



FireWire Audio on Linux

The developers talk

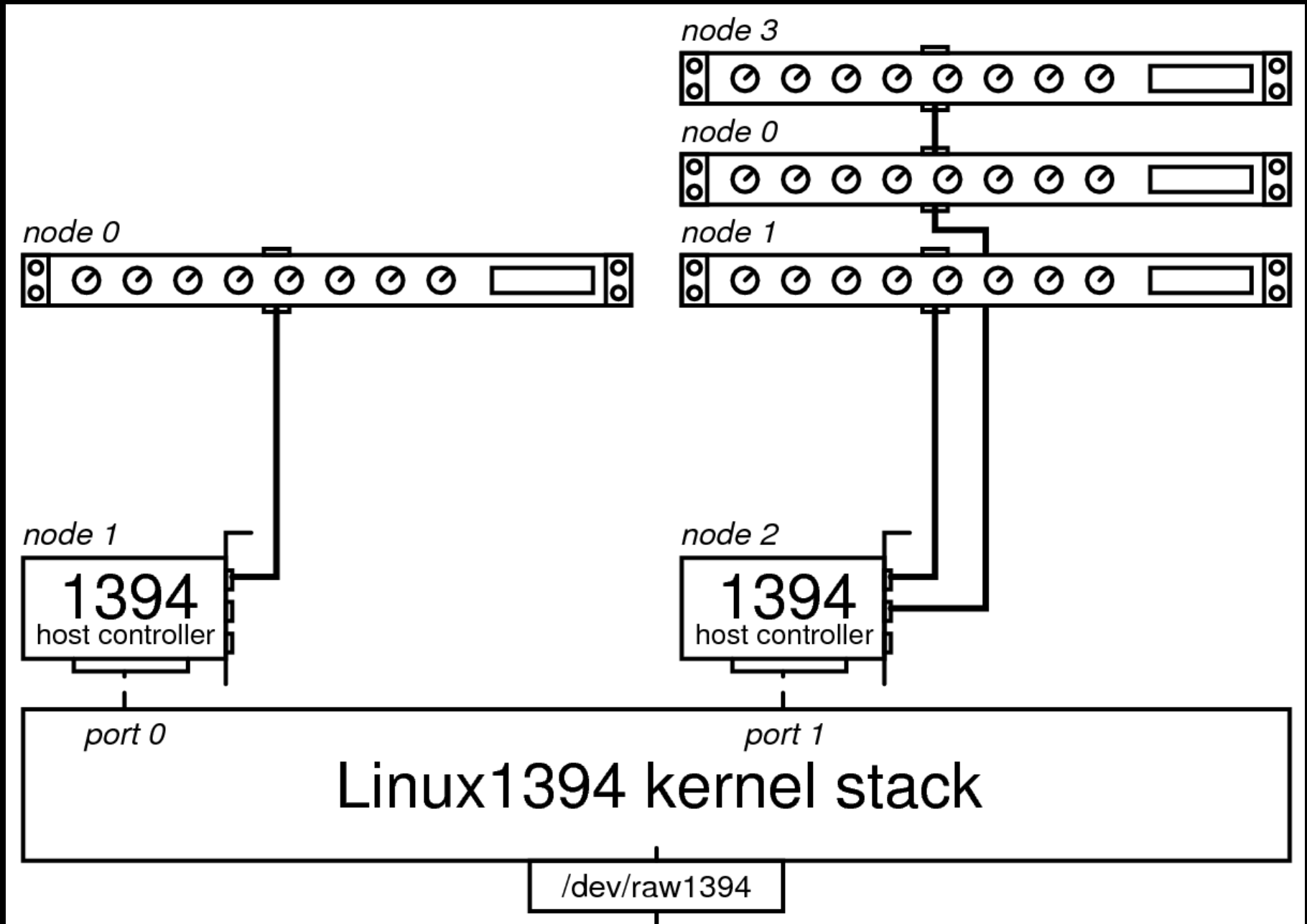
Pieter Palmers

Overview



- “The System”
 - What is FireWire?
 - FireWire on Linux
- “The Project”
 - History
 - Future
- “The Code”
 - Design
 - Implementation
- “The Final Words”

System Overview (1)



What is FireWire?



- a.k.a. IEEE1394, iLink (Sony)
- Serial bus specification
 - IEEE1394a
 - up to 400Mbit/sec
 - Specific twisted pair cable
 - IEEE1394b
 - up to ? 3200Mbit/sec ?
 - “FireWire” cable, CAT5 UTP, Fibre, ...
 - Backwards compatible
- Self configuring, peer-to-peer
 - Different 'roles' are elected (IRM, Bus Master, ...)

Addressing

- One 64-bit address space
 - 10 bit Bus ID
 - 6 bit Node ID
 - 48 bit Node Address space
- Results in
 - 1023 busses (one 'local bus ID')
 - 63 nodes (one broadcast ID)
 - 256 terrabyte device address space
- Node ID's are dynamic
 - (Can) Change every bus reset

Time



- 8000 cycles per second
- One global clock available
 - Cycle Timer (Register)
 - Mapped in memory space
 - 24.576MHz
 - period = “tick”
 - Cycle timer expressed as:
 - 7bit: Seconds (wraps at 127 secs)
 - 13bit: Cycles (wraps at 8000cycles)
 - 12bit: Offset in Ticks (wraps at 3072 ticks)

Data Transfer



- Two data transfer types
 - Isochronous
 - Broadcast one-to-many on a 'channel'
 - Occurs at regular time intervals (125us period)
 - Node can and may send at max one packet per 'cycle'
 - Up to 80% bandwidth can be reserved & guaranteed
 - No acknowledge or response
 - No error detection or correction
 - Asynchronous
 - Send-ack-respond 'transactions'
 - Targets a address (range) in the 64bit memory space
 - Allows error detection & correction
 - Between two specific nodes
 - No bandwidth or timing guarantee

FireWire on Linux



- Kernel modules
 - ohci1394: Host controller access (hardware)
 - ieee1394: General bus & link management
 - raw1394: Application layer (transactions, iso, ...)
- Userspace libraries
 - libraw1394
 - Userspace counterpart of raw1394 module
 - libavc1394 (depends on libraw1394)
 - Basic implementation of some 1394TA AV/C standards
 - libiec61883 (depends on libraw1394)
 - Basic implementation of the IEC61883-x standards

History

- First prototype demo at LAC2005
- FreeBoB 1.0
 - Alpha/beta testing started February 2006
 - Release Candidate 1: May 22, 2006
 - 1.0 release: October 18, 2006
 - 1.0.3 release: May 15, 2007
- Supported features
 - BeBoB devices
 - Audio & Midi streaming
- Unsupported
 - Constant round-trip latency
 - Mixer control, sync, ...
 - ALSA

History



- Meanwhile...
 - somewhere “down under”
 - Jonathan Woithe reverse engineered the MOTU 828mkII
 - at LAC2006...
 - Meeting regarding DICE-II device support
- One-driver-fit's-all idea?
- FreeBoB-2 branch forked
 - Rewrite of the codebase
 - Device agnostic framework

Goals for the Future



- Driver framework for all FireWire audio devices
 - BeBoB, DICE-II, Motu, Metric Halo, ...
 - Device agnostic
- All features supported
 - Complete device control
 - Audio & midi streaming
- Userspace library
- Audio API's:
 - JACK, JACK-MP
 - ALSA plugin
 - ...

Beyond code



- 80% of the interfaces are (will be) based either on a BeBoB or a DICE-II platform
- Platform support only takes you half way
 - Back-end provided by platform vendors
 - Front-end customized by equipment vendors
- First end-user experience is “that other half”
- Serious device driver development means
 - Good platform vendor contacts
 - Good equipment vendor contacts
- Good *vendor contacts mean politics

Bye bye BoB

freeBoB

- Objections on the name have been expressed
- “FreeBoB” + “FreeDICE” ?
- Name changes suck, but...
what's in a name?
- Let's get it over with...



“Free Firewire Audio Drivers (Oh oh)”

Current Status



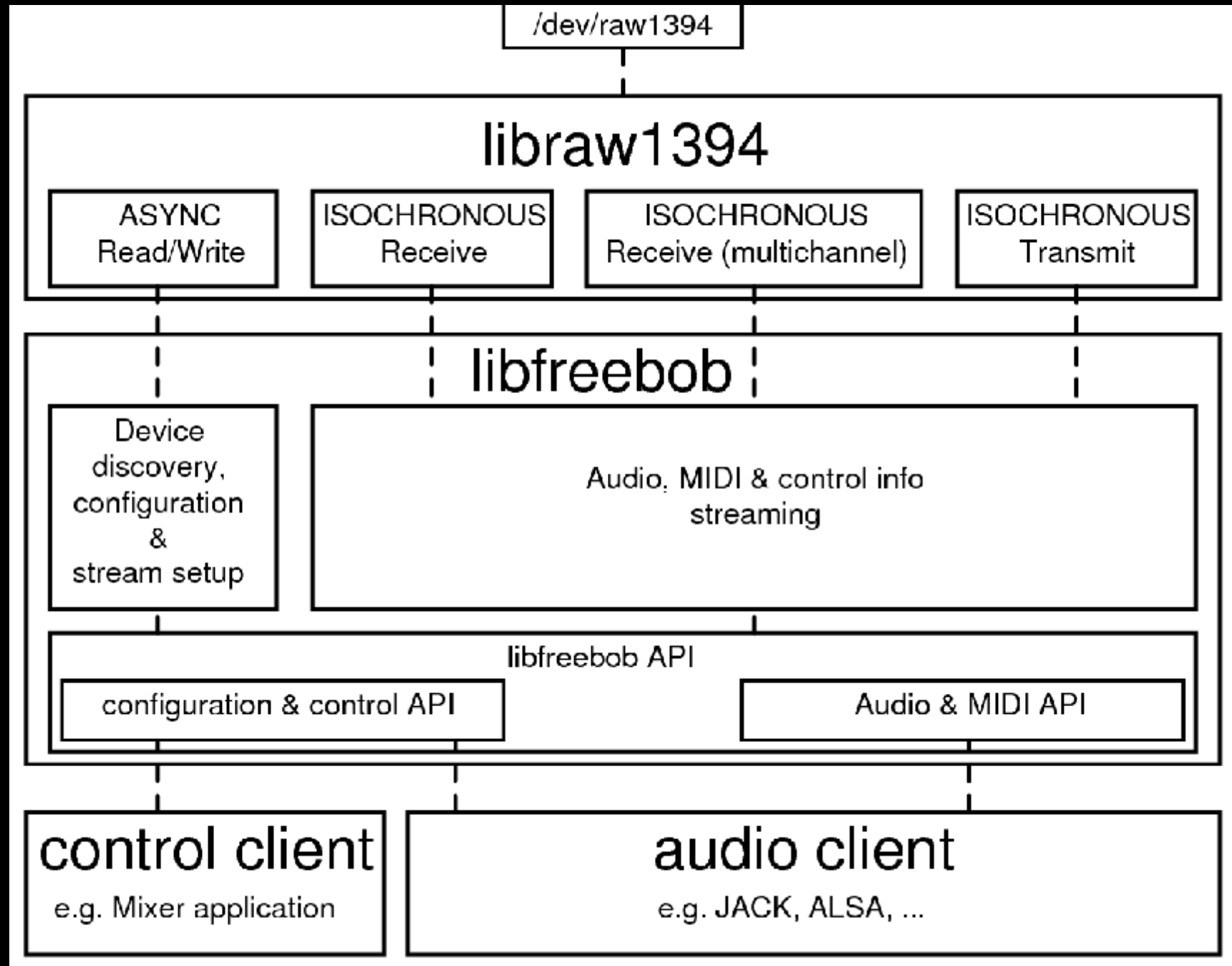
- Work done:
 - Low level device interfaces
 - BeBoB platform
 - DICE-II platform
 - MOTU Traveller & 828mkII
 - Audio & midi streaming @ all samplerates
 - Jack backend
- Work being done:
 - Debugging & testing
 - Device control (mixer, sync setup, ...)
 - Low level code (almost) ready
 - External control API being designed (OSC?)
 - Configuration persistence

Current Status (2)



- Work to be done:
 - Device control
 - Mixer/control application
 - Streaming
 - SPDIF/AC3/DTS pass-through
 - Jack-MP backend port from 1.x
 - ALSA plugin
 - Device interfaces:
 - Metric Halo
 - Port to new 1394 “JUJU” kernel stack

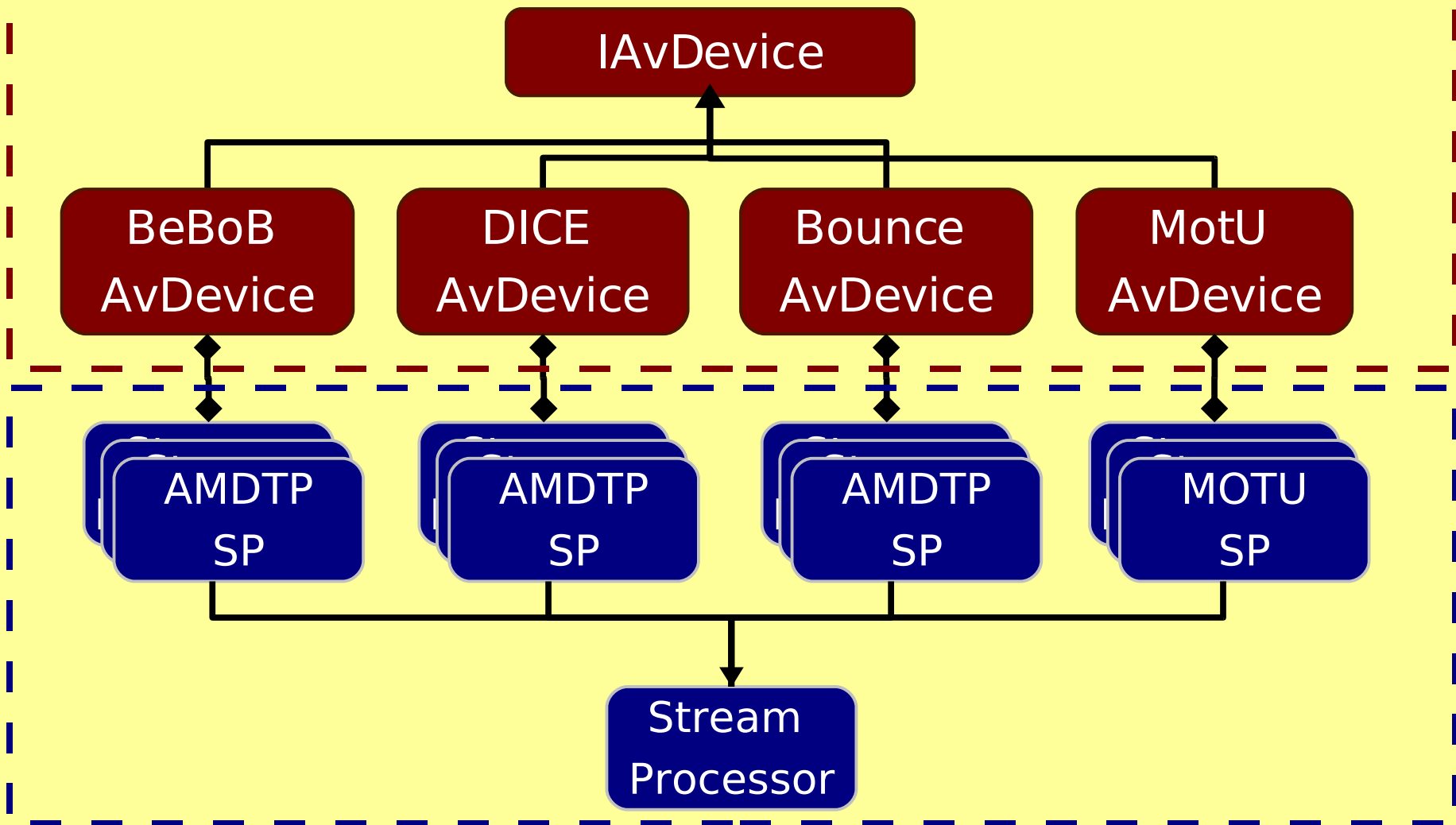
System Overview (2)



Internal Structure

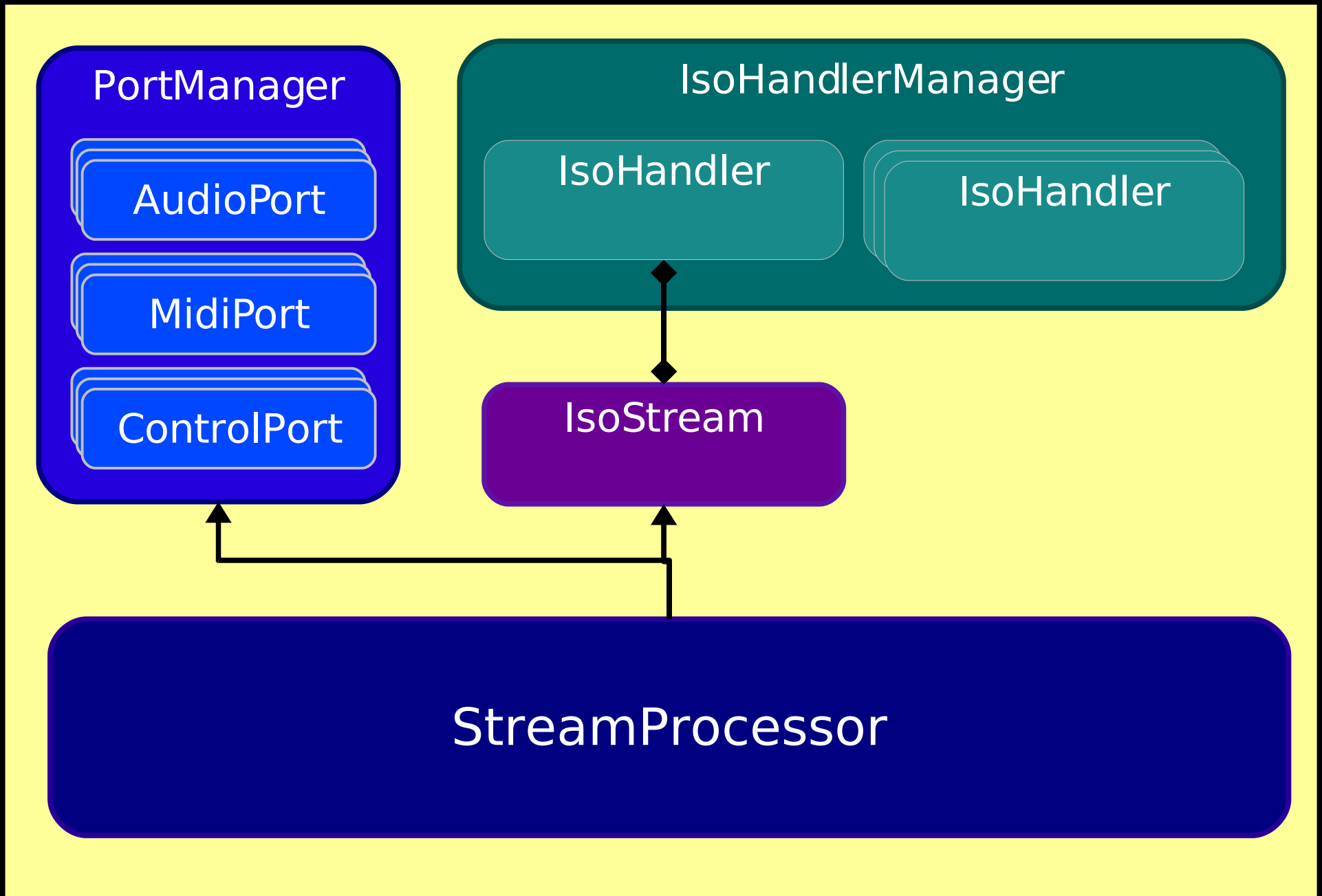


Discovery & Configuration

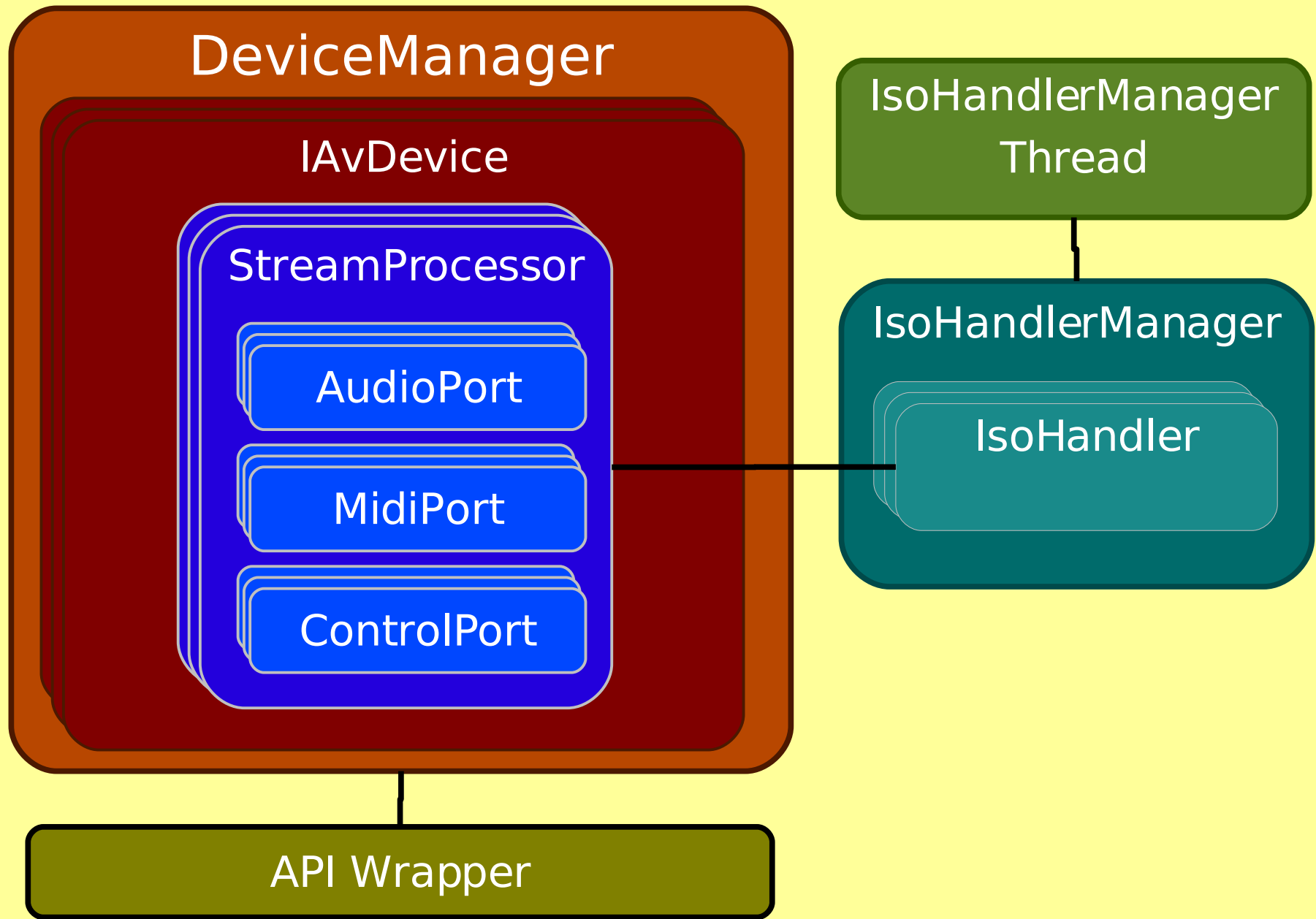


Streaming

Stream Processor



Device Manager



Intermezzo: Synchronization



- IEC61883-6 SYT time stamp: (simplified)
 - Every packet (= 8 frames @ 48k) contains one
 - Value of the FireWire Cycle Timer at the sample instant + TRANSFER_DELAY
 - “The receiver shall present the block at SYT_TIME + PROCESSING_DELAY”
- Implementation is a PITA
 - Convert SYT & CTR from/to ticks (wrap/unwrap)
 - Buffering introduces difficulties
 - Predict the CTR @ 1 buffer ready
 - Map CTR to system time (DLL)
 - Needed extension of kernel API to allow reading the cycle timer and the system time atomically
 - Sleep()
 - Dependencies between 'sync master' and other SP's

Open issues



- What if you want to use multiple topologies?
 - Synchronization
- How do we handle FireWire based “processing units”?
 - TC PowerCore, SSL Duende, Eventide 8000
 - JACK backend, Jack client, LADSPA, LV2, ...
- Inter-process communication
 - Control application feedback muxed into Iso stream
 - Interface as backend & “processing unit” as client or plugin

Conclusions

FreeBoB



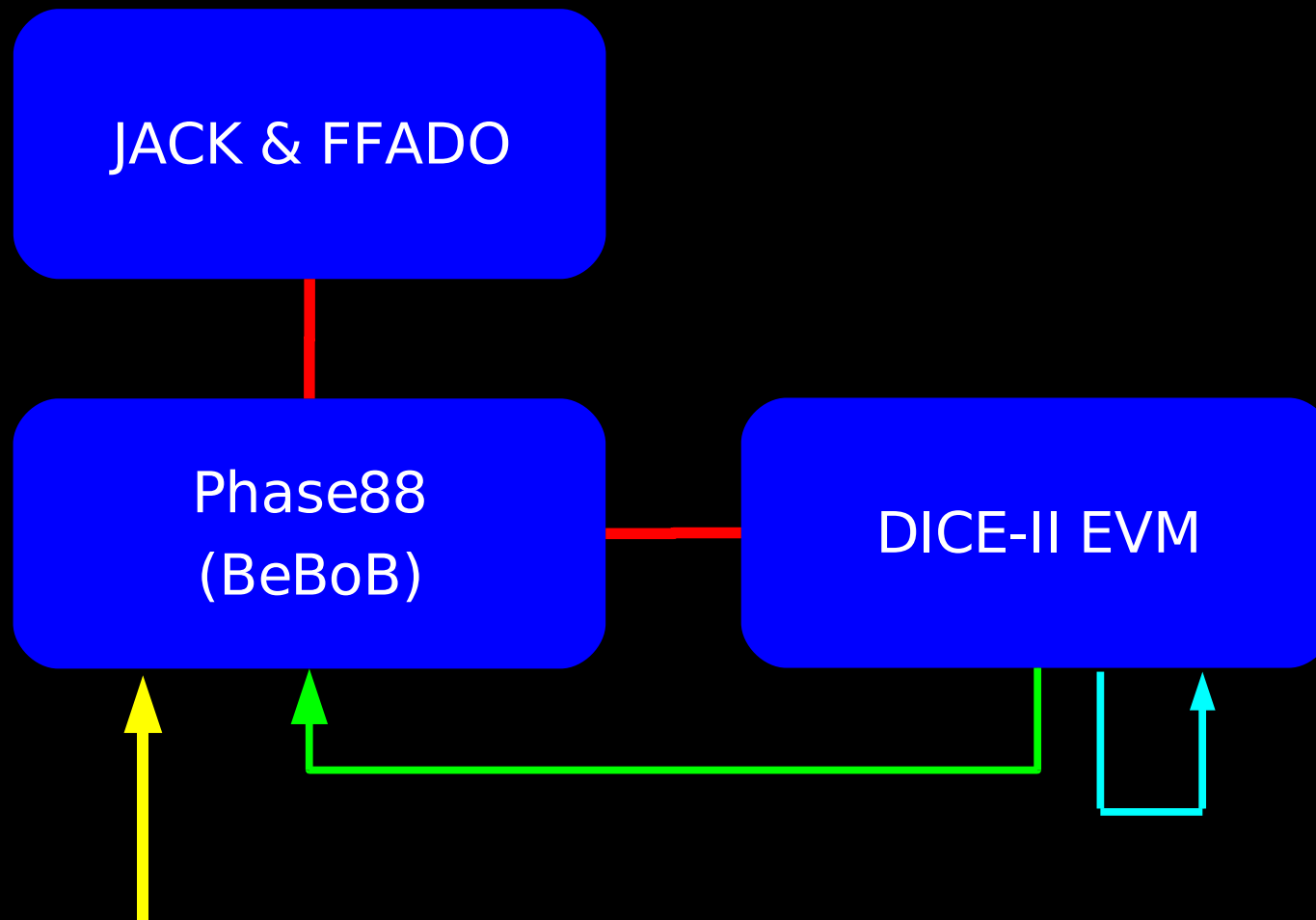
- Completeness
 - FreeBoB 1.X provides 70%
 - FFADO will provide 95%
- Vendor contacts
 - Platform vendors: good
 - Equipment vendors: work in progress
- We could use developers

Acknowledgments



- Daniel Wagner
 - FreeBoB founder (respect)
 - AV/C implementation (respect++)
- Jonathan Woithe, Olaf Christ
 - MotU reverse engineering
- Leonard Ritter
 - Original FreeBoB logo
- Thorsten Wilms
 - New logo design
- Testers, website, ...

Demo (take 2)



FireWire
WordClock
Analog
ADAT