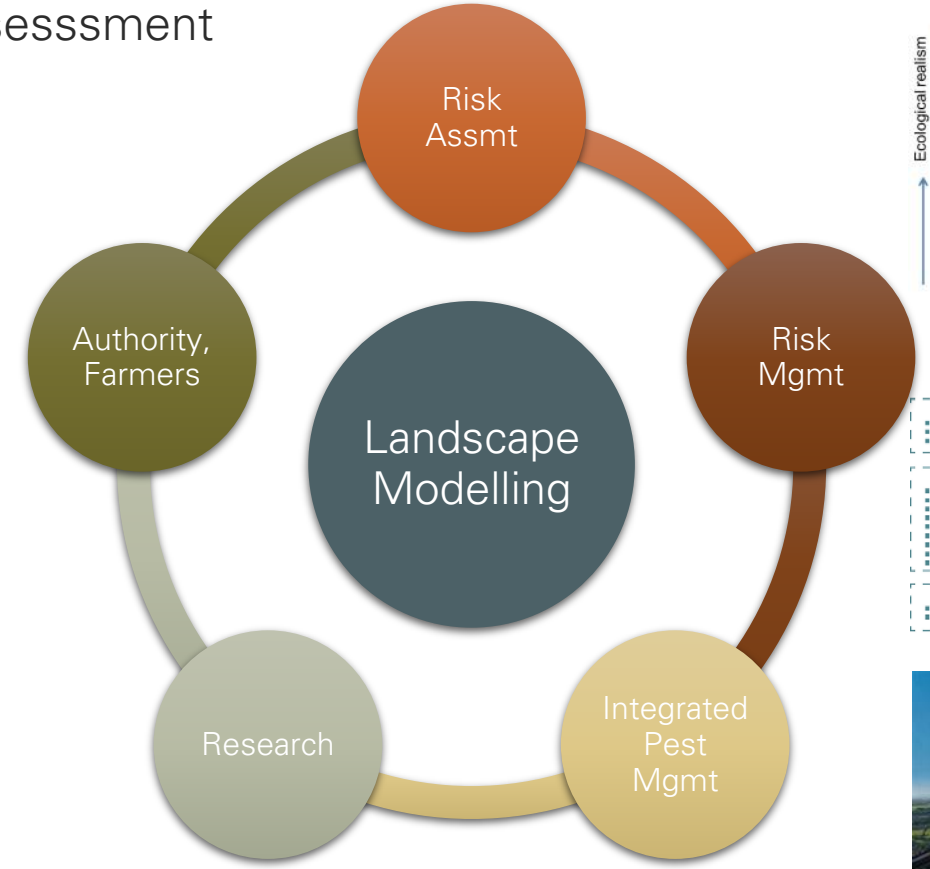

Thorsten Schad (thorsten.schad@landwerk-ev.de)
Sascha Bub (Sascha.bub@rptu.de)
Julian Heinrich (julian.Heinrich@bayer.com)

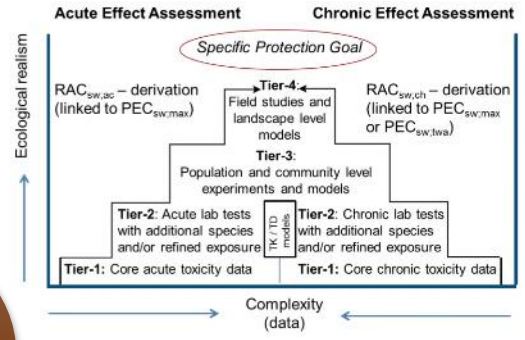
# xLandscape

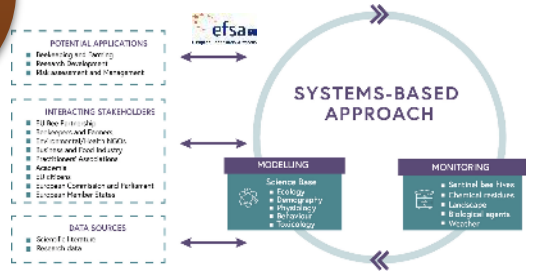
## Modular Landscape Models

# Demand for Landscape Modelling

- **More realistic, higher-tier** risk assesssment
  EFSA: Reference scenario in tiered schemes
- **Improved integration** of processes
- **Holistic view** to pesticide risk, eg,
  - multiple stressors, recovery, **biodiversity**
  - **indirect effects** of weed control
  - integration of **monitoring** and modelling
- **Digital Agriculture**
  integrated pest control & environmental risk
- **Digital Twins** (eg, DestinE) - what if..

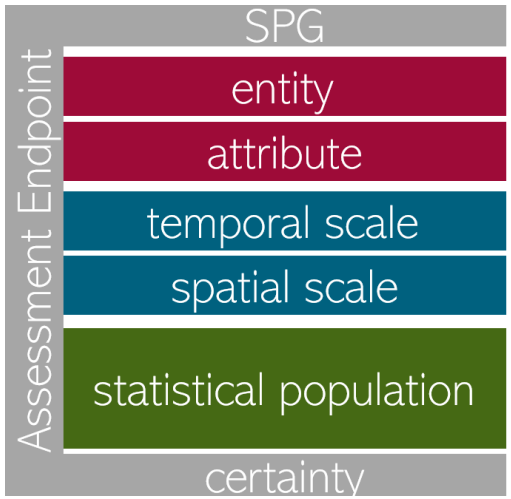


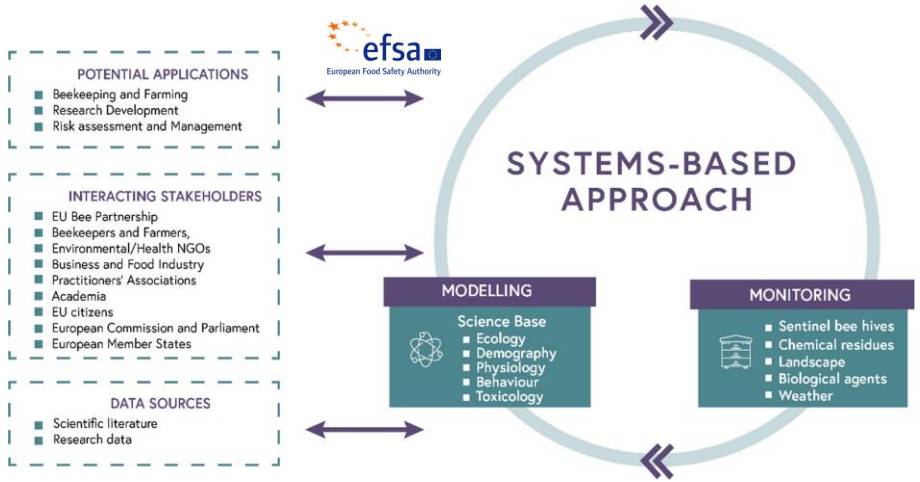

# Drivers for the xLandscape Development


established ~2000




started ~2010

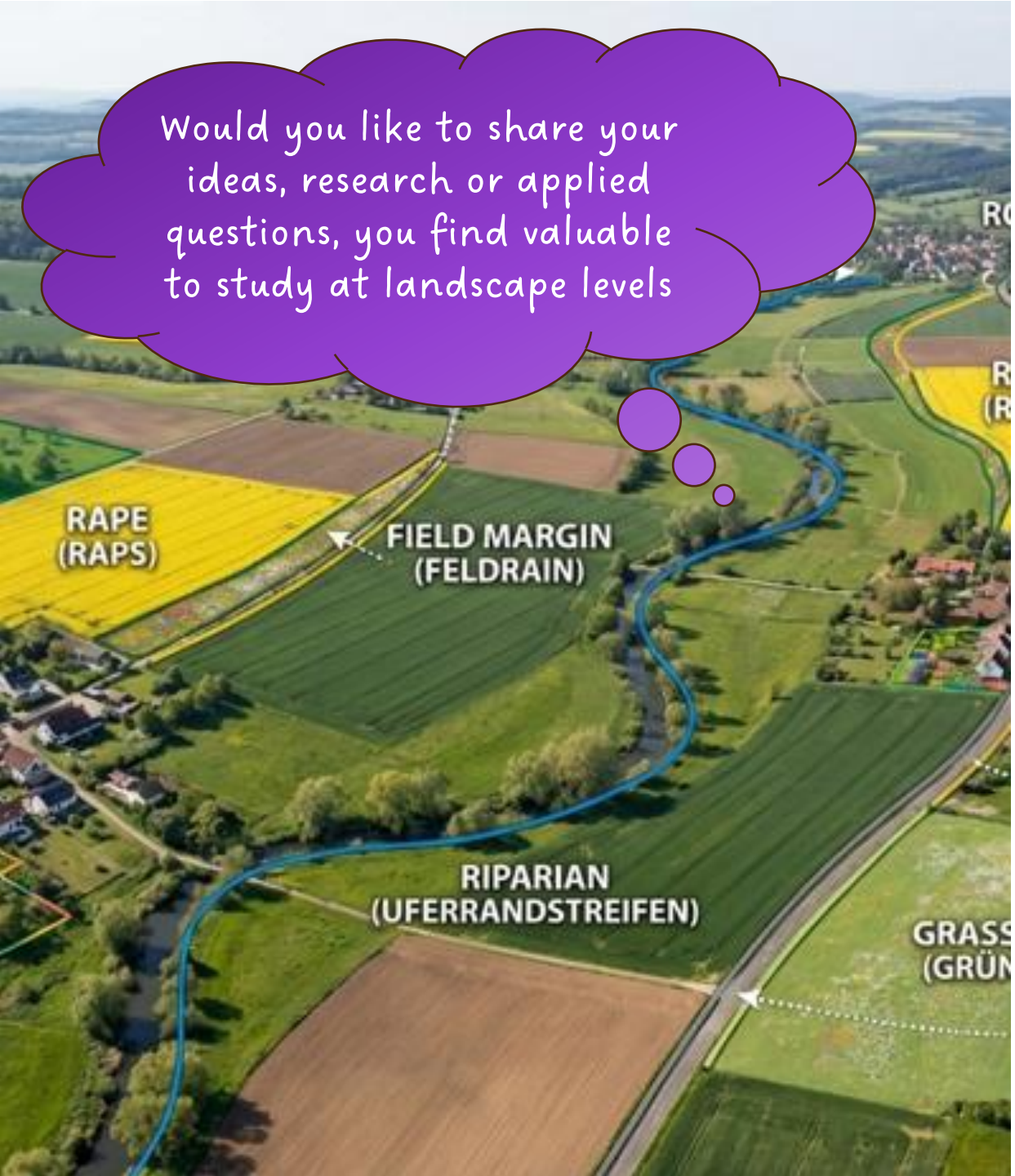
*SPG=Specific Protection Goal




the future: Systems Approaches and Digital Twins

Key concepts driving the xLandscape design:

- **Tiered approach** →flexibility, **adaptivity**, '*get only as complext as necessary*'
- **Specific Protection Goals** →operate in explicit **dimensions** and **scales** (spatial, temporal), explicit **structure** of analysis and communication




# Imagine

you are developing a **new species effect model** (eg, amphibian, wild pollinator, beetle) for pesticide risk assessment or conservation.

- you are interested in **multiple pesticide** exposure and effects across large regions
- you want to **compare risks** of pest control for different **baseline** assumptions
- you study real-world effects of different **risk management** and **landscape design** options
- you want to build a **regional scenario** (or digital twin) to study protection goals and ecosystem services you want this work **consistant** with related projects

if you do this from scratch, it's **quite an effort**

now imagine **you can focus on your craft and can make use of open shared work of your colleagues**

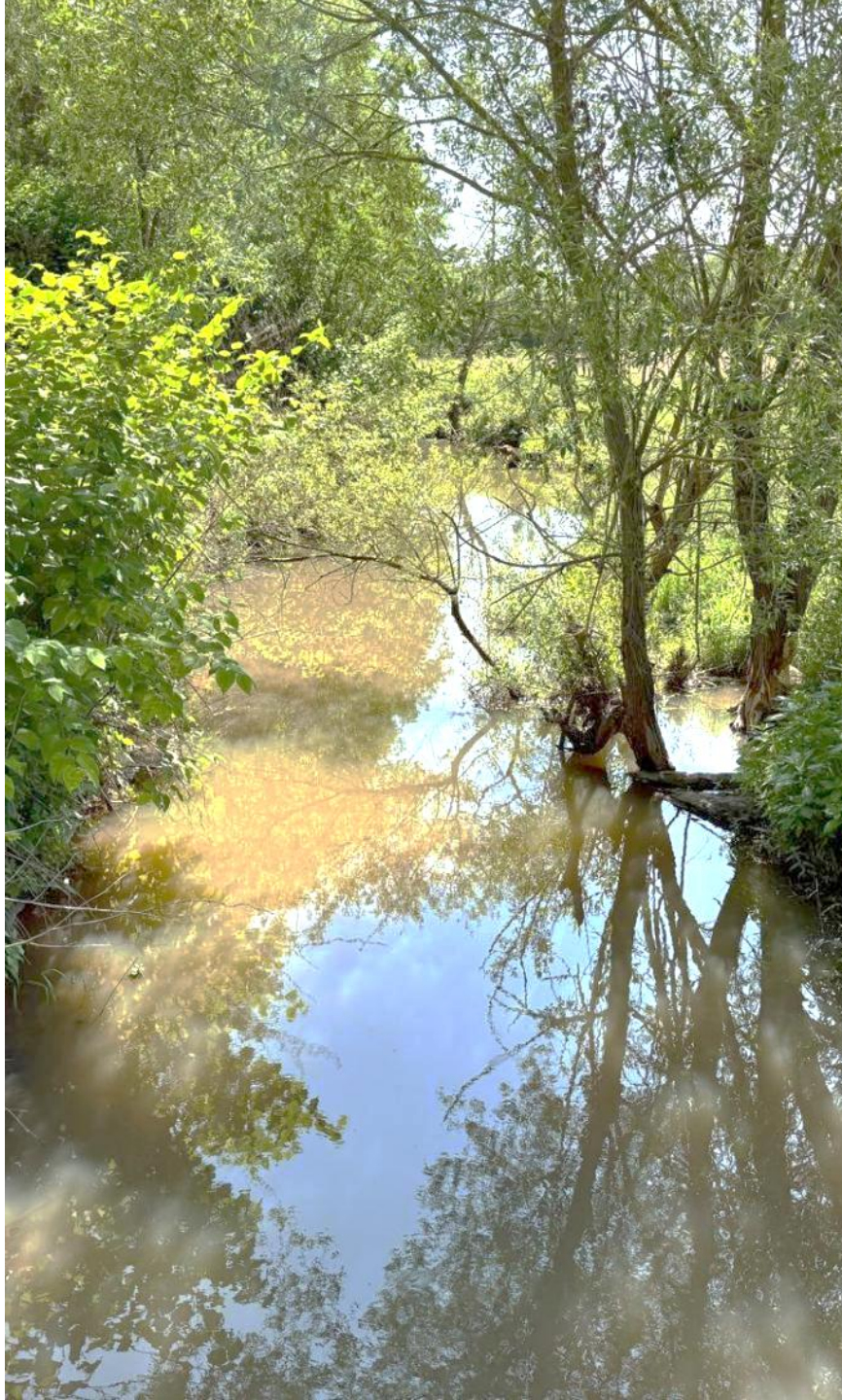
# Imagine...
# Our Vision - why we develop

You have a tool at hand **enabling you to build and run processes and models at landscape-level, using real-world data**.
You can **adapt** this tool to your problem. It's open source.
You can make use of open developments done by your **colleagues**.
You can run the landscapes models on your **laptop or large cloud** systems.

The **conceptual foundation** of the tool is embedded into regulatory-scientific frameworks. It also comes with **tiered scenario development**,
allowing you to **start quickly** on screening levels and to get more detailed as you proceed. **Scenario services** support you on request.

You can keep **consistancy, freeze and versioning a certain status** for full long-term availability and **reproducebility with a regulatory context**.

→ this is why we think an open, modular and collaborative landscape modelling framework is the route to take (→xLandscape)

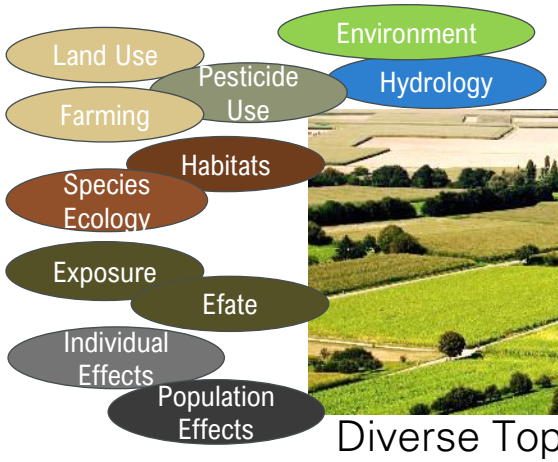


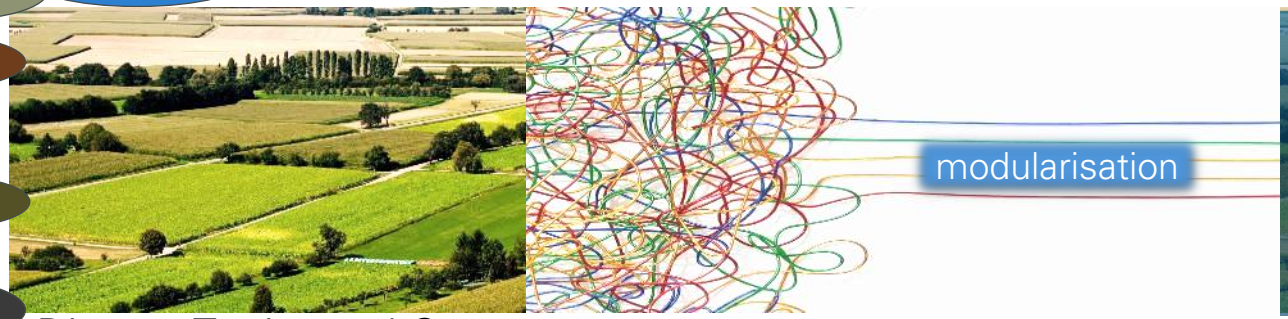
# Modularity enables

do you share positive effects of modularisation?!

## Collaboration

Focus on key expertise
Build on existing work



Diverse Topics and Competences
on landscape level

modularisation

## Harmonisation

use similar solutions to similar problems

## Transparency

Stepwise Validation

**Flexibility** – adapt model complexity

use established models

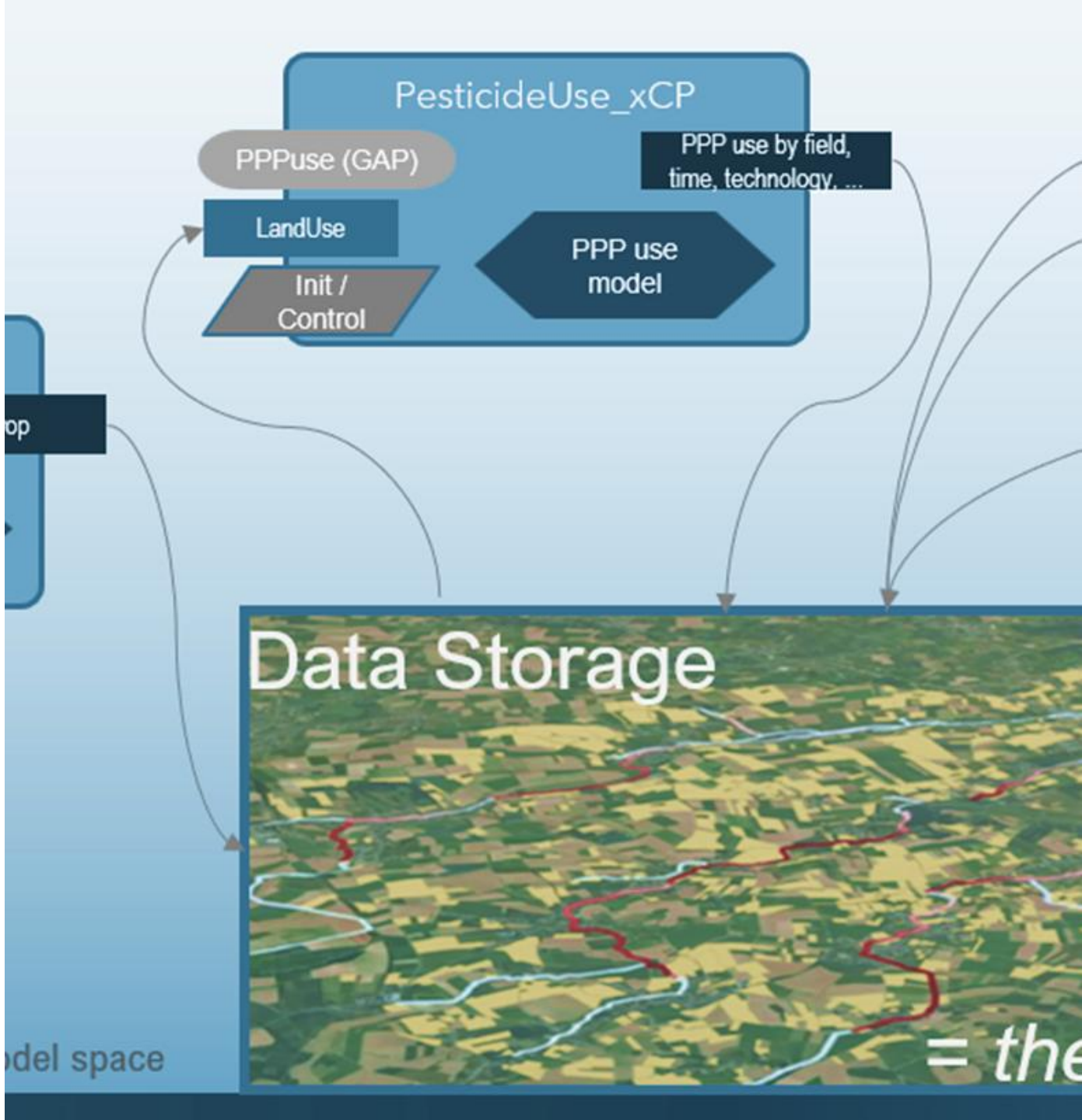
**Consistant Integration**

moduls can become certificated APIs*

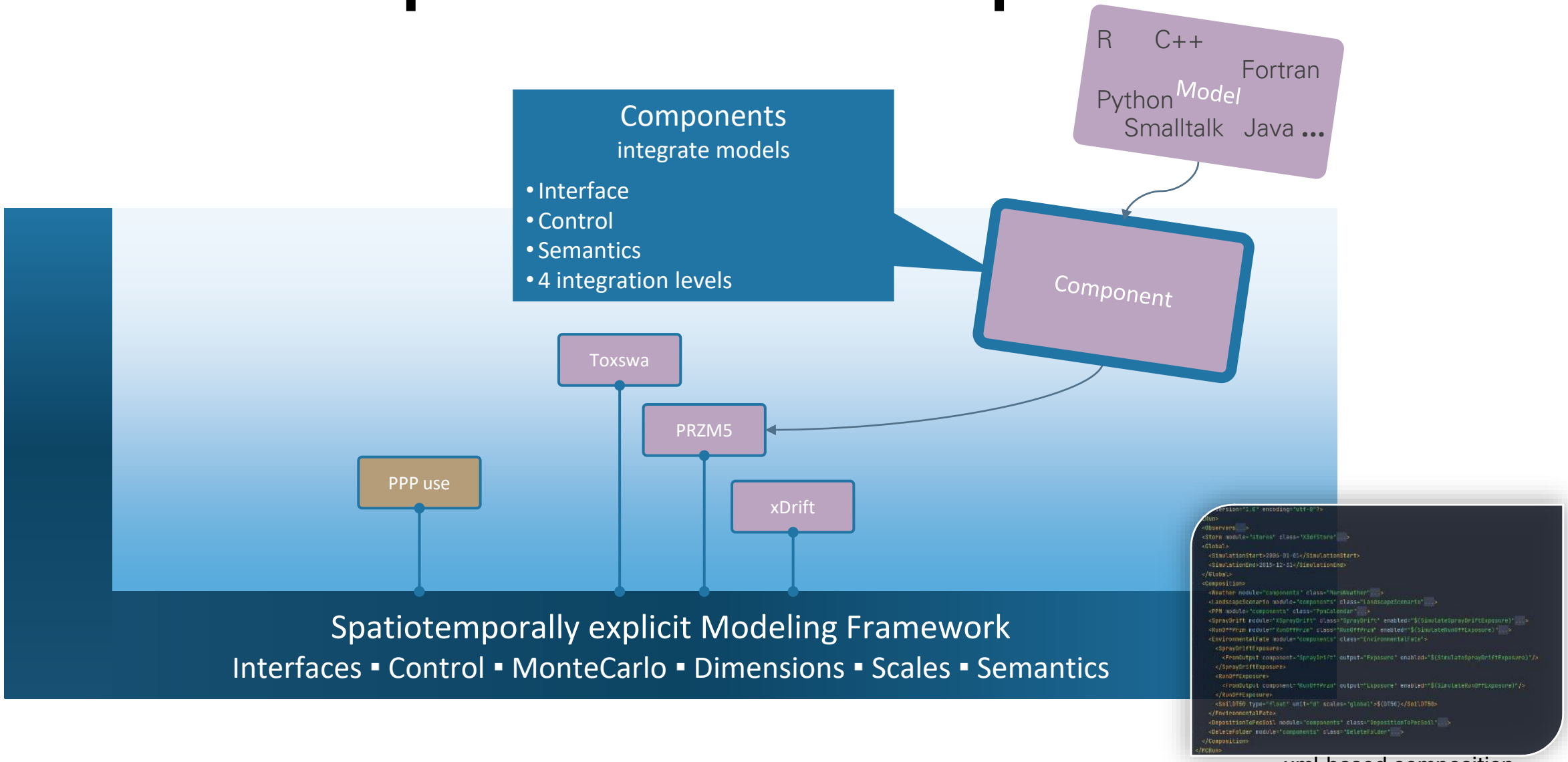
*Application Programming Interfaces

# why
# do we develop a modular landscape modelling framework

simply,
because there is a **demand**
and there is
**nothing like this available**



# Model Composition: Components

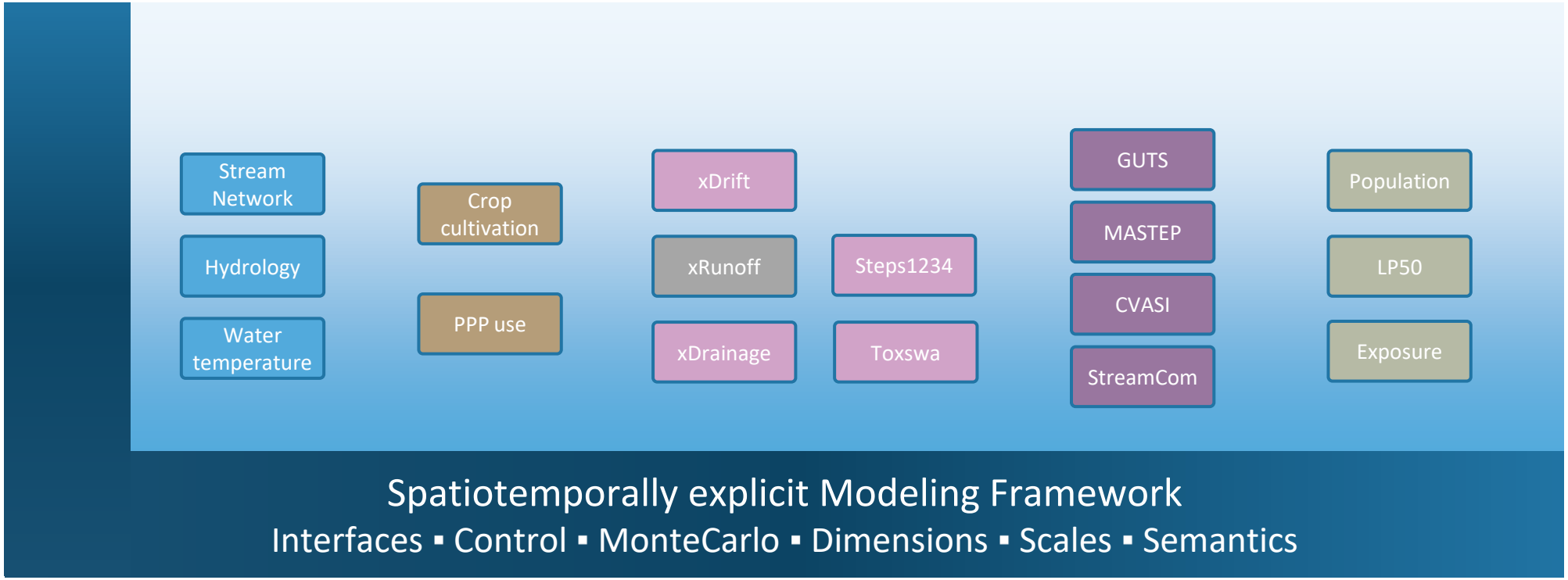



xml-based composition

# The Modelling Approach

## xAquaticRisk

- ✓ **Modular**
- ✓ **Open Source, FAIR**
- ✓ **Makes use of regulatory established models**
- ✓ **Flexible, adaptable**

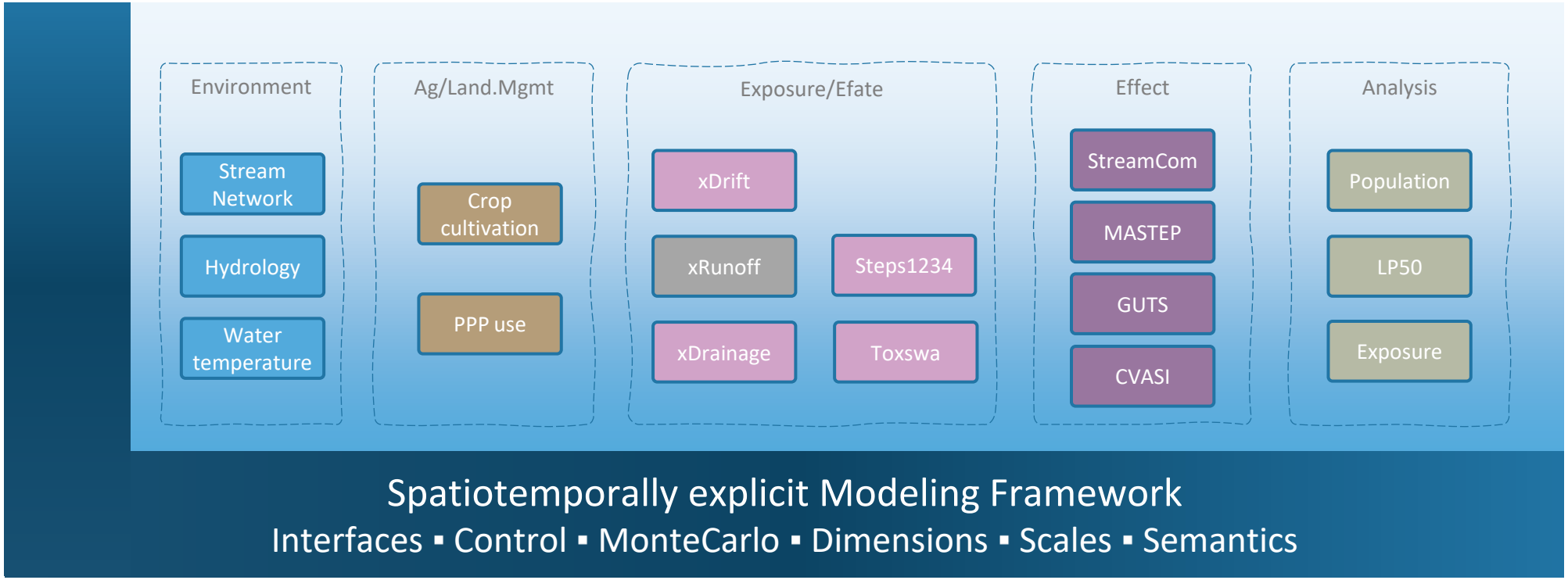



(https://github.com/xlandscape/xAquaticRisk)

# The Modelling Approach

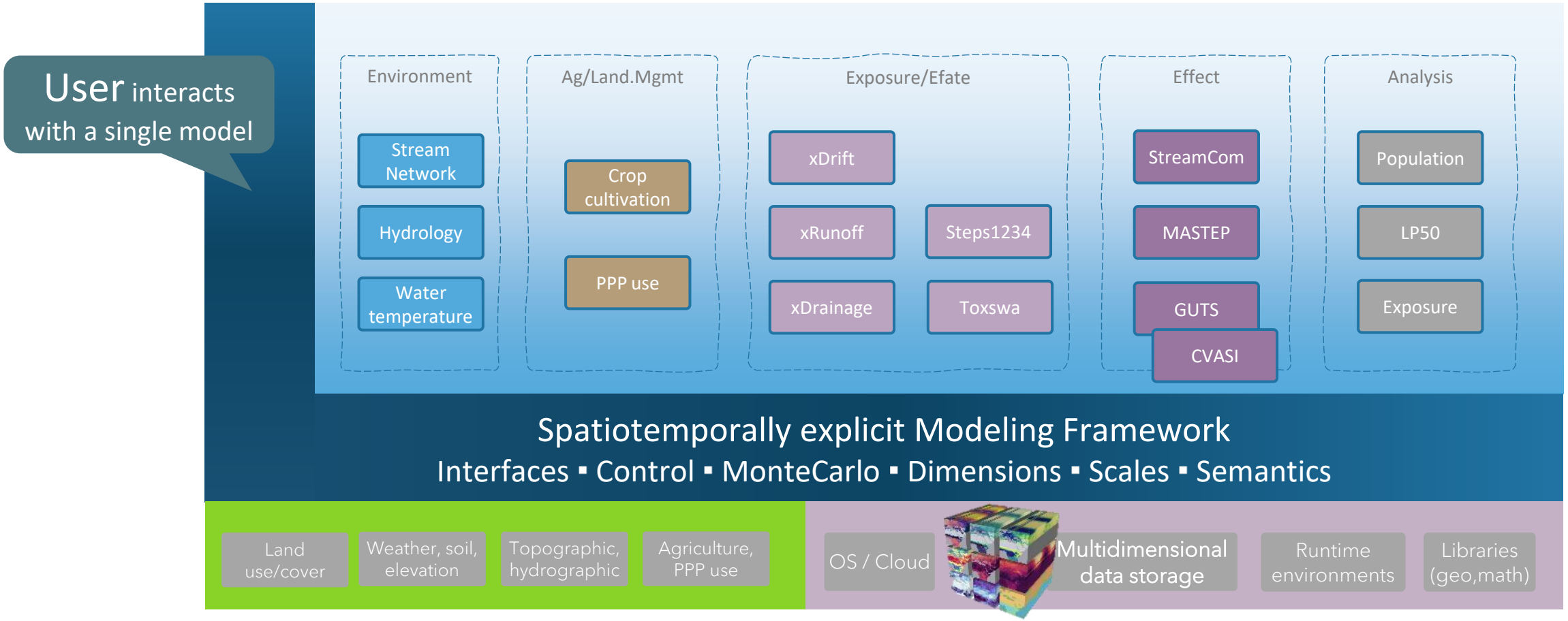
## xAquaticRisk

- ✓ Modular
- ✓ Open Source, FAIR
- ✓ Makes use of regulatory established models
- ✓ Flexible, adaptable

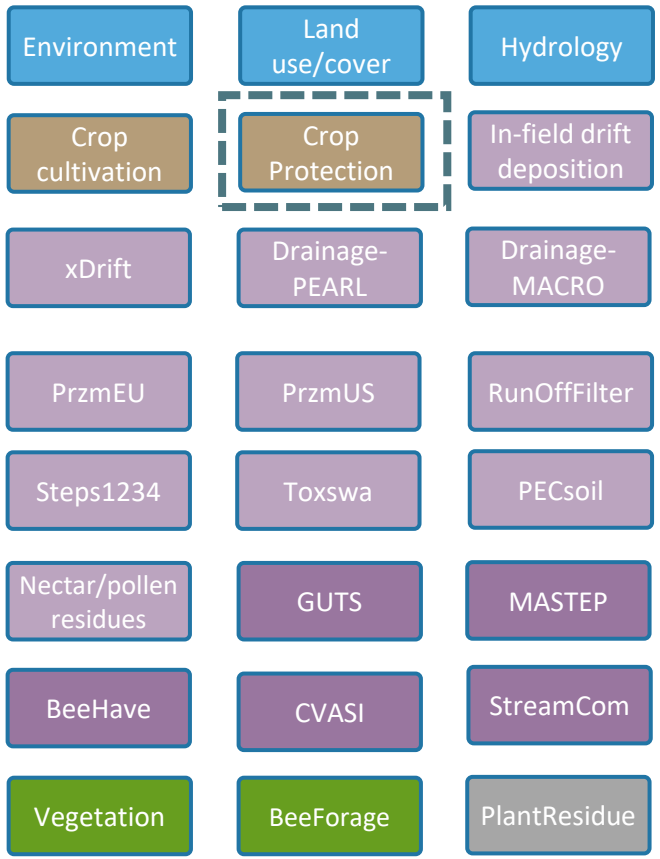


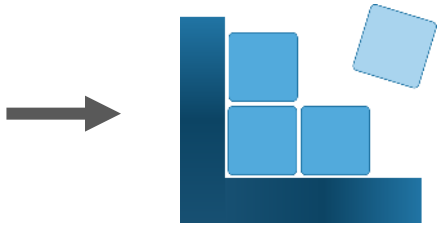

(https://github.com/xlandscape/xAquaticRisk)

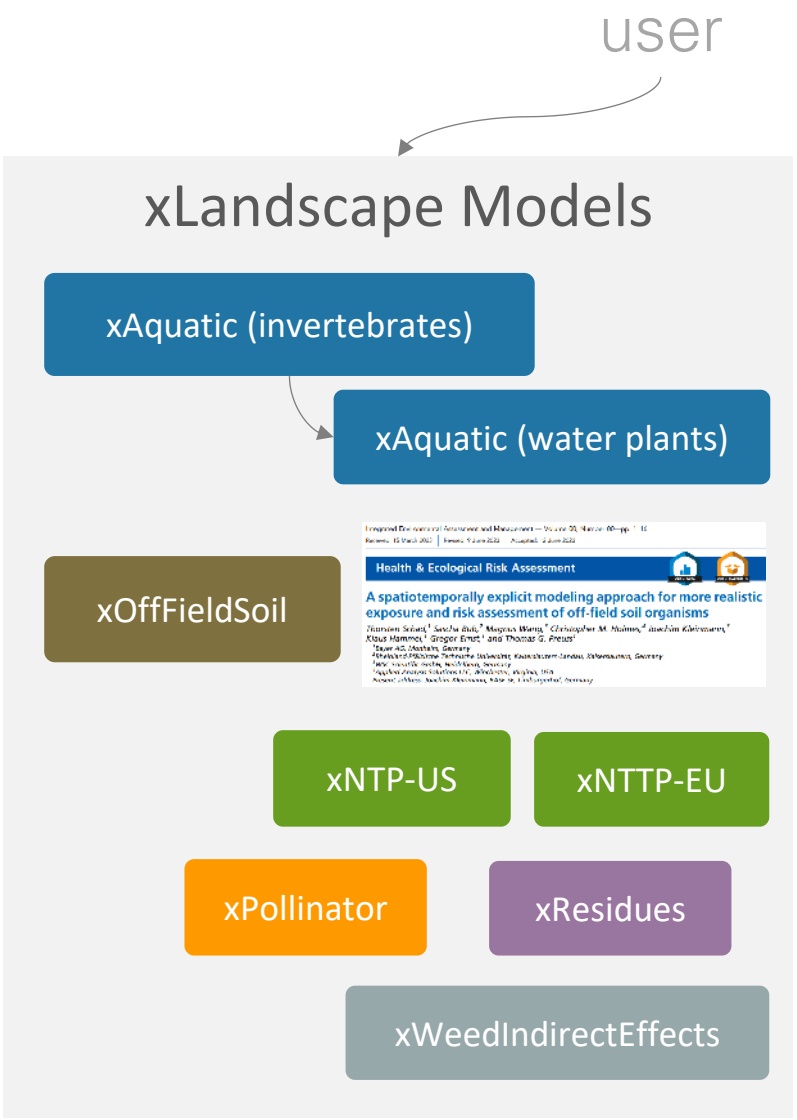
# Model Composition →xAquaticRisk



# Components & Models

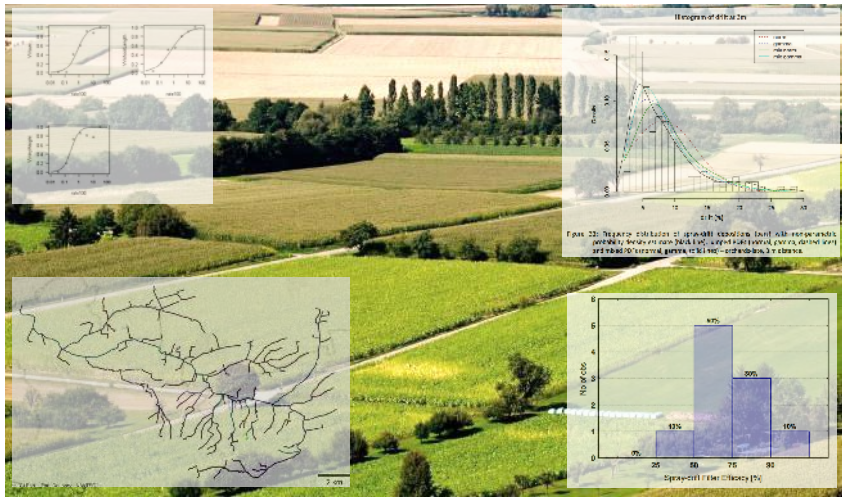


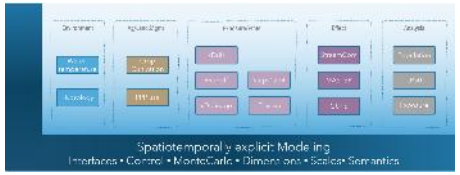



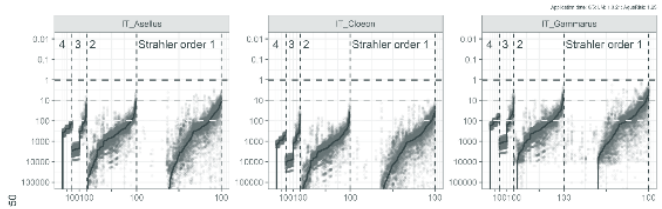



https://github.com/xlandscape

# Fundamentals:
## Propagate real-world variability to model prediction






### Hybrid Model (explicit)

- **Discretisation** (eg, space, time)
- **Monte Carlo** (PDFs*)

### Input variability, eg

› **Agriculture**: land use, pesticide use

› **Environment**: weather, soil, habitat conditions

› **Biology**: species occurrence, life cycle

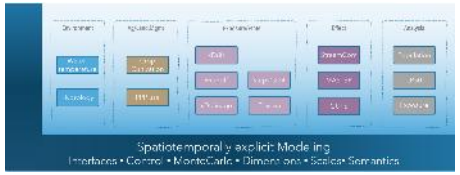
### Output variability, eg

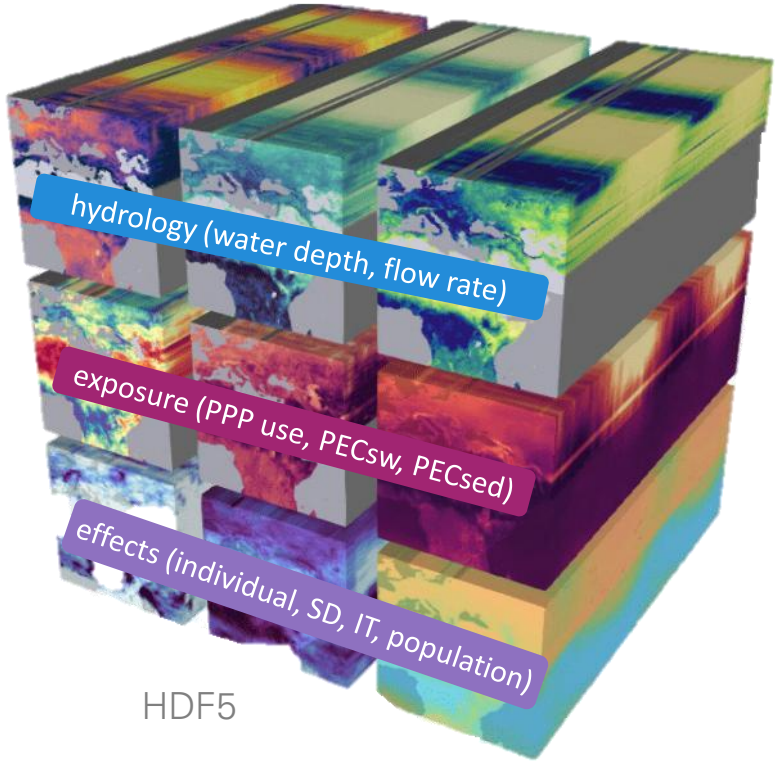
› **Exposure**: PECs, residues

› **Effects**: mortality, growth, population size

› **Environment**: bee forage

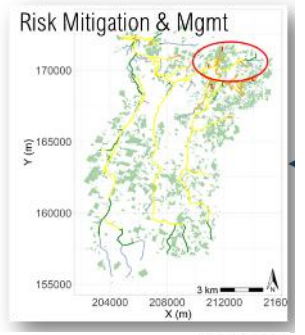
# Technology:
# Multidimensional data arrays

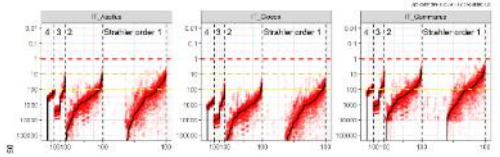


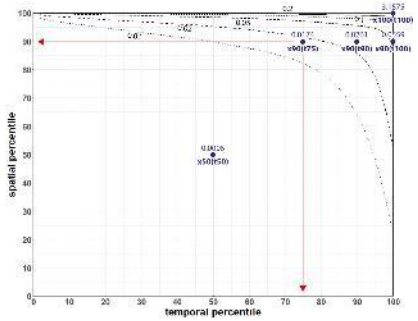




HDF5

R, Python, Jupyter, Matlab, Tableau, Knime, ...






https://www.hdfgroup.org/ - Originally developed at the U.S. National Center for Supercomputing Applications, it is supported by The HDF Group, a non-profit corporation whose mission is to ensure continued development of HDF5 technologies and the continued accessibility of data stored in HDF

# xLandscape User Level and Roles


training focus

| | |
|---|---|
| User - API | •an xLandscape model as backend, called via API (Application Programming Interface)<br>•also: an xLandscape model as a module, embedded in other environments, eg, Digita |
| User - Trained | •Trained xLandscape model user, with basic technical expertise, but within the need for expert domain knowledge<br>•Pre-prepared model parameterisation, execution and compilation of results<br>•In case, the role might include scenario development, requiring geospatial expertise |
| User - Expert | •Ready-touse xLandscape model application (eg, xAquatic), parameterisation (configuration) and execution<br>•Expert knowledge on xLandscape model domains (eg, xAquatic, xOffFieldSoil, xPollinator)<br>•In case, the role might include scenario development, requiring geospatial expertise |
| xLandscape Model Builder | •Expert to build ready-to-use, specific landscape models from using the xLandscaep core and components (eg, xOffFieldSoil, xAquatic)<br>•Note: this role will particularly profit from AI-support |
| xLandscape Component Developer | •Component developer (integrating models/modules, eg, Toxswa, Guts, StreamCom, Mastep, PRZM, CropProtection)<br>•Expertise in specific model to be integrated and xLandscape component development (concepts, interface, python) |
| xLandscape Core Developer | •Core xLandscape developer. Conceptual development and implementation.<br>•Advanced expertise, eg, risk assessment domain, Informatics, semantics |

# xLandscape-based Model Examples


### Aquatic
- *xAquatic*
- *Catchment-level RA for*
- *Invertebrates*
- *Plants*
- *Hydrological scenarios*
- *Ag, Rails, Industrial Vegetation Mgmt*

### Off-Field-Soil
- *xOffFieldSoil*
- *landscape-level exposure and effects*
- *addressing the EFSA SO for soil organisms*

### Pollinator
- *xPollinator*
- *bee forage occurrence in space and time*
- *Scenarios for BeeHave (ApisRAM)*
- *work in progress: bee exposure*

### NTA
- xNTA
- Spatiotemporal exposure pattern, TER, RQ
- work in progress:
  - NTA-Baseline scenarios
  - xBembidion Component

### NTTP
- xNTTP
- landscape-level exposure, TER, RQ, effects
- link to grassland model IBC
- xNTP-US: refinement to EPA worst-case approach

### Insect Decline
- xToxicLoad
- insecticide toxicity
  - in space and time
  - in landscapes
  - due to multiple PPP uses

### Birds & Mammals
- Tier4 new EU Guidance
- work in progress

### Residues in Plants
- xKnowRes

# Next

## Community

Users and developers. Connect to research, develoment and risk assessment teams sharing the demand, thinking and working along the same lines. Trainings. Workshops. Support.

## Readyness and Access

Documentation of xLandscape, Components and Scenarios. Testing.
Simple download and getting started. Simple access to case studies, eg via web.
Scenario development.

## Adapt AI

AI support and integration in all aspects, eg, building Components, Landscape Models and Scenarios.
Model parameterisation (easy ai-based and formal access to pesticide data), execution and analysis, report writing, etc.

## Development

of Components, Models and Scenarios. Maintenance.
xLandscape Version-4: clean architecture (microkernel and component-based software), improved autonomy of components, Digital-twin readiness, coupling of monitoring and modelling for systems-approaches.

# Our Values

## Openness & Collaboration

We believe in **open source** principles, making our models and components accessible for everyone. This **fosters innovation**, enables **participation**, and ensures that solutions are tailored to real-world challenges rather than built in isolation.

## FAIR Principles

We follow the **FAIR guidelines**—Findable, Accessible, Interoperable, and Reusable. By designing modular frameworks and **sharing transparently**, we empower researchers and stakeholders to build on existing work and accelerate progress.

## Invitation to Participate

xLandscape development come with a **mindset**. We **invite everyone** to contribute ideas, share expertise and help shaping the future of landscape modelling for sustainable development.

# Take Home

**xLandscape Modelling =** improved **realism & integration**

**Modularisation** manages complexity and enables **collaboration**

**xLandscape** = **modular landscape modelling approach**

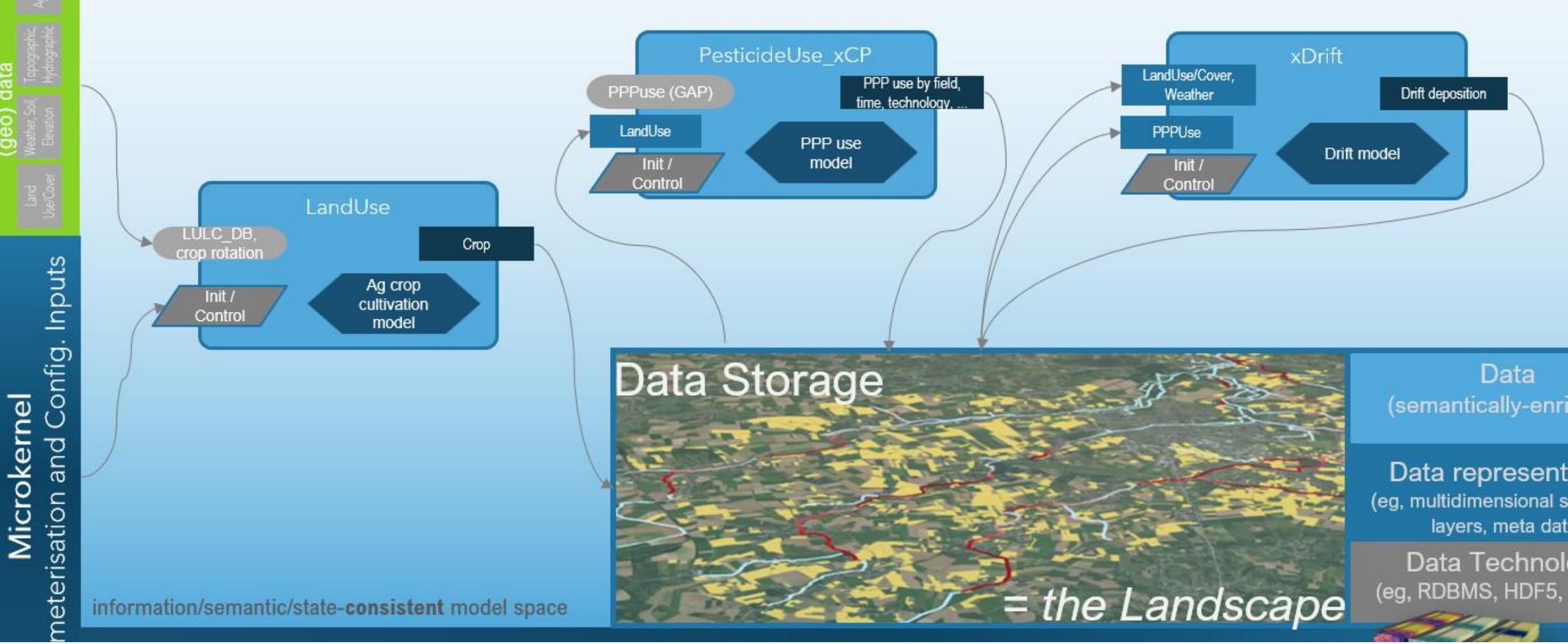
**Components**, **landscape models**, and **scenarios** can be build for a range of problems. **Open source.**

**Our primarily intention is not to establish fixed landscape models**
in a sense of building authoritative standards for years (eg as FOCUS models). Yet, even this is possible.

**We want to enable building flexible, adaptive research and 'higher-tier' landscape models serving individual problems, of highest possible harmonisation, continuity, and transparency.**

# Thanks

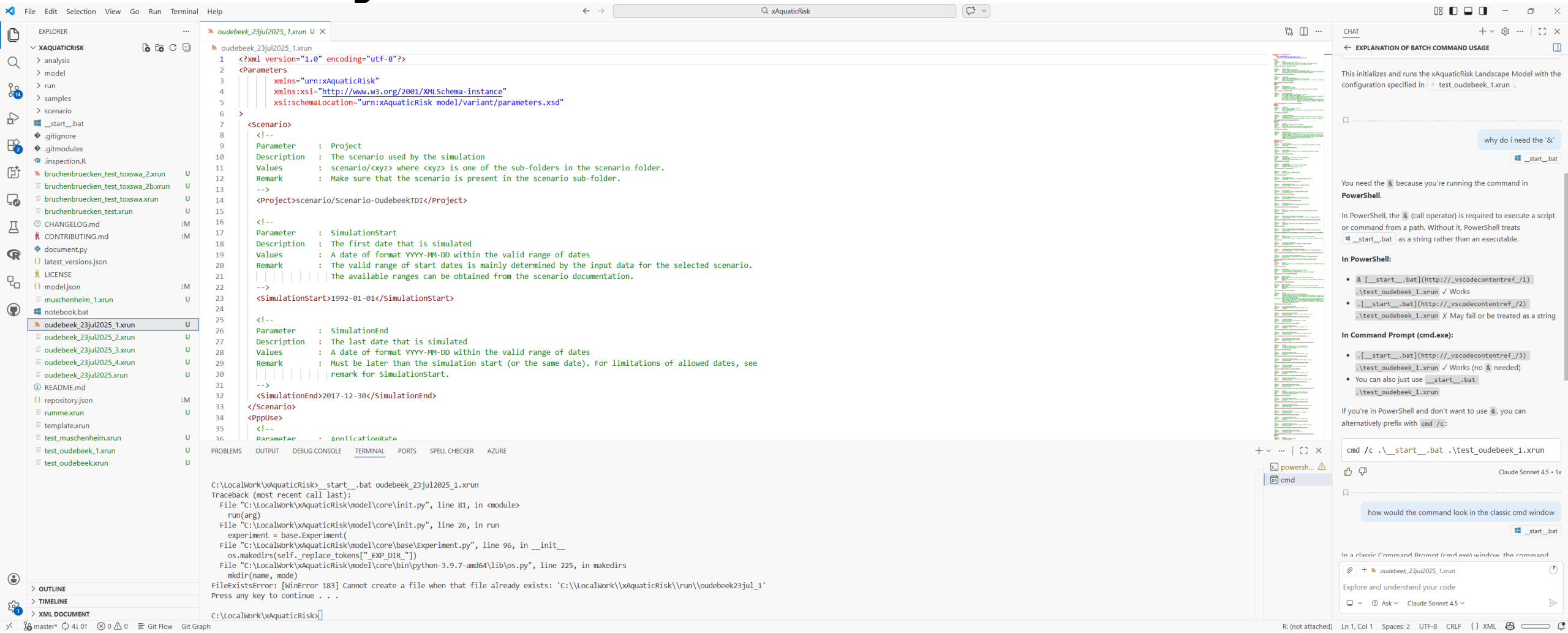
thorsten.schad@landwerk-ev.de



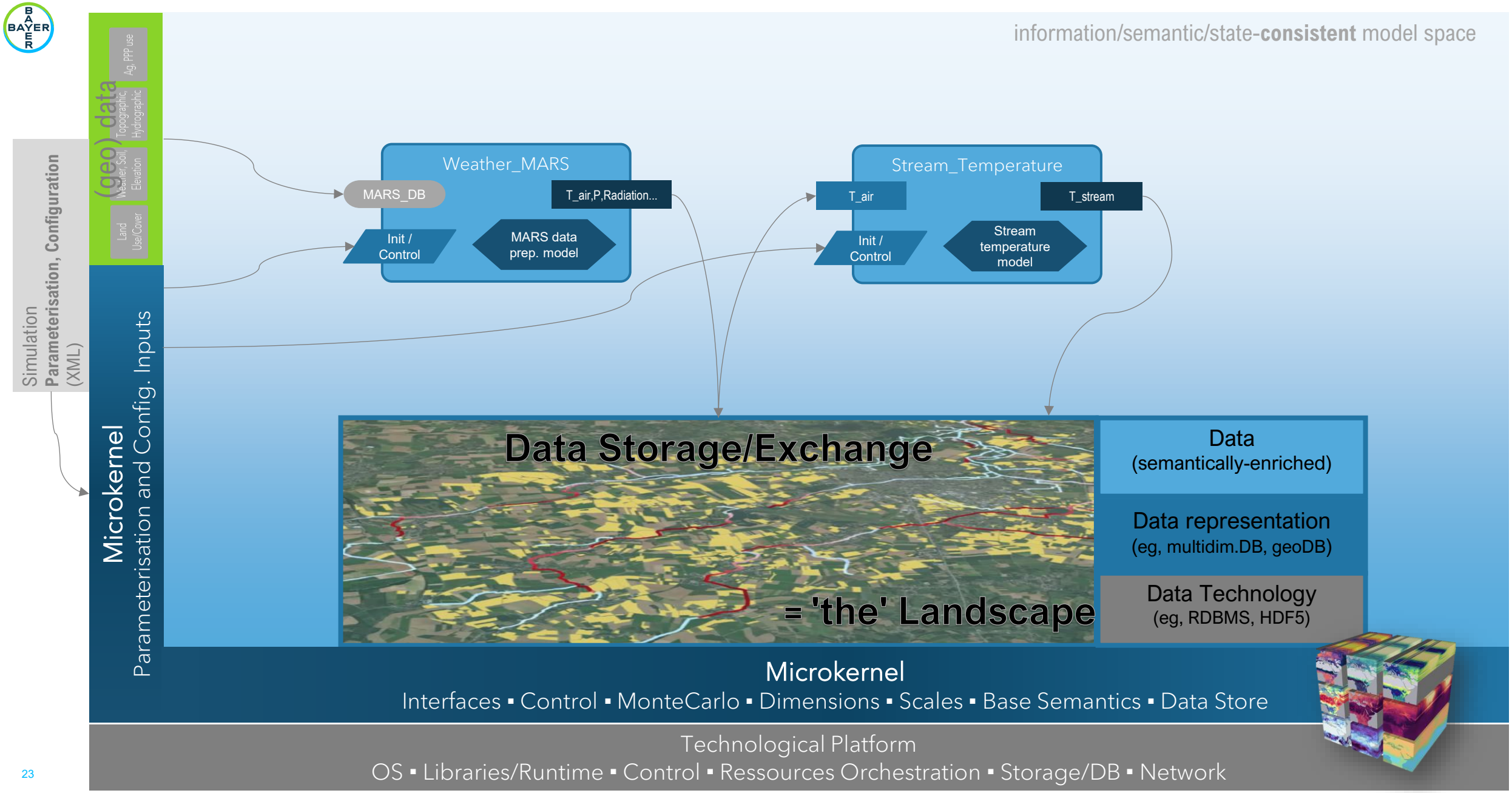

# xLandscape Technical Aspects

# Don't be afraid about coding anymore

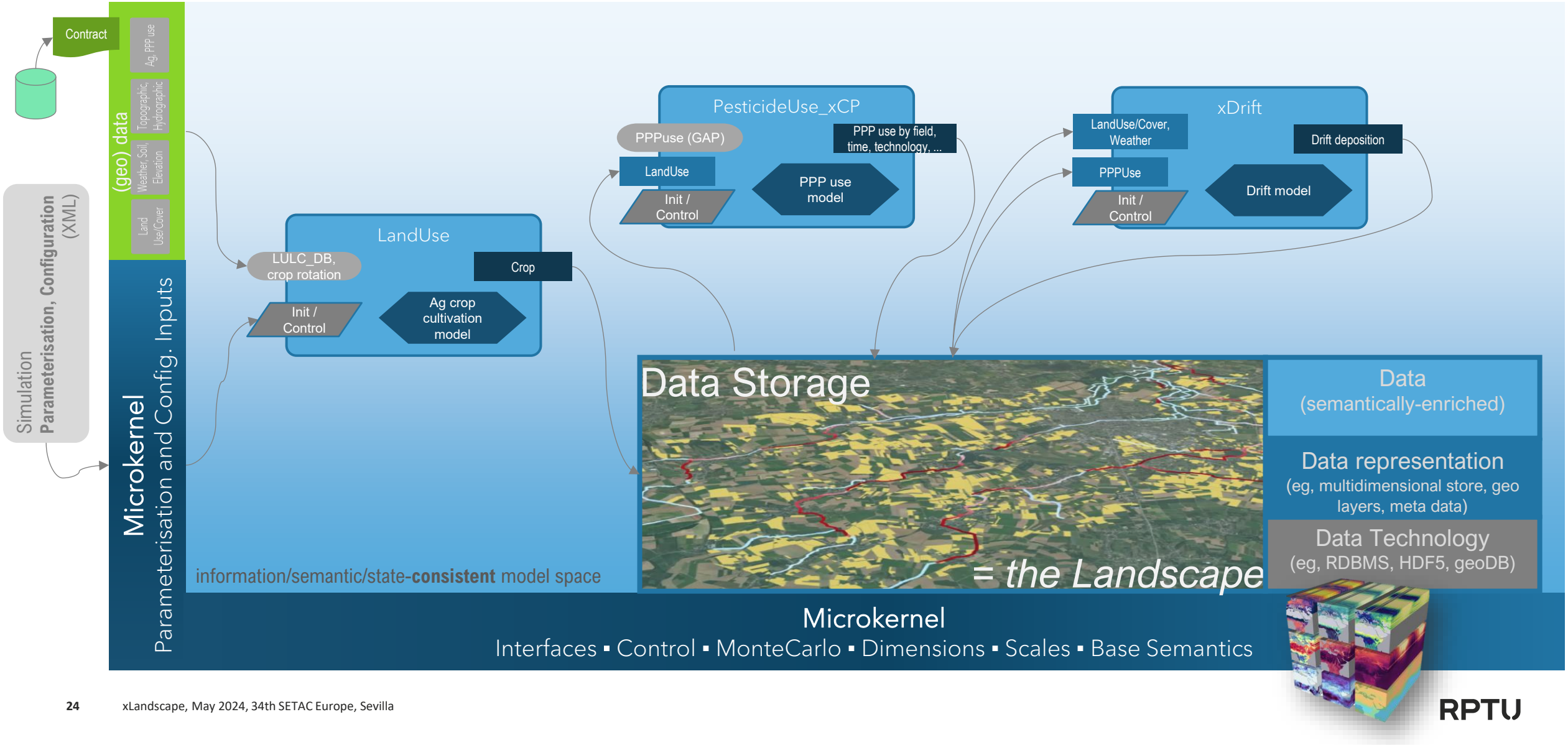
# AI is with you (and me)



# xLandscape Model Building – schematic illustration



# Microkernel and Components






RPTU

# Components

## Definition of inputs and outputs

A component dervies from the Component base class

- Components have a set of inputs
- Inputs specify the data that the component needs to run
- Inputs can (and should) be semantically described
- The Landscape Model framework and the model configuration make sure that the inputs are filled with according data (or deviations are handled)

```
class EnvironmentalFate(base.Component):
    def __init__(self, name: str, default_observer: base.Observer, default_store: typing.Optional[base.Store]) -> None:
        super(EnvironmentalFate, self).__init__(name, default_observer, default_store)
        self._inputs = base.InputContainer(self, inputs: (
            base.Input(
                name: "SprayDriftExposure",
                attributes: (
                    attrib.Class(np.ndarray),
                    attrib.Scales("space_y/1sqm, space_x/1sqm, time/day"),
                    attrib.Unit("g/ha")
                ),
                self.default_observer
            ),
            base.Input(
                name: "RunOffExposure",
                attributes: (
                    attrib.Class(np.ndarray),
                    attrib.Scales("space_y/1sqm, space_x/1sqm, time/day"),
                    attrib.Unit("g/ha")
                ),
                self.default_observer),
            base.Input(
                name: "SoilDT50",
                attributes: (attrib.Class(float), attrib.Scales("global"), attrib.Unit("d")),
                self.default_observer
            )
        ))
        self._outputs = base.OutputContainer(self, outputs: [base.Output( name: "Pec", default_store, self)])
```

The name of the component

Where to send messages to

Where to send data to

The name of the input

The component can specify the data type that it expects

Scales describe the shape of the data: here we expect one value per squaremeter in a regular raster per day

The physical unit of the values

Where to send messages to

Components also have outputs

The name of the output

Where to send data to

# Model composition

## An xLandscape model is build by an XML-based composition

A simulation makes use of
- Components that are run in sequence
- A store that manages the simulation data
- Observers that react on changes in the simulation

All these parts are configurable

The composition defines what components are used by the model and how data is exchanged

The configuration of a component lists the component‘s inputs and specifies its data

Data can be directly specified in the configuration (scalar and 1-d arrays): attached metadata helps the Landscape Model to check and fulfill requirements

```
<?xml version="1.0" encoding="utf-8"?>
<MCRun>
  <Observers...>
  <Store module="stores" class="X3dfStore"...>
  <Global>
    <SimulationStart>2006-01-01</SimulationStart>
    <SimulationEnd>2015-12-31</SimulationEnd>
  </Global>
  <Composition>
    <Weather module="components" class="MarsWeather"...>
    <LandscapeScenario module="components" class="LandscapeScenario"...>
    <PPM module="components" class="PpmCalendar"...>
    <SprayDrift module="XSprayDrift" class="SprayDrift" enabled="$(SimulateSprayDriftExposure)"...>
    <RunOffPrzm module="RunOffPrzm" class="RunOffPrzm" enabled="$(SimulateRunOffExposure)"...>
    <EnvironmentalFate module="components" class="EnvironmentalFate">
      <SprayDriftExposure>
        <FromOutput component="SprayDrift" output="Exposure" enabled="$(SimulateSprayDriftExposure)"/>
      </SprayDriftExposure>
      <RunOffExposure>
        <FromOutput component="RunOffPrzm" output="Exposure" enabled="$(SimulateRunOffExposure)"/>
      </RunOffExposure>
      <SoilDT50 type="float" unit="d" scales="global">$(DT50)</SoilDT50>
    </EnvironmentalFate>
    <DepositionToPecSoil module="components" class="DepositionToPecSoil"...>
    <DeleteFolder module="components" class="DeleteFolder"...>
  </Composition>
</MCRun>
```



RPTU

# Interfaces

## Read and write data

Every component has a run method that is called by the Landscape Model when it is the component's turn in the processing sequence

Inputs have a describe method that gives access to metadata: scales, unit, offsets, element names, geometries...

The read method of inputs loads data (and metadata) from the store into memory; without further specification, all values are read

After reading, the values property contains the actual values

Output values can be directly set or (for large datasets) be prepared

Here, a 3d-NumPy array is prepared (also see scales)

Outputs can (and should) be described in more detail regarding scales, unit, ...

Chunking can be tuned to ensure performant data access and compression

Offsets (among other) help to translate NumPy indices into real-world coordinates

Input data can be read partially using native indices (slices) or real-world coordinates (select)

Output can be written partially into prepared datasets (create=False) and creation of additional metadata can be controlled

```python
def run(self) -> None:
    """..."""
    enabled_exposure_inputs = []
    if self.inputs["SprayDriftExposure"].provider is not None:
        enabled_exposure_inputs.append(self.inputs["SprayDriftExposure"])
    if self.inputs["RunOffExposure"].provider is not None:
        enabled_exposure_inputs.append(self.inputs["RunOffExposure"])
    data_set_info = enabled_exposure_inputs[0].describe()
    soil_dt50 = self.inputs["SoilDT50"].read().values
    self.outputs["Pec"].set_values(
        np.ndarray,
        shape=data_set_info["shape"],
        data_type=data_set_info["data_type"],
        chunks=base.chunk_size( chunk: (None, None, 1), data_set_info["shape"]),
        scales="space_y/1sqm, space_x/1sqm, time/day",
        unit="g/ha",
        offset=data_set_info["offsets"]
    )
    pec_current_day = np.zeros(
        shape: (data_set_info["shape"][0], data_set_info["shape"][1], 1), data_set_info["data_type"])
    for t in range(data_set_info["shape"][2]):
        current_slice = (slice(0, data_set_info["shape"][0]), slice(0, data_set_info["shape"][1]), slice(t, t + 1))
        pec_current_day *= math.exp(-math.log(2) / soil_dt50)
        for exposureInput in enabled_exposure_inputs:
            pec_current_day += exposureInput.read(slices=current_slice).values
        self.outputs["Pec"].set_values(pec_current_day, create=False, slices=current_slice, calculate_max=True)
```



RPTU

# Best Practice

**Align with regulatory requirements:** Ensure that the modelling approach is consistent with the relevant guidance documents, protection goals, and assessment endpoints for the species group and environmental compartment of interest.

**Document the modelling cycle:** Provide transparent documentation of the problem definition, model formulation, implementation, parameterization, and validation steps, including sensitivity and uncertainty analyses.

**Justify scenario selection:** Clearly explain the selection of landscape scenarios, their representativeness, and their applicability to the intended use and region.

**Use validated models and data:** Employ models that have been evaluated for regulatory use, and use high-quality, representative input data. Where possible, validate model outputs against empirical observations or monitoring data.

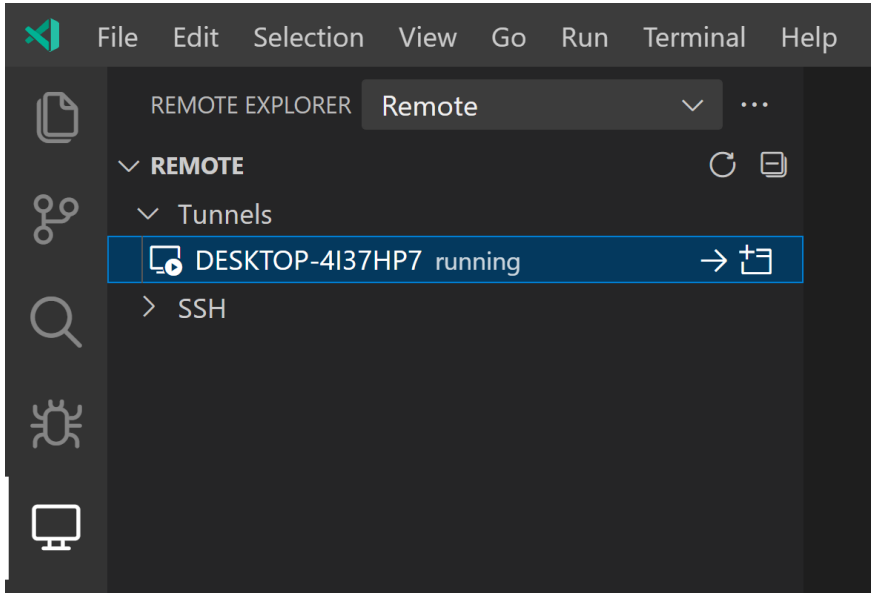
**Integrate exposure and effect assessments:** Where relevant, link exposure modelling with effect models (e.g., population, food web, agent-based) to provide a holistic assessment of risk and recovery.

**Address mitigation and uncertainty:** Evaluate the effectiveness of risk mitigation measures in the landscape context, and provide a systematic assessment of uncertainties and their implications for risk characterization.

**Engage with EFSA and stakeholders:** Participate in pre-submission meetings, public consultations, and stakeholder workshops to ensure alignment with evolving guidance and expectations.

# Run your Model on a Backend

**VS Code Tunnel**



# Microkernel Architecture Pattern

**Pattern Description**

*"The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts. The microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration."*

1. Microkernel architecture pattern
    1. it's not just about linking components (models), **it's about landscape modelling**; this brings the linking of components into a **domain**
        1. modelling happen in a 'geographic world' (coordinate system)
        2. space and time →spatiotemporally explicit
        3. scales → scale-explicit
2. Minimum Kernel Functionality
    1. Component control
    2. Client adapter
    3. storage
    4. isolation, processes, scheduling
    5. messages

**Pattern Description**