Game Architecture

Audio

Today’s Agenda

- What is sound and how do we represent it?
- Attributes of sound in 3D
- Sound engine architecture
- Ad hoc methods

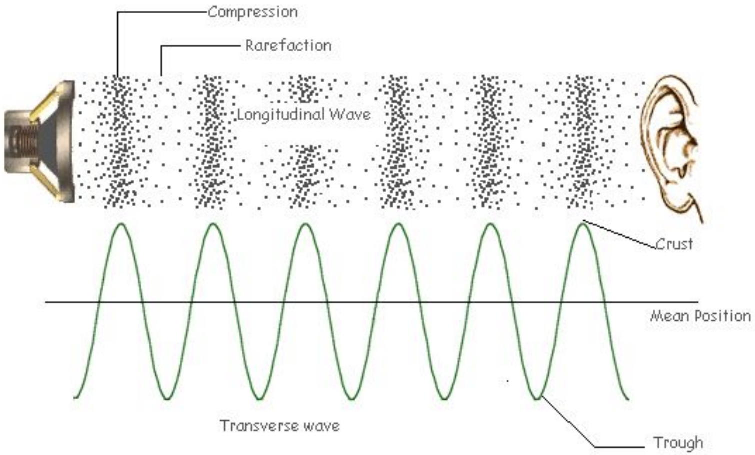
“The secret to great graphics
is great audio.”

What is sound?

- Delta in local atmospheric pressure
  - Waves oscillate in propagation direction
- Constant tone is periodic sound
  - Frequency (Hz) is 1 / Period
- Pitch is 1 / Longest_Period
  - Fundamental harmonic
- Loudness is root mean square of pressure
  - Decibels are units of loudness on log scale

What is sound?

- **Delta in local atmospheric pressure**
  - **Waves oscillate in propagation direction**
- Constant tone is periodic sound
  - Frequency (Hz) is 1 / Period
- Pitch is 1 / Longest_Period
  - Fundamental harmonic
- Loudness is root mean square of pressure
  - Decibels are units of loudness on log scale



What is sound?

- Delta in local atmospheric pressure
  - Waves oscillate in propagation direction
- **Constant tone is periodic sound**
  - **Frequency (Hz) is 1 / Period**
- Pitch is 1 / Longest_Period
  - Fundamental harmonic
- Loudness is root mean square of pressure
  - Decibels are units of loudness on log scale

Typical adult can hear:

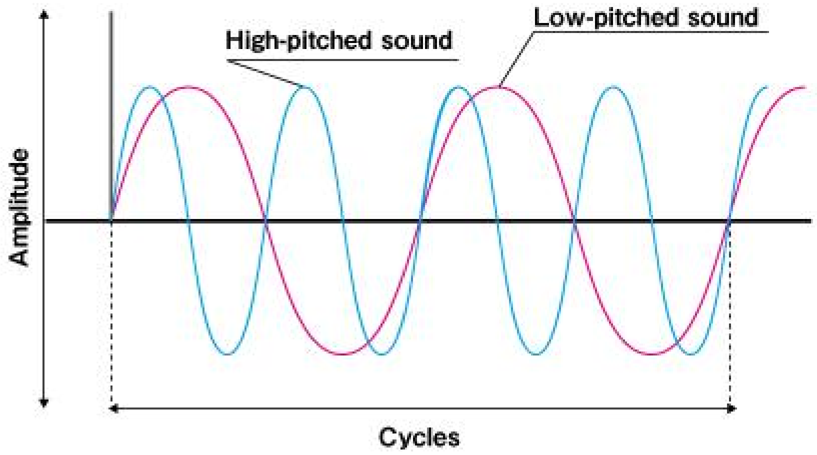
- From 20 Hz up to 20 kHz
- Most sensitive from 2 kHz to 5 kHz
  - Perceptually loudest

Dogs: 67 Hz - 45 kHz

Some bats: Up to 200 kHz!?

What is sound?

- Delta in local atmospheric pressure
    - Waves oscillate in propagation direction
- Constant tone is periodic sound
    - Frequency (Hz) is 1 / Period
- **Pitch is 1 / Longest_Period**
    - **Fundamental harmonic**
- Loudness is root mean square of pressure
    - Decibels are units of loudness on log scale



What is sound?

- Delta in local atmospheric pressure
  - Waves oscillate in propagation direction
- Constant tone is periodic sound
  - Frequency (Hz) is 1 / Period
- Pitch is 1 / Longest_Period
  - Fundamental harmonic
- **Loudness is root mean square of pressure**
  - **Decibels are units of loudness on log scale**

$$x_{\text{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)}$$

$$\overset{\text{Standard pressure level}}{\mathrm{SPL}} = 20\log_{10}\frac{\overset{\text{Pressure}}{P}}{\underset{\text{Standard reference pressure}}{P_0}}\,\mathrm{dB}$$

Representing sound

- Linear time-invariant system
- Convolution of time shifted impulses
- Samples
- Formats

Representing sound

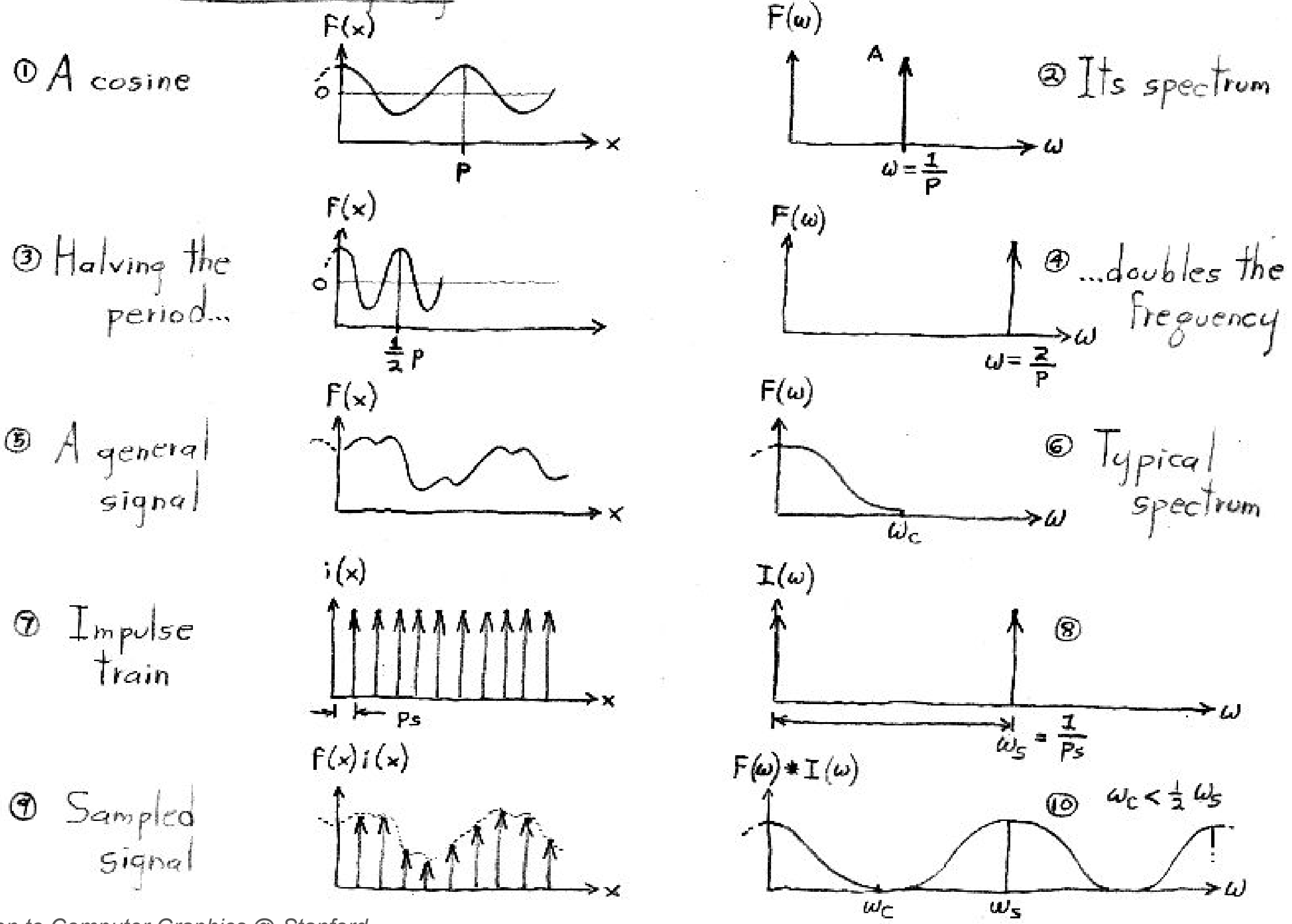
- **Linear time-invariant system**
- Convolution of time shifted impulses
- Samples
- Formats

- Linear
  - Response to N stimuli is the sum of the responses which would have been caused by each stimulus individually
  - If A produces response X and input B produces response Y then input (A + B) produces response (X + Y)
  - *(Wikipedia)*
- Time-invariant
  - Time shifting input shifts output equally

Representing sound

- Linear time-invariant system
- **Convolution of time shifted impulses**
- Samples
- Formats

Ideal sampling:



Representing sound

- Linear time-invariant system
- Convolution of time shifted impulses
- **Samples**
- Formats

- Sampling rate
    - 8 kHz - Telephone
    - 32 kHz - Wii hardware output
    - 44.1 kHz - CD standard
    - 48 kHz - Modern standard output
    - Nyquist-Shannon sampling theorem
        - Use rate double highest frequency for lossless sampling
- Bits per sample
    - 16 - Typical input/output
    - 32 - Typical intermediate
- Channel count
    - 1 - mono
    - 2 - stereo
    - N - Surround, language tracks, other?

Representing sound

- Linear time-invariant system
- Convolution of time shifted impulses
- Samples
- **Formats**

- PCM
  - Pulse code modulation
  - Size = Rate * Bits * Channels
  - Compute = Minimum
- ADPCM
  - Adaptive differential pulse code modulation
  - Adaptively storing deltas between samples
  - Typically fixes bits_per_sample = 4
  - Compute = Small
- MP3, Ogg Vorbis, and friends
  - Variable bit rate, lossy
  - Compute = Relatively expensive

Attributes of Sound in 3D

- Distance attenuation
- Doppler
- Panning
- Reverb
- Models of ears and head

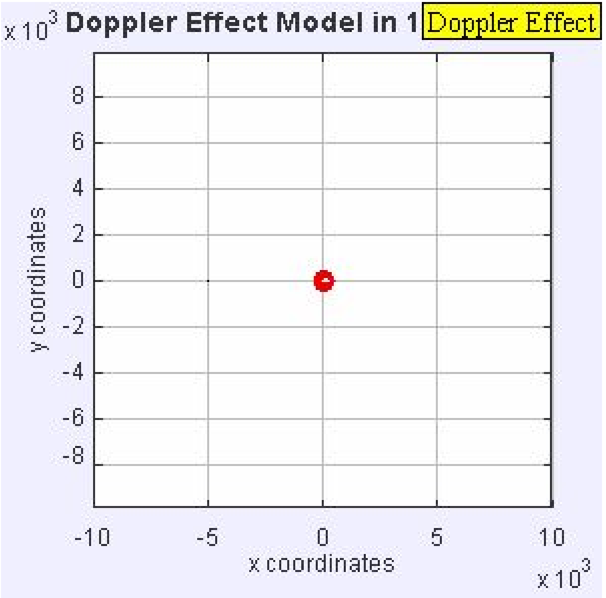
Attributes of Sound in 3D

- **Distance attenuation**
- Doppler
- Panning
- Reverb
- Models of ears and head

- Linear falloff: 1 / distance
  - Drop to zero at some max distance

Attributes of Sound in 3D

- Distance attenuation
- **Doppler**
- Panning
- Reverb
- Models of ears and head



$$f' = \left(\frac{v+v'}{v-v_s}\right)f_s$$

Attributes of Sound in 3D

- Distance attenuation
- Doppler
- **Panning**
- Reverb
- Models of ears and head

Constant power pan law:

- A_left = A * sin(π/2 * pan)
- A_right = A * cos(π/2 * pan)

The "3 dB rule":

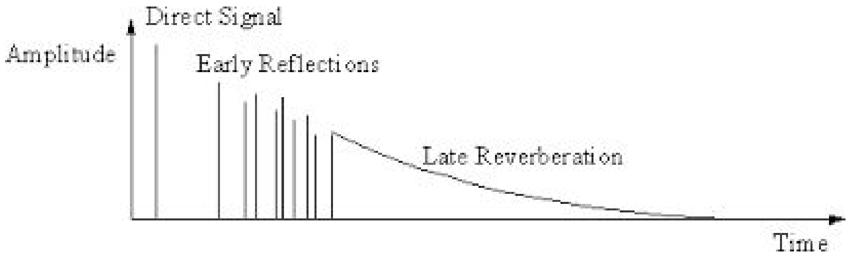
- If sound is coming out of 2 speakers equally, reduce by 3 dB.

Attributes of Sound in 3D

- Distance attenuation
- Doppler
- Panning
- **Reverb**
- Models of ears and head

Components of a sound:

- Dry - sound directly from emitter
- Wet - sound reflected in environment
  - This is reverb.



Sound designers pad their offices to eliminate wet sound.

Attributes of Sound in 3D

- Distance attenuation
- Doppler
- Panning
- Reverb
- **Models of ears and head**

Brain knows a lot about environment from:

- Delta b/w left and right ears
- Delta b/w direct sound and first reflection

We may want to model subtle effects of:

- Ears (their shape and their number)
- Distortion caused by sound traveling through skull (head related transfer function, or HRTF)

But then again, player's speaker config probably has a greater impact on the experience...

Sound engine architecture: Example API

```
// The system
snd_system snd_system_create();
void snd_system_destroy(snd_system system);
void snd_system_set_listener(vec3 pos, quat orient);

// Banks
snd_bank snd_bank_load(snd_system system, string file_name);
void snd_bank_unload(snd_bank bank);

// Groups
snd_group snd_group_create(snd_system system, string name, snd_group parent);
void snd_group_destroy(snd_group group);
void snd_group_set_volume(snd_group group, float volume);
void snd_group_pause(snd_group group, bool paused);

// Instances
snd_instance_handle snd_play(snd_group group, snd_cue cue);
void snd_stop(snd_instance_handle sound);
```

Sound engine architecture: Example API

Owns audio resources and pipeline.

```
// The system
snd_system snd_system_create();
void snd_system_destroy(snd_system system);
void snd_system_set_listener(vec3 pos, quat orient);

// Banks
snd_bank snd_bank_load(snd_system system, string file_name);
void snd_bank_unload(snd_bank bank);

// Groups
snd_group snd_group_create(snd_system system, string name, snd_group parent);
void snd_group_destroy(snd_group group);
void snd_group_set_volume(snd_group group, float volume);
void snd_group_pause(snd_group group, bool paused);

// Instances
snd_instance_handle snd_play(snd_group group, snd_cue cue);
void snd_stop(snd_instance_handle sound);
```

Sound engine architecture: Example API

```
// The system
snd_system snd_system_create();
void ;
void t orient);
```

A collection of sound clips (compressed PCM files) managed as a unit.

```
// Banks
snd_bank snd_bank_load(snd_system system, string file_name);
void snd_bank_unload(snd_bank bank);

// Groups
snd_group snd_group_create(snd_system system, string name, snd_group parent);
void snd_group_destroy(snd_group group);
void snd_group_set_volume(snd_group group, float volume);
void snd_group_pause(snd_group group, bool paused);

// Instances
snd_instance_handle snd_play(snd_group group, snd_cue cue);
void snd_stop(snd_instance_handle sound);
```

Sound engine architecture: Example API

```
// The system
snd_system snd_system_create();
void snd_system_destroy(snd_system system);
void snd_system_set_listener(vec3 pos, quat orient);

// Banks
snd_bank snd_bank_load(snd_system system, string file_name);
```

Hierarchical, high level mixing control.

```
// Groups
snd_group snd_group_create(snd_system system, string name, snd_group parent);
void snd_group_destroy(snd_group group);
void snd_group_set_volume(snd_group group, float volume);
void snd_group_pause(snd_group group, bool paused);

// Instances
snd_instance_handle snd_play(snd_group group, snd_cue cue);
void snd_stop(snd_instance_handle sound);
```

Sound engine architecture: Example API

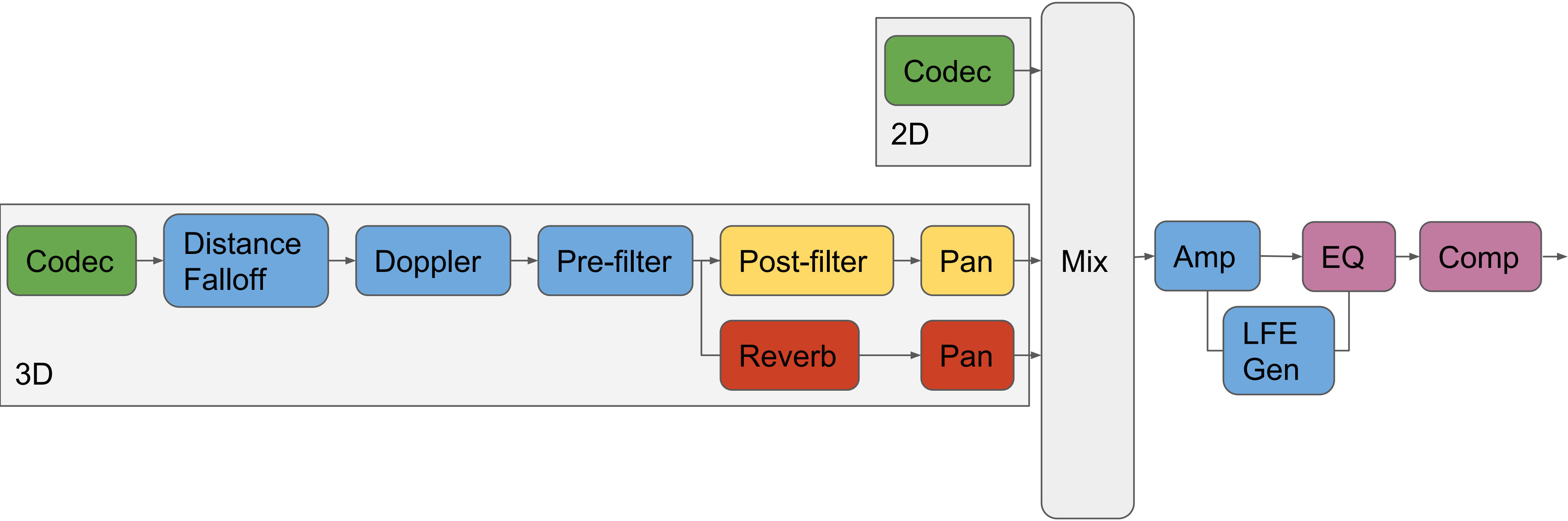
```
// The system
snd_system snd_system_create();
void snd_system_destroy(snd_system system);
void s                                         ient);

// Ba
snd_b                                         g file_name);
void

// Gr
snd_g                                         tring name, snd_group parent);
void
void                                          t volume);
void s                                        sed);

// Instances
snd_instance_handle snd_play(snd_group group, snd_cue cue);
void snd_stop(snd_instance_handle sound);
```

Sound cues are collections of sound clips with some metadata (loop points, mixing information, clip selection criteria).

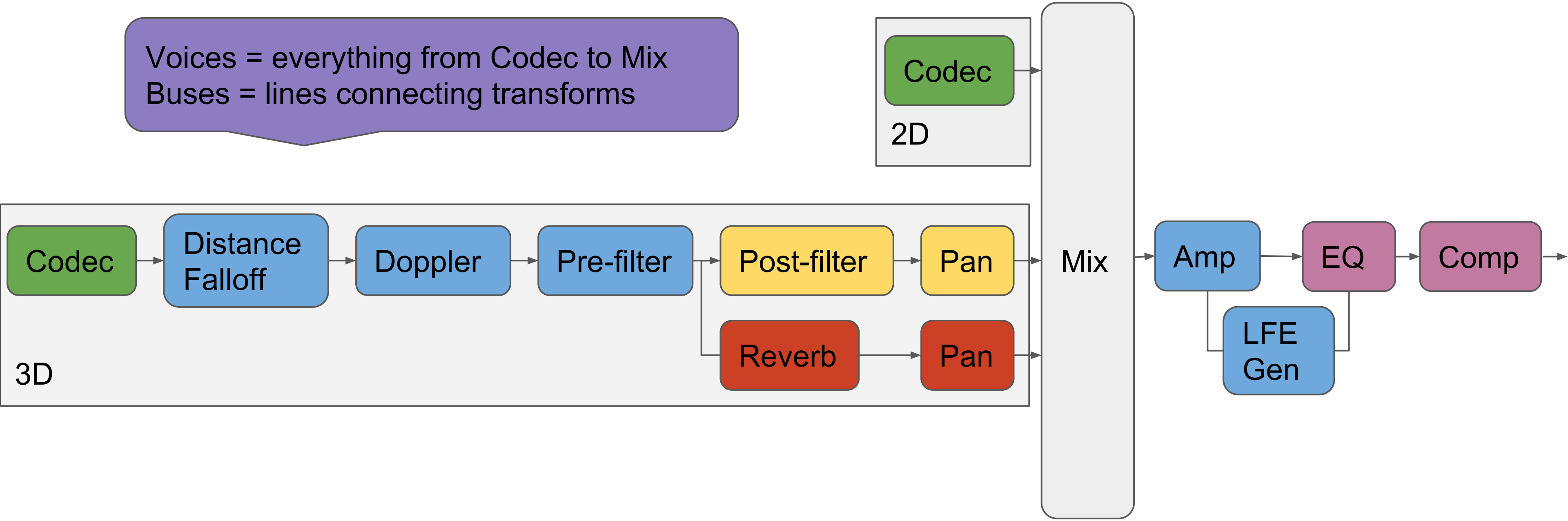
Playing a sound cue allocates a virtual voice and selects a clip.

Sound engine activates virtual voices in range of the listener up to some cap. Active voices go through the mixing pipeline.

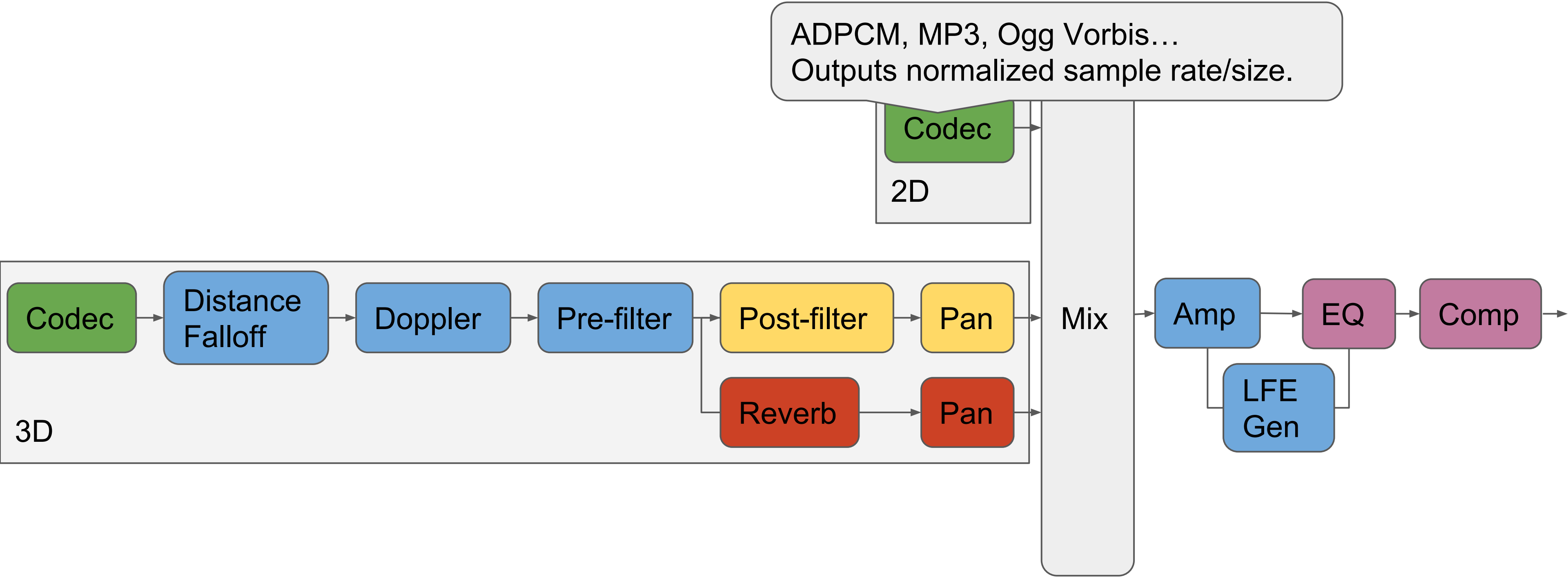
Sound engine architecture: Example Pipeline



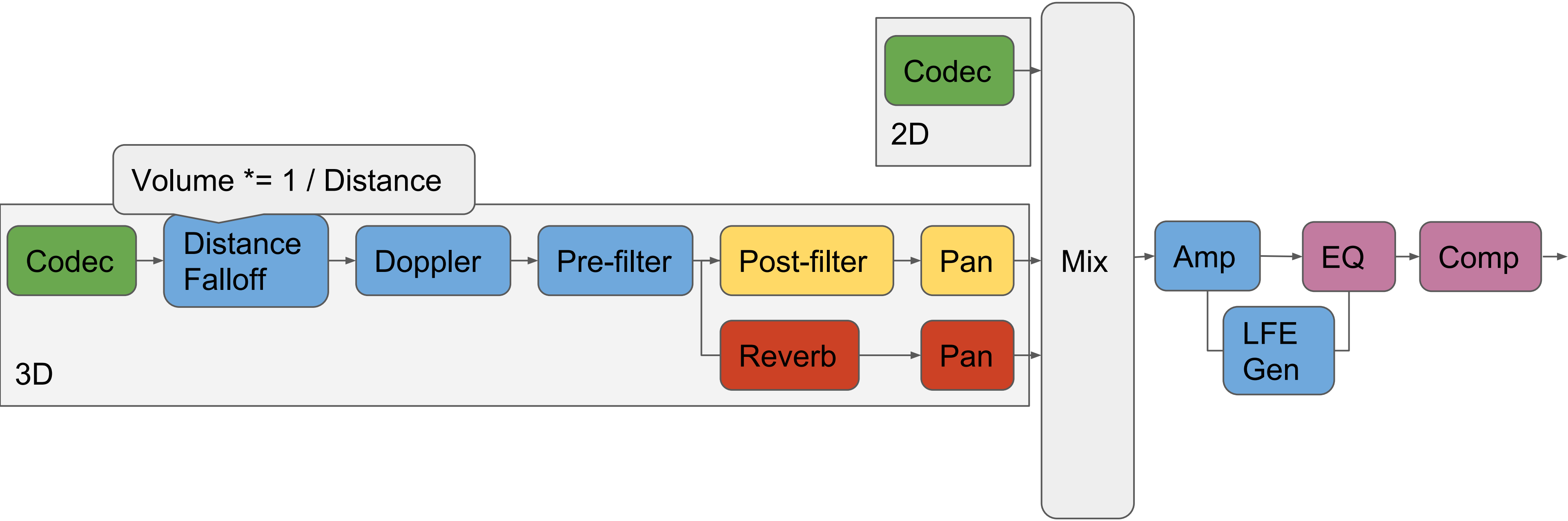
Sound engine architecture: Example Pipeline



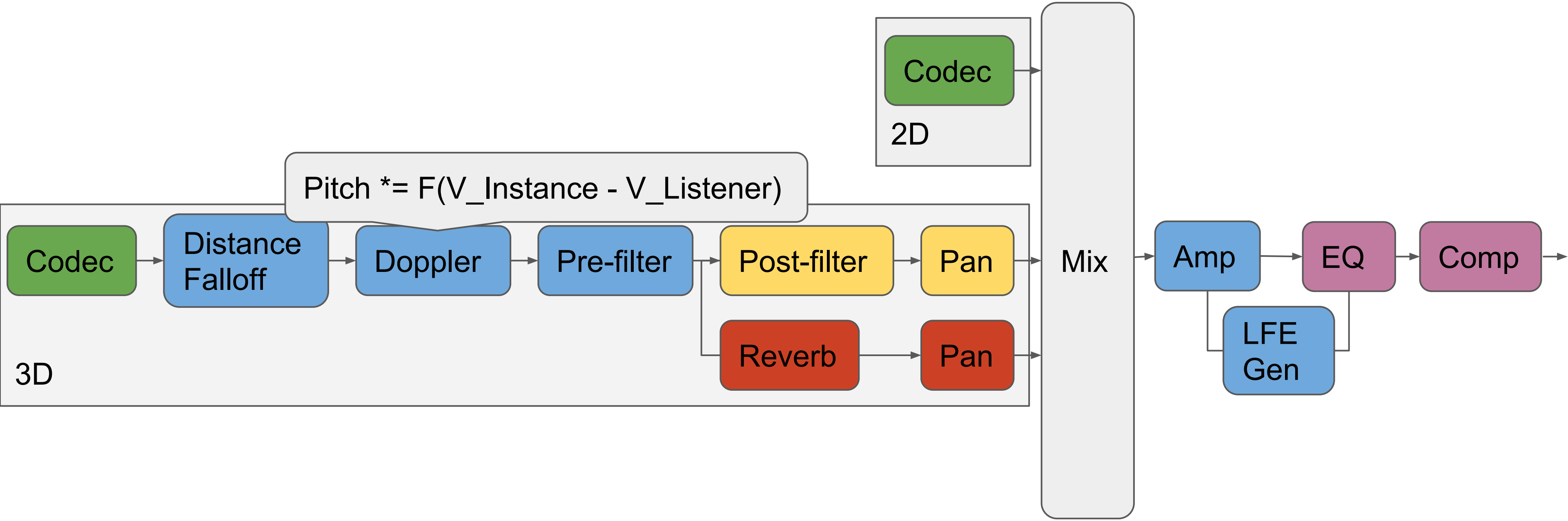
Sound engine architecture: Example Pipeline



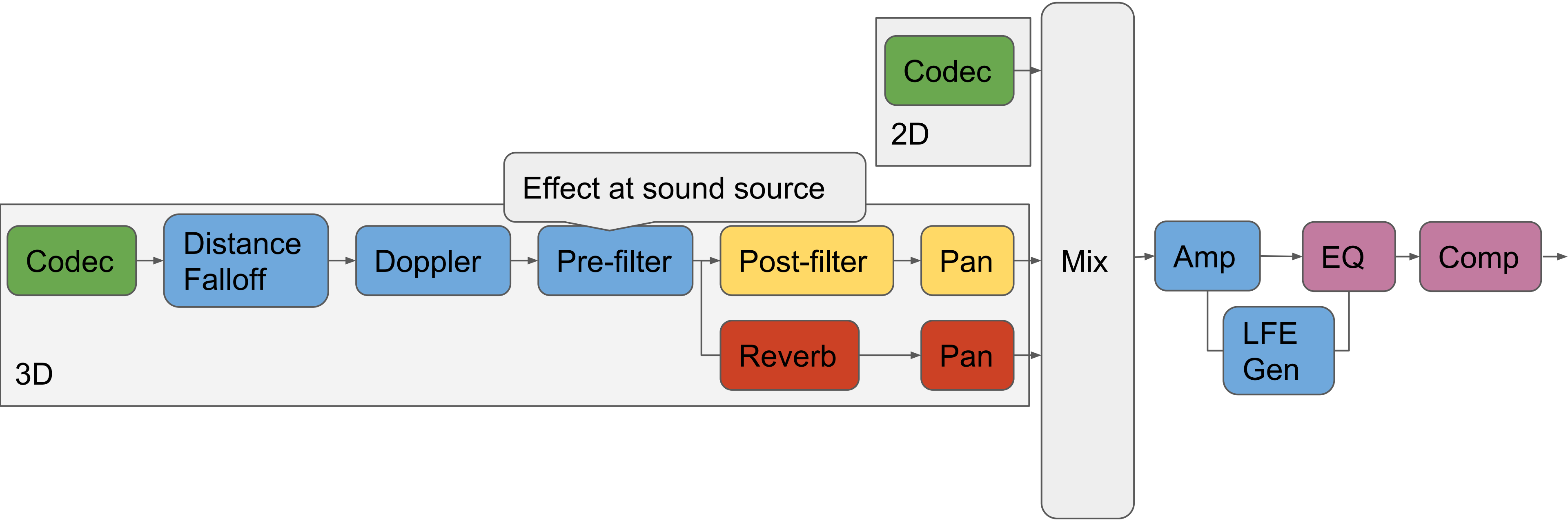
Sound engine architecture: Example Pipeline



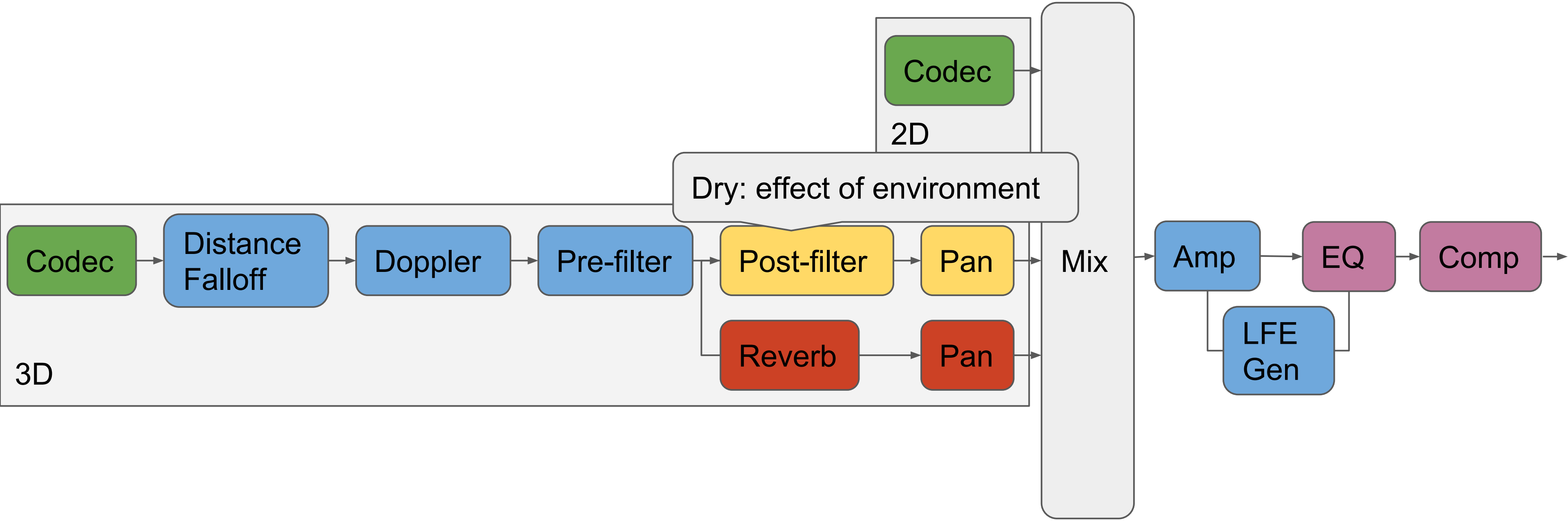
Sound engine architecture: Example Pipeline



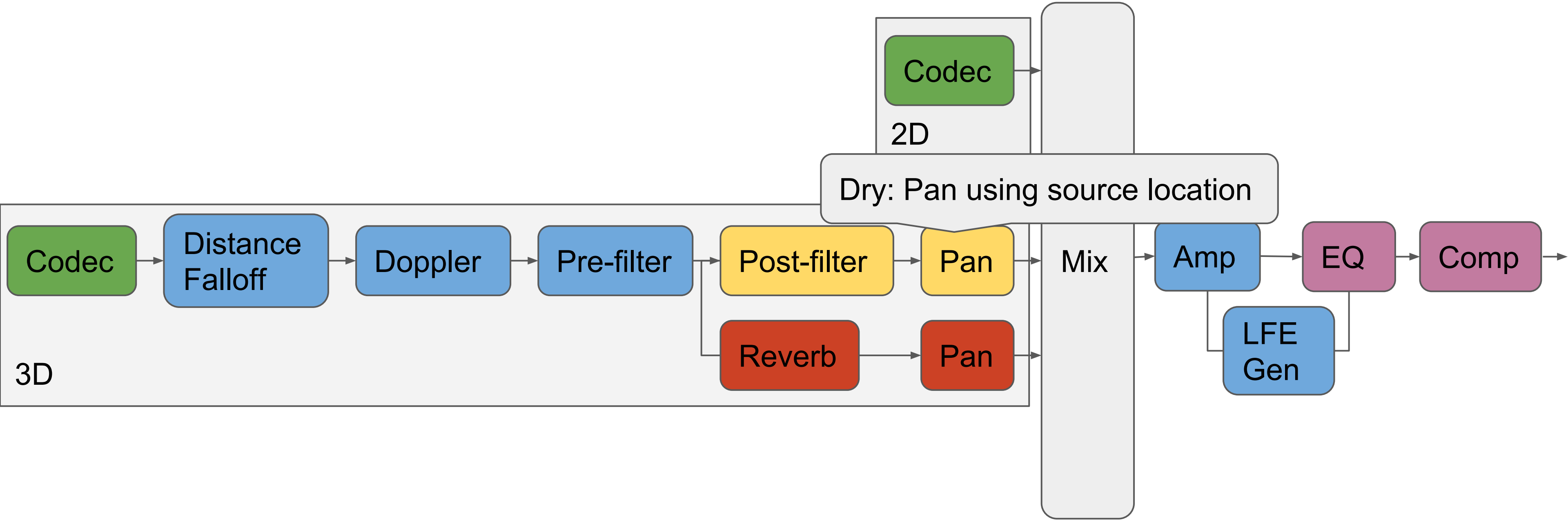
Sound engine architecture: Example Pipeline



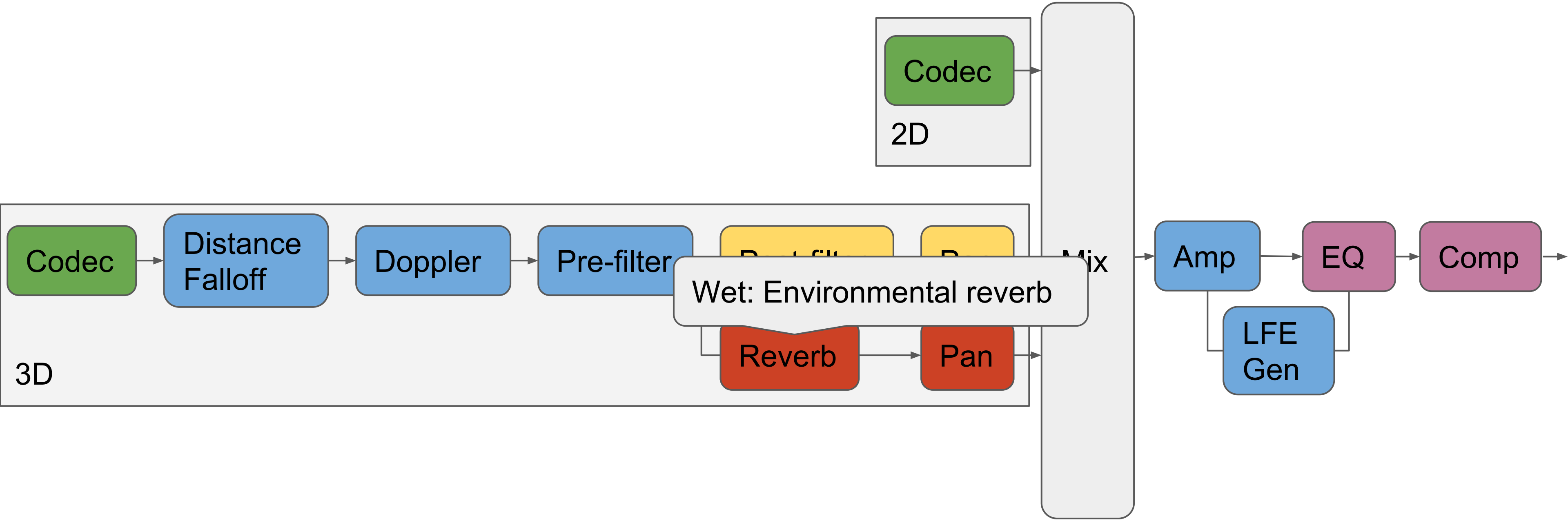
Sound engine architecture: Example Pipeline



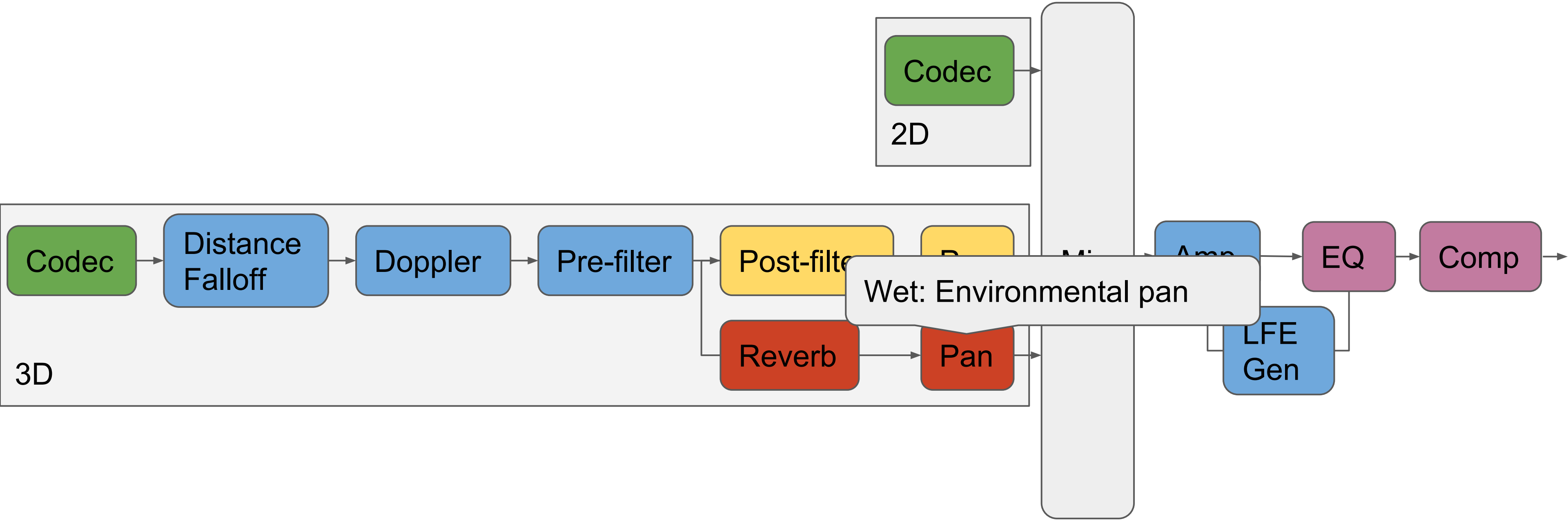
Sound engine architecture: Example Pipeline



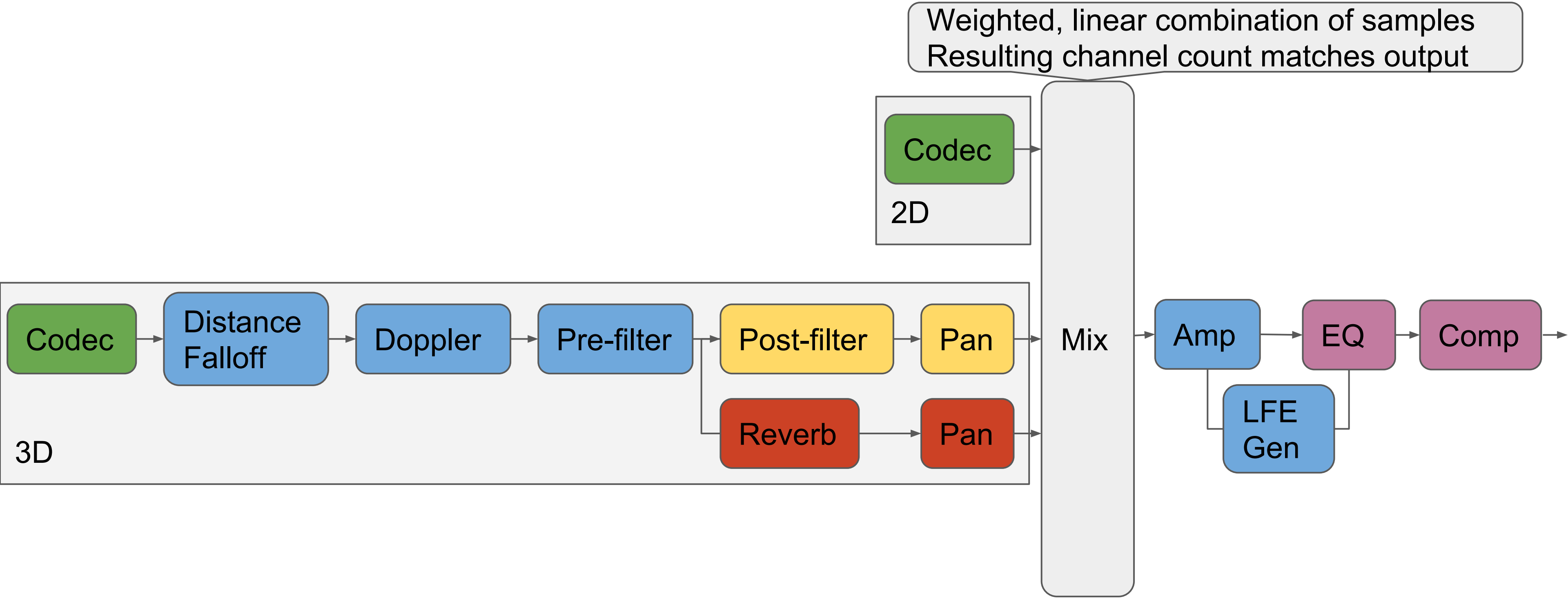
Sound engine architecture: Example Pipeline



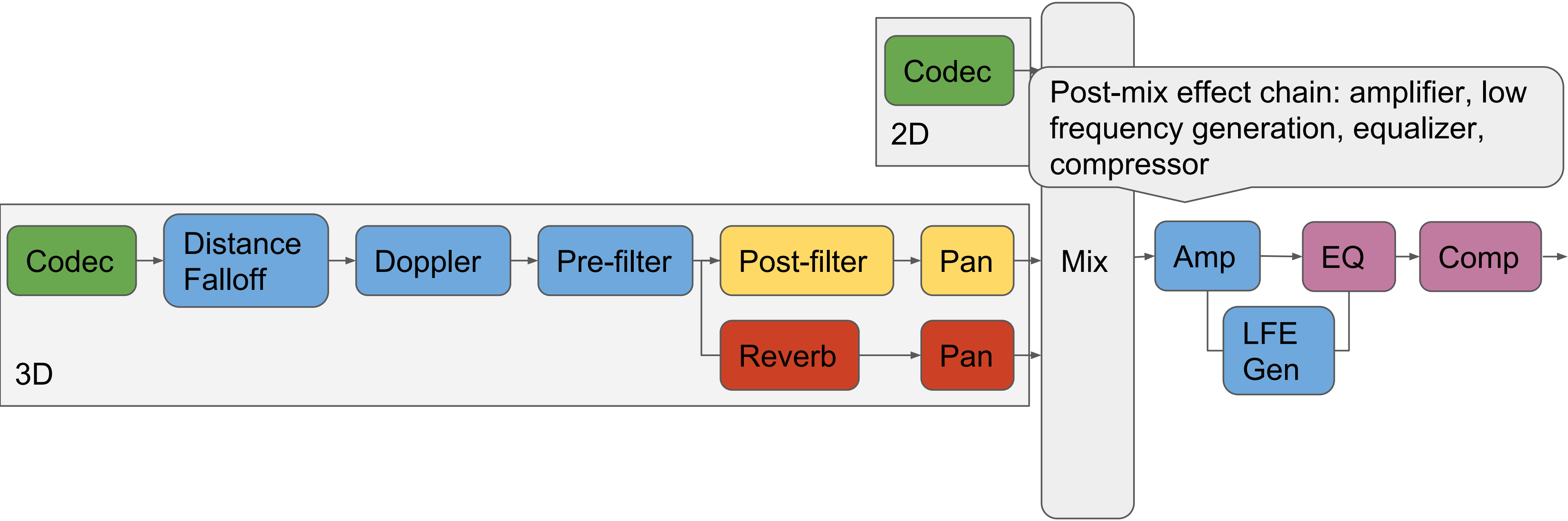
Sound engine architecture: Example Pipeline



Sound engine architecture: Example Pipeline



Sound engine architecture: Example Pipeline



Sound engine architecture: Notes

Some additional considerations:

- Buffering and latency
- Dynamic volume control
- Voice management
- Streaming

Sound engine architecture: Notes

Some additional considerations:

- **Buffering and latency**
- Dynamic volume control
- Voice management
- Streaming

Typically ring buffer stores final output:

- Push newly mixed audio on tail
- DAC consumes from head

Ring buffer size:

- Larger = more latency
- Smaller = greater chance of dropout

Sound engine architecture: Notes

Some additional considerations:

- Buffering and latency
- **Dynamic volume control**
- Voice management
- Streaming

Gameplay considerations:

- Hear an NPC whispering from far away
- Hear important cues over gun fire
- Quiet and loud sections

Must dynamically adjust volume:

- Duck environmental volume when important VO is happening
- Smoothly balance total sound pressure
- Optionally override distance attenuation

Sound engine architecture: Notes

Some additional considerations:

- Buffering and latency
- Dynamic volume control
- **Voice management**
- Streaming

- Hundreds / thousands of sound sources
- Can’t afford to mix them all
- Prioritize sounds
- Playing new high priority sounds to steal voices from low priority sounds

Sound engine architecture: Notes

Some additional considerations:

- Buffering and latency
- Dynamic volume control
- Voice management
- **Streaming**

- Audio is large
- For large clips:
  - Don’t have the whole thing resident
  - Stream into a ring buffer
- But watch out for:
  - Bandwidth limitations
  - Seek times on physical disks

Ad hoc methods

Proper analytic models are expensive. Hack it.

- Sources that get too close to listeners
- Sounds from above and below
- Handling sound occluded by environment

Ad hoc methods

Proper analytic models are expensive. Hack it.

- **Sources that get too close to listeners**
- Sounds from above and below
- Handling sound occluded by environment

Far from listener:

- Source is a point and panning is simple

Close to listener:

- Listener becomes a volume
- Source becomes a volume
- Intersection of volumes defines how sound should be panned
- Quick and dirty approximations are fine

Ad hoc methods

Proper analytic models are expensive. Hack it.

- Sources that get too close to listeners
- **Sounds from above and below**
- Handling sound occluded by environment

- 7.1 surround is purely horizontal
- Sound sources are projected onto horizontal plane
- Post-filter & reverb clue listener into verticality

Ad hoc methods

Proper analytic models are expensive. Hack it.

- Sources that get too close to listeners
- Sounds from above and below
- **Handling sound occluded by environment**

Baseline:

- Tag reverb regions in game world

Dynamically modify filters based on:

- Look for LOS blockage between source and listener -- post-filter attenuates dry portion of sound
- Probabilistically model indirect occlusion with raycasts against collision world -- adjust reverb parameters on wet portion of sound

Summary

- Audio as signal in a linear time-invariant system
- Audio in 3D space
- Pseudo-API for a sound system
- Pseudo-pipeline for a sound system
- Convenient hacks and other considerations

End Lecture

http://sol.gfxile.net/soloud/