C++ Real-Time Audio Programming with Bela

Dr Andrew McPherson

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London

Founder and Director, Bela

Lecture 10: Latency

What you'll learn today:

What latency is, and why it matters
Sources of latency
How to measure latency

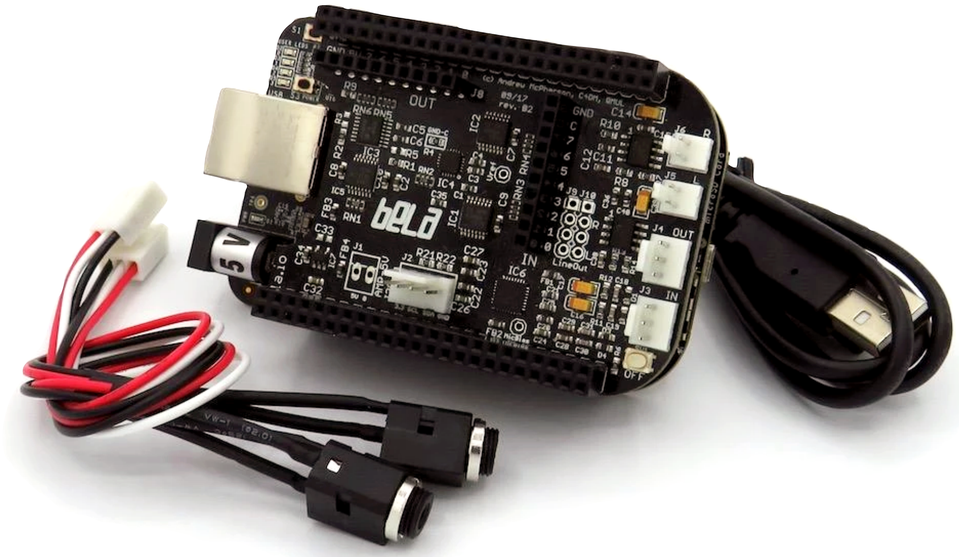
What you'll make today:

An automatic latency tester

Companion materials:

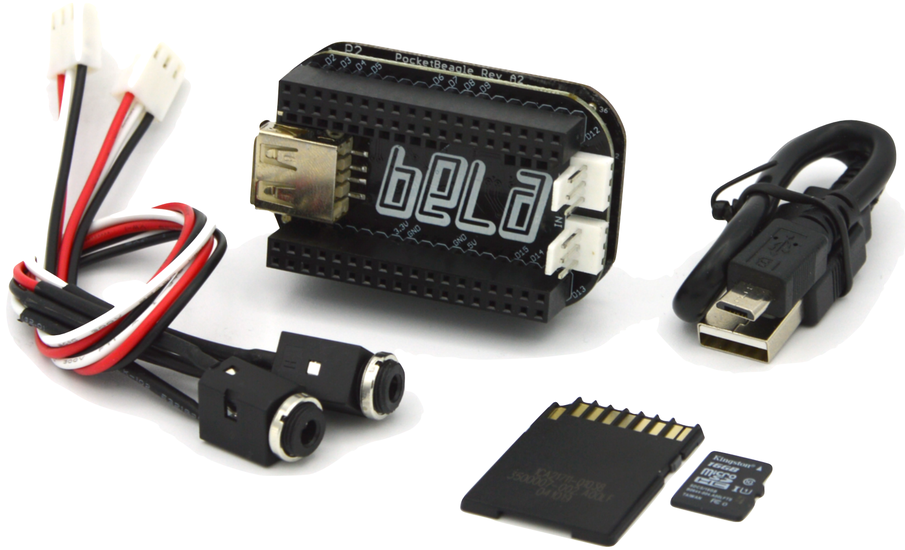
github.com/BelaPlatform/bela-online-course

What you'll need



Bela Starter Kit

or



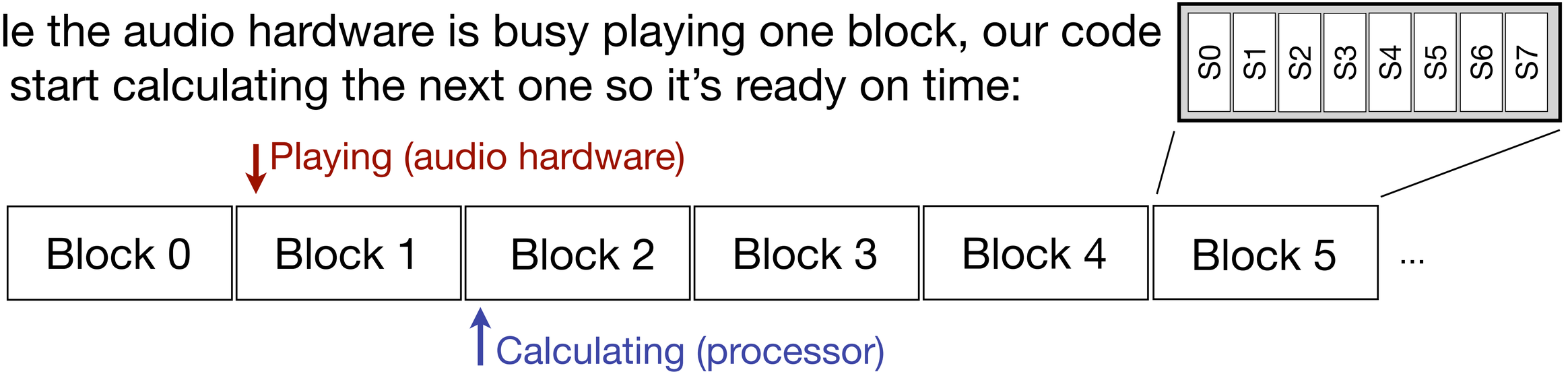
Bela Mini Starter Kit

[shop.bela.io]

Optional for this lecture:
oscilloscope and
function generator

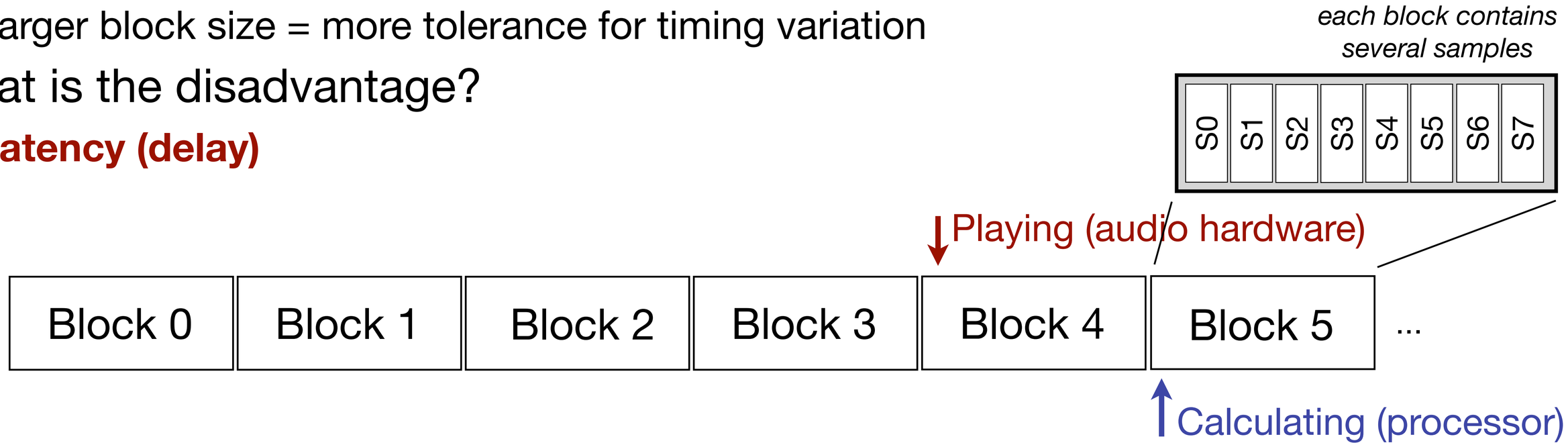
Review: block-based processing

- A common real-time approach: process in blocks of several samples
  - Generate enough samples at a time to get through the next few milliseconds
  - Typical block sizes on GPOSes: 32 to 512 samples
    - Usually a power of 2 for reasons of driver efficiency
  - Typical block sizes on Bela: 2 to 32 samples
    - Default is 16; can be changed in the Settings tab of the IDE
    - Bela runs the audio code in hard real time, so we can make stronger timing guarantees
- While the audio hardware is busy playing one block, our code can start calculating the next one so it's ready on time:



Block-based processing

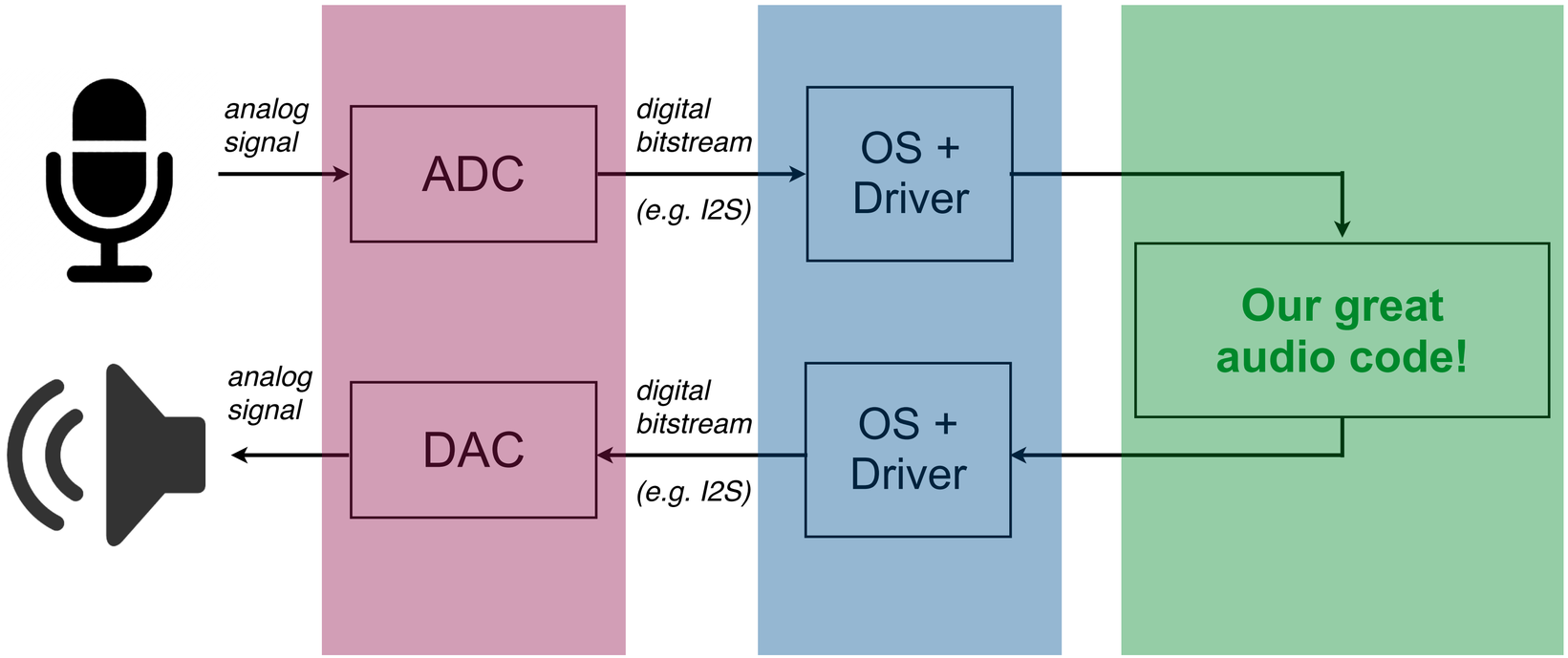
- Advantages of blocks vs. processing 1 sample at a time:
  - We need to run our function less often
  - We always generate 1 block ahead of what is actually playing
  - Suppose that one block lasts 5ms, and running our code takes 1ms
  - Now, we can tolerate a delay of up to 4ms if the OS is busy with other tasks
  - Larger block size = more tolerance for timing variation
- What is the disadvantage?
  - **Latency (delay)**



Latency

Delay between action and reaction, a fundamental property of DSP systems

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

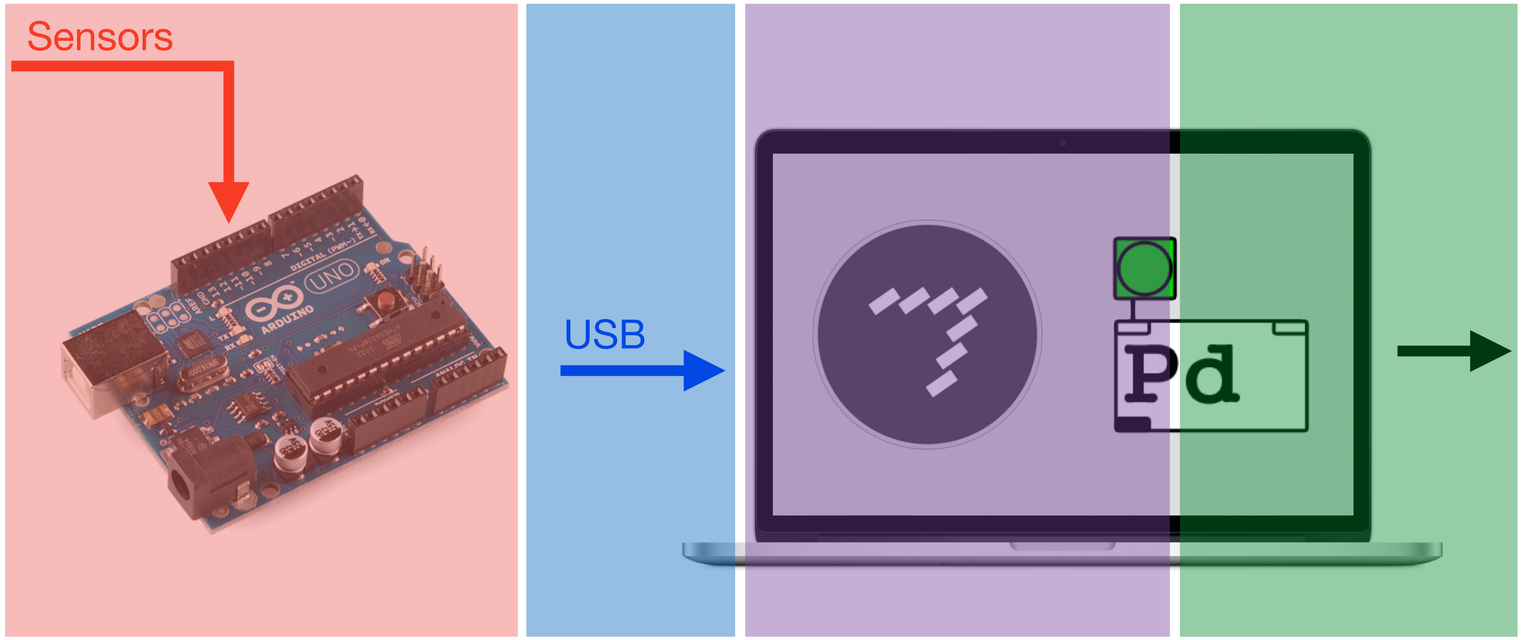
Buffering latency

- 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
- Examples: filters, block-based processing (e.g. FFT)

Interaction latency (computers + peripherals)



Sampling latency

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

Communication latency

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

OS driver latency

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

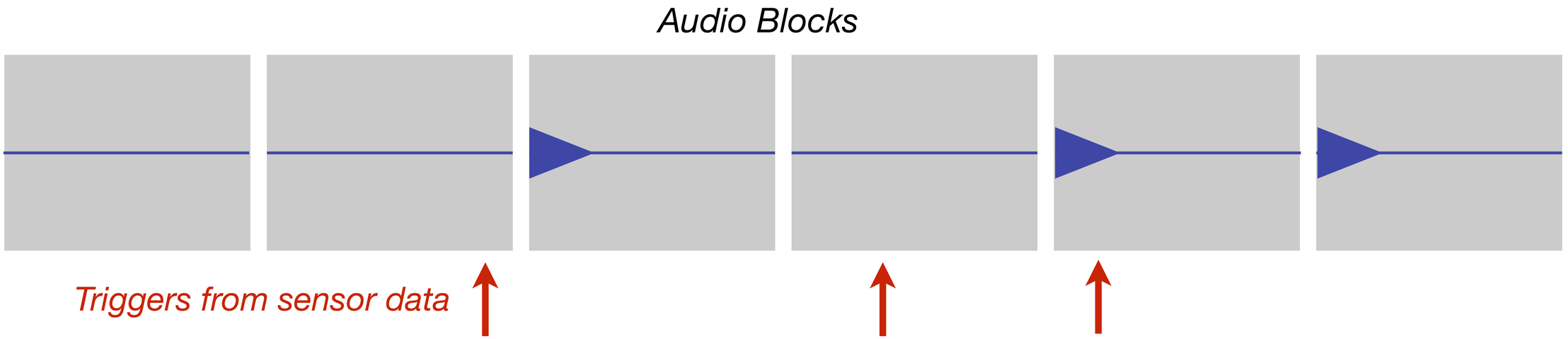
Audio latency

- As before: code, OS, converters

Not applicable for Bela analog and digital I/O!

Jitter

- Round-trip audio latency is usually constant
- Action-to-sound latency can vary randomly
  - Audio and sensors are handled on different threads
  - This variation in latency is known as jitter

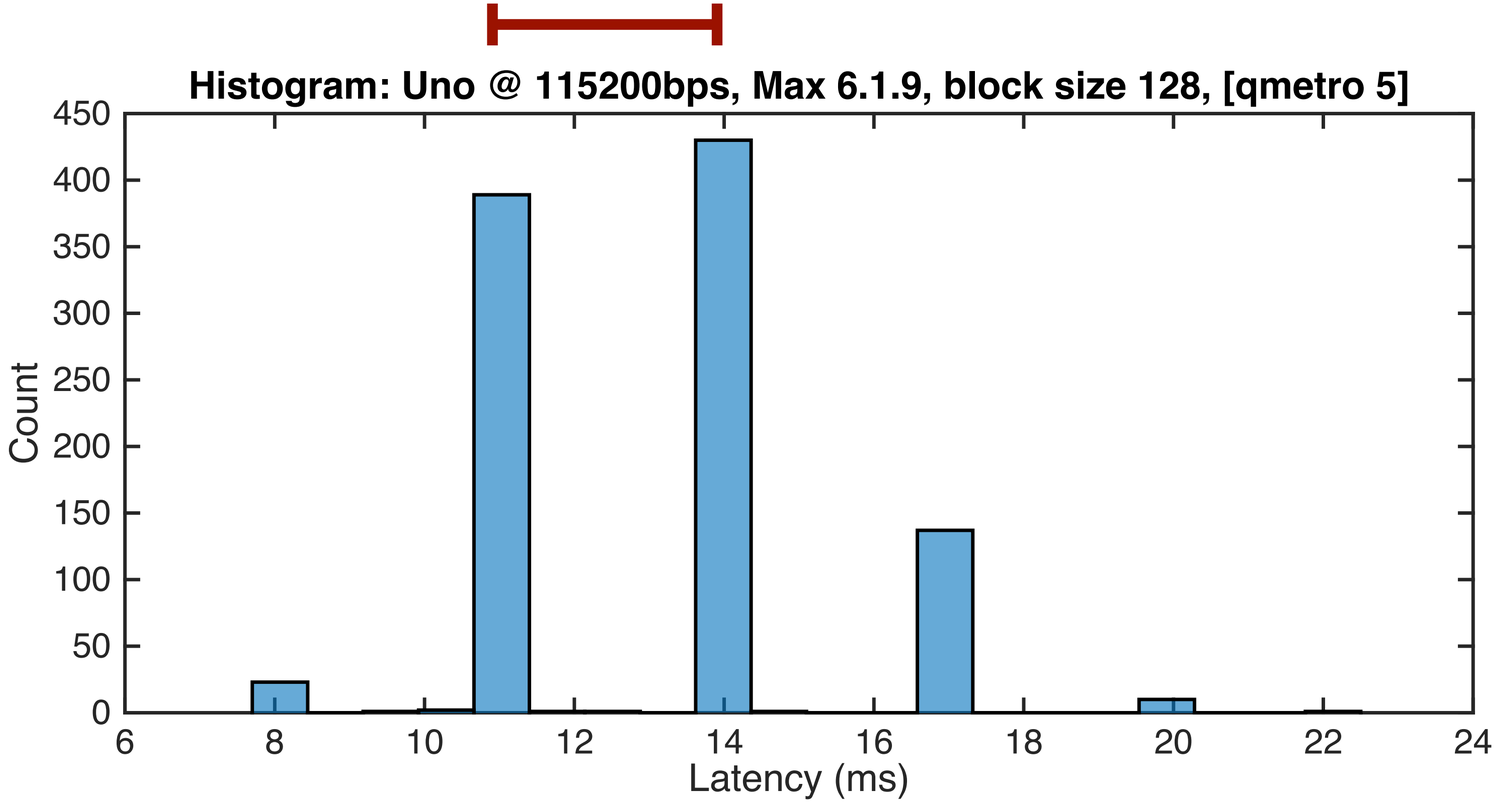


- Unless we carefully plan otherwise, sensor data processed at block intervals
  - Implication: larger block sizes mean more jitter

Jitter example

- Consider a microcontroller (Arduino) attached by USB to a computer
  - We use a reasonably fast serial speed (115.2kbps)
  - 44.1kHz audio sample rate, 128 sample blocks (2.9ms)

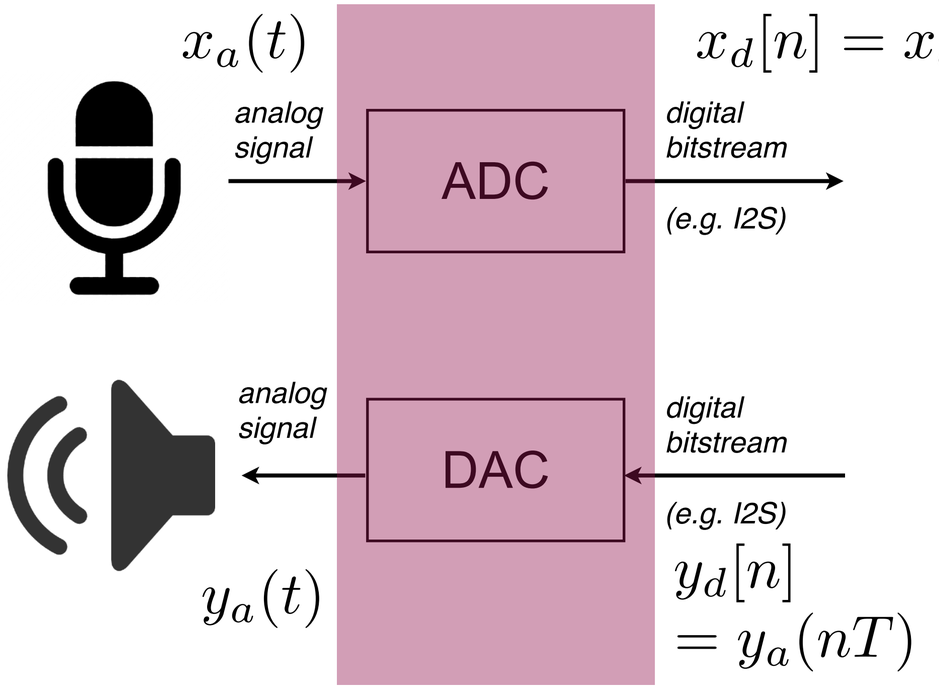
- Take a histogram of 1000+ measurements of trigger-to-sound latency:
  - Jitter shows up at intervals of the audio block size



Impact of latency

- Does latency matter? **Yes!**
- Long latency causes an audible delay
  - But even an imperceptible delay can make an interactive system less responsive
  - Latency is especially problematic with digital musical instruments
- 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.
  - Percussionists notice a reduction in digital instrument quality with 20ms of latency, or 10ms of latency +/- 3ms of jitter
    - R. H. Jack, A. Mehrabi, T. Stockman and A. McPherson. “Action-sound Latency and the Perceived Quality of Digital Musical Instruments: Comparing Professional Percussionists and Amateur Musicians.” *Music Perception* 36(1), 2018.

Conversion latency



Conversion latency

- Ideal ADCs and DACs convert instantaneously between continuous and discrete time
  - In practice, converters introduce latency
- Audio ADCs/DACs use sigma-delta modulation
  - Way of greatly improving output resolution using built-in digital filters in the conversion process
  - But these filters, like all filters, introduce latency
- Conversion latency is an inherent part of the converter
  - Doesn't depend on CPU or operating system
  - Where do we find the specifications?
    - **Look at the datasheet!**

Conversion latency on Bela (DAC)

ELECTRICAL CHARACTERISTICS (continued)

At 25°C, AVDD_DAC, DRVDD, IOVDD = 3.3 V, DVDD = 1.8 V, f_S = 48-kHz, 16-bit audio data (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
DAC DIGITAL INTERPOLATION – FILTER f_S = 48 ksps						
	Pass band		0		0.45 f_S	Hz
	Pass-band ripple			±0.06		dB
	Transition band		0.45 f_S		0.55 f_S	Hz
	Stop band		0.55 f_S		7.5 f_S	Hz
	Stop-band attenuation			65		dB
	Group delay			21/f_S		s
DIGITAL I/O						
V_{IL}	Input low level		–0.3		0.3 × IOVDD	V
V_{IH}	Input high level(4)	IOVDD > 1.6 V	0.7 × IOVDD			V
		IOVDD < 1.6 V	1.1			
V_{OL}	Output low level				0.1 × IOVDD	V
V_{OH}	Output high level		0.8 × IOVDD			V

$21/f_S$ means 21 samples of group delay

Delay time depends on sample rate

At 44.1kHz:
21/44100 = 476µs

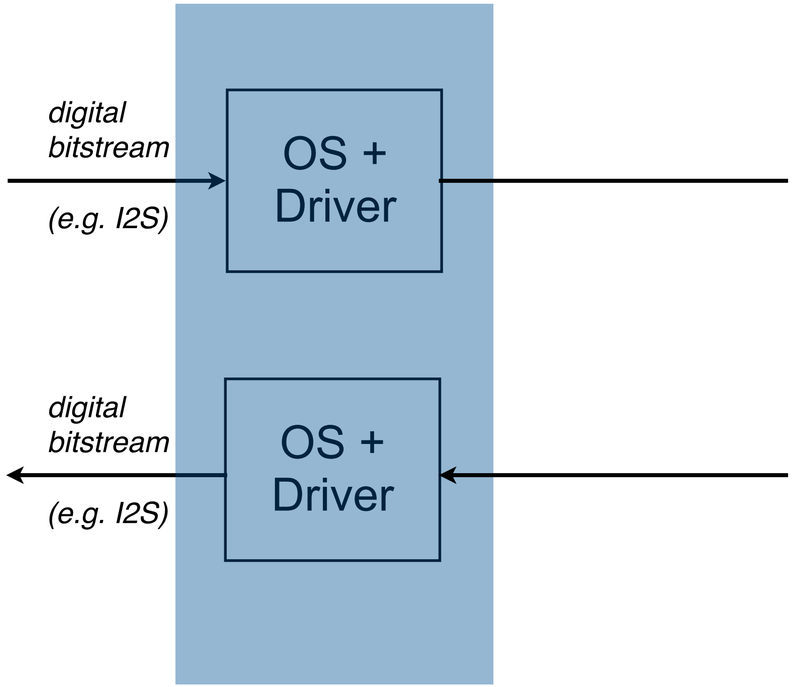
Conversion latency on Bela (ADC)

	Input signal level	Differential input	1.414	V_{RMS}
	Signal-to-noise ratio, A-weighted[1] [2]	f_S = 48 ksps, 0-dB PGA gain, inputs ac-shorted to ground, differential mode	92	dB
THD	Total harmonic distortion	f_S = 48 ksps, 0-dB PGA gain, –2-dB full-scale 1-kHz input signal, differential mode	–91	dB
ANALOG PASS THROUGH MODE				
	Input to output switch resistance, (rds_{ON})	MIC1/LINE1 to LINE_OUT	330	Ω
		MIC2/LINE2 to LINE_OUT	330	
ADC DIGITAL DECIMATION FILTER, f_S = 48 kHz				
	Filter gain from 0 to 0.39 f_S		±0.1	dB
	Filter gain at 0.4125 f_S		–0.25	dB
	Filter gain at 0.45 f_S		–3	dB
	Filter gain at 0.5 f_S		–17.5	dB
	Filter gain from 0.55 f_S to 64 f_S		–75	dB
	Filter group delay		$17/f_S$	s

$17/f_S$ means 17 samples of group delay

At 44.1kHz:
17/44100
= 385µs

Buffering latency

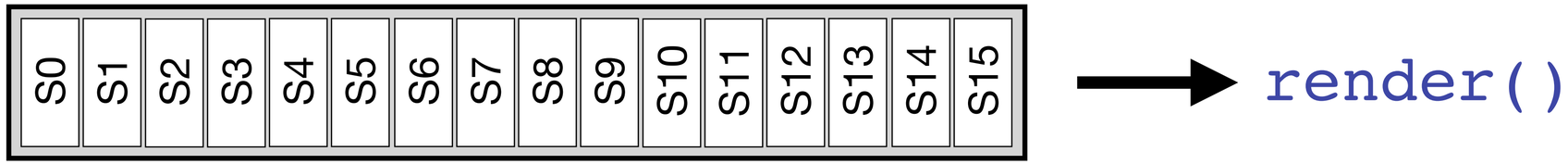


Buffering latency

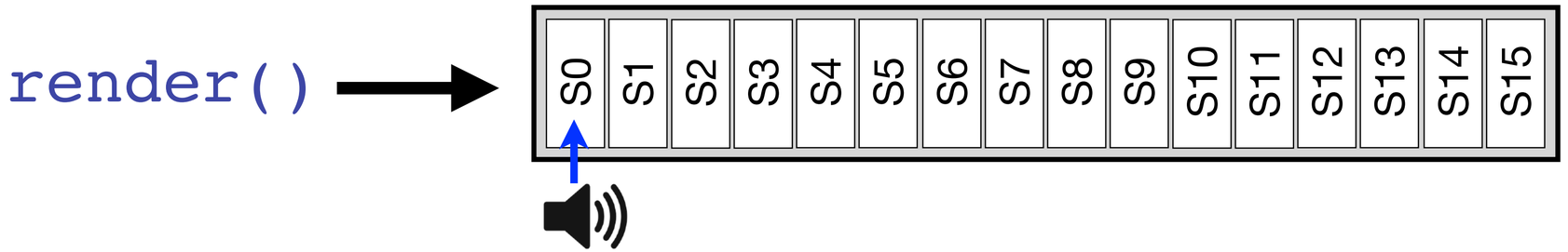
On many platforms, this is the most significant source of latency

Buffering latency

- Block-based processing introduces latency
- On input side: how is a block of samples created?
  - For block size of N: we wait until N samples have arrived from ADC

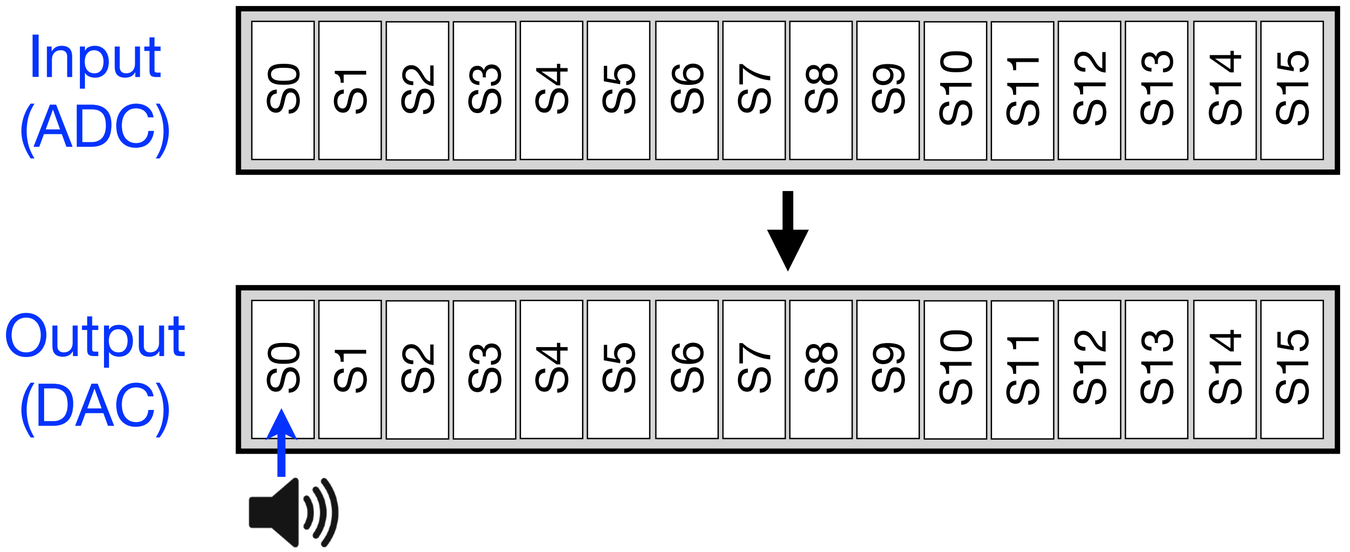


- On output side: how is a block played by DAC?
  - We can only start playing once the block is calculated (i.e. once `render()` has finished)
  - So how long until the last sample is played?
    - N samples after the the block is sent to the hardware



Buffering latency: best case

- Let's consider the best case for buffering latency (i.e. processing by block)
- As soon as block is received, we immediately send it back out:

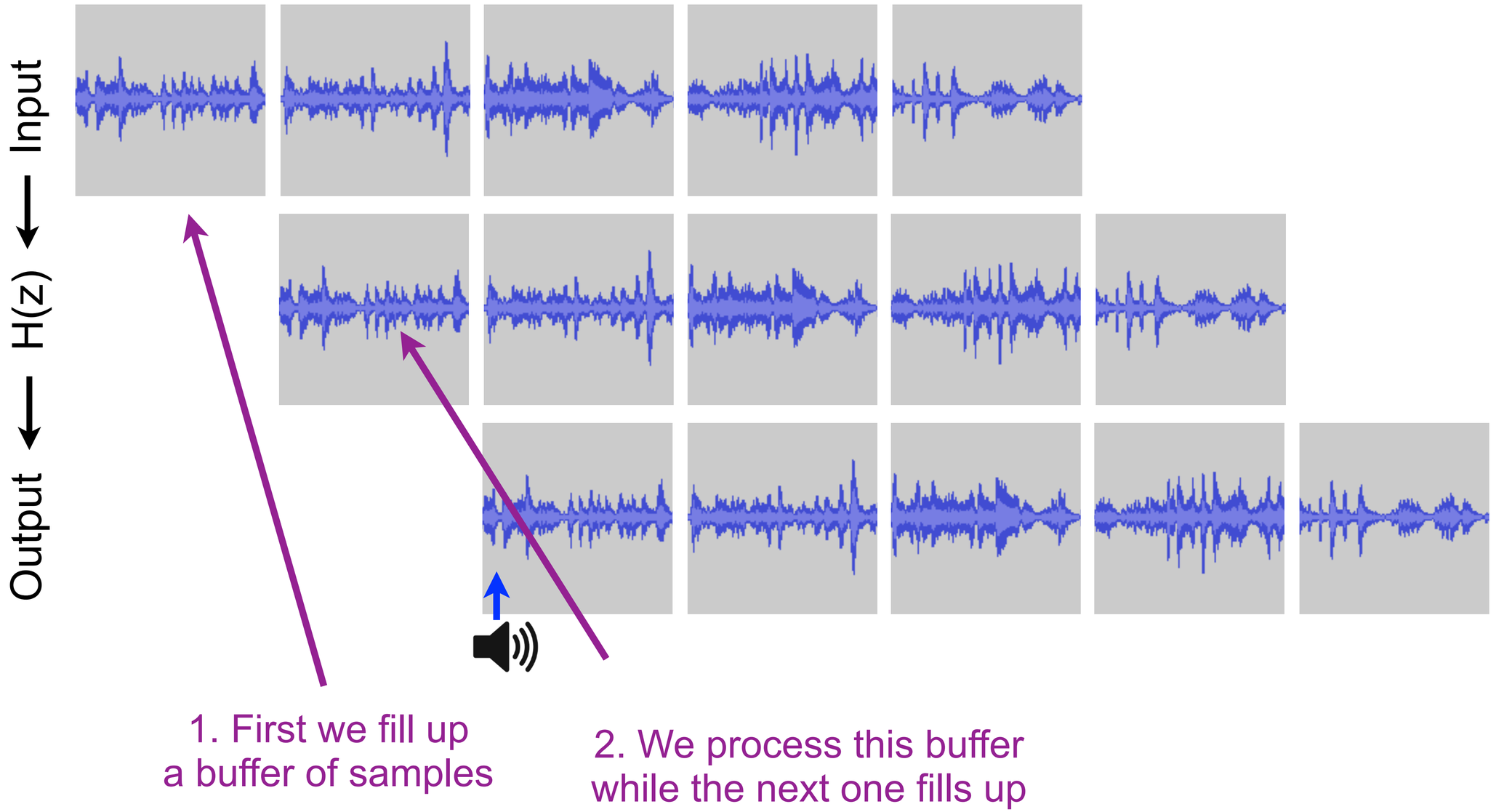


- Here we incur one block length of delay
  - For example, buffer size of 128 at 44.1kHz = 128 samples delay = 2.9ms

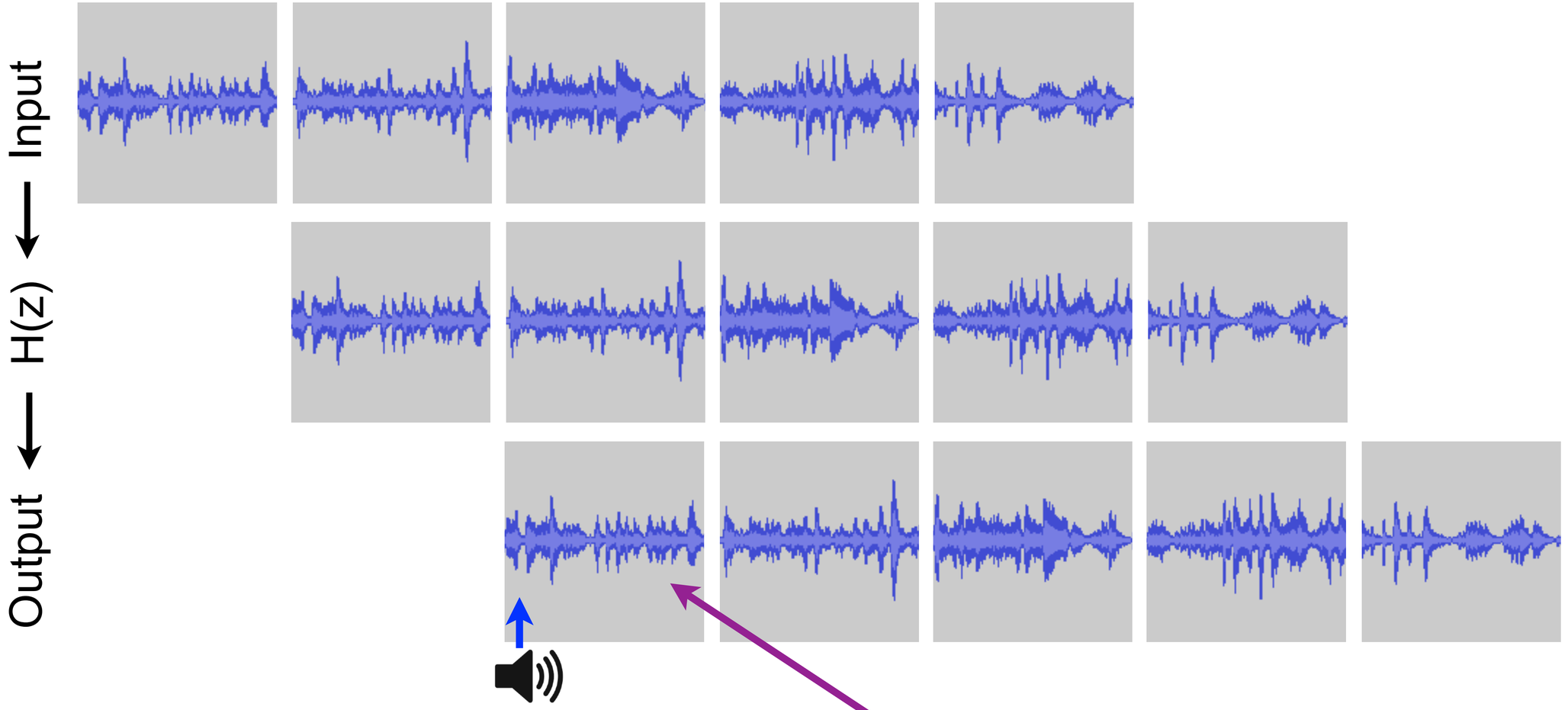
Buffering latency: it gets worse...

- What's the practical problem with previous example?
- When do we do our processing?
  - How about waiting until the block is finished and ready?
  - Now we have to respond instantly if we plan to send it out to the DAC right away
  - This is exactly what we were trying to avoid!
- We need some time to process the block before we send it out
  - i.e. we need time for `render()` to do its computation
  - Solution: add an extra buffer
  - While we process one block, the hardware is filling the next one with samples
  - Adds another N samples latency!

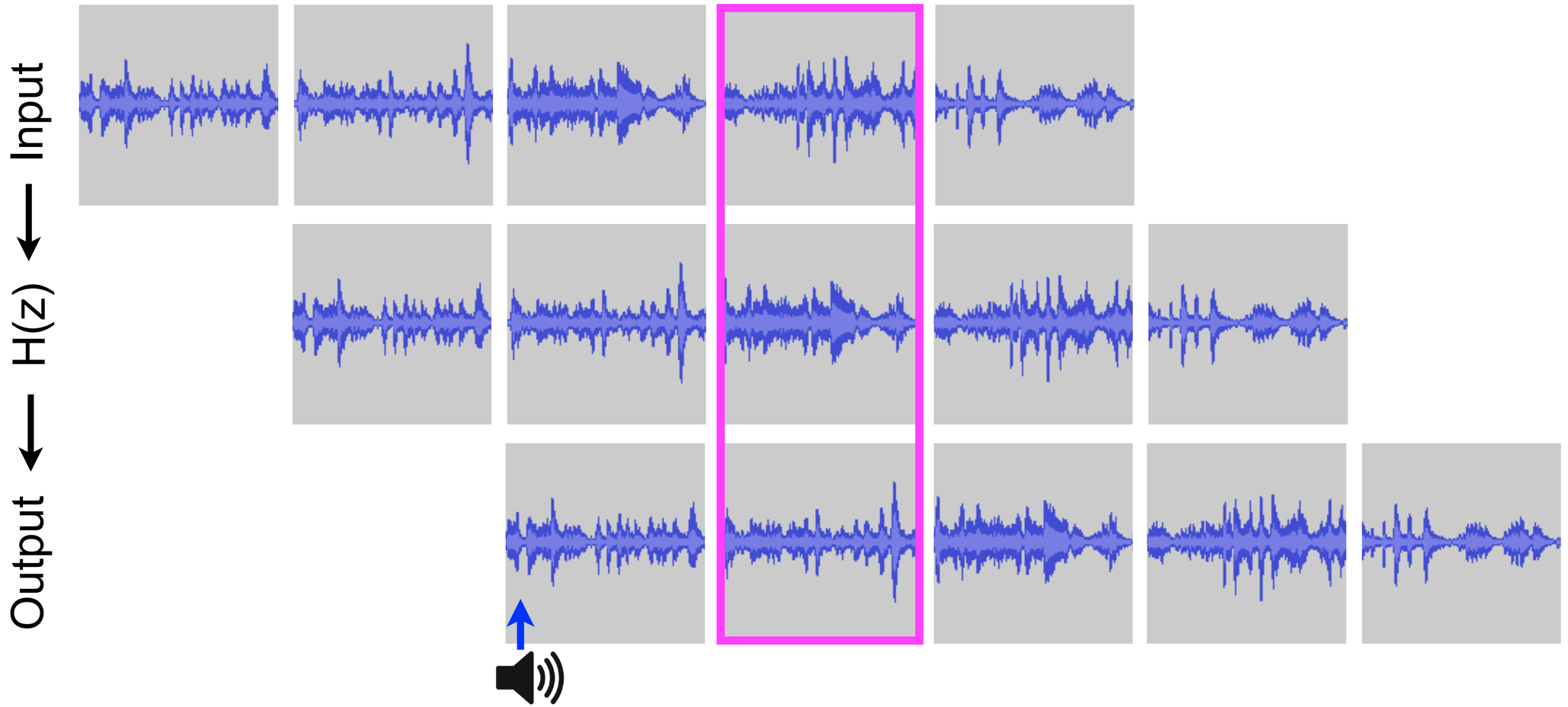
Buffering illustration



Buffering illustration

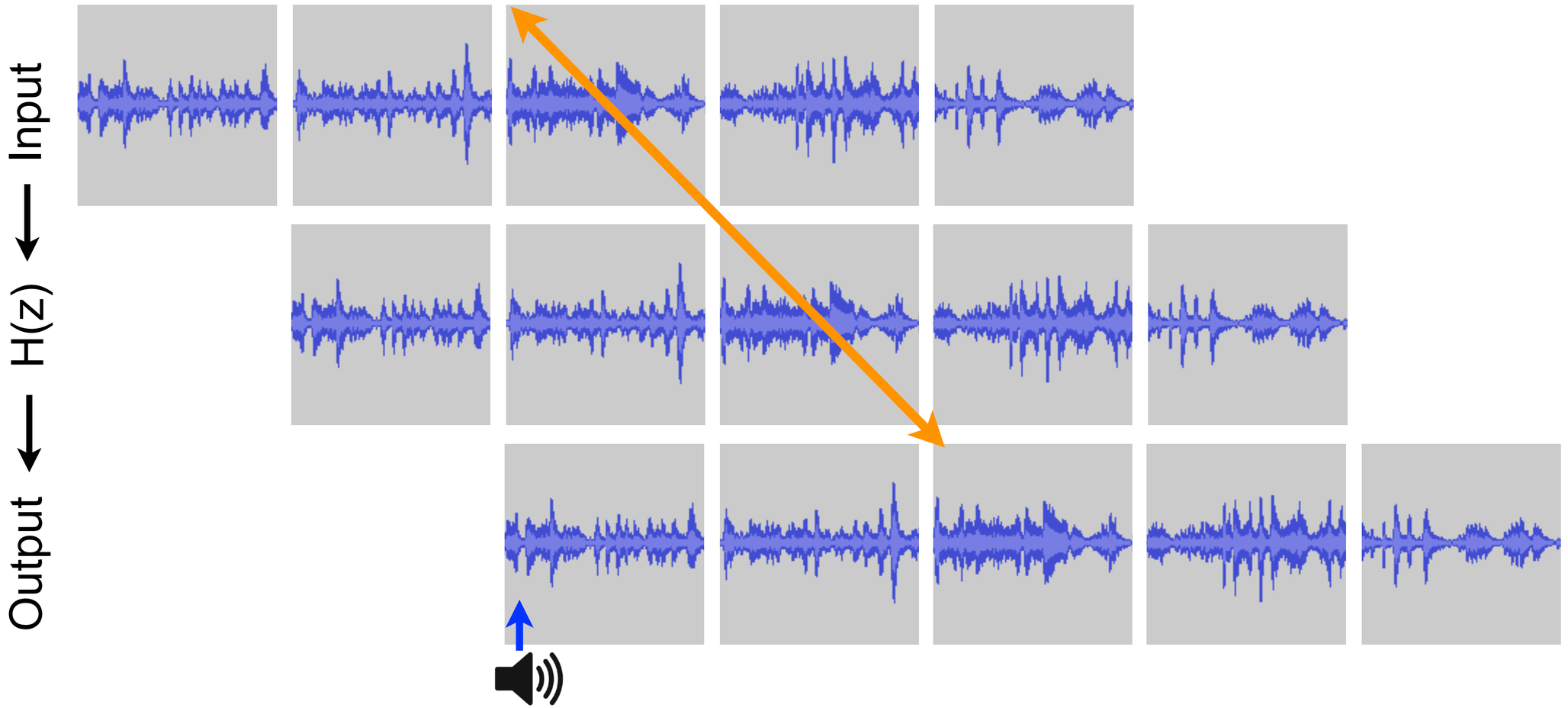


Buffering illustration



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

Buffering illustration



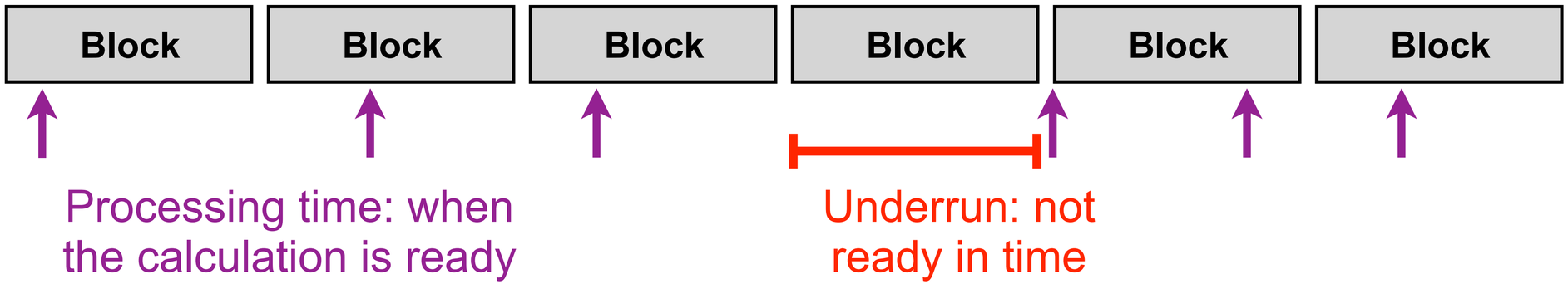
Total latency is 2x buffer length

Underruns and overruns

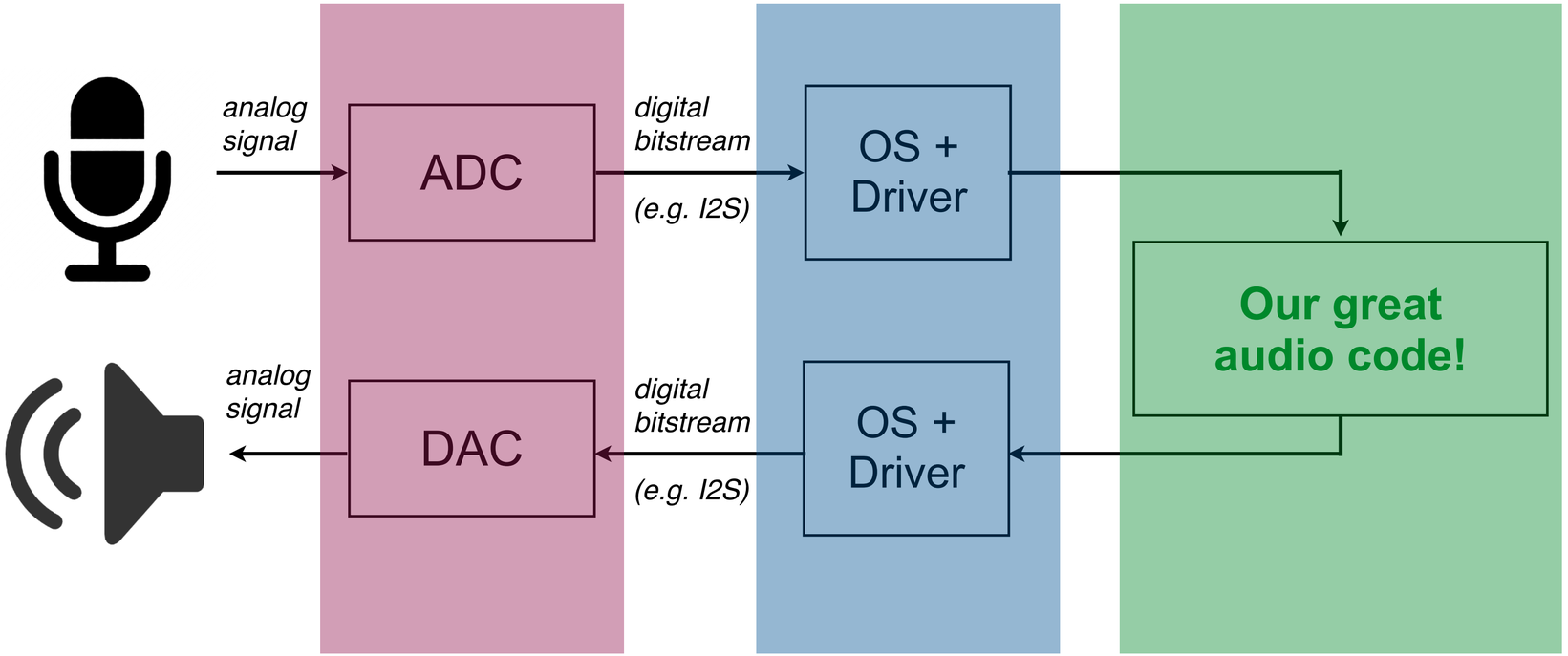
- In previous example, we have one buffer length to finish processing our filter
- What happens if we take longer to compute it?
  - We run out of data! Nothing to send to DAC
  - This is known as a buffer underrun
  - Usually results in a gap or a click in the output
- Alternatively, what if ADC data comes in faster than we can process it?
  - We will drop (lose) some of the input data
  - This is known as a buffer overrun
  - Sometimes both these conditions are referred to as “underruns” or “xruns”
- What happens to the output if we finish calculating sooner than expected?
  - Nothing different: output will play at same time as before

Further considerations

- Can hard real-time systems have underruns?
  - No.* Hard real-time systems meet a guaranteed timing specification
    - * At least not because of external system architecture. Your audio code could still take too long to run.
    - * Also provided our filter can be implemented in real time at all...
  - Underruns are usually due to unpredictable delays in the OS, before it calls the audio callback function
    - e.g. on Bela, before `render()` is called
- Why choose a larger buffer size?
  - Possible to handle more timing uncertainty



Filtering latency



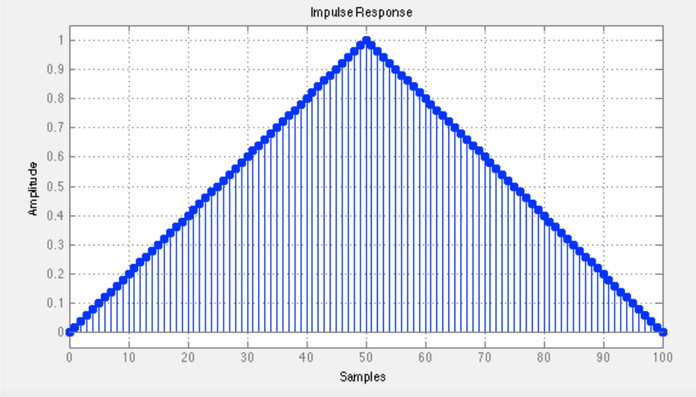
Conversion latency Buffering latency Group delay

Group delay

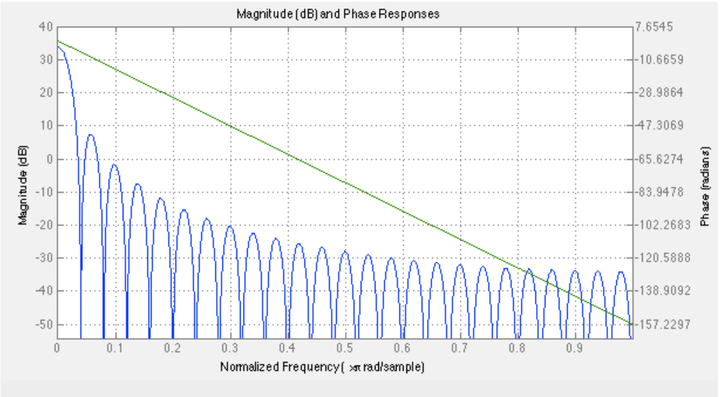
- Every nontrivial filter adds group delay
  - We are considering LTI (linear time-invariant) filters here as is common in DSP
- Consider an FIR filter first: $y[n] = \sum_{k=0}^{M} b_k x[n-k]$
  - What is the group delay assuming symmetrical coefficients? (Remember from DSP?)

$$\tau_g(\omega) = \frac{M-1}{2}$$ (notice: constant with respect to frequency)

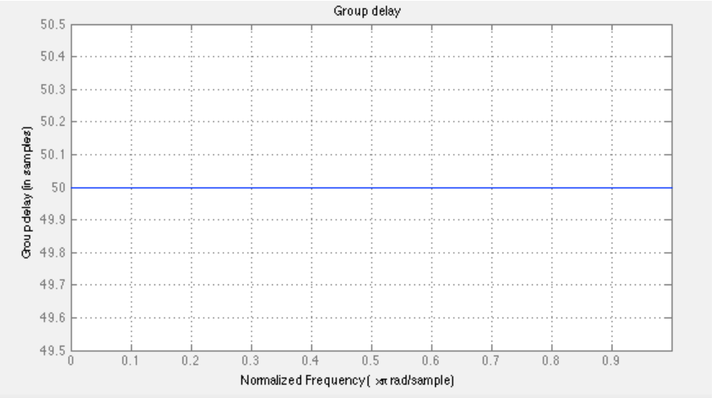
  - Symmetric (and anti-symmetric) FIR filters known as linear phase filters



Impulse Response



Magnitude & Phase Response



Group Delay

Group delay

- How does group delay relate to phase?
  - Group delay is the (negative) derivative of phase:

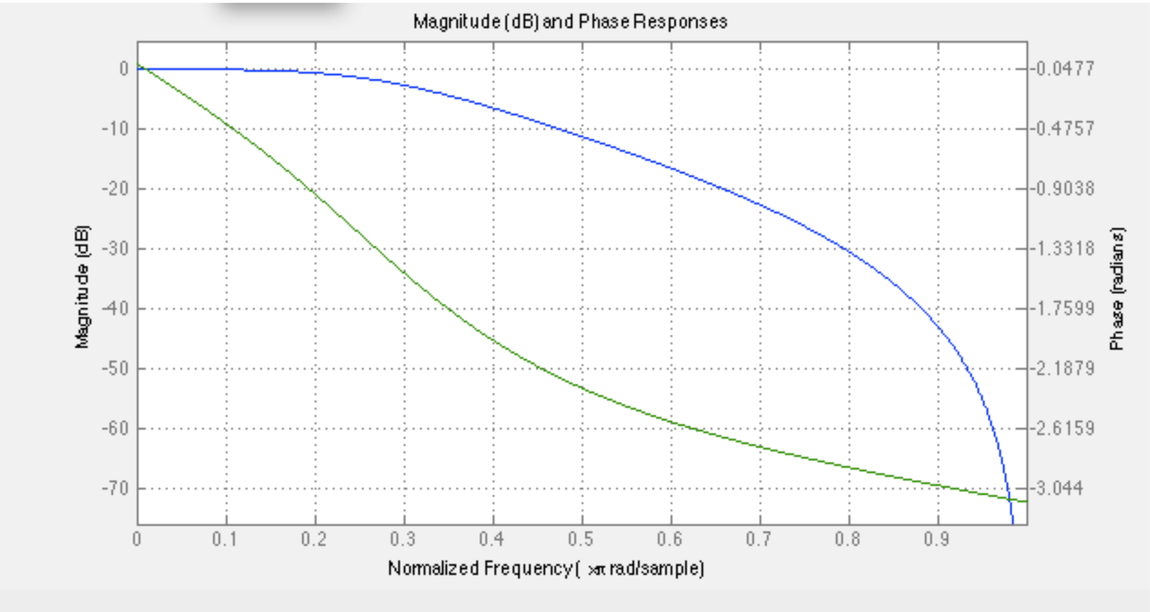
$$\tau_g(\omega) = -\frac{d\phi(\omega)}{d\omega}$$

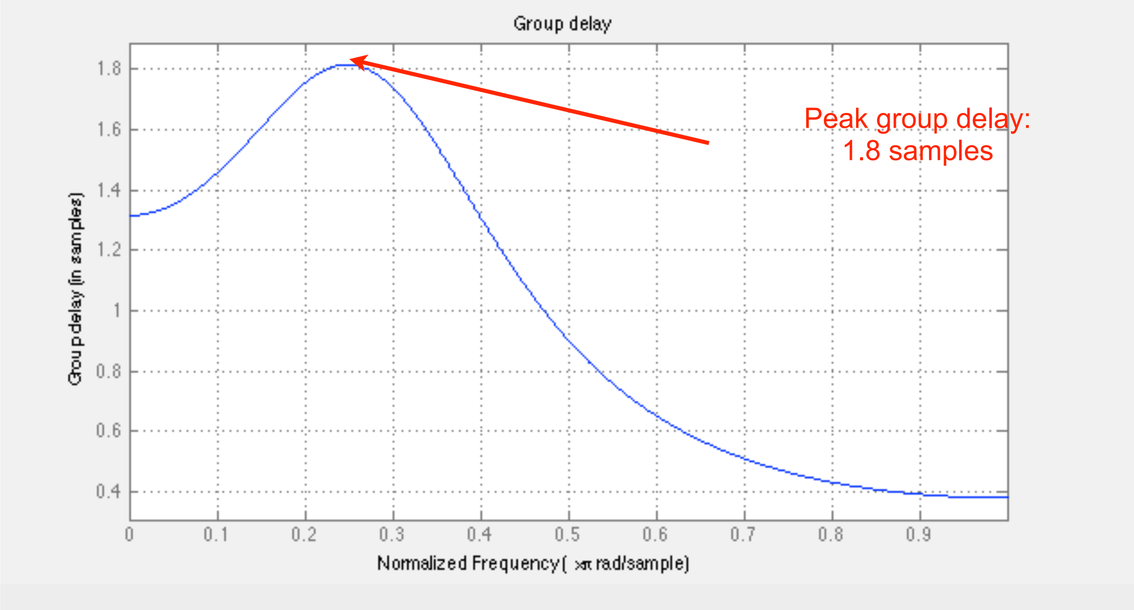
  - In other words: at whatever frequencies the phase changes quickly, those frequency components will be delayed
- Check: why do linear phase FIR filters have constant group delay?
  - Phase response is linear with respect to frequency
  - So derivative is constant with respect to frequency
  - This is not true of IIR filters!

Group delay of IIR filters

- For IIR filters, group delay is often highest around the cutoff frequency
  - This is where the phase changes the most rapidly with respect to frequency
  - In general, higher order and higher Q mean higher group delay

2nd order Butterworth

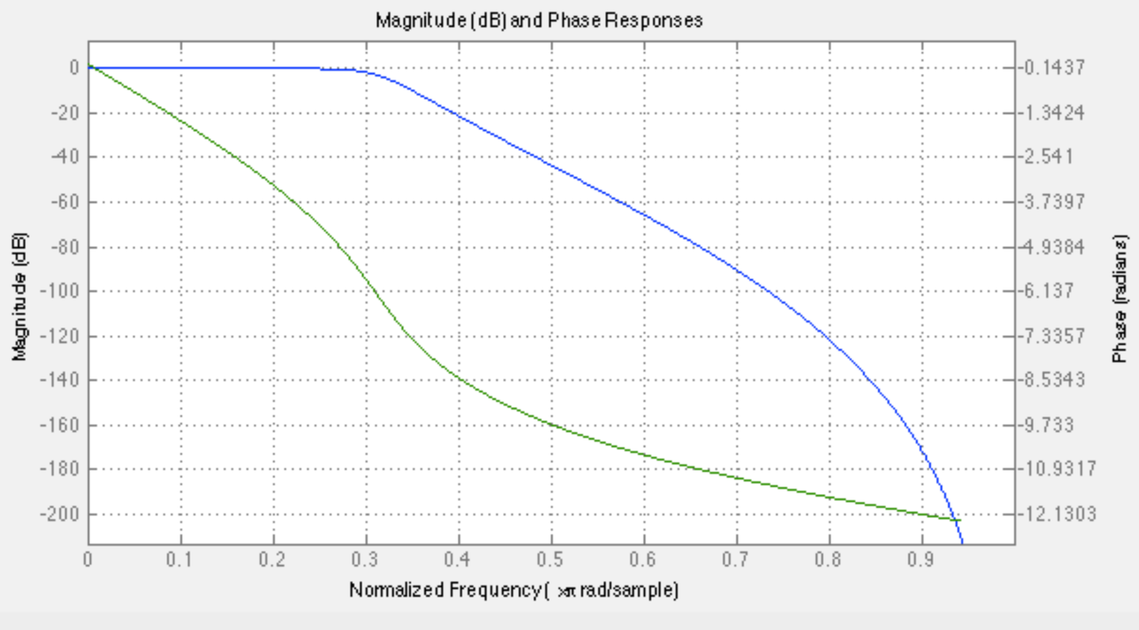


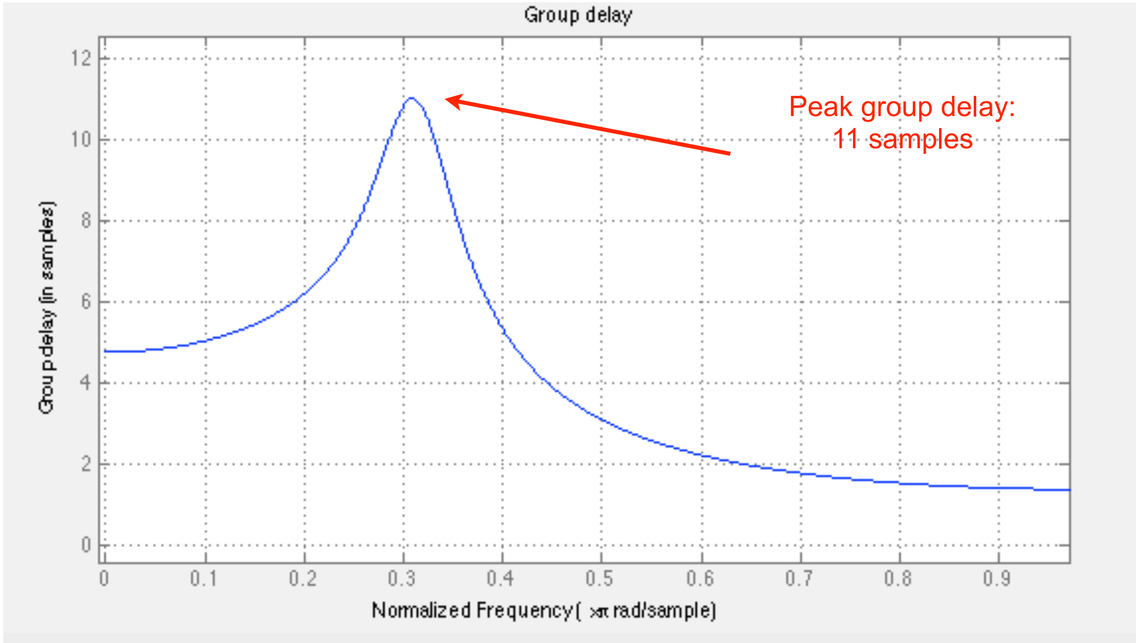


Group delay of IIR filters

- For IIR filters, group delay is often highest around the cutoff frequency
  - This is where the phase changes the most rapidly with respect to frequency
  - In general, higher order and higher Q mean higher group delay

8th order Butterworth





Group delay summary

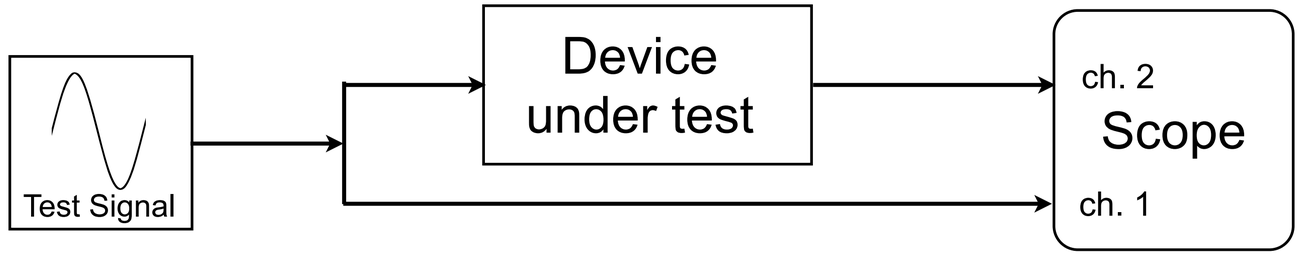
- Here's the point:

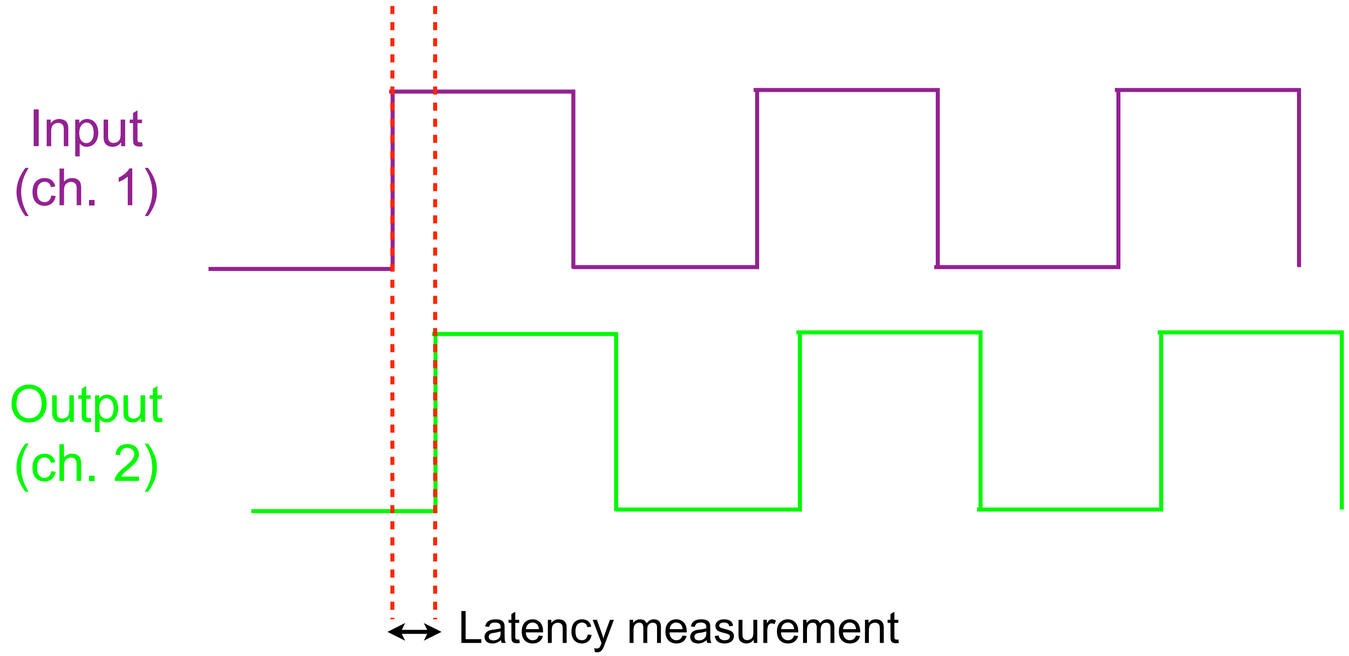
1. Every filter has group delay (i.e. latency)
   - Except the trivial case of multiplying by a constant
2. This is an inherent mathematical property
   - A faster processor won't reduce the group delay
3. Delay can depend on frequency
   - This can disrupt phase relationships, lead to subtle artefacts in audio signals

- Check: what is group delay for a 1-second delay effect?
  - 1 second!
  - How would you represent it as a filter for f_s = 44.1kHz?

Latency testing

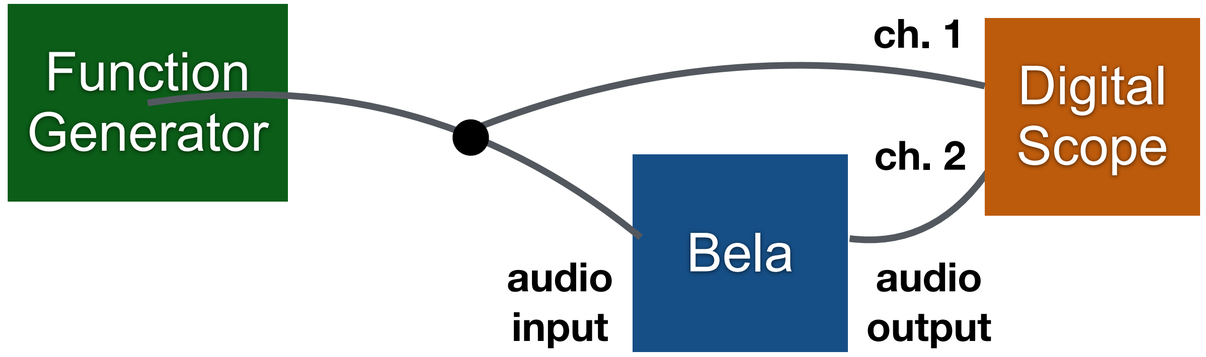
- We are going to measure the latency of Bela
  - Easiest approach: a simple pass-through project that copies the audio input to the audio output
  - Use an oscilloscope (a real benchtop one) to measure the delay
- Load the passthrough project from the Fundamentals section of the Bela examples
  - This passes the input to the output unchanged



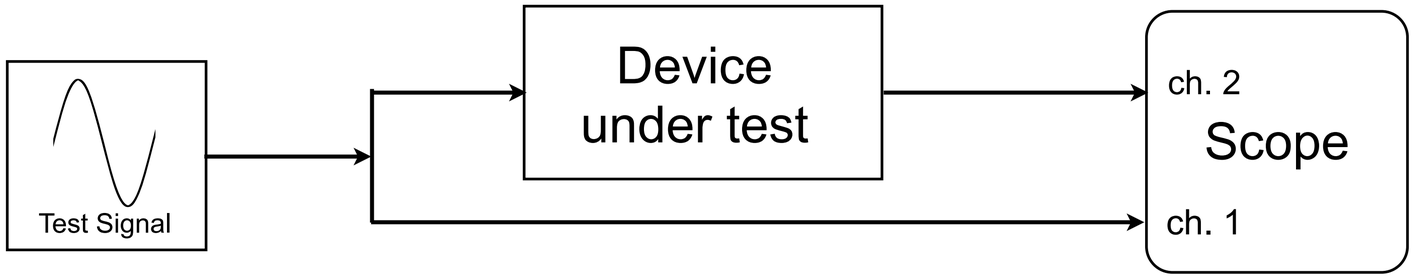


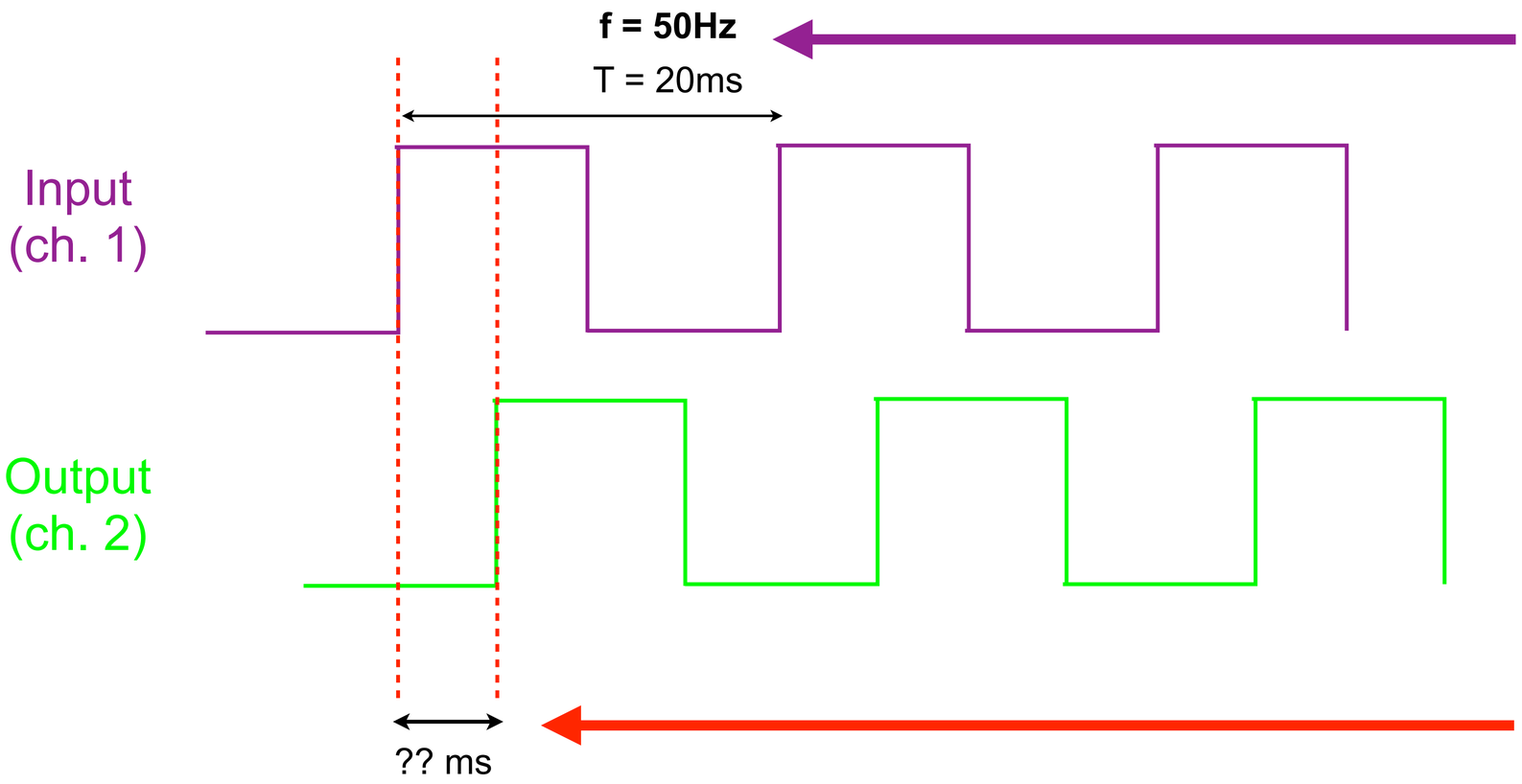
Latency testing

- Step 1: connect the function generator to the Bela audio input
  - You'll need either socket wires or a 3.5mm audio cable with wire leads on the other end
  - Optionally use a breadboard to connect the function generator and Bela together
- Step 2: connect the oscilloscope channel 1 to the Bela audio input
  - Connect to the same row of the breadboard
  - Alternatively, use a BNC T-adapter to connect directly from function generator to scope
- Step 3: connect the Bela audio output to oscilloscope channel 2



Latency testing



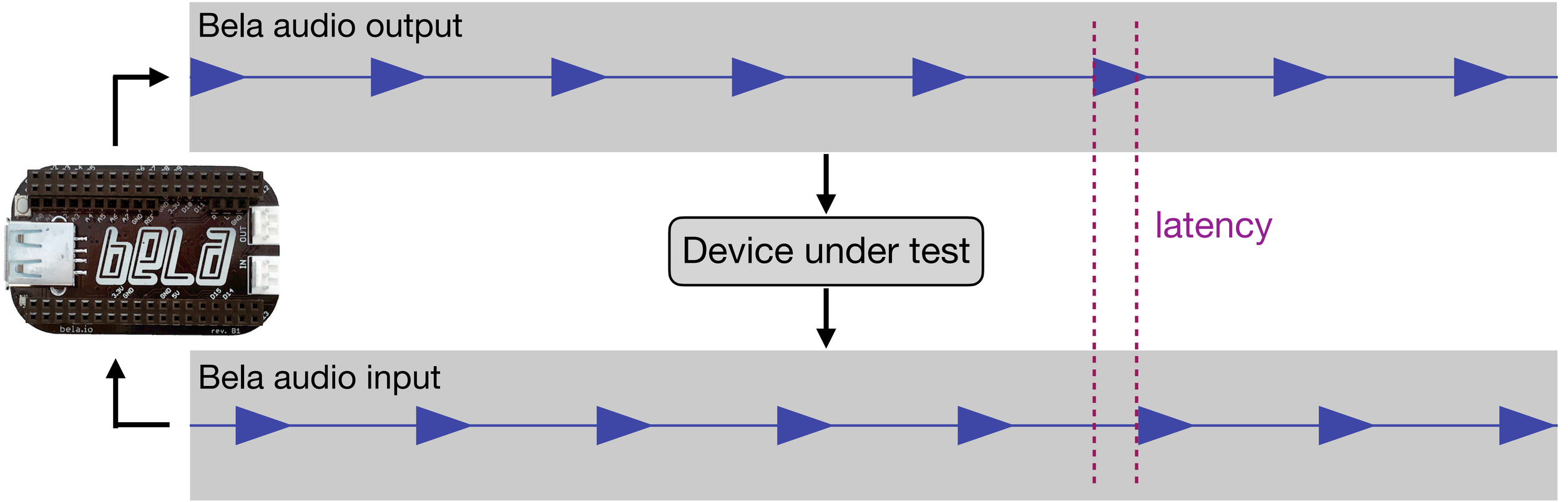


Set the function generator to a 50Hz square wave with no more than 0.5V p-p

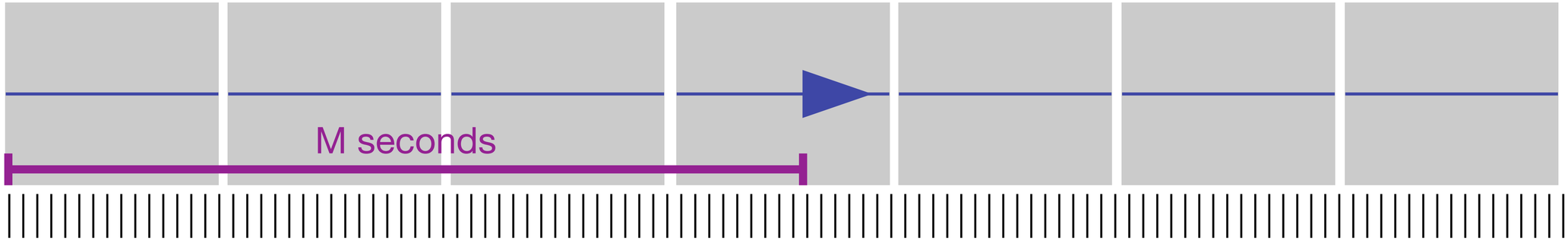
Use the screen divisions or the cursors on the oscilloscope to measure the latency

Automatic latency tester

- Let's use Bela to measure the latency of another audio device
- At a regular intervals, we'll send a pulse to the output and measure how long it takes to come back to the input:



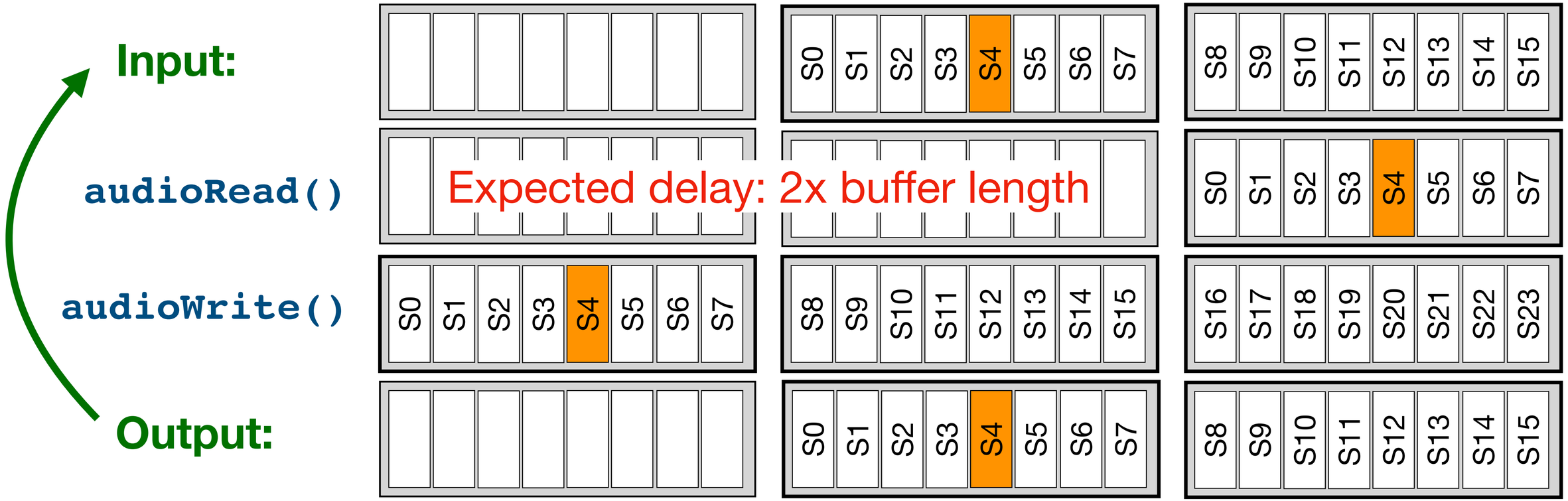
Review: measuring time



- To generate events with a time delay, we count elapsed time as we go along
  - We can't use a delay statement in real-time audio, because we would get an underrun
  - There are system calls on Linux that give us the clock time, though many of these are not safe to run in Xenomai primary mode (i.e. they would cause mode switches on Bela)
- The simplest way to count time is to count audio samples
  - We know the sample rate, so we can convert samples to seconds:
  - (elapsed time in seconds) = (elapsed samples) * (sample rate)
  - Successive events might be in different blocks, so use a global variable to hold the number of elapsed samples

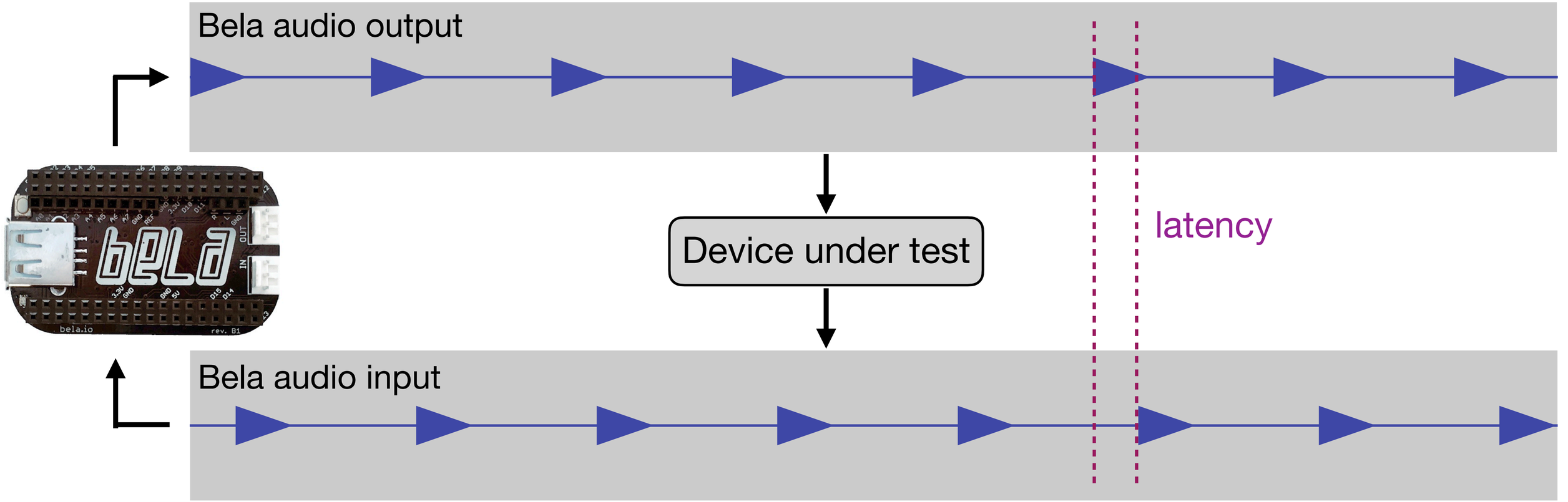
Loopback time

- If we connect the Bela audio output directly back to the input, how much latency do we expect, and why?
  - Presumably we should get the same measurement as we did on the oscilloscope!
  - Some latency in the audio codec itself, but most is in the buffering



Bela latency tester

- Task: in the latency-tester project
  - Fill in the two sections in `render()` marked `TODO`
  - Count samples to initiate a new pulse, and correct for the latency introduced by buffering



Keep in touch!

Social media:

@BelaPlatform

forum.bela.io

blog.bela.io

More resources and contact info at:

learn.bela.io/resources