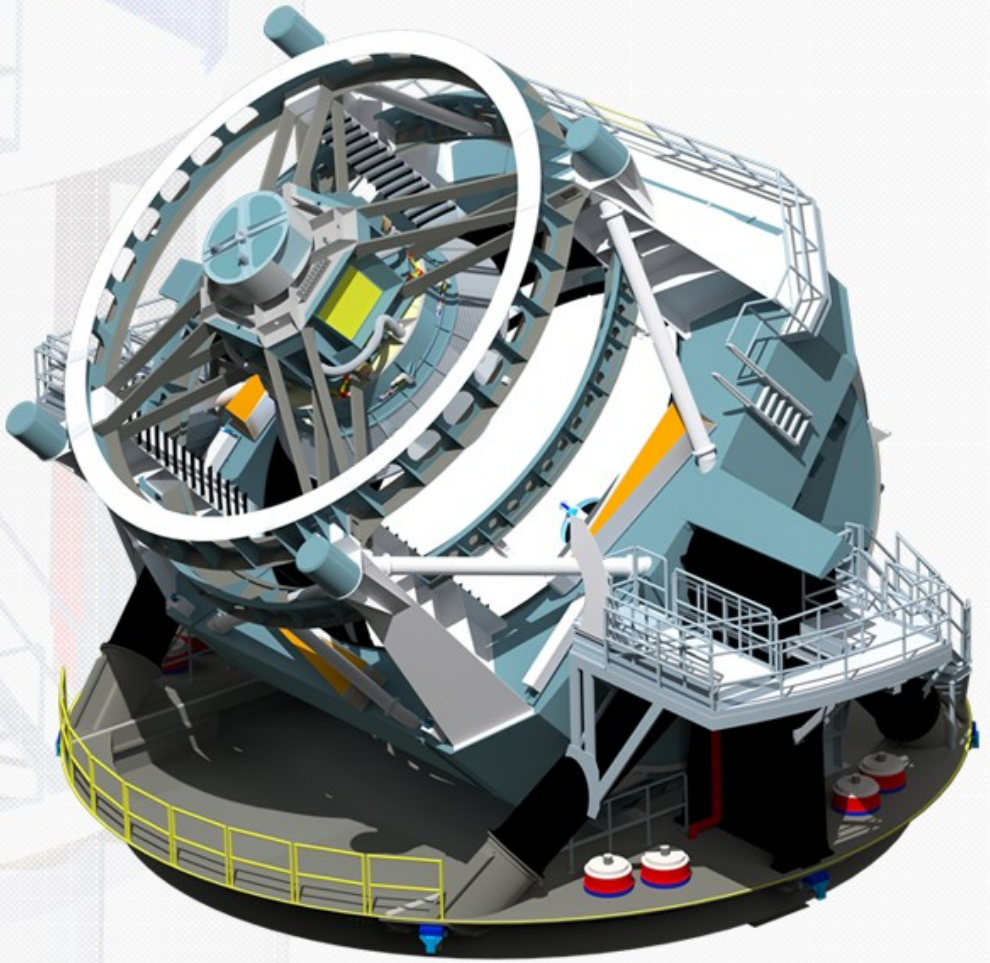
LSST Code, as AstroPy Spin-Off Candidates

Jim Bosch

LSST DRP Scientist

3/26/2016



LSST/AstroPy Summit

3/26-3/27/2016 | University of Washington

Some Categories

- **Obvious counterpart classes:** AstroPy and LSST already have something for these roles, and they're central enough we need to integrate them somehow (pick a winner or make converters, adaptors, etc.).
- **Low-Level Primitives:** basic classes LSST does well (aside from some interface polishing), and needs to have available (including in C++) to build on. Probably some overlap with AstroPy functionality.
- **Critical Middleware:** low-level code for gluing things together and configuring them. Mostly pure Python.
- **Mid-Level Algorithms:** everything you need to replace SExtractor, mostly Python, but building on all of the above.
- **Pipeline Toolkit:** specialize for your camera, add some plugins, then we call you.

Some Categories

- **Obvious counterpart classes:** AstroPy and LSST already have something for these roles, and they're central enough we need to integrate them somehow (pick a winner or make converters, adaptors, etc.).
- **Low-Level Primitives:** basic classes LSST does well (aside from some interface polishing), and needs to have available (including in C++) to build on. Probably some overlap with AstroPy functionality.
- **Critical Middleware:** low-level code for gluing things together and configuring them. Mostly pure Python.
- **Mid-Level Algorithms:** everything you need to replace SExtractor, mostly Python, but building on all of the above.
- **Pipeline Toolkit:** specialize for your camera, extend algorithms, then run on top of our middleware.

The Spin-Off Proposal

For some piece of LSST code:

- Keep (most of) the C++ code
- Rewrite (or expand) the C++/Python boundary to improve the Python interface.
- Support AstroPy objects as inputs and outputs.
- Provide an AstroPy-style build system, as well as an LSST one (be both pip installable and eups declarable).
- Continue to provide a C++ API.
- Rework dependent LSST code to use the new package.
- Submit as an AstroPy affiliate package.

Why Spin-Off?

- Opportunity to do something incrementally we ultimately need to do globally: improve interfaces, documentation, and ease-of-install.
- Ensure community development proceeds in a way that is compatible with LSST.
- Testbed for development/packaging model that may make LSST developers more efficient.
- Provide a template for AstroPy affiliates with C/C++ APIs.
- Encourage community contributions to LSST code.
- Make it easier to use other AstroPy code that LSST would like to build on top of.

Obvious Counterpart Classes: `afw.image` vs. `astropy.nddata`

Provides

- `Image` (`numpy.ndarray` + nonzero integer origin)
- `Mask` (integer `Image` with bits interpreted as mask planes)
- `MaskedImage` (`Image` + `Mask` + variance `Image`)
- `Exposure` (`MaskedImage` + PSF, WCS, ZP, etc.)

What's Good

- Overall model, types of classes (perhaps not *uniquely* good).
- Nonzero origin *critical*.
- Good NumPy/C++ integration (only lacks sugar).
- Mask plane handling usually convenient.

What's Bad

- Mask plane handling sometimes surprising.
- MaskedImage perhaps not general enough (more planes?).
- Exposure metadata management and persistence ugly.

Way Forward

- Start with bidirectional views, no round-tripping.
- Add polish and some duck-type compatibility to LSST Python interfaces.
- Learn from each other's designs, and migrate towards each other slowly.

Obvious Counterpart Classes: `afw.table` vs. `astropy.table`

Provides

- Row-major strided data tables, available in C++ and Python.

What's Good

- Record classes subclassable (very limited but very useful ORM).
- Fast append, modestly efficient sorting.
- NumPy column views available when contiguous.
- *Overall: good for pipeline code that builds tables.*

What's Bad

- Adding columns is a painful, two-step process (define schema, fill later).
- Ugly dependencies (`Boost.Variant`).
- Subclassing is heavy on boilerplate, C++-only.
- *Overall: bad for analysis code that manipulates existing tables.*

Way Forward

- Start with LSST-to-AstroPy views, stop using LSST tables for analysis.
- Consider adding C++-backed row-major tables to AstroPy in addition to independent column tables? Probably don't want LSST code as-is, but could develop replacement jointly.
- What about Pandas?

Obvious Counterpart Classes: `afw.coord` vs `astropy.coordinate`

- We currently have a C++ interface for this.
- We don't love it.
- We only ever use ICRS in C++.

Maybe we can just drop ours and use AstroPy's as-is?

Obvious Counterpart Classes: `afw.image.Wcs` vs `astropy.wcs`

- Everybody uses `wcslib`.
- No one wants to.
- Ideally, we should agree on what replaces it.
- gWCS isn't a perfect fit for LSST; we need:
  - Composable. ✔
  - Pluggable. ***?***
  - Accessible from C++, with no Python calls during transformation. ✘

So we're still exploring our options (led by John Parejko).

Low Level Primitives: `afw.geom`

Provides

- Euclidean points, boxes (floating and integer)
- Affine coordinate transforms
- Polygons (rudimentary, boost-dependent)

Pros of Spin-Off

- Most minimal dependencies.
- Dependency for virtually everything else.
- Small amount of code.

Cons of Spin-Off

- Need to think about integration with `astropy.region`, `lsst.sphgeom`
- Some components (point?) may not add much value: why reuse when reimplementation is trivial?

Low Level Primitives: `afw.geom.ellipses`

Provides

- Different parametrizations of ellipses: (a,b,theta), (xx,yy,xy), etc.
- Utilities for evaluating elliptically symmetric models.

Pros of Spin-Off

- Minimal dependencies.
- Small amount of code.
- Difficult to reimplement (subtly difficult numerical algebra), already battle-tested.

Cons of Spin-Off

- Maybe most valuable in C++.
- Would prefer to maintain dependency on `afw.geom` `Box`, `Transforms`, `Point`.

Low Level Primitives: `afw.cameraGeom`

Provides

- Descriptions of astronomical cameras, including coordinate systems and electronics.

Pros of Spin-Off

- Desirable: important abstraction layer for low-level processing and simulation.
- We've already been through two design iterations, and learned from the mistakes (doing it well is hard).

Cons of Spin-Off

- `afw.table` dependency is hard to work around.
- Should be tightly integrated with WCS; might want to wait for that?

Low Level Primitives: `afw.detection`

Provides

- `Footprint` (like SExtractor segmentation map element)
- Image thresholding for detection

Pros of Spin-Off

- Algorithmic quality from experience not available elsewhere.
- Good low-level code: fast, robust, battle-tested.
- Relatively small C++/Python boundary layer.

Cons of Spin-Off

- Dependency on `afw.table` (not integral, but hard to work around).
- Dependency on `afw.geom` (integral, but easy to work around).
- Needs more LSST code to be most useful to users (see `SourceDetectionTask`).

Low Level Primitives: `afw.math.Kernel`, `Psf`

Provides

- Point spread function model abstractions.
- *Spatially-varying* convolution.
- Image resampling.

Pros of Spin-Off

- Very useful functionality for astronomers.
- PSF is an important component of image classes.
- Easy to get a dead-end core module if this isn't contributed by a team with experience.

Cons of Spin-Off

- Lots of dependencies.
- Interface has been stable, but is still due for big changes.
- Piff might supercede or require big changes.

Low Level Primitives: `afw.math.Statistics`

Provides

- Robust and efficient aggregate statistics on images and arrays.

Pros of Spin-Off

- Generally higher-quality implementation than NumPy/SciPy.
- LSST needs to refactor anyway.
- Very minimal dependencies (once refactored).

Cons of Spin-Off

- Inappropriate for AstroPy? Should ultimately go in NumPy/SciPy instead?
- Already available for Python users in larger SciPy ecosystem?

Low Level Primitives: `shapelet`

Provides

- Elliptical Gauss-Hermite/Gauss-Laguerre functions.
- Model evaluation for multi-Gaussian approximations to Sersic functions.

Pros of Spin-Off

- Battle-tested and fast.
- Difficult to reimplement (subtly difficult numerical algebra).
- Thin C++/Python boundary.

Cons of Spin-Off

- Depends heavily on `afw.geom.ellipses`, `afw.detection.Footprint`.
- Not a dependency for much else (just galaxy model fitting).

Critical Middleware: `The Butler`

Provides

- Abstract I/O interface: refer to datasets by name and data ID instead of filenames and formats.

Pros of Spin-Off

- This is a dependency for any high-level LSST code.
- Pure Python.
- Minimal dependencies.

Cons of Spin-Off

- Under heavy development on LSST.
- Strongly driven by LSST middleware needs.

Critical Middleware: `pex.config`

Provides

- Configuration files written in Python itself, via metaprogramming.
- Multiple levels of overrides and defaults.
- History tracking: why is this option set this way?

Pros of Spin-Off

- This is a dependency for any high-level LSST code.
- Pure Python.
- Minimal dependencies.

Cons of Spin-Off

- Metaprogramming galore = hard to read, maintain (intrinsic to this approach?)
- We're not sure writing configs in Python was the right idea after all.

Critical Middleware: `pipe.base`

Provides

- Callable base class for processing tasks with convenience functionality: logging, configuration.
- Command-line argument parsing and single-node multiprocessing for pipelines.

Pros of Spin-Off

- This is a dependency for any high-level LSST code.
- Pure Python.
- Minimal dependencies.

Cons of Spin-Off

- Need to better separate command-line parsing from execution, parallelization, I/O, to allow extending these separately.
- Can be a bit boilerplate-heavy.
- Still under heavy development.

Mid-Level Algorithms

Provides

- `SourceDetectionTask`: find sources in an image
- `SourceDeblendTask`: deblend neighboring sources at the pixel level
- `SingleFrameMeasurementTask`: measure sources in one image
- `ForcedMeasurementTask`: measure in another image with fixed positions/shapes

Pros of Spin-Off

- *Really* useful for astronomers: replaces and improves SExtractor, but callable, configurable, and *extensible* from Python.

Cons of Spin-Off

- Depends on virtually everything above.

Not Appearing In This Talk: Pipeline Toolkit

- Create an "obs_*" package for your camera.
  - Define camera geometry and electronics.
  - Customize instrument signature removal as needed.
- Configure and extend the pipeline.
  - Change algorithm settings.
  - Swap out different algorithms.
  - Add new algorithms as plug-ins.
- Use our scripts to:
  - Process individual frames.
  - Fit relative astrometry.
  - Build coadds.
  - Detect/Deblend/Measure on (multi-band) coadds.

Breakout Ideas

- Hack on `afw.image` vs. `astropy.nddata` integration
- Hack on `afw.table` vs. `astropy.table` integration
- Brainstorm OO relationships for Modeling
- Brainstorm OO relationships for Regions
- Hack on a spin-off project.
- Hack on build/packaging system integration.