LING 490 – SPECIAL TOPICS IN LINGUISTICS

Fundamentals of Digital Signal Processing

Yan Tang

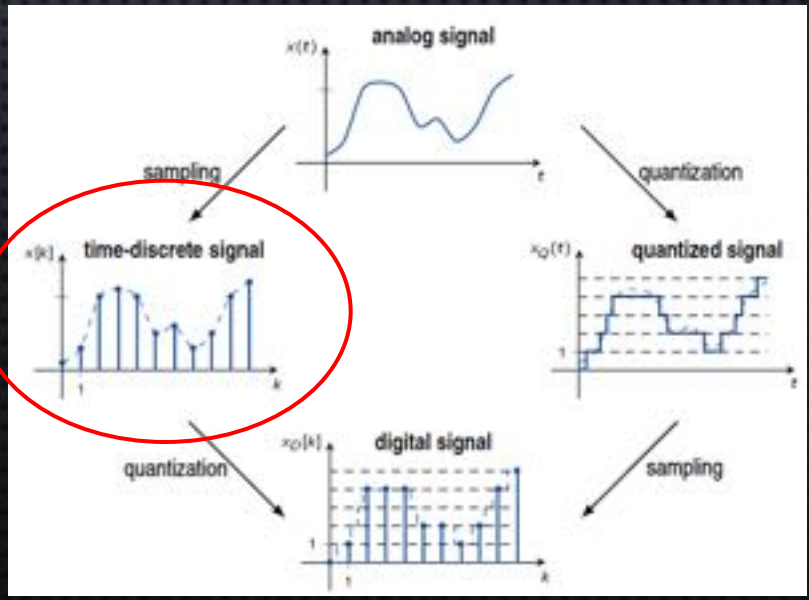
Department of Linguistics, UIUC

Week 4

Last week…

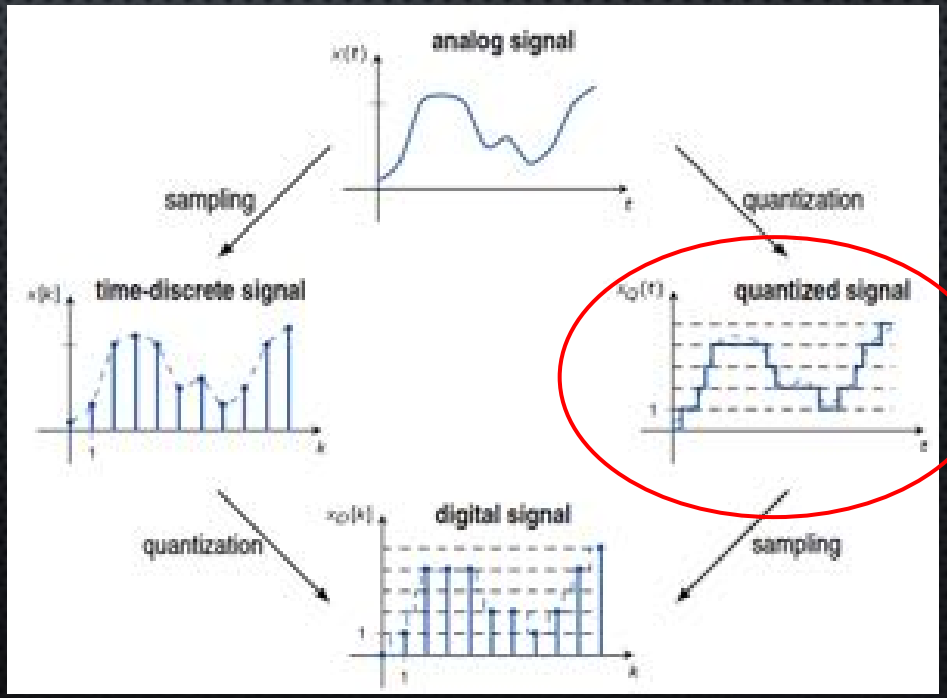
- Digitalisation: Sampling and quantisation
- Sampling
  - Breaking up continuous signal in time
  - Nyquist frequency and sampling theorem
  - Impact of under-sampling: aliasing

$f_s > 2f_{max}$



Last week…

- Quantisation
  - Decimate continuous signal in level
  - Quantisation level = 2^N, N is number of bits
  - Quantisation errors
  - Clipping



Archiving digital signals – digital audio files

- File format for storing digital audio data on a computer
- Common audio formats/types:
  - **.mp3**: MPEG-1 (Motion Picture Experts Group 1) Audio Layer 3
  - **.wma:** Windows Media Audio format
  - **.ogg:** Open-source patent-free compressed audio format
  - **.wav**: Waveform Audio File Format
  - **.aiff**: Audio Interchange File Format
  - …

Audio format and codec

- *Format* !=, ~=, !, ne, not *codec*
- Format:
  - The container that holds audio data
- Codec:
  - To **co**mpress and **dec**ompress (**co**de and **dec**ode)
  - A computer program or algorithm which encodes or decodes digital data
  - E.g. MPEG Layer-3, PCM, GSM6.10, FLAC and AAC

Uncompressed format

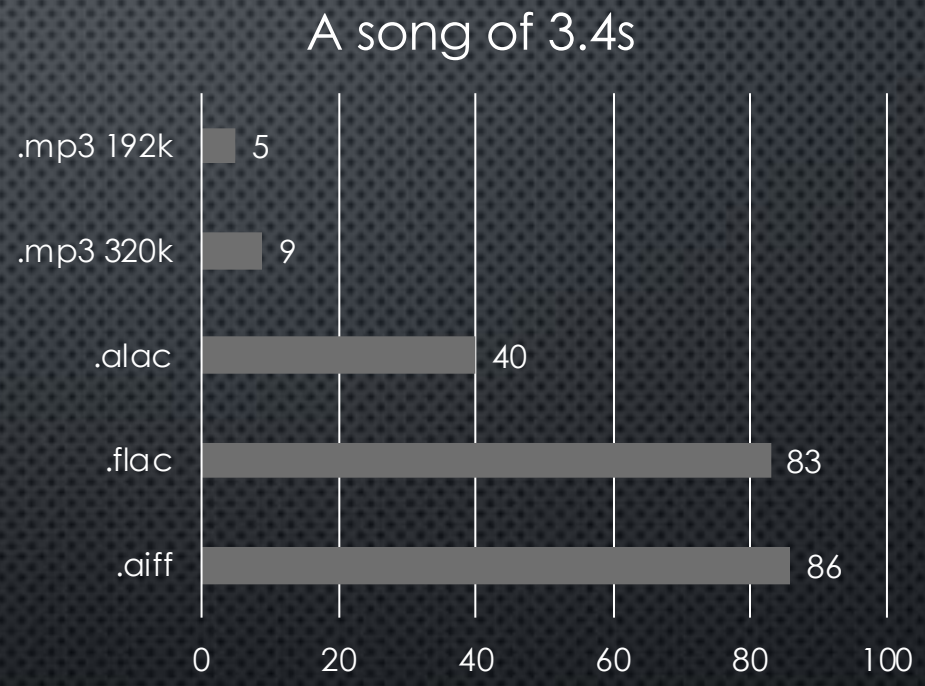
- *Lossless* format
  - Stores original recorded data for high quality playback without being compressed by codec
- Requires very little computational processing
- Provides the highest audio fidelity at a cost of storage space
- .wav and .aiff

Lossless compressed format

- Why compress data?
  - To reduce file size
- *Lossless* compression
  - Preserves entire audio information, with reduced file size by storing the data more efficiently
  - The original data can be restored!
  - Reduces processing time while maintain good compress ratio
  - E.g. .flac and .alac with a compression rate of approx. 2:1

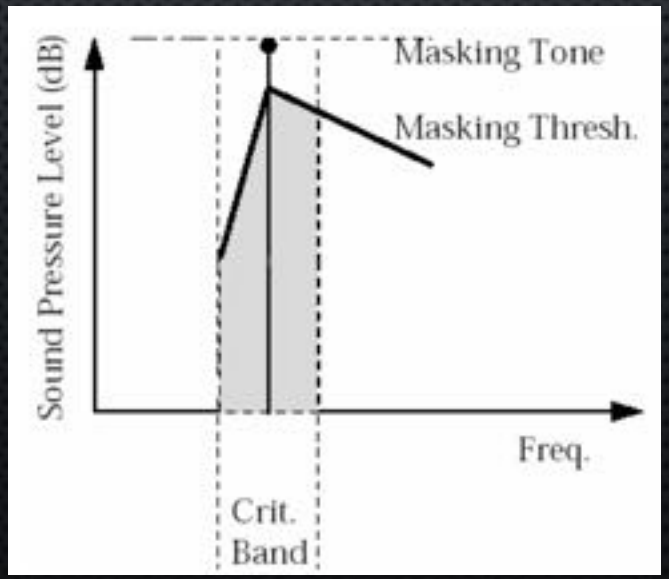
Lossy compressed format

- *Lossy* compression
  - Further reductions in file space, at a cost of some audio information and quality
  - The original data can not be restored!
  - E.g. .mp3 and .m4a. A compression rate of 10:1 can be achieved with little loss of ***perceptual*** quality.



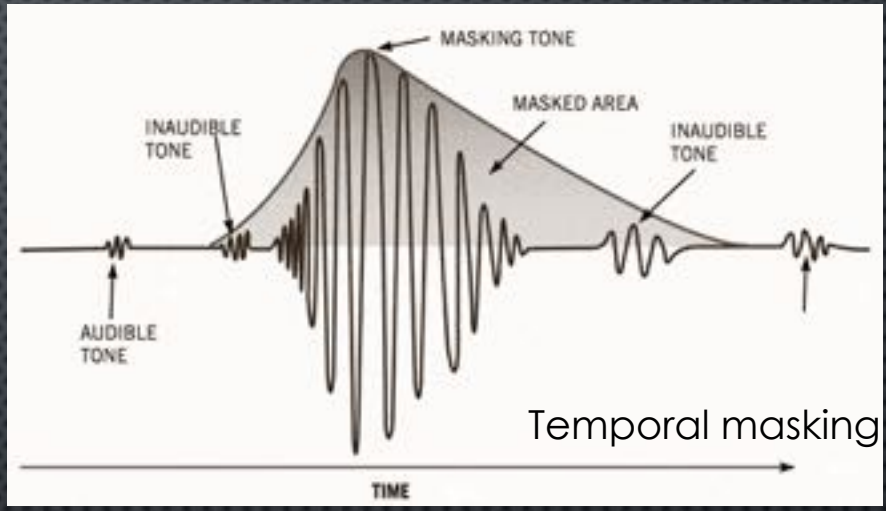
Principle of MP3 compression

- Idea: trick the human auditory system by taking advantage of its perceptual limitation.
  - Sounds below hearing threshold cannot be heard
  - Certain sounds cannot be heard
  - Certain sounds can be heard better than others
  - In the presence of two sounds, louder one is usually heard

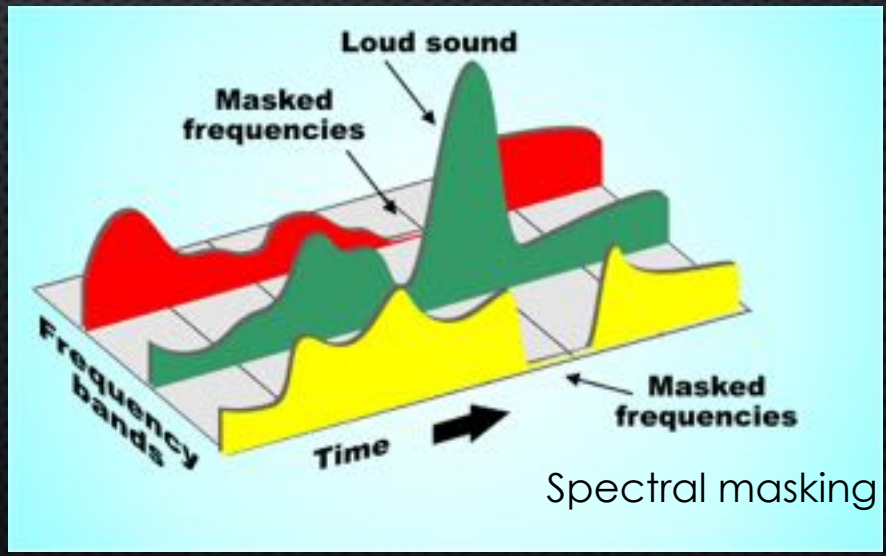


Principle of MP3 compression

- Limit frequency range, e.g. 0-16 kHz. Or use lower resolution
- Discard the data from the part of the sound where the level is below hearing threshold
- In simultaneous events, discard or preserve only limited amount of data for quieter sound(s)



Temporal masking



Spectral masking

Audio file size

- Sampling frequency/rate
  - Number of data points per second
  - e.g. 16 kHz and 44.1 kHz
- Quantisation/bit resolution, or sample size
  - Number of possible integers to represent the signal
  - e.g. 8-bit and 16-bit, i.e. 2^8 and 2^{16} level/integers
- Number of channels
  - e.g. mono – 1, stereo - 2, multichannel – many!
- Duration

Audio file size

- Bit rate
  - Number of bits processed per unite time (s)
  - kbps: kilo bits per second

$$bit\ rate\ (kbps) = \frac{sample\ rate \times bit\ resolution \times channel\ number}{1024}$$

- Byte rate

$$byte\ rate\ (kBps) = \frac{bit\ rate}{8}$$

$$file\ size\ (MB) \approx \frac{byte\ rate}{1024} \times t$$

Don't forget the header of a file!

Common audio file types

Type	Platform	Extension	Compression	Loss	Proprietary
WAV	cross	.wav	No	lossless	No
AIFF	Mac	.aif, .aiff	No	lossless	No
FLAC	cross	.flac	FLAC	lossless	No
MP3	cross	.mp3	MPEG	lossy	Yes
AAC	cross	.m4a, m4b, m4p, m4r, 3gp, mp4, .acc	AAC	lossy	Yes
WMA	Windows	.wma	WMA	lossy	Yes
OGG Vorbis	cross	.ogg, oga	Vorbis	lossy	No

Waveform Audio File Format (.wav)

- A standard of audio file format
  - Introduced by Microsoft and IBM in early 90's
  - Counterpart: AIFF by Apple – audio interchange file format (.aiff)
- Uncompressed format
  - Used for high-quality raw audio data
  - Takes more space than compressed format
- Data structure
  - Three chunks of header + data

The "RIFF" chunk descriptor

Format is "WAVE"

The "fmt" sub-chunk

Includes the sound information in the data sub-chunk

The "data" sub-chunk

Indicates the size of actual sound data and contains the raw data

File offset (bytes)	Field	Field size (bytes)	
4	ChunkID	4	RIFF
8	ChunkSize	4	N+36
12	Format	4	WAV
16	Subchunk1ID	4	fmt
20	Subchunk1Size	4	16
22	AudioFormat	2	1
24	NumChannels	2	1
28	SampleRate	4	44100
32	ByteRate (Bps)	4	88200
34	BlockAlign	2	2
36	BitsPerSample	2	16
40	Subchunk2ID	4	data
44	Subchunk2Size	4	N
filesize	data	Subchunk2 Size	

Header

Raw data

Given the following signal:

- Sample rate: **16000** Hz
- Duration: **120** seconds
- Bit resolution: **8**-bit
- Number of channel: **5**

What is the file size in MB?

File offset (bytes)	Field	Content
4	ChunkID	RIFF
8	ChunkSize	?
12	Format	WAV
16	Subchunk1ID	fmt
20	Subchunk1Size	?
22	AudioFormat	?
24	NumChannels	?
28	SampleRate	?
32	ByteRate (Bps)	?
34	BlockAlign	?
36	BitsPerSample	?
40	Subchunk2ID	data
44	Subchunk2Size	?
	data	

Python API for WAV

- Module: ***wave.py***
  - Use: "***import wave***"
- Methods:
  - *obj = wave.open(FILE, mode)*
  - mode = "rb": *obj* is a *Wave_read* object
  - mode = "wb: *obj* is a *Wave_write* object
- *Wave_read* and *Wave_write* offer different methods for file handling

Wave_read object

- obj.getnchannels()
- obj.getsamplewidth()
- obj.getframerate()
- obj.getnframes()
- obj.getparams()
- obj.setpos(pos)
- obj.tell()
- obj.rewind()
- obj.readframes(n): return all data in a *bytes* object
- obj.close()

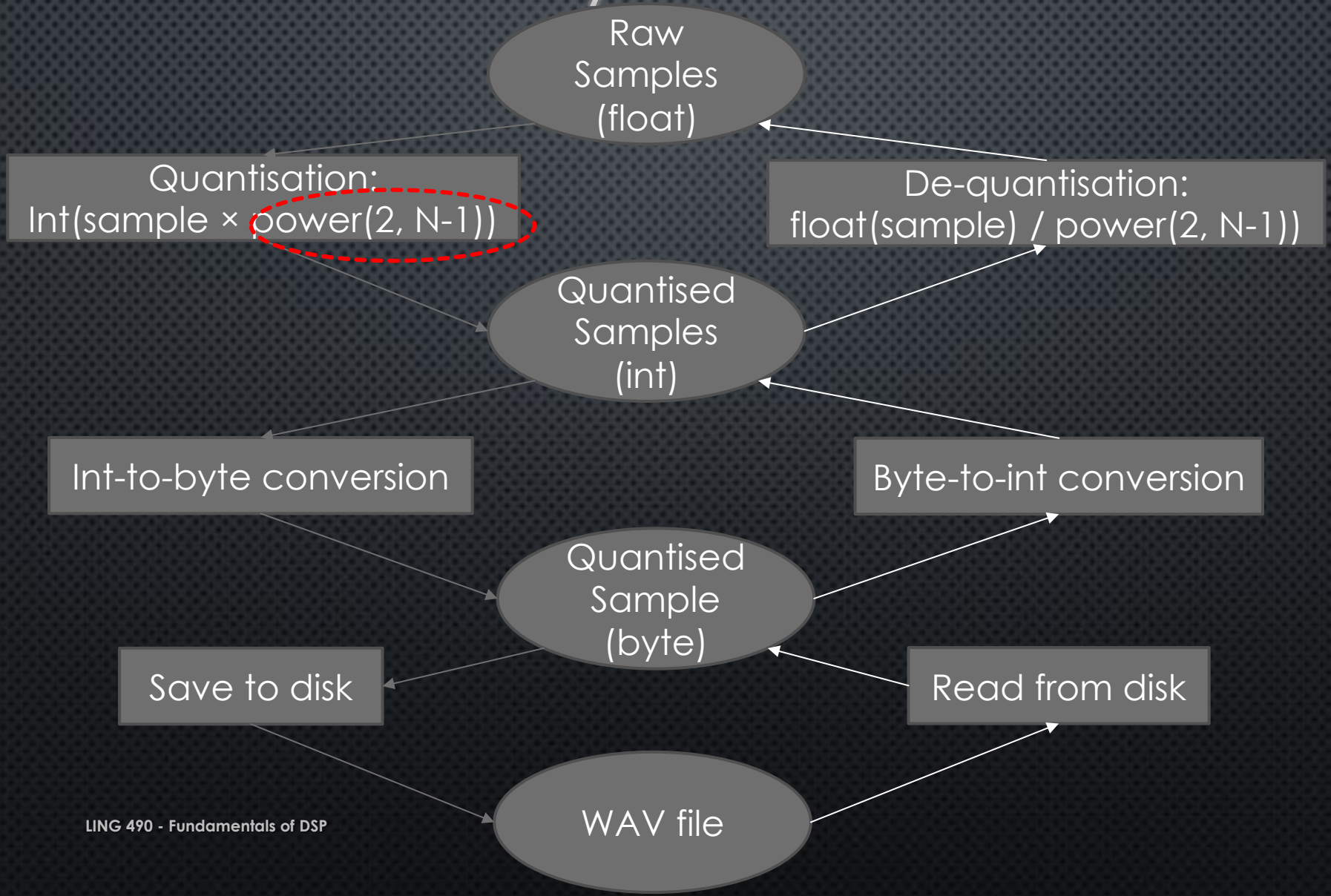
https://docs.python.org/3/library/wave.html

Wave_write object

- obj.setnchannels(n)
- obj.setsamplewidth()
- obj.setframerate(n)
- obj.setnframes(n)
- obj.setparams(tuple)
- obj.tell()
- obj.writeframes(n): where n is a *bytes* object
- obj.close()

https://docs.python.org/3/library/wave.html

Raw data – Bytes – raw data



LIN490 ***WAVReader*** and ***WAVWriter*** classes

- Wrappers for Python *Wave_read* and *Wave_write* objects
- Byte-to-data and data-to-byte conversion
  - __decode() in WAVReader: byte-to-data
  - __encode() in WAVWriter: data-to-byte
- Takes care of clipping/scaling issue during writing
- Header is generated automatically when file is capsulated