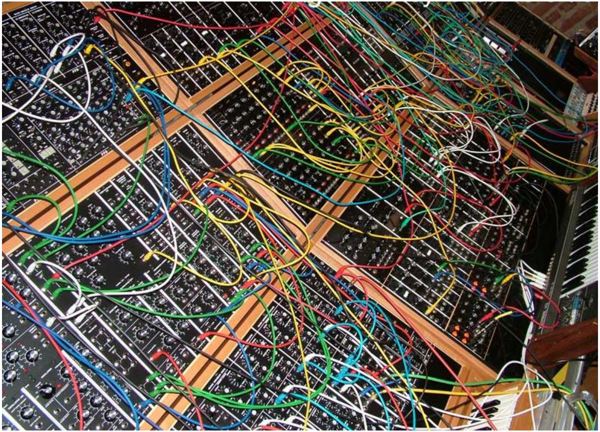
Functional
Audio
Stream

Electronic instruments and audio plug-ins design using Faust

Yann Orlarey, GRAME

BIENNALE COLLEGE – CIMM 2019

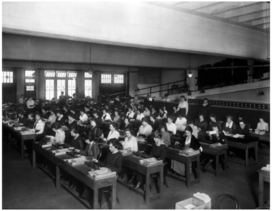
What is Faust?

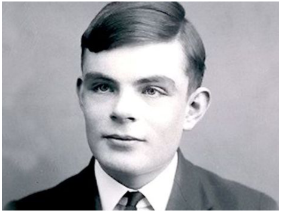


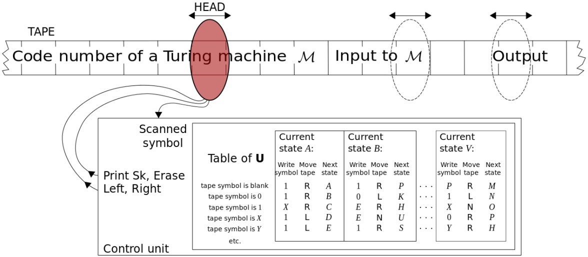
A programming language (DSL) to build electronic music instruments, audio plugins, signal processing applications, etc.

Computers and programming languages

What is a *computer* ?







What is a *computer* ?

- A computer is (a finite approximation of) a Universal Turing Machine.
- If we don't take into account speed and memory size, past, present and future computers (including quantum computers) are all equivalent!

What is *computer programming* ?

- The purpose of *Computer Programming* is to teach a universal machine how to behave.
- A *Programming Language* is a vocabulary and set of grammatical rules used to describe such behaviors.
- The *oldest programming language* is FORTRAN (FORmula TRANslation), designed 1957 by John Backus. FORTRAN is still used today for scientific computing and HPC.

Music Programming Languages

Some Music Languages

- 4CED
- Adagio
- AML
- AMPLE
- Antescofo
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- Chuck
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- Csound
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- Faust
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- Kronos
- Kyma
- LOCO
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MUSIC 6
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musictex
- MUSIGOL
- MusicXML
- Musixtex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- Puredata
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SOUL
- SSSP
- ST
- Supercollider
- Symbolic Composer
- Tidal

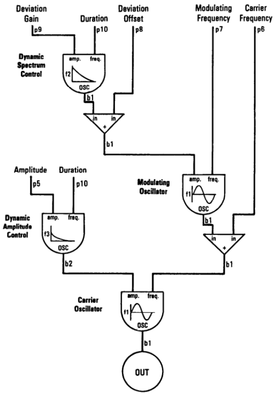
Digital Sound Synthesis

First Languages, Music III/IV/V

- 1960 : Music III introduces the concept of Unit Generators
- 1963 : Music IV, a port of Music III using a macro assembler
- 1968 : Music V written in Fortran (inner loops of UG in assembler)

```
ins 0 FM;
osc bl p9 p10 f2 d;
adn bl bl p8;
osc bl bl p7 fl d;
adn bl bl p6;
osc b2 p5 p10 f3 d;
osc bl b2 bl fl d;
out bl;
```

FM synthesis coded in CMusic



Csound

Originally developed by Barry Vercoe in 1985, Csound is today "a sound design, music synthesis and signal processing system, providing facilities for composition and performance over a wide range of platforms." (see `http://www.csounds.com`)

```
        instr  2
a1      oscil       p4, p5, 1   ; p4=amp
        out         a1          ; p5=freq
        endin
```

Example of Csound instrument

```
f1    0     4096  10 1        ; sine wave

;ins  strt  dur   amp(p4)     freq(p5)
i2    0     1     2000        880
i2    1.5   1     4000        440
i2    3     1     8000        220
i2    4.5   1     16000       110
i2    6     1     32000       55

e
```

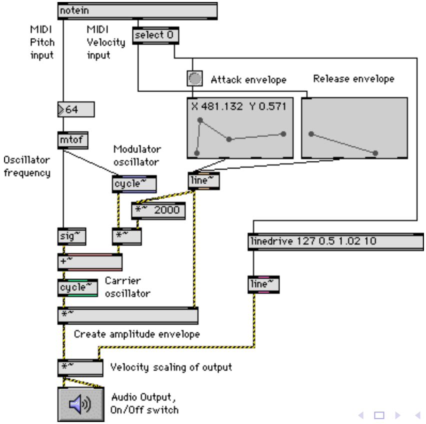
Example of Csound score

Supercollider

SuperCollider (John McCartney, 1996) is an open source environment and programming language for real time audio synthesis and algorithmic composition. It provides an interpreted object-oriented language which functions as a network client to a state of the art, realtime sound synthesis server. (see `http://supercollider.sourceforge.net/`)

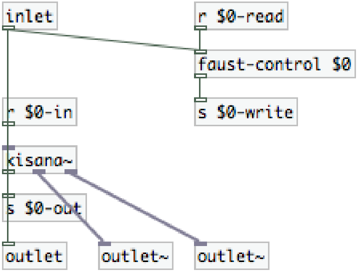
Max

Max (Miller Puckette, 1987), is visual programming language for real time audio synthesis and algorithmic composition with multimedia capabilities. It is named Max in honor of Max Mathews. It was initially developed at IRCAM. Since 1999 Max has been developed and commercialized by Cycling74. (see `http://cycling74.com/`)

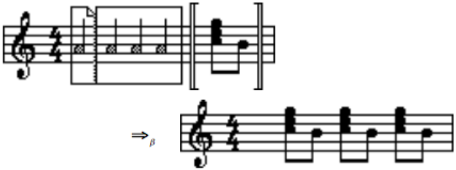
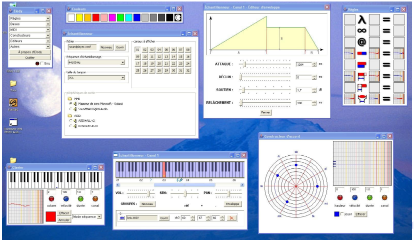


Puredata

Pure Data (Miller Puckette 1996) is an open source visual programming language of the Max family. "Pd enables musicians, visual artists, performers, researchers, and developers to create software graphically, without writing lines of code". (see `http://puredata.info/`)

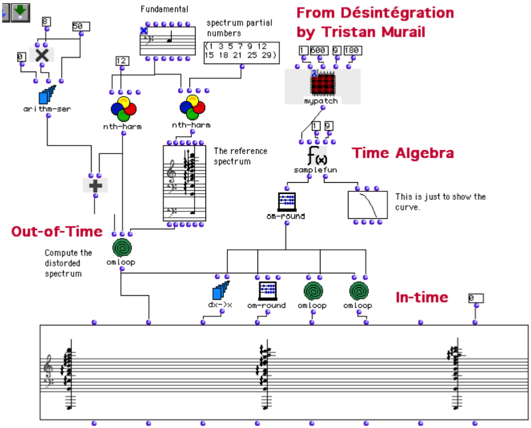


Elody

Elody (Fober, Letz, Orlarey, 1997) is a music composition environment developed in Java. The heart of Elody is a visual functional language derived from lambda-calculus. The languages expressions are handled through visual constructors and Drag and Drop actions allowing the user to play in realtime with the language.



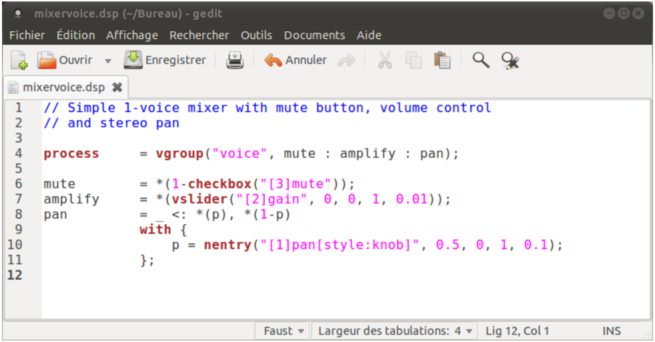
OpenMusic

OpenMusic (Agon et al. 1998) is a music composition environment based on Common Lisp. It introduces a powerful visual syntax to Lisp and provides composers with a large number of composition tools and libraries.



Faust

Faust (Orlarey et al. 2002) is a programming language that provides a purely functional approach to signal processing while offering a high level of performance. FAUST offers a viable and efficient alternative to C/C++ to develop audio processing libraries, audio plug-ins or standalone applications.



ChucK

ChucK (Ge Wang, Perry Cook 2003) is a concurrent, on-the-fly, audio programming language. It offers a powerful and flexible programming tool for building and experimenting with complex audio synthesis programs, and real-time interactive control. (see `http://chuck.cs.princeton.edu`)

```
// make our patch
SinOsc s => dac;

// time-loop, in which the osc's frequency
// is changed every 100 ms
while( true ) {
    100::ms => now;
    Std.rand2f(30.0, 1000.0) => s.freq;
}
```

Reactable

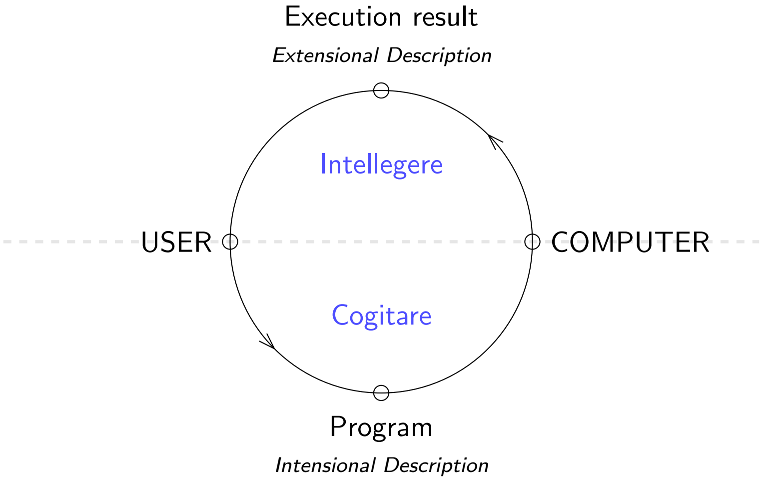
The Reactable is a tangible programmable synthesizer. It was conceived in 2003 by Sergi Jordà, Martin Kaltenbrunner, Günter Geiger and Marcos Alonso at the Pompeu Fabra University in Barcelona.

Creative Programming: Programming as a mean of Invention

Intensional vs Extensional Descriptions

- *Creative Programming* exploits the powerful relation between *Intensional Descriptions* and *Extensional Descriptions*.
- *Intensional Descriptions* are represented by programs.
- *Extensional Descriptions* result from the execution of these programs

Cogitare and Intellegere



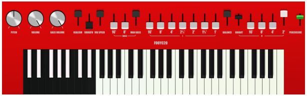
Overview of Faust

What is Faust used for?

- Faust is used on stage for concerts and artistic productions, for education and research, for open sources projects and commercial applications :
- Faust offers end-users a high-level alternative to C to develop audio applications for a large variety of platforms.
- The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, Javascript, etc.).

How is Faust Different ?

- Fully compiled to native code
- Sample level semantics
- Multiple backends: C++, WebAssembly, Rust, etc.
- Code runs on most platforms: from small embedded systems to web pages, mobile devices, plug-ins, standalone applications, etc.



DEMO 1

A very simple example

```
import("stdfaust.lib");
process = button("play") : pm.djembe(60,0.3,0.4,1);
```

https://faust.grame.fr/ide

The Design of Faust

Design Choices

- Purely functional approach focused on signal processing (LC)
- Programming by composition (FP, CL)
- A Compiled high-level specification language for end-users
- Well-defined preservable formal semantics
- Easy deployment

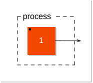
Purely Functional Approach

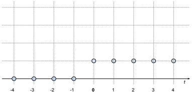
- Signals are functions: $\mathbb{S} = \mathrm{time} \rightarrow \mathrm{sample}$,
- Faust primitives are signal processors: $\mathbb{P} = \mathbb{S}^m \rightarrow \mathbb{S}^n$,
- Faust composition operations (`<:  :>  :  ,  ~`) are binary functions on signal processors: $\mathbb{A} = \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$,
- User defined functions are higher order functions on signal processors: $\mathbb{U} = \mathbb{P}^n \rightarrow \mathbb{P}$,
- A Faust program denotes a signal processor.

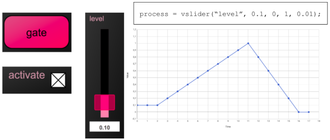
Faust Primitives

Generators: $\mathbb{S}^0 \to \mathbb{S}^1$

```
process = 1;
```





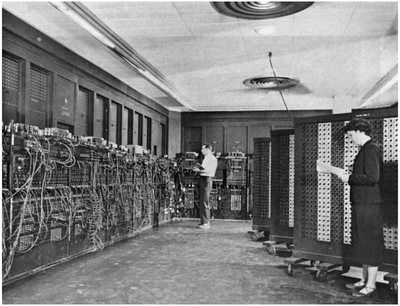


Operations: $\mathbb{S}^n \to \mathbb{S}^m$

- Arithmetic: `+`, `-`,`*`, `/`,...
- Comparison: `<`, `<=`,`!=`,...
- Trigonometric: `sin`, `cos`,...
- Log and Co.: `log`, `exp`,...
- Min, Max: `min`, `max`,...
- Selectors: `select2`,...
- Delays and Tables: `@`,...
- GUI: `button("...")`,...

Block-Diagram Algebra

Programming by patching is familiar to musicians :



Block-Diagram Algebra

Today programming by patching is widely used in Visual Programming Languages like Max/MSP:

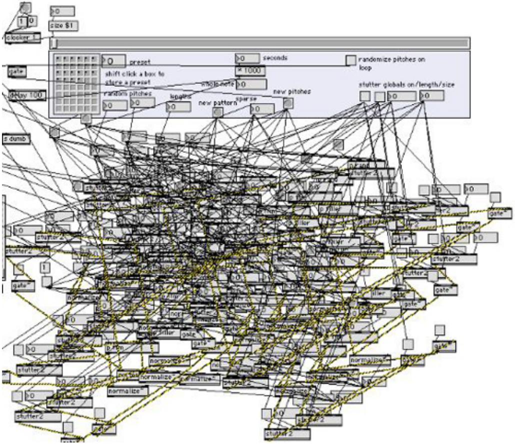


Figure: Block-diagrams can be a mess

Block-Diagram Algebra

Faust allows structured block-diagrams

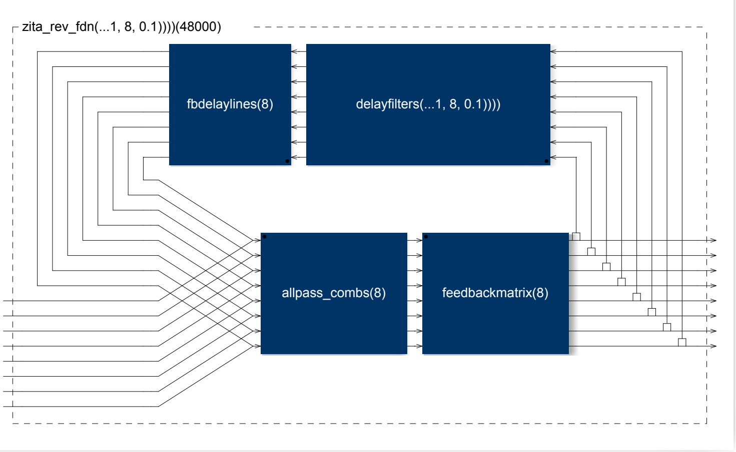


Figure: A complex but structured block-diagram

Block-Diagram Algebra

Faust syntax is based on a *block diagram algebra*

5 Composition Operators

- `(A~B)` recursive composition (priority 4)
- `(A,B)` parallel composition (priority 3)
- `(A:B)` sequential composition (priority 2)
- `(A<:B)` split composition (priority 1)
- `(A:>B)` merge composition (priority 1)

2 Constants

- `!` cut
- `_` wire

Block-Diagram Algebra

Parallel Composition

The *parallel composition* (A, B) is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

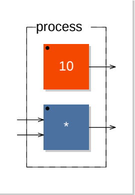


Figure: Example of parallel composition `(10,*)`

Block-Diagram Algebra

Sequential Composition

The *sequential composition* $(A : B)$ connects the outputs of A to the inputs of B. $A[0]$ is connected to $[0]B$, $A[1]$ is connected to $[1]B$, and so on.

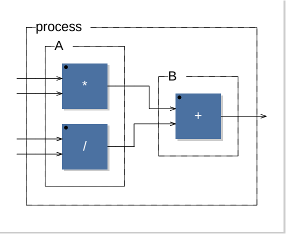


Figure: Example of sequential composition `((*,/):+)`

Note that the number of outputs of A must be equal to the number of inputs of B.

Block-Diagram Algebra

Split Composition

The *split composition* $(A <: B)$ operator is used to distribute A outputs to B inputs.

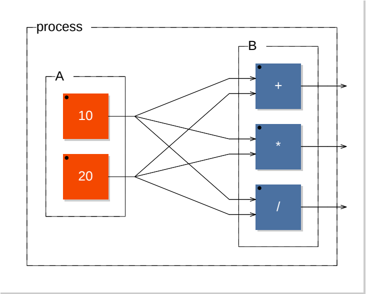


Figure: example of split composition `((10,20) <: (+,*,/))`

Block-Diagram Algebra

Merge Composition

The *merge composition* $(A :> B)$ is used to connect several outputs of A to the same inputs of B.

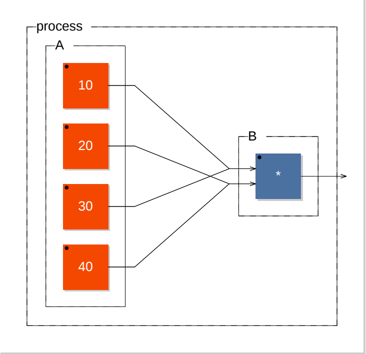


Figure: example of merge composition `((10,20,30,40) :> *)`

Block-Diagram Algebra

Recursive Composition

The *recursive composition* `(A~B)` is used to create cycles in the block-diagram in order to express recursive computations.

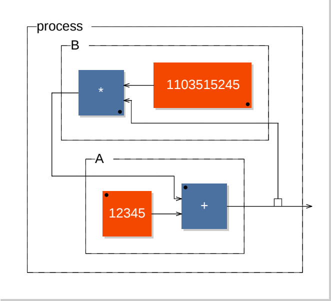


Figure: example of recursive composition `+(12345) ~ *(1103515245)`

DEMO 2

A simple echo. . .

```
process = + ~ (@(delay) : *(feedback))
with {
    delay = hslider("Delay[unit:s]", 0.5, 0.01, 1, 0.001)
          : *(44100) : int;
    feedback = hslider("Feedback[acc:0␣1␣-10␣0␣10]",0,0,0.65,0.01)
             : si.smooth(0.999);
}
```

Faust a language designed for Expressivity, Perfomance, Deployment and Ubiquity

Expressivity Quest

Language Expressivity

- Function Composition
- Partial application
- Lexical environments as first class citizen
- Pattern Matching
- Faust programs as components
- Local definitions

Language Expressiveness

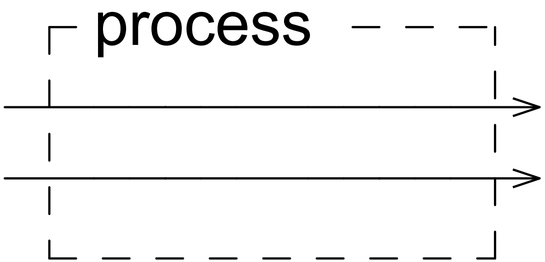
Fast Fourier Transform

```
fft(N) = si.cbus(N) : an.c_bit_reverse_shuffle(N) : fftb(N)
with {
    fftb(1) = _,_;
    fftb(N) = si.cbus(N)
            : (fftb(No2)<:(si.cbus(No2), si.cbus(No2))),
            (fftb(No2) <: (si.cbus(N):twiddleOdd(N)))
            :> si.cbus(N)
        with {
        No2 = int(N)>>1;
        twiddleOdd(N) = par(k,N,si.cmul(cos(w(k)),0-sin(w(k))));
        w(k) = 2.0*ma.PI*float(k)/float(N);
        };
};
```

Language Expressiveness

Fast Fourier Transform

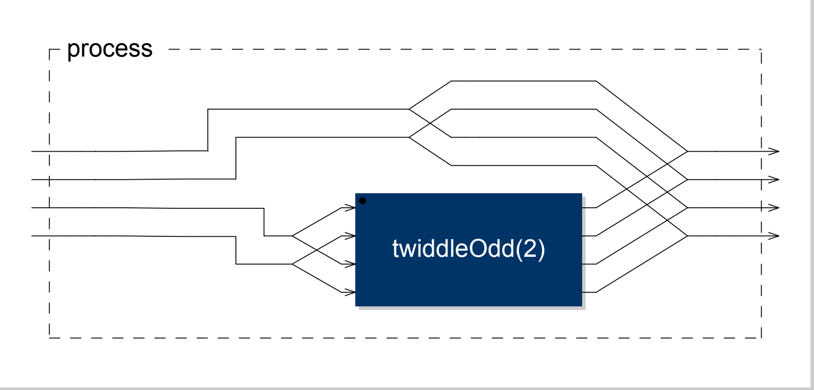
```
fft(1)
```



Language Expressiveness

Fast Fourier Transform

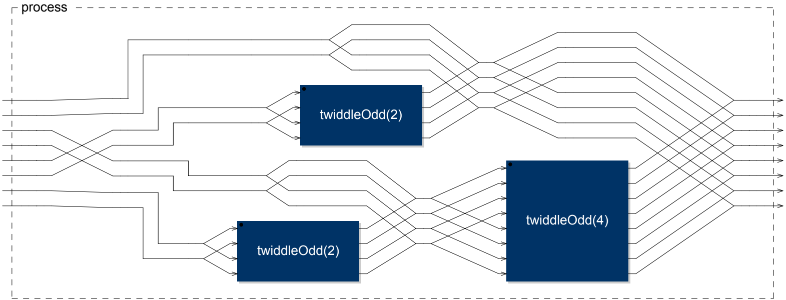
```
    fft(2)
```



Language Expressiveness

Fast Fourier Transform

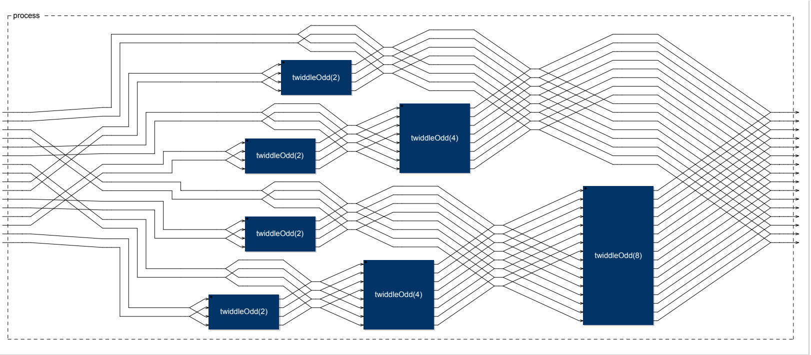
```
        fft(4)
```



Language Expressiveness

Fast Fourier Transform

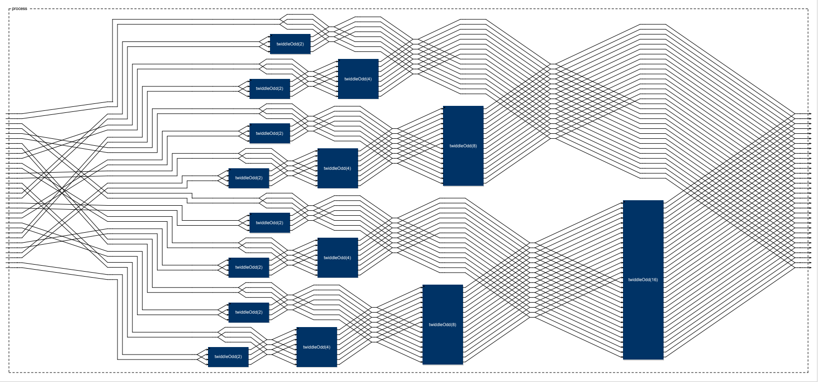
```
fft(8)
```



Language Expressiveness

Fast Fourier Transform

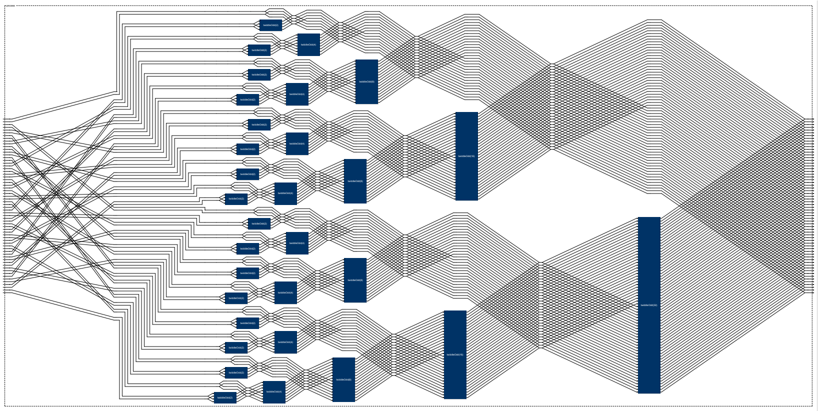
```
fft(16)
```



Language Expressiveness

Fast Fourier Transform

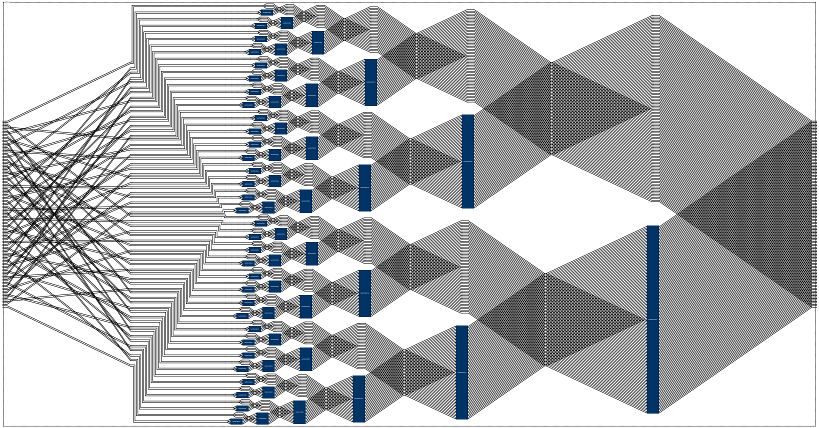
```
fft(32)
```



Language Expressiveness

Fast Fourier Transform

```
fft(64)
```



Performance Quest

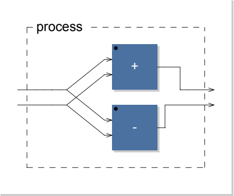
- Fully compiled to native code
- Sample level semantics
- Specification language
- Automatic parallelization

Fully compiled to native code

Faust code:

```
process = _,_ <: +,-;
```

Block-diagram:



C++ translation:

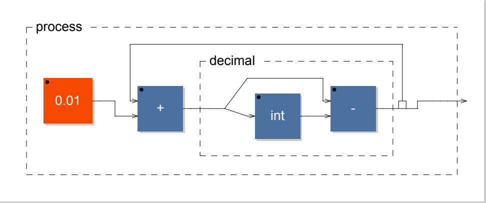
```
for (int i = 0; (i < count); i = (i + 1)) {
    float fTemp0 = input0[i];
    float fTemp1 = input1[i];
    output0[i] = fTemp0 + fTemp1;
    output1[i] = fTemp0 - fTemp1;
}
```

Sample level semantics

Sawtooth signal by step of 0.01:

```
decimal = _ <: _, int : -;
process = 0.01 : (+:decimal) ~ _;
```

Block-diagram:



Signal equation:

$$
\begin{aligned}
y(t < 0) &= 0 \\
y(t \geq 0) &= decimal(y(t-1) + 0.01)
\end{aligned}
$$

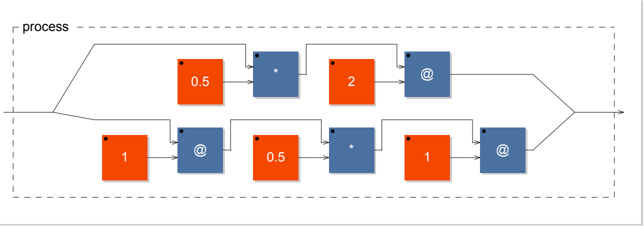
Specification Language

Leave the implementation to the compiler

User's code:

```
process = _<:(*(0.5):@(2)),(@(1):*(0.5):@(1)):>_;
```

Block-diagram:

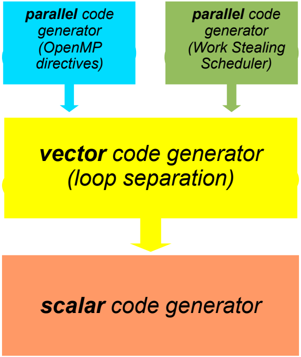


Equivalent, more efficient code

```
process = @(2);
```

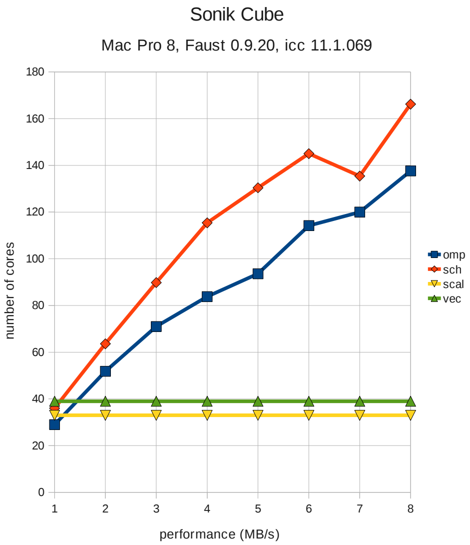
Automatic Parallelization

Code Generators



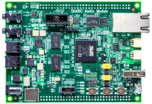
Automatic Parallelization

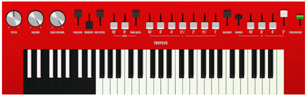
Performances



Easy Deployment Quest

Easy Deployment

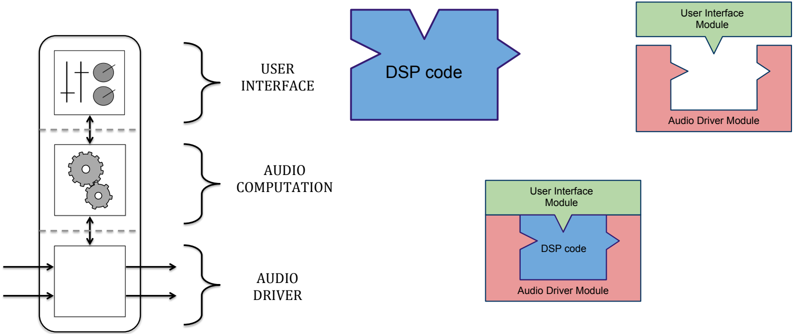




Easy Deployment

Separation of concern

The *architecture file* describes how to connect the audio computation to the external world.



Easy Deployment

Examples of supported architectures

- Audio plugins :
  - AudioUnit
  - LADSPA
  - DSSI
  - LV2
  - Max/MSP
  - VST
  - PD
  - Csound
  - Supercollider
  - Pure
  - Chuck
  - JUCE
  - Unity
- Devices :
  - OWL
  - MOD
  - BELA
  - SAM
- Audio drivers :
  - Jack
  - Alsa
  - CoreAudio
  - Web Audio API
- Graphic User Interfaces :
  - QT
  - GTK
  - Android
  - iOS
  - HTML5/SVG
- Other User Interfaces :
  - MIDI
  - OSC
  - HTTPD

Ubiquity:
Compiling Everywhere

Compiling Everywhere

Language Backends

- **C++**
- C
- Rust
- Java
- Javascript
- Asm.js
- **LLVM**
- **WebAssembly**
- ...

Compiling Everywhere

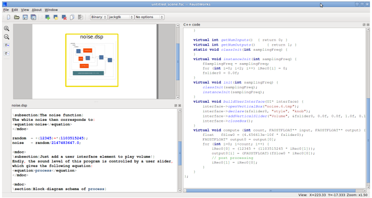
Libfaust

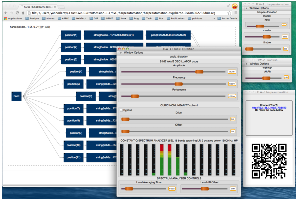
- Libfaust: embeddable version of the Faust compiler coupled with LLVM
- Libfaust.js: embeddable Javascript version of the Faust compiler

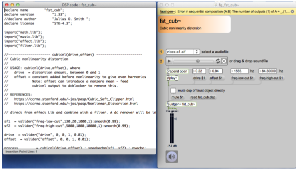
Compiling Everywhere

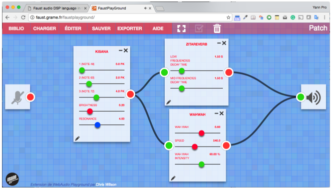
- Command Line Compilers
  - `faust` command line
  - `faust2xxx` command line
  - FaustWorks (IDE)
- Embedded Compilers (libfaust)
  - FaustLive (self contained)
  - Faustgen for Max/MSP
  - Faust for PD
  - Faustcompile, etc. for Csound (V. Lazzarini)
  - Faust4processing
  - Antescofo (IRCAM's score follower)
- Web Based Compilers
  - Faustweb API (`https://faustservice.grame.fr`)
  - Online Development Environment (`https://faust.grame.fr/ide`)
  - Online Editor (`https://faust.grame.fr/editor`)
  - Faustplayground (`https://faust.grame.fr/faustplyaground`)

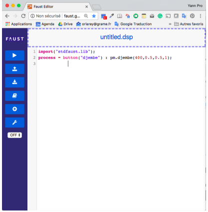
The Faust Ecosystem

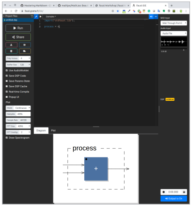












Additional Resources

Where to learn Faust

International:

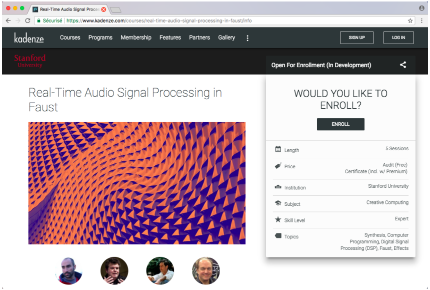
- Stanford U./CCRMA
- Maynooth University
- Louisiana State University
- Aalborg University

France:

- Jean Monnet U., Master RIM
- IRCAM, ATIAM
- PARIS 8

Where to learn Faust

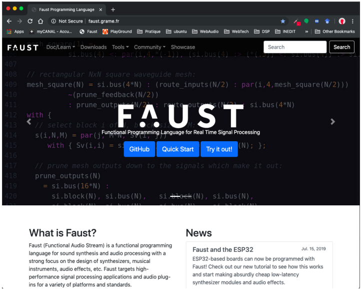
Kadenze course



https://www.kadenze.com/courses/real-time-audio-signal-processing-in-faust/info

Where to learn Faust

Faust website



https://faust.grame.fr