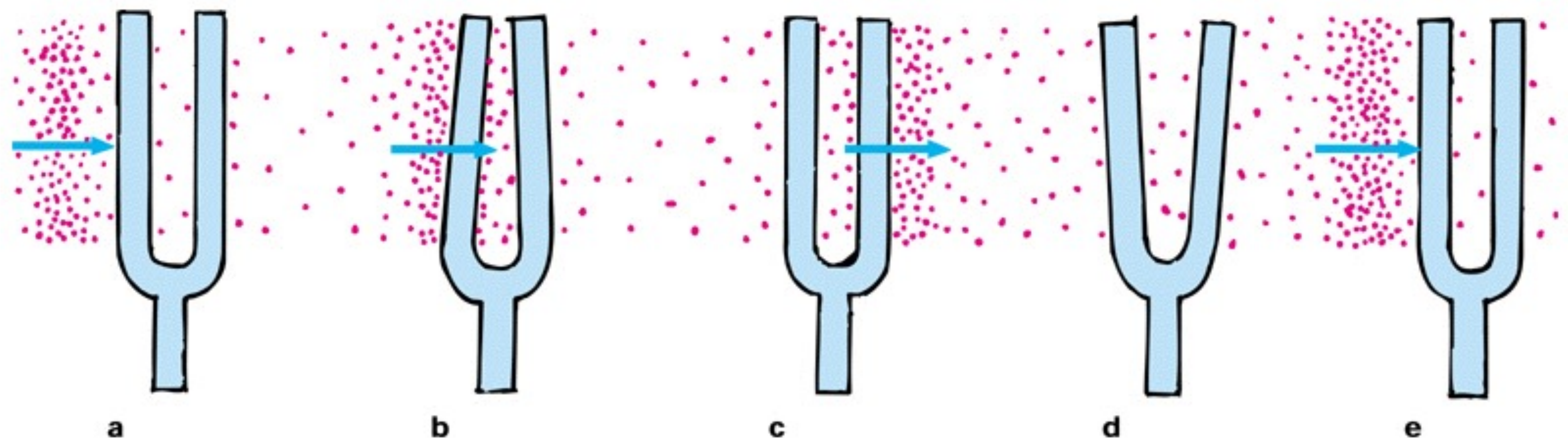


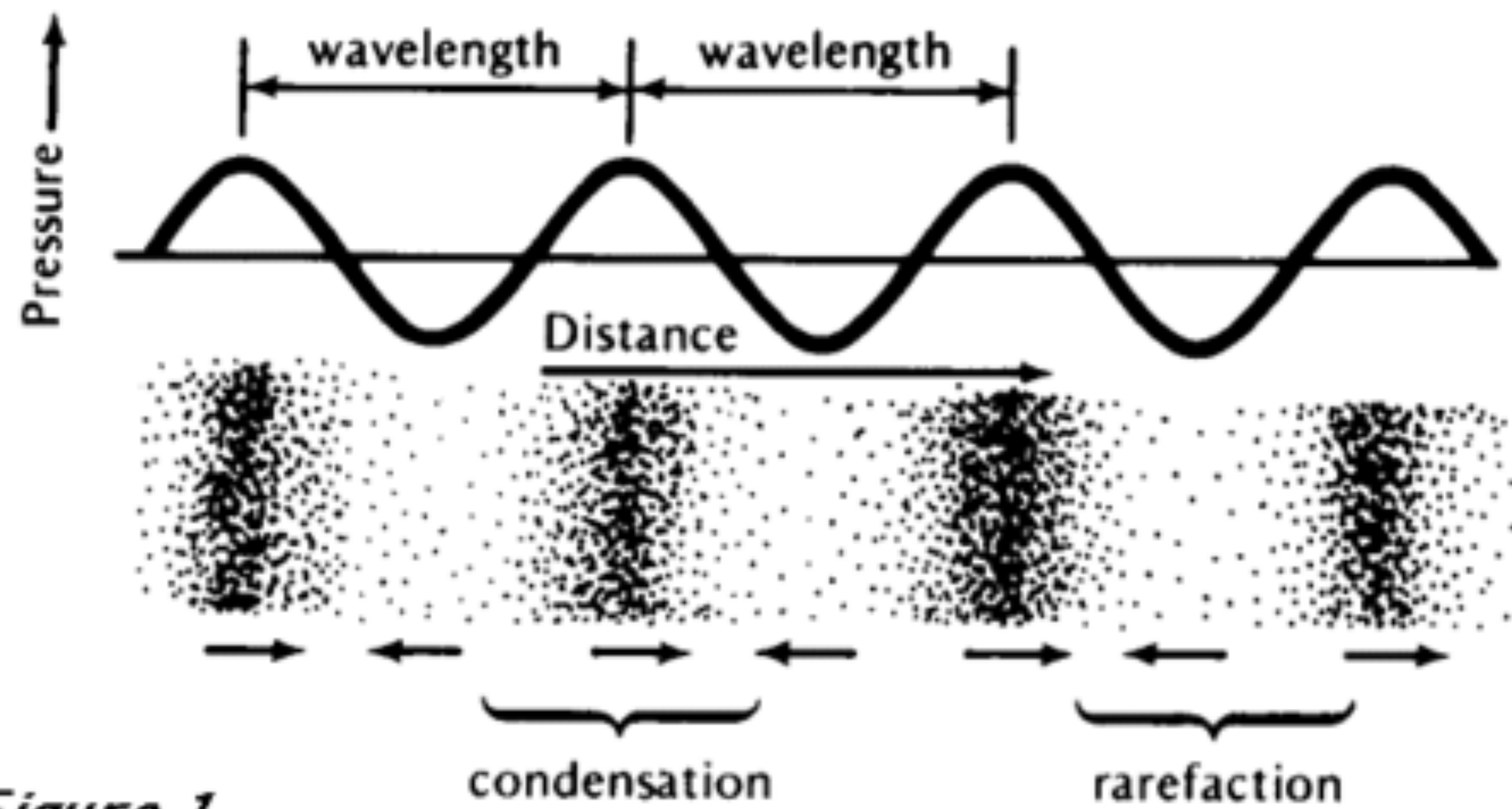
# Playing sound.



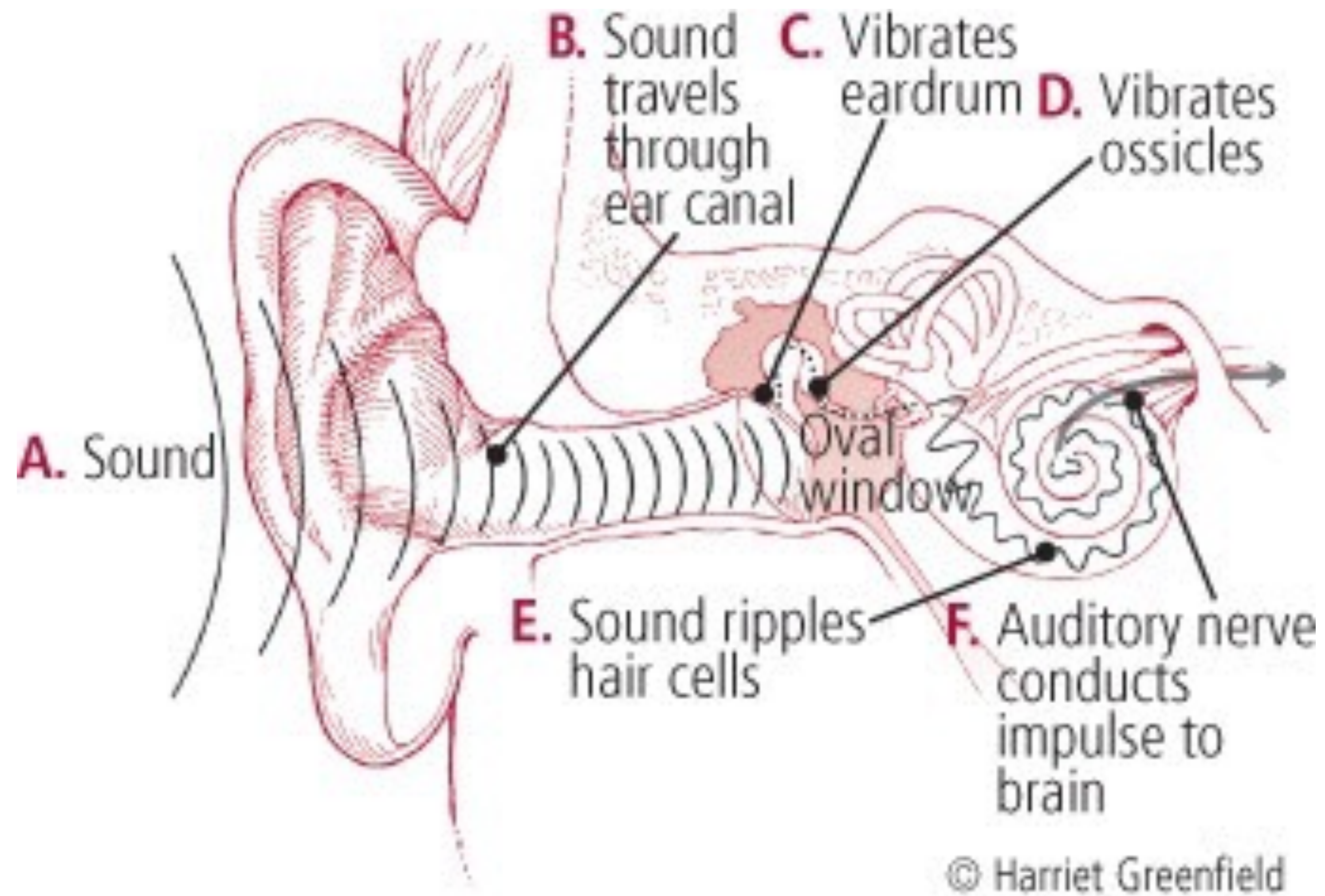
How **sound** works.

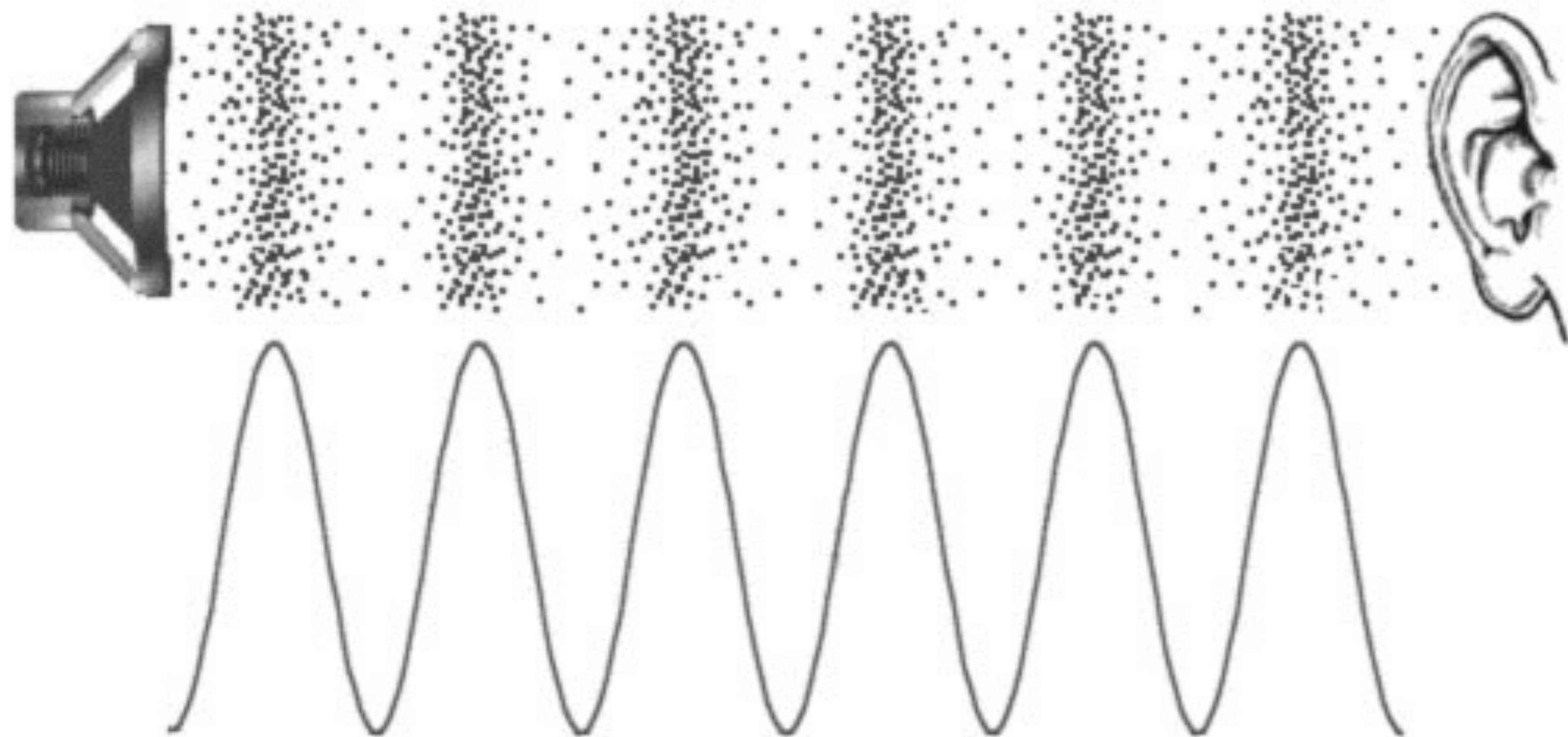




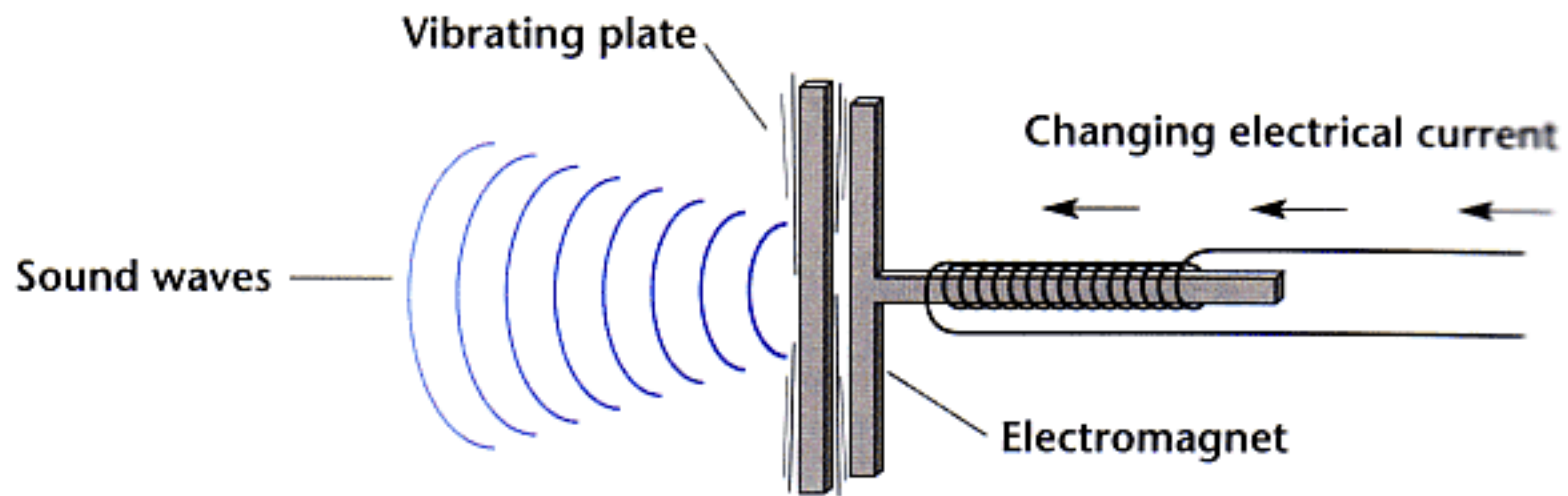


*Figure 1*



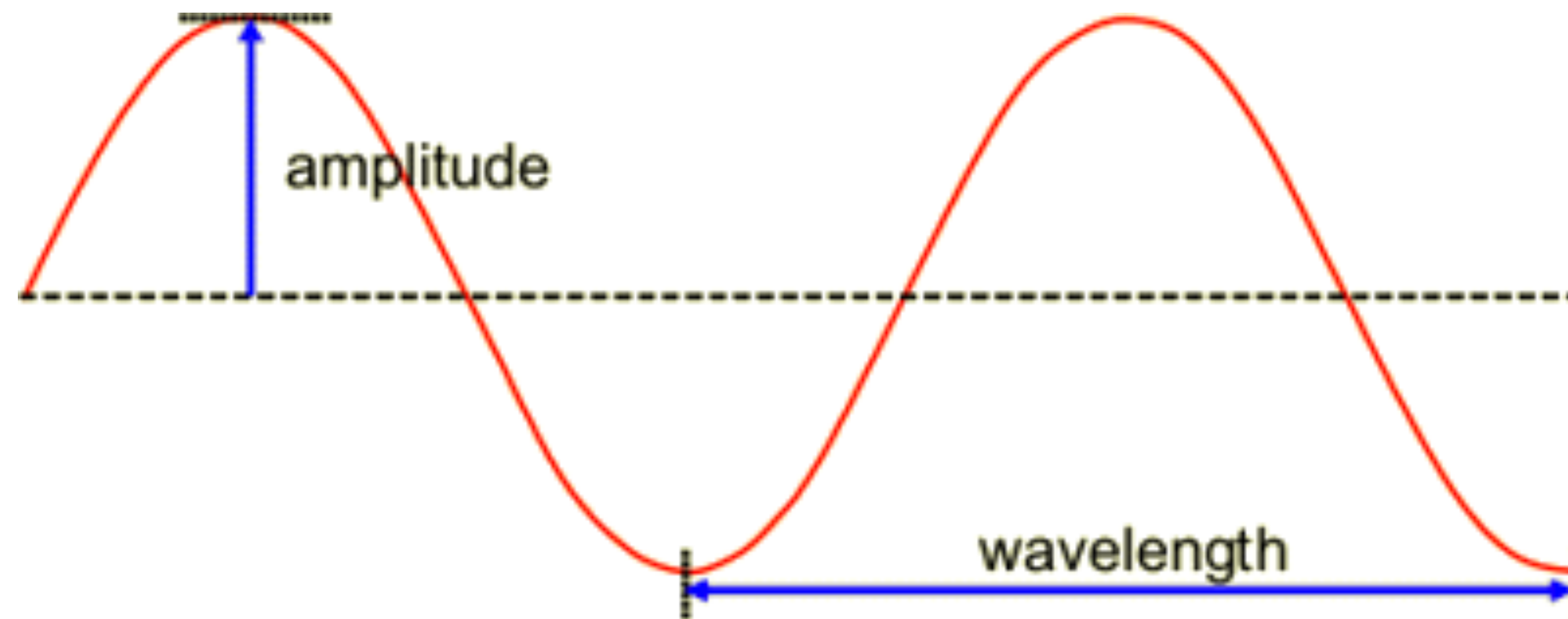


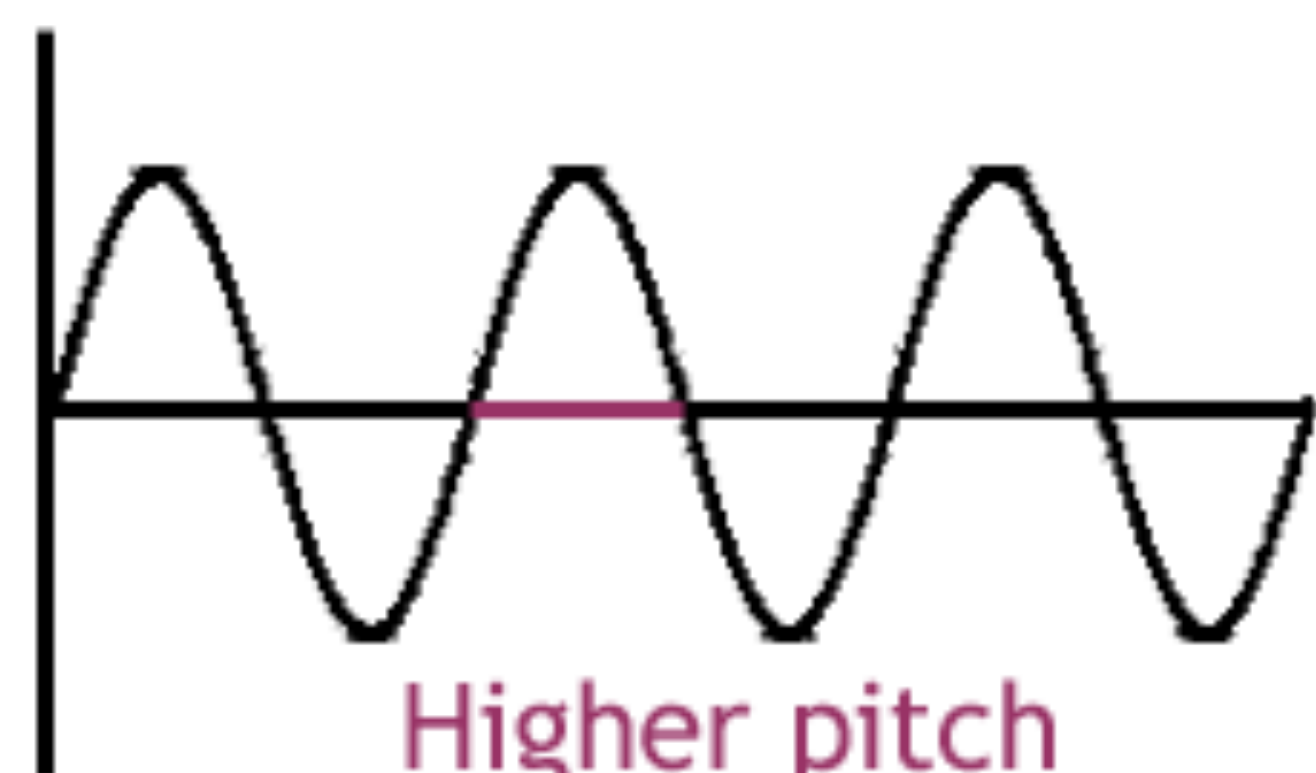
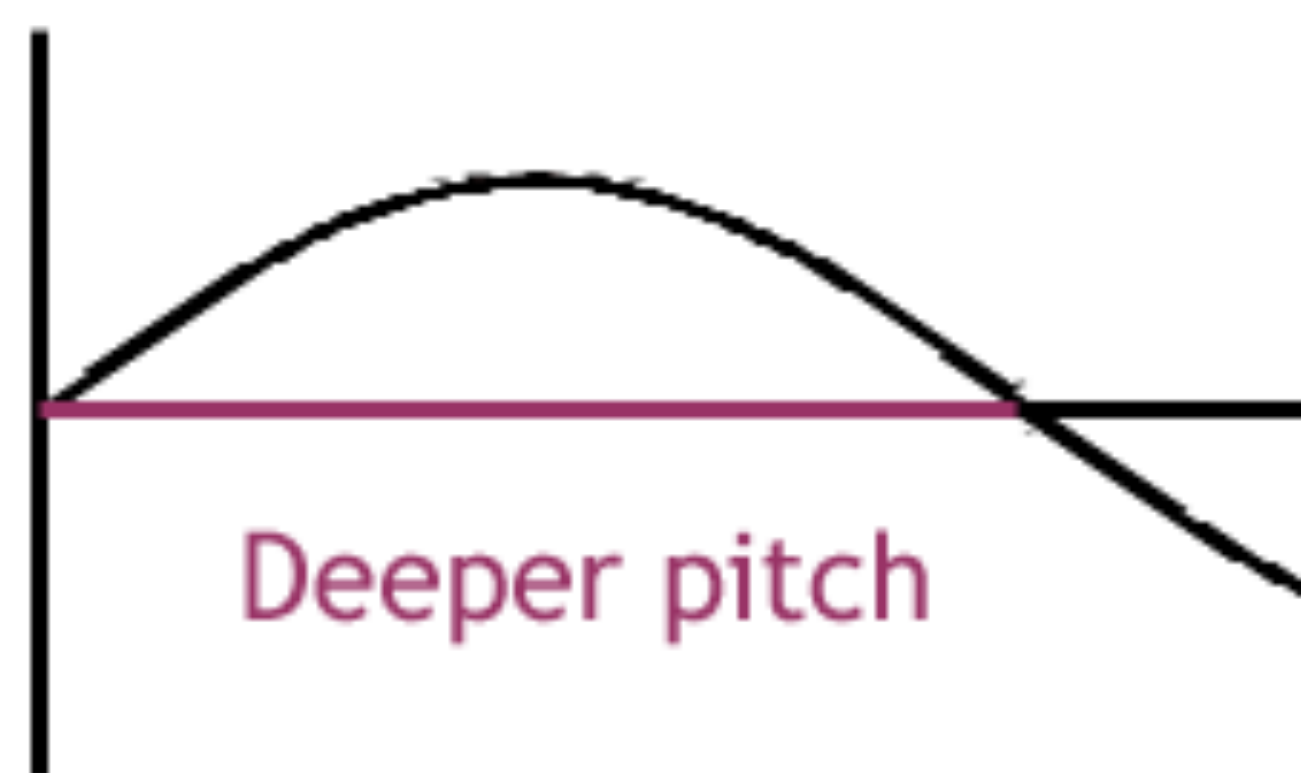
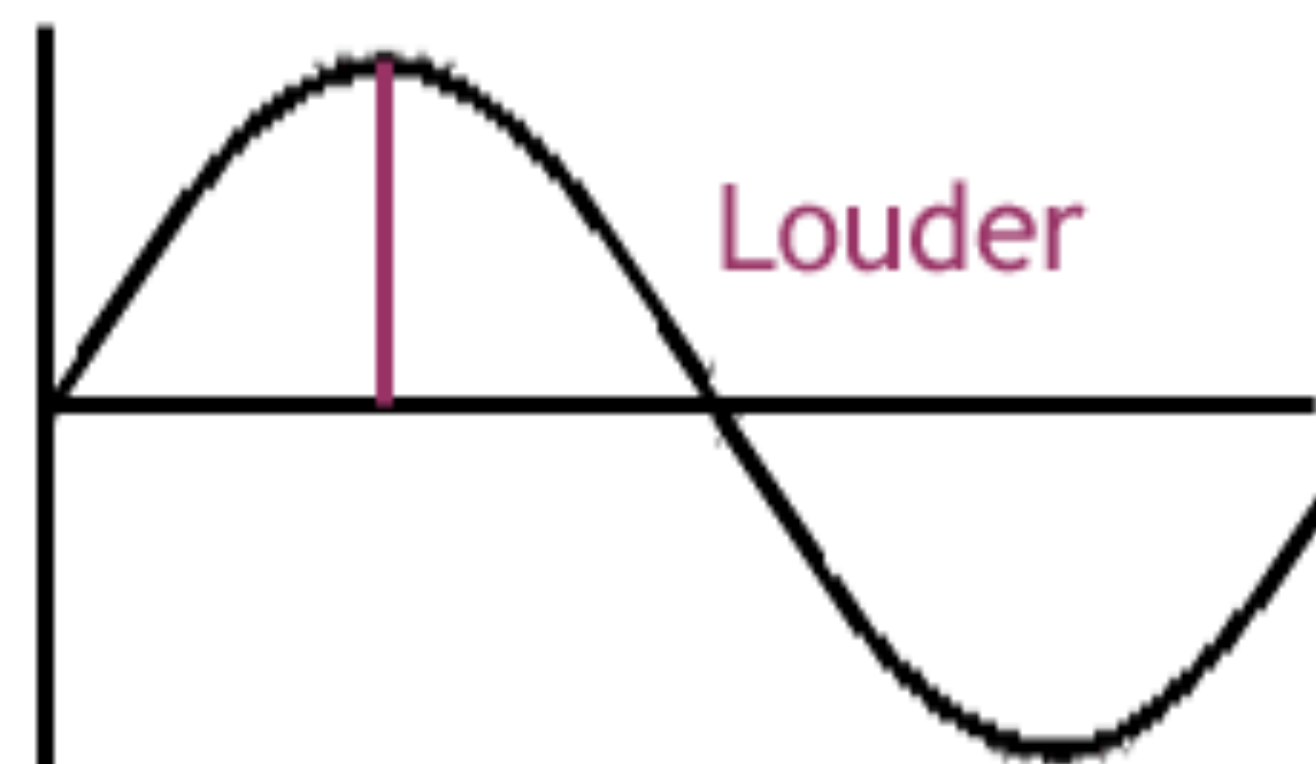
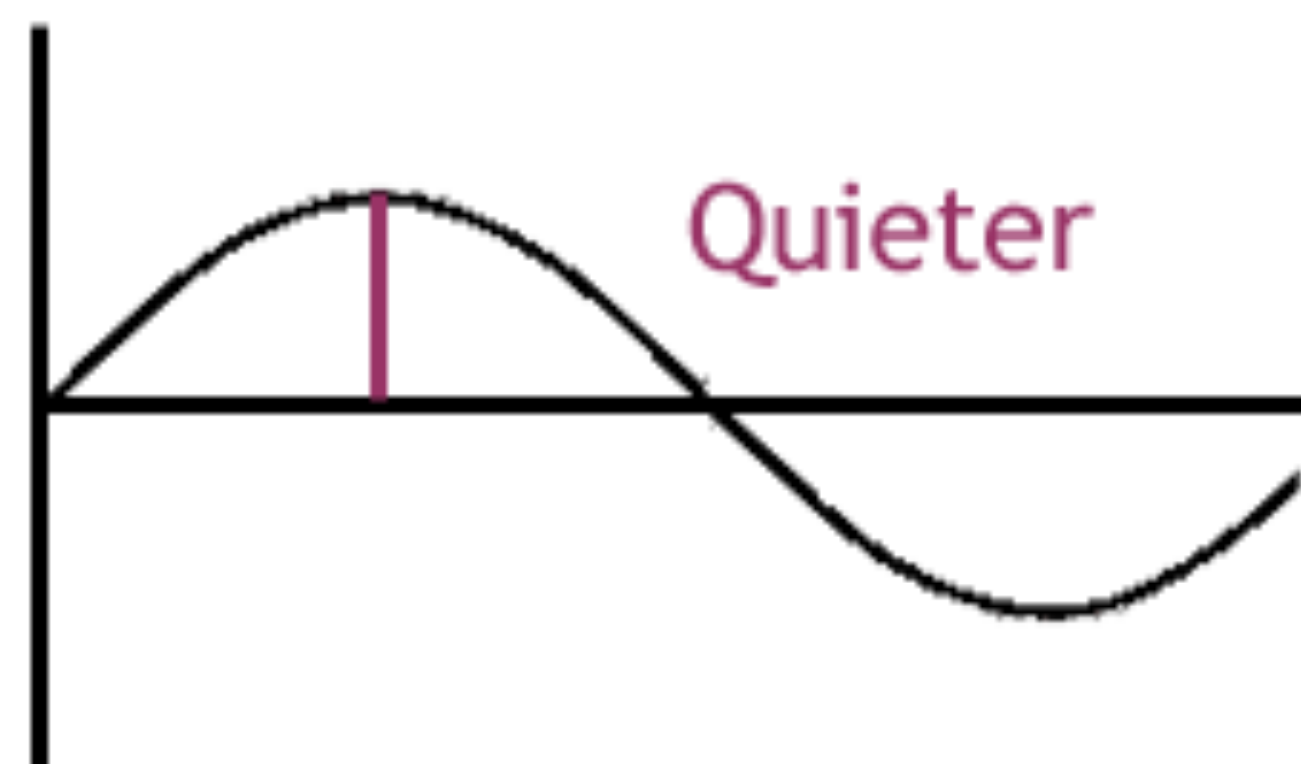




Sound wave properties.



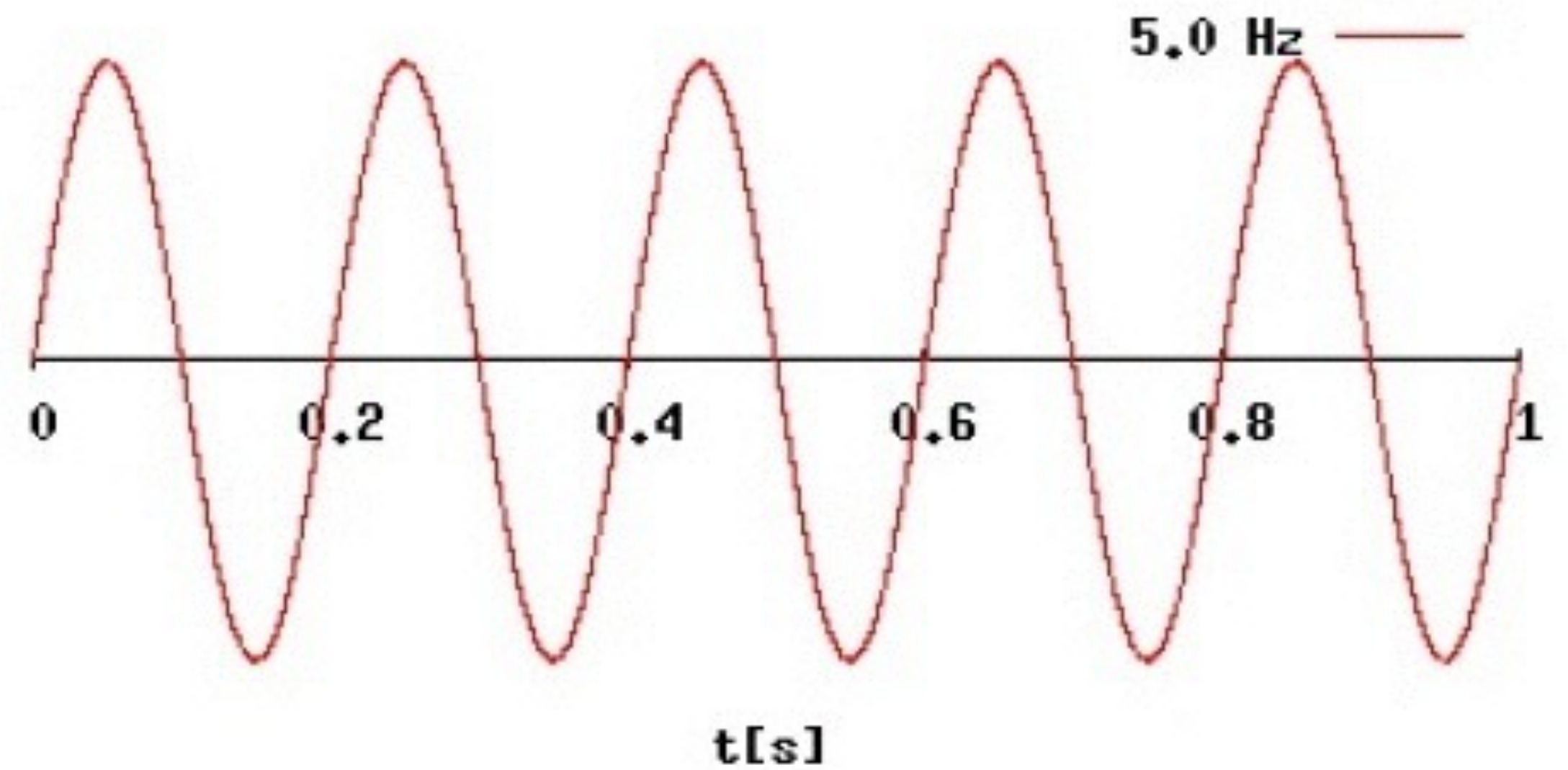
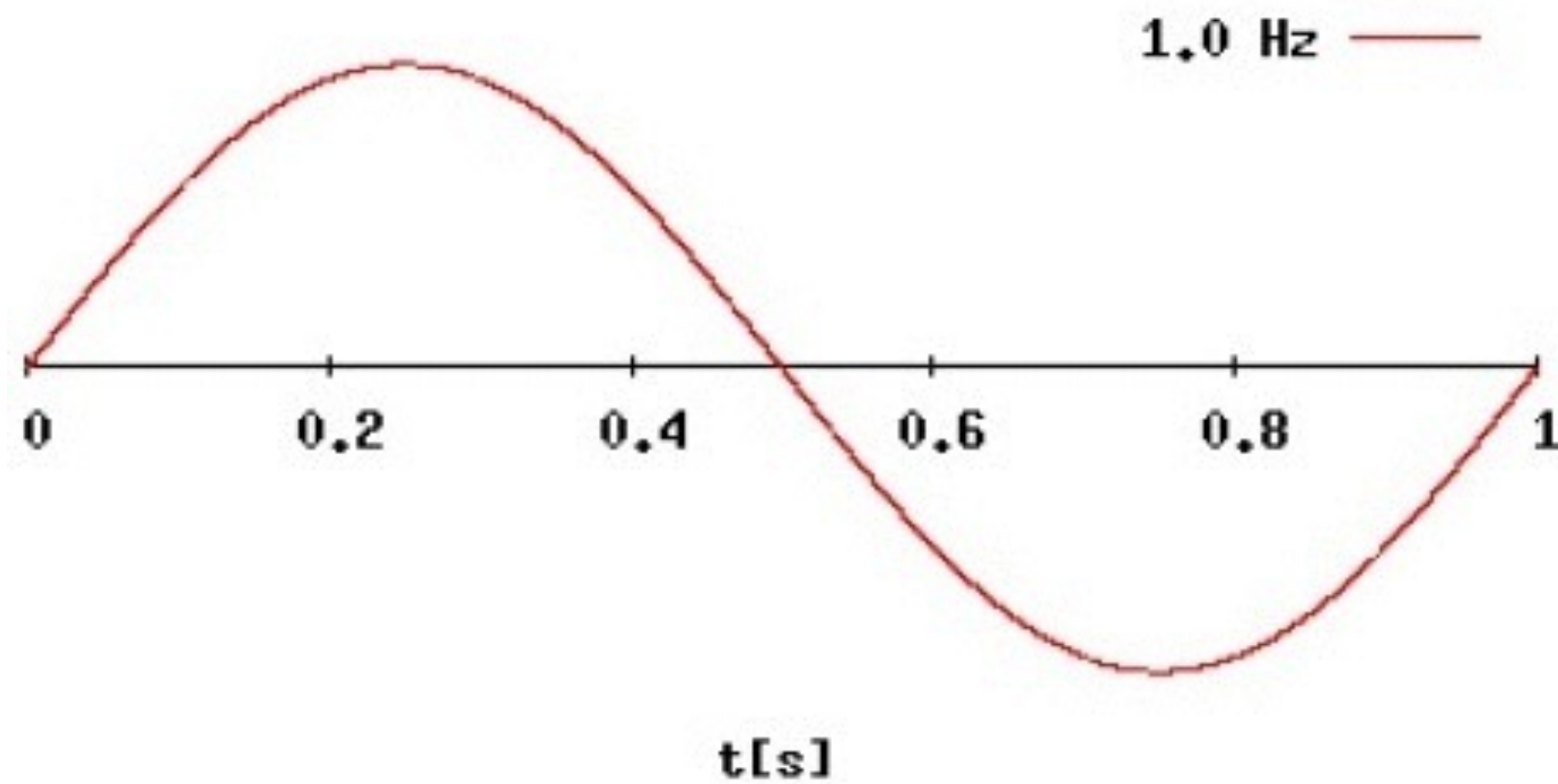




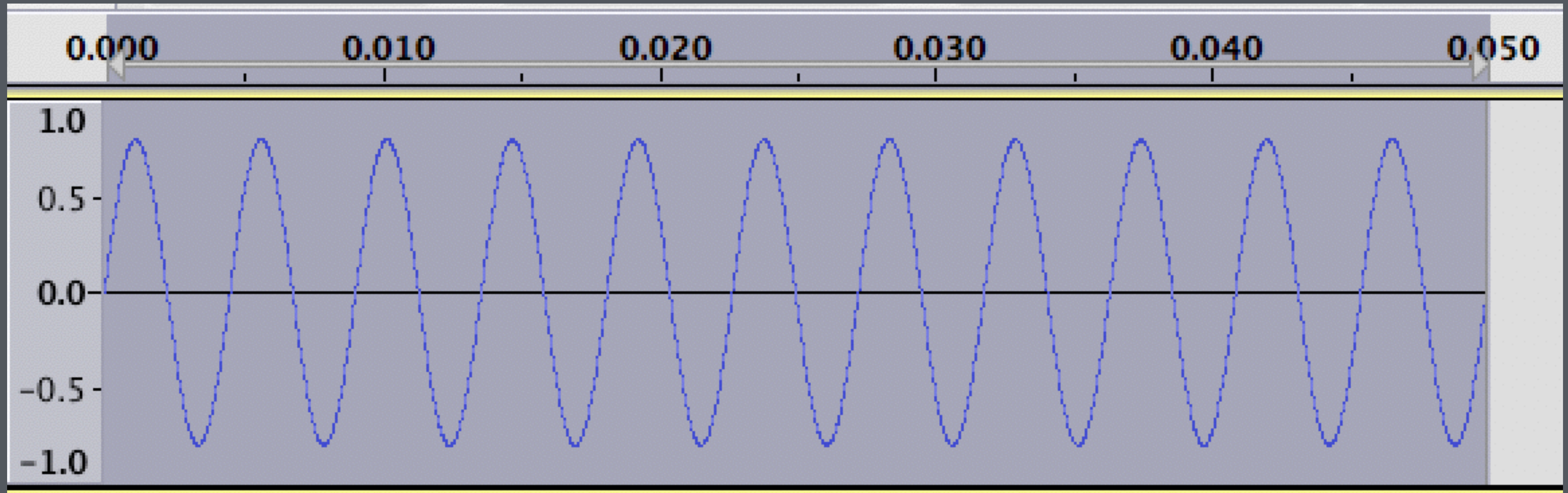
Sound frequency.



hertz (Hz) = 1 cycle per second.

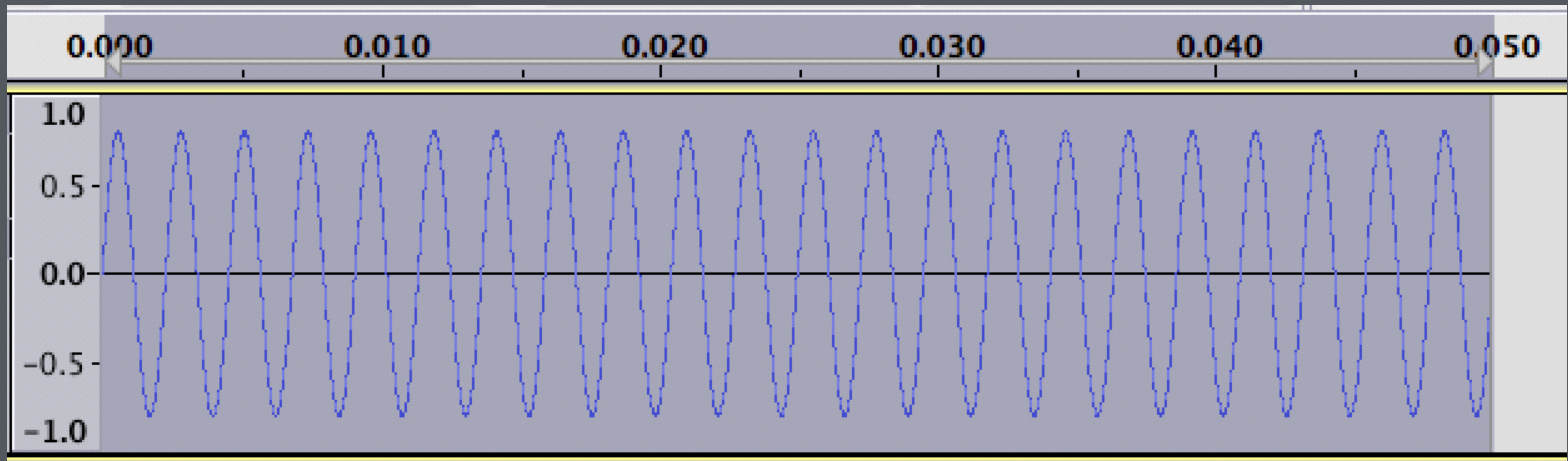


# 220 Hz



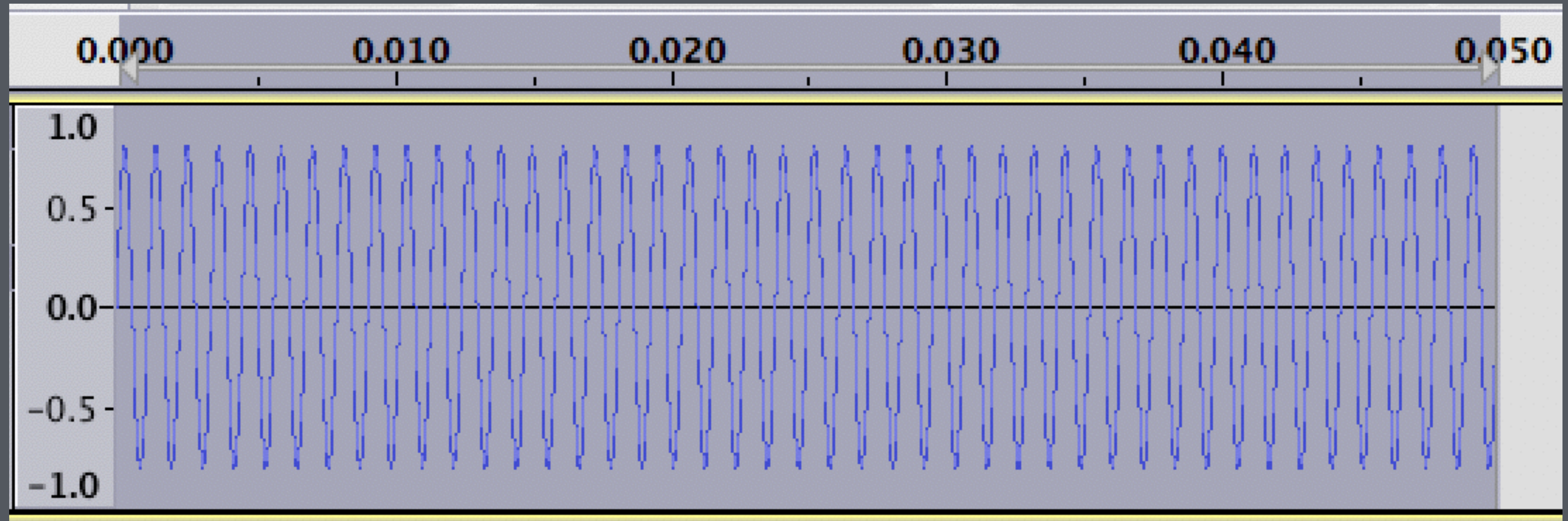


# 440 Hz





# 880 Hz



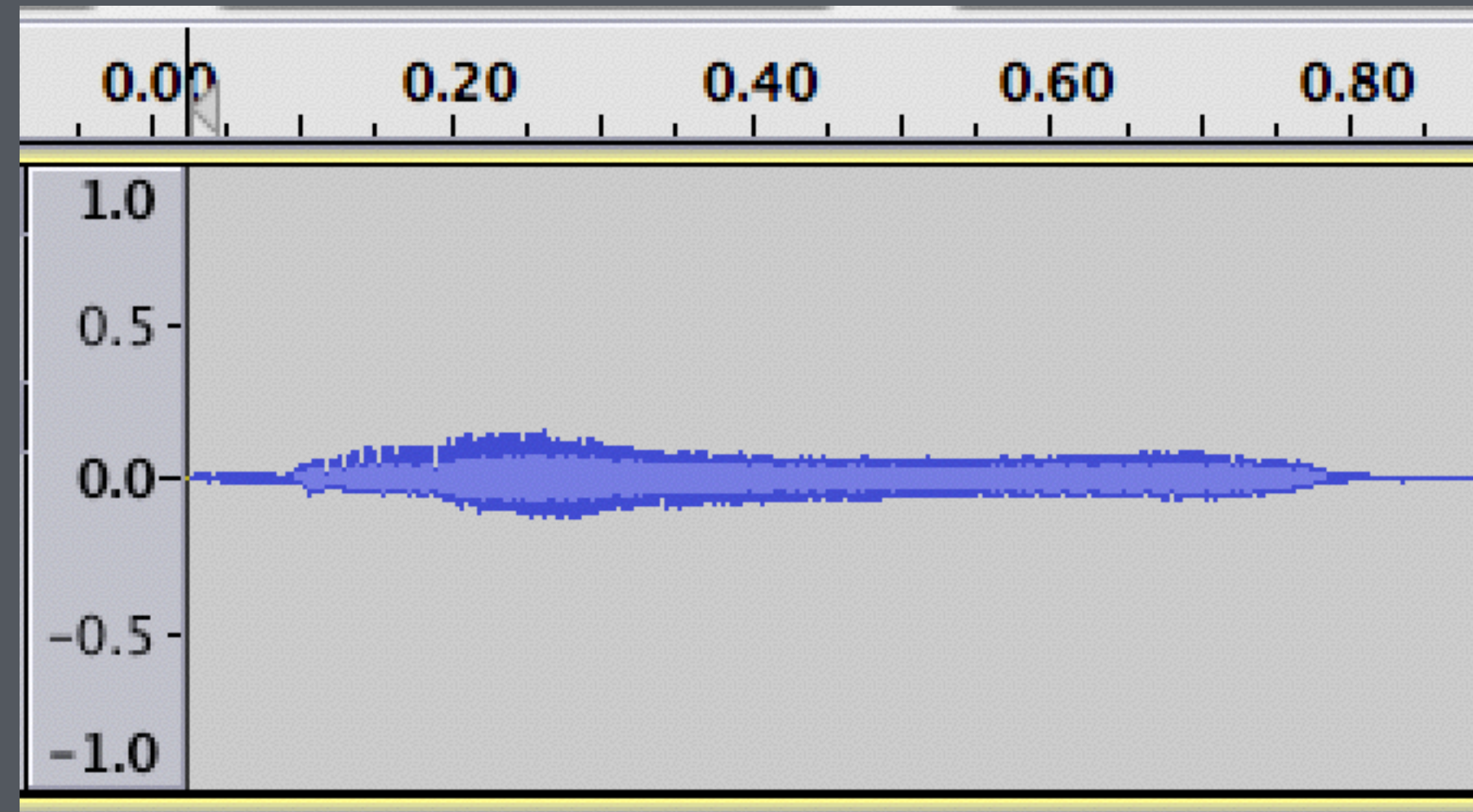
Complex waveforms.

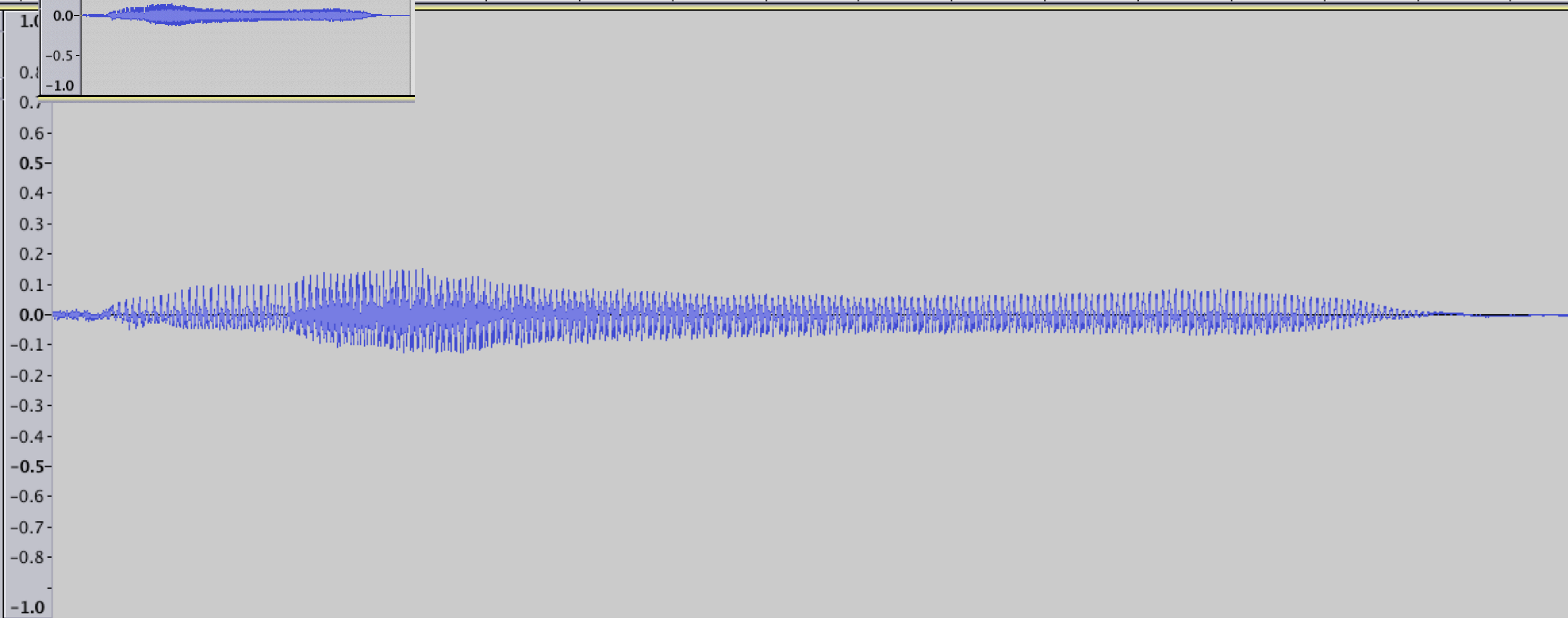
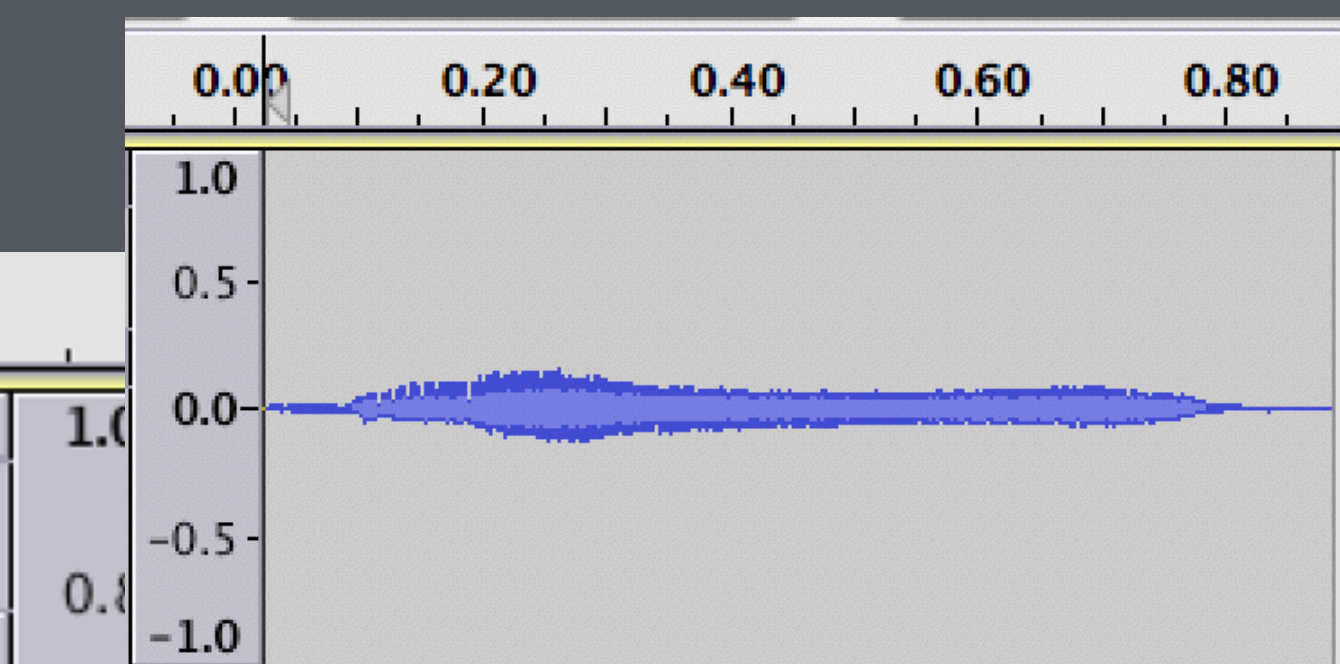
**Modulating frequency.**



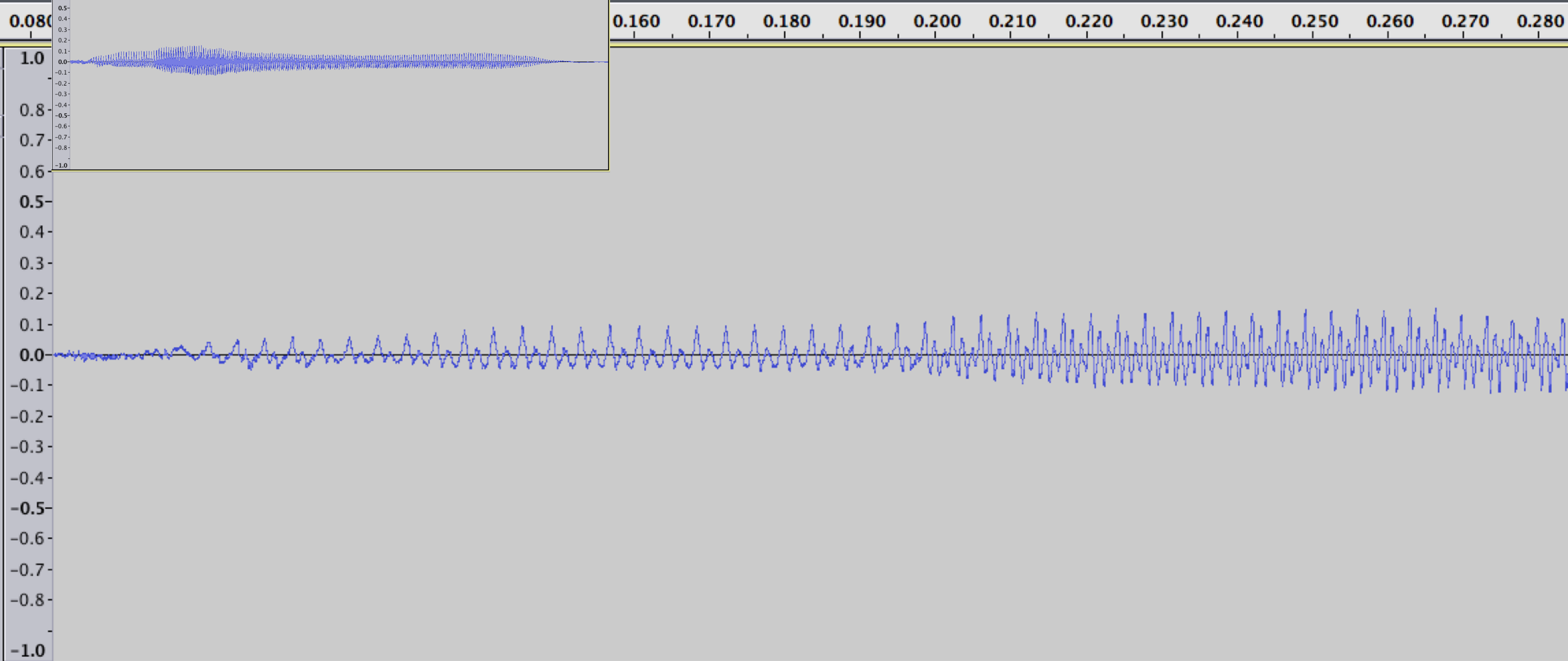
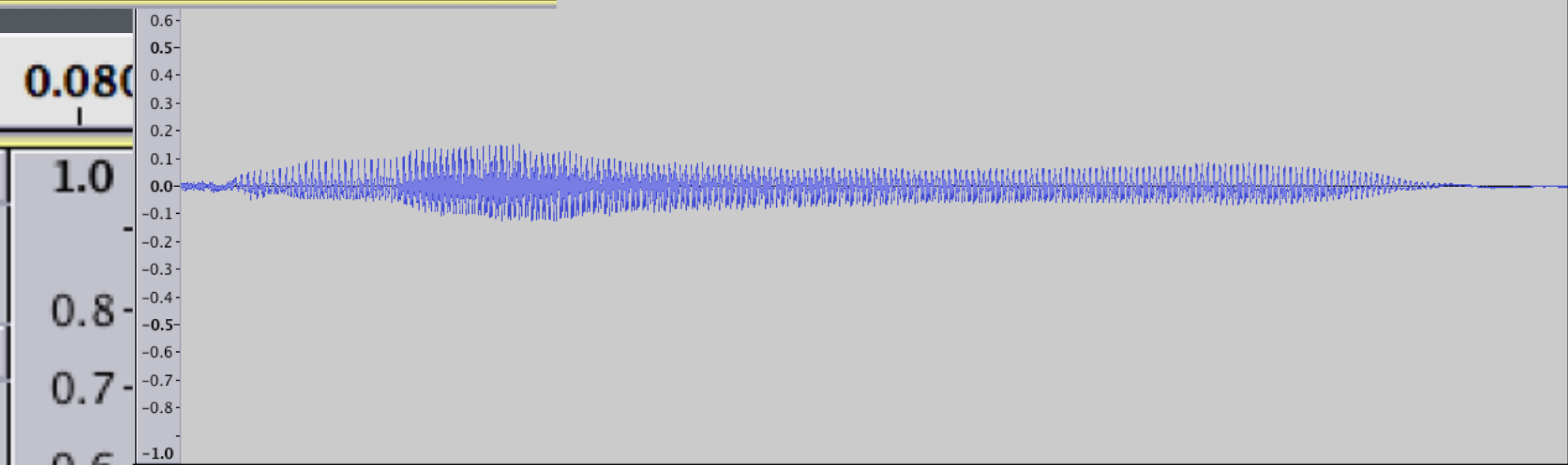
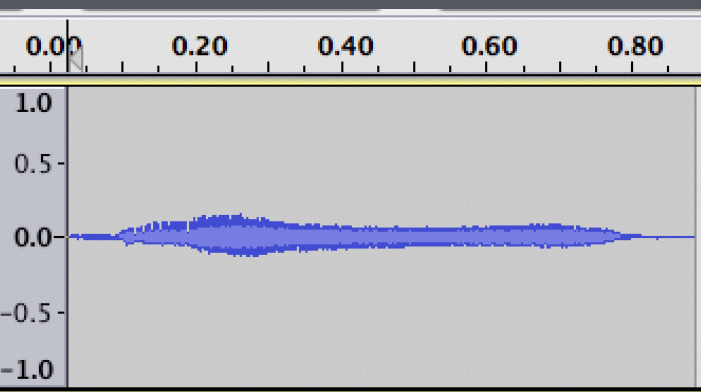


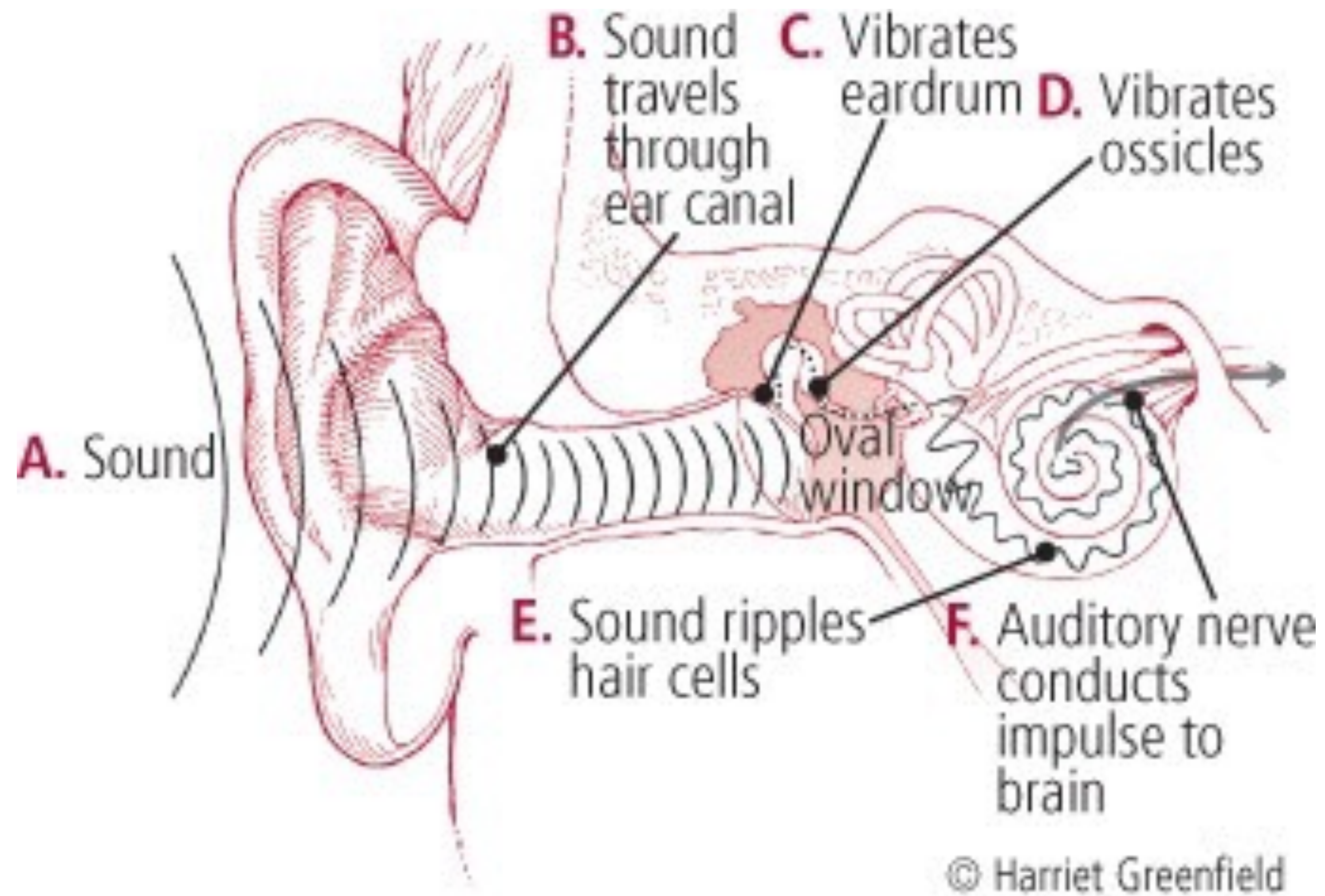
# Hello







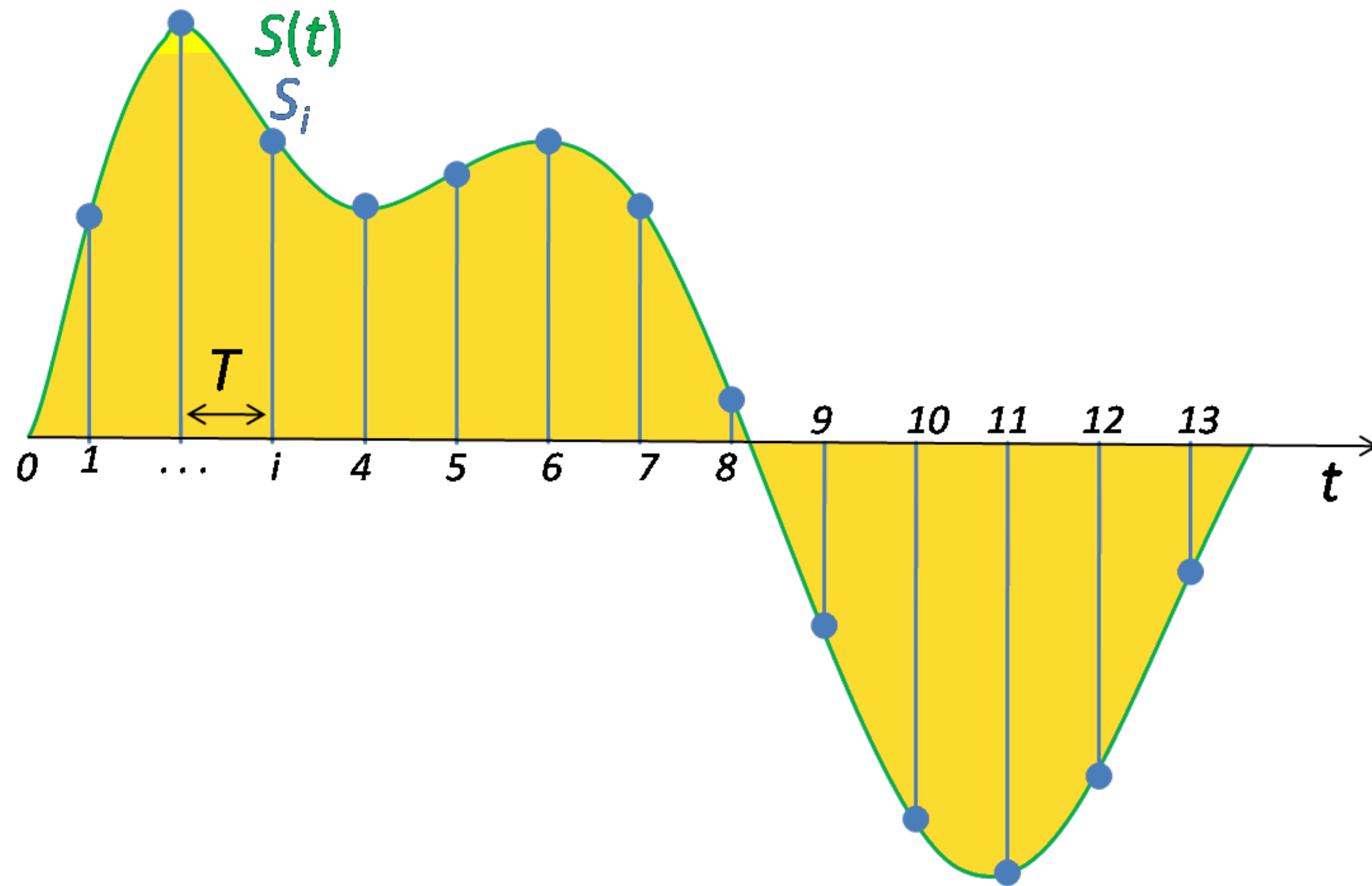




How digital sound works.

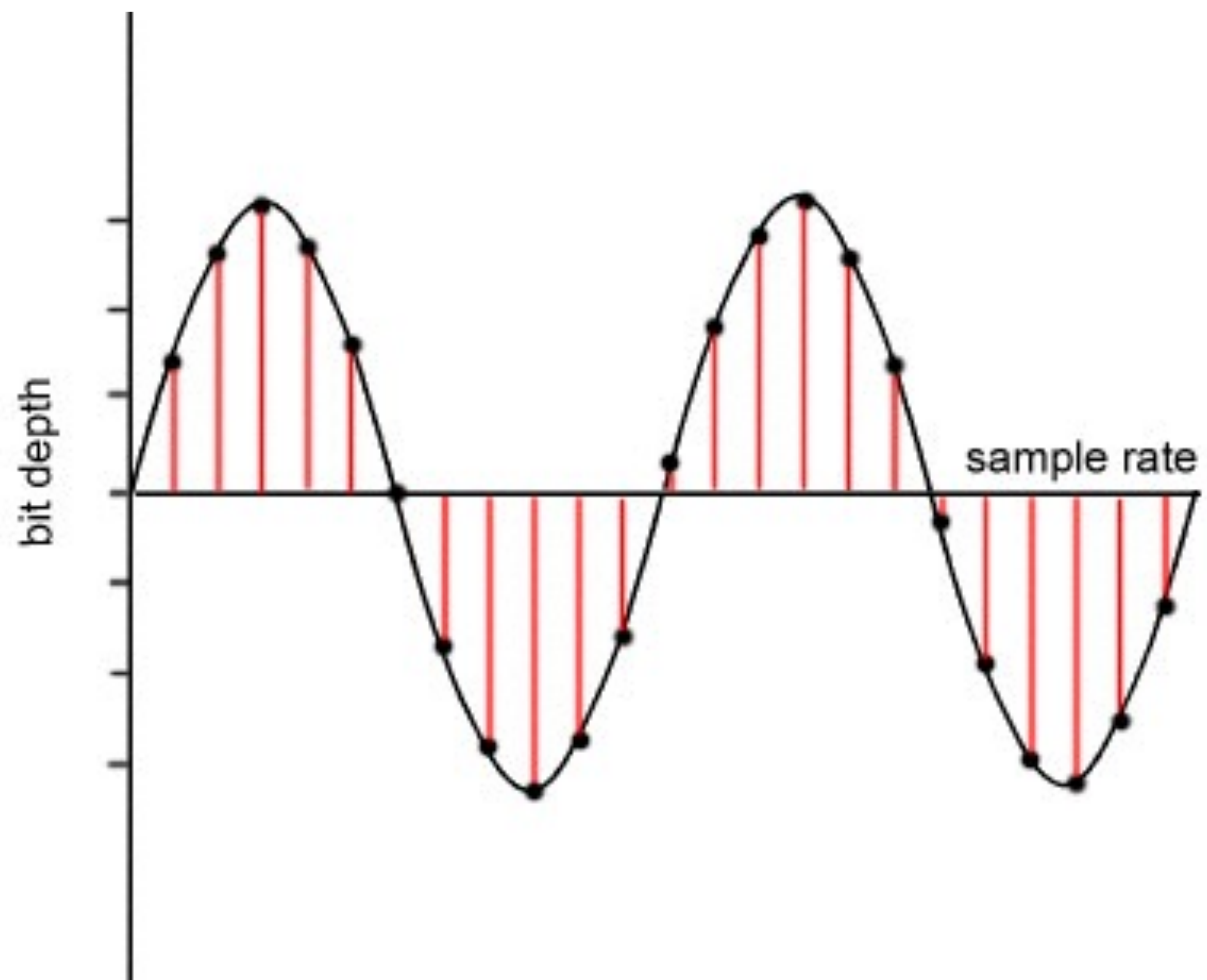
Digital sampling.





Sampling quality.

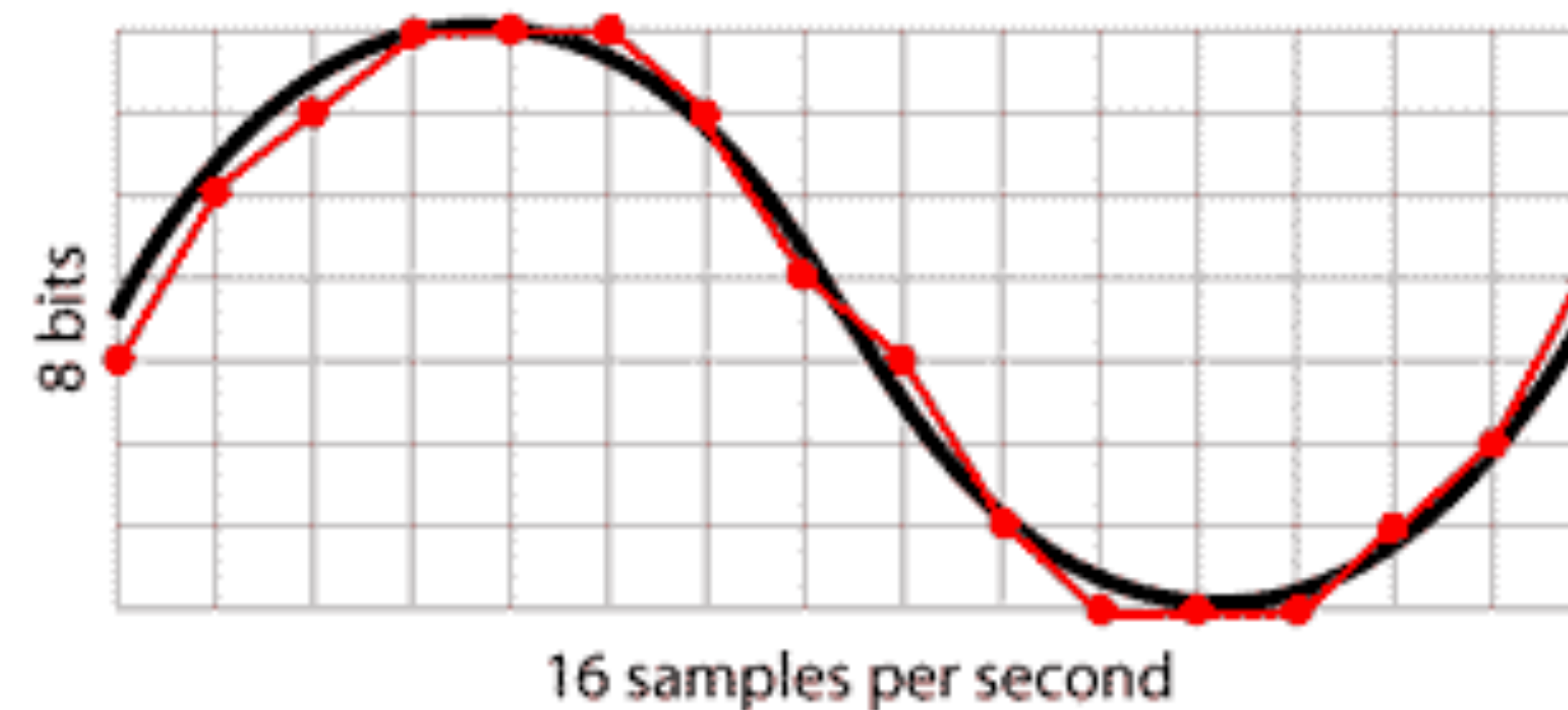
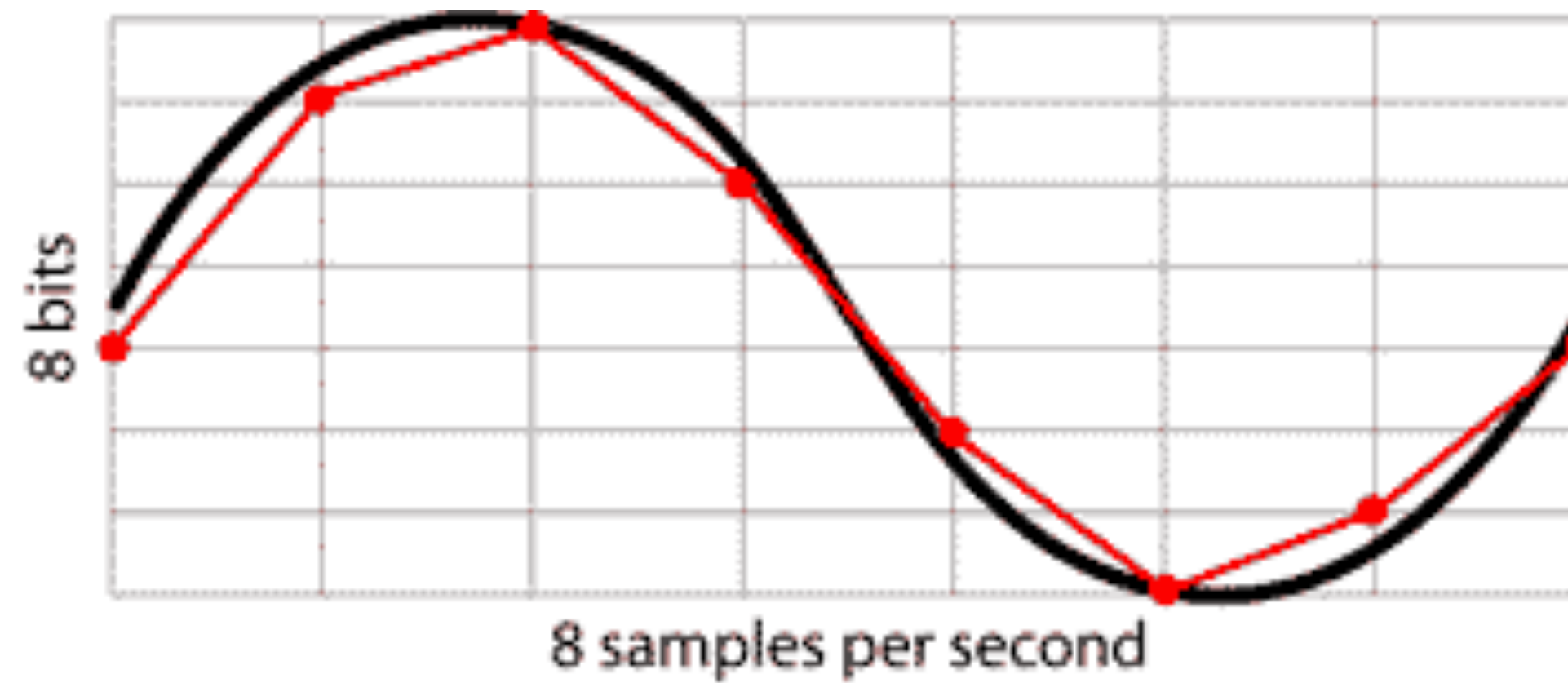
Sample rate and bits per sample.





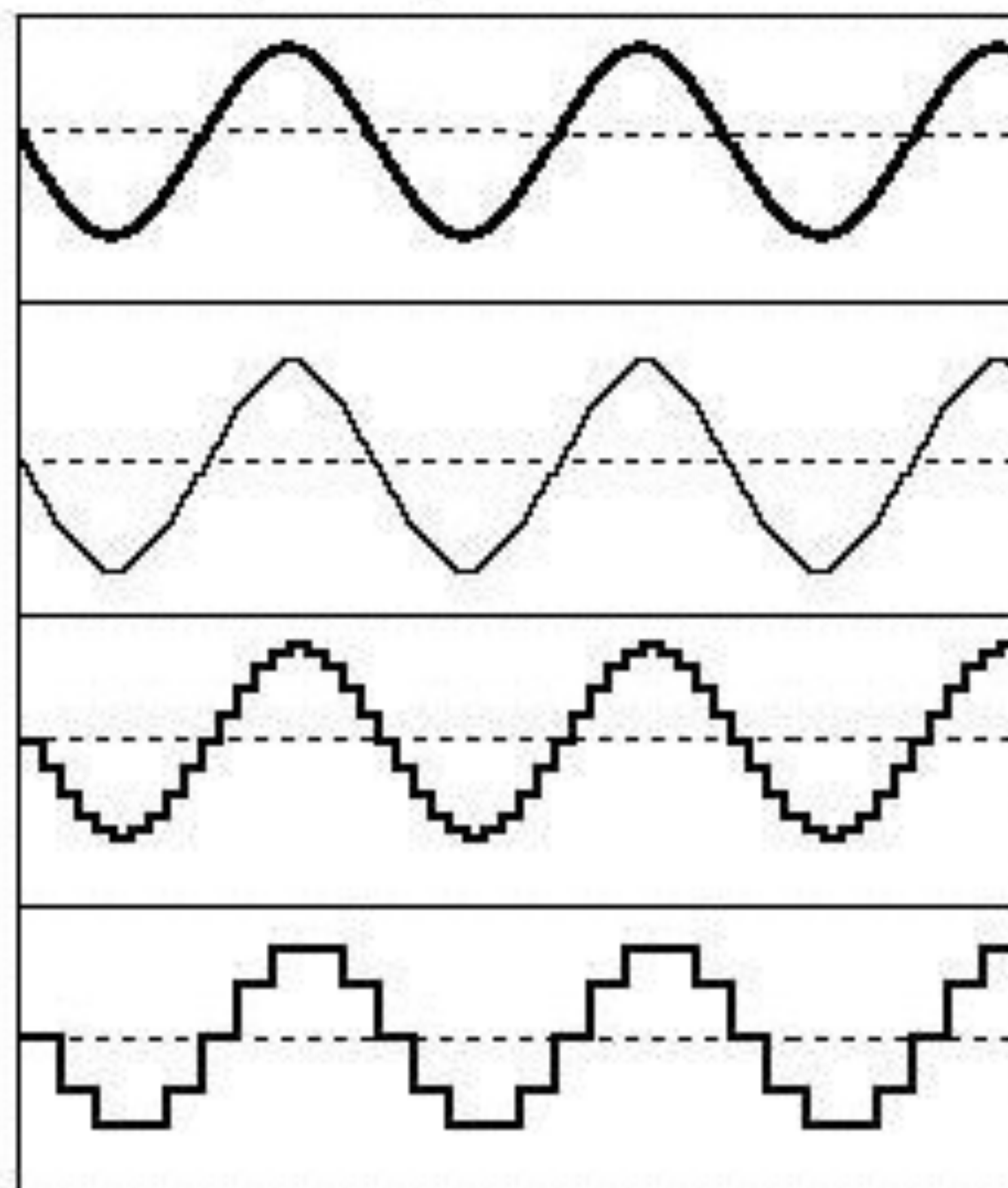
Sample rate.

How many **discrete samples per second** (measured in Hz).



Bits per sample.

### Sound quality and bits.



Audio waveform

Waveform sampled  
at 22 bits.

Waveform sampled  
at 16 bits.

Waveform sampled  
at 8 bits.



Bitrate.

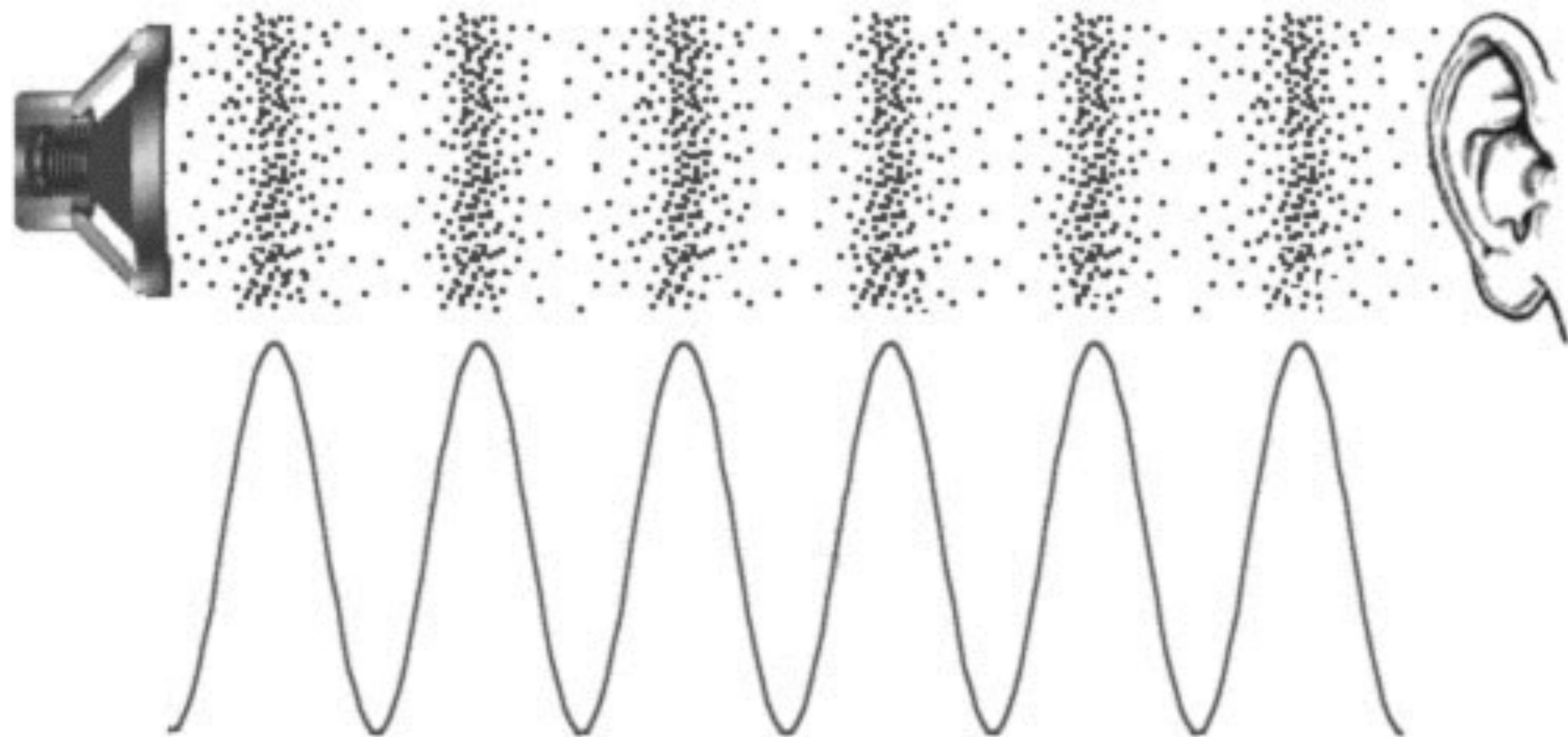
# Bitrate.

**Sampling rate combined with bit depth.**

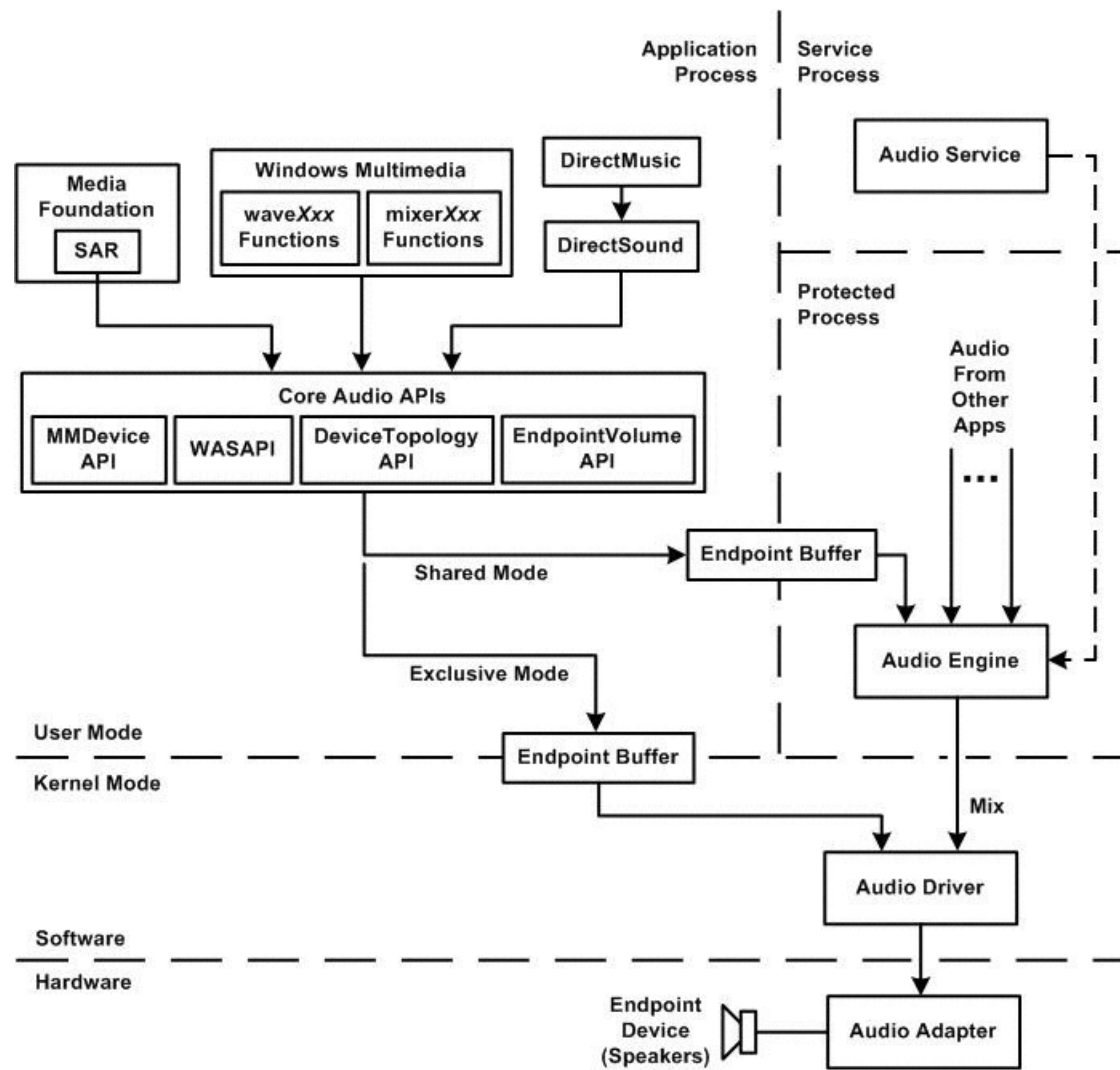
**Reflects the total bit count processed per second.**

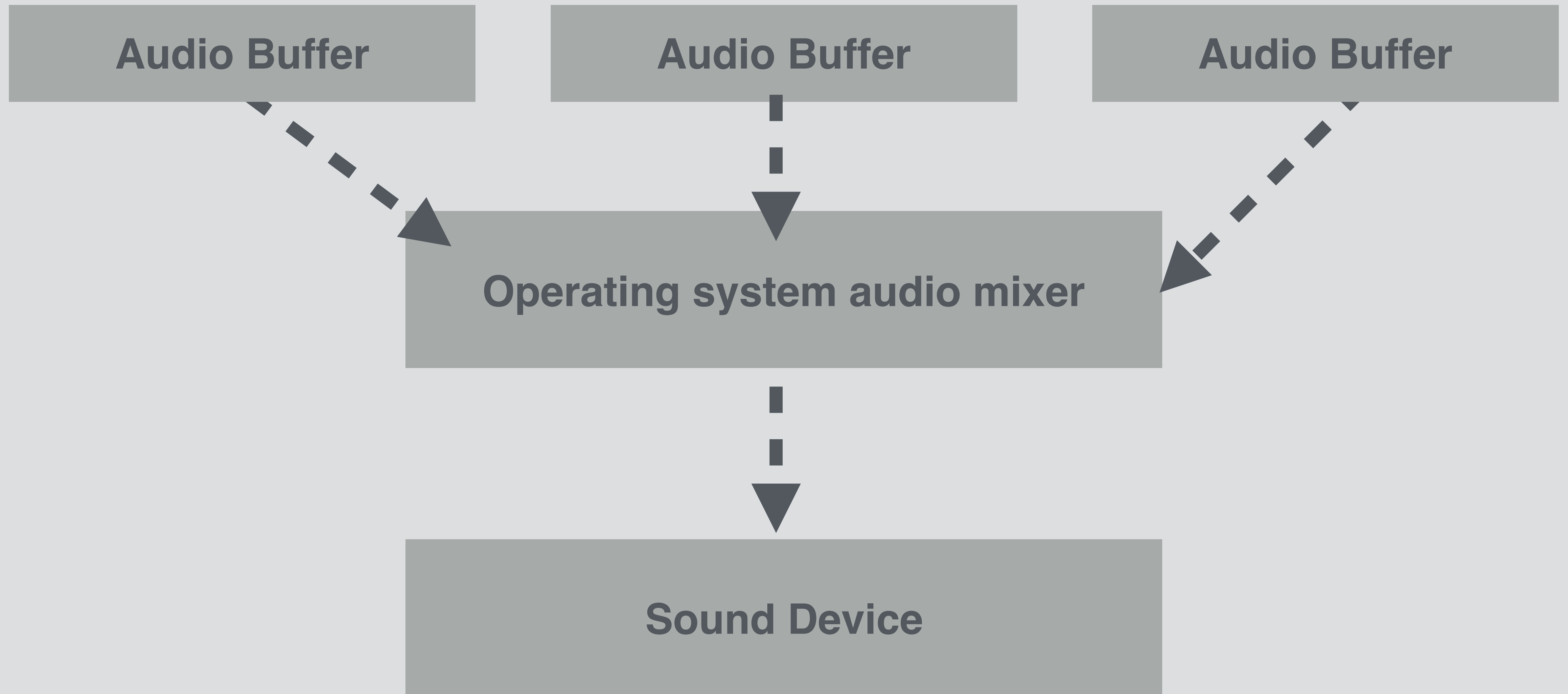
**Measured in bits per second or bit/s (or bps)**

Playing digital audio.

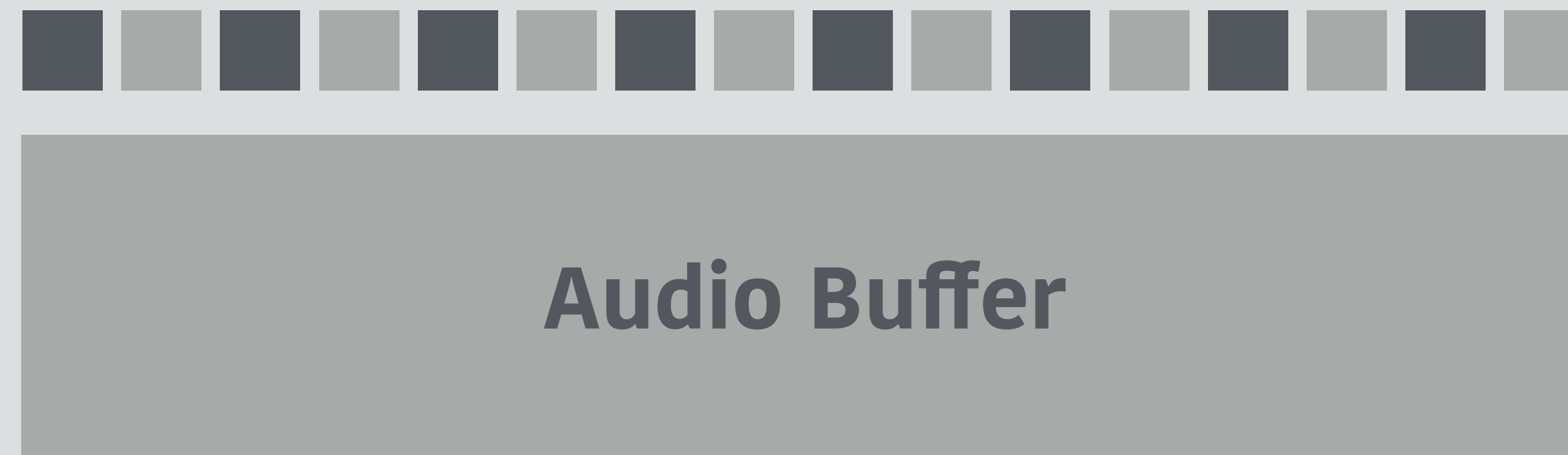








Playing **digital sound** involves **filling an audio buffer** with **audio data** on demand.



# Playing audio with SDL2



**SDL\_mixer** (the easy way).

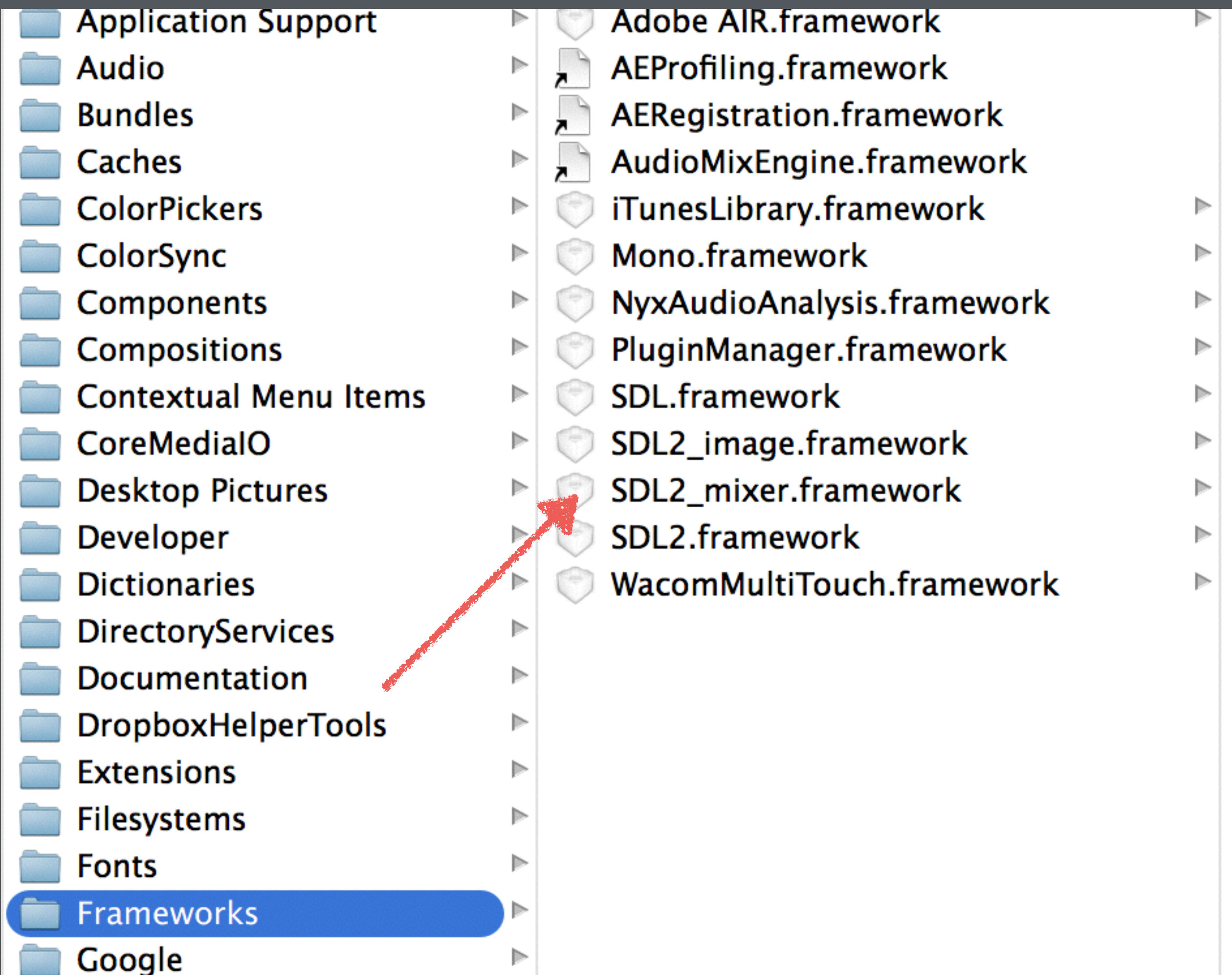
On Mac

Go to the folder:

/Library/Frameworks

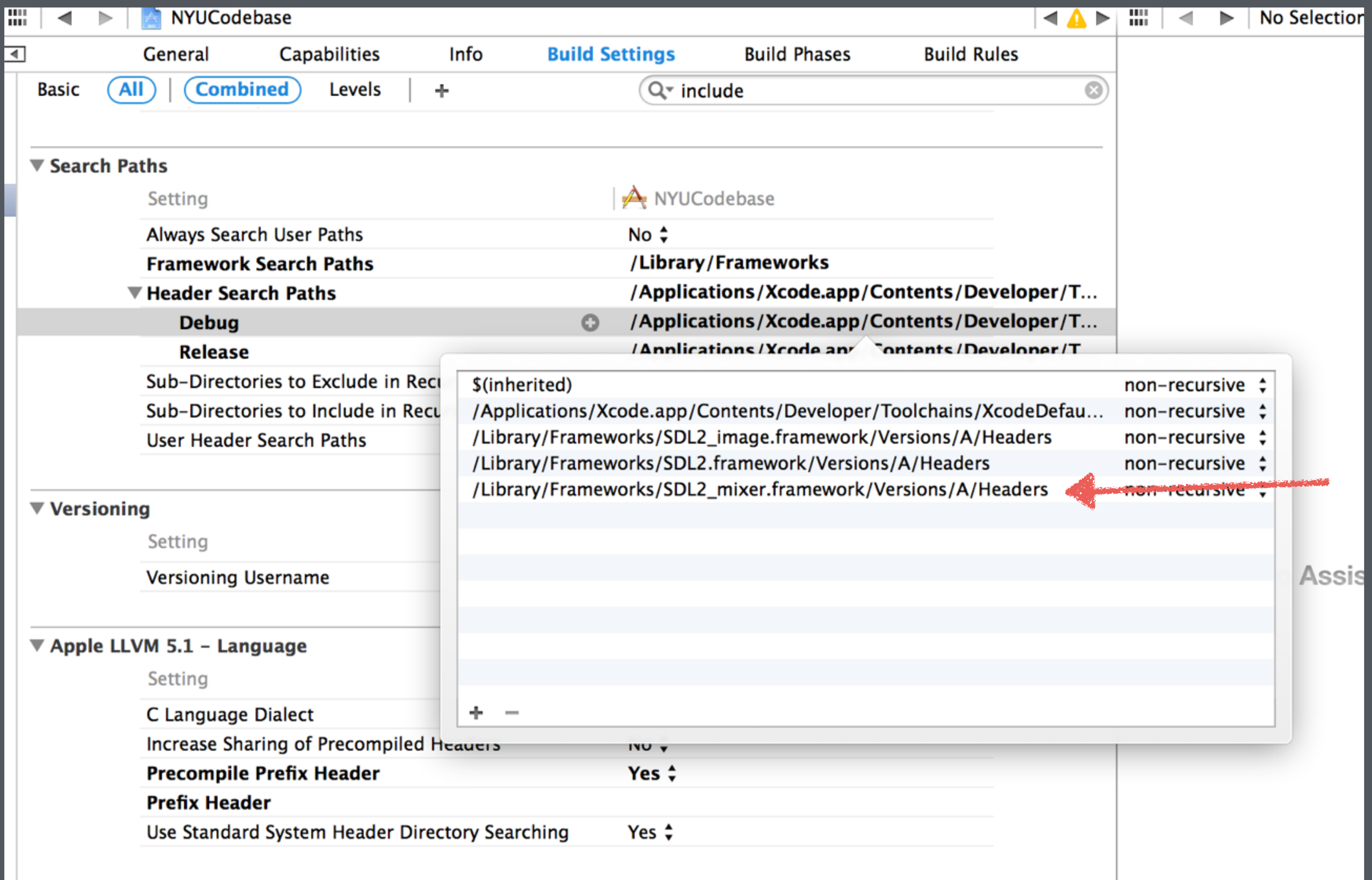
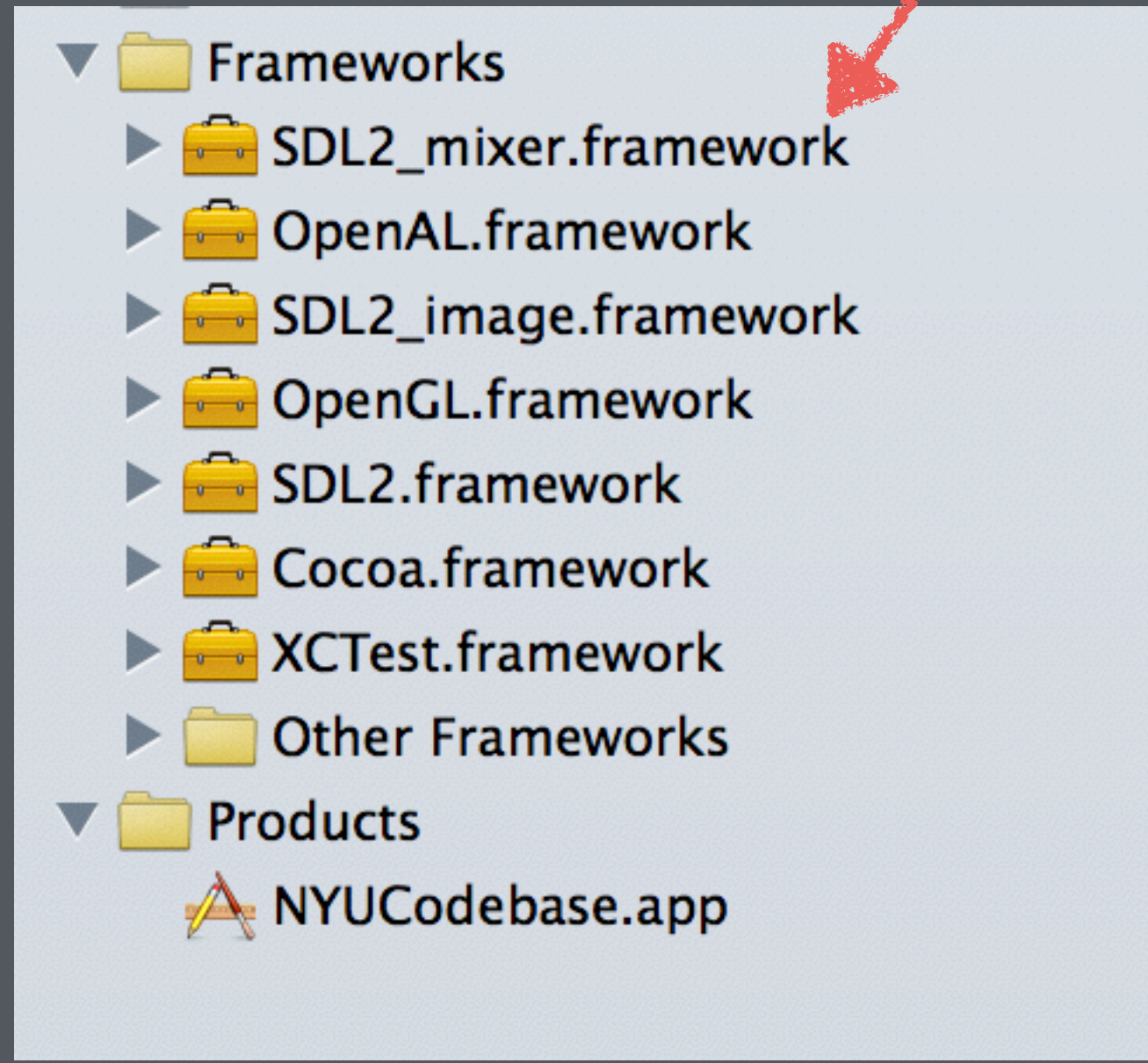
Cancel

Go



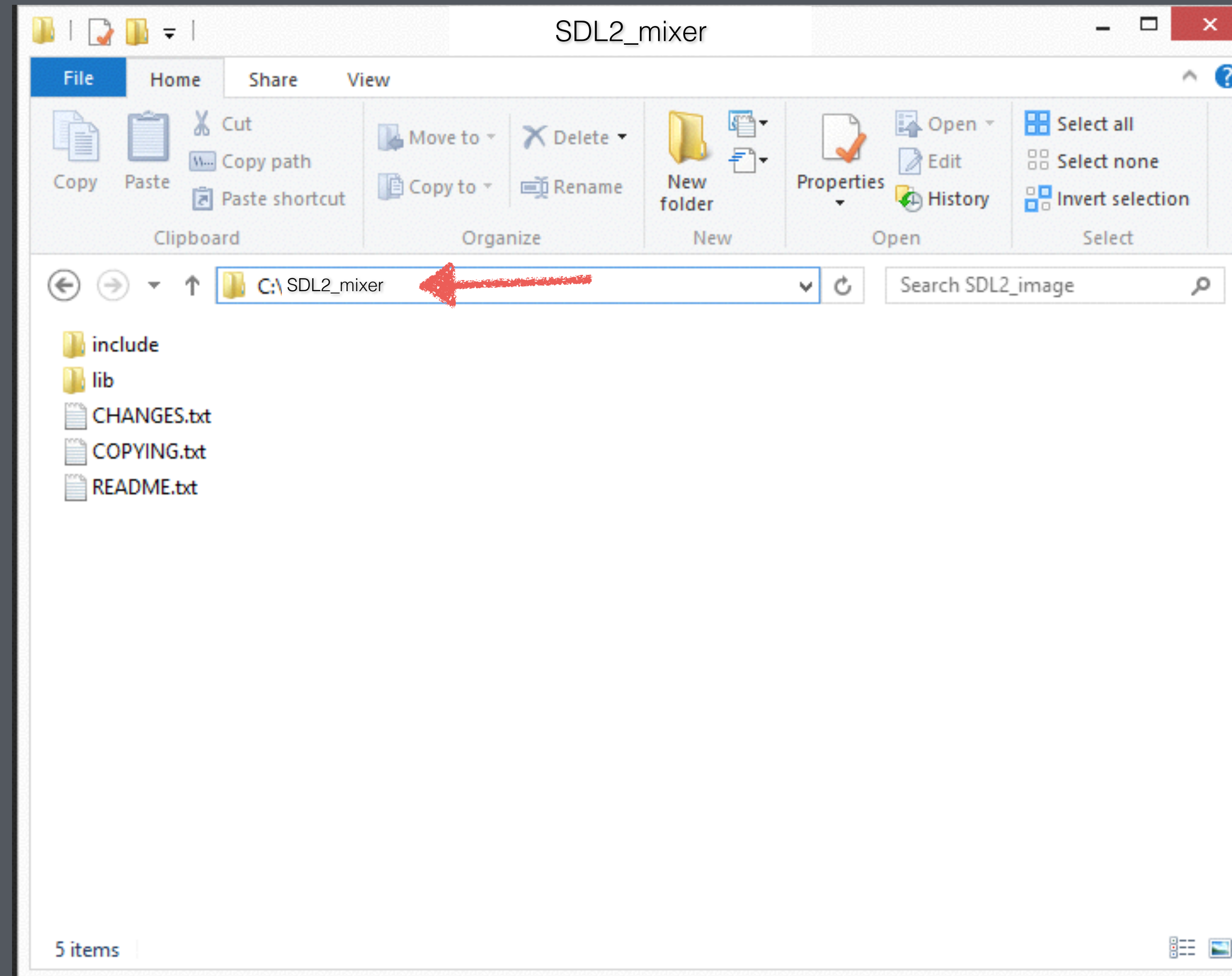


DRAG FROM FINDER

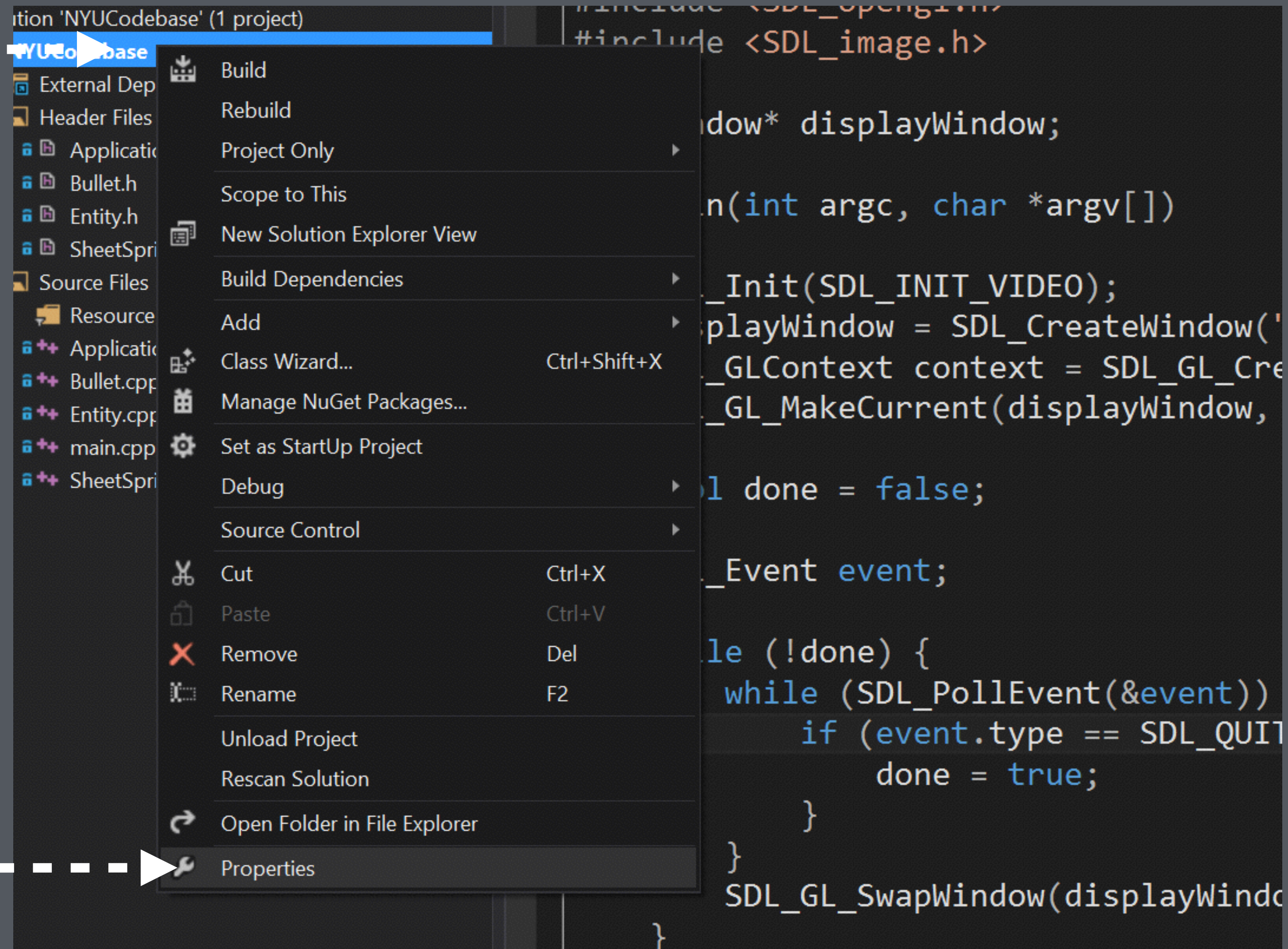




On Windows



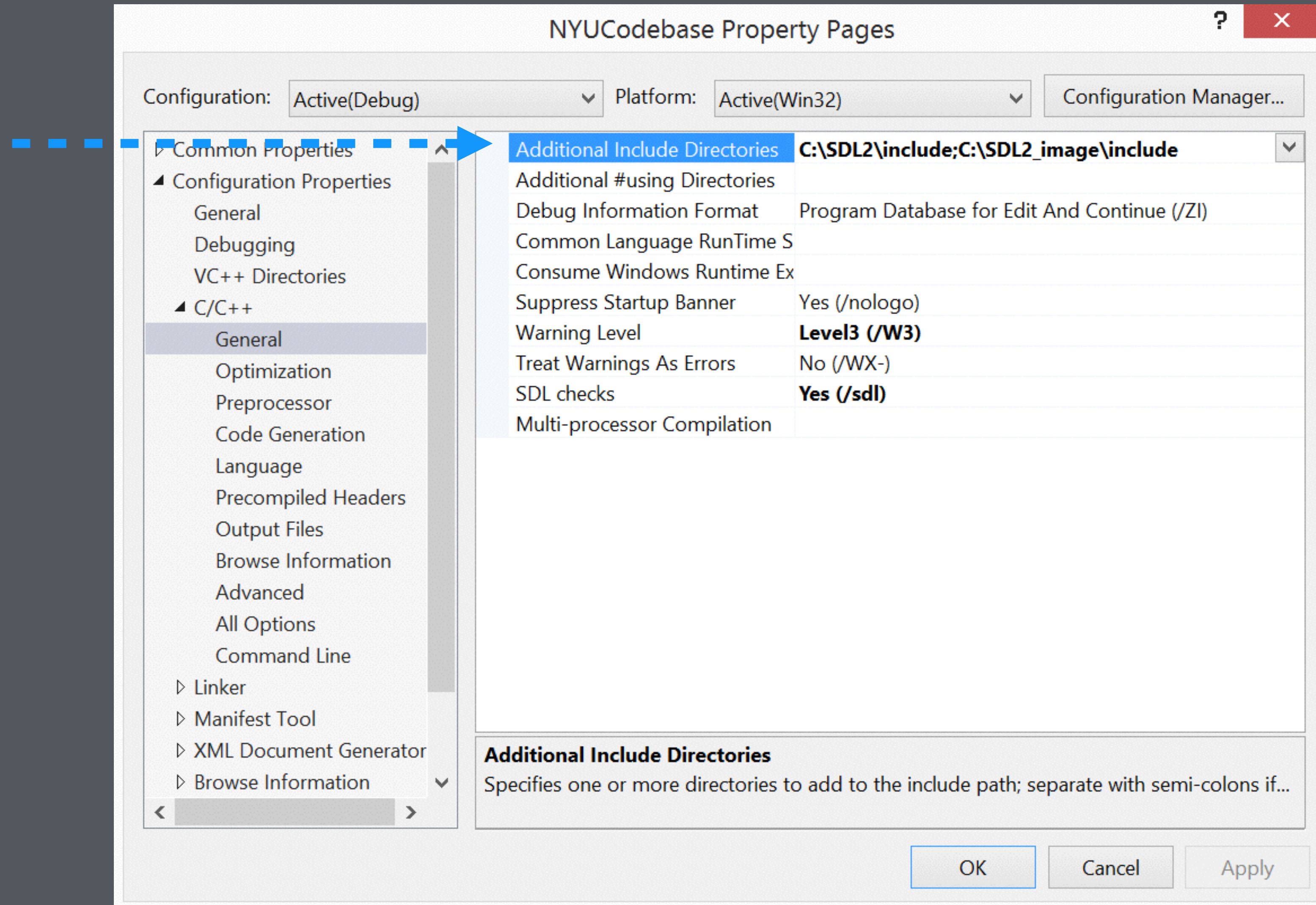
# RIGHT CLICK PROJECT



# GO TO PROPERTIES

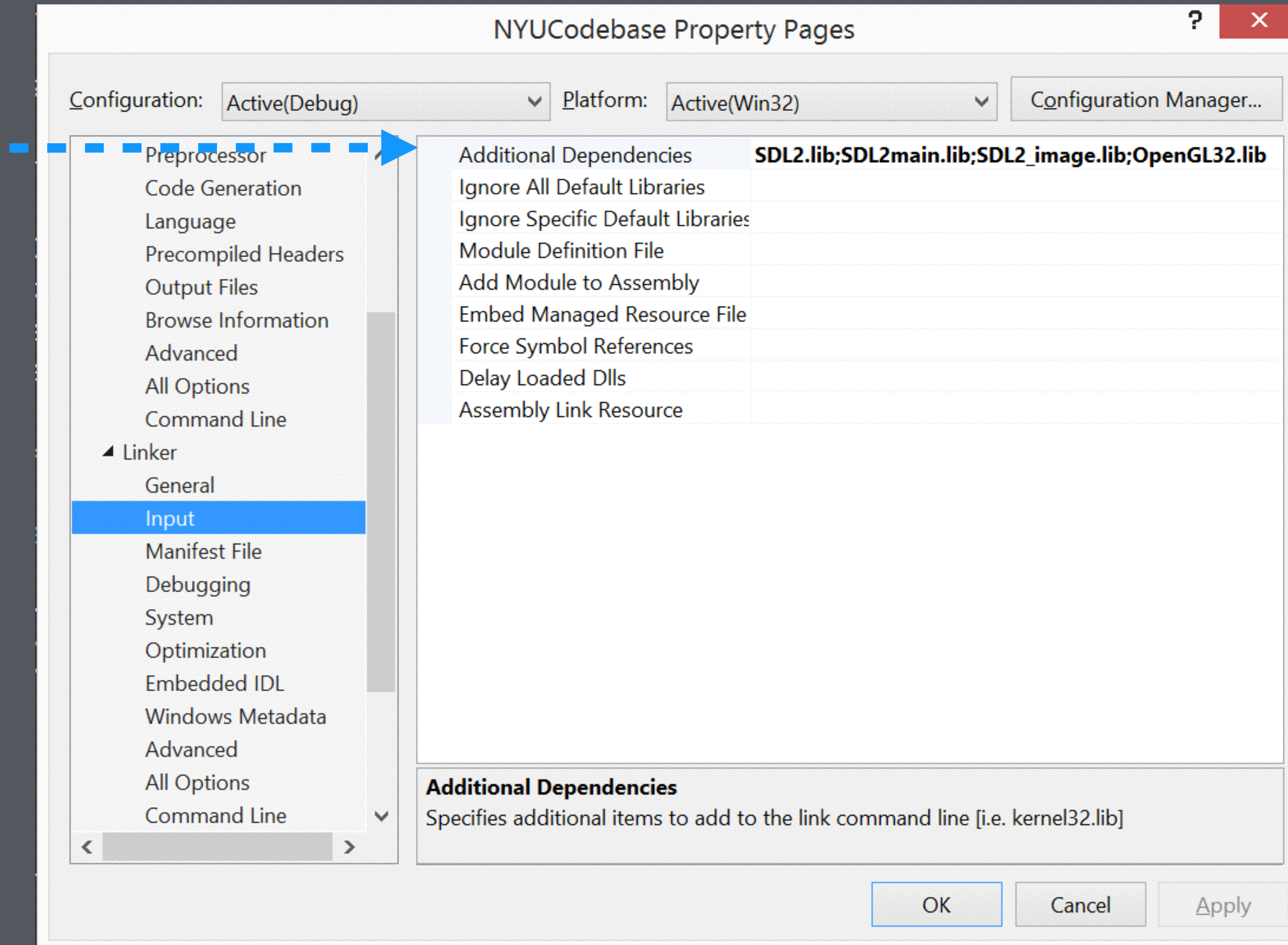


# Add C:\SDL\_mixer\include to additional library directories

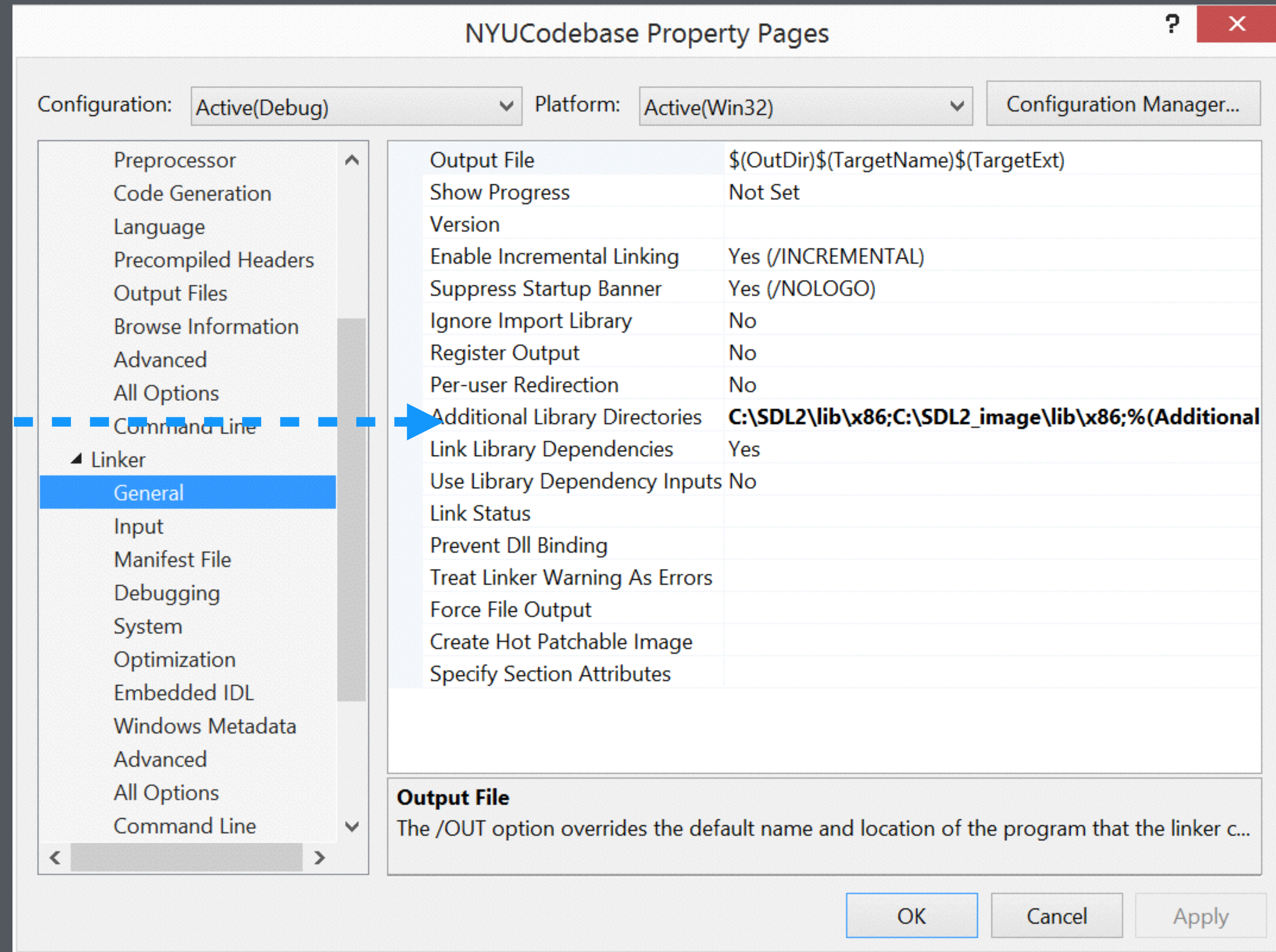




# ADD SDL2\_mixer.lib to additional dependencies



# Add C:\SDL\_mixer\lib\x86 to additional library directories



Do the same thing for both Release and Debug configurations.

Copy all DLL files from C:\SDL\_mixer\lib\x86 to where all the other DLL files are in your project.

Initializing `SDL_mixer`



Include **SDL\_mixer** header.

```
#include <SDL_mixer.h>
```

```
int Mix_OpenAudio(int frequency, Uint16 format, int channels,  
int chunksize);
```

Initializes SDL\_mixer with **frequency**, **format**, **channel** and **buffer size**.

```
Mix_OpenAudio( 44100, MIX_DEFAULT_FORMAT, 2, 4096 );
```



Loading and playing sounds.

# Loading a sound.

```
Mix_Chunk *someSound;  
someSound = Mix_LoadWAV("some_sound.wav");
```

# Playing a sound.

```
int Mix_PlayChannel(int channel, Mix_Chunk *chunk, int loops);
```

Plays a sound on specified channel. You can pass -1 for **channel** to use the first available channel. Loops can be **-1 to loop forever**.

```
Mix_PlayChannel( -1, someSound, 0);
```

Loading and playing **music**.

# Loading music.

```
Mix_Music *music;  
music = Mix_LoadMUS( "music.mp3" );
```

# Playing music.

```
int Mix_PlayMusic(Mix_Music *music, int loops);
```

Plays specified music. Loops can be -1 to loop forever.

```
Mix_PlayMusic(music, -1);
```

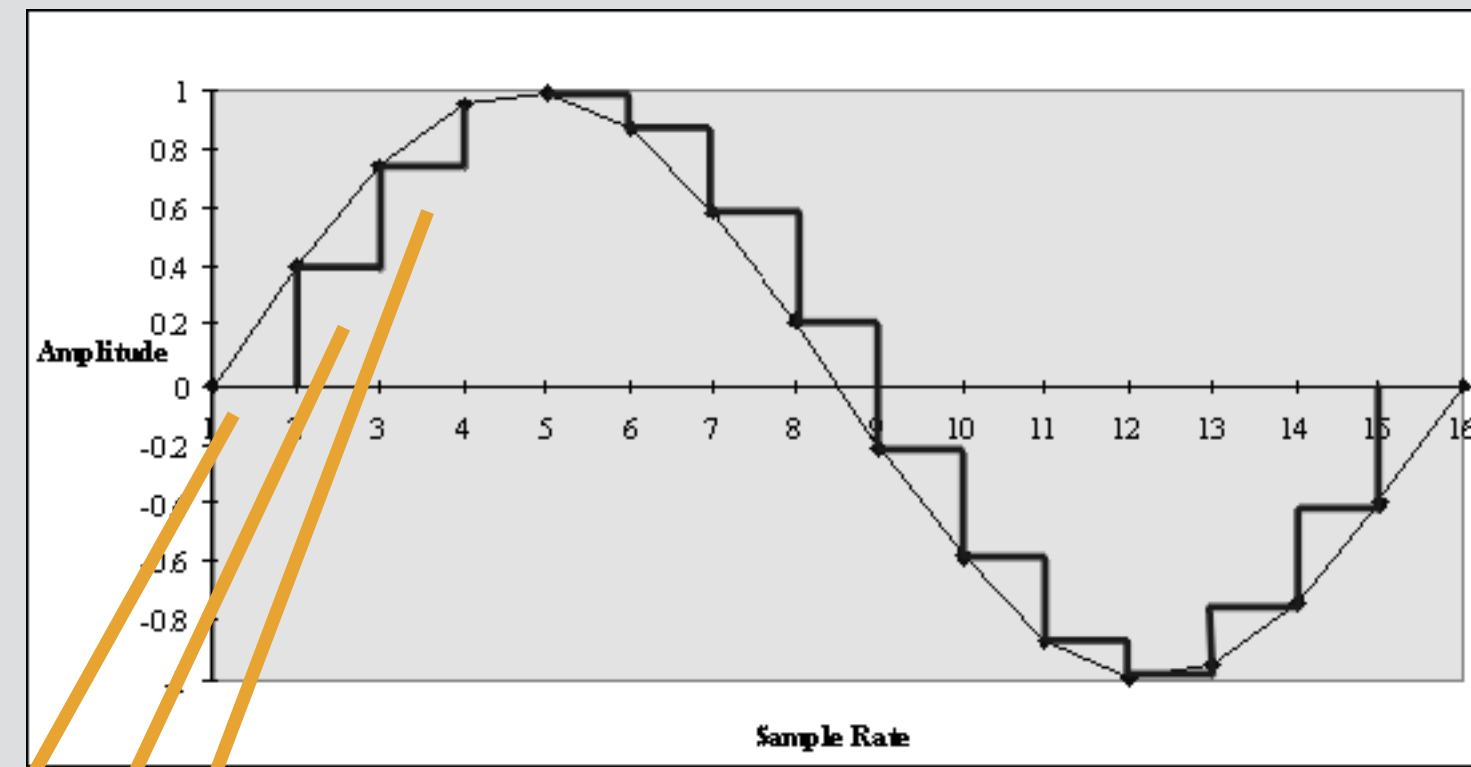
Cleaning up.



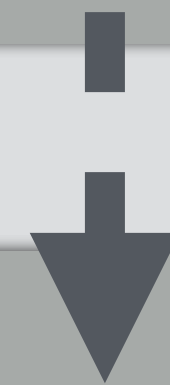
Need to **clean up** music and sounds on quit.

```
DemoApp::~DemoApp() {  
  
    Mix_FreeChunk(someSound);  
    Mix_FreeMusic(music);  
  
    SDL_Quit();  
}
```

Pure audio buffer (the hard way).



**Audio Buffer**



**Operating system audio mixer**

Need to add **SDL\_INIT\_AUDIO** to **SDL\_Init** flags.

```
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
```

# Define your audio callback function.

```
void myAudioCallback(void *userdata, Uint8 *stream, int len) {  
}
```

Open an audio device with your callback and desired settings.

```
SDL_AudioSpec deviceSpec;  
deviceSpec.freq = 44100; // sampling rate (samples a second)  
deviceSpec.format = AUDIO_F32; // audio format  
deviceSpec.channels = 1; // how many channels (1 = mono, 2 = stereo)  
deviceSpec.samples = 512; // audio buffer size in samples (power of 2)  
deviceSpec.callback = myAudioCallback; // our callback function  
  
// open new audio device with our requested settings  
SDL_AudioDeviceID dev = SDL_OpenAudioDevice(NULL, 0, &deviceSpec, 0,  
SDL_AUDIO_ALLOW_FORMAT_CHANGE);  
SDL_PauseAudioDevice(dev, 0); // start playback on this device
```



The audio callback function will be **called by SDL every time it needs us to refill the audio buffer.**

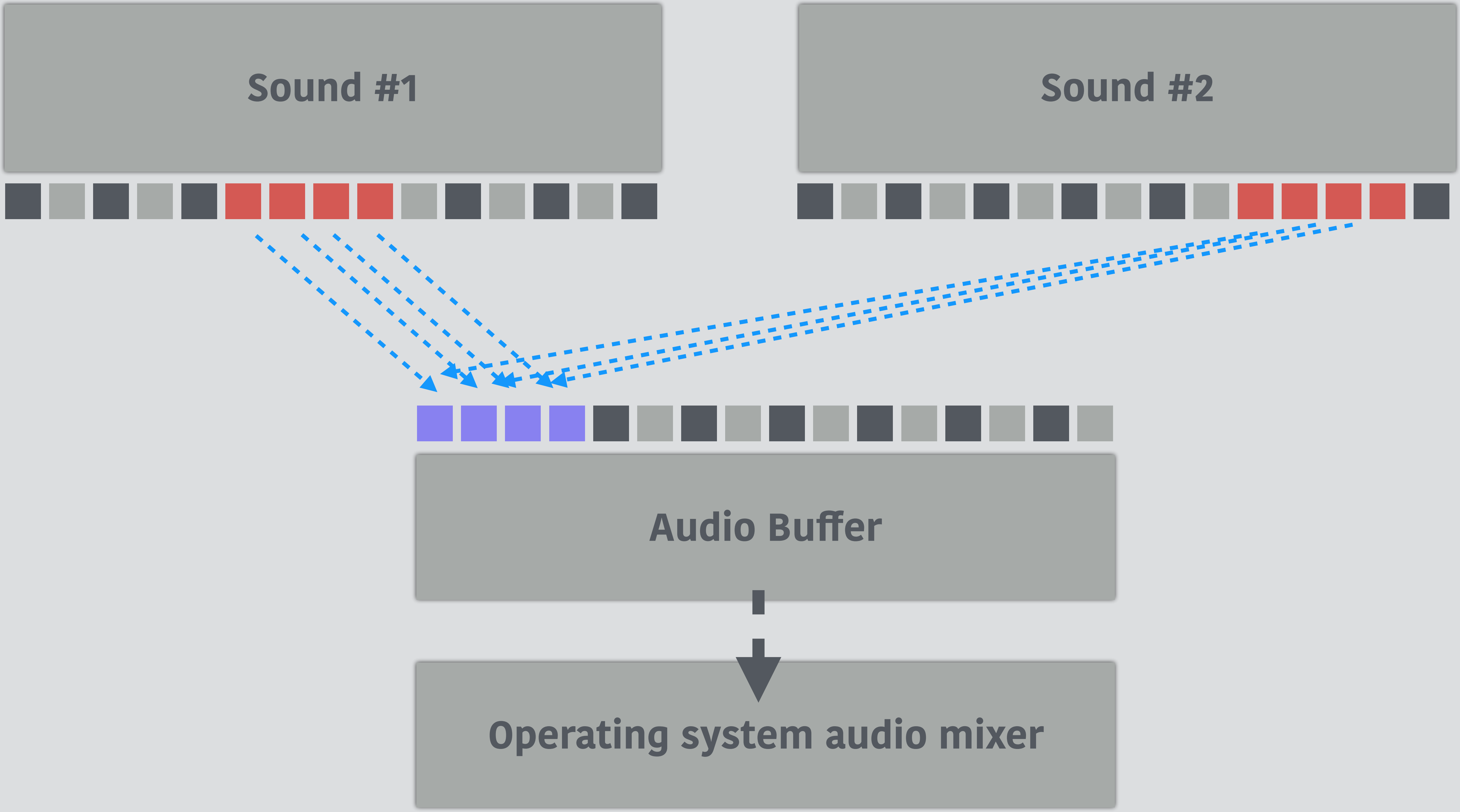
Here is an example of an audio callback function that's generating a 440Hz tone at 44100 sample frequency in a one channel float buffer.

```
unsigned int numSamples = 0;

float getAudioForTime(long numSamples) {
    double elapsed = ((double)numSamples)/44100.0;
    return sin(elapsed * 2.0 * M_PI * 440.0);
}

void myAudioCallback(void *userdata, Uint8 *stream, int len) {
    for(int i=0; i < len/4; i++) {
        ((float*)stream)[i] = getAudioForTime(numSamples);
        numSamples++;
    }
}
```

Writing a simple **mixer**.



```
class MixerSound {
public:
    Uint32 offset;
    Uint32 length;
    Uint8 *buffer;
    float volume;
    SDL_AudioFormat format;
    bool loaded;
    bool playing;
    bool loop;
};

class DemoApp {
public:
    std::vector<MixerSound> mixSounds;
};
```

Create a class for storing  
loaded sound buffers  
and some basic sound  
information.

Our app will keep these  
in a vector.

```

int DemoApp::loadSound(const char *soundFile) {

    Uint8 *buffer;
    SDL_AudioSpec spec;
    Uint32 bufferSize;

    if(SDL_LoadWAV(soundFile, &spec, &buffer, &bufferSize) == NULL) {
        return -1;
    }

    SDL_AudioCVT cvt;
    SDL_BuildAudioCVT(&cvt, spec.format, spec.channels, spec.freq,
deviceSpec.format, deviceSpec.channels, deviceSpec.freq);
    cvt.len = bufferSize;
    cvt.buf = new Uint8[bufferSize * cvt.len_mult];
    memcpy(cvt.buf, buffer, bufferSize);

    SDL_ConvertAudio(&cvt);
    SDL_FreeWAV(buffer);

    MixerSound sound;
    sound.buffer = cvt.buf;
    sound.length = cvt.len_cvt;
    sound.loaded = true;
    sound.offset = 0;
    sound.format = deviceSpec.format;
    sound.volume = 1.0;
    sound.playing = false;
    mixerSounds.push_back(sound);

    return mixerSounds.size()-1;
}

```

**SDL\_LoadWAV** can only  
load **.wav** files.

```
someSound = loadSound("some_wave_file.wav");
```



```

float mixSamples(float A, float B) {
    if (A < 0 && B < 0 ) {
        return (A + B) - (A * B)/-1.0;
    } else if (A > 0 && B > 0 ) {
        return (A + B) - (A * B)/1.0;
    } else {
        return A + B;
    }
}

void DemoApp::appAudioCallback(void *userdata, Uint8 *stream, int len) {
    ClassDemoApp *app = (DemoApp*) userdata;
    memset(stream, 0, len);

    for(int i=0; i < app->mixerSounds.size(); i++) {
        MixerSound *sound = &app->mixerSounds[i];
        if(sound->loaded && sound->playing) {
            for(int s=0; s < len/4; s++) {
                float *sourceBuffer = (float*) (sound->buffer+sound->offset);
                ((float*)stream)[s] = mixSamples(((float*)stream)[s], (sourceBuffer[s] * sound->volume));
            }
            sound->offset += len;
            if(sound->offset >= sound->length-len) {
                if(sound->loop) {
                    sound->offset = 0;
                } else {
                    sound->playing = false;
                }
            }
        }
    }
}

deviceSpec.callback = DemoApp::appAudioCallback;
deviceSpec.userdata = (void*)this;

```

# Playing a sound.

```
void DemoApp::playSound(int soundIndex, bool loop) {  
    if(soundIndex >= 0 && soundIndex < mixerSounds.size()) {  
        mixerSounds[soundIndex].playing = true;  
        mixerSounds[soundIndex].offset = 0;  
        mixerSounds[soundIndex].loop = loop;  
    }  
}
```

```
playSound(someSound, false);
```

**Sound resources.**

# SFXR

<http://www.superflashbros.net/as3sfxr/>

GENERATOR

PICKUP/COIN

LASER/SHOOT

EXPLOSION

POWERUP

HIT/HURT

JUMP

BLIP/SELECT

MUTATE

RANDOMIZE

BACK

FORWARD

SFBTOM

MANUAL SETTINGS

SQUAREWAVE

SAWTOOTH

SINEWAVE

NOISE

ATTACK TIME

SUSTAIN TIME

SUSTAIN PUNCH

DECAY TIME

START FREQUENCY

MIN FREQUENCY

SLIDE

DELTA SLIDE

VIBRATO DEPTH

VIBRATO SPEED

CHANGE AMOUNT

CHANGE SPEED

SQUARE DUTY

DUTY SWEEP

REPEAT SPEED

PHASER OFFSET

PHASER SWEEP

LP FILTER CUTOFF

LP FILTER CUTOFF SWEEP

LP FILTER RESONANCE

HP FILTER CUTOFF

HP FILTER CUTOFF SWEEP

PLAY ON CHANGE ☐

as3sfxr

CLICK ON LABELS  
TO RESET SLIDERS

COPY/PASTE SETTINGS  
TO SHARE SOUNDS

BASED ON SFXR BY  
TOMAS PETTERSSON

VOLUME

PLAY SOUND

LOAD SOUND

SAVE SOUND

EXPORT .WAV

44100 HZ

16-BIT

# Free Music Archive

<http://freemusicarchive.org/>





# Bonus assignment.

- Add **sound** and **music** to a previous game.