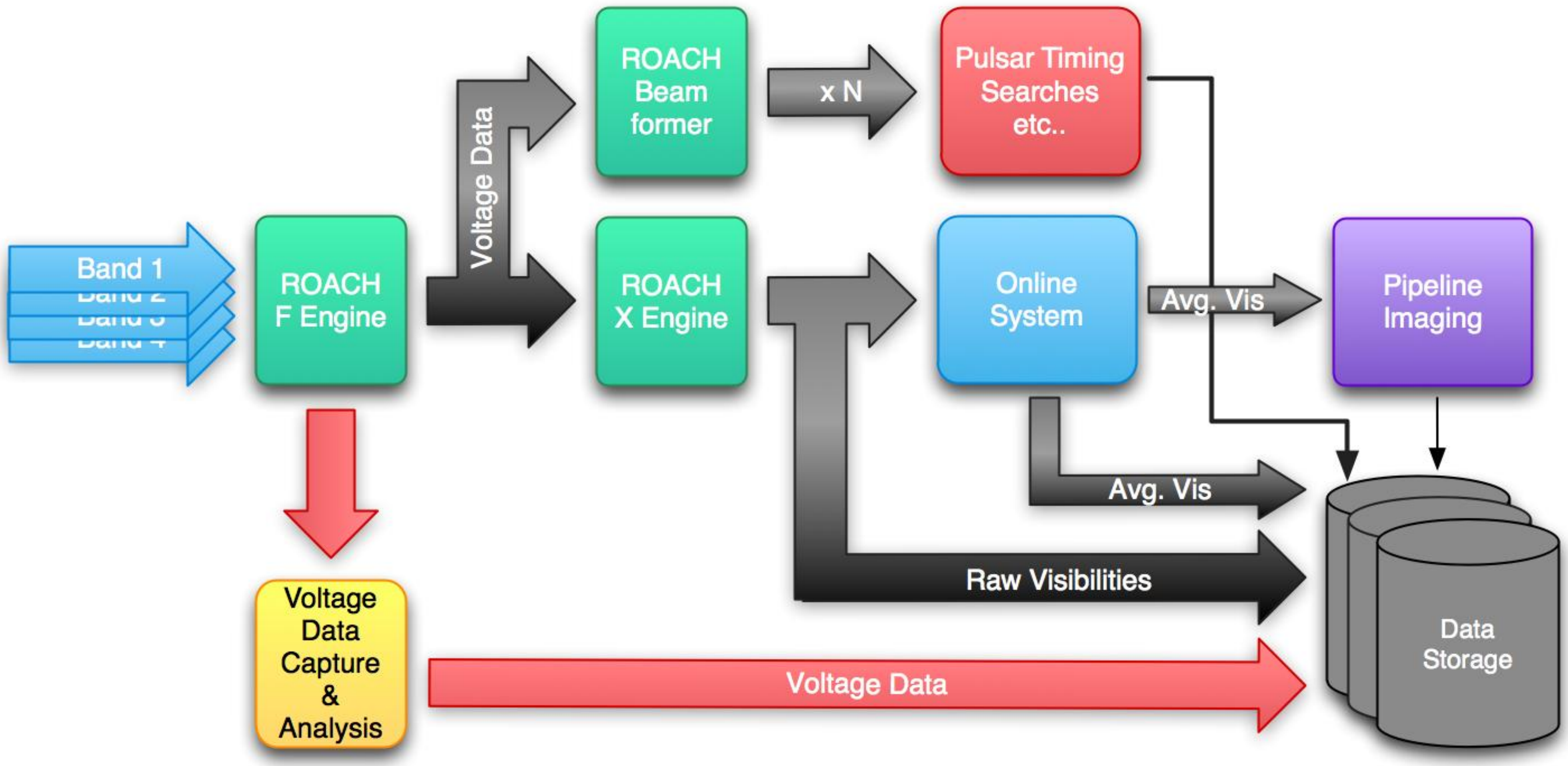
CALIM 2010

MeerKAT Online Processing

Simon Ratcliffe

SKA SOUTH AFRICA
SQUARE KILOMETRE ARRAY

MeerKAT Signal Path



Online System

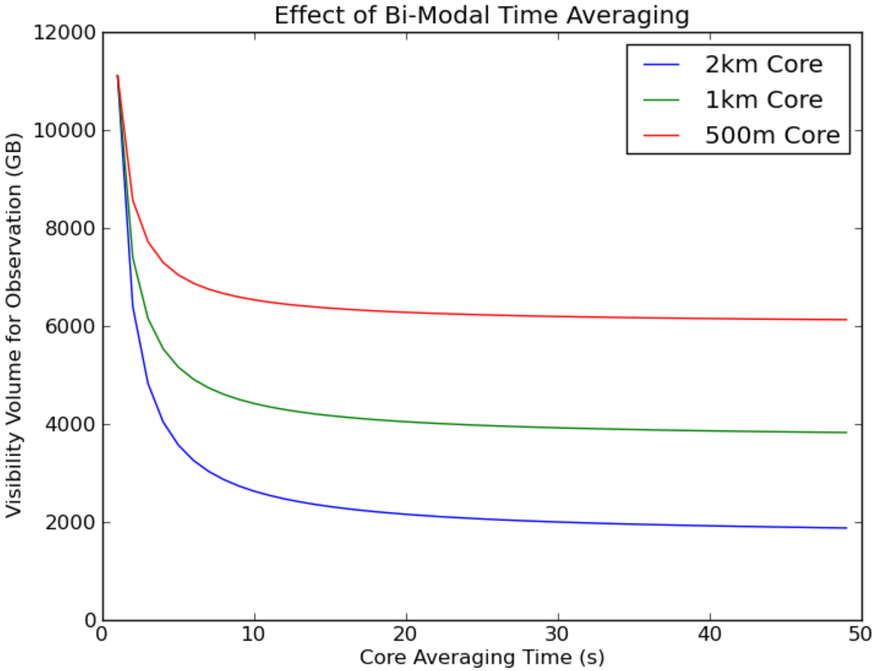
- The online system receives raw visibilities from the correlator at a sufficiently high dump rate to facilitate the following (currently 10 Hz):
  - Continuous Tsys calculation
  - RFI Flagging
  - Baseline dependent time averaging
- The resultant visiblities + cal data + flagging are written to disk in the medium term archive.
- A stream of output data is also produced for real-time downstream consumers, such as the quicklook imager and data spigot users.

Flagging

- On QA Return
  - Bad antenna / polarisation
  - Timing synch
  - Etc...
- RFI Flagging
  - Simple thresholding
  - Known sky pollutants (GEO, LEO, DME, etc...)
  - Recursive fading-memory polynomial filters (we have an IBM System-S implementation, porting to GPUs soon)

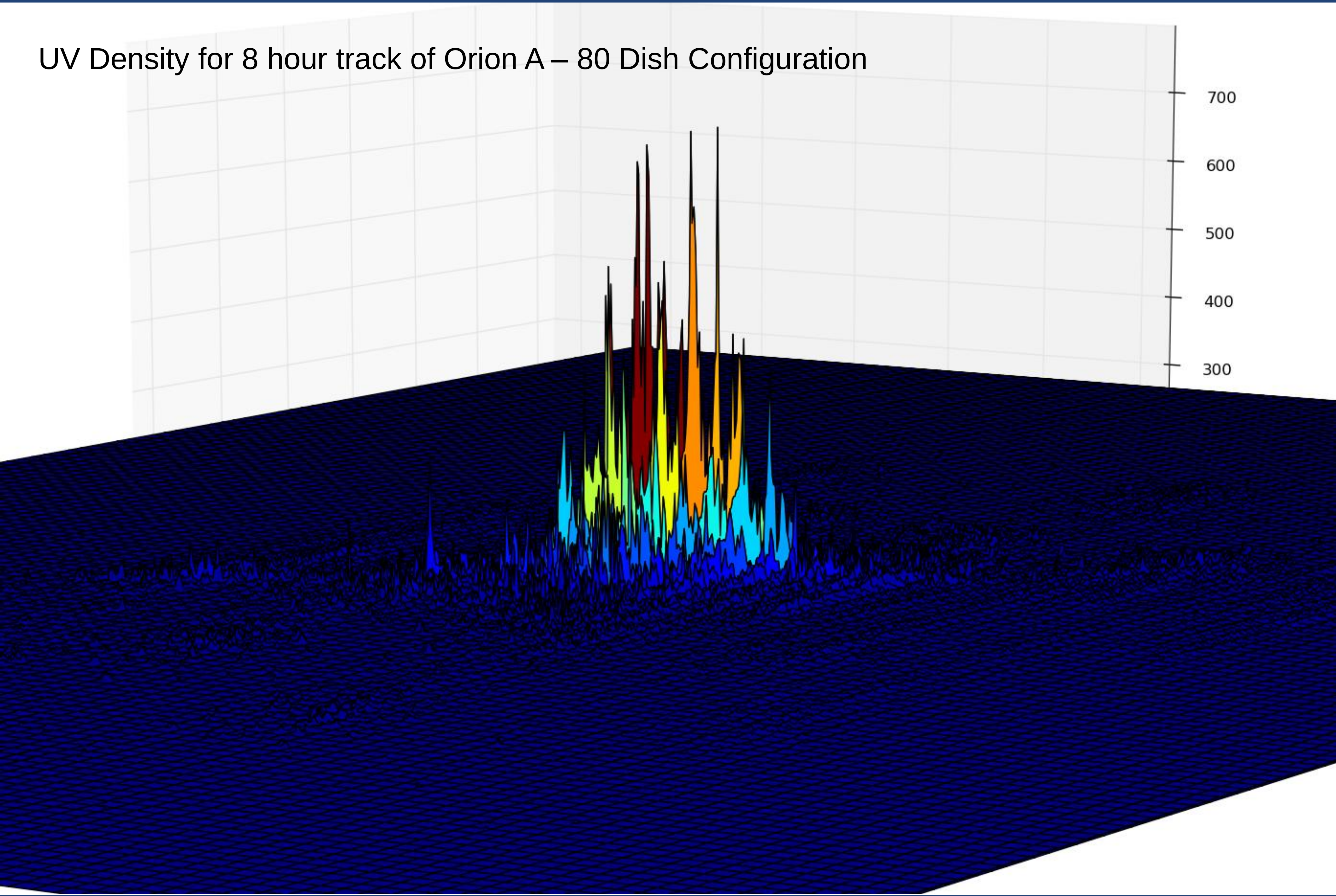
Baseline Dependent Time Averaging

- Simple bimodal averaging scheme can easily give a factor 4 reduction in overall data rates without any increase in time-average smearing.



- Hot spots can be preserved at higher dump rates to help with redundancy.

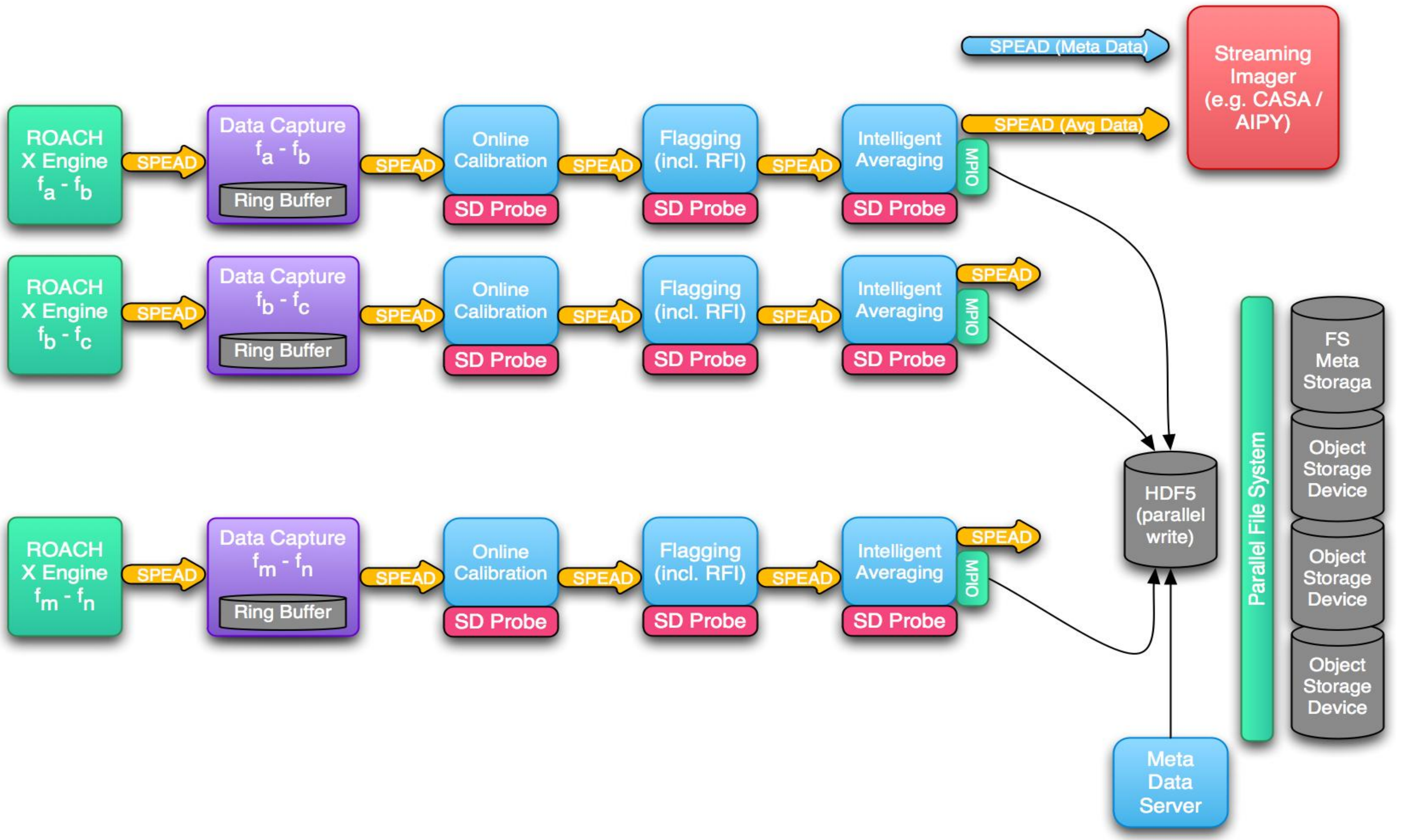
UV Density for 8 hour track of Orion A – 80 Dish Configuration



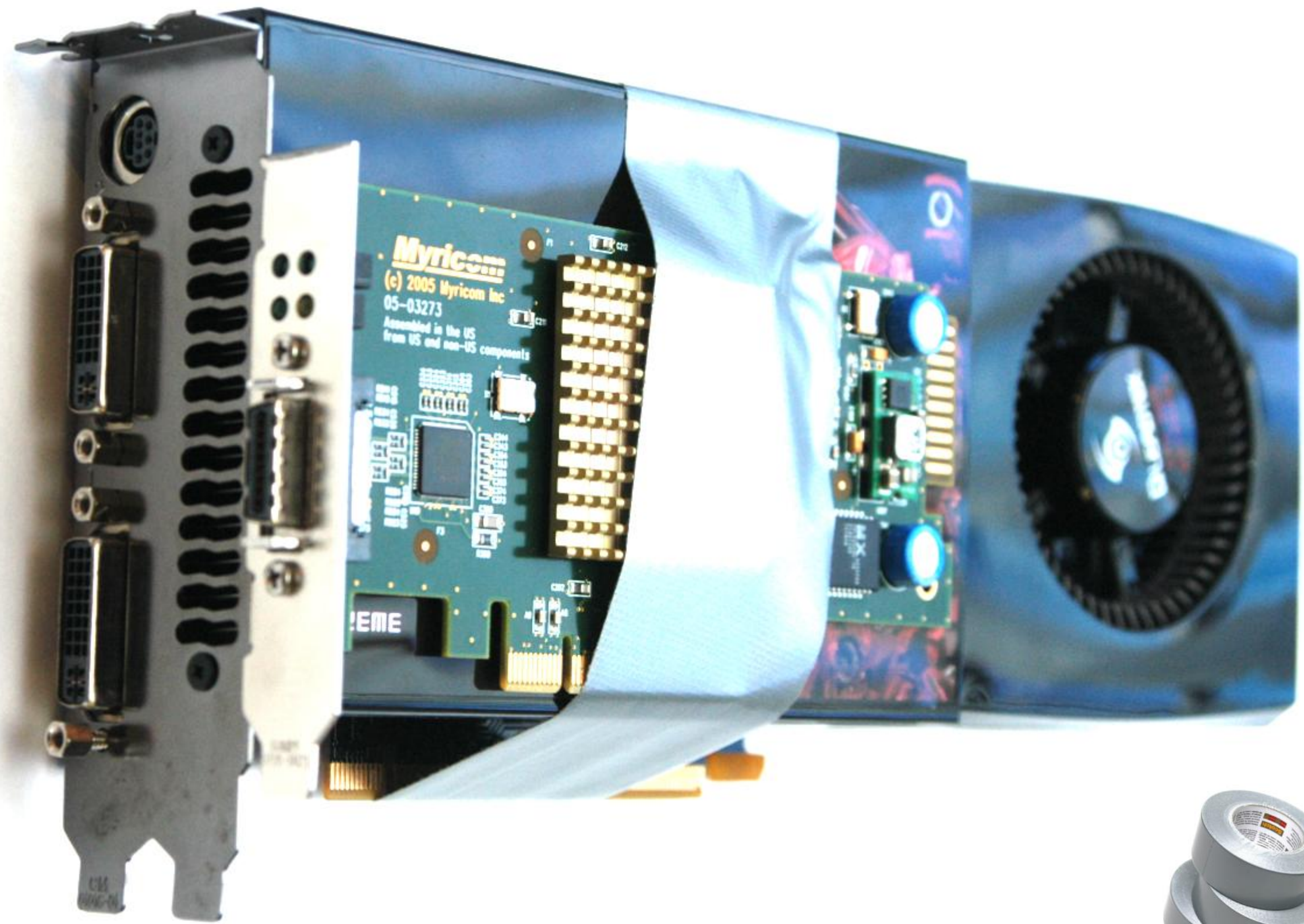
Online System - Detail

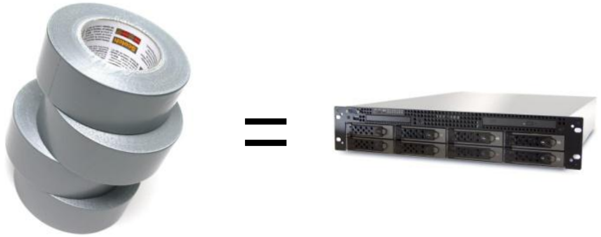
- Correlator output is split into a number of sub bands, each of which is processed in parallel.
- The split depends in the individual capacity of each element of the parallel system.
- With our current technology, 1024 channels can be processed in a single element (with 10 Hz correlator dump) – limited by 10 GbE throughput.
- Parallel HDF5 output file allows multiple simultaneous writes from each system element and provides an on disk corner turn.

Online System - Detail



The building block





Why GPUs

- A single GPU is a good match for current performance networking and has the I/O to keep up with data from both 10 GbE and QDR infiniband.
- Still riding good performance curves and HPC support is getting better all the time.
- Two major players (Nvidia and ATI) can help with vendor lock in (as long as you stick with OpenCL and not CUDA).
- The ecosystem of tools, particularly debuggers is rapidly improving (see Nvidia Nsight for an example of good these are getting)
- IRQ affinity under linux is helping to support multiple GPUs and NICs per machine.
- New developments (AMD Fusion almost a reality) will bring the GPU ever closer to the CPU memory bus, improving throughput and interoperability.

CUDA – Hello World

Kernel – Spawned and executed per GPU thread

```
__global__ void VecScale(float *A, const float B)
{
    int idx = (blockIdx.x * blockDim.x + threadIdx.x);
    A[idx] = A[idx] / B;
}
```

CPU code – C with CUDA markup

```
// Invoke kernel
int threadsPerBlock = tpb;
int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
VecScale<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
```

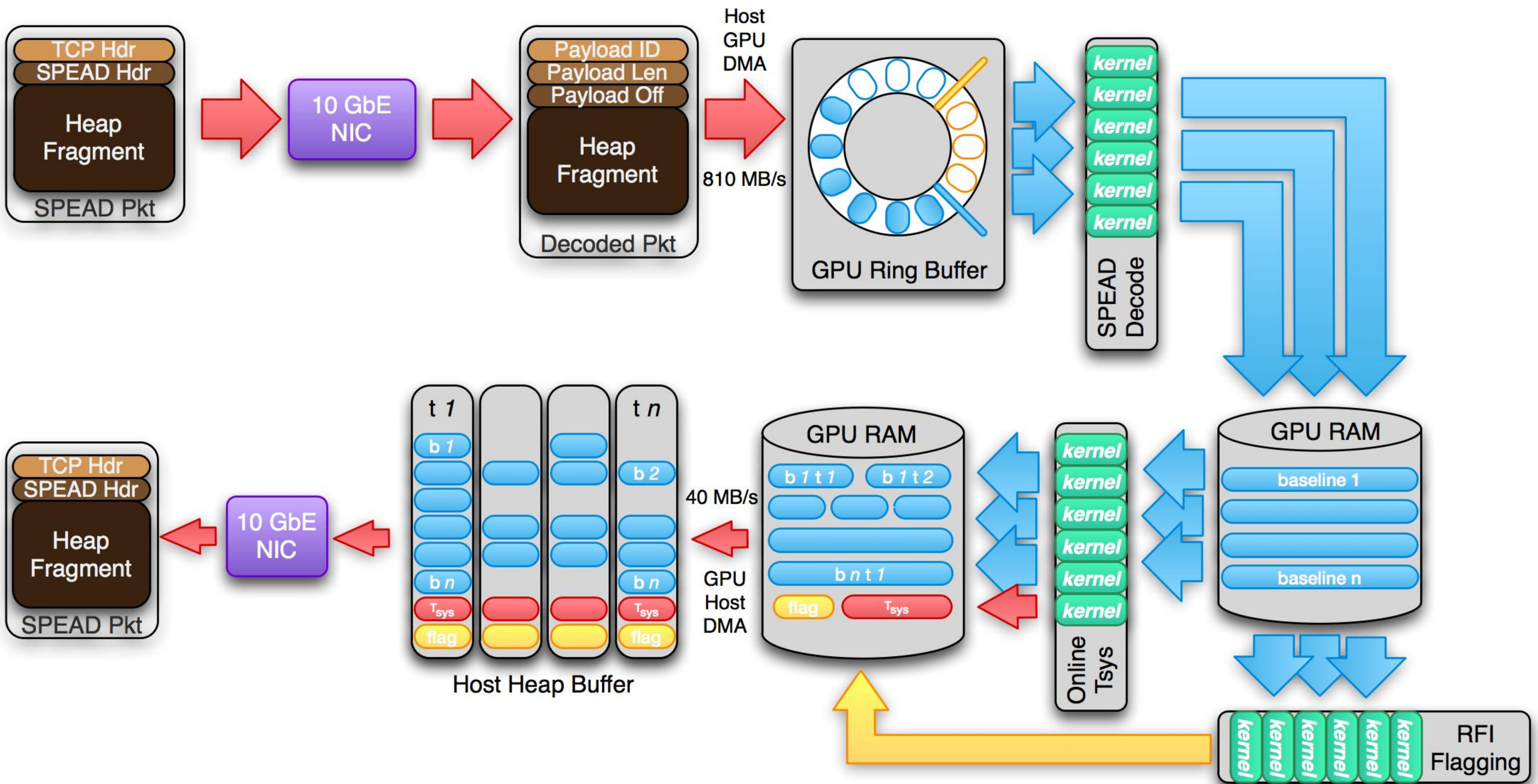
Python wrapper – Do things the easy way :)

```
import numpy as np, meerkapture as mk

mk.gpu_init()
test_vec = np.ones((512,512))
gpu_pointer = mk.gpu_vector_push(test_vec)
mk.gpu_vector_scale(test_vec, 0.5, 256)
return_vec = mk.gpu_vector_pull(test_vec)

print "Residual:", np.sum(test_vec/0.5 - return_vec)
```

Online System – On the GPU



Online System - Performance

- With modest current technology (Nvidia GTX 260, Core i7-940) we can fairly easily max out a 10 GbE port (around 8.6 Gbps).
- Decode of the streaming protocol can be done in CPU or GPU depending on first stage processing to be performed.
- MeerKAT online elements will leave around 3 GB of RAM and of order 2 Tflops processing power per block of channels in the GPU.

The Glue....

RUBBER INFUSED
SPEAD
SUPER GLUE-3
Max Flex Gel
FLEXIBLE
waterbestendig
résistante à l'eau
Alle materialen
Tous matériaux
Flex Gel Max
SECONDELIJM/COLLE INSTANTANÉE 10g
Henkel

SPEAD

CASPER

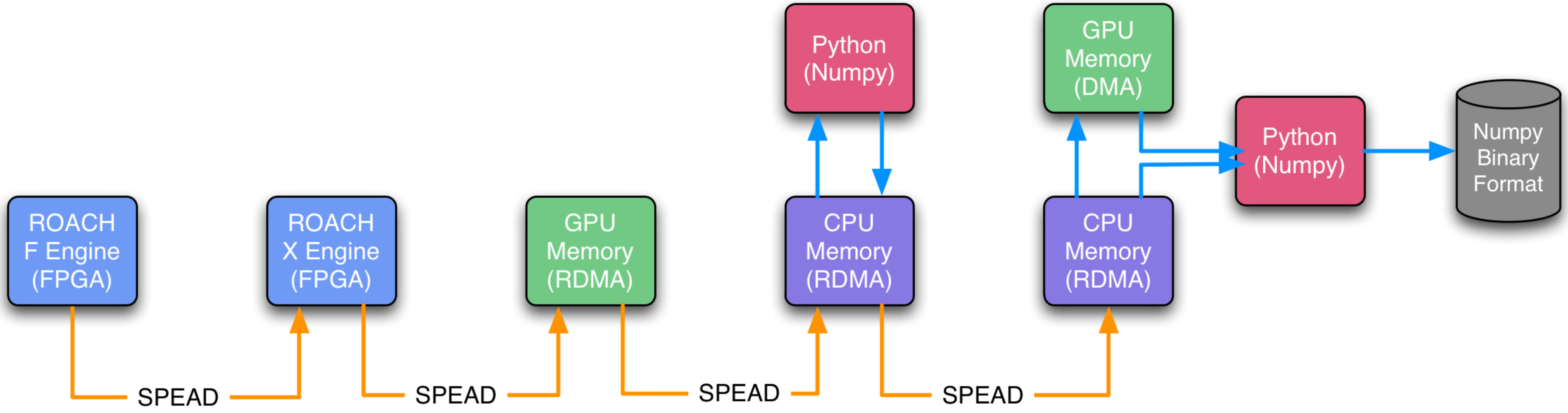
- Streaming Protocol for Exchanging Astronomical Data
- Joint development between SKA South Africa and UC Berkeley as part of the CASPER collaboration.
- Designed to handle a wide variety of astronomical data including voltage, visibility, and sensor data.
- Standard output data format for ROACH based correlators.
- Aim is to have a single coherent protocol throughout the entire processing chain (i.e. from digitisation to imaging)

SPEAD

- There are may formats out there, so why contribute to the malaise by developing another one ?
  - A number of formats pretend to be self describing but still require some a priori information (e.g VDIF)
  - We want to single protocol that bridges the gap between a number of disparate accelerator technologies.
  - We needed a very small number of mandatory headers to ease generation of a SPEAD stream by constrained devices (currently 4 words)
  - Self description extends through the receiver to present the user with an hierarchical, annotated data structure (e.g. numpy record array)
  - Soft Pythonic shell with crunchy C bits fits well with a number of emerging telescopes.

SPEAD

- Can be viewed as an accelerator transport protocol with very tight (and high performance) numpy integration.
- Infiniband style RDMA should be available within the next year or so, allowing userspace to be bypassed for remote copies.



SPEAD – How can it help us ?

- One of the main reasons for being I/O bound is that our problem is often not computationally dense enough.
- By having an accelerator in the I/O path we can use the spare FLOPS to provide value added services, such as on the fly decompression, to reduce the data rates and restore I/O – CPU balance.
- We can also easily slot in another element in the processing chain (be it CPU/GPU/FPGA) without changing our architecture.

SPEAD – How can it help us ?

- As a simple example of this, visibilities do pretty well with bzip:
  - Raw vis: 1120.9 MB (23913 x 512 x 12 x 8 byte)
  - HDF5: 1177.7 MB (incl sensor data)
  - Gzip HDF5: 726.5 MB
  - Bzip2 HDF5: 631.6 MB
- The decompression can be done on the fly by a GPU as the data is read from disk, either for further streaming or GPU/CPU processing.

SPEAD

- Specification is currently in revision L, update coming soon (CASPER workshop next week)
- Reference Python implementation available from:

http://github.com/sratcliffe/PySPEAD.git

- GPU accelerated en/decode not in the public release yet. Still deciding between CUDA and OpenCL.
- Promises to have a fairly large number of users which always helps !

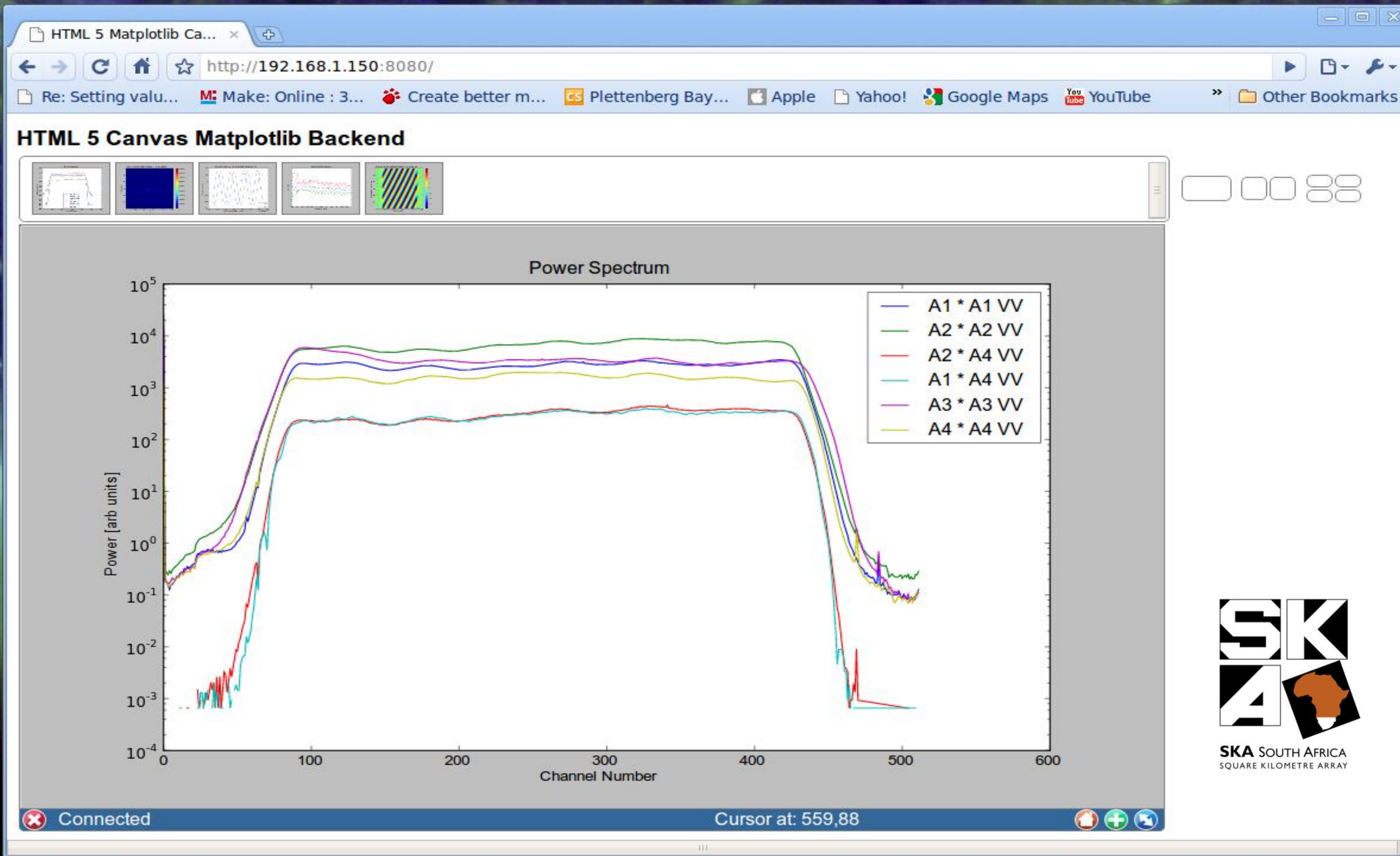
Signal Displays

- A certain subset of the live telescope data is made available (via SPEAD) to subscribing clients.
- This gives real time access to the data, and coupled with a wide variety of canned plots, allows extensive monitoring of the signal path.
- The displays are accessible via the standard MeerKAT iPython control shell.
- Diverse diagnostics such as ADC input histograms, amplitude and phase closures, spectral displays and dirty images can all be shown (and animated in real-time).

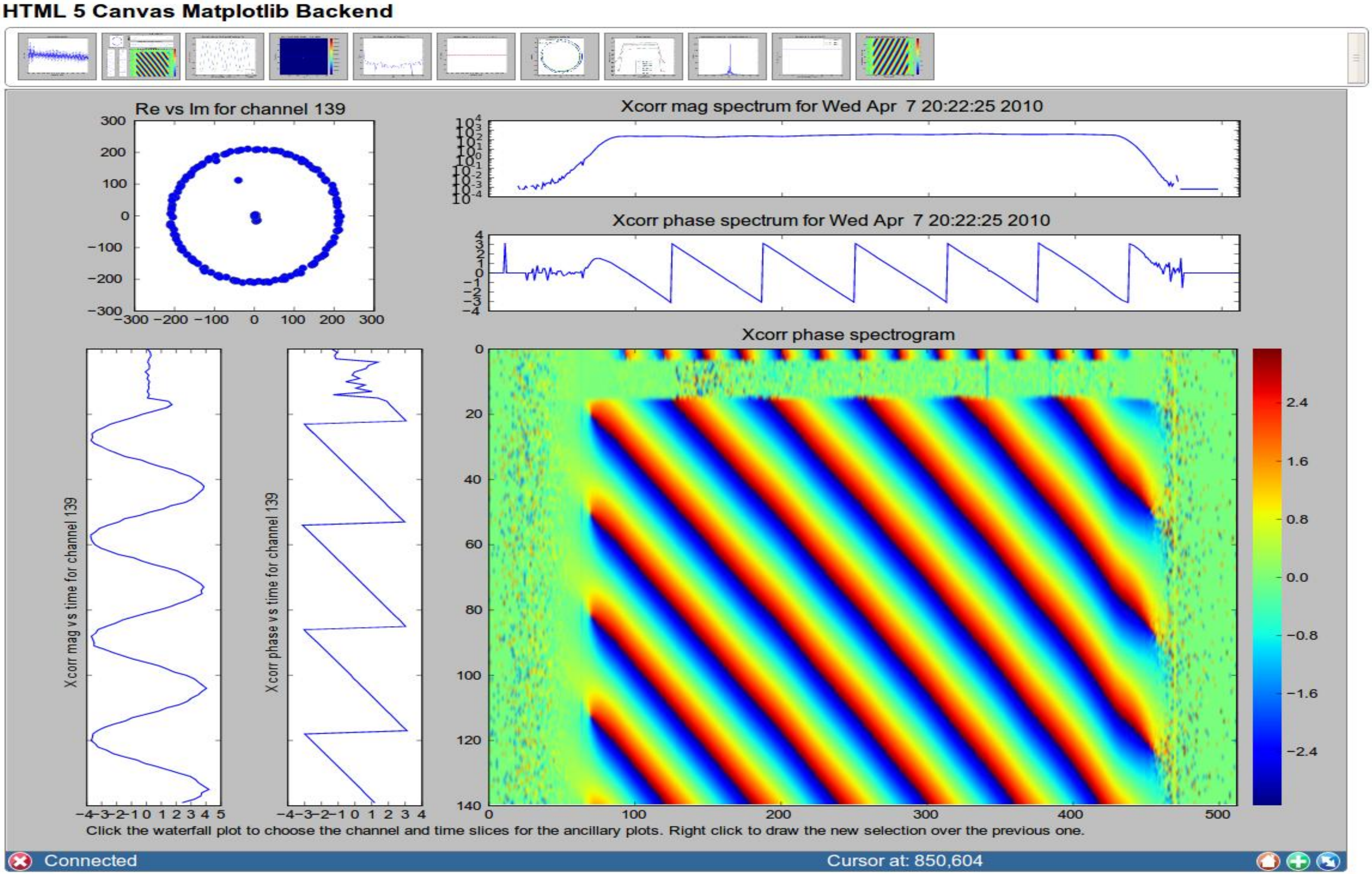
Matplotlib HTML5

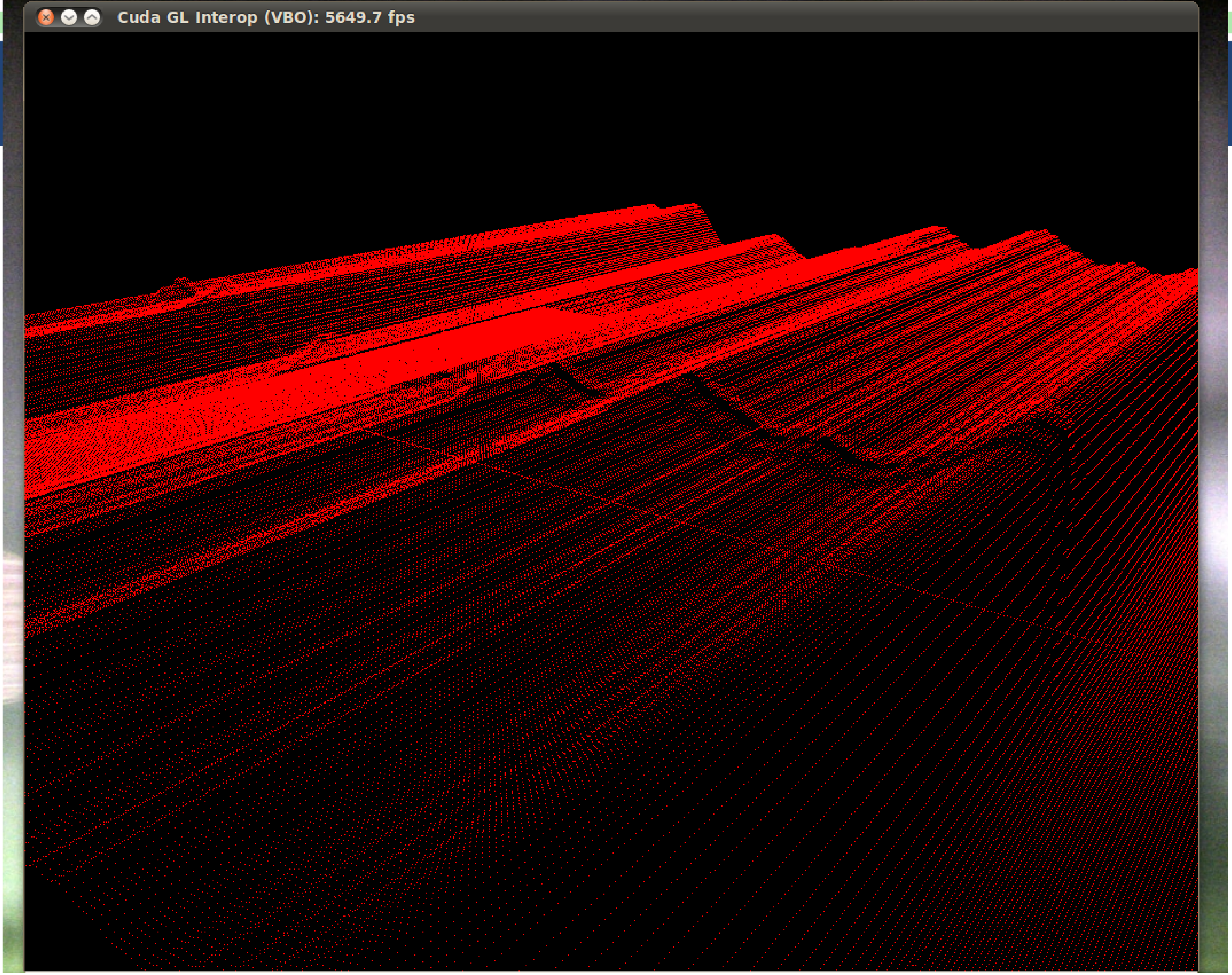
- Plotting for signal displays is handled via matplotlib.
- We have developed an HTML5 based matplotlib backend which allows the plots to be viewed from any location through a web browser.
- This provides a number of benefits:
  - A completely cross platform backend (Firefox 4.0+ / Chrome / Safari 5.0)
  - High speed animation (fairly complex plots can be animated up to 60 fps) and optimal network bandwidth usage (esp. compared to X forwarding)
  - User does not have to be collocated with the data to be processed (uses iPython distributed computing framework)
  - Pure Python module means no extra dependencies.
  - Thumbnail browser shows all available plots and allows easy switching between them.
  - Fully interactive including zooming and clickable axes.
  - Client data can persist through network disconnects and server process being killed.

KAT-7 Signal Displays



KAT-7 Signal Displays




Cuda GL Interop (VBO): 5649.7 fps

In Conclusion

- We have a first pass high performance ecosystem for handling our online processing and visualisation requirements.
- As an extension of the CASPER philosophy we are trying to provide a hardware building block with a range of easily extensible software modules.
- Flexible realtime data inspection critical for set to work and early commissioning, and probably for a long time thereafter.
- We believe that these standard software tools will allow rapid development of GPU based radio astronomy systems.