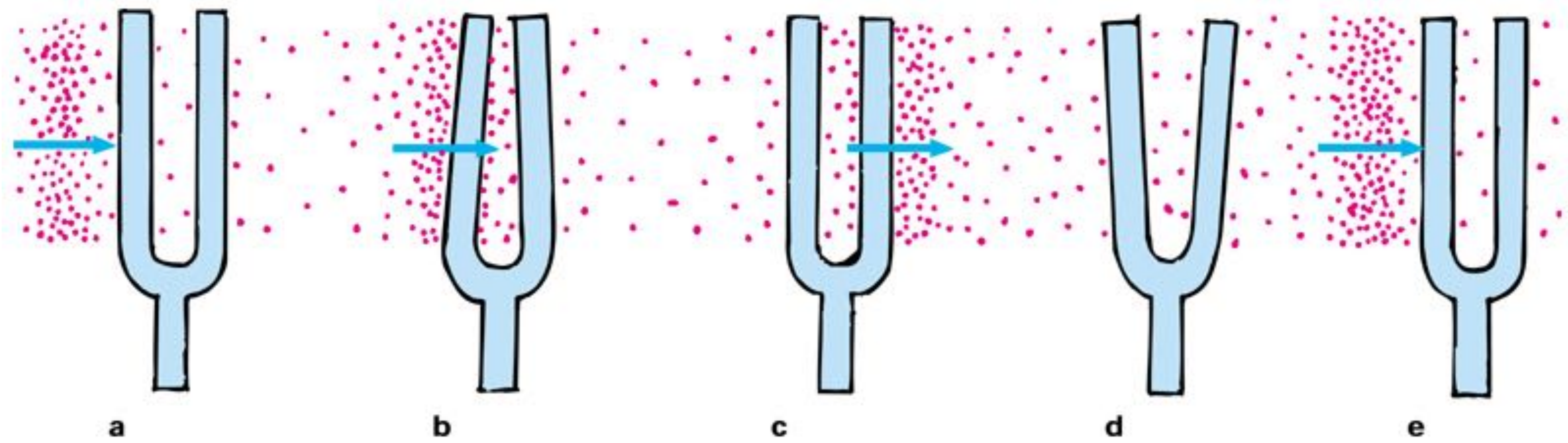
Playing sound.

How sound works.


a
b
c
d
e

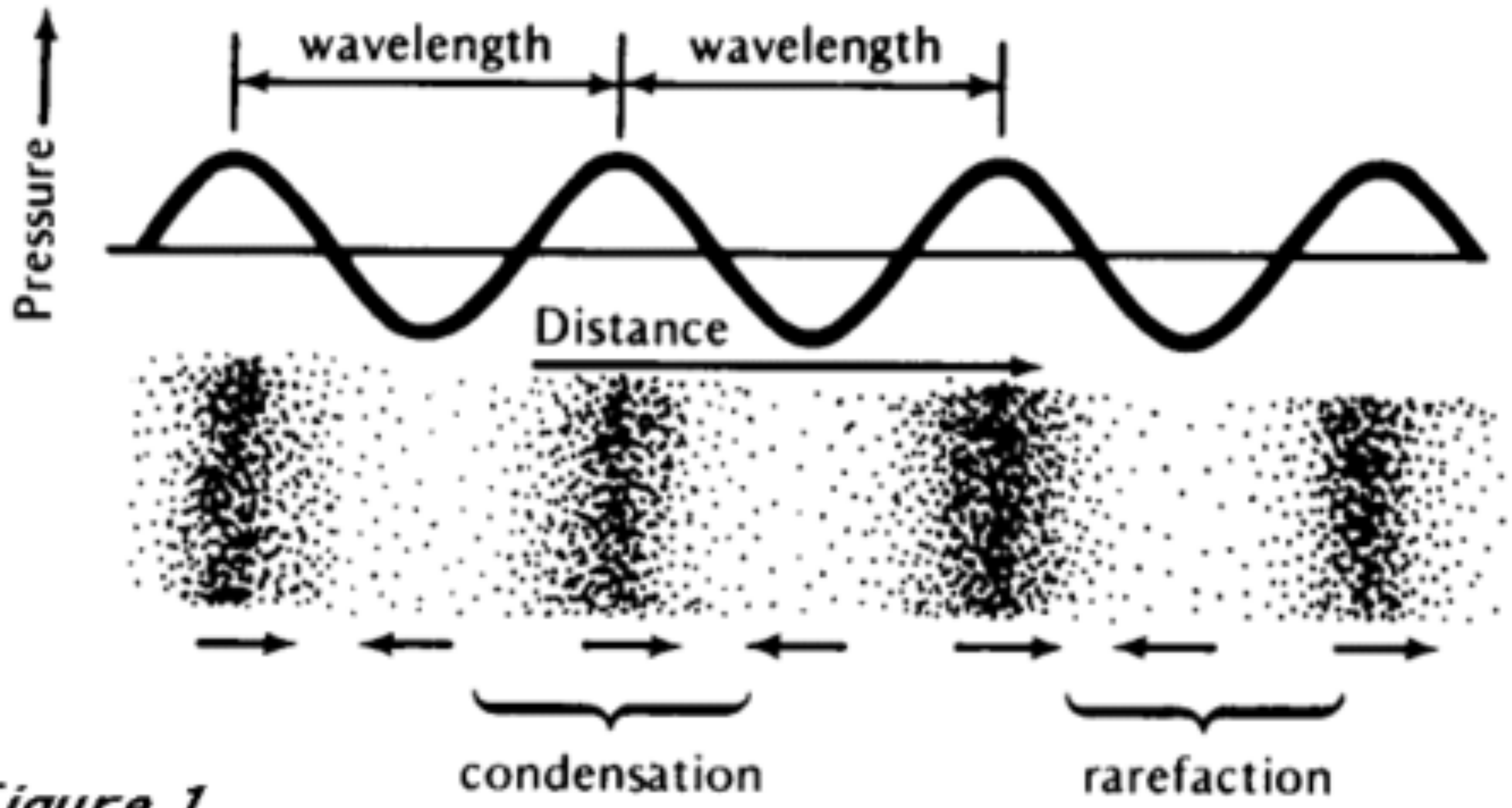
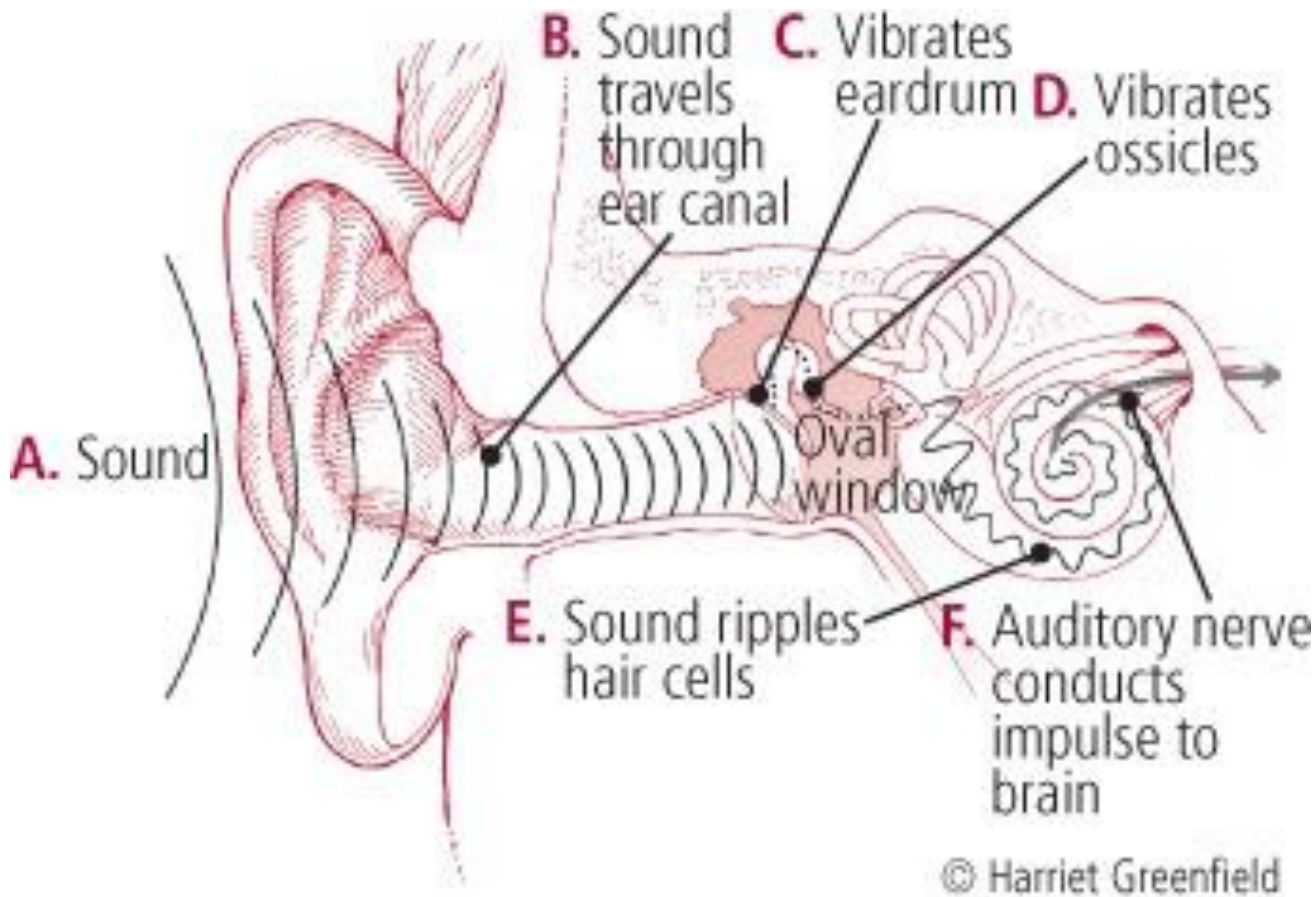
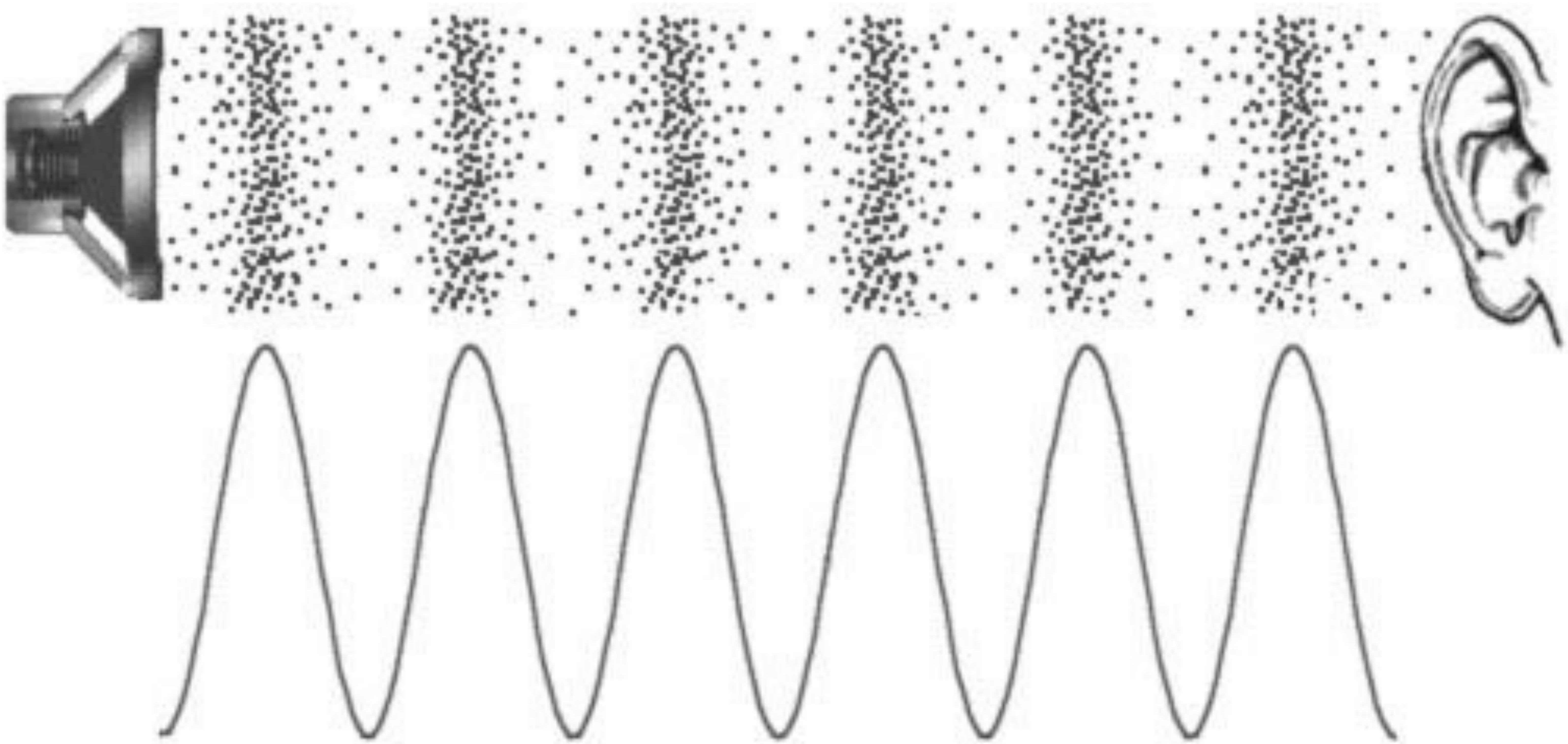
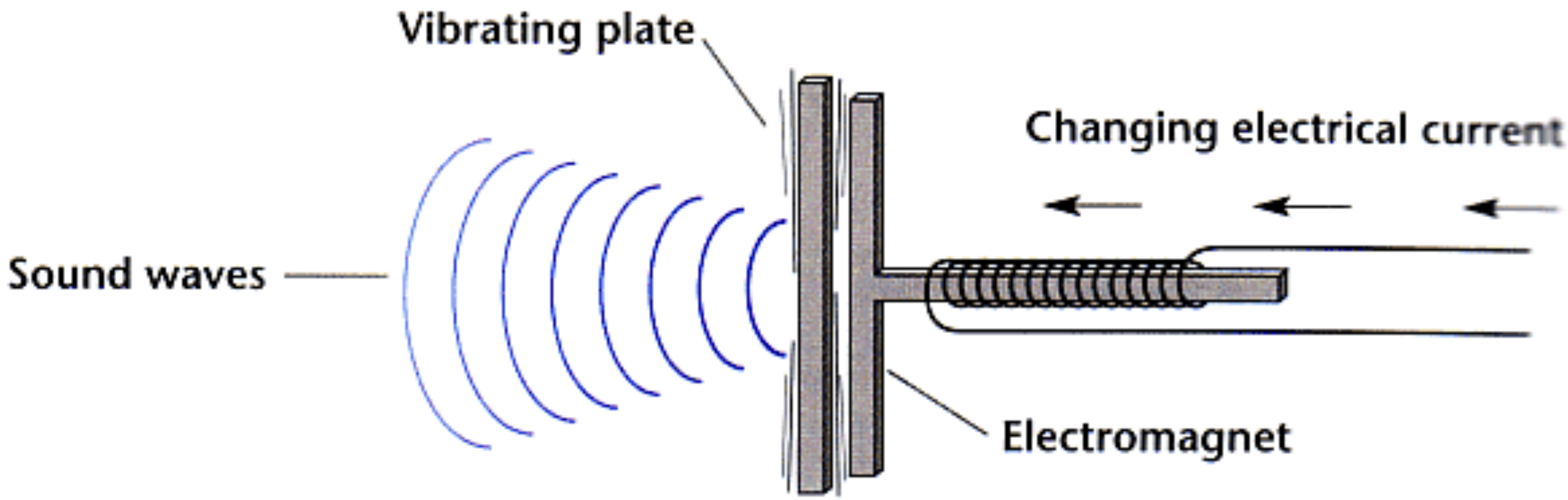

wavelength
wavelength
Pressure
Distance
condensation
rarefaction

Figure 1

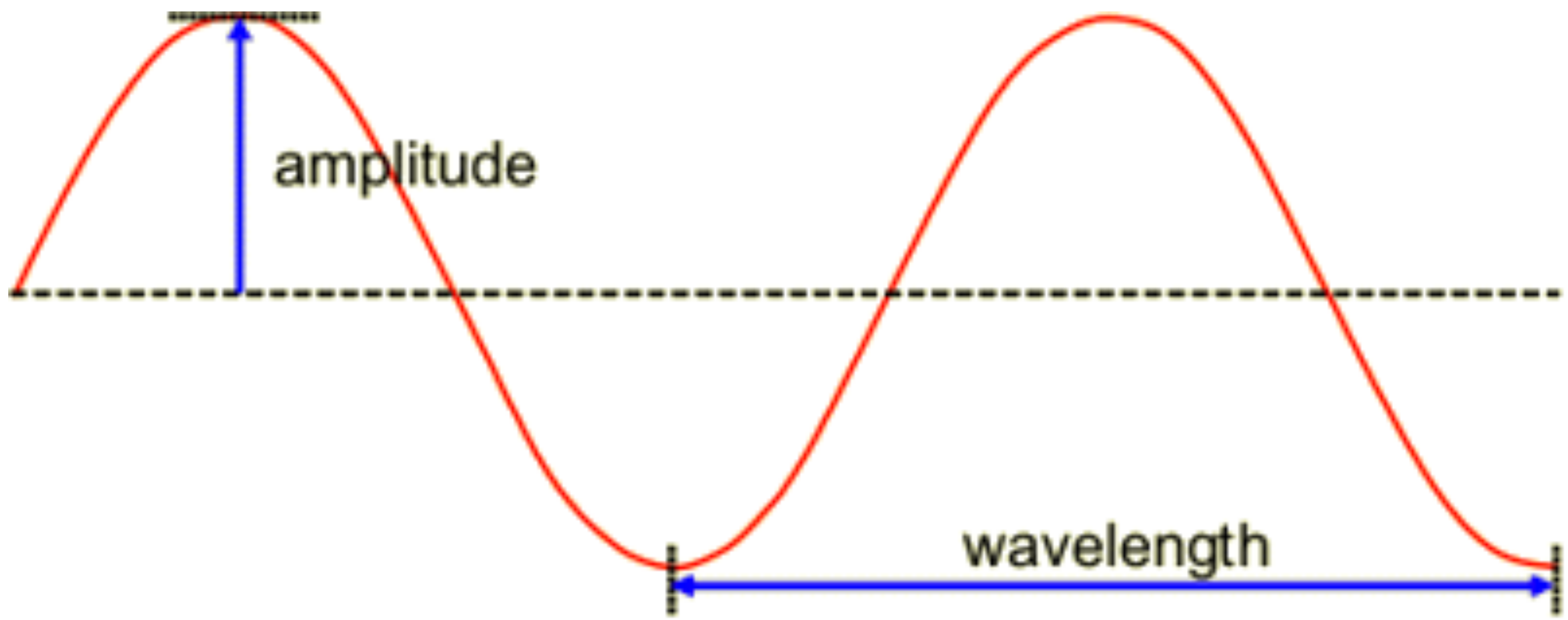


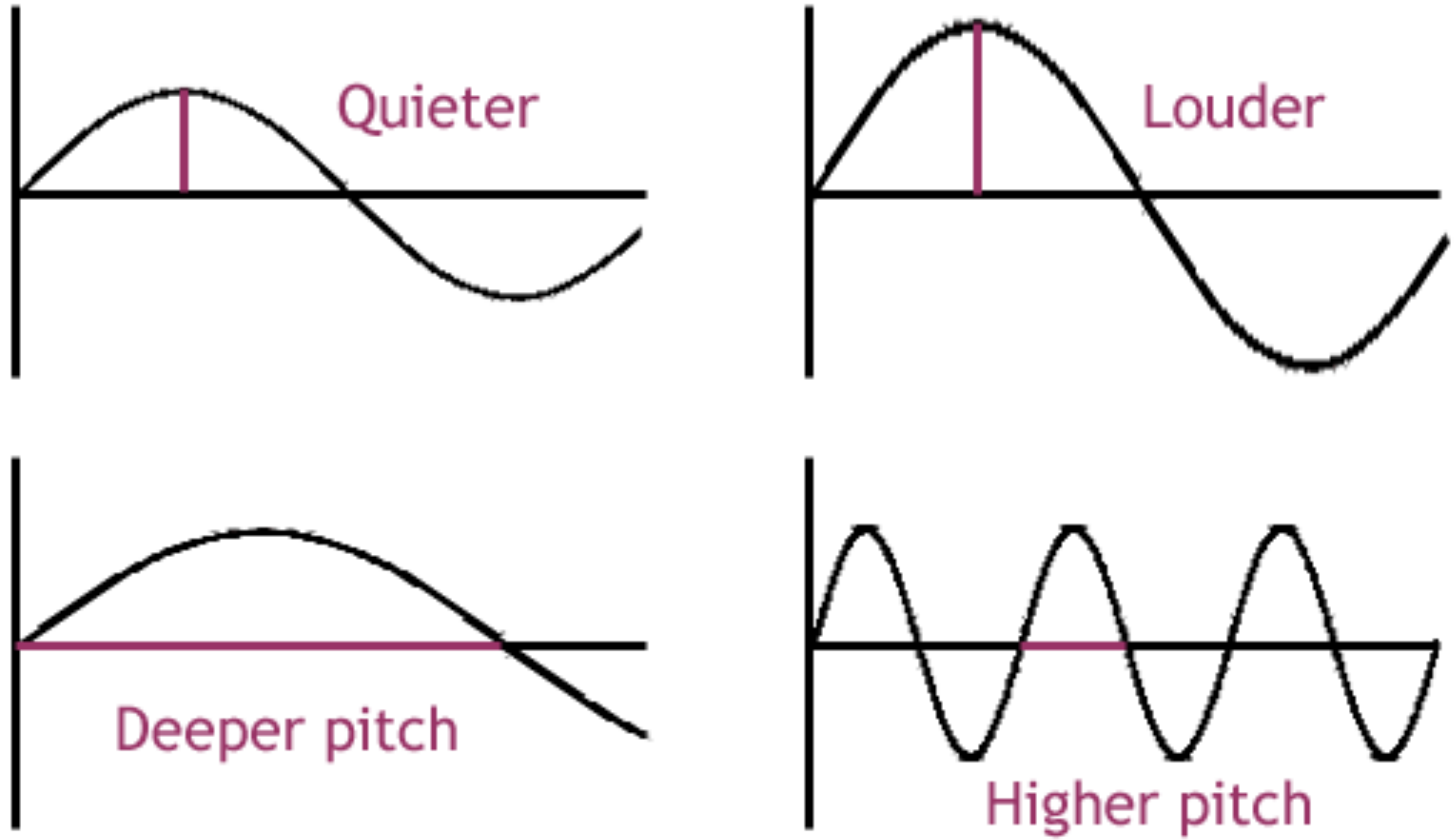
B. Sound travels through ear canal
C. Vibrates eardrum
D. Vibrates ossicles
A. Sound
Oval window
E. Sound ripples hair cells
F. Auditory nerve conducts impulse to brain
© Harriet Greenfield




Vibrating plate
Changing electrical current
Sound waves
Electromagnet

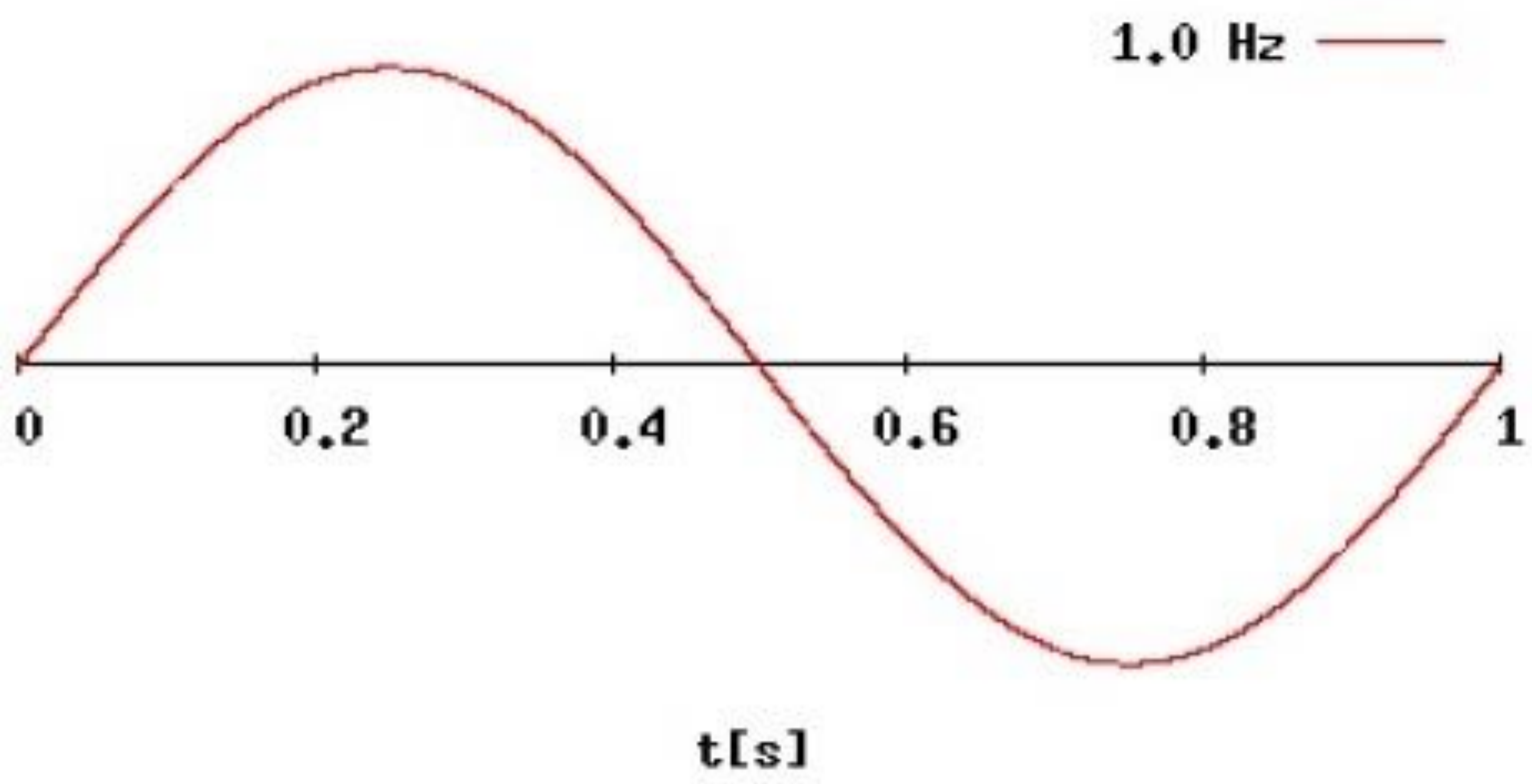
Sound wave properties.

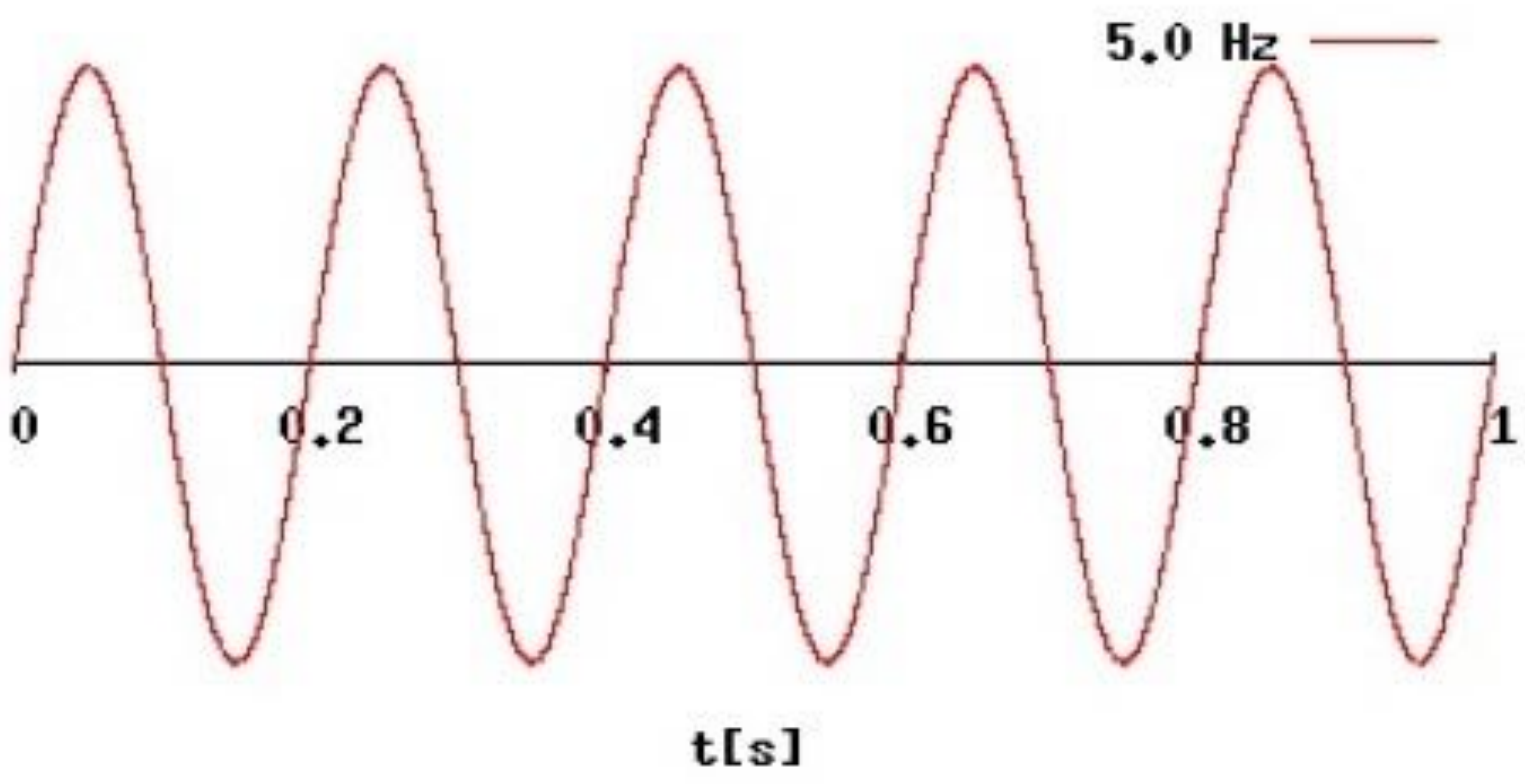

amplitude
wavelength


Quieter
Louder
Deeper pitch
Higher pitch

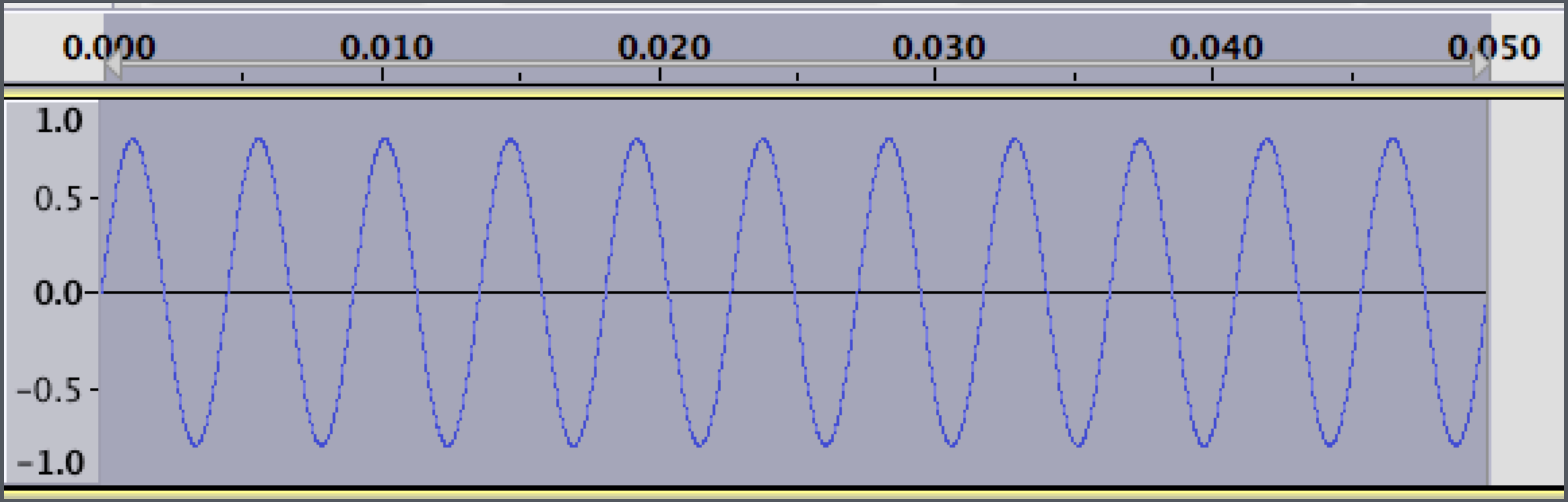
Sound frequency.

hertz (Hz) = 1 cycle per second.

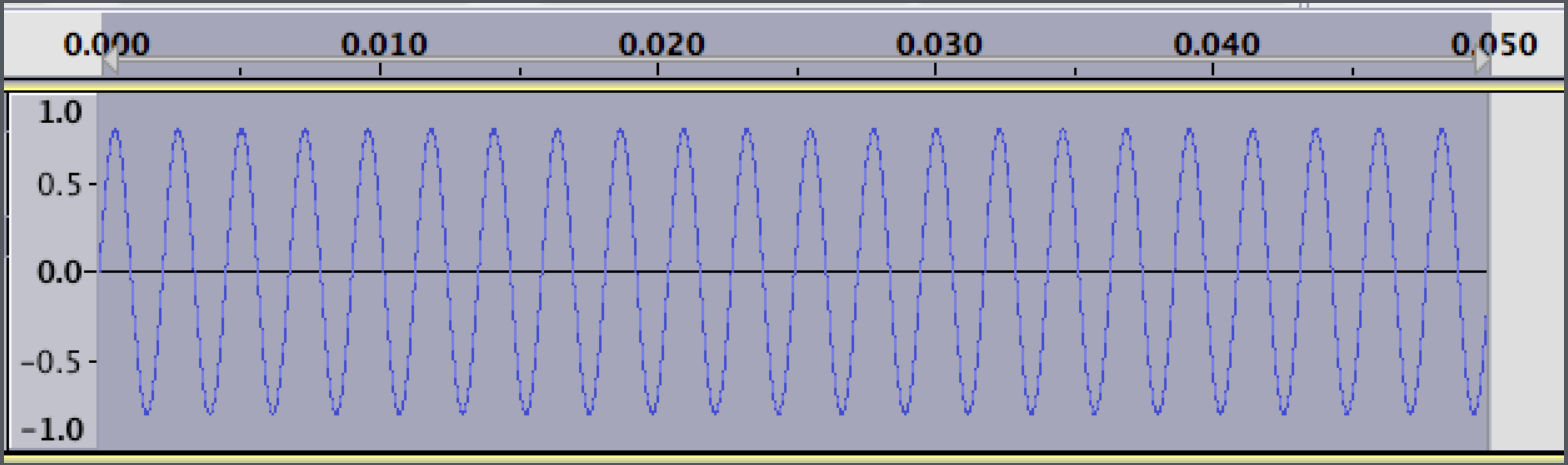




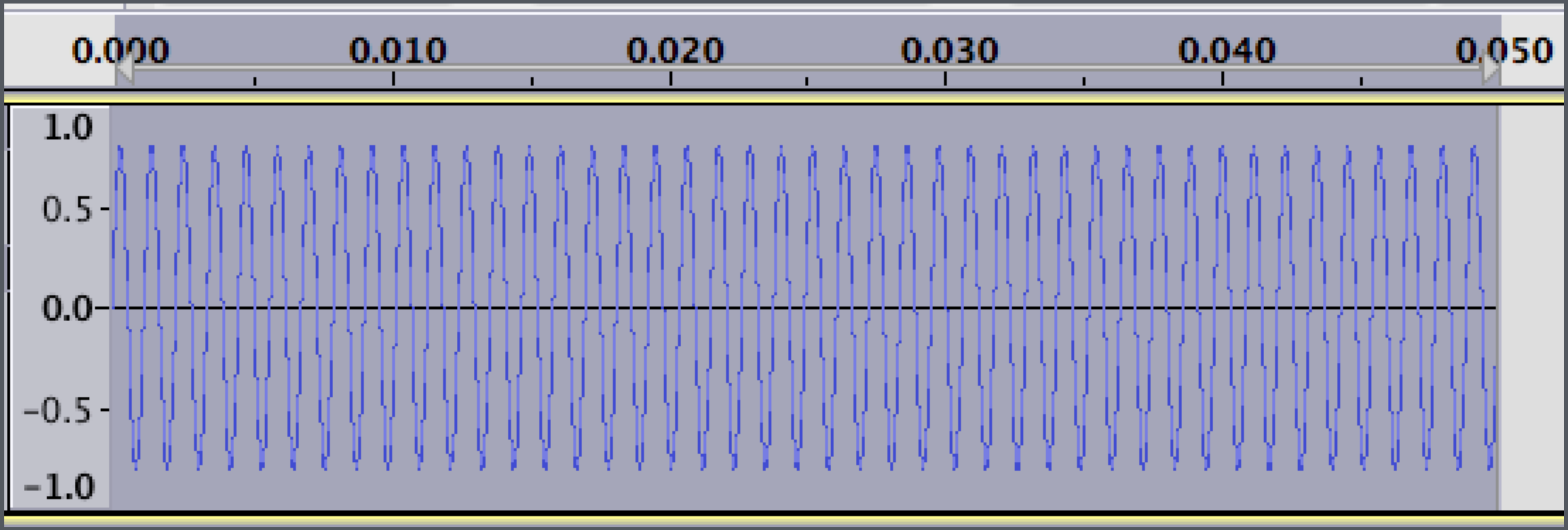
220 Hz



440 Hz



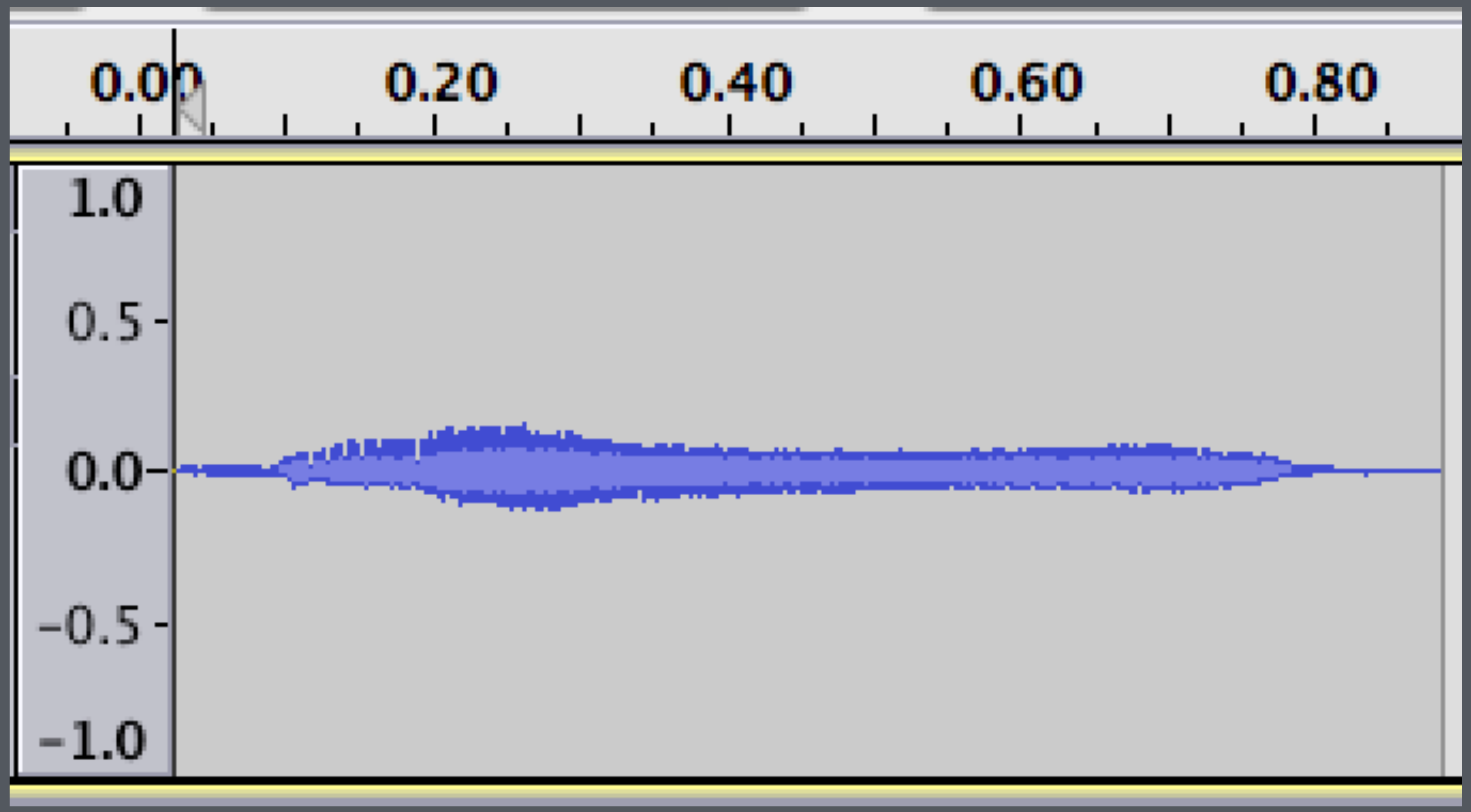
880 Hz



Complex waveforms.

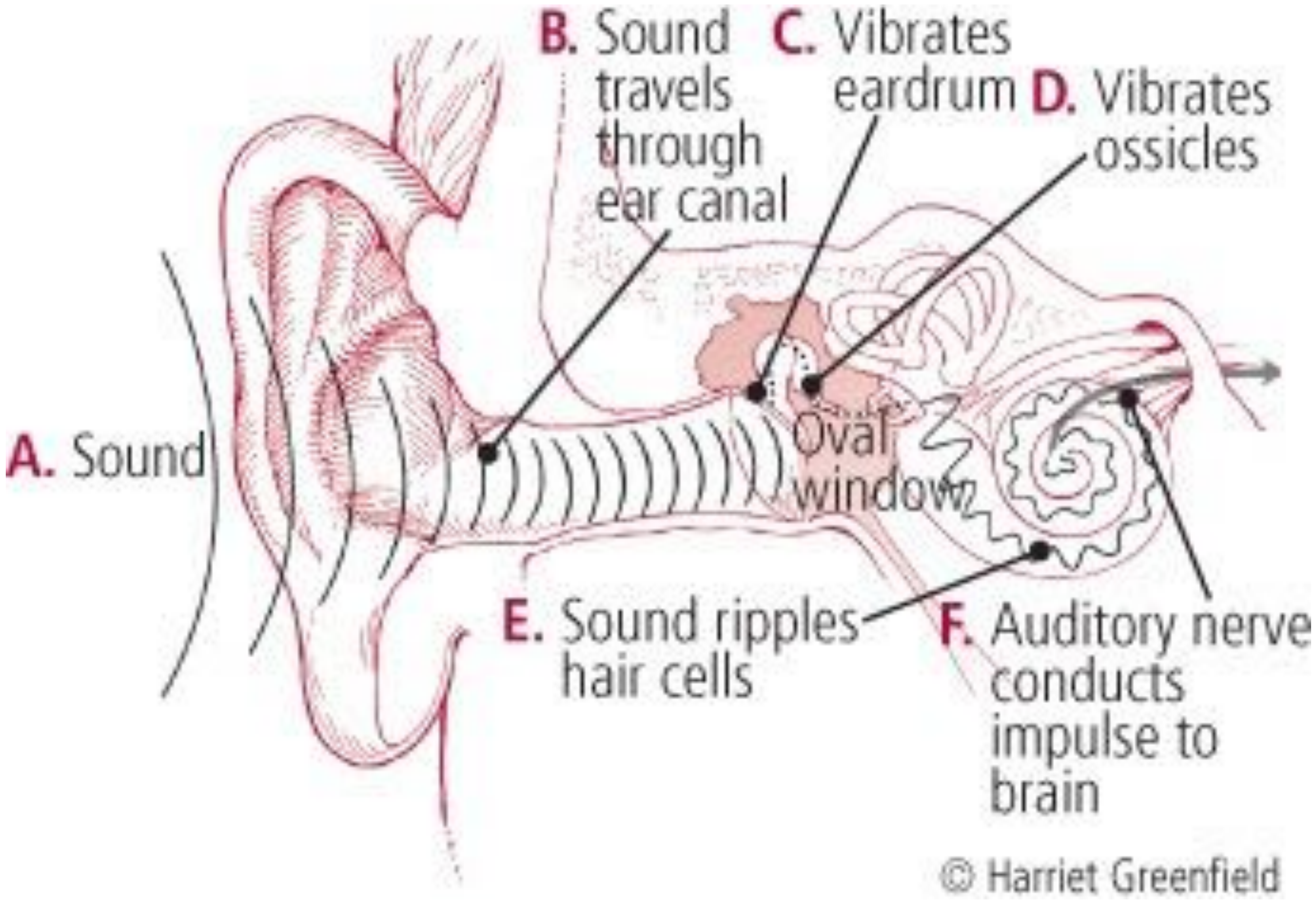
Modulating frequency.

Hello



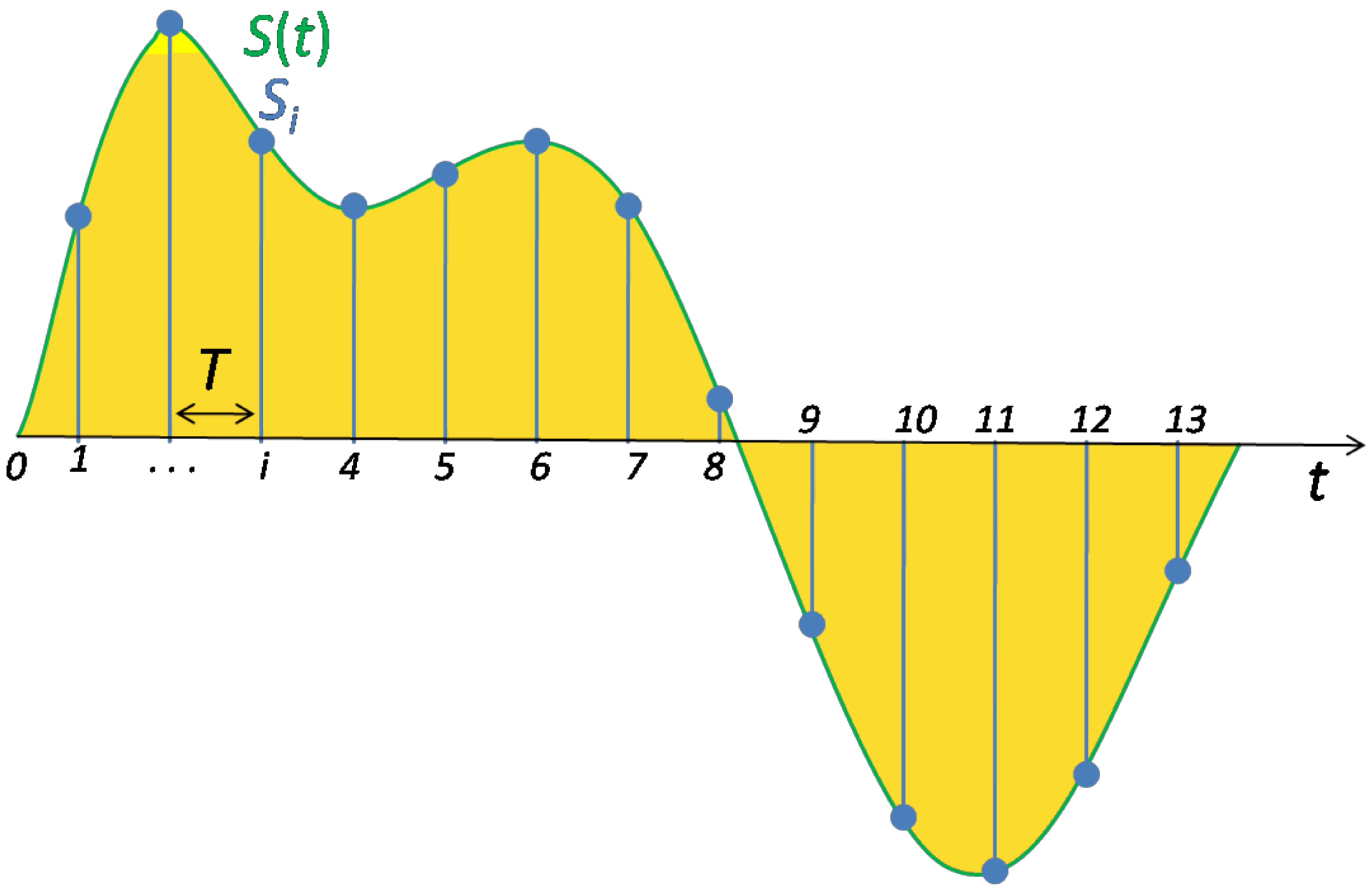

0.00
0.20
0.40
0.60
0.80
1.0
0.5
0.0
-0.5
-1.0
0.30
0.40
0.50
0.60
0.70
0.80
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-1.0


0.00
0.20
0.40
0.60
0.80
1.0
0.5
0.0
-0.5
-1.0
0.30
0.40
0.50
0.60
0.70
0.80
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-1.0
0.080
0.160
0.170
0.180
0.190
0.200
0.210
0.220
0.230
0.240
0.250
0.260
0.270
0.280
1.0
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-1.0


A. Sound
B. Sound travels through ear canal
C. Vibrates eardrum
D. Vibrates ossicles
Oval window
E. Sound ripples hair cells
F. Auditory nerve conducts impulse to brain
© Harriet Greenfield

How digital sound works.

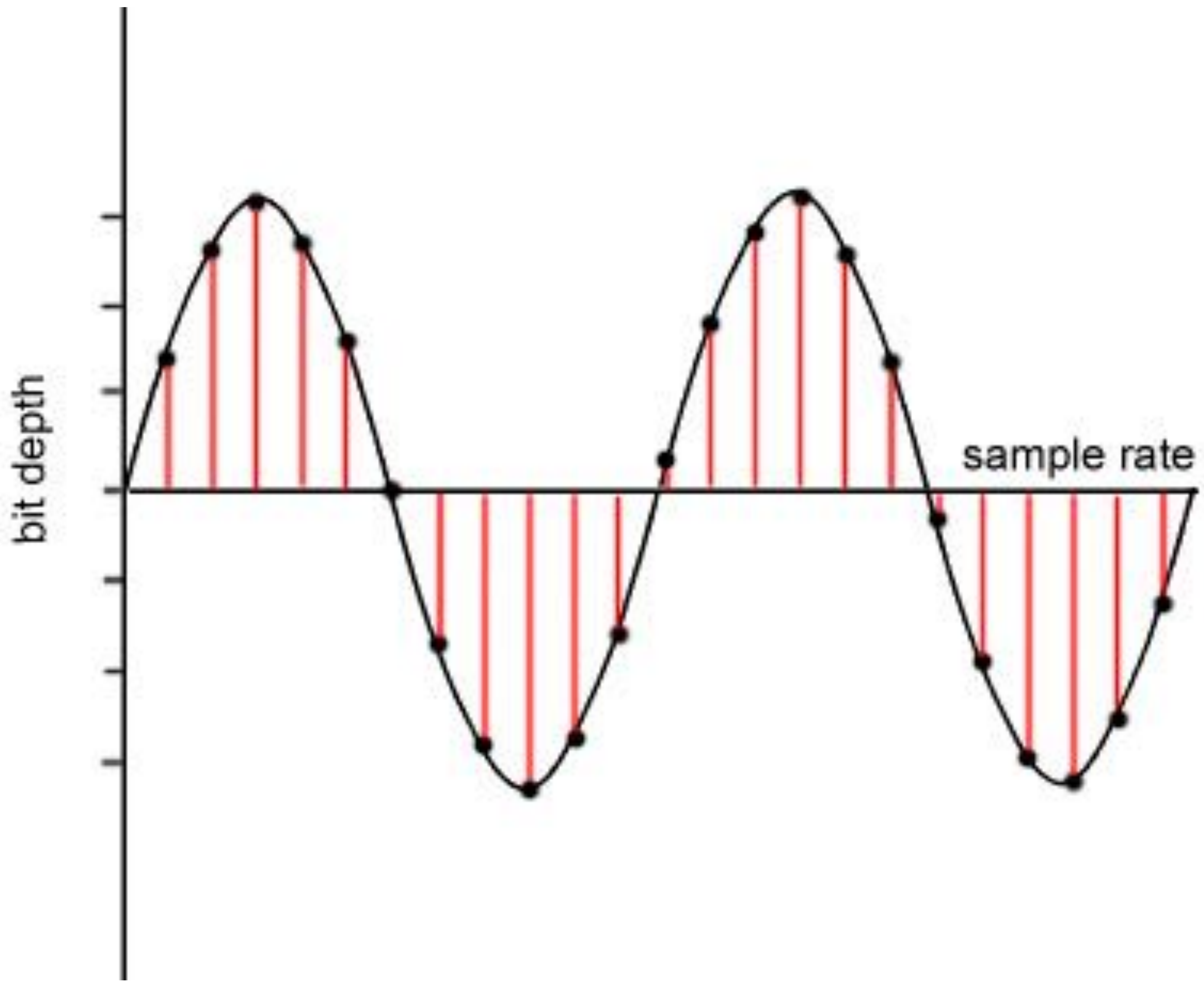
Digital sampling.



S(t)
S_i
T
0
1
. . .
i
4
5
6
7
8
9
10
11
12
13
t

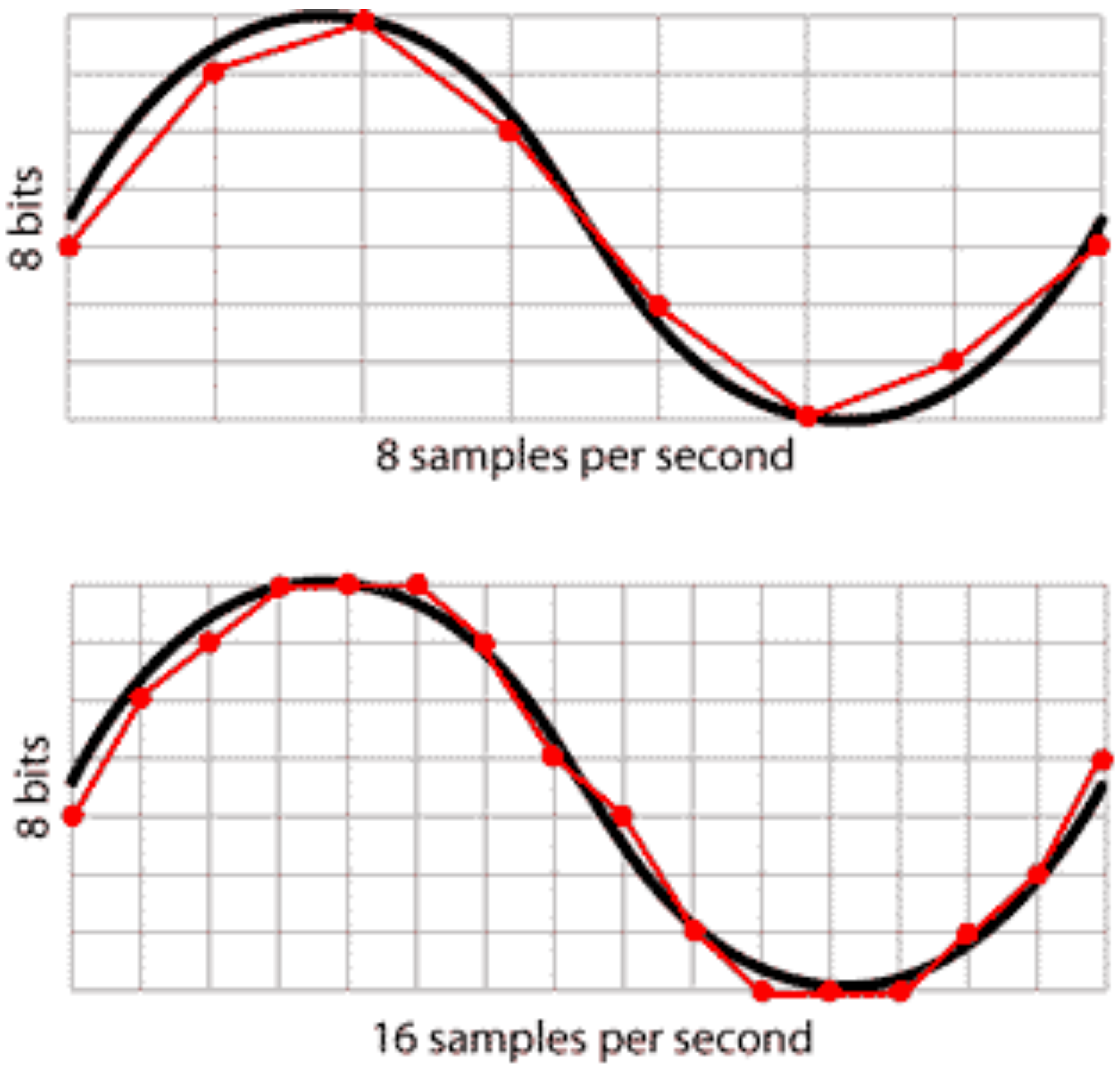
Sampling quality.

Sample rate and bits per sample.


bit depth
sample rate

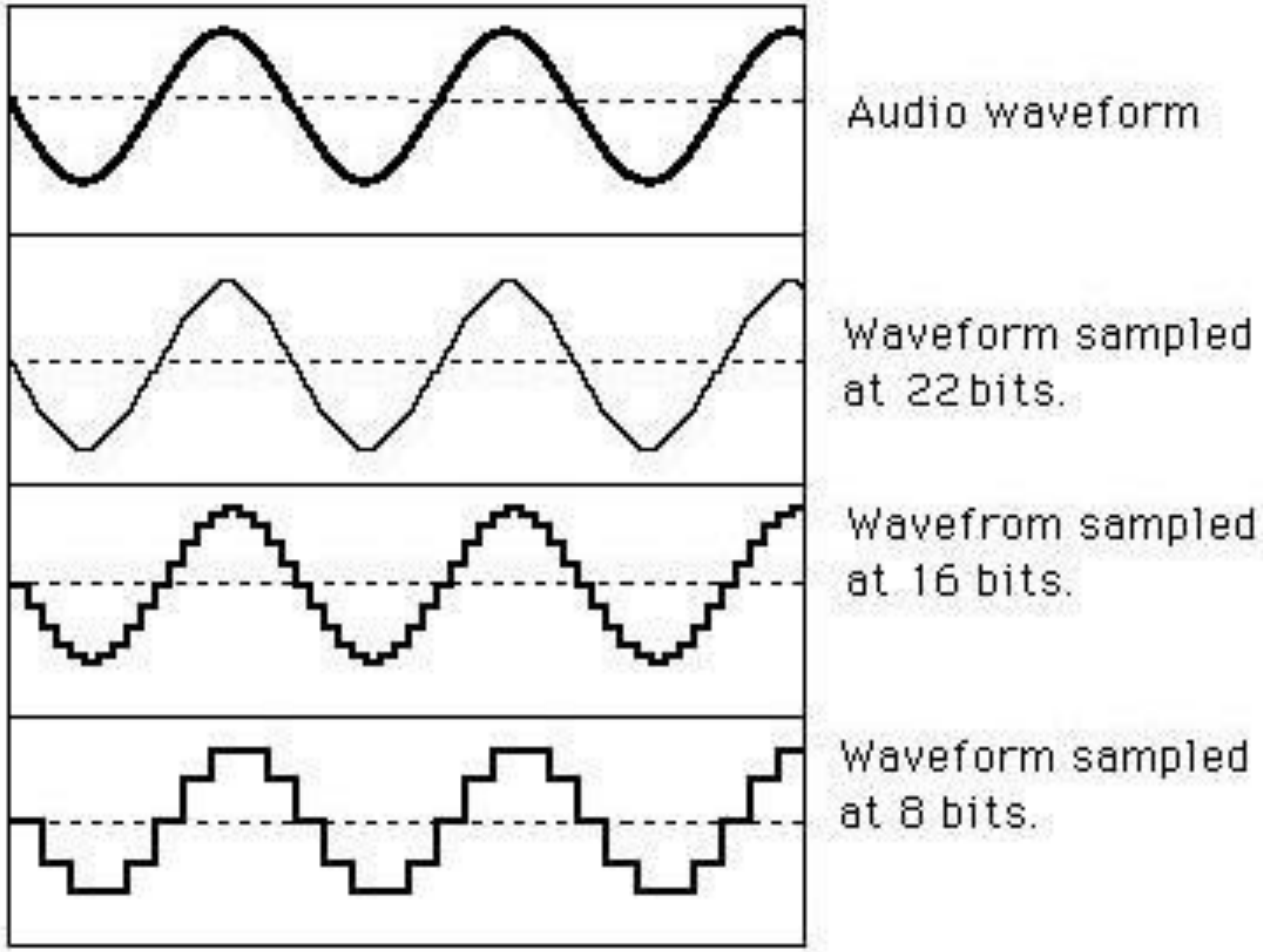
Sample rate.

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



Bits per sample.

Sound quality and bits.



Bitrate.

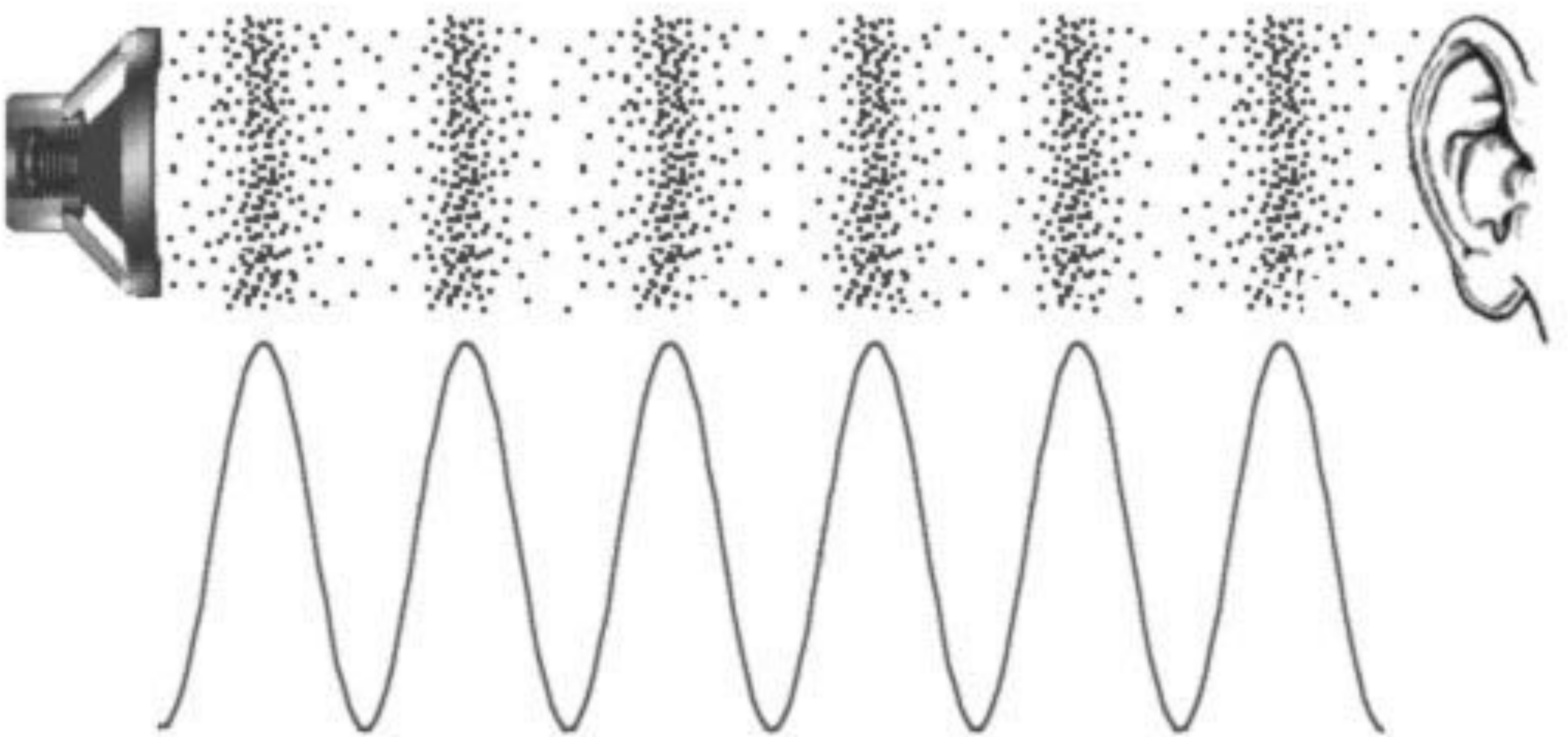
Bitrate.

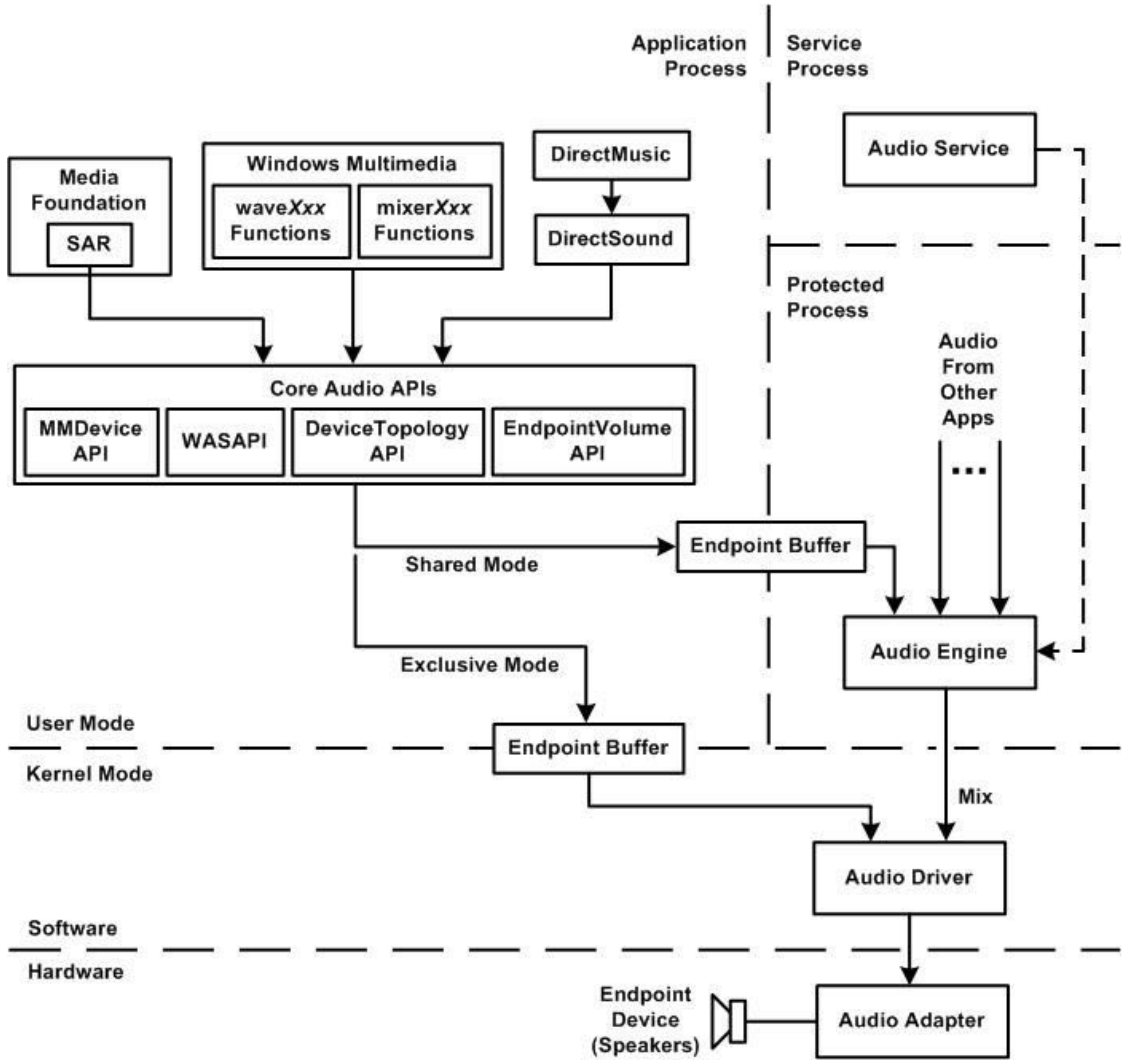
Sampling rate combined with bit depth.

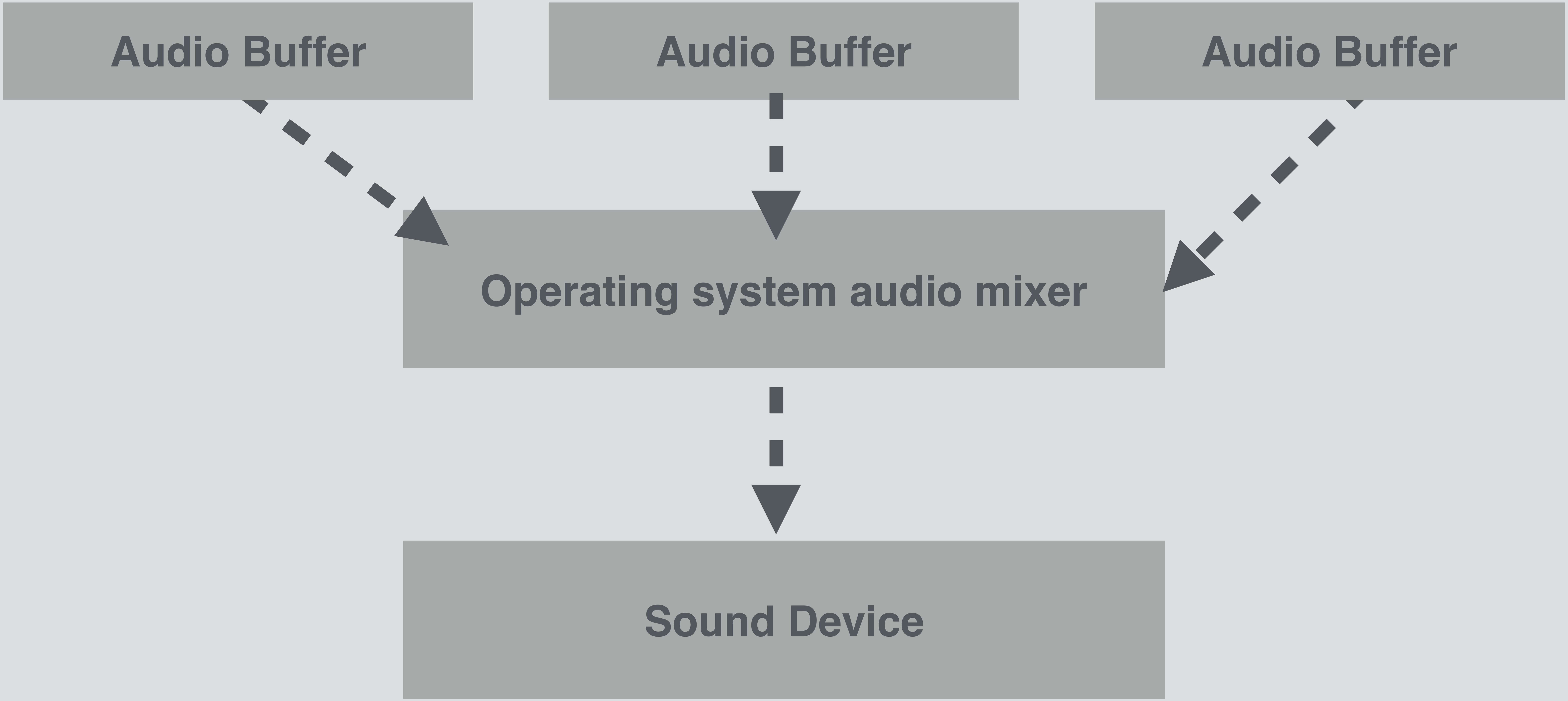
Reflects the total bit count processed per second.

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

Playing digital audio.

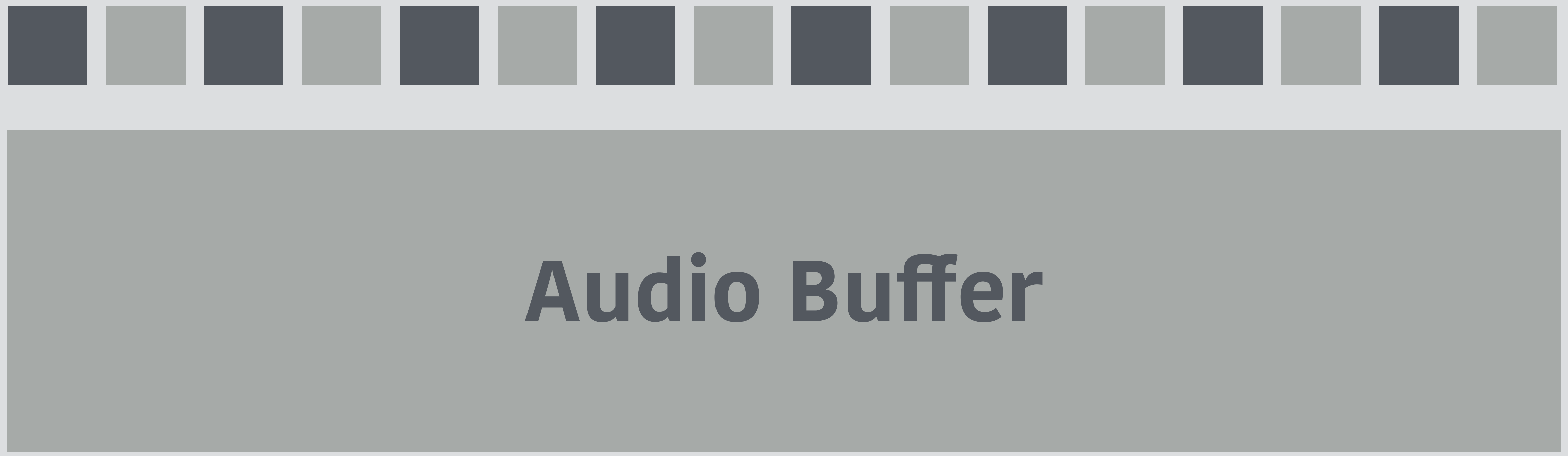



Application Process
Service Process
Audio Service
Media Foundation
SAR
Windows Multimedia
waveXxx Functions
mixerXxx Functions
DirectMusic
DirectSound
Protected Process
Audio From Other Apps
Core Audio APIs
MMDevice API
WASAPI
DeviceTopology API
EndpointVolume API
Endpoint Buffer
Shared Mode
Audio Engine
Exclusive Mode
User Mode
Endpoint Buffer
Kernel Mode
Mix
Audio Driver
Software
Hardware
Endpoint Device (Speakers)
Audio Adapter



Audio Buffer
Audio Buffer
Audio Buffer
Operating system audio mixer
Sound Device

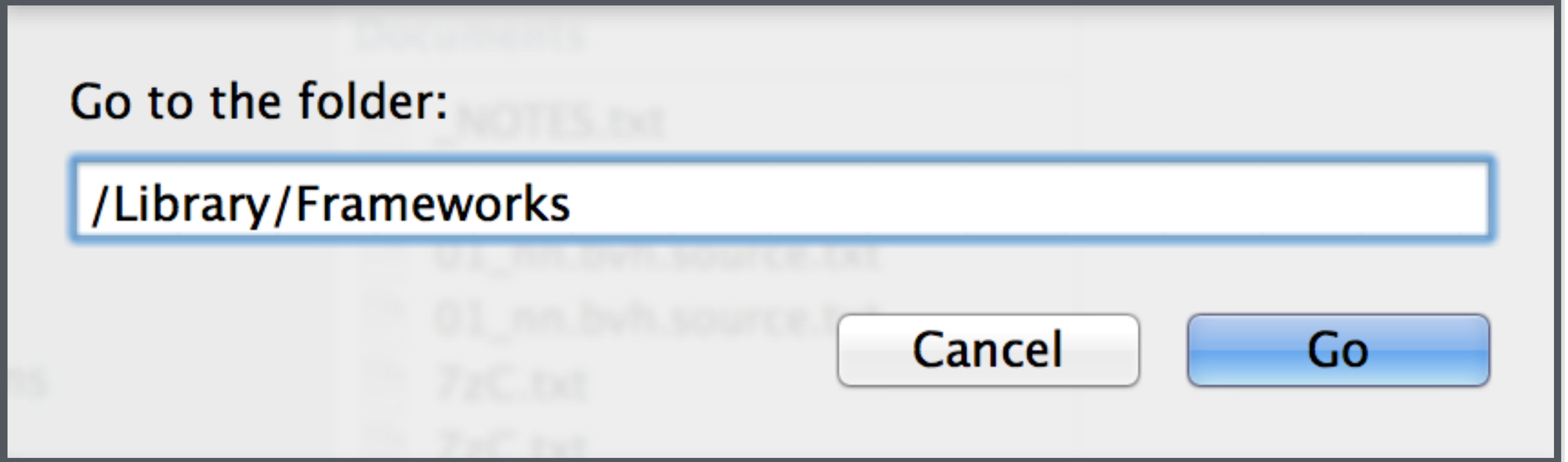
Playing **digital sound** involves **filling an audio buffer** with **audio data** on demand.

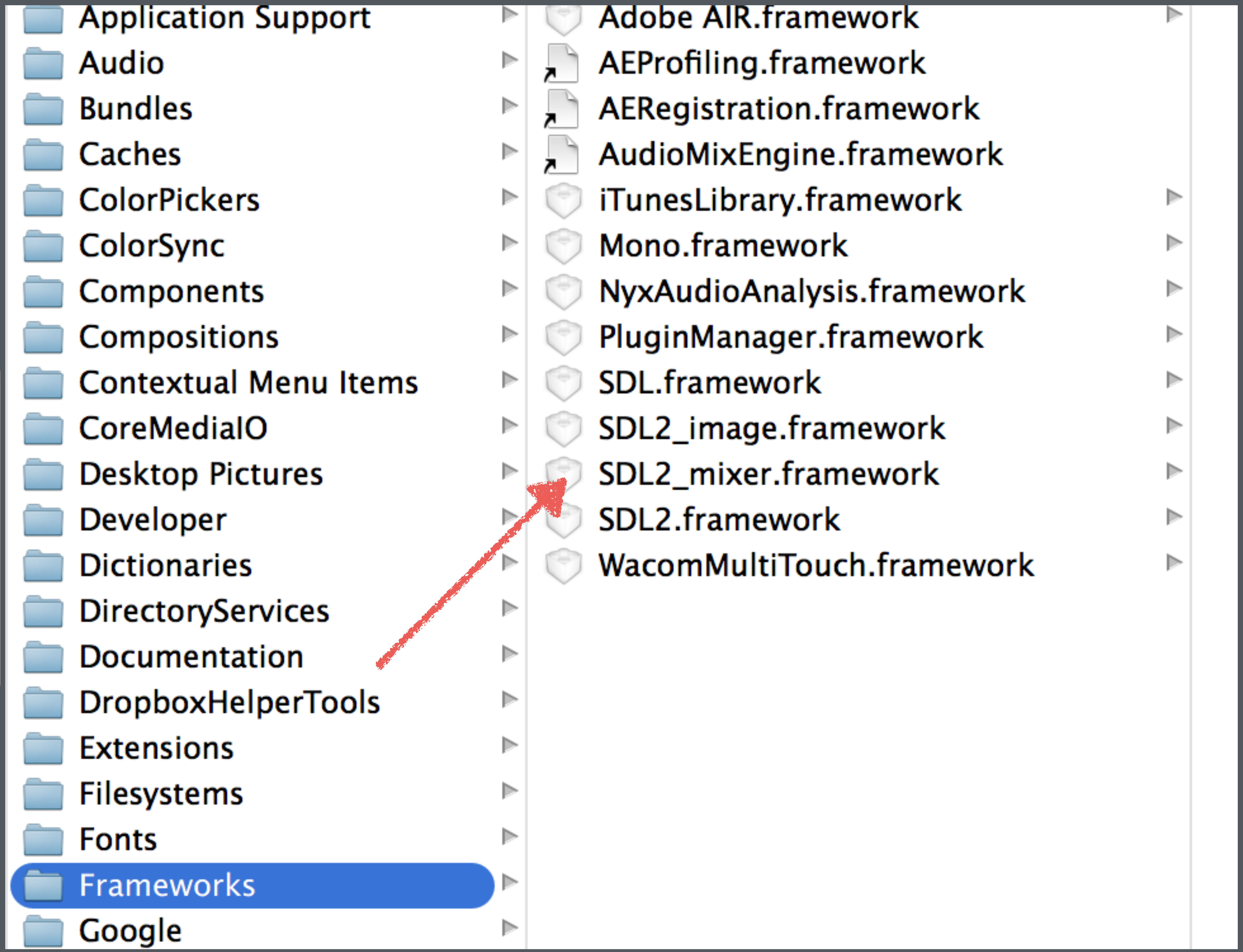


Playing audio with **SDL2**

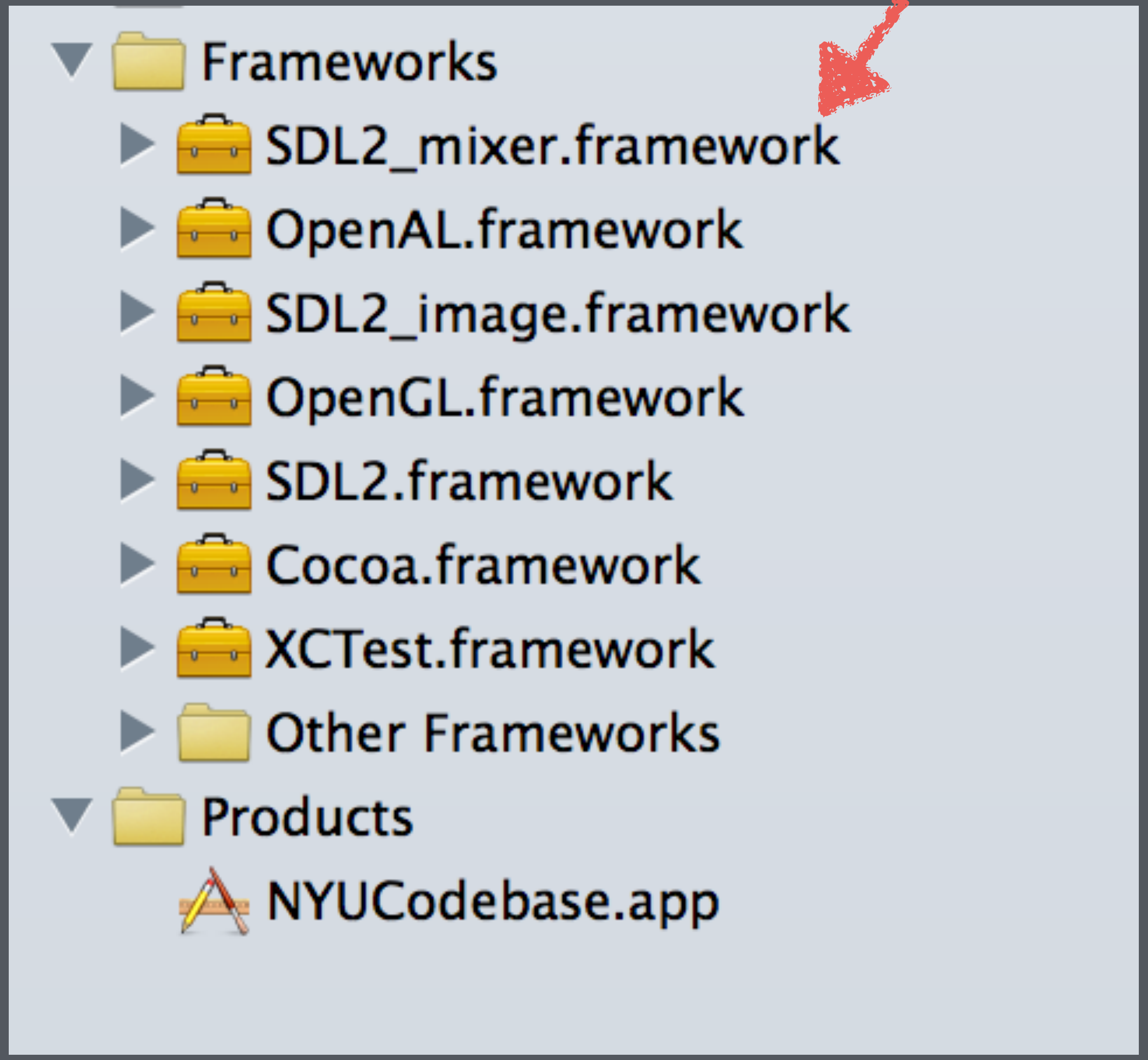
SDL_mixer (the easy way).

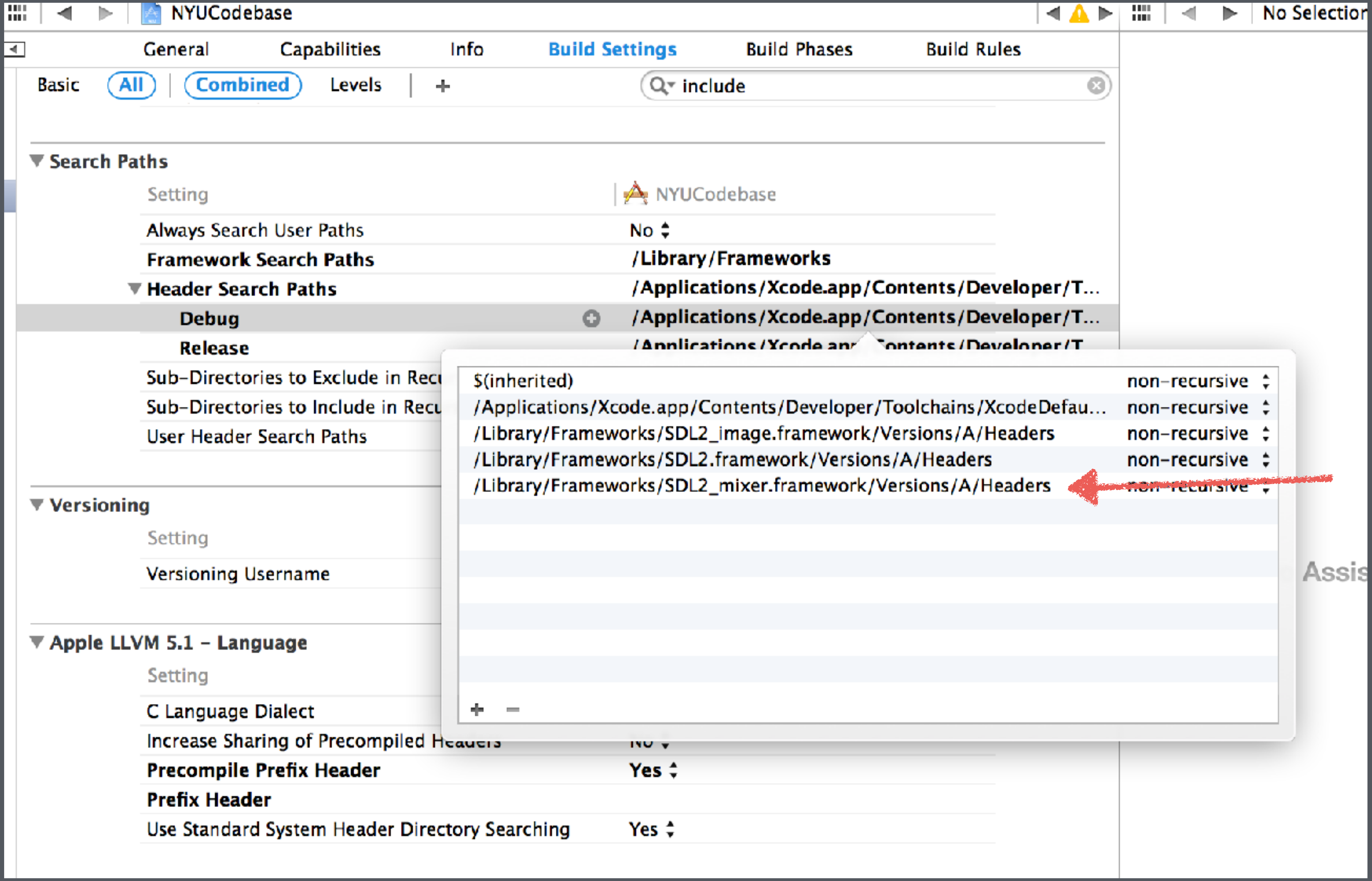
On **Mac**


Go to the folder:
/Library/Frameworks
Cancel
Go

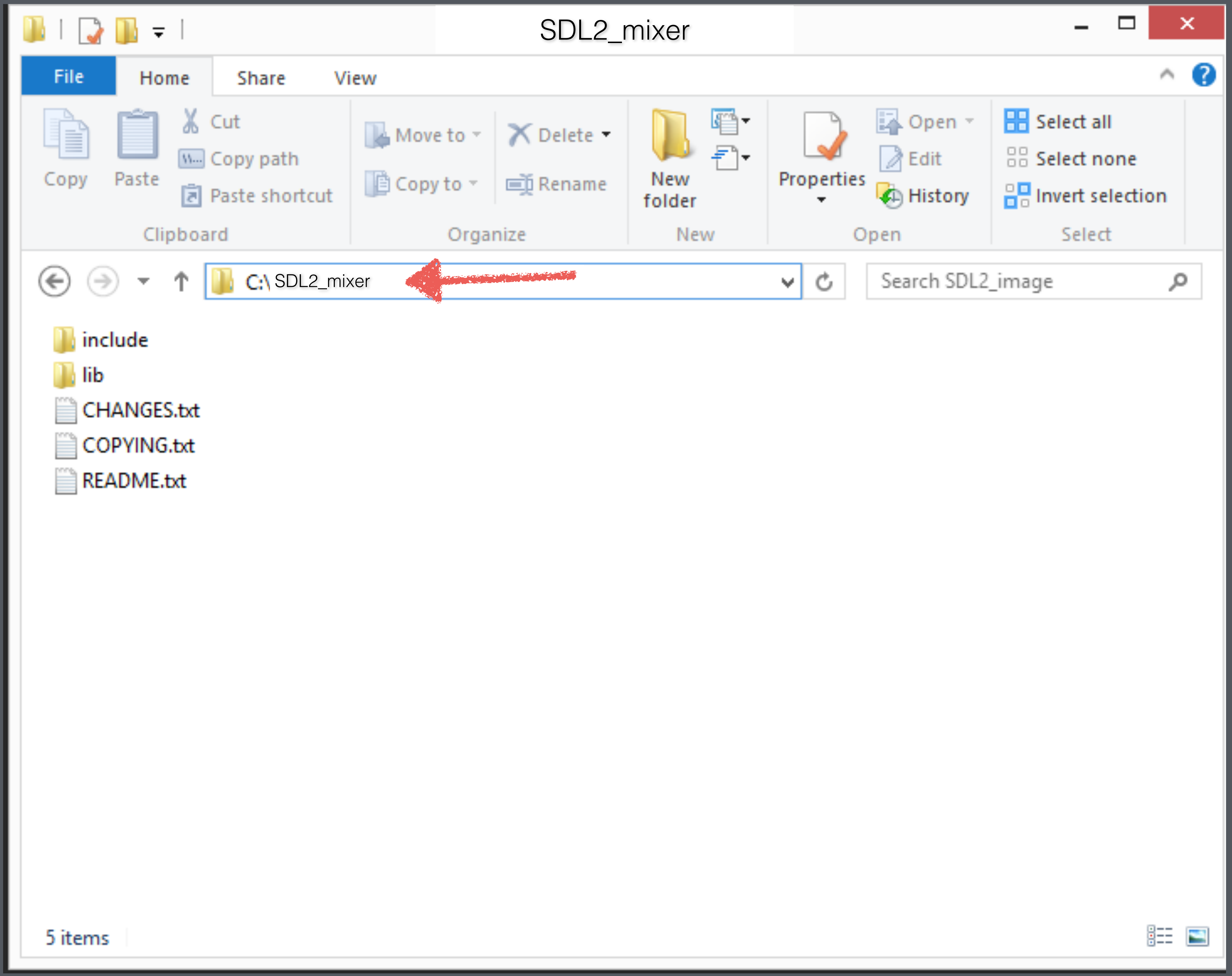

Application Support
Audio
Bundles
Caches
ColorPickers
ColorSync
Components
Compositions
Contextual Menu Items
CoreMediaIO
Desktop Pictures
Developer
Dictionaries
DirectoryServices
Documentation
DropboxHelperTools
Extensions
Filesystems
Fonts
Frameworks
Google
Adobe AIR.framework
AEProfiling.framework
AERegistration.framework
AudioMixEngine.framework
iTunesLibrary.framework
Mono.framework
NyxAudioAnalysis.framework
PluginManager.framework
SDL.framework
SDL2_image.framework
SDL2_mixer.framework
SDL2.framework
WacomMultiTouch.framework

DRAG FROM FINDER



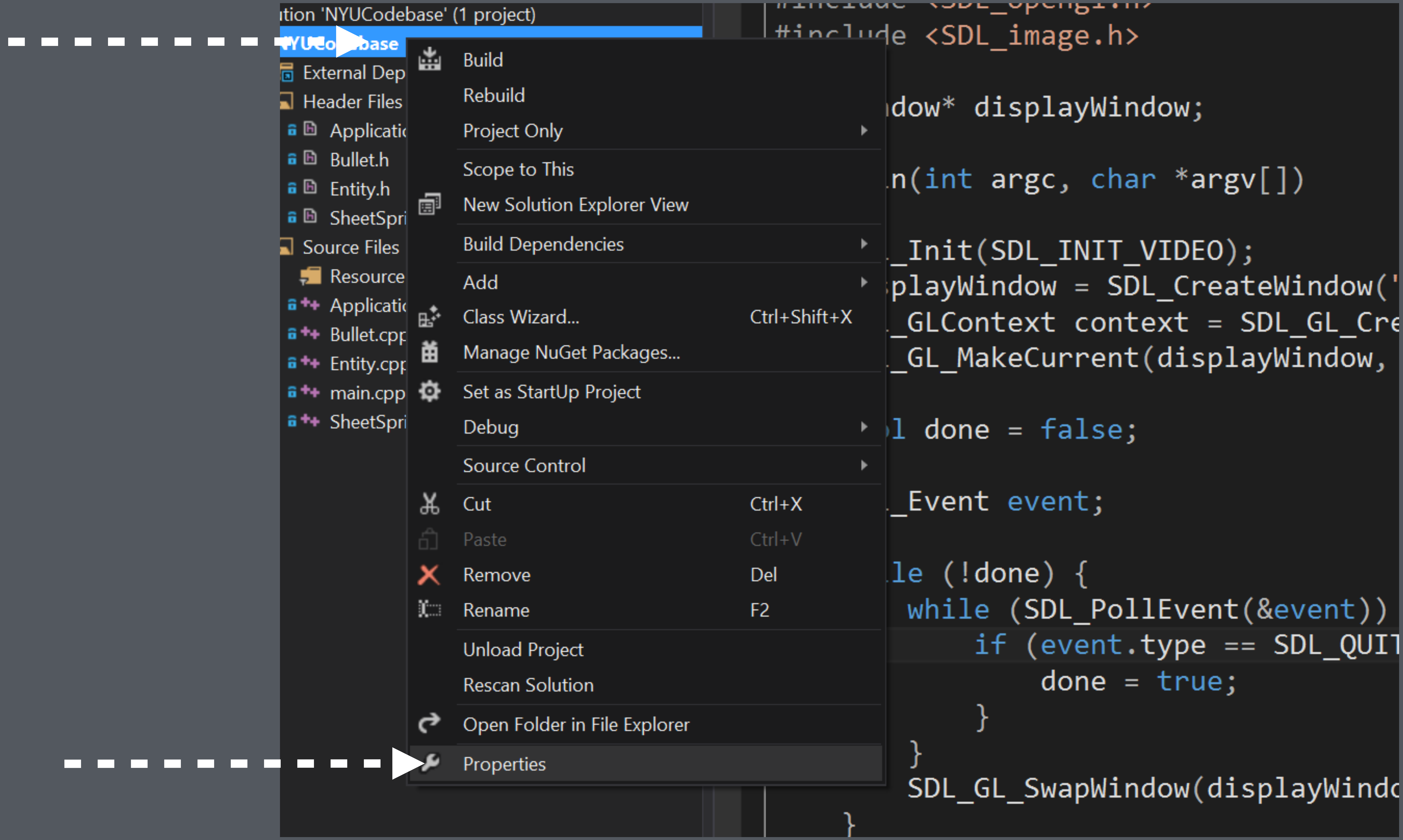


On **Windows**



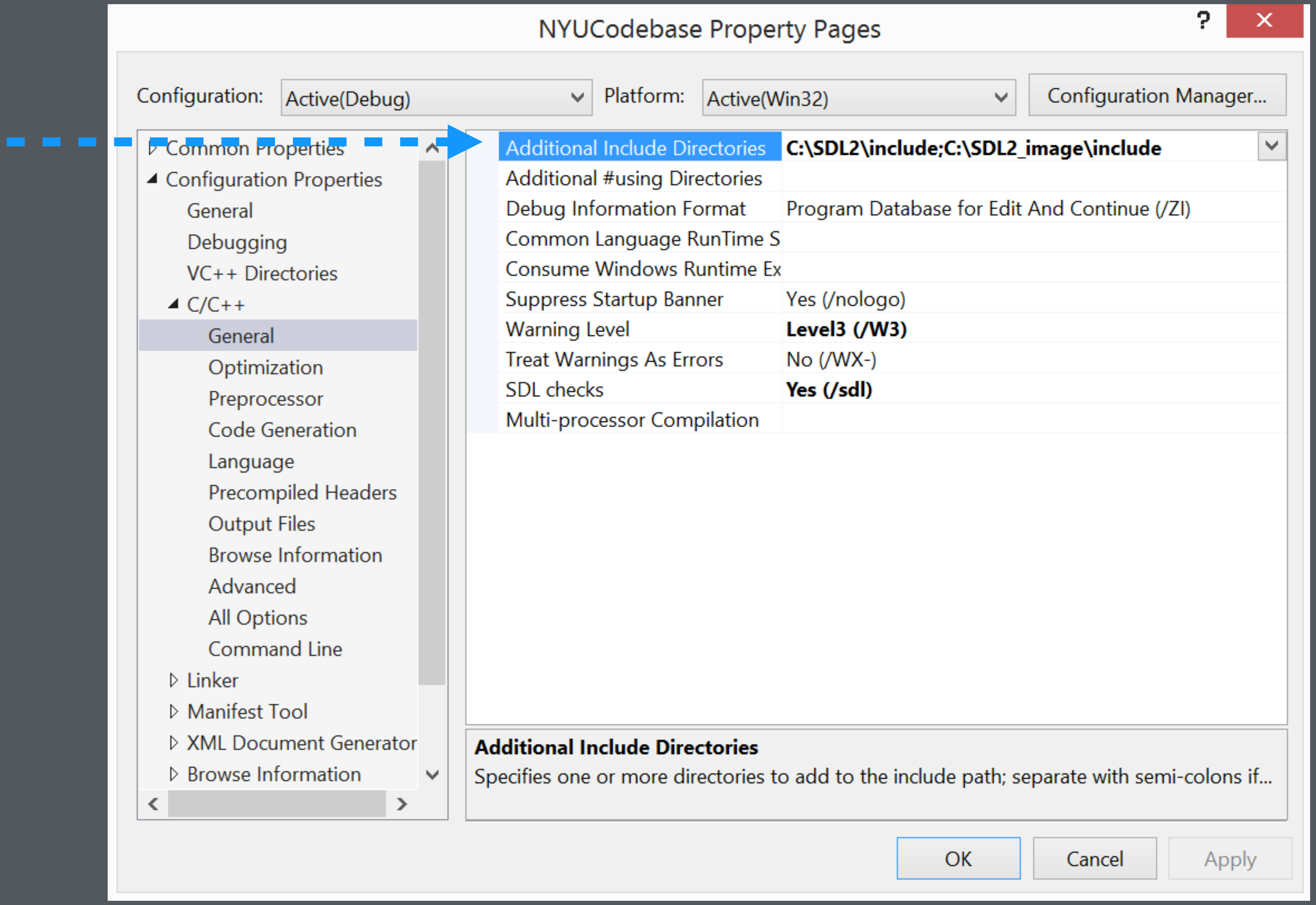
SDL2_mixer
File
Home
Share
View
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
C:\ SDL2_mixer
Search SDL2_image
include
lib
CHANGES.txt
COPYING.txt
README.txt
5 items

RIGHT CLICK PROJECT

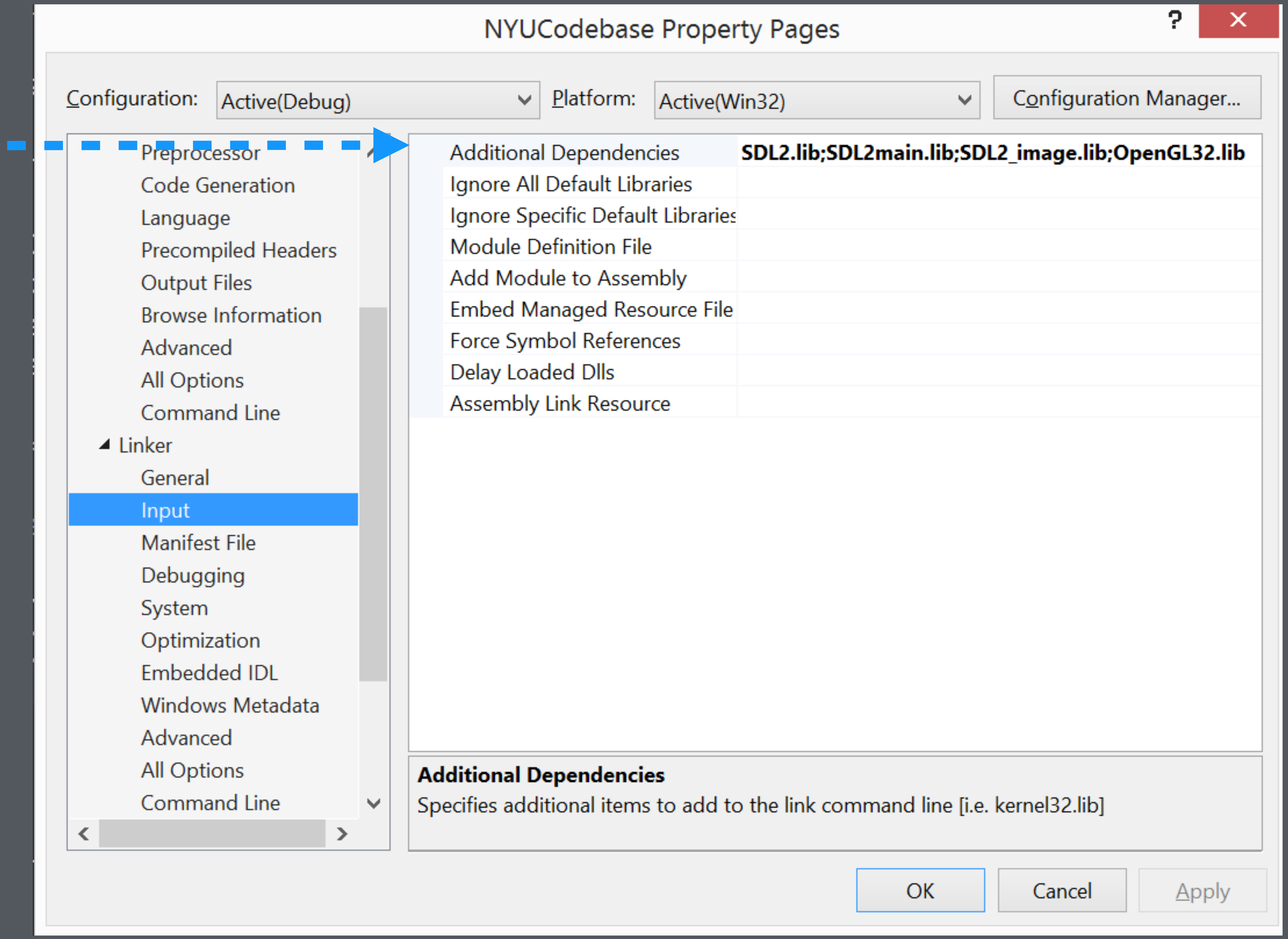


GO TO PROPERTIES

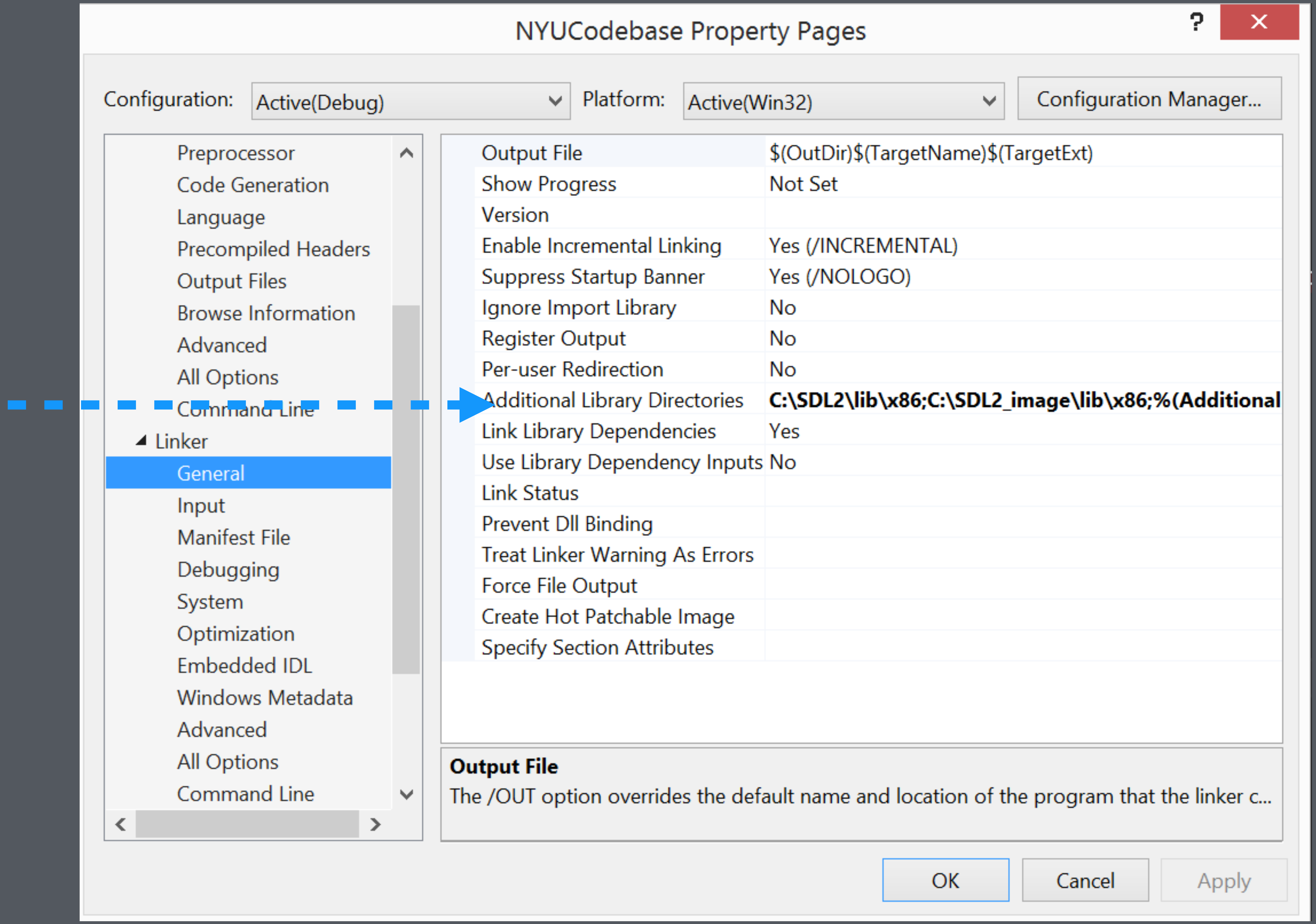
Add C:\SDL_mixer\include to additional library directories



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);
```

Additional useful functions.

Stopping music.

```
Mix_HaltMusic();
```

Setting volume.

```
Mix_VolumeMusic(30); // set music volume (from 0 to 128)

Mix_VolumeChunk(shootSound, 10); // set sound volume (from 0 to 128)
```

Stopping mixing on a channel.

```
Mix_HaltChannel(3); // stop playback on channel 3 (-1 to stop all sound)
```

Cleaning up.

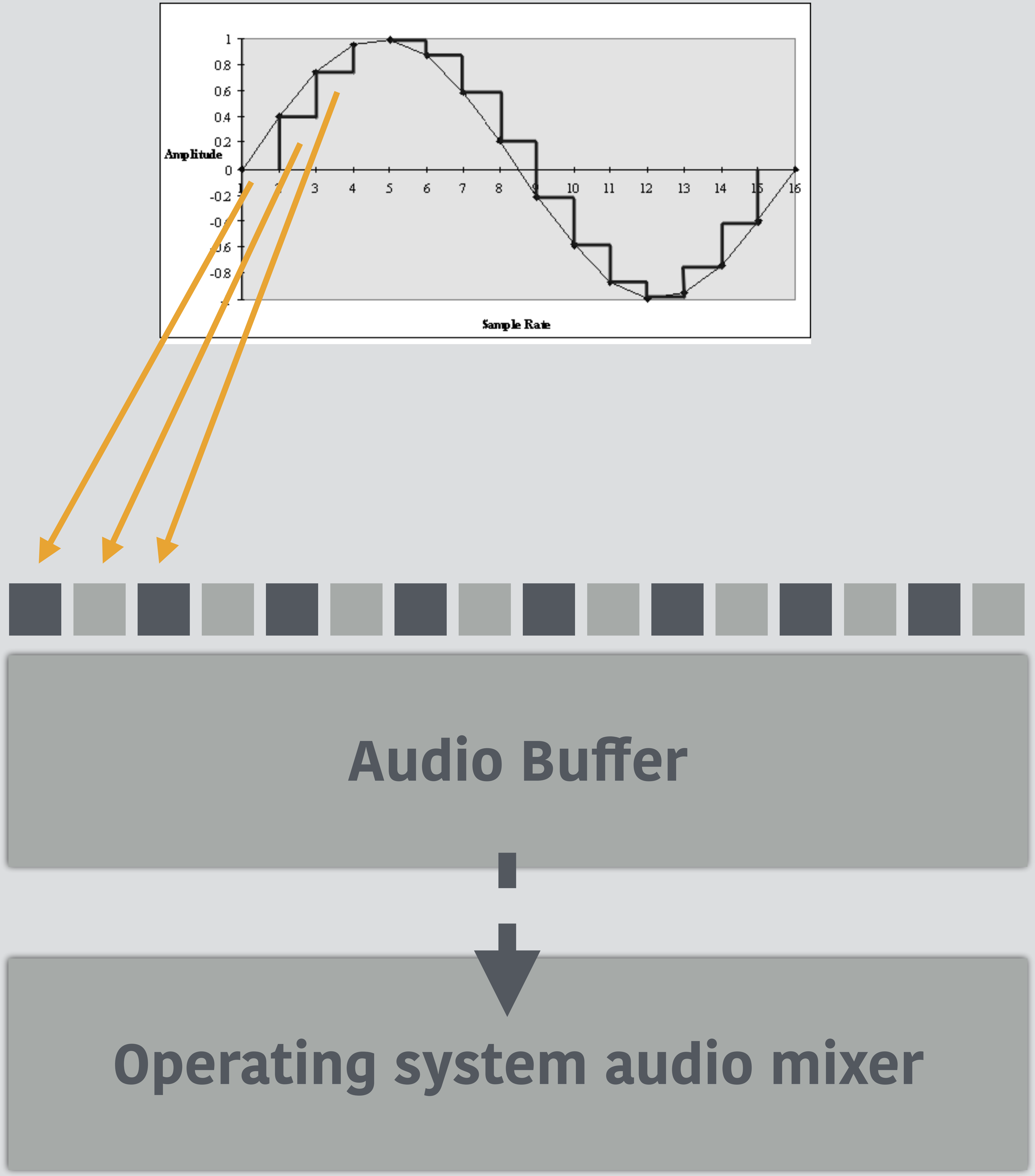
Need to **clean up** music and sounds on quit.

```
DemoApp::~DemoApp() {

    Mix_FreeChunk(someSound);
    Mix_FreeMusic(music);

    SDL_Quit();
}
```

Pure **audio buffer** (the hard way).


Amplitude
Sample Rate
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**.

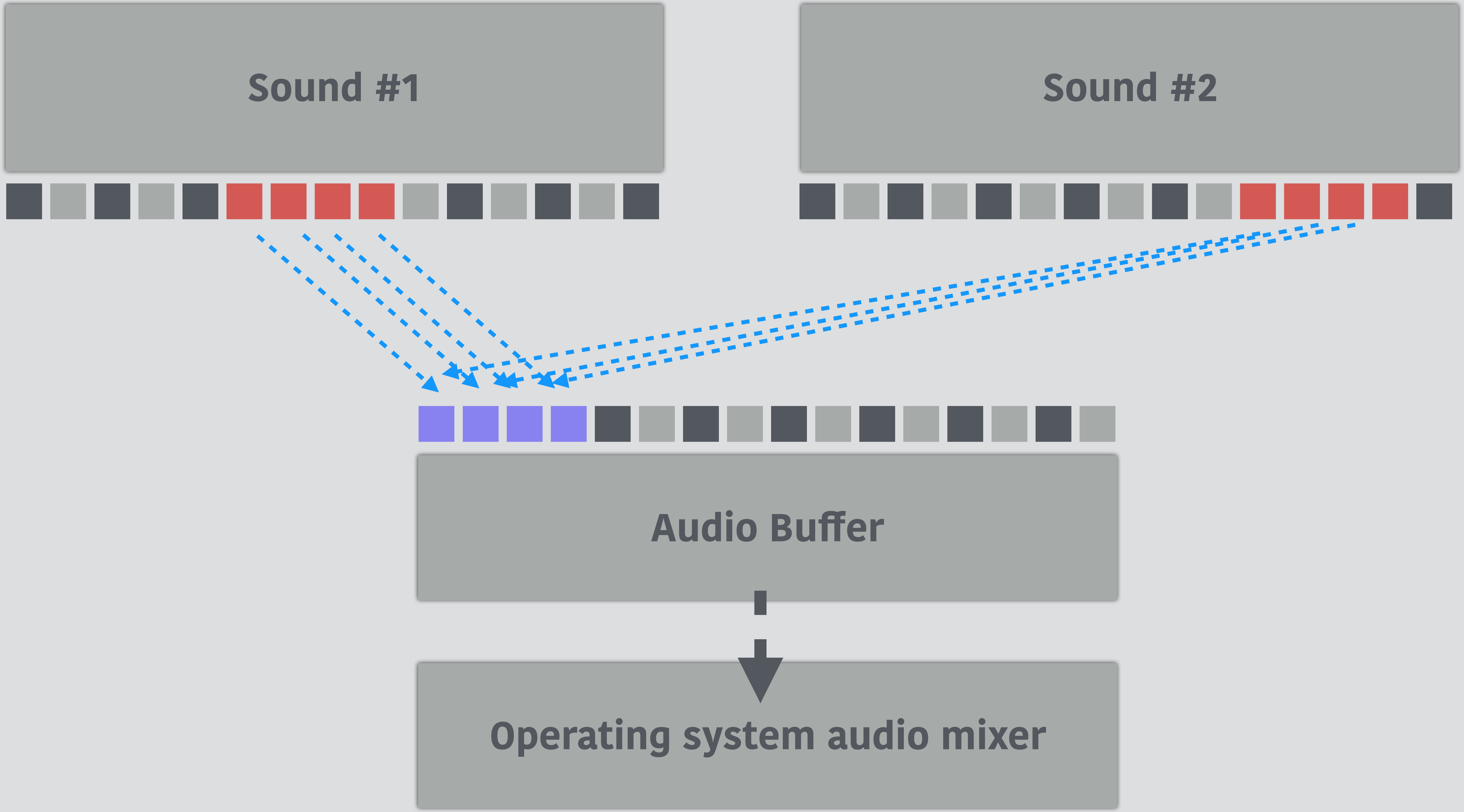
Here is an example of an audio callback function that's generating a 440Hz tone at 44100 sample frequency in a one channel float buffer.

```
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**.


Sound #1
Sound #2
Audio Buffer
Operating system audio mixer

```cpp
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.

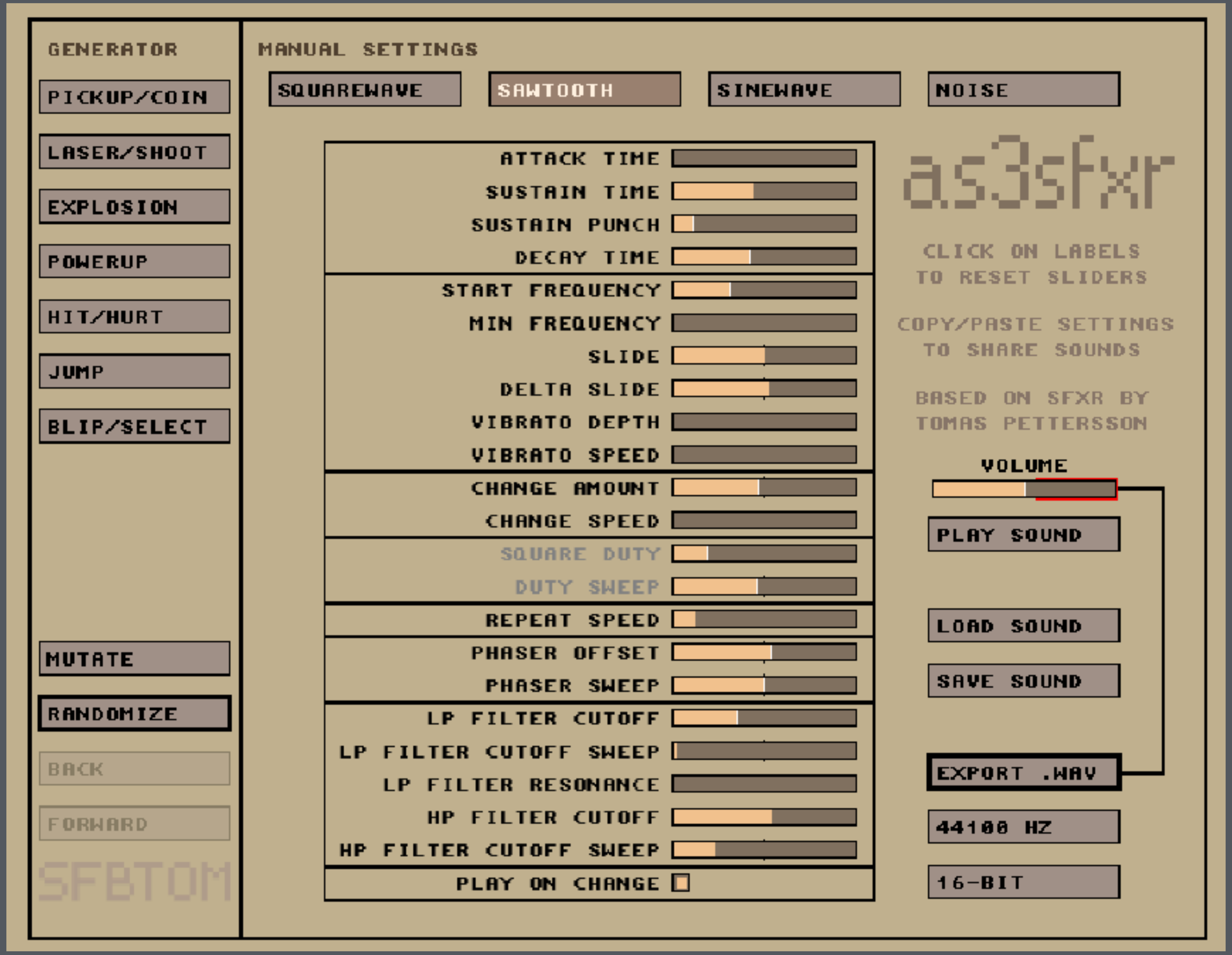
```
void DemoApp::playSound(int soundIndex, bool loop) {
    if(soundIndex >= 0 && soundIndex < mixerSounds.size()) {
        mixerSounds[soundIndex].playing = true;
        mixerSounds[soundIndex].offset = 0;
        mixerSounds[soundIndex].loop = loop;
    }
}
```

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

Sound **resources**.

SFXR

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



Free Music Archive

http://freemusicarchive.org/