Bela: an embedded platform for audio and sensor processing

Giulio Moro

c4dm.eecs.qmul.ac.uk

instrumentslab.org

bela.io

EPSRC
centre for digital music
Queen Mary
University of London

Queen Mary University of London

centre for digital music

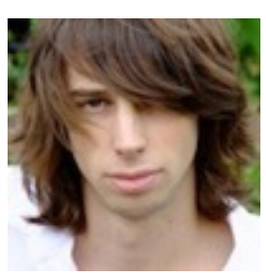
The **Augmented Instruments Laboratory** is a team within the Centre for Digital Music, a research group at Queen Mary University of London. Our work focuses on new musical instruments and interactive audio systems.

Andrew McPherson
Reader in Digital Media
Lab director
Augmented piano

Fabio Morreale
Postdoc
Instrument design for virtuosity

Kurijn Buys
Postdoc
Instrument design for virtuosity

Astrid Bin
PhD student
Study of error and risk in performance



Liam Donovan
PhD student
Active vibration control of strings

Laurel Pardue
PhD student / Postdoc
Augmented violin

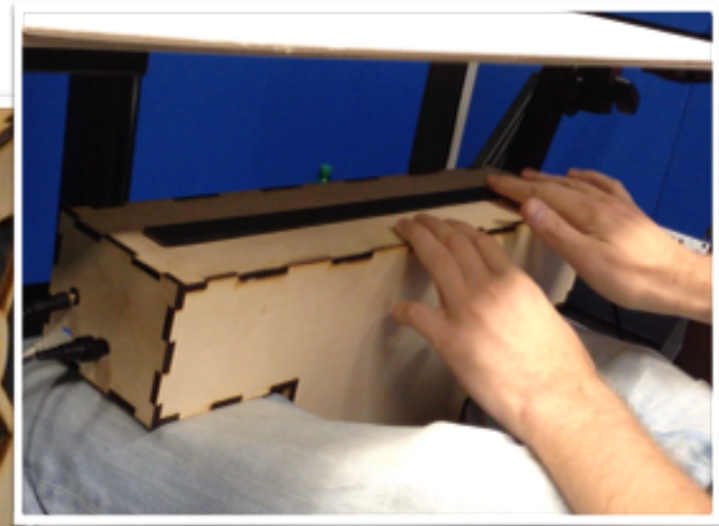
Robert Jack
PhD student
Tactile feedback in digital instruments

Giulio Moro
PhD student
Acoustics of the Hammond Organ

Christian Heinrichs
PhD student / Postdoc
Digital Foley artistry

Jack Armitage
PhD student
Craft of digital musical instruments

Jacob Harrison
PhD student
Accessible musical instrument design

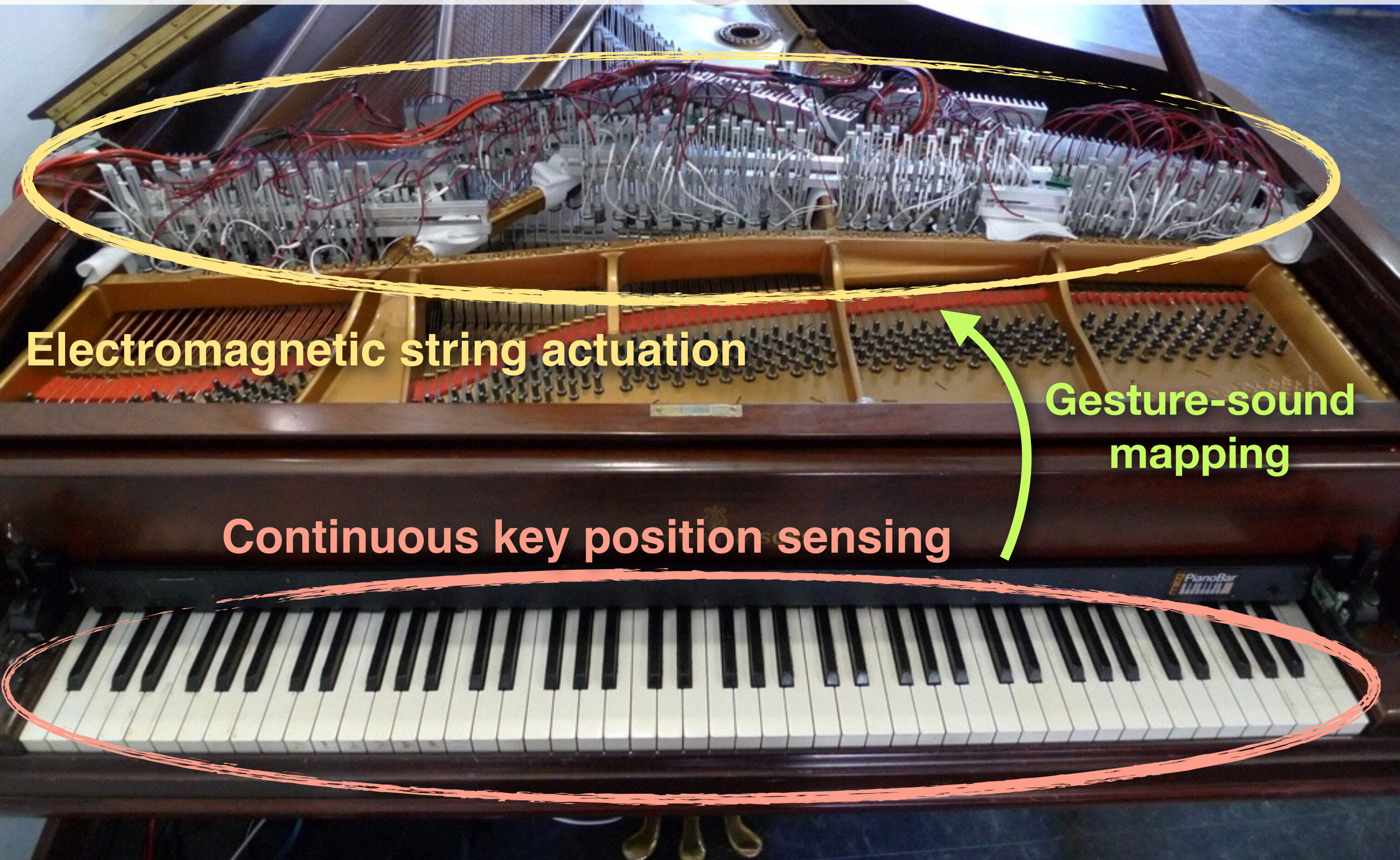


More info on the Augmented Instruments Laboratory:

http://instrumentslab.org

Magnetic Resonator Piano

Electromagnetically-augmented acoustic piano



https://www.youtube.com/watch?v=f79d_oVqv4Y

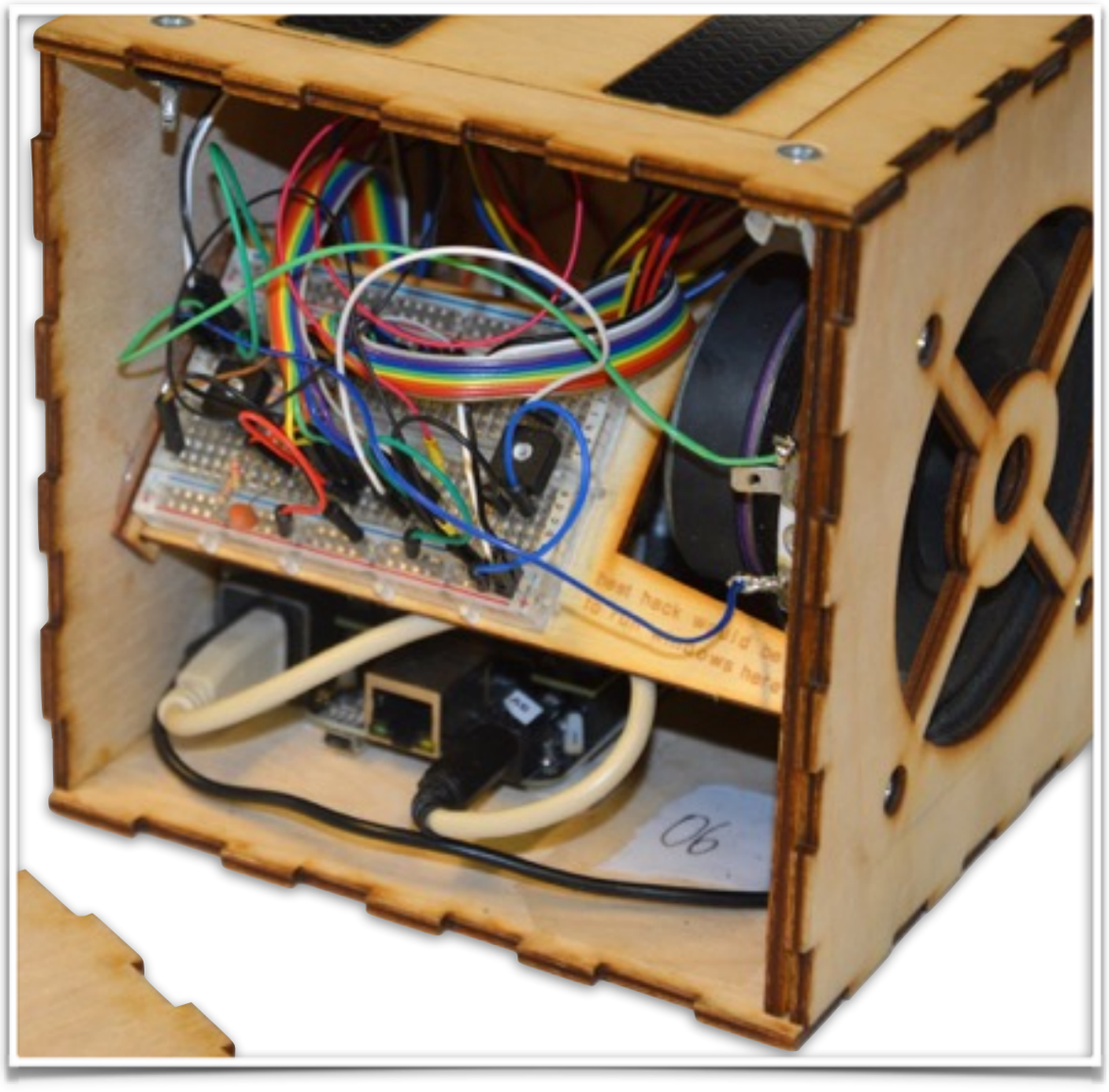
TouchKeys

Capacitive multi-touch sensing on the surface of the keyboard

https://www.youtube.com/watch?v=InNZkNz4NMc

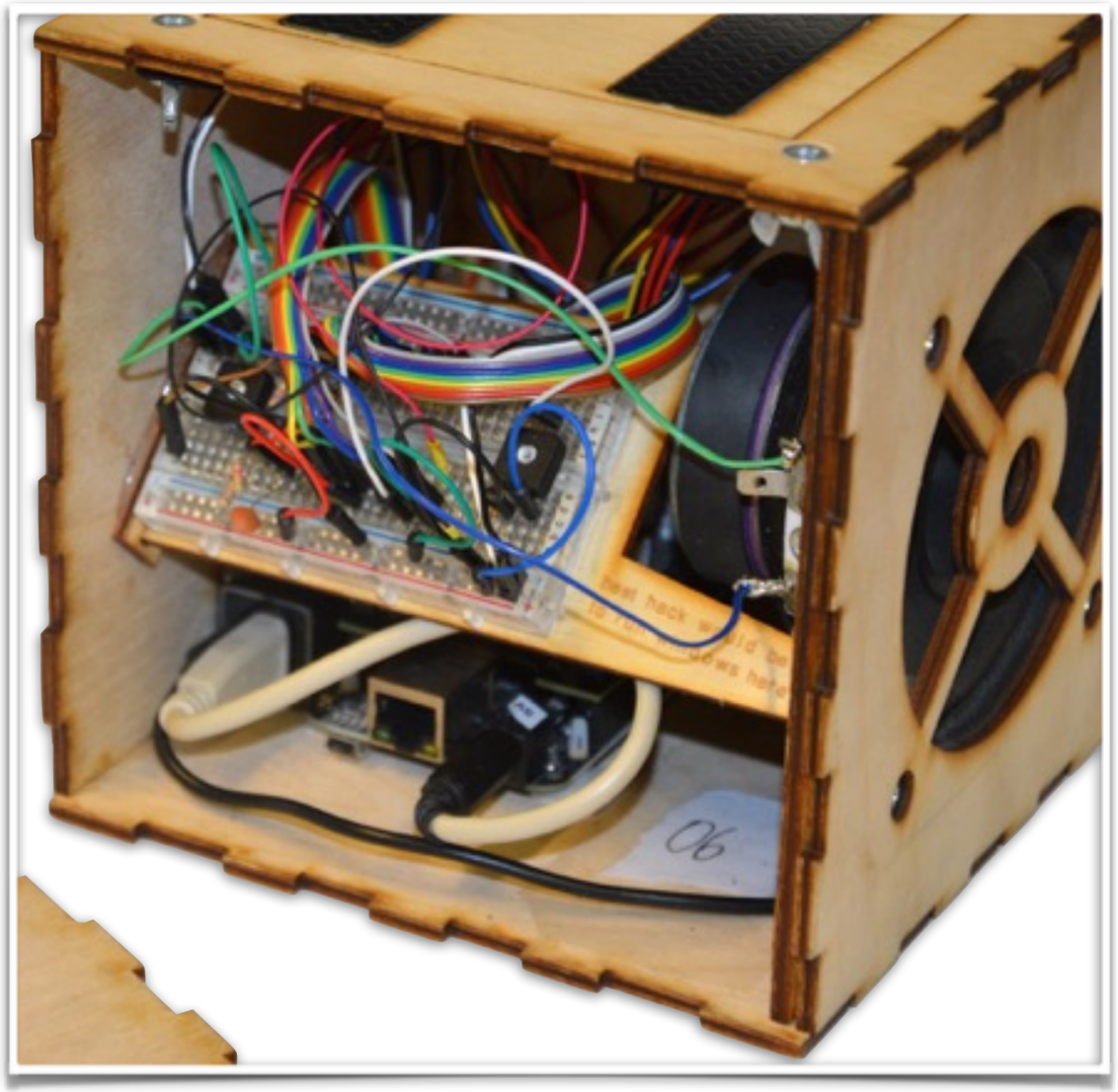
Origins

The **D-Box**: a hackable musical instrument



Origins

The **D-Box**: a hackable musical instrument

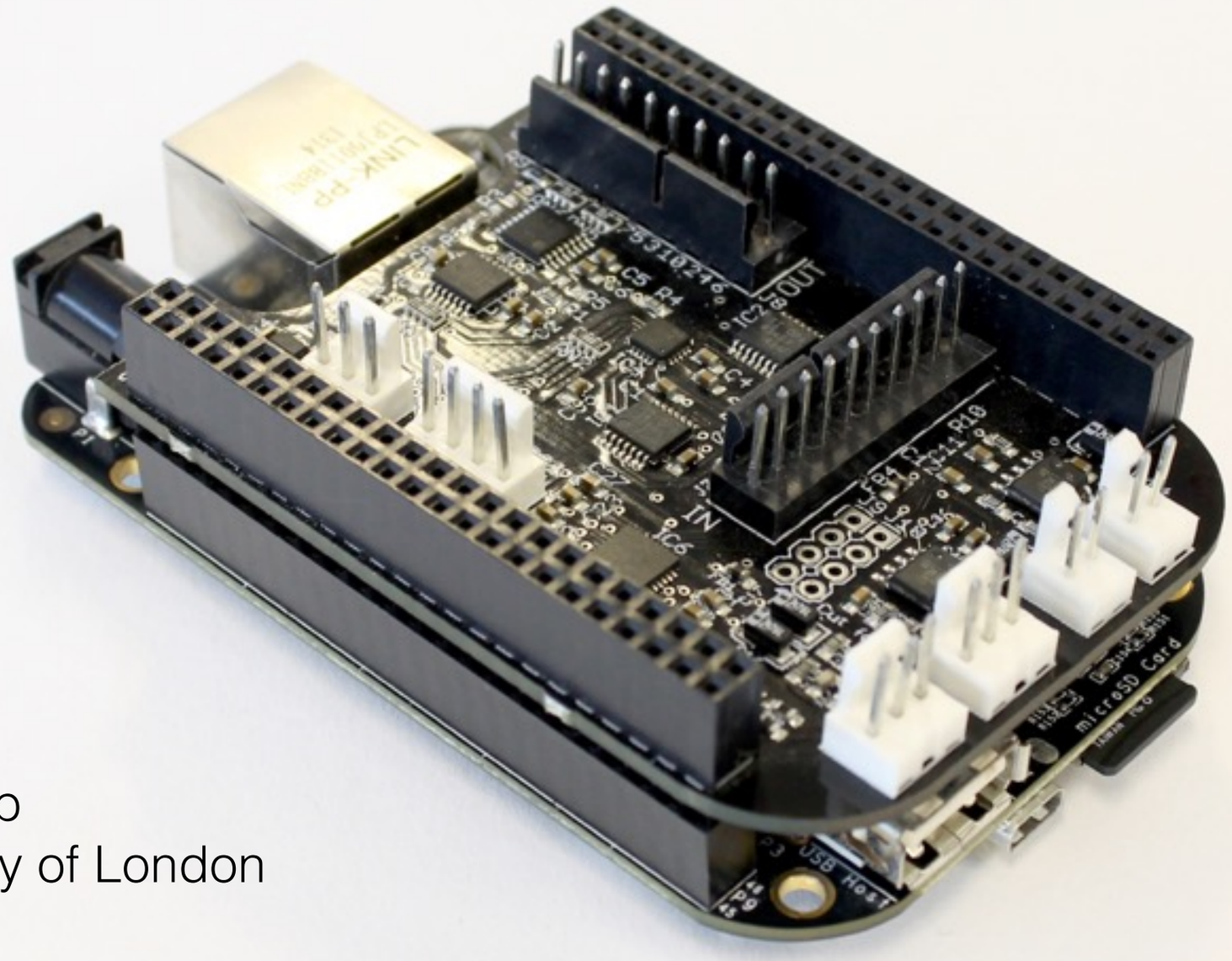


Requirements

- multiple I/O
- low-latency feedback
- speaker amp
- connectivity (I2C, GPIO)

bela

Ultra-low latency audio and sensor processing on the BeagleBone Black



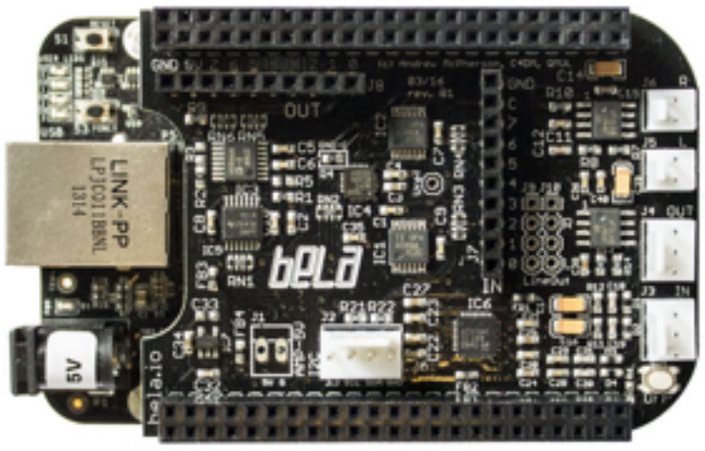
A project by
The Augmented Instruments Lab
at C4DM, Queen Mary University of London

http://bela.io

EPSRC
centre for digital music
Queen Mary University of London

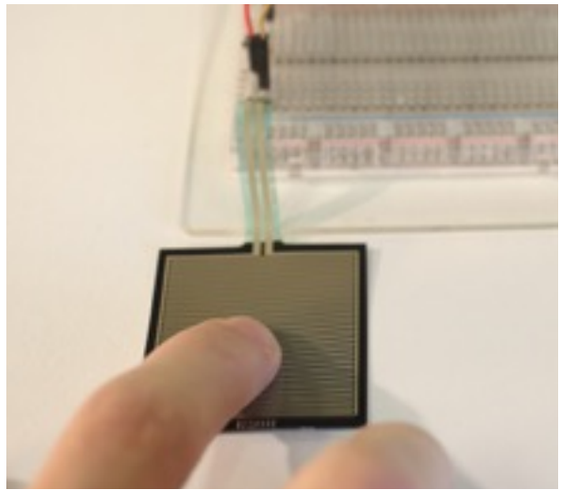
What is Bela?

Robust, high performance audio environment



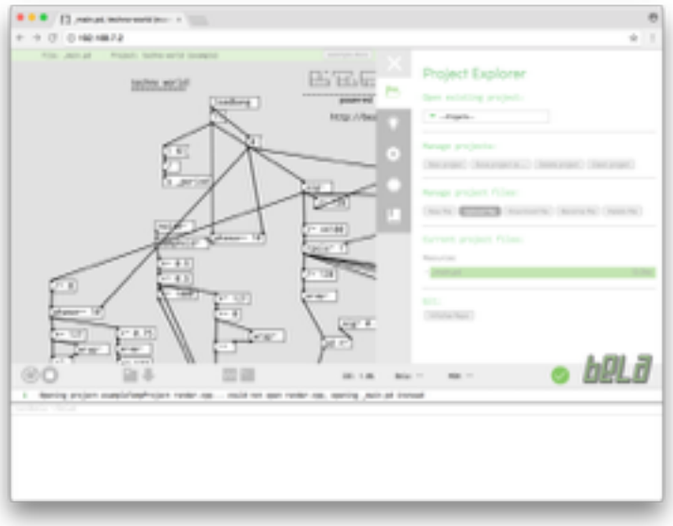
- Ultra low latency (1ms) round-trip
- Hard real time
- Combine benefits of Linux and custom DSP hardware

New approach to high-bandwidth sensor processing



- Analog, digital I/O sampled at audio rate
- Jitter-free alignment between audio and sensors
- Easily apply audio DSP structures to sensors

Open-source maker platform



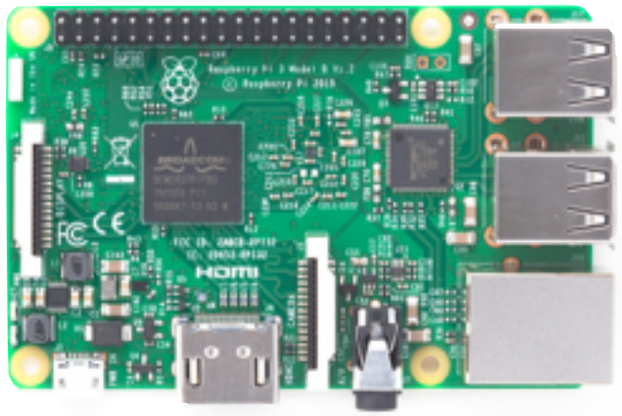
- Open hardware and software
- Targeted at musicians, artists
- Online community resources: forum, wiki, blog.

The goal

High-performance, self-contained audio and sensor processing

ARDUINO

- Easy low-level hardware connectivity
- No OS = precise control of timing
- Very limited CPU (8-bit, 16MHz)
- Not good for audio processing

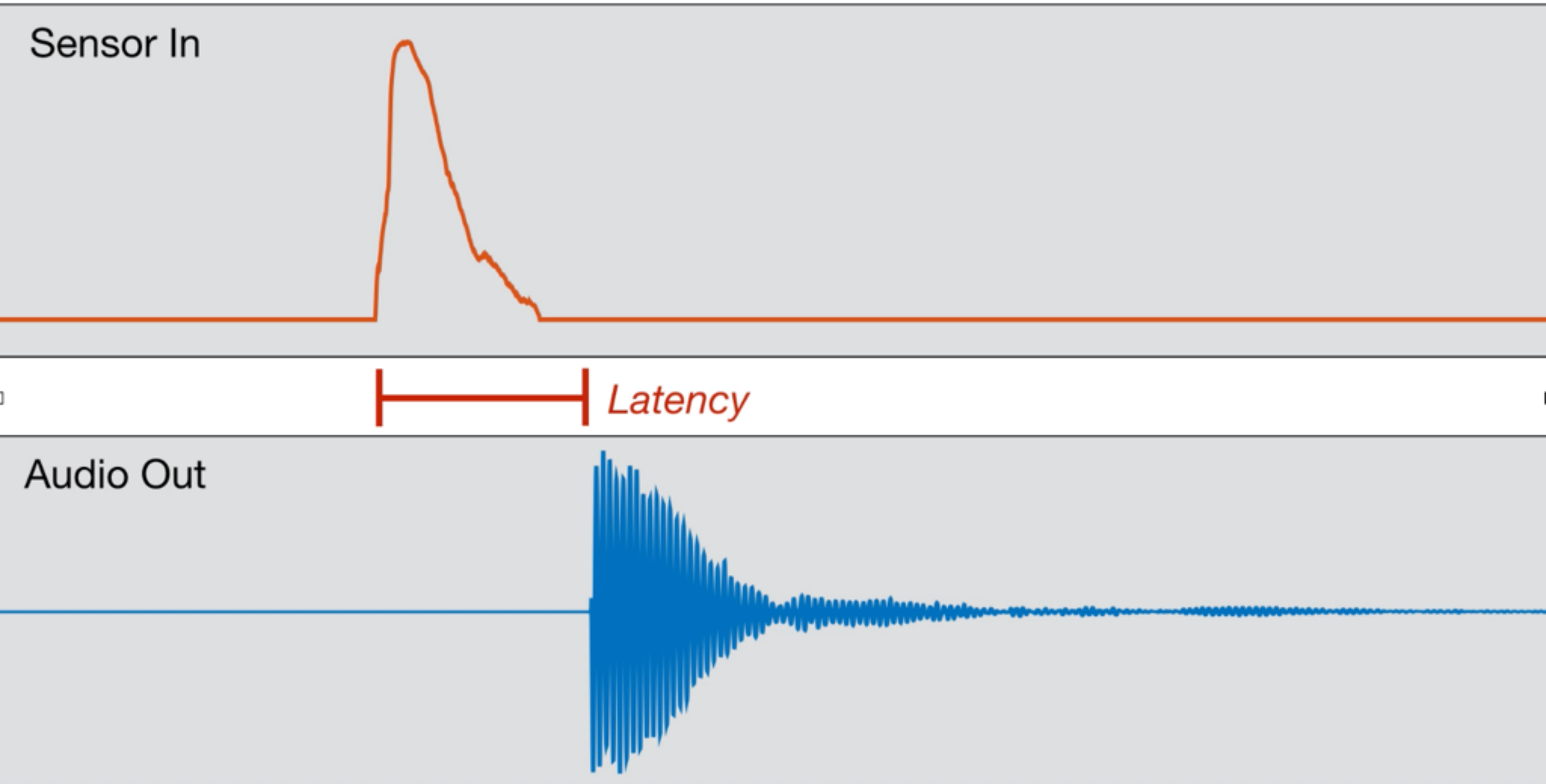


- Reasonable CPU (up to 1.2GHz ARM)
- High-level hardware (USB, network etc.)
- Limited low-level hardware control
- Linux OS = high-latency / underruns

- Fast CPU
- High-level hardware (USB, network etc.)
- Arduino for low-level
- USB connection = high-latency, jitter
- Bulky, not self-contained

Latency

Delay between action and reaction,
a fundamental property of digital interactive systems



Audio Latency

Does it matter? Yes!

- Long latency causes an audible delay
  - But even an imperceptible delay may make an instrument feel less responsive to play
- How low is “low enough”?

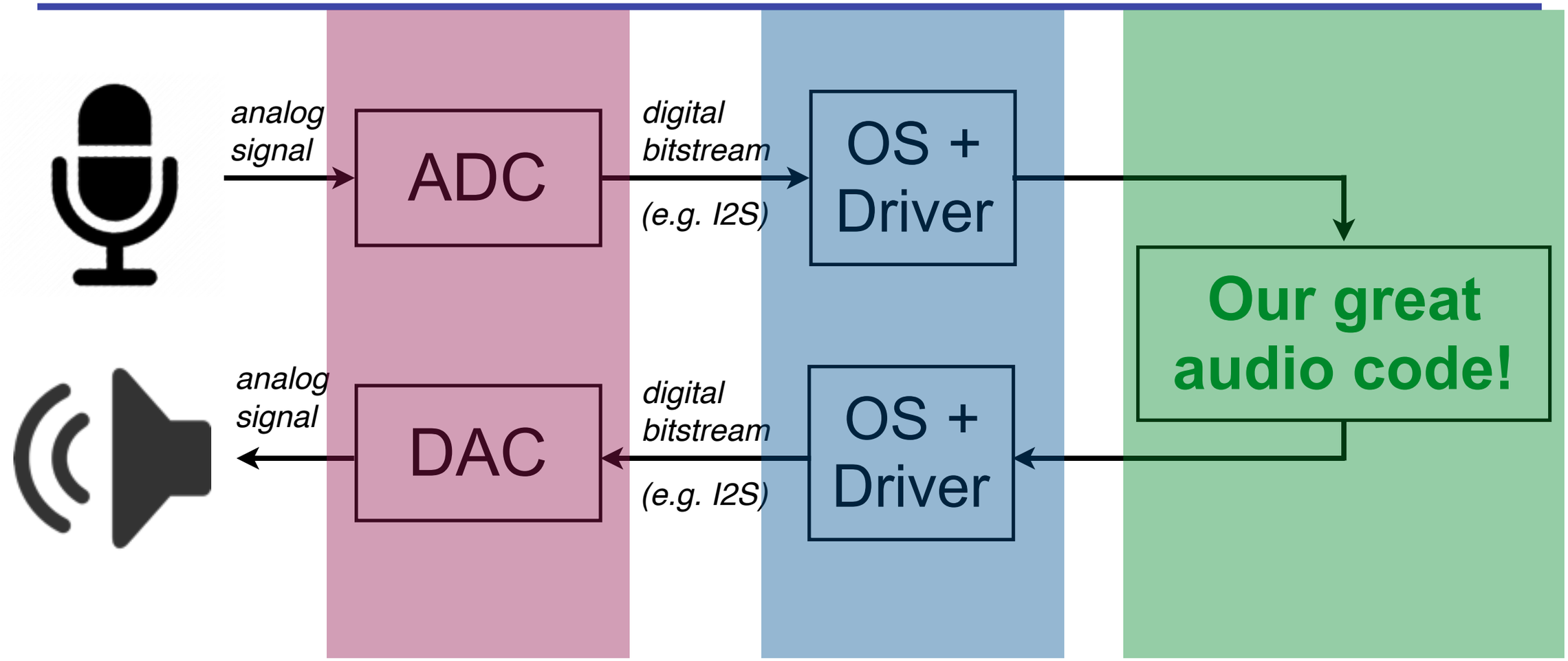
  “We place the acceptable upper bound on the computer’s audible reaction to gesture at 10 milliseconds. ... Low variation of latency is critical and we argue that the range of variation should not exceed 1 ms.”

  -- David Wessel & Matthew Wright, “Problems and Prospects for Intimate Musical Control of Computers”, Computer Music Journal, **2002**.

- Surprisingly few systems for building digital musical instruments meet this threshold!

  A. McPherson, R. Jack and G. Moro. “Action-Sound Latency: Are Our Tools Fast Enough?” Proc. NIME, 2016.

Round-trip audio latency



Conversion latency

- Inherent to sigma-delta codec hardware
- Up to **500µs** each way
- May be extra hardware latency for USB audio

Buffering latency

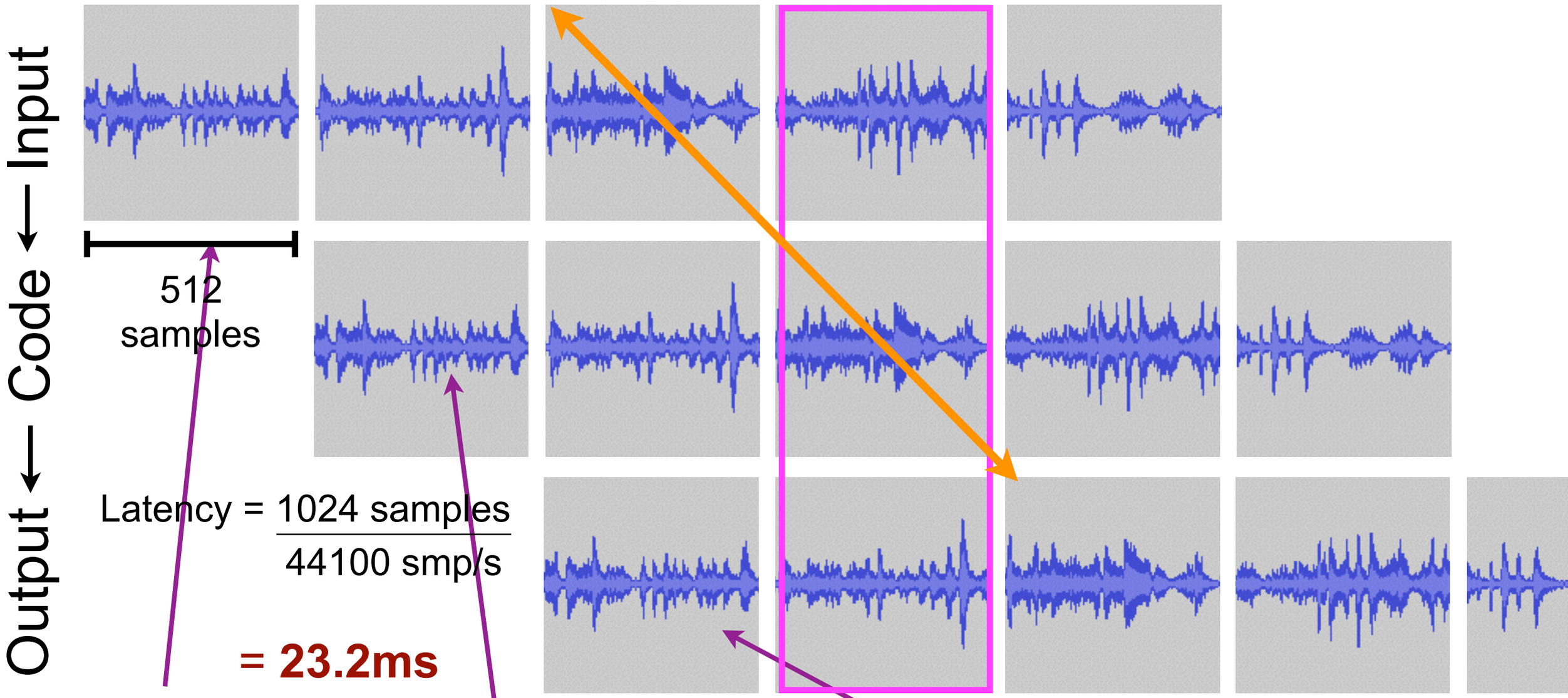
- Function of the OS audio architecture
- Depends on the audio block size
- Anywhere from **1-20ms** (or more!) each way

Group delay

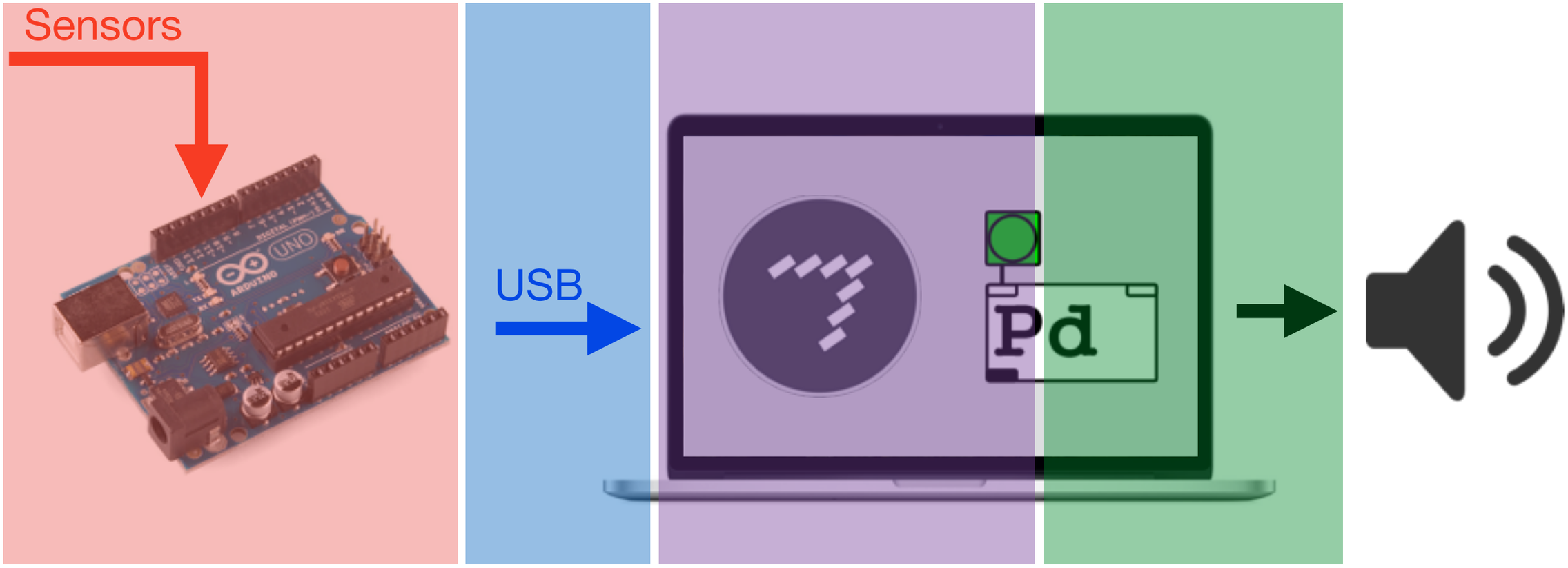
- Mathematical latency inherent in audio code
- Biggest offender: block-based processing (FFT)

Round-trip audio latency

At any given time, we are reading from ADC, processing a block, and writing to DAC



Interaction latency



Sampling latency

- Depends on sample rate of sensors
- Commonly **1-100ms**

OS driver latency

- Serial drivers optimised for data integrity, not speed
- **Unpredictable** latency, usually fast, occasionally very slow!

Communication latency

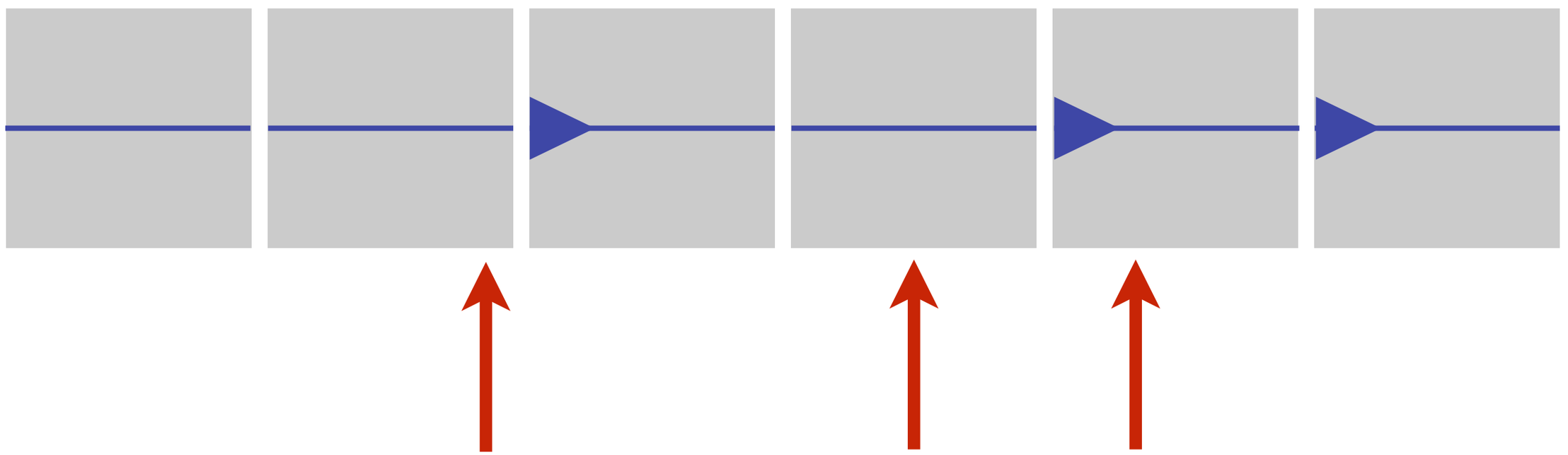
- USB has a **1ms** frame clock
- Hardware serial (e.g. on Uno) is slow: 9600bps ~= **1ms per byte**
- Hardware MIDI (31250bps): about **1ms per 3-byte message**

Audio latency

- As before: code, OS, converters

Jitter

- Round-trip audio latency is usually constant
- Action-to-sound latency can vary randomly
  - Audio and sensors are handled on different threads



- Sensor data processed at block intervals
  - Larger audio block sizes = more jitter

Bela - Design goals

- Minimize roundtrip latency of audio
- Minimize jitter for interaction

Requirements from the D-box:

- multiple I/O
- low-latency feedback
- speaker amp
- connectivity (I2C, GPIO)

Bela hardware

8x, 16-bit Analog Out

16 digital GPIO

8x, 16-bit Analog In

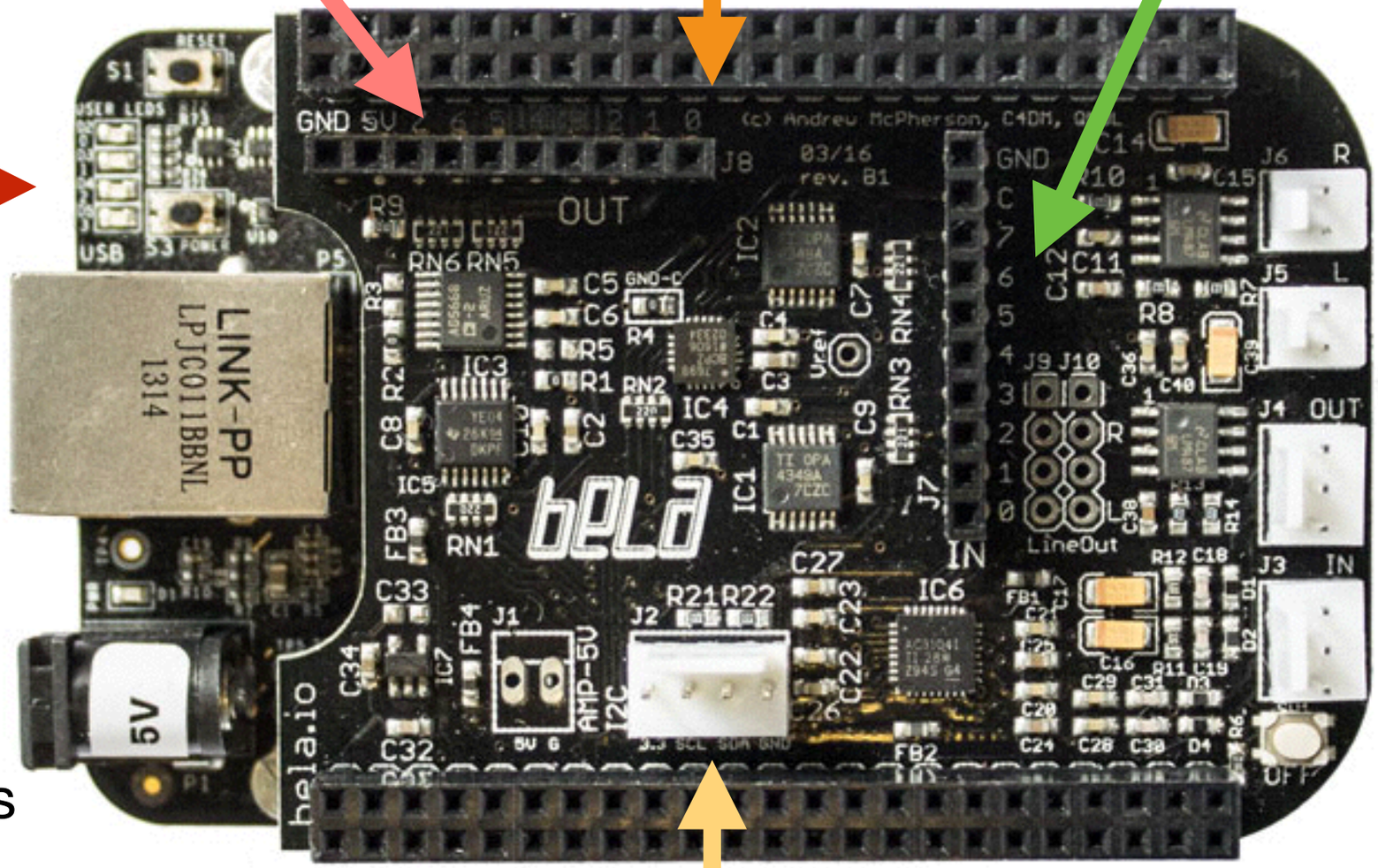
BeagleBone Black

1GHz ARM Cortex-A8

512MB RAM

NEON vector floating point

Dual PRU microcontrollers



Stereo 1.1W speaker amps

Stereo 16-bit audio out

Stereo 16-bit audio in

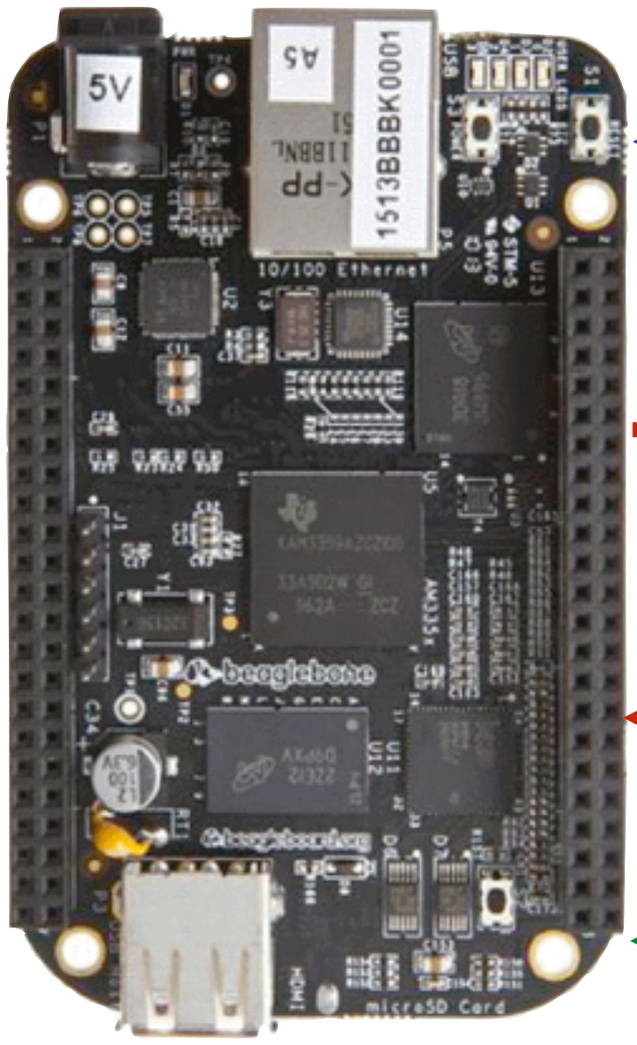
I2C

BeagleBone Black

- Texas Instruments AM3358, which features:
  - 1GHz single-core Cortex A8
  - 2x Programmable Realtime Units
  - 4x 32-bit GPIO banks
  - McASP (multi-channel audio serial port)
  - I2C
  - SPI
  - Serial
  - CAN
  - Quadrature encoder
  - PWM
  - 12-bit analog inputs
  - ....
  - A 4000+ page Technical Reference manual

Hardware architecture

Main real-time I/Os on Bela cape:



I2S (McASP)

Audio Codec
TLV320AIC3104

stereo audio I/O

SPI

16-bit DAC
AD5668

8 analog outs (0-5V)

SPI

16-bit ADC
AD7699

8 analog ins (0-4.096V)

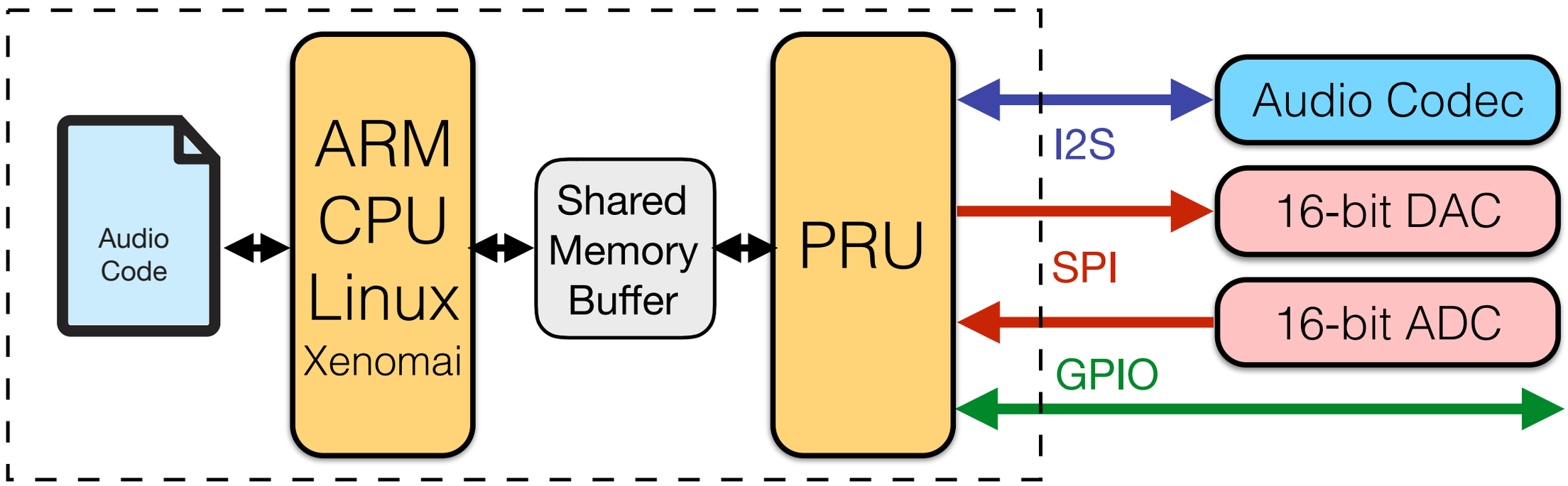
GPIO

16 digital I/Os (3.3V)

Can’t use the kernel drivers for this hardware: too slow!

System architecture

- A sophisticated DMA controller
  - audio, analog, gpio data in a single, time-aligned (jitter-free) stream
- A "responsive" operating system
  - thread needs to run as often as every 45us



PRU

- Programmable Real-time Units
  - Feature of Texas Instruments Sitara SoCs
  - Optimised specifically for timing-sensitive tasks
- Specifications:
  - 200MHz
  - Pipeline-free architecture, with most instructions executing in a single 5ns cycle
  - Full access to SoC memory and peripherals (e.g. McASP, SPI, GPIO), independent of the CPU
  - Dedicated memory plus shared system memory
  - Interrupts to and from CPU and peripherals
- Code on PRU does not need an OS
  - Highly predictable performance

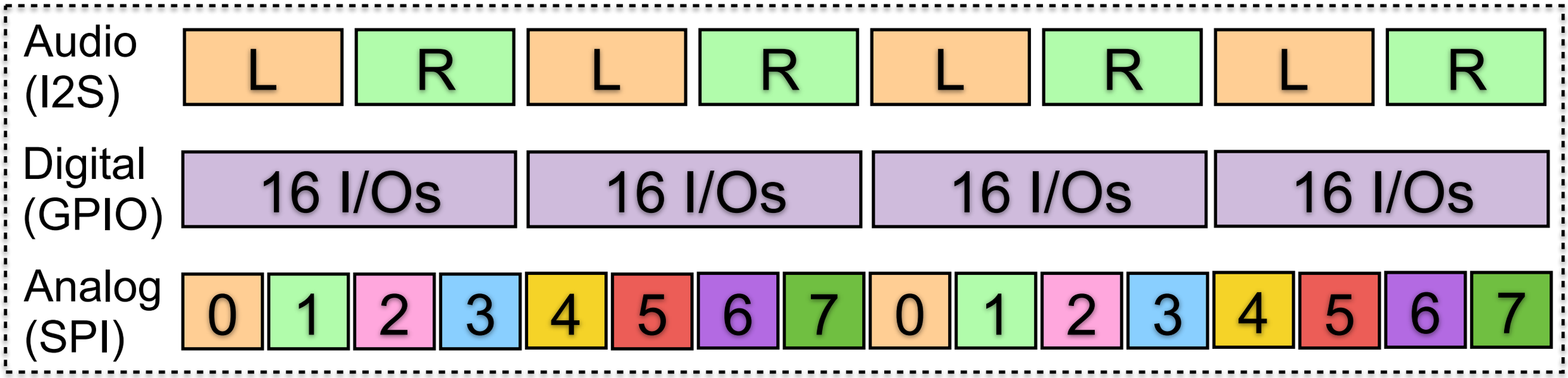
PRU - A sophisticated DMA

PRU I/O "DMA" loop (pseudo-pseudo code):

- while(!gShouldStop)
  - for (blocksize * 2)
    - wait for McASP timer (~11us, 88200Hz)
    - read audio input (L or R)
    - write audio output (L or R)
    - read SPI ADC sample (even)
    - write SPI DAC sample (even)
    - read SPI ADC sample (odd)
    - write SPI DAC sample (odd)
    - do GPIO
  - swap buffers, signal ARM

Synchronous audio and sensors

- On Bela, PRU samples audio, analog and digital I/O synchronously (aligned by sample)



  - Result: we can associate every sensor reading with a specific audio sample
  - No jitter between audio and sensors
  - CPU system time no longer needed: every sensor is identified by when it is sampled, not when it is processed

a real-time audio principle:

Consider the worst-case performance

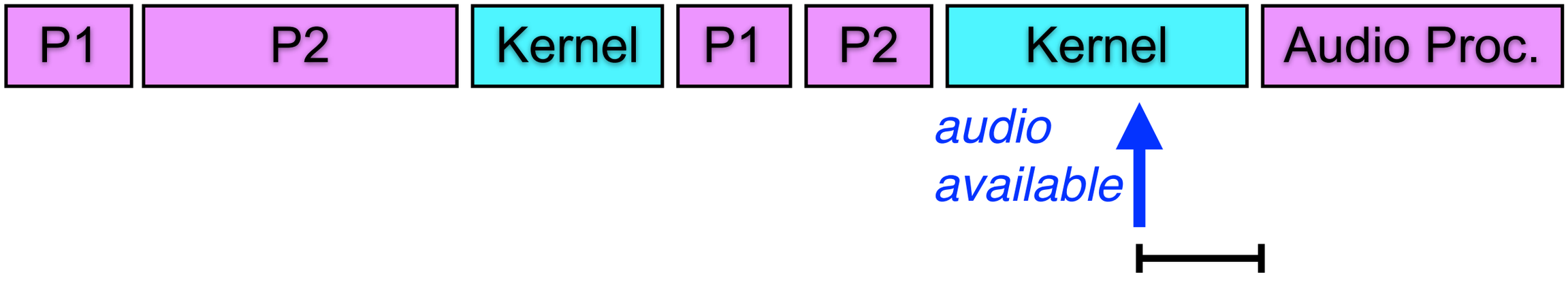
In live performance, glitches are unacceptable

↓

Meeting 99% of deadlines isn’t good enough!

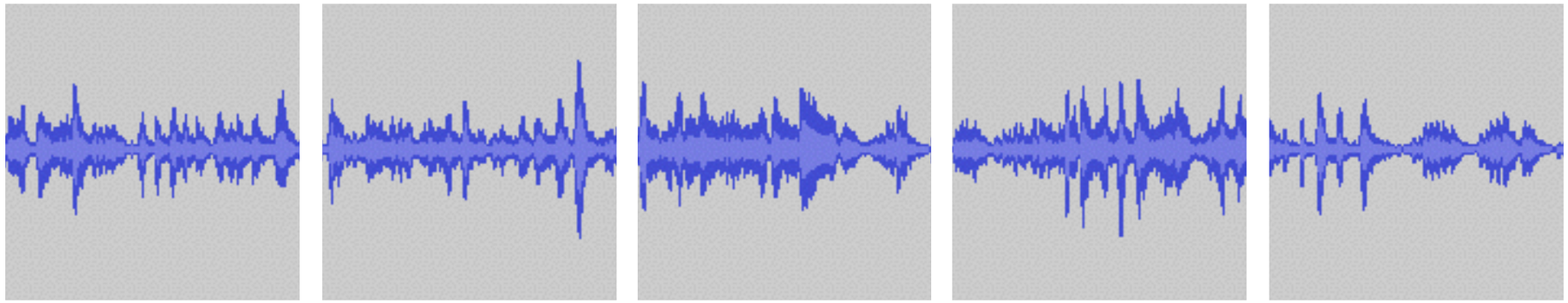
Real-time performance

On Linux, audio code competes for time with other processes and threads:



Implications

- Given an unknown reaction time to audio:



  - The longer the worst-case delay, the larger the block size we need to use to avoid underruns
  - Useful metric: reaction delay as fraction of the audio block period
  - More CPU-intensive audio code reduces the margin

Embedded hardware for audio

- limited processing power: if you want more power you need an operating system
- fight against its scheduler competing with other processes
- latency of waking up the process
- user expectations: user expects no-latency or glitches because "it's hardware! it's not a computer"

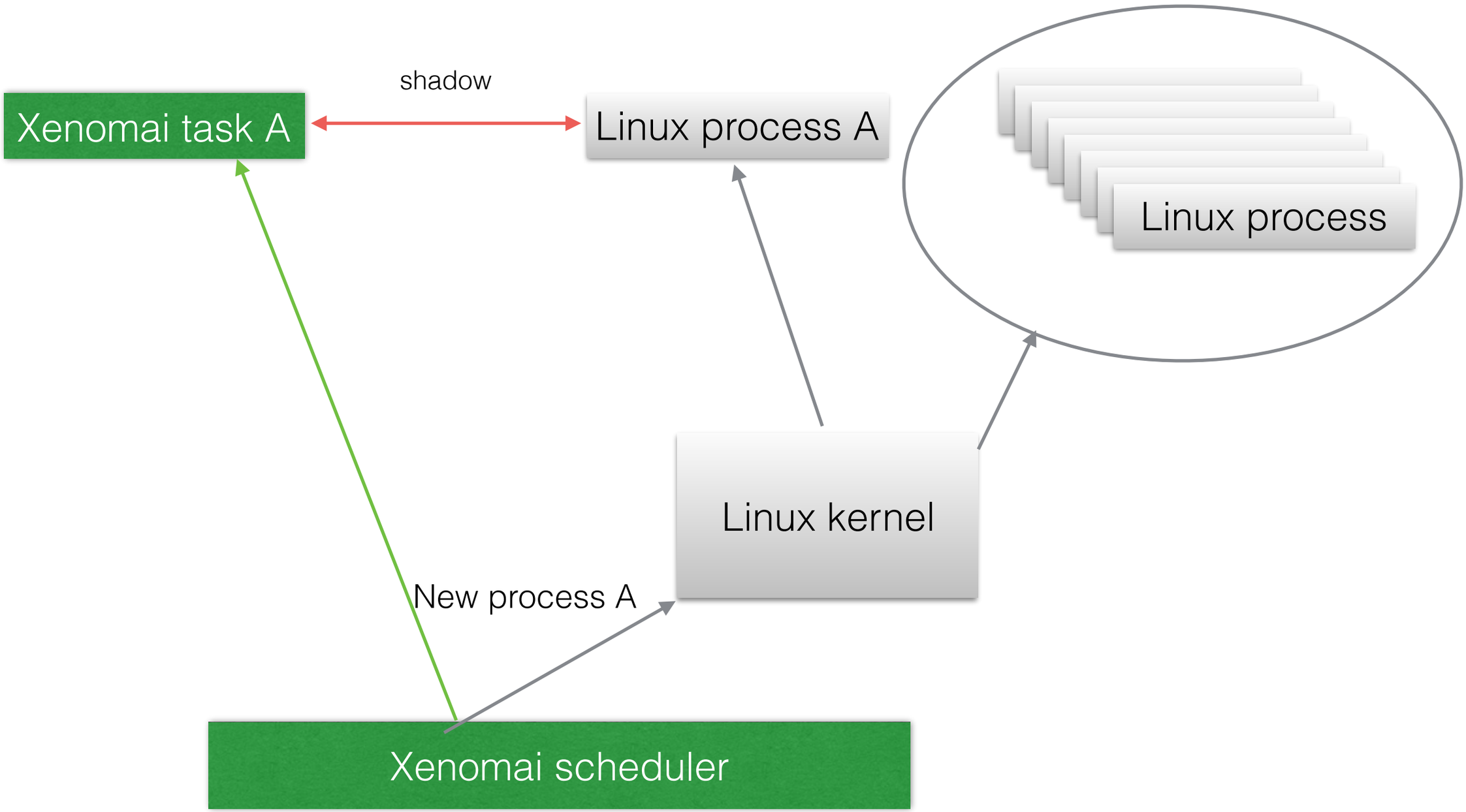
Xenomai

- Dual kernel: RTOS kernel running a "Linux thread" and other threads
- Comes in three parts:
  - i-pipe patch
  - kernel driver
  - user-space API

- Thread modes:
  - "Primary mode" or "Xenomai mode": governed by the Xenomai scheduler
  - "Secondary mode": regular Linux mode

Xenomai

- "Primary mode" or "Xenomai mode": governed by the Xenomai scheduler
- "Secondary mode": regular Linux mode

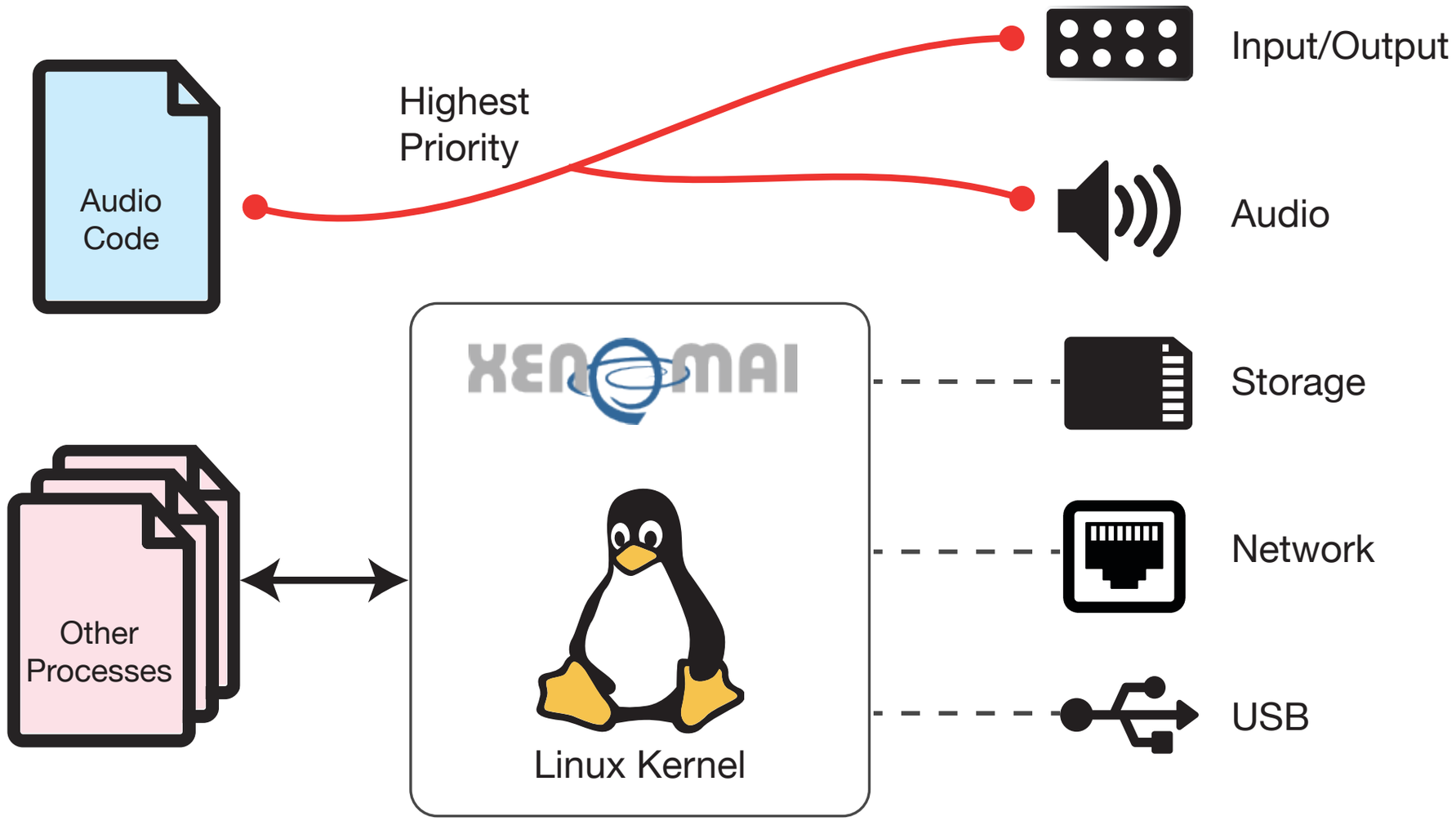


Xenomai - Mode switches

- When a Linux system call takes place, the Xenomai thread switches to Linux mode
- Linux mode needs to be avoided in the audio thread:
  - disk I/O
  - socket
  - printf
  - memory allocation
  - thread, synchronization, timing, messaging services...
    - std::thread
    - std::mutex
    - std::chrono
    - ...

Xenomai and Linux

- Xenomai allows real-time tasks and Linux threads to run side-by-side (even within the same process)
- Tasks can switch transparently from primary to secondary mode, dropping real-time privileges, when a system call is invoked. (“mode switches”)



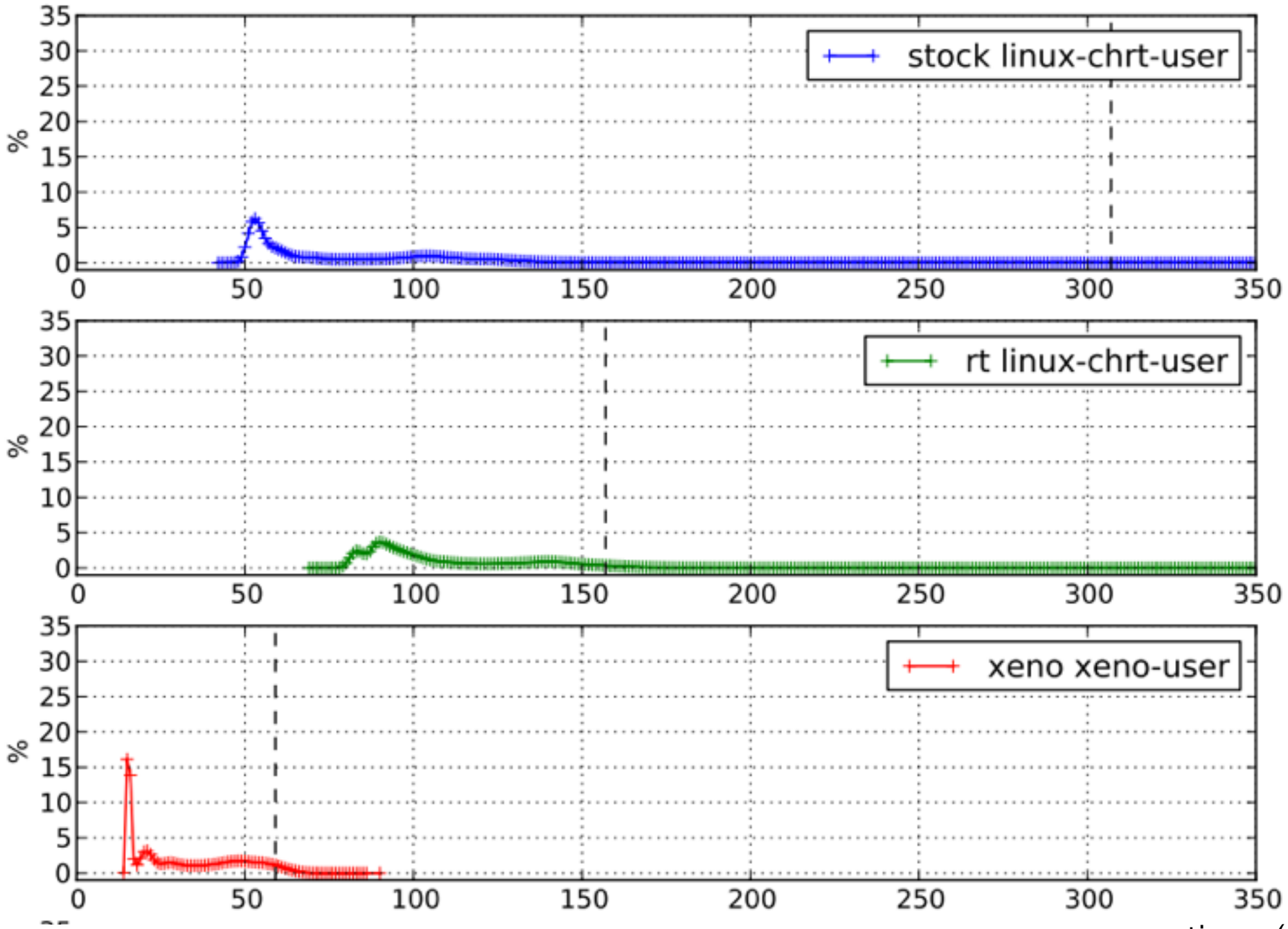
Xenomai - scheduling

- Xenomai: Preemptive Priority-Based Scheduling

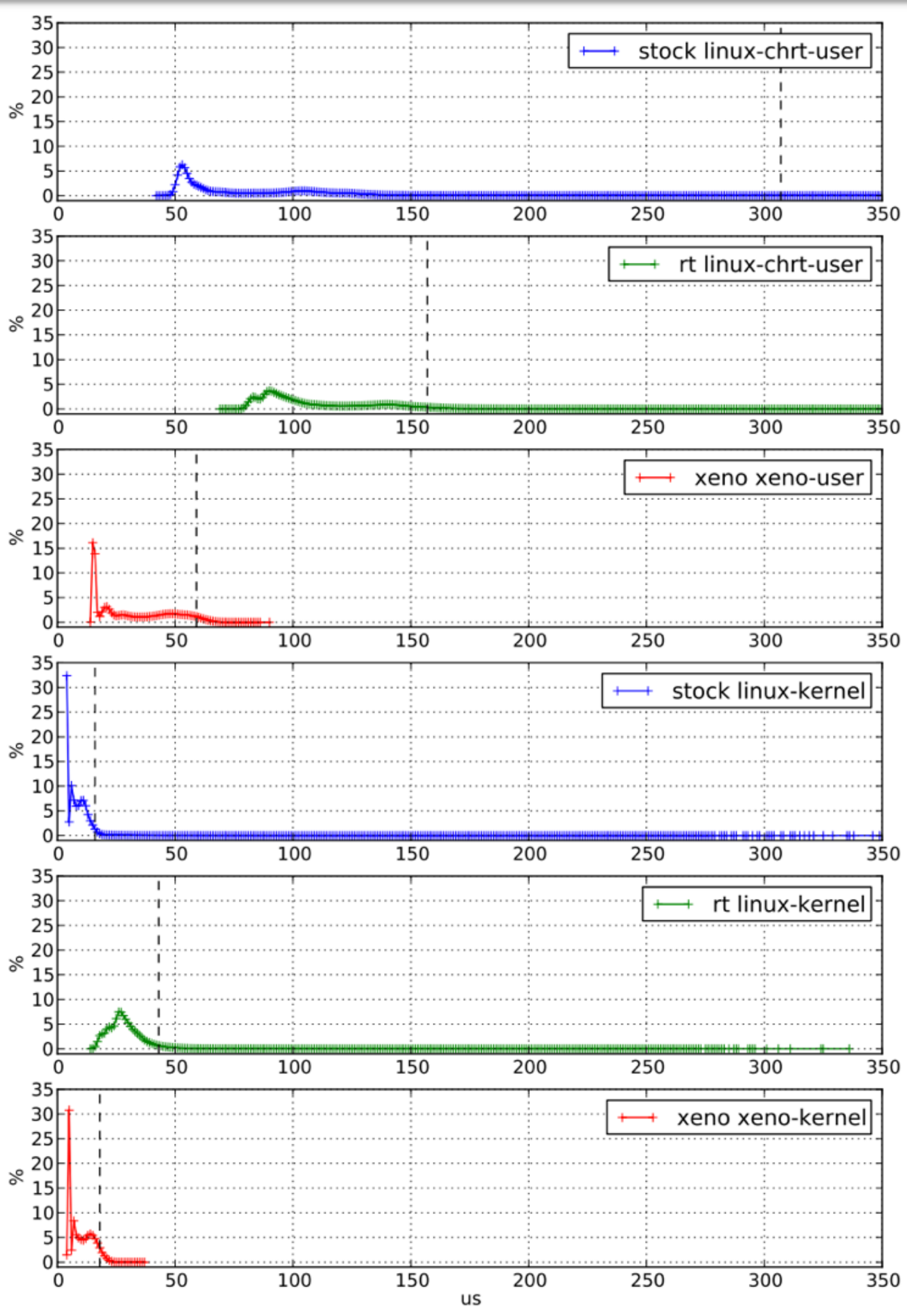
  - Highest priority thread runs first
    - multiple real-time tasks, with assignable priority
    - Linux has priority 0
  - Load of lower priority threads or Linux threads does not affect RT performance
  - except for CPU cache and wakeup latency

Responsiveness

Brown, Jeremy H., and Brad Martin. "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications." proc. of the 12th Real-Time Linux Workshop (RTLWS'12). 2010.



- Thread wake-up latency
- Linux vs RT-linux vs Xenomai, GPIO in to GPIO out



Responsiveness

Brown, Jeremy H., and Brad Martin. "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications." proc. of the 12th Real-Time Linux Workshop (RTLWS'12). 2010.

Performance

- How much CPU can we use?
  - Xenomai will SIGXCPU if you go 100%
  - > 97% is still "*fine*"
  - No glitches till you reach the ceiling

```
Every 0.5s: cat /proc/xenomai/stat                          Sun Oct 30 01:03:47 2016

CPU  PID    MSW        CSW        PF    STAT       %CPU  NAME
  0  0      0          11813352   0     00500080    2.0  ROOT
  0  16324  2          2          0     00b00380    0.0  exampleTempProj
  0  16326  1          2          0     00300380    0.0  bela-midiIn_hw:1,0,0
  0  16327  0          8453734    0     00300186   97.1  bela-audio
  0  0      0          8457781    0     00000000    0.6  IRQ21: bela-pru-irq
  0  0      0          2517665    0     00000000    0.1  IRQ67: [timer]
```

Xenomai - Posix skin

- You can write regular posix C code
  - clock_...
  - timer_...
  - pthread_...
  - pthread_mutex_...
  - pthread_cond_...
  - sem_...
- then passing appropriate linker flag, e.g.:
  - `-Wl,-wrap,pthread_create`
  - `-wrap` resolves calls to `pthread_create` to `__wrap_pthread_create` and calls to `__real_pthread_create` to `pthread_create`
  - automatically gives you a Xenomai thread

Xenomai vs PREEMPT_RT

- Xenomai responds faster, especially in user space
- Linux performance is better with Xenomai
- Mode switches help with debugging
- PREEMPT_RT is closer to mainline Linux
- Xenomai 3 allows to use either

Low-latency, embedded, RT audio

Target:

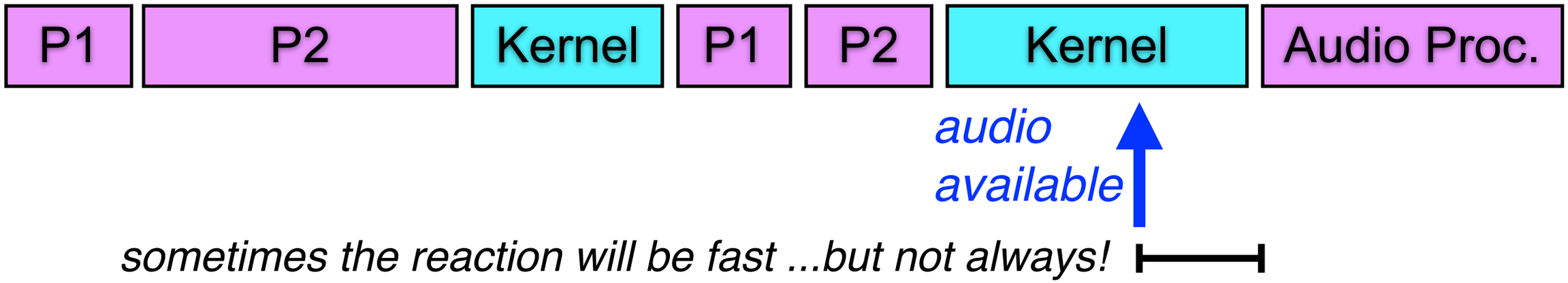
- real-time safe audio callback:
  - nothing that could block

- constant CPU load:
  - no occasional, expensive, computations, use threads

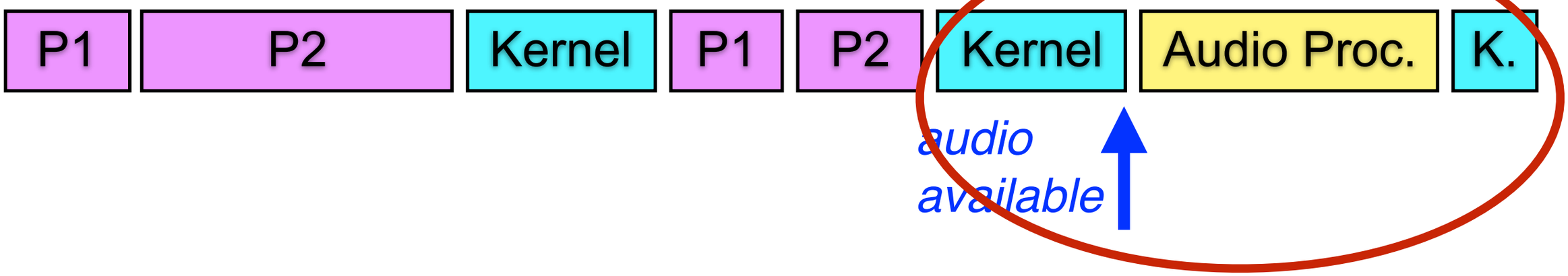
- No "Linux" mutex, clocks, synchronization, queues

Real-time performance with Xenomai

On Linux, audio code competes for time with other processes and threads:



Xenomai tasks have guaranteed priority over everything else on the board, **including the Linux kernel:**



These hard real-time tasks run in what's called "**primary mode**" in Xenomai. Standard Linux threads run in "secondary mode".

Real-time performance

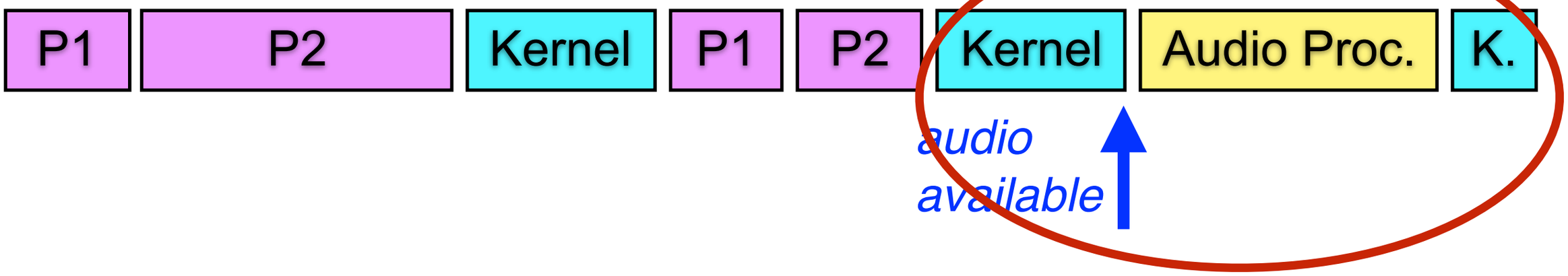
Why doesn't this break the kernel?

Like on a virtual machine, the Linux kernel is not aware of being paused by the real-time audio code.

Nothing in the audio code can change the kernel state.

No system calls in the audio callback!

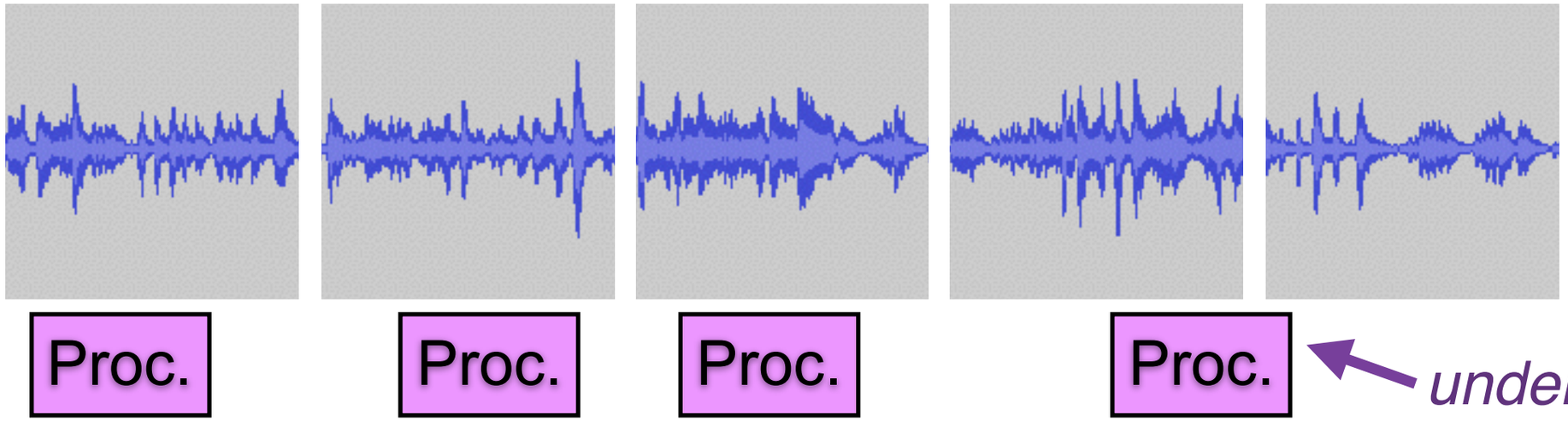
Xenomai tasks have guaranteed priority over everything else on the board, **including the Linux kernel:**



These hard real-time tasks run in what's called "**primary mode**" in Xenomai. Standard Linux threads run in "secondary mode".

Without Xenomai

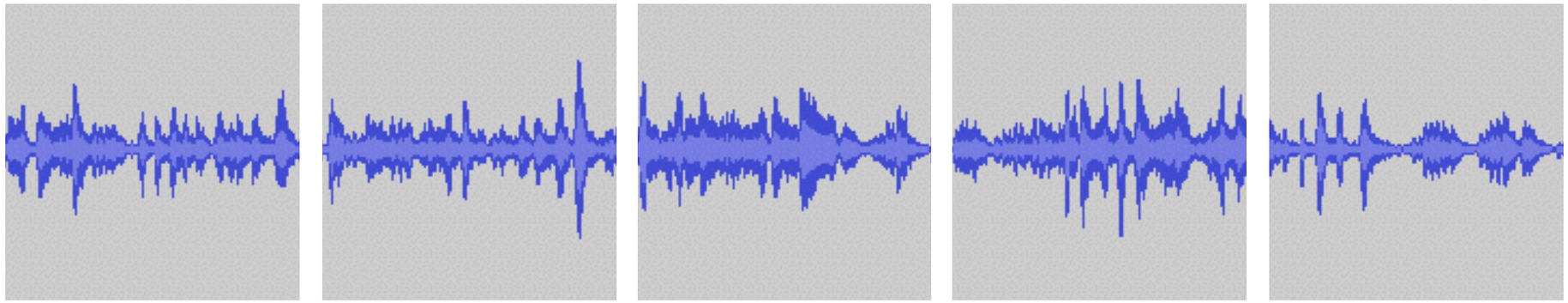
- Given an unknown reaction time to audio:



  - The longer the worst-case delay, the larger the block size we need to use to avoid underruns
  - Useful metric: reaction delay as fraction of the audio block period
  - More CPU-intensive audio code reduces the margin

With Xenomai

- We can also use more of the CPU:



- This is independent of other system load
  - Audio unaffected by other processes or threads
  - The reverse isn't true, of course! But in an embedded audio device, nothing else has the same urgency.
  - We can achieve > 97% CPU usage with no artefacts

Why again?

- Why not Linux and ALSA?
  - multimodal stream of data (audio, analog, digital) not handled by ALSA
  - Xenomai primary mode code (i.e. Bela audio callback) cannot depend on a Linux kernel driver
  - Linux kernel cannot provide sufficient timing guarantees
- Xenomai hardware drivers?
  - Possible to write Xenomai kernel drivers (instead of Linux kernel) for hardware peripherals
  - But Xenomai cannot efficiently respond to every individual audio sample
  - Synchronising multiple data sources would be difficult without dedicated core (PRU or ARM)
- PRU integrates audio and sensors on an individual sample level
  - Synchronised at time of capture

Three layers of real time

- Linux: 1 (large) audio block
  - Reaction time usually fast, but not reliably so
  - Good for tasks running every few milliseconds
  - Timing performance depends on system load
- Xenomai: 1 (small) audio block
  - Reaction in tens of microseconds reliably
  - For efficiency, best for tasks running every 100+ µs
  - Timing performance independent of other load
- PRU: multiple events per audio sample
  - Timing precision of under 1µs, limited primarily by memory and bus activity
  - Ideal for synchronising multiple data sources and gathering them for later processing

Conclusion: benefits of Bela

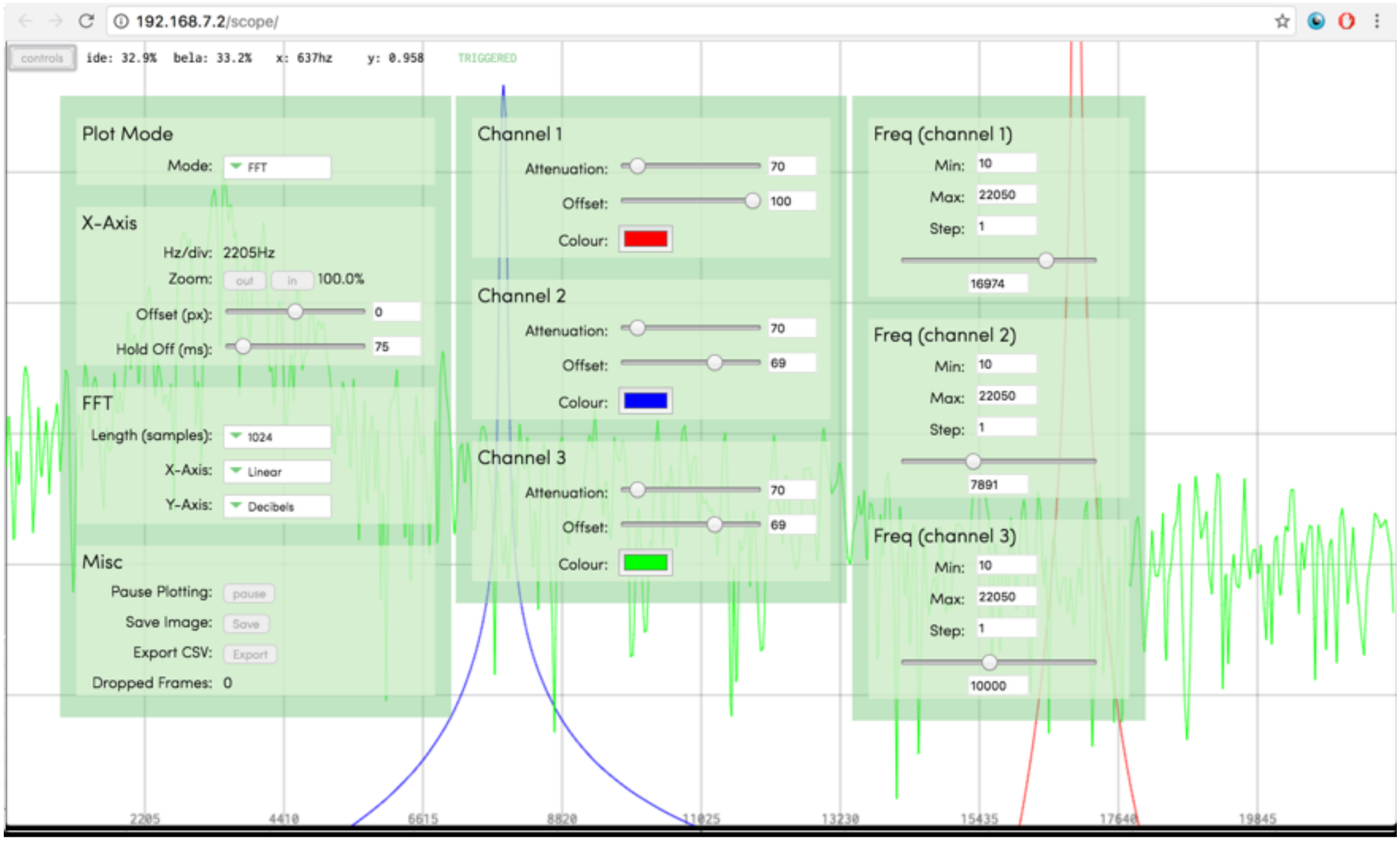
1. Ultra-low latency audio processing
   - Capable of 1ms round-trip audio latency
   - 100µs round-trip using analog/digital I/O
2. Hard real-time performance
   - Insensitive to other system load
   - Can use nearly all of CPU with no underruns
3. Sample-accurate audio-sensor alignment
   - Extremely low jitter between action and sound
4. High sensor bandwidth
   - Sensors automatically sampled at audio rates
5. Convenience and connectivity of Linux
   - Faster development time, better future-proofing

bela

the maker platform

Bela features

Built-in, browser-based IDE with oscilloscope



Bela features

- Programming options:
  - **C/C++** with built-in IDE (or terminal scripts)
  - **Pd** using either libpd or Heavy Audio Tools
  - **FAUST** DSP language (contributed by Stéphane Letz and Yann Orlarey)
  - **Pyo** python audio module (contributed by Olivier Bélanger)
  - **SuperCollider** (collaboration with Dan Stowell, Marije Baalman)
- Open-source hardware and software
- Online repo, wiki and support forum:
  - http://github.com/BelaPlatform
  - http://forum.bela.io

Pd on Bela

Two options for running Pd patches on Bela:

libpd

- Library which runs on Bela interpreting Pd files
- As (in)efficient as Pd itself
- Supports other platforms: ofxpd, webpd, ...
- No network required, instantaneous update
- Fully open source

heavy audio tools

- Cloud-based compiler generating optimised C code for Bela
- Most efficient performance on Bela
- Supports other platforms: JS, VST2, Unity
- Requires net connection and local compiling
- Closed but free to use

Kickstarter
April 2016

Raised £55k
(1100% of goal)
from 530 backers

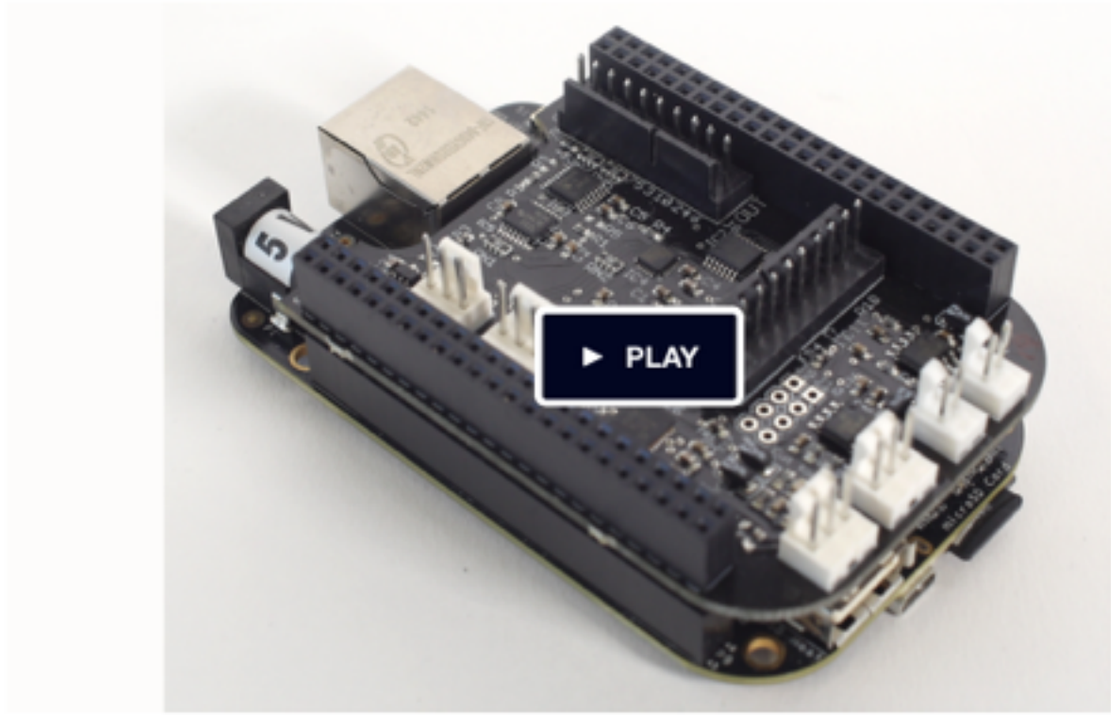
Shipped
summer 2016

530 backers pledged £54,902 to help bring this project to life.

Campaign | Updates 10 | Comments 11 | Community

Share this project

About this project



London, UK | Sound | Project We Love

£54,902
pledged of £5,000 goal

As of 15 March we reached both of our stretch goals! You can order an Audio Expander Capelet which converts Bela's analogue ins and outs into extra audio channels, or a Multiplexer Capelet which gives you 64 analogue inputs. See "Rewards" below for details on ordering.

What is Bela?

Bela is an embedded computing platform developed for high quality, ultra-low latency interactive audio. Bela provides **stereo audio, analogue and digital I/O** in a single self-contained package.

Rewards

Pledge £1 or more

Bela supporter. Stay in the loop on the latest updates, and we will add your name to our supporters page.

ESTIMATED DELIVERY:
Apr 2016

24 backers

Backer report

NO SURVEYS SENT

Pledge £20 or more

Bela T-shirt. Get a black cotton T-shirt with the Bela logo. (Add £20 to any other pledge to get a T-shirt too.)

ESTIMATED DELIVERY:
Apr 2016

SHIPS TO:
Anywhere in the world

7 backers

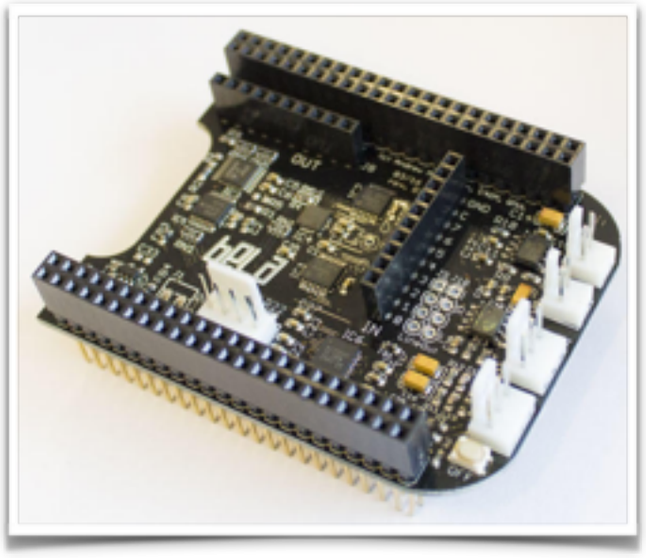
Backer report

SURVEY SENT:
4/27/2016 · 6 responses

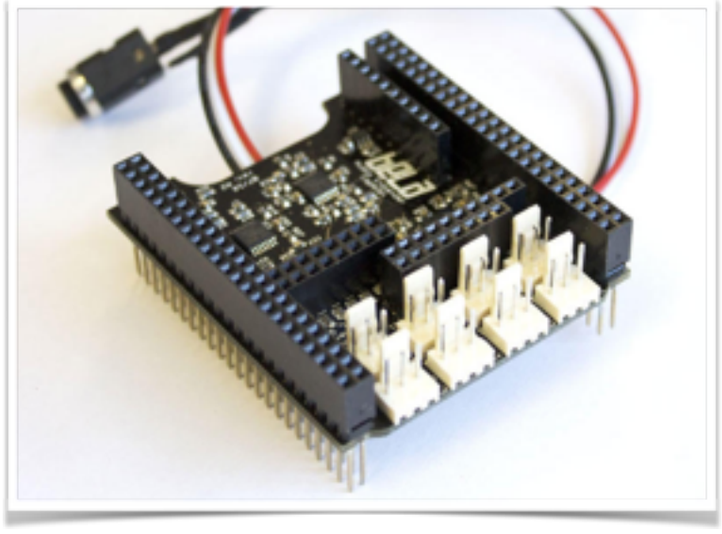
Post-Kickstarter

- Public relaunch October 2016

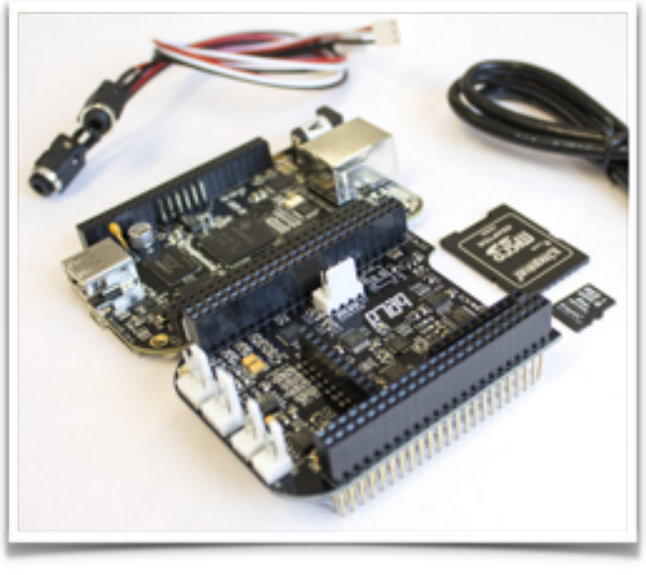
https://shop.bela.io



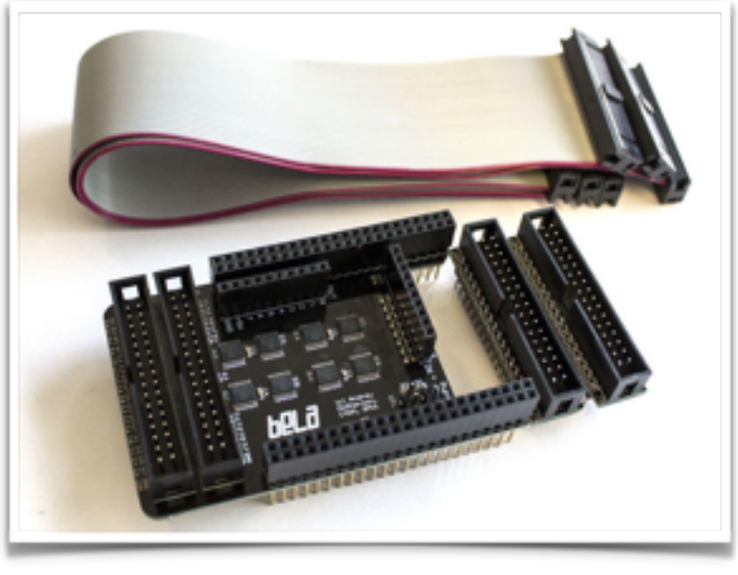
cape



audio expander capelet



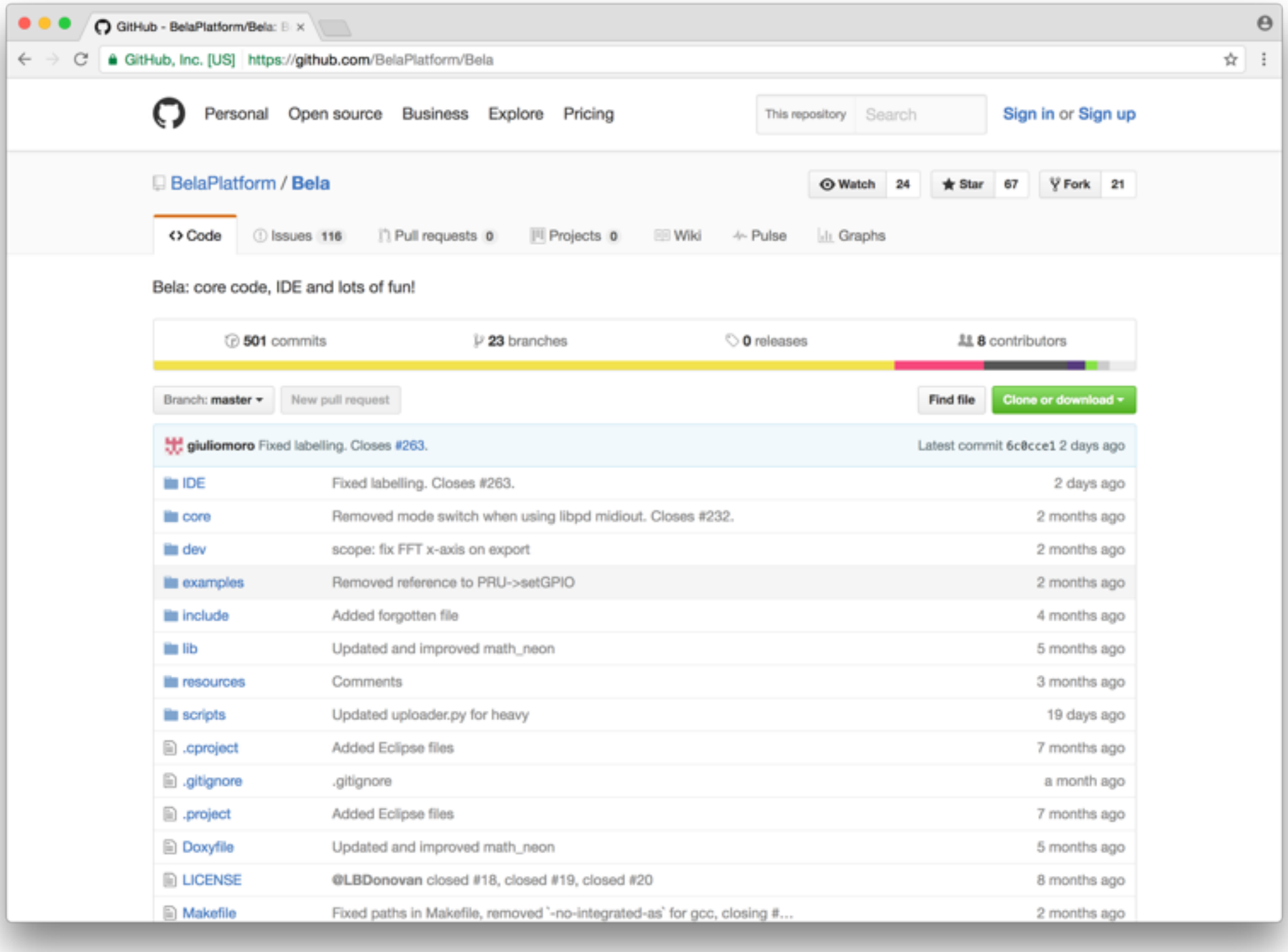
starter kit



multiplexer capelet

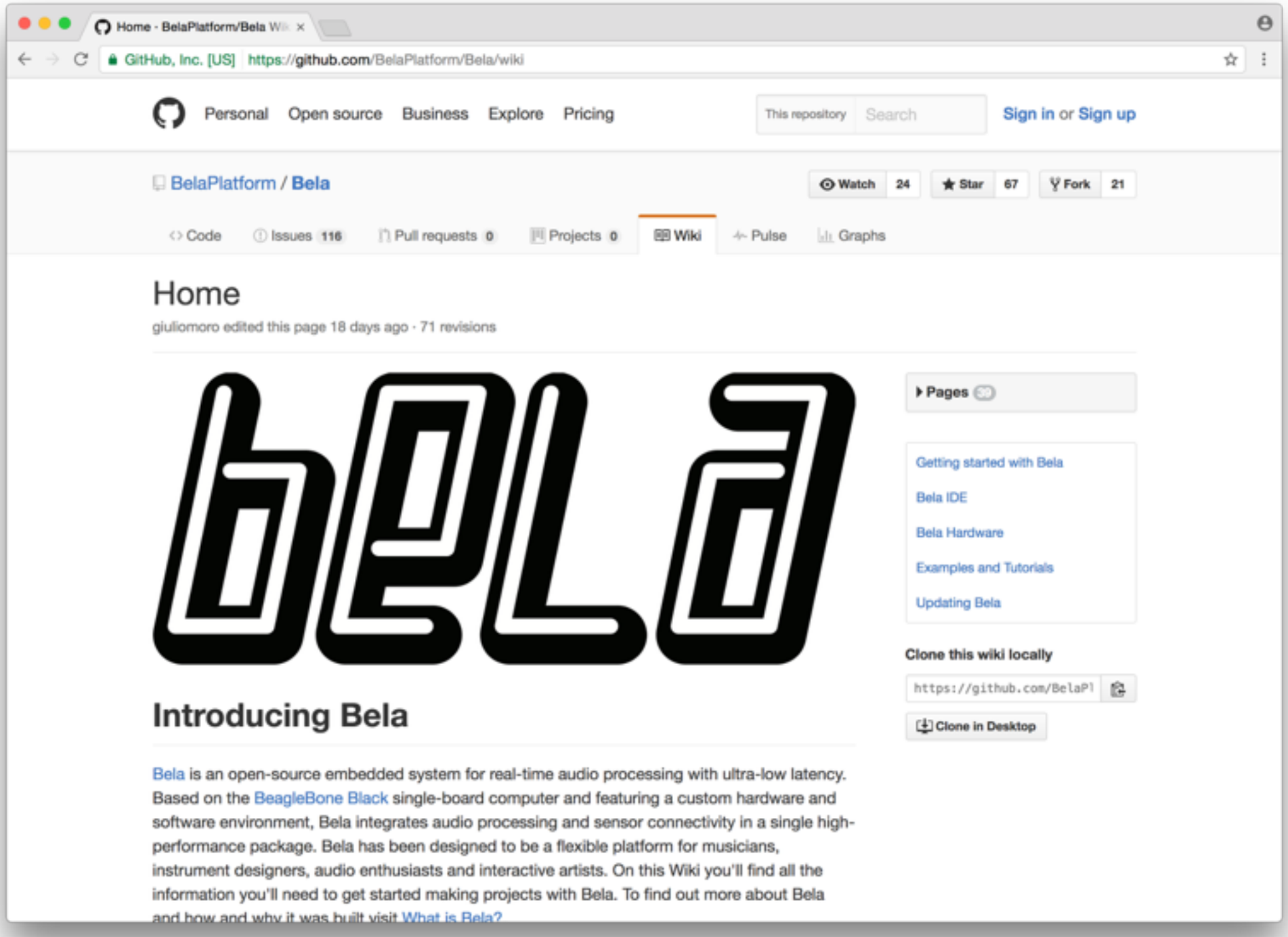
Community resources

repository



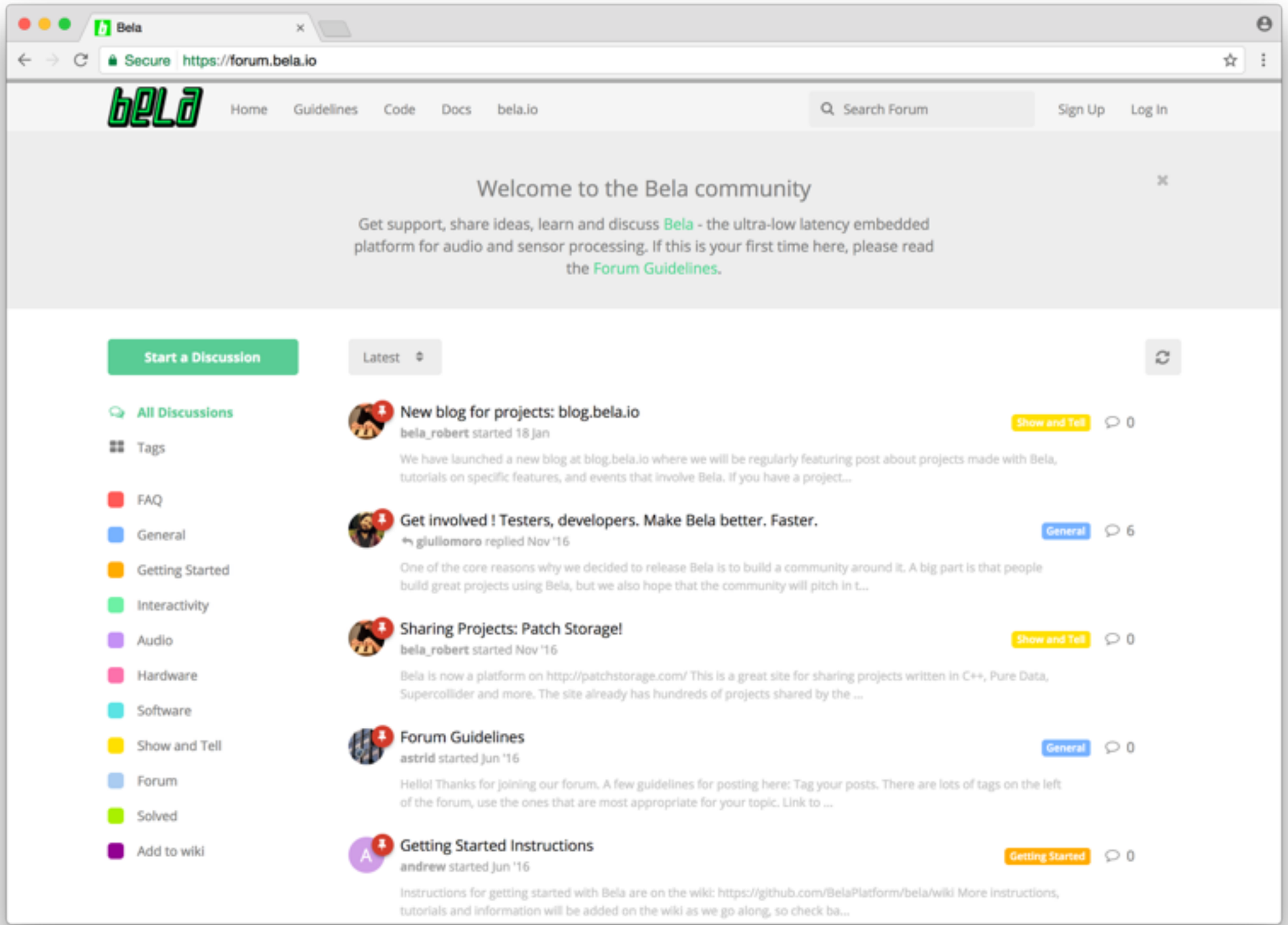
Community resources

tech docs (wiki)



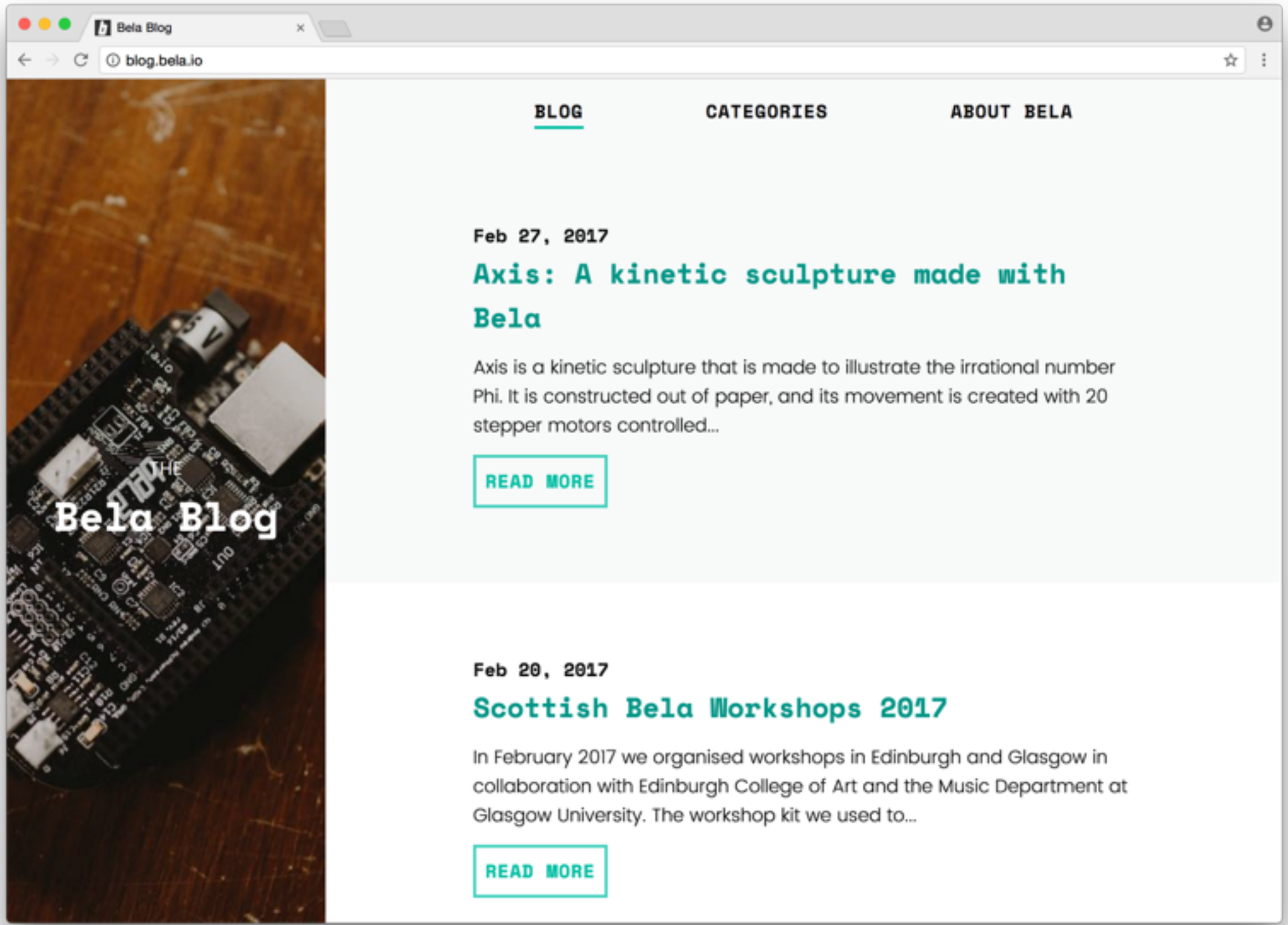
Community resources

forum



Community resources

blog



Questions?

Giulio Moro

giulio@bela.io

@giu_lio_mo_ro

c4dm.eecs.qmul.ac.uk

instrumentslab.org

bela.io

EPSRC