Winter2015 Package Reorganization Planning

Introduction

This page describes a straw-man proposal to reorganize the LSST DM software stack, both at the package (i.e. git/eups/scons boundary) and namespace (C++ namespace and Python package) level. At the package level, it is almost entirely a consolidation of packages. Many existing packages will have components moved into more than one package in the new scheme, but most will have the vast majority of their content move to just one new package, and there will be many fewer packages.

The motivation for moving towards consolidation are:

- Having fewer packages is less intimidating to users and makes the installation process at least *feel* simpler (I hope that it may also *be* simpler, but I don't think we know that yet).
- Having a large number of packages complicates the build, packaging, testing, and documentation systems, requiring more things to be automated to keep developer workflows efficient.
- Testing is easier with a smaller number of packages, as high-level components that may have otherwise been in a separate package can be used when testing low-level components.
- Earlier concerns about revision control collisions due to multiple developers working on the same package have essentially been eliminated by the switch from svn to git.
- A smaller number of packages makes a namespace reorganization easier, by making it more likely that related code can be put within a single namespace without needing that namespace to cross package boundaries.

There are also some arguments against:

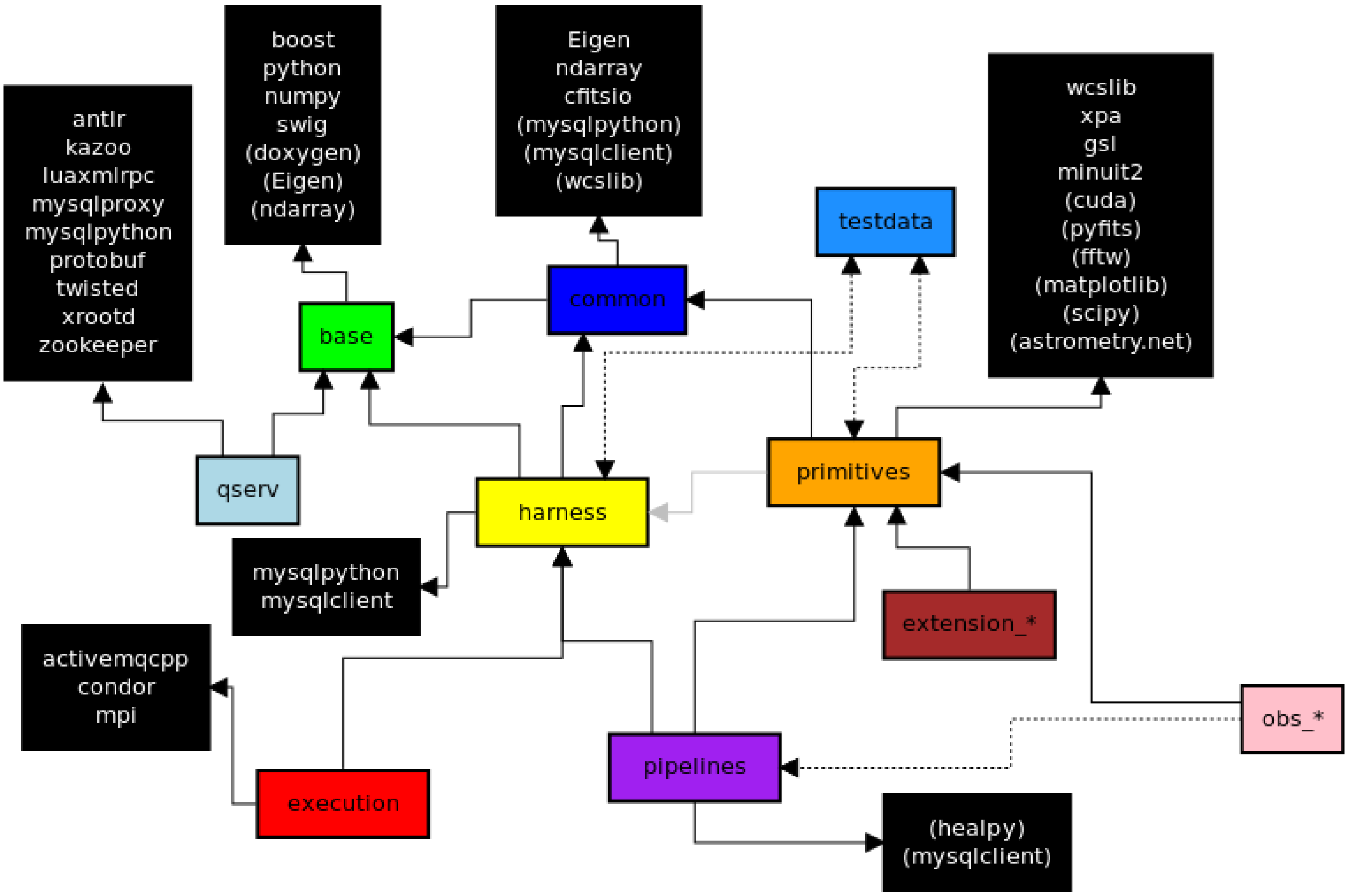
- Partial stack rebuilds that don't reuse local source directories (e.g. those invoked by "eups distrib install") will be slower on average, as we will have to rebuild more code. Note that binary packages installs should be unaffected overall, and developer builds that reuse local source directories may be *slightly* faster, as scons should be able to do a better job of determining what needs to be rebuilt when it doesn't have to cross package boundaries.
- Both internal and external packages will often have dependencies that are larger than they strictly need to be, which will make building just that package slower and more difficult than before, especially if the extra dependencies in our code bring extra third-party package dependencies. The main goal of a good reorganization proposal is that it mitigates this effect.

And there are arguments that are strongly in favor of some kind of *namespace* reorganization, but are agnostic as to whether the corresponding package reorganiation should be a consolidation, a further partition, or neither:

- Our current set of namespaces are largely historical and based on WBS elements that are meaningless to users and most developers.
- Some of our most important logical code units are each scattered across several different namespaces and packages, making them harder to follow than necessary.
- Many of our current namespaces are essentially smorgasbords of unrelated or tenuously-related code, and hence provide little or no organizational utility.

Package Reorganization Proposal

New Packages



In the above diagram, lines indicate dependencies between the proposed new packages (arrow points from the dependent package to the one it depends on). Regular solid lines indicate required dependencies, while dotted lines indicate optional dependencies that are necessary to enable certain features. The gray line between "primitives" and "harness" indicates a possible temporary dependency that may be necessary during the initial transition but should ultimately become unnecessary after some already-planned improvements to the codebase are made.

The third-party package dependencies are discussed in the next section.

All of the names of the above packages are purely provisional; I hate naming things, and I think the content of these packages is a much more important discussion than their names. But I'd rather not change any names until that discussion is done, because it'd be a pain to change all the diagrams.

Here's a general description of the philosophy behind this organization:

- **qserv** is mostly distinct from the rest of the DM stack, and shares only a few low-level components. I've put these shared components in **base,** as well a few others that are both lightweight and of potential use to **qserv** in the future (or closely related to features that are of potential future use). Much of the content of **base** is geometry primitives and algorithms - **qserv** and the science pipeline both need spherical geometry, and it makes sense to keep this with the Euclidean geometry, as the two should probably share components (and while only the science pipeline code and its supporting middleware need the Euclidean geometry at present, the database interfaces may need them in the future). However, I'm actually not sure here where the line between qserv and scisql is being drawn - it's entirely possible that some of the code I've considered a qserv dependency is actually a scisql dependency.
- **common** contains more low-level components that **qserv** probably has no interest in, but are useful as both the building blocks and as a "common language" between middleware and algorithmic code. This includes completely general code like the configuration system as well as more astronomy-specific components like `Image` and `BaseCatalog` (but not their specialized higher-level counterparts, like `Exposure` and `SourceCatalog`), as these are expected to play an important role in the persistence framework (interfaces that define persistable classes are largely housed in **common**, though the `Butler` is not). **common** also contains the `Task` base class, but *not* `CmdLineTask`.
- **harness** contains the `Butler` and its helper classes, as well as `CmdLineTask`, argument parsing, and interfaces (but not implementations) for parallel execution.
- **execution** contains the implementations for parallel execution, i.e. the former `pex_harness` and most of the contents of `ctrl_*`. I've been quite vague about what it contains beyond that, as it's the part (along with qserv) of the codebase I'm least familiar with.
- **primitives** contains the vast majority of the low- and mid-level algorithmic code and the data types it uses - not just most of the previous contents of `afw`, but the previous contents of most of `meas_*` and `ip_*` as well. **pipelines** contains the high-level algorithmic code: most of the previous content of `pipe_tasks`. Essentially, the dividing line between **primitives** and **pipelines** is whether the code needs to make use of the `Butler` or do parallel processing; code that does lives in **pipelines**, while code that doesn't lives in **primitives**. The possible temporary dependency of **primitives** on **harness** represents the fact that a couple of our current algorithmic `Task` classes (`CalibrateTask` and `IsrTask`) currently do depend on the `Butler`, but ultimately should not, and hence should live in **primitives**.
- **testdata** will be, in the near future, a straightforward combination of `afwdata` and `obs_test`. Eventually I'd like it to contain a more sensibly-defined simulated test dataset that's fully butlerized, using an artificial camera that avoids dependencies on any of the real **obs_*** packages. It may be useful for it to contain code used in generating new on-the-fly test data as well. That complicates its relationship with **primitives** and **harness**, because an embedded **obs_*** package implies a dependency on **harness,** and any code to generate test data on-the-fly would have to rely

on **primitives**. At the same time, both of those packages would continue to depend on **testdata** for some of their unit tests. Essentially, we'd have an optional circular dependency - while you'd be able to use either of **primitives** or **harness** without the other, you might not be able to *fully* test either without having all three packages setup. If that's too hard to express to Eups, we could just move such tests to **pipelines**, though I think making the backwards dependency of **testdata** on **primitives** and **harness** implicit would be work as well, because there's no reason to use **testdata** without having at least one of those two set up. Note that I still expect many tests in **harness** or **primitives** to work without **testdata**.

- **extension_*** are packages that represent unofficial extensions to the pipelines. Unlike in the past with meas_extensions_*, I think we should not put code in **extension_*** if we plan to run it regularly as part of the pipeline - this put an unfortunate burden the **obs_*** packages, which were then the only place where frequently-used extensions could be enabled. In this proposal, **extension_*** packages really are "level 3" sort of packages, and if we decide we like something well enough it should be run as part of "level 2", then we should move it to **pipelines** or **primitives**.
- the **obs_*** packages are mostly as they were. I'm expecting they should mostly be able to depend only on **primitives**, not **pipelines**, but given the current dependency of `IsrTask` on the `Butler`, that may not be achieved immediately. And an individual **obs_*** package would still be able to depend on **pipelines** if necessary (though I hope it won't be necessary, and that this makes other potential users of **obs_*** packages, like PhoSim, happier).

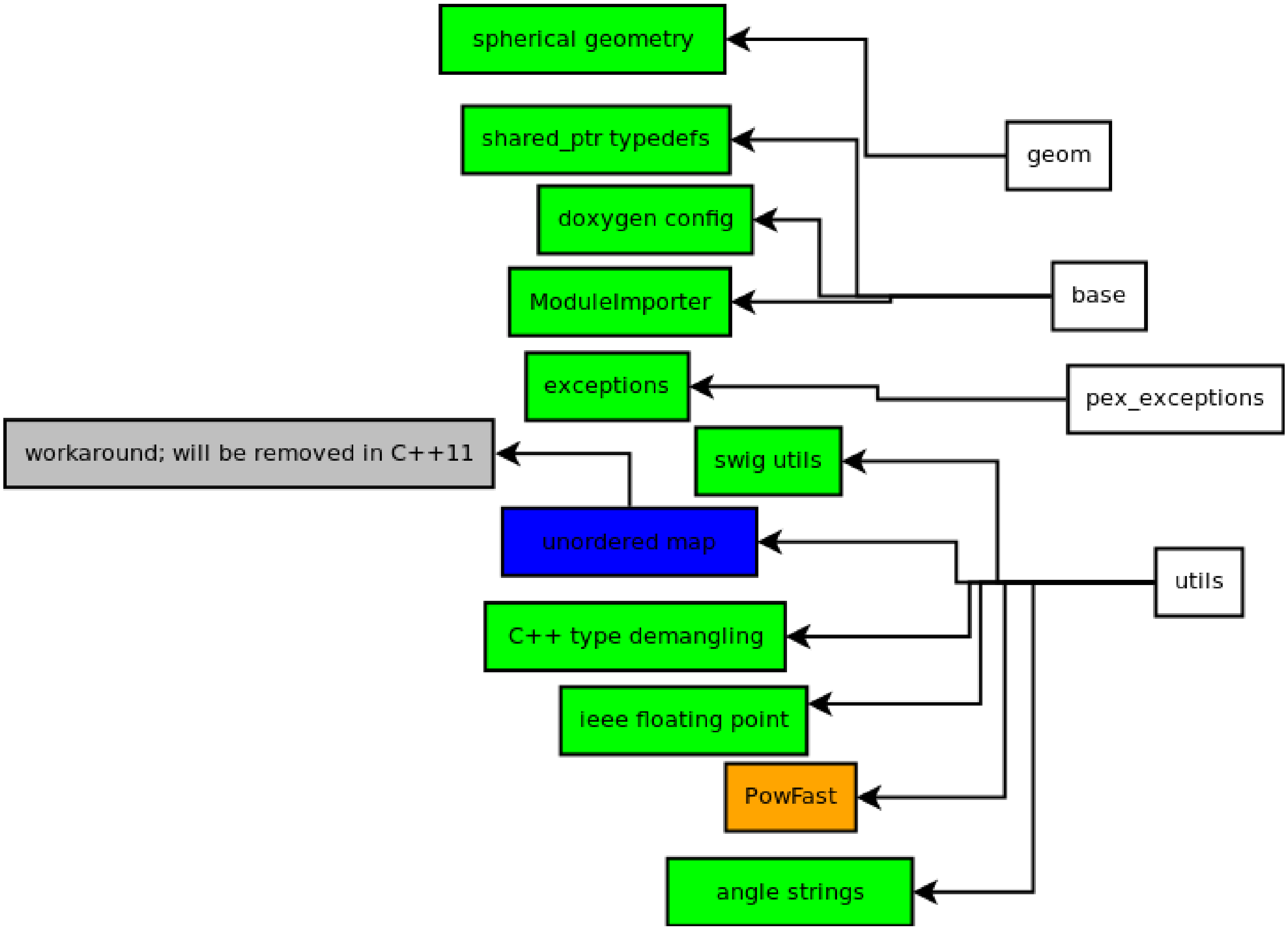
Third-Party Package Dependencies

Third-party package dependencies are shown in the above diagram as black boxes. Dependencies that are optional or relatively straightforward to remove are in parenthesis, with more discussion below.

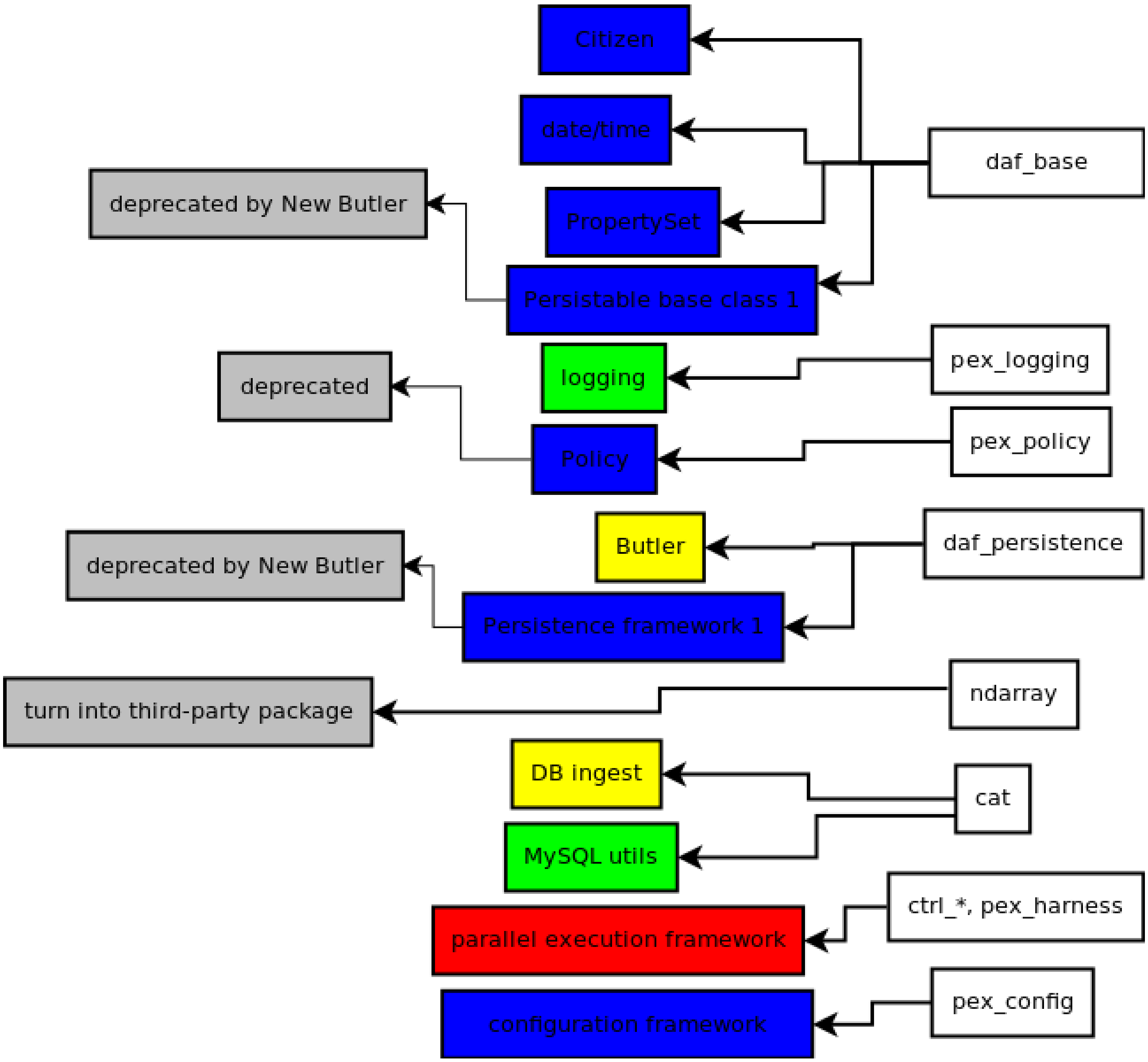
- Nearly all of our code depends on **boost**, **swig**, and **python**, and enough of our low-level code depends on **numpy** that I think we should consider these four essential for any piece of the pipeline. When we've adopted C++11, it may be useful to revisit **boost**, as most of our low-level components only use **boost** components that are now available in the C++11 standard library. I don't think it's worthwhile to try to make that split now, though.
- **qserv** depends on many other packages none of the rest of the stack needs, and it's just as easy in the new layout as it was in the old to keep these from bleeding into the rest of the stack.
- The code I've slated for the **base** package currently depends on **Eigen**, **ndarray** (which I envision moving to third-party status), and **Doxygen**, which means these would become implied dependencies of **qserv**. I think it'd be easy to ultimately remove all of these dependencies, however, by including in **base** only the lowest-level Euclidean geometry components (this is a change from my original proposal, reflected in the detailed mapping below), and simply removing the minimal (and relatively unimportant) dependencies on **Eigen** and **ndarray** that the low-level components currently have. For **Doxygen**, I think it's highly likely we'll be moving to a different documentation build system, which may not require **Doxygen** at a per-package level. In any case, it's always an optional dependency.
- **common** will almost certainly have to depend on **Eigen** and **ndarray**, at least as long as the image classes and the higher-level geometry classes are here. Its dependencies on **mysqlclient** and **mysqlpython** are very much temporary; I expect them to be removed along with the rest of the old Formatter-based persistence framework. I expect these dependencies to resurface in **harness**, which I imagine database ingest code landing. A **common** dependency on **cfitsio** is hard to avoid, especially immediately, though I suppose it's possible this could be moved down to **harness** if we separate `BaseTable` and `Image` persistence from the classes themselves (but even if that's desirable, we shouldn't count on it being easy, or happening soon). I've also made a change to my detailed mapping from my original version that puts the `Wcs` base class in **common** instead of **primitives**. I think that's necessary for the **harness** components that need to be able to load data based on its position on the sky, but it brings along a dependency on **wcslib**. I think we'll be able to push that dependency back up to **primitives** eventually, though, by making the `Wcs` class pure abstract and moving the entirety of the implementation to **primitives**.
- Like **qserv**, **execution** depends on a few packages the rest of the stack doesn't, and it's easy separate things. The only one I'm less certain about is **mpi**, which we might need to move to **harness** if we want to expose some parts of it directly to algorithmic code instead of hiding it completely behind our own message-passing interfaces.
- Essentially all the third-party packages required by the science code are required by **primitives** (and are required by **afw** in the current layout; the pipe_*, meas_*, and ip_* packages above **afw** add *no* additional required dependencies to the stack). Of these, I think **wcslib** is unavoidable, as are **gsl** and **minuit2**, unless we replace these with similar third-party libraries. We could make **xpa** optional by making part of the build system conditional, as is already the case for **cuda**. **Matplotlib**, **pyfits** and **scipy** are only used for diagnostics and tests, are are hence optional, and should remain that way. Interestingly, while **fftw** is listed as a required dependency of **afw**, we actually seem to have no code that uses it (but it seems likely that we would someday). We've already made plans to remove the dependency on **astrometry.net**. The only additional dependencies for **pipelines** are optional: **healpy**, for one of the `skymap` implementations, and **mysqlclient**, which is currently used by **ap**.

Detailed Mapping from Old to New

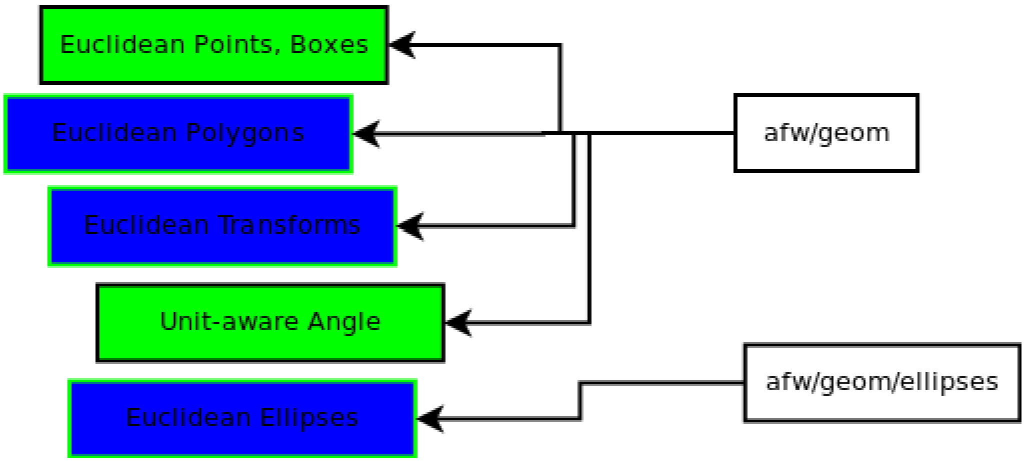
You can get the full color-coded diagram where I did all this work (using https://wiki.gnome.org/Apps/Dia/) here: packages.dia. I've split that into chunks to paste the images below, with a bit of text below each chunk explaining some of my reasoning.



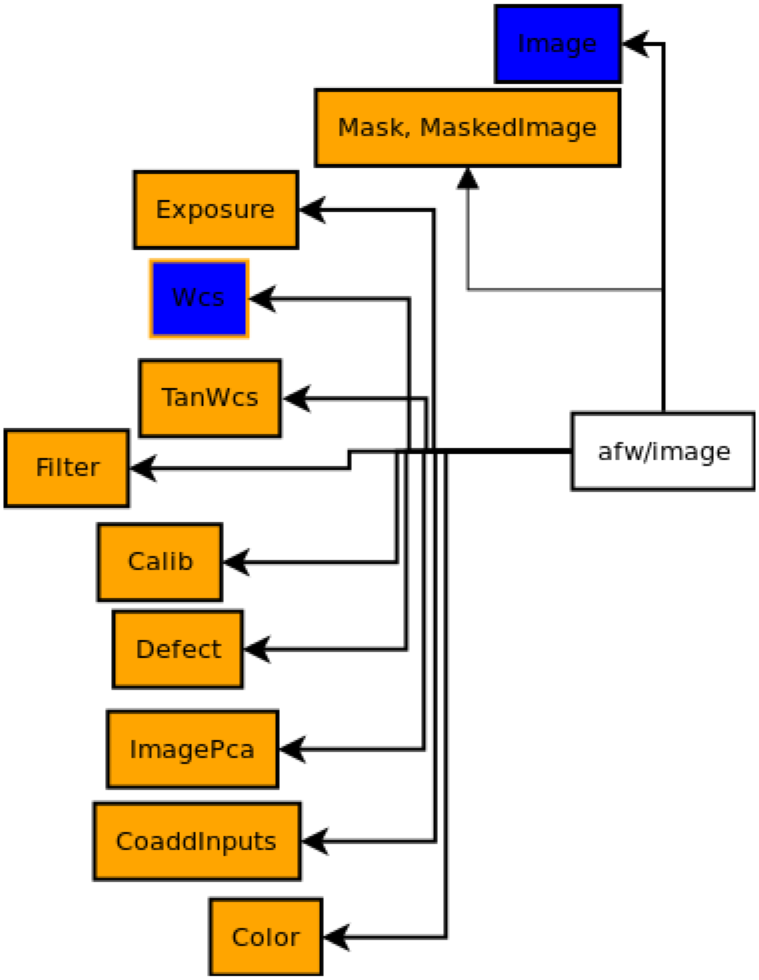
- `base`: Everything moves to the new **base** package. Even if **qserv** doesn't want it, it may someday, and all of this is very lightweight.
- `geom`: Everything moves to the new **base** package, because **qserv** needs it. Will be rewritten in C++, soon, at which point it may need some of the other things being added to **base**, even though it has no dependencies right now.
- `pex_exceptions`: Moves to **base**; lightweight, and **qserv** may want it in the future. No complaint from me if **qserv** doesn't want it and it goes to **common** instead.
- `utils`: Most things move to **base**, as it's all lightweight, and I imagine **qserv** might want to make use of the angle-string and ieee code someday. `PowFast` goes to **primitives**, since only algorithmic code will ever use it, and the temporary `unordered_map` workaround goes to common, at least as long as it lasts.



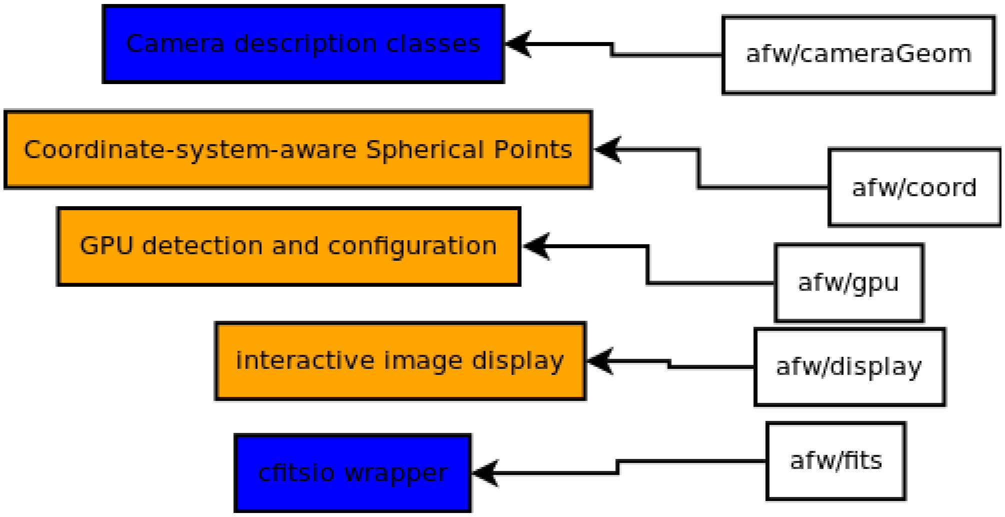
- daf_base: Everything moves to **common**
- pex_logging: Moves to **base**, where it will soon be replaced by the new logging package.
- pex_policy: Moves to **common**, for the rest of its (hopefully short) life.
- daf_persistence: The old, Formatter-based persistence framework, if it lasts this long, moves to common. The Butler is one of the major pieces of **harness**.
- ndarray: We've discussed just switching to using this as a third-party package, and this seems the time to do it.
- cat: DB ingest scripts go in **harness** (though they'll be rewritten pretty soon). I imagine we'll want the MySQL utilities here in **base**, so they can be shared with **qserv**.
- ctrl_*, pex_harness: There's a lot of code in this little box, but I think it all quite straightforwardly belongs in **execution**.
- pex_config: Everything goes to **common**, as it's needed by stuff in both **primitives** and **harness**.



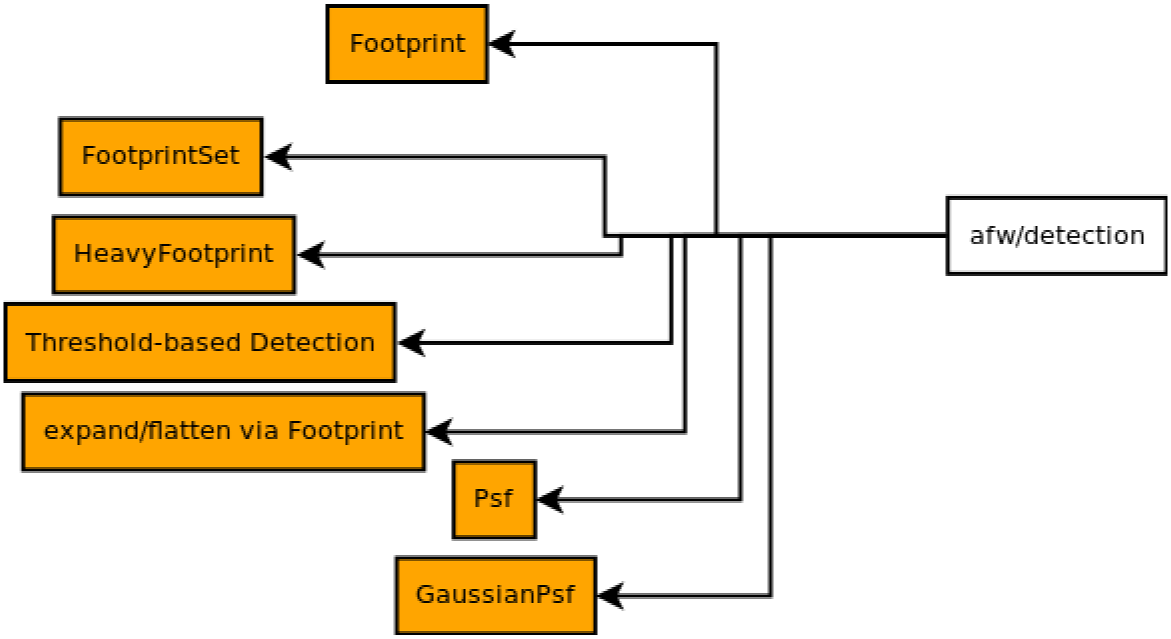
- `afw/geom`: In a change from the first version of this proposal, I'm recommending we move only the lowest-level components to **base**: points, boxes, and Angle. It's easy (and non-harmful) to remove Eigen from these, if desirable, and they're the ones we'd likely want most when interfacing with the spherical geometry code (though we may want `Polygon` for that too, but I worry about having that in **base** in case we want to persist it using `BaseTable`). Unless **qserv** decides that it does want **Eigen**, we'll probably want to put at least the transform objects in **common**, along with the closely-related ellipse objects.



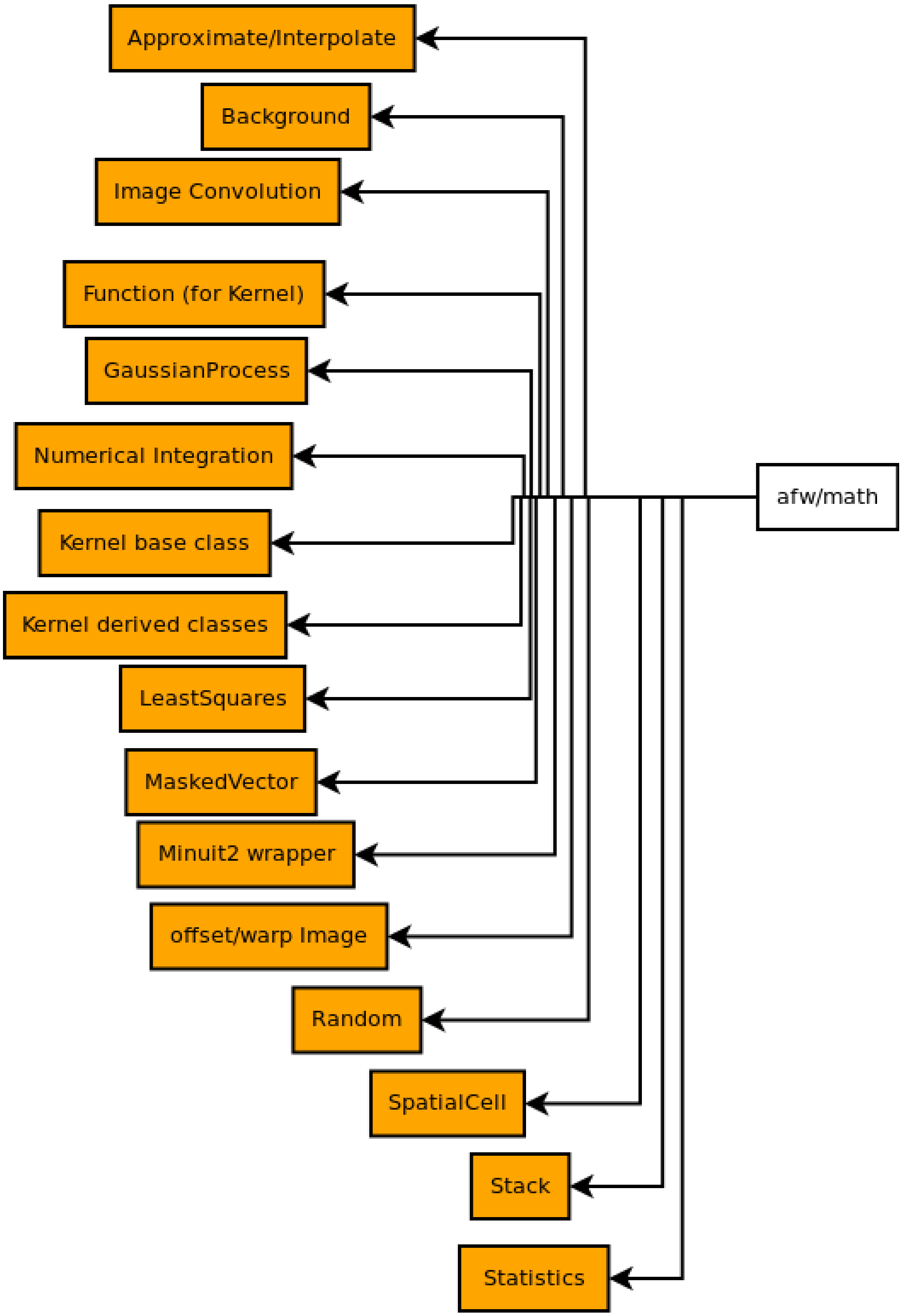
- `afw/image`: Everything goes to **primitives**, with the exception of the `Image` class itself and the `Wcs` base class, which go into common. That's because I think `Image` needs to be a basic building block of our persistence framework (i.e. more complex objects may want to save pieces of themselves as `Images`, in a more fundamental way than they might other class instances). If it turns out we really don't need that (and nothing in **harness** needs `Image` either), then we can move it to **primitives**. I'm pretty sure **harness** will have to know about the `Wcs` as well, in order to be able to organize and index data spatially (a careful observer might have noticed that I already put `skymap` interfaces in **common**, which would be impossible with access to `Wcs`). In fact, I'm a little worried that the `Butler` or the persistence framework base classes might need to know about `Exposure` as well, which would involve moving much more stuff from **primitives** down into **common**, as `Exposure` depends on a ton of other classes.


Camera description classes
afw/cameraGeom
Coordinate-system-aware Spherical Points
afw/coord
GPU detection and configuration
afw/gpu
interactive image display
afw/display
cfitsio wrapper
afw/fits

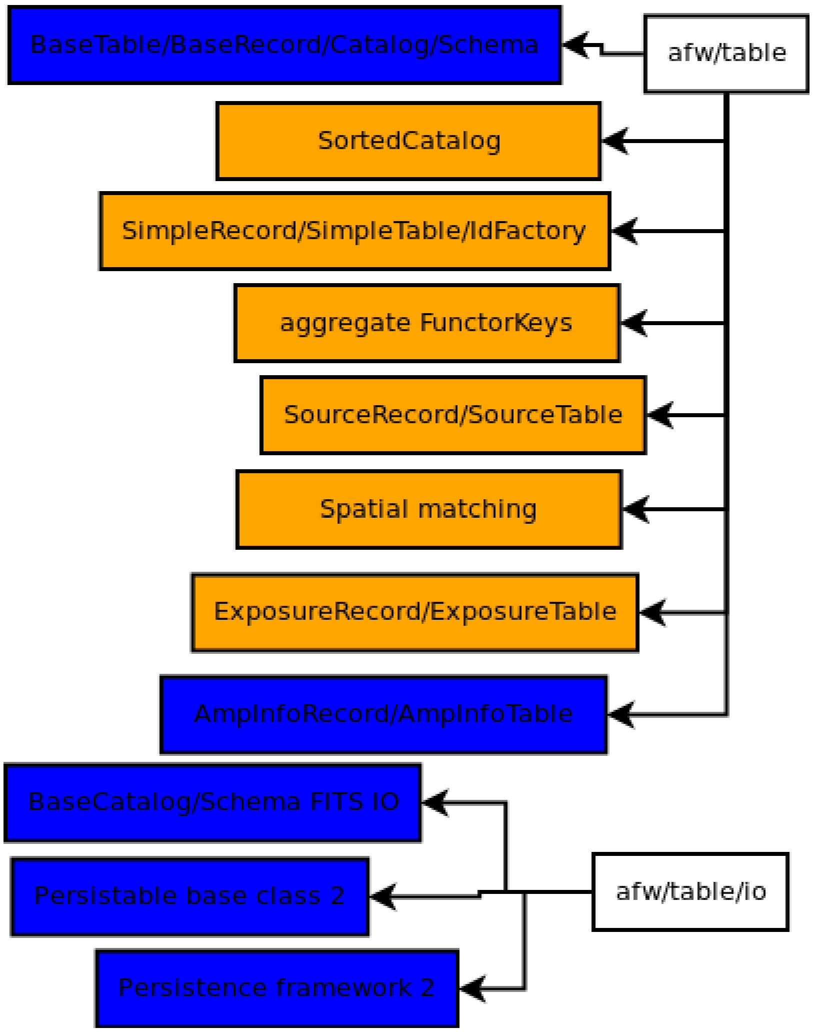
- `afw/cameraGeom`: I think this should go to common instead of **primitives** because we want to keep the interface classes needed to describe a camera in one place, and the interfaces for bookkeeping parts of that (how to create an exposure ID from a data ID, for instance) are things that **harness** needs to know about.
- `afw/coord`: Goes to **primitives**. I don't think **qserv** wants something this high-level as part of its geometry package, but if it does, we'd have to move this (and the WCS code from `afw/image`) down to **base**.
- `afw/gpu`: Goes to **primitives**.
- `afw/display`: Goes to **primitives**.
- `afw/fits`: Moves down to **common**, as FITS is going to be one of our more frequently-used persistence targets, and that means we want the ability to do FITS operations down there.



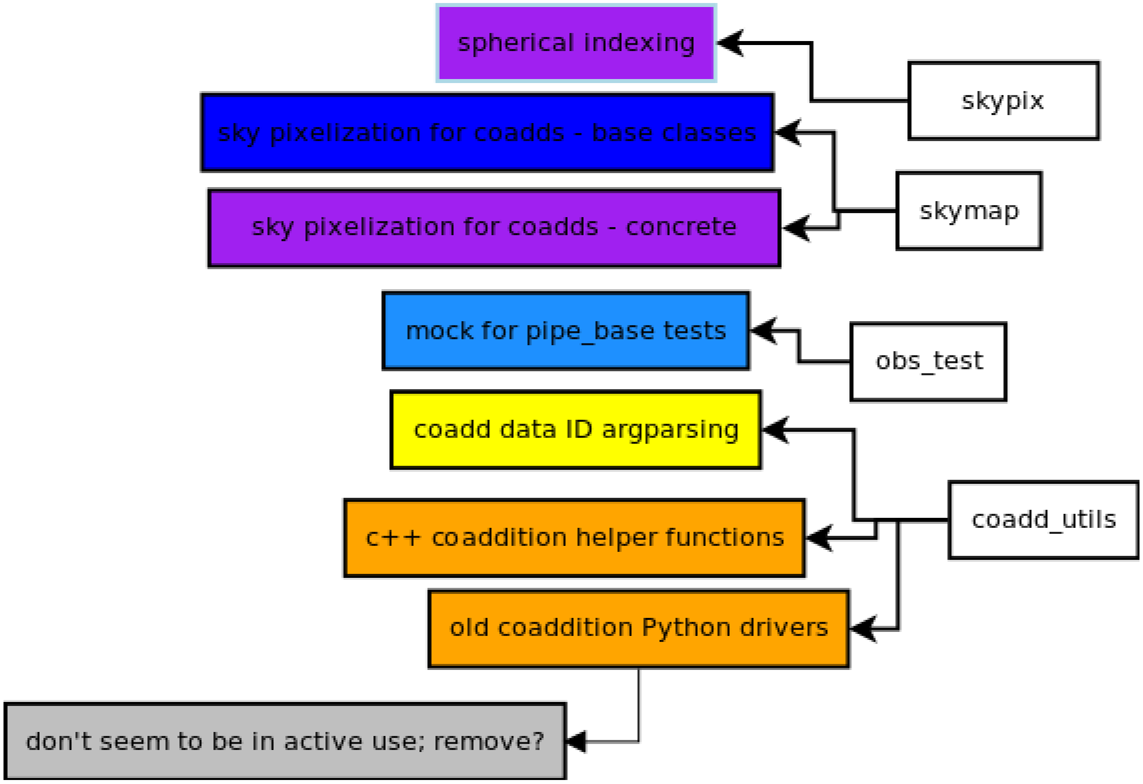
- `afw/detection`: Everything goes into **primitives**, pretty straightforwardly (though, as I'll discuss later, I don't think it all belongs in the same namespace).
- `afw/formatters` (not shown): Goes to **primitives**, until it goes away entirely.



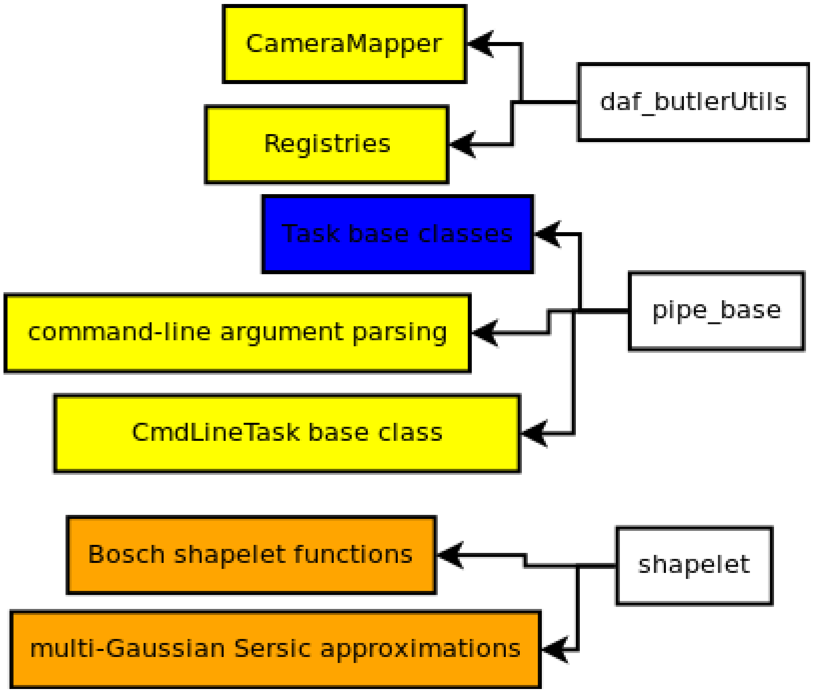
- `afw/math`: Like `afw/detection`, everything goes into **primitives**, but probably not the same namespace.


BaseTable/BaseRecord/Catalog/Schema
afw/table
SortedCatalog
SimpleRecord/SimpleTable/IdFactory
aggregate FunctorKeys
SourceRecord/SourceTable
Spatial matching
ExposureRecord/ExposureTable
AmpInfoRecord/AmpInfoTable
BaseCatalog/Schema FITS IO
Persistable base class 2
afw/table/io
Persistence framework 2

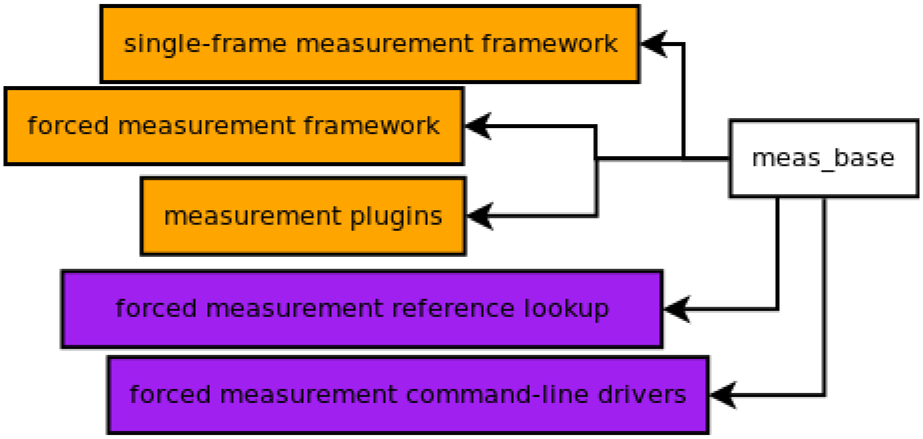
- `afw/table`: The base classes and I/O move to **common**, where they're needed for the persistence framework. The derived classes go to **primitives**, except for `AmpInfo`, which is needed in common because that's where CameraGeom went.


spherical indexing
skypix
sky pixelization for coadds - base classes
sky pixelization for coadds - concrete
skymap
mock for pipe_base tests
obs_test
coadd data ID argparsing
coadd_utils
c++ coaddition helper functions
old coaddition Python drivers
don't seem to be in active use; remove?

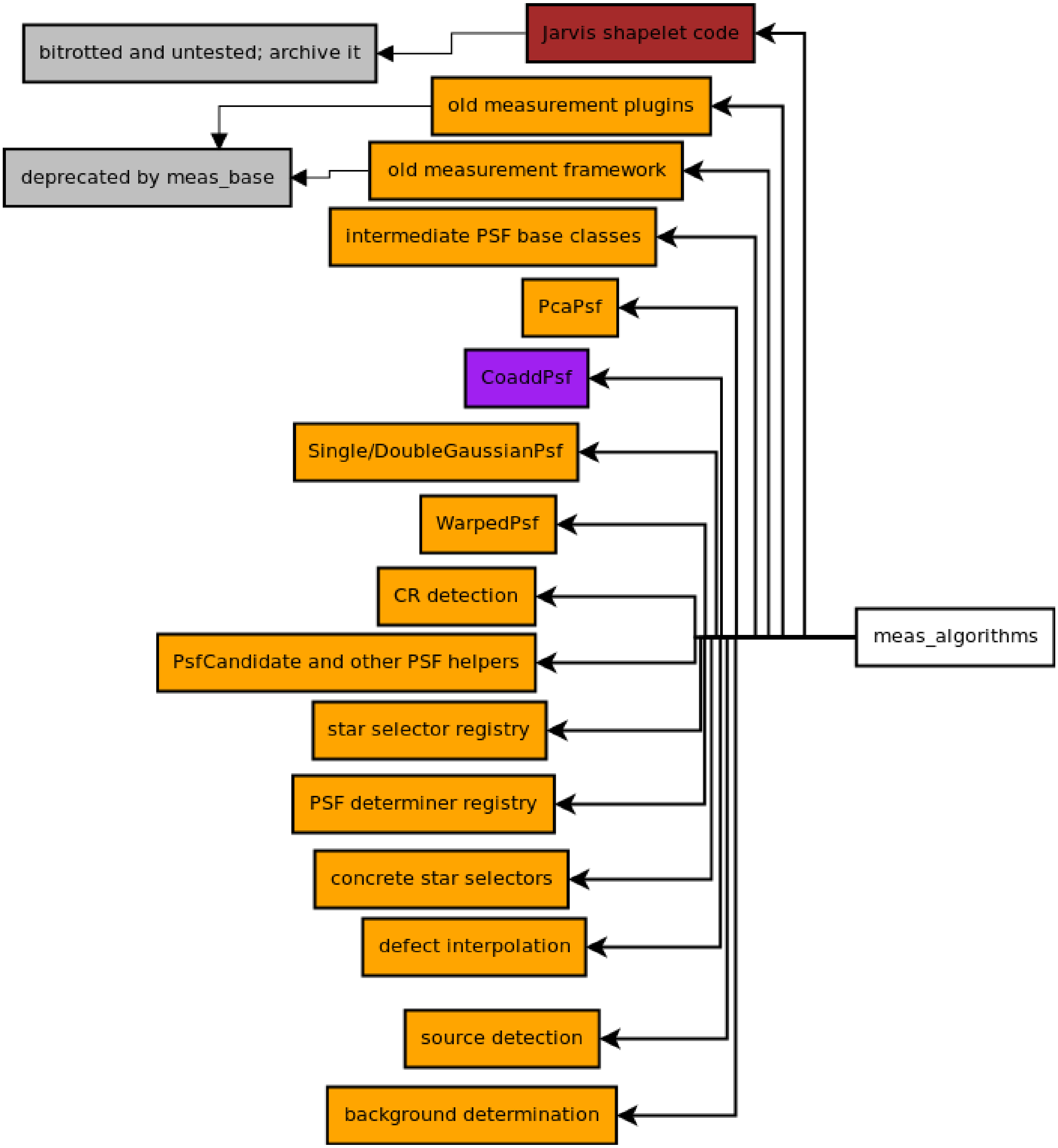
- `skypix`: goes to **pipelines**, as it's only used by **ap**. Could remove it if **ap** goes.
- `skymap`: base classes go to common, as I think the `Butler` will want to know about those interfaces. The derived classes go in **pipelines**, though **primitives** could also be a possibility. I chose **pipelines** simply because it seemed better for organizational purposes to keep them close to the high-level coadd code and the command-line driver that creates skymaps.
- `obs_test`: goes to testdata.
- `coadd_utils`: the coadd data ID argument parsing stuff (recently moved here from `pipe_tasks`) goes to **harness**. I have coaddition helper functions going to **primitives**, just because it's low-level code that doesn't need to go in **pipelines**, but putting it in **pipelines** would have the advantage of keeping it close to the higher-level coaddition code. I'm pretty sure the old coadd driver code here is unused and can just be removed.



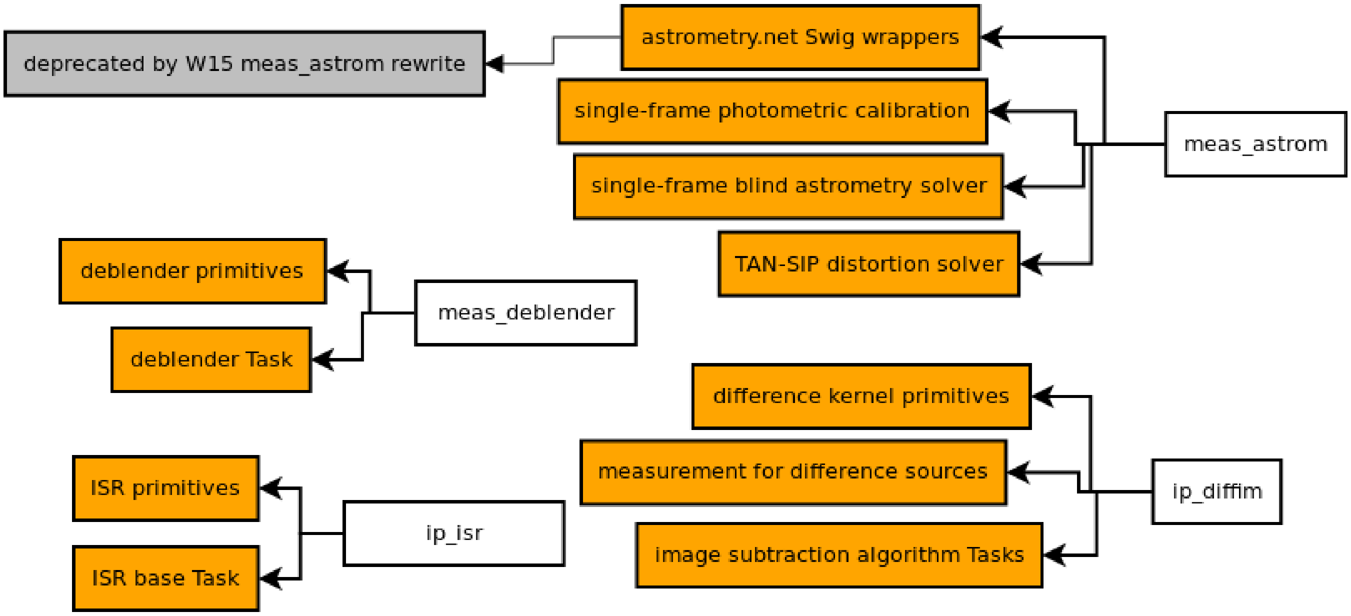
- `daf_butlerUtils`: everything goes to **harness**, where it's united with the rest of the `Butler` stuff from `daf_persistence`. This is a move we need to look into carefully before we actually try to implement it, as I'm worried some stuff here may depend on stuff I've moved to **primitives**. I'm counting on the `Butler` rewrite to address that, and I don't know if that's appropriate. If necessary, we could move some components to **pipelines**, but that could upset the idea of the **obs_*** packages depending only on **primitives**, and I think this code fits better organizationally in **harness**.
- `pipe_base`: As discussed in the overview, I have the `Task` base class moving to **common** so it can be used in `Butler`-free mid-level algorithm scripts in primitives, but the rest going to **harness**.
- `shapelet`: straightforwardly moves to **primitives**.


single-frame measurement framework
forced measurement framework
measurement plugins
meas_base
forced measurement reference lookup
forced measurement command-line drivers

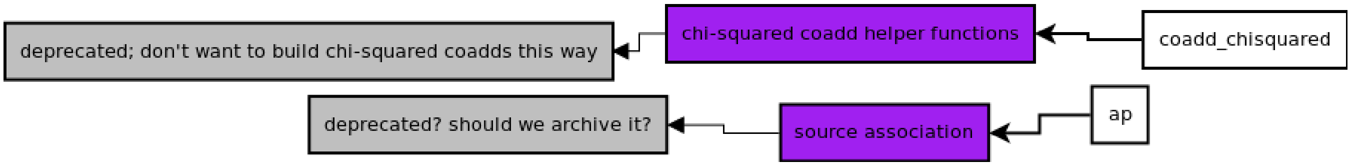
- `meas_base`: reversing a tough decision made in the first `meas_base` design review, I think we should separate the forced measurement command-line drivers from the mid-level plugin mechanism. The plugins and the measurement subtasks go to **primitives**, while the command-line drivers and reference catalog lookup goes to **pipelines**.



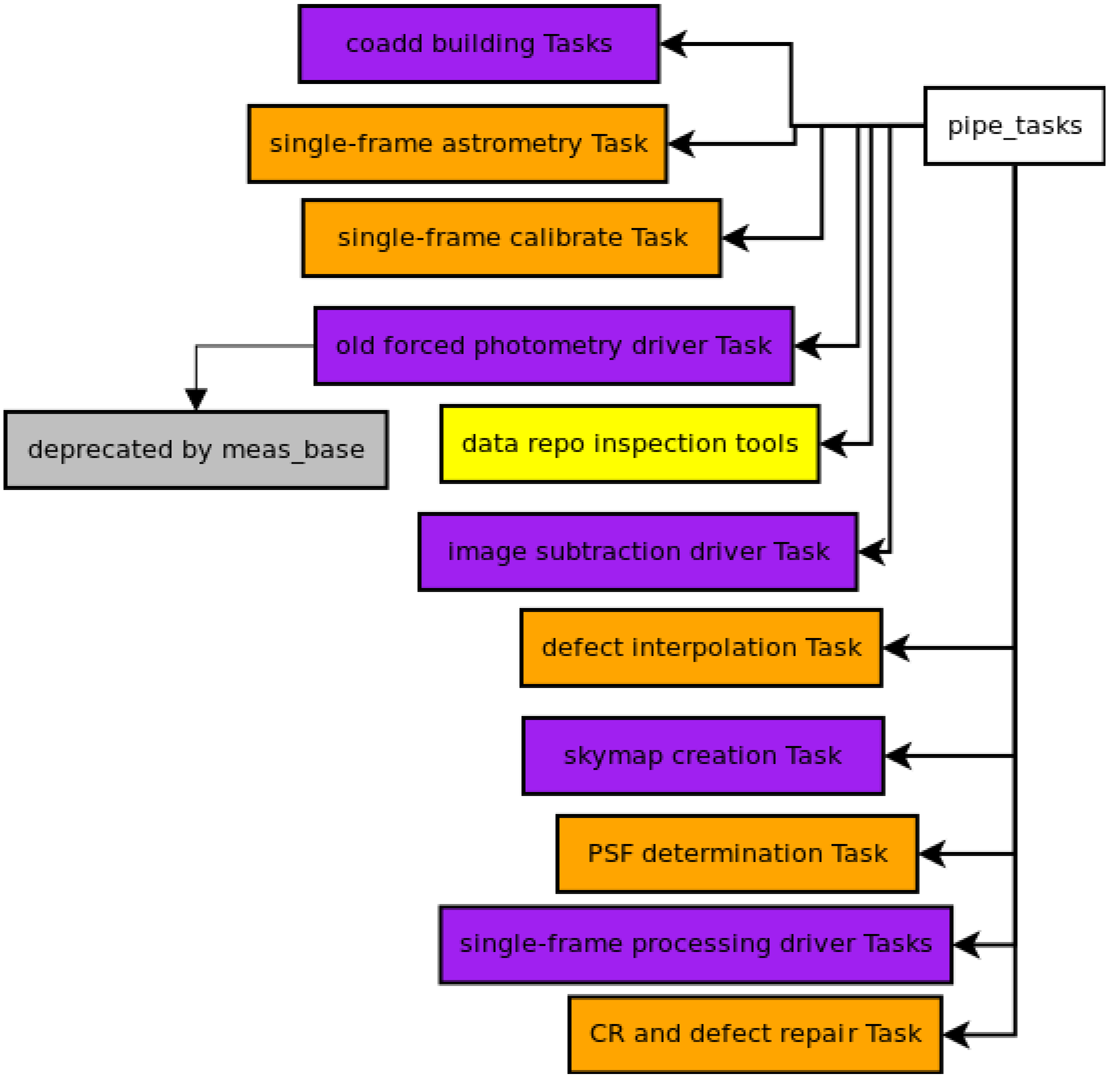
- `meas_algorithms`: most of this goes to **primitives**, but Mike Jarvis' shapelet PSF and shapelet library go to a new extension package for archival, and CoaddPsf goes to **pipelines** to live alongside the rest of the high-level coadd code that worries about the spatial relationships between exposures. Much of the C++ code here and a smaller amount of the Python will ultimately be removed, as it's part of the old measurement framework being replaced by `meas_base`.


deprecated by W15 meas_astrom rewrite
astrometry.net Swig wrappers
single-frame photometric calibration
single-frame blind astrometry solver
TAN-SIP distortion solver
meas_astrom
deblender primitives
deblender Task
meas_deblender
difference kernel primitives
measurement for difference sources
image subtraction algorithm Tasks
ip_diffim
ISR primitives
ISR base Task
ip_isr

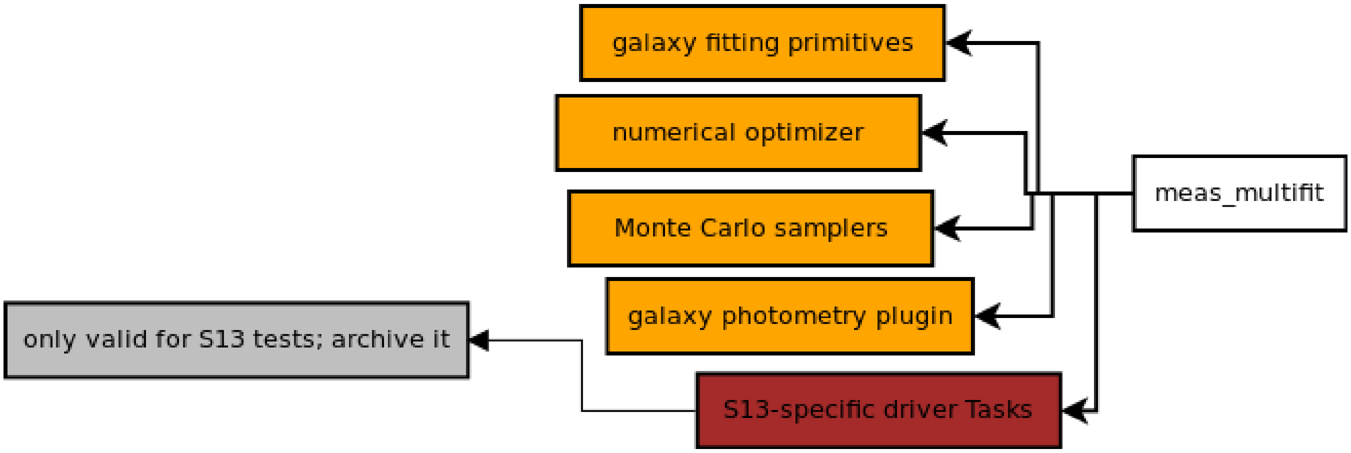
- meas_astrom: goes to **primitives**.
- meas_deblender: goes to **primitives**. May need to consider moving it to **pipelines** when we reimplement it as a multi-epoch deblender, but we'll cross that bridge when we come to it.
- ip_isr: goes to **primitives**.
- ip_diffim: goes to **primitives**. I may have missed a CmdLineTask or two that would go to **pipelines**, but I think the relevant one is in pipe_tasks.


deprecated; don't want to build chi-squared coadds this way
chi-squared coadd helper functions
coadd_chisquared
deprecated? should we archive it?
source association
ap

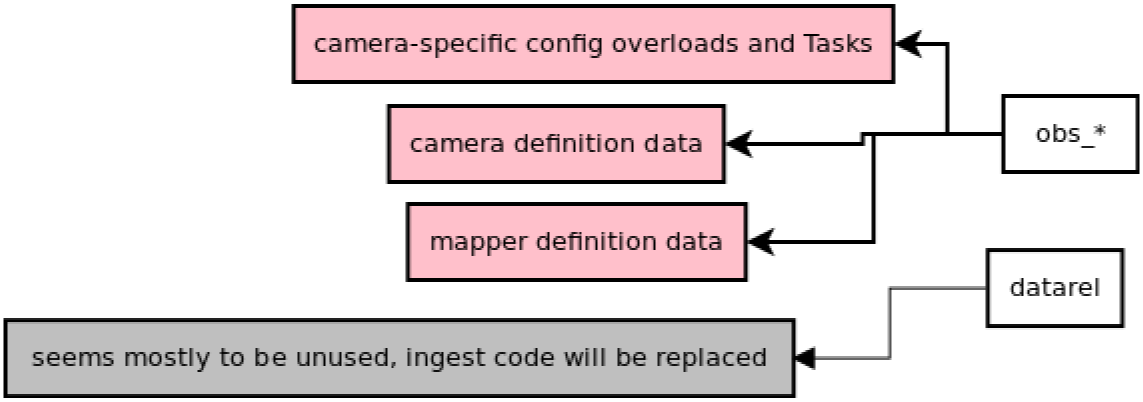
- coadd_chisquared: this goes to pipelines, if we still need it. I'm not sure we do, because I think we've agreed that if we do want to create chi-squared coadds, we'd do it differently.
- ap: this goes to pipelines. I've seen a lot of apathy about keeping it working in the presence of schema and API changes in the catalogs it reads; do we need to keep it alive? Or should we archive it and plan to resurrect it when we have a better idea of what the association pipeline will need to do.


pipe_tasks
coadd building Tasks
single-frame astrometry Task
single-frame calibrate Task
old forced photometry driver Task
deprecated by meas_base
data repo inspection tools
image subtraction driver Task
defect interpolation Task
skymap creation Task
PSF determination Task
single-frame processing driver Tasks
CR and defect repair Task

- `pipe_tasks`: The command-line driver tasks and everything related to coadd-building goes to **pipelines**, while the subtasks that are concerned with processing within a single image go to **primitives**. The data repo inspection tools, like `registryInfo.py`, go to **harness**.


meas_multifit
galaxy fitting primitives
numerical optimizer
Monte Carlo samplers
galaxy photometry plugin
S13-specific driver Tasks
only valid for S13 tests; archive it

- `meas_multifit`; Almost everything in the current `meas_multifit` goes to **primitives**, as it's really concerned with galaxy modeling, and only tangentially related to MultiFit (so it probably won't land in a namespace that includes "multifit" at all). The exceptions are the driver tasks I created to do the S13 proof-of-concept work; they're tied to a very specific set of simulations, and while they may be useful to look at in the future, we shouldn't have them in the main codebase. When we do actually implement a MultiFit measurement framework, at least some of it will have to live in **pipelines**.


camera-specific config overloads and Tasks
camera definition data
mapper definition data
obs_*
datarel
seems mostly to be unused, ingest code will be replaced

- `obs_*`: these remain essentially unchanged. I'm hoping with the move to a smaller number of packages, and the idea that `obs_*` packages should mostly depend on **primitives**, not **pipelines,** will alleviate some of the concerns that lead to an earlier proposal that we split each `obs_*` package in two.
- `datarel`: I'm no expert on this package, but it actually seems like there's nothing here we really need to keep, at least not in the long-term. I'd like someone more familiar with the package to confirm that, however.

Namespace Reorganization Proposal

TBD

Implementation

TBD

Stale/Dead Code

TBD