Lyncs-API

A Python Interface for Lattice QCD applications

Dr. Simone Bacchio

The Cyprus Institute

RESEARCH·TECHNOLOGY·INNOVATION

for the Lyncs-API Community

https://github.com/Lyncs-API

Funded by
PRACE-6IP, WP8 "Forward Looking Software Solutions".
Grant agreement ID: 823767, Project name: LyNcs.

S. Bacchio - 28/07/21 - Lattice21

Lyncs-API in a nutshell

What?

A new community-oriented open-source software for Lattice QCD

Why?

- ➢ **3Ps:** Performance, Portability, Productivity
- ➢ Distributed tasking and modular computing

How?

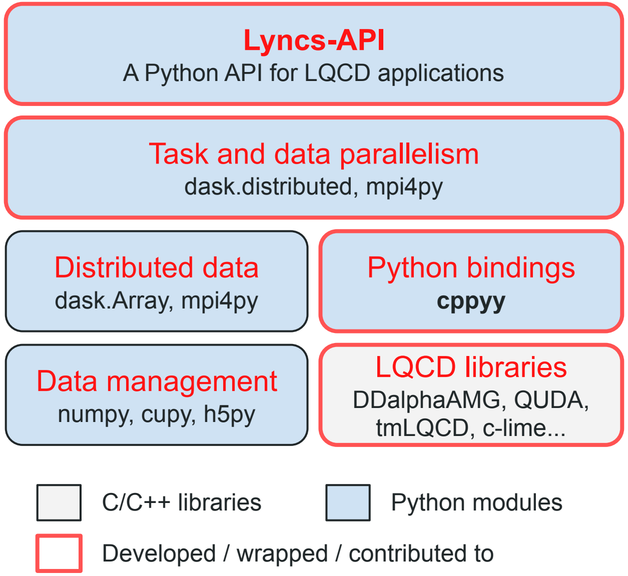
- ➢ **In Python!**
  - Interfaces to Lattice QCD Libraries
  - Low-level Python HPC tools
  - High-level Python API

Where?

On Github: https://github.com/Lyncs-API

When?

Under development! Stay tuned or contribute! :)



Lyncs-API
A Python API for LQCD applications

Task and data parallelism
dask.distributed, mpi4py

Distributed data
dask.Array, mpi4py

Python bindings
cppyy

Data management
numpy, cupy, h5py

LQCD libraries
DDalphaAMG, QUDA, tmLQCD, c-lime...

□ C/C++ libraries
□ Python modules
□ Developed / wrapped / contributed to

Design and originality

- Interface to as many libraries as possible
  - ➢ Performance and portability
  - ➢ Involvement of the community
  - ➢ **Crosschecks** and benchmarks!
  - ➢ Second life to legacy code

LQCD libraries
DDalphaAMG, QUDA, tmLQCD, c-lime...

Grid?

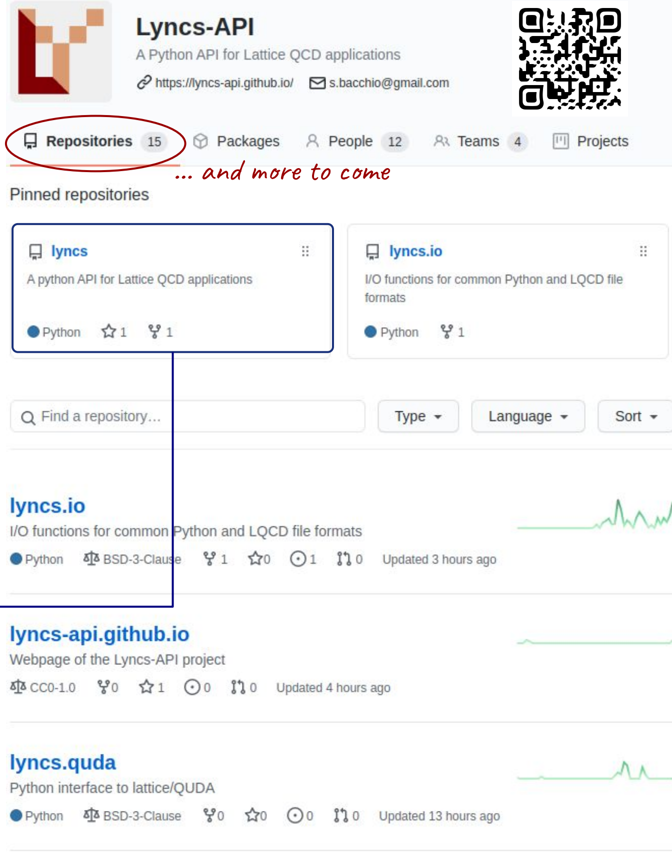
GPT - Grid Python Toolkit
https://github.com/lehner/gpt

- A modular ecosystem
  - ➢ Separation of concerns
  - ➢ Clean dependencies
  - ➢ Easy distribution and reuse

```
$ pip install lyncs_io
$ pip install lyncs_DDalphaAMG
```

- High-level API (Planned)
  - ➢ Wrap-up and combine
  - ➢ Support for parallel tasking
  - ➢ Optimization of execution time

```
$ pip install lyncs[io]
$ pip install lyncs[DDalphaAMG]
```



Lyncs-API
A Python API for Lattice QCD applications
https://lyncs-api.github.io/ s.bacchio@gmail.com

... and more to come

Pinned repositories

lyncs
A python API for Lattice QCD applications
Python ☆1 1

lyncs.io
I/O functions for common Python and LQCD file formats
Python 1

lyncs.io
I/O functions for common Python and LQCD file formats
Python BSD-3-Clause 1 ☆0 1 0 Updated 3 hours ago

lyncs-api.github.io
Webpage of the Lyncs-API project
CC0-1.0 0 ☆1 0 0 Updated 4 hours ago

lyncs.quda
Python interface to lattice/QUDA
Python BSD-3-Clause 0 ☆0 0 0 Updated 13 hours ago

External Python Tools

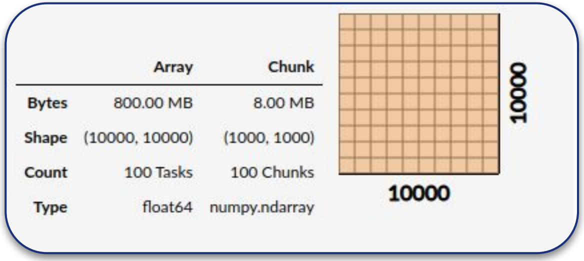
- **Cppyy:** https://cppyy.readthedocs.io/
  - ➢ Automatic binding to C/C++ libraries
  - ➢ Supports C++ templates and overloading
  - ➢ Just in time compiler

```
>>> import cppyy
>>> cppyy.include('zlib.h')        # bring in C++ header
>>> cppyy.load_library('libz')     # load linker symbols
>>> cppyy.gbl.zlibVersion()        # use a zlib API
'1.2.11'
```

```
>>> from cppyy.gbl.std import vector, pair
>>> v = vector[int](range(10))
>>> len(v)
10
>>> v += range(10, 20)
>>> len(v)
20
```

- **Dask:** https://docs.dask.org/
  - ➢ Management of distributed data and tasks
  - ➢ Task scheduling and parallelism via futures
  - ➢ Distributed numpy arrays

```
>>> import dask.array as da
>>> x = da.random.random((10000, 10000), chunks=(1000, 1000))
>>> x
```

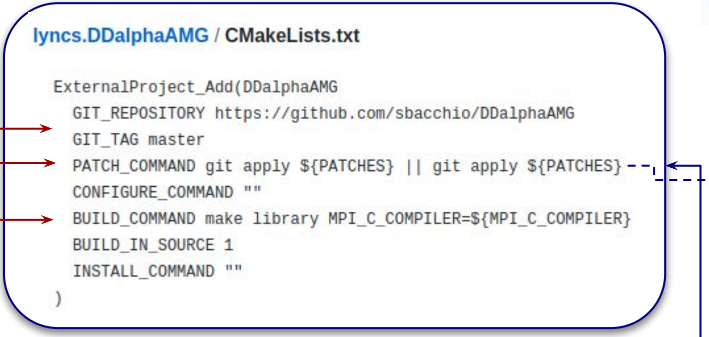


	Array	Chunk
Bytes	800.00 MB	8.00 MB
Shape	(10000, 10000)	(1000, 1000)
Count	100 Tasks	100 Chunks
Type	float64	numpy.ndarray

- Others
  - ➢ Numpy: API for array operations on CPUs
  - ➢ Cupy: Numpy-like interface on GPUs
  - ➢ mpi4py: Python interface to MPI
  - ➢ h5py: Python interface to HDF5

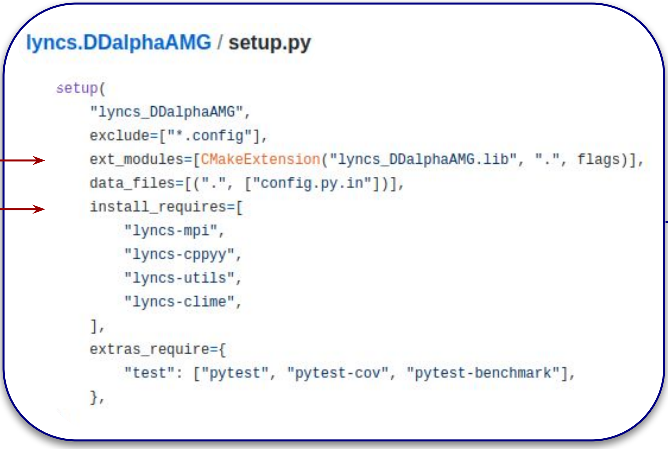
Interfacing to Libraries

1. Installation:
   a. CMakeLists.txt
      i. Clone package
      ii. Apply patches
      iii. Compile and install
   b. setup.py (`lyncs_setuptools`)
      i. Run CMake
      ii. Install dependencies
      iii. Distribute on `pip`



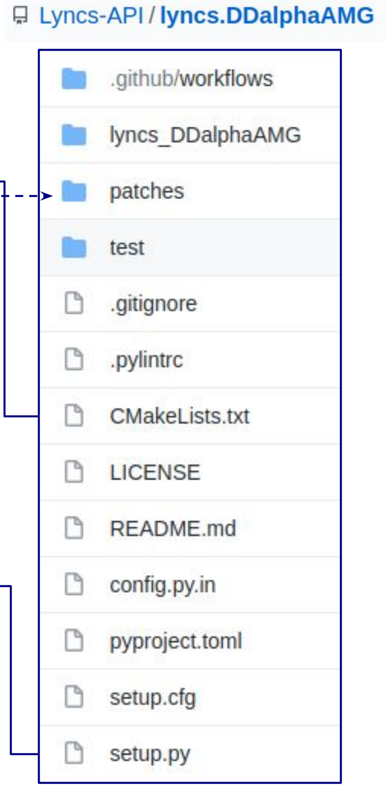
lyncs.DDalphaAMG / CMakeLists.txt

```
ExternalProject_Add(DDalphaAMG
  GIT_REPOSITORY https://github.com/sbacchio/DDalphaAMG
  GIT_TAG master
  PATCH_COMMAND git apply ${PATCHES} || git apply ${PATCHES}
  CONFIGURE_COMMAND ""
  BUILD_COMMAND make library MPI_C_COMPILER=${MPI_C_COMPILER}
  BUILD_IN_SOURCE 1
  INSTALL_COMMAND ""
)
```



lyncs.DDalphaAMG / setup.py

```
setup(
    "lyncs_DDalphaAMG",
    exclude=["*.config"],
    ext_modules=[CMakeExtension("lyncs_DDalphaAMG.lib", ".", flags)],
    data_files=[(".", ["config.py.in"])],
    install_requires=[
        "lyncs-mpi",
        "lyncs-cppyy",
        "lyncs-utils",
        "lyncs-clime",
    ],
    extras_require={
        "test": ["pytest", "pytest-cov", "pytest-benchmark"],
    },
```



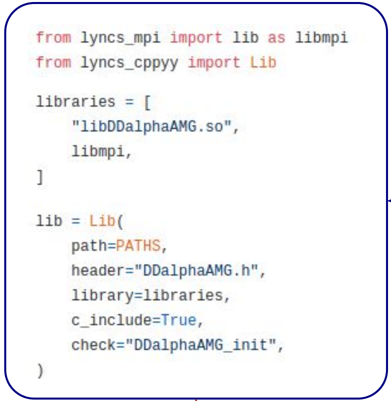
Lyncs-API / **lyncs.DDalphaAMG**

- .github/workflows
- lyncs_DDalphaAMG
- patches
- test
- .gitignore
- .pylintrc
- CMakeLists.txt
- LICENSE
- README.md
- config.py.in
- pyproject.toml
- setup.cfg
- setup.py

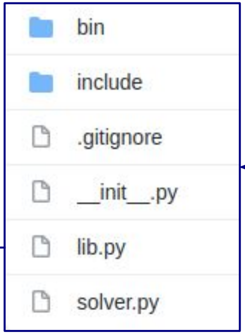
Interfacing to Libraries

1. Installation:
   a. CMakeLists.txt
      i. Clone package
      ii. Apply patches
      iii. Compile and install
   b. setup.py (`lyncs_setuptools`)
      i. Run CMake
      ii. Install dependencies
      iii. Distribute on `pip`

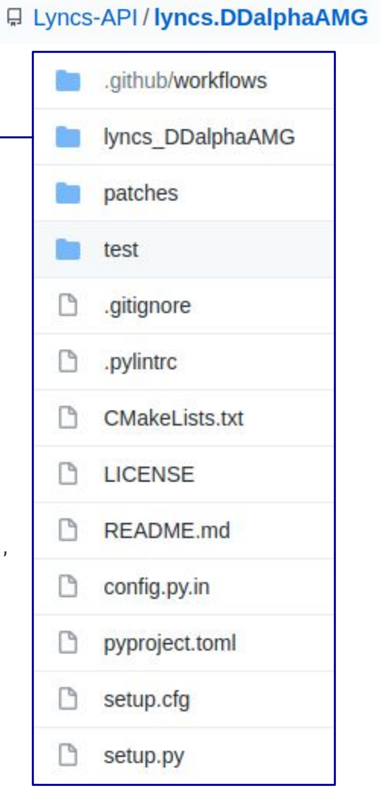
2. Interface:
   a. `lib.py`
      i. Load headers and library
      ii. Manage initialization and global parameters
   b. `solver.py`
      i. High-level python interface
      ii. Follow library structure



```python
from lyncs_mpi import lib as libmpi
from lyncs_cppyy import Lib

libraries = [
    "libDDalphaAMG.so",
    libmpi,
]

lib = Lib(
    path=PATHS,
    header="DDalphaAMG.h",
    library=libraries,
    c_include=True,
    check="DDalphaAMG_init",
)
```

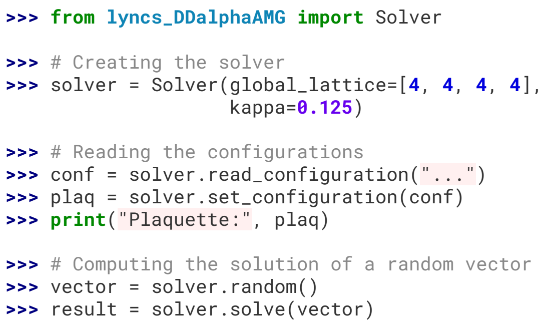


- bin
- include
- .gitignore
- __init__.py
- lib.py
- solver.py



Lyncs-API / **lyncs.DDalphaAMG**

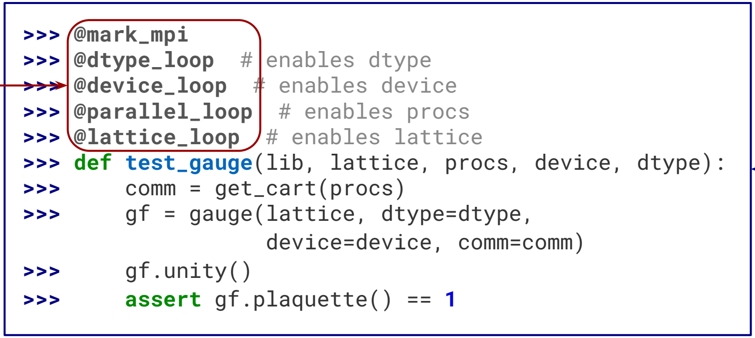
- .github/workflows
- lyncs_DDalphaAMG
- patches
- test
- .gitignore
- .pylintrc
- CMakeLists.txt
- LICENSE
- README.md
- config.py.in
- pyproject.toml
- setup.cfg
- setup.py



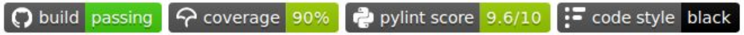
```python
>>> from lyncs_DDalphaAMG import Solver

>>> # Creating the solver
>>> solver = Solver(global_lattice=[4, 4, 4, 4],
                    kappa=0.125)

>>> # Reading the configurations
>>> conf = solver.read_configuration("...")
>>> plaq = solver.set_configuration(conf)
>>> print("Plaquette:", plaq)

>>> # Computing the solution of a random vector
>>> vector = solver.random()
>>> result = solver.solve(vector)
```

Interfacing to Libraries

3. Testing:
   a. Python tests (pytest)
      i. Unit testing
      ii. Loops over all cases
      iii. High **coverage**
   b. CI/CD (Github Actions)
      i. Run tests for every PR …
      ii. and when deps are updated
      iii. Upload new release on pip

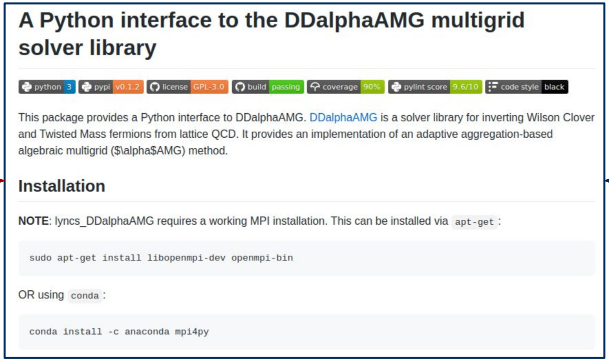


```
>>> @mark_mpi
>>> @dtype_loop  # enables dtype
>>> @device_loop  # enables device
>>> @parallel_loop  # enables procs
>>> @lattice_loop # enables lattice
>>> def test_gauge(lib, lattice, procs, device, dtype):
>>>     comm = get_cart(procs)
>>>     gf = gauge(lattice, dtype=dtype,
                   device=device, comm=comm)
>>>     gf.unity()
>>>     assert gf.plaquette() == 1
```



build passing | coverage 90% | pylint score 9.6/10 | code style black

4. Documentation:
   a. README.md
      i. Installations instructions
      ii. Short documentation and examples
   b. Readthedocs (Planned)
      i. Standard for Python packages
      ii. Collective documentation for Lyncs-API



A Python interface to the DDalphaAMG multigrid solver library

python 3 | pypi v0.1.2 | license GPL-3.0 | build passing | coverage 90% | pylint score 9.6/10 | code style black

This package provides a Python interface to DDalphaAMG. DDalphaAMG is a solver library for inverting Wilson Clover and Twisted Mass fermions from lattice QCD. It provides an implementation of an adaptive aggregation-based algebraic multigrid (αAMG) method.

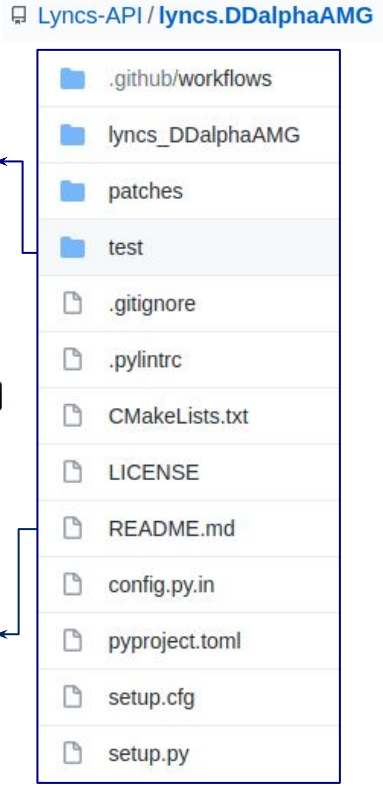
Installation

NOTE: lyncs_DDalphaAMG requires a working MPI installation. This can be installed via `apt-get` :

```
sudo apt-get install libopenmpi-dev openmpi-bin
```

OR using `conda` :

```
conda install -c anaconda mpi4py
```



Lyncs-API / **lyncs.DDalphaAMG**

- .github/workflows
- lyncs_DDalphaAMG
- patches
- test
- .gitignore
- .pylintrc
- CMakeLists.txt
- LICENSE
- README.md
- config.py.in
- pyproject.toml
- setup.cfg
- setup.py

Highlights & Examples

I/O

```
>>> from lyncs_io import load, save

>>> # Load Numpy array
>>> arr = load("array.npy")

>>> # Format deduced from the extension
>>> arr = load("array.npy",
               format="numpy")

>>> # Load in parallel with MPI
>>> arr = load("array.npy", comm=cart)

>>> # Load in parallel with Dask
>>> arr = load("array.npy",
               chunks=(4,4,4))

>>> # Support for HDF5
>>> arr = load("data.h5/array")

>>> # Also lime format
>>> arr = load("conf.lime")

>>> # And the same for saving
>>> save(arr, "array.npy")
>>> save(arr, "data.h5/array")
>>> save(arr, "conf.lime", comm=cart)
```

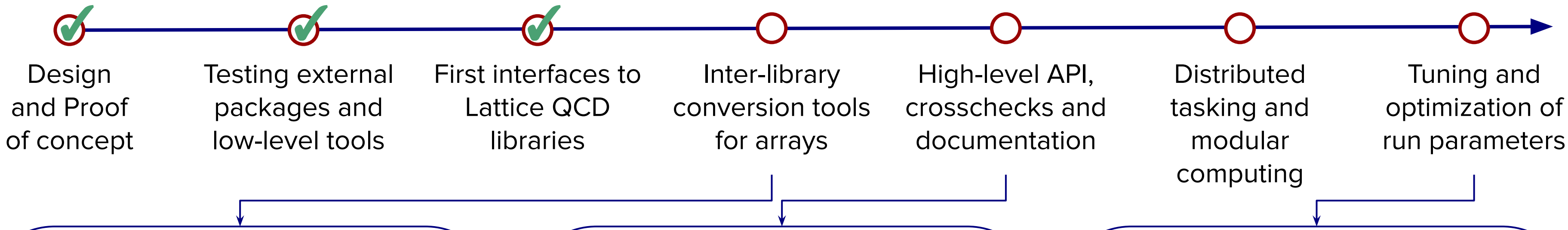
DDalphaAMG

```
>>> from lyncs_DDalphaAMG import Solver
>>> from lyncs_mpi import Client

>>> # Creating a client with 4 workers
>>> client = Client(num_workers = 4)
>>> comm = client.create_comm()
>>> procs = [2, 2, 1, 1]
>>> comm = comms.create_cart(procs)

>>> solver = Solver(
        global_lattice=[4, 4, 4, 4],
        comm=comm, kappa=0.125)

>>> # Reading the configurations
>>> conf = solver.read_configuration(
               "test/conf.random")
>>> plaq = solver.set_configuration(
               conf)
>>> print("Plaquette:", plaq)

>>> # Solution for a random vector
>>> vector = solver.random()
>>> result = solver.solve(vector)
```

QUDA

```
>>> import lyncs_quda as quda
>>> from mpi4py import MPI


>>> lattice = [4, 4, 4, 4]
>>> procs = [2, 2, 1, 1]
>>> comm = MPI.COMM_WORLD
>>> comm = comm.Create_cart(procs)

>>> gauge = quda.gauge(lattice=lattice,
               comm=comm, device="GPU")
>>> gauge.random()

>>> plaq = gauge.plaquette
>>> print("Plaquette:", plaq)

>>> vec = quda.spinor(lattice=lattice,
               comm=comm, device="GPU")
>>> vec.random()

>>> dirac = gauge.Dirac(kappa=0.125)
>>> mat = dirac.MMdag
>>> sol = mat.solve(vec, inv_type="CG")
>>> res = mat(sol) - vec
```

Roadmap



✓ Design and Proof of concept

✓ Testing external packages and low-level tools

✓ First interfaces to Lattice QCD libraries

○ Inter-library conversion tools for arrays

○ High-level API, crosschecks and documentation

○ Distributed tasking and modular computing

○ Tuning and optimization of run parameters

lyncs_field

- Array typing via metaclasses
- Seamlessly conversions

```
>>> from lyncs_field import Shape, CPU, GPU

>>> class XY_CPU(Shape("X","Y"), CPU):
>>>     pass

>>> class YX_GPU(Shape("Y","X"), GPU):
>>>     pass

>>> xy = XY_CPU(np.random.rand(8,8))
>>> yx = YX_GPU(xy)
```

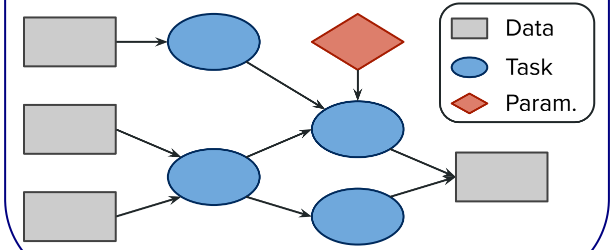
lyncs

- Standardize API for applications
- Libraries kernels as alternatives

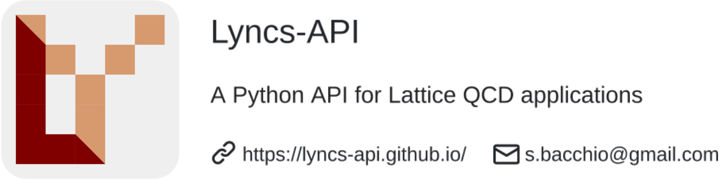
```
>>> @alternatives(
>>>     quda.plaquette,
>>>     tmLQCD.plaquette,
>>>     DDalphaAMG.plaquette,
>>>     Grid.plaquette,
>>>     ...,
>>> )
>>> def plaquette(conf):
>>>     # Python implementation
```

tuneit

- Tune of hyperparameter space
- Usage of computational graph



Conclusions



Lyncs-API

A Python API for Lattice QCD applications

🔗 https://lyncs-api.github.io/ ✉ s.bacchio@gmail.com

- The Lyncs-API is / wants to be ...
  - ➢ Fresh, modern, ambitious
  - ➢ A community-wise effort
  - ➢ Flexible, Portable, Modular
  - ➢ **Pythonic** and user-friendly
- Join the newsletter!
  - ➢ News, roundtable meetings, etc... https://groups.google.com/g/lyncs-api
- Do you want to be part of the effort?
  - ➢ Contact me! s.bacchio@gmail.com

Thank you for your attention!

Questions?

See you in Gather!