Voice Shifting

Frequency Domain Processing with Tympan

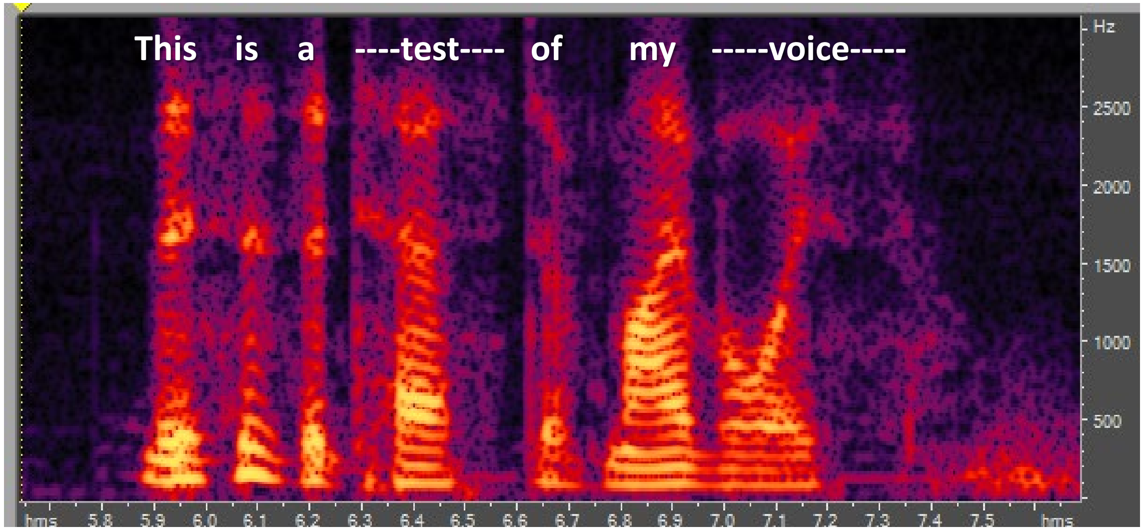
William (Chip) Audette
Creare LLC, Hanover, NH, USA

github.com/tympan
openaudio.blogspot.com

Time Domain vs Frequency Domain

Time-Domain Waveform

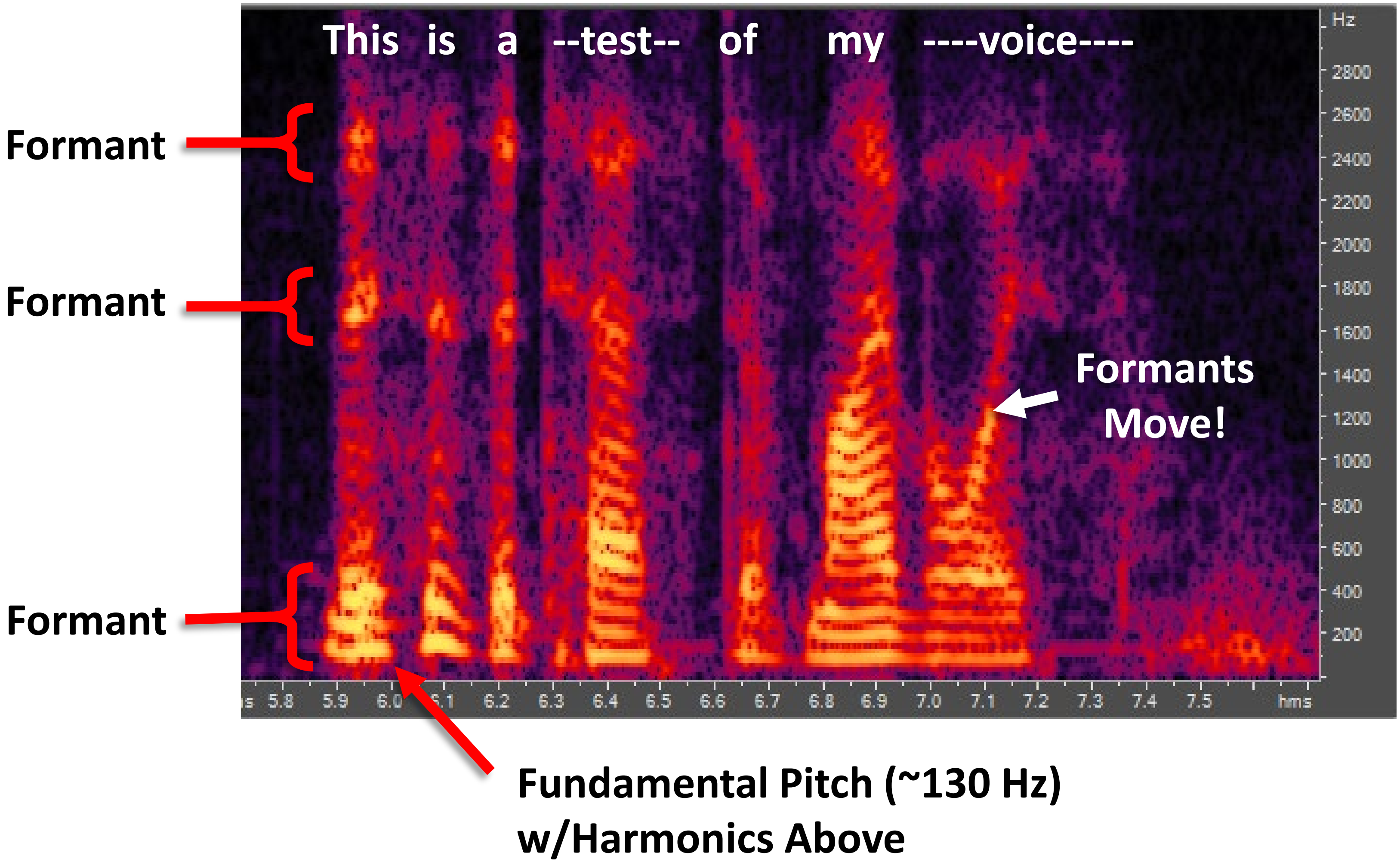
Easy to think in terms of amplitude-related processing



Spectrogram Shows Time and Frequency

Easy to think in terms of **frequency**-related processing

Frequency-Domain Features of Speech



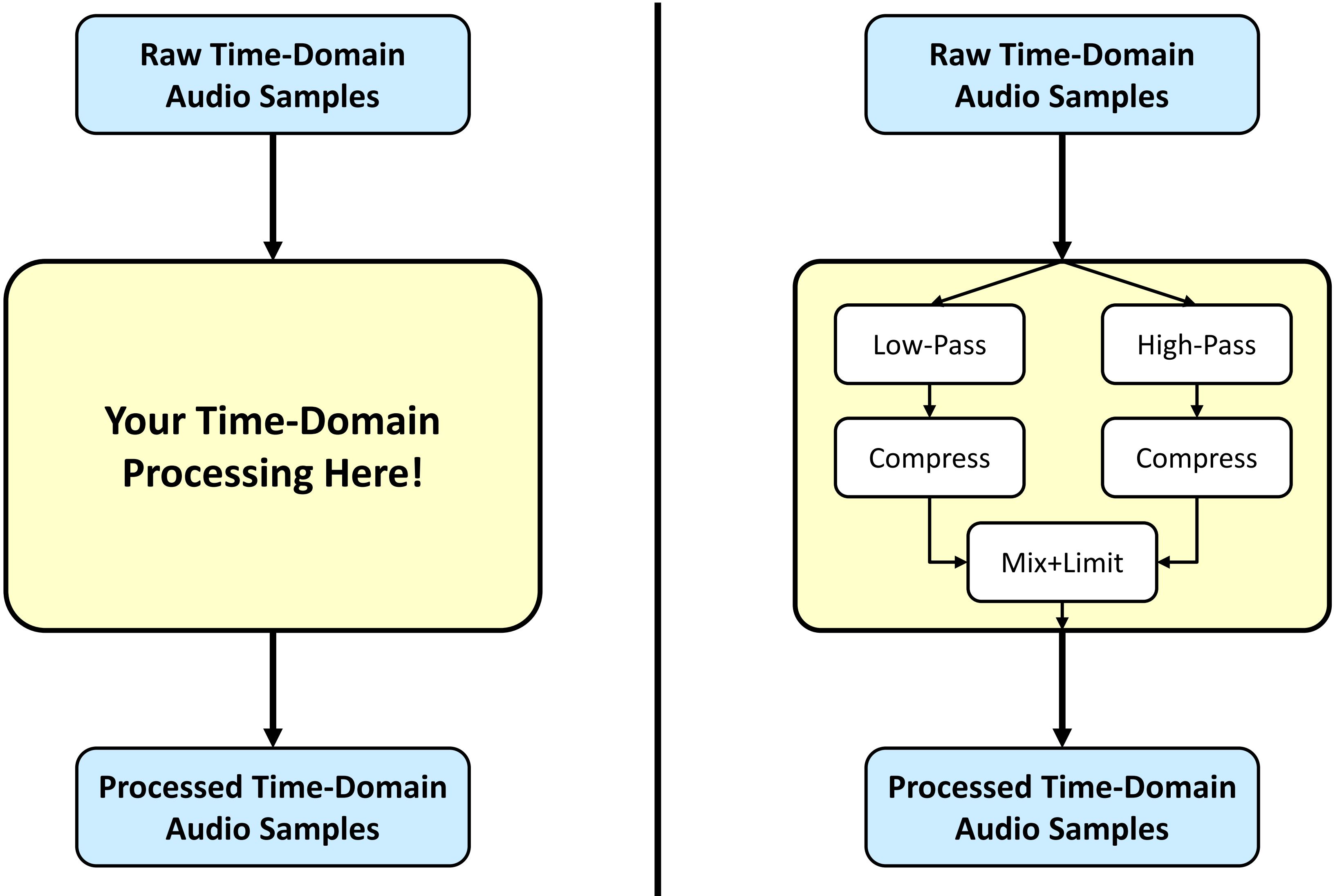
Processing in the Frequency Domain

- **What might you want to do?**
  - Shift the pitch to ease hearing
  - Shift the formants to ease hearing
  - Detect and amplify the speech (not the noise)
  - Separate simultaneous talkers (or birds, or heartbeats, or machinery)
- **How might you do it now? Post-Processing**
  - You would make recordings.
  - You would post-process them in Matlab/Python.
  - You would play processed recordings to human listeners for evaluation
- **How might you do it better? Real-Time Processing**
  - Implement candidate algorithms in real time (on Tympan or other)
  - Test live audio on humans in the laboratory
  - Test live audio with humans out in the world

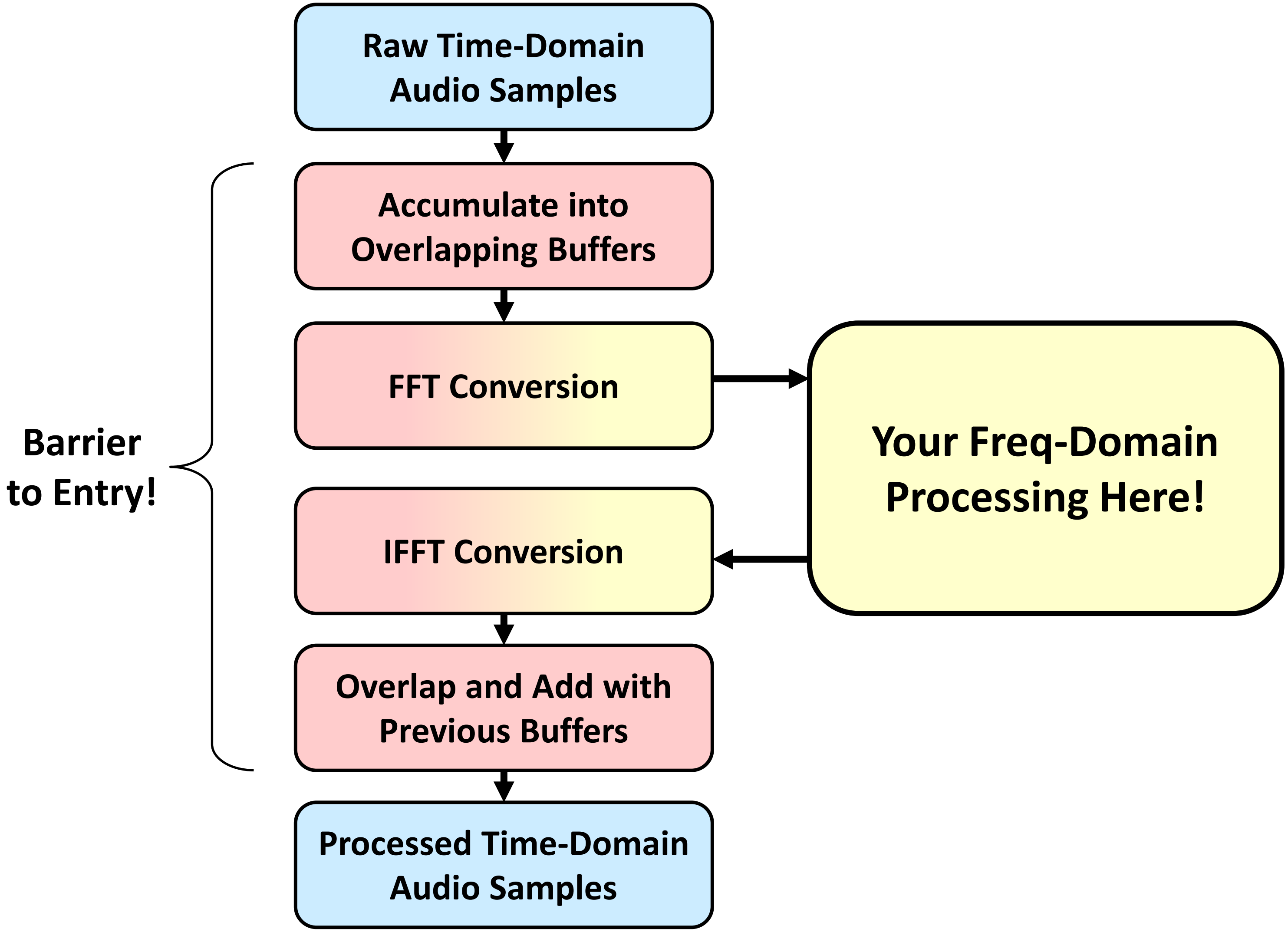
Transitioning to Real-Time

- **Time-domain algorithms are easier to transition**
  - Time-domain means processing your data sample-by-sample, which is natural since it is how the data arrives
  - Your processing modules (filters, gain, etc) are often the same whether post-processing or real-time
- **Frequency-domain algorithms are harder**
  - FFTs often need to be numerically optimized for your platform
  - Data is processed in blocks…typically overlapping blocks
  - Both amplitude and phase need to be managed

General Recipe of TD Processing

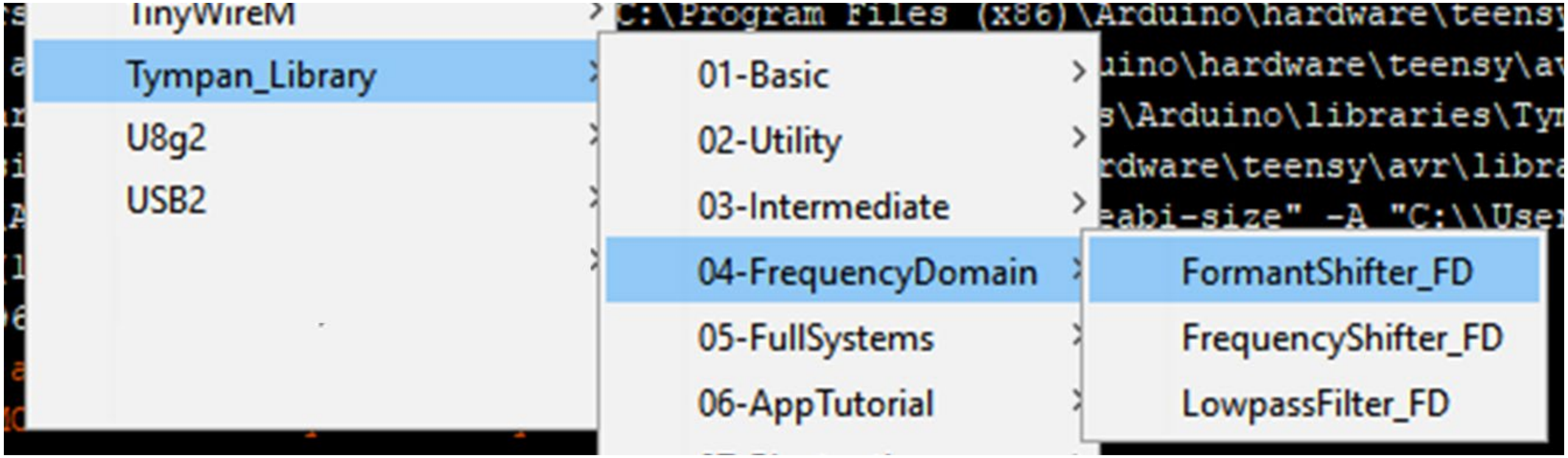


General Recipe for FD Processing



Tympan

Examples in Tympan Library

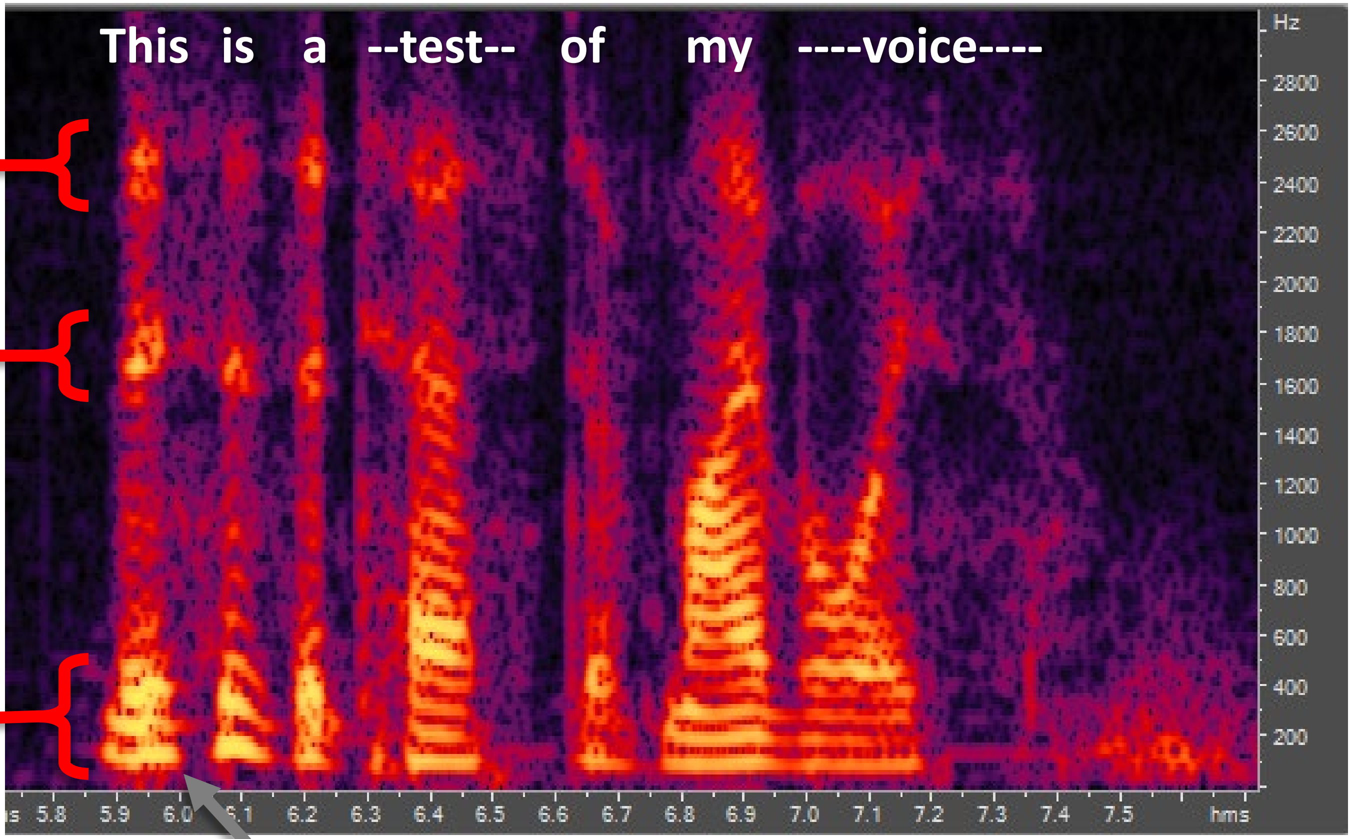


Formants

Formant

Formant

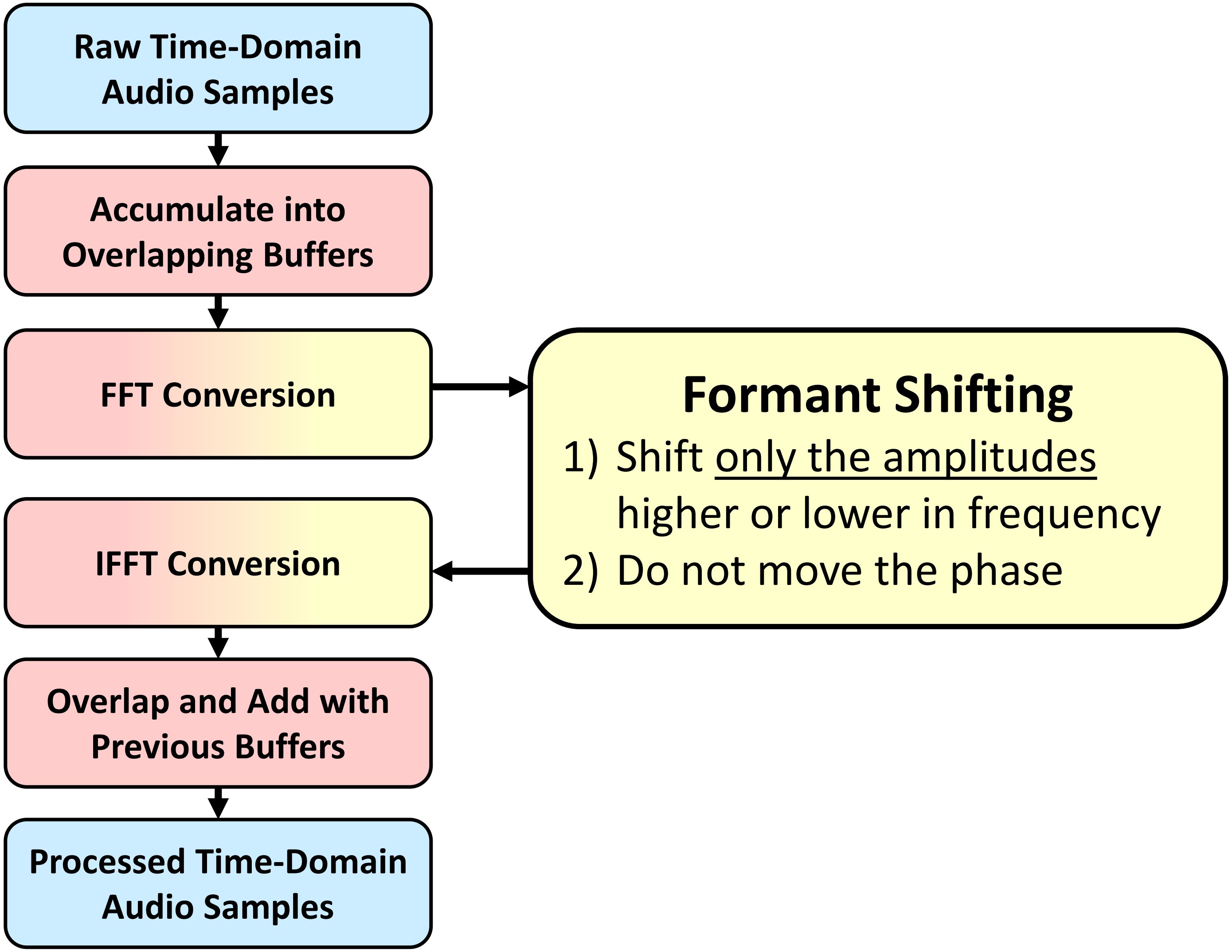
Formant



Fundamental Pitch (~130 Hz)
w/Harmonics Above

Tympan

Formant Shifting

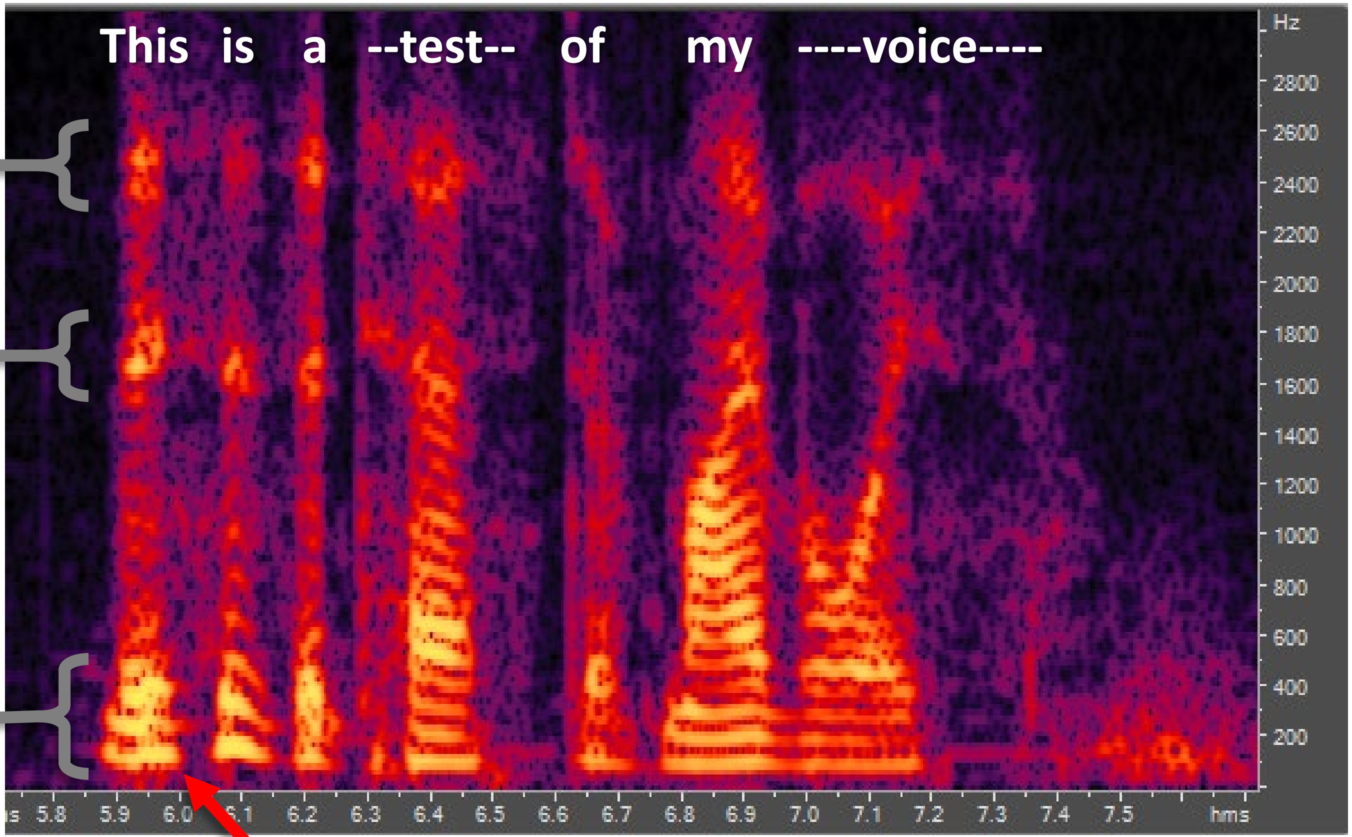


Fundamental Pitch (Frequency)

Formant

Formant

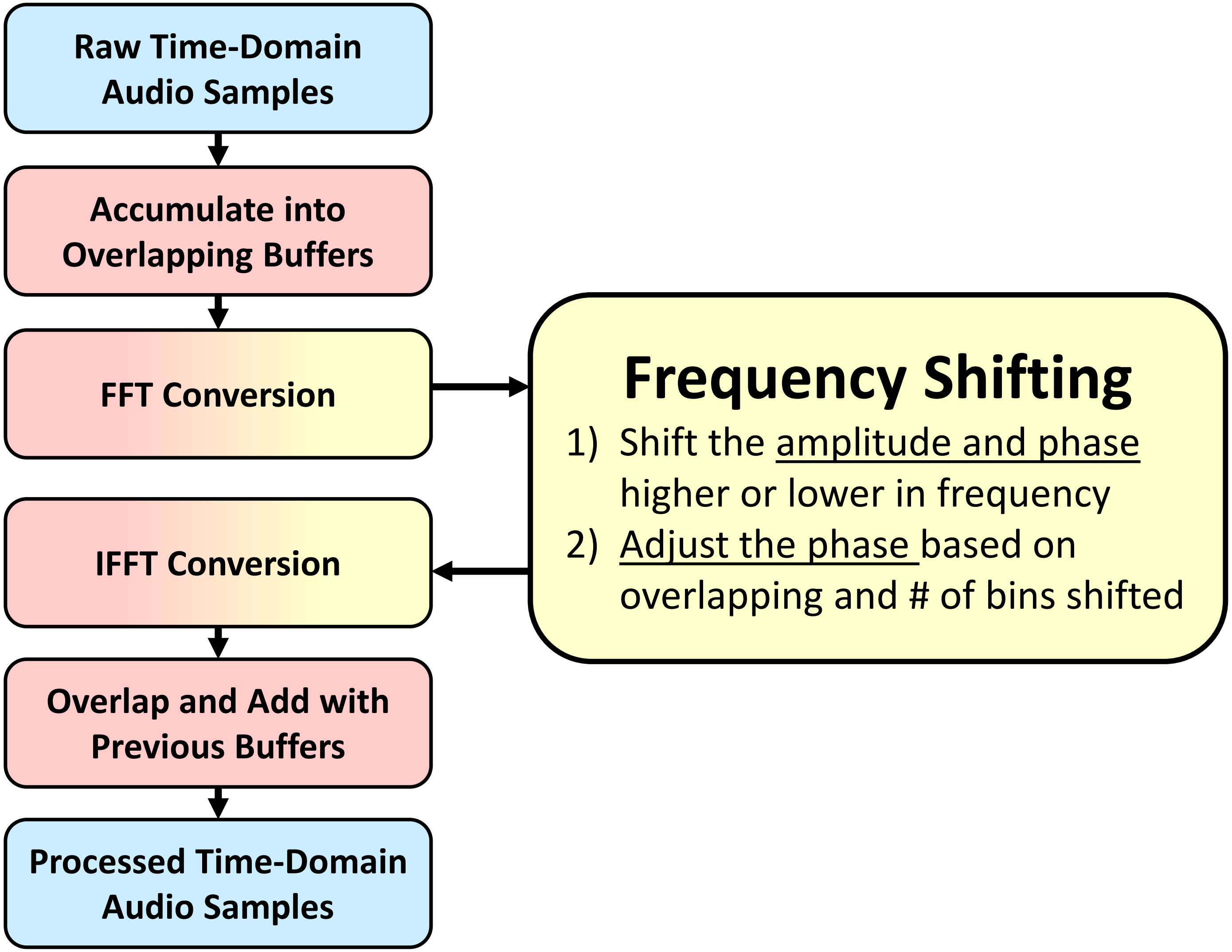
Formant



**Fundamental Pitch (~130 Hz)
w/Harmonics Above**

Tympan

General Recipe for FD Processing



Formant Filter Demo

https://www.youtube.com/watch?v=xmHOsX0pL0w

Tympan

Backup

Frequency Domain Filtering

LowpassFilter_FD | Arduino 1.8.13

File Edit Sketch Tools Help

LowpassFilter_FD § AudioEffectLowpassFD_F32.h

Specify

```
//set the sample rate and block size
const float sample_rate_Hz = 24000.f; ; //24000 or 44117 (or other frequencies in the table in AudioOutputI2S_
const int audio_block_samples = 32;     //for freq domain processing choose a power of 2 (16, 32, 64, 128) but
AudioSettings_F32 audio_settings(sample_rate_Hz, audio_block_samples);
Tympan audioHardware(TympanRev::D);   //TympanRev::D or TympanRev::E
```

Create

```
//create audio library objects for handling the audio
AudioInputI2S_F32          i2s_in(audio_settings);          //Digital audio *from* the Tympan AIC.
AudioEffectLowpassFD_F32   audioEffectLowpassFD(audio_settings);  //create the frequency-domain processing bloc
AudioOutputI2S_F32         i2s_out(audio_settings);         //Digital audio *to* the Tympan AIC.;
```

Connect

```
//Make all of the audio connections
AudioConnection_F32        patchCord1(i2s_in, 0, audioEffectLowpassFD, 0);    //connect the left input to the F
AudioConnection_F32        patchCord12(audioEffectLowpassFD, 0, i2s_out, 0);  //connect to the Left output
AudioConnection_F32        patchCord13(audioEffectLowpassFD, 0, i2s_out, 1);  //connect to the Right output
```


Tympan

Lowpass Filter Example

Get the Data

Do the FFT

Get the Cutoff Frequency Info

Attenuate the High Freq Bins

Do the IFFT

Done!

LowpassFilter_FD §   AudioEffectLowpassFD_F32.h §

```
void AudioEffectLowpassFD_F32::update(void)
{
  //get a pointer to the latest data
  audio_block_f32_t *in_audio_block = AudioStream_F32::receiveReadOnly_f32();
  if (!in_audio_block) return;

  //convert to frequency domain
  myFFT.execute(in_audio_block, complex_2N_buffer);
  AudioStream_F32::release(in_audio_block);  //We just passed ownership to myFFT, so release it here.

  // ////////////// Do your processing here!!!

  //this is lowpass, so attenuate the bins above the cutoff freq
  int NFFT = myFFT.getNFFT();
  int nyquist_bin = NFFT/2 + 1;
  float bin_width_Hz = sample_rate_Hz / ((float)NFFT);
  int cutoff_bin = (int)(lowpass_freq_Hz / bin_width_Hz + 0.5); //the 0.5 is so that it rounds instead of truncates
  if (cutoff_bin < nyquist_bin) {

    //how much do you want to attenuate these high frequency bins?
    float gain_dB = -30.0;
    float new_gain = sqrt(powf(10.0, gain_dB /10.0f));

    //only loop over the high frequency bins
    for (int i=cutoff_bin; i < nyquist_bin; i++) {
      complex_2N_buffer[2*i] = new_gain * complex_2N_buffer[2*i];      //real
      complex_2N_buffer[2*i+1] = new_gain * complex_2N_buffer[2*i+1];  //imaginary
    }
  }
  myFFT.rebuildNegativeFrequencySpace(complex_2N_buffer); //set the negative frequency space based on the positive

  // ///////////// End do your processing here

  //call the IFFT
  audio_block_f32_t *out_audio_block = myIFFT.execute(complex_2N_buffer); //out_block is pre-allocated in here.

  //send the returned audio block.  Don't issue the release command here because myIFFT will re-use it
  AudioStream_F32::transmit(out_audio_block); //don't release this buffer because myIFFT re-uses it within its own code
  return;
};
```