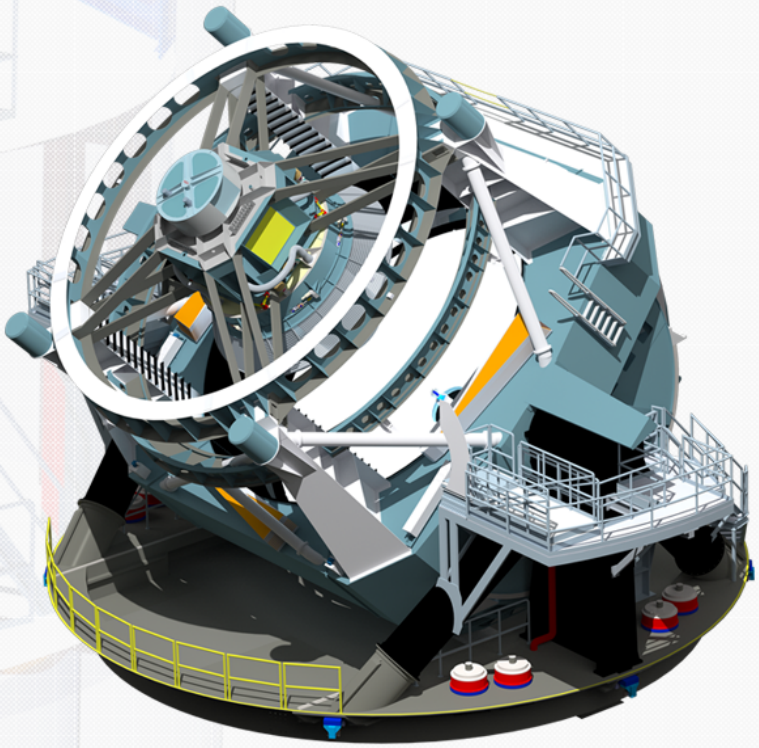
DM Code Structure

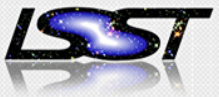
Tim Jenness

DM Deputy System Architect

October 6th 2015

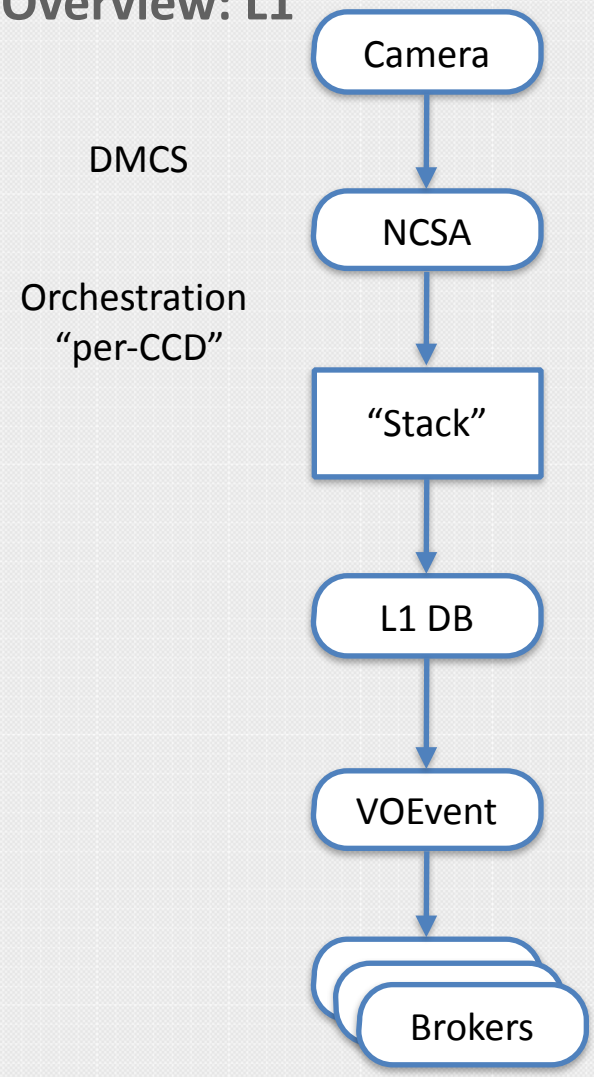


Overview



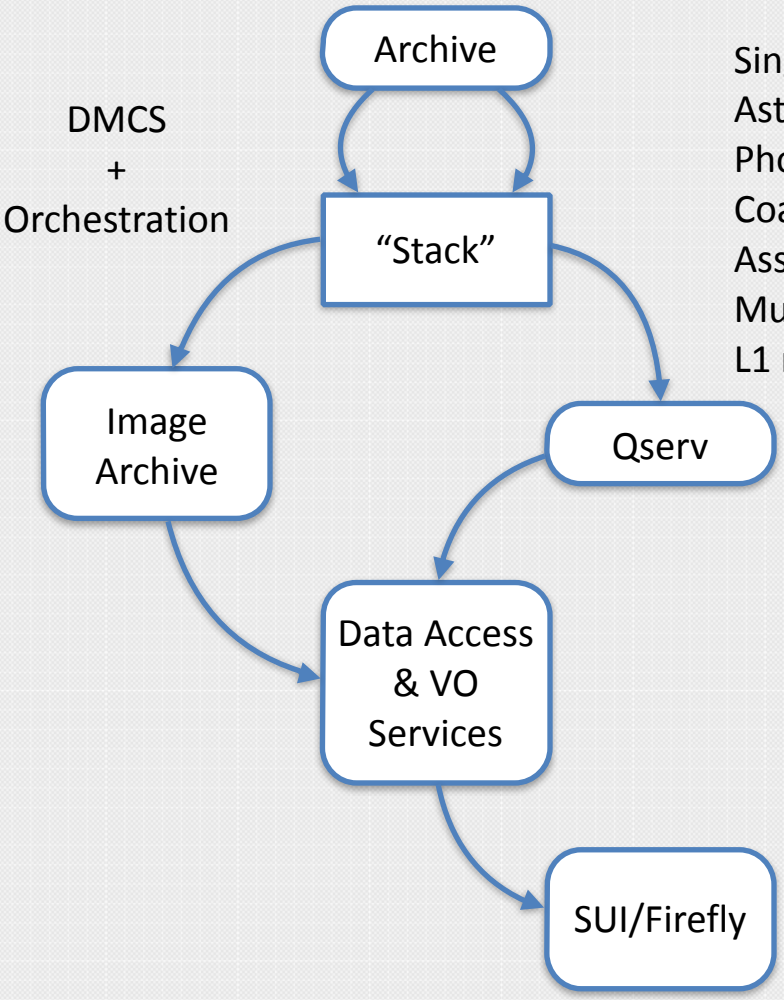
- Combination of C++11 and Python 2.7:
  - C++ used where performance is critical.
  - Prefer to use Python by default.
- Intending to support Python 3 soon.
- GCC 4.8 minimum supported compiler.
- Clang supported.
- Science User Interface uses Java 1.6 server side, Javascript on client side.
- Linux CentOS 6 & 7 and OS X Yosemite and Mavericks are test platforms. No El Capitan support at the moment.
- All LSST code is in an `lsst` namespace.

Processing Overview: L1



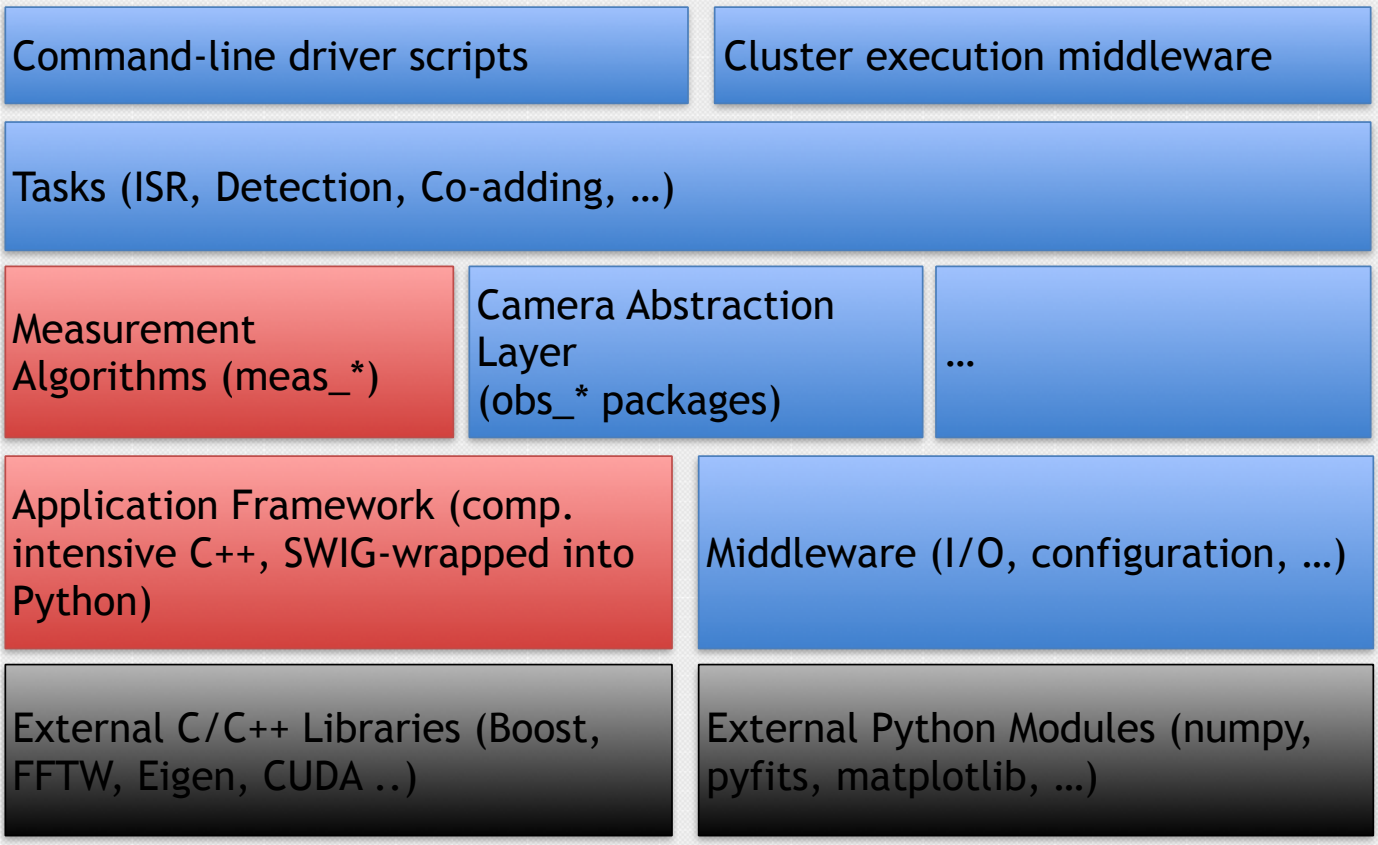
Single frame processing
Astrometry
Photometry
Difference Imaging
Association

Processing Overview: L2



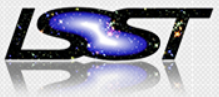
Single frame processing
Astrometry
Photometry
Coadding
Association/Deblending
Multi-fit
L1 reprocessing

Structure of the stack



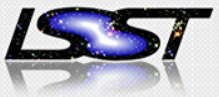
Red: Mostly C++ (but Python wrapped); Blue: Mostly Python; Black: External Libraries

Package overview



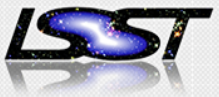
- Orchestration: ctrl
- Data Access Framework (aka Butler): daf
- pipeline execution support: pex
- General application framework: afw
- Image Processing: ip
- Measuring properties of datasets: meas
- Co-addition of images: coadd
- Pipeline infrastructure: pipe
- Control display tools: display
- Camera-specific mappings: obs
- Web visualization of catalogs and images: firefly (Java)
- Data Access: dax
- Full list: https://confluence.lsstcorp.org/pages/viewpage.action?pageId=9929032

Middleware: logging



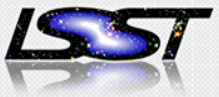
- The log package is used for logging.
- C++ and Python interfaces.
- INFO, WARN, DEBUG, TRACE, FATAL levels.
  - logger.info("An informational message")
  - logger.warn("WARNING message")
- Log levels can be controlled at run time.
- Replaces the pex_logging package.
  - Use log for new code.
  - pex_logging will be removed from the existing codebase.
- Docs: https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/log.html

Middleware: exceptions



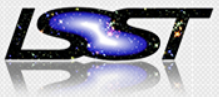
- pex_exceptions provides a set of C++ exceptions that can be caught in python code.
- Use native Python exceptions when appropriate.
- Exceptions include:
  - LogicError, DomainError, InvalidParameterError, LengthError, OutOfRangeError, RuntimeError, RangeError, OverflowError, UnderflowError, NotFoundError, MemoryError, IoError, TypeError, TimeoutError.
- Docs: https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/pex_exceptions.html

Useful support packages



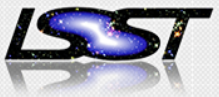
- daf_base
  - Date and time handling:
    - TAI/UTC, MJD/JD/J2000, string conversions.
  - PropertySet/List:
    - simple key/value pairs (e.g. FITS headers).
  - https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/namespacelsst_1_1daf_1_1base.html
- ndarray: C++ n-dimensional arrays compatible with numpy.
  - https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/ndarray.html
- geom: Cartesian and spherical geometry in Python.
  - https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/namespacelsst_1_1geom.html
- db: Database access utilities.

Third-party packages



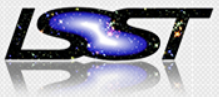
- 3rd party packages are distributed as EUPS-managed packages installable using eups distrib or lsstsw.
  - They are versioned.
  - LSST packages list them as explicit dependencies.
- Numpy and matplotlib are presumed to have been installed by other means (but this is checked).
- Some 3rd party packages are not expected to be called directly: e.g. log4cxx
- New packages can be requested via the RFC process.
  - Using 3rd party packages is encouraged rather than reimplementing a wheel.
- 3rd party Python packages are currently distributed in this way and not via pip.

Third-party packages



- Available packages include:
  - cfitisio: Read/write FITS files,
  - eigen: linear algebra: matrices, vectors, numerical solvers, and related algorithms,
  - boost: general C++ extensions,
  - wcslib: World coordinates handling,
  - fftw: Fast Fourier transforms,
  - gsl: Gnu Scientific Library — mathematical routines such as random number generators, special functions and least-squares fitting,
  - minuit2: function minimization.
- Full list: https://confluence.lsstcorp.org/display/DM/DM+Third+Party+Software

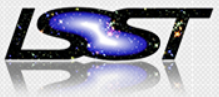
Structure of a package



bin	Executables (includes python scripts)
doc	Documentation
examples	Python and C++ example code
include	C++ include files
lib	Compiled shared libraries (empty on check out)
python	Python code and Swig interface files
src	C++ source code
tests	Test code (Python and C++)
ups	EUPS configuration (dependencies, environment variables)

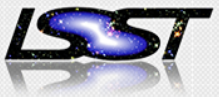
See: https://github.com/lsst/templates for a detailed example

Building a package: Scons



- SCons is used to build LSST software.
  - http://scons.org
  - A “Software Construction tool”
  - Written entirely in Python.
- To build and test a package:
  - setup -k -r . — to ensure the tests use the newly-built code.
  - scons -j 4 opt=3 — for a parallelized build.
- This will build the C++ (including the shared libraries) and Python code, build the examples, run the tests and build the documentation (currently using doxygen).
- scons -j4 python will just build the python code.
- SConstruct file used by scons to configure the build.
- SConscript files are used for subsidiary configuration in subdirs.

LSST extensions: sconsUtils

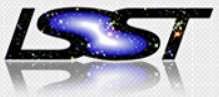


- Scons does not know anything about LSST software conventions.
- sconsUtils adds all the LSST-specific knowledge:
  - EUPS version and dependency tracking.
  - Compiler detection (clang vs gcc and how to add C++11 support).
  - Swig interface building.
  - How to run tests.
- Simplest possible SConstruct file:

```
# -*- python -*-
from lsst.sconsUtils import scripts
scripts.BasicSConstruct("meas_base")
```

for a package laid out in the standard form with no special features.

Python code

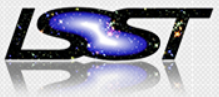


- Lives in the python/lsst directory.
- If your package is named something_else the code will be located in python/lsst/something/else.
- Each sub-directory (lsst and below) must have an __init__.py file that must contain at least:

```
import pkgutil, lsstimport
__path__ = pkgutil.extend_path(__path__, __name__)
```

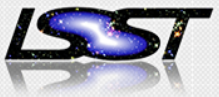
- This is required to ensure that all the packages can be found when spread across the filesystem with EUPS using PYTHONPATH sharing the same lsst namespace.

Python/C++ Interfaces: SWIG



- Simplified Wrapper and Interface Generator
  - http://www.swig.org
- Parses a C++ header file and generates the Python wrapper code.
- Interface is defined in ".i" files that live in the python/lsst/ tree.
  - meas_base .i files are in python/lsst/meas/base/
- Swig generates .py file and shared library. For meas_base baseLib.i generates baseLib.py and _baseLib.so.
  - The other .i files are included by baseLib.i.
  - The SConscript file in the same directory tells scons that only baseLib is relevant
  - Look in the _wrap.cc file to see what Swig has generated.

3rd Party Package Structure



- 3rd Party packages are wrappers around the standard distribution files.
- EUPS, via eupspkg, knows how to build different styles of distro: Python setup.py, autoconf, cmake.

ups	EUPS configuration, including dependencies, and special build instructions.
upstream	Upstream unmodified distribution tar file.
patches	Patches to be applied to the distribution before building it.

Testing

- Each package has associated unit tests
- Python tests use unittest.
  - assertEqual, assertLess etc.
  - Only use assertTrue if you are really testing truth.
- C++ tests use boost.
- Aim for new code to come with associated unit tests.
  - Code coverage is less than ideal at present but aiming to begin gathering metrics on this.
  - Integration tests are used to test the stack as a whole.
- Python tests will soon be run via a standard python test environment such as nose or py.test.
  - This gives us significantly better test output handling in the Jenkins continuous integration system.

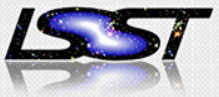
Documentation

- Currently the doc directory contains doxygen format files with a root document page in either mainpage.dox or main.dox.
- Doxygen format used for method and class descriptions inline.
- Currently investigating migrating to ReST and numpydoc format and aiming to integrate into ReadTheDocs.

Those “ups” configuration files

- ups directory teaches EUPS and sconsUtils how the package relates to other packages and how to configure it when it is “setup”.
- .table file lists dependencies and environment variables that EUPS should set.
- .cfg file contains configuration information for sconsUtils that might be needed from packages depending on this package.
- eupspkg.cfg.sh provides overrides and additional information to allow eupspkg to build a package.

Coding Standards



- Python coding standard: https://confluence.lsstcorp.org/display/LDMDG/Python+Coding+Standard
  - Considering recasting standard as a delta relative to PEP8
- C++ coding standard: https://confluence.lsstcorp.org/pages/viewpage.action?pageId=16908666