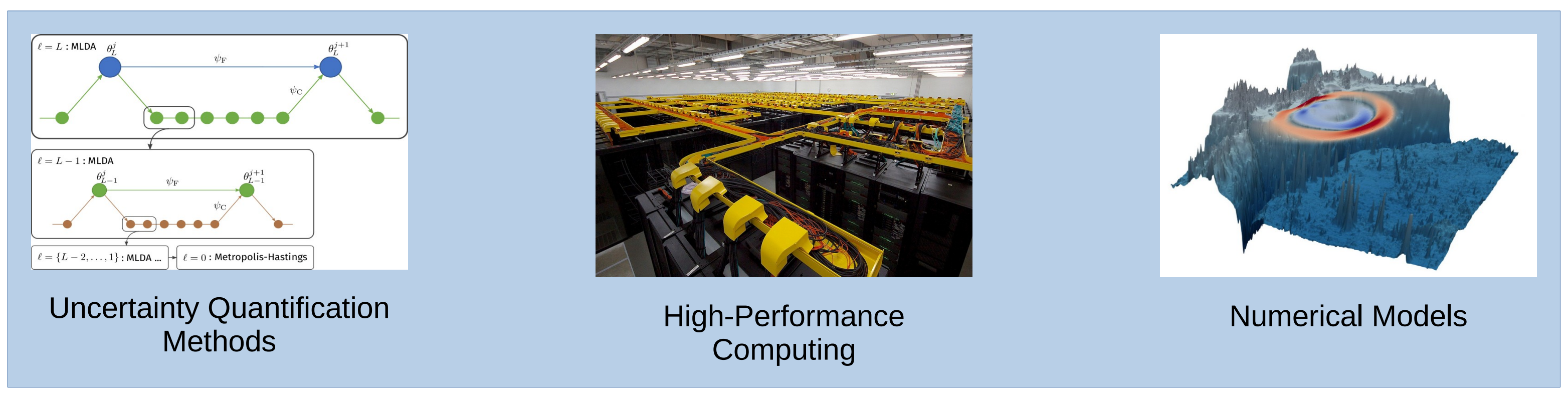
Advanced UQ Algorithms and Turn-Key HPC via UM-Bridge

Linus Seelinger[1], Anne Reinarz[2], Robert Scheichl[1]

Institute for Applied Mathematics, Heidelberg University[1]
Department of Computer Science, Durham University[2]

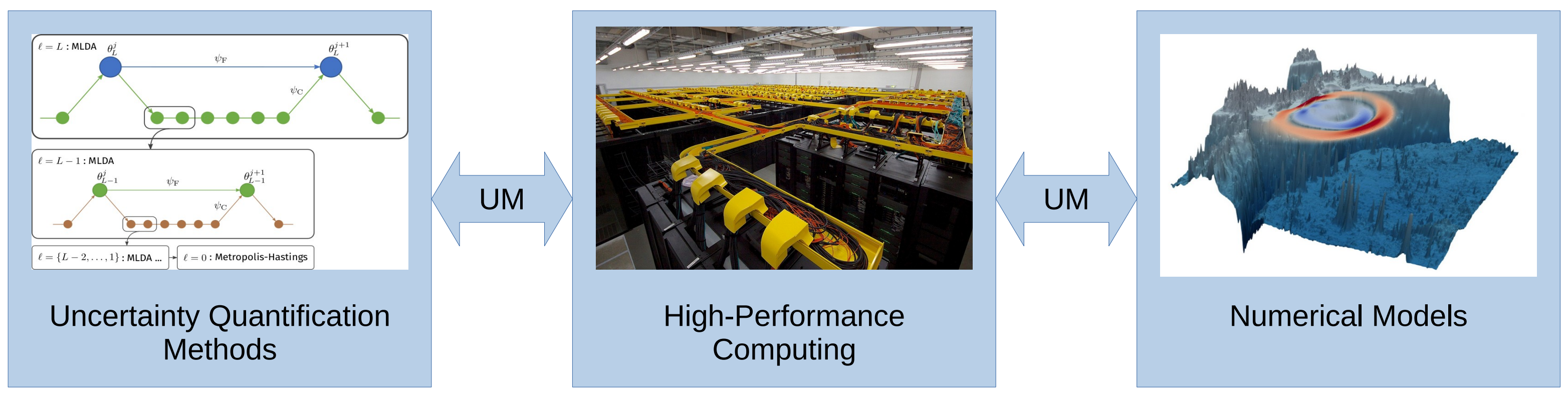
Accelerating UQ from Prototype to HPC

Realistic UQ problems can be extremely costly $\implies$ Need to combine best of UQ, numerical models and HPC to solve them!


Uncertainty Quantification Methods
High-Performance Computing
Numerical Models

The resulting complexity, need for expert knowledge across three fields and technical barriers (incompatible languages, parallelization paradigms etc.) are holding back both UQ method development and widespread adoption of UQ.

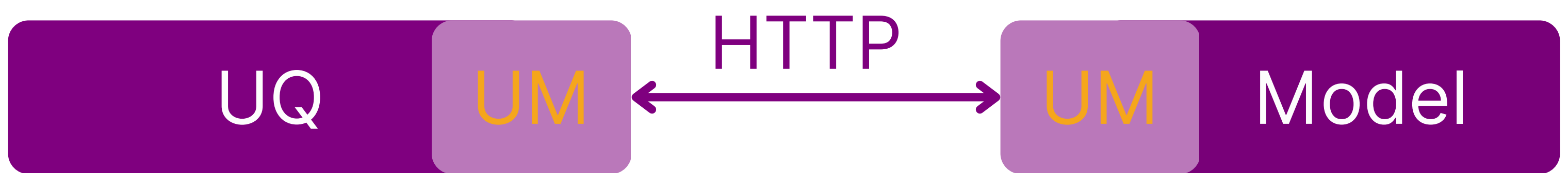
UM-Bridge introduces a microservice-inspired architecture for UQ software, offering separation of concerns between experts.


Uncertainty Quantification Methods
UM
High-Performance Computing
UM
Numerical Models

Gains: Can now link arbitrary UQ and model codes through a simple interface, define reproducible benchmarks, and transparently scale UQ applications to clusters.
$\implies$ UM-Bridge accelerates development along all stages and enables complex applications!

UM-Bridge

A model in UQ is often just a function $F : \mathbb{R}^n \to \mathbb{R}^m$ taking a parameter vector onto a model outcome. In addition, Jacobians or Hessians may be required.
UM-Bridge [1] provides this mathematical "interface" as an equally universal software interface:



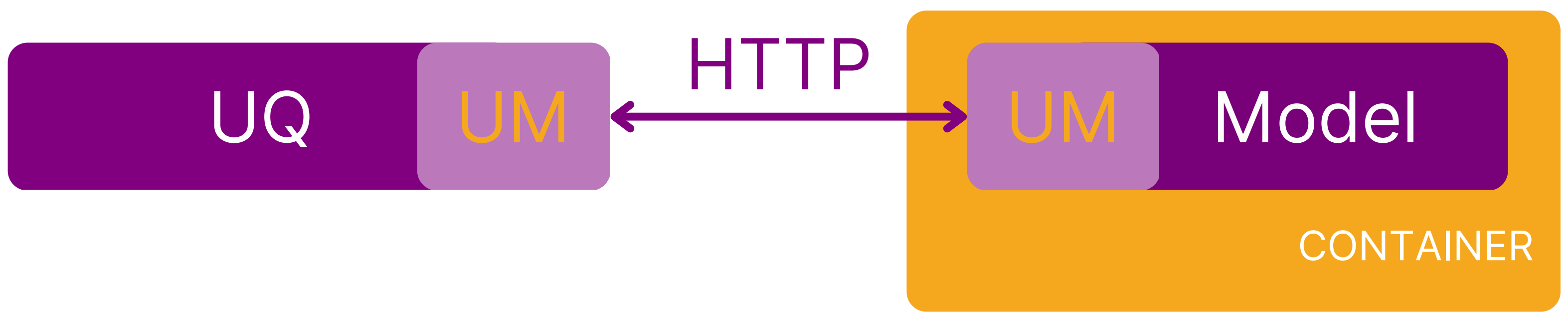
UQ and model are run as independent applications, exchanging model evaluation requests and results via UM-Bridge. Behind the scenes, UM-Bridge is using network communication, and can therefore link arbitrary programming languages and frameworks.

Easy to use integrations are available for C++, Python, R, and Matlab. Current framework support includes emcee, MUQ, PyMC, QMCPy, Sparse Grids Matlab Kit, tinyDA, and TT Toolbox.

With these integrations, implementing a model comes down to implementing a simple class, and UQ codes can query them through regular function calls.

Containerized Models

UM-Bridge models can be containerized for portability and reproducibility:



Containerized models can easily be published or shared between collaborators, and no machine-specific software setup is needed to run them.
$\implies$ Separation of concerns between model and UQ experts!

Example: Run the published tsunami model container (from the application on the right) on your system via:

```
docker run -p 4242:4242 linusseelinger/model-exahype-tsunami
```

Request model evaluation $F(\theta)$ for parameter vector $(18, 100)^\top$ from a Python script:

```
model = umbridge.HTTPModel("http://localhost:4242", "forward")
model([[18.0, 100.0]])
```

The exact same request can also be performed in C++ with no change to the model:

```
umbridge::HTTPModel model("http://localhost:4242", "forward");
model.Evaluate({{18.0, 100.0}});
```

UQ Benchmark Library

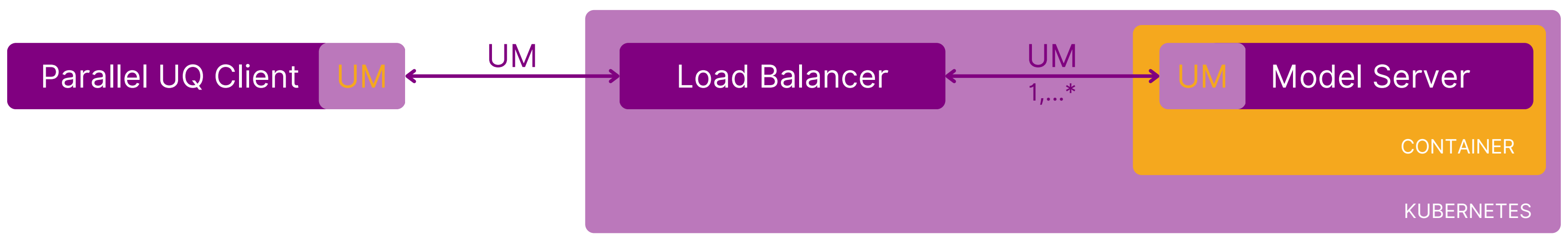
We provide a community-driven collection of UQ models and benchmark problems, all available as ready-to-run containers supporting UM-Bridge.

The library currently contains >20 ready-to-run models and UQ problems from >15 contributors across >10 institutions.

→ `https://um-bridge-benchmarks.readthedocs.io`

Turn-Key Cloud Computing for UQ

UM-Bridge provides a reusable kubernetes configuration for running parallel model instances in the cloud, automatically distributing work among them.

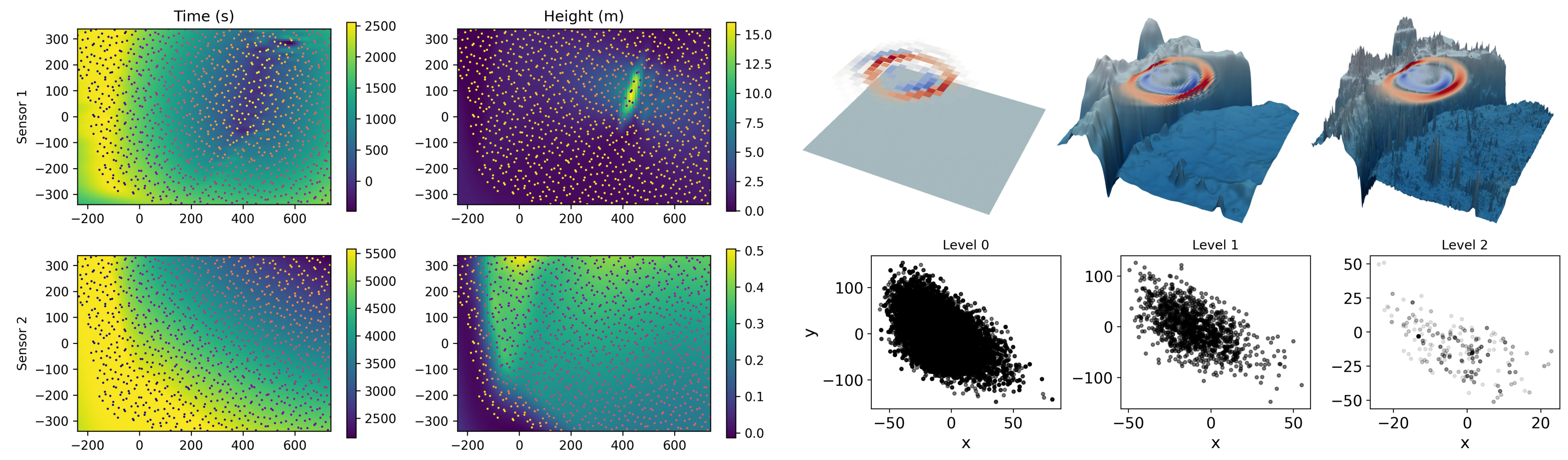


There is no need to adapt model or UQ code, since both are oblivious of the cluster. Simple thread-parallel UQ codes running on a laptop can now transparently offload model evaluations to HPC-scale clusters! The approach is proven to scale to thousands of processor cores [3].

Applications

Applications in [3] demonstrate the power of UM-Bridge. They involve a variety of UQ codes applied to a variety of models on different systems. No modification to UQ codes or models was necessary beyond a simple UM-Bridge integration.

Application	UQ	Parallelization	System		System	Model
L2-Sea	SGMK *Matlab*	Matlab parfor	Laptop	↔	GCP (48 cores)	L2-Sea *Fortran*
Composites	QMCPy *Python*	Threads	Workstation	↔	Workstation (k3s)	DUNE *C++*
Tsunami	tinyDA *Python*	Ray	Workstation	↔	Workstation GCP (2800 cores)	ExaHyPE *C++, codegen*



The tsunami model (top right) is coupled with a Multilevel Delayed Acceptance method implemented in `tinyDA`, using a Gaussian Process surrogate as coarsest level (left). The Gaussian Process surrogate was evaluated directly on the workstation, while costly numerical model runs were offloaded to a 2800 core kubernetes cluster on Google Cloud Platform. Using UM-Bridge, this setup was developed in a few days, while a previous monolithic approach [2] took months to complete.

Future Work

Support for SLURM job scheduling is planned, extending the parallel architecture from kubernetes to HPC systems. The Gaussian Process code from the tsunami application will be further developed into a reusable, general-purpose microservice between UQ and models. Finally, the benchmark library will be published soon.

Resources

Mail: `mail@linusseelinger.de`
GitHub: `www.github.com/UM-Bridge/umbridge`
Docs, tutorial and benchmarks: `https://um-bridge-benchmarks.readthedocs.io`

[1]	JOSS *2023*	UM-Bridge: Uncertainty Quantification and Modeling Bridge *L. Seelinger, V. Cheng-Seelinger, A. Davis, M. Parno, A. Reinarz*
[2]	SC '21 *2021*	High Performance UQ with Parallelized Multilevel MCMC *L. Seelinger, A. Reinarz, L. Rannabauer, M. Bader, P. Bastian, R. Scheichl*
[3]	Arxiv *2023*	Lowering the Entry Bar to HPC-Scale UQ *L. Seelinger, A. Reinarz, J. Bénézech, M. Lykkegaard, L. Tamellini, R. Scheichl*