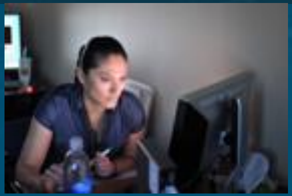
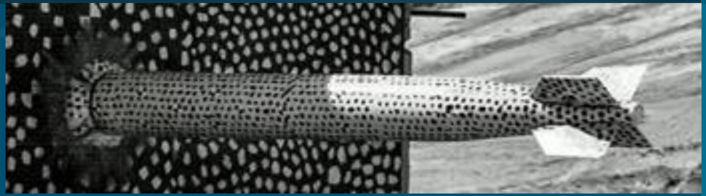
Sandia National Laboratories

The GR PDU Utilities

PRESENTED BY

Jacob Gilbert – at the 2019 GNU Radio Conference

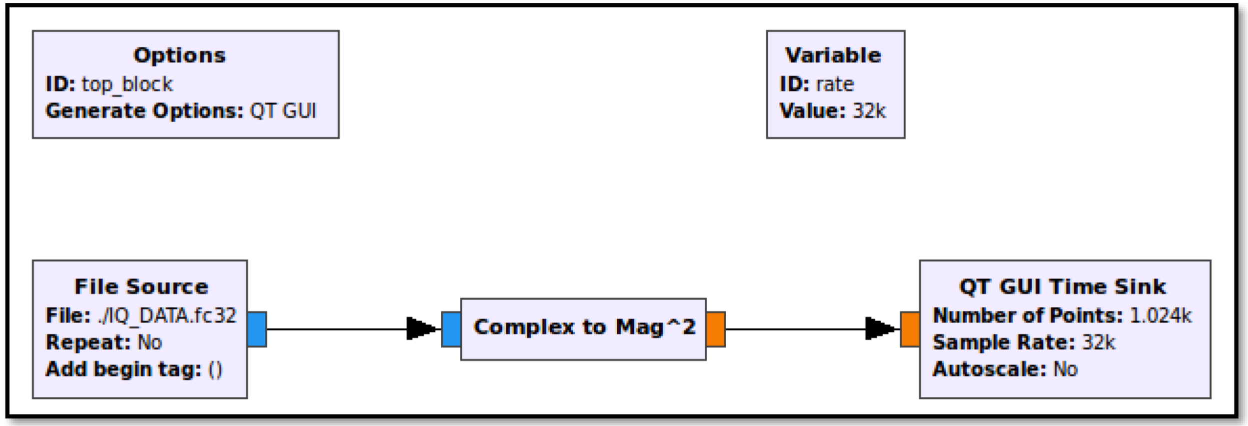
jacob.gilbert@sandia.gov

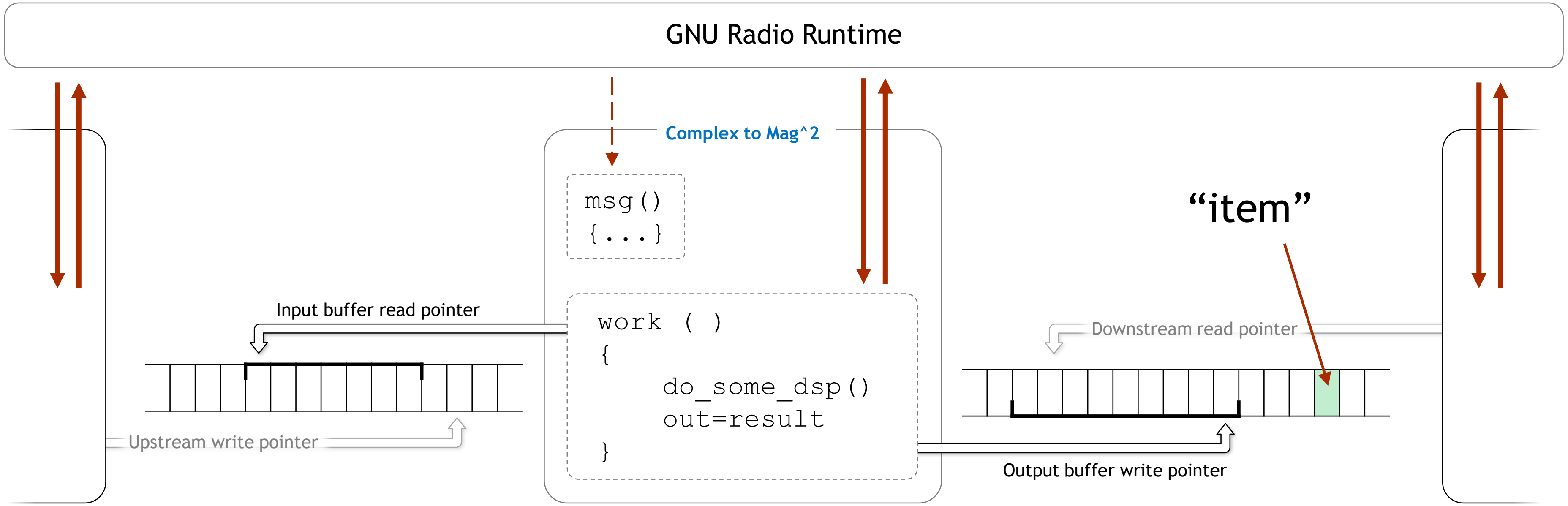
SAND2019-10616C

U.S. DEPARTMENT OF ENERGY NNSA

GR Operation: Streaming

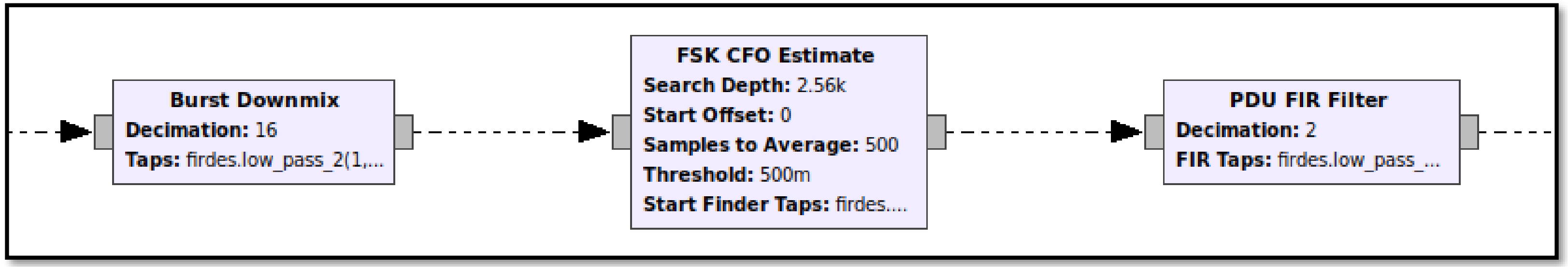
SAND2019-10616C

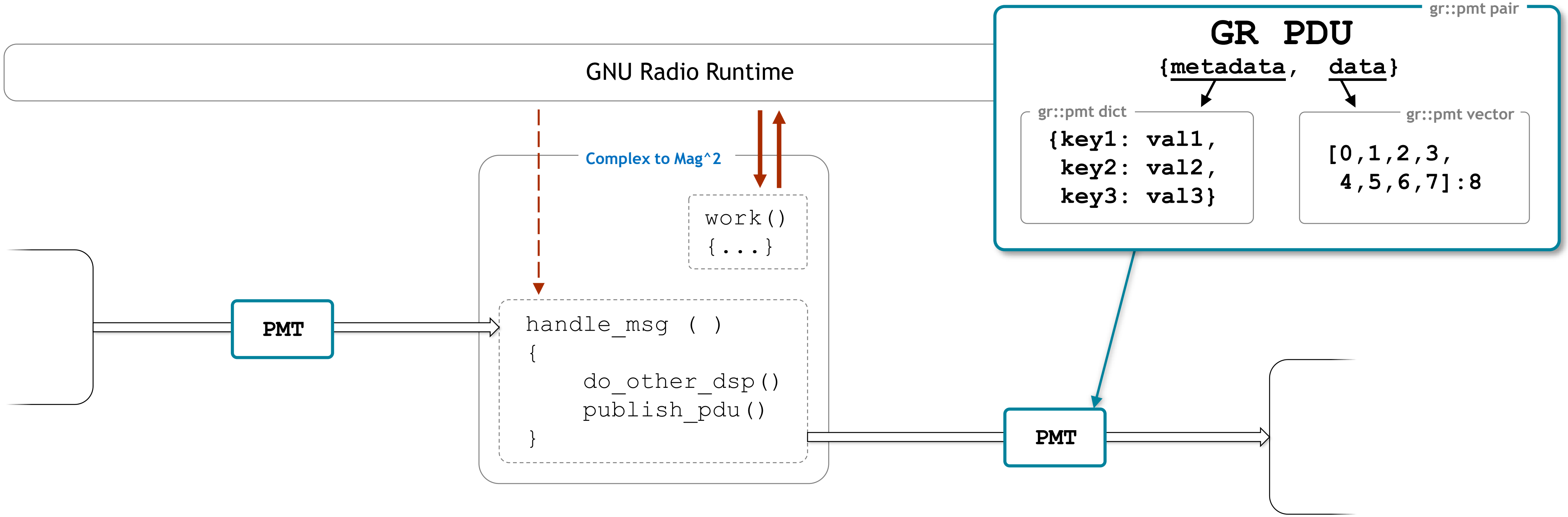




SAND2019-10616C

GR Asynchronous Message Passing / PDUs





SAND2019-10616C

Motivation

SAND2019-10616C

Motivation for the PDU Utilities

- Receive a burst of information, respond with burst based on the received data
- Current tools exist but have limitations:
  - gr-uhd Stream Tags
  - Tagged Stream Blocks
  - GR Packet Communications API
  - gr-eventstream
- Complicated by real-world radio protocol constraints:
  - Minimum RF turnaround time (latency)
  - Other TDMA system constraints such as relative burst timing
  - Frequency agility, RF tuning and tracking
- 'We tried GNU Radio but there was too much latency so we used (libuhd / C++ / RFNoC / X-Midas) instead'

SAND2019-10616C

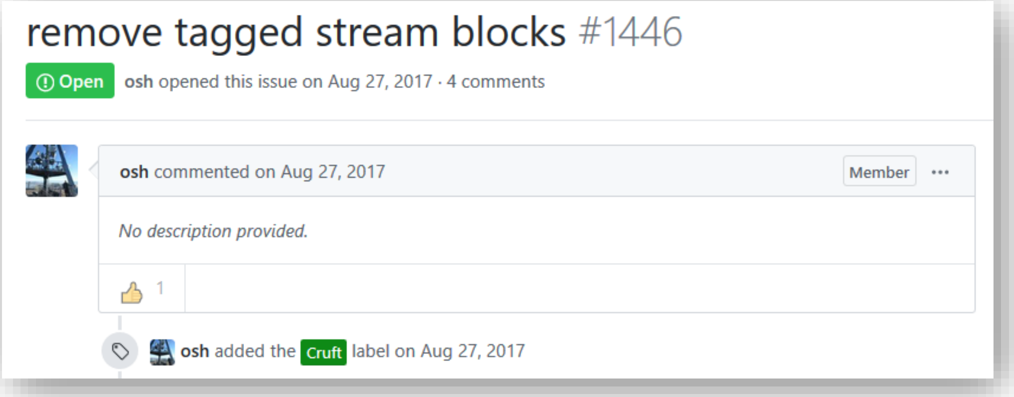
Required Capabilities

- RX: Convert received bursts of energy into discrete datasets for further processing
  - ~Sample accurate time estimate of bursts
  - Fixed or arbitrary frequency operation
- TX: Convert data into bursts of modulated data
  - ~Sample-accurate transmission time
  - Minimize RX/TX latency through SDR system
  - Fixed or arbitrary frequency operation
- Portable: minimize processing hardware dependencies
- Robustness: recover from overflows or dropped data
- Efficient: design for lower power embedded hardware

SAND2019-10616C

Existing Toolset

Existing In Tree Tools



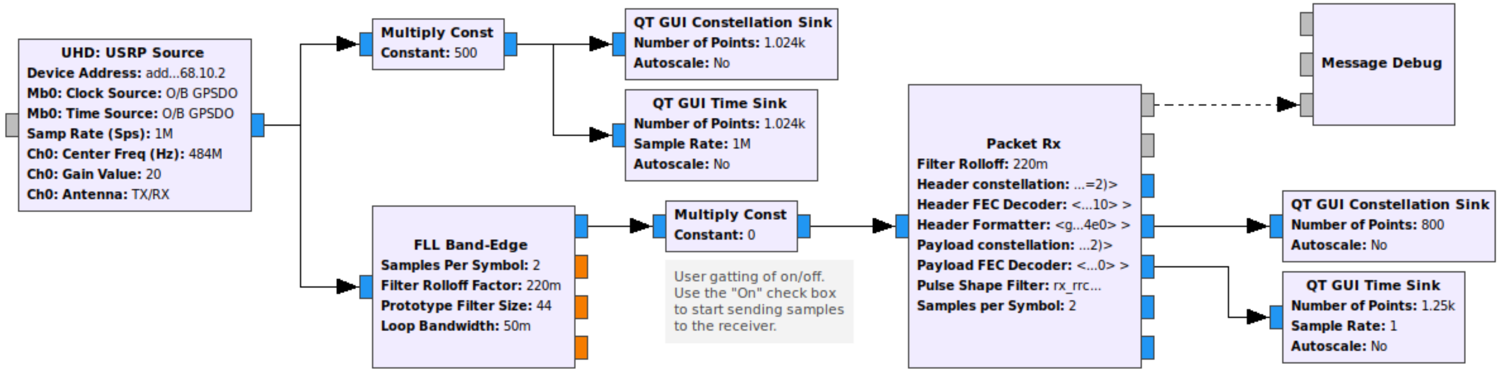
- GR Tagged Stream API
  - Support for burst processing within the streaming architecture of GR
  - Going to be deprecated?

"there's a lot of bugs and inefficiencies, and quite some grief[1] and undocumented contracts connected to TSBs" – M. Müller, 2017

- gr-uhd Stream Tags
  - 'rx_time' provides for RX time tracking, 'tx_time' allows synchronized, timed transmissions
  - 'tx_sob' and 'tx_eob' tags allow for single port half-duplex transceiver
  - Not a complete solution, still streaming based
  - Since this relies on hardware, simulation can be difficult (weird SW emulation)
  - Still very useful – leveraged by gr-pdu_utils and extended to Sidekiq hardware

Existing In Tree Tools: Packet Communications API

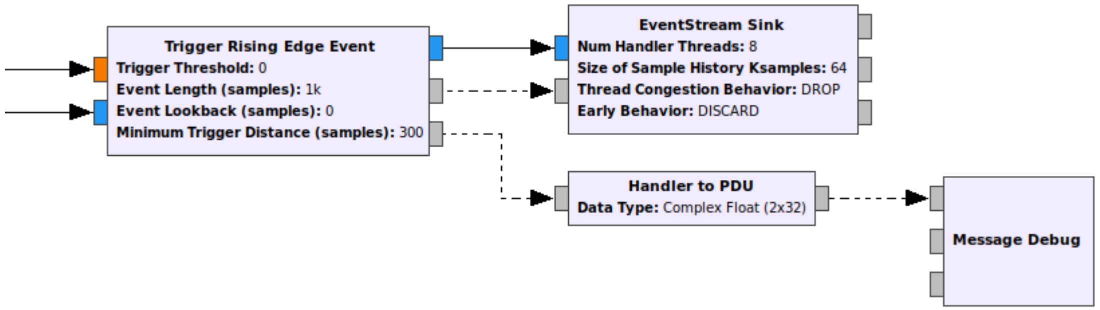
- Powerful packet processing schema, somewhat complicated (makes use of TSBs…)

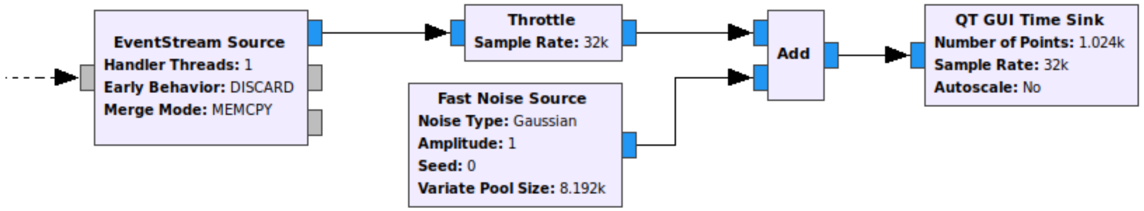


- Appears to use open loop timing, lots of rework to meet our requirements
- Doxygen Documentation: https://www.gnuradio.org/doc/doxygen/page_packet_comms.html
- Interesting Module Example: https://www.gnuradio.org/deptofdefense/gr-uaslink
- Reasonable option for custom protocols and more complicated modulations
- We found the PDU Utilities to be easier for existing / simple protocols

OOT Modules: gr-eventstream

- Allows burst processing with closed loop timing by interleaving bursts into TX stream
- Relies on conventional GR backpressure: buffers will impact latency
  - Minimize number of streaming blocks between ES source and HW Sink
- Good description of functionality: https://oshearesearch.com/index.php/tag/eventstream/





- Limitations for our use: T/R latency, different overall approach
  - Possible that we will integrate ES concepts in future

SAND2019-10616C

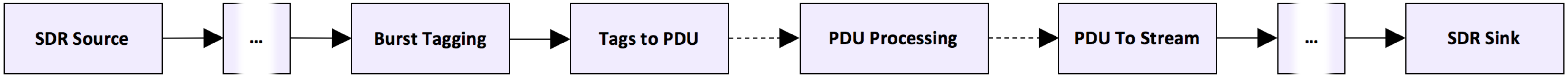
Theory of Operation

GRCon 2019 HUNTSVILLE, AL

Usage of the PDU Utilities

SAND2019-10616C

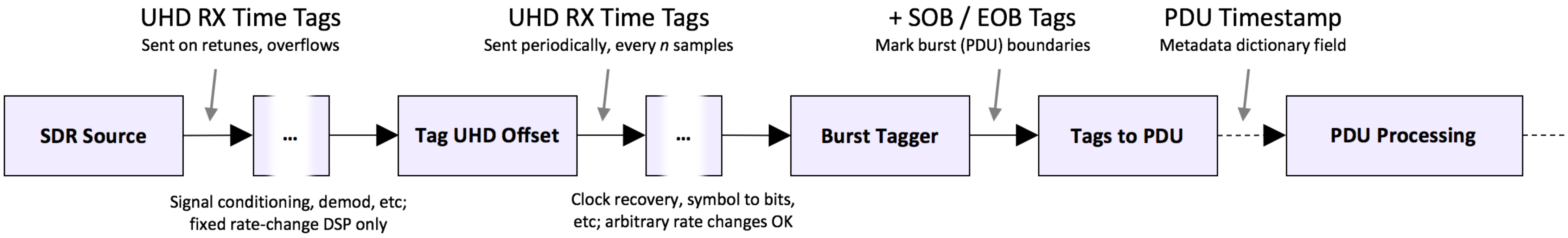
- Robust set of blocks for converting between streaming and async message (PDU)

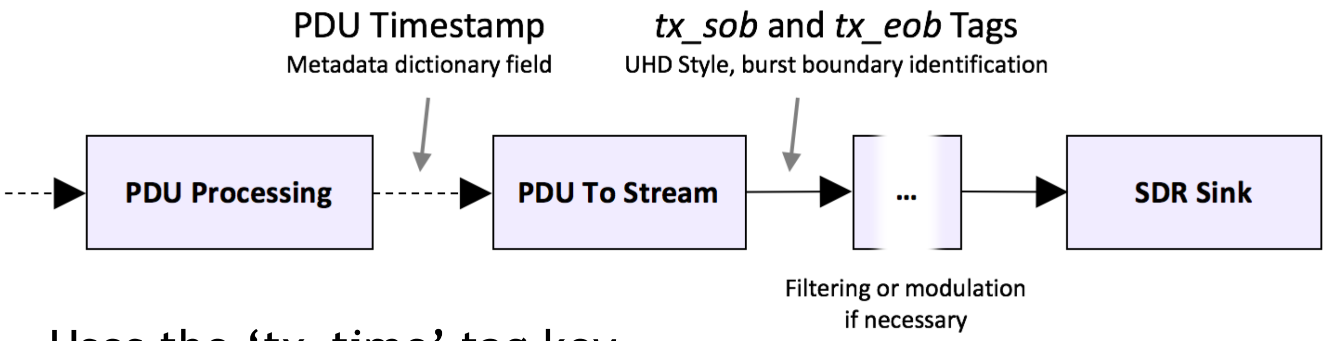


- Receive energy, basic preprocessing
- Identify start/stop of burst
- Convert required data elements to PDU
- Keep track of time
- Do higher layer processing
- Convert PDU to stream if required
- Emit RF energy at specified time

SAND2019-10616C

PDU Utilities Time Management





Uses the ‘tx_time’ tag key

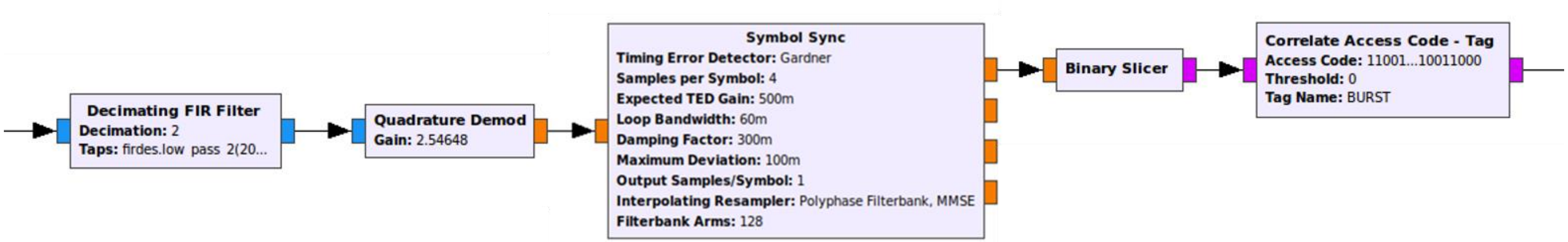
The Timing Utilities Module

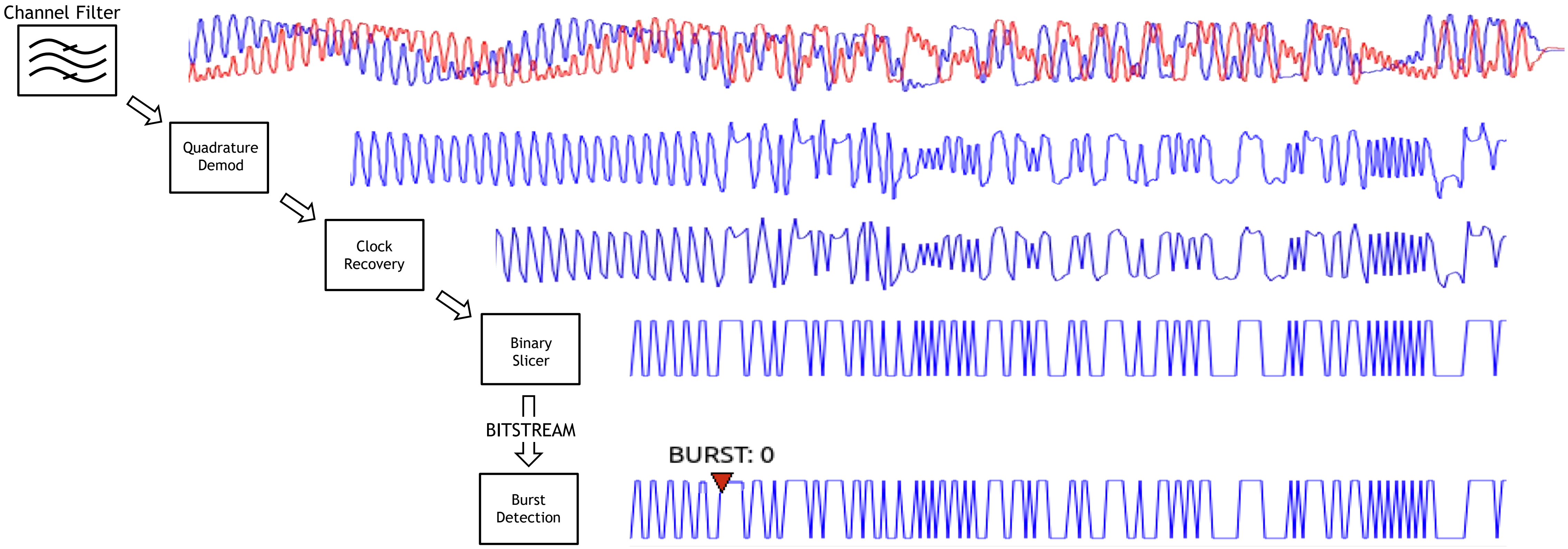
- Separated from gr-pdu_utils to contain various time tracking functions
  - Some complexity is now OBE
- Lots of time tuple related code
  - Tracking UHD Time Tuples
  - Adding periodic Time Tuples
- Does this need to be a separate module?
  - No, BUT…

UHD Time Tuple:	(rx_time	{1432	0.912341})		
	[Key]	[Integer Secs]	[Fractional Secs]	[Sample Offset]	[Sample Rate]
Timing Utils Tuple:	(rx_time	{1432	0.912341	243123411	1000000.0})

SAND2019-10616C

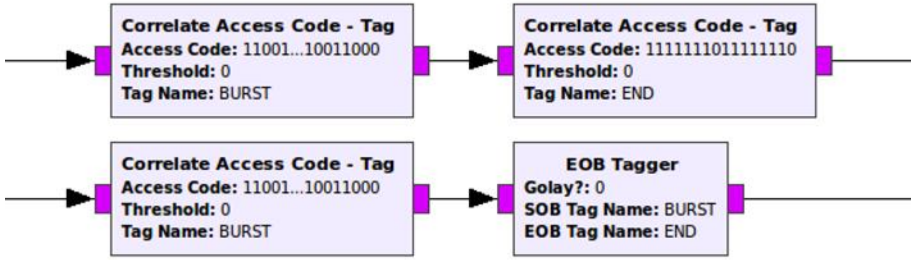
Extending The ‘Basic FSK Receiver’ Example

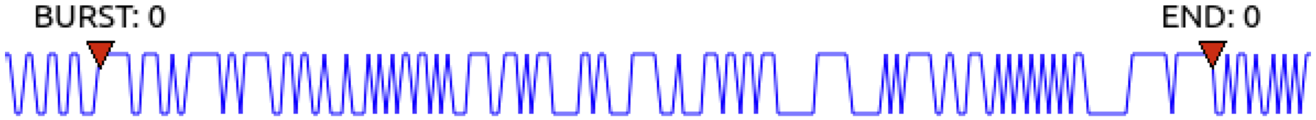




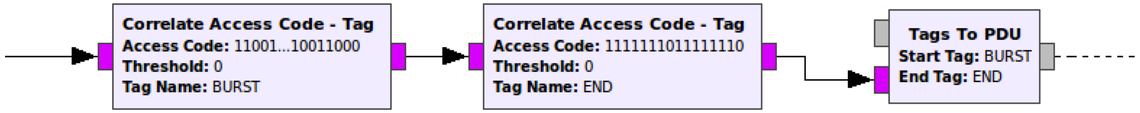
Extending The 'Basic FSK Receiver' Example

- We have a bitstream and know where bursts start…now what?
- Encapsulate a complete burst of data
  - End-of-burst sequence detection
  - Encoded length field in header
  - Blind (fixed length after burst detection)





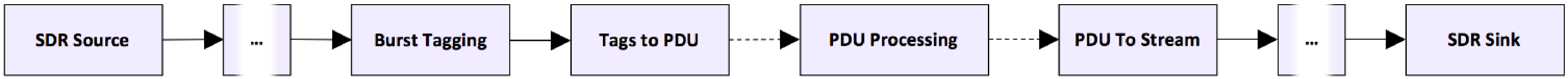
- Create a PDU from received data




```
* MESSAGE DEBUG PRINT PDU VERBOSE *
((pdu_num . 1) (pdu_id . 1262624438) (time_type .
uhd_time_tuple) (burst_time . {0 0.103434}))
pdu_length = 72
contents =
0000: 00 00 01 00 01 01 00 01 01 01 00 01 00 01 00 00
0010: 00 00 00 00 01 01 01 01 00 01 01 00 01 00 00 01
0020: 01 00 01 01 00 00 00 01 00 01 01 01 00 00 01 01
0030: 00 01 00 01 00 00 00 00 01 00 01 00 00 01 00 00
0040: 00 00 00 00 01 01 01 01
```

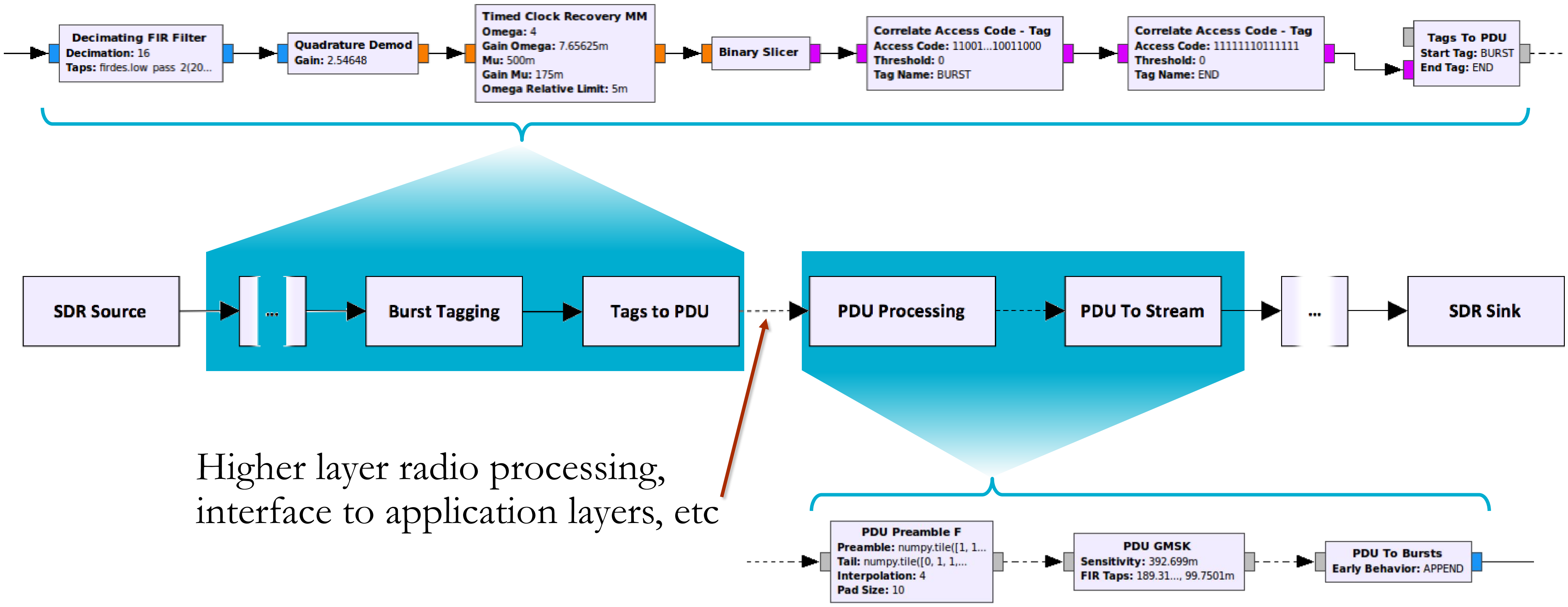
- Pass this information to a higher-layer processor

Building a Simple TDMA FSK Modem



SAND2019-10616C

Building a Simple TDMA FSK Modem



Higher layer radio processing, interface to application layers, etc

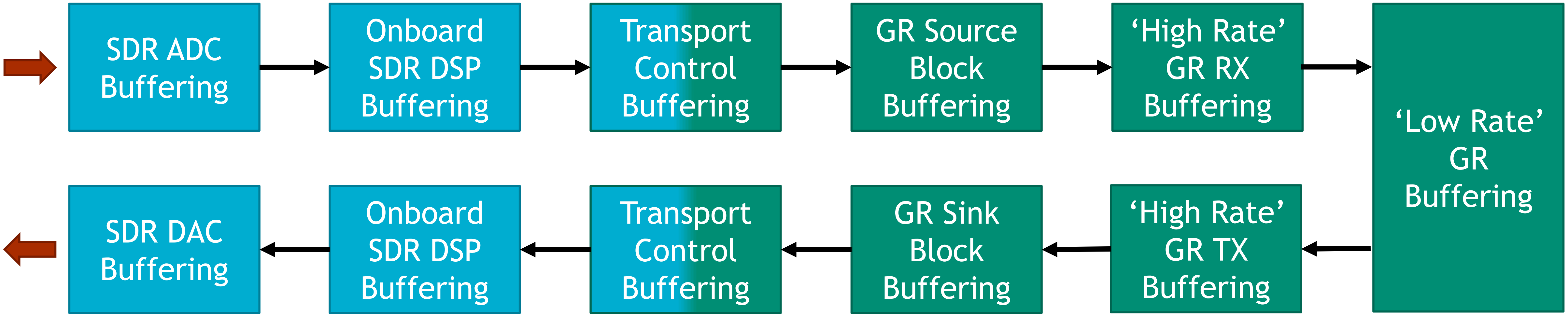
SAND2019-10616C

"Low Latency"…?

- 'Low latency' is a key design consideration
  - Minimize time from burst RX to burst TX
- Blame the OS / Scheduler?
  - Its usually buffering…
  - Tradeoff…small buffers often means less efficient processing

$$TR\_latency_{min} \geq \left(\frac{data\_buffer\_size}{sample\_rate}\right)_{max}$$

✓ Reduce Data Buffer Sizes
✓ Increase Sample Rate
X More CPU Utilization



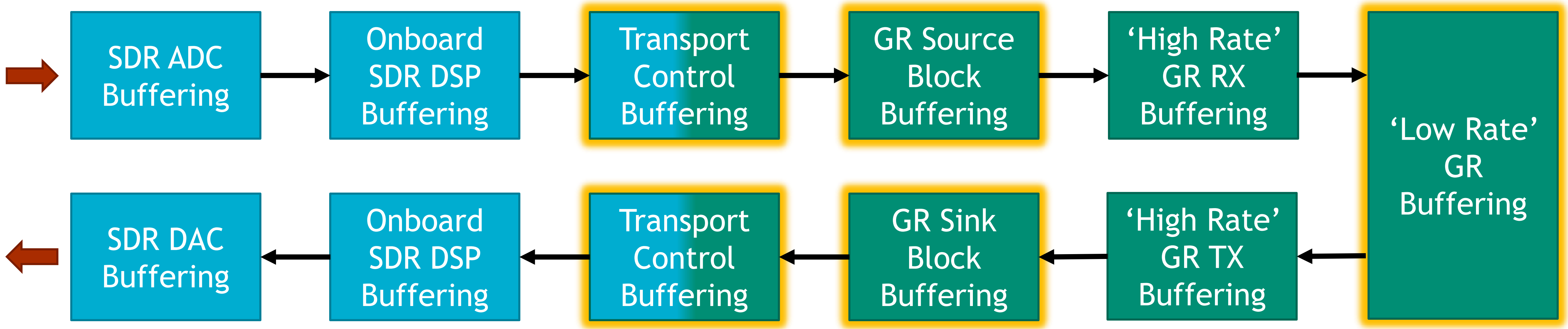
SAND2019-10616C

"Low Latency"…?

- 'Low latency' is a key design consideration
  - Minimize time from burst RX to burst TX
- Blame the OS / Scheduler?
  - Its usually buffering…
  - Tradeoff…small buffers often means less efficient processing

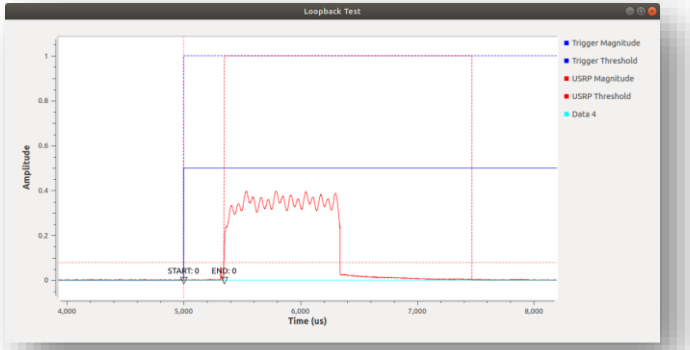
$$TR\_latency_{min} \geq \left(\frac{data\_buffer\_size}{sample\_rate}\right)_{max}$$

✓ Reduce Data Buffer Sizes
✓ Increase Sample Rate
X More CPU Utilization

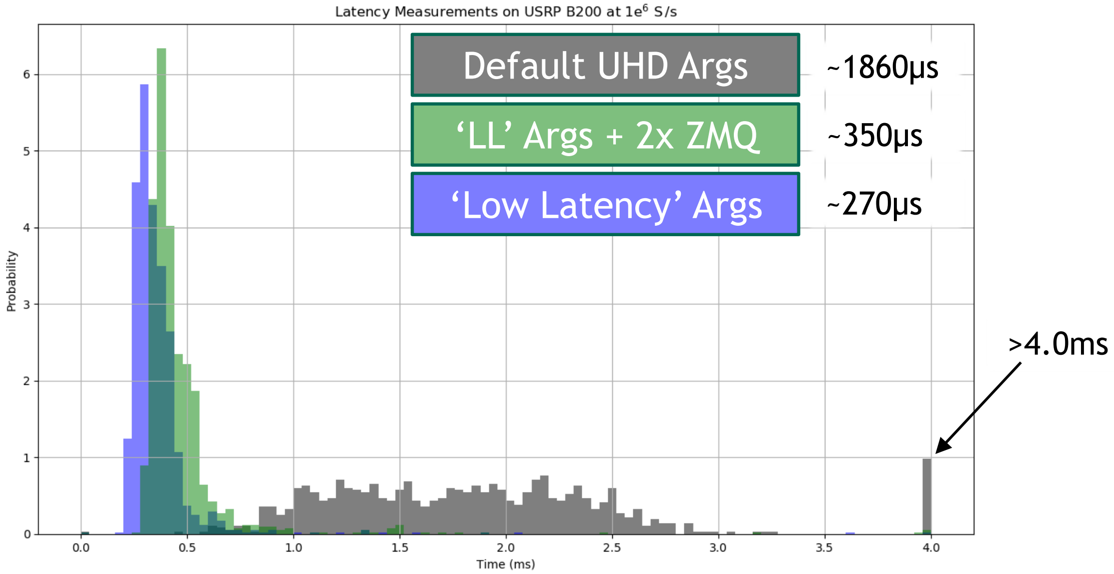


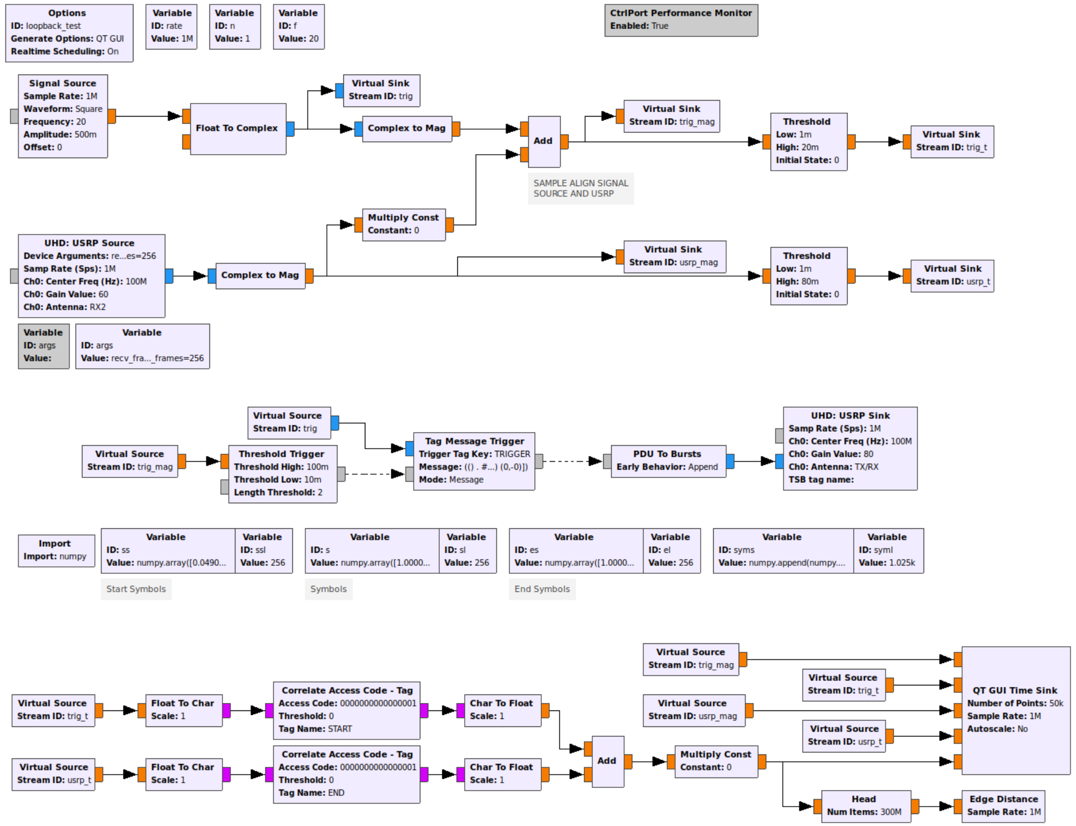
SAND2019-10616C

Characterizing Round Trip Latency



- Flowgraph included in Timing Utilities
  - PULSE → RX → PDU → […] → TX → SCOPE latency requires external equipment
  - TRIG → PDU → […] → TX → RX → TIME SINK is simpler (terminate TX port of USRP)





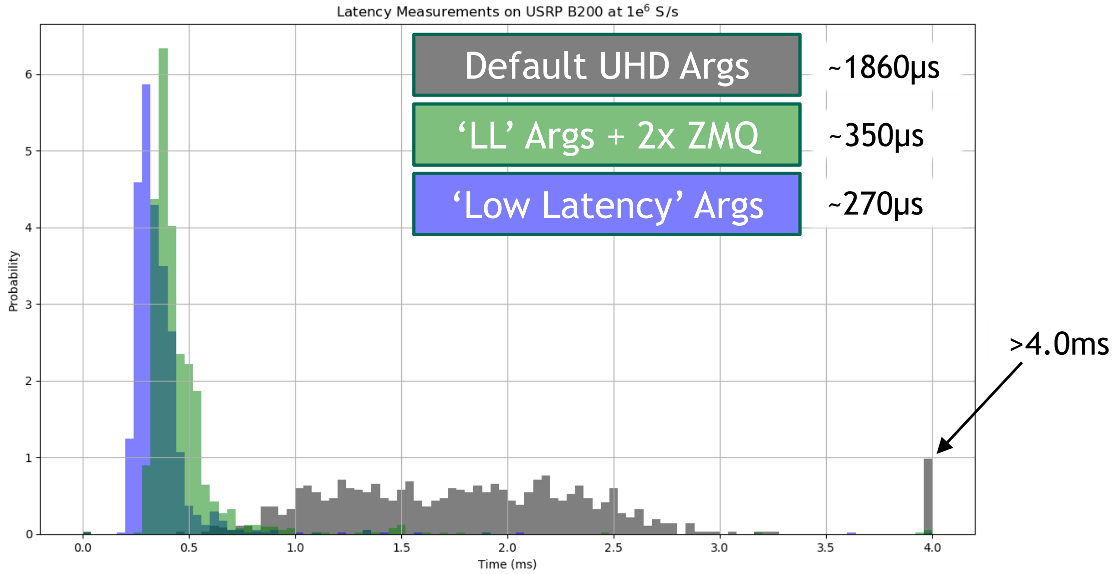
Low Latency UHD Arguments:

"recv_frame_size=256, num_recv_frames=256, send_frame_size=256, num_send_frames=256"

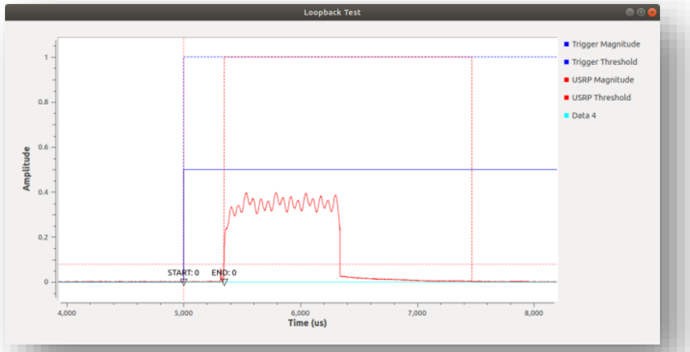
SAND2019-10616C

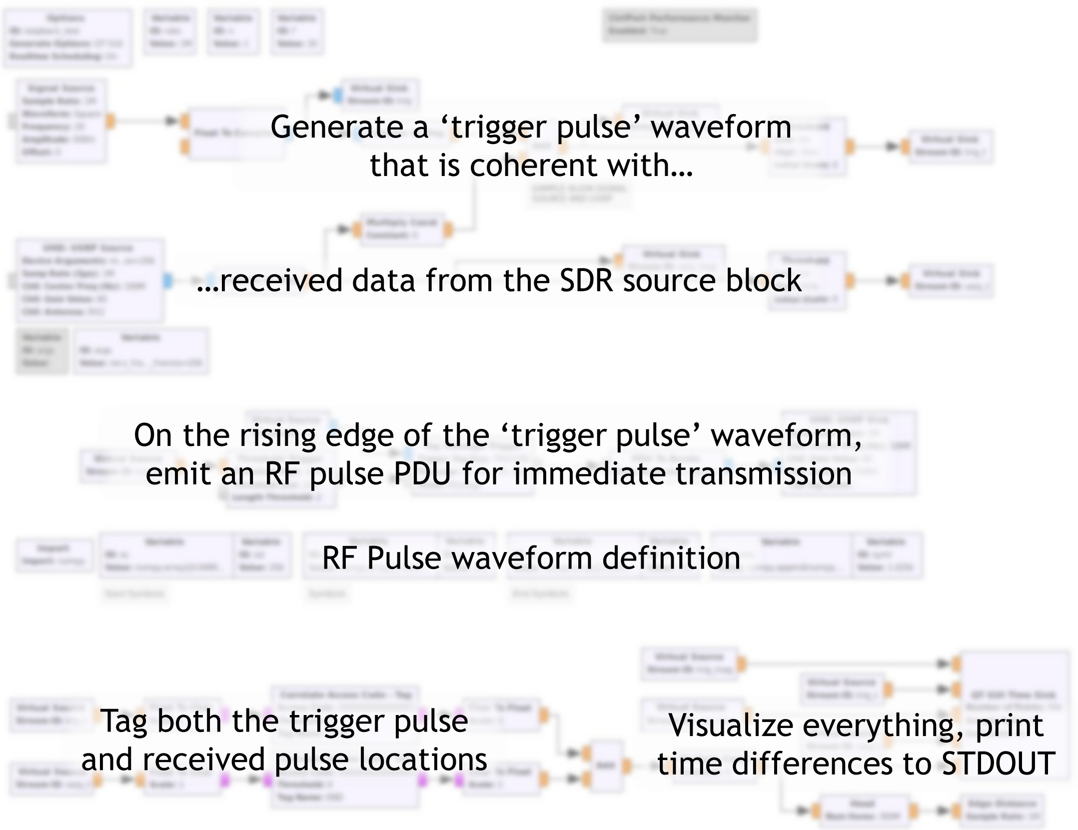
Characterizing Round Trip Latency

- Flowgraph included in Timing Utilities
  - PULSE → RX → PDU → […] → TX → SCOPE latency requires external equipment
  - TRIG → PDU → […] → TX → RX → TIME SINK is simpler (terminate TX port of USRP)



Low Latency UHD Arguments:
"recv_frame_size=256, num_recv_frames=256, send_frame_size=256, num_send_frames=256"



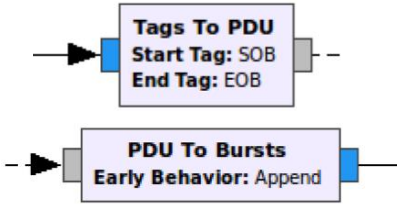


Notable Blocks in the PDU Utilities Module

- PDU/Stream Conversion:
  - Tags to PDU
  - PDU to Bursts
  - Take/Skip to PDU
  - Tag Message Trigger
- In-Tree Streaming Analog:
  - Complex to Mag^2
  - Keep 1 in N
  - Binary Arithmetic
  - Up/Downsample
  - PFB Arb Resampler
  - FIR Filter
  - FM Modulator
  - Quadrature Demod
- New PDU Manipulation Blocks:
  - Pack/Unpack
  - PDU Alignment
  - Commutator
  - Data Encoding
  - Extract Field
  - PDU WPCR
  - PDU Logger
  - Flow Controller
- Debugging Tools:
  - PDU Counters
  - QT PDU Source
  - PDU Message Gate
  - Random Drop

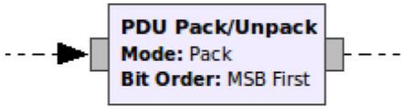
SAND2019-10616C

Notable Blocks



- PDU / Stream Conversion have been covered pretty well

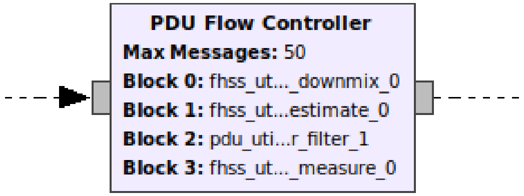
- Emits messages based on certain conditions
  - Robust arming/triggering architecture based on messages or stream tags
  - Re-trigger holdoff, arm timeout, TX limits, etc
  - Can emit fixed Messages or Timed PDUs



- Converts data within U8 PDU formats
  - Stuff U8 bits into U8 bytes, bit swap, etc
  - Requires work for higher order PDU translation

- Whole Packet Clock Recovery Symbol Synch
  - Based on M. Ossmann's WPCR project[1]
  - Operates well between 4 and 60 Samples/Symbol

[1] https://github.com/mossmann/clock-recovery/blob/master/wpcr.py



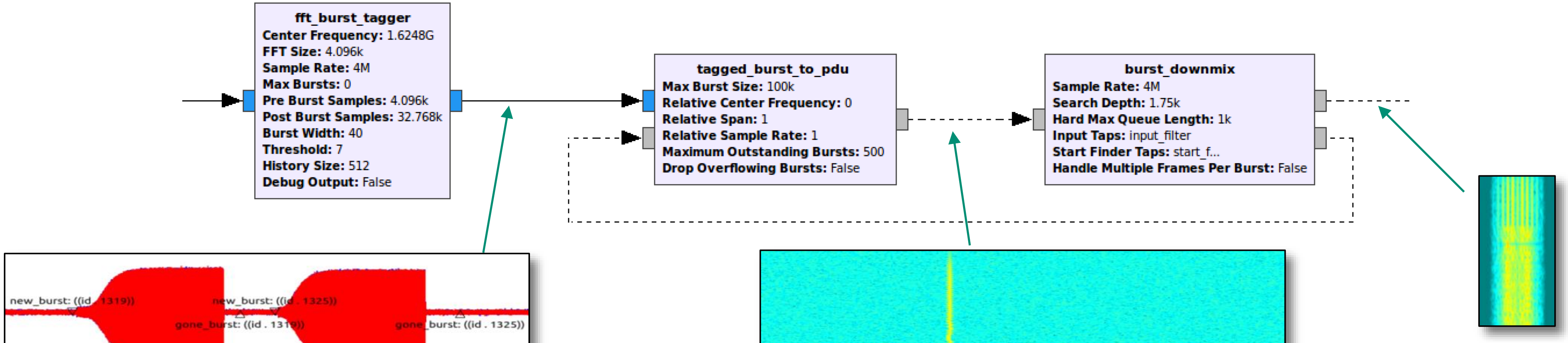
- Asynchronous message passing: no flow control
  - Block queues are checked, PDUs block if >Max
  - Helpful for highly variable input situations

SAND2019-10616C

Other Sandia GR Modules

FHSS Utilities

SAND2019-10616C

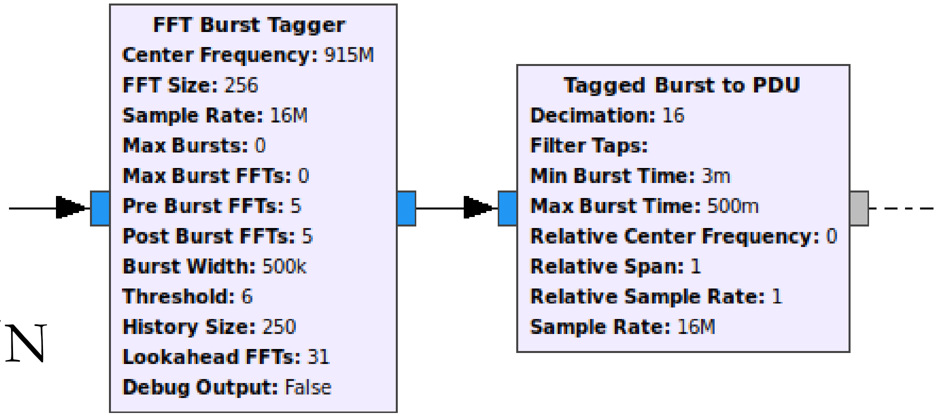


Decimated and Centered Bursts

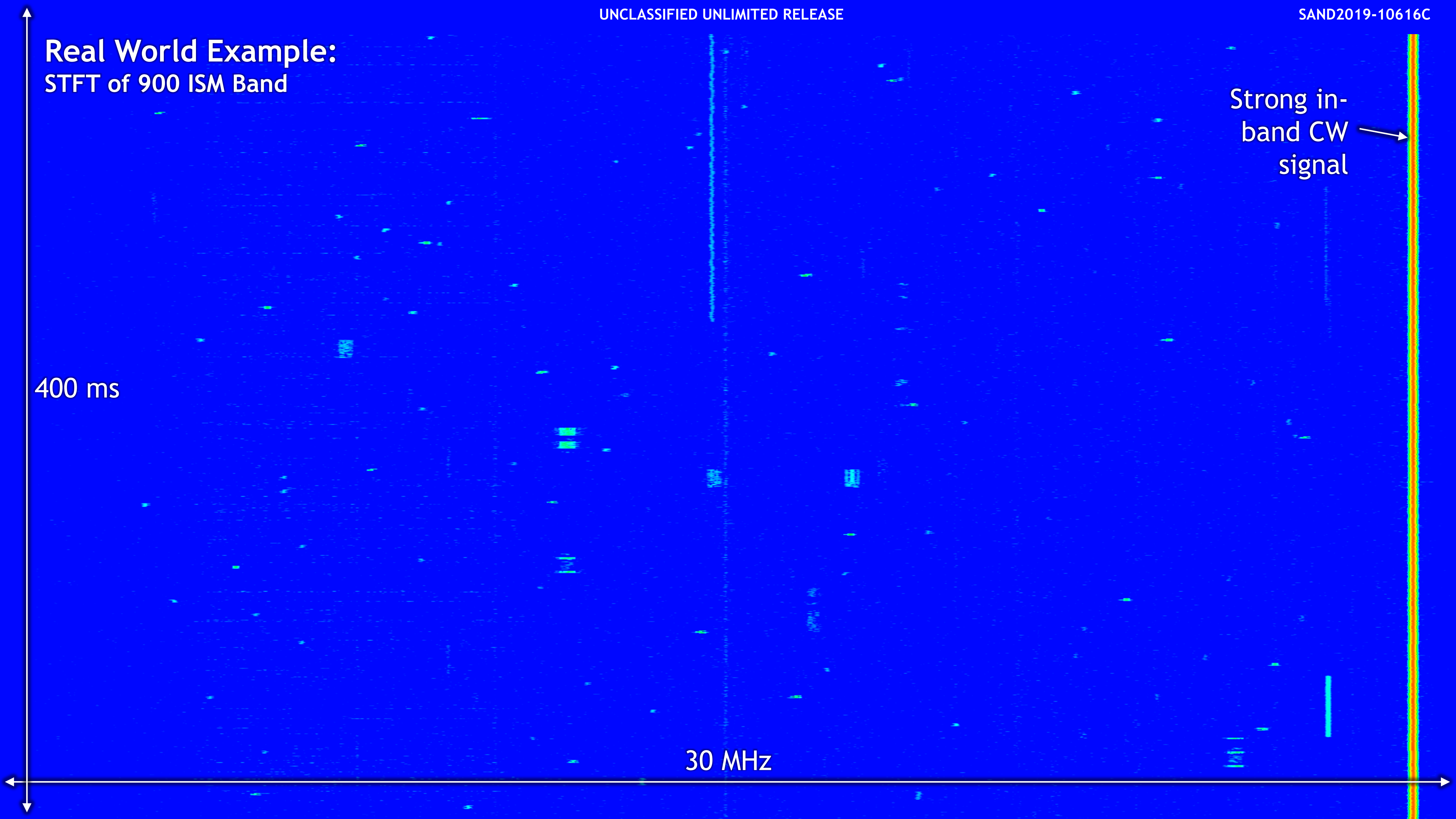
Bursts Tagged in Input Data Stream

Broadband Time-Sliced Bursts

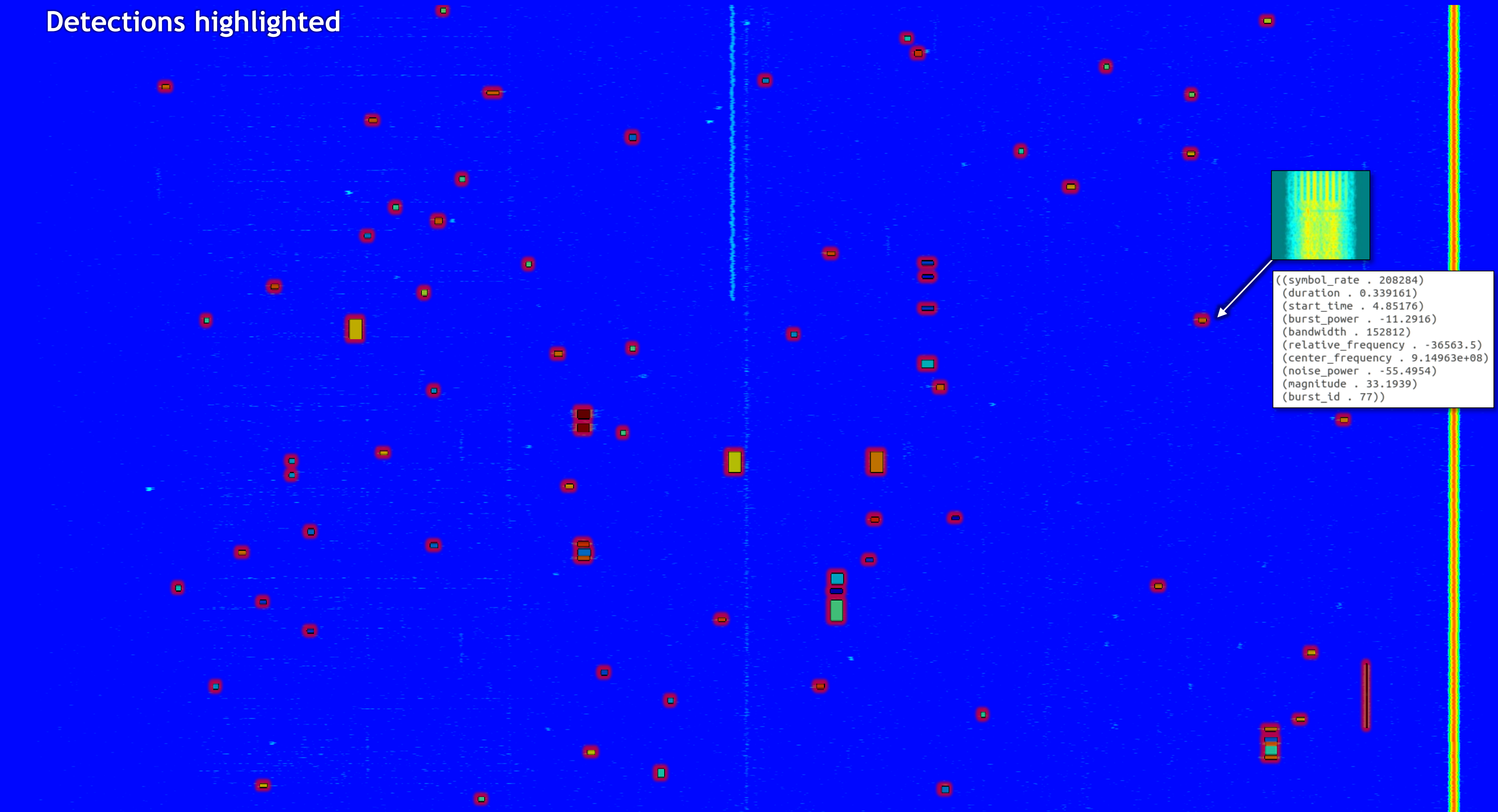
- Built for ISM FHSS receiver applications
  - Bursts of energy on arbitrary frequencies
- Derived from CCC's gr-iridium
  - Similar approach, generalized
  - Efficient implementation, good sensitivity to ~6dB C/N



Real World Example:
STFT of 900 ISM Band



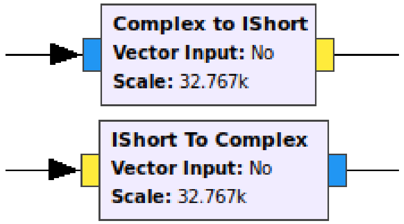
Detections highlighted

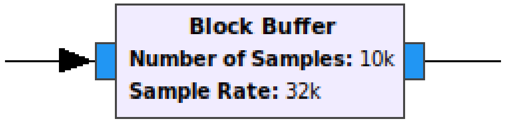


SAND2019-10616C

Sandia Utilities

- Assorted useful blocks that don't fit well into other modules
- Holding area for in-tree improvements we have yet to get released
- Mostly oriented at debug applications
- Usually minor documentation if any
- Blocks are generally not extensively tested…may be dangerous





- Sometimes things get stuck in here before being moved or upstreamed…
  - Useful to install if you are using Sandia's GR Toolkit

[Yet another] gr-sidekiq

- Coming soon…
- Extends many UHD-style features to the Sidekiq hardware
  - DDC / DUCs in FPGA
  - Command queues in FPGA (separate for instantaneous and timed commands)
  - Time tags generated by source, support for TX tag based flow/time/frequency control
- Supports direct PDU-based transmission
- Efficient implementation (thread-per-channel)
- Currently works on the M.2 and VPX2 Sidekiq variants

SAND2019-10616C

Summary

SAND2019-10616C

Takeaways

- PDU's are underutilized, and very useful…With only *a bit* of frustrating PMT behavior…

```
        // OK, here comes the horrible part. Pairs pass is_dict(), but they're not dicts. Such
        // dicks.
        try {
            // This will fail if msg is a pair:
            pmt::pmt_t keys = pmt::dict_keys(msg);
```

- The PDU Utilities help bridge GNU Radio's streaming and PDU APIs
- There are straightforward tools available for developing bursty transceivers within GR
- Significant enhancement to GR's in-tree PDU capabilities are available
- GNU Radio can be used for 'reliable' 'low latency' applications
- The FHSS Utilities module has a robust and efficient arbitrary burst detector and associated signal processing

SAND2019-10616C

Thank you!

Questions?