WebAudio API: HTML5 and JavaScript

Nick Chemsak 2-27-2017

About Me

MTSU Grad (2003) - Music Industry (Production & Technology)

Gibson Guitar / Baldwin Piano - 5 years

IT for a credit union - Past 7 years

Nashville Software School - JavaScript/Angular & Python/Django

Demo - eWave Studio

...

https://ewavestudio-e15d0.firebaseapp.com/

WebAudio API

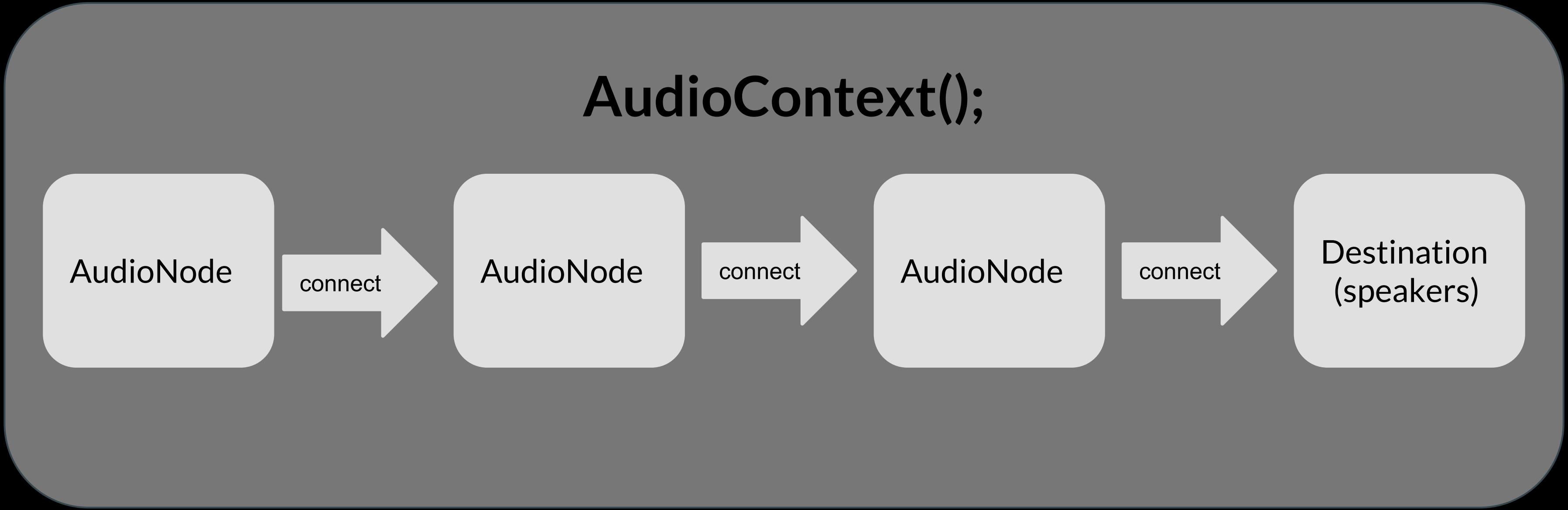
High-level JavaScript API for processing audio in web apps.

Everything you need is in your web browser!

The foundation of the Web Audio API is **AudioContext();**

Everything is created **inside** the AudioContext();

Basic Audio Routing Graph



Examples of AudioNodes

Oscillator	Panner
Gain	Audio Buffer Source
Filter	Delay

Open the Console in Dev Tools

And turn your computer volume to a reasonable level

Chrome

Windows:	Ctrl + Shift + j
OS X:	Cmd + Opt + j

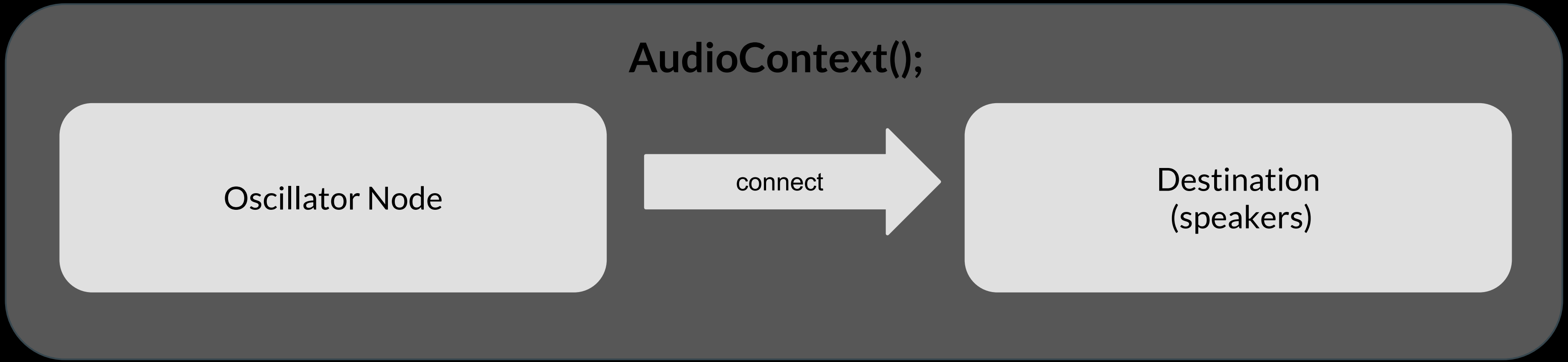
Firefox

Windows:	Ctrl + Shift + k
OS X:	Cmd + Opt + k

What is an Oscillator?

- For our use, an oscillator is a repeating waveform .
- It produces a different sound depending on the shape of the waveform.
- The most common waveforms are Sine, Square, Sawtooth, Triangle.

Basic Oscillator



Basic Oscillator

Javascript	Description
`let context = new AudioContext();`	Creates the AudioContext
`let osc = context.createOscillator();`	createOscillator is method on context. Creates the oscillator node.
`osc.frequency.value =261.6;`	Assigns a frequency to the oscillator (440 is default)
`osc.type = "triangle";`	Declares the type of oscillator ("sine" is default)
`osc.connect(context.destination);`	Connects the oscillator to speakers
`osc.start();`	Starts the oscillator
`osc.stop();`	Stops the oscillator

Basic Oscillator

let context = new AudioContext();

Oscillator

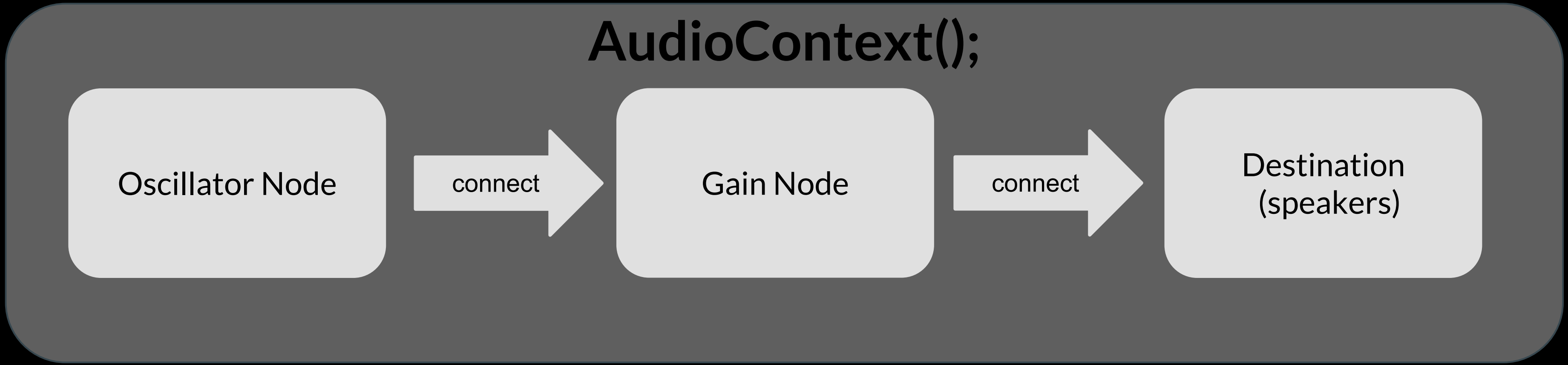
let osc = context.createOscillator();
osc.frequency.value = 261.6;
osc.type = “triangle”

osc.connect

Speakers

(context.destination)

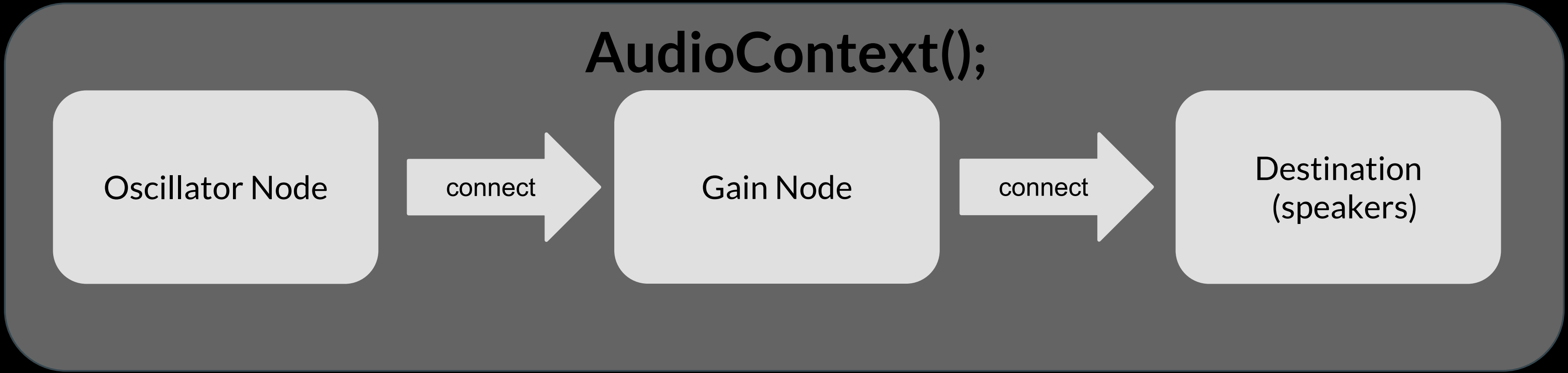
Oscillator with Gain Control



Oscillator with Gain Control

Javascript	Description
let context = new AudioContext();	
let osc = context.createOscillator();	
let gain = context.createGain();	Creates a Gain Node
gain.gain.value = 0.5;	Sets the volume at half (1 is default)
osc.connect(gain);	Connects Oscillator Node to the Gain Node
gain.connect(context.destination);	Connects Gain Node to speakers
osc.start();	

Oscillator Kick Drum



Kick Drum Acoustics

- The kick drum sound begins at a (*relatively*) higher frequency. This is during the ‘attack’, when the drum head is struck.
- The frequency quickly falls to a lower frequency.
- Simultaneously, while the frequency drops, the volume of the sound ‘decays’ away.

We can automate these actions in a very simple way with the following built in functions:

- setValueAtTime(*value*, *time*);
- exponentialRampToValueAtTime(*value*, *time*);

Oscillator Kick Drum

Javascript

```
var context = new AudioContext();
var now = context.currentTime;

var osc = context.createOscillator();
    osc.frequency.setValueAtTime(100, now);
    osc.frequency.exponentialRampToValueAtTime(0.001, now + 0.5);

var gain = context.createGain();
    gain.gain.setValueAtTime(1, now);
    gain.gain.exponentialRampToValueAtTime(0.001, now + 0.5);

osc.connect(gain);
gain.connect(context.destination);
osc.start();
```

Description

This sets a variable “now” to current time.

Sets the frequency immediately to 100 hz

Changes the frequency to nearly 0 over half a second.

Sets the volume at full (1) immediately.

Changes the volume to near 0 over half a second.

Creating Effects

Every AudioNode can connect to any other node as shown in previous slides.

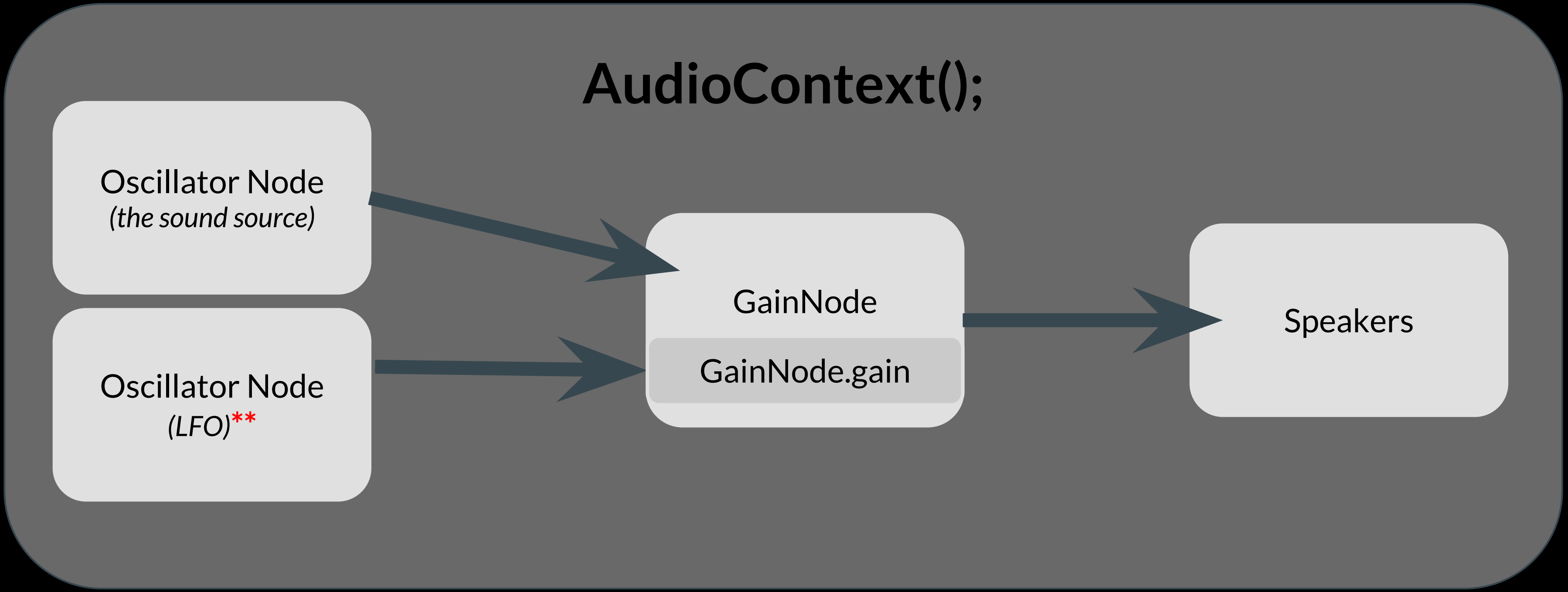
BUT, one of the coolest features?

- Any audioNode can connect to any other node’s Params.

What does this mean?

- You can connect the output of an oscillator Node (set to a low frequency) to a GainNode’s “gain” param to control volume of another sound source.

Tremolo Effect



**An LFO is a Low Frequency Oscillator.

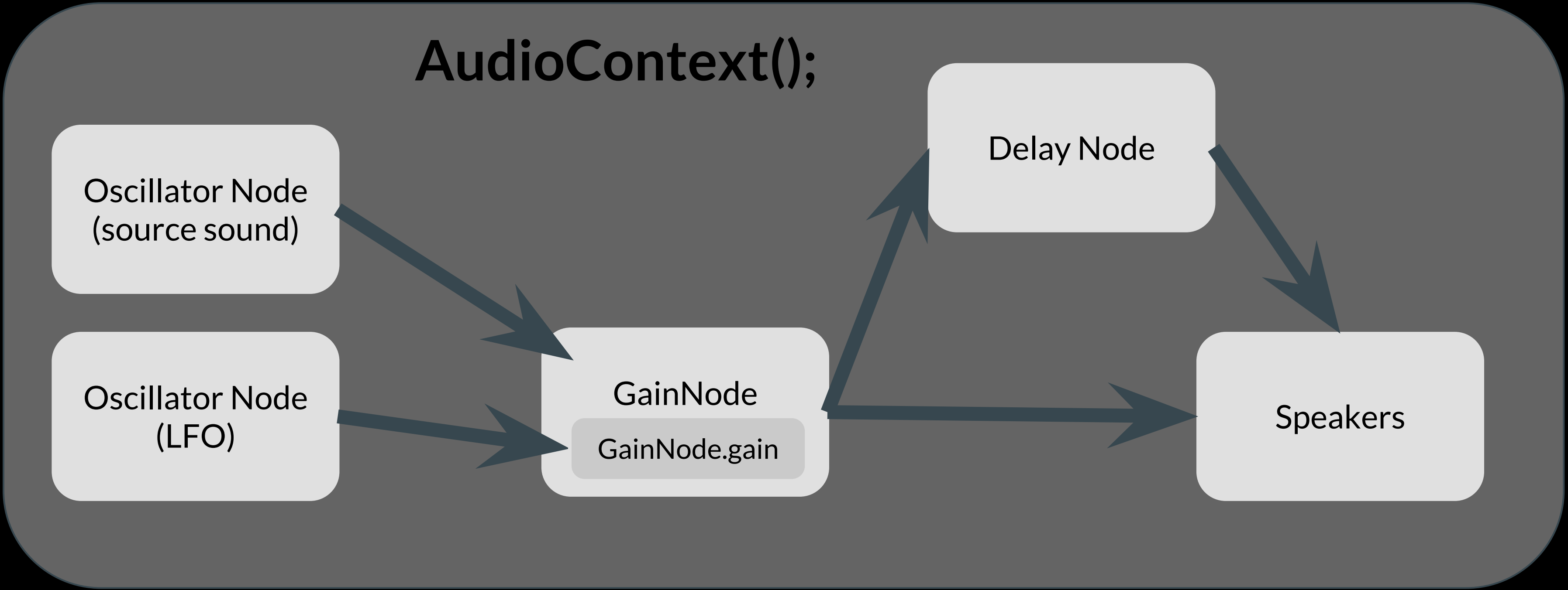
- An oscillator that functions at the frequencies from 0-20Hz, or below the threshold of human hearing.

Tremolo Effect

Javascript	Description
`let context = new AudioContext();`	
`let osc = context.createOscillator();`	
`let gain = context.createGain();`	
`let LFO = context.createOscillator();`	Creates the LFO Oscillator Node
`LFO.frequency.value = 5;`	Assigns a frequency (hertz)
`LFO.type = "sine";`	Assigns a wave shape
`LFO.connect(gain.gain);`	Connects LFO to the gain's gain param
`osc.connect(gain);`	Connects Oscillator to gain
`gain.connect(context.destination);`	Connects VCA to speakers
`LFO.start();`	Starts the LFO
`osc.start();`	Starts the oscillator

Delay Effect

This representation creates a tremolo oscillator (taken from the previous example) and applies a delay to it.



Delay Effect

Javascript

```
let context = new AudioContext();

let osc = context.createOscillator();
let gain = context.createGain();

let LFO = context.createOscillator();
        LFO.frequency.value = 1;
        LFO.type = "square";

let delay = context.createDelay(2.0);
        delay.delayTime.value = 1.75;

LFO.connect(gain.gain);
osc.connect(gain);
gain.connect(delay);

gain.connect(context.destination);
delay.connect(context.destination);

LFO.start();
osc.start();
```

Description

Creates a DelayNode (in this example - max value of 2 seconds **)
Assigns the delay time in seconds (in this example 1.75 seconds)

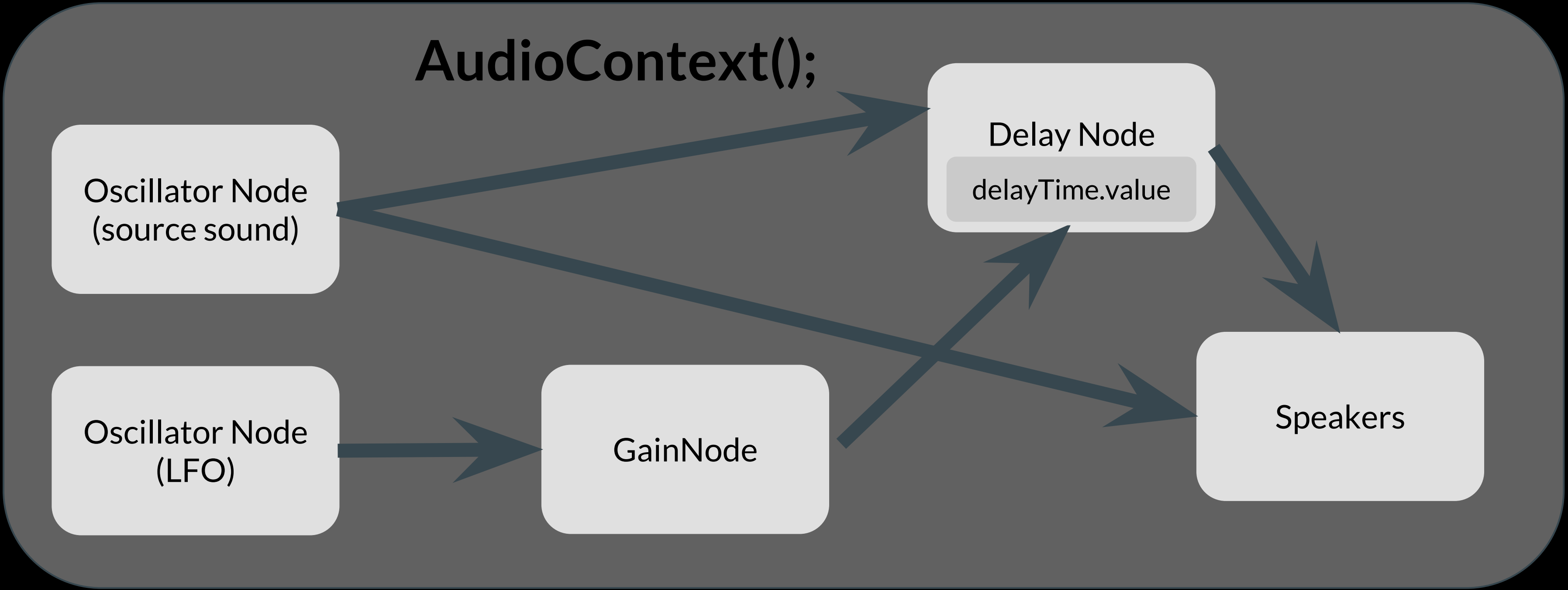
Connects gain to the delay

Connects gain to speakers (“dry” signal / no delay)
Connects delay to speakers (“wet” signal / with delay)

**Delay Time is expressed in seconds, its minimal value is 0, and its maximal value is defined by the argument of the AudioContext.createDelay()

Flanger & Chorus Effect

The difference between Chorus and Flanger is that
Chorus uses a longer delay between the two signals.



Flanger & Chorus Effect

```
let context = new AudioContext();

let osc = context.createOscillator();
	osc.type = "triangle";

let LFO = context.createOscillator();
	LFO.type = "sine";
	LFO.frequency.value = .02;

let gain = context.createGain();
	gain.gain.value = .002;

let delay = context.createDelay();
	delay.delayTime.value = .02;

osc.connect(delay);
LFO.connect(gain);
gain.connect(delay.delayTime);
osc.connect(context.destination);
delay.connect(context.destination);

osc.start();
LFO.start();
```

The delay for a flanger effect is typically less than 20 milliseconds (.002).

To create a Chorus effect in this example, change this value to **1.5;**

AudioBufferSource

Used to load sound files (.wav, .mp3, etc)

XMLHttpRequest (XHR) gets the sound files.

Audio file data is binary, so you assign the XHR’s ‘responseType’ to an ‘arraybuffer’.

An ArrayBuffer is a generic container for binary data. The buffer allows you to replay sounds repeatedly without needing to reload them.

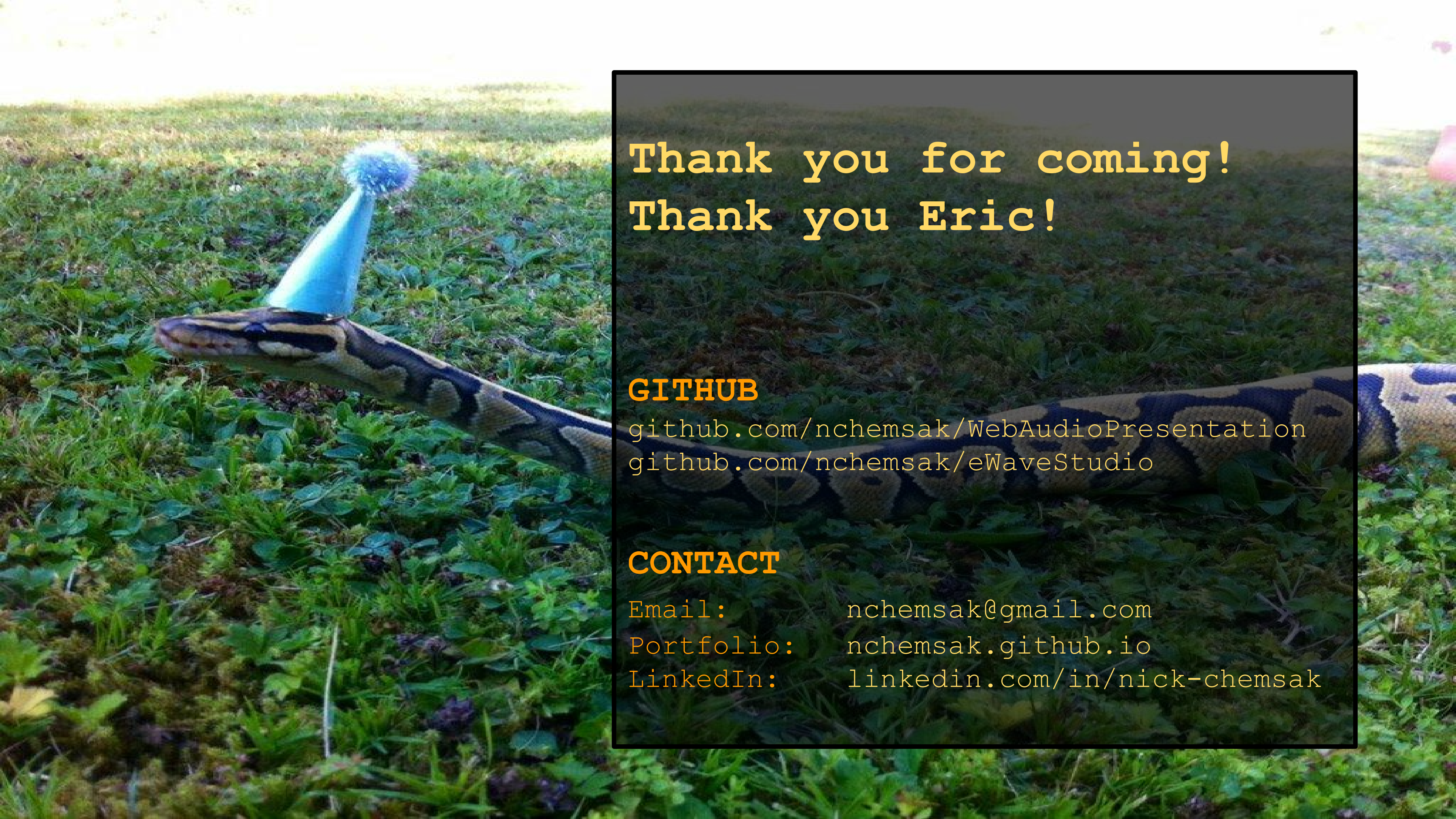
Decode asynchronously using the AudioContext.decodeAudioData() method & JavaScript Promises.

AudioBufferSource

```
let context = new AudioContext();

let url = "audio/snare.wav";
let xhr = new XMLHttpRequest();
          xhr.open('GET', url, true);
          xhr.responseType = 'arraybuffer';

xhr.onload = function() {
 context.decodeAudioData(xhr.response)
  .then(function(buffer) {
   myBuffer = buffer;
  });
}
xhr.send();

function play(event) {
 source = context.createBufferSource();
 source.buffer = event;
 source.connect(context.destination);
 source.start();
}
function stop() {
 if (source) {
  source.stop();
 }
}
```

path of audio file you want to be played.

Decode:
Once the (undecoded) audio file data has been received, it can be decoded using JS Promises.


Thank you for coming!
Thank you Eric!
GITHUB
github.com/nchemsak/WebAudioPresentation
github.com/nchemsak/eWaveStudio
CONTACT
Email: nchemsak@gmail.com
Portfolio: nchemsak.github.io
LinkedIn: linkedin.com/in/nick-chemsak