Developing End User Tools for DKIST

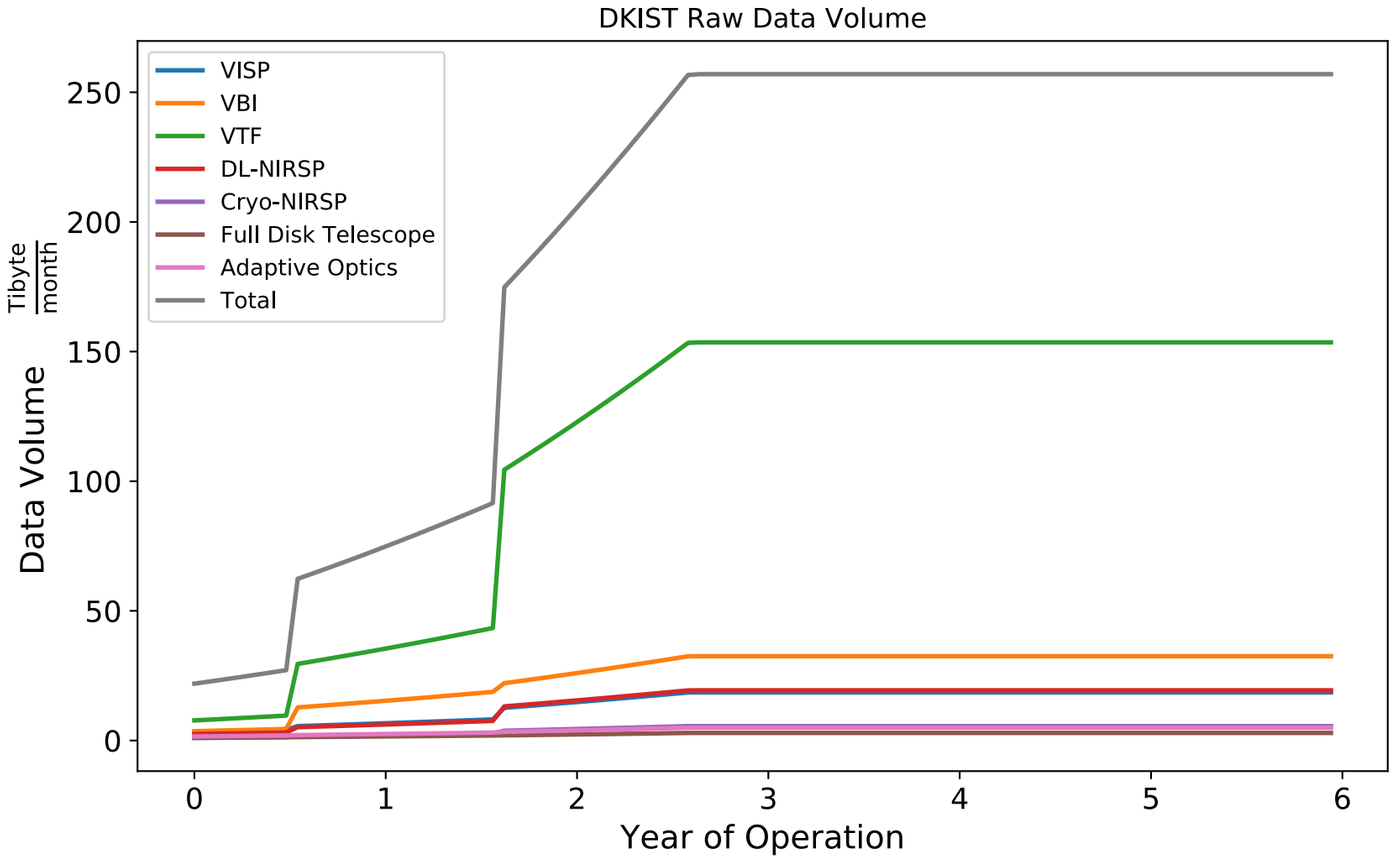
Stuart Mumford

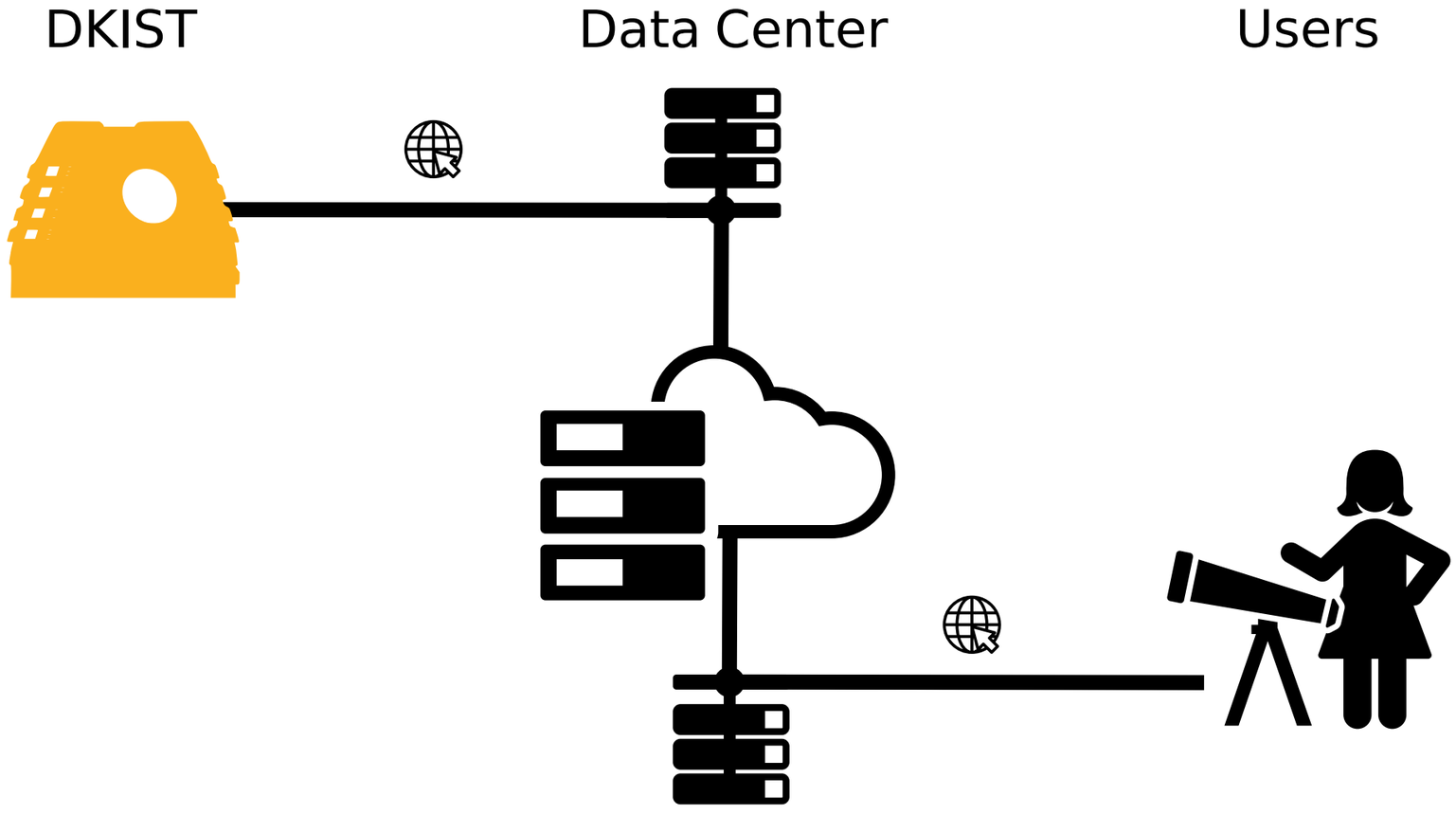
(http://nso.edu) (http://shef.ac.uk)
(http://dkist.nso.edu)

The Daniel K. Inouye Solar Telescope (DKIST)

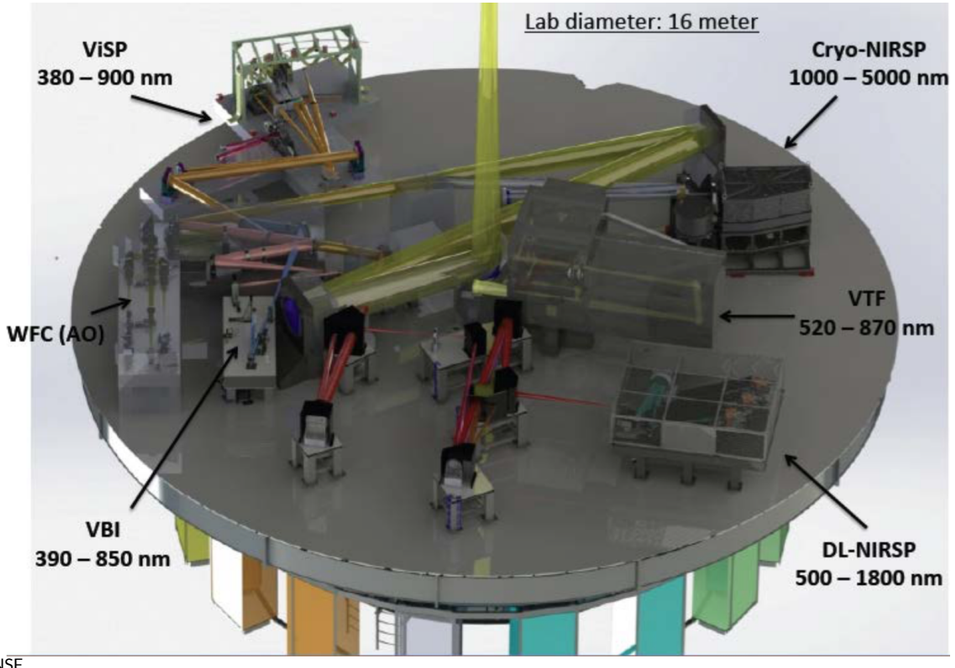
- 4m Mirror.
- ≈ 12 kW of power on the primary mirror.
- 5 First light instruments.
- Proposal based observations, run in service mode like a satellite.
- Operational early 2020.

3 PB raw data per year




DKIST
Data Center
Users

Physical Data Types



Five disparate instruments with many modes.

All give combinations of:

stokes, wavelength, time, space, space

All the calibrated data can be represented as dense hyper-cubes.

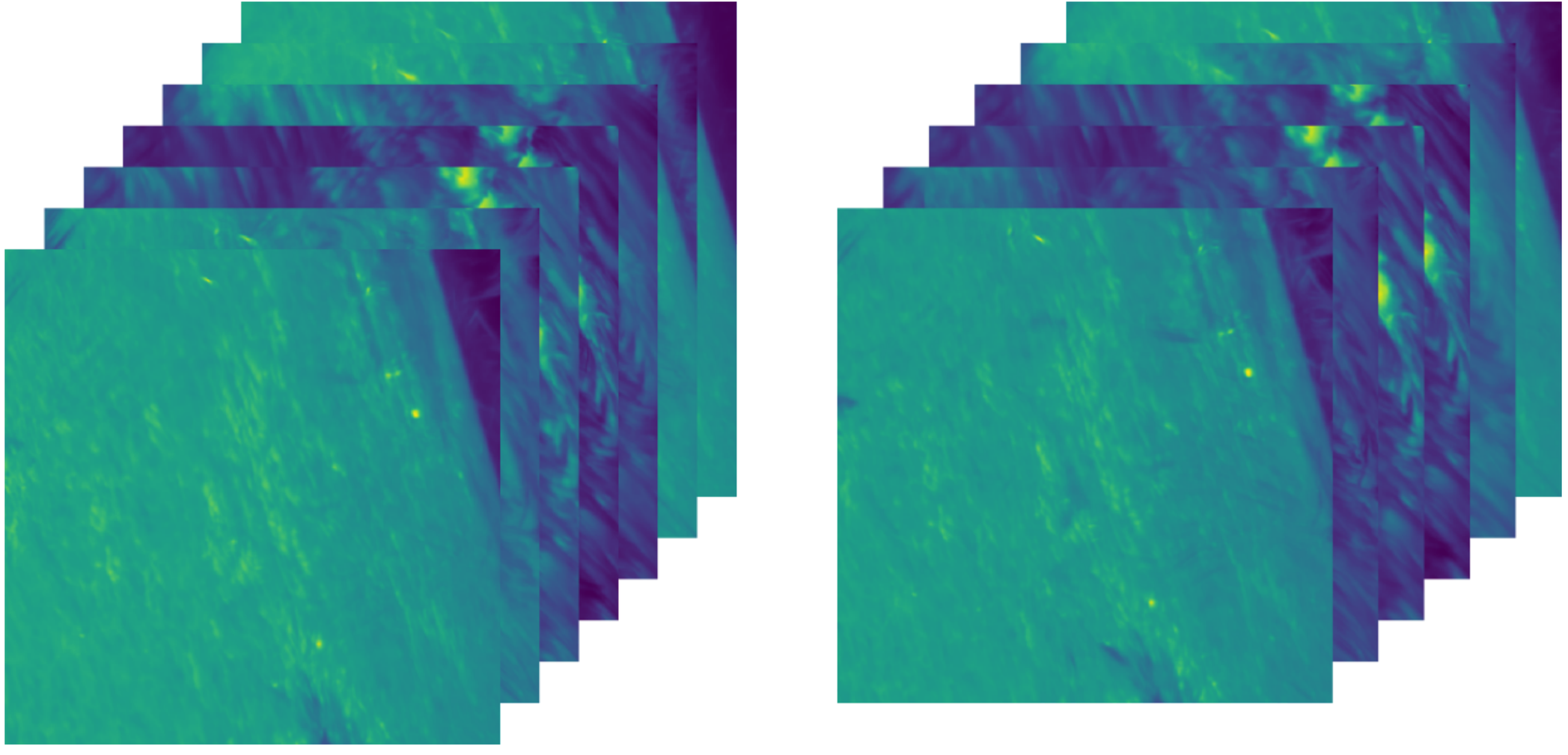
Calibrated Data Challenges

The goal of the user tools is to provide an easy to use Pythonic interface to the calibrated data products.

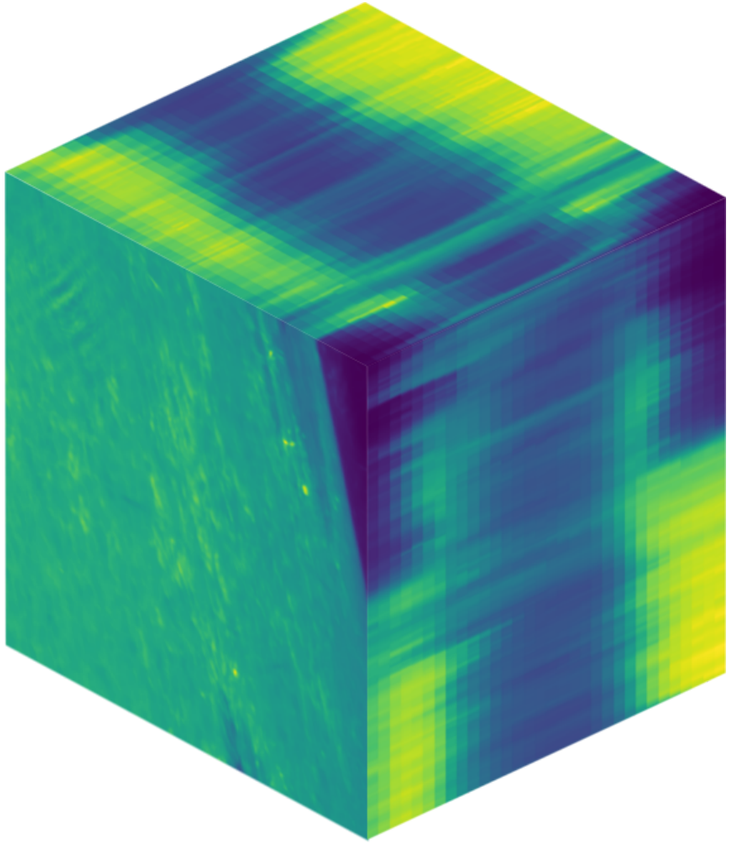
- **No post-processing in the data centre**: What is stored is what you download.

- To give maximum flexibility data are stored as individual "calibrated frames".

make this...



look like this:



In 5 Dimensions

Minimize Open Files!

Dask

```
import dask.array as da

class DelayedIO:
    def __init__(self, filename):
        self.filename = filename
        self.shape = (4096, 4096)
        self.dtype = np.int16

    def __getitem__(self, slc):
        with fits.open(self.filename, memmap=True) as hdul:
            hdul.verify('fix')
            return hdul[1].data[slc]
```

```
arr = da.stack([da.from_array(DelayedIO(f), chunks=(4096, 4096)) for f in filenames])
arr
```

```
dask.array<stack, shape=(10, 4096, 4096), dtype=int16, chunksize=(1, 4096, 4096)>
```

```
arr *= 5
arr
```

```
dask.array<mul, shape=(10, 4096, 4096), dtype=int16, chunksize=(1, 4096, 4096)>
```

```
arr = arr[1:3,500:510, 500:510]
arr
```

```
dask.array<getitem, shape=(2, 10, 10), dtype=int16, chunksize=(1, 10, 10)>
```

```
arr.compute()
```

```
array([[[  0,   0,   5,   5,   5, -10,   5,  -5,   0,  -5],
        [  5,   5,   0,   0,   5,   0,   5,   0,   0,  -5],
        [  0,   5,  -5,   5,  10,   5,   5,   5,   0,   0],
        [  0,  -5,   5,   5,   0,  -5,   0,   5,   0,   0],
        [  5,   5,   0,   5,   0,   0,  -5,   0,   0,  -5],
        [  0,  10,   0,   5,  -5,   5,   5,   0,  10,   0],
        [  0,   0,   0,  -5,   5,   0,  10,   5,   5,   5],
        [  5,  -5, -10,   5,   0,  -5,   5,   0,   0,   0],
        [ -5,   5,  -5,  -5,   0,   0,   5,   0,   0,   0],
        [  0,   5,   0,   5,   5,   0,   5,   0,  -5,  -5]],

       [[ 10,   0,  -5,  -5,   0,   5,   0,  -5,   0, -10],
        [ -5,  -5,   0,  -5,   5, -10,  15,   5,   0,   0],
        [  5,   0,   0,  10,  -5,  -5, -10,   0,   5,   5],
        [ -5,  10,   0,   0,  -5,   0,  10, -10,   0,  10],
        [ 10,  15,  10, -10,  -5,   0,   5,  10,   5,   0],
        [-10, -10,   0,   0,  -5, -10,   5,   5,   5,   5],
        [ -5,   0,   5, -10,   5,  -5,   0,   5,  -5,   5],
        [ 10,   5,   0,   5,   0,   0,  10, -10,   0,  -5],
        [  5,   5,   0,  -5,   0,  10,  -5,   0,   5,   0],
        [ -5,   5,  10,  -5,  -5,   0,   5,  -5,   5,   5]]], dtype=int16)
```

Metadata!

Need to be able to construct a Dask array and a WCS for all the files without having to open and process all the headers.

asdf to the rescue

An array of references to arrays in FITS HDUs:

```
- !core/externalarray-1.0.0
      datatype: int16
      fileuri: aia.lev1_euv_12s.2017-09-06T120010Z.131.image_lev1.fits
      shape: [4096, 4096]
      target: 1
```

and a gWCS object.

Recap: Calibrated Data Products

- Each "frame" in a FITS file.
- A "dataset" comprises of many FITS files.
- An asdf file describes a dataset:
  - How to construct a Dask array.
  - a gWCS object for coordinate information.
- The array and the WCS can both be constructed without ever opening a FITS file.

Python Interface: NDCube

Provide an interface to these datasets which provides:

- Simple construction.
- Slicing of both data and WCS together.
- World <> Pixel coordinate conversions.
- Visualisation helpers.

```
from dkist.dataset import Dataset

dset = Dataset.from_directory("/home/stuart/sunpy/data/jsocflare/")
```

```
dset.data
```

```
dask.array<reshape, shape=(7, 11, 4096, 4096), dtype=int16, chunksize=(1, 11, 4096, 4096)>
```

```
dset.pixel_to_world(*(0,0,0,0,0)*u.pixel)
```

```
(<Quantity 211.0 Angstrom>,
 <Time object: scale='utc' format='isot' value=2017-09-06T11:59:59.120>,
 <SkyCoord (Helioprojective: obstime=2017-09-06T11:59:57.630, rsun=696000000.0 m, observer=<HeliographicStonyhurst Co
ordinate (obstime=2017-09-06T11:59:57.630): (lon, lat, radius) in (deg, deg, m)
    ( 0.,  7.234445,   1.50782400e+11)>): (Tx, Ty) in arcsec
    (-1221.65197224, -1229.40182539)>)
```

Future NDCube / NDData / WCS Development

To make this work in astropy core I would like to see:

- The "high-level" WCS object in APE 14 implemented.
- The addition of slicing support to both gWCS and APE 14 APIs.
- WCSAxes support for the APE 14 API.

Future User Tools Features

- Search and Indexing of local and remote data.
- Downloading of some frames in a hyper cube as needed.
- Interactive exploration - Glue?
- Tutorials to teach people Python and DKIST together!

Thank You