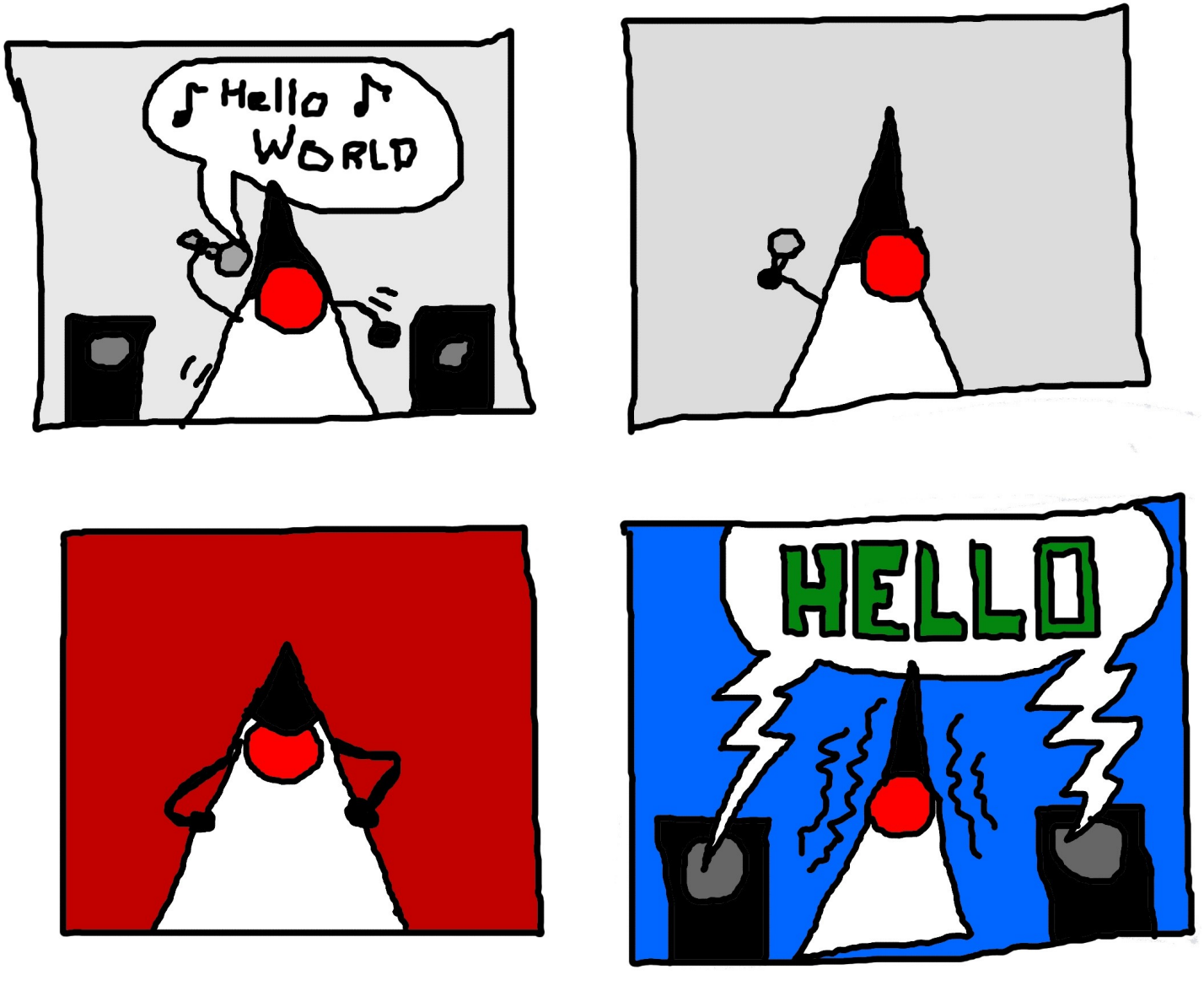
# Low latency in Java Sound & Gervill

Karl Helgason
Fosdem 2011

# Extreme example of latency



# Measuring latency

- **1 msec**

  = thousandth (1/1,000) of a second.

  = cycle time for frequency 1 kHz

  = time taken for sound wave to travel ~34 cm (14.5 inches)

  = 48 samples at samplerate 48 kHz

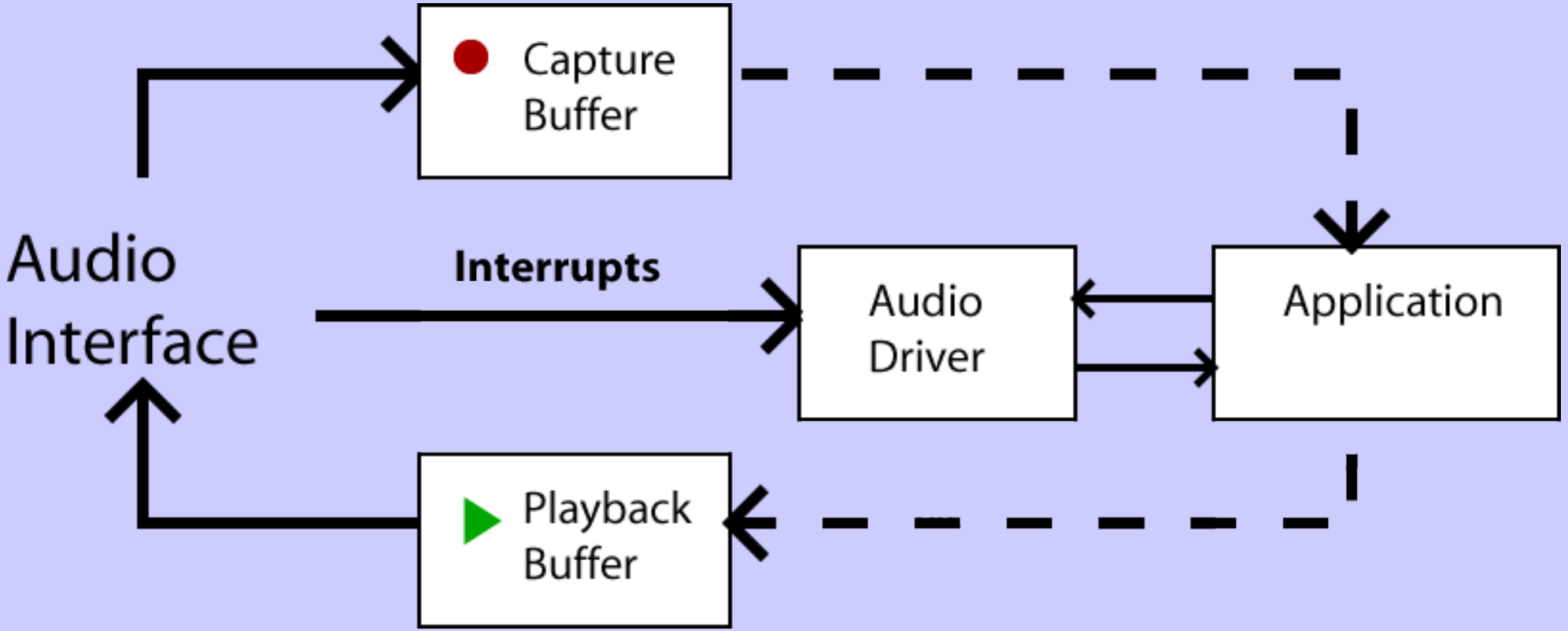
- **1 sample at samplerate 48 kHz**

  = ~7 mm (0.276 inches)

  = 0.02 msec

  = 20µs (microseconds)

# Audio I/O


Capture Buffer
Audio Interface
Interrupts
Audio Driver
Application
Playback Buffer

# Audio Latency

- Latency is the delay from when signal enters system and exit it.
- About 10 msec is considered acceptable
- However less is better

# Low latency

- Most operatings system today are capable of low-latency audio.
- But not that is not always the case when using Java Sound.

# “**The enemies**” of low-latency

- **Buffering (latency v.s. glitches)**
- OS Overhead
- Thread priorites
- Audio Drivers
- **Software Mixers**
- **Blocking I/O**
- Garbage collector

# Java Sound Latency

- On Linux using ALSA ~ 5 msec
  - Cons: Sharing not possible
- On Linux using PulseAudio ~ 25 msec
- On Windows 7 ~ 100 msec
- On Mac ~ even worse
  - Only “Java Sound Engine” available.

# Alternatives...

- On Linux using Jack whould be more ideal
  - Pros: Sharing audio between applications.
- On Windows using ASIO or WaveRT
- And on Mac using CoreAudio.

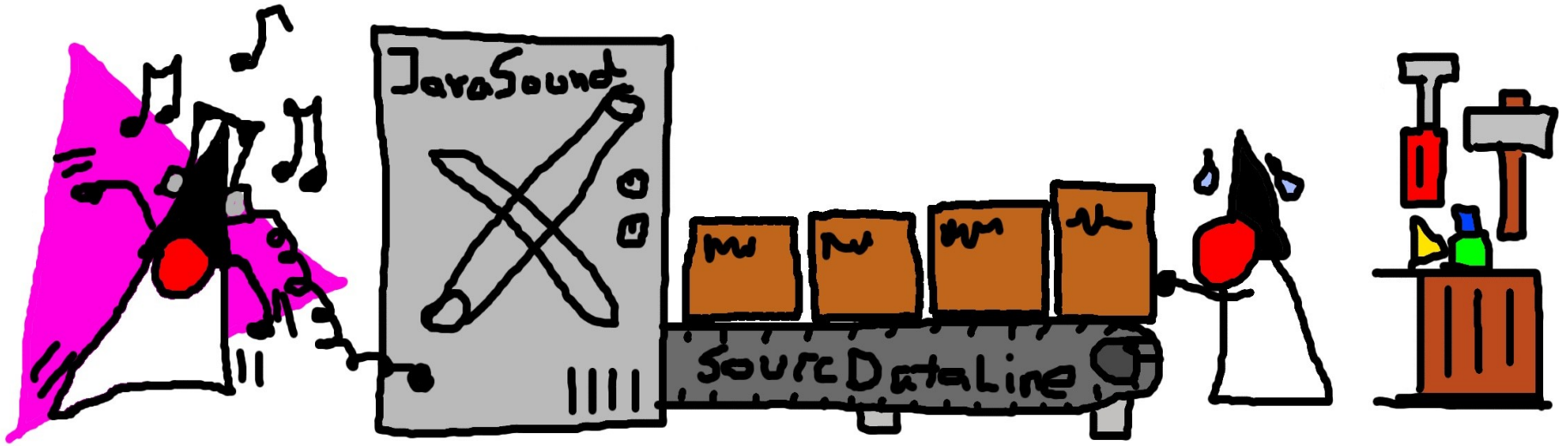
- However these Audio API's can't be optimally translated to Java Sound interfaces.

# Push v.s. Pull

- ALSA on Linux, DirectSound on Windows are both based on Blocking I/O
- However Jack on Linux, ASIO on Windows, CoreAudio on Mac are different.
- And Java Sound is based on Blocking I/O

# Blocking I/O

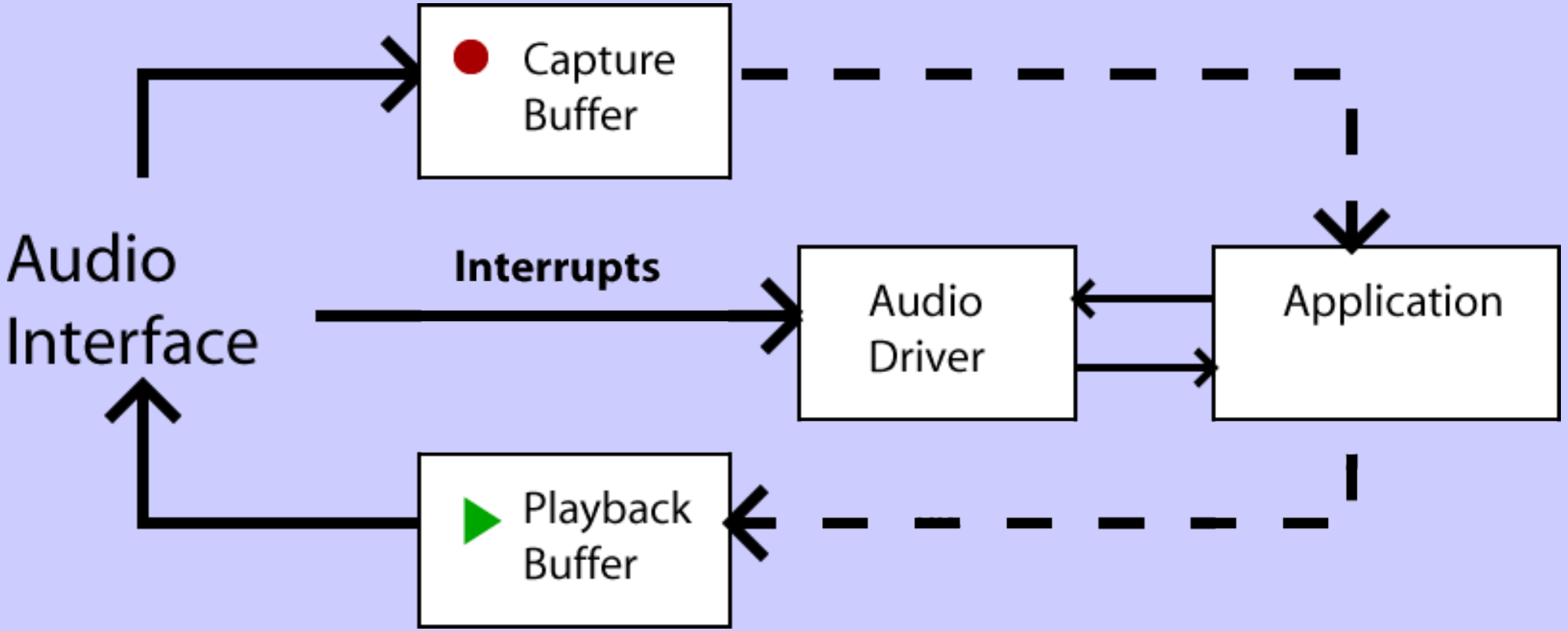
- The Audio Thread fills the SourceDataLine until it blocks.
- This buffering reduces glitches in playback put adds latency instead.



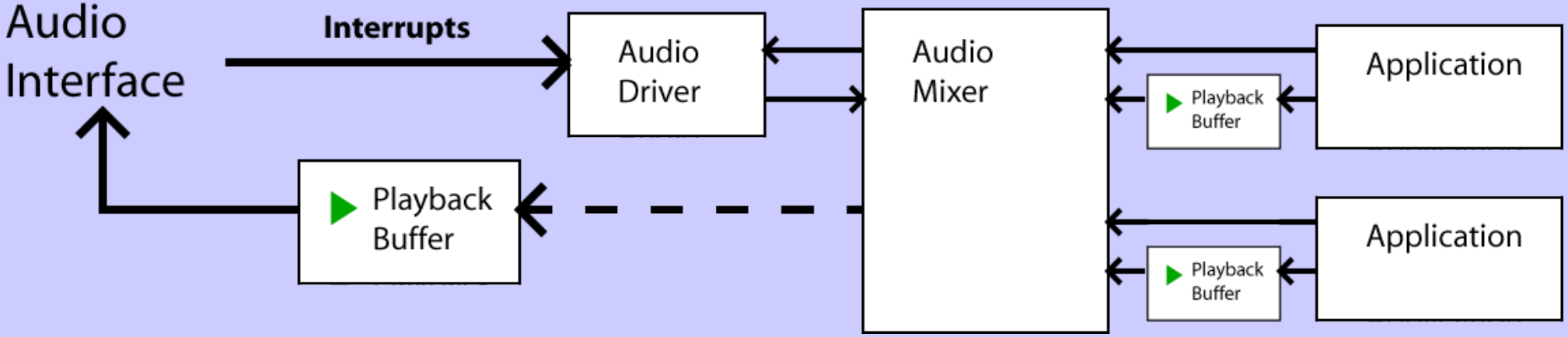
# “Pull” audio API

- Audio buffers are computed on-demand
- This ensures the user only hears freshly made buffers :)

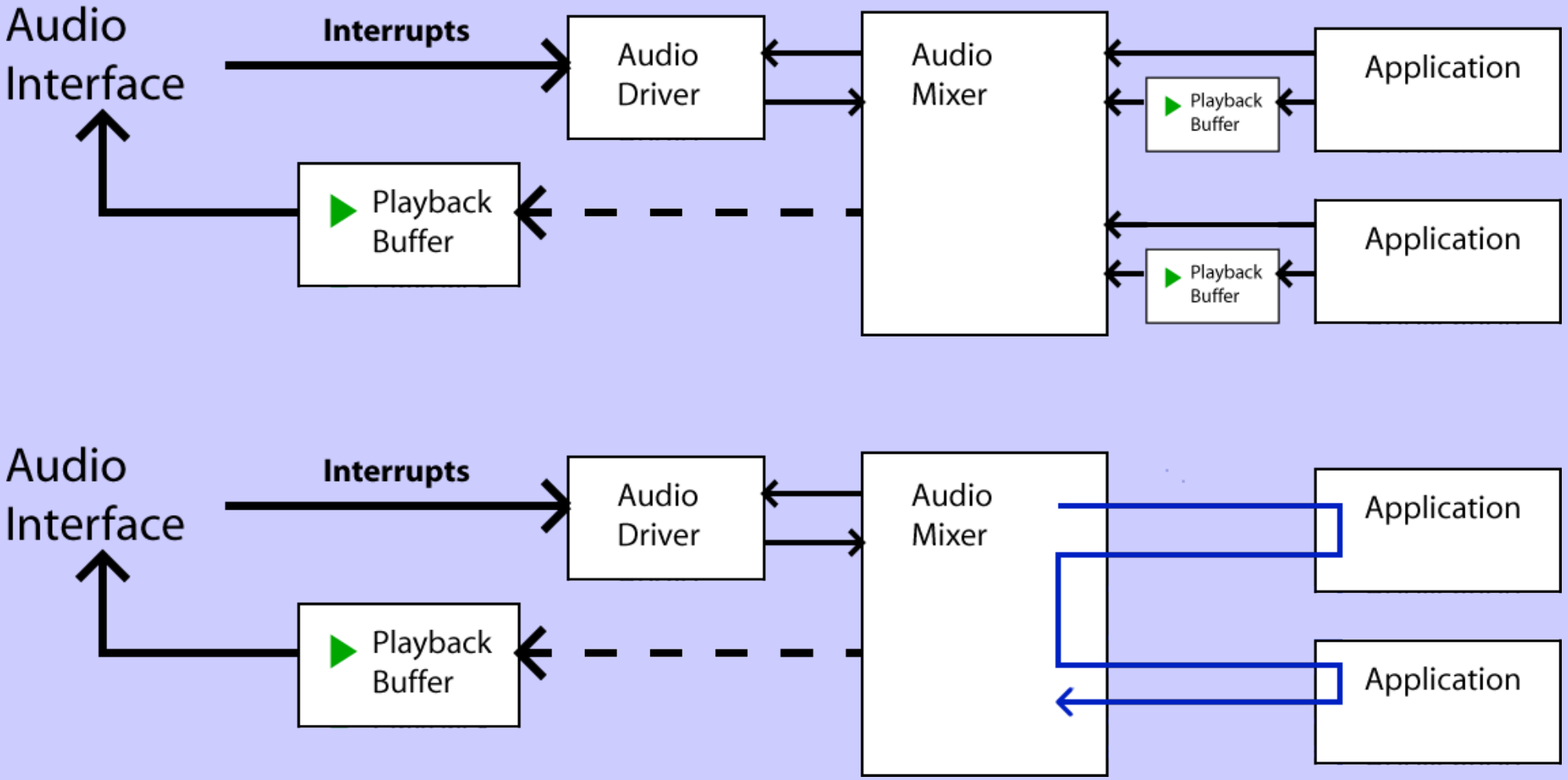
# Audio I/O


Capture Buffer
Audio Interface
Interrupts
Audio Driver
Application
Playback Buffer

# Software Mixers



# Software Mixers



# Current workarounds

- On Linux: JJack
  - http://jjack.berlios.de/
- On Windows: JasioHost
  - https://github.com/mhroth/jasiohost
- On Mac: Mandoline M3D Mixer
  - http://www.mandolane.co.uk/
- Both JJack & JAsioHost have custom interfaces to achieve optimial latency.

# The problem

- Application have to be especially written for each Audio API to achieve Low-latency performance.
- It goes against write once and run anywhere.

# The suggestion

- Improve Java Sound by adding necessary interfaces to allow pull-based API.
- This would enable users to use Jack, ASIO and Core Audio using standard interfaces.
- Pull-based API are also approiated for use with ALSA, DirectSound and WaveRT.

# Requirements

- Pull-based API
- Access to native buffers using NIO
- Non-interlaced buffers
  - E.g. each channel in seperate buffer.
- Bidirectional processing
  - Allow processing on input and output in same call.
- RealTime priority on threads when possible
  - On Windows 7: Multimedia Class Scheduler Service

# Example #1

**AudioInputStream** audiostream = synthesizer.open(format, null);

**DataLine.Info** info = **new** DataLine.Info(AudioPlayer.class, format);

**AudioPlayer** player = (AudioPlayer)AudioSystem.getLine(info);

player.open(audiostream);

player.start;

# Example #2

```
AudioProcessor processor = new AudioProcessor() {
  float[] buffer;
  public void processAudio(AudioEvent event) {
     int len = event.getBufferSize();
     if(buffer == null | buffer.length != len) buffer = new float[len];
     event.getInput(0).getFloatBuffer().read(buffer);
     for(int i = 0; i < len; i++) buffer[i] *= 0.5f;
     event.getOutput(0).getFloatBuffer().write(buffer);
  }
};
AudioClient.Info info = new DataLine.Info(AudioClient.class, format);
AudioClient client = (AudioClient)AudioSystem.getLine(info);
client.open(processor, format);
client.start();
```

END