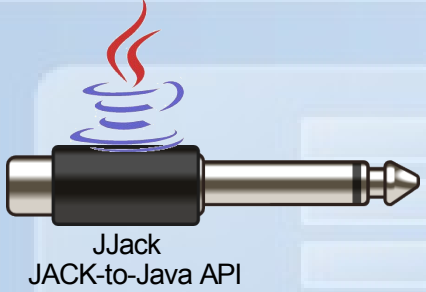
JJack

http://jjack.berlios.de

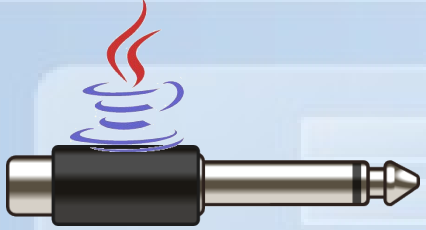
Using the JACK Audio Connection Kit with Java

Jens Gulden, jgulden@cs.tu-berlin.de



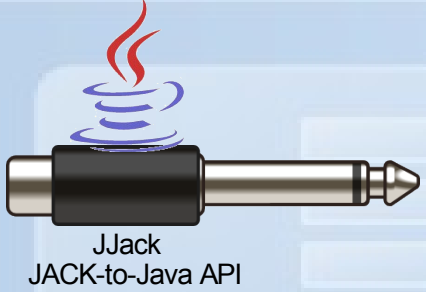
What's JJack for?

- Write **Java programs** that act as interconnectable **JACK clients**
- **Record** / **generate** / **transform** audio with Java, JACK-compliant
- Possible alternative to internal JavaSound API (`javax.sound.sampled.*`)

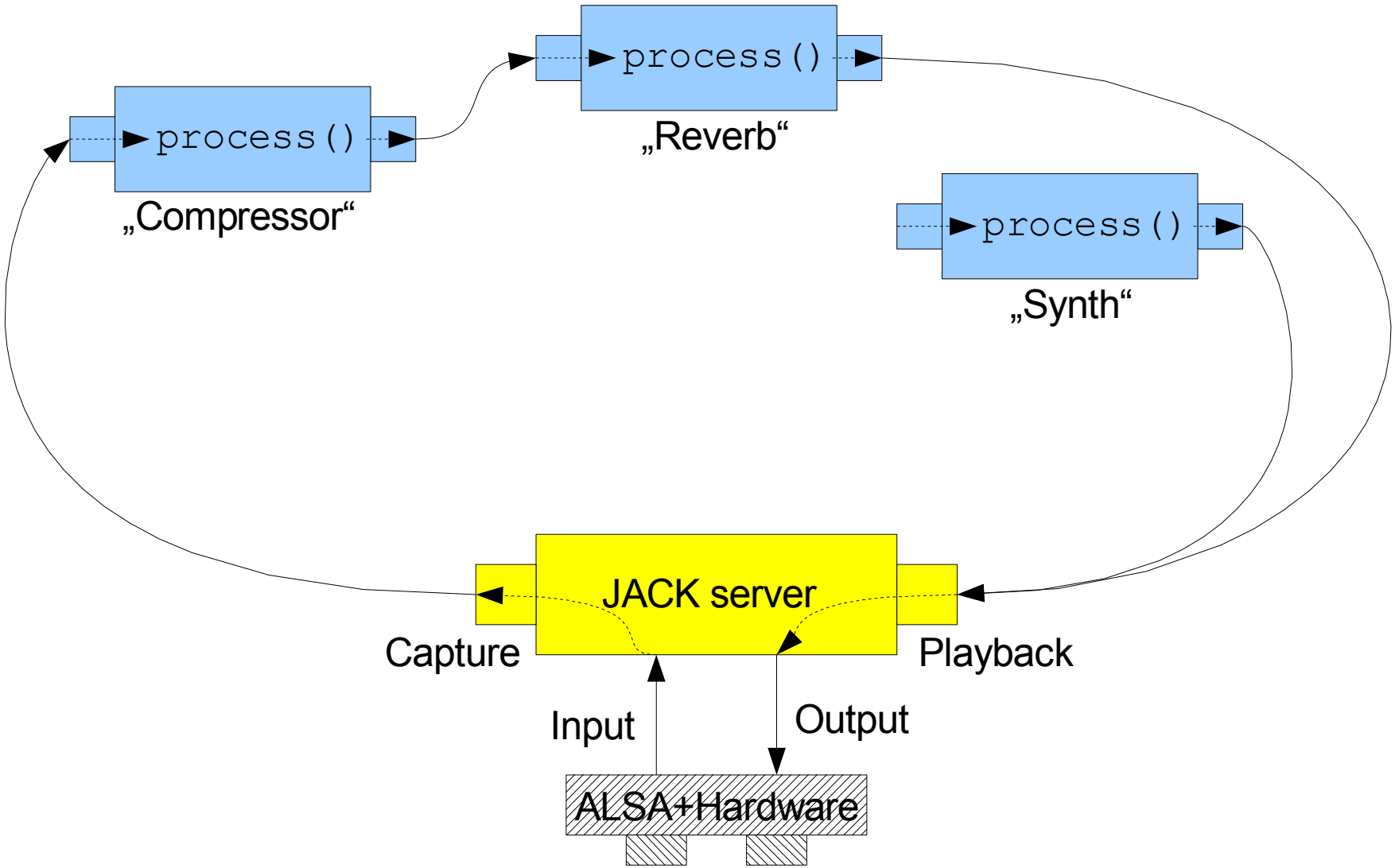


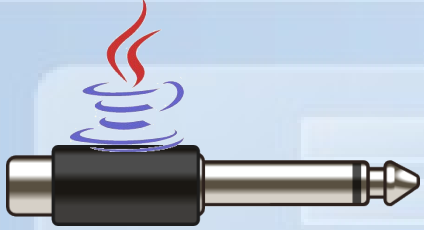
Overview

- JACK
- JACK & Java
- JJack direct JACK reflection
- JJack high-level API
- Examples



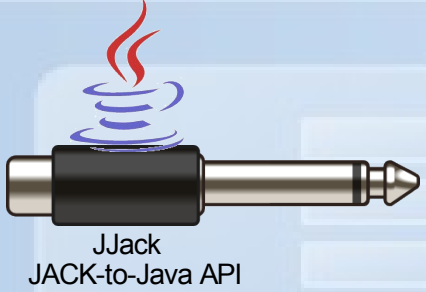
JACK processing loop



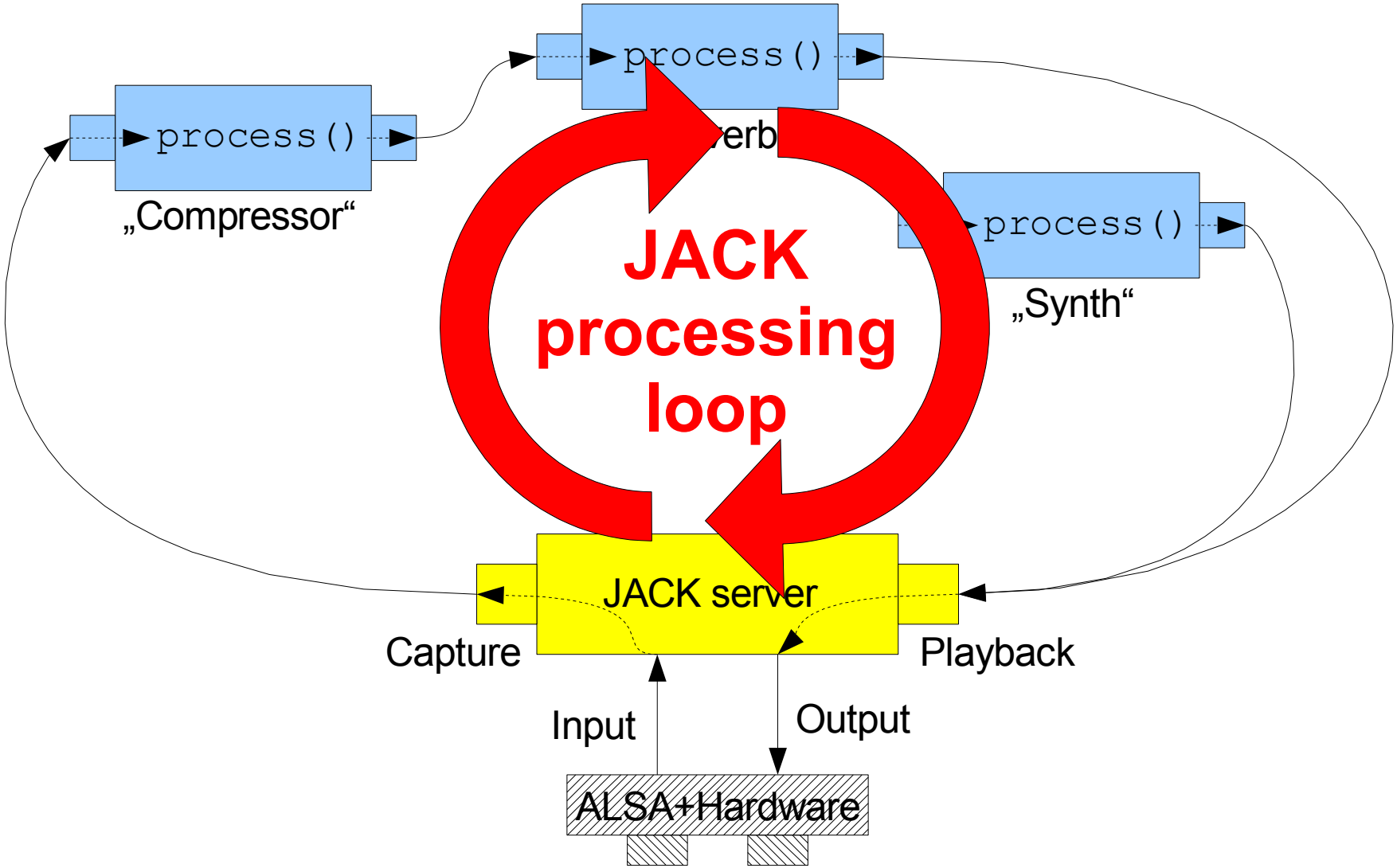


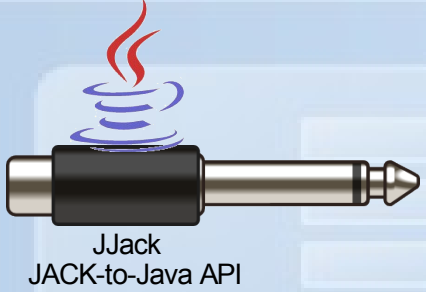
JACK processing loop

process()
„Reverb“
process()
„Compressor“
process()
„Synth“
Capture
Input
ALSA
JACK Audio Connection Kit [(default)] Started.
Start
Stop
Started -- 4.1 % 44100 Hz
19 (21)
00:00:00
Stopped --
Quit
Messages
Status
Setup...
Connect
Patchbay
Play
Pause
About...
Connections - JACK Audio Connection Kit
Audio
MIDI
Readable Clients / Output Port
alsa_pcm
Compressor
Reverb
Synth
Writable Clients / Input Ports
alsa_pcm
Compressor
Reverb
Connect
Disconnect
Disconnect All
Refresh

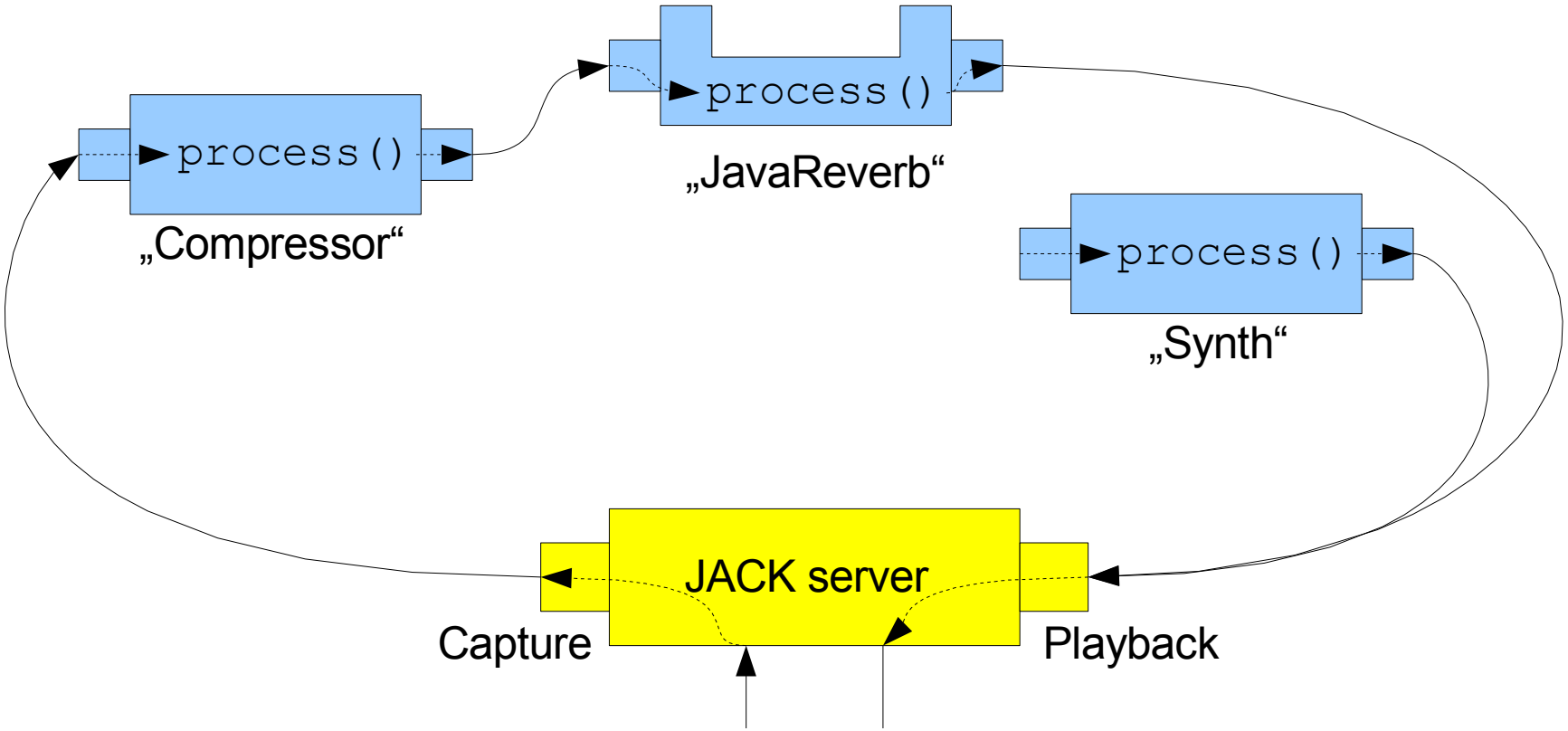


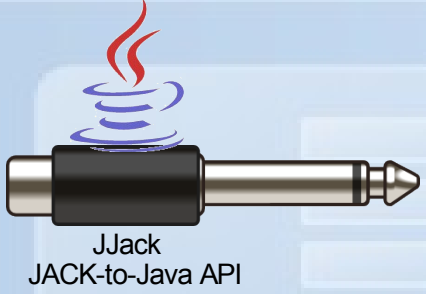
JACK processing loop



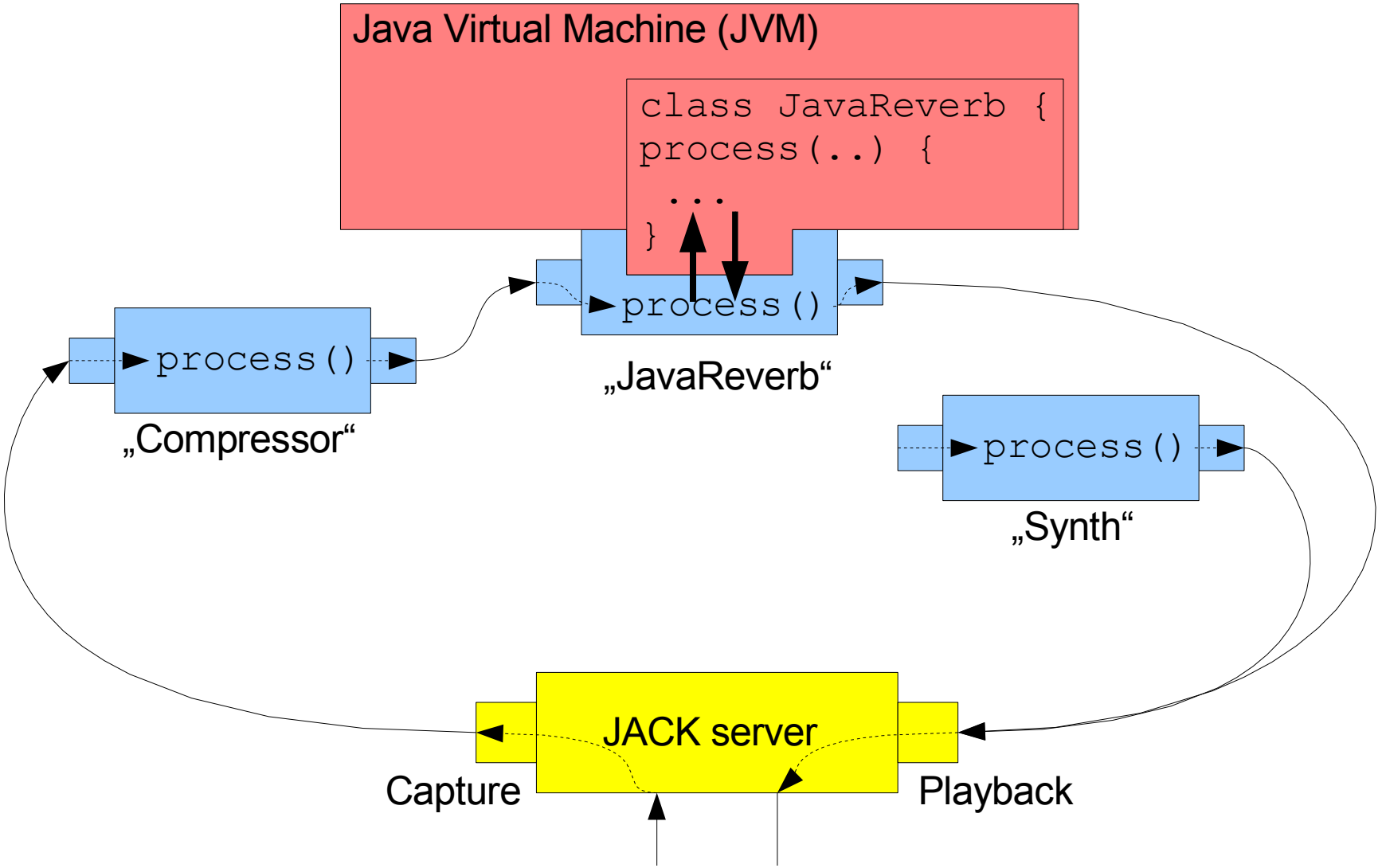

JJack
JACK-to-Java API

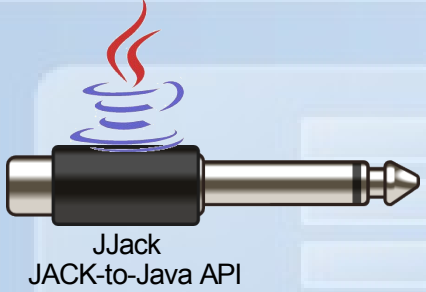
JACK & Java


process()
„Compressor“
process()
„JavaReverb“
process()
„Synth“
JACK server
Capture
Playback


JJack
JACK-to-Java API

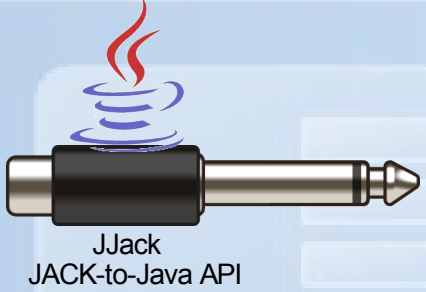
JACK & Java


Java Virtual Machine (JVM)
class JavaReverb {
process(..) {
...
}
process()
„JavaReverb“
process()
„Compressor“
process()
„Synth“
JACK server
Capture
Playback

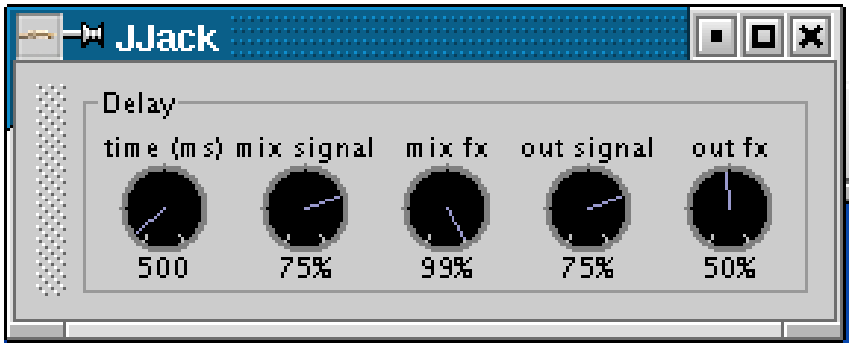


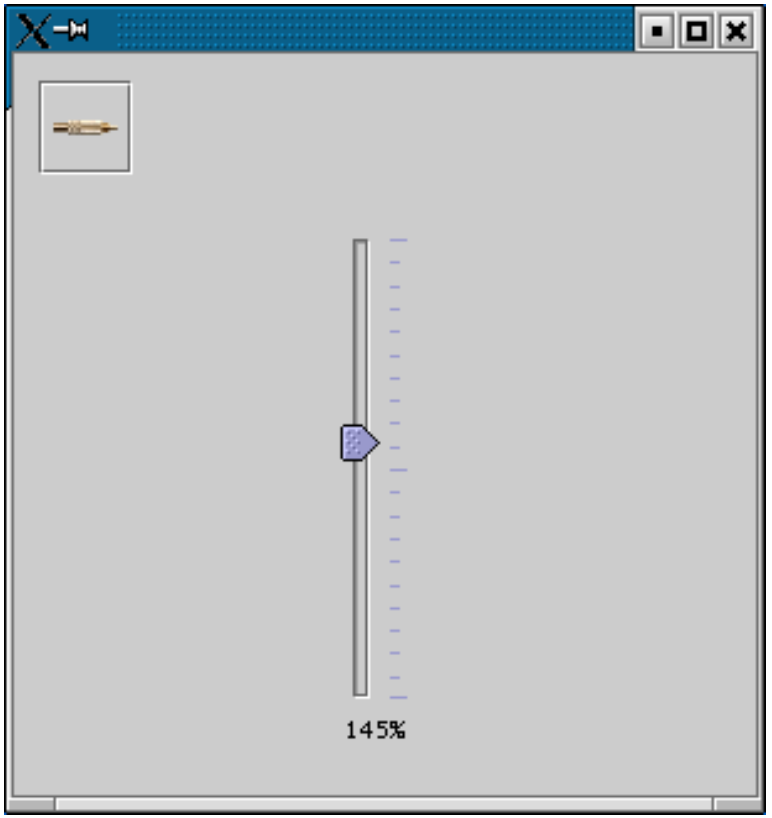
JJack Approach

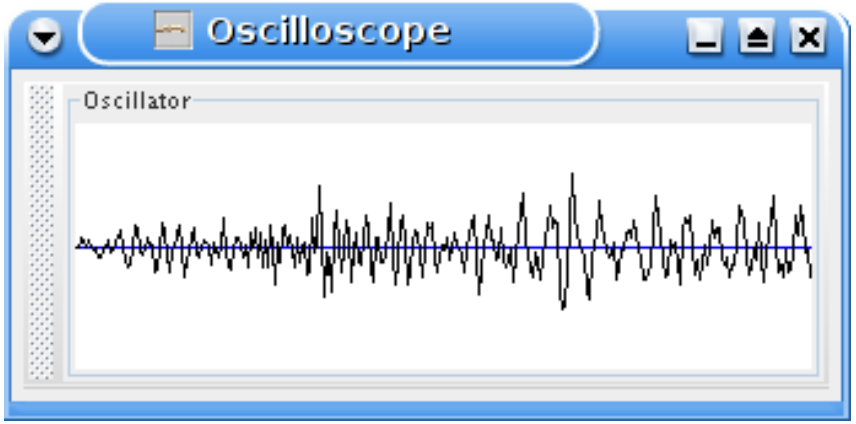
- Native library acts as JACK client (lib/i386/linux/libjjack.so)
- Bound to class JJackSystem by Java Native Interface (JNI)
- Native call to `process()` is delegated to Java-method `JJackSystem.process(..)`

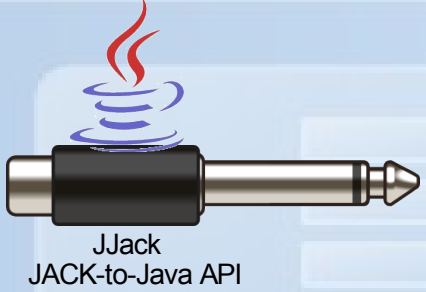


JJack – Examples 1



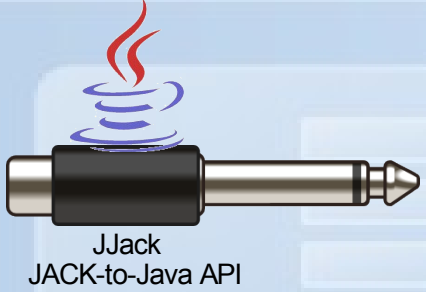






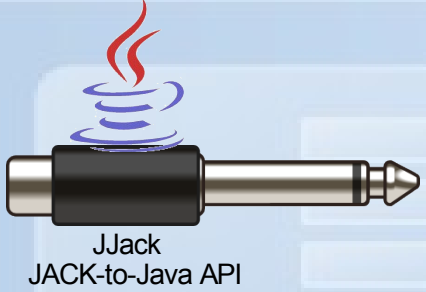
JJack Architecture

- 2 levels of architecture:
  - **Direct** JACK reflection (simple)
  - **High-level** API



Direct JACK reflection

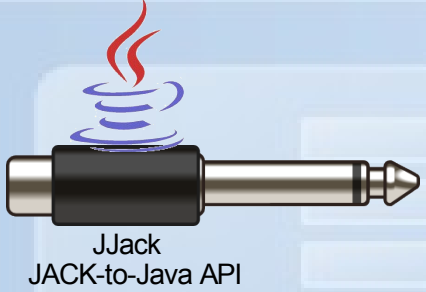
- Single `process()` method in Java does the job of a JACK client
- Easy to use
- Sufficient for most applications



Direct JACK reflection

- Simple JJack client in Java:

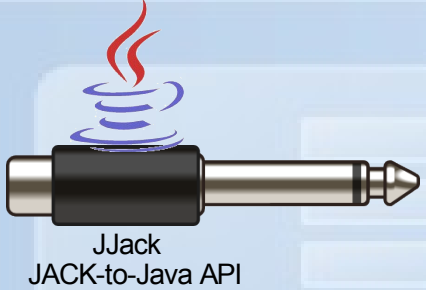
```
public void process(JJackAudioEvent e) {
    float v = getVolume(); // parameter from gui

    for (int i = 0; i < e.countChannels(); i++) {
        FloatBuffer in = e.getInput(i);
        FloatBuffer out = e.getOutput(i);
        int cap = in.capacity();
        for (int j = 0; j < cap; j++) {
            float a = in.get(j);
            a *= v; // amplify signal
            if (a>1f) a=1f; else if (a<-1f) a=-1f;
            out.put(j, a);
        }
    }
}
```



Direct JACK reflection

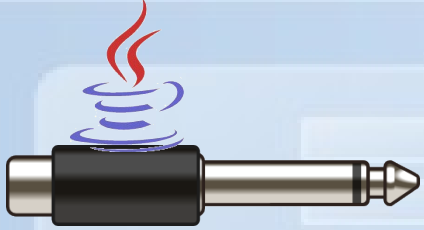
- Deploying:

```
public class JJackTest {

    public static void main(String[] args) {
        // get JACK system's sample rate, initialize
        int sampleRate = JJackSystem.getSampleRate();
        System.out.print("Sample-rate: "+ sampleRate);

        // set single processing client
        MyJJackClient client = new MyJJackClient();

        JJackSystem.setClient(client);

        /* ... */
    }
}
```

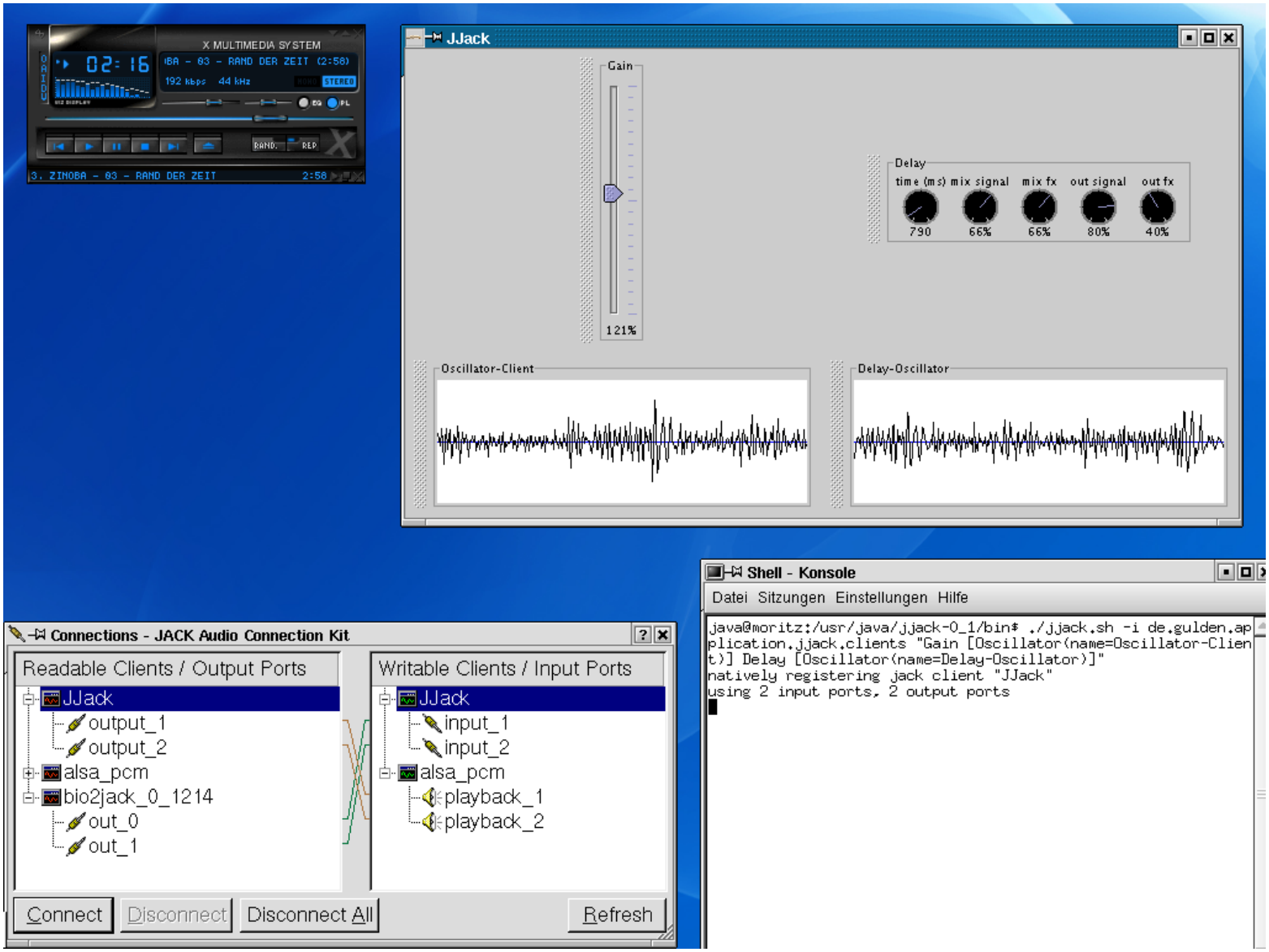


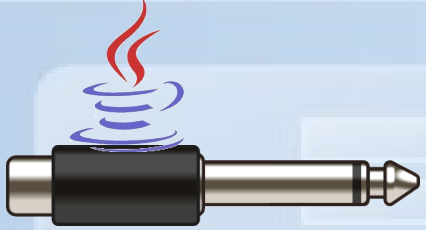
High-Level API

- Optional model for audio signal-routing, **internal** to Java
- Object-oriented API for **multiple** audio-processors
  - e.g. `Channel`, `Port`, `AudioProducer`, `AudioConsumer`...
- **JavaBeans**-compatible interconnection mechanism

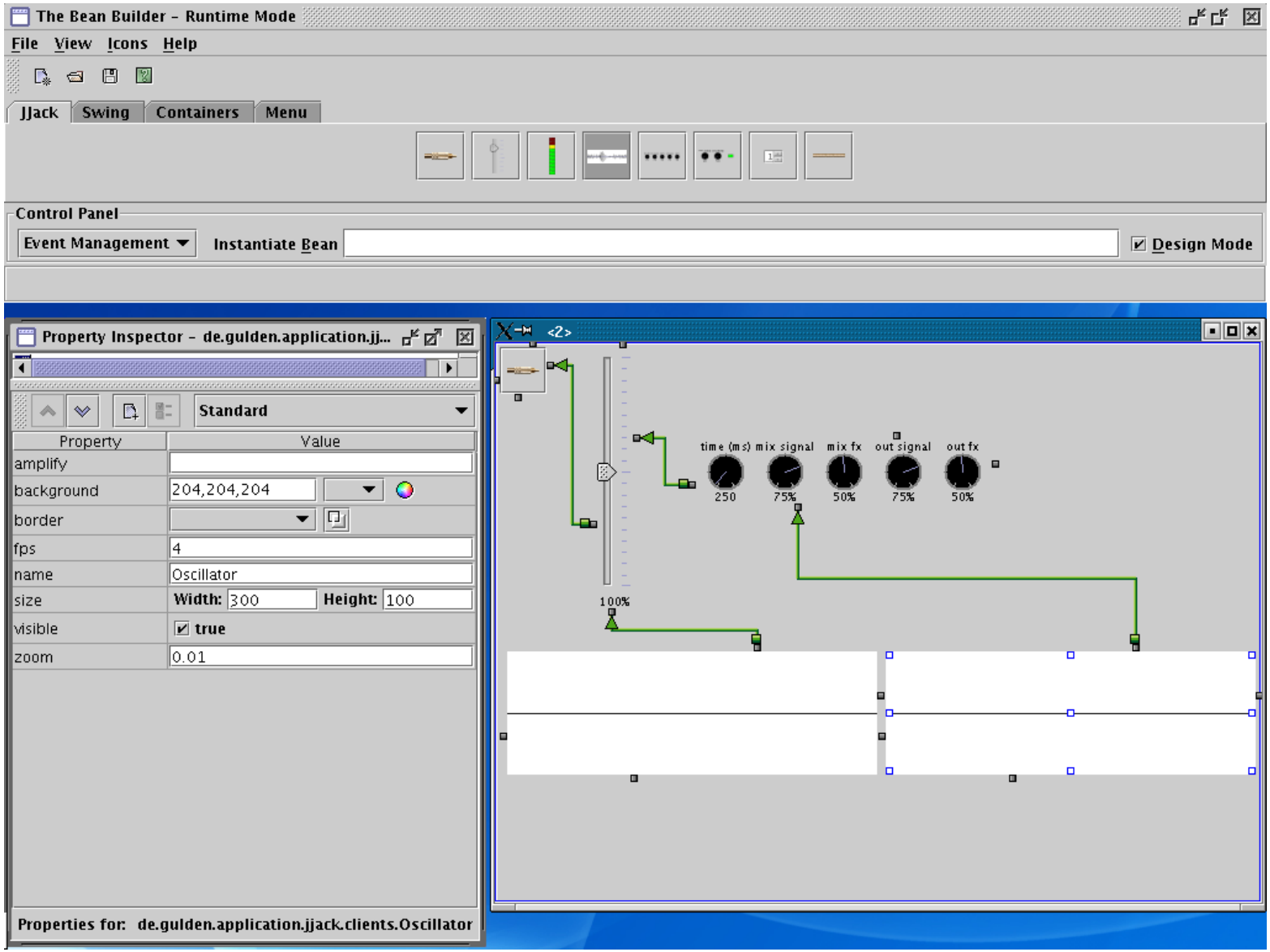


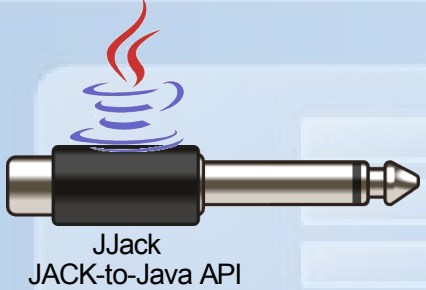
Example: High-Level API





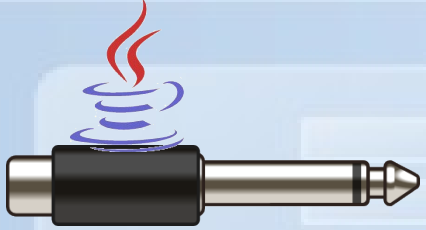
Example: BeanBuilder



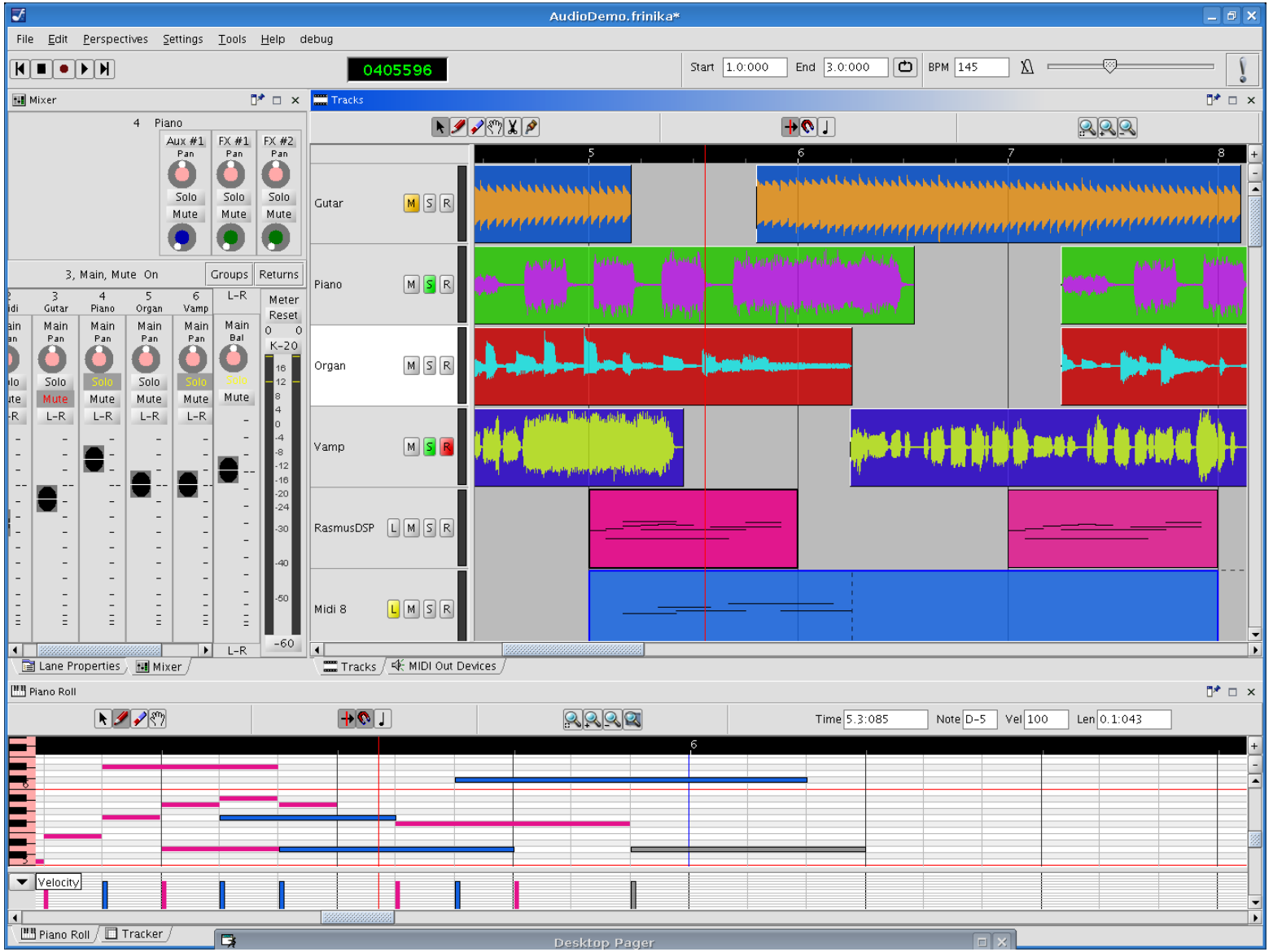


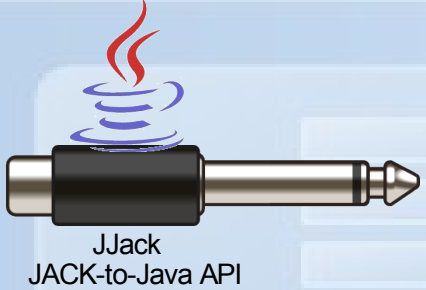
Real-World Application

- Music sequencer **Frinika** (GPL, `http://www.frinika.com/`)
- Entirely written in Java, implements sequencer model `javax.sound.midi.*`
- Audio I/O either via JJack or JavaSound



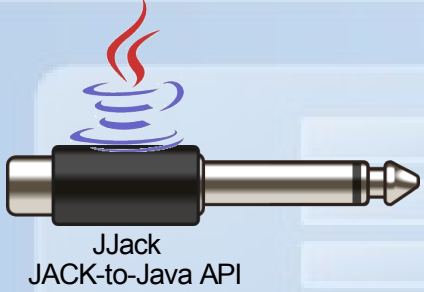
Example: Frinika





Native Bridge Implementation

- High performance thanks to JDK1.4 `java.nio.*` („New I/O“)
- Native `*float` memory-blocks get directly mapped onto Java `float[]` arrays (`java.nio.DirectByteBuffer`)
- *No conversion*, even no copying



Native Bridge Implementation

Java Virtual Machine (JVM)

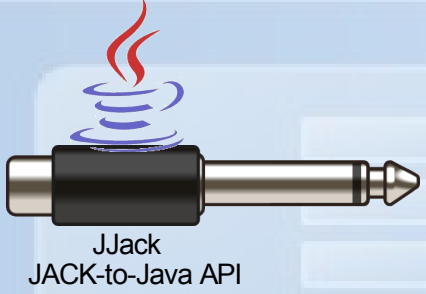
```
class MyClient {
public void process(...)
```

```
class JJackSystem {
public void process(
 JJackAudioEvent e) {
  client.process(e);
} }
```

JVM

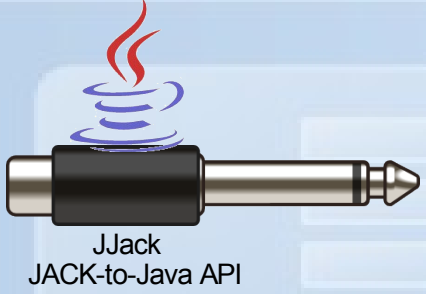
JNI, libjjack

```
int process(jack_nframes_t nframes, void *arg) {
 ...
 (*jvm)->AttachCurrentThread(jvm, ...);
 ...
 byteBuffer = (*env)->NewDirectByteBuffer(...);
 ...
 /* callback to Java, JJackSystem.process() */
(*env)->CallStaticVoidMethod(...);
 }
```



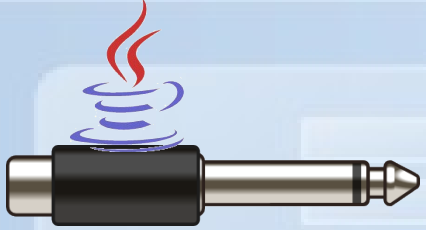
Comparing JJack and JavaSound (1/2)

	JJack	JavaSound
interoperability	high, virtual cable-connectivity	low, device I/O oriented
timing latency	JACK timing, potential low-latency below 5 ms on fast machines	device-dependent, varying among OSs and JDK versions
realtime-capability	yes (application can run in a user-thread, JACK in a realtime thread)	no (unless the whole JVM runs in a realtime thread)



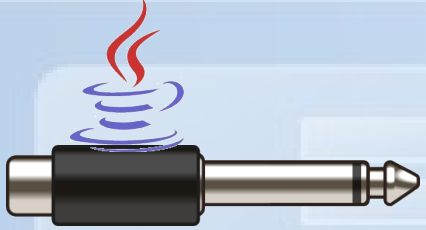
Comparing JJack and JavaSound (2/2)

process architecture	pull-architecture (JACK thread calls `process()`)	push-architecture (application is responsible for delivering data)
internal data-format	32-bit float (less aliasing, higher performance for interconnection)	16-bit integer (faster processing for simple clients, else conversion)
number of channels	any	according to hardware driver or software mixer
operating system	Linux or Mac (JACK required)	Any (JDK >= 1.3)



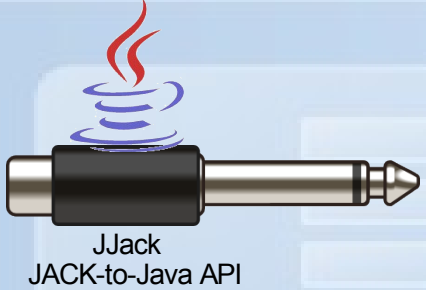
Future Plans

- Improve current codebase (with Frinika developers)
- Garbage-collection issues?
- JACK-MIDI?



Conclusion

- JJack **enables** Java as a programming language for **interconnectable** audio clients
- Audio-specific **architecture**
- May provide better **performance** than JavaSound (some systems)
- **Alternative** to JavaSound API

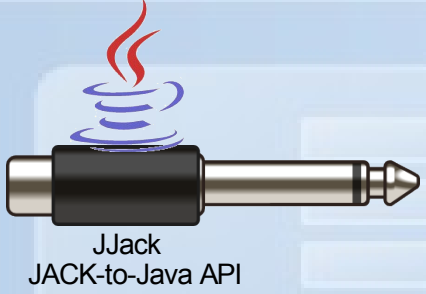


JJack – Using JACK with Java

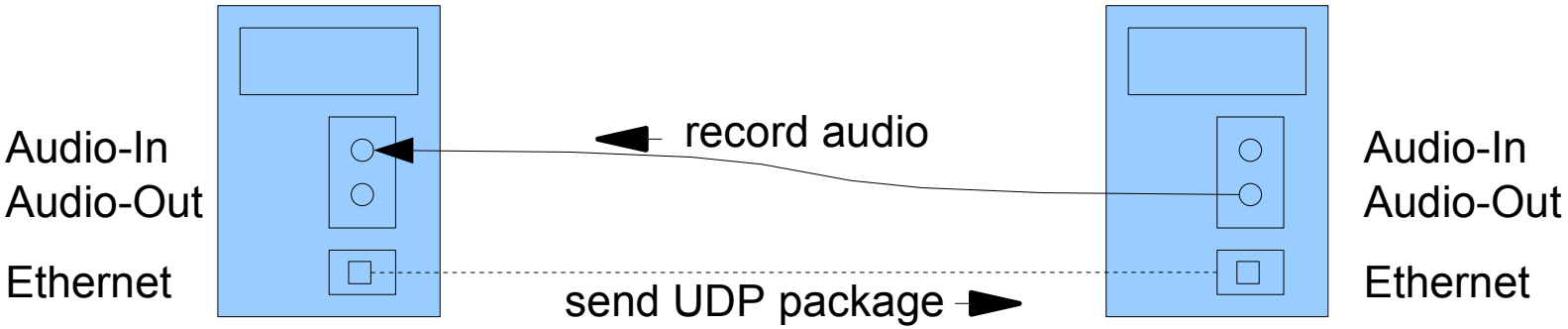
Thank you!

http://jjack.berlios.de/

Jens Gulden, jgulden@cs.tu-berlin.de



Appendix: Latency Measurements



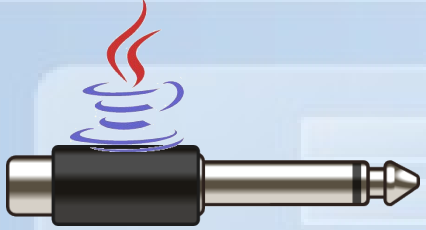
Test-Client

Linux 2.5.15-4-realtime, JJack with JACK-period 256, JDK1.5, onboard VIA hardware

Test-Server

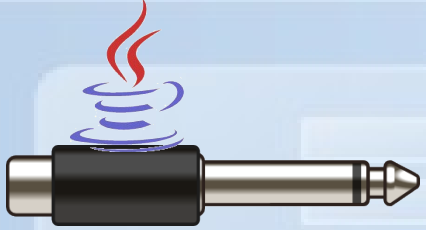
alternative configurations:

- Linux 2.6.15-4-realtime / Linux 2.6.15-5 / Windows XP
- JDK1.4 / JDK1.6
- Onboard-sound / USB-sound



Appendix: Latency Measurements

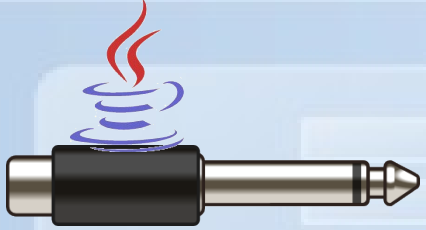
- Benchmark application:
  `de.gulden.framework.jjack.util.benchmark.AudioBenchmark`
  (current CVS)
- Note: test-client induces latency itself, suitable for comparing setups, but no absolute results



Appendix: Latency Measurements

Onboard-sound	JACK period	JJack JDK1.4	JJack JDK1.6	JavaSound JDK1.4	JavaSound JDK1.6	JavaSound Buffer
Linux 2.6.15-4-realtime	512			333.71		131071
	256	18.82				88200
	128				3.71	65536
	64	8.62				32768
	32		4.94	325.18		16384
Linux 2.6.15-5	512			329.47		131071
	256					88200
	128			336.95	4.13	65536
	64		6.82	337.56		32768
	32	6.86	5.39	342.26		16384
Windows XP				178.67		131071
					28.26	88200
						65536
						32768
				185.79	27.9	16384

The values are each averaged from at least 20 individual measurements.



Appendix: Latency Measurements

USB-sound	JACK period	JJack JDK1.4	JJack JDK1.6	JavaSound JDK1.4	JavaSound JDK1.6	JavaSound Buffer
Linux 2.6.15-4-realtime	512			372.3		131071
	256			354.34	19.52	88200
	128					65536
	64	10.74	10.34			32768
	32				20.04	16384
Linux 2.6.15-5	512					131071
	256			21.17	20.58	88200
	128					65536
	64	10.17	9.53			32768
	32				20.64	16384
Windows XP				179.5		131071
					31.02	88200
						65536
						32768
					30.71	16384

The values are each averaged from at least 20 individual measurements.