



Synthetic Source Injection

Sophie Reed, Lee Kelvin, Josh Meyers



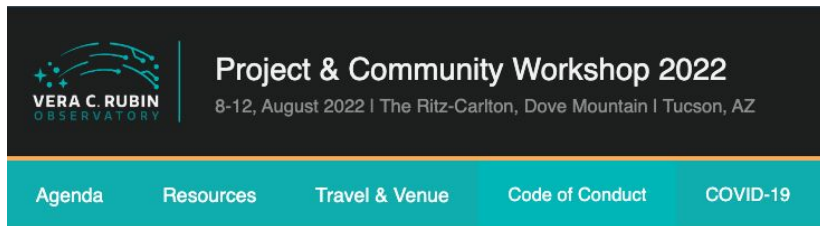
U.S. DEPARTMENT OF
ENERGY

SLAC

CHARLES AND LISA SIMONYI FUND
*** FOR ARTS AND SCIENCES ***

LSST
CORPORATION

Friendly reminders - CoC & Covid




[Home](#) » [Code of Conduct](#)


Code of Conduct

Harassment and unprofessional conduct (including the use of offensive language) of any kind is not permitted at any time and should be reported.


Rubin Observatory adheres to the principles of kindness, trust, respect, diversity, and inclusiveness in order to provide a learning environment that produces rigor and excellence.




Handshakes OK
Fold Here



Elbow/Fist Bump OK
Fold Here



I Need My Space
Fold Here



Thank you for
masking indoors!

Check name-tags for these contact
comfort level stickers.

Use the confidential email rubin2022-covid@lists.lsst.org
to request a test, report your test results, or ask questions.

Reporting bullying, harassment, or aggression.

The Rubin 2022 Organizing Committee has appointed designated contacts:

- Ranpal Gill (rgill@lsst.org)
- Andrew Connolly (ajc@astro.washington.edu)
- Melissa Graham (mlg3k@uw.edu)

Contact via email, Slack, or the Community Forum.

Friendly reminders - virtual participation



Virtual participants should be muted when they're not speaking.



In-person participants should speak into the room microphone(s), or the chair should repeat all questions into the microphone, so that the virtual participants can hear what is said.



In the Rubin2022_PCW Slack Space, all participants can use the session's channel for Q&A and discussion.

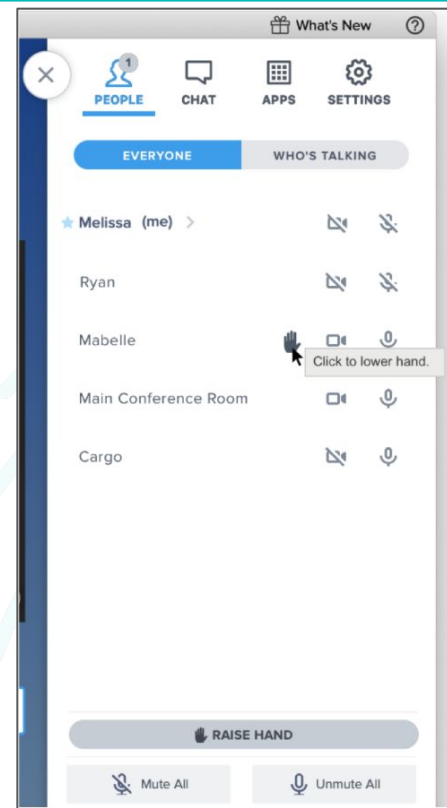
The channel name convention is, e.g.: #day1-mon-slot3a-intro-to-rubin



In BlueJeans, virtual participants should use the **BlueJeans chat** functionality to:

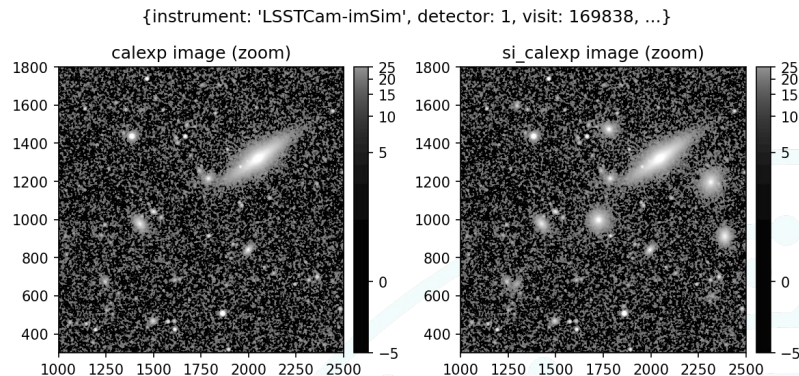
- indicate you have a question and would like to unmute, or
- type your question so that someone in the room can speak it.

BJ "raise hand" feature is hard for moderators to track, not preferred.



Introduction to SSI

- 1) Why do we want to do this anyway?
- 2) What could we do before / what is new?
- 3) Ok, but how do we do that?
- 4) What's going to be regularly available without me having to do any extra work myself?
- 5) What have we looked into so far?
- 6) Discussion of what else would be useful
 - a) Expected use cases
 - b) User feedback
 - c) Thoughts on extensions or additional features
 - d) Input on regularly processed SI datasets



Additional information → [SSI Workshop Community post](#) (June 2021)

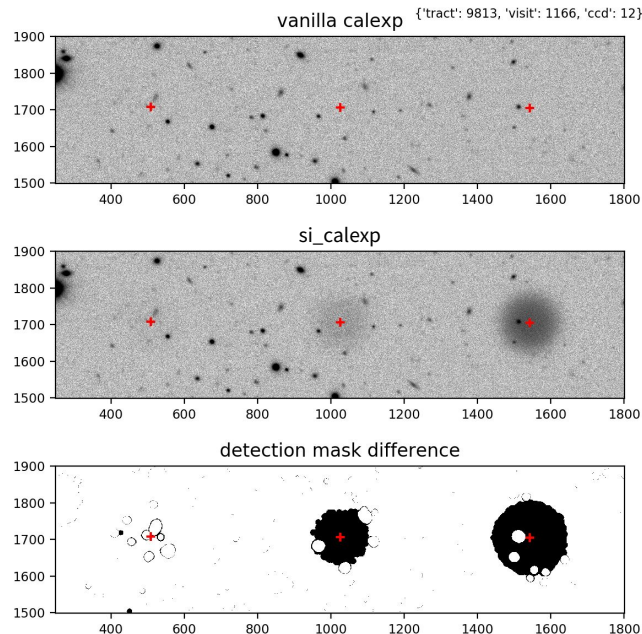
Introduction to SSI - Why?

DP0 doesn't include some object types:

- Solar system objects
- Unrealistic sources for QA stress testing
- High redshift quasars (we will show a demo injecting these sources later)

QA

- SI an incredibly useful tool for QA analysis, where the 'truth' is known.
- In particular, good to know if the measured parameters coming out are anything like those going in!



Data: Aaron Watkins / LSST:UK LSB WG

Introduction to SSI - Current Code

In gen 2 we had the ability to insert into visit level images or coadds.

→ this has now been converted to gen 3.

We want to take this opportunity to add more functionality and tidy up various things as we remove all traces of gen 2.

In particular, making it more flexible and modular.

```
class InsertFakesTask(PipelineTask, CmdLineTask):
    """Insert fake objects into images.

    Add fake stars and galaxies to the given image, read in through the dataRef. Galaxy parameters are read in
    from the specified file and then modelled using galsim.

    `InsertFakesTask` has five functions that make images of the fake sources and then add them to the
    image.

    """
    def addPixCoords(self, dataRef, calExp):
        """Add pixel coordinates to the catalog of fake sources.
        Adds an ``x`` and ``y`` column to the catalog of fake sources.
        """
    def trimFakeCat(self, dataRef, calExp):
        """Trim the fake cat to about the size of the input image.
        """
    def mkFakeGalsimGalaxies(self, dataRef, calExp):
        """Use Galsim to make fake double sersic galaxies for each set of galaxy parameters in the input file.
        """
    def mkFakeStars(self, dataRef, calExp):
        """Use the PSF information from the calExp to make a fake star using the magnitude information from the
        input file.
        """
    def cleanCat(self, dataRef, calExp):
        """Remove rows of the input fake catalog which have half light radius, of either the bulge or the disk,
        that are 0.
        """
    def addFakeSources(self, dataRef, calExp):
        """Add the fake sources to the calExp.
        """

    Notes
    -----
    The ``calExp`` with fake sources added to it is written out as the datatype ``calExp_fakes``.
    """
```

Introduction to SSI - Current Code Limitations

- Current framework is monolithic
 - Difficult to modularize how and when sources are injected
 - Changes/updates to single frame processing do not automatically filter down into `ProcessCcdWithFakesTask`
- Input catalogues must be ingested into the butler using a Python script
 - No recommended script publicly available(?), and no dedicated Rubin tool to facilitate this
 - Raises the bar for new users who wish to inject synthetic sources
- Input catalogue formats are strict: bulge+disk and/or PSF
 - Does not facilitate alternative models, e.g., bulge+disk+bar
 - Requires a complex translation layer to convert user supplied catalogues into GalSim format
- “Fakes”
 - Trying to move to the terminology ‘Source Injection’ or ‘SI’ wherever possible, or Synthetic Source Injection (SSI) in certain cases

Introduction to SSI - New Source Injection Repo

We have begun migrating all existing SI code into a new source injection repo:

https://github.com/lsst/source_injection

<https://jira.lsstcorp.org/browse/DM-34253>

source_injection

`source_injection` is a package in the [LSST Science Pipelines](#).

The Source Injection package contains Synthetic Source Injection (SSI) tasks and configuration.

Inject_engine.py - free functions

```
def inject_sources(exposure, objects, calibFluxRadius=12.0, logger=None):  
    """Inject sources into given exposure
```

Inject_base.py - base classes

```
class BaseInjectTask(PipelineTask):  
    """Class to inject into visit level images.  
    """
```

Inject_coadd.py - coadd classes

```
class CoaddInjectTask(inject_base.BaseInjectTask):  
    """Class to inject into coadd-level images.  
    """
```

Inject_visit.py - visit classes

```
class VisitInjectTask(inject_base.BaseInjectTask):  
    """Class to inject into visit level images.  
    """
```

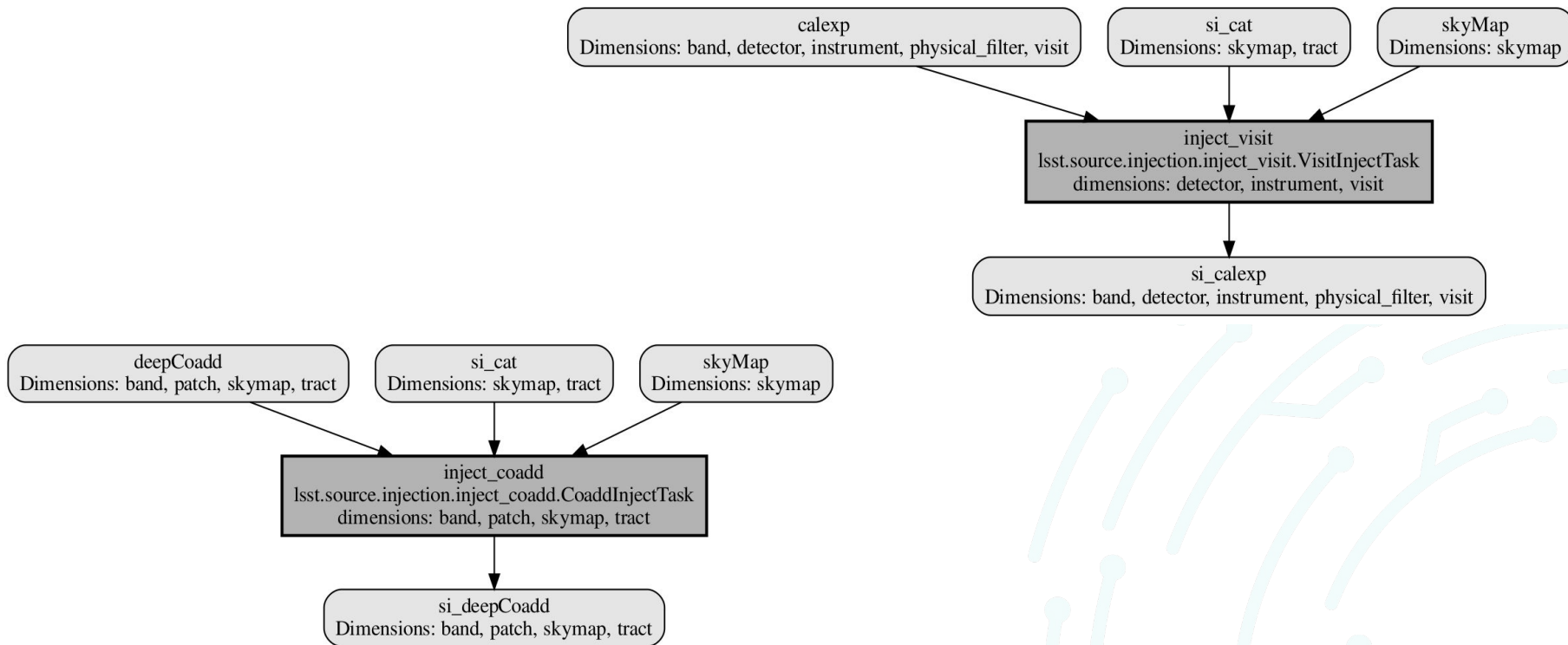

Introduction to SSI - New Source Injection Repo

All SI code will live in the new source injection repo (`$SOURCE_INJECTION_DIR`)

Notable changes:

- Code is now completely modular
 - SI can happen at multiple different points within the data reduction process
- Galaxy types are now all single-Sérsic components
 - No more specification of bulge+disk systems
 - Complex systems can still be constructed by injecting multiple sources at the same coordinates
- New data production pipeline YAMLS
 - `$DRP_PIPE_DIR/pipelines/HSC/DRP-RC2+SI.yaml`
 - `$DRP_PIPE_DIR/pipelines/LSSTCam-imSim/DRP-DP0.2+SI.yaml`
- Significant refactoring of the code base
 - Standardization of configuration arguments and input catalogue column names

Introduction to SSI - New Pipeline Subsets



Demo 1

Source Injection Into a DC2 Visit

https://github.com/lsst/source_injection/blob/tickets/DM-34253/examples/si_demo_dc2_visit.ipynb

Demo 2

Source Injection Into an HSC Coadd

How? - Step 1: Upload the SI Catalogue

```
from lsst.daf.butler import Butler, DatasetType, DimensionUniverse, CollectionType
from lsst.skymap import BaseSkyMap
from astropy.table import Table

df = Table.read("fakeQuasars.fits").to_pandas()
butler = Butler("/repo/main/", writeable=True)
siCatalogs = [(df, 9813)]

siType = DatasetType("si_cat",
                     dimensions=["skymap", "tract"],
                     storageClass="DataFrame",
                     universe=DimensionUniverse())

butler.registry.registerDatasetType(siType)
butler.registry.registerCollection("u/sr525/ssiCatsQuasars", type=CollectionType.RUN)

for cat, tract in siCatalogs:
    dataId = {"tract": tract, "skymap": "hsc_rings_v1"}
    butler.put(cat, fakesType, dataId, run=run)
```

How? - Step 2: Make a Pipeline

You can find example pipelines in `drp_pipe`, look for the pipelines with `+SI`.

```
description: The DRP pipeline specialized for the HSC RC2 dataset with source injection.  
instrument: lsst.obs.subaru.HyperSuprimeCam  
tasks:  
  inject_coadd:  
    class: lsst.source.injection.inject_coadd.CoaddInjectTask
```

How? - Step 3: Add to the Data

For this example we are going to add to coadds:

```
pipetask run -b /repo/main  
-i HSC/runs/RC2/w_2022_24/DM-35231,u/sr525/ssiCatsQuasars3  
-p $DRP_PIPE_DIR/pipelines/HSC/DRP-RC2+SI.yaml#inject_coadd  
-d "instrument='HSC' and tract=9813 and skymap='hsc_rings_v1' and (band='g' or band='r' or band='i' or band='z' or  
band='y')"  
-o u/sr525/siQuasars3
```

The inputs, here we have one of the regular reprocessing runs followed by where the si catalogue was placed in the previous step.

The pipeline file and step you want to run, in this case it is labelled inject_coadd.

The output name

The datald that tells it what you want to process

How? - Step 3: Add to the Data

Let's check to see if the fakes are in there and where we expect them.



si_deepCoadd_calexp
with the positions
from the input
catalogue
overplotted.

They are!

How? - Step 4: Run the Rest of the Processing

Need to run:

Detection

Merge Detections

Deblend

Measure

Merge Measurements

Write Object Table

Transform Object Table

Forced Phot Coadd

Or more if you added to visits

How? - Step 4: Run the Rest of the Processing


```
description: Process si from coadd onwards
instrument: lsst.obs.subaru.HyperSuprimeCam
imports:
  location: "$DRP_PIPE_DIR/ingredients/HSC/DRP.yaml1"
tasks:
  detection:
    class: lsst.pipe.tasks.multiBand.DetectCoaddSourcesTask
    config:
      connections.inputCoaddName: "si_deep"
      connections.outputCoaddName: "si_deep"
  mergeDetections:
    class:
lsst.pipe.tasks.mergeDetections.MergeDetectionsTask
    config:
      connections.inputCoaddName: "si_deep"
      connections.outputCoaddName: "si_deep"
  deblend:
    class:
lsst.pipe.tasks.deblendCoaddSourcesPipeline.DeblendCoaddSou
rcesMultiTask
    config:
      connections.inputCoaddName: "si_deep"
      connections.outputCoaddName: "si_deep"
```

```
measure:
  class:
lsst.pipe.tasks.multiBand.MeasureMergedCoaddSourcesTask
  config:
    connections.inputCoaddName: "si_deep"
    connections.outputCoaddName: "si_deep"
  mergeMeasurements:
    class:
lsst.pipe.tasks.mergeMeasurements.MergeMeasurementsTask
  config:
    connections.inputCoaddName: "si_deep"
    connections.outputCoaddName: "si_deep"
  writeObjectTable:
    class: lsst.pipe.tasks.postprocess.WriteObjectTableTask
    config:
      connections.coaddName: "si_deep"
  transformObjectTable:
    class:
lsst.pipe.tasks.postprocess.TransformObjectCatalogTask
  config:
    connections.coaddName: "si_deep"
  forcedPhotCoadd:
    class: lsst.meas.base.forcedPhotCoadd.ForcedPhotCoaddTask
    config:
      connections.inputCoaddName: "si_deep"
      connections.outputCoaddName: "si_deep"
```

How? - Step 4: Run the Rest of the Processing

```
pipetask run -b /repo/main  
-i u/sr525/siQuasars3  
-p coaddPipeline.yaml#detection  
-d "instrument='HSC' and tract=9813 and skymap='hsc_rings_v1'"  
-o u/sr525/siQuasars3_detection
```

Running one step at a time, here
detection.



Then repeat for the rest of the steps

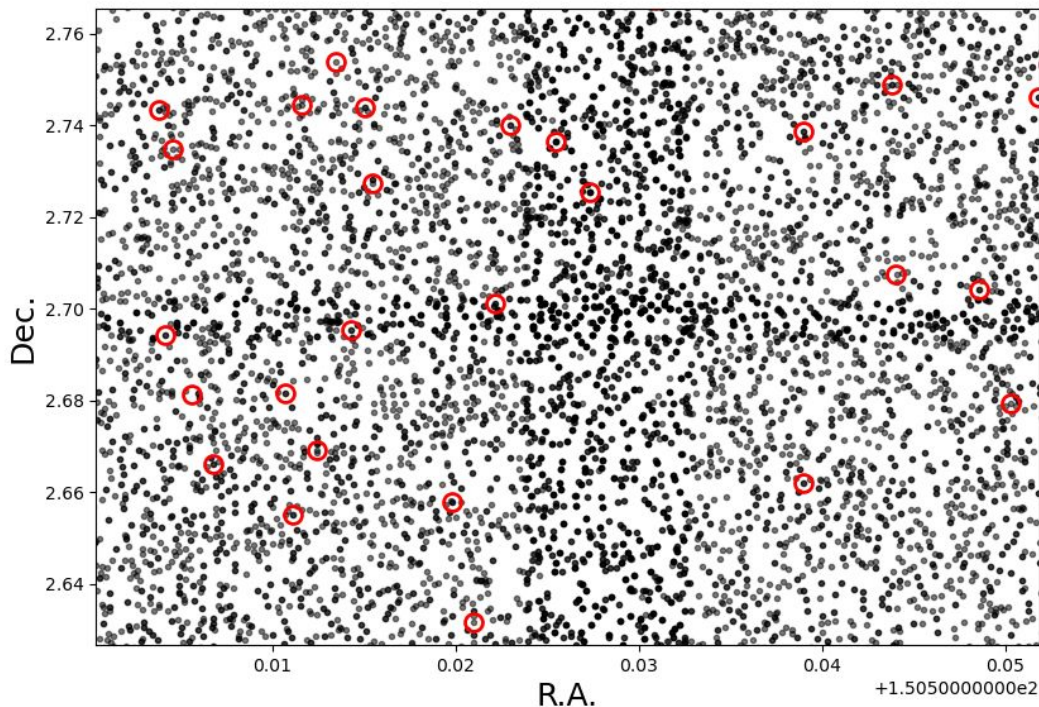
How? - Step 5: SCIENCE!!

Did the fakes make it all the way to the end?

Yes! They are even where we expected them to be.

Added in 9800 - got 8393 out

- Not the entire tract made it through processing
- Should have started before two days ago



How? - Step 5: SCIENCE!!

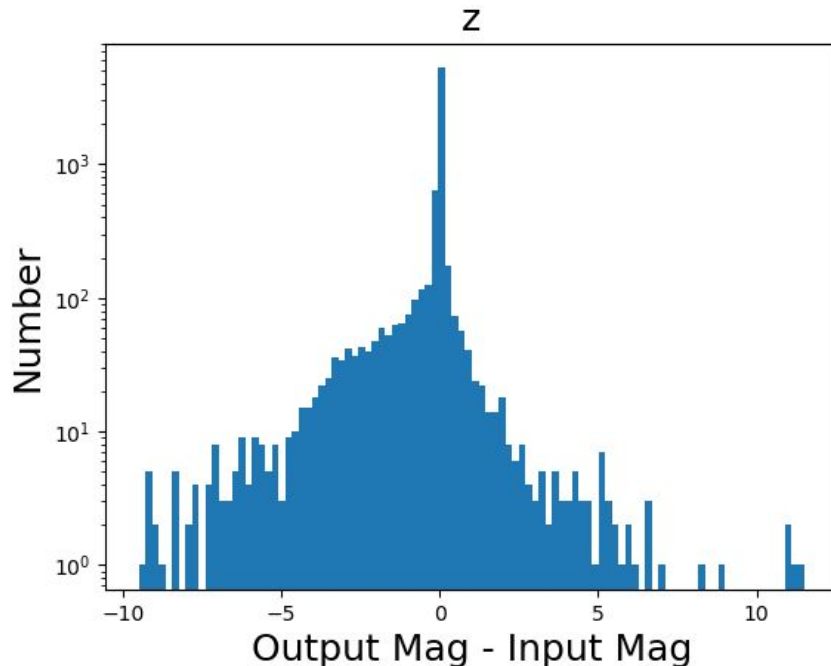
z band magnitude comparison

There are z drops in this data

Peak is at 0 though so that is good.

Rough quasar selection is returning about 40% of the potential objects.

Next science step is to find out why and to check out different selection methods.
Can't do this without synthetic data.



What Will be Available?

Current:

Continuously process at least one tract with a standard catalogue

- Will be the same input each run

Can ingest your own bespoke catalogues - as seen in demo

Planned Additions:

Special but pre ingested catalogues

- Eg. Extended source, point source

Fits images

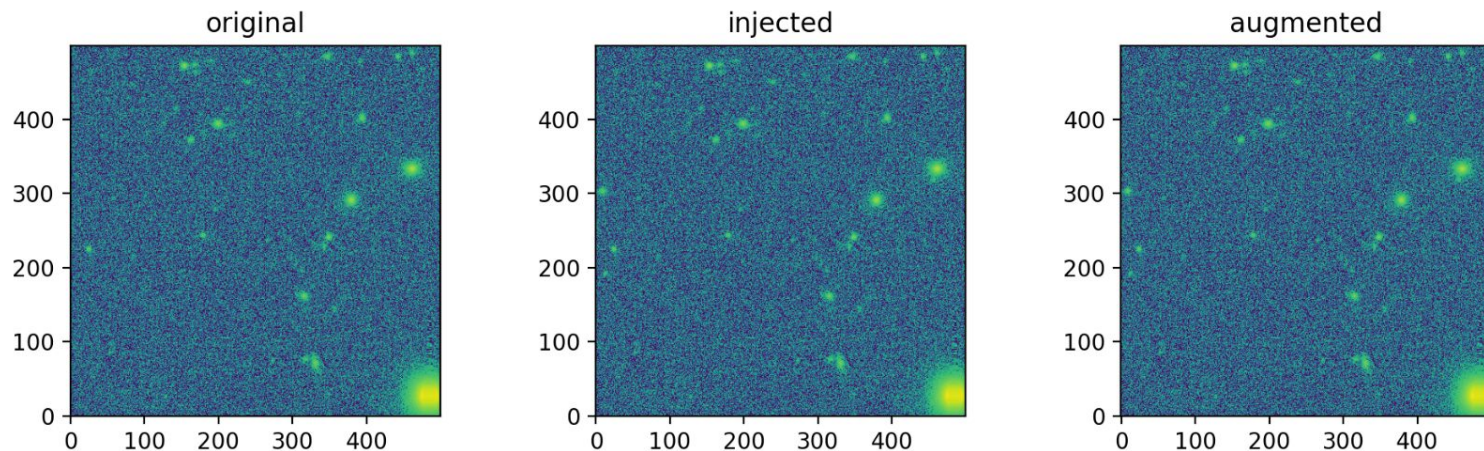
Testing SSI

Josh Meyers (LLNL)

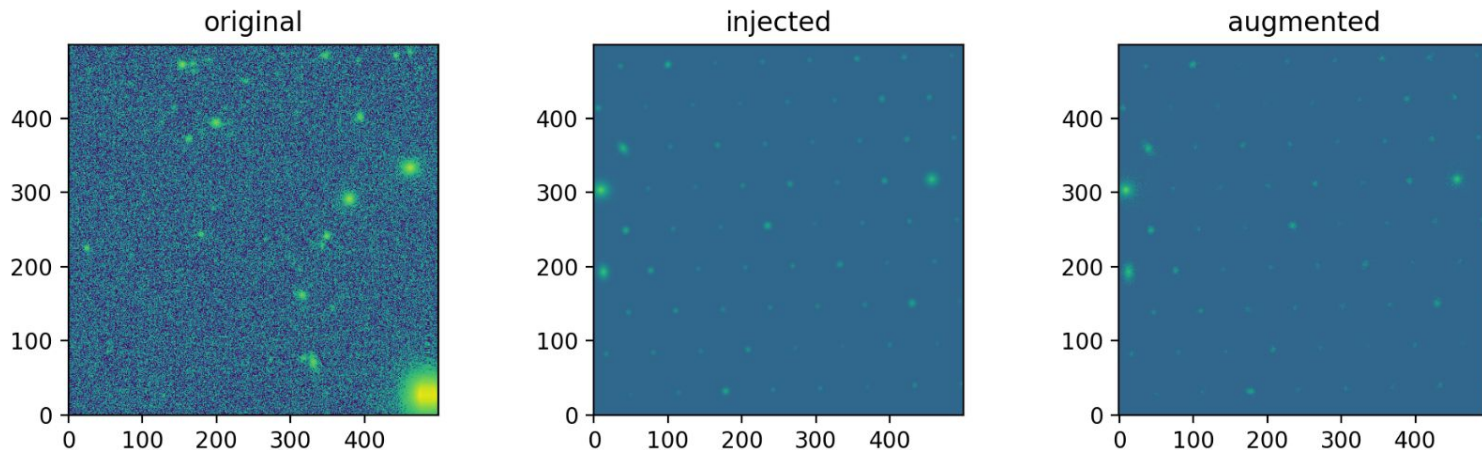
Testing SSI - injection vs resimulation for DP0

- SSI requires using *inferred* PSF, WCS, calibration instead of *true*.
- But! With DP0 we have access to the truth.
- Compare two methods of inserting new sources:
 - SSI
 - Resimulation with augmented catalog
- Done the above for 1 DC2 CCD image.
 - Input catalog from cosmo_DC2 clipped to $i < 28$, placed on hexagonal grid covering 1 tract.
- Caveat: done with “old” source injection code.

Injection / Augmentation demo

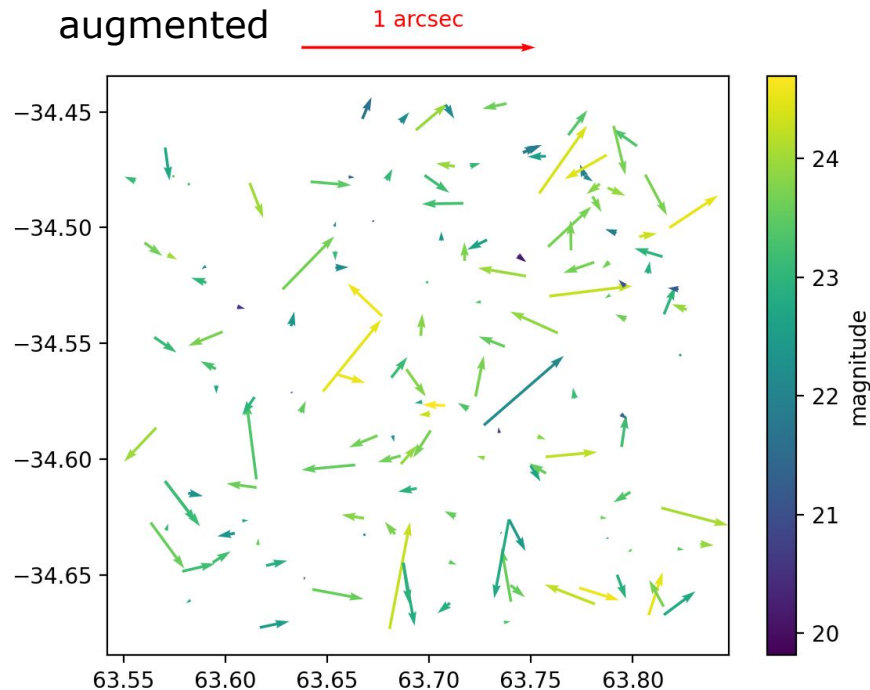


Injection / Augmentation demo



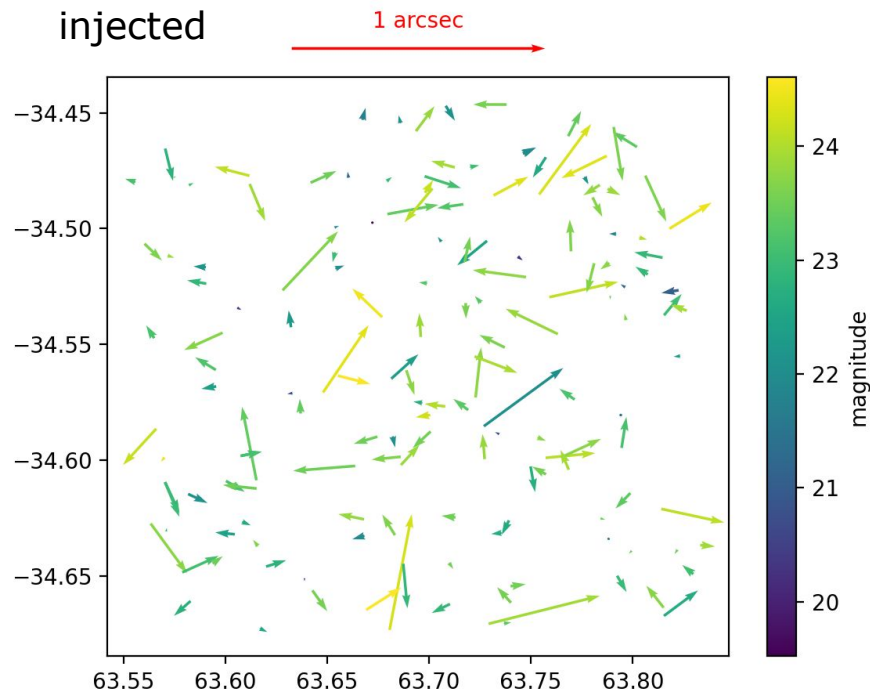
Astrometry Comparison

- $< \text{arcsecond}$ galactic astrometry
- Very similar augmented vs injected



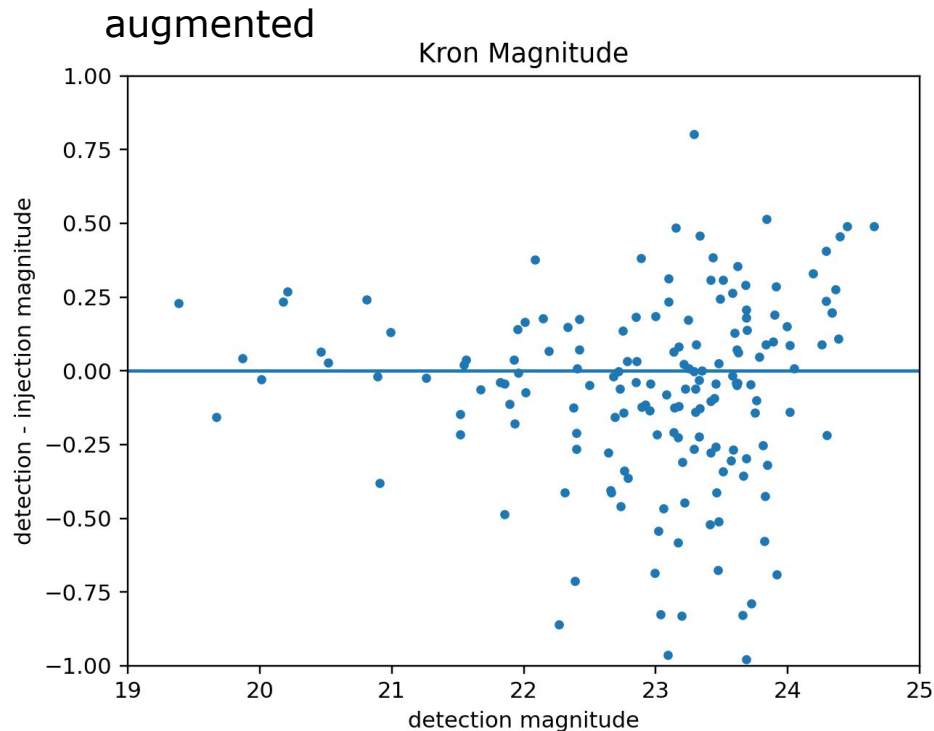
Astrometry Comparison

- $< \text{arcsecond}$ galactic astrometry
- Very similar augmented vs injected



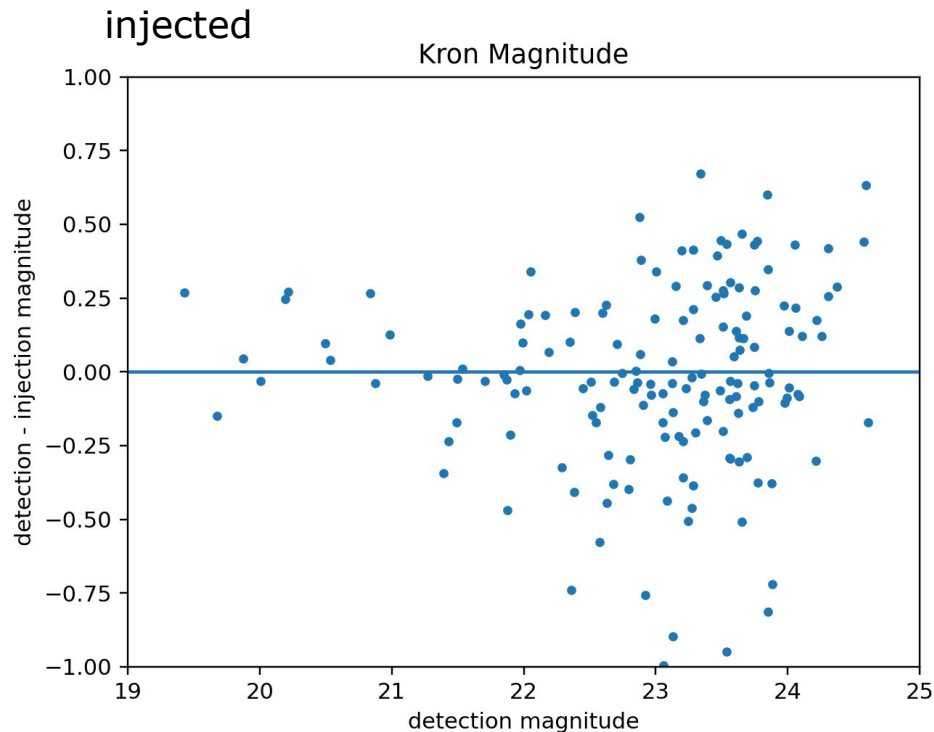
Photometry Comparison

- Maybe hope for better results at bright end?
- But low-number stats.
- Very similar augmented vs injected



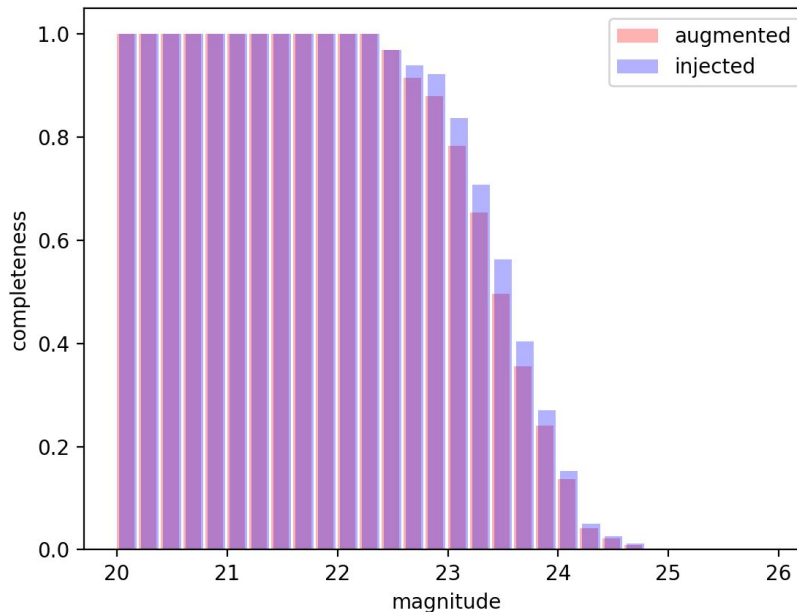
Photometry Comparison

- Maybe hope for better results at bright end?
- But low-number stats.
- Very similar augmented vs injected



Completeness

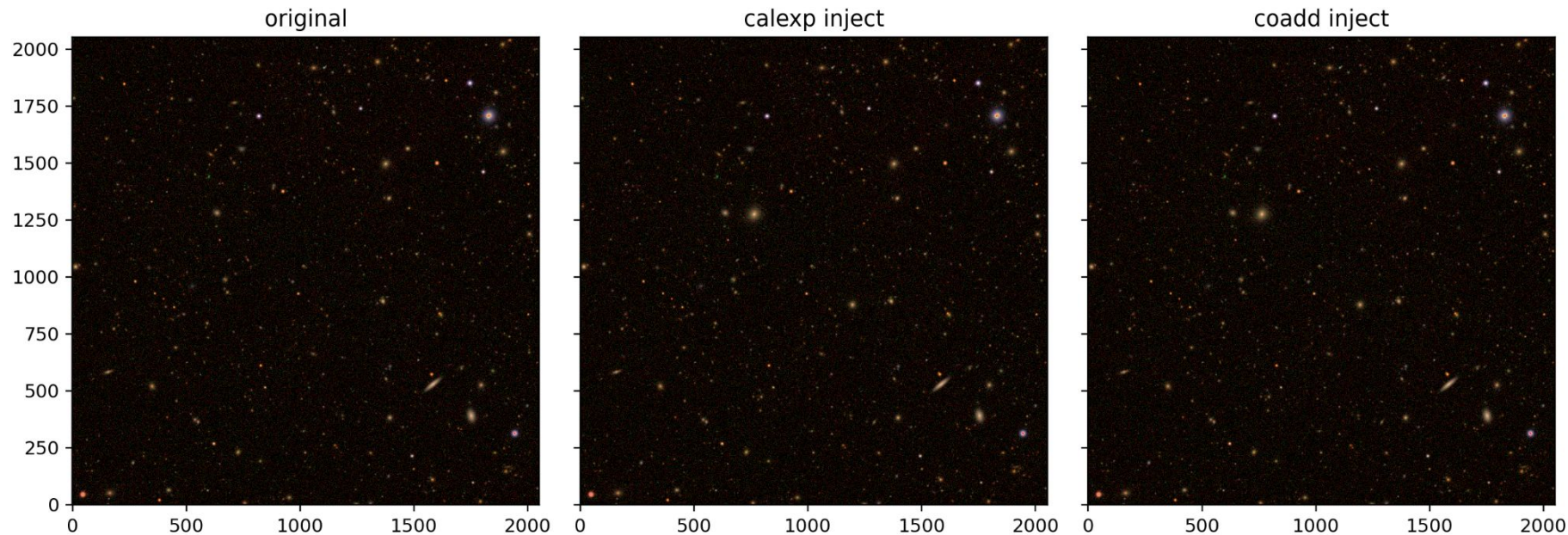
- Some small differences.
- Differences are due to:
 - Detection and Measurement settings not being identical. This *should* be much easier to synchronize in the current modular injection framework.
 - Injections are smooth, Poisson-free images.
 - Variance plane *not* currently modified during injection.
 - Using *inferred* PSF, WCS, photometric calibration, etc.



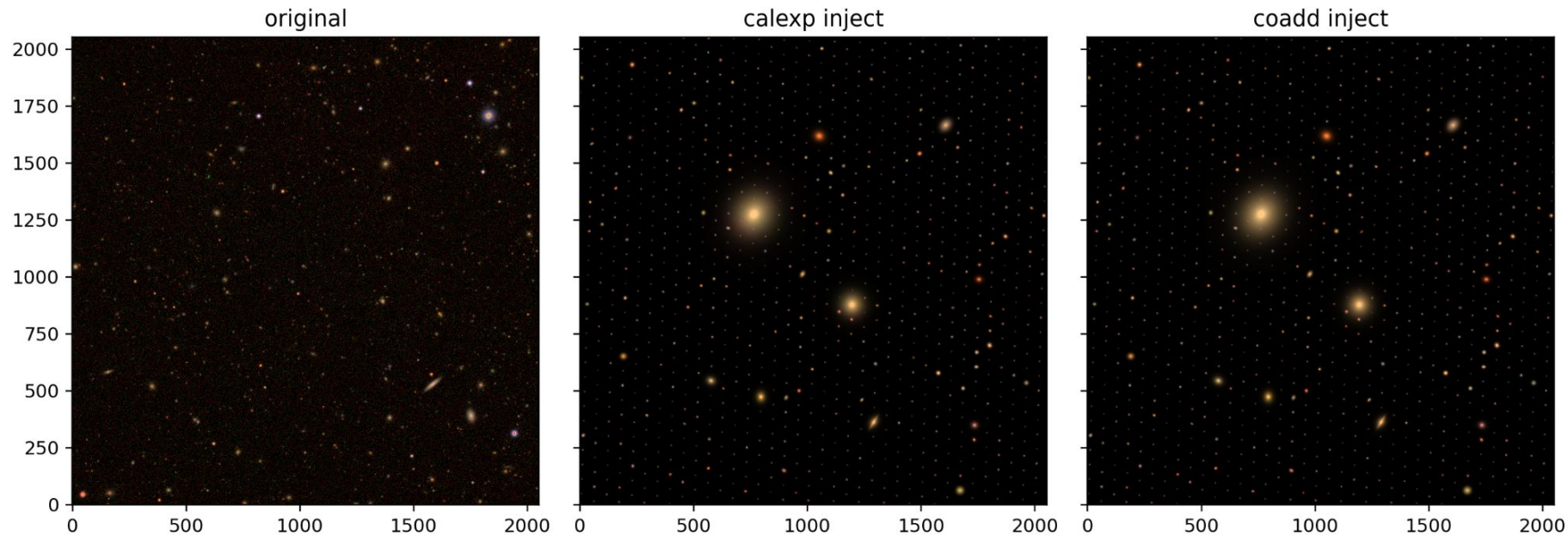
Testing SSI : do coaddition and injection commute?

- Workflow for injecting into single exposures includes rerunning single frame processing and coaddition; detection/measurement on coadds
- Workflow for injecting into coadds only includes detection/measurement on coadds; so can be much faster, especially when coadding many exposures.
- So can we just do the fast thing?
 - Comparing *injection directly into coadds* with *injection into individual exposures followed by coaddition*.
- Used more modern SSI pipeline tools to test above on one patch of DP0 in ugrizy.
https://github.com/LSSTDESC/ssi_vs_resim/tree/main/reprocess (bps)

Injection example

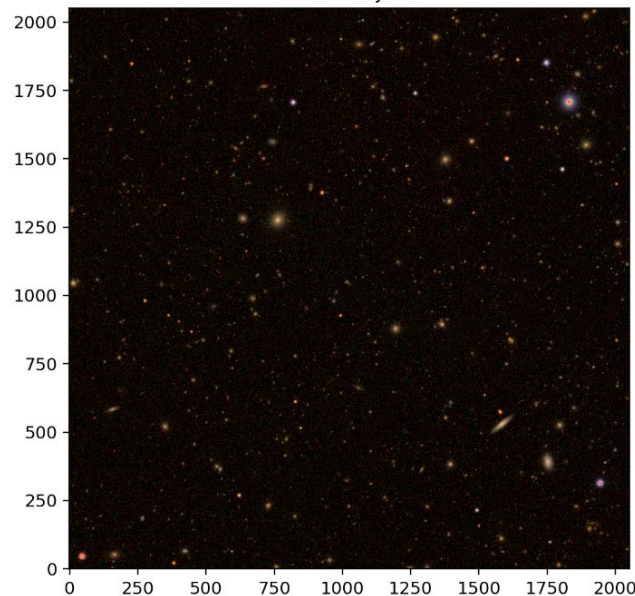


Injection example

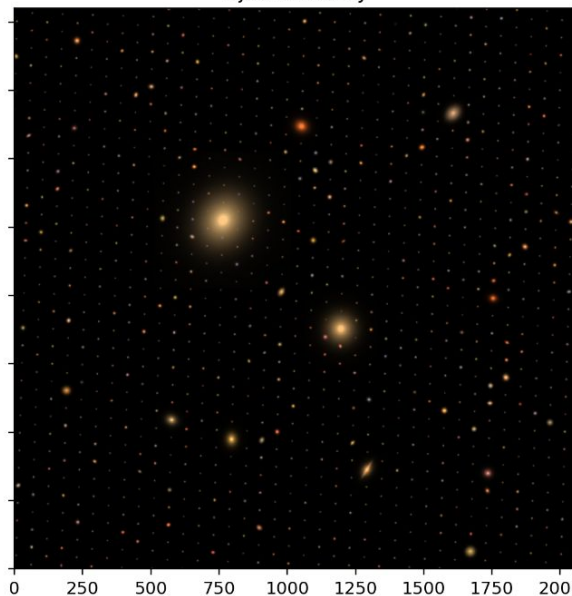


Does coaddition commute with injection?

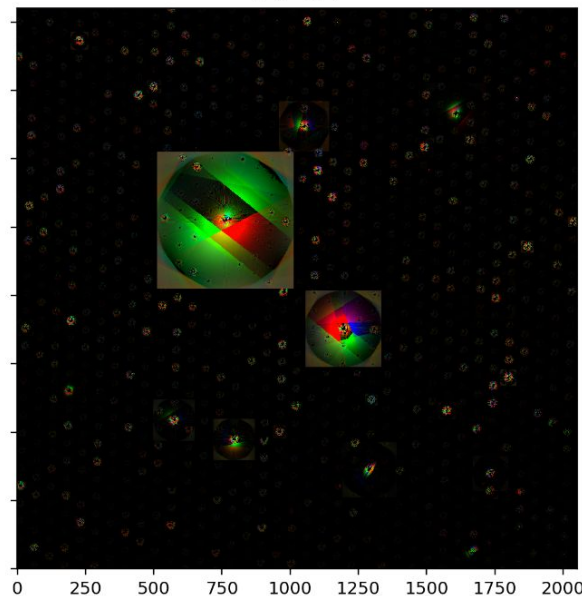
Coadd inject



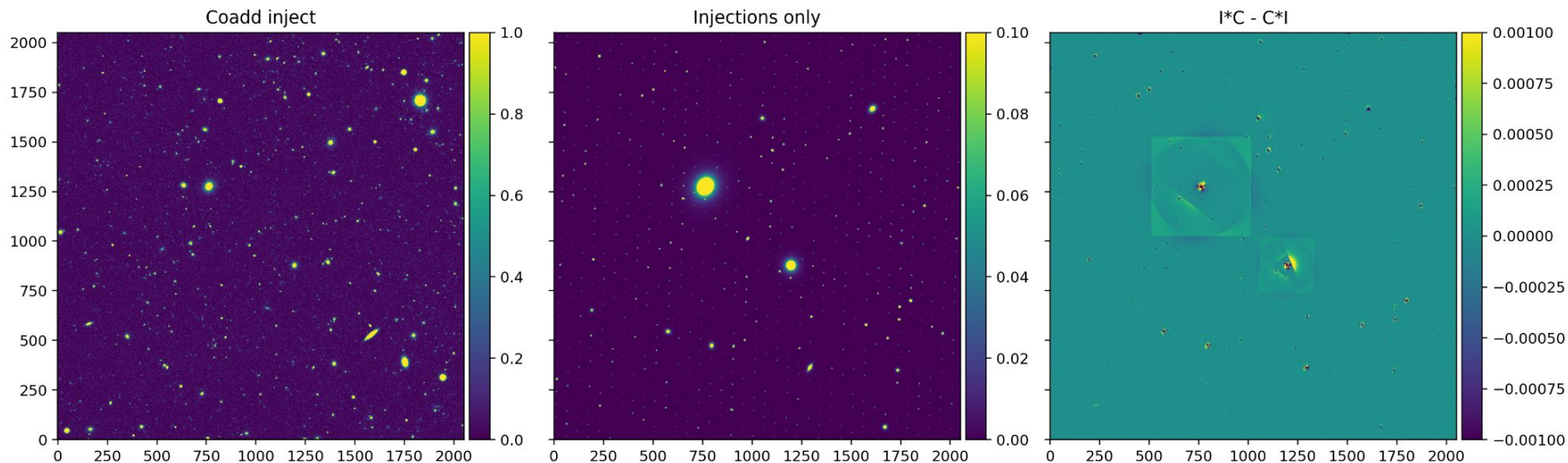
Injections only



$I * C - C * I$



Does coaddition commute with injection?

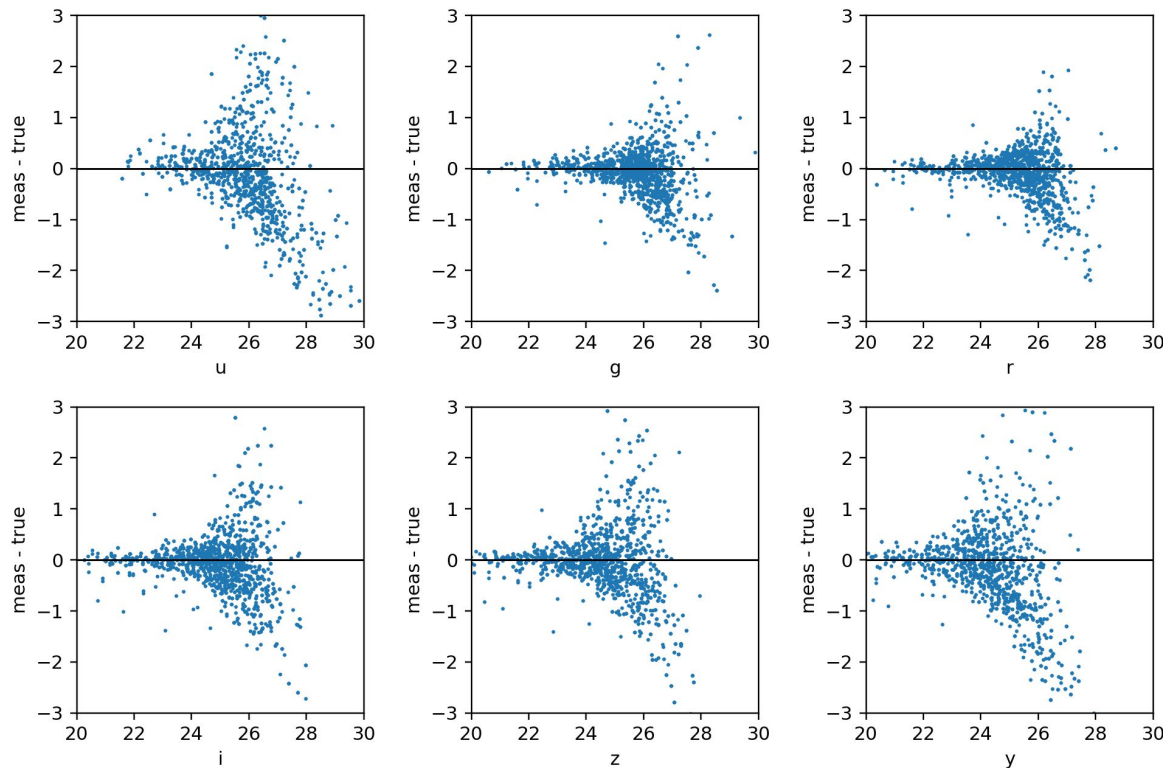


Why is the commutator $\neq 0$?

- Single exposure injections modify boxes aligned with original exposures. These may have many relative rotations in coadd coordinates.
- We also assume PSF is constant over the box we're injecting into. Edges of single visits spoil this assumption.
- Small alignment errors may exist when forming the coadd image, these may not all be captured in the coadd PSF.

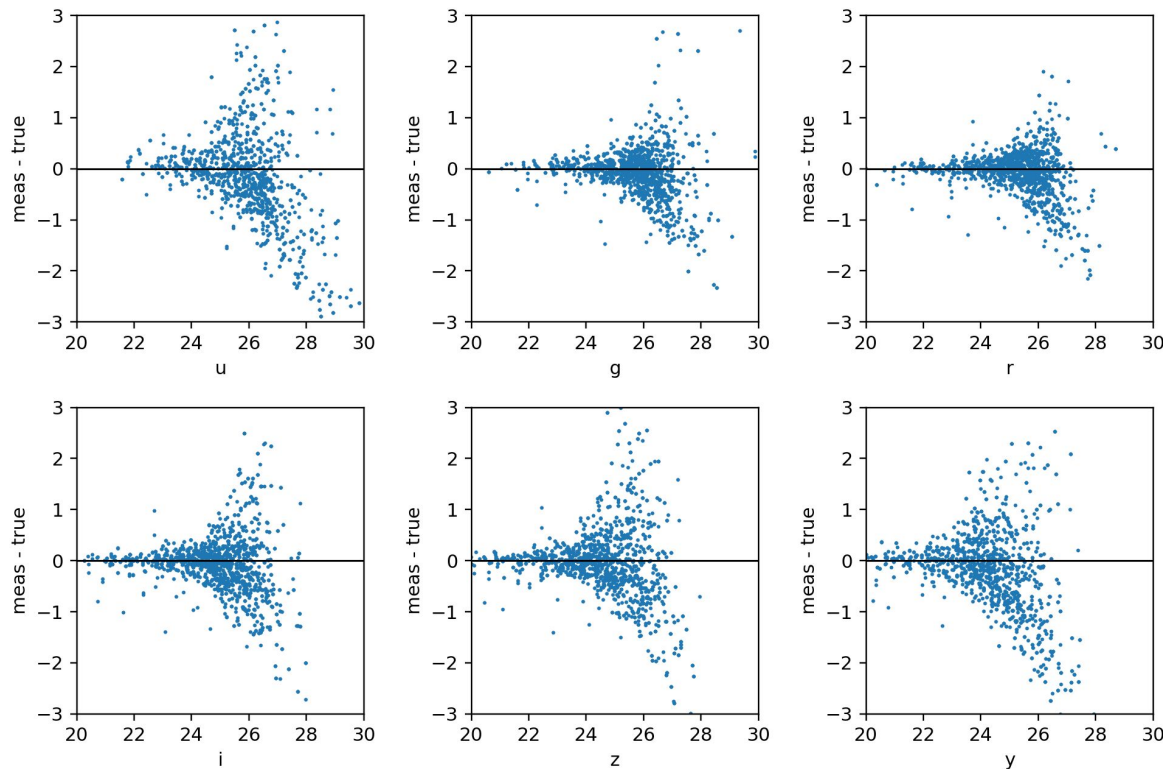
Photometry Comparison

- coadd-injection
- Moderate residuals to truth



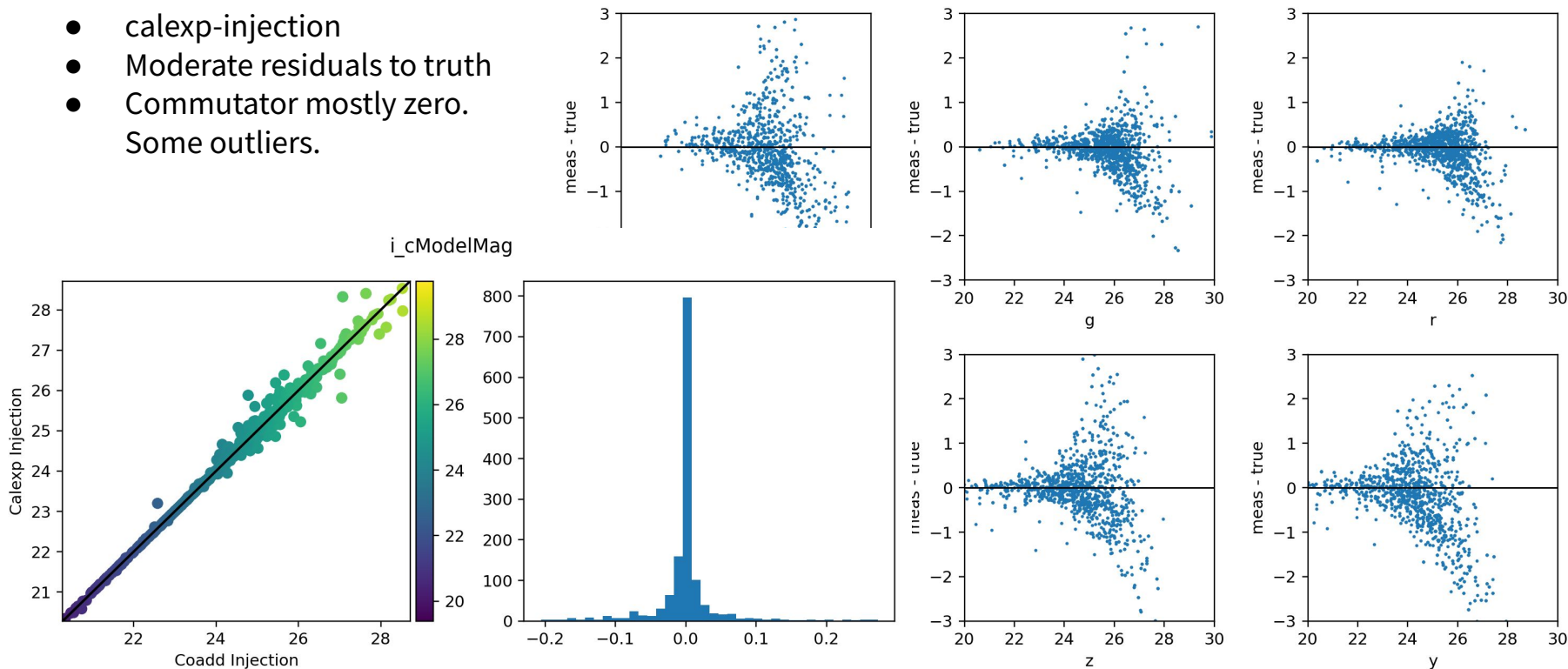
Photometry Comparison

- calexp-injection
- Moderate residuals to truth



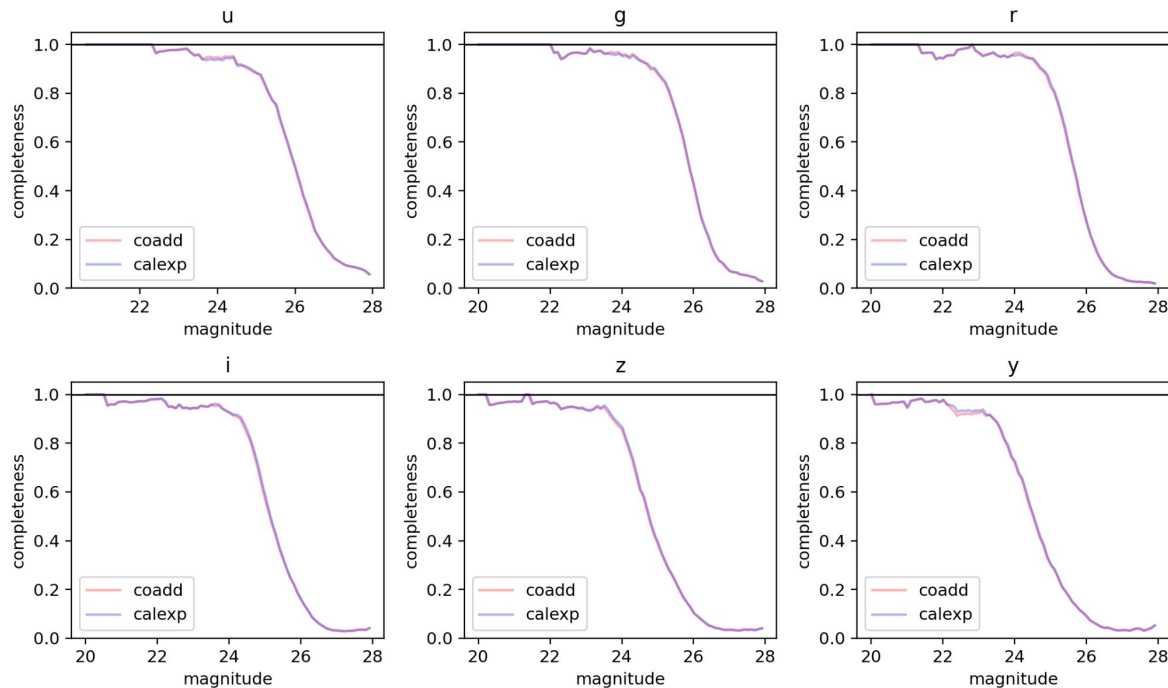
Photometry Comparison

- calexp-injection
- Moderate residuals to truth
- Commutator mostly zero.
Some outliers.



Completeness

- Need to zoom in pretty far to see differences between detections in coadd-injections and detections in coadds of calexp-injections.



SSI testing conclusions

- Would be good to
 - Add Poisson noise to injections
 - Modify the variance plane consistently.
- Need to be careful that same task configurations are used on injected images as original images. This is getting easier with modular source injection framework.
- Injections into coadds are subtly different than coadds of injections into single exposures.

What else would be useful?

What do we need to include to allow you to do your science easily?

