VERA C. RUBIN OBSERVATORY

Processing LoVoCCS Clusters in the LSST Science Pipelines

Anthony Englert

NSF NOIRLab AURA SLAC U.S. DEPARTMENT OF ENERGY

LSST Science Pipelines (LSSTPipe)

- LSST will collect on the order of petabytes of data every year, eventually reaching ***~0.5 zettabytes*** by Y10.
  - Data must be readily accessible and processed efficiently
  - Requires communication between Rubin Data Management (Rubin-DM) and science community
- LSST Science Pipelines (LSSTPipe) is actively developed with community contributions
  - Includes methods to built local repositories which will be applied to LSST (The Butler)
  - Includes the algorithms which will be used on LSST (Pipetasks and Pipelines)
- Our work is a precursor dataset which presents an opportunity to match LSST Y1-2 Depth

EX: load LSSTPipe

- Really, LSSTPipe is a shell which includes a miniconda installation and several specific commands
  - For gen3 processing, we store the pipelines in:
    `/oscar/data/idellant/lsst_stack_v26_0_0/`
  - To enter the shell, navigate to an installation (use v26.0.0) and run:
    `$ source loadLSST.bash`
  - Then, to setup the relevant packages, run:
    `$ setup lsst_distrib`
- This shell happens to include every python module we use for LoVoCCS, so once this is setup we're good to go.
  - If you want to experiment, I've installed ipython in every installation

LSSTPipe

- The butler is responsible for adding data to, and extracting data from, a repository
  - For LoVoCCS, the backend is powered by sqlite3
  - Eventually we'd like to move to postgres… but there are some administrative issues we need to hash-out
- Data is processed via *pipetasks*, which themselves are connected to form a *pipeline*
  - LoVoCCS has a central pipeline, DRP-LoVoCCS.yaml, with default configs set appropriately

EX: query the butler

- Now that we have LSSTPipe setup, let's use the butler to print some information about datasets and a collection.
  - A dataset is a unique piece of data characterized by a unique data-id
    - Data-id usually includes instrument, band, visit (or skymap/patch for coadds)
    - These are the *dimensions* of the dataset
  - A collection is a set of datasets which are stored together

- Both of these can be queried by the butler, e.g. navigate to gen3_processing/A85 and try running:

```
$ butler query-collections repo/repo DECam/processing/coadd*

$ butler query-datasets repo/repo --collections DECam/processing/coadd_3a
--where "instrument='DECam' AND patch=66 AND band='r' AND skymap='A85_skymap'
deepCoadd
```

- Run `$ butler -h` for more methods provided by the butler

EX: query the butler

Command-line can only get you so far. For getting data you can manipulate, the butler can also be accessed via Python.

```
from lsst.daf.butler import Butler

butler = Butler('repo/repo')

query = butler.registry.query_datasets('deepCoadd',dataId={'instrument':
'DECam'},collections='DECam/processing/coadd_3a')

for i in query:
    print(i)
```

From here, the secret to success is digging through the API here for what you need.

Collections

- The repository stores data in a hierarchical manner:
  - datasets of specific dataset-types are used to store images, catalogs, etc.
  - collections are used to store many datasets and have a few types
    - RUN – directly store datasets, created by pipetasks
    - CHAINED – hold a collection of collections
    - CALIBRATION – store calibrations certified to a specific timeframe
    - TAGGED – created by gathering collections together

- RUN and CHAINED collections represent file-paths from the repository, but TAGGED and CALIBRATION are stored within the sqlite
  - When running a query, you can view the type of collection:

```
butler query-collections repo/repo DECam/calib/*
```

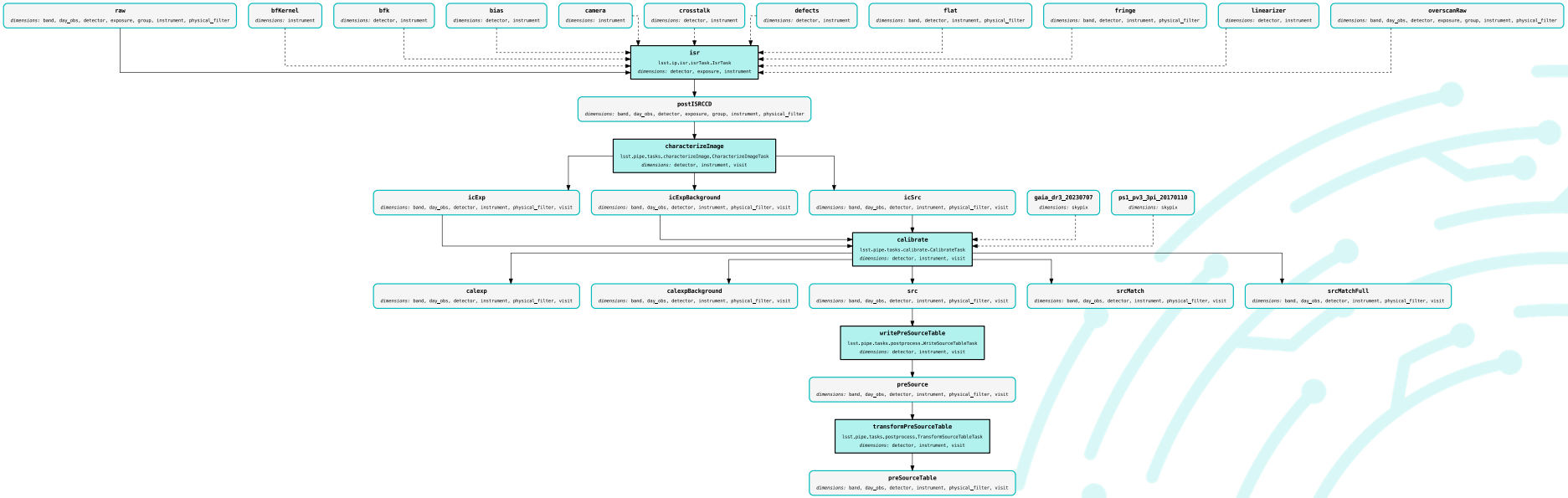
Pipetasks & Pipelines

- LSSTPipe runs processing in the form of pipetasks and pipelines
  - pipetasks are Classes in Python which operate on datasets in a collection
  - pipelines are .yaml files which call a series of pipetasks and appends their inputs/outputs together appropriately
- DRP-LoVoCCS.yaml is our primary pipeline, which is broken down into a handful of separate steps, we'll break these down later

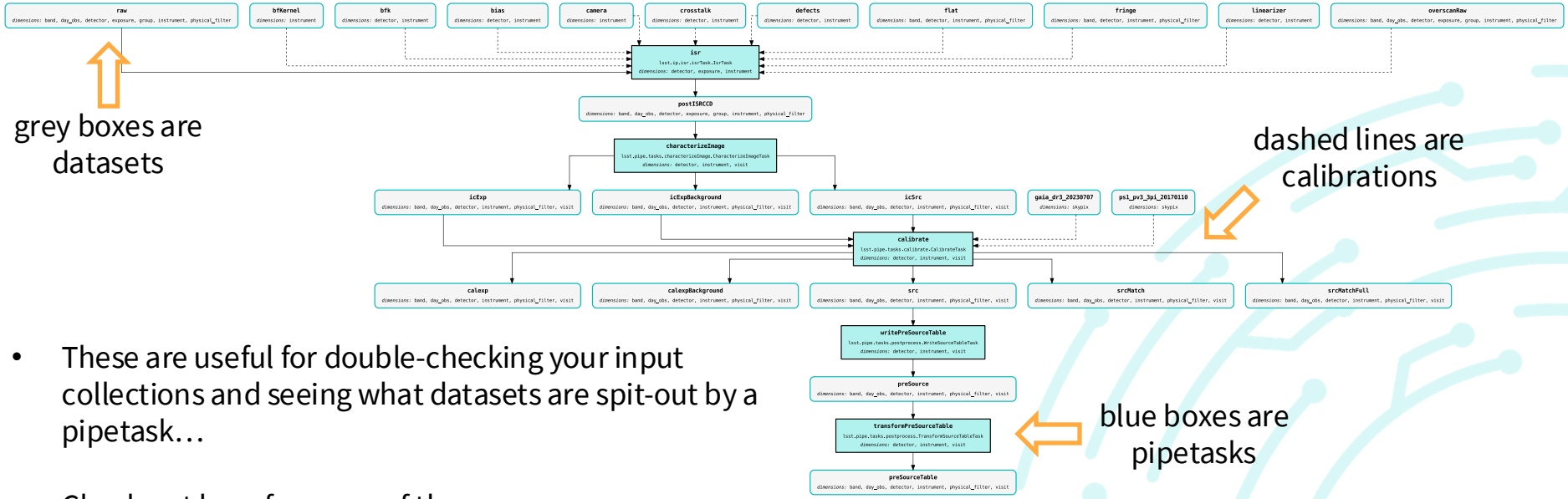
- LSSTPipe has a nice little feature which let's you visualize a pipeline, let's try it out:

```
pipetask build -p DRP-LoVoCCS.yaml#step1 -pipeline-dot tmp.dot
dot -Tpdf tmp.dot > DRP-LV_step1.pdf
```

Pipetasks & Pipelines



Pipetasks & Pipelines



grey boxes are datasets

dashed lines are calibrations

blue boxes are pipetasks

- These are useful for double-checking your input collections and seeing what datasets are spit-out by a pipetask…
- Check out here for more of them:

`…/data/Clusters/gen3_processing/automatic_pipeline_gen3`

Actually Using a Pipetask/Pipeline…

- There are two primary ways of actually running pipetasks:
  - `bps submit -p [PIPELINE] [OPTIONS] bps_config.yaml`
  - `pipetask run -p [PIPELINE] [OPTIONS]`

- BPS – Batch Processing Service
  - Distributes jobs across multiple cores and across nodes
  - Requires a config-file specifying the resource allocation
  - Good for bulk processing with long runtimes
- pipetask run
  - pipetask runs on whichever machine has submitted the script
  - Useful for quick bug-testing

- Funnily enough… we actually have a use for both of these in the full pipeline

What BPS Really Does…

- BPS was built to resolve two issues with pipetasks:
  - Writing to a repository without read/write conflicts
  - Distributing work across cores/nodes efficiently

- It fixes these by:
  - Writing the collections/datasets to the repository at the end of processing (and use a “quantum butler” during runtime)
  - Using Python wrappers to interface directly with SLURM (bps_ctrl_parsl)

- Really what bps does is:
  - Generate the qgraph
  - Create a new script (the ‘administrator’) which assigns managers/workers jobs
  - Monitors their progress and reports completion
  - ***Two batch scripts: one for the administrator and another for managers/workers***

Batch Processing Service Outputs

- BPS formats outputs in a kinda unique way… the default log dumps:
  - Output from qgraph building
  - Status of ongoing jobs
  - Filepaths for detailed per-job outputs

- Practically speaking, the way to debug is to find a failed task in the bps logs, then follow the filepath to the actual output in the per-job log

EX: Batch Processing Service

- Both bps and pipetask require that input/output collections are specified and contain the prerequisite datasets specified in the pipeline (see renders of the pipeline)

- To actually submit a job, try the following:

```
bps submit -h # this dumps some documentation

# see processing_step/process_ccd_r.sh for INPUTS and OUTPUTS

bps submit –b repo/repo -p DRP-LoVoCCS.yaml#step1 –i $INPUTS –o $OUTPUTS –d
“instrument=‘DECam’ AND band=‘r’ AND exposure.day_obs=20220526”
configs/bps_config.yaml
```

BPS Config

- BPS requires a config file which specifies the resources to request/utilize

```
wmsServiceClass: lsst.ctrl.bps.parsl.ParslService
computeSite: slurm
  site:
    slurm:
      class: lsst.ctrl.bps.parsl.sites.Slurm
      nodes: 7
      cores_per_node: 20
      mem_per_node: 100
      walltime: "48:00:00"
```

The pipeline will alternate between running just one pipeline (e.g. coadd_3a) and running one pipeline for each band (e.g. process_ccd)

- When many pipelines are run at once, use 1 nodes
- When a single pipeline is submitted, use 7 nodes
- 20 cores is a reasonable compromise btwn run-time and queue-time
- Depending on your CCV Account, you may want to distribute differently

EX: Pipetasks

- The inputs following pipetask run are much the same as with bps… but rather than specifying resource allocations in a config you can specify the number of cores via the –j option

```
# run a pipetask which will launch 20 tasks

pipetask run –b repo/repo -p DRP-LoVoCCS.yaml#step1 –i $INPUTS –o $OUTPUTS
–d “instrument=‘DECam’ AND band=‘r’ AND exposure.day_obs=20220526” -j 20
```

- The logs dumped from this file include the qgraph building along with the per-job output logs

The Quantum Graph

- Before running a processing step or a pipeline, LSSTPipe builds a “quantum-graph”
  - It’s not really “quantum” nor is it really a “graph”
  - A pipeline is broken down into ***quanta***, where a single pipetask is ran on a specific dataset
  - During this building stage, the pipeline checks that all prerequisite datasets exist and, if one is missing, fails to generate and ends the script
  - It also checks to make sure that, if $OUTPUT already exists, the input collections are the same as the input collections used to originally create $OUTPUT

- ***This is where most of the errors in the pipeline will occur… qgraph will fail due to missing collections or a mismatch between the existing $OUTPUT and $INPUT***

Building a Pipeline

- Pipelines are stored in .yaml files and specify specific tasks (with config options) in addition to subsets, for example a pipeline to carry out one task might look like

```
instrument: lsst.obs.decam.DarkEnergyCamera

tasks:
  jointcal:
    class: lsst.jointcal.JointcalTask
    config:
      connections.inputSourceTableVisit: preSourceTable_visit

subsets:
  test:
    - jointcal
```

Building a Pipeline

- Pipelines are stored in .yaml files and specify specific tasks (with config options) in addition to subsets, for example a pipeline to carry out one task might look like

```
instrument: lsst.obs.decam.DarkEnergyCamera

tasks:
  jointcal:
    class: lsst.jointcal.JointcalTask
    config:
      connections.inputSourceTableVisit: preSourceTable_visit

subsets:
  test:
    - jointcal
```

We'll almost always use DECam

This creates a task called "jointcal" which uses the JointcalTask class

We can override any default config options as well

Now by calling `pipetask run -p test.yaml#test …`, jointcal will be executed on the input datasets

Configuring a Pipeline/Pipetask

- The default config options for a pipetask can be found in the [appropriate repository on Github]
  - But pipetasks often inherit other tasks or extensions which also have config options…
  - The best way to view all configs for a given pipetask or pipeline is to dump the config

- Say I wanted to see the configs for step1 of the pipeline DRP-Merian:

```
pipetask build –p $DRP_PIPE_DIR/pipelines/DECam/DRP-Merian.yaml#step1 --
show dump-config > drp_step1_configs.txt
```

Configuring a Pipeline/Pipetask

Say you want to tell ISR to skip the bias frame… figure out which config option corresponds to that by dumping the configs, then you can either create a text file containing your override or pass it directly via the command-line

```
pipetask run -p $DRP_PIPE_DIR/pipelines/DECam/DRP-Merian.yaml#step1
-i $INPUT -o $OUTPUT -b repo/repo -c isr:doBias=False
```

Or create a file called config.py containing the line “config.doBias = False” and run

```
pipetask run -p $DRP_PIPE_DIR/pipelines/DECam/DRP-Merian.yaml#step1
-i $INPUT -o $OUTPUT -b repo/repo -C isr:config.py
```

Like with pipetask/bps, both of these have a place in our pipeline

Configuring a Pipeline/Pipetask

- One last note on this… pipetask supports –c and –C, but bps ***does not***
  - Before bps, we need to “build” the pipeline beforehand with our configs
  - Really all this does is write a new .yaml containing just the subset of tasks you specify
  - Along with setting any configs you pass or a filepath to your config file

```
pipetask build –p $DRP_PIPE_DIR/pipelines/DECam/DRP-Merian.yaml#step1
–c isr:doBias=False –s test_step1_override.yaml
```

You’ll find that the isr task will be updated

```
tasks:
  isr:
    configs:
    …
    - doBias: ‘False’
```

Debugging!

- Things will hopefully-not… maybe… probably go wrong and we'll have to use the resulting output logs to debug!
  - When using BPS, there are two levels of logs
    1. Administrator-level: qgraph building and tracking general progress
    2. Worker-level: the actual warnings/logs output by the task

- In rare cases, it may look like a job was successfully completed on the admin-level but if you investigate the worker-level logs there may be errors!
  - The true test of a job completing successfully is the existence of its output datasets on-disk

Debugging!

- Let's work through an output… the standard output from the admin is long… but it's not that scary:

First it writes a log as it builds the qgraph, this line is the input collection and other info

The butler then searches for the datasets it needs in the input collections

```
1 lsst.ctrl.bps.drivers INFO: DISCLAIMER: All values regarding memory consumption reported below are approximate and
  may not accurately reflect actual memory usage by the bps process.
2 lsst.ctrl.bps.drivers INFO: The workflow is sumitted to the local Data Facility.
3 lsst.ctrl.bps.drivers INFO: Starting submission process
4 lsst.ctrl.bps.drivers INFO: Initializing execution environment
5 lsst.ctrl.bps.drivers INFO: Initializing execution environment completed: Took 13.1883 seconds; current memory usage:
  0.212 Gibyte, delta: 0.041 Gibyte, peak delta: 0.048 Gibyte
6 lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.219 Gibyte (main), 0.000 Gibyte (largest child
  process)
7 lsst.ctrl.bps.drivers INFO: Starting acquire stage (generating and/or reading quantum graph)
8 lsst.ctrl.bps.pre_transform INFO: Creating quantum graph
9 lsst.ctrl.bps.pre_transform INFO: /users/idellant/data/Clusters/gen3_processing/lsst_stack_v27_0_0/stack/miniconda3-
  py38_4.9.2-8.0.0/Linux64/ctrl_mpexec/g6725e54719+7d18088329/bin/pipetask --long-log --log-level=VERBOSE qgraph --
  butler-config repo/repo -i DECam/processing/quality_detectors_g,DECam/processing/quality_detectors_r,DECam/processing/
  quality_detectors_i,DECam/processing/quality_detectors_z,DECam/processing/final_visit_summary_g,DECam/processing/
  final_visit_summary_r,DECam/processing/final_visit_summary_i,DECam/processing/final_visit_summary_z,DECam/calib/
  skyframes,DECam/calib/unbounded,DECam/raw/all -o DECam/processing/skycorr --output-run DECam/processing/skycorr/
  20240904T113949Z --pipeline /gpfs/data/idellant/Clusters/gen3_processing/A85/pipeline_yamls/DRP_step3d.yaml --save-
  qgraph /gpfs/data/idellant/Clusters/gen3_processing/A85/submit/DECam/processing/skycorr/20240904T113949Z/
  DECam_processing_skycorr_20240904T113949Z.qgraph --qgraph-datastore-records -d "instrument='DECam' AND band IN
  ( 'g','r','i','z' )"
10 lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:40:15.925-04:00 lsst.pipe.base.quantum_graph_builder ()
   (quantum_graph_builder.py:360) - Processing pipeline subgraph 1 of 1 with 4 task(s).
11
12 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.925-04:00 lsst.pipe.base.quantum_graph_builder ()
   (quantum_graph_builder.py:366) - Subgraph tasks: [skyCorr, makeWarp, assembleCoadd, detection]
13
14 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.926-04:00 lsst.pipe.base.quantum_graph_builder ()
   (all_dimensions_quantum_graph_builder.py:494) - Querying for data IDs with arguments:
15
16 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.926-04:00 lsst.pipe.base.quantum_graph_builder ()
   (all_dimensions_quantum_graph_builder.py:495) -   dimensions=['band', 'instrument', 'skymap', 'day_obs', 'detector',
   'group', 'physical_filter', 'tract', 'exposure', 'patch', 'visit'],
17
18 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.927-04:00 lsst.pipe.base.quantum_graph_builder ()
   (all_dimensions_quantum_graph_builder.py:496) -   dataId={'instrument': 'DECam'},
19
20 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.927-04:00 lsst.pipe.base.quantum_graph_builder ()
   (all_dimensions_quantum_graph_builder.py:498) -   where="instrument='DECam' AND band IN ( 'g','r','i','z' )",
21
22 lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:40:15.927-04:00 lsst.pipe.base.quantum_graph_builder ()
   (all_dimensions_quantum_graph_builder.py:500) -   datasets=['calexp', 'skyMap', 'raw', 'finalVisitSummary',
   'calexpBackground'],
23
```

Debugging!

“Quanta” are created, each represents a piece of data to be processed by a task

Once the qgraph is created it dumps info on the creation process

```
lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:41:05.623-04:00 lsst.pipe.base.quantum_graph_builder (all_dimensions_quantum_graph_builder.py:241) - Found 6121 overall-input dataset(s) of type 'raw'.

lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:41:05.917-04:00 lsst.pipe.base.quantum_graph_builder (all_dimensions_quantum_graph_builder.py:241) - Found 1 overall-input dataset(s) of type 'skyMap'.

lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:42:13.242-04:00 lsst.pipe.base.quantum_graph_builder (quantum_graph_builder.py:588) - Generated 133 quanta for task skyCorr.

lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:42:15.692-04:00 lsst.pipe.base.quantum_graph_builder (quantum_graph_builder.py:588) - Generated 5824 quanta for task makeWarp.

lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:42:15.942-04:00 lsst.pipe.base.quantum_graph_builder (quantum_graph_builder.py:588) - Generated 469 quanta for task assembleCoadd.

lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:42:16.028-04:00 lsst.pipe.base.quantum_graph_builder (quantum_graph_builder.py:588) - Generated 469 quanta for task detection.

lsst.ctrl.bps.pre_transform INFO: INFO 2024-09-04T07:43:03.427-04:00 lsst.ctrl.mpexec.cmdLineFwk ()(cmdLineFwk.py:882) - QuantumGraph contains 6895 quanta for 4 tasks, graph ID: '1725450182.5098953-154479'
Quanta     Tasks
------ -------------
   133       skyCorr
  5824      makeWarp
   469 assembleCoadd
   469     detection

lsst.ctrl.bps.pre_transform INFO: VERBOSE 2024-09-04T07:43:03.427-04:00 lsst.ctrl.mpexec.cmdLineFwk ()(cmdLineFwk.py:667) - Writing QuantumGraph to '/gpfs/data/idellant/Clusters/gen3_processing/A85/submit/DECam/processing/skycorr/20240904T113949Z/DECam_processing_skycorr_20240904T113949Z.qgraph'.

lsst.ctrl.bps.pre_transform INFO: Completed creating quantum graph: Took 203.6582 seconds
lsst.ctrl.bps.pre_transform INFO: Reading quantum graph from '/gpfs/data/idellant/Clusters/gen3_processing/A85/submit/DECam/processing/skycorr/20240904T113949Z/DECam_processing_skycorr_20240904T113949Z.qgraph'
lsst.ctrl.bps.pre_transform INFO: Completed reading quantum graph: Took 16.8734 seconds
lsst.ctrl.bps.drivers INFO: Acquire stage completed: Took 220.5408 seconds; current memory usage: 0.607 Gibyte, delta: 0.395 Gibyte, peak delta: 0.388 Gibyte
lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.607 Gibyte (main), 0.819 Gibyte (largest child process)
lsst.ctrl.bps.drivers INFO: Starting cluster stage (grouping quanta into jobs)
lsst.ctrl.bps.drivers INFO: Cluster stage completed: Took 0.2074 seconds; current memory usage: 0.607 Gibyte, delta: 0.000 Gibyte, peak delta: 0.000 Gibyte
lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.607 Gibyte (main), 0.819 Gibyte (largest child
```

Debugging!

BPS now does it's thing and builds a workflow where each "quanta" is assigned a "job"

At this point a new batch-job is launched utilizing the resources in `bps_config.yaml`

```
667) - Writing QuantumGraph to '/gpfs/data/idellant/Clusters/gen3_processing/A85/submit/DECam/processing/skycorr/
20240904T113949Z/DECam_processing_skycorr_20240904T113949Z.qgraph'.
60
61 lsst.ctrl.bps.pre_transform INFO: Completed creating quantum graph: Took 203.6582 seconds
62 lsst.ctrl.bps.pre_transform INFO: Reading quantum graph from '/gpfs/data/idellant/Clusters/gen3_processing/A85/submit/
DECam/processing/skycorr/20240904T113949Z/DECam_processing_skycorr_20240904T113949Z.qgraph'
63 lsst.ctrl.bps.pre_transform INFO: Completed reading quantum graph: Took 16.8734 seconds
64 lsst.ctrl.bps.drivers INFO: Acquire stage completed: Took 220.5408 seconds; current memory usage: 0.607 Gibyte,
delta: 0.395 Gibyte, peak delta: 0.388 Gibyte
65 lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.607 Gibyte (main), 0.819 Gibyte (largest child
process)
66 lsst.ctrl.bps.drivers INFO: Starting cluster stage (grouping quanta into jobs)
67 lsst.ctrl.bps.drivers INFO: Cluster stage completed: Took 0.2074 seconds; current memory usage: 0.607 Gibyte, delta:
0.000 Gibyte, peak delta: 0.000 Gibyte
68 lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.607 Gibyte (main), 0.819 Gibyte (largest child
process)
69 lsst.ctrl.bps.drivers INFO: ClusteredQuantumGraph contains 6895 cluster(s)
70 lsst.ctrl.bps.drivers INFO: Starting transform stage (creating generic workflow)
71 lsst.ctrl.bps.drivers INFO: Generic workflow name 'DECam_processing_skycorr_20240904T113949Z'
72 lsst.ctrl.bps.drivers INFO: Transform stage completed: Took 1.2499 seconds; current memory usage: 0.607 Gibyte,
delta: 0.000 Gibyte, peak delta: 0.000 Gibyte
73 lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.607 Gibyte (main), 0.819 Gibyte (largest child
process)
74 lsst.ctrl.bps.drivers INFO: GenericWorkflow contains 6897 job(s) (including final)
75 lsst.ctrl.bps.drivers INFO: Starting prepare stage (creating specific implementation of workflow)
76 lsst.ctrl.bps.parsl INFO: Writing workflow with ID=/gpfs/data/idellant/Clusters/gen3_processing/A85/submit/DECam/
processing/skycorr/20240904T113949Z
77 lsst.ctrl.bps.drivers INFO: Prepare stage completed: Took 1.2107 seconds; current memory usage: 0.610 Gibyte, delta:
0.003 Gibyte, peak delta: 0.003 Gibyte
78 lsst.ctrl.bps.drivers INFO: Peak memory usage for bps process 0.610 Gibyte (main), 0.819 Gibyte (largest child
process)
79 lsst.ctrl.bps.drivers INFO: Starting submit stage
80 lsst.ctrl.bps.submit INFO: Submitting run to a workflow management system for execution
81 parsl.dataflow.dflow INFO: Starting DataFlowKernel with config
82 Config(
83     app_cache=True,
84     checkpoint_files=None,
85     checkpoint_mode='task_exit',
86     checkpoint_period=None,
87     executors=[HighThroughputExecutor(
88         address='node1327.oscar.ccv.brown.edu',
89         address_probe_timeout=None,
90         available_accelerators=[],
```

Debugging!

These are the filepaths to the “per-job” logs

The remainder of this script monitors the progress of tasks… when a job fails there will be a message here telling you

```
143     strategy='simple',
144     usage_tracking=False
145 )
146 parsl.dataflow.dflow INFO: Parsl version: 2023.06.12
147 parsl.dataflow.dflow INFO: Run id is: 419c3bd7-a7a0-4b57-9f54-5b42c55186bc
148 parsl.dataflow.memoization INFO: App caching initialized
149 parsl.executors.status_handling INFO: Scaling out by 1 blocks
150 parsl.executors.status_handling INFO: Allocated block ID 0
151 parsl.dataflow.dflow INFO: Task 0 submitted for App skyCorr, not waiting on any dependency
152 parsl.dataflow.memoization INFO: Task 0 had no result in cache
153 parsl.dataflow.dflow INFO: Parsl task 0 try 0 launched on executor slurm with executor id 1
154 parsl.dataflow.dflow INFO: Standard output for task 0 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
    submit/DECam/processing/skycorr/20240904T113949Z/logs/skyCorr/690519/31c5bae2-d845-4a9d-b325-
    aab36d89bee0_skyCorr_690519_.stdout
155 parsl.dataflow.dflow INFO: Standard error for task 0 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
    submit/DECam/processing/skycorr/20240904T113949Z/logs/skyCorr/690519/31c5bae2-d845-4a9d-b325-
    aab36d89bee0_skyCorr_690519_.stderr
156 parsl.dataflow.dflow INFO: Task 1 submitted for App makeWarp, waiting on task 0
157 parsl.dataflow.dflow INFO: Task 2 submitted for App assembleCoadd, waiting on task 1
158 parsl.dataflow.dflow INFO: Task 3 submitted for App detection, waiting on task 2
159 parsl.dataflow.dflow INFO: Task 4 submitted for App makeWarp, waiting on task 0
160 parsl.dataflow.dflow INFO: Task 5 submitted for App assembleCoadd, waiting on task 4
161 parsl.dataflow.dflow INFO: Task 6 submitted for App detection, waiting on task 5
162 parsl.dataflow.dflow INFO: Task 7 submitted for App skyCorr, not waiting on any dependency
163 parsl.dataflow.memoization INFO: Task 7 had no result in cache
164 parsl.dataflow.dflow INFO: Parsl task 7 try 0 launched on executor slurm with executor id 2
165 parsl.dataflow.dflow INFO: Standard output for task 7 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
    submit/DECam/processing/skycorr/20240904T113949Z/logs/skyCorr/783028/
    dfd2469f-732d-435b-9461-227cad21f919_skyCorr_783028_.stdout
166 parsl.dataflow.dflow INFO: Standard error for task 7 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
    submit/DECam/processing/skycorr/20240904T113949Z/logs/skyCorr/783028/
    dfd2469f-732d-435b-9461-227cad21f919_skyCorr_783028_.stderr
167 parsl.dataflow.dflow INFO: Task 8 submitted for App makeWarp, waiting on task 7
168 parsl.dataflow.dflow INFO: Task 9 submitted for App assembleCoadd, waiting on task 8
169 parsl.dataflow.dflow INFO: Task 10 submitted for App detection, waiting on task 9
170 parsl.dataflow.dflow INFO: Task 11 submitted for App makeWarp, waiting on task 7
171 parsl.dataflow.dflow INFO: Task 12 submitted for App assembleCoadd, waiting on task 11
172 parsl.dataflow.dflow INFO: Task 13 submitted for App detection, waiting on task 12
173 parsl.dataflow.dflow INFO: Task 14 submitted for App skyCorr, not waiting on any dependency
174 parsl.dataflow.memoization INFO: Task 14 had no result in cache
175 parsl.dataflow.dflow INFO: Parsl task 14 try 0 launched on executor slurm with executor id 3
176 parsl.dataflow.dflow INFO: Standard output for task 14 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
    submit/DECam/processing/skycorr/20240904T113949Z/logs/skyCorr/956344/e510063b-
```

Debugging!

There is a timer which monitors job completion on each thread/core/job…

Once all jobs are finished, a summary will be dumped specifying any failed quanta (along with the quanta which failed due to a dependency)

```
      b2ce-00fe1f0c9098_detection_0_88_i_.stderr
63314 parsl.process_loggers DEBUG: Normal ending for _general_strategy on thread JobStatusPoller-Timer-
      Thread-139876267457936
63315 parsl.process_loggers DEBUG: Normal ending for _general_strategy on thread JobStatusPoller-Timer-
      Thread-139876267457936
63316 parsl.process_loggers DEBUG: Normal ending for _general_strategy on thread JobStatusPoller-Timer-
      Thread-139876267457936
63317 parsl.process_loggers DEBUG: Normal ending for _general_strategy on thread JobStatusPoller-Timer-
      Thread-139876267457936
63318 parsl.dataflow.dflow INFO: Task 4926 completed (launched -> exec_done)
63319 parsl.dataflow.dflow INFO: DFK cleanup initiated
63320 parsl.dataflow.dflow INFO: Summary of tasks in DFK:
63321 parsl.dataflow.dflow INFO: Tasks in state States.unsched: 0
63322 parsl.dataflow.dflow INFO: Tasks in state States.pending: 0
63323 parsl.dataflow.dflow INFO: Tasks in state States.running: 0
63324 parsl.dataflow.dflow INFO: Tasks in state States.exec_done: 6831
63325 parsl.dataflow.dflow INFO: Tasks in state States.failed: 1
63326 parsl.dataflow.dflow INFO: Tasks in state States.dep_fail: 63
63327 parsl.dataflow.dflow INFO: Tasks in state States.launched: 0
63328 parsl.dataflow.dflow INFO: Tasks in state States.fail_retryable: 0
63329 parsl.dataflow.dflow INFO: Tasks in state States.memo_done: 0
63330 parsl.dataflow.dflow INFO: Tasks in state States.joining: 0
63331 parsl.dataflow.dflow INFO: Tasks in state States.running_ended: 0
63332 parsl.dataflow.dflow INFO: End of summary
63333 parsl.dataflow.dflow INFO: Done checkpointing 1 tasks
63334 parsl.dataflow.dflow INFO: Standard output for task 4926 available at /gpfs/data/idellant/Clusters/gen3_processing/
      A85/submit/DECam/processing/skycorr/20240904T113949Z/logs/detection/0/65/i/63839d87-
      fe3b-44b1-8122-410a99ec02a0_detection_0_65_i_.stdout
63335 parsl.dataflow.dflow INFO: Standard error for task 4926 available at /gpfs/data/idellant/Clusters/gen3_processing/A85/
      submit/DECam/processing/skycorr/20240904T113949Z/logs/detection/0/65/i/63839d87-
      fe3b-44b1-8122-410a99ec02a0_detection_0_65_i_.stderr
63336 parsl.dataflow.dflow INFO: Closing job status poller
63337 parsl.dataflow.dflow INFO: Terminated job status poller
63338 parsl.dataflow.dflow INFO: Scaling in and shutting down executors
63339 parsl.dataflow.dflow INFO: Scaling in executor slurm
63340 parsl.dataflow.dflow INFO: Shutting down executor slurm
63341 parsl.executors.high_throughput.executor INFO: Attempting HighThroughputExecutor shutdown
63342 parsl.executors.high_throughput.executor INFO: Finished HighThroughputExecutor shutdown attempt
63343 parsl.dataflow.dflow INFO: Shut down executor slurm
63344 parsl.dataflow.dflow INFO: Shutting down executor _parsl_internal
63345 parsl.dataflow.dflow INFO: Shut down executor _parsl_internal
63346 parsl.dataflow.dflow INFO: Terminated executors
63347 parsl.dataflow.dflow INFO: DFK cleanup complete
63348 parsl.process_loggers DEBUG: Normal ending for cleanup on thread MainThread
```

Debugging!

- If we follow the filepath, we get the standard debug/info/failure/warning logs that used to be dumped to the .err/.out in gen2
  - When running a job, check that the output datasets specified in the pdf are created on-disk
  - If they aren't, check slurm_outputs for the administrator error logs
  - Then watch for a qgraph error OR watch for per-task failures

```
1 --> executable follows <--
2 ${CTRL_MPEXEC_DIR}/bin/pipetask --long-log --log-level=VERBOSE run-qbb repo/repo /gpfs/data/idellant/Clusters/
  gen3_processing/A85/submit/DECam/processing/skycorr/20240904T113949Z/DECam_processing_skycorr_20240904T113949Z.qgraph --
  qgraph-node-id dfd2469f-732d-435b-9461-227cad21f919
3 --> end executable <--
4 INFO 2024-09-04T08:24:20.242-04:00 lsst.ctrl.mpexec.cmdLineFwk ()(cmdLineFwk.py:882) - QuantumGraph contains 1 quantum
  for 1 task, graph ID: '1725450182.5098953-154479'
5 Quanta  Tasks
6 ------ -------
7      1 skyCorr
8 INFO 2024-09-04T08:24:20.325-04:00 lsst.ctrl.mpexec.singleQuantumExecutor (skyCorr:{instrument: 'DECam', visit: 783028,
  band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(singleQuantumExecutor.py:234) -
  Preparing execution of quantum for label=skyCorr dataId={instrument: 'DECam', visit: 783028, band: 'z', day_obs:
  20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'}.
9 VERBOSE 2024-09-04T08:24:20.344-04:00 lsst.daf.butler.datastores.fileDatastore (skyCorr:{instrument: 'DECam', visit:
  783028, band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(fileDatastore.py:1518) -
  Number of datasets found in datastore: 58 out of 58 datasets checked.
10 VERBOSE 2024-09-04T08:24:20.369-04:00 lsst.daf.butler.datastores.fileDatastore (skyCorr:{instrument: 'DECam', visit:
  783028, band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(fileDatastore.py:1518) -
  Number of datasets found in datastore: 44 out of 44 datasets checked.
11 VERBOSE 2024-09-04T08:24:20.716-04:00 lsst.daf.butler.datastores.fileDatastore (skyCorr:{instrument: 'DECam', visit:
  783028, band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(fileDatastore.py:1518) -
  Number of datasets found in datastore: 1 out of 44 datasets checked.
12 INFO 2024-09-04T08:24:20.716-04:00 lsst.ctrl.mpexec.singleQuantumExecutor (skyCorr:{instrument: 'DECam', visit: 783028,
  band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(singleQuantumExecutor.py:465) - No
  dataset artifact found for raw@{instrument: 'DECam', detector: 1, exposure: 783028, band: 'z', day_obs: 20181012,
  group: '783028', physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'} [sc=Exposure] (run=DECam/raw/all id=e61ae4cd-
  d1b4-58f1-b01e-82e08426fdc2)
13 INFO 2024-09-04T08:24:20.717-04:00 lsst.ctrl.mpexec.singleQuantumExecutor (skyCorr:{instrument: 'DECam', visit: 783028,
  band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(singleQuantumExecutor.py:465) - No
  dataset artifact found for raw@{instrument: 'DECam', detector: 3, exposure: 783028, band: 'z', day_obs: 20181012,
  group: '783028', physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'} [sc=Exposure] (run=DECam/raw/all
  id=6b6822e0-1d7d-5a68-a962-3725150a73ee)
14 INFO 2024-09-04T08:24:20.717-04:00 lsst.ctrl.mpexec.singleQuantumExecutor (skyCorr:{instrument: 'DECam', visit: 783028,
  band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(singleQuantumExecutor.py:465) - No
  dataset artifact found for raw@{instrument: 'DECam', detector: 4, exposure: 783028, band: 'z', day_obs: 20181012,
  group: '783028', physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'} [sc=Exposure] (run=DECam/raw/all
  id=92a4179b-9009-5863-bdb0-a0a819cfcd3f)
15 INFO 2024-09-04T08:24:20.717-04:00 lsst.ctrl.mpexec.singleQuantumExecutor (skyCorr:{instrument: 'DECam', visit: 783028,
  band: 'z', day_obs: 20181012, physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'})(singleQuantumExecutor.py:465) - No
  dataset artifact found for raw@{instrument: 'DECam', detector: 5, exposure: 783028, band: 'z', day_obs: 20181012,
  group: '783028', physical_filter: 'z DECam SDSS c0004 9260.0 1520.0'} [sc=Exposure] (run=DECam/raw/all
  id=1a46b588-5a99-5a98-aa54-3fed8e4f9b3f)
```

DRP-LoVoCCS

- DRP LoVoCCS is broken into “3” steps
  - step#1 is equivalent to processccd.py from gen2
  - step#2 runs creates visit-level tables and runs jointcal
  - step#3 creates the final coadds and runs measurements

- Our survey goes against the wishes of the grandmasters in LSST-DM… we often use different reference catalogs in each band
  - Many steps are broken into components (a,b,c,d) where custom configs are used to pass the correct refcat to a pipetask
  - Other divisions are to separate tasks that often fail or that, for our survey, are best to run per-band separately (e.g. isolatedStarAssociation)

DRP-LoVoCCS Step #0-1

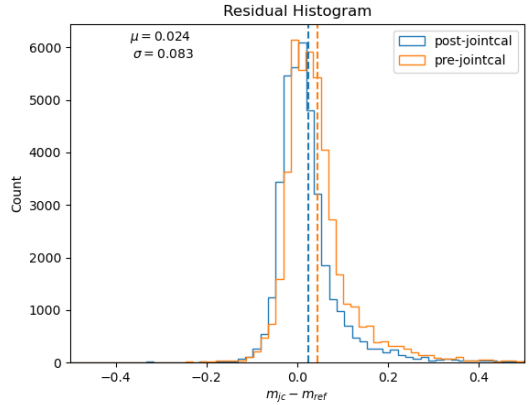
- First, we run Instrumental Signal Removal (ISR) to correct for:
  - Flat/Bias/Fringe
  - Overscan/Nonlinearity/Brighter-Fatter/Crosstalk
  - step0 generates the “crosstalk-sources” (overscan-corrected raws), step1 applies them

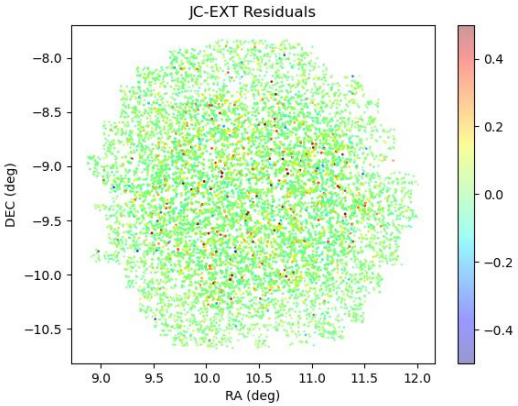
- Then, individual CCD’s are characterized (first round of measurements), background-subtracted, and eventually calibrated
  - Initial per-detector catalogs are created

- Before moving on, we pass these through check_visit/select_visit (discussed later) to select lensing-quality visits

DRP-LoVoCCS Step #2

- Step2a consolidates detector-level tables into visit-level summaries (step2a)
- Jointcal (step2b) runs on these visit-level summaries
  - jointcal models visit-to-visit variations in magnitude/position using a polynomial
  - Fits the polynomial by minimizing the astrometric/photometric chi-squared
  - The best order for jointcal is that which minimizes the residuals...
- The jointcal data is then applied to catalogs (step2d) to create final visit summaries for use during coaddition





Acronyms & Glossary

DRP-LoVoCCS Step #3

- Finally, visits are projected onto the sky and stacked
  - A psf-matched stack produced to build a model of the sky used to identify and mask transient artifacts (satellite trails, etc.)
  - Artifacts are masked and a stack with the default background subtraction enabled is created for general science (3a)
  - Sky-correction is run, and a sky-corrected stack is built (3d)

- Detection is run on the general and sky-corrected stack
  - Measurements and Deblending are only run on the general science stack
  - Final catalogs including forced photometry are created

DRP-LoVoCCS

- All of the code supporting this has been placed on GitHub.
  - Eventually we'll make the repository, lovoccs_pipe, public

- If you want to contribute, please feel free!
  - Open a pull/push request
  - Create an Issue if you're having trouble with something
  - Let us know any suggestions for new features!

Actually processing a cluster…

- The workflow is mostly carried over from the previous version
  - `run_steps_gen3.sh` contains a series of functions which use text-replacement to fill in templates
  - Each step is called a “processing_step”
  - Collections are generally saved according to the primary output of a processing_step and the band (if necessary)
    - e.g. process_ccd is saved in `DECam/processing/calexp_BAND`

- To start processing a cluster, create a folder in gen3_processing with the Cluster Name and update the field in `run_steps_gen3.sh`
  - Uncomment each function at the bottom of the script one-by-one, then run the script
  - The breakdown that follows here is complete but really only summarizes what we do… just message me if you’d like more details

First steps

1. `create_output:` Creates initial directories and copies over Python scripts along with DRP-LoVoCCS.yaml from …/`gen3_processing/automatic_pipeline_gen3`

2. `download_raw:` overhauled from gen2
   a) Launches 8 downloads in parallel
   b) Continuously monitors and resubmits downloads until all files are downloaded
   c) Should be run overnight due to bottlenecking-issues with NoirLab

3. `check_raws/move_corrupt_raws:` Opens each downloaded raw individually to make sure the file is not corrupt, then move them to a separate directory
   a) Opens each portion of the fits in the Python and watch for a crash
   b) When a crash occurs, the file is corrupt and is flagged
   c) This has been parallelized with job_arrays, runs in ~10min

Initialization and Ingestion

4. `initialize_repo`: This creates a repository located in repo/repo
   a) Assigns the instrument (DECam)
   b) Builds “curated-calibrations” (e.g. defects, PTC, etc.)

5. `ingest_data`: This ingests the raw-data, calibrations, and catalogs
   a) These have been moved from their gen2 locations
   b) Flat/Bias/Fringe/BF/Sky are located in `data/Clusters/calib_catalog_repo`
   c) Catalogs are in `data/Clusters/calib_catalog_repo/gen3_formatted_new/CLN`

Note: Both steps include `--transfer direct`, so files themselves are not copied to the new repository and instead the repo saves filepaths pointing to them which are called as needed

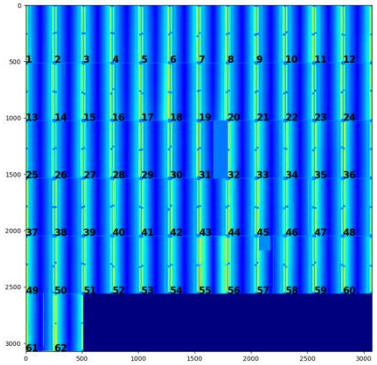
Notes on Raws/Calibs/Catalogs

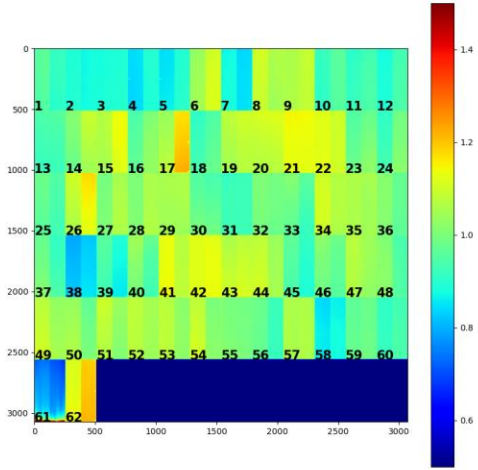
- Calibrations have all been created with gen3 currently up to 2022-12-31
  - Recently observed clusters will require new calibrations
  - Pipelines to do this have been created in `calib_catalog_repo`
  - Usually a week-long process
  - BFK/Fringe are (kinda) stable with time, only Flat/Bias need to be updated

- Raws are now queried over a larger field-of-view to catch stray-pointings overlapping a Cluster
  - Catalogs have been expanded accordingly
  - Plots of coverage are in `calib_catalog_repo/catalogs_new/CLN`

Notes on Calibs

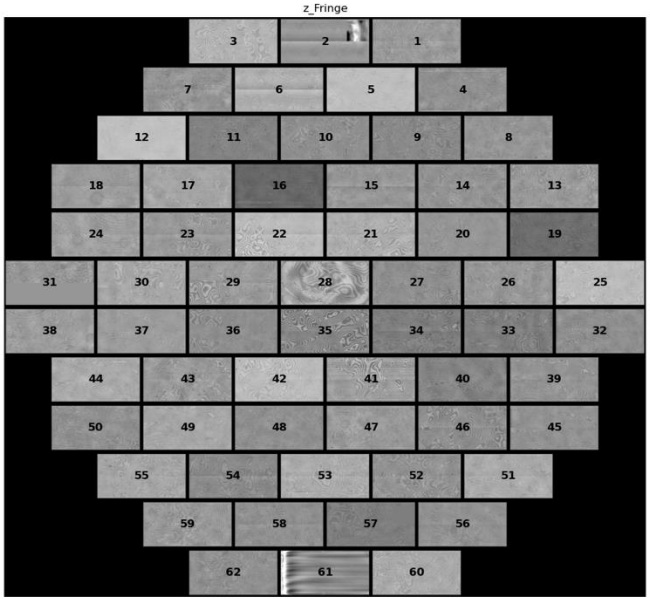
- There are several “calibration frames” which go into our analysis…
  - First are the standard bias and flats
    - Measure the instruments response with zero exposure time
    - Measure the instruments response to a uniform illumination





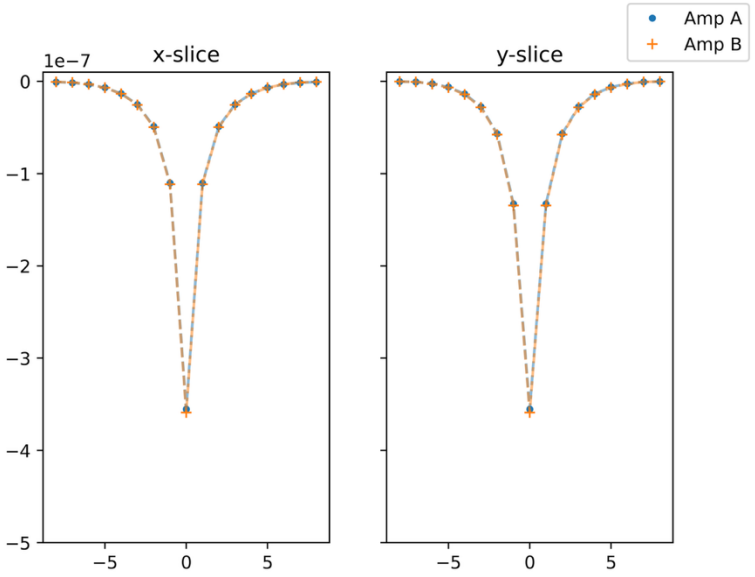
Notes on Calibs

- There are several “calibration frames” which go into our analysis…
  - First are the standard bias and flats
  - Second are the fringe frames
    - Sufficiently long wavelengths (on the order of the filter thickness) can scatter differently
    - Leads to the creation of “tree-rings” and other complex forms



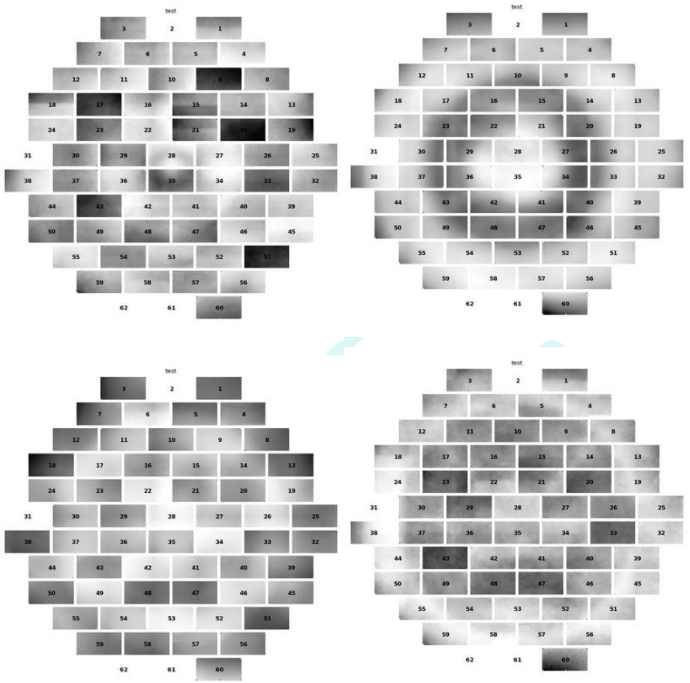
Notes on Calibs

- There are several “calibration frames” which go into our analysis…
  - First are the standard bias and flats
  - Second are the fringe frames
  - Third are the BFK’s
    - Deep CCD’s can accumulate charge in their wells sufficient to displace incident electrons
    - Brighter stars appear Fatter!



Notes on Calibs

- There are several “calibration frames” which go into our analysis…
  - First are the standard bias and flats
  - Second are the fringe frames
  - Third are the BFK’s
  - Fourth is the skyframe
    - Not a traditional calibration frame…
    - Models the response of the instrument to the sky



Process CCD

6. `process_ccd`: Runs the first portion of DRP
    1. Generates crosstalk-sources
    2. Runs ISR
    3. Carries out initial per-detector characterization/calibration

- The arguments are formatted such that you specify a reference catalog and a band for that catalog
    - All configs will be formatted per the arguments passed in run_steps:

    `process_ccd sdss,u` => run process_ccd on u-band with SDSS as the photometric refcat

- This is a step where you should adjust bps_config.yaml to use only one node
    - This lets all bands be processed in parallel
    - For Priority Accounts, all bands can run at once… otherwise two(-ish) bands at a time
    - Should be run overnight, even with Priority Accounts the run time is ~12hrs (140 cores)

Check/Select Visit

7. `check_visit`: Loads each visit and measures the PSF by fitting a Moffat Profile to reference stars and measures the shapes of stars
   a) Our cuts are specified in LV I:
      - FWHM < 6.4px/1.68” (4.4px/1.15” r-band)
      - Ellipticity < 0.33 (0.13 r-band)
   b) Plots visualizing seeing are also drawn, CCD’s with red-text fail to pass these checks

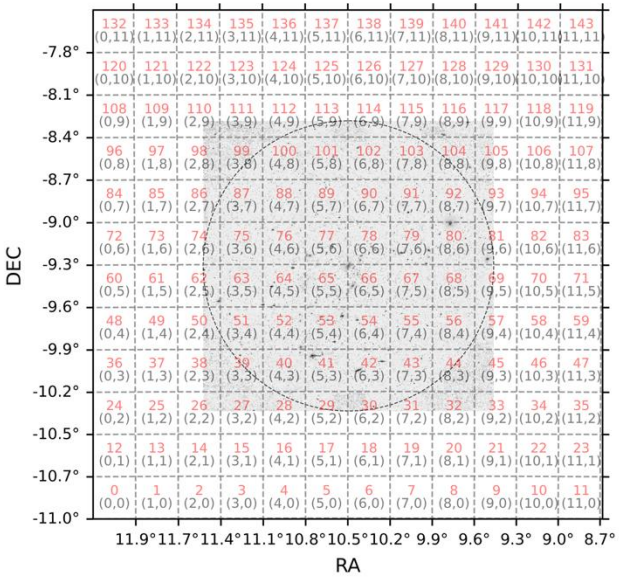
8. `select_visit`: Loops through the tables output by check_visit and creates a collection of “lensing-quality” visits, `DECam/processing/quality_detectors_BAND`
   a) Drops detectors >1.5deg from X-Ray peak (hard-coded in `select_visit.py`)
   b) Drops visits/detectors with fewer than μ-2σ/μ-3σ reference stars
   c) Drops visits with fewer than 10 “lensing-quality” detectors

`check_visit` has been parallelized, `select_visit` runs in serial but has a short runtime

Skymaps

- After `select_visit`, a skymap is created which covers the FOV of our pointings
  - Specifies where in the sky your pointings are located
  - Specifies the projection they are warped onto

- Skymaps are broken-down into patches which have a unique index
  - Following `select_visit`, a script is called which draws a skymap over the cluster
  - Background data is r-band from the DECam Legacy Survey (DECaLs)
  - Generally, our dithers/pointings thoroughly cover patches (3,3)-(8,8)



Visit-Summaries/Jointcal

9. `visit_summary:` collects the per-detector catalogs and creates tables summarizing each visit

10. `jointcal`: runs joint-calibration, which builds a joint photometric/astrometric model
    a) Technically… there is a better solution in the form of modelling the telescope optics and the atmospheric transmission (FGCM)
    b) For this to perform better than jointcal it requires a much larger FOV
    c) We use pipetasks to submit this since it's generally a single-core task

11. `final_visit_summary`: applies jointcal-derived calibrations to the `visit_summary` tables

These take ~10min, ~1hr, ~1hr (20 cores) per-band respectively.

Coaddition

12. `coadd_3a:` Creates a deepCoadd and runs detection/deblend on it
    a) Detection is a simple peak-finding algorithm which expands initial footprints based on the FWHM
    b) Deblending is powered by `SCARLET-Lite`
    c) Runs in ~3hrs (140-cores)

13. `coadd_3b`: Runs measurement on the deepCoadd
    Runs in ~4hrs (20-cores/band)

14. `coadd_3c`: Runs forced-photometry and creates final catalogs
    Runs in ~5-7hrs (140-cores)

15. `coadd_3d`: Runs skycorrection and creates a sky-corrected deepCoadd
    Runs in ~1 hour (140-cores)

Post-LSSTPipe

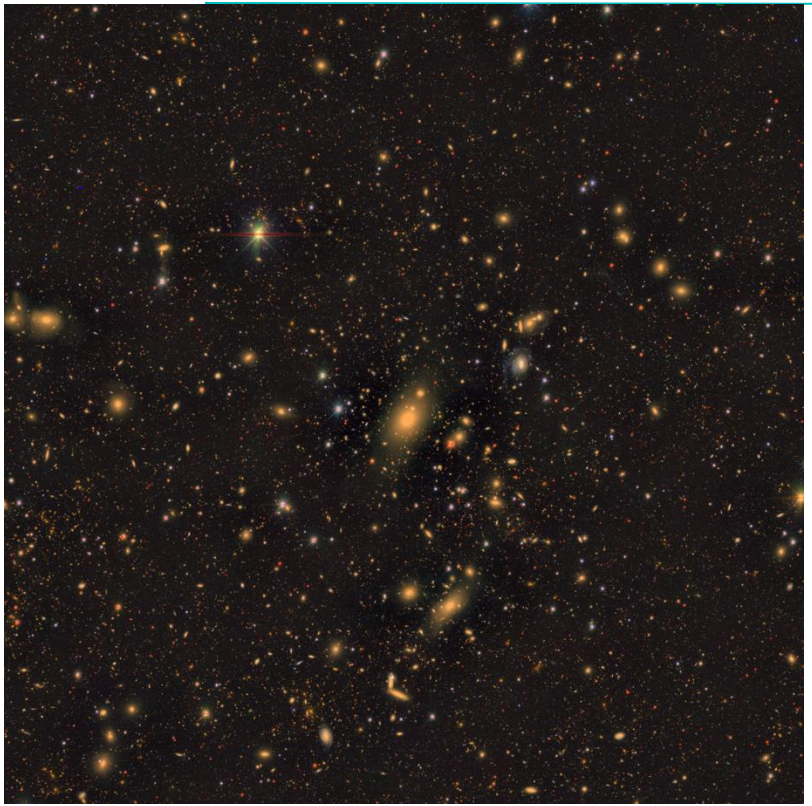
16. `export_data`: exports catalogs and codds from the repository
    a) Full catalogs are written to `read_catalog_all_output`
    b) Fits files containing each image is written to `combine_patch_color_output`
    c) irg color-png's are also written there

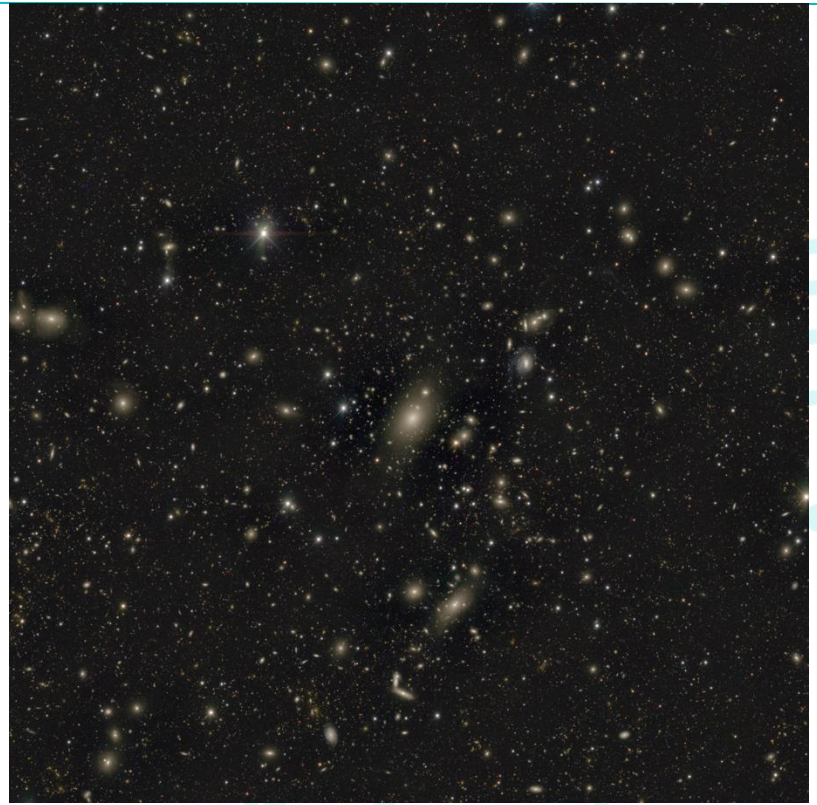
Note on coloring:

- deepCoadd png's are "relative-color-correct"
    - In the original image, if one object is redder than another than that is true after stretching as well
    - Useful for PR, but not the best for science

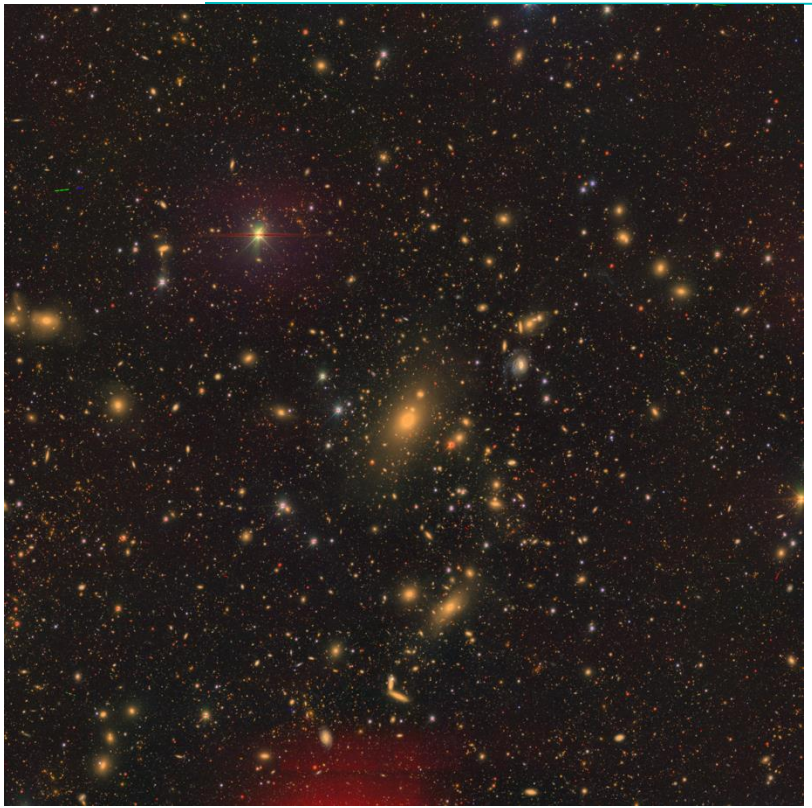
- deepCoadd_Lupton04 png's "photometric-color-correct"
    - Colors are preserved after stretching
    - Ideal for science, but sources appear bloated…

RGB Images





RGB Images w/ SkyCorrection



Post-LSSTPipe

From here, the remaining steps are completed outside of the LSP…

- These have been overhauled from their Gen2/LVI counterparts with code written from scratch
  - Improved runtime
  - Major bugs addressed
  - Fully configurable (filter-maps, filepaths, and more live in python _scripts/configs/…)
  - More easily applicable to different instruments

- There are also a few new steps that have been added
  - Metacalibration!
  - Shear calibration is now a separate step!

Post-LSSTPipe

17. `photometric calibration:` Runs a bunch of small-steps together
    a) star-galaxy separation
    b) de-reddening
    c) color-terms
    d) stellar-locus
    e) zero-point correction (color-terms griz + stellar-locus u)

18. `photo_z`: Computes photometric redshifts using BPZ

19. `shear_calibration`: Apply a user-specified shear-calibration algorithm (hsc/naïve)

Post-LSSTPipe

20. `mass_map`: Compute a map of the mass across a cluster
    - We've switched to using a direct-estimator for the mass aperture statistic, which helps achieve a higher signal-to-noise ratio (SNR)
    - By default this is computed over a ~1deg array centered on the cluster for a variety of smoothing lengths (or *apertures*)
    - A handful of filters are available to use, more to come eventually!

21. `mass_fit`: Computes the mass of the cluster
    - By default, fits a single NFW to the highest-SN peak in the mass_map
    - Eventually we'll implement multi-peak fitting…
    - Also spits-out the shear-profile and other quality-of-fit statistics

22. `quality_check`: Draw figures to verify the quality of our data

Post-LSSTPipe

23. `red_sequence`: Identify and create a density-map of the RS
    a) We select peaks in a CMD of the cluster
    b) Then smooth with a kernel of fixed bandwidth at the lens-plane (200kpc)

Metadetect!

24-28. `metadetect:` This is a bunch of steps to run metadetection for shear-calibration, it works remarkably well even for our partial implementation!

a) `meta_4a`: Create five copies of coadds in the central bands with synthetic shears
b) `meta_4b`: Re-run detection/deblend/measure/forced_photometry on sheared coadds
c) `meta_export`: Export the meta-catalogs
d) `meta_process`: Re-process sheared catalogs w. the same zero-point correction and recompute photo-z's
e) `meta_lensing`: Create a calibration using the sheared catalogs and re-run lensing code on the central patches

This works quite well, even with a temporary per-cluster calibration in-place! Eventually we need to build a calibration from the complete collection of processed LV data, but that will come later

Post-LSSTPipe

29. `gotta_blast:` Clear out the repo!
    a) We run out of disk space quite quickly while processing, so we added this step to clear intermediate files and zip-up the logs!

Congratulations, you've finished your cluster!

We keep a running log of gen3 clusters here.

If you want to contribute to the code, check-out our running TODO's for run_steps