exmoplay

A JVM based movie player engine with some exclusive features

Outline of Talk

- Introduction: History/Objectives/Motivation
- Introduction to Video & Xuggler
- Architecture of exmoplay
- Lessons Learned
- Demo

History of Video on the JVM

- Java Media Framework (JMF),
  - last updated 2001
- Fobs4JMF (JMF backend based on FFMPEG)
  - last updated 2008, now officially discontinued
- JavaFX Video in Swing (unsupported hack and license did not allow it), 2008
- VLCJ: Java API for VLC media player (native libvlc),
- Xuggler (FFMPEG wrapper), since 2009

Why writing my own video player?

- Not flexible enough
  - Frame-by-frame navigation
  - Change position forwards and backwards as an animation
- Bugs
  - Player froze or even crashed JVM after a longer time
  - Unexplicable sudden mute (app restart needed)
- Difficult to work with
  - Not possible to set player position while playing
  - delayed start/stop

Introduction to Video & Xuggler

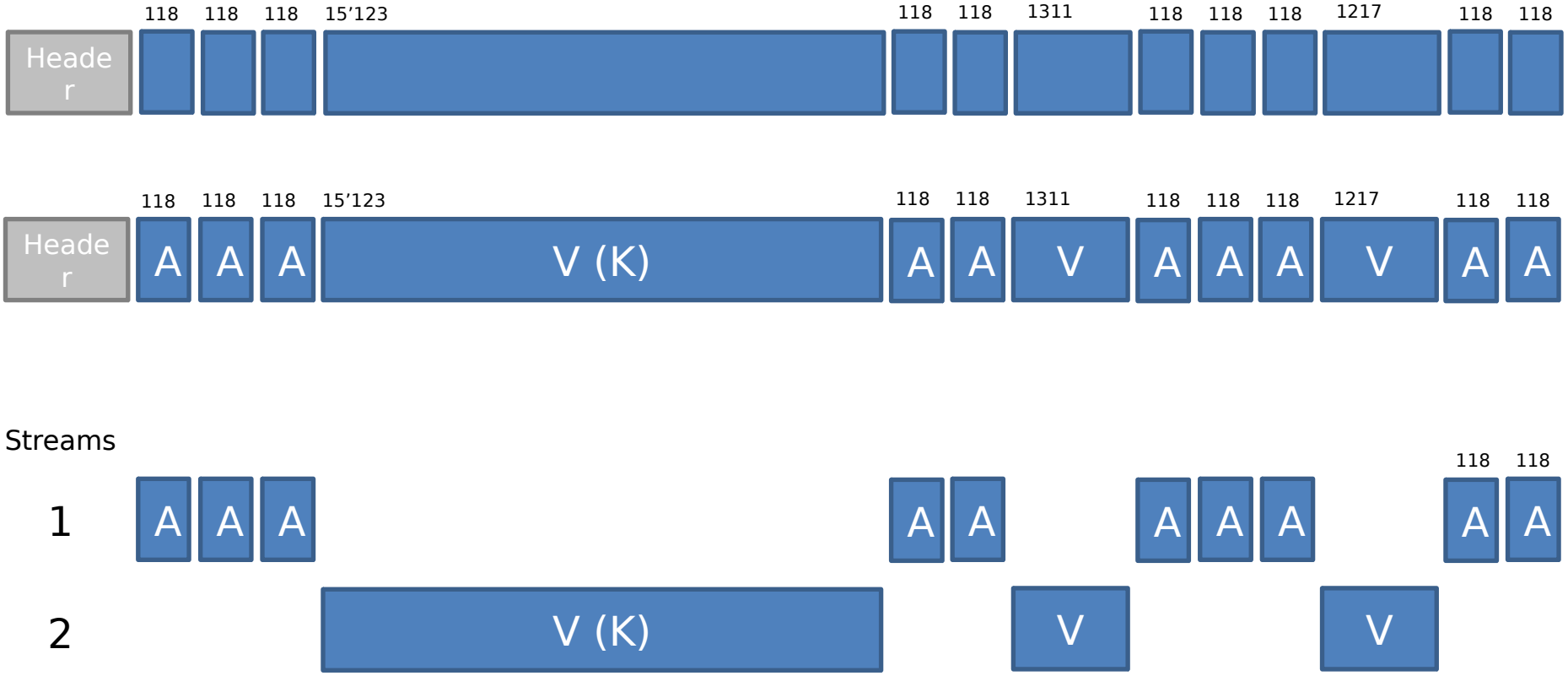
Video Basics

- Container Formats:
  - AVI
  - Flash Video (.flv)
  - Apple Quick Time (.mov)
  - Mp4
  - Ogg
  - Matroska
  - 3GPP (.3gp)

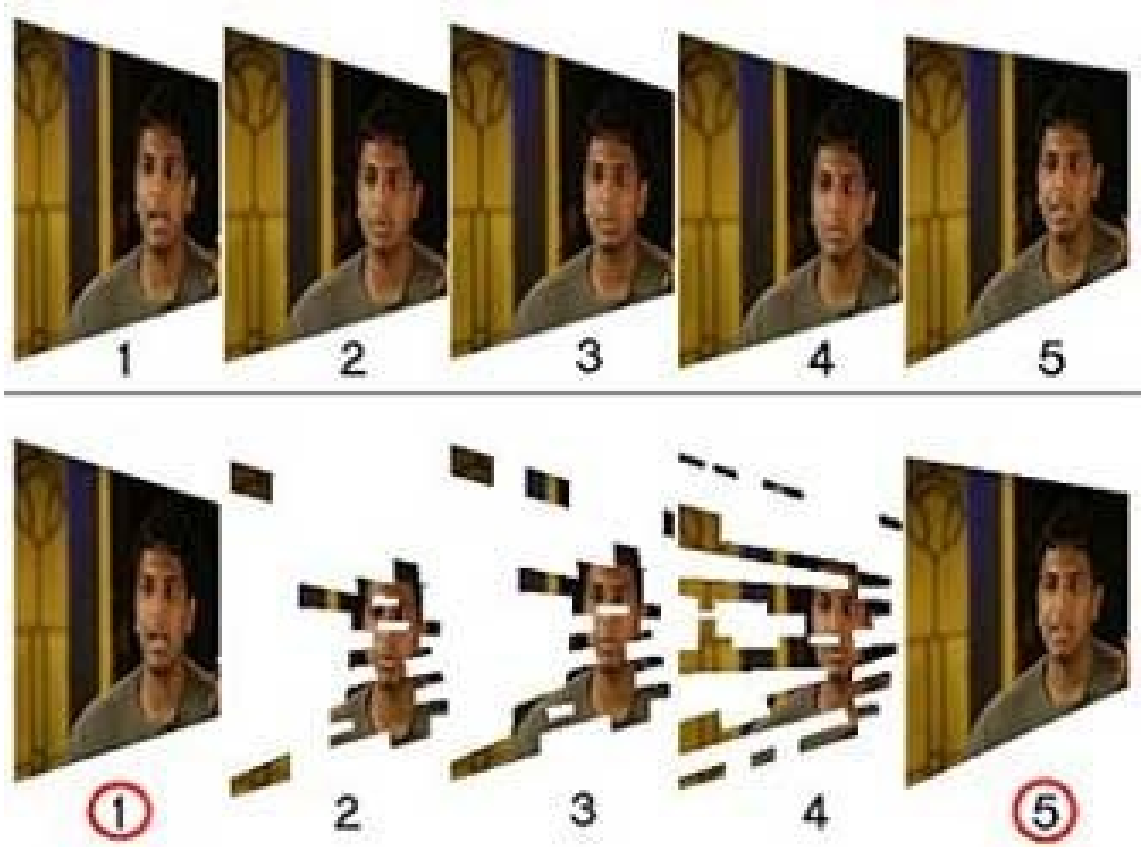
Video Basics

- Audio Codecs
  - MP3 (MPEG-1, Layer 3)
  - AAC (Advanced Audio Coding)
  - Vorbis
  - Flac
- Video Codecs
  - H264/MPEG-4 AVC
  - Theora
  - Divx

Container Format - Packets



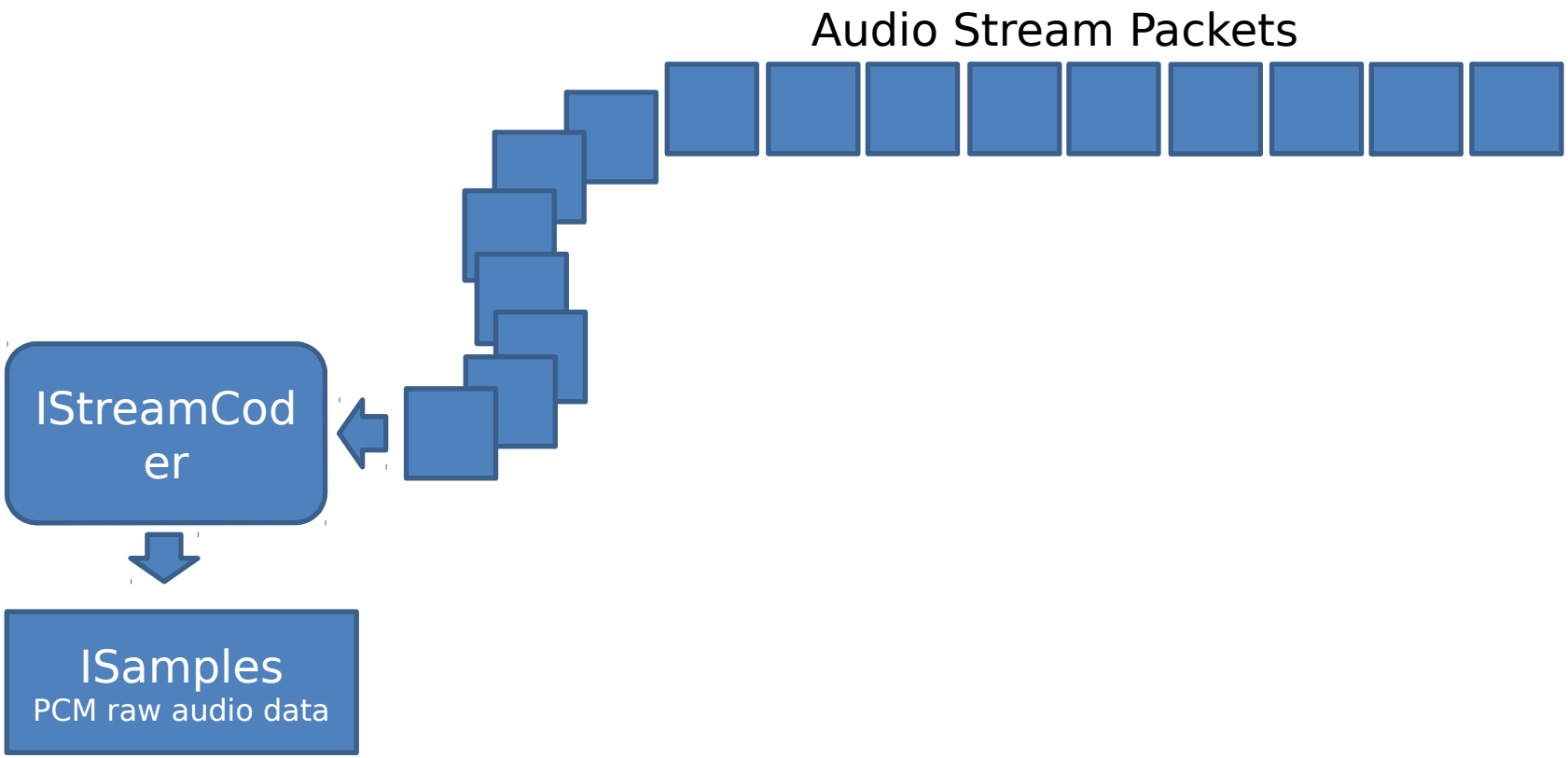
Key Frames



Xuggler Example Code

```
IContainer container = IContainer.make();
if (container.open("myfile.flv", IContainer.Type.READ, null) <0)
    throw new RuntimeException("failed to open");
int numStreams = container.getNumStreams();
for(i = 0; i < numStreams; i++) {
    IStream stream = container.getStream(i);
    //query IStream for stream information...
}
IPacket packet = IPacket.make();
while(container.readNextPacket(packet) >= 0) {
    //Do something with the packet...
}
container.close();
```

Decoding Audio



Xuggler Decoding Example

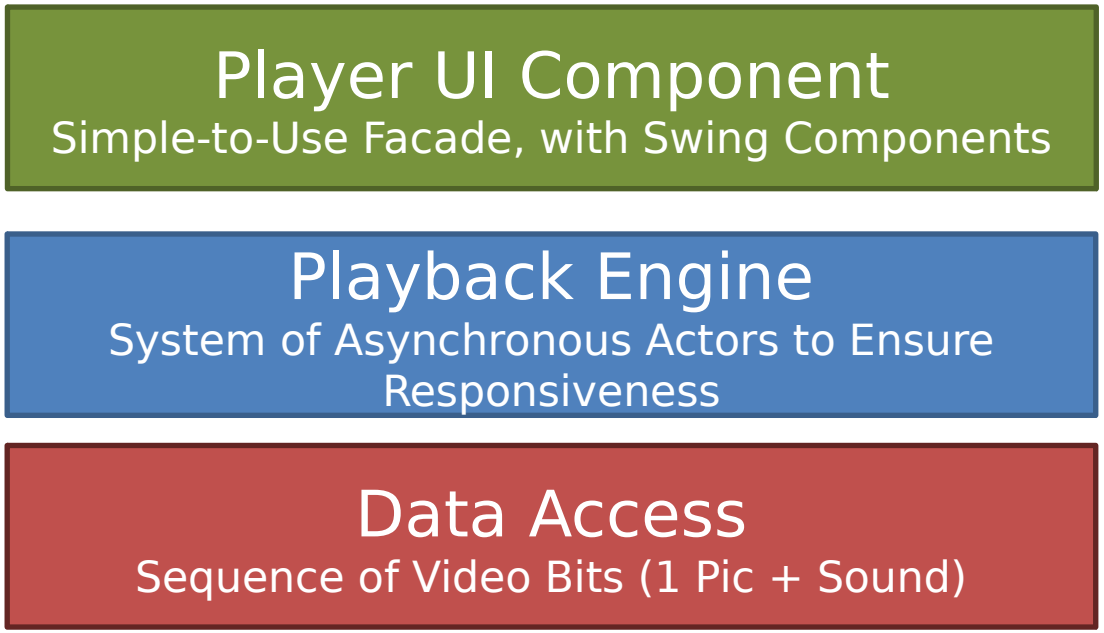
```
IStreamCoder audioCoder = audioStream.getStreamCoder();
IAudioSamples samples = IAudioSamples.make(1024,
        audioCoder.getChannels());
int offset = 0;
// Keep going until we've processed all data
while(offset < packet.getSize()) {
    int bytesDecoded = audioCoder.decodeAudio(samples, packet, offset);
    if (bytesDecoded < 0)
        throw new RuntimeException("error decoding audio");
    offset += bytesDecoded;
    if (samples.isComplete()) {
        byte[] rawBytes = samples.getData()
                .getByteArray(0, aSamples.getSize())
    }
}
```

Architecture of exmoplay

Requirements

- Exact and immediate random access, down to each individual frame
- Playback forward and backwards at different speeds
- Reliably controllable: start, stop and new position allowed in any state of the player
- Multiple screens and control bars possible

Architecture



Data Access - MediaInputStream

MediaFrame createFrame()

void readFrame(MediaFrame mf)

long getPosition()

long setPosition(long millis)

long getDuration()

VideoFormat getVideoFormat()

AudioFormat getAudioFormat()

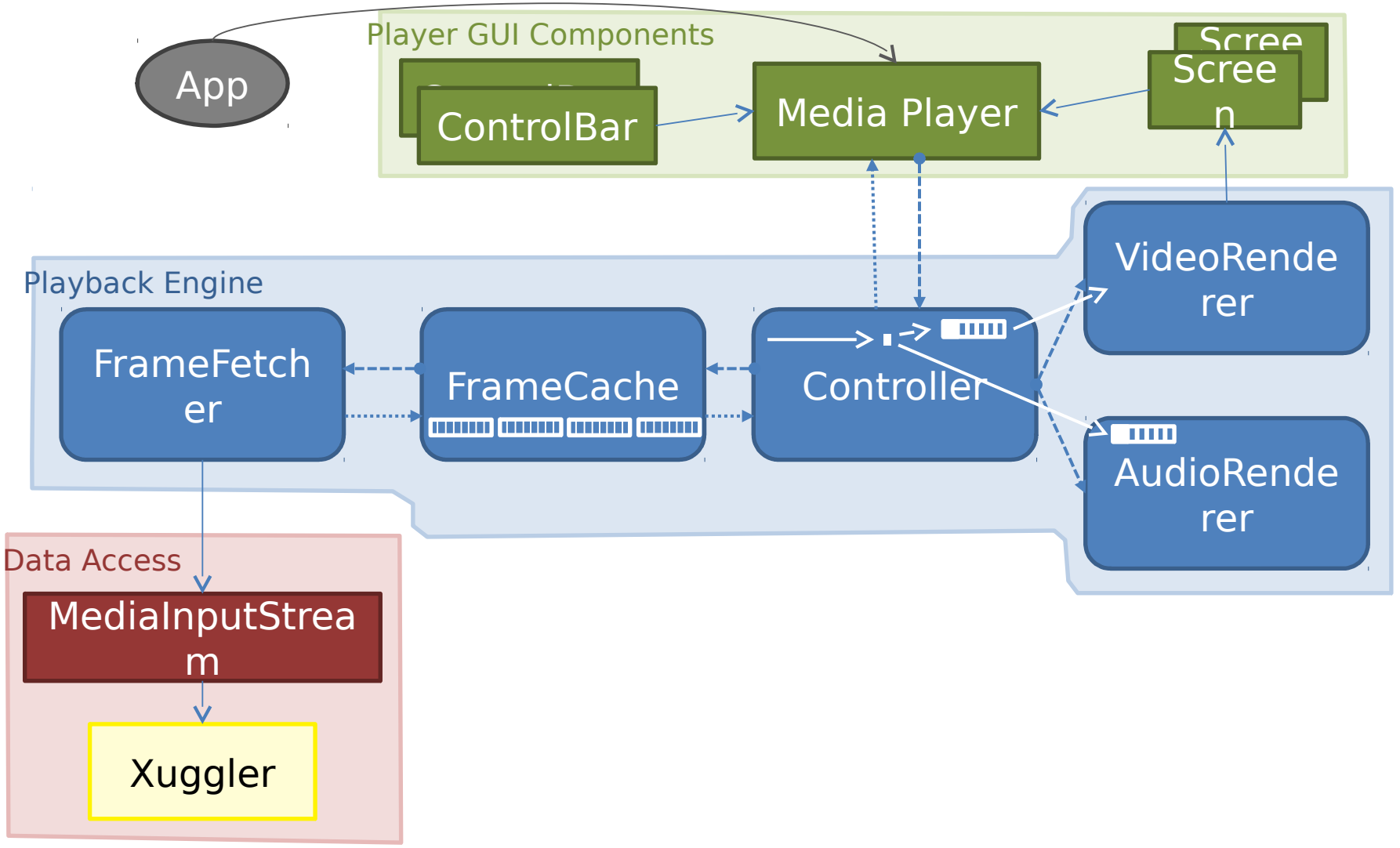
void close()

MediaFrame

- AudioBuffer (byte[] audioData, int size)
- VideoBuffer (IVideoPicture picture, BufferedImage image)

VideoFormat (String encoding, Dimension size, double frameRate)

Playback Engine



Player UI Component API

```
class MediaPlayer {
    VideoScreen createScreen()
    ControlBar createControlBar()
    void setActiveScreen(VideoScreen screen)
    void openVideo(File file, MediaInfo mediaInfo,
    long initialPosition, long timerMin, long timerMax)
    AudioFormat getAudioFormat()
    VideoFormat getVideoFormat()
    long getPosition()
    void setPosition(long position)
    void setPositionAnimated(long position)
    void start()
    void stop()
    void addPositionListener(final PositionListener listener)
    void restrictPositionRange(Long min, Long max)
    CachedFrame getFrame(long seqNum) // must call recycle() on frame after use
}
```

Feature Summary

- Fully asynchronous
  - based on own minimal actor framework
- LRU cache of frame blocks
  - Never blocking
  - Map with waiting requests
- Anti-Queue-Up mechanism to guarantee responsiveness

Feature Summary

- Multiple screen components possible
- Each command possible at any time (not depending on player state)
- Separation between user playing state and engine playing state
  - allows animated video position changes

General Lessons Learned

- Java Sound on Linux is CPU hungry and unreliable: Java Bindings for OpenAL (JOAL)
- An older version of Xuggler loaded native libs via LD_LIBRARY_PATH. Bad idea, because of conflicts with system libraries.
  - Explicitly load each of them including their dependencies using System.load() (not System.loadLibrary())
- GC can interrupt video playback.
  - Tune for low latency

Video Lessons Learned

- Decoded audio depends not just on the packet that was passed in, but on previous ones too
- Video meta data is not always correct
  - Expected size of decoded audio data
  - Frame rate, duration, time bases
- Seeking is not exact
  - Audio frames before video key frame missing
  - “false” keyframes can mean that seeking for 1100ms, which should go to 800ms, goes to 1600ms (“false” keyframe at 1000ms)
- Timestamps on video packets can be inaccurate ▭ only use those on decoded pictures
- Video decoders can be in a state where remaining frames have to be pulled out, before a key frame is accepted ▭ create new decoders after each seek
- Packets and frames often relate one to one, but not always
- For some formats, packets are only in order within a stream:
  - E.g. 10 video packets, 15 audio packets, 10 video packets, etc.

Unresolved Issues

- Video/Audio not in sync (different videos have different delays)
- Some videos (WMV) show only very few video key frames (sometimes 8 seconds without key frames, 8*30 = 240 frames)

Demo