

Interactive Music II

SuperCollider入門

2014年10月2日

東京藝術大学芸術情報センター(AMC)

田所 淳

Interactive Music II について

Interactive Music II について

- ▶ Interactive Music I (松村先生) では、Pure Dataを開発言語として、インタラクティブな音楽を創作
- ▶ Interactive Music II では、この講義を引き継いで、また別の角度からプログラミング言語を用いた、インタラクティブな音楽の創作を探求したい
- ▶ ビジュアルプログラミング言語(Pd, Maxなど)ではなく、一般的なテキストベースのプログラミング言語による音楽の創作
- ▶ 今期は、SuperColliderを取り上げたい

Interactive Music II について

- ▶ この講義、もう1つの目標：
- ▶ ライブイベントの開催したい!!
- ▶ いまのところ、ノーアイデア
- ▶ ご意見募集

Interactive Music II について

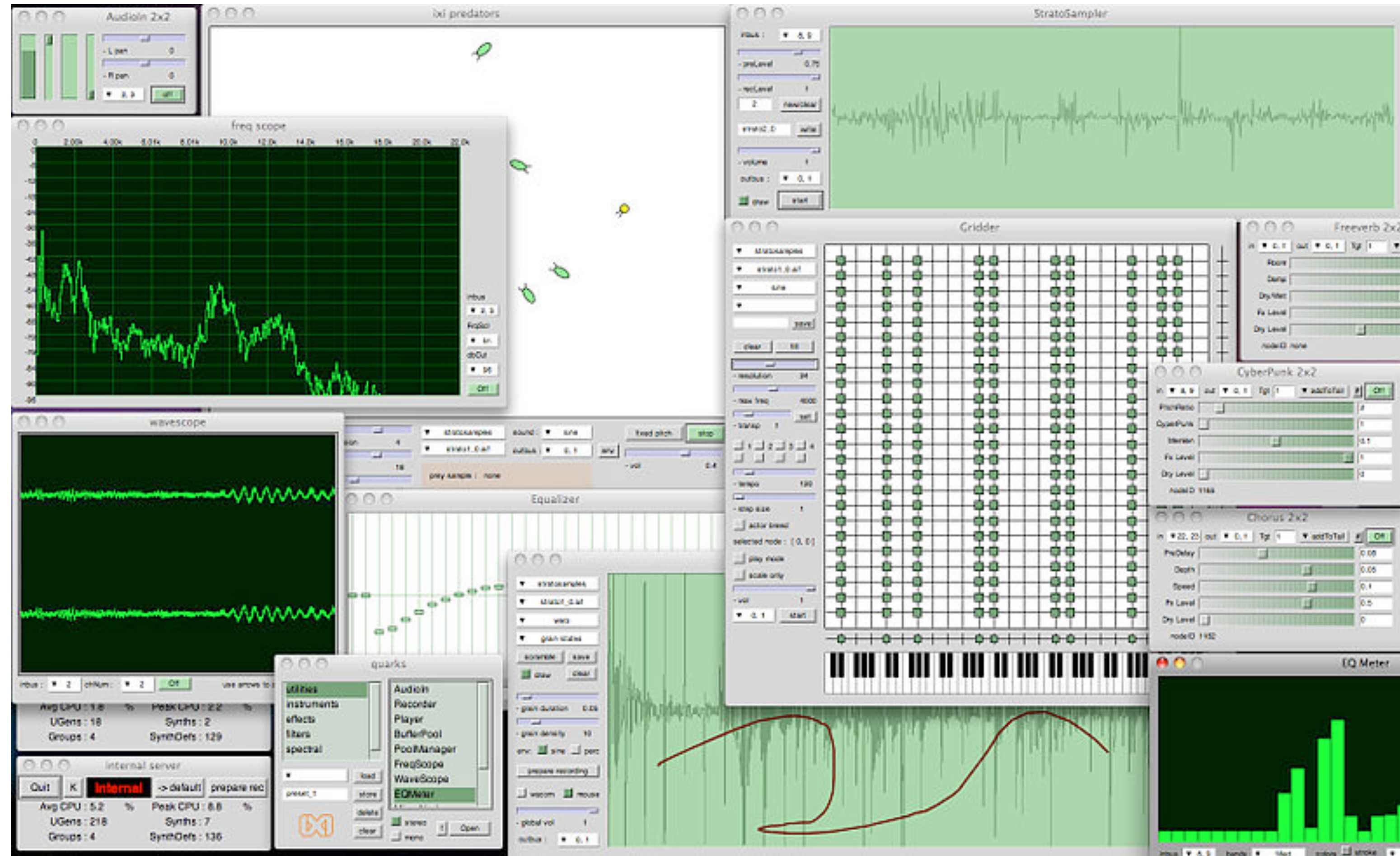
- ▶ 次週以降、毎回持ってきて欲しいもの
- ▶ ヘッドホン (or イヤホン)
- ▶ もし今から買うのであれば、モニター用のもの



SuperCollider Basics

SuperCollider Basics

► SuperColliderとは?



SuperCollider Basics

- ▶ SuperColliderとは?
- ▶ リアルタイムな音響合成やアルゴリズムック・コンポジションのためのプログラミング言語
- ▶ SmallTalkライクなオブジェクト指向言語
- ▶ リアルタイムに音響を生成できる → ライブコーディング!

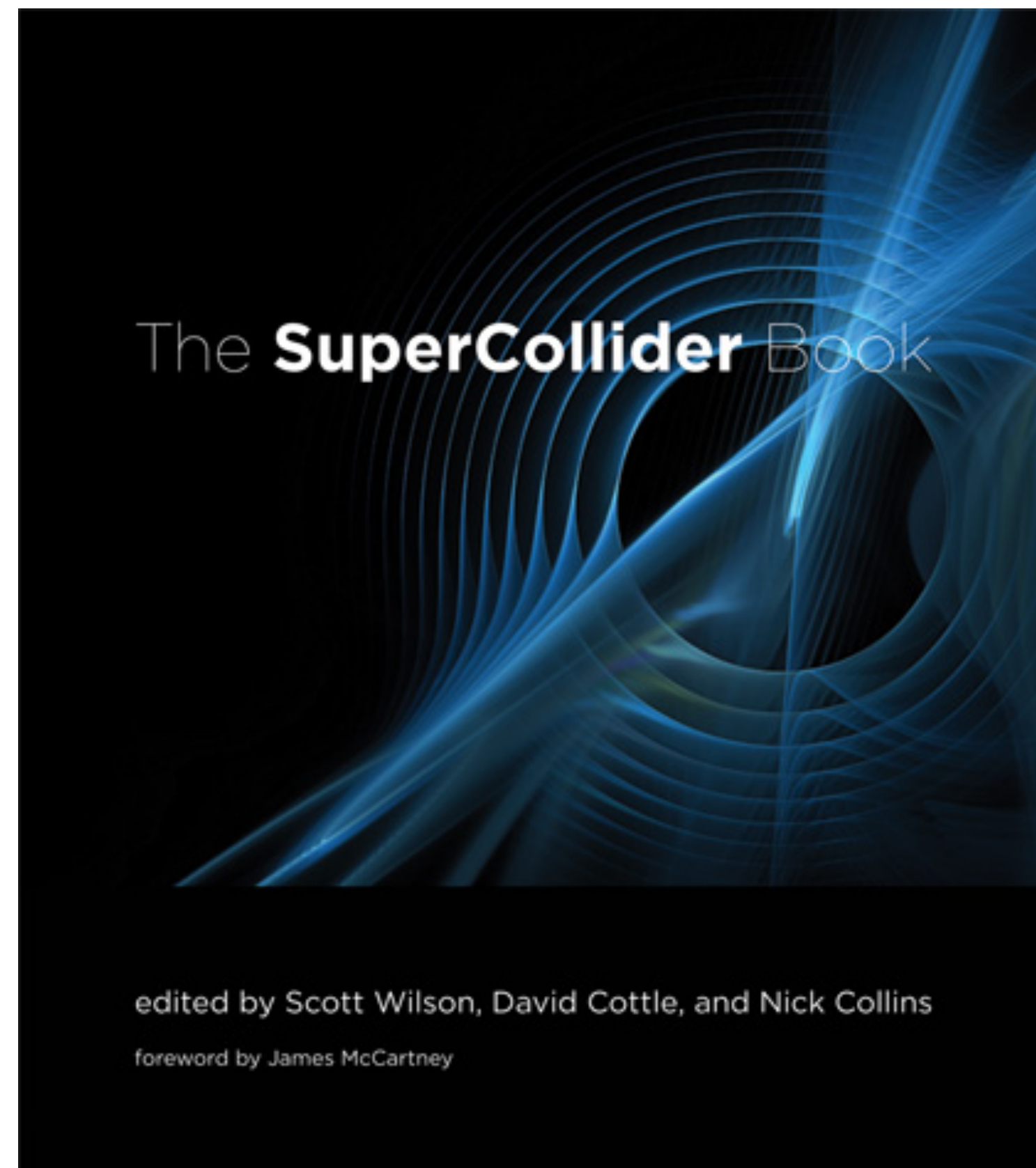
SuperCollider Basics

- ▶ 1996年にJames McCartneyによりリリース
- ▶ James McCartneyの就職(Apple!)などの事情で、オープンソースとして公開
- ▶ 現在は、GPLライセンスとして多くの開発者により更新されている



SuperCollider Basics

- ▶ 参考図書：The SuperCollider Book
- ▶ Wilson, S., Cottle, D. and Collins, N. (eds). 2011. The SuperCollider Book. Cambridge, MA: MIT Press



SuperCollider Basics

- Website: <http://supercollider.sourceforge.net/>

SuperCollider

real-time audio synthesis and algorithmic composition

Pages

[About](#)
[Downloads](#)
[Audio/code examples](#)
[Videos](#)
[Learning](#)
[Community](#)
[Meetings](#)
[Developers](#)
[sc140](#)
[Symposium](#)

Resources

[Documentation](#)
[Plugins](#)
[Quarks](#)
[Wiki](#)

SC elsewhere

[sc@facebook](#)
[sc@flickr](#)
[sc@sourceforge](#)
[sc@twitter](#)
[sc@vimeo](#)
[sc@youtube](#)

About



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 to a **state of the art, realtime sound synthesis server**.

SuperCollider was written by [James McCartney](#) over a period of many years, and is now an open source (GPL) project maintained and developed by various people. It is used by **musicians, scientists, and artists** working with sound. For some background, see [SuperCollider described by Wikipedia](#).

[Screenshots](#) | [Audio/code examples](#) | [Video examples](#)

Downloads

The latest stable version is **3.5.5** (September 2012).

- [Download SuperCollider for Mac, Linux, or Windows](#).

Also download [plugins and extensions](#) to enhance SuperCollider.

The SuperCollider Book

The official SuperCollider Book is now [available from MIT Press](#). Edited by Scott Wilson, David Cottle and Nick Collins, it includes contributions on all aspects of SuperCollider, from beginners' tutorials through to advanced sound synthesis and composition techniques.

"This book documents the SuperCollider language to an extent never before achieved and shows how it can be used to realize a wide variety of musical and technical applications. The scholarship is sound, as the chapter authors are leaders in the field and deeply

Recent news

Unleash & Collide: SuperCollider IDE preview party

SuperCollider workshop 3rd Sep, La Paz, Bolivia

SuperCollider 3.5.3 released

SuperCollider and ixi lang workshop in Paris – June 19th, 2012

Realtime Sound Spatialization with Wave Field Synthesis. Workshop in Barcelona 18–20/6 2012

SuperCollider source code now at GitHub

SuperCollider Workshop in Morelia, Mexico (5–6 Oct 2012)

SuperCollider Basics

- ▶ SuperColliderをダウンロード
- ▶ 最新版は、version 3.6.5 (2013年10月現在)
- ▶ <http://supercollider.sourceforge.net/downloads/>

Downloads



For Mac:

- [SuperCollider 3.5.5 installer](#)

For older Macs (10.4, 32-bit universal build):

- [SuperCollider3.4.4-10.4](#)



SuperCollider builds from source on all common Linux systems. Latest stable release:

- [SuperCollider 3.5.5 source](#)

Furthermore:

-  [Ubuntu packages available here](#)
- [Debian packages available here](#)
- [RedHat or Fedora packages provided by PlanetCCRMA](#)



SuperCollider 3.5.5 for Windows (32-bit):

- [SuperCollider 3.5.5 installer](#)

Older release:

- [Release Candidate 1 of SuperCollider 3.4](#)

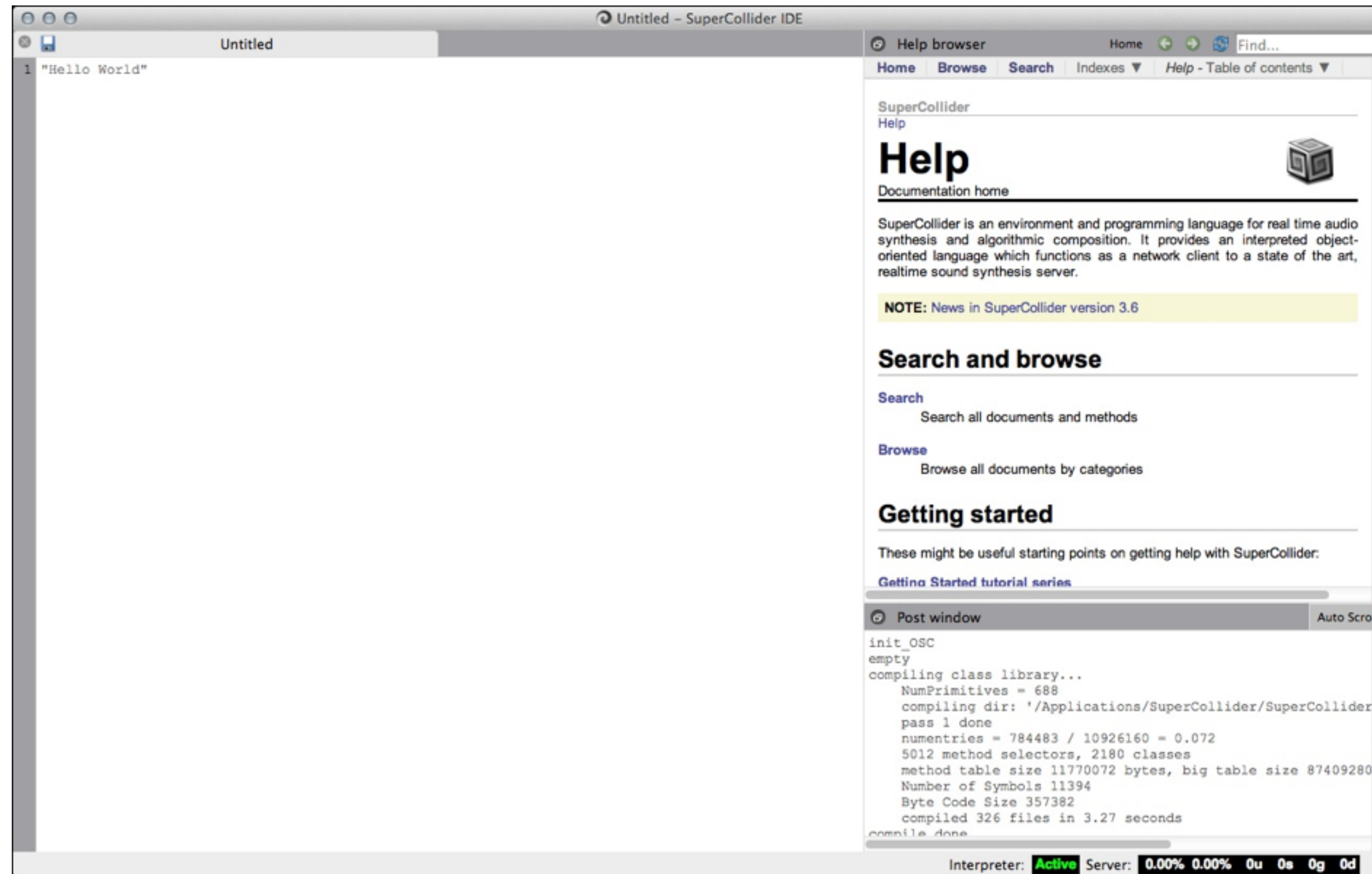
The SuperCollider Language

- ▶ SuperColliderの独特な文法
- ▶ Smalltalkのオブジェクト指向な言語構造と、C言語系の制御構造などの機能を足したような感じ

```
// print "Hello world!"
"Hello world!".postln;
// play a mixture of an 800 Hz sine tone and pink noise
{ SinOsc.ar(800, 0, 0.1) + PinkNoise.ar(0.01) }.play;
// 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;
// list iteration: multiply the elements of a collection by their indices
[1, 2, 5, 10, -3].collect {
    arg elem, idx;
    elem * idx;
};
// factorial function
f = {
    arg x;
    if(x == 0) { 1 } { f.(x-1) * x }
};
```

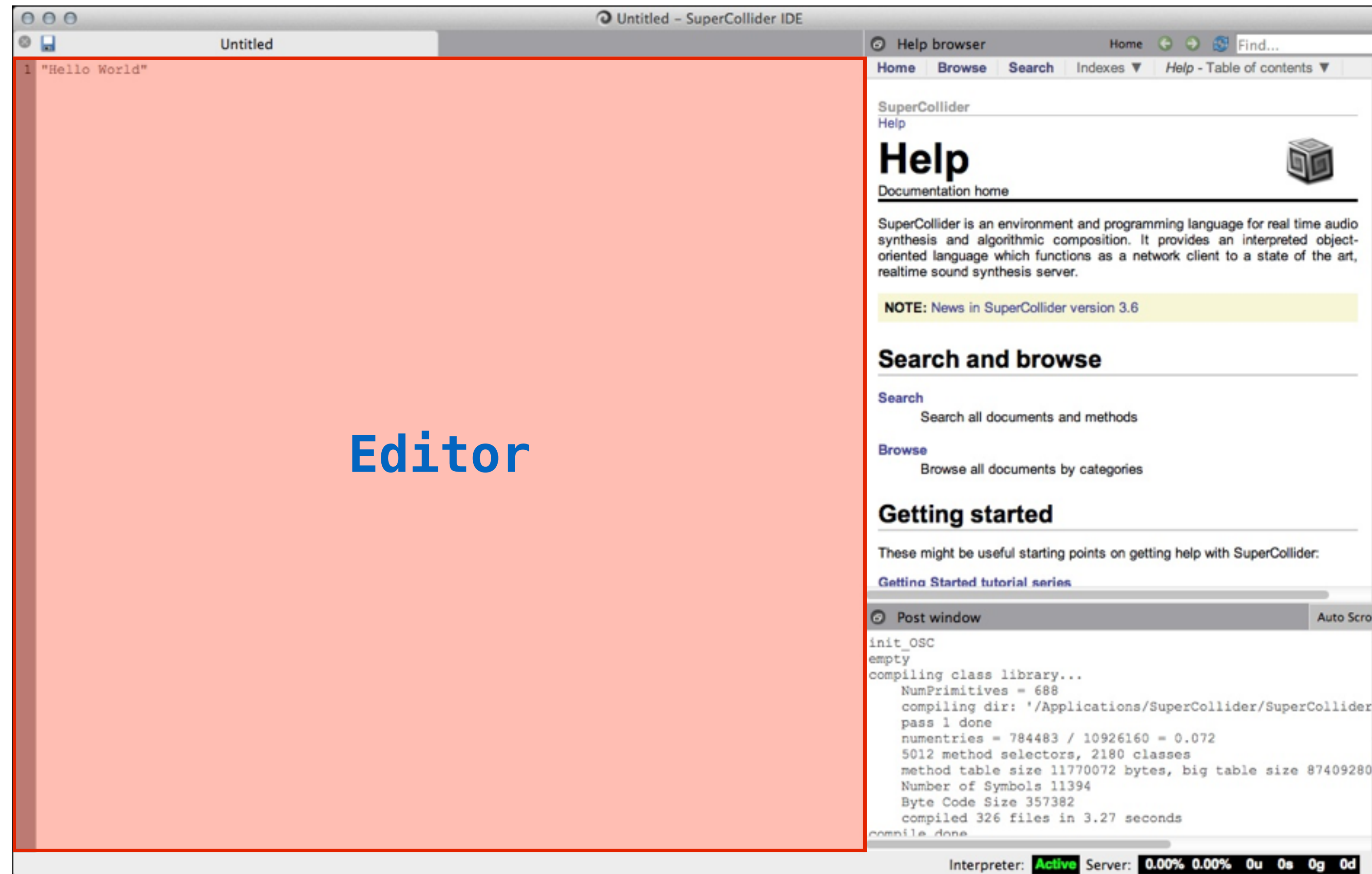

The SuperCollider Language

- ▶ SuperCollider.app を起動
- ▶ 最新バージョン3.6は、統合開発環境(IDE)となっている



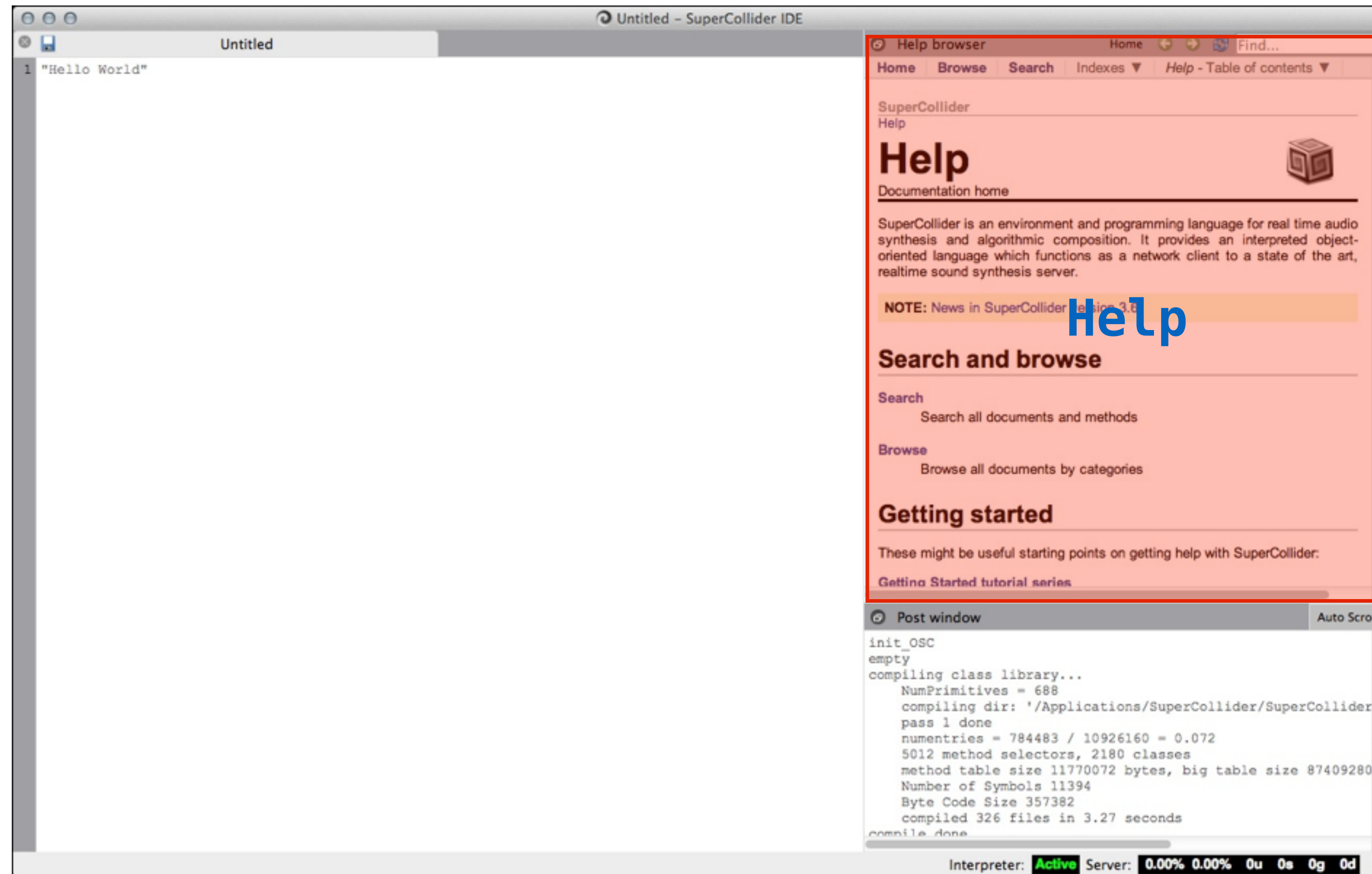
The SuperCollider Language

- ▶ 左側のエディターにプログラムを入力
- ▶ タブで複数のファイルを同時に編集可能



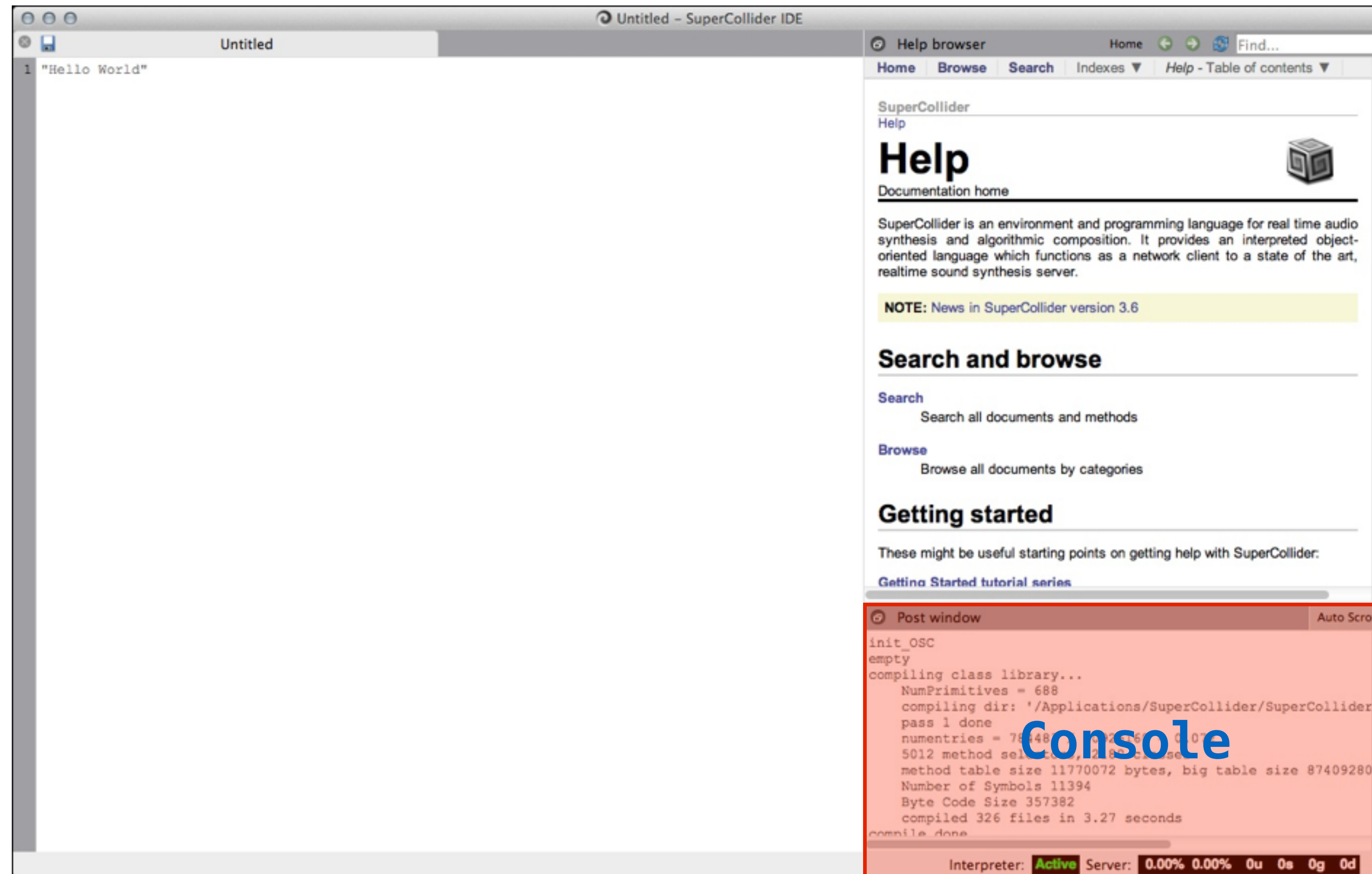
The SuperCollider Language

- ▶ 右上には、ヘルプ画面
- ▶ チュートリアルや、様々なドキュメントが参照可能



The SuperCollider Language

- ▶ 右下はコンソール
- ▶ システムの状態、プログラムからの出力などを表示



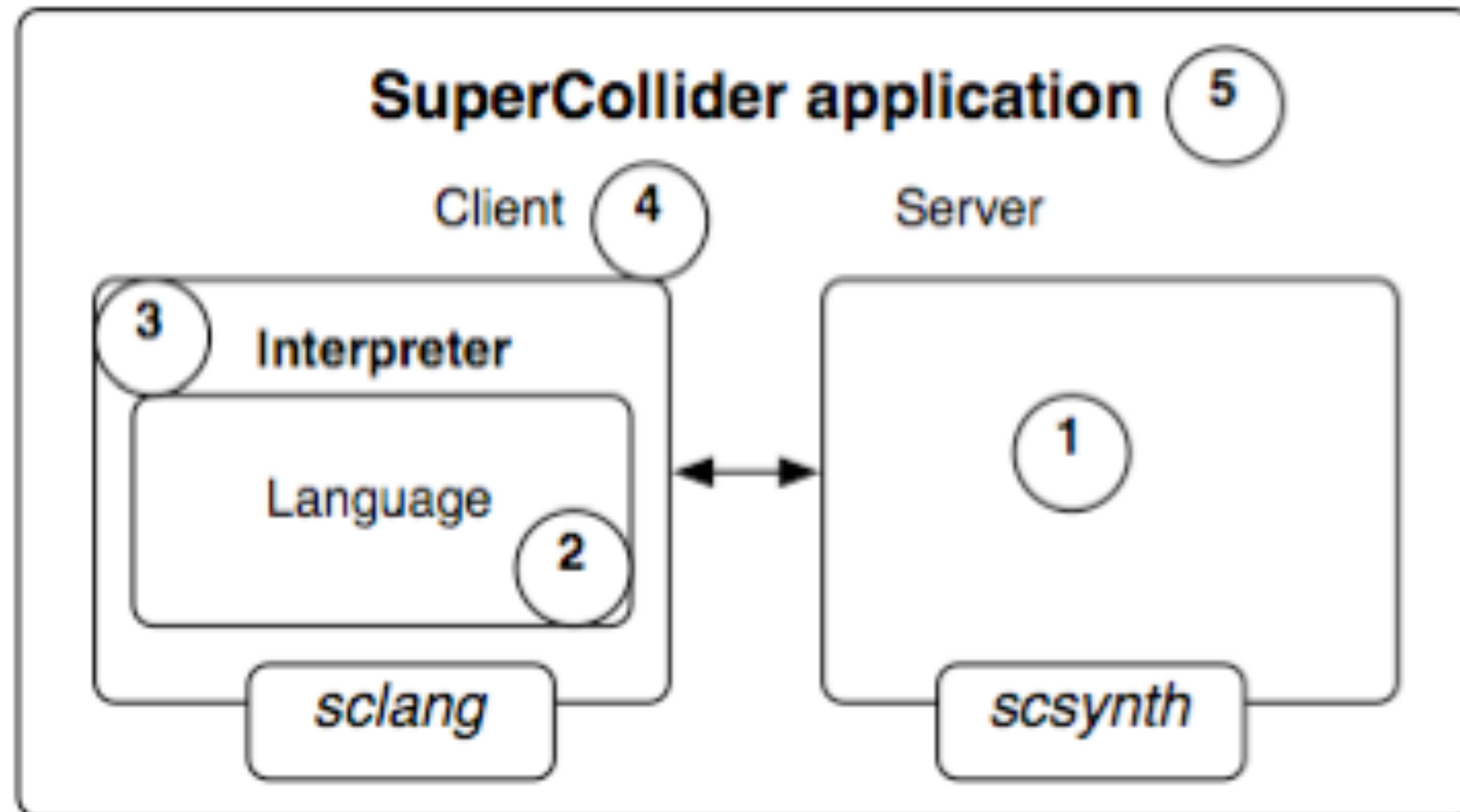
The SuperCollider Language

- ▶ とりあえず、“Hello World”

```
"Hello world"
```

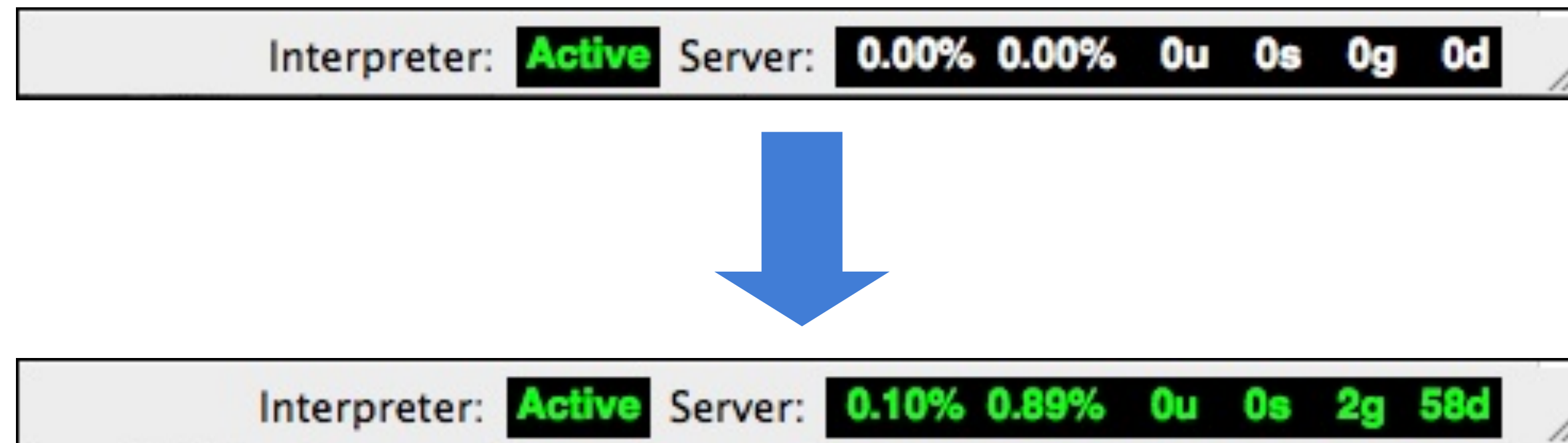
The SuperCollider Language

- ▶ SuperCollider アプリケーションの構造



The SuperCollider Language

- ▶ 音を生成するには、まずSC Serverを起動しなくてはならない
- ▶ メニューから、Language > Boot Server を選択
- ▶ もしくは「Command + B」キーでもOK
- ▶ 以下のように表示が変化すれば準備完了



The SuperCollider Language

- ▶ 音を鳴らしてみる
- ▶ localhost serverを起動.
- ▶ 下記のコードを記入して、“Command + enter”

```
play({SinOsc.ar(LFNoise0.kr(12, mul: 600, add: 1000), 0.3)})
```

- ▶ 音を止めるには“Command + . (period)”
- ▶ コード内の数値を変化させてみる
- ▶ 何が変化したか?
- ▶ 12, 600, 100, 0.3

The SuperCollider Language

- ▶ コードサンプル: Wire 140
- ▶ 世界中のアーティストからよせられた22曲
- ▶ 全てのコードは、140文字(Twitter!)以内で書かれている
- ▶ それにも関わらず、バラエティーに富んだ音楽が楽しめる
- ▶ <http://supercollider.sourceforge.net/sc140/>

The SuperCollider Language

► Example: Wire 140

01

Nathaniel Virgo

```
{LocalOut.ar(a=CombN.ar(BPF.ar(LocalIn.ar(2)*7.5+Saw.ar([32,33],0.2),  
2**LFNoise0.kr(4/3,4)*300,0.1).distort,2,2,40));a}.play//#supercollider
```

02

LFSaw

```
{Splay.ar(Ringz.ar(Impulse.ar([2, 1, 4], [0.1, 0.11, 0.12]), [0.1, 0.1, 0.5])) *  
EnvGen.kr(Env([1, 1, 0], [120, 10]), doneAction: 2)}.play
```

03

Tim Walters

```
play{({|k|({|i|y=SinOsc; y.ar(i*k*k, y.ar(i*k**i/  
[4,5])*Decay.kr(Dust.kr(1/4**i), y.ar(0.1)+1*k+i, k*999))}!8).product}!16).sum}//  
#supercollider
```

04

Nathaniel Virgo

```
b=Buffer.read(s,"sounds/a11wlk01.wav");play{t=Impulse.kr(5);PlayBuf.ar(1,b,  
1,t,Demand.kr(t,0,Dseq(1e3*[103,41,162,15,141,52,124,190],4)))!2}
```

...

The SuperCollider Language

- ▶ 別のサンプル

```
play({RLPF.ar(Dust.ar([12, 15]), LFNoise1.ar(1/[3, 8], 1500, 1600), 0.02)})
```

- ▶ 数値を変化させてみよう!
- ▶ [12, 15]
- ▶ [3, 8]
- ▶ 1500
- ▶ 1600
- ▶ 0.02

The SuperCollider Language

- ▶ このコードをわかりやすく分解すると下記のようなになる

```
play({RLPF.ar(Dust.ar([12, 15]), LFNoise1.ar(1/[3, 8], 1500, 1600), 0.02)})
```

```
play(  
  {  
    RLPF.ar(  
      Dust.ar(  
        [12, 15]  
      ),  
      LFNoise1.ar(  
        1/[3, 8],  
        1500,  
        1600  
      ),  
      0.02  
    )  
  }  
)
```


Enclosures

- ▶ たくさんの括弧が存在する
 - ▶ (parentheses)
 - ▶ [brackets]
 - ▶ {braces}
 - ▶ "quotation"
- ▶ これらの括弧の役割を知ることがSuperCollider理解のコツ

Enclosures

- ▶ “...” - ダブルクォーテーション
- ▶ 「文字列」をあらわす

- ▶ ‘...’ - シングルクォーテーション
- ▶ 「シンボル」をあらわす
- ▶ \aSymbol = ‘aSymbol’

Enclosures

- ▶ (...) - 小括弧
- ▶ 関数の引数(argument)をあらわす

```
message(arg1, arg2, arg3...)
```

- ▶ その他、計算の順番を意味する場合も

```
5 + (10 * 4)
```


Enclosures

- ▶ [...] - 大括弧
- ▶ アイテムの集合をあらわす - 配列(Array)など
- ▶ Arrayには様々なデータを格納できる - numbers, texts, functions, all of patch ...etc.
- ▶ Arrayにメッセージを与えると、様々な操作が可能
- ▶ reverse, scramble, mirror, rotate, midicps, choose, permute ...etc.

Enclosures

- ▶ [...] - 大括弧
- ▶ 音楽に応用した例

```
[0, 11, 10, 1, 9, 8, 2, 3, 7, 4, 6, 5].reverse  
12 - [0, 11, 10, 1, 9, 8, 2, 3, 7, 4, 6, 5].reverse  
[0, 2, 4, 5, 6, 7, 9, 11].scramble  
[60, 62, 64, 67, 69].mirror  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].rotate  
[60, 62, 64, 65, 67, 69, 71].midicps.round(0.1)  
[1, 0.75, 0.5, 0.25, 0.125].choose  
0.125 * [1, 2, 3, 4, 5, 6, 7, 8].choose  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].permute(6)
```


Enclosures

- ▶ [...] - 大括弧
- ▶ 大括弧は、複数のチャンネルを意味することがある

```
{Blip.ar(25, LFNoise0.kr(5, 12, 14), 0.3)}.play  
{Blip.ar(25, LFNoise0.kr([5, 10], 12, 14), 0.3)}.play  
{Blip.ar(25, LFNoise0.kr([5, 10, 2, 25], 12, 14), 0.3)}.play  
{Blip.ar(25, LFNoise0.kr([5, 4, 7, 9, 5, 1, 9, 2], 12, 14), 0.3)}.play
```

Enclosures

- ▶ {...} - 中括弧
- ▶ 「関数」をあらわす
- ▶ 下の2つのコードを比較してみる (dupはくりかえしのこと)

```
dup(rand(1000.0), 5)  
dup({rand(1000.0)}, 5)
```

- ▶ 下の行は、以下の書き方と同じ意味になる

```
[rand(1000.0), rand(1000.0), rand(1000.0),  
 rand(1000.0), rand(1000.0)]
```

Nesting

- ▶ ネスティングの例 (構造が別の構造を取り囲むこと)

```
exprand(1.0, 1000.0)
dup({exprand(1.0, 1000.0)}, 100)
sort(dup({exprand(1.0, 1000.0)}, 100))
round(sort(dup({exprand(1.0, 1000.0)}, 100)), 0.01)
```

- ▶ 音響合成におけるネスティング

```
play(
  {
    CombN.ar(
      SinOsc.ar(
        midicps(
          LFNoise1.ar(3, 24,
            LFSaw.ar([5, 5.123], 0, 3, 80)
          )
        ),
        0, 0.4),
    1, 0.3, 2)
  }
)
```


Enclosures

- ▶ { ... } - 中括弧
- ▶ 関数はメッセージを理解して動作する

```
{LFNoise0.ar}.play  
{LFNoise0.ar(10000)}.plot  
{LFNoise0.ar(10000)}.scope  
{100.rand}.dup(10)  
{100.rand} ! 10  
{100.rand}.dup(10).postln.plot  
{100.rand}.dup(100).sort.plot
```

Messages and Arguments.

- ▶ Supercolliderのコードは、メッセージと引数から構成される

```
message(arg1, arg2, arg3...)
```

- ▶ e.g.

```
rand(100)  
exprand(1.0, 100.0)
```

- ▶ メッセージはオブジェクト(レシーバ)とともに用いられることもある

```
SinOsc.ar(arglist)  
Mix.fill(arglist)
```


Messages and Arguments.

▶ メッセージの例

```
dup("echo", 20)
round([3.141, 5.9265, 358.98], 0.01)
sort([23, 54, 678, 1, 21, 91, 34, 78])
round(dup({exprand(1, 10)}), 100), 0.1)
sort(round(dup({exprand(1, 10)}), 100), 0.1))
```

Receiver and Message, UGens

```
Receiver.message(arglist)
```

- ▶ レシーバとメッセージ
- ▶ レシーバ
 - ▶ メッセージを受け取るオブジェクト
 - ▶ 大文字から始まる
- ▶ メッセージ
 - ▶ オブジェクトに特定の操作を依頼する
 - ▶ 引数をもつ場合もある

Receiver and Message, UGens

- ▶ Numbers, Functions, Arrays, Stringsなどはレシーバーになることができる

```
[45, 13, 10, 498, 78].sort  
"echo".dup(20)  
50.midicps  
444.cpsmidi  
100.rand  
{100.rand}.dup(50)  
[1.001, 45.827, 187.18].round(0.1)  
"I've just picked up a fault in the AE35 unit".speak
```


Receiver and Message, UGens

- ▶ “.(ピリオド)”を接続していったネスト構造を実現

```
1000.0  
1000.0.rand  
1000.0.rand.round(0.01)  
1000.0.rand.round(0.01).post  
{1000.0.rand.round(0.01).postln}.dup(100).plot  
{1000.0.rand.round(0.01).postln}.dup(100).postln.sort.plot  
1000.0.rand.round(0.01).postln.asString.speak
```

Receiver and Message, UGens

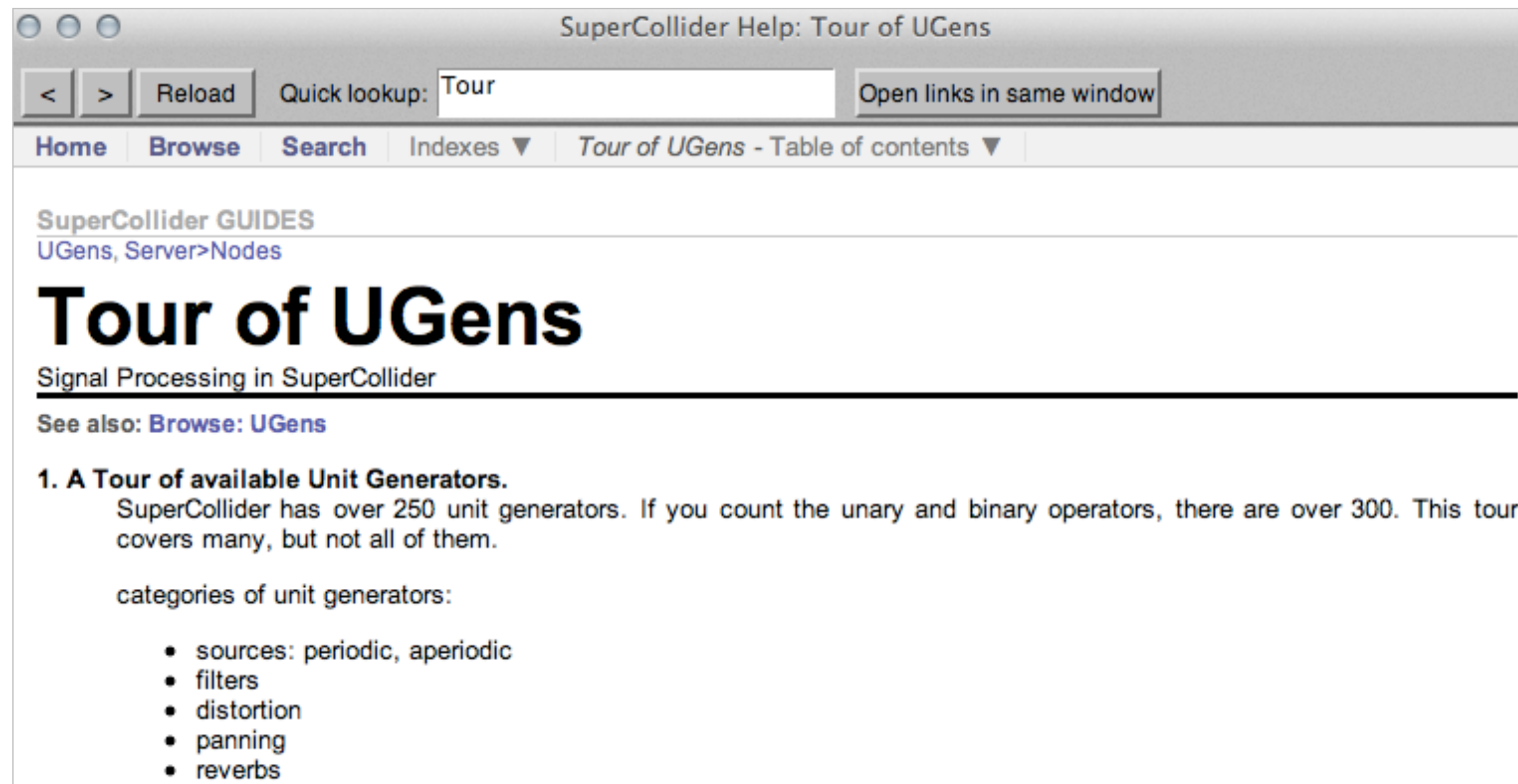
- ▶ 音響合成のためのレシーバ(オブジェクト)が多数存在
- ▶ これらを総称して UGen (ユニットジェネレータ) と呼ぶ
- ▶ CombN, SinOsc, LFNoise1, LFSaw ...etc.

- ▶ Ugenは信号の計算に特化している
- ▶ これらがソフトウェア楽器を定義する際の基本パーツとなる
- ▶ Ugenで生成する信号には二つ
 - ▶ オーディオ信号: Ugen.ar(...)
 - ▶ コントロール信号: Ugen.kr(...)

```
LFNoise1.kr(10,100)
```

Receiver and Message, UGens

- ▶ SCのヘルプにはとても丁寧にUGenの解説とサンプルが掲載されている
- ▶ ヘルプメニューで“Tour of Ugens”でサーチしてみる
- ▶ UGenの機能を外観するツアーが用意されている



Variables

- ▶ 変数:
- ▶ aからzのアルファベットは宣言なしに使用可能

```
(  
  a = 440;  
  b = 3;  
  c = "math operations";  
  [c, a, b, a*b, a + b, a.pow(b), a.mod(b)]  
)
```

- ▶ 上のコードは下と同じ意味になる

```
["math operations", 440, 3, 440*3, 440 + 3, 440.pow(3), 440.mod(3)]
```

Variables

▶ 変数の使用例

```
(  
  {  
    r = MouseX.kr(1/3, 10);  
    SinOsc.ar(mul: Linen.kr(Impulse.kr(r), 0, 1, 1/r))  
  }.play  
)
```

Variables

▶ もう少し複雑な例

```
(  
p = {  
  r = Line.kr(1, 20, 60);  
  t = Impulse.kr(r);  
  e = Linen.kr(t, 0, 0.5, 1/r);  
  f = TRand.kr(1, 10, t);  
  Blip.ar(f*100, f, e)  
}.play  
)
```


Variables

▶ さらに複雑な例

```
(
{
  r = Impulse.kr(10);
  c = TRand.kr(100, 5000, r);
  m = TRand.kr(100, 5000, r);
  PM0sc.ar(c, m, 12)*0.3
}.play
)

(
{
  var rate = 4, carrier, modRatio;
  carrier = LFNoise0.kr(rate) * 500 + 700;
  modRatio = MouseX.kr(1, 2.0);
  PM0sc.ar(carrier, carrier*modRatio, 12)*0.3
}.play
)
```

来週につづく！
