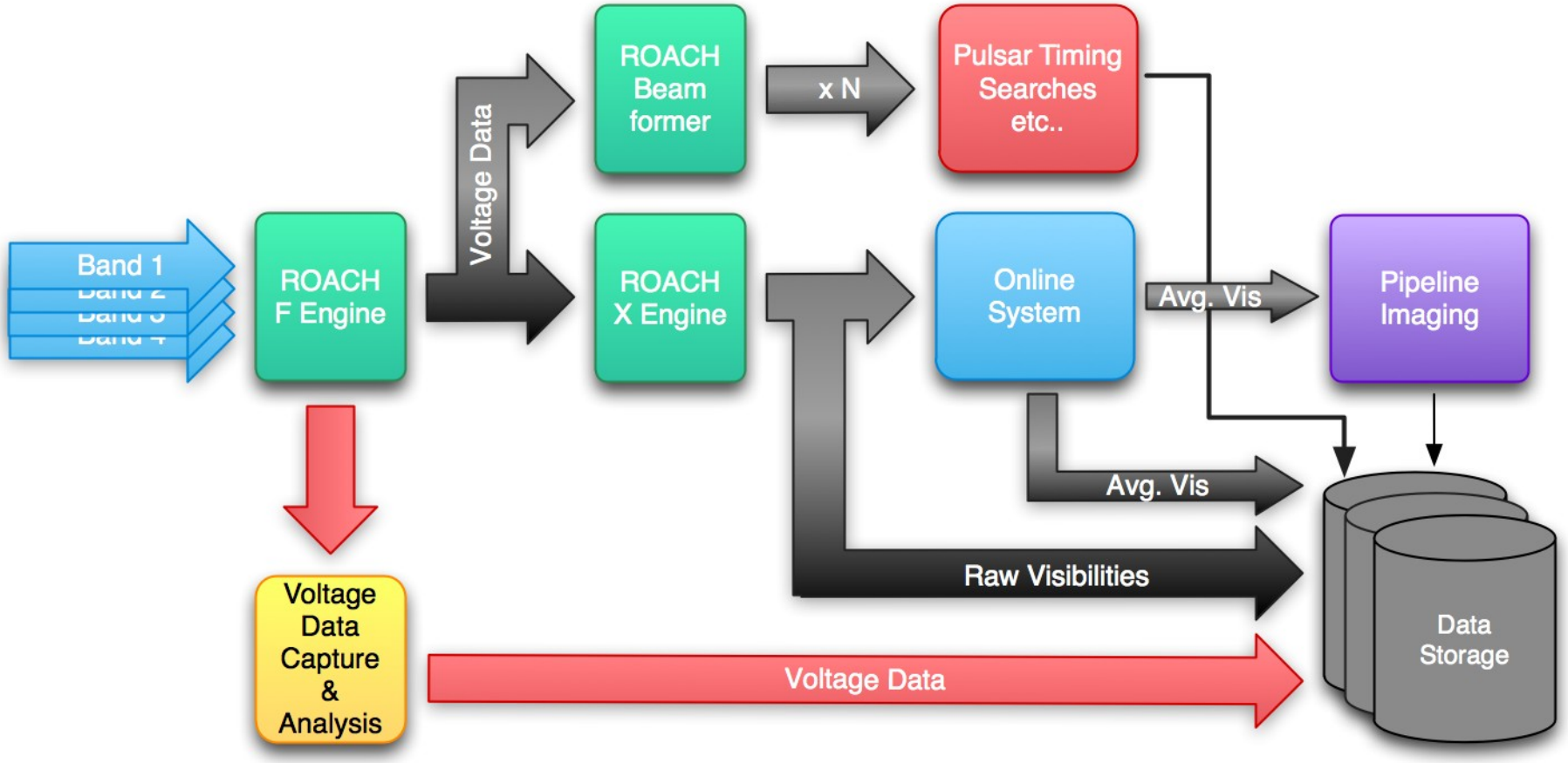
MeerKAT Data Architecture

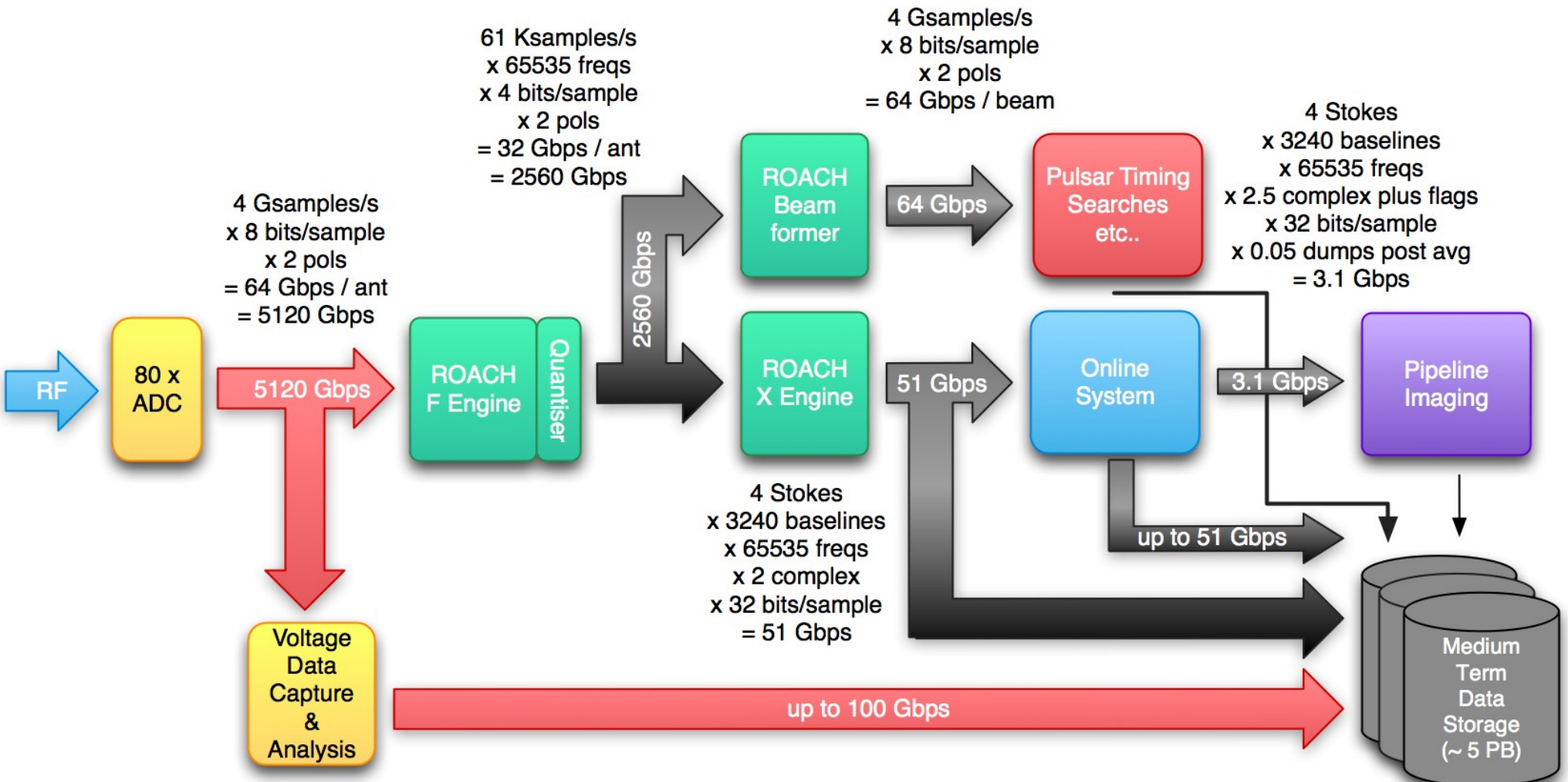
Simon Ratcliffe

MeerKAT Signal Path



SKA South Africa
Square Kilometre Array

MeerKAT Data Rates



SKA SOUTH AFRICA
SQUARE KILOMETRE ARRAY

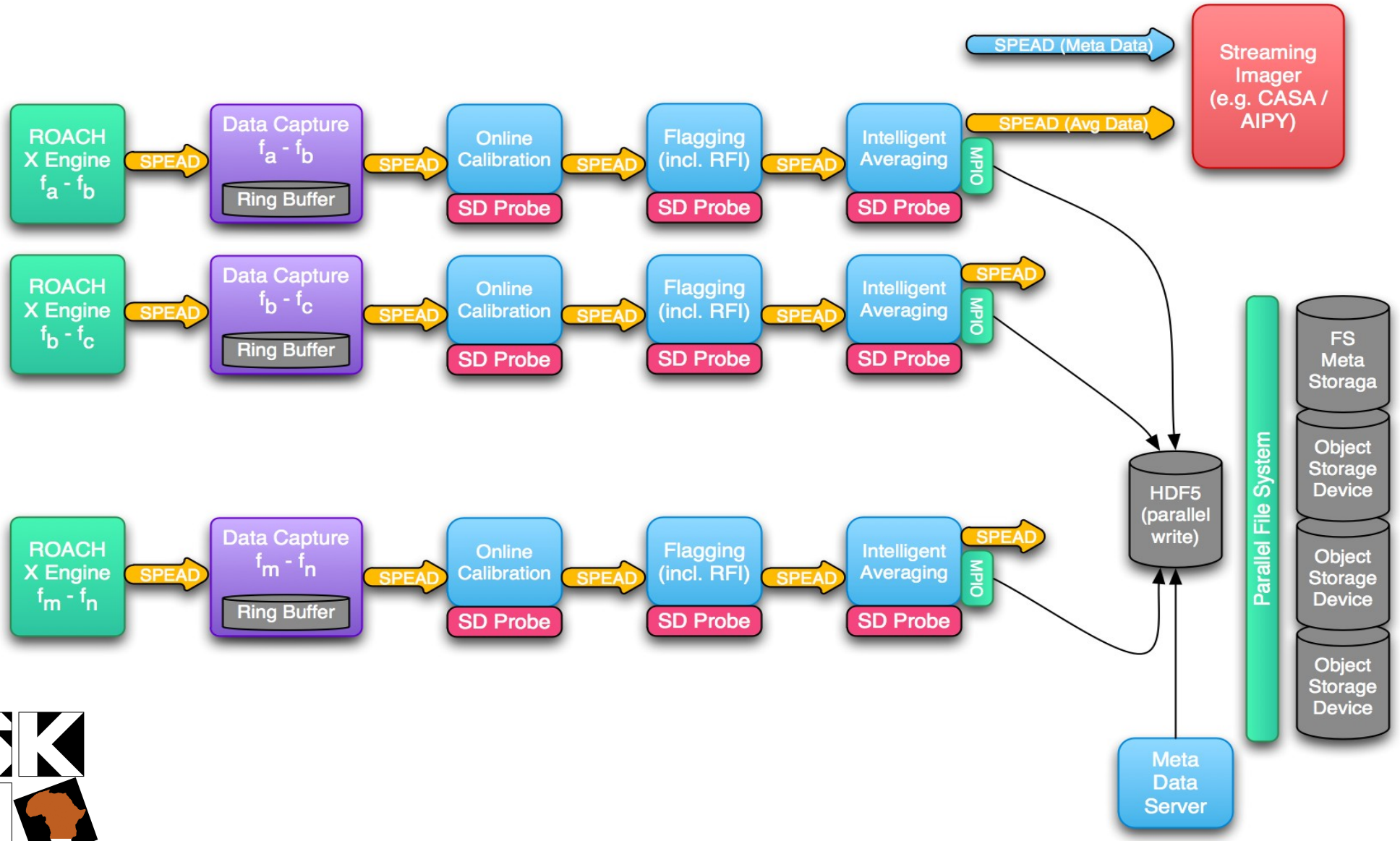
Online System

- The online system receives raw visibilities from the correlator at a sufficiently high dump rate to facilitate the following:
  - Continuous Tsys calculation
  - RFI Flagging
  - Baseline dependent time averaging
- The resultant visiblities + cal data + flagging are written to disk in the medium term archive. The averaging for this stream is under user control and variable up to no time averaging.
- A SPEAD stream of output data is also produced for downstream consumers such as the pipelined imager.

Online System Detail

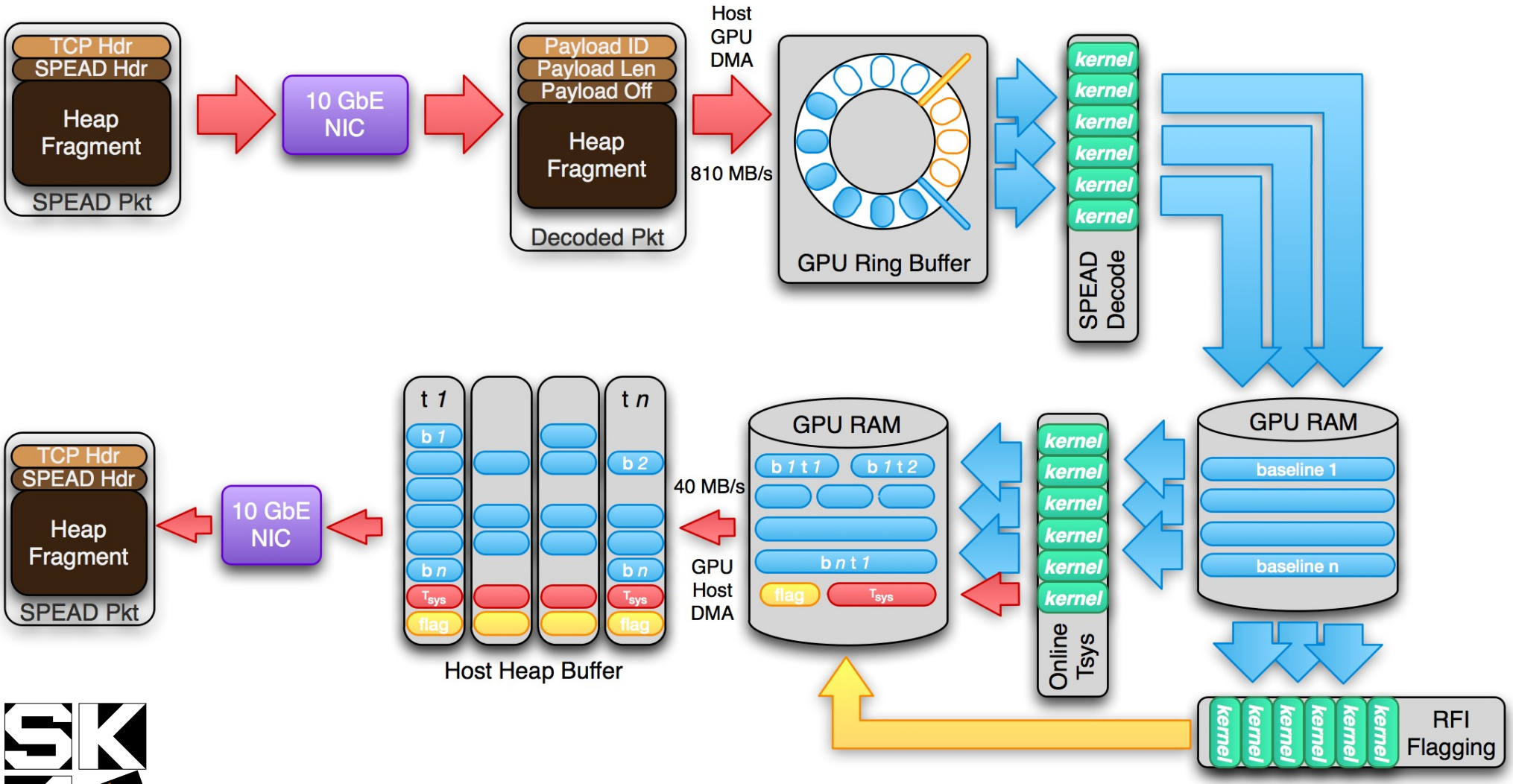
- Correlator output is split into a number of sub bands, each of which is processed in parallel.
- The split depends in the individual capacity of each element of the parallel system.
- With current technology, 8192 channels can be processed in a single element (with 1s correlator dump time) – limited by 10 GbE throughput.
- Parallel HDF5 output file allows multiple simultaneous writes from each system element.

Online System Detail



SKA SOUTH AFRICA
SQUARE KILOMETRE ARRAY

Online System Detail



SKA South Africa
Square Kilometre Array

Online Element 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.

SPEAD

- Streaming Protocol for Exchanging Astronomical Data
- Joint development between SKA South Africa and UC Berkeley.
- 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 needed a very small number of mandatories headers to ease generation of a SPEAD stream by lower powered devices (i.e. 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.

Transmitter

**Consider a grouping of data.
It has a static string value
that is not changing,
a counter called "MyCntr",
and an array of
32-bit unsigned ints called "MyArray".**

**We wish to propagate a change in both
"MyCntr" and "MyArray" to a receiver.**

**We construct a HEAP
containing the two ITEMs
we wish to transmit.**

**Item MyCntr has ID 0x167.
Item MyArray has ID 0x168.**

**This HEAP is then fragmented
into PACKETs.**

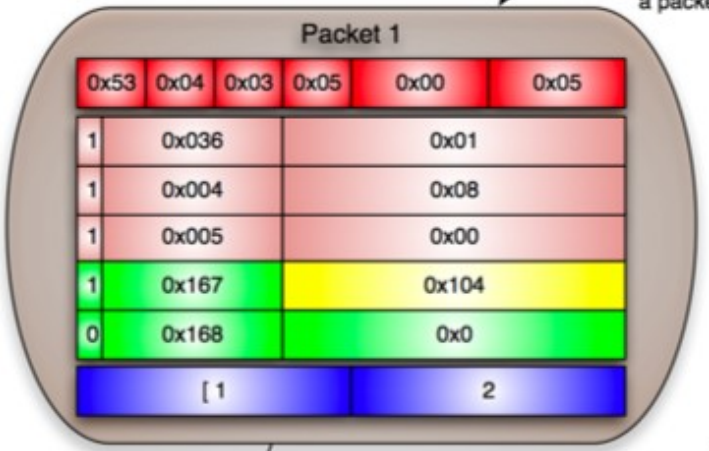
Silly example where
packets are very small.

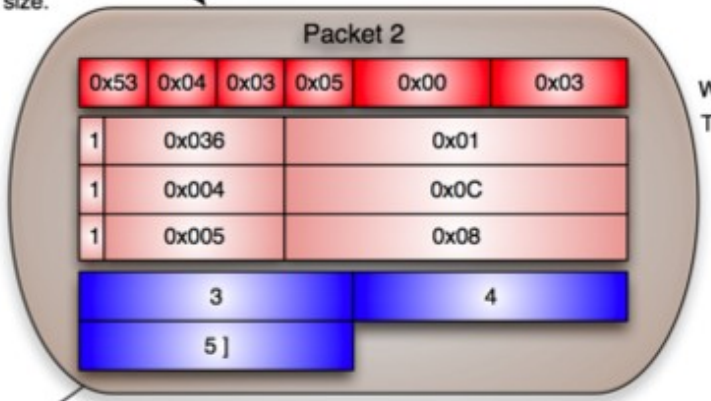
SPEAD does not mandate
a packet size.

SPEAD-40-24 header:
We have 5 items in this packet.

Required items:
This packet belongs to heap 1.
Payload is 8 bytes long.
Payload should be loaded
into start of heap (0x00).

User-defined item 0x167's value
of 0x104 can be found right here.
User-defined item 0x168's value
can be found at heap offset 0x0.
Heap payload
starting at address 0x0.

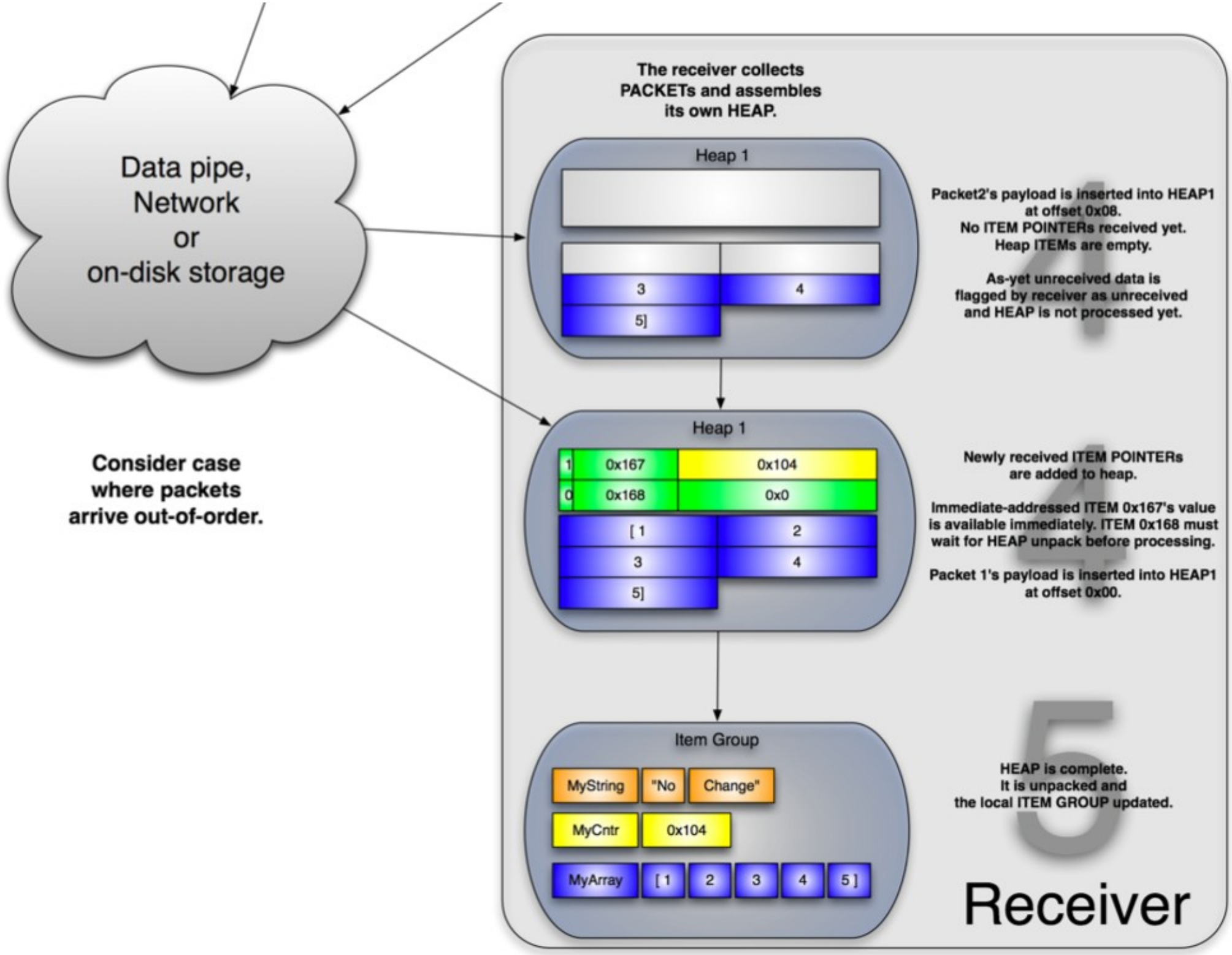




SPEAD-40-24 header:
We have 3 items in this packet.

Required items:
This packet belongs to heap 1.
Payload is 12 bytes long.
Payload should be loaded
into heap offset 0x08.

Heap payload
starting at address 0x08.


Data pipe,
Network
or
on-disk storage
Consider case
where packets
arrive out-of-order.
The receiver collects
PACKETs and assembles
its own HEAP.
Heap 1
3
4
5]
Packet2's payload is inserted into HEAP1
at offset 0x08.
No ITEM POINTERs received yet.
Heap ITEMs are empty.
As-yet unreceived data is
flagged by receiver as unreceived
and HEAP is not processed yet.
Heap 1
1
0x167
0x104
0
0x168
0x0
[1
2
3
4
5]
Newly received ITEM POINTERs
are added to heap.
Immediate-addressed ITEM 0x167's value
is available immediately. ITEM 0x168 must
wait for HEAP unpack before processing.
Packet 1's payload is inserted into HEAP1
at offset 0x00.
Item Group
MyString
"No
Change"
MyCntr
0x104
MyArray
[1
2
3
4
5]
HEAP is complete.
It is unpacked and
the local ITEM GROUP updated.
Receiver

SPEAD

- Specification is currently in revision K.
- Reference Python implementation available from:

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

- MeerKAT will use SPEAD within the correlator, online systems, and general access pipelines.
- Meta-data from telescope sensors will be broadcast as SPEAD streams for use throughout the processing chain.

File Output Support

- SPEAD is our standard on the wire protocol.
- Projects bringing their own equipment will be encouraged (and helped) to use this as their input format.
- HDF5 will most likely be our on disk format for both voltage and visibility data (mostly due to support for parallel writes).
- In the engineering phase we will support MS and uvfits. Other adapters easy to write due to availability of both meta and signal data streams.
- Likely MS will move to HDF5 based format at some stage

Signal Displays

- A certain subset of the live data is made available in real time to subscribing clients.
- This gives realtime 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 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

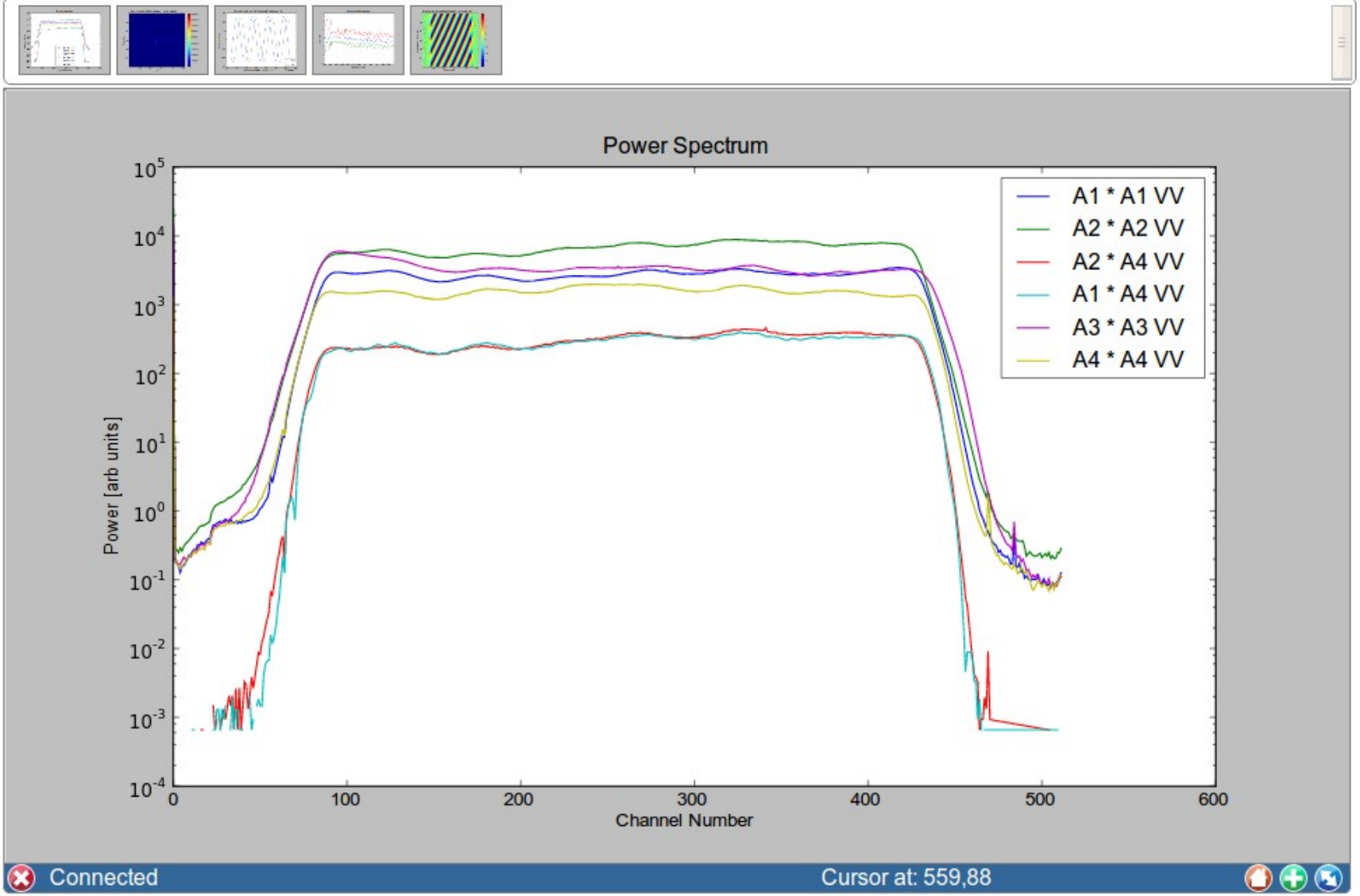
- Plotting for signal displays is handled via matplotlib.
- We have developed an HTML5 based matplotlib backend which allows the plots to be viewed from any location through a web browser.
- This provides a number of benefits:
  - A completely cross platform backend (any OS supported by either Chrome or Firefox)
  - 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.

HTML 5 Matplotlib Ca...

http://192.168.1.150:8080/

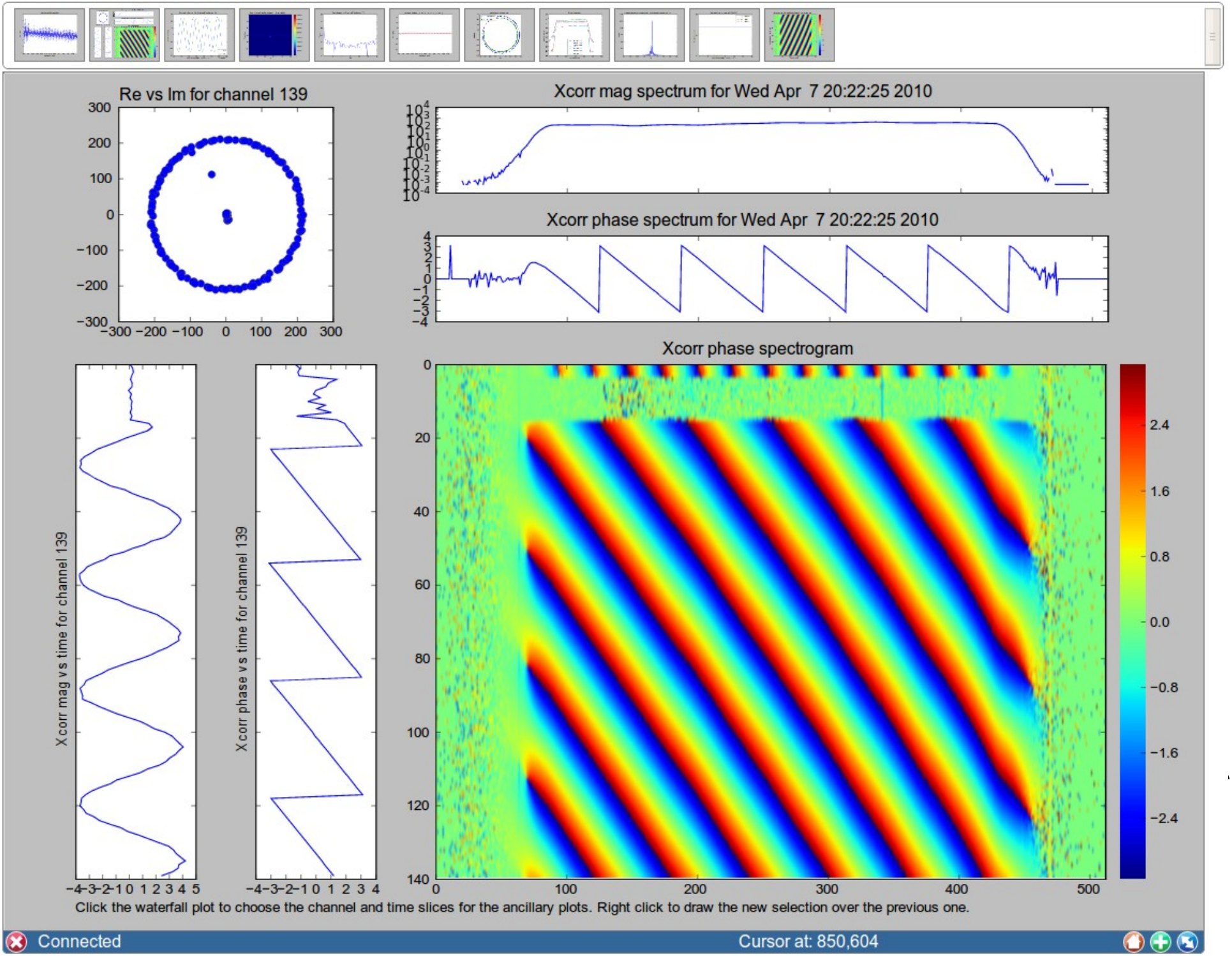
Re: Setting valu... | Make: Online : 3... | Create better m... | Plettenberg Bay... | Apple | Yahoo! | Google Maps | YouTube | Other Bookmarks

HTML 5 Canvas Matplotlib Backend



SKA SOUTH AFRICA
SQUARE KILOMETRE ARRAY

HTML 5 Canvas Matplotlib Backend



Early Access and Collaboration

- We are just beginning our work on the post correlator architecture.
- Feedback and involvement from the user community will greatly aid us in developing and refining the requirements.
- Early involvement in these discussions will naturally lead to early access to both KAT-7 and MeerKAT :)

In Summary

- We hope to have a functional and flexible data architecture for MeerKAT within the next year.
- This will be built out to include a range of standard products, as well as interfacing to more custom projects.
- Users will be able to request data from a variety of stages at a variety of rates.
- Inspection tools should be useful to both engineering staff and scientific end users.