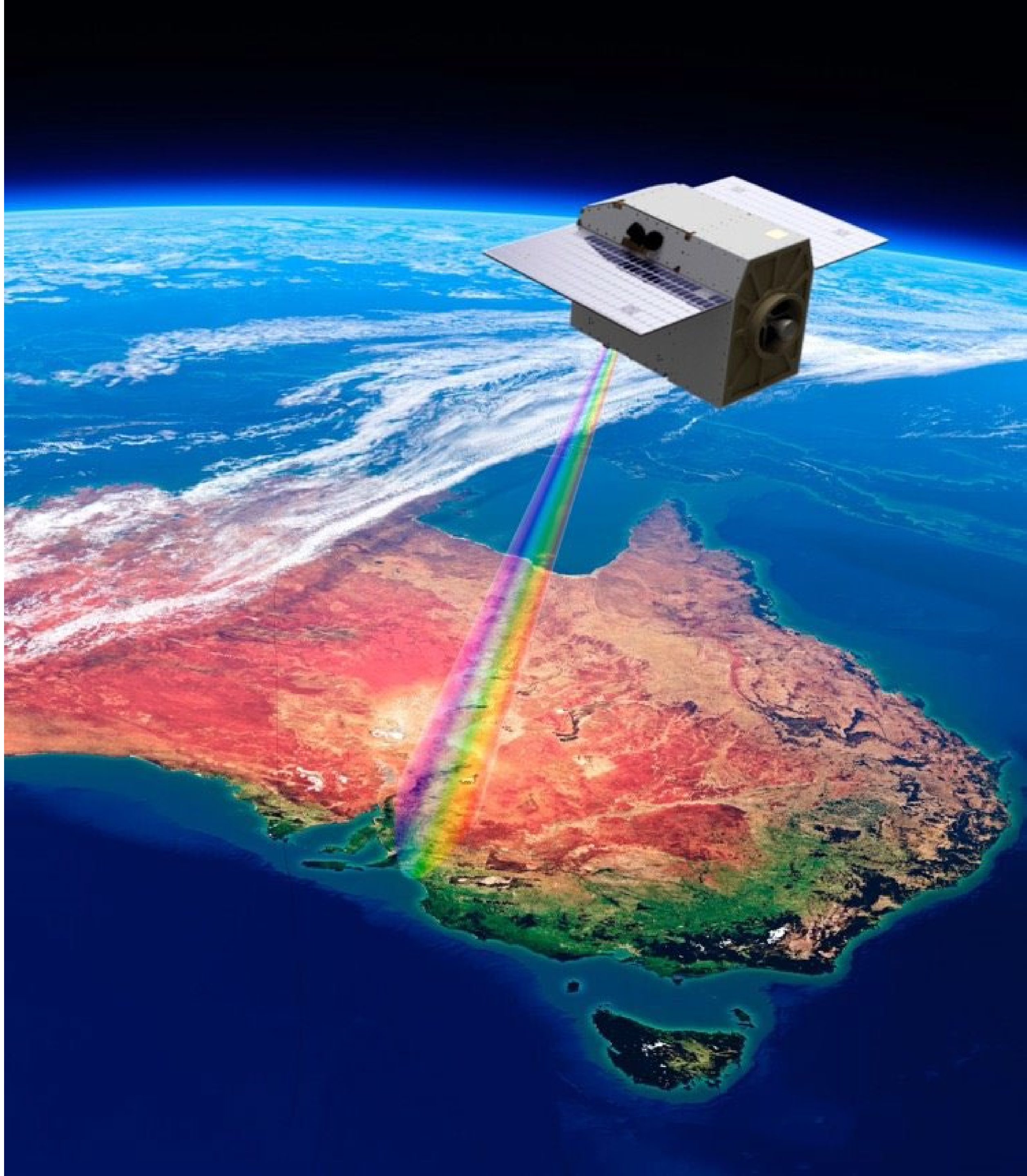
CSIRO

Modernizing Legacy: Wrapping a 25+ Year Computational Fluid Dynamics Codebase

Duy Nguyen, Tisham Dhar, Peter Wang, Jean-Michel Perraud, Klaus Joehnk

November 2025

Australia’s National Science Agency



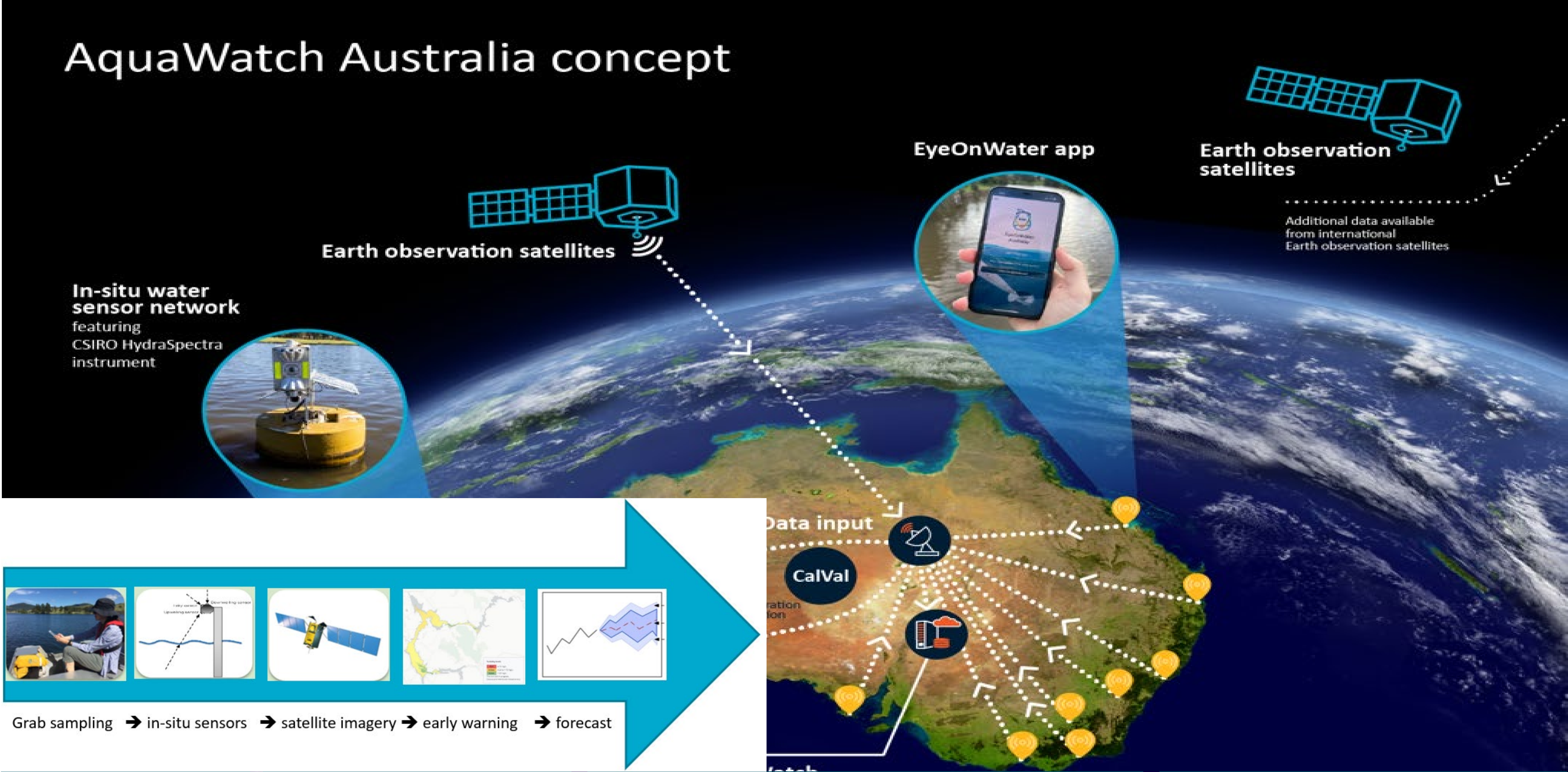
Water Quality:
a global challenge

- Over three billion people are at risk of illness from poor water quality due partly to a lack of monitoring (UN, 2023).
- Aquatic ecosystems rapidly degrading: 35% of wetlands and 15% of coral has been lost since 1970 (Convention on Wetlands, 2021; Souter et al., 2021).
- Comprehensive monitoring of inland and coastal waters needed for effective management and conservation.

6 CLEAN WATER AND SANITATION

14 LIFE BELOW WATER

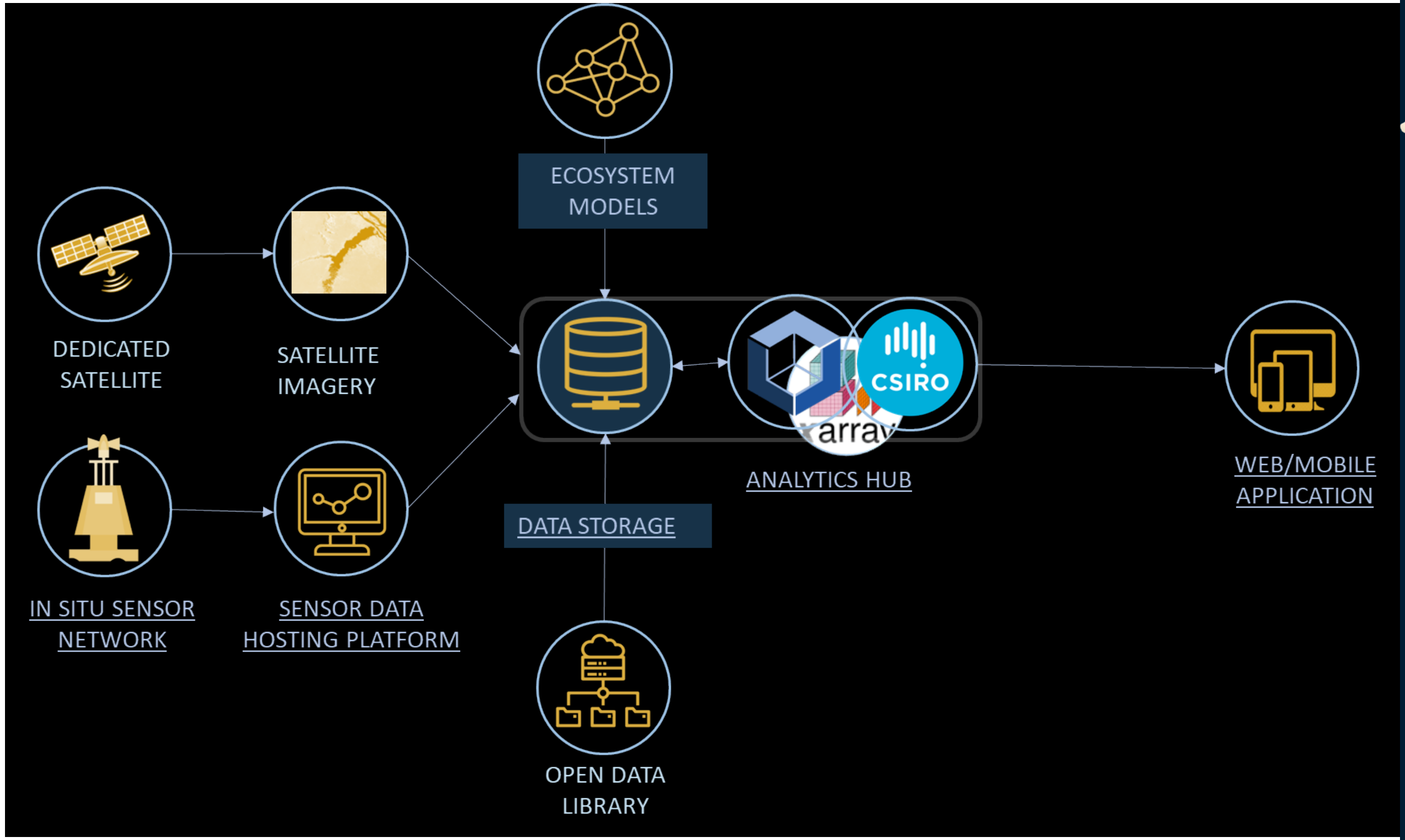
17 PARTNERSHIPS FOR THE GOALS

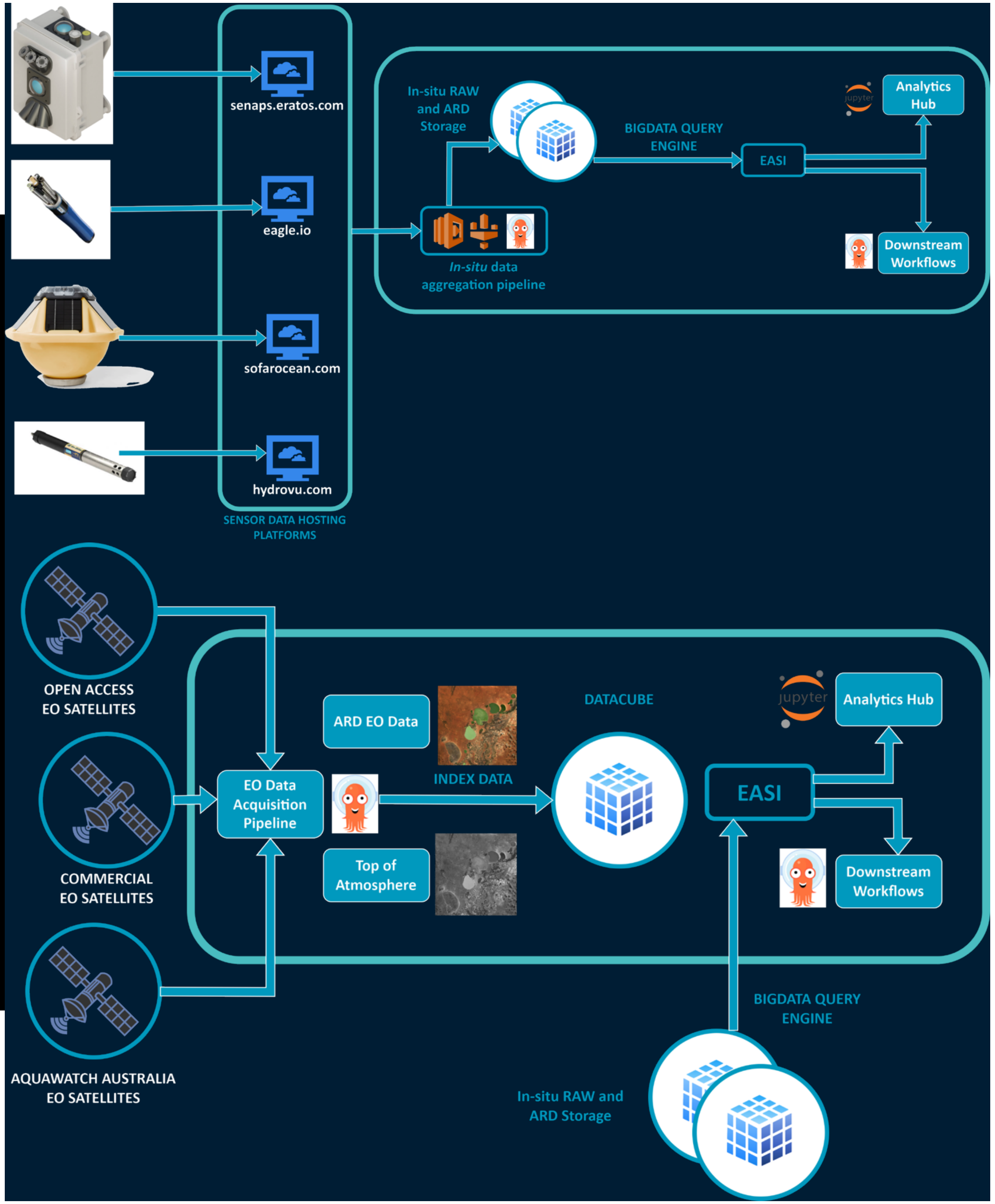


In-situ sensor network	Satellite missions for water quality	ADIAS – AquaWatch Data Integration and Analytics System	Water quality modelling and forecast

AquaWatch Data System

A cloud-based solution for water quality monitoring and forecasting

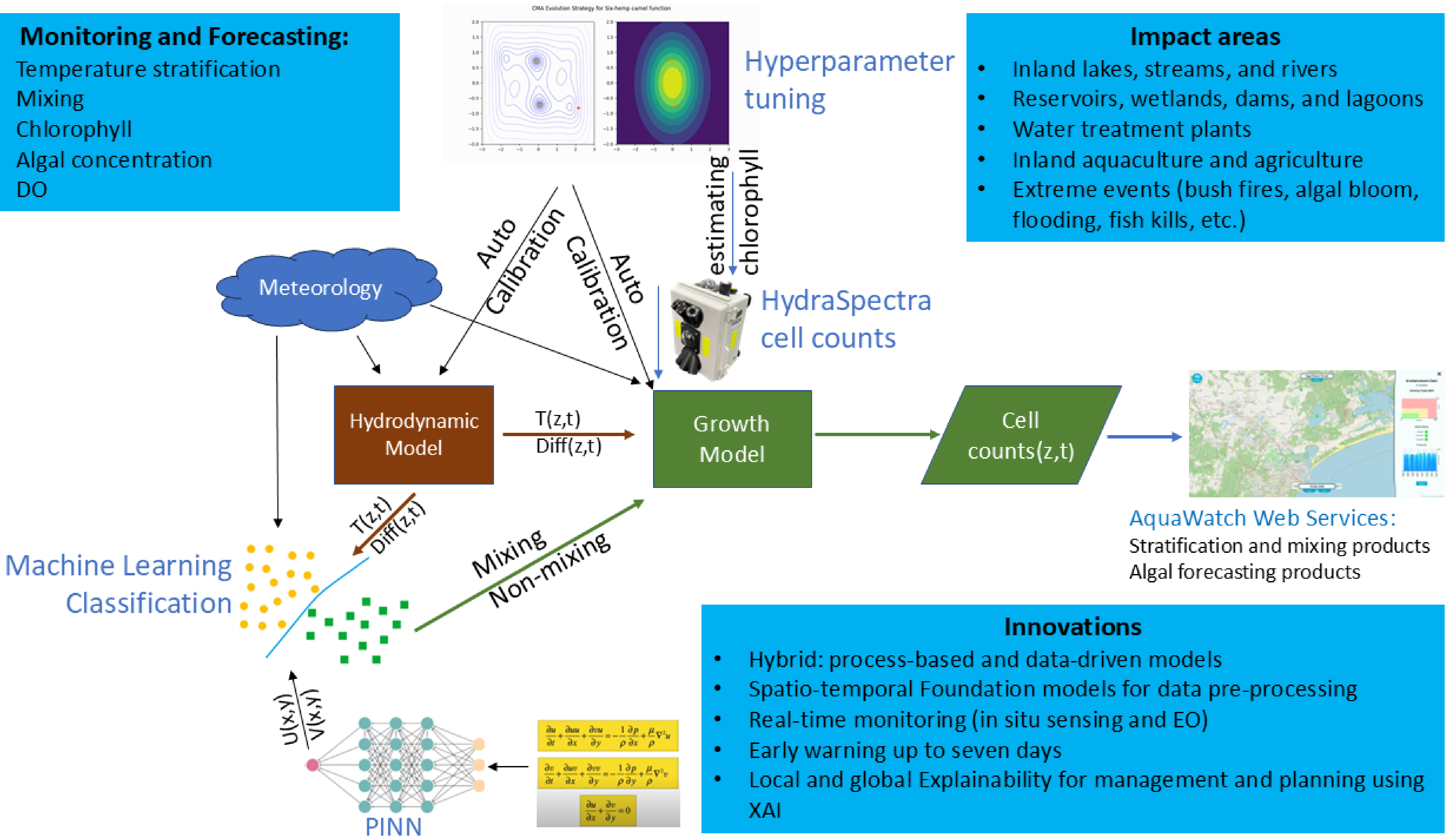




Inland Water Quality Modeling and Forecasting

Monitoring and Forecasting:
- Temperature stratification
- Mixing
- Chlorophyll
- Algal concentration
- DO

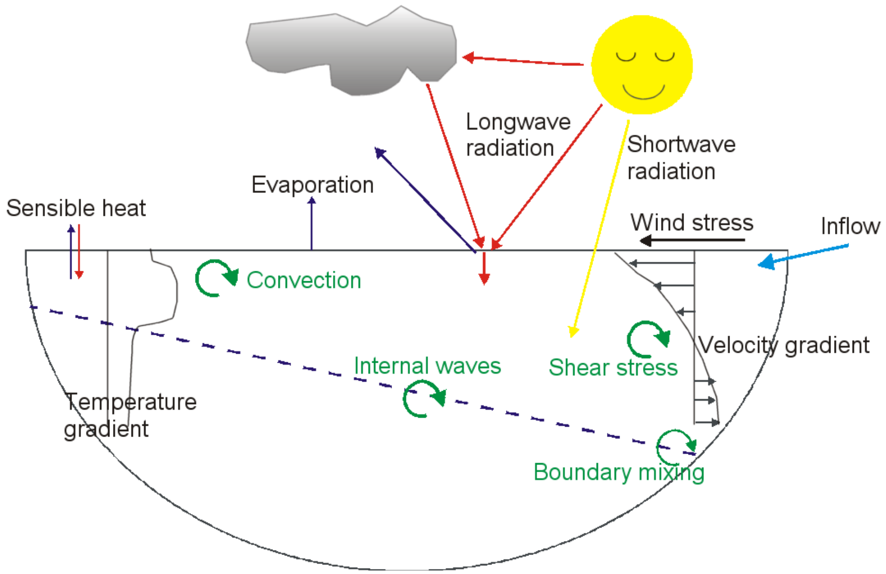
Impact areas
- Inland lakes, streams, and rivers
- Reservoirs, wetlands, dams, and lagoons
- Water treatment plants
- Inland aquaculture and agriculture
- Extreme events (bush fires, algal bloom, flooding, fish kills, etc.)



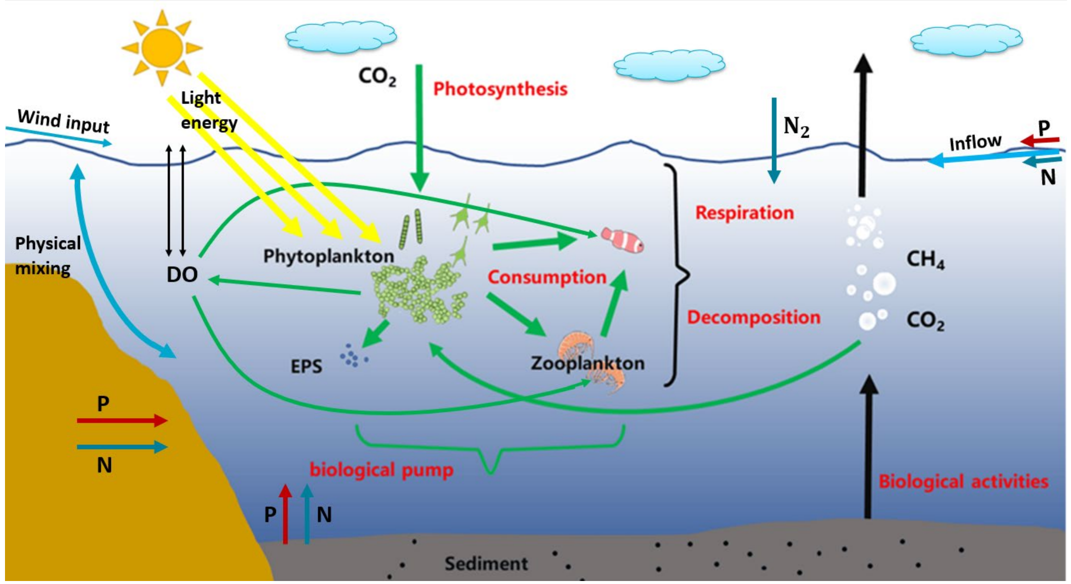
Innovations
- Hybrid: process-based and data-driven models
- Spatio-temporal Foundation models for data pre-processing
- Real-time monitoring (in situ sensing and EO)
- Early warning up to seven days
- Local and global Explainability for management and planning using XAI

CSIRO

The CFD Governing Equations



Physical processes in a lake



Biological competition processes in a lake

Momentum

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial z}\left((\nu + \nu_t)\frac{\partial u}{\partial z}\right) - c_D u^2 \frac{1}{A}\frac{\partial A}{\partial z}$$

Heat

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial z}\left(\left(\chi + \frac{\nu_t}{\sigma_T}\right)\frac{\partial T}{\partial z}\right) + \frac{1}{\rho_0 c_p}\frac{\partial I}{\partial z}$$

k−ε−turbulence model

$$\frac{\partial k}{\partial t} = \frac{\partial}{\partial z}\left(\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial z}\right) + P + G - \varepsilon$$

$$\frac{\partial \varepsilon}{\partial t} = \frac{\partial}{\partial z}\left(\left(\nu + \frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial z}\right) + (c_1 P + c_3 G - c_2 \varepsilon)\frac{\varepsilon}{k}$$

$$\nu_t = c_\mu \frac{k^2}{\varepsilon} \qquad G = \frac{\nu_t}{\sigma_T}N^2 \qquad N^2 = -\frac{g}{\rho}\frac{\partial \rho}{\partial z}$$

Population dynamics

$$\frac{\partial N_i}{\partial t} = \left(\mu_i(I,T) - m_i(T)\right)N_i + U_i\frac{\partial N_i}{\partial z} + \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_{N_i}}\frac{\partial N_i}{\partial z}\right)$$

Specific growth rate

$$\mu_i(I,T) = \mu_{\max,i}(T)\frac{I}{H_i + I} \quad , \quad H_i = \mu_{\max,i}/\alpha$$

Light field

$$I(z) = I_{in}\exp\left(-\int_0^z\left[\sum_{i=1}^{n}\kappa_i N_i(\sigma,t)\right]d\sigma - K_{bg}z\right)$$

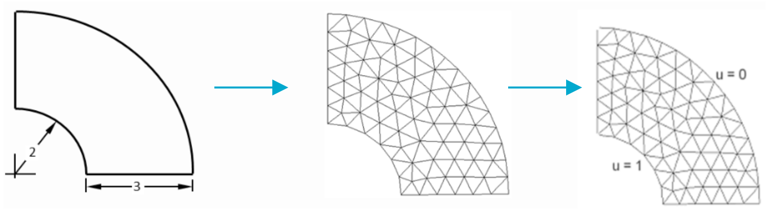
Growth/loss function

$$\mu_{\max,i}(T) = b_{i1}\left(R_i^{T-20} - R_i^{b_{i2}(T-b_{i3})} + b_{i4}\right)$$

$$m_i(T) = m_i(20)Q_i^{T-20}$$

CSIRO

The CFD Workflow



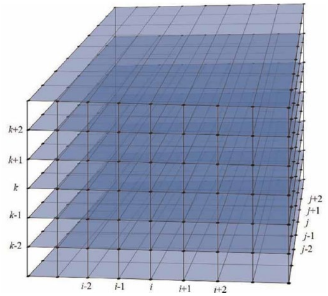
Continuity: $$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}+\frac{\partial(\rho w)}{\partial z}=0$$

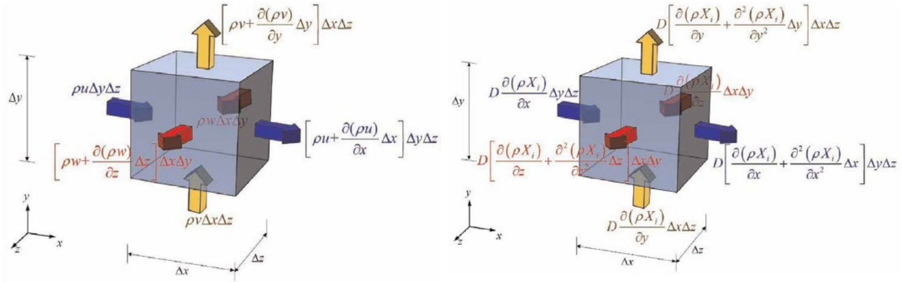
X – Momentum: $$\frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho u^2)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}+\frac{\partial(\rho uw)}{\partial z}=-\frac{\partial p}{\partial x}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xx}}{\partial x}+\frac{\partial \tau_{xy}}{\partial y}+\frac{\partial \tau_{xz}}{\partial z}\right]$$

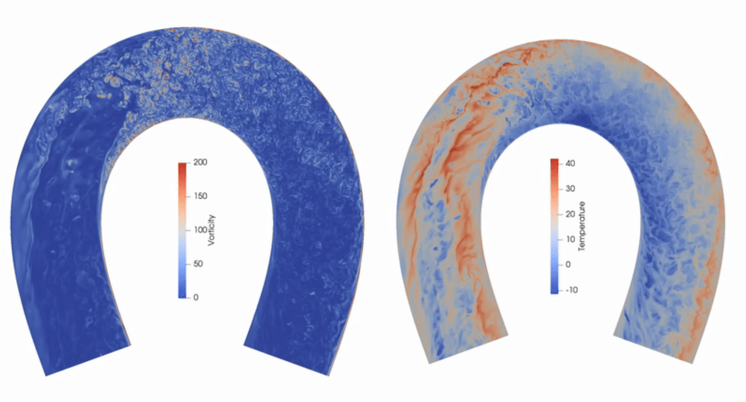
Y – Momentum: $$\frac{\partial(\rho v)}{\partial t}+\frac{\partial(\rho uv)}{\partial x}+\frac{\partial(\rho v^2)}{\partial y}+\frac{\partial(\rho vw)}{\partial z}=-\frac{\partial p}{\partial y}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xy}}{\partial x}+\frac{\partial \tau_{yy}}{\partial y}+\frac{\partial \tau_{yz}}{\partial z}\right]$$

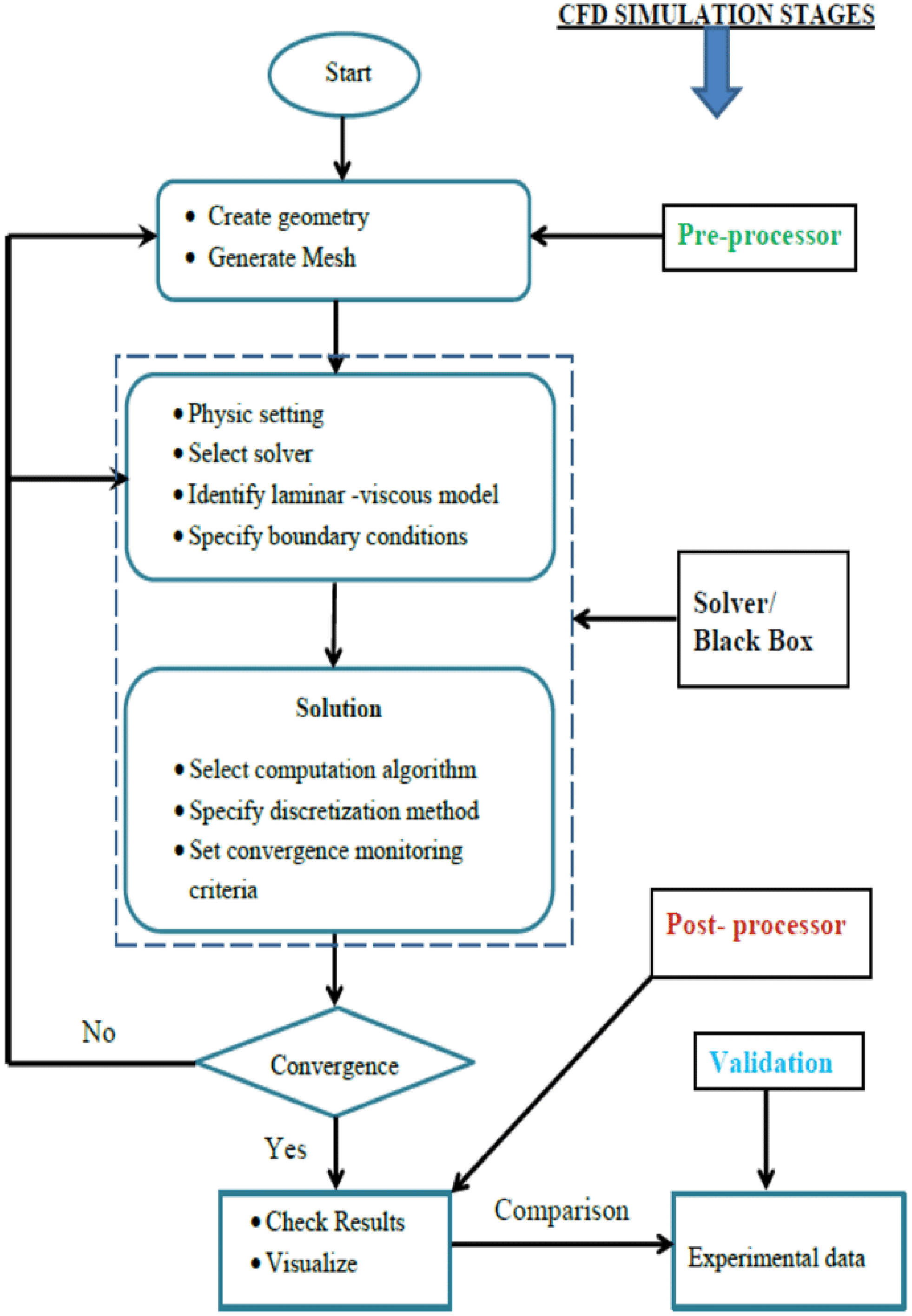
Z – Momentum $$\frac{\partial(\rho w)}{\partial t}+\frac{\partial(\rho uw)}{\partial x}+\frac{\partial(\rho vw)}{\partial y}+\frac{\partial(\rho w^2)}{\partial z}=-\frac{\partial p}{\partial z}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xz}}{\partial x}+\frac{\partial \tau_{yz}}{\partial y}+\frac{\partial \tau_{zz}}{\partial z}\right]$$

Energy:
$$\frac{\partial(E_T)}{\partial t}+\frac{\partial(uE_T)}{\partial x}+\frac{\partial(vE_T)}{\partial y}+\frac{\partial(wE_T)}{\partial z}=-\frac{\partial(up)}{\partial x}-\frac{\partial(vp)}{\partial y}-\frac{\partial(wp)}{\partial z}-\frac{1}{Re_r Pr_r}\left[\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right]$$
$$+\frac{1}{Re_r}\left[\frac{\partial}{\partial x}(u\tau_{xx}+v\tau_{xy}+w\tau_{xz})+\frac{\partial}{\partial y}(u\tau_{xy}+v\tau_{yy}+w\tau_{yz})+\frac{\partial}{\partial z}(u\tau_{xz}+v\tau_{yz}+w\tau_{zz})\right]$$









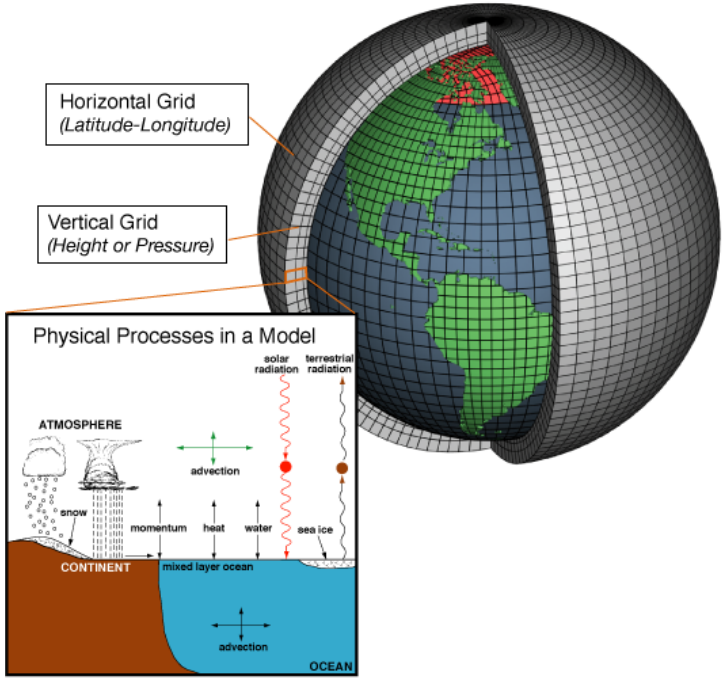
OFFICIAL

CSIRO

The CFD Solver – Why Fortran and C++?

```
+==================================================================+
|                  FORTRAN / C++ FOR HPC & PARALLEL                |
+==================================================================+

1. COMPILED LANGUAGES
   ---------------------------------------------------------------
   - Translated directly into optimized machine code
   - Minimal runtime overhead → faster execution

2. MEMORY CONTROL
   ---------------------------------------------------------------
   - Precise control over memory allocation and layout
   - Contiguous arrays improve cache efficiency
   - Reduces memory fragmentation for large-scale simulations

3. ARRAYS & DATA STRUCTURES
   ---------------------------------------------------------------
   - Fortran: column-major arrays → ideal for PDE solvers
   - C++: row-major arrays, vectors, custom layouts
   - Enables predictable memory access → higher performance

4. LOOPS & VECTORIZATION
   ---------------------------------------------------------------
   - Compilers can unroll loops automatically
   - SIMD/vector instructions utilized
   - Critical for inner numerical kernels in CFD & climate models

5. PARALLELISM & HPC SUPPORT
   ---------------------------------------------------------------
   - Native support for MPI, OpenMP, CUDA, OpenACC
   - Scales to thousands of cores efficiently
   - Essential for climate, aerospace, and CFD simulations

6. PROVEN PERFORMANCE
   ---------------------------------------------------------------
   - Decades of optimization in scientific computing
   - Mature, validated numerical libraries
   - Reliable for large-scale HPC applications

+==================================================================+
```

```
                   ┌──────────────────────────┐
                   │  Computational Fluid     │
                   │     Dynamics (CFD)       │
                   └────────────┬─────────────┘
                                │
         ┌──────────────────────┴──────────────────────┐
         │                                             │
 ┌───────────────┐                            ┌──────────────────┐
 │   Fortran     │                            │      C++         │
 │ (Since 1950s) │                            │  (Since 1980s)   │
 └───────┬───────┘                            └────────┬─────────┘
         │                                             │
 - Optimized for numerical arrays             - Object-oriented design
 - Simple, fast compilers                     - Complex data structures
 - Massive legacy CFD codes (50+ years)       - Better memory management
 - Proven stability on supercomputers         - Modern software practices
 - Trusted in aerospace, climate, energy      - Widely adopted in HPC projects
         │                                             │
         └────────────────┐              ┌─────────────┘
                          │              │
                 ┌──────────────────┐────────────┐
                 │ Why Still Used?  │            │
                 └──────────────────┘            │
            • Extreme performance for HPC simulations │
            • Robust parallelization (MPI, OpenMP)    │
            • Large validated codebases               │
            • Community trust in accuracy & reliability│
                                                      │
                        ┌──────────────────────────────────┐
                        │              Python              │
                        │    (Modern Wrapper & Glue)       │
                        └──────────────────────────────────┘
                        • Easy prototyping, scripting
                        • Strong ecosystem (NumPy, SciPy)
                        • Connects Fortran/C++ via wrappers
                        • Great for pre/post-processing, AI/ML
                        • BUT: too slow for HPC kernels
```



CSIRO

The CFD Solver with Python?

Fortran/C++ vs Python in CFD

Fortran / C++

- High-performance solvers
- Legacy, validated codebases
- Parallel computing (MPI/OpenMP)
- Numerical libraries (BLAS, LAPACK)
- Critical applications: aerospace, climate, engineering

Python

- Pre-processing & meshing
- Wrappers for Fortran/C++ solvers
- Post-processing & visualization
- Machine learning integration
- Rapid prototyping of ideas

```
Python (High-level)
   ↓
Wrappers / Glue
   ↓
Fortran / C++ Solver (HPC)
   ↑
Results back to Python for visualization
```

Feature	Fortran	C++	Python (alone)
Speed / Performance	Excellent	Excellent	Poor (unless wrapped)
Memory control	Very good	Very good	Limited
Parallel / HPC	Native	Native	Via wrappers
Legacy libraries	Many decades	Growing	Limited
Ease of prototyping	Low	Medium	Excellent
Numerical stability	Excellent	Excellent	Limited

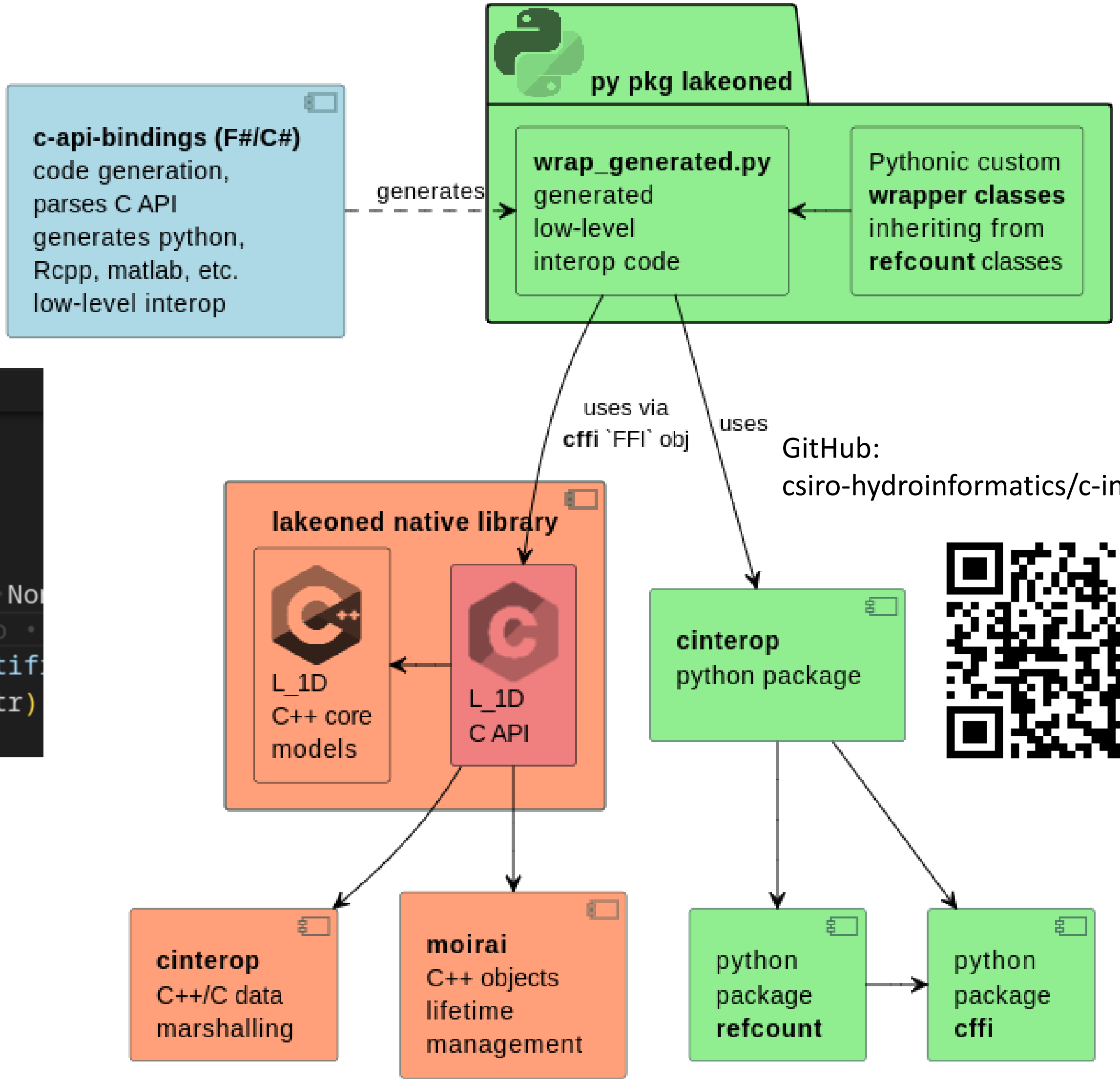
CSIRO

LAKEoneD Python wrapper using *cffi*

```
> classes.py > LodSimulation > record_vector
class LodSimulation(ModelStates):
    def record_vector(self, name:str) -> None:
        """Records a variable with vertical profile (vector)
        lwg.RecordVectorLms_py(self, name)

    def record_scalar(self, name) -> None:
```

```
> wrap > lakeoned_wrap_generated.py > RecordVectorLms_py

@_lod_wrap.check_exceptions
def _RecordVectorLms_native(simulation, variableIdentifier):
    lakeoned_so.RecordVectorLms(simulation, variableIdentifier)

def RecordVectorLms_py(simulation:'LodSimulation', variableIdentifier:str) -> No
    simulation_xptr = wrap_as_pointer_handle(simulation)        You, 4 years ago •
    variableIdentifier_c_charp = wrap_as_pointer_handle(as_bytes(variableIdentif
    _RecordVectorLms_native(simulation_xptr.ptr, variableIdentifier_c_charp.ptr)
    # no cleanup for const char*
```

```
LAKEoneD > lake_lib > include > lakeoned > C extern_c_api.h > ...
  extern "C" {
    LAKEONED_API void RecordVectorLms(
        LAKE_MODEL_SIMULATION_PTR simulation,
        const char* variableIdentifier);

    LAKEONED_API lod_matrix_2d* GetRecordedVectorLms(LAKE_MO
        You, 5 years ago • Enable recording vector states i.e.
```



GitHub:
csiro-hydroinformatics/c-interop

CSIRO

Hydrodynamics Python wrapper from notebooks

lake_hume.ipynb

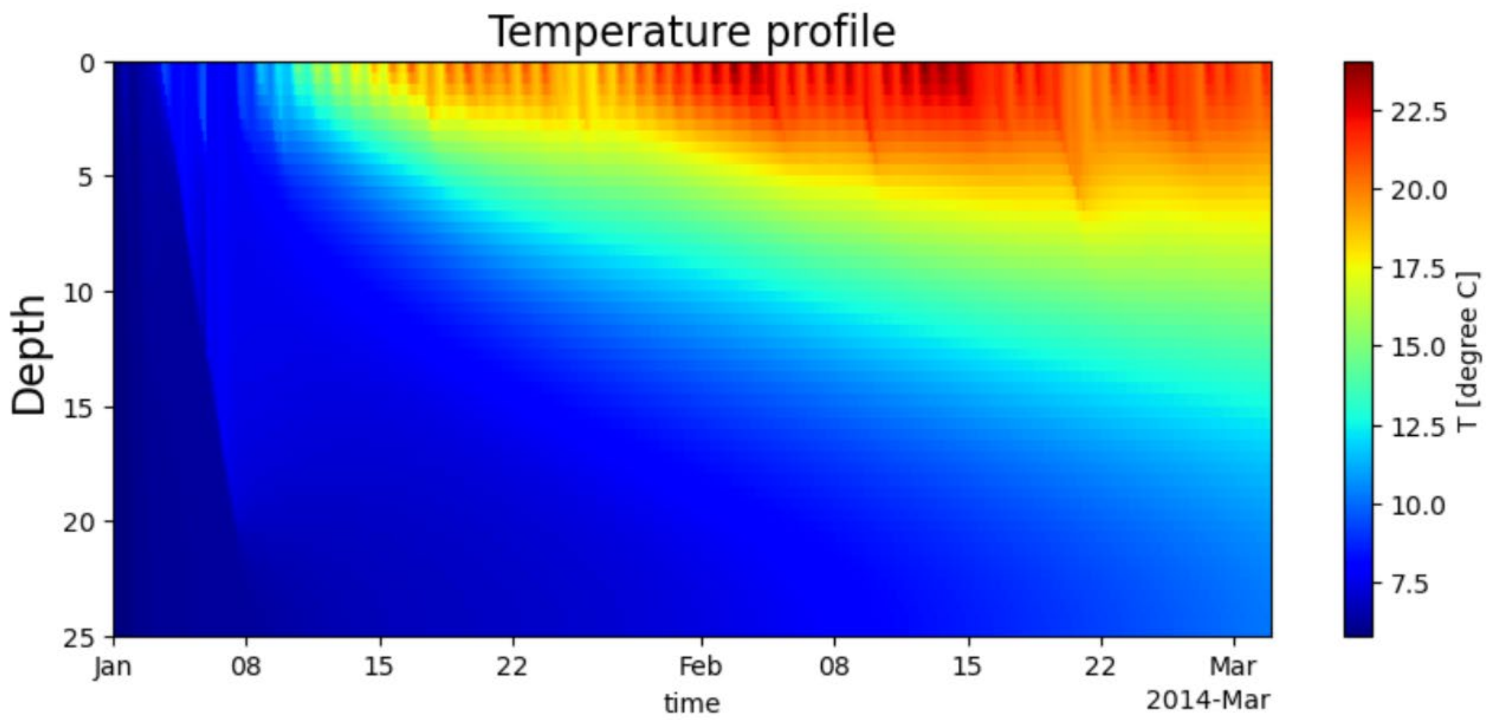
Code

object `sim` is a `lakeoned.classes.LodSimulation`, wraps native pointer of type id "LAKE_MODEL_SIMULATION_PTR"

[38]:
```
sim.execute_simulation()
```

```
T:xr.DataArray = sim.recorded_vector('T') •••
```

```
a = plot_vec_ts(T, x=TIME_DIMNAME, y=DEPTH_DIMNAME, •••
```



CSIRO

Hypertuning

- Manual method:
  - Trial and error by guessing the parameter values (floats)
  - Run lake1D, which takes a few minutes.
  - Compare the results with the ground truth.
  - Rinse, repeat maybe 100s of times, and still not be near the global optima.
- Better approach:
  - We want to do the "trial and error" search in parallel and at scale.
  - We need to automate this, so it finds the best parameter set for us auto-magically.
  - We want the "trial and error" search to be smart and directed towards the global optima.
  - We want to do it with as little code as possible and as efficiently as possible.
  - Optuna!

Hyperparameters

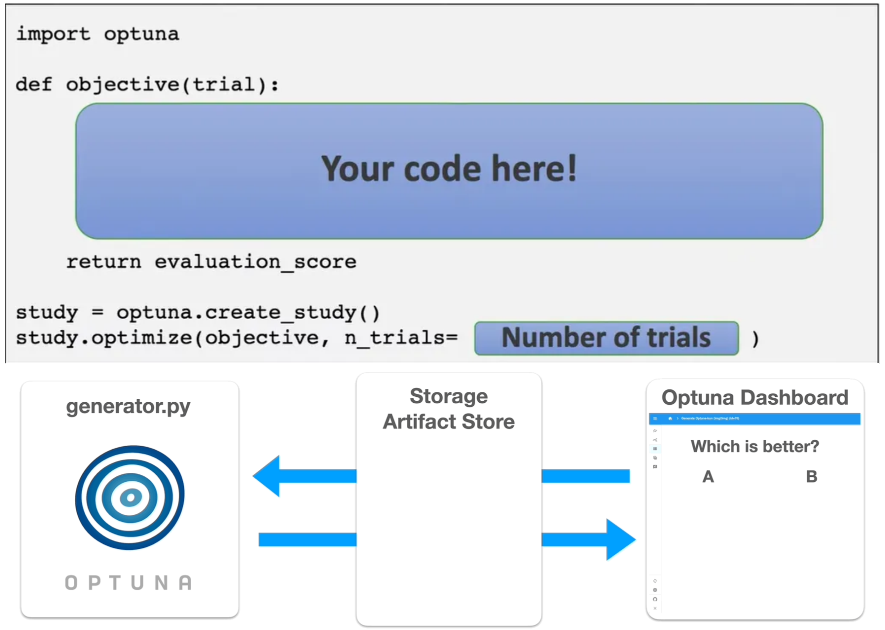
- Light:
  - *clear_water_att*
- Bottom boundary:
  - *bottom_stress_coeff*
- Wind:
  - *wind_factor*
- Turbulence:
  - *min_tdiff*
- Meteorological scaling factors:
  - *irr_scale*
  - *vel_scale*
  - *hum_scale*
  - *temp_scale*

Friend: how long have you been working on that **tuning**?
Me: since 5pm
Friend: but it's 4pm
Me:

Credit: Viky Akbartama

Hypertuning – Optuna

- What is Optuna:
  - Open-source framework for automated hyperparameter optimisation.
  - Built for machine learning, deep learning, and general optimisation.
  - Designed to find the best model configuration with minimal manual tuning.
- Why we chose Optuna:
  - Optuna provides the best balance of power, flexibility, and usability, making it ideal for most modern machine learning workflows with great results with minimal code/effort.



Credit: Optuna core-dev. GitHub: c-bata

- Comparisons with other frameworks:

Feature / Framework	Optuna	Hyperopt	Ray Tune	Scikit-Optimize	Bayesian Optimization (bayes_opt)
Language	Python	Python	Python	Python	Python
Search Algorithms	Many (incl, TPE, CMA-ES, Grid, Random)	TPE, Random	Many (incl. TPE, PBT, BOHB)	GP, RF, Random	Bayesian (GP), Random
Define-by-Run	Yes	No	Yes	No	No
Pruning Support	Yes	Limited	Yes	No	No
Parallelisation	Yes	Limited	Yes	No	Limited
Ease of Use	Very Easy	Easy	Complex	Simple	Easy

CSIRO

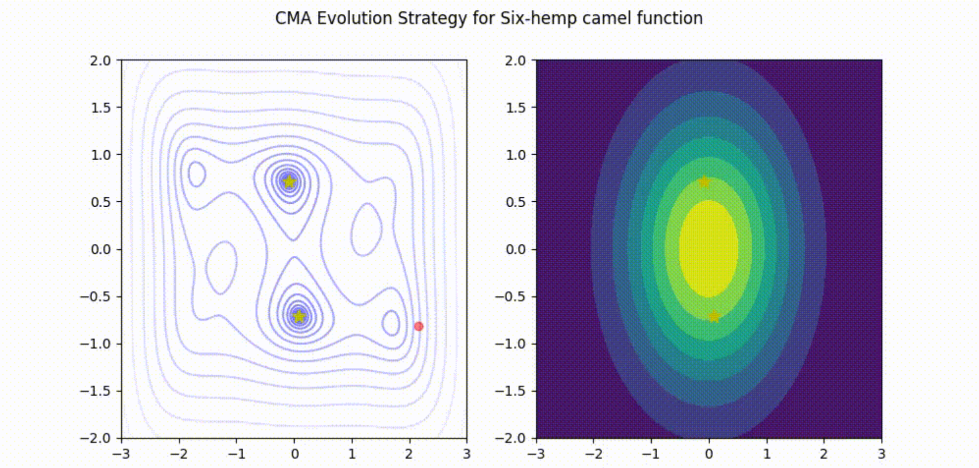
Hypertuning – Optuna - CmaEsSampler

	RandomSampler	GridSampler	TPESampler	CmaEsSampler	NSGAIISampler	QMCSampler	GPSampler	BoTorchSampler	BruteForceSampler
Float parameters	✓	✓	✓	✓	▲	✓	✓	✓	✓ (✗ for infinite domain)
Integer parameters	✓	✓	✓	✓	▲	✓	✓	▲	✓
Categorical parameters	✓	✓	✓	▲	✓	▲	✓	▲	✓
Pruning	✓	✓	✓	▲	✗ (▲ for single-objective)	✓	▲	▲	✓
Multivariate optimization	▲	▲	✓	✓	▲	▲	✓	✓	▲
Conditional search space	✓	▲	✓	▲	▲	▲	▲	▲	✓
Multi-objective optimization	✓	✓	✓	✗	✓ (▲ for single-objective)	✓	✗	✓	✓
Batch optimization	✓	✓	✓	✓	✓	✓	▲	✓	✓
Distributed optimization	✓	✓	✓	✓	✓	✓	▲	✓	✓
Constrained optimization	✗	✗	✓	✗	✓	✗	✗	✓	✗
Time complexity (per trial) (*)	$O(d)$	$O(dn)$	$O(dn \log n)$	$O(d^3)$	$O(mp^2)$ (***)	$O(dn)$	$O(n^3)$	$O(n^3)$	$O(d)$
Recommended budgets (#trials) (**)	as many as one likes	number of combinations	100 – 1000	1000 – 10000	100 – 10000	as many as one likes	– 500	10 – 100	number of combinations

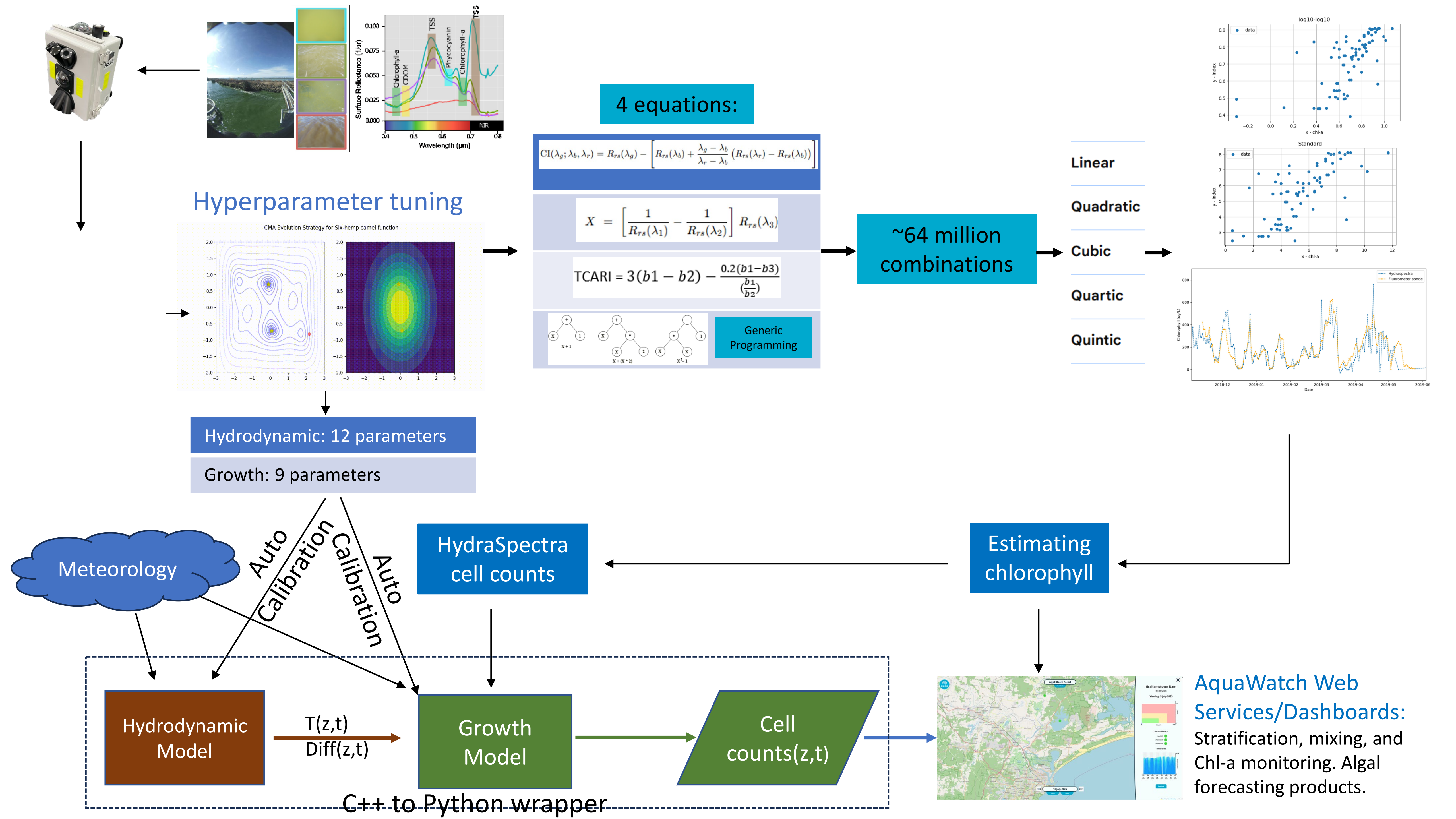
✓: Supports this feature. ▲: Works, but inefficiently. ✗: Causes an error, or has no interface.

The CmaEsSampler is based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a powerful evolutionary algorithm that's particularly well-suited for complex, multi-variate, continuous, high-dimensional, non-linear, and noisy optimization problems:

- Less likely to get stuck on local optima, CMA-ES can explore the global space effectively.
- Learns parameter correlations. This lets it explore interactions between hyperparameters intelligently.
- Useful for small trial budgets. CMA-ES often finds good solutions quicker.
- Having said that, it's trivial to plug-in another sampler and try!



Credit: Optuna core-dev. GitHub: c-bata

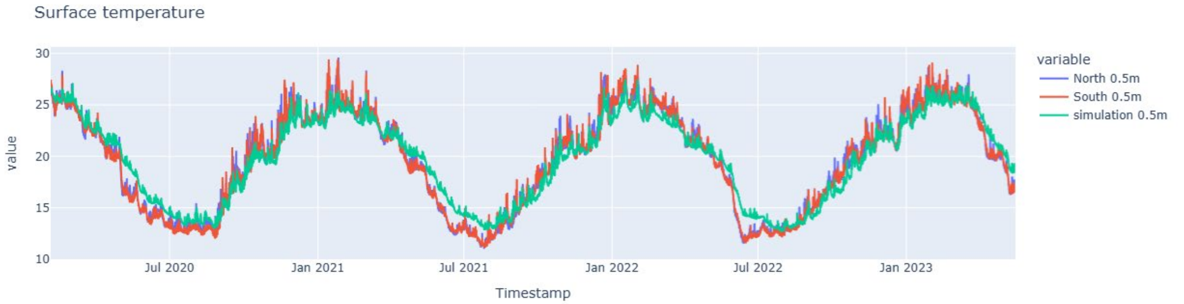


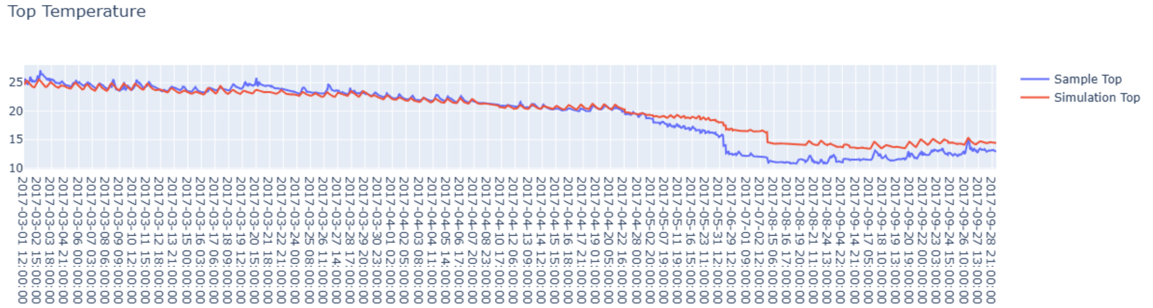
TSS
Phycocyanin
Chlorophyll-a
CDOM
Surface Reflectance (1/sr)
Wavelength (µm)
NIR
Hyperparameter tuning
CMA Evolution Strategy for Six-hemp camel function
4 equations:
$\mathrm{CI}(\lambda_g;\lambda_b,\lambda_r) = R_{rs}(\lambda_g) - \left[R_{rs}(\lambda_b) + \frac{\lambda_g-\lambda_b}{\lambda_r-\lambda_b}\left(R_{rs}(\lambda_r) - R_{rs}(\lambda_b)\right)\right]$
$X = \left[\frac{1}{R_{rs}(\lambda_1)} - \frac{1}{R_{rs}(\lambda_2)}\right] R_{rs}(\lambda_3)$
$\mathrm{TCARI} = 3(b1-b2) - \frac{0.2(b1-b3)}{(\frac{b1}{b2})}$
Generic Programming
~64 million combinations
Linear
Quadratic
Cubic
Quartic
Quintic
log10-log10
Standard
data
x - chl-a
y - index
Hydraspectra
Fluorometer sonde
Chlorophyll (ug/L)
Date
Hydrodynamic: 12 parameters
Growth: 9 parameters
Meteorology
Auto Calibration
Auto Calibration
HydraSpectra cell counts
Estimating chlorophyll
Hydrodynamic Model
T(z,t)
Diff(z,t)
Growth Model
Cell counts(z,t)
C++ to Python wrapper
AquaWatch Web Services/Dashboards:
Stratification, mixing, and Chl-a monitoring. Algal forecasting products.

CSIRO

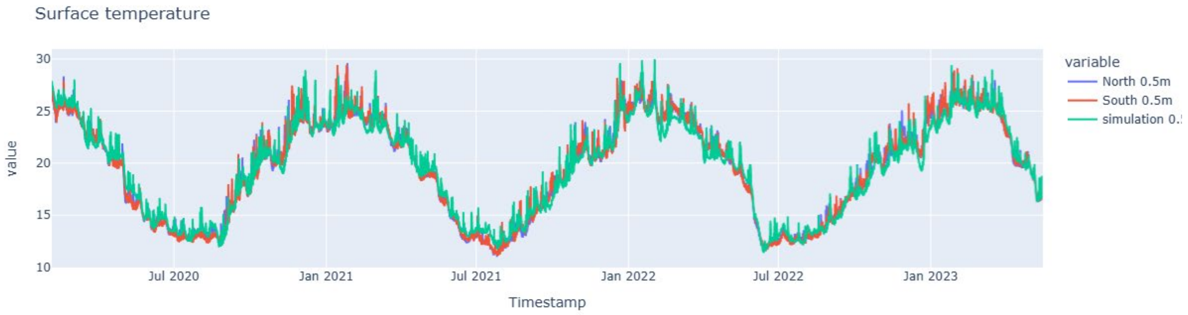
Hypertuning – Results

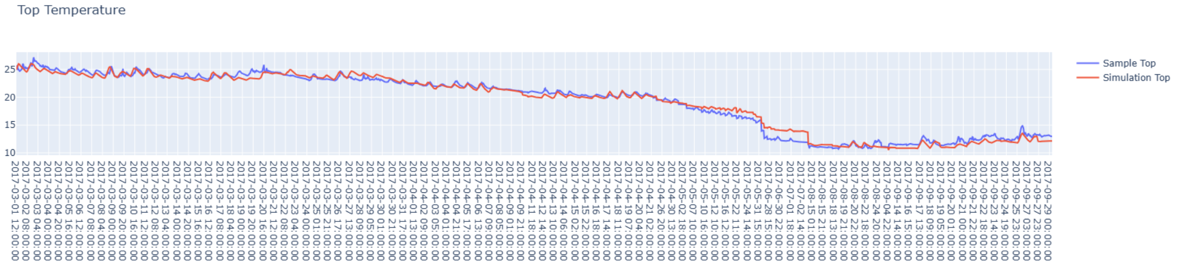
Manually



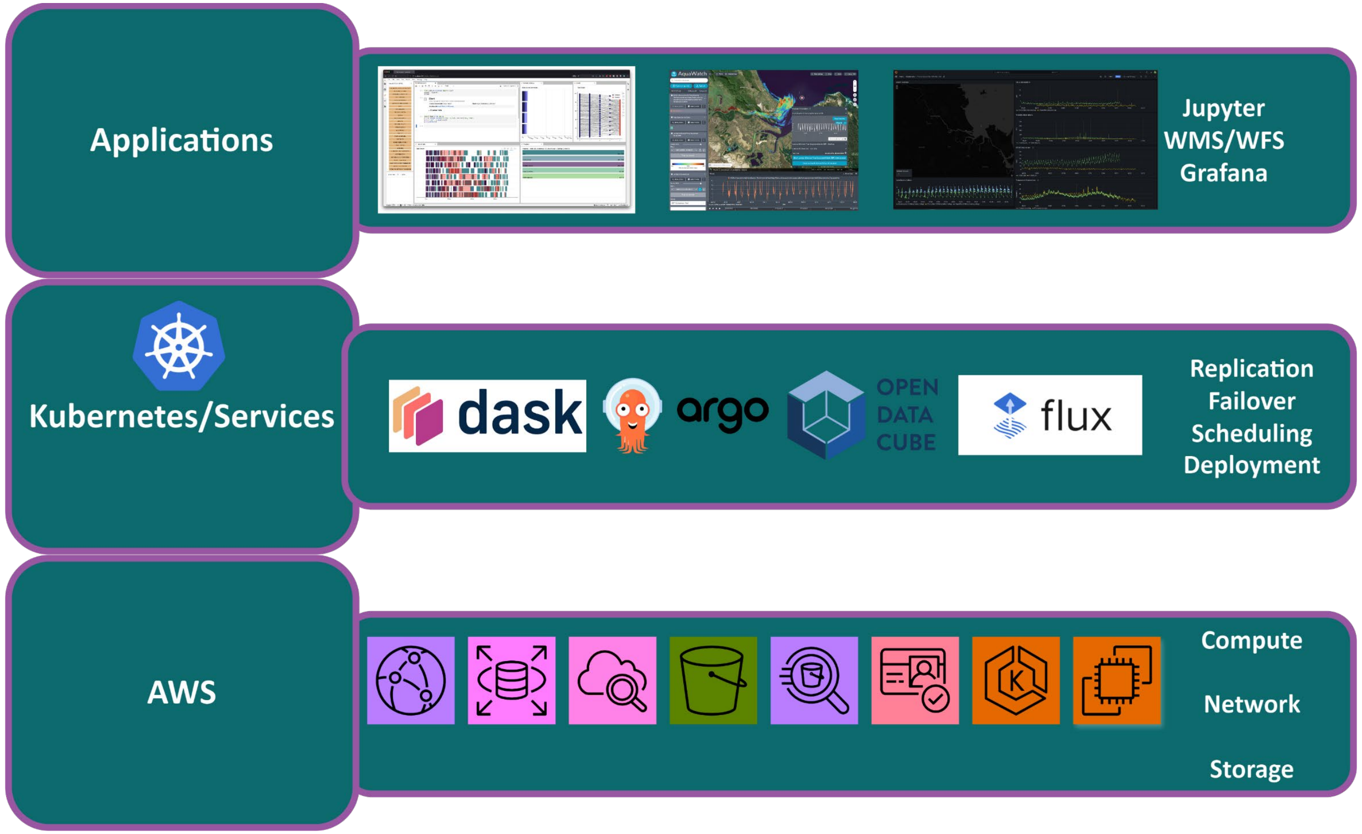


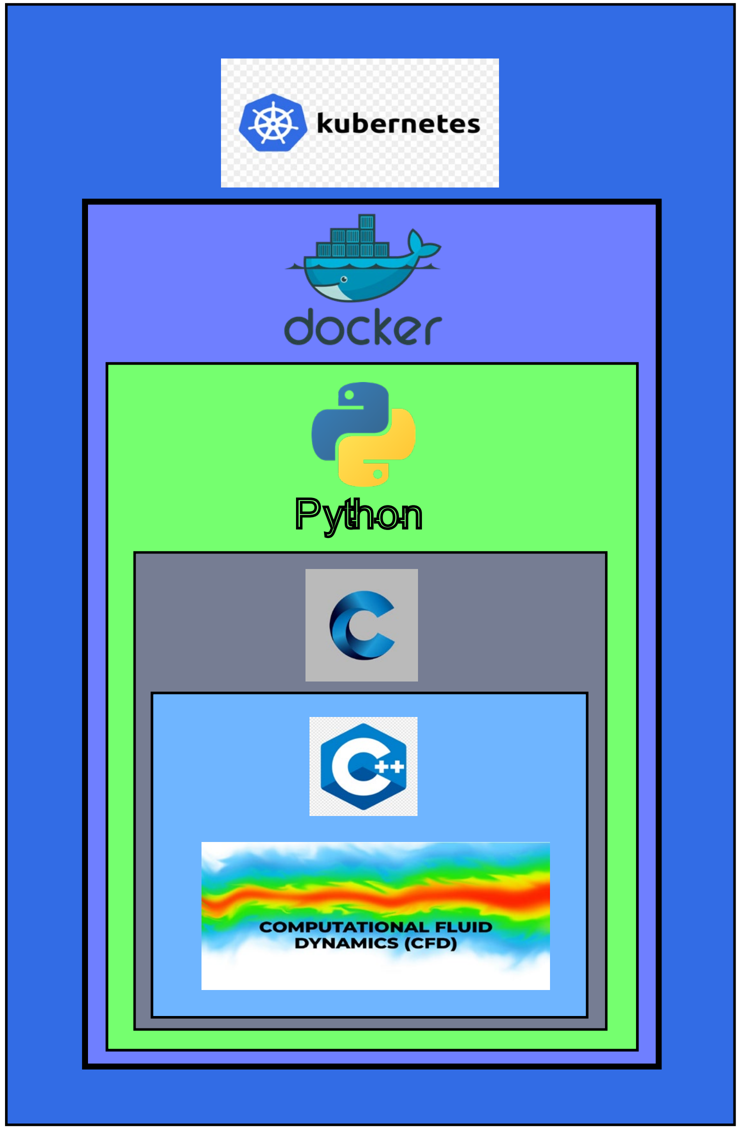
Optuna





CSIRO



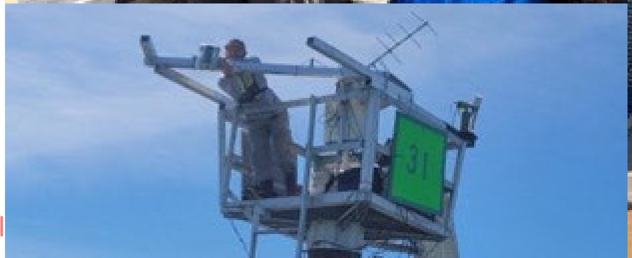


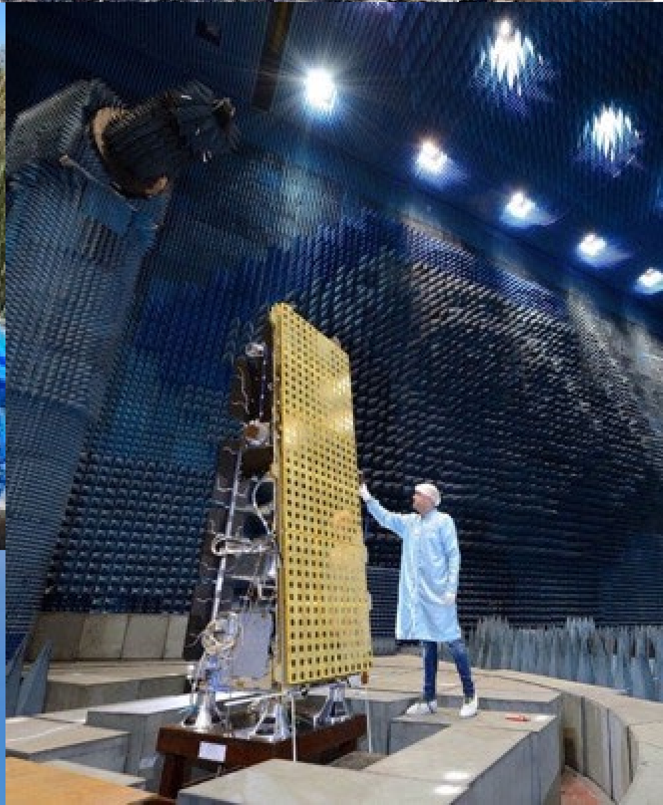
Layers of abstraction are needed to distribute CFD code for operational use

CSIRO

Partnership Opportunities

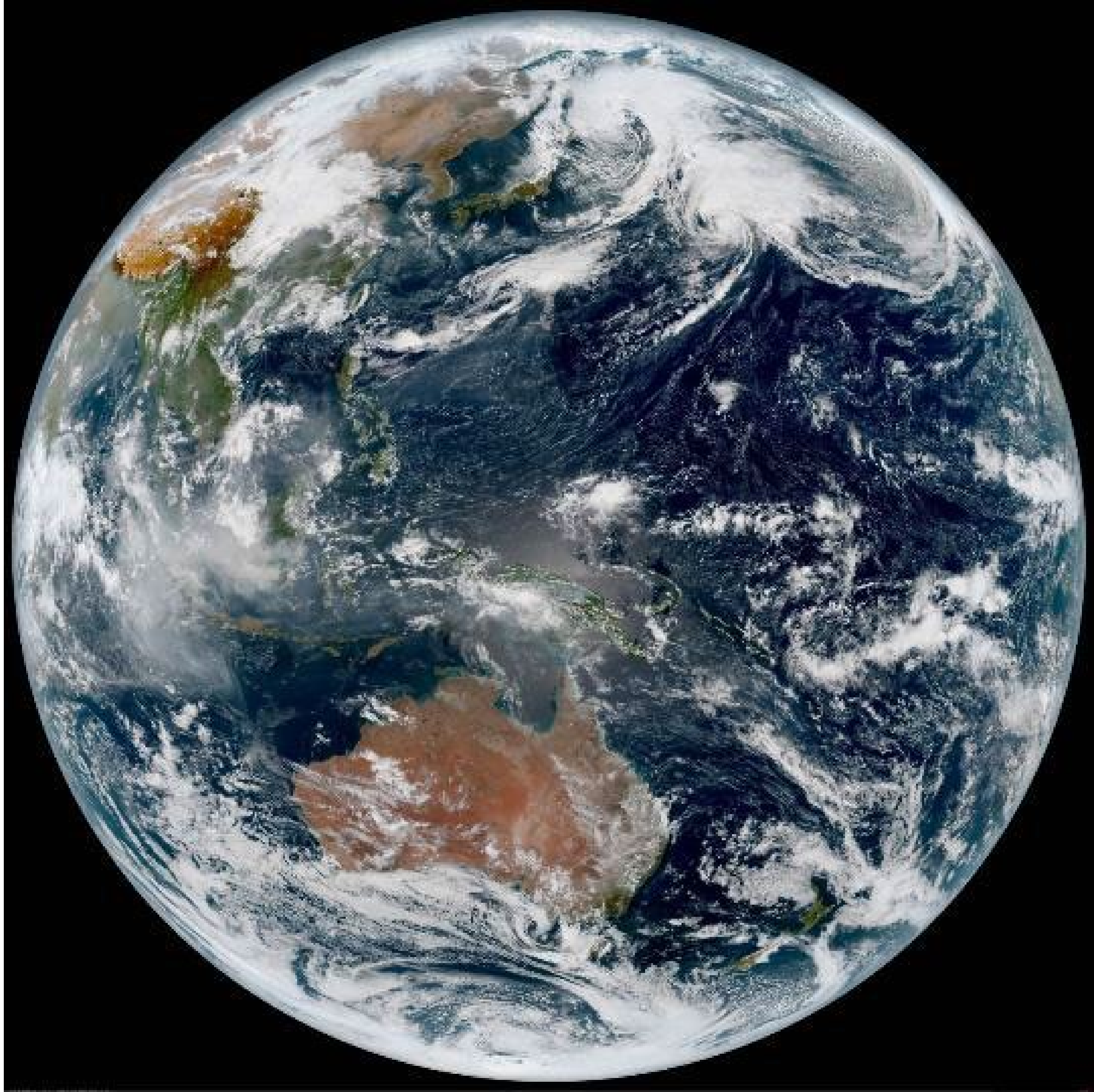
- On-ground pilot sites and product validation (eg California – UCMerced)
- R&D collaboration on key AquaWatch technology/science areas
- Citizen-science collaborations w. First Nations and Education Organisations
- Implementation of integrated AquaWatch system in new countries
- Cloud-computing data analytics platform implementation in host organisation
- Ground-to-Space in-situ data-relay trials and implementation
- Space Optics Collaborations (via CSIRO Manufacturing)
- Earth observation constellation partnership and development via eg. PPP
- Space Segment - Dual-use Options





CSIRO

THANK YOU



CSIRO ENVIRONMENT & AQUAWATCH AUSTRALIA

Dr Duy Nguyen

E: duy.nguyen@csiro.au