



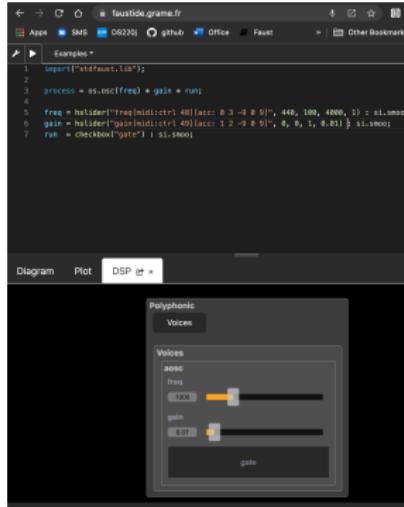
Functional  
Audio  
Stream

## An introduction to Faust

Yann Orlarey (GRAME), Tanguy Risset (INSA)  
Equipe Emeraude

Synchron 2021

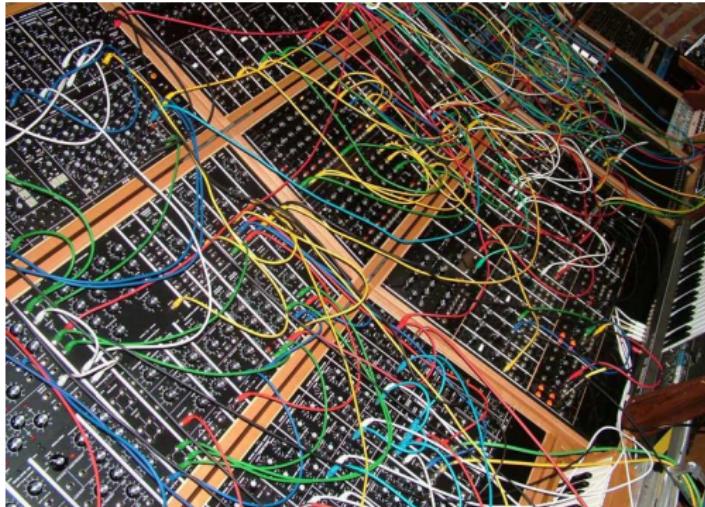
# What is Faust?



A screenshot of the faustide graphical interface. At the top, there's a browser-like header with tabs for Examples, Diagram, Plot, and DSP. Below the header, there's a code editor window containing the following Faust code:

```
1 import "stdfaust.ls";
2
3 process = os.oscifres * gain + run;
4
5 freq = halider("freq[minidicr 401[acc: 0 3 -9 0 0]", 440, 100, 4000, 1) : si.smoo;
6 gain = halider("gain[minidicr 401[acc: 1 2 -9 0 0]", 0, 0, 1, 0.81] : si.smoo;
7 run = checkbox("gate") : si.smoo;
```

Below the code editor is a graphical editor window titled "Polyphonic Voices". It shows two voices with parameters: freq (with a slider at 100) and gain (with a slider at 0.81). There are also buttons for "gate".



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

# 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
- 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

# Chronology

- 1960 : Music III (Max Mathews) Unit Generators ;
- 1968 : Music V written in Fortran ;
- 1985 : Csound (Barry Vercoe) a port of Music 11 ;
- 1986 : SuperCollider (John McCartney) OOP, client-server ;
- 1987 : Max (Miller Puckette) visual programming ;
- 1991 : Max/MSP (Miller Puckette) signal processing ;
- 1996 : Pure Data (Miller Puckette) open Source ;
- 2002 : Faust (Yann Orlarey) compiled, multi targets ;
- 2003 : Chuck (Ge Wang, Perry Cook) Live Coding ;
- 2011 : Gen (Graham Wakefield) compiled, multi targets.

# 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.).

# Faust is fully compiled

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



# Tutorial Part

## Using the Faust IDE

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

<https://faustide.grame.fr>

# 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

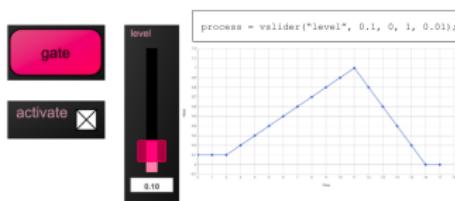
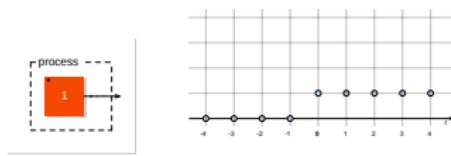
# Purely Functional Approach

- Signals are functions:  $\mathbb{S} = \text{time} \rightarrow \text{sample}$ ,
- 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:  $\mathbb{S}^0 \rightarrow \mathbb{S}^1$

```
process = 1;
```

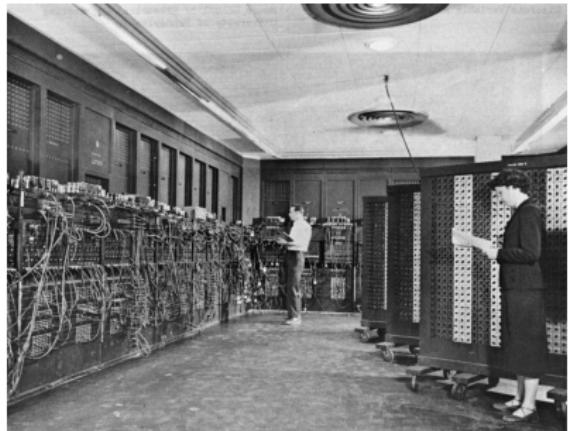


Operations:  $\mathbb{S}^n \rightarrow \mathbb{S}^m$

- 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:

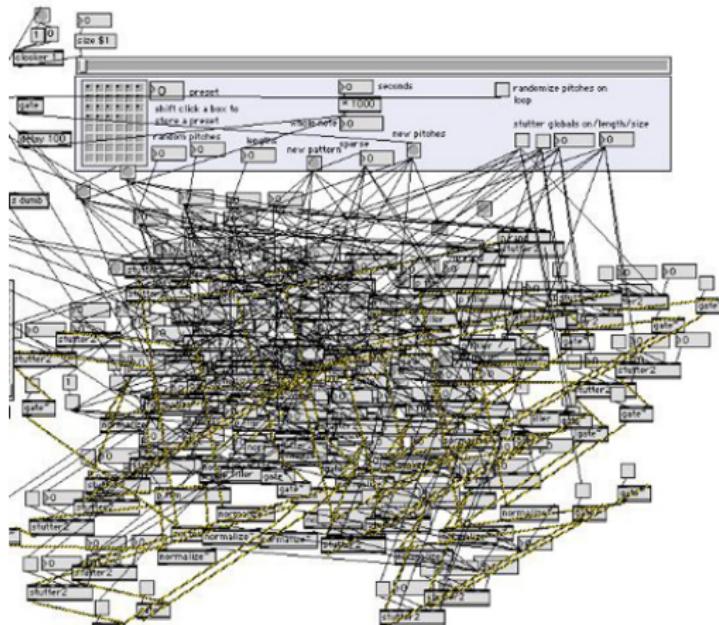


Figure: Block-diagrams can be a mess

# Block-Diagram Algebra

Faust allows structured block-diagrams

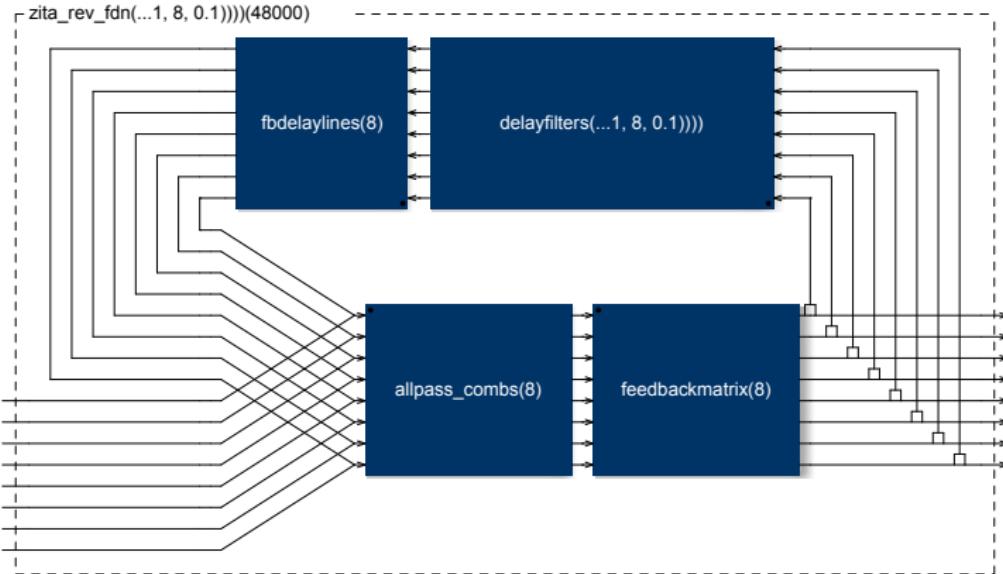


Figure: A complex but structured block-diagram

# Block-Diagram Algebra

Faust syntax is based on a *block diagram algebra*

## 5 Composition Operators

- $(A^{\sim}B)$  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.

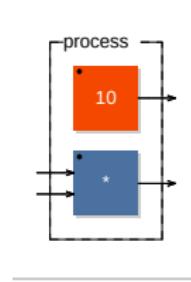


Figure: Example of parallel composition  $(10, *)$

# 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.

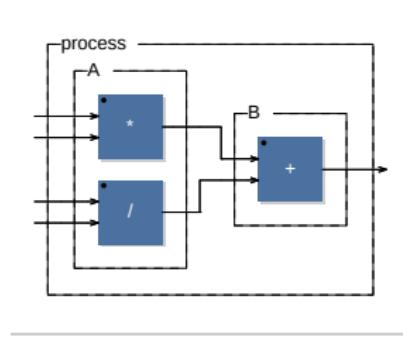


Figure: Example of sequential composition  $((*,/):+)$

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.

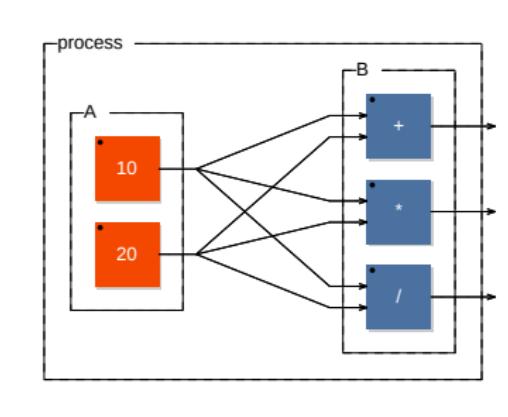


Figure: example of split composition  $((10, 20) <: (+, *, /))$

# Block-Diagram Algebra

## Merge Composition

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

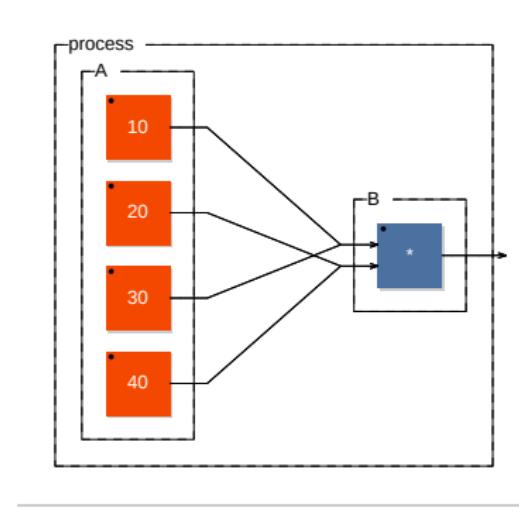


Figure: example of merge composition  $((10, 20, 30, 40) :> *)$

# Block-Diagram Algebra

## Recursive Composition

The *recursive composition* ( $A^\sim B$ ) is used to create cycles in the block-diagram in order to express recursive computations.

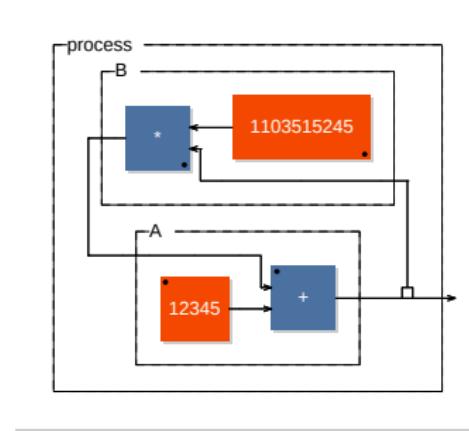


Figure: example of recursive composition  $+(12345) \sim *(1103515245)$

# 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, Performance,  
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

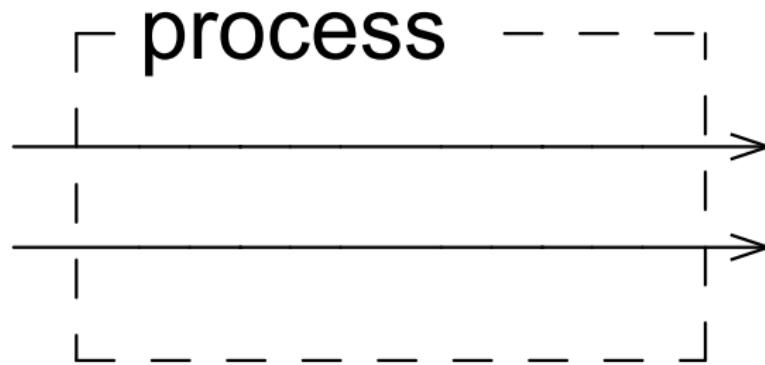
## Fast Fourier Transform

```
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

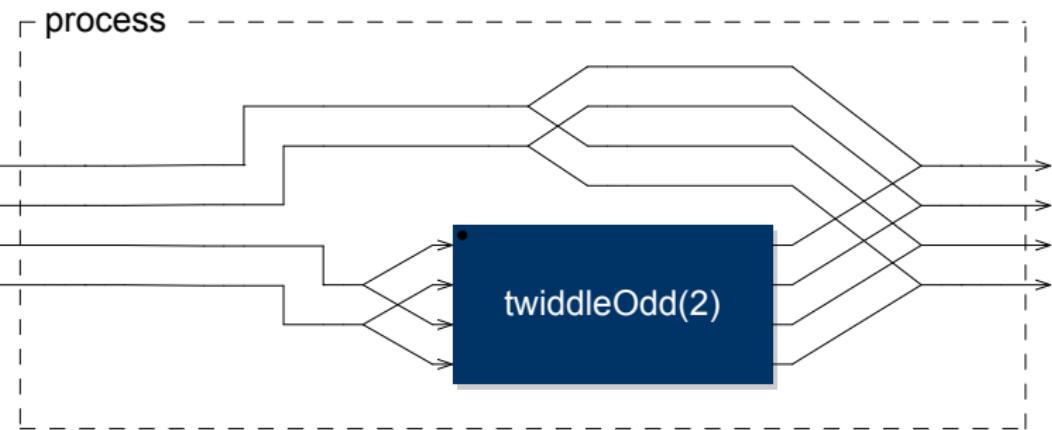
fft(1)



# Language Expressiveness

## Fast Fourier Transform

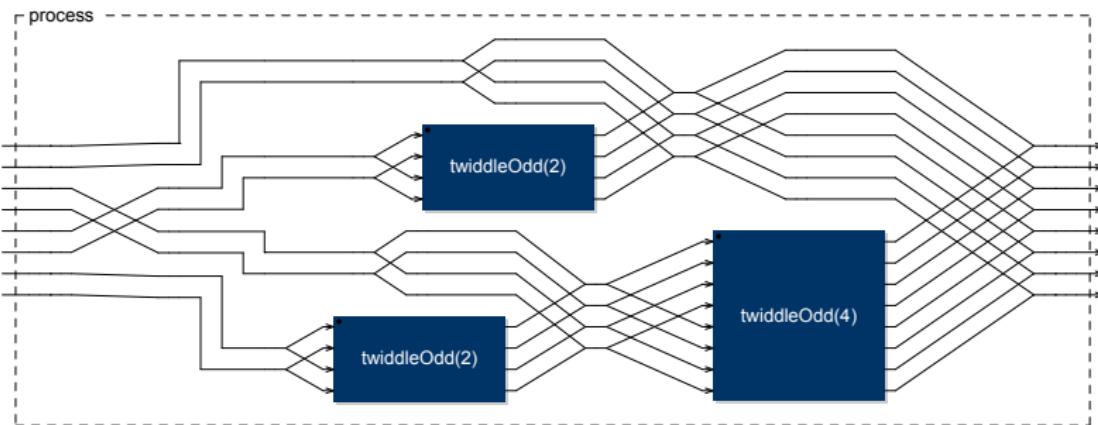
fft(2)



# Language Expressiveness

## Fast Fourier Transform

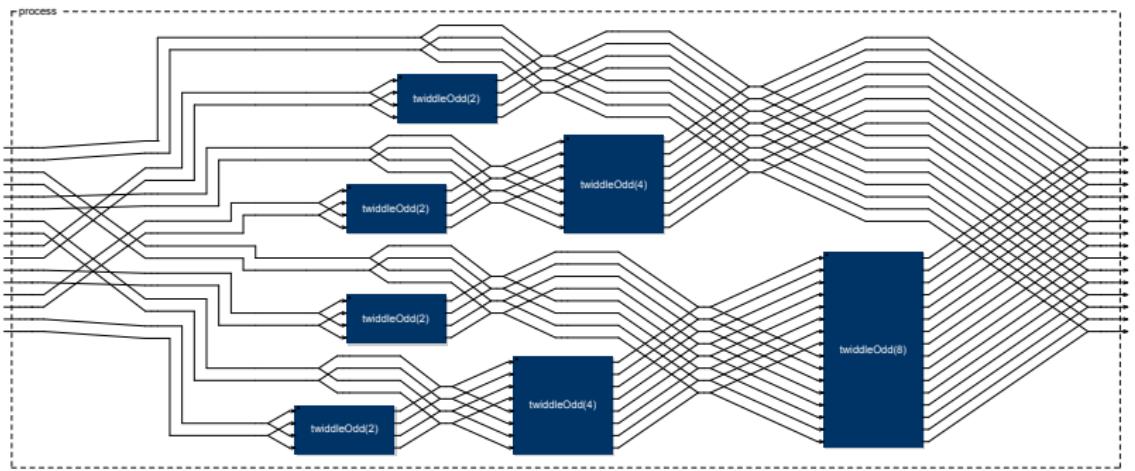
fft(4)



# Language Expressiveness

## Fast Fourier Transform

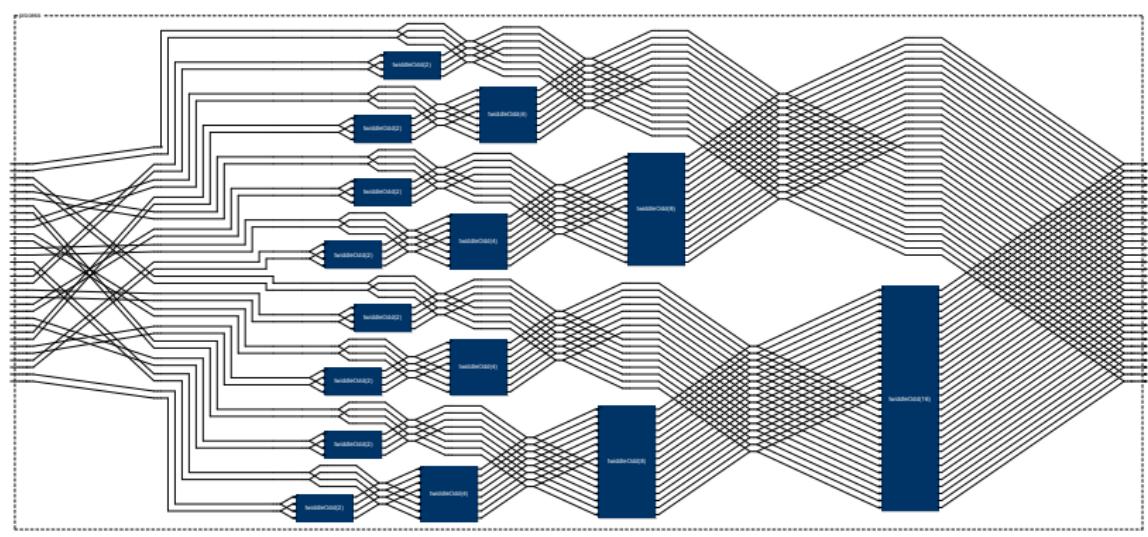
fft(8)



# Language Expressiveness

## Fast Fourier Transform

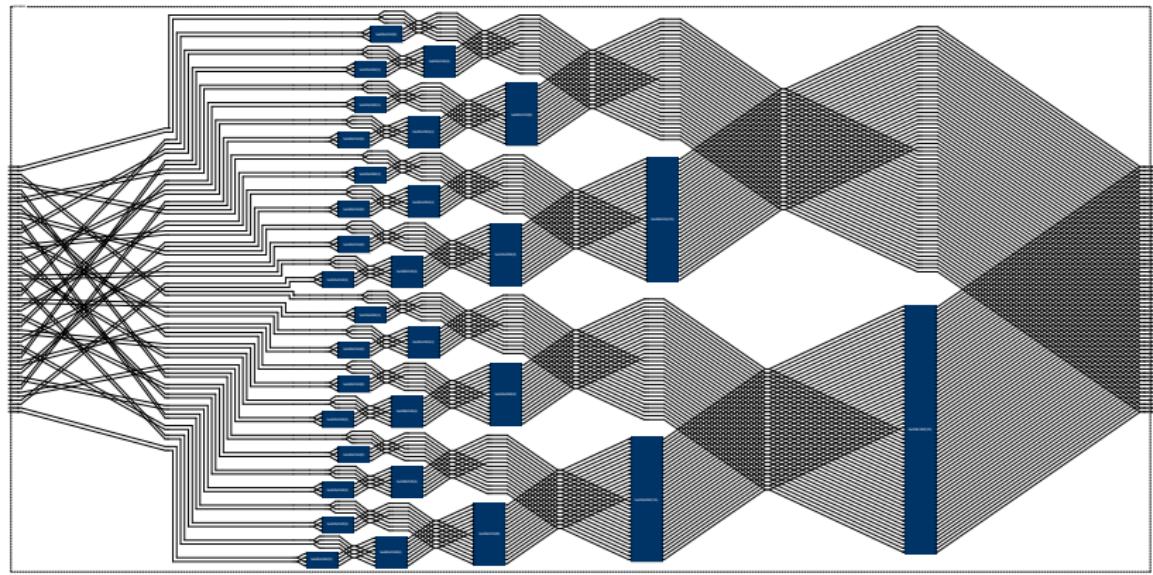
fft(16)



# Language Expressiveness

## Fast Fourier Transform

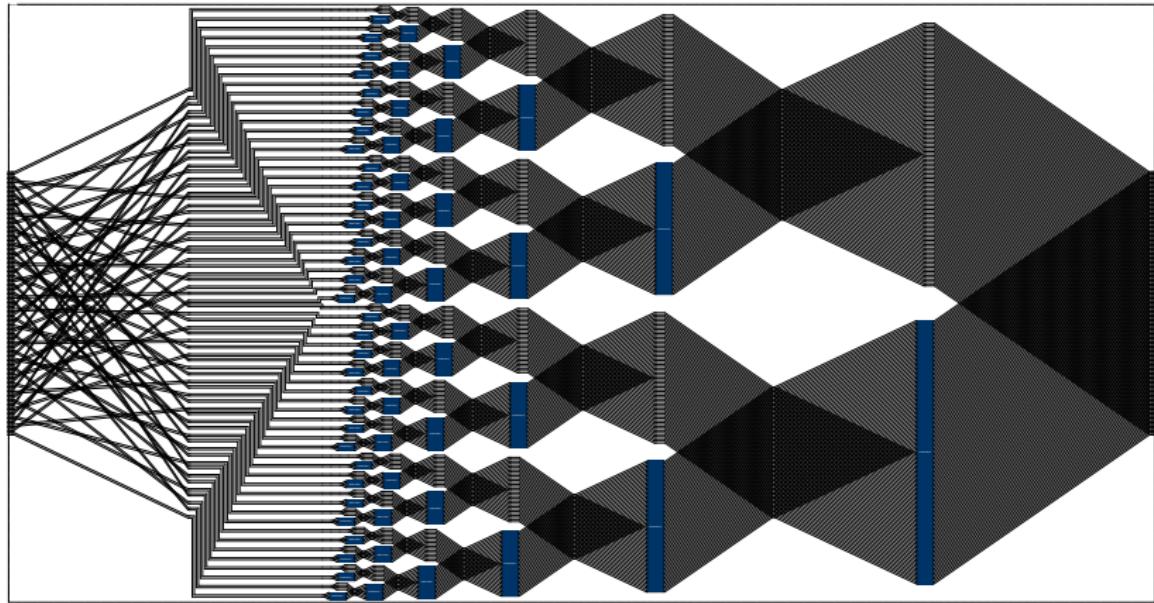
fft(32)



# Language Expressiveness

## Fast Fourier Transform

fft(64)



# Performance Quest

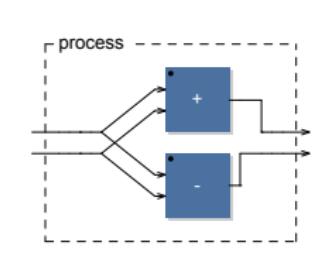
- Fully compiled to native code
- Sample level semantics
- Specification language
- Automatic parallelization

# Fully compiled to native code

Faust code:

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

Block-diagram:



C++ translation:

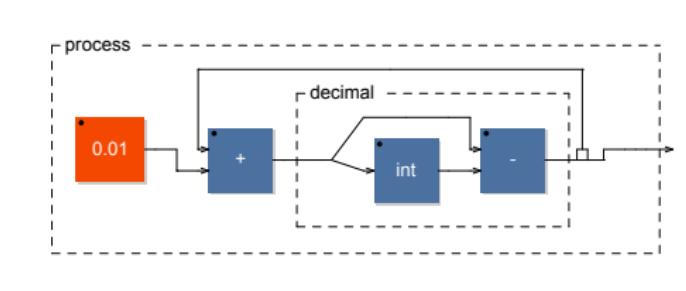
```
for (int i = 0; (i < count); i = (i + 1)) {  
    float fTemp0 = input0[i];  
    float fTemp1 = input1[i];  
    output0[i] = fTemp0 + fTemp1;  
    output1[i] = fTemp0 - fTemp1;  
}
```

# Sample level semantics

Sawtooth signal by step of 0.01:

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

Block-diagram:



Signal equation:

$$y(t < 0) = 0$$

$$y(t \geq 0) = decimal(y(t-1) + 0.01)$$

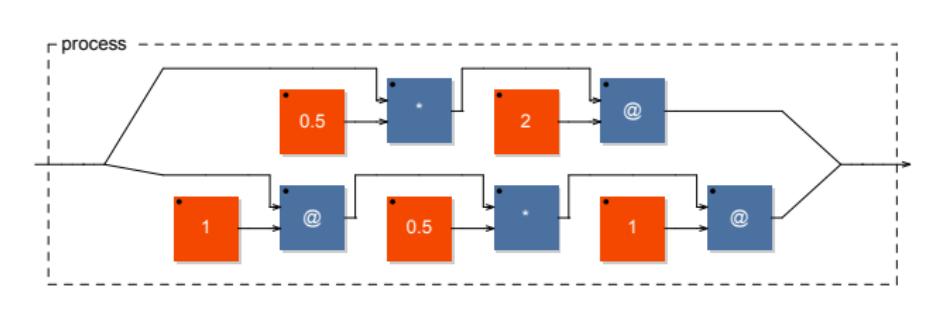
# Specification Language

Leave the implementation to the compiler

User's code:

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

Block-diagram:

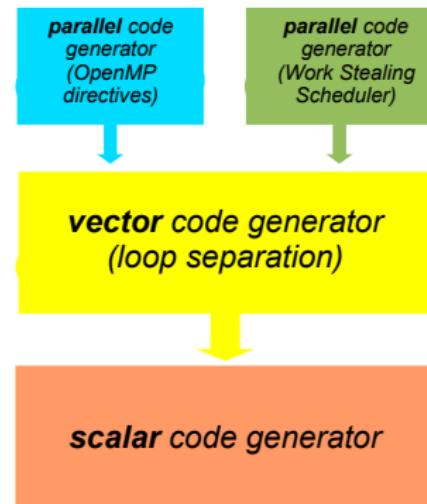


Equivalent, more efficient code

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

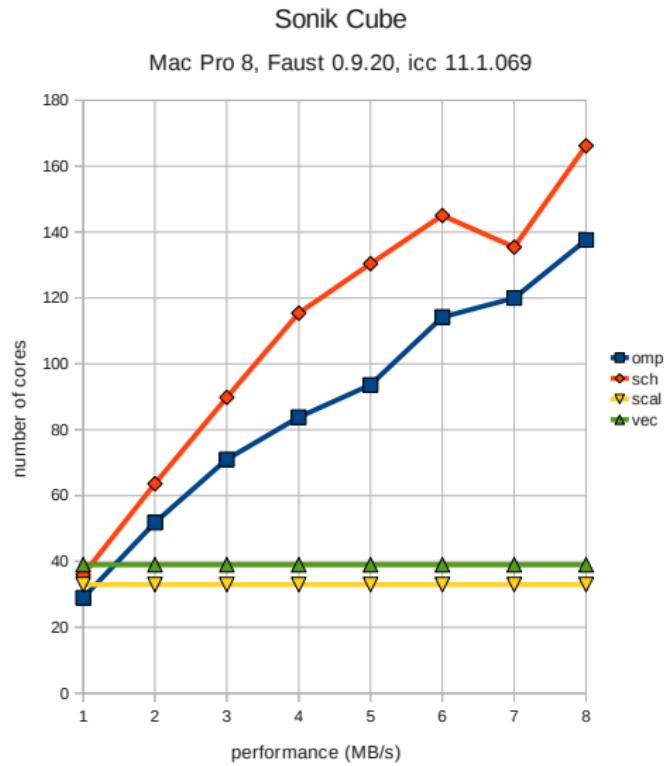
# Automatic Parallelization

## Code Generators



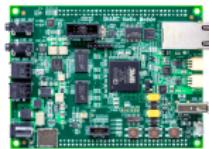
# Automatic Parallelization

## Performances



# Easy Deployment Quest

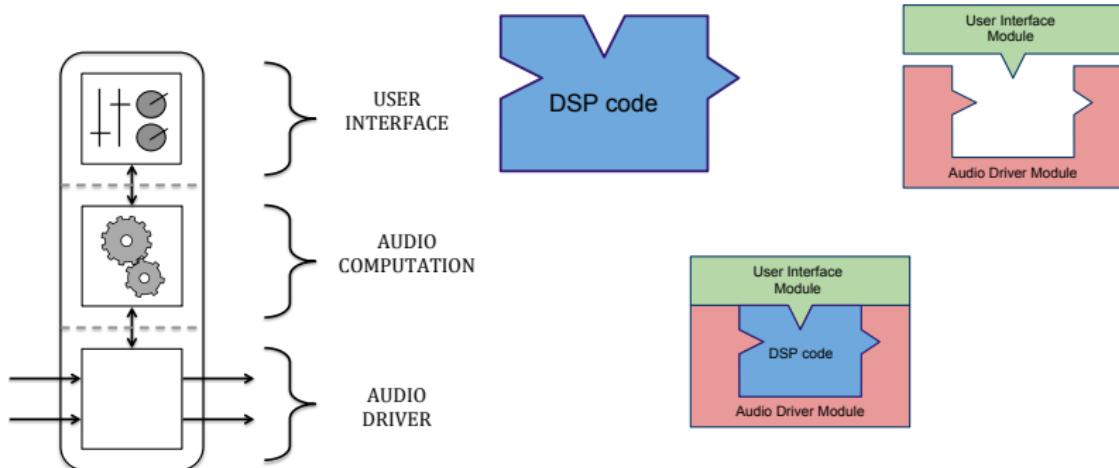
# Easy Deployment



# 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

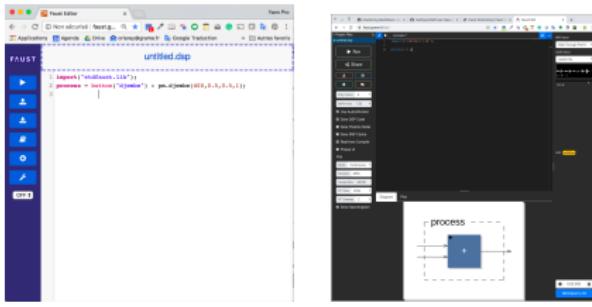
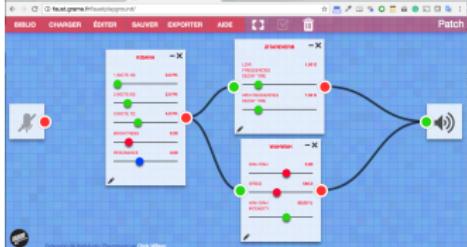
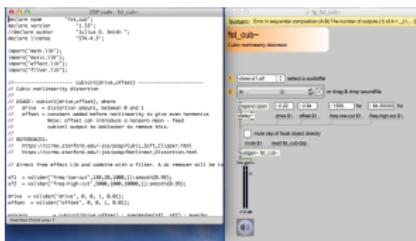
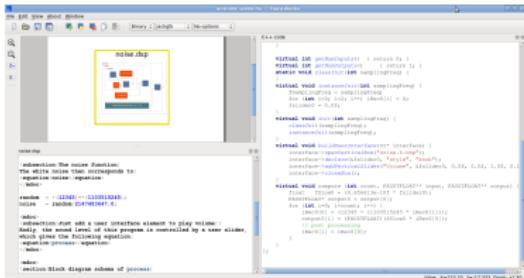
## Libfaust

- 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://faustide.grame.fr/>)
  - ▶ Online Editor (<https://fausteditor.grame.fr/>)
  - ▶ Faustplayground (<https://faustplayground.grame.fr/>)

# 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

The screenshot shows a web browser window for the Kadenze platform. The URL in the address bar is <https://www.kadenze.com/courses/real-time-audio-signal-processing-in-faust/info>. The page title is "Real-Time Audio Signal Processing in Faust". The page header includes the Kadenze logo, navigation links for Courses, Programs, Membership, Features, Partners, and Gallery, and buttons for SIGN UP and LOG IN. A banner at the top right indicates "Open For Enrollment (In Development)". The main content area features a large image of a colorful, abstract geometric pattern. Below the image are four circular profile pictures of course instructors. To the right, there is a sidebar titled "WOULD YOU LIKE TO ENROLL?" with an "ENROLL" button. The sidebar lists course details: Length (5 Sessions), Price (Audit (Free) Certificate (Ind. w/ Premium)), Institution (Stanford University), Subject (Creative Computing), Skill Level (Expert), and Topics (Synthesis, Computer Programming (OSSP), Faust, Effects).

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

# Where to learn Faust

Faust website

The screenshot shows the Faust Programming Language website. At the top, there's a search bar and a navigation menu with links like "Documentation", "Downloads", "Tools", "Community", and "Showcases". Below the menu, there's a code editor window displaying Faust code for a "rectangular NDN square waveguide mesh". The code uses various Faust constructs like `mesh\_square`, `par`, and `with`. To the right of the code editor, there's a large "FAUST" logo with the tagline "Functional Programming Language for Real Time Signal Processing". Below the logo, there are three buttons: "GitHub", "Quick Start", and "Try it out!". On the left side of the main content area, there's a section titled "What is Faust?" with a brief description of the language. On the right side, there's a "News" section with a single item: "Faust and the ESP32" dated Jul. 15, 2019. The news item discusses how Faust programs can run on the ESP32 and includes a link to a video.

What is Faust?

Faust (Functional Audio Stream) is a functional programming language for sound synthesis and audio processing with a strong focus on the design of synthesizers, musical instruments, audio effects, etc. Faust targets high-performance signal processing applications and audio plug-ins for a variety of platforms and interactors.

News

Faust and the ESP32 Jul. 15, 2019

ESP32 microcontrollers can now be programmed with Faust! Check out our new tutorial to see how this works and start making absurdly cheap low-latency synthesizer modules and audio effects.

<https://faust.grame.fr>