



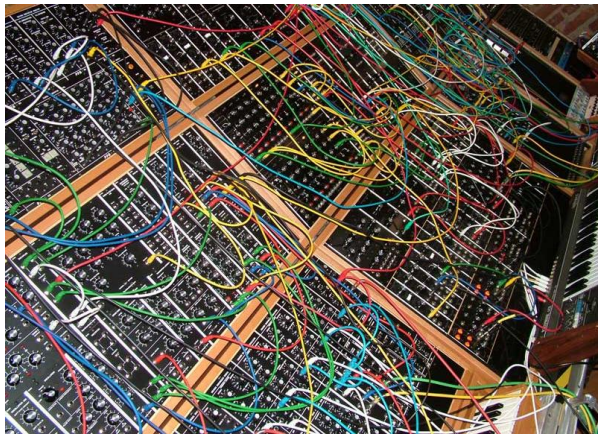
Functional
Audio
Stream

Electronic instruments and audio plug-ins design using Faust

Yann Orlarey, GRAME

BIENNALE COLLEGE – CIMM 2019

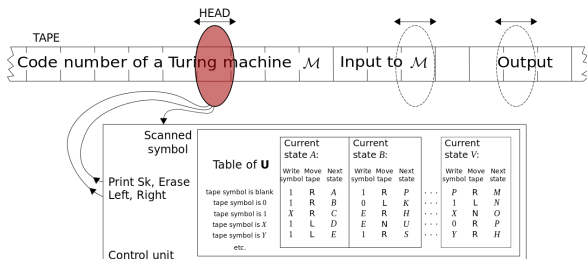
What is Faust?



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

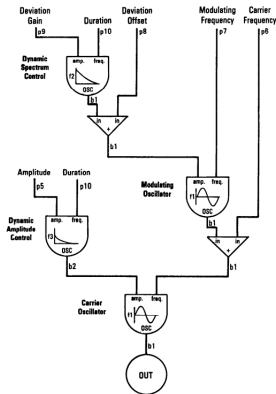
Digital Sound Synthesis

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 b1 p9 p10 f2 d;  
adn b1 b1 p8;  
osc b1 b1 p7 f1 d;  
adn b1 b1 p6;  
osc b2 p5 p10 f3 d;  
osc b1 b2 b1 f1 d;  
out b1;
```

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

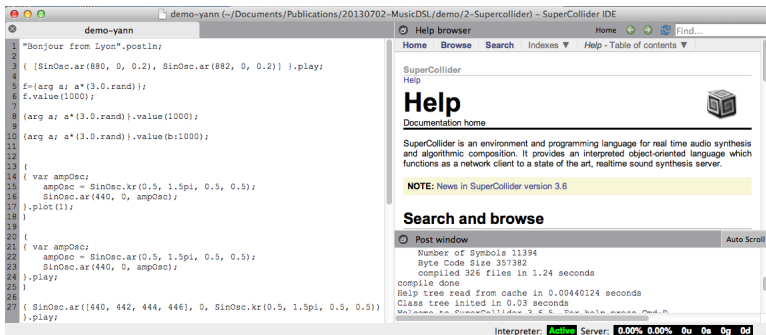
;ins strt dur  amp(p4)  freq(p5)
i2  0      1    2000     880
i2  1.5    1    4000     440
i2  3      1    8000     220
i2  4.5    1    16000    110
i2  6      1    32000     55

e
```

Example of Csound score

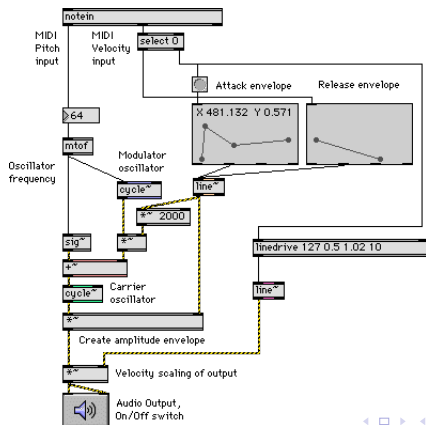
Supercollider

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/>)



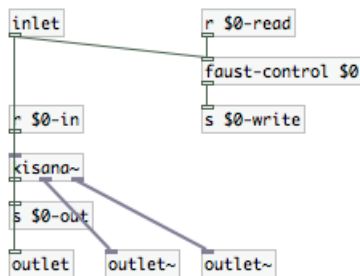
Max

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/>)



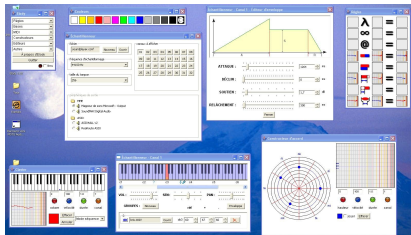
Puredata

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/>)



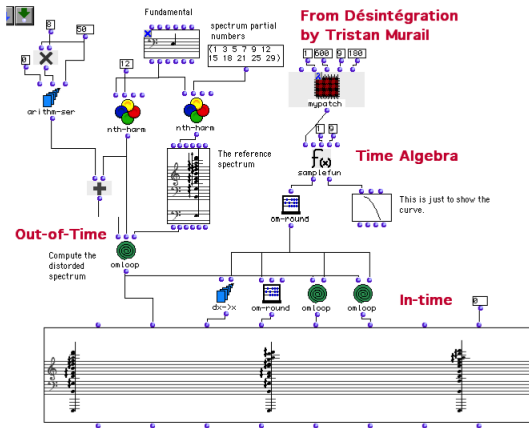
Elody

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.



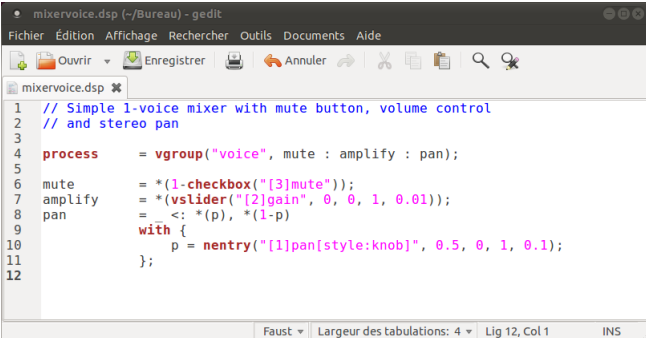
OpenMusic

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.



```
1 // Simple 1-voice mixer with mute button, volume control
2 // and stereo pan
3
4 process      = vgroup("voice", mute : amplify : pan);
5
6 mute        = *(1-checkbox("[3]mute"));
7 amplify     = *(vslider("[2]gain", 0, 0, 1, 0.01));
8 pan         = <: *(p), *(1-p)
9             with {
10                 p = nentry("[1]pan[style:knob]", 0.5, 0, 1, 0.1);
11             };
12
```

The screenshot shows a gedit editor window titled 'mixervoice.dsp (~/.Bureau) - gedit'. The menu bar includes 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Outils', 'Documents', and 'Aide'. The toolbar contains icons for opening, saving, printing, undo, redo, cut, copy, paste, search, and zoom. The file 'mixervoice.dsp' is open, and the code is displayed with line numbers 1 through 12. The code defines a 'process' block that combines a 'voice' group, a 'mute' checkbox, a 'gain' slider, and a 'pan' knob. The status bar at the bottom indicates 'Faust', 'Largeur des tabulations: 4', 'Lig 12, Col 1', and 'INS'.

Chuck

ChuckK (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.

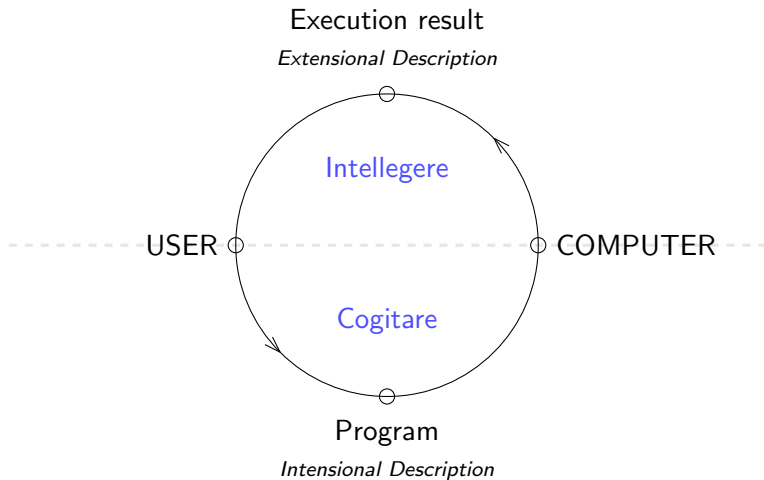


Creative Programming: Programming as a mean of Invention

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

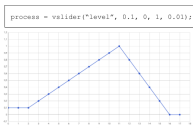
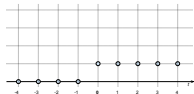
Purely Functional Approach

- Signals are functions: $\mathbb{S} = \text{time} \rightarrow \text{sample}$,
- Faust primitives are signal processors: $\mathbb{P} = \mathbb{S}^m \rightarrow \mathbb{S}^n$,
- Faust composition operations ($<: \ :> : , \sim$) 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;
```

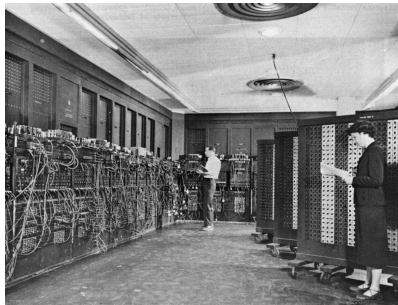


Operations: $\mathbb{S}^n \rightarrow \mathbb{S}^m$

- 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:

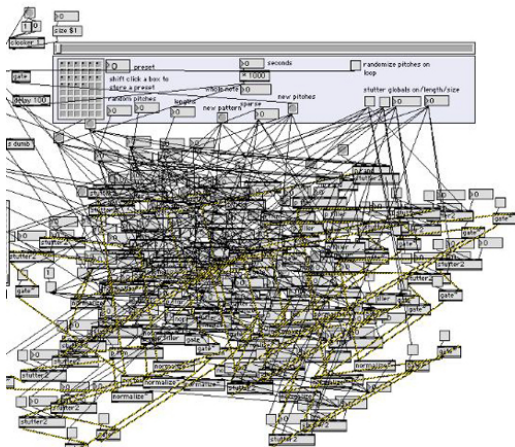


Figure: Block-diagrams can be a mess

Block-Diagram Algebra

Faust allows structured block-diagrams

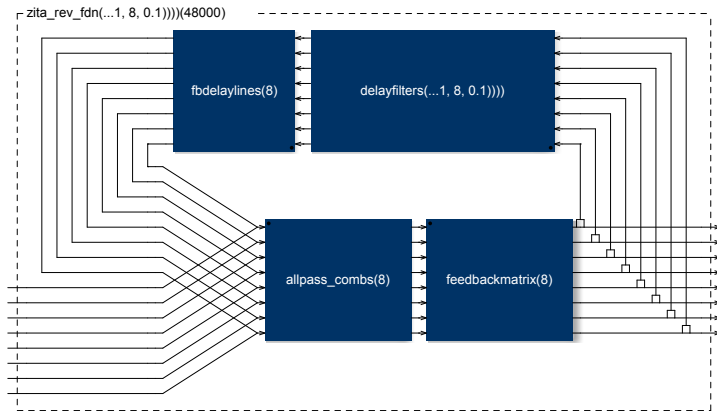


Figure: A complex but structured block-diagram

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.

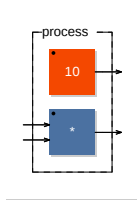


Figure: Example of parallel composition $(10, *)$

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.

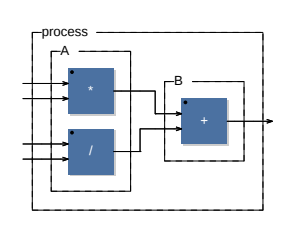


Figure: Example of sequential composition $((*, /) : +)$

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* ($A \leq B$) operator is used to distribute A outputs to B inputs.

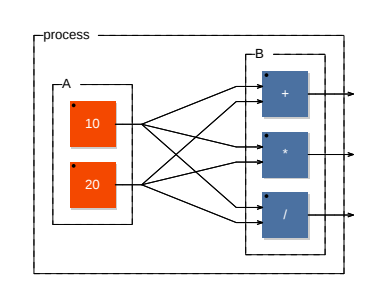


Figure: example of split composition $((10,20) \leq (+,*,/))$

Block-Diagram Algebra

Merge Composition

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

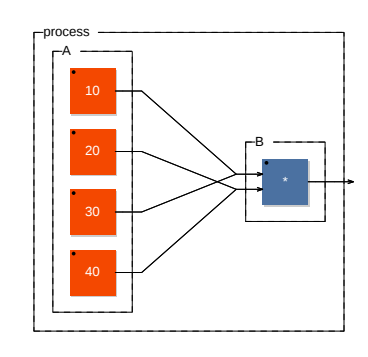


Figure: example of merge composition $((10, 20, 30, 40) :> *)$

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.

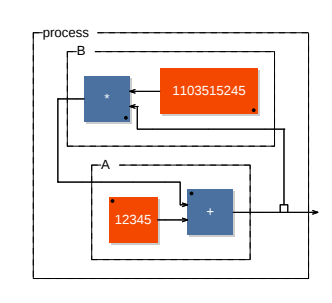


Figure: example of recursive composition $+(12345) \sim *(1103515245)$

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

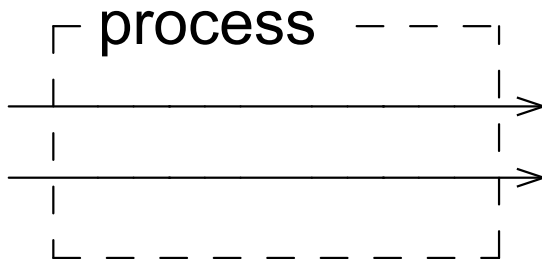
Fast Fourier Transform

```
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

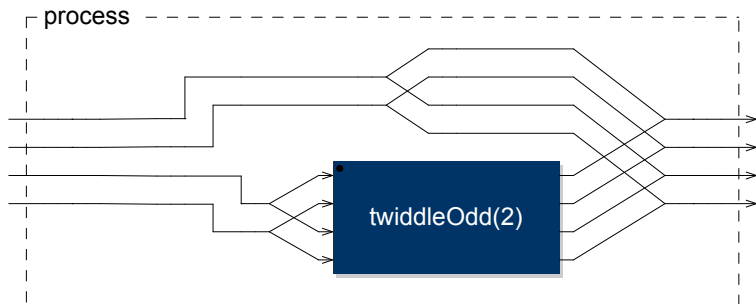
`fft(1)`



Language Expressiveness

Fast Fourier Transform

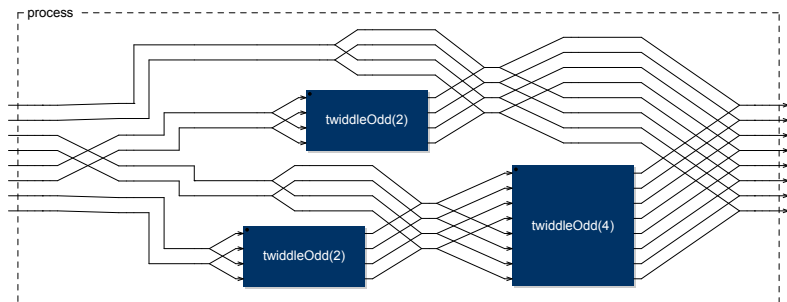
`fft(2)`



Language Expressiveness

Fast Fourier Transform

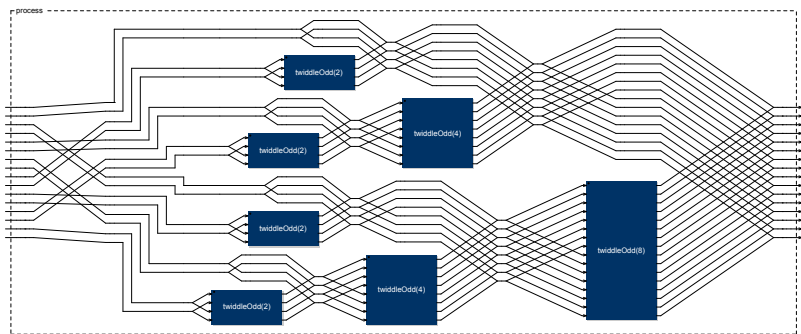
`fft(4)`



Language Expressiveness

Fast Fourier Transform

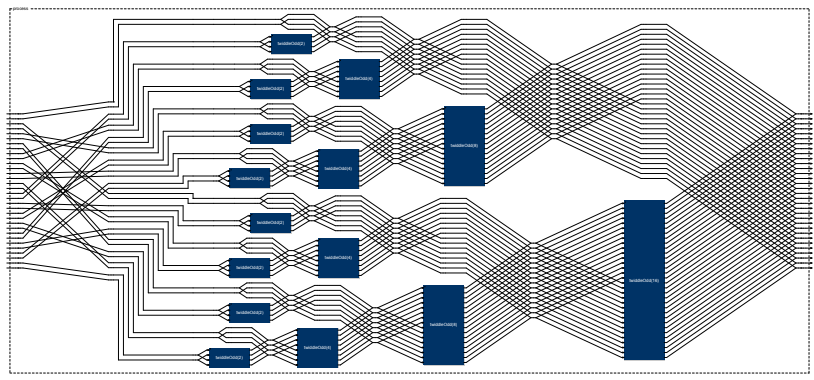
`fft(8)`



Language Expressiveness

Fast Fourier Transform

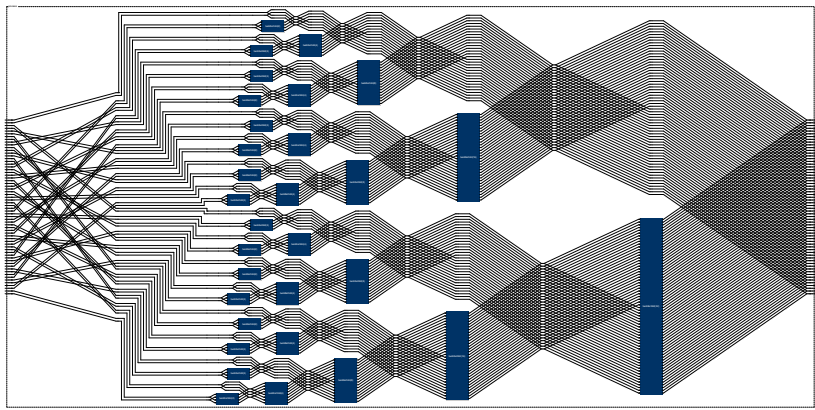
`fft(16)`



Language Expressiveness

Fast Fourier Transform

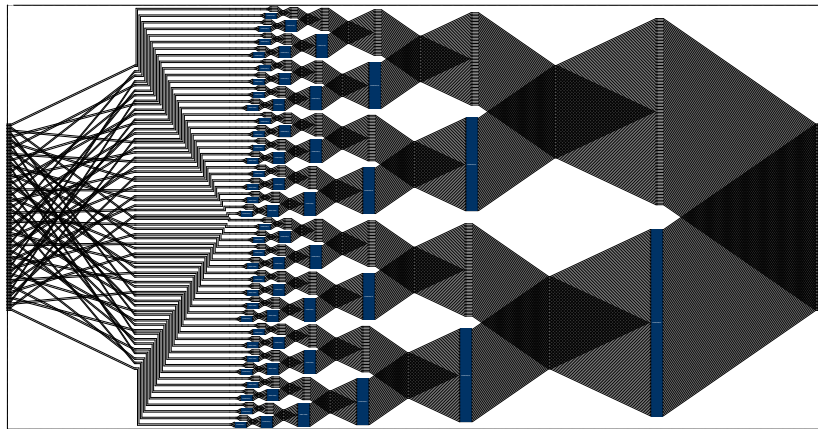
`fft(32)`



Language Expressiveness

Fast Fourier Transform

`fft(64)`



Performance Quest

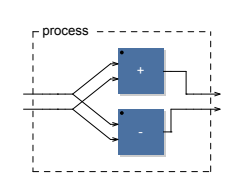
- Fully compiled to native code
- Sample level semantics
- Specification language
- Automatic parallelization

Fully compiled to native code

Faust code:

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

Block-diagram:



C++ translation:

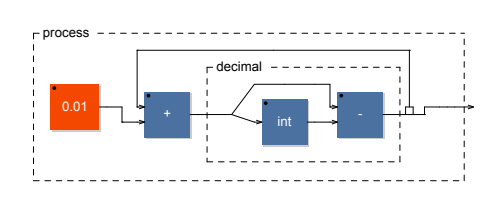
```
for (int i = 0; (i < count); i = (i + 1)) {  
    float fTemp0 = input0[i];  
    float fTemp1 = input1[i];  
    output0[i] = fTemp0 + fTemp1;  
    output1[i] = fTemp0 - fTemp1;  
}
```


Sample level semantics

Sawtooth signal by step of 0.01:

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

Block-diagram:



Signal equation:

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

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

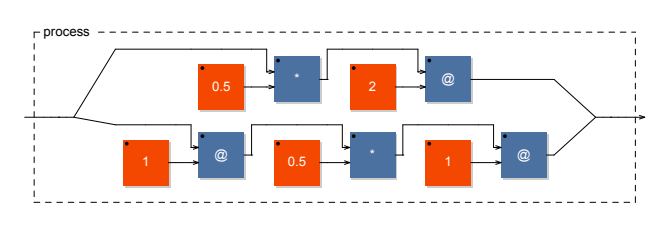
Specification Language

Leave the implementation to the compiler

User's code:

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

Block-diagram:

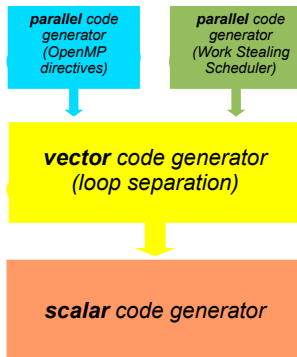


Equivalent, more efficient code

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

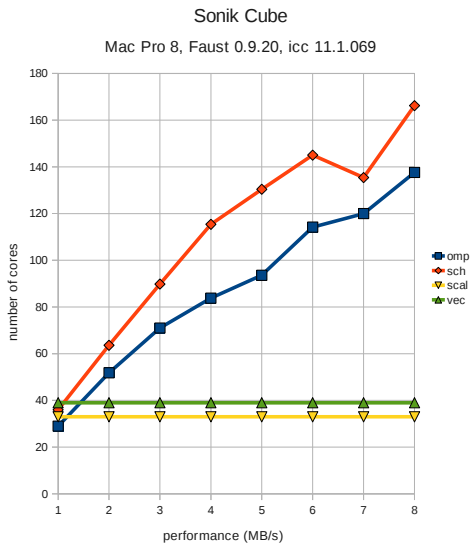
Automatic Parallelization

Code Generators



Automatic Parallelization

Performances



Easy Deployment Quest

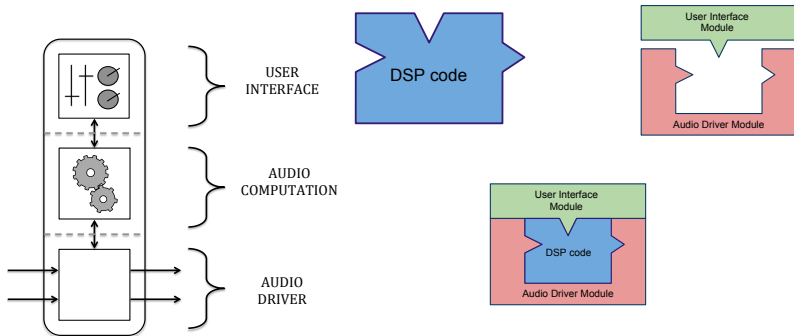
Easy Deployment



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

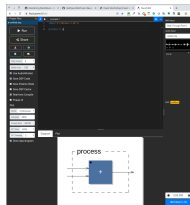
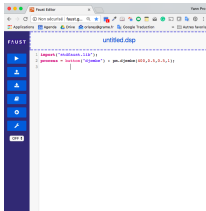
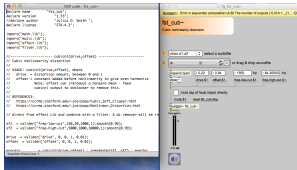
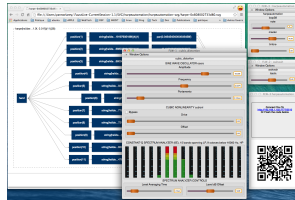
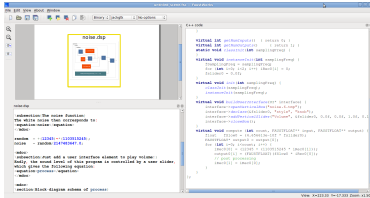
Libfaust

- 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://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

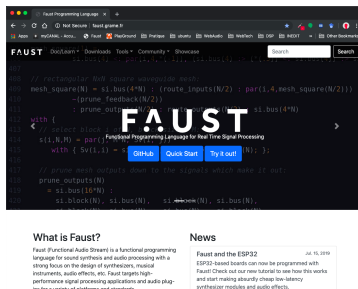
The screenshot shows a web browser displaying the Kadenze website. The URL in the address bar is <https://www.kadenze.com/courses/real-time-audio-signal-processing-in-faust/info>. The website header includes the Kadenze logo, navigation links (Courses, Programs, Membership, Features, Partners, Gallery), and buttons for 'Sign Up' and 'Log In'. Below the header, the course title 'Real-Time Audio Signal Processing in Faust' is displayed above a large, abstract, orange and purple geometric pattern. Below the pattern are four small circular profile pictures of the course instructors. To the right of the main content area, a sidebar titled 'WOULD YOU LIKE TO ENROLL?' contains an 'ENROLL' button and a table of course details.

WOULD YOU LIKE TO ENROLL?	
Length	5 Sessions
Price	Audio (Free) Certificate (incl. w/ Premium)
Institution	Stanford University
Subject	Creative Computing
Skill Level	Expert
Topics	Synthesis, Computer Programming, Digital Signal Processing (DSP), 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 website in a web browser. The page has a dark theme. At the top, there's a navigation bar with links: "FAUST", "Docs/learn", "Downloads", "Tools", "Community", and "Showcase". A search bar is on the right. Below the navigation bar, there's a large code editor displaying Faust code. A large "FAUST" logo is overlaid on the code. Below the code, there are three buttons: "OnHub", "Quick Start", and "Try it out!". Below the code editor, there's a section titled "What is Faust?" with a paragraph of text. To the right of this section is a "News" section with a date "Jul 15, 2019" and a title "Faust and the ESP32".

```
// rectangular 8th square waveguide mesh;
mesh_square(N) = s1.bus(4*N) : (route_inputs(N/2) : par(1,4,mesh_square(N/2)))
    -(prune_feedback(N/2))
    : prune_outputs(N/2) : s1.bus(4*N)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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, 2019
ESP32-based boards can now be programmed with Faust! Check out our new tutorial to see how this works and start making abundantly cheap low-latency synthesizer modules and audio effects.

<https://faust.grame.fr>