LSST2014
PHOENIX, AZ

LSST Software Stack

Users Tutorial

Dick Shaw

The Menu for Today

Session	Time	Presenter
LSST Software Stack User Tutorial	11:00	Dick Shaw & Friends
DM Stack Boot Camp	1:30	Paul Price
CameraGeom and All That Jazz	3:30	(Karl) Simon Krughoff

The Menu for Today

Session	Time	Presenter
LSST Software Stack User Tutorial	11:00	Dick Shaw & Friends
DM Stack Boot Camp	1:30	Paul Price
CameraGeom and All That Jazz	3:30	(Karl) Simon Krughoff

Who's in the audience today?

Scientists, Engineers, SW Developers?

What experience do you have?

Programming in Python, C++ development in Unix, survey science?

What do you hope to learn?

Scientists: how to use the stack to bulk process data

Developers: how the system works from a user point of view

Topics for This Tutorial

- What is the **LSST Stack**, and what does it do?
- How do I fetch and build it?
- How is it organized?
- How do I use it?
  - As **cmd-line apps**, tasks, library
- Some key LSST concepts (if time allows)
- How do I get help?

Topics for This Tutorial

- What is the **LSST Stack**, and what does it do?
- How do I fetch and build it?
- How is it organized?
- How do I use it?
  - As **cmd-line apps**, tasks, library
- Some key LSST concepts (if time allows)
- How do I get help?

*A strong background in **python** programming is **not a prerequisite** for this session. But it couldn't hurt.*

Disclaimers

The LSST Stack is a ***prototype*** of the DMS software system that will produce calibrated images, catalogs, alerts, and other data products from LSST exposures. It was built to demonstrate that the Project could build such a system within the specified cost and schedule envelope during construction.

- The design documents describe the system as it is intended to exist at the end of construction.
- The LSST Stack (v9.2) documentation describes the prototype system as it exists today, at the start of construction.

If you intend to use the system, bear in mind the following:

- The stack is not a finished, polished, end-user product.
- It will not work out-of-the-box for cameras other than:
  - LSST_Sim, SDSS, Subaru/HSC, CFHT/MegaCam (and soon, DECam)
- Expect to have to write some Python to make the greatest use of the Stack

Overview of the Stack

The Stack consists of ~70 packages, including:

- Software written by the DMS team (~a couple of dozen packages)
- Supporting third-party libraries, utilities, etc.

Languages:

- Python v2.7.x
- C++ (where needed for performance)

Platforms:

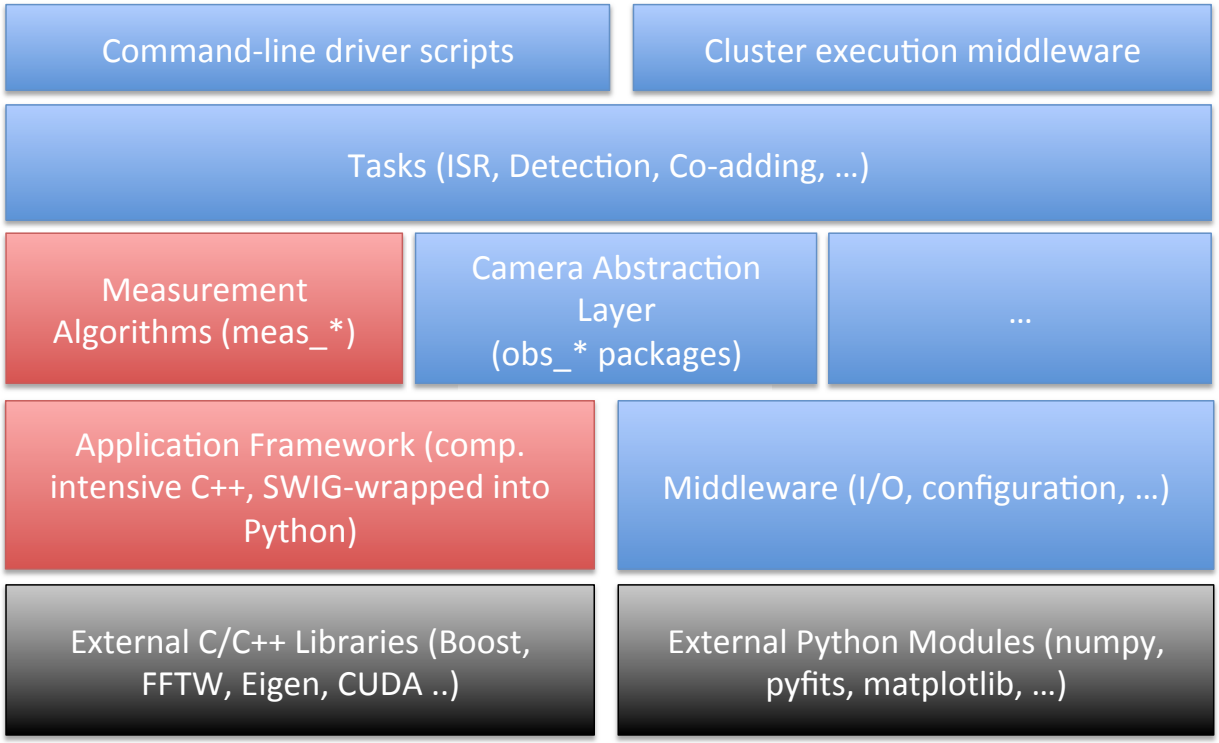
- **RHEL 6** (reference platform; soon to be CentOS)
- Mac OS X v10.7+, Ubuntu 10.4+; other Unix-y platforms

Support tools:

- SCons – build system
- Git – version control
- EUPS for package distribution, managing dependencies among multiple versions

The Stack Architecture

The LSST Stack is organized into a large number of independent packages that include applications and supporting libraries that are logically connected.



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

What does the Stack do?

- Bulk, parallel processing of imaging data
  - Instrument signature removal
  - Source detection & measurement
    - Point-sources & extended sources
  - Single-frame calibration
  - Image re-projection & co-addition
  - Source association
    - via spatial coincidence of source detections
  - Forced photometry
- Basic analysis of images
- Catalog construction & query

Installing the Stack

- Preparation
  - Be sure the prerequisite packages are installed (platform dependent)
- Run the install script

```
# Create an installation directory, then
% curl -O http://sw.lsstcorp.org/eupspkg/newinstall.sh
% bash newinstall.sh
```

- Build a fairly complete set of packages (**don't do this now!**)

```
% source loadLSST.sh   # bash users
% eups distrib install –t v9_2 lsst_distrib
```

This takes ~1.5 hr on a modest desktop.

- It helps to have a good internet connection and substantial memory
- Don't plan to do significant computing on your desktop while the system builds

See ***Building the LSST Stack from Source*** for details:
https://confluence.lsstcorp.org/display/LSWUG/Building+the+LSST+Stack+from+Source

Installing the Stack - 2

Most folks are able to install with no difficulty. However,

- Known issues
  - Mac platforms: be sure **XCode** is up to date
  - Remove/fix conflicting environment variables set in .XXXrc files
  - Review: https://confluence.lsstcorp.org/display/LSWUG/Known+Installation+Issues+for+v9.x but note that the 9.x releases are very new.
- Problems?
  - Post to lsst-dm-stack-users@lsstcorp.org

Testing the Installation

You can run a small test demo to process two SDSS fields in 5 colors

– see https://confluence.lsstcorp.org/display/LSWUG/Testing+the+Installation

```
# Create a demo installation directory, then:
% git clone http://git.lsstcorp.org/contrib/demos/lsst_dm_stack_demo.git
% cd lsst_dm_stack_demo
% setup obs_sdss
% ./bin/demo.sh
```

Verify the output, which includes:

- An ASCII catalog, calibrated images, FITS catalogs of sources

You can also visualize the output with an iPython notebook

– see http://ipython.org/ipython-doc/stable/notebook/notebook.html

```
% conda install ipython-notebook   # if needed
% export DATA_DIR=$PWD/output
% ipython notebook

# In the notebook, select StackDemo.ipynb and run it.
```

Code Cells in StackDemo.ipynb

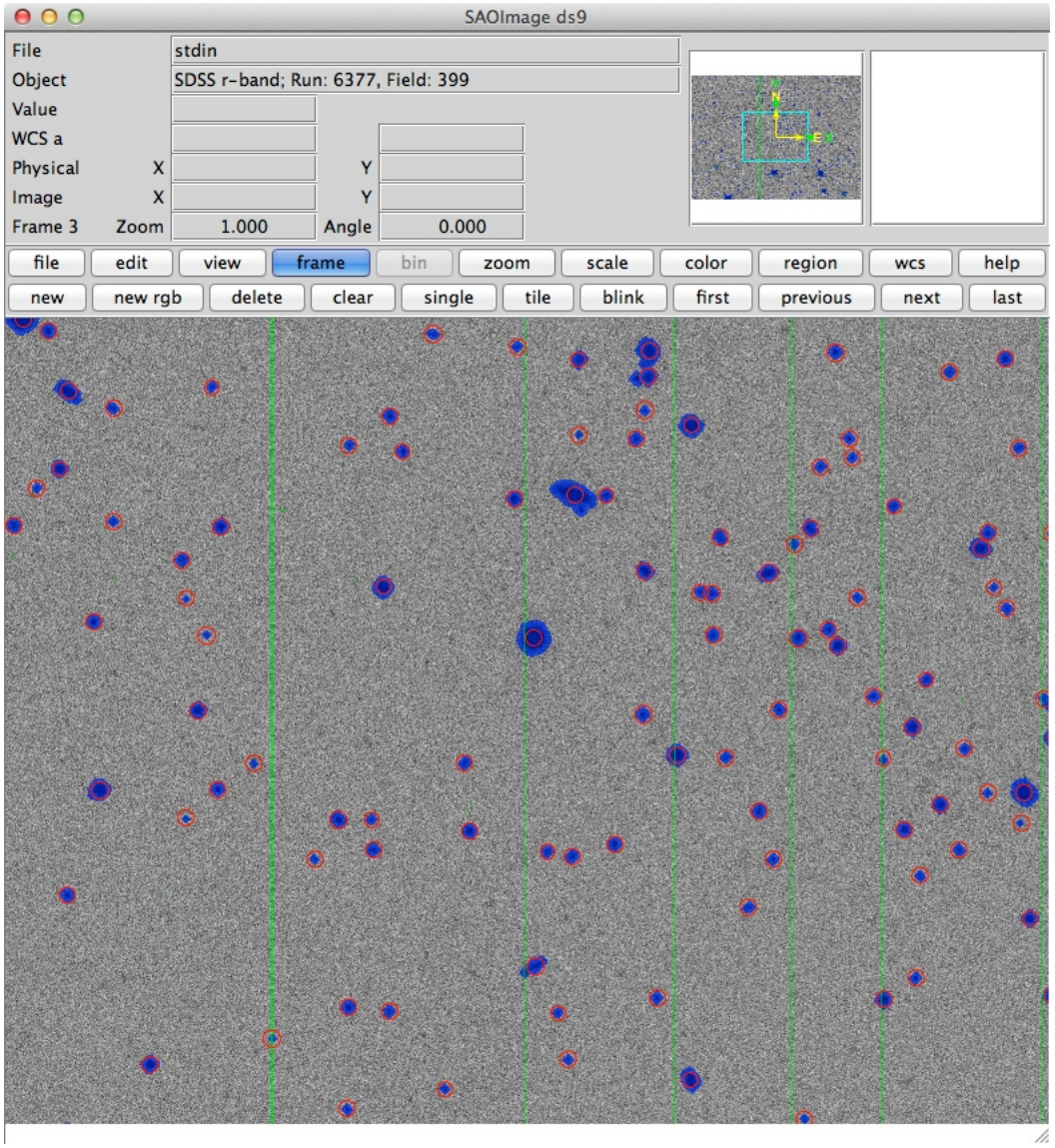
In [1]:
```
import os
import lsst.daf.persistence  as dafPersist
import lsst.afw.image        as afwImage
import lsst.afw.display.ds9  as ds9
```

In [3]:
```
# Create a butler instance to provide access to the data
DATA_DIR = os.environ.get('DATA_DIR')
butler = dafPersist.Butler(DATA_DIR)

# Fetch the r-band exposure for the specified run/field/camcol/filter
dataId = dict(run=6377, field=399, filter='r', camcol=4)
exposure = butler.get('calexp', dataId)

# display the exposure, with mask overlay
settings = {'scale':'linear', 'zscale':'', 'zoom': 1, 'mask' : 'transparency 40'}
title = "SDSS r-band; Run: 6377, Field: 399"
ds9.mtv(exposure, frame=1, title=title, settings=settings)
```

In [3]:
```
# Fetch the sources
sources = butler.get('src', dataId)

# Overplot circles at the locations of detected sources
symbol = "o"
with ds9.Buffering():
    for i,source in enumerate(sources):
        ds9.dot(symbol, source.getX(), source.getY(), ctype=ds9.WHITE, size=5, frame=1, silent=True)
```

Key:

- Green: bad pixels
- Blue: detections
- Red: sources



Using the Stack

There are many ways to use the stack; the best way depends on your purpose and your level of comfort with programming. This session will focus on command-line tasks that are used to bulk-process data. Little knowledge of python scripting is required. More advanced usage, using tasks within python classes, will be covered in the next session.

Tutorials for using the Stack are available:

- https://confluence.lsstcorp.org/display/LSWUG/Using+the+LSST+Stack

Every every process (and shell) using the LSST Stack requires initializing the LSST environment; often folks do this in their .XXXrc files.

```
% source $INSTALL_DIR/loadLSST.sh    # bash users
```

Scripts have been prepared to assist with some tutorial steps. Download them from the source code repository

```
% git clone http://dev.lsstcorp.org/git/contrib/tutorials.git
% export PS_BIN=$PWD/tutorials/<tutorial>/python
```

Using the Stack – cont'd

These are the tutorials I'll describe today:

- Generate & process PhoSim images
- SDSS Stripe 82 Data Release Production

One of the things the LSST Stack does best is **bulk processing of images** (in *parallel* on multiple cores, if your hardware is capable), and creating photometric catalogs from them. In these tutorials you will be exposed to certain key constructs of the system:

- Data Repositories
- Configuration
- Data object Identifier syntax

In the interest of time, some details will be omitted; consult the applicable pages for details. Finally, note that command line tasks offer command-line help:

```
% processCcd.py --help
```

Generation & Processing of PhoSim Images

PhoSim is a high-fidelity package for simulating full focal-plane LSST Images, given a specification of an observation, the physical properties of the camera and detectors, and a catalog of sources that cover a region of sky to be observed. This tutorial show how to simulate images from a single sensor in the LSST focal plane array for subsequent processing using the **LSST Stack**. The examples are easily extensible to multiple passbands, a larger portion of the FPA, a larger number of visits, etc.

.

PhoSim Image Generation & Processing

- Install PhoSim
  - Note: PhoSim and the LSST Stack have 2 packages in common
    - You will need to specify paths to **fftw** and **cfitsio**
- Create Simulated Images
  - Obtain a Reference Catalog
    - We'll use the *Stripe 82 Standard Star Catalog* of Ivezic et al. (2007)

```
% curl -O http://www.astro.washington.edu/users/ivezic/sdss/catalogs/
stripe82calibStars_v2.6.dat.gz
% gunzip stripe82calibStars_v2.6.dat.gz
```

  - Setup some packages

```
% setup cfitsio
% setup fftw
% setup pipe_tasks
```

See ***Process PhoSim Images*** for details:

https://confluence.lsstcorp.org/display/LSWUG/Process+PhoSim+Images

PhoSim Processing - 2

- Create Simulated Images – cont'd
  - Create PhoSim Instance Catalogs

```
% refCalCat.py --filter 'r' --ID 2014042 --path flatSED \
    --ralim -0.5 0.5  --declim -0.5 0.5 \
    stripe82calibStars_v2.6.dat > S82_r.trim
```

  - Run PhoSim

```
# Specify the sensors to be processed and a working+output directories
% export SENSORS="R22_11"
% mkdir work

% python $PHOSIM_DIR/phosim.py $PWD/S82_r.trim \
    -w $PWD/work  -o $PWD/2014042.out -c $PWD/clean.params –p $NCORES \
    -s $SENSORS
```

Wash, rinse, repeat for other passbands.

PhoSim Processing - 3

- Calibrate the images
  - Create the data repository

```
# Create a directory for the data repository
% mkdir imSim

# Bash loop to populate the repository:
% cd /your/phoSim/working/directory
for i in $( ls -d *.out); do
    cd $i
    python $PHODEMO_BIN/rename_images.py ../imSim *E000.fits.gz
    cd ../
done
```

  - Create the registry

```
% export SIM_BIN=$OBS_LSSTSIM_DIR/bin
% echo "lsst.obs.lsstSim.lsstSimMapper.LsstSimMapper" >> imSim/_mapper
% $SIM_BIN/genInputRegistry.py ./imSim -o ./imSim/registry.sqlite3
```

PhoSim Processing - 4

- Process the Images

```
% export SIM_DIR=$PWD
% $SIM_DIR/processEimage.py $SIM_DIR/imSim \
    --output $SIM_DIR/calexp_dir --clobber-output \
--id visit=2014040..2014044
```

Processing a **single sensor** takes >1 hr on (admittedly, old) Mac hardware, even with most options (including background) turned off.

The outputs of the processing include:

- **Calexp** images contain:
  - science array, mask, variance plane, PSF representation, metadata
- Photometric catalogs (per image)

SDSS Stripe 82 DRP

The 2012 DRP processed SDSS Stripe 82, which was useful for evaluating the scientific performance of the algorithms, and may be useful for your science as well. The mechanics of running a full DRP are involved, but they do give a sense of what's involved.

Access to the SDSS **`seasonFieldQuality`** table at NCSA is required. You will need a login to access the DB and build your catalogs. This restriction will be removed.

This tutorial illustrates:

- Retrieving SDSS images & creating a data repository
- Single-frame processing (excluding instrument signature removal)
- Co-addition
- Forced photometry
- Source association & catalog generation

See ***Process SDSS Stripe 82 Images*** for details:
https://confluence.lsstcorp.org/display/LSWUG/Process+SDSS+Stripe+82+Images

SDSS Stripe 82 DRP - 2

1. Create a **skymap**, or mapping from the sky to the output geometry of the Co-Adds:

```
% makeSkyMap.py $DATA_DIR/ --config skyMap,active.decRange=-1.2,1.2 \
    skyMap.active.patchInnerDimensions=2000,1907 \
    skyMap.active.patchBorder=30 skyMap.active.tractOverlap=0.0154 \
    --output tempSkyMap_dir
```

2. Identify the **patches** in the output geometry that cover it in the sky ROI:

```
% reportPatches.py tempSkyMap_dir \
    --config raDecRange="5.0,0.05,5.2,0.25" \
    --id tract=0 patch=0,0 > patches.txt
```

3. Identify the images (i.e., SDSS fields) to process:

```
% reportImagesToCoadd.py tempSkyMap_dir \
    --config raDecRange="5.0,0.05,5.2,0.25" select.strip=Both \
    select.quality=None select.rejectWholeRuns=False \
    showImageIds=True \
    --id filter=r > rawInputs.txt
```

SDSS Stripe 82 DRP - 3

4. Acquire input data, which includes:
   - Fetching pre-built astrometry_net index files for the WCS solution
     - The index files were built for the Summer 2012 DRP
   - Fetching image datasets from the SDSS archive

```
% $TUT_DIR/python/getRetrieveList.py rawInputs_r.txt retrieve.txt
% wget -r -b -R "index.html*" -np -nH --cut-dirs=1 -P ./runs \
    -i retrieve.txt
```

   - Creating a **registry** and a **mapper**

```
% genInputRegistry.py runs
% mv registry.sqlite3 ./runs
% echo "lsst.obs.sdss.sdssMapper.SdssMapper" > runs/_mapper
```

We're almost ready to begin processing. One of the great features of the image processing tasks is the capability to run on multiple cores. Define an environment variable to contain the number of available cores on your machine:

```
% export NCORES=$((sysctl -n hw.ncpu || (test -r /proc/cpuinfo && grep
processor /proc/cpuinfo | wc -l) || echo 2) 2>/dev/null)
```

SDSS Stripe 82 DRP - 4

5. Process the images with all cores. This may take an hour or so.

```
% processCcdSdss.py $DATA_DIR/ \
    --output $DEMO_DIR/calexp_dir –configfile processConfig.py \
    @rawInputs_r.txt \
    -j $NCORES
```

6. Create a results database and ingest the processing metadata.

```
% export DB_NAME="<name>_Stripe82_demo"
% prepareDb.py $DB_NAME –camera=sdss

# Create an output directory for the intermediate csv files
% mkdir $DEMO_DIR/ingestProcessed_csv_dir
% ingestProcessed.py --database=$DB_NAME \
    $DEMO_DIR/ingestProcessed_csv_dir $DEMO_DIR/calexp_dir \
    --camera=sdss
```

So we've processed all the SDSS fields. Now to make the Co-Adds, from which positions of (deep) sources will be measured.

SDSS Stripe 82 DRP - 5

7. Make the SkyMap and create warped Co-Add images.

```
# Make the SkyMap, and place in /coadd_r_dir
% makeSkyMap.py $DEMO_DIR/calexp_dir
    --config skyMap.active.decRange=0.05,0.25 \
    skyMap.active.patchInnerDimensions=2000,1907 \
    skyMap.active.patchBorder=30 skyMap.active.tractOverlap=0.0154 \
    --output coadd_r_dir

# Make the temporary Co-Add images
% makeCoaddTempExp.py $DEMO_DIR/calexp_dir \
    --output $DEMO_DIR/coadd_r_dir \
    @$DEMO_DIR/patches_r.txt \
    -j $NCORES
```

8. Assemble the Co-Add images, subtracting background along the way:

```
% assembleCoadd.py $DEMO_DIR/coadd_r_dir \
    --config maxMatchResidualRatio=1.2 maxMatchResidualRMS=0.5 \
    scaleZeroPoint.selectFluxMag0.database=$DB_NAME \
    @$DEMO_DIR/patches_r_runs.txt  -j $NCORES
```

SDSS Stripe 82 DRP - 6

9. Detect sources on the Co-Add images.

```
% processCoadd.py $DEMO_DIR/coadd_r_dir –configfile processConfig.py \
    @$DEMO_DIR/patches_r.txt
    -j $NCORES
```

10. Ingest the Co-Add sources:

```
# Create a staging directory to facilitate the process
% mkdir $DEMO_DIR/ingestCoadd_r_csv_dir

% ingestCoadd.py --camera=sdss --database=$DB_NAME \
    $DEMO_DIR/ingestCoadd_r_csv_$DEMO_DIR/dir coadd_r_dir \
    --coadd-names=deep --create-views
```

11. Perform **forced photometry** on input frames using coords from Co-Add:

```
% forcedPhot.py $DEMO_DIR/calexp_dir --output $DEMO_DIR/forcedPhot_dir \
    --configfile $TUT_DIR/config/forcedPhotConfig.py \
    --config references.dbName=$DB_NAME references.filterName=r \
    @$DEMO_DIR/forcedPhotInputs_r.txt -j $NCORES
```

SDSS Stripe 82 DRP - 7

12. Ingest the forced-source photometry.

```
% ingestForcedSource.py --camera=sdss --database=$DB_NAME \
    --coadd-name=deep --create-views $DEMO_DIR/forcedPhot_csv_dir \
    $DEMO_DIR/forcedPhot_dir
```

13. Associate sources:

```
% sourceAssoc.py $DEMO_DIR/coadd_r_dir \
    --output $DEMO_DIR/sourceAssoc_dir \
    --config measSlots.modelFlux='flux.gaussian'
```

Source association will take a ***very*** long while (perhaps hours on a desktop platform).

14. Ingest the associated sources (i.e., ***objects***):

```
% $DR_PATH/ingestSourceAssoc.py -d $DB_NAME –camera sdss –create-views \
    -j $NCORES $DEMO_DIR/sourceAssoc-csv $DEMO_DIR/sourceAssoc_dir
```

SDSS Stripe 82 DRP - 8

15. Enable the DB keys. These SQL commands take the form:

```
ALTER TABLE <table name> ENABLE KEYS;
```

Place the SQL commands in a file, and execute them:

```
% mysql $DB_NAME -h <hostname> -u <user> -p < $TUT_DIR/enable_keys.sql
```

And we are done! To review, the DRP does the following

- Basic image processing
  - calexp generation, single-frame source detection
- Co-Add construction
  - Deep source measurement
- Forced photometry on individual input calexp's
- Source Association

Some planned steps are not yet included in the production, including ubercalibration.

Key LSST Data Concepts

It is helpful to be aware of certain data concepts to understand the Stack software.

- See: https://confluence.lsstcorp.org/display/LSWUG/Key+LSST+Data+Concepts

- Measurement
- Representation of a Camera
- Data Repositories
- Tasks: the unit of code re-use
- Data Productions
  - Definition and representation of data products that DMS will produce

The first four items will be described in more detail in the next breakout: **DM Bootcamp.** A few highlights follow.

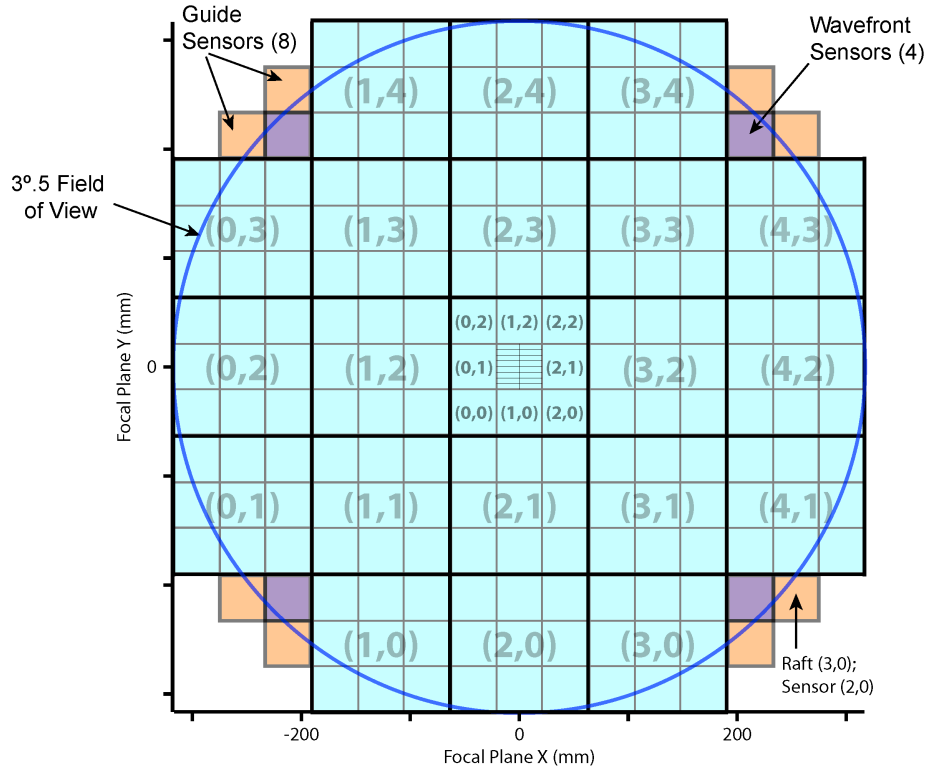
Measurement

See the confluence page:
https://confluence.lsstcorp.org/display/LSWUG/Measurement+in+the+LSST+Stack
where measurement algorithms are described. This will change significantly in the next few months.

- Positions
  - Source detection & measurement
- Background
- Source photometry
  - PSF, aperture photometry, etc.
  - Shapes of extended sources
  - Overlapping sources
- Photometric Quality

Representation of a Camera

- Camera model includes
  - Sensors tiling an FPA
  - Camera geometry
- Camera geometry
  - Coordinate transforms
  - Location of sensors in FPA
  - Sensor attributes (e.g. pixel size, amps, overscan location, etc.)
- Organization of data
  - Software for mapping data into memory
- Rolling your own camera
  - See KSK talk this pm



Data Repositories

The **butler** provides a convenient API for efficient access to large collections of data. Data are organized into hierarchies in the file system; the organization is specific to a particular camera. A data repository consists of the following

- An organization of data in a file system
  - Raw data vs. processed
  - Calibration reference files (flats, etc.)
- A registry (sqlite3) with an inventory of metadata
  - Filter used, epoch of exposure, etc.
- A `_mapper` file to cue the system to which camera mapper to use
  - `lsst.obs.sdss.sdssMapper.SdssMapper`
- A directory of astrometry_net index files
  - Covering the area of sky applicable to the input data

Tools must be built for each camera to construct a repository.

Summary of Resources for Users

- LSST Software User Guide (on Confluence site):
  - https://confluence.lsstcorp.org/display/LSWUG/LSST+Software+User+Guide
  - Glossaries for DMS, Astronomy
  - Confluence Questions: https://confluence.lsstcorp.org/questions/topics
- Source code repositories:
  - https://dev.lsstcorp.org/cgit/
- E-mail lists *@lsstcorp.org
  - lsst-dm-stack-users, lsst-data
- DOxygen source code documentation (tasks, APIs, etc.):
  - http://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/
- JIRA issue tracker:
  - https://jira.lsstcorp.org/browse/DM/
- Electronic
  - HipChat, GoToMeeting, Google, phone