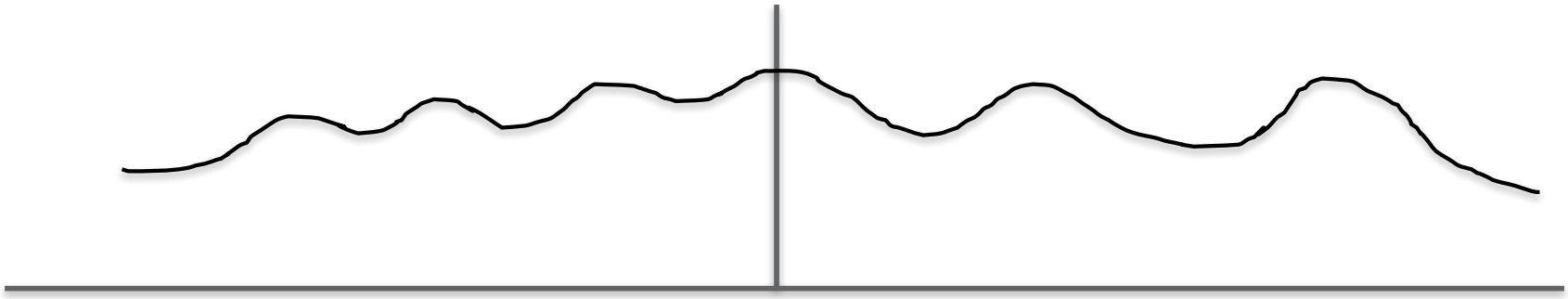
Lab 4
More background

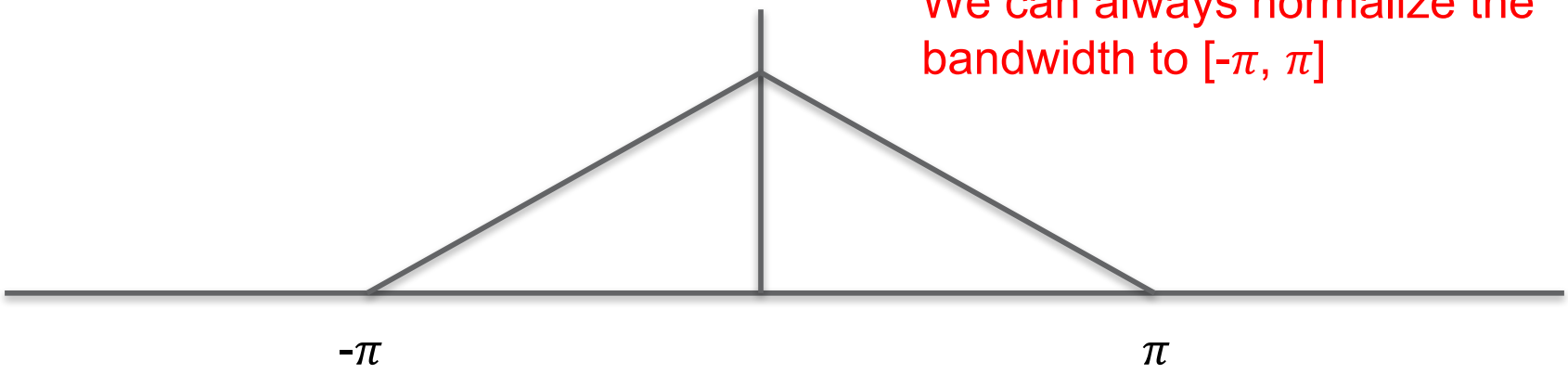
Signal and Spectrum (Review)

- **A continuous time signal *c(t)***



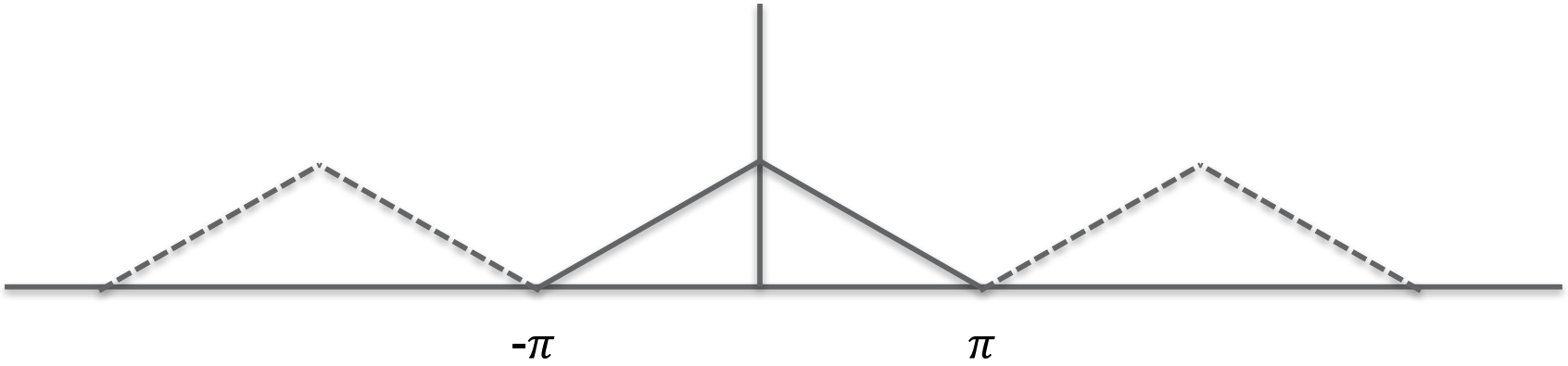
- **A lowpass spectrum**

We can always normalize the bandwidth to [$-\pi$, π]



Fourier Series Representation of a Lowpass Spectrum

- **A purely lowpass spectrum can be represented by a Fourier Series via periodic extension**



- **The coefficients of the FS representation turn out to be samples of the waveform, *c(t)*, acquired at the Nyquist frequency**

$$C(\omega) = \sum_{n=-\infty}^{\infty} c(-n)e^{i\omega n} = \sum_{n=-\infty}^{\infty} c(n)e^{-i\omega n}$$

This is the Nyquist sampling theorem!
(Why? becauase we can reconstruct *c(t)* from its spectrum, i.e., its FS)

Discrete Time Fourier Transform (DTFT)

- **Therefore, it makes sense to define the DTFT of a discrete-time signal as**

$$C(\omega) = \sum_{n=-\infty}^{\infty} c(n) e^{-i\omega n}$$

- **The corresponding signal, *c(t)*, can be recovered from *c(n)* by an ideal lowpass filter. The LPF:**
  - "Blocks" the replicated spectra created by the FS
  - Converts the discrete-time samples to a continuous time signal

The LPF process is referred to as sinc function interpolation (the impulse response of an LPF is a sinc function)

DFT / FFT Review

- **We saw that the DTFT of a sequence is given by**

$$C(\omega) = \sum_{n=-\infty}^{\infty} c(n) e^{-i\omega n}$$

- **The DFT gives samples of the DTFT of a finite-length sequence**

$$C(\omega_k) = \sum_{n=0}^{N-1} c(n) e^{-i(2\pi k/N)n} \qquad \text{where} \qquad \omega_k = \frac{2\pi k}{N}$$

- **The FFT is a fast algorithm for computing the DFT**

The FFT can therefore be used to approximate the spectrum of a continuous-time signal by using a finite-number of samples

Digital Convolution

- **We convolve two digital sequences by "flip and drag"**
  - Convolve (x_0, x_1, x_2, x_3, ...) with (c_0, c_1, c_2)
  - First "flip" the *x(n)* sequence
    - ✓ (..., x_3, x_2, x_1, x_0)
  - Then "drag" it past the *c(n)* sequence, computing dot products at each step to get the output sequence, *y(n)*

Step one

$$\begin{array}{cccc} \ldots x_3 & x_2 & x_1 & x_0 \\ & c_0 & c_1 & c_2 \end{array}$$

Compute dot product
$y_0 = (x_2, x_1, x_0) \cdot (c_0, c_1, c_2)$

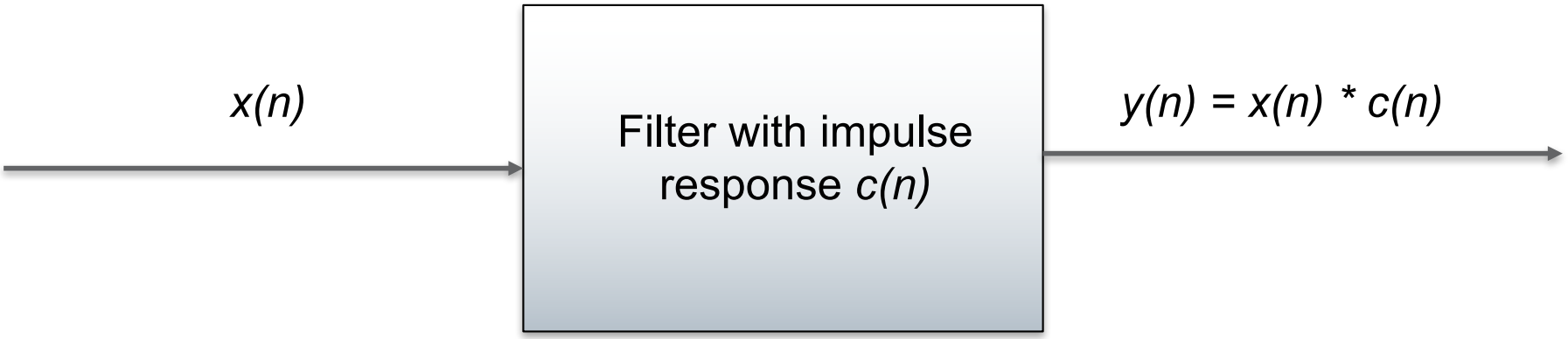
Step two

$$\begin{array}{cccc} \ldots x_4 & x_3 & x_2 & x_1 \\ & c_0 & c_1 & c_2 \end{array}$$

Compute dot product
$y_1 = (x_3, x_2, x_1) \cdot (c_0, c_1, c_2)$

Filter Review

- **Block diagram of a filter**

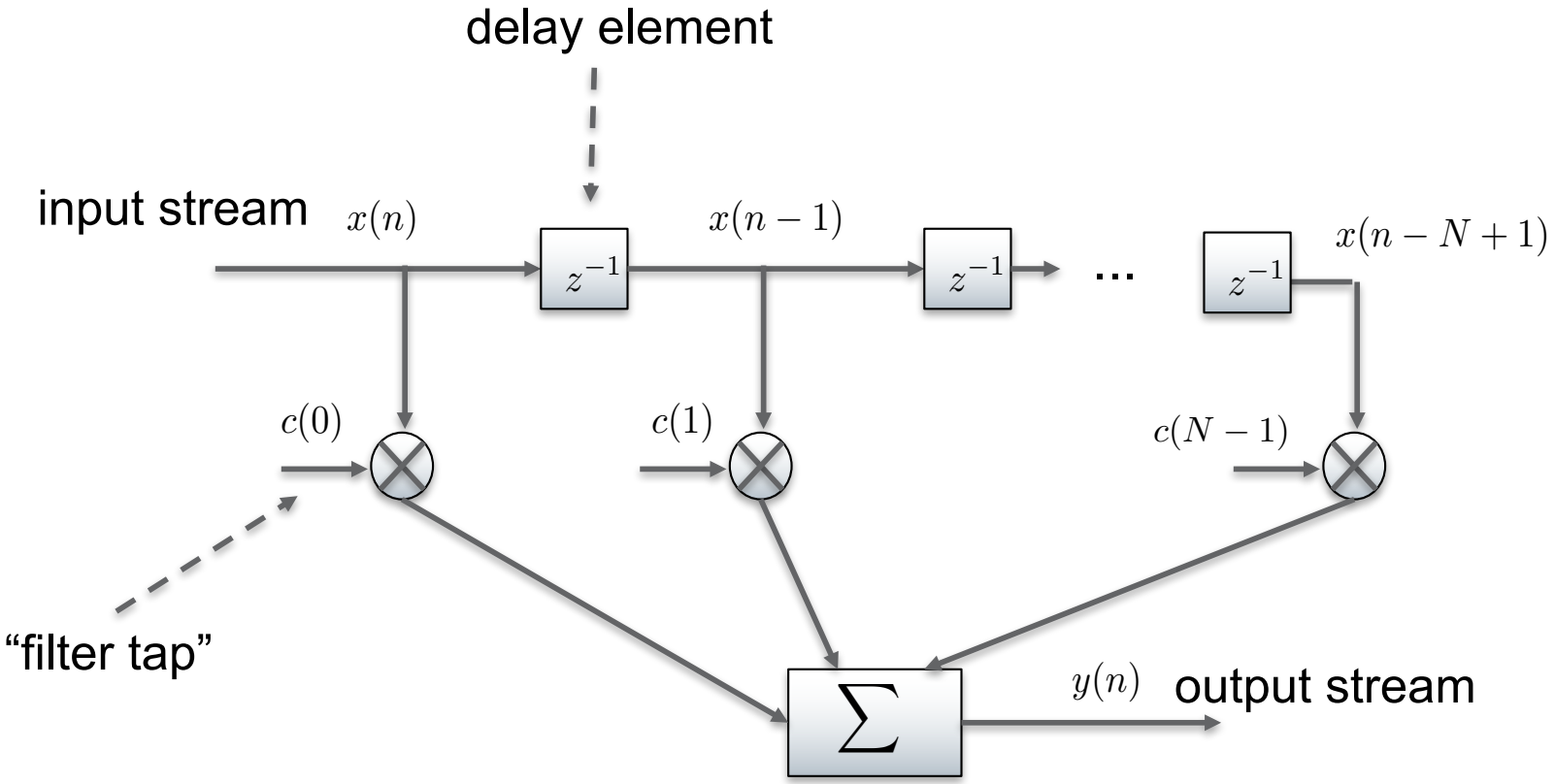


The choice of *c(n)* determines the frequency response of the filter

$$C(\omega) = \mathrm{D}TFT(c(n))$$

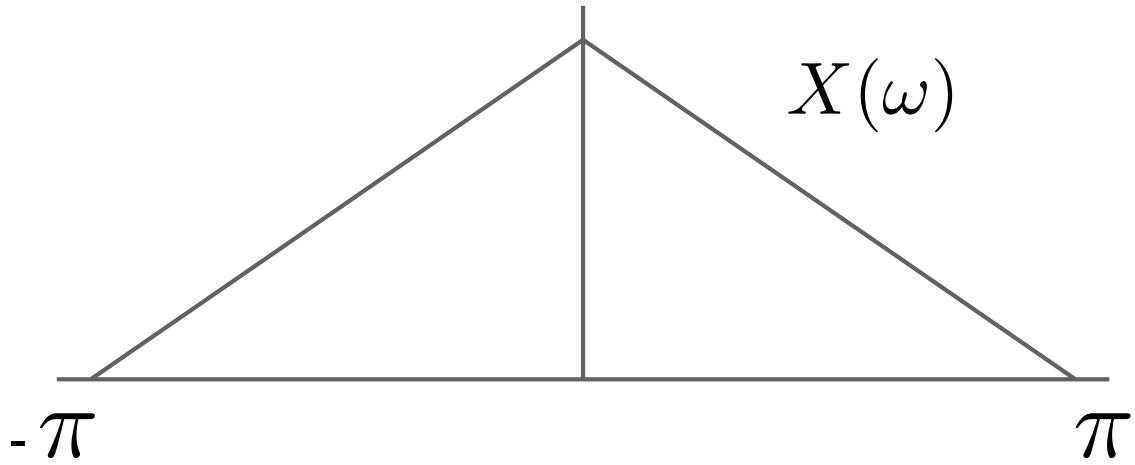
FIR filter

- **The circuit below convolves *x(n)* with *c(n)***
  - It implements a Finite Impulse Response (FIR) filter



Spectrum (DTFT) of an audio signal

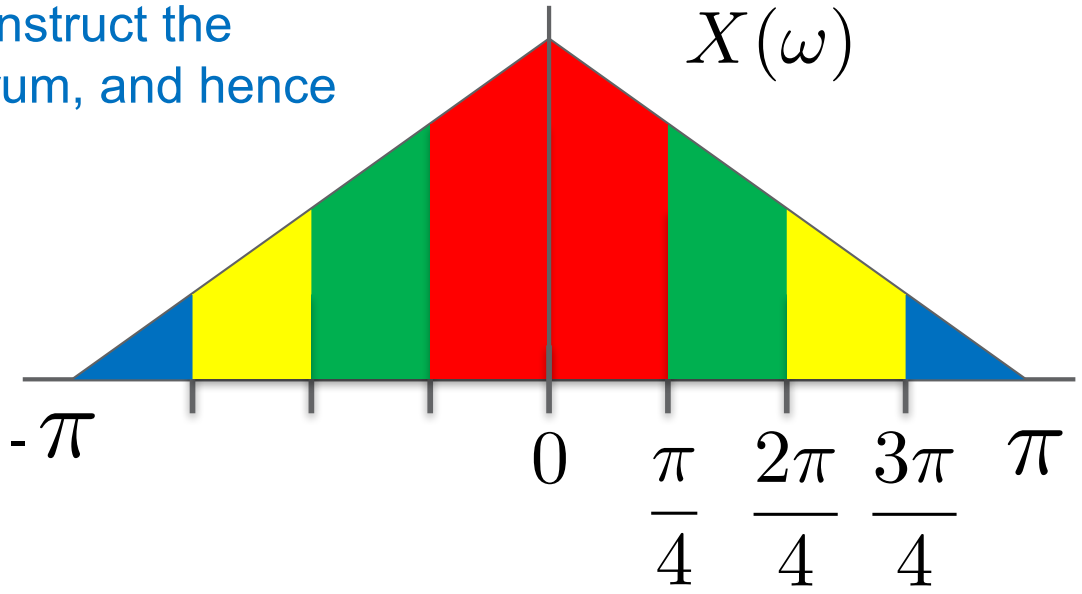
- **Assume our digital audio signal, *x(n)*, has a DTFT spectrum as shown below**



Divide spectrum into bandpass chunks

- **We can divide its spectrum into bandpass chunks (here we divide into 4 pieces)**

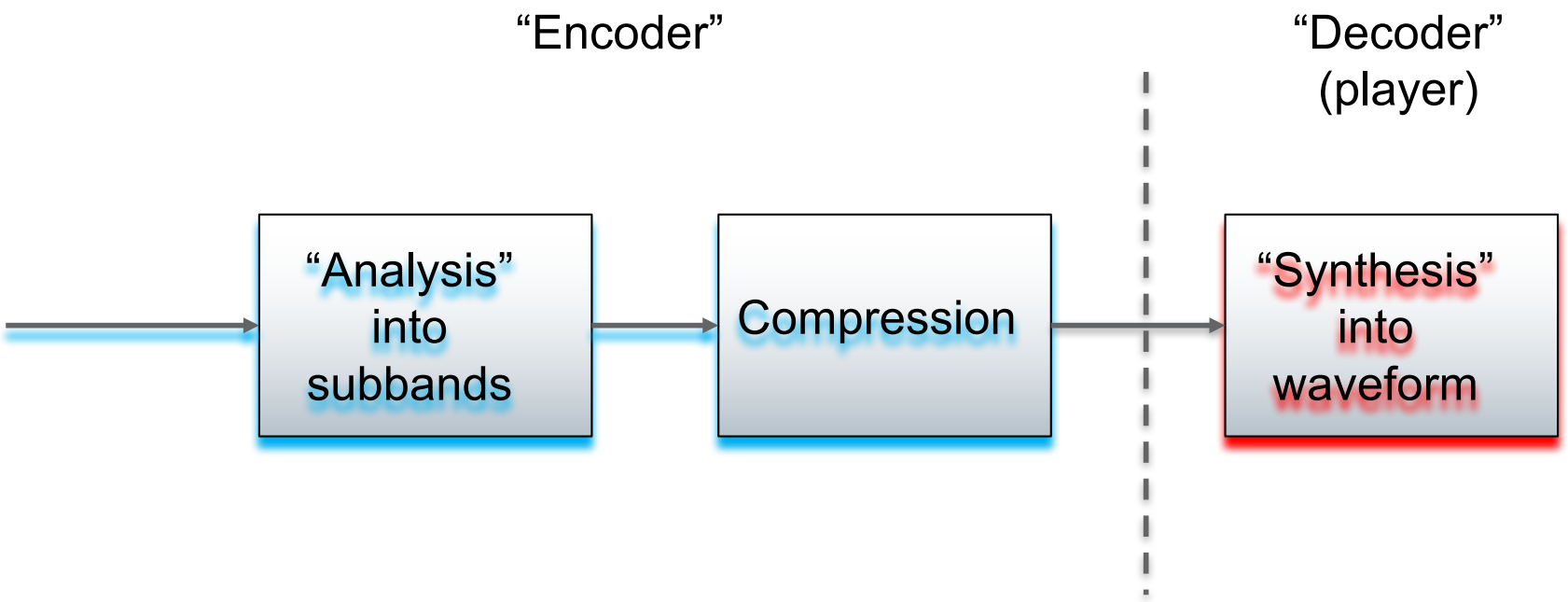
If we know each “chunk” we can reconstruct the entire spectrum, and hence the signal!



Why consider sub-bands for audio?

- **Some subbands matter more than others**
  - Some carry more energy
  - The ear is more sensitive to some frequencies than others
  - The subbands are not equally important perceptually!
    - We can tolerate more error in some subbands than others when listening to music
    - It allows us to allocate our quantizer bits where they will do the most good

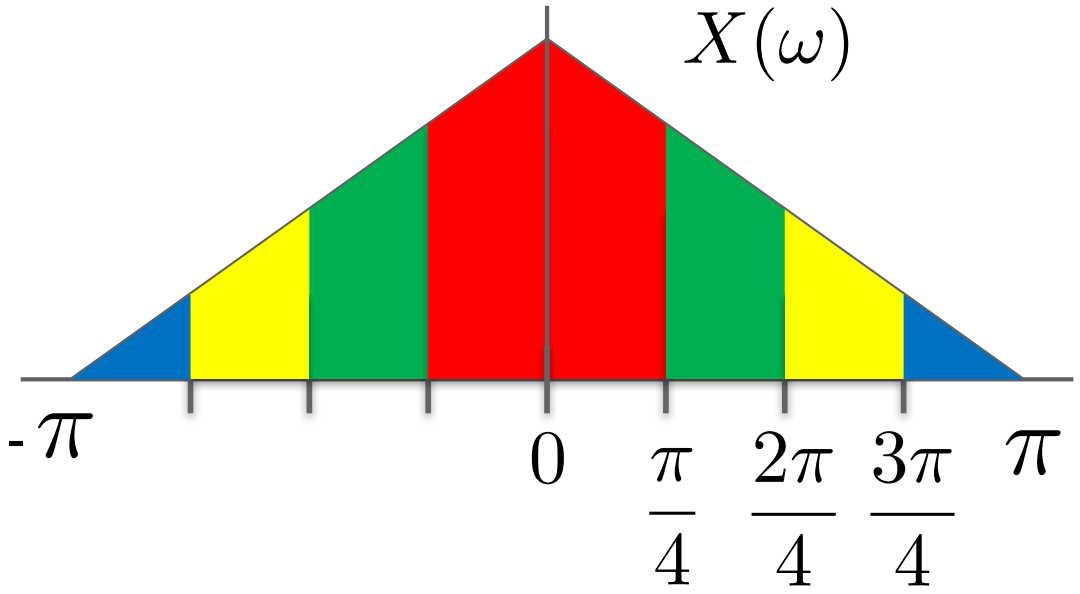
High-level MP3 system blocks



The encoder “analyzes” a signal into subbands, compresses subbands by quantizing, and creates an MP3 file. The “decoder” reads the mp3 file, “synthesizes” the subbands back into a waveform, and plays it for the listener.

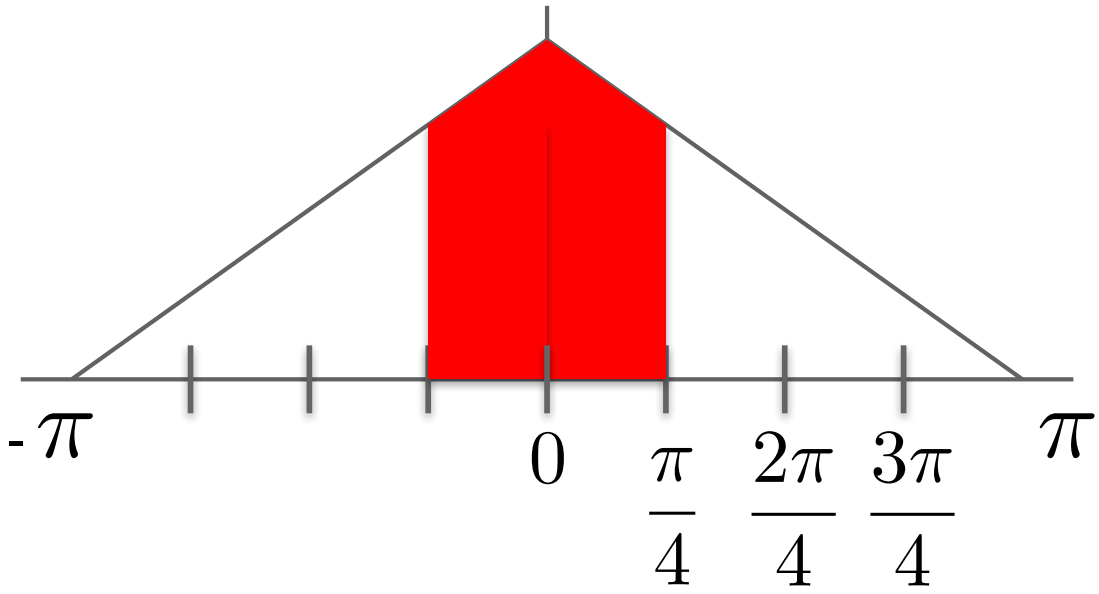
"Analysis" with DSP filter banks

- **Assume we want to digitally create the following bandpass chunks of our audio signal**



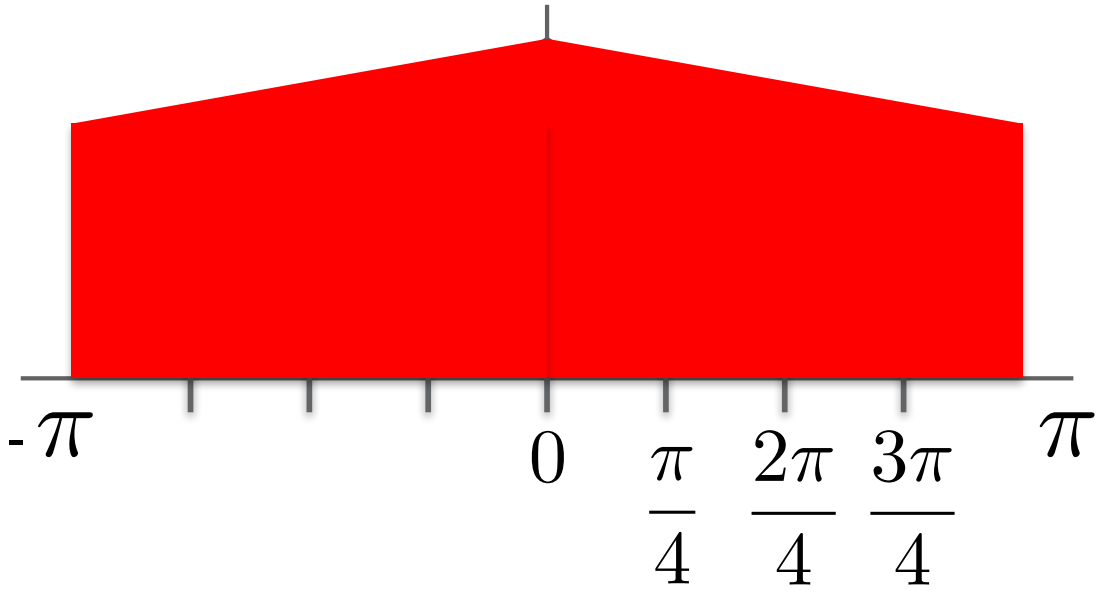
Bandpass filter “red” band

- **Filter to retain only the “red” band**



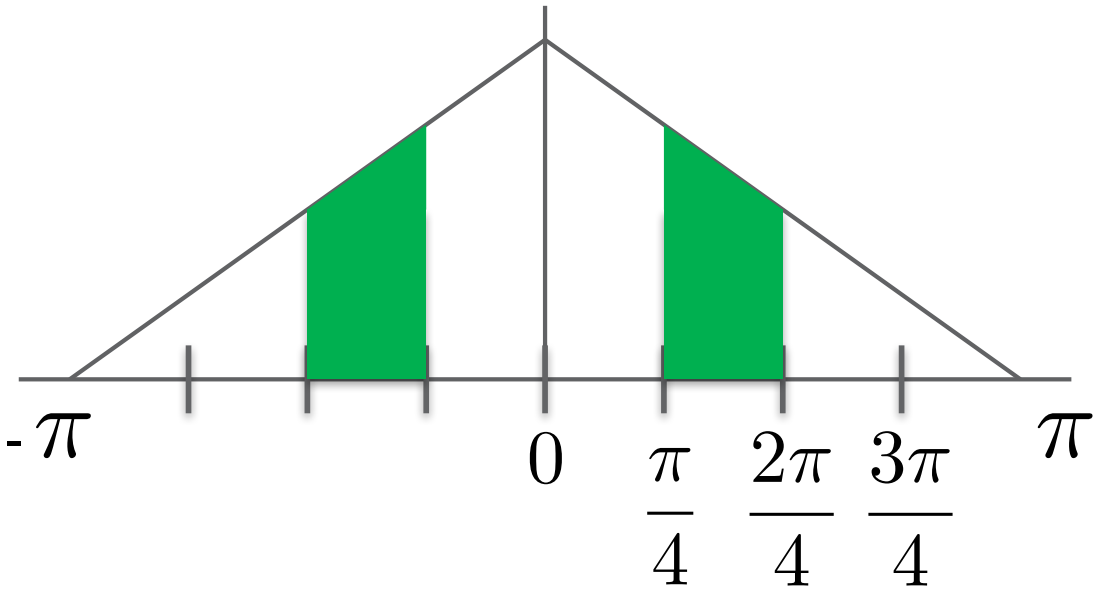
Subsample “red” BPF signal by 4

- **The subsample, which ”stretches” the spectrum by a factor of 4 to occupy the range -π to π**



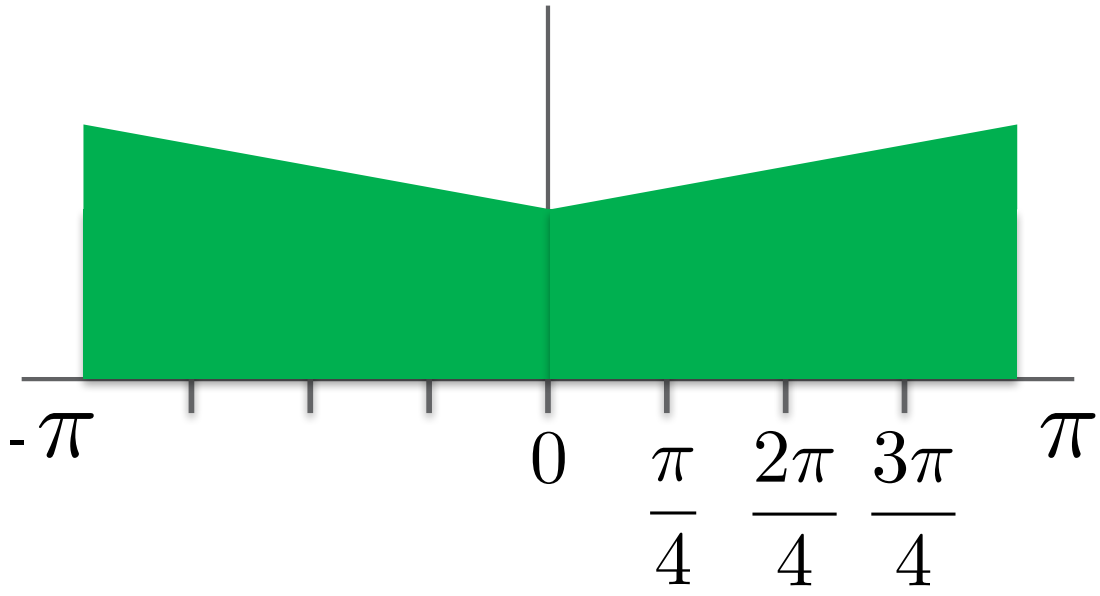
Next, bandpass filter “green” band

- **Filter to retain only the “green” band**



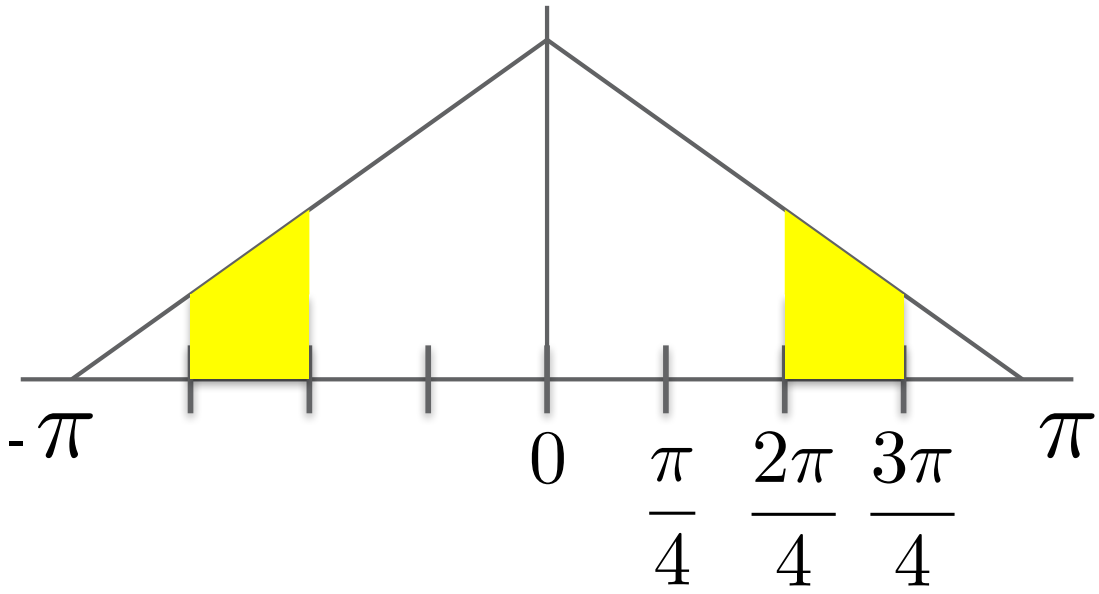
Subsample "green" BPF signal by 4

- **After sub-sampling, the stretched spectrum appears between the range -π to π but it is <u>frequency inverted</u>**
  - We can "fix" the frequency inversion later



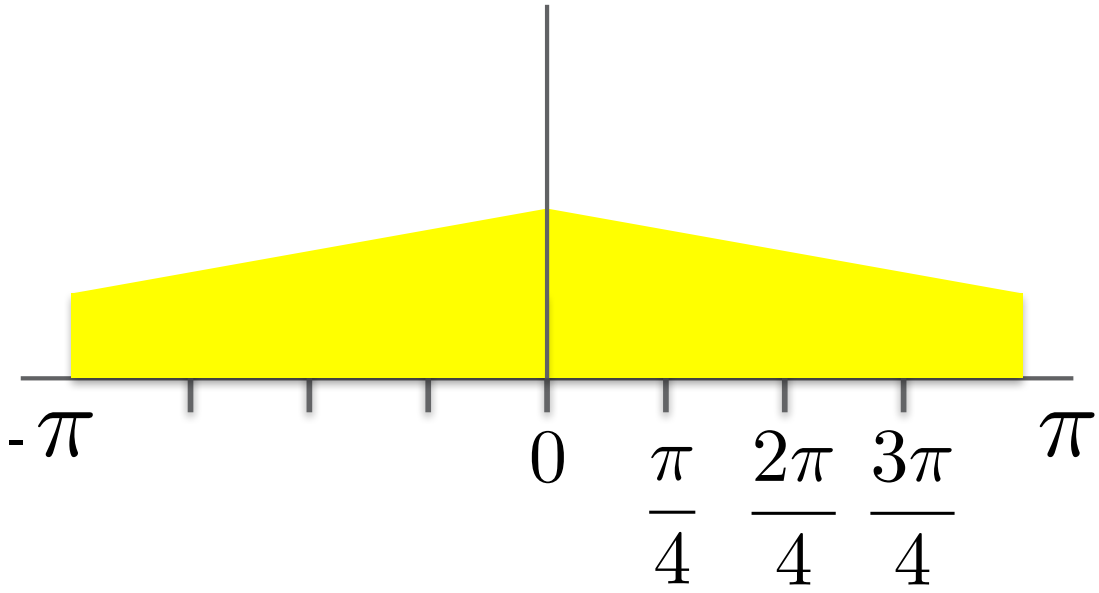
Bandpass filter ”yellow” band

- **Filter to retain only the “yellow” band**



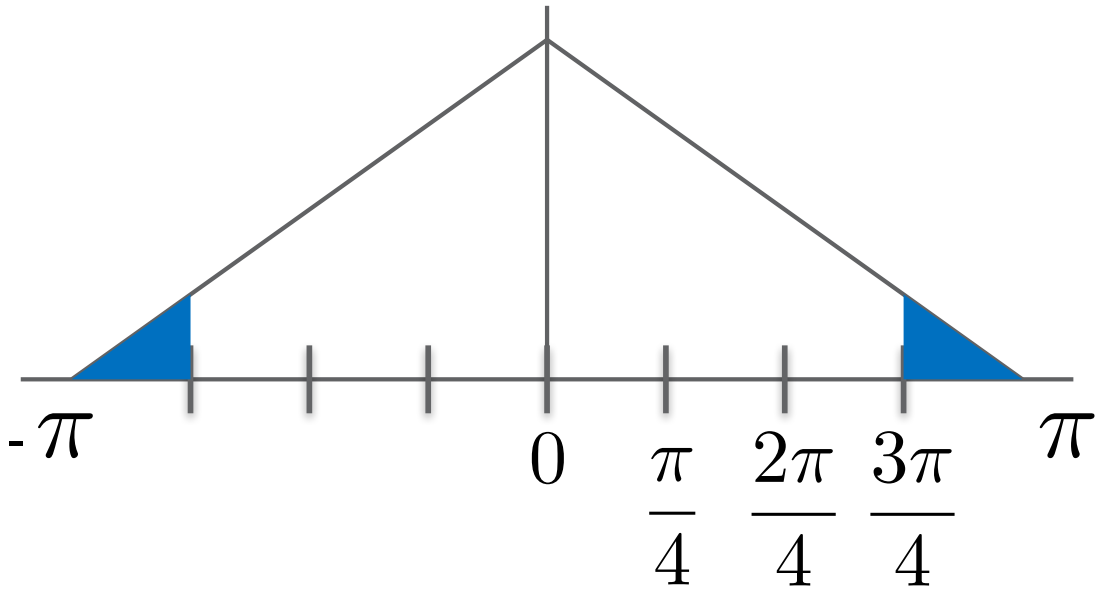
Subsample “yellow” BPF signal by 4

- **The spectrum stretches by a factor of 4 to occupy the range -π to π**



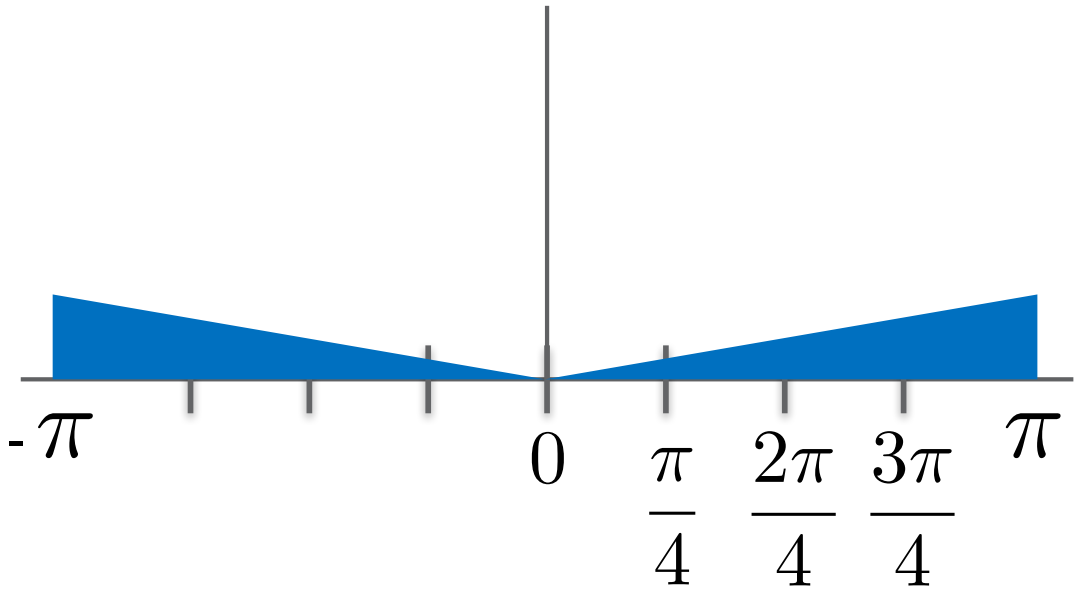
Bandpass filter ”blue” band

- **Filter to retain only the “blue” band**



Subsample “blue” BPF signal by 4

- **The stretched spectrum appears between the range -π to π but it is again frequency inverted**



Where do the stretching and inversions come from?

- **To understand why the spectrum spreads out, and sometimes frequency inverts, we need to look at the math of sub-sampling**
- **The DTFT of sequence *x(n)* is given by**

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-i\omega n}$$

Downsampling

- **Downsampling by N means taking every N'th sample**
  - The retained samples of a signal downsampled by 3 are shown below

$$\{x_0, x_3, x_6, \ldots\}$$

After down-sampling, the spectrum is a summation of stretched and translated versions of the original $X(\omega)$

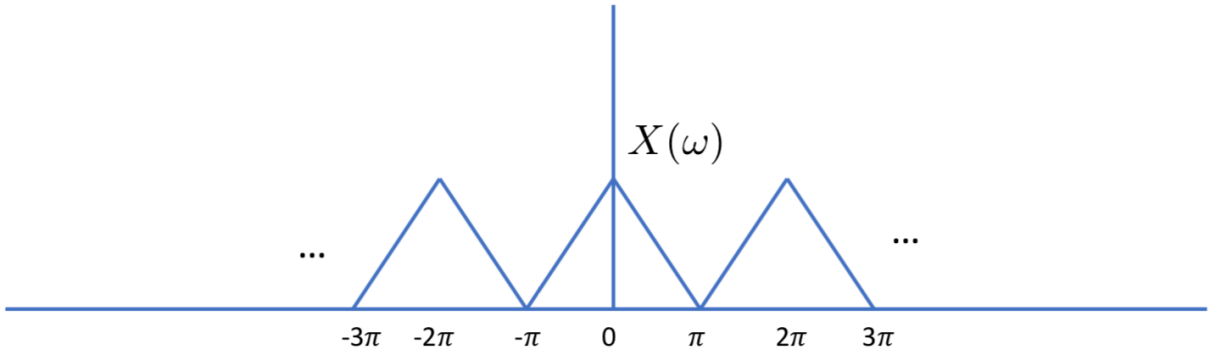
$$\frac{1}{N}\sum_{p=0}^{N-1} X\left(\frac{\omega - 2\pi p}{N}\right)$$

Downsampling discussion

- **The N in the denominator of each term "stretches" (spreads out) the original spectrum by a factor of N**
  - The stretched spectra has periodicity of $2N\pi$
- **Replicas of the stretched spectra appear at $2\pi p$, and there are N of them in total (p = 0, 1, ..., N-1)**

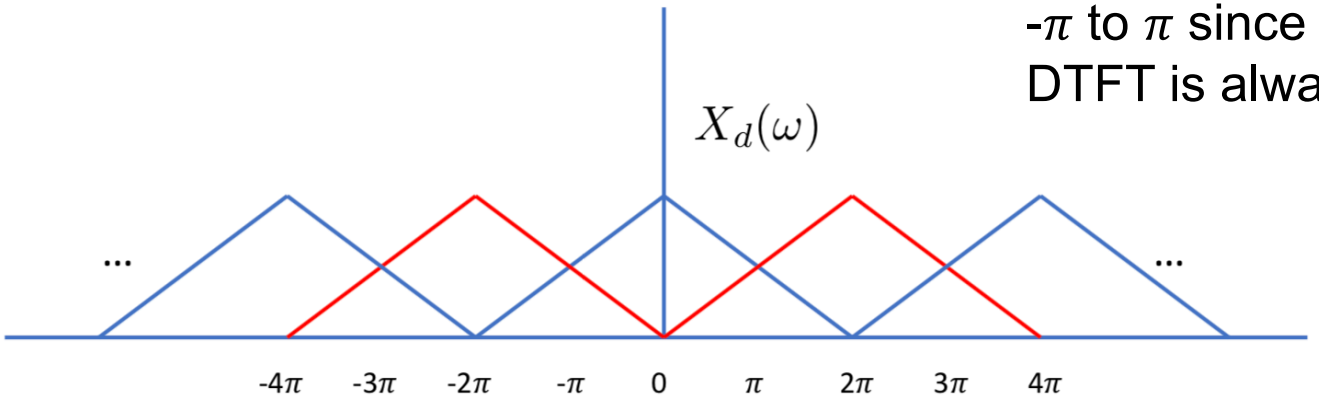
The easiest case: downsampling by 2

- **Spectrum before downsampling by 2**



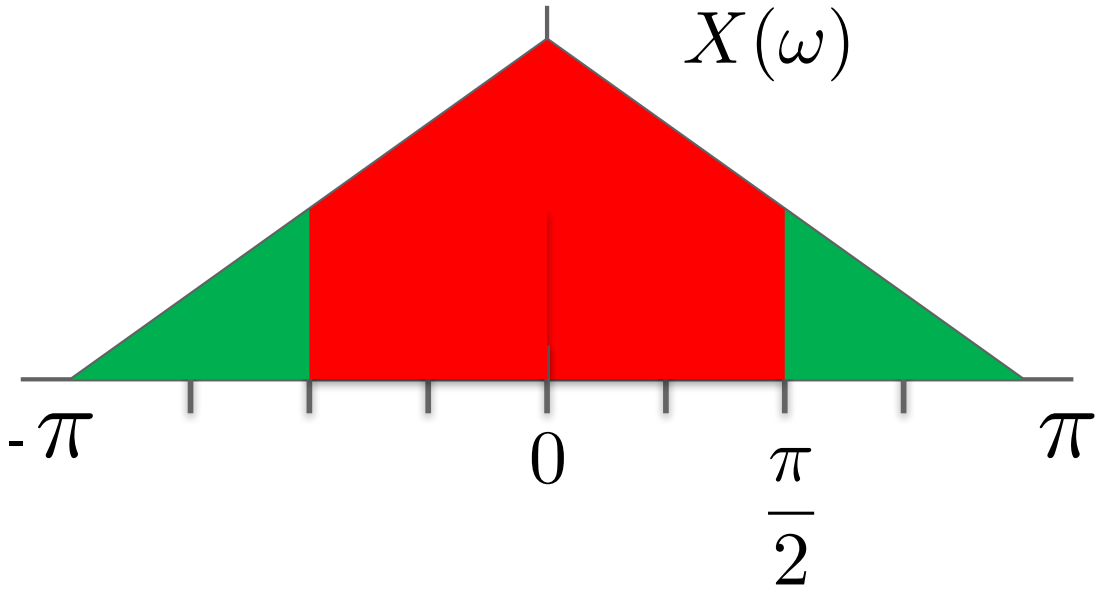
- **Spectrum after downsampling by 2 is the sum of the 2 replicas below**

We focus on the range $-\pi$ to π since the final DTFT is always 2π periodic



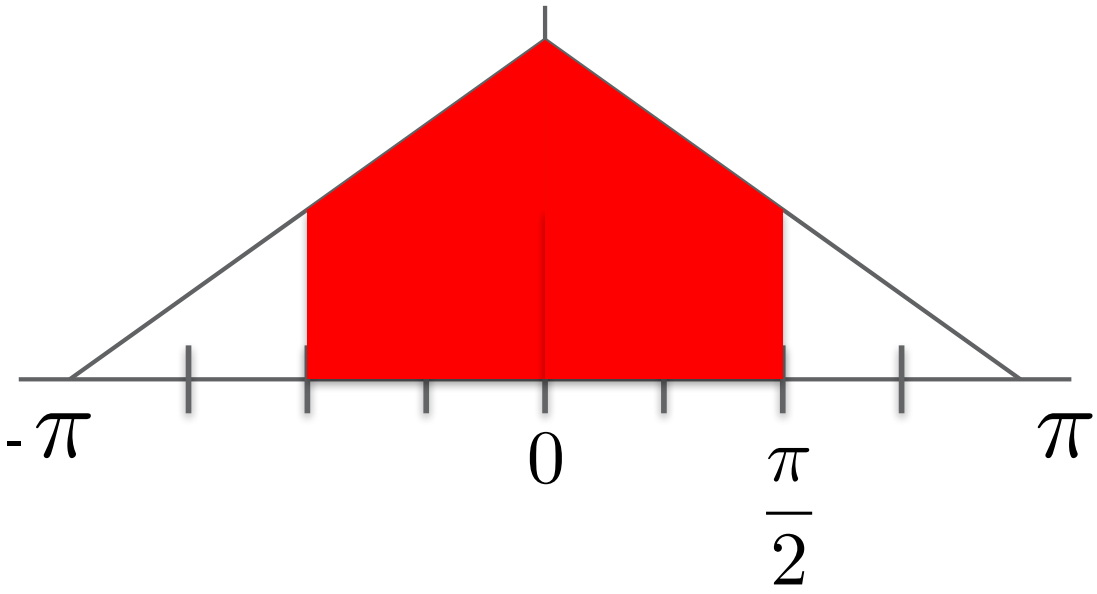
Divide spectrum into two chunks

- **We can divide the spectrum into bandpass chunks**



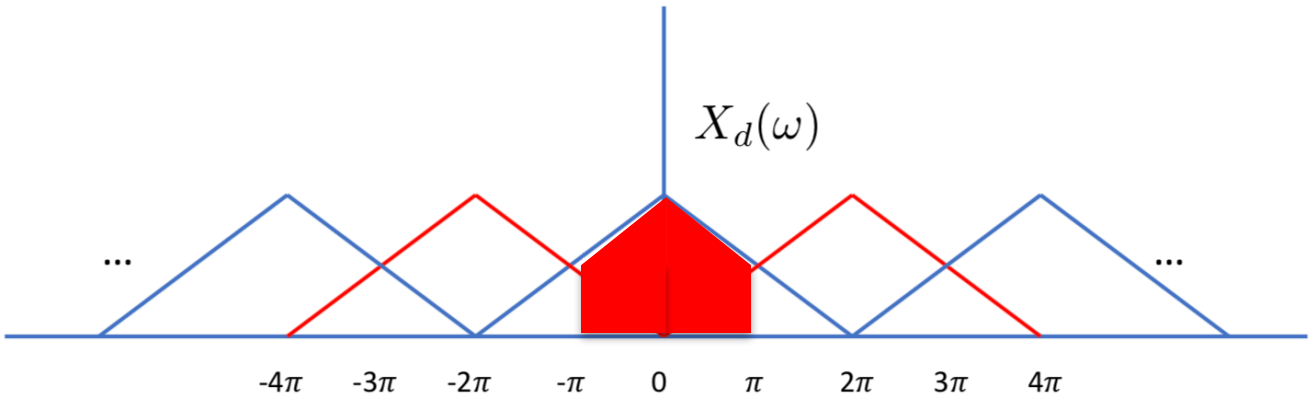
If filter to retain only “red”

- **BPF to retain only the ”red” piece**



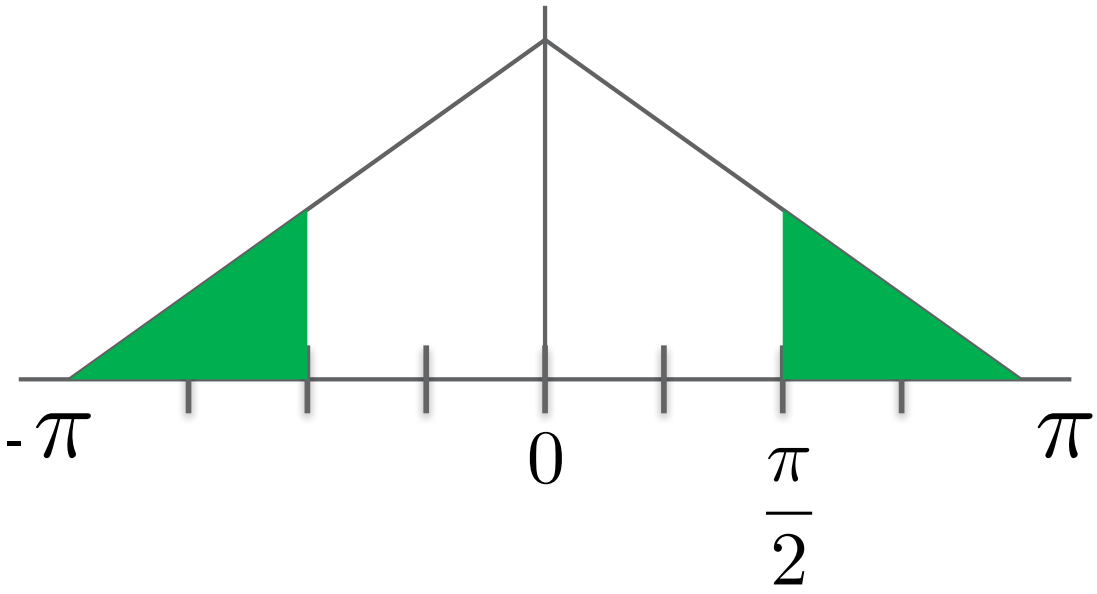
After downsampling "red" BPF signal

- **Where are the red pieces after downsampling?**
  - No "green" pieces appear between -π and π because they were removed prior to downsampling by the filter
  - There is NO frequency inversion



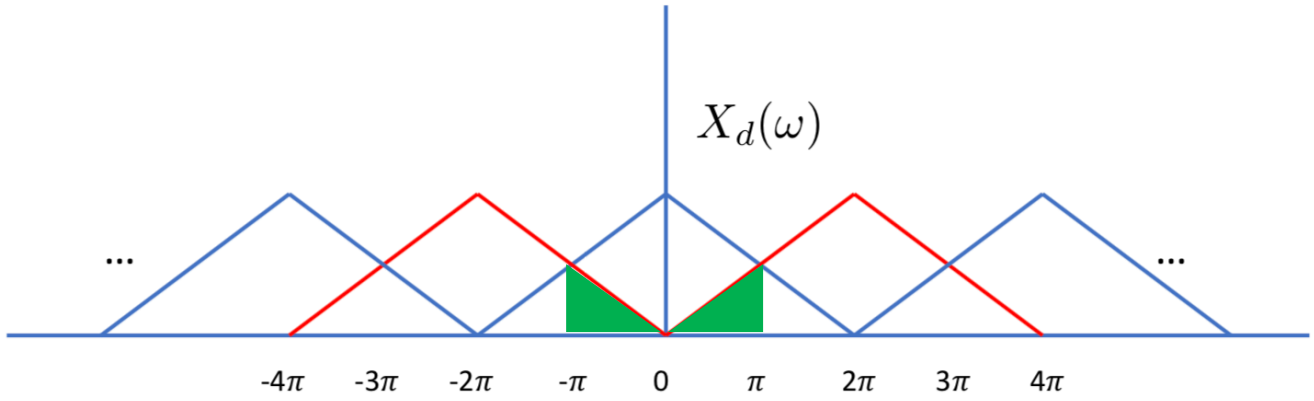
If filter to retain only “green”

- **BPF to retain only the ”green” piece**



After downsampling "green" BPF signal

- **Where are the green pieces after downsampling?**
  - No "red" pieces appear between -π and π because they were removed prior to downsampling by the filter
  - There is a frequency inversion!

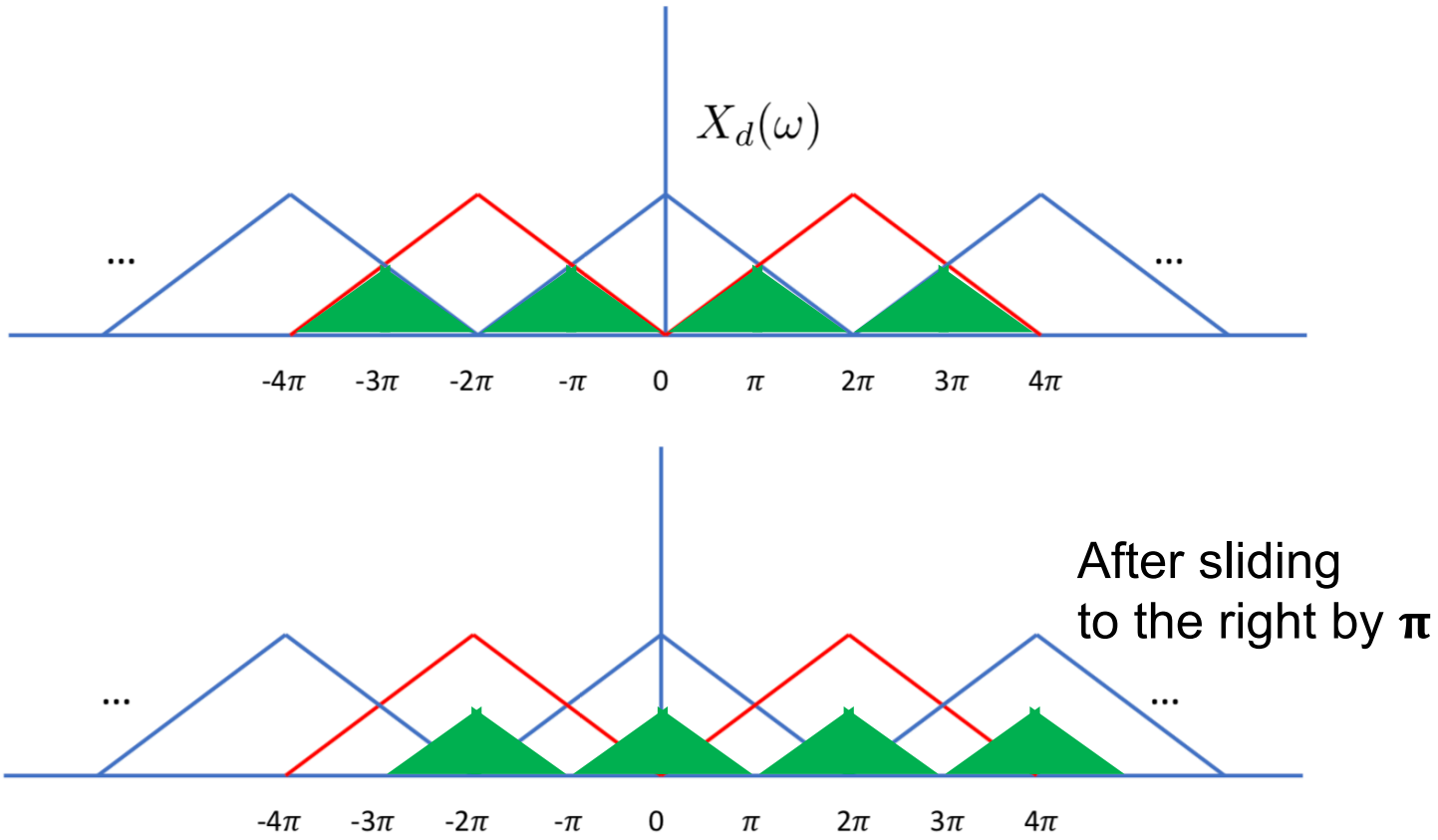


Every other spectral chunk is frequency inverted

- **The inversion effect just described repeats for every other spectral chunk when we downsample by a factor greater than 2**
  - Go back and look at the example for downsampling by 4 again
- **Can we fix the frequency inversion of the baseband signal?**
  - This can be desirable if we plan to further process the sub-sampled signal, and want the frequency components to be in order

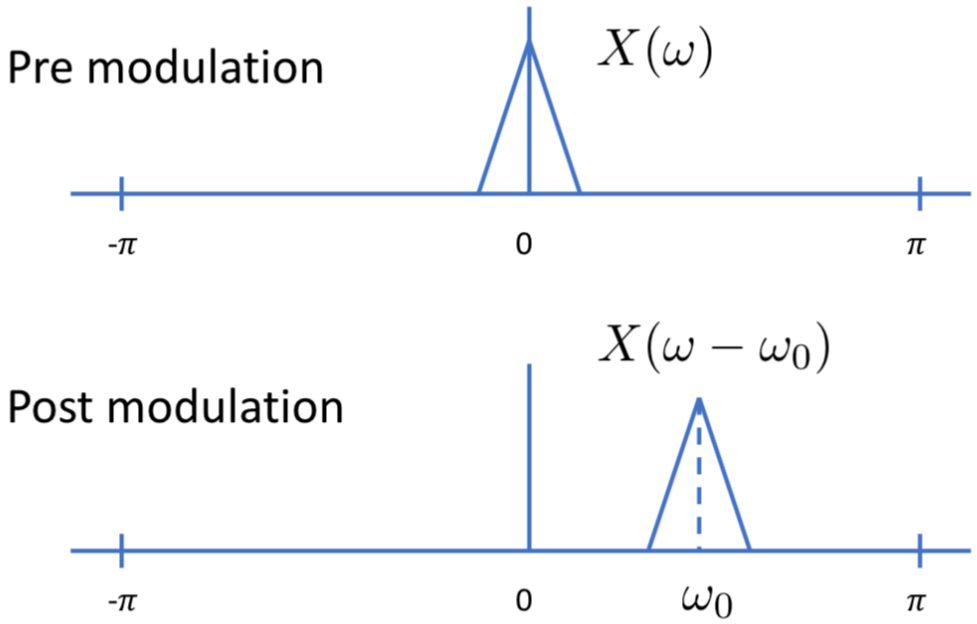
Fix the inversion by sliding the spectra by π

- **Remember, the DTFT is periodic with period 2π**
  - So sliding the entire spectrum to the right by π will "un-invert" the spectral components



The DTFT modulation property

- **If we multiply, i.e. modulate, *x(n)* by a complex exponential, its spectrum changes as follows**



Let

$$y(n) = x(n)e^{i\omega_0 n}$$

Then

$$Y(\omega) = X(\omega - \omega_0)$$

Fixing inversion by modulation

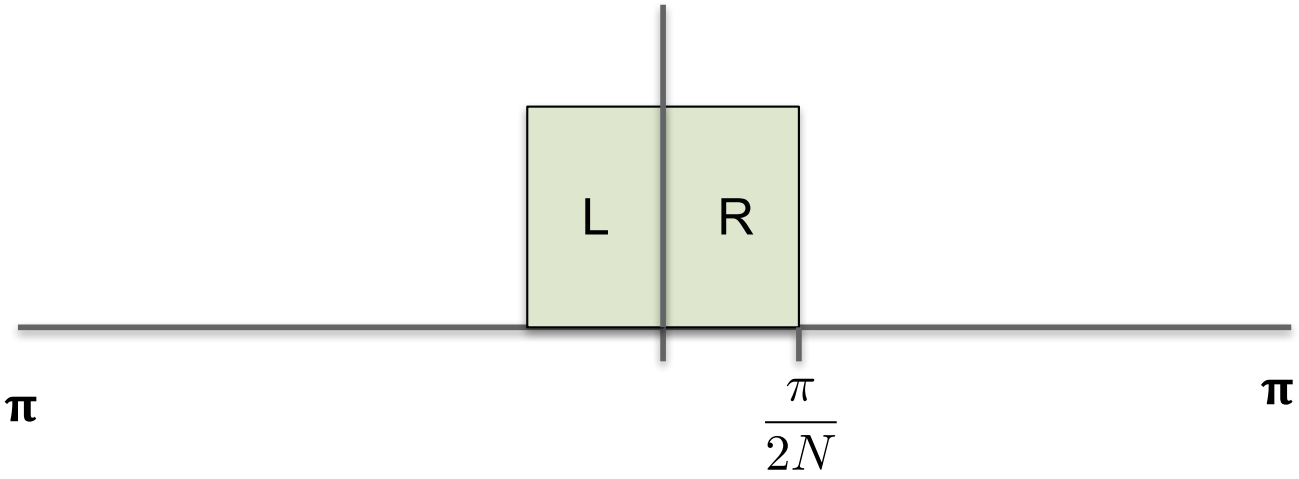
- **We can therefore fix the frequency inversion by modulating every other sub-band with the factor**

$$e^{i\pi n}$$

Modulating with {1, -1, 1, -1, ...} slides the spectrum to the right by π

Building BPFs from a prototype LPF

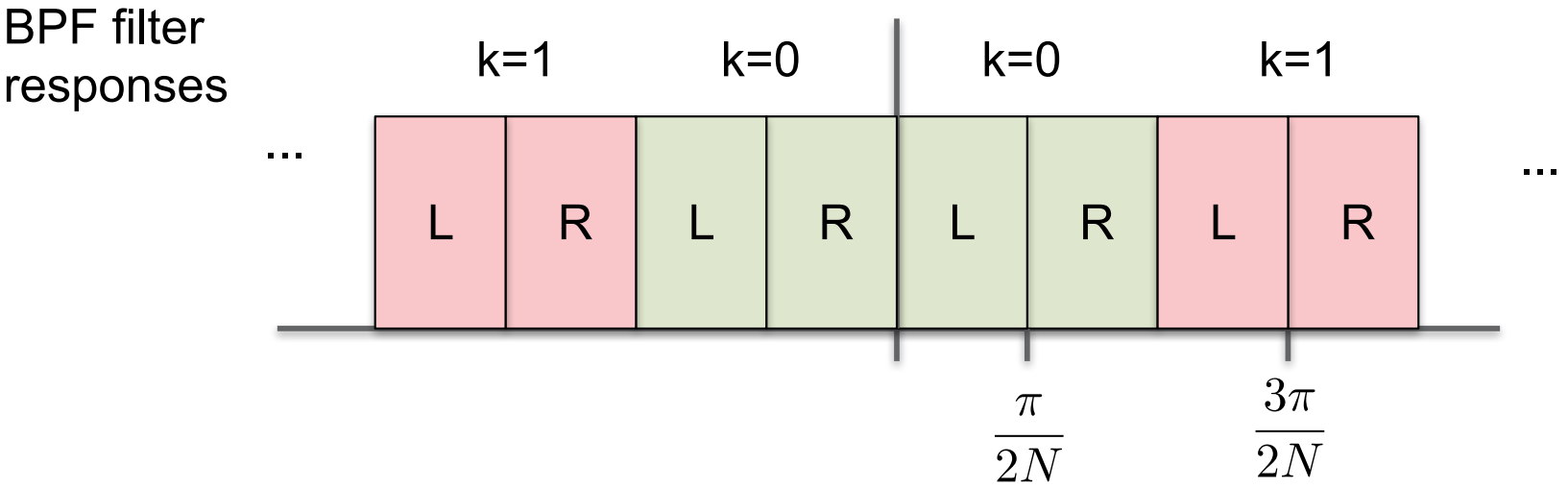
- **Assume we have an LPF with the ideal frequency response shown below and impulse response *h(n)***
  - *N* is the number of filter banks



Modulation by a cosine

- **If we multiply *h(n)* by a cosine, we move its spectrum both up and down**
  - This gives us the impulse response of a bandpass filter

$$h_k(n) = h(n)\cos\left(\frac{(2k+1)m\pi}{2N}\right) \quad k = 0, 1, \ldots, N-1$$



Synthesis uses up-sampling

- **Up-sampling by N means inserting N-1 zeros between samples.**
  - Below is up-sampling by 3. Note that the resulting data rate has been increased by a factor of 3

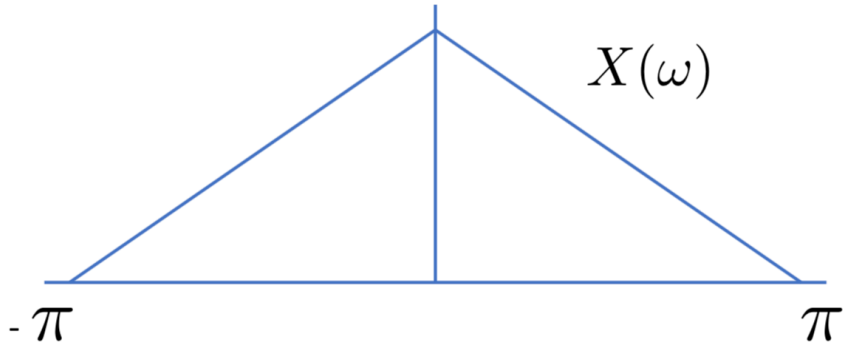
$$\{x_0, 0, 0, x_1, 0, 0, x_2, 0, 0, \ldots\}$$

The spectrum after up-sampling is

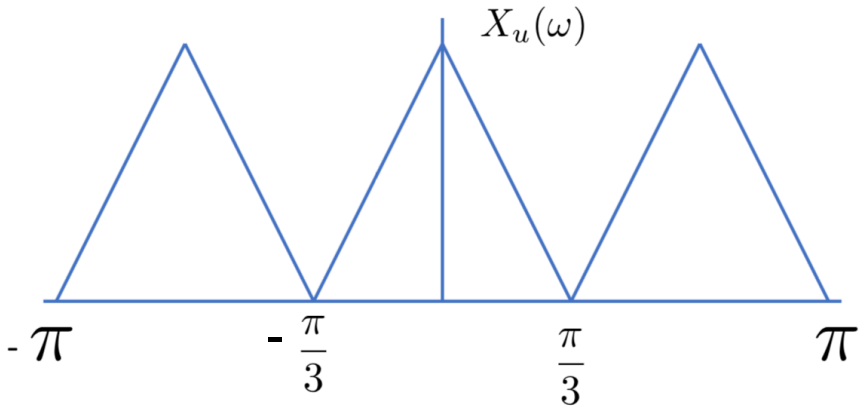
$$X(N\omega)$$

Upsampling "replicates" the spectra

- **Spectrum before up-sampling**



- **Spectrum after up-sampling by 3**



BPF up-sampled signal

- **After up-sampling, the signal must be bandpass filtered to leave only the desired spectral piece in place**

Python hints

- **Input the arrays C and D at the beginning of your program**
- **After reading in an audio track, I found it useful to create a 2-D buffer of time domain samples as follows**
  - x = np.reshape(y, (-1, 32))
    - ✓ the columns of x hold the “packets” of 32 audio samples we will shift into the large X vector described in the lab writeup.
    - ✓ can pass this matrix into your pqmf( ) function

Matlab hints

- **I found it useful to create an array of subband samples**
  - ss = np.zeros_like(x) Each row represents one set of subband samples
  - pqmf( ) can return this array to the calling function for graphing, etc.
- **Pay attention to how you shift data into buffers!**
  - A pure vector copy doesn't put things into the buffer in the right order
    - ✓I found the numpy command np.flip( ) useful