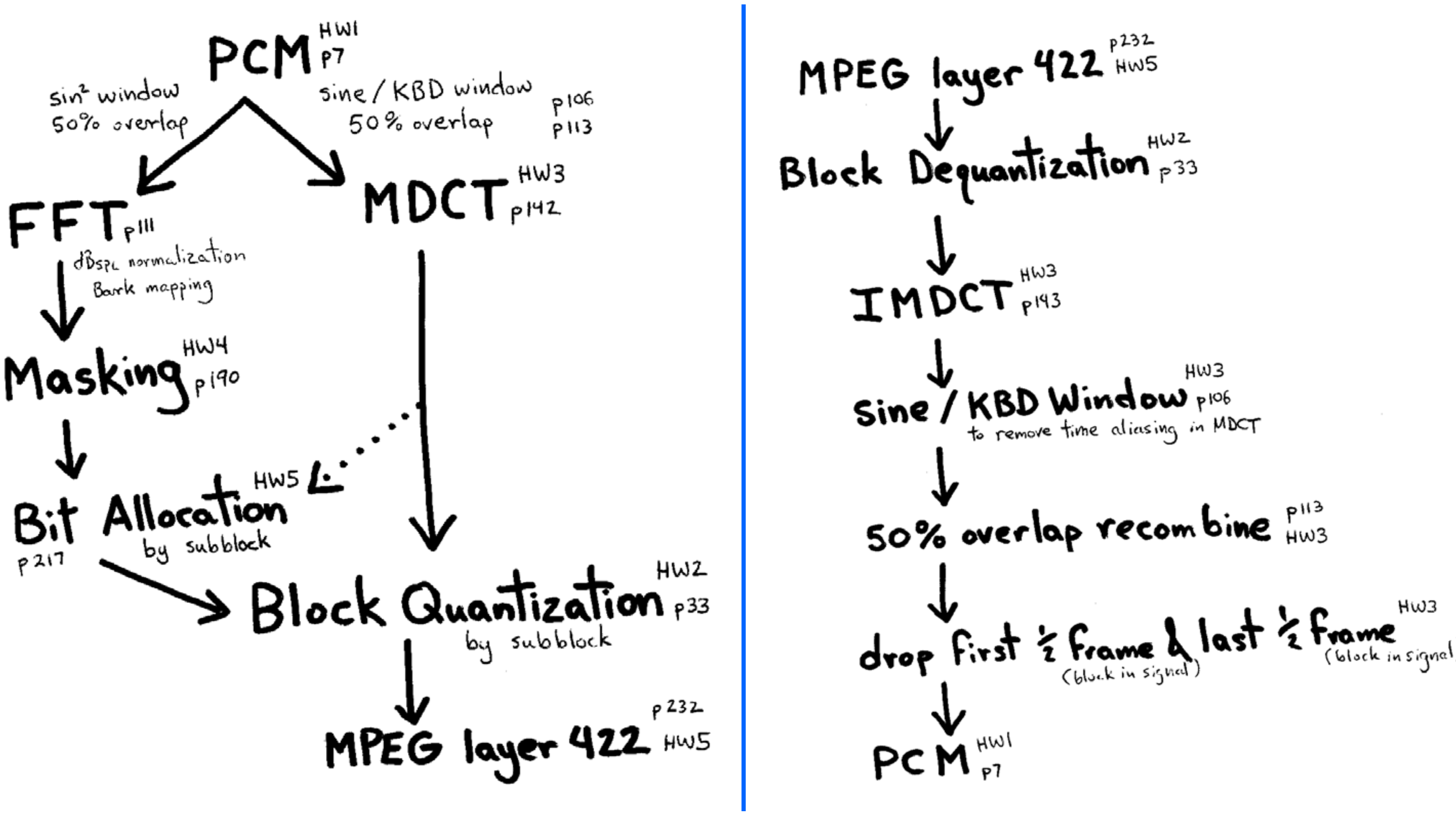
Music 422
Perceptual Audio Coding
Review Lecture

Craig Stuart Sapp

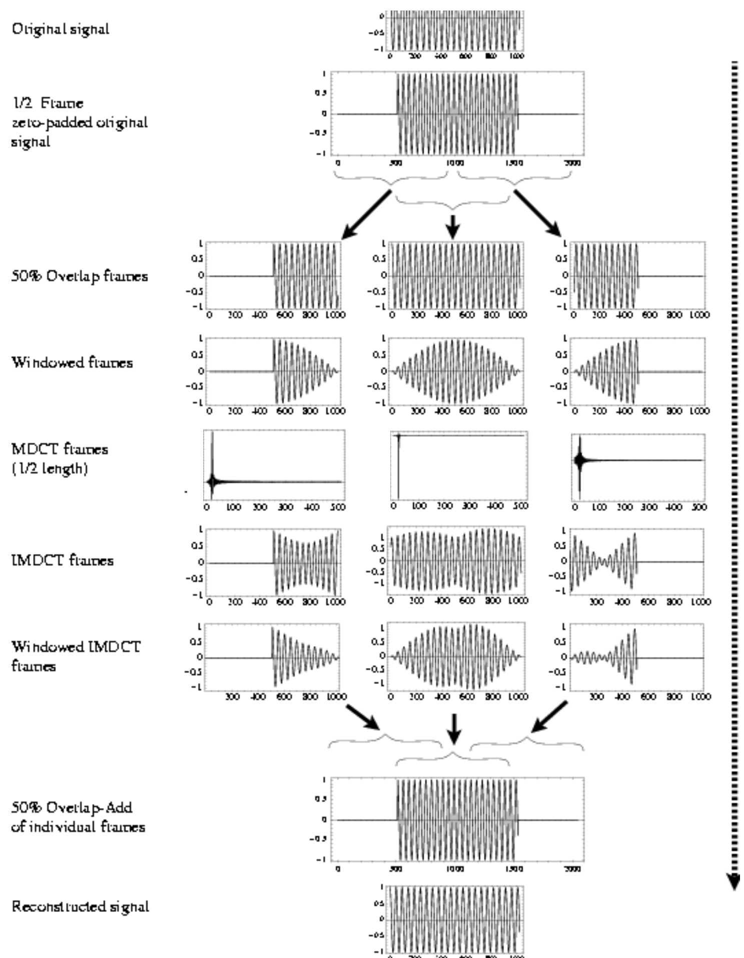
19 February 2010

Stanford University

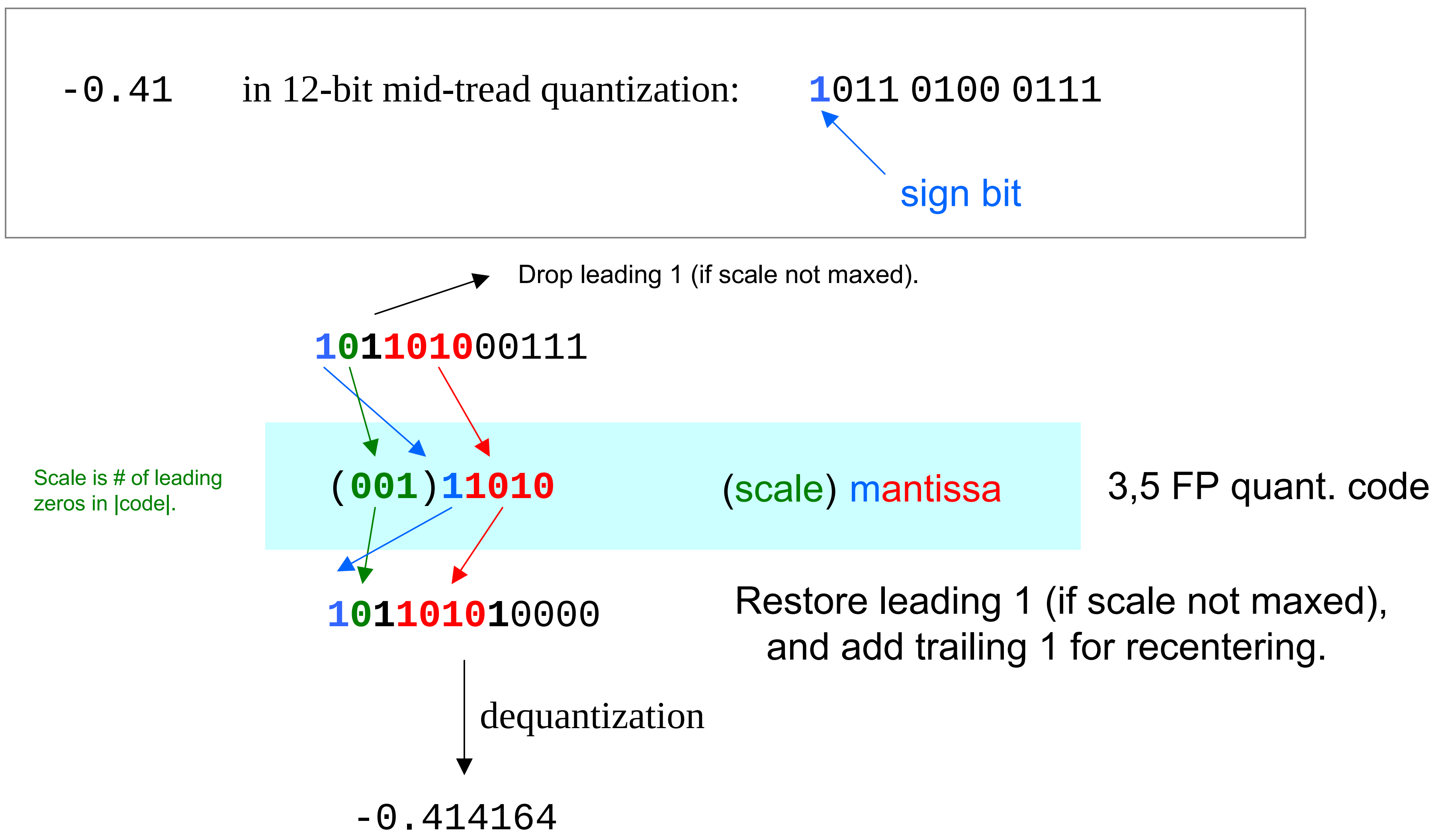
Encoding/Decoding Architecture



Time Domain Alias Cancellation



Floating Point Quantization



Block Floating-Point Quantization

- Share a single scale value with multiple mantissas.
- Scale is derived from maximum absolute value in block.
- Must store leading 1s (so 6 dB noisier than plain Floating-Point).

-1.00	1111 1111 1111	(000) 11111	1111 1100 0000
-0.97	1111 1100 0010	11111	1111 1100 0000
-0.41	1011 0100 0111	10110	1011 0100 0000
0.00	0000 0000 0000	00000	0000 0000 0000
0.008	0000 0001 0000	00000	0000 0000 0000
0.51	0100 0001 0100	01000	0100 0100 0000
0.82	0110 1000 1111	01101	0110 1100 0000
0.999	0111 1111 1101	01111	0111 1100 0000
1.0	0111 1111 1111	01111	0111 1100 0000

scale shared for all mantissas

If |mantissa| is zero, then don't add trailing 1!

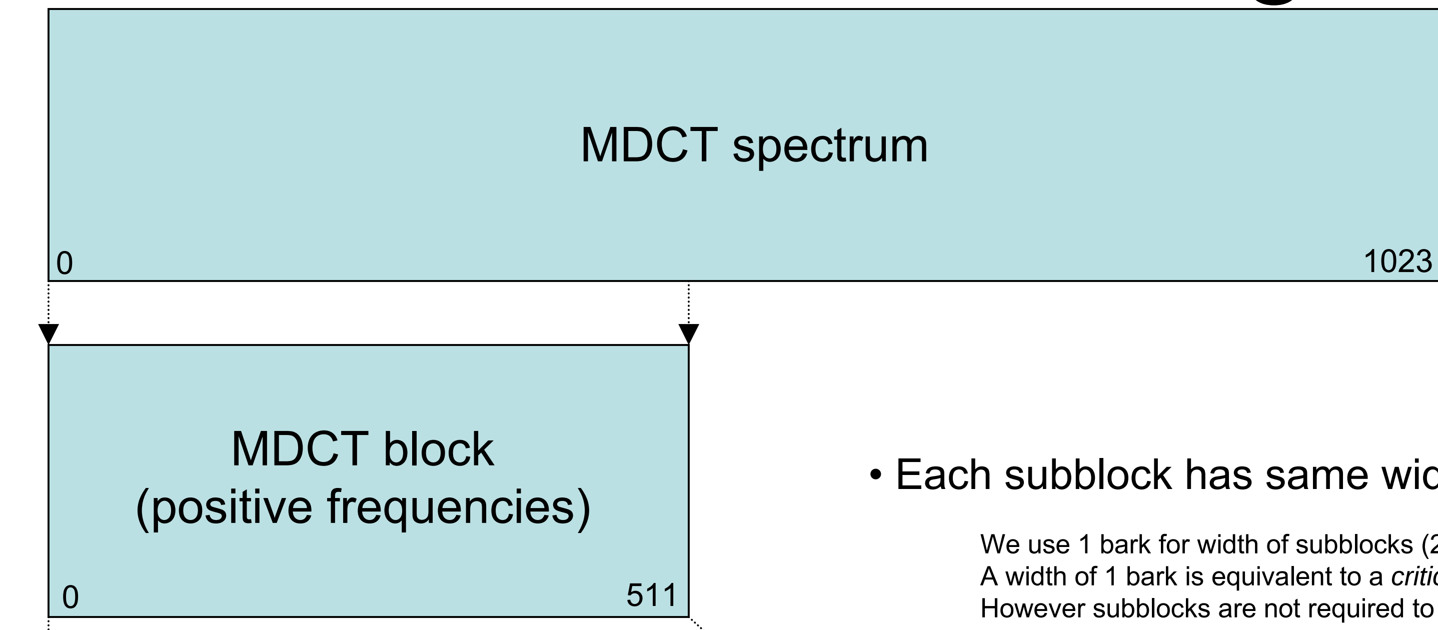
Don't remove any leading 1s

Max value's scale is 0

Don't add leading ones (because they were not taken out)

- Example BFP uses 48 bits (3*1+5*9) compared to FP at 72 bits (3*9+5*9).
- Block floating-point used in quantizing sub-block spectral amplitudes (HW5).

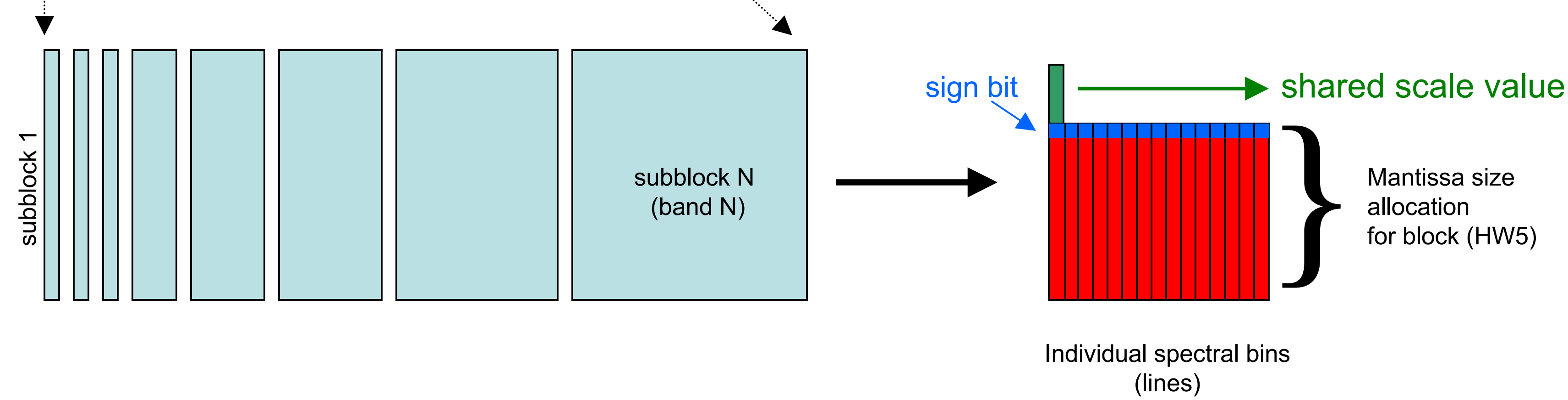
Subblock Block Floating Point



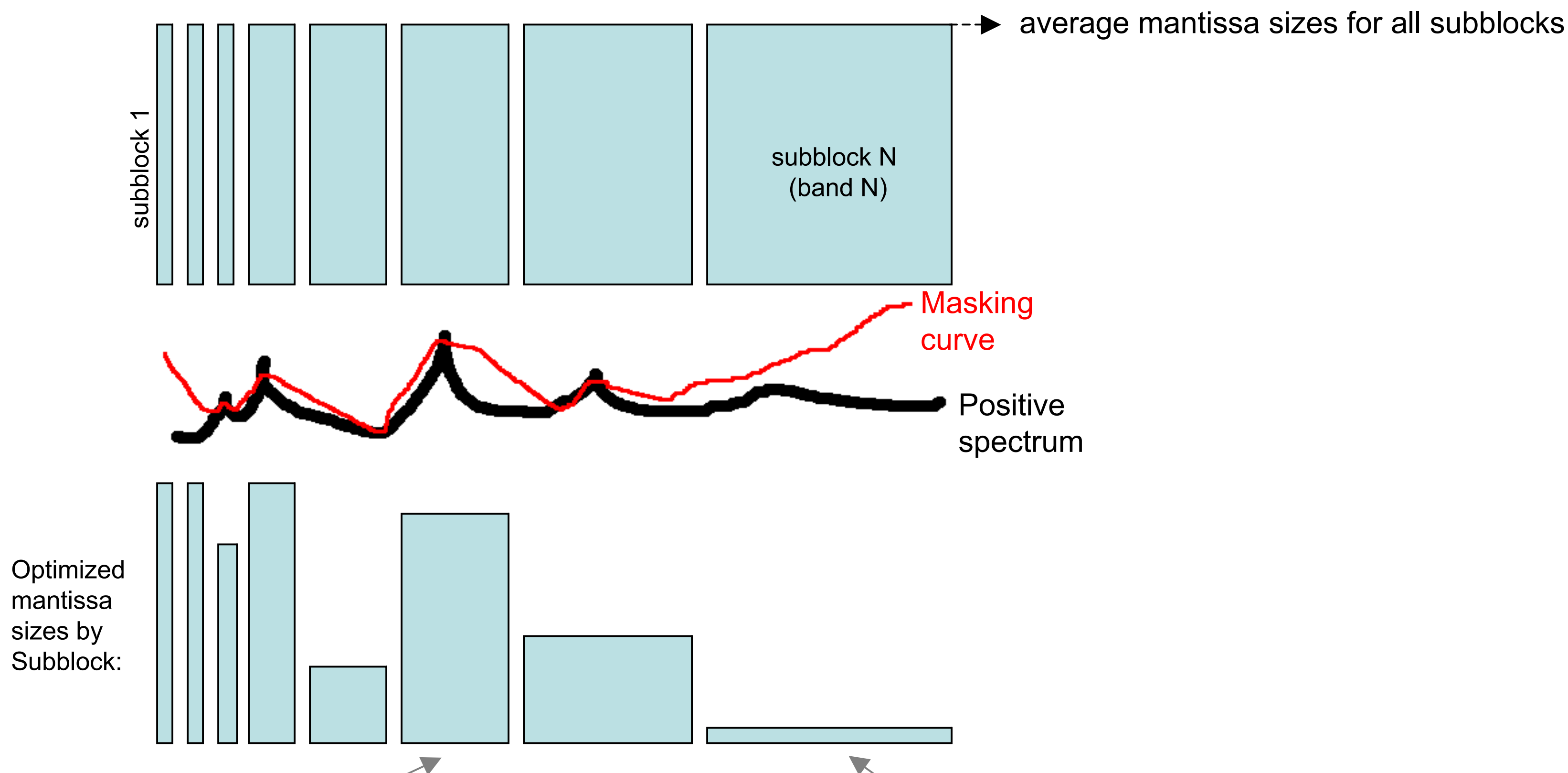
- Each subblock has same width in barks

  We use 1 bark for width of subblocks (25 subblocks).
  A width of 1 bark is equivalent to a *critical band*.
  However subblocks are not required to be 1 critical band.
  For example 30 subblocks would each be 0.83 barks.

  - **Project idea: try different subblock counts (Bark and/or ERB based)**



Bit-Allocation Optimization



Can't quantize too much because signal is audible in this subblock

Can quantize heavily since not very audible

Bit-Allocation Optimization (2)

Bit pool: number of bits available to all mantissas.

Signal-to-mask ratio for subblock *b*.

# of spectral bins in subblock *c*.

$$R_b^{opt} = \frac{P}{K_p} + \frac{\log_2(10)}{20}\left[\mathrm{SMR}_b - \frac{1}{K_p}\sum_{R_c \neq 0} K_c\ \mathrm{SMR}_c\right]$$

Optimized mantissa bit allocation for each subblock *b*.

Number of non-zero spectral bins in block.

Summation over all subblocks *c* which are not pre-assigned an allocation of 0 mantissa bits due to a poor SMR.

A constant

Without pre-screening by subblock SMR:

$$R_b^{\mathrm{opt}} = R_{\mathrm{avg}} + \frac{\log_2(10)}{20}\left[\mathrm{SMR}_b - \frac{1}{K}\sum_{c} K_c\ \mathrm{SMR}_c\right]$$

Average bits per mantissa in block

Total bins in block.

Signal-To-Mask Ratio

$$\mathrm{SMR}_b = 20\left(\log_{10}\left|x_{\mathrm{max}_b}\right| - log_{10}\ M_b\right)$$

Maximum spectral bin amplitude in subblock

Masking amplitude for subblock

- **Project idea: compare FFT/MDCT as x_max_b**

Equivalent:

$$\mathrm{SMR}_b = \mathrm{dB}_{(\mathrm{SPL})}\,\text{of max amplitude in subblock} - \mathrm{dB}_{(\mathrm{SPL})}\,\text{of mask amplitude for subblock}$$

Alternative:

$$\begin{aligned}\mathrm{SMR}_b &= \max_n\left[\mathrm{dB}_{(\mathrm{SPL})}\,\text{of amplitude}_n\ \text{in subblock} - \mathrm{dB}_{(\mathrm{SPL})}\,\text{of mask amplitude}_n\ \text{in subblock}\right] \\ &= \max_n\left[SMR_n\right]\end{aligned}$$

Mantissa Bit-Allocation (3)

SMR's Expressed in amplitues:

$$R_b = R_{\text{avg}} + \log_2\left(\frac{x_{\max_b}}{M_b}\right) - \frac{1}{K}\sum_c K_c \,\log_2\left(\frac{x_{\max_c}}{M_c}\right)$$

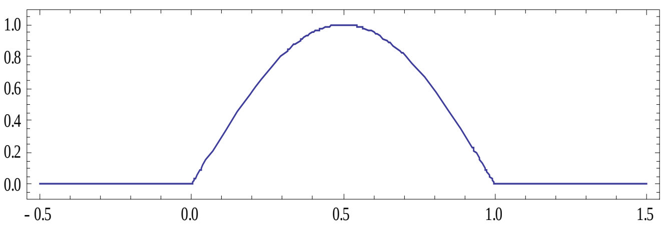
Bit-allocation optimization, ignoring masking:

$$R_b = R_{\text{avg}} + \log_2 x_{\max_b} - \frac{1}{K}\sum_c K_c \,\log_2 x_{\max_c}$$

Even/Odd Window Definition

Continuous normalized window (causal definition):

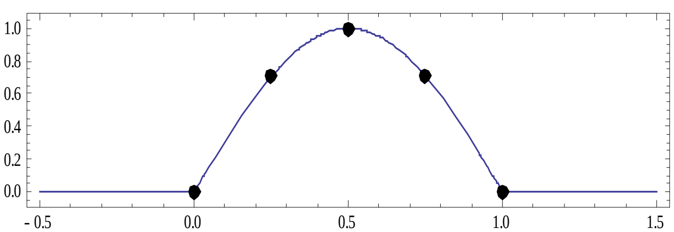
sin(pi * t)   for t>=0 and t <=1



Odd-length sampled window:

N=5
sin(arange(N)*pi/(N-1))

- N-1 for stretching sample points to center of window (otherwise not symmetric: {0, .2, .4, .6., .8}).

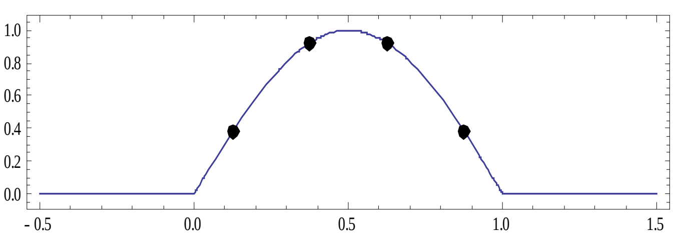


$$\left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\} \longrightarrow \left\{0, \frac{1}{\sqrt{2}}, 1, \frac{1}{\sqrt{2}}, 0\right\}$$

Even-length sampled window:

N=4
sin((arange(N)+0.5)*pi/(N))
(c.f. p. 107)

- 0.5 offset for sampling from center of window, not left justified in window range (otherwise {0, ¼, ½, ¾}).
- not sin(arange(N)*pi/(N-1)) since COLA of window^2 @50% overlap would not occur (otherwise range would be {0, 1/3, 2/3, 1}).



$$\left\{\frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}\right\} \longrightarrow \{0.382683, 0.92388, 0.92388, 0.382683\}$$

***c.f.* Parallels to DFT and MDCT**

Even & Odd Window Lengths

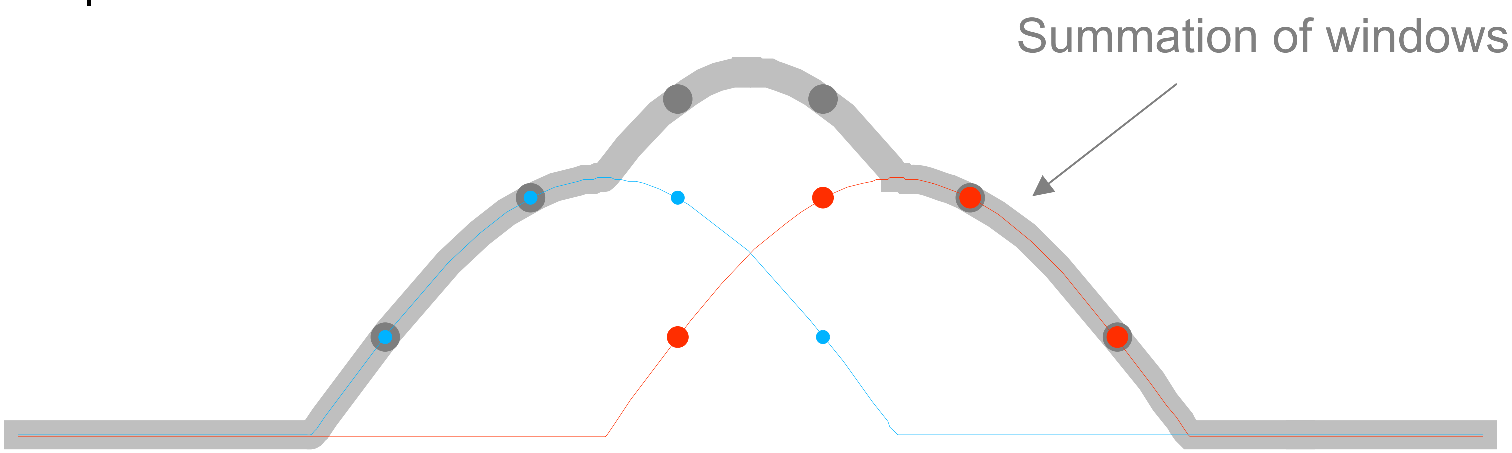
Optimized for 50% (25%, 12.5%…) overlap add systems

```
import numpy as np

def SineWindow(length):
  N = int(length)
  assert N > 1
  if (N%2 == 0):     # even sampling of continuous window definition
    halfwindow = np.sin( (np.arange(N/2)+0.5) * np.pi /  N)
    return np.concatenate([halfwindow, np.flipud(halfwindow)])
  else:                   # odd sampling of continuous window definition
    halfwindow = np.sin( np.arange(N/2+1) * np.pi / (N-1))
    # don't repeat the middle sample of an odd window:
    return np.concatenate([halfwindow, np.flipud(halfwindow[0:-1])])

def HanningWindow(length, power=2):
  return SineWindow(length)**power
```

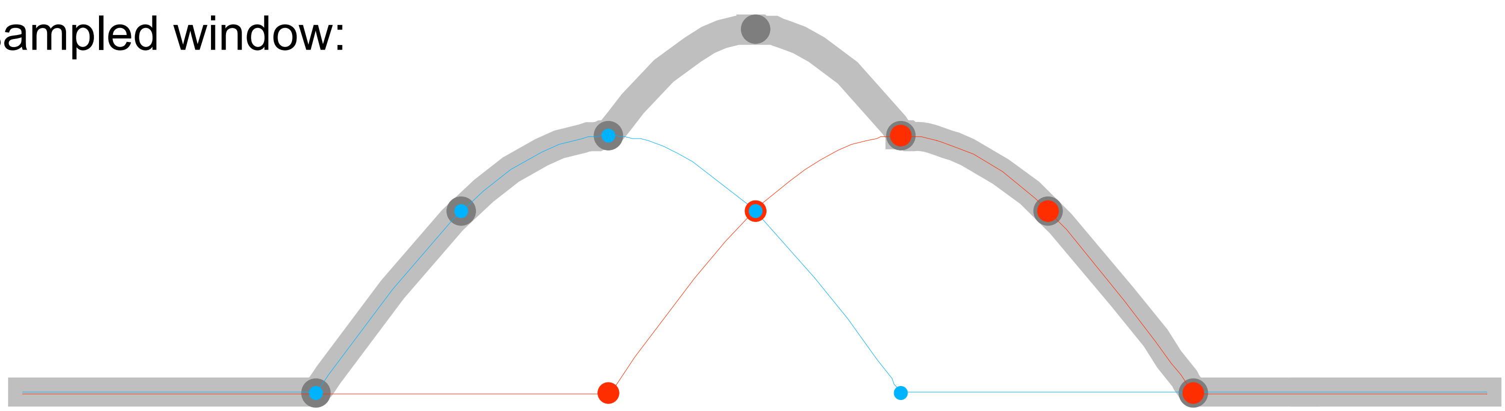
50% Overlap Add

(Sine Window)

Evenly sampled window:



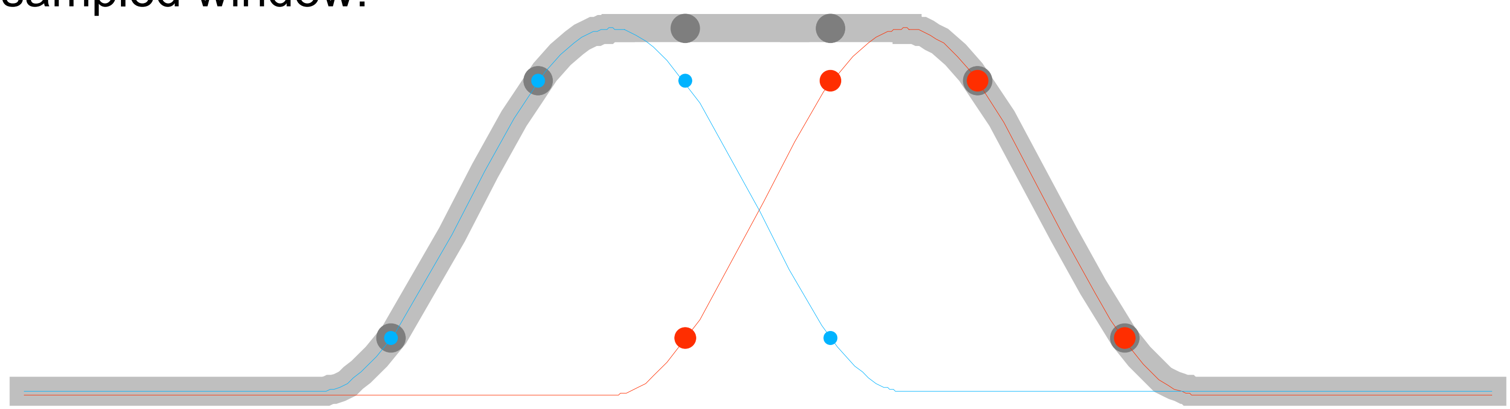
Oddly sampled window:



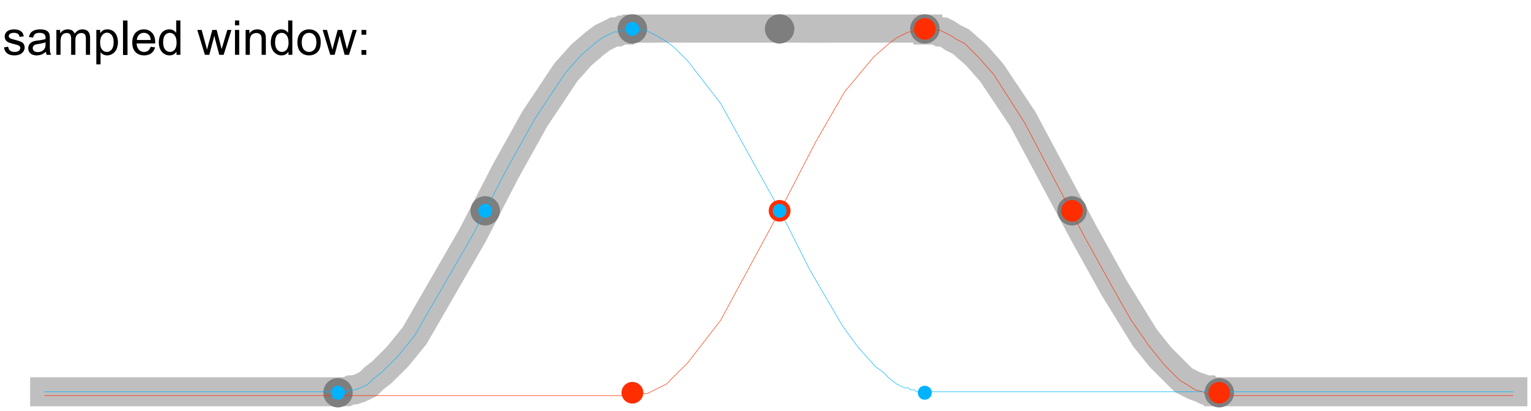
Constant OverLap Add (COLA)

(Hanning Window = Sine^2 Window)

Evenly sampled window:

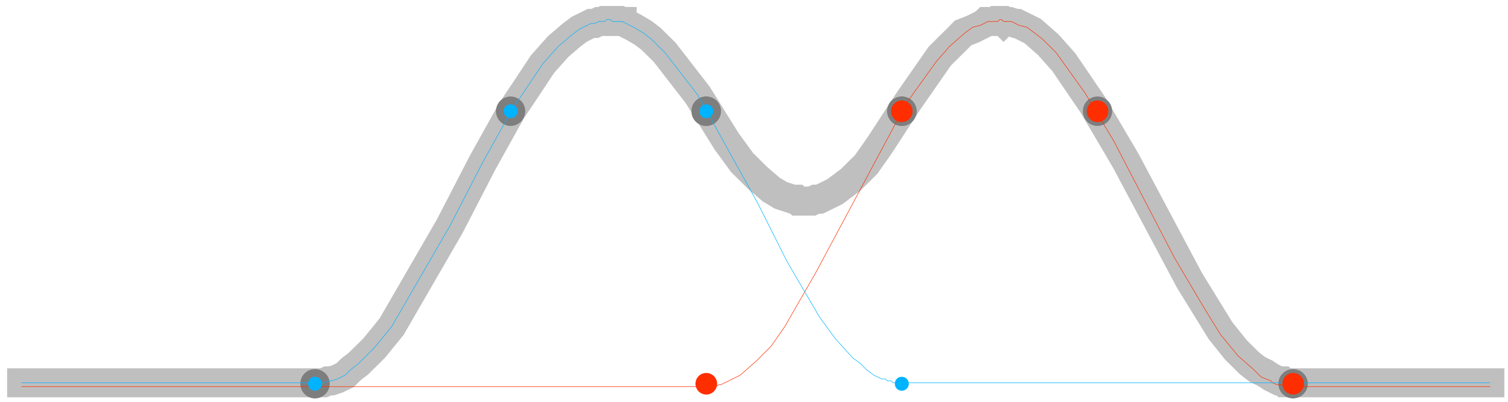


Oddly sampled window:



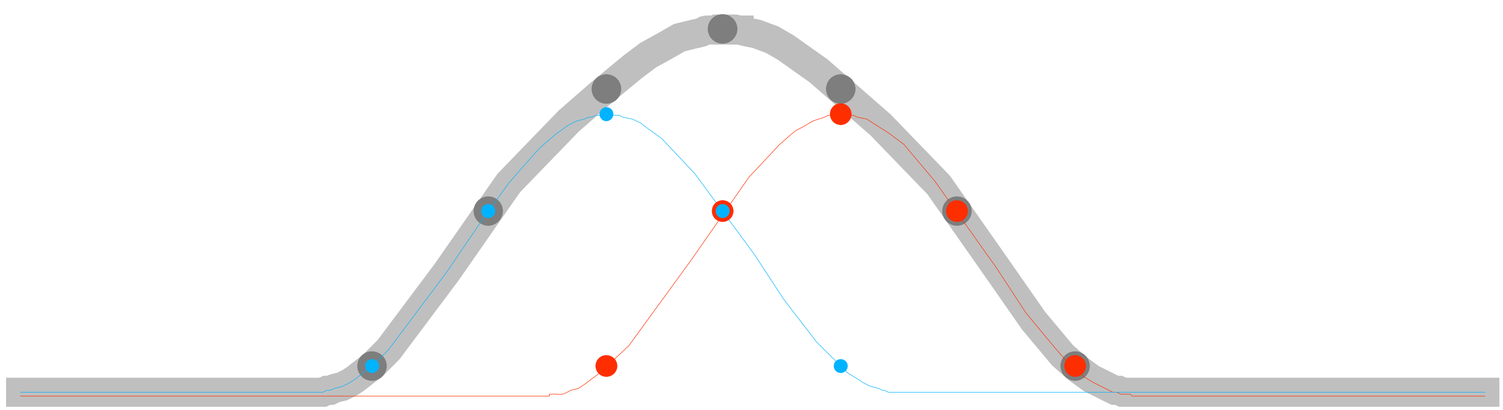
Bubbly COLA

Using even length with odd window sampling definition (less than 50% overlap):

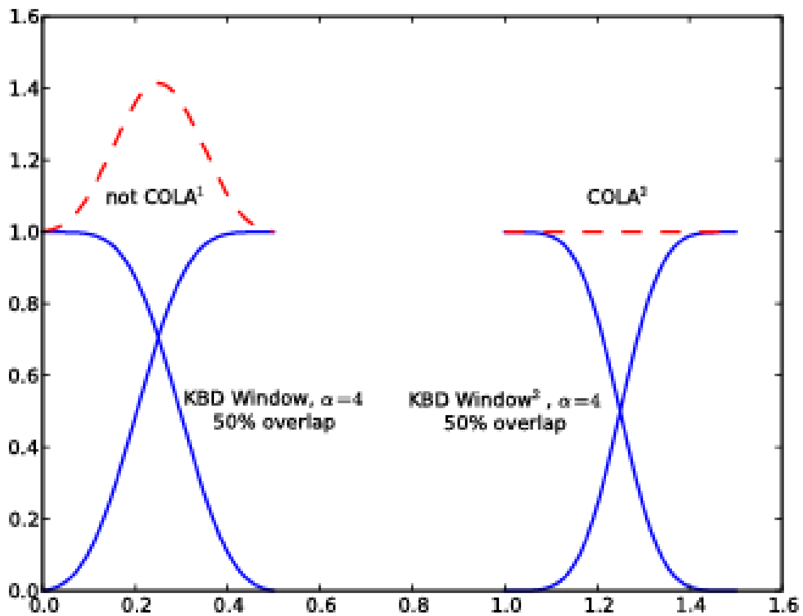


N.B.: numpy.hanning(4) → [0, 0.75, 0.75, 0]

Using odd length with even window sampling definition (more than 50% overlap):



Kaiser-Bessel Derived Window




```
import pylab as pl
N = 1024
window = KBDWindow(1024, 4.0)
winl = window[0:N/2];   winr = window[N/2:N]
overlapsum = winl + winr
win2l = winl**2;   win2r = winr**2
overlap2sum = win2l + win2r
n = np.arange(N)/(N-1.0) # normalized range
nl = n[0:N/2];   nr = n[N/2:N]
pl.plot(nl,   winl,       'b-');   pl.plot(nl,   winr,       'b-')
pl.plot(nl,   overlapsum,  'r--')
pl.plot(nl+1, win2l,       'b-');  pl.plot(nl+1, win2r,       'b-')
pl.plot(nl+1, overlap2sum, 'r--')
pl.text(0.25, 1.1, 'not COLA$^1$', \
     horizontalalignment='center', verticalalignment='center')
pl.text(1.25, 1.1, 'COLA$^2$', \
     horizontalalignment='center', verticalalignment='center')
pl.text(0.5, 0.5, 'KBD Window, $\\alpha=4$\n50% overlap', \
     horizontalalignment='center', verticalalignment='center')
pl.text(1.0, 0.5, 'KBD Window$^2$, $\\alpha=4$\n50% overlap', \
     horizontalalignment='center', verticalalignment='center')
pl.show()
```

KBD Window in Python

```
import numpy as np

def KBDWindowKVH(length, beta):
  """Kaiser Bessel Derived Window
    Arguments:
      length: Length of output window array.  Must be positive & even.
      beta:   Kaiser window parameter alpha times pi.
  """
  N      = int(length)
  assert N%2 == 0
  assert N > 0
  M      = N/2
  w      = np.kaiser(M+1, beta)
  wMsum  = 1.0 / np.sqrt(w[0:M+1].sum())
  wnsuml  = np.sqrt(np.cumsum(w[0:M]))
  wl         = wnsuml * wMsum
  return np.concatenate([wl, np.flipud(wl)])
```

Note: $\beta = \pi \alpha$

c.f. sinc()

```
import numpy as np
import scipy.special as sps

def KBDWindowCSS(length, alpha):
  """Kaiser Bessel Derived Window
    Arguments:
      length: Length of output window array.  Must be positive & even.
      alpha:  Kaiser window parameter.
  """
  N         = int(length)
  assert N % 2 == 0
  assert N > 0
  halfN       = N/2
  beta         = np.pi * alpha
  bessels    = sps.i0(beta * np.sqrt(1.0-(4.0*np.arange(halfN)/N-1.0)**2.0))
  halfwin     = np.cumsum(bessels)
  normalize = halfwin[-1] + sps.i0(beta*np.sqrt(1.0-(4.0*halfN/N-1.0)**2.0))
  halfwin     = np.sqrt(halfwin/normalize)
  return np.concatenate([halfwin, np.flipud(halfwin)])
```

KBD Window in Python (2)

```
import numpy as np

def kaiserWindow(length, alpha): # Colin Raffel's implementation
   N = int(length)
   n = np.arange(N)
   w = (n - (N-1)/2.0)/((N-1)/2.0)
   w = w**2
   w = np.sqrt(1-w)
   w = np.pi * alpha * w
   w = np.i0(w)
   w = w/np.i0(np.pi*alpha)
   return w


def KBDWindowCAR(length, alpha):  # Colin Raffel's implementation
   N      = int(length)
   assert N%2 == 0
   assert N > 0
   wKai   = kaiserWindow(N/2+1, alpha)
   wHalf  = np.sqrt(np.cumsum(wKai[0:-1])/np.sum(wKai))
   return np.concatenate([wHalf, np.flipud(wHalf)])
```

Decibels

10 decibels = 1bel

decibel = 10 $\log_{10}$(**amplitude²** / *reference²*) (reference is an amplitude)
decibel = 20 $\log_{10}$(|**amplitude**|/*reference*) (extracting square from log innards)

- amplitude² is proportional to Energy, Intensity, Power:

decibel = 10 $\log_{10}$(**intensity**/*reference*) (reference is an intensity)
decibel = 10 $\log_{10}$(**energy**/*reference*) (reference is in energy units)
decibel = 10 $\log_{10}$(**power**/*reference*) (reference is in power units)

- amplitude² is proportional to Pressure², RMS² and Voltage²:

decibel = 10 $\log_{10}$(**pressure²**/*reference²*) (reference is in pressure units)
decibel = 20 $\log_{10}$(|**pressure**|/*reference*) (reference is in pressure units)
decibel = 10 $\log_{10}$(**voltage²**/*reference²*) (reference is in voltage units)
decibel = 20 $\log_{10}$(|**voltage**|/*reference*) (reference is in voltage units)
decibel = 10 $\log_{10}$(**rms²**/*reference²*) (reference is in some type of rms amplitude)
decibel = 20 $\log_{10}$(|**rms**|/*reference*) (reference is in some type of rms amplitude)

- units of reference and number being compared have to **match**

Purpose of Decibels

- to relate numbers on a wide range from very small and very large.

Reference: 1 meter (distances are proportional to amplitude)

- Diameter of the Universe: 2.60×10^{26} m → 528 dB
- Diameter of the Milky Way: $9.5\text{x}10^{20}$ m → 419 dB
- Distance to Proxima Centauri: $4.1\text{x}10^{16}$ m → 332 dB
- Distance to the Sun: $1.478\text{x}10^{11}$ m → 223 dB
- Stanford to South Africa: $1.7\ \text{x}10^{7}$ m → 144 dB
- Diameter of Earth: $1.276\text{x}10^{7}$ m → 142 dB
- Blue Whale: 27 m → 29 dB
- Giraffe: 5.5 m → 15 dB
- Human: 2 m → 6 dB
- Mouse: 0.03 m → -30 dB
- Virus: $2\text{x}10^{-5}$ m → -93 dB
- Diameter of hydrogen atom: $1.11\text{x}10^{-11}$m → -199 dB
- Diameter of proton: 10^{-15} m → -300 dB
- Diameter of electron: $<10^{-16}$ m → -320 dB
- Plank length: $1.616\text{x}10^{-35}$ m → -695 dB

- Universe has a dynamic range of 1223 dB (203 bits)

Sound Pressure Level in Decibels

$$Db_{SPL} = 20 \log_{10} (\text{pascals}/0.000020)$$

- *Reference pressure*: 20 μPa = 20×10^{-6} kg/(m-s^2)
- 1 pound per square inch = 6,894.76 pascals
  20 μPa = 2.9×10^{-9} p.s.i.
  1 atmosphere = 101,325 pascals = 194 dB_{SPL}
- My bike tires are inflated to 150 p.s.i.:
  20 log(150 / (2.9 X 10-9)) = 214 dB_{SPL}

- Noise from blood flow in ears is about 0 dB_{SPL}
- Noise from air molecules randomly hitting eardrum is very roughly –6 dB_{SPL}

http://www.silcom.com/~aludwig/Physics/Noise_floor/Molecular_noise_and_blackbody.htm

Other Decibels

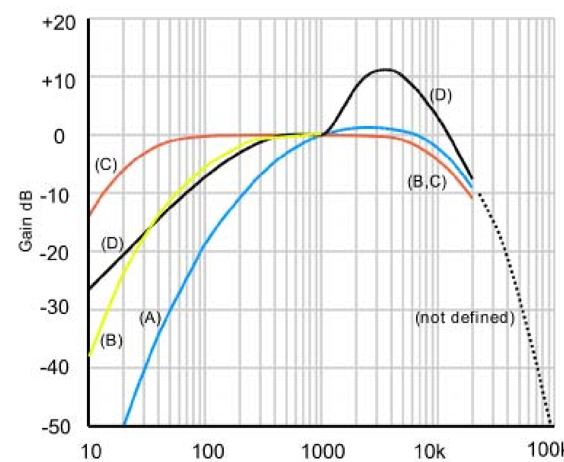
- Decibel **differences** are the same regardless of the reference.
  - — For example, 6 dB difference when doubling amplitude for any reference:
    $20 \log(2/r) - 20 \log(1/r) = 20 \log[(2/r)/(1/r)] = 20 \log(2) = 6.0206$ dB difference
- But ***don't*** subtract decibels that use different references.

- Other references:

dBm: 1 mW (power)
dBW: 1 W (power)
dBV: 1 V_{RMS}
dBu: 0.775 V_{RMS}
dB_{SIL}: 10^{-12} W/m^2
dBfs: max

Loudness
(perceptual weighting):

dB(A)
dB(B)
dB(C)
dB(D)
dB(Z)



A-weighting (blue), B (yellow), C (red), and D-weighting (blk)

en.wikipedia.org/wiki/A-weighting

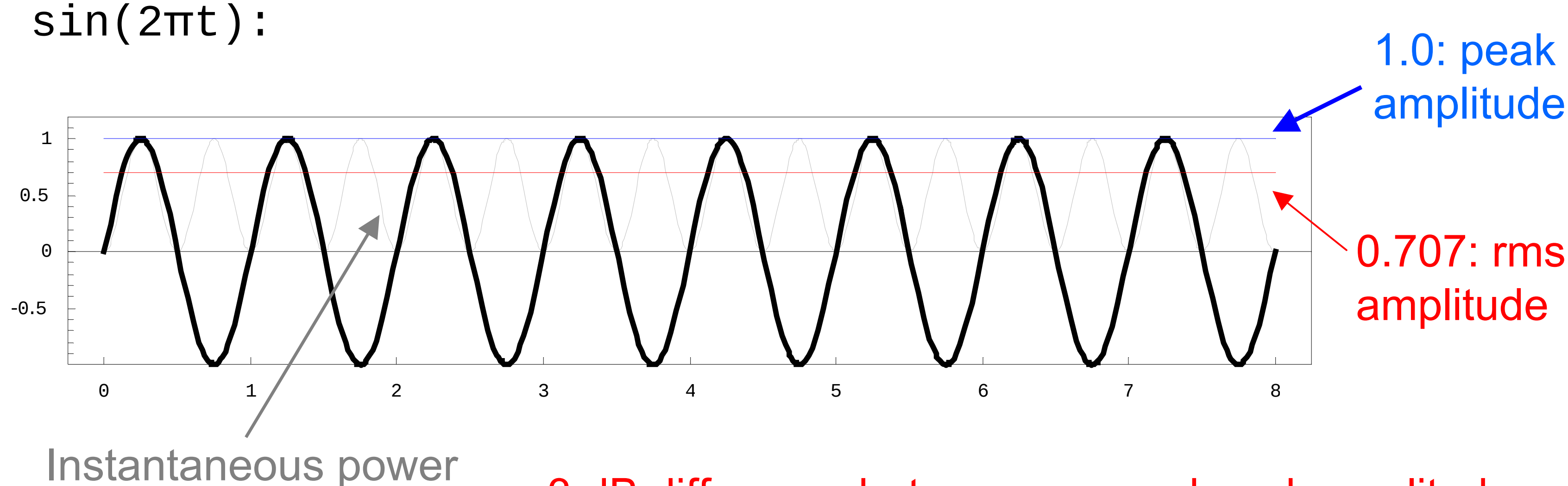
RMS

- ***Root Mean Squared*** usually the measured pressure value of a physical sound:

$$x_{\mathrm{rms}} = \sqrt{< x^2 >} = \sqrt{\frac{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}{n}}$$

- Square root of the average power in a waveform (units in amplitude or pressure).

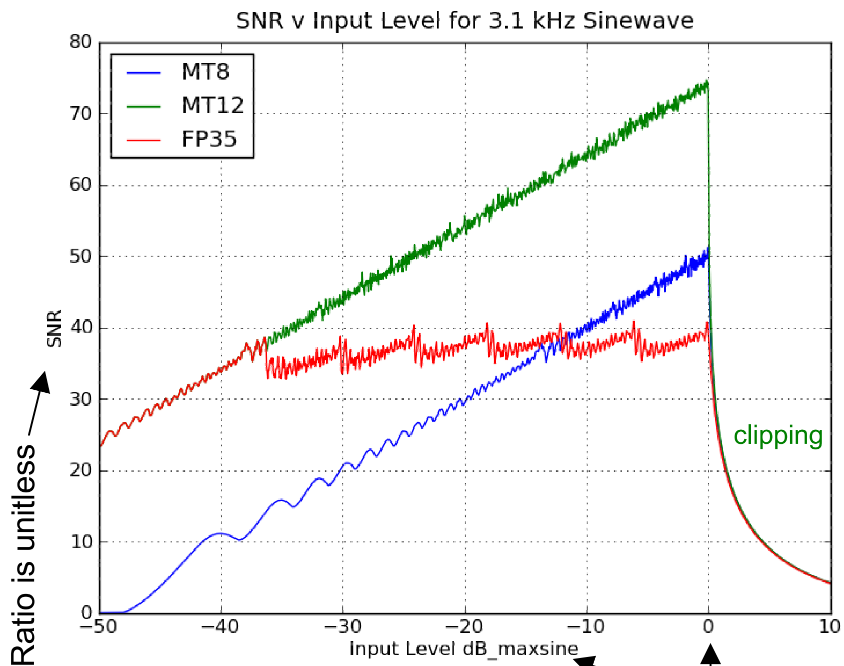
sin(2πt):

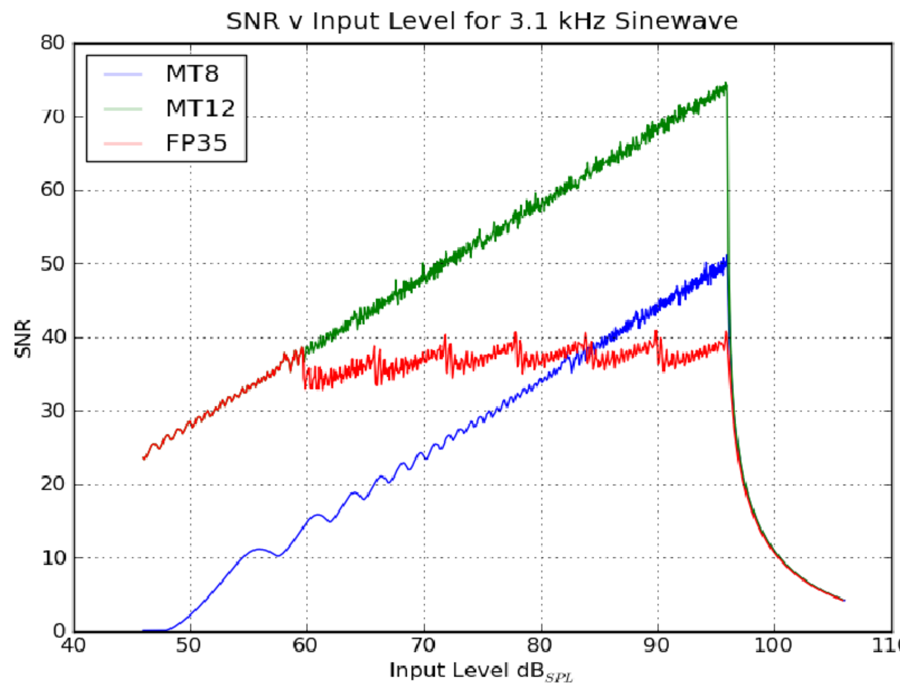


- 3 dB difference between rms and peak amplitudes

dB always in relation to a reference

- Good practice to identify reference as a subscript after “dB”.





$20 \text{ Log}_{10}(\text{peak}/1.0)$

$20 \text{ Log}_{10}(\text{peak}/0.0000158489)$

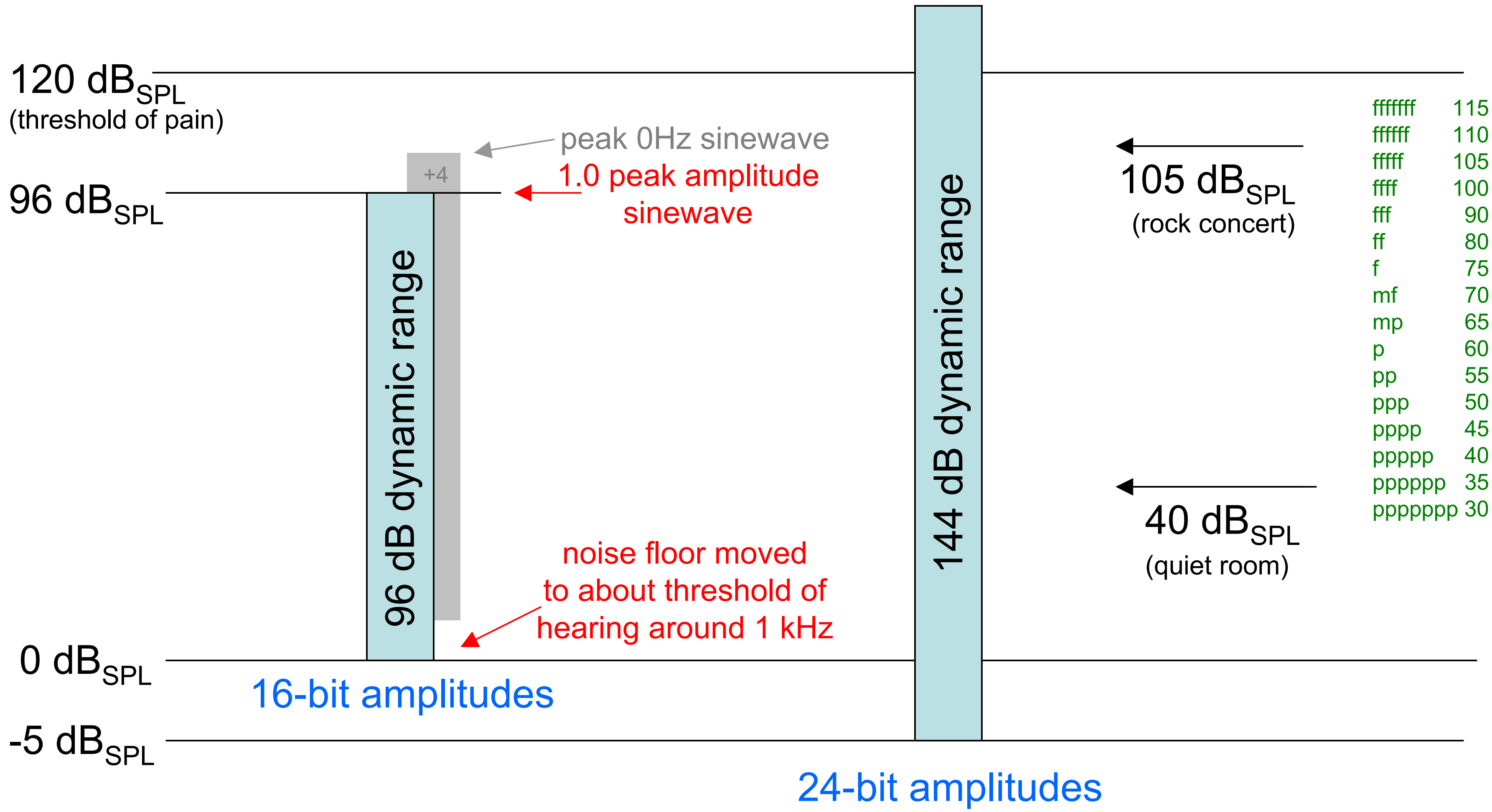
0 dB means reference

$20 \log(\text{ref}/\text{ref}) = 20 \log(1) = 0$

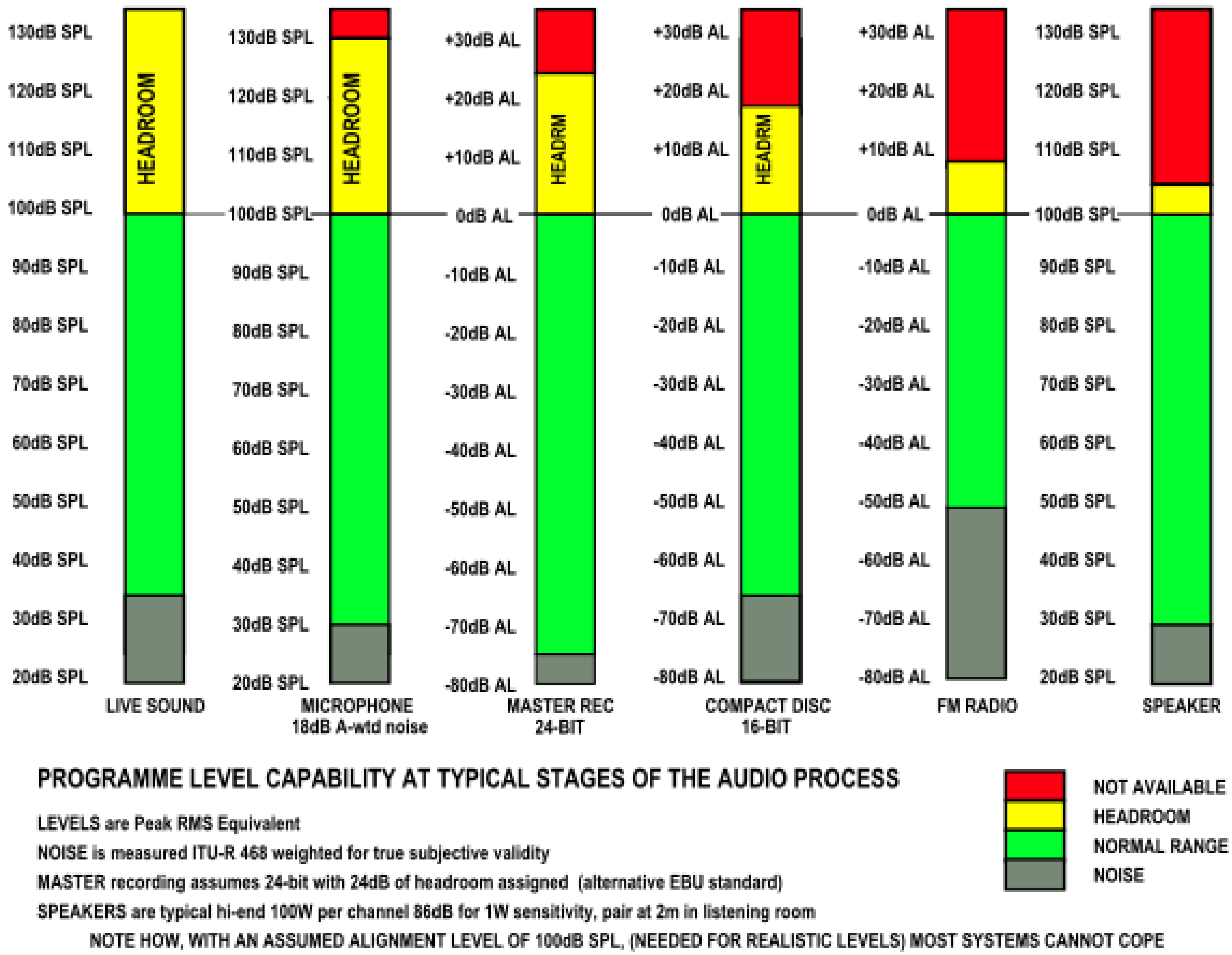
Reference when setting peak amplitude of 1.0 to 96 dB_{SPL}.

$96 = 20 \log(1/r)$ :: $r = 10$^(-96/20)

16-bit vs 24-bit dynamic range

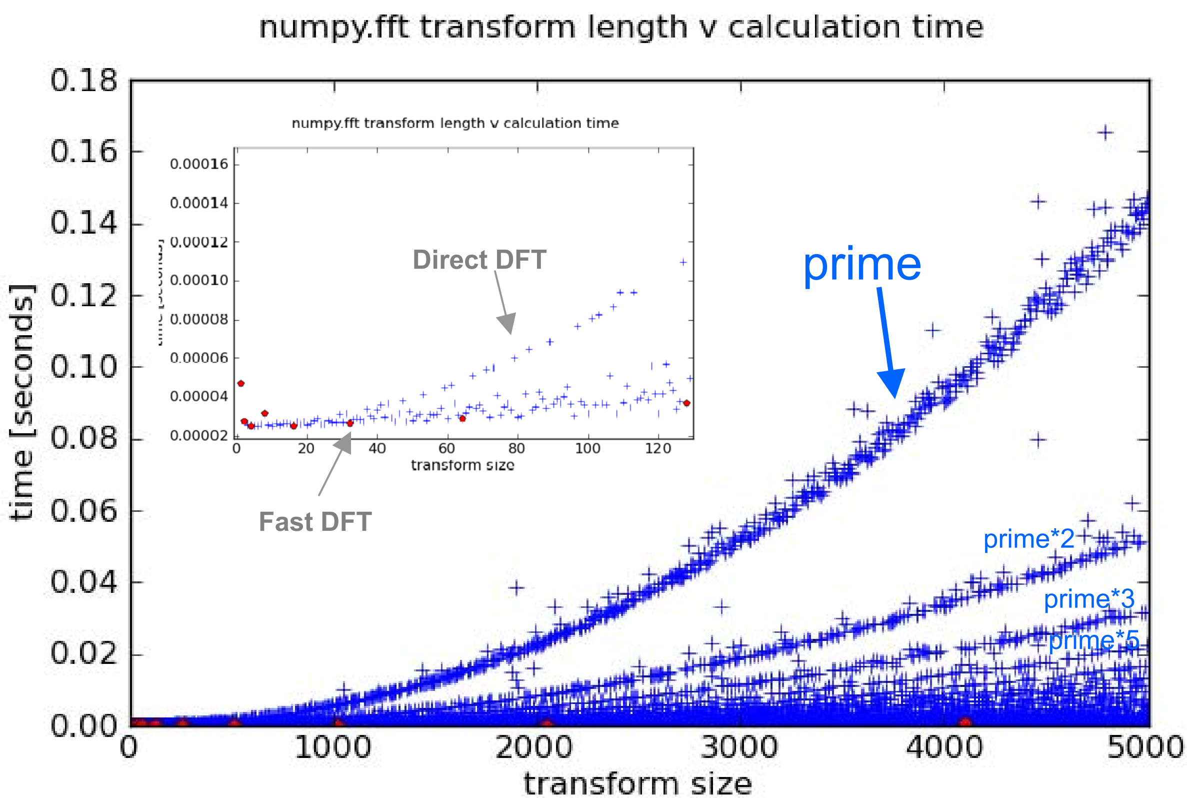


Dynamic Ranges



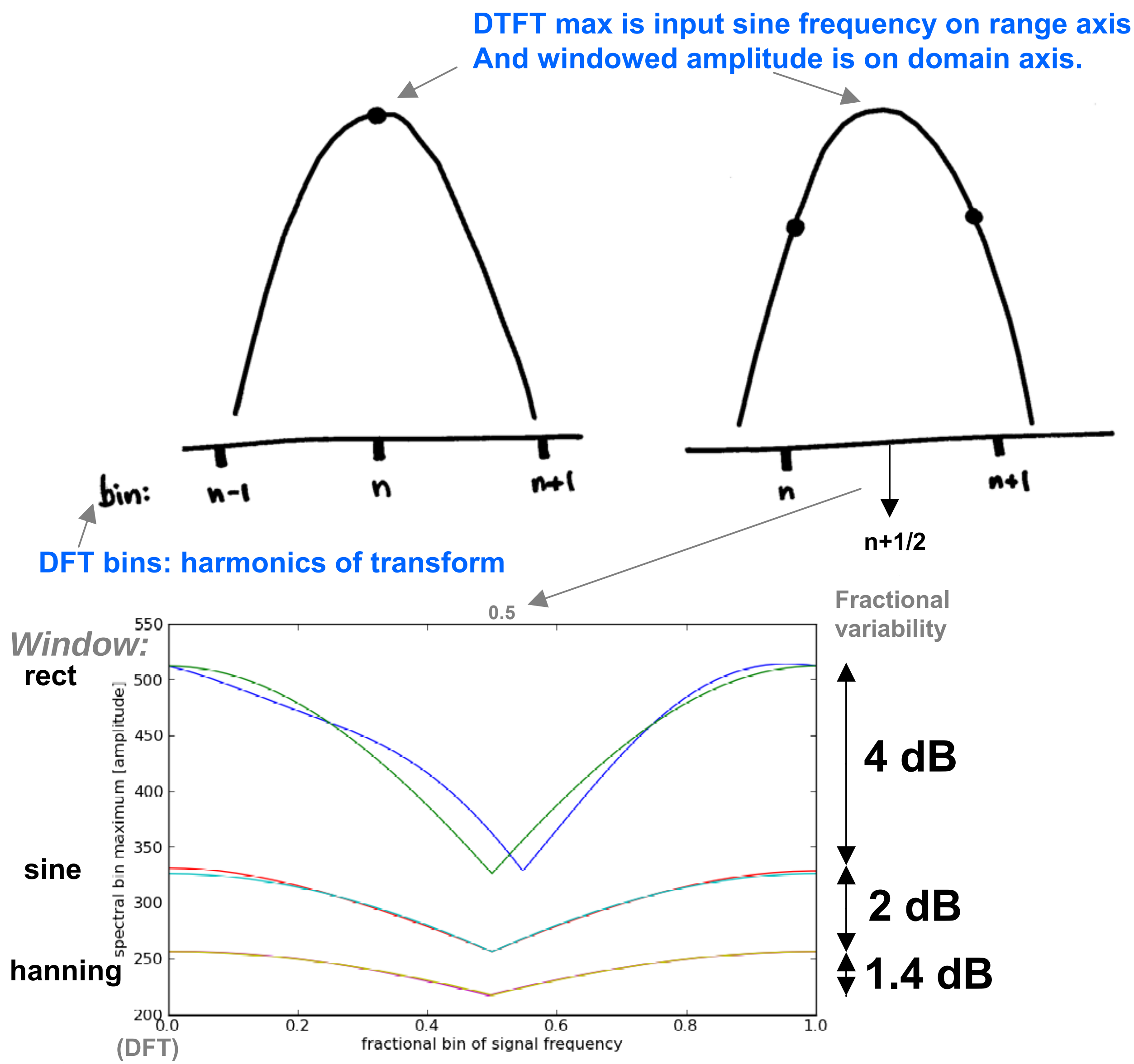
http://en.wikipedia.org/wiki/Headroom_%28audio_signal_processing%29

FFT speed




```
import pylab as pl
import numpy as np
import numpy.fft as npf
import time
N = 5000
domain = np.arange(1,N+1)
range  = np.zeros(N)
dummy  = np.zeros(N)
pow2i  = [2**n-1 for n in xrange(1 \
     +int(np.log2(N)))]

for i in xrange(N):
   time1 = time.clock()
   dummy = npf.fft(np.arange(domain[i]))
   time2 = time.clock()
   range[i] = time2 - time1

pl.plot(domain, range, 'b+')
pl.plot(domain[pow2i], range[pow2i], 'rp')
pl.xlabel('transform size')
pl.ylabel('time [seconds]')
pl.suptitle('numpy.fft length v calculation time')
pl.show()
```

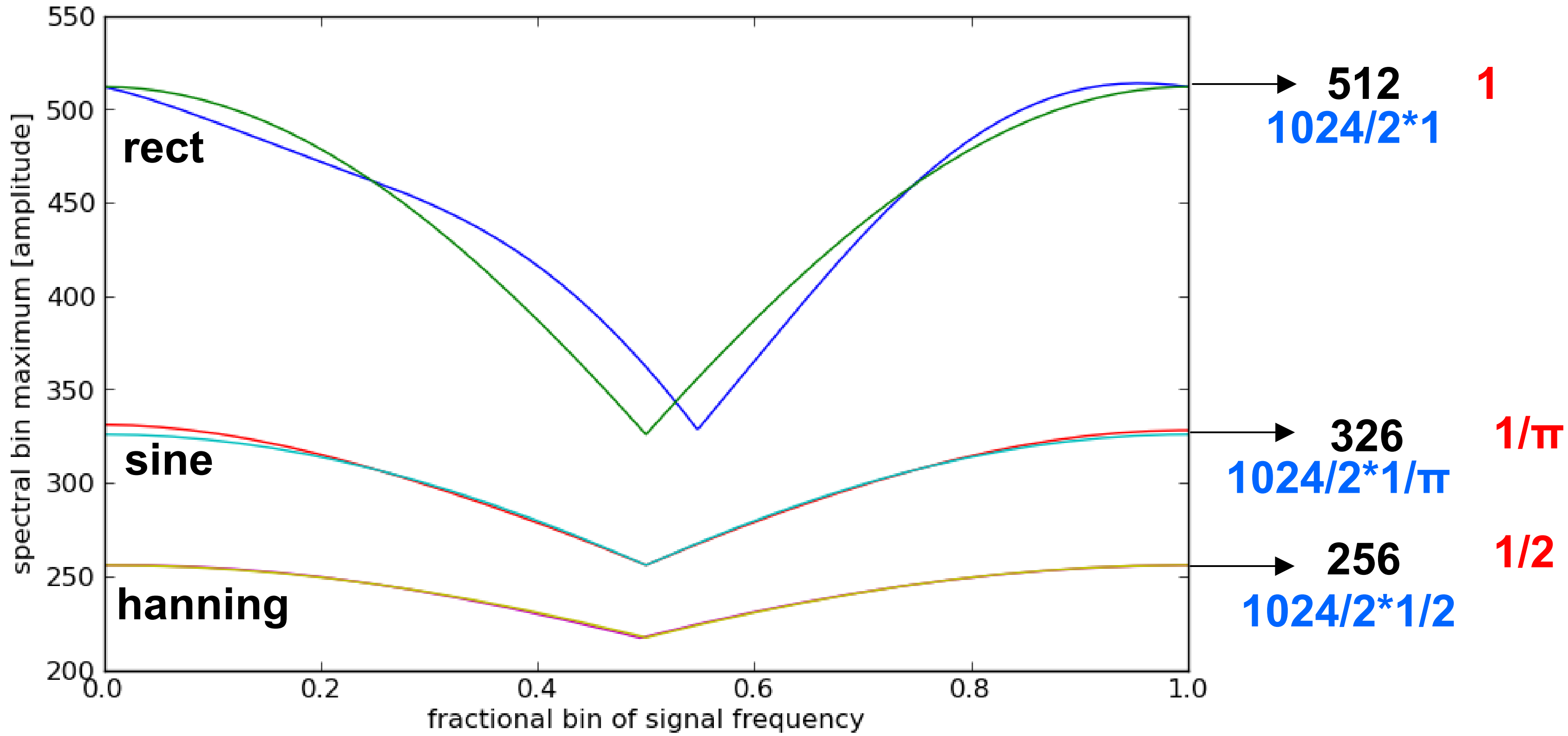
Fractional bin sinewave




```
import pylab     as pl
import numpy     as np
import numpy.fft as npf
N = 1024
from craigwindow import *
sinwin = SineWindow(N)
hanwin = HanningWindow(N)

lowrange     = np.zeros(N+1)
lowrangesin  = np.zeros(N+1)
lowrangehan  = np.zeros(N+1)
for i in xrange(N+1):
   sig           = np.sin(2 * np.pi * (2.0+1.0*i/N) * np.arange(N) / N)
   spec          = npf.fft(sig)
   absspec       = np.abs(spec)
   lowrange[i]    = max(absspec)
   spec          = npf.fft(sig*sinwin)
   absspec       = np.abs(spec)
   lowrangesin[i] = max(absspec)
   spec          = npf.fft(sig*hanwin)
   absspec       = np.abs(spec)
   lowrangehan[i] = max(absspec)

hirange      = np.zeros(N+1)
hirangesin   = np.zeros(N+1)
hirangehan   = np.zeros(N+1)
for i in xrange(N+1):
   sig           = np.sin(2 * np.pi * (N/4+1.0*i/N) * np.arange(N) / N)
   spec          = npf.fft(sig)
   absspec       = np.abs(spec)
   hirange[i]     = max(absspec)
   spec          = npf.fft(sig*sinwin)
   absspec       = np.abs(spec)
   hirangesin[i]  = max(absspec)
   spec          = npf.fft(sig*hanwin)
   absspec       = np.abs(spec)
   hirangehan[i]  = max(absspec)

pl.clf()
pl.plot(np.arange(N+1.0)/N,lowrange,
       np.arange(N+1.0)/N,hirange,
       np.arange(N+1.0)/N,lowrangesin,
       np.arange(N+1.0)/N,hirangesin,
       np.arange(N+1.0)/N,lowrangehan,
       np.arange(N+1.0)/N,hirangehan)
pl.xlabel("fractional bin of signal frequency")
pl.ylabel("spectral bin maximum [amplitude]")
pl.show()
```

FFT Sine Amplitude v Window

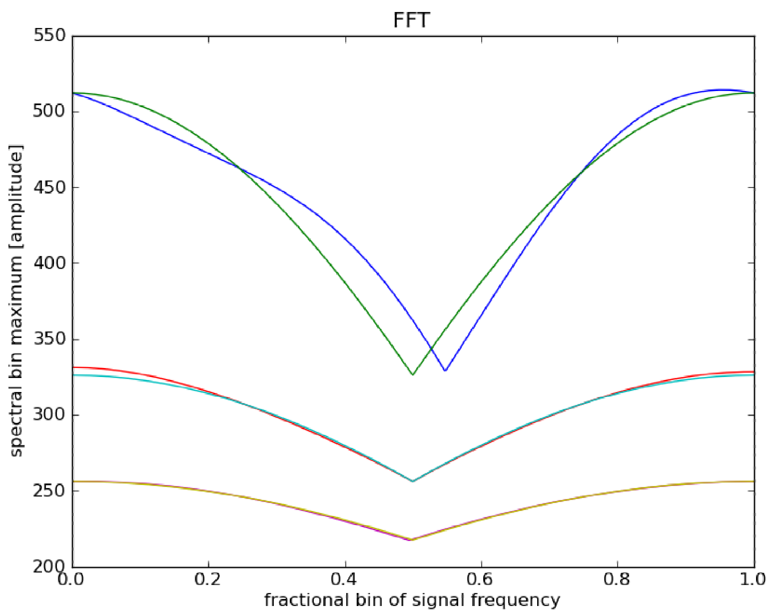
N=1024

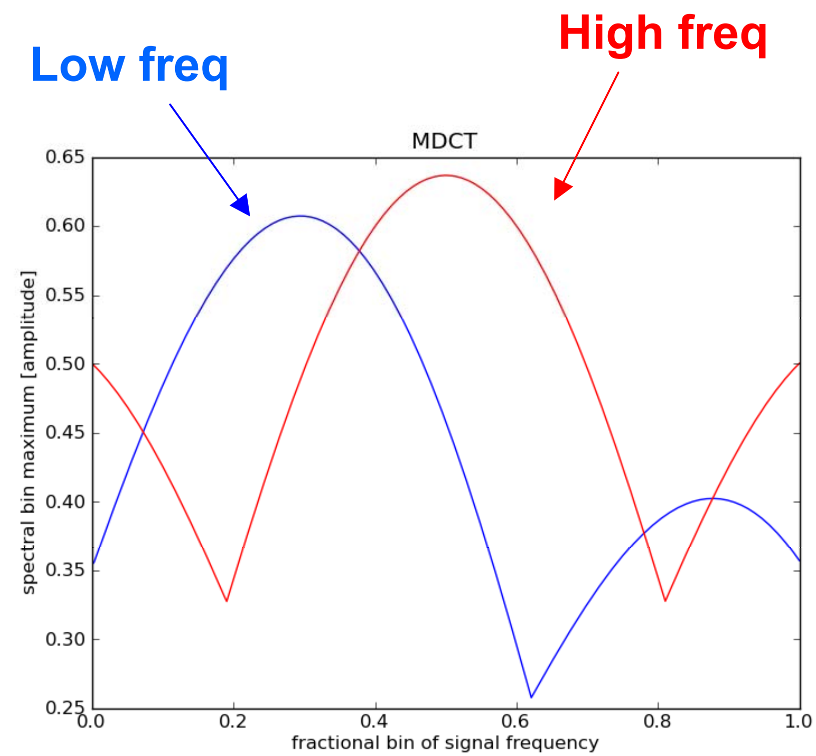


Spectral |amplitude| is sine peak amplitude/2 * integration of signal window

(signal window normalized by rectangular window)

MDCT Sine Amplitude





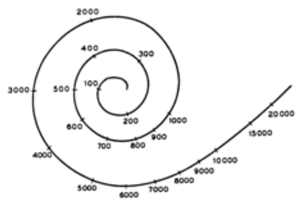
Maximum signal with sinwindow
Generate 1/sqrt(2) magnitude

Bark Scale

- Bark scale is a perceptual frequency mapping expressing how humans hear.

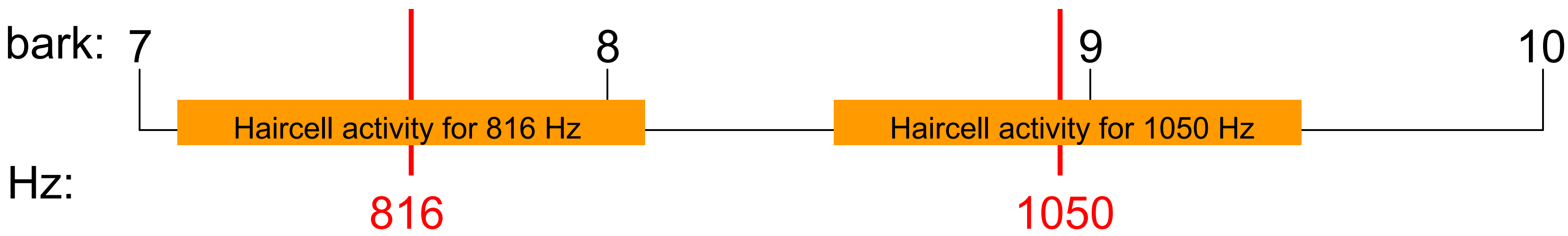
- Approximately equivalent to the spatial positon on the basilar membrane.

(note inner hair cells don't have equal density along basilar menbrane, so only an approximation of distance)



bark	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Hz	20	100	200	300	400	510	630	770	920	1080	1270	1480	1720	2000	2320	2700	3150	3700	4400	5300	6400	7700	9500	12k	15.5k	20k

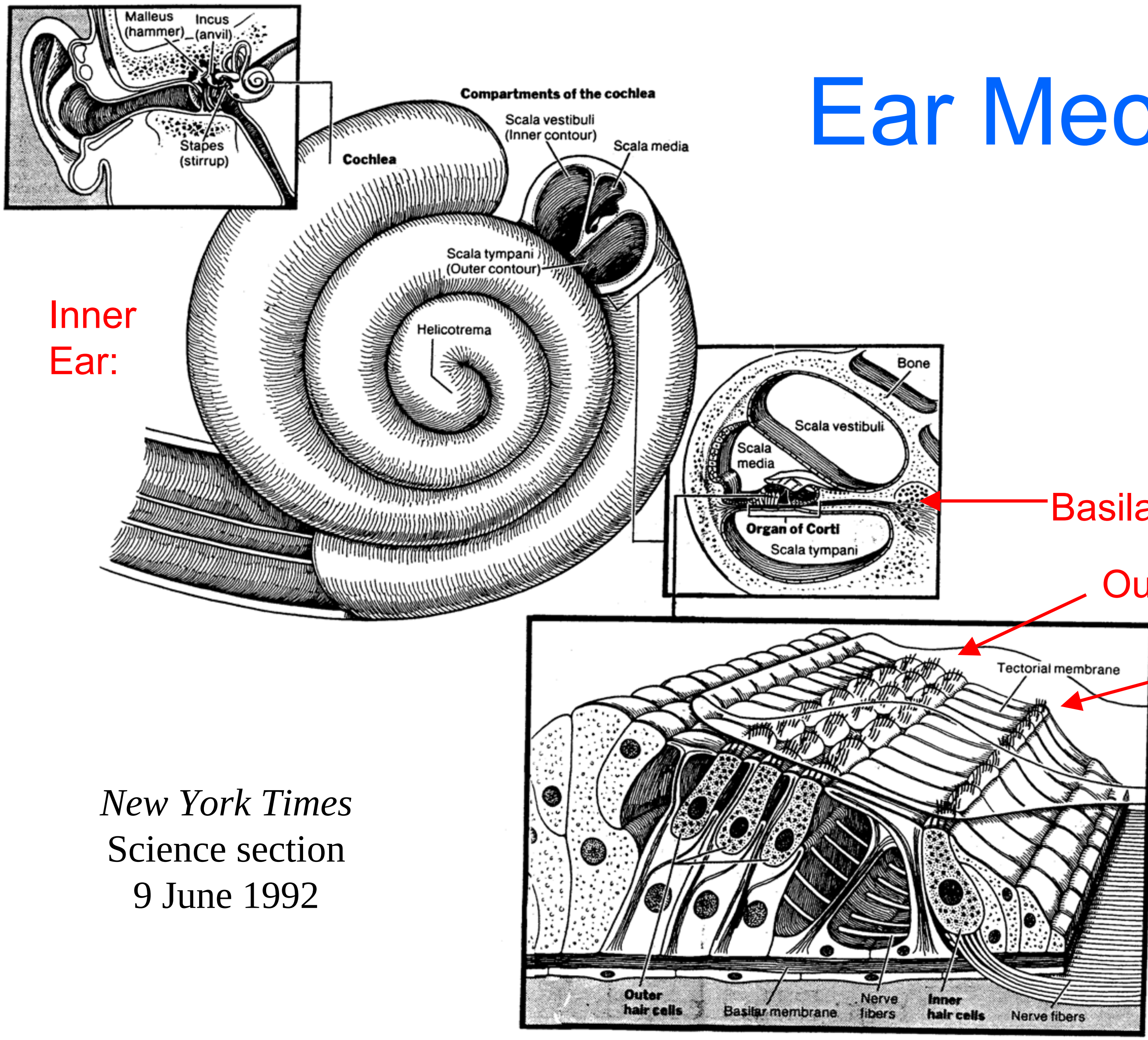
- Difference of one bark is a rough estimate of a critical band which is an estimate how many hair cells are activated by a sinewave (ignoring amplitude variations).



- 3,500 inner haircells = about 140 cells /critical band.

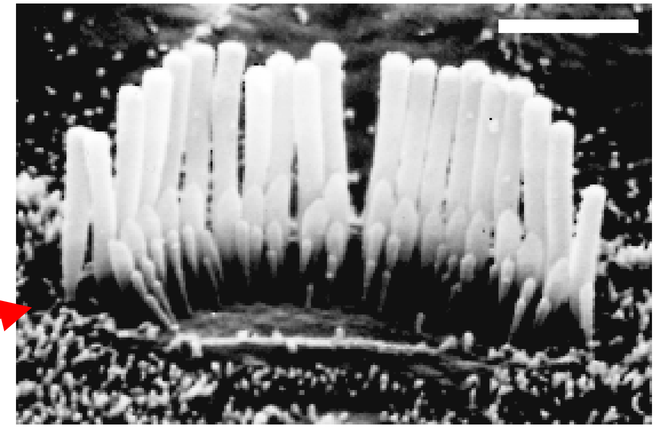
Ear Mechanics

Inner Ear:

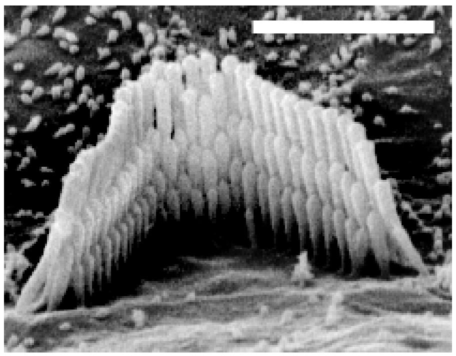


New York Times
Science section
9 June 1992

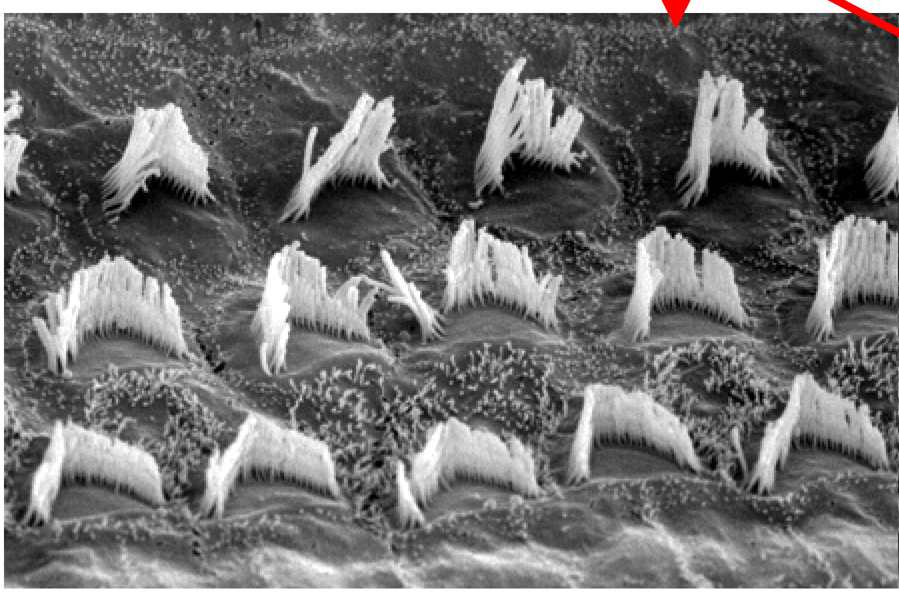
Ear Mechanics (2)

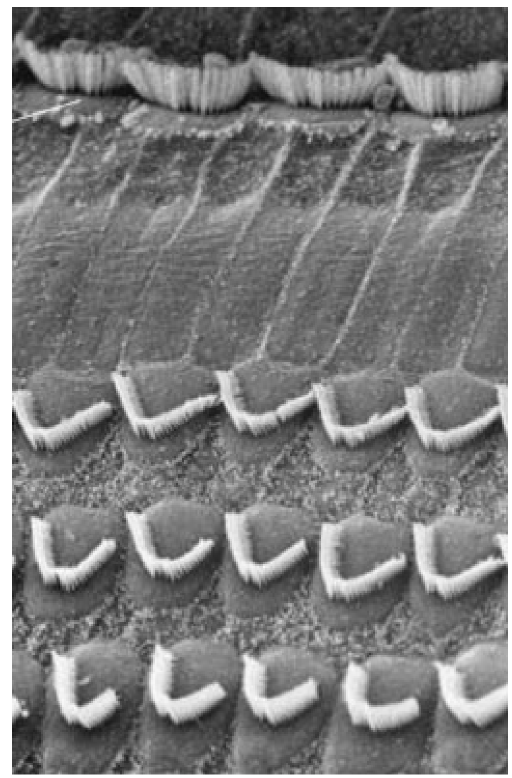


Inner Hair Cells:
each analogous to
FFT/MDCT bin



Outer Hair Cells:
resolution
enhancement





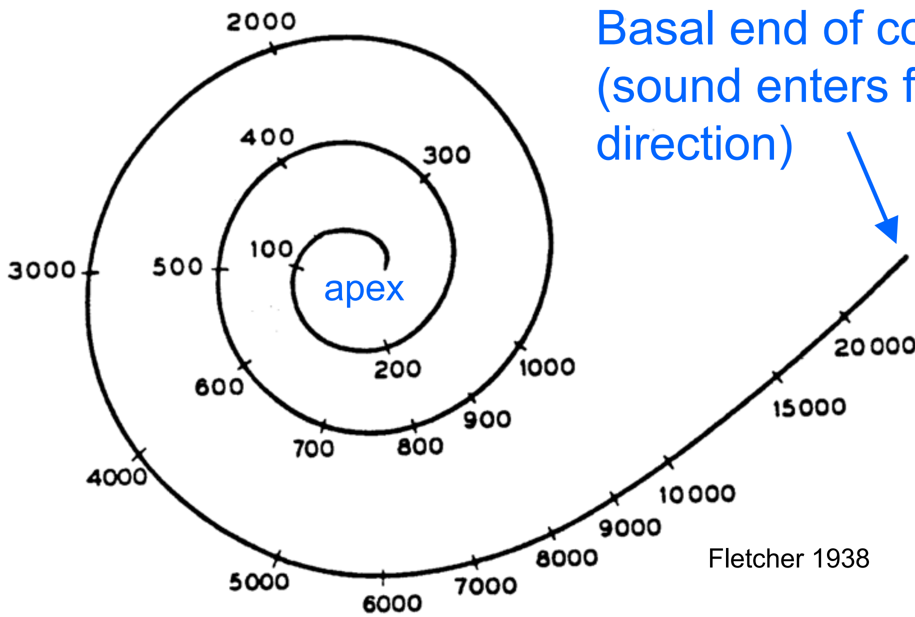
http://www.einstein.yu.edu/aif/gallery/haircells/haircell.gif

http://scienceblogs.com/retrospectacle/upload/2006/06/hair%20cells.bmp

Ear Mechanics (3)

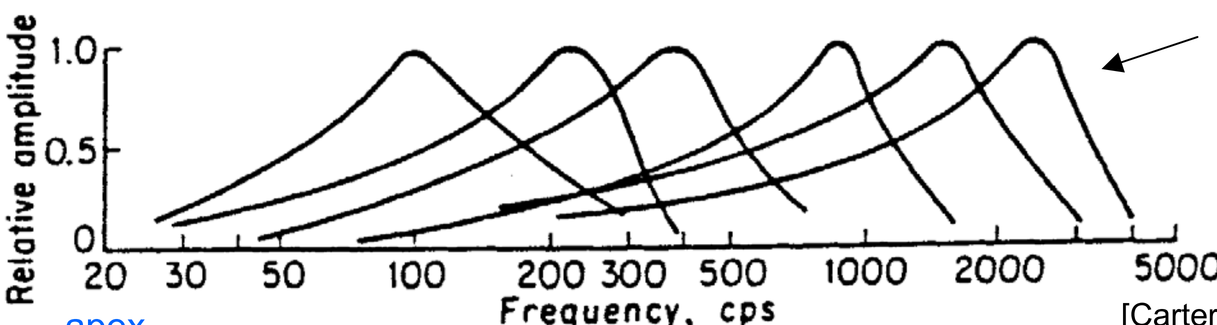
- About 80 inner hair cells per mm at the basal end and 155 cells per millimeter at the apical end [Bredberg 1968]

Basal end of cochlea (sound enters from this direction)



Fletcher 1938

- 3,500 inner haircells along basilar membrane



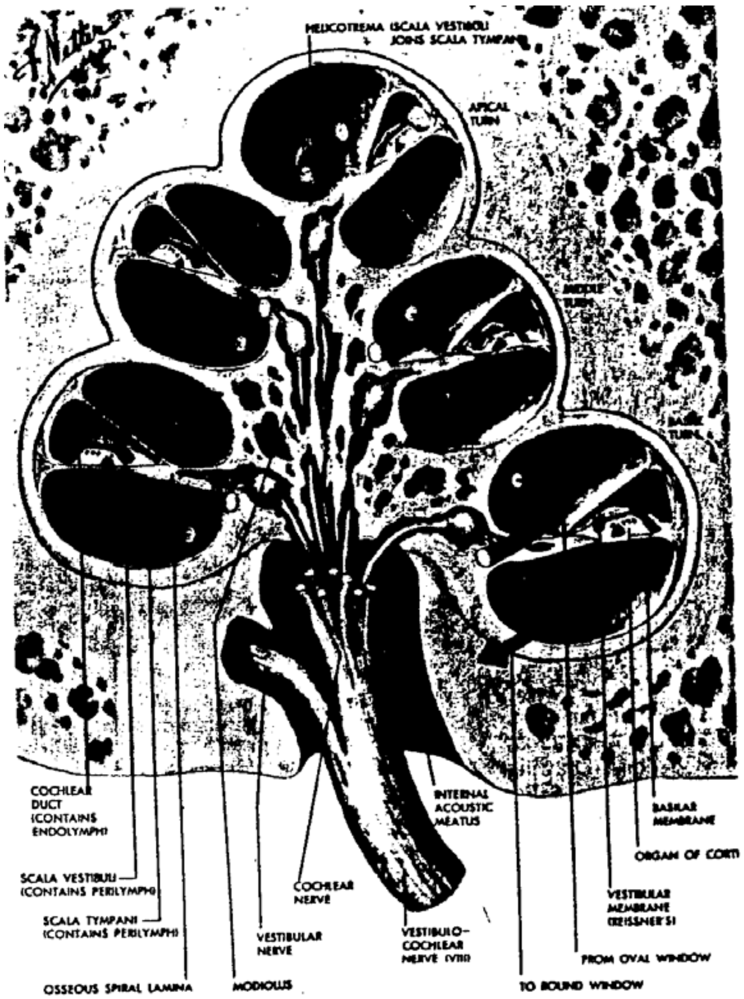
apex

Activation of hair cells for a given sinewave.

From *Experiments in Hearing*, Békésy 1960

[Carterette: Handbook of Perception: IV Hearing]

More Schematics of Cochlea



[Frank Netter]

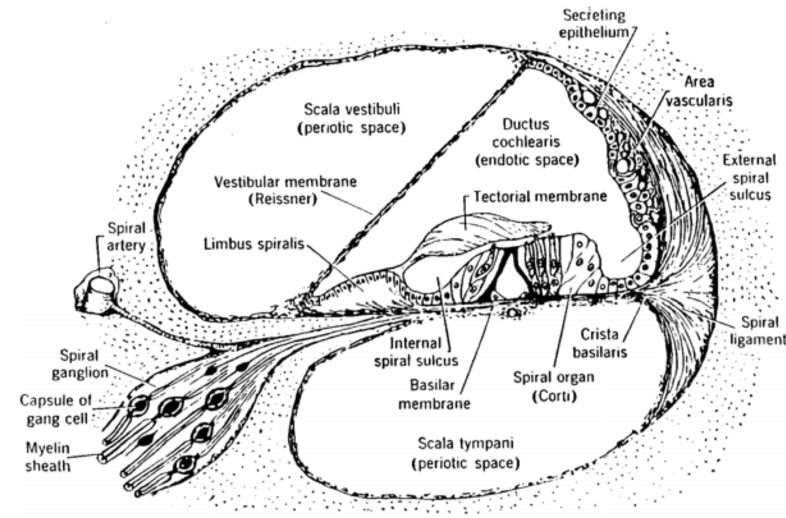


FIG. 1. Diagrammatic cross section of a cochlear canal. The ductus cochlearis (or scala media) contains the organ of Corti with its hair cells, the ultimate end organs of hearing. (From Rasmussen, 1943.)

[Stevens p. 1117]

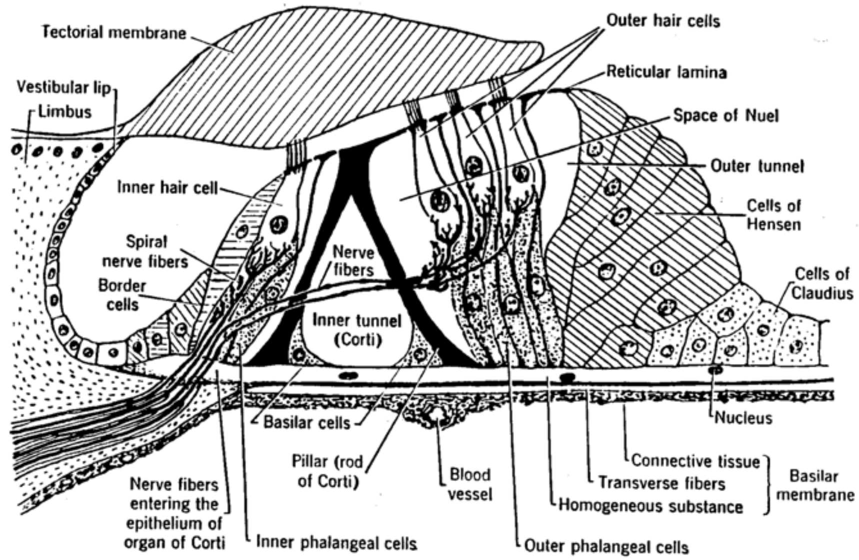
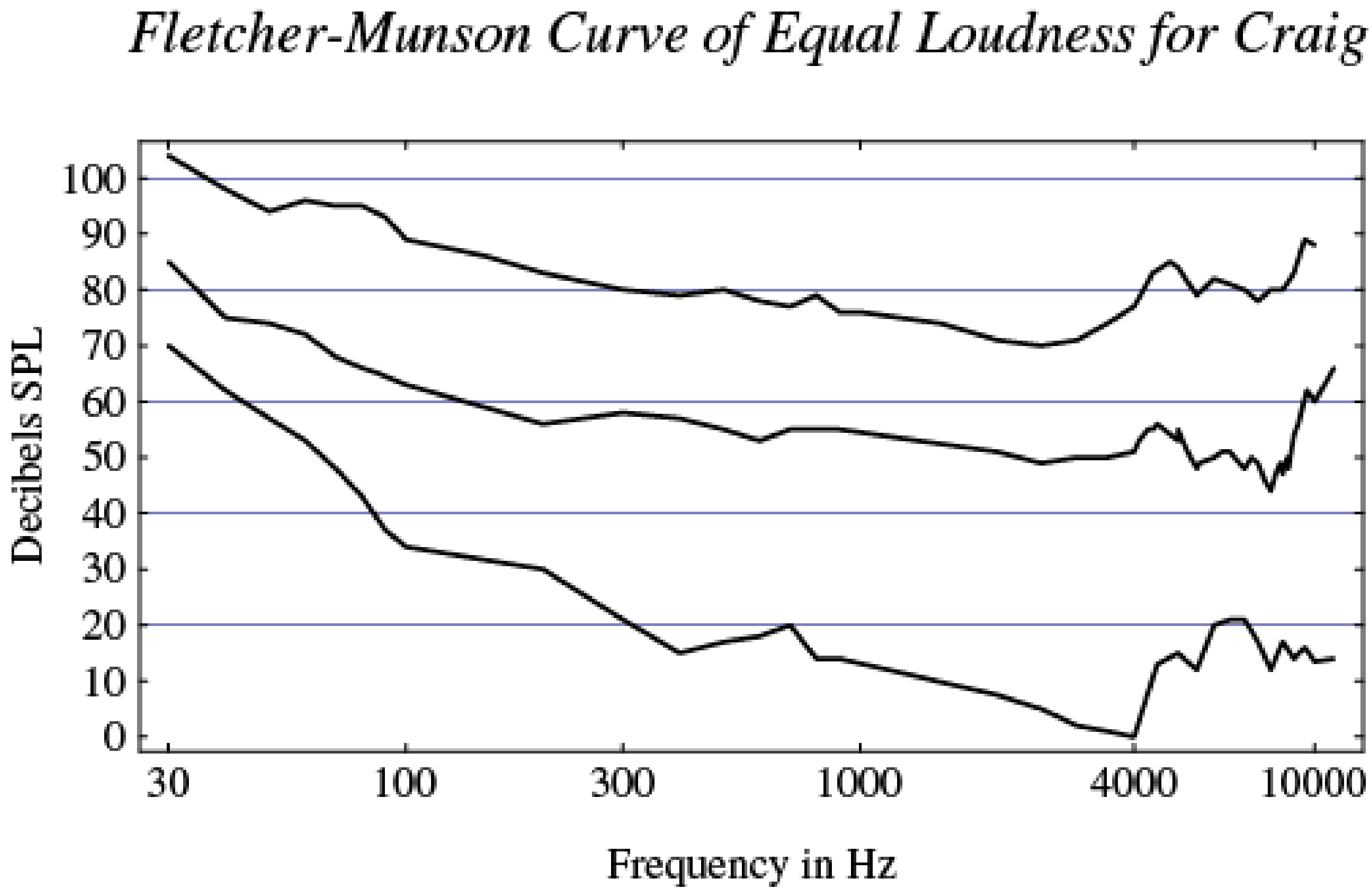


FIG. 2. Diagrammatic cross section of the organ of Corti. The outer hair cells are supported by their respective phalangeal cells, which rest in turn on the movable basilar membrane. The phalangeal cells supporting the inner hair cells rest on bone. Motion of the basilar membrane presumably distorts the hair cells. (From Rasmussen, 1943.)

[Stevens p. 1118]

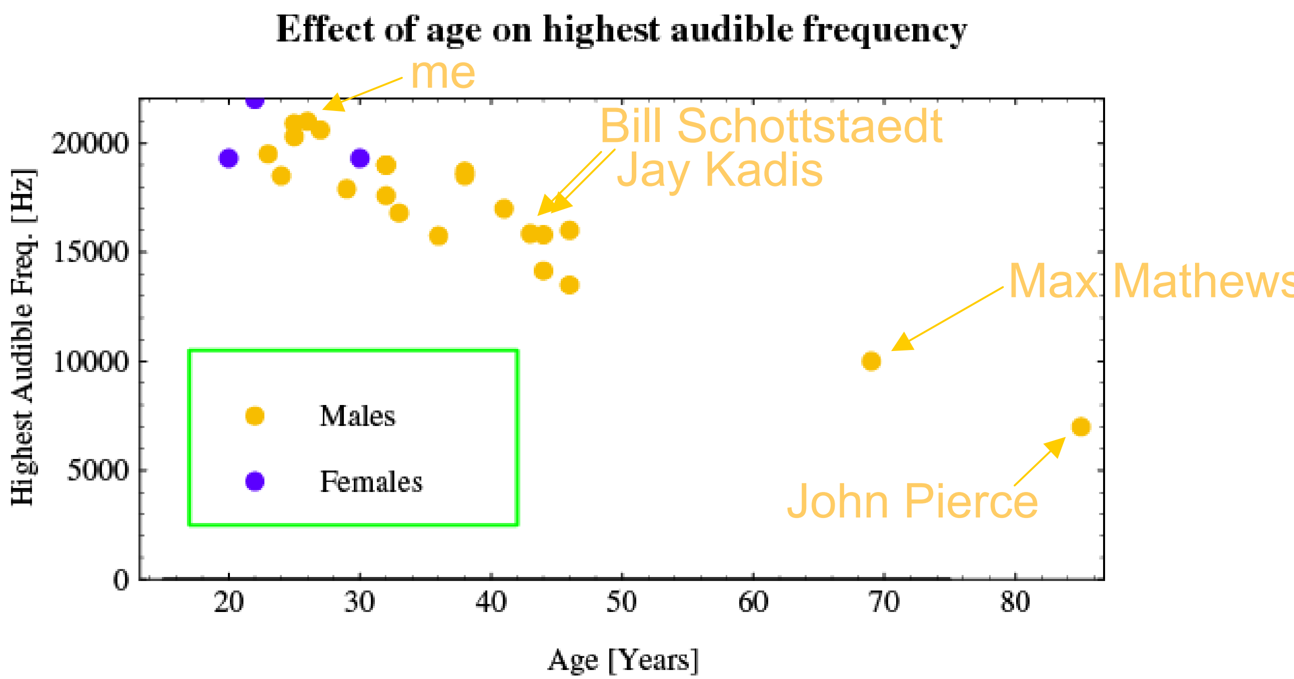
Equal Loudness Curves



http://ccrma.stanford.edu/CCRMA/Courses/SummerWorkshops/96/Psychoacoustics/labs/loudness

- Project idea: Use your own threshold of hearing in the masking model.

Highest Audible Frequency

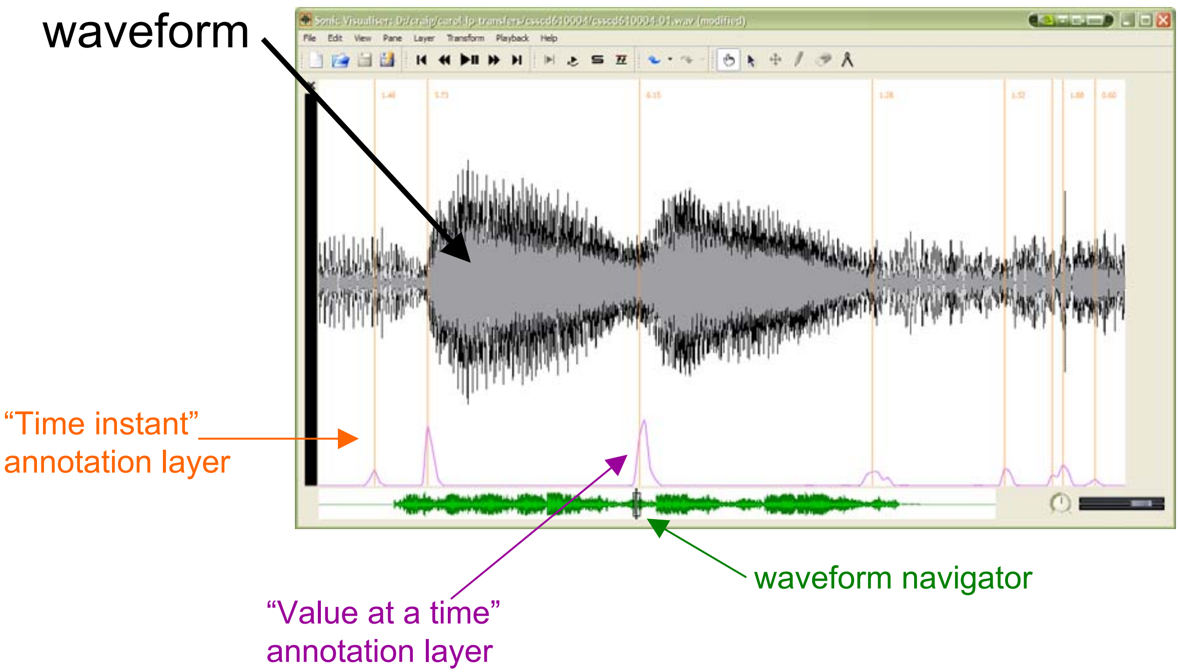


- http://ccrma.stanford.edu/CCRMA/Courses/SummerWorkshops/96/Psychoacoustics/labs/loudness
- http://www.bbc.co.uk/wiltshire/content/articles/2006/04/04/mosquito_sound_wave_feature.shtml

- About 5% of <20 year olds can hear up to 25 kHz.
- Easy to do high-compression ratios for Senior Citizens.

SONIC VISUALISER

• Audio annotation program: http://www.sonicvisualiser.org/



• **Vamp Plugins**: audio analysis plugins in C++ for Sonic Visualiser

http://www.vamp-plugins.org

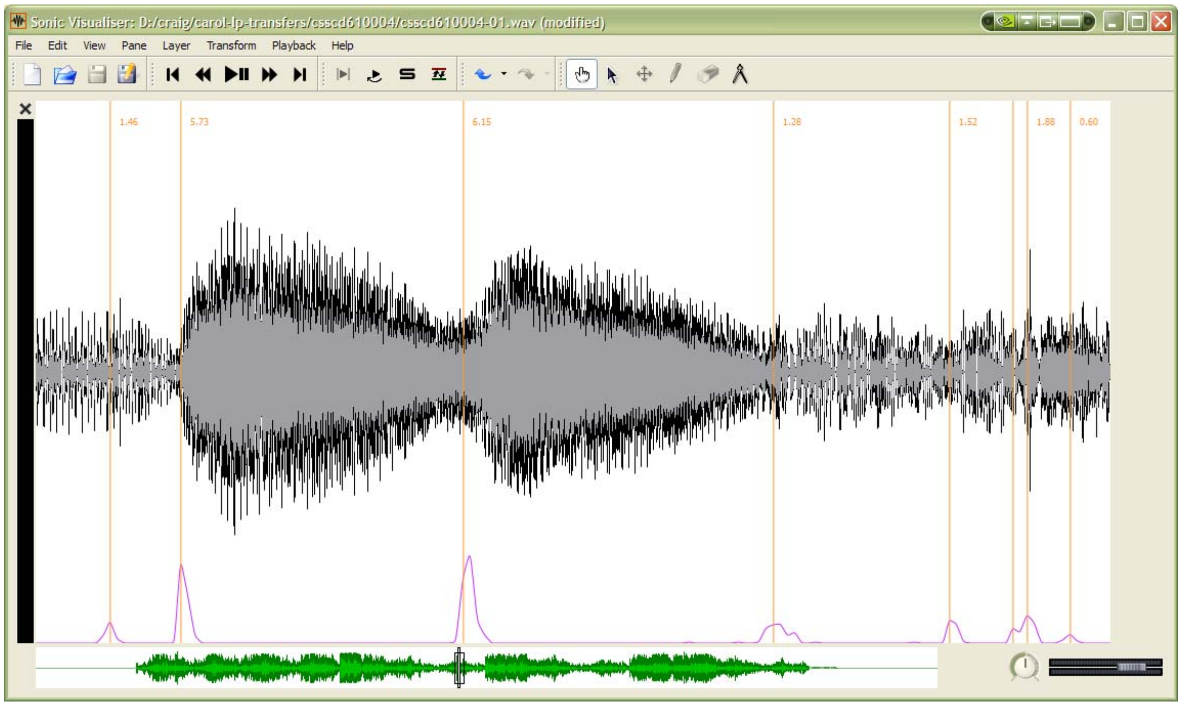
• **VamPy:** Python interface to Vamp plugins for Sonic Visualiser

http://www.vamp-plugins.org/vampy.html

Spectral Reflux Plugin

http://sv.mazurka.org.uk/download
(linux and windows)

- Useful for a window-switching project



Peaks (onset times):

58.318208616	5.73
58.718208616	6.15
59.158208616	1.28
59.408208616	1.52

Analysis (every 10 ms):

58.178208616	0
58.188208616	0
58.198208616	0
58.208208616	0.522996
58.218208616	**1.46201**
58.228208616	0.240144
58.238208616	0
58.248208616	0
58.258208616	0
58.268208616	0
58.278208616	0
58.288208616	0
58.298208616	0
58.308208616	0
58.318208616	**5.72949**
58.328208616	3.21772
58.338208616	0.4205
58.348208616	0
58.358208616	0
58.368208616	0
58.378208616	0
58.388208616	0
58.398208616	0
58.408208616	0
58.418208616	0
58.428208616	0
58.438208616	0
58.448208616	0
58.458208616	0
58.468208616	0
58.478208616	0
58.488208616	0
58.498208616	0
58.508208616	0
58.518208616	0

58.528208616	0
58.538208616	0
58.548208616	0
58.558208616	0
58.568208616	0
58.578208616	0
58.588208616	0
58.598208616	0
58.608208616	0
58.618208616	0
58.628208616	0
58.638208616	0
58.648208616	0
58.658208616	0
58.668208616	0
58.678208616	0
58.688208616	0
58.698208616	0
58.708208616	0.0118438
58.718208616	4.44351
58.728208616	**6.15389**
58.738208616	1.40132
58.748208616	0.508545
58.758208616	0
58.768208616	0
58.778208616	0
58.788208616	0
58.798208616	0
58.808208616	0
58.818208616	0
58.828208616	0
58.838208616	0
58.848208616	0