Theoretical Backgrounds of Audio & Graphics

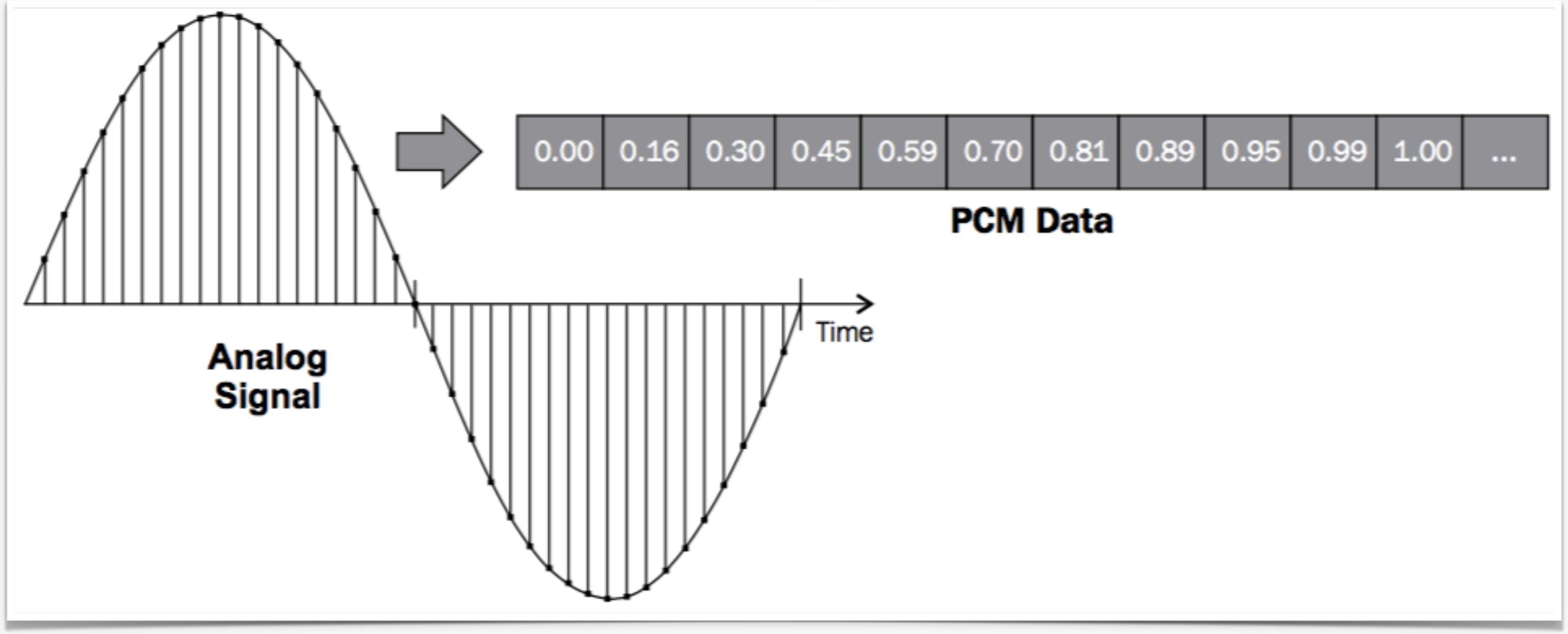
Audio Buffers

Angela Brennecke | Prof. Dr.-Ing.
Audio & Interactive Media Technologies

Filmuniversität Babelsberg
KONRAD WOLF

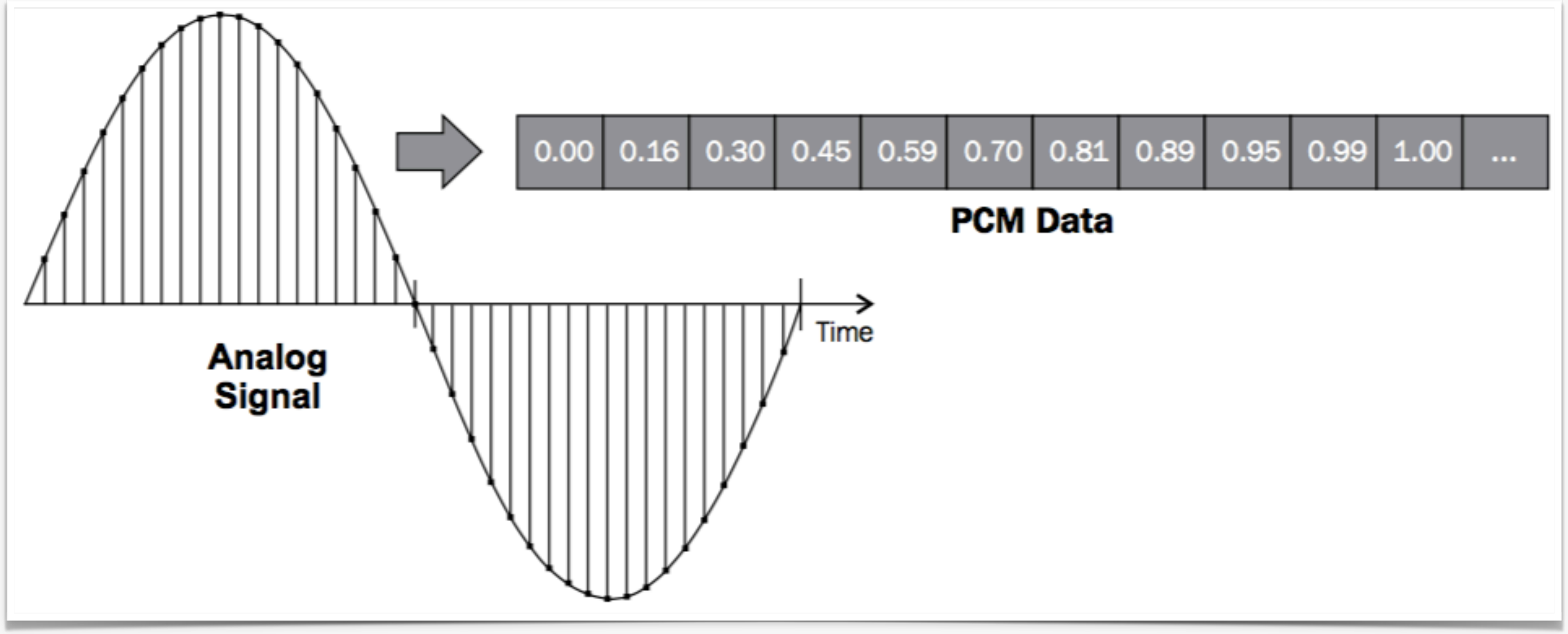
Audio Buffers

- The **audio buffer** is central to sound recording & playback as it stores amplitude values of any digitized sound wave
- Each **index** of the audio buffer represents a point in **time** whereas the value at the index represents the corresponding **amplitude** value



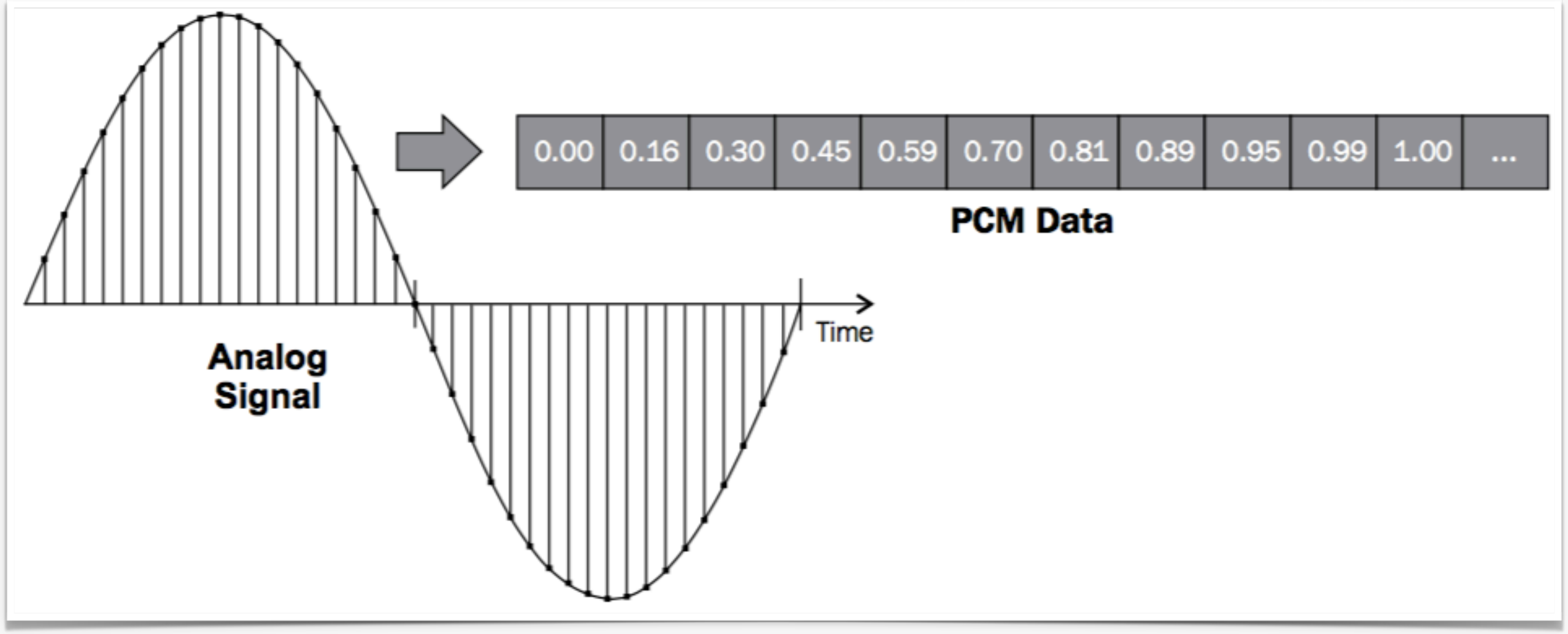
Audio Buffers

- In order to generate sound synthetically, we need to calculate a sound wave and store its amplitude values per time step in the audio buffers



Audio Buffers

- The **sampling rate** is crucial for filling & playing back the audio buffer as it represents our time unit
- Most important question:
  How to relate the sinusoid's time index to the sampling rate?



Filling the Audio Buffer

- Create a digital sound of **440 Hz** that lasts for **2 seconds** and plays back at a frequency of **44.1 kHz** (sampling rate)

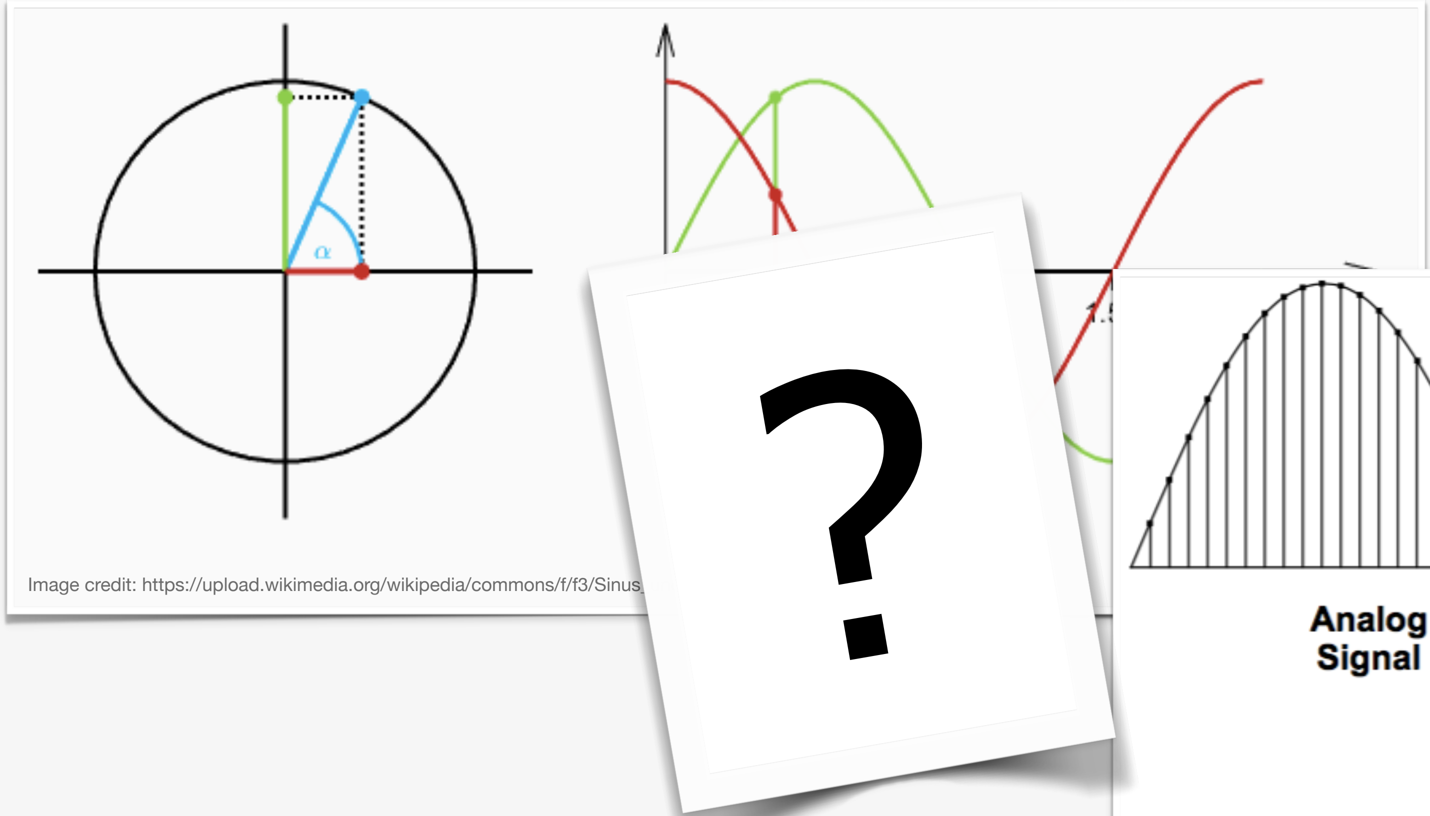


Image credit: https://upload.wikimedia.org/wikipedia/commons/f/f3/Sinus

$$y = A \sin(2\pi f t + ø)$$

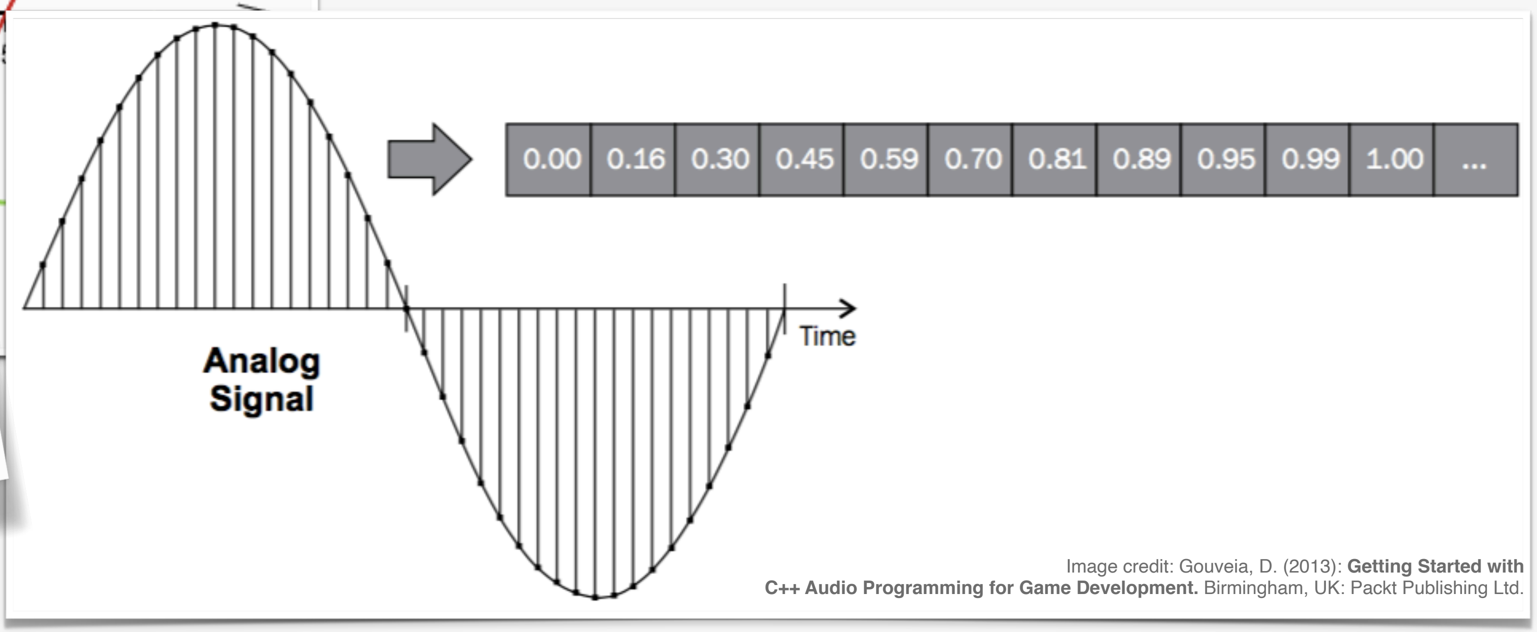


Image credit: Gouveia, D. (2013): **Getting Started with C++ Audio Programming for Game Development.** Birmingham, UK: Packt Publishing Ltd.

Filling the Audio Buffer

- Create a digital sound of **440 Hz** that lasts for **2 seconds** and plays back at a frequency of **44.1 kHz** (sampling rate)

$$y = A \sin(2\pi f t + \phi)$$

- **Todo**
  - Create an audio buffer of size 2 seconds
  - Fill the audio buffer with sine wave values at frequency 440 Hz
  - Relate time index t to the sampling frequency 44.1kHz

Filling the Audio Buffer

- Create a digital sound of **440 Hz** that lasts for **2 seconds** and plays back at a frequency of **44.1 kHz** (sampling rate)

$$y = A \sin(2\pi f t + \phi)$$

- **Todo continued**
  - Create an audio buffer of size 2 seconds: **2 * 44100**
  - Fill the audio buffer with sine wave values at frequency 440 Hz
  - Relate time index t to the sampling frequency: **t / 44100**

Filling the Audio Buffer

- Create a digital sound of **440 Hz** that lasts for **2 seconds** and plays back at a frequency of **44.1 kHz** (sampling rate)

$$y = A \sin(2\pi f t + ø)$$

```
audioBuffer = array[44100 * 2];

for (t = 0; t < 88200; t++) {
  A = 1;
  y = A * sin (2π * 440 * (t / 44100));
  audioBuffer[ t ] = y;
}
```

More on Audio Buffers

Continuous Signal

- Audio buffers are data structures used to track all values & properties required to reconstruct the waveform of a continuous analog signal

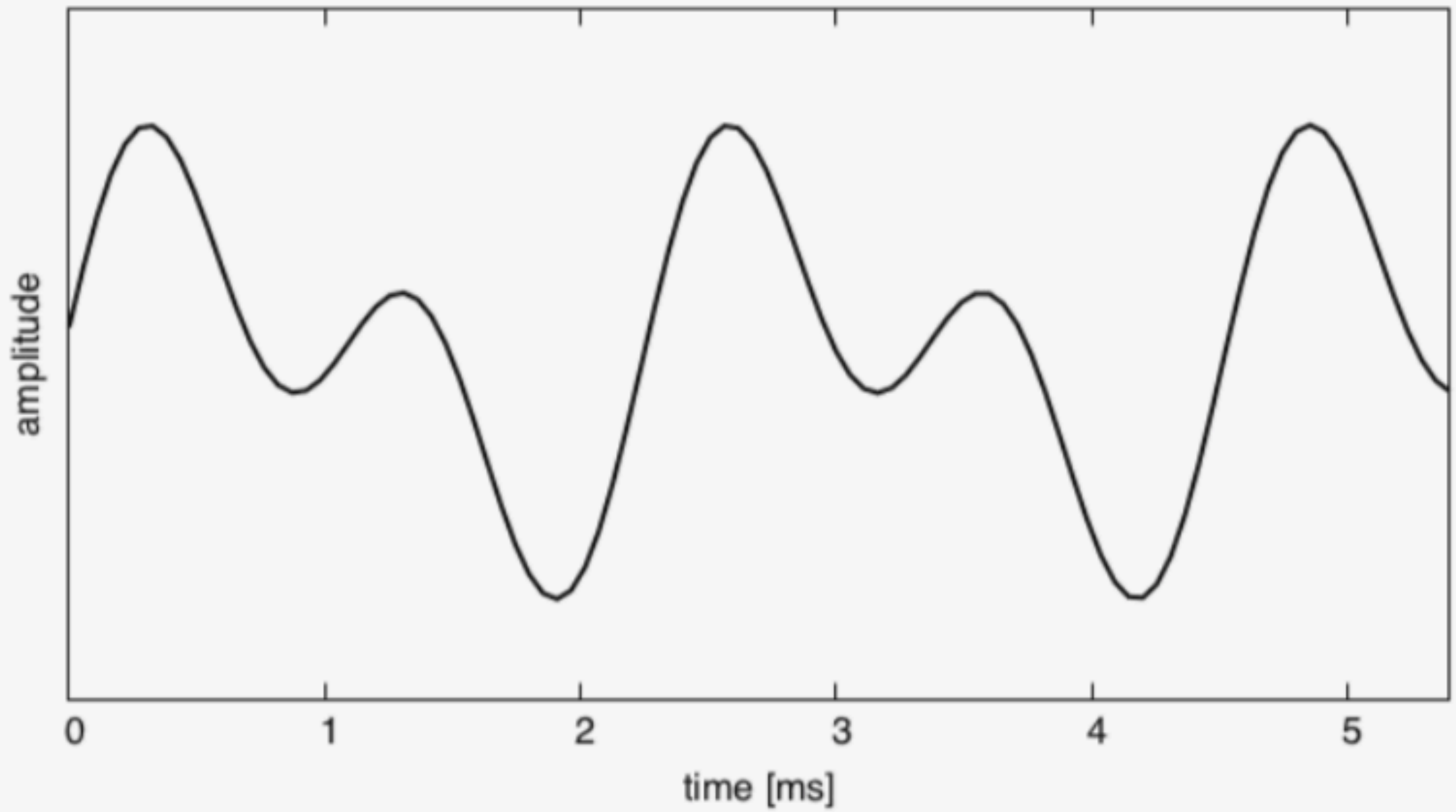


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Sampled Signal

- Audio buffers store the sampled amplitude values per time index in a data array
- Additionally, they store the sampling rate for correctly reading & writing the data

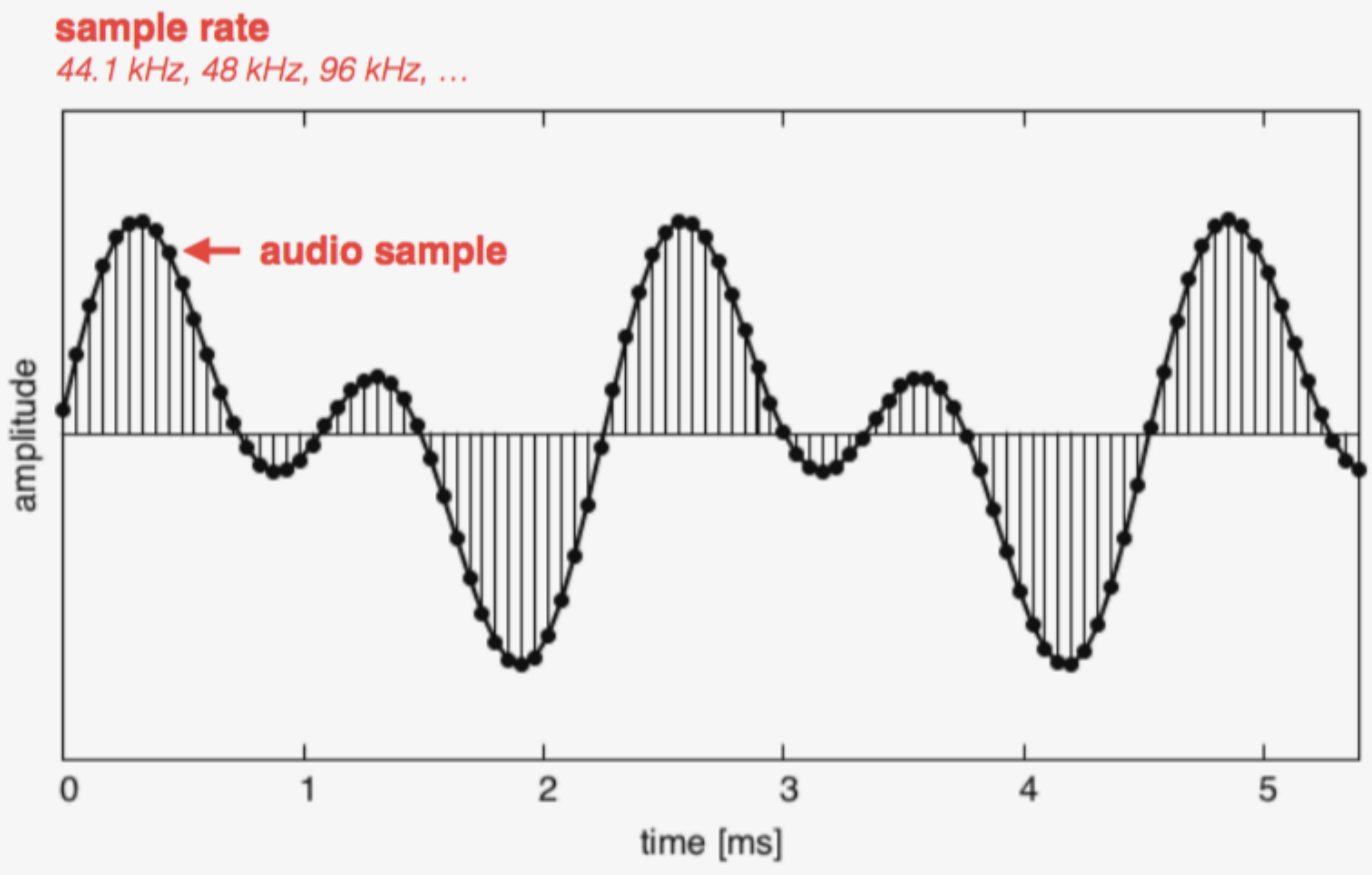


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Amplitude Range

- Amplitude values are approximated based on the sample size / bit depth
- Audio buffers usually scale those values to floating point range [-1.0, 1.0]

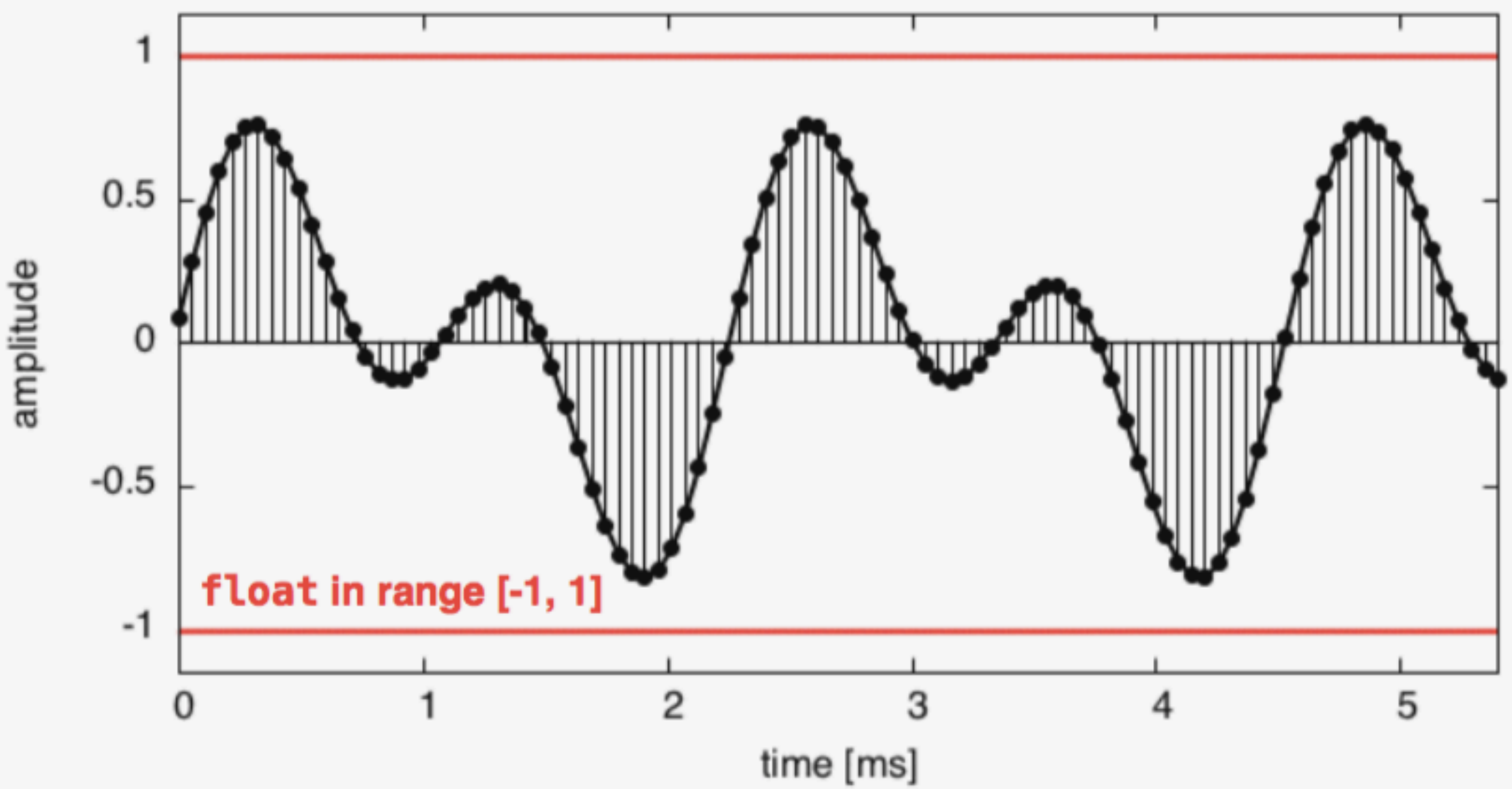


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Channels & Frames

- Audio buffers organize amplitude values per channel (mono, stereo, …)
- An audio frame represents a (set of) sample per channel

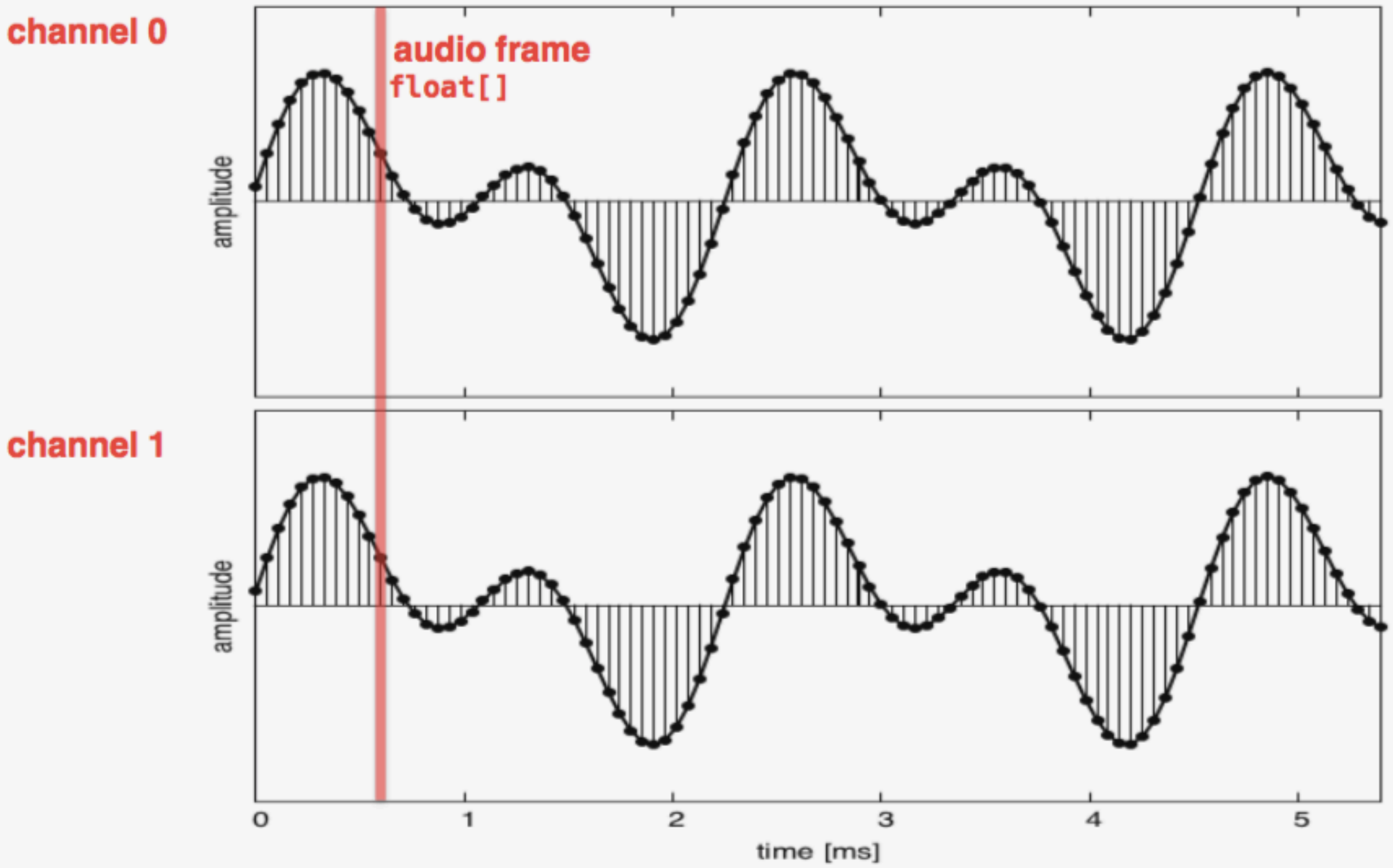


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Audio Buffers

- Audio buffers store the sampled values per channels for all channels
- Different organization forms are used, i.e., interleaved and non-interleaved

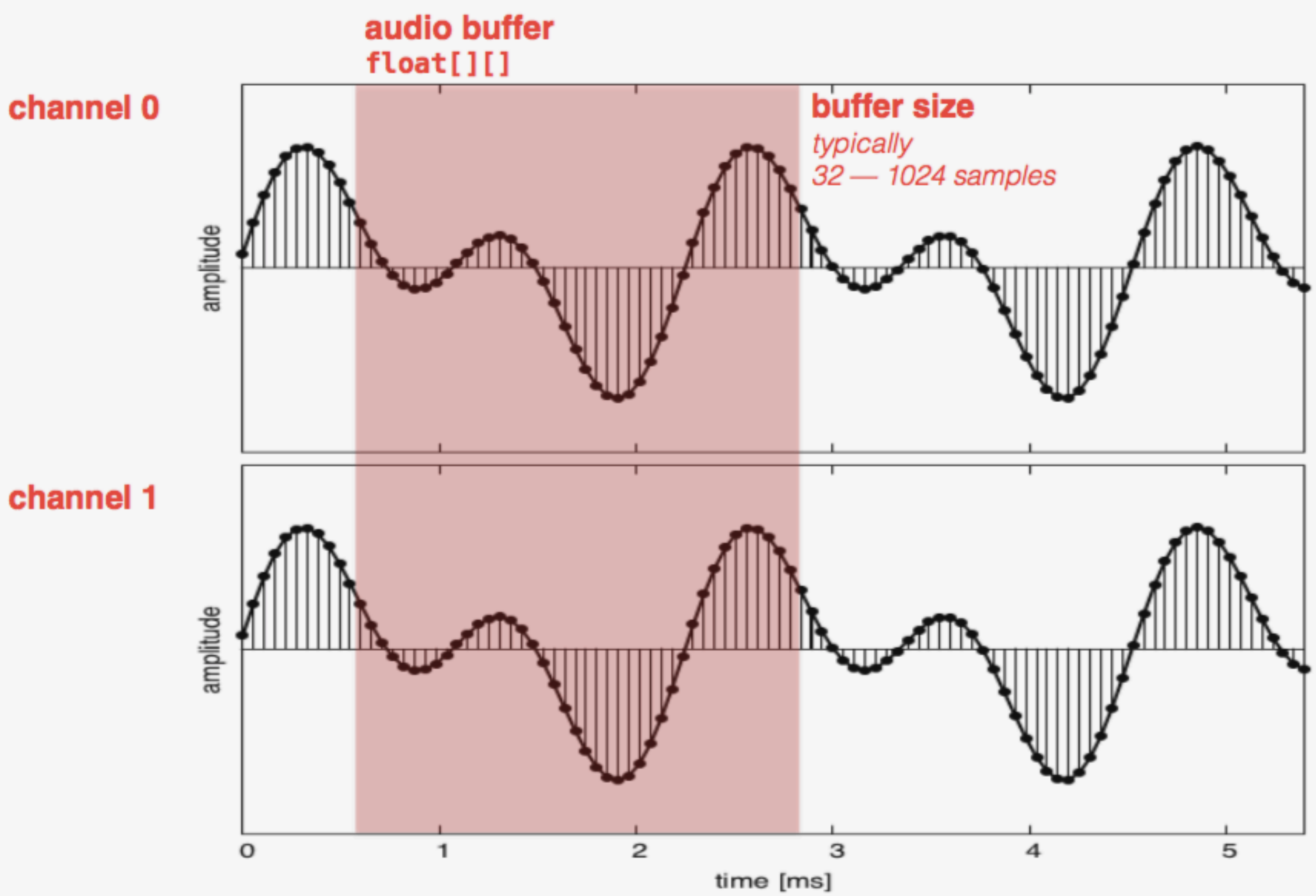


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Audio Buffer Sizes

- Audio buffer sizes are usually kept small to ensure continuous processing of audio data streams in real-time — depending on hardware & driver capabilities

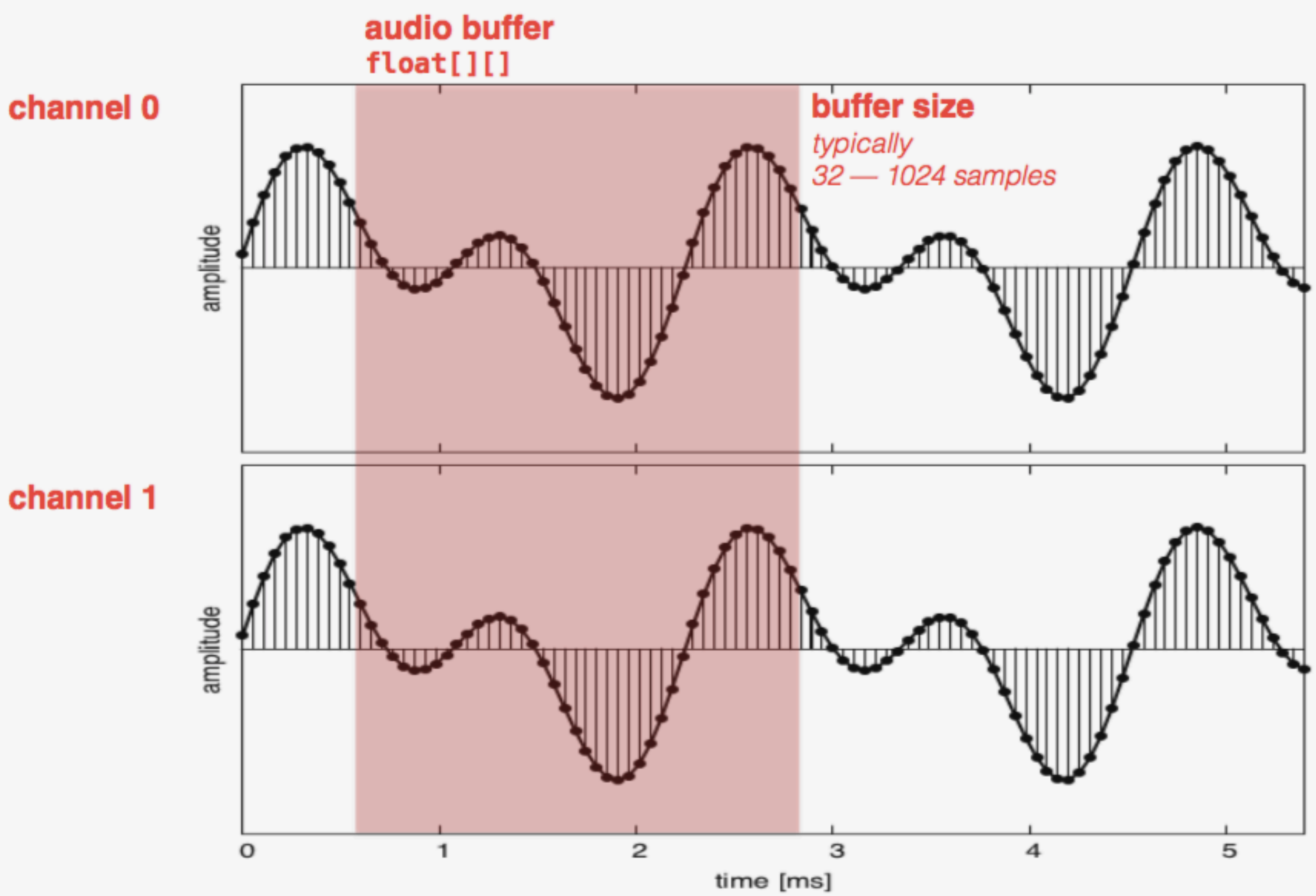


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

Real-Time Processing

- Real-time processing requires a continuous stream of data to avoid latency issues
- Latency basically describes the time delay between audio input to & output of a system

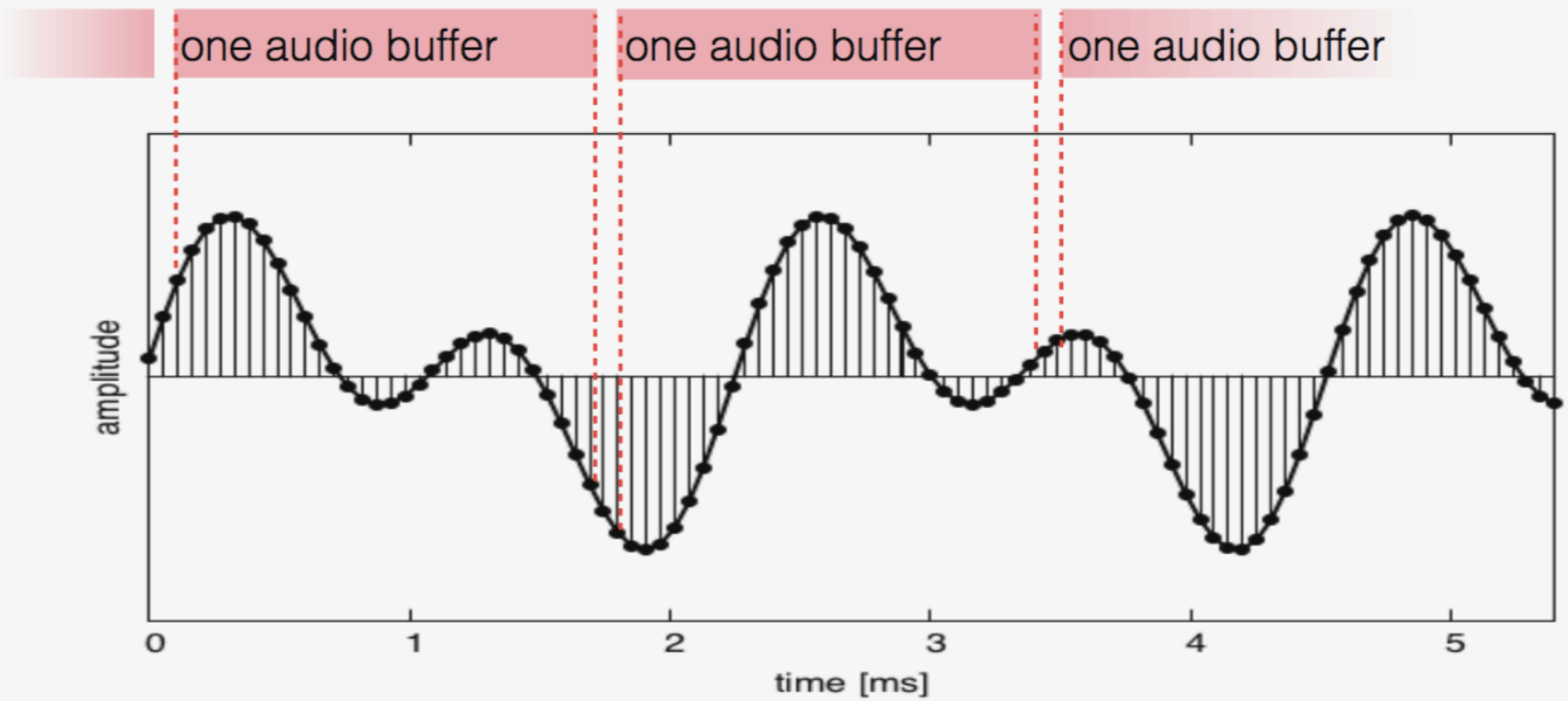


Image credit: Doumler, T. (2015): **Cpp in the Audio Industry.** Presentation at CppCon 2015. https://github.com/CppCon/CppCon2015/

WebAudio API

WebAudio API

- WebAudio API is a high level programming interface for developing audio applications for the web

- Main features
  - modular routing
  - audio I/O nodes
  - audio FX nodes
  - audio context

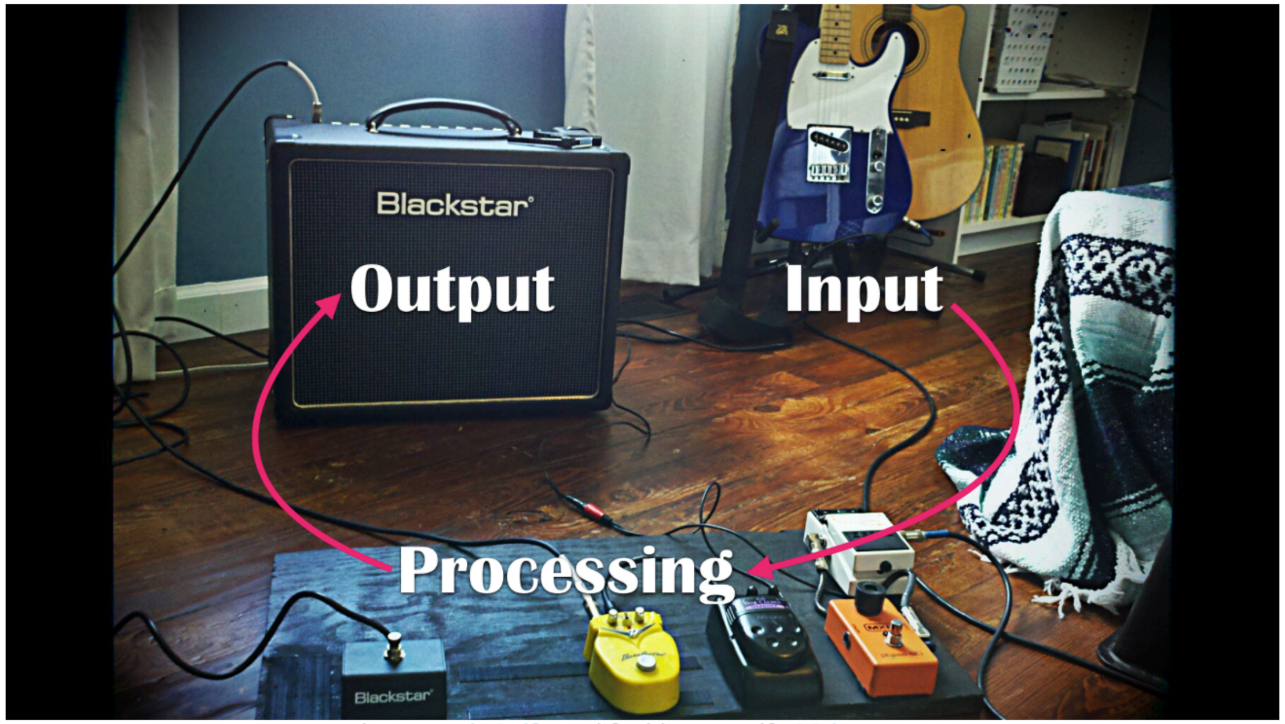
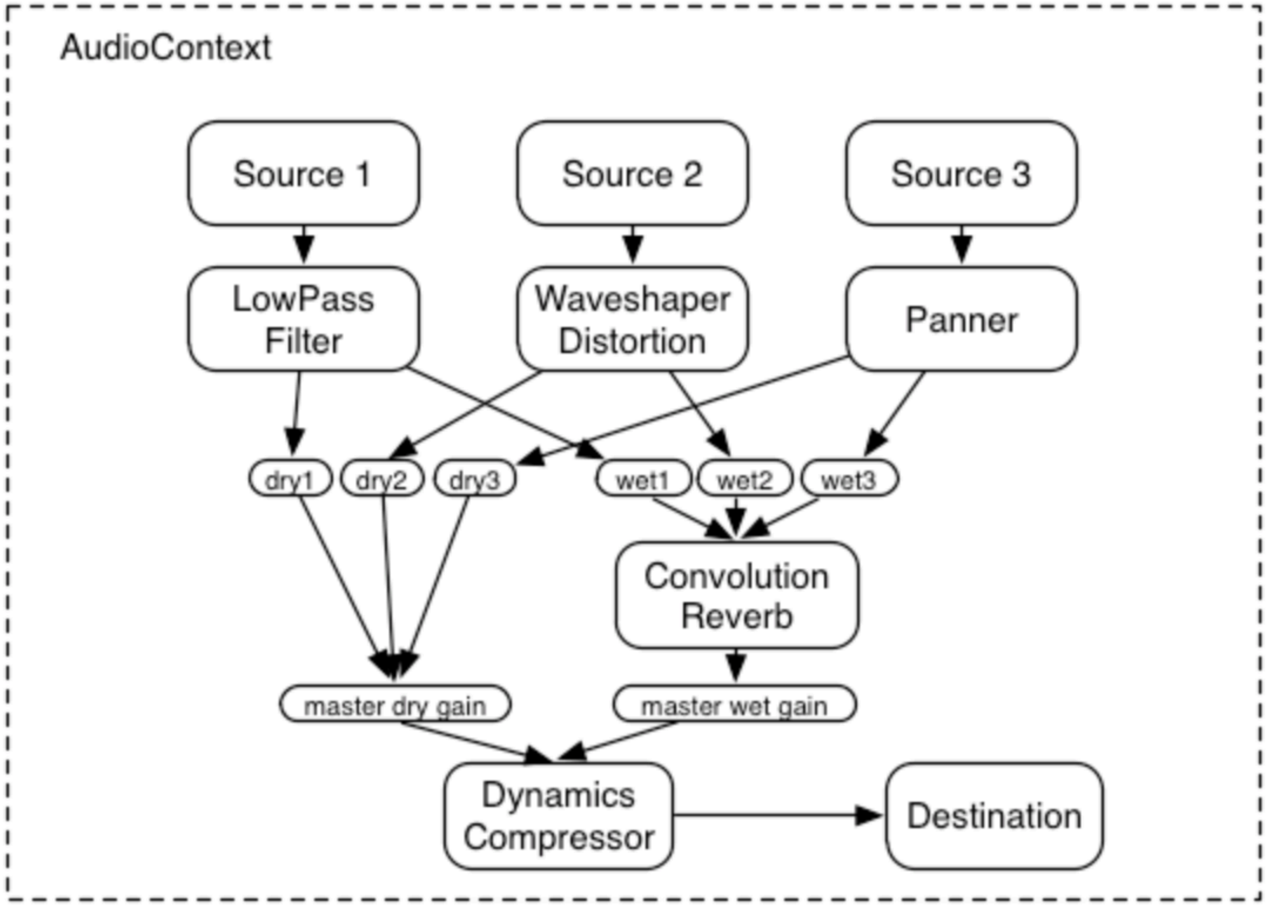


Photo: Josh Self. Modified by myself. CC by-nc-sa.

http://teropa.info/blog/2016/08/19/what-is-the-web-audio-api.html



https://www.w3.org/TR/2018/CR-webaudio-20180918/

WebAudio API

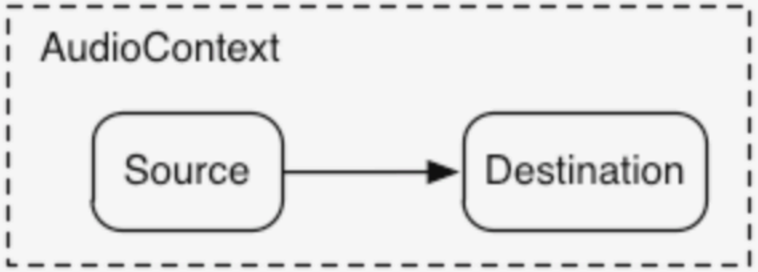


Figure 1 *A simple example of modular routing.*

Illustrating this simple routing, here's a simple example playing a single sound:

```
EXAMPLE 1
var context = new AudioContext();

function playSound() {
  var source = context.createBufferSource();
  source.buffer = dogBarkingBuffer;
  source.connect(context.destination);
  source.start(0);
}
```

https://www.w3.org/TR/2018/CR-webaudio-20180918/

Coding Example

```
// Source code courtesy of
// http://teropa.info/blog/2016/08/19/what-is-the-web-audio-api.html

const REAL_TIME_FREQUENCY = 440;
const ANGULAR_FREQUENCY = REAL_TIME_FREQUENCY * 2 * Math.PI;


let audioContext = new AudioContext();
let myBuffer = audioContext.createBuffer(1, 88200, 44100);
let myArray = myBuffer.getChannelData(0);


for (let sampleNumber = 0 ; sampleNumber < 88200 ; sampleNumber++)
{
    myArray[sampleNumber] = generateSample(sampleNumber);
}


function generateSample(sampleNumber)
{
  let sampleTime = sampleNumber / 44100;
  let sampleAngle = sampleTime * ANGULAR_FREQUENCY;
  return Math.sin(sampleAngle);
}


let src = audioContext.createBufferSource();
src.buffer = myBuffer;
src.connect(audioContext.destination);
src.start();
```

Javascript & WebAudio API coding excerpt that creates

- a digital sound of **440 Hz**
- with a duration of **2 secs**
- playback rate of **44.1 kHz**