JIVE
Joint Institute for VLBI
ERIC

CAMRAS * DWINGELOO * TELESCOOP *

# Getting the most out of your FFT

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N}kn}$$

Paul Boven - PE1NUT
p.boven@xs4all.nl

# Goals

- **Spectral Line (e.g. 21cm) observing**
  - **Detect weak, very distant sources**
  - **Determine their frequency, line profile**

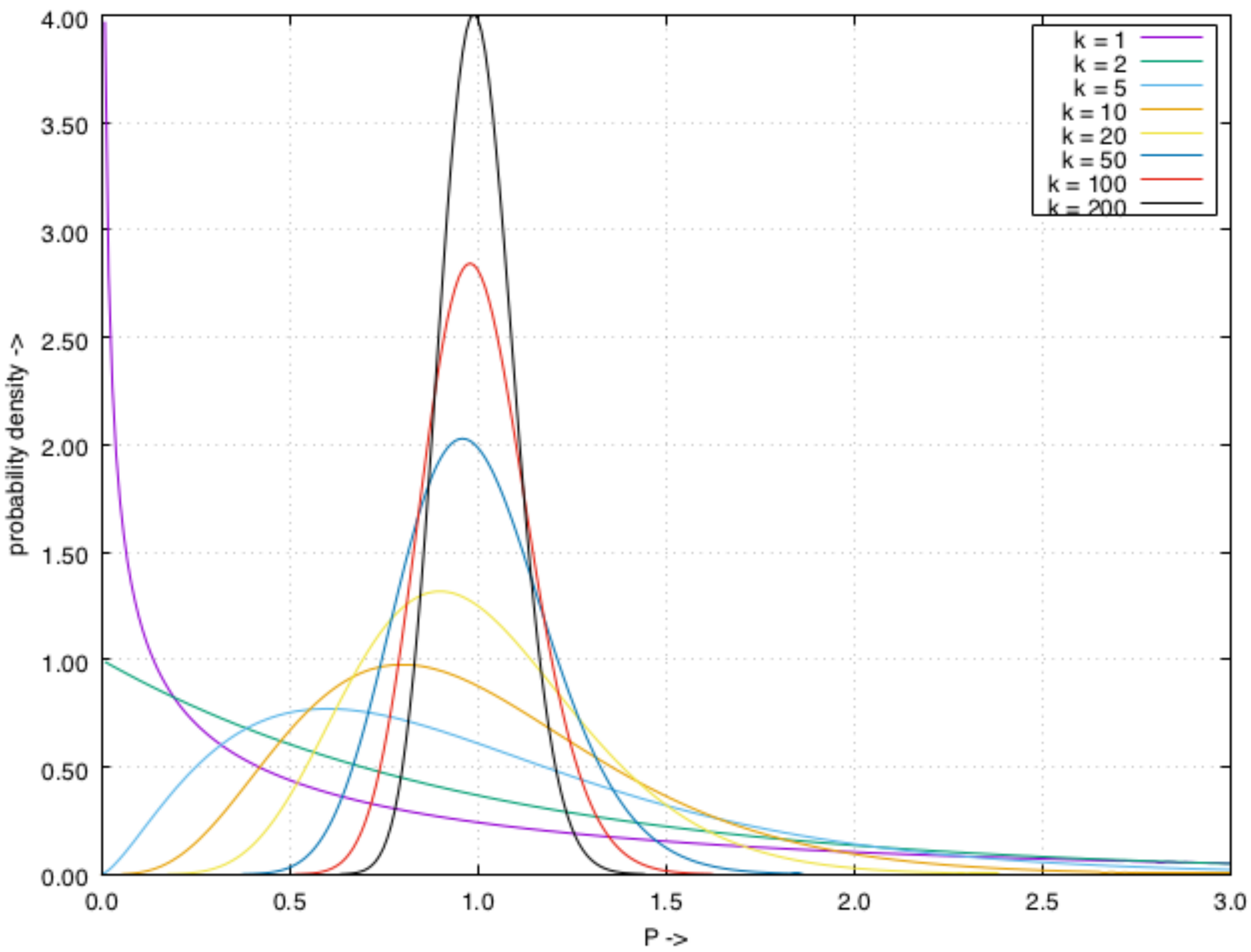
**Requires:**

- **High Sensitivity**
- **Sharp Frequency Bins**
- **Low Computational Effort**

# Measuring Noise Power

- Input: Gaussian Noise
- Square each sample, scales with power
- Average the power measurements
- For large N, uncertainty scales with $1/\sqrt{N}$

- $P = U \cdot I$
- $U = I \cdot R$
- $P = U^2 / R$

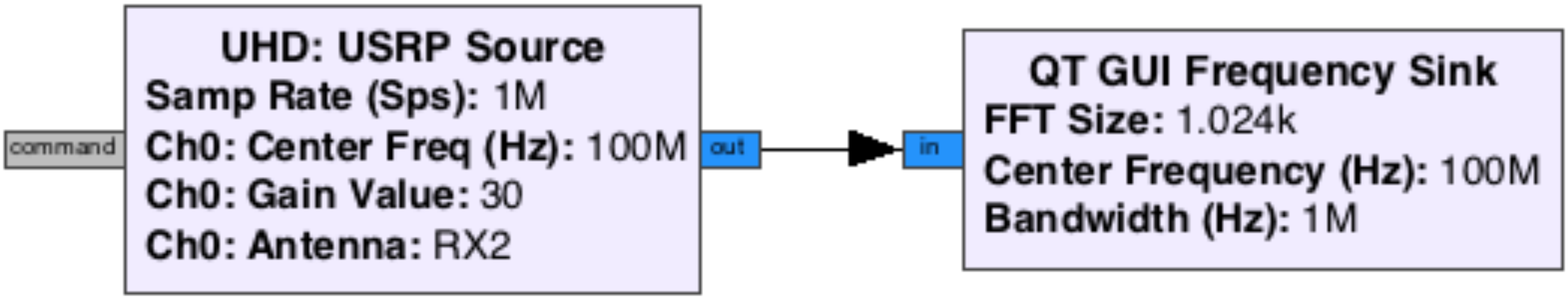



To get large N:

- Long Observation
- High Bandwidth

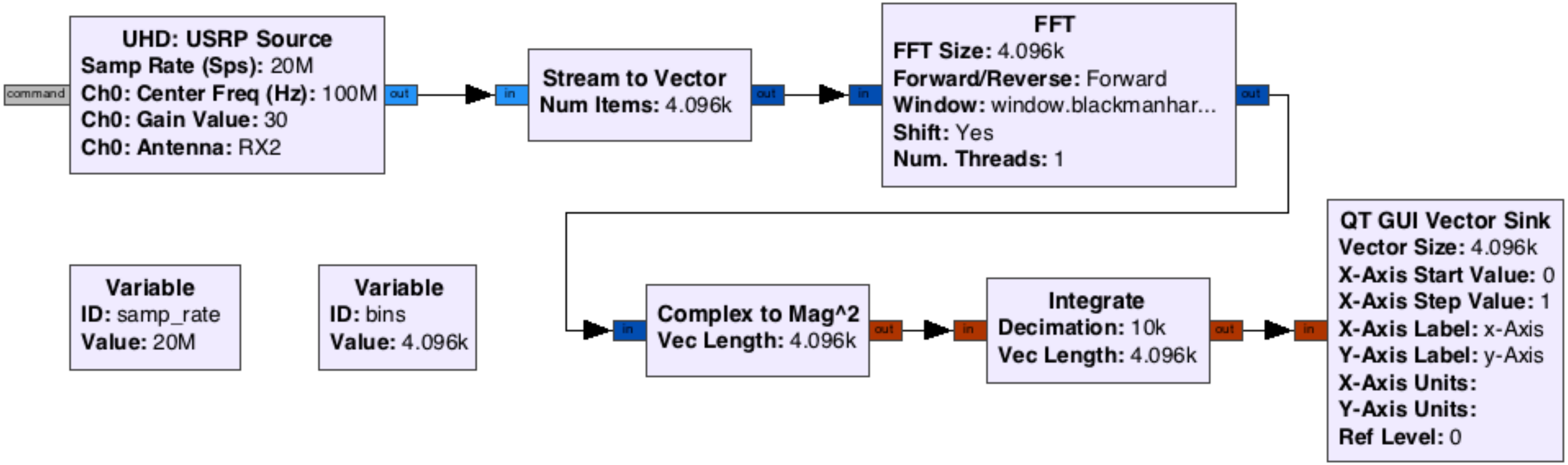
To get 1% accuracy in noise power: $N > 10{,}000$

# Hello World



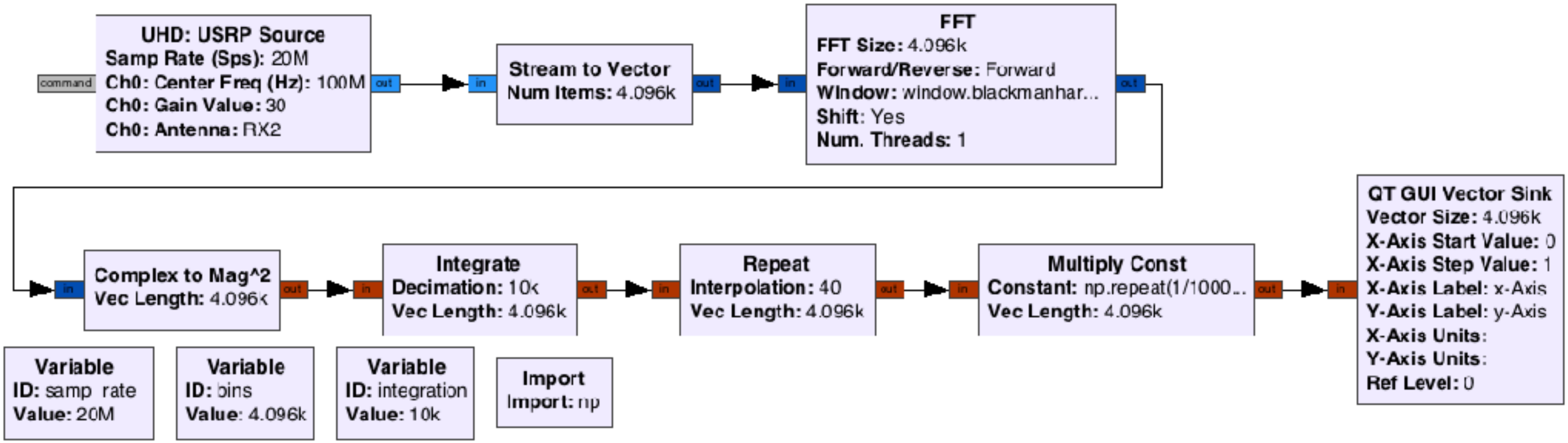

- Good for high dynamic range (log scale)
- Low CPU impact (runs only 10 times per second)
- This example: only 1% of samples get used
  - 1024 bins, 10 Hz, 1MS/s
- Poor sensitivity
- Long time averaging required

# FFT all the samples




- Resolution: 20MHz / 4096 = 5 kHz
- 1% power accuracy
- Update rate is samp_rate / bins / integrations = 2s
- Lag is tens of seconds !

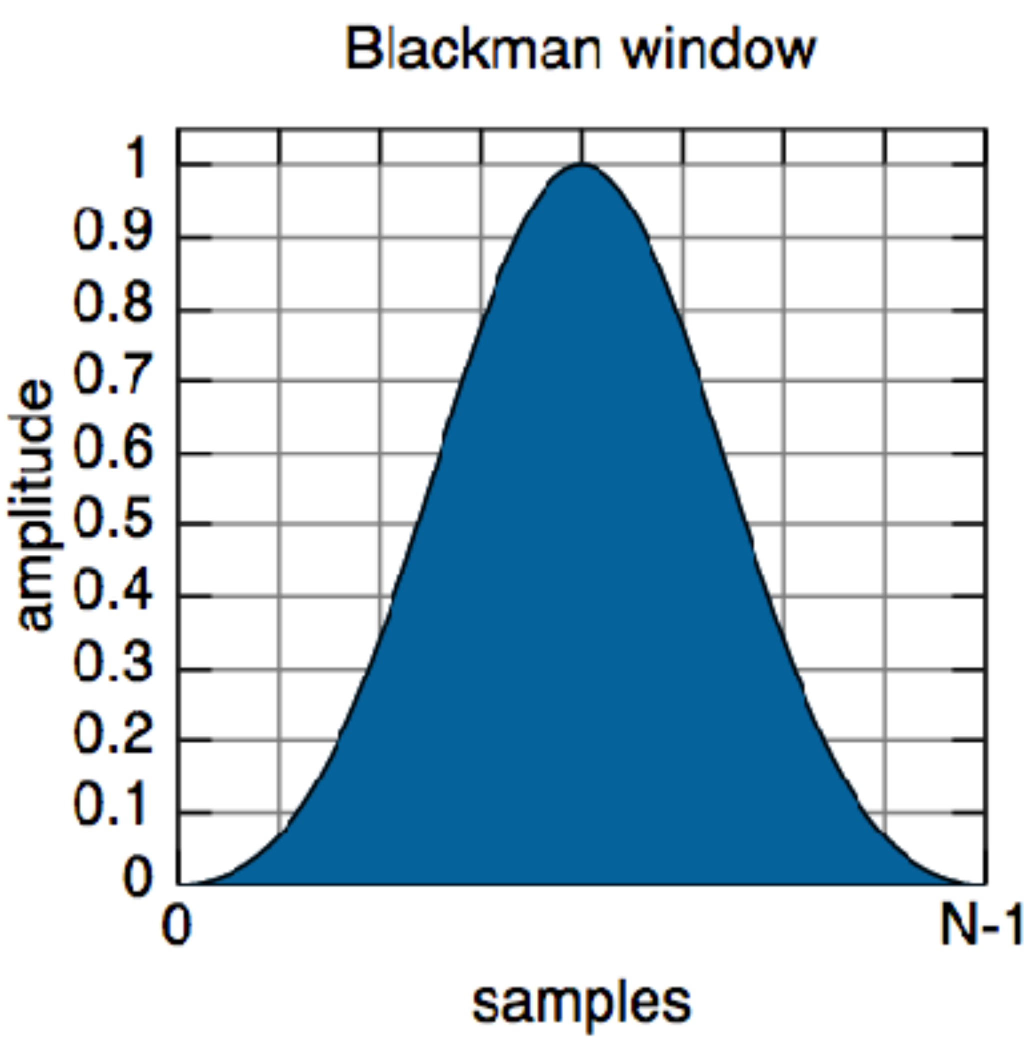
# Flushing the Pipeline

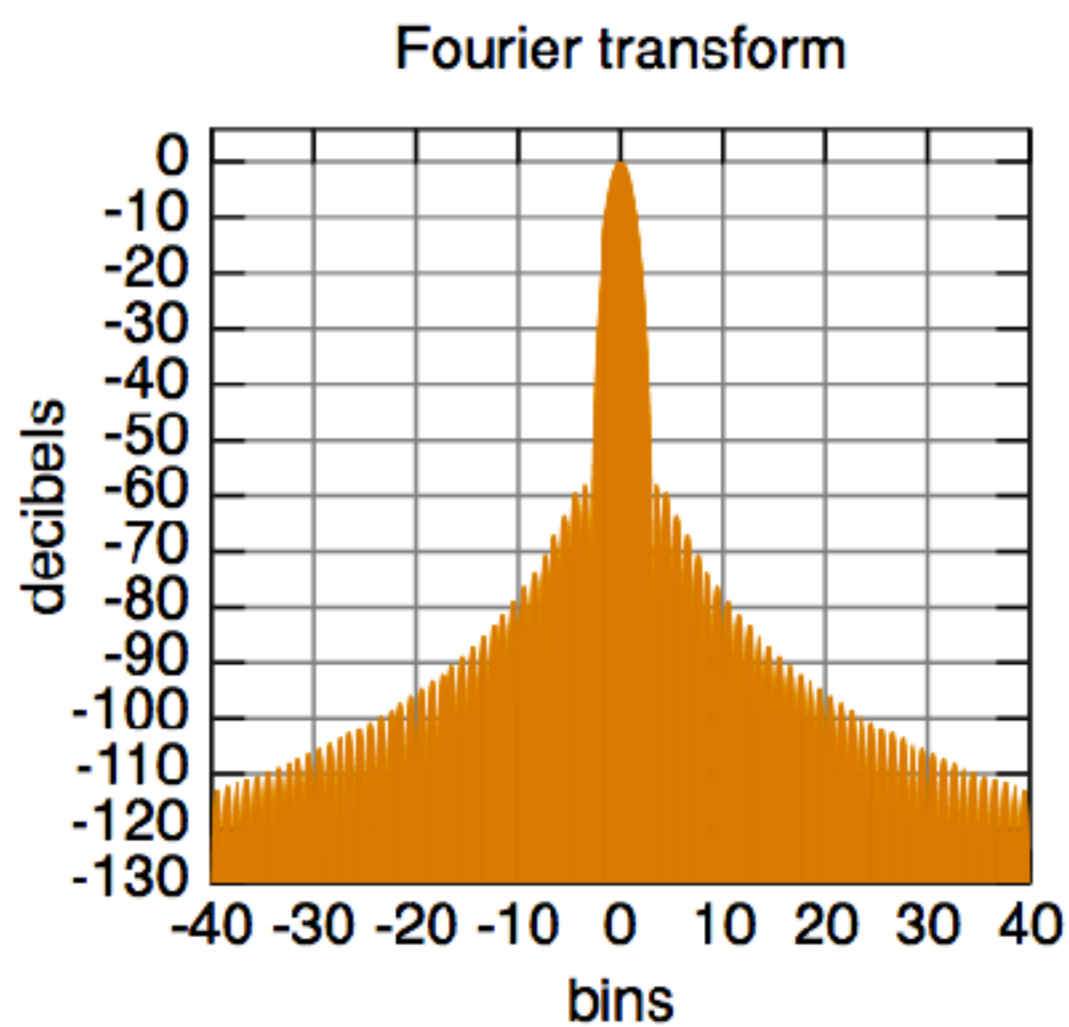



- Repeat the output of the integrator
- Only happens every 2 seconds anyway
- Flushes buffers so Vector Sink updates immediately
- Also introduced normalisation:
  - np.repeat(1/integration, bins)
  - After integration, so it takes far less CPU

# The Window Function

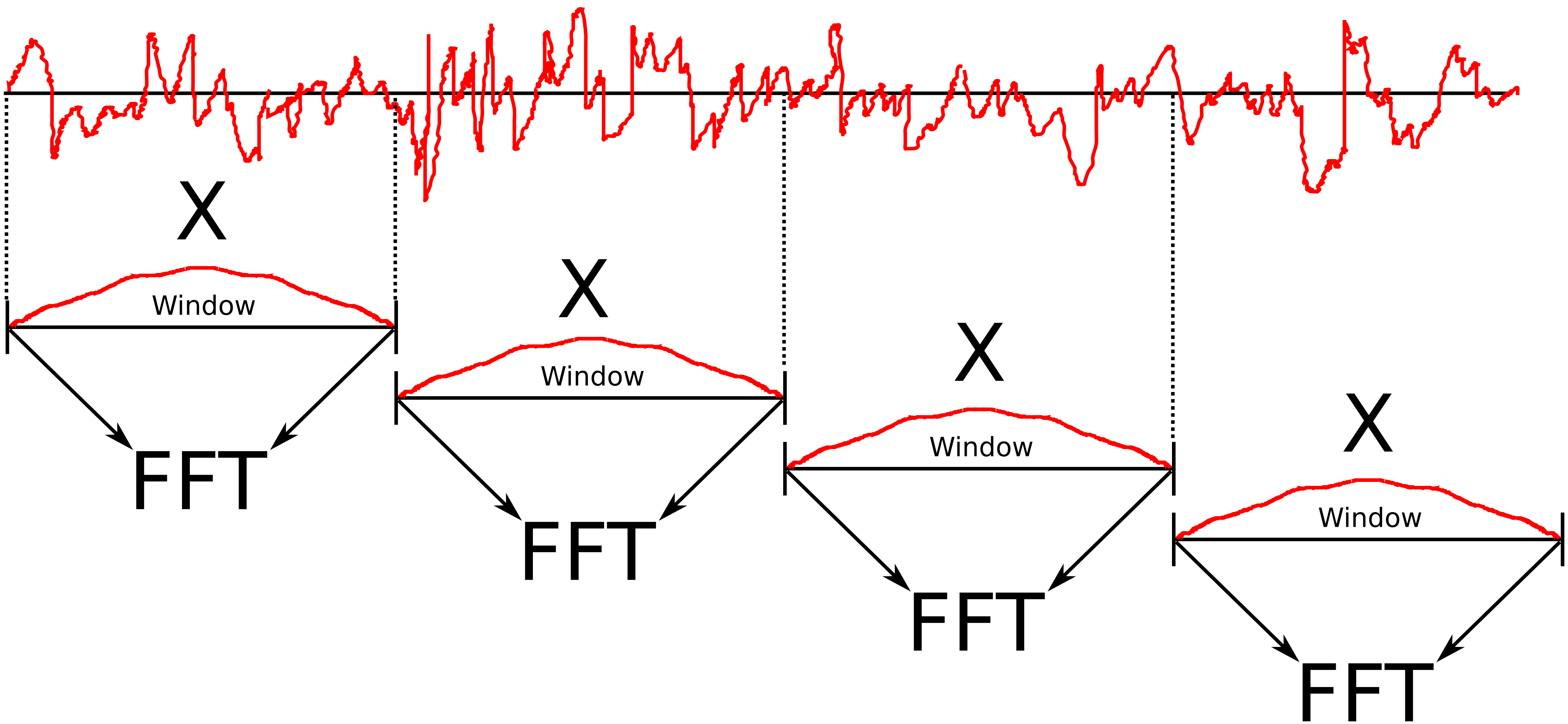
- Applied cyclically to the input data before FFT
- Goal: suppress side-lobes for $f \neq f_s / N$
- Side effects:
  - Reduction in Frequency Resolution
  - Reduction in SNR: Samples at edges have low weight



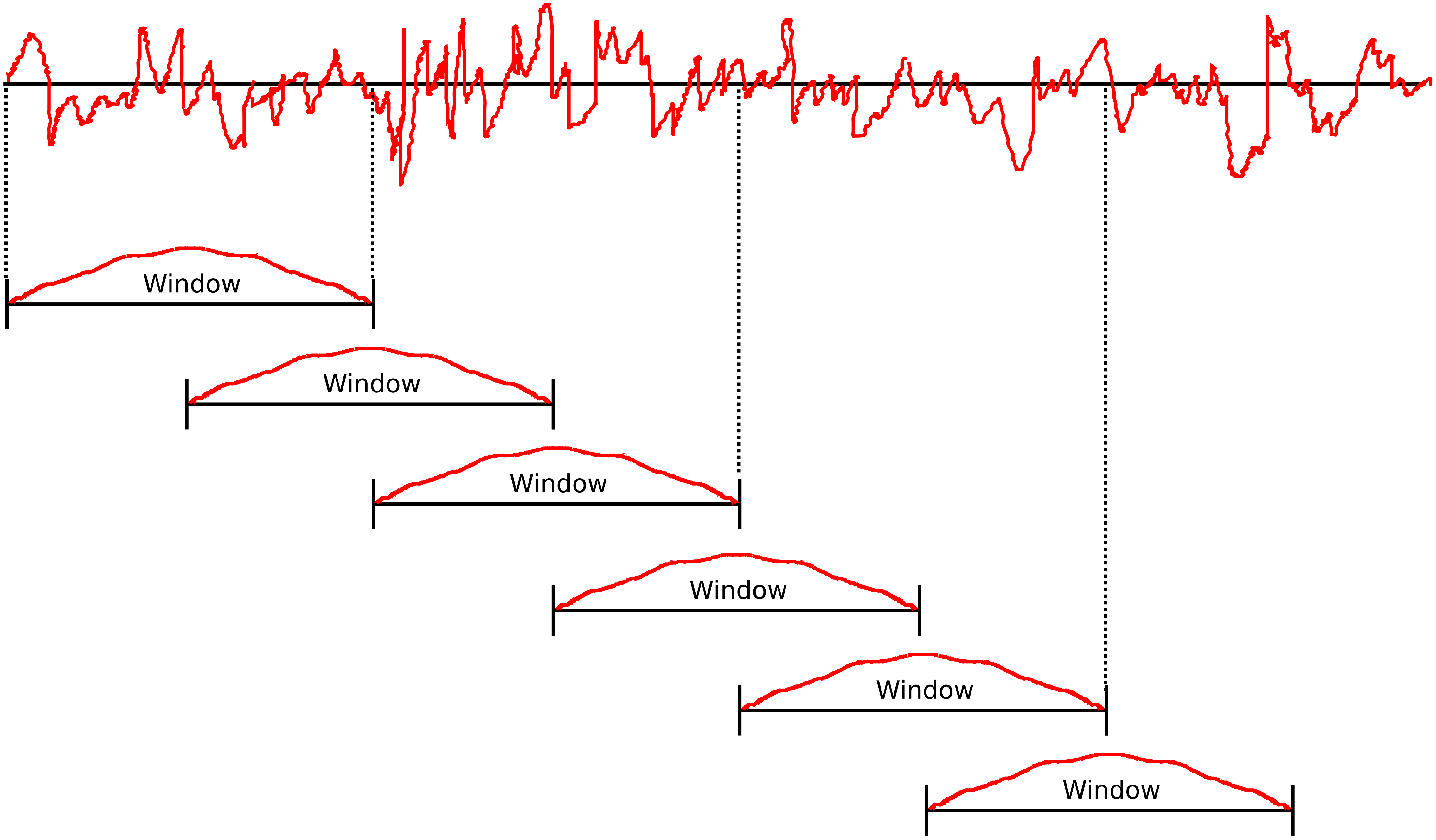

Source: Wikipedia

# FFT without Overlap



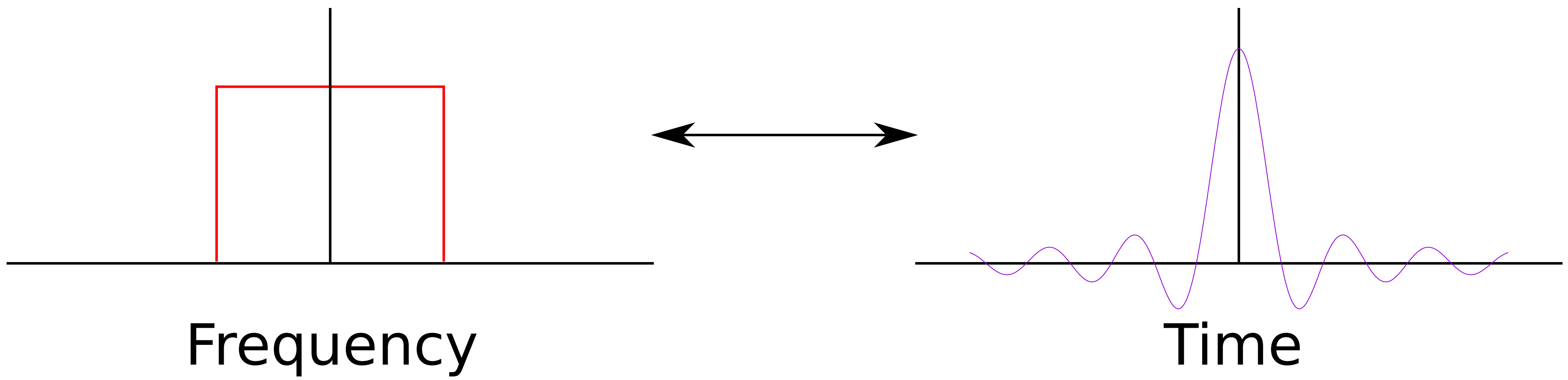

- Low (zero) weight for samples at window edge
- Loss of sensitivity, as samples are thrown away

# FFT with Overlap



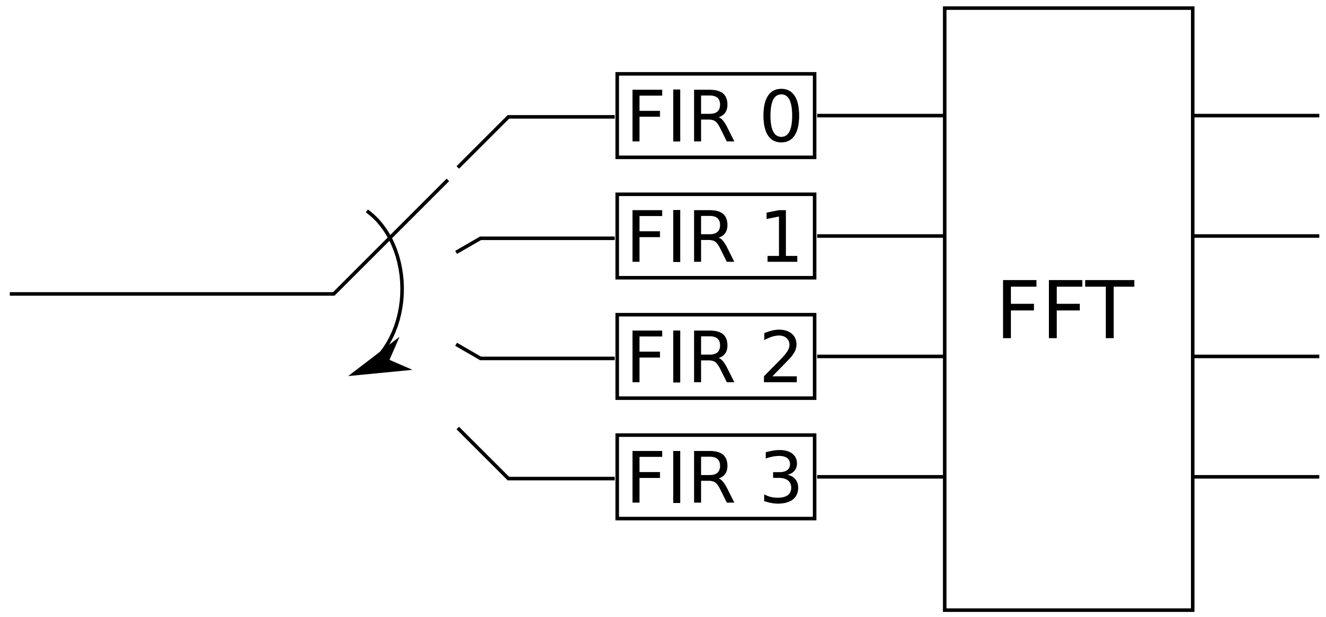

- Increase in CPU resources scales with overlap factor !

# Sync Window




- The dual of a rectangular (box) function is the sinc
- sinc(x) = sin(x)/x
  - For x=0, sinc(x) = 1
- Extends into infinity (in time domain)
- Truncation in time required
  - Less perfect ‘box’ shape in frequency space
  - Window much longer than FFT length
- Scale of Sinc determines width of frequency ‘box’
  - Determine overlap or gap between bins

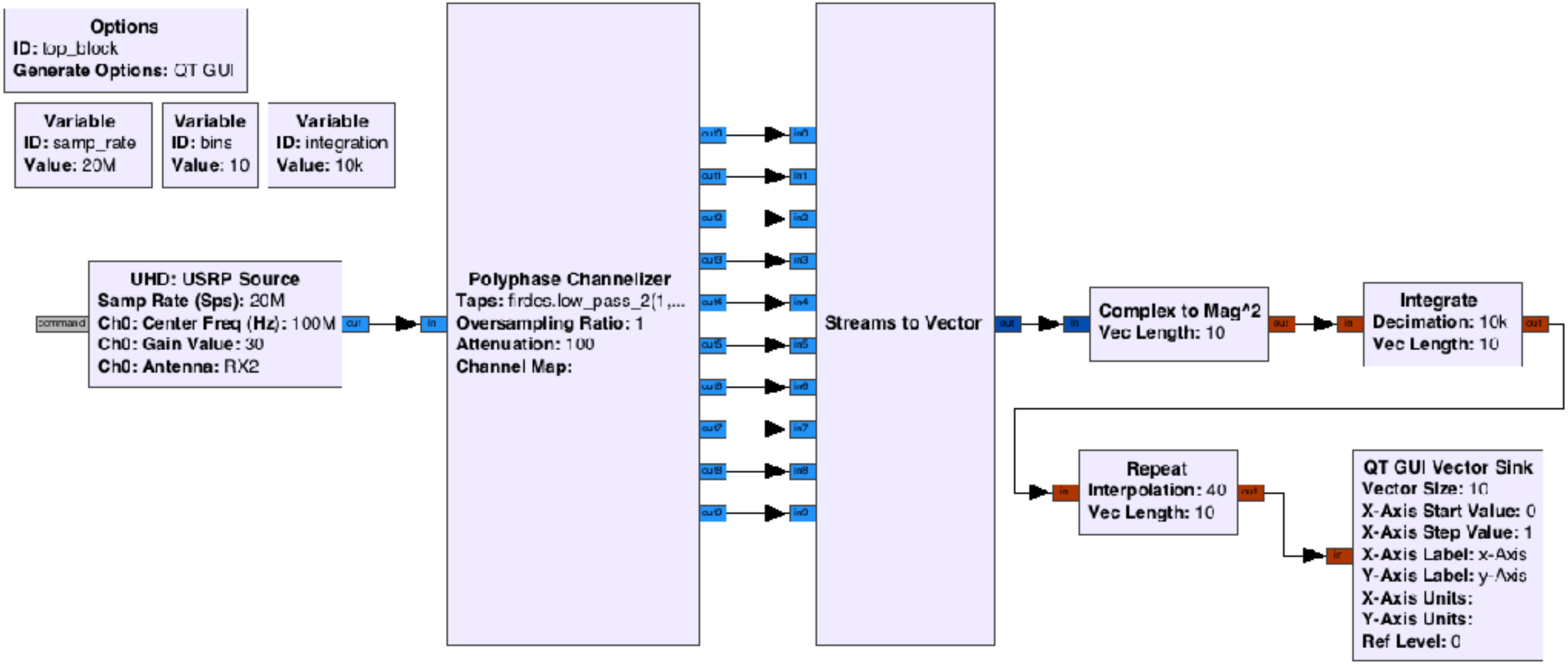
# Polyphase




- Implement long window as group of FIR filters
  - Polyphase decomposition
  - GRC does this for us

- Longer filter gives better frequency response
  - But loses time resolution

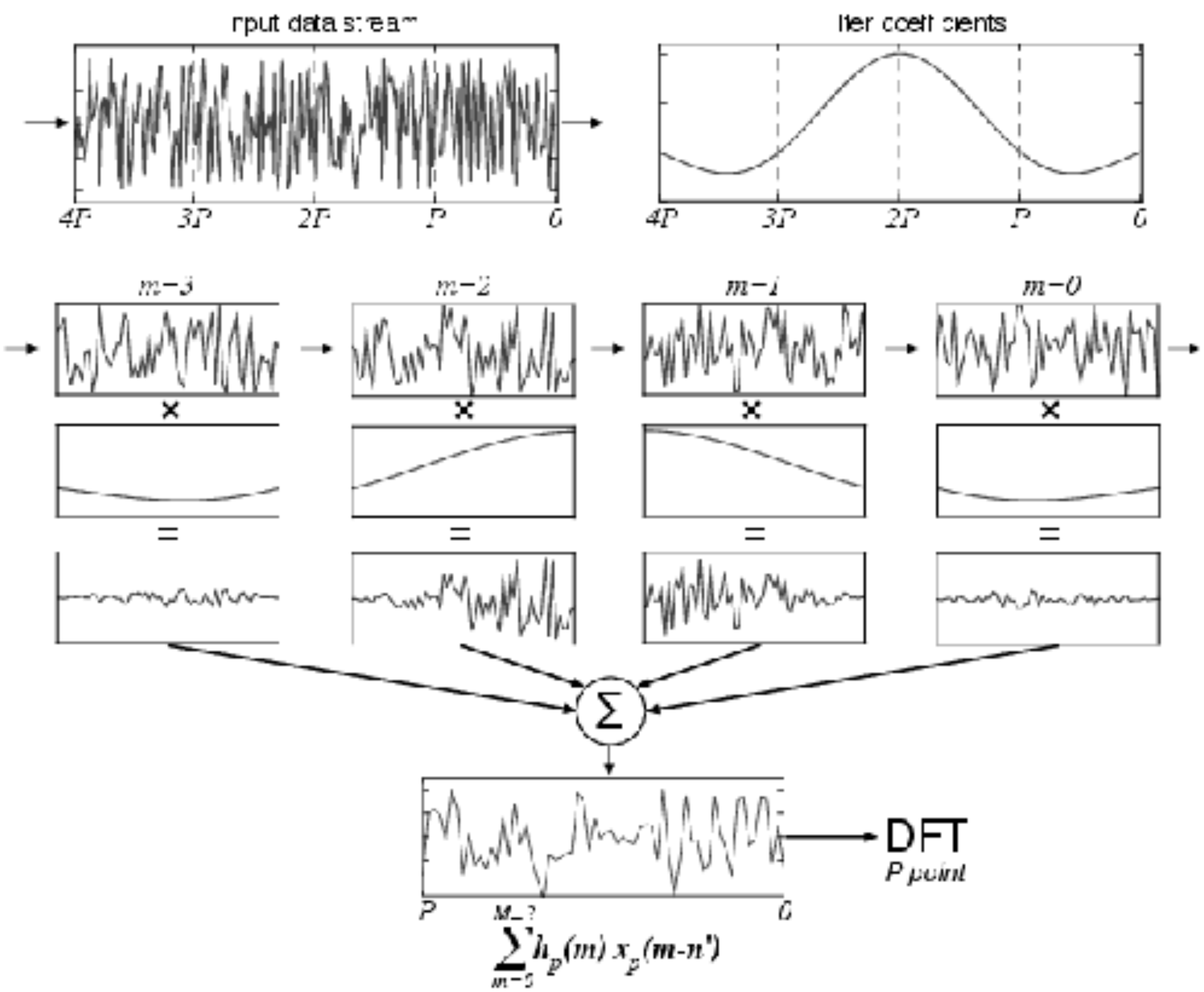
# Polyphase in GRC

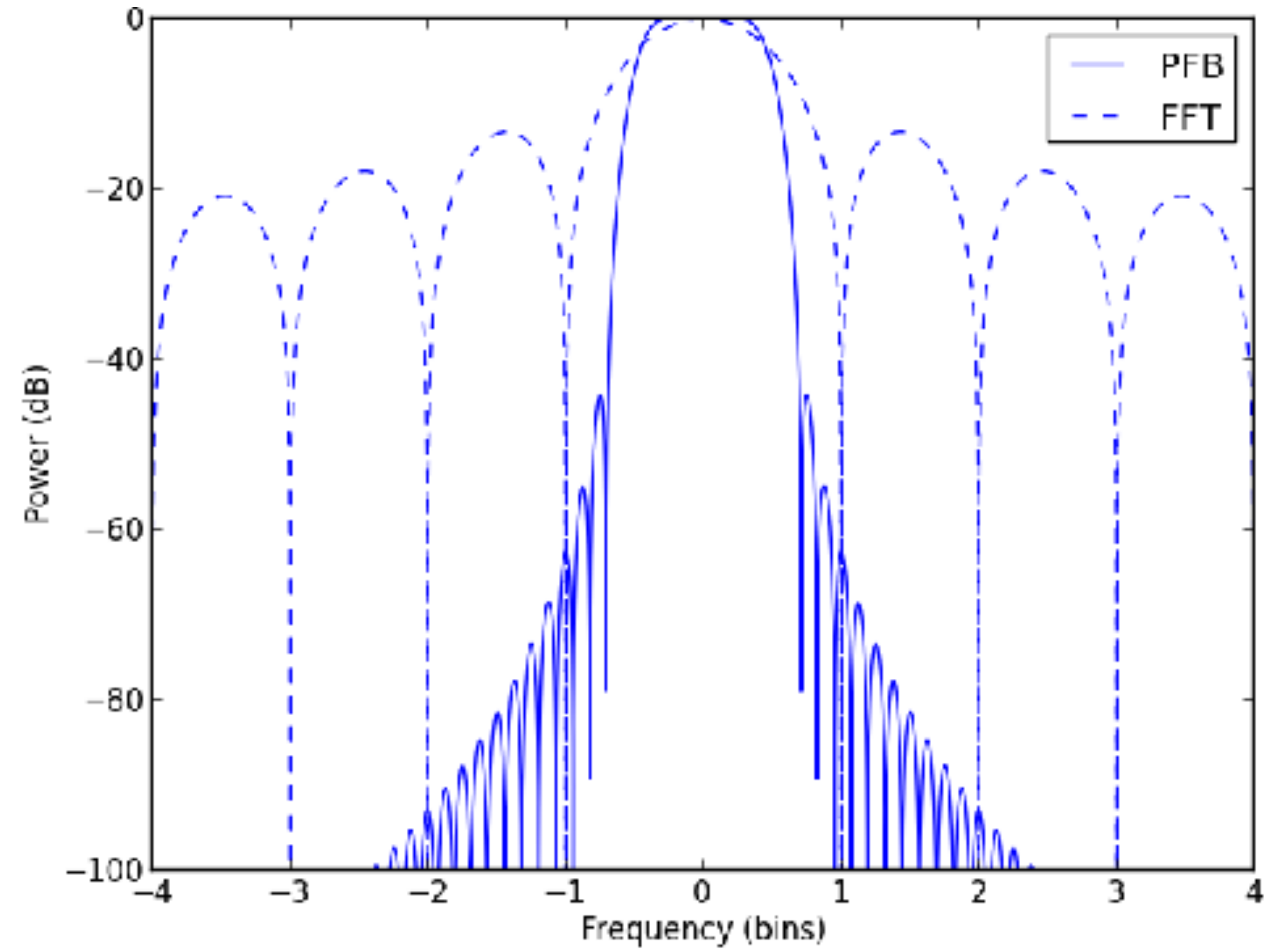



- Would be much more useful with a vectorised output !

- One bin ‘wraps around’, contains highest and lowest frequencies
  - For a small number of channels, shift frequency by half a bin

# Weight Overlap Add

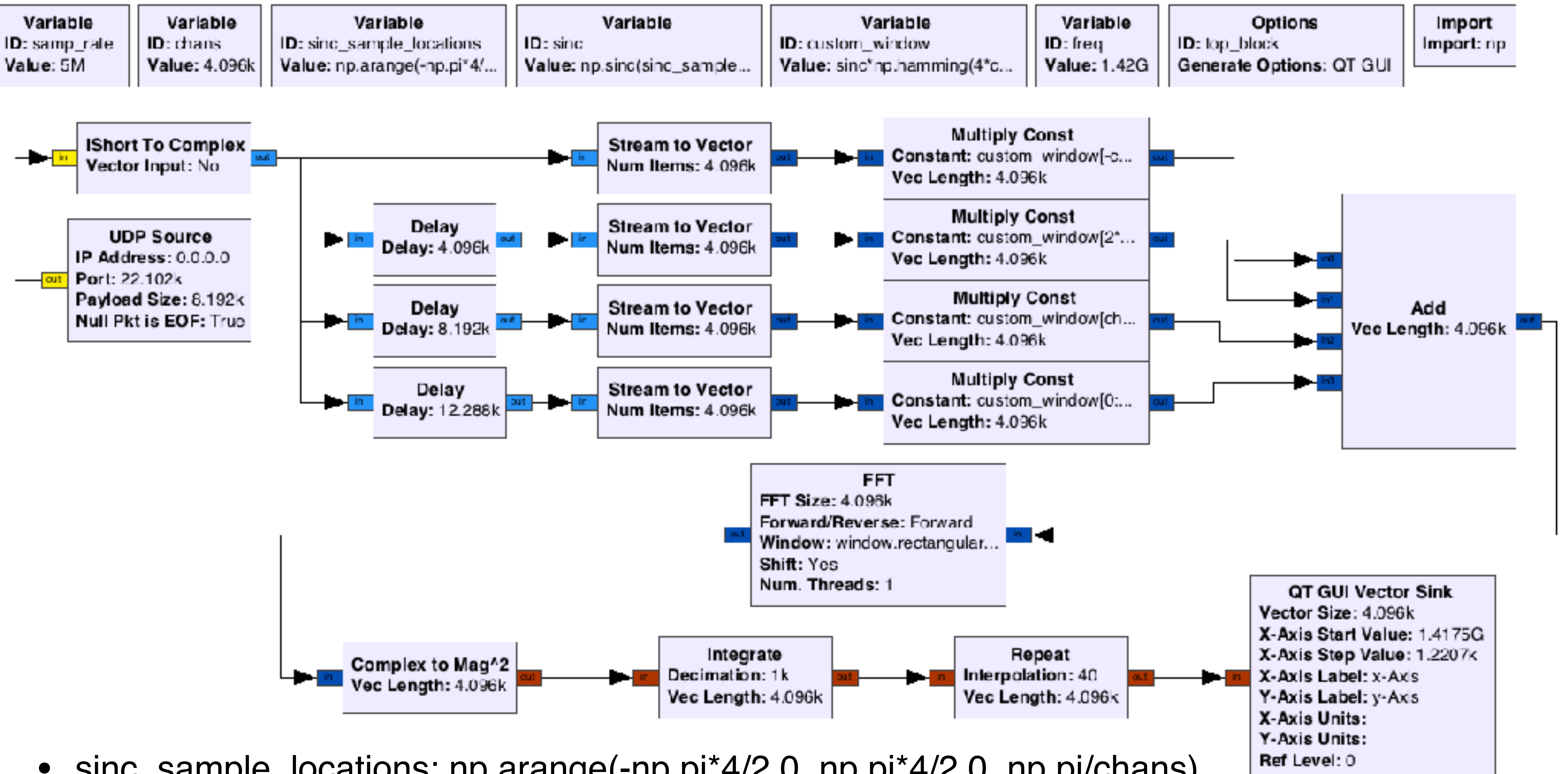



**Source: https://arxiv.org/pdf/1607.03579.pdf**

- Sinc window (perhaps multiplied with e.g. hamming)
- Same behaviour as polyphase, just different implementation
- More overlaps allows less truncated sinc(x)
    - Better frequency box shape
    - Worse time resolution

# Weight Overlap Add (WOLA)




- sinc_sample_locations: np.arange(-np.pi*4/2.0, np.pi*4/2.0, np.pi/chans)
- sinc: np.sinc(sinc_sample_locations/np.pi)
- custom_window: sinc*np.hamming(4*chans)
- Top to bottom: custom_window[-chans:], [2*chans:3*chans], [chans:2*chans], [0:chans]
- Based on: http://wvurail.org/dspira/labs/05/

# Merci de votre attention