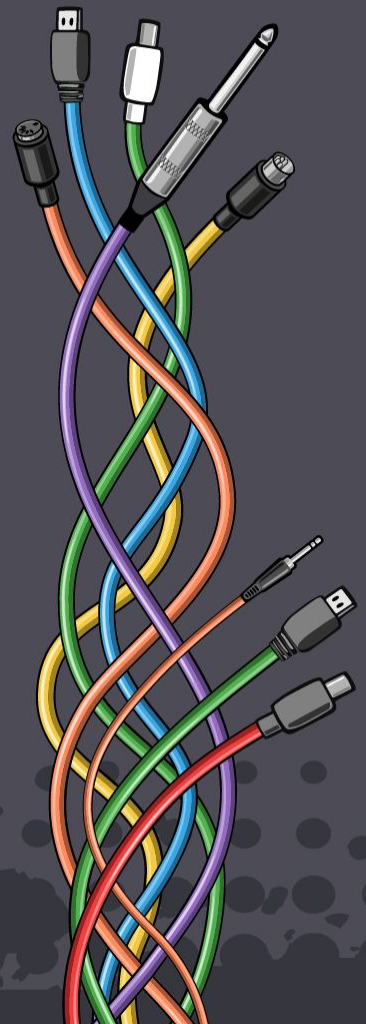


2.5 YEARS LATER

A C++ FRAMEWORK FOR AUDIO ML RESEARCH PROTOTYPING

MAXIME COUTANT

ADC25 Bristol

First things first

Who am I ?

- Maxime Coutant, software engineer

- 6 years of experience in the audio industry:
  - 3 years in a startup (Audionamix)
  - 2.5 years in a research lab (ADASP - Télécom Paris)
  - around a year on hobby and open source projects

This talk

- About creating working prototypes and binaries from research results
- Not about prototyping during the research process itself
- Not a ready to distribute framework for plugin development

The Audible Project

The Audible project

- Public R&D french project

- Gathered four entities
  - Greenwaves Technologies: RISC-V custom DSP processor
  - Orosound: Headset integrating the DSP processor
  - Télécom Paris: providing software to run on the DSP processor and other devices
  - Another research lab about EEG

- The goal was to provide a base common platform for these entities to license and improve on

The Audible project - ADASP role

- Our role was to provide a new generation of algorithms, ML based, adapted to the target platforms: embedded DSP processor, mobile and cloud

- The team was composed of
  - 4 PHDs
  - 2 Post-docs
  - 6 Engineers
  - 1 Lead researcher

The Audible project - my role

- Integrate the neural networks provided by the team into compiled libraries:
  - custom DSP processor
  - generic mobile processor
  - raspberry pi 0 and 4 for prototyping
  - linux cloud cpu

- Developed a C++ framework to implement such libraries

The Audible project - current state

- The project is mostly finished
- Not everything went as planned
- Some code will slowly deprecate on Télécom Paris’s servers

The framework

Goals & Features

The general concept

- Must allow to reproduce accurately research results

- Must allow for iterative design process
  - Research can evolve quickly

- Must supports multiple use cases
  - Rendering vs Classification
  - Possible need for distributed computation

- Cross compiling: linux, android, iOS

Reproducing research results

- Supports fp32 and fp64 precision
- Supports for custom processing
- Reproducible filter banks (ERB, MFC, custom)
  - Loads from Numpy
  - Configurable epsilon for log

Technical features

- Real-time compatible interface
- Async processing
  - Local processing
  - Remote processing (client/server)
- Multiple backends
  - FFT
  - Inference engine
- Basic encryption for NN weights

A note about research

- Research evolves => Features evolves

- Some research failures can heavily impacts the vision of the project

The framework

Software design

Let’s get started

- Chain of nodes connected by “wires”

- Generic data view
  - Up to 3 dimensions: frames, channels, frame size
  - Scalar, complex
  - Serves as the I/O types for the nodes, can be concatenated as tuples

- Base Node class, highly configurable

Let’s get started

- Wide variety of features, each configuration known at compile-time
  - C++ templates and constexpr

- Templated DSP chain, composed of nodes

```
template <typename... Nodes>
class Chain;

using MyDSPChain = Chain<FirstNode, SecondNode>;
```

- Templated base Node class

```
template <typename Derived, typename InputView, typename OutputView, bool NeedExternalWire>
class ProcessorNode;
```

Sync & Async Processing

Synchronous real-time processing: the **`Processor`** class

```
template <typename NodeChain>
class Processor;

using MyProcessor = Processor<Chain<FirstNode, SecondNode>>;
```

Real-time processing interface:

```
bool pause(bool pause) noexcept REALTIME;

void process(InputType input, OutputType output) noexcept REALTIME;
```

Asynchronous processing: new nodes

- **`TransmissionNode`**
  - Has no output, send data on a **`Connection`**
  - Can only be found at the end of a chain

- **`ReceptionNode`**
  - Has no input, gets its data from a **`Connection`**
  - Can only be found at the start of a chain

- **Connection**: abstraction over a connection between two DSP chains
  - Can be local: e.g. FIFO
  - Can be over the network: e.g. websockets

Asynchronous processing: the **`AsyncProcessor`** class

- Supports all three modes using template parameters and requires clauses
  - Local async mode
  - Client mode
  - Server mode

```
template <typename UserAsyncNodeChain,
          typename UserPreprocessingChain = nodes::EmptyChain>
class AsyncProcessor;
```

Asynchronous processing: Local async mode

- User provides:
  - An async chain with input and output (no transmission/reception nodes)
  - Optional: A prechain with input and output (no transmission/reception nodes)

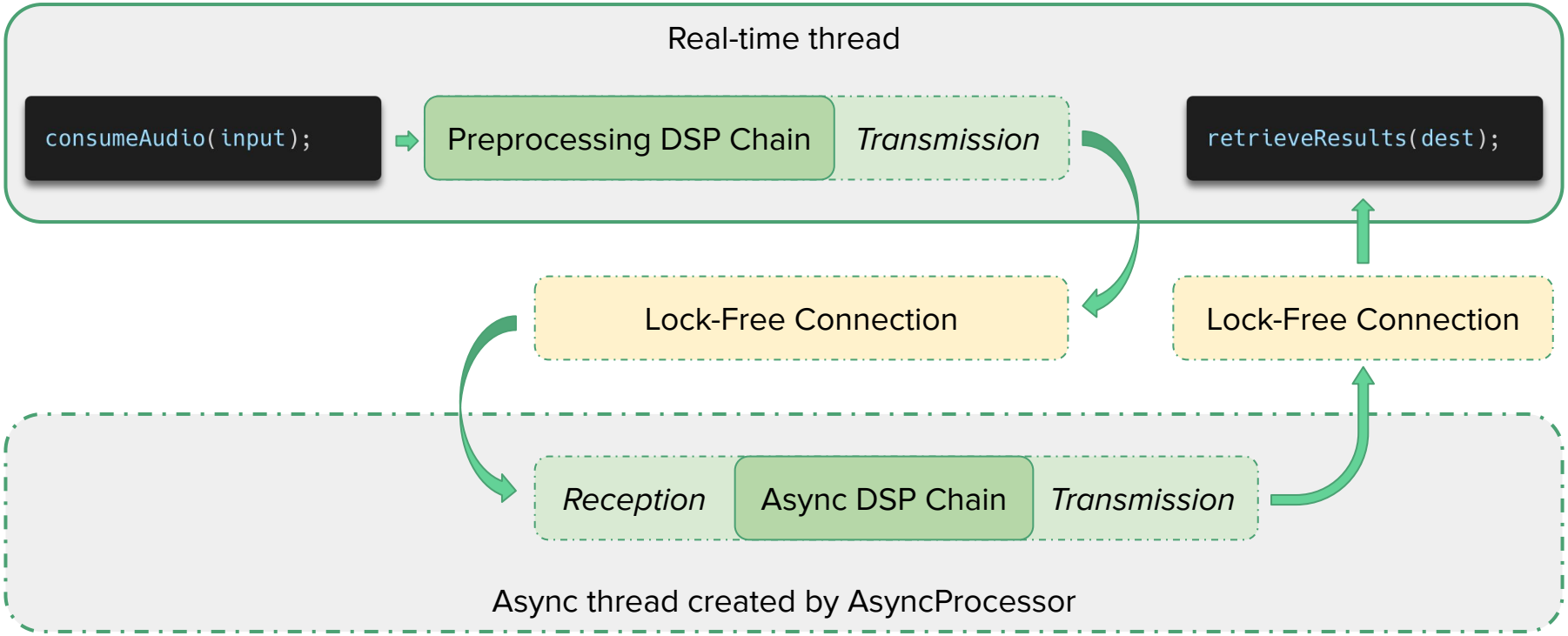
- AsyncProcessor will introduce:
  - **`TransmissionNode<LockFreeFIFO>`** at the end of the prechain and the async chain
  - **`ReceptionNode<LockFreeFIFO>`** at the beginning of the async chain

- Real-time interface:

```
void consumeAudio(InputType input) noexcept REALTIME;

bool retrieveResults(OutputType dest, int timeout_ns) noexcept REALTIME;
```

Asynchronous processing: Local async mode

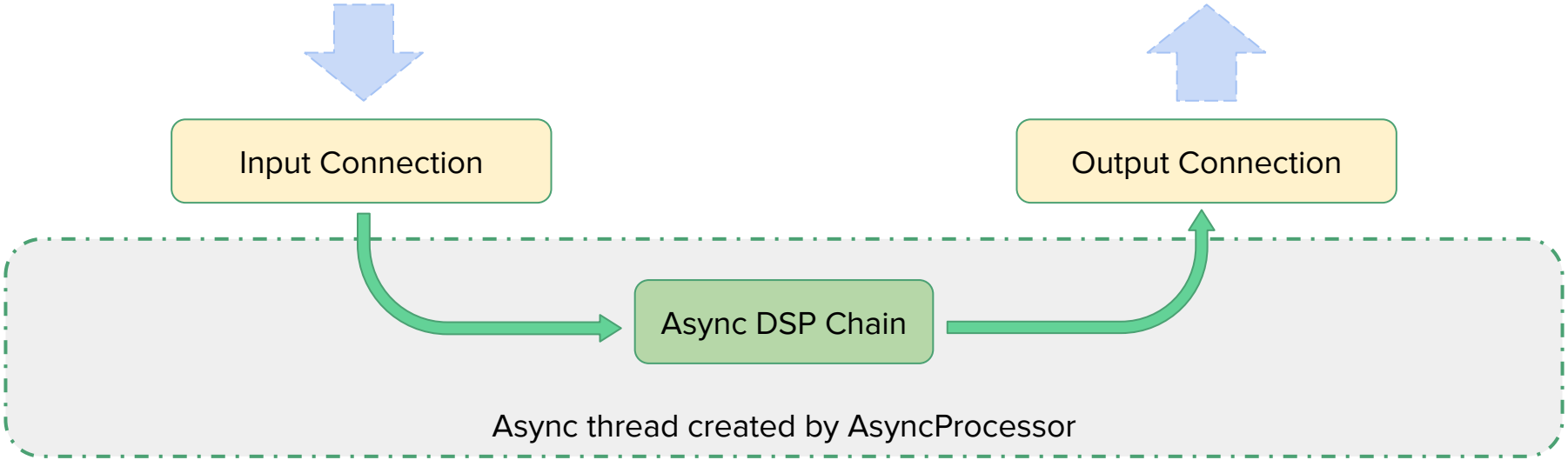


Asynchronous processing: Server mode

- User provides:
  - An async chain with a reception node **and** a transmission node
  - A specific input connection to use
  - A specific output connection to use

- No real-time interface. It will start on init and stop on cleanup.

Asynchronous processing: Server mode



Asynchronous processing: Client mode

- User provides:
  - An async chain with a reception node
  - A prechain with a transmission node
  - A specific input connection to use
  - A specific output connection to use

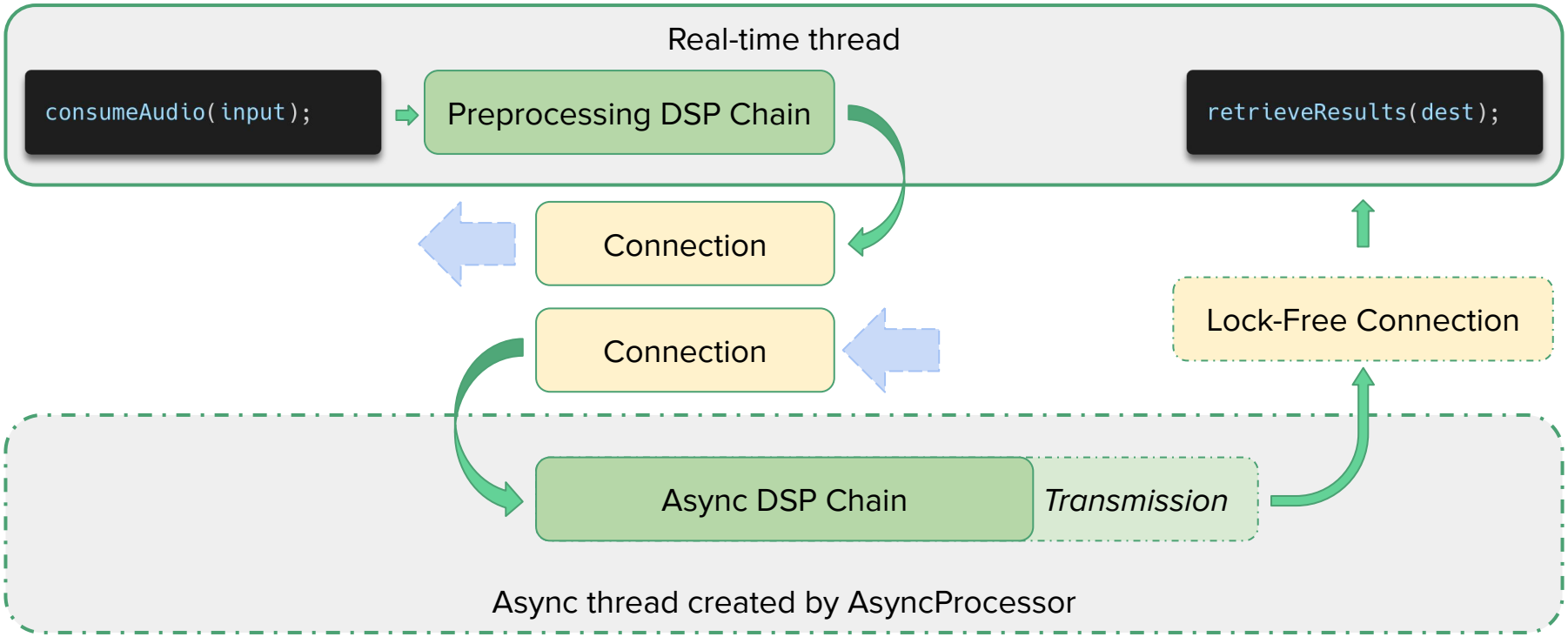
- AsyncProcessor will introduce:
  - **`TransmissionNode<LockFreeFIFO>`** at the end of the async chain

- Real-time interface: same as async local mode

```
void consumeAudio(InputType input) noexcept REALTIME;

bool retrieveResults(OutputType dest, int timeout_ns) noexcept REALTIME;
```

Asynchronous processing: Client mode



Backends

Backend selection

- One library per task: provide the base code

```
stft
inference
async
```

- One library per backend: provide the node classes

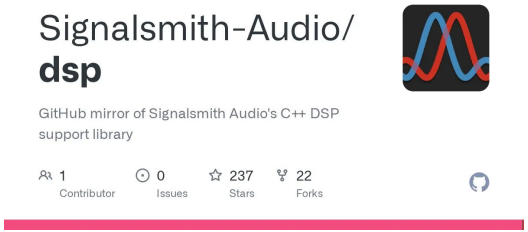
```
stft_signalsmith
stft_fftw
```

- User only links the backend library

List of backends

- Inference

- STFT



Inference

Inference support: tensors

- Standardized tensors
  - **data_in_<n>** / **data_out_<m>** tensors: I/O of the node
  - **parameter_<p>** tensors: controlled by the user
  - **state_in_<s>** / **state_out_<s>** tensors (ONNXRuntime only)

  - Tensors up to 4D, need to be bind to an data view type, will be statically enforced

```
// Must match the corresponding data view
using ShapeType = Shape<
  Dimension<DimensionType::kChannels, 2>,
  Dimension<DimensionType::kBlock, 512>>;
```

Inference support: model loading

- Each engine has its format

- Can be loaded from:
  - filepath
  - in-memory file
  - in-memory encrypted file with AES-CBC 256 bits

Nodes features

Nodes features:

- Nodes might ask for external memory to store its output
  - External memory is shared between nodes
  - Different function prototypes

```
void process(InputViewType input, OutputViewType output) noexcept REALTIME;
```

```
OutputViewType process(InputViewType input) noexcept REALTIME;
```

- Gated nodes:
  - A Gated node does not always produce an output for an input
  - No gated nodes are allowed in a Processor chain, only in AsyncProcessor

Some more details

Numpy loading

- Using compile-time python script to load it into a std::array

About real-time safety

- Some backends are not real-time safe:
  - onnxruntime
  - tensorflow lite
  - fftw
- Some backends are:
  - executorch
  - signalsmith dsp
- Tests are sanitized using rtsan, but the unsafe backend are deliberately ignored

About Security

- AES-CBC 256 bits is provided by mbedtls
- **`async_websockets`** is not yet configured for security

Licensing

- rt-machine-cpp is under the MIT license

- The FFTW backend is specifically under GPL3 license

iOS support

- The framework was built and tested for iOS 16.1

- Some inference backends are not supported
  - TensorFlow Lite
  - Executorch

Some interesting failures

- Integrating the DSP chip inference engine into the framework was impossible
  - Chip did not support C++

- Various intermediate design
  - Sync/Async was correlated with the data types: bad idea !

CRTP vs virtual methods

- Nodes use virtual methods
  - Better error messages
  - Only the size of vtables of overhead

- Backends use CRTP
  - No overhead
  - Nodes originally used CRTP

Thank you !

https://github.com/tpt-adasp/rt-machine-cpp