STORING REPRODUCIBLE RESULTS FROM COMPUTATIONAL EXPERIMENTS USING SCIENTIFIC PYTHON PACKAGES

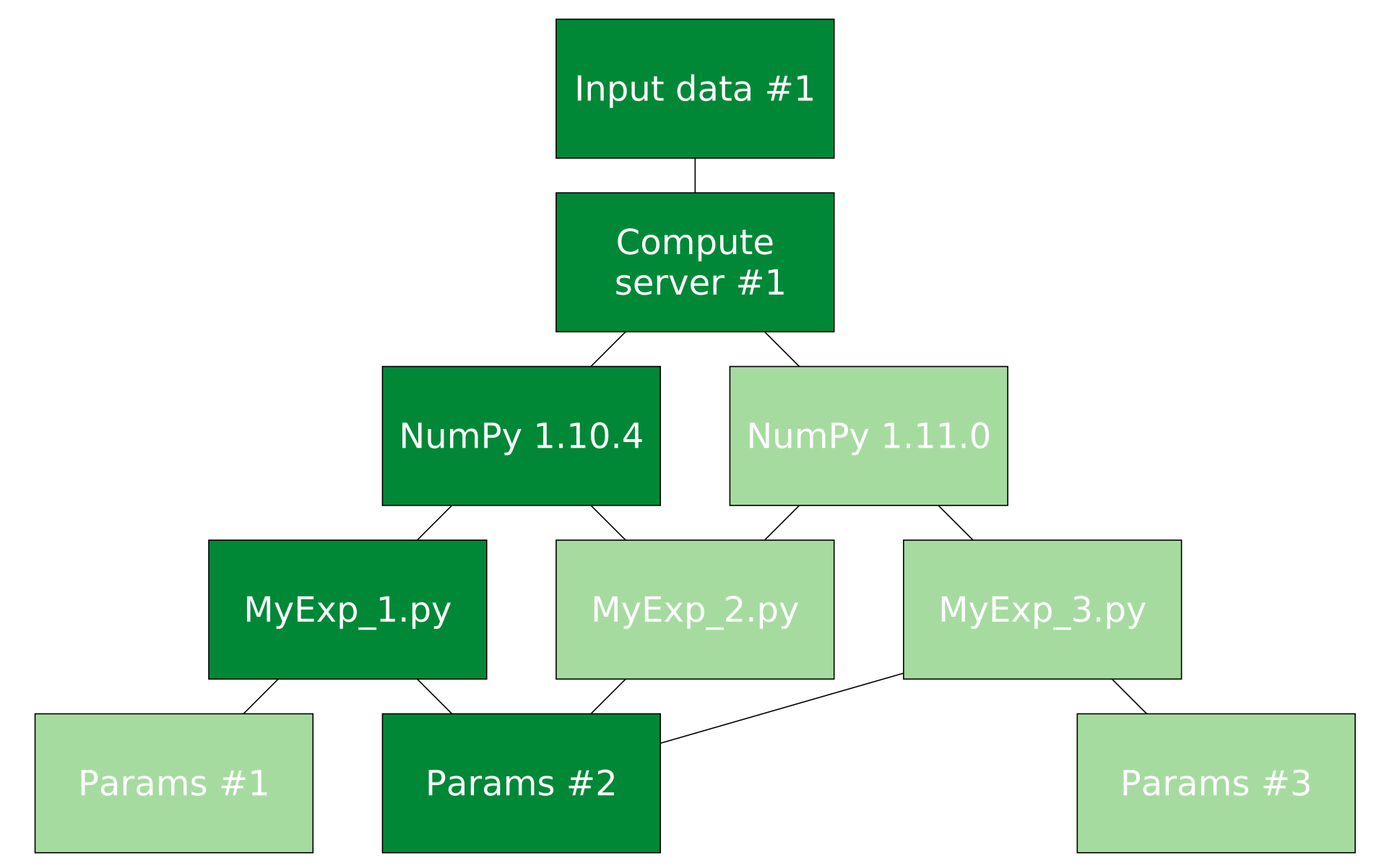
Christian Schou Oxvig, Thomas Arildsen, and Torben Larsen
Faculty of Engineering and Science, Department of Electronic Systems, Signal and Information Processing Section, Aalborg University, Denmark

Proposal

The Data Management Problem

Exactly how did I produce the computational results stored in this file?

Most data scientists and researchers have probably asked this question at some point. Making the results reproducible by tracking provenance and storing descriptive metadata helps answering it. This data management problem, illustrated in the below layered graph, amounts to tracking all choices for each layer, e.g. input data, HW platforms, SW libraries, algorithmic/experimental setup, and parameter values.



An important part of it is keeping track of the context of a single computational experiment, i.e. a single combination (as highlighted above). In this work we focus on solving that particular problem by storing metadata along with experimental results.

A Motivating Example

Consider a very simple Python script like

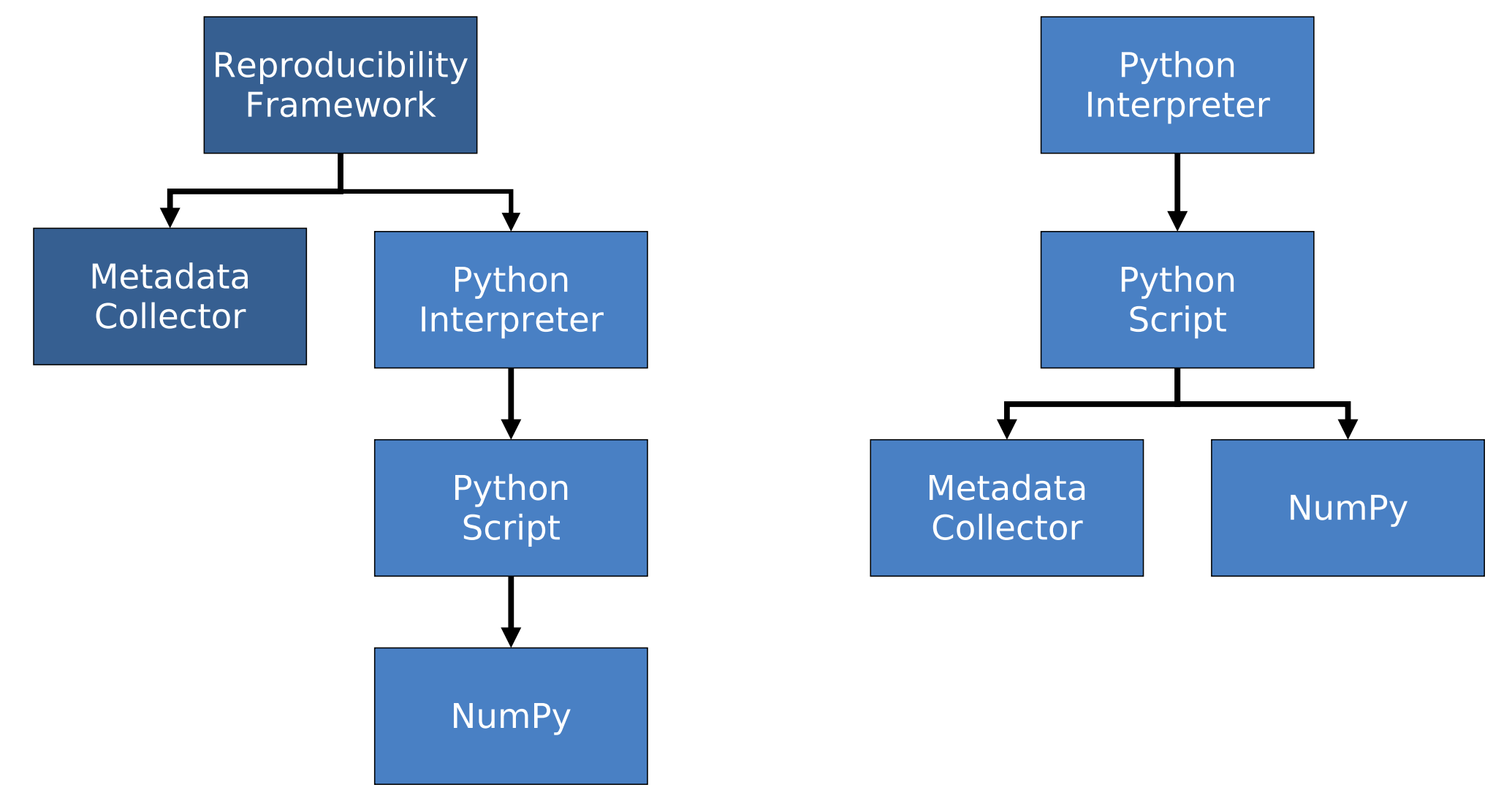
```
import sys
import numpy as np

ix = float(sys.argv[1])
result = np.arange(4, 10)[ix]
```

Obviously, the `result` depends on the input parameter used to determine the index. However, it is also highly dependent on the version of Numpy since indexing with floats was deprecated in Numpy 1.9.0 and will become an error in the near future. Furthermore, the `result` is likely also OS and HW platform dependent - at least to some degree.

Design Criteria

Existing reproducibilty aiding tools such as Sumatra, Madagascar, or ActivePapers generally wrap the entire computational experiment (left figure below). We propose using an importable library (right figure below) to best fit with the typical scientific Python workflow of defining the computational experiment in a single Python script which imports all external libraries.



The stored metadata must be tightly connected to the results and it must be easily inspected. Thus, our main ideas are

1. Create a Python importable reproducibility aiding library.
2. Store metadata along with results in a single file.
3. Store metadata in a format that is easily inspected.

Suggested Library Design

We suggest a solution based on

- ▶ Storing results along with data in an HDF5 database.
- ▶ Storing metadata as JSON serialized HDF5 string arrays.

and an interface to the scientific Python workflow similar to

```
import some_library
import some_other_library
import reproducibility_library

def some_func(...):
    ...

def run_my_experiment(...):
    ...

if __name__ == '__main__':
    reproducibility_library.store_metadata(...)
    run_my_experiment(...)
```

Implementation

`magni.reproducibilty` Interface

A reference implementation is available as part of the open source Magni Python package.

The main modules of `magni.reproducibility` are

- ▶ `data`: Extract metadata such as date, time, git revision, platform info, and source file info.
- ▶ `io`: Create new metadata annotated HDF5 databases or write and read metadata to/from an existing database.

Usage Examples

Extracting platform info using `magni.reproducibility.data`

```
>>> from pprint import pprint
>>> from magni import reproducibility as rep
>>> pprint(rep.data.get_platform_info())
{'libc': '["glibc", "2.2.5"]',
 'linux': '["debian", "jessie/sid", ""]',
 'mac_os': '["", ["", "", ""], ""]',
 'machine': '"x86_64"',
 'node': '"eagle1"',
 'processor': '"x86_64"',
 'python': '"3.5.1"',
 'release': '"3.16.0-46-generic"',
 'status': 'All OK',
 'system': '"Linux"',
 'version': '"#62~14.04.1-Ubuntu SMP ~"',
 'win32': '["", "", "", ""]'}
```

A Python script that uses `magni.reproducibility.io` to create an HDF5 database containing metadata

```
import tables
from magni import reproducibility as rep

def run_my_experiment(...):
    ...

def store_result(h5, result):
    ...

if __name__ == '__main__':
    hdf5_db = 'database.hdf5'
    rep.io.create_database(hdf5_db)
    result = run_my_experiment(...)
    with tables.File(hdf5_db, mode='a') as h5:
        store_result(h5, result)
```

- ▶ Extensive example at `doi:10.5278/VBN/MISC/MagniRE`

Perspective

Conclusions

We have found that

- ▶ Metadata should be stored along with computational results in an easily readable format in order to make the results reproducible.
- ▶ All necessary tools for making the results reproducible should be available as an importable package.
- ▶ JSON serialized arrays may be used as metadata storage format and an HDF5 database may be used as storage container.
- ▶ All of this is readily achievable using scientific Python packages.

Details on Magni

The Magni Python package is fully documented and comes with an extensive test suite. It has been developed using best practices for developing scientific software and all code has been reviewed by at least one other person than its author prior to its inclusion in Magni. All code adheres to the PEP8 style guide and no function or class has a cyclomatic complexity exceeding 10. The source code is under version control using Git and a continuous integration system based on Travis CI is in use for the git repository.

Resources

- ▶ Official releases: `doi:10.5278/VBN/MISC/Magni`
- ▶ Online documentation: `http://magni.readthedocs.io`
- ▶ GitHub repo: `https://github.com/SIP-AAU/`
- ▶ Software Metapaper: `doi:10.5334/jors.bk`

Acknowledgements

This work was supported in part by the Danish Council for Independent Research (DFF/FTP) under Project 1335-00278B/12-134971 and in part by the Danish e-Infrastructure Cooperation (DeIC) under Project DeIC2013.12.23.