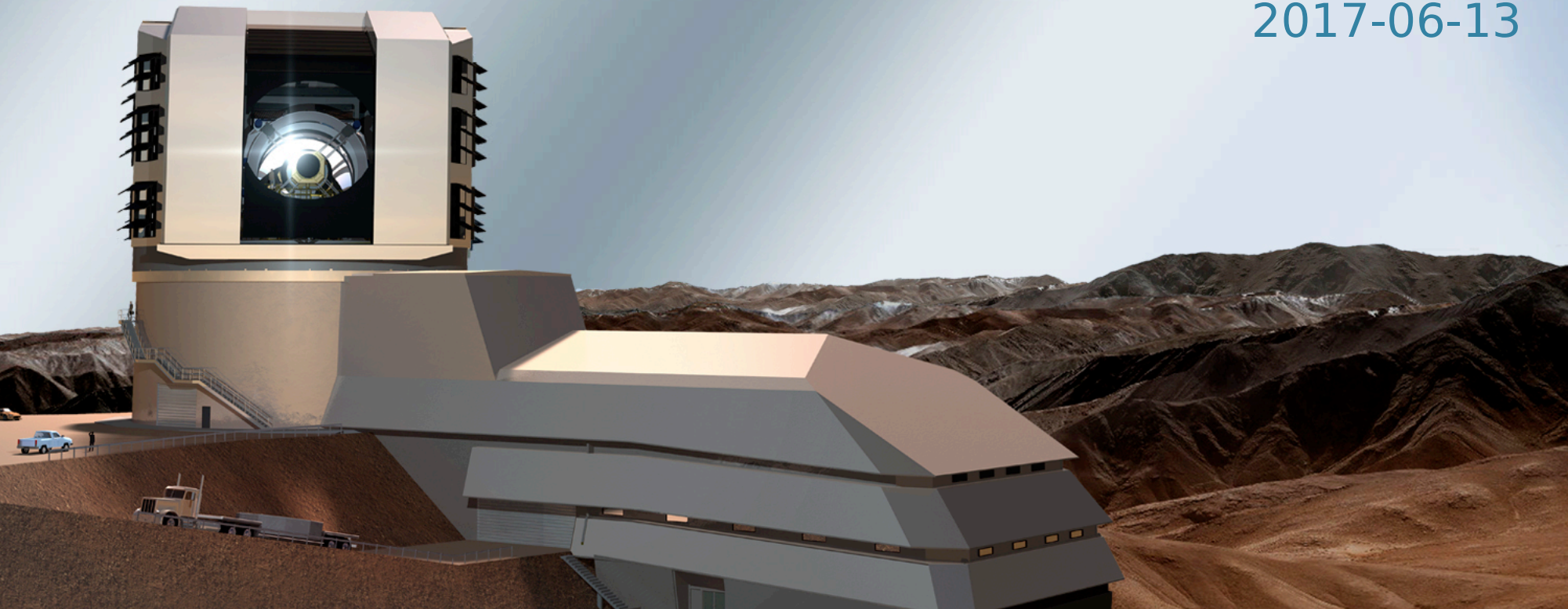
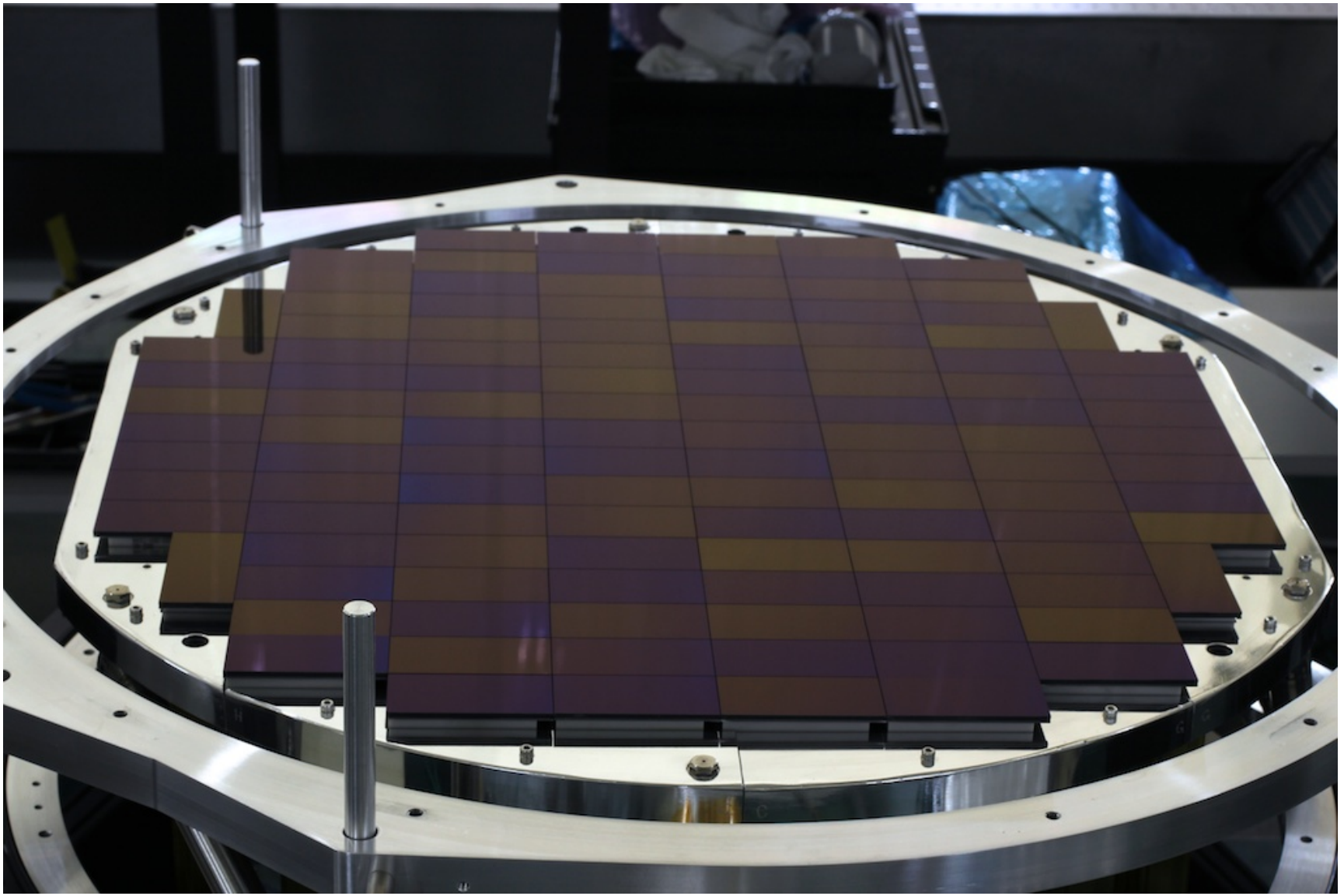
Robert Lupton
LSST Pipeline Scientist

HSC pixels meet LSST Pipelines

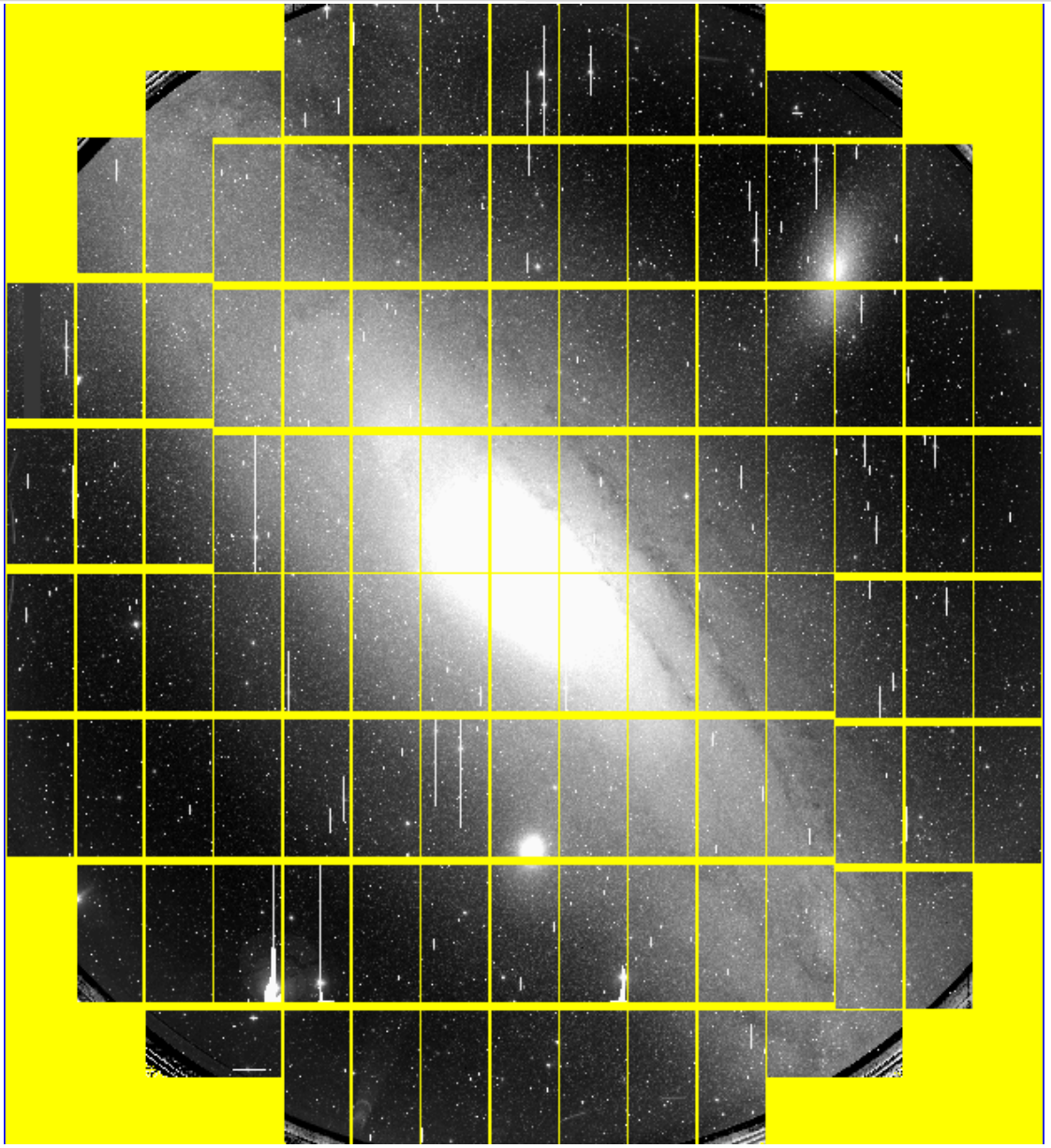
2017-06-13



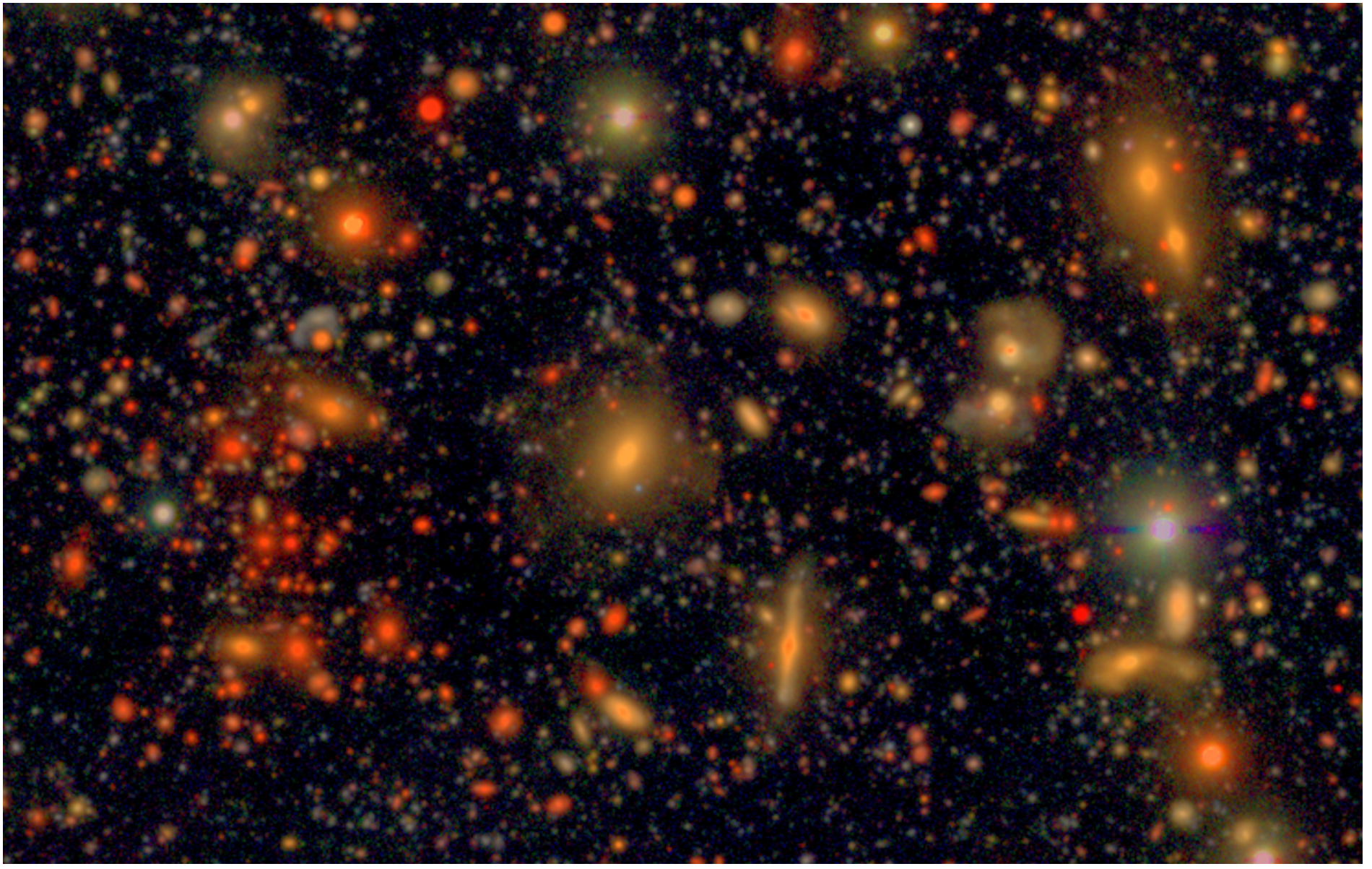
HSC’s Pixels



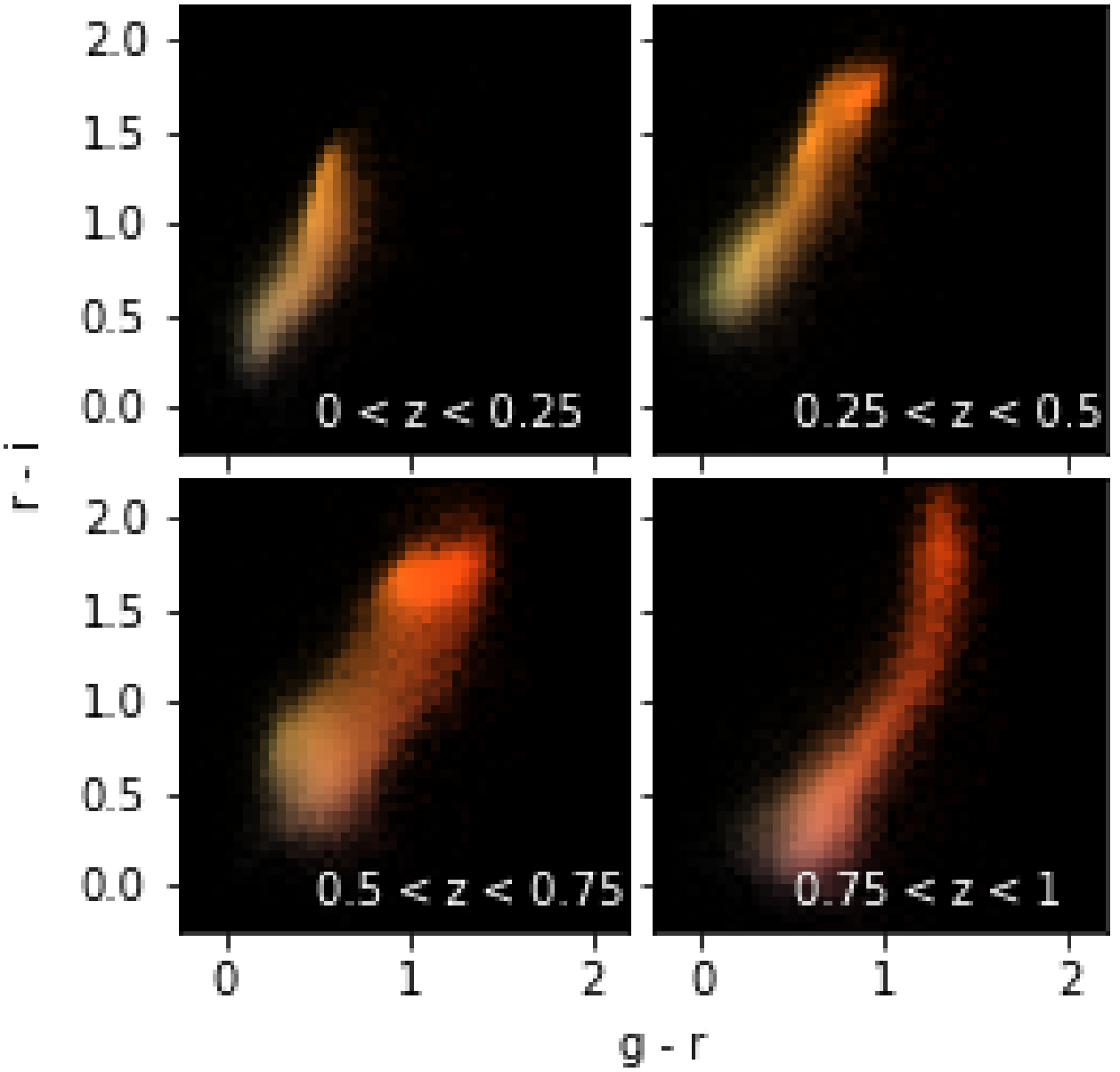
HSC's 1.8 deg^2 Field of View



Cosmos



Cosmos



Strategy

- Use the LSST prototype codes

Strategy

- Use the LSST prototype codes
- Develop them fast enough to handle HSC data

Strategy

- Use the LSST prototype codes
- Develop them fast enough to handle HSC data

Risky, but I *think* we got away with it.

Strategy

- Use the LSST prototype codes
- Develop them fast enough to handle HSC data

Risky, but I *think* we got away with it. I hope Tanaka-san agrees.

Strategy

- Use the LSST prototype codes
- Develop them fast enough to handle HSC data

Risky, but I *think* we got away with it. I hope Tanaka-san agrees.

I’m pretty confident that we’ll have come out ahead by the time the HSC ‘SSP’ survey is done.

Create the tasks

In [6]:
```
schema = afwTable.SourceTable.makeMinimalSchema()
algMetadata = dafBase.PropertyList()

config = CharacterizeImageTask.ConfigClass()
config.psfIterations = 1
if False:
    config.measurePsf.psfDeterminer.name = "pca"
else:
    config.measurePsf.psfDeterminer.name = "psfex"

charImageTask =         CharacterizeImageTask(None, config=config)

config = SourceDetectionTask.ConfigClass()
sourceDetectionTask =   SourceDetectionTask(schema=schema, config=config)

sourceDeblendTask =     SourceDeblendTask(schema=schema)

config = SingleFrameMeasurementTask.ConfigClass()
config.slots.apFlux = 'base_CircularApertureFlux_12_0'
sourceMeasurementTask = SingleFrameMeasurementTask(schema=schema, config=config,
                                                   algMetadata=algMetadata)
```

LSST Pipelines

Process the data

In [7]:
```
butler = dafPersist.Butler("/Volumes/RHLData/hsc-v13_0")
```

In [8]:
```
dataId = dict(visit=29352, ccd=50)
exposure = butler.get('calexp', dataId)

tab = afwTable.SourceTable.make(schema)

result = charImageTask.characterize(exposure)
psfCellSet = result.psfCellSet     # we'll look at this data structure later

result = sourceDetectionTask.run(tab, exposure)
sources = result.sources

sourceDeblendTask.run(exposure, sources)

sourceMeasurementTask.run(exposure, sources)

sources.writeFits("outputTable.fits")
exposure.writeFits("example1-out.fits")
```

Outline of Data Release Pipeline

- Single `visit` processing:
  - flat fielding *etc.*
  - source detection
  - astrometric/photometric calibration
  - PSF estimation
  - deblending
  - measurement

Outline of Data Release Pipeline

- Single `visit` processing:
  - flat fielding *etc.*
  - source detection
  - astrometric/photometric calibration
  - PSF estimation
  - deblending
  - measurement
- Relative calibration (photometric and astrometric) of all visits on the scale of the focal plane.

Outline of Data Release Pipeline

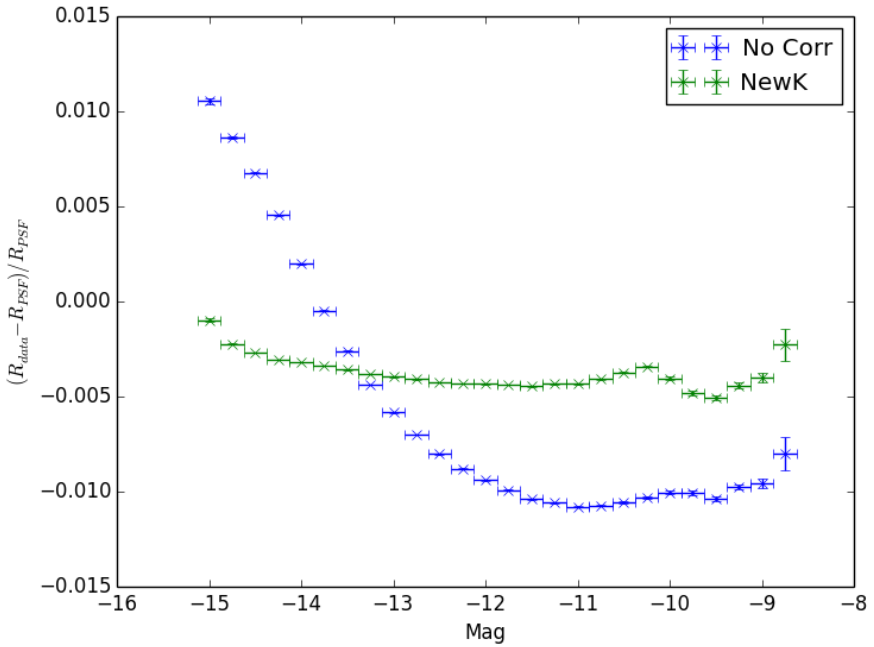
- Single `visit` processing:
  - flat fielding *etc.*
  - source detection
  - astrometric/photometric calibration
  - PSF estimation
  - deblending
  - measurement
- Relative calibration (photometric and astrometric) of all visits on the scale of the focal plane. We're currently using PanSTARRS catalogues, but will soon switch to Gaia (at least for astrometry, maybe photometry)

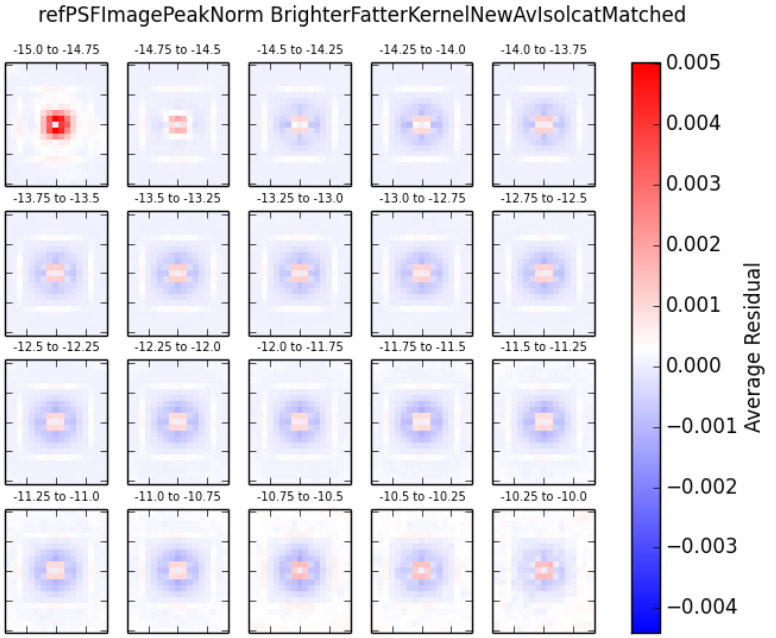
Brighter-fatter

One of the parts of "flat fielding *etc.*" was Brighter-Fatter.

Brighter-fatter

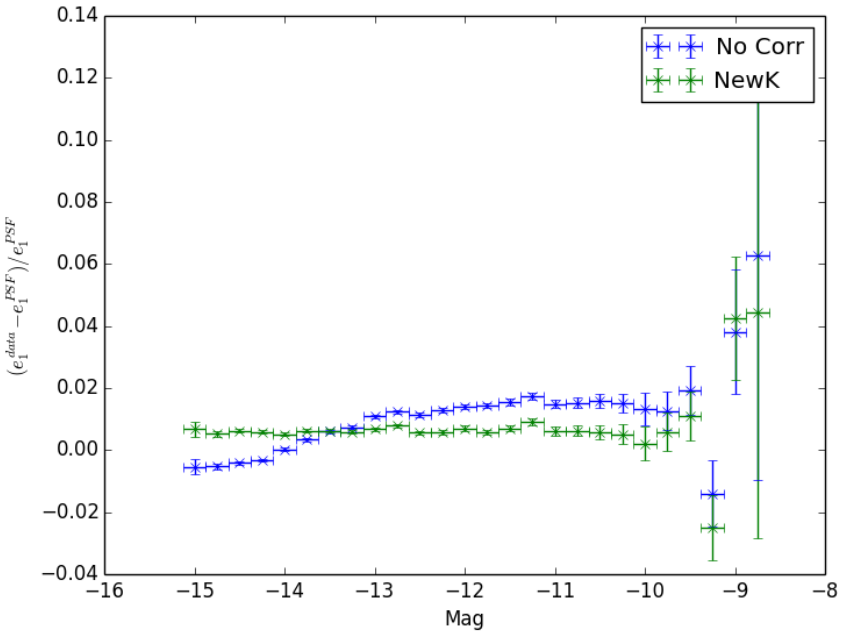
One of the parts of "flat fielding *etc.*" was Brighter-Fatter.



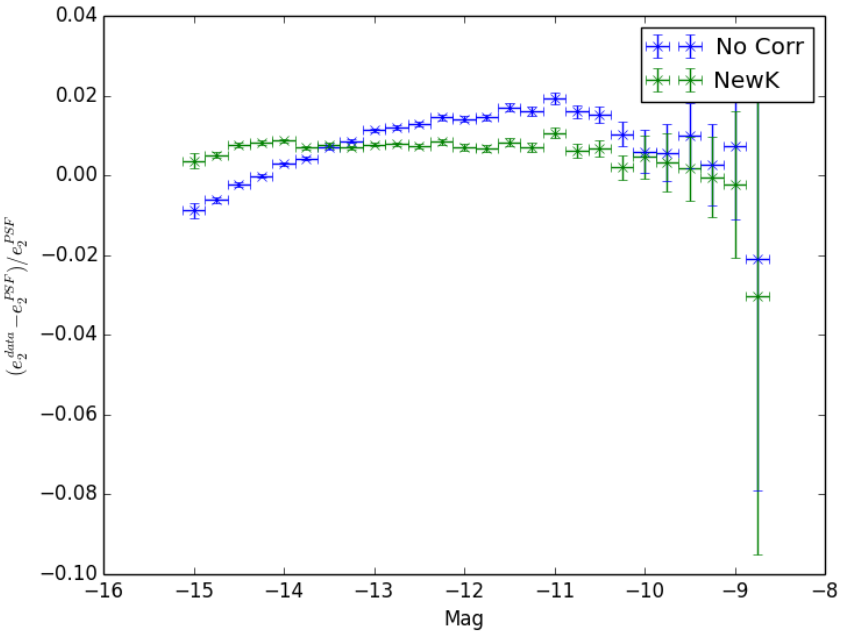


Brighter-fatter

One of the parts of "flat fielding *etc.*" was Brighter-Fatter.



(a) The difference ϵ_1 of ellipticity of stars and the average star both with and without the correction.



(b) The difference ϵ_2 of ellipticity of stars and the average star both with and without the correction.

PSF Estimation

We currently use `psfEx` restructured as a plugin, but are having trouble once the seeing is better than *c.* 0.45" (2.7 pixels).

PSF Estimation

We currently use `psfEx` restructured as a plugin, but are having trouble once the seeing is better than *c.* 0.45" (2.7 pixels). We'll replace it with something better, but what?

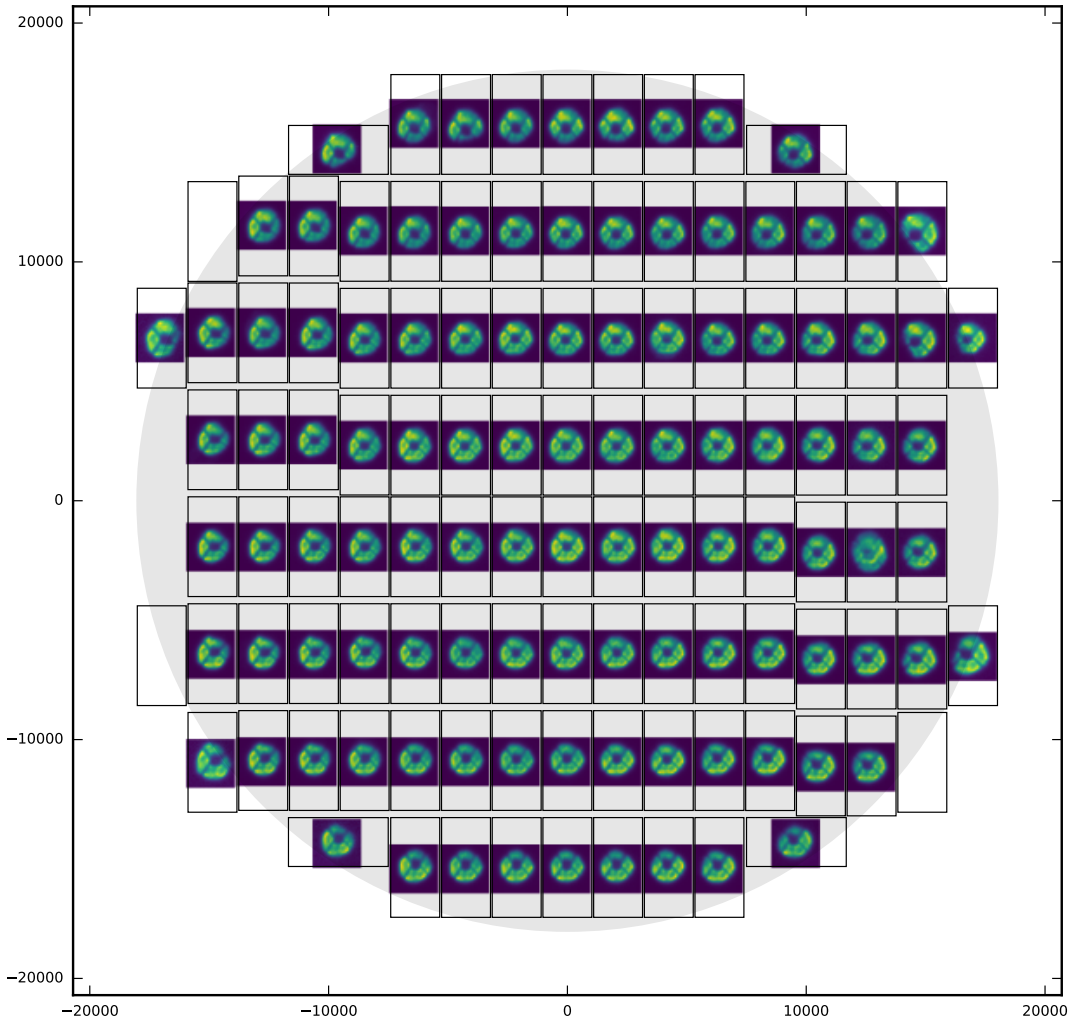
- `psfEx++`?
- `piff`?
- CosmoStat@Saclay's RCA?
- Wavefront-based estimation?

PSF Estimation

We currently use `psfEx` restructured as a plugin, but are having trouble once the seeing is better than *c.* 0.45" (2.7 pixels). We'll replace it with something better, but what?

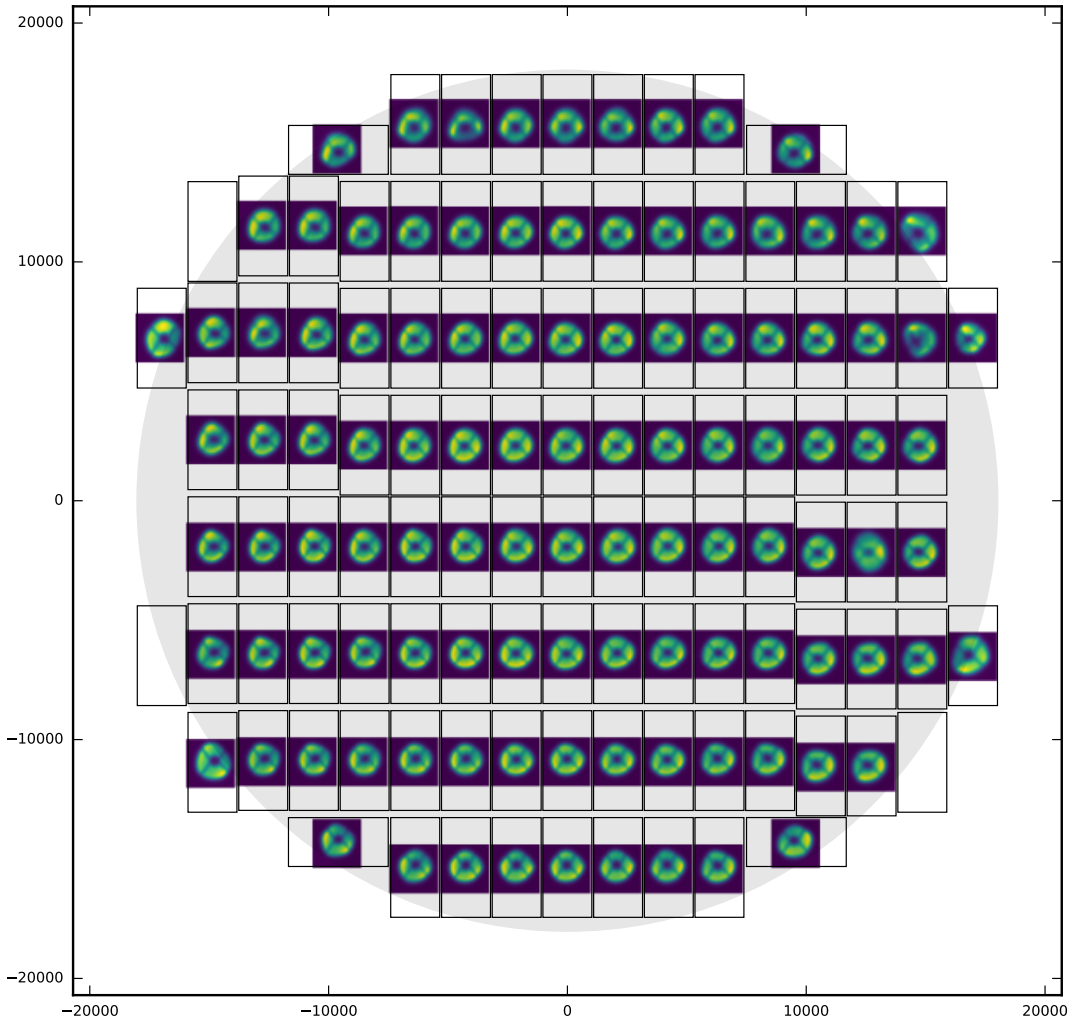
- `psfEx++`?
- `piff`?
- CosmoStat@Saclay's RCA?
- Wavefront-based estimation?
- Your Code?

HSC Wavefronts



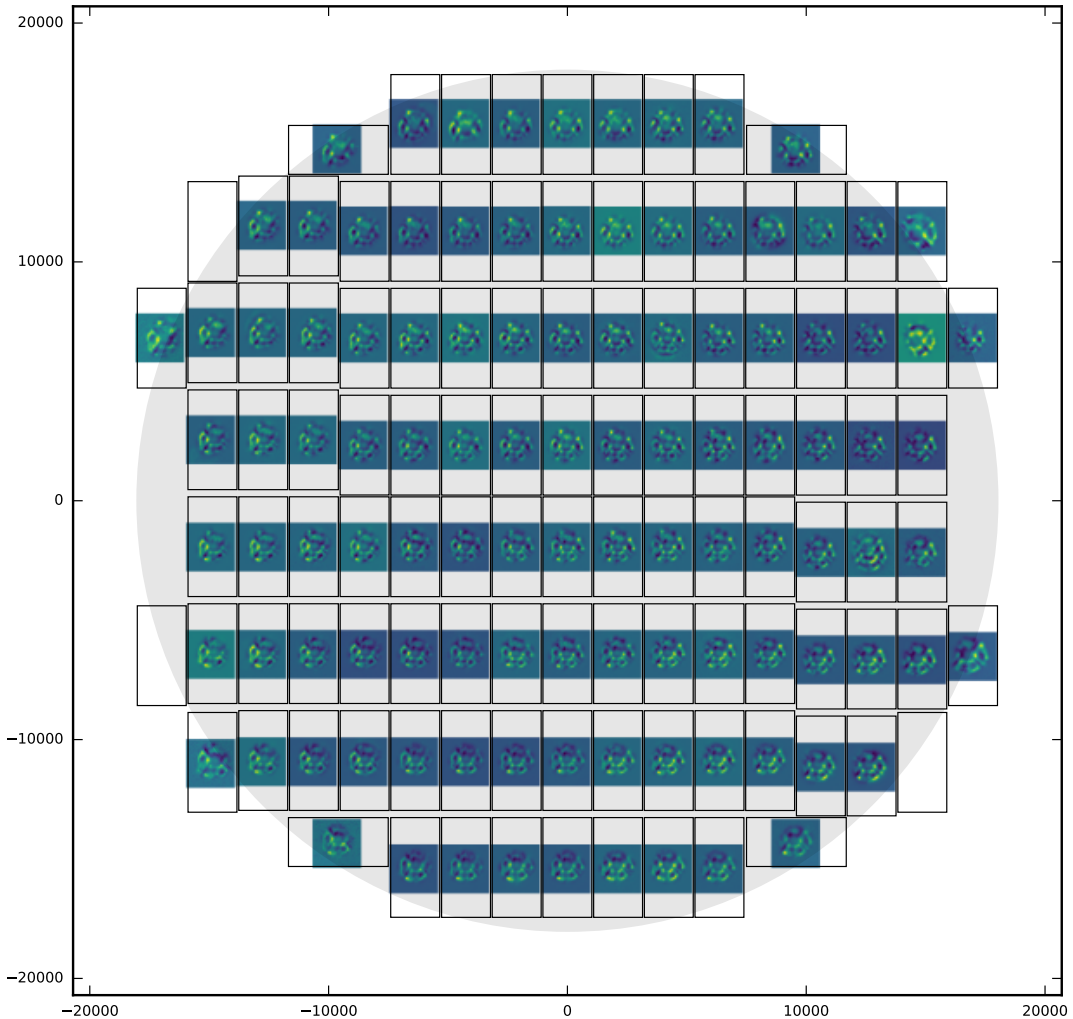
Data

HSC Wavefronts



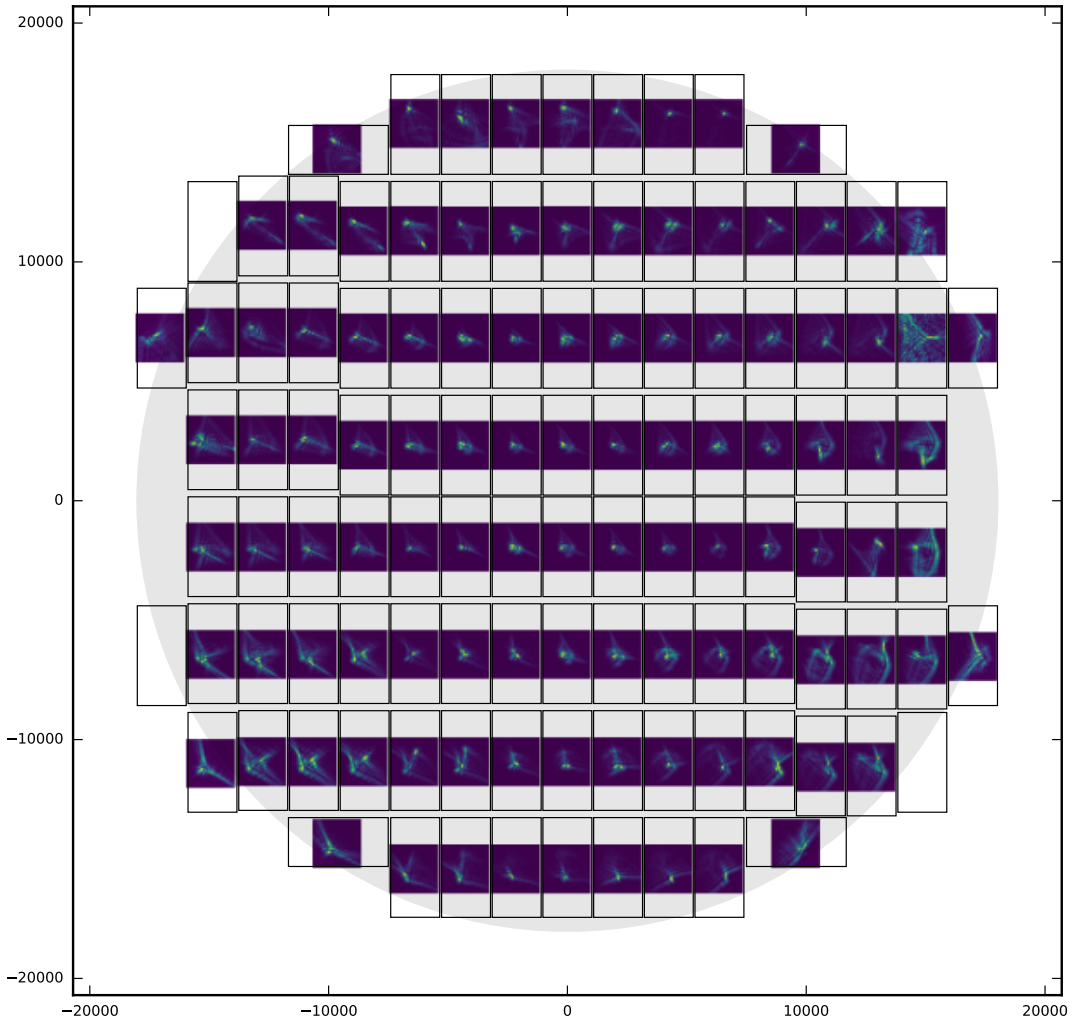
Model

HSC Wavefronts

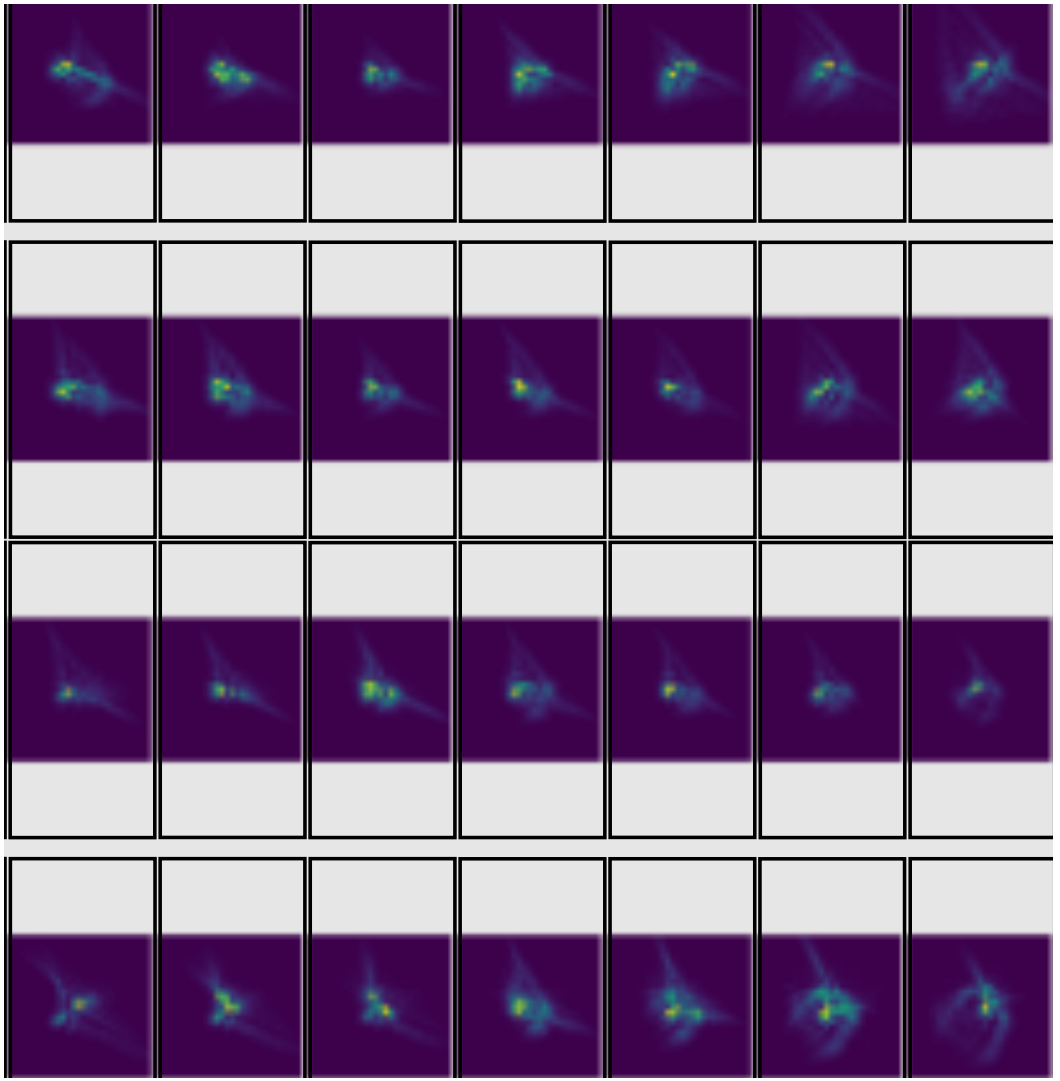


Residual

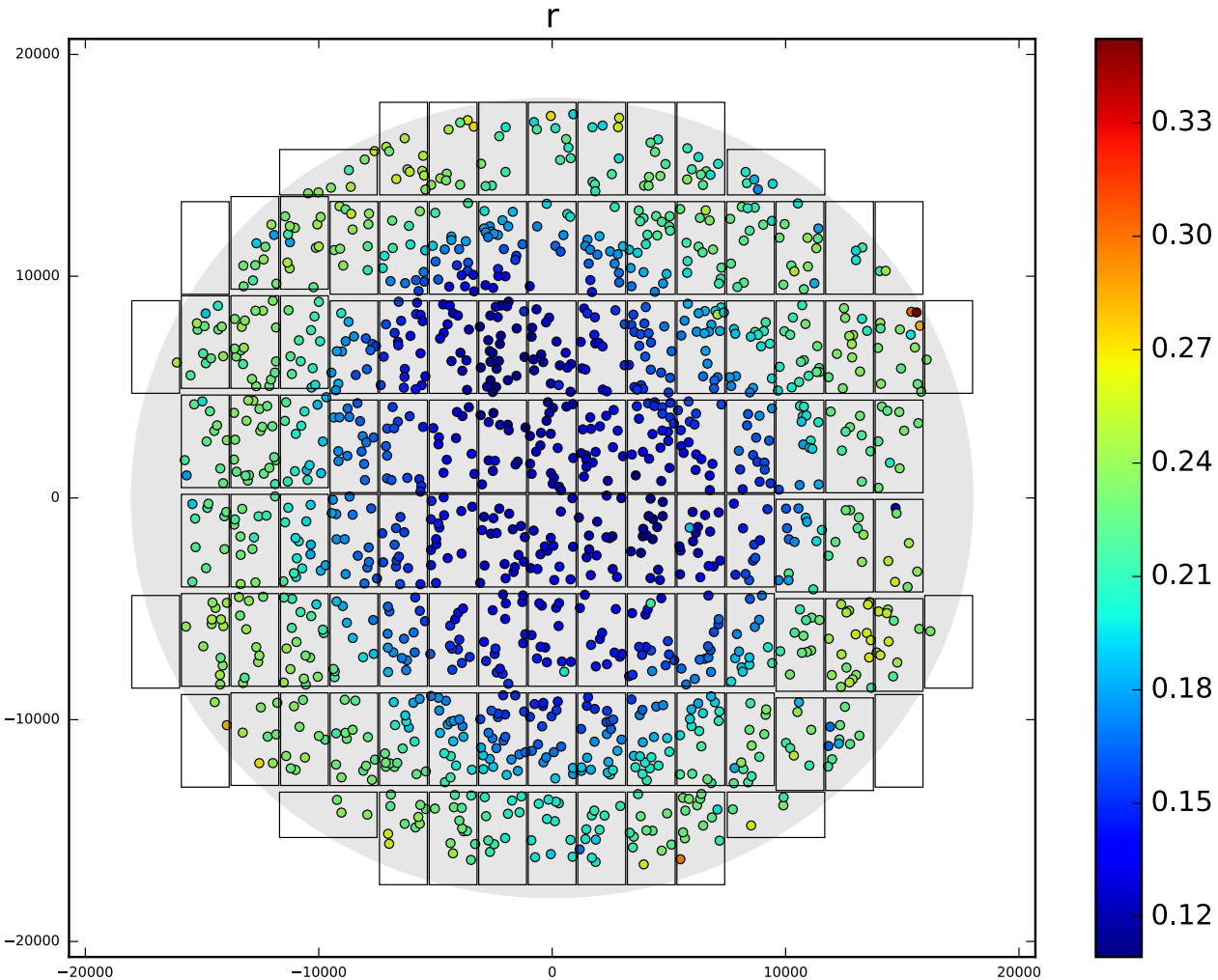
HSC Wavefronts



HSC Wavefronts

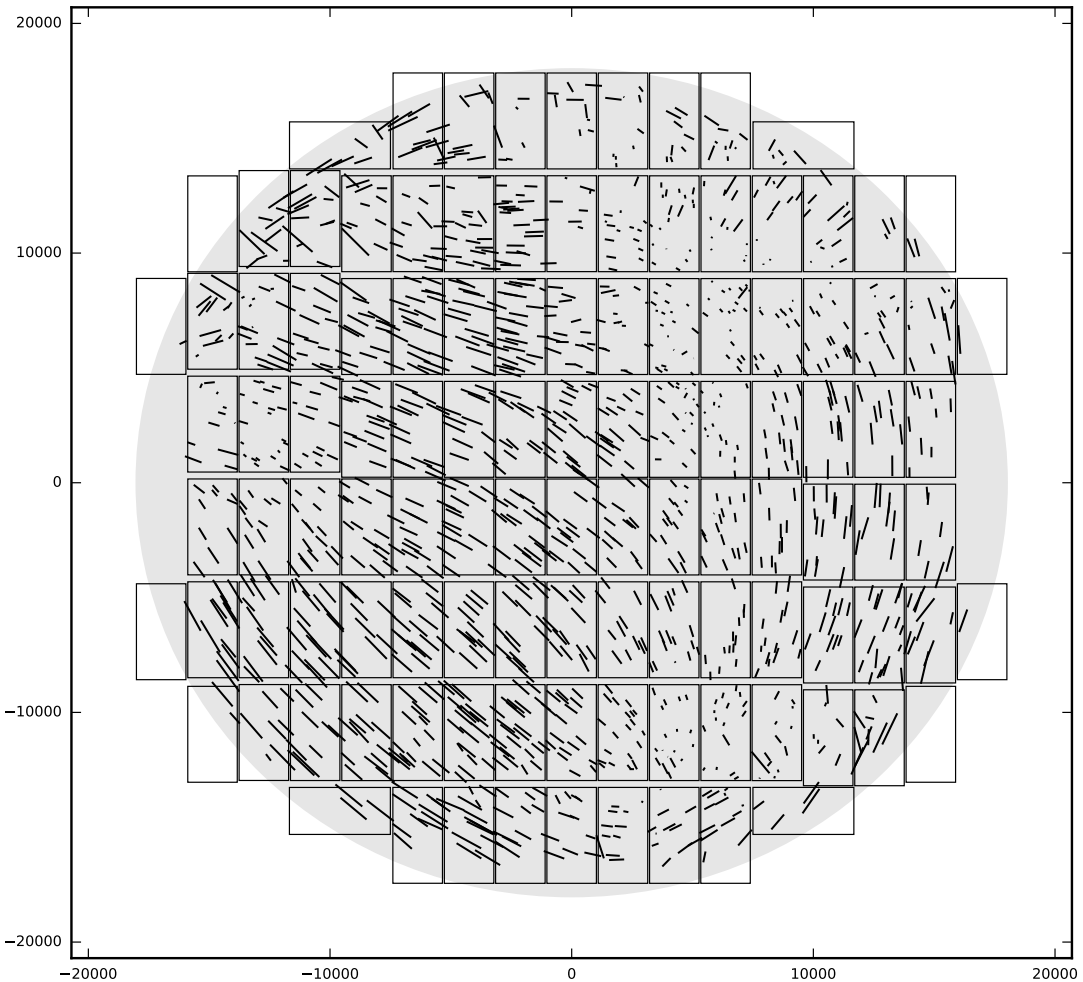


HSC Wavefronts



PSF radius

HSC Wavefronts



e_1, e_2

Processing Flow

- Coadd creation:
  Per-band coadds are created in *grizy*.
- PSF estimation

Processing Flow

- Coadd creation:
  Per-band coadds are created in *grizy*.
- PSF estimation
  The PSF is discontinuous at CCD and field boundaries and wherever you've masked pixels, so we add up all the contributing PSFs for each object to define its personal PSF

Processing Flow

- Coadd creation:
  Per-band coadds are created in *grizy*.
- PSF estimation
  The PSF is discontinuous at CCD and field boundaries and wherever you've masked pixels, so we add up all the contributing PSFs for each object to define its personal PSF "coaddPsf" – I first heard this suggested by James Jee.

Processing Flow

- Coadd source detection:
  The primary detection is on the scale of the PSF resulting in a set of pixels — a `Footprint`. Each `Footprint` has one or more Peaks and these sets of Peaks define our objects.

Processing Flow

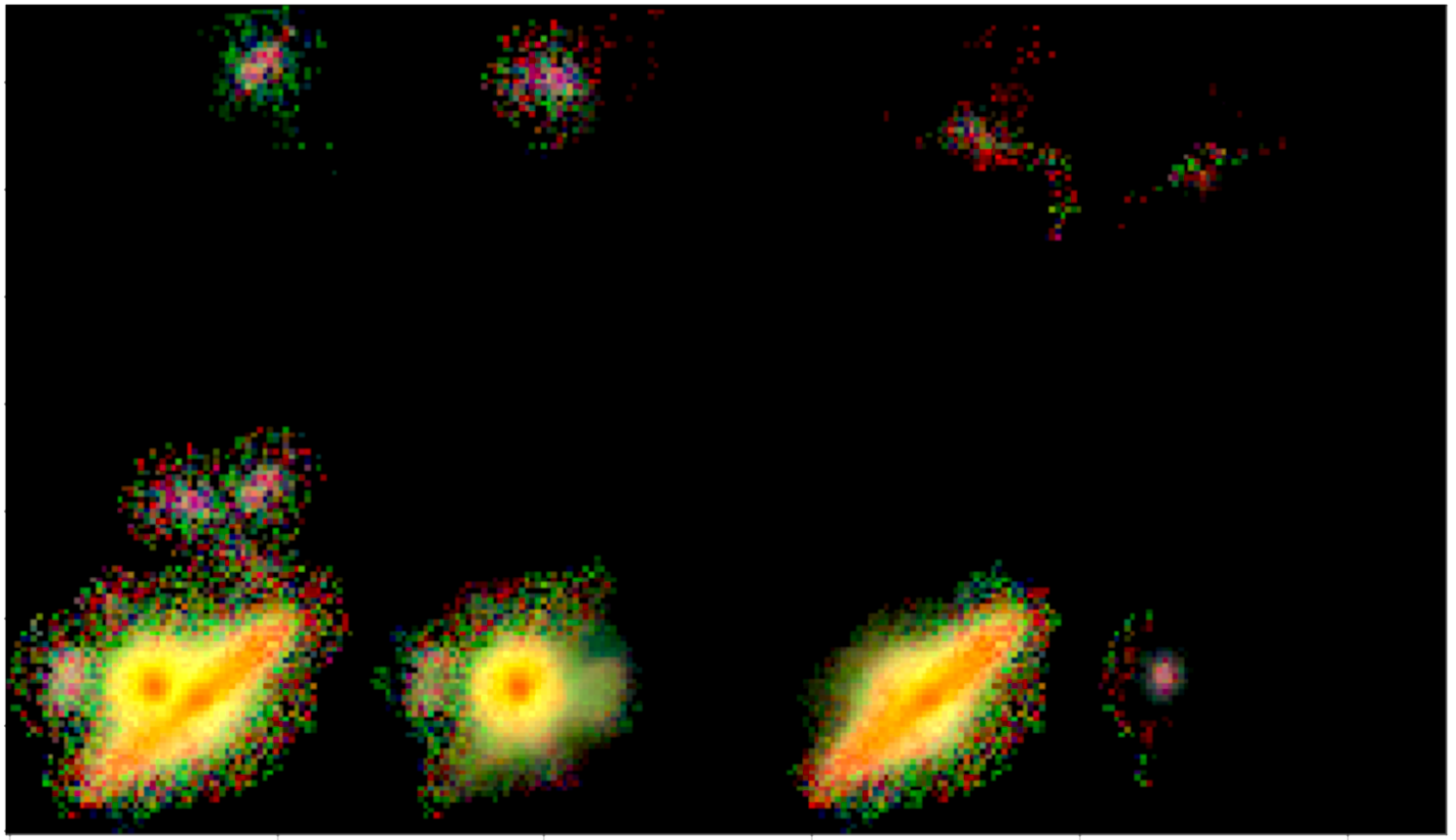
- Coadd source detection:
  The primary detection is on the scale of the PSF resulting in a set of pixels — a `Footprint`. Each `Footprint` has one or more Peaks and these sets of Peaks define our objects.
- Association and deblending:
  Synthesize a list of unique objects by merging Peaks from each band.
  Output a list of uncharacterized deblended (*i.e.* isolated) `Objects`.

Deblender

We’re currently using a variant of my SDSS deblender, which relies on 2-fold rotional symmetry of objects.

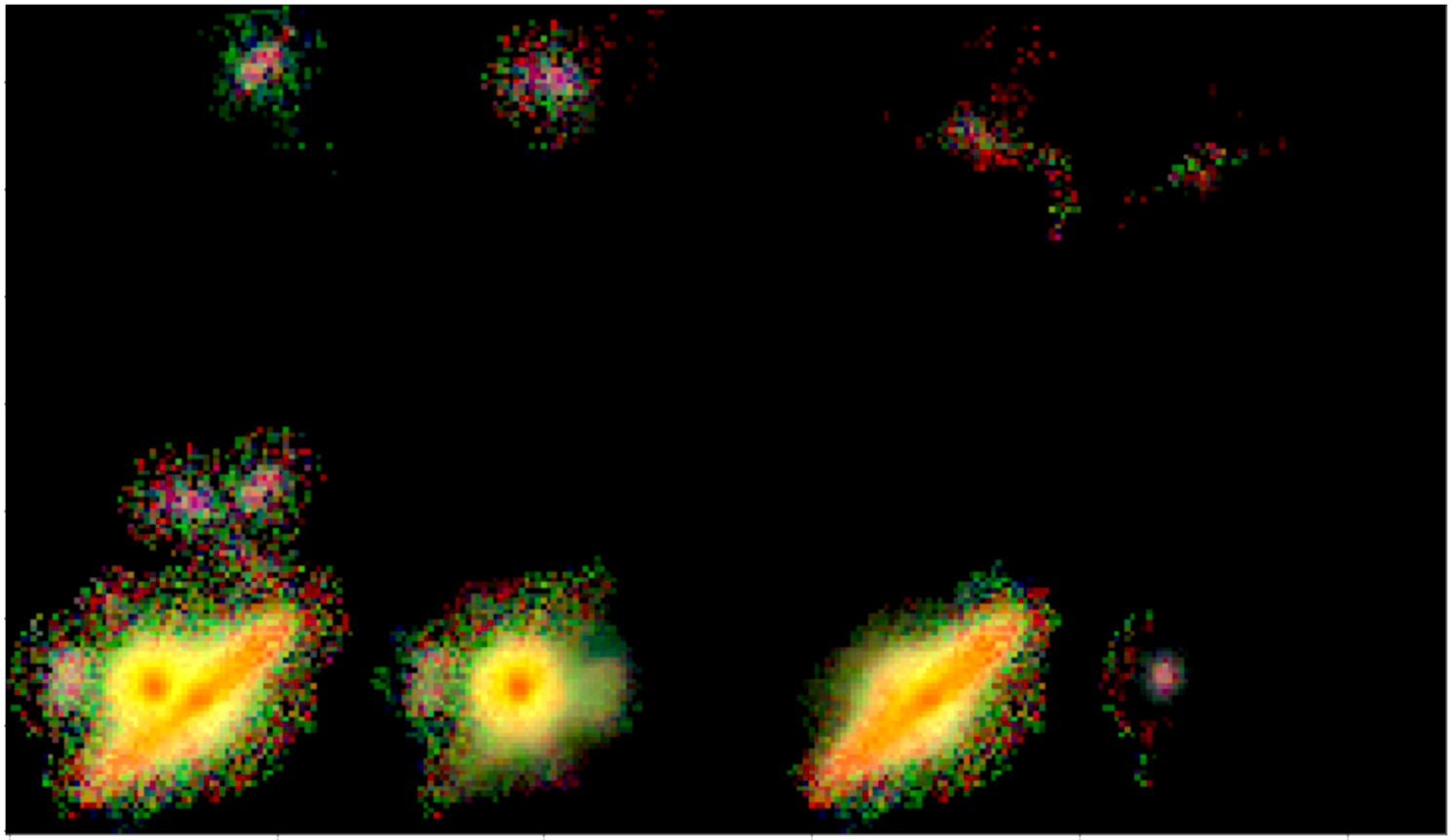
Deblender

We’re currently using a variant of my SDSS deblender, which relies on 2-fold rotional symmetry of objects. This works pretty well, except in cores of clusters.



Deblender

We’re currently using a variant of my SDSS deblender, which relies on 2-fold rotional symmetry of objects. This works pretty well, except in cores of clusters. See Peter Melchior’s talk.



Processing Flow

- `Object` characterization on coadds:
  Measure `Object` properties such as:
  - centroids
  - adaptive moments
  - PSF and Kron/Petrosian fluxes
  - "fibre" aperture fluxes
  - simple "cmodel" galaxy models
  - "HSM" shapes

Processing Flow

- `Object` characterization on coadds:
  Measure `Object` properties such as:
  - centroids
  - adaptive moments
  - PSF and Kron/Petrosian fluxes
  - "fibre" aperture fluxes
  - simple "cmodel" galaxy models
  - "HSM" shapes
- Forced Photometry:
  Measure fluxes for every `Object` on every visit keeping all non-photometric parameters fixed

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

The only options that I know of:

- PSF match, then clip

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

The only options that I know of:

- PSF match, then clip. But faint features still survive.

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

The only options that I know of:

- PSF match, then clip. But faint features still survive.
- Use a direct mean of the pixel values

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

The only options that I know of:

- PSF match, then clip. But faint features still survive.
- Use a direct mean of the pixel values. But this allows the junk to leak into the coadd.

Coadd Types

Individual images contain asteroids, satellites, ghosties, ghoulies, cosmic rays, and other undesireable features; collectively "junk".

If you use σ-clipping (or a median) to remove the junk you also modify objects' PSF, depending on their signal-to-noise.

In particular, you cannot use bright stars to estimate the PSF.

The only options that I know of:

- PSF match, then clip. But faint features still survive.
- Use a direct mean of the pixel values. But this allows the junk to leak into the coadd.
- Make use of the differences between images to remove the junk, then use a mean

"Safe Clipping"

In the long run we'll detect-and-mask on difference images, but we're currently using a simple scheme which is better than nothing

"Safe Clipping"

In the long run we'll detect-and-mask on difference images, but we're currently using a simple scheme which is better than nothing:

- Make a heavily clipped coadd
- Calculate a direct mean coadd
- Subtract the `coadds`
- Detect the junk (and cores of bright objects) on the difference
- Remove the junk that's in only one image (leaving the cores of bright objects)

"Safe Clipping"

In the long run we'll detect-and-mask on difference images, but we're currently using a simple scheme which is better than nothing:

- Make a heavily clipped coadd
- Calculate a direct mean coadd
- Subtract the `coadds`
- Detect the junk (and cores of bright objects) on the difference
- Remove the junk that's in only one image (leaving the cores of bright objects)
- Calculate a direct mean of what's left, a "Safe Clipped Coadd"

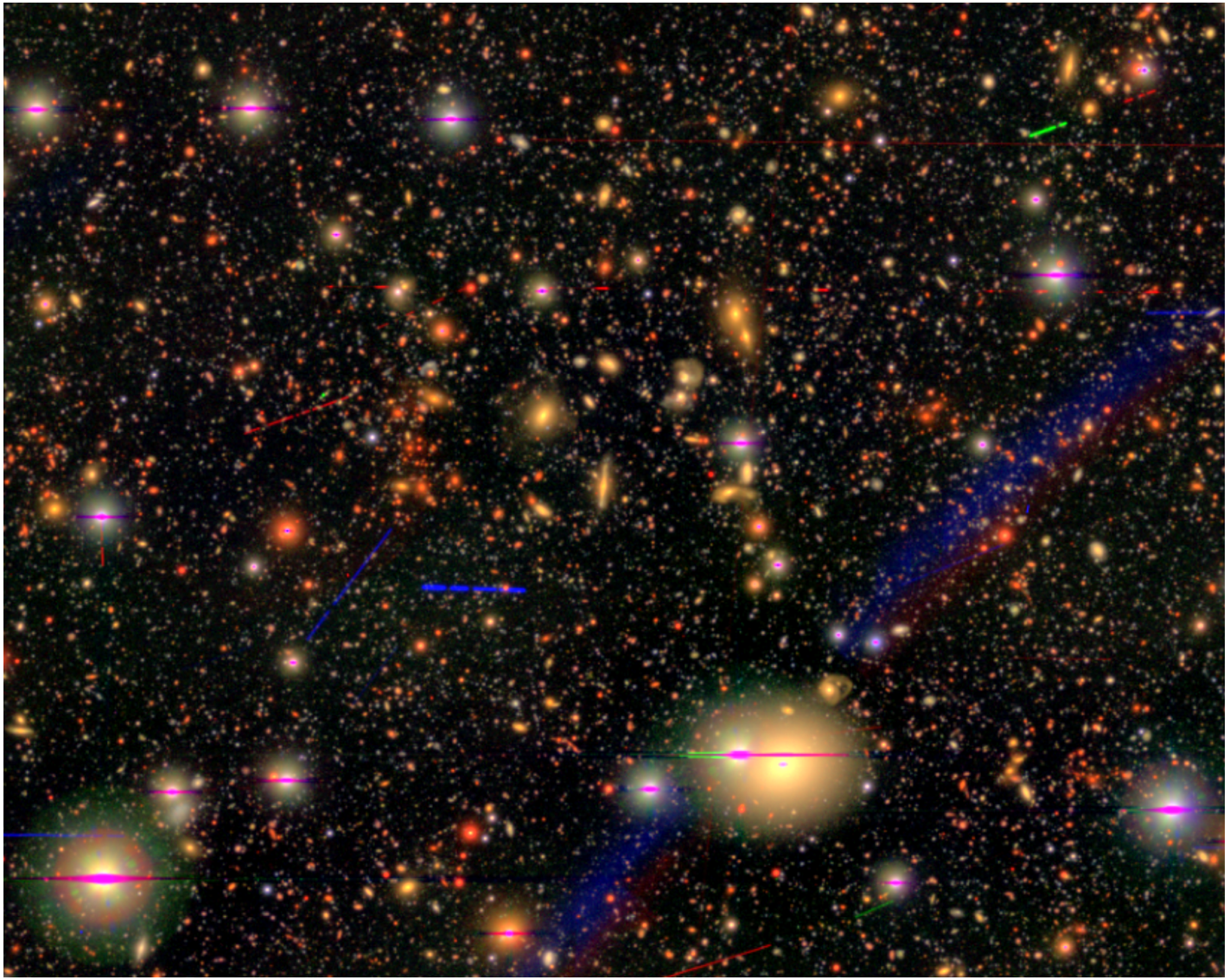
PSF Matched Coadds

We have code to create PSF-matched images, but it leads to correlated noise.

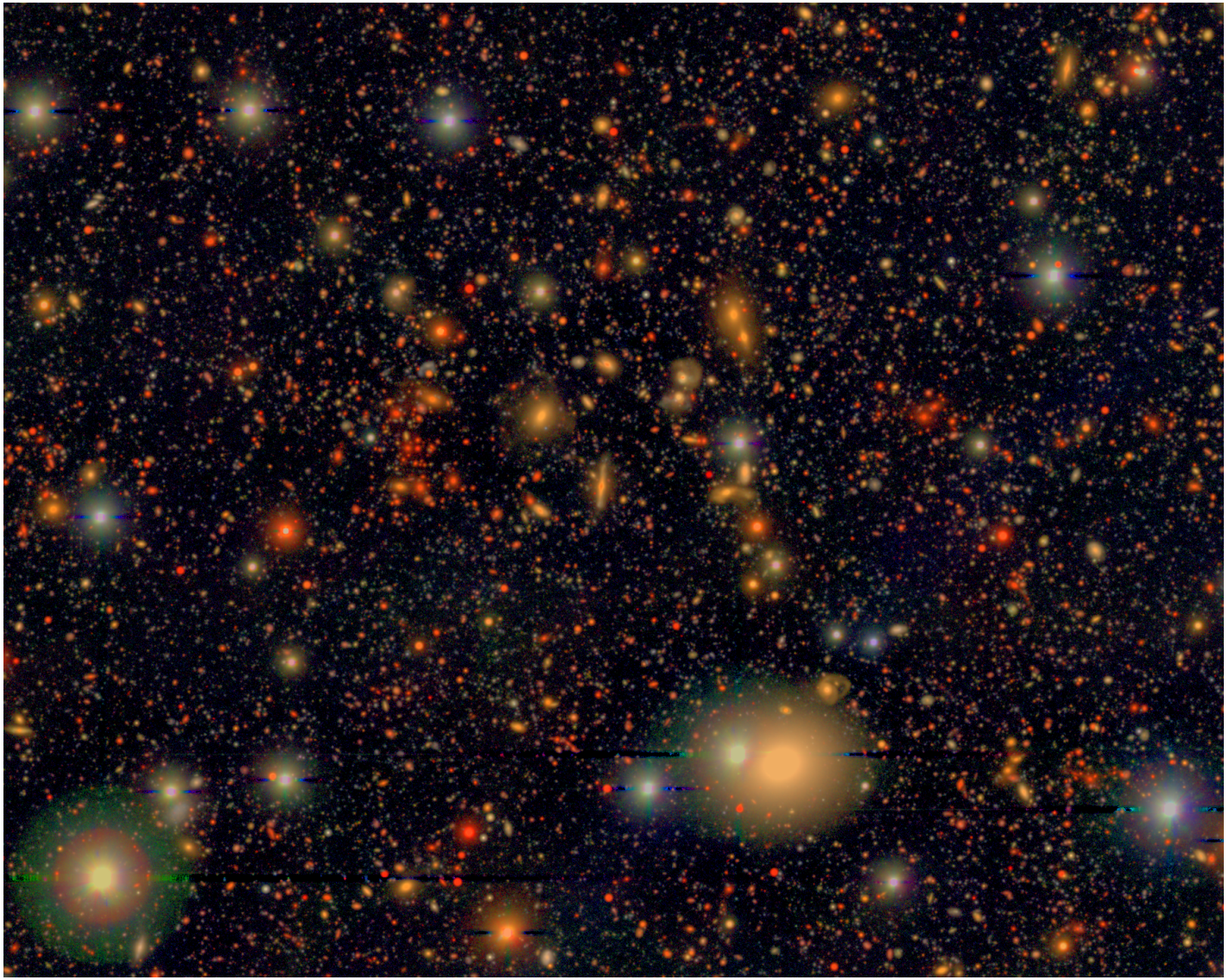
PSF Matched Coadds

We have code to create PSF-matched images, but it leads to correlated noise.
As Nick Kaiser pointed out in 2001 (PSDC-002-011-00), and Zackay, Ofek, and Gel-Yam recently rediscovered, you can build a coadd with optimal properties by down-weighting components that are suppressed by the images' PSFs, and then flattening the noise.
We don't currently use these coadds for HSC (it's a bit complicated with a spatially varying PSF)

Safe-Clipping Coadds



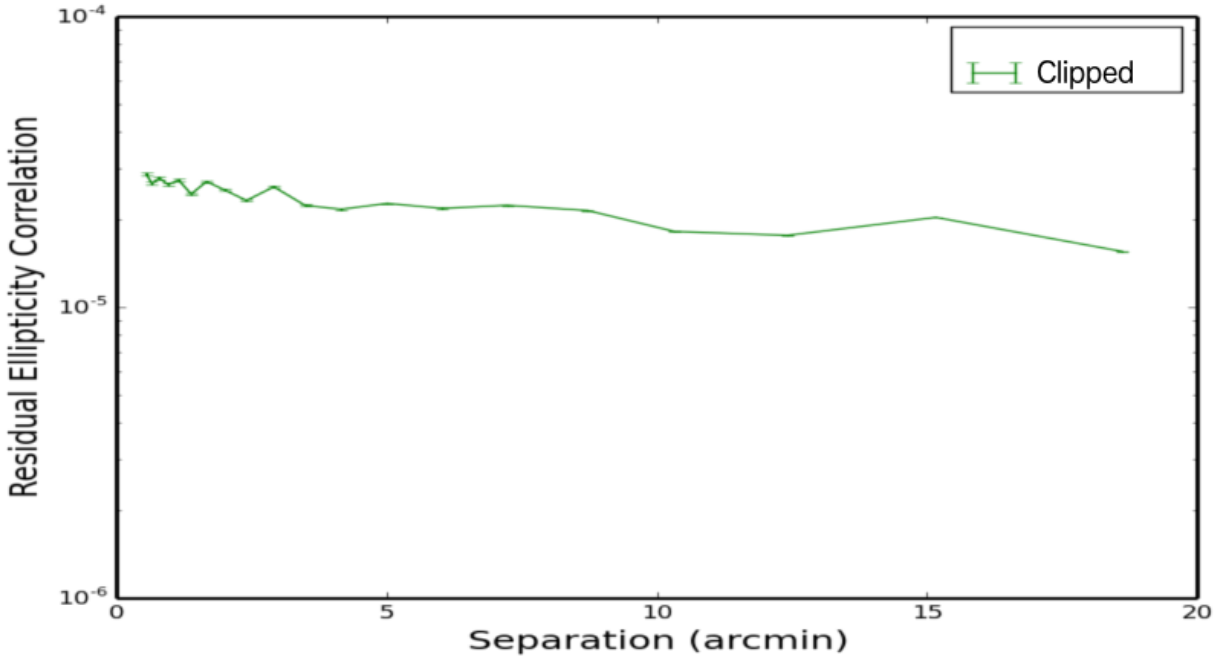
PSF-matched Coadds



Does Clipping Really Matter?

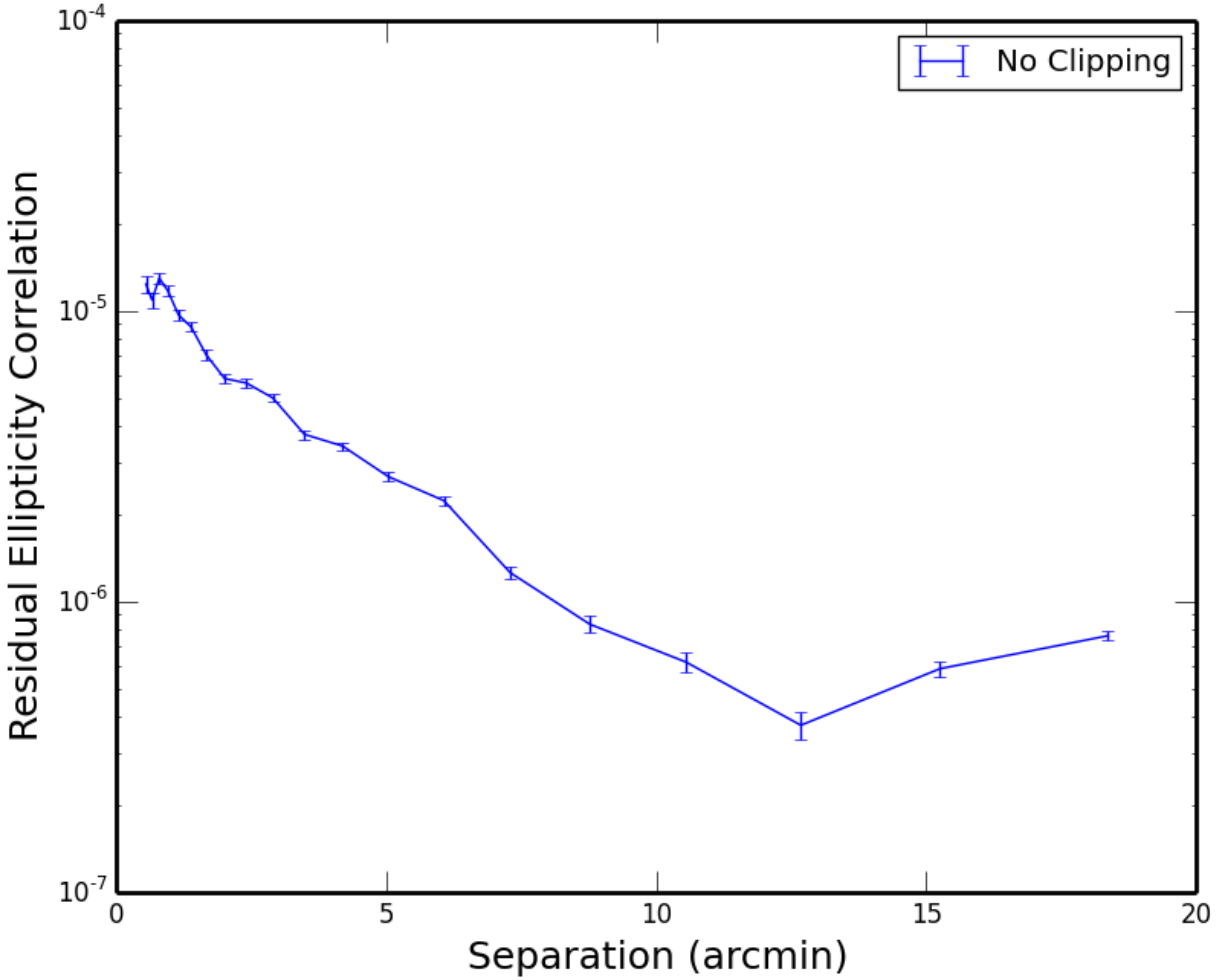
Does Clipping Really Matter?

Courtesy Bob Armstrong σ Clipped



Does Clipping Really Matter?

Courtesy Bob Armstrong Mean



"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

- SDSS Cas/Casjobs was an early attempt to do this

"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

- SDSS Cas/Casjobs was an early attempt to do this
- So are the NAOJ HSC catalogues

"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

- SDSS Cas/Casjobs was an early attempt to do this
- So are the NAOJ HSC catalogues

Unfortunately SQL isn't a very expressive language.

"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

- SDSS Cas/Casjobs was an early attempt to do this
- So are the NAOJ HSC catalogues

Unfortunately SQL isn't a very expressive language.
Recently many people have realised the *Jupyter Notebooks* provide a way that people can really compute close to the data.

"Take the code to the data"

For 20 years we've been saying that we should "take the code to the data"

- SDSS Cas/Casjobs was an early attempt to do this
- So are the NAOJ HSC catalogues

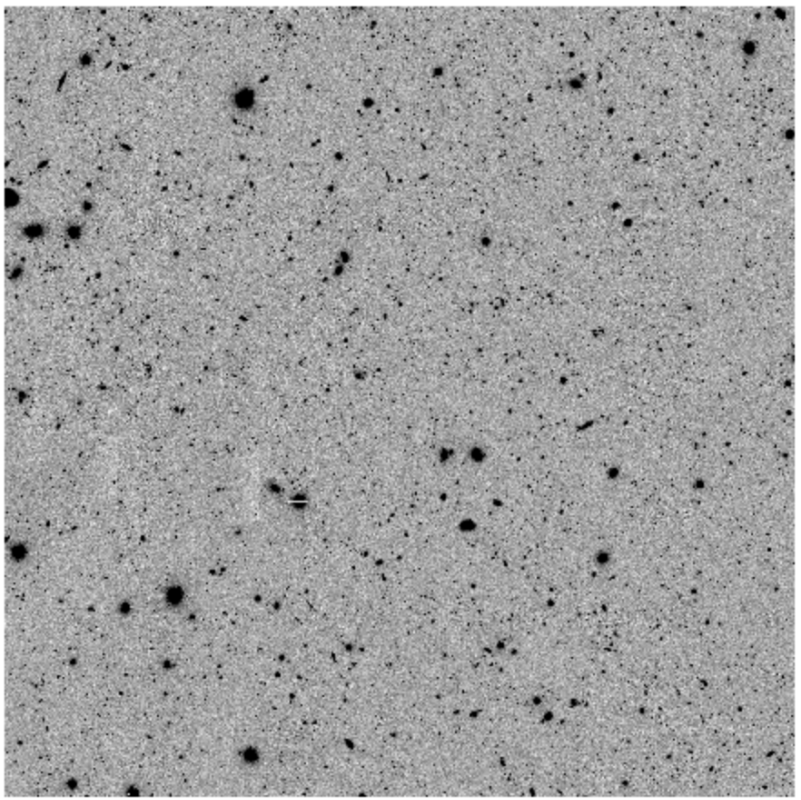
Unfortunately SQL isn't a very expressive language.
Recently many people have realised the *Jupyter Notebooks* provide a way that people can really compute close to the data.
I think it's time to move to the next phase:

Keep the Code with the Data

Processing HSC data with Jupyter

Get HSC calibrated exposure

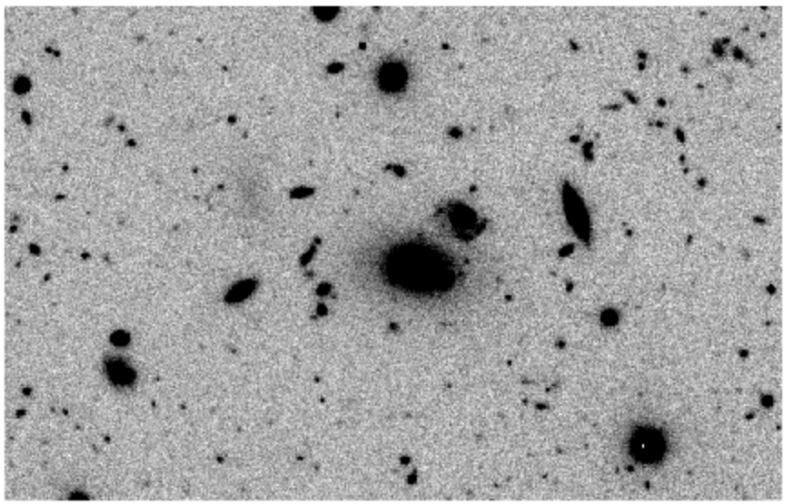
In [5]:
```
dataId = dict(tract=9348, patch='7,6', filter='HSC-I')
exp = butler.get('deepCoadd_calexp',dataId)
display_image(exp.getMaskedImage().getImage().getArray(), figsize=(8,8));
```



Processing HSC data with Jupyter

Make cutout

In [6]:
```
bbox = afwGeom.Box2I()
bbox.include(afwGeom.Point2I(2700, 3600))
bbox.include(afwGeom.Point2I(3330, 4000))
exp_cutout = exp.Factory(exp, bbox, afwImage.LOCAL)

mi = exp_cutout.getMaskedImage()
mask = mi.getMask()
mask.removeAndClearMaskPlane('CR', True)

display_image(mi.getImage().getArray());
```



Processing HSC data with Jupyter

Smooth at PSF scale

In [7]:
```
psf = exp.getPsf()
sigma = psf.computeShape().getDeterminantRadius()
ngrow = int(3*sigma)
mi_smooth = smooth_gauss(mi, sigma)
print('psf fwhm = {:.2f} arcsec'.format(sigma*0.168*2.355))
```

```
psf fwhm = 0.52 arcsec
```

Image thresholding

- Find high-threshold sources that are 6 σ above the background and create a mask plane

In [8]:
```
high_thresh = 6
fpset_high = afwDetect.FootprintSet(
    mi_smooth, afwDetect.Threshold(high_thresh, afwDetect.Threshold.STDEV), 'HIGH_THRESH', 1)
fpset_high = afwDetect.FootprintSet(fpset_high, ngrow)
fpset_high.setMask(mask, 'HIGH_THRESH')
```

- Find low-threshold sources that are 1.5 σ above the background and create a mask plane

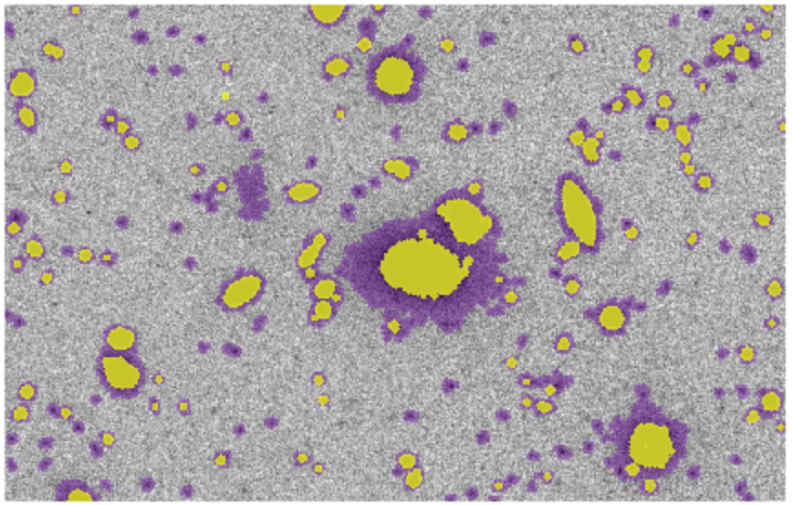
In [9]:
```
low_thresh = 1.5
fpset_low = afwDetect.FootprintSet(
    mi_smooth, afwDetect.Threshold(low_thresh, afwDetect.Threshold.STDEV), 'LOW_THRESH', 10)
fpset_low = afwDetect.FootprintSet(fpset_low, ngrow)
fpset_low.setMask(mask, 'LOW_THRESH')
```

Processing HSC data with Jupyter

Visualize segmentation maps

- low-threshold footprints are purple
- high-threshold footprints are yellow

In [10]:
```
seg_high = fpset_high.insertIntoImage(True).getArray().copy().astype(float)
seg_low = fpset_low.insertIntoImage(True).getArray().copy().astype(float)
seg_low[seg_low>0] = 1.0
seg_low[seg_low==0] = np.nan
seg_high[seg_high>0] = 1.0
seg_high[seg_high==0] = np.nan

ax = display_image(mi.getImage().getArray())
kws = dict(cmap='plasma', vmin=0, vmax=1, origin='lower')
ax.imshow(seg_low*0.2, alpha=0.4, **kws)
ax.imshow(seg_high*1.0, alpha=0.8, **kws);
```



Processing HSC data with Jupyter

Remove bright sources and associated diffuse light

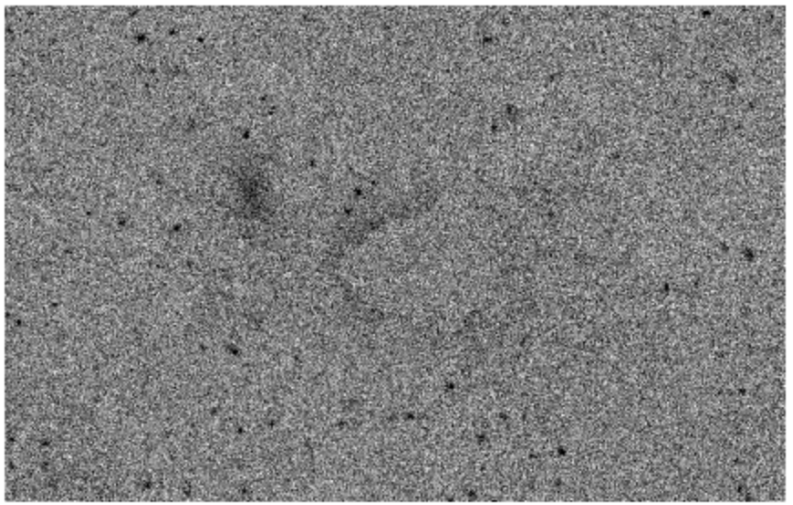
- associate a low-thresh footprint with a high-thresh footprint if more than 5% of its light is above the high threshold

In [11]:
```
max_frac = 0.05

# loop over low threshold footprints
# and save those with >max_frac high
# threshold pixels
fp_list = afwDetect.FootprintList()
for fp in fpset_low.getFootprints():
    hfp = afwDetect.HeavyFootprintF(fp, mi)
    pix = hfp.getMaskArray()
    bits_high = (pix & mask.getPlaneBitMask('HIGH_THRESH')!=0).sum()
    if bits_high/float(fp.getArea()) > max_frac:
        fp_list.append(fp)

# build a new footprint set
fpset_replace = afwDetect.FootprintSet(mi.getBBox())
fpset_replace.setFootprints(fp_list)

# create a mask plane for "cleaned" footprints
if 'CLEANED' in mask.getMaskPlaneDict().keys():
    mask.clearMaskPlane(mask.getMaskPlane('CLEANED'))
mask.addMaskPlane('CLEANED')
fpset_replace.setMask(mask, 'CLEANED')
```

Processing HSC data with Jupyter

- replace footprints associated with bright sources with sky noise

In [12]:
```
exp_clean = exp_cutout.clone()
mi_clean = exp_clean.getMaskedImage()
back_rms = mi.getImage().getArray()[mask.getArray()==0].std()
noise_array = back_rms*np.random.randn(*mask.getArray().shape)
replace = mask.getArray() & mask.getPlaneBitMask('BRIGHT_OBJECT') != 0
replace |= mask.getArray() & mask.getPlaneBitMask('CLEANED') != 0
mi_clean.getImage().getArray()[replace] = noise_array[replace]
display_image(mi_clean.getImage().getArray());
```



Processing HSC data with Jupyter

- smooth to emphasize extended low-surface-brightness structures

In [13]:
```
smoothed = smooth_gauss(mi_clean, 4)
display_image(smoothed.getImage().getArray());
```