On-line training of Deep Surrogates models

Numerical Analysis School EDF-INRIA-CEA

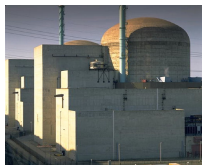
Saclay, 19 june 2025

Bruno.raffin@inria.fr
Alejandro.Ribes@edf.fr

GENERAL CONTEXT:
EDF – ELECTRICITÉ DE FRANCE

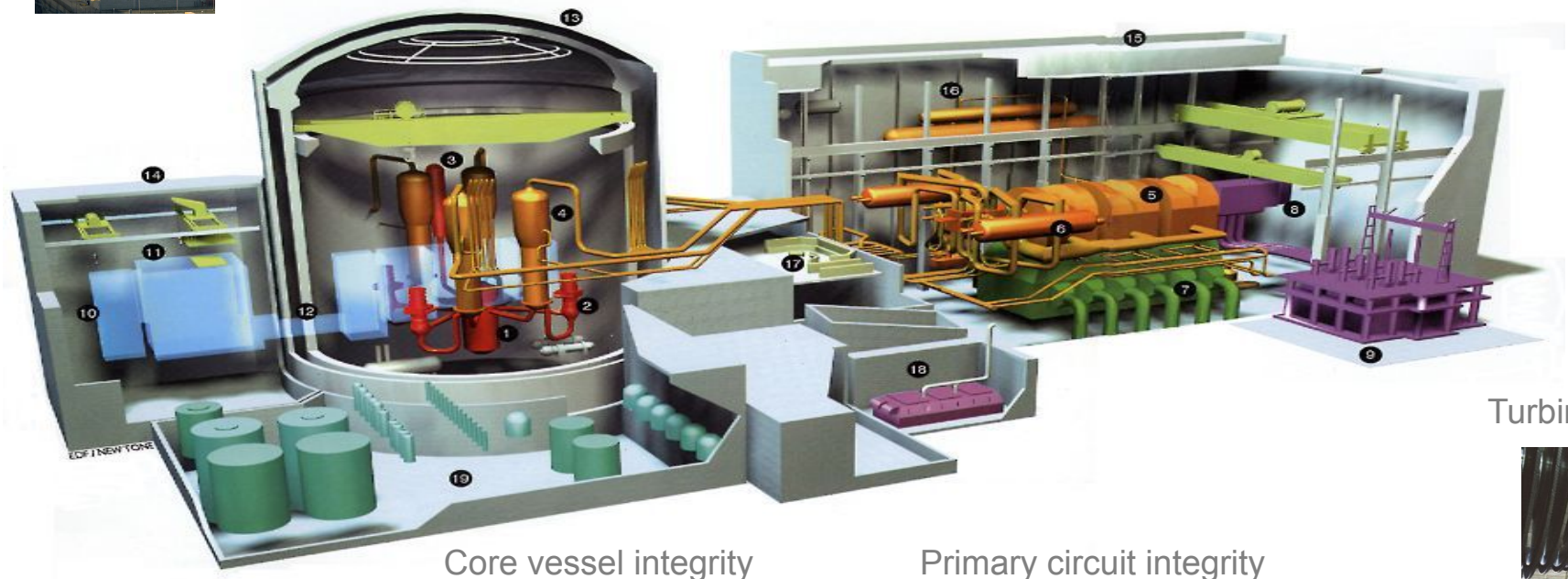
- **Electric utility company**
- **58 active nuclear reactors in France (all PWRs)**
- **EDF Energy in UK**
  - 8 nuclear power stations (7 AGR)
- **EDF R&D**
  - About 2,000 researchers
  - Saclay
  - Several top500 supercomputers
    - Currently 3 clusters
  - Extensive use of numerical simulation

GENERAL CONTEXT: NUMERICAL SIMULATIONS INVOLVING MECHANICS

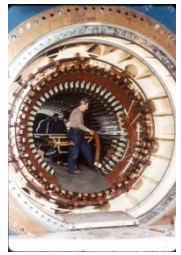


Containment building Integrity

Seismic analysis

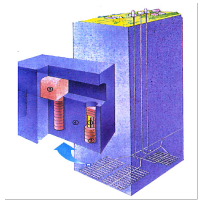


Alternator behaviour



Storage

Turbine behaviour



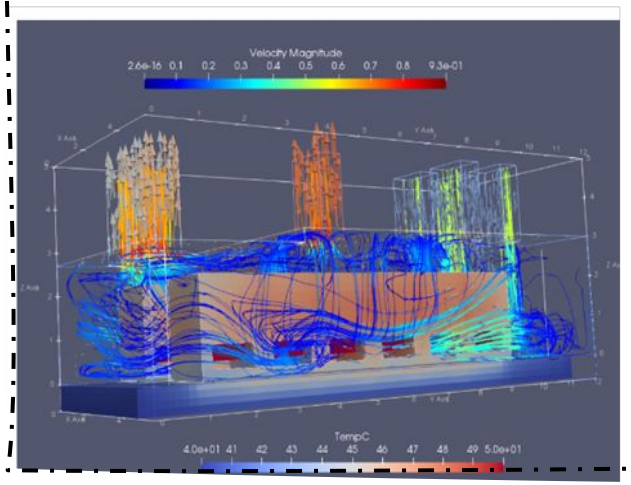
Core vessel integrity

Primary circuit integrity

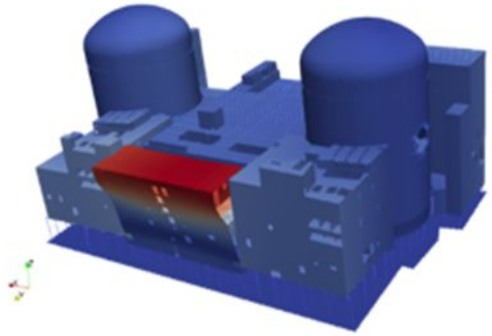
An arsenal of home-tailored numerical simulation softwares

CODE saturne

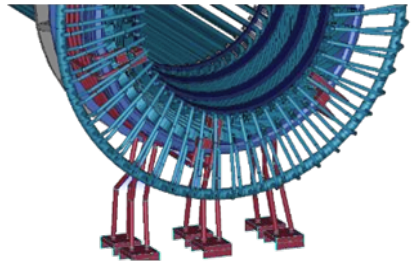
NEPTUNE_CFD



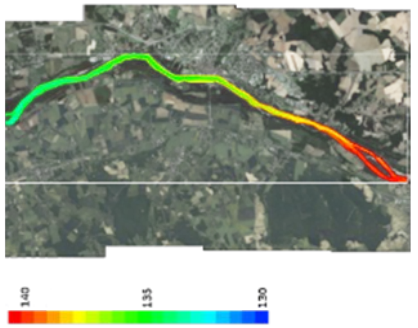
CODE aster



code_Carmel

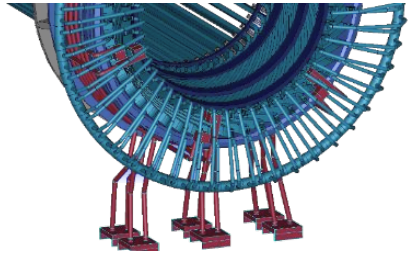


open TELEMAC

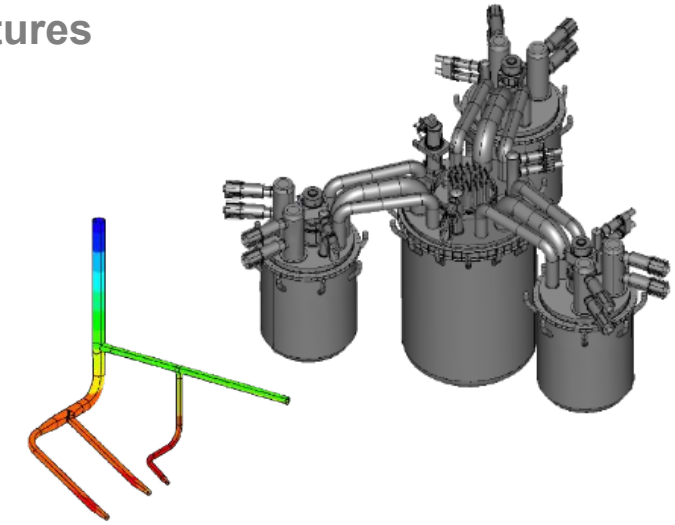


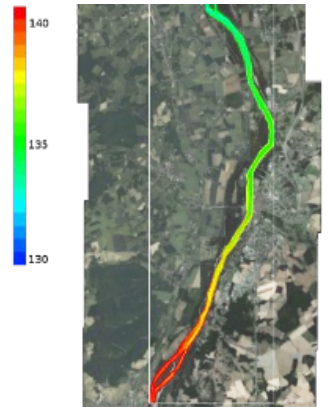
SALOME

GENERAL CONTEXT



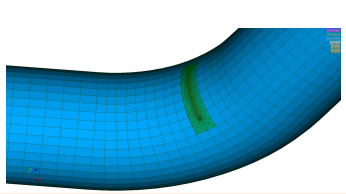
- **Numerical modelling of EDF components and structures**
  - Structural mechanics (*Code_Aster*)
  - Thermohydraulics (*Code_Saturne*, *NEPTUNE_CFD*)
  - Electromagnetism (*Code_CARMEL3D*)
  - Neutronics (*ANDROMEDE*)
  - Surface hydraulics (*TELEMAC-MASCARET*)

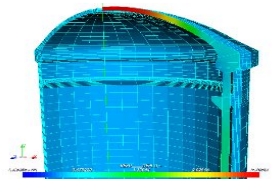




- **All these physics domains require generic functions for numerical simulations**

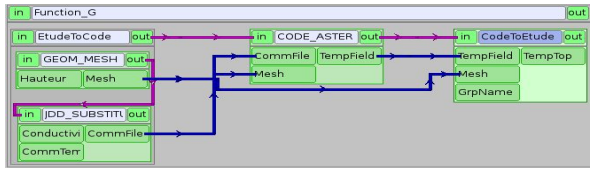
3D Modelling
(CAD, meshing, vizualisation)





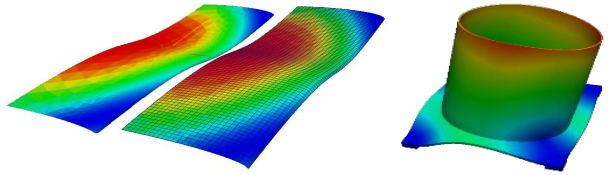
+

Computation scheduling
(workflow, distribution)



+

Complex data processing
(fields, matrix, etc)



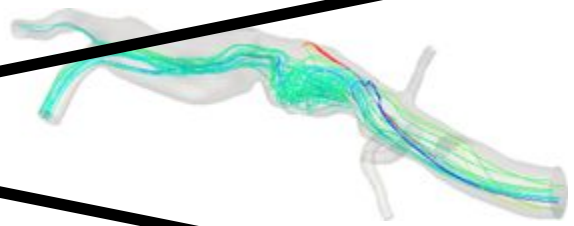
= SALOME Platform

IN-SITU VISUALIZATION

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \mu\nabla^2\vec{v}$$

Simulation Code

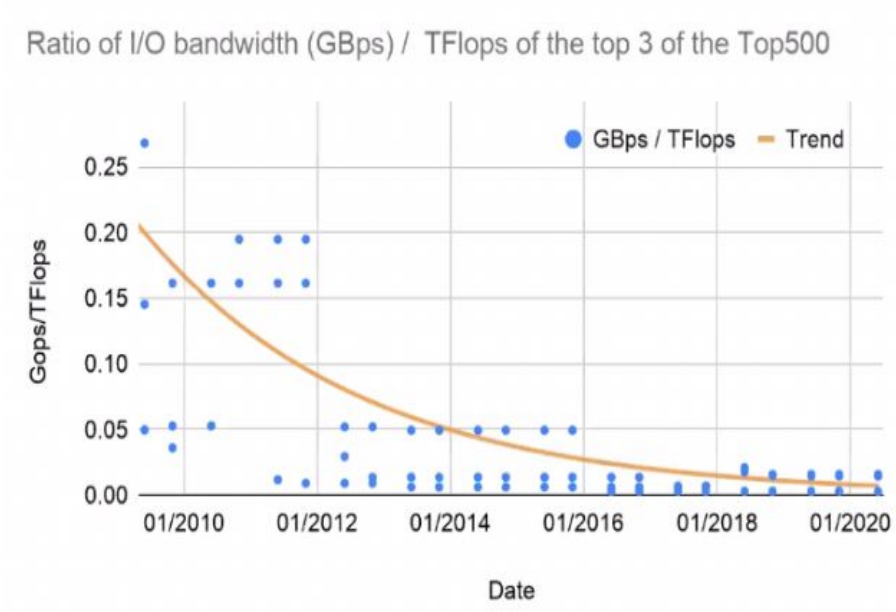
Visualization



ParaView

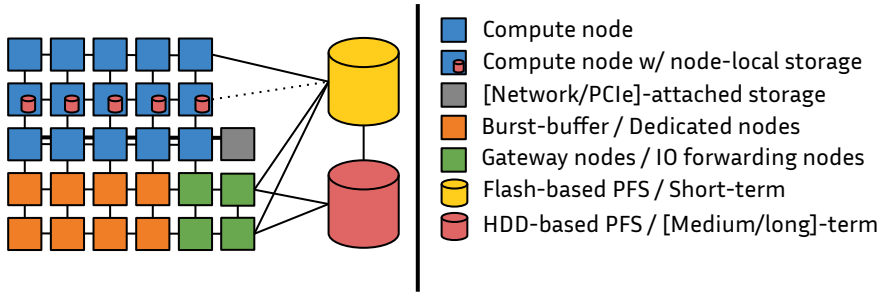
HPC Achilles Heel: I/Os

Applications efficiency can be significantly impaired by I/Os



Flops growing faster than I/O bandwidth

The hardware solution: more complex storage heirarchies

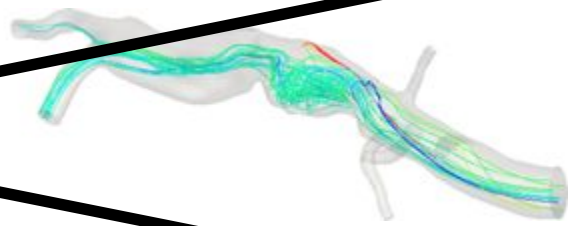


The software solution: perform less I/Os

IN-SITU VISUALISATION

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \mu\nabla^2\vec{v}$$

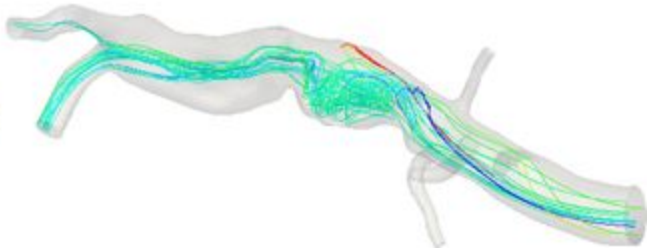
Simulation Code → Visualization



Traditional Visualization

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \mu\nabla^2\vec{v}$$

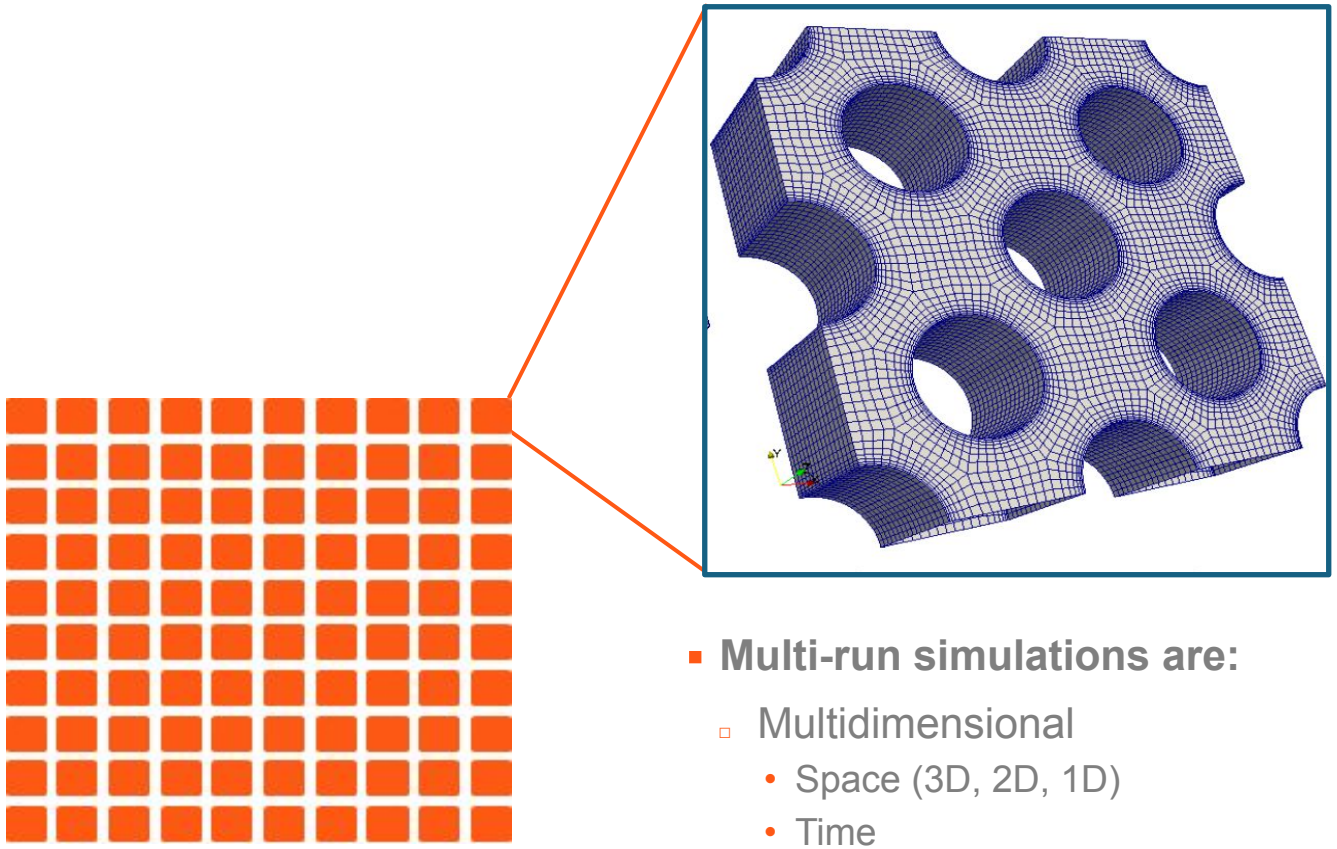
Simulation Code + Visualization



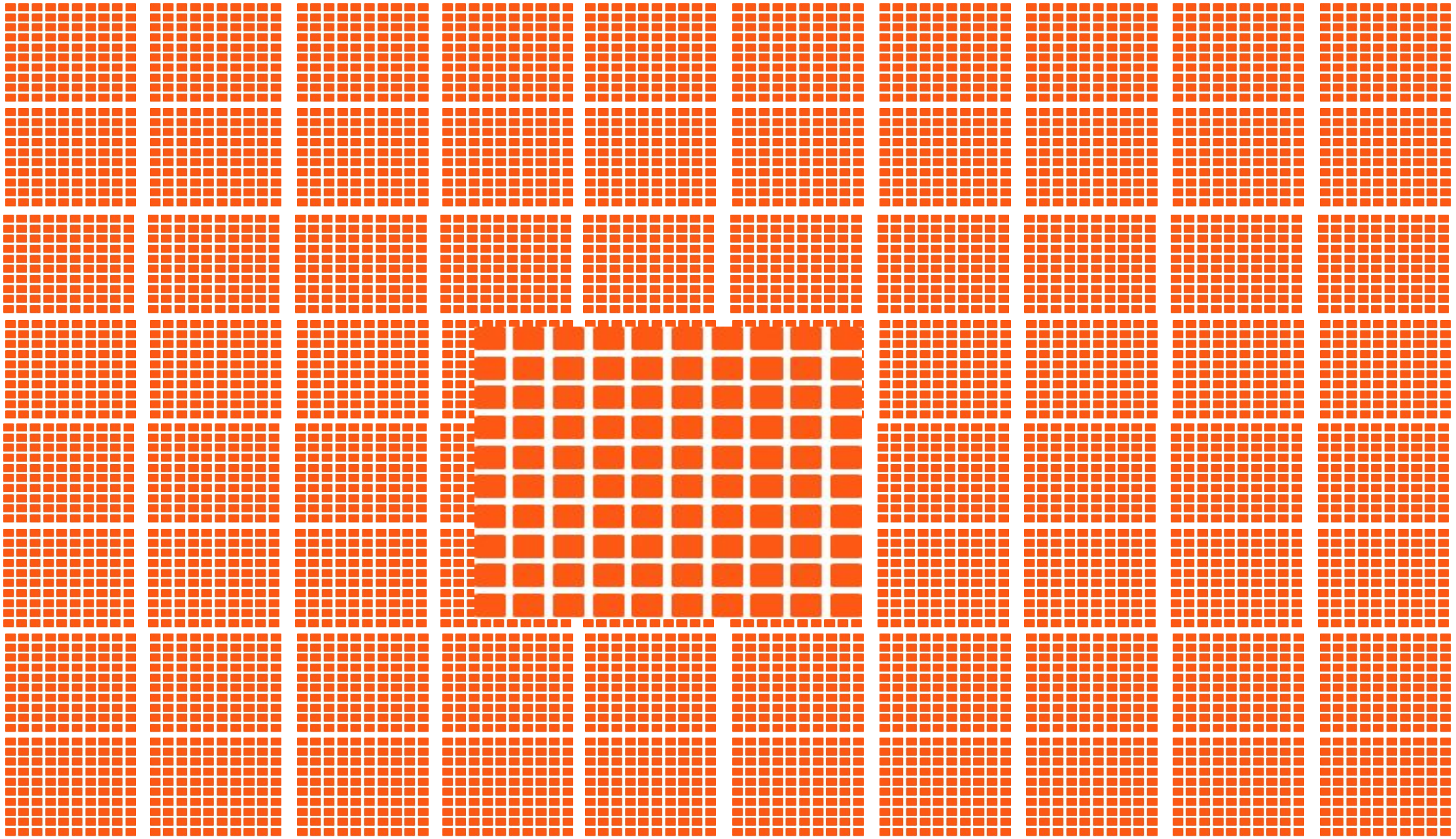
In-situ Visualization

ParaView Catalyst

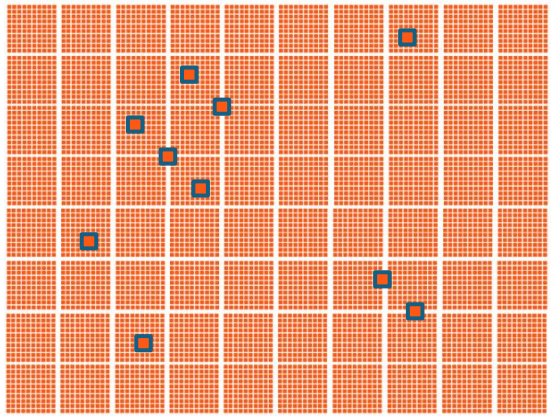
LARGE PARAMETRIC STUDIES: **PROBLEM**



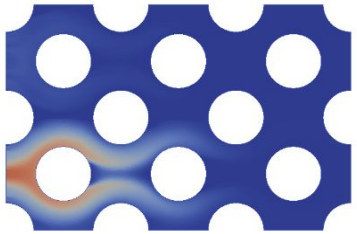
- **Multi-run simulations are:**
  - Multidimensional
    - Space (3D, 2D, 1D)
    - Time
  - Multivalued (temperature, pression, height, etc)
  - Multivariate (1,000 or 100,000?)



LARGE PARAMETRIC STUDIES: **PROBLEM**



Post-traitement



PROBLEM:

48TB

some MB

Mean

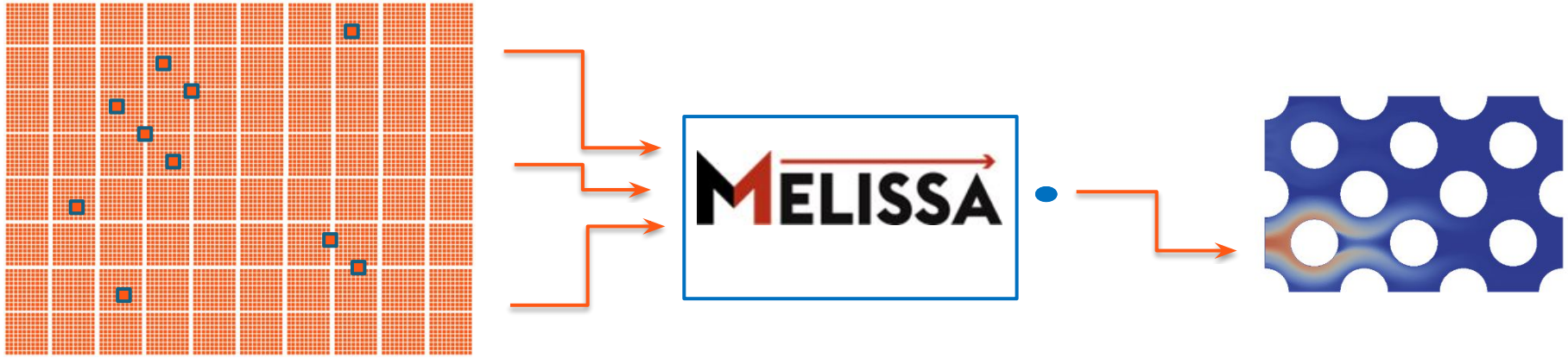
Variable

Spatial Coordinates

Time Step

$$\mu(X,t) = \frac{\sum_{i=1,N} u(i,X,t)}{N}$$

LARGE PARAMETRIC STUDIES: ***THE SOLUTION IS MELISSA***



48To

Some Mo

<u>Zero intermediate files</u> thanks to iterative statistics

Iterative Mean (i^{th} update):

$$\mu_i(X,t) = \mu_{i-1}(X,t) + \frac{1}{i}(u(i,X,t) - \mu_{i-1}(X,t))$$

Iterative standard deviation (i^{th} update):

$$V_i(X,t) = V_{i-1}(X,t) + (u(i,X,t) - \mu_{i-1}(X,t))(u(i,X,t) - \mu_i(X,t))$$

Iterative statistical library (in Python):

https://github.com/IterativeStatistics

MELISSA WORKS SC17

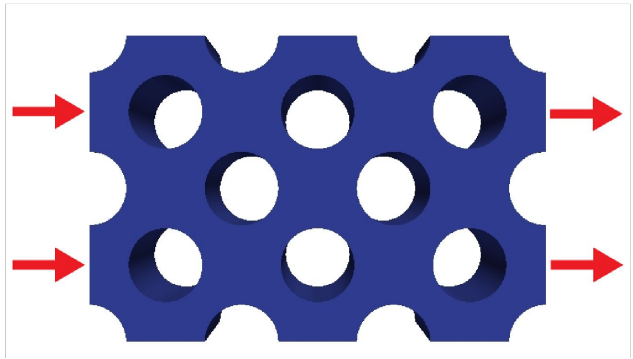
Curie Machine (80K cores)

Fluid simulation with Code_Saturne [EDF]

9M hexahedral mesh – 100 timesteps

6 parameters,
3 per injector:

- Dye concentration
- Injection width
- Injection duration

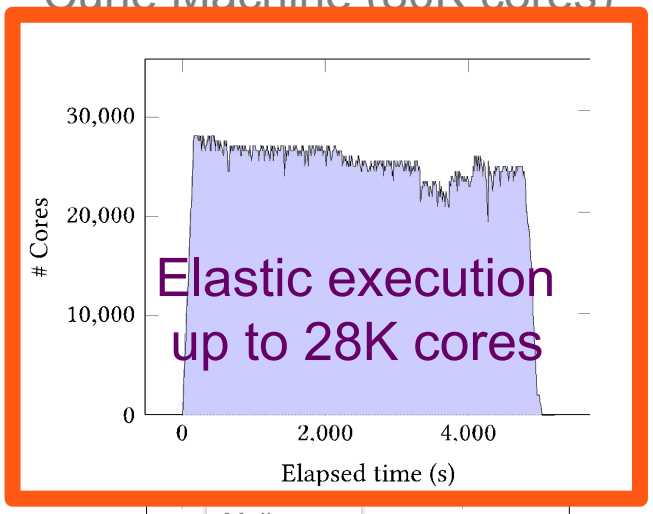


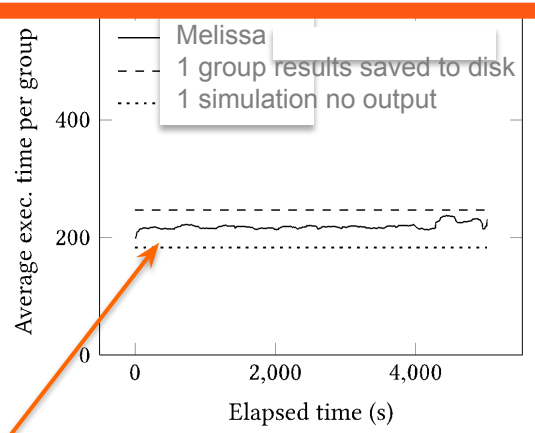
Ubiquitous Sobol' indices: 9x100X2=1800M indices (dye concentration)

8 simulations per group, 1000 groups, each one running on 512 cores

Generate 48TB of intermediate results

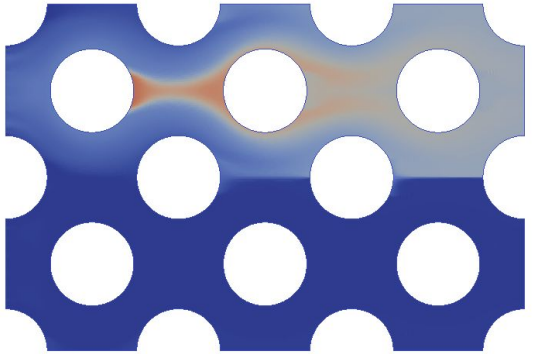
Server size: enough nodes to work in memory (491GB)



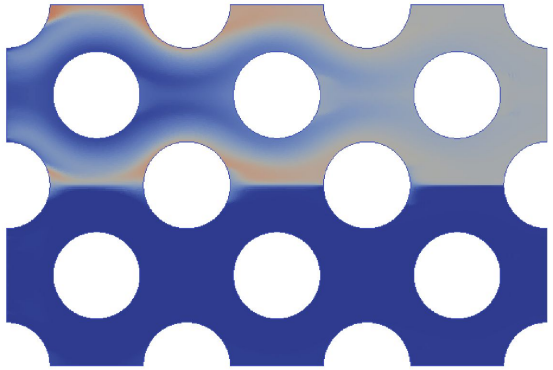


Run 13% faster on average than when writing to disk

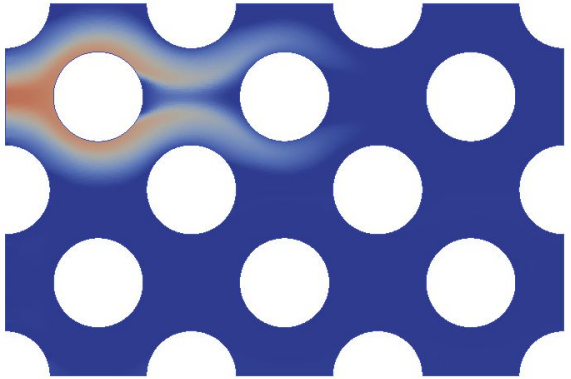
Injector 1



Die concentration



Width



Duration

MELISSA WORKS

SC18 Dallas, TX | hpc inspires.

Supercomputing 2018

80 000 simulations

On-line processing of 288T

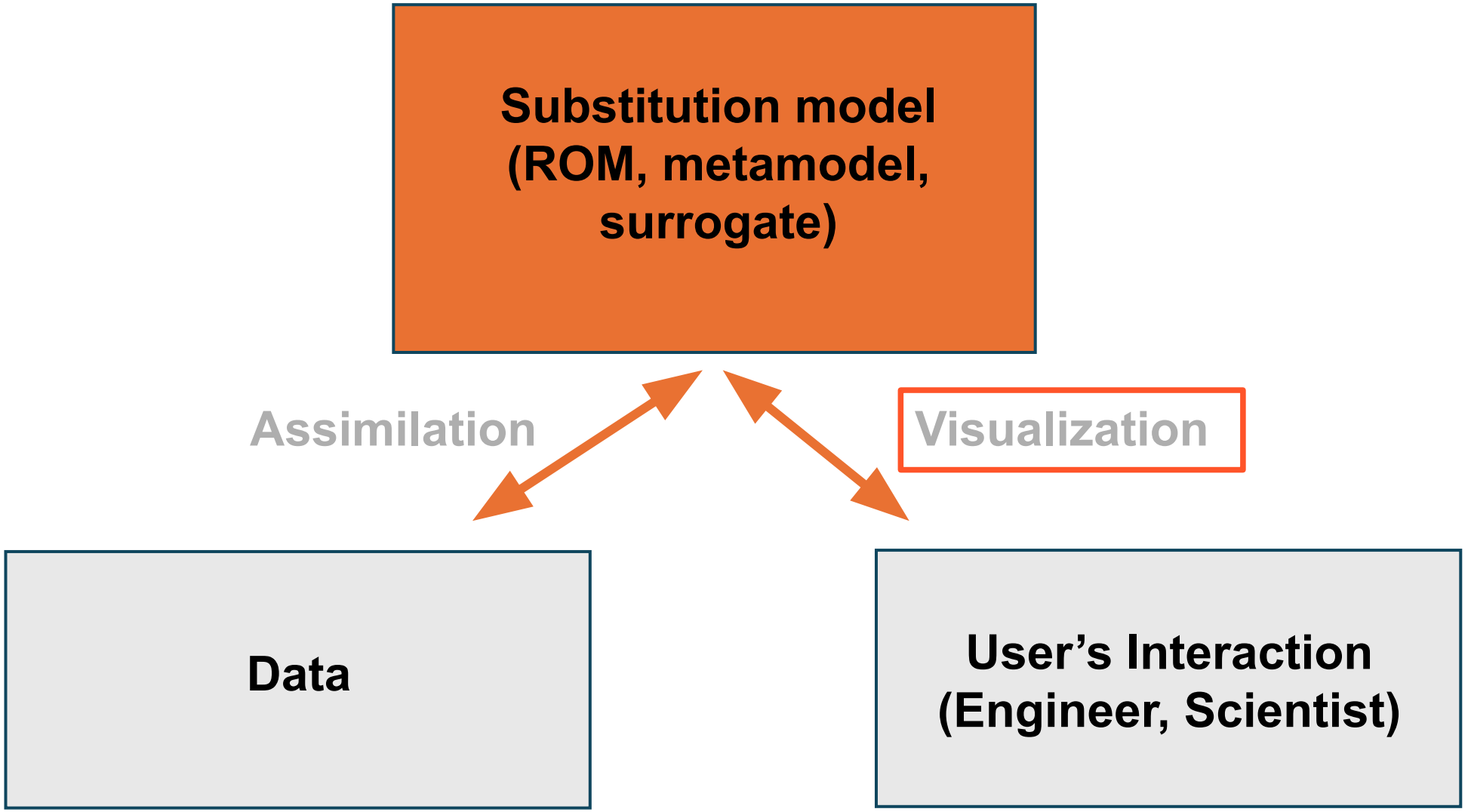
Sobol Indices

Quantiles

A. Ribes, T. Terraz, Y. Fournier, B. Iooss, and B. Raffin. Large Scale Computation of Quantiles using Melissa. In Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis, Dallas, Texas USA, November 2018 (SC'18).

Inria EDF

Surrogates and Digital Twins :

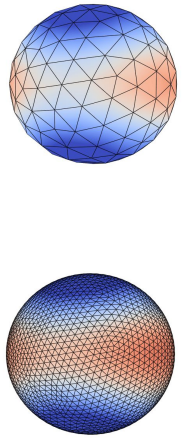


Numerical Simulations

Generate a trajectory:

$$(u_0, u_1, u_2, \ldots, u_T)$$

$$\begin{cases} \dfrac{\partial u}{\partial t} - \alpha \Delta u = f, \forall x \in \Omega, \ \forall t \in [0, T] \\ u(t=0) = u_0 \\ u(x,t) = u_{\partial\Omega}, \forall x \in \partial\Omega, \ \forall t \in (0, T] \end{cases}$$

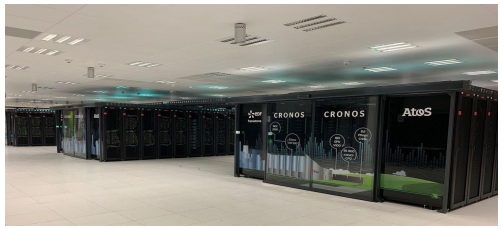


$$u_{t+1} = f(u_t)$$

Equation (PDE) → Discretization → Solver code

Scaling requires parallelized solvers to run on supercomputers.

The essence of HPC !



Deep Surrogates

Train a NN approximating the PDE solution.

Expected benefits: faster than traditional solver, less memory consuming

Two main approaches:

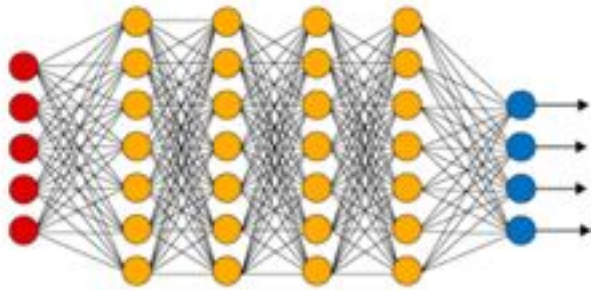
- No training data set required: PINNs
- Training data set required:

Neural Operators (FNO), GNN, Vision Transformer(ViT), ….

Use a classical solver to generate the training data set (synthetic data :-)

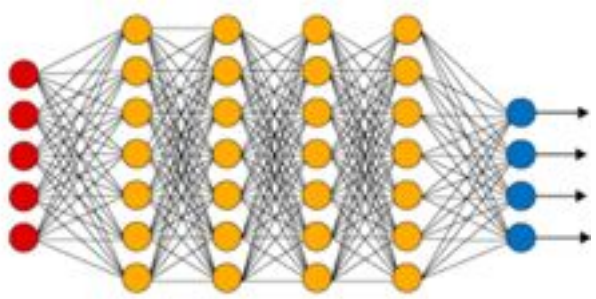
The NN can be:

- Autoregressive: $u_{t+1} = f_\theta(u_t)$
- Direct: $u_t = f_\theta(x_0, t)$



Deep Surrogates

$$u_X^{t+1} = f_\theta(u_X^t)$$



Training from one trajectory: $(u_0, u_1, u_2, \ldots, u_T)$ Neural-GCM-arXiv:2311.07222

Training from multiple trajectories:

- Varying initial conditions

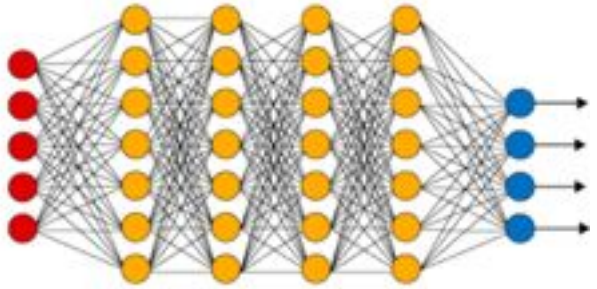
$$\begin{cases} \dfrac{\partial u}{\partial t} - \alpha \Delta u = f, \forall x \in \Omega, \ \forall t \in [0, T] \\ u(t=0) = u_0 \\ u(x,t) = u_{\partial\Omega}, \forall x \in \partial\Omega, \ \forall t \in (0, T] \end{cases}$$

- Varying the discretization: GNS-arXiv:2002.09405
- Varying the PDE - Fondation models for PDES
  - Poseidon- arXiv:2405.19101
  - PDE-Transformer -arXiv:2505.24717

Let’s go back to the basics

Software Stack: Pytorch, Jax

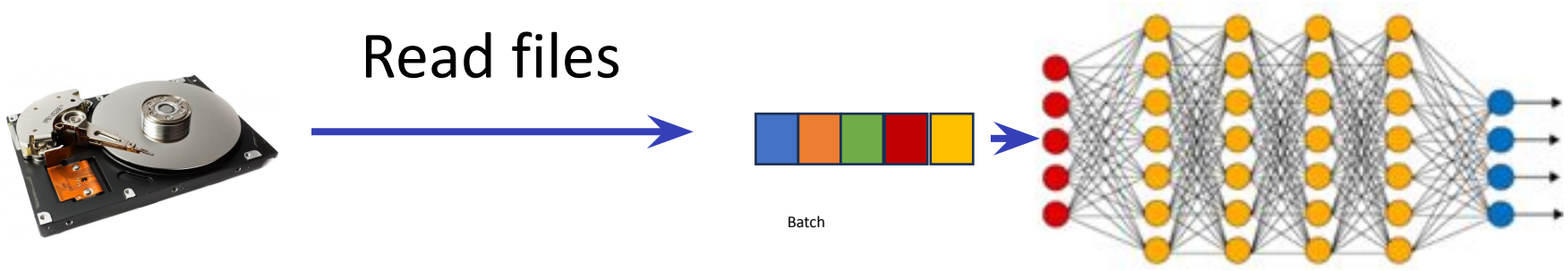
A dataset



A neural architecture

Compute resources (GPUs)

Offline Training



Repeat data through epochs if not enough available.

Multi-GPU Training

The gradient descent is performed once per batch and the associated parameter update is:

$$w_j^{t+1} = w_j^t + \alpha \sum_{i \in batch} \frac{\delta E}{\delta w_j}\ (x_i)$$

If we split the batch in tow parts:

$$batch = \text{batch}_{\text{GPU1}} \cup \text{batch}_{\text{GPU2}}$$

We can rewrite the weight update:

$$w_j^{t+1} = w_j^t + \alpha\ (\sum_{i \in batch_{GPU1}} \frac{\delta E}{\delta w_j}\ (x_i) + \sum_{i \in batch_{GPU2}} \frac{\delta E}{\delta w_j}\ (x_i)\ )$$

We've got one way to parallelize training ! This is called **Distributed Data Parallelism (DDP)**

Distributed Data Parallelism

Algorithm with 2 GPUs:

1. Duplicate the Neural Network on each GPU
2. Split the batch in two equal parts $batch = \text{batch}_{\text{GPU1}} \cup \text{batch}_{\text{GPU2}}$
3. Each neural network processes its ½ batch and computes the associated gradients:

$$\sum_{i \in batch_{GPUk}} \frac{\delta E}{\delta w_j}(x_i)$$

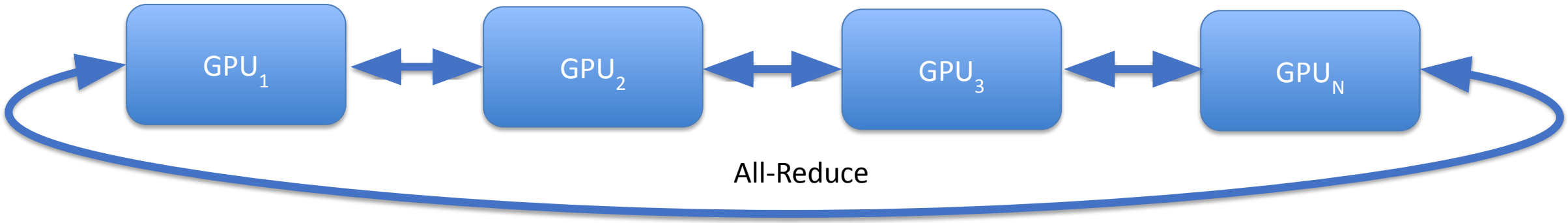
1. Exchange the results between GPUs (all-reduce op)

2. Each NN can updates its weights with the global gradient (same update on each duplicate, so NN copies stay consistent):

$$w_j^{t+1} = w_j^t + \alpha \left(\sum_{i \in batch_{GPU1}} \frac{\delta E}{\delta w_j}(x_i) + \sum_{i \in batch_{GPU2}} \frac{\delta E}{\delta w_j}(x_i) \right)$$

We can go almost twice faster!

DDP: All-Reduce



Organizing the communications in a smart way, it's possible to get a cost that grows sub-linearly with the number of GPUs:

$$T(n) = 2\alpha \log(p) + 2\, n/\beta\, (p-1)/p + \gamma \,.\, n \,.\, (p-1)/p$$

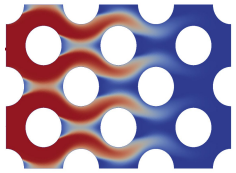
It's called an All-Reduce operation, available in standard collective communications libraries like MPI, Gloo, NCCL, RCCL.

It's embedded in libraries like Pytorch, Jax and easy to deploy.

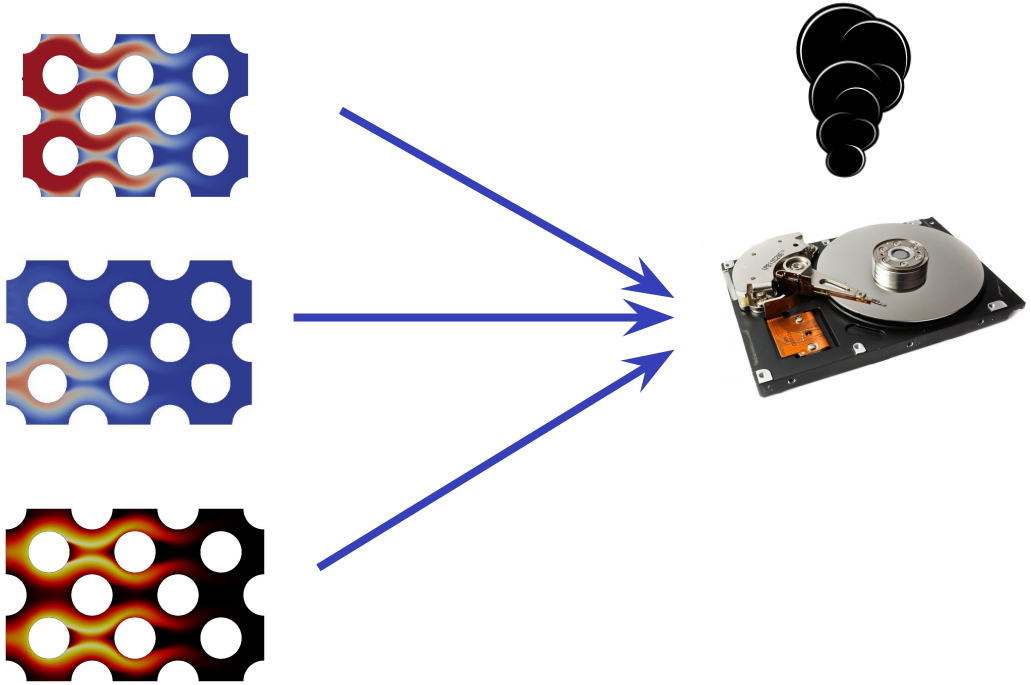
Beware that at high GPU counts you need to increase the size of the batch to keep them busy, which in turn, requires an adapted learning rate decay strategies.

Offline Training

Run solver to generate a data set

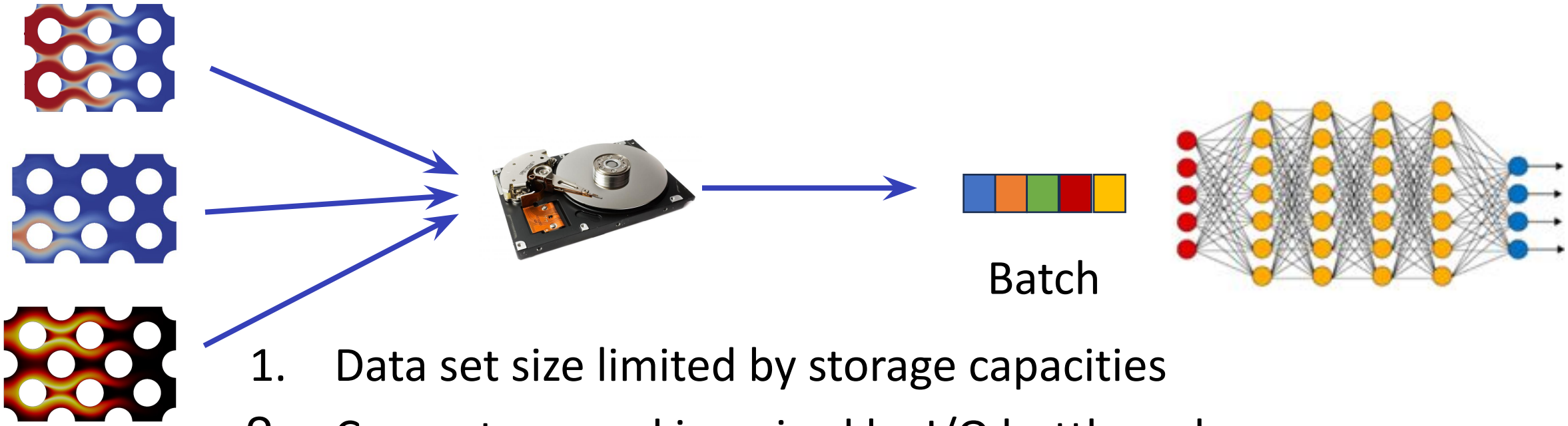


Write to files



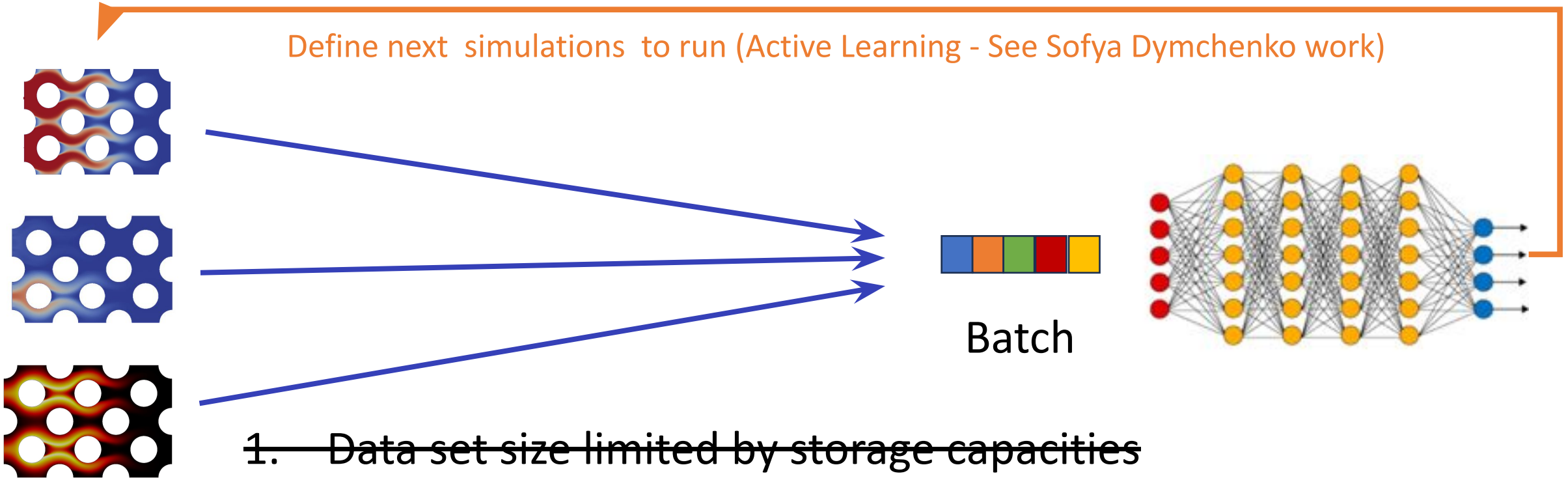
Multi-parametric: run ensemble of solver instances

Offline Training



1. Data set size limited by storage capacities
2. Compute speed impaired by I/O bottleneck
3. No possibility to steer data generation based on NN needs (active learning)

Online Training

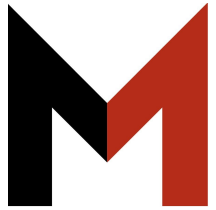


~~1. Data set size limited by storage capacities~~

~~2. Compute speed impaired by I/O bottleneck~~

~~3. No possibility to steer data generation based on NN needs~~

Melissa

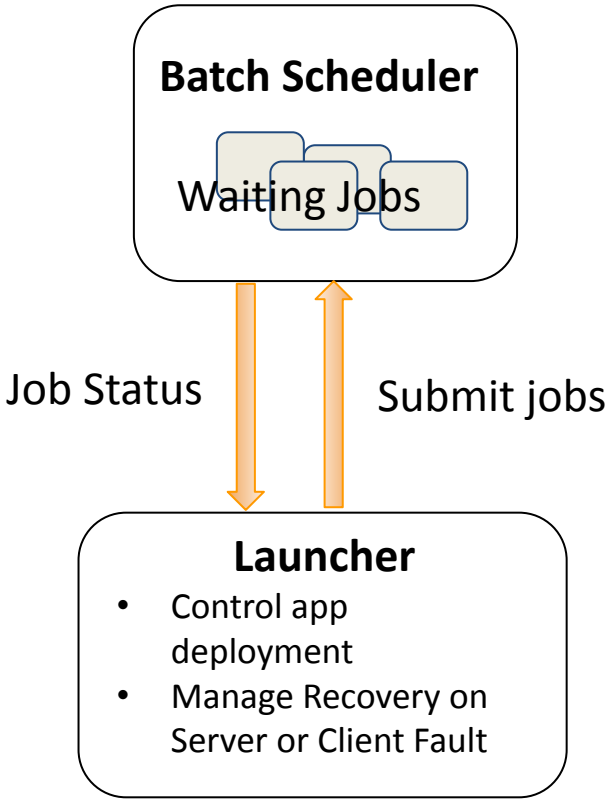


A HPC empowered framework for handling online training at large scale:

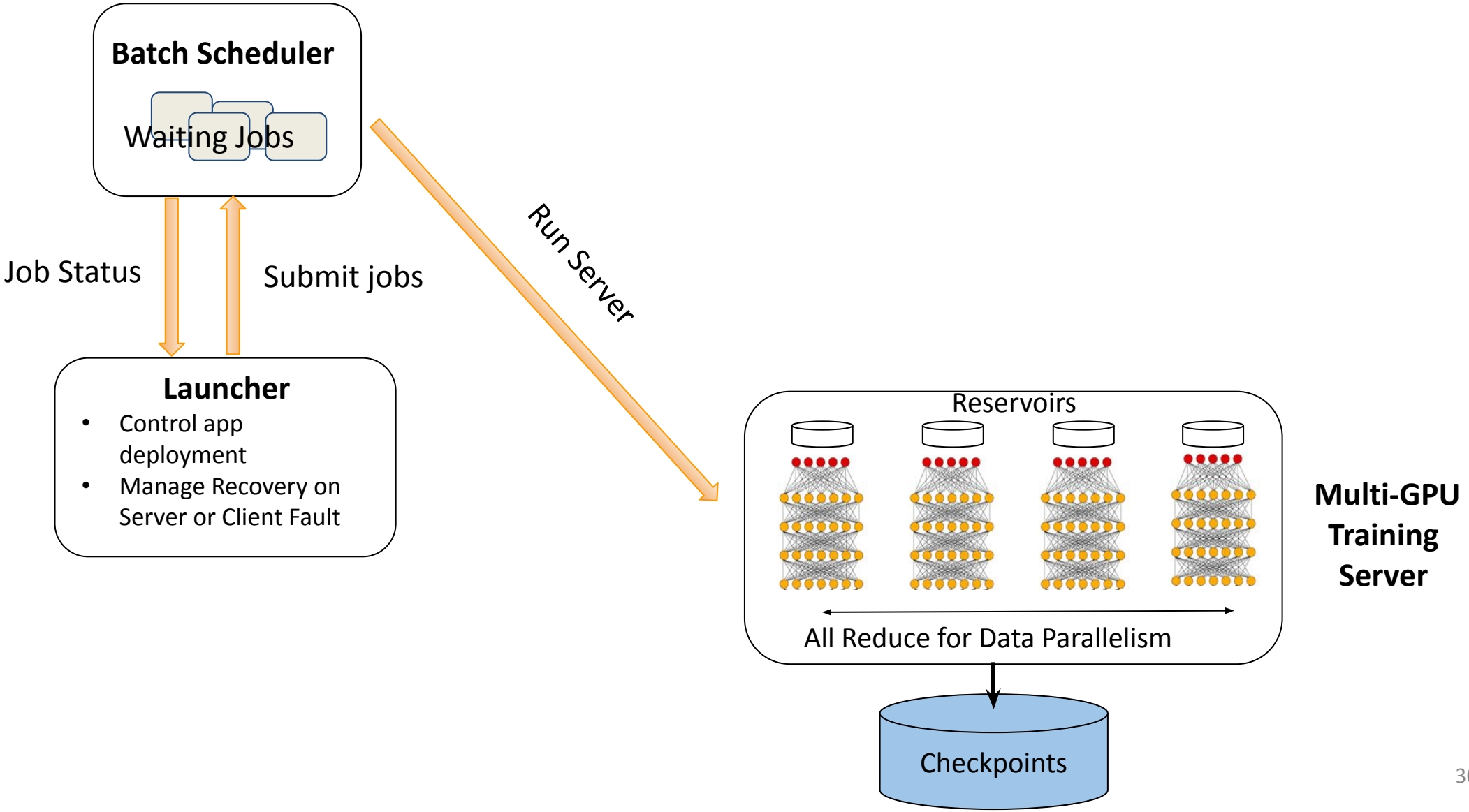
- Scalable (parallel simulation, parallel training, concurrent simulation executions)
- Elastic (number of concurrent running simulations can vary over time to adapt to compute resource availability)
- Fault-tolerant (automatic component restart in case of failure)

https://gitlab.inria.fr/melissa

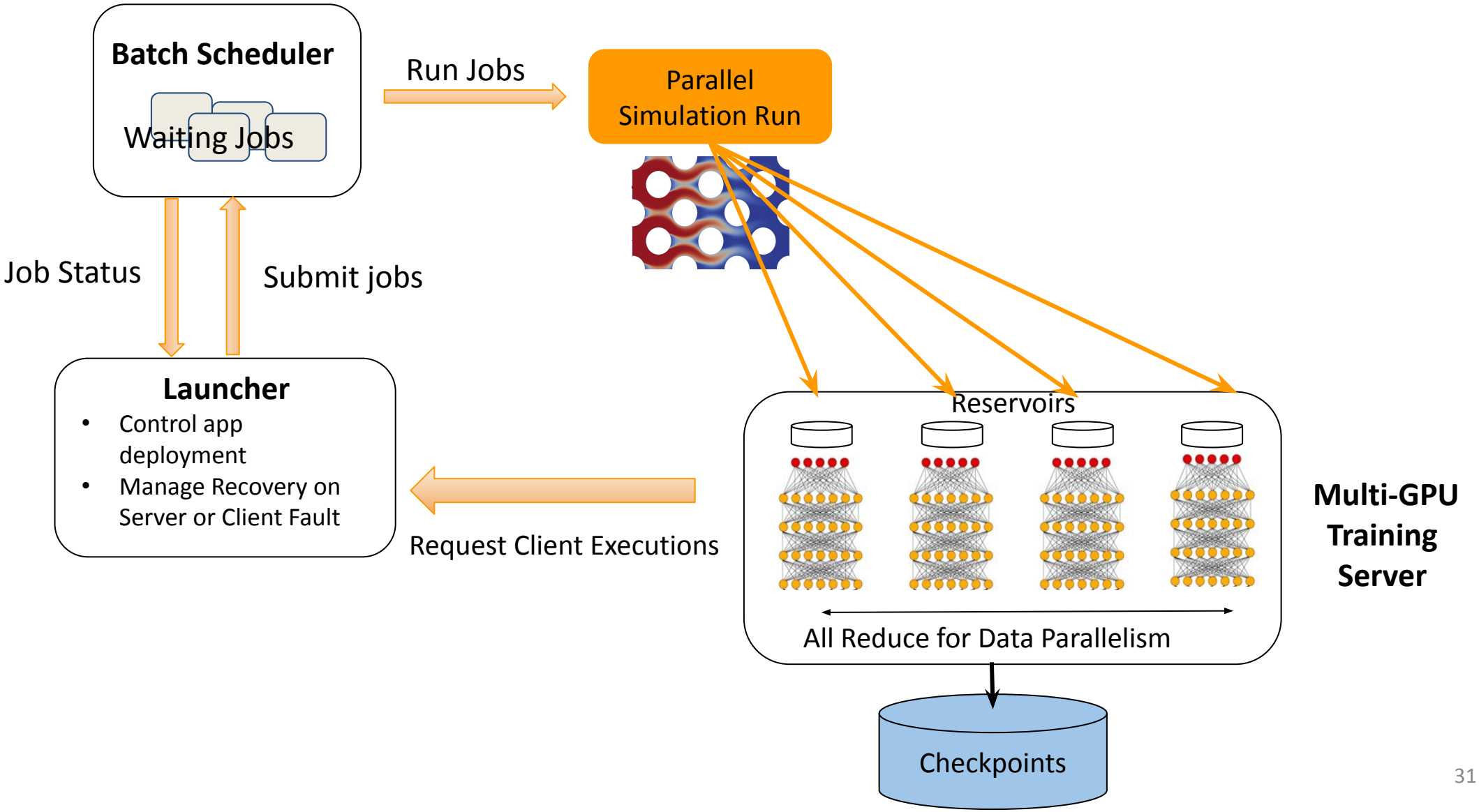
Melissa Architecture



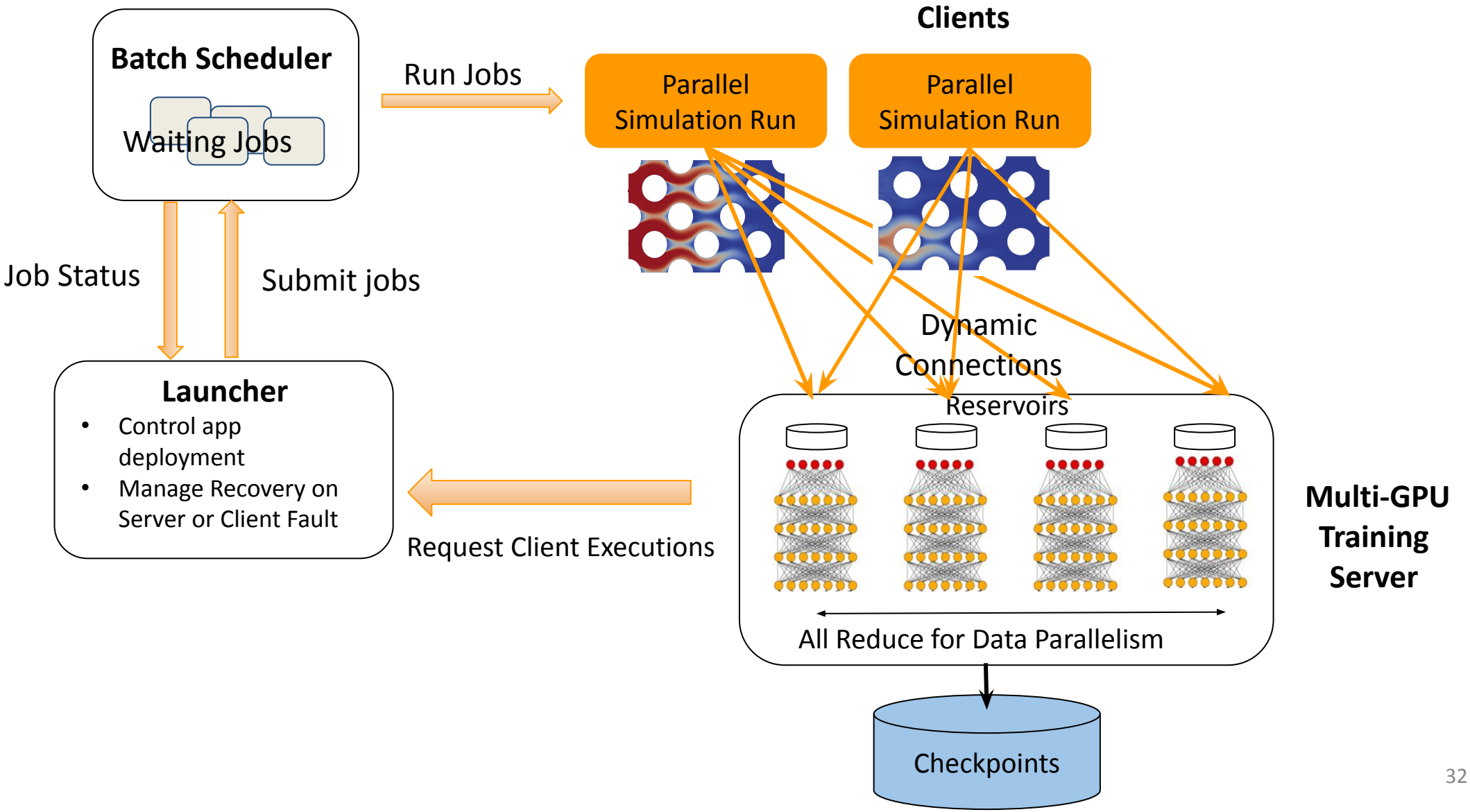
Melissa Architecture



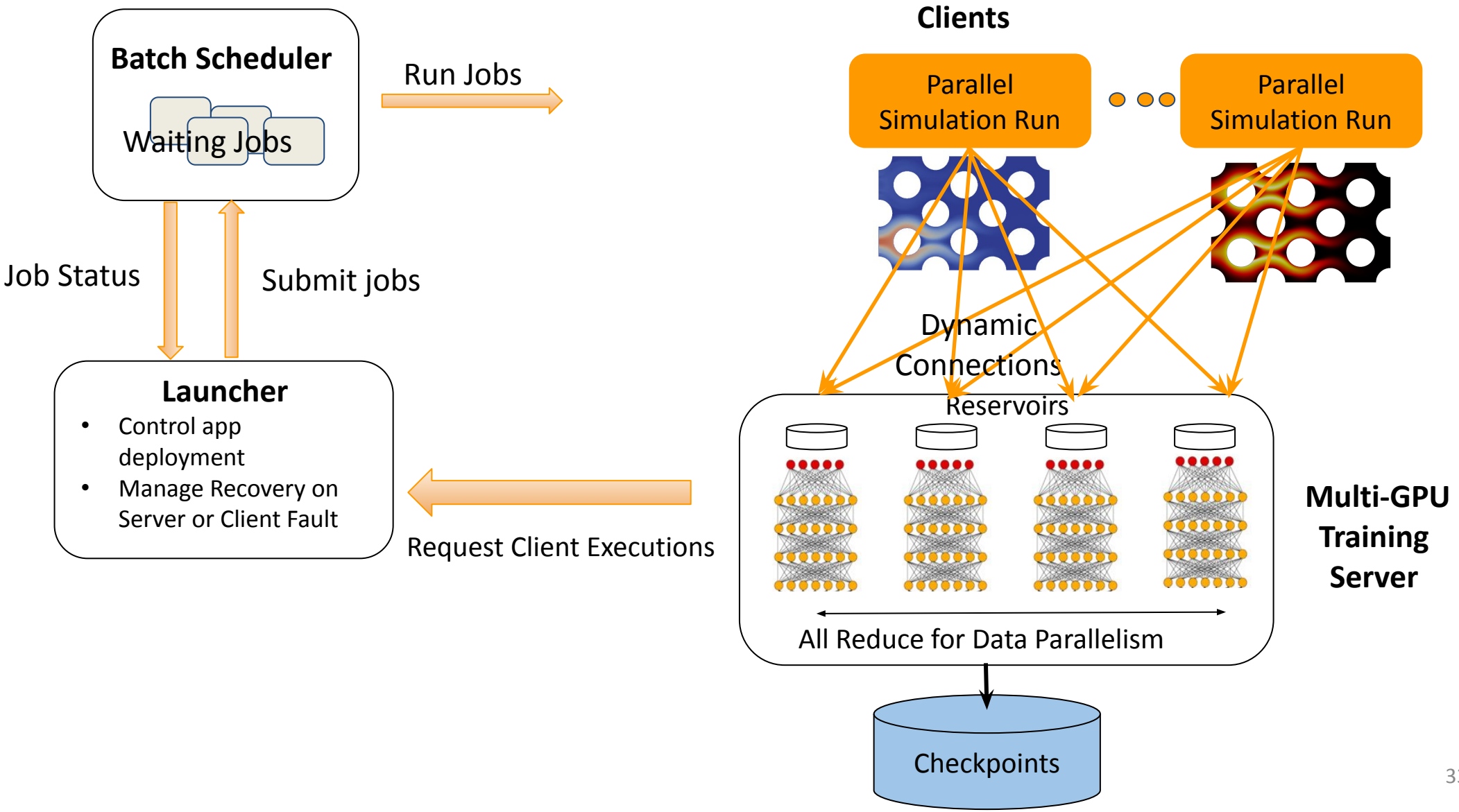
Melissa Architecture



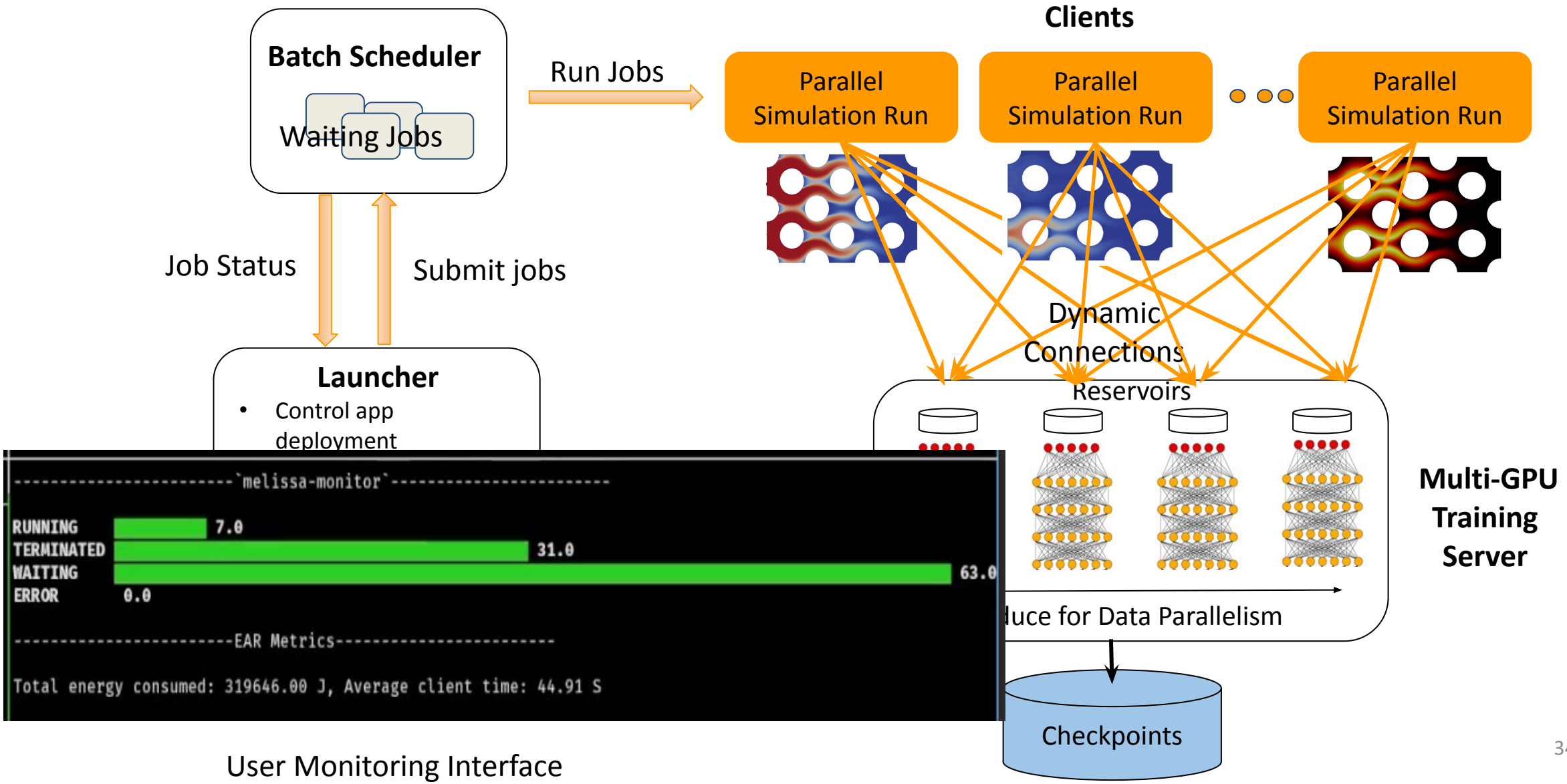
Melissa Architecture



Melissa Architecture

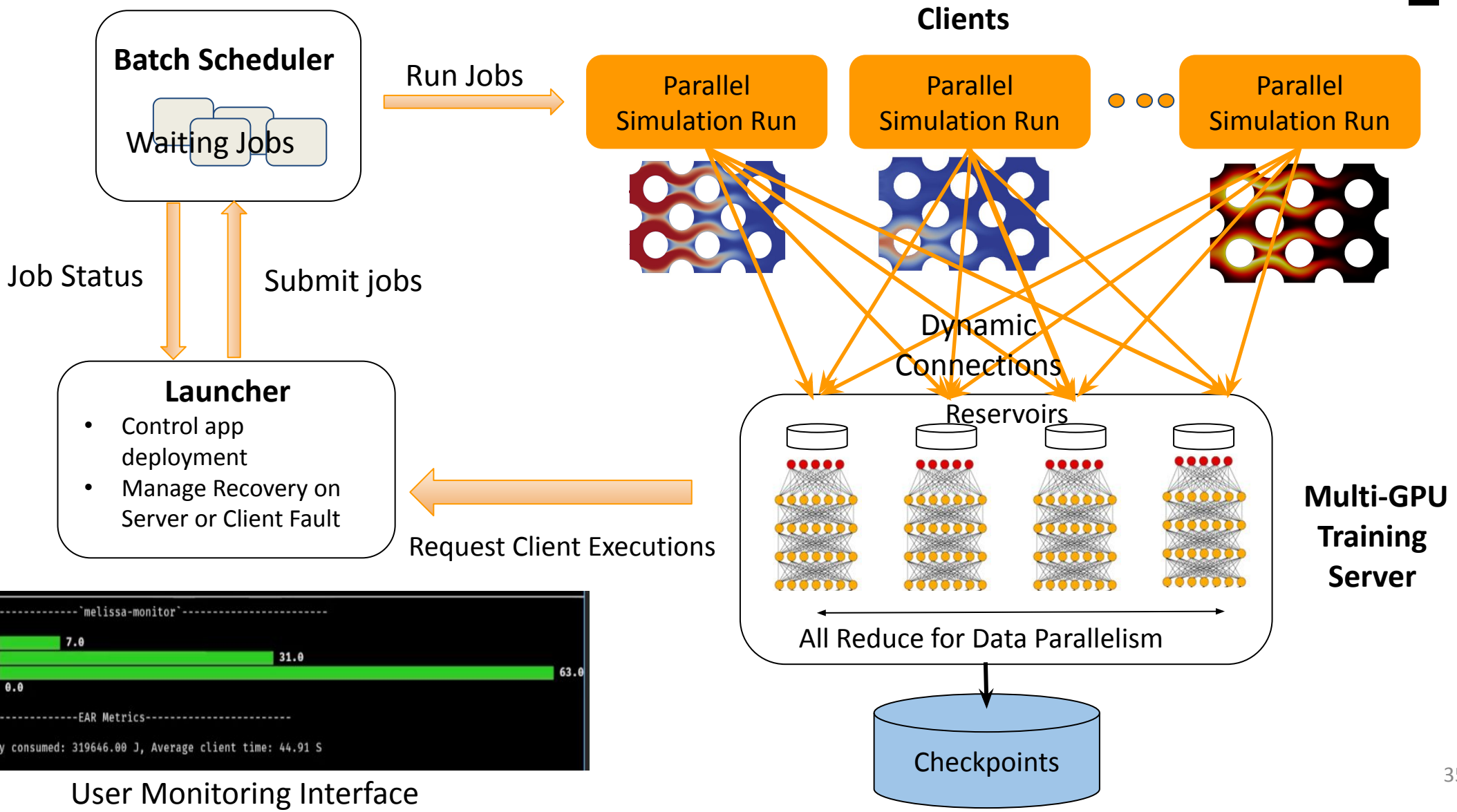


Melissa Architecture



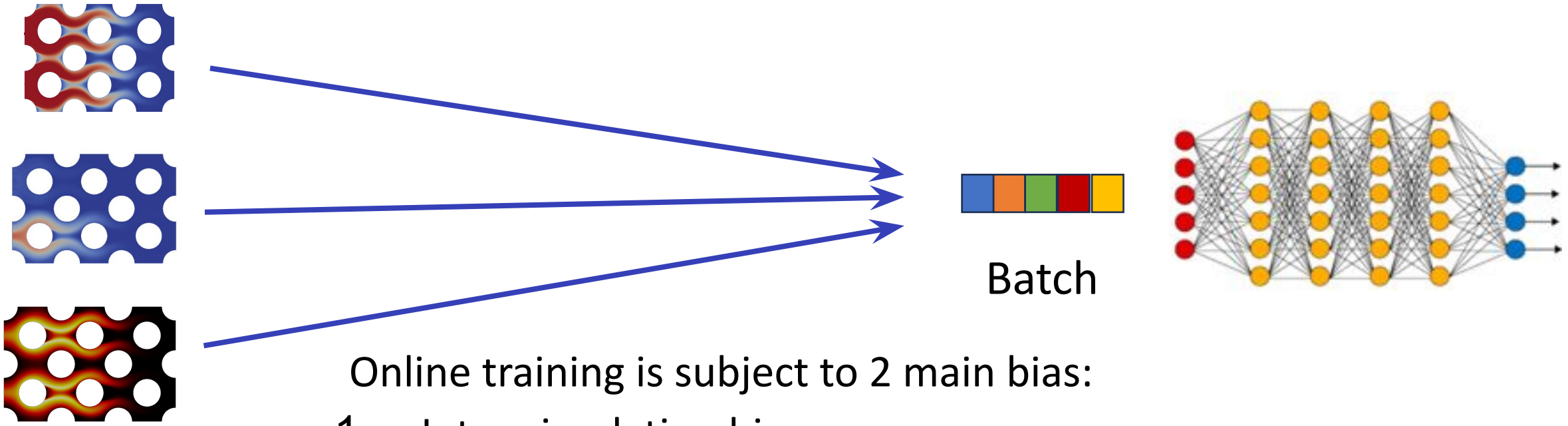
User Monitoring Interface

Melissa Architecture



User Monitoring Interface

Online Training



Online training is subject to 2 main bias:

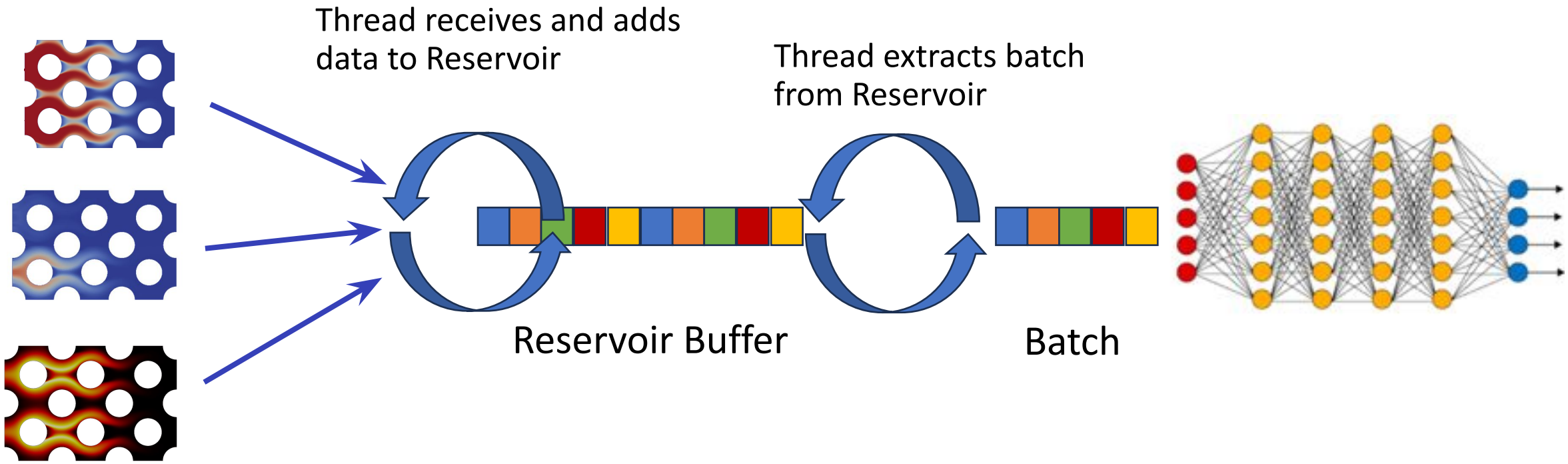
1. Intra-simulation bias:

   Data are produced in time order (t_0, t_1, t_2,…,t_n)

2. Inter-simulation bias:

   The number of concurrent running simulations depends on compute resource availability

Mitigation Strategy: Reservoir Buffer

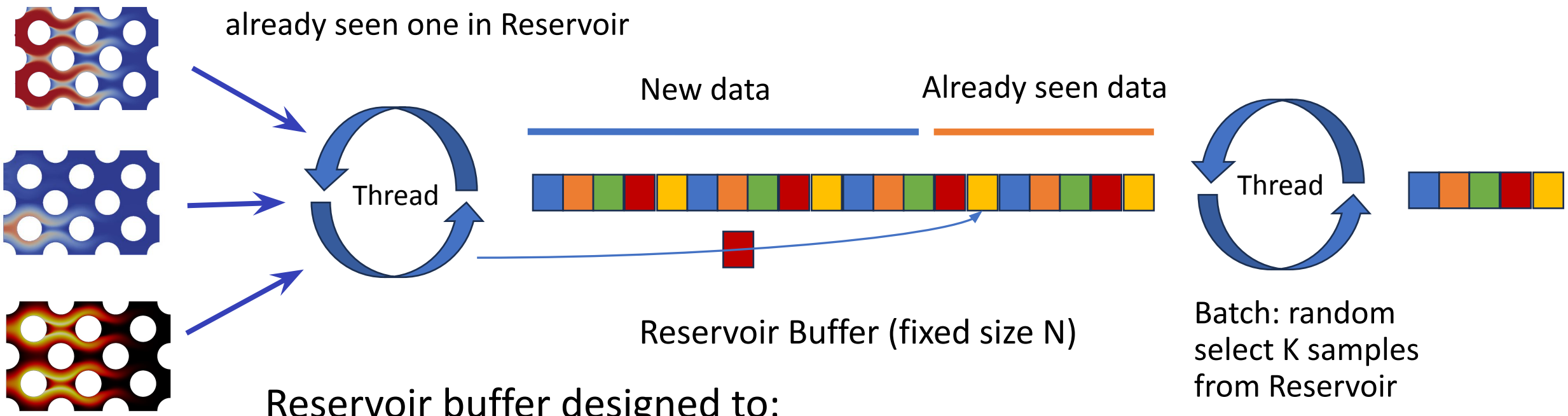


Reservoir buffer designed to:

1. Keep GPU from starving (potentially repeating data in batches)
2. Increase data mixing to mitigate bias due to online training

Reservoir Buffer

Expected sample residency time : N-1

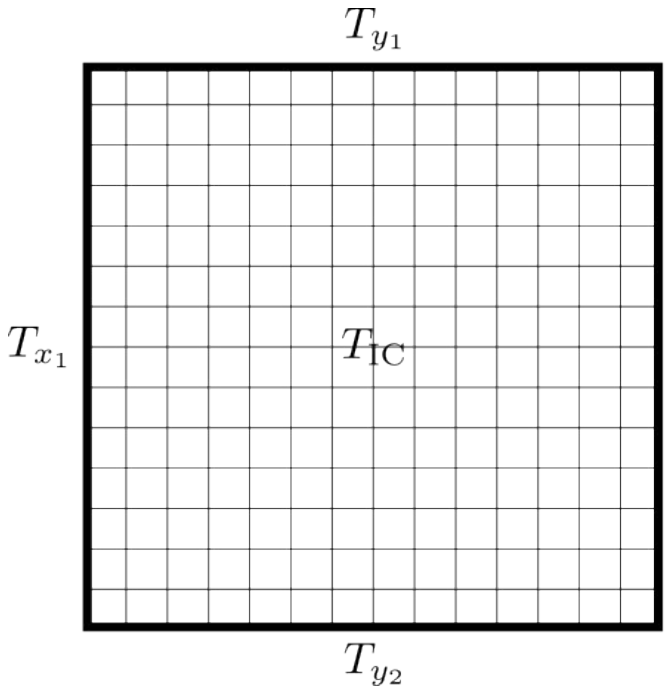


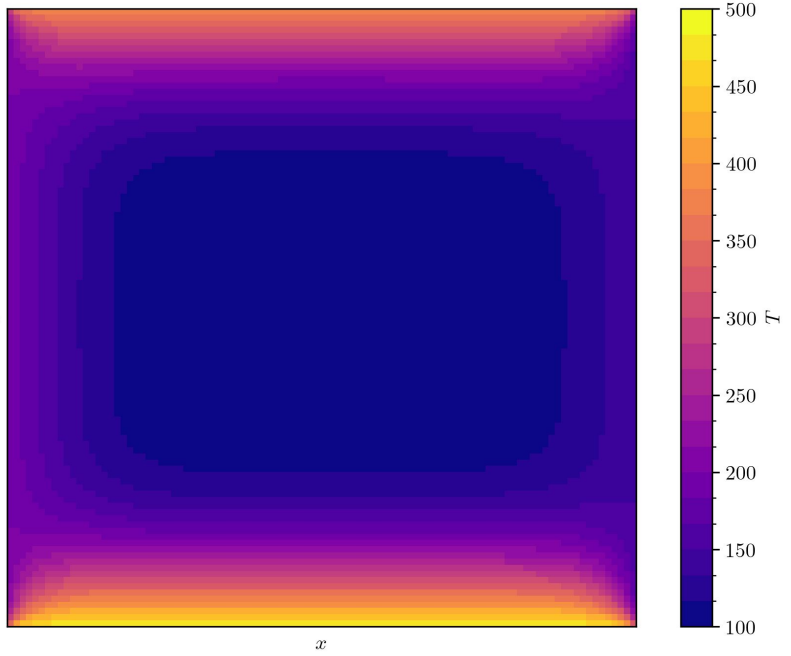
Reservoir buffer designed to:

1. Keep GPU from starving:
   - Build batch of already seen data
2. Increase data mixing to mitigate bias due to online training:
   - Random selection in Reservoir (only once watermark reached)

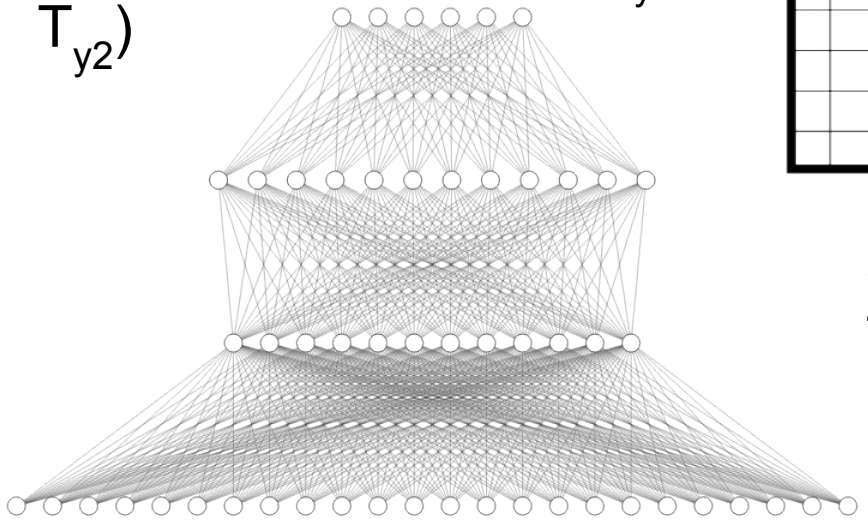
Experiments: 2D Heat PDE

$$\begin{cases} \dfrac{\partial T}{\partial t} = \alpha \Delta T, \\ T(x,y,0) = T_{IC} \\ T(0,y,t) = T_{x_1}, T(L,y,t) = T_{x_2} \\ T(x,0,t) = T_{y_1}, T(x,L,t) = T_{y_2} \end{cases}$$





Input (t, T_{IC}, T_{x1}, T_{x2}, T_{y1}, T_{y2})

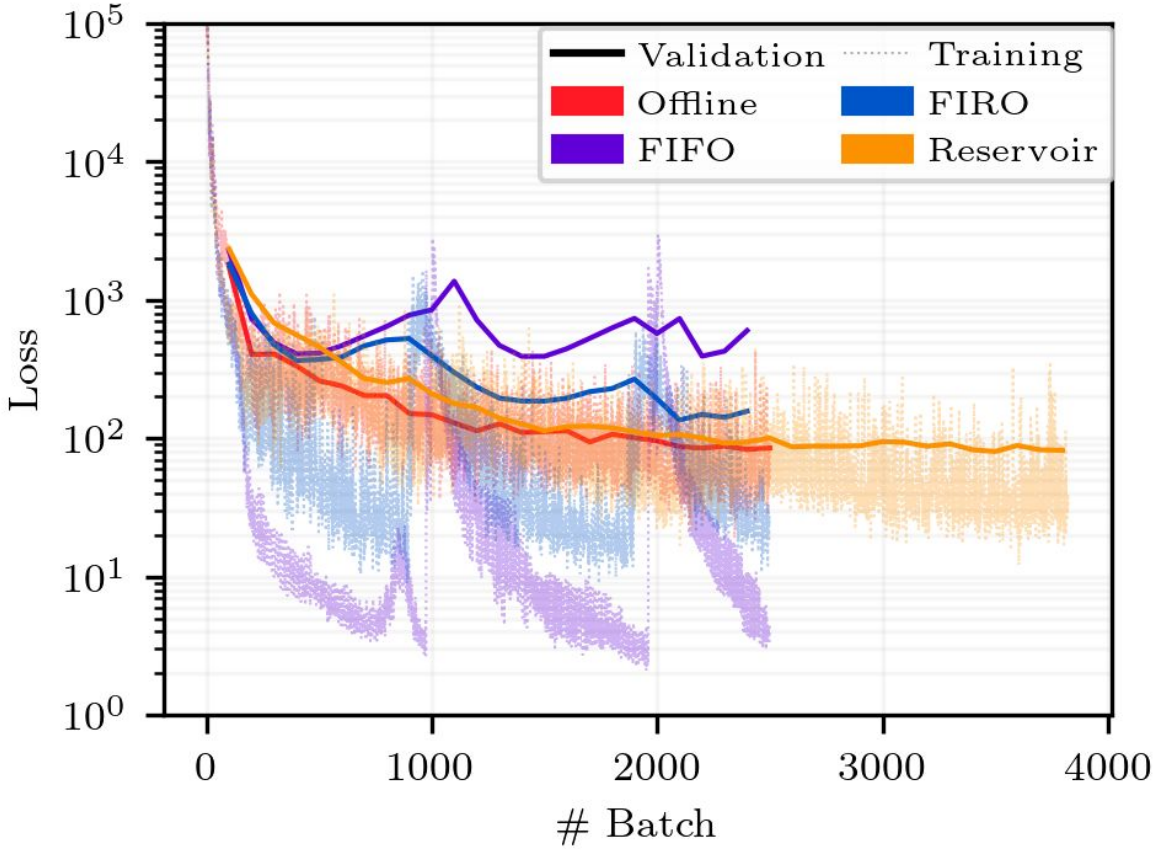


Output (T)

6 parameters (5 temperatures + time)
Temperature in [100,500]K
100 time steps, Cartesian grid 1000 x 1000

340M parameters

Bias Mitigation



250 simulations, 1 GPU for training, 50 nodes for data generation.
Learning rate updates at 1000 and 2000 batches

Offline training:

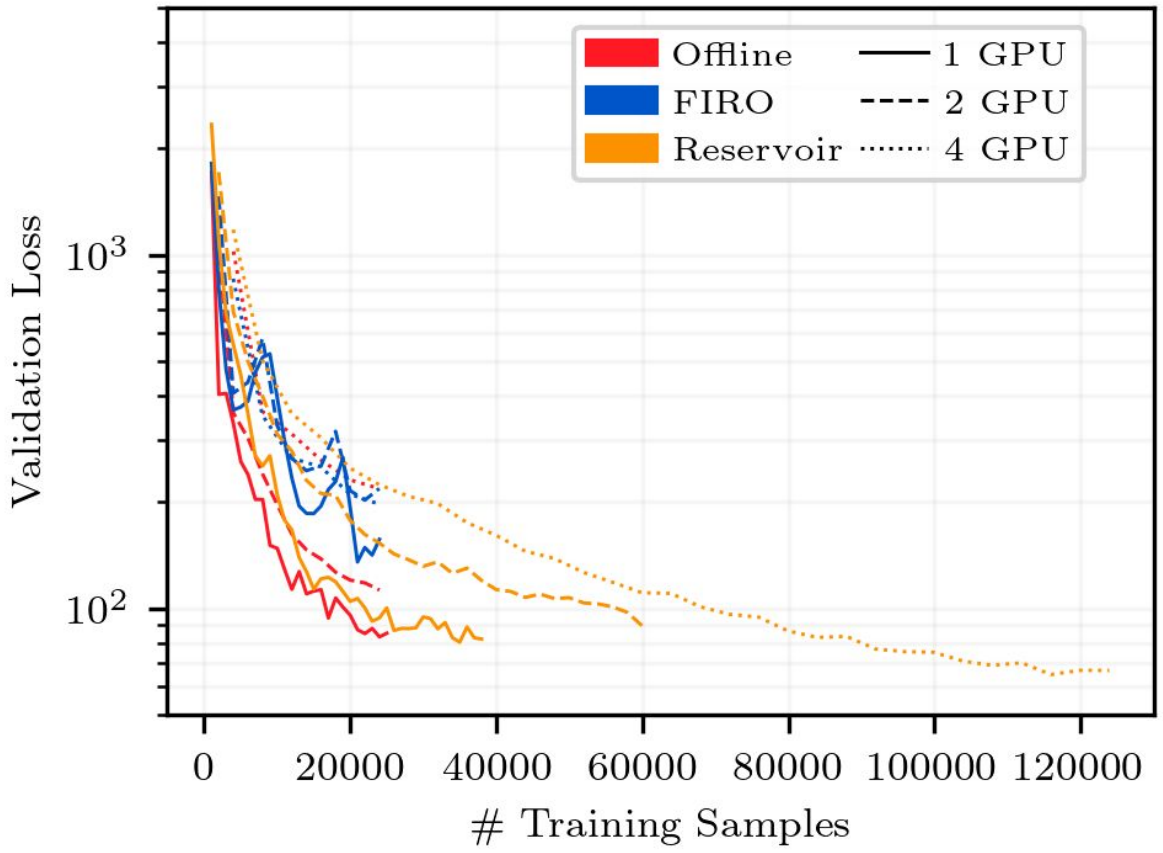
- Single epoch
- Reference MSE (no bias)

Online training:

- FIFO: First-in First-out Buffer
- FIRO: First-in Random-out Buffer
- Reservoir

Buffer	MSE	Time (*Hours*)
Offline	83.1	0.93
FIFO	391	**0.081**
FIRO	135	0.083
Reservoir	**80.3**	0.093

Multi GPU Training



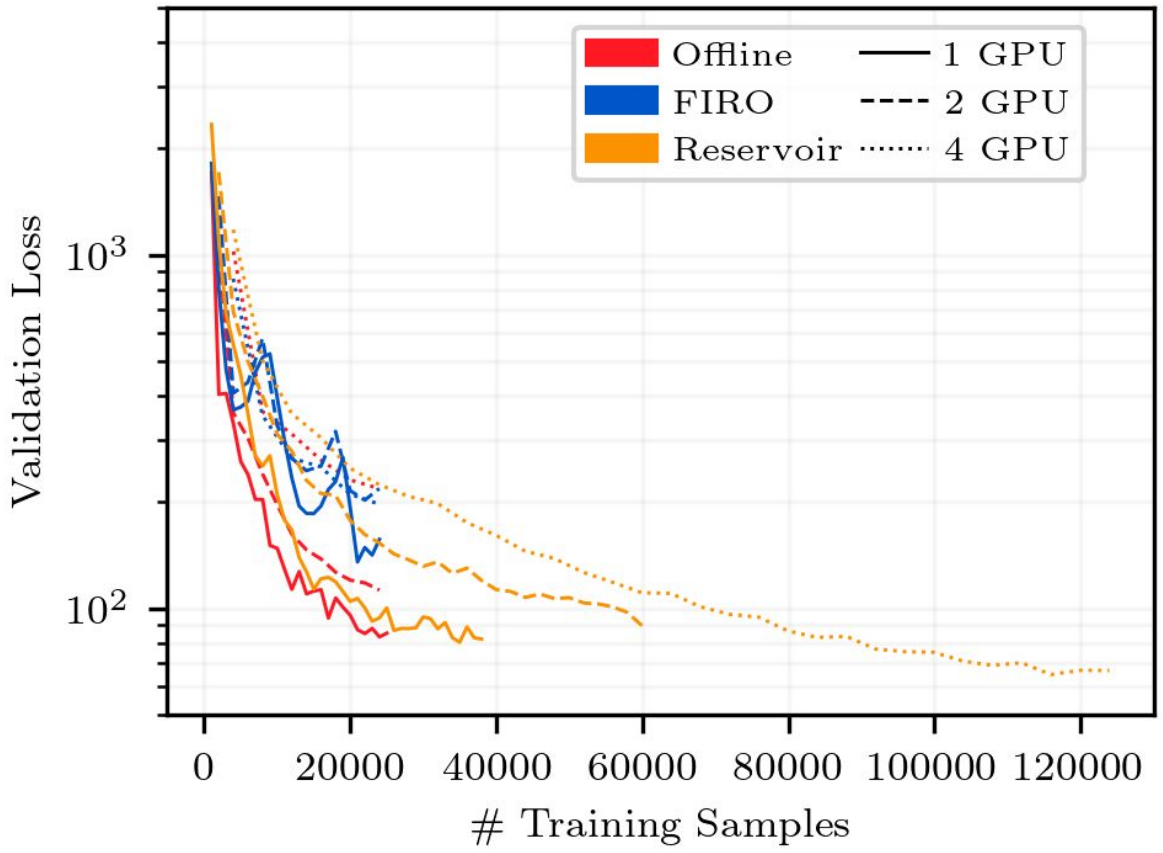
Learning rate halved every 10k sample

As before (250 simulations) but with 2&4 GPUs. Expected same MSE (same data):

- **Offline & FIRO:** lower validation as #GPU increases: larger batch size & less optimization steps
- **Reservoir:** Not enough data, so compensate with already seen data, training on almost 5x samples, leading to a better validation MSE (but not time gains)

Buffer	GPUs	MSE	Time (*Hours*)
Offline	1	83.1	0.93
Offline	4	218	0.10
FIRO	1	135	0.083
FIRO	4	197	**0.082**
Reservoir	1	80.3	0.093
Reservoir	4	**65.0**	0.095

Multi GPU Training



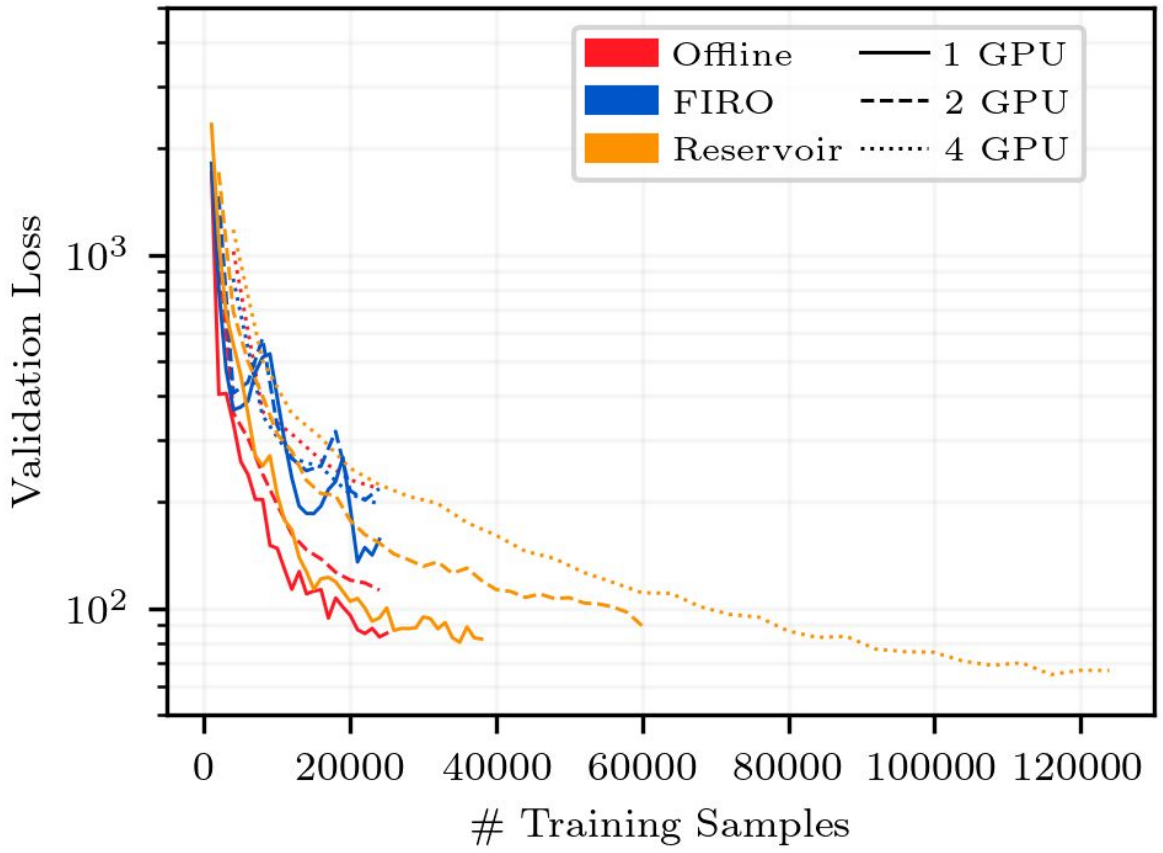
Learning rate halved every 10k sample

As before (250 simulations) but with 2&4 GPUs. Expected same MSE (same data):

- **Offline & FIRO:** lower validation as #GPU increases: larger batch size & less optimization steps
- **Reservoir:** Not enough data, so compensate with already seen data, training on almost 5x samples, leading to a better validation MSE (but no time gains)

Buffer	GPUs	MSE	Time (*Hours*)
Offline	1	83.1	0.93
Offline	4	218	0.10
FIRO	1	135	0.083
FIRO	4	197	**0.082**
Reservoir	1	80.3	0.093
Reservoir	4	**65.0**	0.095

Multi GPU Training



Learning rate halved every 10k sample

As before (250 simulations) but with 2&4 GPUs. Expected same MSE (same data):

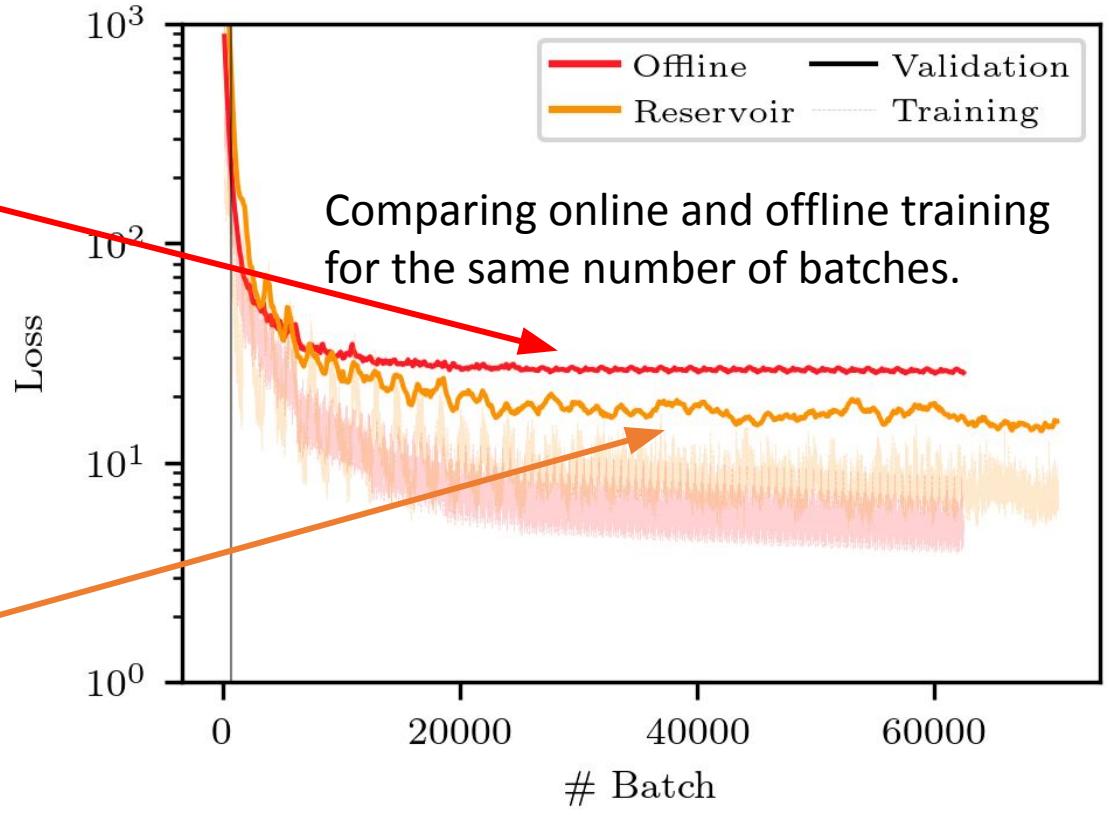
- **Offline & FIRO:** lower validation as #GPU increases: larger batch size & less optimization steps
- **Reservoir:** Not enough data, so compensate with already seen data, training on almost 5x samples, leading to a better validation MSE (but not time gains)

Buffer	GPUs	MSE	Time (*Hours*)
Offline	1	83.1	0.93
Offline	4	218	0.10
FIRO	1	135	0.083
FIRO	4	197	**0.082**
Reservoir	1	80.3	0.093
Reservoir	4	**65.0**	0.095

Unleashing Online Training

Offline training: 250 simulations, 100 epochs, **100 GB** dataset, **24.5h training on 4 GPUs**

Online training: 5 120 cores to run 20 000 simulations generating **8TB** of data processed online with **4 GPUs in 1.9 h**



Training online is 12x faster, and validation MSE 47% lower.

Cost

Resource	Cost (*€*)
CPU (*kh/core)*	6.36
GPU (*kh/GPU)*	382
SSD Storage (*TB)*	**59.4**

Source GENCI IDRIS

Training Setting	Dataset Size (*GB*)	Time (*Hours*)	Cost (*€*)
Offline with data generation	100	24.5	51.60
Offline without data generation	100	24.3	43.33
Online	**8,000**	**1.97**	63.8
Hypothetic offline	8,000	24.3	**512**

Cost

Resource	Cost (*€*)
CPU (*kh/core)*	6.36
GPU (*kh/GPU)*	382
SSD Storage (*TB)*	**59.4**

Source GENCI IDRIS

Training Setting	Dataset Size (*GB*)	Time (*Hours*)	Cost (*€*)
Offline with data generation	100	24.5	51.60
Offline without data generation	100	24.3	43.33
Online	**8,000**	**1.97**	63.8
Hypothetic offline	8,000	24.3	**512**

Cost

Resource	Cost (*€*)
CPU (*kh/core)*	6.36
GPU (*kh/GPU)*	382
SSD Storage (*TB)*	**59.4**

Source GENCI IDRIS

Training Setting	Dataset Size (*GB*)	Time (*Hours*)	Cost (*€*)
Offline with data generation	100	24.5	51.60
Offline without data generation	100	24.3	43.33
Online	**8,000**	**1.97**	63.8
Hypothetic offline	8,000	24H at least	**512**

Steps to move to on-line training

1. Instrument the simulation code
   - 3 functions API: melissa_init, melissa_send, melissa_finalize
   - Supports: Fortran, C/C++ and Python
2. Inherit a base server class and customize to your needs:

   Set the sampler for the simulation parameters:

```python
class HeatPDEServerDL(TorchServer):
    """Use-case specific server"""

    def __init__(self, config_dict: Dict[str, Any]):
        super().__init__(config_dict)
     # Get access to configuration variables:
        study_options = self.config_dict["study_options"]
        self.mesh_size = study_options["mesh_size"]
        Tmin, Tmax = study_options['parameter_range']
```

```python
        # Set random uniform sampling
            self.set_parameter_sampler(\
            sampler_t=ParameterSamplerType.RANDOM_UNIFORM,\
            l_bounds=[Tmin],\
            u_bounds=[Tmax],\
          seed=123
        )
```

Steps to move to on-line training

Data transformation from reservoir to batch:

```
@override
    def process_simulation_data(self, msg: SimulationData, config_dict: dict):
        field = "temperature"
        # cast msg.data to float32
        x = torch.from_numpy(
            np.array(msg.parameters[-self.nb_parameters:] + [msg.time_step], dtype=np.float32)
        )
        y = torch.from_numpy(msg.data[field].astype(np.float32))
        return x, y
```

Steps to move to on-line training

3. Adapt the configuration to your need (here set up config for using Slurm):

```
"launcher_config": {
        "scheduler": "slurm-semiglobal",
            "scheduler_arg_server": [
            "--qos=qos_gpu-dev",
            "--account=igf@v100",
            "--nodes=1",
            "--ntasks=2",
            "--gres=gpu:2",
            "--cpus-per-task=5",
            "--threads-per-core=1",
            "--time=01:00:00"
        ],
         "scheduler_arg_client": [
            "--ntasks=1"
            ….
        ],
        ….
```

Steps to move to on-line training

4. Start everything:

```
melissa-launcher --config_name my_config
```

Custom data management (usually at Reservoir side), may be needed depending on your training scheme for autoregressive models

$$u_X^{t+1} = f_\theta(u_X^t)$$

Conclusion

linktr.ee/melissa.inria

Online training for deep surrogates with Melissa: unlock training from very large data sets

- Gains on generalization, training time, GPU usage

Very soon: support for APE-Benchmark suite

(arXiv:2411.00180)

Project: support of SBI (Simulation Based Inference)

https://sbi-dev.github.io/sbi/latest/