

# Design and Implementation of Speech Recognition Systems

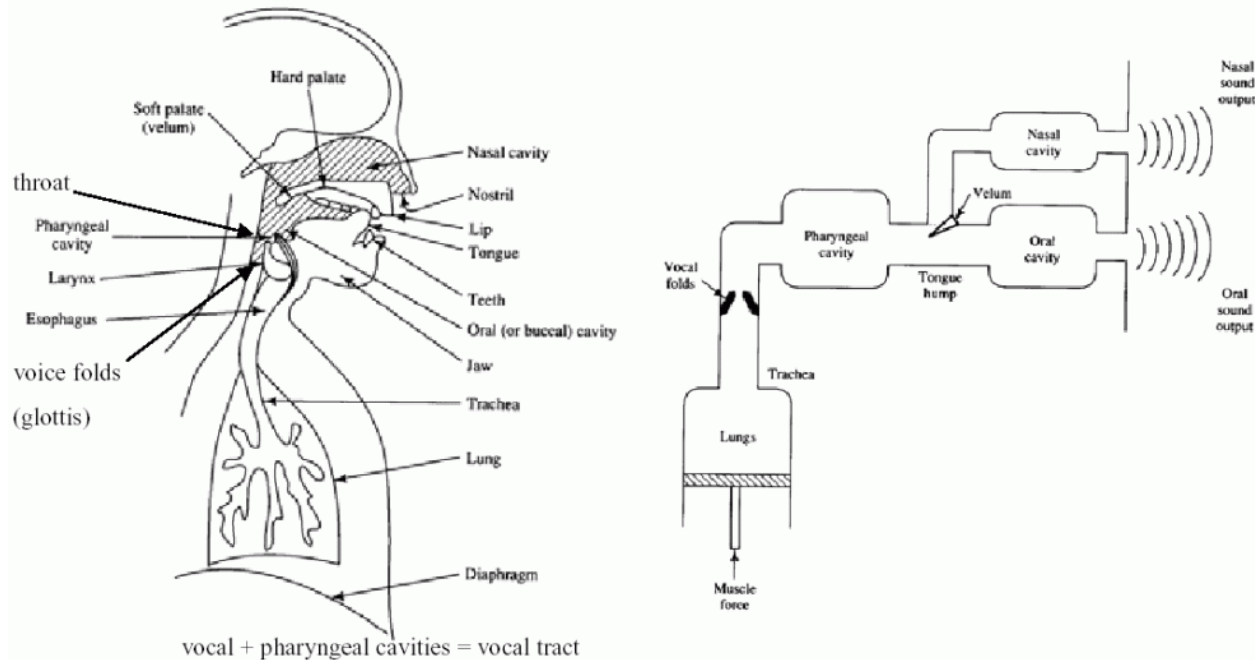
*Spring 2011*

*Bhiksha Raj, Rita Singh*

## **Class 2: Data Capture**

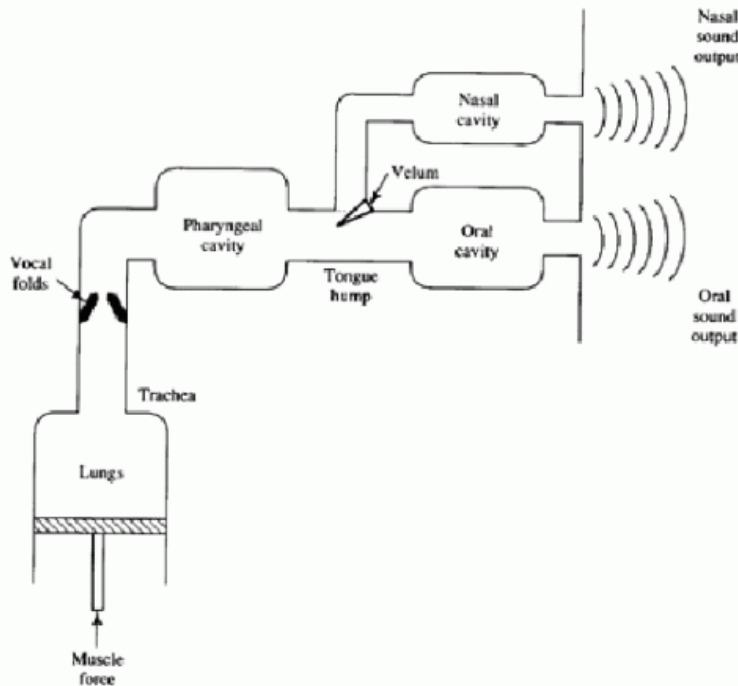
24 Jan 2011

# Producing the Speech Signal



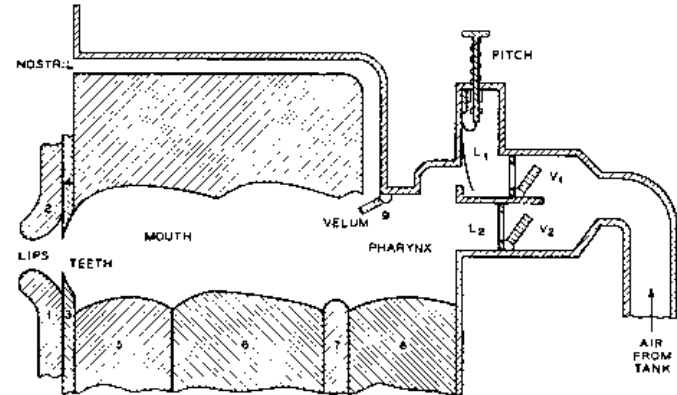
- All sounds are actually pressure waves
- The speech producing mechanism is resonant chamber that produces pressure waves
  - Left: Cross section of human vocal tract
  - Right: A physical “model”

# The speech production mechanism



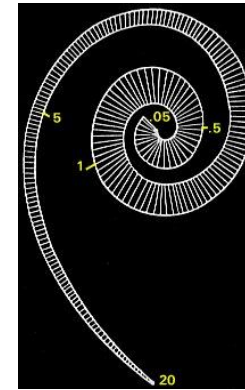
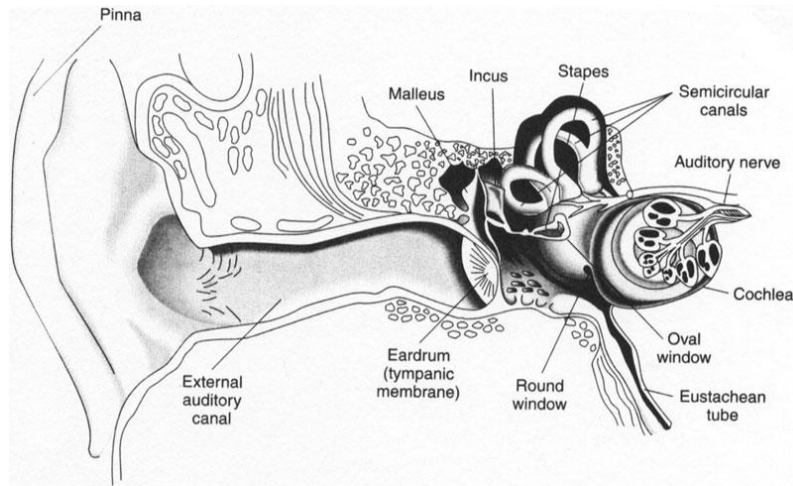
- The lungs are like bellows: they pump air out to produce speech
  - You only speak while breathing *out*
- The vocal folds may vibrate to add a basic (fundamental) frequency
  - To get semi-periodic puffs of air going up the vocal tract
- The muscles shape the vocal tract to add other resonances (and anti-resonances) in the air traveling up the tract
- The lips affect the final nature of the air puffs coming out of the mouth.

# Some Models



- Left: Wolfgang von Kemeplén's bellows: 1773
  - Imitated the bellows and tube nature of the vocal tract
    - Included a bellows, a reed (larynx) and tubes
- Right: Robert Reisz's 1937 machine
- Could produce simple vowel- and consonant-like sounds

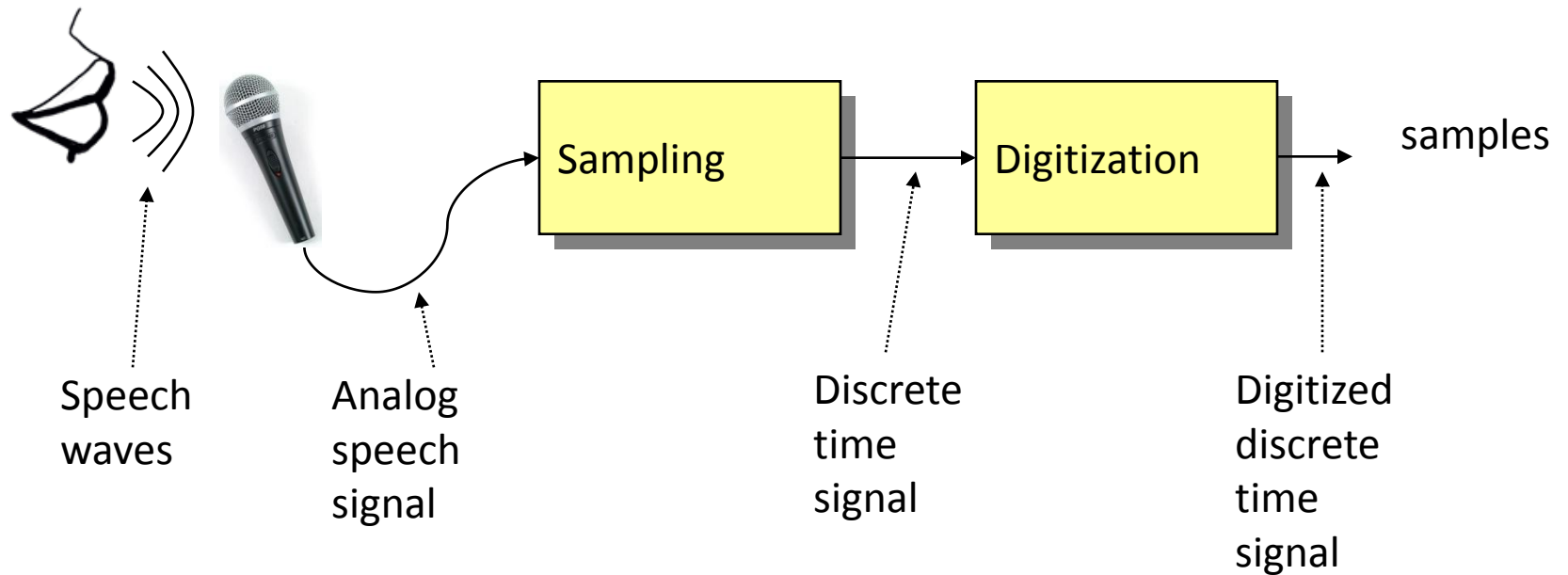
# The *Hearing* Apparatus



- The human auditory system consists of
  - A membrane that is moved back and forth by the pressure waves
  - A complex mechanism that converts the movement of this membrane to the motion of fluid
  - A collection of tuning forks that resonate to the fluid motion
  - Our brain interprets the movement of the turning forks
    - Actually hair fibres in the cochlea
- The key component is the moving membrane that translates pressure waves to the movement of fluid

# Signal Capture

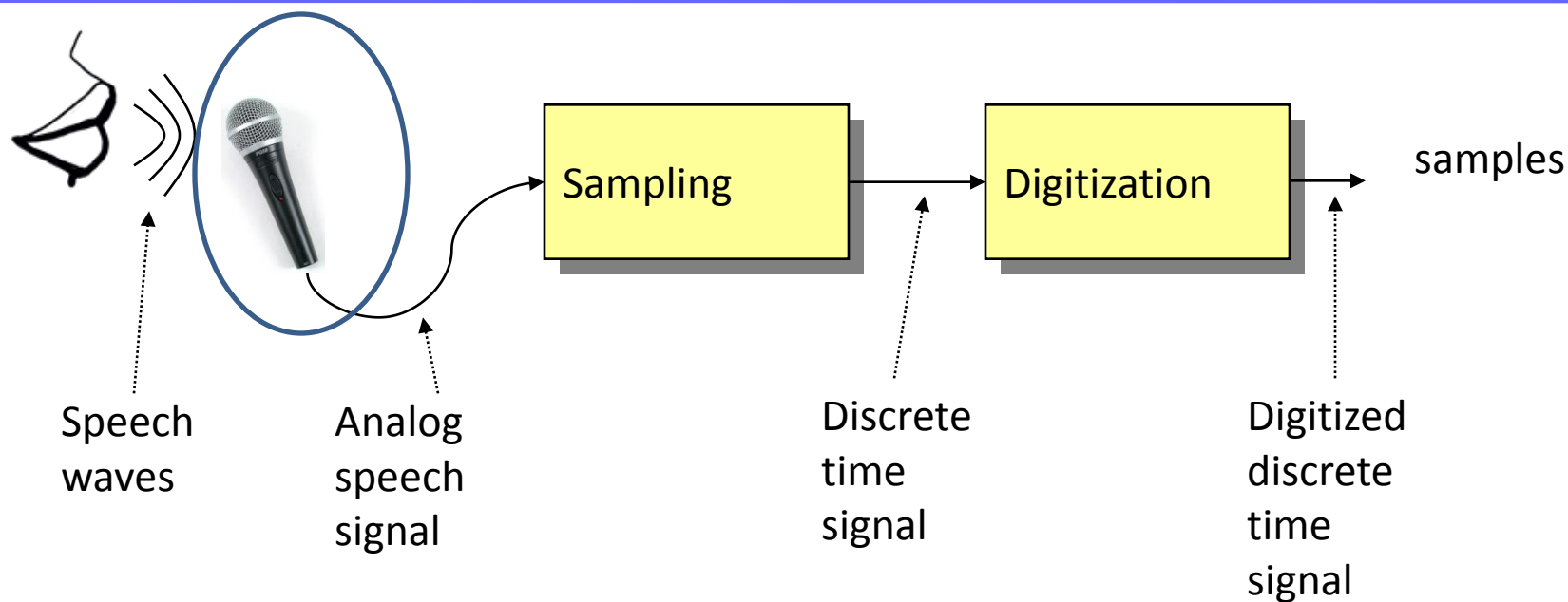
---



- The goal of signal capture is to convert pressure waves to a series of numbers

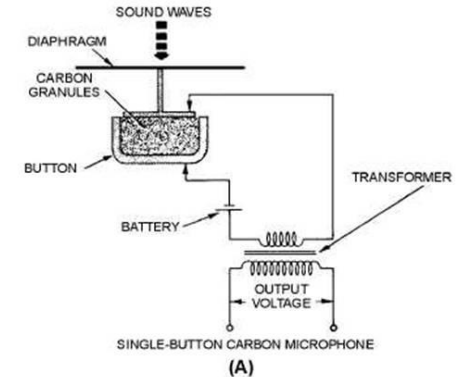
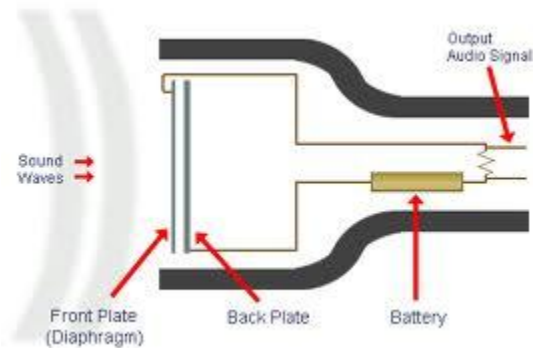
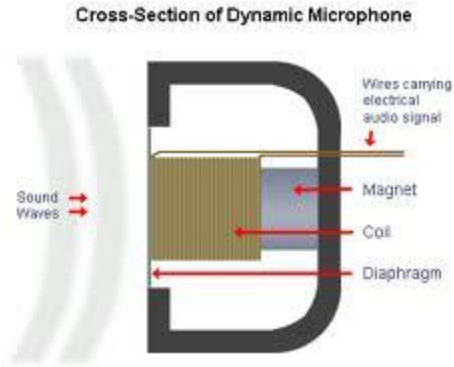
# First Stage: Microphones

---



- The goal of signal capture is to convert pressure waves to a series of numbers
- To do so, we imitate the auditory system
  - A membrane that is moved by pressure waves
  - A transduction mechanism that converts the movement of the membrane to a meaningful signal

# Microphones



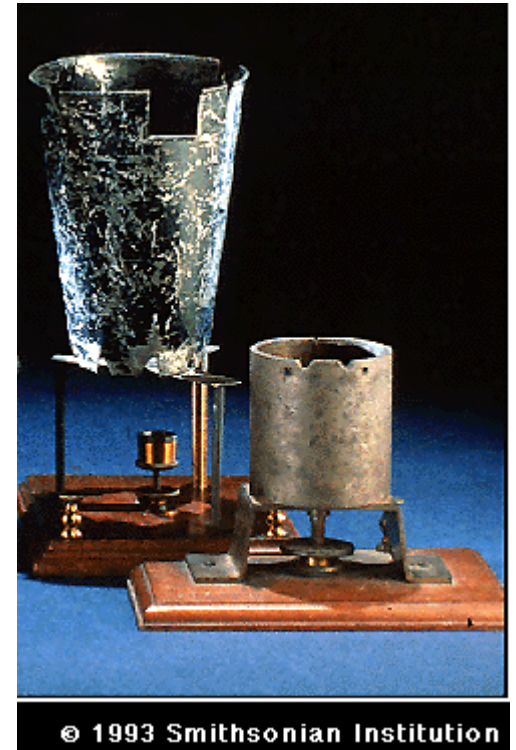
- Comprise
  - a membrane that is moved by the pressure wave
  - Some mechanism for converting this motion to a time-varying electrical signal
- The time varying signal is what we record.



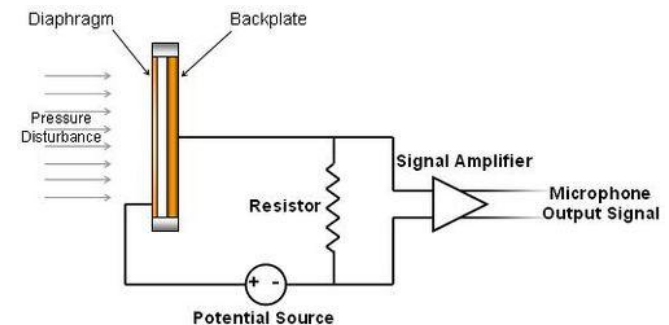
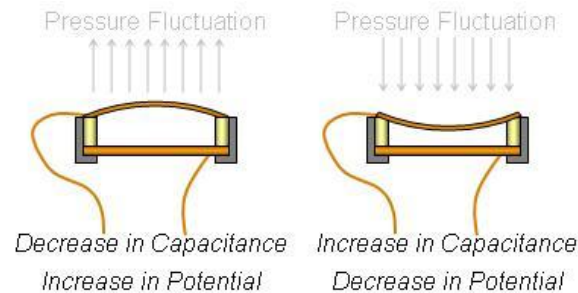
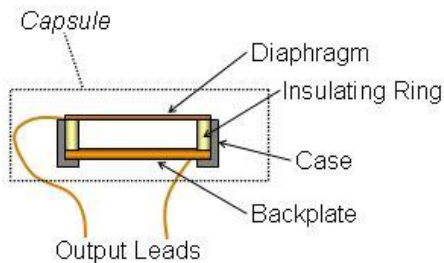
# The First Microphone

---

- Graham Bell's "liquid" microphone
  - Metal cup filled with water, mixed with Sulphuric acid
  - Needle suspended in water (stuck to a diaphragm)
  - Sound waves move the needle up and down
    - Changing the resistance of the cup
  - A fixed voltage induces a time-varying current
- The design was actually "copied" from Elisha Gray
  - Apparently from a drawing Bell saw in the patent office
  - In later incarnations he discarded the liquid microphone

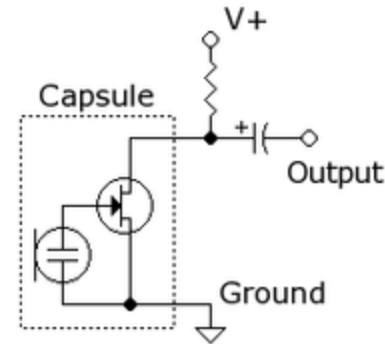
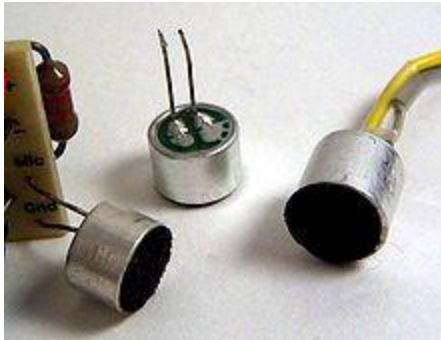


# Condenser Mic: Capacitance based



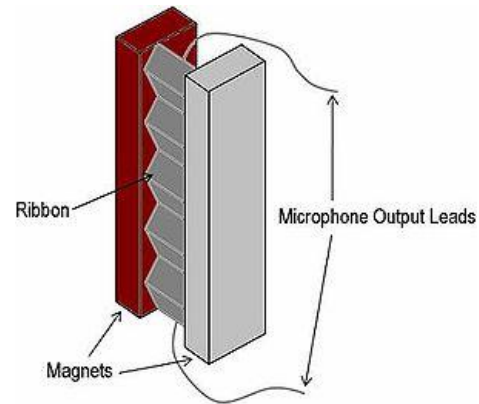
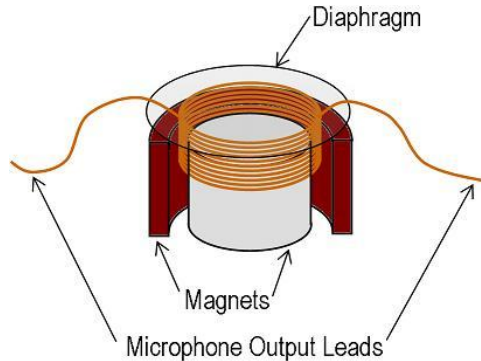
- Diaphragm and back plate maintained at different charge levels
- Pressure waves change the thickness of intervening dielectric (air)
  - Changing the capacitance of the capsule
  - Thereby changing the potential difference across the plates
    - $Q = C \times V$
- The changing potential difference is measured across the large resistor

# Electret Microphones: Capacitive



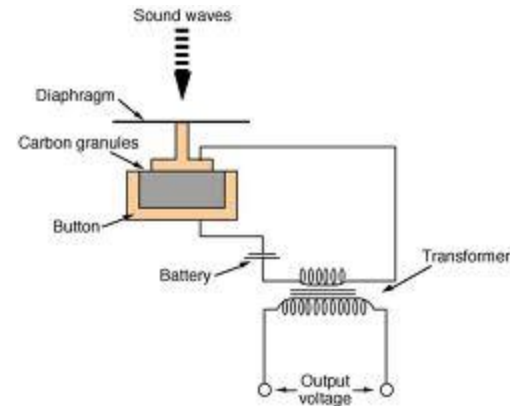
- Electret microphones are Condenser mics with fixed charge level
  - Either the diaphragm or the backplate carry a fixed charge
  - No external voltage source needed to maintain charge
    - But do require power for preamp
    - Usually from the DC
  - Used to be low quality, no longer
- Most computer microphones are electrets

# Dynamic Mics: Inductive



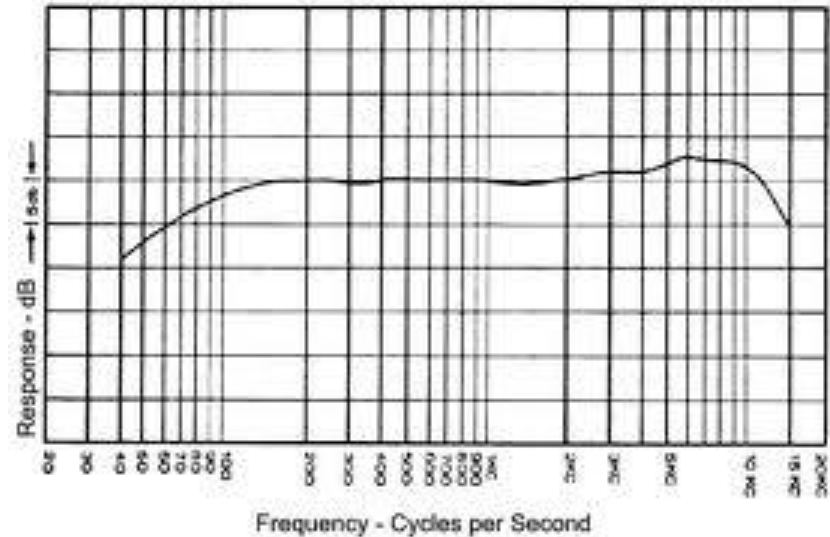
- Based on inductance, rather than capacitance
- Time-varying voltage levels created by moving an element in an electromagnetic field
  - Moving coil mic: Diaphragm connected to a coil, suspended between magnets
    - Movement of diaphragms moves the coil and generates a voltage
  - Ribbon mic: A metal ribbon suspended between magnets
- Does not require phantom power
- Frequency response generally not flat
  - Also, small mics may be noise prone

# Carbon Button Microphones: Resistive



- Based on *resistance*
- Carbon granules between a diaphragm and a backplate
- Motion of the diaphragm changes the resistance of the microphone
- This changes the current in the circuit
  - Typically transformed to a voltage as shown
- Typical in older telephone handsets
  - Cellphones and recent handsets use electrets

# Choice of Microphones

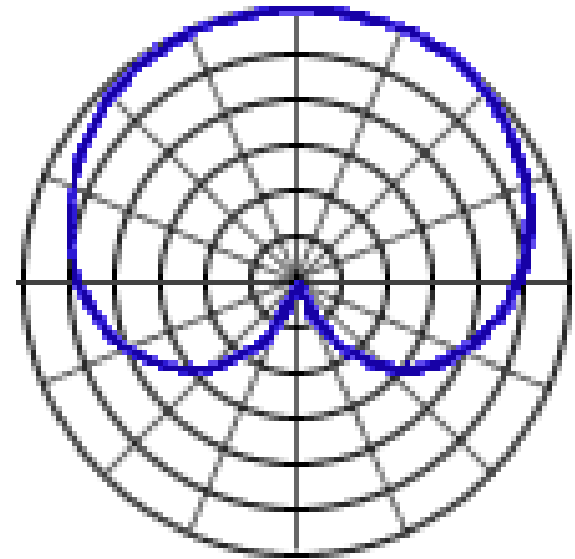


- Characteristics to look for:
  - Frequency response
    - Must be flat in the frequencies of interest
  - Directivity
    - More on this later
  - Noise level
    - Usually stated as dB(A) SPL (Indicates the noise floor of the mic)
    - Lower is better
      - Good microphones: 20dB SPL, Ultra very good mics: 0dB
- Frequency response: Condenser microphones are usually used for high-quality recordings
  - Even cheap electret microphones can be surprisingly good

# The “Directionality” of a Microphone

---

- The response of a microphone depends on the direction of the sound source
- This depends on the construction of the microphone, as well as the shape of its chassis
- The response is typically represented as a polar plot
  - The distance of the curve from the “center” of the plot in any direction shows the “gain” of the mic to sounds from that direction

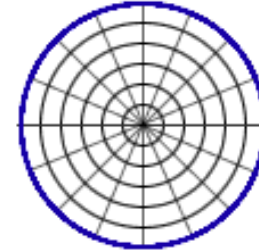


# Typical Directivity Patterns

---

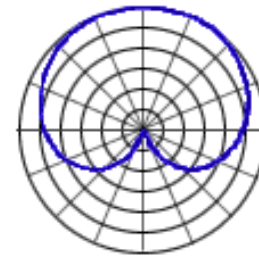
- Omnidirectional

- Picks up sound uniformly from all directions



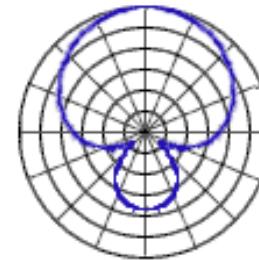
- Cardioid

- Picks up sound from a relatively wide angle of directions from the front and some sound from the side



- Hyper cardioid

- Picks up sound from a relatively narrow angle to the front, but also some noise from behind



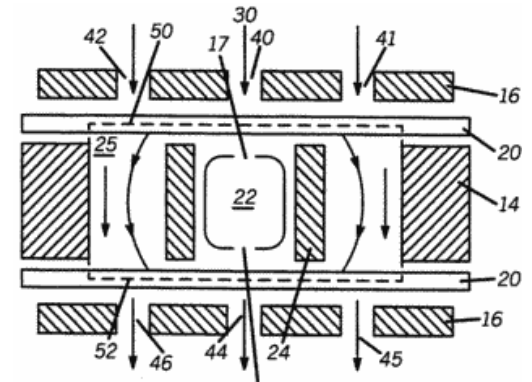


# Directional Patterns

---

- Omnidirectional microphones are useful when we wish to pick up sounds from all directions
  - For speech applications, however, this will result in picking up a lot of noise from non-speech directions
- Where the speaker location can be somewhat localized, Cardioid microphones are more precise
  - Pick up less non-speech noise
  - Still susceptible to noise
- Hypercardioids are a better choice when the speaker location can be well localized
  - Risk of picking up noise from behind
- In general, utilizing microphone directionality is the best way of minimizing extraneous noise from recordings

# Noise Cancelling Mics



- Noise-cancelling microphones are designed to reduce ambient noise in close-talking scenarios
  - Headset mics for example
  - The noise field in the front and back are roughly the same
  - The speech is much more energetic in the front than the back
    - Because of  $1/r^2$  fall off in energy
  - Subtracting the signal from the back to the signal from the front results in selective reduction of noise
    - Also results in a natural high-pass filtering effect that attenuates very-low frequencies
  - Similar results can be obtained by using two microphones

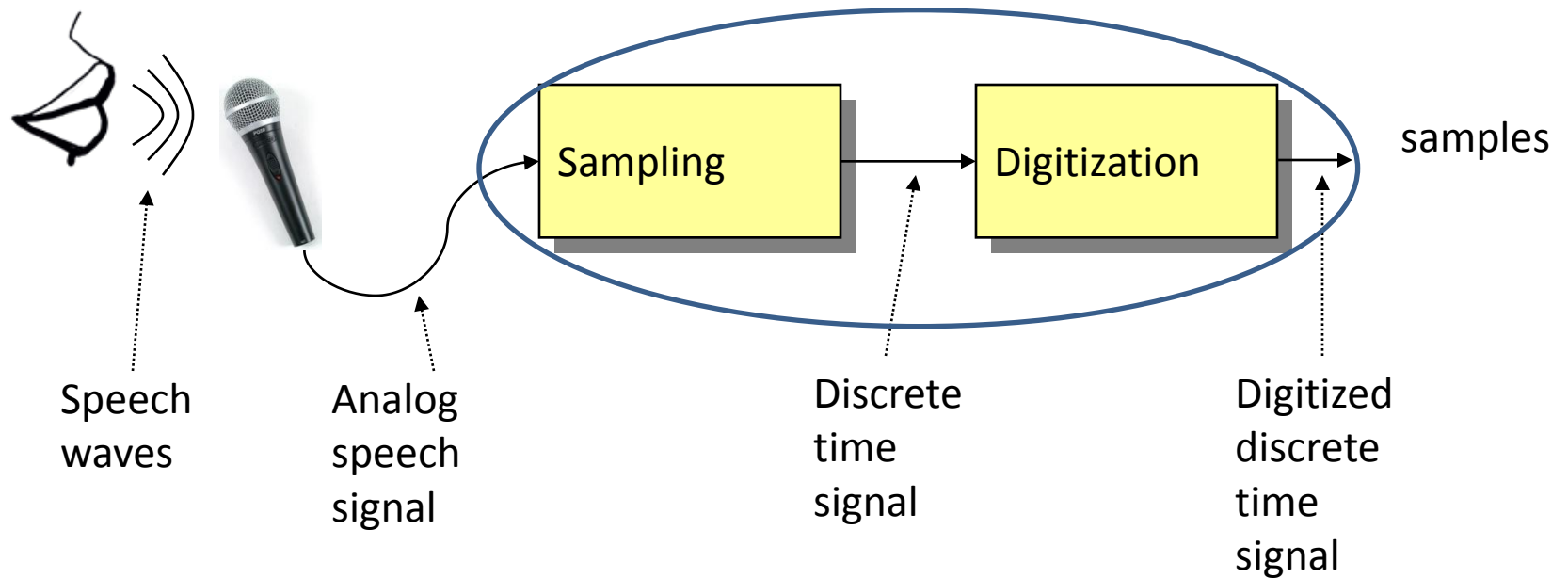
# Wearing/Positioning the Microphone

---



- Microphone must ideally be close enough for high speech level
  - Speech level significantly greater than background noise
- Positioning it too close can result in puffs of air from mouth saturating the mic
  - E.g. sounds such as “P”, “F”, etc.
- To avoid this, desktop mics frequently use a wind sock
  - A foam/sponge covering
- Head-mounted mics must be placed slightly to the side
  - Away from the direct path of puffs

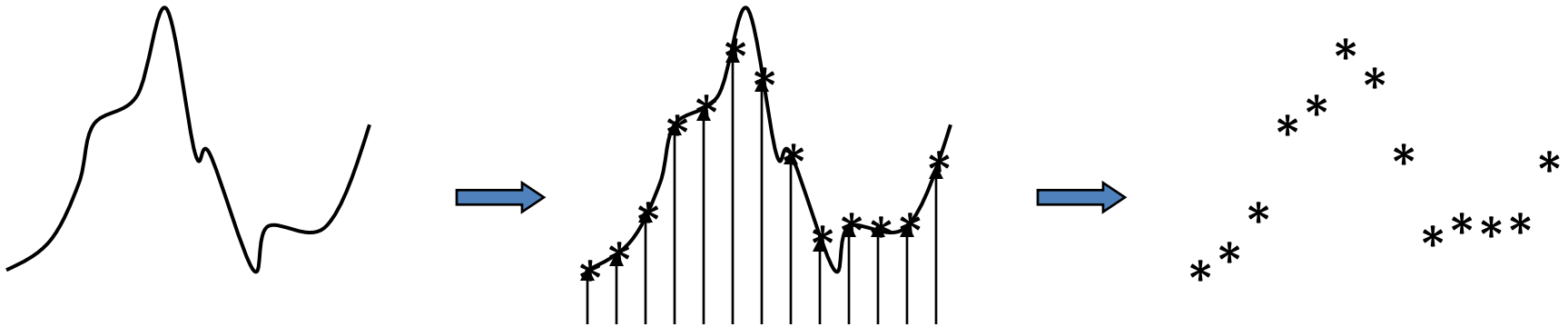
# Sampling and Digitization



- The signal from the microphone goes through a pre-amp
  - The gain of which can be adjusted
  - The output of the pre-amp is a continuous-time electrical signal
    - Usually a voltage signal
- The signal is digitized by an analog to digital converter
  - The signal value is sensed at regular, periodic intervals of time
    - Sampling
  - The value at each instant is quantized to one of a number of fixed values
    - Quantization

# Sampling

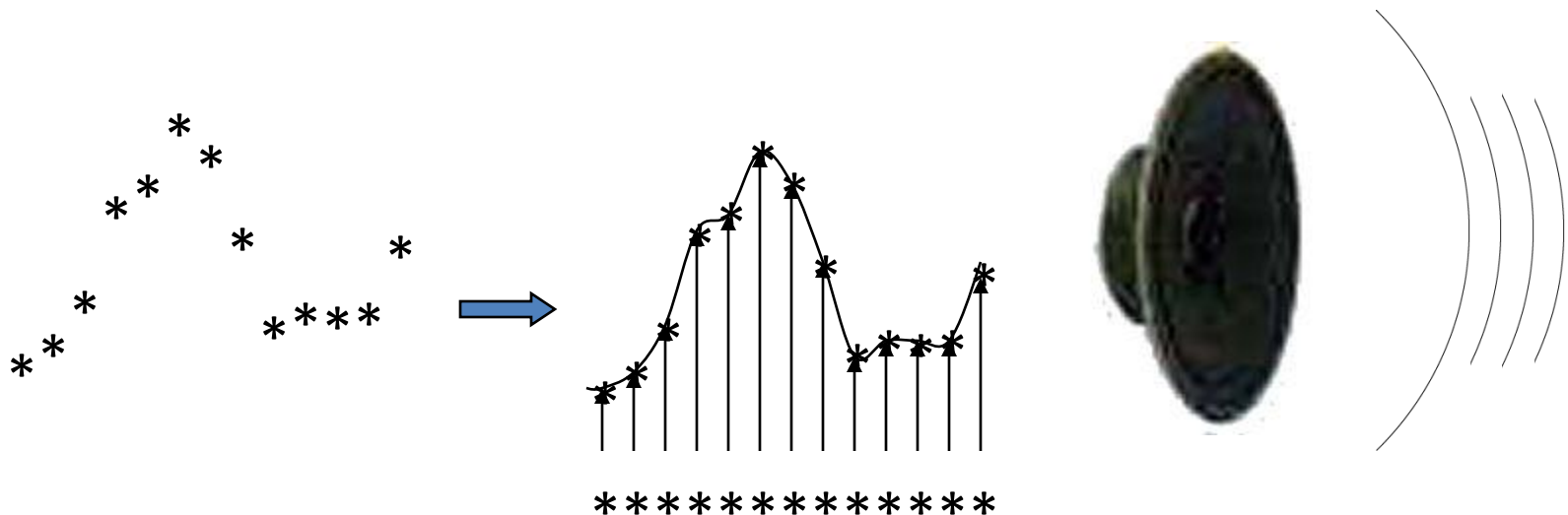
---



- Sampling is the process of capturing snapshots of the signal at discrete instants of time
  - For speech recognition, these samples will be uniformly spaced in time
- Requirement: The sequence of numbers must be sufficient to reconstruct the original signal perfectly
  - To retain all the information in the signal

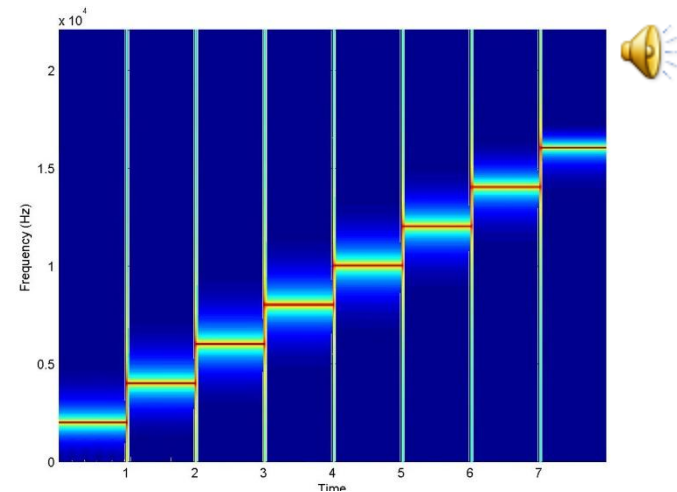
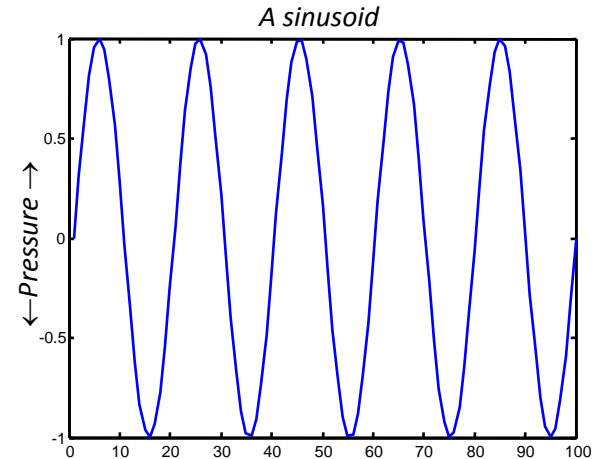
# Sufficiency of Samples

- How to determine if the captured samples are enough?
  - Recreate the sense of sound
- The numbers are used to control the levels of an electrical signal
- The electrical signal moves a diaphragm back and forth to produce a pressure wave
  - That we sense as sound
- The recreated sound must be identical to the original



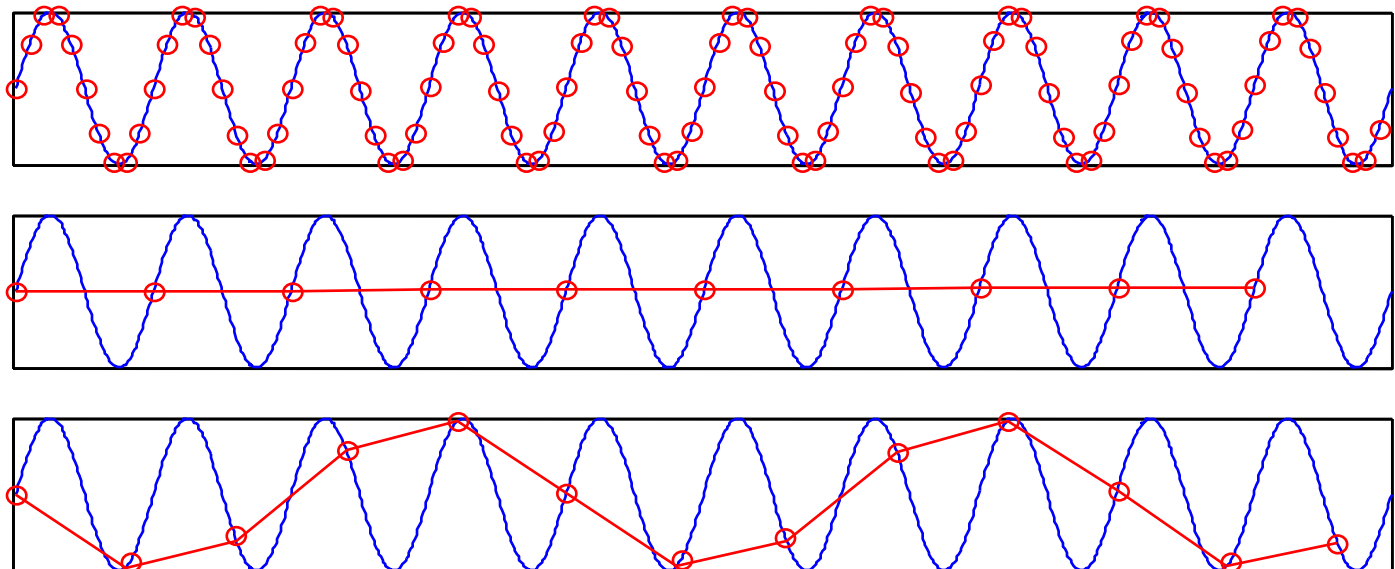
# How many samples a second

- Convenient to think of sound in terms of sinusoids with frequency
- Sounds may be modeled as the sum of many sinusoids of different frequencies
  - Frequency is a physically motivated unit
  - Each hair cell in our inner ear is tuned to specific frequency
- Any sound has many frequency components
  - We can hear frequencies up to 16000Hz
    - Frequency components above 16000Hz can be heard by children and some young adults
    - Nearly nobody can hear over 20000Hz.



# Aliasing

- *Sampling frequency* is the number of samples taken per second
  - Usually measured in hertz (Hz) : 1 Hz = 1 sample per second
- Low sample rates result in *aliasing*
  - High frequencies are misrepresented
  - Frequency  $f_1$  will become (sample rate -  $f_1$ )
- **NYQUIST THEOREM: To represent any frequency  $f$  accurately, you need at least  $2f$  samples per second**

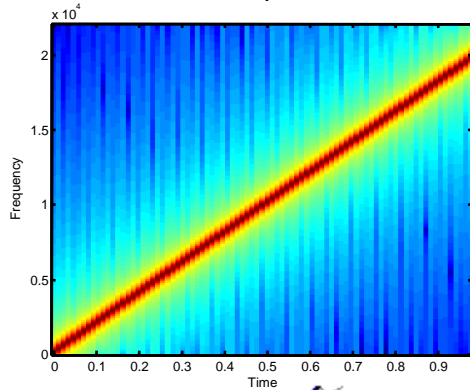




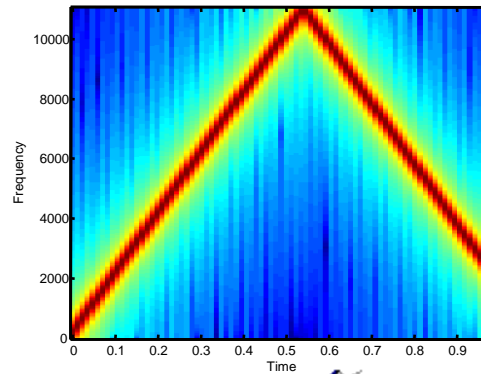
# Aliasing examples

Sinusoid sweeping from 0Hz to 20kHz

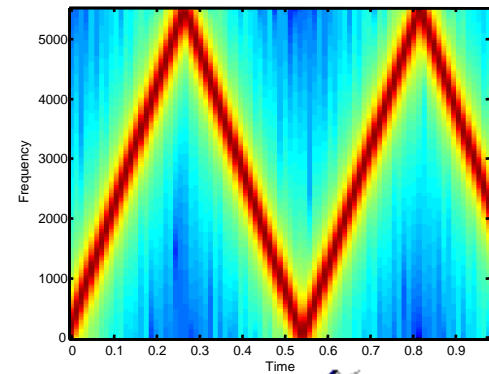
44kHz SR, is ok



22kHz SR, aliasing!



11kHz SR, double aliasing!



On real sounds



at 44kHz



at 11kHz



at 4kHz



at 22kHz



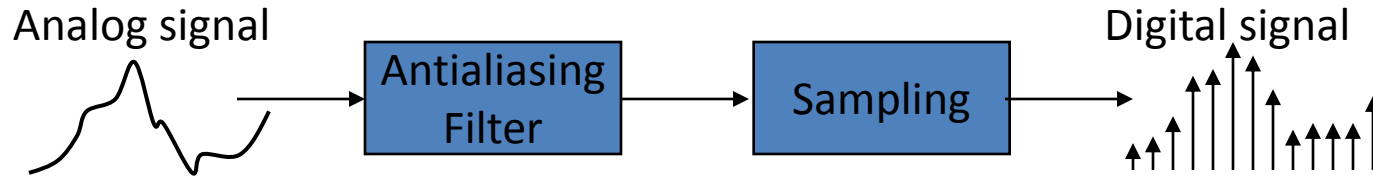
at 5kHz



at 3kHz

# Avoiding Aliasing

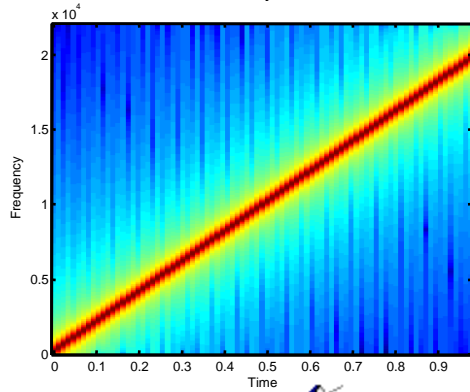
---



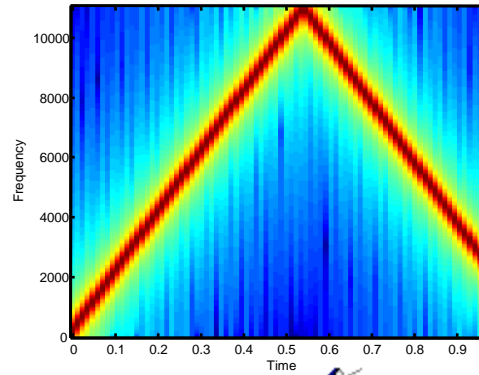
- Sound naturally has all frequencies
  - And then some
  - Cannot control the rate of variation of pressure waves in nature
- Sampling at *any* rate *will* result in aliasing
- Solution: *Filter the electrical signal* before sampling it
  - Cut off all frequencies above  $\text{sampling frequency}/2$
  - E.g., to sample at 44.1Khz, filter the signal to eliminate all frequencies above 22050 Hz

# Aliasing examples: Sweep 0-20kHz

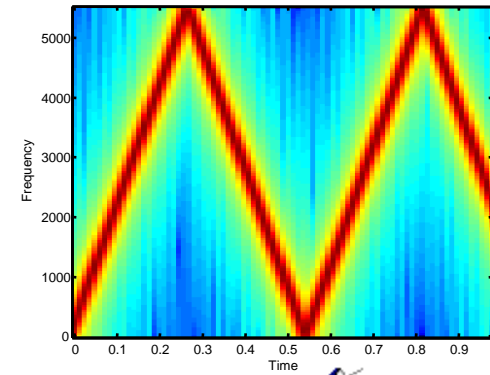
44kHz SR, is ok



22kHz SR, aliasing!

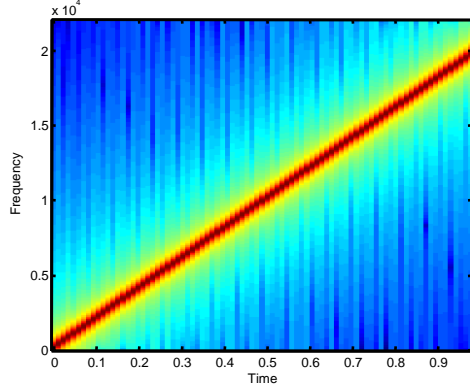


11kHz SR, double aliasing!

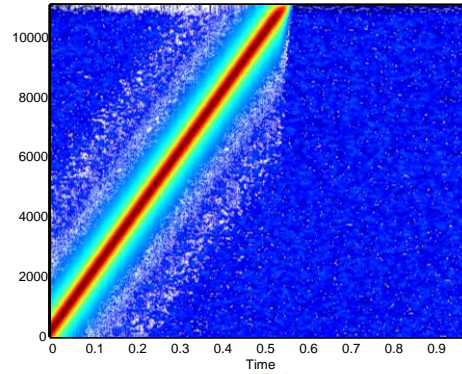


# Anti Aliasing examples: Sweep 0-20kHz

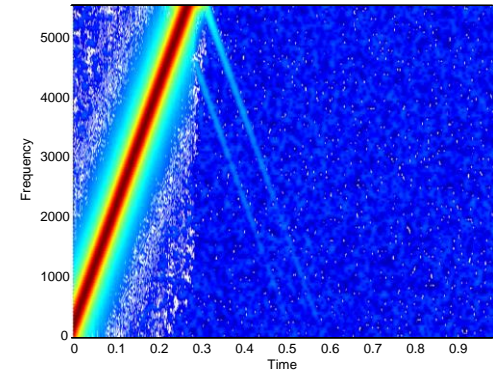
44kHz SR, is ok



22kHz SR, antialiased!

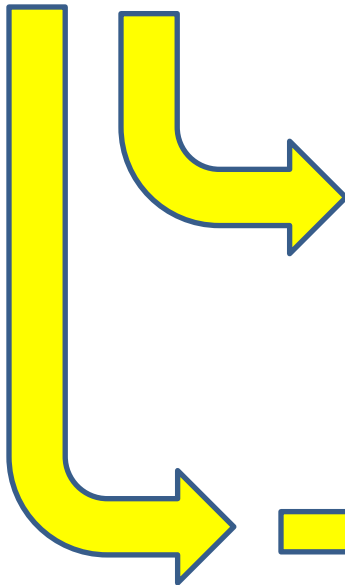
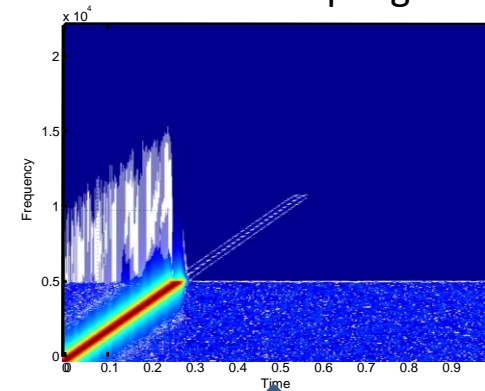
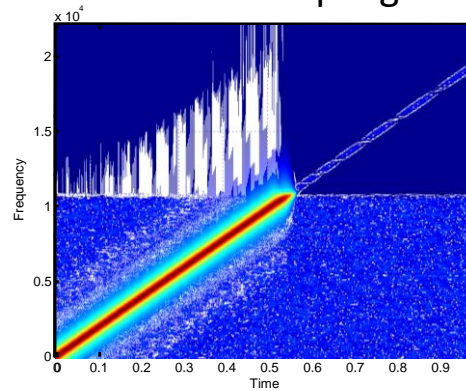


11kHz SR, antialiased!



Prior to 22kHz  
downsampling

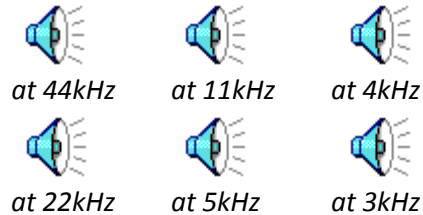
Prior to 11kHz  
downsampling



# Anti Aliasing Examples: Real sounds

---

## Aliased



## Anti aliased



# Anti Aliasing

---

- Speech signal frequencies are audible up to 20 kHz
  - Which means a sampling frequency of 40 kHz should capture all relevant information
- However, the signal has frequencies going up to over 100kHz
  - Sampling at 40 kHz causes frequencies  $> 20\text{kHz}$  to be aliased
  - These frequency components will “fold over” and distort lower frequencies
- The analog signal must first be filtered to eliminate frequencies above 20 kHz to sample at 40 kHz

# Typical Sampling Frequencies

---

- Speech remains intelligible if we filter out high frequencies
  - Telephone speech only has 300Hz – 3.3kHz
- Practical systems use a variety of sampling rates
  - For speech 8kHz, 11.5kHz and 16kHz and 44.1 kHz are all used
  - Sampling hardware will usually employ appropriate anti-aliasing
    - Not always – be careful
- The best speech recognition performance is obtained when the antialiasing filter retains frequencies till at least 8kHz
  - With sampling rates 16kHz or higher
- Lower frequency cutoffs result in poorer performance
  - E.g. recognition with telephone speech is worse than that with wideband speech (going up to 8kHz)
  - *Oversampling* does not increase frequency content
    - *I.e.* there is no benefit from sampling telephone speech at 16kHz
    - We normally sample telephone speech at 8kHz

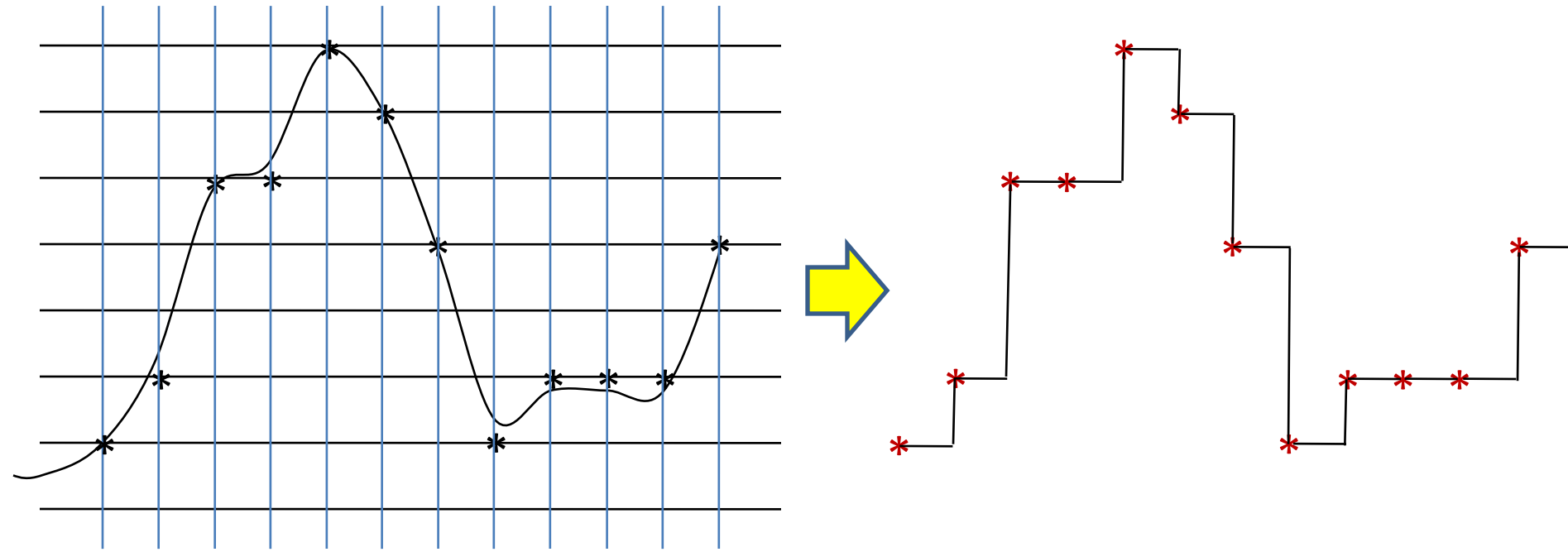
# The Speech Signal: Sampling

---

- Audio hardware typically supports several standard rates
  - *E.g.:* 8, 16, 11.025, or 44.1 KHz ( $n \text{ Hz} = n \text{ samples/sec}$ )
  - CD recording employs 44.1 KHz per channel – high enough to represent most signals most faithfully
- Speech recognition typically uses 8KHz sampling rate for telephone speech and 16KHz for wideband speech
  - Telephone data is *narrowband* and has frequencies only up to 4 KHz
  - Good microphones provide a *wideband* speech signal
    - 16KHz sampling can represent audio frequencies up to 8 KHz
    - This is considered sufficient for speech recognition

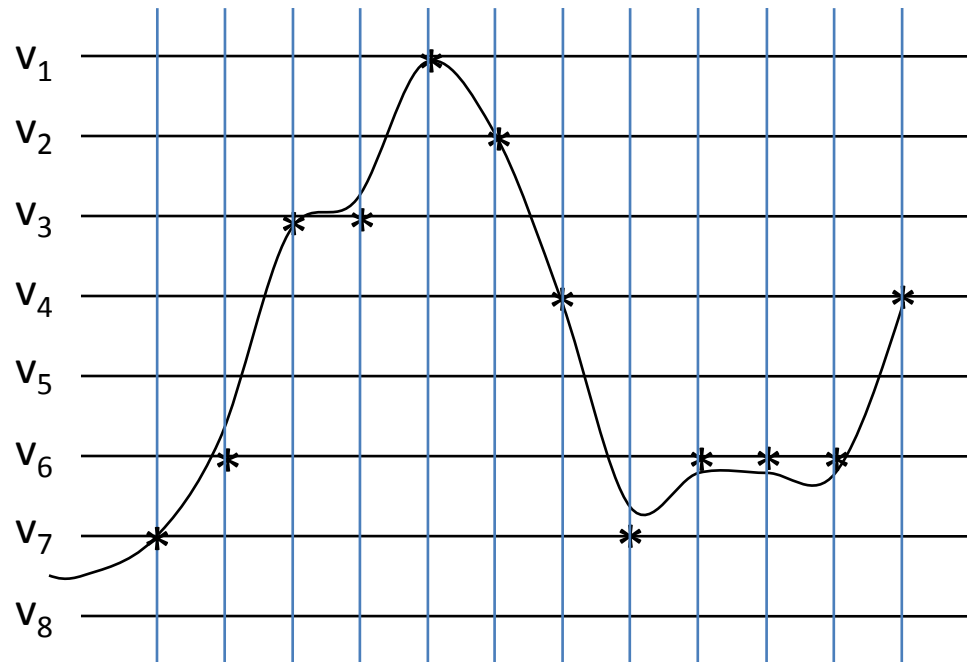


# Sample Resolution



- Samples cannot take any value
- They are restricted to a finite set of values
  - Typically one of  $2^N - 1$ , for  $N=8, 16, 32$
  - Can be represented using integers

# Process of Quantization



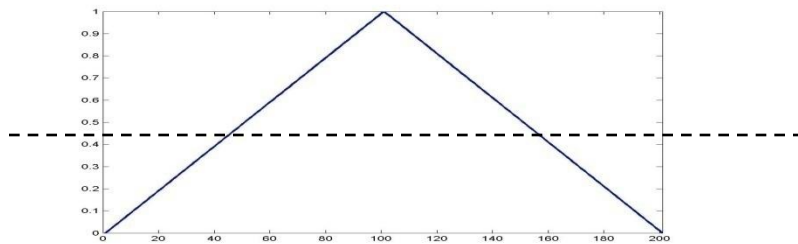
index	Value
0	$V_1$
1	$V_2$
2	$V_3$
3	$V_4$
4	$V_5$
5	$V_6$
6	$V_7$
7	$V_8$

- Determine the values the signal will be rounded to
- Maintain a table of values
- Store only the indices into the table
  - The actual values are read from the table when recomposing speech
    - The table may be implicit

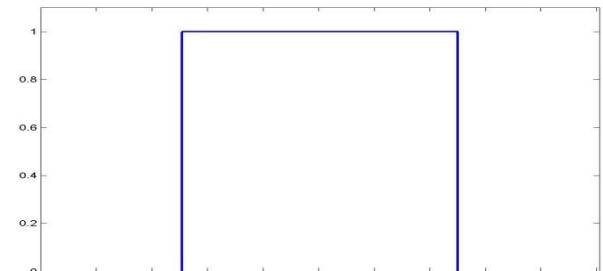
# Mapping signals into bits

- Example of 1-bit *Uniform* sampling table: Only 2 levels

Signal Value	Bit sequence	Mapped to
$S > 2.5v$	1	$1 * \text{const}$
$S \leq 2.5v$	0	0



Original Signal

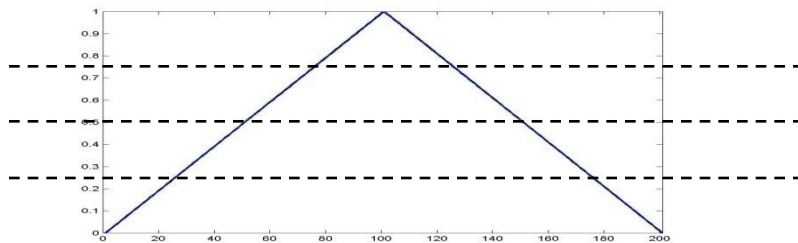


Quantized approximation

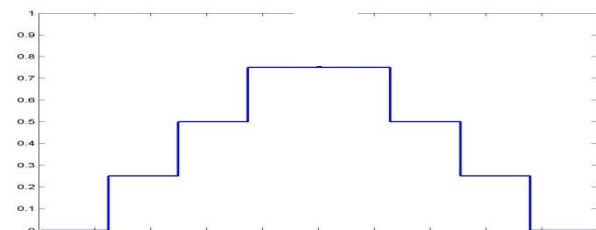
# Mapping signals into bits

- Example of 2-bit *Uniform* sampling table: 4 equally-spaced levels

Signal Value	Bit sequence	Mapped to
$S \geq 3.75\text{v}$	11	3 * const
$3.75\text{v} > S \geq 2.5\text{v}$	10	2 * const
$2.5\text{v} > S \geq 1.25\text{v}$	01	1 * const
$1.25\text{v} > S \geq 0\text{v}$	0	0



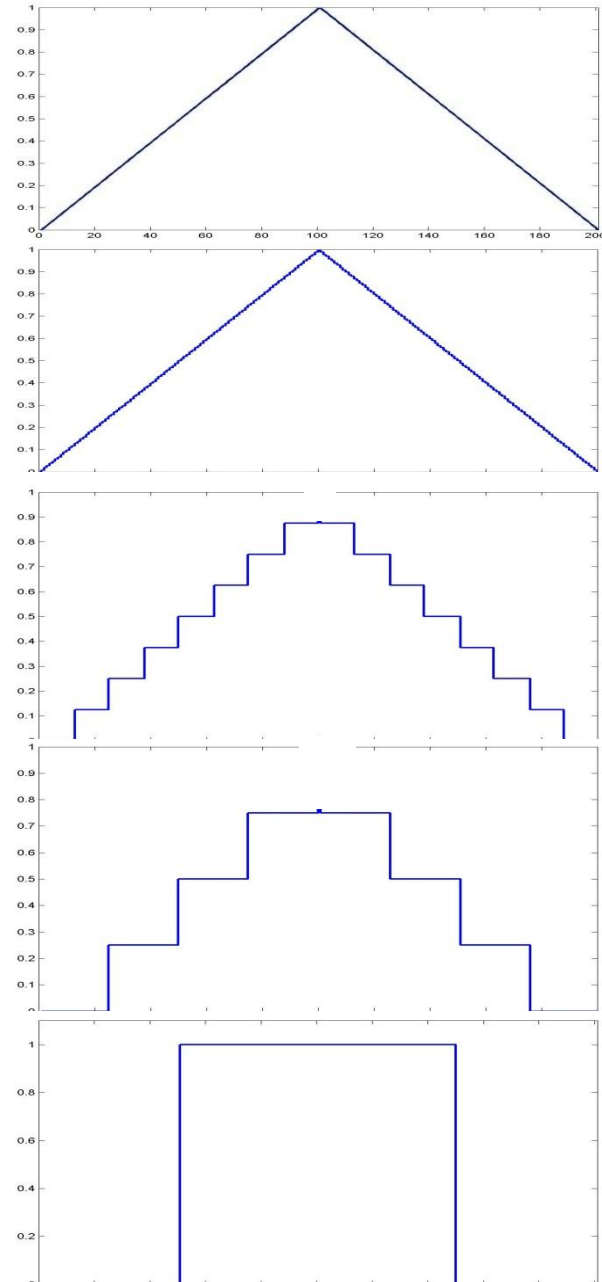
Original Signal



Quantized approximation

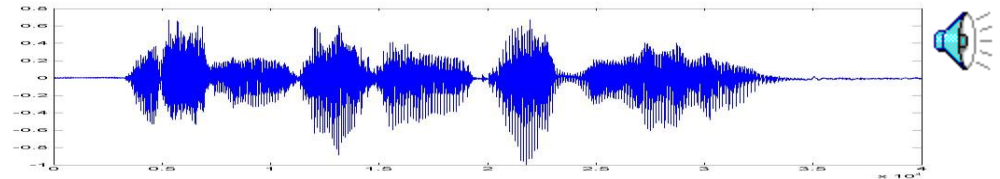
# Uniform Sampling, different levels

- The original signal
- 8 bit quantization
- 3 bit quantization
- 2 bit quantization
- 1 bit quantization

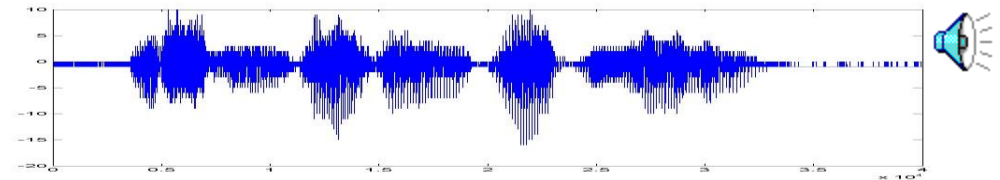


# Tom Sullivan Says his Name

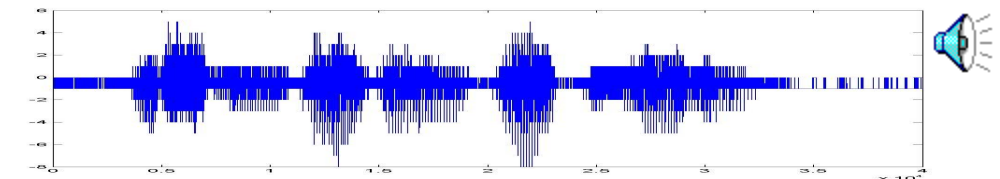
- 16 bit sampling



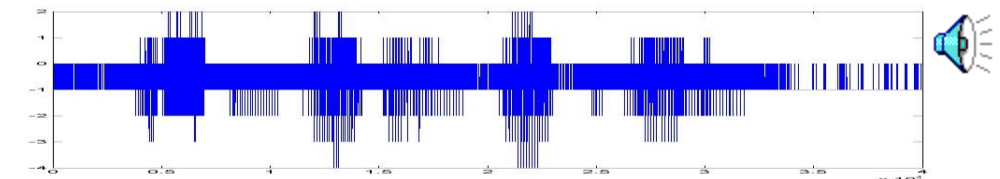
- 5 bit sampling



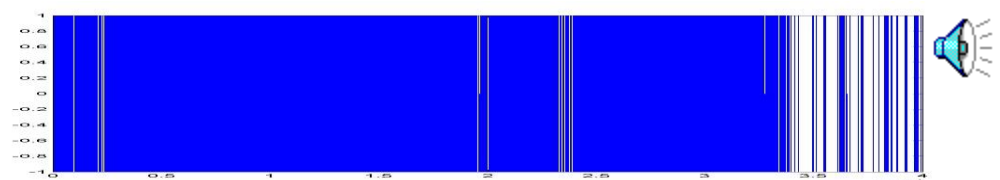
- 4 bit sampling



- 3 bit sampling

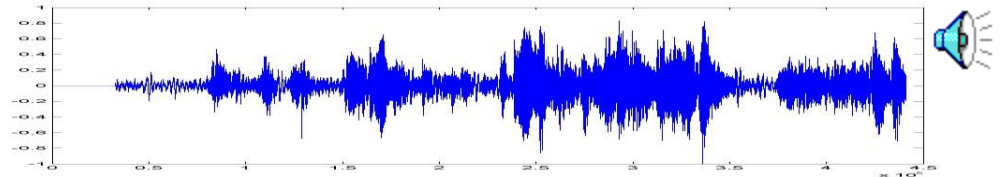


- 1 bit sampling

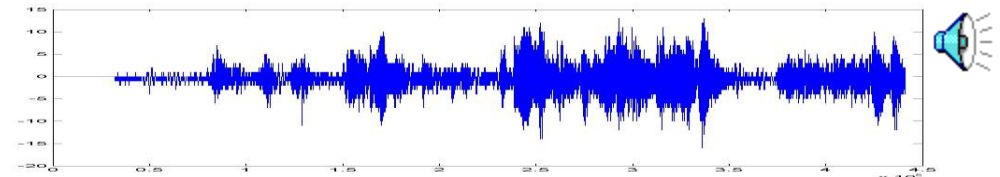


# A Schubert Piece

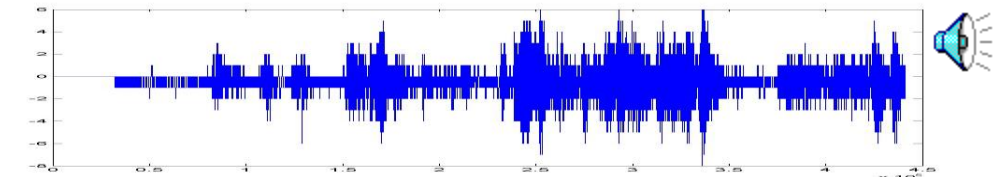
- 16 bit sampling



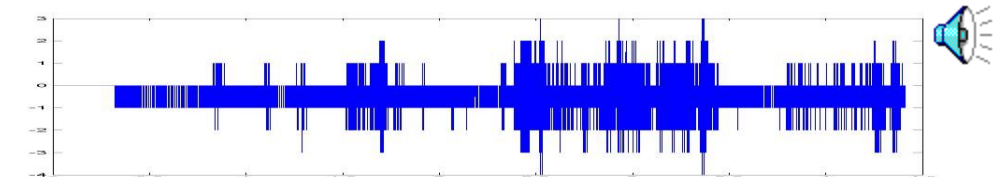
- 5 bit sampling



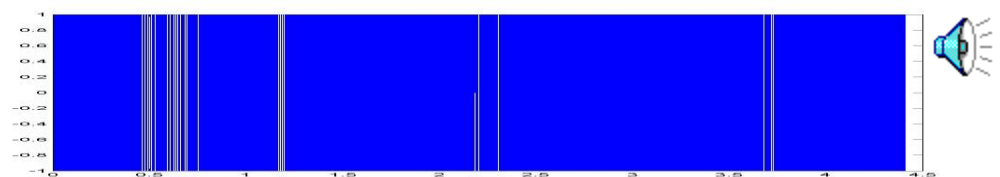
- 4 bit sampling



- 3 bit sampling



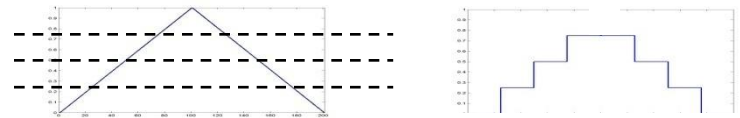
- 1 bit sampling



# Sampling Formats

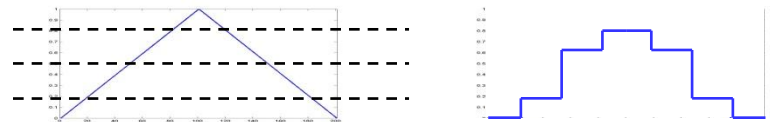
- *Uniform* sampling: Sample values equally spaced out

Signal Value	Bits	Mapped to
$S \geq 3.75v$	11	$3 * \text{const}$
$3.75v > S \geq 2.5v$	10	$2 * \text{const}$
$2.5v > S \geq 1.25v$	01	$1 * \text{const}$
$1.25v > S \geq 0v$	0	0



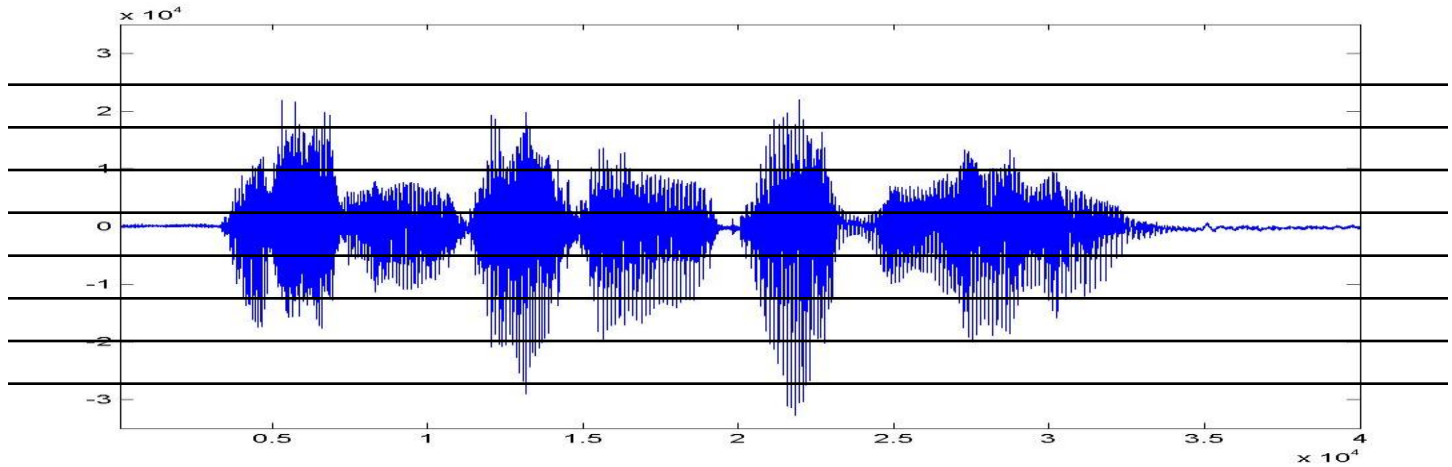
- Sampling need not be uniform: nonuniform sampling

Signal Value	Bits	Mapped to
$S \geq 4v$	11	$4.5 * \text{const}$
$4v > S \geq 2.5v$	10	$3.25 * \text{const}$
$2.5v > S \geq 1v$	01	$1.25 * \text{const}$
$1.0v > S \geq 0v$	0	$0.5 * \text{const}$

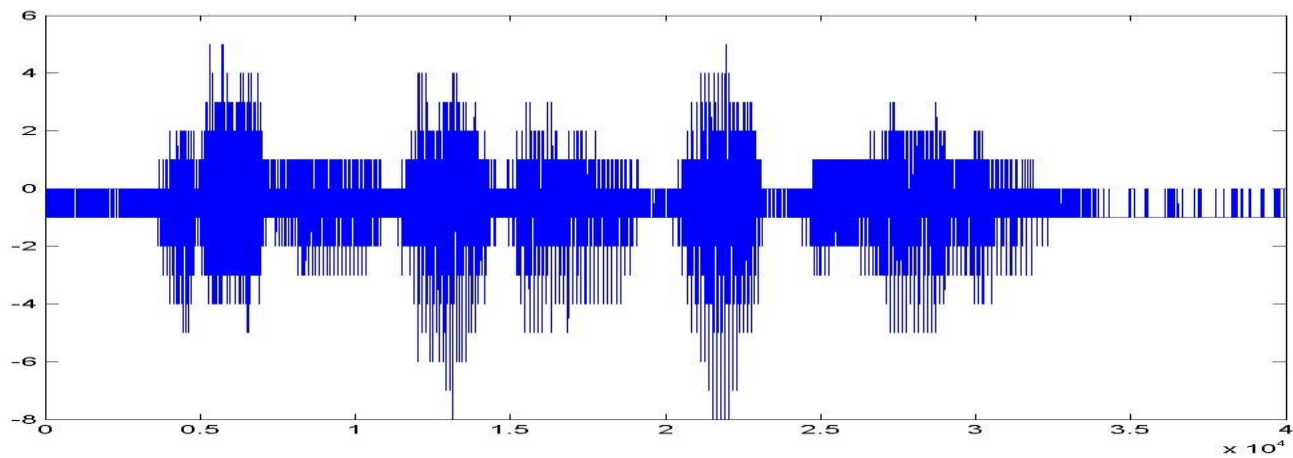




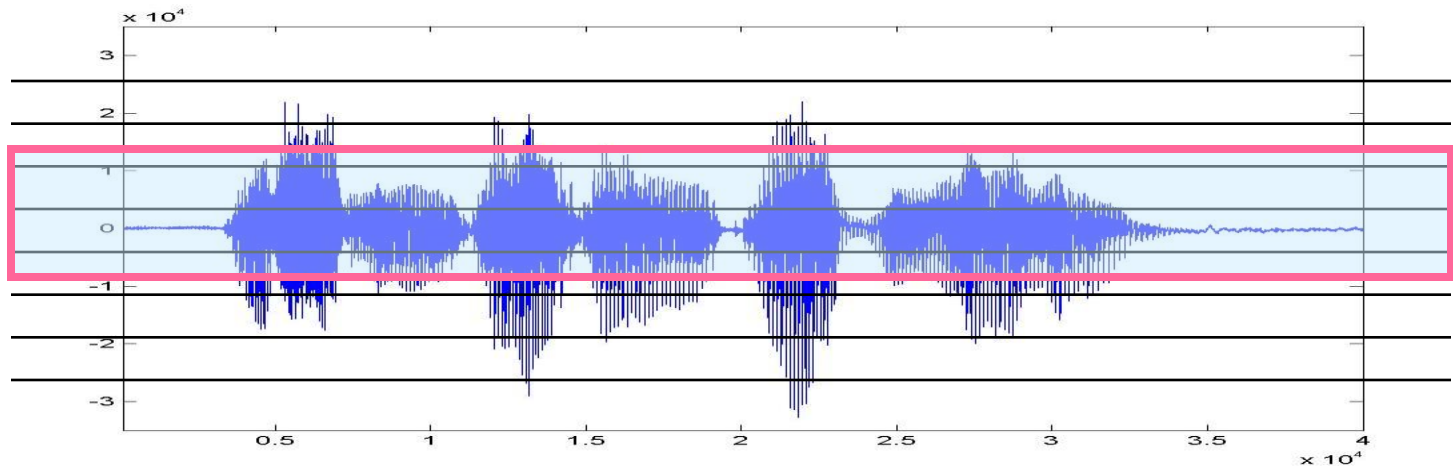
# Uniform Sampling



UPON BEING SAMPLED AT ONLY 3 BITS (8 LEVELS)

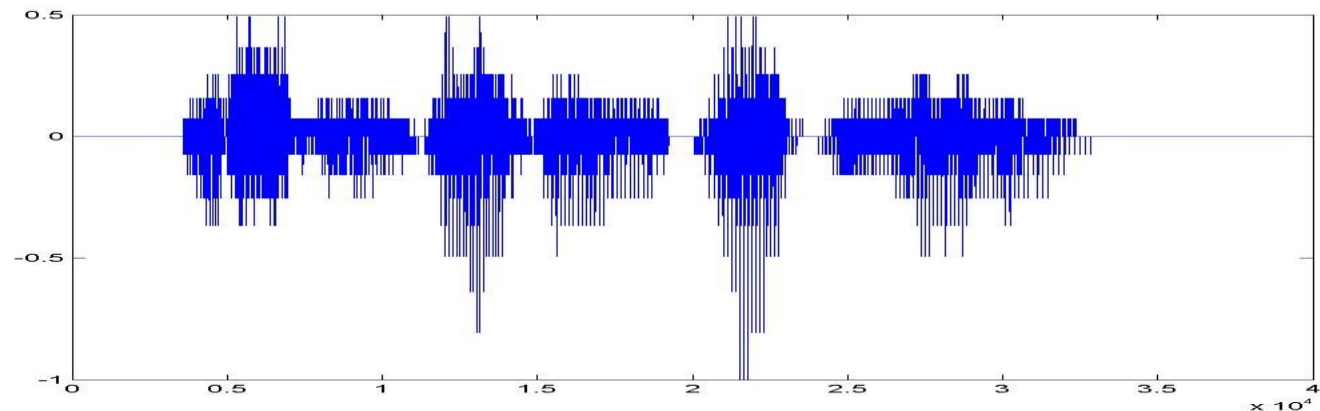
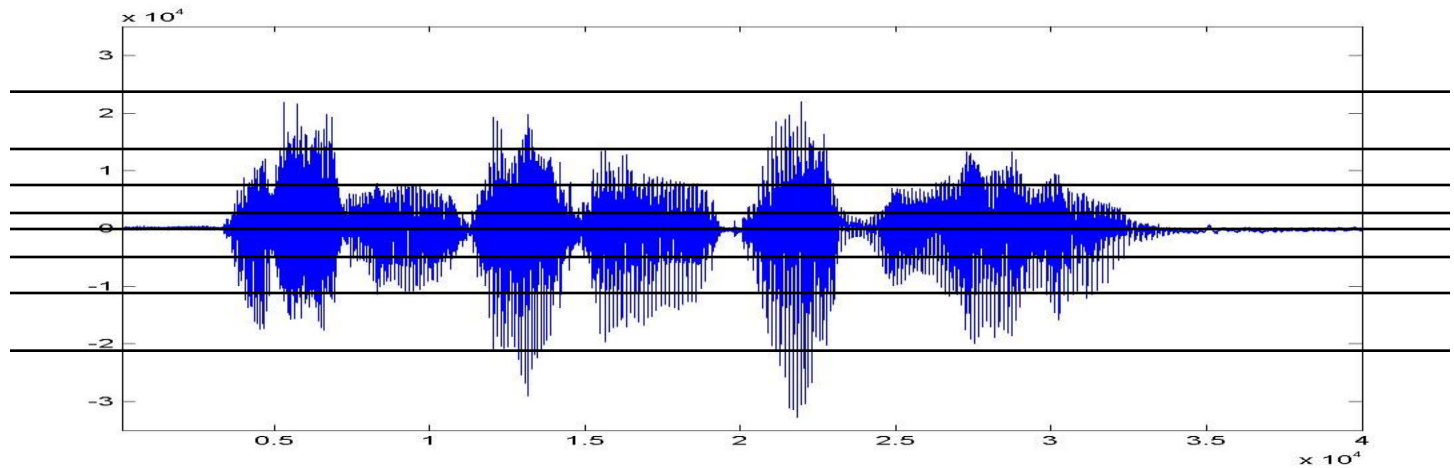


# Uniform Sampling



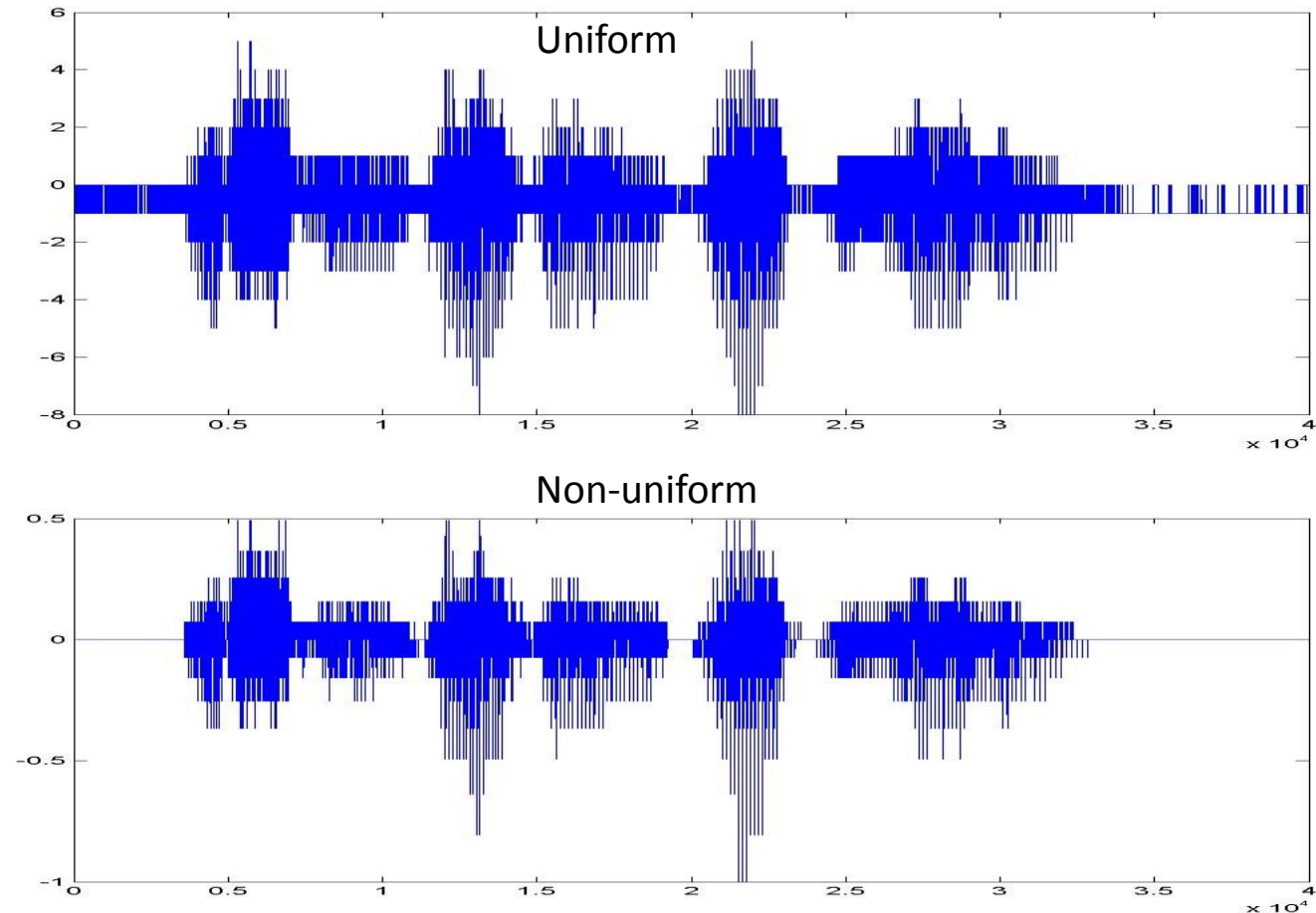
- There is a lot more action in the central region than outside.
- Assigning only four levels to the busy central region and four entire levels to the sparse outer region is inefficient
- Assigning more levels to the central region and less to the outer region can give better fidelity
  - for the same overall number of levels

# Non-uniform Sampling



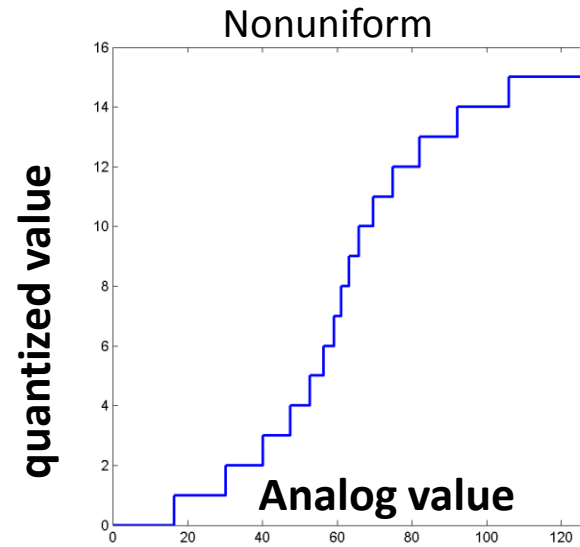
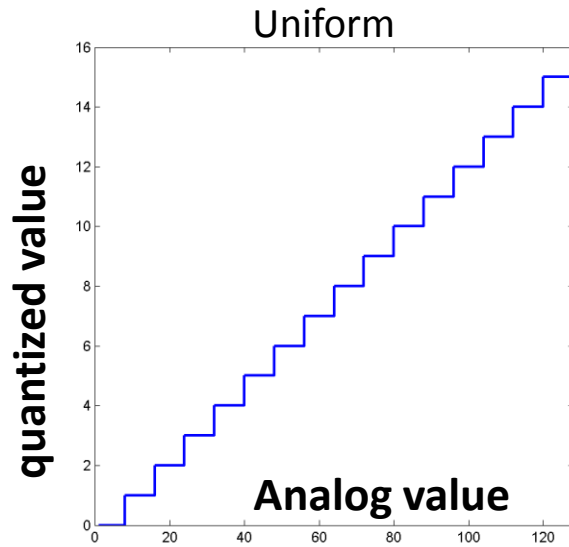
- Assigning more levels to the central region and less to the outer region can give better fidelity for the same storage

# Non-uniform Sampling



- Assigning more levels to the central region and less to the outer region can give better fidelity for the same storage

# Non-uniform Sampling



- Uniform sampling maps uniform widths of the analog signal to units steps of the quantized signal
- In non-uniform sampling the step sizes are smaller near 0 and wider farther away
  - The curve that the steps are drawn on follow a logarithmic law:
    - Mu Law:  $Y = C \cdot \log(1 + \mu X/C)/(1+\mu)$
    - A Law:  $Y = C \cdot (1 + \log(a \cdot X)/C)/(1+a)$
- One can get the same perceptual effect with 8 bits of non-linear sampling as 12 bits of linear sampling

# Capturing / Reading Audio

---

- The numbers returned by an ADC or read from a file are *indices* into the table
- The indices must be converted back to actual values
- For uniformly sampled data the indices are proportional to the value and can be used directly
  - Called “Linear PCM” or “PCM” encoding
- For non-uniform sampling, table lookup (or function inversion) is required
  - Often performed automatically by the audio data read / capture functions in most audio libraries
  - If we write the read / capture functions ourselves, we must perform appropriate table lookup

# Audio capture: Conversion summary

Signal Value	Bits	Mapped to
$S \geq 3.75v$	11	3
$3.75v > S \geq 2.5v$	10	2
$2.5v > S \geq 1.25v$	01	1
$1.25v > S \geq 0v$	0	0

Signal Value	Bits	Mapped to
$S \geq 4v$	11	4.5
$4v > S \geq 2.5v$	10	3.25
$2.5v > S \geq 1v$	01	1.25
$1.0v > S \geq 0v$	0	0.5

- Capture / read audio in the format provided by the file or hardware
  - Linear PCM, Mu-law, A-law, Coded
- Convert to 16-bit PCM value
  - I.e. map the bits onto the number on the right column
  - This mapping is typically provided by a table computed from the sample compression function
  - No lookup for data stored in PCM
- Conversion from Mu law:
  - <http://www.speech.cs.cmu.edu/comp.speech/Section2/Q2.7.html>

# The Effect of Signal Quality

---

- The quality of the digitized signal depends critically on many factors:
  - The electronics performing sampling and digitization
    - Poor quality electronics can severely degrade signal quality
      - *E.g.* Disk or memory bus activity can inject noise into the analog circuitry
  - Improper Antialiasing
    - Not using an antialiasing filter is a cause for many problems
  - Insufficient quantization levels
    - Minimally 16 bit PCM or 8 bit Mu / A law is needed
  - Proper setting of the recording level
    - Too low a level underutilizes available signal range, increasing susceptibility to noise
    - Too high a level can cause *clipping*
  - The microphone quality
  - Ambient noise in recording environment
- Suboptimal signal quality can affect recognition accuracy to the point of being completely useless

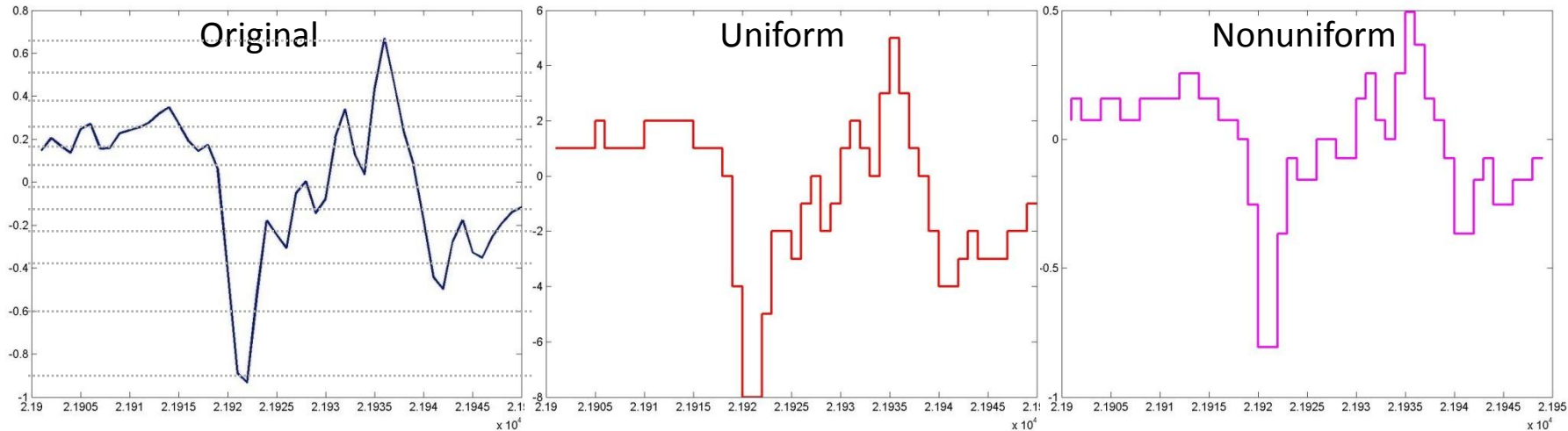


# The Effect of Signal Quality

---

- The quality of the digitized signal depends critically on many factors:
  - The electronics performing sampling and digitization
    - Poor quality electronics can severely degrade signal quality
      - *E.g.* Disk or memory bus activity can inject noise into the analog circuitry
  - Improper Antialiasing
    - Not using an antialiasing filter is a cause for many problems
  - Proper setting of the recording level
    - Too low a level underutilizes available signal range, increasing susceptibility to noise
    - Too high a level can cause *clipping*
  - The microphone quality
  - Ambient noise in recording environment
- Suboptimal signal quality can affect recognition accuracy to the point of being completely useless

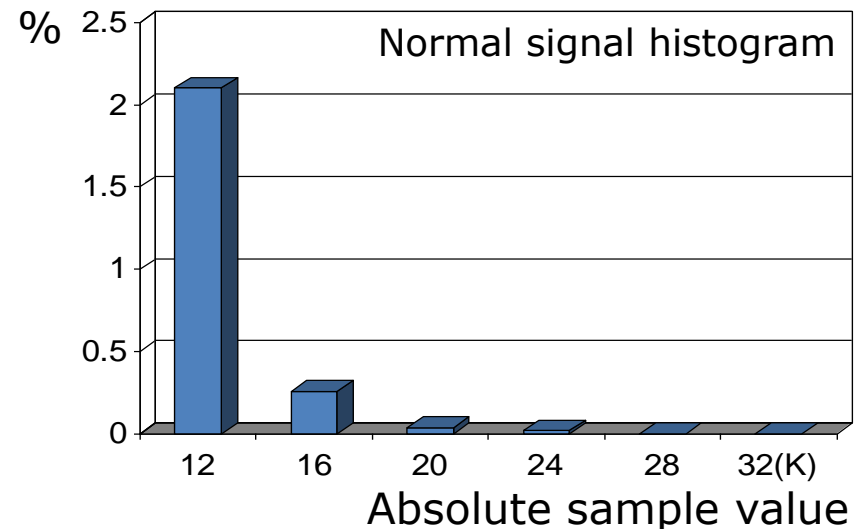
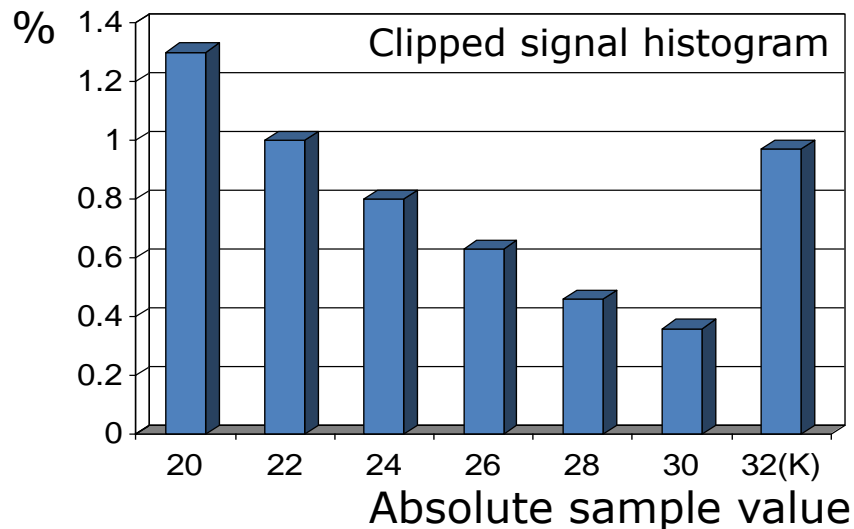
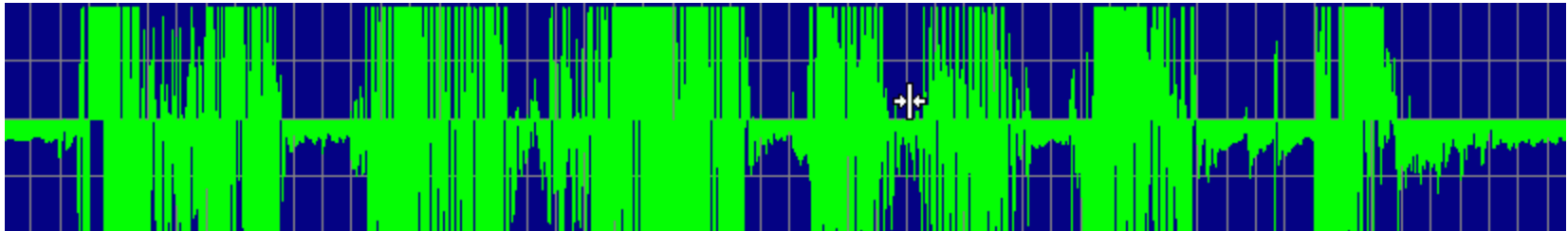
# Uniform and Non-uniform Sampling



- At the sampling instant, the actual value of the waveform is rounded off to the nearest level permitted by the quantization
- Values entirely outside the range are quantized to either the highest or lowest values

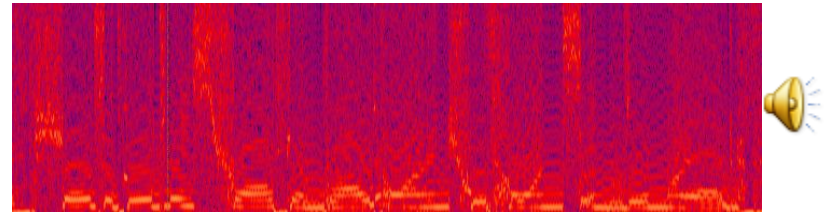
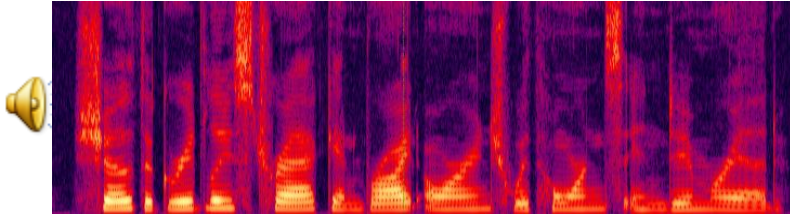
# Clipping in Speech Signals

- Clipping and non-linear distortion are the most common and most easily fixed problems in audio recording
  - Simply reduce the signal gain (but AGC is not good)



# The Effect of Noise

---



- Non-speech signals in the background can corrupt the recording
  - Diffuse background noise, e.g. in an automobile
  - Localized non-speech sounds, e.g. air conditioner
  - Background talkers, music..
- These result in degraded recognition accuracy
- Solution
  - Denoising algorithms for cancelling noise
  - Or, much (much) better – use directional microphones that will not pick up the noise

# The Effect of Recording Channel

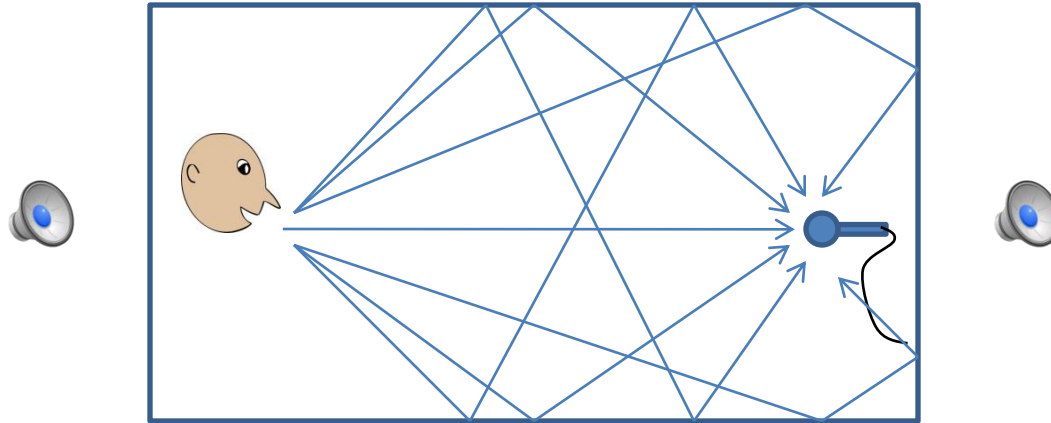
---



- Recording channels/media can result in signal distortions
  - Telephone channels reduce the bandwidth of the signal
    - 300-3300Hz
    - Loss of information results in poorer recognition accuracy
  - Cellphone / VOIP channels introduce *coding* distortion
    - Coding and decoding speech introduces distortions
  - They can also result in *discontinuities* from dropped packets
  - Poor microphones can result in spectral distortions
  - Computer recordings may be affected by memory and disk activity
- Distortions cannot usually be recovered from
  - Recognizer must learn to perform well even on distorted data

# Reverberation

---



- Reflective walls etc. can introduce *reverberation* into a recording
  - Due to repeated addition of a signal with delayed reflections of itself
- Reverberation can be viewed as the combination of a linear-channel distortion and additive noise
  - Linear channel – represents effect of immediate past on current signal
  - Additive noise – represents effect of extended past on current signal
- Small levels of reverberation is usually beneficial to human perception, but even small levels of reverberation degrade automatic recognition
- Best option: Select recording environments with minimal reverberation

# Reading/Capturing the signal

---

- Audio can be obtained by:
  - Read from files
    - for offline/batch processing
  - Captured directly from hardware
    - for online processing
- **Reading from a prerecorded file:** Various library routines/calls
  - Matlab: `[s,fs,c,...] = wavread('filename', args)`
  - Libraries in C, Java, python, fortran..
- **Offline capture into a file:** You can use tools available for your favorite OS
  - Windows provides a “Windows recorder”
  - Several audio capture tools are also available for windows
  - Linux and most Unix machines provide “arecord” and “aplay”
    - If these are not already on your machine, you can download them from the web
  - Other tools are also available for linux

# Storing Audio/Speech Files

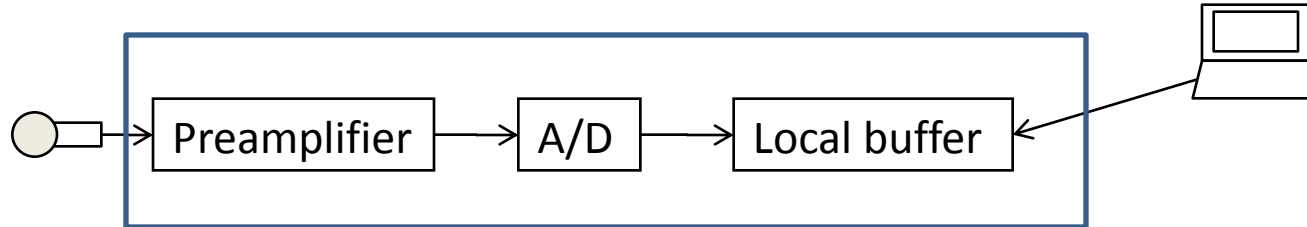
---

- There are many storage formats in use. Important ones:
  - PCM raw data (\*.raw)
  - NIST (\*.sph)
  - Microsoft PCM (\*.wav)
  - Microsoft ADPCM (\*.wav)
  - SUN (\*.au, \*.snd) etc.
- The data are typically written in binary, but many of these formats have headers that can be read as ascii text.
  - Headers store critical information such as byte order, no. of samples, coding type, bits per sample, sampling rate etc.
- Speech files must be converted from stored format to linear PCM format for further processing
  - Audio I/O library routines will usually process the headers to obtain the necessary information and perform the appropriate conversion
  - If we write the I/O, we must do this ourselves



# *Capturing* speech signals

---



- Your computer must have a sound card or an external A/D converter, and audio input devices such as a microphone, line input etc.
- Capture
  - Signal is captured by a microphone
  - Preamplified
  - Digitized
  - Stored in a buffer on the sound card
- Processor
  - Reads from buffer
  - At some prespecified frequency
    - Too frequent: can use up all available CPU cycles
    - Too infrequent: High latency

# Capturing Audio

---

- Capturing audio from your audio device
  - Open the audio device
    - Syntax is OS dependent
  - Set audio device parameters
    - Sampling rate, quantization type/level, no. of channels.
  - Record blocks of audio
  - Close audio device
- Recorded audio can be stored in a file or used for live decoding
- Two modes of audio capture for live-mode decoding
  - **Blocking**: Application/decoder requests audio from audio device when required
    - The program waits for the capture to be complete, after a request
  - **Callback**: An audio program monitors the audio device and captures data. When it has sufficient data it calls the application or decoder

# Capturing speech signals

---

- Example linux pseudocode for capturing audio on an HP IPaq (for single-channel 16khz 16bit PCM sampling):

```
fd = open("/dev/dsp", O_RDONLY);
ioctl(fd, SOUND_PCM_WRITE_BITS, 16);
ioctl(fd, SOUND_PCM_WRITE_CHANNELS, 1);
ioctl(fd, SOUND_PCM_WRITE_RATE, 16000);
while (1) {
    read(fd, buffer, Nsamples*sizeof(short));
    process(buffer);
}
close(fd);
```

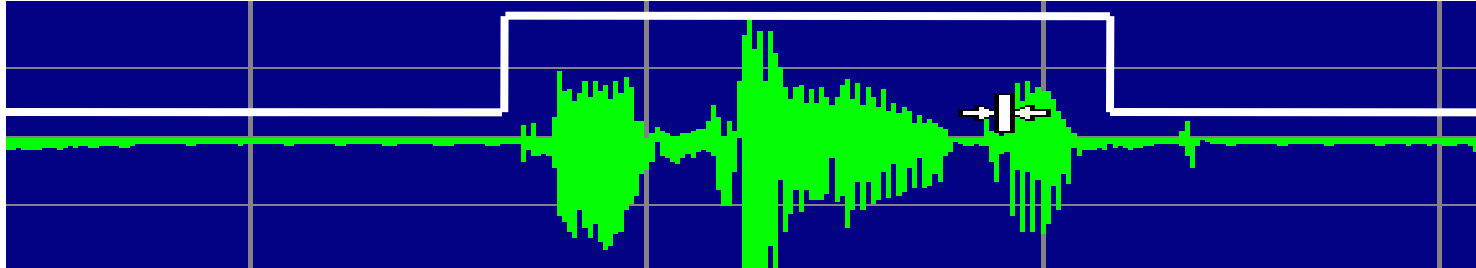
# Speech Capture Models

---

- *Push* Model: The application captures the speech signal
  - And “pushes” the audio into the speech recognition system for recognition
    - And requests a full or partial result when required
  - The speech recognizer does not need to know about speech capture
- *Pull* Model: The application requests the speech recognition for recognition output
  - The recognizer is responsible for capturing the audio
- The former provides the application more flexibility, but also imposes greater responsibility

# Endpointing

---



- A very key component of automatic speech recognition is to know the end points of an utterance
  - *I.e.* Endpointing
- Must avoid attempting to recognize speech in non-speech regions
  - Unnecessary computation load
  - May spuriously recognize words where there are none
- Accurate endpointing is very important in both online and offline (batch) recognition scenarios

# Online Endpointing Formats

---

- Push to talk:
  - Manually identify speech regions
    - E.g. hitting a button at the beginning and end of an utterance
    - E.g. maintaining a button pushed while speaking
- Hit to talk:
  - Manually identify the *beginning* of an utterance
    - E.g. hit a button/key and start speaking
  - System automatically determines when the utterance ends
- Continuous listening:
  - The system “listens” to incoming audio continuously
  - It automatically determines which regions of captured audio are speech and must be recognized

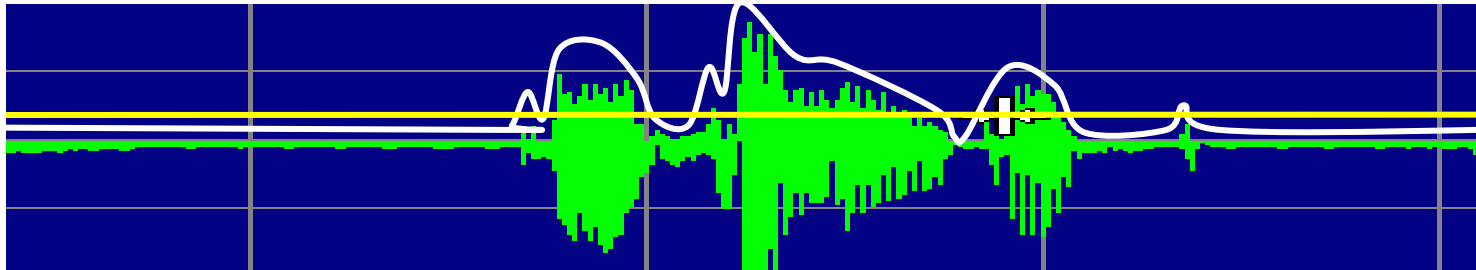
# Batch Endpointing

---

- Similar to the mechanism used for continuous listening, with a couple of options
  - Perform endpointing based on acoustic properties of the signal
    - The same mechanism used by continuous listening systems
  - Use the speech recognizer itself to determine where speech is present
    - *Multiple passes* of processing are allowed
  - Combination of the above
    - A first pass of conservative acoustic-based endpointing
    - Subsequent refinement by recognition

# A Simple Endpointing Scheme

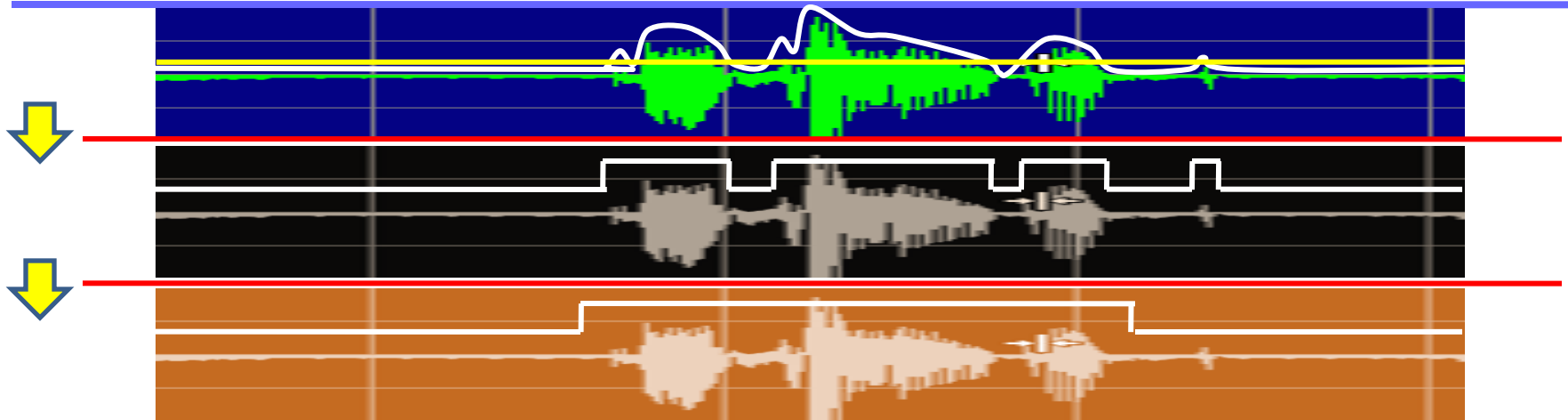
---



- Energy in speech regions is typically higher than the energy in non-speech regions
- Simple algorithm: apply a threshold on the current energy level of the signal
  - If energy exceeds the threshold, it is probably speech
- *Energy-based endpointing*



# Energy Based Endpointing



- Compute the energy at each time  $t$ :
  - Energy at time= $t$  is computed as the sum of squared sample values in a short “frame” of samples around  $t$
  - Typically expressed in decibels (dB):  $10 \log(\sum x_i^2)$ , where  $x_i$  are the sample values in the frame
- Compare the energy to a pre-defined threshold
  - If the energy is above the threshold, we label the signal at time  $t$  as speech
- Smooth the labels:
  - Minimum silence and speech segment length limits may be imposed
  - Segments of silence that are shorter than (e.g.) 100 ms are relabeled as speech, provided the resulting speech segment is longer than (e.g.) 250 ms
  - Extend the boundaries of the speech segment by 250ms at the beginning and end
- This scheme works reasonably well under quiet background conditions

# An Adaptive Endpointing Algorithm

---

```
Function classifyFrame(audioframe):
    current = EnergyPerSampleInDecibel(audioframe)
    isSpeech = False
    level = ((level * forgetfactor) + current) / (forgetfactor+ 1)
    if (current < background):
        background = current
    else:
        background += (current - background) * adjustment
    if (level < background): level = background
    if (level - background > threshold): isSpeech = True
    return isSpeech
```

- “forgetfactor” = forgetting factor; typical value  $\geq 1$
- “level” = smoothed signal level; initial value typically set to energy of first frame
- “threshold” = upper (onset) threshold
- “background” = background signal level; initially set to avg energy of first 10 frames
- “adjustment” = adaptive update factor for background; higher values assume faster varying background. Typical value 0.05

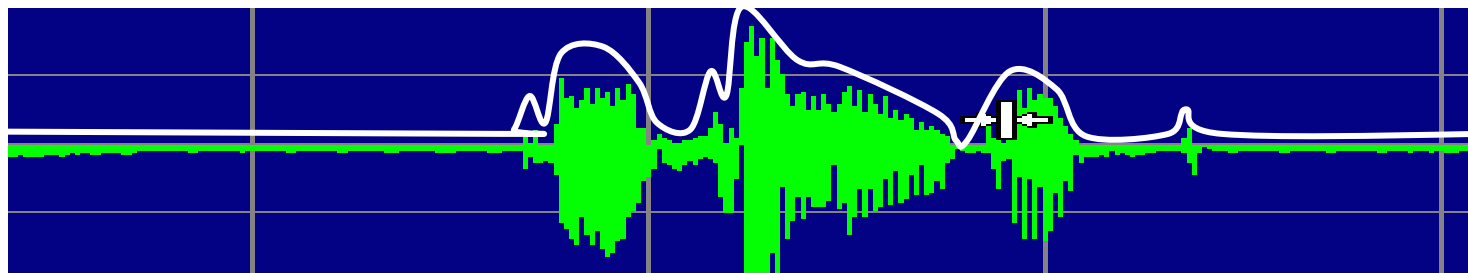
# Adaptive Endpointing

---

- The adaptive endpointing
  - Compares a smoothed signal level to a fast-varying estimate of background
  - *Adaptation* achieved by adapting background level
- Smoothing of speech/non-speech labels is still required

# A More Complex Algorithm

---



- The *onset* of speech is indicated by a sudden large increase in speech energy
  - As compared to the “background” energy level
  - This is apparent even in high noise
- The *termination* of speech is indicated a smaller fall in energy level
  - Which may not be apparent in noise
- The more complete endpointing algorithm uses *two* thresholds to represent these phenomena
  - The thresholds are also made adaptive to account for changing noise conditions

# A Two-Threshold Formula

---

OnsetThreshold = X

OffsetThreshold = X – dynamic\_range

If (inspeech):

    if (energy-background < OffsetThreshold): inspeech = 0

Else:

    if (energy-background > OnsetThreshold): inspeech = 1

If (inspeech):

    background += (energy-background)\*trackingfactor

Else:

    background = alpha\*background + (1-alpha)\*energy

- As before, the speech energy may also be smoothed
- The actual label sequence obtained must be smoothed to prevent unnecessary chopping up of audio
  - But oversmoothing can cause a lot of non-speech segments to be labelled as speech

# End pointing: Hit to talk

---

- In Hit-to-Talk the *start* of speech is indicated
  - However actual speech may begin shortly after the button is clicked
  - You may also record the button click
    - Undesired signal
- You only need to determine the end of speech
- Some adjustments are needed in the algorithm to determine background level from the early segment of the recording
  - Alternately, we may listen continuously in order to update background estimate, but only indicate speech when the button is hit

# Assignment

---

- Write a program for data capture
- Must include:
  - Endpointing with Hit-to-talk
  - Data capture
    - 16kHz, 16bit PCM data
  - Endpointed segment must be written to file
- You can use portaudio for the audio capture