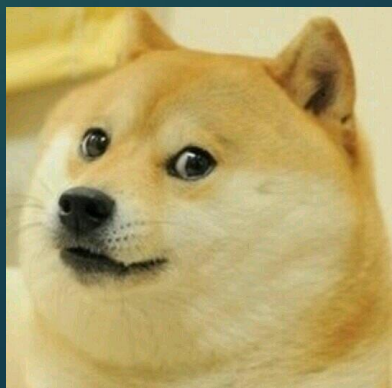
Efficient Processing of Bursty Information Streams with GNU Radio

THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

TIM O'SHEA (TIM.OSHEA753@GMAIL.COM // TIM` @ IRC.FREENODE.NET)

RESEARCH ASSOCIATE, VIRGINIA TECH, ARLINGTON, VA

PRINCIPAL, O'SHEA RESEARCH LLC



Virginia Tech 1872 Invent the Future | Ted and Karyn Hume Center for National Security and Technology

Rapid Overview

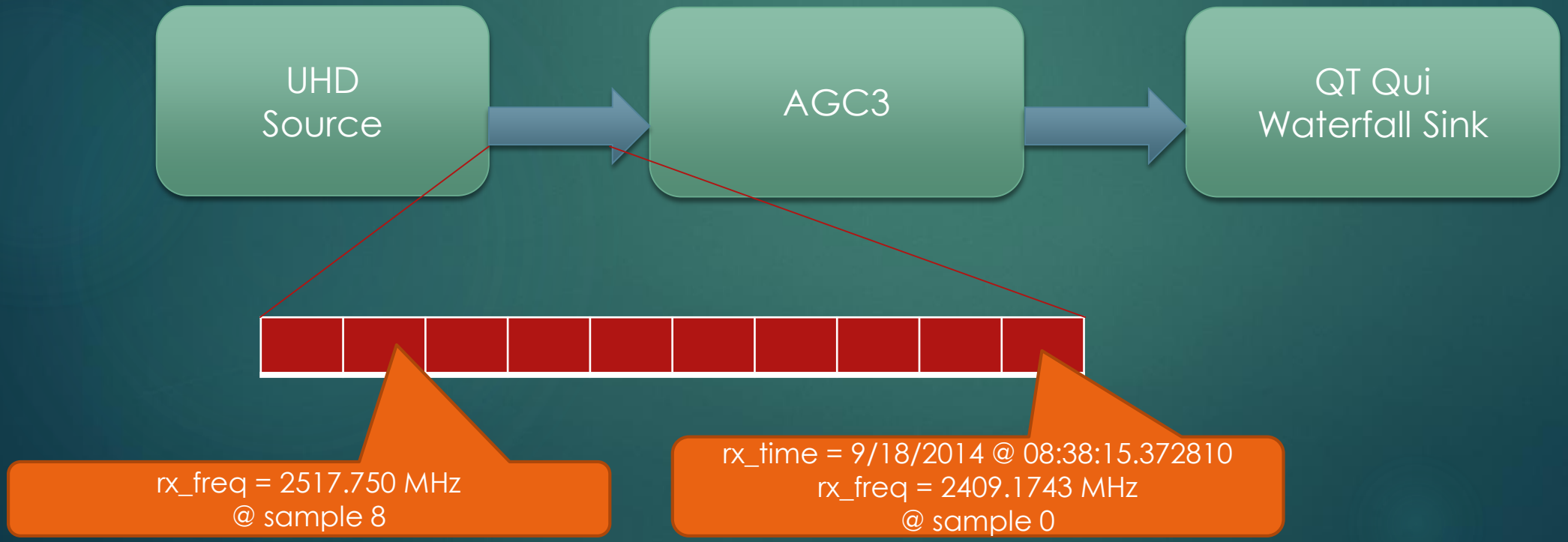
- An Overview of GNU Radio's Processing Model
  - Brief primer on the Stream Tag System
  - Brief primer on GNU Radio's Polymorphic Type
  - The Concept of a PMT PDU
- Burst System Motivation
- Tagged Stream Block (TSB) Based Burst Design
- Message Port (PDU) Based Burst Design
- Burst Design Performance Measurement (gr-chunky)
- Translation Between Modem Design Segments
- FAß Working Group
- PyBOMBS Usage Statistics
- Performance stats: stats.gnuradio.org
- Rapid GPU Blocks with gr-theano

GNU Radio Stream Processing

- We all know and love this
- It runs forever, the only notion of an “Item” is a “float32” (n bytes)

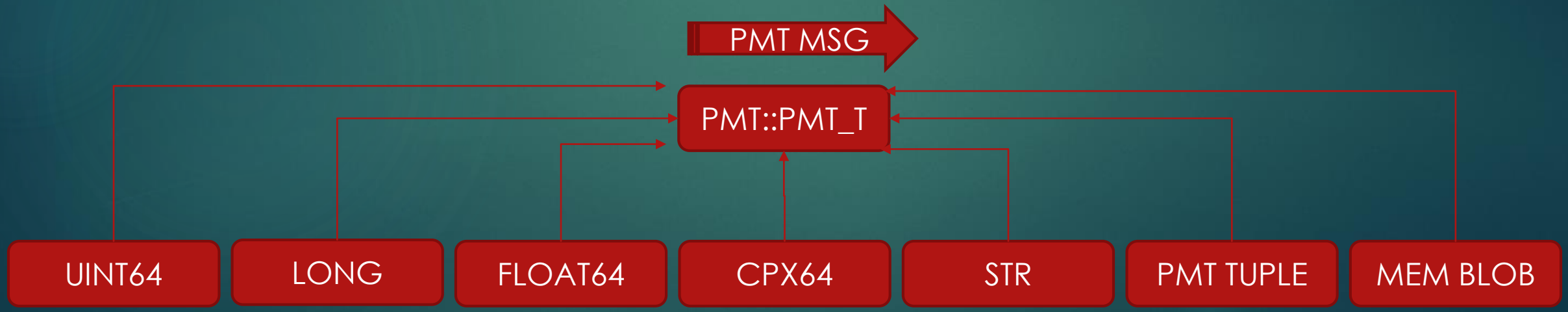
GNU Radio Stream Tags

- Stream tags allow us to annotate events occurring in a stream at precise sample times
- Great for time tagging, synchronously responding to events in streams, etc



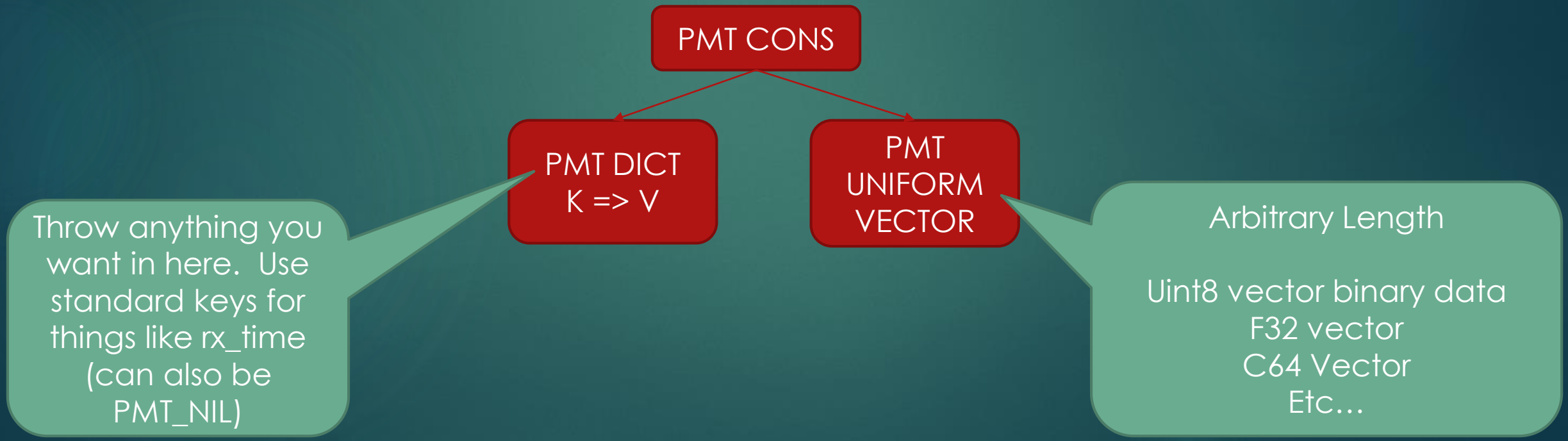
GNU Radio Polymorphic Type (PMT)

- Functional Programming in C++ ? Why Not
- Facilitates Safe Message Passing through several concepts
  - Everything is a PMT :: OO Inheritance
  - PMT's are typically write-once constructs – avoid threading hazards
  - PMT Lifecycles are managed by Boost Shared Pointer ref counting
- Some PMT Examples



The PMT Protocol Data Unit (PDU)

- A "Standard" Interface to carry information between blocks
- A PDU is a Cons with standard Car and Cdr formats
- Car = (pmt::dict or pmt::PMT_NIL), Cdr = (pmt::uvector or list(pmt::uvector))
- A brief story about underdefined message types (i.e. SCA configure/query strings)
- Need to start a "standard" keys wiki page on GR site? Uhd has defined some "Standards" already

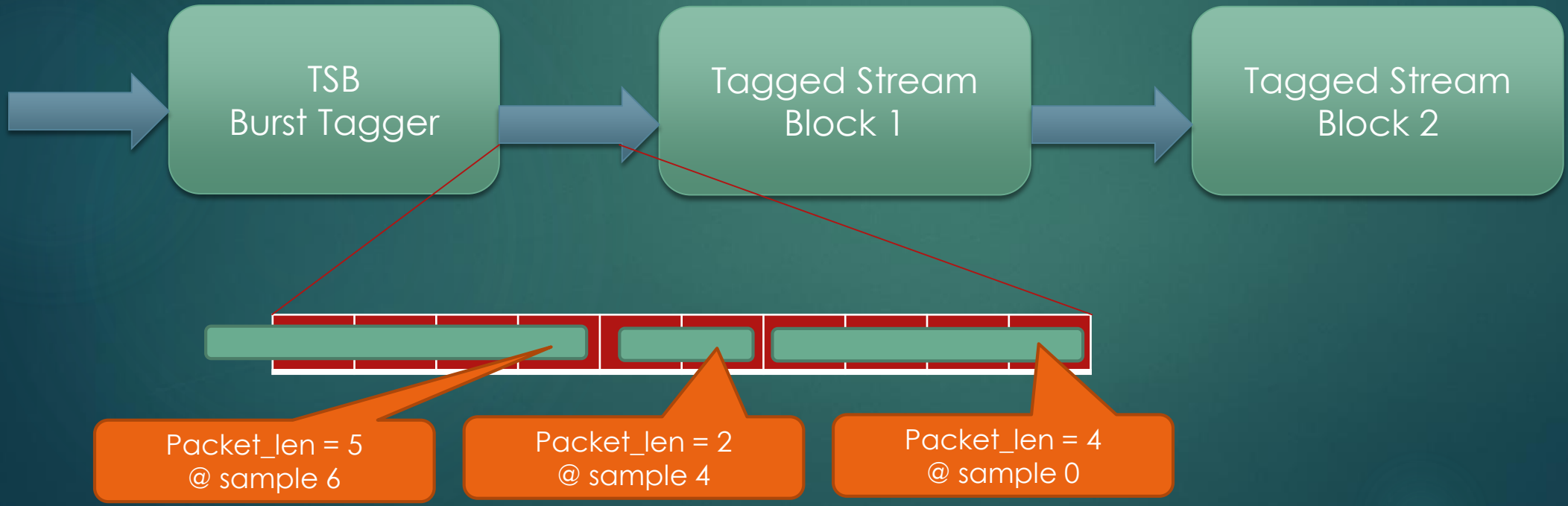


Motivation for Burst System Design

- Most systems today are burst or packet based
  - Systems perform packet based multi-user slot / burst assignment
  - Synchronous reconfiguration required in many systems
- Stopping and starting flow graphs is generally not a good way to reconfigure synchronously
- Using switch blocks typically becomes a nightmare quickly
- Monolithic stream blocks with lots of internal state are not a great solution
  - Can be efficient, but generally sacrifice code-reuse & portability

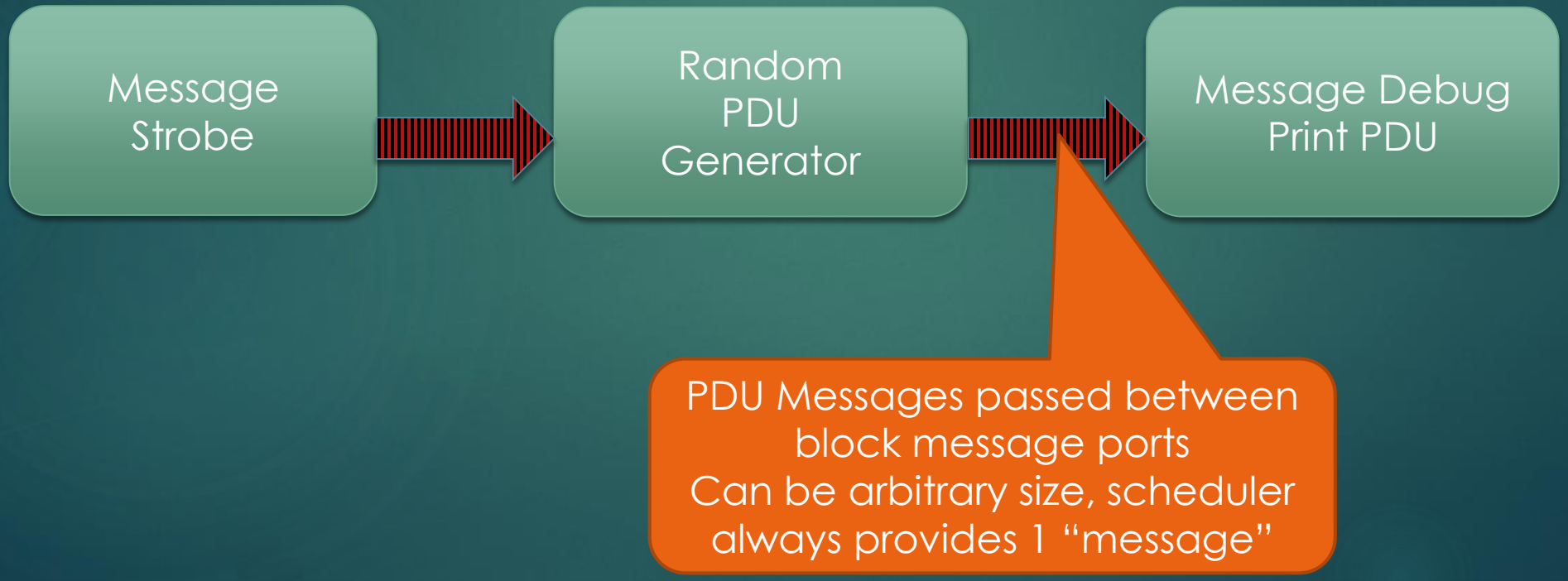
GNU Radio Tagged Stream Block (TSB)

- Traditional streaming mode buffers can be used in conjunction with stream tags to pass "bursts" around between blocks intended to process these "bursts"
- This has the nice side effect that you can often pass them through existing stream blocks without any modification
- The scheduler is largely unaware this is going on which makes it sad

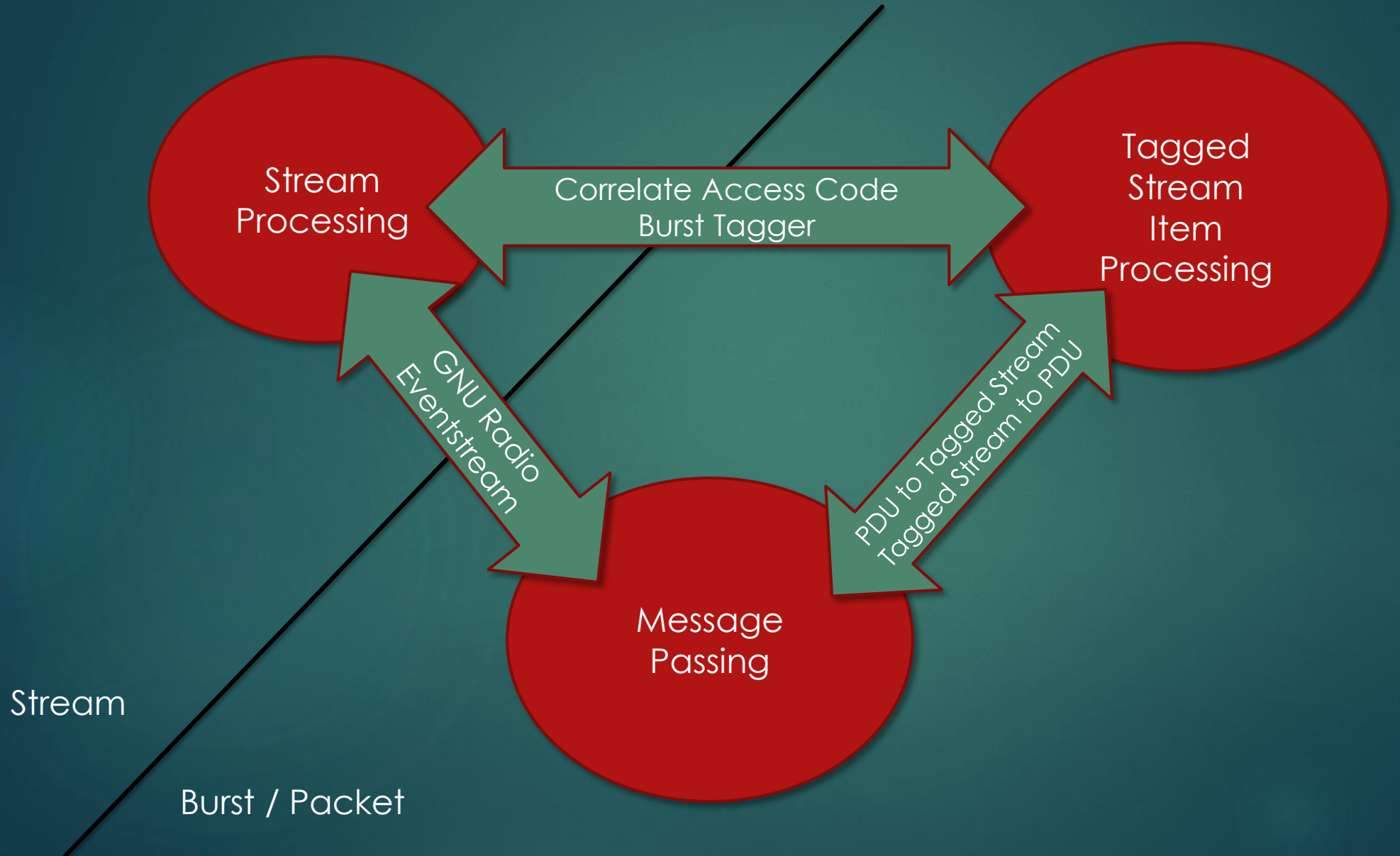


GNU Radio Message Connections (PDU)

- Messages don't have to be PDU format, but that is a handy standard
- Abandon stream ports entirely!!
- Move to message passing model for "burst" portions of a modem

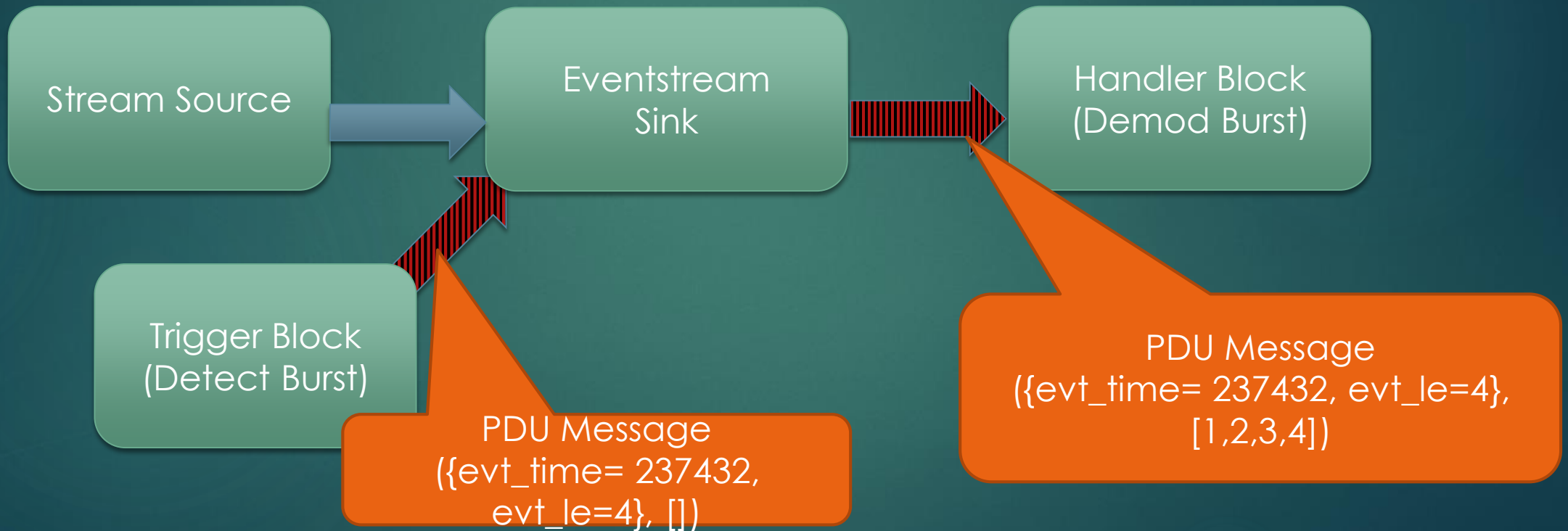


Domain Model Translation Blocks



gr-eventsteam: a brief mention again

- Provides stream to and from message direct translation
- Also on github @ https://github.com/osh/gr-eventstream
- Also provides efficient model for concurrent handler execution

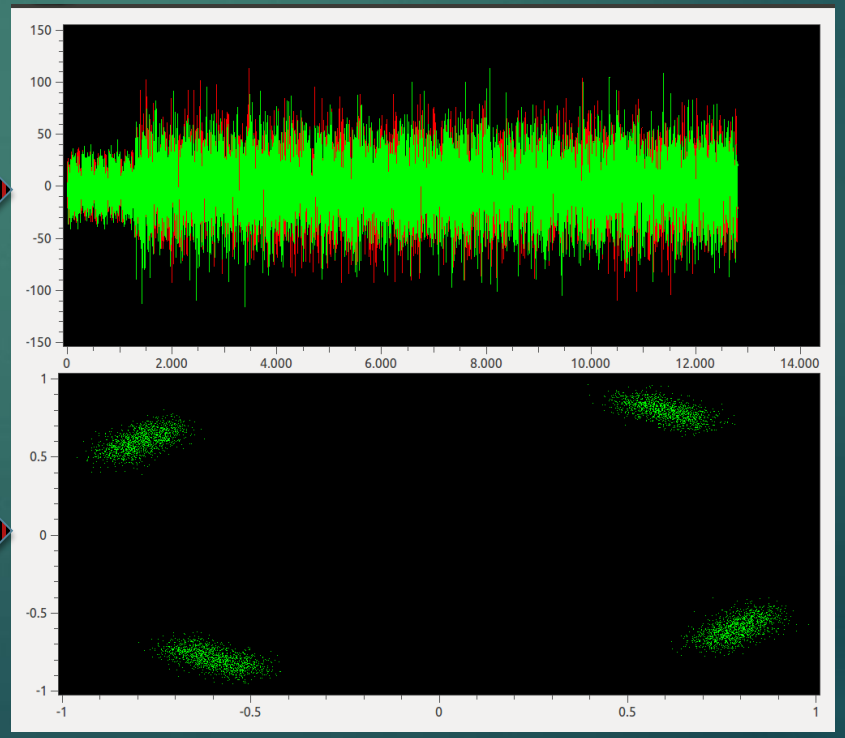


PyQT Burst Plotting Tools (OOT Module)

- To enable debugging of message based burst applications
- Plots one message vector at a time instead of N samples at a time
- Allows you to view one burst, one ofdm symbol, etc at a time
- Available in pybombs and on github @ https://github.com/osh/gr-pyqt

PDU
cons(pmt::dict,
pmt::c32vector)

PDU
cons(pmt::dict,
pmt::c32vector)



Functional Architectures for Signal Processing Systems (FAß)

- Focus on Modem Design Methods in GNU Radio
  - Specifically TSB & PDU based burst design
- Ensure we have the tools necessary to built first class systems
- Improve interoperability and blocks for burst modems
- Find and fix existing problems in TSB and PDU systems
- Work on PDU and TSB performance improvement and characterization
- Enable awesome dynamic systems!

- Stick around for the WG shortly after this talk to discuss this!

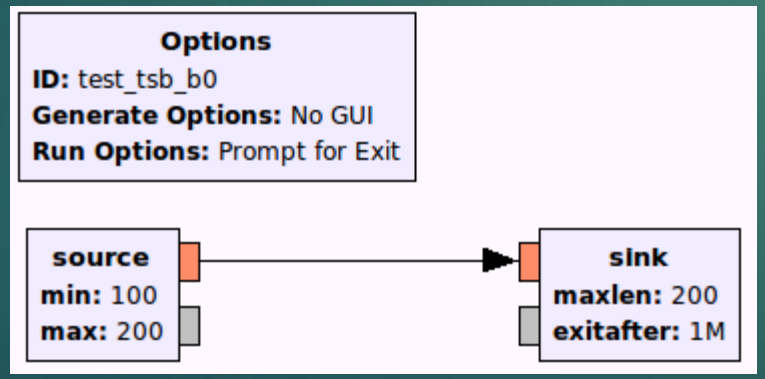
TSB / PDU Design Pros and Cons

	Pros	Cons	Mitigations
Tagged Stream Blocks	• Interoperability / Existing Blocks • Re-use efficient circular buffers	• Scheduler not "burst" aware • Buffer imposed size limitations	• Scheduler Improvements?
Message Passing	• Simpler Implementation • Scheduler is burst-size agonistic	• Memory Allocation / De-allocation overhead • Memory lifecycle tracking overhead • Interoperability	• Message Allocation Pools? • Joint TSB/PDU block base class

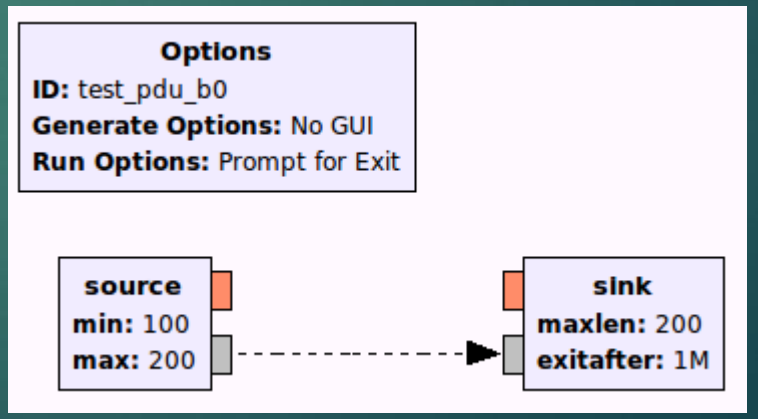
Measuring Overhead in Burst Systems

- Start with several simple flowgraphs (~1.8 femtobalints each)
- Let them run unthrottled and measure max throughput and latency
- Both use the "same" work function for pdus and message ports

- Warning: These were conducted with 100-200 item packets, there are lots of variables in this benchmark and this represents one test … not a conclusive result yet.

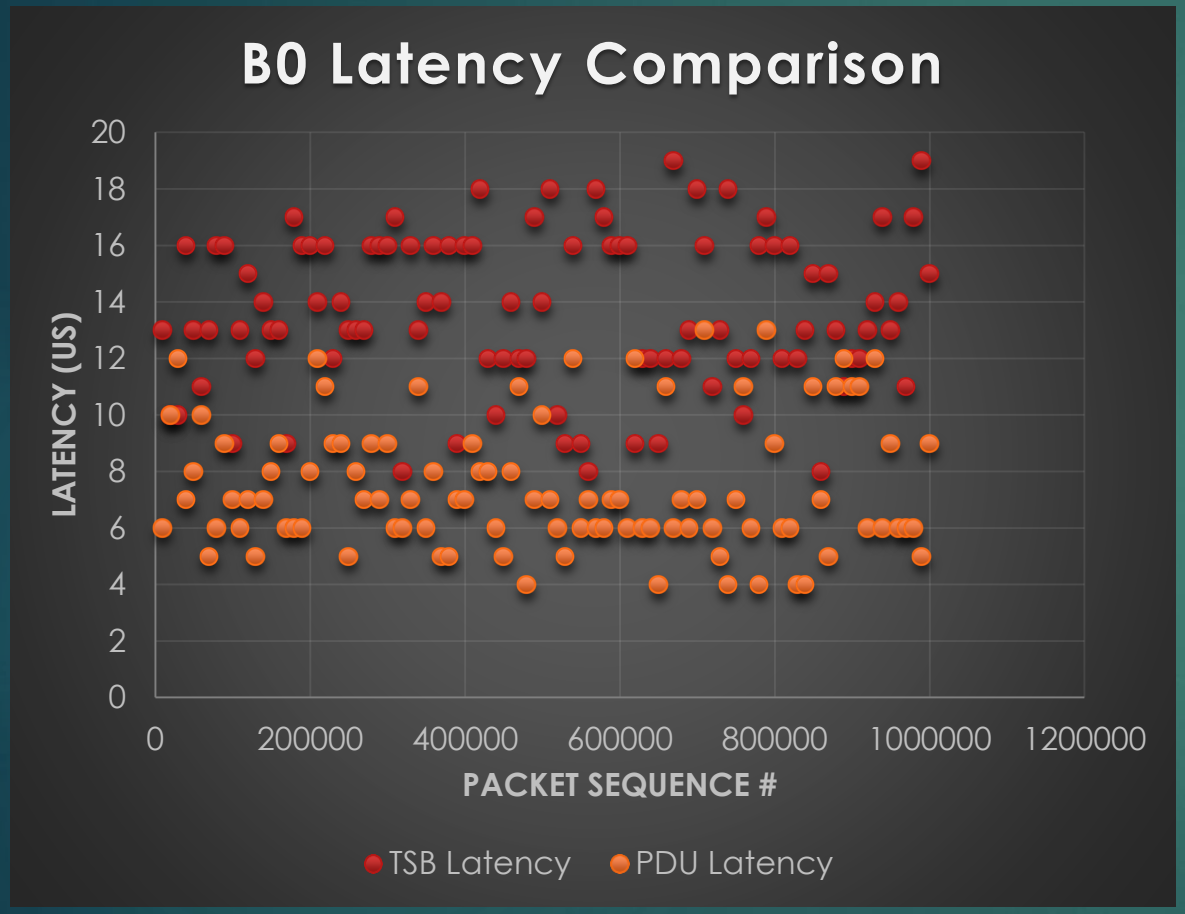


TSB 0 Blocks (B0)

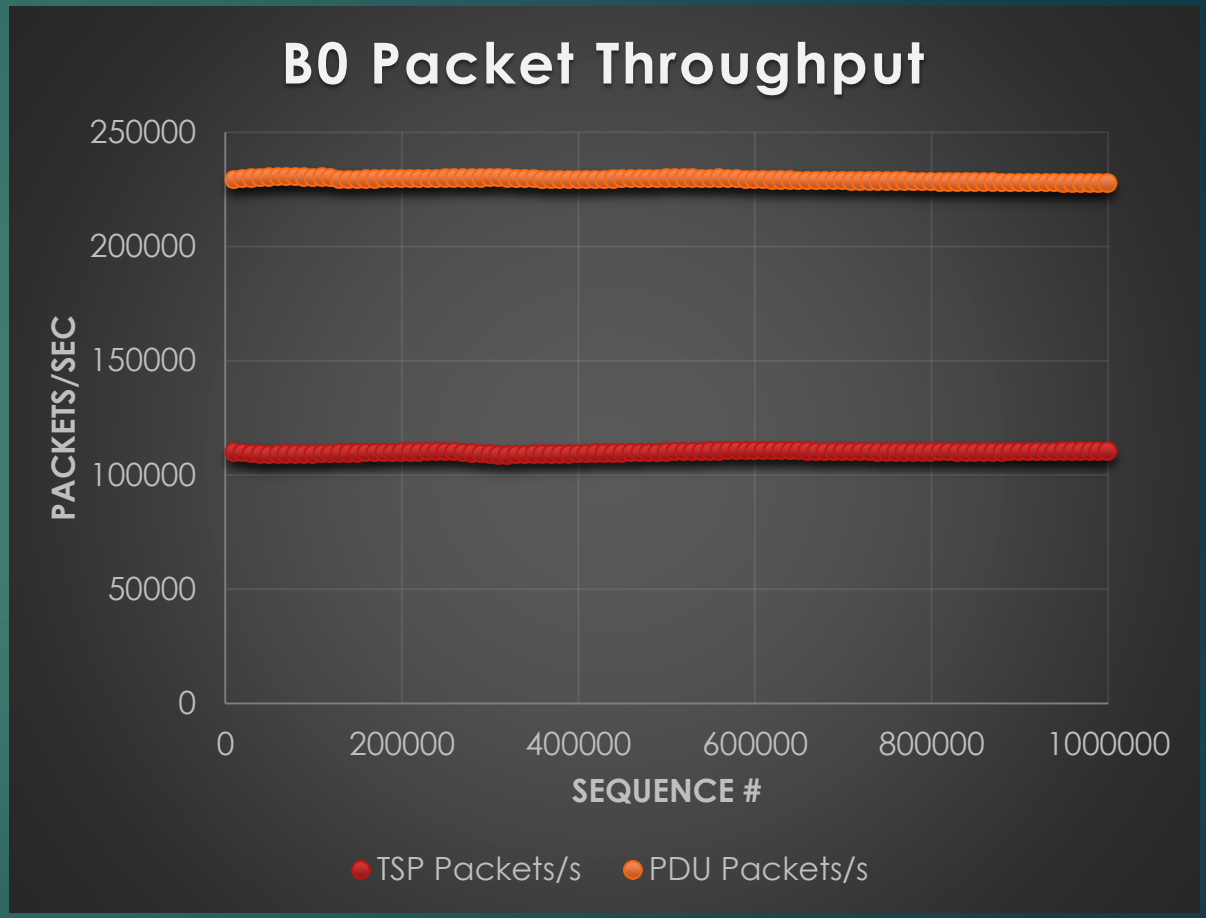


PDU 0 Blocks (B0)

Measuring Overhead in Burst Systems



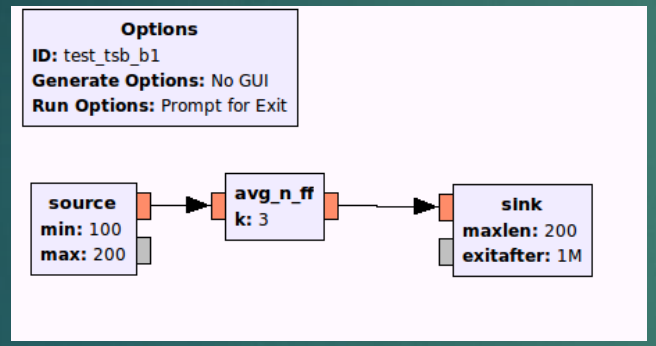
TSB 0 Blocks (B0)



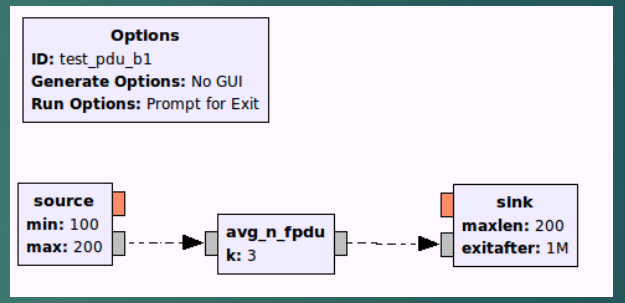
PDU 0 Blocks (B0)

Measuring Overhead in Burst Systems

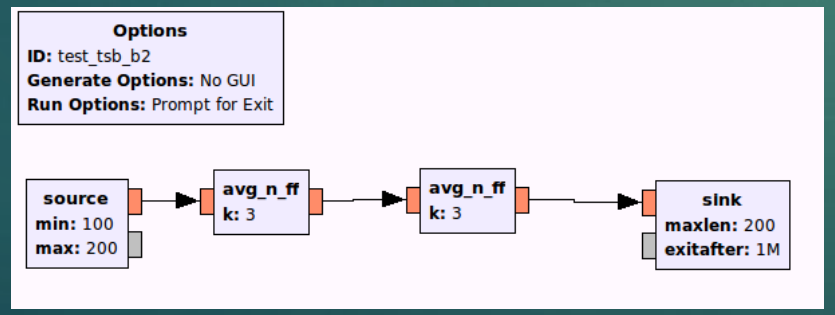
- Similar B1 and B2 tests, measure latency and throughput through simple burst processing blocks (function is a block average which shortens packets by averaging many little chunks together in this case … operation is identical for PDU and TSB implementations.)



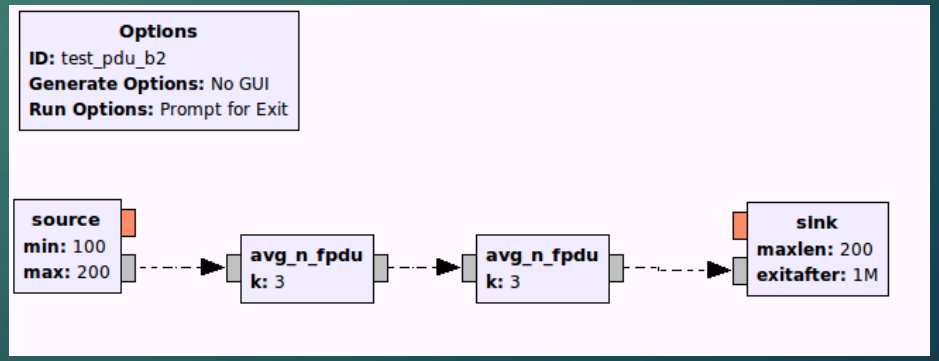
TSB 1 Blocks (B1)



PDU 1 Blocks (B1)

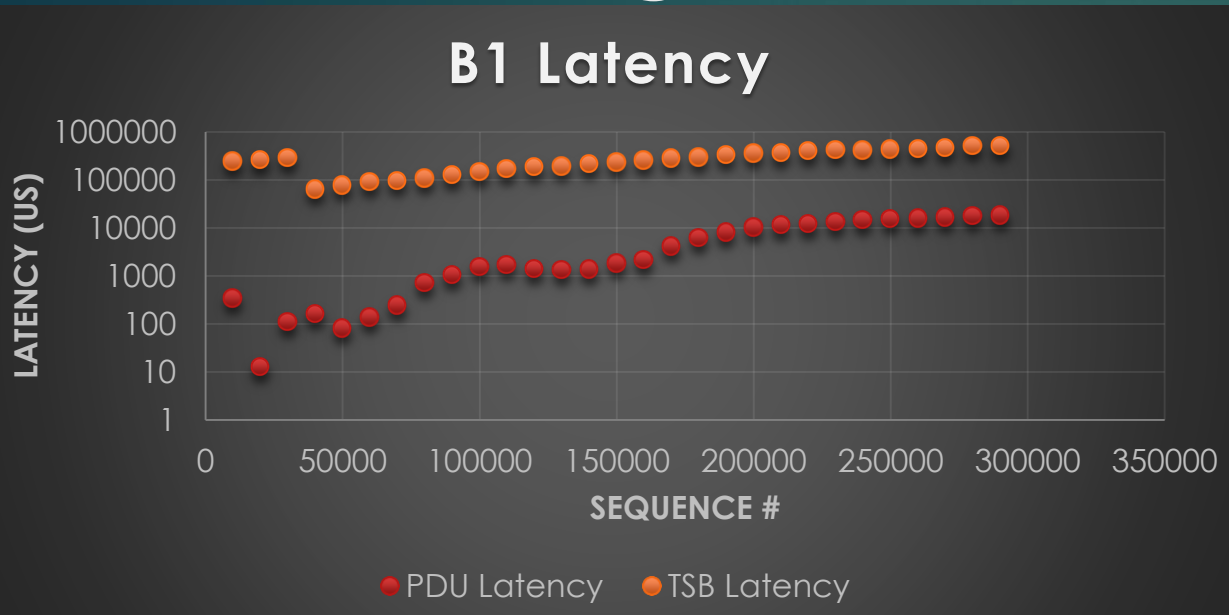


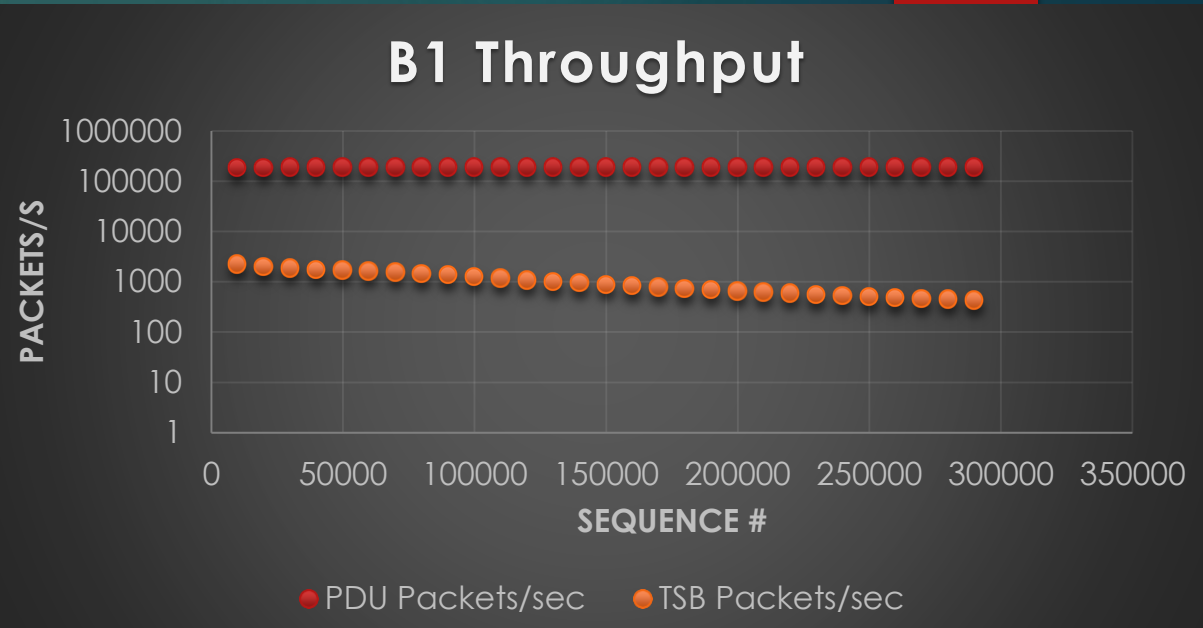
TSB 2 Blocks (B2)

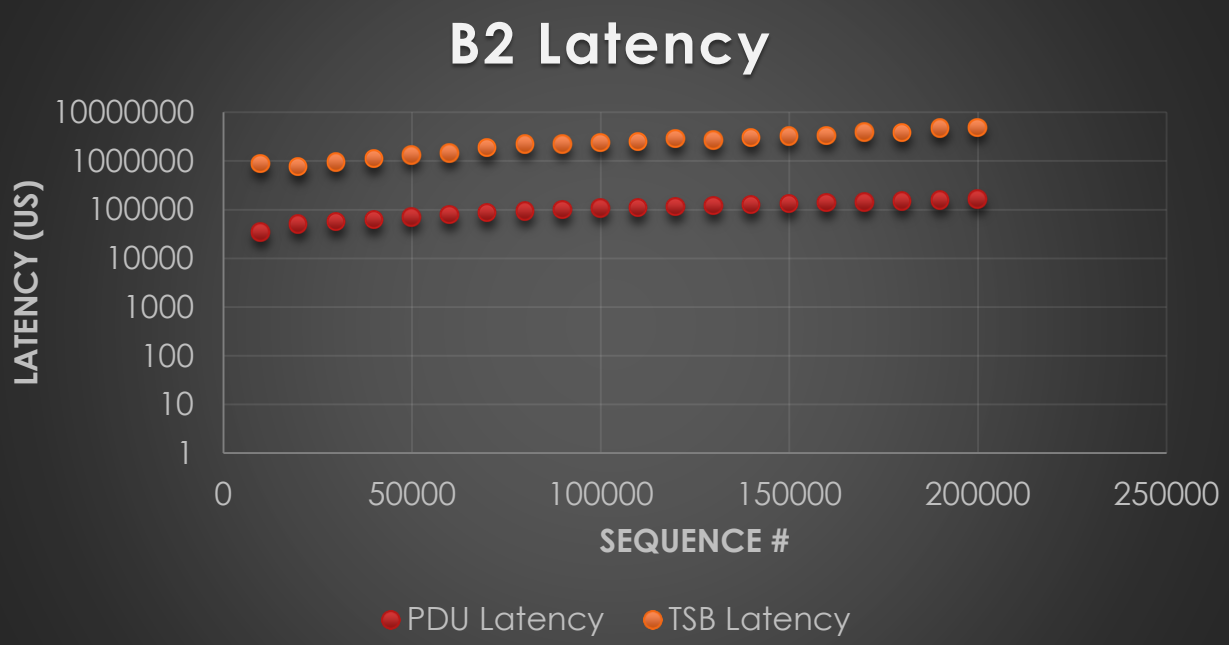


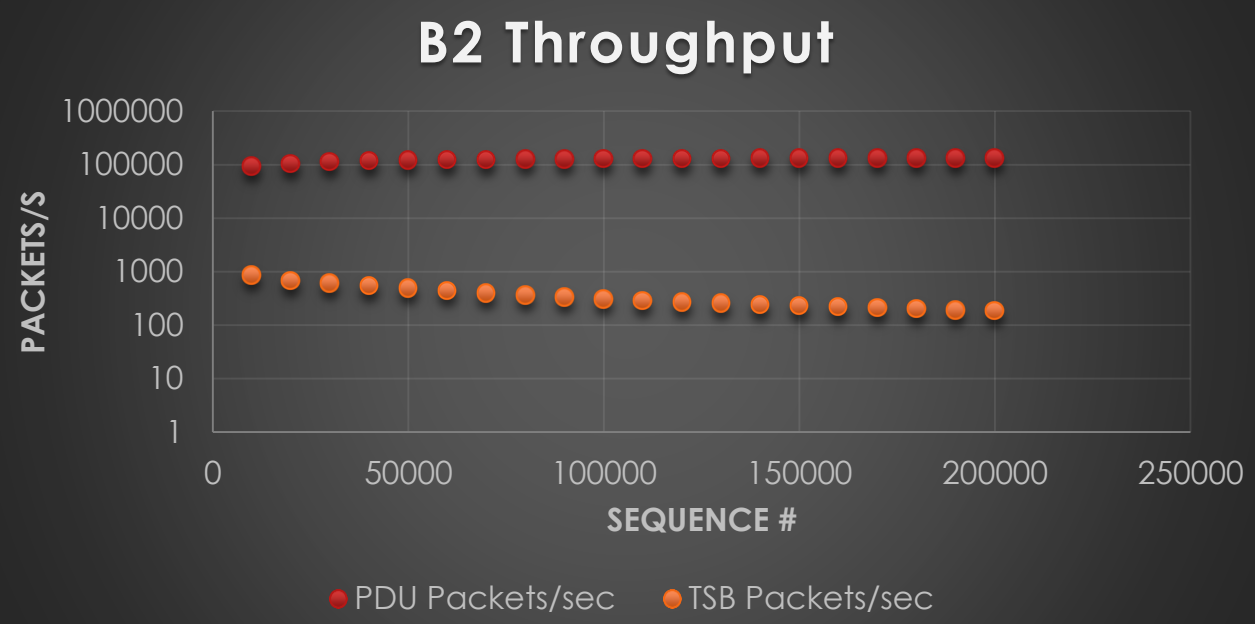
PDU 2 Blocks (B2)

Measuring Overhead in Burst Systems







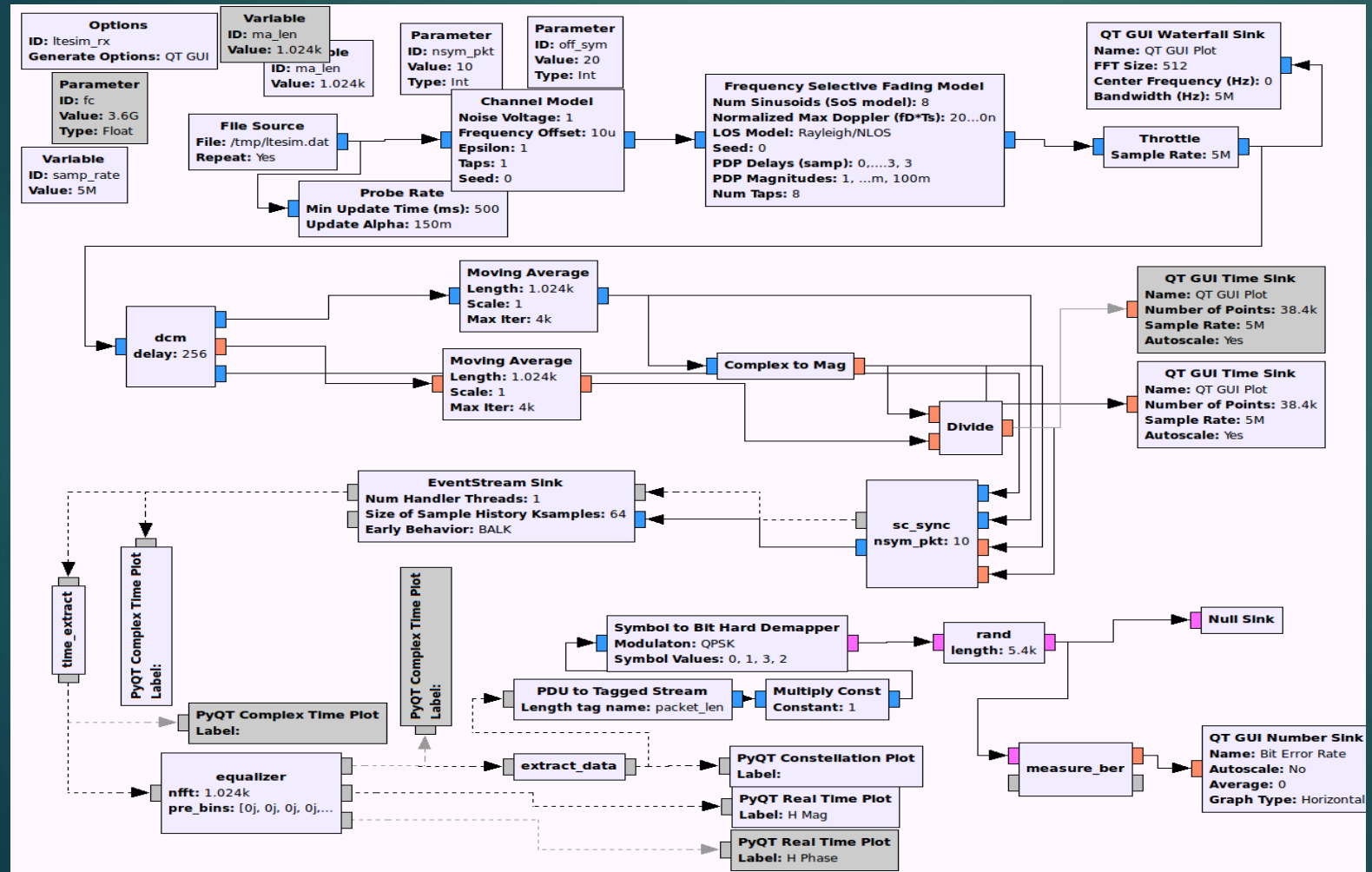


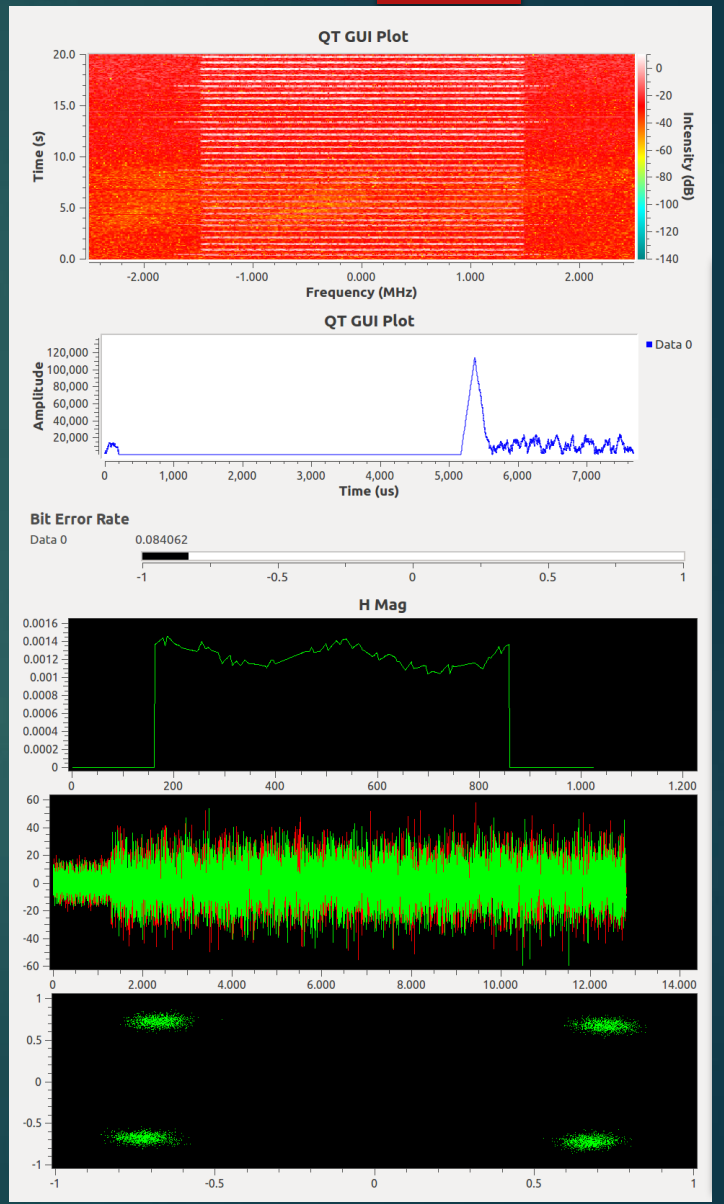
Measuring Overhead in Burst Systems

- Conclusions:
  - More measurement is required
  - PDU performance seems to be far better than some have speculated …
- Next steps
  - Third party verification (code is on github @ https://github.com/osh/gr-chunky)
  - Automate and standardize measurement process
  - Ensure all blocks are best-case optimized
  - Common base class for TSB/PDU processing blocks
  - Measurement on alternative platforms (i.e. Zynq)

GR-OFDMA Message Based Modem

- Now lets build some modems with message connections!
- Almost posted on github … soon





http://pybombs.info

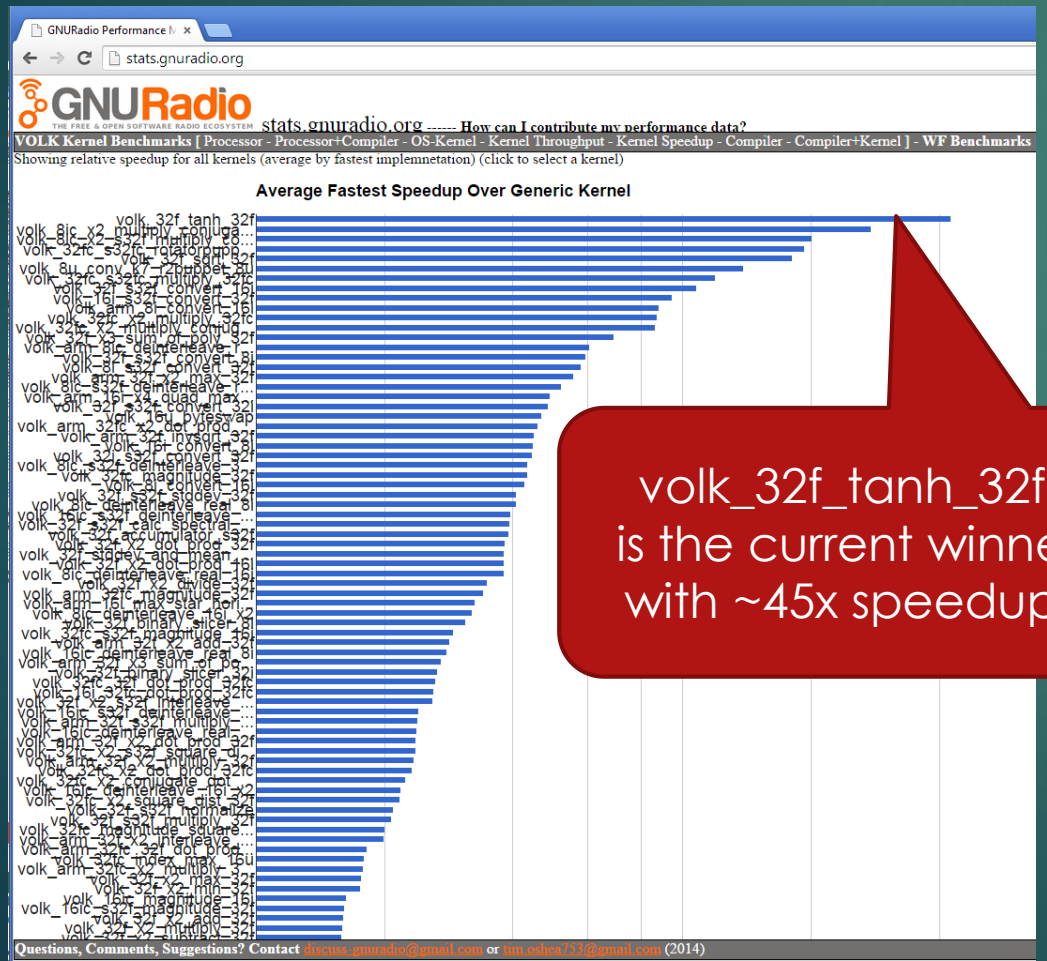
Handy new URL …

Fancier “app store”
Does anyone actually use this?

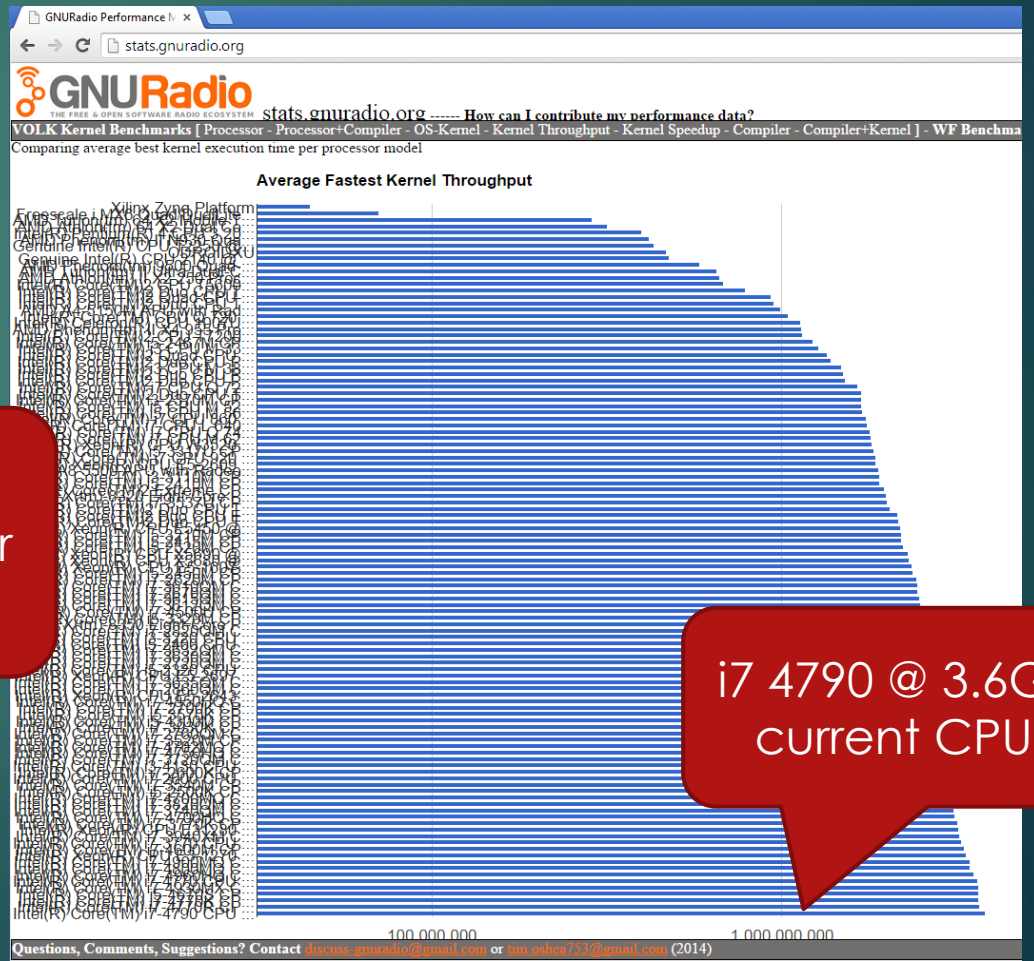
Please send github pull requests for new recipes to add!

http://stats.gnuradio.org

- Summarized VOLK kernel and waveform benchmark performance
- Processing performance comparisons
- Needs some help from someone good with web visualization



volk_32f_tanh_32f is the current winner with ~45x speedup



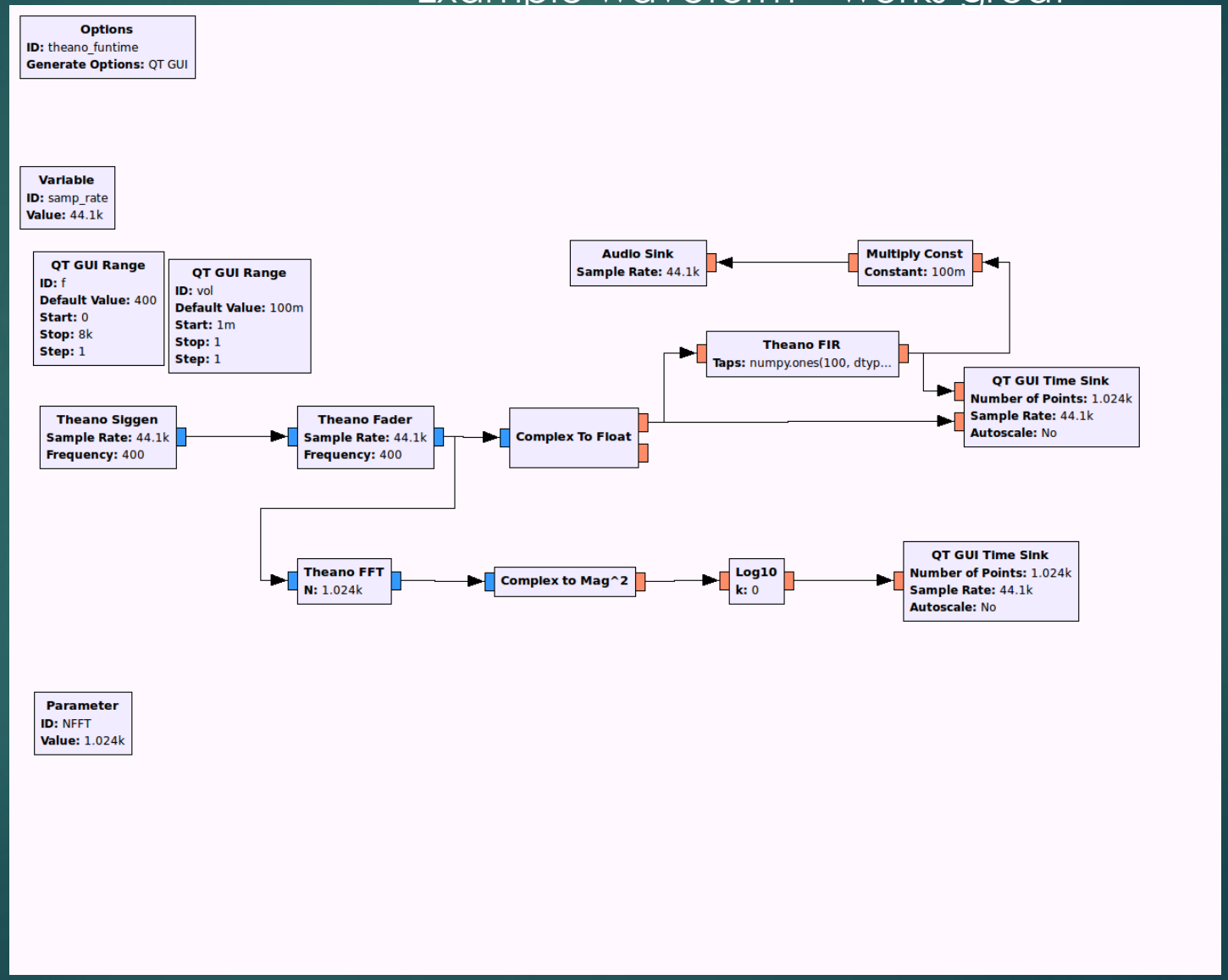
i7 4790 @ 3.6GHz is the current CPU leader

gr-theano

- Theano is a stand alone python library used for accelerating large matrix mathematical calculations on GPU cards
- Heavily used in the Machine Learning community
- OpenCL and CUDA backends (CUDA is the most mature for now)
- Optimizes numpy-like python vector operations and compiles them to kernels for you, manages the data transport for you
- Great for building a monolithic algorithm block – still problematic for chaining gpu-blocks together
- Makes writing python-based gpu blocks insanely easy
- Online benchmarks claim 20x speedup on GPU for some algorithms
- Gr-Theano still needs benchmarking …
- Gr-Theano is on github @ https://github.com/osh/gr-theano

gr-theano

Example waveform – works great



gr-theano

Two handy example blocks …. So concise, wow

```
class fir(gr.sync_block):
    x = T.matrix("x")

    def set_taps(self, taps):
        print "set_taps"
        self.b = theano.shared(numpy.vstack([taps]), name="b")
        self.set_history(taps.size)

    def __init__(self, taps):
        gr.sync_block.__init__(self,
            name="theano_fir",
            in_sig=[numpy.float32],
            out_sig=[numpy.float32])
        self.set_taps(taps)

        self.f = theano.function(
            inputs = [self.x],
            outputs=[T.signal.conv.conv2d(self.x,self.b)],
            updates={},
            name ="f")

    def work(self, input_items, output_items):
        out = output_items[0]
        o = self.f( numpy.vstack([ input_items[0] ]) );
        out[:] = o[0][0,:];
        return len(output_items[0])
```

```
class fft(gr.sync_block):
    def __init__(self, N):
        gr.sync_block.__init__(self,
            name="theano_fft",
            in_sig=[numpy.complex64],
            out_sig=[numpy.complex64])

        self.set_output_multiple(N);
        self.N = N
        x = T.cmatrix("x")
        w = theano.shared(numpy.ones(self.N,
dtype="complex64"), name="w")
        self.f = theano.function(
            inputs=[x],
            outputs=[T.fourier.fft(x*w, n=N, axis=1)],
            updates={},
            name = "f")

    def work(self, input_items, output_items):
        n = len(input_items[0])/self.N
        for i in range(0,n):
            inmat =
numpy.vstack([input_items[0][self.N*i:self.N*(i+1)]]);
            omat = self.f(inmat);
            output_items[0][self.N*i:self.N*(i+1)] = omat[0];
        return len(output_items[0])
```

That’s it for this year