C++ Real-Time Audio Programming with Bela

Dr Andrew McPherson

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London

Founder and Director, Bela

Course topics

Programming topics

Working in real time
Buffers and arrays
Parameter control
Classes and objects
Analog and digital I/O
Filtering
Timing in real time
Circular buffers
State machines
MIDI
Block-based processing
Threads
Fixed point arithmetic
ARM assembly language

Music/audio topics

Oscillators
Samples
Wavetables
Control voltages
Gates and triggers
Filters
Metronomes and clocks
Delays and delay-based effects
Envelopes
ADSR
MIDI
Additive synthesis
Phase vocoders
Impulse reverb

Today

Lecture 20: Phase vocoder, part 3

What you'll learn today:

Modifying signals in the frequency domain
Converting frequency to phase
Analysis and synthesis windows

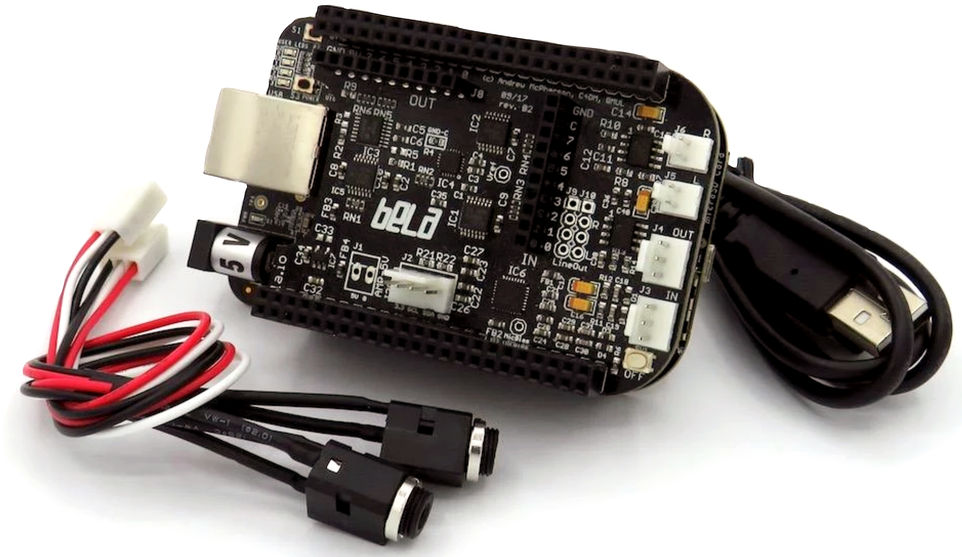
What you'll make today:

Pitch shifting, robotisation and whisperisation effects

Companion materials:

github.com/BelaPlatform/bela-online-course

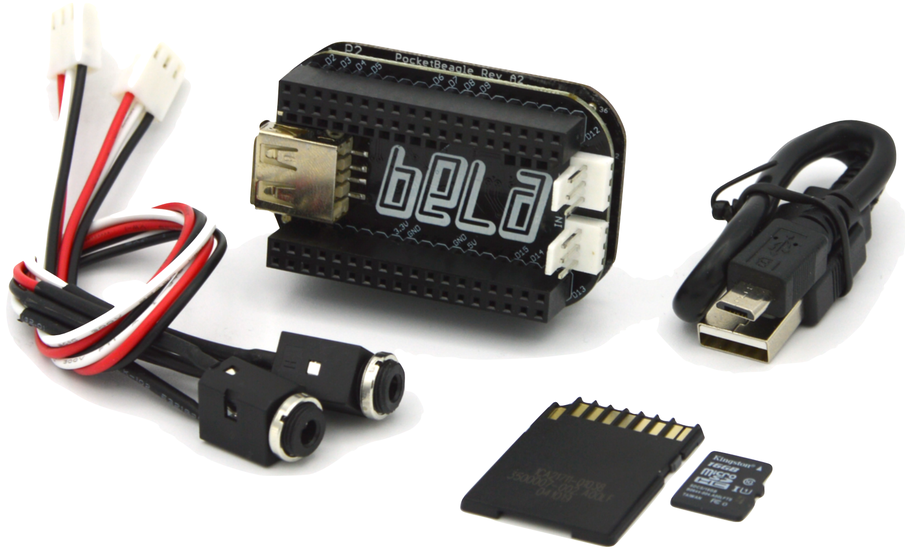
What you'll need



Bela Starter Kit

or

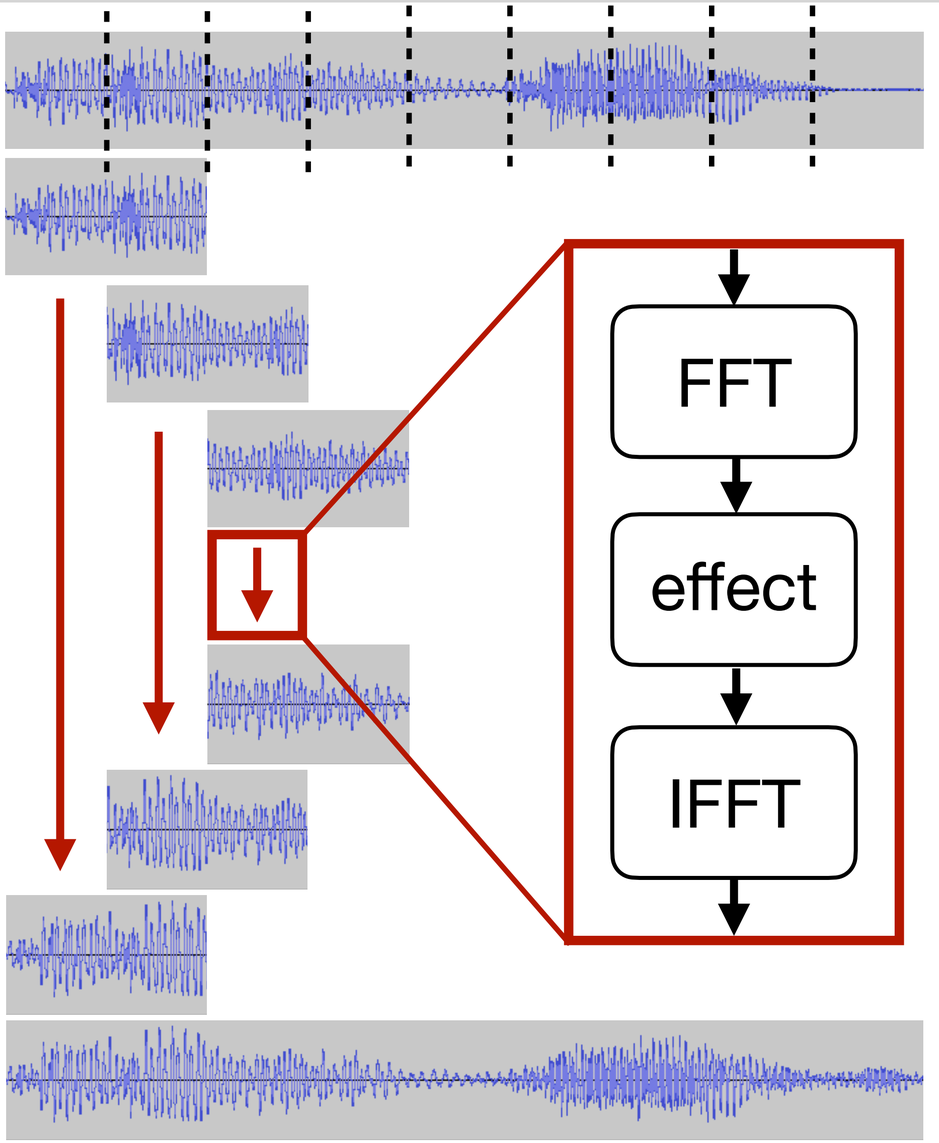
[shop.bela.io]



Bela Mini Starter Kit

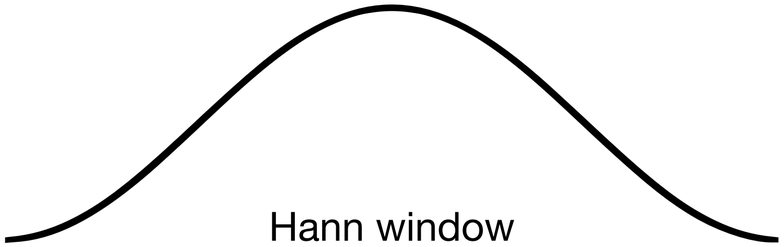
Overlap-Add

- A standard approach for block-based processing with overlapping blocks is called overlap-add:
  1. Isolate a block of length M using a windowing function
  2. Take an FFT of length N ≥ M of the segment
     - If N > M, zero-pad the block (add zeros to end of the window)
  3. Do something interesting to frequency-domain data
  4. Take an IFFT to get a new time domain segment
  5. Add the segment to an output buffer which also contains the preceding segments
     - As we'll see, we can't write the segment directly to the audio output
  6. Advance by the hop size (H) to the next frame and repeat
     - In real time, count samples until H more have arrived
     - Hop size H is less than window size M, hence the **overlap**

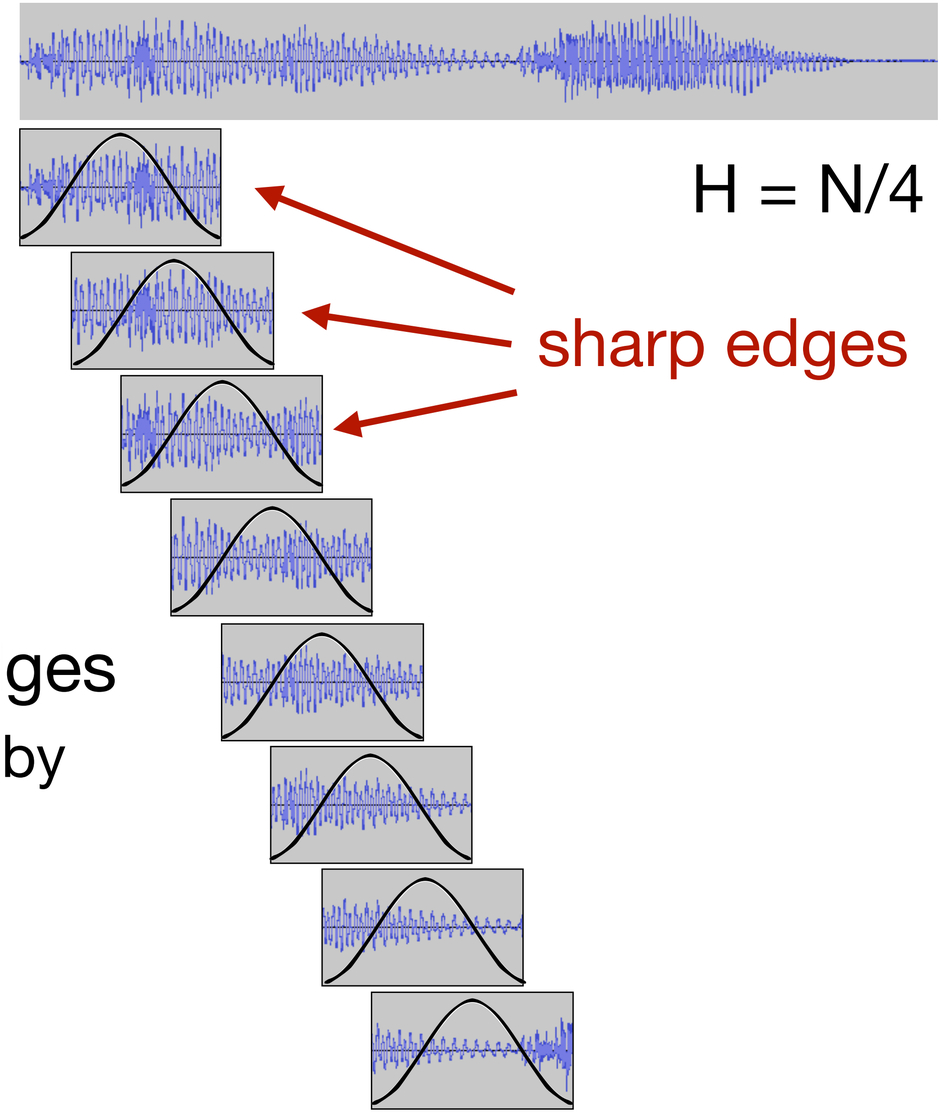


Windowing

- Overlap-Add process segments the signal into finite-length windows
  - These segments can have discontinuities at the edges (assuming 0 outside the window)
  - We might not notice this if we just reconstruct the signal with no frequency-domain processing
  - If we modify the signal, these sharp edges can lead to crackling or buzzing sounds
- We can use a window function to taper the edges
  - Pre-calculated shape that we multiply the segment by before FFT



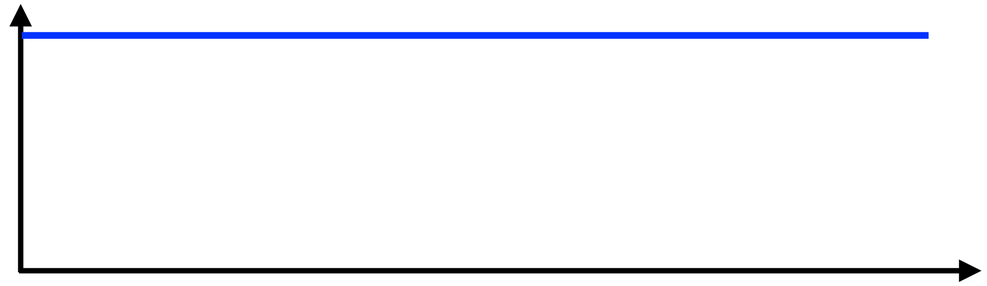
Hann window



Types of window function

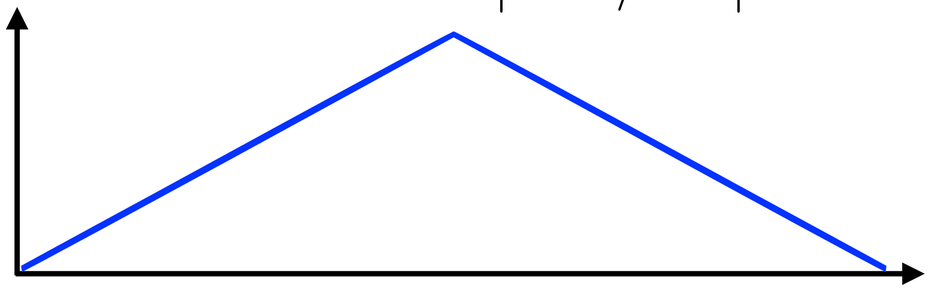
Rectangular

$$w[n] = 1 \quad \text{for } 0 \leq n < N$$



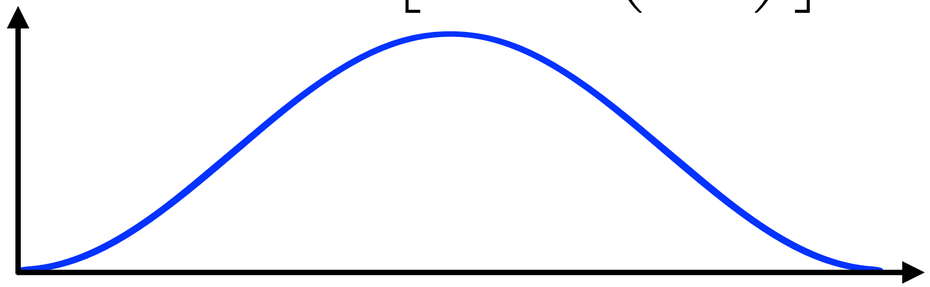
Triangular (Bartlett)

$$w[n] = 1 - \left| \frac{n - N/2}{N/2} \right|$$



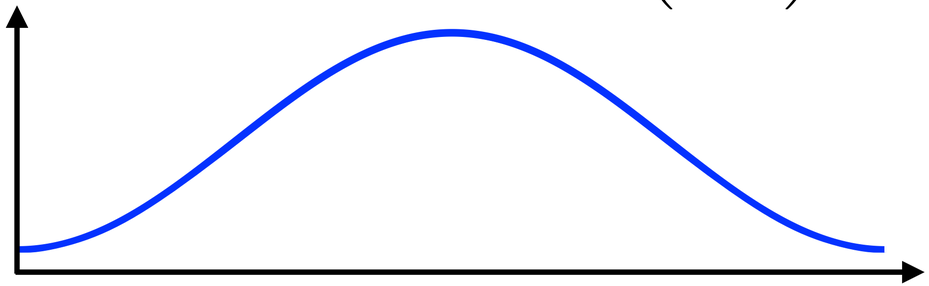
Hann

$$w[n] = 0.5 \left[1 - \cos\left(\frac{2\pi n}{N} \right) \right]$$



Hamming

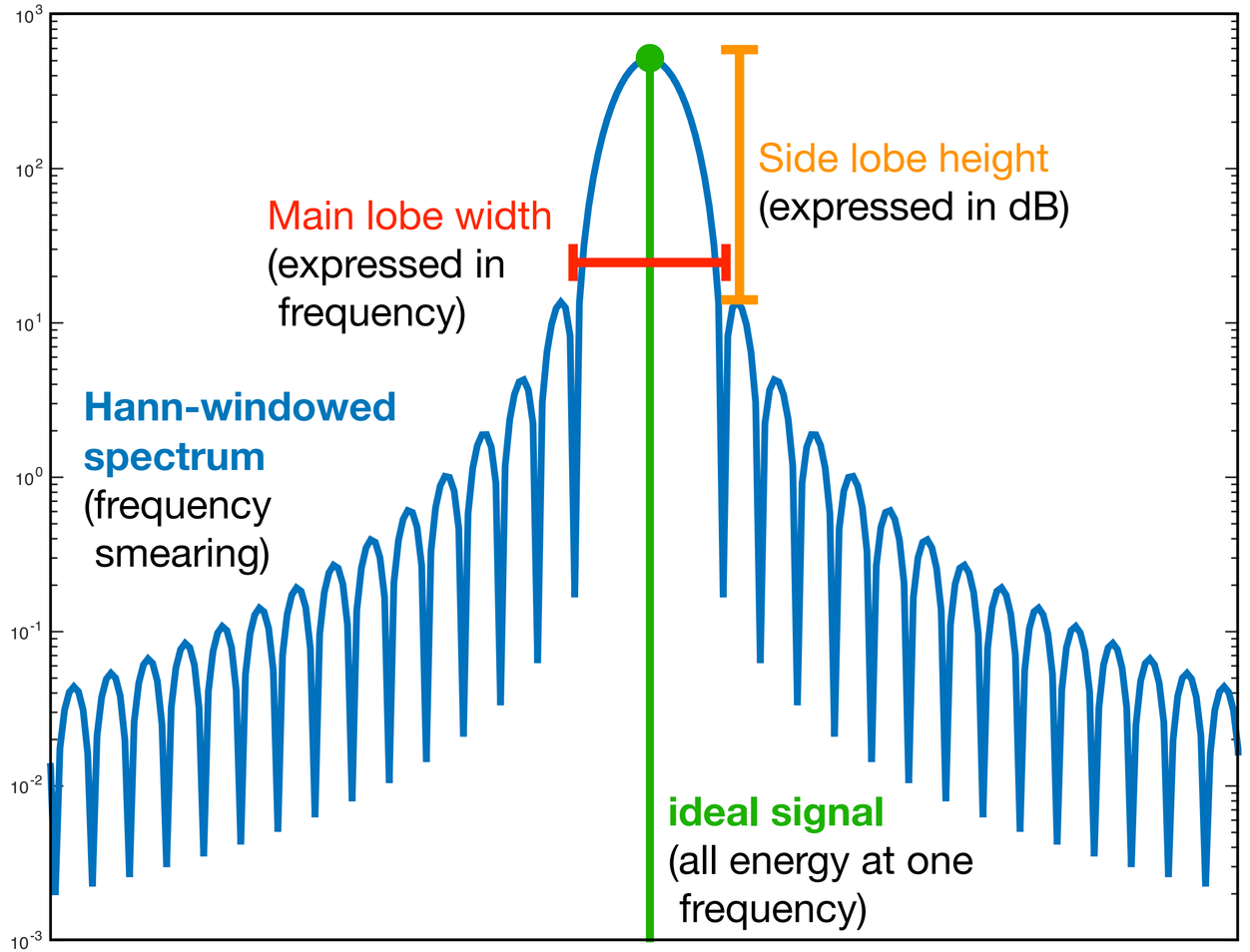
$$w[n] = \frac{25}{46} - \frac{21}{46} \cos\left(\frac{2\pi n}{N} \right)$$



And more types: Blackman, Kaiser, Flat Top, ...

Effects of windowing

- Windowing lets us isolate a particular time segment of our signal
- The cost is smearing of energy in the frequency domain
  - Energy appears at frequencies where it doesn't exist in the original signal
- Each frequency component of the signal appears as a series of lobes
  - Main lobe: a broad peak centred around the actual frequency
  - Side lobes: secondary peaks at other frequencies
- We typically want to minimise:
  - Main lobe width
  - Side lobe height

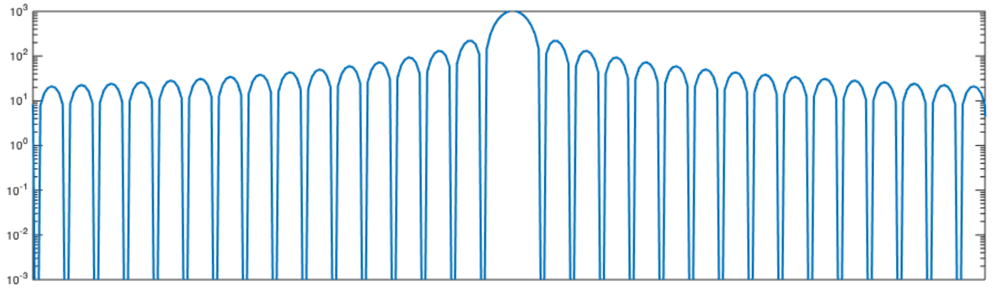


Window spectra

Rectangular

Main lobe width: 4π/N

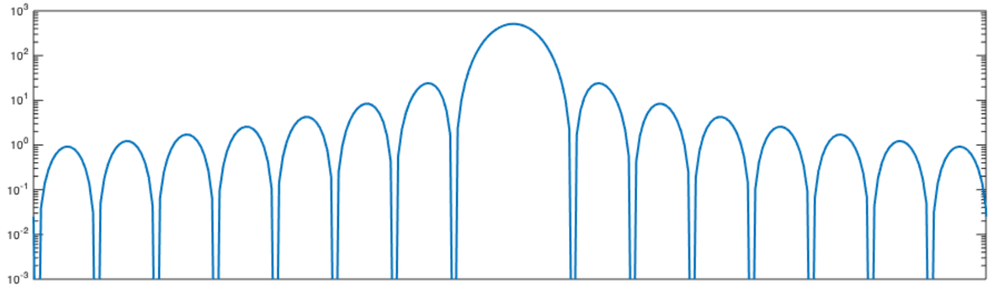
Side lobe height: -13dB



Triangular (Bartlett)

Main lobe width: 8π/N

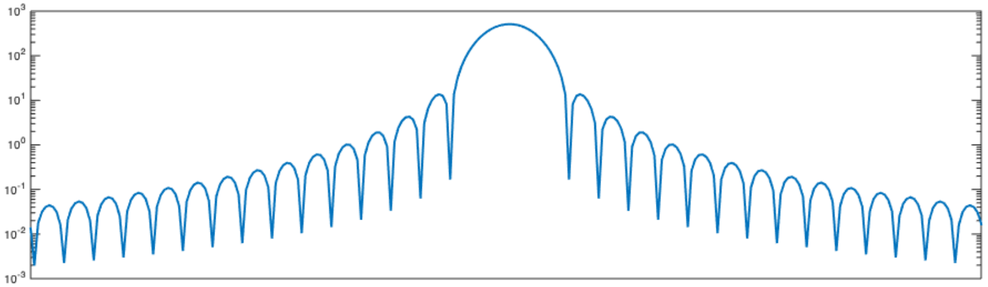
Side lobe height: -26.5dB



Hann

Main lobe width: 8π/N

Side lobe height: -31.5dB

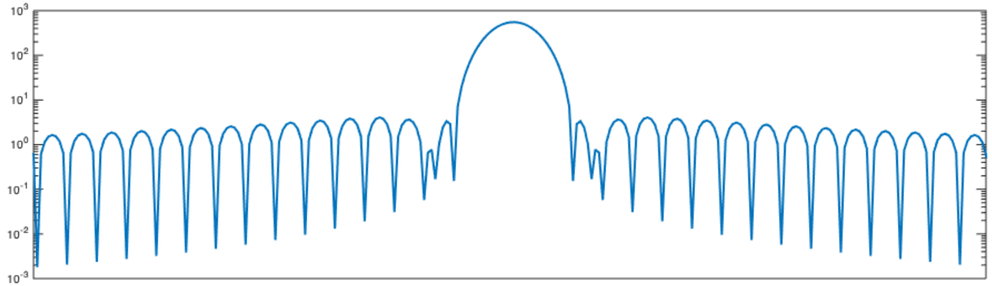


→ When in doubt: use a Hann window

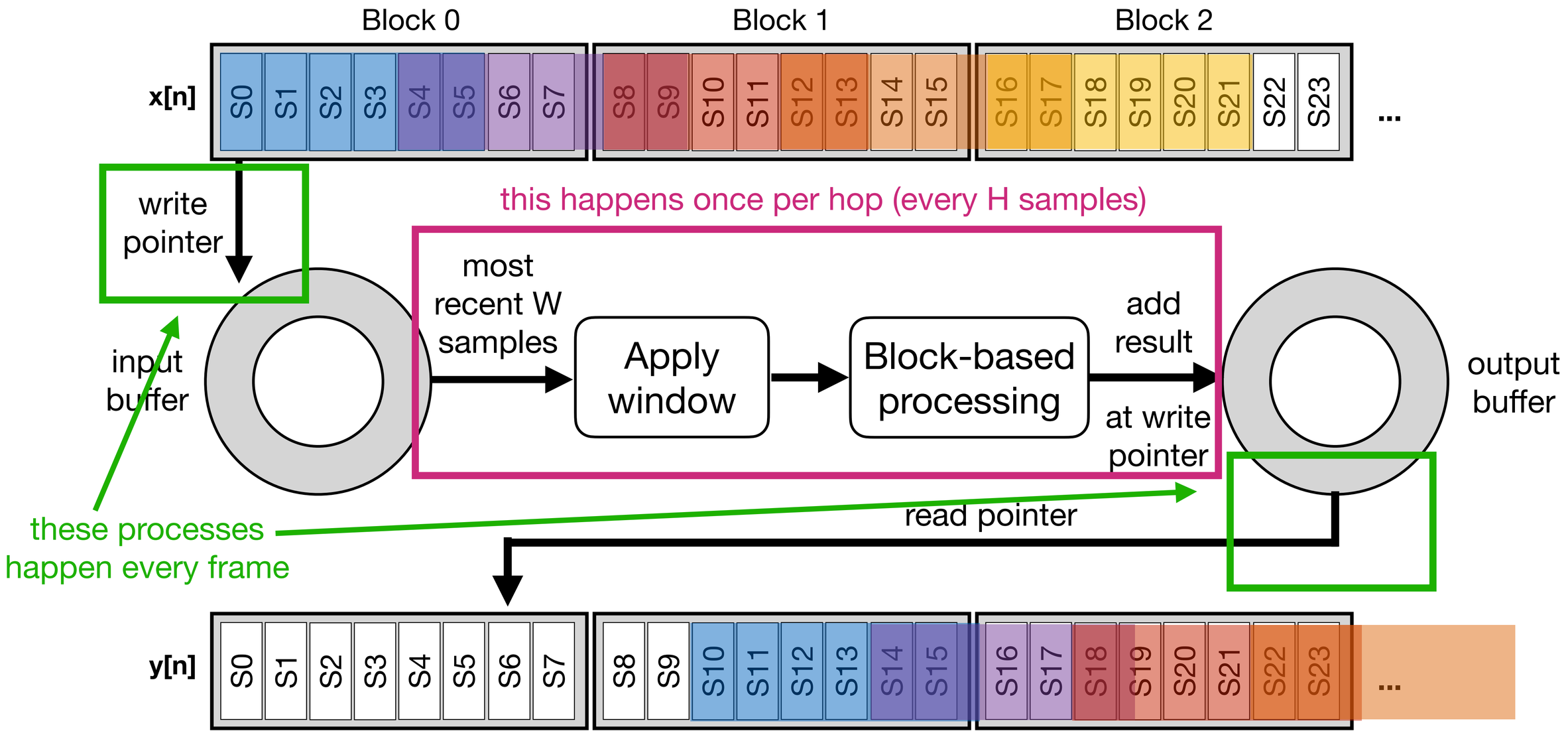
Hamming

Main lobe width: 8π/N

Side lobe height: -42.8dB

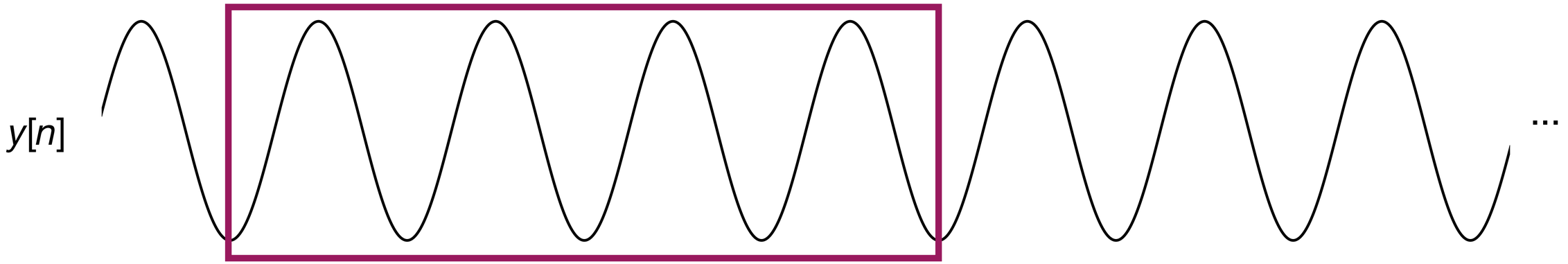


The full signal chain



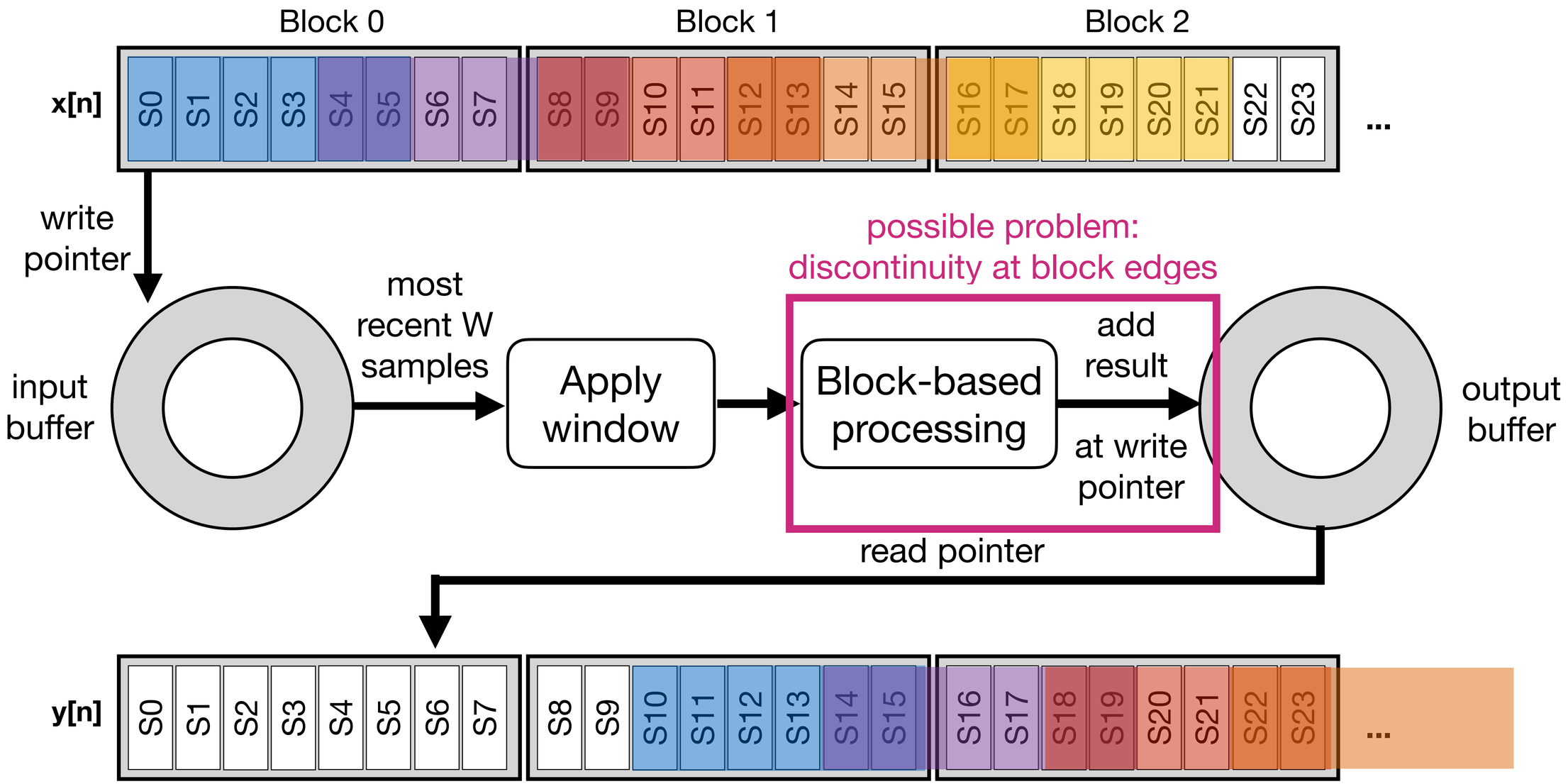
Robotisation

- A simple phase vocoder effect to generate robot-like voices
- At each hop, set the phase of every bin to 0
  - But preserve the magnitude of the bin, preserving the shape of the spectrum
- What does zeroing the phase do?
  - No phase change between hops
  - That means that hops always advance by an integer number of cycles

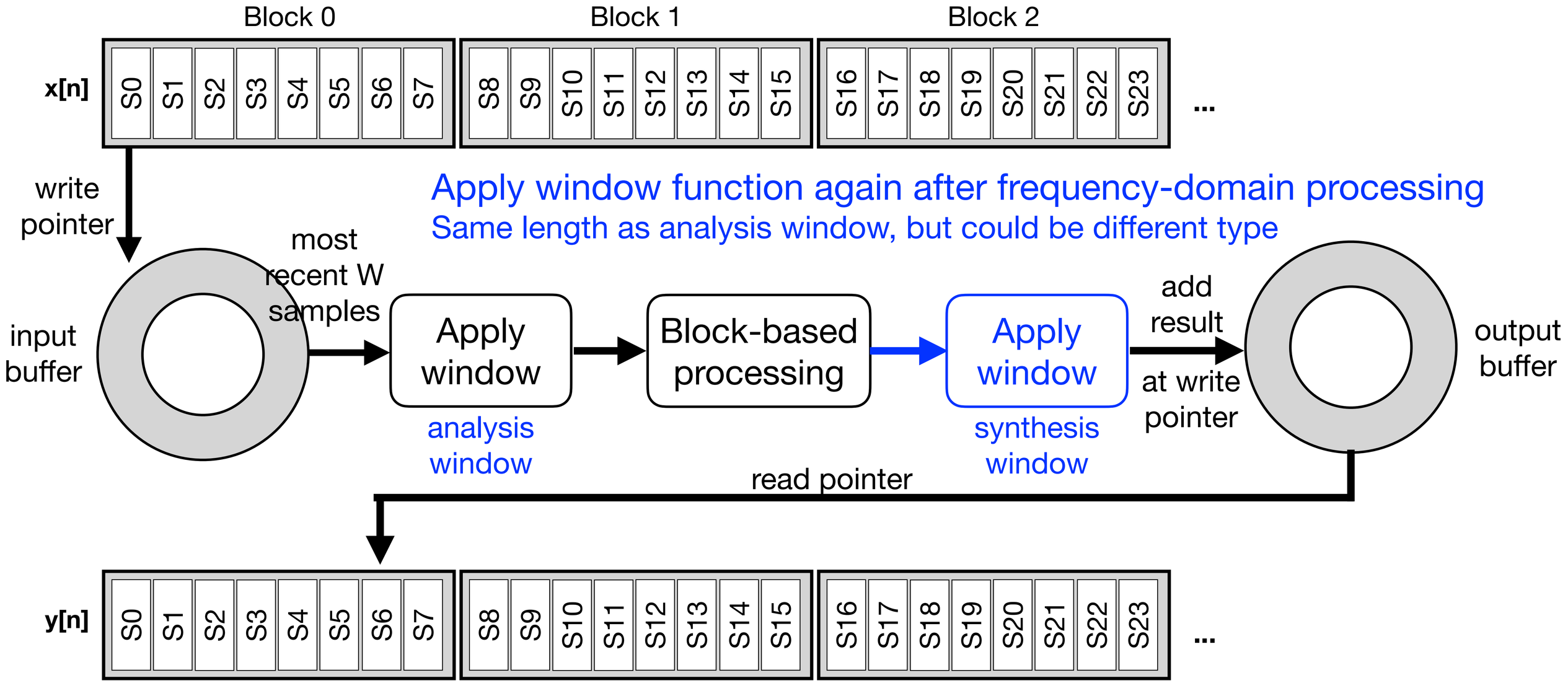


  - Thus the period of every bin must be an integer division of the hop size
    and the frequency of each bin must be a multiple of the the hop frequency f_s/H
  - So the robotisation effect imposes a constant pitch determined by the hop size

Review of signal chain

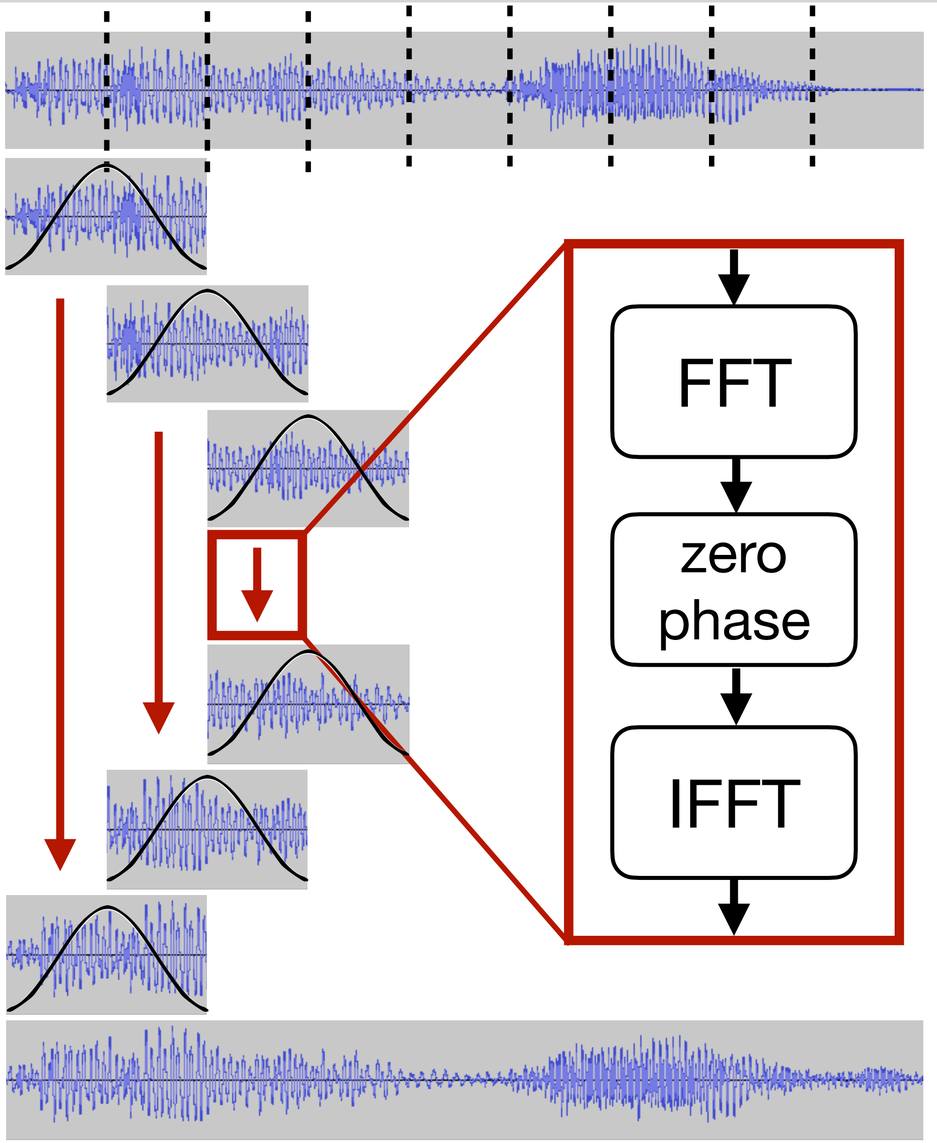


Analysis and synthesis windows



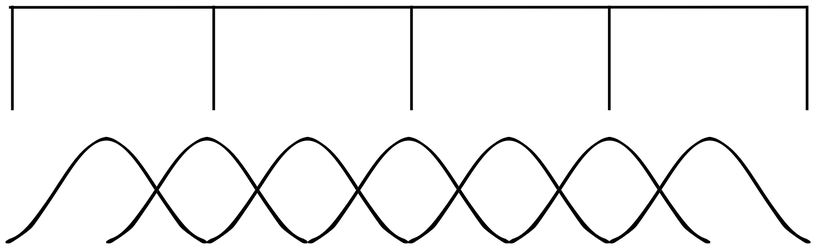
Robotisation

- For the cleanest effect, we need to apply both analysis and synthesis windows
  - Analysis window smoothes the edges of each block coming into the effect
  - However, changing the phase will scramble the shape of the block, so a synthesis window is also needed to smooth the edges again
- Task: in project fft-robotisation
  - Implement a Hann window for analysis and synthesis
  - You can calculate the window once in `setup()` and use the same array for analysis and synthesis
  - Multiply the block by the window in `process_fft()`
  - How does the window change the sound?
  - Next, make the hop size adjustable with the GUI



The COLA criterion

- Often, we want to exactly reconstruct a signal we windowed and transformed to frequency domain
  - i.e. when we do no processing in the frequency domain, we want to be able to get the same result back when done
- Exact reconstruction works for only certain combinations of window function and hop size
- Constant Overlap-Add (COLA) criterion:
  - Means that windows of length *W*, added together with a given hop size *H*, sum to a constant
  - For rectangular windows: *H* = *W*, *W*/2, *W*/3, W/4, ...
    - i.e. any integer division of the window size
  - For Hann, Hamming, Barlett: *H* = *W*/2, *W*/3, W/4, ...
    - i.e. divisions at most half the window size
  - Using Hann windows for analysis and synthesis: *H* = *W*/3, W/4, ...
    - Applying the window twice means it is effectively squared



Whisperisation

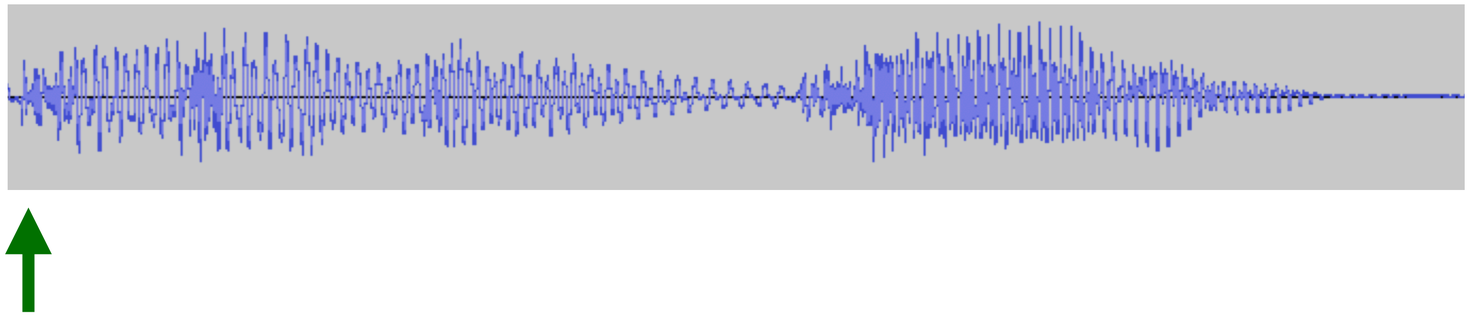
- A simple phase vocoder effect to obscure the pitch of a sound
  - On speech, it can produce a whisper-like effect
- At each hop, randomise the phase of each bin
  - This means the difference in phase between hops is also randomised
  - Therefore the frequency of the bins is also randomised (within a certain range)
- How much frequency variation can whisperisation introduce?
  - Phase difference between hops must be in the range ±π
  - How much difference is that? From earlier: $\omega[n] = \frac{\text{wrap}(\phi_r[n])}{H} + \omega_k$
  - So the frequency of each bin could differ by up to ±π/*H*
  - Smaller hop sizes allow more frequency variation
- As with robotisation, analysis and synthesis windows improve the effect
  - You might also find smaller FFT size produces a more convincing effect

Whisperisation task

- Task: in fft-robotisation code, change the code to whisperise the sound
  - In `process_fft()`, randomise the phase of each bin
    - Hint: to generate a random float between 0 and 1, use this expression: `(float)random() / (float)RAND_MAX`
  - Once you have the magnitude and phase, calculate real and imaginary components
    - We saw how to do this in Lecture 19
    - Use `cosf_neon()` and `sinf_neon()` for efficiency
  - Try different settings for window size and hop size

Pitch shifting

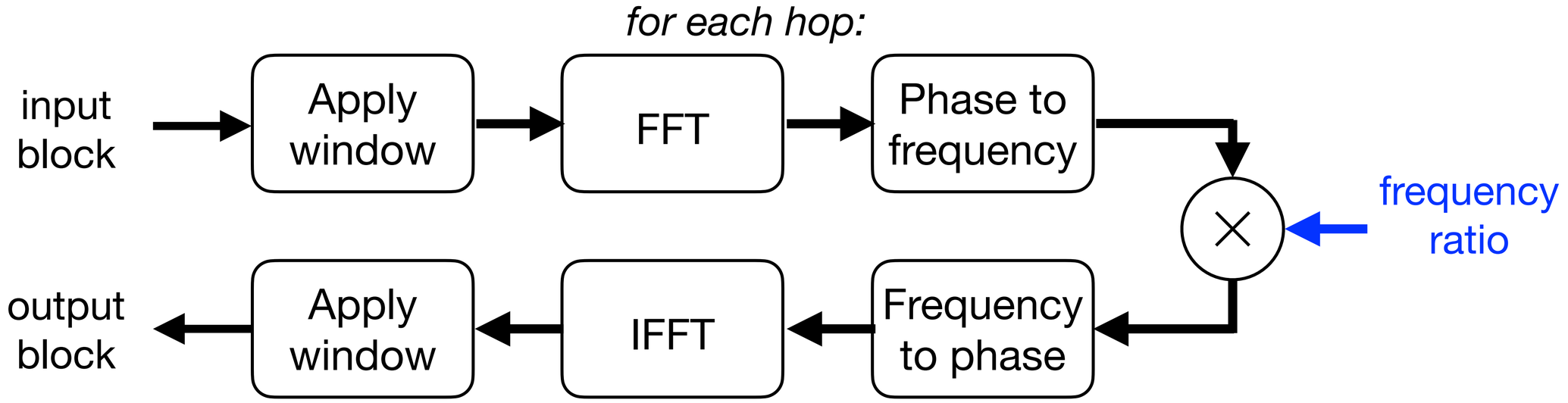
- When playing audio samples, playback speed and pitch are usually coupled
  - This is not a digital phenomenon: it's the same on turntables and audio tapes



- How, then, can we change the pitch of a sound without changing its speed?
- This cannot be done through filtering because the effect is nonlinear
  - Linear systems cannot create energy at frequencies that were not present in the input
  - However, for a pitch shifter, the output signal will have energy at different frequencies than the input signal
- Even though we can't use simple filters, the phase vocoder can help...

Pitch shifting

- We've seen that any signal can be represented as the sum of sinusoids
- The DFT gives us a snapshot of magnitudes and phases
  - From phase, we calculated precise frequencies for each bin
- Pitch shifting in the frequency domain just involves scaling each frequency
  - Multiply each frequency by a constant ratio: >1 raises the pitch, <1 lowers the pitch
  - Then we need to convert frequencies back to magnitudes and phases of each bin



Review: phase to frequency

- First we calculate the phase remainder: $\phi_r[n] = (\phi[n] - \phi[n-H]) - \frac{2\pi k H}{N}$

  measured **phase values** from bin *k* on **two successive hops**

  centre frequency of bin *k* multiplied by the **hop size**

- Then use remainder to calculate the deviation from the bin centre frequency:
  - We should wrap the phase to be between -π and π (i.e. the principal argument)

$$\omega[n] - \omega_k = \frac{\text{wrap}(\phi_r[n])}{H}$$

centre frequency of bin *k* (= 2π*k*/*N*)

- Finally, we can rearrange to get the exact frequency of our sine wave:

$$\omega[n] = \frac{\text{wrap}(\phi_r[n])}{H} + \omega_k$$

  - We could also calculate in terms of (fractional) FFT bins
    - Use the fact that bins are spaced 2π/*N* apart in frequency

fractional bin number of the sine wave

$$b[n] - k = \frac{\text{wrap}(\phi_r[n])N}{2\pi H} \longrightarrow b[n] = \frac{\text{wrap}(\phi_r[n])N}{2\pi H} + k$$

Frequency scaling

- Here we will talk about analysis and synthesis frequencies
  - The analysis frequency $\omega_a[n]$ is the frequency we calculated from our input signal
  - The synthesis frequency is a scaled version of it: $\omega_s[n] = R\omega_a[n]$
    - Where *R* is the pitch shift ratio
  - The notation involving *n* is to remind us that the frequencies change over time
  - Remember, there is one frequency for each bin of the FFT
- Problem: $\omega_s[n]$ might be in a different FFT bin than $\omega_a[n]$
  - Need to find the bin with the closest centre frequency
- Let's look at the analysis in terms of (fractional) bins: $b_a[n] = \frac{\text{wrap}(\phi_r[n])N}{2\pi H} + k$
  - The above was calculated for FFT bin *k*
  - Pitch shifting gives us a new fractional bin index $b_s[n] = Rb_a[n]$
  - Round to the nearest bin to figure out where this frequency goes: $k' = \text{floor}(Rk + 0.5)$
    - For $R > 1$, energy is likely to move into higher bins, and the reverse for $R < 1$

Frequency to phase

- Based on the bin frequency, we calculate a phase difference since last hop
  - Not an absolute phase; we need to remember each bin's phase from last hop
- The process of calculating the bin phase is the reverse of the analysis
  - The frequency shift gave us a bin *k'* and a fractional bin number $b_s[n]$
  - Use it to calculate the phase remainder: $\phi_{rs}[n] = \dfrac{2\pi H(b_s[n] - k')}{N}$
    - How much more or less will the phase advance at the synthesis frequency vs. the bin centre frequency
  - Then consider the expected phase shift based on the bin centre frequency, and add these to the phase of bin *k'* at the last hop

$$\phi_s[n] = \text{wrap}\left(\phi_s[n-H] + \phi_{rs}[n] + \frac{2\pi k' H}{N}\right)$$

  - Wrap the phase to be in the ±π range
- Meanwhile, the magnitude of the bin doesn't change in the pitch shift process

Frequency to phase: code

- We are calculating the phase difference from one hop to the next
  - Therefore, we need a global variable to hold the synthesis phases from the last hop
  - Since we need to hold a value for each FFT bin, this variable should be an array
- Each hop, we add the calculated phase difference to the output phase
  - Magnitude stays the same
  - Then we convert back to real and imaginary values, as before:
  - Real component: $a = M\cos(\phi)$ Imaginary component: $b = M\sin(\phi)$
  - The functions `cosf_neon()` and `sinf_neon()` from the Ne10 library are faster than `sin()` and `cos()`
- Another efficiency trick: exploit the symmetry of the FFT
  - We only need to calculate up to (and including) bin *N*/2
  - Above this, the bins are conjugate symmetric:
    - Same real component; negative imaginary component

Pitch shift task

- Task: in the project fft-pitchshift, implement the pitch shifter
  - The overlap-add process is implemented for you (see Lecture 18 for explanation)
  - Your code goes in `process_fft()`
  - The analysis step (phase to frequency) is already done from Lecture 19
- You should implement the synthesis step:
  1. Multiply each frequency component by the pitch shift value: $b_s[n] = Rb_a[n]$
  2. Check which bin the component belongs in: $k' = \text{floor}(Rk + 0.5)$
  3. Calculate the phase of the new bins:
  4. Convert magnitude and phase to real and imaginary components

$$\phi_{rs}[n] = \frac{2\pi H(b_s[n] - k')}{N}$$

$$\phi_s[n] = \text{wrap}\left(\phi_s[n - H] + \phi_{rs}[n] + \frac{2\pi k' H}{N}\right)$$

- To test:
  - Change pitch shift value in the GUI
  - If you like, try a real-time audio input using `audioRead()`

Pitch shift limitations

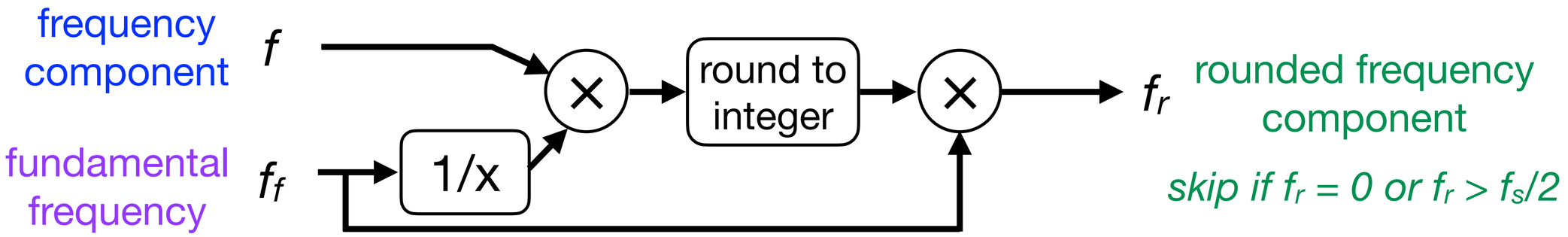
- Limitations of the pitch shift algorithm
  - When shifting pitch downward, components which started out in different bins might end up in the same bin (since frequencies get closer together when $R < 1$)
  - Artefacts from windowing (e.g. the main lobe and side lobes) will also be shifted
  - If the original signal has frequency components spaced more closely than the FFT bins, they may not get resolved and shifted correctly
  - No global preservation of phase which distorts transient events
    - "Phasiness" is a classic problem with phase vocoder effects
  - Latency: the longer the FFT, the better the pitch resolution but the longer the latency
- There is a big market for high-quality pitch shifting effects
  - Antares Auto-Tune, introduced in 1997, has become ubiquitous in pop music
  - Many different pitch shifting guitar pedals on the market
  - These effects often involve many careful optimisations for sound quality and efficiency

Revisiting robotisation

- The robotisation effect worked by setting all phases to 0 each hop
- The fundamental frequency of the effect was f_s/H
  - f_s is the sample rate, H is the hop size
  - For example, $f_s = 44100$, $H = 256$ yields a base frequency of 172Hz
- Limitation 1: H must be an integer
  - This means the frequency can't be finely tuned
- Limitation 2: H typically doesn't meet the COLA criterion
  - It's not an integer division of the FFT size
  - Non-COLA isn't a big deal since we don't need exact reconstruction, but it is a limitation
- Limitation 3: can't do other frequency transformations at the same time
  - Other phase vocoder effects will assume a more regular hop size
  - It's computationally expensive to run two simultaneous overlap-add FFT processes

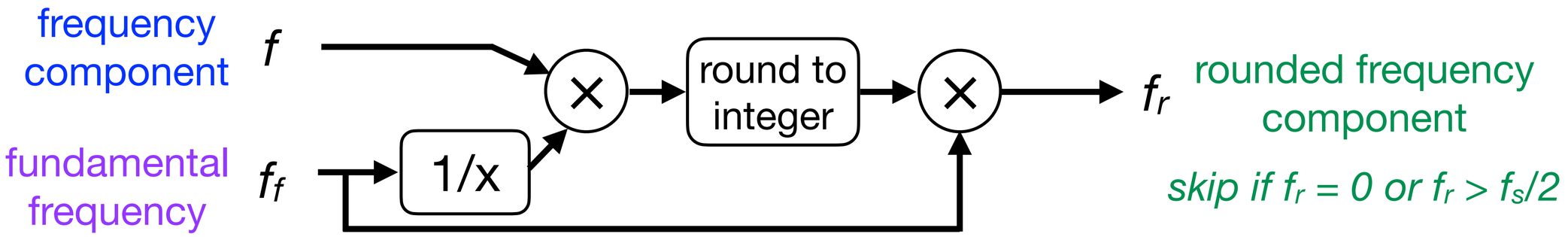
Revisiting robotisation

- How else could we achieve the robotisation effect?
  - Manipulate the frequency components with the phase vocoder
- The pitch shifter used a magnitude-frequency representation of the signal
  - The hard part was calculating that representation (and getting back again)
  - The pitch shift itself was simple: multiply by a constant
- Let's use the same magnitude-frequency representation for robotisation
  - Round each frequency to the nearest integer multiple of the desired fundamental
  - Keep the magnitudes of each frequency component constant



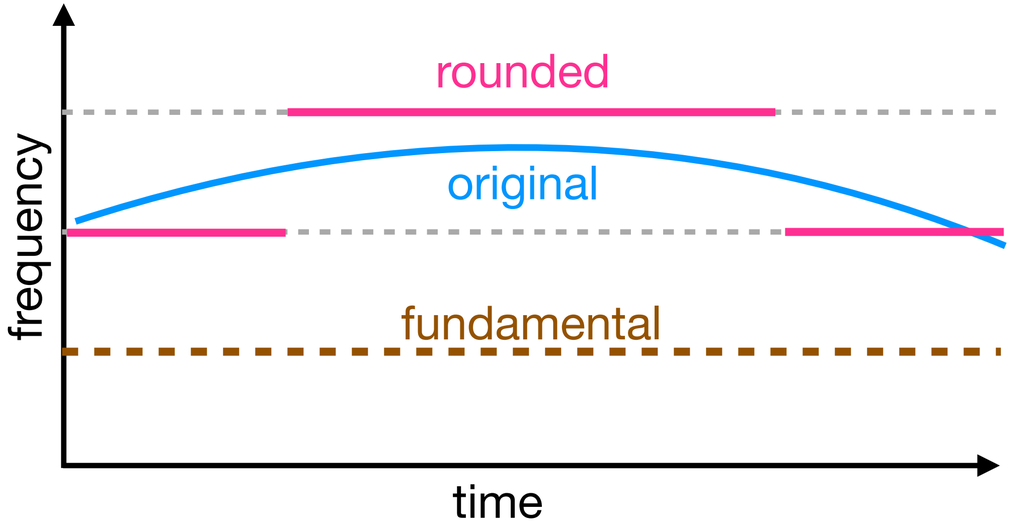
Revisiting robotisation: task

- Our previous projects expressed frequency in fractional bin numbers
- Here, it may be easier to think about actual (discrete time) frequency
  - In discrete time, frequency ranges from 0 to 2π (corresponding to the sample rate)
  - We need to rescale our target fundamental frequency by multiplying by $2\pi/f_s$
- Task: in project fft-robotisation-v2
  - Round the frequency of each bin to a multiple of the target fundamental
  - Remember that the rounded frequency might be in a different FFT bin
  - Keep magnitudes of each frequency component the same



Improving the robotisation effect

- You might have heard some artefacts
  - Frequency components glide smoothly up and down in the original signal
  - Now, they jump from one harmonic to another
  - The result is a trill-like effect



- Alternative approach: interpolation
  - For each frequency component, generate energy at harmonics above and below
  - Scale the magnitude of the upper and lower harmonics depending on which harmonic is closer to the original frequency
  - This is a form of linear interpolation (see Lecture 3 for more)
- Task: in fft-robotisation-v2, implement linear interpolation
  - Create two synthesis frequencies for each analysis frequency
  - Will often need to mix multiple components together in one FFT bin

Phase vocoder: more ideas

- Ways to improve the quality of the effects:
  - When mixing components into an FFT bin, use RMS amplitude rather than linear addition
  - When mixing components into an FFT bin, find the weighted average of the frequencies
  - Implement peak tracking to reduce some of the effects of window lobes
- Other effects using the phase vocoder:
  - Combine a frequency detector with a pitch shift to implement auto-tuning
  - Cross-synthesis: combine frequencies from one signal with magnitudes from another
  - Noise reduction: suppress frequency components below a threshold or matching a known profile
- As the effects become more complex, CPU becomes a limitation
  - Try larger hop sizes (which may also require larger FFTs)
  - Pre-calculate frequently used constants (e.g. bin frequencies)
  - Look for other optimisations to reduce the number of multiplies per hop
  - The Bela NEON library (`math_neon`) contains optimised versions of common functions

Keep in touch!

Social media:

@BelaPlatform

forum.bela.io

blog.bela.io

More resources and contact info at:

learn.bela.io/resources