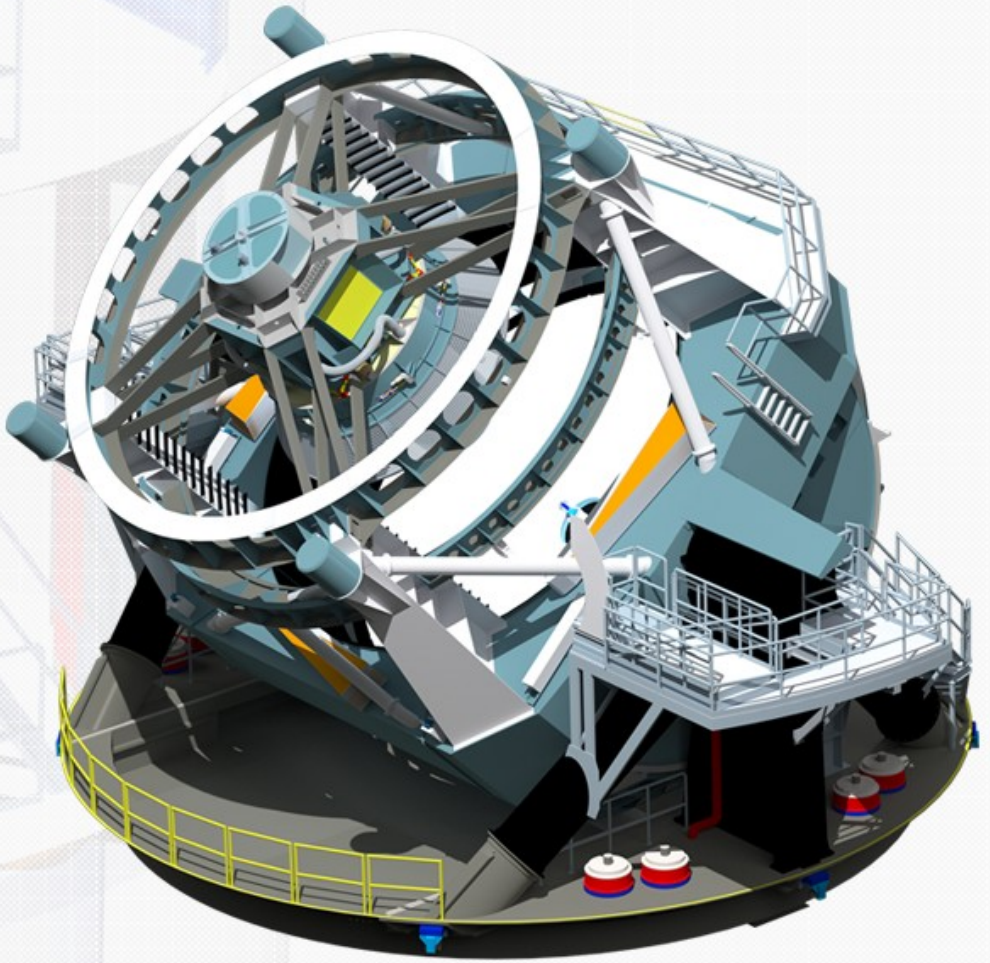
Source Measurement & Tables

Jim Bosch

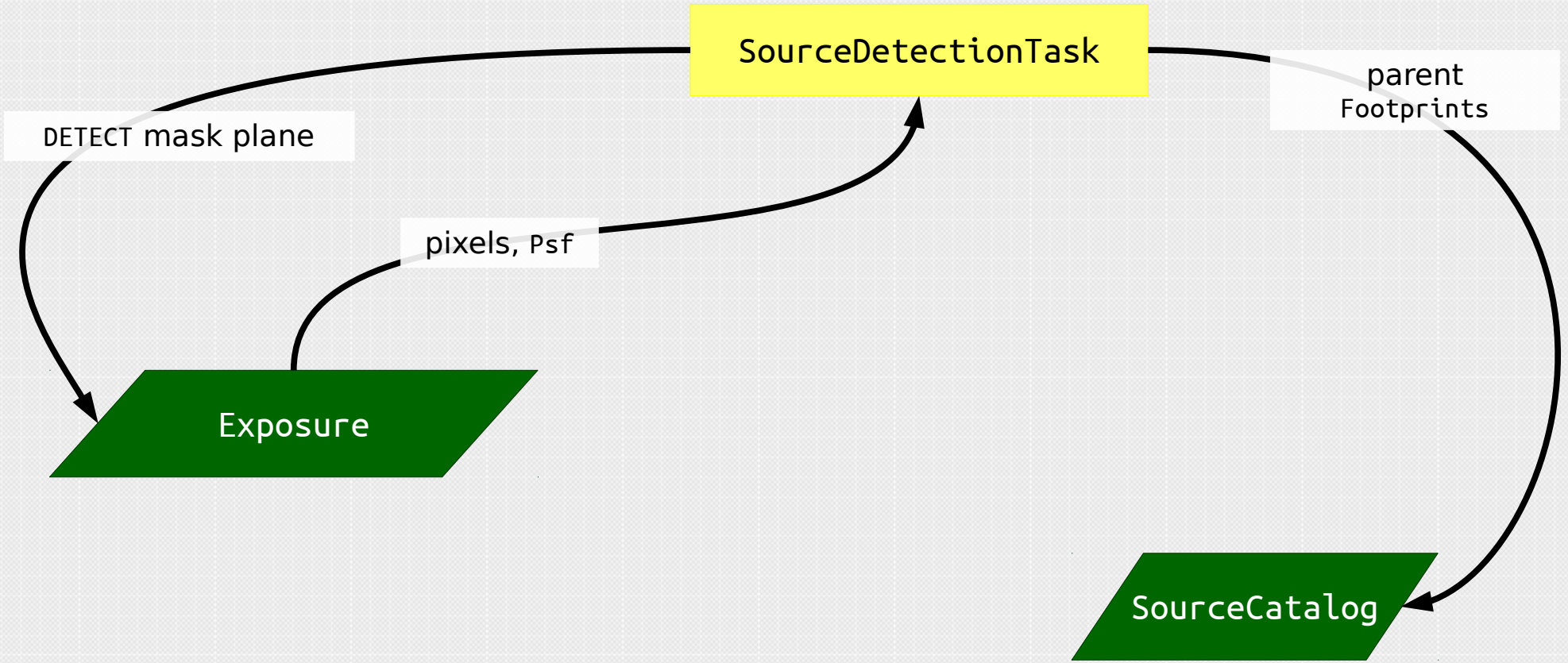
October 7, 2015



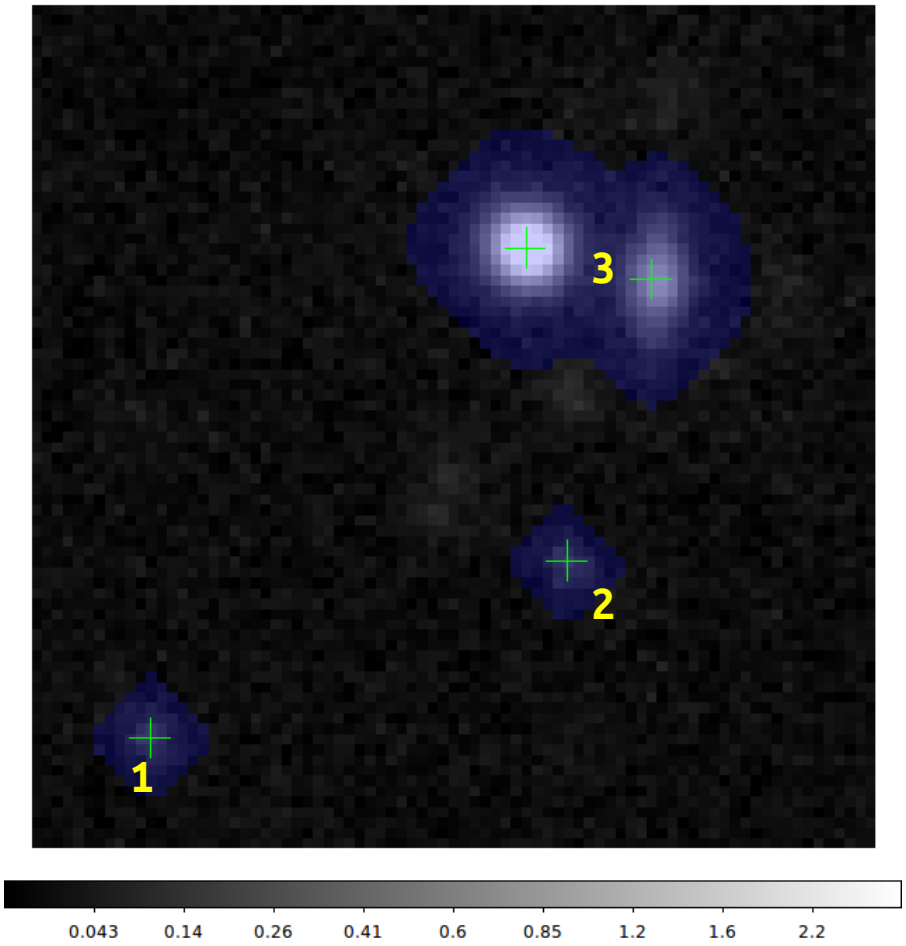
Outline

- **Measurement**
  - Detection, Deblending, and Measurement
  - Inside Measurement
  - Using `SingleFrameMeasurementTask`
  - Writing a `SingleFramePlugin`
  - Unit Transformation
  - Forced Measurement
- **Tables**
  - Records, Tables, and Catalogs
  - Deep and Shallow Copies
  - Blocks, Columns, and Contiguousness
  - `SchemaMapper`
  - Slots and Aliases
  - FunctorKeys

Detect, Deblend, Measure

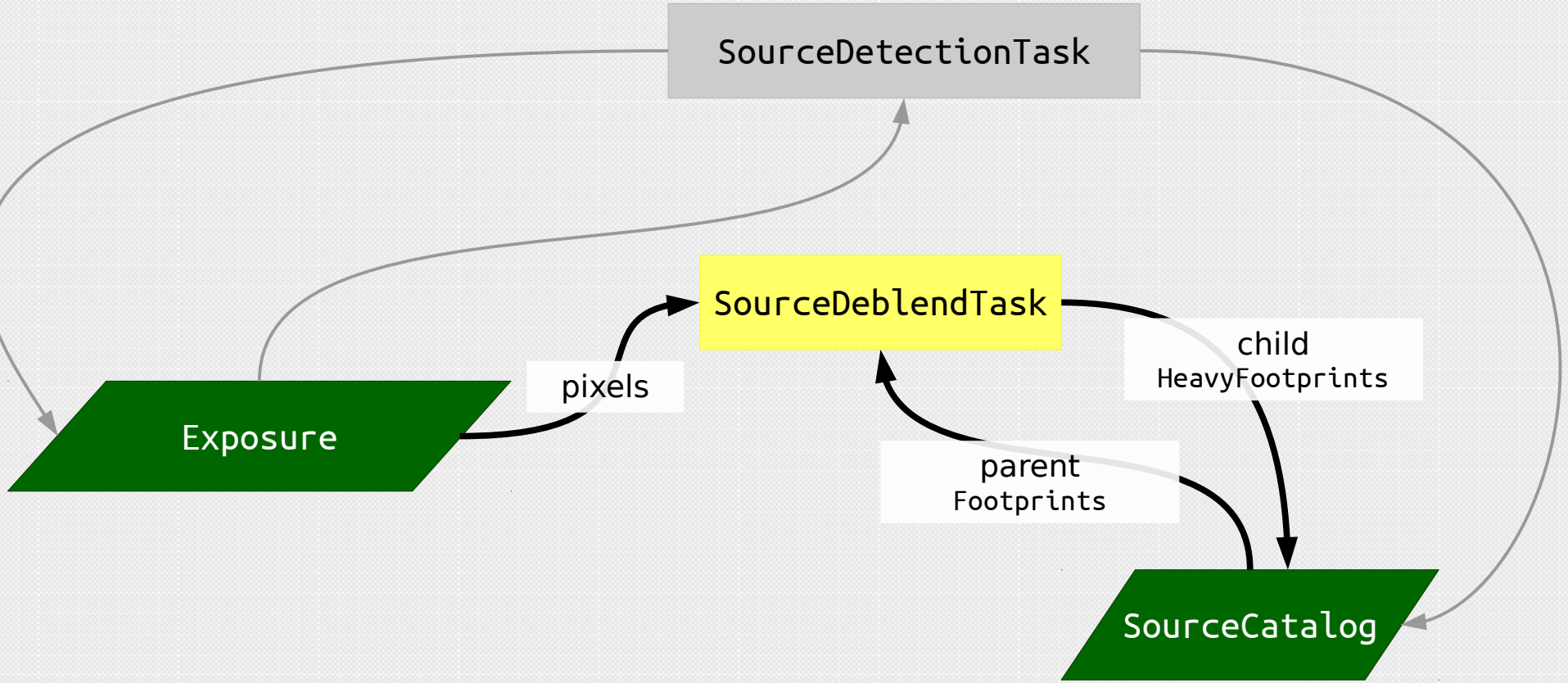


Detect, Deblend, Measure

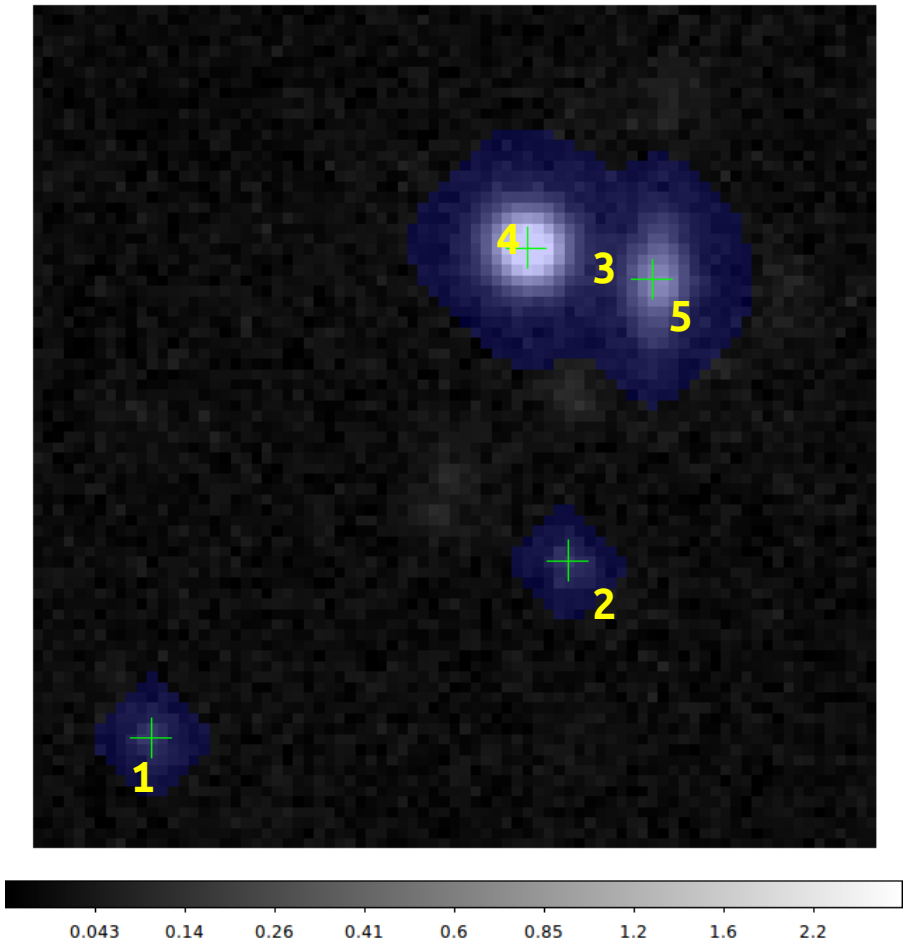


id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	0		regular

Detect, ***Deblend***, Measure



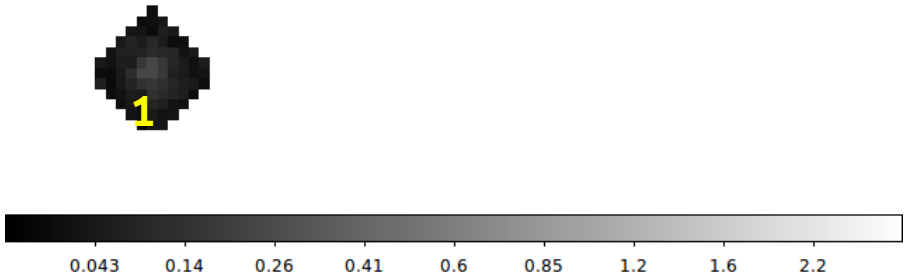
Detect, Deblend, Measure



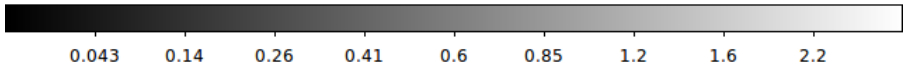
id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

Detect, Deblend, Measure

id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

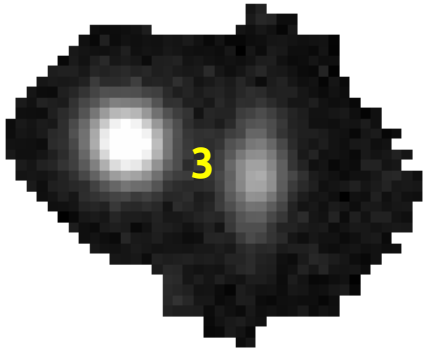


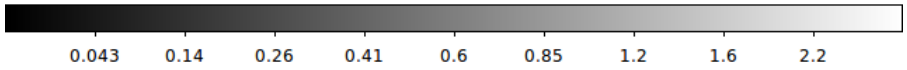
Detect, Deblend, Measure



id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

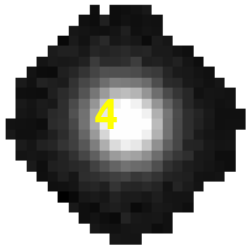
Detect, Deblend, Measure

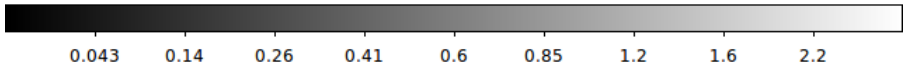




id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

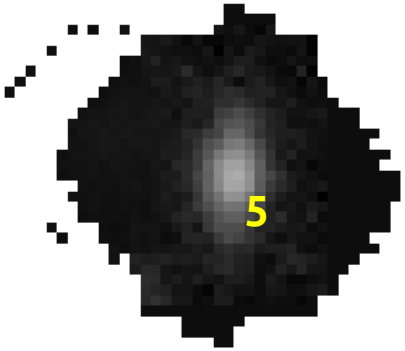
Detect, Deblend, Measure

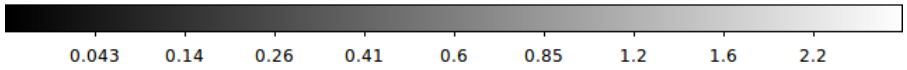




id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

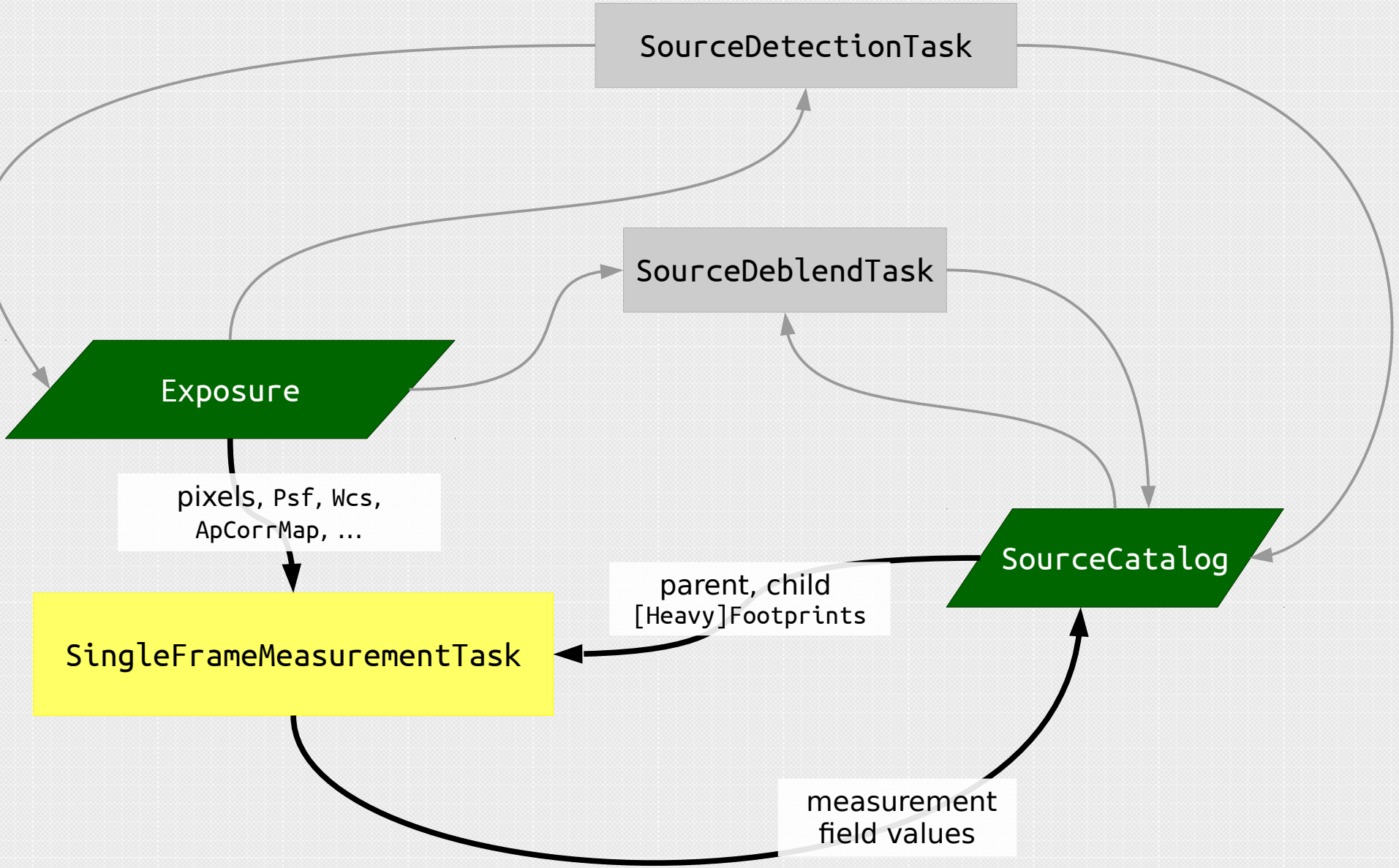
Detect, Deblend, Measure



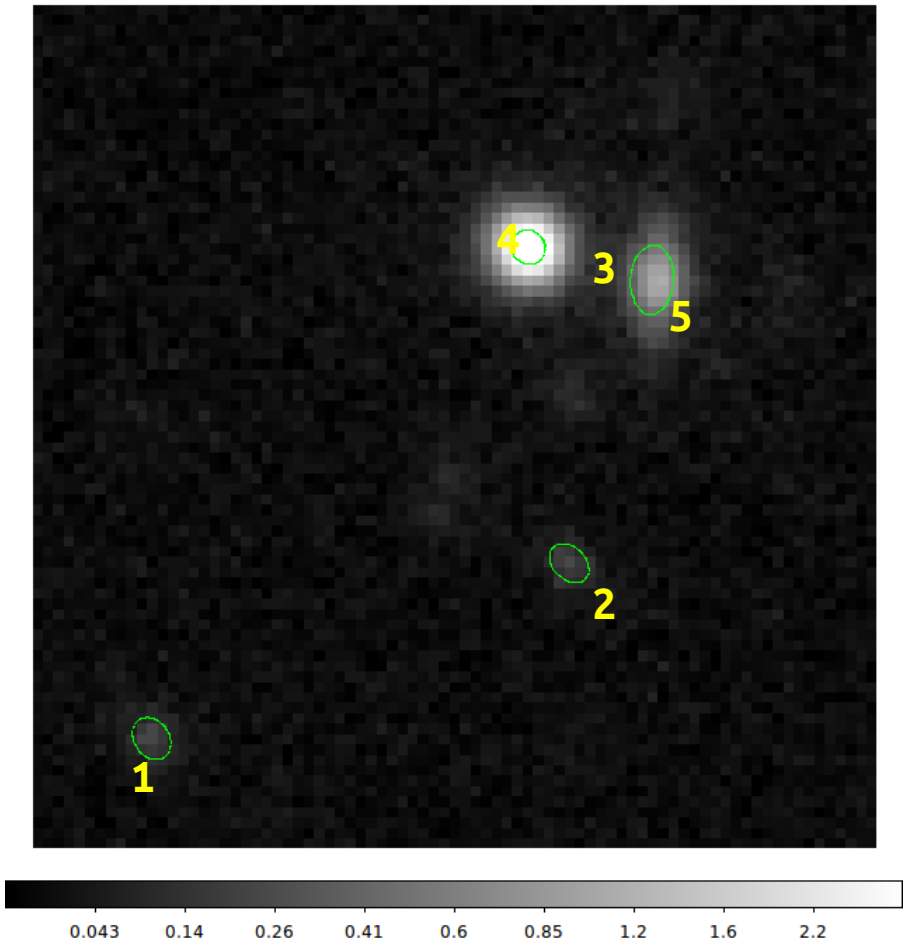


id	parent	nchild	(measurements)	(footprint)
1	0	0		regular
2	0	0		regular
3	0	2		regular
4	3	0		heavy
5	3	0		heavy

Detect, Deblend, **Measure**

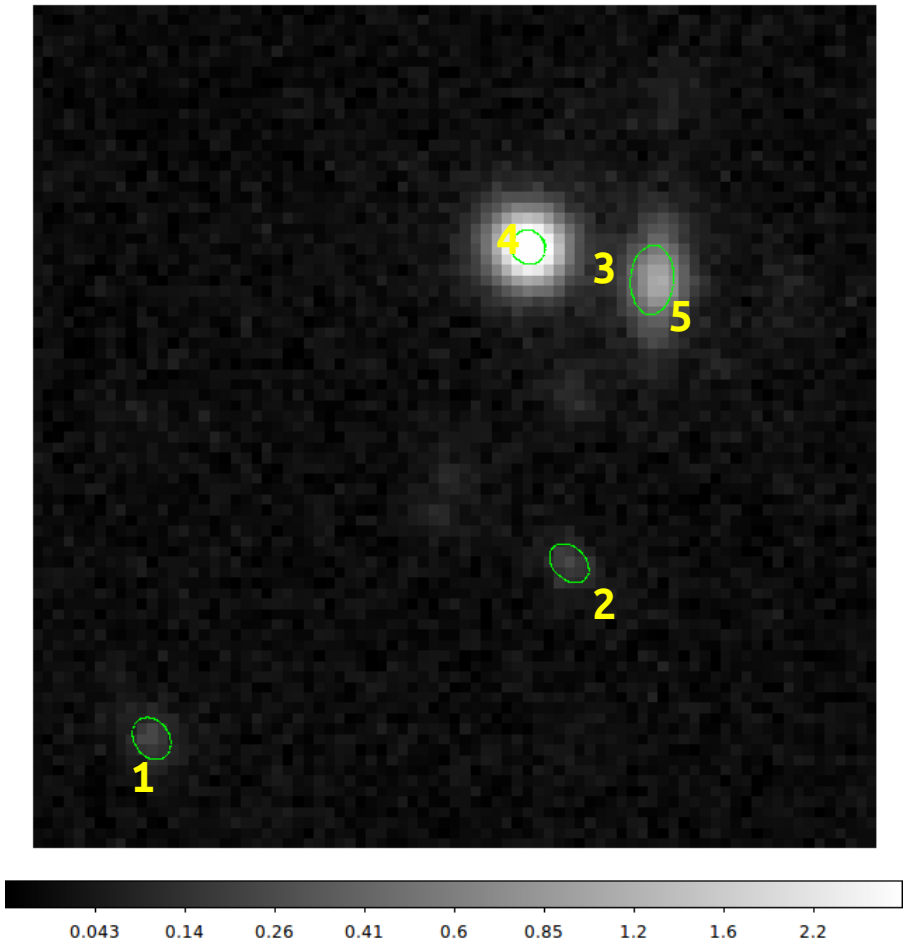


Detect, Deblend, ***Measure***



id	parent	nchild	(measurements)	(footprint)
1	0	0	(filled)	regular
2	0	0	(filled)	regular
3	0	2	(filled)	regular
4	3	0	(filled)	heavy
5	3	0	(filled)	heavy

Inside Measurement

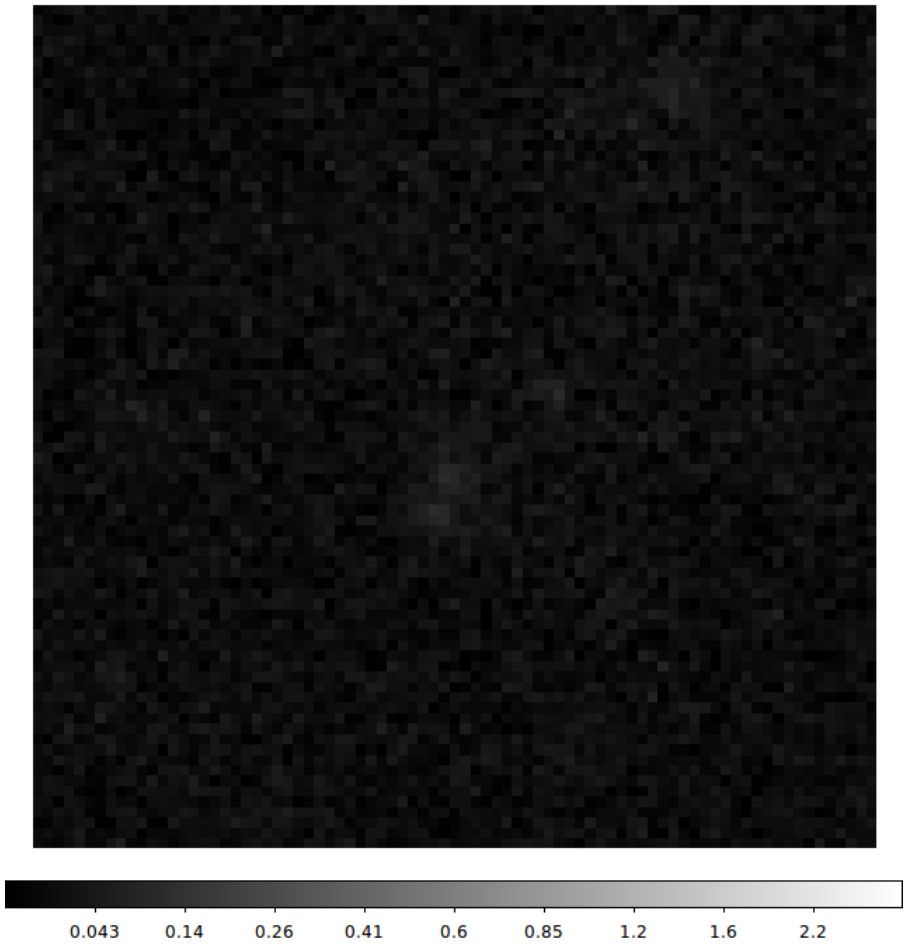



```
replace all detections with noise

for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Inside Measurement

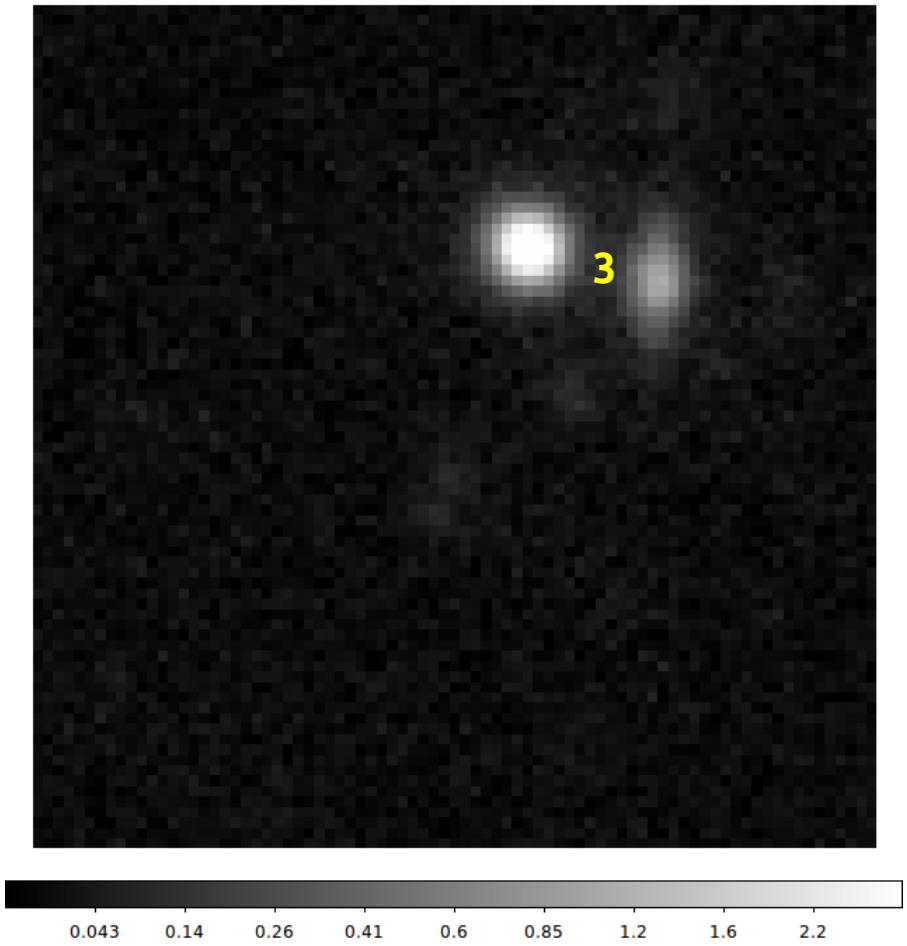


replace all detections with noise

```
for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Inside Measurement

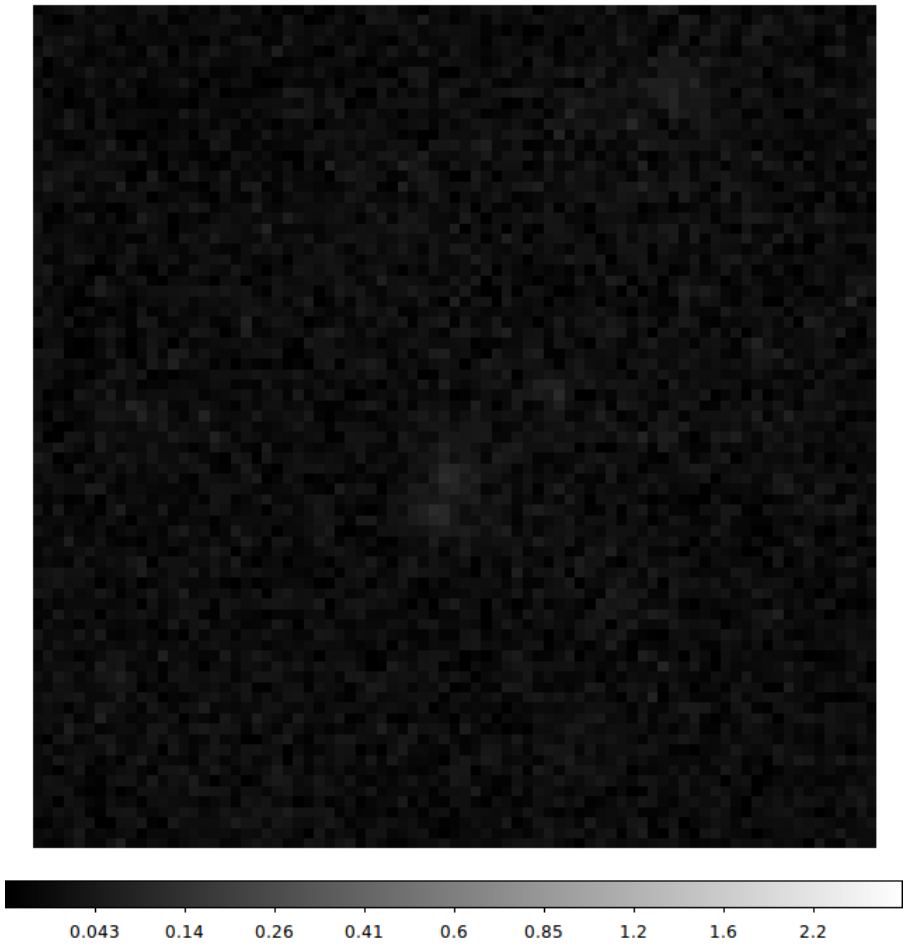



```
replace all detections with noise

for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Inside Measurement

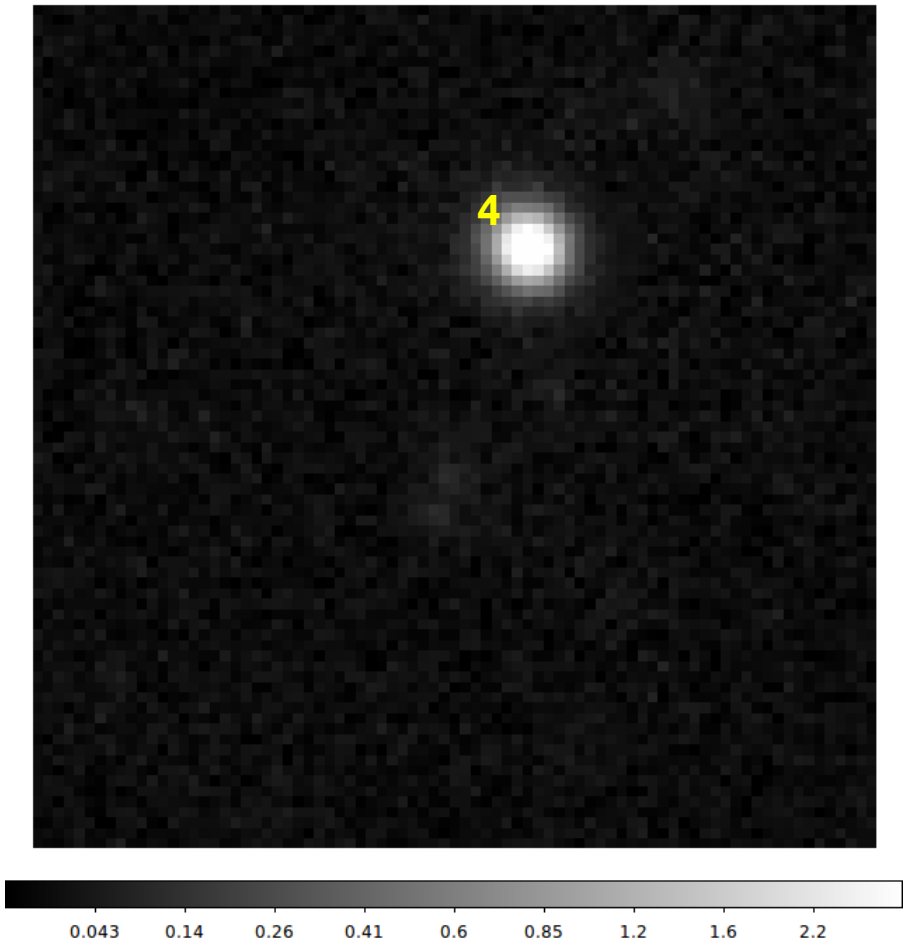



```
replace all detections with noise

for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Inside Measurement

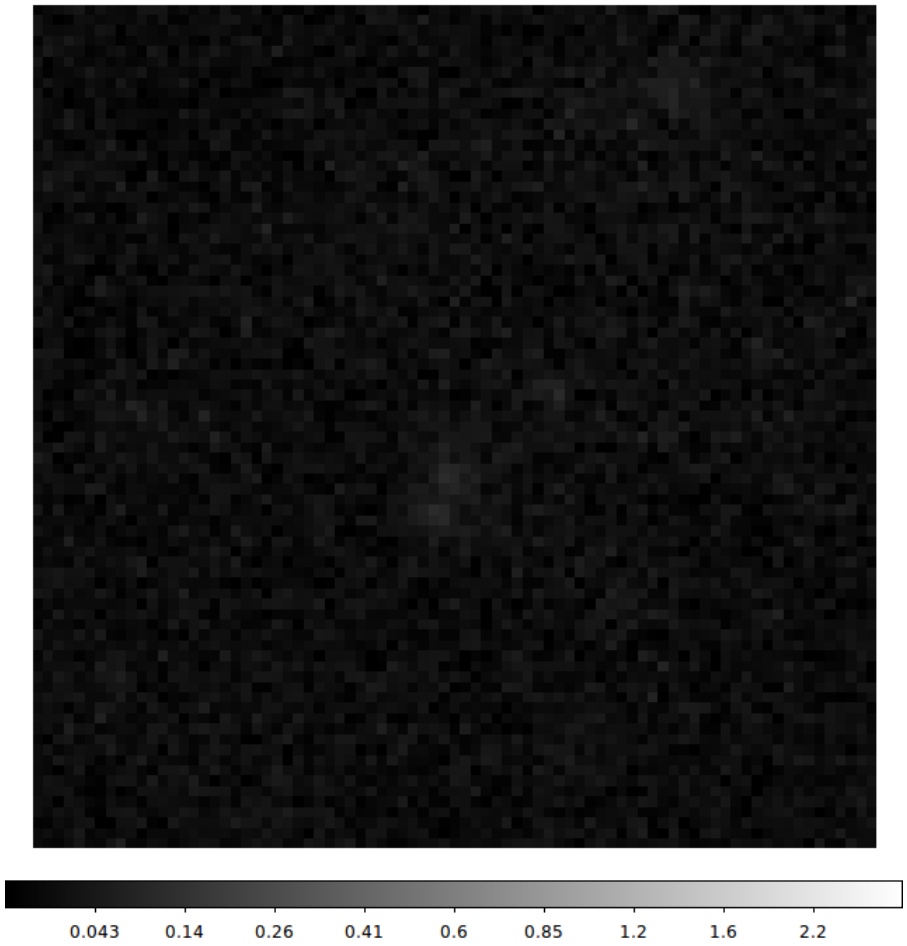



```
replace all detections with noise

for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Inside Measurement




```
replace all detections with noise

for record in catalog:
   restore pixels from HeavyFootprint
   run plugins
   re-replace pixels with noise

apply aperture corrections
```

Before Initializing `SingleFrameMeasurementTask`

```
# We have to initialize all tasks before using any of them:
# multiple tasks will write to the same Schema, and we can't create an output
# catalog until we've finished defining that Schema.

# Start with a minimal schema - only the fields all SourceCatalogs need
schema = lsst.afw.table.SourceTable.makeMinimalSchema()

# Customize the detection task a bit (optional)
detectConfig = lsst.meas.algorithms.SourceDetectionConfig()
detectConfig.returnOriginalFootprints = False   # should be the default
detectConfig.thresholdValue = 10                # only 10-sigma detections

# Create the detection task.  We pass the schema so the task can declare a few flag fields
detectTask = lsst.meas.algorithms.SourceDetectionTask(config=detectConfig, schema=schema)

# Create a task for deblending (optional, but almost always a good idea).
# Again, the task defines a few flag fields it will later fill.
deblendTask = lsst.meas.deblender.SourceDeblendTask(schema=schema)
```

Configuring SingleFrameMeasurementTask

```
measureConfig = lsst.meas.base.SingleFrameMeasurementConfig()

# Modify the set of active plugins ('.names' behaves like a Python set)
measureConfig.plugins.names.remove("base_GaussianCentroid")

# Enable some plugins - import the Python module first to make them available
measureConfig.plugins.names |= ["modelfit_ShapeletPsfApprox", "modelfit_CModel"]

# Change which plugin's output we "bless" as the "Model Flux"
measureConfig.slots.modelFlux = "modelfit_CModel"

# Modify the internal configuration of one of the plugins
measureConfig.plugins["base_ClassificationExtendedness"].fluxRatio = 0.985

# Actually create the Task.  This initializes all the plugins and that defines the
# rest of the schema
measureTask = lsst.meas.base.SingleFrameMeasurementTask(
    config=measureConfig,
    schema=schema
)
```

Running SingleFrameMeasurementTask

```
# Create a SourceTable from the Schema.  SourceTable is somewhat misleadingly named; it's
# really a factory for SourceRecords, not a container for them.
table = lsst.afw.table.SourceTable.make(schema)

# We pass the SourceTable and an Exposure to SourceDetectionTask.run(), and it'll return
# a SourceCatalog with empty records for the parents only.  Those will have Footprints
# attached to them.
detectResult = detectTask.run(table, exposure)
catalog = detectResult.sources

# We then pass the exposure and the catalog to the SourceDeblendTask, which adds new
# records for all the children, and attaches HeavyFootprints to them.
# Annoyingly, we have to pass the psf separately (DM-3987)
deblendTask.run(exposure, catalog, psf=exposure.getPsf())

# Finally, we can now run measurement.  Note the transposed argument order :/
measureTask.run(catalog, exposure)
```

Writing a `SingleFramePlugin` (in Python)

```
class BoxFluxConfig(lsst.meas.base.SingleFramePluginConfig):
    <new code here>

@lsst.meas.base.register("ext_BoxFlux")
class BoxFluxPlugin(lsst.meas.base.SingleFramePlugin):

    ConfigClass = BoxFluxConfig

    @classmethod
    def getExecutionOrder(cls):
        return cls.FLUX_ORDER

    def __init__(self, name, schema, metadata):
        lsst.meas.base.SingleFramePlugin.__init__(self, name, schema, metadata)
        <new code here>

    def measure(self, measRecord, exposure):
        <new code here>
```

Writing a `SingleFramePlugin` (in Python)

```
class BoxFluxConfig(lsst.meas.base.SingleFramePluginConfig):

    width = lsst.pex.config.Field(
        dtype=float, default=50,
        doc="approximate width of rectangular aperture"
    )

    height = lsst.pex.config.Field(
        dtype=float, default=50,
        doc="approximate height of rectangular aperture"
    )
```

Writing a `SingleFramePlugin` (in Python)

```
def measure(self, measRecord, exposure):

    centroid = <get a previously-measured centroid from measRecord>

    # Create a single-pixel box
    point = lsst.afw.geom.Point2I(centroid)
    box = lsst.afw.geom.Box2I(point, point)

    # Grow the box to the desired size
    box.grow(lsst.afw.geom.Extent2I(self.config.width//2, self.config.height//2))

    # Horrible syntax to create a subimage.  Can't use [] because it doesn't pay
    # attention to xy0 :-(
    subMaskedImage = exposure.getMaskedImage().Factory(
        exposure.getMaskedImage(),
        box,
        lsst.afw.image.PARENT
    )

    # compute the flux by extracting and summing NumPy arrays.
    flux = subMaskedImage.getImage().getArray().sum()
    fluxSigma = subMaskedImage.getVariance().getArray().sum()**0.5

    <stuff the results into measRecord>
```

Writing a `SingleFramePlugin` (in Python)

```
def __init__(self, config, name, schema, metadata):
    lsst.meas.base.SingleFramePlugin.__init__(self, config, name, schema, metadata)

    # Get a FunctorKey that can quickly look up the "blessed" centroid value.
    self.centroidKey = lsst.afw.table.Point2DKey(schema["slot_Centroid"])

    # Add some fields for our outputs, and save their Keys.
    doc = "flux in a {0.width} x {0.height} rectangle".format(self.config)
    self.fluxKey = schema.addField(
        schema.join(name, "flux"), type=float, units="dn", doc=doc
    )
    self.fluxSigmaKey = schema.addField(
        schema.joint(name, "fluxSigma"), type=float, units="dn",
        doc="1-sigma uncertainty for BoxFlux"
    )
```

Writing a `SingleFramePlugin` (in Python)

```
def measure(self, measRecord, exposure):

    centroid = measRecord.get(self.centroidKey)

    # Create a sing                    x
    point
    box =

    # Grow the box to the desired size
    box.grow(lsst.afw.geom.Extent2I(self.config.width//2, self.config.height//2))

    # Horrible syntax to create a subimage.  Can't use [] because it doesn't pay
    # attention to xy0 :-(
    subMaskedImage = exposure.getMaskedImage().Factory(
        exposure.getMaskedImage(),
        box,
        lsst.afw.image.PARENT
    )

    # compute the flux by extracting and summing NumPy arrays.
    flux = subMaskedImage.getImage().getArray().sum()
    fluxSigma = subMaskedImage.getVariance().getArray().sum()**0.5

    measRecord[self.fluxKey] = flux
    measRecord[self.fluxSigmaKey] = fluxSigma
```

why not `measRecord[self.centroidKey]`?

Error Handling in Plugins

If we try running the plugin as it stands now, we get:

```
lsst::pex::exceptions::LengthError: 'Box2I(Point2I(-14,-15),Extent2I(51,51))
doesn't fit in image 81x81'
...
NotImplementedError: The algorithm 'BoxFluxPlugin' thinks it cannot fail, but it
did; please report this as a bug (the full traceback is above).
```

That's because we haven't overridden `fail()`, and the default implementation assumes the algorithm is infallible.

- If it's a misconfiguration or something else that will cause every measurement to fail on every single source, raise **`lsst.meas.base.FatalAlgorithmError`**.
- If it's a known failure mode, the plugin should set at least two flags: a general failure flag for the plugin, and a specific flag indicating what went wrong. That can be done in two ways:
  - Just set the flags in **`measure()`**.
  - Re-raise as **`lsst.meas.base.MeasurementError`**, and set flags in **`fail()`**.
- All other exceptions will trigger warnings, and **`fail()`** will be called to set the general failure flag.

Error Handling in Plugins

```
def __init__(self, config, name, schema, metadata):
    lsst.meas.base.SingleFramePlugin.__init__(self, config, name, schema, metadata)

    # Get a FunctorKey that can quickly look up the "blessed" centroid value.
    self.centroidKey = lsst.afw.table.Point2DKey(schema["slot_Centroid"])

    # Add some fields for our outputs, and save their Keys.
    doc = "flux in a {0.width} x {0.height} rectangle".format(self.config)
    self.fluxKey = schema.addField(
        schema.join(name, "flux"), type=float, units="dn", doc=doc
    )
    self.fluxSigmaKey = schema.addField(
        schema.joint(name, "fluxSigma"), type=float, units="dn",
        doc="1-sigma uncertainty for BoxFlux"
    )
    self.flagKey = schema.addField(
        schema.join(name, "flag"), type="Flag",
        doc="general failure flag for BoxFlux"
    )
    self.edgeFlagKey = schema.addField(
        schema.join(name, "flag", "edge"), type="Flag",
        doc="flag set when rectangle used by BoxFlux doesn't fit in the image"
    )
```

Error Handling in Plugins

```
@lsst.meas.base.register("ext_BoxFlux")
class BoxFluxPlugin(lsst.meas.base.SinglePlugin):

    ConfigClass = BoxFluxConfig

    FAILURE_EDGE = 1

    @classmethod
    def getExecutionOrder(cls):
        return cls.FLUX_ORDER

    def __init__(self, confi
        ...

    def measure(self, measRec
        ...

    def fail(self, measRecord, error=None):
        measRecord.set(self.flagKey, True)
        if error is not None:
            assert error.getFlagBit() == self.FAILURE_EDGE
            measRecord.set(self.edgeFlagKey, True)
```

`error` is guaranteed to be either an instance of `MeasurementError` or None

Error Handling in Plugins

If we try running the plugin again, we get a warning:

```
measurement WARNING: Error in ext_BoxFlux.measure on record 1:
...
lsst::pex::exceptions::LengthError: 'Box2I(Point2I(-14,-15),Extent2I(51,51))
doesn't fit in image 81x81'
```

That's because we're still throwing a `LengthError` in `measure()`, and that means we're not setting our new edge flag.

Error Handling in Plugins

```
def measure(self, measRecord, exposure):

    ...

    # Horrible syntax to create a subimage.  Can't use [] because it doesn't pay
    # attention to xy0 :-(
    try:
        subMaskedImage = exposure.getMaskedImage().Factory(
            exposure.getMaskedImage(),
            box,
            lsst.afw.image.PARENT
        )
    except lsst.pex.exceptions.LengthError as err:
        raise lsst.meas.base.MeasurementError(err, self.FAILURE_EDGE)

    # compute the flux by extracting and summing NumPy arrays.
    flux = subMaskedImage.getImage().getArray().sum()
    fluxSigma = subMaskedImage.getVariance().getArray().sum()**0.5

    measRecord[self.fluxKey] = flux
    measRecord[self.fluxSigmaKey] = fluxSigma
```

We've been doing things the hard way. There are a number of utility classes in lsst.meas.base to make things easier:

- `FlagHandler` will manage a set of flag keys for different error conditions, and implement `fail()` for you.
- `SafeCentroidExtractor` and `SafeShapeExtractor` get centroid and shape values from previous measurements while handling flags appropriately.
- `FluxResult[Key]`, `CentroidResult[Key]`, and `ShapeResult[Key]` map simple structs to records and make it easier to create output fields for common types of measurements.

Plugins can also be written in C++, using the `Algorithm` base class. You can also use `SimpleAlgorithm` to implement both a `SingleFramePlugin` and a `ForcedPlugin` at the same time (something we haven't provided an easy way to do in Python).

See `PsfFluxAlgorithm` for an annotated simple example, and `SdssShapeAlgorithm` for a more complex one.