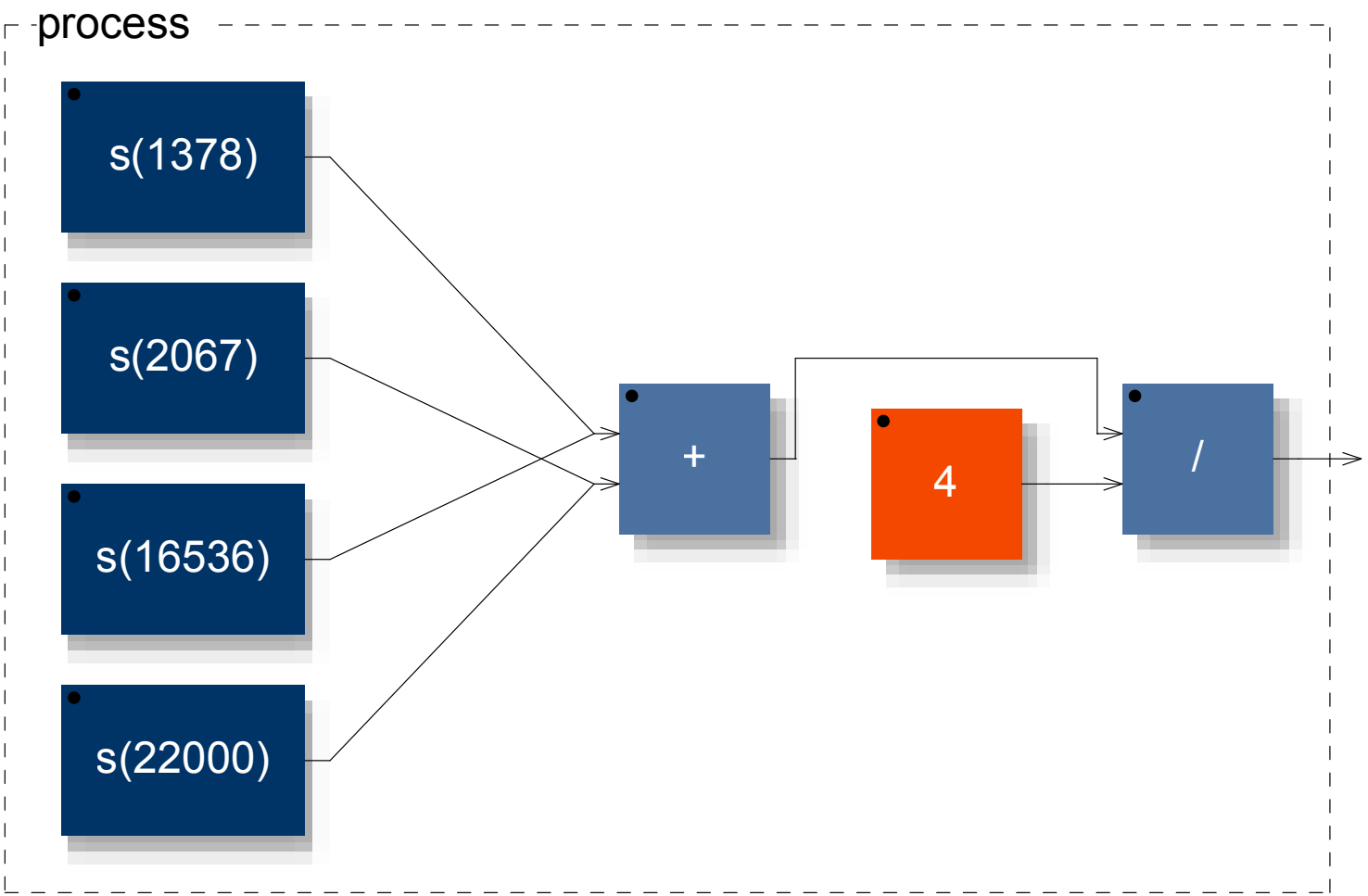
Faustine

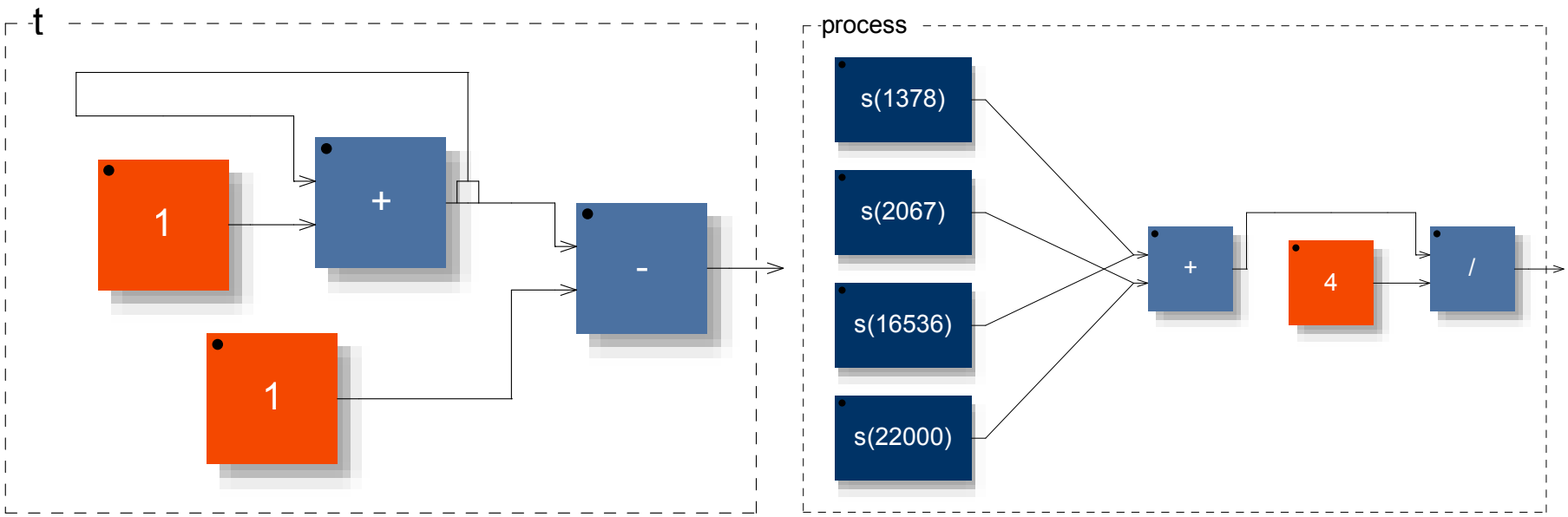
une plate-forme Faust vectorielle pour le traitement de signal multimédia

Karim Barkati, Haisheng Wang, Pierre Jouvelot
CRI MINES ParisTech
5 décembre 2013

Somme de 4 sinus normalisée



```
import("math.lib");  samplerate = 44100;
process = s(1378) , s(2067) , s(16536) , s(22000) :> + : /(4) ;
s(f) = 2.0*PI*f*t/samplerate : sin;
t = ( +(1) ~ _) - 1;
```




```
output0[i] = (FAUSTFLOAT)(0.25f * (sinf((0.1963173136719884f * iTemp0)) +
((sinf((3.13446886072451f * iTemp0)) + sinf((0.294497597050798 3f * iTemp0))) +
sinf((2.355980776406 3865f * iTemp0)))));
```

```
import("math.lib");  samplerate = 44100;
process = s(1378) , s(2067) , s(16536) , s(22000) :> + : /(4) ;
s(f) = 2.0*PI*f*t/samplerate : sin;
t = ( +(1) ~ _) - 1;
```

```
virtual void compute (int count, FAUSTFLOAT** input, FAUSTFLOAT** output)
{
   FAUSTFLOAT* output0 = output[0];
   for (int i=0; i<count; i++) {
   iRec0[0] = (1 + iRec0[1]);
   int iTemp0 = (iRec0[0] - 1);
   output0[i] = (FAUSTFLOAT)(0.25f * (sinf((0.1963173136719884f * iTemp0))
       + ((sinf((3.134468860724 51f * iTemp0)) + sinf((0.294497597050 7983f
       * iTemp0))) + sinf((2.355980776406 3865f * iTemp0)))));
   // post processing
   iRec0[1] = iRec0[0];
   }
}
```

- FAUST : « Functional Audio **Stream** », DSL
- *« The quick path from ideas to efficient DSP »*
- Algèbre de blocs-diagrammes
- Traitement du signal audio temps réel
- Orienté performance
- Compilateur écrit en C++
- Génération de code:
  C, C++, LLVM, Java, javascript ;
  MaxMSP, Pure Data, SuperCollider ;
  LaTeX, SVG, Qt, etc.

Syntax	Type	Description
n	$\mathbb{S}^0 \rightarrow \mathbb{S}^1$	integer number: $y(t) = n$
$n.m$	$\mathbb{S}^0 \rightarrow \mathbb{S}^1$	floating point number: $y(t) = n.m$
_	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	identity function: $y(t) = x(t)$
!	$\mathbb{S}^1 \rightarrow \mathbb{S}^0$	cut function: $\forall x \in \mathbb{S}, (x) \rightarrow ()$
int	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an int signal: $y(t) = (int)x(t)$
float	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an float signal: $y(t) = (float)x(t)$
+	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	addition: $y(t) = x_1(t) + x_2(t)$
-	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	subtraction: $y(t) = x_1(t) - x_2(t)$
*	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	multiplication: $y(t) = x_1(t) * x_2(t)$
^	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	power: $y(t) = x_1(t)^{x_2(t)}$
/	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	division: $y(t) = x_1(t)/x_2(t)$
%	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	modulo: $y(t) = x_1(t)\%x_2(t)$
&	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical AND: $y(t) = x_1(t)\&x_2(t)$
|	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical OR: $y(t) = x_1(t)\|x_2(t)$
xor	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical XOR: $y(t) = x_1(t) \wedge x_2(t)$
«	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift left: $y(t) = x_1(t) << x_2(t)$
»	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift right: $y(t) = x_1(t) >> x_2(t)$
<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less than: $y(t) = x_1(t) < x_2(t)$
<=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less or equal: $y(t) = x_1(t) <= x_2(t)$
>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater than: $y(t) = x_1(t) > x_2(t)$
>=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater or equal: $y(t) = x_1(t) >= x_2(t)$
==	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	equal: $y(t) = x_1(t) == x_2(t)$
!=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	different: $y(t) = x_1(t) != x_2(t)$

a : b

a , b

a ~ b

a <: b

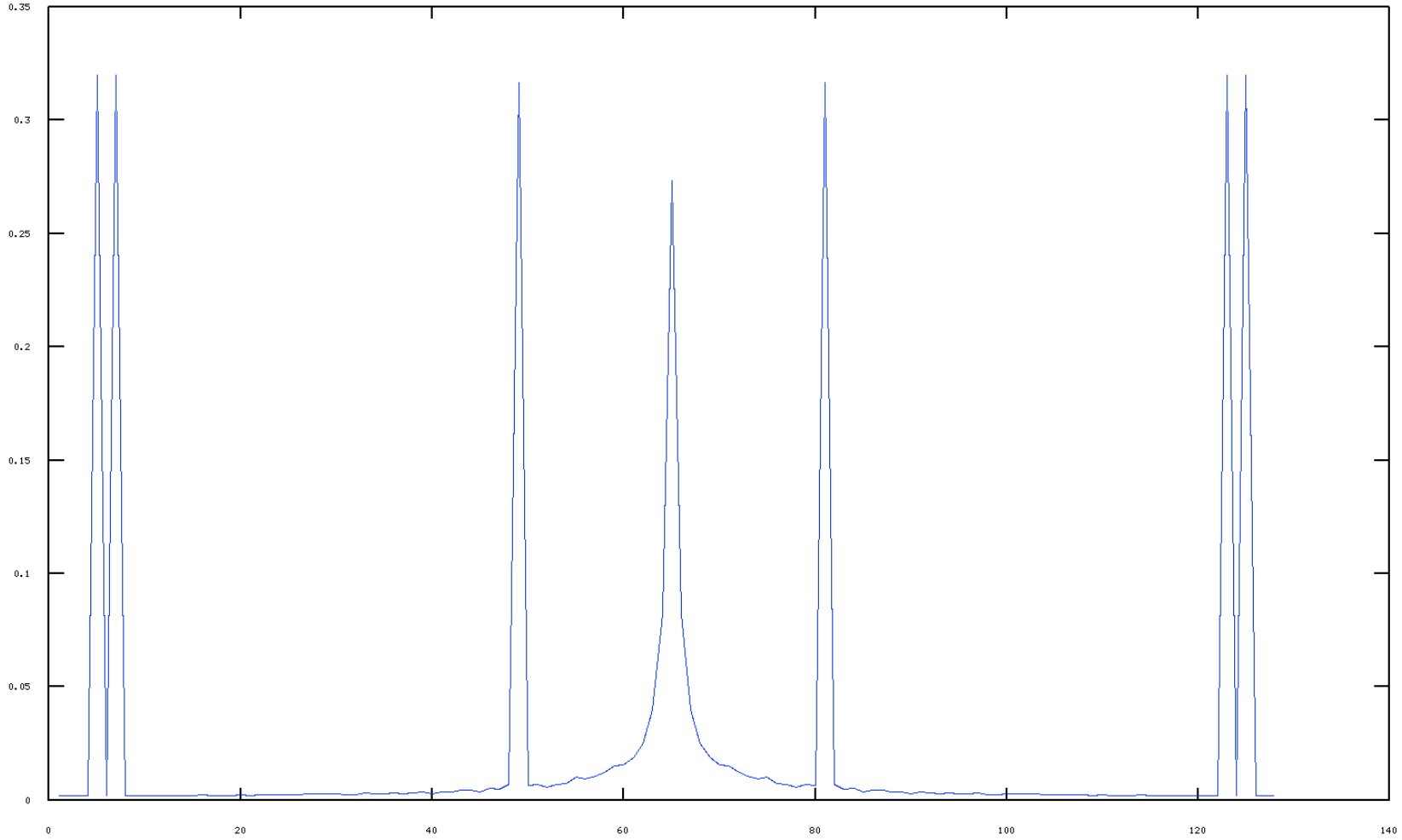
a :> b

Plan

1. Faust
2. FFT / vecteurs
3. LicensePlate / opérations morphologiques
4. Faustine
5. Discussion : design et performances
6. Perspectives

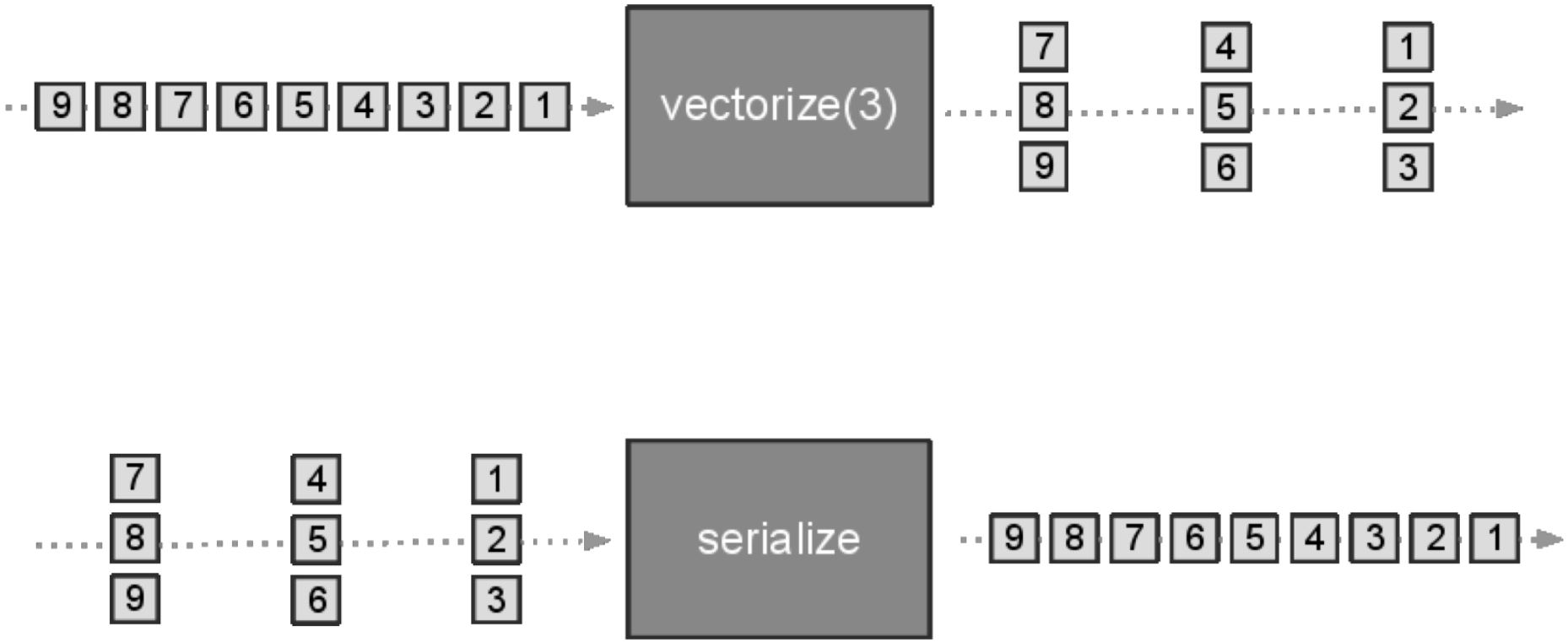
Pas de FFT?!

faustine -f fft.dsp -i sinsum.wav :-)

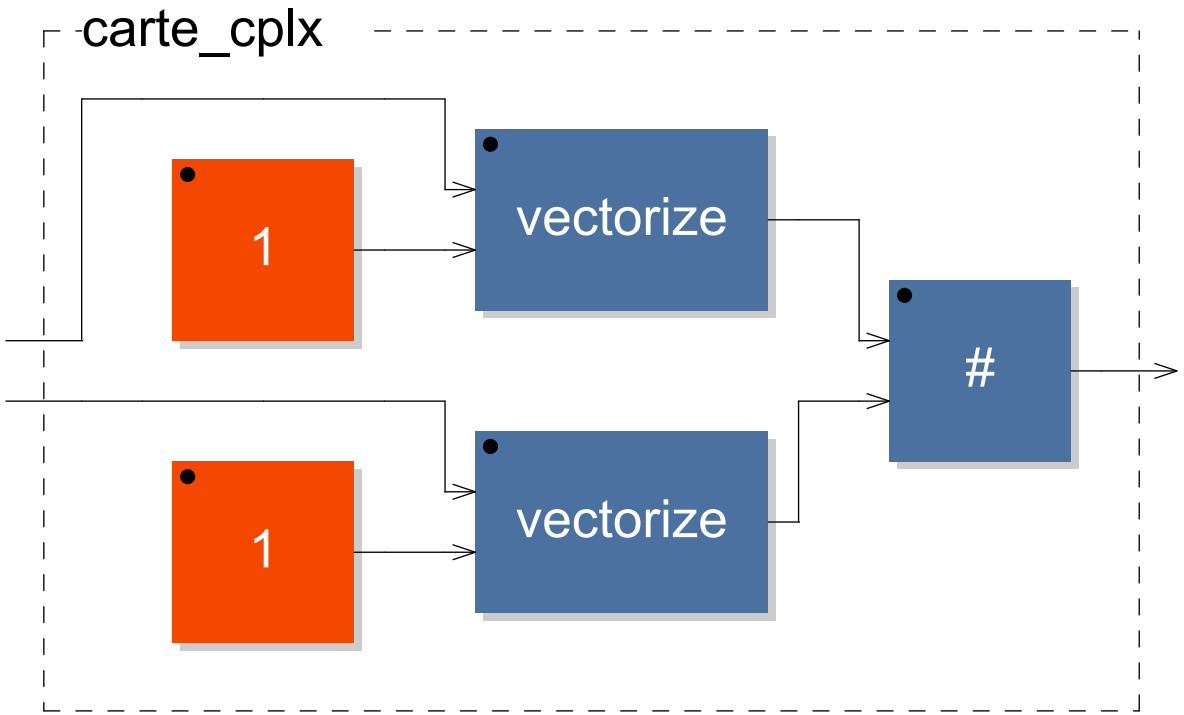


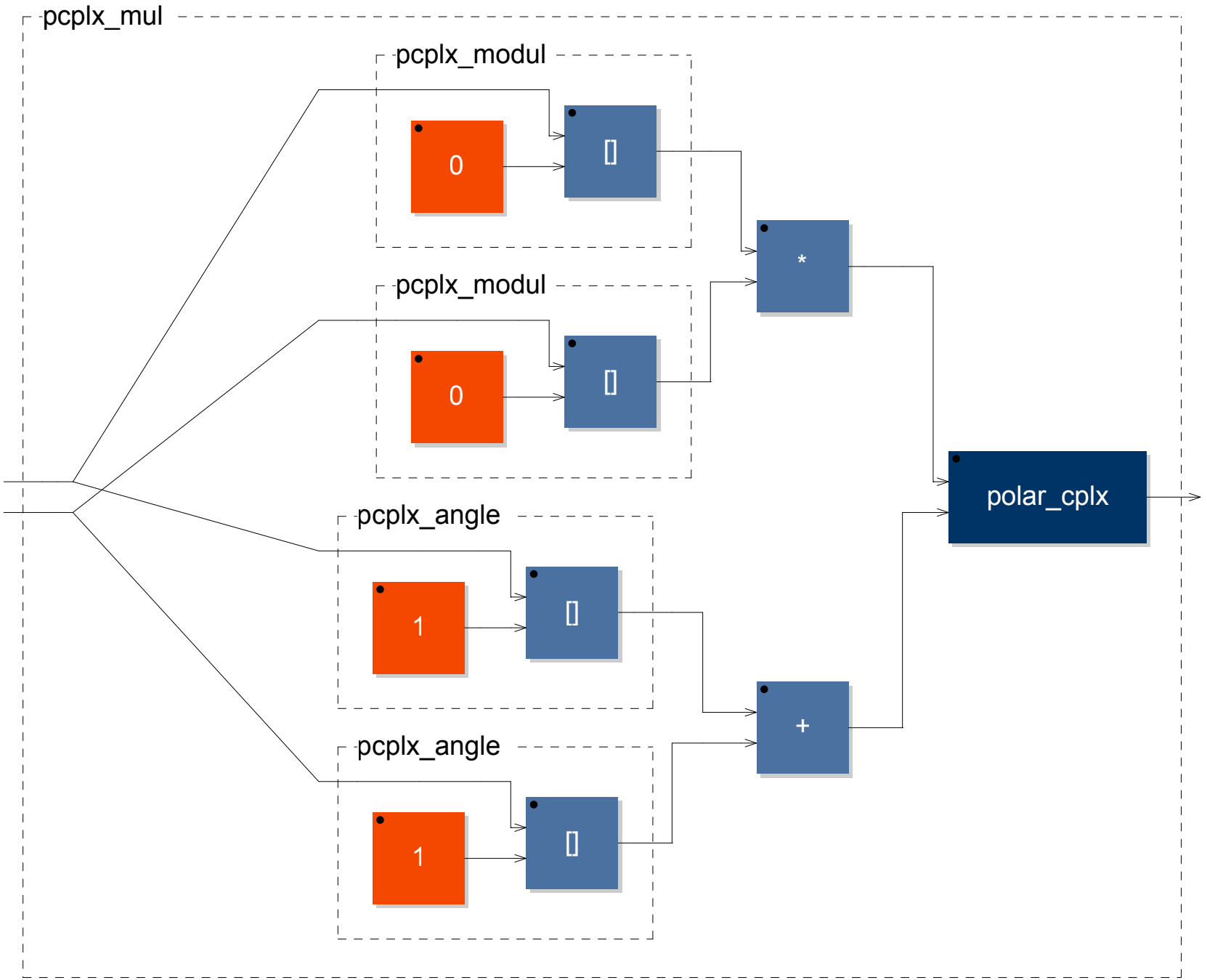
Pierre Jouvelot and Yann Orlarey. Dependent vector types for data structuring in multirate Faust. *Comput. Lang. Syst. Struct.*, 37:113–131, July 2011.

- vectorize : $(\tau^f, \mathrm{int}[n,n]^{f'}) \to (\mathrm{vector}_n(\tau)^{f/n})$
- serialize : $(\mathrm{vector}_n(\tau)^f) \to (\tau^{f \times n})$
- [ ] : $(\mathrm{vector}_n(\tau)^f, \mathrm{int}[0,n-1]^f) \to (\tau^f)$
- # : $(\mathrm{vector}_m(\tau)^f, \mathrm{vector}_n(\tau)^f) \to (\mathrm{vector}_{m+n}(\tau)^f)$


9 8 7 6 5 4 3 2 1
vectorize(3)
7 8 9
4 5 6
1 2 3
7 8 9
4 5 6
1 2 3
serialize
9 8 7 6 5 4 3 2 1

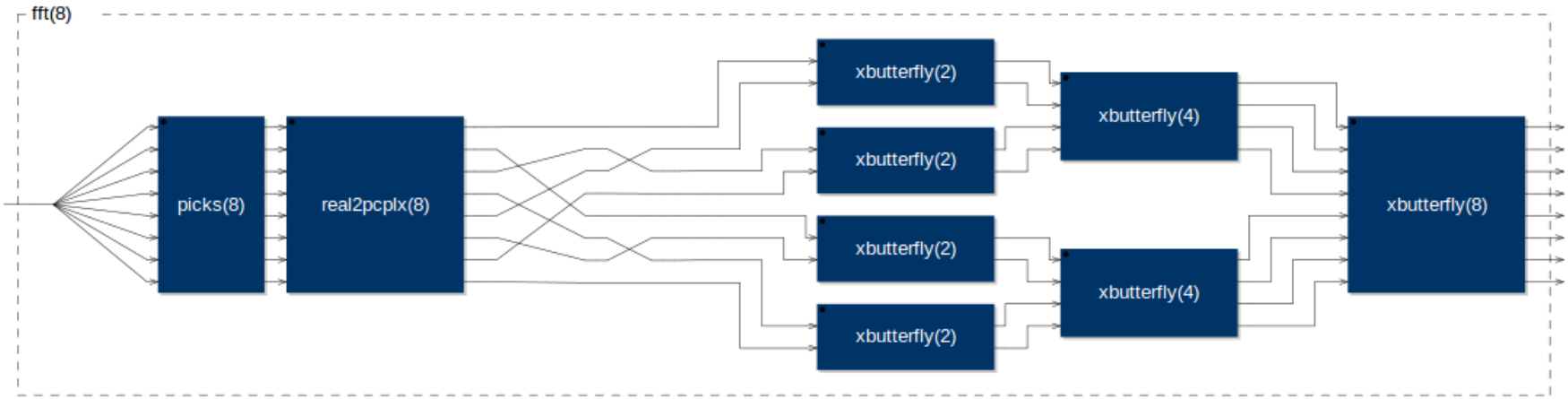
complex.lib




pcplx_mul
pcplx_modul
0
[]
pcplx_modul
0
[]
*
pcplx_angle
1
[]
pcplx_angle
1
[]
+
polar_cplx

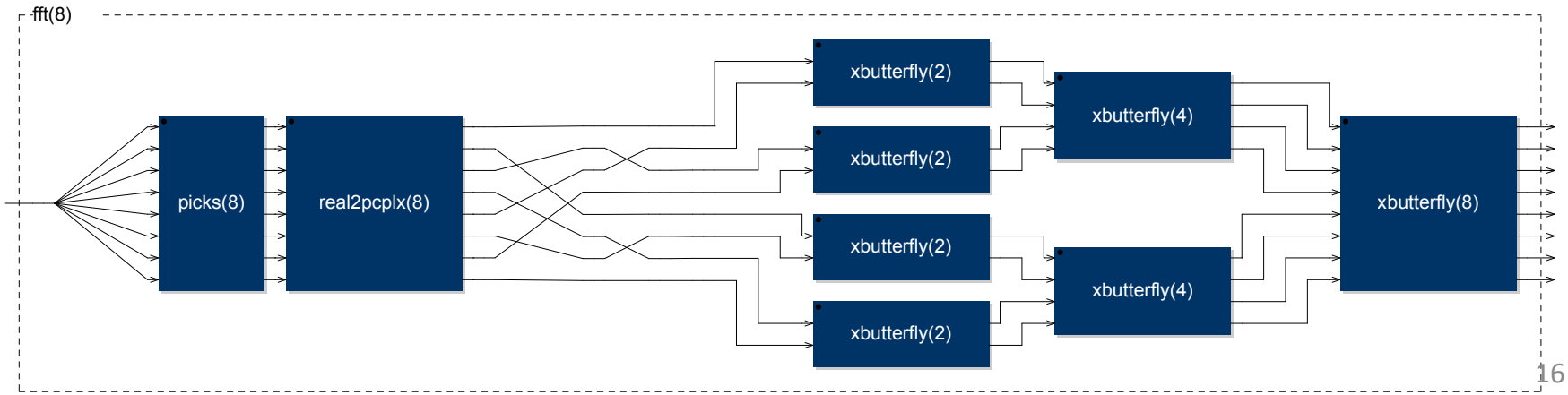
```
subsampling(n) = (_,n) : vectorize : [0];

t = _ ~ (1,_ : +);
process = t : subsampling(2);
```

Butterfly: $O(N^2)$ -> $O(N \log(N))$



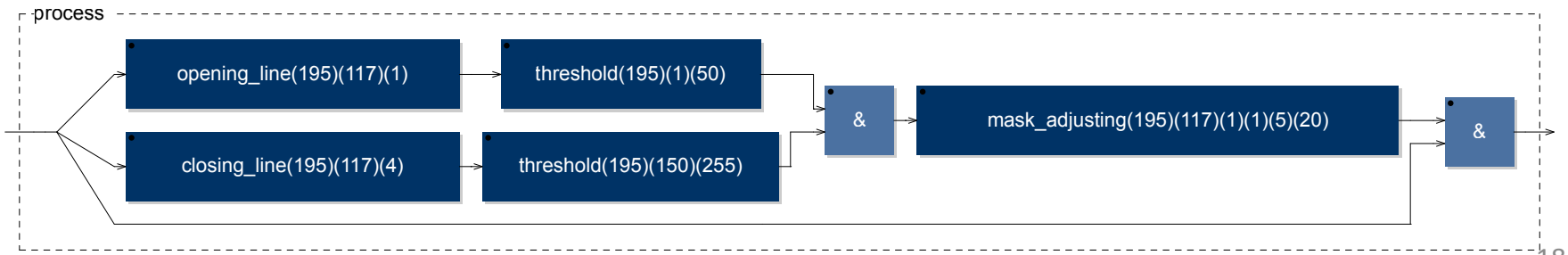
$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk}}_{\text{DFT of even-indexed part of } x_m} + e^{-\frac{2\pi i}{N} k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk}}_{\text{DFT of odd-indexed part of } x_m}$$

```
import ("fft.lib"); import ("complex.lib");
// Complex add and sub of butterfly-interleaved signals
butterfly(n) = (bus(n/2) , par(k, n/2, twiddle_mult(k, n))) <:
  interleave(n/2, 2) , interleave(n/2, 2) :
  par(i, n/2, pcplx_add) , par(i, n/2, pcplx_sub);
butterflies(2) = xbutterfly(2);
butterflies(n) = (butterflies(n/2) , butterflies(n/2)) : butterfly(n);
picks(n) = par(i, n, [ i ]);
fft(n) = _ <: picks(n) : real2pcplx(n) : shuffle(n) : butterflies(n);
process = vectorize(8) : fft(8) : pcplx_modules(8) : nconcat(8) : serialize;
```

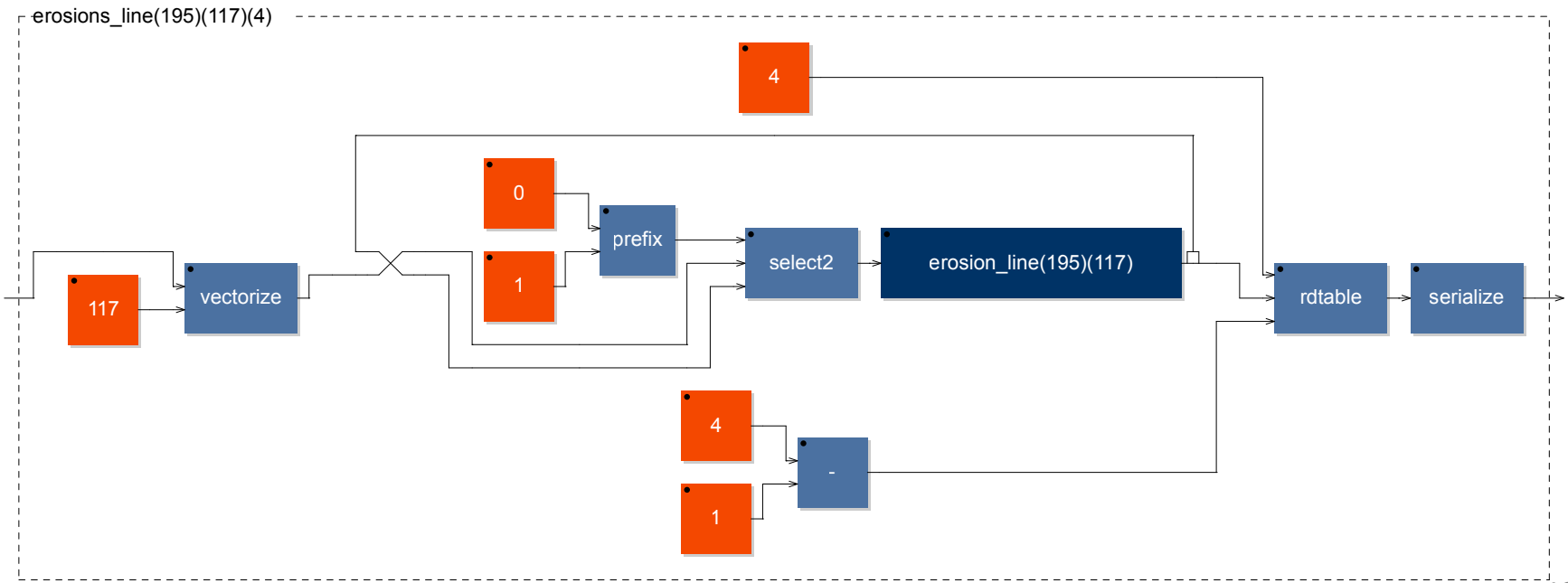


LicencePlate chez Faustine

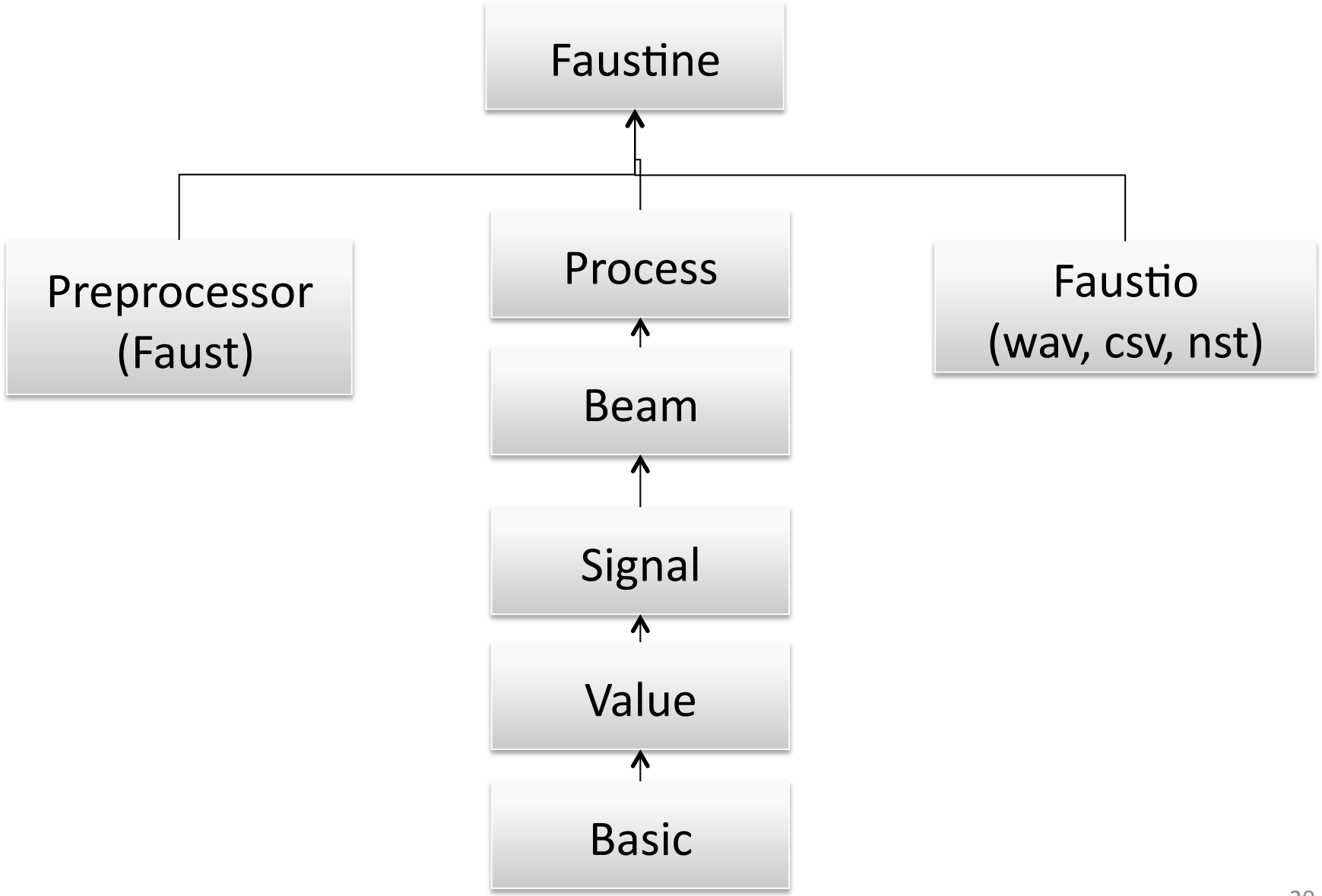
```
import("morpho.lib");
process = licenceplate(195, 117, 1, 4, 1, 50, 150, 255, 1, 1, 5, 20);

licenseplate(x, y, o1, c2, t3, t4, t5, t6, o7, o8, d9, d10) =
  _ <: (opening_line(x, y, o1) , closing_line(x, y, c2) :
  threshold(x, t3, t4) , threshold(x, t5, t6) : and :
  mask_adjusting(x, y, o7, o8, d9, d10)) , _ : and;

mask_adjusting(x, y, o7, o8, d9, d10) =
  opening_column(x, y, o7) : opening_line(x, y, o8) :
  dilations_square(x, y, d9) : dilations_line(x, y, d10);
```



```
erosions_line(x, y, iter) = _ : (iter, ((cross : ((0, 1 : prefix), _, _ : select2) :
    erosion_line(x, y))~_), (iter - 1)) : rdtable;

erosion_line(x, y) = serialize : eroding(x) : vectorize(y); // matrix io

eroding(n) = strel_shift_erosion, _, strel_shift_erosion : #, _ : # :
    spray_by_three(n) : tri_mins(n) : nconcat(n); // vector io
```



Faustine Architecture



Discussion

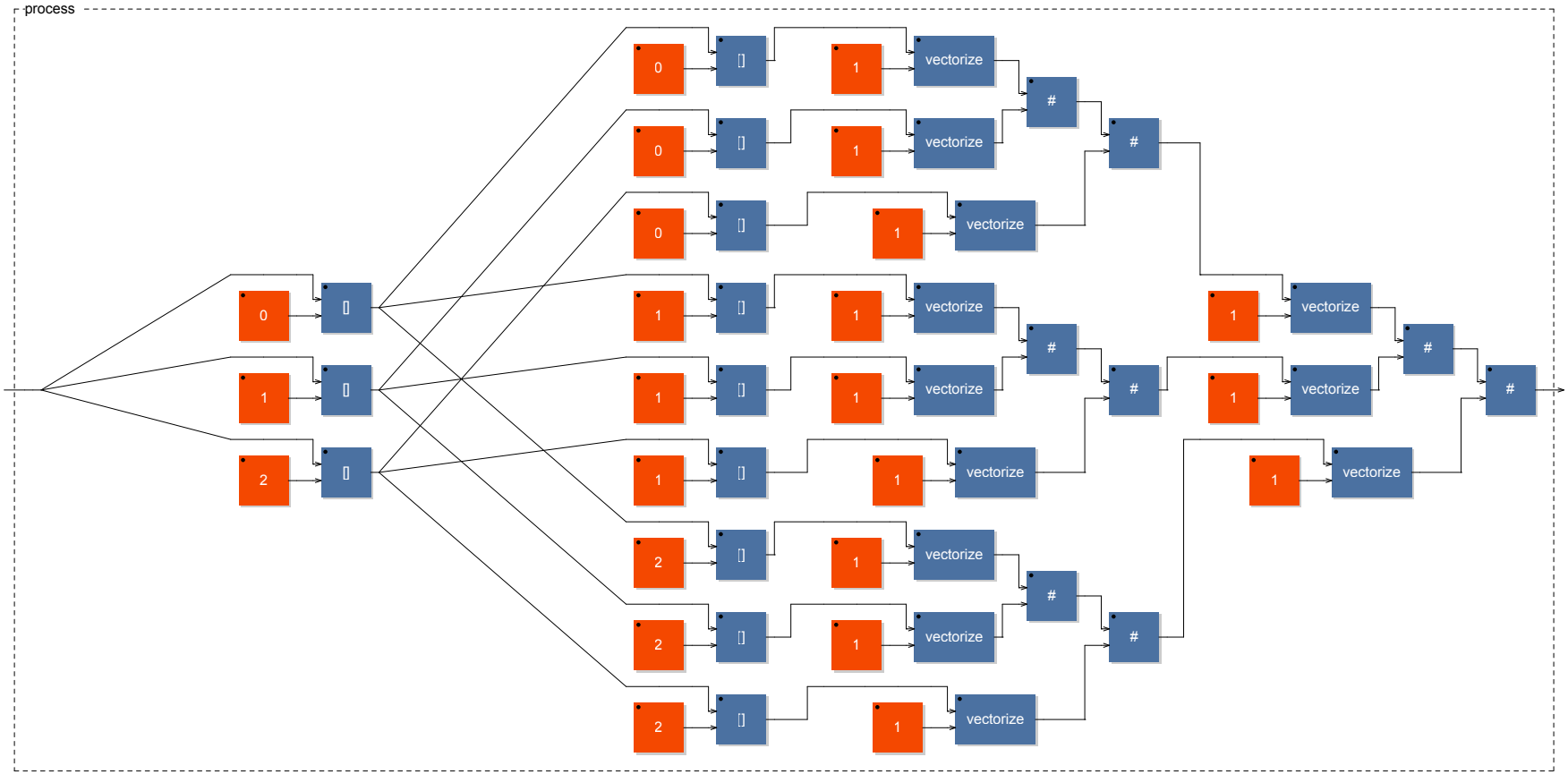
- Fonctions GUI
- Macros dédiées aux vecteurs
- Elargissement du domaine d'application

- Performance de conception
- Performance d'exécution
- Macro expansion

```
_ : access x as {
        [0] => x;
        [1] => -x;
    }
```

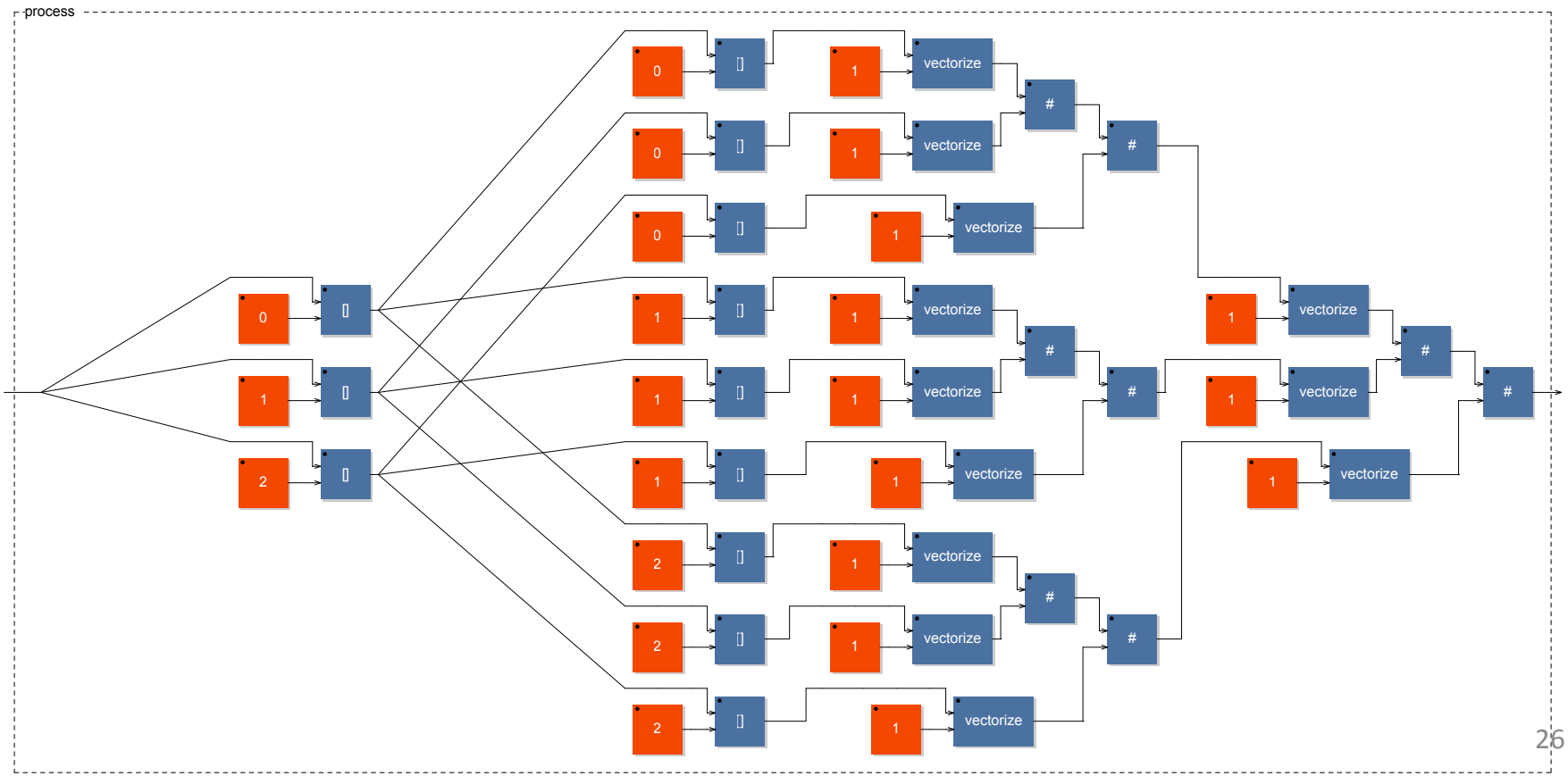
- Ubuntu 12.04 LTS desktop avec deux Intel Core 2 Duo CPU E8600 64-bit 3.3GHz chacun, et 3.7 Go de RAM.
- FFT: 22.4 secondes pour une fenêtre de 128 échantillons en 64 bits.
- LicensePlate: 812 secondes, pour une petite image de 195 x 117 pixels.
- LicensePlate: > 15 heures… pour une image de 640 x 383 pixels… (macro-expansion)

Transposition 3x3



```
process = matrix_transpose(3,3);

matrix_transpose(n, m) =
  _ <: par(i, n, [ i ]) <: par(j, m, (par(i, n, [ j ]) :
    concats(n))) : concats(m);

concats = case {
    (1) => vectorize(1);
    (n) => concats(n-1) # vectorize(1);
};
```

```
matrix_transpose(n, m) = _ <: par(i, n, [ i ]) <:
    par(j, m, (par(i, n, [ j ]) : concats(n))) : concats(m);
concats = case {
    (1) => vectorize(1);
    (n) => concats(n-1) # vectorize(1); };
```



Perspectives

- Vérification de type en statique
- Exploration de l'espace de design de DSL
- Tests de compromis spécificité-vs-généralité

- Vecteurs imbriqués
- FLOPS 2014, « Faustine: a Vector Faust Test Bed for Multimedia Signal Processing », KB, HW, PJ
- Interpréteurs pour la conception de DSLs

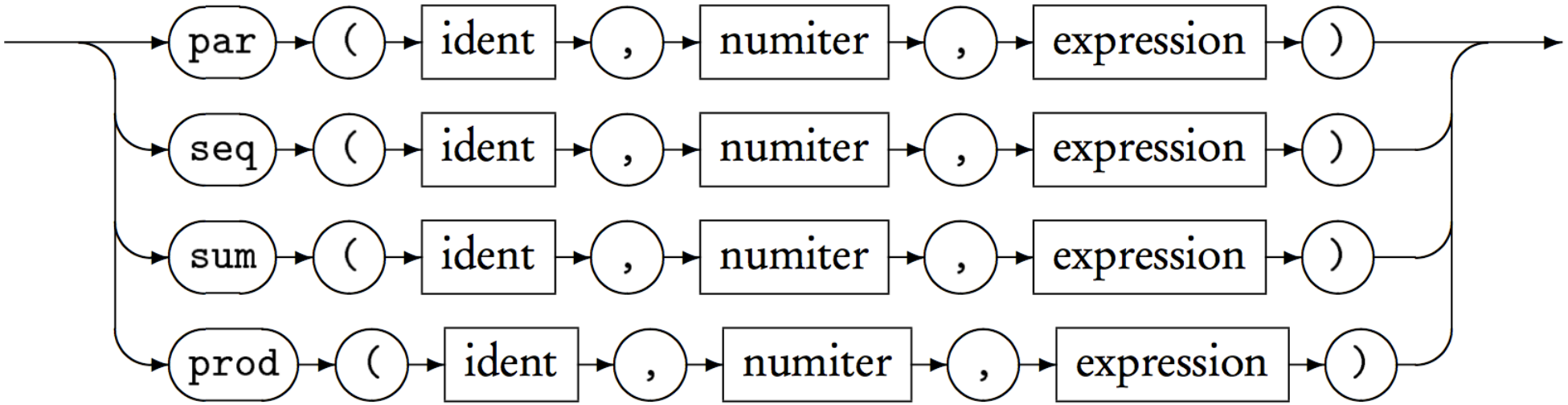
- Projet ANR FEEVER 2014-2017 (Mines, Grame, St-Etienne, INRIA)

Faustine

une plate-forme Faust vectorielle pour le traitement de signal multimédia

Karim Barkati, Haisheng Wang, Pierre Jouvelot
CRI MINES ParisTech
5 décembre 2013

Syntax	Pri.	Assoc.	Description
expression ~ *expression*	4	left	recursive composition
expression , *expression*	3	right	parallel composition
expression : *expression*	2	right	sequential composition
expression <: *expression*	1	right	split composition
expression :> *expression*	1	right	merge composition



Definition 15 (Beam Bound $\mu(m, z)$).
The bound $\mu(m, z)$, in $\mathbb{N} \cup \{\omega\}$, of a beam m of beam type z is defined by

$$\mu(m, z) = \min_{i \in [1, |z|]} \lfloor \underline{m[i]} / \sharp(z[i]) \rfloor \ ,$$

where, for all $n \in \mathbb{N}$, we have $n/0 = \omega$.

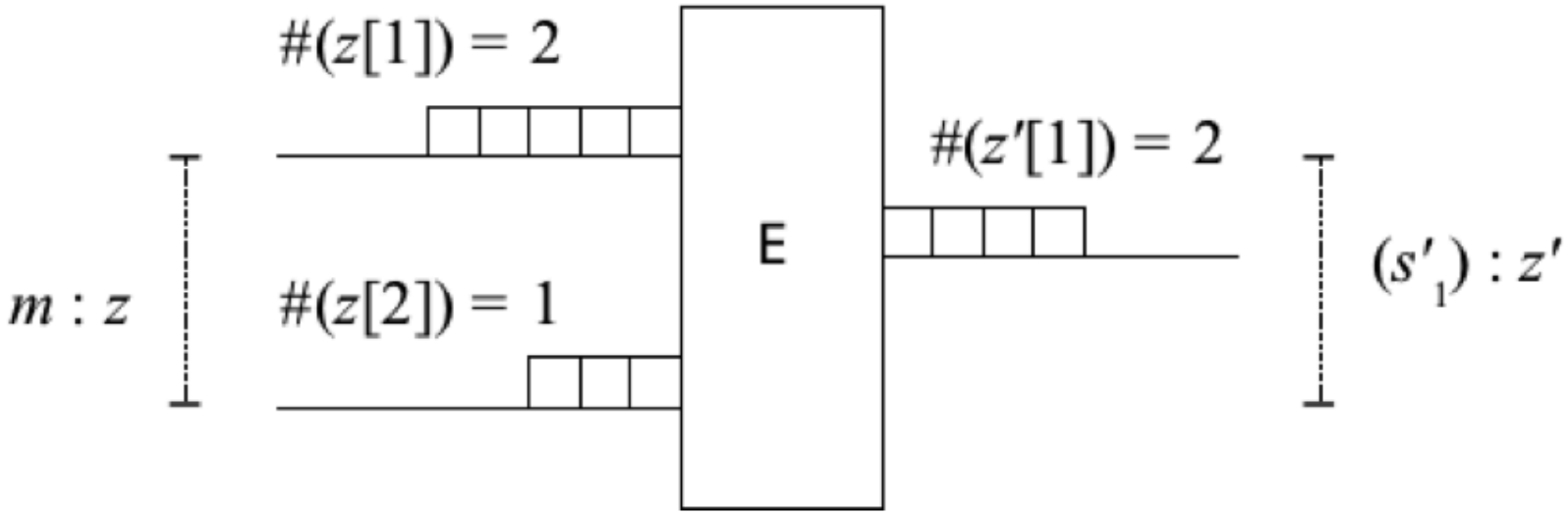


Figure 5: A beam bound example, with $m = (s_1, s_2)$ and $\mu(m, z) = 2$.

Theorem 4 (Rate Correctness).
For all $\mathtt{E}, T, z, z', c, r, m$ *and* m'*, if*

$$r : T \ ,$$
$$m : z \ ,$$
$$T \vdash \mathtt{E} : z \to z' \ ,$$

then $|z'| = 0 \ \vee \ \mu(m', z') \leq \mu(m, z) + @_T(\mathtt{E})$*, where* $m' = p(m) : z'$ *and* $p = E[\![\mathtt{E}]\!]r$.

OVERVIEW
========

Faustine is an interpreter for multi-rate FAUST programs, written in OCaml, at CRI of MINES ParisTech, and covered by the GNU Public License V3 (see LICENSE.txt).

FAUST (Functional Audio Stream) is a functional programming language specifically designed for real-time signal processing and synthesis. It targets high-performance signal processing applications and audio plug-ins for a variety of platforms and standards. <http://faust.grame.fr>

Faustine provides support for some vector features presented in the paper "Semantics for Multirate Faust", written by Pierre Jouvelot and Yann Orlarey, 2009:

- vectorize
- serialize
- [ ] (pick an element from a vector)
- # (concatenate two vectors)

```
CONTENTS
--------

  benchmarks/          benchmark result files
  Changes.txt          what's new with each release
  configure            compilation configuration script
  examples/            vector examples (fft, image processing...)
  INSTALL.txt          Faustine installation instructions
  interpreter/         Faustine's interpreter source code
  lib/                 library files in Faustine (fft.lib,
   morpho.lib...)
  LICENSE.txt          license and copyright notice
  Makefile             main Makefile to compile and install
  README.txt           this file
```

```
Installing Faustine on a Unix machine
=====================================


PREREQUISITES
-------------

* OCaml is needed (tested versions: 3.12.1, 4.00.1).
* Faust is needed (tested version: 0.9.24).
* The GNU C compiler gcc is recommended.
* Standard development utilities are required, such as
   `make' (install XCode command line tools on Mac OS).


INSTALLATION INSTRUCTIONS
-------------------------

0- Faustine's git repository can be cloned calling:

        git clone https://scm.cri.ensmp.fr/git/Faustine.git

1- Configure the system. From within the Faustine directory,
```

Multimedia : wave file (.wav), csv file (.csv), nested vector (.nst)

.nst example :

4 : scalar

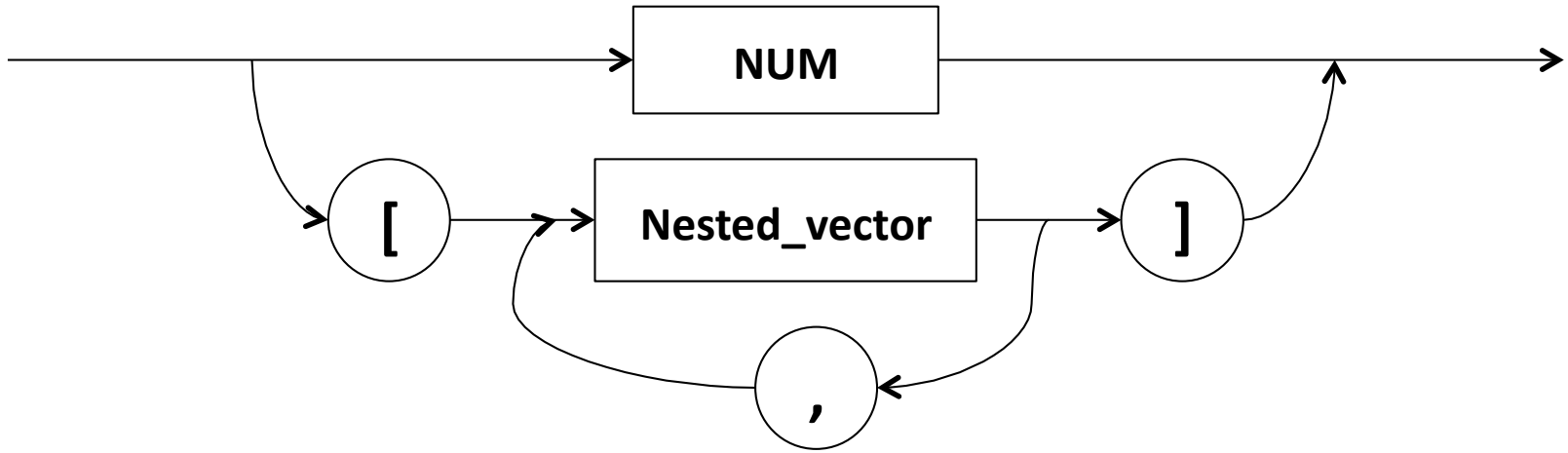
[4] : vector of one element

[4, 5] : vector of two elements 4 and 5

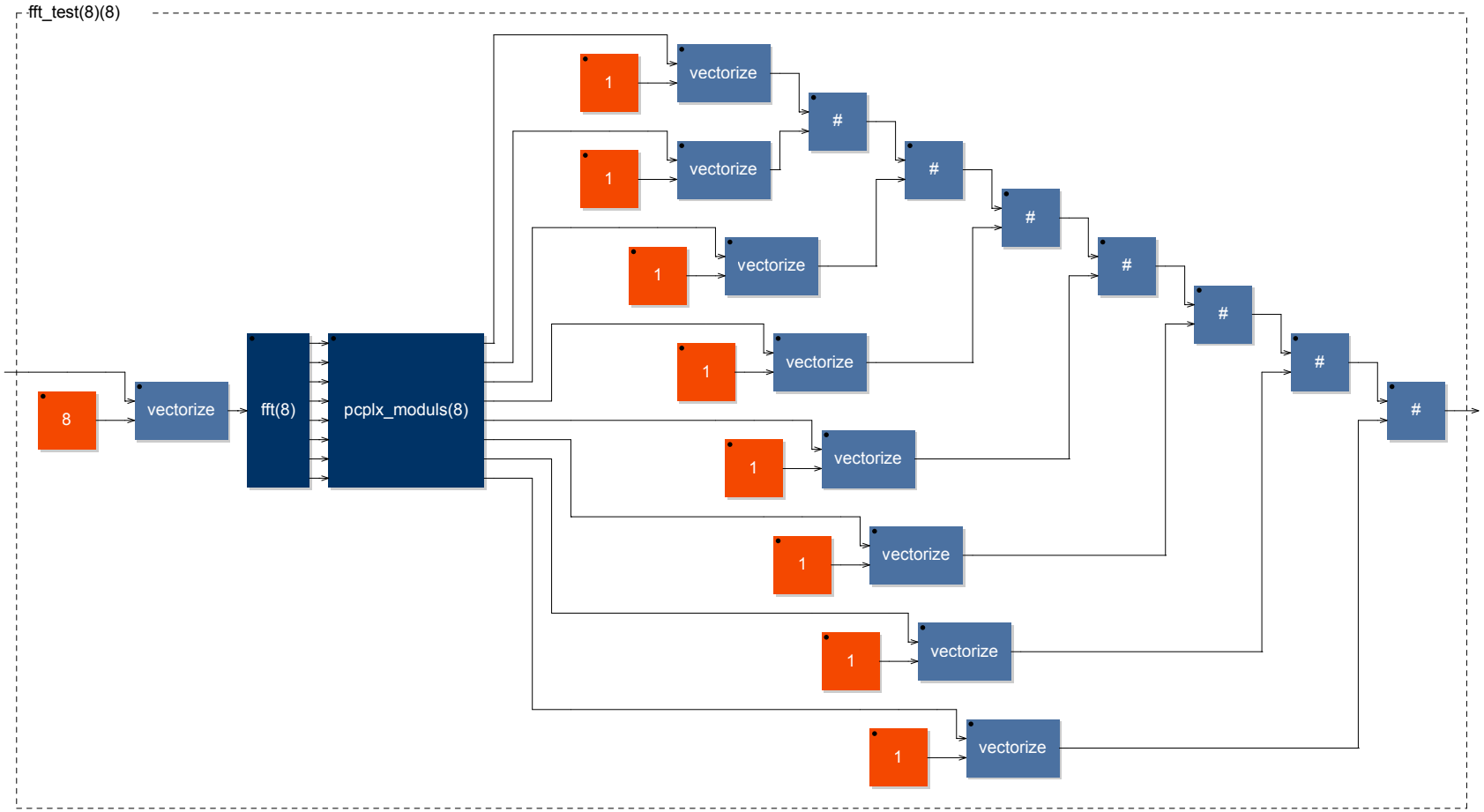
[[1, 2, 3], [4, 5, 6]] : vector of vectors

[[[1], [2]], [[3], [4]]] : vector of vectors of vectors

Nested_vector



nconcats



FAUST programs

describes a signal processor.

The role of a signal processor is to transform a group of (possibly empty) input signals in order to produce a group of (possibly empty) output signals.

Most audio equipments can be modeled as signal processors.