INSTITUTE OF SONOLOGY

Royal Conservatoire The Hague

# Programming and Music

*Programming and Music*

*http://www.bjarni-gunnarsson.net*

*Class*

# Course Description

The course covers **programming** fundamentals, **synthesis**, **composition**, and **interaction** approaches. Topics are studied using the ***SuperCollider*** programming environment.

The course starts by going through the basic concepts of *programming* and *computer science* while gradually introducing topics related to **algorithmic composition** and **sound synthesis**.

Finally, interaction processes using **graphical user interfaces** and **external controllers** are studied to create original systems capable of generating music.

# Objectives

At the end of this course, you:

- Know the basics of **programming** in SuperCollider and how to use programming for musical projects

- Have basic knowledge of a**lgorithmic composition** and **programming sounds**

- Can implement **user interfaces** and use **external controllers** for musical applications

# Assessment Criteria

- Computer **programming** basics
- Ability to read and write **computer code**
- Clarity in implementing **technical solutions**
- Knowledge of **computer music** fundamentals

# Prerequisites

An **interest** in learning **programming** and **computer music**

Having a **computer** capable of running **SuperCollider** or be willing to come to the **Sonology** studios to use it.

Some experience with computers and digital audio is useful but not required.

# Course Format

A class will usually focus on a specific **topic** or collection of topics.

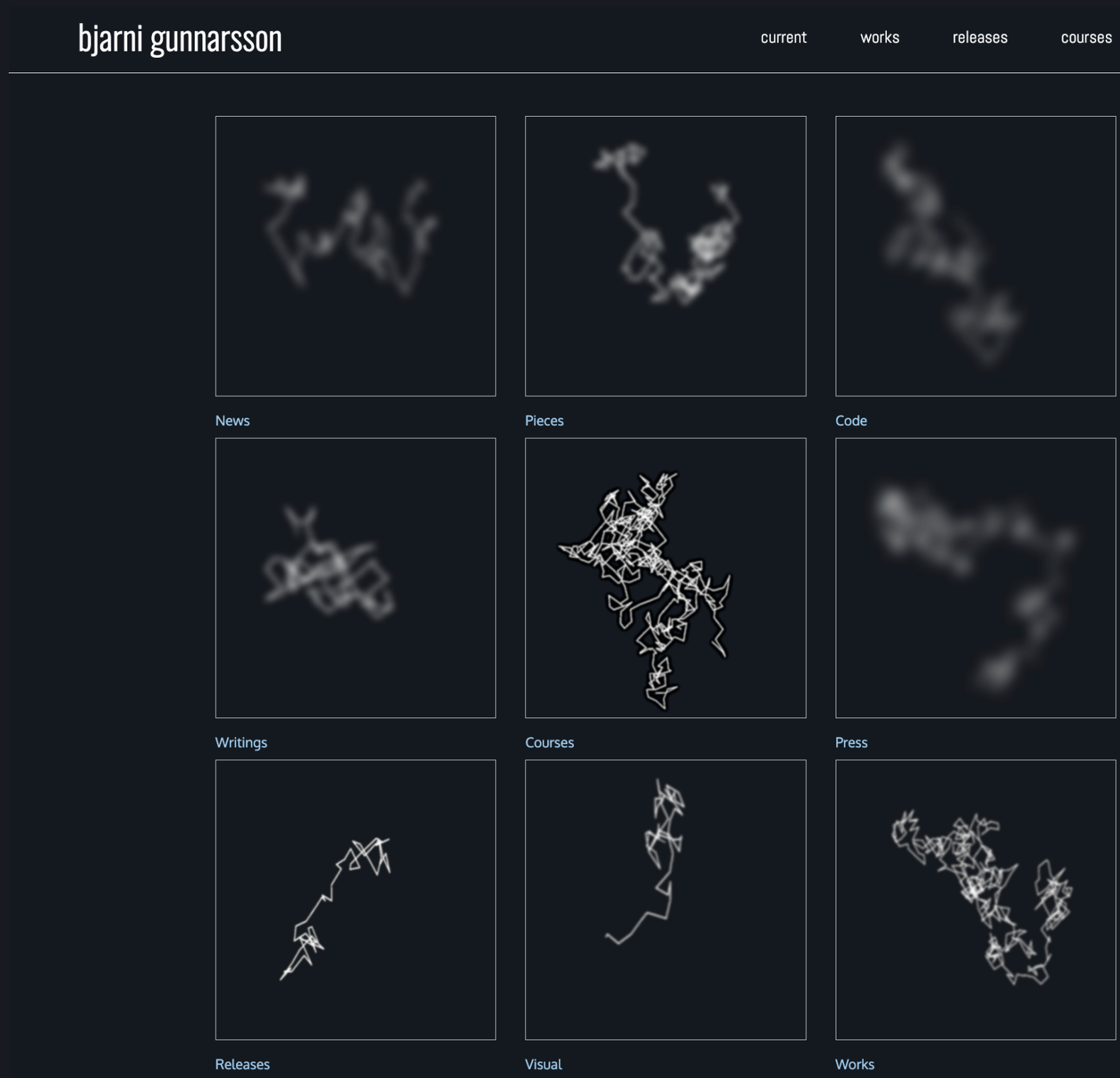
Part of the lesson will be a **presentation** of, and **discussion** on, the *topic* in question where a lecture will be given and slides will be presented.

The other part of the lesson will focus on hands-on **experimentation** in *SuperCollider* or the other featured environments.

Discussion and interaction should take place as much as possible.

*If you can, bring your laptops to classes.*

# Materials




*/courses . code == ‘programming’*

# Assessment

**Three** practical **assignments** must be handed in.

The assignments involve writing **computer programs** for different problems related to *music* and *sound*.

**Documentation** must be included explaining the chosen solutions and their motivations.

The assignments each value **30%** of the final grade.

**Attendance** counts for the remaining **10%**.

*All assignments will have to be **completed** in order to pass this course.*

# Topics

**- (01) Programming -**

01 - Introduction
02 - Syntax
03 - Control Flow
04 - Functions
05 - Data Structures
06 - Workshop 1
07 - Approaches
08 - Review (Assignment 1)

***Assignment 1 (30 %)***

# Topics

**- (02) Music and Sound -**

09 - Routines
10 - Clocks
11 - Signals
12 - Workshop 2
13 - Synthesis 1
14 - Synthesis 2
15 - Patterns 1
16 - Patterns 2

***Assignment 2 (30 %)***

# Topics

**- (03) Control and Processing -**

17 - GUI
18 - MIDI & Mapping
19 - Workshop 3
20 - Effects
21 - Buffers
22 - Workshop 6
23 - Examples 1
24 - Examples 2

***Assignment 3 (30 %)***

SuperCollider

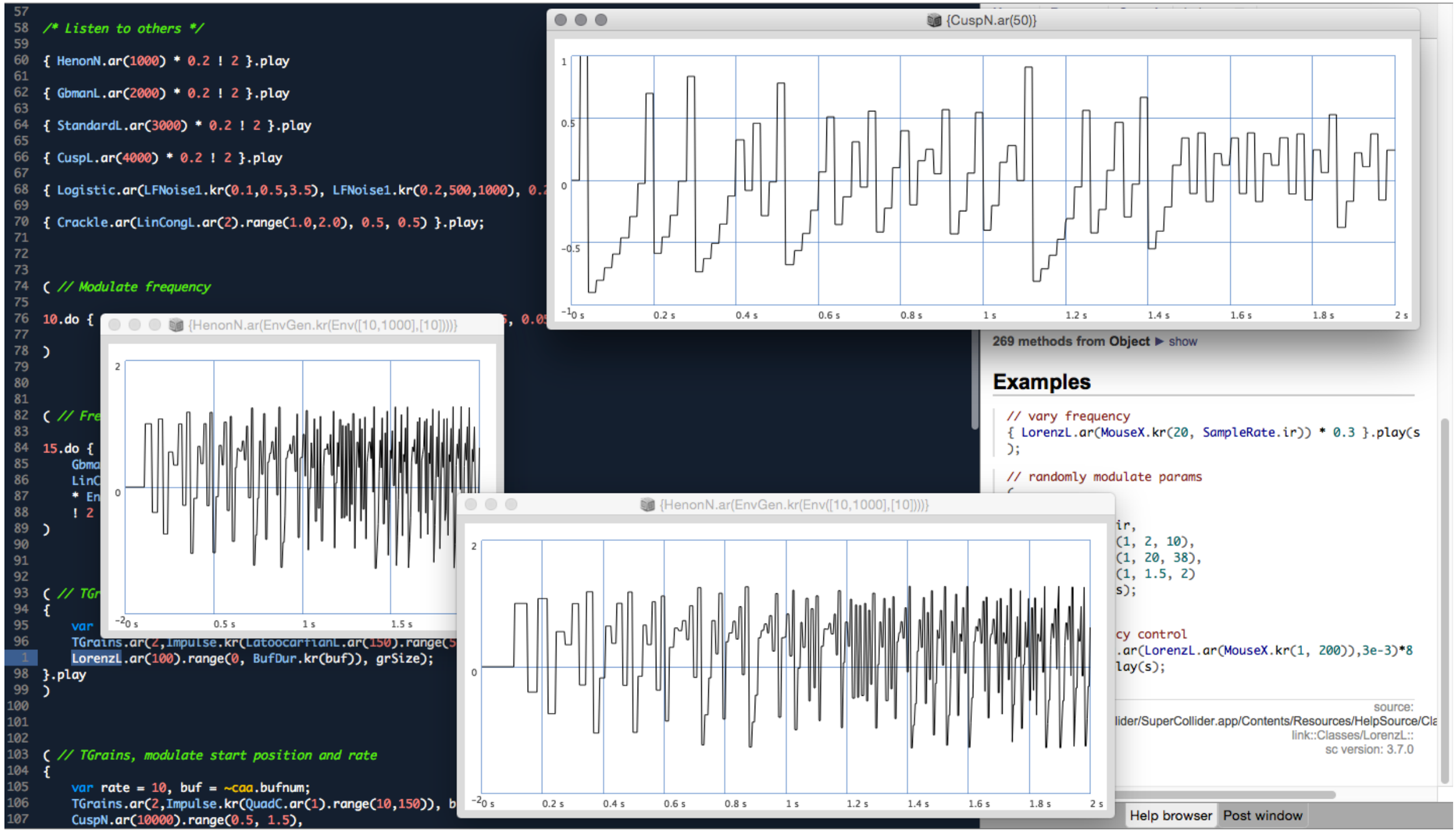
# SuperCollider

**SuperCollider** is an environment for *real time audio* and *composition*.

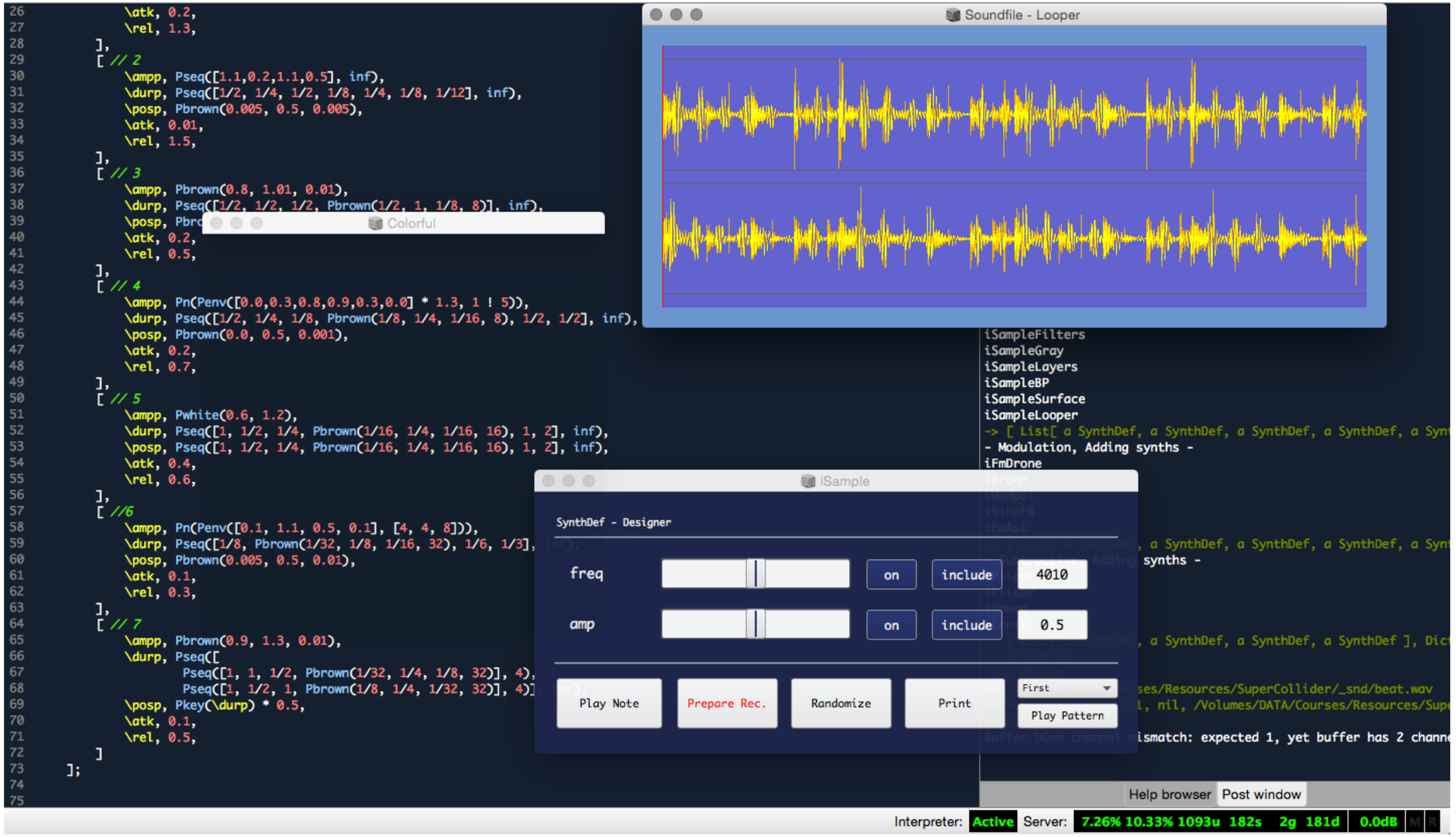
SuperCollider consists of an interpreted **object-oriented language** and a state of the art **realtime sound synthesis server**.

SuperCollider supports different activities such as sound synthesis, digital signal processing, algorithmic composition, live electronics and live coding.

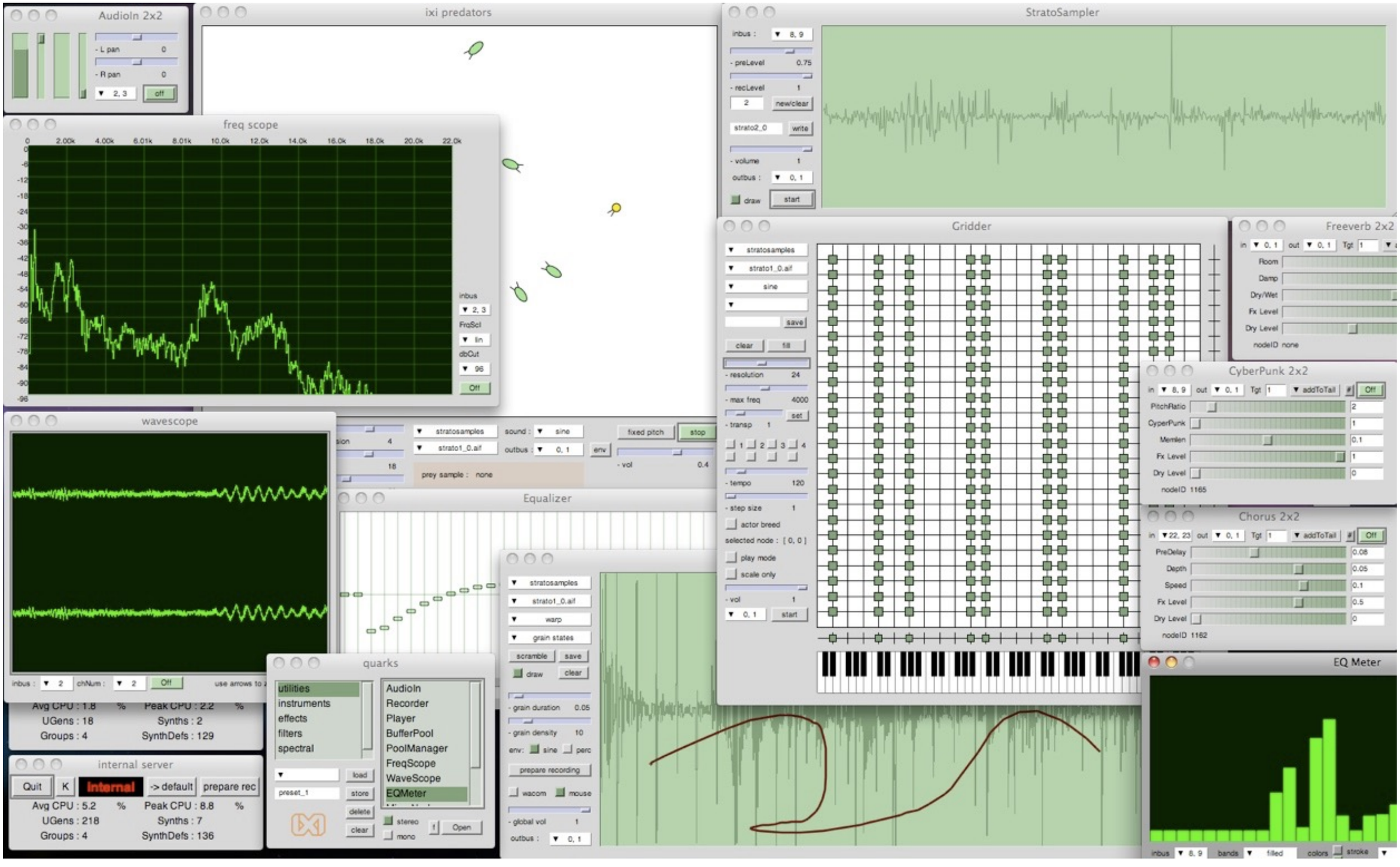
# SuperCollider



# SuperCollider



# SuperCollider




IXI Quarks

# SuperCollider

News Download Wiki Docs Star 2,107 Fork 397

SuperCollider is a platform for audio synthesis and algorithmic composition, used by musicians, artists, and researchers working with sound. It is free and open source software available for Windows, macOS, and Linux.

SuperCollider features three major components:

- **scsynth**, a real-time audio server, forms the core of the platform. It features 400+ unit generators ("UGens") for analysis, synthesis, and processing. Its granularity allows the fluid combination of many known and unknown audio techniques, moving between additive and subtractive synthesis, FM, granular synthesis, FFT, and physical modeling. You can write your own UGens in C++, and users have already contributed several hundred more to the **sc3-plugins** repository.
- **sclang**, an interpreted programming language. It is focused on sound, but not limited to any specific domain. sclang controls scsynth via Open Sound Control. You can use it for algorithmic composition and sequencing, finding new sound synthesis methods, connecting your app to external hardware including MIDI controllers, network music, writing GUIs and visual displays, or for your daily programming experiments. It has a stock of user-contributed extensions called Quarks.
- **scide** is an editor for sclang with an integrated help system.

SuperCollider was developed by James McCartney and originally released in 1996. In 2002, he generously released it as free software under the GNU General Public License. It is now maintained and developed by an active and enthusiastic community.

## Examples

- **Code examples**

```
// modulate a sine frequency and a noise amplitude with another sine
// whose frequency depends on the horizontal mouse pointer position
{
        var x = SinOsc.ar(MouseX.kr(1, 100));
        SinOsc.ar(300 * x + 800, 0, 0.1)
        +
        PinkNoise.ar(0.1 * x + 0.1)
}.play;
```

*http://supercollider.github.io*

# Design Goals

To realize sound processes that are **different** every time they are played.

To write pieces in a way that **describes a range of possibilities** rather than a fixed entity

To facilitate **live improvisation** by a composer/performer.

*(McCartney, Rethinking the Computer Music Language: SuperCollider)*

# About

The SuperCollider **language** is based on *Smalltalk* and is used for creating **programs** that communicate with the **synthesis server** in order to make sounds.

**Unit Generators** (UGens) are used for generating and processing audio signals within the synthesis server. Interconnected *UGens* are packaged into a **SynthDef** that describes which UGens are used and how they connect.

The **objects** of the SuperCollider language objects together data and methods that act on that data.

# About

**Classes** describe *attributes* and *behavior* that objects have in common.

Sounds that are played or transformed are stored in **buffers** that exist inside the synthesis server.

Audio channels that SuperCollider synths use for sending their sound through are called audio **buses**.

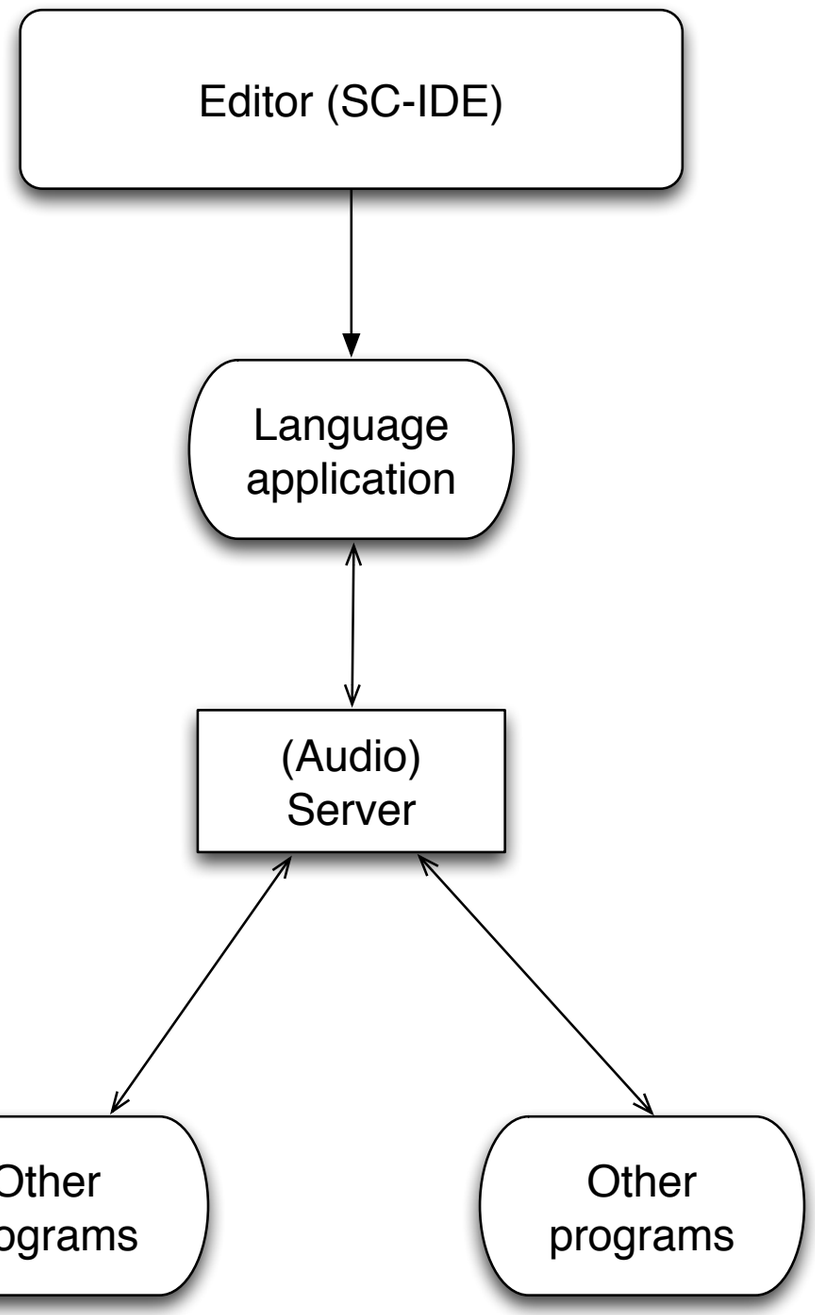
*Compositional logic* and scheduling is implemented with **routines**, **tasks** and **patterns**.

# *Architecture*

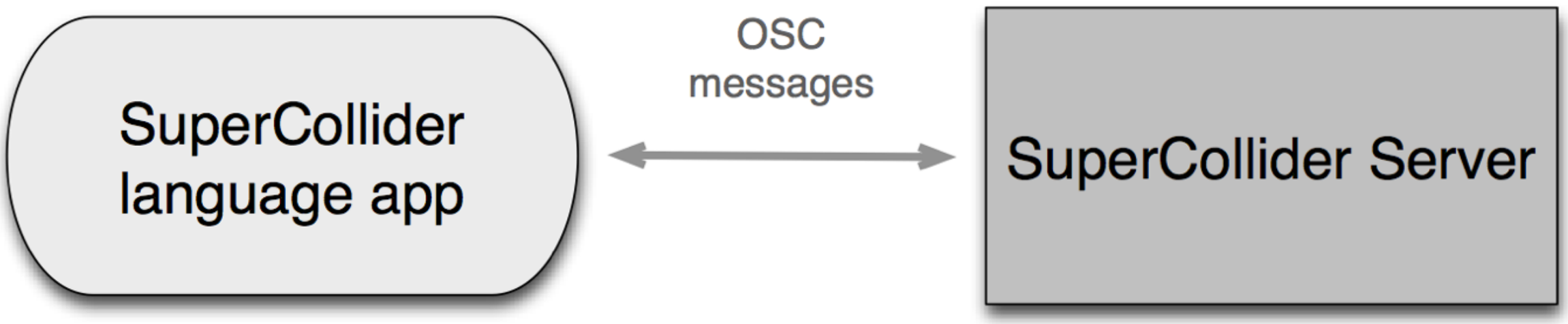
SuperCollider

supercollider.sourceforge.net

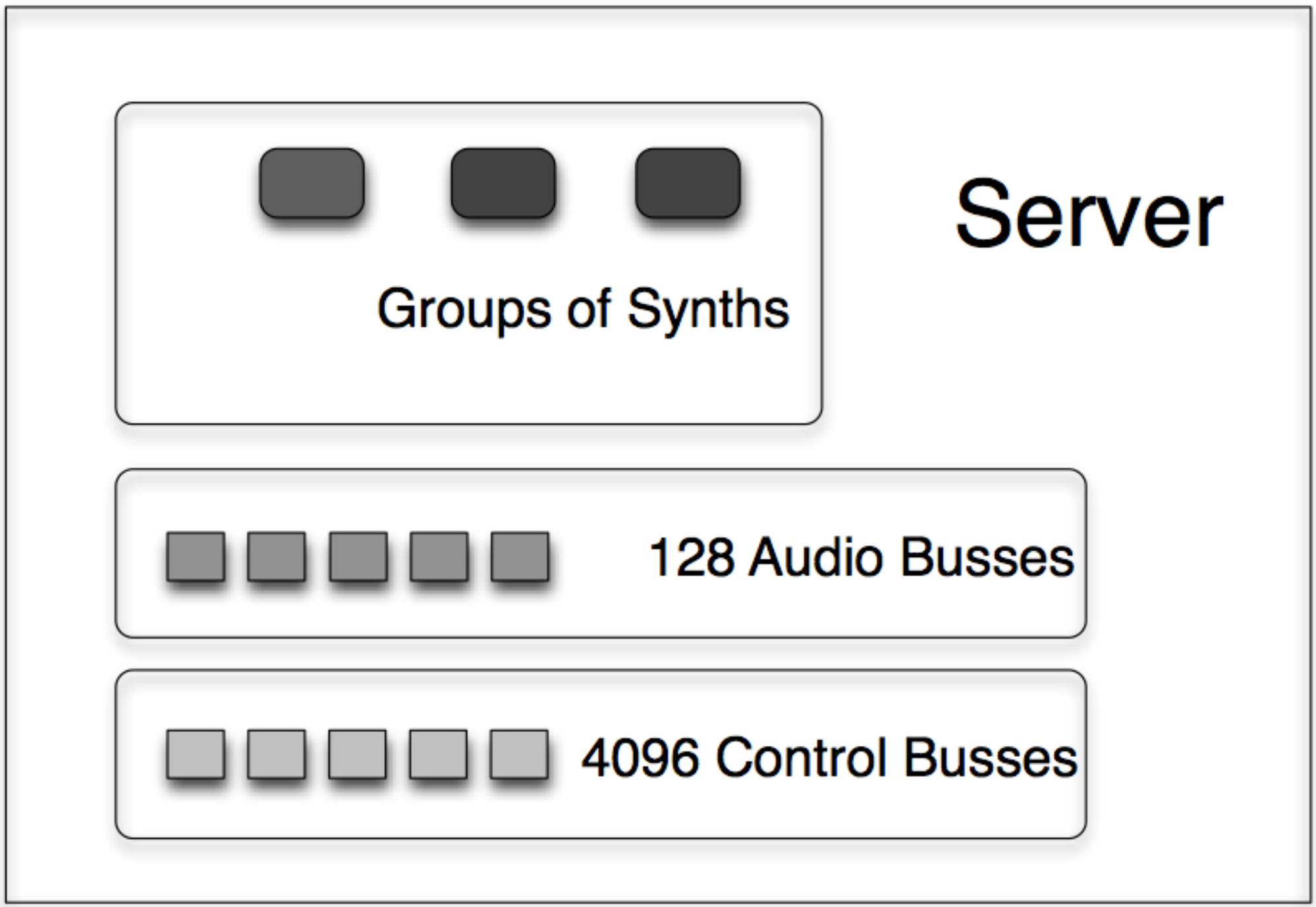
Version 3.6



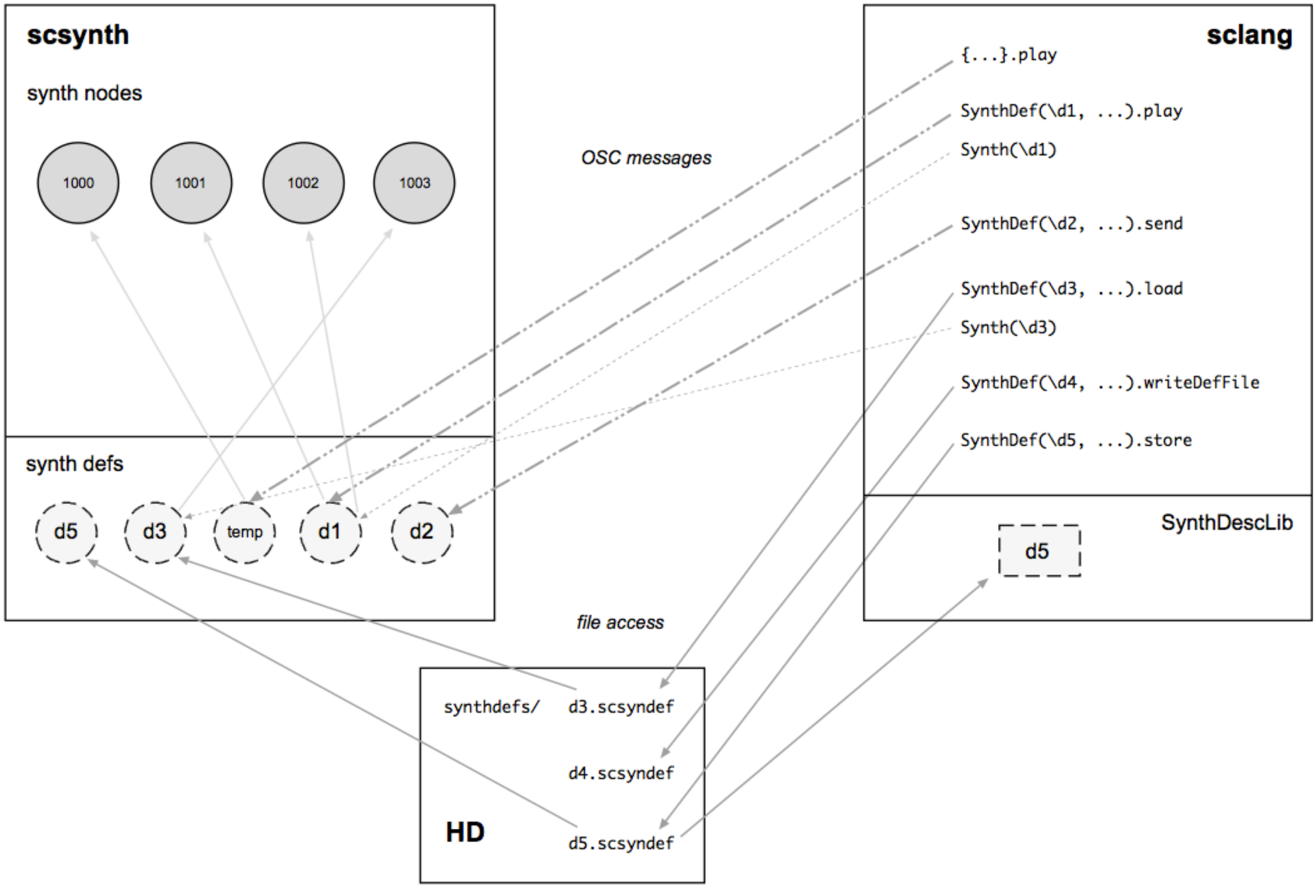
# *Messaging*


SuperCollider
language app
OSC
messages
SuperCollider Server

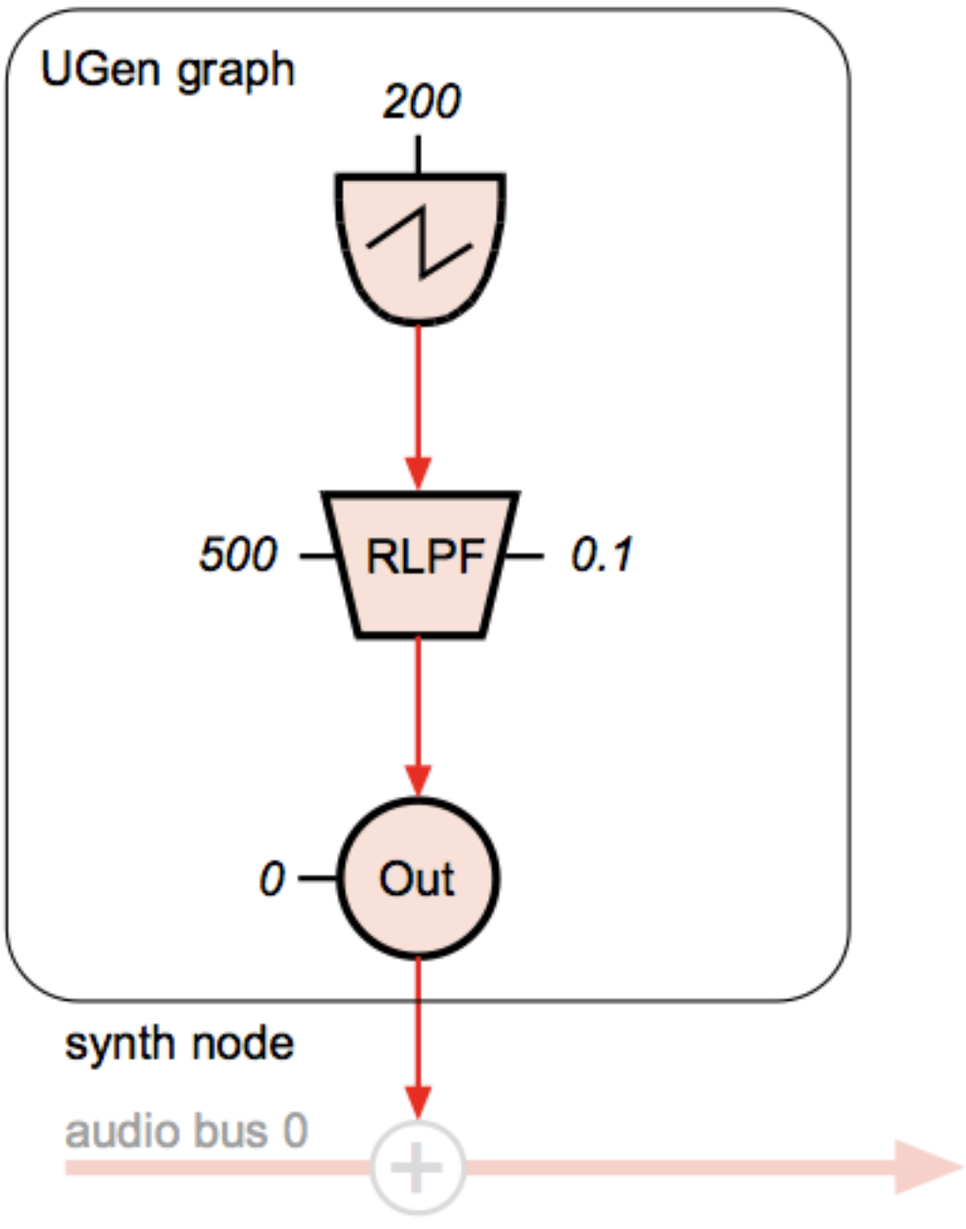
# *Architecture*



# Architecture



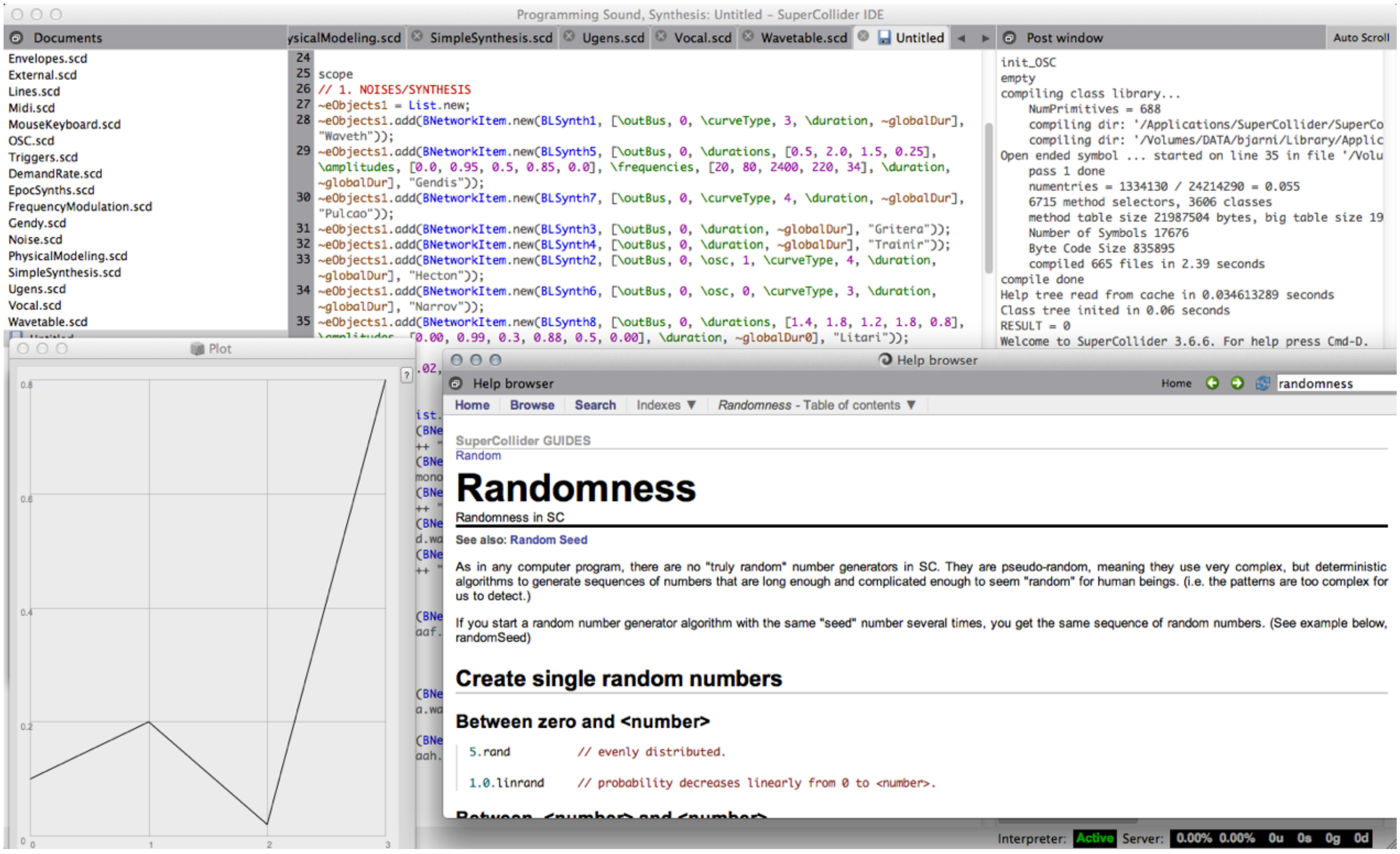
# *Synthesis Graphs*



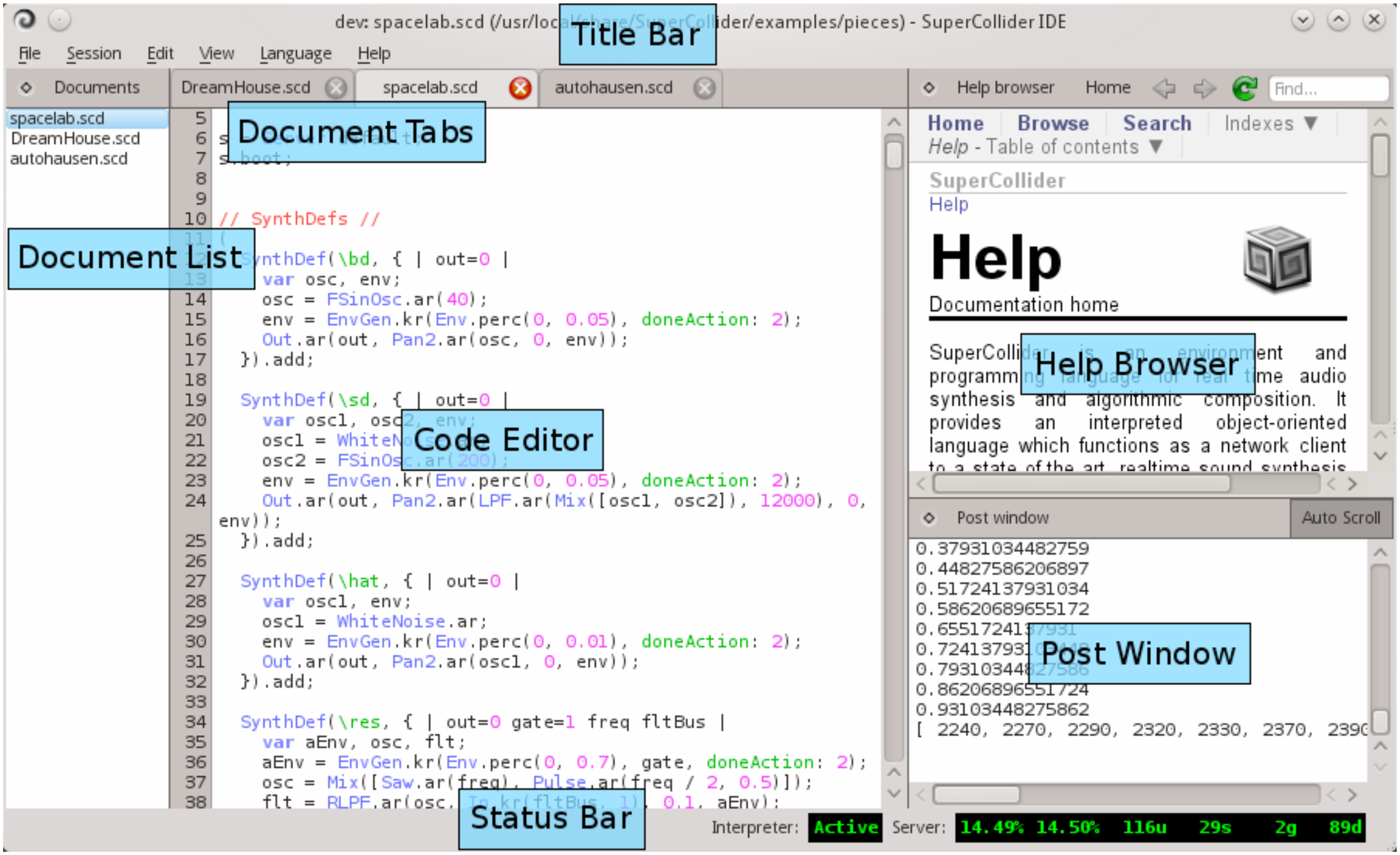

**Synth Nodes and UGen graphs**

```
(
SynthDef("simple", {
var sig;
sig = Saw.ar(200);
sig = RLPF.ar(sig, 500, 0.1);
Out.ar(0, sig);
}).play;
)
```

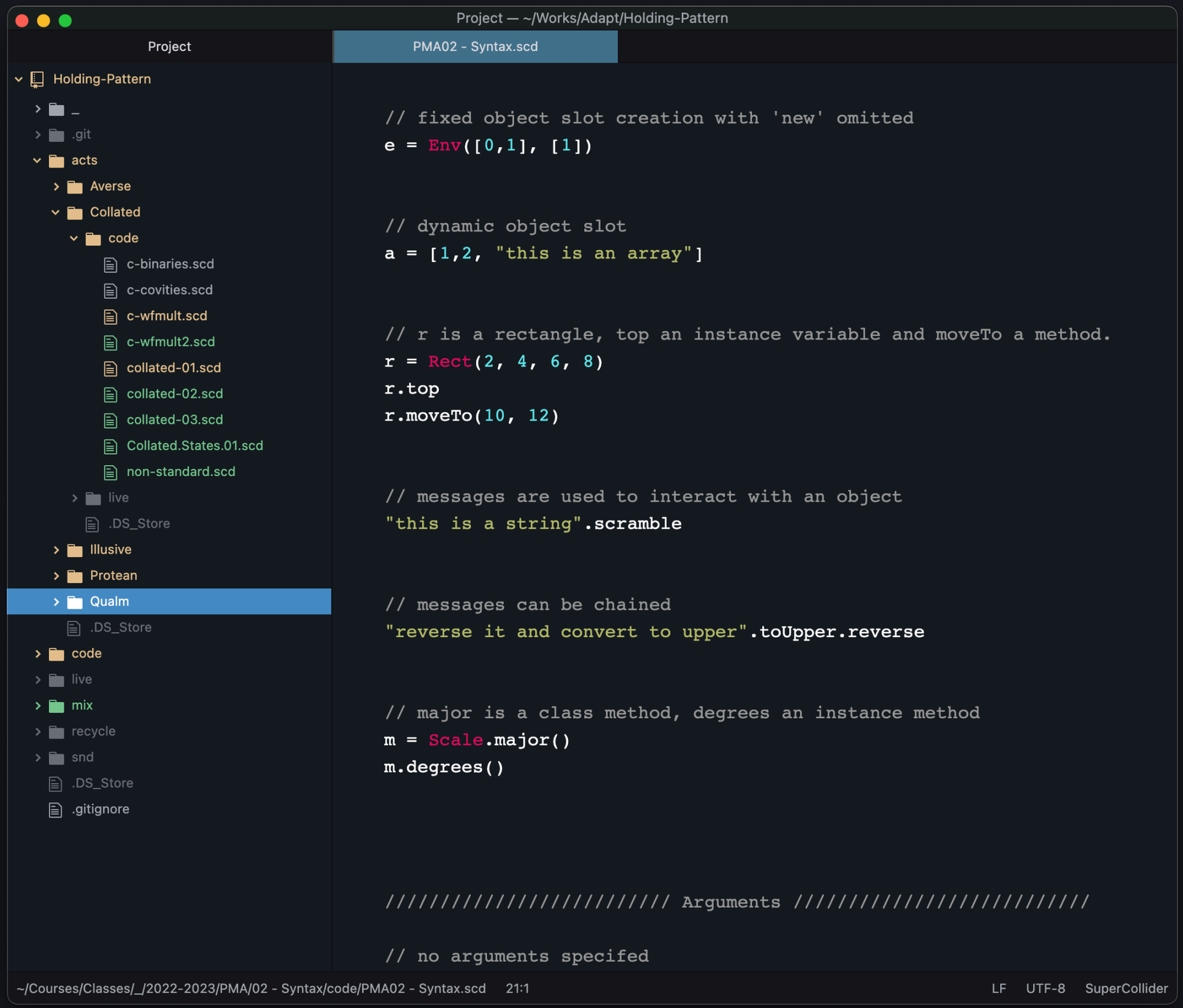
# IDE


Programming Sound, Synthesis: Untitled – SuperCollider IDE
Documents
Envelopes.scd
External.scd
Lines.scd
Midi.scd
MouseKeyboard.scd
OSC.scd
Triggers.scd
DemandRate.scd
EpocSynths.scd
FrequencyModulation.scd
Gendy.scd
Noise.scd
PhysicalModeling.scd
SimpleSynthesis.scd
Ugens.scd
Vocal.scd
Wavetable.scd
ysicalModeling.scd
SimpleSynthesis.scd
Ugens.scd
Vocal.scd
Wavetable.scd
Untitled
24
25 scope
26 // 1. NOISES/SYNTHESIS
27 ~eObjects1 = List.new;
28 ~eObjects1.add(BNetworkItem.new(BLSynth1, [\outBus, 0, \curveType, 3, \duration, ~globalDur], "Waveth"));
29 ~eObjects1.add(BNetworkItem.new(BLSynth5, [\outBus, 0, \durations, [0.5, 2.0, 1.5, 0.25], \amplitudes, [0.0, 0.95, 0.5, 0.85, 0.0], \frequencies, [20, 80, 2400, 220, 34], \duration, ~globalDur], "Gendis"));
30 ~eObjects1.add(BNetworkItem.new(BLSynth7, [\outBus, 0, \curveType, 4, \duration, ~globalDur], "Pulcao"));
31 ~eObjects1.add(BNetworkItem.new(BLSynth3, [\outBus, 0, \duration, ~globalDur], "Gritera"));
32 ~eObjects1.add(BNetworkItem.new(BLSynth4, [\outBus, 0, \duration, ~globalDur], "Trainir"));
33 ~eObjects1.add(BNetworkItem.new(BLSynth2, [\outBus, 0, \osc, 1, \curveType, 4, \duration, ~globalDur], "Hecton"));
34 ~eObjects1.add(BNetworkItem.new(BLSynth6, [\outBus, 0, \osc, 0, \curveType, 3, \duration, ~globalDur], "Narrov"));
35 ~eObjects1.add(BNetworkItem.new(BLSynth8, [\outBus, 0, \durations, [1.4, 1.8, 1.2, 1.8, 0.8],
[0.00, 0.99, 0.3, 0.88, 0.5, 0.00], \duration, ~globalDur0], "Litari"));
Post window
Auto Scroll
init_OSC
empty
compiling class library...
NumPrimitives = 688
compiling dir: '/Applications/SuperCollider/SuperCo
compiling dir: '/Volumes/DATA/bjarni/Library/Applic
Open ended symbol ... started on line 35 in file '/Volu
pass 1 done
numentries = 1334130 / 24214290 = 0.055
6715 method selectors, 3606 classes
method table size 21987504 bytes, big table size 19
Number of Symbols 17676
Byte Code Size 835895
compiled 665 files in 2.39 seconds
compile done
Help tree read from cache in 0.034613289 seconds
Class tree inited in 0.06 seconds
RESULT = 0
Welcome to SuperCollider 3.6.6. For help press Cmd-D.
Plot
0.8
0.6
0.4
0.2
0
0
1
2
3
Help browser
Help browser
Home
randomness
Home
Browse
Search
Indexes ▼
Randomness - Table of contents ▼
SuperCollider GUIDES
Random
Randomness
Randomness in SC
See also: Random Seed
As in any computer program, there are no "truly random" number generators in SC. They are pseudo-random, meaning they use very complex, but deterministic algorithms to generate sequences of numbers that are long enough and complicated enough to seem "random" for human beings. (i.e. the patterns are too complex for us to detect.)
If you start a random number generator algorithm with the same "seed" number several times, you get the same sequence of random numbers. (See example below, randomSeed)
Create single random numbers
Between zero and <number>
5.rand // evenly distributed.
1.0.linrand // probability decreases linearly from 0 to <number>.
Interpreter: Active Server: 0.00% 0.00% 0u 0s 0g 0d

# IDE


Title Bar
Document Tabs
Document List
Code Editor
Help Browser
Post Window
Status Bar
dev: spacelab.scd (/usr/local/share/SuperCollider/examples/pieces) - SuperCollider IDE
File Session Edit View Language Help
Documents
DreamHouse.scd
spacelab.scd
autohausen.scd
Help browser
Home
Find...
Home Browse Search Indexes
Help - Table of contents
SuperCollider
Help
Help
Documentation home
SuperCollider is an 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
Post window
Auto Scroll
0.37931034482759
0.44827586206897
0.51724137931034
0.58620689655172
0.86206896551724
0.93103448275862
[ 2240, 2270, 2290, 2320, 2330, 2370,
Interpreter: Active
Server: 14.49% 14.50% 116u 29s 2g 89d

# Pulsar


Project — ~/Works/Adapt/Holding-Pattern
Project
PMA02 - Syntax.scd
Holding-Pattern
_
.git
acts
Averse
Collated
code
c-binaries.scd
c-covities.scd
c-wfmult.scd
c-wfmult2.scd
collated-01.scd
collated-02.scd
collated-03.scd
Collated.States.01.scd
non-standard.scd
live
.DS_Store
Illusive
Protean
Qualm
.DS_Store
code
live
mix
recycle
snd
.DS_Store
.gitignore
// fixed object slot creation with 'new' omitted
e = Env([0,1], [1])
// dynamic object slot
a = [1,2, "this is an array"]
// r is a rectangle, top an instance variable and moveTo a method.
r = Rect(2, 4, 6, 8)
r.top
r.moveTo(10, 12)
// messages are used to interact with an object
"this is a string".scramble
// messages can be chained
"reverse it and convert to upper".toUpper.reverse
// major is a class method, degrees an instance method
m = Scale.major()
m.degrees()
////////////////////////// Arguments //////////////////////////
// no arguments specifed
~/Courses/Classes/_/2022-2023/PMA/02 - Syntax/code/PMA02 - Syntax.scd 21:1
LF UTF-8 SuperCollider

# Resources

**Supercollider home page**

*https://supercollider.github.io/*

**Original Supercollider home page**

*http://www.audiosynth.com*

**Code examples**

*http://sccode.org*

**Forum**

*https://scsynth.org*

**The Supercollider book**

*https://mitpress.mit.edu/books/supercollider-book*

**Eli Fieldsteel's video tutorials**

*https://www.youtube.com/user/elifieldsteel*

**Reflectives**

*https://www.youtube.com/channel/UCypLRZiSllQjsT_7J4Vz35Q*

# Resources

**A Gentle Introduction to SuperCollider - CCRMA**

*https://ccrma.stanford.edu/~ruviaro/texts/A_Gentle_Introduction_To_SuperCollider.pdf*

**Mapping and visualization with SuperCollider**

*http://marinoskoutsomichalis.com/mapping-and-visualization*

**Nick Collins tutorial**

*https://composerprogrammer.com/teaching/supercollider/sctutorial/tutorial.html*

**Thor Magnússon tutorial**

*http://www.ixi-software.net/content/body_backyard_tutorials.html*

**Stelios Manousakis course**

*http://modularbrains.net/portfolio/supercollider-real-time-interactive-course-sc-code/*

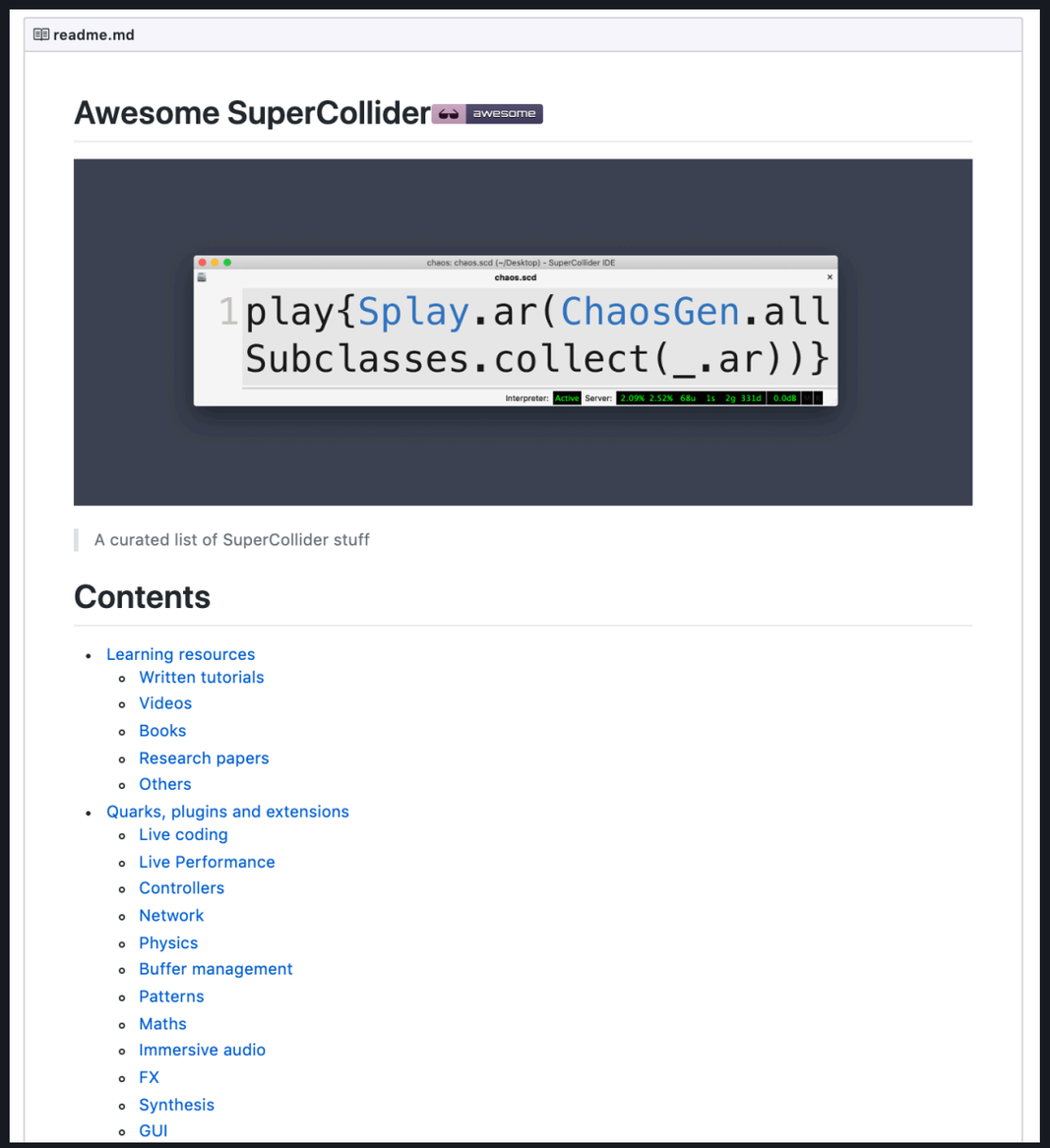
**Fredrik Olofsson tutorials**

*http://www.fredrikolofsson.com/pages/code-sc.html*

# Resources

## Awesome SuperCollider

*https://github.com/madskjeldgaard/awesome-supercollider*



# Programming

```
rrand(3, 20)

[60, 62, 64, 65, 67, 69, 71, 72].midicps

f = {arg a, b; rrand(a, b) }

Array.series (10,1,100)
```

```
10.do({arg i; i.postln})

if ( (0.5.coin), { {SinOsc.ar(500,0,0.5)}.play },
{ {SinOsc.ar(1000,0,0.5)}.play} )
```

# Synthesis

```
{ Saw.ar(36.midicps, 0.3) }.play

{ SinOsc.ar(440, 0, LFNoise2.kr(5).exprange(0.01,
0.9)) }.play

{ RLPF.ar(Saw.ar([100, 101], 0.5),
LFNoise1.kr(1).range(300, 1200), 0.1) }.play

20.do { arg i;  { SinOsc.ar(exprand(200.0, 1800.0), 0,
LFNoise2.kr(rrand(0.3, 1.0)).exprange(0.001,0.2)) }.play }
```

# Patterns

```
Pwhite(40,80,10).asStream;

Pbind(\instrument, \noise,
    \freq,Pseq([200,400,800], 2),
    \dur,0.2).play

Pbind(\instrument, \sine,
    \freq, Pwhite(200,1000),
    \dur, Pwhite(0.1,0.2),
    \amp, 0.5,
).play
```