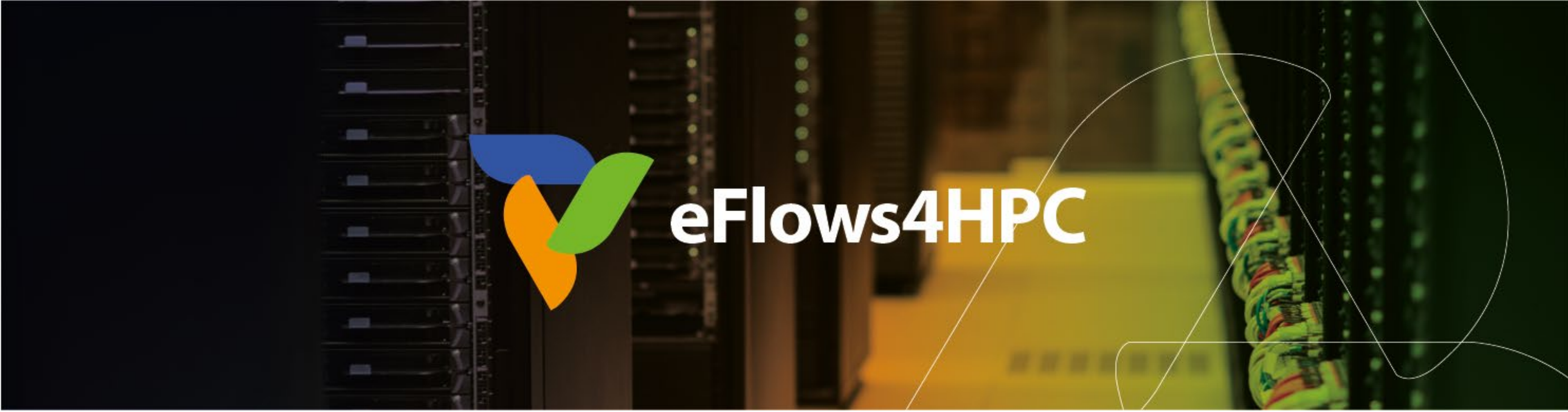


Pillar I

Riccardo Rossi, Joaquin Hernandez, Sebastian Ares de Parga, Nicolas Sibuet, Raul Bravo – CIMNE
Gianluigi Rozza, Giovanni Stabile, Nicola Demo, Karim Yehia - SISSA
Mario Ricchiuto, Nicolas Barral, Sourabh Bhat, Pierre Clouzet - INRIA

Barcelona, Spain

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

PILLAR I
Digital Twin in Manufacturing

Goals of the Pillar

- Provide an integrated workflow enabling the development of ROMs from their inception to their deployment

- Enable the use of HPC resources to speed up the generation process and enable the solution of large problems

PILLAR CONTRIBUTORS (jointly with BSC and UPV)

SIEMENS

CIMNE

KRATOS MULTI-PHYSICS

upgrade your meshes

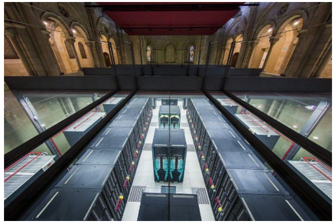
Inria

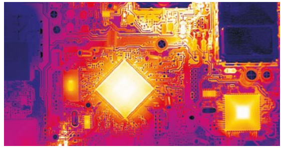
SISSA
Scuola
Internazionale
Superiore di
Studi Avanzati

“Standard” ML/AI workflow

TRAINING (BIG DATA)

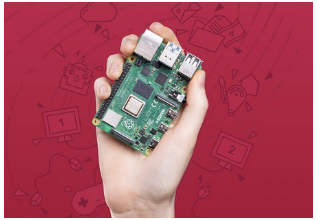
M
L





DATA

INFERENCE



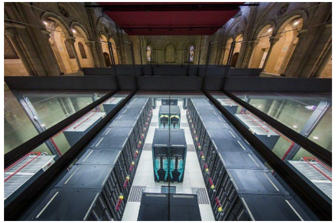
“BLOCK BOX” MODEL

Reduced Order Modelling workflow

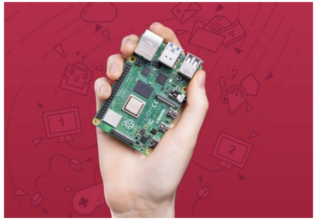
TRAINING

BIG DATA

ML

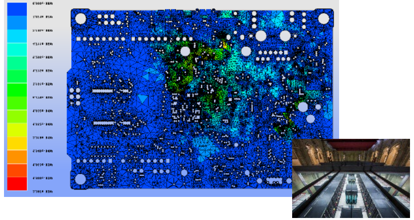


INFERENCE

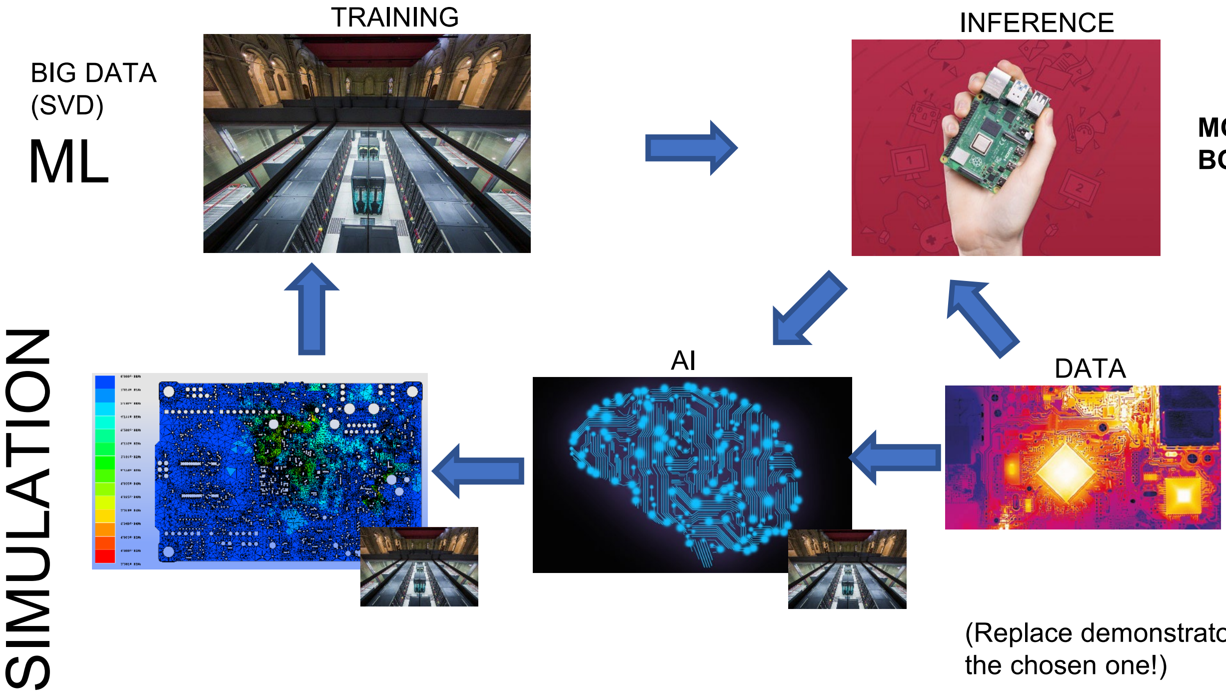


MODELS ARE NOT BLACK BOXES!!

SIMULATION



Project Vision

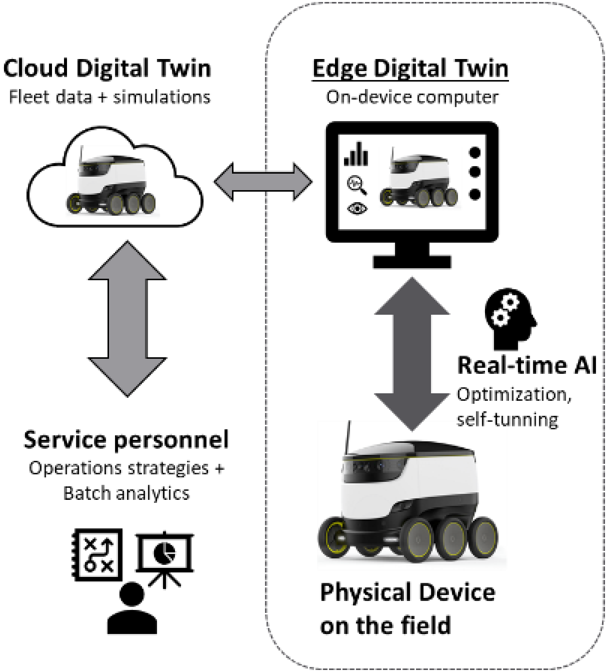


MODELS ARE NOT BLACK BOXES!!

(Replace demonstrator by the chosen one!)

The big picture

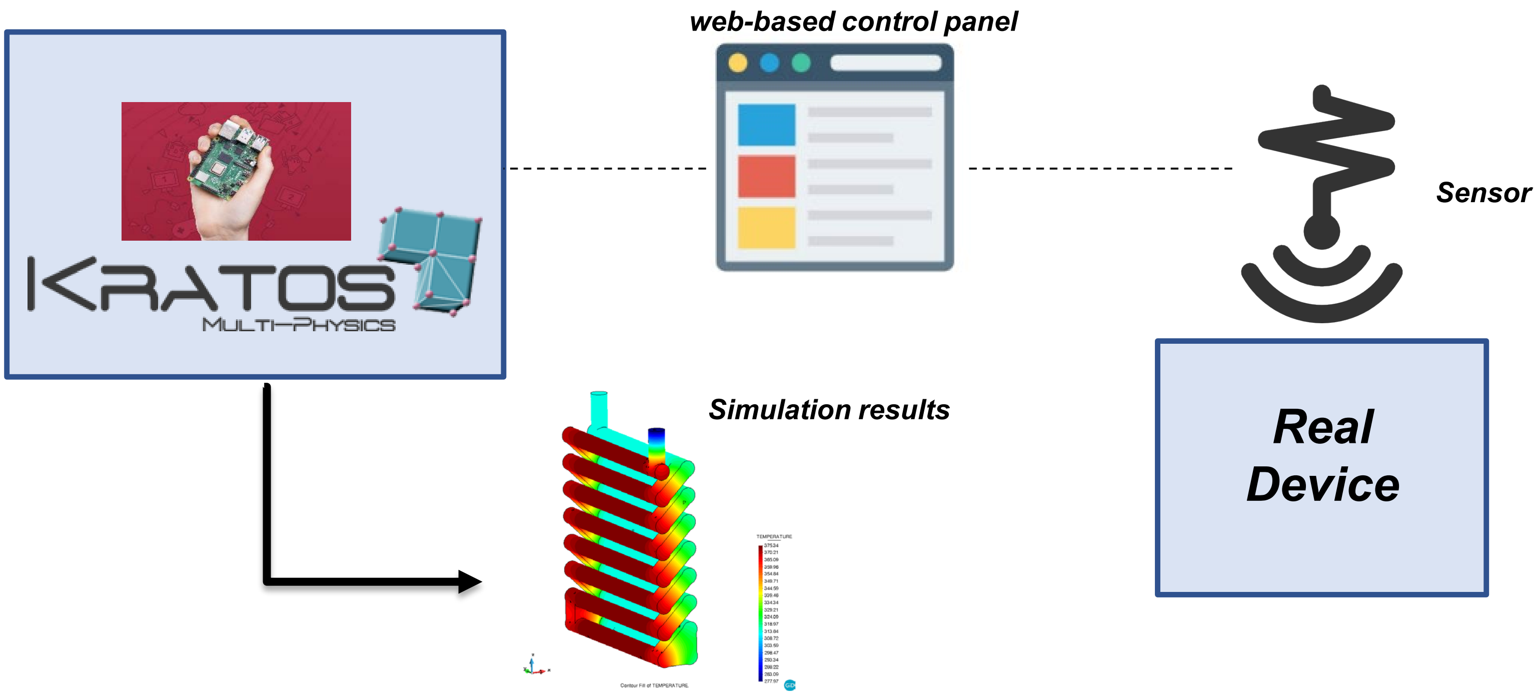
SERVER:
- Hi compute capabilities
- Needs communication
- Cannot be “vital”
- Prone to “cyberattacks”?



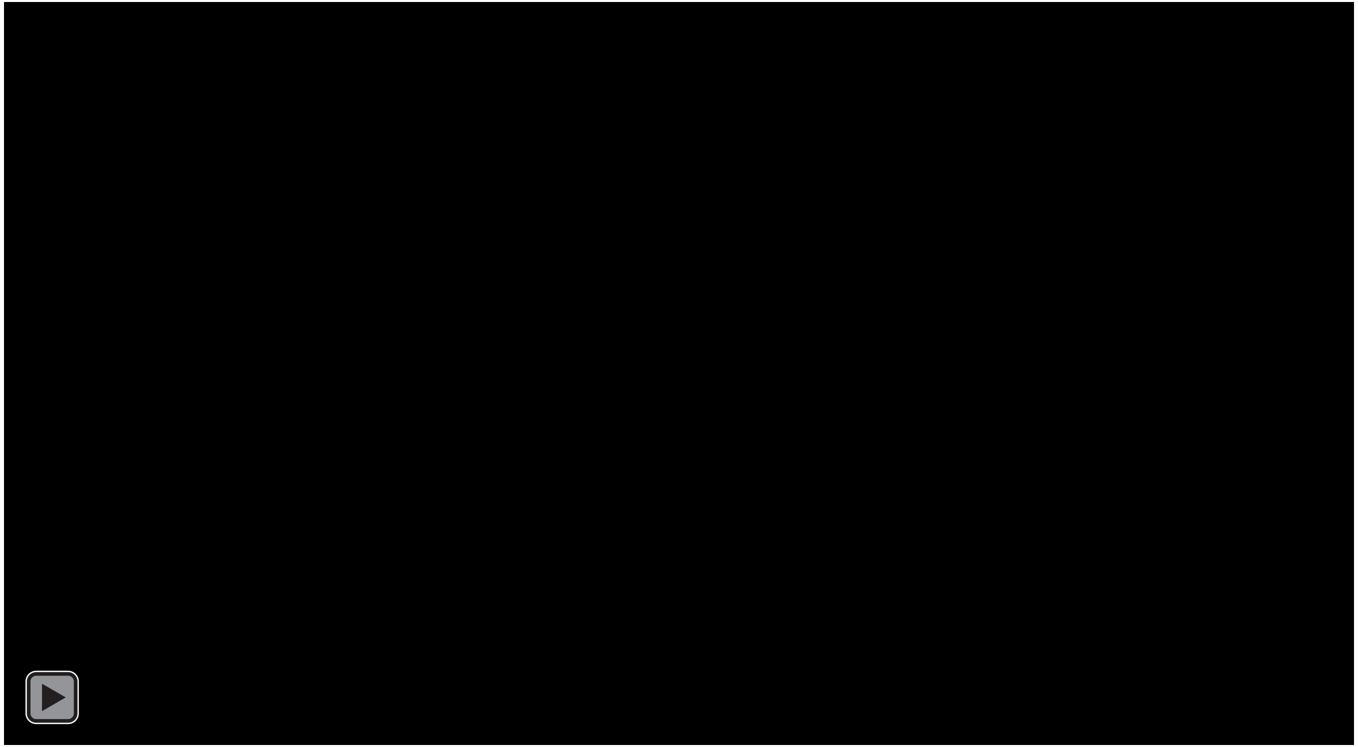
EDGE:
- Low compute capabilities
- No communication needed
- Can take autonomous decisions
- Recomputing “cheaper” than moving data around?
- Safer?

The “link to the rest of the world”

Sensor feedback can be incorporated in the ROM once such a model is available.

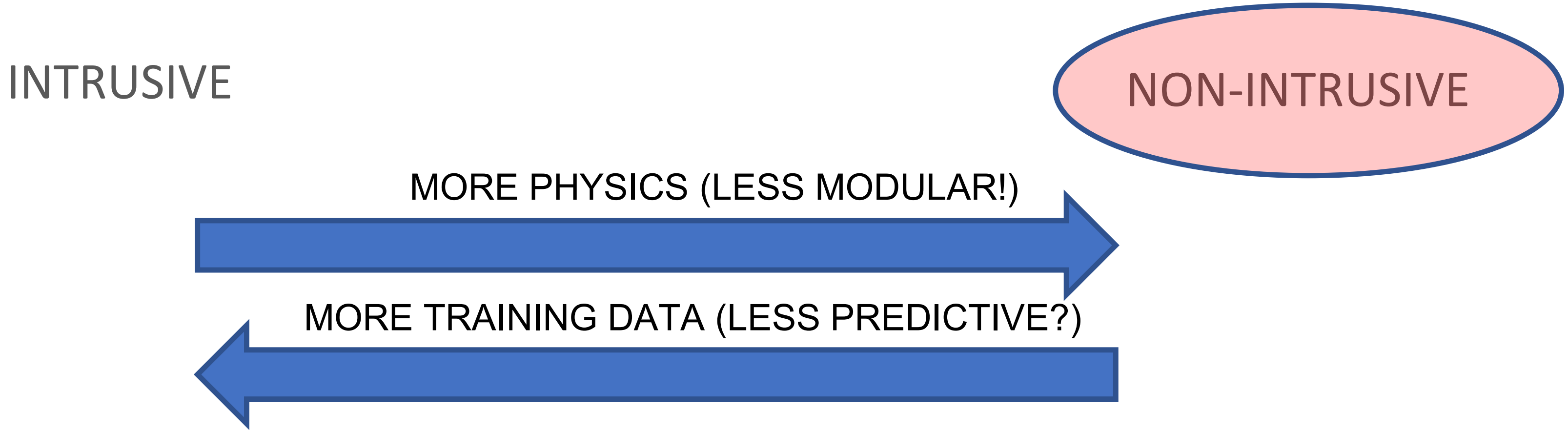


- A thermal problem

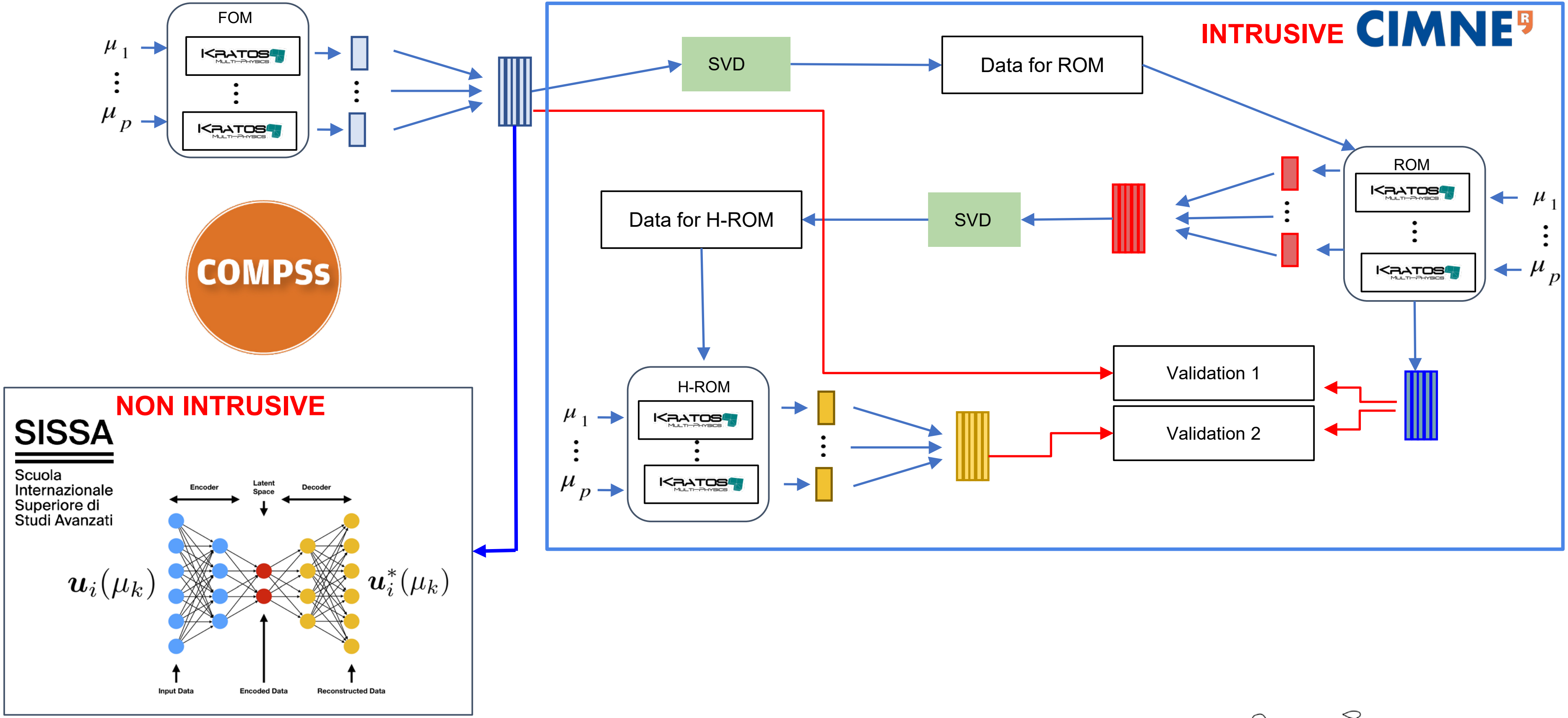


How Is that achieved?

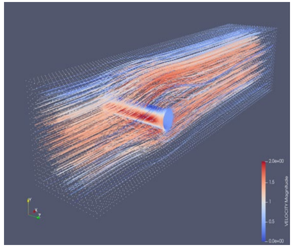
THE AMLETIC DILEMMA:



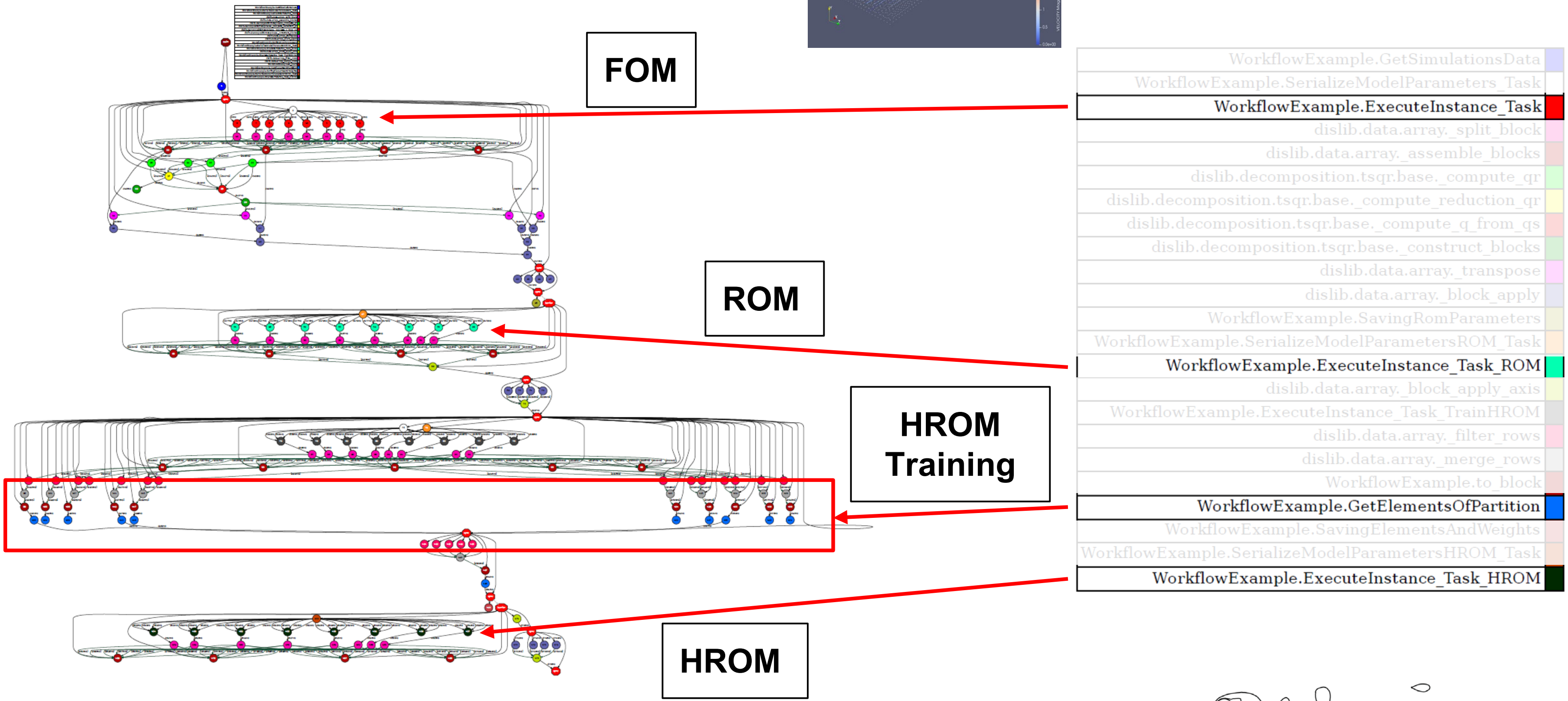
ROM workflow



Parallelized using a Task-Based Approach



CFD Example with 1M dofs
Execution in 4 nodes of supercomputer Nord3



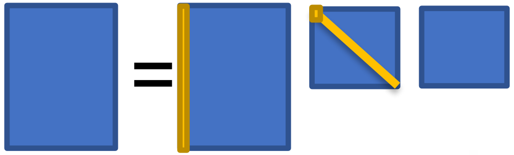
Intrusive ROM workflow

Based On the Idea of “Projection”

In essence we harvest past simulations for “patterns” which we then leverage to accelerate simulations

Singular Value Decomposition (or other autoencoder)

- Example

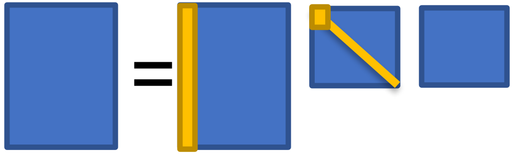


Original Picture

SVD

Picture with 5% of the modes

Singular Value Decomposition (or other autoencoder)



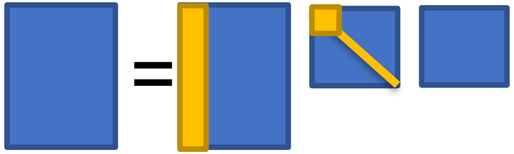
- Example

Original Picture

SVD

Picture with 10% of the modes

Singular Value Decomposition (or other autoencoder)



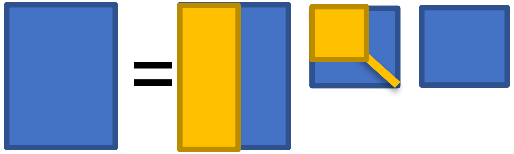
- Example

Original Picture

SVD

Picture with 25% of the modes

Singular Value Decomposition (or other autoencoder)



- Example

Original Picture

SVD

Picture with 50% of the modes

Based On the Idea of “Projection”

IDEA:

only do operations using few modes holding the “important” information

… as in modal analysis … except:

- Done “a posteriori” from exploring available solutions
- Applicable to nonlinear problems

Evolution: From Idea to Product

$\int d\Omega$

First Idea

- May lead to a breakthrough solution
- High Risk
- Academic Task (Post-Doc, PhD or master projects)

Evolution: From Idea to Product

$\int d\Omega$

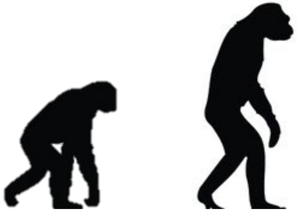
First Idea

Research applications

- Some experimental development
- A working prototype with less associated risk
- Suitable for a small test problem

Evolution: From Idea to Product

$\int d\Omega$



First Idea

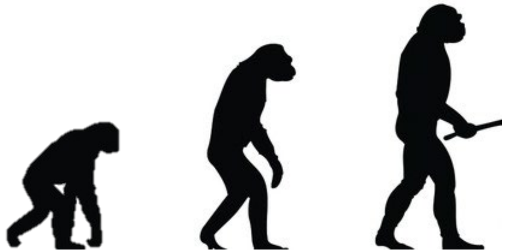
Research applications

Applications

- Rewriting the prototype in more efficient way (Cpp/Fortran)
- Making it more generic
- Avoiding common bottlenecks
- Increasing the code quality
- Adding QA

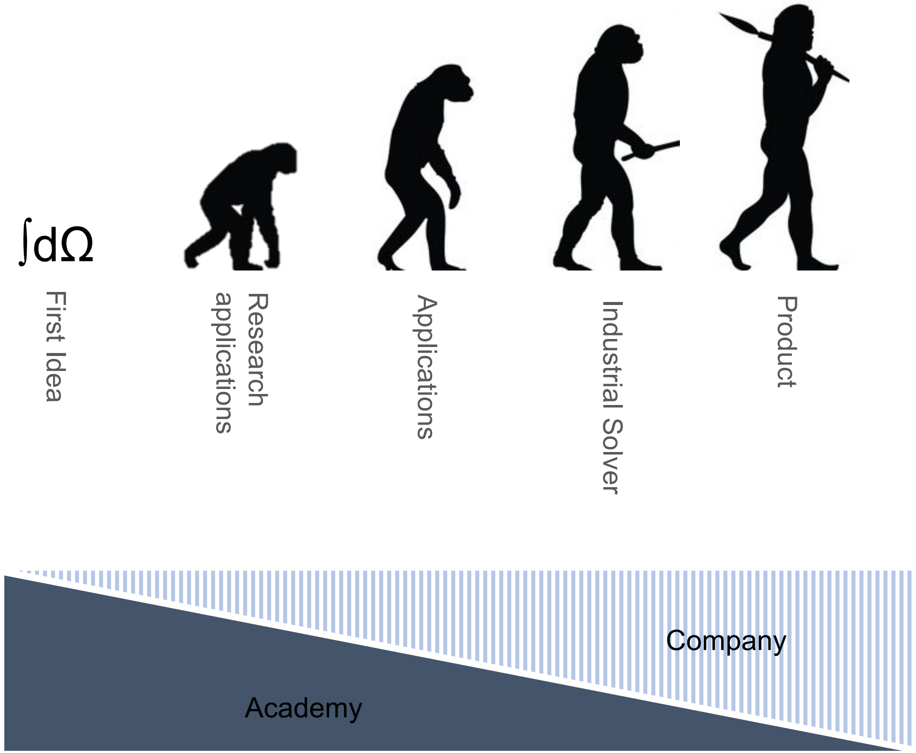
Evolution: From Idea to Product

$\int d\Omega$



- More physics and calibration
- Improving the robustness
- More QA
- Improving the code quality

Evolution: From Idea to Product



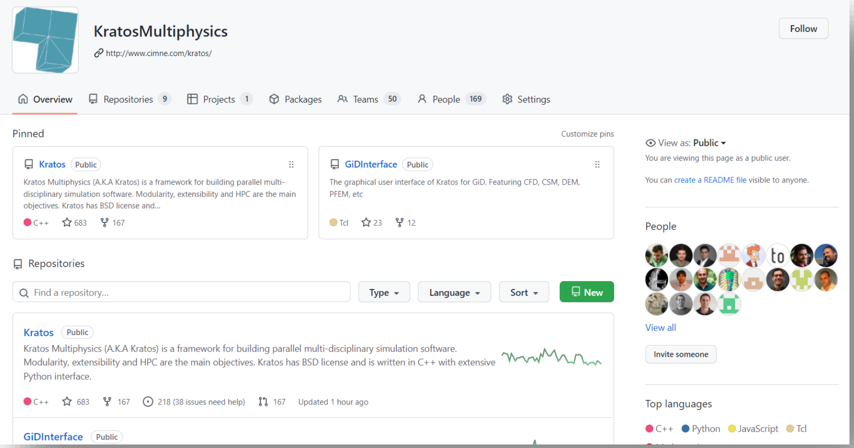
The Solver +

- GUI
- Validation
- Automatization
- Final tuning
- Maintenance
- Documentation
- Distribution
- Support

Kratos Multiphysics

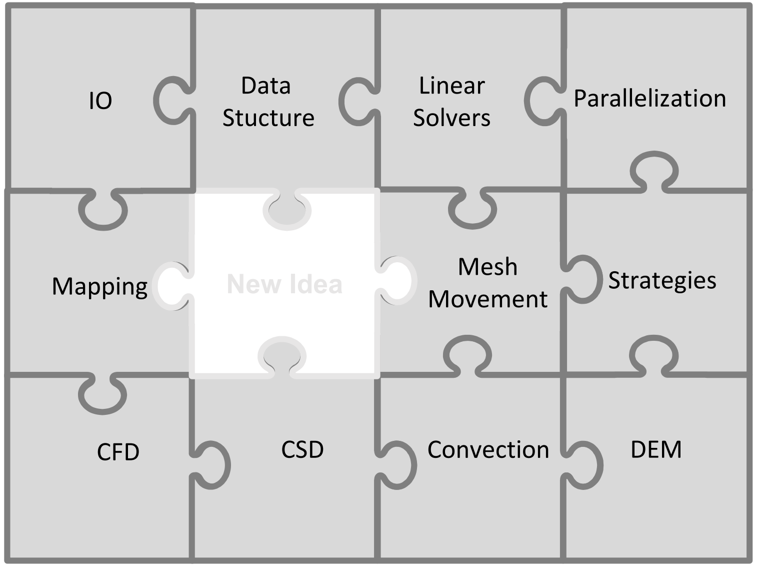
An open framework for parallel Multi-physics programs development

- Since 2001
- ~ 1 million lines of code
- ~ 100 developers
- 20 years of development
- Very modular
- High performance
- Object Oriented C++
- Extensive Python interface
- Open Source and Free (BSD Licence)

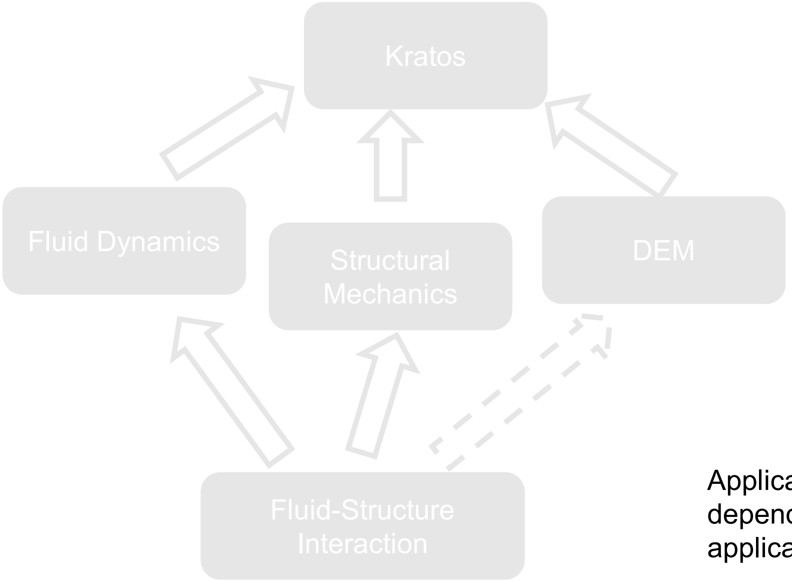


https://github.com/KratosMultiphysics

Kratos Multiphysics



Highly Modular Design & Multi-disciplinarity

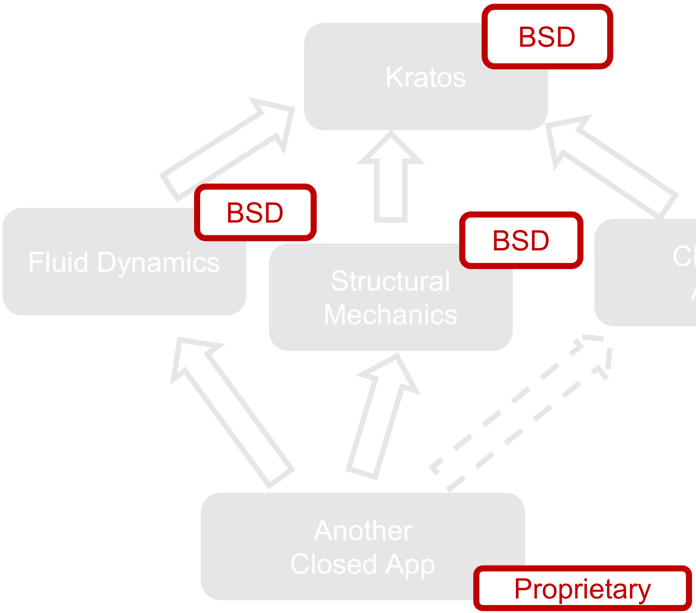


Numerical and programming core

Physics of the problem

Applications can depend on other applications

Flexible License



Kratos Core is BSD and Free

Kratos main apps are also BSD

Each application can have different license
Some can be closed

Dependent applications can also have different license
In case of depending to the proprietary app the restrictions would be imposed

Non-Intrusive ROM workflow

Non-Intrusive ROM workflow (Goal)

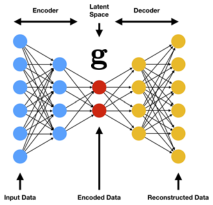
Goal: Implementation of a completely **data-driven non-intrusive ROM** using **nonlinear dimensionality reduction** (**AE**) in a **distributed** environment.

Linear Approach (R-SVD)

Nonlinear Approach (AE)

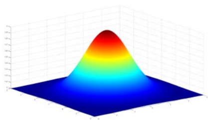
1. Dimensionality Reduction Step

Linear: **R-SVD** → $\mathbf{u} \approx \sum_{\mathrm{i}=1}^{\mathrm{N_r}} \mathbf{a}_{\mathrm{i}}(\mu)\phi_{\mathrm{i}}$

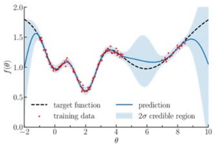
Nonlinear: $\mathbf{u} \approx \mathbf{f}(\mathbf{g})$ → $\mathbf{u}_{\mathrm{i}}(\mu_{\mathrm{k}})$  $\mathbf{u}^{\mathrm{r}}_{\mathrm{i}}(\mu_{\mathrm{k}})$

2. Approximation Step

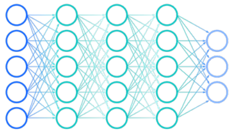
Linear:



RBF

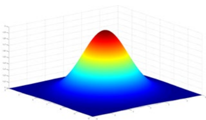


GPR

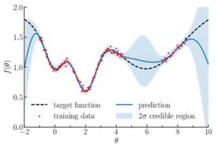


ANN

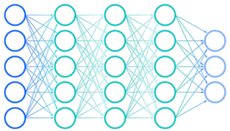
Nonlinear:



RBF



GPR



ANN

Non-Intrusive ROM workflow (Non-Linear Reduction + Data Parallelism)

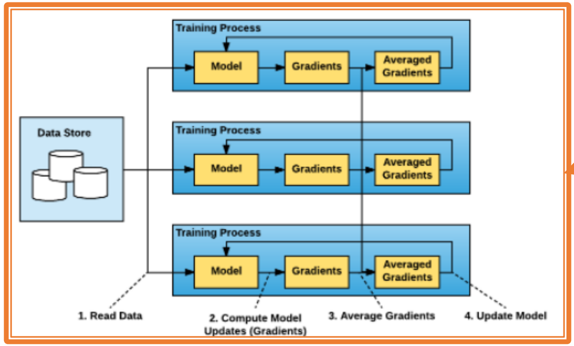
eFlows4HPC

EZyRB:

- ➔ Open source Python package developed by SISSA mathLab
- ➔ Completely data-driven, Non-intrusive ROM (linear & non-linear reduction)

EZyRB structure:

- ➔ **Database** class
- ➔ **Reduction** classes:
  1. POD (SVD, RSVD, SVD via correlation matrix)
  2. FFAE (PyTorch - sequential)
  3. FFAE (new class - PyEDDL - data parallelism - same API)
- ➔ **Approximation/interpolation** classes:
  1. Linear
  2. RBF
  3. GPR
  4. ANN
  5. KN
  6. RN
- ➔ **Error** methods
  1. loo_error
  2. Kfold error

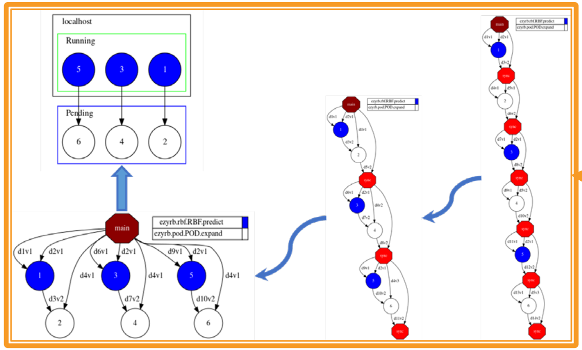


Non-Intrusive ROM workflow (Parallel Execution)

EZyRB + PyEDDL } COMPSs

To benefit from **distributed environments** we need to parallelize all EZyRB classes:

- ➔ **Database** class
- ➔ **Reduction** classes:
  1. POD (SVD, RSVD, SVD via correlation matrix)
  2. FFAE (PyTorch - sequential)
  3. FFAE (new class - PyEDDL - data parallelism - same API)
- ➔ **Approximation/interpolation** classes:
  1. Linear
  2. RBF
  3. GPR
  4. ANN
  5. KN
  6. RN
- ➔ **Error** methods
  1. loo_error
  2. Kfold error



The **simultaneous execution** using **PyCOMPSs** can enhance the process of **multiple predictions** or **error calculations** where the model has to be executed multiple times for different parameters.

THANK YOU!