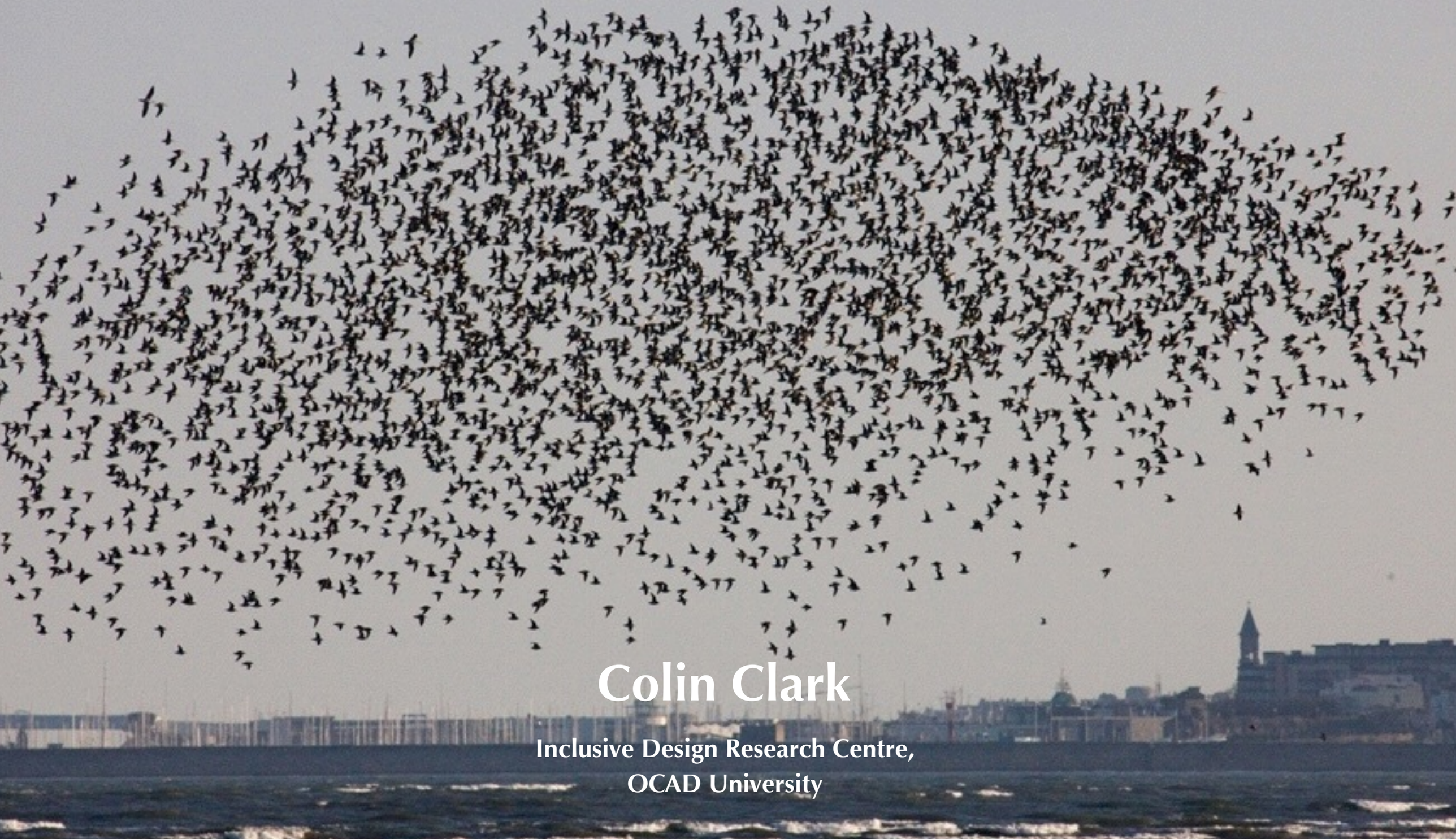
Flocking

creative audio synthesis for the web



Colin Clark

Inclusive Design Research Centre,
OCAD University

FLOCKING

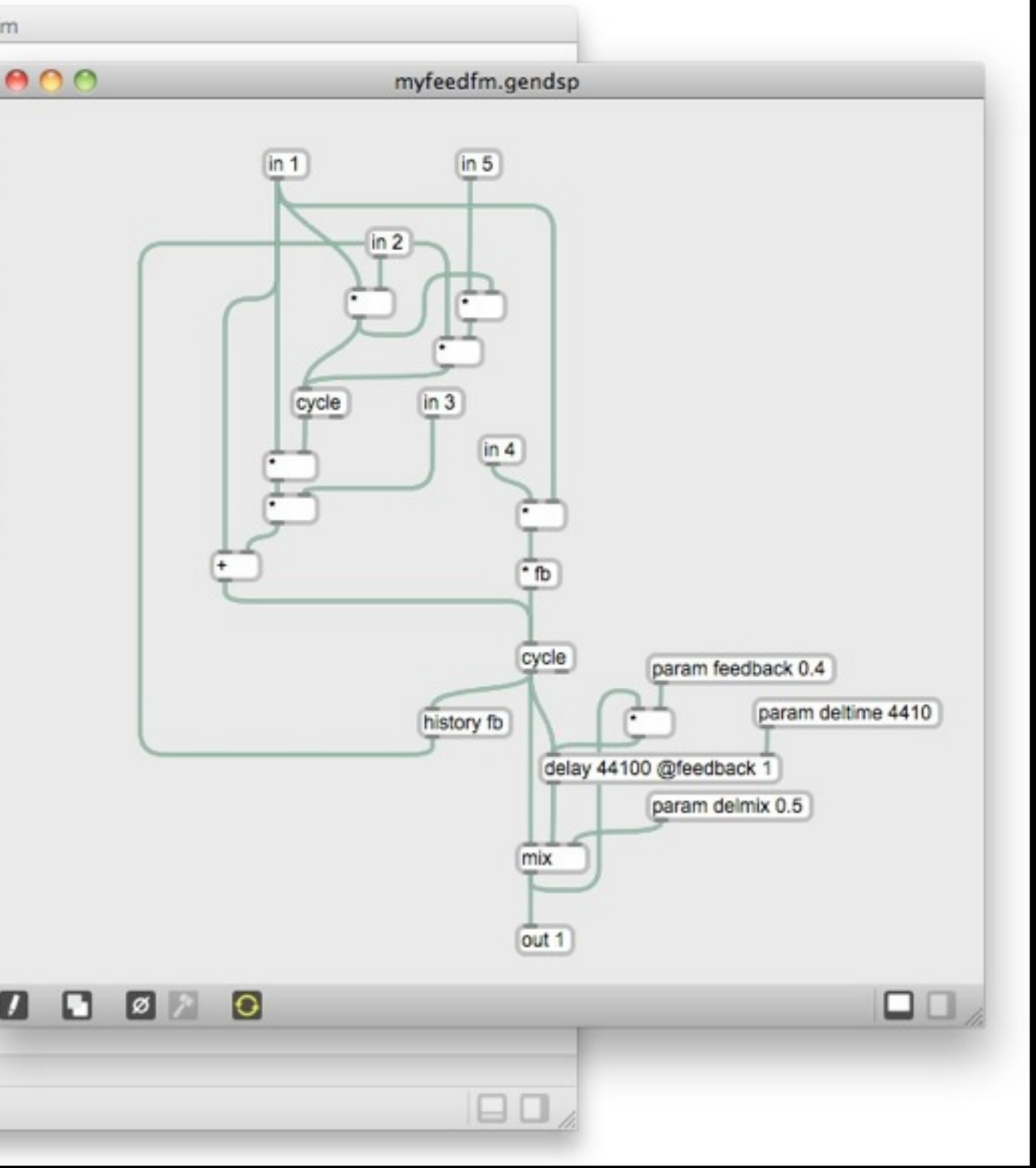
- Audio synthesis framework written entirely in JavaScript
- Inspired by SuperCollider, but increasingly different
- Very open: dual MIT/GPL license

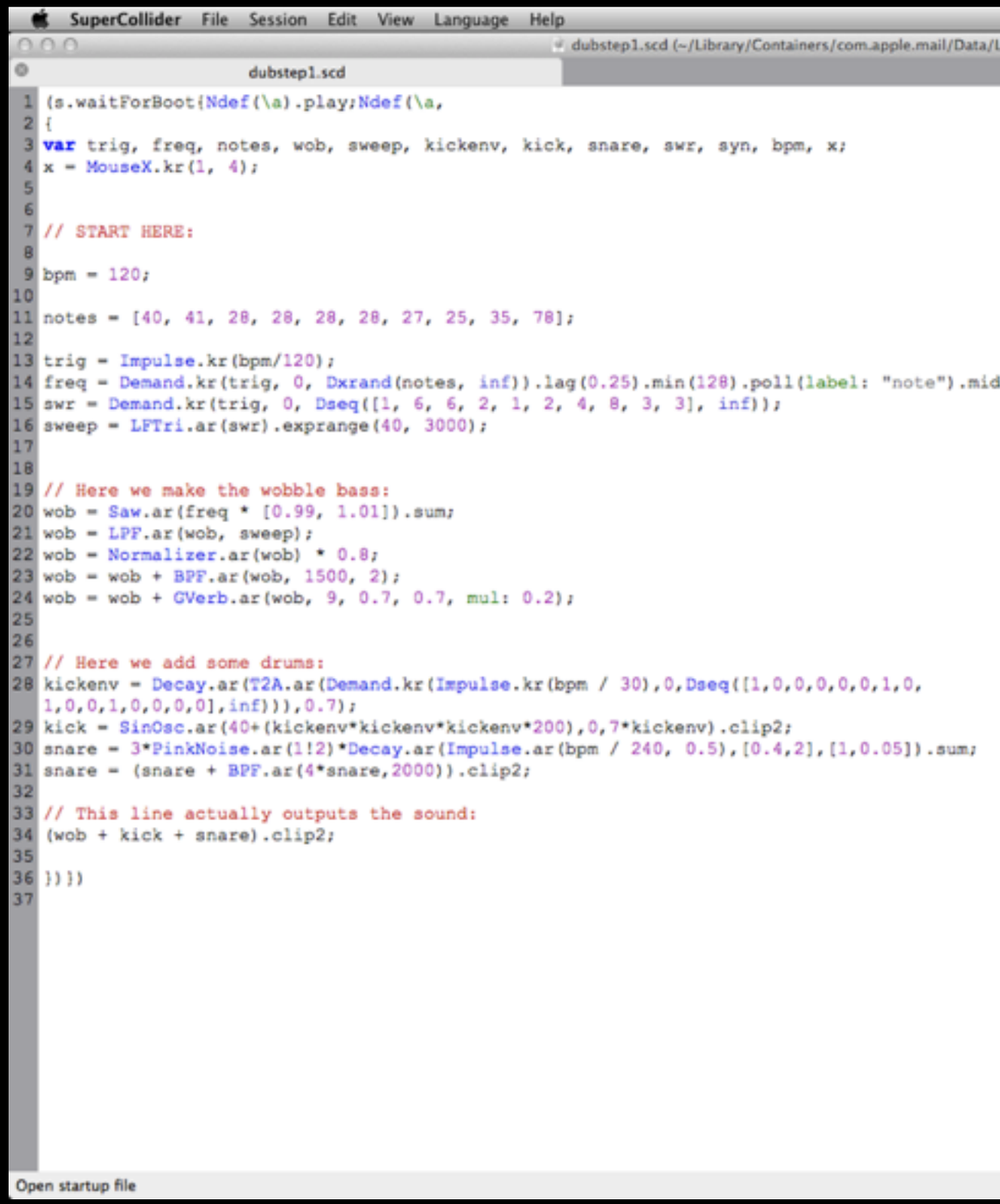
`flockingjs.org`

`github.com/colinbdclark/flocking`

Context and Motivations

- Explosion of devices and platforms
- Prevalence of “walled garden” computer music environments
- Challenge of building polished user interfaces and sharing them


myfeedfm.gendsp
in 1
in 5
in 2
cycle
in 3
in 4
+
* fb
cycle
param feedback 0.4
history fb
param deltime 4410
delay 44100 @feedback 1
param delmix 0.5
mix
out 1

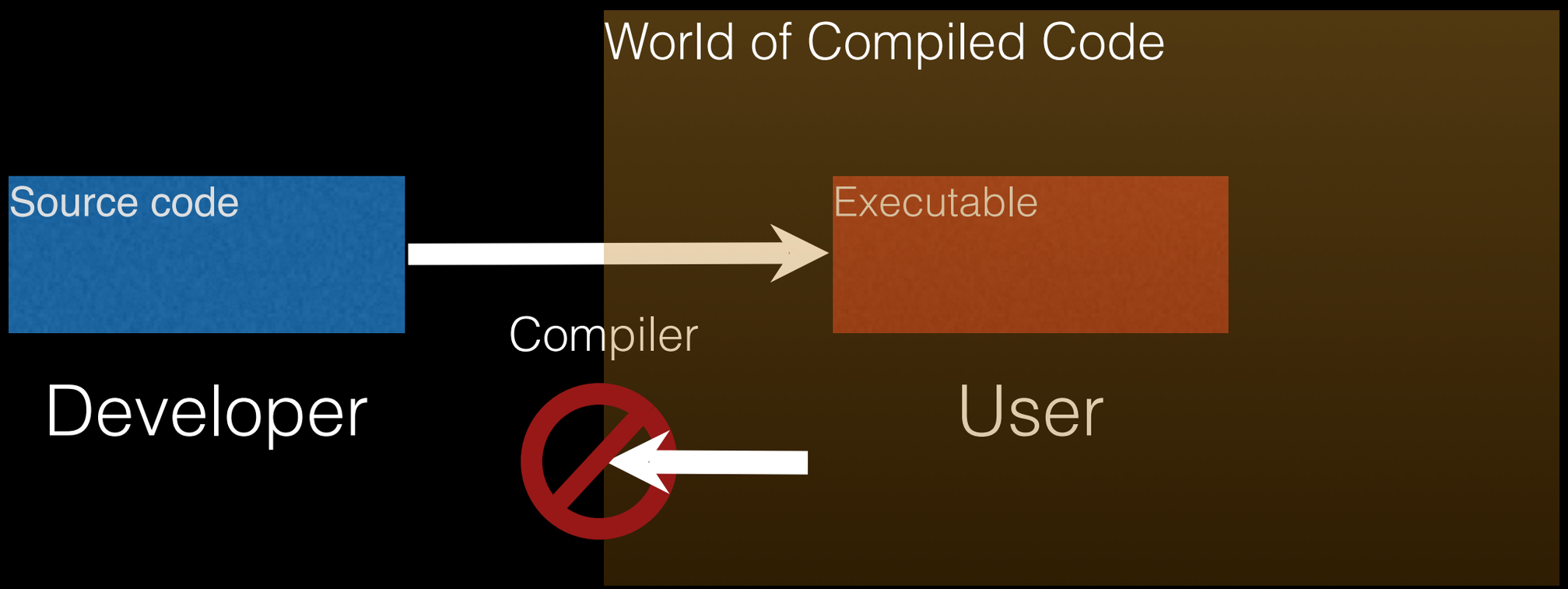

SuperCollider File Session Edit View Language Help
dubstep1.scd (~/Library/Containers/com.apple.mail/Data/L
dubstep1.scd
(s.waitForBoot{Ndef(\a).play;Ndef(\a,
{
var trig, freq, notes, wob, sweep, kickenv, kick, snare, swr, syn, bpm, x;
x = MouseX.kr(1, 4);
// START HERE:
bpm = 120;
notes = [40, 41, 28, 28, 28, 28, 27, 25, 35, 78];
trig = Impulse.kr(bpm/120);
freq = Demand.kr(trig, 0, Dxrand(notes, inf)).lag(0.25).min(128).poll(label: "note").mid
swr = Demand.kr(trig, 0, Dseq([1, 6, 6, 2, 1, 2, 4, 8, 3, 3], inf));
sweep = LFTri.ar(swr).exprange(40, 3000);
// Here we make the wobble bass:
wob = Saw.ar(freq * [0.99, 1.01]).sum;
wob = LPF.ar(wob, sweep);
wob = Normalizer.ar(wob) * 0.8;
wob = wob + BPF.ar(wob, 1500, 2);
wob = wob + GVerb.ar(wob, 9, 0.7, 0.7, mul: 0.2);
// Here we add some drums:
kickenv = Decay.ar(T2A.ar(Demand.kr(Impulse.kr(bpm / 30),0,Dseq([1,0,0,0,0,0,1,0,
1,0,0,1,0,0,0,0],inf))),0.7);
kick = SinOsc.ar(40+(kickenv*kickenv*kickenv*200),0,7*kickenv).clip2;
snare = 3*PinkNoise.ar(1!2)*Decay.ar(Impulse.ar(bpm / 240, 0.5),[0.4,2],[1,0.05]).sum;
snare = (snare + BPF.ar(4*snare,2000)).clip2;
// This line actually outputs the sound:
(wob + kick + snare).clip2;
}))
Open startup file

vs.

Approaches to Interoperability

1. Inter-environment messaging (e.g. Open Sound Control, SuperCollider server)
2. Code generators/compilers (e.g. Faust)
3. Cross-language APIs (e.g. Max's Patcher API for Java and JavaScript)
4. Macros and metaprogramming (e.g. Lisp, Extempore)

The Unidirectionality of Compilers

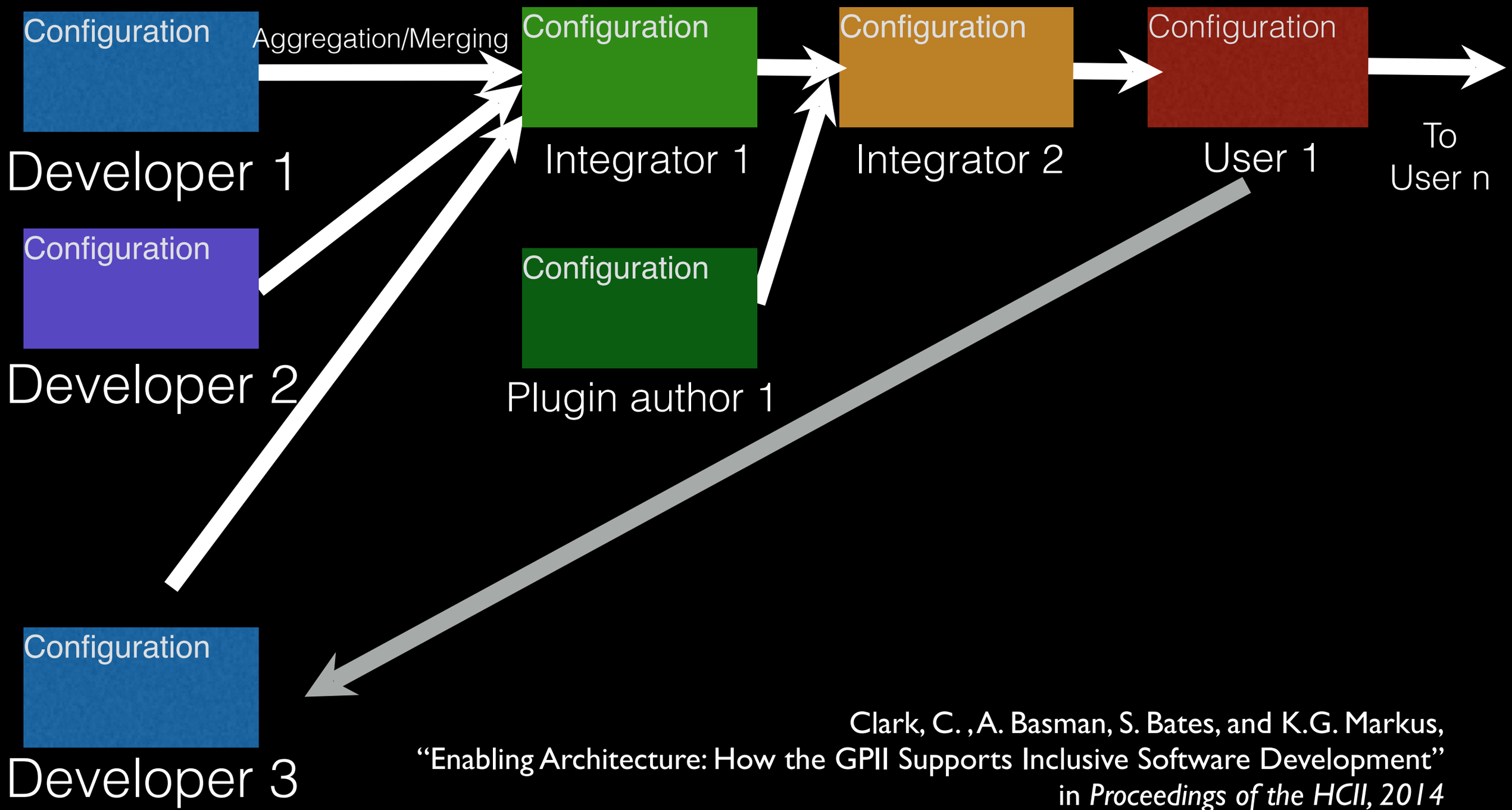


Clark, C. , A. Basman, S. Bates, and K.G. Markus,
"Enabling Architecture: How the GPII Supports Inclusive Software Development"
in *Proceedings of the HCII, 2014*

Bi-directional Interoperability

Preserves sufficient semantics and landmarks so that a computer music artifact can be inspected, overridden, and extended by humans, graphical tools and generative algorithms not only at creation time but *throughout the process of being used and maintained.*

Goal: Bi-directionality



Clark, C. , A. Basman, S. Bates, and K.G. Markus,
"Enabling Architecture: How the GPII Supports Inclusive Software Development"
in *Proceedings of the HCII, 2014*

Declarative Programming

"Lisp has no syntax… You write code in parse trees… [that] are fully accessible to your programs. You can write programs that manipulate them… programs that write programs."

Paul Graham, *Beating the Averages*, `http://paulgraham.com/avg.html` (2001)

Flocking is Declarative

- You write *data*, not code to design an instrument or define a score
- Unit generators provide a consistent abstraction for operations on signals
- Synthesis graphs are built up by declaring trees of named unit generators

Programming in JSON

JavaScript Object Notation, a standard format for data exchange on the web.

```
{
    “key”: "value",
    “meaning”: 42.42,
    “isLoud”: true
}

[“tenney”, “risset”, “schmickler”]
```

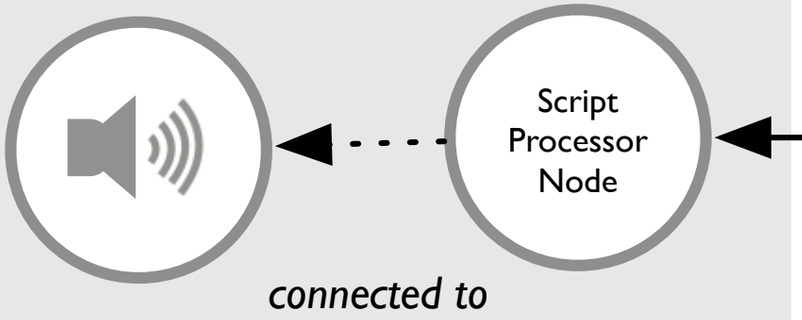
```
{
    "synthDef": {
          "ugen": "flock.ugen.granulator",
          "numGrains": {
              "ugen": "flock.ugen.line",
              "start": 1,
              "end": 40,
              "duration": 20
          },
          "grainDur": {
              "ugen": "flock.ugen.line",
              "start": 0.1,
              "end": 0.005,
              "duration": 100
          },
```

How it Works

Primary Components

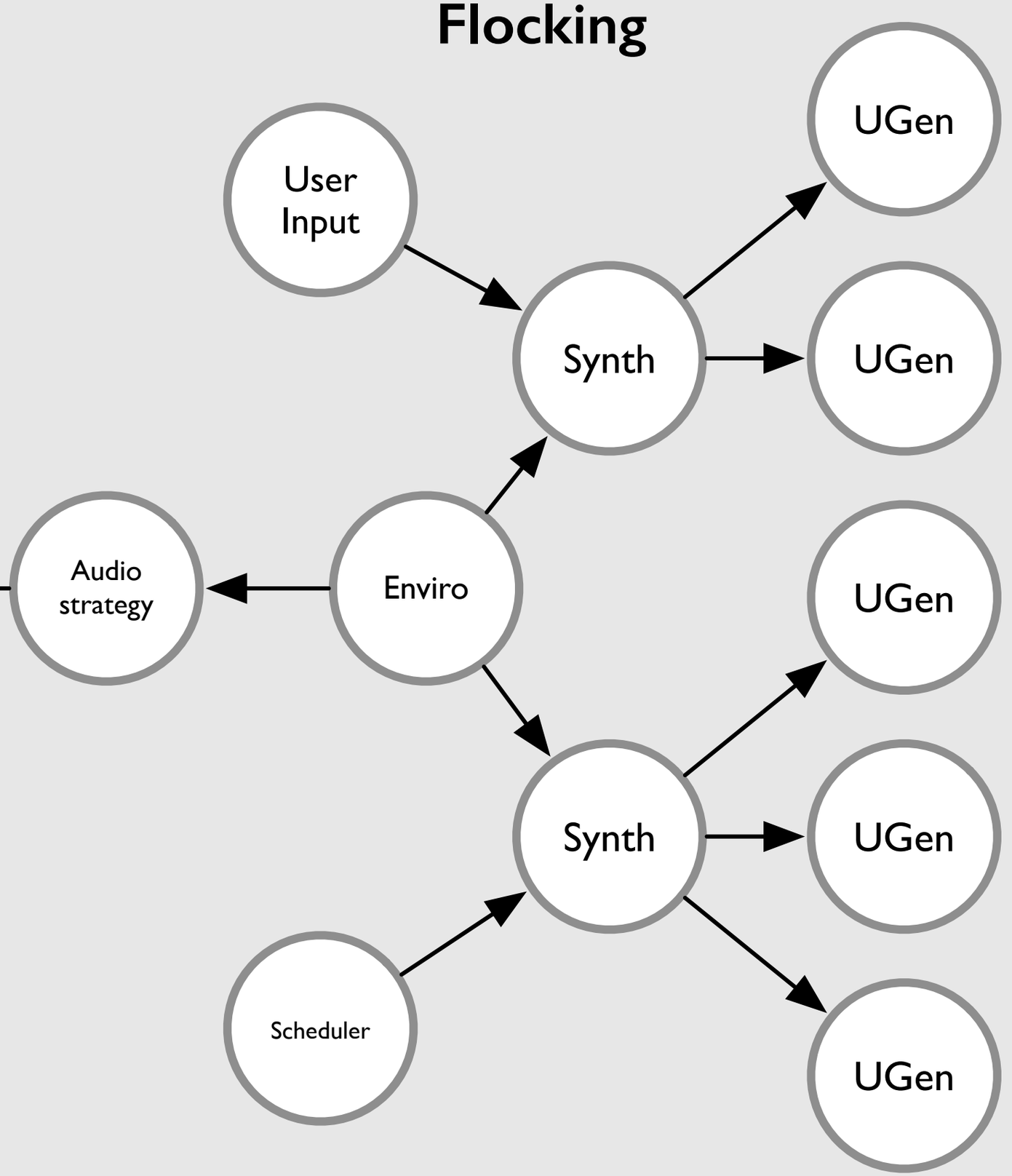
- Framework: parses, traverses and instantiates synths, unit generators, and buffers from JSON
- Environment: overall audio system with pluggable back-ends
- Unit generators: familiar sample-generating primitives
- Synths: named collections of signal generators
- Scheduler: schedules *value synths* in time

Web Audio API



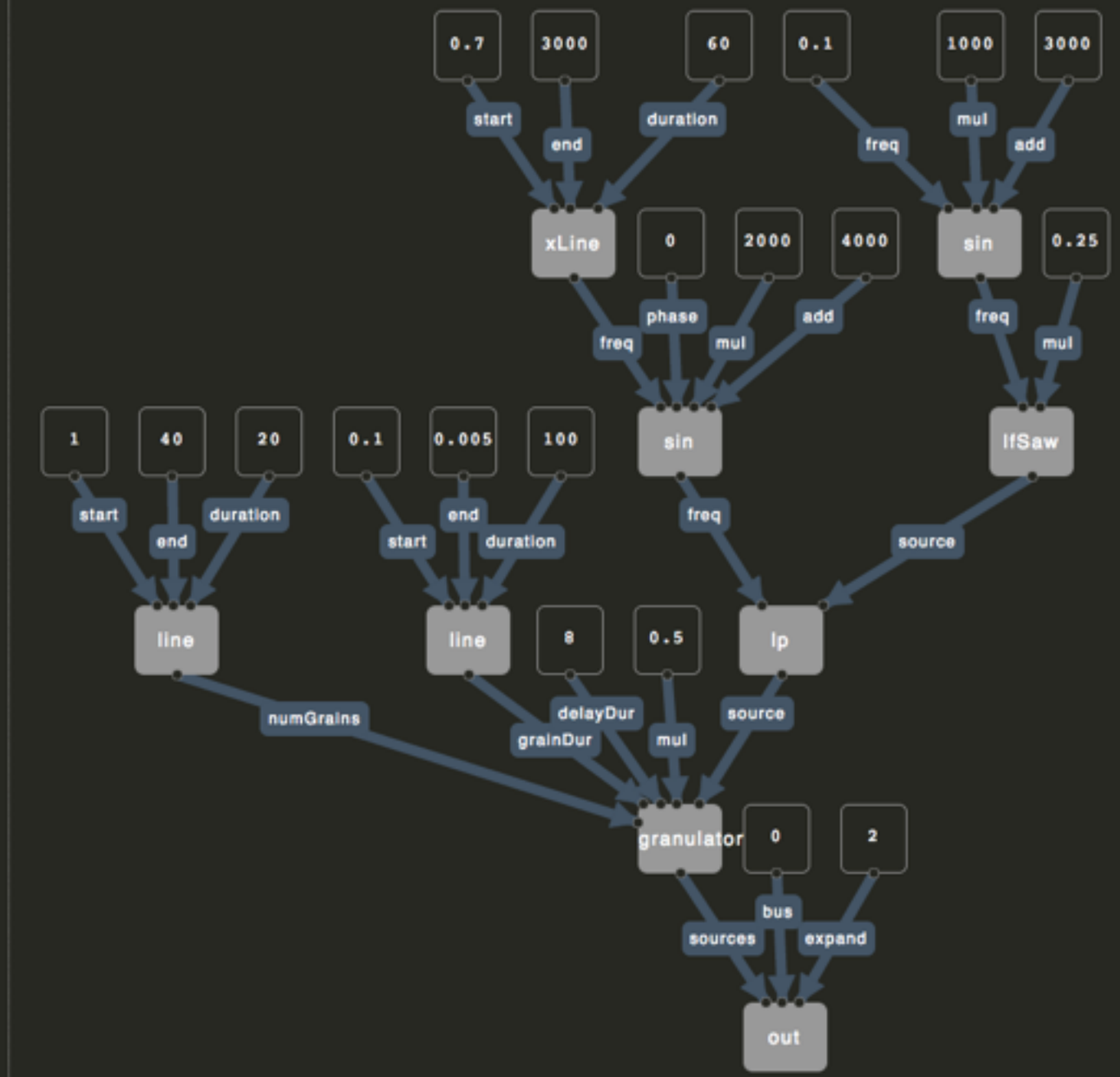
connected to

Flocking



Demo: Granulator

```
{
    "description": "Granulates a filtered sawtooth wave. Demo by Mayank Sanganeria
    "synthDef": {
        "ugen": "flock.ugen.granulator",
        "numGrains": {
            "ugen": "flock.ugen.line",
            "start": 1,
            "end": 40,
            "duration": 20
        },
        "grainDur": {
            "ugen": "flock.ugen.line",
            "start": 0.1,
            "end": 0.005,
            "duration": 100
        },
        "delayDur": 8,
        "mul": 0.5,
        "source": {
            "ugen": "flock.ugen.filter.biquad.lp",
            "freq": {
                "ugen": "flock.ugen.sin",
                "rate": "control",
                "freq": {
                    "ugen": "flock.ugen.xLine",
                    "rate": "control",
                    "start": 0.7,
                    "end": 3000,
                    "duration": 60
                },
                "phase": 0,
                "mul": 2000,
                "add": 4000
            },
            "source": {
                "ugen": "flock.ugen.lfSaw",
                "freq": {
                    "ugen": "flock.ugen.sin",
                    "freq": 0.1,
                    "mul": 1000,
                    "add": 3000
                },
                "mul": 0.25
            }
        }
    }
}
```



Unit Generators

```
{
    ugen: "flock.ugen.sinOsc",
    freq: 440,
    mul: 0.25
}
```

Global Names

```
{
    ugen: "flock.ugen.sinOsc",
    freq: 440,
    mul: 0.25
}
```

Inputs

```
{
    ugen: "flock.ugen.sinOsc",
    freq: 440,
    mul: 0.25
}
```

Rates

```
{
    ugen: "flock.ugen.sinOsc",
    rate: "control",
    freq: 440,
    mul: 0.25
}
```

Named unit generators

```
{
    id: "carrier",
    ugen: "flock.ugen.sinOsc",
    rate: "control",
    freq: 440,
    mul: 0.25
}
```

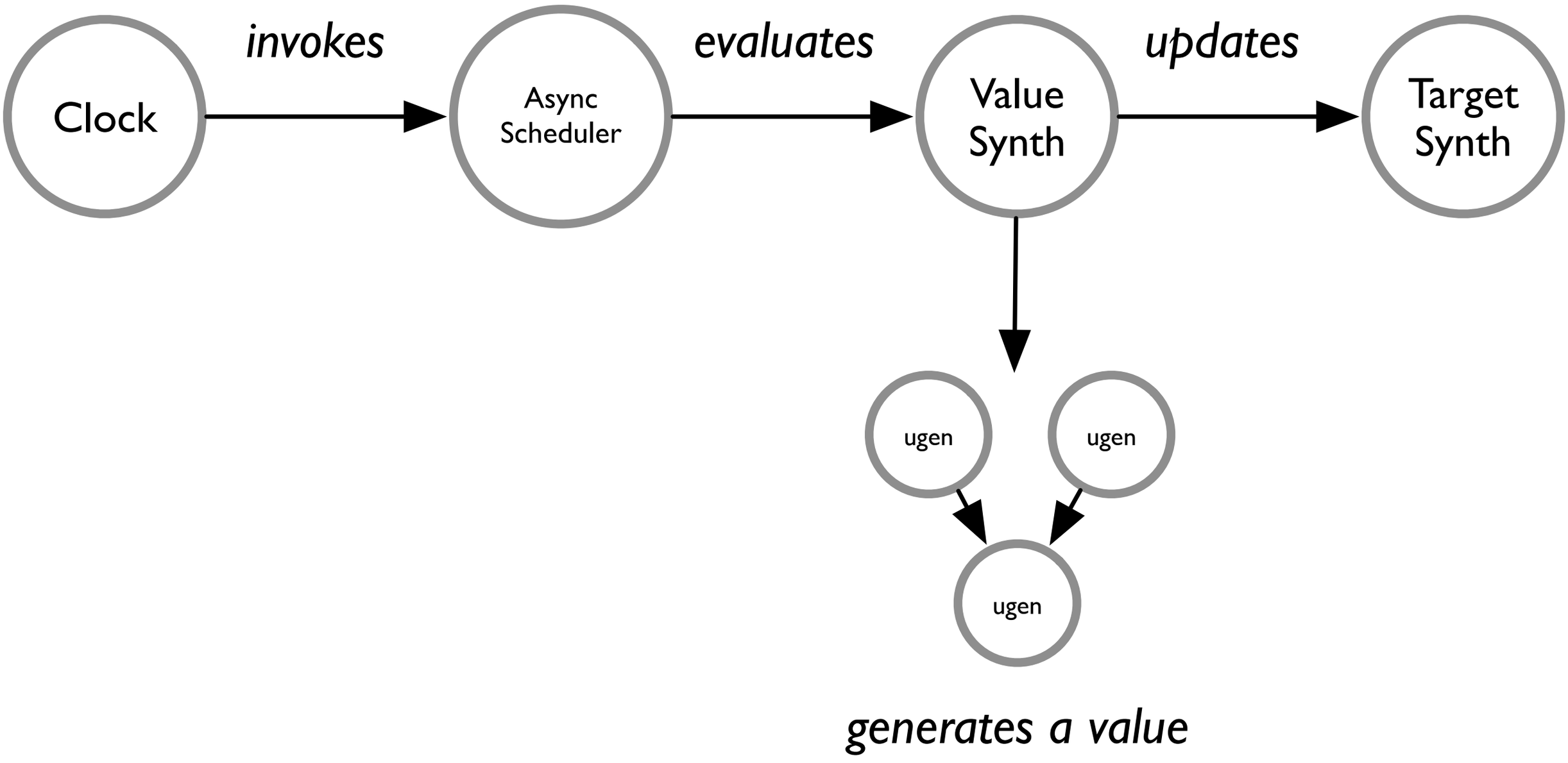
Change Specifications

```
amSynth.set({
    "carrier.freq": 440,
    "modulator.mul": {
        "ugen": "flock.ugen.line",
        "start": 0.25,
        "end": 0.0,
        "duration": 2.0
    },
    "modulator.add": 0.0
});
```

```
{
    "type": "flock.scheduler.async",
    "options": {
        "components": {
            "synthContext": "{amSynth}"
        },
        "score": [
            {
                "interval": "repeat",
                "time": 1.0,
                "change": {
                   "values": {
                      "carrier.freq": {
                          "synthDef": {
                               "ugen": "flock.ugen.sequence",
                               "list": [330, 440, 550, 660, 880, 990]
                          }
                      }
                   }
                }
            }
        ]
    }
}
```

Scheduling Changes

The Scheduler



```
"quneo": {
    "type": "flock.midi.controller",
    "options": {
        "components": {
            "synthContext": "{amSynth}"
        },
        "controlMap": {
            "0": {
                "input": “carrier.freq",
                "transform": {
                    "mul": 10,
                    "add": 120
                }
            },
            "1": {
                "input": "carrier.mul",
                "transform": {
                    "mul": 0.00787
                }
            }
        }
    }
}
```

Binding MIDI

```
"quneo": {
    "type": "flock.midi.controller",
    "options": {
        "components": {
            "synthContext": "{amSynth}"
        },
        "controlMap": {
            "0": {
                "input": “carrier.freq",
                "transform": {
                    "mul": 10,
                    "add": 120
                }
            },
            "1": {
                "input": "carrier.mul",
                "transform": {
                    "mul": 0.00787
                }
            }
        }
    }
}
```

Binding MIDI

```
"quneo": {
    "type": "flock.midi.controller",
    "options": {
        "components": {
            "synthContext": "{amSynth}"
        },
        "controlMap": {
            "0": {
                "input": “carrier.freq",
                "transform": {
                    "ugen": “flock.ugen.value”,
                    "mul": 10,
                    "add": 120
                }
            },
            "1": {
                "input": "carrier.mul",
                "transform": {
                    "mul": 0.00787
                }
            }
        }
    }
```

Binding MIDI

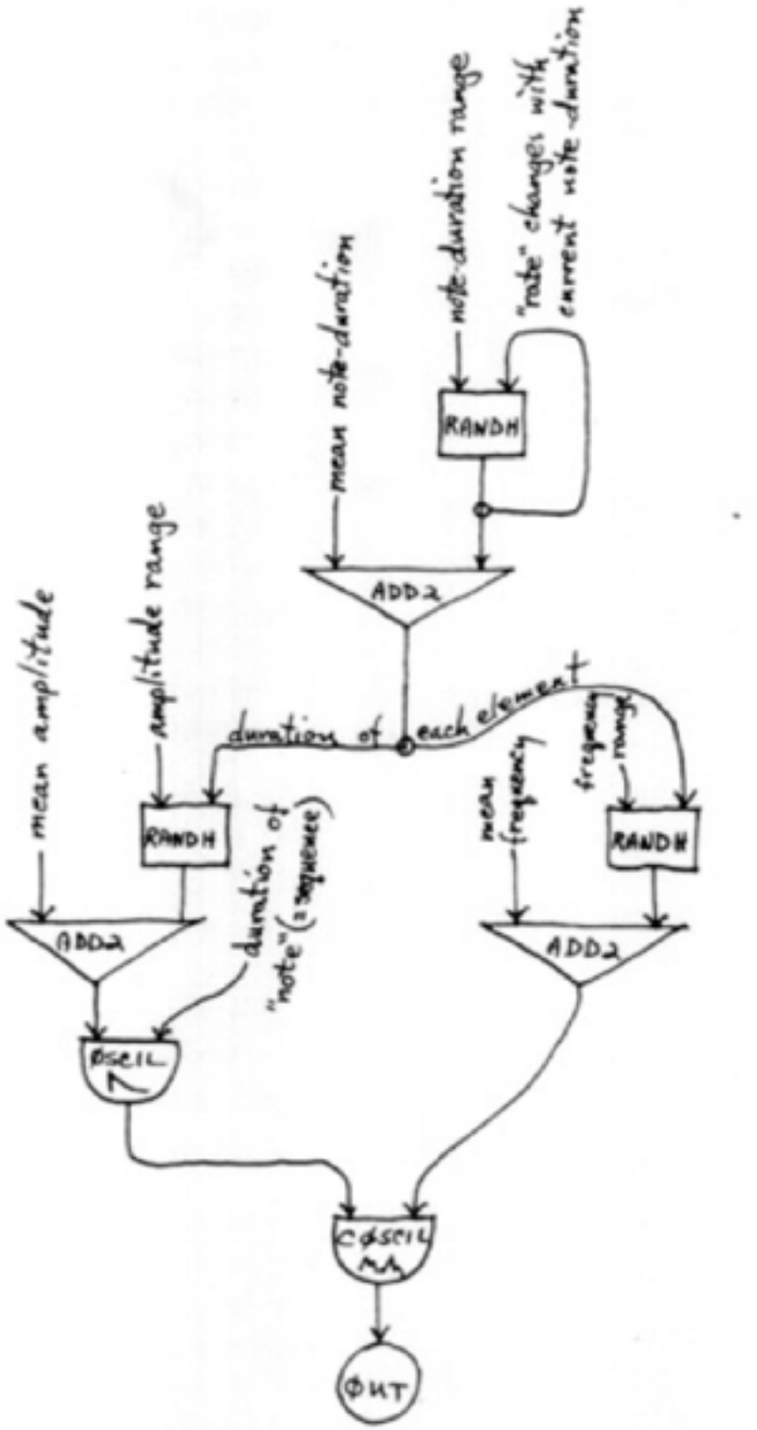


Figure 8. Instrument for generating random sequences.

Tenney, James. "Computer Music Experiences, 1961-1964" in *Electronic Music Reports* (1), 23-60, 1969.

Document Oriented Programming

- JSON specifications are interpreted and instantiated into components by a framework (Flocking and Fluid Infusion)
- Reuse by merging: A document can be overlaid on top of another, adding, changing, or eliding values
- State, event bindings, and behaviour are all configurable

Names as Landmarks

```
fluid.defaults("colin.amSynth", {
    gradeNames: ["flock.synth"],
    synthDef: {
        ugen: "flock.ugen.sinOsc",
        freq: 440,
        mul: {
            ugen: "flock.ugen.triOsc",
            freq: 5,
            mul: 0.25,
            add: 0.25
        }
    }
});
```

Extension by Overlay

```
fluid.defaults("your.betterAMSynth", {
    gradeNames: [“colin.amSynth”],
    synthDef: {
        mul: {
            ugen: "flock.ugen.sinOsc"
        }
    }
});
```

Principle of Least Power:

"Nowadays we have to appreciate the reasons for picking not the most powerful solution but the least powerful. The reason for this is that the less powerful the language, the more you can do with the data stored in that language. If you write it in a simple declarative form, anyone can write a program to analyze it in many ways."

Tim Berners-Lee, *Axioms of Web Architecture*, 1998

`http://www.w3.org/DesignIssues/Principles.html#PLP`

Web Audio: Challenges

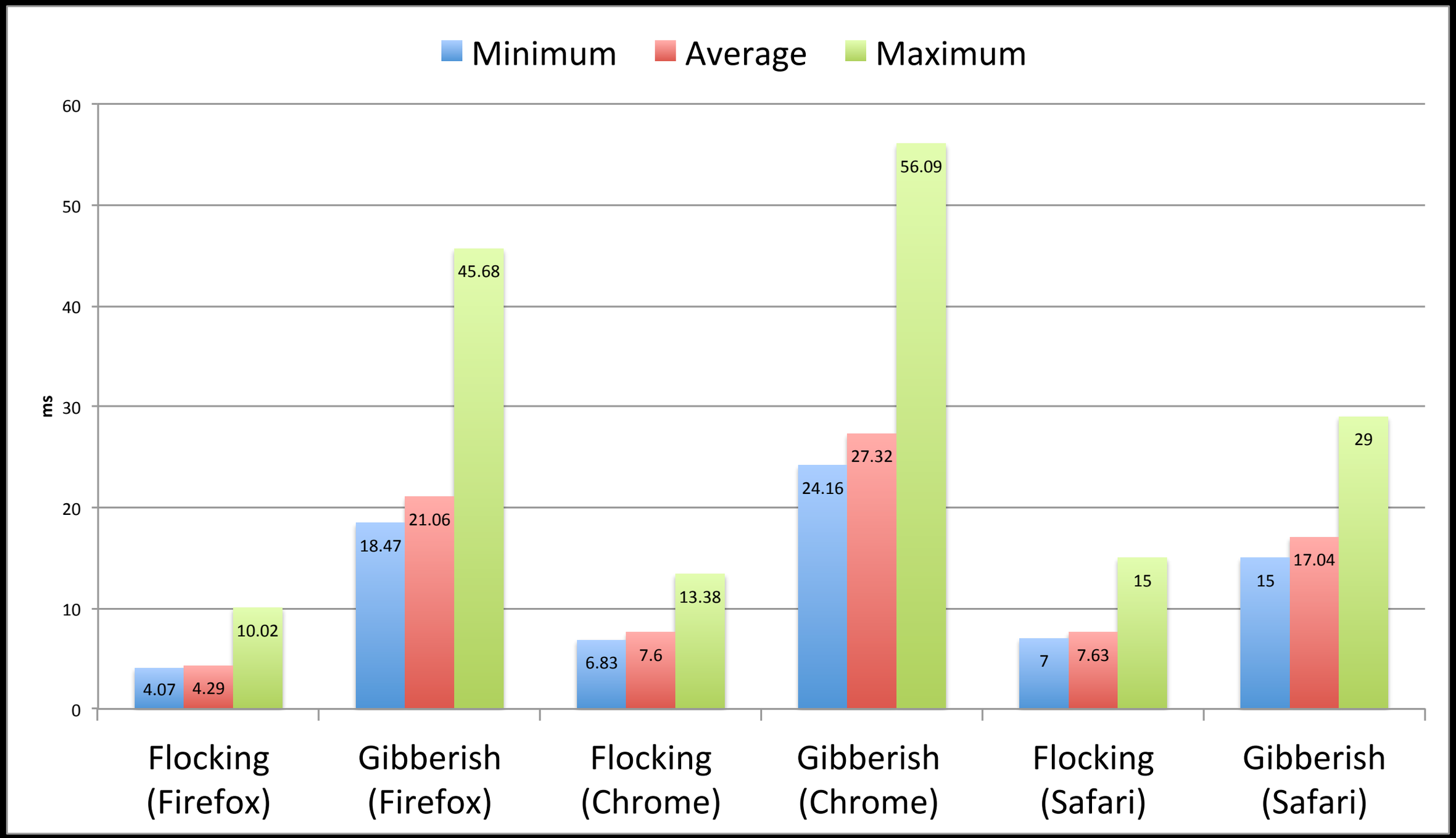
Performance

- A lot of myths out there about JavaScript
- Browser wars mean JS is now the fastest dynamic language today (JIT compilation)
- In “The Web Browser As Synthesizer And Interface” (NIME 2013), Roberts et al use micro benchmarks and complex code generation scheme in Gibberish
- But JIT compilers thrive on hot, type stable code!

Flocking's Approach

- Keep it simple: Flocking stores unit generators in an ordered list; on each tick, iterate and evaluate
- Called 700x per second (in 64 sample blocks), this represents a hot, stable path for JIT compilation
- Avoid premature optimization

Comparative Benchmarks



Gibberish Amplitude modulation demo. Smaller bars are faster.

Performance Findings

- Making architectural trade-offs for performance optimization prematurely is a risk
- Stable, invariant code will be compiled. Don't spill the JIT's cache with eval() or type changes.
- Avoid generating garbage wherever possible
- Use a block-based architecture

Future R&D

Live Data Merging

```
{
    "filterSynth": {
        "gaussian": {
            "sigma": 100,
            "mu": 20
        },
        "cutoff": {
            "freq": {
                "freq": 1.5
            },
            "mul": 600
        },
        "resonance": 4.5
    },
    "noiseSynth": {
        "envelope": {
            "freq": {
                "freq": 0.02,
                "add": 0.5
            },
            "mul": {
                "freq": 1.75
            }

        }
    }
}
```

15:52:29.202: more flock, less talk


FLOCKING
What is Flocking?
Resources
acer

Questions?

Colin Clark
Inclusive Design Research Centre,
OCAD University, Toronto

e: cclark@ocadu.ca
t: @colinbdclark

flockingjs.org
github.com/colinbdclark/flocking

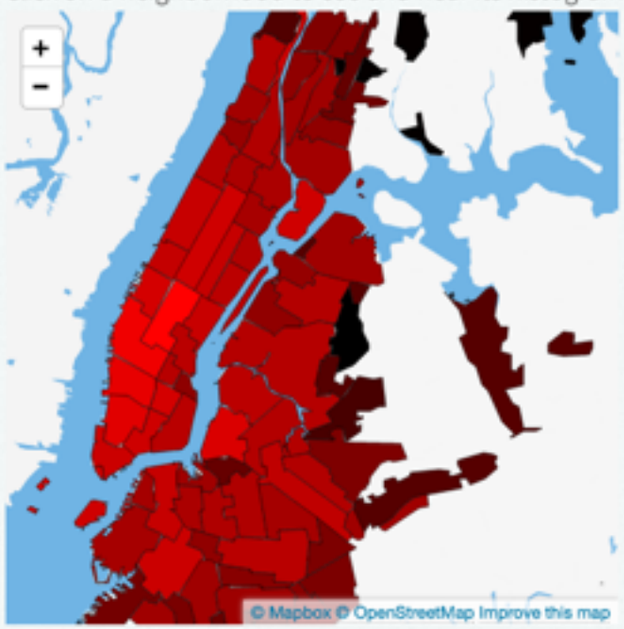
moogy

The sound of a neighborhood

Instagram reports that its 150M monthly active users post 55M times per day, worldwide.
The dataset we worked with contained Instagram posts in New York City, inclusive of several of the most "Instagramed" spots in the world (like Times Square, Central Park, and the High Line). We wanted to explore the usage patterns of the 101,775 users represented in our data, who posted to Instagram from New York locations over 300,000 times during the first week of October 2013.

After we did some initial explorations of the full dataset (see below), we noticed certain high-volume neighborhoods like Midtown, daytime-intensive spots like New York City parks, and late-night activity in Williamsburg and the Lower East Side. We began to imagine: What does an average day of Instagram posts sound like in each New York neighborhood? if you mapped locations or users to frequencies?

Click on a neighborhood to see and hear its histogram



These histograms, laid out like 24-hour clocks, represent Instagram post volume as it changes over an average day in a particular neighborhood. In Midtown, for instance, there is a much larger volume of posts during the work day. Parks have spikes in the daylight hours. The sound that plays along with each