Investigating interoperability of the LSST Data Management software stack with Astropy

Tim Jenness[a], James Bosch[b], Russell Owen[c], John Parejko[c], Jonathan Sick[a], John Swinbank[b], Miguel de Val-Borro[b], Gregory Dubois-Felsmann[d], K-T Lim[e], Robert H. Lupton[b], Pim Schellart[b], K. Simon Krughoff[c], and Erik J. Tollerud[f]

[a]LSST Project Management Office, Tucson, AZ, U.S.A.
[b]Princeton University, Princeton, NJ, U.S.A.
[c]University of Washington, Seattle, WA, U.S.A
[d]Infrared Processing and Analysis Center, California Institute of Technology, Pasadena, CA, U.S.A.
[e]SLAC National Laboratory, Menlo Park, CA, U.S.A.
[f]Space Telescope Science Institute, 3700 San Martin Dr, Baltimore, MD, 21218, USA

ABSTRACT

The Large Synoptic Survey Telescope (LSST) will be an 8.4 m optical survey telescope sited in Chile and capable of imaging the entire sky twice a week. The data rate of approximately 15 TB per night and the requirements to both issue alerts on transient sources within 60 seconds of observing and create annual data releases means that automated data management systems and data processing pipelines are a key deliverable of the LSST construction project. The LSST data management software has been in development since 2004 and is based on a C++ core with a Python control layer. The software consists of nearly a quarter of a million lines of code covering the system from fundamental WCS and table libraries to pipeline environments and distributed process execution.

The Astropy project began in 2011 as an attempt to bring together disparate open source Python projects and build a core standard infrastructure that can be used and built upon by the astronomy community. This project has been phenomenally successful in the years since it has begun and has grown to be the de facto standard for Python software in astronomy. Astropy brings with it considerable expectations from the community on how astronomy Python software should be developed and it is clear that by the time LSST is fully operational in the 2020s many of the prospective users of the LSST software stack will expect it to be fully interoperable with Astropy.

In this paper we describe the overlap between the LSST science pipeline software and Astropy software and investigate areas where the LSST software provides new functionality. We also discuss the possibilities of re-engineering the LSST science pipeline software to build upon Astropy, including the option of contributing affiliated packages.

Keywords: LSST, Astropy, Astronomy Software, Python, Code Reuse

1. INTRODUCTION

One major component of the Data Management system[1] for the Large Synoptic Survey Telescope (LSST)[2–4] is a pair of software pipelines capable of reducing LSST images to catalogs, one to produce alerts while observing and the other to produce data releases at regular intervals. The alert pipeline must be able to process data from the 3.2 gigapixel camera* and issue alerts on transient sources within 60 seconds of observing.[5] The data release pipeline must be able to efficiently process the 11 data releases that will be made during the ten year survey.

Further author information: (Send correspondence to T.J.)
T.J.: E-mail: tjenness@lsst.org

*This equates to approximately 15 TB per night. See http://lsst.org/scientists/keynumbers for additional numbers.

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 99130G · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2231313

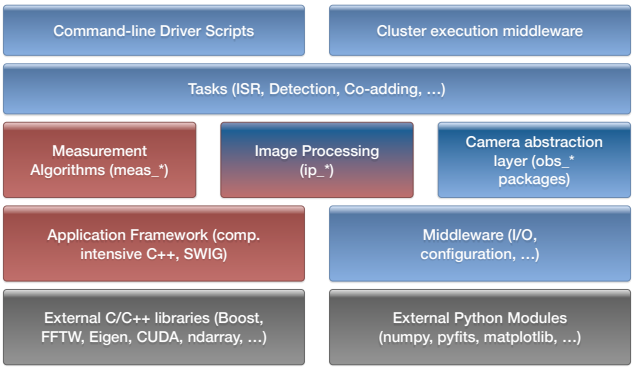

Figure 1. Structure of the LSST data management software. At the bottom there are external packages (black), on the lower left (red) are the computationally intensive packages that are mainly written in C++ with SWIG wrappers making the code available in Python, and the remainder (blue) are mostly Python.

In addition to efficiency, the LSST pipelines must also provide unprecedented levels of control over systematic errors across a broad range of scientific domains. Most consumers of LSST data will likely do no image processing of their own, and even those that do will rely heavily on catalogs generated by the LSST pipeline. Furthermore, much of LSST data analysis is expected to happen in LSST's "Level 3 Environment," a combination of hardware and software that will allow scientists to run their own code and customized versions of LSST pipeline algorithms on LSST's middleware and hardware. While algorithmically advanced pipelines and public catalog releases have become the norm for large surveys over the past 20 years, LSST's will be the first for which the software must be accessible to and usable by scientists outside the pipeline team.

The difficulty of sharing code between telescopes has long been a topic of discussion in the astronomy software community.[6–9] Historically there have been some successes in fundamental libraries, such as SOFA,[10] CFITSIO[11] and wcslib,[12] and in environments, such as IRAF,[13] AIPS[14] and Starlink.[15] Despite these, there is still a tendency for a new project to embark on the creation of a new environment or software infrastructure if there are slight requirements mismatches and no external political will to adopt a specific software suite. The growth in the Open Source software movement[16] and recent explosion in support tools, such as git and GitHub for code sharing,[17] makes the process of software reuse far easier than it has been in the past.

The LSST data processing pipeline software has been in development as an open source project since 2004[18–20] and during these 12 years of development much has changed in the Python, C++ and astronomy software world.

2. THE LSST SCIENCE PIPELINE SOFTWARE

As with other software developed in the early 2000s, such as CASA,[21] the LSST science pipeline software is based on a C++ core with a Python control layer, where for LSST the Python/C++ interface layer is handled via SWIG[22] bindings. The general layout of the code is shown in Fig. 1. At the time of writing the LSST science pipelines codebase is split between C++ and Python with of order 105,000 lines of Python and 125,000 lines of C++[†].

The software is spread among many different LSST root packages:

afw Provides core LSST utility classes including geometric shapes, images, source detection, convolution and high-level exposure objects.

daf The data access framework, also known as *the Data Butler*, provides an abstraction layer hiding data formats, data transport and locations from the user and providing them with standardized Python objects

[†]This total does not include comment or blank lines, nor does it include the 80,000 lines of Python and 1.6 million lines of C++ generated for the SWIG wrappers.

representing the data. It provides facilities for handling simple versioning and time-based data access with validity ranges.

ip Low-level image processing algorithms such as instrument signature removal and image differencing.

meas Algorithms focused on processing individual sources in an image, including PSF modeling, WCS fitting, and source detection, deblending, and measurement.

obs Observatory plugins to teach the system how to handle files from different instruments.

pipe_base Pipeline processing infrastructure. This allows discrete units of computation to be linked together to form pipelines.

pipe_tasks High-level algorithms, including initial exposure-level processing and calibration, image coaddition (with sophisticated artifact rejection and coaddition of PSF models), and consistent multi-passband processing of coadds to measure colors. The Task framework provides a way of packaging science algorithms so that they can be executed in a data-parallel fashion, with further extensions to allow MPI-based tightly-synchronized execution. The framework includes a Python-based configuration language that has default values, overrides from code as well as the end user, and hooks for validation of configured values.

The LSST codebase currently contains components that are not of short-term interest for Astropy integration but might become so in the longer run given related work by Astropy contributors in some of these areas.[23–25] These include the Data Butler and the Task framework. In terms of overlap with Astropy and affiliated packages the `afw` and `meas` packages are most relevant for immediate investigation as they provide some functionality that is of broader interest to the Astropy community but also have functionality that is also present within Astropy.

The algorithms in the LSST codebase are not yet at the level of scientific quality ultimately needed for LSST, but they are already being used to process data from the Hyper Suprime-Cam Survey on the Subaru telescope,[26,27] one of the major precursors surveys for LSST, and generally represent the current state-of-the-art.

3. ASTROPY

Python has become the dominant programming language for astronomy[28–32] and is now the default choice for new projects and for teaching. The Astropy[33] project grew out of a support mailing list in 2011 to become a very successful hub for the development of Python packages for astronomy. At the time of writing Astropy consists of approximately 105,000 lines of Python code. Astropy has two key approaches to drive adoption. The first is to place generic astronomy code into the *core* package that is brought in via `import astropy` to provide a solid, well-tested base for astronomy software development; the second is to support and encourage the concept of *affiliated* packages, which use the `astropy` package and follow the same philosophy on distribution and openness. Together these aim to provide a consistent environment of highly-usable, well-documented, easy-to-install code for astronomers using Python in their research.

4. SYNERGIES

The LSST project is constrained by the requirements of the funding agencies to deliver a data management system on schedule and on budget. Involvement with Astropy is justifiable if it helps the project improve schedule or performance or if it improves community usability while not impacting deliverables. It is also possible that making it easier for the community to adopt LSST algorithms can improve robustness of the code and find bugs earlier. The community will be encouraged to try out the LSST software sooner if it integrates well with Astropy. This will likely facilitate adoption of the Level 3 processing system during LSST operations, resulting in a system that is more scientifically productive.

The Astropy and LSST software projects are both attempting to write astronomical software in Python that will benefit the community. In this section we look at the areas in which code can be contributed by LSST to Astropy and where collaboration between LSST and Astropy can enhance both projects. In Section 5 we address the areas where LSST can benefit directly from improving interoperability with Astropy software.

4.1 Generalized World Coordinate Systems

The James Webb Space Telescope (JWST)[34] and LSST have requirements for accurate world coordinate conversions that cannot be represented in the current FITS World Coordinate Systems (FITS-WCS) framework[35–37] and none of the current extensions handling focal plane distortions[38,39] are suitably flexible. This lack of flexibility in FITS-WCS has been a long standing issue and the Starlink AST library[40] provides one solution to this problem that has been in use for nearly 20 years. For JWST a pure-Python equivalent, the Generalized World Coordinate Systems (GWCS) package, is in development[41] and is an Astropy affiliated package that builds on the `astropy.modeling` framework. LSST expects to need to perform world coordinate transformations in the C++ layer for performance reasons, and this may preclude direct adoption of GWCS by LSST, given the performance limitations of a pure-Python implementation. It would be beneficial to both parties if the future LSST WCS implementation and GWCS could interoperate easily. We are therefore investigating a text-based serialization of WCS that could be usable as an interchange format. AST currently supports a number of serialization schemes including multiple dialects of FITS headers, the International Virtual Observatory Alliance (IVOA)[‡] Space-Time Coordinate Linear String format (STC-S)[42] for region handling[43] and an internal text-based format. GWCS can serialize to FITS-WCS and also into YAML[44] using the Advanced Scientific Data Format (ASDF).[45] We have decided to investigate a common format that is based on YAML and the proposed new variant of STC, version 2.0. STC-2[46] is an evolution of STC that includes support for chainable mappings and seems like it could be a suitable starting point for this investigation. Both the Astropy and IVOA communities would benefit if such a standard could be established and it could be a significant factor in bringing together the competing distortion models currently in use in the FITS community.

4.2 Region handling

The handling of regions such as circles and polygons is not yet supported in Astropy core. The Astropy affiliated `pyregions` package does have code for handling region files from the DS9 application,[47] with some discussions concerning adopting the IVOA STC standard[42] as a region serialization format. In addition, the `photutils` affiliated package has some basic support for rasterizing analytic regions in pixel coordinates for photometry. A new affiliated package[§], hopes to unify this functionality and provide a consistent interface for all region operations in Astropy, with the expectation that this functionality will move to Astropy core once the interface stabilizes.

LSST's codebase actually includes two distinct geometry libraries: `afw.geom` includes points, boxes, polygons and ellipses for Cartesian coordinate systems, while `sphgeom` provides similar geometric primitives in spherical polar coordinates. These do not currently interoperate smoothly, reflecting the fact that they have thus far been used in completely different components of the codebase. Integrating these libraries will clearly be important for LSST in the future, suggesting that now is an appropriate time for LSST and Astropy to collaborate on a new unified regions library. LSST could provide mature implementations of individual geometry classes to such a collaboration, while Astropy brings serialization and visualization code as well as a broader sense of the requirements for a unified library. A major challenge for a jointly-developed library is reconciling LSST's need for C++ geometry classes (with Python bindings) with Astropy's preference for pure Python, and when necessary Cython (see Sec. 5.4), for easier installation and maintenance.

STC-S is already being used to define sky regions in the VO community and is also understood by the GAIA visualization tool[48] and Starlink software.[43,49] If Astropy and LSST cannot share sky-to-pixel region handling code directly, there is a possibility that STC could be adopted to allow interoperability.

Most of the above work focuses on the lower half of Fig. 2, but Astropy's unified region plans also include rasterization to images (upper right corner). Pixelizing LSST geometries typically involves yet another class: `afw.detection.Footprint` (Fig. 3). `Footprint` is a compact serialized representation of a mask, listing the pixel indices that contribute to the shape in rasterized order, using a run-length encoding algorithm to conserve space. A `HeavyFootprint` is a `Footprint` that includes the pixel values themselves. `Footprint` is a critical component of LSST's detection algorithm, as it is used to represent an above-threshold region in an image

[‡] http://www.ivoa.net

[§] https://github.com/astropy/regions

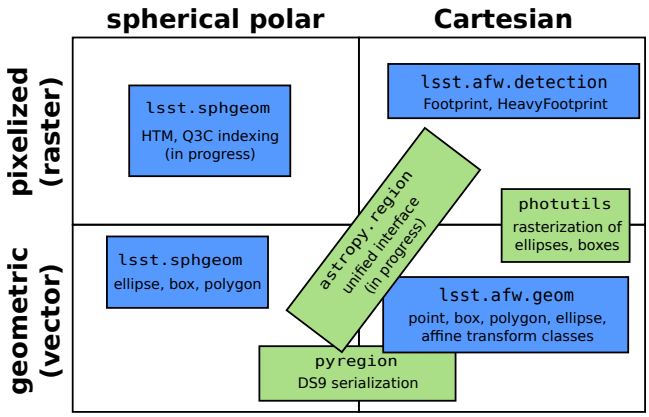

Figure 2. Four ways of representing regions, and the approximate domains of the LSST (blue) and Astropy (green) packages for manipulating them.

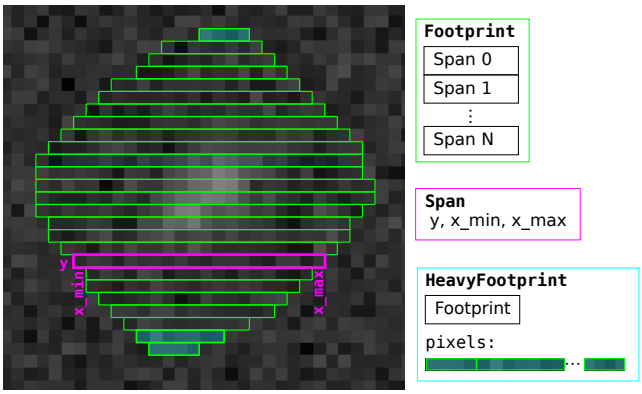

Figure 3. LSST's `Footprint` data structure. A `Footprint` describes a pixelized region as a sequence of `Spans`, each of which holds the y position and minimum and maximum x positions. A `HeavyFootprint` combines a `Footprint` with the values of the pixels it covers by simply storing the pixels of each span back-to-back.

(similar to SExtractor's "segmentation map"). Similarly, `HeavyFootprint` plays an important role in deblending and source measurement, where it is used to define the (fractional) pixel values that define a deblended object. `Footprint` and `HeavyFootprint` represent a convenient intermediate between geometries and images, but their current interfaces reflect a C++-centric design (though they are also available in Python, of course). A Python-centric redesign of these interfaces as part of LSST's involvement in Astropy's unified region library could yield valuable dividends, particularly in providing natural and efficient vectorization over irregular regions in images.

LSST does have some C++ code for sky pixelization (upper left of Fig. 2), supporting schemes such as HTM[50] and Q3C.[51] This code is currently unavailable in Python, though this is expected to change in the future. There are currently no plans to handle Multi-Order Coverage maps (MOC)[52,53] or other HEALPix approaches.[54] As the Astropy regions package develops, it may be reasonable for LSST to also donate this code to Astropy.

4.3 LSST Measurement Algorithms

There are a number of high-level algorithms in LSST that could be made available as Astropy affiliated packages,[55] including all of the steps necessary to build catalogs from images and combine images to build coadds. These provide essentially the same functionality as the popular Astromatic suite,[56] with an interface that is already more Python-friendly than Astromatic's combination of command-line tools and configuration files. The algorithms are also typically more sophisticated than their Astromatic equivalents (and correspondingly are a bit harder to configure). Repackaging just the source detection, deblending, and measurement algorithms (roughly equivalent

to Astromatic's ubiquitous SExtractor) as Astropy affiliated packages would make a very important (and very useful) subset of LSST's algorithms more readily available to a broader community.

Unfortunately, it is difficult to disentangle these algorithms from the lower-level LSST primitive classes they build on, and hence an independent Astropy affiliated package with this functionality will likely depend on not just integrating LSST's region, image, and table classes with Astropy, but spinning these off as Astropy affiliated (or perhaps core) packages as well. A more feasible approach to making these algorithms available to the Astropy community may be to focus on making all of LSST's packaging more Astropy-friendly (see Section 5.1), rather than repackaging algorithms as Astropy affiliates.

Regardless of the packaging, allowing users to seamlessly call these algorithms from code that exists in the Astropy ecosystem is one of the more important goals in Astropy/LSST integration; it would provide the Astropy community with high-quality implementations of important astronomical algorithms that are not currently available in Python, while providing LSST with feedback and testing on some of its most important components. Unfortunately, due to its dependency on virtually every other integration effort discussed in this paper, it is likely not a short-term goal.

4.4 NDData

The Astropy `NDData` class is an implementation of the common astronomy data structure[57] of a data array, mask and uncertainty with attached metadata. LSST has an `Exposure` class (in C++ it is templated on the data type of the image) represented by a `MaskedImage` and structured metadata. The class layout of an `Exposure` is shown in Fig. 4. It would be beneficial if an LSST `Exposure` could implement the `NDData` interface. The mask in an `Exposure` is a bit mask, allowing each bit to represent a different reason for masking, and not a Boolean mask. Uncertainties in an `Exposure` are always represented as variances (although covariance has been discussed). These are supportable as sub-classes and variants of them are already available in the Astropy ecosystem. One item that is currently missing from the Astropy arena is the notion of a pixel origin. In LSST the pixel origin can be used to specify the coordinates of the data array relative to a parent data array, such as allowing the coordinates of image cutouts to reference where they came from in the parent. LSST nomenclature is to use *PARENT* to indicate the abstract pixel coordinate and *LOCAL* to indicate that the coordinates are relative to the 0-based origin of the data array itself.[¶] This attachment of a data array to a parent pixel coordinate system is a critical item and we plan to work with Astropy on making this functionality available to `NDData` users.

5. SUGGESTED MODIFICATIONS TO LSST

In addition to working with the Astropy community, there is also the question of how integrated the LSST software system should be with Astropy. This covers the adoption of distribution standards and use of Astropy classes in our codebase. These changes will result in a more supportable and sustainable codebase allowing LSST to focus on core deliverables without having to support general infrastructure. In this section we explore the areas where the LSST software could benefit from adoption of Astropy.

5.1 Packaging

The LSST data management software uses the EUPS tool[58] to manage versions and dependencies. It is used to ensure that the versions of all packages used for a particular processing run can be assigned a single number and also to track the packages that were part of a single build. Development versions can easily be included along with versions from builds done in the past. This can significantly simplify debugging cycles and aid with provenance tracking but it is a complex environment with a steep learning curve and most users of LSST software outside of the data center and LSST developer community do not need this power. EUPS can be thought of as a more powerful Python virtual environment where each package can have a specific version enabled to allow mixing and matching of multiple packages, rather than just having entire installations in different virtual environments.

LSST packages are built using the SCons tool.[59] SCons is an alternative tool to Make and CMake that is written entirely in Python. This allows developers to use Python for their library code and build system whilst

[¶]The Starlink nomenclature[57] is *PIXEL* and *GRID* where *GRID* follows the FITS convention of starting at 1.

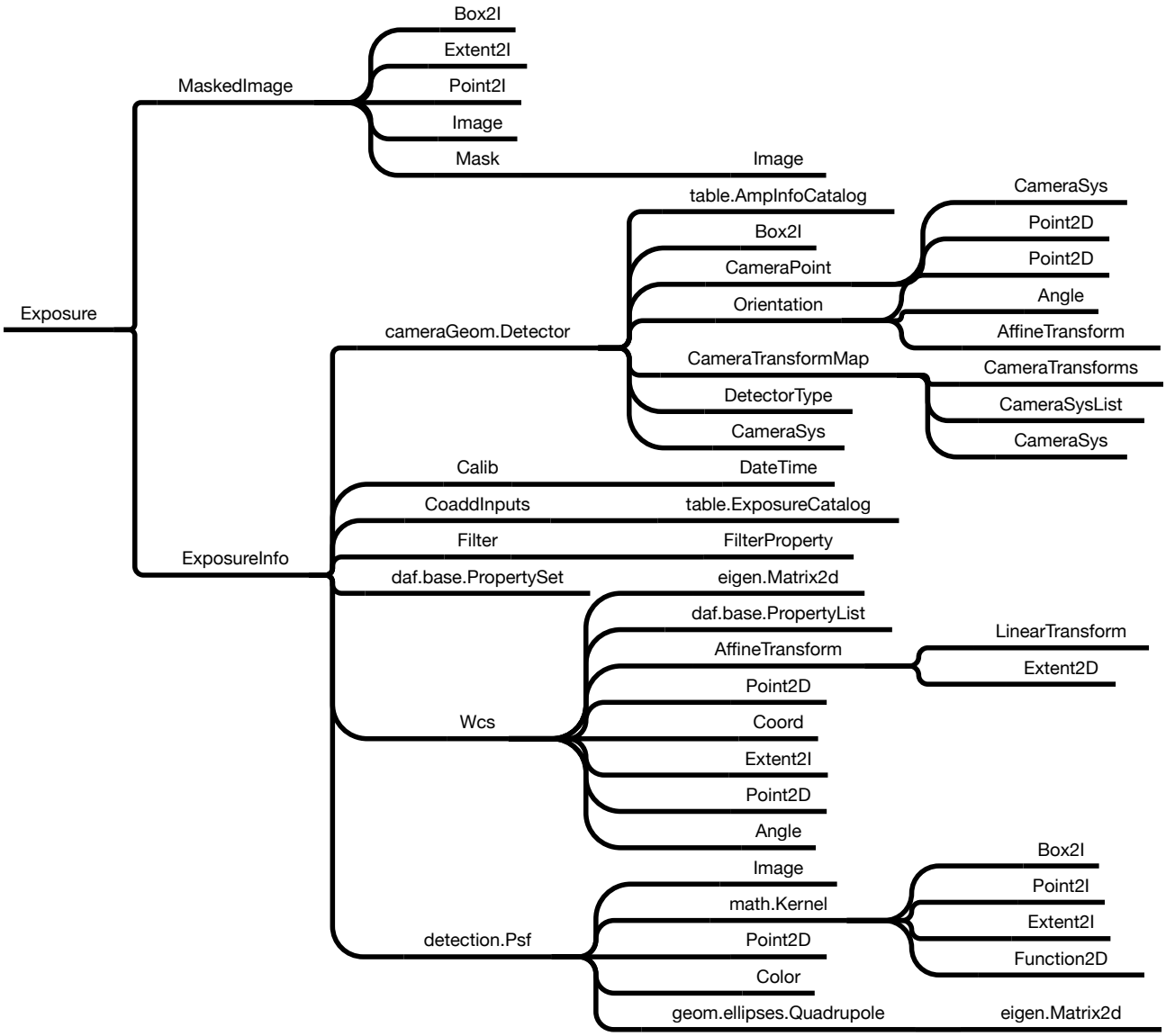

Figure 4. The LSST classes that are used to define a `lsst.afw.image.Exposure`.

making use of the powerful source code dependency tracking provided by SCons to prevent wasteful recompilation of files that have not been affected by local edits.

The majority of the Python community have adopted a build system based around the convention of a `setup.py` file that can be executed to do the build. This file uses `distutils` or `setuptools` to determine what should be built and can handle most cases where external C/C++ code should be compiled into the Python package. There is a strong expectation that any packages made available to the Astropy community should follow the standard Python build convention of supporting `python setup.py build` and this is a requirement if we wish to make some LSST packages Astropy affiliated packages. This also allows packages to be distributed on the Python Package Index (PyPI[‖]) providing the ability for a general user to use `pip install lsst_apps` to install the LSST software.

[‖] https://pypi.python.org

As a proof of concept we have demonstrated that an LSST package using the standard LSST directory layout can be built using `setuptools`. EUPS is used to determine dependencies and this information can be converted to a standard `setuptools` dependency request. To get this working on PyPI it is necessary either to generate the `setup.py` using a SCons target or else use EUPS directly in the `setup.py`. The latter would require that EUPS is also available from PyPI (it currently is not) but might be less prone to errors.

5.2 Astronomical Coordinates

The LSST `afw.coord` package provides coordinate conversions between ICRS, FK5, Galactic, ecliptic and topocentric coordinate systems. It is written in C++ and has no external dependencies on standard coordinate libraries. The `astropy.coordinates` package provides a very usable Python API on top of the standard ERFA astrometry C library.[**] It supports all the coordinate systems provided by the LSST package and many more including FK4.

The LSST C++ code does not use any system other than ICRS; therefore we are proposing to remove this package and replace it with a simple spherical coordinates class. All the Python code will then be modified to use `astropy.coordinates` and at the Python/C++ boundary coordinates will be converted to/from ICRS.

5.3 Tables

LSST and Astropy both have table libraries, but these have different data models and hence different strengths. LSST's are better for row-based access and provide a limited ORM layer that's valuable for pipeline use when the columns are known in advance and data are appended to the table, while Astropy's excel at column-based access and use in interactive analysis. Furthermore, Astropy tables now support export to pandas DataFrames,[60] opening up a richer environment for query-based analysis. However, the LSST and Astropy table data models are similar enough to allow read-write, memory-sharing Astropy views to LSST tables to be constructed in Python, and a proof-of-concept implementation of this functionality has already been developed. As this view interface improves, consumers of table-based LSST data products will be able to simply use Astropy tools for analysis, removing much of the need for LSST to develop its own support for many table operations.

5.4 C++ Bindings

The LSST Data Management team adopted SWIG as the standard method for providing a Python interface to the C++ code. Astropy has adopted Cython[61] when it is required to call C code. The Astropy core libraries currently do not call C++ and while Cython support for C++ exists, this support is relatively new and somewhat experimental. The Astropy project has no hard rules concerning how C/C++ code should be included in an affiliated package but has a strong preference for Cython. LSST is not averse to switching away from SWIG (which results in massive code bloat and extended compile times as currently implemented) and has agreed to investigate Cython[62] and pybind11.[63††] Any move away from SWIG can only be done incrementally and so it must be possible for a C++ object produced from SWIG to be passed into a C++ interface created with the new approach. This has been demonstrated successfully as part of the Cython and pybind11 investigations.

5.5 Quantities and Units

With the exception of an angle class, LSST does not use any unit conversion machinery. Astropy's `Quantity`, a value with an associated unit, is a very powerful class that can remove an entire category of bugs from scientific software by ensuring that mathematical operations between quantities use compatible units. The units package in Astropy has been adapted from the units classes in the `pynbody` analysis module.[64] For heavily-used public APIs it would seem to be clear that arguments should be quantities wherever possible. The question for LSST is how far down in the stack quantities should be used given the performance hit associated with quantities over plain variables. Additionally, how much more complicated does the C++python interface become if SWIG has to be taught how to convert quantities to the correct form before passing the value to the C++. This is especially critical for large arrays, supplied externally, being passed through where we do not want to do a copy operation before entering C++ and also do not want to normalize units for millions of data points at that time. Large

[**] ERFA is a relicensed version of the standard SOFA library.[10]

[††] http://pybind11.readthedocs.org/

data arrays and, in particular, table columns returned from C++ layers would benefit from being returned as quantities, as this gives much more flexibility to users of the data and minimizes the risk of misinterpretation of values if units are later changed for a column or the user relies entirely on documentation to interpret values. The proposal is that LSST software adopts quantities on a case-by-case basis, and its developers will be continuously on the lookout for any performance impact associated with quantities.

5.6 Time

LSST uses a C++ time class, `lsst::daf::base::DateTime` to provide basic time conversion functionality for Modified Julian Dates and UTC. The Astropy `astropy.time` package has far more functionality and adopting it in Python LSST code is quite attractive. The complication is in how to make a Python view of a `daf_base` time object look like an `astropy.time` and vice versa when crossing the Python/C++ boundary. One way forward may be to implement the C++ class using `astropy.time` objects. An experimental implementation does exist demonstrating that this can be made to work, Python calling C++ that calls Python can be relatively brittle and can break if there is a Python shared library mismatch. Rather than adding this complexity to the C++ codebase it is likely we will adopt a solution similar to that described in § 5.2 whereby we simplify the C++ class so it is tracking nanoseconds in epoch and then convert it to `astropy.time` in the Python layer.

6. PROPOSAL

The concrete proposals from this investigation are therefore:

- Change EUPS packaging to use a `setup.py` and upload it to PyPI.
- Investigate the addition of a `setup.py` for standard LSST Python packages in addition to our current Scons-based build system.
- Support views of LSST tables as Astropy tables.
- Determine whether STC-2, in YAML form, is a suitable basis for WCS interoperability.
- Remove LSST astronomical coordinate classes and replace with Astropy coordinates in Python and a generic spherical coordinates class in C++.
- Simplify the LSST C++ date time classes and use switch to using `astropy.time` in Python code.
- Investigate migration away from SWIG for C++ bindings.
- Contribute the LSST `Footprint` code to Astropy.
- Create an Astropy affiliated package from the `afw.detection` package.
- Investigate rationalizing of LSST and Astropy spherical geometry region handling.
- Work with the Astropy project to add required functionality such as pixel origins to `NDData`.
- Modify `Exposure` to implement an `NDData`-like interface.
- LSST will allow Astropy quantity objects to be used in public interfaces on a case by case basis.

With these changes implemented, LSST will have a solid base from which to assess whether it is feasible to migrate more LSST code to the Astropy community as affiliated packages. It is probable that LSST will start to migrate the existing codebase entirely to Astropy classes where appropriate classes exist. In particular `astropy.coordinates` could be adopted fairly rapidly. Switching time libraries may require significantly more thought.

7. CONCLUSION

The LSST project has a codebase that has developed over a number of years and has been developed independently of the wider Python astronomy software community. We have looked at the Astropy package and worked with the Astropy team to understand areas where synergies exist and where LSST can contribute more fully to the community without delaying our primary deliverables. The proposals outlined in this document go a long way towards integrating LSST into the wider software community and should bring key benefits to LSST and research astronomers. Work on these changes has already begun and the intent is to allocate effort to this migration in the second half of 2016 and prioritize the integration of `astropy` into the LSST codebase.

ACKNOWLEDGMENTS

We thank those who attended the LSST/Astropy Summit at the University of Washington in March 2016 that forms the basis for the approaches outlined in this paper. We thank Chris Walter, Ben Emmons, Adam Ginsburg and Brigitta Sipocz for comments on the draft manuscript. This material is based upon work supported in part by the National Science Foundation through Cooperative Support Agreement (CSA) Award No. AST–1227061 under Governing Cooperative Agreement 1258333 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE–AC02–76SF00515 with the SLAC National Accelerator Laboratory. Additional LSST funding comes from private donations, grants to universities, and in-kind support from LSSTC Institutional Members.

REFERENCES

[1] Juric, M. et al., "The LSST Data Management System," in [*Astronomical Data Analysis Software & Systems XXV*], Lorente, N. P. F. and Shortridge, K., eds., *ASP Conf. Ser.* **in press**, arXiv:1512.07914, ASP, San Francisco (2016).

[2] Ivezić, Ž. et al., "LSST: from Science Drivers to Reference Design and Anticipated Data Products," *ArXiv e-prints* **arXiv:0805.2366** (May 2008).

[3] Kahn, S., "Final design of the Large Synoptic Survey Telescope," in [*Ground-Based and Airborne Telescopes IV*], Hall, H. J., Gilmozzi, R., and Marshall, H. K., eds., *Proc. SPIE* **9906**, in press (2016).

[4] Gressler, W., DeVries, J., Hileman, E., Neill, D. R., Sebag, J., Wiecha, O., Andrew, J., Lotz, P., and Schoening, W., "LSST Telescope and site status," in [*Ground-based and Airborne Telescopes V*], Stepp, L. M., Gilmozzi, R., and Hall, H. J., eds., *Proc. SPIE* **9145**, 91451A (July 2014).

[5] Kantor, J., "Transient Alerts in LSST," in [*The Third Hot-wiring the Transient Universe Workshop*], Wozniak, P. R., Graham, M. J., Mahabal, A. A., and Seaman, R., eds., 19–26 (2014).

[6] Mandel, E. and Murray, S. S., "Other People's Software," in [*Astronomical Data Analysis Software and Systems VII*], Albrecht, R., Hook, R. N., and Bushouse, H. A., eds., *ASP Conf. Ser.* **145**, 142 (1998).

[7] Economou, F., Bridger, A., Wright, G. S., Jenness, T., Currie, M. J., and Adamson, A., "ORAC-DR: Pipelining With Other People's Code," in [*Astronomical Data Analysis Software and Systems VIII*], Mehringer, D. M., Plante, R. L., and Roberts, D. A., eds., *ASP Conf. Ser.* **172**, 11 (1999).

[8] Shortridge, K., "Astronomical Software Strategies," in [*Organizations and Strategies in Astronomy, Vol. II*], A., H., ed., *Astrophysics and Space Science Library* **266**, 163, Kluwer Academic Publishers (2001).

[9] Economou, F., Jenness, T., and Rees, N. P., "Sharing code and support between heterogeneous telescopes: the UKIRT and JCMT joint software projects," in [*Observatory Operations to Optimize Scientific Return III*], Quinn, P. J., ed., *Proc. SPIE* **4844**, 321–330 (Dec. 2002).

[10] Hohenkerk, C., "Standards of Fundamental Astronomy," *Scholarpedia* **6**, 11404 (2011).

[11] Pence, W., "CFITSIO, v2.0: A New Full-Featured Data Interface," in [*Astronomical Data Analysis Software and Systems VIII*], Mehringer, D. M., Plante, R. L., and Roberts, D. A., eds., *ASP Conf. Ser.* **172**, 487 (1999).

[12] Calabretta, M. R., "Wcslib and Pgsbox." Astrophysics Source Code Library (Aug. 2011). ascl:1108.003.

[13] Tody, D., "The IRAF Data Reduction and Analysis System," in [*Instrumentation in astronomy VI*], Crawford, D. L., ed., *Proc. SPIE* **627**, 733 (Jan. 1986).

[14] van Moorsel, G., Kemball, A., and Greisen, E., "AIPS Developments in the Nineties," in [*Astronomical Data Analysis Software and Systems V*], Jacoby, G. H. and Barnes, J., eds., *ASP Conf. Ser.* **101**, 37 (1996).
[15] Disney, M. J. and Wallace, P. T., "STARLINK," *QJRAS* **23**, 485 (Dec. 1982).
[16] DiBona, C., Cooper, D., and Stone, M., eds., [*Open Sources 2.0: The Continuing Evolution*], O'Reilly (2006). ISBN 0-596-00802-3.
[17] Lima, A., Rossi, L., and Musolesi, M., "Coding Together at Scale: GitHub as a Collaborative Social Network," in [*Proceedings of the Eigth International Conference on Weblogs and Social Media*], **arXiv:1407.2535**, AAAI Press (2014).
[18] Axelrod, T., Connolly, A., Ivezic, Z., Kantor, J., Lupton, R., Plante, R., Stubbs, C., and Wittman, D., "The LSST Data Processing Pipeline," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **36**, #108.11 (Dec. 2004).
[19] Axelrod, T., Kantor, J., Lupton, R. H., and Pierfederici, F., "An open source application framework for astronomical imaging pipelines," in [*Software and Cyberinfrastructure for Astronomy*], Radziwill, N. M. and Bridger, A., eds., *Proc. SPIE* **7740**, 774015 (July 2010).
[20] Jenness, T., "The LSST Data Processing Software Stack: Summer 2015 Release," in [*Astronomical Data Analysis Software & Systems XXV*], Lorente, N. P. F. and Shortridge, K., eds., *ASP Conf. Ser.* **in press**, arXiv:1511.06790, ASP, San Francisco (2016).
[21] Petry, D. and CASA Development Team, "Analysing ALMA Data with CASA," in [*Astronomical Data Analysis Software and Systems XXI*], Ballester, P., Egret, D., and Lorente, N. P. F., eds., *ASP Conf. Ser.* **461**, 849 (Sept. 2012).
[22] Beazley, D. M., "Automated scientific software scripting with SWIG," *Future Generation Computer Systems* **19**(5), 599–609 (2003).
[23] Turner, J. E. H., "Streamlining an IRAF data reduction process with AstroPy and NDMapper," in [*Python in Astronomy 2016*], (Mar. 2016). doi:10.5281/zenodo.50991.
[24] Bushouse, H., Droettboom, M., and Greenfield, P., "The JWST Data Calibration Pipeline," in [*Astronomical Data Analysis Software and Systems XXV*], Lorente, N. P. F. and Shortridge, K., eds., *ASP Conf. Ser.*, in press, ASP, San Francisco (2016).
[25] Bushouse, H., Droettboom, M., and Greenfield, P., "The James Webb Space Telescope Data Calibration Pipeline," in [*Proceedings of the 14th Python in Science Conference*], Huff, K. and Bergstra, J., eds., 44 (July 2015).
[26] Miyazaki, S. et al., "Hyper Suprime-Cam," in [*Ground-based and Airborne Instrumentation for Astronomy IV*], *Proc. SPIE* **8446**, 84460Z (Sept. 2012).
[27] Matsuoka, Y. et al., "Subaru high-z exploration of low-luminosity quasars (SHELLQs). I. Discovery of 15 quasars and bright galaxies at $5.7 < z < 6.9$," *ArXiv e-prints* **arXiv:1603.02281** (Mar. 2016).
[28] Greenfield, P. and White, R. L., "A New CL for IRAF Based On Python," in [*Astronomical Data Analysis Software and Systems IX*], Manset, N., Veillet, C., and Crabtree, D., eds., *ASP Conf. Ser.* **216**, 59 (2000).
[29] Hook, R. N., Maisala, S., Oittinen, T., Ullgren, M., Vasko, K., Savolainen, V., Lindroos, J., Anttila, M., Solin, O., Møller, P. M., Banse, K., and Peron, M., "PyMidas–A Python Interface to ESO-MIDAS," in [*Astronomical Data Analysis Software and Systems XV*], Gabriel, C., Arviset, C., Ponz, D., and Enrique, S., eds., *ASP Conf. Ser.* **351**, 343 (July 2006).
[30] Kettenis, M., van Langevelde, H. J., Reynolds, C., and Cotton, B., "ParselTongue: AIPS Talking Python," in [*Astronomical Data Analysis Software and Systems XV*], Gabriel, C., Arviset, C., Ponz, D., and Enrique, S., eds., *ASP Conf. Ser.* **351**, 497 (July 2006).
[31] Greenfield, P., "What Python Can Do for Astronomy," in [*Astronomical Data Analysis Software and Systems XX*], Evans, I. N., Accomazzi, A., Mink, D. J., and Rots, A. H., eds., *ASP Conf. Ser.* **442**, 425 (July 2011).
[32] Jeschke, E., Inagaki, T., and Kackley, R., "A next-generation open-source toolkit for FITS file image viewing," in [*Software and Cyberinfrastructure for Astronomy II*], Radziwill, N. M. and Chiozzi, G., eds., *Proc. SPIE* **8451**, 845102 (Sept. 2012).
[33] Astropy Collaboration, "Astropy: A community Python package for astronomy," *Astronomy & Astrophysics* **558**, A33 (Oct. 2013).
[34] Gardner, J. P. et al., "The James Webb Space Telescope," *Space Science Reviews* **123**, 485–606 (Apr. 2006).

[35] Hack, W. J., Dencheva, N., and Fruchter, A. S., "DrizzlePac: Managing Multi-component WCS Solutions for HST Data," in [*Astronomical Data Analysis Software and Systems XXII*], Friedel, D. N., ed., *ASP Conf. Ser.* **475**, 49 (Oct. 2013).

[36] Calabretta, M. R. and Greisen, E. W., "Representations of celestial coordinates in FITS," *Astronomy & Astrophysics* **395**, 1077–1122 (Dec. 2002).

[37] Thomas, B., Jenness, T., Economou, F., et al., "Learning from FITS: Limitations in use in modern astronomical research," *Astronomy and Computing* **12**, 133–145 (Sept. 2015).

[38] Shupe, D. L., Laher, R. R., Storrie-Lombardi, L., Surace, J., Grillmair, C., Levitan, D., and Sesar, B., "More flexibility in representing geometric distortion in astronomical images," in [*Software and Cyberinfrastructure for Astronomy II*], *Proc. SPIE* **8451**, 84511M (Sept. 2012).

[39] Valdes, F., "TPV convention for Representing FITS Image Distortions," (2012). http://fits.gsfc.nasa.gov/registry/tpvwcs.html.

[40] Berry, D. S., Warren-Smith, R. F., and Jenness, T., "AST: A library for modelling and manipulating coordinate systems," *Astronomy and Computing* **15**, 33 – 49 (2016).

[41] Dencheva, N., Greenfield, P., and Droettboom, M., "GWCS - A Library for Managing World Coordinate Systems," in [*Astronomical Data Analysis Software and Systems XXV*], Lorente, N. P. F. and Shortridge, K., eds., *ASP Conf. Ser.*, in press, ASP, San Francisco (2016).

[42] Rots, A. H., "Space-Time Coordinate Metadata for the Virtual Observatory Version 1.33." IVOA Recommendation (Oct. 2007). arXiv:1110.0504.

[43] Berry, D. and Draper, P., "Using the AST Library to Create and Use STC-S Region Descriptions," in [*Astronomical Data Analysis Software and Systems XIX*], Mizumoto, Y., Morita, K.-I., and Ohishi, M., eds., *ASP Conf. Ser.* **434**, 213 (Dec. 2010).

[44] Ben-Kiki, O., Evans, C., and döt Net, I., "YAML Ain't Markup Language (YAML) Version 1.2," (Oct. 2009). http://www.yaml.org/spec/1.2/spec.html (retrieved May 2016).

[45] Greenfield, P., Droettboom, M., and Bray, E., "ASDF: A new data format for astronomy," *Astronomy and Computing* **12**, 240–251 (Sept. 2015).

[46] Rots, A., "STC-2 Design and Development Status," in [*IVOA Interop, Sesto*], (June 2015). http://wiki.ivoa.net/internal/IVOA/InterOpJune2015DM/STC2.pdf.

[47] Joye, W. A. and Mandel, E., "The Development of SAOImage DS9: Lessons Learned from a Small but Successful Software Project," in [*Astronomical Data Analysis Software and Systems XIV*], Shopbell, P., Britton, M., and Ebert, R., eds., *ASP Conf. Ser.* **347**, 110 (Dec. 2005).

[48] Draper, P. W., Berry, D. S., Jenness, T., and Economou, F., "GAIA – Version 4," in [*Astronomical Data Analysis Software and Systems XVIII*], Bohlender, D. A., Durand, D., and Dowler, P., eds., *ASP Conf. Ser.* **411**, 575 (Sept. 2009).

[49] Currie, M. J., Berry, D. S., Jenness, T., Gibb, A. G., Bell, G. S., and Draper, P. W., "Starlink Software in 2013," in [*Astronomical Data Analysis Software and Systems XXIII*], Manset, N. and Forshay, P., eds., *ASP Conf. Ser.* **485**, 391 (May 2014).

[50] Kunszt, P. Z., Szalay, A. S., and Thakar, A. R., "The Hierarchical Triangular Mesh," in [*Mining the Sky*], Banday, A. J., Zaroubi, S., and Bartelmann, M., eds., 631 (2001).

[51] Koposov, S. and Bartunov, O., "Q3C, Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL," in [*Astronomical Data Analysis Software and Systems XV*], Gabriel, C., Arviset, C., Ponz, D., and Enrique, S., eds., *ASP Conf. Ser.* **351**, 735 (July 2006).

[52] Fernique, P. and Boch, T., "Easy Comparison of Dataset Footprints with MOC," in [*Astronomical Data Analysis Software and Systems XXI*], Ballester, P., Egret, D., and Lorente, N. P. F., eds., *ASP Conf. Ser.* **461**, 347 (Sept. 2012).

[53] Fernique, P., Boch, T., Donaldson, T., Durand, D., O'Mullane, W., Reinecke, M., and Taylor, M., "MOC - HEALPix Multi-Order Coverage map Version 1.0." IVOA Recommendation (June 2014). arXiv:1505.02937.

[54] Reinecke, M. and Hivon, E., "Efficient data structures for masks on 2D grids," *Astronomy & Astrophysics* **580**, A132 (Aug. 2015).

[55] Bosch, J., "LSST Classes, as AstroPy Spin-Off Candidates," (Mar. 2016). doi:10.5281/zenodo.48435.

[56] Bertin, E., Delorme, P., and Bouy, H., "AstrOmatic Software in the Era of Large Stellar Photometric Surveys," *Astrophysics and Space Science Proceedings* **29**, 71 (2012).

[57] Jenness, T., Berry, D. S., Currie, M. J., Draper, P. W., Economou, F., Gray, N., McIlwrath, B., Shortridge, K., Taylor, M. B., Wallace, P. T., and Warren-Smith, R. F., "Learning from 25 years of the extensible N-Dimensional Data Format," *Astronomy and Computing* **12**, 146–161 (Sept. 2015).

[58] Padmanabhan, N., Lupton, R., and Loomis, C., "EUPS — a Tool to Manage Software Dependencies." https://github.com/RobertLuptonTheGood/eups (2015).

[59] Knight, S., "Building software with SCons," *Computing in Science Engineering* **7**, 79–88 (Jan 2005).

[60] McKinney, W., "Data Structures for Statistical Computing in Python," in [*Proceedings of the 9th Python in Science Conference*], van der Walt, S. and Millman, J., eds., 51 – 56 (2010).

[61] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K., "Cython: The best of both worlds," *Computing in Science & Engineering* **13**(2), 31–39 (2011).

[62] Schellart, P., "Wrapping C++ with Cython," (2016). LSST Data Management Technical Note DMTN-013, http://dmtn-013.lsst.io.

[63] Schellart, P., "Wrapping C++ with pybind11," (2016). LSST Data Management Technical Note DMTN-014, http://dmtn-014.lsst.io.

[64] Pontzen, A., Roškar, R., Stinson, G. S., Woods, R., Reed, D. M., Coles, J., and Quinn, T. R., "pynbody: Astrophysics Simulation Analysis for Python," (2013). Astrophysics Source Code Library, ascl:1305.002.