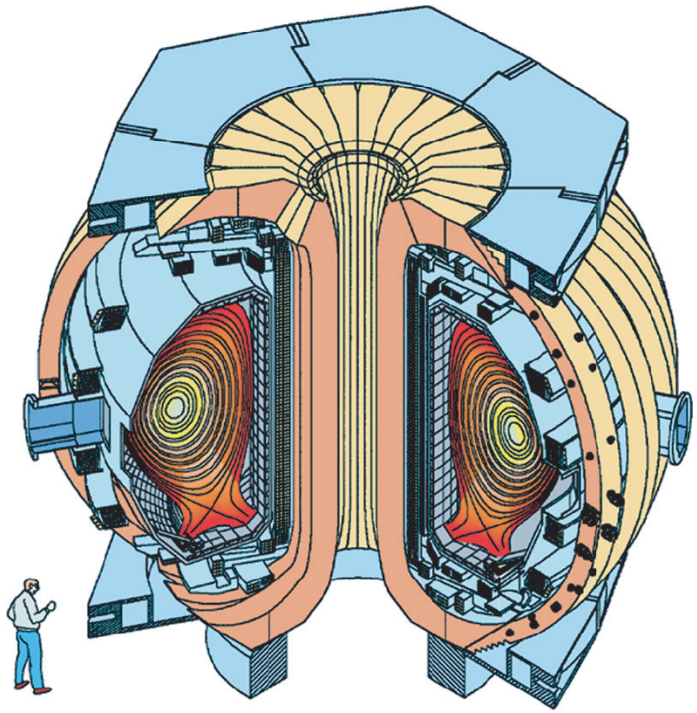
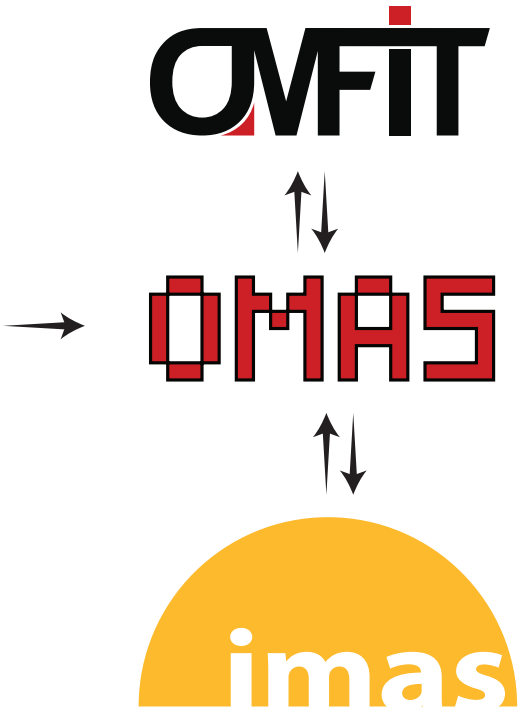
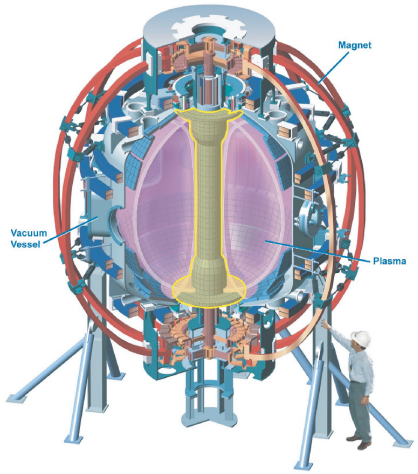
# Ushering the US integrated modeling community into the ITER IMAS era

**O. Meneghini**, D. Eldon, S.P. Smith, T. Slendebroek,
J. McClenaghan, B. C. Lyons, L. Lyons, K.E. Thome, J. Candy

APS-DPP
Thursday Nov. 11, 2021



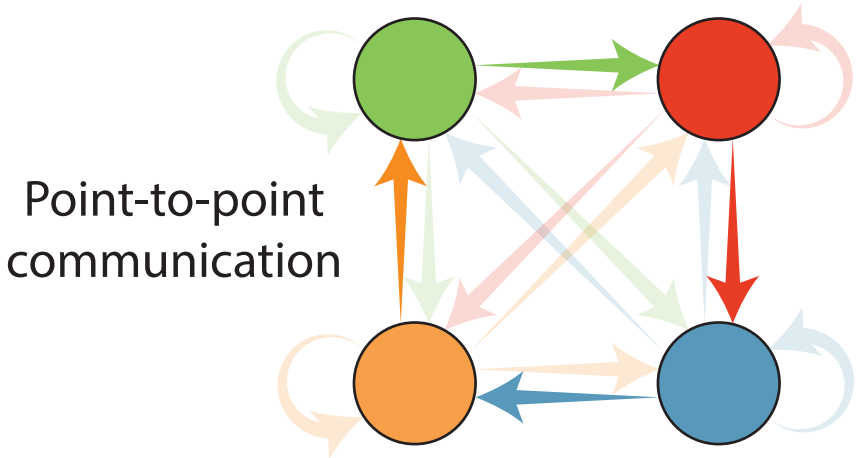


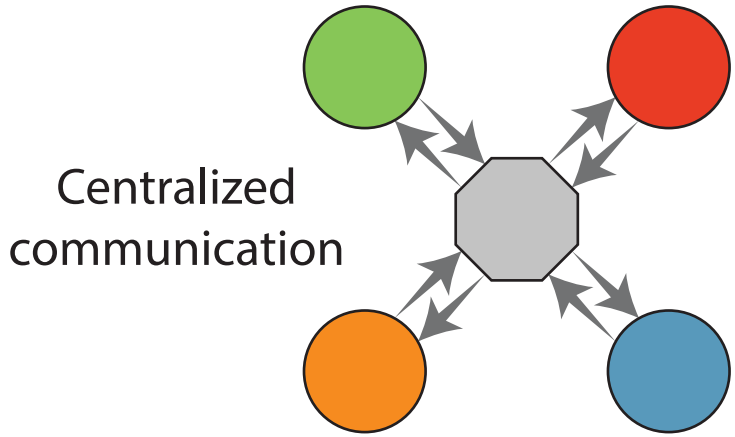




GENERAL ATOMICS

ATOM

# Integrated modeling can be greatly facilitated by wide-spread adoption of a standardized data format




**Free-for-all data formats:**

+ Does not require agreement with other parties
- Integration effort scales as $\mathcal{O}(N^2)$...
- ...or integration effort depends on workflow being executed

**Standardized data format:**

- Requires coordination among ALL parties
+ Integration effort scales as $\mathcal{O}(N)$
+ Support plug & play of new components and arbitrary workflows

# imas is fusion's best attempt to build such standard

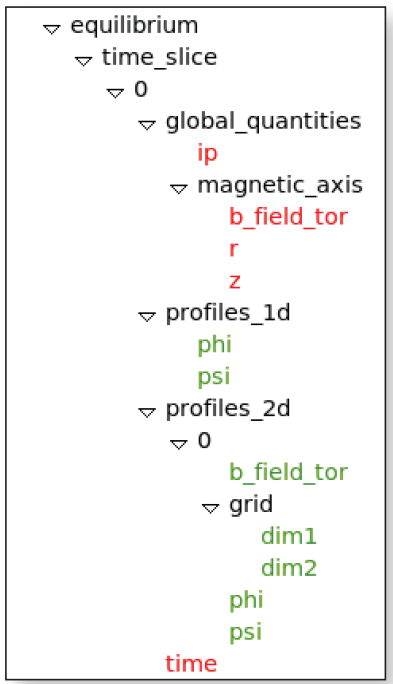
**ITER defined a standard for handling its data: IMAS**

- For both experimental and simulated data
- All ITER data will only be available in this format

**IMAS data is organized ~ 70 Interface Data Structures**

- Physics IDSs: (equilibrium, core_profiles,...)
- Engineering IDSs: (magnetics, thomson_scattering, ...)
- Each IDS is structured as a hierarchical tree

**Not ITER specific, is being adopted worldwide**



# OMAS – Ordered Multidimensional Array Structure

**Web:** `https://gafusion.github.io/omas`
**Pub:** O. Meneghini et al 2021 Nucl. Fusion 61 026006


**Developed under AToM to ease interface of Python codes with IDSs**

✓Open-source ✓Tested ✓Documented ✓Independent of OMFIT

1. Stores data compatibly with the IMAS standard
2. Offers convenient services/features beyond simple data storage
3. Trivial to install and use anywhere

```
> pip install omas
```

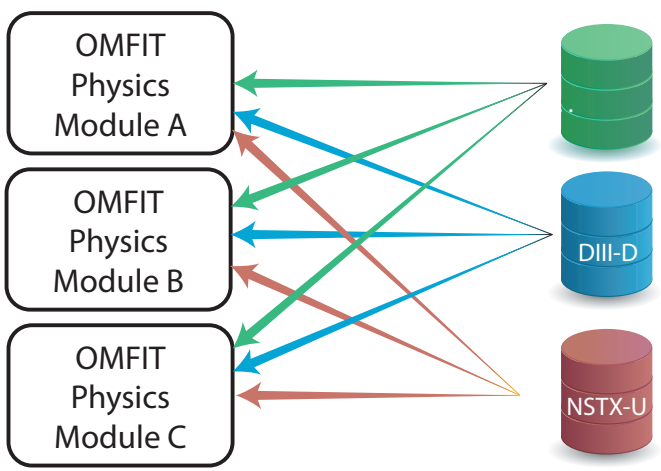
# Mapping experimental data to IDSs isi becoming a priority for both ITER and GA integrated modeling programs

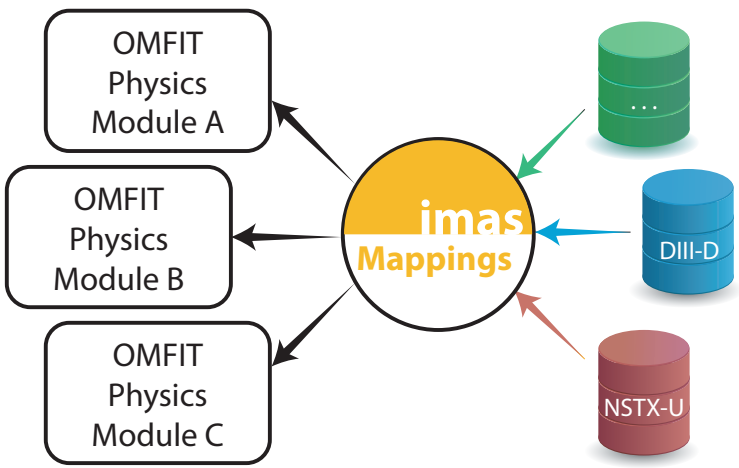
**ITER:** One area in particular stands out that we cannot do alone: mapping unprocessed experimental data into IDSs. The Members' existing research programs are essential to building and validating the ITER data processing pipelines

**T. Luce @ IMEG 2021**

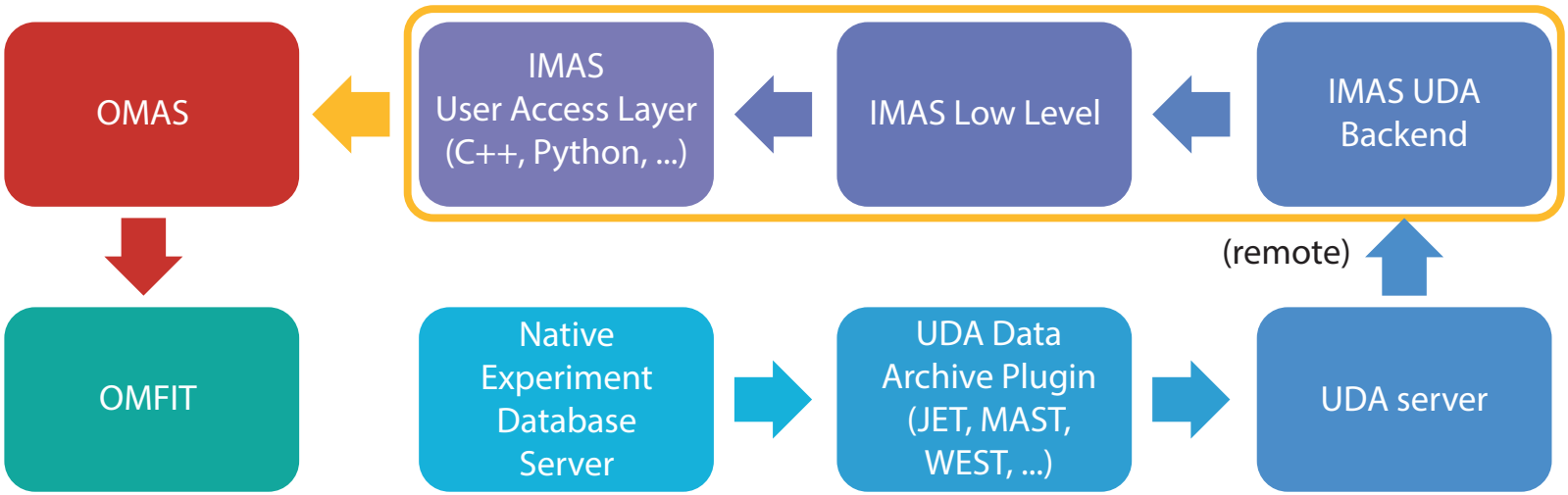
**GA:**

- Generalization of GA tools to machines other than DIII-D
- Offer DIII-D data in IMAS format for ITER, ITPA, and broader community
- Be ready to leverage ITER data



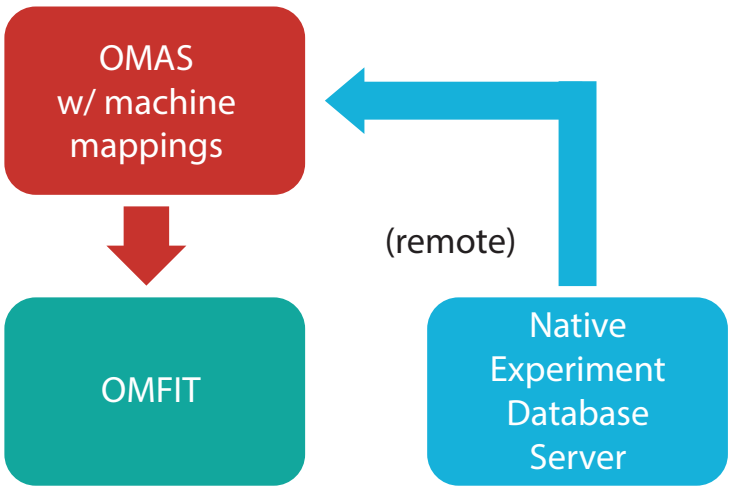



# IMAS has an infrastructure for on-the-fly mapping of experimental data to IDSs, via UDA server



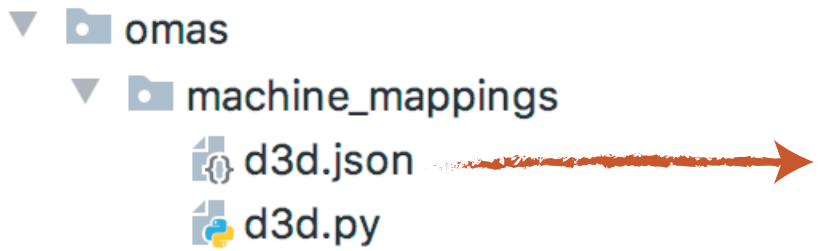

- Approach followed by EU tokamaks, KSTAR, HL-2A/2M
- UDA idea is to have a "smart server" doing the mapping
  - UDA plugins know how to map data from native format to IMAS
- Undesirable features:
  - Additional UDA data server increases administrative cost
  - Tall software stack, adds complexity/latency

# What if machine mappings were done directly in OMAS?




- Idea here is to have "smart clients" and "same old dumb server"
- Directly connect to native database → **A lot simpler & faster**
  - Only native data server → no extra maintenance
  - No middle man → minimize latency
  - No need for local IMAS installation → minimize complexity
- Lever native server capabilities for remote access, credentials, parallel data fetching, server-side ops, ...

# OMAS machine mappings are defined in a Json file, and use either direct MDS+ TDI expressions or Python



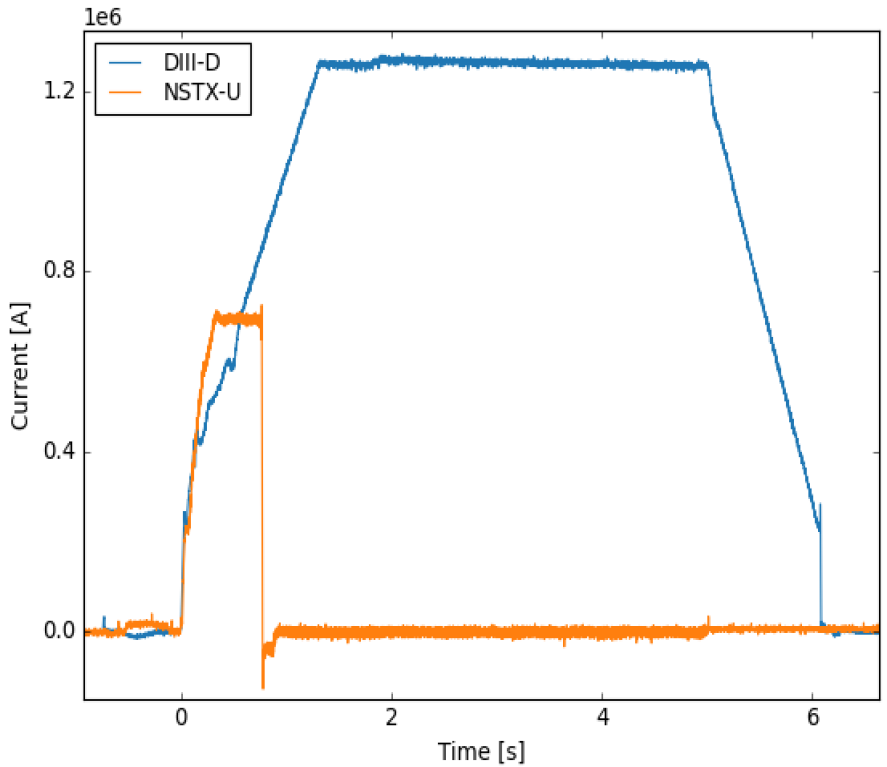

```
{
 "__mdsserver__": "atlas.gat.com:8000",
 "__options__": {
  "EFIT_tree": "EFIT01",
 },
 "dataset_description.data_entry.pulse_type": {
  "VALUE": "pulse"
 },
 "equilibrium.time_slice.:.global_quantities.ip": {
  "TDI": "data(\\{EFIT_tree}::TOP.RESULTS.GEQDSK.CPASMA)",
  "treename": "{EFIT_tree}",
  "COCOSIO": 1
 },
 "pf_active.coil.:.current.data": {
  "COCOSIO": 11,
  "PYTHON": "pf_active_coil_current_data_d3d(ods, {pulse}
}},
```

```
def pf_active_coil_current_data_d3d(ods, pulse):
    """
    Load DIII-D poloidal field coil currents

    :param ods: ODS instance

    :param pulse: int
    """
    with omas_environment(ods, cocosio=1):
        time = mdsvalue('d3d', 'D3D', pulse, f'ptdata2("PCF1A",{pulse})').dim_of(0)
        for k in range(18):
            fcid = 'F{}{}'.format((k % 9) + 1, 'AB'[int(k // 9)])
            ods[f'pf_active.coil[{k}].current.time'] = time
            ods[f'pf_active.coil[{k}].current.data'] = mdsvalue(machine='d3d', treename='D3D', pulse=pulse,
                                                                TDI=f'ptdata2("PC{fcid}",{pulse})').data()
```

# OMAS machine data mappings work behind the scenes to help users get their data seamlessly (1/3)

- **IDS abstraction** allows getting data from different machines independently of how their data is stored in native databases
- **Lazy loading**: just access ODS to trigger database retrieval
  No syntactical difference of accessing data in memory or in DB

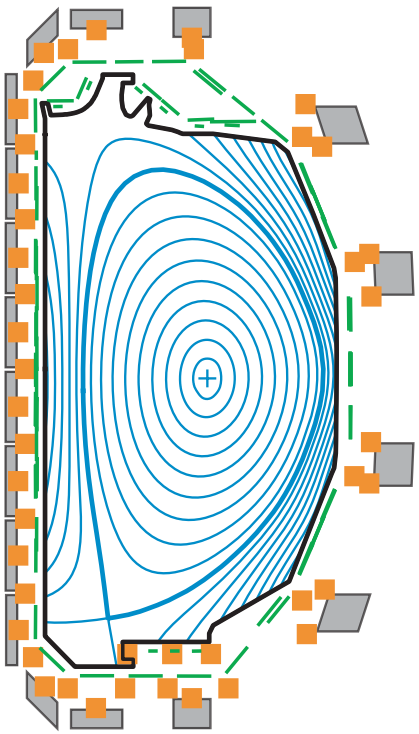
```
# DIII-D data
ods_d3d = ODS()
with ods_d3d.open('d3d', 168830):
    plot(ods_d3d['magnetics.ip[0].time'],
         ods_d3d['magnetics.ip[0].data'],
         label='DIII-D')

# NSTX-U data
ods_nstxu = ODS()
with ods_nstxu.open('nstxu', 204202):
    plot(ods_nstxu['magnetics.ip[0].time'],
         ods_nstxu['magnetics.ip[0].data'],
         label='NSTX-U')
```



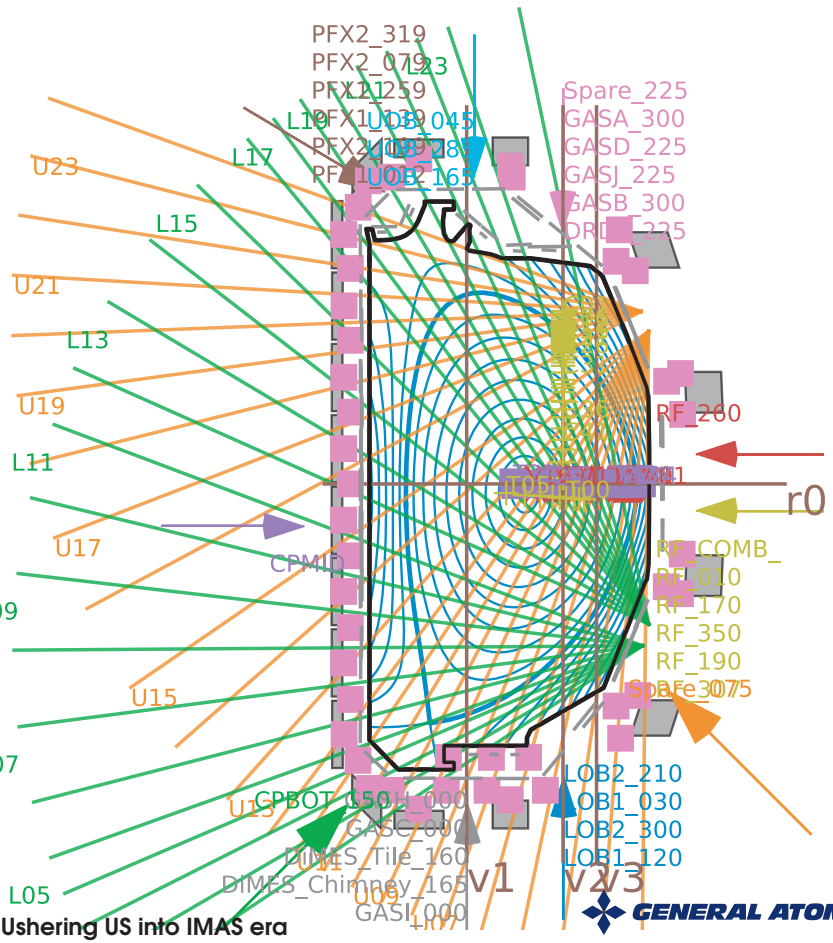
# OMAS machine data mappings work behind the scenes to help users get their data seamlessly (2/3)

- Lazy loading brings in only minimum set of data, as needed

```
# eg. Plot experiment equilibrium reconstruction w/ magnetics and PF coils
ods = ODS()
with ods.open('machine', 'd3d', 168830, options={'EFIT_tree': 'EFIT02'}):
    ods.plot_equilibrium_CX(time=2.1)
    ods.plot_overlay(wall=True,
                     magnetics=True,
                     pf_active=True)
```



bolometer
charge_exchange
gas_injection
interferometer
langmuir_probes
magnetics
pf_active
thomson_scattering
wall



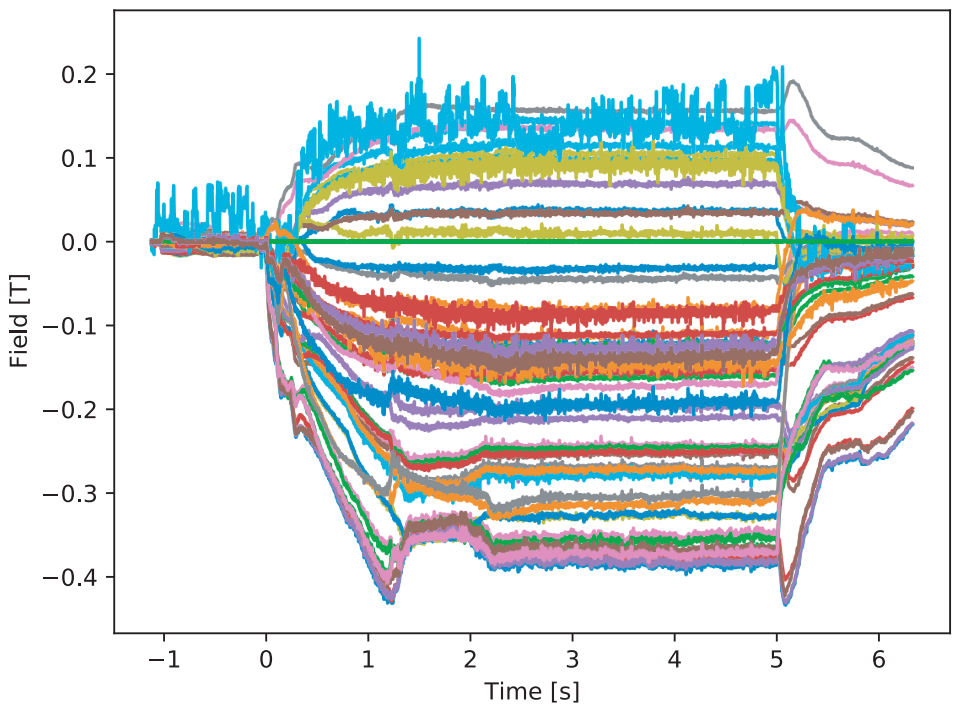
# OMAS machine data mappings work behind the scenes to help users get their data seamlessly (3/3)

- **Negligible overhead** over native MDS+ operations
- Supports modern MDS+ func. to request many signals at once

```
# eg. Plot magnetic probes
ods = ODS()
with ods.open('machine', 'd3d', 168830):
    plot(ods[f'magnetics.b_field_pol_probe.:.field.time'].T,
         ods[f'magnetics.b_field_pol_probe.:.field.data'].T)
    xlabel('Time [s]')
    ylabel('Field [T]')
```

~5 seconds to get in ODS
76 data + 1 time traces
of ~30k samples each

~7 seconds from my home
with SSH tunneling

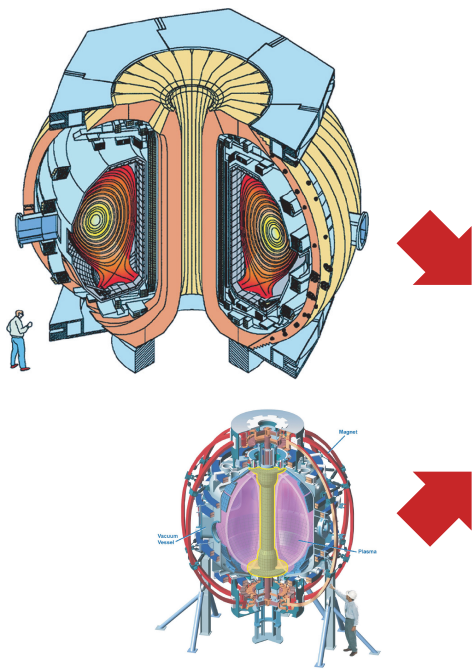


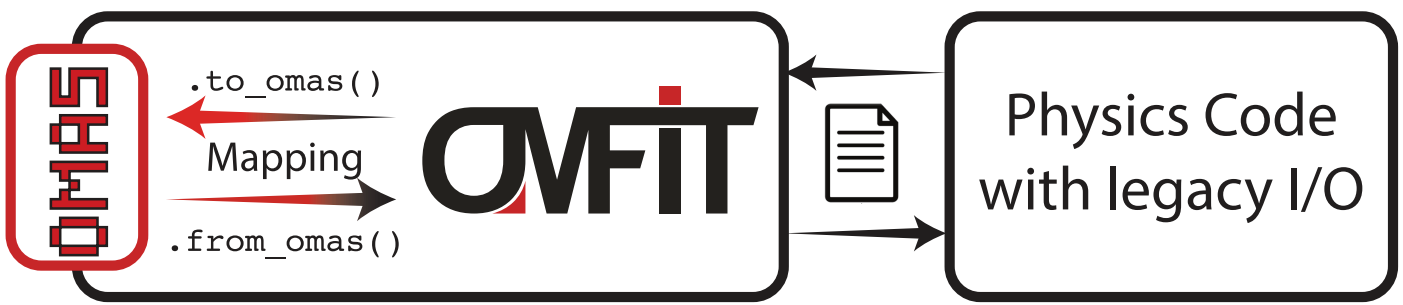
# We are leveraging OMAS machine mappings and OMFIT classes to generalize creation of kEQDSK EFIT input files

Three steps:

1. Dynamic map of experiments data to IDSs
2. Generate `equilibrium` IDS constraints from experimental IDSs
3. Generate EFIT kEQDSK input files from `equilibrium` IDS constraints

```
# Generate a kEQDSK file from experimental data
ods=ODS()
with ods.open('nstxu', 204202):
    kEQDSK = OMFITkeqdsk().from_omas(ods, time=0.369)
```

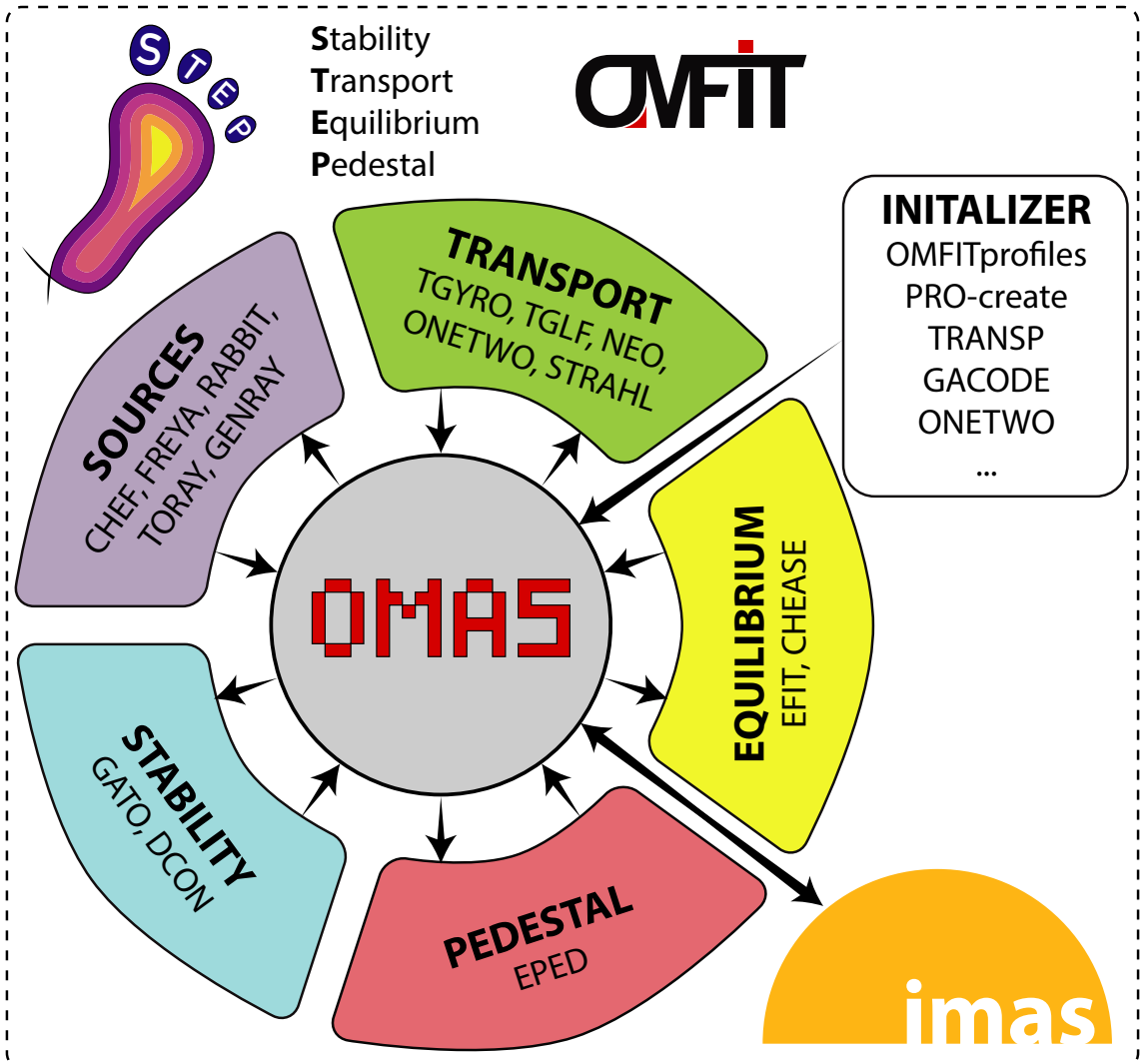




# OMFIT physics modules are rapidly being converted to use OMAS, and reap the benefits of centralized data approach

Eg. Centerpiece of STEP module to make self-consistent, theory-based predictions of tokamak plasmas

- Decouples data-flow from work-flow
- Plug-and-play different modules




**NI02.1 - B. Lyons:** Predicting performance and stability of tokamak plasmas using flexible, integrated modeling

**UO07.14 - T. Slendebroek** Elevating zero-dimensional predictions of tokamak plasmas to self-consistent theory-based simulations

**JO07.15 - D. Weisberg:** An integrated design study for the EXhaust and Confinement Integration Tokamak Experiment

# IMAS data dictionary is the fusion standard of the future, start using it today!

**OMAS facilitates adoption of IMAS IDSs for integrated fusion simulations**

- Trivial to install, does not require IMAS installation
- Easy to use, goes beyond simple data storage
- Open-source, mature, independent of OMFIT

**OMAS can dynamically map experiments data to IDSs:**

- Easy: No added software complexity or administrative burden
- Fast: Minimize latency by avoiding middle man

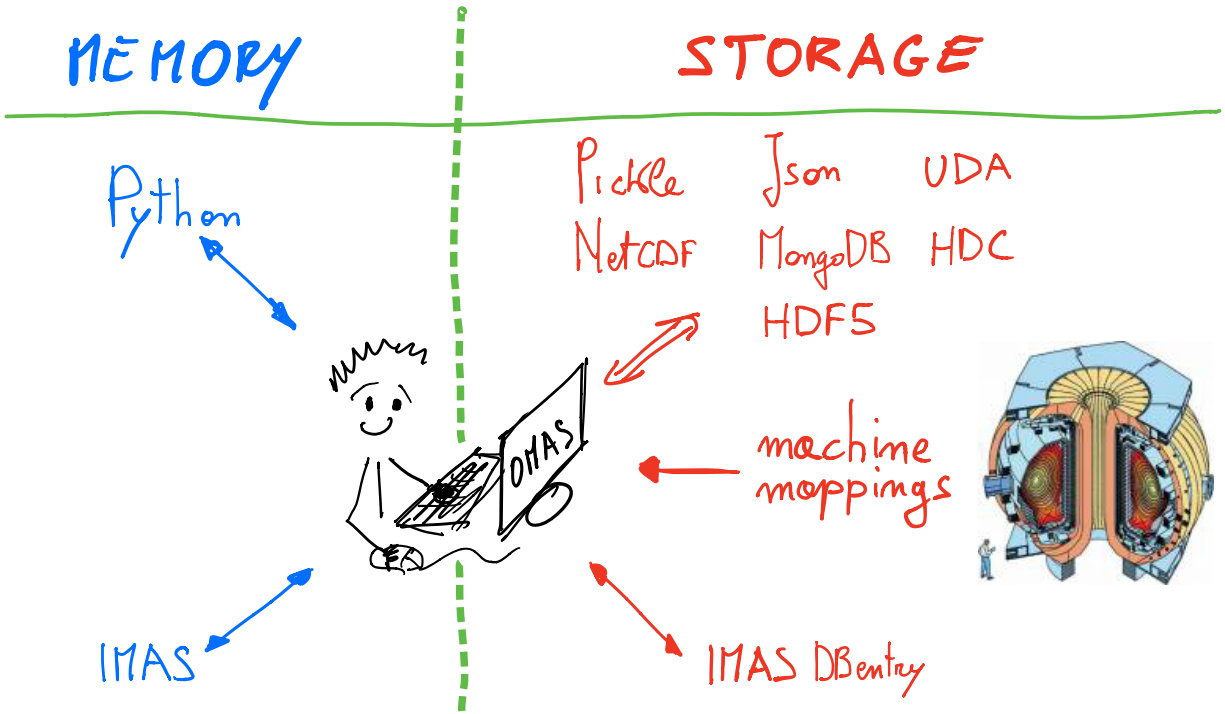
**OMFIT is rapidly being converted to lever OMAS**

- Generalization of experimental analysis modules
- Predictive simulations via OMFIT STEP module

# Extra slides

# OMAS is highly interoperable with IMAS

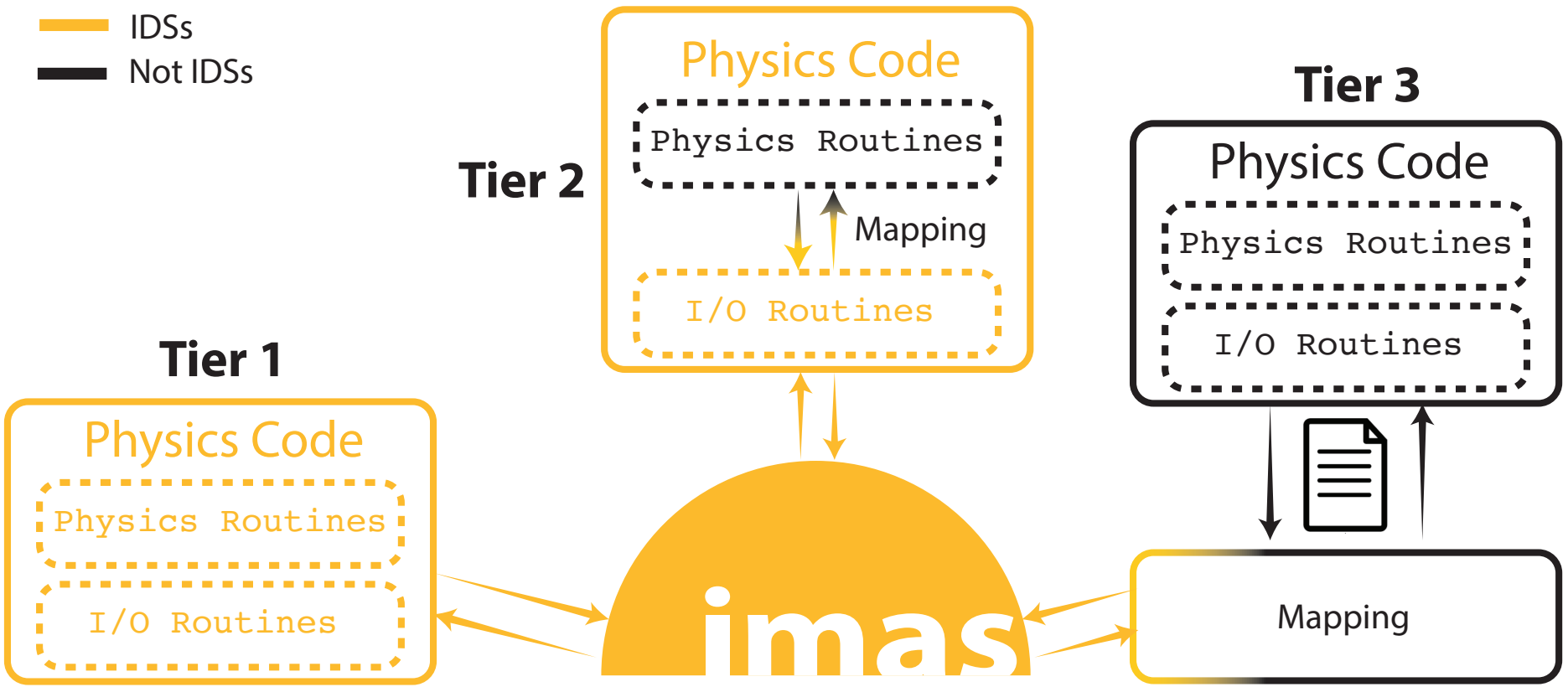
- Support for different memory backends, for different applications:
  - Pure Python: does not require IMAS installation
  - IMAS: allows seamless data transfer to IMAS Python actors



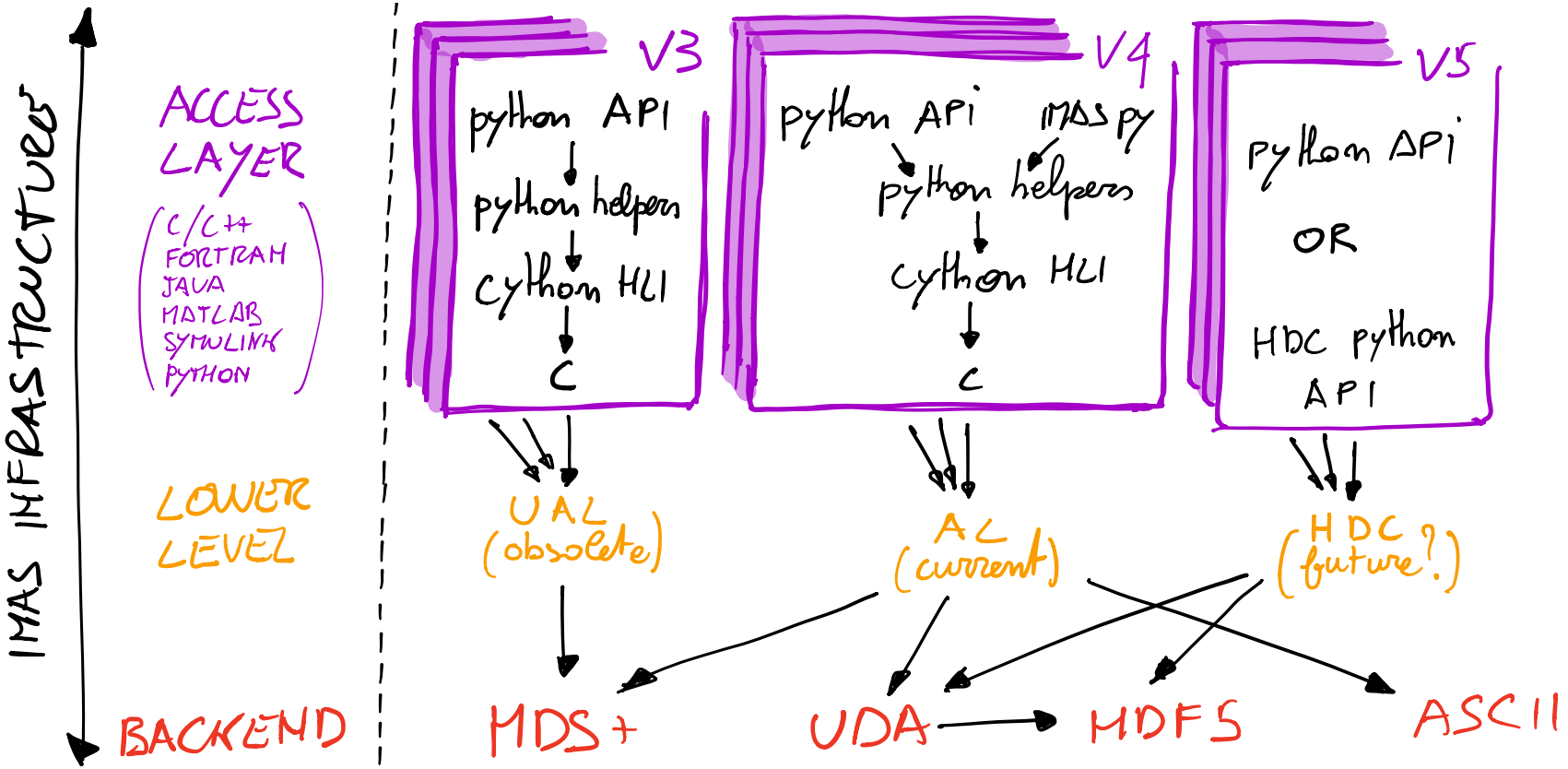

- OMAS can now mimic native IMAS Python API
  - Used for regression testing where IMAS is not installed (eg. GitHub CI)
  - Seamlessly adopt OMAS where IMAS API was previously used

# Adoption of IMAS can occur at different levels

1. Both physics routines and I/O "speak" IDSs
2. Mapping between IDSs and internal variables within physics codes
3. Mapping between IDSs and files outside of physics codes

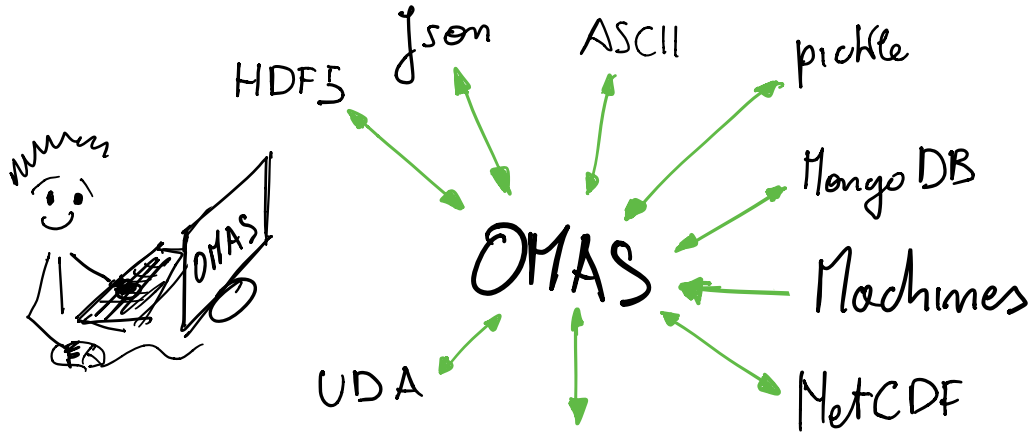


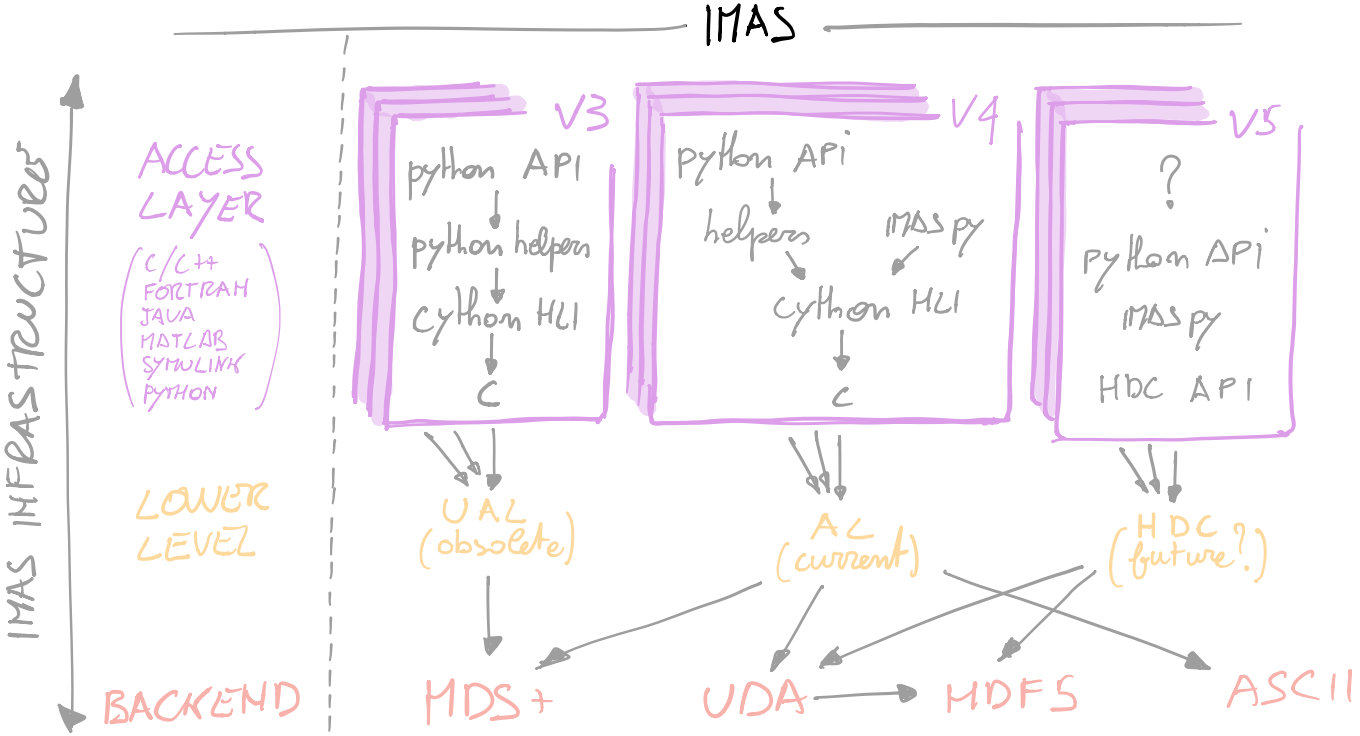
# The IMAS software stack is tall and evolving




- Hard to build on top of an infrastructure changing at its foundations
- IMAS infrastructures is heavy, hard to install, and difficult to manage

# In OMAS users can choose in what format to save their IDSs

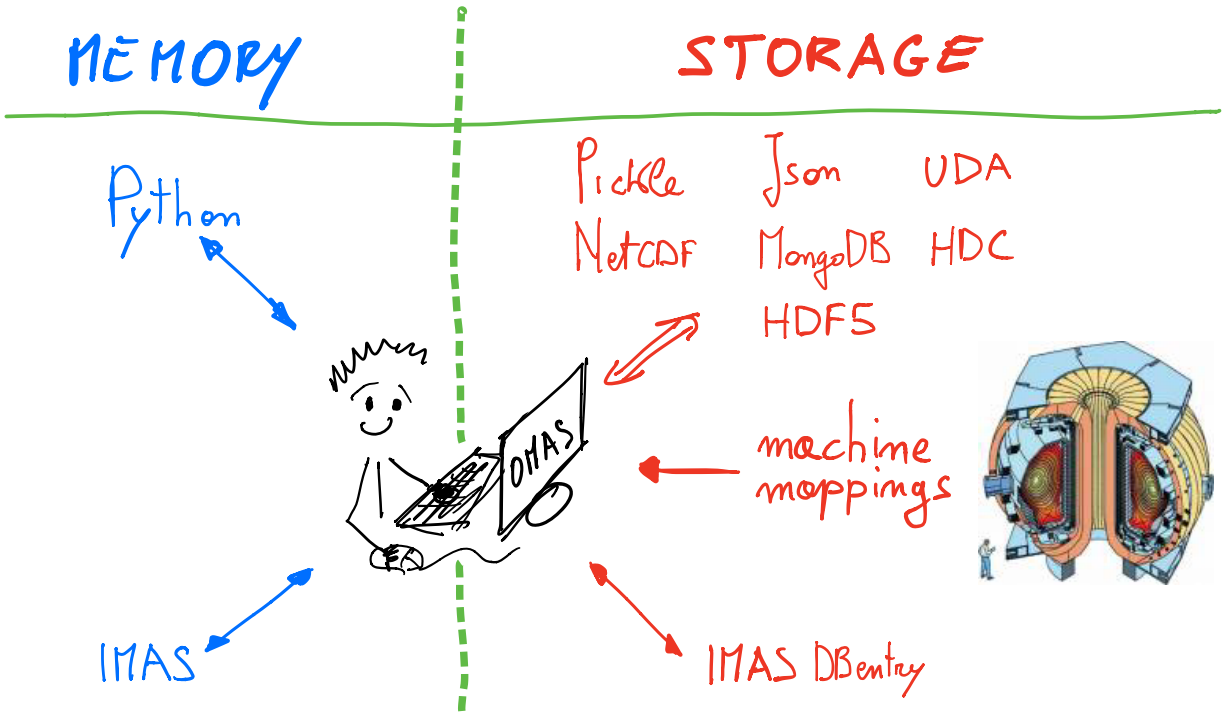



IMAS database storage is just one of the supported formats, and it is optional

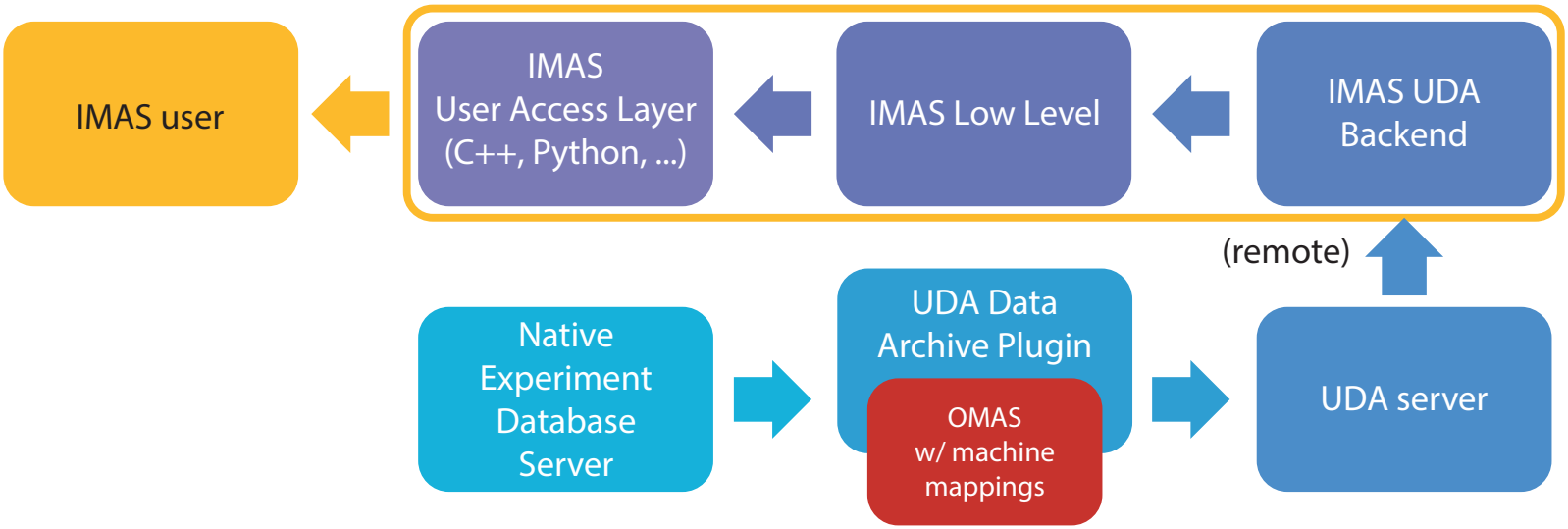
# OMAS is highly interoperable with IMAS

- Support for different memory backends, for different applications:
  - Pure Python: does not require IMAS installation
  - IMAS: allows seamless data transfer to IMAS Python actors



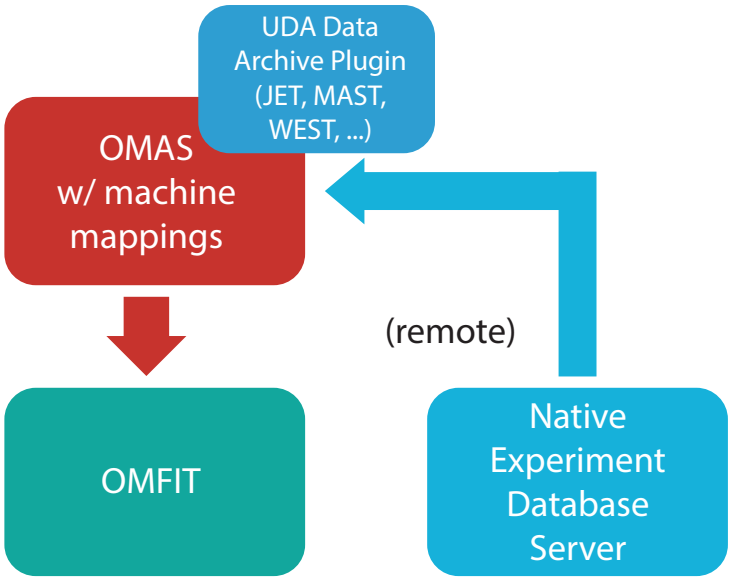

- OMAS can now mimic native IMAS Python API
  - Used for regression testing where IMAS is not installed (eg. GitHub CI)
  - Seamlessly adopt OMAS where IMAS API was previously used

# Interoperability with IMAS: OMAS w/ machine mappings could be used as UDA plugin




- UDA mapping plugin can be in any language, also Python

# Interoperability with IMAS:
# Existing UDA mappings could be used directly by OMAS




- Give OMAS direct access to data from JET, MAST, WEST, KSTAR, ...