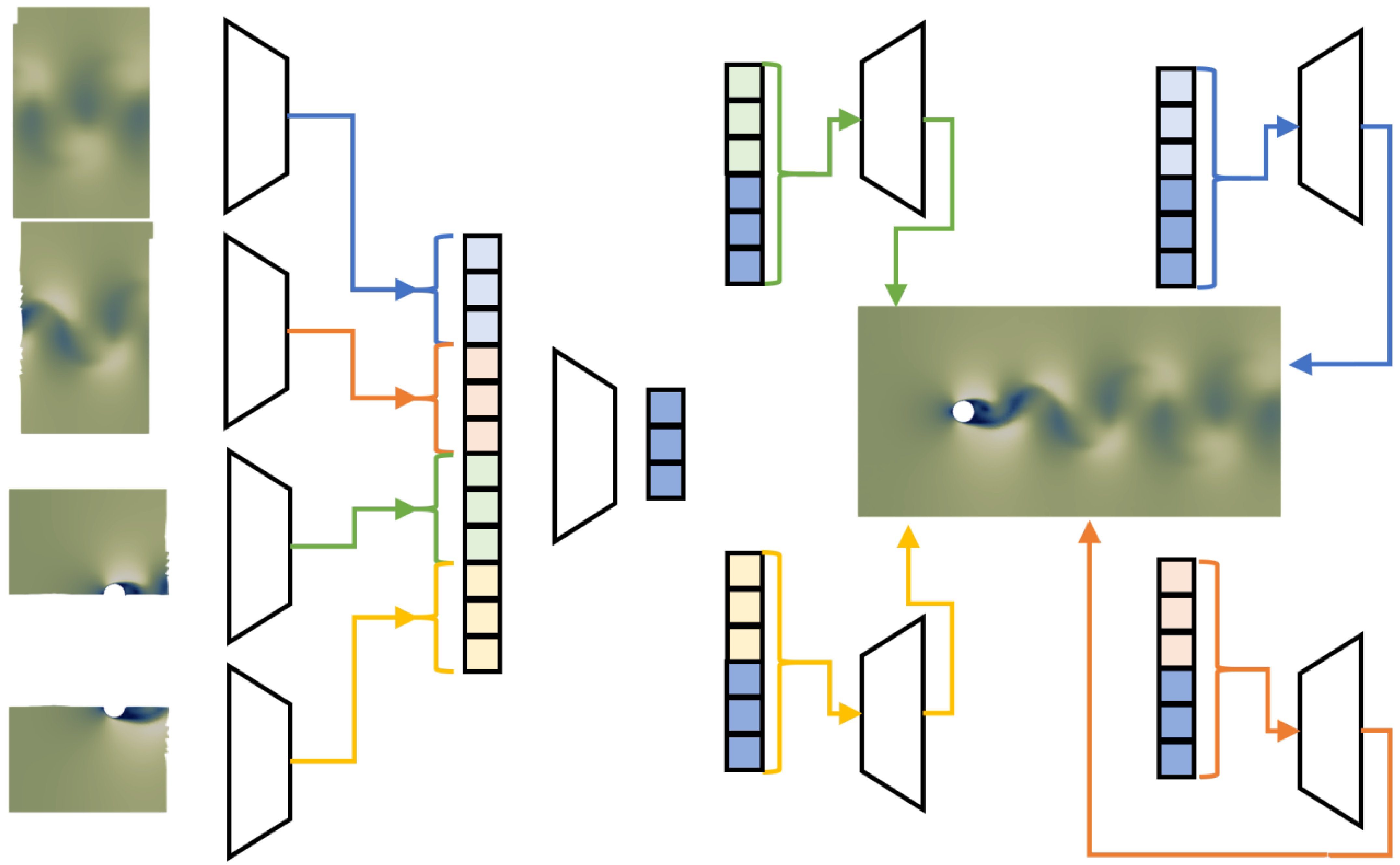
Some things we're currently playing around with:

Parallel training on spatially decomposed data

Pretty simple application problem decomposed onto 4 processors

Trained 9 different models in parallel in a coupled fashion.

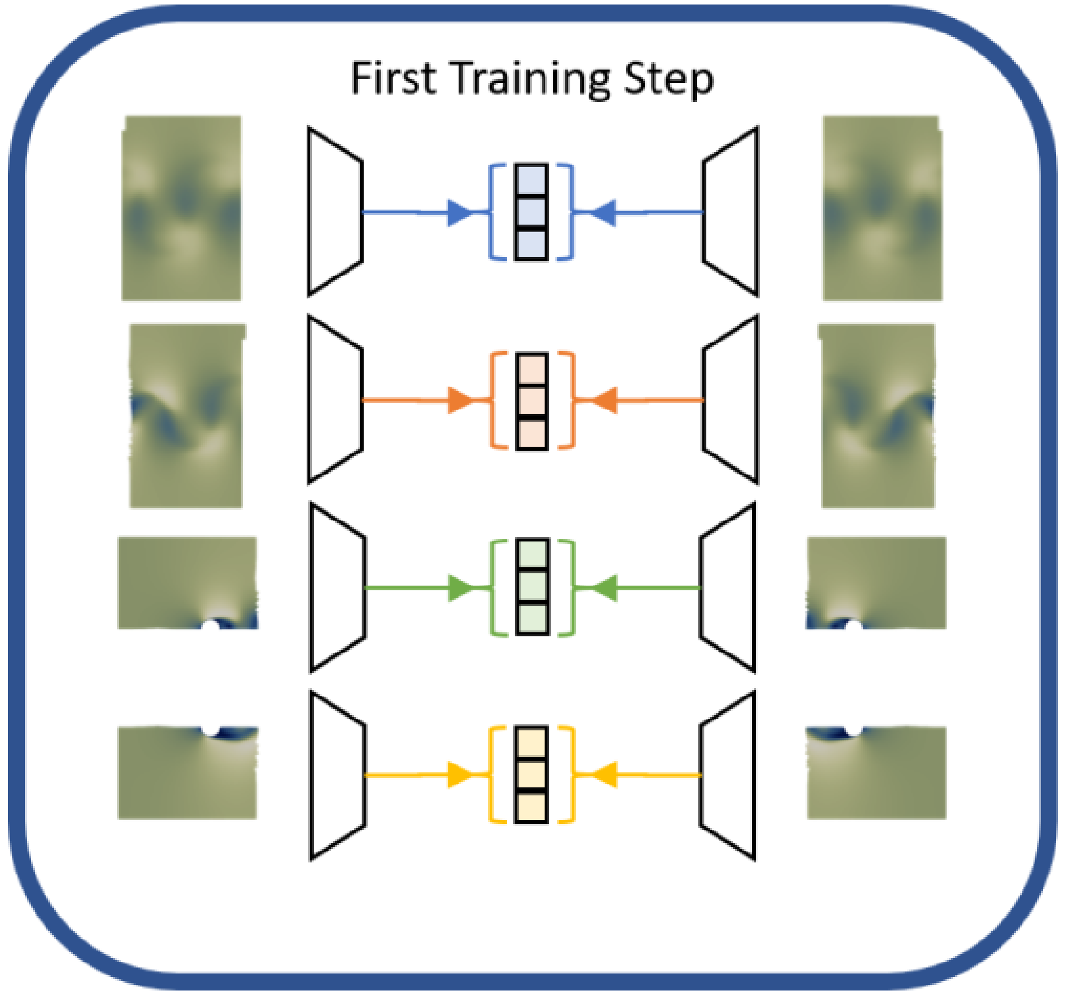
Additional "global encoder" consolidates information across processor boundaries seems like a good idea, but introduces a lot of extra communication and complexity without providing a ton of benefit.

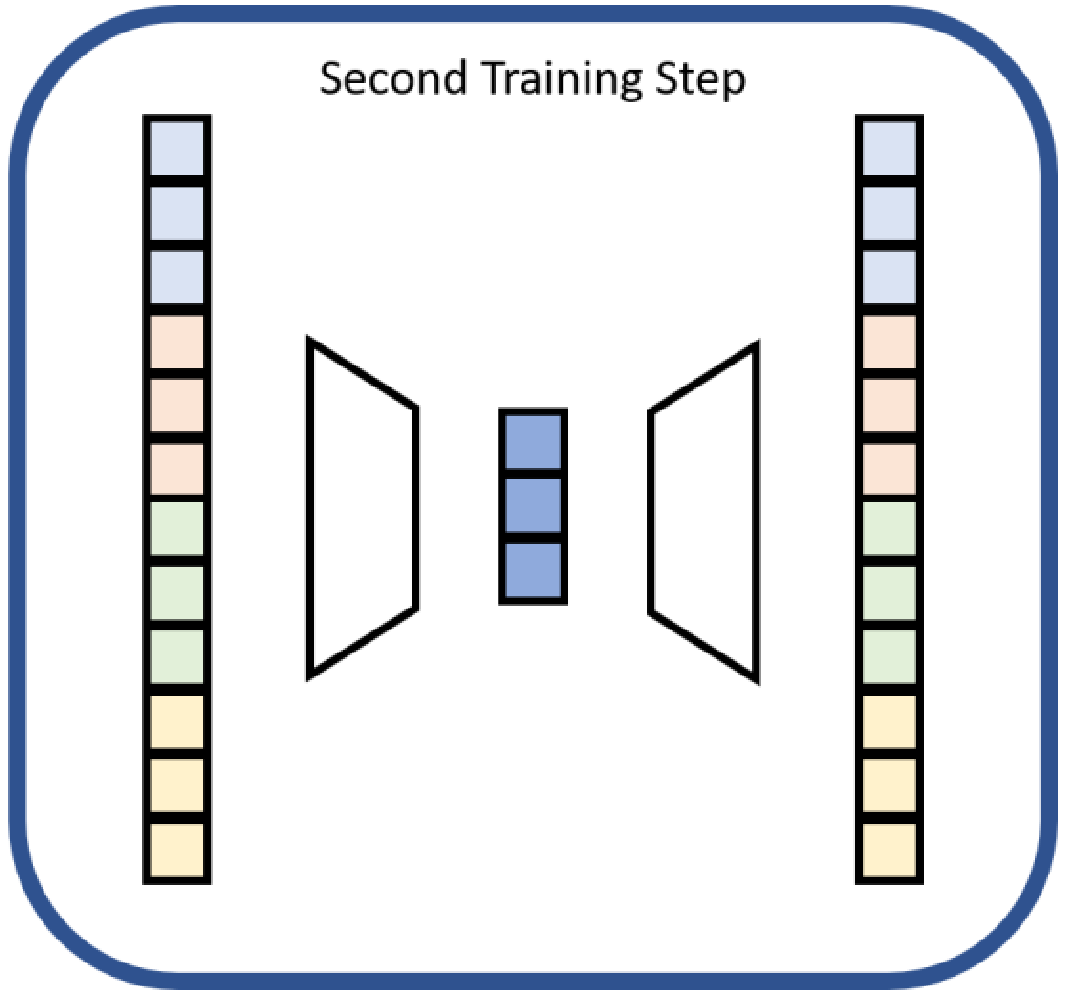


Some things we're currently playing around with:

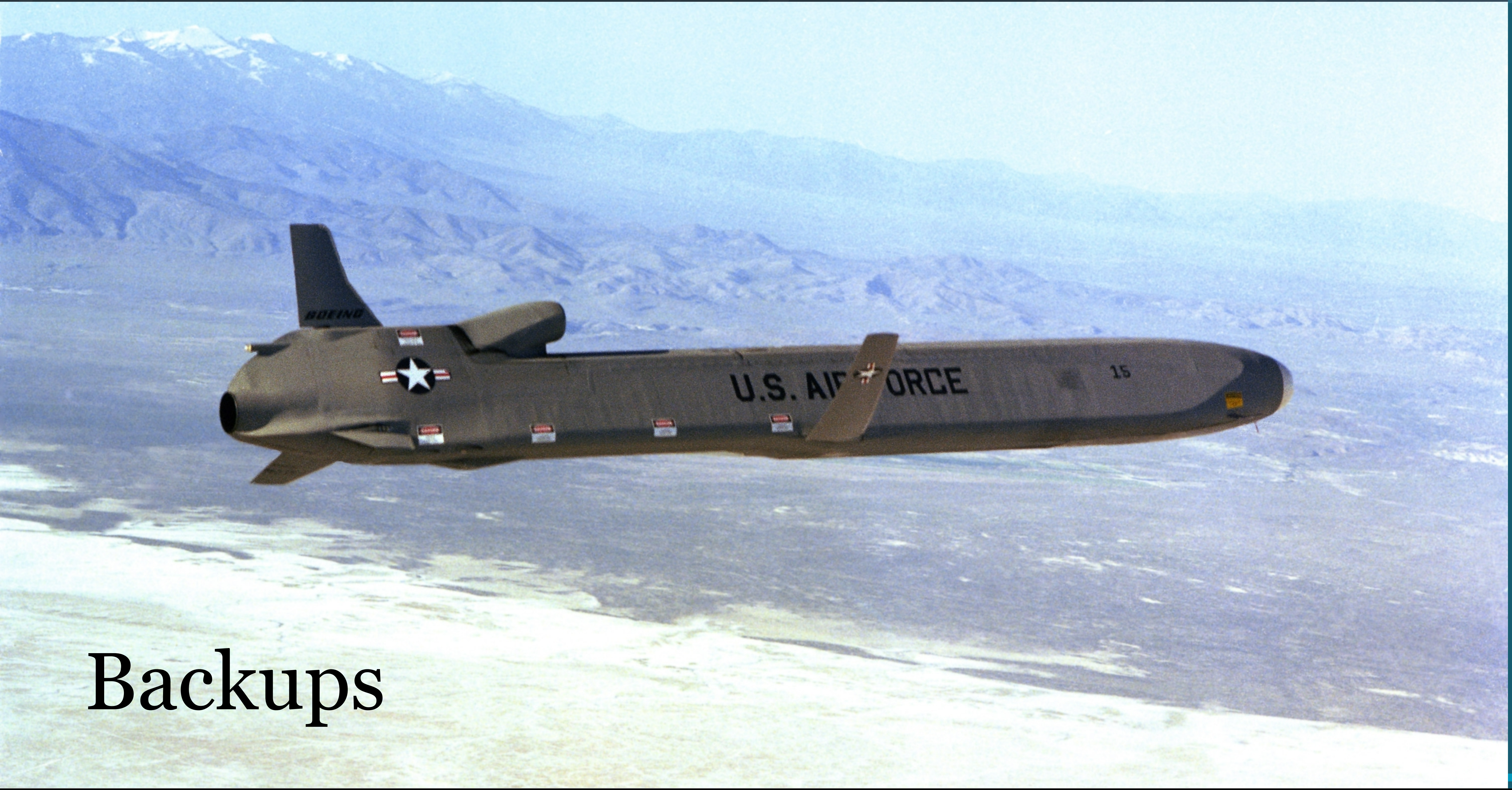
Parallel training on spatially decomposed data

- Can get comparable performance with a 2-step procedure where local autoencoders are trained separately and then used to define a single global latent space.
- Needed an additional penalty term at the processor boundaries. Errors on processor interfaces count double. This avoided the need to couple the trainings.
- The second step is probably only necessary for larger processor counts.
- The chunks might need to be smaller than what is typical for CFD simulations. When we tried this for a larger proprietary problem, the individual local autoencoders took hours to train and we weren't able to fully parallelize due to hardware resource constraints.



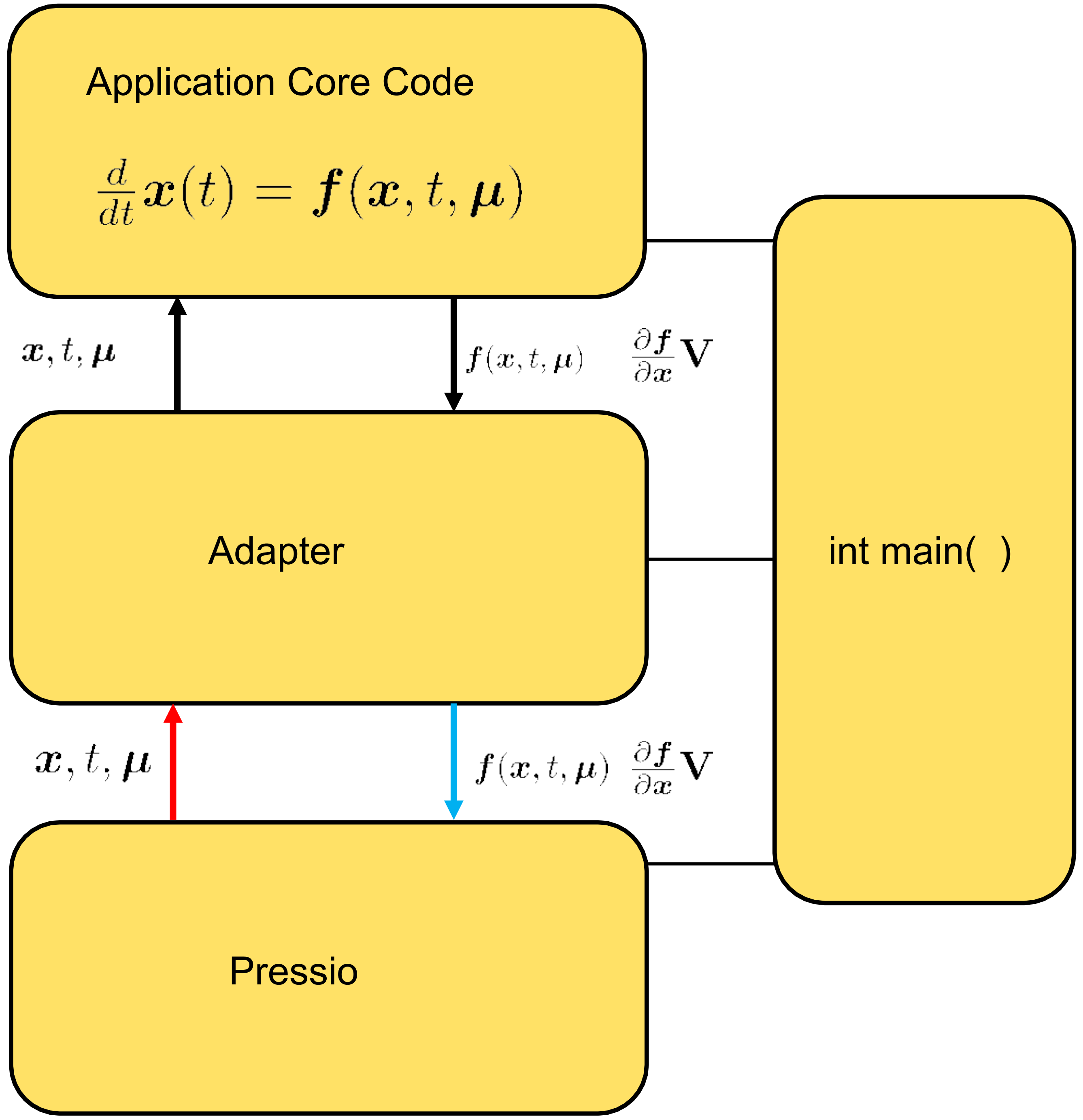


Sandia National Laboratories



Backups

How does Pressio talk to an application?



Application Core Code
$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{\mu})$
$\boldsymbol{x}, t, \boldsymbol{\mu}$
$\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{\mu})$
$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\mathbf{V}$
Adapter
int main( )
$\boldsymbol{x}, t, \boldsymbol{\mu}$
$\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{\mu})$
$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\mathbf{V}$
Pressio

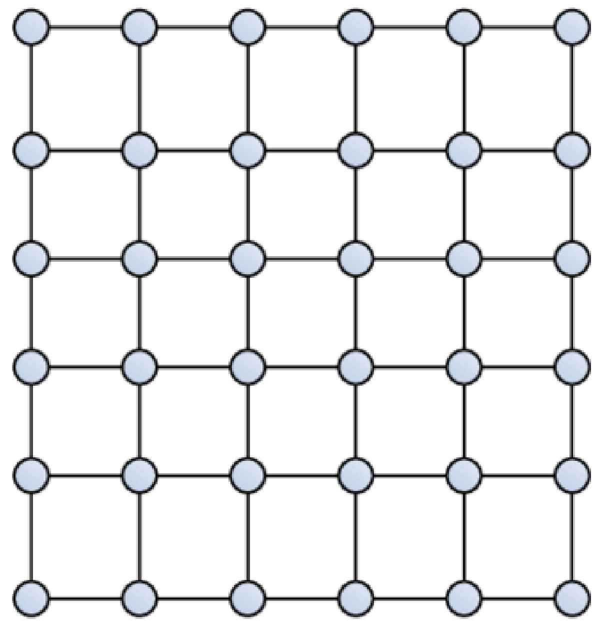
Pressio’s API requires the application to expose two main functions:

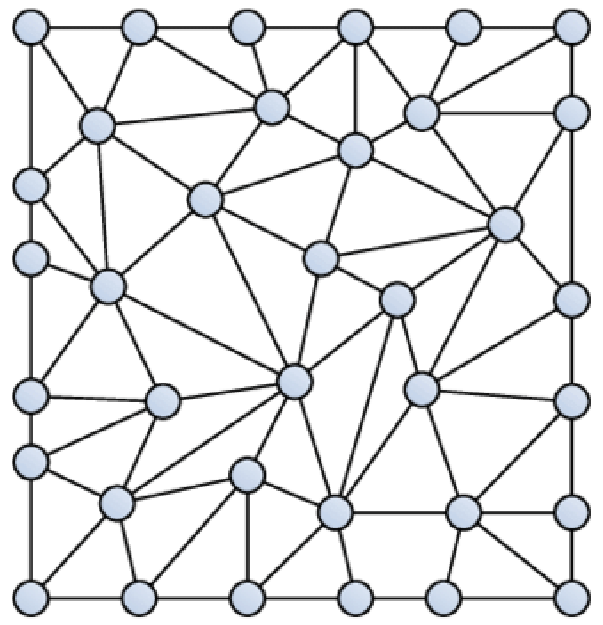
1. velocity: $\boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{\mu})$
2. applyJacobian: $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\mathbf{V}$

Pressio uses these functions to construct the ROMs

Convolutional Layers for Unstructured Data

(Especially PDE data)





Commonly learned filters are often closely related to differential operators

∇_x

-1	0	1
-4	0	4
-1	0	1

x- Sobel Filter

∇_y

1	4	1
0	0	0
-1	-4	-1

y- Sobel Filter

Δ

-1	-1	-1
-1	8	-1
-1	-1	-1

Edge Detection

- Use differential operators defined by the underlying spatial discretization to propagate information.
- Operators can be computed offline or on-the-fly.
- Resulting learned weights will be discretization independent.
- Drop-in replacement for convolutional layers in autoencoder networks.