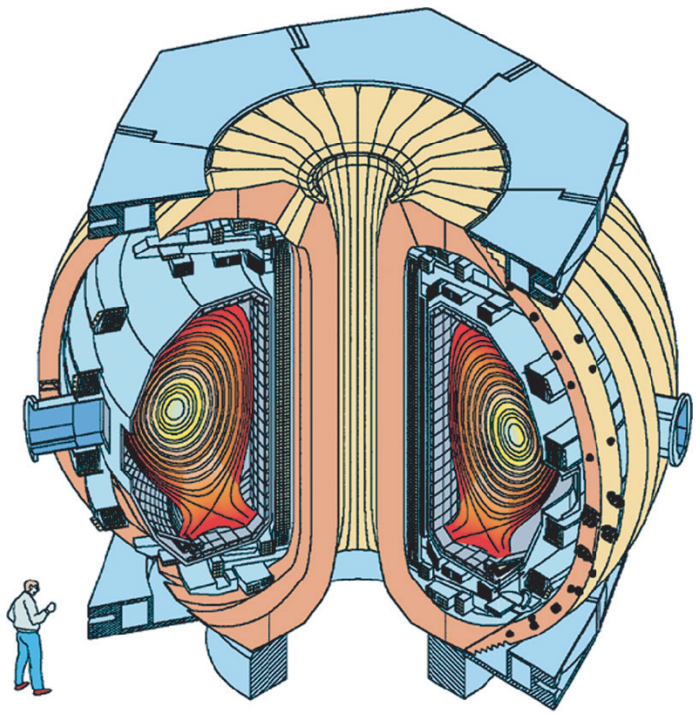
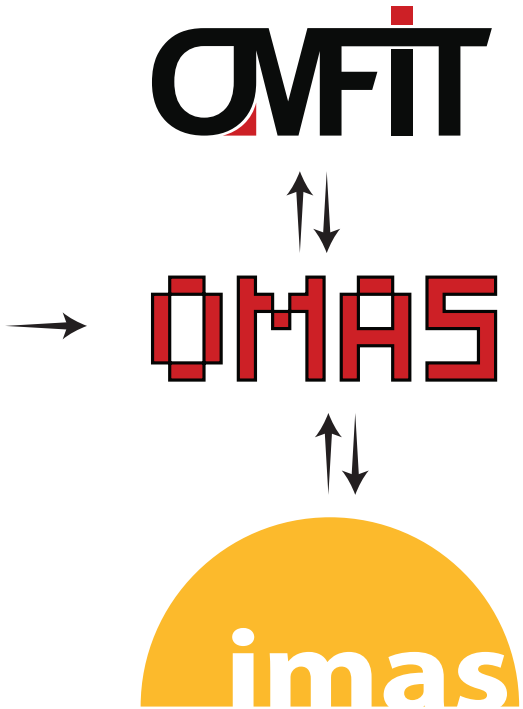
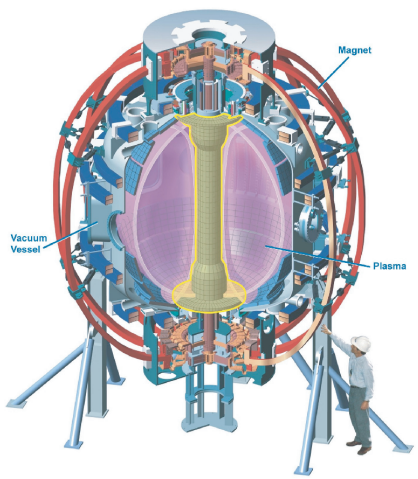
# Generalizing OMFIT DIII-D workflows to NSTX-U with the OMAS library

**O. Meneghini**,
G.Avdeeva, J. McClenaghan, S.P. Smith, K. E.Thome

NSTX-U Science Meeting – Monday June 21 2021





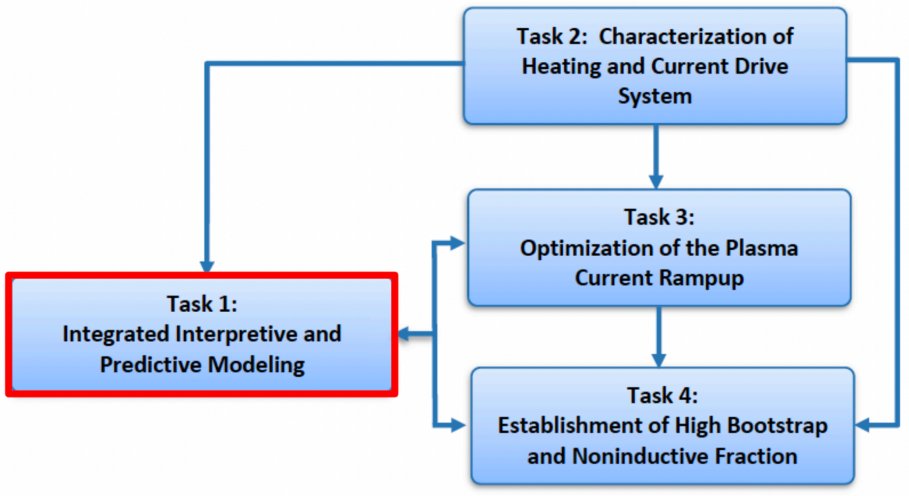




GENERAL ATOMICS

# GA is working with NSTX-U team to create stationary, noninductive scenarios with high $\beta$ and $f_{\mathrm{BS}}$ - K. Thome PI

**Task 1** will streamline workflows and provide support to enable robust, routine equilibrium, transport and EP analysis/predictions

- **Year 1:** Adapt OMFIT-based integrated modeling workflows used on DIII-D to NSTX-U
  - Kinetic equilibrium reconstructions
  - Power, particle, momentum balance analyses
  - Predictive scenario development
- **Year 2 - 5:** Implement and validate improved H&CD and transport models, and apply predictive capabilities




**We need an efficient way to generalize OMFIT experimental analyses modules that started as very DIII-D centric**

# imas standard is used to generalize our workflows

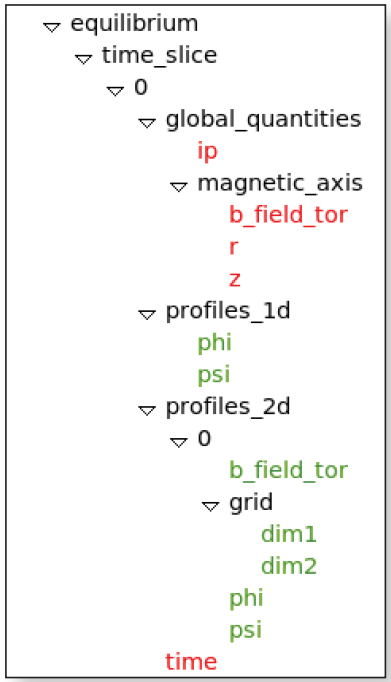
**ITER defined a standard for handling its data: IMAS**

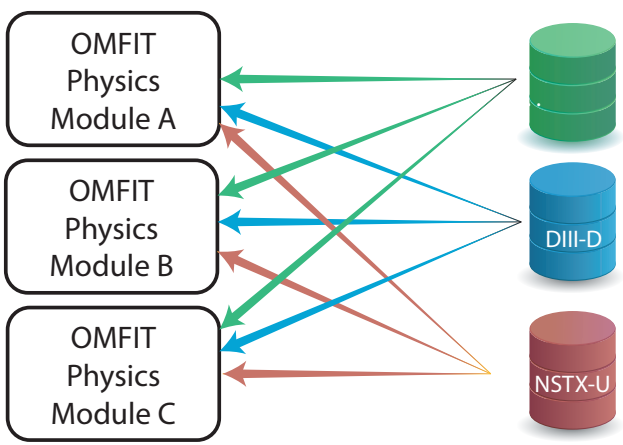
- For both experimental and simulated data
- All ITER data will only be available in this format

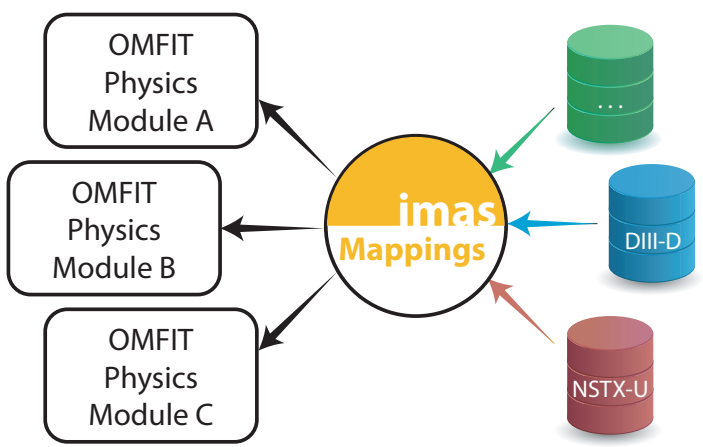
**IMAS data is organized ~ 70 Interface Data Structures**

- 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 library helps us interface with IMAS

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

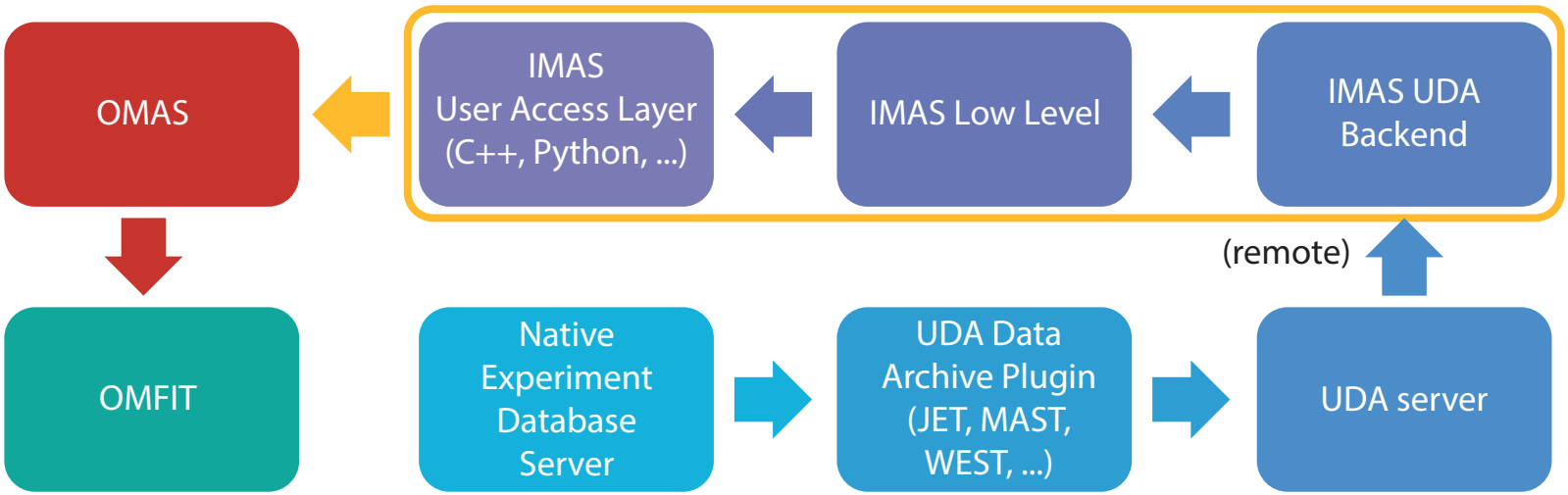

**Developed under AToM to ease interface of Python codes with IDSs**

✓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

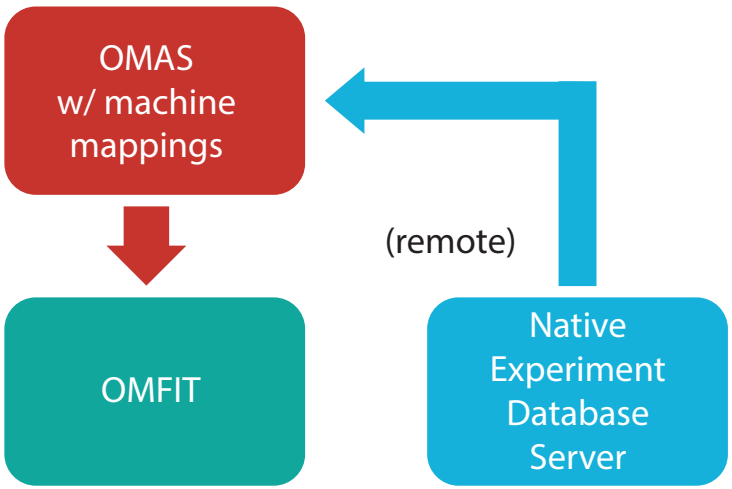
```
> pip install omas
```

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



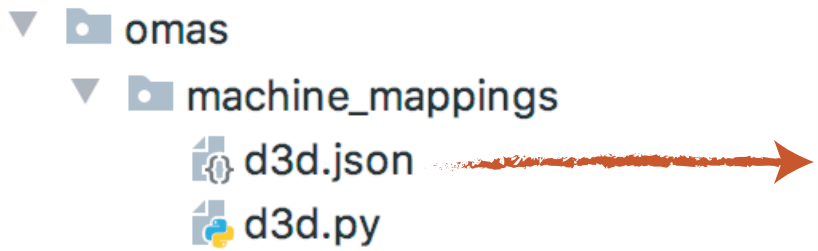

- Approach followed by EU tokamaks, KSTAR, HL-2A/2M
- UDA idea is to have a "smart server" doing the mapping
  - UDA plugins know how to map data from native format to IMAS
- Undesirable features:
  - Additional UDA data server increases administrative cost
  - Tall software stack, adds complexity/latency

# 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



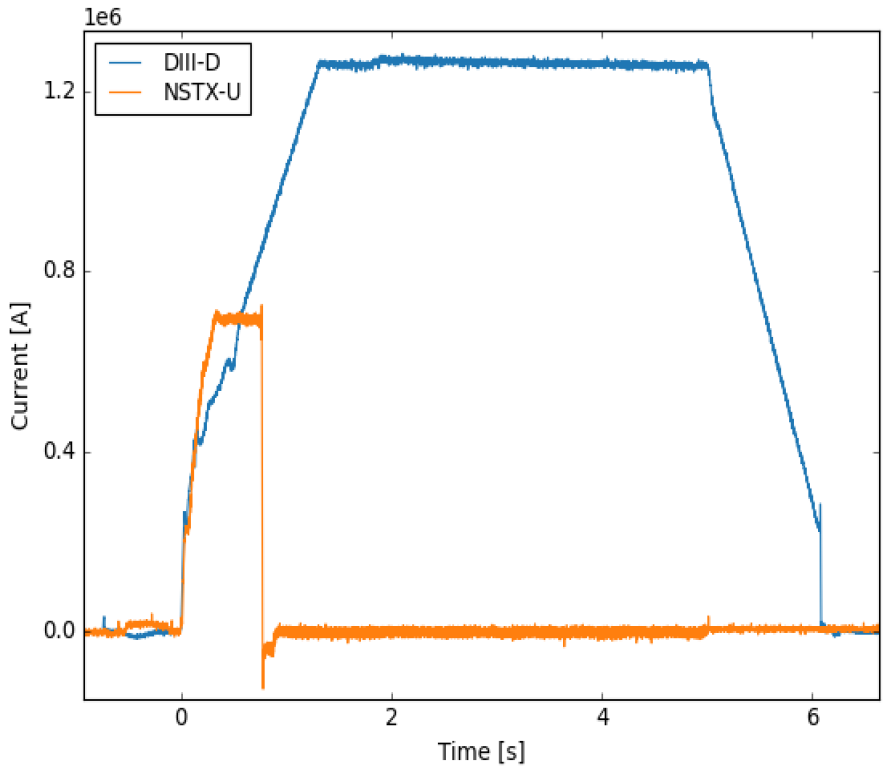

```json
{
 "__mdsserver__": "atlas.gat.com:8000",
 "__options__": {
  "EFIT_tree": "EFIT01",
 },
 "dataset_description.data_entry.pulse_type": {
  "VALUE": "pulse"
 },
 "equilibrium.time_slice.:.global_quantities.ip": {
  "TDI": "data(\\{EFIT_tree}::TOP.RESULTS.GEQDSK.CPASMA)",
  "treename": "{EFIT_tree}",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

```python
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

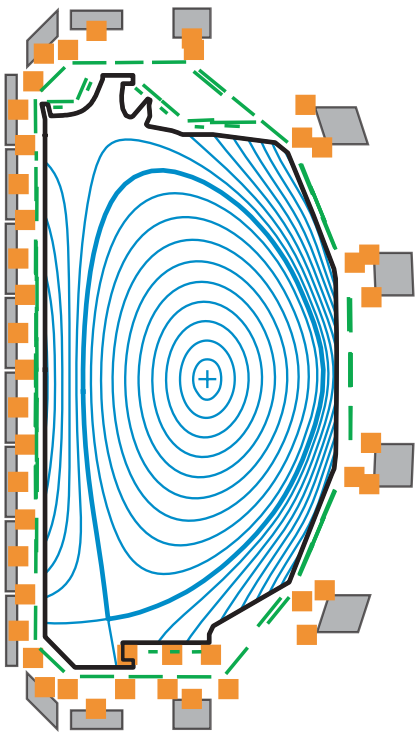
```
# DIII-D data
ods_d3d = ODS()
with ods_d3d.open('d3d', 168830):
    plot(ods_d3d['magnetics.ip[0].time'],
         ods_d3d['magnetics.ip[0].data'],
         label='DIII-D')

# 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')
```



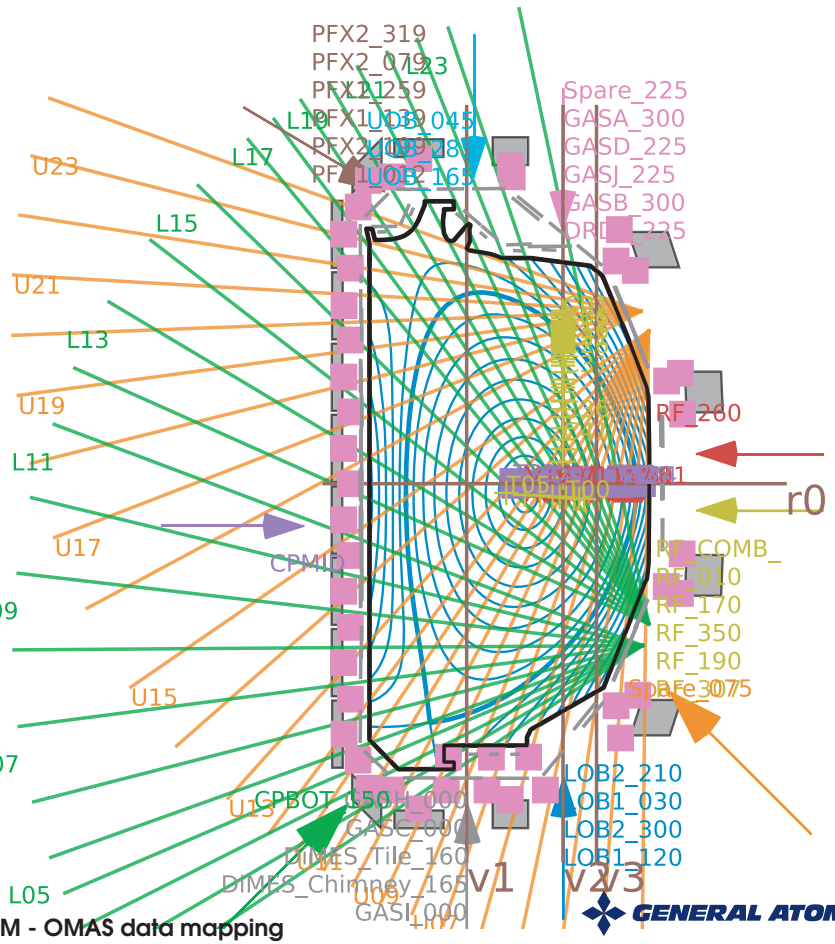
# OMAS machine data mappings work behind the scenes to help users get their data seamlessly (2/3)

- No need to know what data needs to be loaded to perform a task

```
# 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)
```



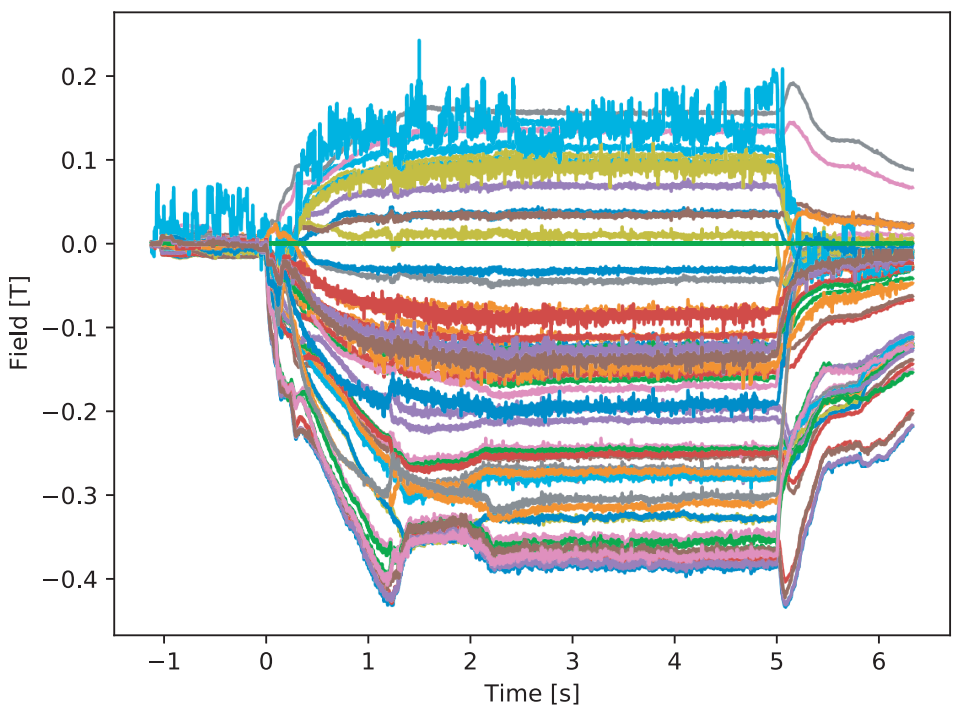
bolometer
charge_exchange
gas_injection
interferometer
langmuir_probes
magnetics
pf_active
thomson_scattering
wall



# 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

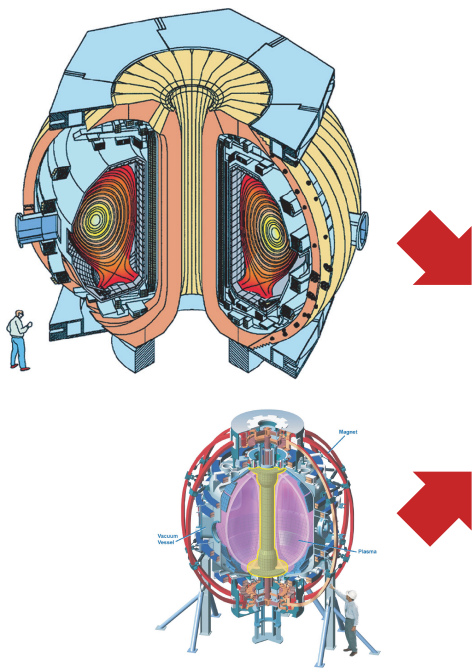
~7 seconds from my home
with SSH tunneling

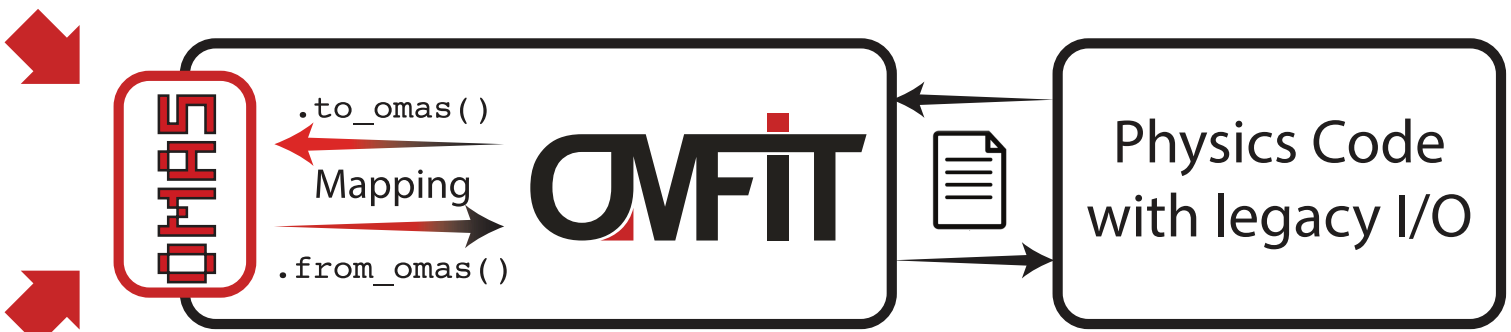
# We are leveraging OMAS machine mappings and OMFIT classes to generalize creation of kEQDSK EFIT input files

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

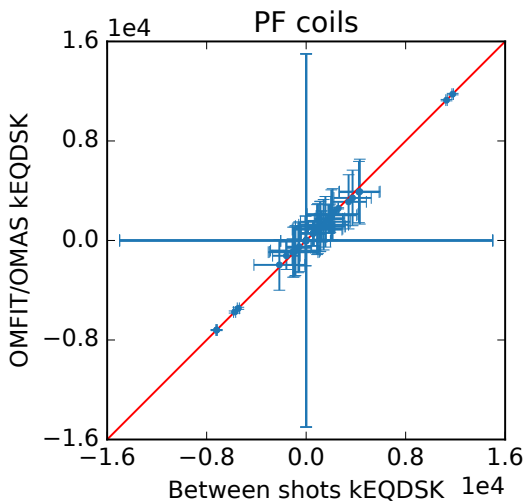
```
# Generate a kEQDSK file from experimental data
ods=ODS()
with ods.open('nstxu', 204202):
    kEQDSK = OMFITkeqdsk().from_omas(ods, time=0.369)
```

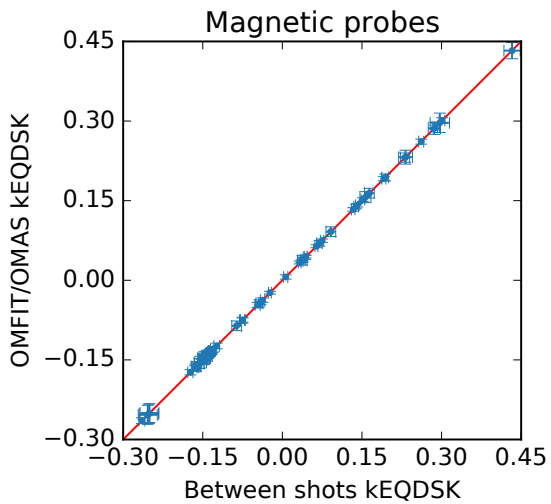


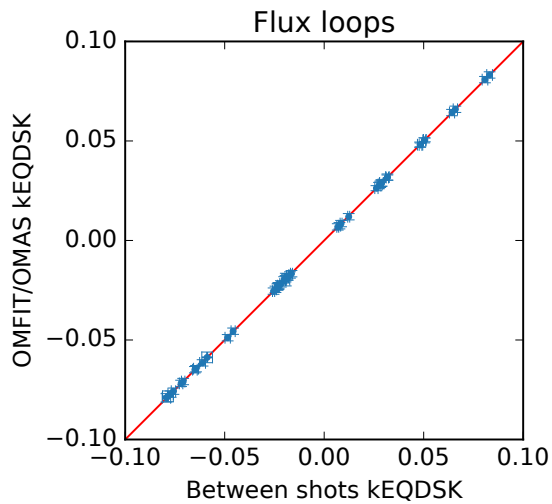


# Generating kEQDSK starting directly from raw experimental data gives us full control as to what goes into EFIT

- Started from magnetic-only equilibrium reconstruction
- A learning experience: better understanding of NSTX-U diagnostics
  - Many thanks to NSTX-U team for their support and insights

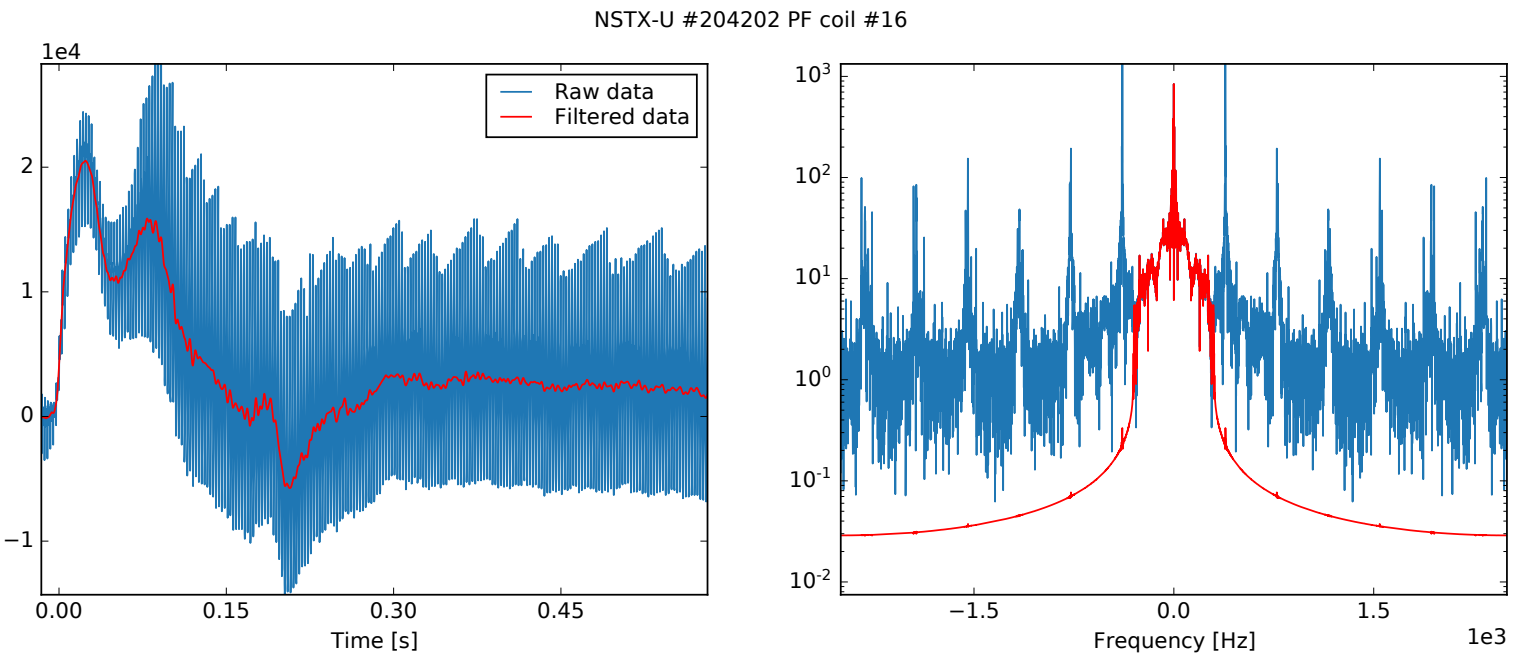


# Generating kEQDSK starting directly from raw experimental data gives us full control as to what goes into EFIT

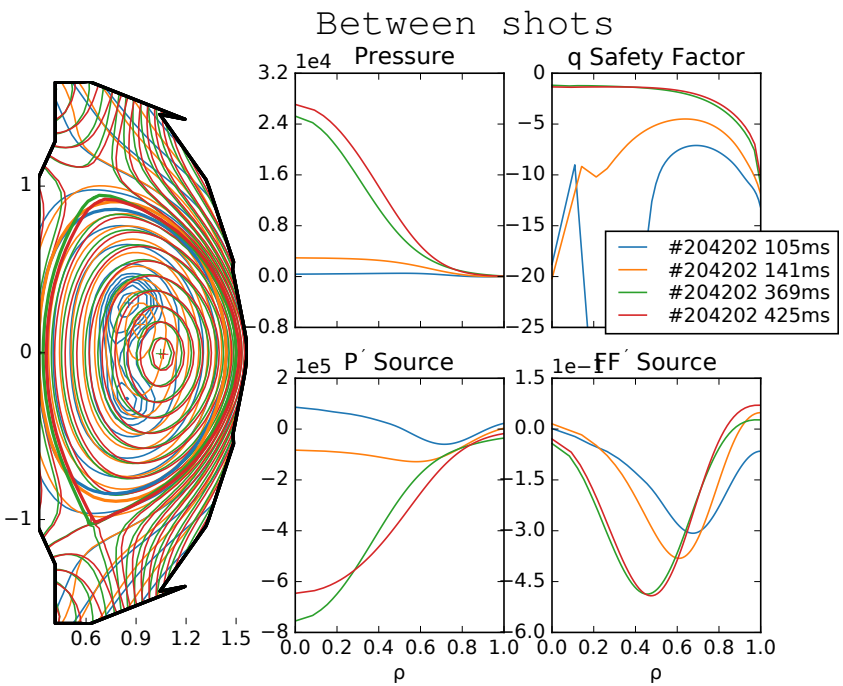
- We strive for transparency as to how data is processed
  - Easy to follow trail from raw data to input data
  - After data is mapped to IDS, then all processing routines are shared between different machines: shared effort and experience
- Give users flexibility depending on analyses of interest
  - Time-basis and averaging for individual diagnostics (eg. causal), data pre-processing, data filtering (eg. ELMS) and synchronization, ...
  - Eg. 300 Hz filter for NSTX-U PF measurements

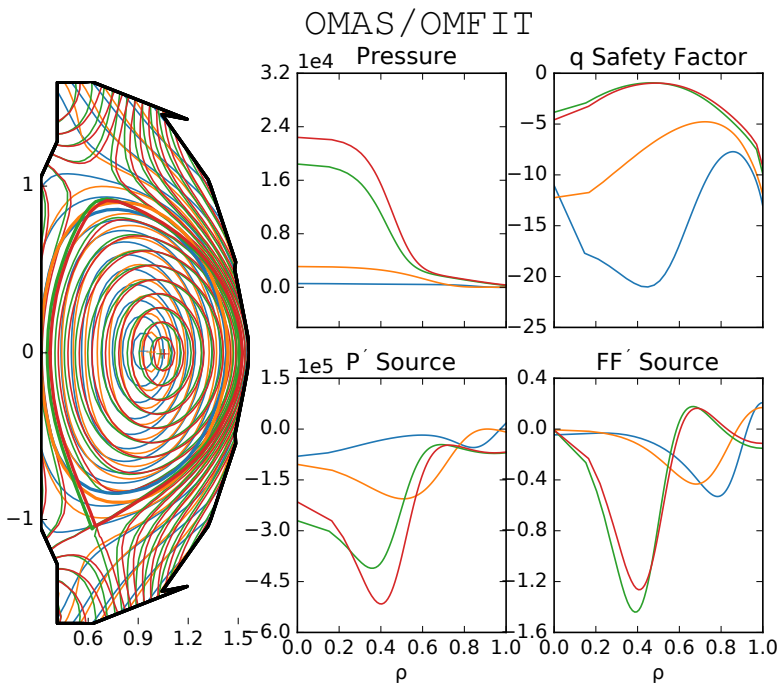


# Ongoing work focuses adding internal equilibrium constraints to kEQDSK files, also via OMAS

Without internal measurements P and q profiles not well constrained

- Few % differences in magnetics affect equilibrium significantly




- Working on generating kEQDSK internal constraints through IDSs:
  - MSE constraint (~ complete)
  - Pressure and edge current constraints (IDSify kinetic EFIT workflow)
  - Iso-thermal constraint
- Also, ongoing work to support high resolution EFIT (~ complete)

# Conclusions

**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:**

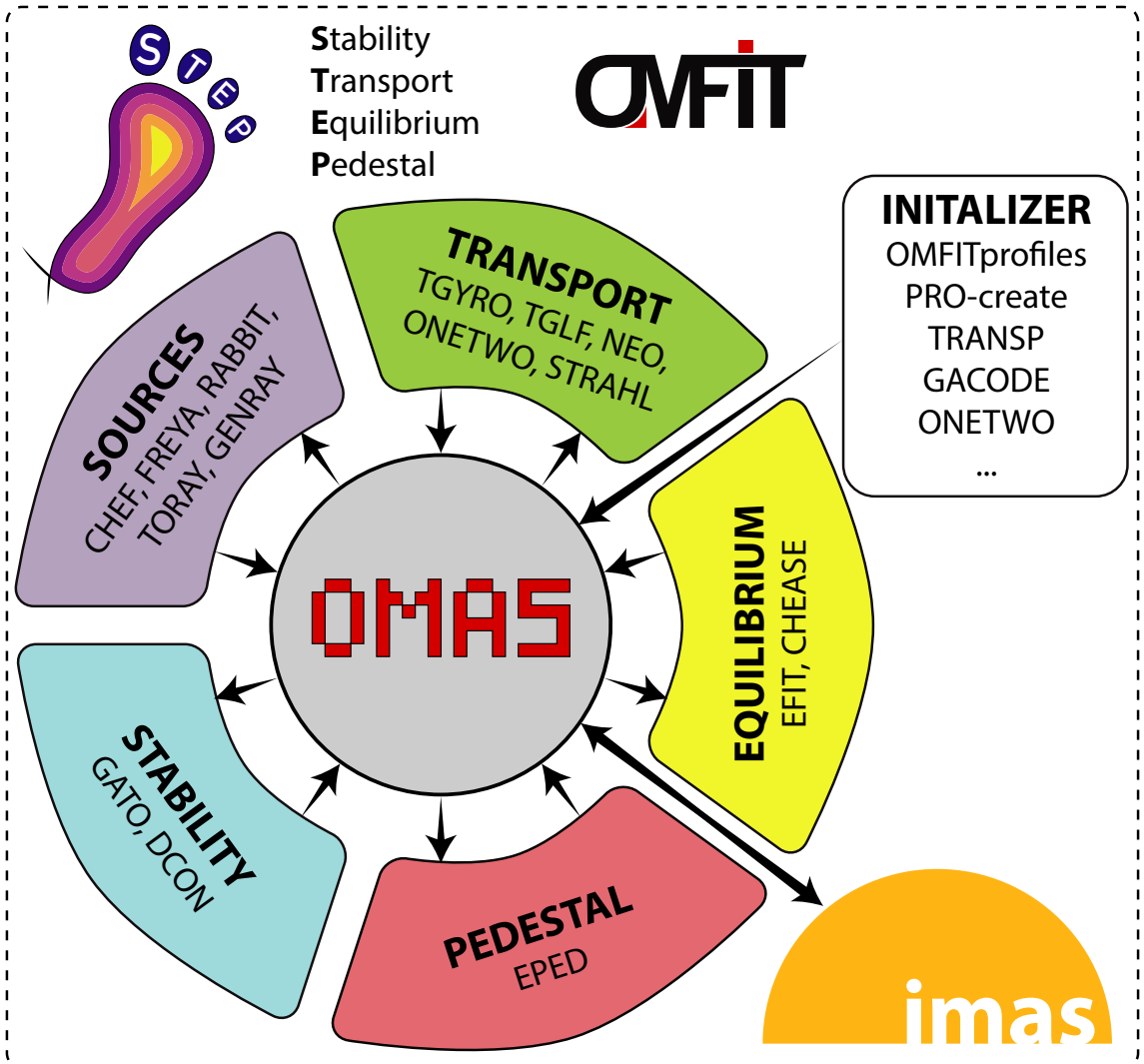
- Easy: No added software complexity or administrative burden
- Fast: Minimize latency by avoiding middle man

**OMFIT physics modules are rapidly being converted to use OMAS, and GA NSTX-U collaboration builds on top of such momentous acitivty**

- Generalize experimental analyses across different machines
- Predictive simulations via OMFIT STEP module

# STEP module in OMFIT uses OMAS to make self-consistent, theory-based predictions of tokamak plasmas

- Physics code are wrapped into *steps* that exchange data via OMAS:
  - Device agnostic
  - Adding a new code is $\mathcal{O}(N)$ not $\mathcal{O}(N^2)$
  - *Steps* can be executed in arbitrary order
- Permits a variety of workflows
  - Open loop prediction
  - Feedback control
  - Optimization
- To be applied to NSTX-U as part of the our collaboration

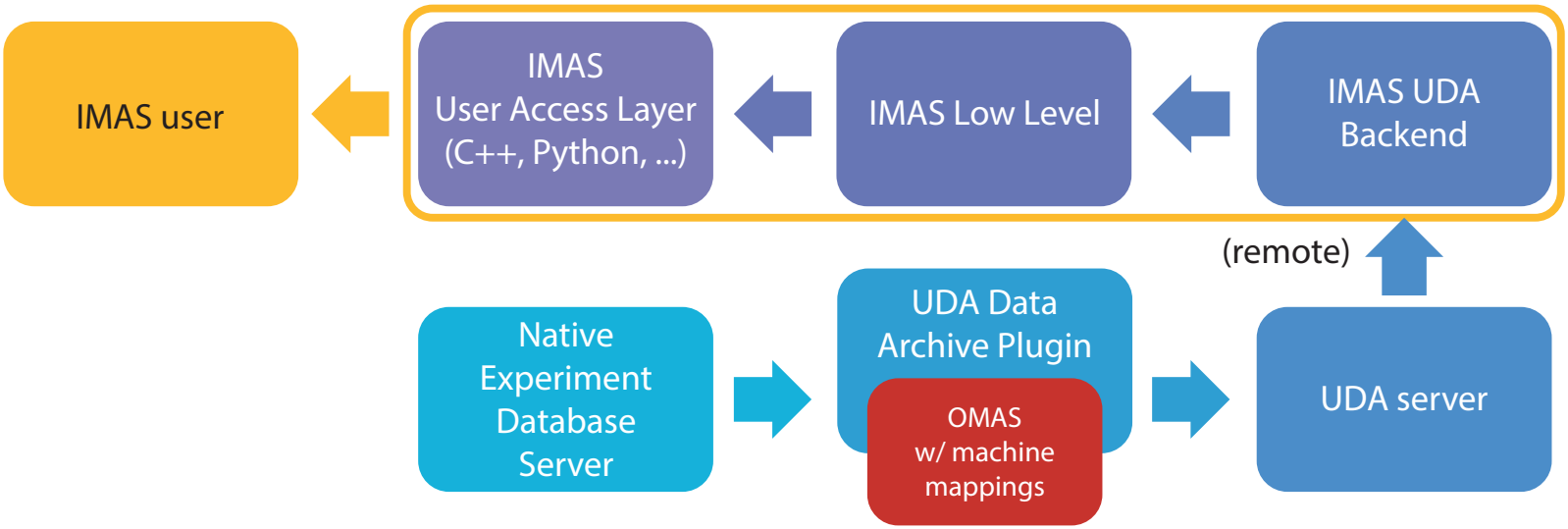


# Where we could get some extra help

- **Good examples and pointers**
  - Your favorite (exotic?) shots
  - NSTX-U shots with MSE
  - Iso-thermal constrained EFITs
- **Install latest version of MDS+ server**
  - Current version of NSTX-U MDS+ server does not support fetching multiple signals at once
  - Less efficient and slower (a lot slower, if many signals are involved)
  - No support for Python expressions
  - Everything would benefit from it, not only OMFIT/OMAS
- **Provide access to SQL server via ssh tunnel**
- **Synchronize SQL tables of EFIT runs to current state**
- **Provide PGF version of NTCC libraries for ONETWO**

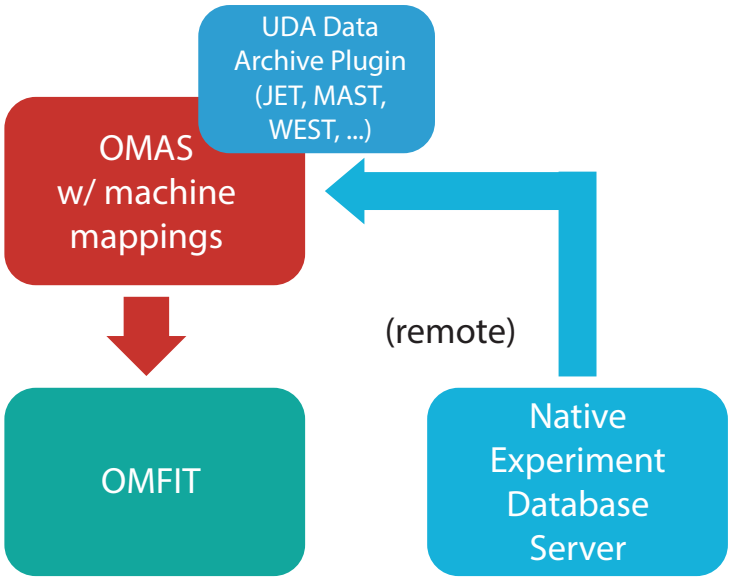
# Extra slides

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




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

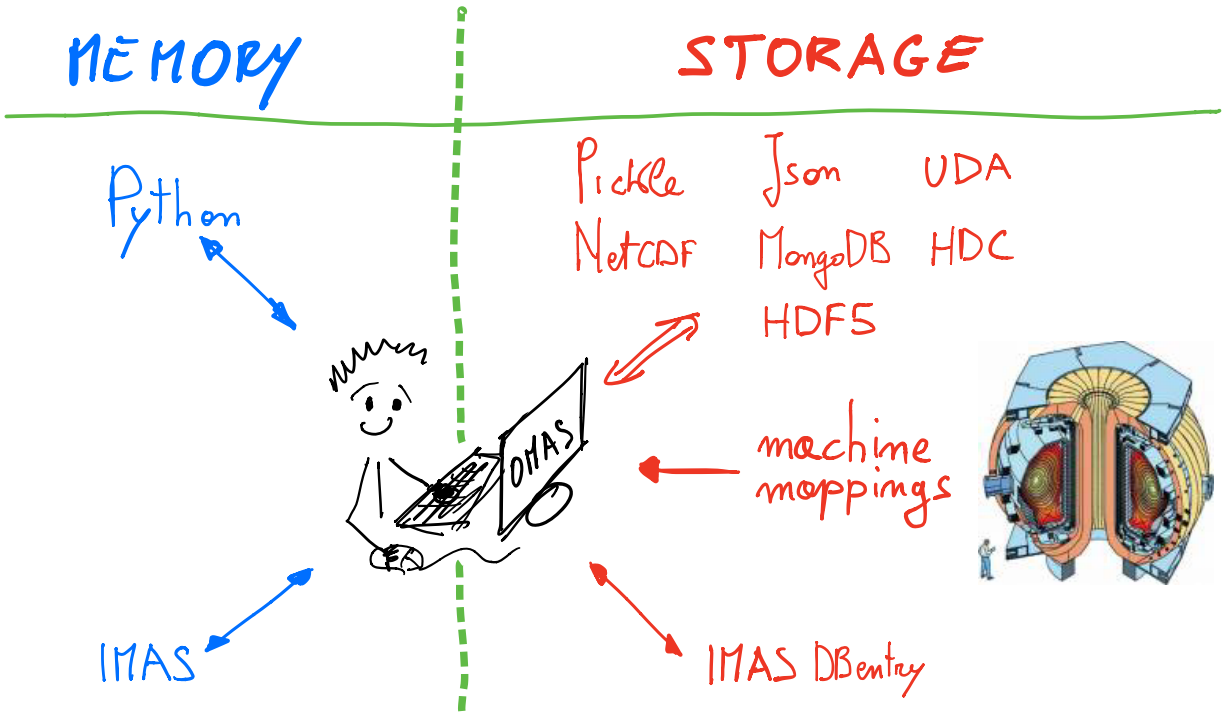
# Interoperability with IMAS:
# Existing UDA mappings could be used directly by OMAS




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

# 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

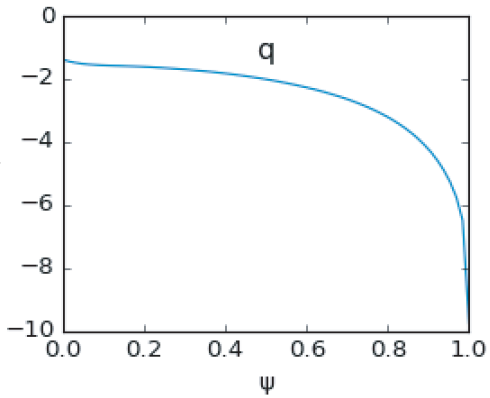
# OMAS dynamic data fetching works also for IMAS

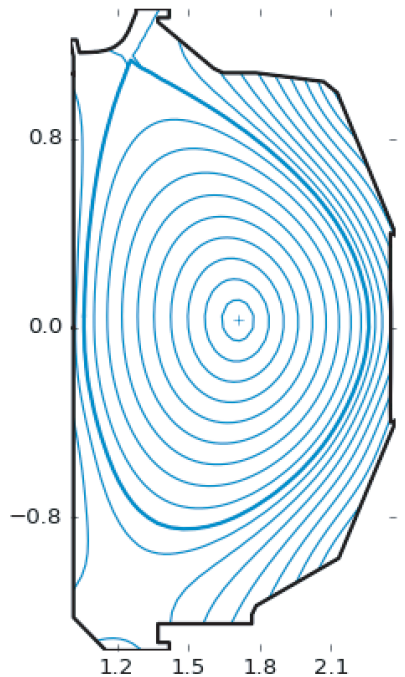
```
# OMAS with machine mappings dynamically loads data
ods = ODS()
with ods.open('machine', 'd3d', 168830):
    print(ods['equilibrium.time_slice.:.global_quantities.ip'])


# OMAS uses same approach for IMAS data access
ods = ODS()
with ods.open('imas', 'public', 'west', 55866, 0):
    print(ods['equilibrium.time_slice.:.global_quantities.ip'])
```

# OMAS provides convenient features beyond data storage → e.g. translate equilibrium data in IMAS schema

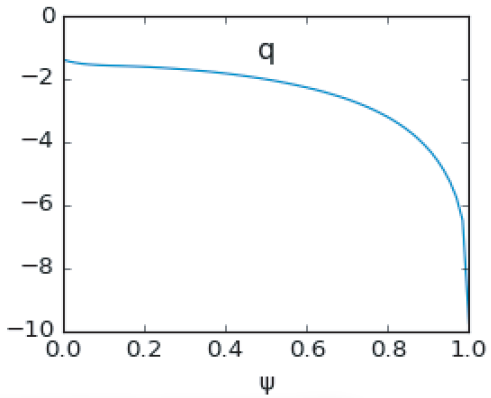
```
# Read EFIT equilibrium file "gfile"
gEQDSK = OMFITgeqdsk('./g123456.12345')

# Instantiate OMAS data structure object
ods = ODS()
```

```
# Shortcut
eq = ods['equilibrium']['time_slice'][0]
# 0D
eq['global_quantities.ip'] = gEQDSK['CURRENT']
# 1D
eq['profiles_1d.psi'] = np.linspace(0.0, 1.0, len(gEQDSK['QPSI']))
eq['profiles_1d.q'] = gEQDSK['QPSI']
# 2D
eq['profiles_2d.0.psi'] = gEQDSK['PSIRZ']
```



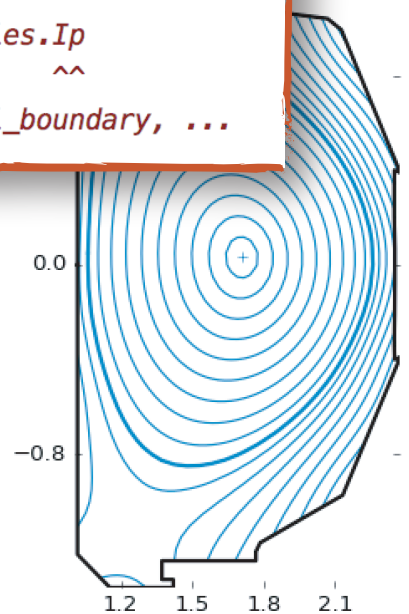
# OMAS enforces IMAS schema with graceful error handling




```
# Read EFIT equilibrium file "gfile"
gEQDSK = OMFITgeqdsk('./g123456.12345')

# Instantiate OMAS data structure object
ods = ODS()
```

```
LookupError: Not a valid IMAS 3.30.0 location:
             equilibrium.time_slice.0.global_quantities.Ip
                                                        ^^

Did you mean: ip, beta_pol, psi_axis, length_pol, psi_boundary, ...
```

```
    # Shortcut
    eq = ods['equilibrium']['time_slice'][0]
    # 0D
    eq['global_quantities.Ip'] = gEQDSK['CURRENT']
    # 1D
    eq['profiles_1d.psi'] = np.linspace(0.0, 1.0, len(gEQDSK['QPSI']))
    eq['profiles_1d.q'] = gEQDSK['QPSI']
    # 2D
    eq['profiles_2d.0.psi'] = gEQDSK['PSIRZ']
```

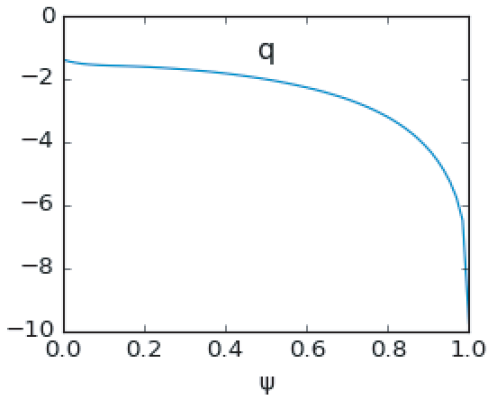


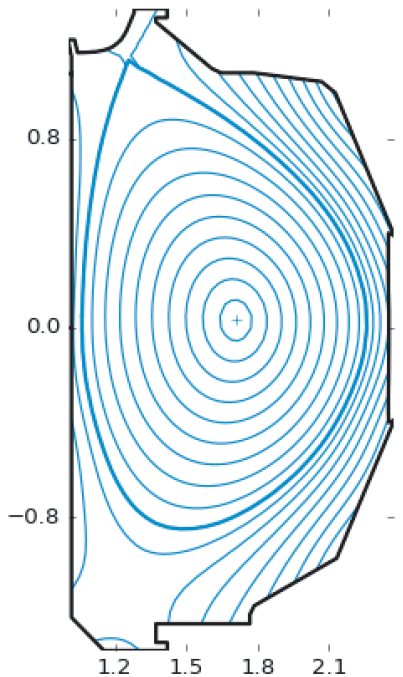
# Use of different syntaxes makes it easy to construct paths programmatically

```
# Read EFIT equilibrium file "gfile"
gEQDSK = OMFITgeqdsk('./g123456.12345')

# Instantiate OMAS data structure object
ods = ODS()
```

Notice access ODS sub-trees with different syntaxes

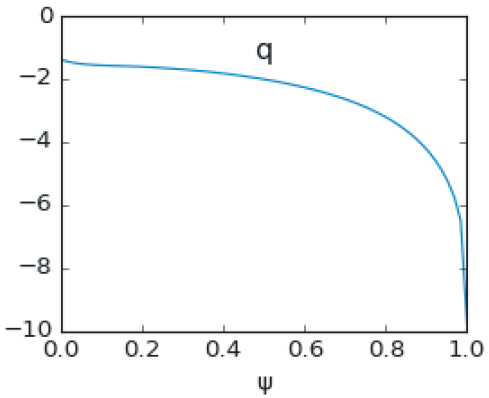
```
# Shortcut
eq = ods['equilibrium']['time_slice'][0]
# 0D
eq['global_quantities.ip'] = gEQDSK['CURRENT']
# 1D
eq['profiles_1d.psi'] = np.linspace(0.0, 1.0, len(gEQDSK['QPSI']))
eq['profiles_1d.q'] = gEQDSK['QPSI']
# 2D
eq['profiles_2d.0.psi'] = gEQDSK['PSIRZ']
```



# Automatic COordinate COnventionS translation greatly simplifies mapping of legacy data

```
# Read EFIT equilibrium file "gfile"
gEQDSK = OMFITgeqdsk('./g123456.12345')

# Instantiate OMAS data structure object
ods = ODS()

# Environment saying that ODS will be filled with data in given COCOS
# IMAS uses COCOS = 11
with omas_environment(ods, cocosio=1):
    # Shortcut
    eq = ods['equilibrium']['time_slice'][0]
    # 0D
    eq['global_quantities.ip'] = gEQDSK['CURRENT']
    # 1D
    eq['profiles_1d.psi'] = np.linspace(0.0, 1.0, len(gEQDSK['QPSI']))
    eq['profiles_1d.q'] = gEQDSK['QPSI']
    # 2D
    eq['profiles_2d.0.psi'] = gEQDSK['PSIRZ']
```

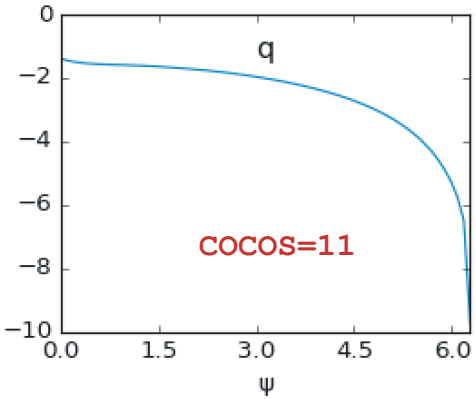



16 possible COCOS:

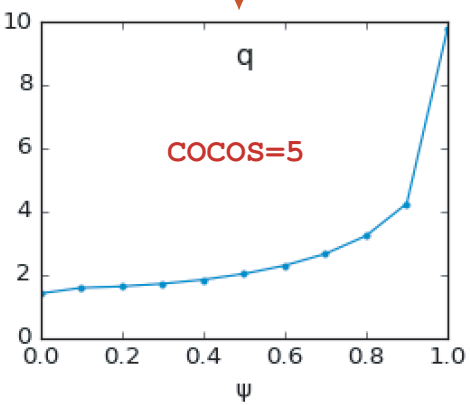
- Direction of $\varphi$
- Direction of $\theta$
- Sign of $\nabla\varphi \times \nabla\psi$
- $2\pi$ normalization $\nabla\varphi \times \nabla\psi$

# COCOS translation avoids mistakes, and automatic grid interpolation enables true centralized data communication

```
# IMAS uses COCOS = 11
plt.plot(eq['profiles_1d.psi'], eq['profiles_1d.q'])
```

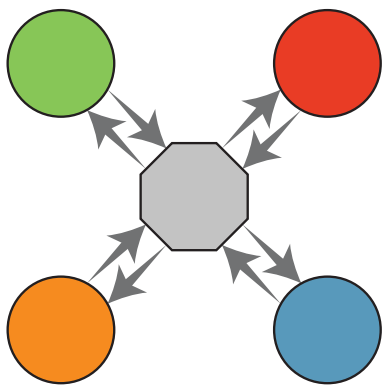



```
# Retrieve data from ODS with a given COCOS and grid, independently of how it was filled
coordsio = {'equilibrium.time_slice.0.profiles_1d.psi': np.linspace(0, 1, 11)}
with omas_environment(ods, cocosio=5, coordsio=coordsio):
    plt.plot(eq['profiles_1d.psi'], eq['profiles_1d.q'])
```



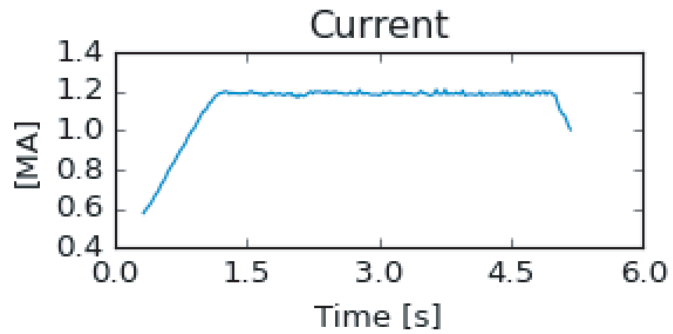

True centralized data communication requires data retrieval to be independent of how data was filled

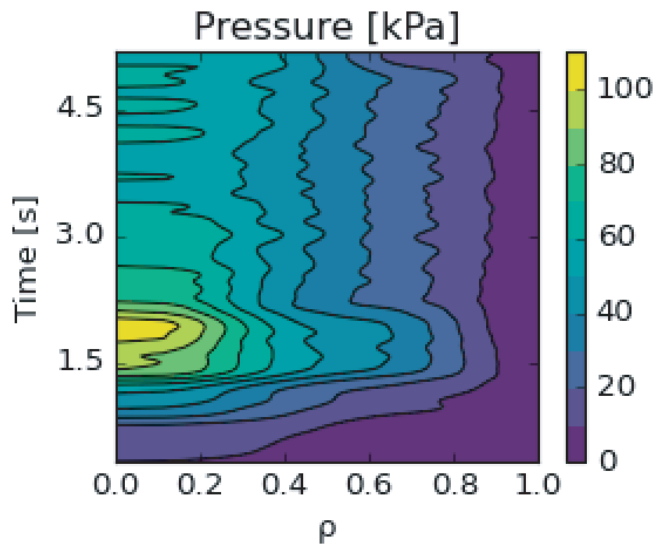
Centralized data communication



# Data slicing across arrays of structures avoids explicit loops

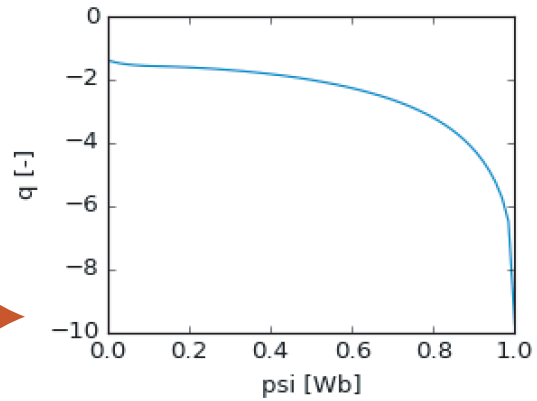
```
# Slice arrays of structures with `:`
ods['equilibrium.time_slice[:].global_quantities.ip']
ods['equilibrium.time_slice[:].profiles_1d.pressure']
ods['core_profiles.profiles_1d[:].ion[:].density']
```



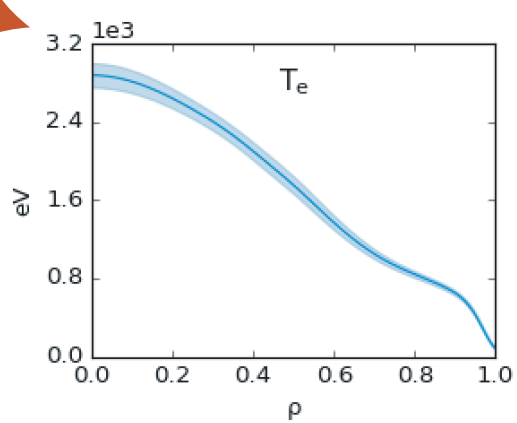
# OMAS can cast its data as xarray complete with units and coordinates

```
# Slice arrays of structures with `:`
ods['equilibrium.time_slice[:].global_quantities.ip']
ods['equilibrium.time_slice[:].profiles_1d.pressure']
ods['core_profiles.profiles_1d[:].ion[:].density']


# xarray.DataArray representation of individual quantities
# with coordinates and attributes (such as for units)
xa = ods.xarray('equilibrium.time_slice[0].profiles_1d.q')
xa.plot()
```



# OMAS seamlessly handles uncertain data and units
# No need to handle error in separate IMAS location

```
# Slice arrays of structures with `:`
ods['equilibrium.time_slice[:].global_quantities.ip']
ods['equilibrium.time_slice[:].profiles_1d.pressure']
ods['core_profiles.profiles_1d[:].ion[:].density']


# xarray.DataArray representation of individual quantities
# with coordinates and attributes (such as for units)
xa = ods.xarray('equilibrium.time_slice[0].profiles_1d.q')
xa.plot()


# Seamlessly handle uncertain data (`uncertainties` package) and units (`pint` package)
ureg = pint.UnitRegistry()
ods['thomson_scattering.channel[0].t_e.data'] = unumpy.uarray(te, te_err) * ureg.keV
```



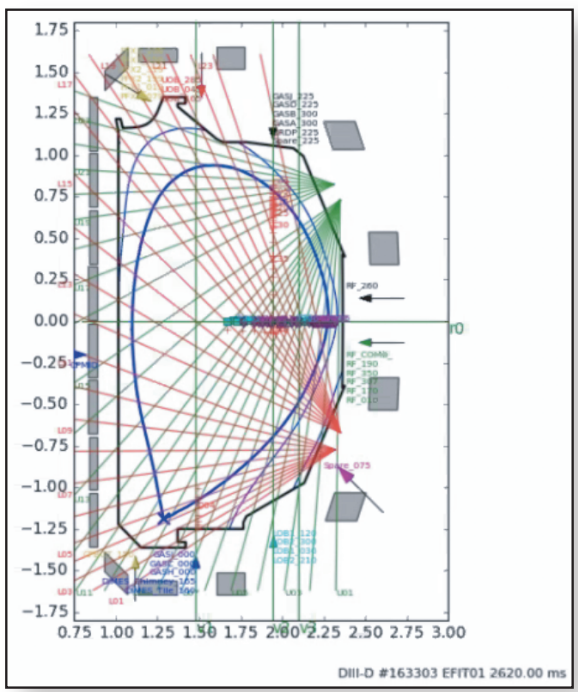
# Automatically translate code parameters from/to XML representation (as required by IMAS standard)

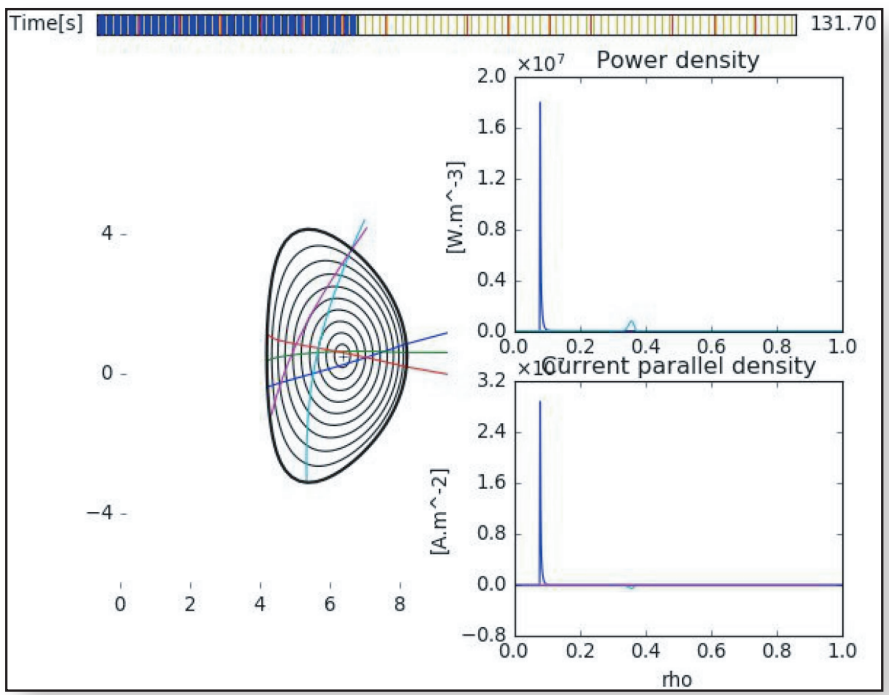
```python
# Slice arrays of structures with `:`
ods['equilibrium.time_slice[:].global_quantities.ip']
ods['equilibrium.time_slice[:].profiles_1d.pressure']
ods['core_profiles.profiles_1d[:].ion[:].density']


# xarray.DataArray representation of individual quantities
# with coordinates and attributes (such as for units)
xa = ods.xarray('equilibrium.time_slice[0].profiles_1d.q')
xa.plot()


# Seamlessly handle uncertain data (`uncertainties` package) and units (`pint` package)
ureg = pint.UnitRegistry()
ods['thomson_scattering.channel[0].t_e.data'] = unumpy.uarray(te, te_err) * ureg.keV


# Automatic handling of XML code.parameters when interacting with IMAS
ods['equilibrium.code.parameters.test.parameter1'] = 1
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<parameters>
  <test>
    <parameter1>1</parameter1>
  </test>
</parameters>
```

# OMAS comes with a rich library of physics and plotting routines

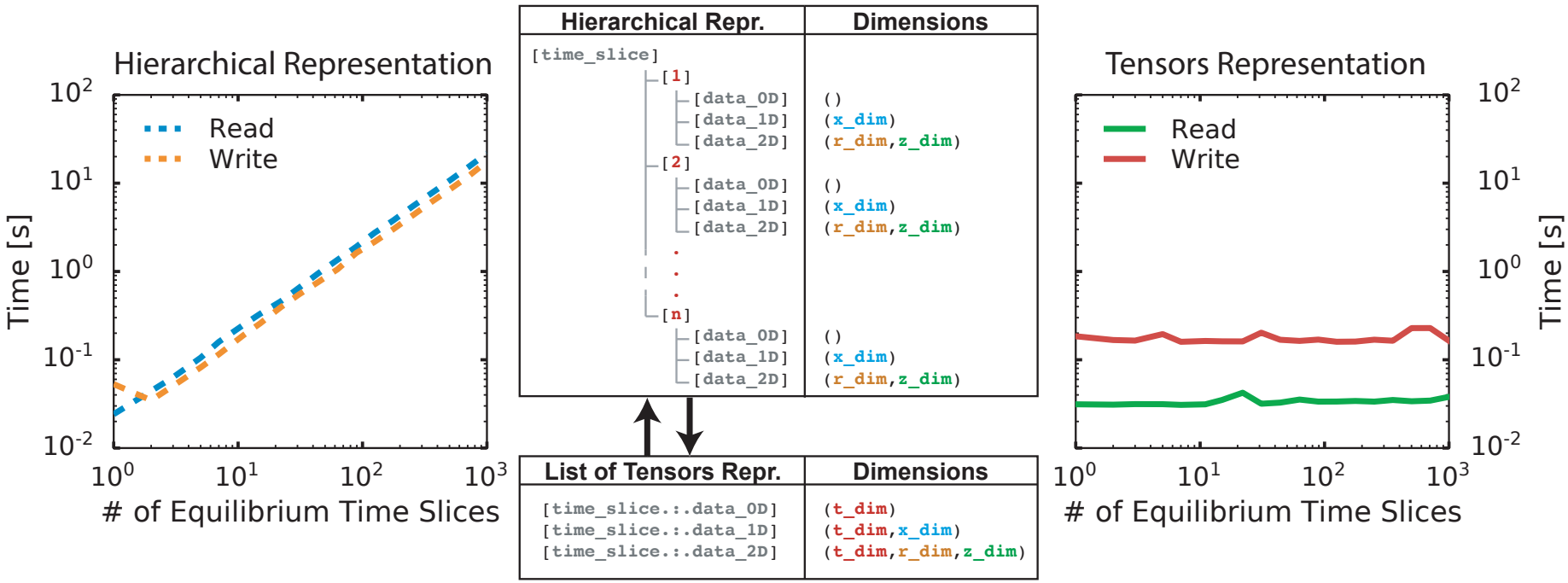
```
# Library of physics routines: e.g.
ods.physics_core_profiles_consistent()  # pressures, densities, zeff, ...
ods.physics_equilibrium_ggd_to_rectangular()  # eq GGD to rectangular
ods.physics_summary_consistent_global_quantities()  # summary IDS from all other IDSs

# Library of (time-dependent) plot routines: e.g.
ods.plot_overlay()  # bolometer, cx, gas, interferometer, magnetics, pf, ts, ...
ods.waves_beam_summary()  # equilibrium, core_profiles, transport, ...
```



# OMAS prototyped tensor data representation used to improve IMAS HDF5 I/O performance of large datasets

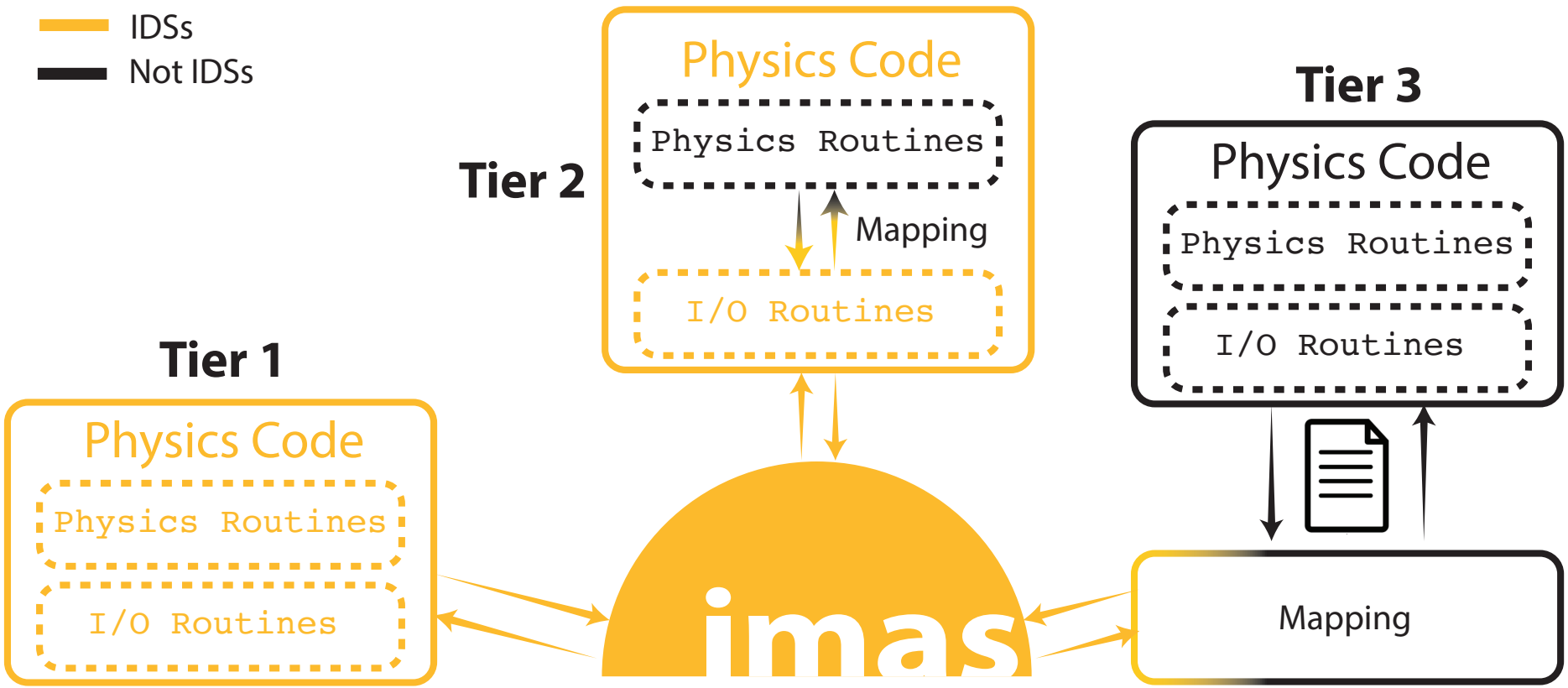
- Hierarchical representation does not allow bulk read/write of data
- Tensor representation commonly used for HPC and ML applications
- Mapping requires constant grids across arrays of structures



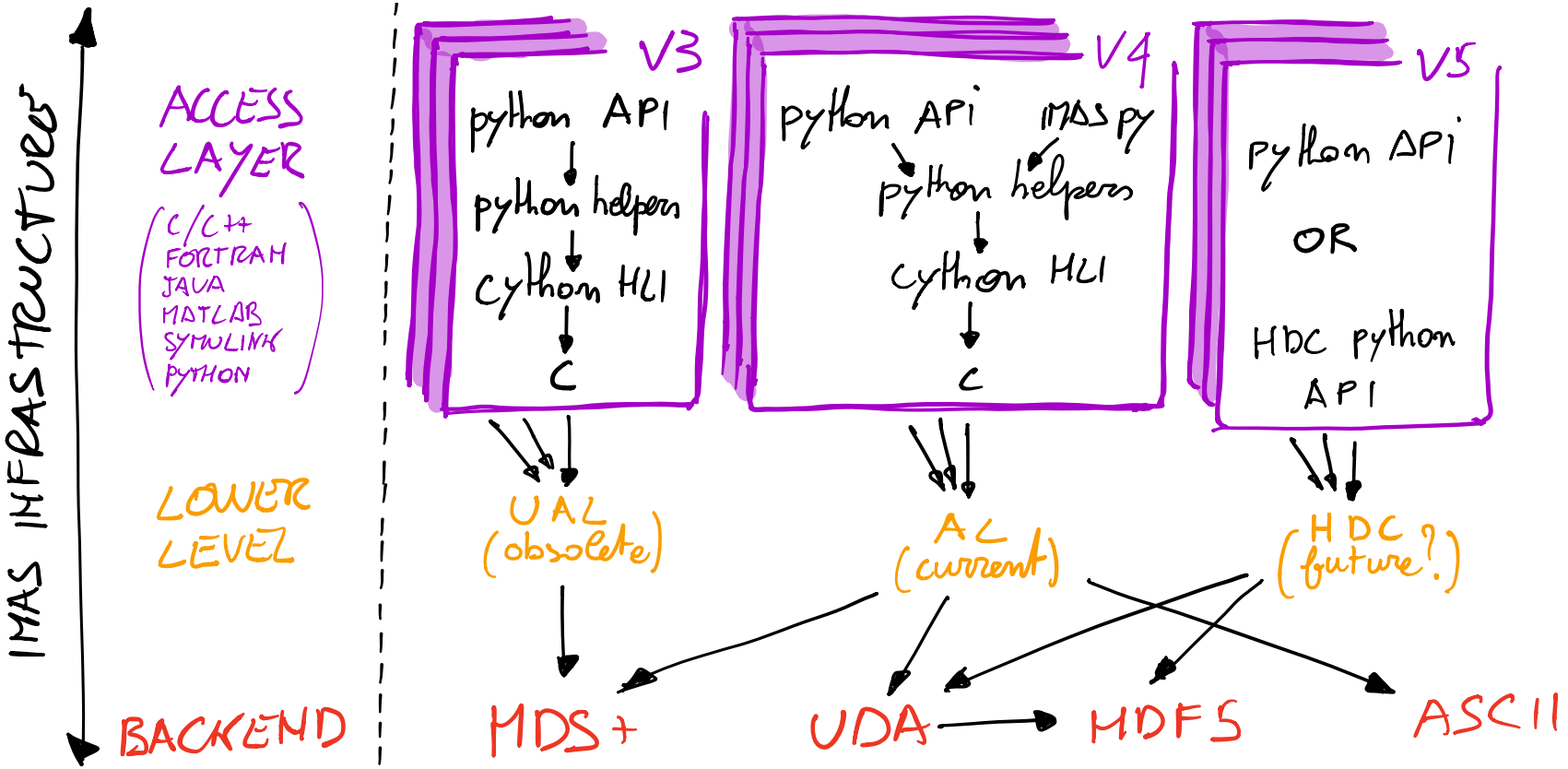

```
# ODX: a tensorized representation of an ODS
odx = ods.to_odx()
# ODX maintains same API of ODS hierarchical representation
odx['equilibrium.time_slice.:.profiles_1d.q'] # slicing
odx['equilibrium.time_slice.0.profiles_1d.q'] # individual
```

# 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

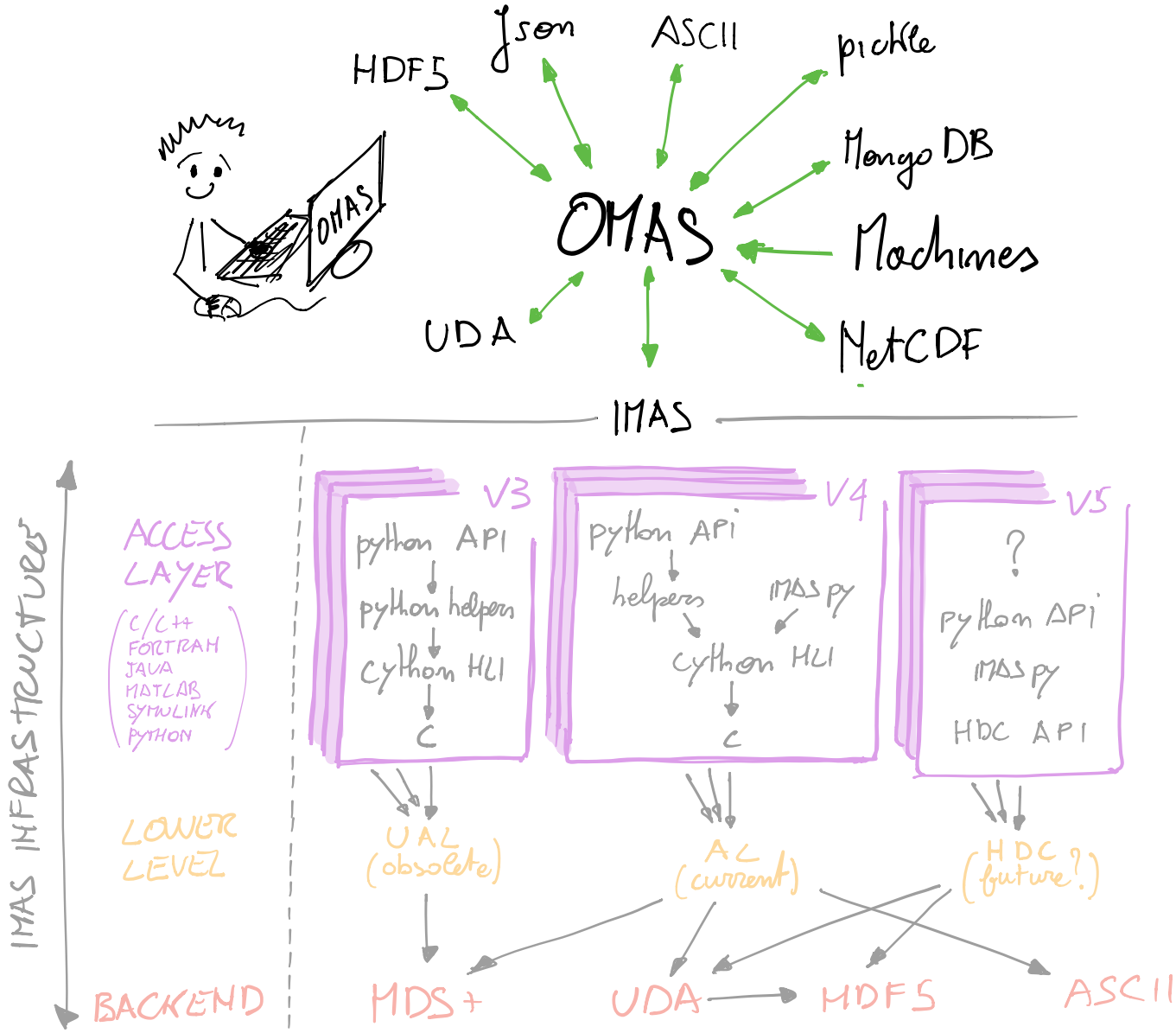


# The IMAS software stack is tall and evolving




- 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