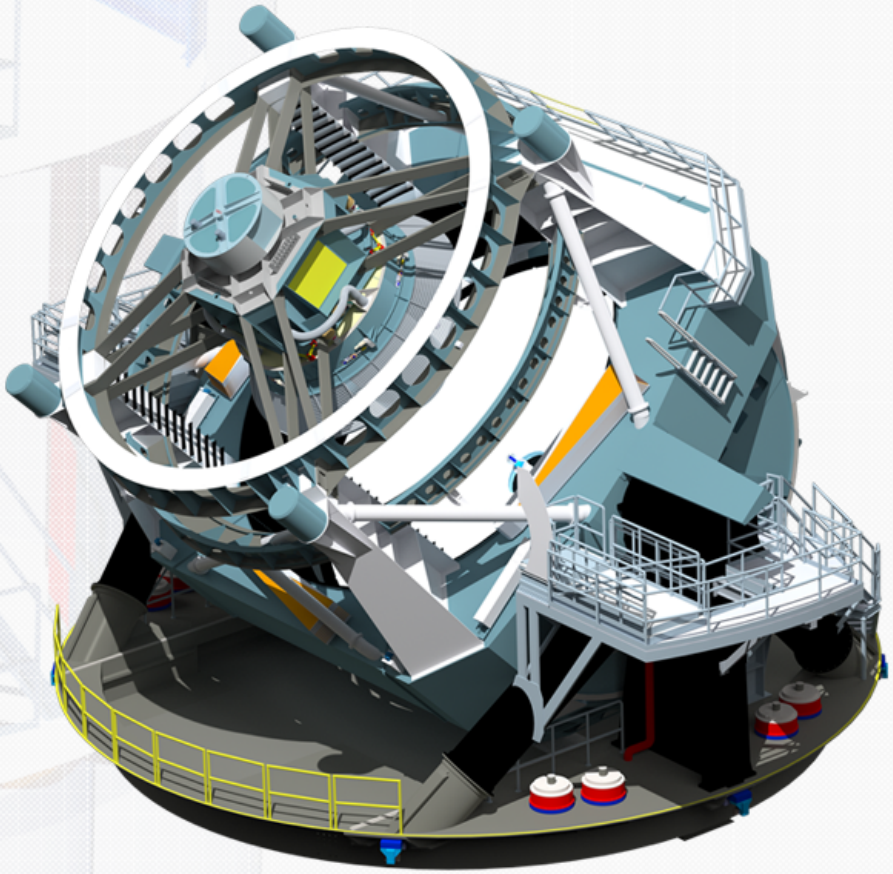
afw

Simon Krughoff
UW DM T/CAM

October 6, 2015



Contents of afw

- cameraGeom
  - A system for representing transformation between different coordinate systems in the optical system.
  - Utilities for building cameras
  - Cameras are typically built by the instrument mapper (in the obs_ package)
  - Utilities for visualizing camera layouts

Contents of afw

- Coord
  - Coordinate construction and conversion utilities
  - Implements the following coordinate system
    - FK5
    - ICRS
    - Ecliptic
    - Galactic
    - Topocentric
  - Very basic Observatory container (lat, lon, elevation)

Contents of afw

- Detection
  - Footprint (and HeavyFootprint)
  - Threshold
    - bitmask, value, sigma, sigma per pixel
  - Psf

Contents of afw

- geom (not lsst.geom…)
  - Simple geometry constructs – Angle, Box, Extent, Point, Span
  - More complicated geometry – Ellipse, Polygon
  - XYTransforms – Affine, Identity, Inverted, Multi, Radial, Separable

Contents of afw

- image
  - Image like things:
    - Image – a single grid of pixels (float, double, int, uint)
    - Mask – a grid of bit mask pixels with associated mask plane definitions.
    - DecoratedImage – Image with metadata (deprecated)
    - MaskedImage – Image + Mask + Variance
    - Exposure – MaskedImage with associated image things: WCS, Psf, metadata, calibration info
  - Other associated things – Defect, Filter, Calib (this is really photometric calibration), Wcs
  - Utilities for dealing with images

Contents of afw

- math
  - Statistics – mean, stdev, var, median, inner quartile range, clipped stats, min, max, sum
  - Kernels
  - Convolution
  - Interpolation and approximation
  - Fitting
  - Functions – Gaussian, Polynomial, Chebyshev, Double Gaussian
  - Splines
  - Random number generator
  - Warping – Lanczos, bilinear, NN

Contents of afw

- table
  - Tables are really catalogs with fixed schema. The schema is flexible and can be set up to do lots of things.
    - Store amplifier electronics info: AmpInfoTable
    - Source catalogs
    - Matched reference catalogs to source catalogs

How to find things

- Doxygen
  http://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/afw.html
- GitHub code search
  - Can be useful, but has significant limitations (full word search only)
  - I find the tree browsing features very useful
- Unit tests
- Help strings in Python
  - Useful info is not always forwarded from C++
- Searching with an editor
  - Sublime Text, Emacs, and vim are all popular choices

Let's take a look at afw

This code comes from the short script at:
https://github.com/lsst-dm/Oct15_bootcamp/blob/master/code/afw_talk.py

The intent is just to introduce a few of the useful afw classes in a toy scenario.

Make sure you setup the display_ds9 package if you want to display to be output to your local ds9.

Let's take a look at afw

Start by making an image from a bounding box:

```
import lsst.afw.image as afwImage
import lsst.afw.geom as afwGeom
from lsst.pex.exceptions import LengthError

n_objects = 1000

box = afwGeom.BoxI(afwGeom.PointI(300, 500),
                   afwGeom.ExtentI(2000, 2048))

im = afwImage.ImageF(box)
print im.getXY0()
>>> (300, 500)
```

Let's take a look at afw

Note that the LLC is not at 0,0. This bounding box is relative to a global coordinate system called PARENT (the default).

Now try to construct a view into a sub-region of the image we just made:

```
subbox = afwGeom.BoxI(afwGeom.PointI(10, 10),
                      afwGeom.ExtentI(100, 100))
try:
    im2 = afwImage.ImageF(im, subbox)
except LengthError:
    im2 = afwImage.ImageF(im, subbox, afwImage.LOCAL)
```

Let's take a look at afw

We have an image. Let's put something in it. First create some random positions to put down Gaussian spots.

```
import lsst.afw.math as afwMath

rand = afwMath.Random()
buffer_xy = 150 # Don't put objects near the edges
x_positions = [rand.uniformInt(im.getWidth() - 2*buffer_xy)
               + buffer_xy for i in xrange(n_objects)]
y_positions = [rand.uniformInt(im.getHeight() - 2*buffer_xy)
               + buffer_xy for i in xrange(n_objects)]
```

Let's take a look at afw

Set up the display and create an image of a Gaussian PSF to put in the image.

```
import lsst.afw.detection as afwDetect
import lsst.afw.display as afwDisplay

display = afwDisplay.getDisplay()
display.setMaskTransparency(50, None

psf_size = 121 # This has to be odd
sigma = 0.7/0.2 # seeing in arcsec/pixel size in arcsec
peak_val = 6000

psf = afwDetect.GaussianPsf(psf_size, psf_size, sigma)
psf_im = psf.computeImage()
```

Let's take a look at afw

Not shown, the PSF image is normalized to a realistic value. Now add the images at the random positions. Look out for the gotcha.

```
for x, y in zip(x_positions, y_positions):
    x0 = x - (psf_size - 1)/2
    y0 = y - (psf_size - 1)/2
    box = afwGeom.BoxI(afwGeom.PointI(x0, y0),
                       afwGeom.ExtentI(psf_size, psf_size))
    subim = afwImage.ImageF(im, box, afwImage.LOCAL)
    try:
        subim += psf_im
    except NotImplementedError:
        psf_im = psf_im.convertF()
        subim += psf_im
```

Let's take a look at afw

We now have an image populated with Gaussian blobs. Let's add some background noise, and turn this into a proper MaskedImage.

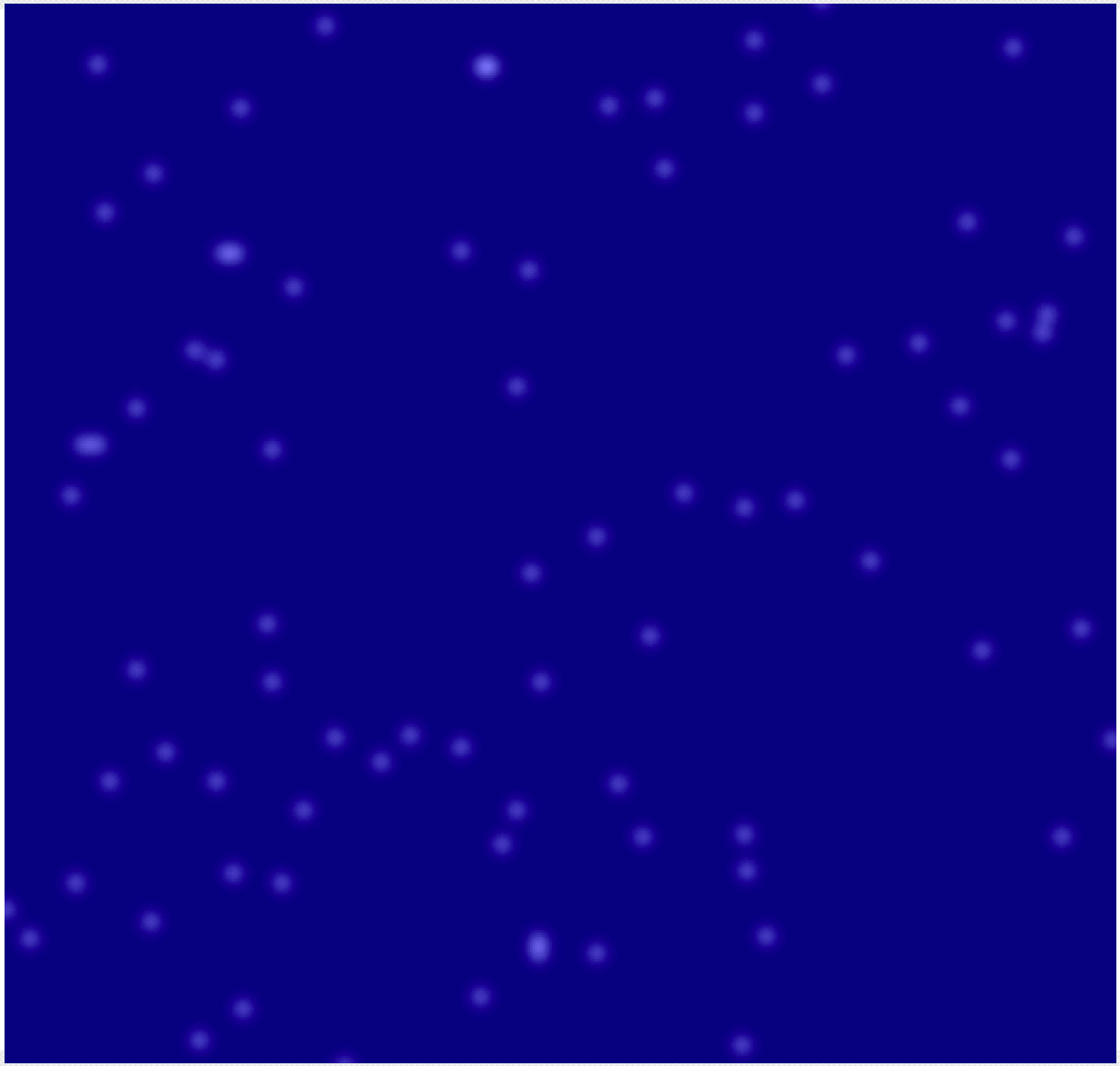
```
back_im = afwImage.ImageF(im.getBBox())
afwMath.randomPoissonImage(back_im, rand, 1000)
im += back_im
display.mtv(im)
display.incrDefaultFrame()

mask = afwImage.MaskU(im.getBBox())
masked_im = afwImage.MaskedImageF(im, mask, im)
```

Let's take a look at afw

And now to do a naïve detection. This will set the DETECTED mask plane so we should be able to look at the footprints in ds9.

```
threshold = afwDetect.createThreshold(5., 'stdev')
fs = afwDetect.FootprintSet(masked_im, threshold,
                            'DETECTED')
display.mtv(masked_im)
display.incrDefaultFrame()
```

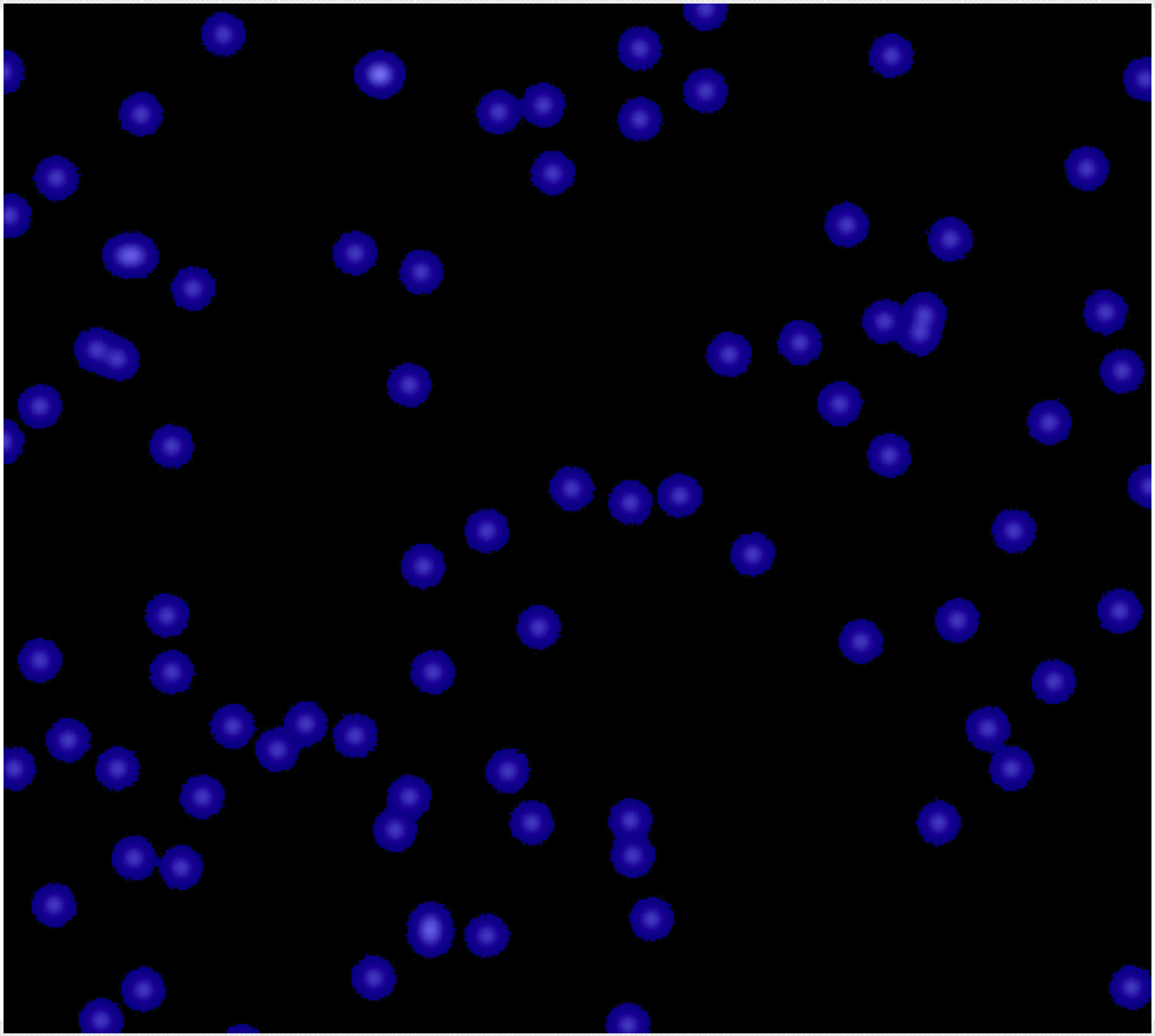


Let's take a look at afw

Of course everything is detected because the background noise floor is more than 5 sigma from zero. We need to estimate then subtract the background.

```
bctrl = afwMath.BackgroundControl(11, 11)
bkgd = afwMath.makeBackground(masked_im, bctrl)
masked_im -= bkgd.getImageF()

masked_im.getMask().set(0) # reset mask
fs = afwDetect.FootprintSet(masked_im, threshold,
                            'DETECTED')
display.mtv(masked_im)
display.incrDefaultFrame()
```



Let's take a look at afw

There are lots of other features of afw. Here are a few.

```
# numpy arrays from images
im, mask, var = masked_im.getArrays()
print type(im)
print im.dtype
```

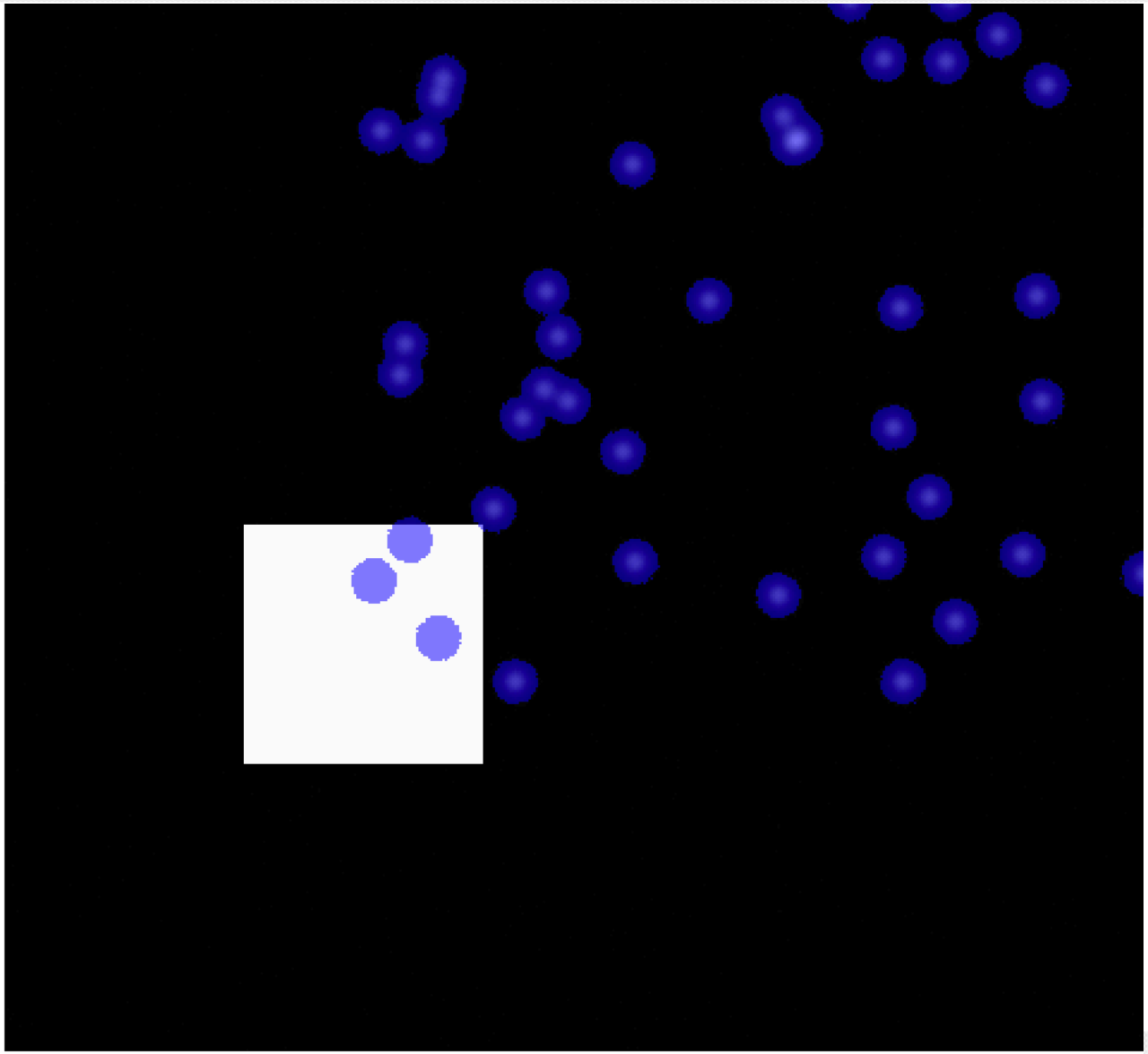
Let's take a look at afw

There are lots of other features of afw. Here are a few.

```
# arrays are views
xy0 = masked_im.getXY0()
xy0 = masked_im.getXY0()
xy0.shift(afwGeom.ExtentI(100, 120))
box = afwGeom.BoxI(xy0, afwGeom.ExtentI(100, 100))

subim = afwImage.ImageF(masked_im.getImage(), box)
sub_arr = subim.getArray()
sub_arr[:][:] = im.max()

display.mtv(masked_im)
```

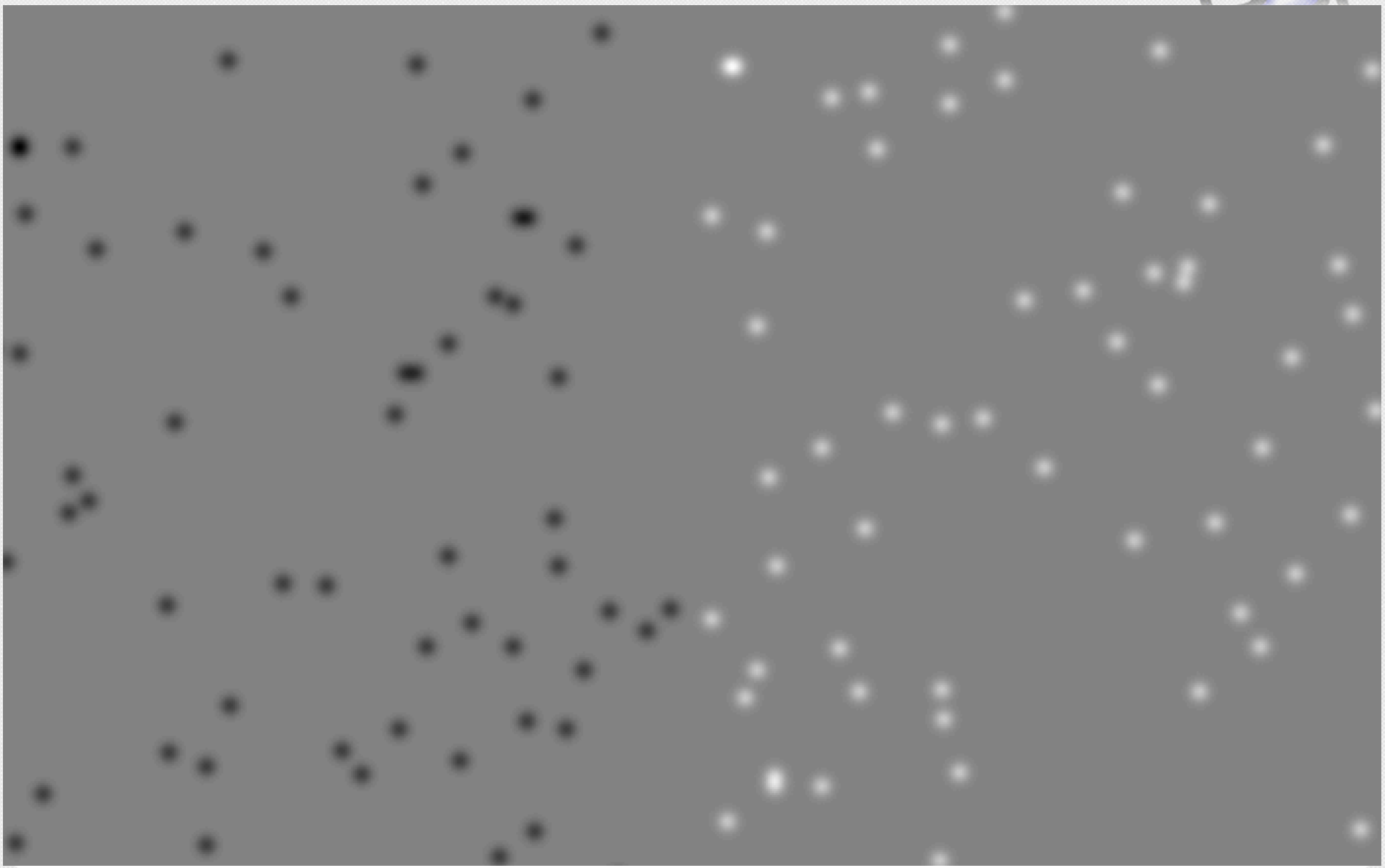


Let's take a look at afw

There are lots of other features of afw. Here are a few.

```
# The >>= operator
left_box = afwGeom.BoxI(afwGeom.PointI(0,0),
                        afwGeom.ExtentI(1000, 2048))
right_box = afwGeom.BoxI(afwGeom.PointI(1000, 0),
                         afwGeom.ExtentI(1000, 2048))
im = masked_im.getImage()
new_im = afwImage.ImageF(masked_im.getBBox())
left_subim = afwImage.ImageF(im, left_box, afwImage.LOCAL)
right_subim = afwImage.ImageF(im, right_box, afwImage.LOCAL)
left_subim *= -1

new_subim = afwImage.ImageF(new_im, left_box, afwImage.LOCAL)
new_subim <<= left_subim
new_subim = afwImage.ImageF(new_im, right_box, afwImage.LOCAL)
new_subim <<= right_subim
```



Understanding afw

- The original intent was to keep the C++ environment rich.
  - This leads to classes (not just functions) defined in C++
- There is a strong effort to expose as much C++ in Python as possible.
  - This can lead to more than one way to do things.

- Takeaway: The Python/C++ line is hard to draw. We should continue to be observant and strive to make the stack as generally useful/useable as possible.