



**Hewlett Packard**  
Enterprise



# **SMARTSIM: ONLINE ANALYTICS & ML FOR HPC SIMULATIONS**

---

Sam Partee, Machine Learning Engineer, HPE

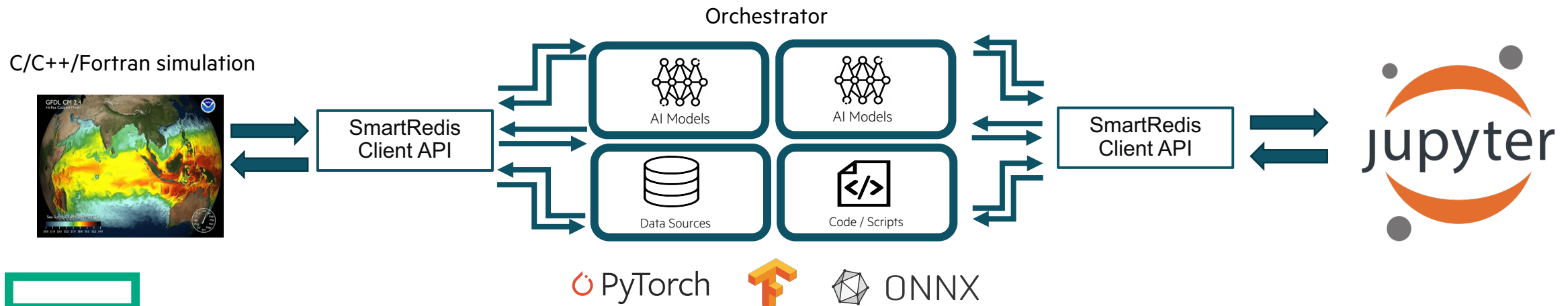
June 16<sup>th</sup>, 2021



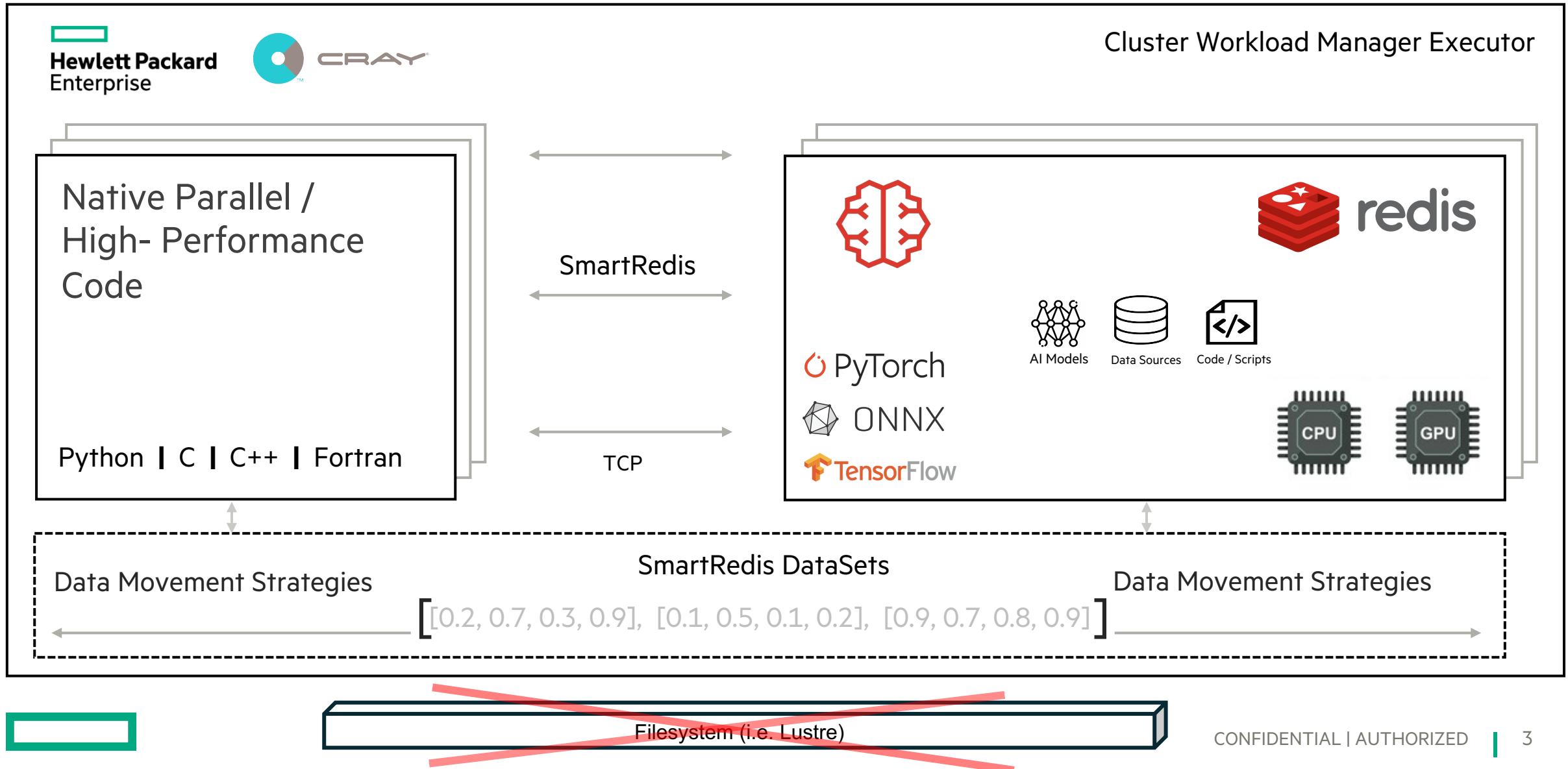
An open-source library dedicated to accelerating the convergence of AI, analytics, and data science with HPC simulation models. SmartSim can connect models written in Fortran, C, and C++ to the modern data science stack online.

### The SmartSim Stack enables scientists to:

- Embed calls to Machine Learning models into your Fortran/C/C++ simulation.
- Execute, monitor, and analyze Fortran/C/C++ simulations in a Jupyter notebook.
- Easily communicate numerical data (and metadata) between C, C++, Fortran, and Python (NumPy).
- Create, configure, and execute ensembles of simulations.
- Analyze data streamed from simulations in real time.
- Bypass the filesystem.



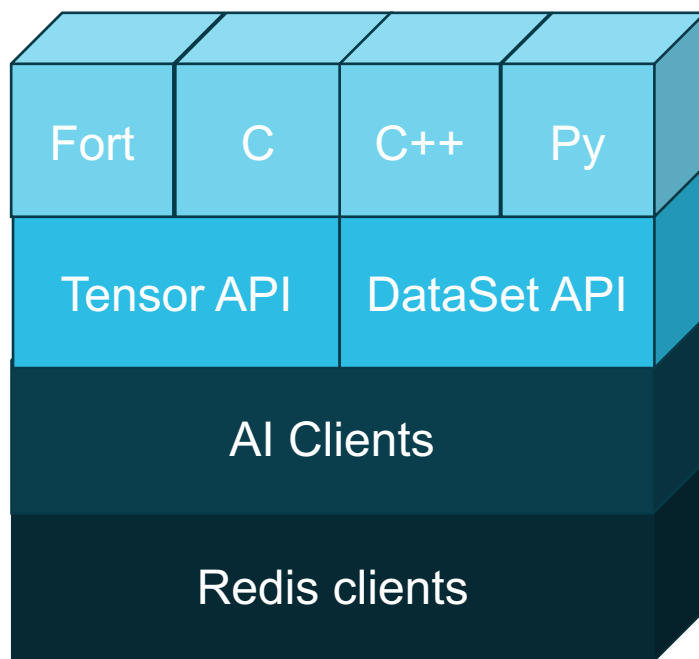
# HIGH-LEVEL ARCHITECTURE



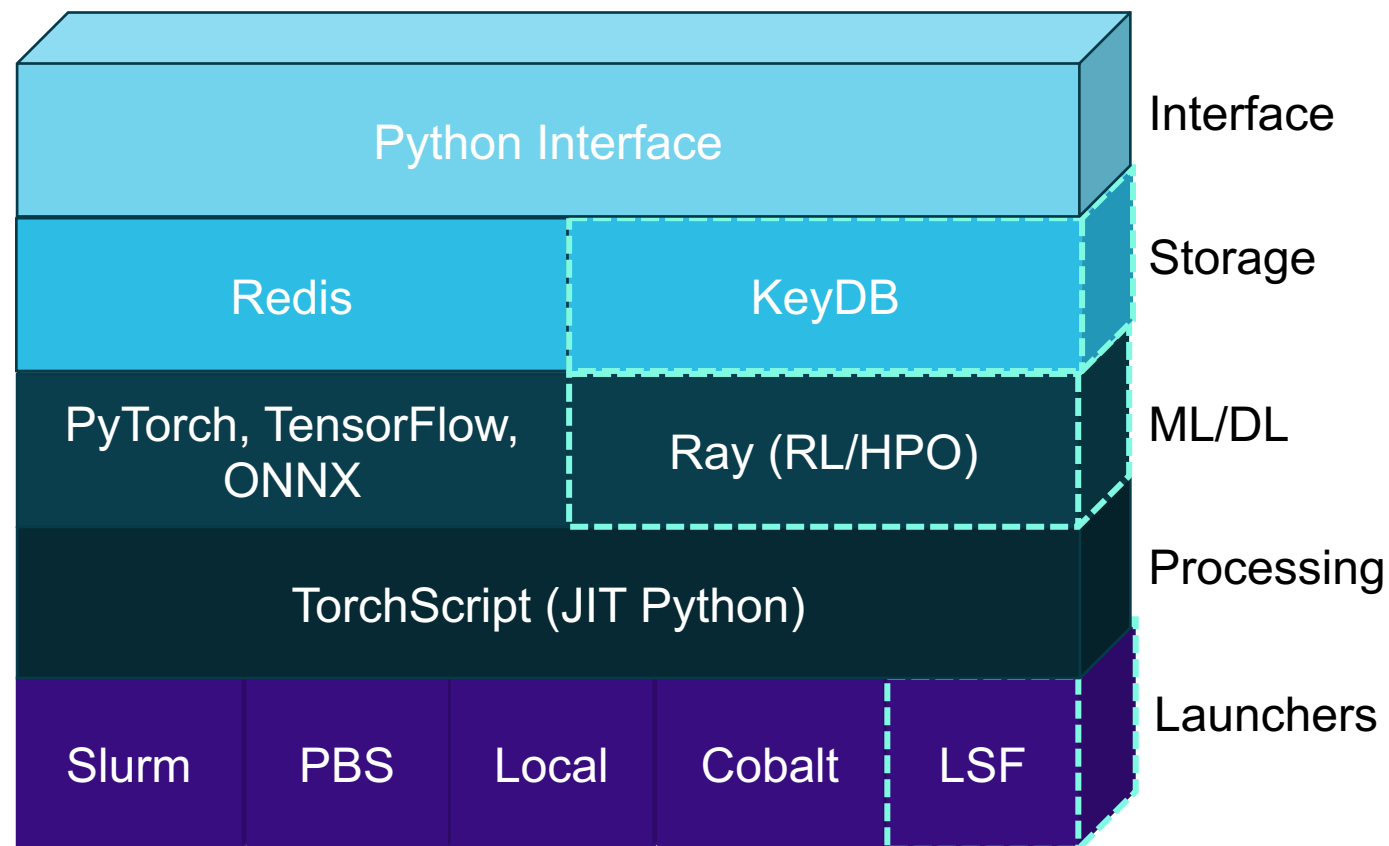
# SMARTSIM COMPONENTS

- SmartSim is composed of two libraries.
- The **infrastructure library** facilitates application and ML/DL deployment.
- The **client library** enables any application to communicate data and remotely execute ML/DL models and scripts from C++, C, Fortran, and Python.

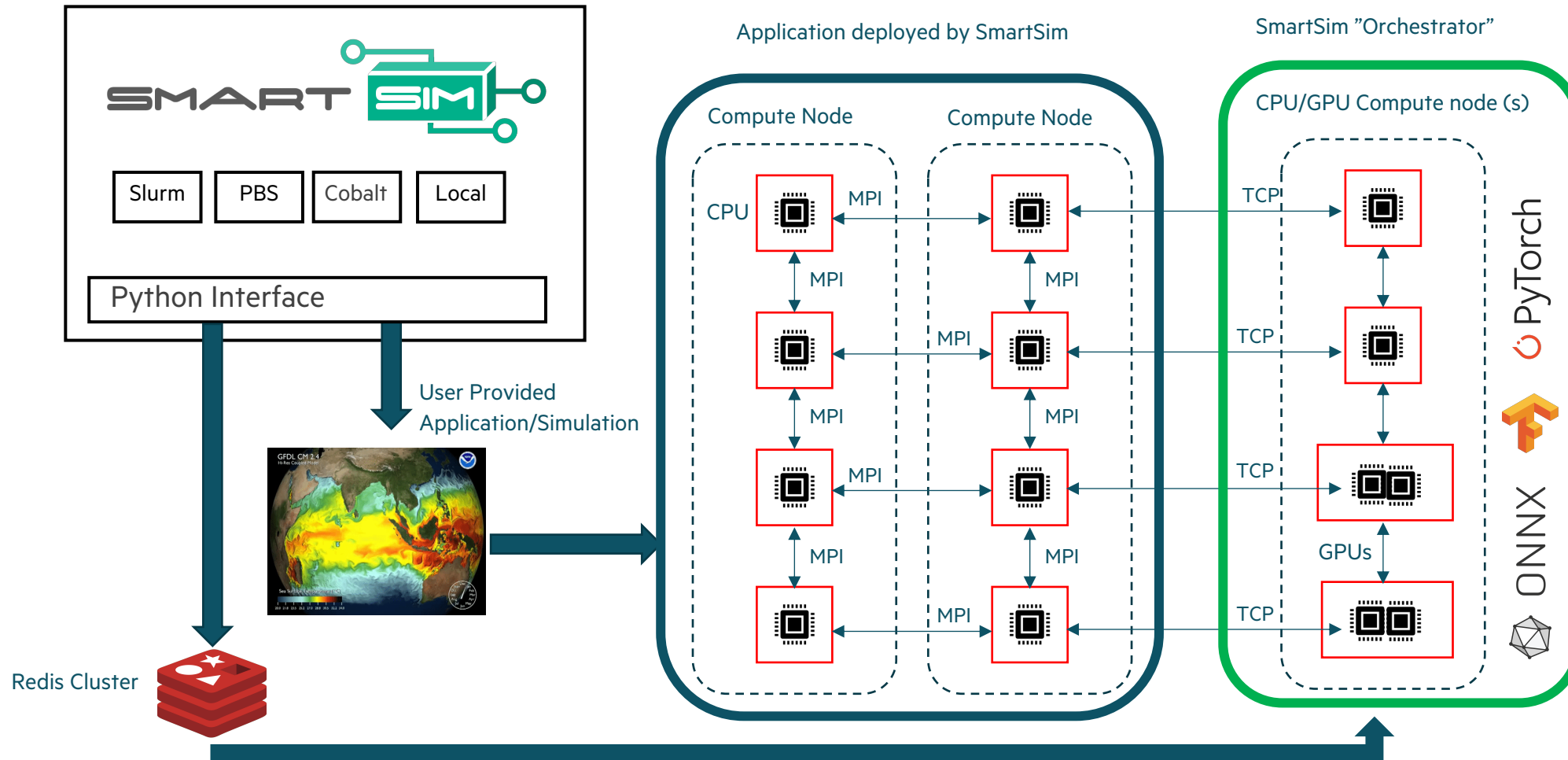
## Client Library (SmartRedis)



## Infrastructure Library



# SMARTSIM INFRASTRUCTURE LIBRARY



**Abstract**



# SMARTREDIS API

## Tensor data types

- Double
- Float
- Int8
- Int16
- Int32
- Int64
- UInt8
- UInt16

## Metadata types

- Double
- Float
- Int64
- Int32
- UInt64
- UInt32
- String

Same API in Python, Fortran, C, C++



Client Data Structure	API	Description
Tensor	Client.put_tensor()	Set a tensor
	Client.get_tensor()	Retrieve a tensor
	Client.unpack_tensor()	Fill user-provided memory space with retrieved tensor
Model	Client.set_model()	Store and distribute ML/DL model
	Client.get_model()	Retrieve model
	Client.run_model()	Execute a ML/DL model on some stored data
Script	Client.set_script()	Store and distribute TorchScript script
	Client.get_script()	Retrieve a TorchScript script
	Client.run_script()	Run a TorchScript function within a script on some data

DataSet actions	API	Description
Construct	DataSet.add_tensor()	Add a tensor to the DataSet
	DataSet.add_meta_scalar()	Add a metadata value to a scalar field
	DataSet.add_string_scalar()	Add a metadata value to a string field
Store	Client.put_dataset()	Set a DataSet
	Client.get_dataset()	Retrieve a DataSet
Inspect	DataSet.get_tensor()	Get a tensor from the DataSet
	DataSet.unpack_tensor()	Fills user-provided memory space with tensor
	DataSet.get_meta_scalars()	Get values associated with scalar metadata field
	DataSet.get_meta_strings()	Get values associated with string metadata field



**Hewlett Packard**  
Enterprise

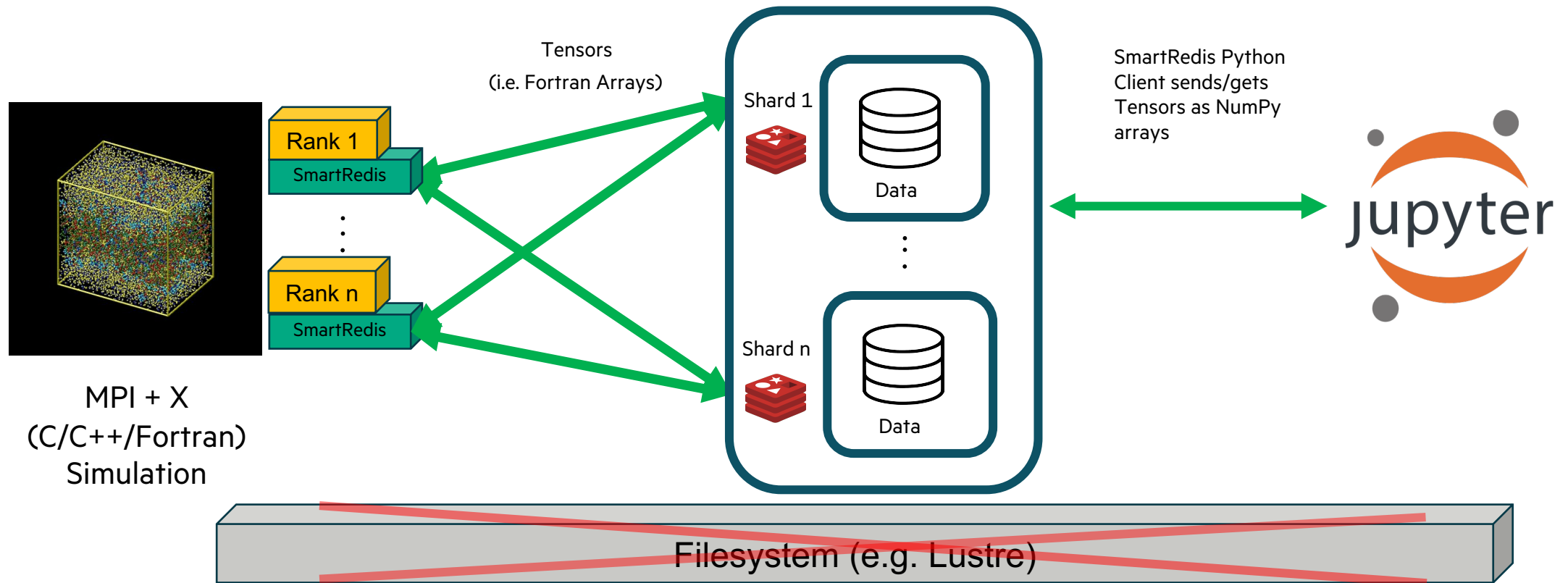
# **WHAT CAN I DO WITH SMARTSIM?**





# ONLINE ANALYSIS

- Stream data from C/C++/Fortran simulations for analysis, and visualization in real time.
- Arrays transformations between languages handled by SmartRedis (Fortran -> NumPy)
- Clients are lightweight: single static library
- No reading/writing to slow shared filesystems



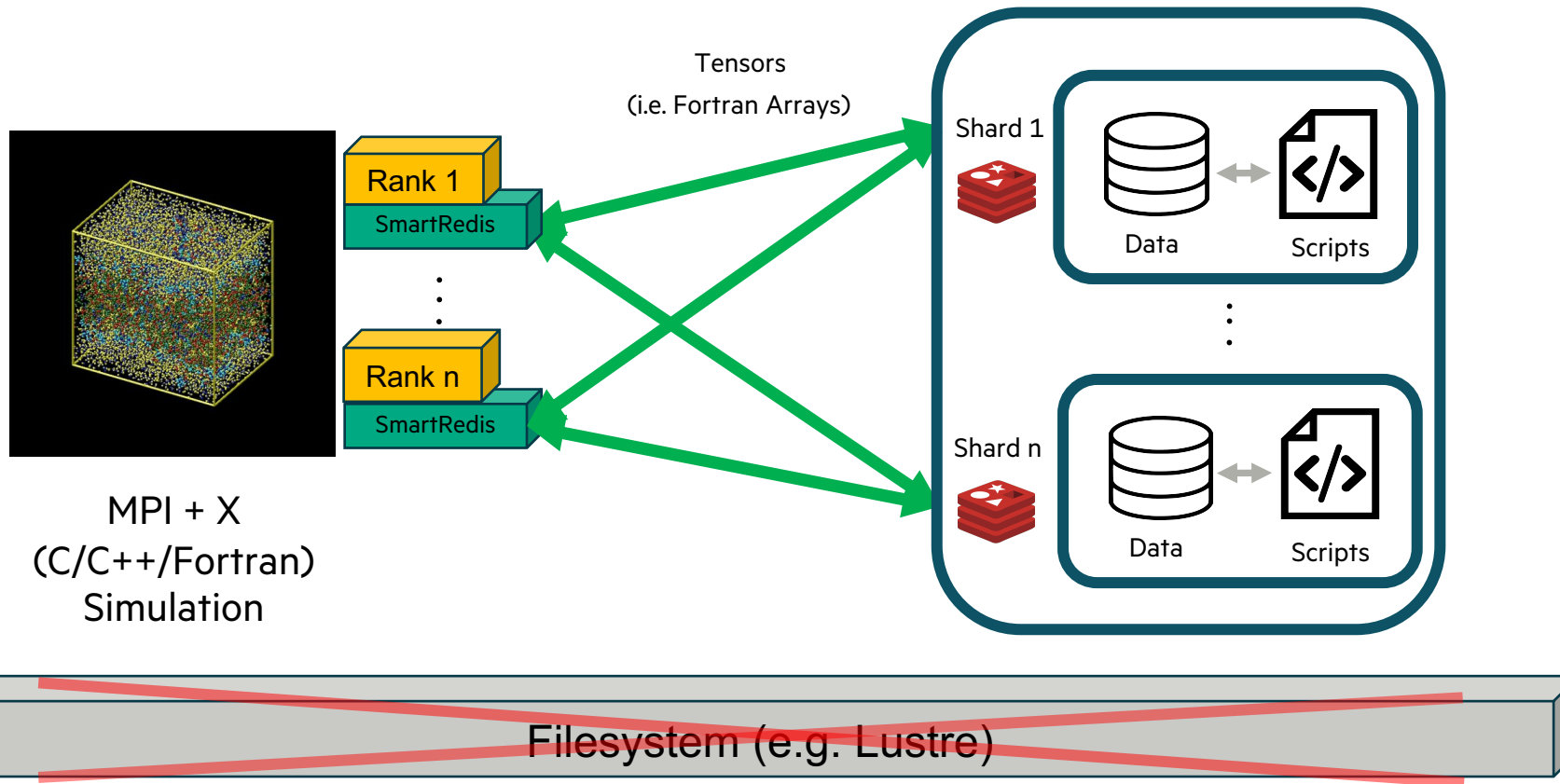
# ONLINE PROCESSING

- Same SmartRedis Clients can call TorchScript scripts (JIT-traced Python) stored inside database
- Can be set from any language and called from any language (i.e. set from Python, called from Fortran)
- Simple example calculates Singular Value Decomposition (SVD) of tensors streamed to database

## Serial SVD example

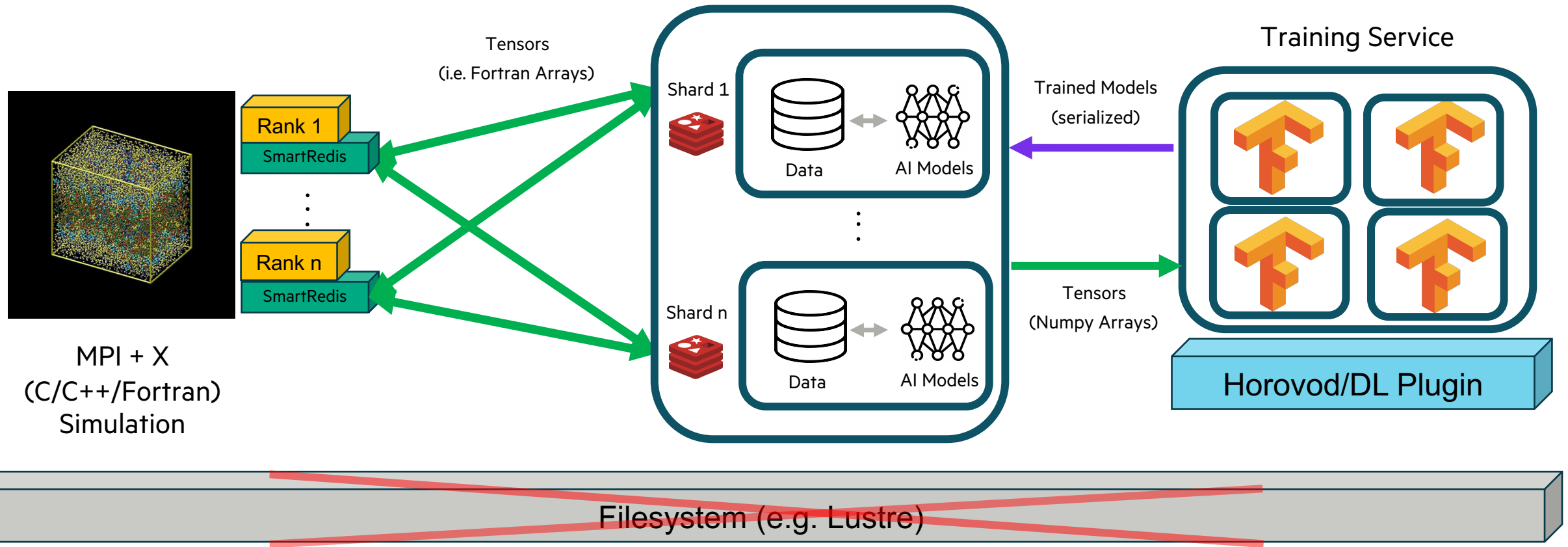
red=Infrastructure blue=Client

```
1 import numpy as np
2 from smartsim import Experiment
3 from smartsim.database import Orchestrator
4 from smartredis import Client
5
6 def calc_svd(input_tensor):
7     return input_tensor.svd()
8
9 exp = Experiment("svd", launcher="local")
10 db = Orchestrator()
11 exp.start(db)
12
13 client = Client(address="127.0.0.1:6379",
14                 cluster=False)
15 tensor = np.random.randint(0, 100,
16                             size=(30, 10, 2))
17 client.put_tensor("input", tensor)
18 client.set_function("svd", calc_svd)
19 client.run_script("svd", "calc_svd",
20                  "input", ["U", "S", "V"])
21 U = client.get_tensor("U")
22 S = client.get_tensor("S")
23 V = client.get_tensor("V")
24
25 print(f"U: {U}, S: {S}, V: {V}")
26 exp.stop(db)
```



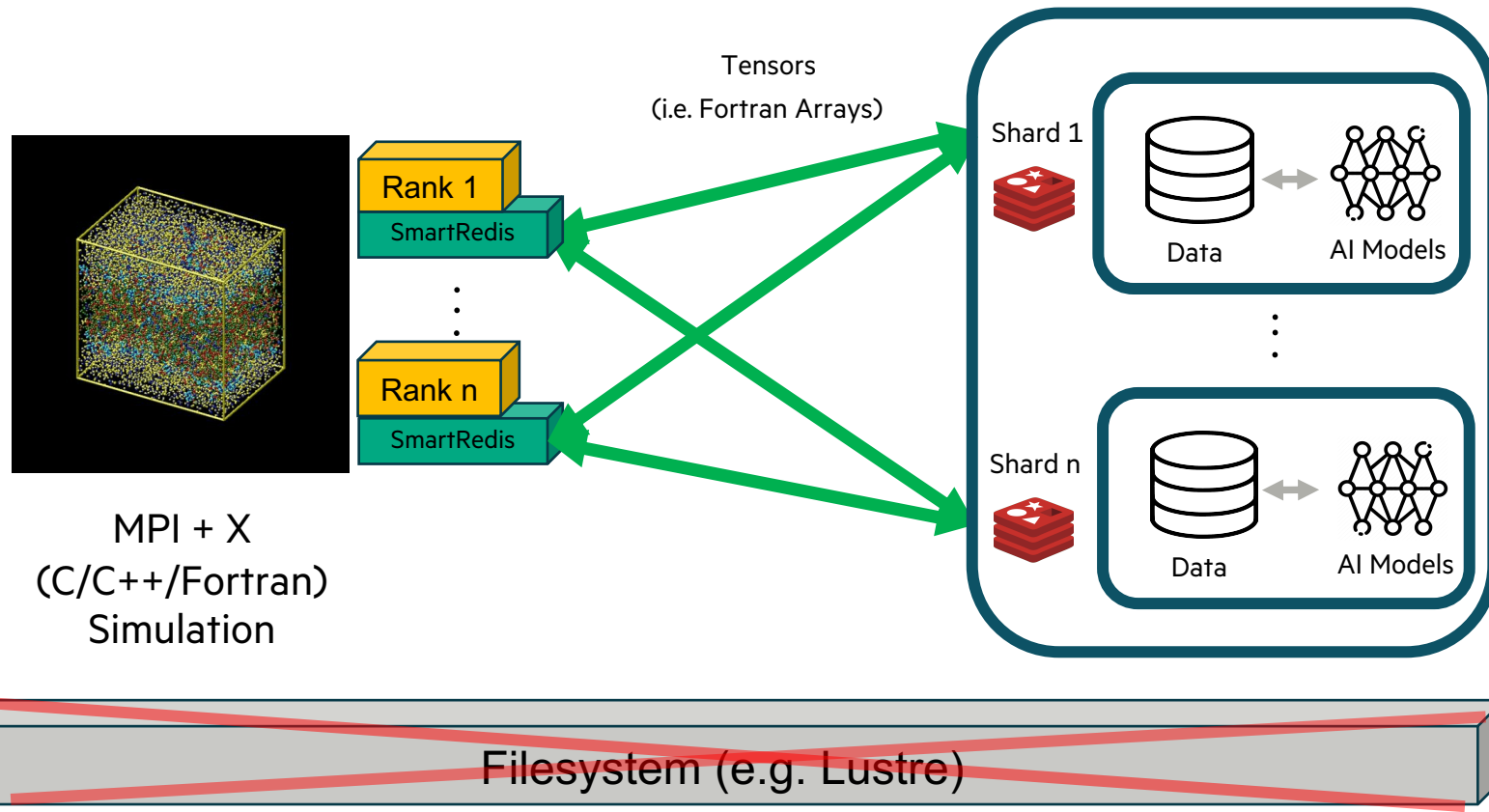
# ONLINE TRAINING

- SmartRedis Client can be used inside DataLoaders for TensorFlow and PyTorch to perform Stream Training
- Trained models can be checkpointed, saved and later sent to the database for online inference.
- Can be set from any language and called from any language (i.e. model set from Python, called from Fortran)



# ONLINE INFERENCE

- Call TensorFlow, PyTorch, ONNX (scikit-learn, spark, etc) models stored inside the database.
- Result of inference stored at new key for retrieval when needed. Minimizes data movement.
- Multiple levels of batching – database auto-batches to optimize GPU memory usage



## Serial Torch Inference example

red=Infrastructure blue=Client

```

1 import io
2 import torch
3 import numpy as np
4 import torch.nn as nn
5 from smartsim import Experiment
6 from smartsim.database import Orchestrator
7 from smartredis import Client
8
9 class Net(nn.Module):
10     def __init__(self):
11         super(Net, self).__init__()
12         self.conv = nn.Conv2d(1, 1, 3)
13
14     def forward(self, x):
15         return self.conv(x)
16
17 exp = Experiment("svd", launcher="local")
18 db = Orchestrator()
19 exp.start(db)
20
21 client = Client(address="127.0.0.1:6379",
22                 cluster=False)
23 n = Net()
24 example_forward_input = torch.rand(1, 1, 3, 3)
25 module = torch.jit.trace(n, example_forward_input)
26 model_buffer = io.BytesIO()
27 torch.jit.save(module, model_buffer)
28
29 client.set_model("cnn", model_buffer.getvalue(),
30                 "TORCH", device="CPU")
31 client.put_tensor("input", example_forward_input.numpy())
32 client.run_model("cnn", inputs=["input"], outputs=["output"])
33
34 output = client.get_tensor("output")
35 print(f"Prediction: {output}")
36
37 exp.stop(db)
  
```

## LATEST RESEARCH PAPER

<https://arxiv.org/pdf/2104.09355.pdf>

- Code for Paper:  
[https://github.com/CrayLabs/NCAR\\_ML\\_EKE](https://github.com/CrayLabs/NCAR_ML_EKE)
- Models and Datasets for Paper:  
<https://zenodo.org/record/4682270#.YMkM525IDUJ>
- Seminar given at NCAR:  
<https://www.youtube.com/watch?v=2e-5j427AS0>

## Using Machine Learning at Scale in HPC Simulations with SmartSim:

### An Application to Ocean Climate Modeling

Sam Partee  
Hewlett Packard Enterprise  
Seattle, WA  
spartee@hpe.com  
<https://orcid.org/0000-0001-6005-5116>

Matthew Ellis  
Hewlett Packard Enterprise  
Seattle, WA  
matthew.ellis@hpe.com  
<https://orcid.org/0000-0002-5782-5447>

Alessandro Rigazzi  
Hewlett Packard Enterprise  
Switzerland  
alessandro.rigazzi@hpe.com  
<https://orcid.org/0000-0003-2132-7726>

Scott Bachman  
National Center for Atmospheric Research  
Boulder, CO  
bachman@ucar.edu  
<https://orcid.org/0000-0002-6479-4300>

Gustavo Marques  
National Center for Atmospheric Research  
Boulder, CO  
gmarques@ucar.edu  
<https://orcid.org/0000-0001-7238-0290>

Andrew Shao  
University of Victoria  
Victoria, CA  
aeshao@uvic.ca  
<https://orcid.org/0000-0003-3658-512X>

Benjamin Robbins  
Hewlett Packard Enterprise  
Seattle, WA  
benjamin.robbs@hpe.com

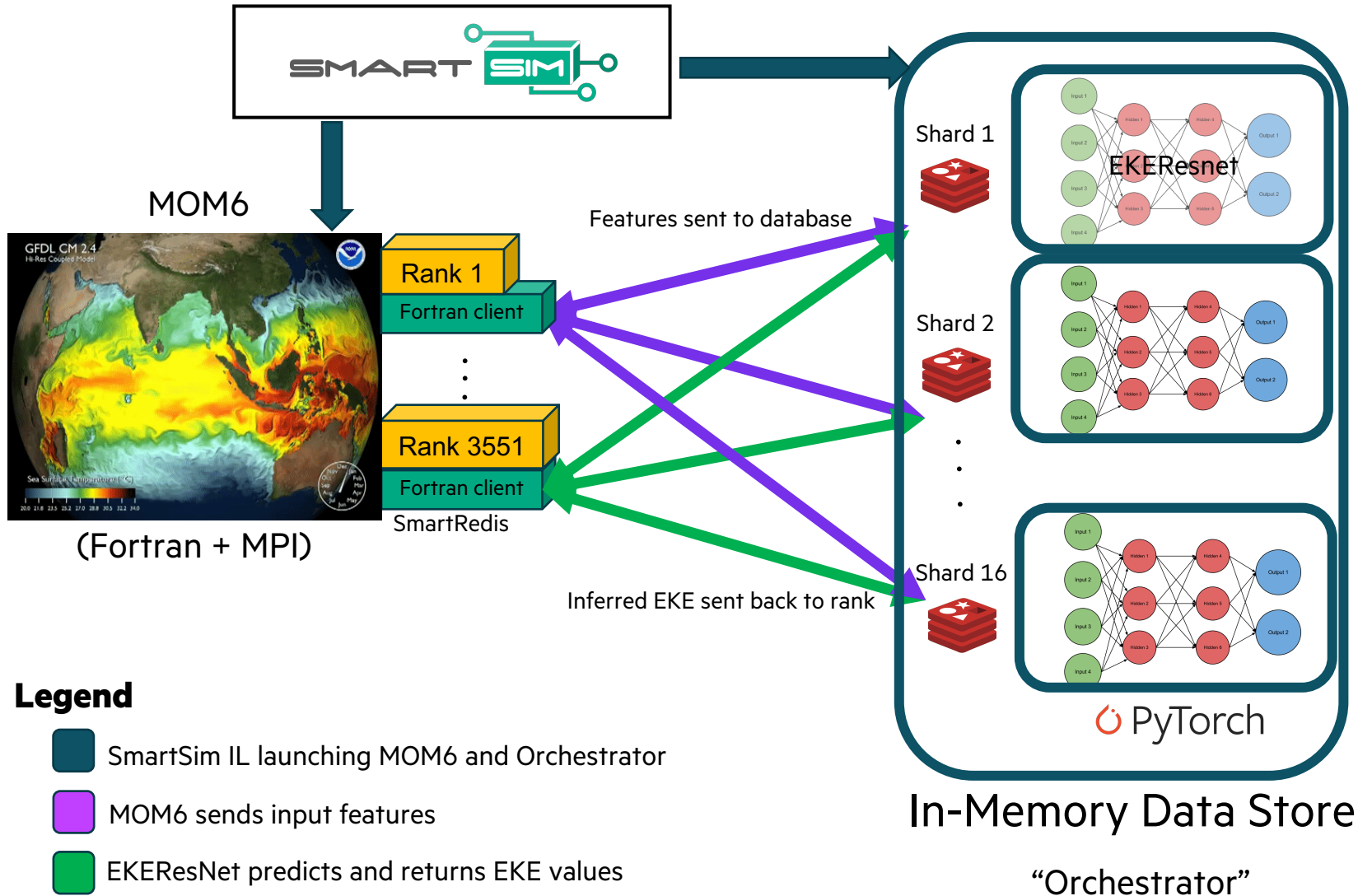
**Abstract**—We demonstrate the first climate-scale, numerical ocean simulations improved through distributed, online inference of Deep Neural Networks (DNN) using SmartSim. SmartSim is a library dedicated to enabling online analysis and Machine Learning (ML) for traditional HPC simulations. In this paper, we detail the SmartSim architecture and provide benchmarks including online inference with a shared ML model on heterogeneous HPC systems. We demonstrate the capability of SmartSim by using it to run a 12-member ensemble of global-scale, high-resolution ocean simulations, each spanning 19 compute nodes, all communicating with the same ML architecture at each simulation timestep. In total, 970 billion inferences are collectively served by running the ensemble for a total of 120 simulated years. Finally, we show our solution is stable over the full duration of the model integrations, and that the inclusion of machine learning has minimal impact on the simulation runtimes.<sup>1</sup>

dependence on file input/output (I/O), and large variance in compute resource requirements for scientific applications makes it difficult to perform analysis, training, and inference with most ML and data analytics packages at the scale needed for HPC numerical simulations.

On its surface, the problem of being able to interface HPC applications with ML libraries is one of language interoperability and software interface design. However, for the full convergence of these two disparate paradigms, the true difficulty (and opportunity) in bridging these workloads needs to be reformulated in terms of data exchange. That is, how is data passed between a simulation and ML model at scale while making efficient use of heterogeneous computational

Kiv:2104.09355v1 [cs.CE] 13 Apr 2021

# COLLABORATION WITH NCAR

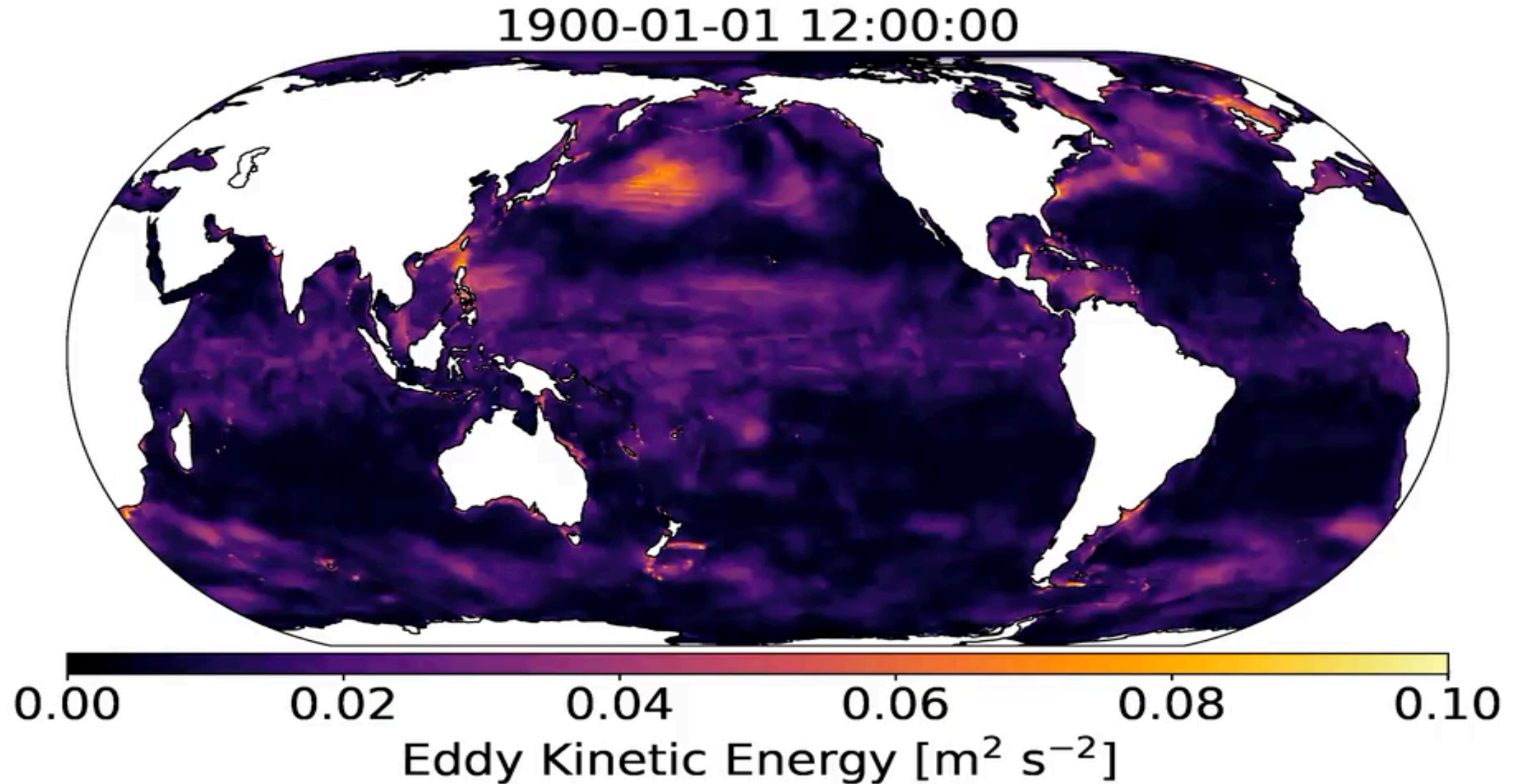


## Replacing the MEKE Parameterization

- Goal:** Augment MOM6 MEKE parameterization with Machine Learning (ML) surrogate model
- Augmented Simulation:
  - Modular Ocean Model 6**
  - GFDL's OM4 1/4 configuration (Adcroft et al., 2019)
- Training Data simulation
  - 1/10 ° nominal resolution (tx0.1, same grid used in POP)
  - Ocean (MOM6) and sea ice (CICE5) components
- AI Model
  - Modified ResNet – **EKEResnet**
- Input Features
  - $MKE_{sfc}$ ,  $Slope_z$ ,  $Rd_{dx_z}$ ,  $Rel_{vort_{sfc}}$
- Inference
  - 16 GPUs – 16 copies of EKEResnet
  - MOM6 – 3551 ranks per model



# Online surface EKE prediction via SmartSim: GFDL OM4 1/4



# PAPER: COMPUTATIONAL SETUP



## Experiment in the Paper

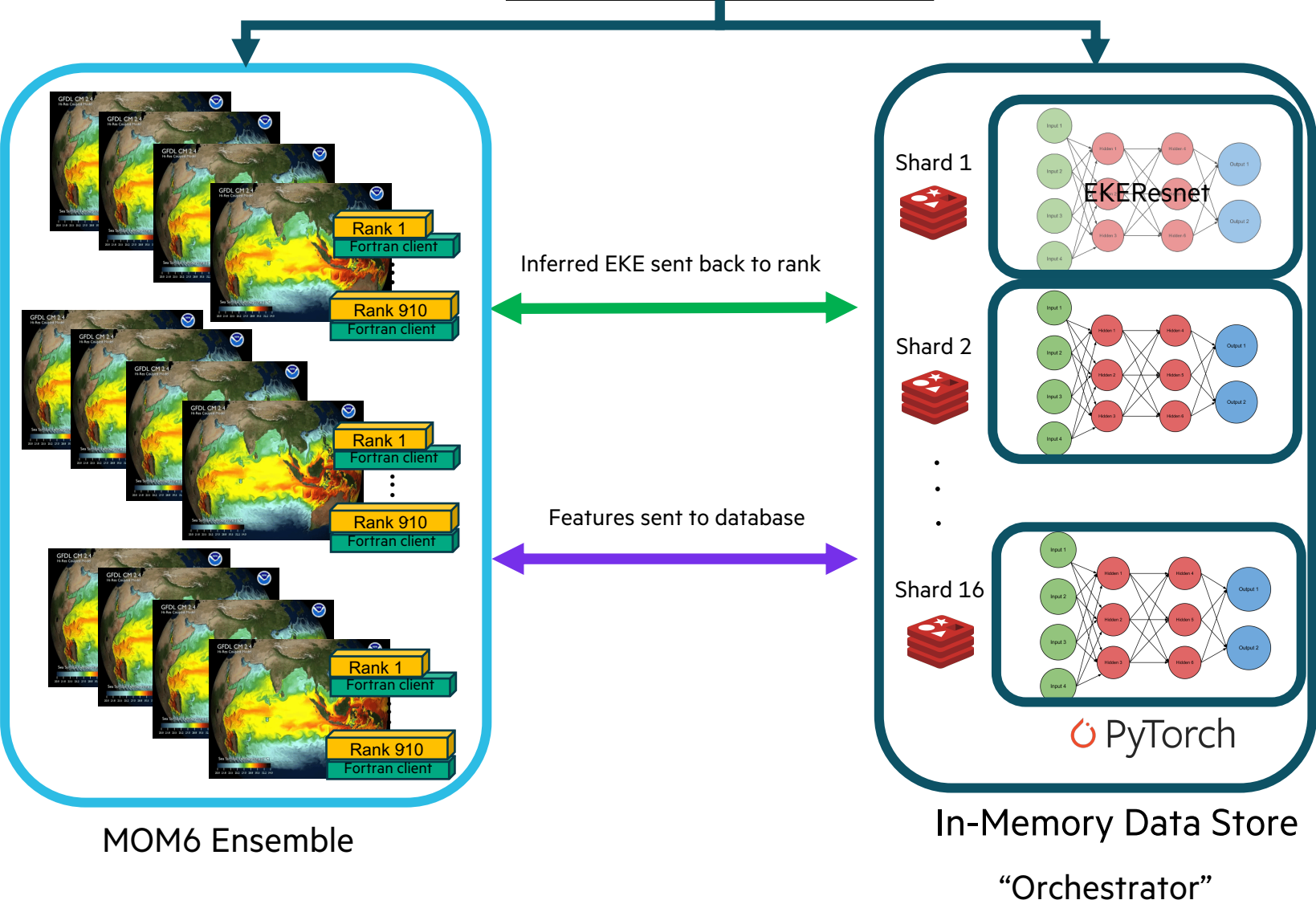
- MOM6+PyTorch ensemble orchestrated via SmartSim
- 12 ensemble members – each member running on 19 nodes on Cray XC
- 10 year of simulation time – ~7 hours of computation time

970 billion inferences throughout 10 simulation years  
~10 trillion for full simulation

Negligible slowdown of the simulations compared previous SOTA (< 1.5%) despite adding 11 ensemble members to the same machine learning infrastructure.

## Legend

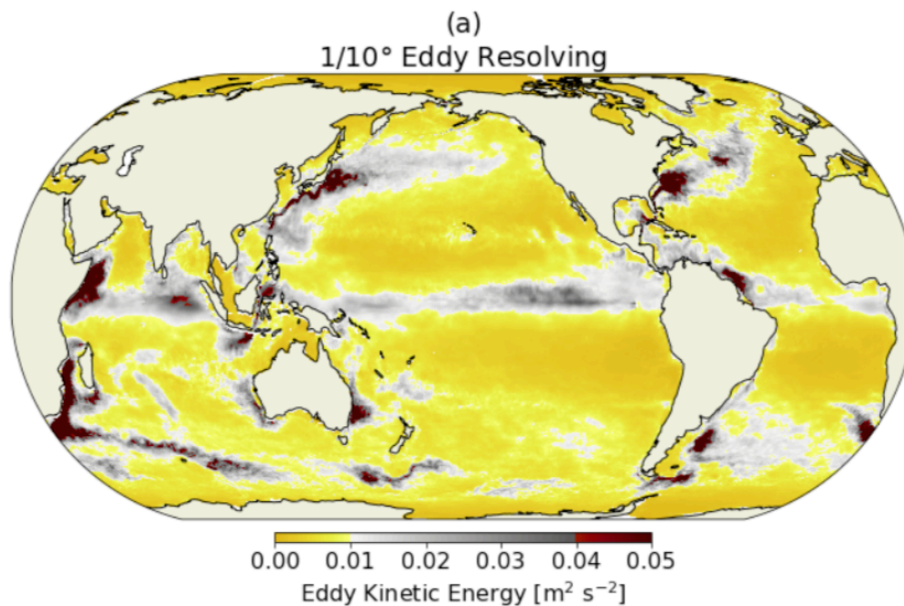
- SmartSim IL launching MOM6 and Orchestrator
- MOM6 sends input features
- EKEResNet predicts and returns EKE values



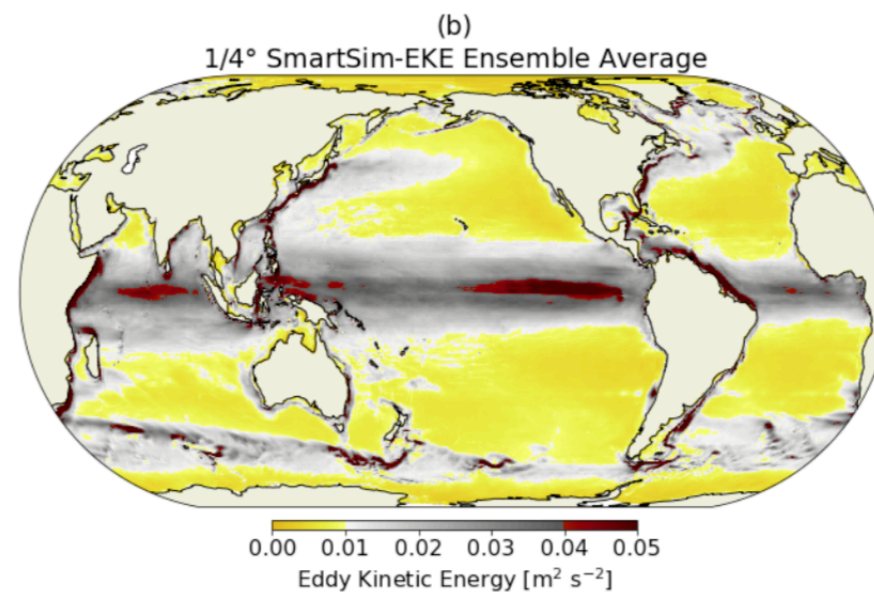


# RESULTS

## “Truth” data



## With SmartSim



## Previous SOTA

(Jansen et al 2015)

<https://www.sciencedirect.com/science/article/abs/pii/S1463500315001341>

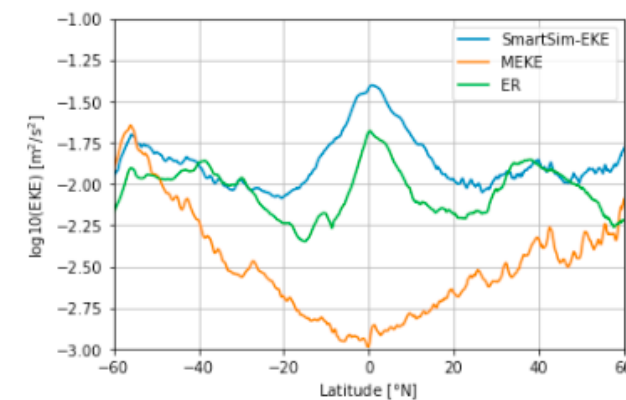
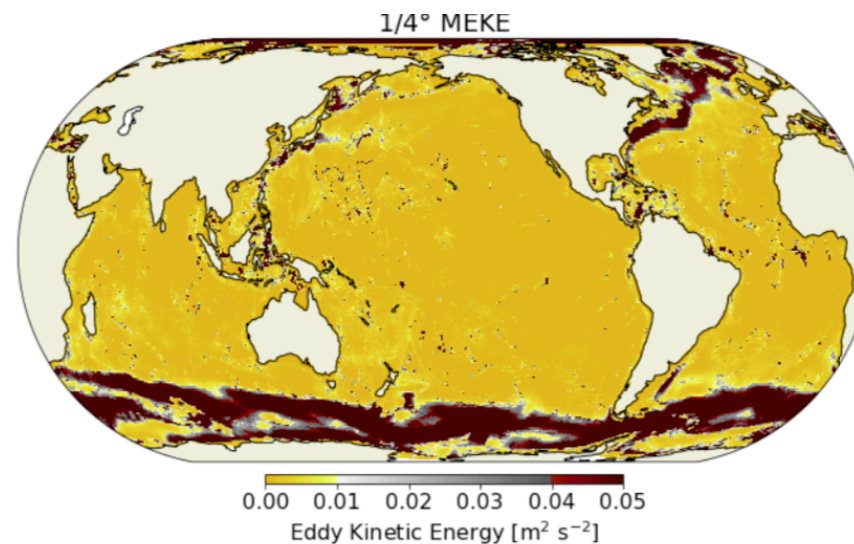


Fig. 7. Zonally averaged EKE (on a log10 scale) as a function of latitude from the ER, SmartSim-EKE, and MEKE.

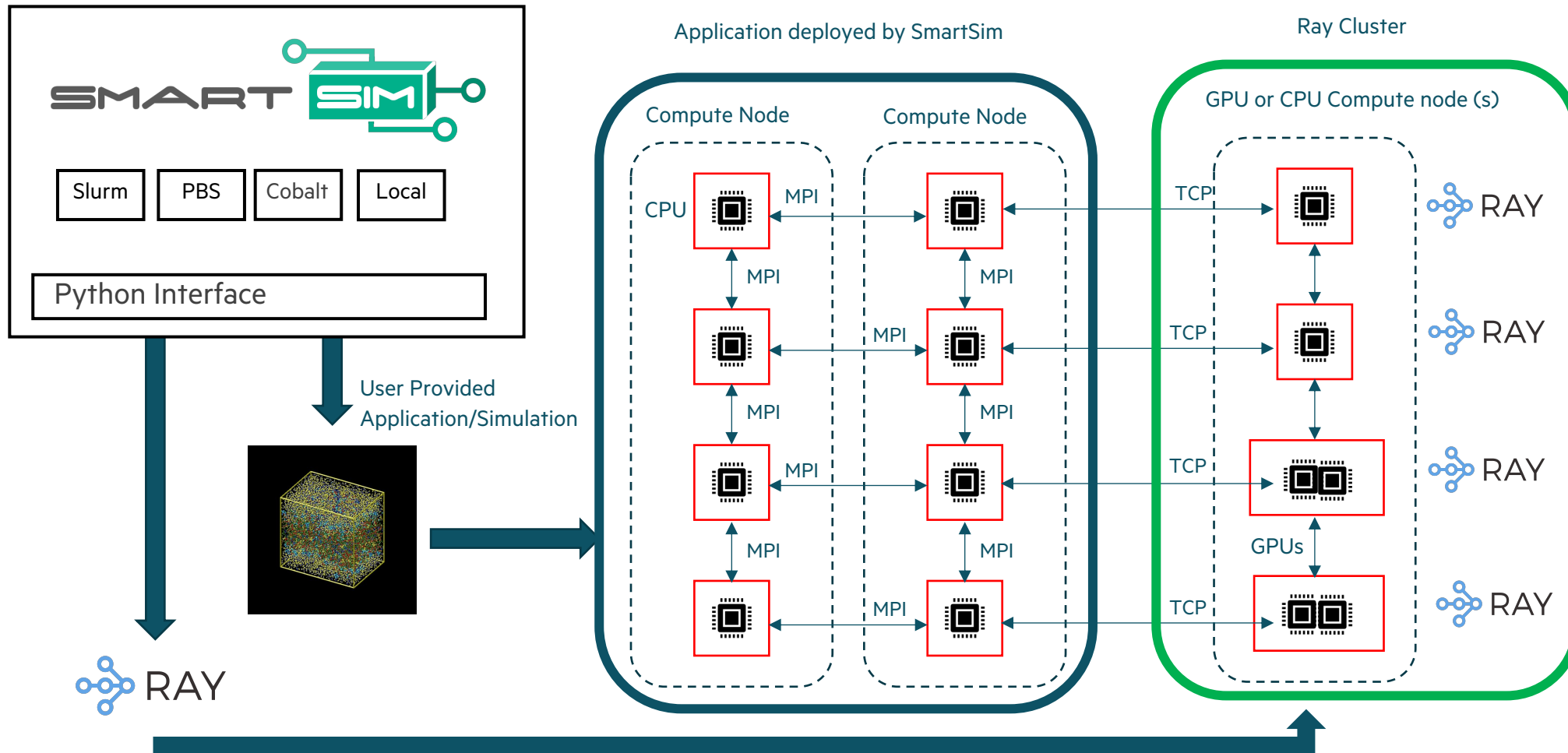


**Hewlett Packard**  
Enterprise

# **CURRENT RESEARCH AND DEVELOPMENT**

---

# SMARTSIM REINFORCEMENT LEARNING WITH RAY



Functionality currently in progress, let us know if you want to try it!

# SMARTREDIS WITH XARRAY

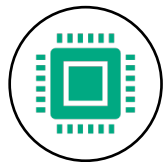
## WHY?

### Multi-language access to `xarray.DataArray`



Scientific applications written in C++, C, and Fortran can send data to analysis or visualization tools using the `xarray` data format and `xarray` data can be streamed back into those simulations.

### Use of `xarray.DataArray` in TensorFlow, PyTorch, and ONNX models



With the transformation of `xarray.DataArray` into the SmartRedis data format, the `DataArray` data can be used in machine learning models hosted in the Orchestrator (Redis)

### Distributed data storage with Redis cluster



SmartRedis has been optimized for Redis cluster configurations, and as a result, SmartRedis can efficiently store `xarray` data in large clusters for HPC applications.

## HOW?

### Idea: Extend SmartRedis API to utilize Xarray

- The `SmartRedis::DataSet` data structure stores tensors alongside metadata
- `SmartRedis::DataSet` can be expanded to support the storage and retrieval of `xarray.DataArray` objects
- `DataArray` attributes, coords, and dims are stored as metadata in the `SmartRedis::DataSet`
- The `DataArray` data is stored as a standard `SmartRedis::DataSet` tensor
- Upon retrieval, the `SmartRedis::DataSet` metadata fields are used to create an `xarray.DataArray` object for the user

This functionality can be embedded in simple functions:

- `SmartRedis::Client.put_tensor(key, xarray.DataArray)`
- `SmartRedis::Client.get_xarray(key)`

# WHERE CAN I FIND MORE?

## Other SmartSim Resources

- SmartSim Repository: <https://github.com/CrayLabs/SmartSim>
- SmartRedis Repository: <https://github.com/CrayLabs/SmartRedis>
- SmartSim Scaling Repository: <https://github.com/CrayLabs/SmartSim-Scaling>
- Tutorial: [https://www.craylabs.org/docs/tutorials/01\\_getting\\_started/01\\_getting\\_started.html](https://www.craylabs.org/docs/tutorials/01_getting_started/01_getting_started.html)
- Repository for Paper: [https://github.com/CrayLabs/NCAR\\_ML\\_EKE](https://github.com/CrayLabs/NCAR_ML_EKE)
- Documentation: <https://www.craylabs.org/build/html/overview.html>
- Seminar given at NCAR: <https://www.youtube.com/watch?v=2e-5j427AS0>
- RedisConf 2021 Presentation: <https://www.youtube.com/watch?v=-djLz8HgM-Q>
- PyTorch Ecosystem Day Poster: <https://assets.pytorch.org/pted2021/posters/J8.png>
- PyPI SmartSim: <https://pypi.org/project/smartsim/>
- PyPI SmartRedis: <https://pypi.org/project/smartredis/>

PyTorch  
ECOSYSTEM  
DAY

 redisconf2021

