the gamedesigninitiative at cornell university

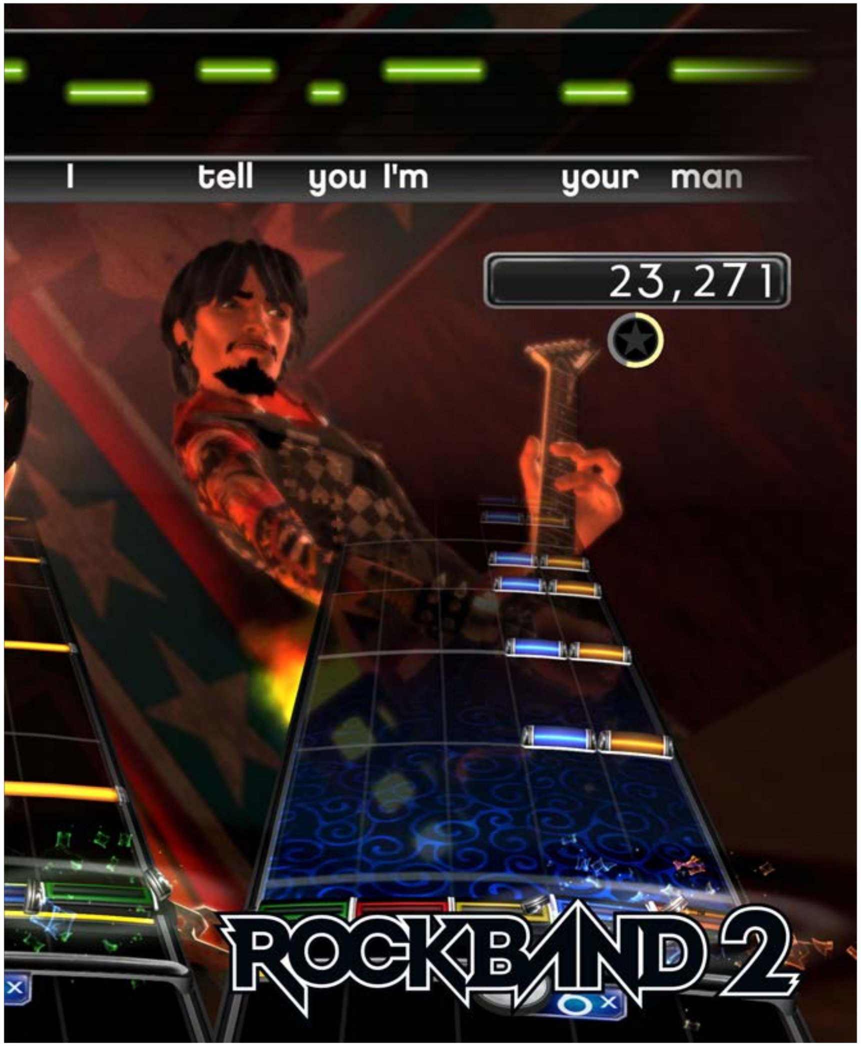
Lecture 28

Game Audio

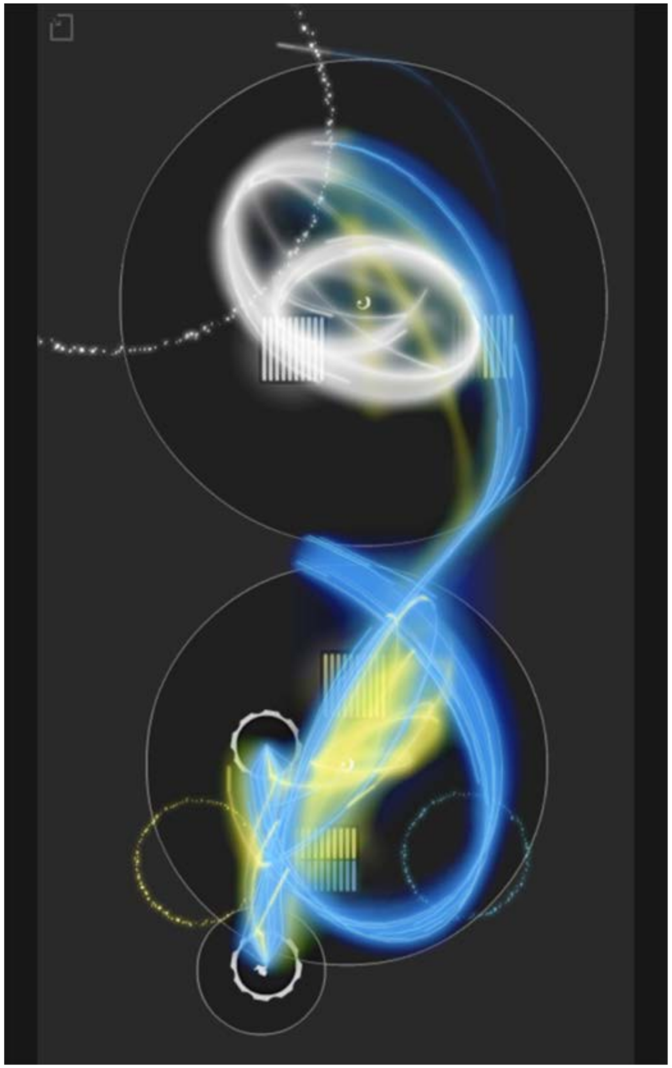
The Role of Audio in Games

Engagement

- **Entertains** the player
  - Music/Soundtrack
- Enhances the **realism**
  - Sound effects
- Establishes **atmosphere**
  - Ambient sounds
- Other reasons?



The Role of Audio in Games



Feedback

- **Indicate** off-screen action
  - Indicate player should move
- **Highlight** on-screen action
  - Call attention to an NPC
- Increase **reaction** time
  - Players react to sound faster
- Other reasons?

History of Sound in Games

Basic Sounds

- Arcade games
- Early handhelds
- Early consoles

Early Sounds: *Wizard of Wor*

History of Sound in Games

Basic Sounds → Recorded Sound Samples

- Arcade games
- Early handhelds
- Early consoles

- Starts w/ MIDI
- 5^{th} generation (Playstation)
- Early PCs

History of Sound in Games

Basic Sounds → Recorded Sound Samples → Some Variability of Samples

- Arcade games
- Early handhelds
- Early consoles

- Starts w/ MIDI
- 5th generation (Playstation)
- Early PCs

- Sample selection
- Volume
- Pitch
- Stereo pan

History of Sound in Games

Basic Sounds → **Recorded Sound Samples** → **Some Variability of Samples** → **More Variability of Samples**

Basic Sounds

- Arcade games
- Early handhelds
- Early consoles

Recorded Sound Samples

- Starts w/ MIDI
- 5th generation (Playstation)
- Early PCs

Some Variability of Samples

- Sample selection
- Volume
- Pitch
- Stereo pan

More Variability of Samples

- Multiple samples
- Reverb models
- Sound filters
- Surround sound

The Technical Challenges

- Sound **formats** are not (really) cross-platform
  - It is not as easy as choosing MP3
  - Android, iOS favor different formats
- Sound playback **APIs** are not standardized
  - LibGDX is a layer over many different APIs
  - So behavior is not the same on all platforms
- Sound playback crosses **frame boundaries**
  - Mixing sound with animation has challenges

File Format vs Data Format

File Format

- The data storage format
  - Has data other than audio
- Many have many encodings
  - .caf holds MP3 *and* PCM
- **Examples:**
  - .mp3, .wav
  - .aac, .mp4, .m4a (Apple)
  - .flac, .ogg (Linux)

Data Format

- The actual audio encoding
  - Basic audio codec
  - Bit rate (# of bits/unit time)
  - Sample rate
    (digitizes an analog signal)
- **Examples:**
  - MP3, Linear PCM
  - AAC, HE-AAC, ALAC
  - FLAC, Vorbis

Data Formats and Platforms

Format	Description	iOS	Android
MP3	You know what this is	Yes	Yes
(HE-)AAC	A lossy codec, Apple's MP3 alternative	Yes	Yes
Linear PCM	Completely uncompressed sound	Yes	Yes
MIDI	**NOT SOUND**; Data for an instrument	Yes	Yes
Vorbis	Xiph.org's alternative to MP3	**Maybe**	Yes
FLAC	Xiph.org's alternative lossless codec	**Maybe**	Yes
ALAC	Apple's lossless codec (but compressed)	Yes	**No**
iLBC	Internet low bit-rate codec (VOIP)	Yes	**No**
IMA4	Super compression for 16 bit audio	Yes	**No**
μ-law	Like PCM, but optimized for speech	Yes	**No**

The Associated File Formats

Format	File Types
MP3	.mp3
(HE-)AAC	.aac, .mp4, .m4a
Linear PCM	.wav
MIDI	.mid

- Any other file format is **not cross-platform**
- Apple/iOS is pushing the .caf file
  - Stands for Core Audio Format
  - Supports MP3, (HE-)AAC, PCM, ALAC, etc…
  - But not cross-platform

The Associated File Formats

Format	File Types
MP3	.mp3
(HE-)AAC	.aac, .mp4, .m4a
Linear PCM	.wav
MIDI	.mid

Limited support due to patent issues

Uncompressed

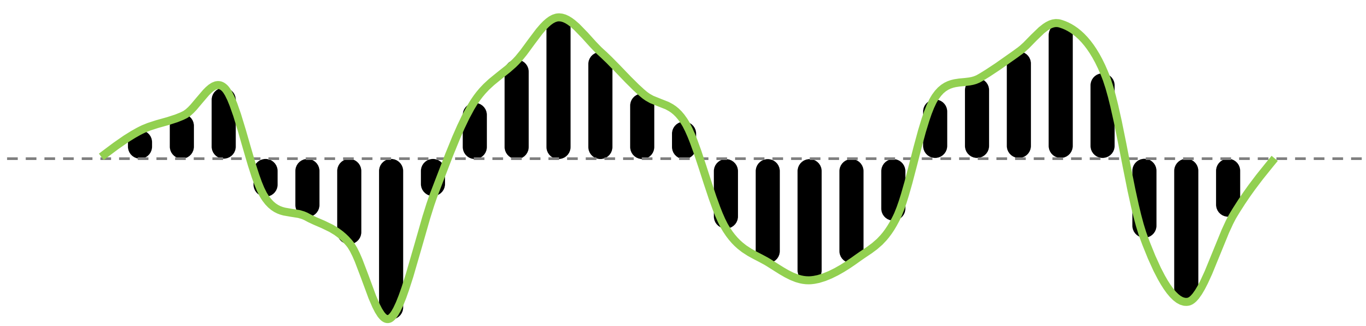
- Any other format is **not** (completely) **cross-platform**
- Apple/iOS is pushing the (prioprietary) .caf file
  - Stands for Core Audio Format
  - Supports MP3, (HE-)AAC, PCM, ALAC, etc…
- OGG has become a popular format for gaming

Linear PCM Format

- Sound data is an array of **sample** values

0.5	0.2	-0.1	0.3	-0.5	0.0	-0.2	-0.2	0.0	-0.6	0.2	-0.3	0.4	0.0

- A sample is an **amplitude** of a sound wave



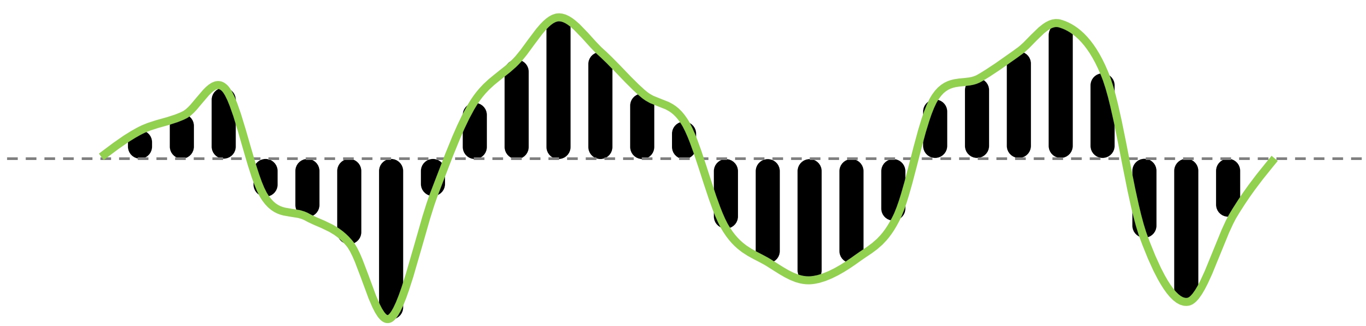
- Values are normalized -1.0 to 1.0 (so they are floats)

Linear PCM Format

- Sound data is an array of **sample** values

0.5	0.2	-0.1	0.3	-0.5	0.0	-0.2	-0.2	0.0	-0.6	0.2	-0.3	0.4	0.0

- A sample is an **amplitude** of a sound wave

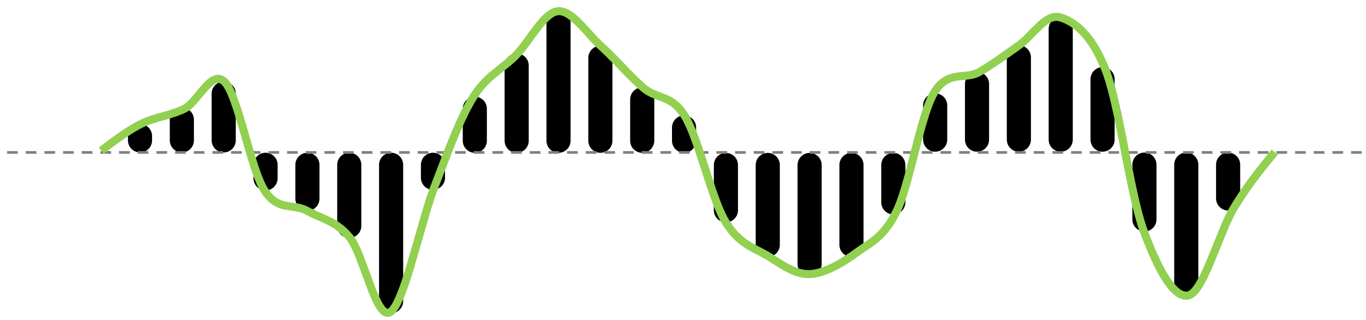


- Sometimes encoded as shorts or bytes MIN to MAX

Linear PCM Format

- Sound data is an array of **sample** values

0.5	0.2	-0.1	0.3	-0.5	0.0	-0.2	-0.2	0.0	-0.6	0.2	-0.3	0.4	0.0

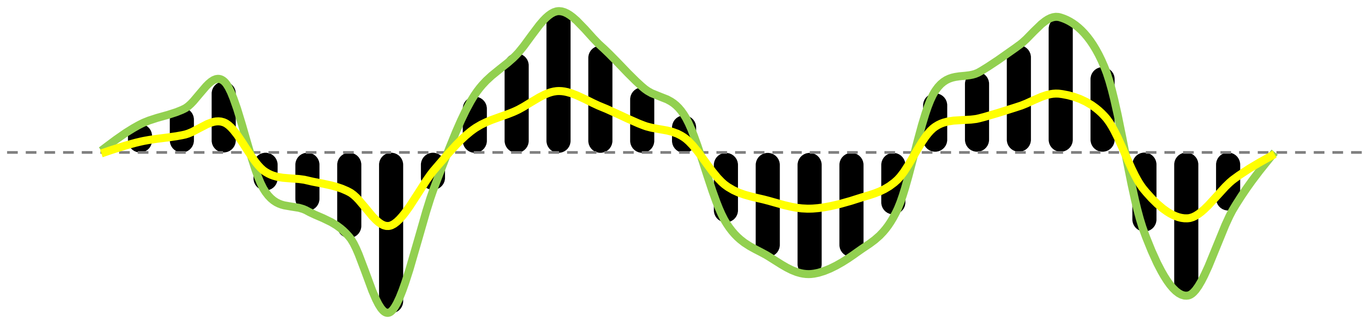


- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume

Linear PCM Format

- Sound data is an array of **sample** values

0.5	0.2	-0.1	0.3	-0.5	0.0	-0.2	-0.2	0.0	-0.6	0.2	-0.3	0.4	0.0



- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume

Linear PCM Format

- Samples are organized into (interleaved) **channels**

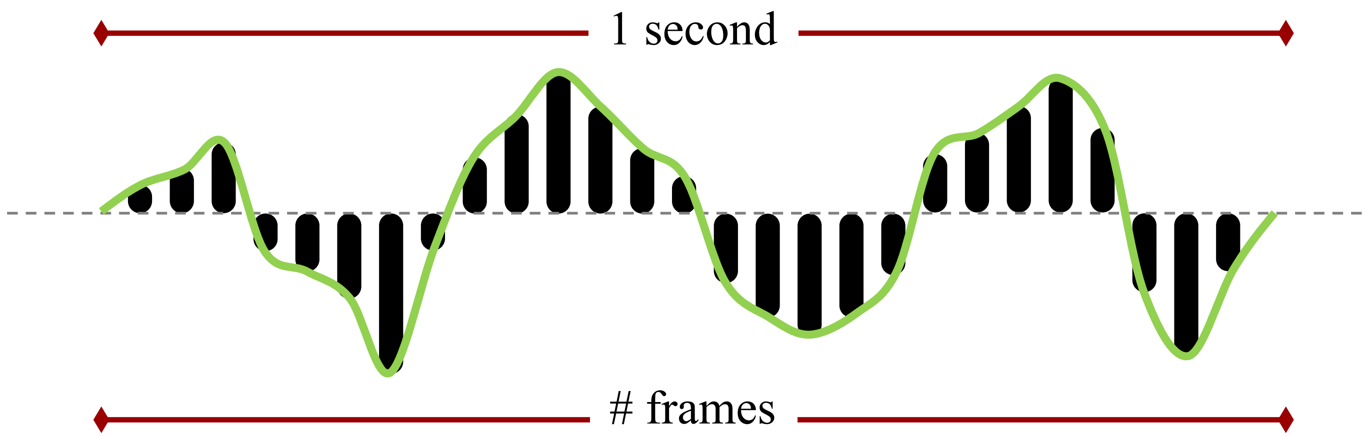
0.5	0.2	-0.1	0.3	-0.5	0.0	-0.2	-0.2	0.0	-0.6	0.2	-0.3	0.4	0.0

frame

- Each channel is essentially a **speaker**
  - Mono sound has one channel
  - Stereo sound has two channels
  - 5.1 surround sound is ***six*** channels
- A **frame** is set of simultaneous samples
  - Each sample is in a separate frame

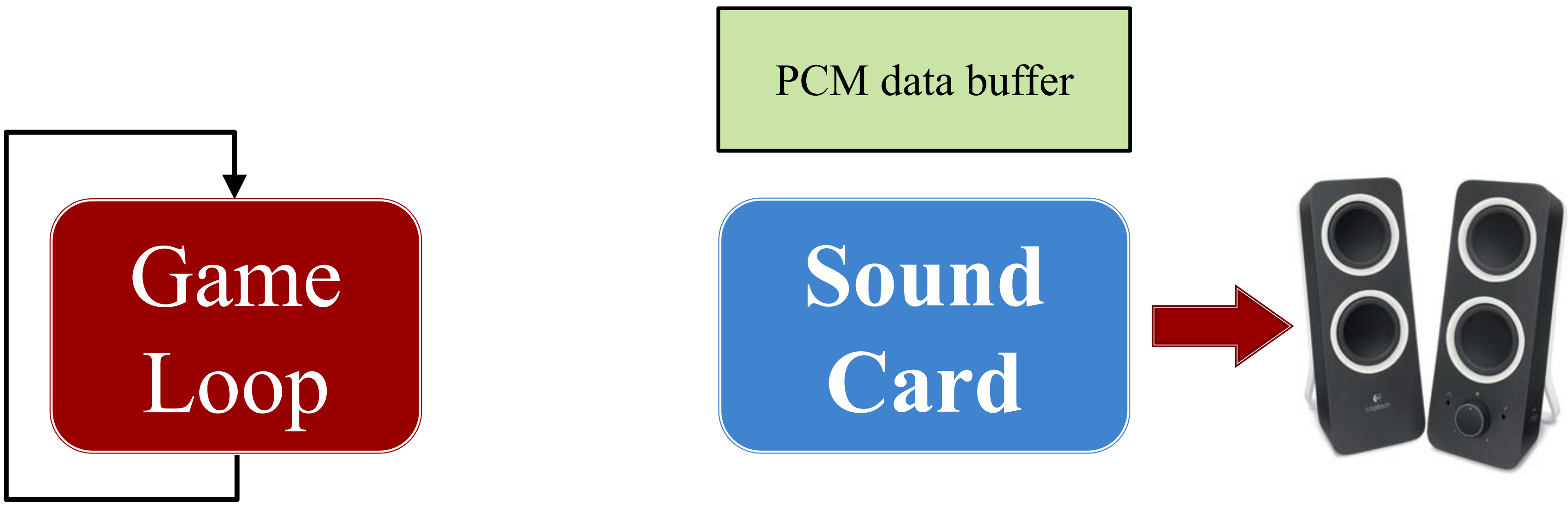
Linear PCM Format

- The sample rate is frames per second

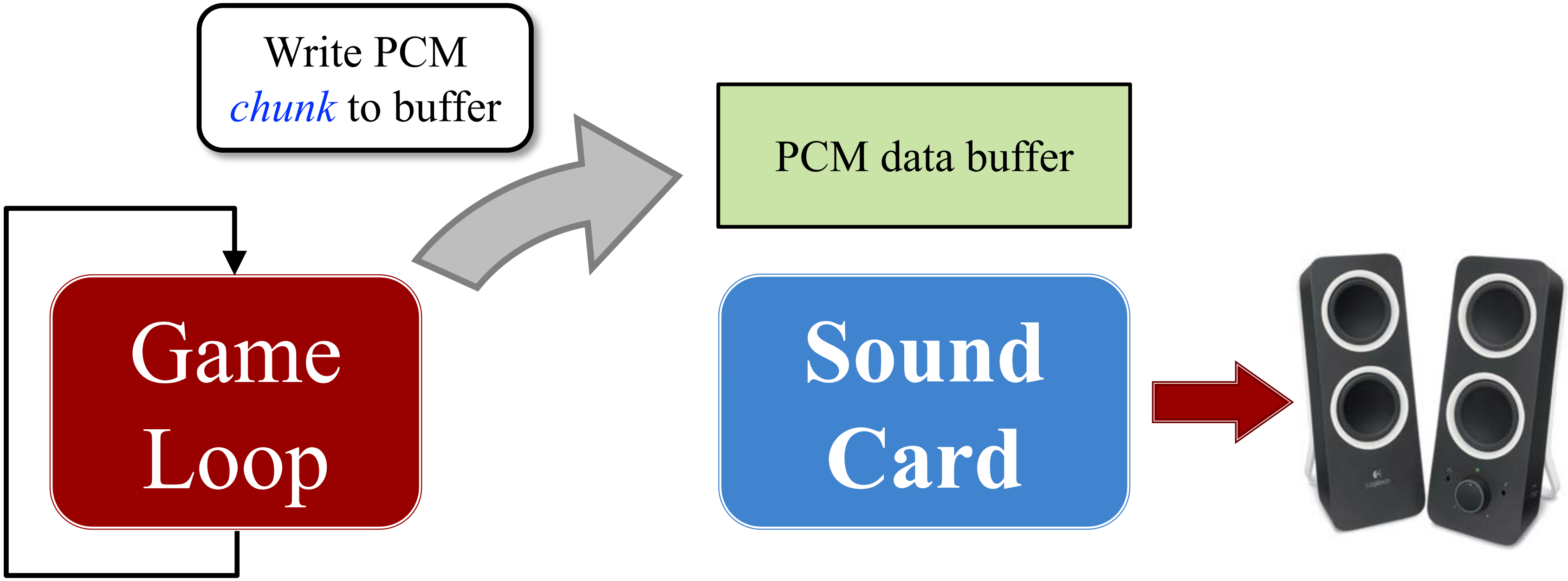


- **Example**: 0.5 seconds of stereo at 44.1 kHZ
  - 0.5 s * 44100 f/s = 22050 frames
  - 2 samples/frame * 22050 frames = 44100 samples
  - 4 bytes/sample * 44100 samples = 176.4 kBytes
- 1 minute of stereo CD sound is 21 MB!

Playing Sound Directly



Playing Sound Directly



Direct Sound in LibGDX: AudioDevice

- ```
  /**
   * Writes the array of float PCM samples to the audio device.
   *
   * This method blocks until they have been processed.
   */
  void writeSamples(float[] samples, int offset, int numSamples)
  ```

- ```
  /**
   * Writes array of 16-bit signed PCM samples to the audio device.
   *
   * This method blocks until they have been processed.
   */
  void writeSamples(short[] samples, int offset, int numSamples)
  ```

Direct Sound in LibGDX: AudioDevice

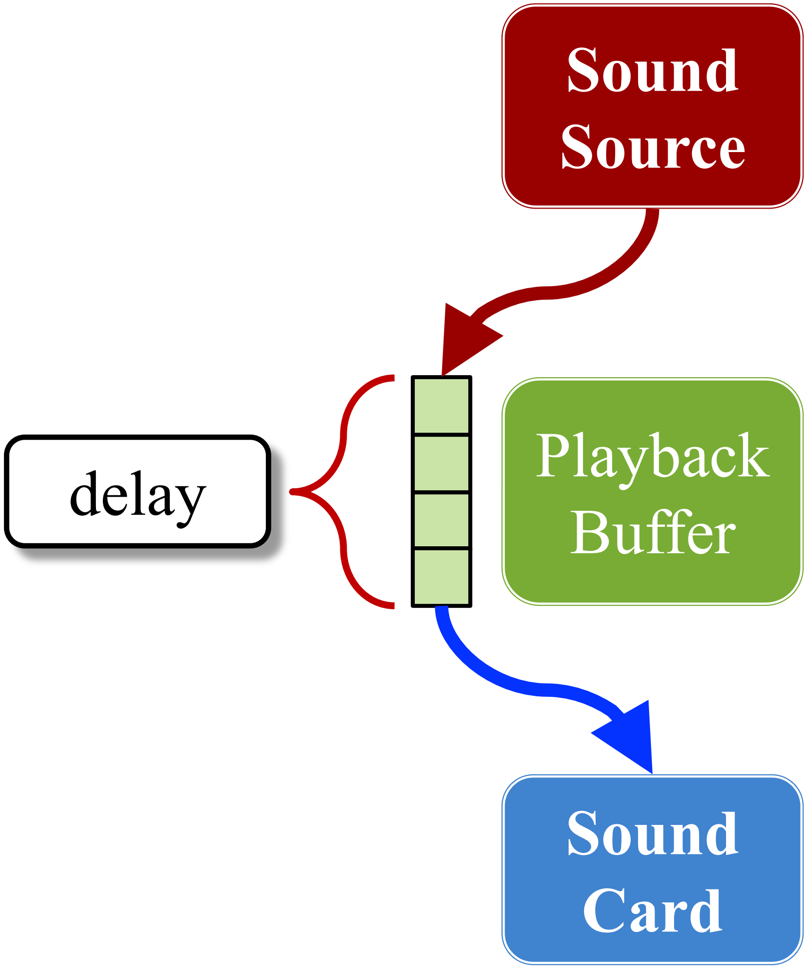
- ```
  /**
   * Writes the array of float PCM samples to the audio device.
   *
   * This method blocks until they have been processed.
   */
  void writeSamples(float[] sam ples)
  ```

Requires separate *audio thread*

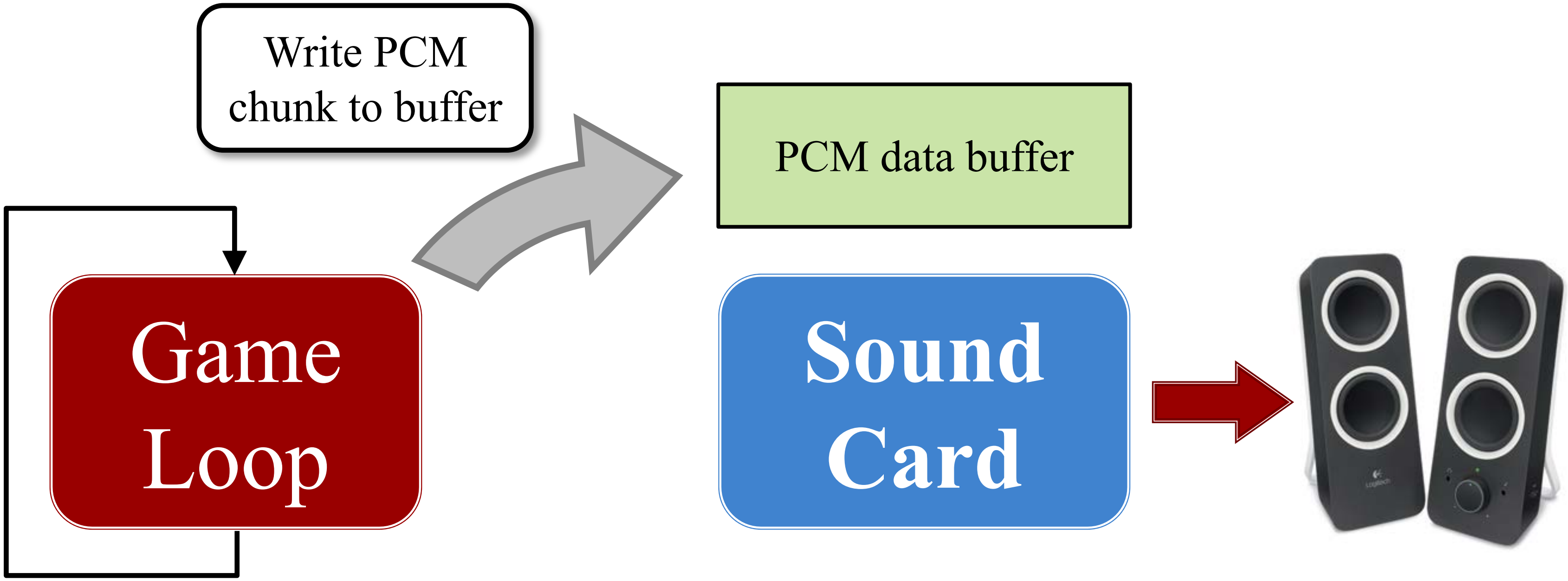
- ```
  /**
   * Writes array of 16-bit signed PCM samples to the audio device.
   *
   * This method blocks until they have been processed.
   */
  void writeSamples(short[] samples, int offset, int numSamples)
  ```

The Latency Problem

- Buffer is really a *queue*
  - Output from queue front
  - Playback writes to end
  - Creates a *playback delay*
- **Latency**: amount of delay
  - Some latency must exist
  - Okay if latency ≤ framerate
  - **Android latency is ~90 ms!**
- Buffering is a necessary evil
  - Keeps playback smooth
  - Allows real-time *effects*



Playing Sound Directly

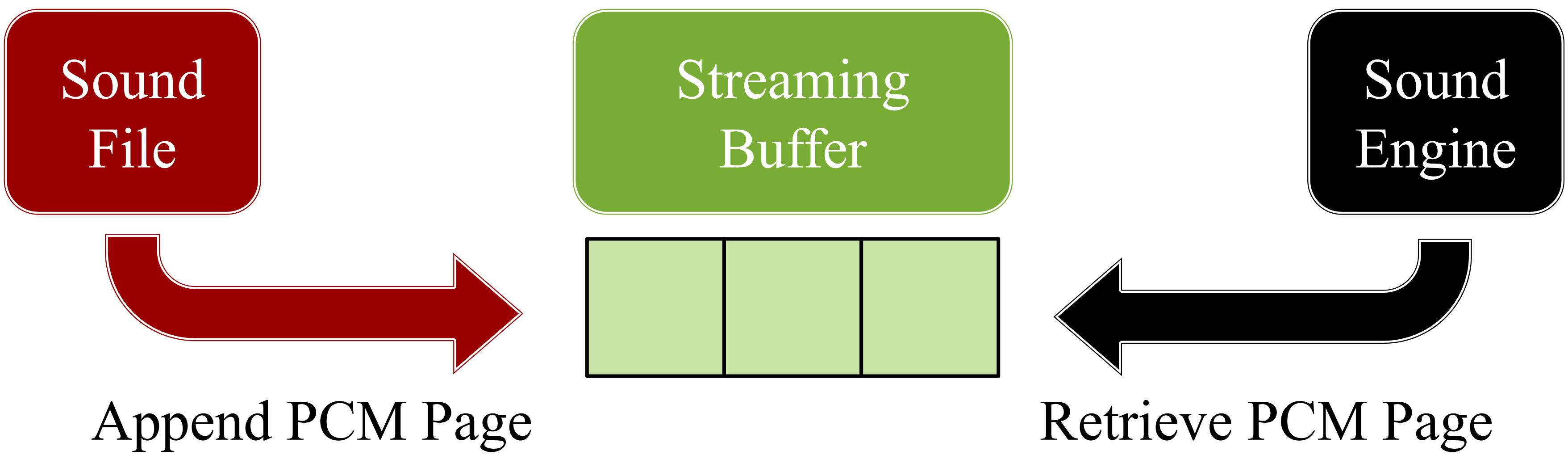


Choice of buffer size is important!

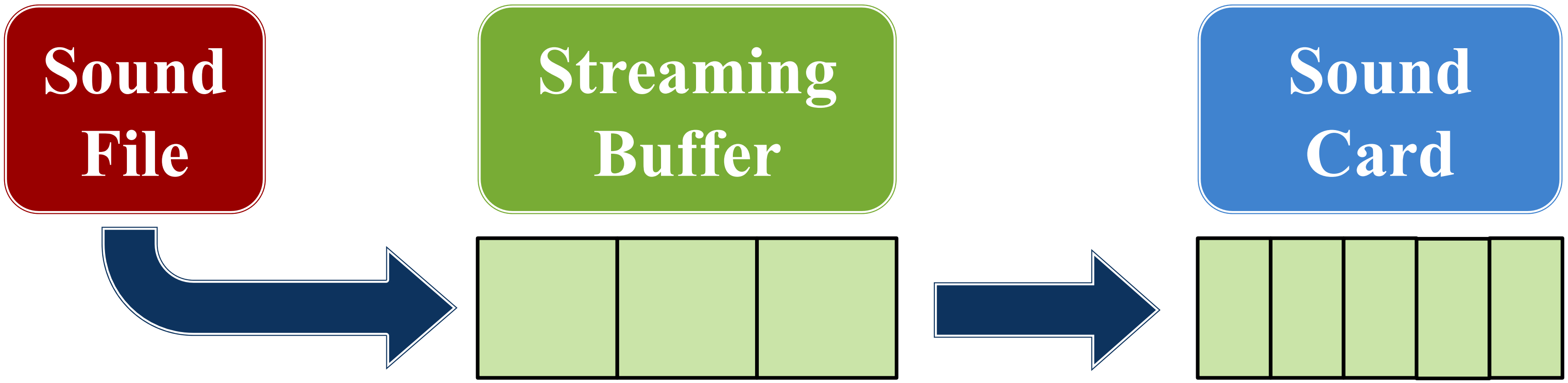
- **Too large**: *long* latency until next sound plays
- **Too small**: buffers swap too fast, causing audible pops

How Streaming Works

- All sound cards **only** play PCM data
  - Other files (MP3 etc.) are decoded into PCM data
  - But the data is *paged-in* like memory in an OS
- This is how OGG is added to most engines



How Streaming Works

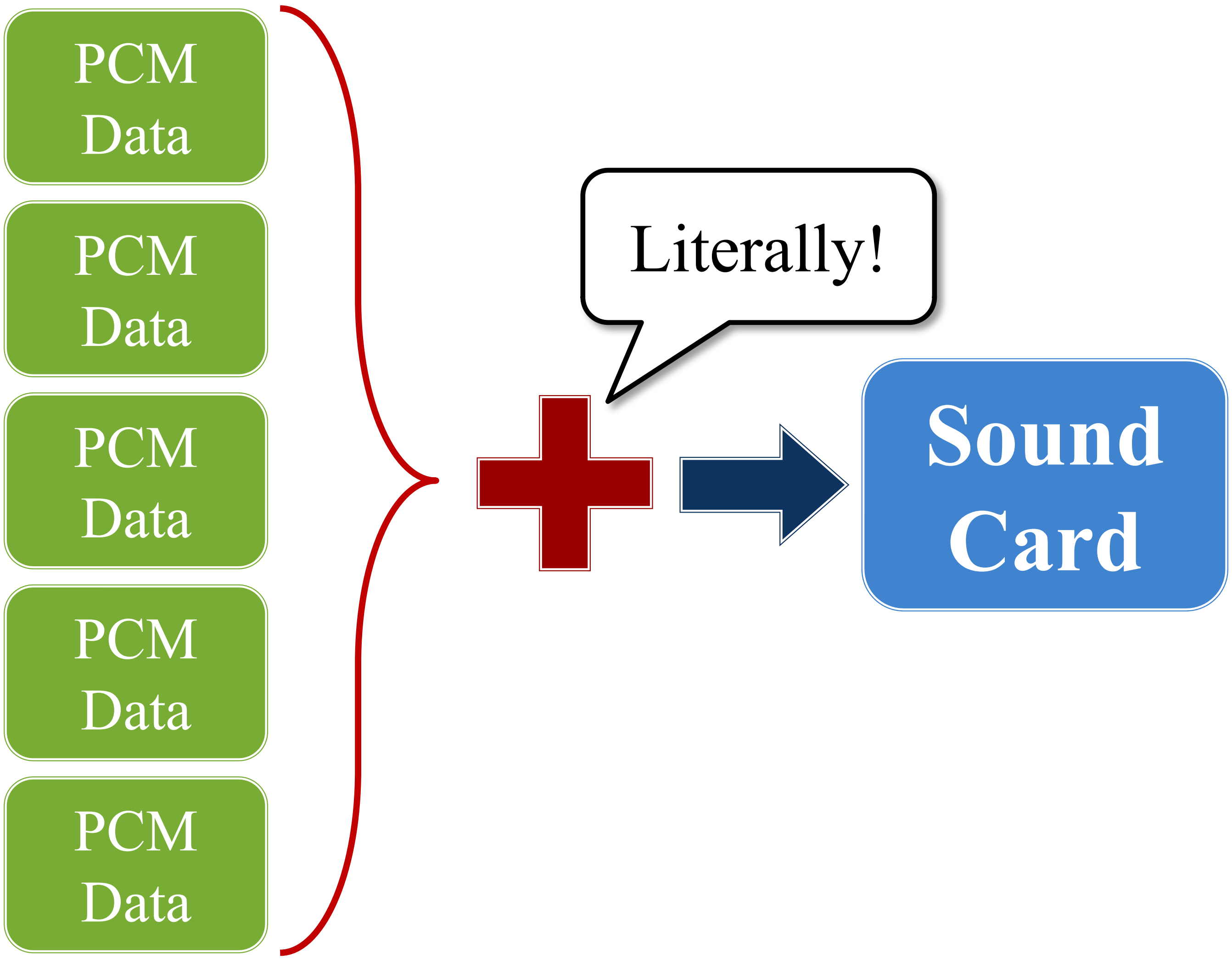


Page size set by file format

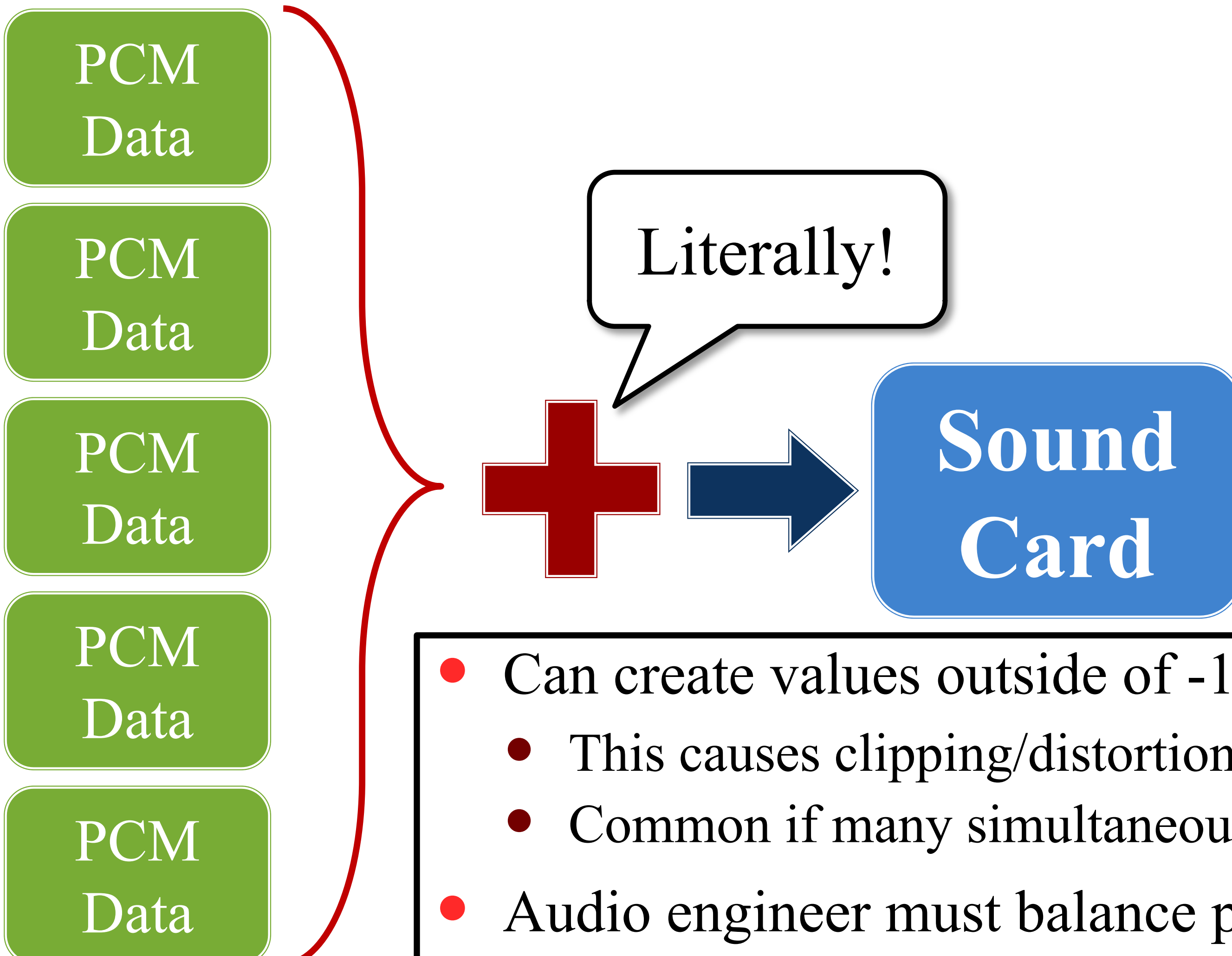
Chunk size set by audio API

- **Sound**: Sound asset that is *preloaded* as full PCM
- **Music**: Sound asset that is *streamed* as PCM pages

Handling Multiple Sounds



Handling Multiple Sounds



- Can create values outside of -1 to 1
  - This causes clipping/distortion
  - Common if many simultaneous sounds
- Audio engineer must balance properly

We Want Something Simpler!

- Want ability to **play** and **track** sounds
  - Functions to load sound into card buffer
  - Functions to detect if sound has finished
- Want ability to **modify** active sounds
  - Functions for volume and pitch adjustment
  - Functions for stereo panning (e.g. left/right channels)
  - Functions to pause, resume, or loop sound
- Want ability to **mix** sounds together
  - Functions to add together sound data quickly
  - Background process for dynamic volume adjustment

We Want Something Simpler!

- Want ability to **play** and **track** sounds
  - Functions to load sound into card buffer
  - Functions to detect if sound has finished
- Want ability to **modify** active sou…
  - Fu…
  - …ers)
  - …pause, resume, or loop sound

This is the purpose of a **sound engine**

- Want ability to **mix** sounds together
  - Functions to add together sound data quickly
  - Background process for dynamic volume adjustment

Standard Industry Sound Engines

- OpenAL
  - Created in 2000 by Loki Software for Linux
  - Was an attempt to make a sound standard
  - Loki went under; last stable release in 2005
  - Still used heavily in the Indie space
- FMOD/WWISE
  - Industry standard for game development
  - Mobile support is possible but not easy
  - Not free; but no cost for low-volume sales

The LibGDX Sound Classes

Sound

- Primary method is play()
  - Returns a long integer
  - Represents sound *instance*
  - `loop()` is a separate method
- Has no public constructor
  - Use `Audio.newSound(f)`
  - `Audio` can cache/preload
- Must dispose when done

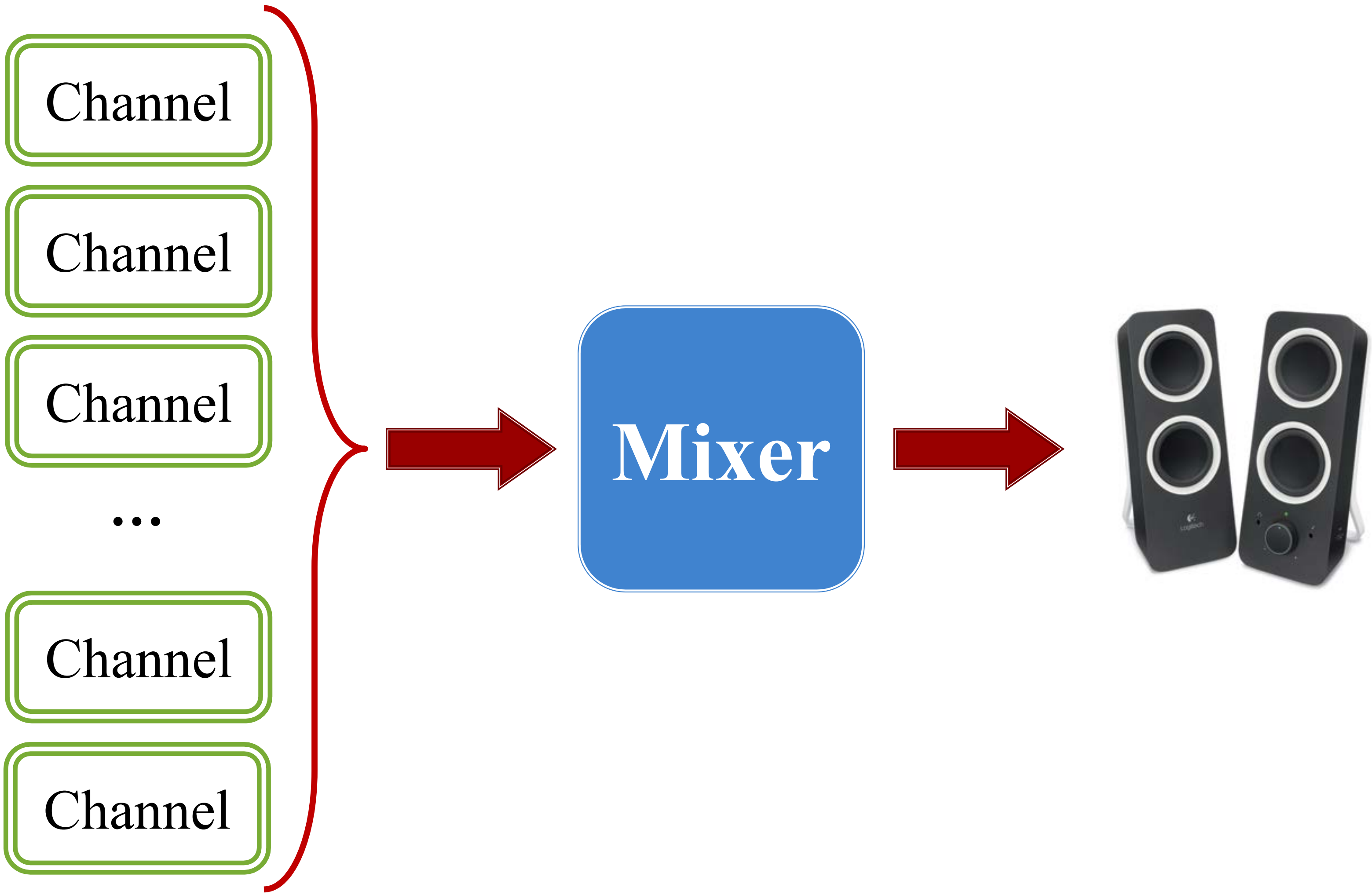
Music

- Primary method is play()
  - This is a `void` method
  - Only allows **one instance**
  - `loop` is an attribute of music
- Has no public constructor
  - Use `Audio.newMusic(f)`
  - `Audio` can cache the file
- Must dispose when done

Playing a Sound

- Playback may include **multiple sounds**
  - Sounds may play simultaneously (offset)
  - Simultaneous sounds may be same asset
  - Asset (source) vs. Instance (playback)
- Playback crosses **frame boundaries**
  - It may span multiple animation frames
  - Need to know when it stops playing
  - May need to stop (or pause) it early

Classic Model: **Channels**



Classic Model: **Channels**

Channel

Channel

Channel

…

Channel

Channel

Engine has fixed number of channels (historically 24)

Mixer

Classic Model: **Channels**

Channel

Channel

Channel

…

Channel

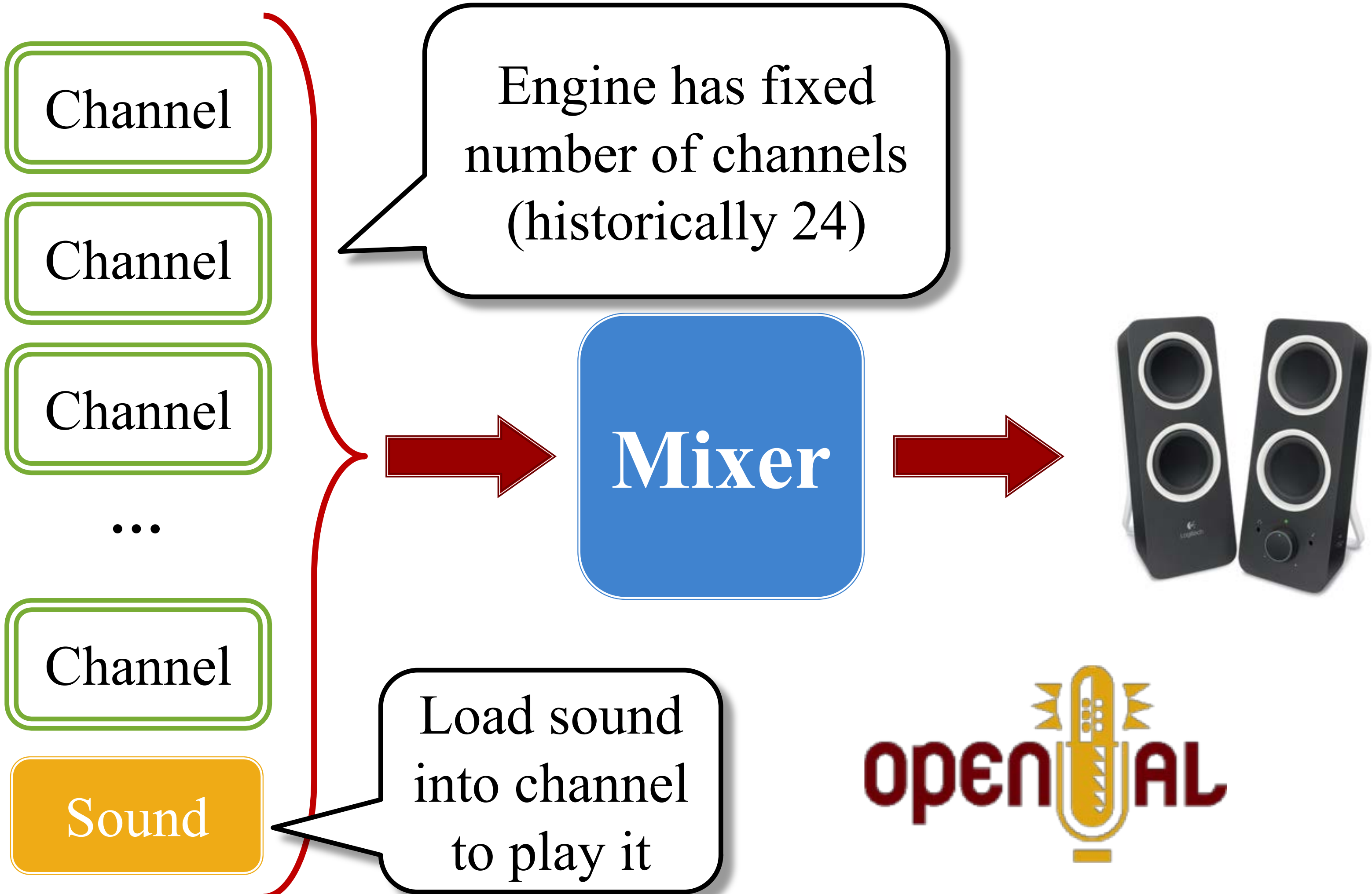
Sound

Engine has fixed number of channels (historically 24)

Mixer

Load sound into channel to play it

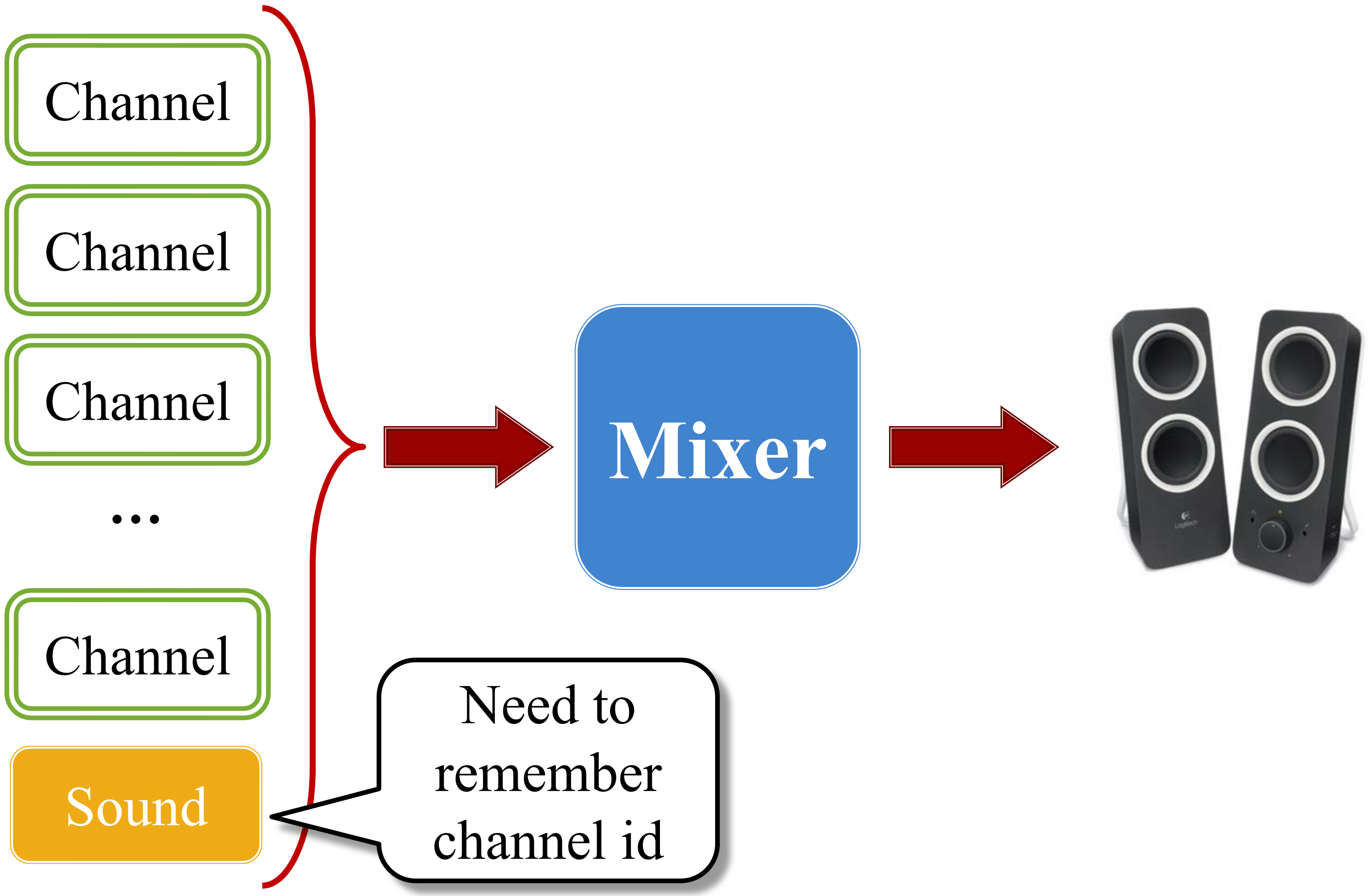
Classic Model: Channels



Playing a Sound with Channels

- **Request** a sound channel for your asset
  - If none is available, sound fails to play
  - Otherwise, it gives you a id for a channel
- **Load** asset into the channel (but might stream)
- **Play** the sound channel
  - Playing is a property of the channel, not asset
  - Channel has other properties, like volume
- **Release** the channel when the sound is done
  - This is usually done automatically

Application Design



The Sound API

- ```
  /**
   * @return channel id for sound playback
   *
   * If no channel is available, returns -1
   * @param volume  The sound volume
   * @param pitch   The pitch multiplier (>1 faster, <1 slower)
   * @param pan     The speaker pan (-1 full left, 1 full right)
   */
  public long play(float volume, float pitch, float pan);
  ```
- `public void stop(long audioID);`
- `public void resume(long audioID);`
- `public void setLooping(long audioID, boolean loop);`
- `Public void setVolume(long audioID, float volume);`
  ```

The Sound API

- ```
  /**
    * @return channel id for sound playback
    *
    * If no chan                          -1
    * @param v                          e
    * @param p                          ier (>1 faster, <1 slower)
    * @param                            (-1 full left, 1 full right)
    */
  public long play(float volume, float pitch, float pan);
  ```

Returns available channel id

- `public void stop(long audioID);`
- `public void resume(long audioID);`
- `public void setLooping(long audioID, boolean loop);`
- `Public void setVolume(long audioID, float volume);`

Need to remember channel id
  ```

Why This is Undesirable

- Tightly couples architecture to sound engine
  - All controllers need to know this channel id
  - Playback must communicate the id to all controllers
- Instances usually have a *semantic meaning*
  - **Example**: Torpedo #3, Ship/crate collision
  - Meaning is independent of the channel assigned
  - Would prefer to represent them by this meaning
- **Solution**: Refer to instances by *keys*

The SoundController Class (Lab 4)

- ```
  /**
   * @return true if the given sound could be played
   *
   * @param  key     the reference key for the sound effect
   * @param  file    the sound effect file to play
   * @param  loop    whether to loop indefinitely
   * @param  volume the sound volume
   */
  public boolean play(string key, string file, bool loop, float volume);
  ```
- `public void stop(string key);`
- `public void isActive(string key);`
- Other methods I forgot to write

Refer to instance logically
  ```

Stopping Sounds

- Would like to know when a sound is finished
  - To free up the channel (if not automatic)
  - To stop any associated animation
  - To start a follow-up sound
- Two main approaches
  - **Polling**: Call an isPlaying() method
  - **Callback**: Pass a function when play

Cannot do in android.media

Stopping Sounds

- Would like to know when a sound is finished
  - To free up the channel (if not automatic)
  - To stop any associated animation
  - To start a follow-up sound
- Two main approaches
  - **Polling**: Call an isPlaying() method
  - **Callback**: Pass a function when play
- **LibGDX cannot tell you anything!!!**

Cannot do in android.media

SoundController: The Ugly Hacks

- ```
  /**
   * Sets the maximum # of frames a sound can run
   */
  public void setTimeLimit(long timelimit);
  ```
- ```
  /**
   * Sets the number of frames before a key can be reused
   */
  public void setCoolDown(long cooldown);
  ```
- ```
  /**
   * Sets the maximum # of sounds per animation frame
   */
  public void setFrameLimit(int framelimit);
  ```
  ```

SoundController: The Ugly Hacks

```
/**
  * Sets the maximum # of frames a sound
  */
public void setTimeLimit(long timelimit);
```

Garbage collect done sounds

```
/**
  * Sets the number of frames before a key can be reused
  */
public void setCoolDown(long cooldown);
```

```
/**
  * Sets the maximum # of sounds per animation frame
  */
public void setFrameLimit(int framelimit);
```

SoundController: The Ugly Hacks

- ```
  /**
   * Sets the maximum # of frames a sound
   */
  public void setTimeLimit(long timelimit);
  ```

  Garbage collect done sounds

- ```
  /**
   * Sets the number of frames before a key
   */
  public void setCoolDown(long cooldown);
  ```

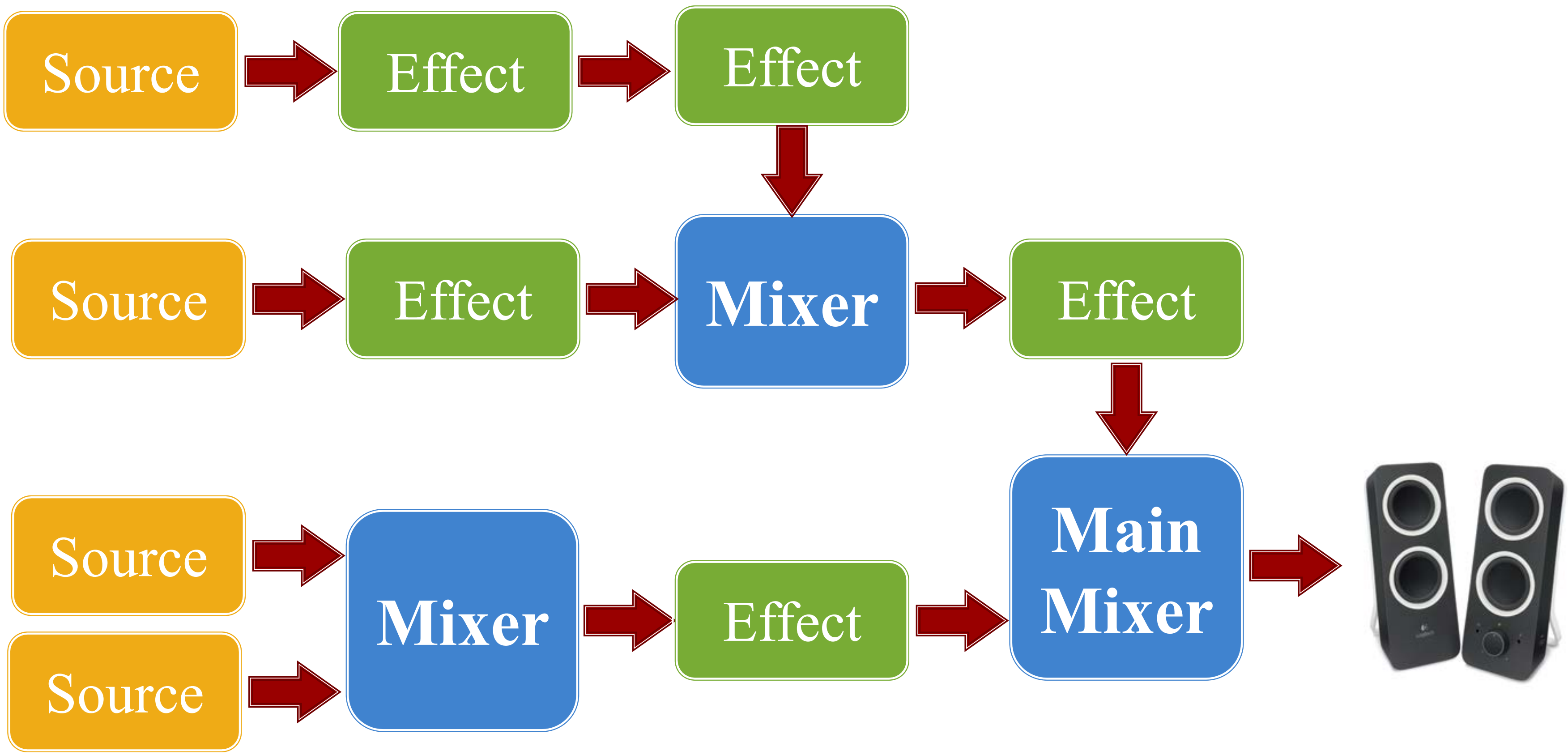
  Prevent stopping recent sounds

- ```
  /**
   * Sets the maximum # of sounds per animation frame
   */
  public void setFrameLimit(int framelimit);
  ```
  ```

SoundController: The Ugly Hacks

- ```
  /**
   * Sets the maximum # of frames a sound
   */
  public void setTimeLimit(long timelimit);
  ```
- ```
  /**
   * Sets the number of frames before a key
   */
  public void setCoolDown(long cooldown);
  ```
- ```
  /**
   * Sets the maximum # of sounds per ani
   */
  public void setFrameLimit(int framelimit);
  ```

Garbage collect done sounds

Prevent stopping recent sounds

Limit overhead of changing mixer graph
  ```

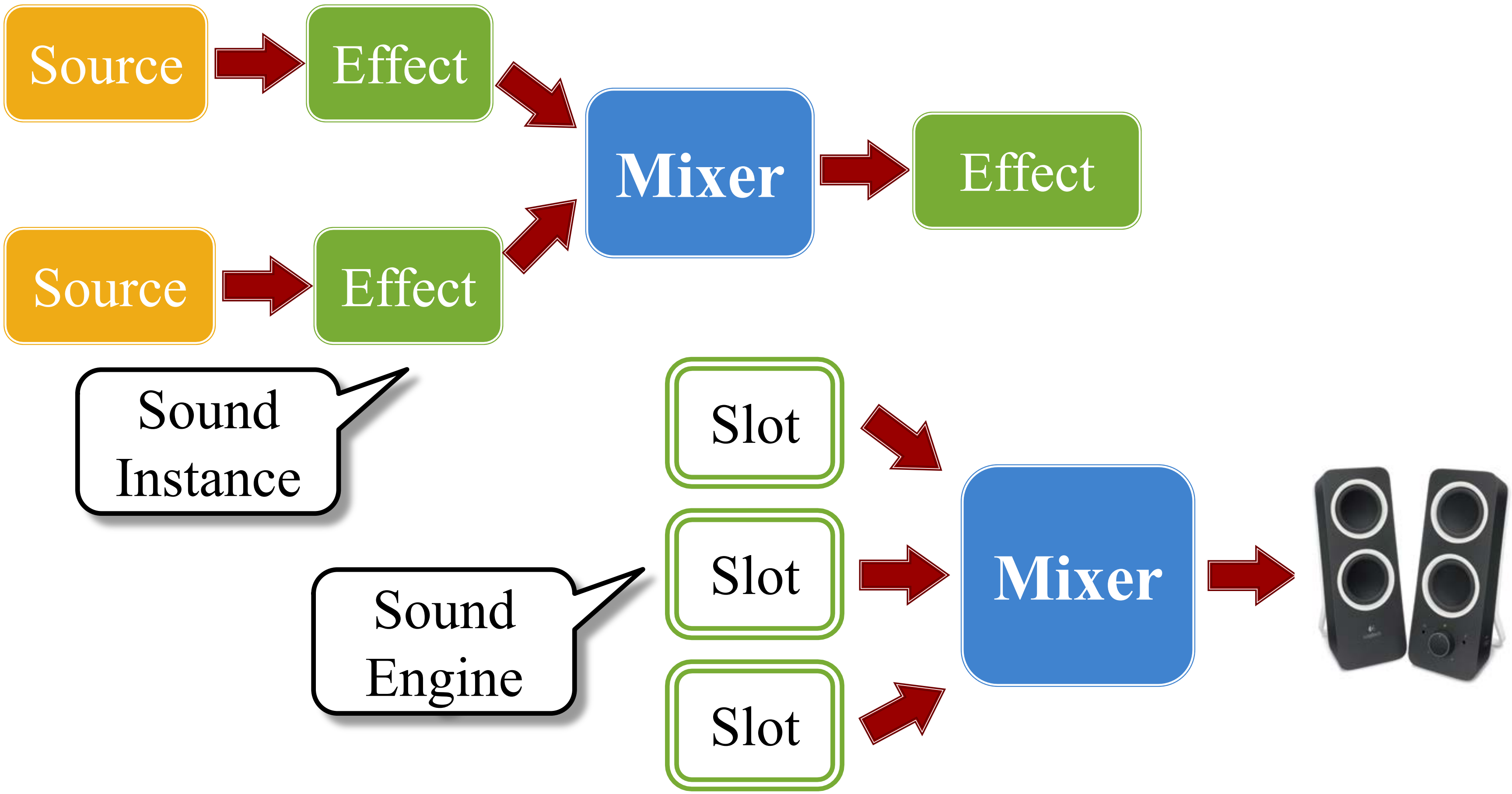
Problem with the Channel Model

- All controls are embedded in the channel
  - **Example**: Volume, looping, play position
  - Restricted to a *predetermined* set of controls
- Modern games want *custom sound-processing*
  - User defined sound filters (low pass, reverb)
  - Advanced equalizer support
  - Support for surround and 3D sound
  - Procedural sound generation

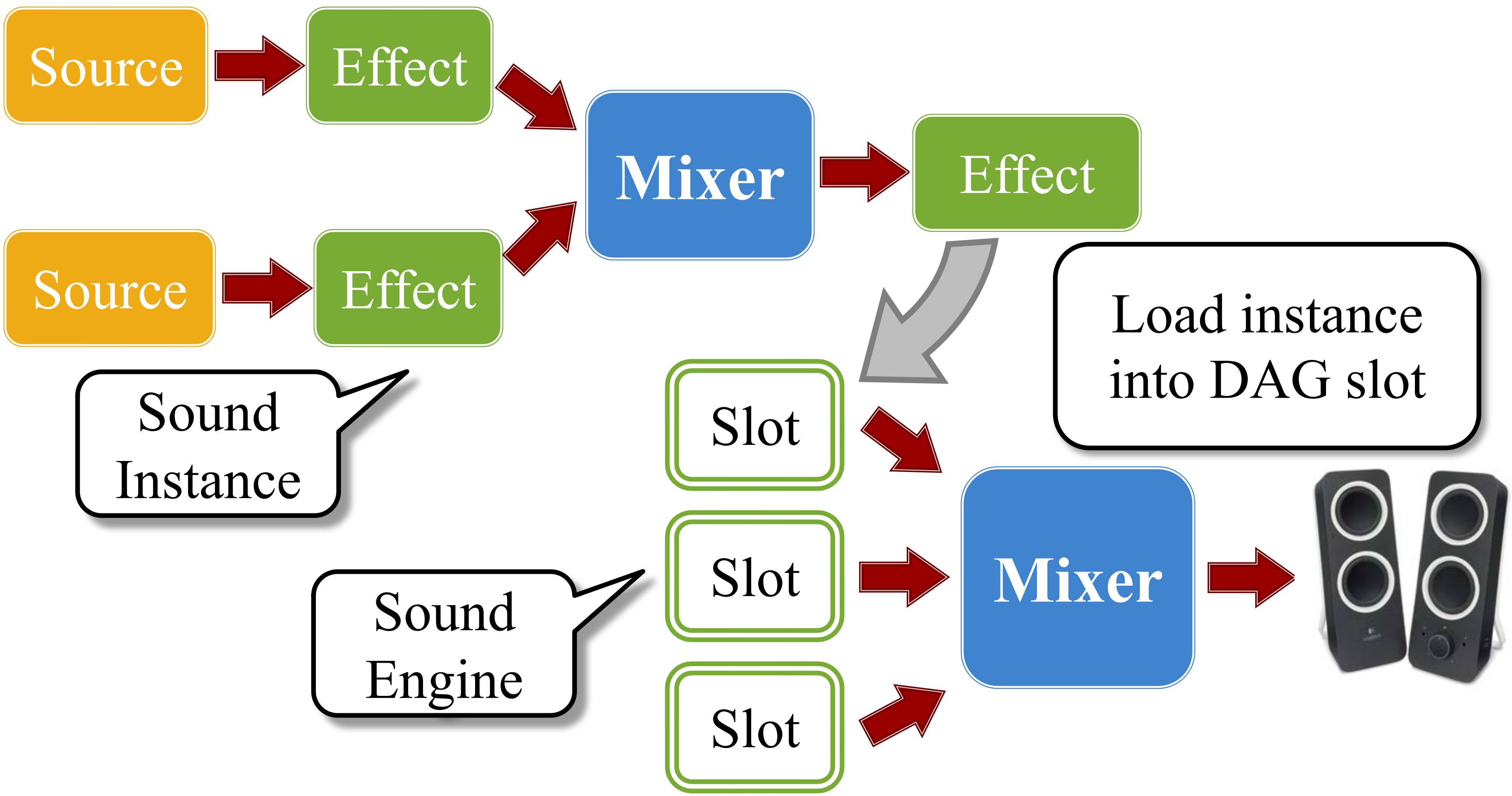
DSP Processing: The Mixer DAG



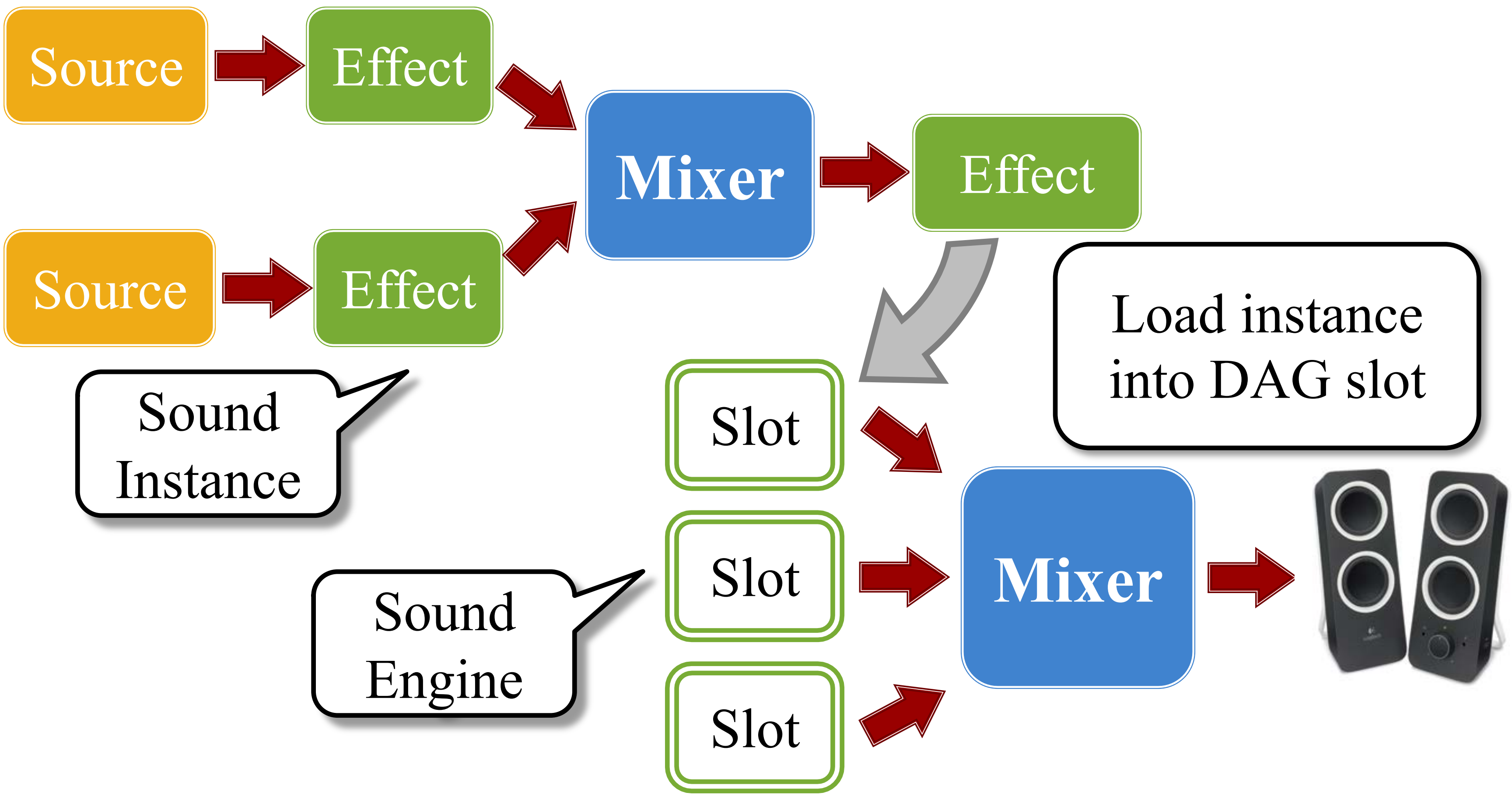
Idea: Sound Instance is a Sub-DAG



Idea: Sound Instance is a Sub-DAG

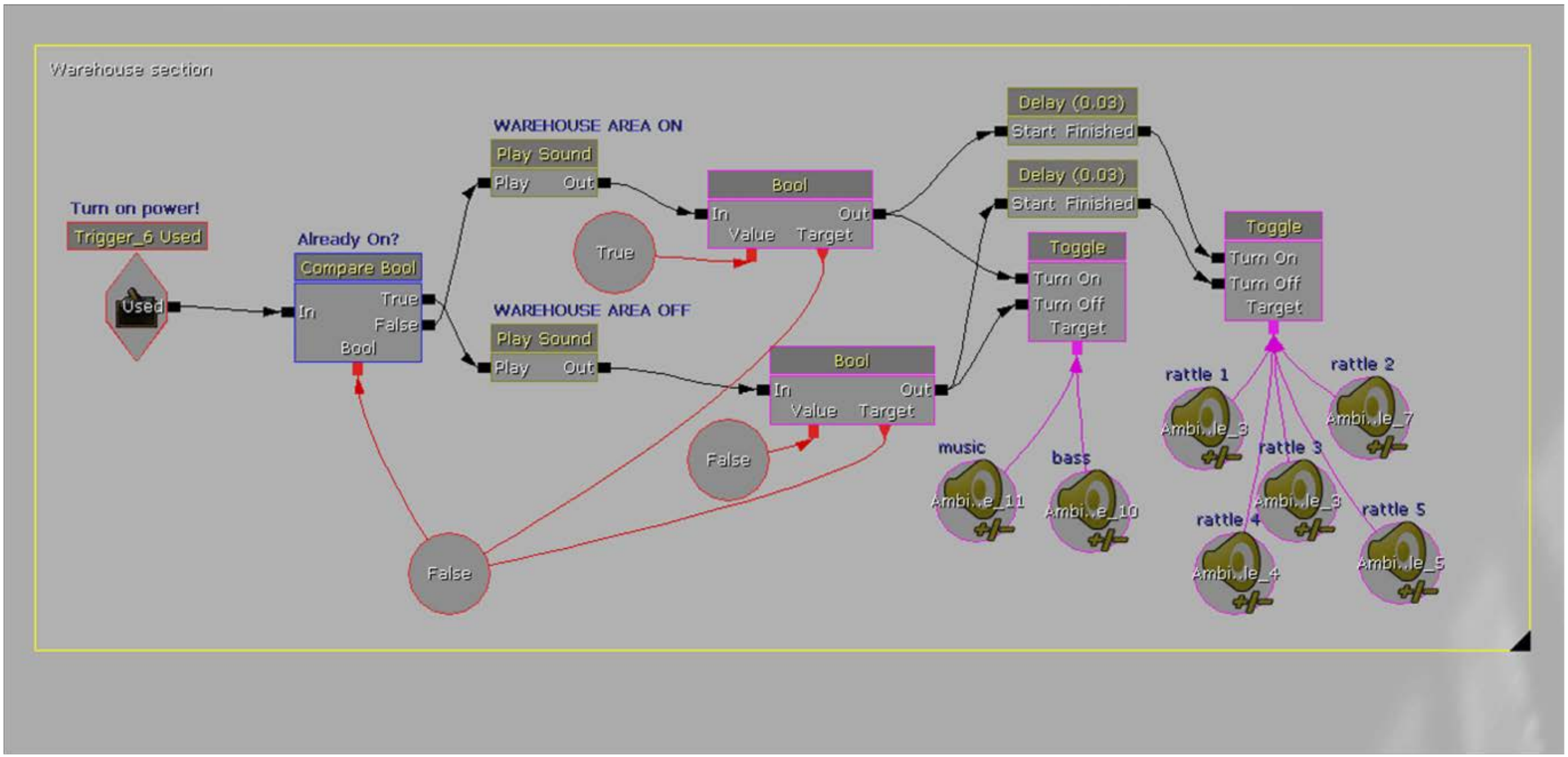


Idea: Sound Instance is a Sub-DAG

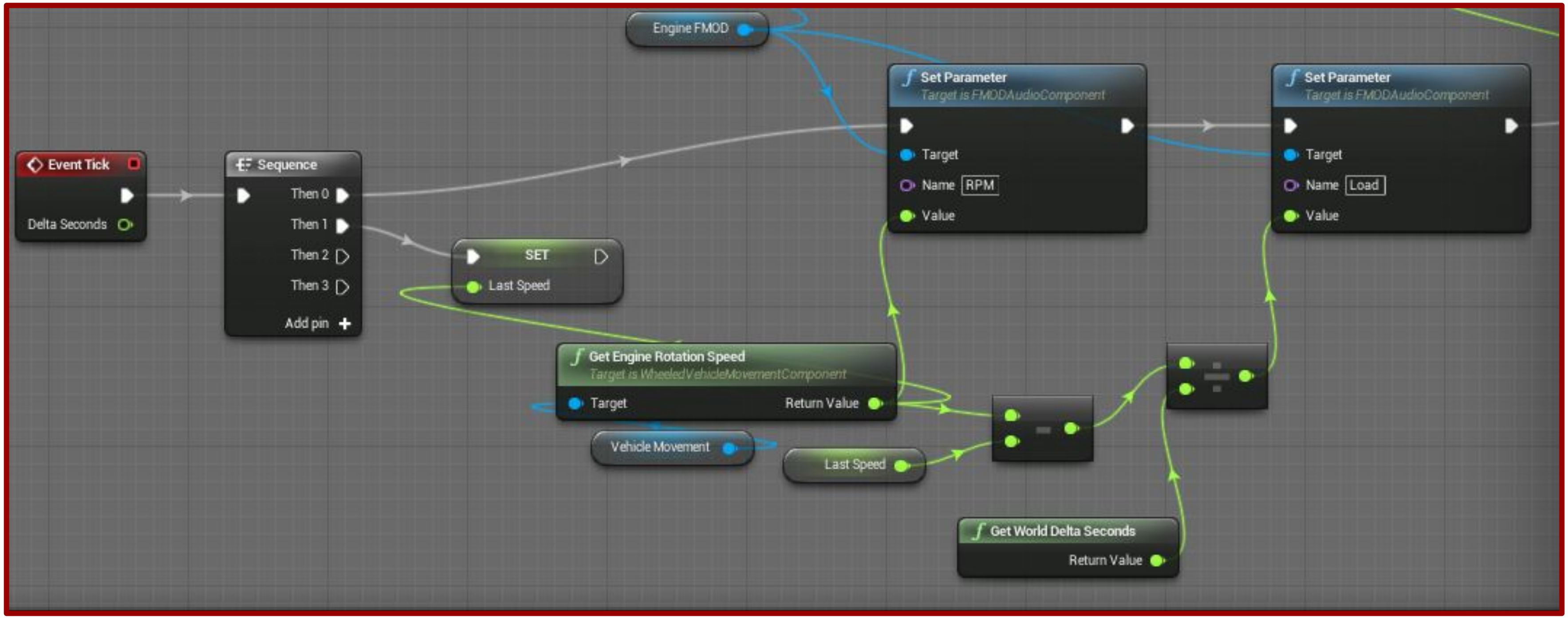


Channel model is a special case of this DAG

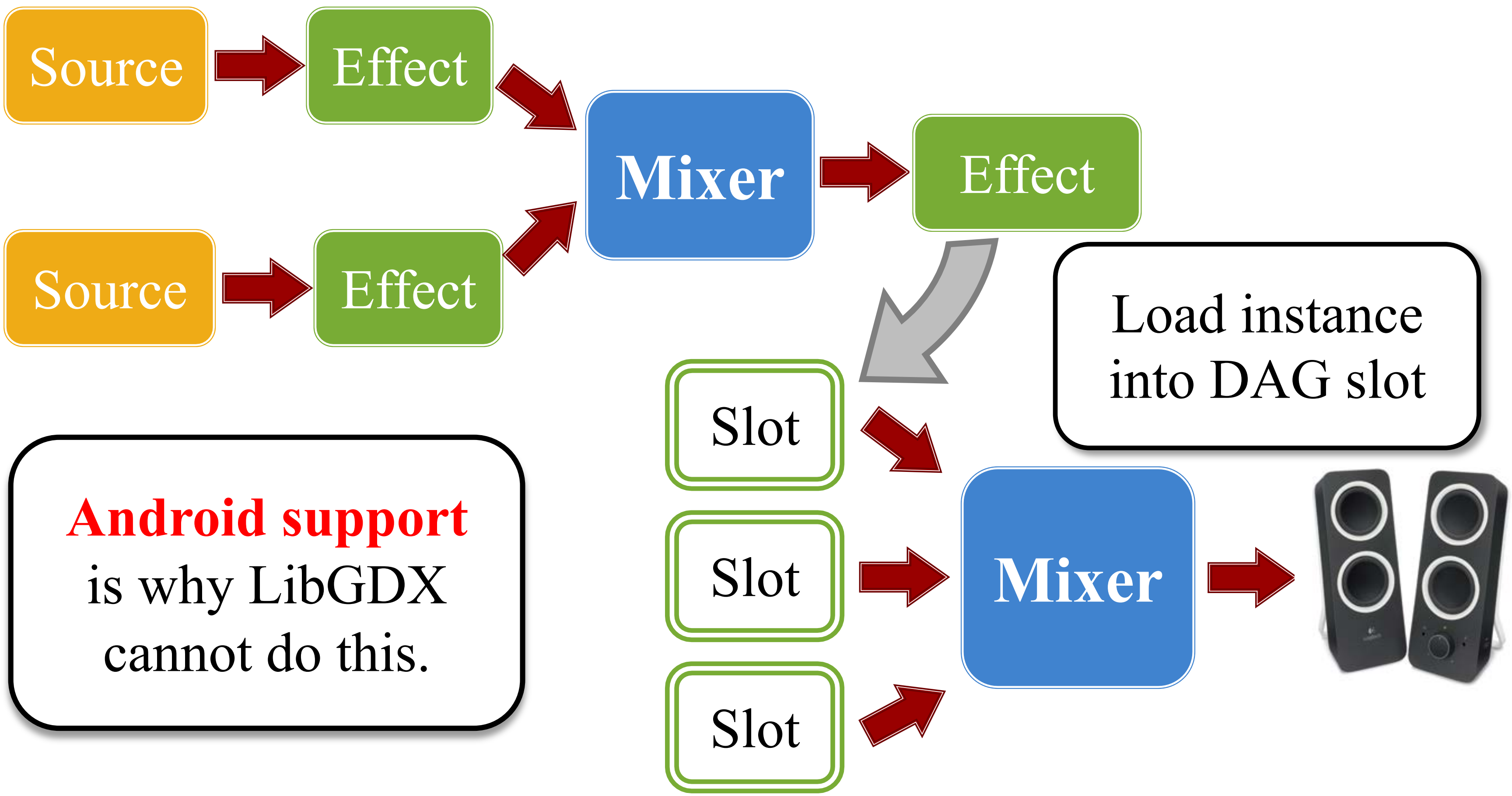
Example: UDK Kismet



Example: FMOD



Idea: Sound Instance is a Sub-DAG



Summary

- Audio design is about creating soundscapes
  - Music, sound effects, and dialogue
  - Combining sounds requires a sound engine
- Cross-platform support is a problem
  - Licensing issues prevent a cross-platform format
  - Very little standardization in sound APIs
- Best engines use digital signal processing (DSP)
  - Mixer graph is a DAG supporting sound effects
  - Android prevents us from doing this in LibGDX