



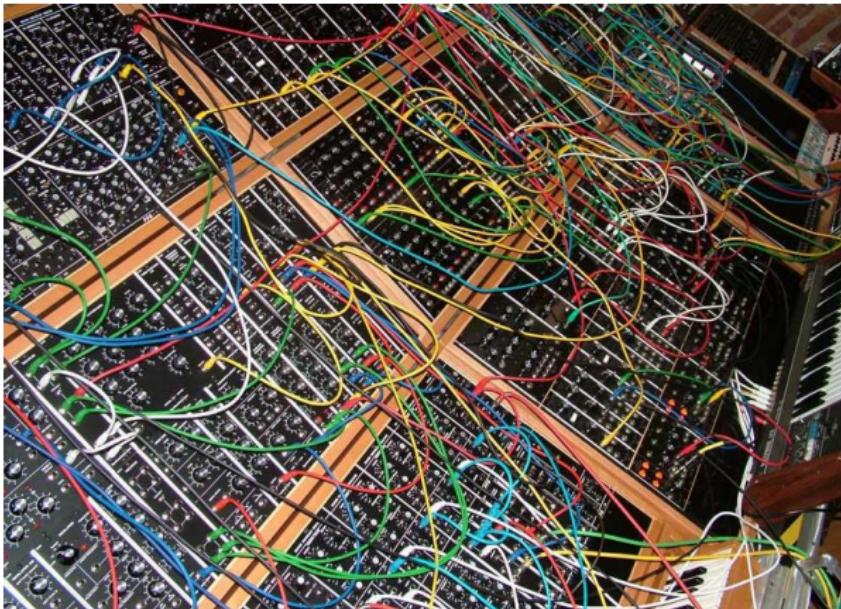
Functional
Audio
Stream

The Faust Programming Language

Yann Orlarey, EMERAUDE

INSA 2026

What is Faust?



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

Music Languages

Music Languages

Some examples

- | | | | | |
|--|--|---|--|--|
| <ul style="list-style-type: none">■ 4CED■ Adagio■ AML■ AMPLE■ Antescofo■ Arctic■ Autoklang■ Bang■ Canon■ CHANT■ Chuck■ CLCE■ C-major■ CMIX■ Cmusic■ CMUSIC■ Common Lisp Music■ Common Music■ Common Music Notation■ Csound■ CyberBand | <ul style="list-style-type: none">■ 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 | <ul style="list-style-type: none">■ 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 | <ul style="list-style-type: none">■ 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 | <ul style="list-style-type: none">■ PLAY2■ PMX■ POCO■ POD6■ POD7■ PROD■ Puredata■ PWGL■ Ravel■ RNBO■ SALIERI■ SCORE■ ScoreFile■ SCRIPT■ SIREN■ SMDL■ SMOKE■ SOUL■ SSSP■ ST■ SuperCollider■ Symbolic Composer■ Tidal |
|--|--|---|--|--|

Music Languages

Chronology

- 1960 : Music III (Mathews) Unit Generators ;
- 1968 : Music V written in Fortran ;
- 1985 : Csound (Vercoe) a port of Music 11 ;
- 1987 : Max (Puckette) visual programming ;
- 1991 : Max/MSP (Puckette) signal processing ;
- 1996 : SuperCollider (McCartney) OOP, client-server ;
- 1996 : Pure Data (Puckette) open Source ;
- 1997 : Open Music (Assayag, Agon,...) Visual Common Lisp
- 2002 : Faust (Orlarey, Letz, Fober) compiled, multi targets ;
- 2003 : Chuck (Wang, Cook) Live Coding ;
- 2008 : Antescofo (Cont) Score follower and language ;
- 2011 : Gen (Wakefield) compiled, multi targets;
- 2022 : RNBO (Cycling 74) compiled, multi targets.

Overview of Faust

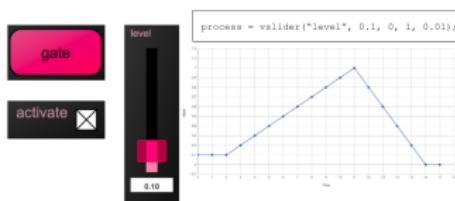
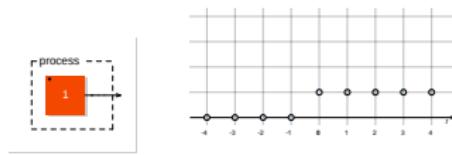
Building Audio Circuits with Faust

- A Faust program describes an *audio circuit* that operates on *audio signals*
 - ▶ *Signals* are functions of time: $\mathbb{S} = \mathbb{Z} \rightarrow \mathbb{R}$
 - ▶ *Audio circuits* are functions on signals: $\mathbb{C} = \mathbb{S}^m \rightarrow \mathbb{S}^n$
- Complex audio circuits are built by assembling primitive audio circuits using:
 - ▶ An algebra of five composition operations (`<:` `:` `:` `,` `~`):
 $\mathbb{A} = \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$
 - ▶ User-defined functions: $\mathbb{U} = \mathbb{C}^n \rightarrow \mathbb{C}$
- Using a user-friendly high-level programming language:
 - ▶ A purely functional approach (λC)
 - ▶ Programming by composition (FP, CL)
 - ▶ Efficient code generation
 - ▶ Well-defined preservable formal semantics
 - ▶ Easy deployment

Faust Primitives

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

```
process = 1;
```

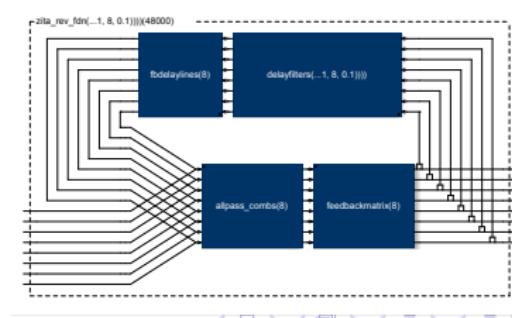
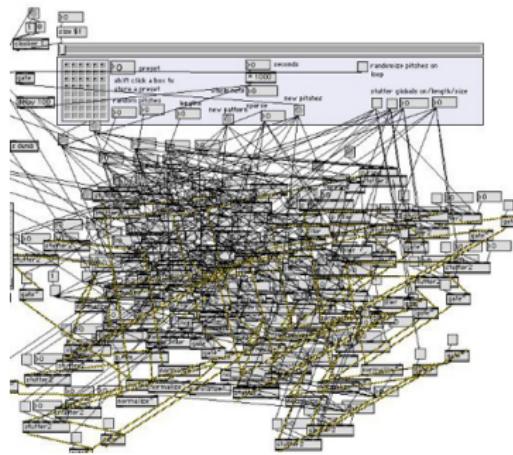
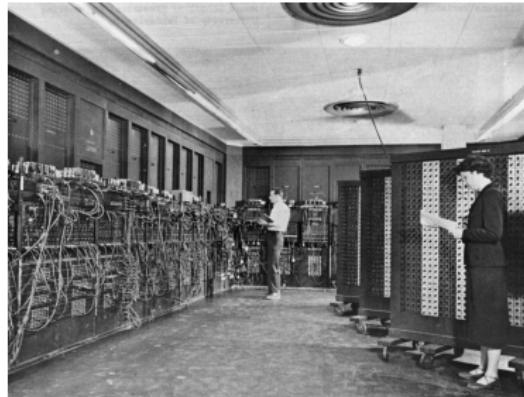


Operations: $\mathbb{S}^n \rightarrow \mathbb{S}^m$

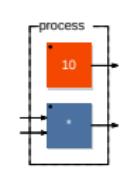
- Arithmetic: `+`, `-`, `*`, `/`, ...
- Comparison: `<`, `<=`, `!=`, ...
- Trigonometric: `sin`, `cos`, ...
- Log and Co.: `log`, `exp`, ...
- Min, Max: `min`, `max`, ...
- Selectors: `select2`, ...
- Delays and Tables: `@`, ...
- GUI: `button("...")`, ...

Programming by Composition

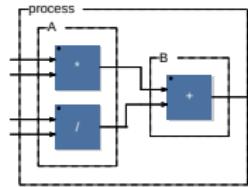
Programming by patching



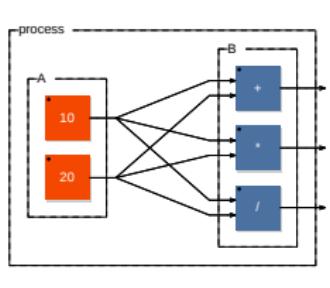
Programming by Composition



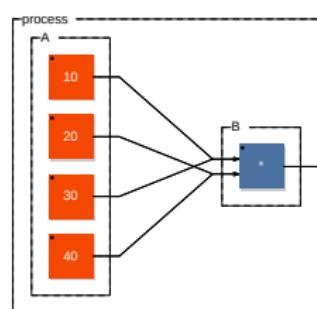
par: $(10, *)$



seq: $((*, /) : +)$

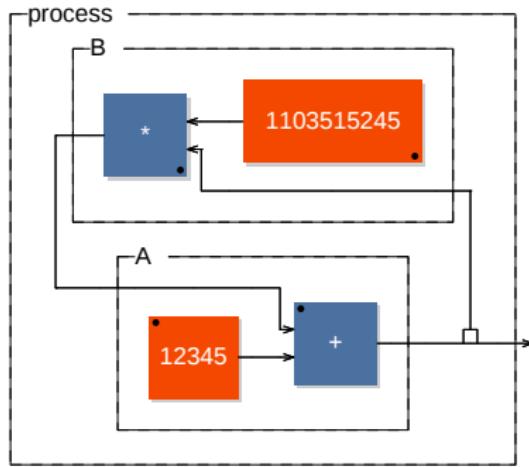


split: $((10, 20) <: (+, *, /))$



merge: $((10, 20, 30, 40) :> *)$

Programming by Composition



rec: +(12345) ~ *(1103515245)

$$y(t) = 12345 + y(t - 1) * 1103515245$$

Programming by Composition

5 Composition Operators

- **(A,B)** parallel composition:
 $(\mathbb{S}^n \rightarrow \mathbb{S}^m) \times (\mathbb{S}^{n'} \rightarrow \mathbb{S}^{m'}) \rightarrow (\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'})$
- **(A:B)** sequential composition:
 $(\mathbb{S}^n \rightarrow \mathbb{S}^m) \times (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$
- **(A<:B)** split composition:
 $(\mathbb{S}^n \rightarrow \mathbb{S}^m) \times (\mathbb{S}^{k.m} \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$
- **(A:>B)** merge composition:
 $(\mathbb{S}^n \rightarrow \mathbb{S}^{k.m}) \times (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$
- **(A~B)** recursive composition:
 $(\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'}) \times (\mathbb{S}^{m'} \rightarrow \mathbb{S}^{n'}) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^{m+m'})$

2 Constants

- **! cut:** $\mathbb{S}^1 \rightarrow \mathbb{S}^0$
- **_ wire:** $\mathbb{S}^1 \rightarrow \mathbb{S}^1$

Why use Faust?

Faust offers end-users a high-level alternative to C to develop realtime audio applications for a large variety of hardware and software platforms;



Expressivity Quest

Language Expressivity

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

Language Expressiveness

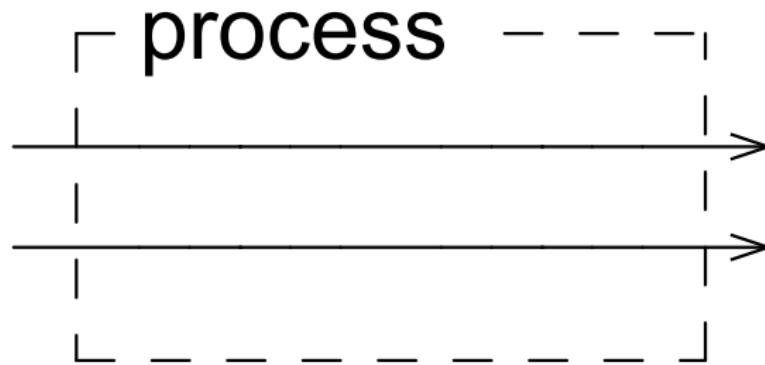
Fast Fourier Transform

```
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

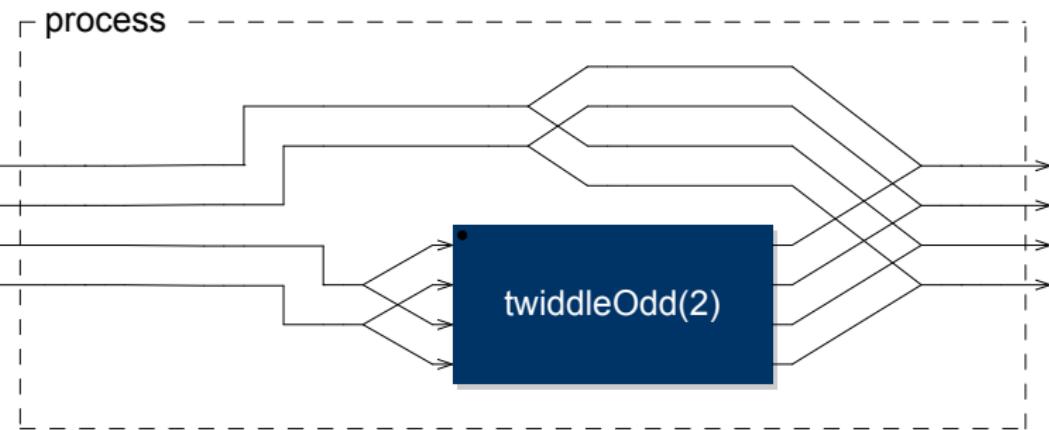
fft(1)



Language Expressiveness

Fast Fourier Transform

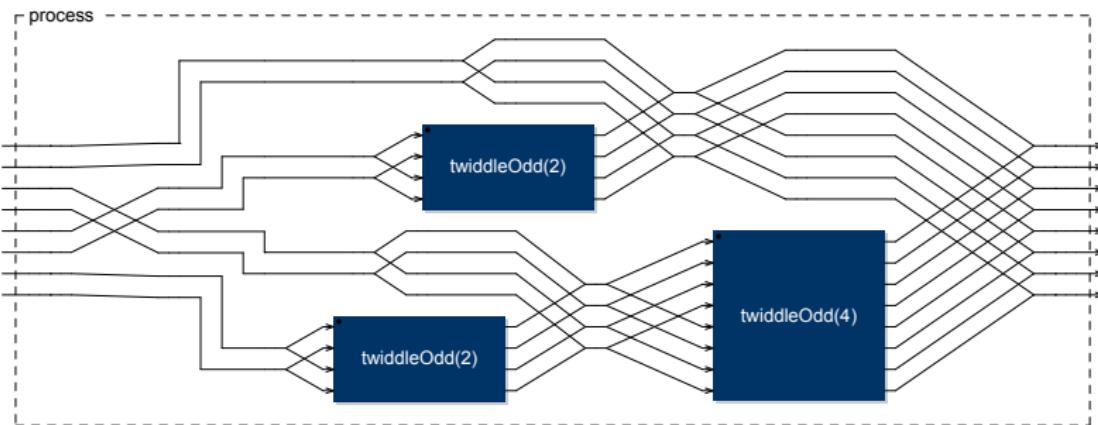
fft(2)



Language Expressiveness

Fast Fourier Transform

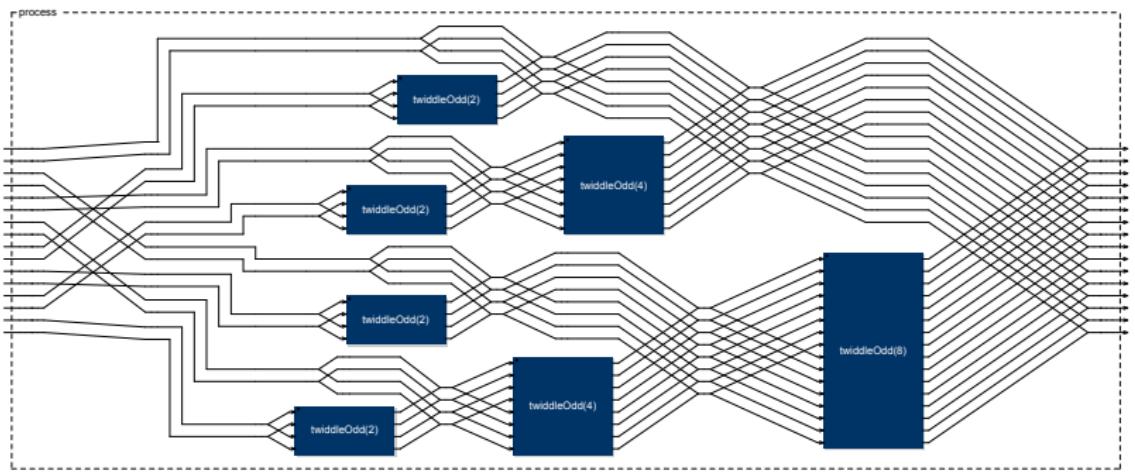
fft(4)



Language Expressiveness

Fast Fourier Transform

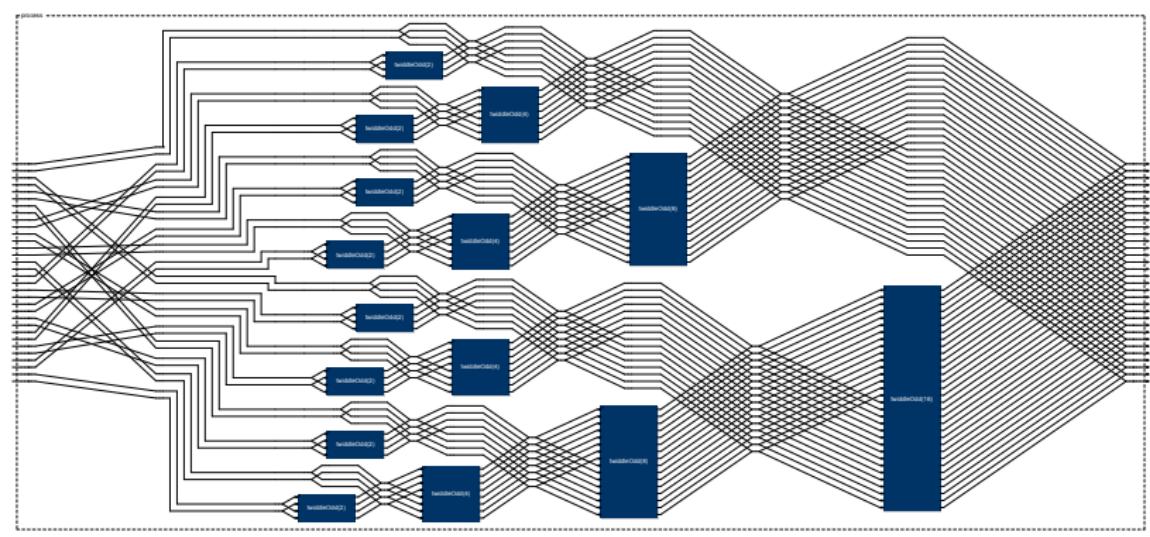
fft(8)



Language Expressiveness

Fast Fourier Transform

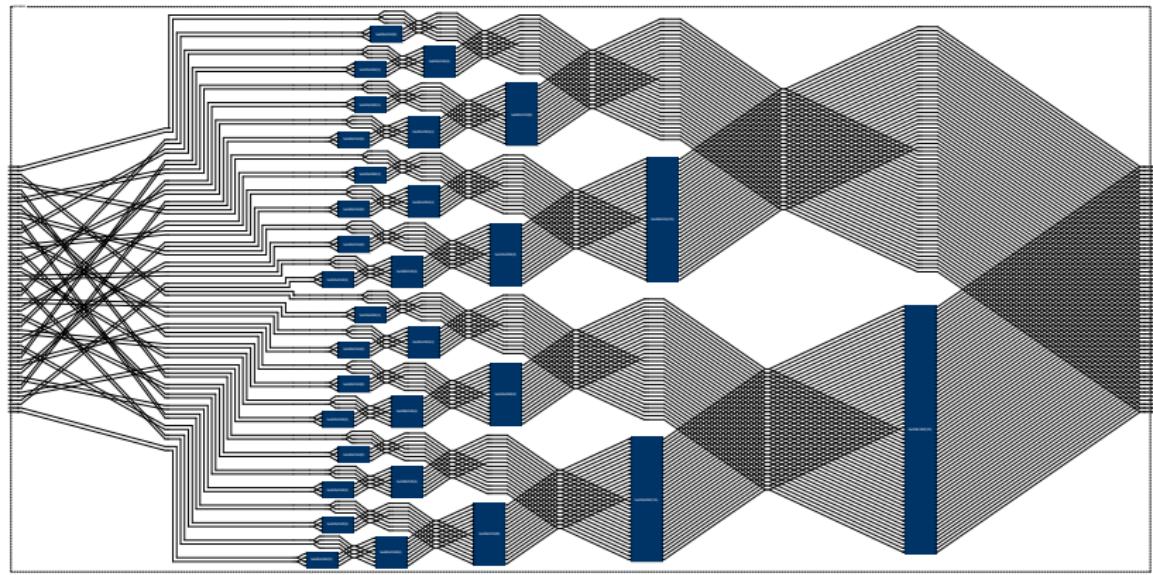
fft(16)



Language Expressiveness

Fast Fourier Transform

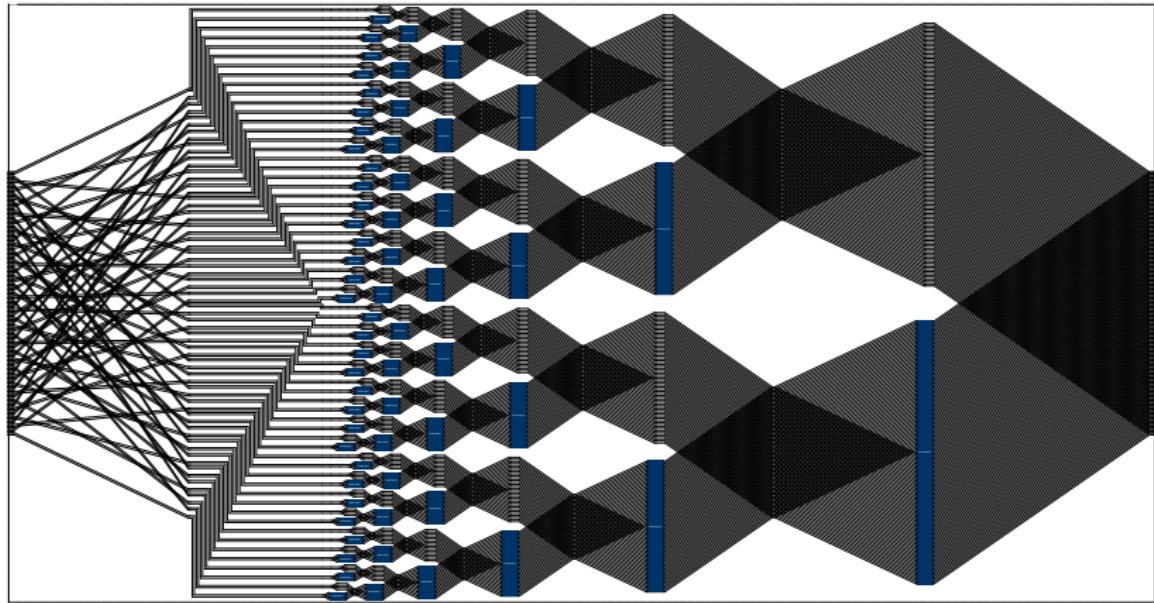
fft(32)



Language Expressiveness

Fast Fourier Transform

fft(64)



Performance Quest

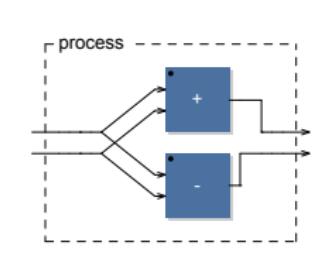
- Fully compiled to native code
- Sample level semantics
- Specification language
- Automatic parallelization

Fully compiled to native code

Faust code:

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

Block-diagram:



C++ translation:

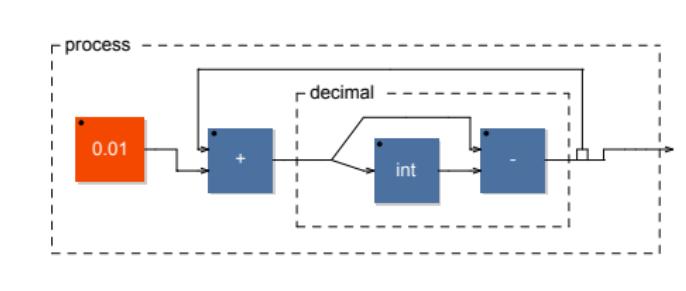
```
for (int i = 0; (i < count); i = (i + 1)) {  
    float fTemp0 = input0[i];  
    float fTemp1 = input1[i];  
    output0[i] = fTemp0 + fTemp1;  
    output1[i] = fTemp0 - fTemp1;  
}
```

Sample level semantics

Sawtooth signal by step of 0.01:

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

Block-diagram:



Signal equation:

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

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

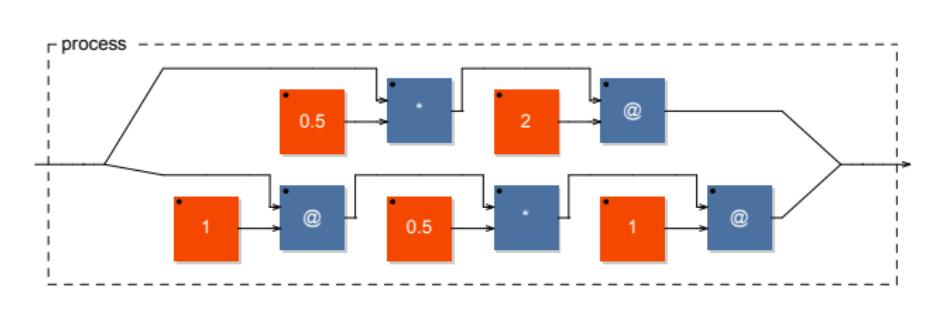
Specification Language

Leave the implementation to the compiler

User's code:

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

Block-diagram:

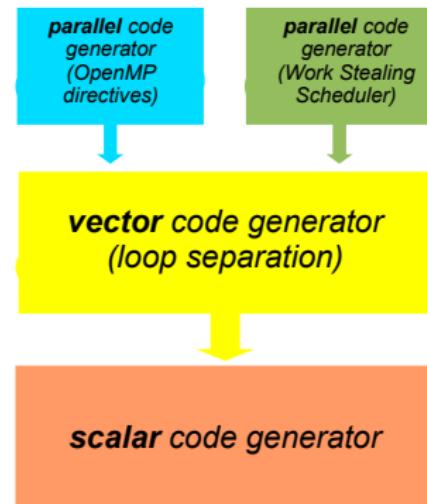


Equivalent, more efficient code

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

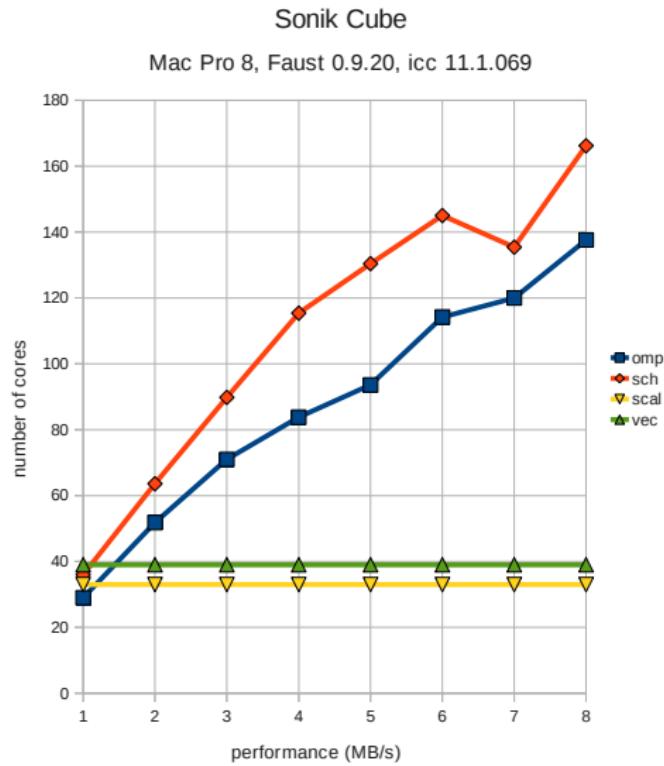
Automatic Parallelization

Code Generators



Automatic Parallelization

Performances



Easy Deployment Quest

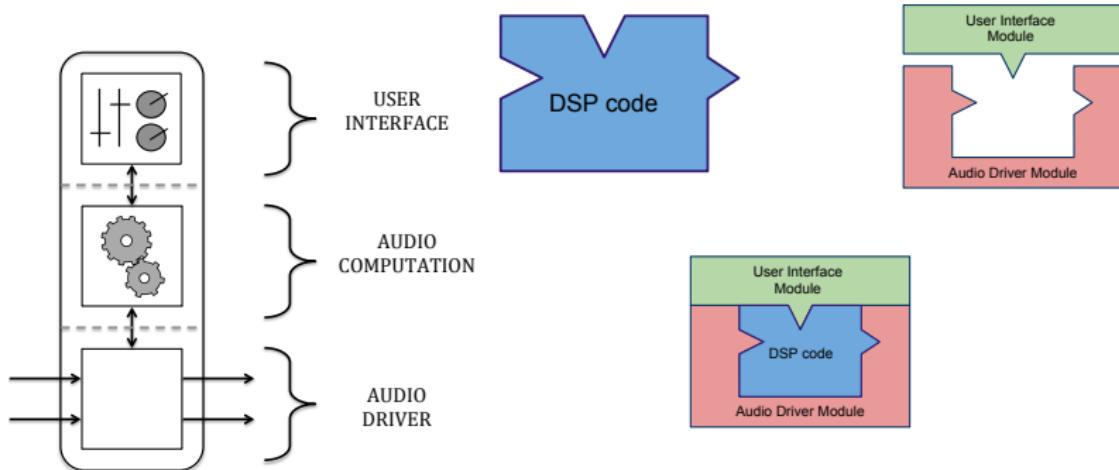
Easy Deployment



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

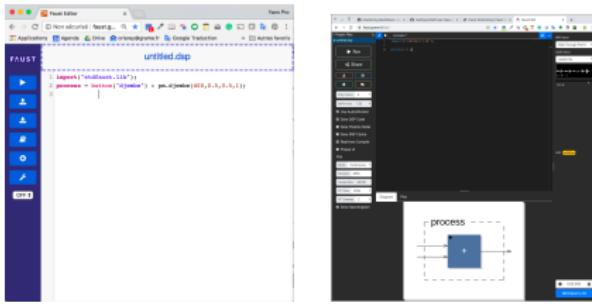
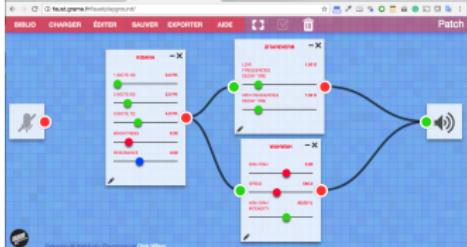
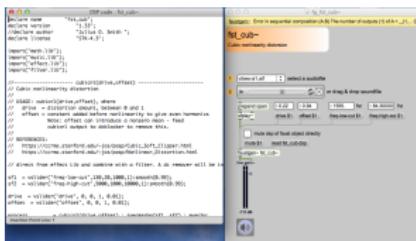
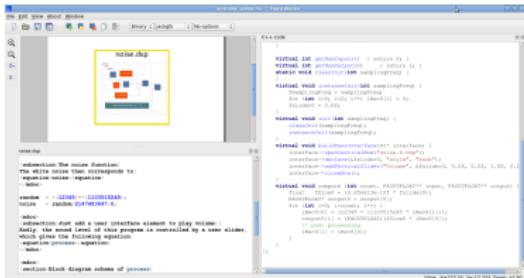
Libfaust

- 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 course is listed as "Open For Enrollment (In Development)". The main content area features a large, abstract graphic of orange and blue triangles. Below the graphic are four circular profile pictures of course instructors. To the right, there is a sidebar titled "WOULD YOU LIKE TO ENROLL?" with a "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 web interface. At the top, there's a navigation bar with links like 'Home', 'About', 'Noteworthy', 'Recent', 'Search', 'Documentation', 'API', 'Examples', 'Feedback', 'Help', 'Support', 'Discussions', 'Issues', 'Pull Requests', 'Contributors', and 'About'. Below the navigation is a search bar with two 'Search' buttons. The main content area has a large 'FAUST' logo. To its right is a 'Functional Programming Language for Real-Time Signal Processing' subtitle and a 'GitHub' button. The code editor contains FAUST code for a rectangular NIN square waveguide mesh. The code includes imports for `parIn`, `prune`, `feedback`, and `route`. It defines a mesh square and routes signals through it. A 'with' block selects a block, and another 'with' block handles SVL(1,1). The code ends with a 'prune' block that outputs signals down to the mesh. At the bottom left, there's a 'What is Faust?' section and a 'News' section. The 'News' section features a 'Faust and the ESP32' article by Julian Bittner. The bottom right corner has a 'Jul. 18, 2019' timestamp.

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 standards.

News

Faust and the ESP32 Jul. 15, 2018
ESP32-based boards 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>