Audio in standard C++

Timur Doumler

@timur_audio

C++Now
8 May 2019

This talk

- Audio in C++ today
- How does audio actually work?
  - Physical audio (sound waves, human ear, speakers, microphones…)
  - Digital audio (linear PCM, samples, buffers, channels, frames…)
  - Audio I/O (devices, callbacks…)
  - Higher-level functionality
- P1386: Audio for the C++ standard library
  - API design
  - Update: revision R2 in progress!
  - Reference implementation
  - Live demo!
  - Future steps
  - Open questions

Audio in C++ today

- All desktops, laptops and phones have audio I/O
- Many embedded devices have audio I/O

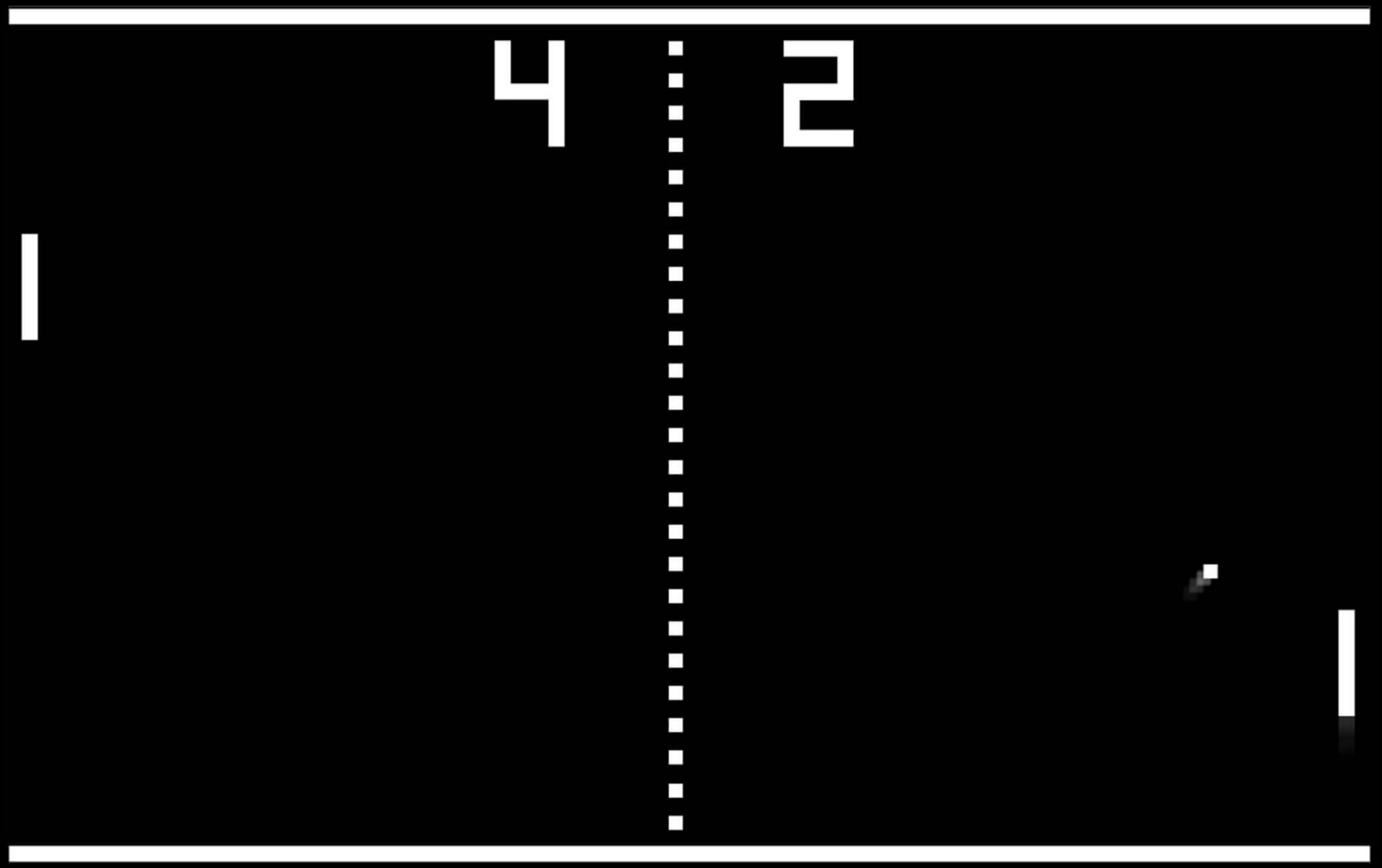
Audio in C++ today

- User interfaces on desktop + mobile
- Communication software
- Multimedia software

  (Media players, streaming software, …)
- Games
- Pro Audio

  (Digital instruments & effects, music creation/production software, …)
- Creative Coding
- Science & Research
- Education

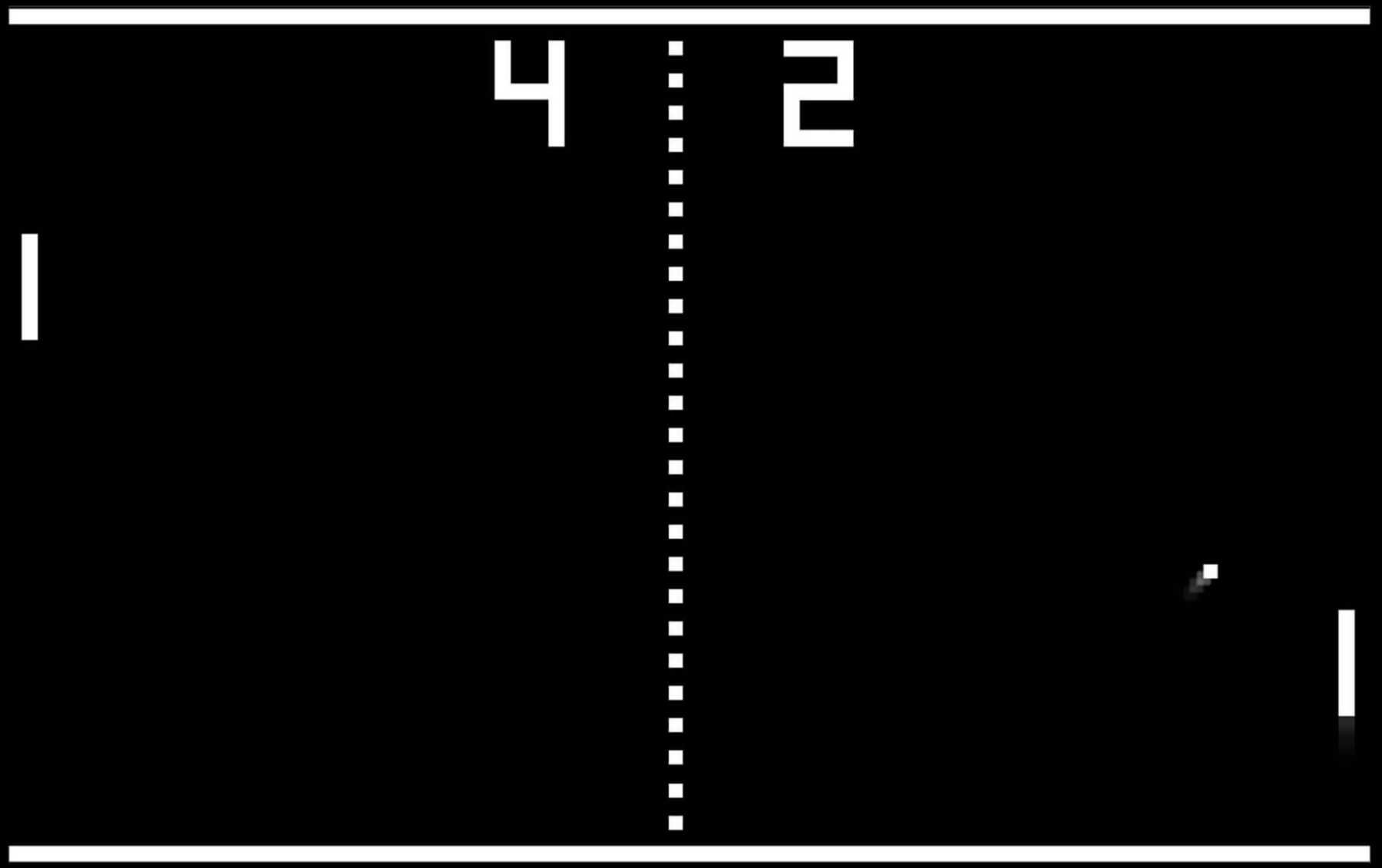
Audio is one of the basic aspects
of human-computer interaction.


4
2

Audio in C++ today

Audio in C++ today

```cpp
#include <iostream>

int main() {
    std::cout << '\a';
}
```


4
2

Audio in C++ today

- Native Audio APIs
  - Windows: WASAPI, ASIO
  - macOS/iOS: CoreAudio
  - Linux: JACK, ALSA, OSS
  - Android: OpenSL ES, AAudio, Oboe

Audio in C++ today

- Native Audio APIs
  - Windows: WASAPI, ASIO
  - macOS/iOS: CoreAudio
  - Linux: JACK, ALSA, OSS
  - Android: OpenSL ES, AAudio, Oboe
- Cross-platform middleware
  - General purpose: SDL, portaudio, RtAudio, libsoundio, …
  - Games: FMOD, WWise, Fabric
  - Pro Audio: JUCE, dozens of in-house frameworks

Audio in C++ today

- Native Audio APIs
  - Windows: WASAPI, ASIO
  - macOS/iOS: CoreAudio
  - Linux: JACK, ALSA, OSS
  - Android: OpenSL ES, AAudio, Oboe
- Cross-platform middleware
  - General purpose: SDL, portaudio, RtAudio, libsoundio, …
  - Games: FMOD, WWise, Fabric
  - Pro Audio: JUCE, dozens of in-house frameworks
- Many of the above are not C++, but C (or Java!)

Audio in C++ today

- Native Audio APIs
  - Windows: WASAPI, ASIO
  - macOS/iOS: CoreAudio
  - Linux: JACK, ALSA, OSS
  - Android: OpenSL ES, AAudio, Oboe
- Cross-platform middleware
  - General purpose: SDL, portaudio, RtAudio, libsoundio, …
  - Games: FMOD, WWise, Fabric
  - Pro Audio: JUCE, dozens of in-house frameworks
- Many of the above are not C++, but C (or Java!)
- At low-level, the same thing happens in all of them

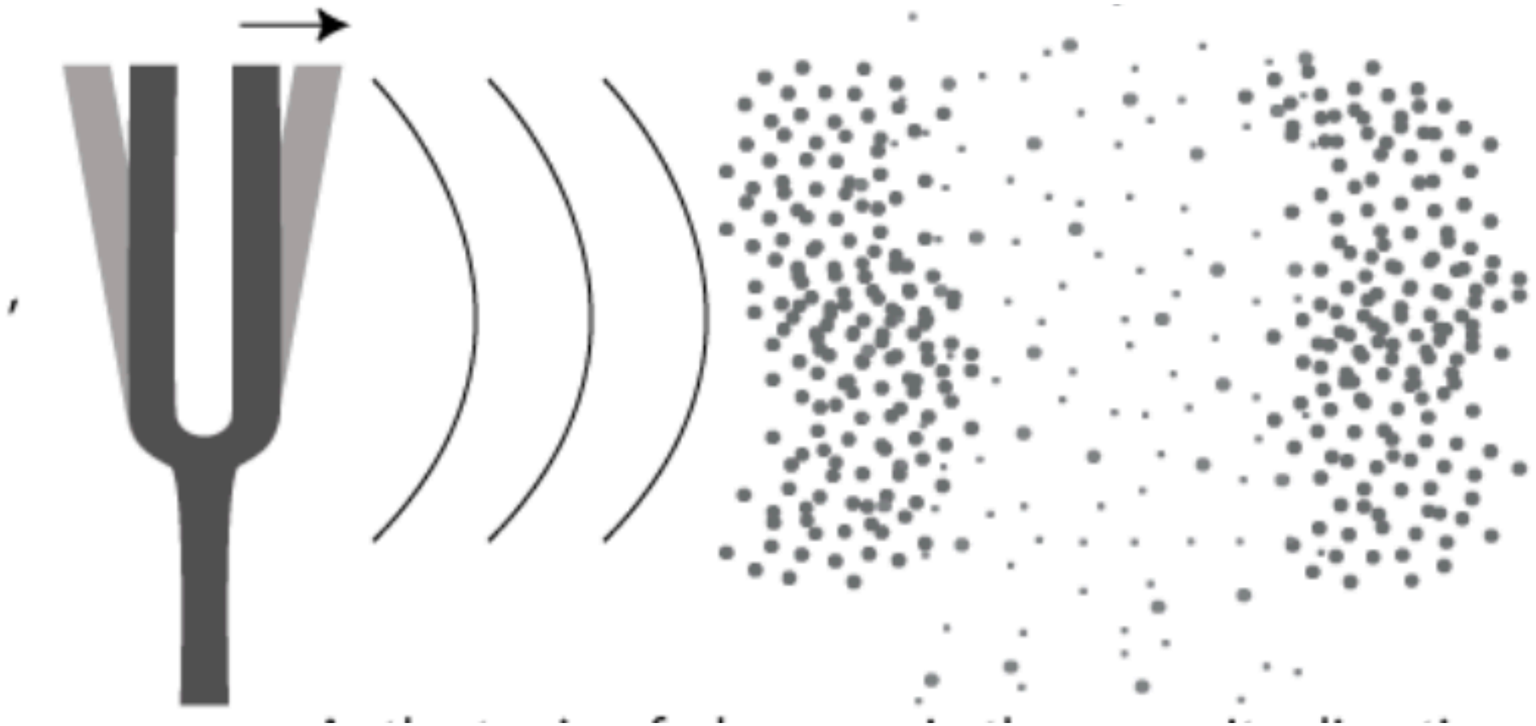
Audio in C++ today

- Native Audio APIs
  - Windows: WASAPI, ASIO
  - macOS/iOS: CoreAudio
  - Linux: JACK, ALSA, OSS
  - Android: OpenSL ES, AAudio, Oboe
- Cross-platform middleware
  - General purpose: SDL, portaudio, RtAudio, libsoundio, …
  - Games: FMOD, WWise, Fabric
  - Pro Audio: JUCE, dozens of in-house frameworks
- Many of the above are not C++, but C (or Java!)
- At low-level, the same thing happens in all of them
- Hasn’t substantially changed since the 90s

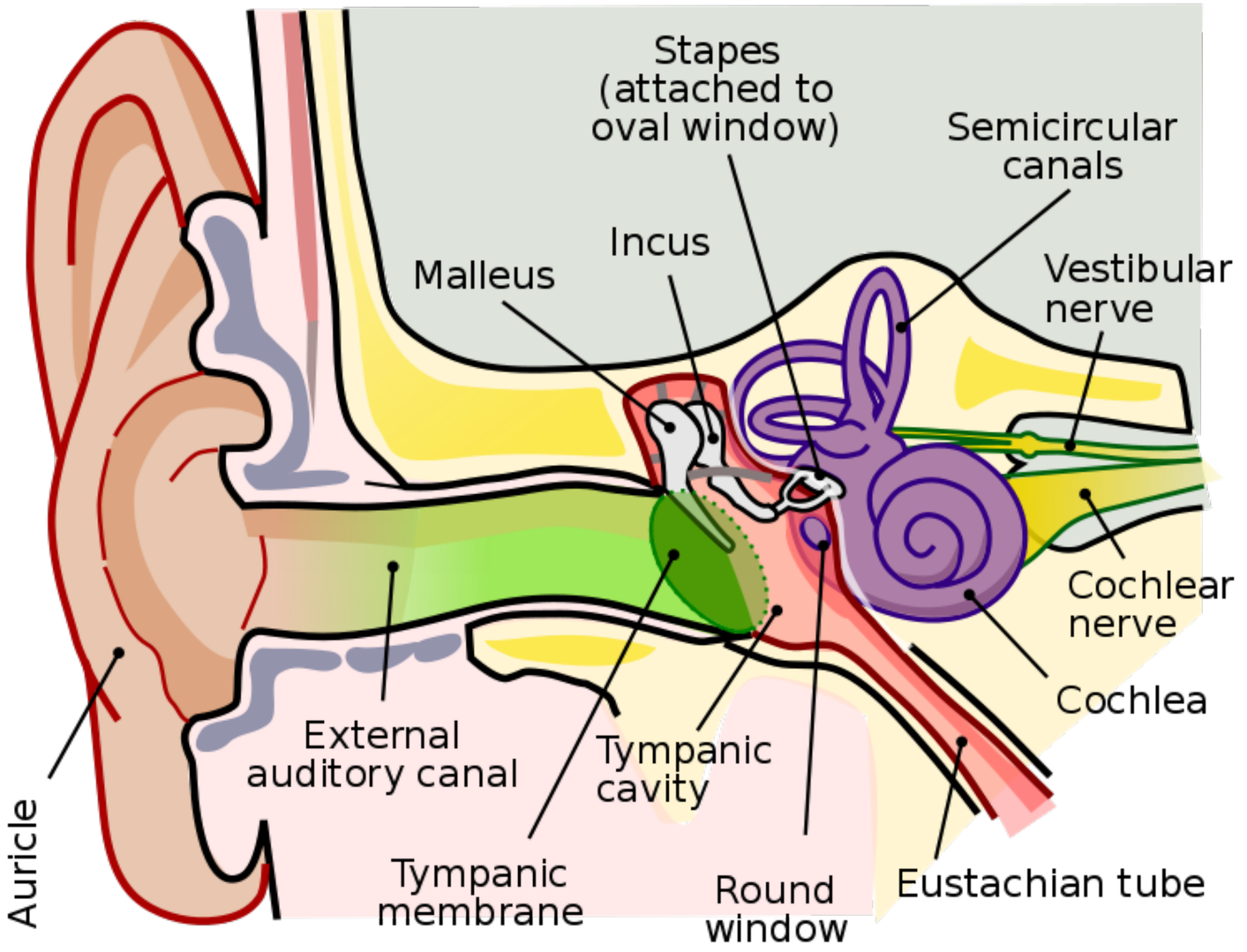
“C++ is there to deal with hardware at a low level, and to abstract away from it with zero overhead.”

– Bjarne Stroustrup

Sound waves



How we hear

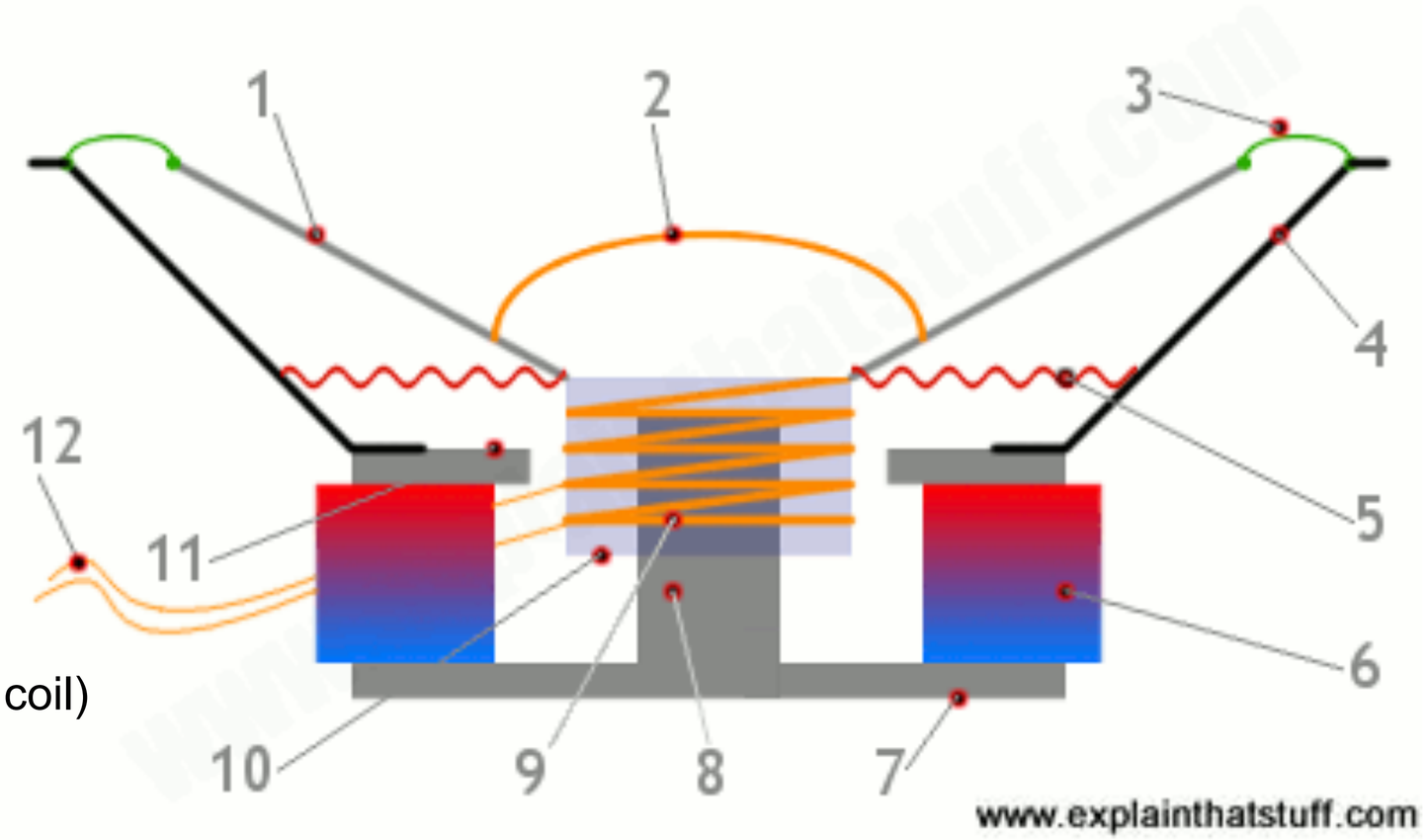


A diagram of the anatomy of the human ear.
15 February 2009
Lars Chittka; Axel Brockmann

Microphones

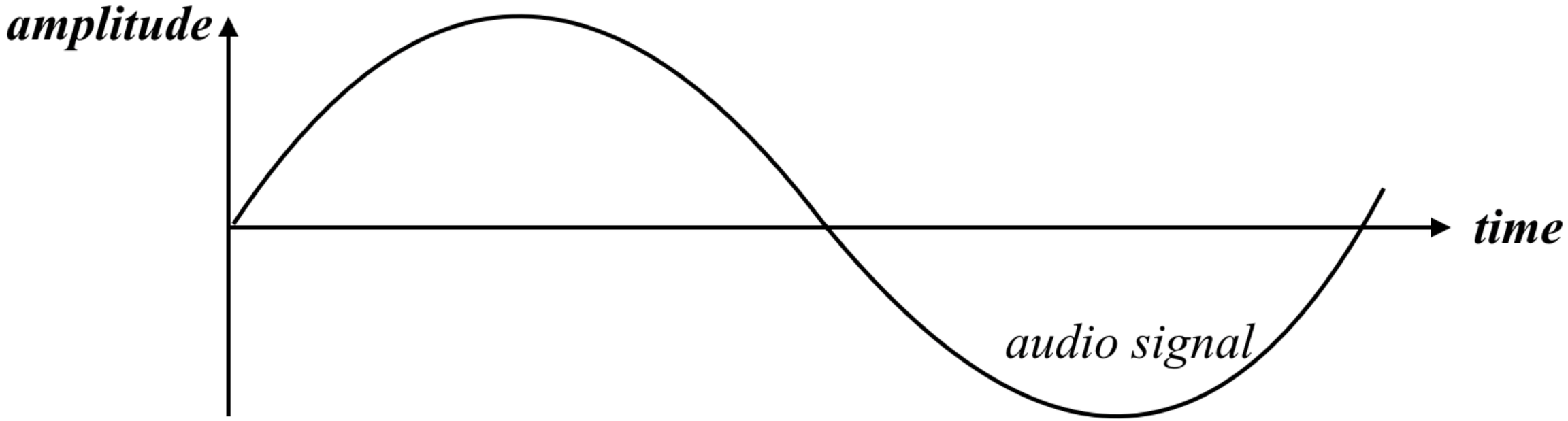
Speakers

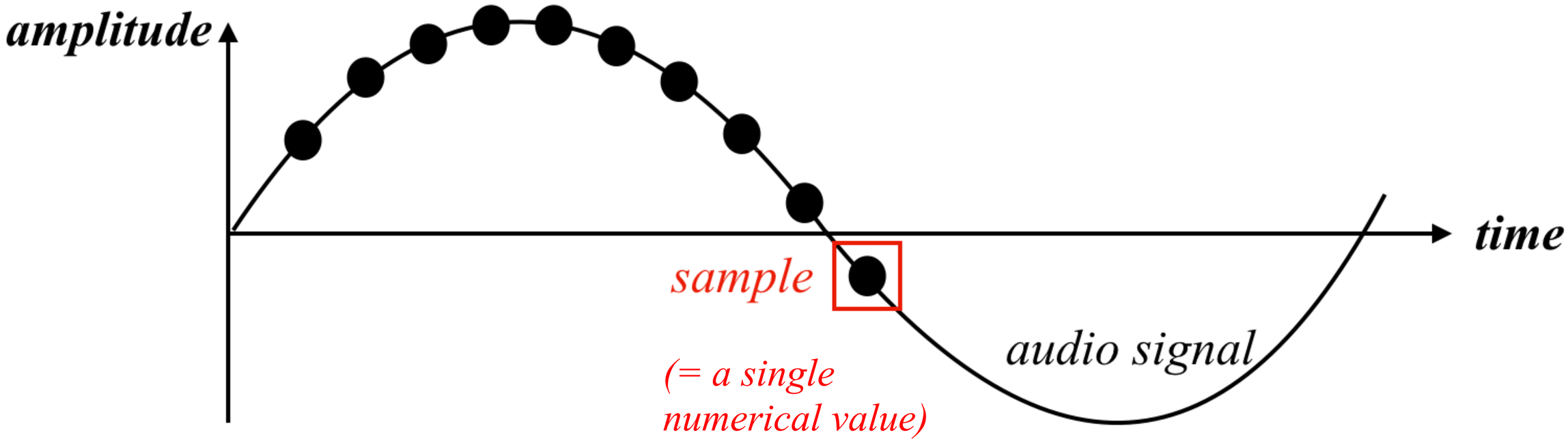
1. Diaphragm
   (moves in and out to push air and make sound)
2. Dust cap
3. Surround
   (flexibly fastens the diaphragm to the basket)
4. Basket (outer frame)
5. Spider (suspension)
6. Magnet
7. Bottom plate
8. Pole piece
   (concentrates magnetic field produced by voice coil)
9. Voice coil
(moves the diaphragm back and forth)
10. Former
    (cylinder onto which coil is wound)
11. Top plate
12. Cables
    (connect stereo amplifier unit to voice coil)



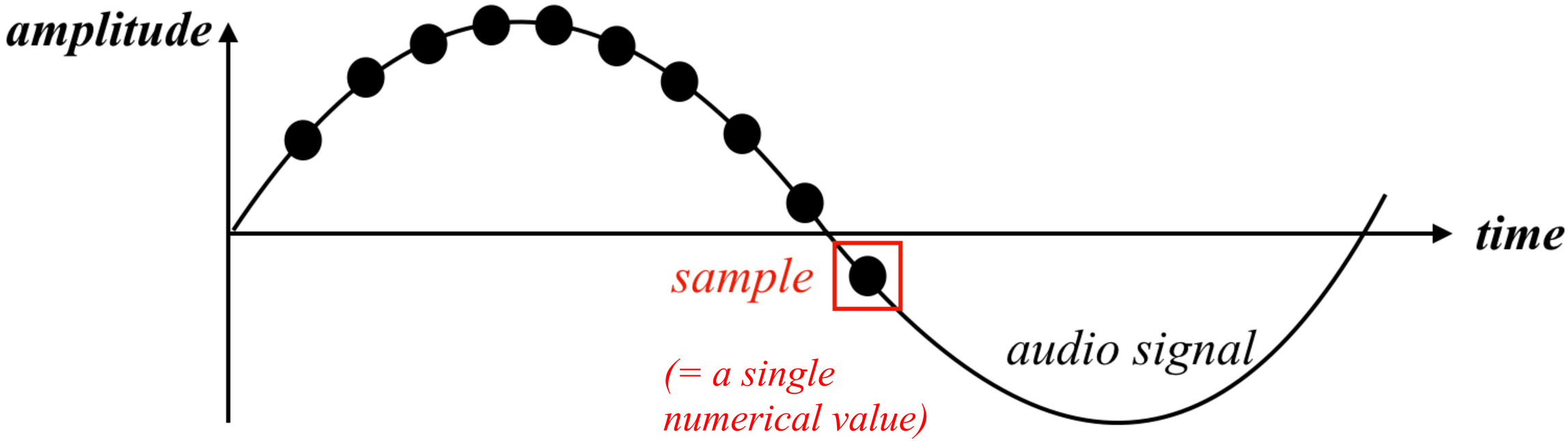
"Loudspeakers"
by Chris Woodford, 2018.

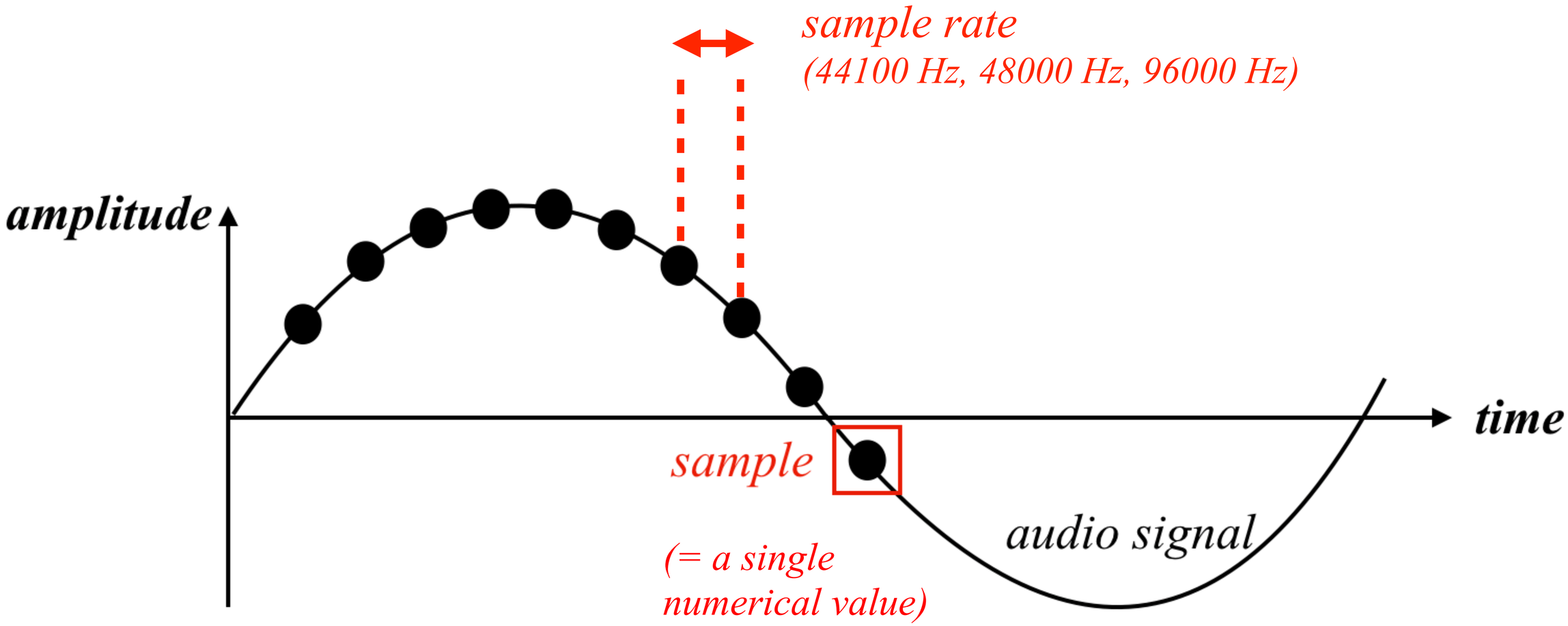
Representing audio data

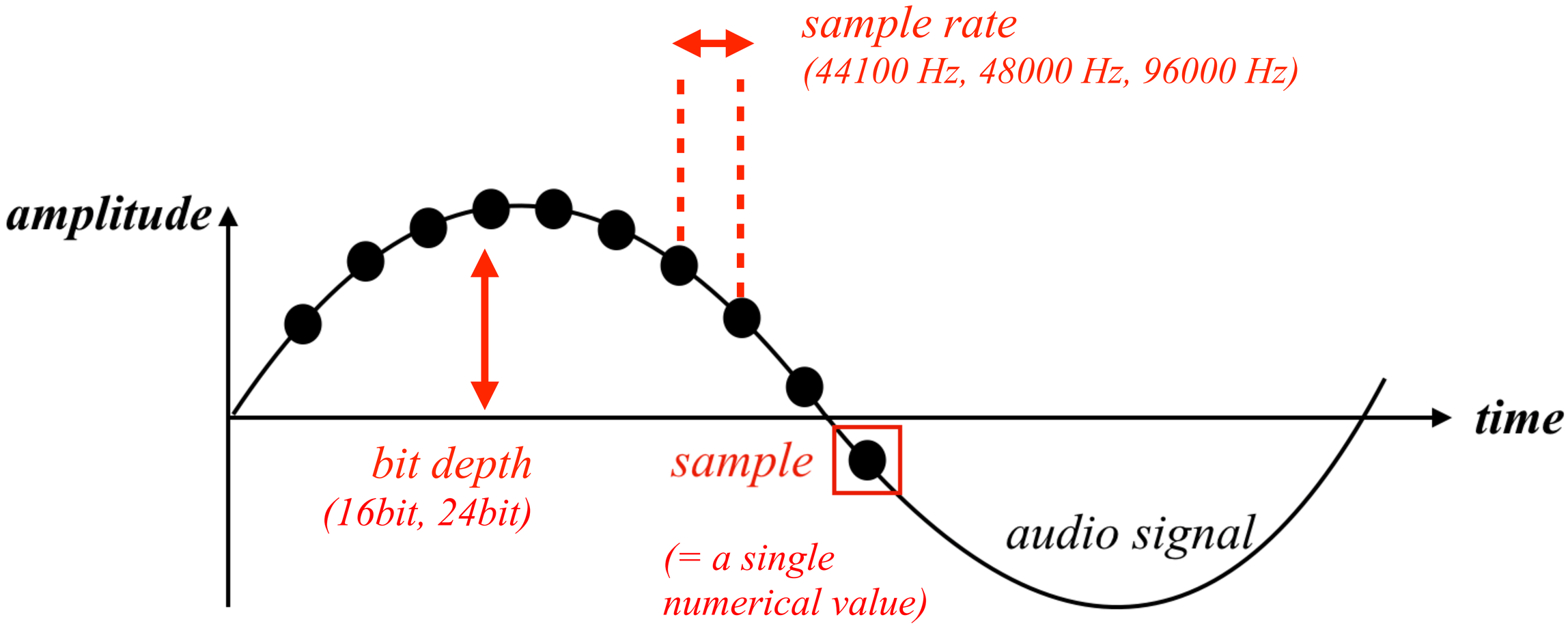

amplitude
time
audio signal


amplitude
time
sample
(= a single
numerical value)
audio signal

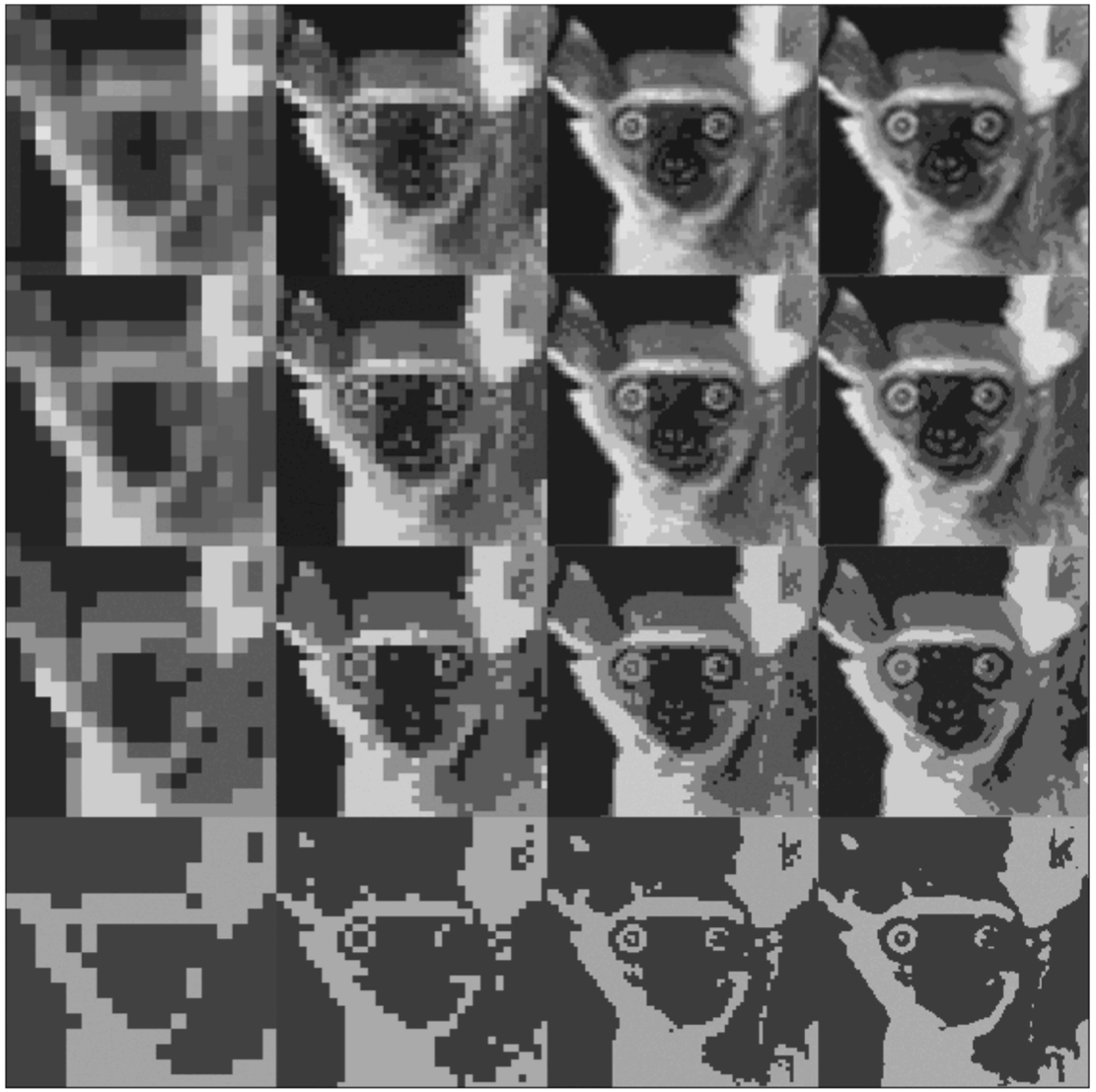
Linear Pulse Code Modulation (PCM)

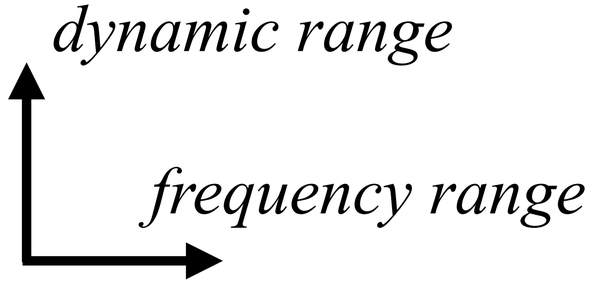



sample rate
(44100 Hz, 48000 Hz, 96000 Hz)
amplitude
time
sample
audio signal
(= a single
numerical value)


sample rate
(44100 Hz, 48000 Hz, 96000 Hz)
amplitude
time
bit depth
(16bit, 24bit)
sample
(= a single
numerical value)
audio signal

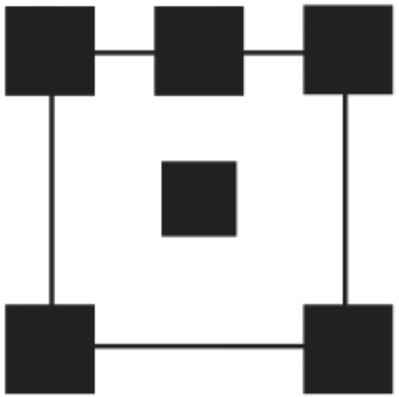
Quantisation noise





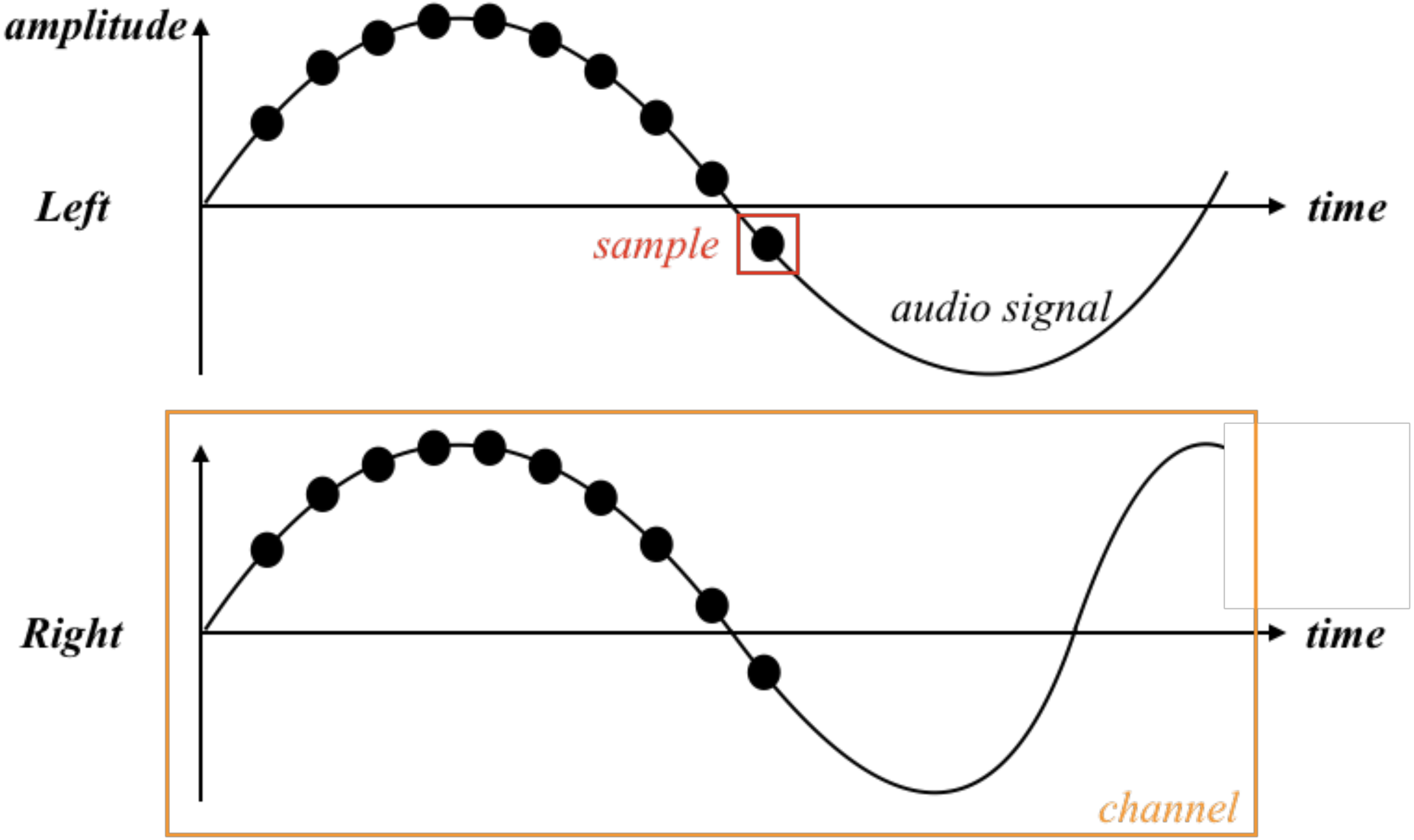
From Chris Adamson & Kevin Avila: “Learning Core Audio”

Channels

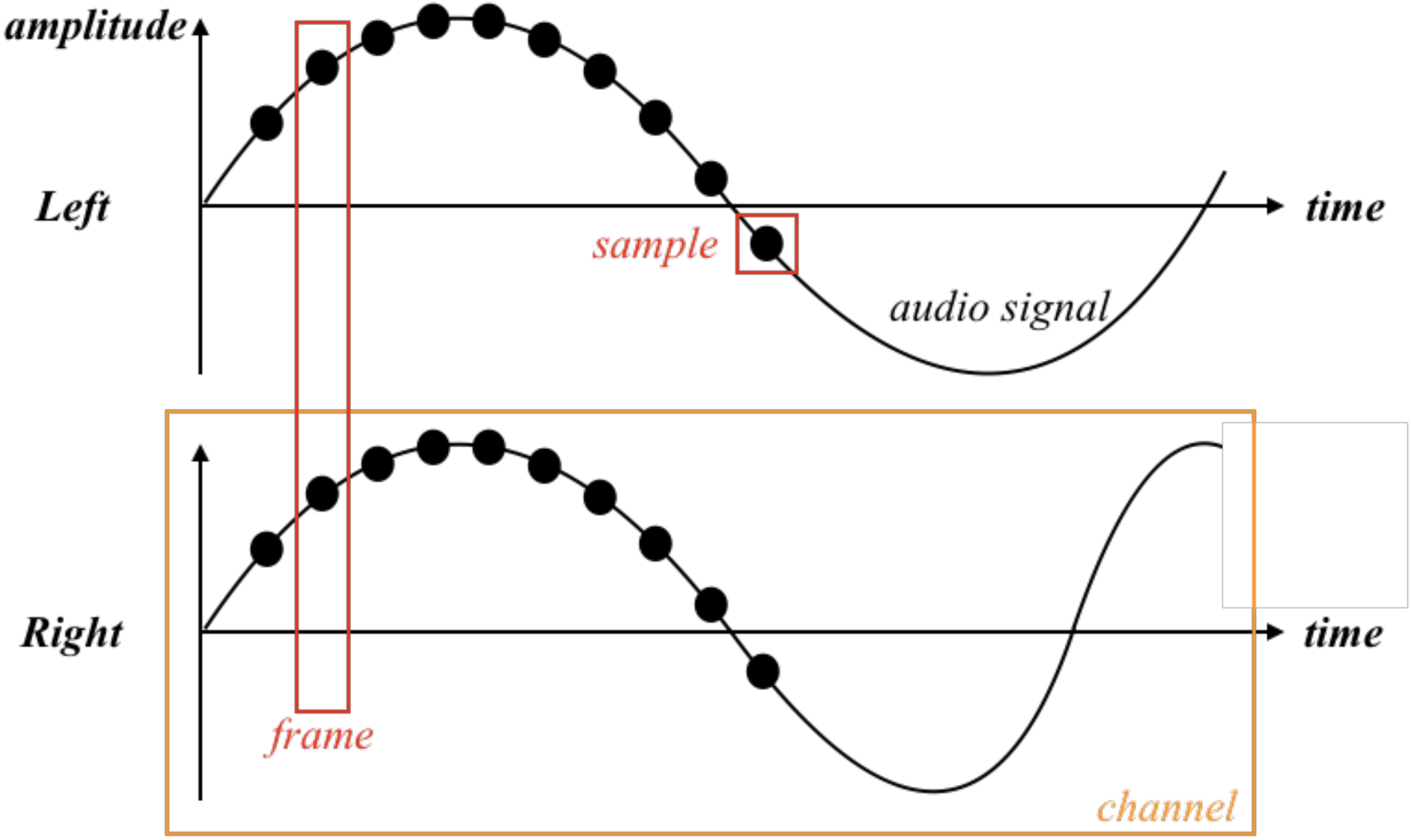


5.1 SURROUND SOUND

Channels


amplitude
Left
time
sample
audio signal
Right
time
channel

Channels


amplitude
Left
time
sample
audio signal
Right
time
frame
channel

Channels

- Channels have names

  (e.g. “left”, “right”, “surround left”, “low frequency emitter (LFE)”)

- There are different channel layouts

  (e.g. 6 output channels can be 3 x stereo or 5.1 surround)

- Channel order not portable

  (e.g. left-right-centre vs. left-centre-right)

Audio devices

Audio devices

- Input-only (e.g. built-in microphone)
- Output-only (e.g. built-in speakers, headphone output)

Audio devices

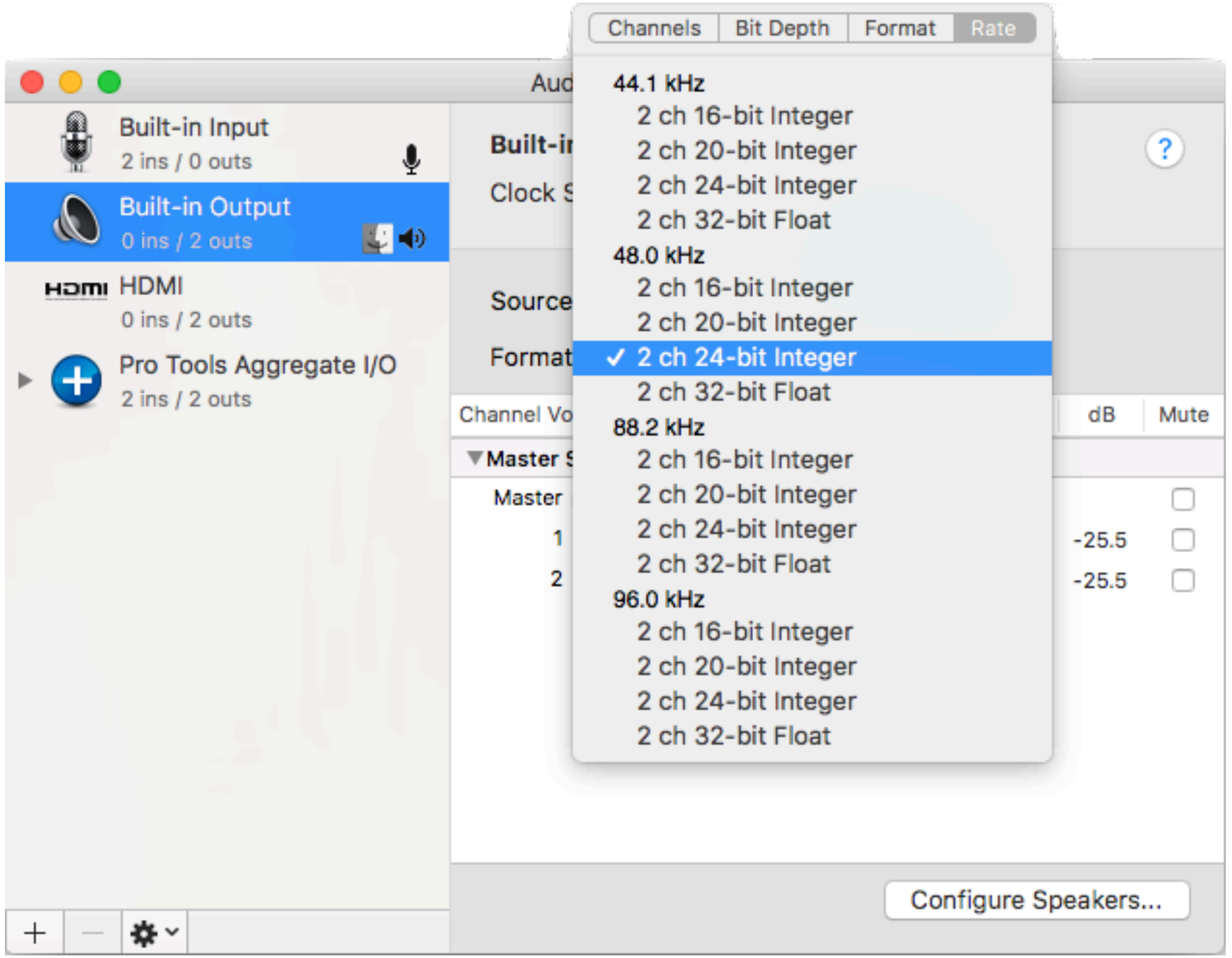
- Input-only (e.g. built-in microphone)
- Output-only (e.g. built-in speakers, headphone output)
- Input+Output
- Many-channel interfaces

Audio devices

- Sample type is device-specific!
  - Desktop hardware: typically 32-bit float between -1.0f and +1.0f
    (but also 24-bit packed, big vs. little endian, etc.)
  - Some phones: 16-bit integer
  - Embedded: 8-bit integer, fixed point, ...

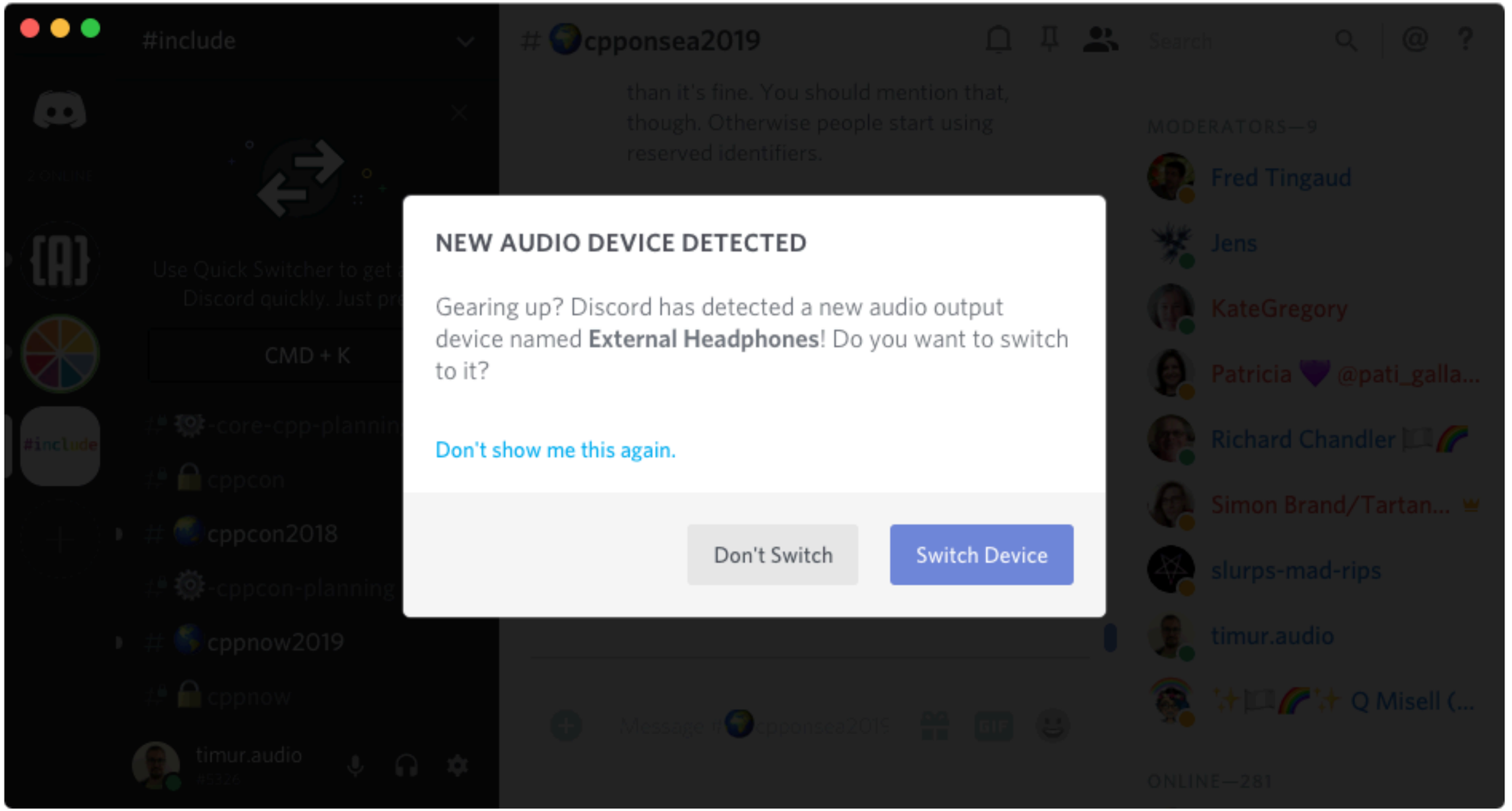
Audio devices

Sample type,
sample rate,
and channel layouts
can all change at runtime!



Audio devices

- There is typically a device selected as “default input” in the OS
- ...and one for “default output”
- An app might choose to use this device (or another one)
- The device configuration might change during runtime


#include
#cpponsea2019
than it's fine. You should mention that, though. Otherwise people start using reserved identifiers.
MODERATORS—9
Fred Tingaud
Jens
KateGregory
Patricia @pati_galla...
Richard Chandler
Simon Brand/Tartan...
slurps-mad-rips
timur.audio
Q Misell (...
ONLINE—281
CMD + K
cppcon
cppcon2018
cppnow2019
cppnow
timur.audio
NEW AUDIO DEVICE DETECTED
Gearing up? Discord has detected a new audio output device named External Headphones! Do you want to switch to it?
Don't show me this again.
Don't Switch
Switch Device

Audio devices

- There is typically a device selected as “default input” in the OS
- ...and one for “default output”
- An app might choose to use this device (or another one)
- The device configuration might change during runtime
- OS setting for global volume/pan
- OS-level mixer

Audio devices

- There is typically a device selected as “default input” in the OS
- ...and one for “default output”
- An app might choose to use this device (or another one)
- The device configuration might change during runtime
- OS setting for global volume/pan
- OS-level mixer
- “Shared mode” vs. “Exclusive mode”

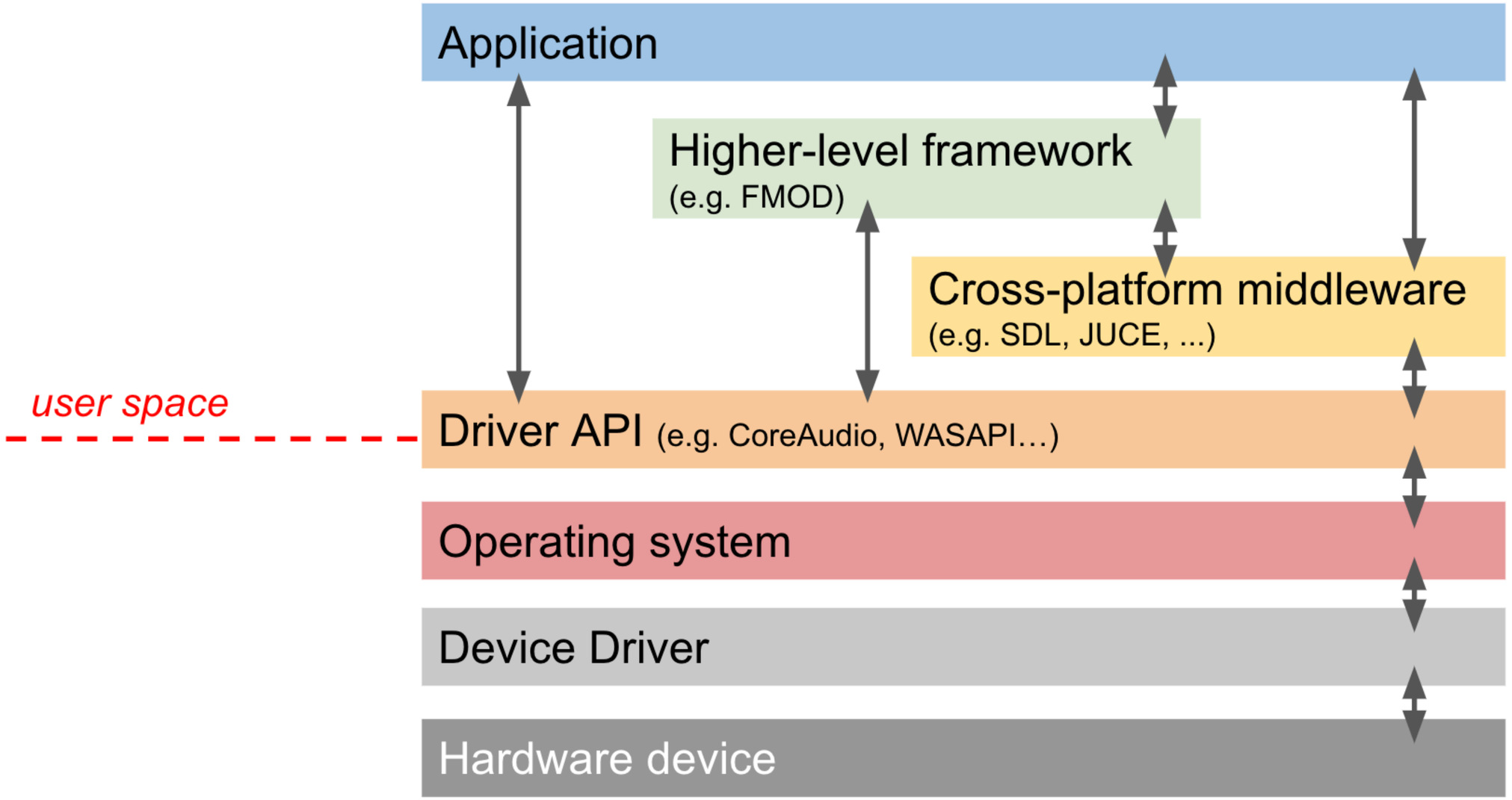
Audio devices

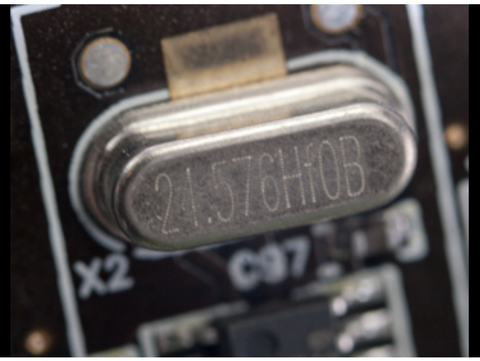
- There is typically a device selected as “default input” in the OS
- ...and one for “default output”
- An app might choose to use this device (or another one)
- The device configuration might change during runtime
- OS setting for global volume/pan
- OS-level mixer
- “Shared mode” vs. “Exclusive mode”
- Phones: “audio session”, “audio focus”

Audio devices

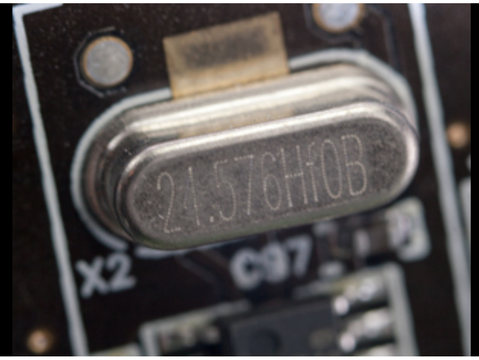
- There is typically a device selected as “default input” in the OS
- ...and one for “default output”
- An app might choose to use this device (or another one)
- The device configuration might change during runtime
- OS setting for global volume/pan
- OS-level mixer
- “Shared mode” vs. “Exclusive mode”
- Phones: “audio sessions”, “audio focus”
- Policies: “allow app to access microphone?”

Audio I/O


Application
Higher-level framework
(e.g. FMOD)
Cross-platform middleware
(e.g. SDL, JUCE, ...)
user space
Driver API (e.g. CoreAudio, WASAPI…)
Operating system
Device Driver
Hardware device



The audio callback



The audio callback

...

one audio buffer

one audio buffer

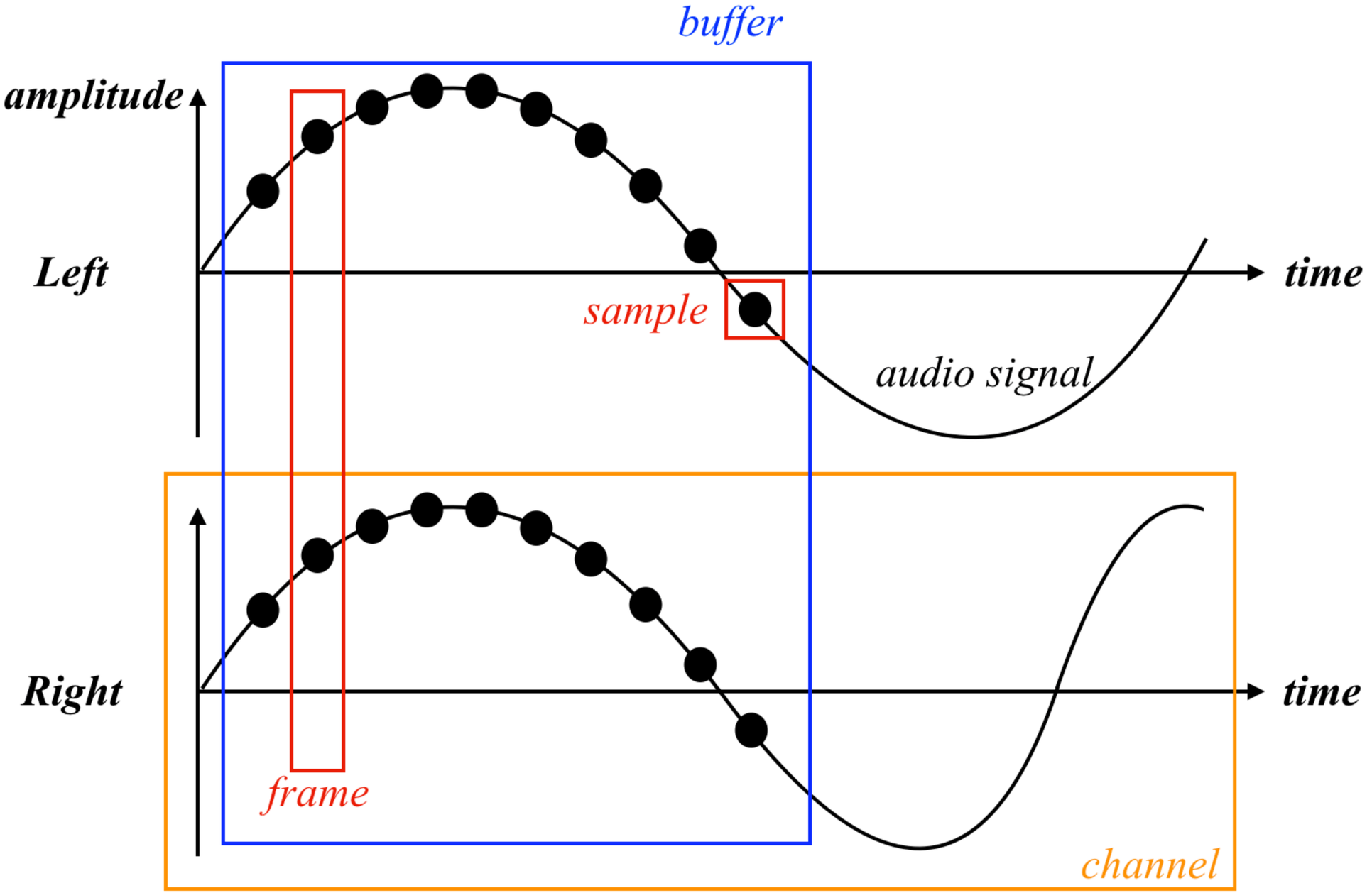
one audio buffer

...

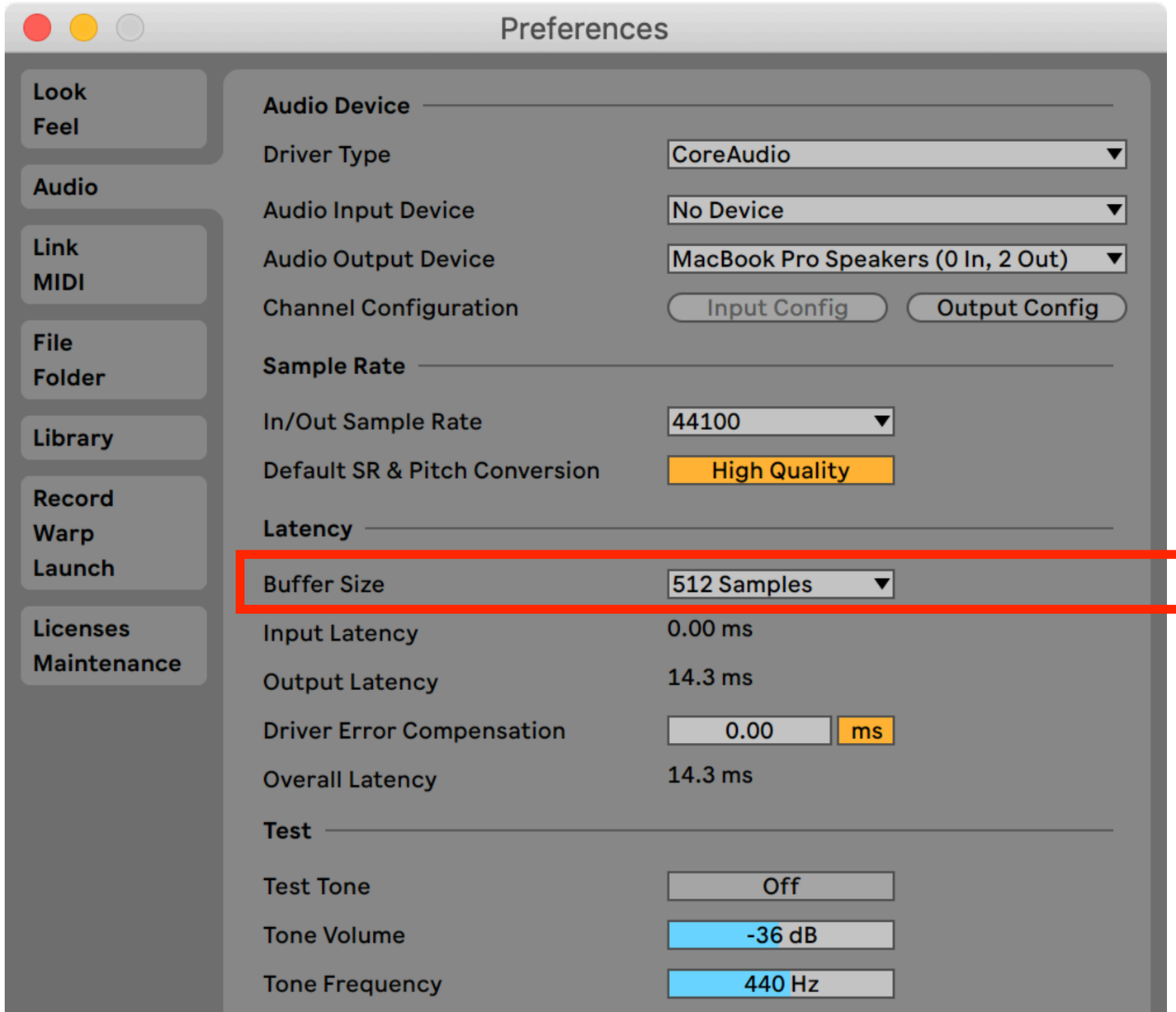
amplitude

0 1 2 3 4 5

time [ms]


buffer
amplitude
Left
time
sample
audio signal
Right
time
frame
channel

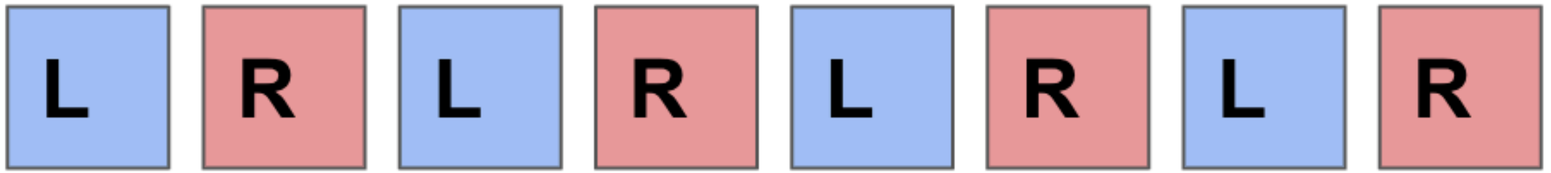
Buffer size



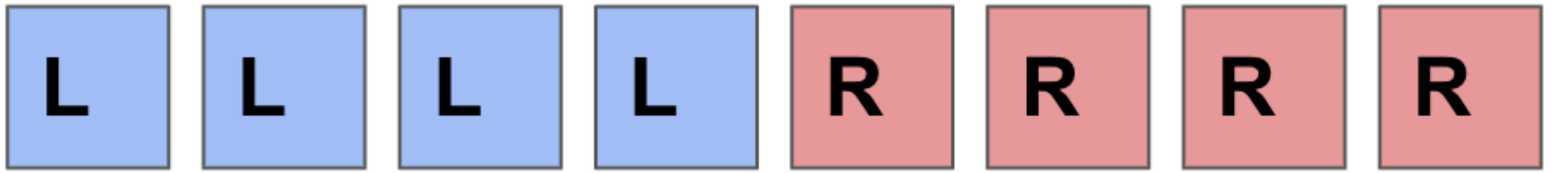
Tradeoff between latency and performance

Buffer order

Interleaved:

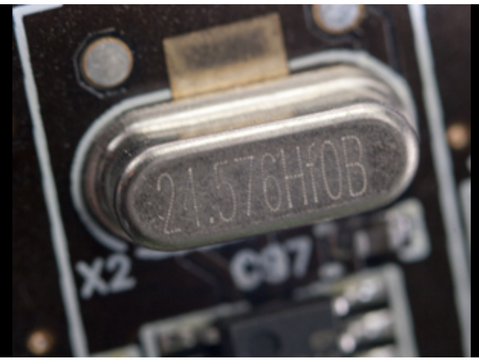


Deinterleaved:

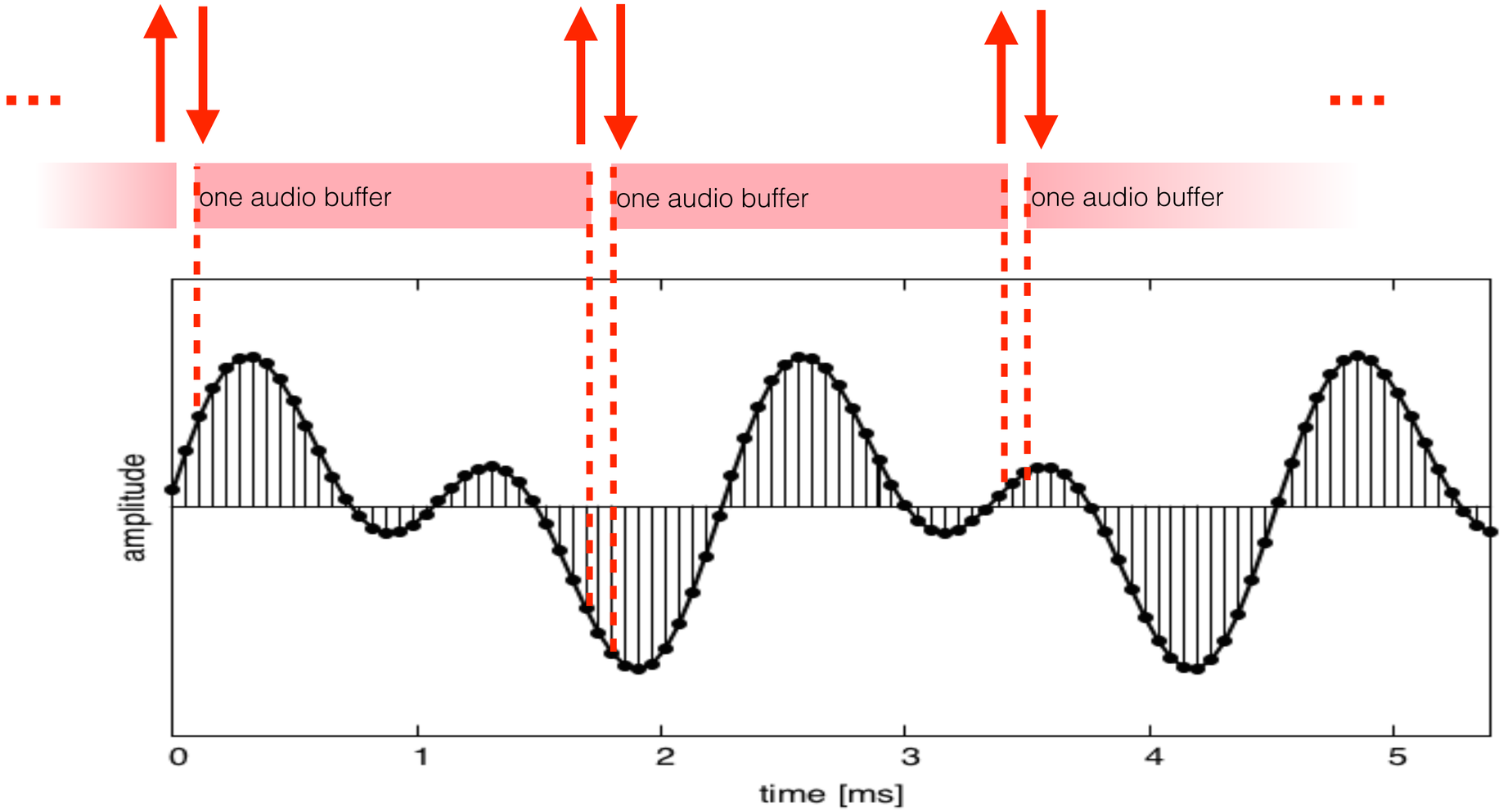


```
void audioCallback(float** inputChannelData,
                   int numInputChannels,
                   float** outputChannelData,
                   int numOutputChannels,
                   int numFrames)
{
  // processing...









}
```

```
float frequency_hz = 440.0;
float delta = 2.0 * M_PI * frequency_hz / sample_rate;
float phase = 0;

void audioCallback(float** inputChannelData,
                   int numInputChannels,
                   float** outputChannelData,
                   int numOutputChannels,
                   int numFrames)
{
  for (int i = 0; i < numFrames; ++i)
  {
    float next_sample = std::sin(phase);
    phase = std::fmod(phase + delta);
    for (int j = 0; j < numOutputChannels; ++j)
    {
      outputChannelData[i][j] = next_sample;
    }
  }
}
```



The audio callback



Real time!

- The audio callback is typically “near real-time”
- ~ 1ms to successfully read/write data from/to audio buffer
- Even a single missing sample causes an audio dropout → audible glitch!

Real time!

- The audio callback is typically “near real-time”
- ~ 1ms to successfully read/write data from/to audio buffer
- Even a single missing sample causes an audio dropout → audible glitch!
    - You cannot fail

Real time!

- The audio callback is typically “near real-time”
- ~ 1ms to successfully read/write data from/to audio buffer
- Even a single missing sample causes an audio dropout → audible glitch!
  - You cannot fail
- Processing happens on high-priority thread

Real time!

- The audio callback is typically “near real-time”
- ~ 1ms to successfully read/write data from/to audio buffer
- Even a single missing sample causes an audio dropout → audible glitch!
  - You cannot fail
- Processing happens on high-priority thread
  - Can’t use mutexes
  - Can’t allocate memory
  - Can’t do file or network I/O
  - Can’t call code with non-deterministic runtime
  - Lock-free thread synchronisation is crucial
  - `std::atomic` and lock-free queues are essential tools!

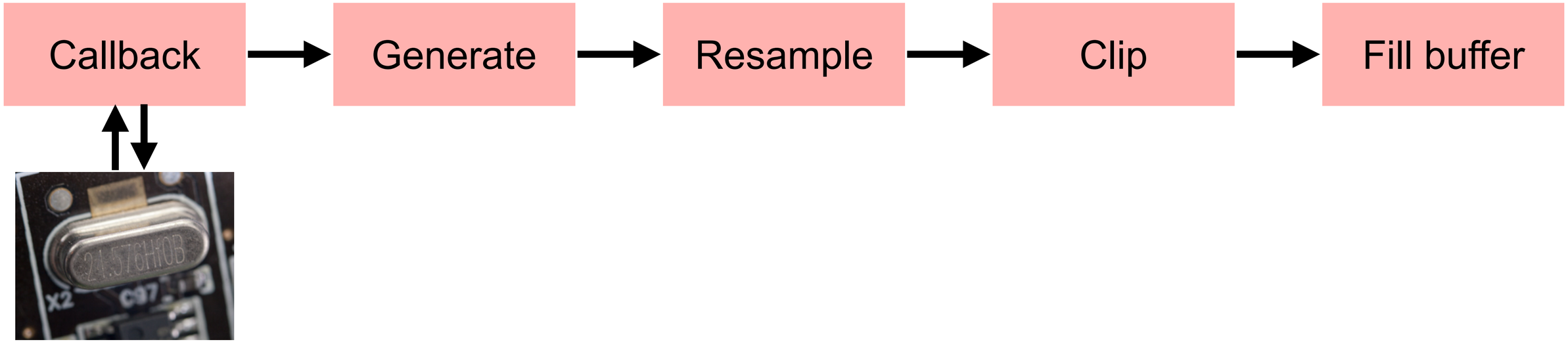
Higher level functionality

Higher level functionality

```
play(“beep.wav”);
```

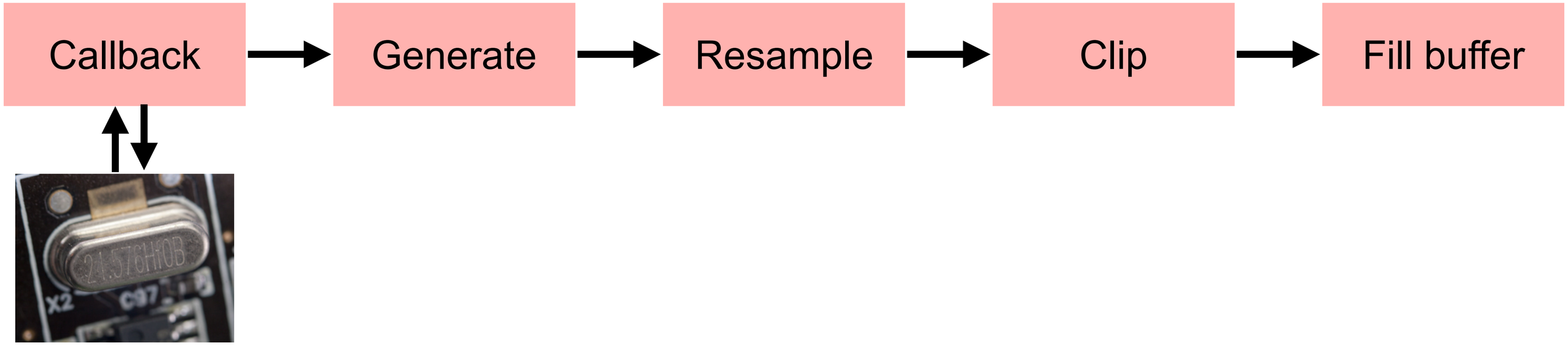
Higher level functionality

Audio processing thread:



Higher level functionality

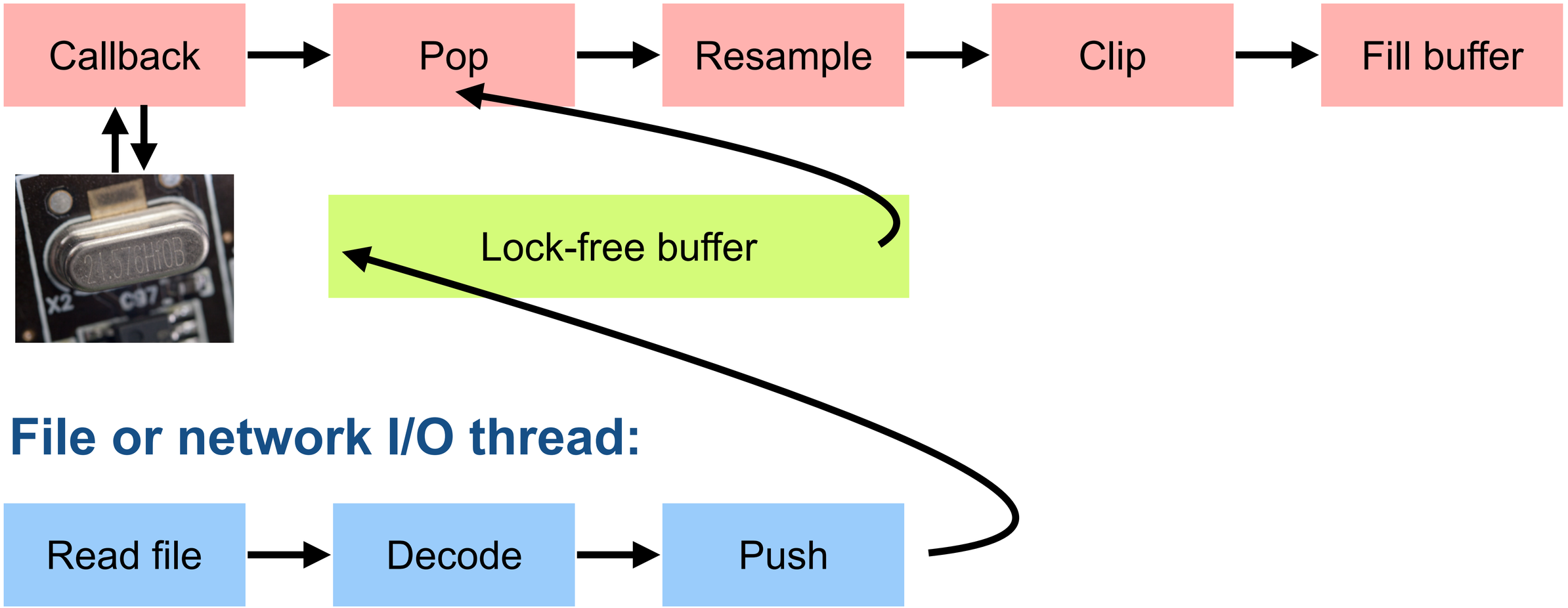
Audio processing thread:



File or network I/O thread:

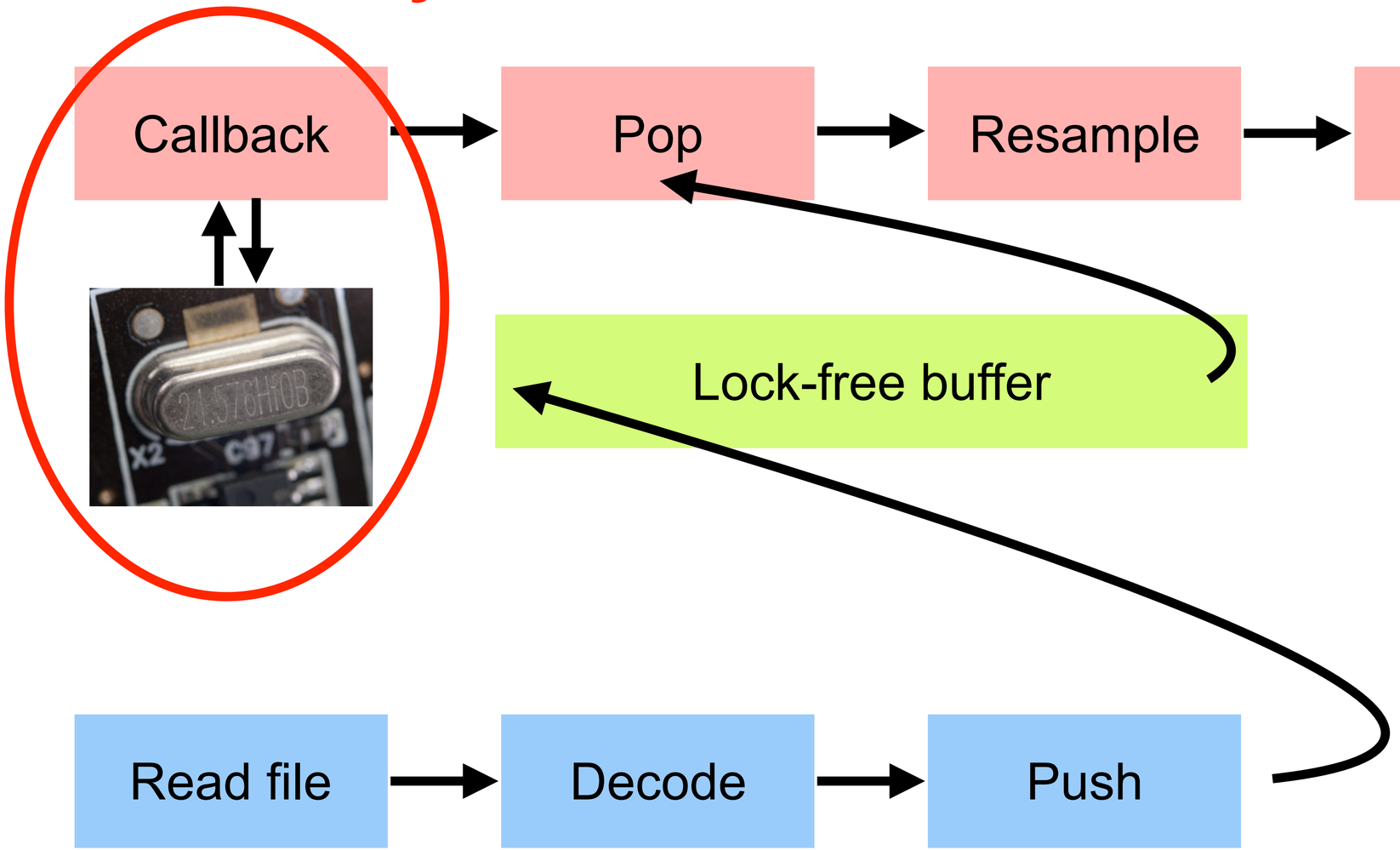
Higher level functionality

Audio processing thread:



File or network I/O thread:

This is not portably doable in C++ today!



What belongs in the standard library?

- Standardise existing practice!
  - Things whose design will not change in a decade.
  - No ongoing research and development.
  - Implementation experience.
  - Usage experience.

What belongs in the standard library?

- Standardise existing practice!
  - Things whose design will not change in a decade.
  - No ongoing research and development.
  - Implementation experience.
  - Usage experience.
- Standardise a clear need. Either:
  - Widely applicable use (std::vector),
  - Encapsulates nonportability (std::filesystem),
  - Difficult to implement correctly (std::shared_ptr), or
  - Requires language support (<type_traits>)

What belongs in the standard library?

- Standardise existing practice!
  - Things whose design will not change in a decade.
  - No ongoing research and development.
  - Implementation experience.
  - Usage experience.
- Standardise a clear need. Either:
  - Widely applicable use (std::vector),
  - Encapsulates nonportability (std::filesystem),
  - Difficult to implement correctly (std::shared_ptr), or
  - Requires language support (<type_traits>)

What belongs in the standard library?

- Standardise existing practice!
  - Things whose design will not change in a decade.
  - No ongoing research and development.
  - Implementation experience.
  - Usage experience.
- Standardise a clear need. Either:
  - Widely applicable use (std::vector),
  - Encapsulates nonportability (std::filesystem),
  - Difficult to implement correctly (std::shared_ptr), or
  - Requires language support (<type_traits>)

“C++ is there to deal with hardware at a low level, and to abstract away from it with zero overhead.”

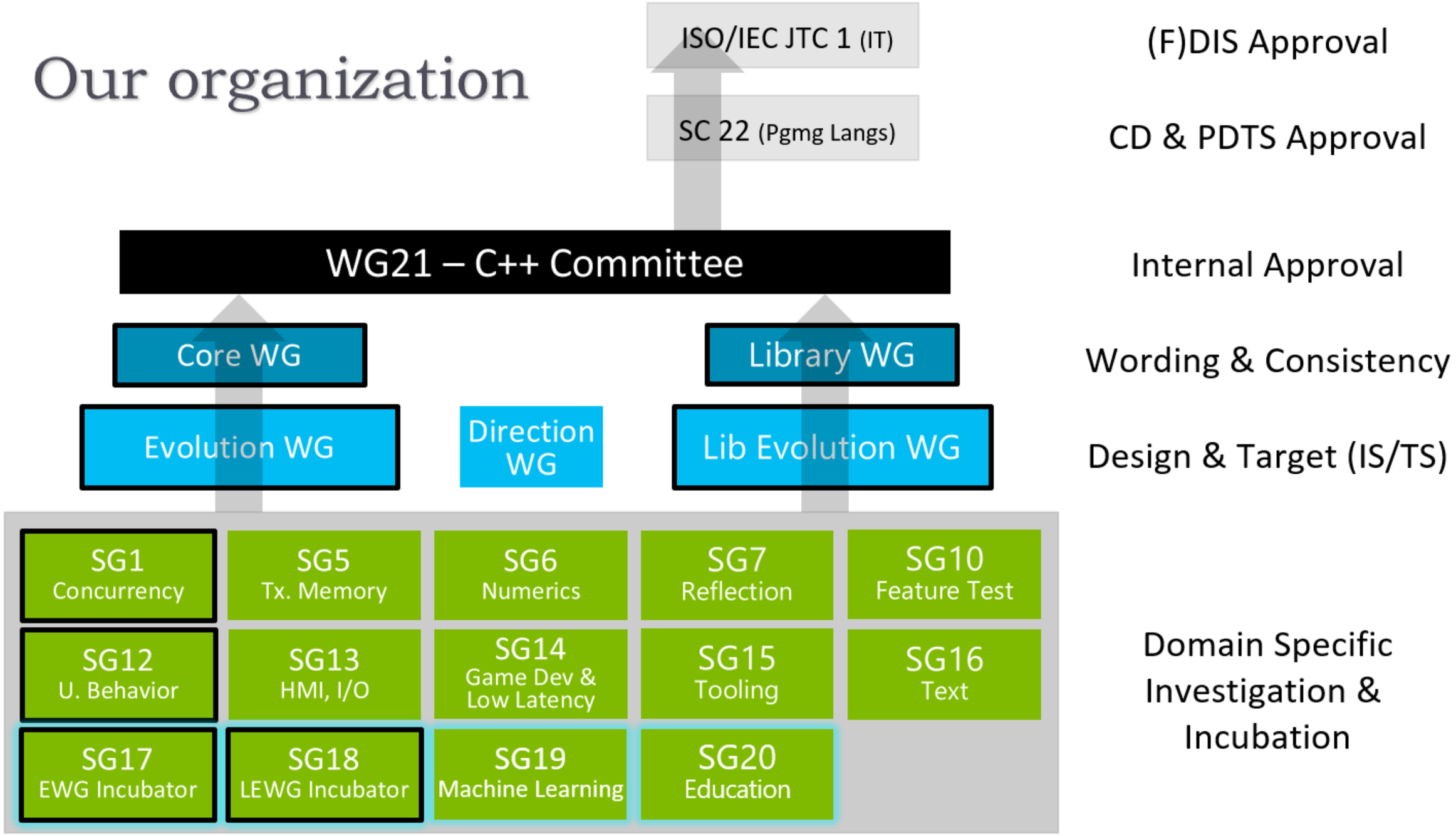
– Bjarne Stroustrup

std::audio

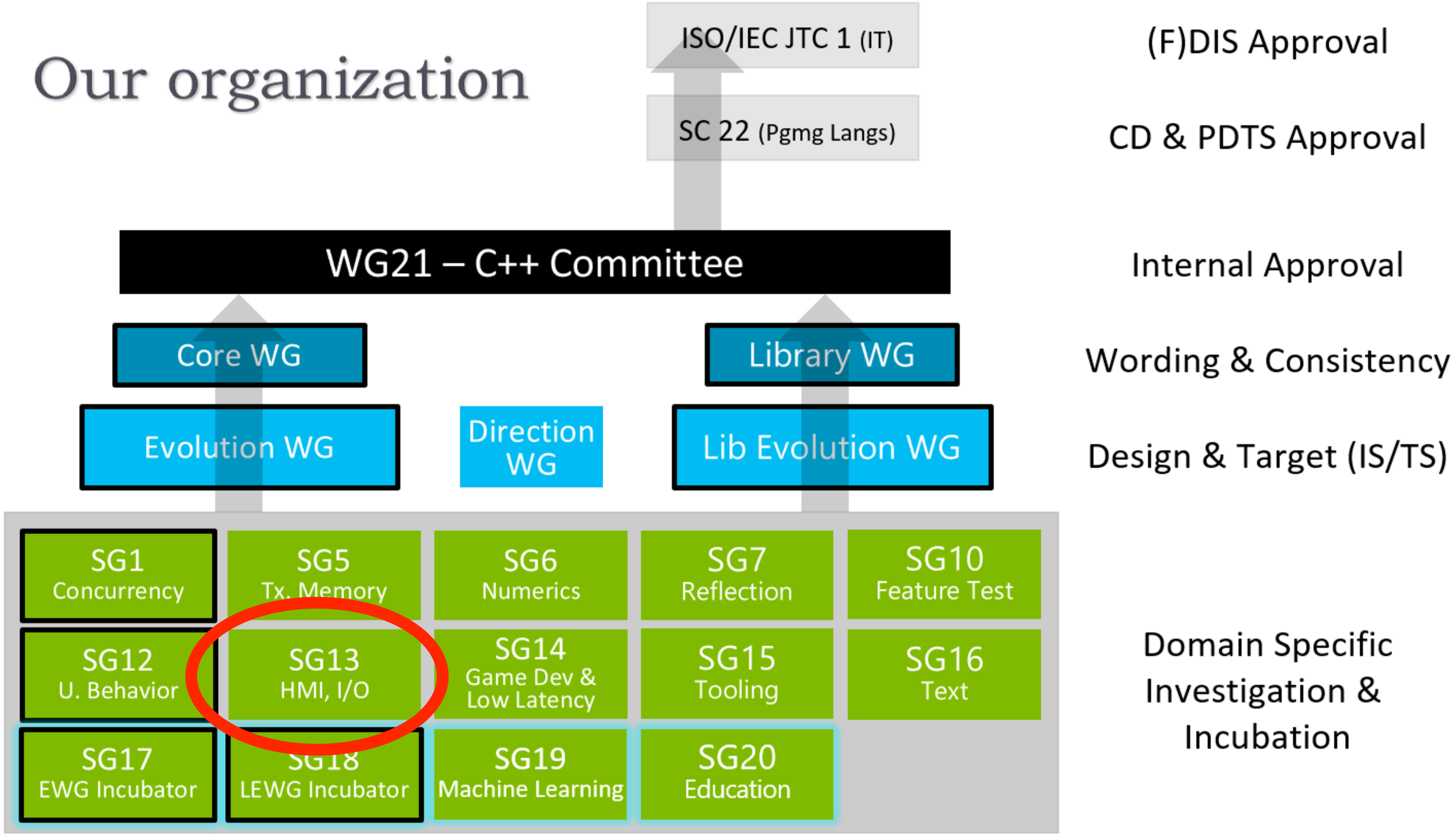
Audio Developer Conference (November 2018)

Guy Somberg, Timur Doumler, & Guy Davidson - Towards std::audio

Our organization



Our organization



WG21 Meeting, San Diego (November 2018)

A Standard Audio API for C++: Motivation, Scope, and Basic Design

Guy Somberg (guy@gameaudioprogrammer.com)
Guy Davidson (guy@hatcat.com)
Timur Doumler (papers@timur.audio)

Document #: P1386R0
Date: 2019-01-21
Project: Programming Language C++
Audience: SG13

WG21 Meeting, Kona, Hawaii (February 2019)

A Standard Audio API for C++: Motivation, Scope, and Basic Design

Guy Somberg (guy@gameaudioprogrammer.com)
Guy Davidson (guy@hatcat.com)
Timur Doumler (papers@timur.audio)

Document #: P1386R1
Date: 2019-03-11
Project: Programming Language C++
Audience: SG13, LEWG

Upcoming WG21 Meeting, Cologne, July 2019

P1386**R2**

(work in progress)

Reference implementation:

github.com/stdcpp-audio/libstdaudio

```
audio_buffer
audio_device
audio_device_io
audio_device_list
```

```
template <typename SampleType>
struct audio_buffer
{
    using index_type = size_t;

    index_type size_channels() const noexcept;
    index_type size_frames() const noexcept;
    index_type size_samples() const noexcept;
    index_type size_bytes() const noexcept;

    SampleType& operator()(index_type frame_index, index_type channel_index);
};
```

```
class audio_device
{
public:
    string_view name() const;
```

```cpp
class audio_device
{
public:
    string_view name() const;

    using device_id_t = /* implementation-defined */;
    device_id_t device_id() const noexcept;
```

```
class audio_device
{
public:
    string_view name() const;

    using device_id_t = /* implementation-defined */;
    device_id_t device_id() const noexcept;

    bool is_input() const noexcept;
    bool is_output() const noexcept;
    int get_num_input_channels() const noexcept;
    int get_num_output_channels() const noexcept;
```

```
class audio_device
{
public:
    using sample_rate_t = /* implementation-defined */;
    sample_rate_t get_sample_rate() const noexcept;
    span<sample_rate_t> get_supported_sample_rates() const noexcept;
    bool set_sample_rate(sample_rate_t);
```

```
class audio_device
{
public:
    using sample_rate_t = /* implementation-defined */;
    sample_rate_t get_sample_rate() const noexcept;
    span<sample_rate_t> get_supported_sample_rates() const noexcept;
    bool set_sample_rate(sample_rate_t);

    using buffer_size_t = /* implementation-defined */;
    buffer_size_t get_buffer_size_frames() const noexcept;
    span<buffer_size_t> get_supported_buffer_sizes_frames() const noexcept;
    bool set_buffer_size_frames(buffer_size_t);
```

```
class audio_device
{
public:
    bool start();  // R1 version
    bool stop();
    bool is_running() const noexcept;
```

```
class audio_device
{
public:
    bool start();  // R1 version – does not work: async callback needed!
    bool stop();
    bool is_running() const noexcept;
```

```
class audio_device
{
public:
    bool start(Callback&& started, Callback&& stopped);   // new in R2
    bool stop();
    bool is_running() const noexcept;

};
```

```cpp
using Callback = std::function<void(audio_device&)>;
```

```cpp
class audio_device
{
public:
    bool start(Callback&& started, Callback&& stopped);
    bool stop();
    bool is_running() const noexcept;

};
```

```cpp
using Callback = std::any_invocable<void(audio_device&)>;  // P0228



class audio_device
{
public:
    bool start(Callback&& started, Callback&& stopped);
    bool stop();
    bool is_running() const noexcept;

};
```

```
class audio_device
{
public:
    template <typename F1, typename F2>
    void start(F1&& started, F2&& stopped);

    bool stop();
    bool is_running() const noexcept;

};
```

```
template <typename F>
concept AudioDeviceCallback = std::is_invocable_v<F, audio_device&>;
```

```
class audio_device
{
public:
    template <typename F1, typename F2>
    requires AudioDeviceCallback<F1> && AudioDeviceCallback<F2>
    void start(F1&& started, F2&& stopped);

    bool stop();
    bool is_running() const noexcept;

};
```

```
template <typename SampleType>
class audio_device_io
{
public:
    optional<audio_buffer<SampleType>> input_buffer;
    optional<audio_buffer<SampleType>> output_buffer;
};
```

```
template <typename SampleType>
class audio_device_io
{
public:
    optional<audio_buffer<SampleType>> input_buffer;
    optional<chrono::time_point<audio_clock_t>> input_time;    // new in R2

    optional<audio_buffer<SampleType>> output_buffer;
    optional<chrono::time_point<audio_clock_t>> output_time;   // new in R2
};
```

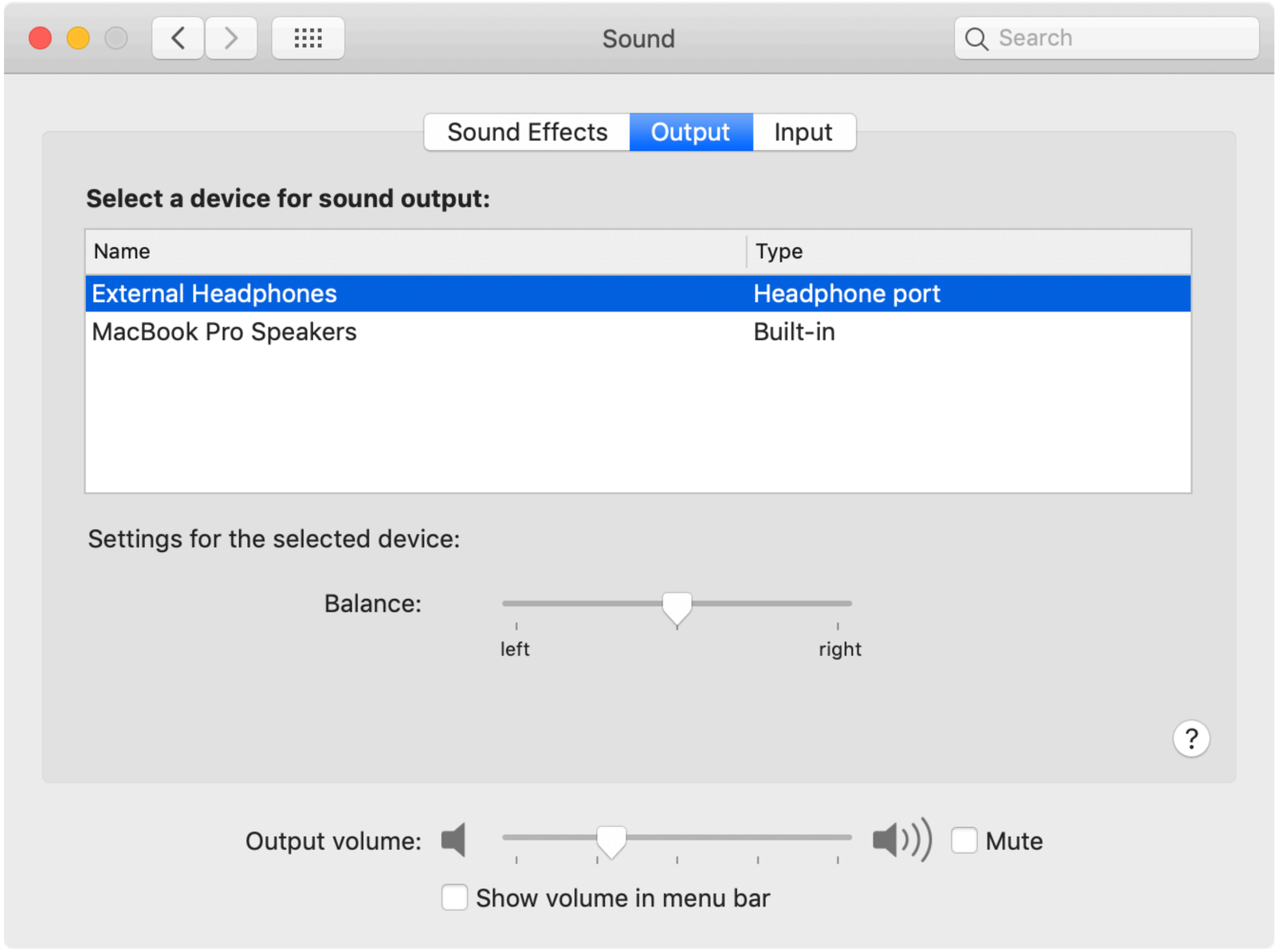
```
class audio_device
{
public:
    void connect(AudioIOCallback&& io_callback);
```

```
template <typename F>
concept AudioIOCallback = std::is_invocable_v<F, audio_device&, audio_device_io&>;

class audio_device
{
public:
    void connect(AudioIOCallback auto&& io_callback);
```

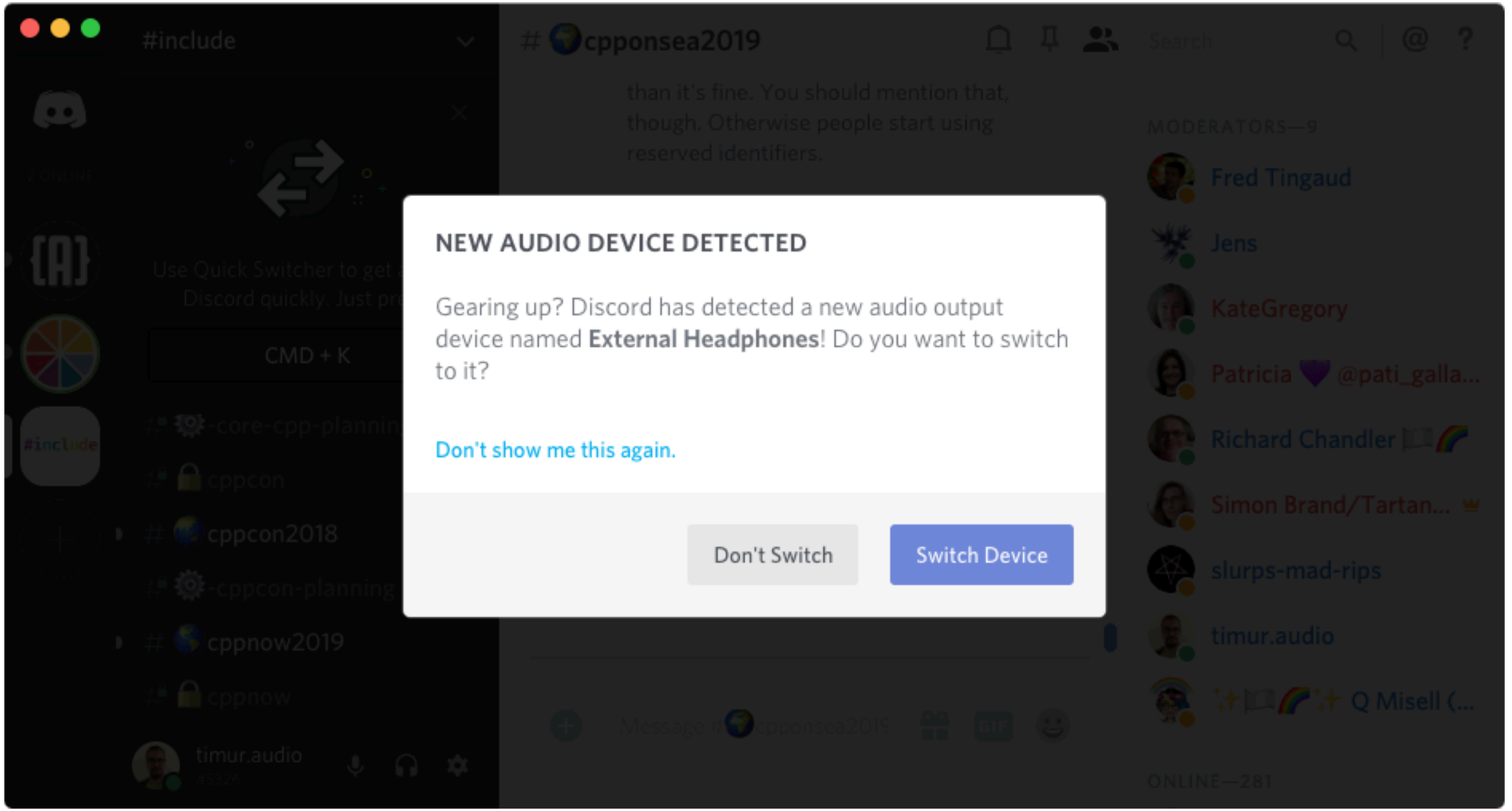
```cpp
template <typename F>
concept AudioIOCallback = std::is_invocable_v<F, audio_device&, audio_device_io&>;

class audio_device
{
public:
    // push-based processing:
    void connect(AudioIOCallback auto&& io_callback);

    // pull-based processing:
    void wait() const;
    void process(AudioIOCallback auto& io_callback);
```

```cpp
template <typename F>
concept AudioIOCallback = std::is_invocable_v<F, audio_device&, audio_device_io&>;

class audio_device
{
public:
    // push-based processing:
    void connect(AudioIOCallback auto&& io_callback);

    // pull-based processing:
    void wait() const;

    template<class Rep, class Period>
    void wait_for(std::chrono::duration<Rep, Period> rel_time) const;

    template<class Clock, class Duration>
    void wait_until(std::chrono::time_point<Clock, Duration> abs_time) const;

    void process(AudioIOCallback auto& io_callback);
```

```
class audio_device
{
public:
    // push-based processing:
    constexpr bool can_connect() const noexcept;

    void connect(AudioIOCallback auto&& io_callback);

    // pull-based processing:
    constexpr bool can_process() const noexcept;

    void wait() const;

    template<class Rep, class Period>
    void wait_for(std::chrono::duration<Rep, Period> rel_time) const;

    template<class Clock, class Duration>
    void wait_until(std::chrono::time_point<Clock, Duration> abs_time) const;

    void process(AudioIOCallback auto& io_callback);
```


Sound
Search
Sound Effects
Output
Input
Select a device for sound output:
Name
Type
External Headphones
Headphone port
MacBook Pro Speakers
Built-in
Settings for the selected device:
Balance:
left
right
?
Output volume:
Mute
Show volume in menu bar

```
optional<audio_device> get_default_audio_input_device();
optional<audio_device> get_default_audio_output_device();

audio_device_list get_audio_input_device_list();
audio_device_list get_audio_output_device_list();

class audio_device_list {
public:
    iterator begin();
    iterator end();
};
```


#include
#cpponsea2019
than it's fine. You should mention that, though. Otherwise people start using reserved identifiers.
Search
MODERATORS—9
Fred Tingaud
Jens
KateGregory
Patricia @pati_galla...
Richard Chandler
Simon Brand/Tartan...
slurps-mad-rips
timur.audio
Q Misell (...
ONLINE—281
CMD + K
cppcon
cppcon2018
cppnow2019
cppnow
timur.audio
NEW AUDIO DEVICE DETECTED
Gearing up? Discord has detected a new audio output device named External Headphones! Do you want to switch to it?
Don't show me this again.
Don't Switch
Switch Device

```
optional<audio_device> get_default_audio_input_device();
optional<audio_device> get_default_audio_output_device();

audio_device_list get_audio_input_device_list();
audio_device_list get_audio_output_device_list();

class audio_device_list {
public:
    iterator begin();
    iterator end();
};

// new in R2:

template <typename F>
concept AudioDeviceListCallback = std::is_invocable_v<F>;

void connect_audio_device_list_callback(AudioDeviceListCallback auto&& adlcb);
```

Live demo!

Audio buffer layout

Interleaved

Deinterleaved

Audio buffer layout

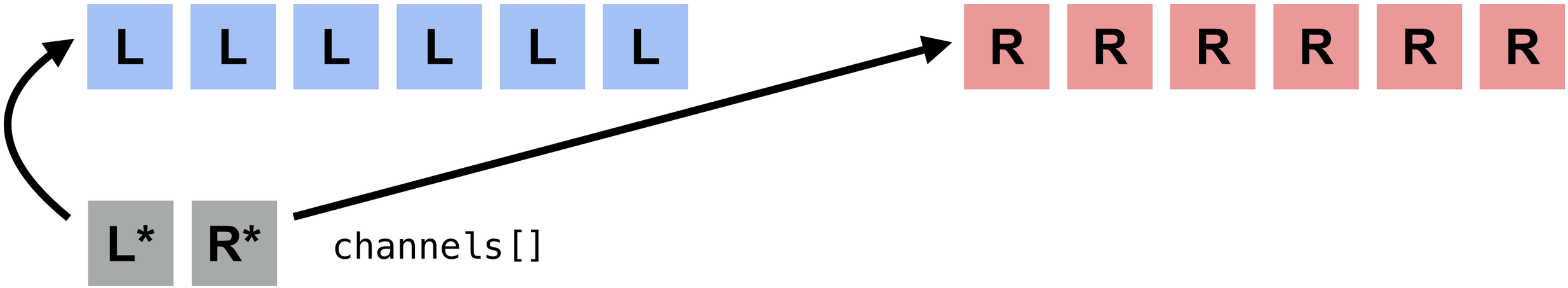
Interleaved

Deinterleaved

Audio buffer layout

Interleaved

Deinterleaved

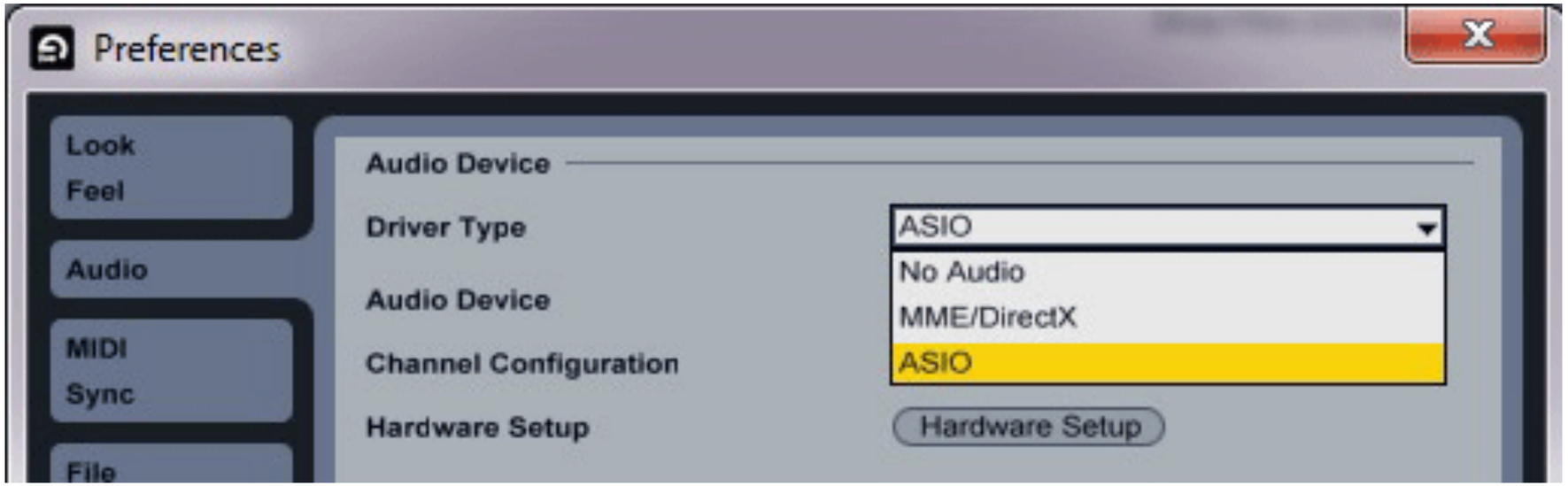


```cpp
template <typename SampleType>
struct audio_buffer
{
    using index_type = size_t;

    index_type size_channels() const noexcept;
    index_type size_frames() const noexcept;
    index_type size_samples() const noexcept;
    index_type size_bytes() const noexcept;

    SampleType& operator()(index_type frame_index, index_type channel_index);
};
```

```cpp
template <typename SampleType>
struct audio_buffer
{
    using index_type = size_t;

    index_type size_channels() const noexcept;
    index_type size_frames() const noexcept;
    index_type size_samples() const noexcept;
    index_type size_bytes() const noexcept;

    // R2: underlying layout is implementation-defined
    SampleType& operator()(index_type frame_index, index_type channel_index);
};
```

```cpp
template <typename SampleType>
struct audio_buffer
{
    using index_type = size_t;

    index_type size_channels() const noexcept;
    index_type size_frames() const noexcept;
    index_type size_samples() const noexcept;
    index_type size_bytes() const noexcept;

    // R2: underlying layout is implementation-defined!
    SampleType& operator()(index_type frame_index, index_type channel_index);

private:
    // R2: user cannot construct an audio_buffer!
    audio_buffer(...);
};
```

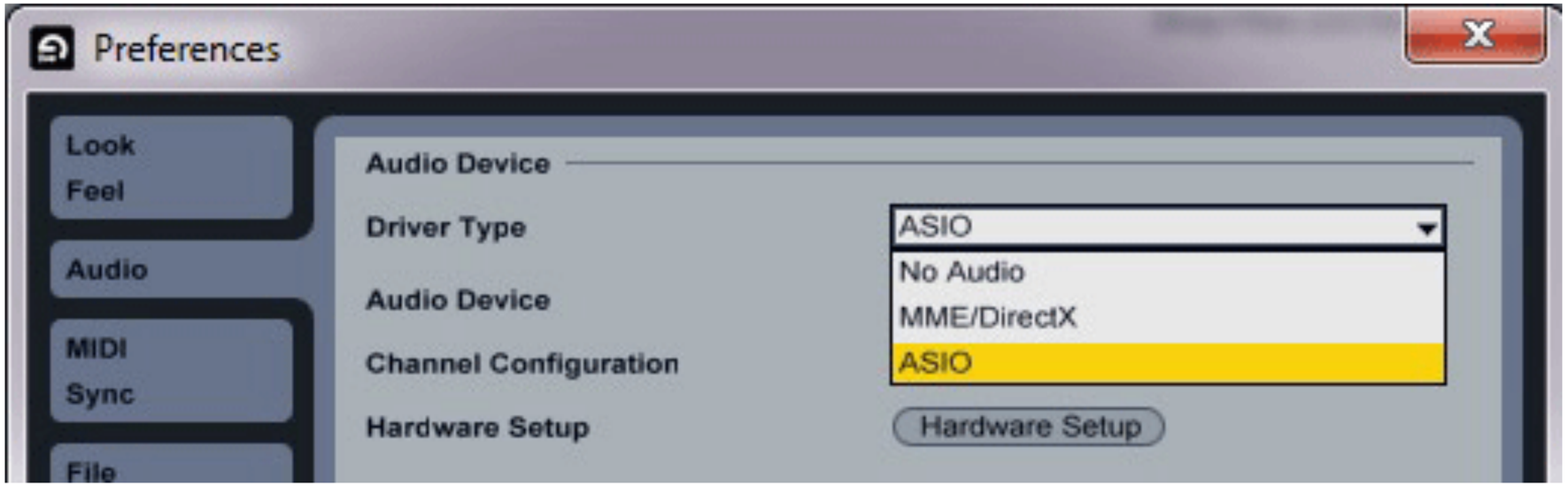
Problem: different driver types



Problem: different driver types

- There exist platforms with more than one native driver API

  e.g. Windows: WASAPI and ASIO
- Pro audio people want to use both in the same app!
- “null driver” is sometimes useful



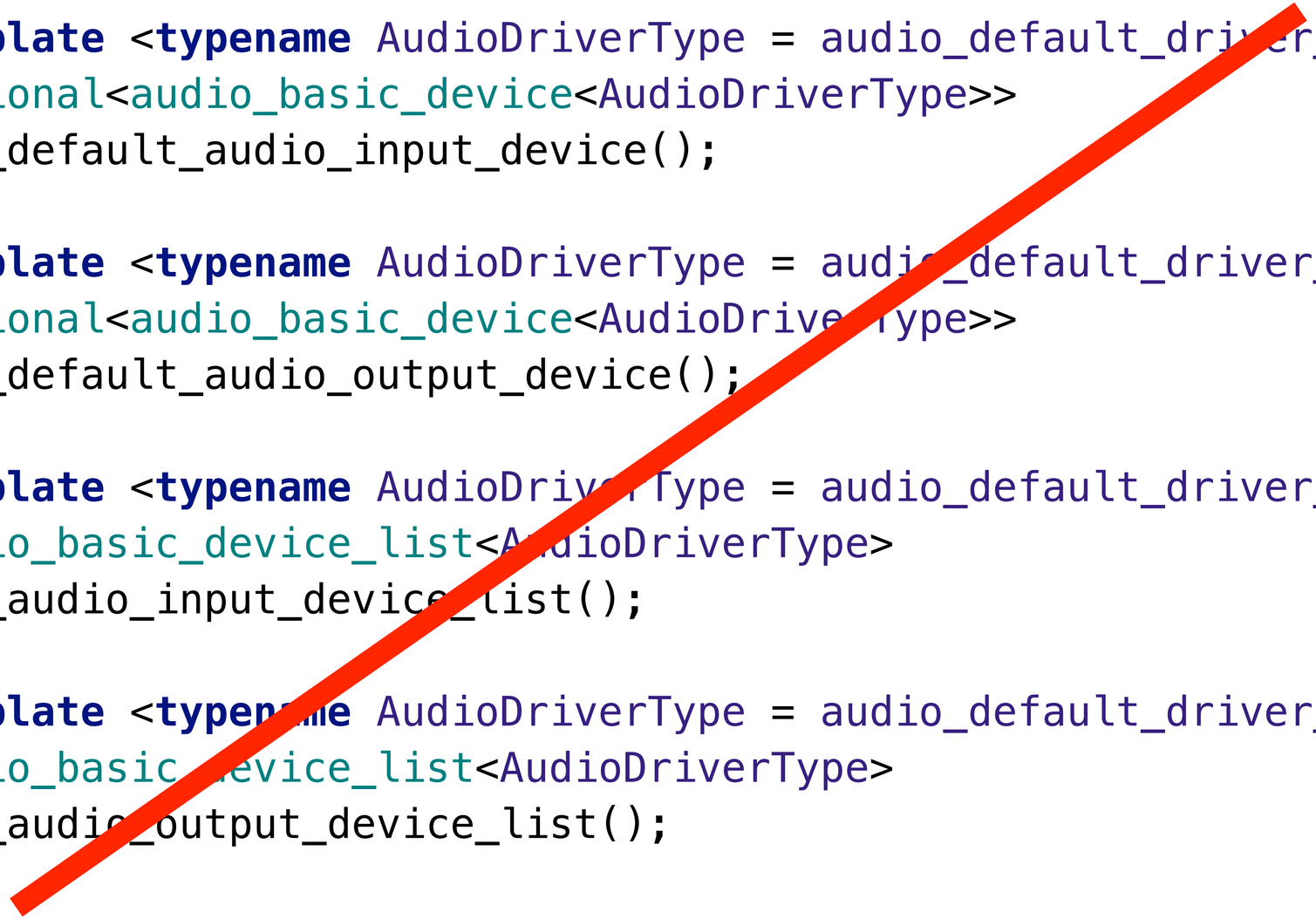
```cpp
// Always provided:
struct audio_null_driver_t {};

// Implementation-defined:
struct __wasapi_driver_t {};
struct __asio_driver_t {};

// Always defined:
using audio_default_driver_t = /* Implementation-defined */ ;
```

```cpp
template <typename AudioDriverType>
class audio_basic_device
{
  // ...
}

using audio_device = audio_basic_device<audio_default_driver_t>;
```

```
template <typename AudioDriverType>
class audio_basic_device
{
  // ...
}

using audio_device = audio_basic_device<audio_default_driver_t>;

template <typename AudioDriverType>
class audio_basic_device_list {
public:
    iterator begin();
    iterator end();
};

using audio_device_list = audio_basic_device_list<audio_default_driver_t>;
```

```
template <typename AudioDriverType = audio_default_driver_t>
optional<audio_basic_device<AudioDriverType>>
get_default_audio_input_device();

template <typename AudioDriverType = audio_default_driver_t>
optional<audio_basic_device<AudioDriverType>>
get_default_audio_output_device();

template <typename AudioDriverType = audio_default_driver_t>
audio_basic_device_list<AudioDriverType>
get_audio_input_device_list();

template <typename AudioDriverType = audio_default_driver_t>
audio_basic_device_list<AudioDriverType>
get_audio_output_device_list();
```

```
template <typename AudioDriverType = audio_default_driver_t>
optional<audio_basic_device<AudioDriverType>>
get_default_audio_input_device();

template <typename AudioDriverType = audio_default_driver_t>
optional<audio_basic_device<AudioDriverType>>
get_default_audio_output_device();

template <typename AudioDriverType = audio_default_driver_t>
audio_basic_device_list<AudioDriverType>
get_audio_input_device_list();

template <typename AudioDriverType = audio_default_driver_t>
audio_basic_device_list<AudioDriverType>
get_audio_output_device_list();
```



- TO DO:
  - Channel names + channel layouts

- TO DO:
  - Channel names + channel layouts
  - Error handling: `std::error_code`?

- TO DO:
  - Channel names + channel layouts
  - Error handling: `std::error_code`?

- Open questions:
  - Coroutines?

- TO DO:
  - Channel names + channel layouts
  - Error handling: `std::error_code`?

- Open questions:
  - Coroutines?
  - “Shared mode” vs. “Exclusive mode”?

- TO DO:
  - Channel names + channel layouts
  - Error handling: `std::error_code`?
- Open questions:
  - Coroutines?
  - “Shared mode” vs. “Exclusive mode”?
- Out of scope:
  - Mobile-specific: “audio sessions”, policies
  - Proprietary spatial audio formats (e.g. Dolby Atmos)
  - File I/O
  - Audio algorithms

Ship vehicle?

- Audio TS?
- HMI (2D Graphics + audio + user input) TS?
- C++23? C++26?
- Boost.Audio?
- Just another library?


Thank you!
@timur_audio
blog: https://timur.audio
includecpp.org