Software Infrastructure, Tools and Hints for Integrated Modelling

P. Abreu / O. Hoenen

Paulo.Abreu@iter.org

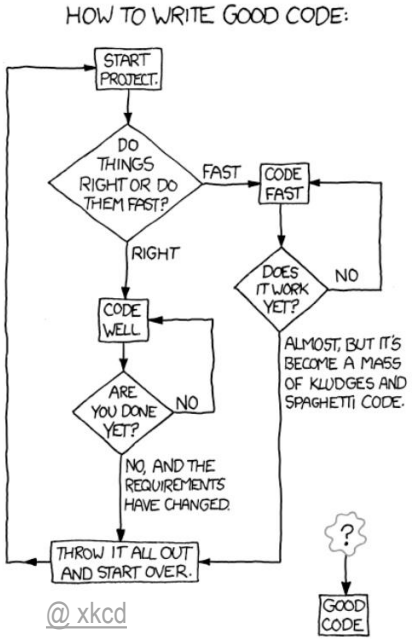
ITER Organization

Disclaimer: The views and opinions expressed herein do not necessarily reflect those of the ITER Organization

Preamble

- Who am I?
  - Physics Engineer (PhD in multiscale PIC, from GPU to Grid)
  - **JET intershot RO** from 2012 to 2021
  - Scientific Data Processing Coordinator at ITER (Science Division)
    - **Synthetic Diagnostics** Modelling Coordinator
    - **In-pulse Analysis Workflows** Project Manager

- What this lecture is about?
  - **Integrated Modelling** (IM) from the supporting software standpoint
  - Some hints on software development good practices
  - Specific tools and tips with the **IMAS** eco-system
  - Two **example applications**: H&CD workflow and WEST PRC



Food for thoughts, no canned recipes!

Introducing the context

What is Integrated Modelling?

Different physics, different scales (spatial & temporal), numerical approximations, sub-systems, …

Code integration into a common environment, coupling and execution.

Integrated Modelling refers to the process of combining multiple models (often from different disciplines, domains, or systems) into a unified framework to better understand, simulate, or predict complex phenomena. This approach is especially useful when dealing with systems that are interconnected, such as environmental systems, socio-economic systems, or large-scale engineering projects.

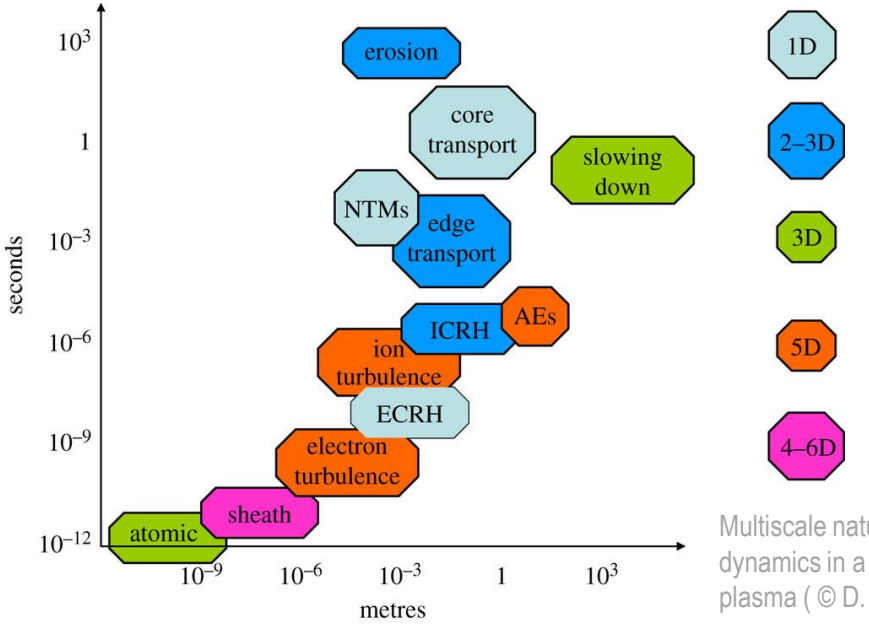
[ChatGPT]

When traditional modelling does not capture sufficient physics or is too expensive when applied to the entire tokamak/plant system.

Collaboration of scientists with different background, language, datasets, coding style, …

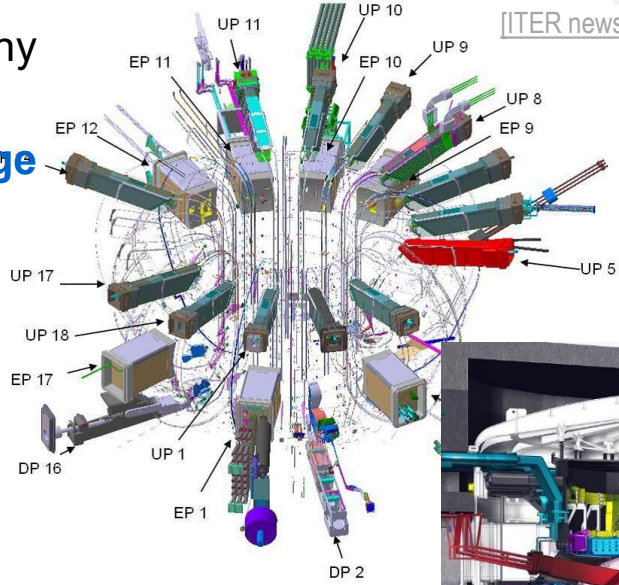
Why using IM for ITER?

- A tokamak is a complex device with many sub-systems and data sources
- A plasma spans effects over a very **large range of scales** (in time and distance)



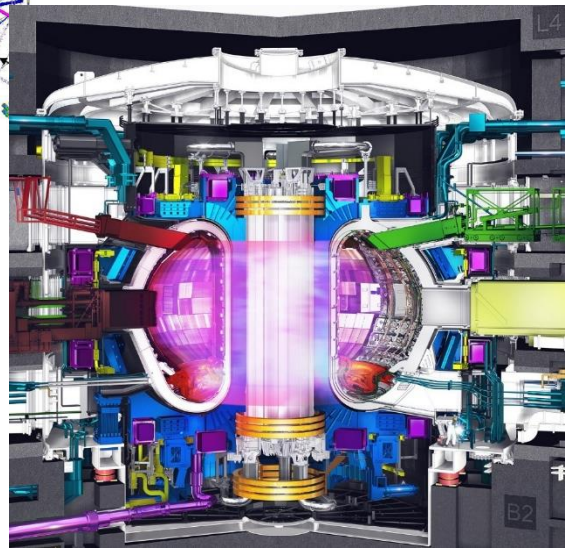
Multiscale nature of the dynamics in a fusion plasma (© D. Coster)

[ITER newsline#3360]



~ 60 major diagnostic systems

~ 20 actuators



Key software challenges in IM

- **Integration**: API compatibility, scale bridging
- **Data management**: consistency, provenance tracking, memory management
- **Validation & Verification (V&V)**: integration testing, uncertainty propagation, benchmarking
- **Collaboration**: cross disciplinary communication, source code access, IP
- **Performance**: I/O overheads, load balancing, scalability
- **Maintenance**: version control, dependency management, portability, error handling

- **Legacy software**: static typing (old-Fortran), lack of documentation, tests, portability and developer's availability

General tips and hints

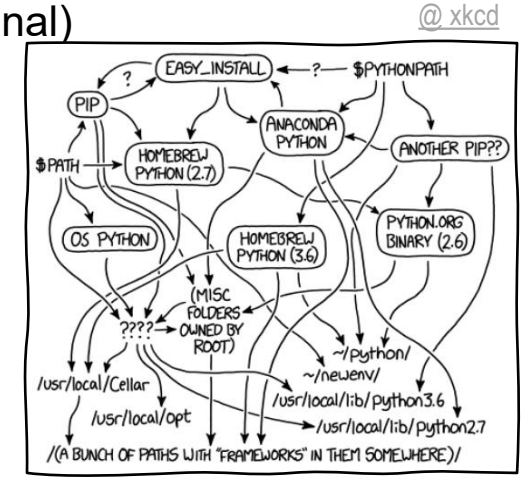
The case for open-source licensing software

- Improve **collaboration and communication**
  - Focus on technical issues, not administrative ones
  - Open to collaborators outside of the domain (AI/ML, data-science)
  - Avoid duplicated efforts, share and find issues and solutions, …
- Benefit from a **huge eco-system from free development platforms**
  - Code versioning, collaboration channels (issues/discussions), project management
  - CI/CD, packaging distributions, code documentation
- Thinking about integration (IM)
  - Take into account the type of IM platform and the need to modify models
  - Consider license compatibility (permissive, intermediate, copyleft)

⚠ Reminder: **no license == no permissions** for users (unless specifically granted)

The importance of a portable build

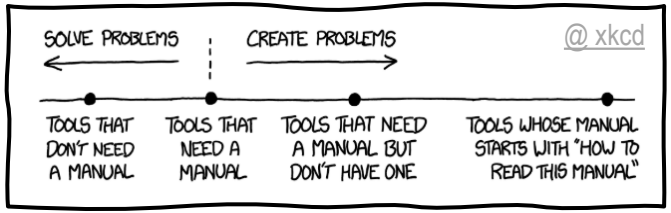
- Why
  - **Lower entry barrier** → improve uptake (tradeoff with user support effort)
  - Higher complexity in IM when combining codes from different developers together
- What is important
  - Avoid site-specific configurations
  - **Describe and resolve dependencies** (closed ones as optional)
  - Stick to most common build environments for the language
  - Target most common usage first (OS, packaging, …)
- Consider package managers (if many dependencies)
  - Spack, EasyBuild
  - Conda, Docker, …



The importance of documenting software

- Why
  - Integrating codes from **other developers**
  - Research codes tend to be **multi-purposes**
- What is important
  - General context, scope + link to papers
  - Describe inputs, ouputs, parameters (use)
  - Separate user's from developer's concerns
  - Keep the documentation close to the source code
  - Easy to navigate and read (practical examples)

Standardese
Pydoc
Doxygen
Swagger
Sphinx
f90doc
MkDocs
FORD
Pdoc



The importance of automated testing

- Testing individual models
  - IM software depend on reliability of individual models
  - Testing a single model is less complex and costly
  - Unit testing (routine level), use fixtures
  - Integration testing (user level)

- Continuous integration (CI)
  - Make sure the model can compile, install, generate the doc
  - Automate the execution of tests (every commit/pull-request/…)

- Testing IM code
  - Prerequisite: each sub-models are passing their own tests
  - Check model ↔ API (unit), model ↔ model (integration)
  - Verify invariants, analytic cases

pytest Catch2 doctest
doctest Google Test pFUnit
unittest naturalFRUIT

Coupling approaches

coupling **direction**

- **One-way**: A→B, simplest, DAG, does not capture feedback effects
- **Two-way**: A↔B, feedback loops, risk of deadlock (synchronization)

model **execution** & data **exchange**

- **Loose coupling**: A,B run independently, infrequent data exchange at fixed intervals (usually via files), may not capture fast effects
- **Tight coupling**: A,B run simultaneously, data exchange at each iteration (via shared memory or message passing), needs careful coordination

other aspects

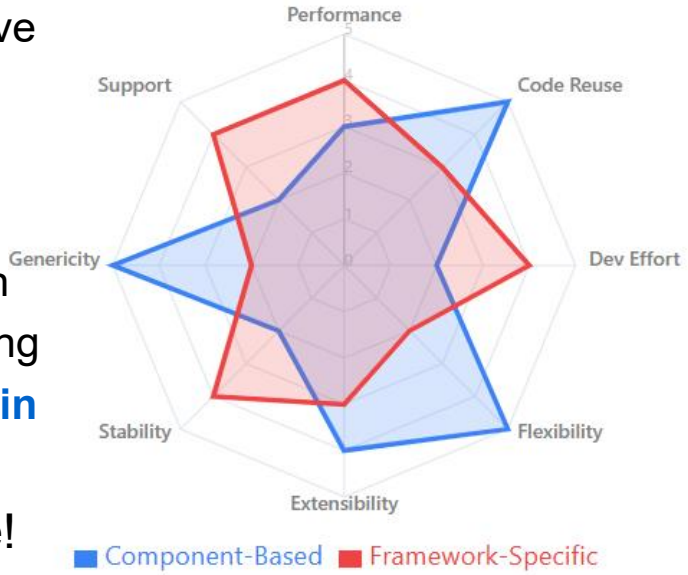
- **Embedding**: B is part of A, tightest integration, important modifications to pre-existing models and increased maintenance costs
- **Hierarchical coupling**: **M**acro–**m**icro (multiscale systems), **M** provides boundary conditions to **m**, requires good load-balancing
- **Service-oriented**: A and B are web-services called on-demand, can be distributed, may introduce communication overheads

Models coupling platforms

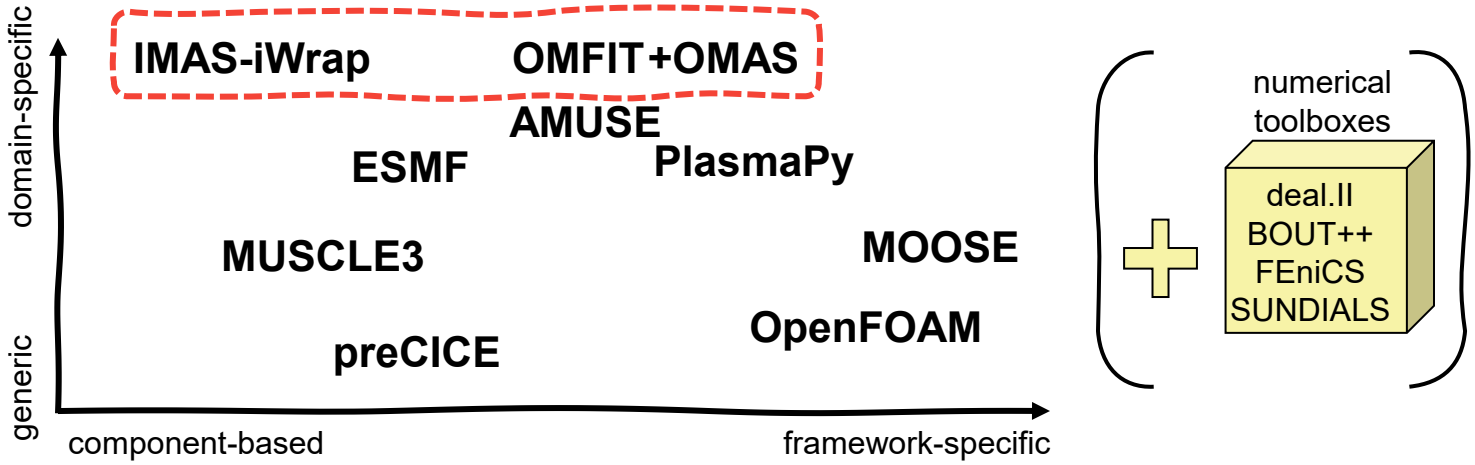
- **The component-based approach**
  - Components expose a **well-defined API** (control and data exchange)
  - Existing models **encapsulated** as components (hide internal implementation details)
  - Components can be **combined** in different configurations (reuse, composability)
  - Performance **overheads, stability** is harder to achieve

- **Framework-specific approach**
  - **Built-in** physics models and solvers, good support
  - **Unified** development environment for tight integration
  - Usually more adapted to **high performance** computing
  - Existing model **needs rewrite, learning curve, lock-in**

→ **Combining** different approaches is possible!



Examples of simulation platforms



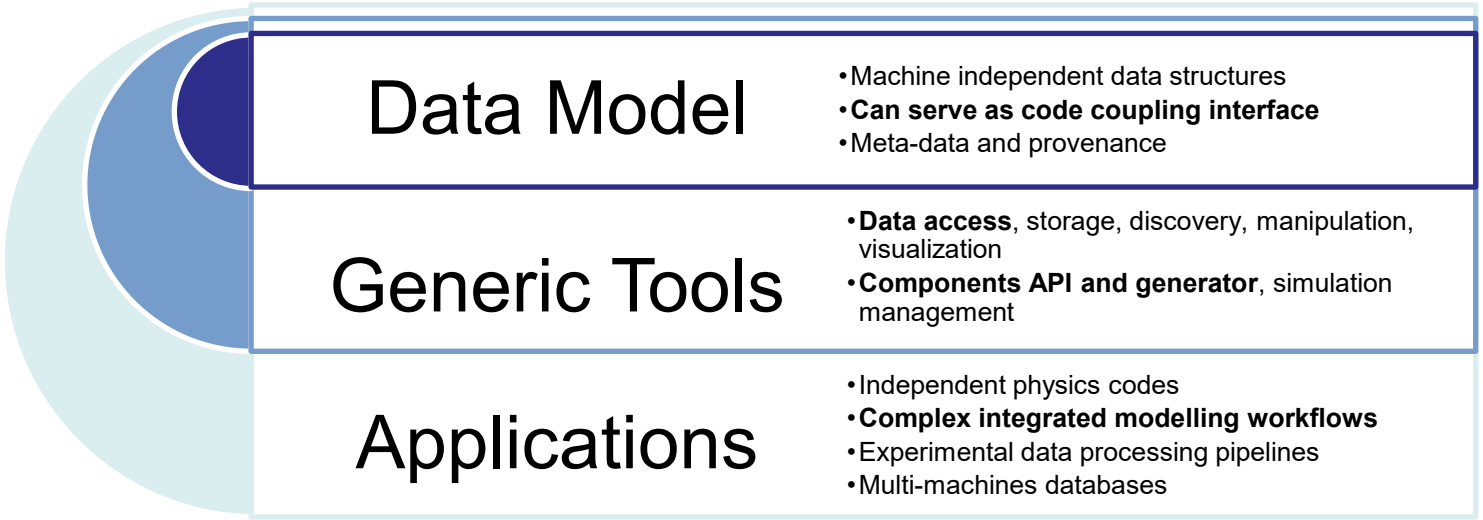
- Different focus (multi-physics, multiscale, CFD, domain specific, …)
- In practice the categorization is less clear (many options, bridges)
- Different programming languages support (natively)
- Standards: FMI (systems engineering), OpenMI (environmental), **IMAS (fusion)**
  - ↳ Avoids **lock-in** and risks of a software being **abandoned**

IMAS

What is IMAS?

Integrated Modelling & Analysis Suite is the ***collection of physics software*** *that will be used to* ***support ITER operations and research*** *as defined in the ITER Integrated Modelling Programme.*

*…****plasma modelling tools for systematic planning*** *(evaluate candidate plasma operating scenarios and assist in the development of plasma control strategies)* ***and analysis of each ITER discharge.***



[F. Imbeaux *et al* 2015 *Nucl. Fusion* **55**]

Data Model

Covering **simulation** and **experimental** data

- Set of **Interface Data Structure** (**IDS**)
- Implemented in the **Data Dictionary** (**DD**)

Made **machine agnostic** by design

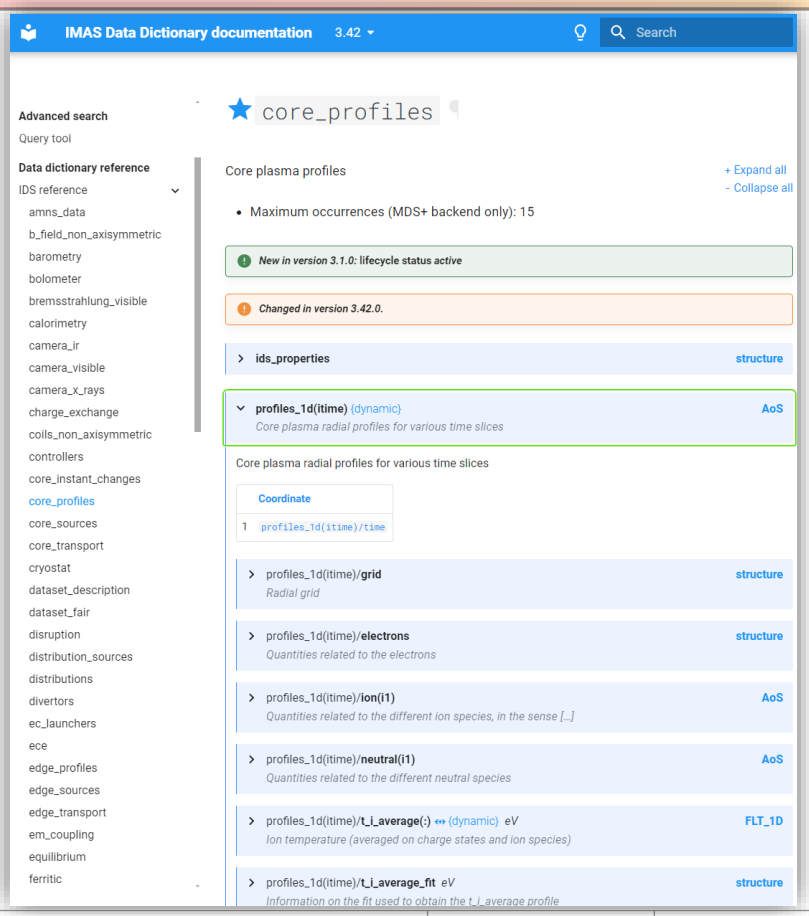
- Use in experiments (ITER, WEST)
- **Mapping** data from existing experiments

→ **validation** of ITER software in realistic conditions

Can serve as a **data standard** for fusion

- Ease **comparison**, **benchmarks**
- Multi-machines databases → **data science**
- Step towards **FAIR*** principles

  * **F**indable **A**ccessible **I**nteroperable **R**eusable



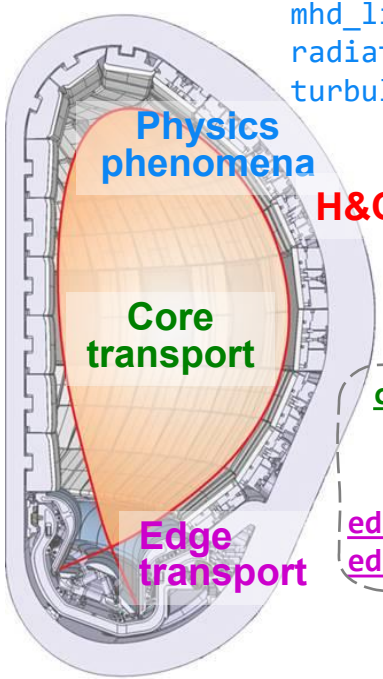
IDS Coverage (4.0.0)

Diagnostics

barometry bolometer
bremsstrahlung_visible calorimetry
camera_ir camera_visible camera_x_rays
charge_exchange ece focs hard_x_rays
interferometer langmuir_probes **magnetics**
mse neutron_diagnostic
operational_instrumentation polarimeter
reflectometer_fluctuation
reflectometer_profile refractometer
soft_x_rays spectrometer_mass
spectrometer_uv spectrometer_visible
spectrometer_x_ray_crystal
thomson_scattering

Electro-Magnetics

b_field_non_axisymmetric
coils_non_axisymmetric em_coupling
equilibrium ferritic iron_core
pf_active pf_passive pf_plasma tf



Physics phenomena

disruption gyrokinetics_local mhd
mhd_linear ntms plasma_initiation
radiation runaway_electrons sawteeth
turbulence

H&CD

distribution_sources distributions
ec_launchers
ic_antennas lh_antennas **nbi** waves

Fuelling

gas_injection gas_pumping
pellets spi

Core transport

core_instant_changes **core_profiles**
core_sources **core_transport**
plasma_profiles plasma_sources
plasma_transport

Edge transport

edge_profiles **edge_sources**
edge_transport

Data Management

amns_data controllers dataset_description
dataset_fair pulse_schedule real_time_data
summary temporary transport_solver_numerics
workflow

Other plant systems

cryostat divertors **wall**

The DD in practice

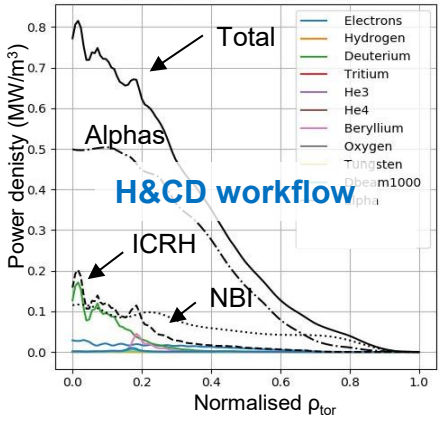
- It's now **open access**!
  - Find the documentation at https://imas-data-dictionary.readthedocs.io/
  - Join to ask questions, raise issues, propose extensions
  - https://github.com/iterorganization/IMAS-Data-Dictionary →
  - To install: `pip install imas-data-dictionary`
    - Versions at the moment: 3.39, 3.42, 4.0

- A **standard for describing tokamak data**
  - Comprehensive, explicit definition → ease **collaboration**
  - Need to **improve backward compatibility** (`alpha`→`active` IDS)

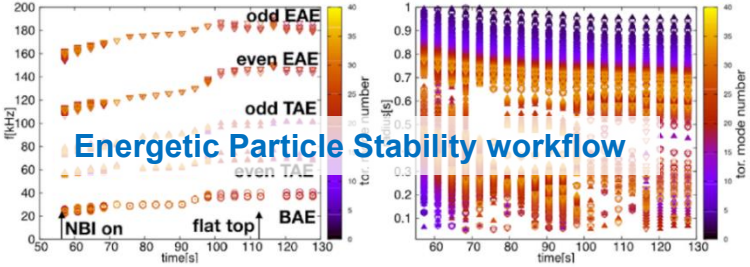
- A **standard for models interface**
  - Class of models sharing the same input/output (IDS) → **code re-use, coupling**
  - Some IDS have too many alternate descriptions → **interoperability issue**

Example of Applications

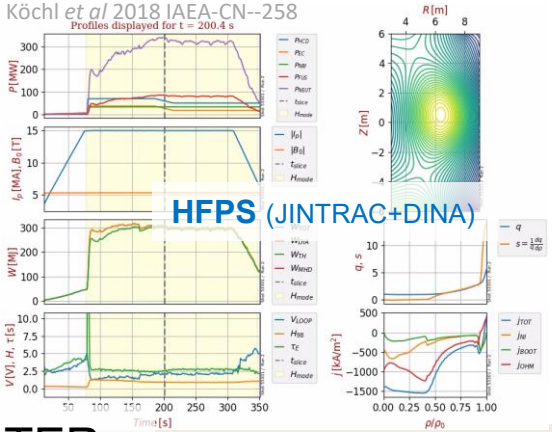
M. Schneider *et al* 2021 *Nucl. Fusion* **61** 126058



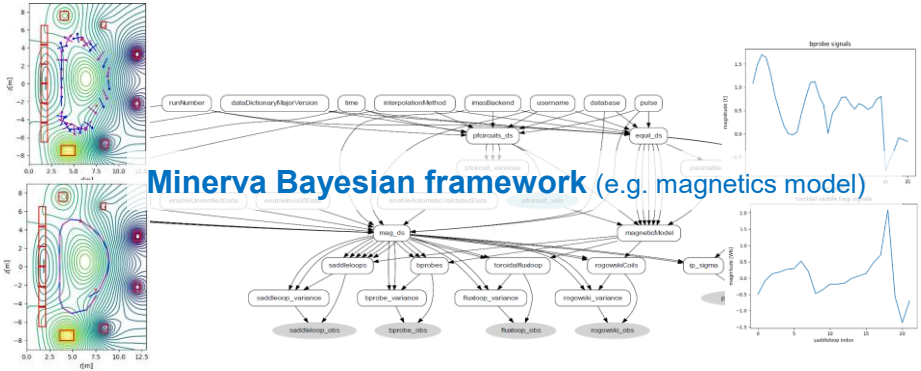
V.-A. Popa *et al* 2023 *Nucl. Fusion* **63** 126008

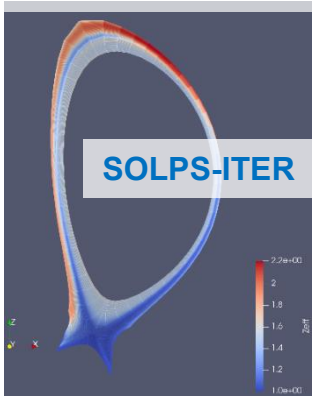


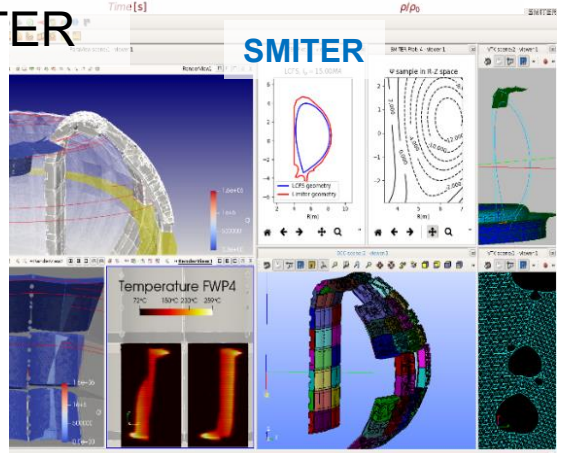
F. Köchl *et al* 2018 IAEA-CN--258



IMAS codes and workflows used at ITER







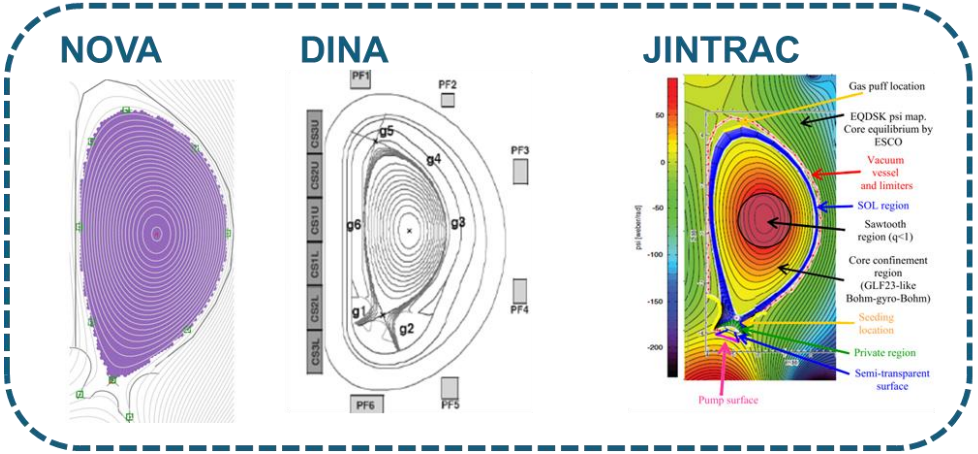
J.-S. Park et al 2021 Nucl. Fusion 61, 016021

L. Kos *et al* 2019 Fusion Eng. Des., 146

Multi-fidelity

[S. Mcintosh]

Meta-models <<



<< First principles models

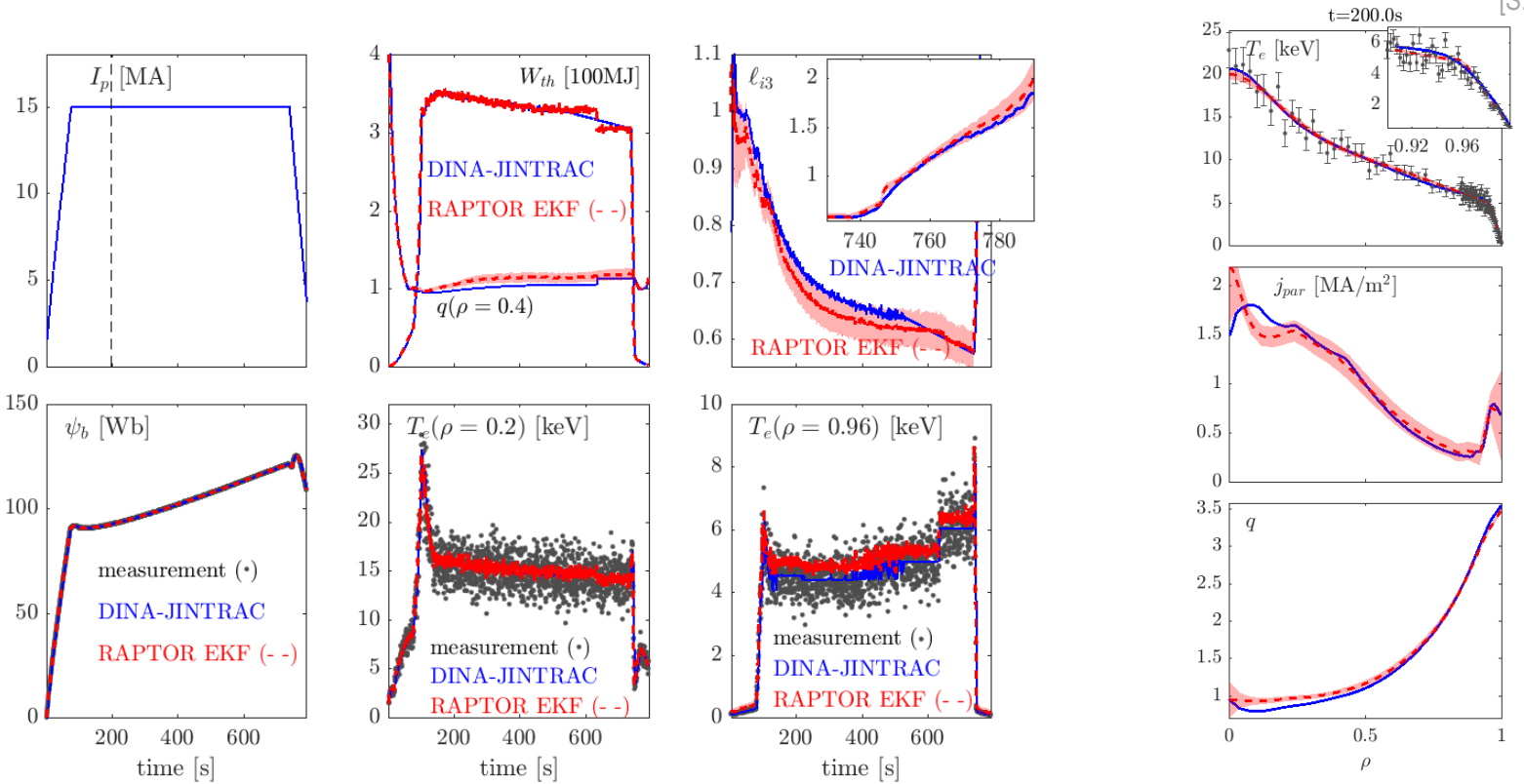
→ Codes may be replaced by others that share the **same IDS interface**

⚠ Current IDS data access may not fit pseudo real-time fast models

Interface interoperability requires additional work during integration

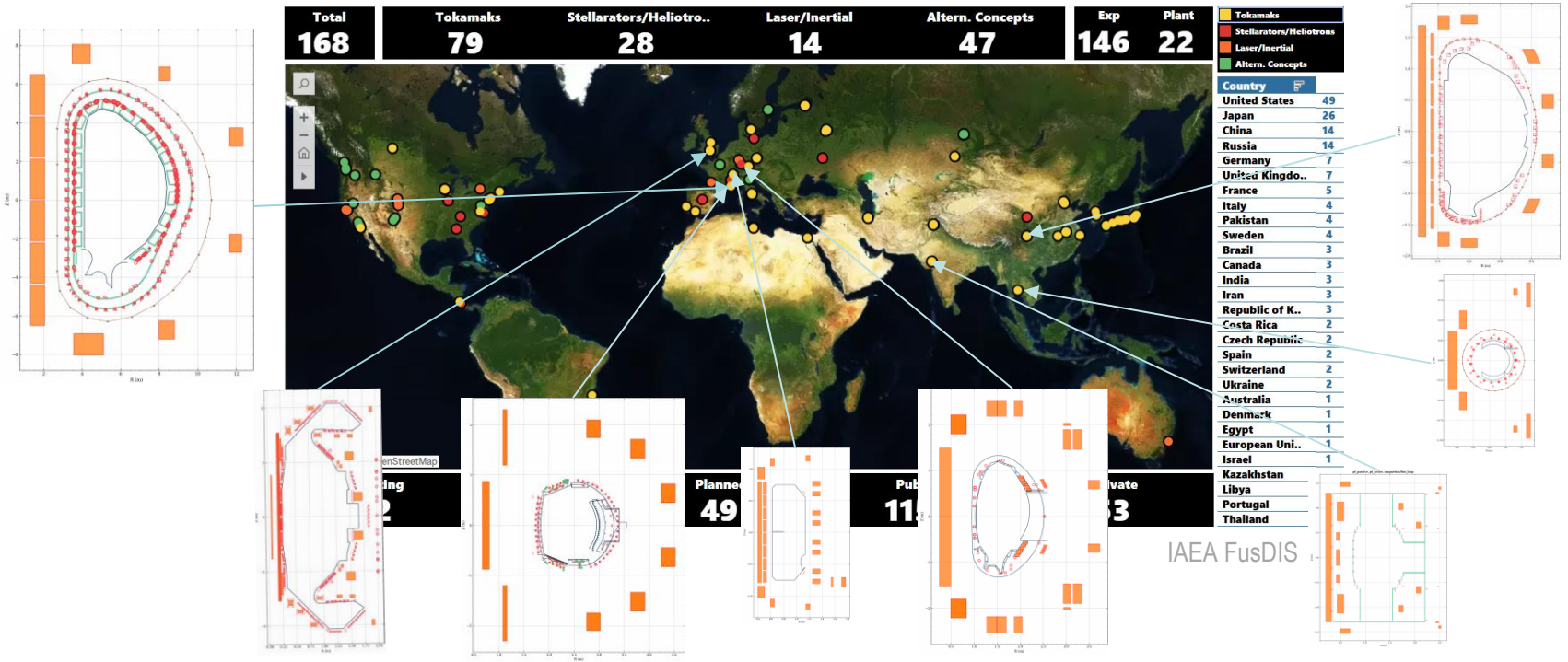
Preparing ITER operation with synthetic diagnostics

[S. Van Mulders]



→ Synthetic diagnostic models require **Machine Description** in IMAS

Application to other tokamaks



→ Test and **validate** IMAS applications in-situ (prepare for ITER operation)

→ New **discoveries** through access to a wide range of **parameters/shapes/materials…**

Provenance capture in IMAS

Aims

- Reproducibility of results
- Re-use of data
- Data lineage
  - allow propagation of errors' correction
  - allow attribution (e.g. publications)

Systematic recording when possible

- Tradeoff with overheads, requirements for automation tools
- Separation data / metadata

Tracking information at various levels

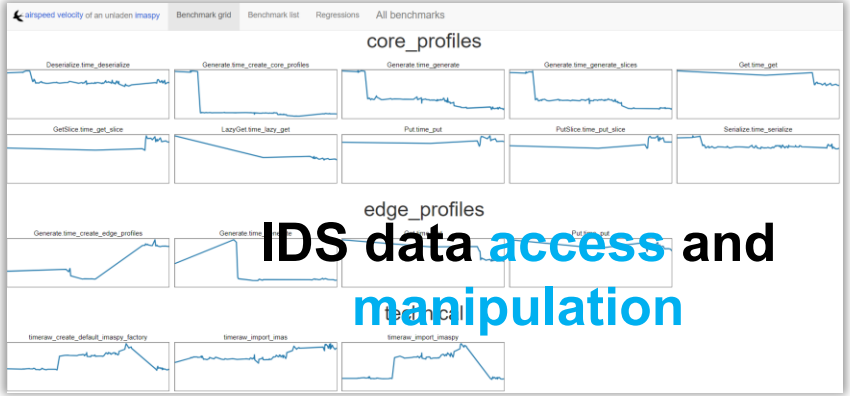
Quantity
- Field of an IDS
- Links to quantities it was derived from

IDS
- Domain specific self consistent set of fields
- Data Model version specific
- Produced by a unique process / code

Dataset
- Coherent set of IDS
- Describing the results / outputs of a given simulation or experiment
- License, ownership, etc…

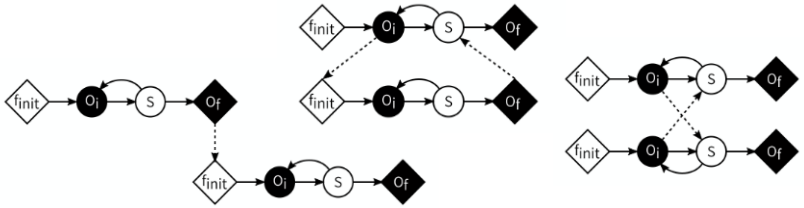
Simulation
- Inputs, outputs, configuration, description of a given simulation or experiment
- Quality flags, usage

Generic tools in IMAS

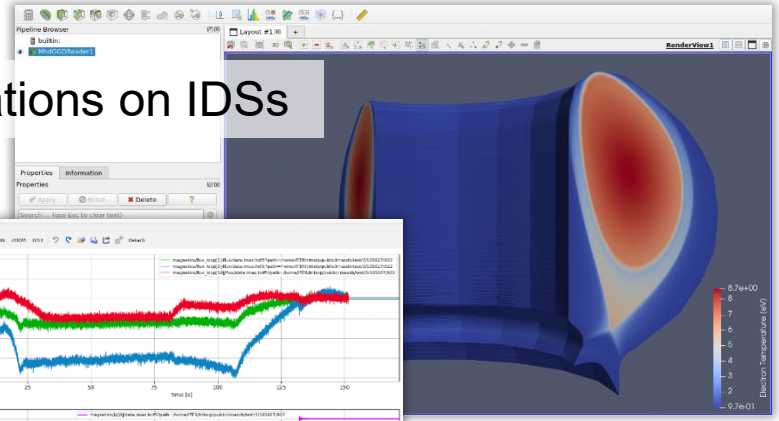


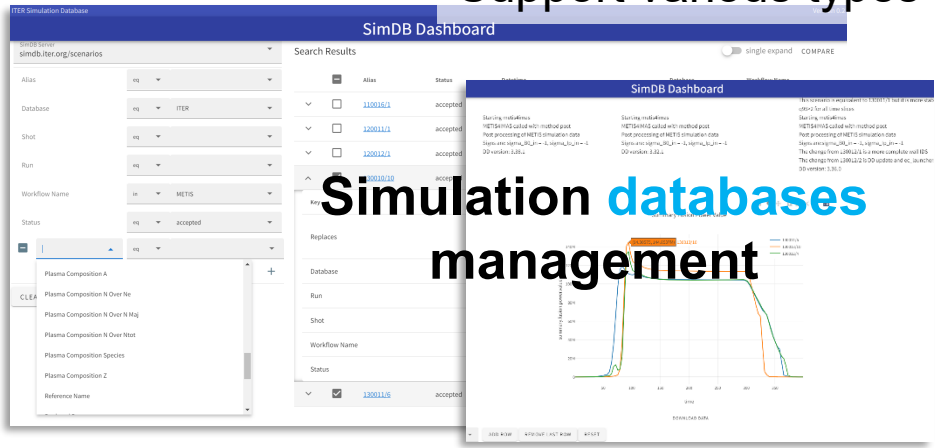
IDS data access and manipulation

Code coupling and workflow frameworks

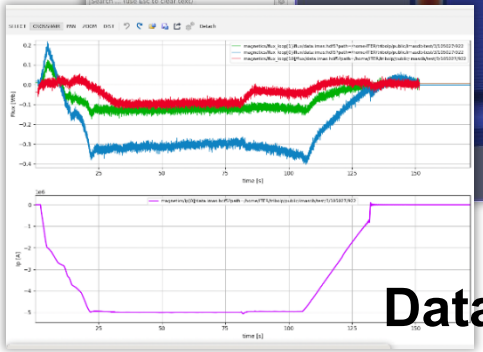


Support various types of operations on IDSs





Simulation databases management



Data visualization

IDS data access

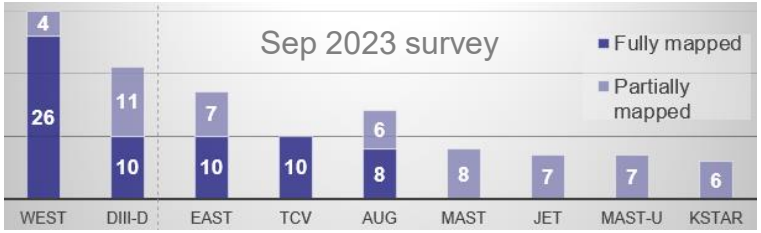
- Via an API to separate the code from data storage details
  - Support for most-used languages: Fortran, C++, Matlab, Java and Python
  - Support for data stored in MDSplus, HDF5, plain text, memory
  - Remote access with UDA, serialization with flexbuffers

`imas-core`

open-source: OMAS, IMAS-Python

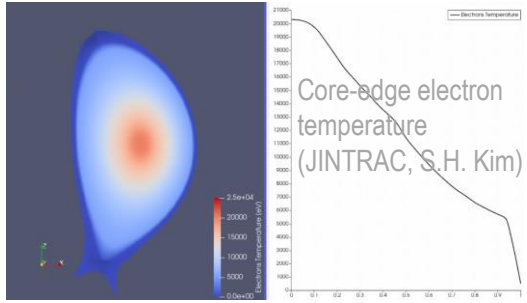
- IMAS-Python in practice
  - **DD version selection and conversion** at runtime
  - Embed DD **metadata**, allow ducktyping, **lazy-loading**, work with **Xarray**
  - I/O with **`imas-core`** (if available) or **netCDF** (self-described, tensors)

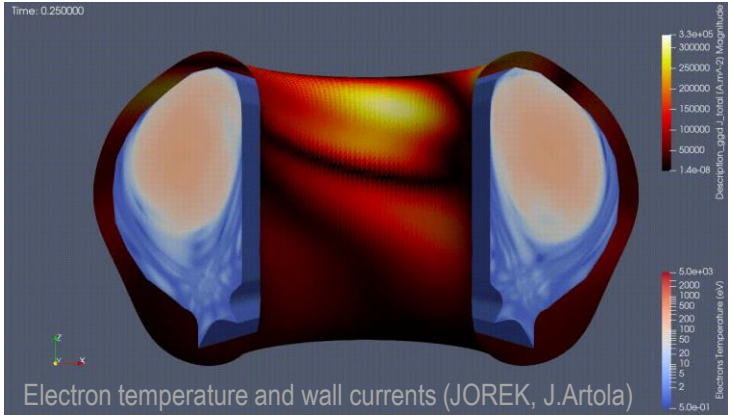
- Experimental data
  - Via mappings (ad-hoc or UDA-JSON)
  - Mapping effort is ongoing



IDS data visualization

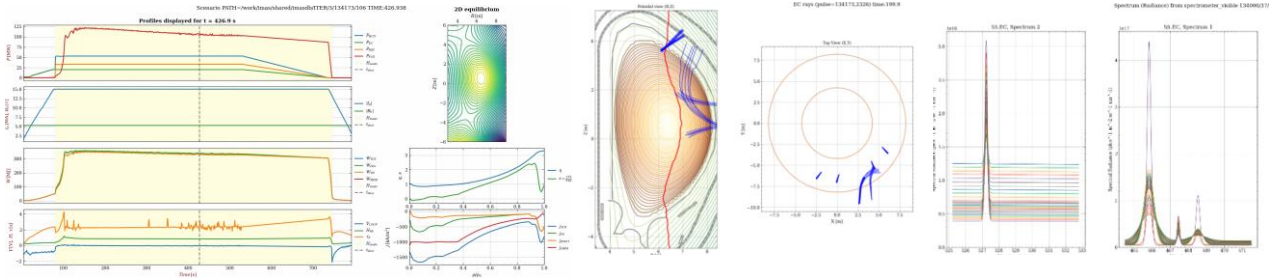
- Get IDS data in Python or Matlab and build your own, or re-use existing tools



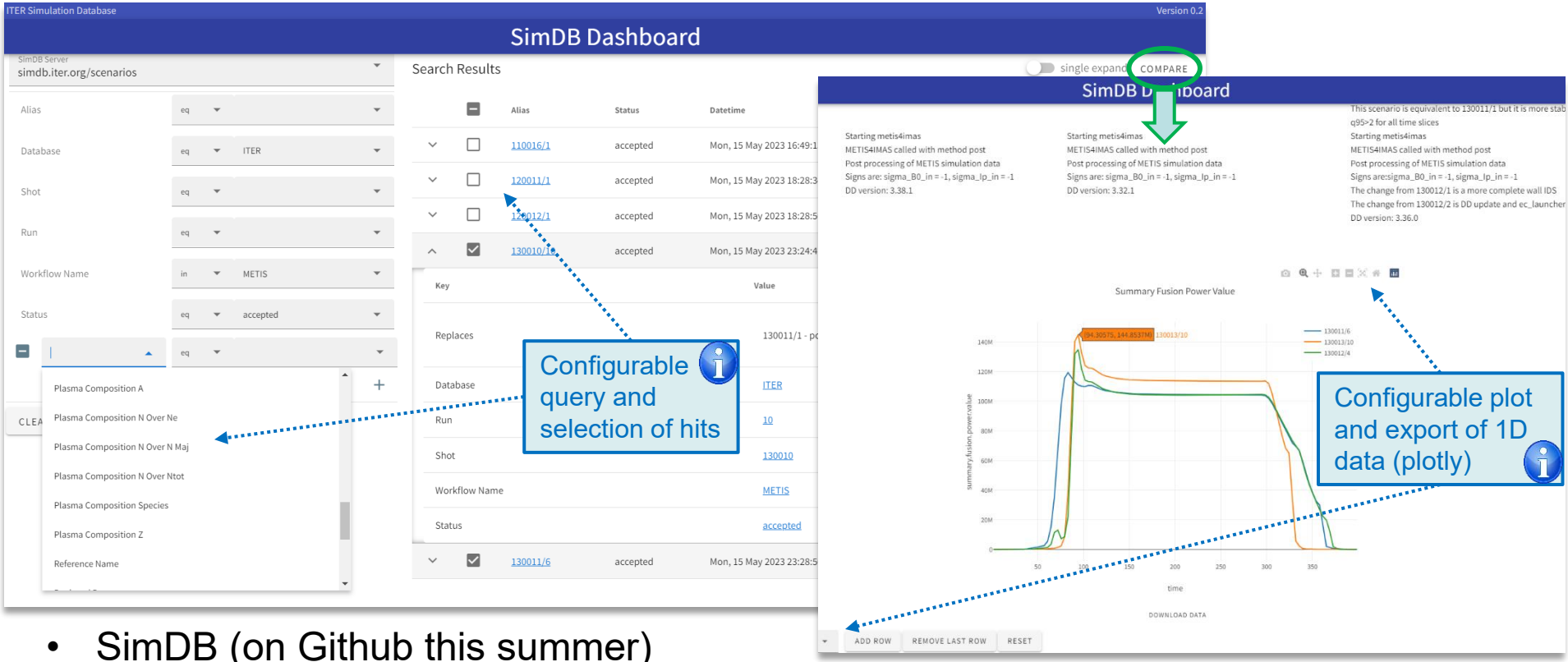


- IMAS-ParaView
  - Open-source plugin to visualize GGD data in VTK
  - Other data source readers (non-GGD data)

- IDStools helper scripts
  - CLI, matplotlib
  - Open-source soon…



IMAS simulation management

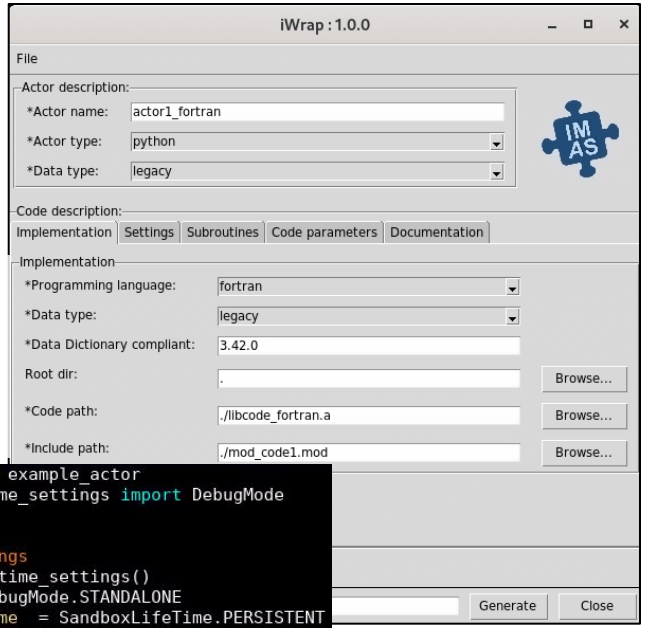


- SimDB (on Github this summer)
  - CLI client (ingest and queries), remote server (REST API), web frontend for queries

Rapid prototyping of workflows in Python

iWrap (IMAS-specific, **component-based approach**)

- Tool to generates IMAS Python actors with a standard API
  - From existing **Fortran/C++/Java** code (I/O = **IDS+codeparam**)
- Native code description in GUI or directly in Yaml
- Code runs from **shared library** or **executable** (MPI, debug)
- Can generate **skeleton** for manual wrapping
- Workflow **implemented** as a Python script




```
        parameters: ./input/input_physics.xml
        schema: ./input/input_physics.xsd
arguments:
-   name: equilibrium_in
    type: equilibrium
    intent: IN
-   name: equilibrium_out
    type: equilibrium
    intent: OUT
documentation: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
    consequat. '
settings:
    compiler_cmd: $FC
    mpi_compiler_cmd:
    open_mp_switch: false
    extra_libraries:
        pkg_config_defined:
            - xmllib
        path_defined:
```

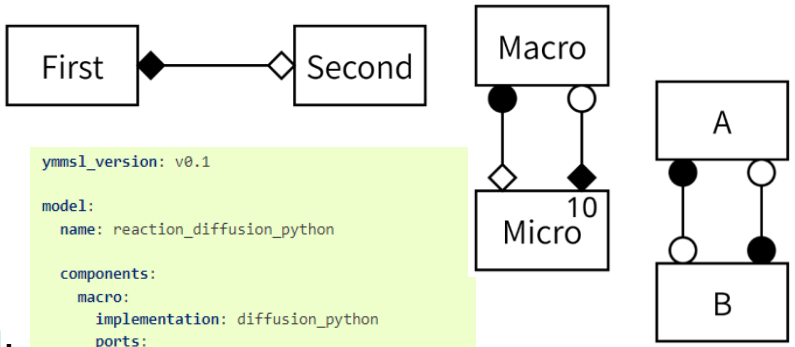
```
from example_level2.actor import example_actor
from example_level2.common.runtime_settings import DebugMode
actor = example_actor()

# Configuration of runtime settings
runtime_settings = actor.get_runtime_settings()
runtime_settings.debug_mode = DebugMode.STANDALONE
runtime_settings.sandbox.life_time  = SandboxLifeTime.PERSISTENT

# Configuration of code parameters
code_parameters = self.actor.get_code_parameters()
code_parameters.set_parameter('factor_X', 0.5)
actor.initialize(runtime_settings, code_parameters)

input_eq = DB.get_slice("equilibrium",time,CLOSEST_SAMPLE)
# Call of MAIN
output_eq = actor(input_eq)
# Saving output data
DBout.put(output_eq)
actor.finalize()
```

Building more complex actors and workflows

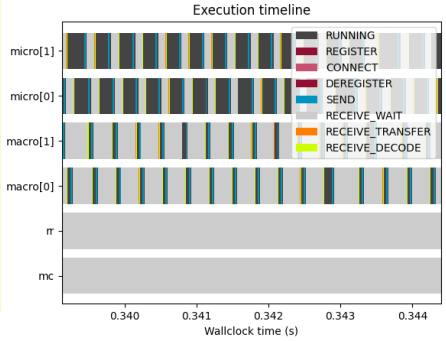
Persistent Actor Framework

- Rely on MUSCLE3 library (open source)
  Actors developed as independent programs
  - Suited for MPI for keeping the code's internal state
  - Serialized IDS are transferred by MUSCLE3 (TCP)
- Workflow declared in a configuration file (yMMSL)
- Features check-point and restart mechanism, profiling, detection of shared resources, running from containers

Various coupling patterns




```
ymmsl_version: v0.1

model:
  name: reaction_diffusion_python

  components:
    macro:
      implementation: diffusion_python
      ports:
        o_i: state_out
        s: state_in

    micro:
      implementation: reaction_python
      ports:
        f_init: initial_state
        o_f: final_state

  conduits:
    macro.state_out: micro.initial_sta
    micro.final_state: macro.state_in

resources:
  macro:
    threads: 1
  micro:
    threads: 1
```



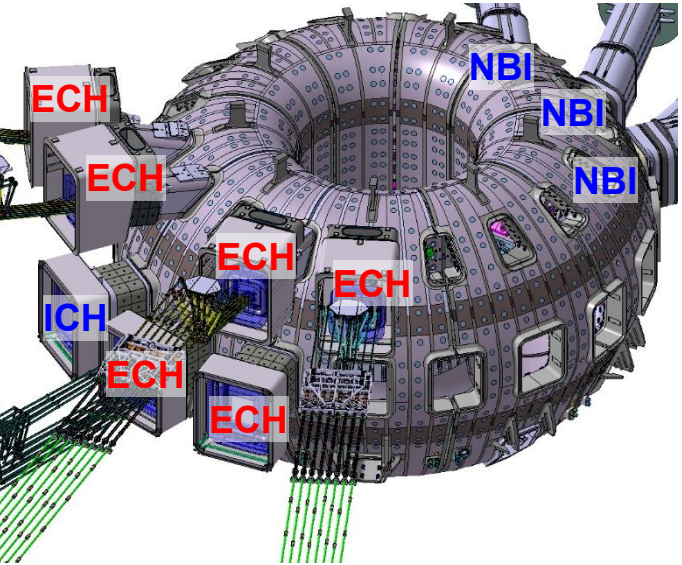
IMAS simulators being developed/adapted

- JINTRAC with coupling to DINA and H&CD [P. Fox]
- ETS-PAF [D. Coster] and BRIDGE [F. Poli]
- ITER's PDS with NTM control [M. Schneider]

Application use-case: H&CD workflow

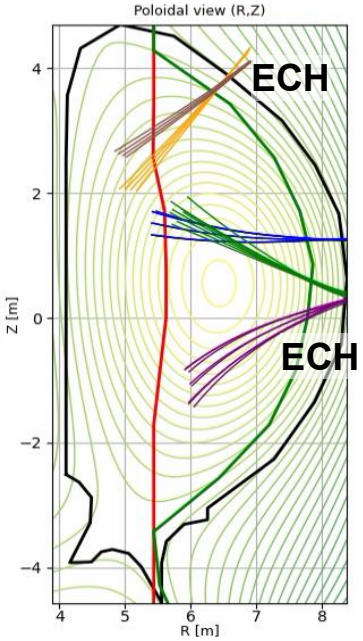
Heating & Current Drive workflow (H&CD): target

- Predicting the impacts of H&CD sources on plasmas performance with various heating mixes



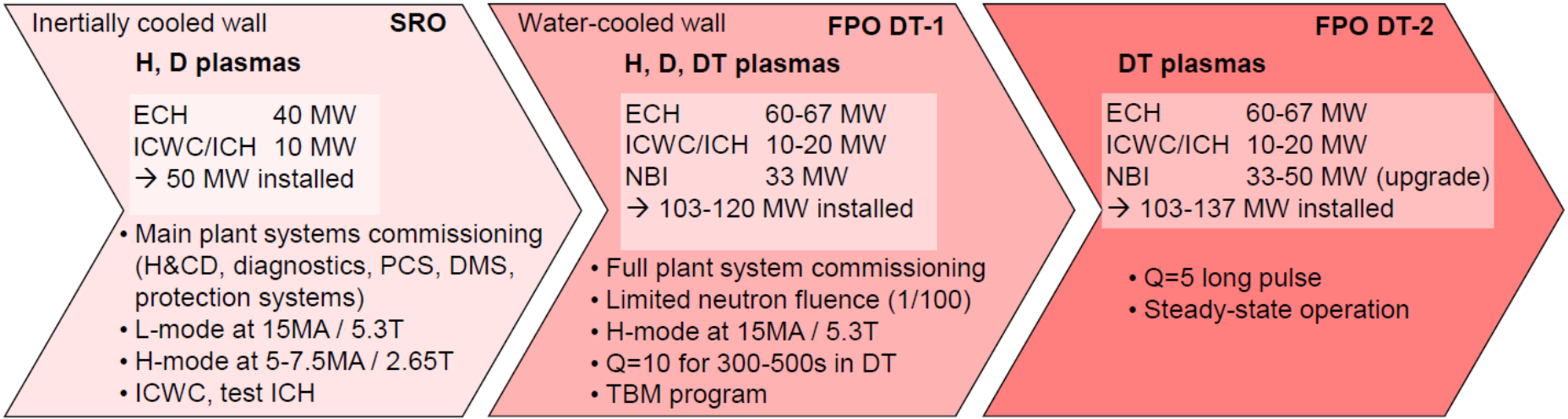
- Three H&CD sources:
  - Electron Cyclotron Heating (ECH)
  - Ion Cyclotron Heating (ICH)
  - Neutral Beam Injection (NBI)
- Synergistic effect may take place e.g. by heating ions with different systems (e.g. NBI + ICH)
- Such effects are captured by the H&CD workflow

[M. Schneider, ITPA-IOS, June 2025]



H&CD workflow: target (cont.)

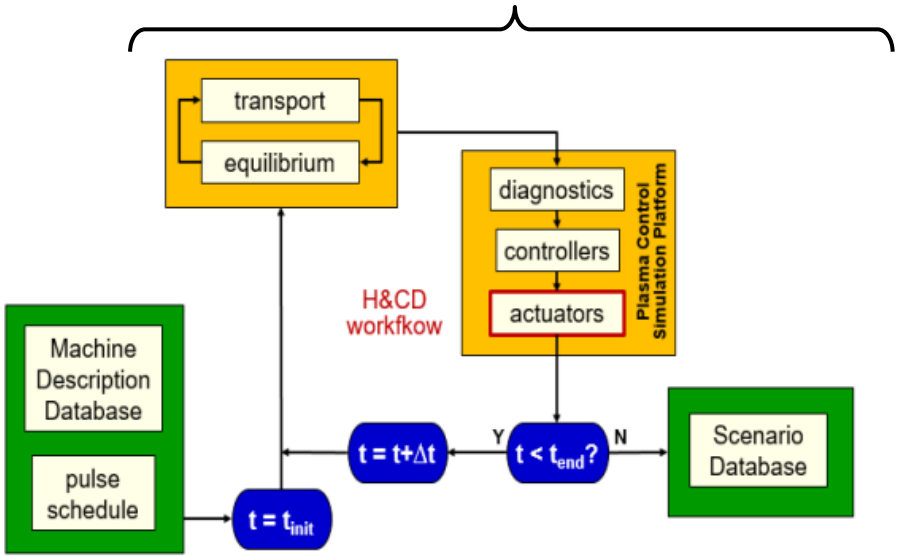
- More H&CD power with the new baseline, with dominant ECH
- Evolutive/modular design to follow the evolution of the heating systems and the possible upgrades at different phases of ITER operation



[M. Schneider, ITPA-IOS, June 2025]

H&CD workflow: design principia

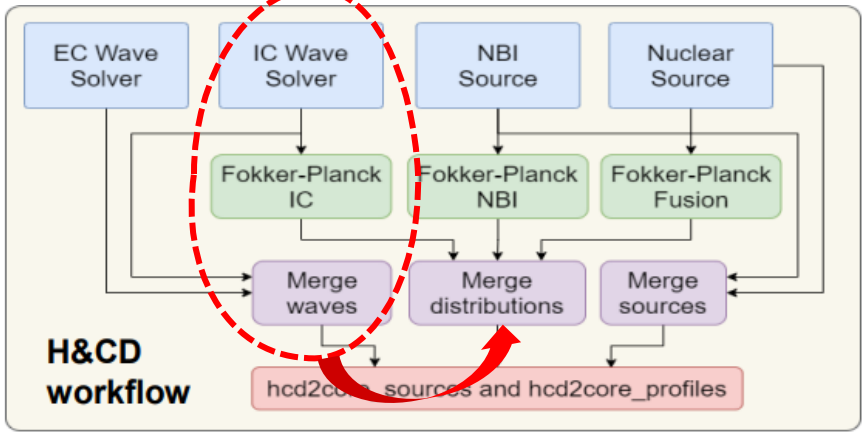
- Swap codes with common interface (IMAS)
- A flexible algorithm, can model synergies
- Integrated in high-fidelity plasma simulator



[M. Schneider et al 2021 Nucl. Fusion 61]

	ECRH	ICRH	NBI	Nuclear reactions
Wave or particle source	GENRAY GRAY GRAYSCALE TORBEAM TORAY-FOM	CYRANO LION PION TOMCAT TORIC	BBNBI NEMO RABBIT	AFSI SPOT (α)
Fokker-Planck	RELAX	FOPLA PION ASCOT SPOT	ASCOT SPOT RISK RABBIT	ASCOT SPOT

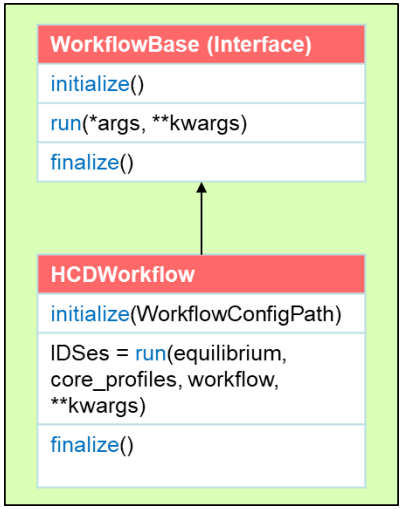
[M. Schneider, IIS2023]



H&CD workflow: implementation

- Follow the component-based approach
  - Initial coupling done in Java using Kepler was difficult to extend and maintain
  - Use Python *actors* (wrappers around F90/C++ physics routines)
  - Assemble the workflow by *coding* it directly in Python (easy to add glue code)
  - Give *workflows* a similar API to *actor components* (encapsulation)

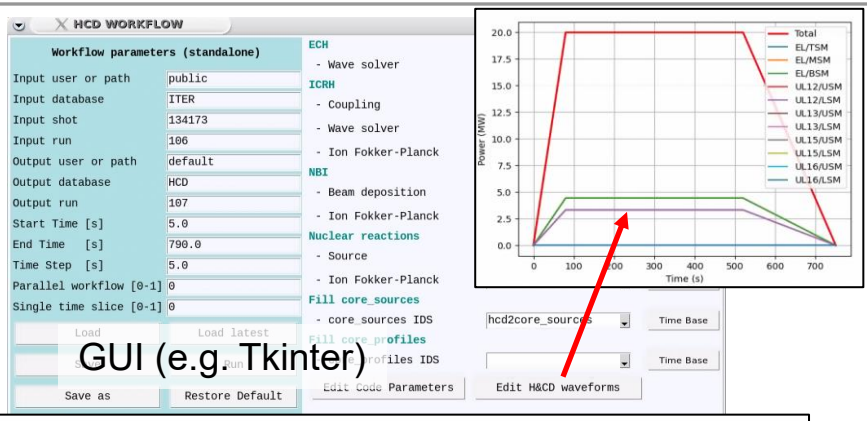
```python
class actor_class_name:
    def get_runtime_settings(self) -> RuntimeSettings:
        ...
    def get_code_parameters(self) -> CodeParameters:
        ...

    def initialize(self, runtime_settings: RuntimeSettings = None,
                   code_parameters: CodeParameters = None) -> None:
        ..
    def run(self, *args)->None:
        ...
    def __call__(self, *args):
        return self.run( *args )
    def finalize(self) -> None:
        ...

    def get_state(self) -> str:
        ...
    def set_state(self, state: str) -> None:
        ...
    def get_timestamp(self) -> float:
        ...
```



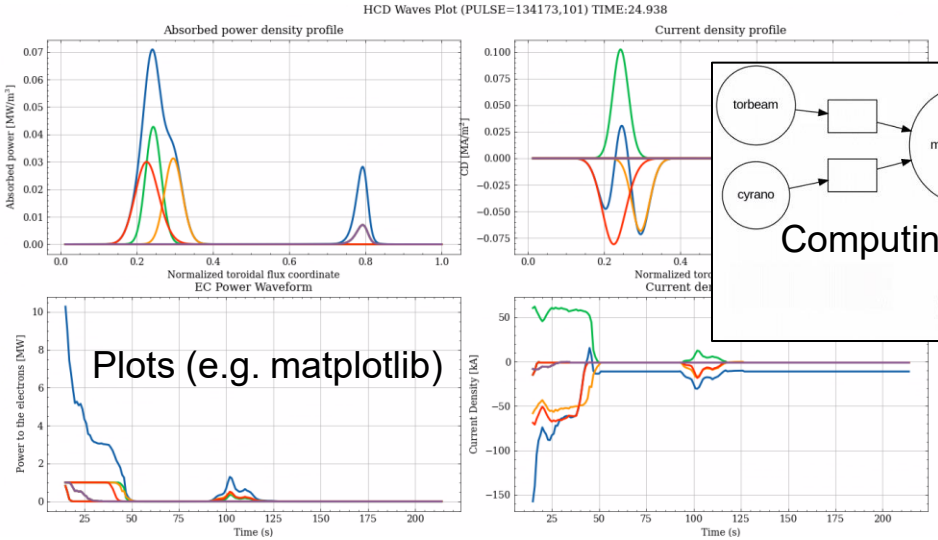
[P. Sawantdesai]

H&CD workflow: discussion

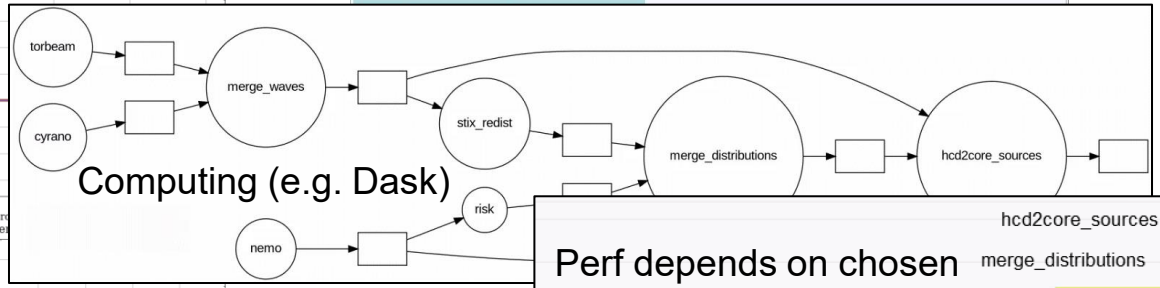
- Leveraging the Python eco-system ☺
- Procedural description (vs declarative) 😐
- Not best fit for stateful components ☹



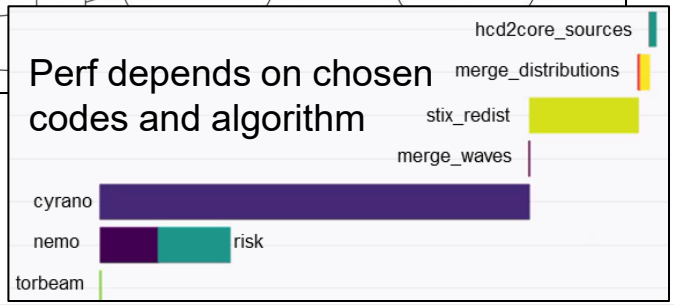
GUI (e.g. Tkinter)



Plots (e.g. matplotlib)



Computing (e.g. Dask)

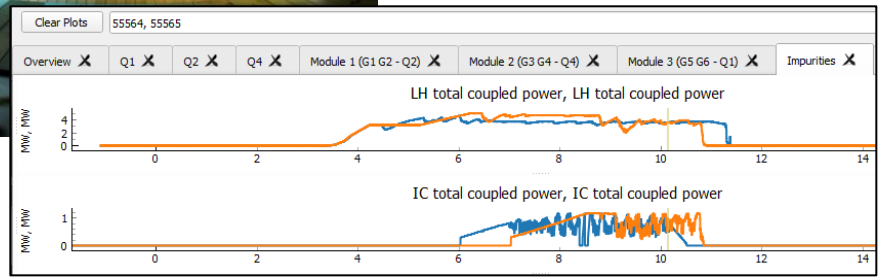


Application use-case: WEST PRC

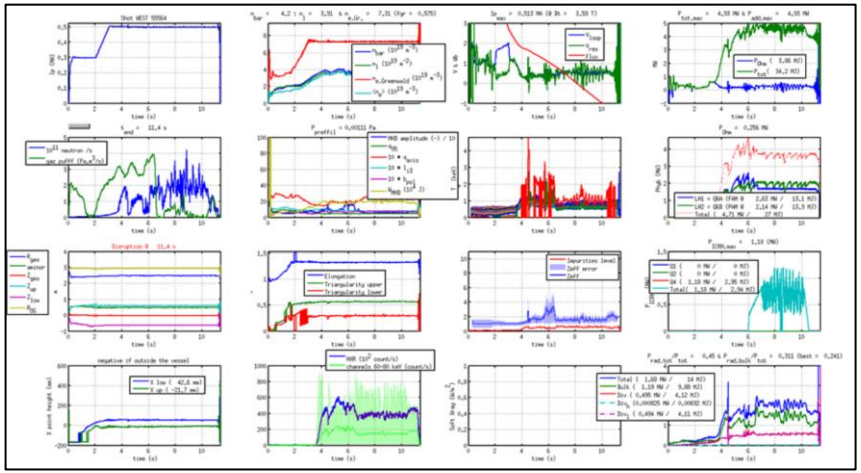
WEST Plasma Reconstruction Chain (PRC): target

Calculate equilibrium and plasma profiles diagnostic signals of WEST tokamak

- Used in the control room
  - During the pulse, relying on real-time data processing
  - Between pulses (post-treatment chain), requires fast processing



[J. Hillairet]

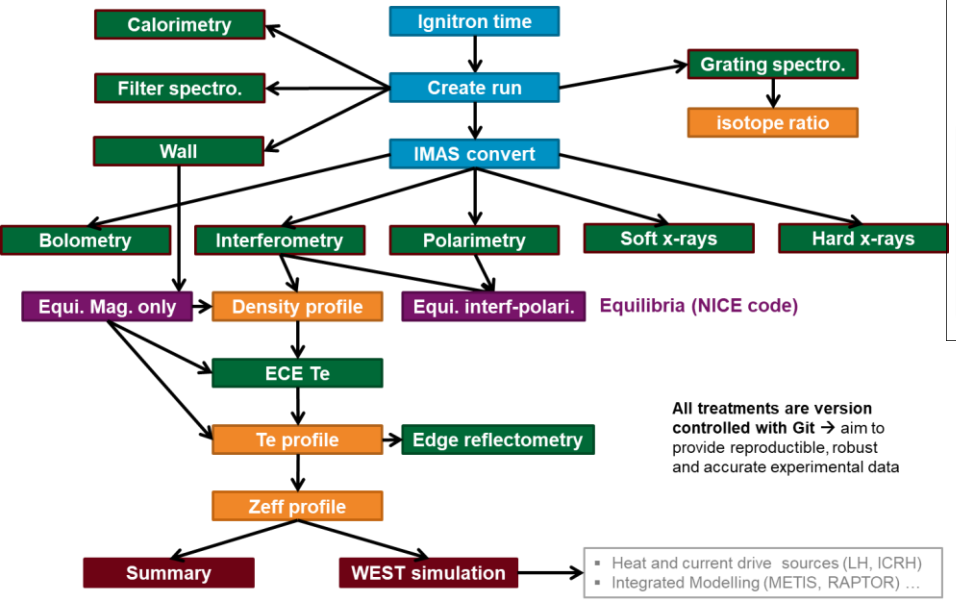


[JF. Artaud]

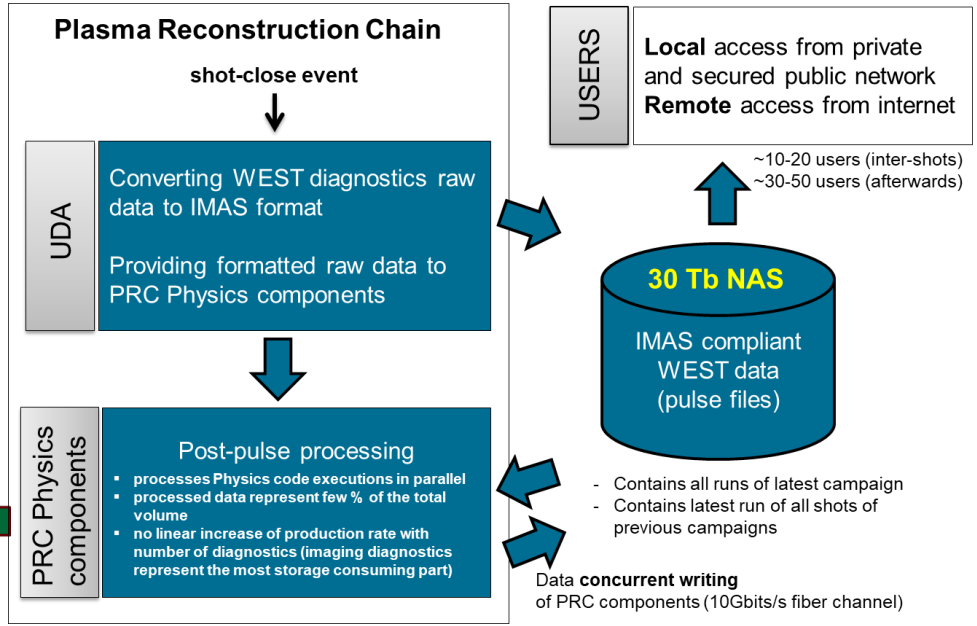
- And outside the control room
  - In-depth data analysis relying on heavier simulation codes and modelling

WEST PRC in IMAS: design

- Adaptation to IMAS since 2016
- Triggered at the end of the discharge (last magnetics signal)



All treatments are version controlled with Git → aim to provide reproductible, robust and accurate experimental data



- Calculations from different diagnostics, e.g. equilibrium from magnetics only or from interfero-polarimer

[L. Fleury et al, SOFT 2020]

WEST PRC in IMAS: output data

- Produce ~25 different IDSs
  - For analysis and modelling
- Some have several *occurrences*
  - Real-time and post-pulse calculations
  - Different signals, combinations, codes

equilibrium	0	Equilibrium reconstruction with magnetics only (NICE more info.)
	1	Equilibrium reconstruction with magnetics and interfero-polarimetry (NICE)
	2	Separatrix reconstruction in real time (VacTH)
	3	CHEASE equilibrium for WEST (more info.)

summary	0	SHOT SUMMARY, complete time signals
	1	SHOT SUMMARY, statistics (mean and standard deviations) on total power and toroidal current plateaus

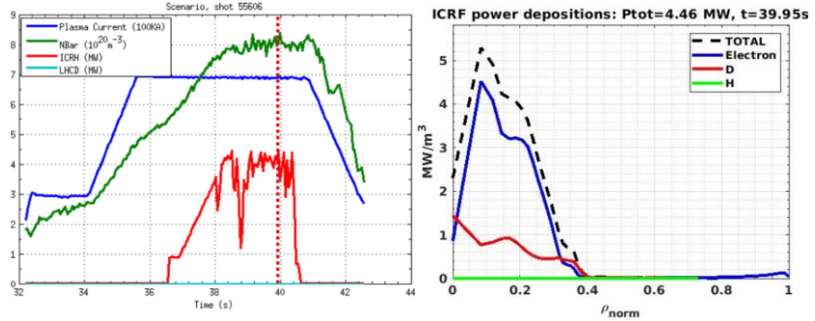
interferometer	0	Post-pulse line integrated electron density (more info.)
	1	Real Time line integrated electron density
	2	Synthetic reconstruction from NPROF_PARAM (for validation)

core_profiles	0	Fitted electron density and temperature (from occ 2,3)
	1	Fitted electron density profile from interferometry for equilibrium using splines (NICE)
	2	Fitted electron density profile from interferometry using hyperbolic tangent (NPROF_PARAM)
	3	Fitted electron temperature profile from ECE (TEPROF_ECE)
	4	Fitted electron temperature profile from X rays spectrometer, XICS (more info.)

WEST PRC in IMAS: data consumers

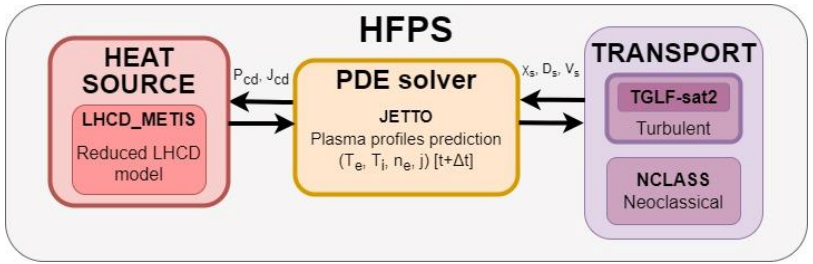
Modelling

- ICRH workflow to compute the power deposition profiles (CYRANO+StixRedist)



[P. Huynh]

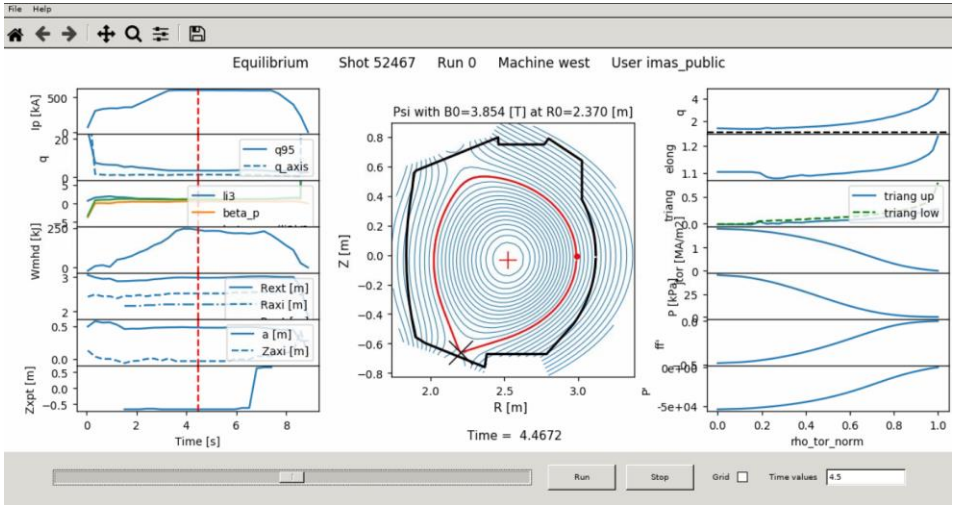
- IM with HCD and transport predictions



[T. Fonghetti]

Visualization

- IMAS_VIZ to browse signals
- Plugin for time-evolving equilibrium



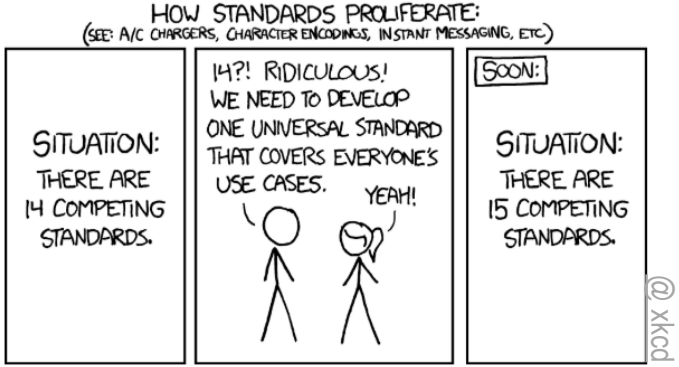
[J. Morales, L. Fleury]

Wrapping up

Take-home message

- Integrated modelling is attractive, but its realization is difficult
- Follow good software practices when developing models makes IM easier (a bit) → will benefit the individual models
- IMAS provides an environment & tools that help you get started → standard data model is the most essential part

Find a tradeoff between flexibility and standardization



https://github.com/iterorganization

- Open source is getting momentum in fusion → ride the wave!

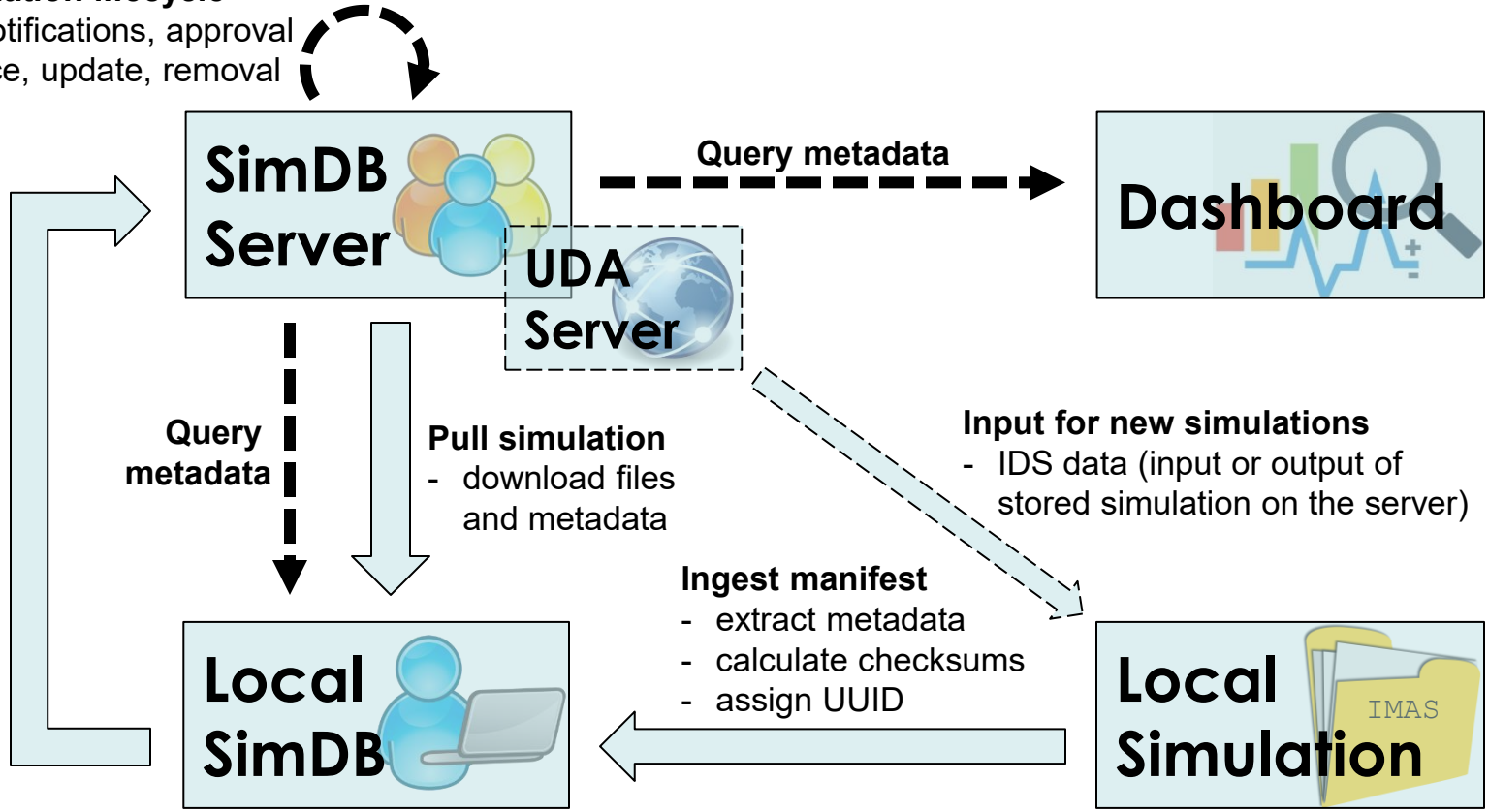
OSSFE 2026

→ For IMAS software not yet under open-access: https://jira.iter.org/servicedesk/customer/portal/1/create/257

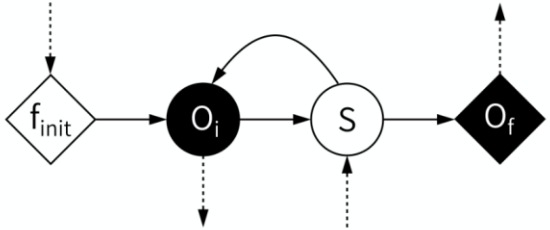
Extras

IMAS simulations management (cont.)

Manage simulation lifecycle
- watchers, notifications, approval
- obsolescence, update, removal

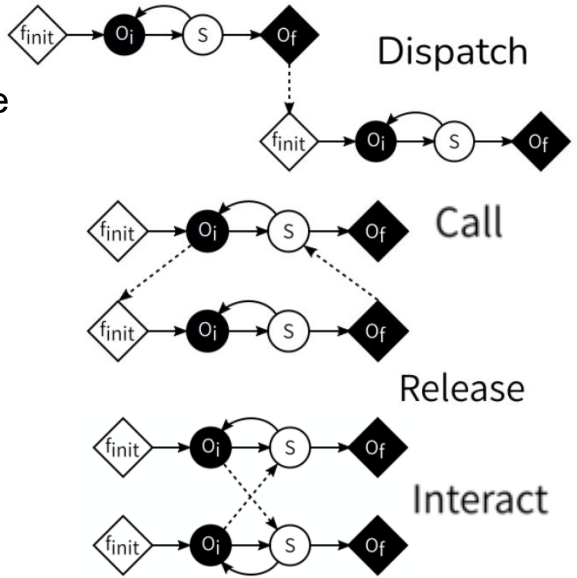


Diving more into MUSCLE3's foundations



$\mathbf{f_{init}}$	model initialization step
$\mathbf{O_i}$	intermediate state observation
S	state update (timestep)
$\mathbf{O_f}$	final state observation (end of time scale)

- MMSF theory
  - A model is described by a **Submodel Execution Loop**
  - And a scale separation map (time & space)
  - It defines a set of 3 possible **coupling templates** that are sufficient to describe all multiscale models



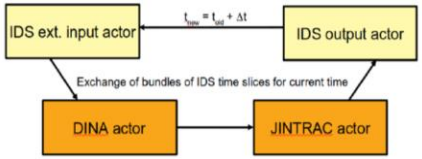
- MUSCLE3 library provide an API
  - That provide port operators f_{init}, O_i, S and O_f
  - To detect the end of a submodel, to deal with its multiplicity…

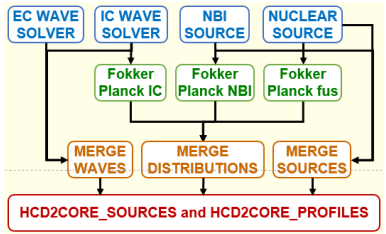
- More info: webinar by D.van Vugt and online course by L. Veen

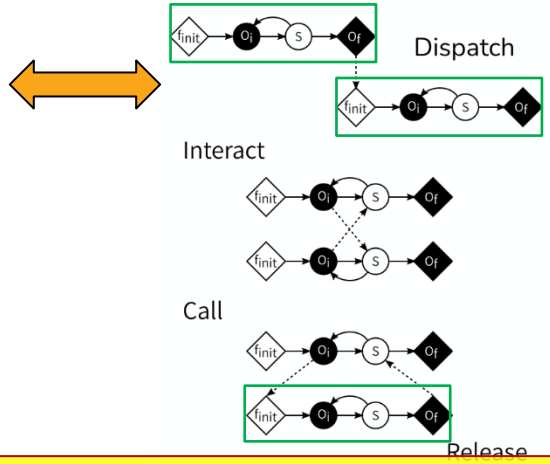
Comparing iWrap and MUSCLE3 actors

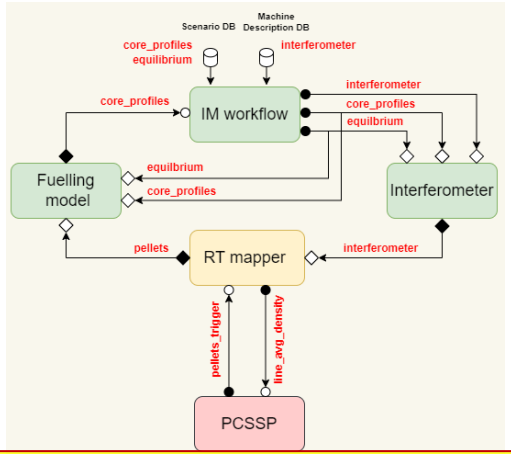
- **iWrap**
  - Generate standard actors in Python
  - Data flows as IDS objects centrally controlled by the Python script
  - No internal state remains between two calls (unless global)
  - Extended to generate MUSCLE3 actors

- **Persistent Actor Framework**
  - Implement actors in Fortran, C++ or Python
  - Data flows directly between connected actors (byte buffers over TCP/IP)
  - Actors are running in separate processes (keeping their internal state)
  - Some actors can be generated by iWrap





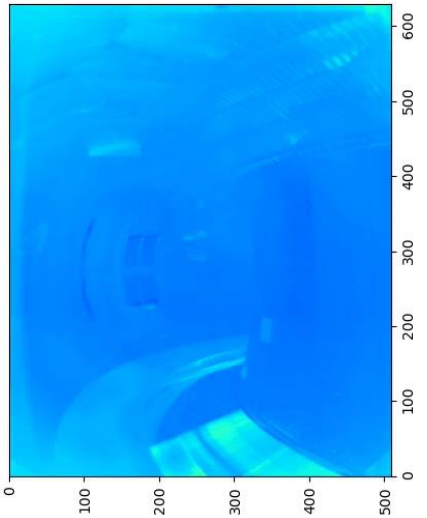




You can use the tools, combine them, invent your own or develop **your application** manually and still benefit from the IMAS interface

WEST PRC in IMAS: discussion

- Usage in control room
  - IMAS (Fortran) data access is too slow: due to IDS granularity (may be improved by partial IDS retrieval or lazy loading) and because of UDA mapping overhead
  - Real-time application are reading raw data directly from memory space
- Usage outside the control room
  - IMAS allow reading experimental data as well as synthetic data
  - Allow running external codes without adapting them specifically to WEST experimental data: CHEASE, HFPS (JETTO + H&CD), IDA, …
- Going further
  - Librir create/read compressed videos (16 bpp, lossless x5)
  - Adapted to IMAS via a dedicated *plugin* for the camera_ir IDS



[IR wide-angle camera shot #56922, V. Moncada, L. Fleury,]