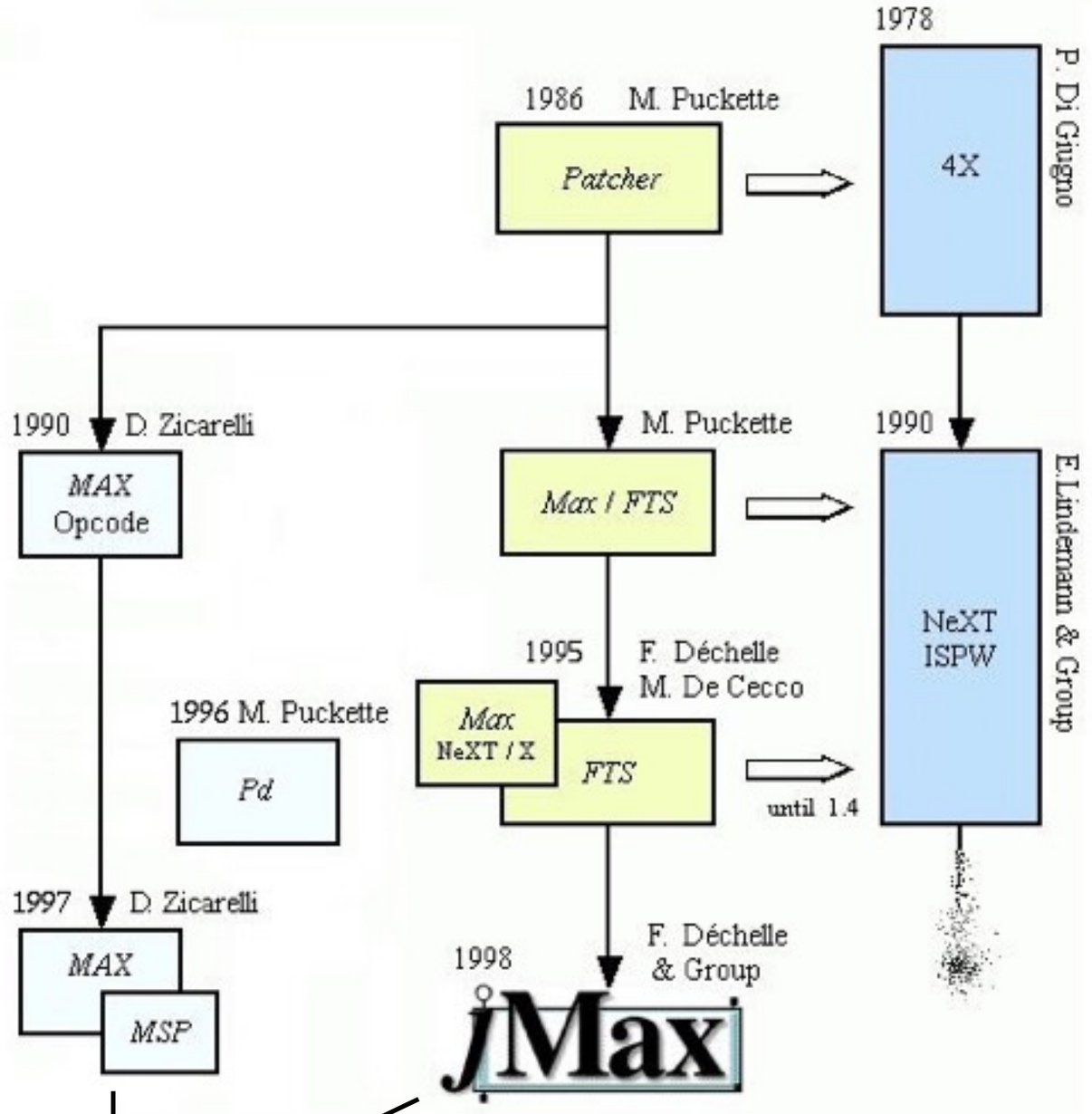
MuBu — Multi-Buffer Package for Max

MuBu Introduction

- Development lead by Norbert Schnell 2009–2017
  - collaborators:
    Riccardo Borghesi, Diemo Schwarz, Jean-Philippe Lambert, Frédéric Bevilacqua, ISMM team
- available on Max package manager or Ircam Forum (most up to date, currently v1.9.11):
  https://forum.ircam.fr/projects/detail/mubu/
- more info: **http://ismm.ircam.fr/mubu**
- support on Ircam Forum discussion groups

History

from *A brief history of MAX*
(François Dechelle)



FTM, MnM, Gabor N. Schnell & team 2003

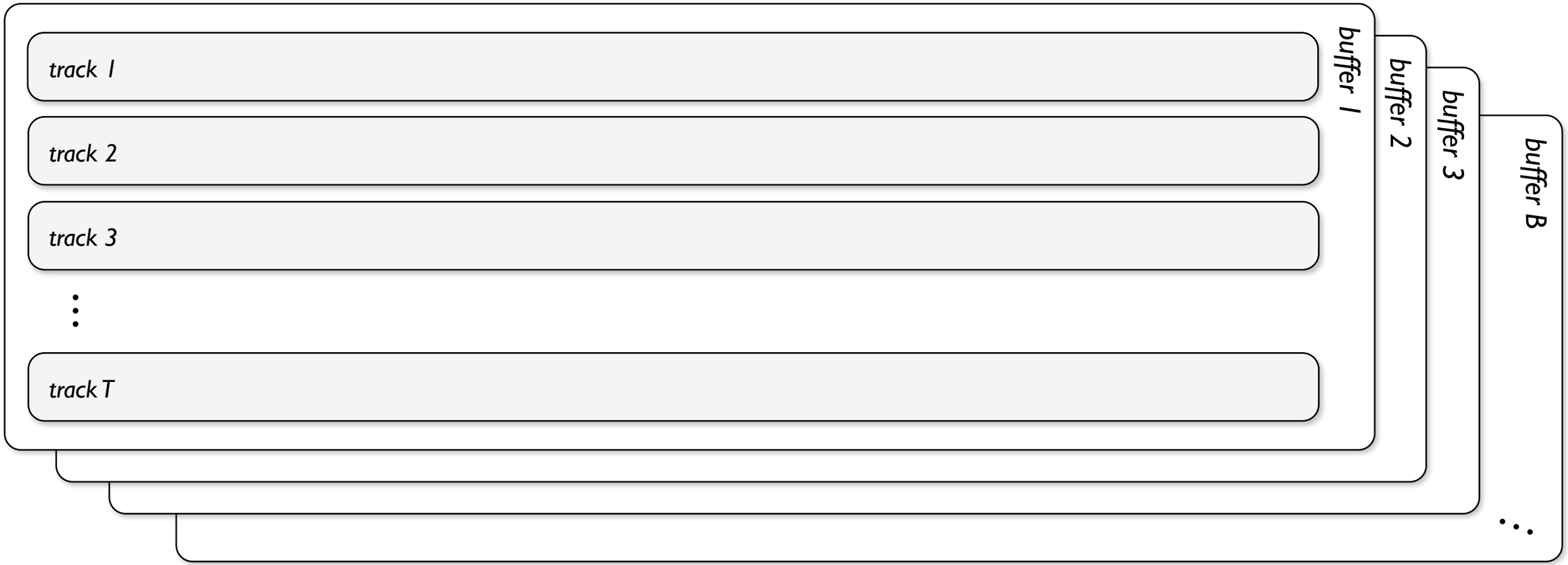
MuBu N. Schnell & team 2010

Motivation

- Create a solid and open framework for the experimentation with recorded data streams of multiple representations in Max/MSP
  - audio samples
  - audio descriptors
  - gesture and motion capture data
  - spectral audio representations
  - symbolic representations
  - segmentation and annotations

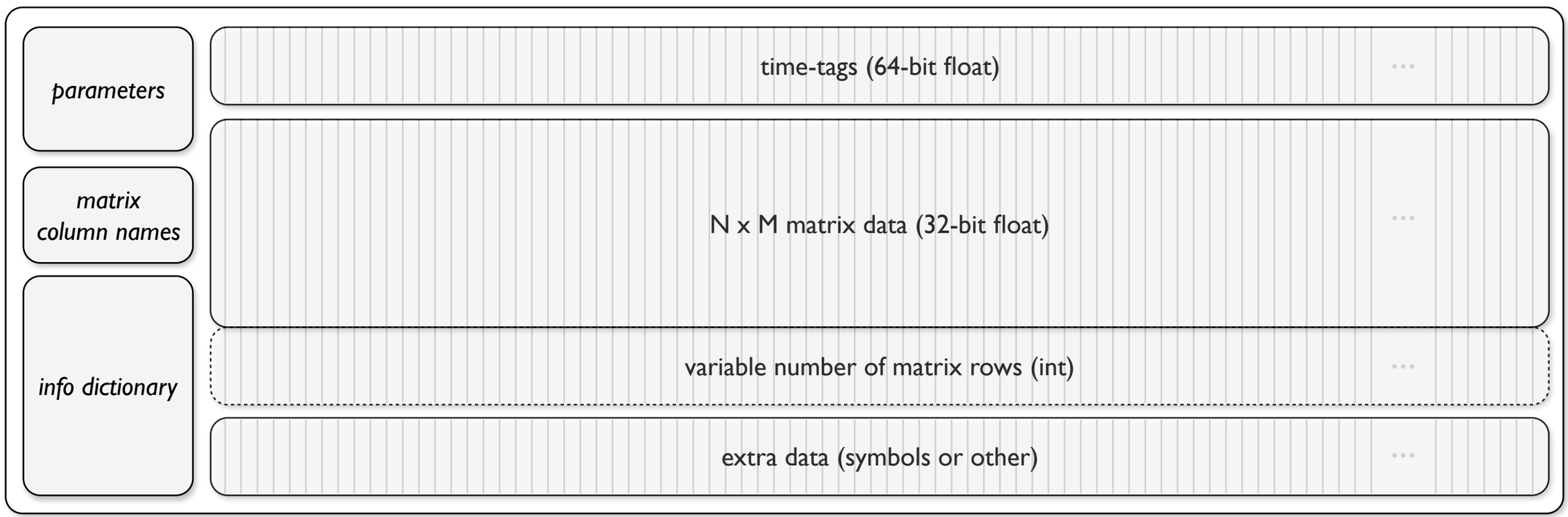
Container Data Structure

- Array of *buffers*
  - each buffer has the same array of *tracks*



Track Data Structure

- Array of *frames,* with time-tags or constant frame rate
  - each frame contains:
    - 2D matrix data with named columns,
      optional: variable number of rows
    - “extra” data (labels)
  - Track meta-data

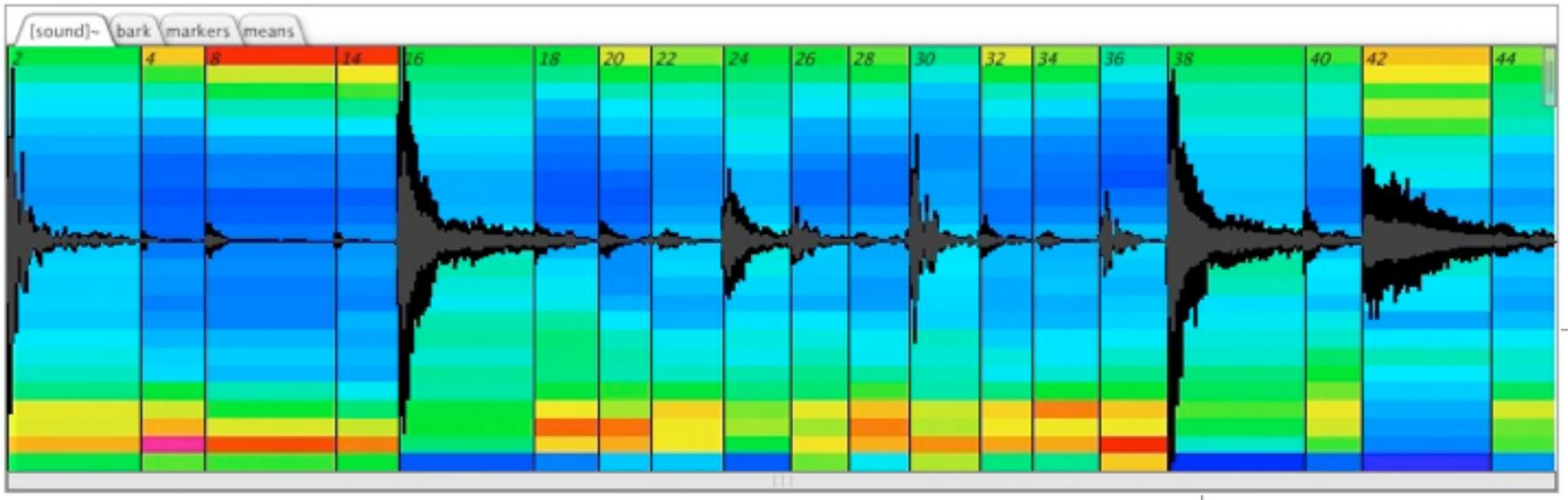


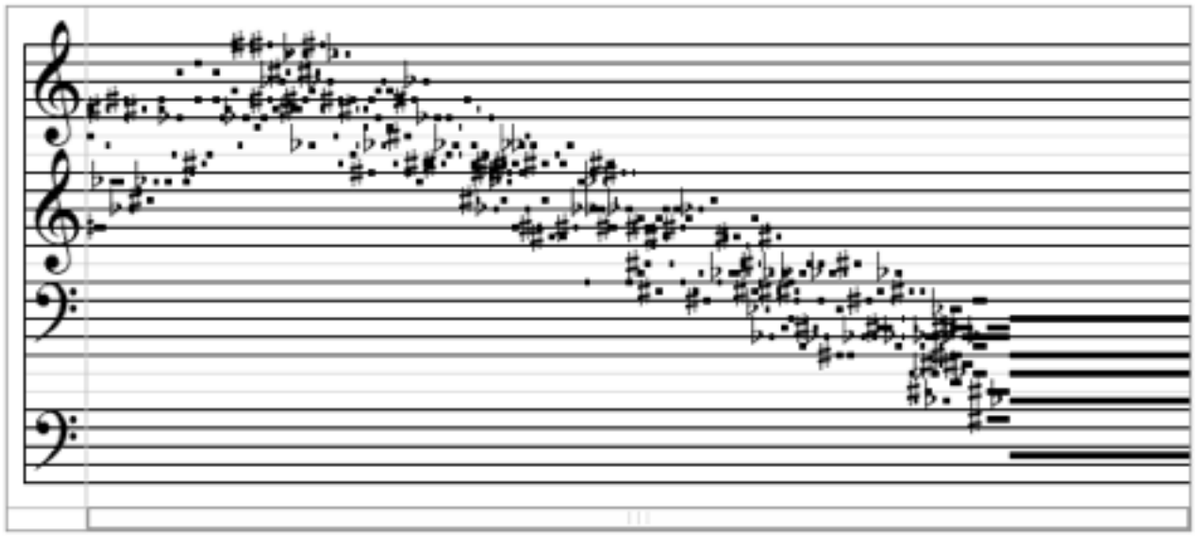
File Formats

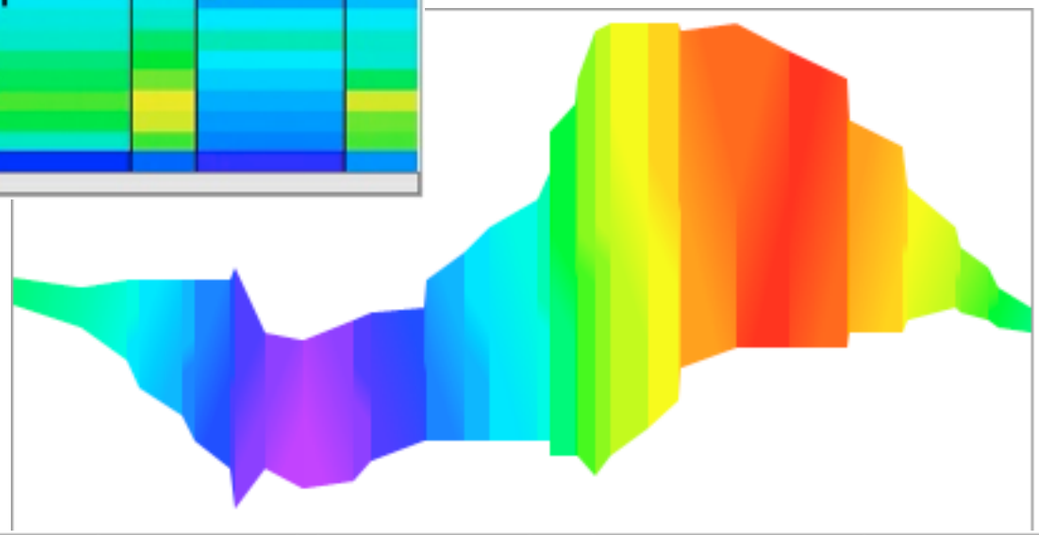
- SDIF import and export
- Plain text data, markers and labels file import and export
- MIDI standard file import
- MusicXML import
- Save/load data with Max patcher (for small amounts of data)

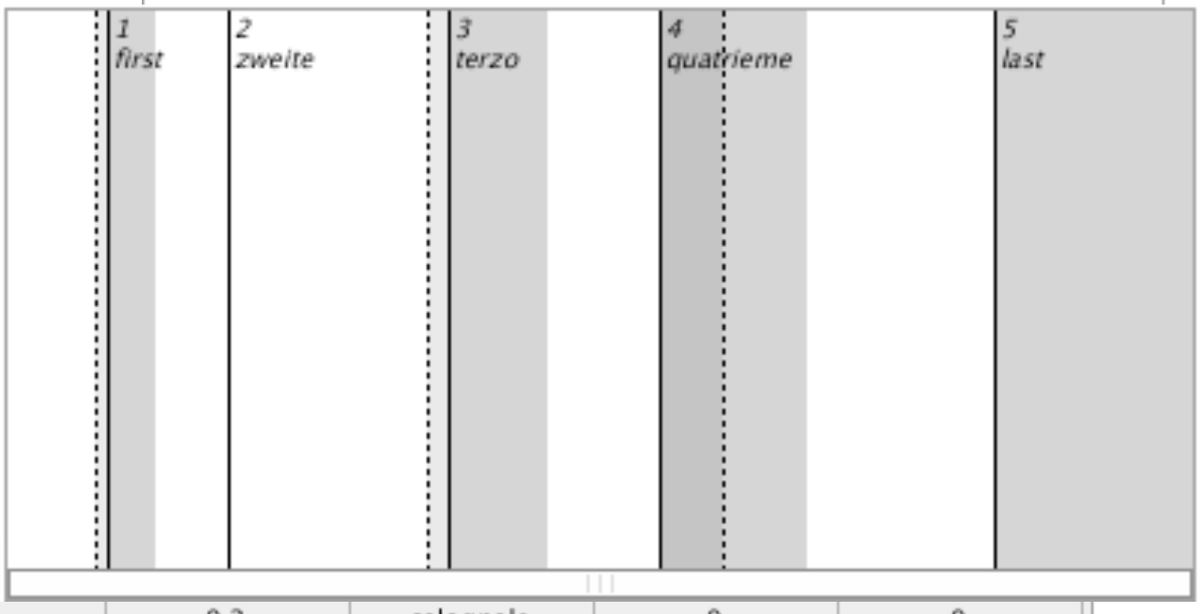
Visualization and Editing

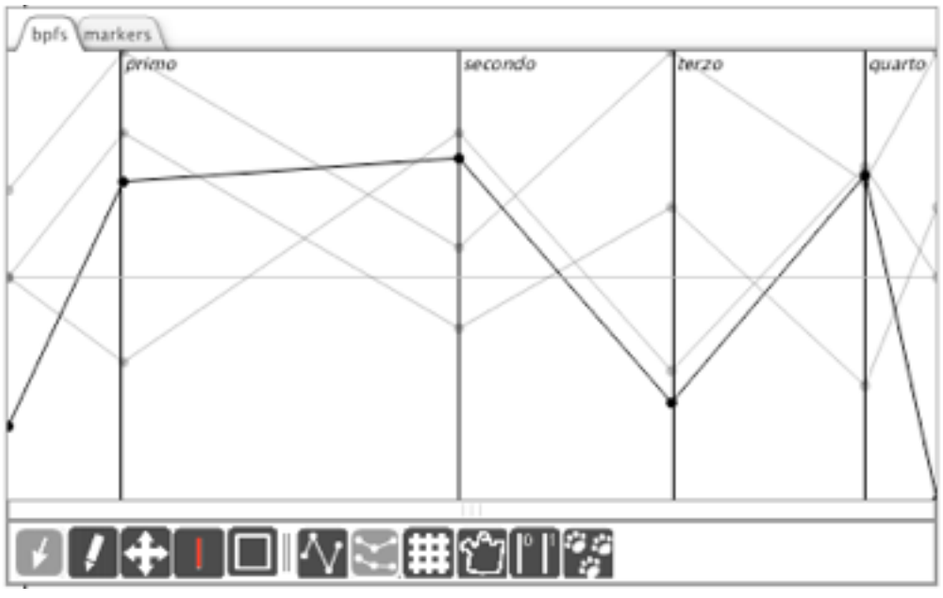
- Based on the Juce framework
- Set of editor/visualization components
  - waveform (single or multi-channel)
  - break-point function (single or also multi-channel)
  - sonogram (of sampled or time-tagged data)
  - markers (with duration and offset)
  - textual tables/matrices
  - piano roll (or simplified staffs)
  - traces (waveform or bpf with color and thickness)
  - Control components (scroll bar, rulers, toolbar, tabs, buffer chooser, etc.)





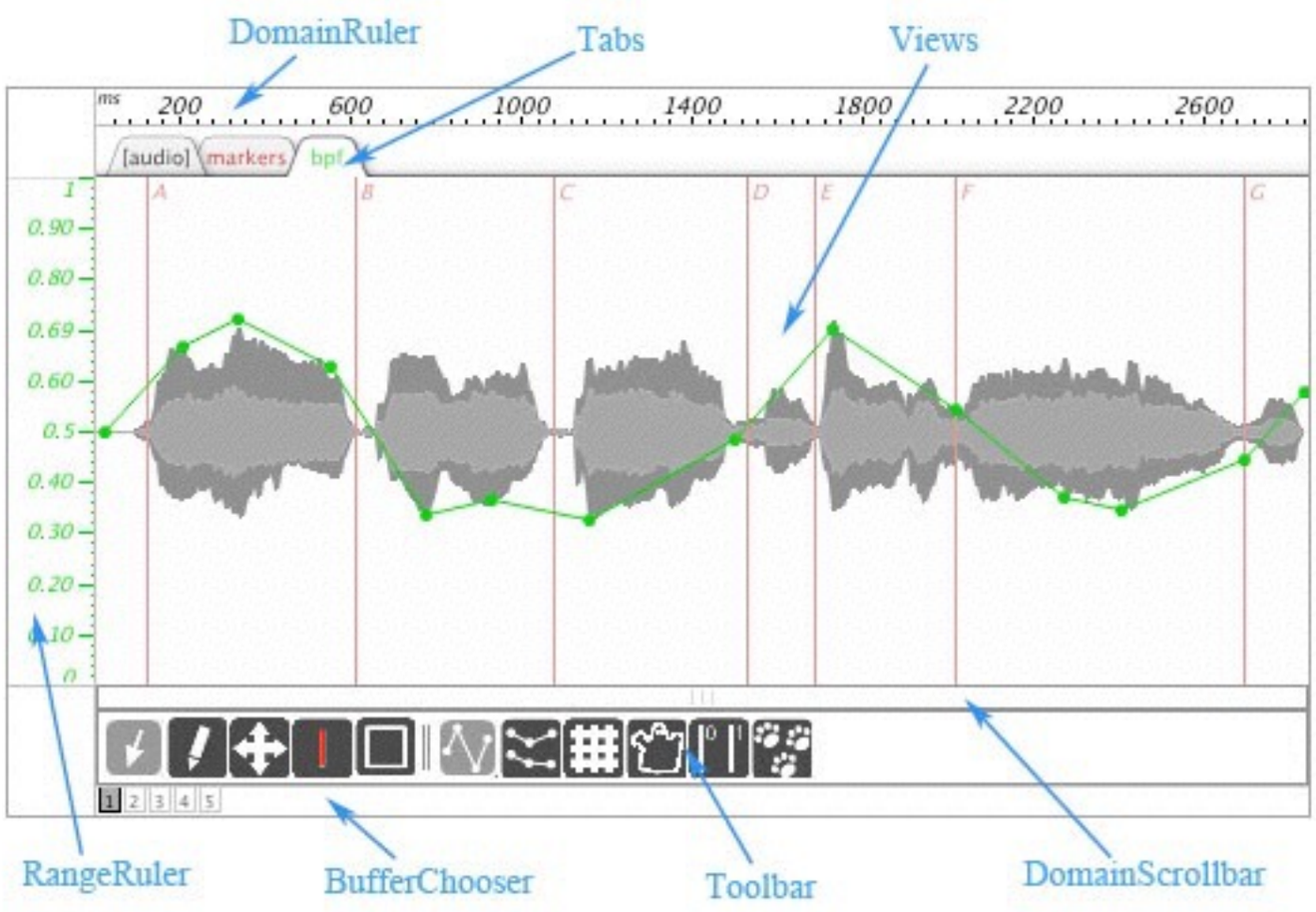




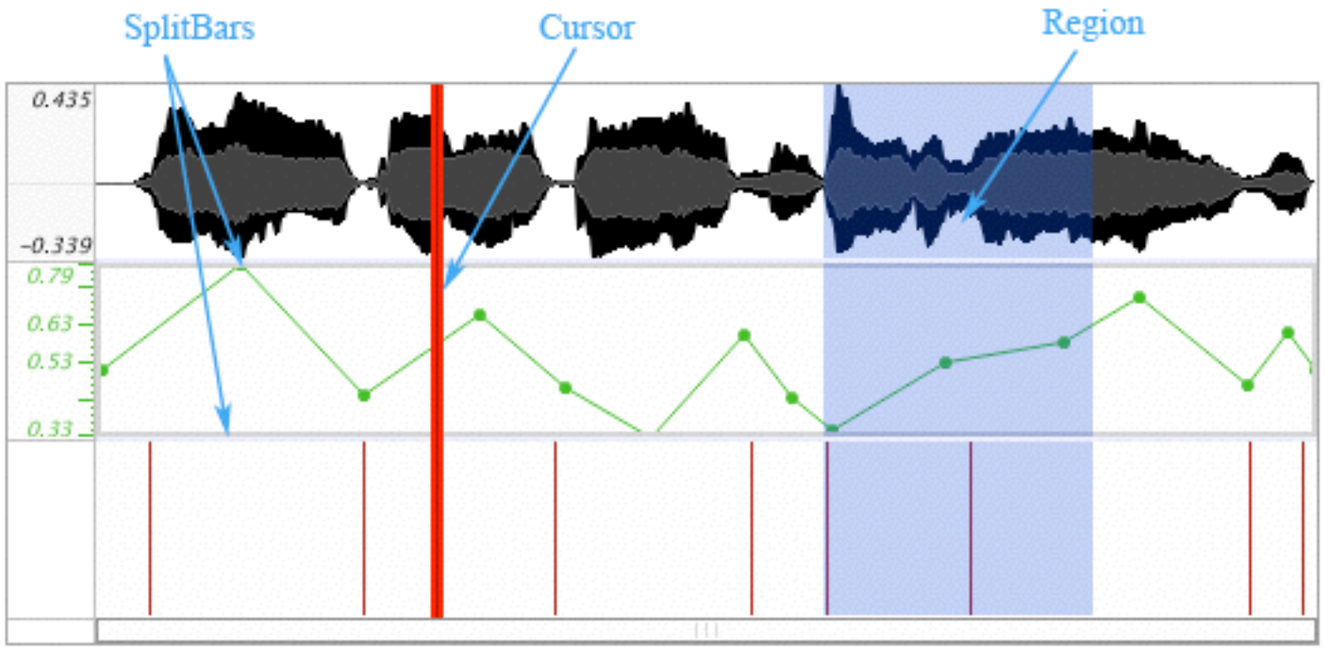


	0				
0	1	0.2	colognole	0	0
1	0	bpf [#1]	0	0	0
2	0	0	fmat [#1]	0	0
3	123	0	maggiorana	0	cotoletta
4	0	0	0	0	0

Editor Control Components (1/2)



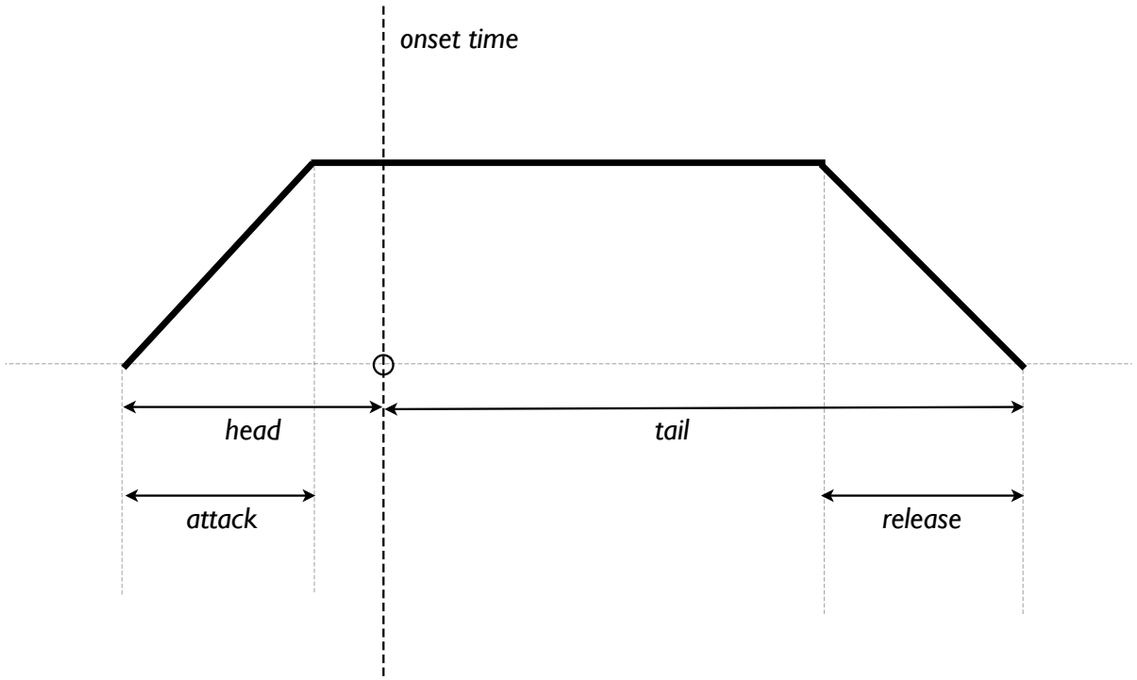
Editor Control Components (2/2)

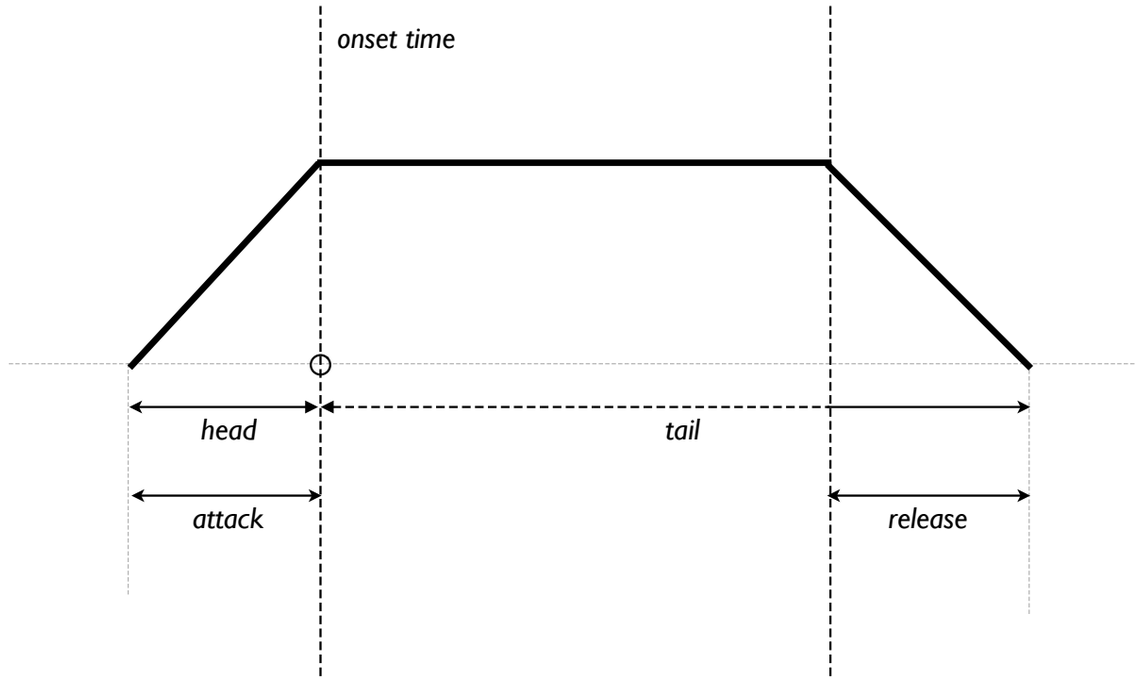


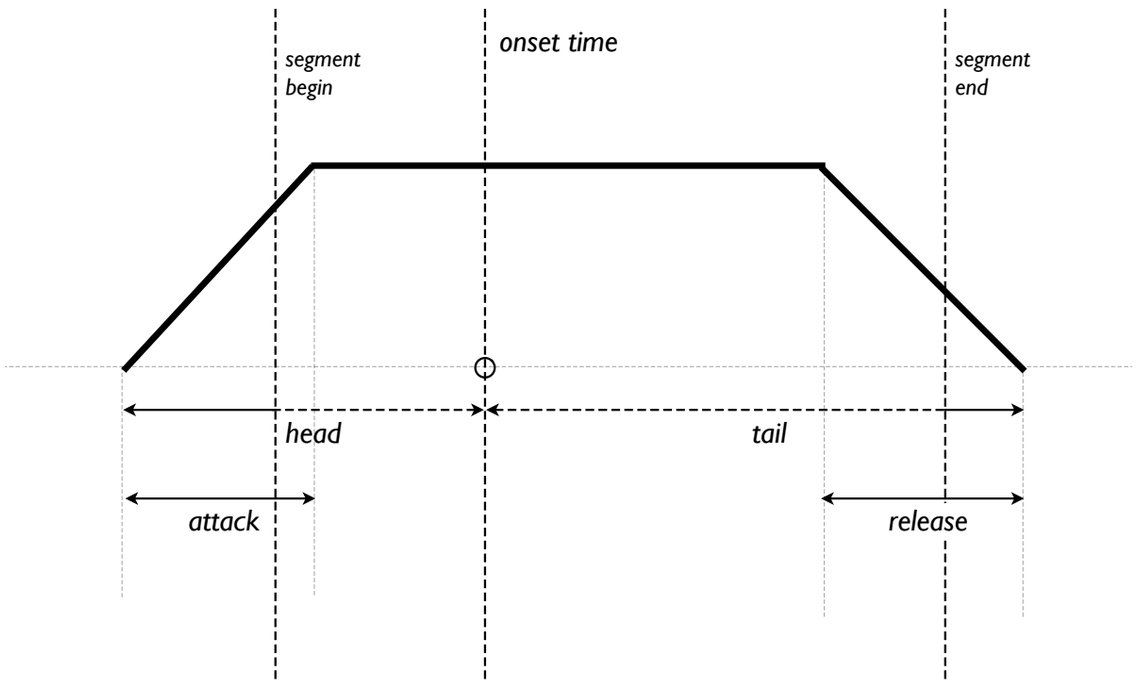
Synthesis

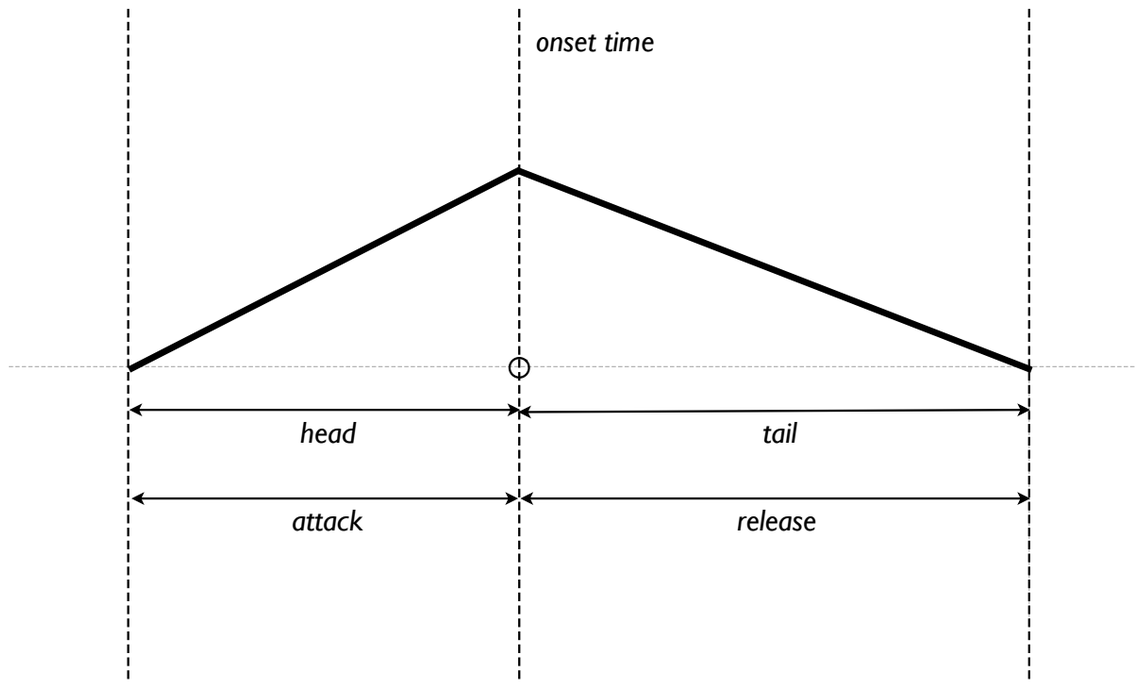
- **mubu.granular~** granular synthesis, PSOLA
- **mubu.concat~** segment-based granular and concatenative synthesis
- **mubu.play~** general audio and data playback
- **mubu.additive~** FFT-1 additive synthesis

Granular Envelopes and Markers

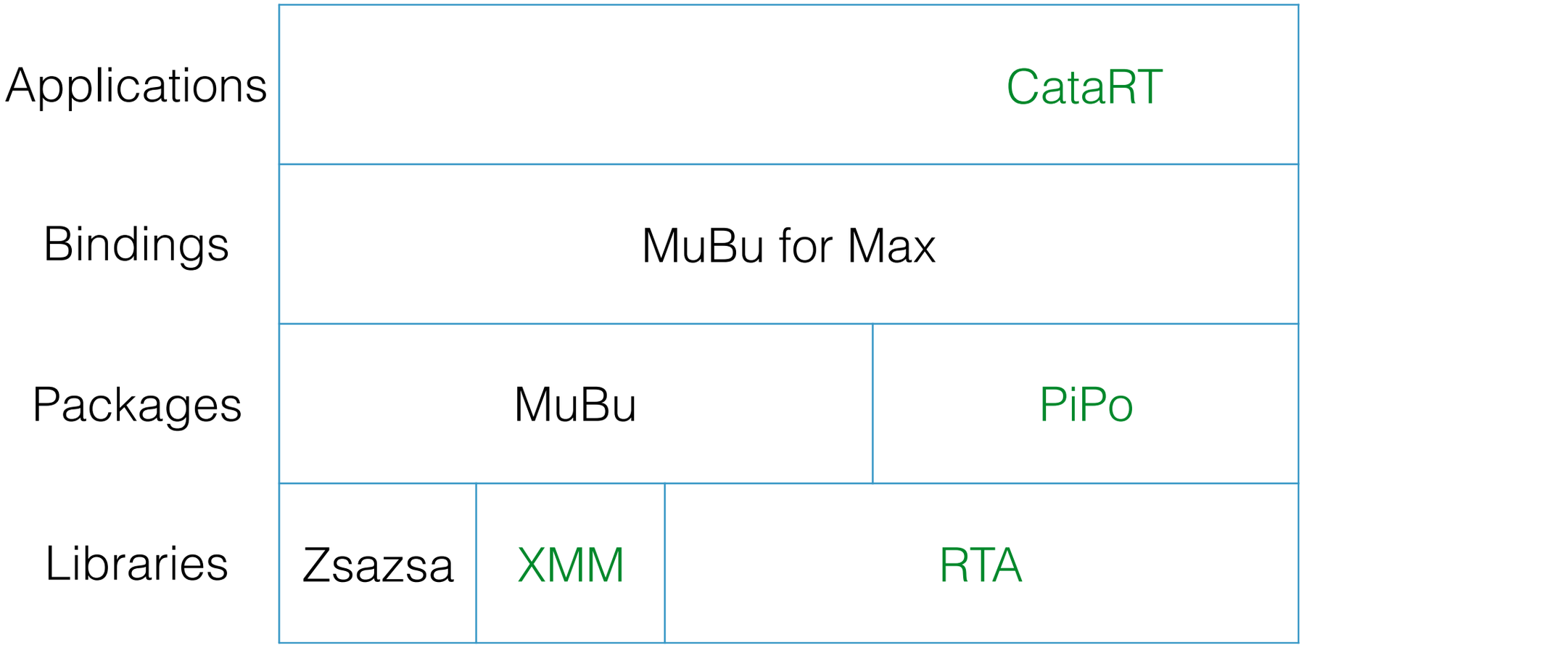








Components



Open Source

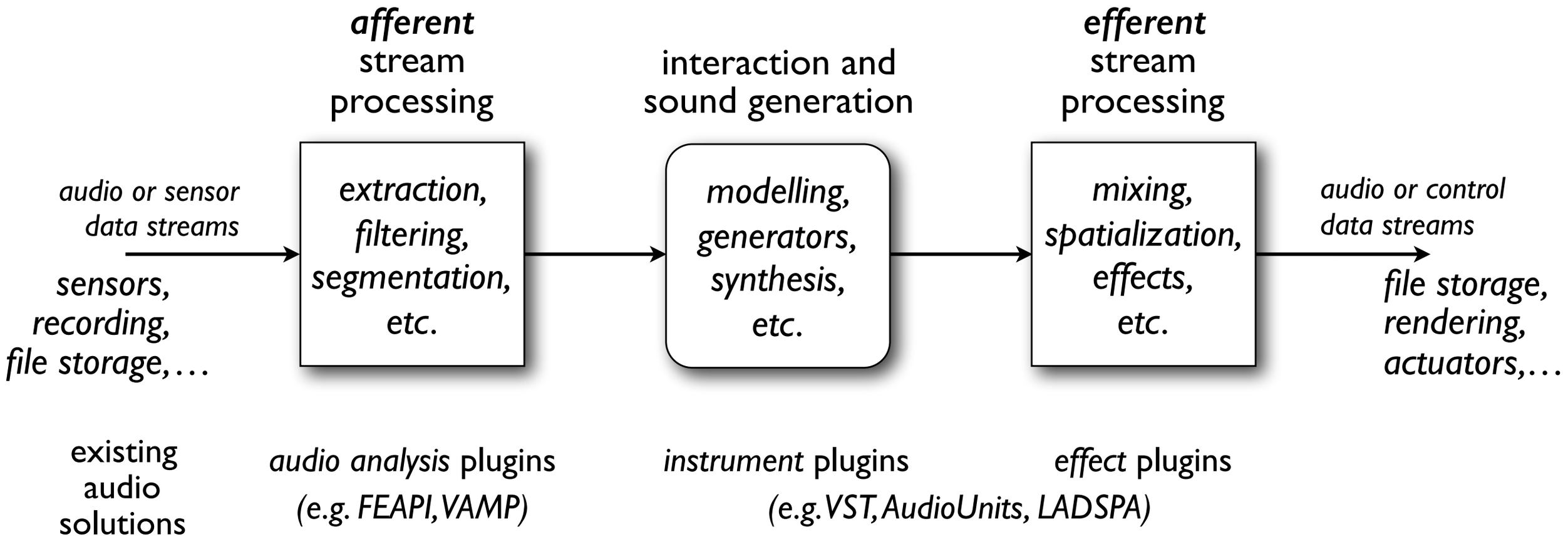
PiPo — Plugin Interface for Processing Objects

Context: Content-based Audio Processing

development of interactive audio and movement applications

collaboration with artists and other researchers

analyse and annotate audio and motion sensor streams



Motivations

integration of algorithms of different origins (i.e. developers) into a given application and into different contexts and applications, applied to data streams in **real-time and offline**

use/comparison of different algorithms of thesimilar functionalities in a given context (e.g. applying different filters, extractors or classifiers to the same input stream)

Top Level Requirements

Multi-Modality input data can be motion and other data not only audio

User-Composability of Modules e.g. chaining feature extractors, smoothing filters, segmentation, and temporal modeling in the host environment, *without having to write and compile code* → experimentation, rapid prototyping

Dynamic Loading of Plugins add processing modules as plugins to an existing host installation (e.g. as shared libraries)

Functional Requirements

Scheduling Processing in batch on files or buffers, or real-time input stream

Segmentation Several streams of segmentations in parallel and overlapping segments, or implicit segmentation, where segments are analysis frames, elementary waveforms, or whole sound files.

Temporal Modeling Integrate any number of temporal modeling algorithms, universal (modeling all descriptors, e.g. mean) or specific (modeling specific descriptors only).

Data Type Data can be numeric scalars, vectors, matrices, or strings

Implementation Requirements

Easy Integration and Efficiency Any platform and environment, real-time and resource-constrained systems (SBCs) → API must be in C or C++

Efficient Modularisation Efficient implementation, by sharing calculation results (FFT), avoiding copying by direct write to destination

External Data Streams and sources of segmentation (human tapping or existing analysis files) integrateable into the data flow

Reanalysis Recalculate a subset of descriptors or only segmentation and subsequent temporal modeling

API Overview

Simplicity

only 2 to 4 methods need to be implemented, single input/output, helper classes to declare module parameters

Data stream

succession of frames as module input and output

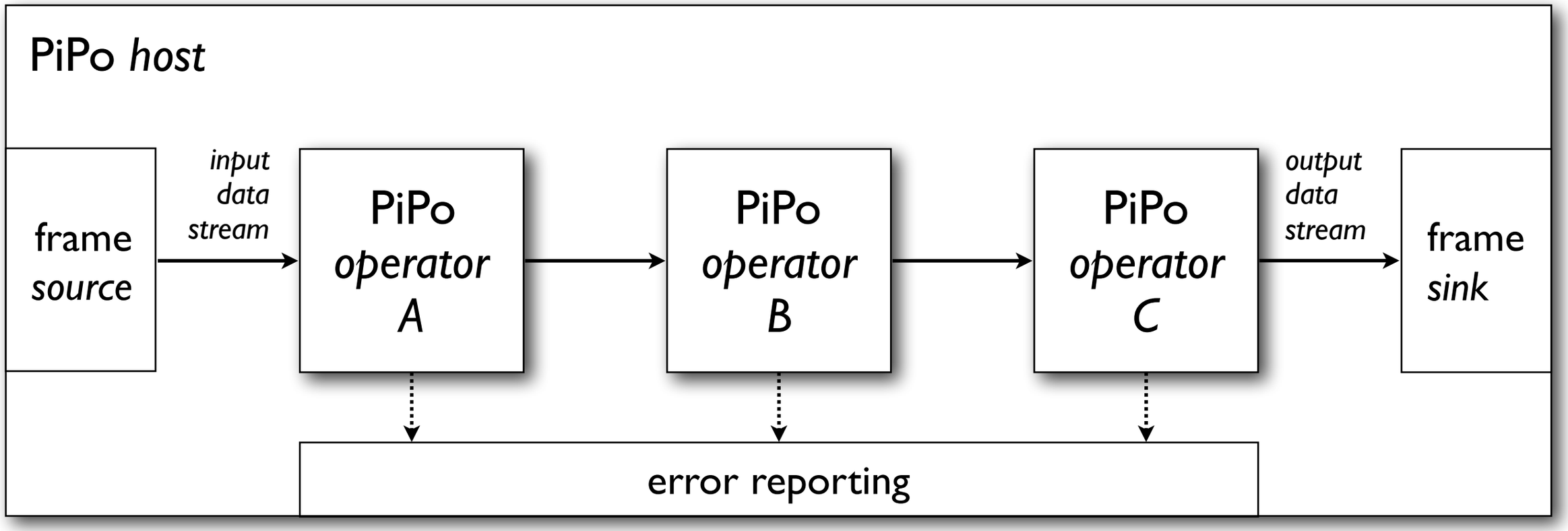
frames are float matrices with named columns, sampled or time tagged

Phases

setup	passing stream attributes through module graph, init modules, allocate memory: `streamAttributes()` method
processing	pass data through graph: `frames()` method
cleanup	`finalize()` pending data, `reset()` stream

PiPo Host

takes care of source/sink, building and instantiating module graph, error reporting



API Example http://github.com/Ircam-RnD/pipo-sdk

```
class PiPoGain : public PiPo
{
private:
  std::vector<PiPoValue> buffer;

public:
  PiPoScalarAttr<double> factor;

  PiPoGain (Parent *parent, PiPo *receiver = NULL)
: PiPo(parent, receiver),
  factor(this, "factor", "Gain Factor", false, 1.0) { }

~PiPoGain (void) { }

int streamAttributes (bool hasTimeTags, double rate,
  double offset, unsigned int width, unsigned int height,
  const char **labels, bool hasVarSize,
  double domain, unsigned int maxFrames)
{ // can not work in place, create output buffer
  buffer.resize(width * height * maxFrames);

  return propagateStreamAttributes(hasTimeTags, rate,
           offset, width, height, labels,
           hasVarSize, domain, maxFrames);
}
```

API Example http://github.com/Ircam-RnD/pipo-sdk

```
int frames (double time, PiPoValue *values,
            unsigned int size, unsigned int num)
{ // get gain factor here, it could change while running
  double f = factor.get();
  PiPoValue *ptr = &buffer[0];

  for (unsigned int i = 0; i < num; i++)
  {
    for (unsigned int j = 0; j < size; j++)
      ptr[j] = values[j] * f;

    ptr    += size;
    values += size;
  }

  return propagateFrames(time, &buffer[0], size, num);
 }
};
```

Module Library

Stream Processing: slice (windowing), scale, sum, select (get columns), const

Filtering: biquad (biquad filter), mvavrg (moving average filter), median (median filter), delta (derivative), finitedif, bayesfilter

Segmentation: onseg (segments starting at signal onset), chop (segments of regular intervals), gate (segments excluding weak sections), sylseg (syllable segmentation)

Temporal Modeling: mean, std, meanstd, min, max

Analysis: descr (basic audio descriptors), yin (pitch extractor), moments (centroid, spread, skewness, kurtosis), lpc (linear predictive coding), lpcformants (formant extraction), psy (pitch synchronous markers), ircamdescriptor

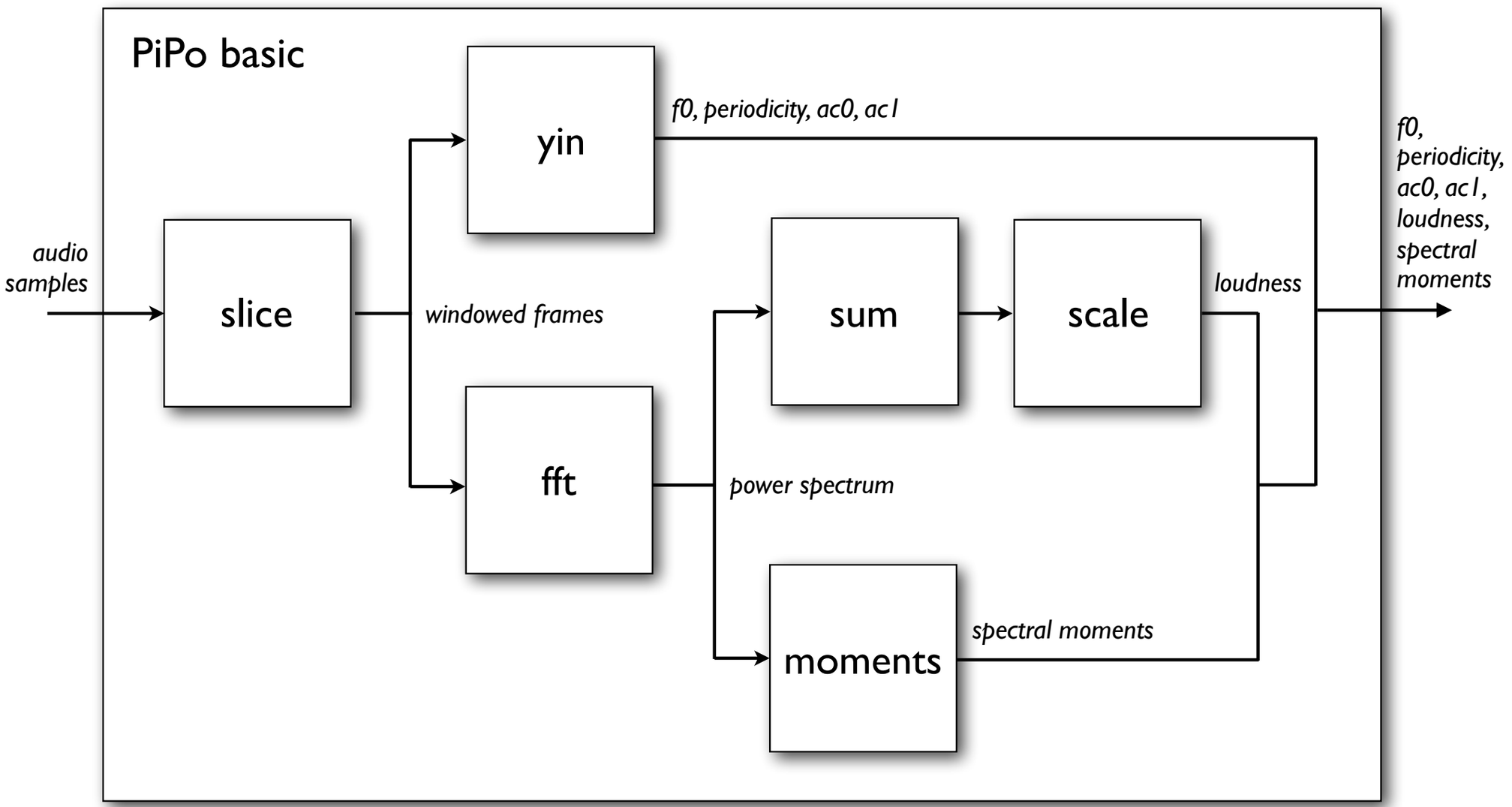
Frequency Domain Processing: fft (FFT from pre-windowed frames), dct (discrete cosine transform), bands (mel bands and similar from power or amplitude spectrum), mel (mel bands from audio stream), mfcc (MFCC from audio stream), wavelet (wavelet transform from audio stream)

PiPo Graphs

Modules in **sequence** or **parallel**, merge of output concatenates columns

User level graph syntax

```
slice<yin,fft<sum:scale,moments>>
```



Bindings

Max/MSP

via MuBu package **http://ismm.ircam.fr**:
real-time with pipo~ or pipo, offline with mubu.process

Juce, OpenFrameworks, OpenMusic, Unity3D

via IAE (Interactive Audio Engine) library

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme H2020-ICT-2014-1 under grant agreement No 644862