Luis Sanz | Aug 2025 | www.luissanz.ch

# SuperCollider Architecture

### 1. Client

- This is what you interact with directly — the **IDE*** (**sclang**) where you type code.
- It handles things like **programming**, **scheduling, and sending instructions.**
- The client doesn't make sound by itself — it just **tells the server what to do.**

*** IDE** = Integrated Development Environment

It's a piece of software that gives you a complete environment for writing, running, and debugging code.
Instead of just a plain text editor, an IDE usually includes: **Code editor, Evaluator / Runner, Error console.**

### 2. Server

- This is a separate program called **scsynth** (sometimes also supernova).
- The server does the heavy work: it actually **runs the UGens** (oscillators, filters, effects) and **produces audio**.
- You don't write code *inside* the server, but you can send it instructions in the form of **SynthDefs** and **Synths**.

### Example of the Distinction

- When you write:

```
SynthDef(\sineTone, { SinOsc.ar(440) * 0.1 }).add;
```

The **SynthDef** is created on the client.

When you call **.add**, it gets sent to the server and stored there.

Then, when you write:

```
x = Synth(\sineTone);
```

- You're telling the **client** to ask the **server** to make a new **Synth instance** from the definition it already has.
- The **server** then generates the actual sound.

### Simple Analogy

- **Client** = the **composer/conductor** writing the score and giving instructions.
- **Server** = the **orchestra** that actually plays the music.
- The **score (SynthDef)** lives in the client until you hand it to the orchestra — then the orchestra can play it as many times as you like.

## What is a SynthDef in SuperCollider?

SynthDef stands for Synth Definition. It's basically a blueprint (a recipe) for creating sound-producing (or processing) nodes on the SuperCollider server (scsynth). You use a SynthDef to define the structure of a sound: what UGens (oscillators, filters, envelopes, etc.) are used, and how they are connected. Once defined, you can create multiple Synths (instances) from the same SynthDef, each with different parameter values.

#### General Syntax

```
SynthDef(\name, { | arg1=default, arg2=default, … |
    // signal chain
}).add;
```

**\name** → symbol name of the synth.

Inside the **{ ... }** function, you describe the audio graph with UGens.

**.add** → sends the definition to the server, so you can use it.

#### Example: A Simple Sine Wave Synth

// Define the synth

```
(
SynthDef(\sineTone, { |freq = 440, amp = 0.1 |
       var env = EnvGen.kr(Env.perc(0.01, 0.2), doneAction:2);
       var sig = SinOsc.ar(freq) * amp * env;
       Out.ar(0, sig!2); // send stereo output
}).add;
)

// Play an instance
x = Synth(\sineTone, [\freq, 440, \amp, 0.2]);

// Control it
x.set(\freq, 660);   // change frequency
x.set(\amp, 0.05);   // change amplitude

// Release it
x.free;
```

#### Explanation of the Example

**Args** → freq, amp, gate are control arguments you can change later.
**Envelope** → EnvGen.kr with Env.perc ensures smooth attack/release instead of clicks.
**Signal** → SinOsc.ar(freq) is the sine oscillator.
**Output** → Out.ar(0, sig!2) sends the sound to both left & right channels.
**Usage** → Synth(\sineTone, [...]) creates an instance.

## Scales and Degrees in SuperCollider

In SuperCollider, **Scales** and **Degrees** are used to organize pitches in a musical way.

#### 1. Scale

- A Scale is a collection of notes.
- SuperCollider already includes many common ones, such as:

```
Scale.major      // C D E F G A B
Scale.minor      // C D Eb F G Ab Bb
Scale.pentatonic // C D E G A
Scale.chromatic  // all 12 semitones
```

- You can choose a scale and use it to generate melodies.

Think of a **scale** as the palette of notes you want to work with.

#### 2. Degree

- A Degree is the position of a note inside the scale.
- Example in C major (C–D–E–F–G–A–B):
    - Degree 0 = C
    - Degree 1 = D
    - Degree 2 = E
    - Degree 3 = F
    - Degree 4 = G
    - etc.

Degrees let you **refer to notes by number** instead of exact frequencies.

#### 3. Using Scale + Degree in Code

Here's how to play a simple melody:

```
(
s.boot;

Pbind(
    \instrument, \default,
    \scale, Scale.major,        // choose a scale
    \degree, Pseq([0, 1, 2, 3, 4, 5, 6, 7], inf), // play degrees
    \octave, 5,                 // base octave
    \dur, 0.25                  // note duration
).play;
)
```

This plays the **major scale step by step**.
You can change the scale (Scale.minor, Scale.pentatonic, etc.) or the degrees [0, 2, 4] to make chords and melodies.

**Summary**:

- **Scale** = which notes you want to use.
- **Degree** = which step of the scale you play.
- Together, they let you make melodies without worrying about exact frequencies.

**1.- How I make chords?**

Chords in SuperCollider with **scales + degrees** are super easy once you see the trick.

A chord = play **several degrees at once**.
Instead of a single number, give SuperCollider an **array of degrees**:

```
(
Pbind(
    \instrument, \default,
    \scale, Scale.major,
    \degree, Pseq([[0,2,4], [1,3,5], [2,4,6]], inf), // chords!
    \octave, 5,
    \dur, 1
).play;
)
```

This plays a I–II–III chord progression in the major scale.

- [0,2,4] = root triad (C–E–G if root is C)
- [1,3,5] = next triad (D–F–A)
- [2,4,6] = next triad (E–G–B)

**2. Chord Shapes**

Common chord degree patterns:

- **Major triad**: `[0,2,4]`
- **Minor triad**: `[0,2,4] in minor scale`
- **Seventh chord**: `[0,2,4,6]`
- **Suspended chord**: `[0,3,4] (sus4)`

Example:

```
(
Pbind(
    \instrument, \default,
    \scale, Scale.minor,
    \degree, Pseq([ [0,2,4,6] ,  [3,5,7,9] ], inf), // 7th chords
    \octave, 4,
    \dur, 1.5
).play;
)
```

**3. Arpeggios (breaking chords into melody)**
Instead of playing all chord notes at once, you can sequence them:

```
(
Pbind(
    \instrument, \default,
    \scale, Scale.major,
    \degree, Pseq([0,2,4,2], inf),  // play triad as arpeggio
    \octave, 5,
    \dur, 0.25
).play;
)
```

This time, the notes are played one after another:

`C → E → G → E →`

That's an **arpeggio** (broken chord).

**Chord vs Arpeggio:**

```
(
// chord
Pbind(\instrument, \default, \scale, Scale.major,
    \degree, Pseq([ [0,2,4] ], inf), \dur, 1).play;
)

(
// arpeggio
Pbind(\instrument, \default, \scale, Scale.major,
    \degree, Pseq( [0,2,4] , inf), \dur, 0.25).play;
)
```

- First: notes **together** = chord.
- Second: notes **one after another** = arpeggio.

**Summary**:

- **Single degree (e.g., 0)** → single note
- **Array of degrees (e.g., [0,2,4])** → chord
- Change the scale → changes chord color (major, minor, pentatonic, etc.)

**\strum and \legato**

**\strum**
- Used when you play a **chord** (an array of degrees, e.g. [0,2,4]).
- Instead of triggering all notes **at the exact same time**, \strum delays each note slightly.
- Makes it sound more like a guitar strum or an arpeggiator.

Example:

```
(
// Chord without strum = all notes at once
Pbind(
    \instrument, \default,
    \scale, Scale.major,
    \degree, Pseq([[0,2,4]], inf), // C major chord
    \dur, 1
).play;
)

(
// Same chord, but strummed
Pbind(
    \instrument, \default,
    \scale, Scale.major,
    \degree, Pseq([[0,2,4]], inf),
    \dur, 1,
    \strum, 0.05   // delay between notes (in seconds)
).play;
)
```

With \strum, you'll hear C → E → G instead of all three at once, but still inside the same beat.

**\legato**
- Controls how long a note lasts relative to its \dur.
- \dur = spacing between notes.
- \legato = proportion of that spacing used for the note's length.

```
(
// short notes (staccato-like)
Pbind(
    \instrument, \default,
```

```
    \scale, Scale.major,
    \degree, Pseq([0,2,4,5], inf),
    \dur, 0.5,
    \legato, 0.3   // note length = 30% of duration
).play;
)

(
// overlapping notes (smooth/legato)
Pbind(
    \instrument, \default,
    \scale, Scale.major,
    \degree, Pseq([0,2,4,5], inf),
    \dur, 0.5,
    \legato, 1.5   // note length = 150% of duration
).play;
)
```

**Summary:**

- \legato < 1 → shorter notes (gaps between them).
- \legato = 1 → notes last exactly until next one.
- \legato > 1 → overlapping notes (smooth).

So in our context (scales, degrees, chords):

- Use **\strum** to make chords sound more natural (like plucked/rolled instead of blocky).
- Use **\legato** to shape how connected or detached your notes sound.

**Summary:**

- **Scale** = set of notes
- **Degree** = step number in the scale
- Array = chord, Sequence = arpeggio
- **\strum** = spread chord notes in time
- **\legato** = control note length / overlap