Plume: A Plugin Mechanism for Earth System Models

Antonino Bonanni[1*], James Hawkes[1], Tiago Quintino[1]

(1) ECMWF; (*) antonino.bonanni@ecmwf.int

1. Plume (Plugin Mechanism)

Plume is a plugin mechanism that brings the flexibility of plugins to Earth System models. Developed at ECMWF for the Destination Earth initiative, plume allows the development of additional model functionalities as plugins.

Plugins are loaded at runtime and are granted (read)-access to model fields in memory to perform specific data processing tasks. Model fields are accessed through a well-defined interface provided by the Atlas library (github.com/ecmwf/atlas).

The biggest benefit of such mechanism is that plugins can perform processing tasks while data is still distributed in the computing nodes, avoiding to write model data to disk first. Saving I/O is particularly relevant for the very high-resolution Destination Earth simulations and for high time-frequency post-processing. Plume API is currently available for both C++ and Fortran programming languages.

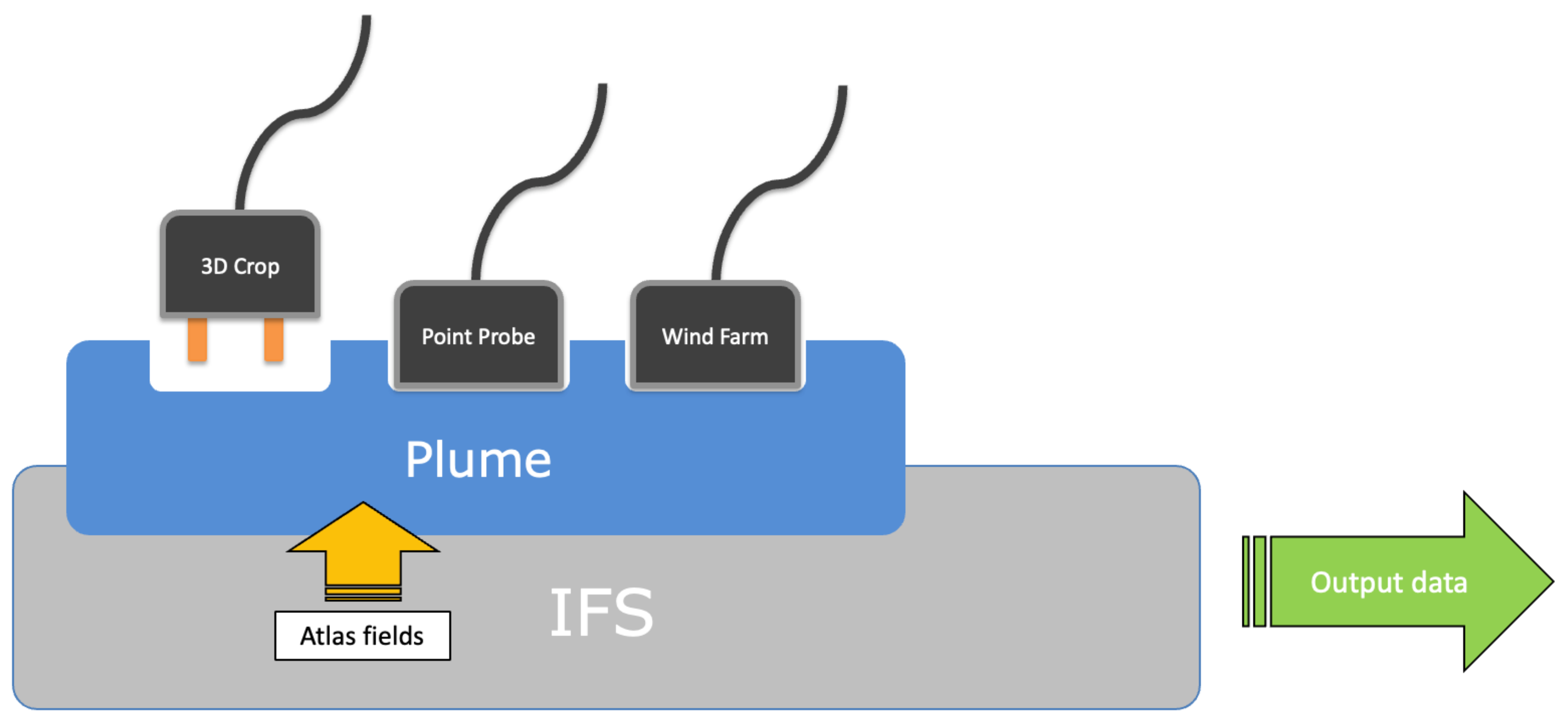
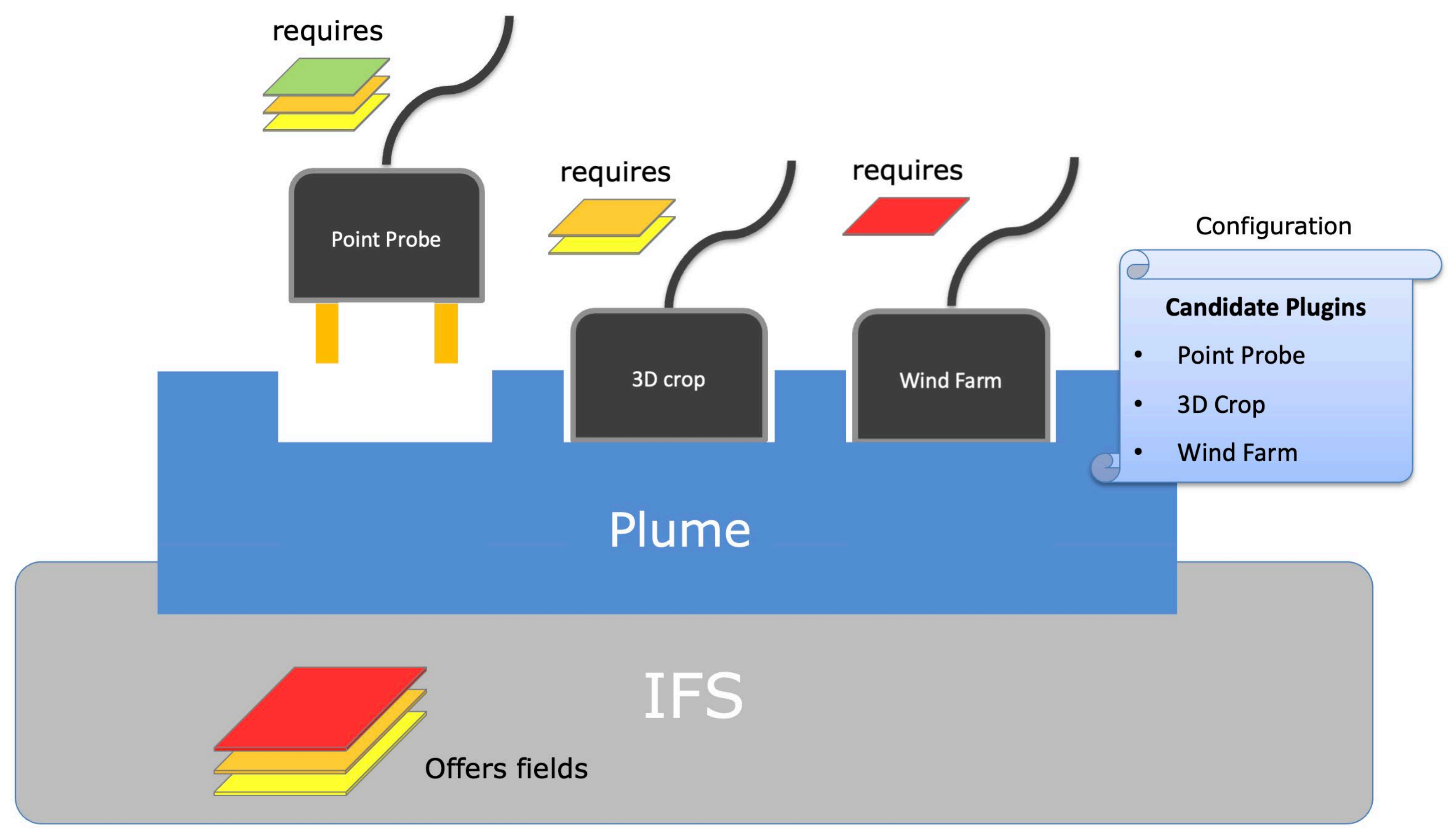


Diagram of plume library interfaced with ECMWF Integrated Forecasting System (IFS). Numerical fields are passed on to plugins through Atlas library.

- **What is plume?**
  - An open-source plugin system github.com/ecmwf/plume
  - Designed for Earth System Models
- **What can a plume plugin do?**
  - Read and process model data in memory
  - Implement a specific model functionality
- **Why plume?**
  - Modular, exchangeable and re-usable plugins
  - Interfaces available for C++ and Fortran
  - Well-defined interface to model data through Atlas github.com/ecmwf/atlas
  - Access data directly from model memory => no unnecessary I/O!

3. Loading Plugins Through Negotiation

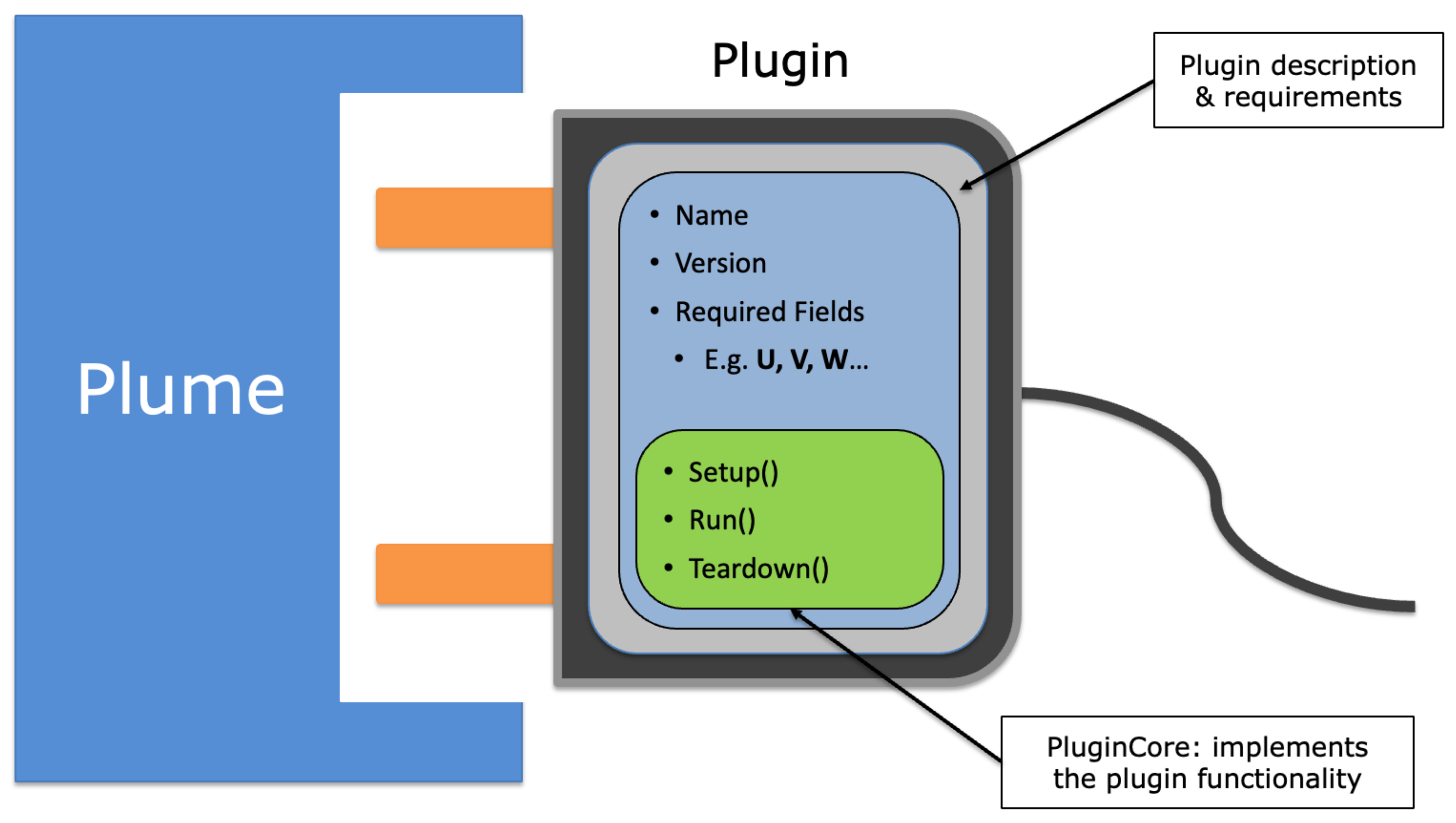
Plume plugins are loaded at runtime upon successful negotiation with the plume “manager”. During negotiation, each candidate plugin declares which parameters it requires to run and the manager cross checks them against the parameters available for the simulation. Only plugins whose requirements can be met are set “active” and run. A diagram of the negotiation phase is depicted here below.



Example of plume negotiation mechanism. Before the simulation begins, the plume manager checks the requirements of each plugin and decides which plugin is eligible to run according to the available parameters.

4. Plugin Structure

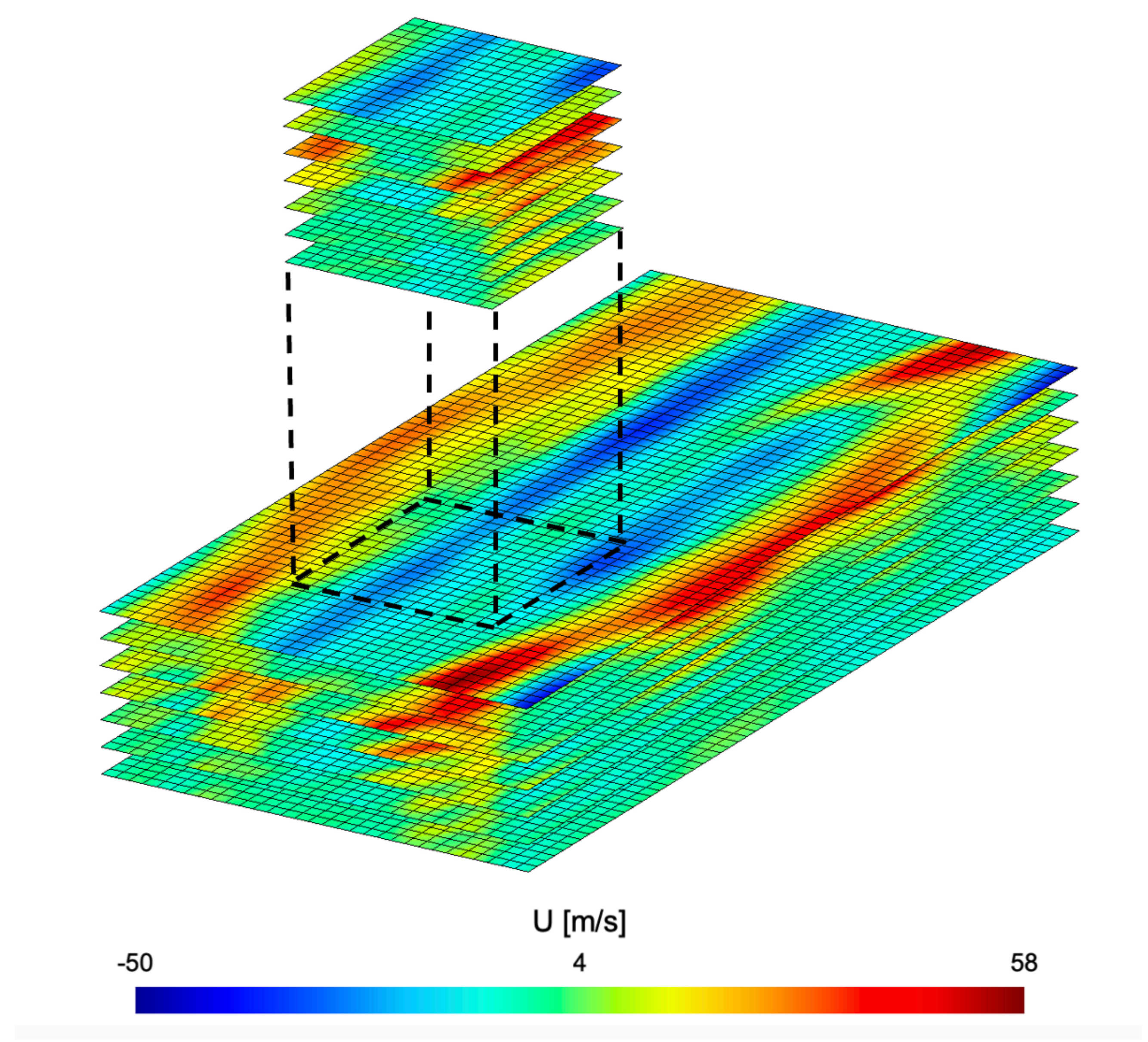
A plume plugin is a dynamically loaded library that implements a specific data-processing functionality. Plume interacts with the plugin at the beginning of the simulation to decide whether the plugin is eligible to run (see Negotiation section). Upon successful negotiation, the plugin runs along with the model and perform its core functionality (diagram below).


Plume
Plugin
Plugin description & requirements
Name
Version
Required Fields
E.g. U, V, W...
Setup()
Run()
Teardown()
PluginCore: implements the plugin functionality

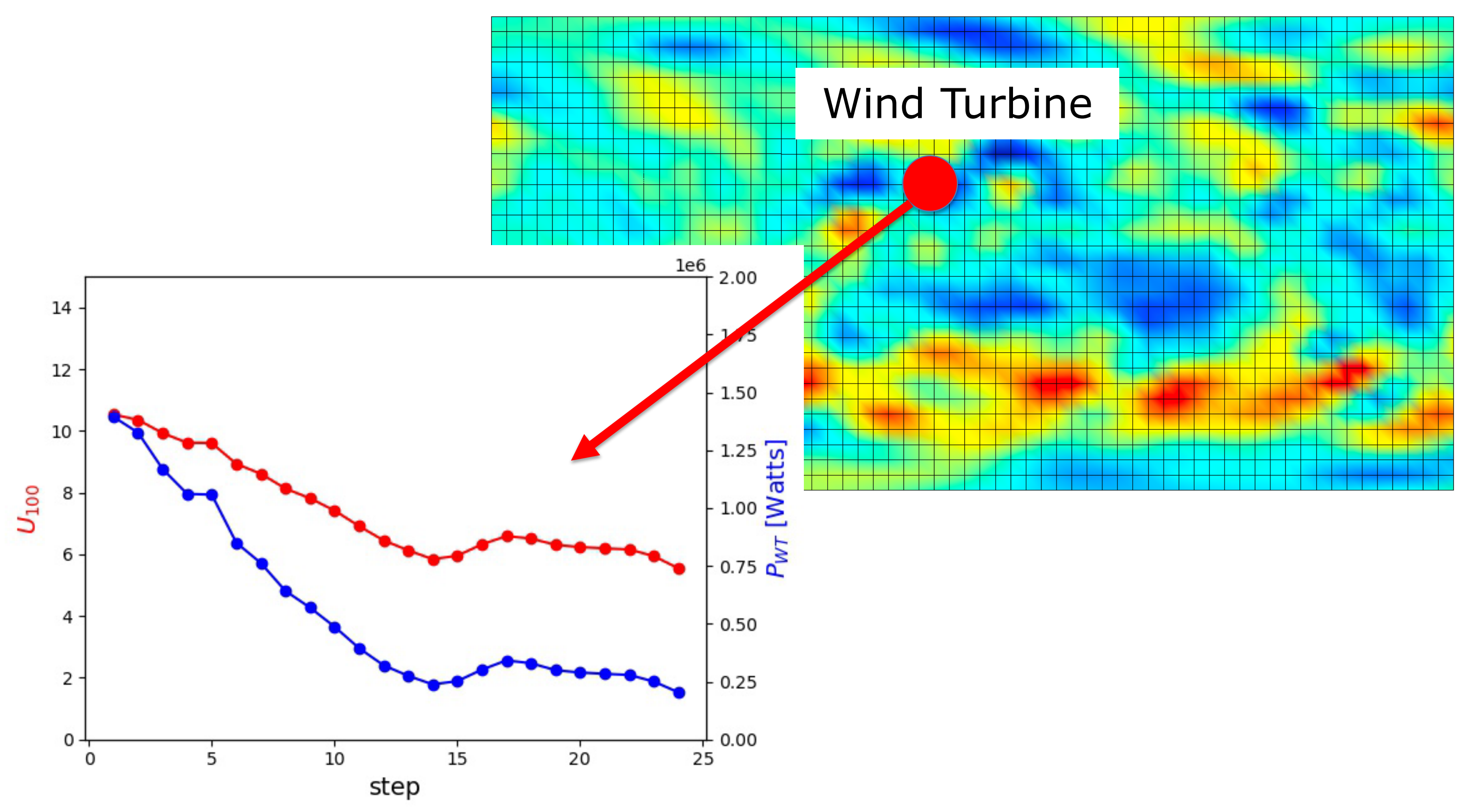
Schematic structure of a plume plugin. The plugin declares which parameters are needed to perform its task and implements a specific functionality to be run along with the Earth System model.

5. Initial Testing

Plume has been tested on several HPC systems and various potentially useful plugin examples have been explored and open sourced (github.com/ecmwf/plume-examples). This section presents some of the initial results obtained by running plume interfaced with IFS on the LUMI system for the first Destination Earth Minimum Viable Product (MVP) run, successfully completed in June 2023.



Example of 3D area cropping plume plugin: it uses Atlas API to perform a 3D cropping of the wind field (at all vertical level) in a user-defined area of the global field.



Example of plugin that implements a simplified wind farm model. It extracts the wind components at 100m height on user-defined locations and runs a model that predicts the generated power according to the wind turbine specifications.

Funded by the European Union
Destination Earth
implemented by ECMWF esa EUMETSAT