Introduction To Writing Signal Processing Applications In Python Using GNURadio

What is it?

What is built with it?

How to build something with it!

Outline

- What is GNURadio
- Building a block
- Connecting Blocks
- Packaging and deploying

What GNURadio Provides

- Framework
  - Currently C++ and Python
- Development tools
  - Tools to packages and interface your code with others
- Scheduler
  - Use all the cores
- Interface and abstraction of hardware
  - Both radio and some acceleration
- Library of DSP
- Graphical interface / GNURadio Companion

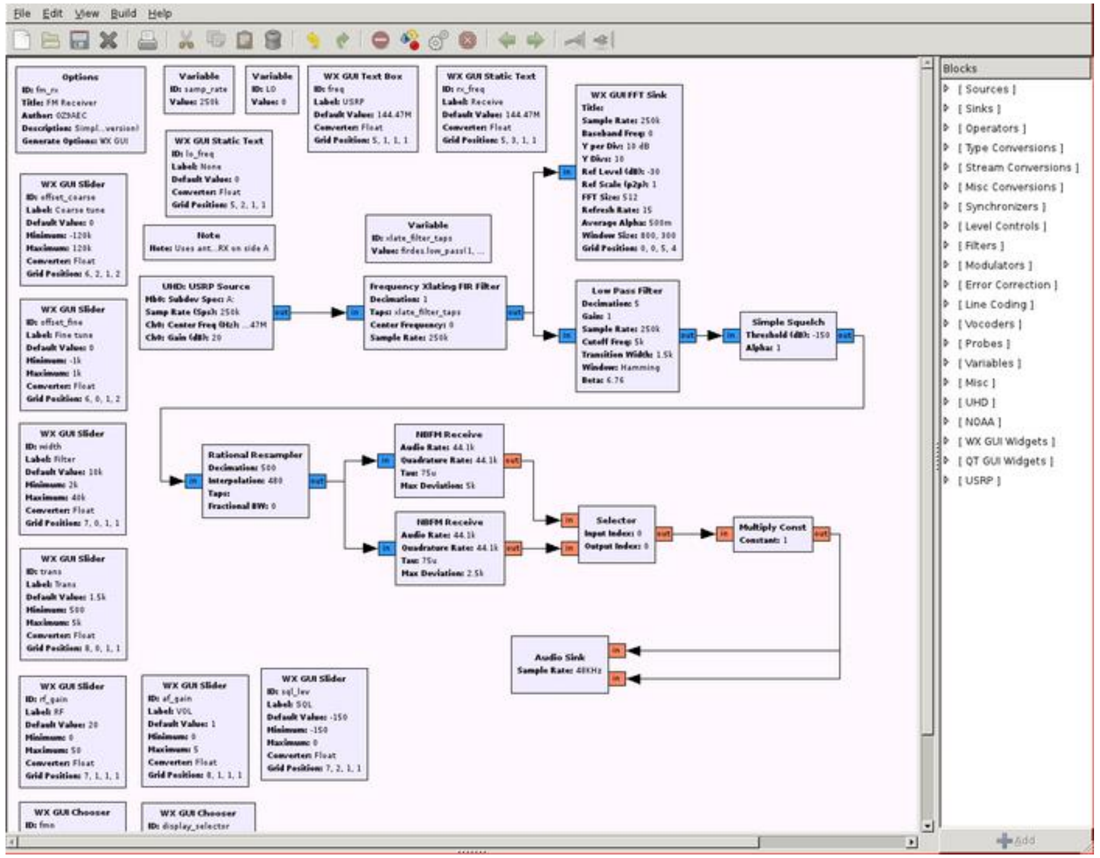
What is GNURadio

- Collection of stuff needed for software defined radio
  - Also applies to other signal processing applications
  - Collection of functions and algorithms
  - Standard means of interacting with the outside
  - Many of the things you don’t want to write
- Things build with GNURadio
  - gr-specest
  - gr-lte
  - gr-radar
  - gr-sigmf
  - gr-air-modes
  - gr-limesdr
  - gr-pdu_utils
  - gr-satellites
  - gr-iio
  - gr-doa
  - gr-fosphor

Virtual GRCon20

Graphical interface / GNURadio Companion

- Most recognized feature of GNURadio
  - Maybe least useful
    - Just an opinion
- Provides graphical means of connection operations
  - Color shows type
- Makes descriptive graphics
- Not used in talk



Virtual GRCon20

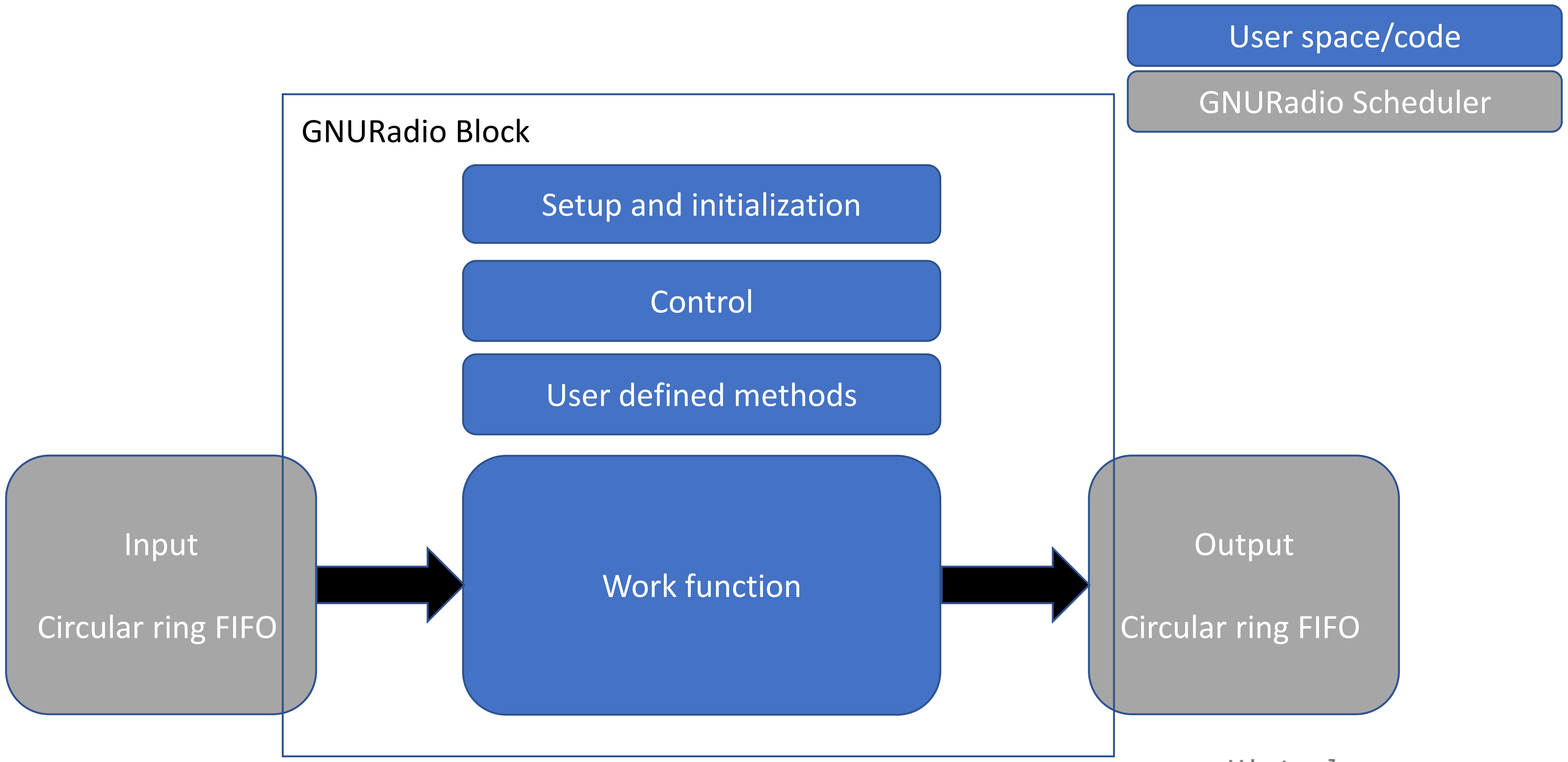
Library of DSP

- Gr-filter
  - Filter stuff
- Gr-digital
  - Modulation, demodulation
- Gr-analog
  - AGC, AM, FM...
- Gr-fec
  - Error correction
- Gr-vocoder
  - Vocoders
- Gr-channel
  - Channel simulation
- Gr-fft
  - FFT
- Volk
  - Vector optimized library of kernels

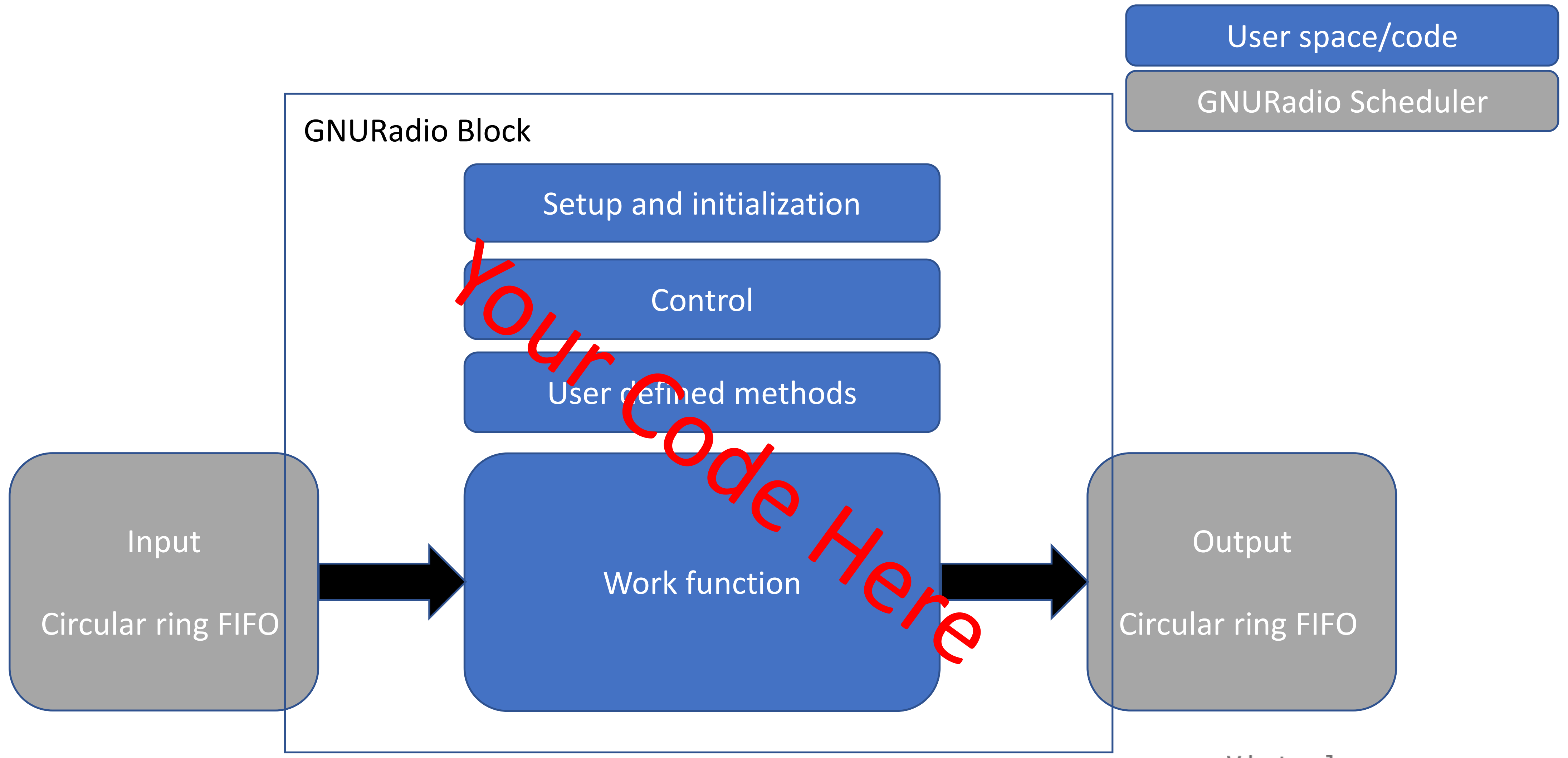
Let's Build a Block

Think of some piece of code you want in a block

What is a block?



What is a block?



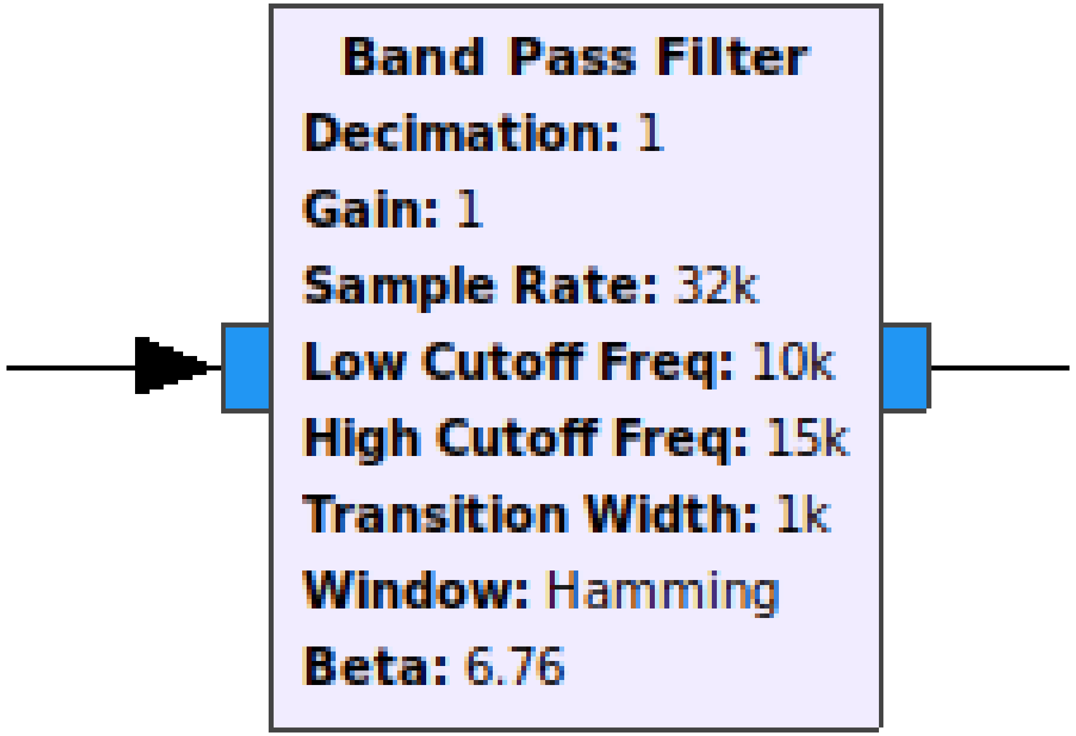
Virtual GRCon20

Types of blocks

- Sync
  - Synchronous, one in one out
- Source
  - Where the data comes from
- Sink
  - Where the data goes
- Interpolator
  - One in many out
- Tagged stream
  - Labeled in labeled out
- General
  - Something in maybe something else out
- Decimator
  - Many in one out
- Hier
  - Blocks in blocks

What does a block need?

- What does a block needs
  - Setup
    - constructor or __init__
  - Work
    - Code to do the task
  - Schedule
    - For data in how much comes out
- Inherits from base class
- Not needed but nice to have
  - Graphical description for GRC
    - Was XML, +3.8.0 uses YAML
  - Works with tags
  - Unit testing and profiling
  - Version control


Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 32k
Low Cutoff Freq: 10k
High Cutoff Freq: 15k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Example Empty Block

```
form gnuradio import gr

class MyBlock(gr.sync_block):
    def __init__(self, *args, **kwargs):
         gr.sync_block.__init__(self,
             in_sig=[complex],
             out_sig=[complex])
         # Add constructor stuff
    def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        out[:] = in0
        # Add signal processing
        Return len(out)
```

Constructor and Initializer

- Builds the block
- Inherits from one of the block types
  - Source, Sink, Sync, Interpolator, Decimator, Tagged stream, Hier
  - Must call parent constructor/initializer
- Executes once
  - Contains user's setup code

```
def __init__(self, *args, **kwargs):
        gr.sync_block.__init__(self,
            in_sig=[complex],
            out_sig=[complex])
        # Add constructor stuff
```

Work Function

- Where the work happens
- Executes are on each buffered chuck of data
  - Called multiple times
- Maps buffers as input arguments
  - input_items, output_items
  - Each input is a list of buffers
  - Python buffer protocol
  - Numpy array
- Returns the number of items written to the output buffer

```
def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        out[:] = in0
        return len(out)
```

Building with Blocks

Assembling your application with blocks

Virtual GRCon20

Top block

- Provides common interface to a signal processing chain
  - How the schedular interacts with your code
- Contains all the blocks
- Distributes across multiple processes and cores
- Graphical tools produce sub-class of top block

- Implementation uses inheritance
- Provides methods
  - Start
    - Gets the data moving
  - Run
    - Gets the data moving and waits for it to finish
  - Wait
    - Waits for it to finish
  - Stop
    - Stops the movement of data

Virtual GRCon20

Connecting the blocks

```
class FlowGraph(gr.top_block):
  def __init__(self):
    gr.top_block.__init__(self, name="Flow graph")
    # Creating the blocks
    self.block1 = Block1(...)
    self.block2 = Block2(...)
    # Connecting the blocks
    self.connect((self.block1, 0), (self.block2, 0))
    # Connect takes 2 tuples of the block instance and the number
```

Note on Types and Vectors

- Types
  - Complex float 32
  - Float 32
  - Int 32
  - Short 16
  - int 8 / byte
- Vectors
  - Vectors of samples

- Currently no explicit support for heterogenous types or structures
  - easily be worked around

Starting and Stopping the Process

- Starting the top block
  - Start method
    - None blocking
  - Run method
    - Blocking

- Stopping the top block
  - Call the stop method on the top block
  - Blocks work functions returns a negative number
  - Raise an exception

Questions ?

Comments

Concerns

Complaints

Accusations

Allegations

The End

Thank you for your time

And thanks to the following people for helping

Richard Gutierrez, Marissa Navarro, Derek Kozel

Missing from this talk

- Reconfiguration
- Packaging and deployment
- Tagged streams
- Polymorphic types
- Vectors
- Libraries of existing components
- MAC Layer
- Proper unit testing
- GRC and GUI
- Messaging
- C++
- Building useful things
- Hardware
- Simulation

Packaging everything into an OOT

Out Of Tree package

What is OOT

- Standard way of packaging code
  - Directory structure
- Build tools
  - Cmake, make
- Testing tools
  - Ctest, python nose, CppUnit
- Install tools
  - Make install

Build, Test, Install, Deploy, Reuse

- Build
  - $ mkdir ./build
  - $ cd ./build
  - $ cmake ../
  - $ make
- Test
  - $ make test
- Install
  - $ make install

Graphical part

- GNURadio 3.8 forward
  - YAML file describing
    - Input and output ports
    - Graphical appearance
    - Construction arguments
    - Documentation
- GNURadio 3.7
  - XML

Outline

- What is GNURadio
- Building a block
- Connecting Blocks
- Packaging and deploying