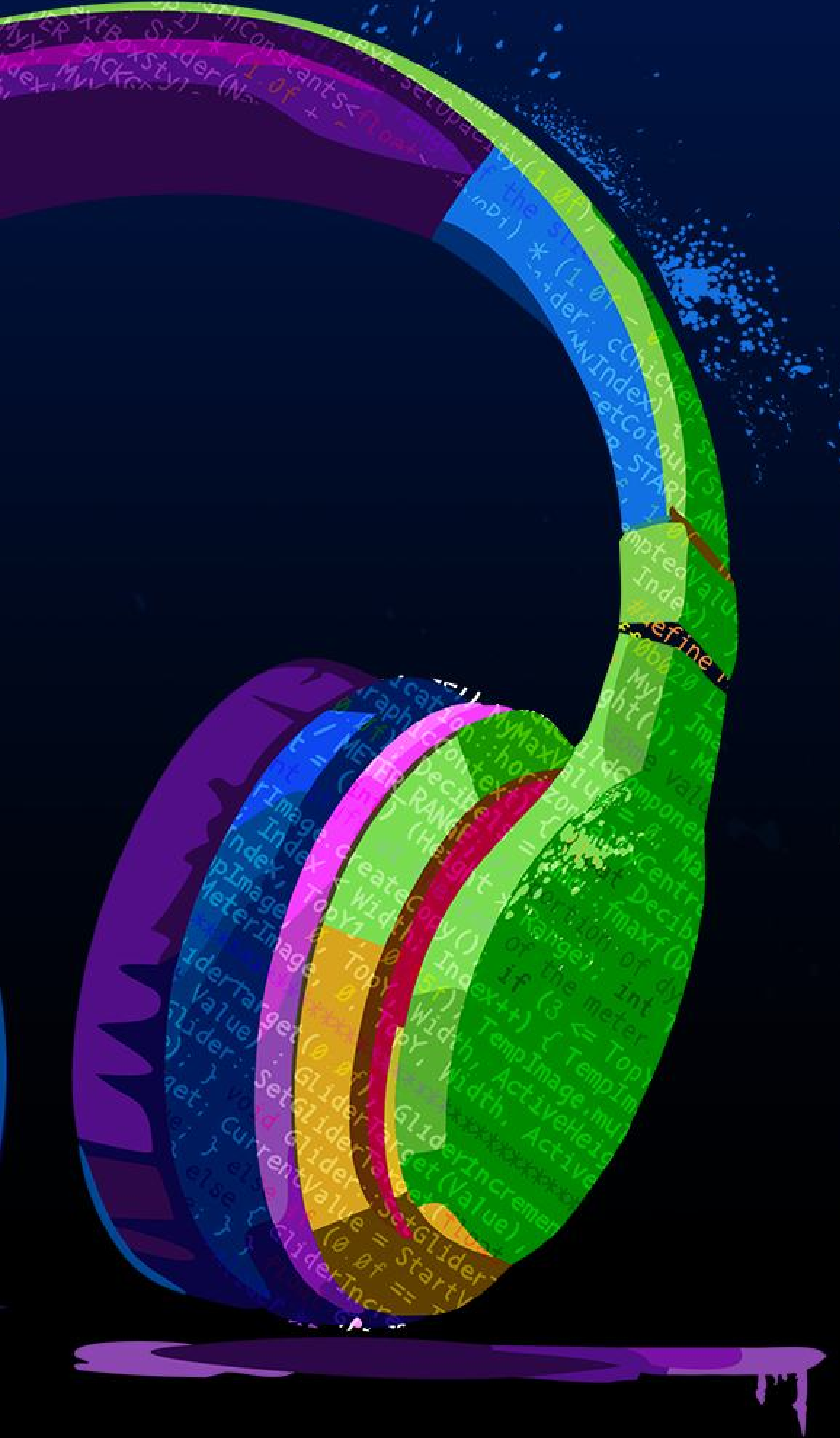


ADC24 Bristol

REAL-TIME INFERENCE OF NEURAL NETWORKS

A PRACTICAL APPROACH FOR DSP ENGINEERS – PART II

FARES SCHULZ
& VALENTIN ACKVA

Audio Developer Conference
Bristol 2024

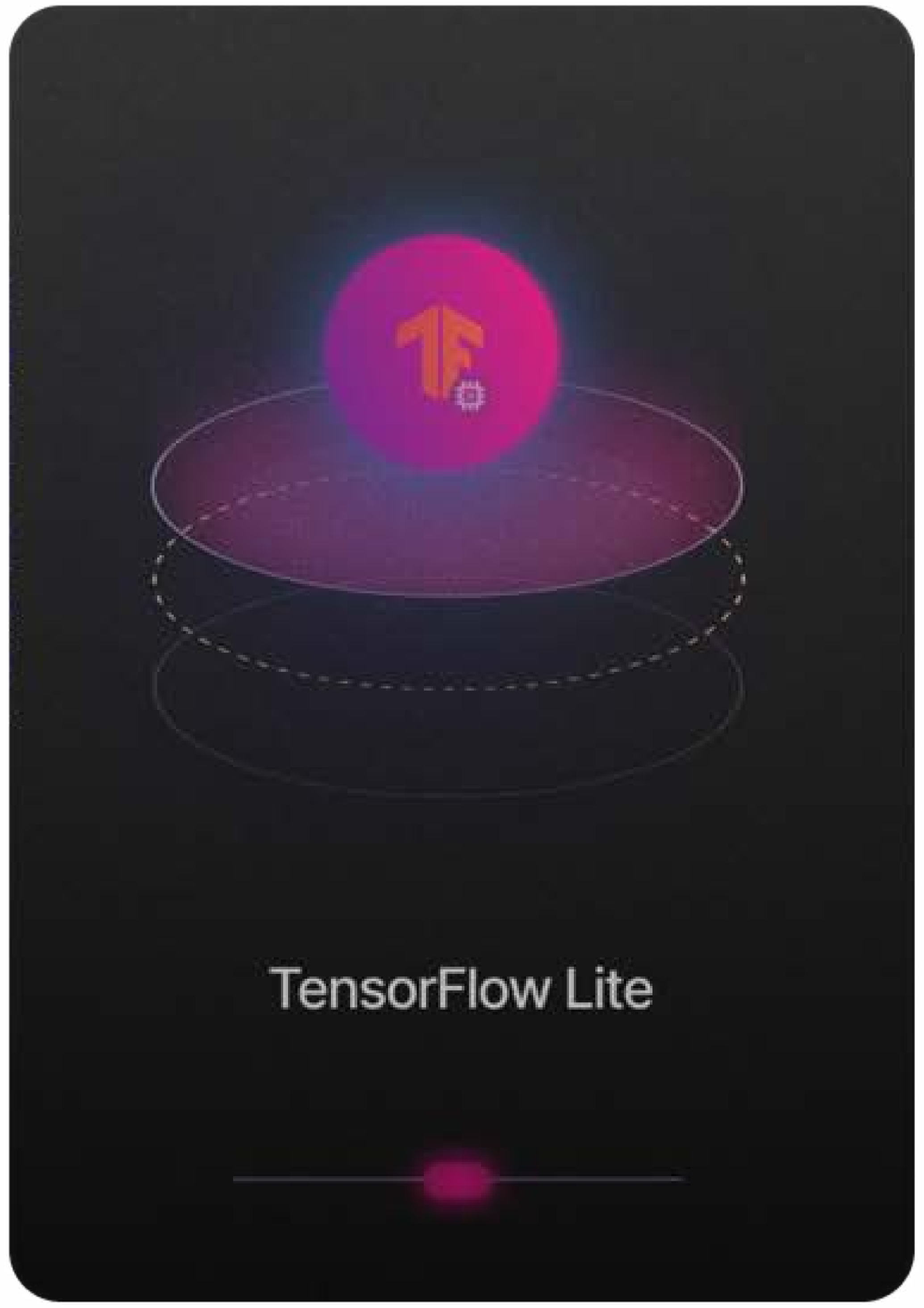
Real-Time Inference of Neural Networks

A Practical Approach for DSP Engineers – Part II

Fares Schulz   Valentin Ackva

Background

London, ADC 2023



nn-inference-template

About us

Fares Schulz

Researcher / Head of TU Studio

Technische Universität Berlin, Germany

Valentin Ackva

Audio Software Developer

INSONE GmbH in Leipzig, Germany

Table of Content

Chapter I

Introduction

Recap Part I, Relevance

Neural Network Integration

Train Neural Network

using Pytorch / Tensorflow

Python

Export NN Model

format for inference stage

Python

Implement in Audio Environment

ensure real-time safety

C++

How to Implement the Inference?

Write inference yourself

using

std::lib, Eigen, SIMD

Use specialized libraries

known as

inference engines

How to Implement the Inference?

Write inference yourself

using

std::lib, Eigen, SIMD

Use specialized libraries

known as

inference engines

Major Inference Engines

Onnx Runtime

LibTorch

TensorFlow Lite

Real-time Principles

- These inference engines favor average execution times
- None of the libraries gives real-time safety guaranties
- Confusion on real-time safety of major inference engines
  - Chowdhury finds no real-time safe (2021)
  - Stefani et al. conclude real-time safety after first inference (2022)
- Noteworthy: RTNeural – a real-time safe inference engine
  - Fast for small networks
  - Very limited layer support

Chowdhury, J. (2021). Rtneural: Fast neural inferencing for real-time systems. *arXiv preprint arXiv:2106.03037.*

Stefani, D., Peroni, S., & Turchet, L. (2022). A comparison of deep learning inference engines for embedded real-time audio classification. In *Proceedings of the International Conference on Digital Audio Effects, DAFx* (Vol. 3, pp. 256-263).

RTNEURAL

In the Last Talk

- Outline of the pipeline for implementing neural networks in audio plug-ins
- Overview of major inference engines
- Presentation of a real-time safe plug-in template
- Basic benchmark for inference engines inferring a neural network model
- Discussion about continuous signals / streamability of neural networks

Part I - ADC23

In This Talk

- Quantification of real-time violations of inference engines
- Cross-platform library - ANIRA: An Architecture for Neural Network Inference in Real-Time Audio Applications
  - Streamlines the use of neural networks in any real-time audio environment
  - Significant improvements for the use of multiple instances
  - Refined latency calculation
  - Built-in benchmarking capabilities
- Performance impact of various factors on inference runtimes

Chapter II

Examine Real-Time Violations

Methods, Results

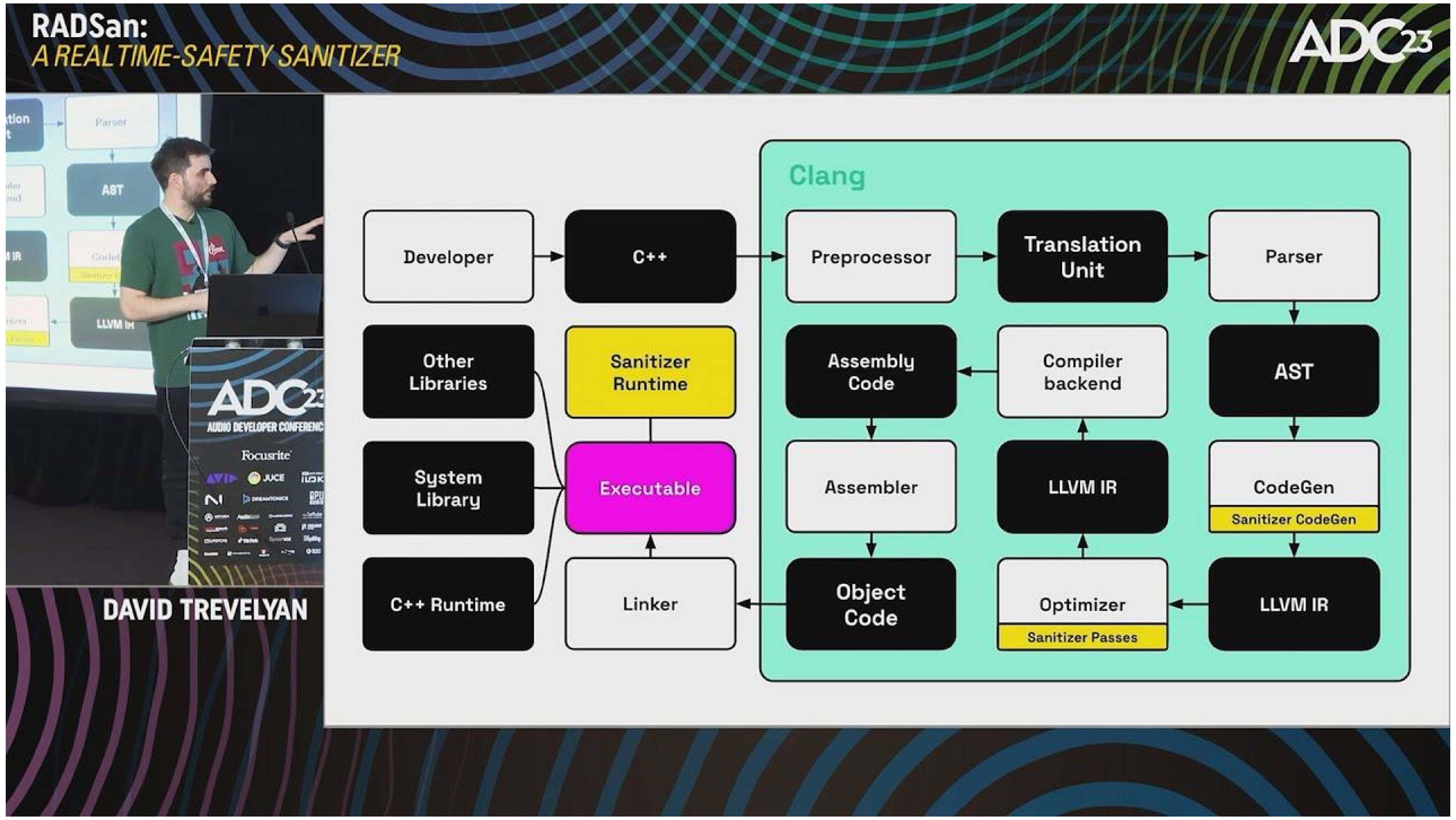
Real-Time Violation Checks

How can we check real-time violations by external libraries?

Verification Method

Real Time Sanitizer - RTSan (prev. RADSan):

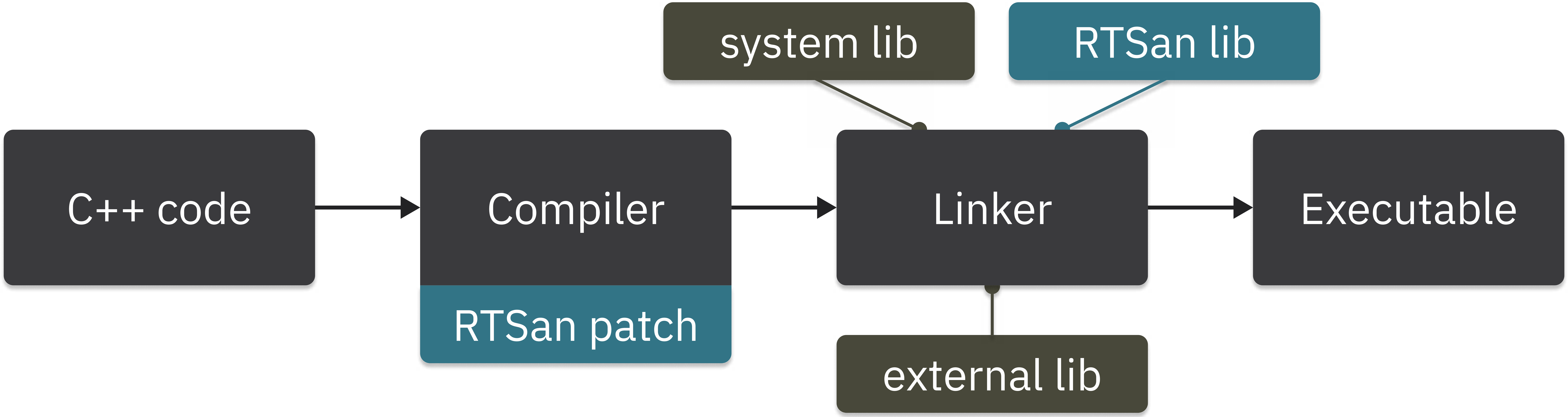
- Part of clang compiler and runtime library



Verification Method

How does RTSan work?

- Introduces real-time context and intercepts for non-real-time safe operations like `malloc` and `free`



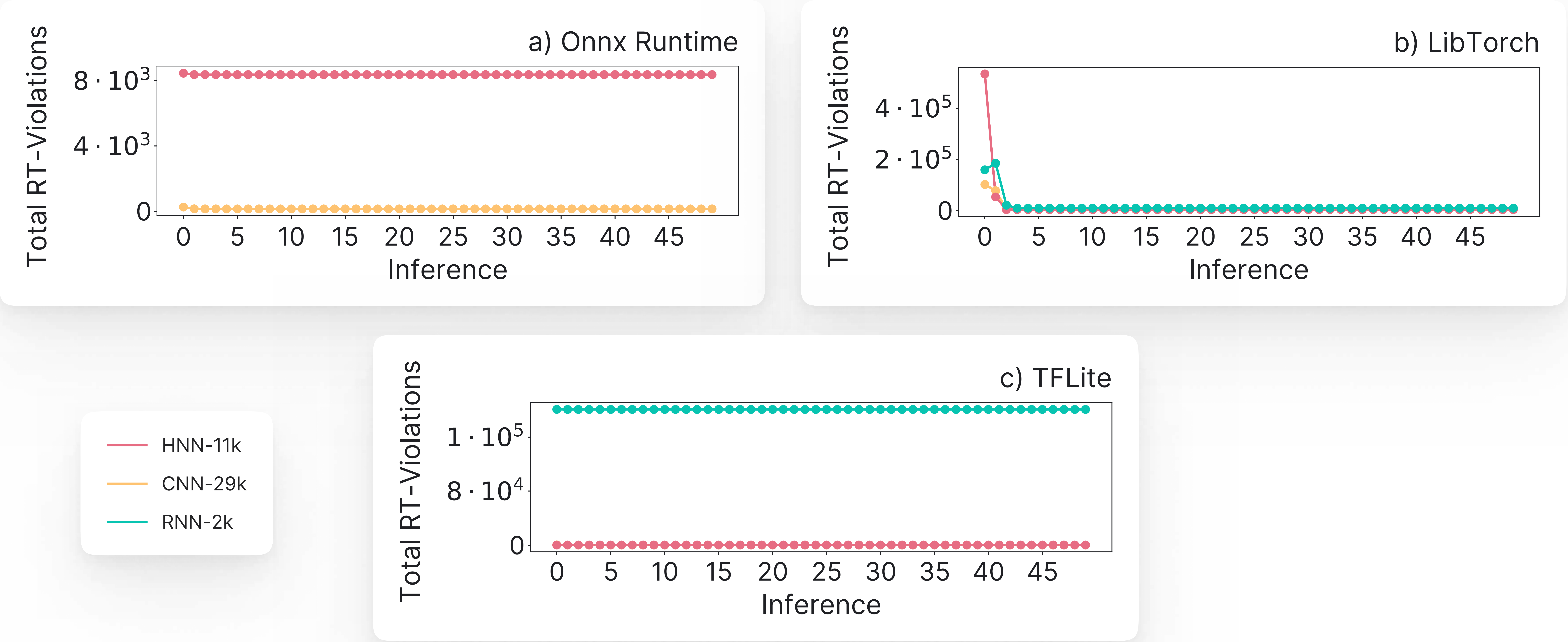
Test Setup Overview

OnnxRuntime | LibTorch | TensorFlow Lite

Convolutional Neural Network | Recurrent Neural Network | Hybrid Neural Network

1 2 3 4 5 6 ... 48 49 50

Results


a) Onnx Runtime
Total RT-Violations
$8 \cdot 10^3$
$4 \cdot 10^3$
0
0 5 10 15 20 25 30 35 40 45
Inference
b) LibTorch
Total RT-Violations
$4 \cdot 10^5$
$2 \cdot 10^5$
0
0 5 10 15 20 25 30 35 40 45
Inference
c) TFLite
Total RT-Violations
$1 \cdot 10^5$
$8 \cdot 10^4$
0
0 5 10 15 20 25 30 35 40 45
Inference
HNN-11k
CNN-29k
RNN-2k

Inference Engine Integration

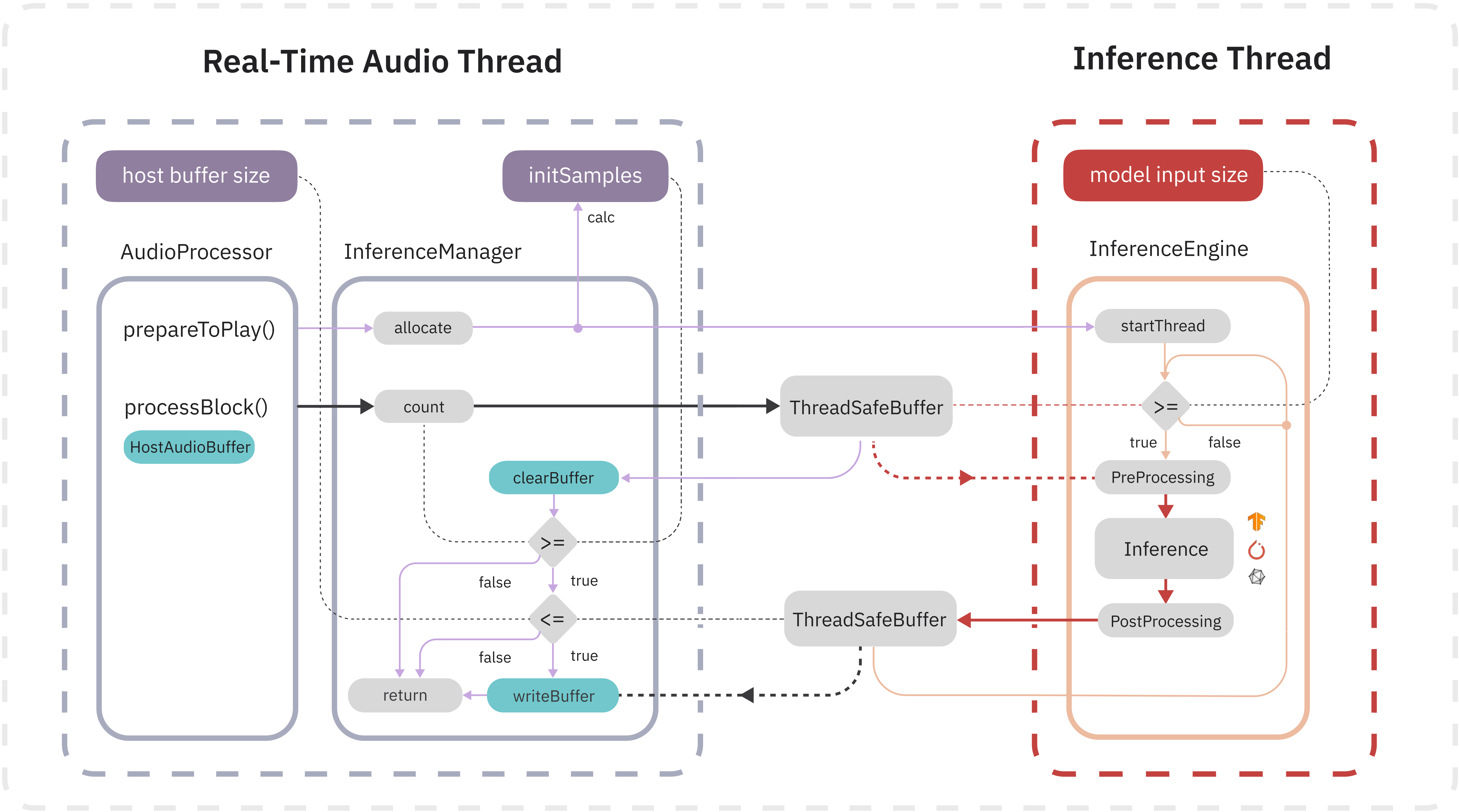
We should not run these engines on the real-time thread!

Chapter III

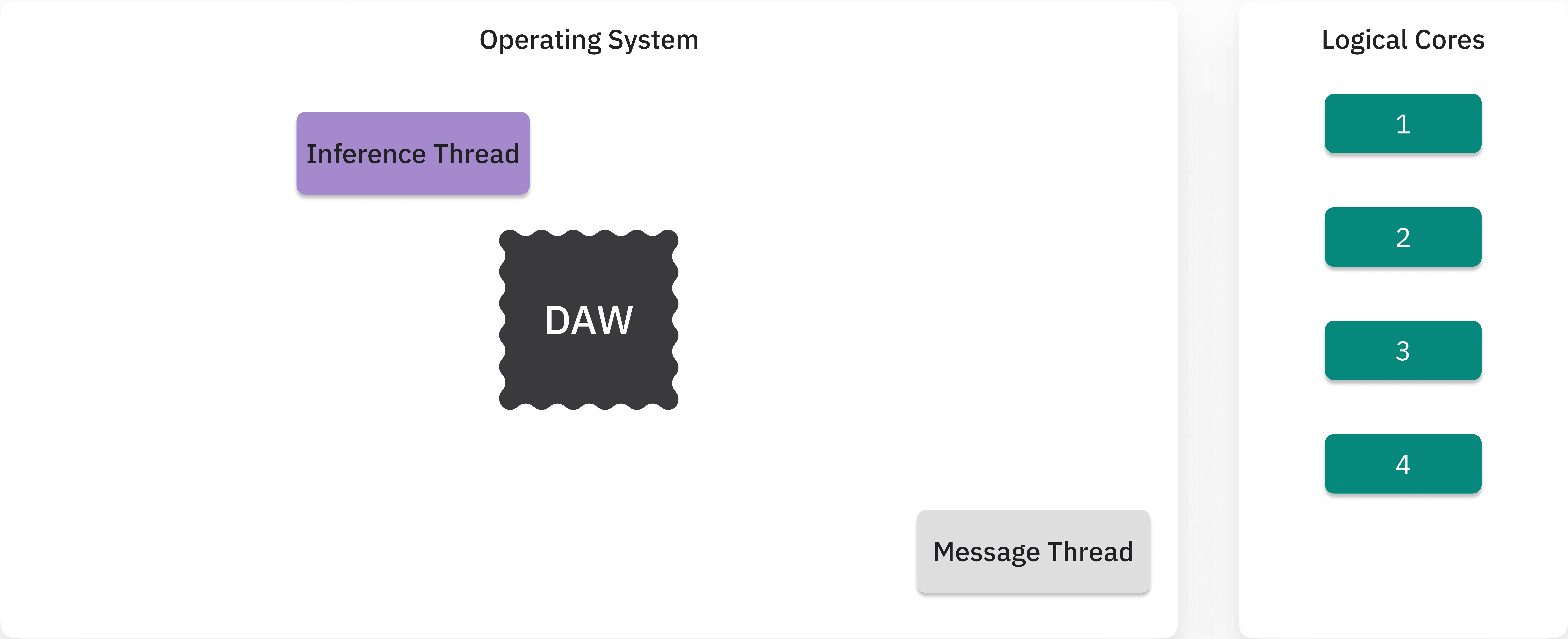
Library Architecture

Solving Limitations, Interface

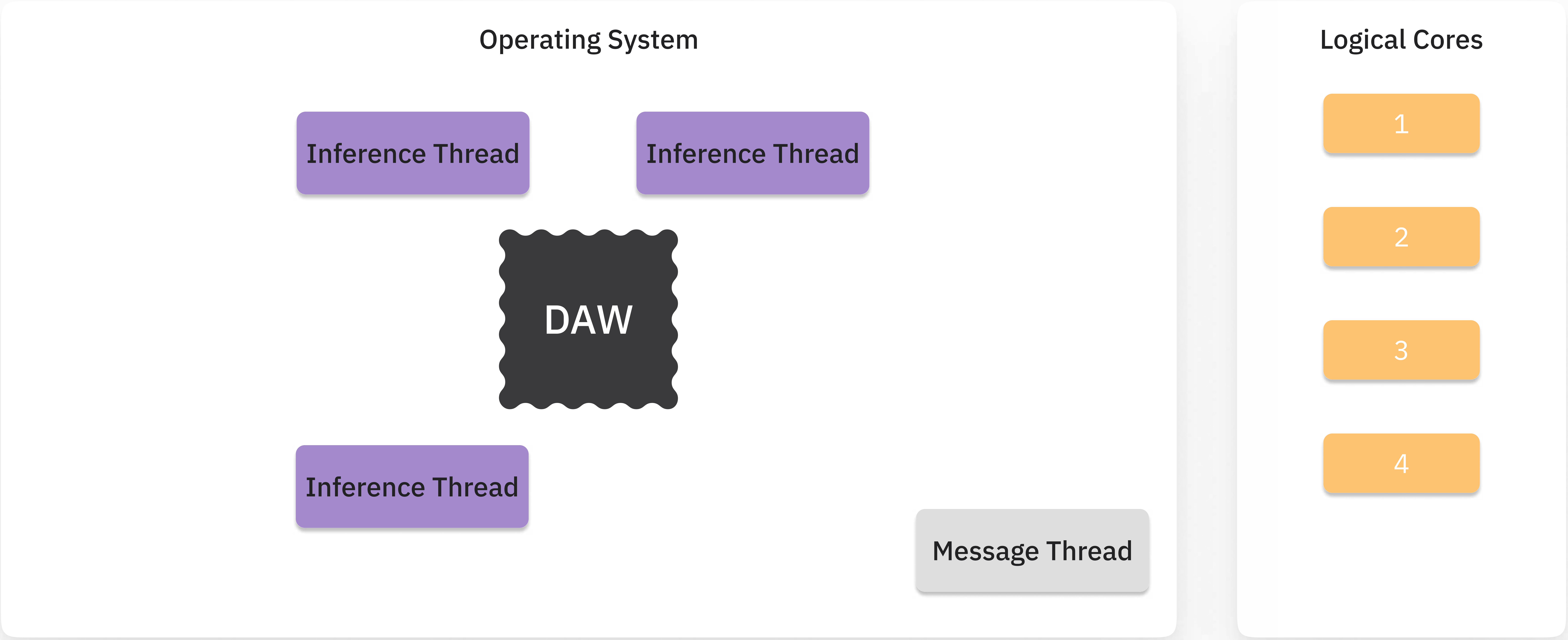
Proposed Architecture - ADC23



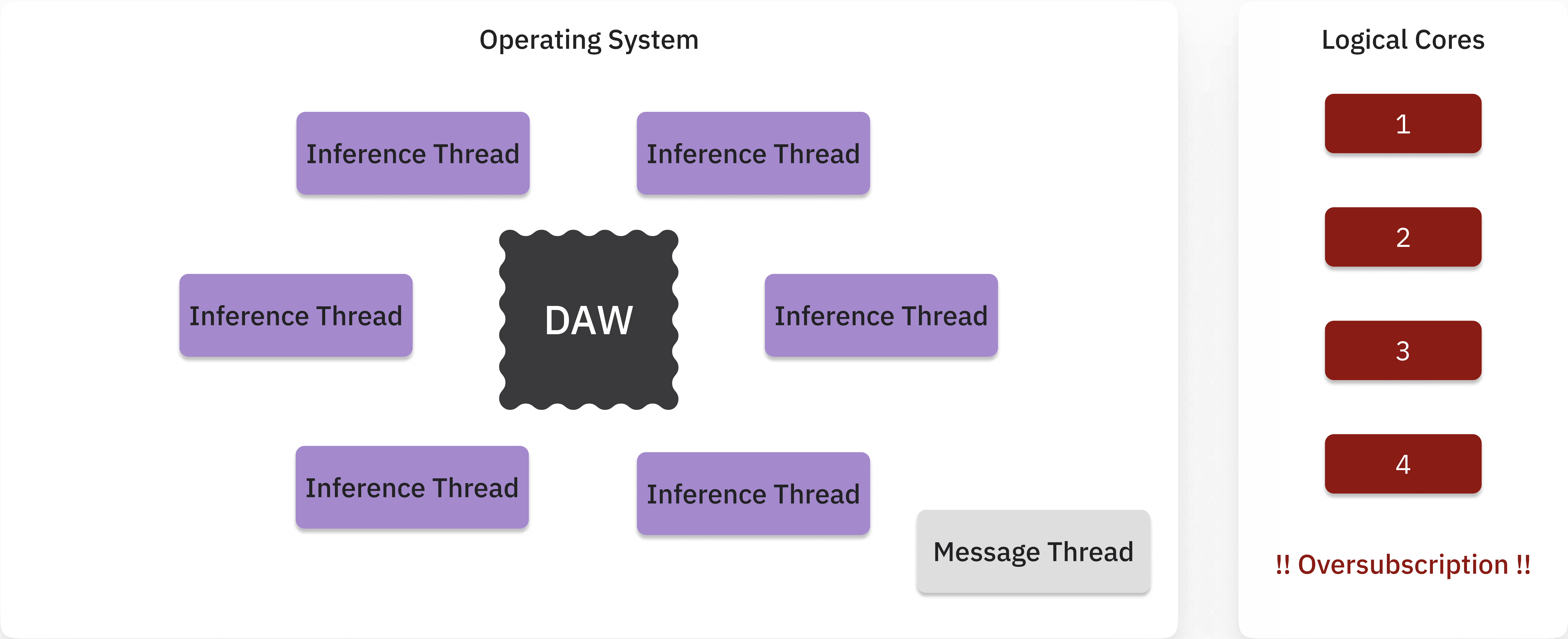
Architecture Limitation



Architecture Limitation



Architecture Limitation



Architecture Limitation

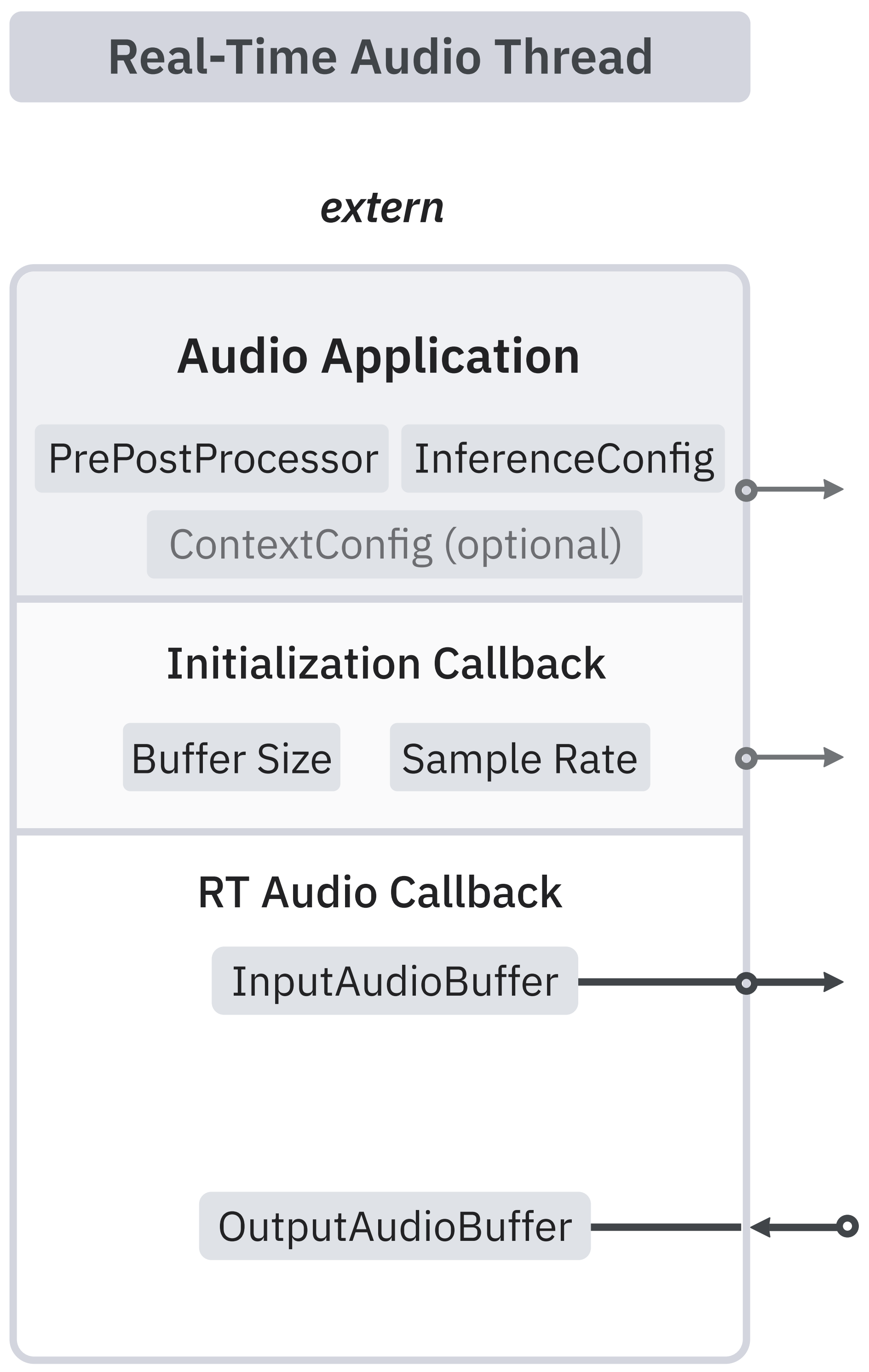
Oversubscription Problem:

- If number of high-priority threads > number of logical cores
- Causes threads to compete for the same cores
- Can be especially problematic for real-time processes

Solution - Static Thread Pool Design:

- Shares inference threads across instances (e.g. different plugins)
- Can enable parallel execution for faster inference
- Implementation requires developing a **dedicated library**

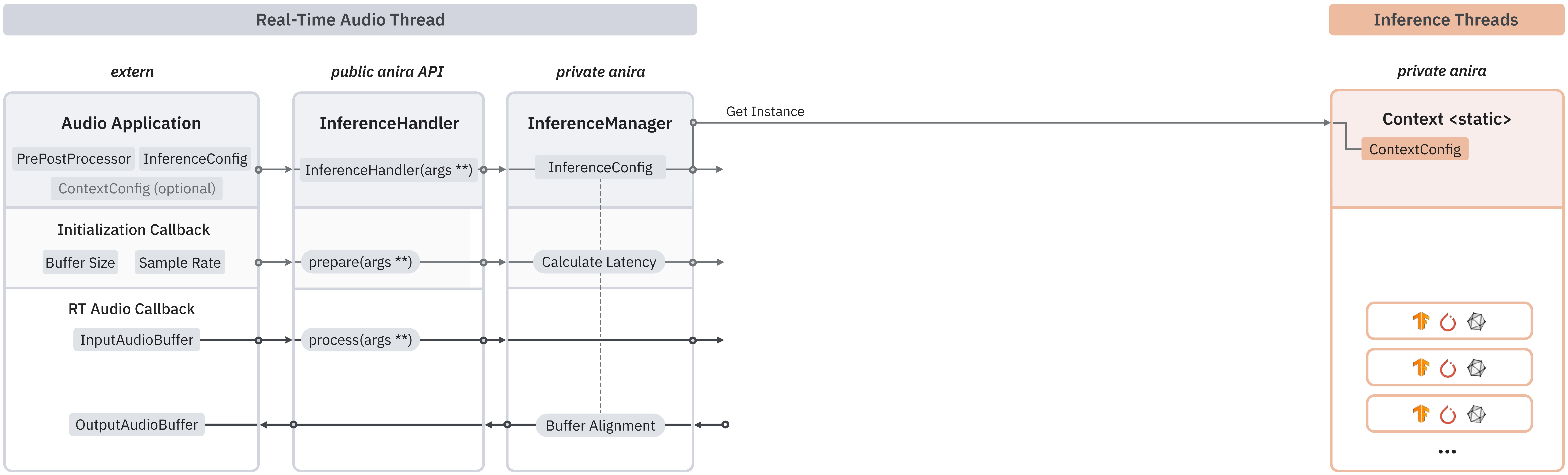
Architecture



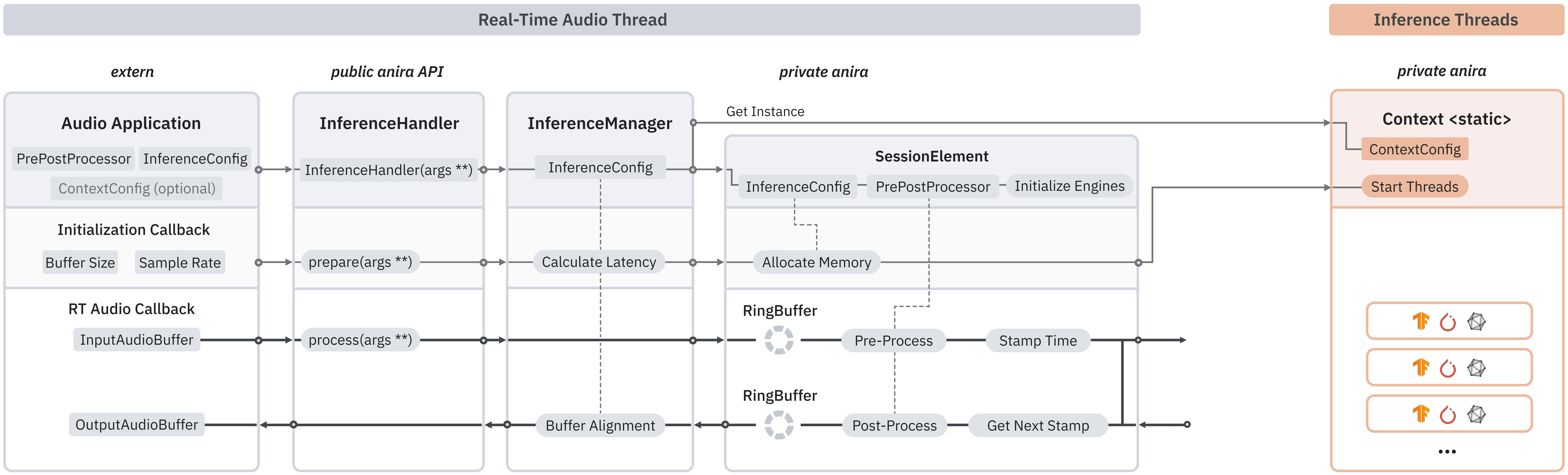
Architecture

Architecture

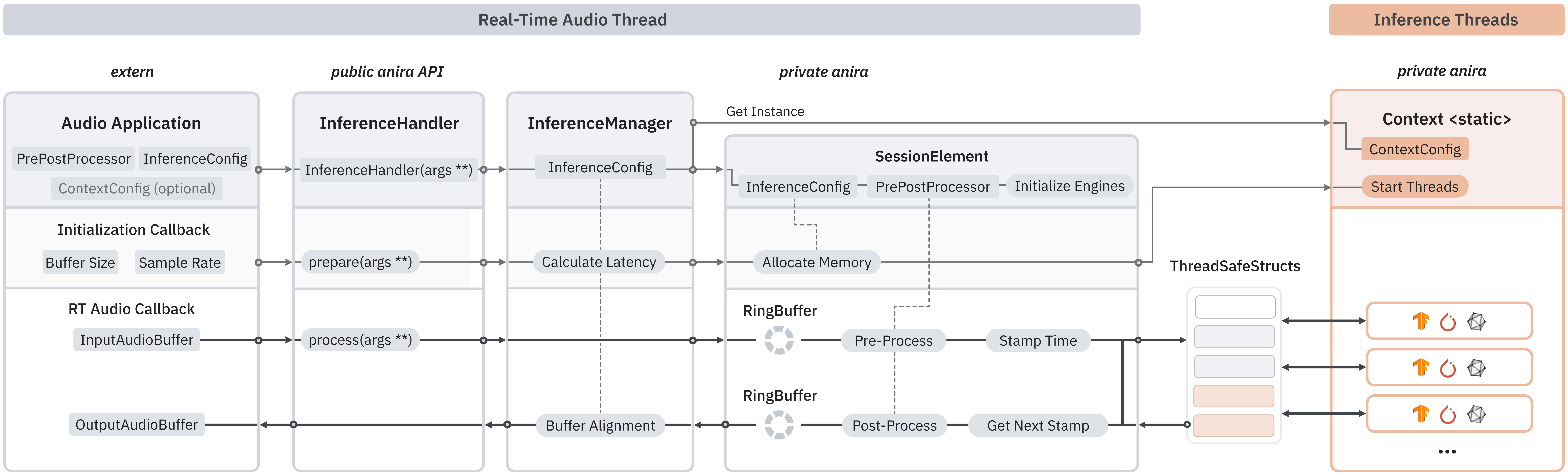
Architecture



Architecture



Architecture



Library Integration

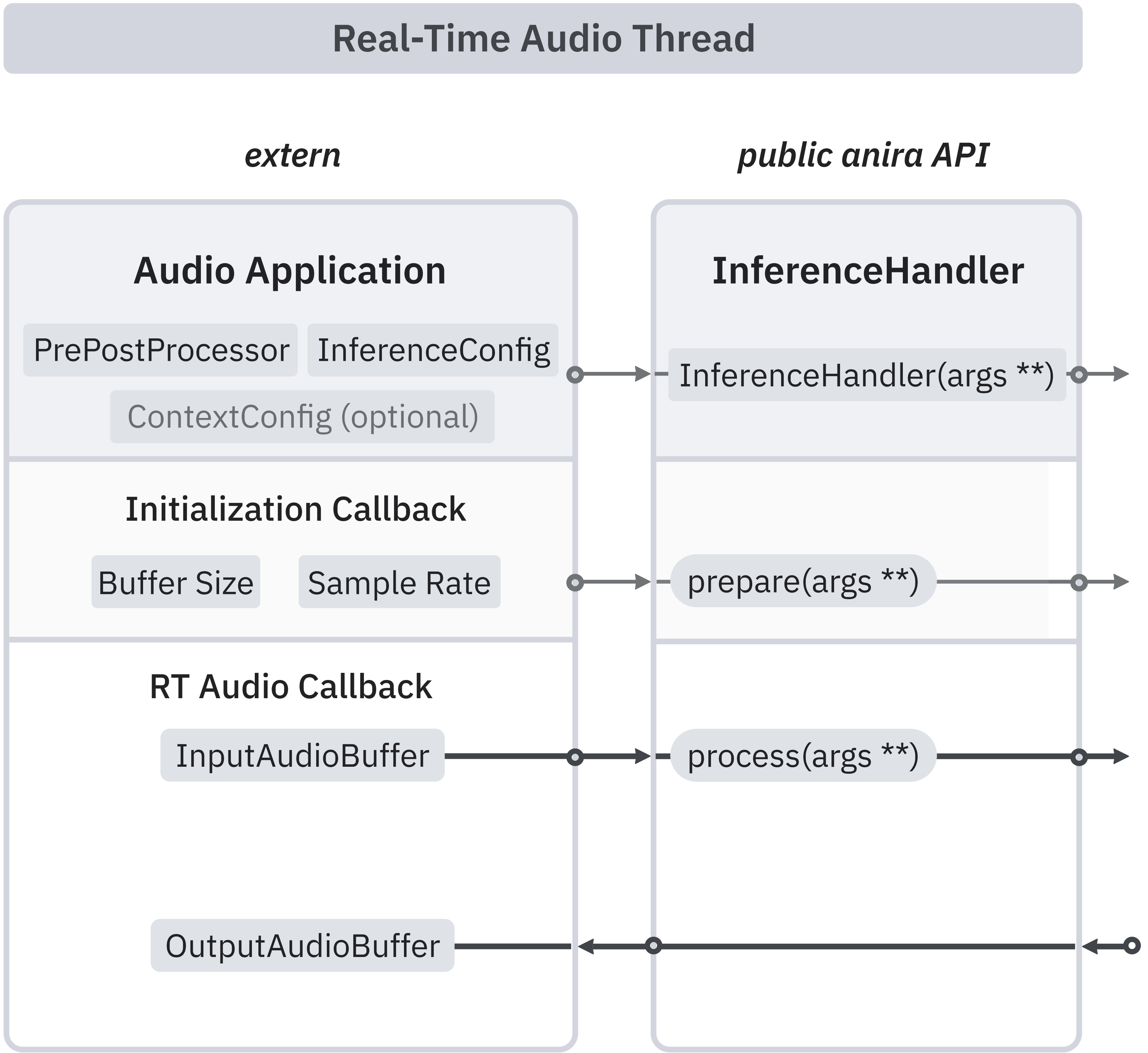
How does the interaction with the library work?

Interface

In order to use the anira library :

- Configure inference parameters in `anira::InferenceConfig`
- Optional: Define a `anira::ContextConfig`
- Optional: Define a custom `anira::PrePostProcessor`

⟶ **Everything else** will be handled automatically, regardless of the neural network type



Interface

Necessary Parameters:

- Model data path*
- Model shapes*
- Max inference time

`* definable for multiple inference engines`

Context (optional, shared):

- Number of threads

Optional Parameters:

- Model latency
- Warm-up inference
- Number of channels
- Bind session to processor
- Number of parallel processors
- ...

Code Example

```cpp
#include <anira/anira.h>

anira::InferenceConfig inference_config(
     {“path/to/model.onnx", anira::InferenceBackend::ONNX} // Model data
     {{{1, 1, 512}}, {{1, 1, 512}}}, // Input- Output-Tensor shapes
     5.0f, // Maximum inference time in ms
);

anira::PrePostProcessor pp_processor; // Create default pre- and post-processor
anira::InferenceHandler inference_handler(pp_processor, inference_config); // InferenceHandler

inference_handler.prepare({buffer_size, sample_rate}); // Allocate memory
int latency_in_samples = inference_handler.get_latency(); // Get latency of the inference process

inference_handler.set_inference_backend(anira::InferenceBackend::ONNX); // Select the backend

process_block(float** audio_data, int num_samples) {
    inference_handler.process(audio_data, num_samples); // Real-time safe audio processing
}
```

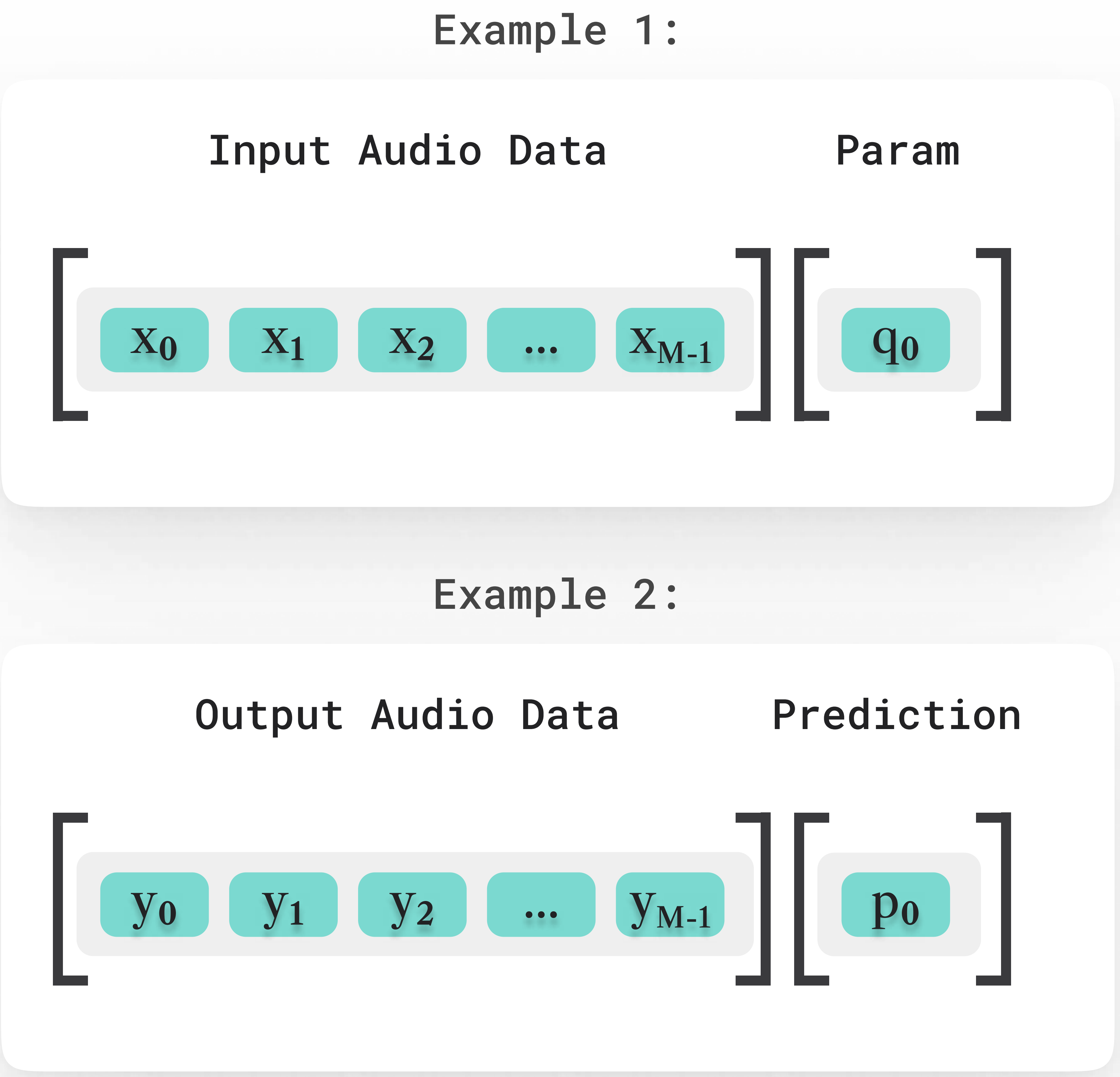
Custom PrePostProcessor

Not all networks process a fixed number of samples in and out:

- Need for a custom pre- and postprocessing

A thread-safe mechanism is provided to:

- Pass parameters
- Inject or retrieve state information
- submit a receptive field



Supported Inference Engines

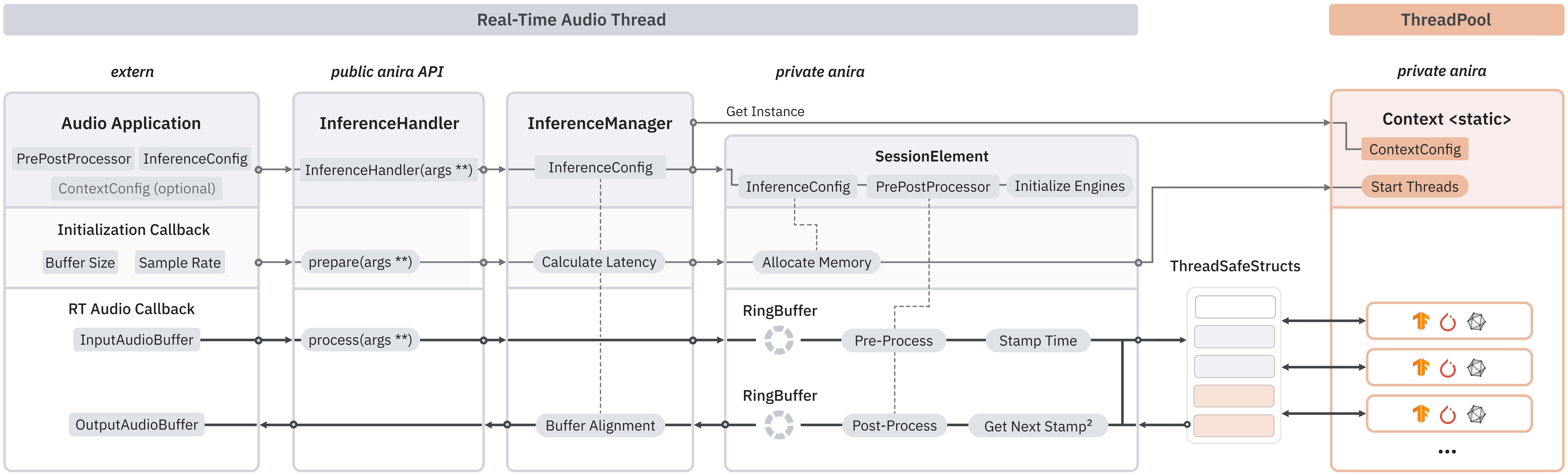
Onnx Runtime	LibTorch	TensorFlow Lite	Custom
			Any Backend / Customization

Chapter IV

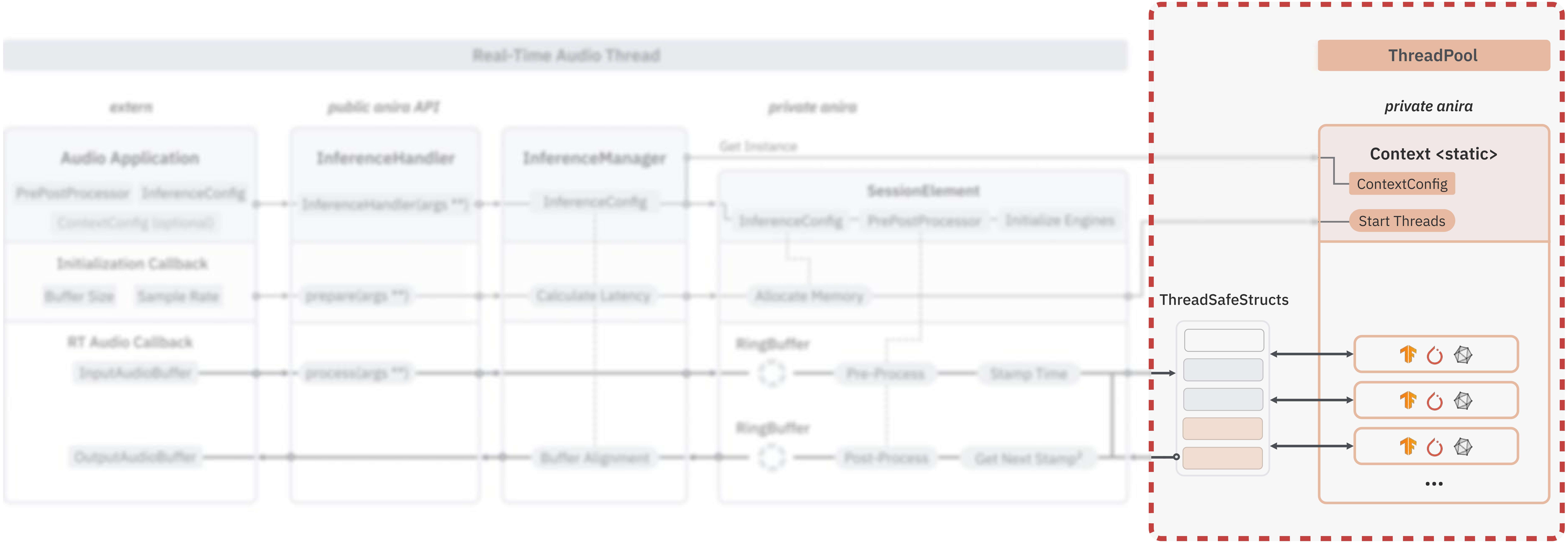
Deep Dive Thread Pool and Latency

Static Thread Pool Design, Minimum Latency

Architecture Overview

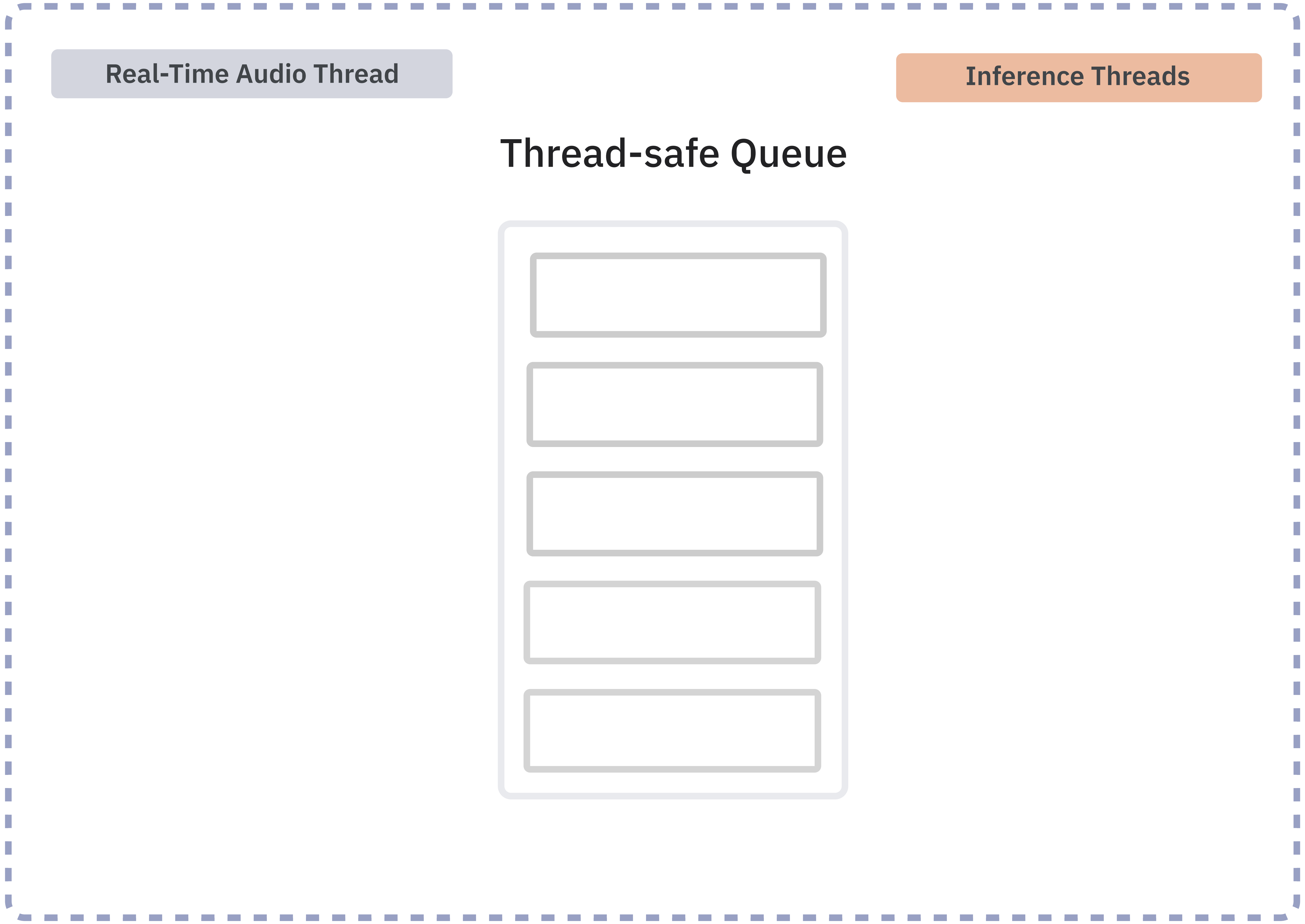


Architecture Overview



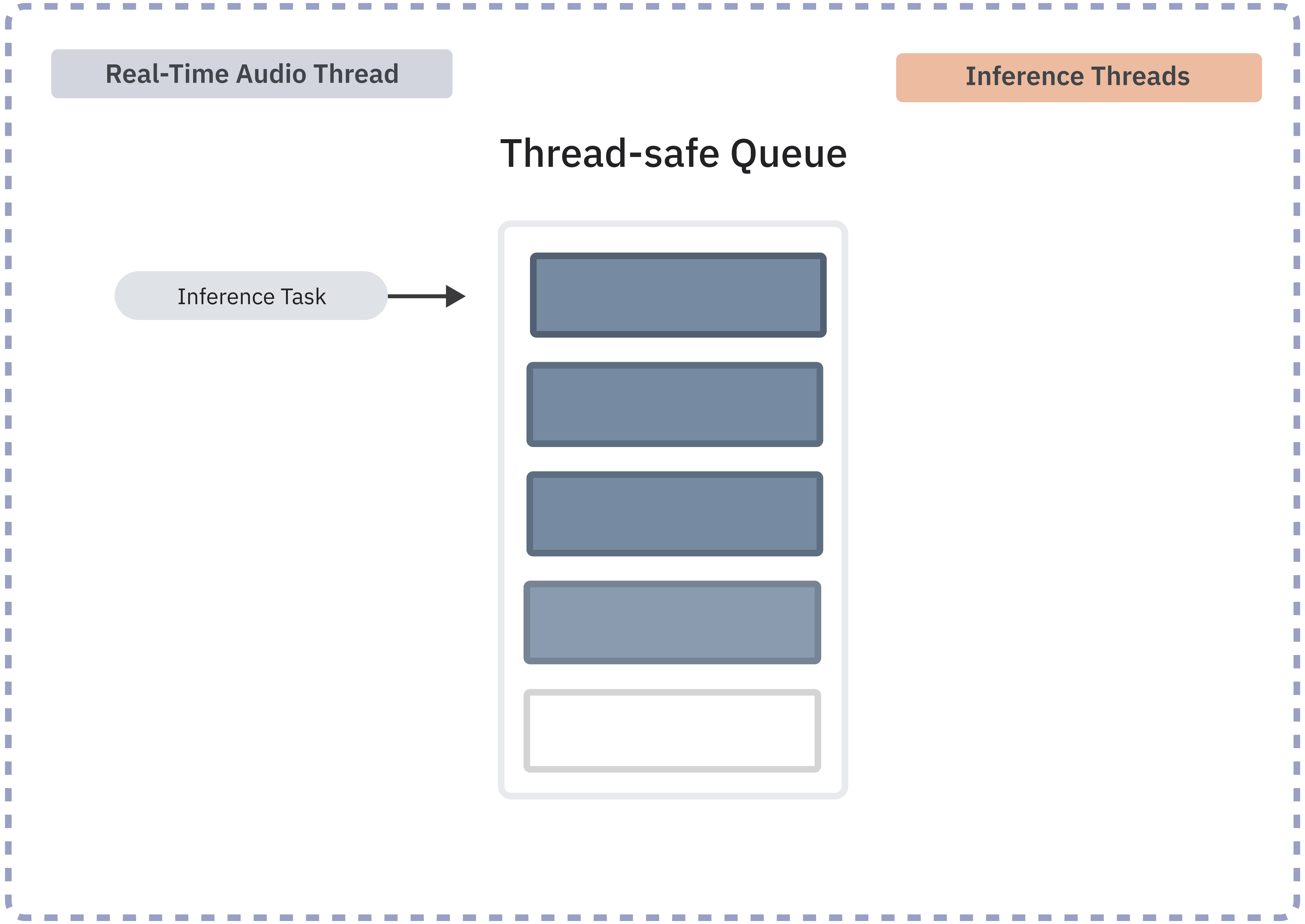
Coordinating Inference Tasks

- Thread-safe structures shared by multiple threads
- Use of `std::atomics` to avoid data races
- Thread-safe queue to coordinate job submission



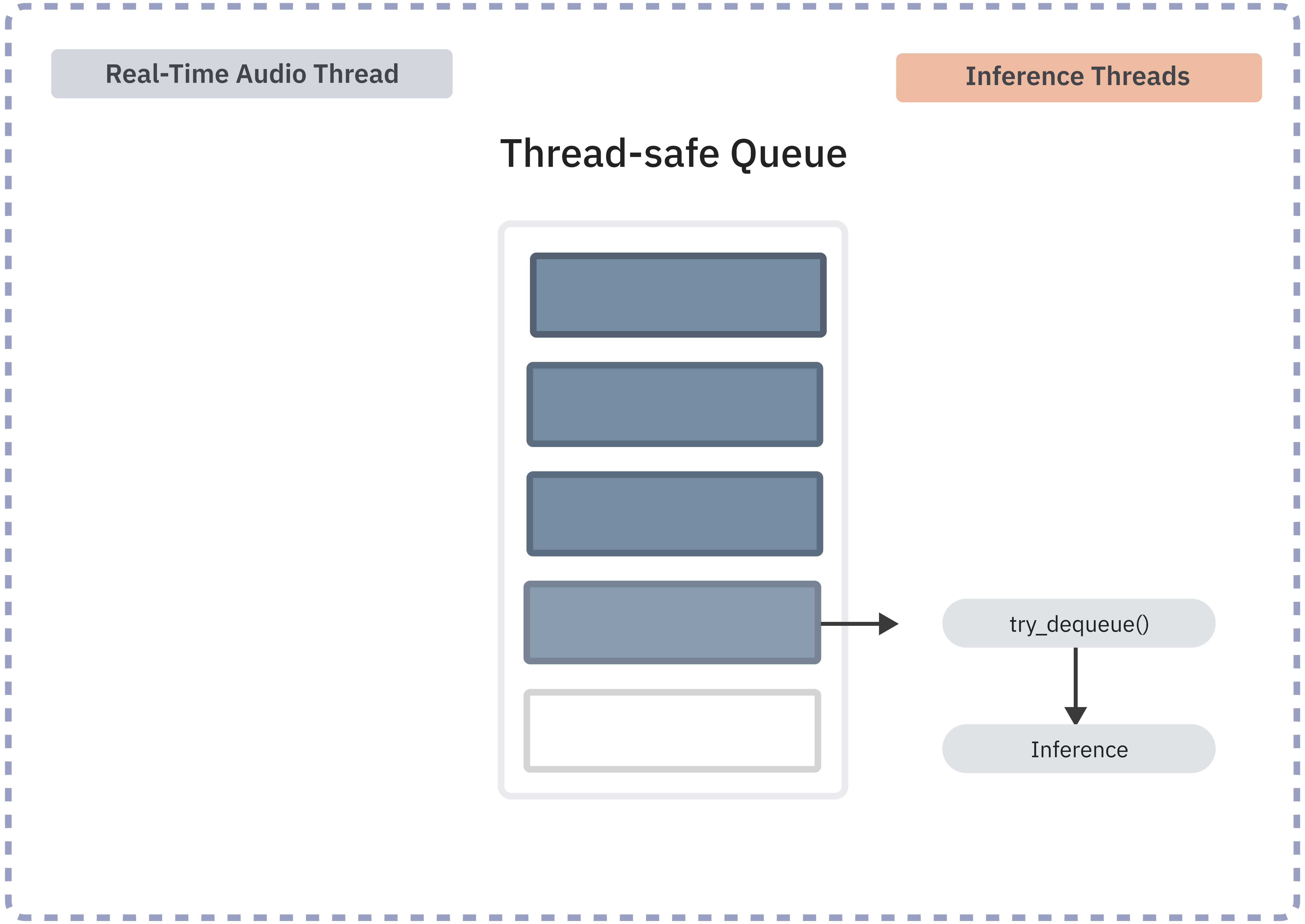
Coordinating Inference Tasks

- Thread-safe structures shared by multiple threads
- Use of `std::atomics` to avoid data races
- Thread-safe queue to coordinate job submission
- Audio thread enqueues inference task



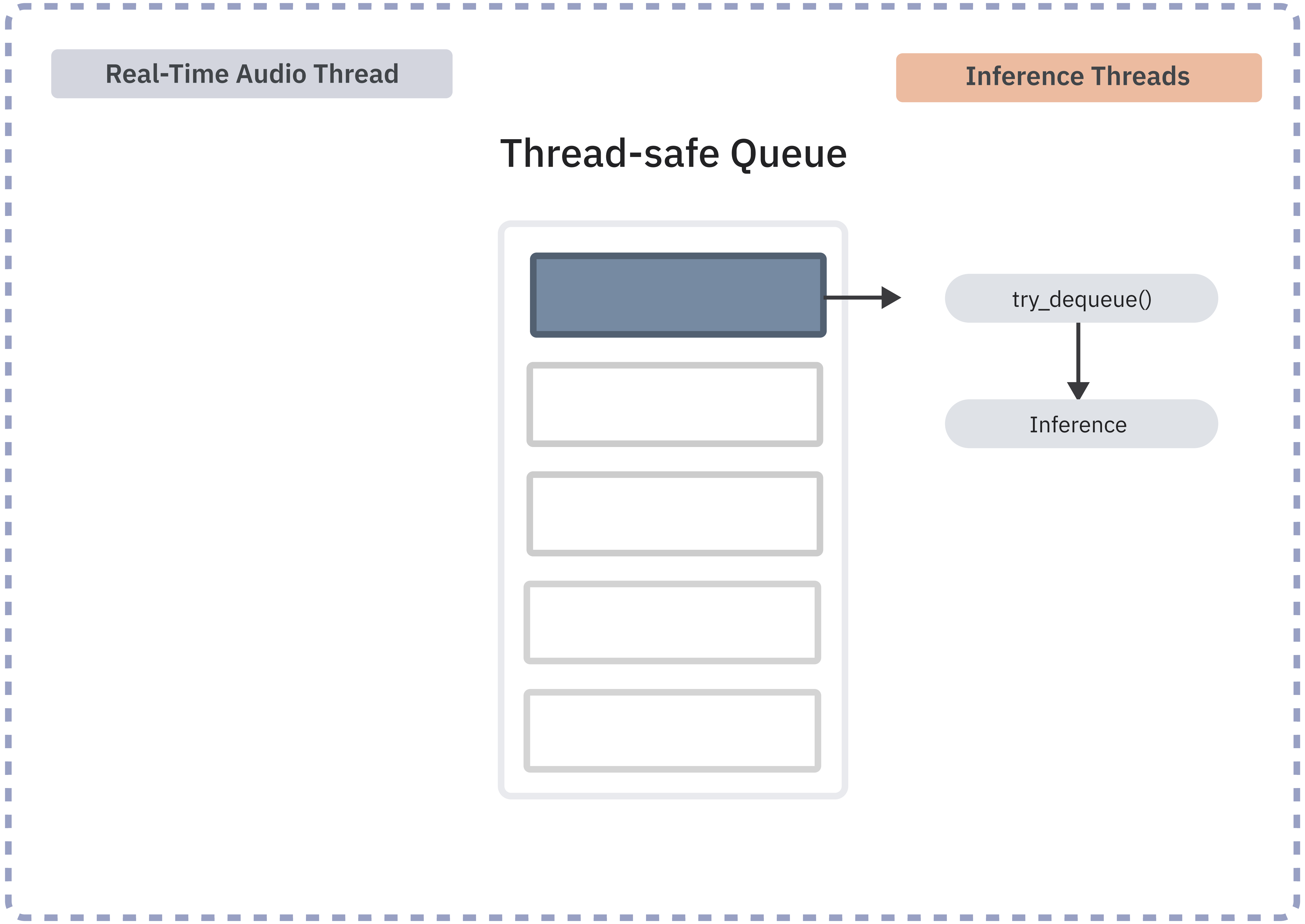
Coordinating Inference Tasks

- Thread-safe structures shared by multiple threads
- Use of `std::atomics` to avoid data races
- Thread-safe queue to coordinate job submission
- Audio thread enqueues inference task
- Inference threads infer oldest entry



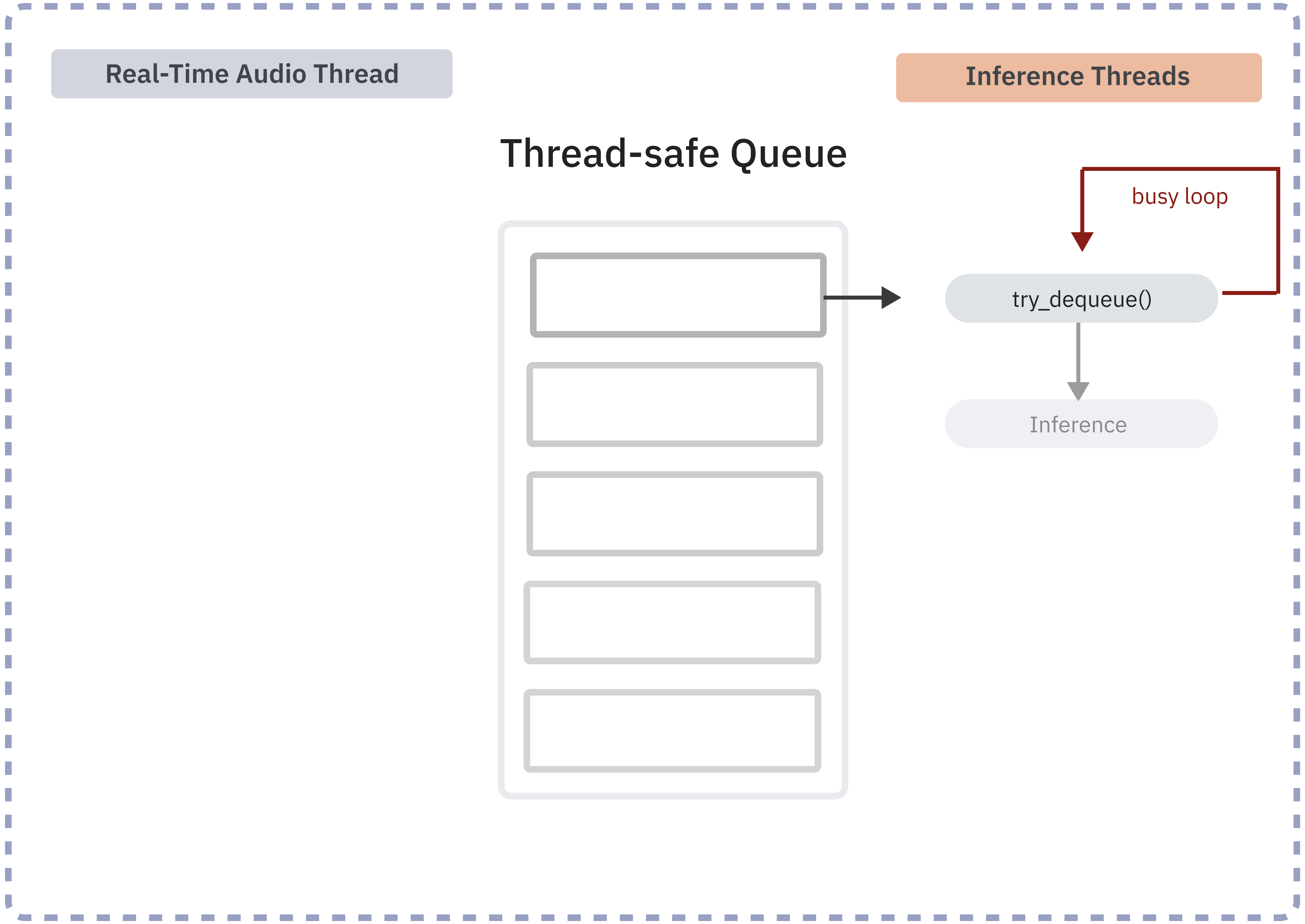
Coordinating Inference Tasks

- Thread-safe structures shared by multiple threads
- Use of `std::atomics` to avoid data races
- Thread-safe queue to coordinate job submission
- Audio thread enqueues inference task
- Inference threads infer oldest entry



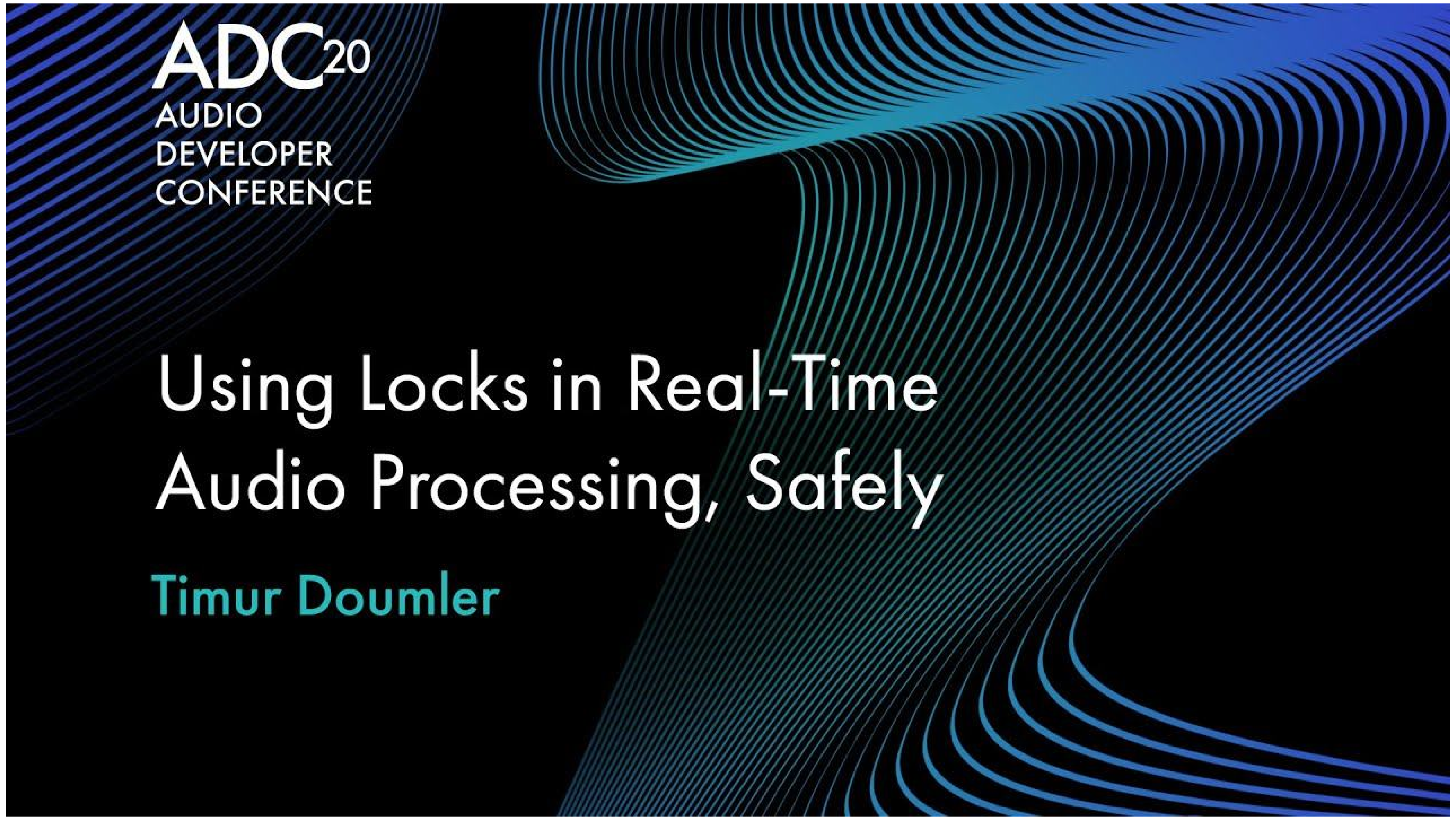
Coordinating Inference Tasks

- Thread-safe structures shared by multiple threads
- Use of `std::atomics` to avoid data races
- Thread-safe queue to coordinate job submission
- Audio thread enqueues inference task
- Inference threads infer oldest entry
- Busy loop if queue is empty

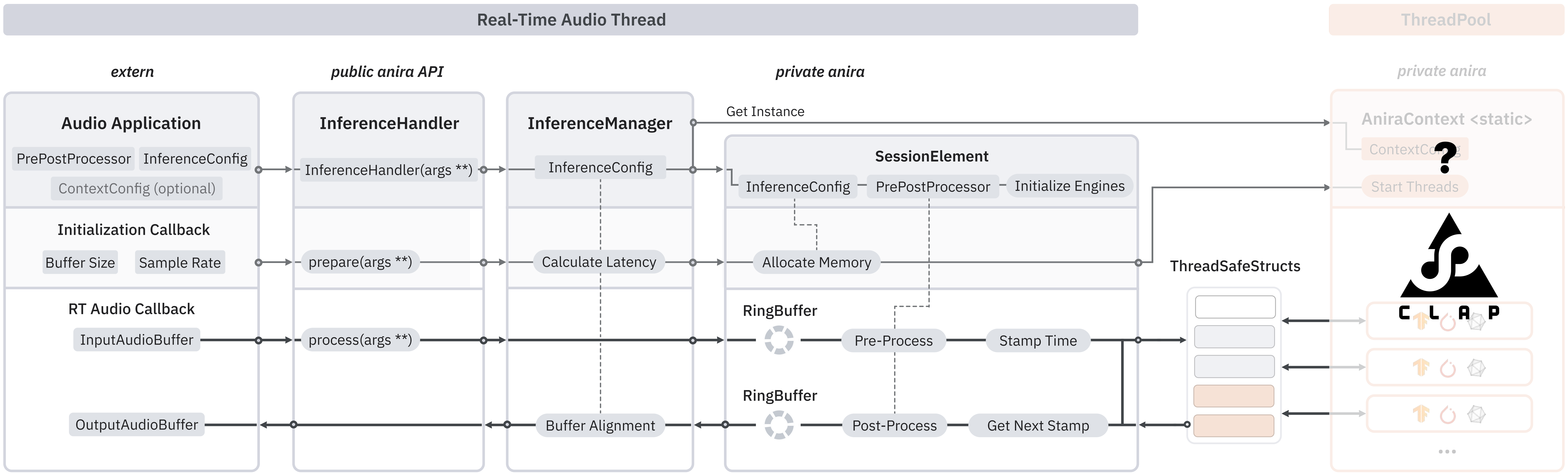


Safe Spin Locks

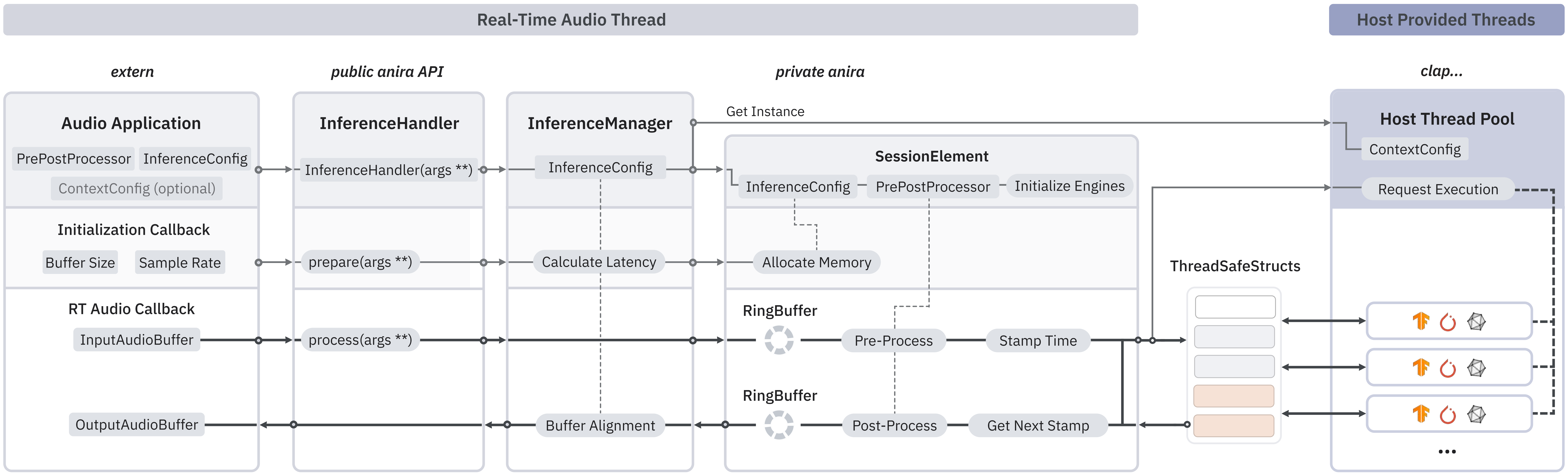
- Keeping threads alive without using all CPU resources
- The magic keyword is exponential backoff
- Great talk on this topic:



Host Provided Threads ?

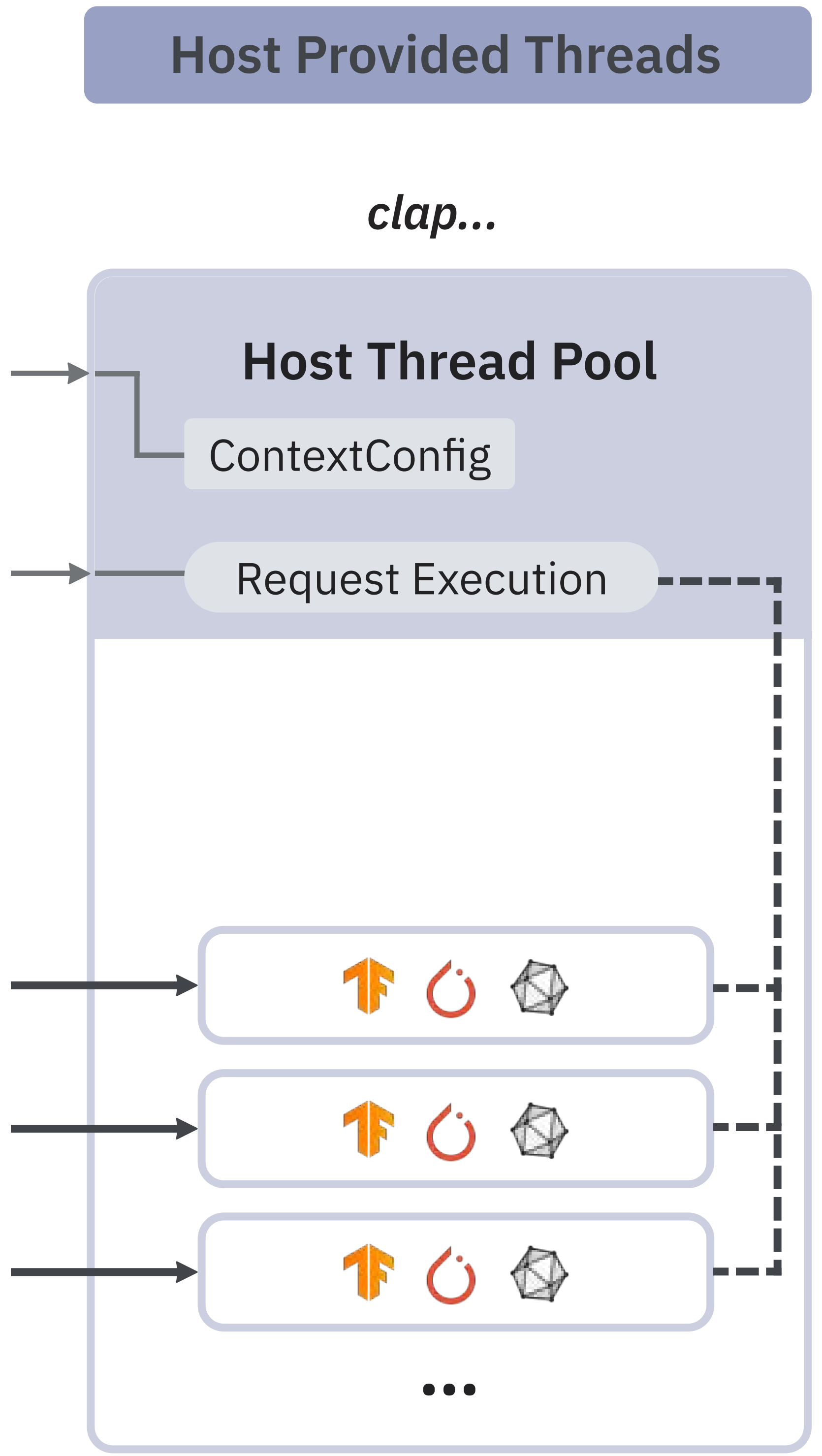


Host Provided Threads ?



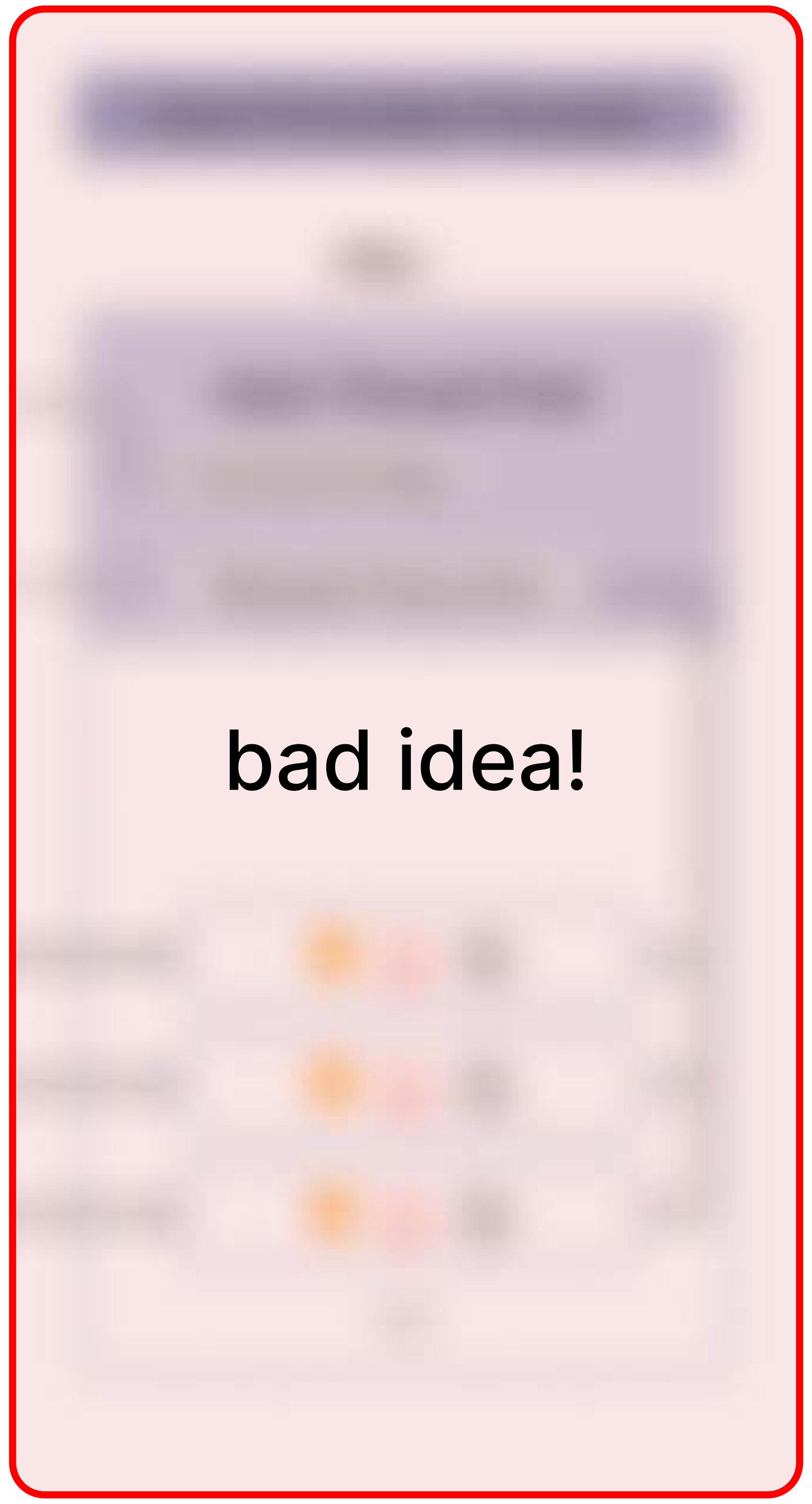
Host Provided Threads ?

```c
typedef struct clap_host_thread_pool {
   // Schedule num_tasks jobs in the host thread pool.
   // Will block until all the tasks are processed.
   // This must be used exclusively for realtime processing within the process call.
   // It can't be called concurrently or from the thread pool.
   // Returns true if the host did execute all the tasks, false if it rejected the request.
   // The host should check that the plugin is within the process call, and if not,
   // reject the exec request.
   // [audio-thread]

   bool(CLAP_ABI *request_exec)(const clap_host_tQ*host, uint32_t num_tasks);
} clap_host_thread_pool_t;
```



Host Provided Threads ?

```c
typedef struct clap_host_thread_pool {
   // Schedule num_tasks jobs in the host thread pool.
   // Will block until all the tasks are processed.
   // This must be used exclusively for realtime processing within the process call.
   // It can't be called concurrently or from the thread pool.
   // Returns true if the host did execute all the tasks, false if it rejected the request.
   // The host should check that the plugin is within the process call, and if not,
   // reject the exec request.
   // [audio-thread]

   bool(CLAP_ABI *request_exec)(const clap_host_tQ*host, uint32_t num_tasks);
} clap_host_thread_pool_t;
```



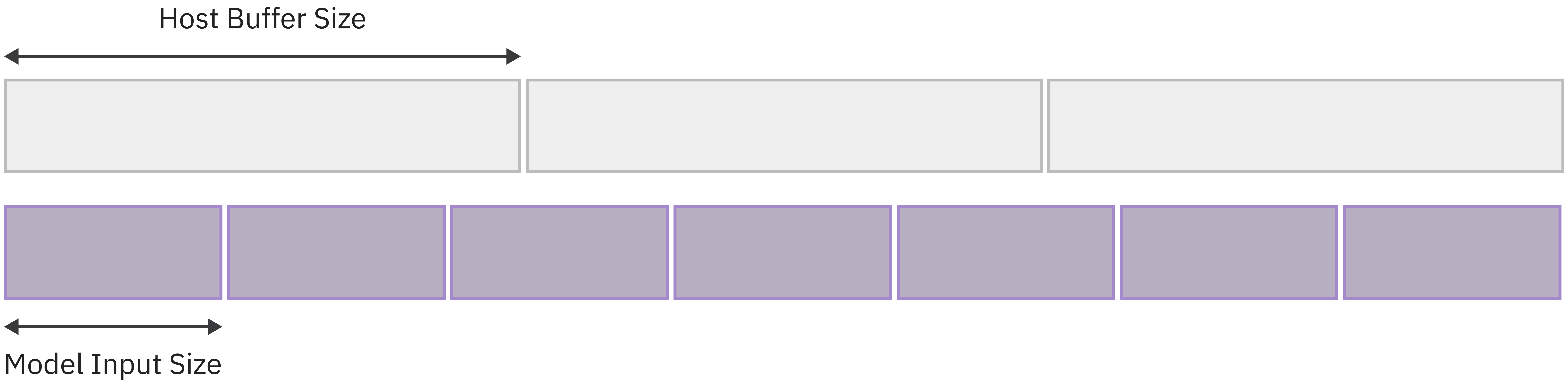
Latency Influences

Host Buffer Size	Model Input Size	Model Latency	Max Inference Time
varying	fixed	fixed	varying
known at run-time	known at compile-time	known at compile-time	system dependent

Latency

We often face situations where the host and model input size don't match

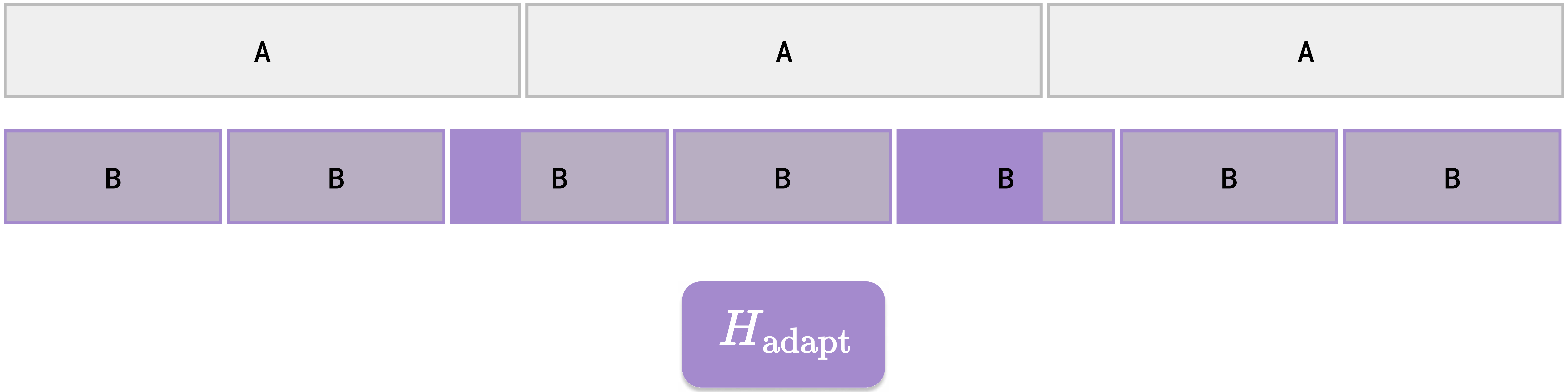
→ leaving a remainder when divided



Latency

To minimize latency we use Callback adaptation techniques: (Letz, 2001)

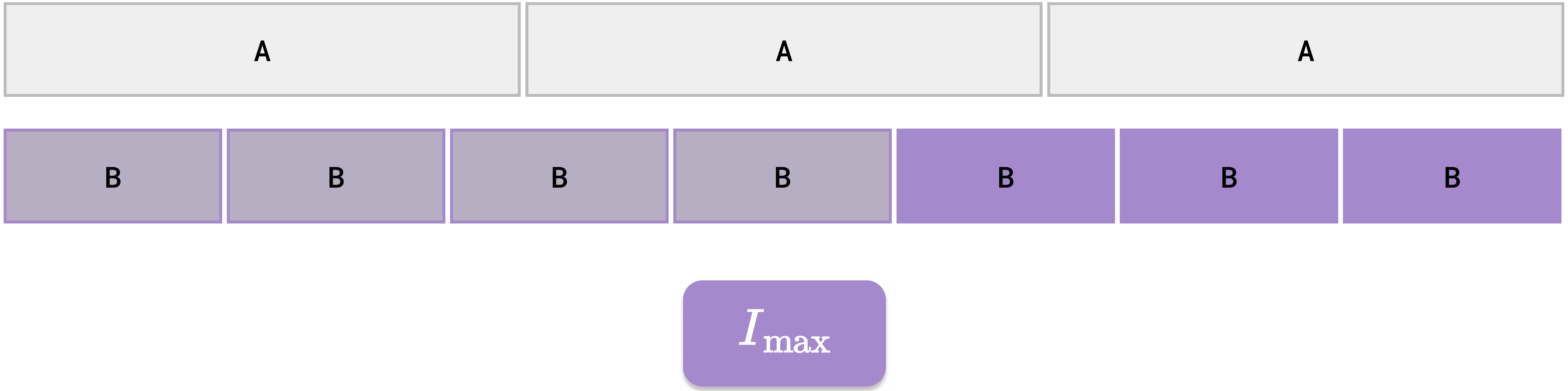
1. Calculate the largest remainder of nA divided by B until a pattern repeats



Letz, S. (2001). *Callback adaptation techniques* (Doctoral dissertation, GRAME).

Latency

2. Calculate the maximum number of inferences for one buffer A
3. Multiply this number with the worst case execution time of one inference



Latency

$$L_{\text{total}} = H_{\text{adapt}} + \left\lceil \frac{I_{\text{max}}}{H_{\text{host}}} \right\rceil \cdot H_{\text{host}} + M_{\text{int}}$$

L_{total} Total Latency

H_{adapt} Largest Remainder

I_{max} Max. Inference Time

H_{host} Host Buffer Size

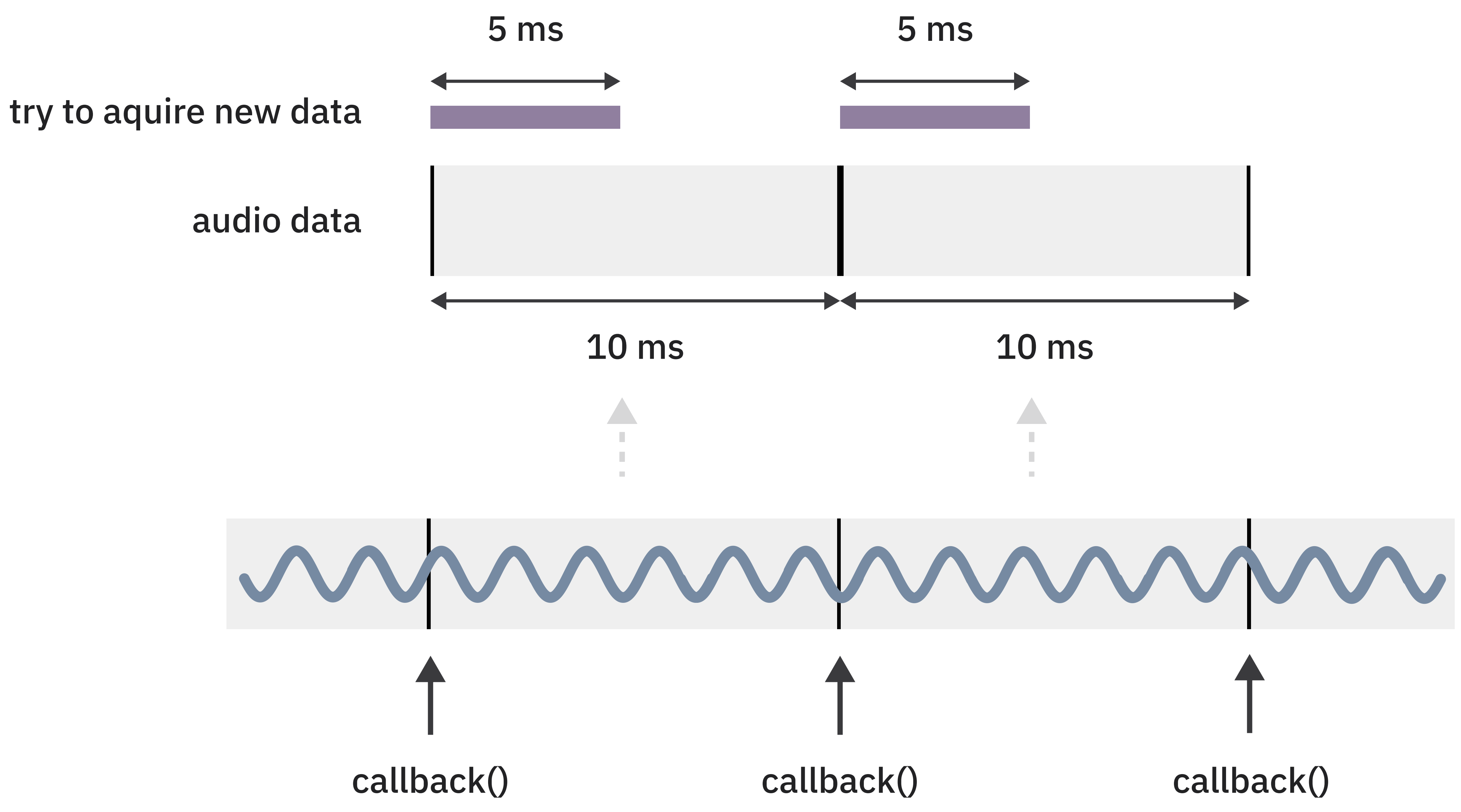
M_{int} Model Latency

Controlled Blocking Operation

Optional solution to further reduce the latency:

`std::binary_semaphore::try_acquire_until`

- Controlled blocking operation
- Maximizes the time available to receive data from inference threads
- Disabled by default as it is highly controversial
- Useful especially for scenarios with only one instance

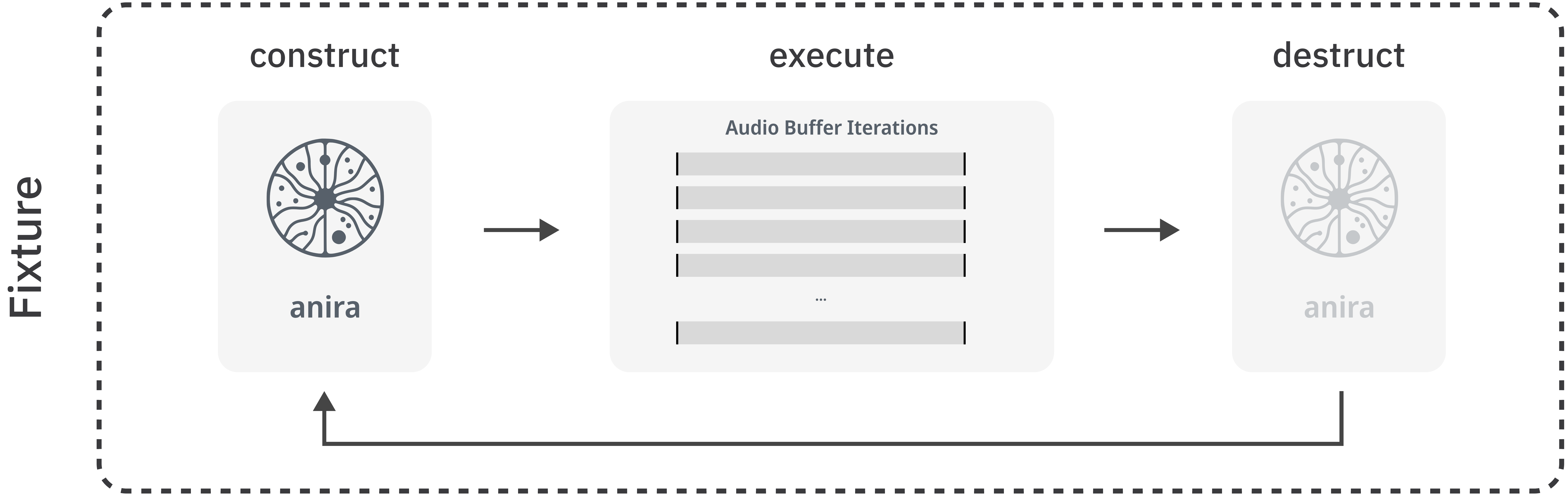


Chapter V

Impact on Inference Runtimes

Various Factors Affecting Performance

Benchmarking



Arg1: Buffer Size
[64, 128, ..., 2048]

Arg2: Backend
[ONNX, TFLITE, LIBTORCH]

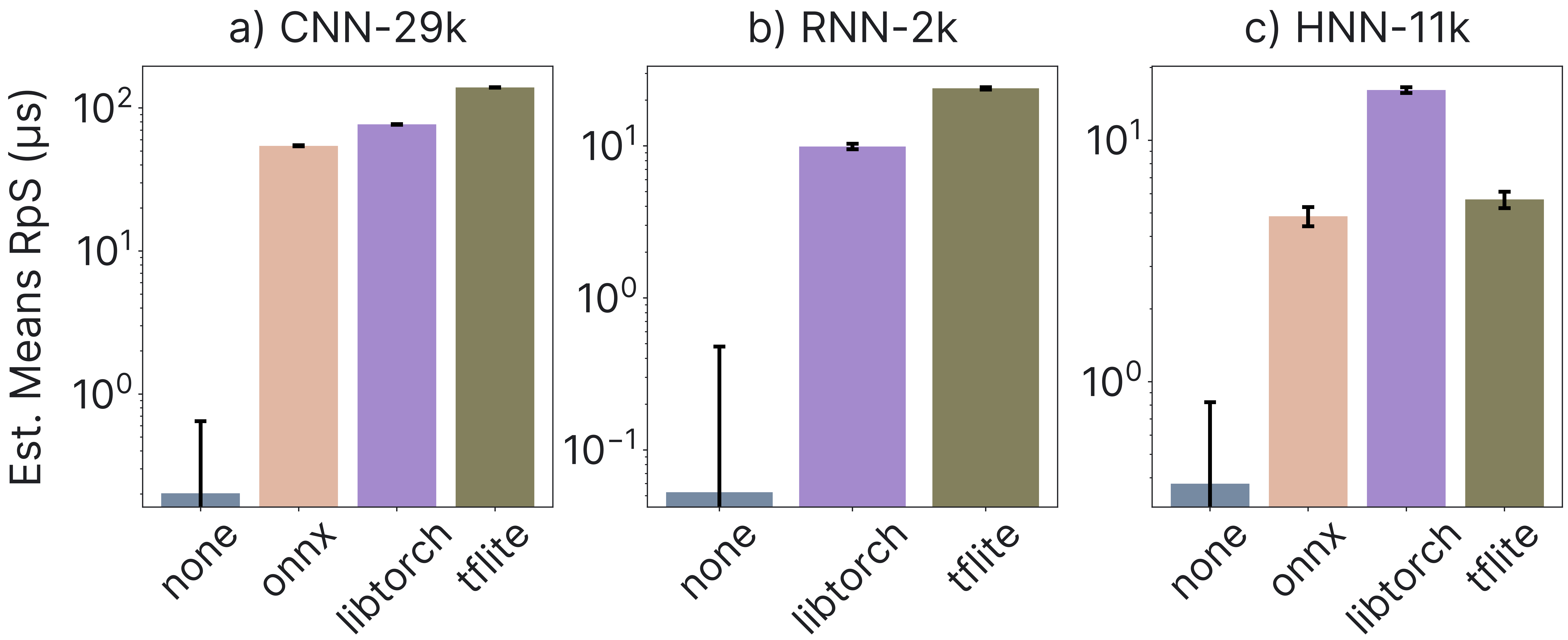
Arg3: Model
[CNN, RNN, HYBRID]

Arg4: Repetition
[10]

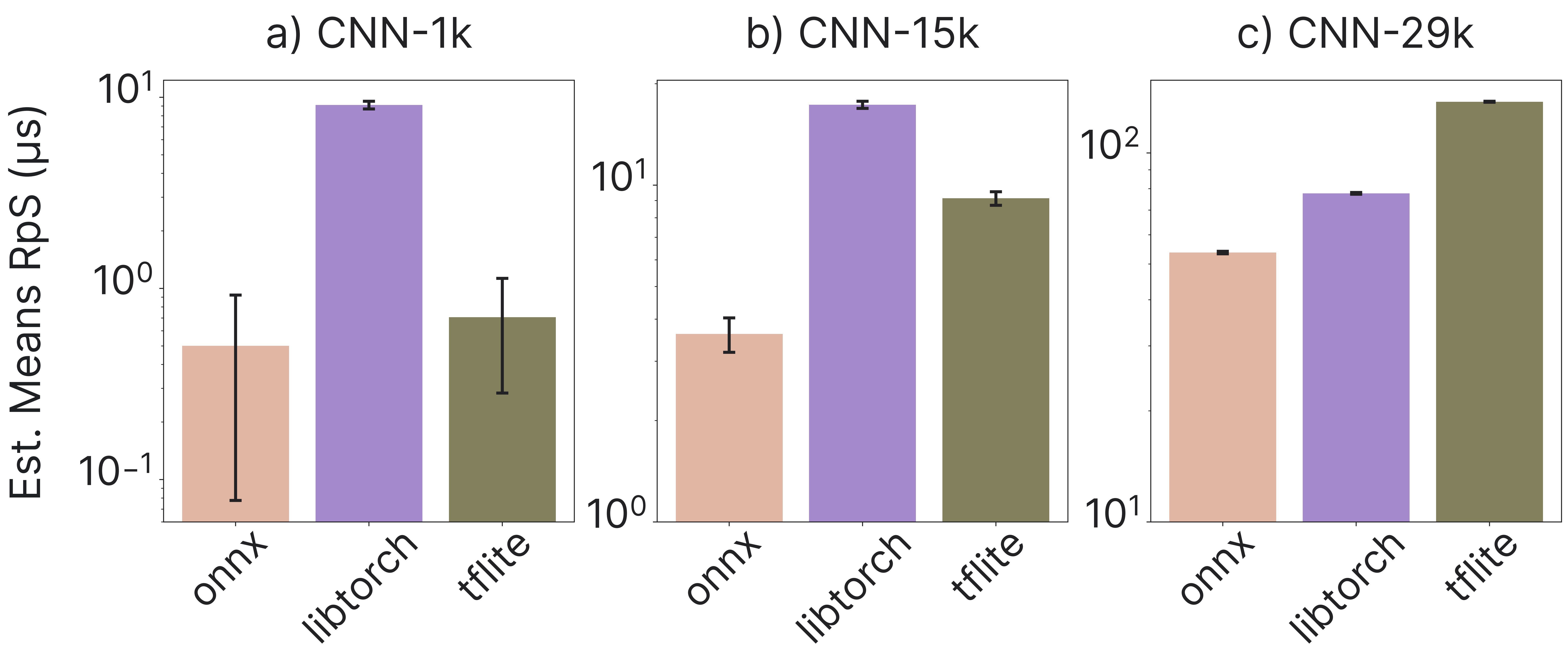
Measurements

system	*engine*	*model*	*buffer size*	*repetition*	*iteration*
level 1	level 2	level 3	level 4	level 5	level 6
Linux – Intel MacBook	OnnxRuntime	HNN-11k	64	1	1
MacOS – M1 MacBook	Libtorch	RNN-2k	128	2	2
Windows – HP ZBook	TFLite	CNN-29k	...	...	...
	Bypass-Engine	CNN-15k	8192	10	50
		CNN-1k			

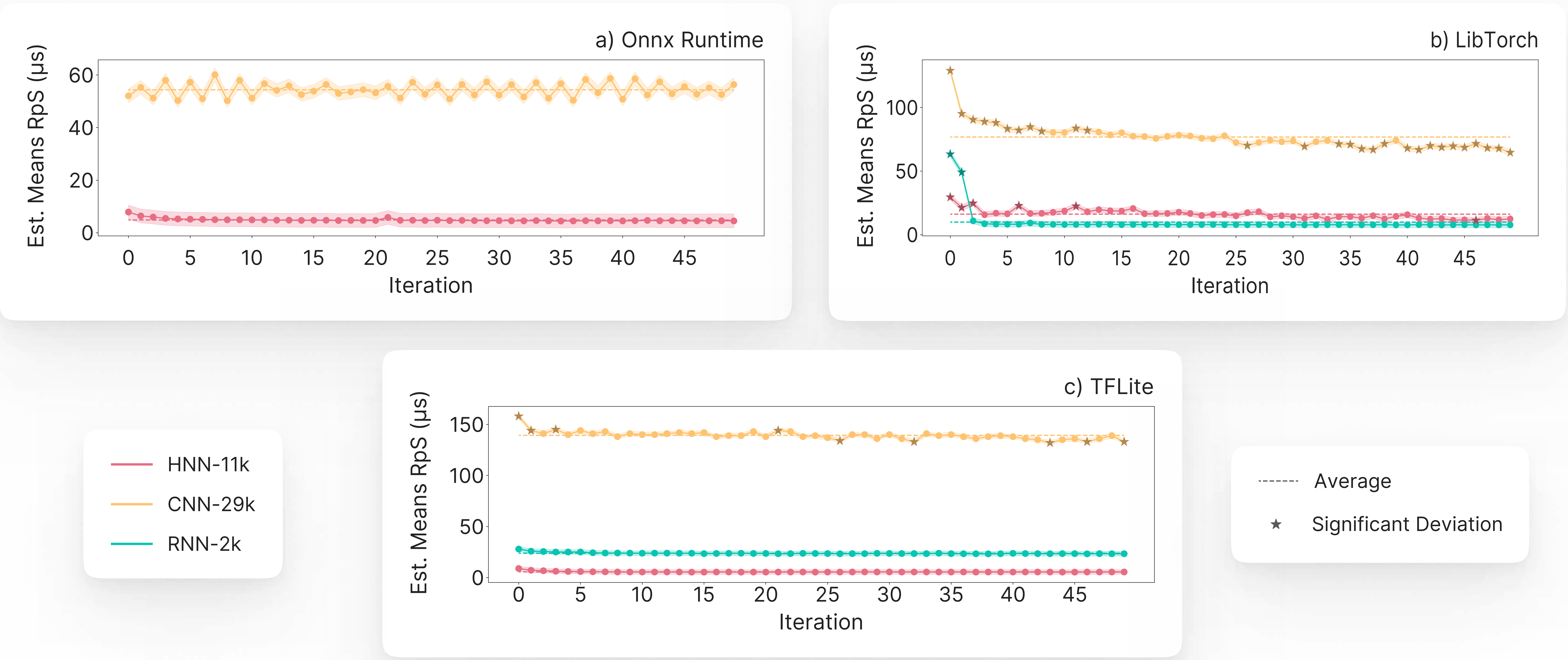
Inference Engine Comparison



CNN Hyperparameter Comparison

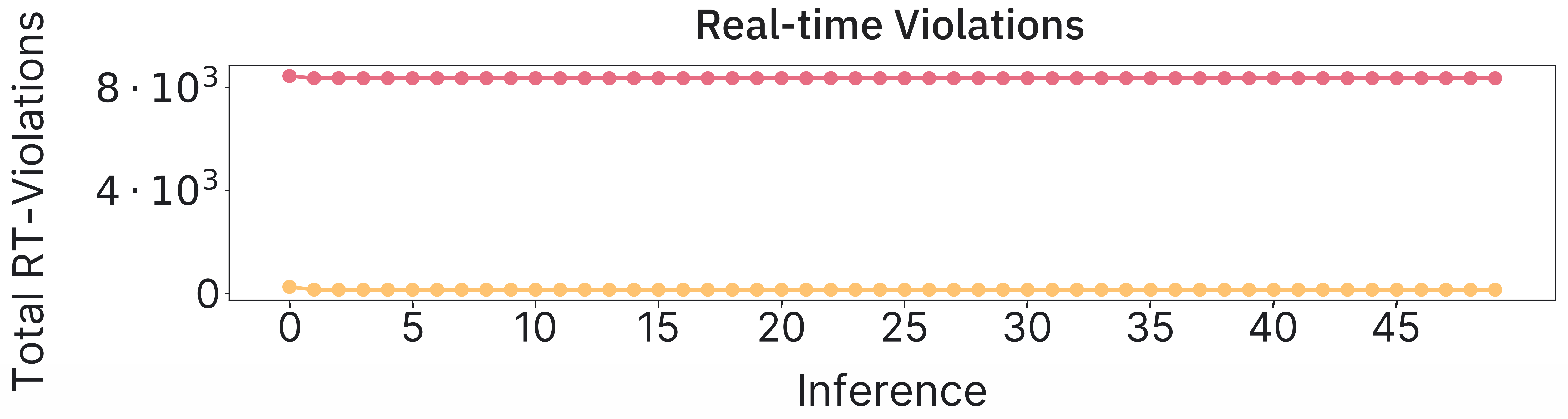


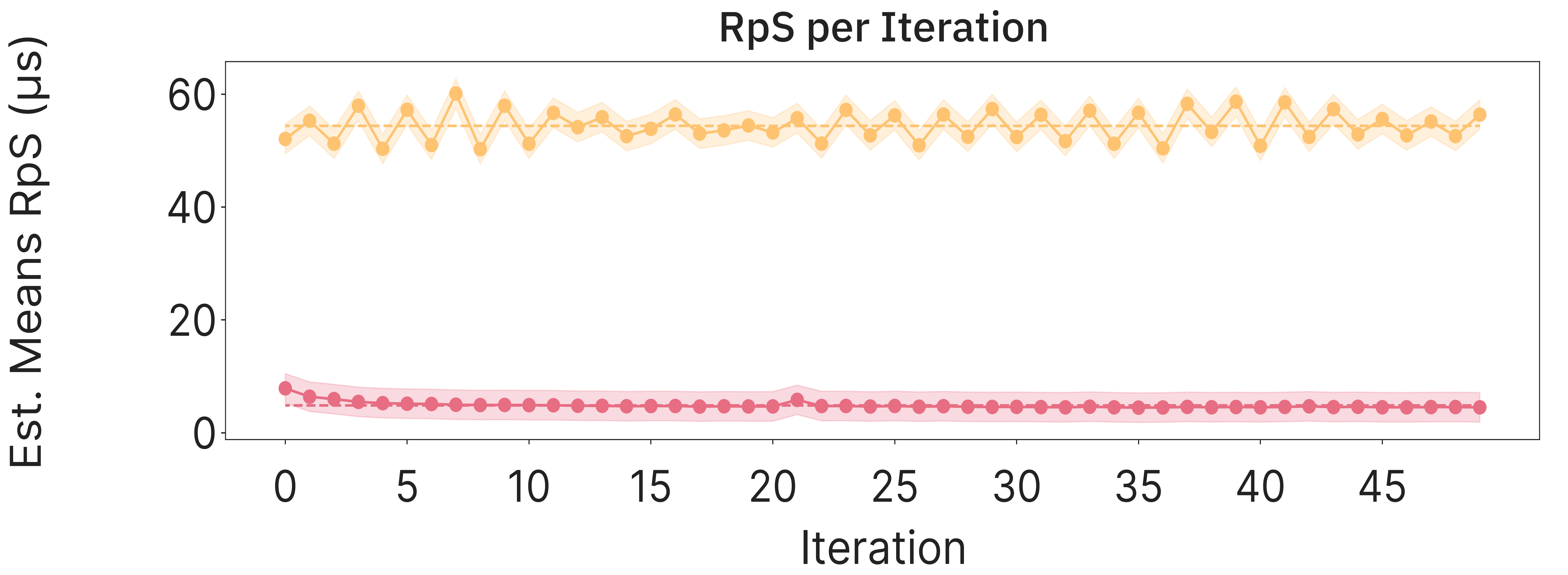
Influence of the Iteration

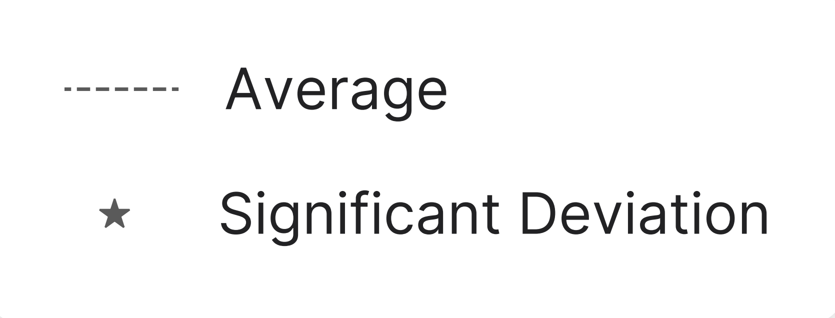


Violations vs. Runtime Performance - LibTorch

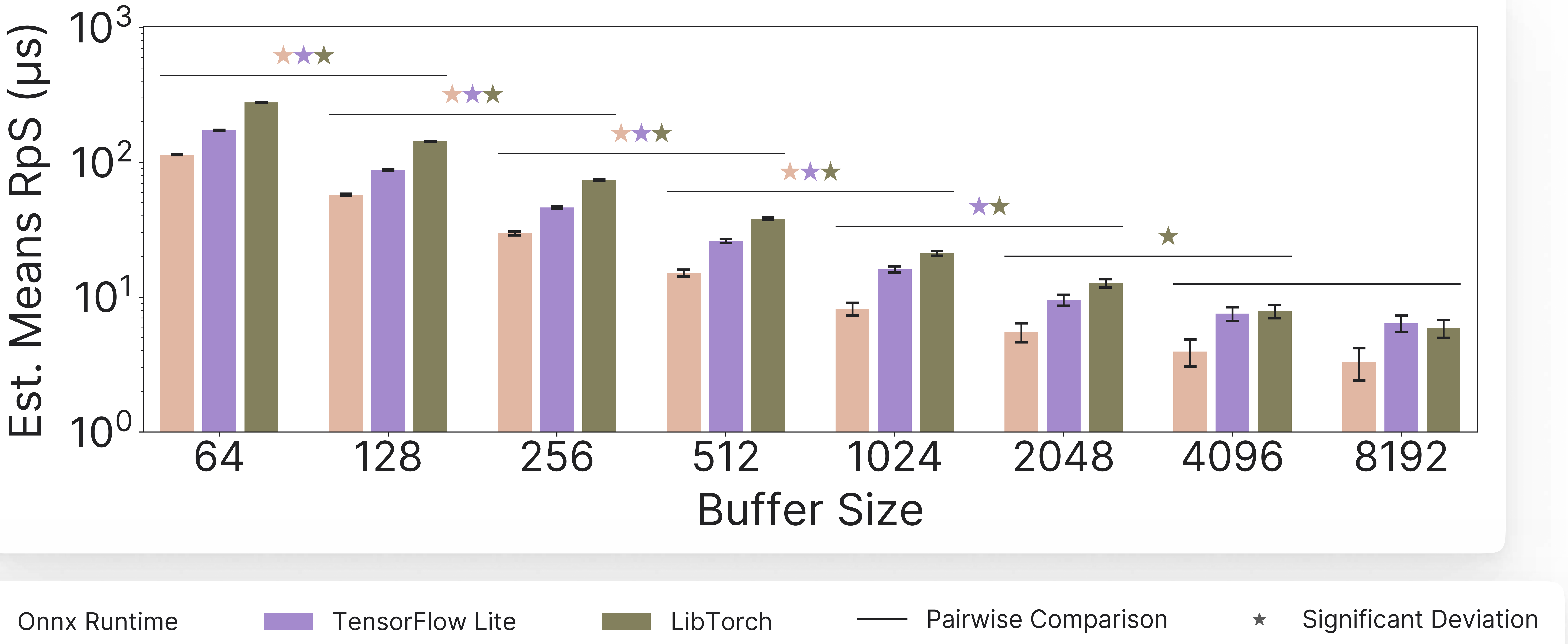
Violations vs. Runtime Performance - Onnx Runtime







Influence of the Buffer Size



Chapter VI

Conclusion

Summary, Getting Started, Questions

Library Feature Overview

Thread-Safe Inference Engine Wrapper Library:

- Supports major inference engines
- Compatible with various neural network types
- Enables handling of multiple session
  - Across different plugins (and formats) with multiple instances
  - Across various neural network types

C++ Cross Platform Library:

- available on Windows (x64), Linux (x64, aarch64, armv7l), macOS (x64, arm64)

Permissive License: Apache 2.0

Objectives of the Library

Implementing real-time safe inference requires expertise across various domains

The library aims to simplify real-time safe inference implementation

Getting Started

We provide five example neural networks, each featuring:

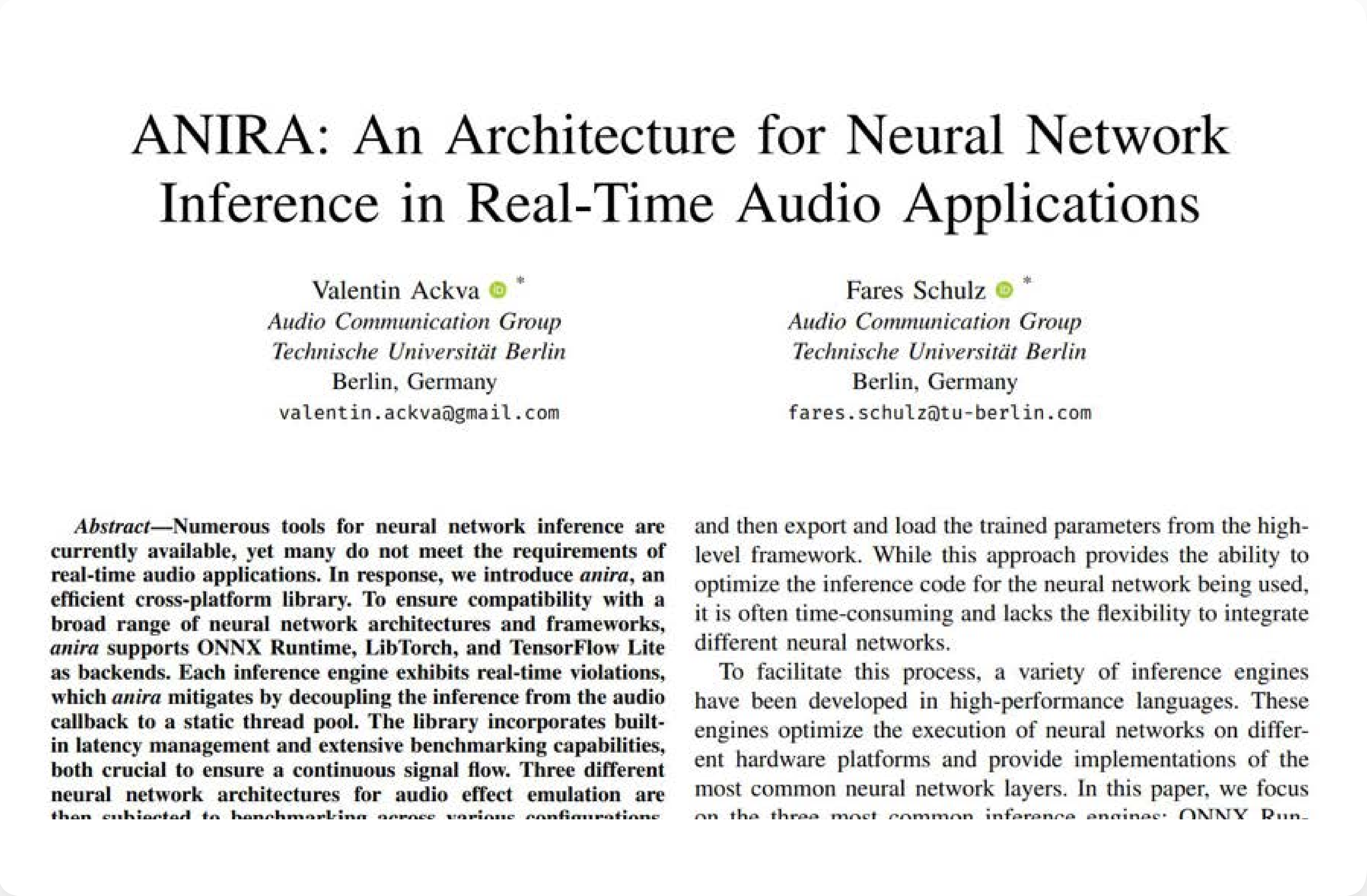
- ML implementation
- Training code and data
- Pre-trained models

Every example can be adapted into a real-time application, available as:

- Audio plugin using the JUCE framework
- Native CLAP plugin, Bela embedded application
- Soon: JACK client, Max external

Additionally, we include example benchmarks to validate runtimes

Paper



ANIRA: An Architecture for Neural Network Inference in Real-Time Audio Applications

Valentin Ackva *
Audio Communication Group
Technische Universität Berlin
Berlin, Germany
valentin.ackva@gmail.com

Fares Schulz *
Audio Communication Group
Technische Universität Berlin
Berlin, Germany
fares.schulz@tu-berlin.com

***Abstract*—Numerous tools for neural network inference are currently available, yet many do not meet the requirements of real-time audio applications. In response, we introduce *anira*, an efficient cross-platform library. To ensure compatibility with a broad range of neural network architectures and frameworks, *anira* supports ONNX Runtime, LibTorch, and TensorFlow Lite as backends. Each inference engine exhibits real-time violations, which *anira* mitigates by decoupling the inference from the audio callback to a static thread pool. The library incorporates built-in latency management and extensive benchmarking capabilities, both crucial to ensure a continuous signal flow. Three different neural network architectures for audio effect emulation are then subjected to benchmarking across various configurations**

and then export and load the trained parameters from the high-level framework. While this approach provides the ability to optimize the inference code for the neural network being used, it is often time-consuming and lacks the flexibility to integrate different neural networks.

To facilitate this process, a variety of inference engines have been developed in high-performance languages. These engines optimize the execution of neural networks on different hardware platforms and provide implementations of the most common neural network layers. In this paper, we focus on the three most common inference engines: ONNX Run-

https://doi.org/10.1109/IS262782.2024.10704099

Repository

anira

an architecture for neural network inference
in real-time audio applications

https://github.com/anira-project/anira

Thank you for listening

Do you have questions?

Repository:

Audio Communication Group
Technische Universität Berlin