

# WORKSHOP: PROGRAMMING MUSIC AND SYNTHEZIZERS ON-THE-FLY WITH PHARO

DOMENICO CIPRIANI



ADC<sub>25</sub>  
*Bristol*

# (temporary) AGENDA

1. Foundations [ 45 minutes ]
2. Rhythm and Melodies [ 75 minutes ]
3. Sound Design with Phausto [ 75 minutes ]
4. Performance [ 20 minutes ]
5. Visual Interface and Control [ 25 minutes ]



# GOALS

- 1. Become friends with live-coding**
- 2. Meet and install Pharo, Coypu and Phausto**
- 3. Create rhythm and melodies with Coypu**
- 4. Create a small set of synthesisers with Phausto and perform with them.**
- 5. Design a basic User Interface with Bloc/CoypulDE**

# DISCLAIMER

*Coypu and Phausto are still in an alpha stage, so we may encounter a few quirks (and maybe occasional crashes) .*

*Thanks for your patience and sense of adventure!*

*We're constantly improving them and can't wait to see what you think. You're among the bold early explorers using tools that many others will only discover many years from now 🚀.*

*As you dive into Pharo, we hope that exploring a new syntax and system of tools will feel like an enlightening and chill programming experience*



# About me

Domenico Cipriani  
a.k.a. Lucretio (the Analogue Cops / Rage Therapy)

**Dj/live performer/live coder/electronic music producer**

- (Tresor, Berghain, Fabric, Rex, Concrete, Faust, LoveParade 2006, Dimensions Festival, Twisted Pepper..Barcelona, Frankfurt, Rome, Hanoi, San Francisco, Detroit, Belgrade, Tokyo, Osaka, Kiev...)
- More than 150 vinyl releases including records with Blawan, Objekt ...
- Performed at ICLC24 and ICLC25 (+workshop).

**Currently researching in computer music for the Evref team of Inria**

**Educator and Artist Partner for Soundwave, Italian distribution of Elektron machines**

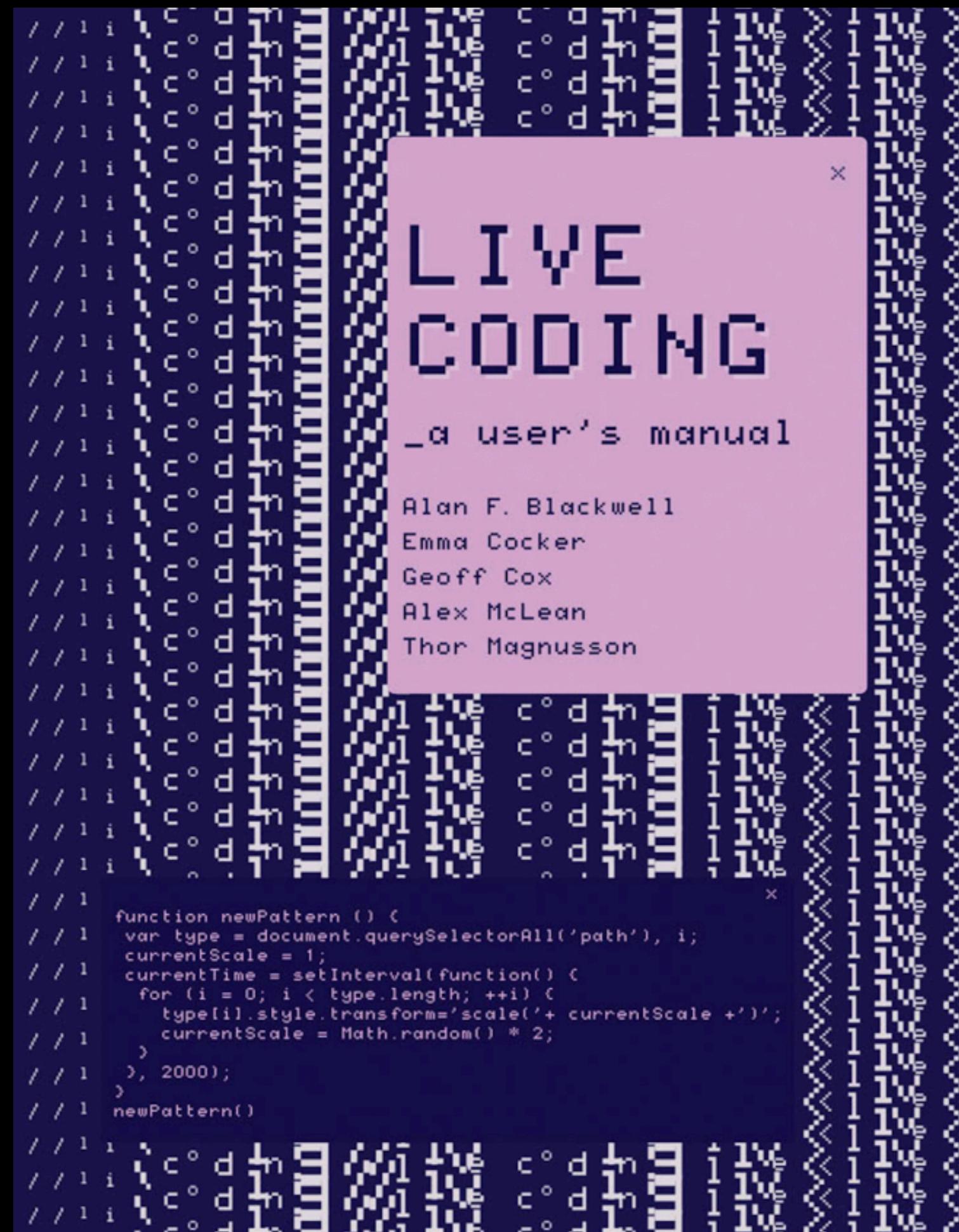
# PART 1: FOUNDATIONS



Inria



# What is live coding?



- Live coding is about people interacting with the world, and each other, in real time, via code.
  - Live coding is about making software live.
  - Live coding is a performance practice that operates as an adventure and exploration, deliberately rejecting fixed definitions, remaining heterogeneous in nature.
  - With no formal definition (or at least only one that includes the possibility of its own redefinition) also comes resistance to hierarchical control, live coding cannot really become owned by established practitioners or institutions

# What is an algorave?

- An **Algorave** is an event where people dance to music generated by algorithms, often performed through live coding
- Born in London in 2011 from the minds of **Alex McLean** and **Nick Collins**, it has since evolved into a worldwide community celebrating the art of coding sound and visuals.
- Code should be projected to the audience.
- Any **Algorithmic Music** is welcome when it is

*“wholly or predominantly characterised by the emission of a succession of repetitive conditionals”*

# Thinking in many tongues

just to name a few

<b>TidalCycles</b>	Haskell-based language for pattern-based algorithmic music.
<b>Chuck</b>	Strongly timed language for on-the-fly sound synthesis and composition.
<b>Sonic Pi</b>	Accessible Ruby environment for live-coded music and education.
<b>ORCA</b>	Esoteric, grid-based environment for composing and sequencing live.
<b>Strudel</b>	Web version of TidalCycles for browser-based live coding.
<b>FoxDot</b>	Python live coding interface for SuperCollider.
<b>Overtone</b>	Clojure environment for synthesis, sampling, and collaborative music.
<b>Mercury</b>	Minimal language for web browser or MaxMSP
<b>Extempore</b>	Lisp-like language for real-time programming of sound and interaction.
<b>Hydra</b>	Browser-based system for coding visuals and video synthesis live.



# Pharo, Coypu, Phausto

**PHARO**

Free, open-source, general-purpose  
language + cross-platform IDE

**COYPU**

API and Domain Specific Language for  
programming music on-the-fly

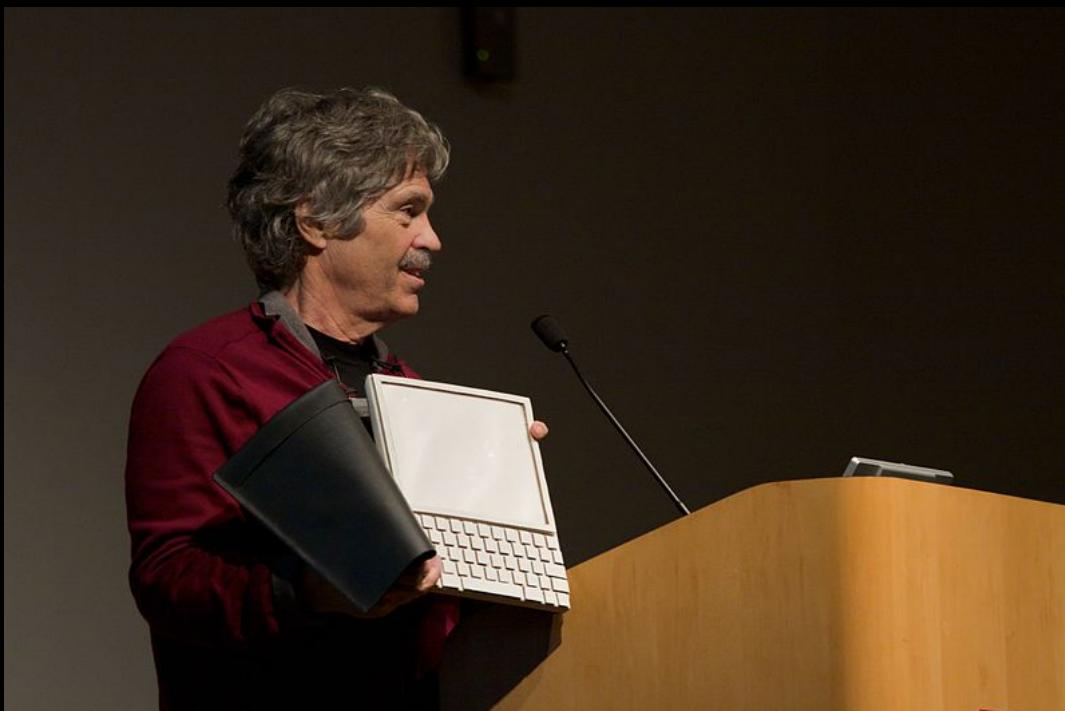
**PHAUSTO**

Library and API for DSP programming  
that enables sound generation within  
Pharo



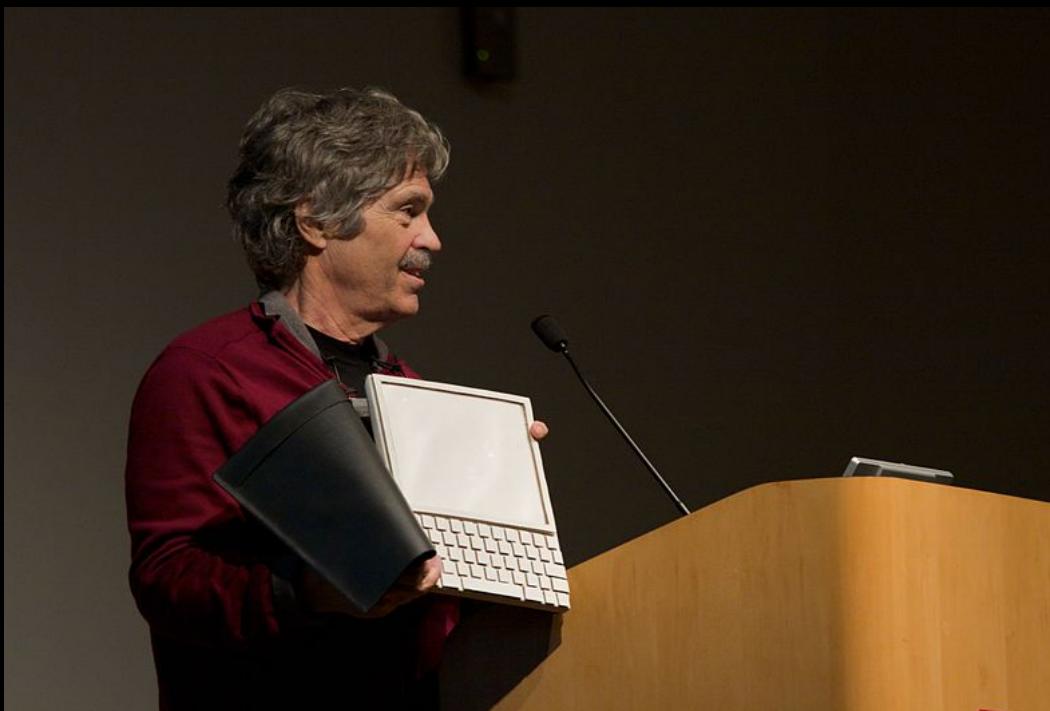
# Smalltalk

- A pure **Object-Oriented** Programming language developed at the *Learning Research Group* at Xerox PARC in the 1970s by Alan **Kay**, Dan **Ingalls**, Adele **Goldberg**, Ted **Kaehler**.
- Designed for educational use following principles of *Constructionism*.
- Deeply influenced by **Simula**, developed by Ole-Johan **Dahl** and Kristen **Nygaard** in the 1960s at the Norwegian Computing Center in Oslo.
- Not just a language, also an IDE users can inspect and modify.
- Programs written in Smalltalk are compiled into byte code and interpreted by a virtual machine.
- Smalltalk has influenced *Objective C*, *Ruby*, *SuperCollider*.



# Pharo

- A pure **Object-Oriented** Programming language developed at the *Learning Research Group* at Xerox PARC in the 1970s by Alan **Kay**, Dan **Ingalls**, Adele **Goldberg**, Ted **Kaehler**.
- Designed for educational use following principles of *Constructionism*.
- Deeply influenced by **Simula**, developed by Ole-Johan **Dahl** and Kristen **Nygaard** in the 1960s at the Norwegian Computing Center in Oslo.
- Not just a language, also an IDE users can inspect and modify.
- Programs written in Smalltalk are compiled into byte code and interpreted by a virtual machine.
- Smalltalk has influenced *Objective C*, *Ruby*, *SuperCollider*.

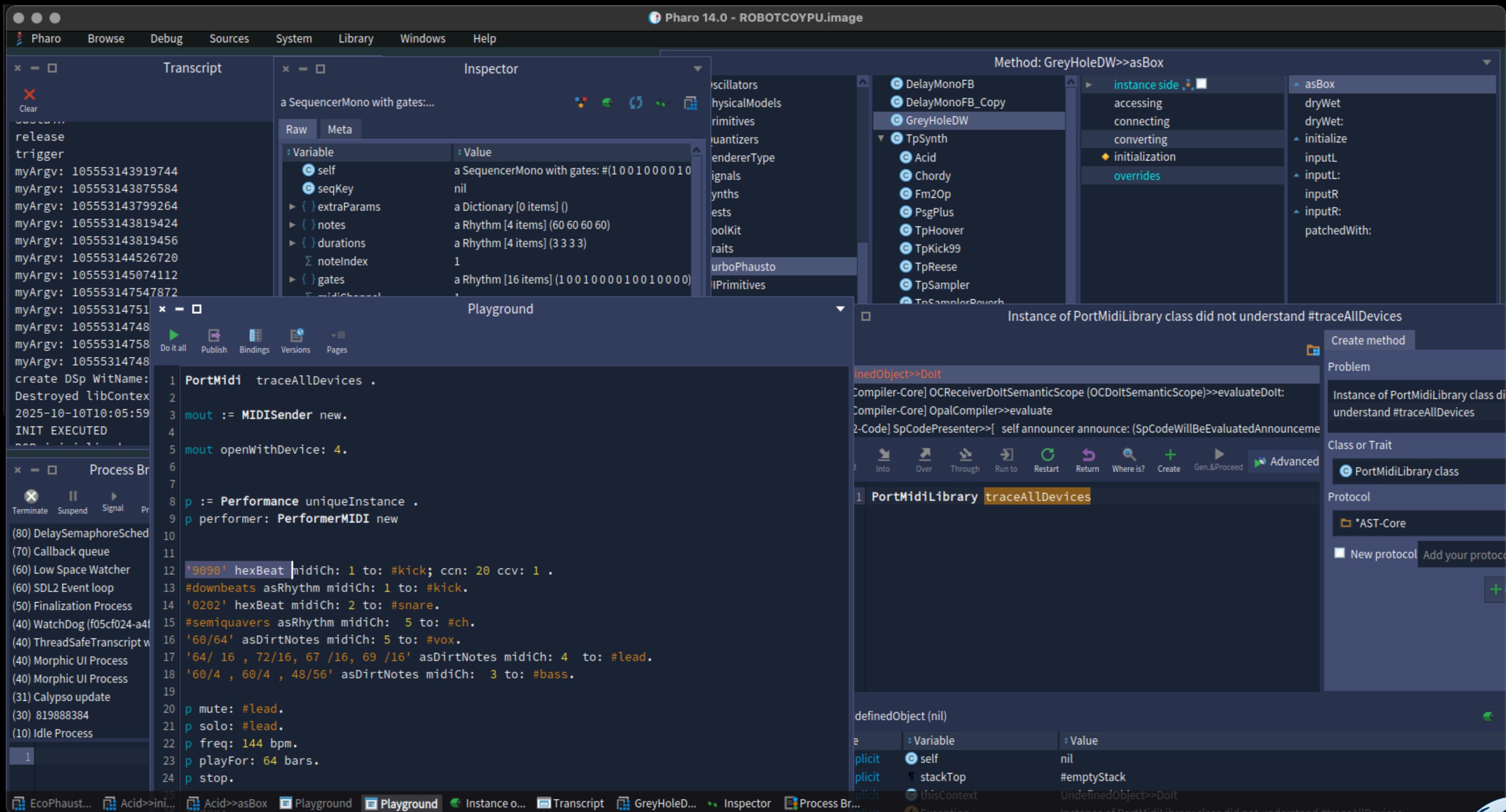


# Pharo



- A pure object-oriented, dynamically typed, and reflective language with a platform independent IDE.
- Its syntax fits in a postcard
- Like Smalltalk, Pharo provides many *live programming* features such as immediate object manipulation, live updates, and just-in-time compilation (JIT).
- Integrated *Git* support and a framework for SUnit Tests.

# The Pharo IDE



# Install Pharo

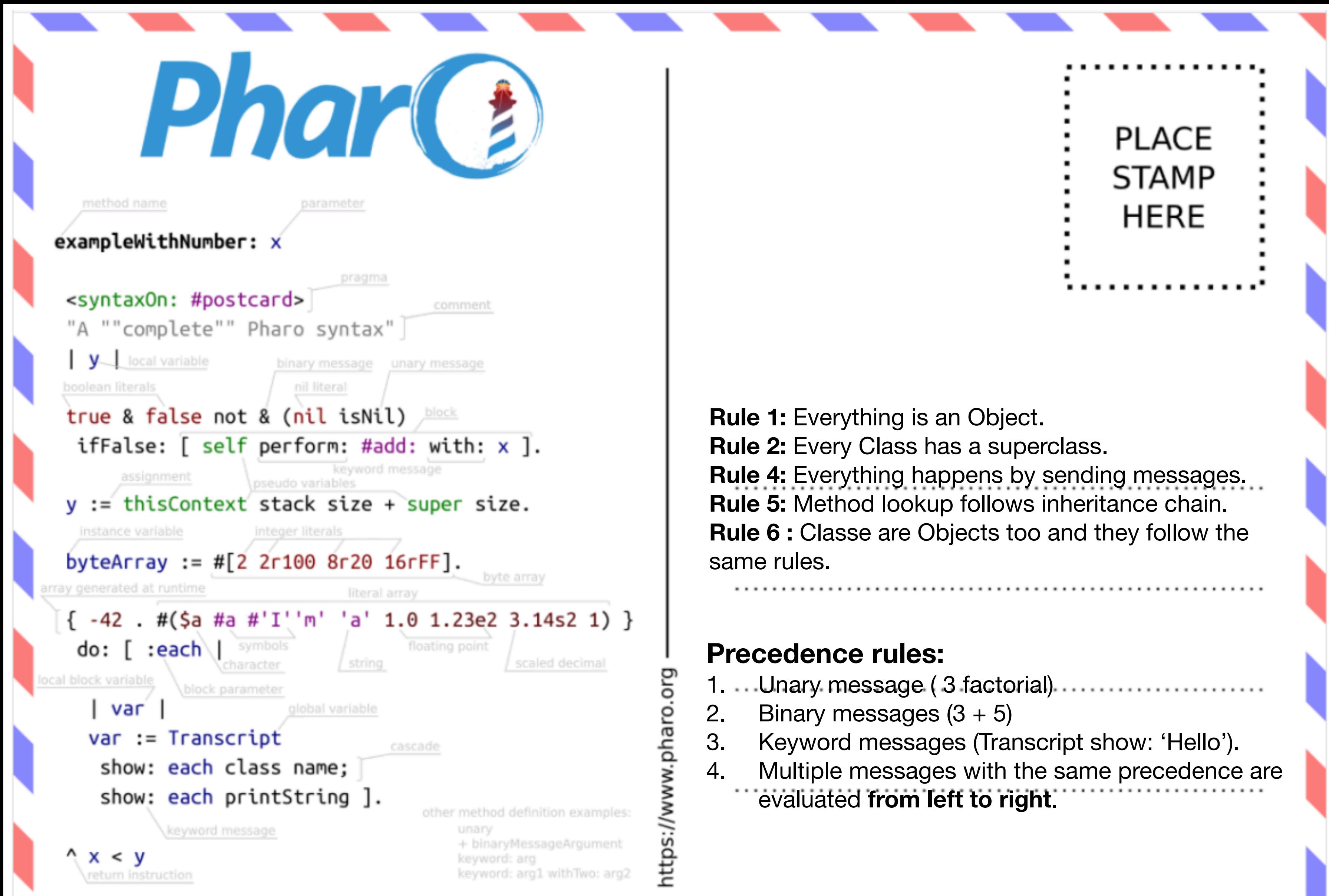
Download the Pharo Launcher to download a Pharo Image:

<https://pharo.org/download>

A Pharo image is an object space + Pharo Core Libraries + the virtual machine



# Pharo syntax on a postcard



# The Pharo IDE / shortcuts

<b>CMD/CTRL + OP</b>	Open Playground
<b>CMD/CTRL + D</b>	Evaluate (Do)
<b>CMD/CTRL + P</b>	Print Object
<b>CMD/CTRL + I</b>	Inspect Object
<b>CMD/CTRL + B</b>	Browse Object
<b>CMD/CTRL + M</b>	List Implementors

# Installing Coypu and Phausto

Go to the *Coypu* GitHub repositories:

<https://github.com/lucretiomsp/Coypu>

Copy the *Metacello* script into your *Playground*.

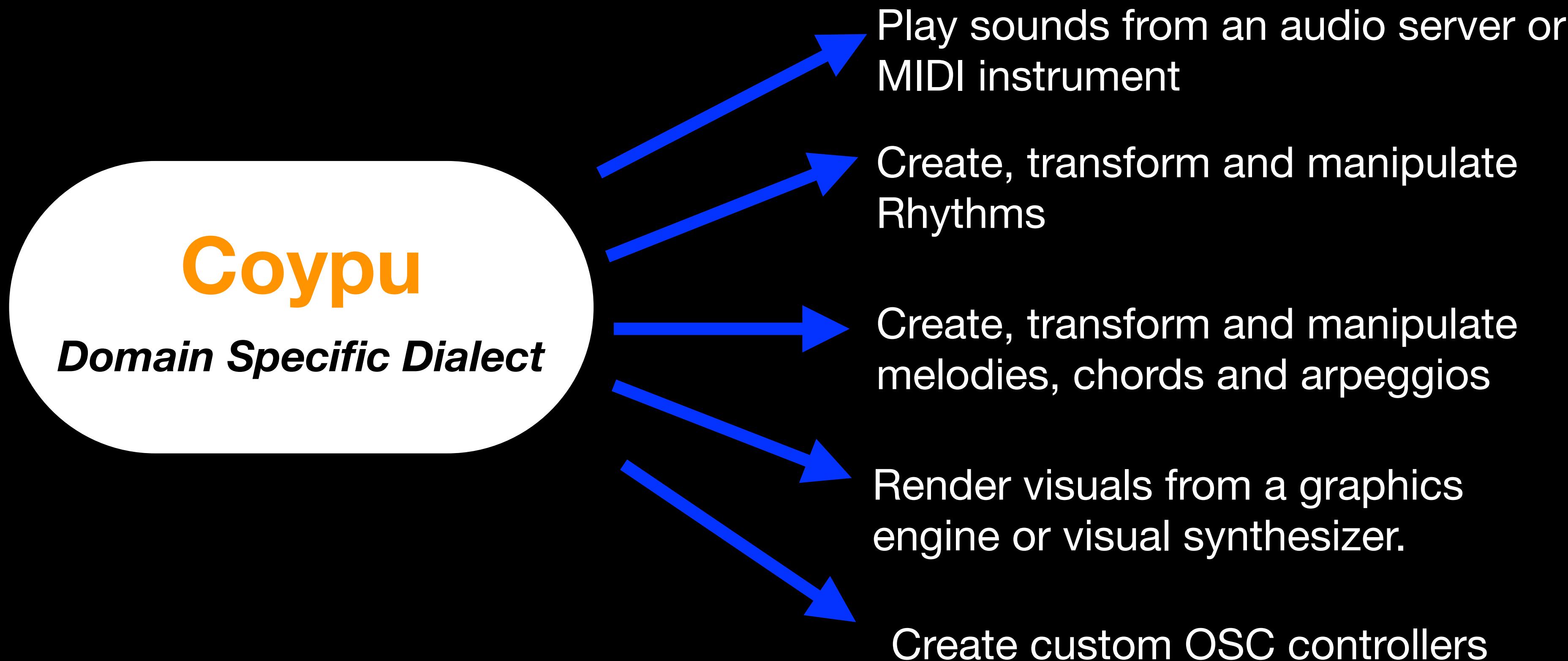
```
Metacello new  
    baseline: 'Coypu';  
    repository: 'github://lucretiomsp/coypu:master';  
    load
```

Select all the text and evaluate (**CMD/CTRL + D**)  
\*[*Coypu* already comes together with *Phausto*]

# PART 1: FOUNDATIONS

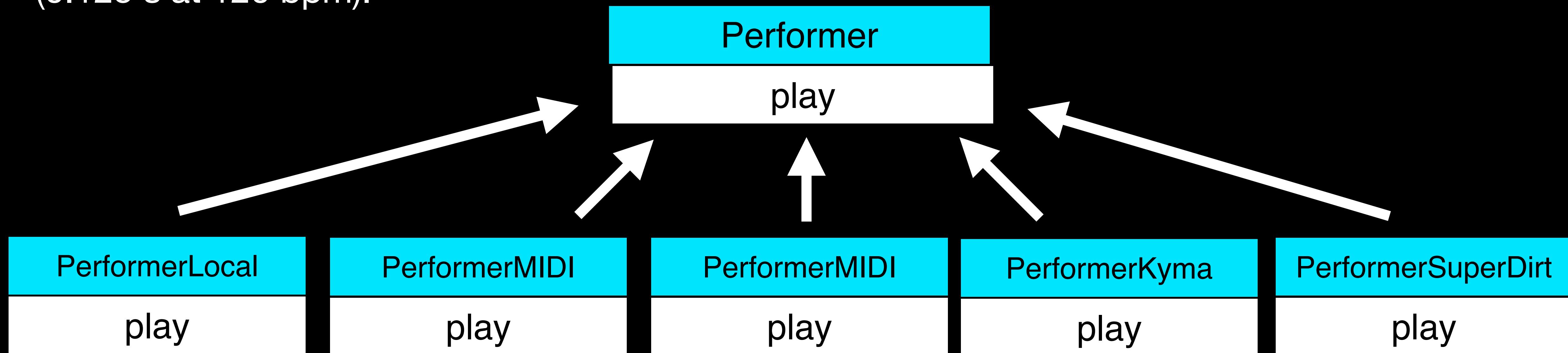


# Coypu's anatomy



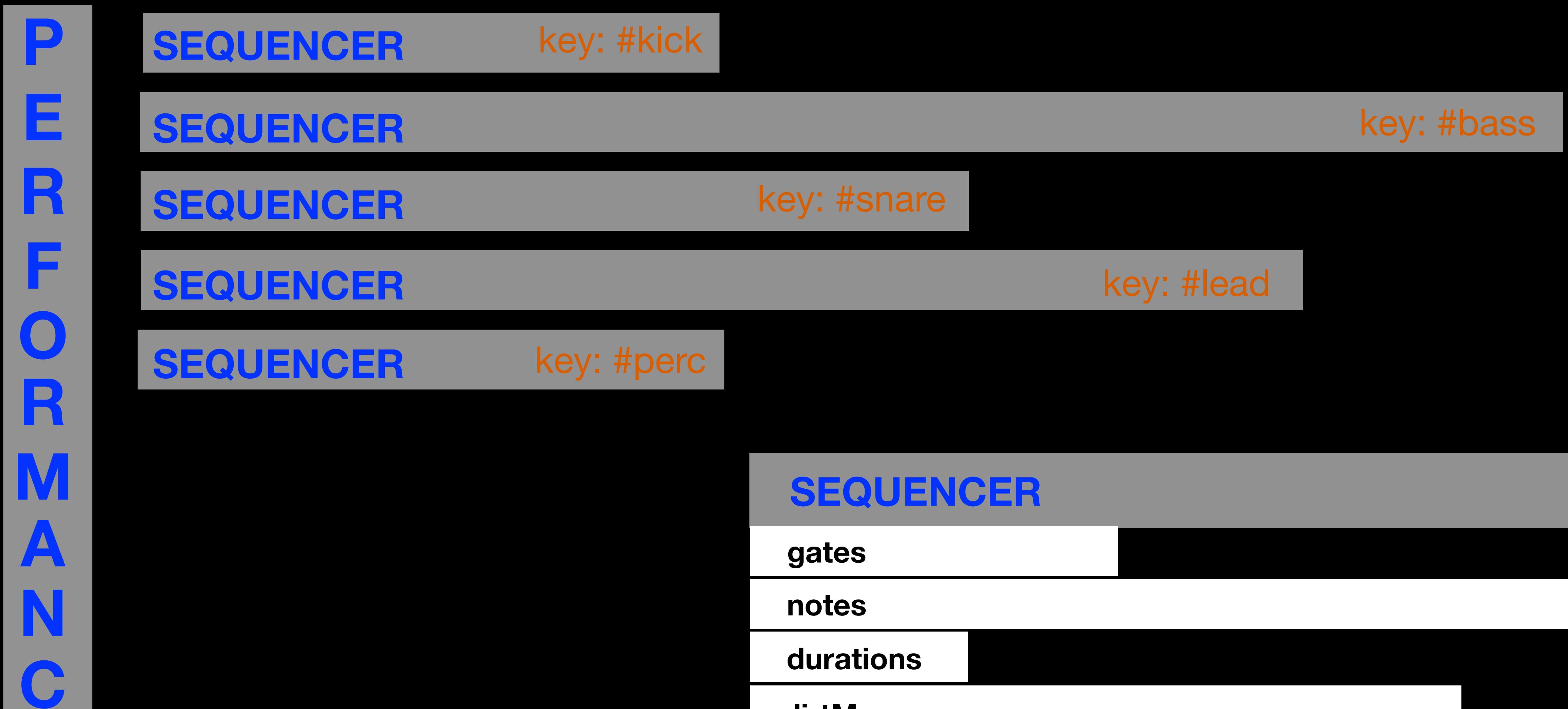
# Coypu's anatomy

- The **Performance** class is a *Singleton*.
- A **Performer** must be assigned to the Performance; the Performer selects the audio backend.
- The *play* method starts the Performance, and increments the *transportStep* every *freq* seconds (0.125 s at 120 bpm).



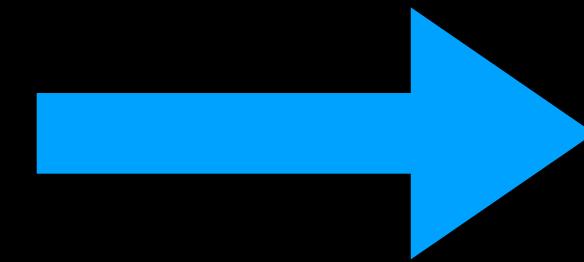
# Coypu's anatomy

*transportStep* increments of 1 every **Performance** uniqueInstance freq (seconds)



# Principles

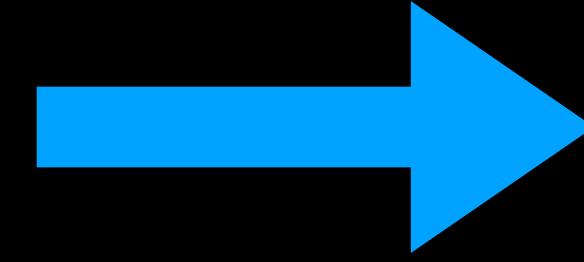
**Iconicity**



**Written code should resemble what we hear**

16 upbeats

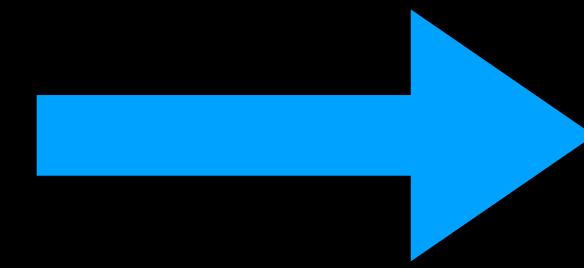
**Economy**



**The less we type, the better**

`#(60 63 67) + 16`

**Polysemy**



**Many ways to do the same thing**

16 downbeats.

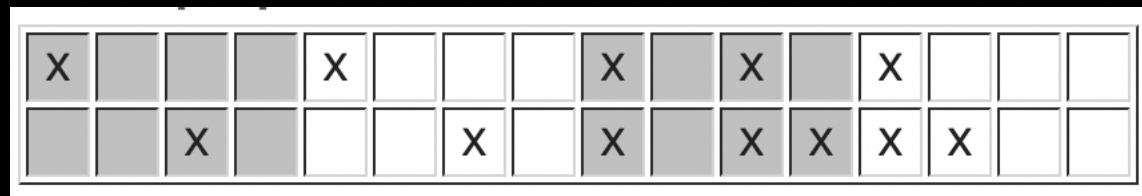
`#downbeats asRhythm.`

`'60 , ~ , ~ , ~ , 60 , ~ , ~ , ~ , 60 , ~ , ~ , ~ , 60 , ~ , ~ , ~'` `asDirtNotes.`  
 `#(1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0) asSeq.`

# Creating rhythms

- Sequencers are inspired by traditional hardware sequencers, where triggers (~noteOn) events are notated in **Time Unit Box System** (TUBS).

TUBS



Traditional



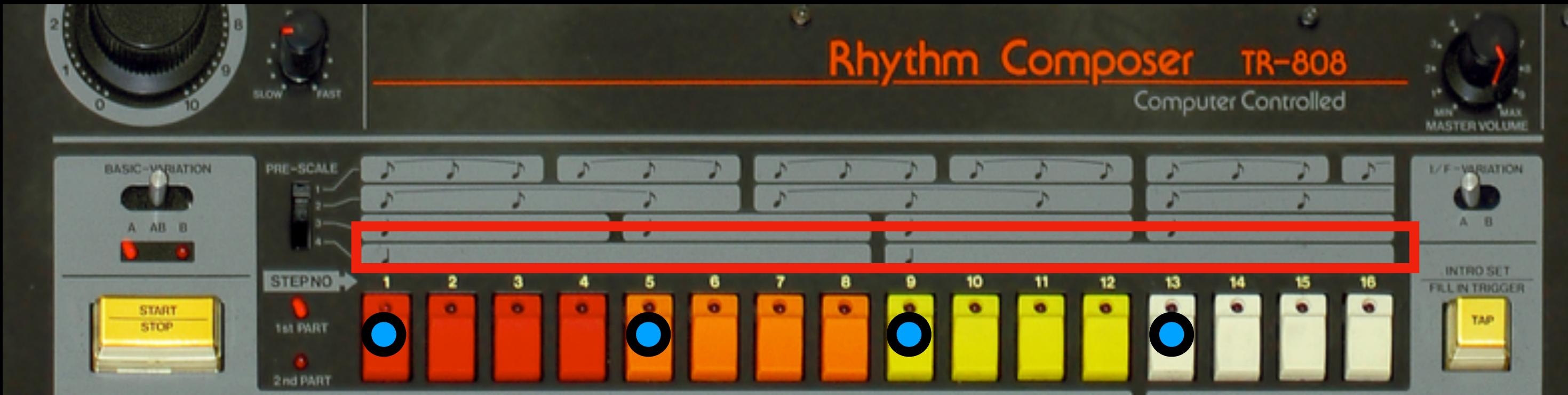
Coypu

```
#( 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 ) asSeq
```

- Performance can be thought as a multitrack player for Sequencers.
- Sequencer are filled with parameters.
- **A Performer must be assigned to the Performance.**

```
1 p := Performance uniqueInstance. "assign the unique instance of the Performer class to a variable"
2 p performer: PerformerLocal new. "assign a local OSC performer to the Performance"
3 p freq: 136 bpm. "change the performances speed"
4 #( 1 0 0 0 1 1 0 0 1 0 ) asSeq notes: '32 37 38' to: #bass. "create a Sequencer and assign it to the performance"
5 p playFor: 64 bars. "play the performance for 64 bars".
6 p stop. "stop the performance"
```

# Creating rhythms



A rhythm can be represented as an array of 0s and 1s, where each 1 represents a trigger.



PHARO

```
#(1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0)
```

COYPU

```
#downbeats asRhythm
```

COYPU

16 downbeats

COYPU

'8888' hexBeat

# Creating rhythms /2

The name of a rhythm  
*(the message)*

16 quavers

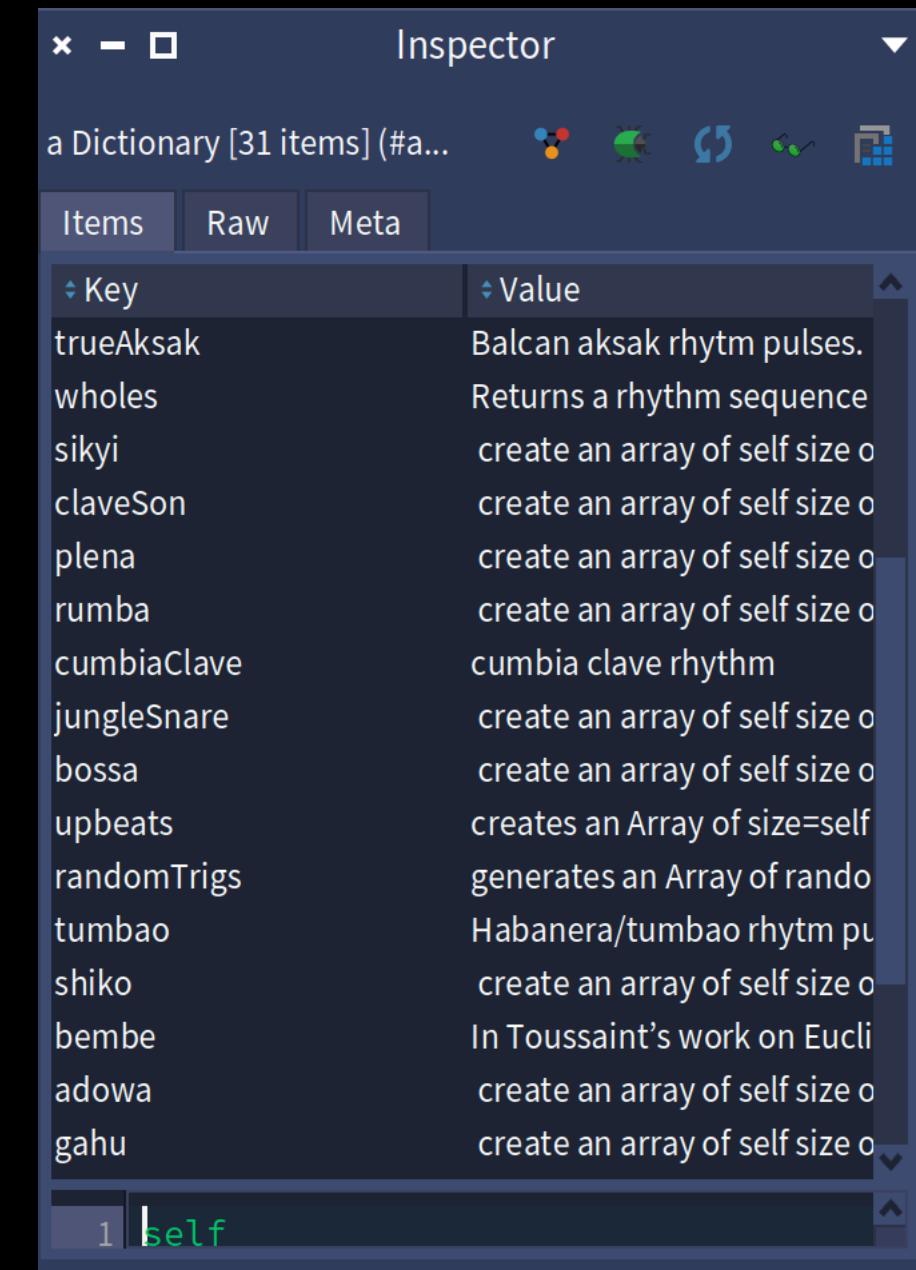
Integer number of steps on the Sequencer  
*(the receiver)*

= #quavers asRhythm

The name of a rhythm preceded by # - becomes a symbol -  
*(the receiver)*

(the message)

Rhythm list inspect

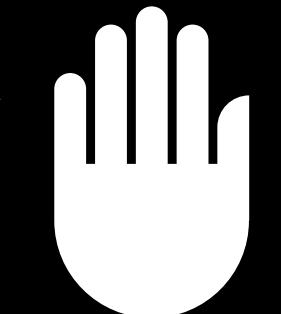


Exceptions!

4 breves.

8 semibreves.

(Here, the integer receiver specifies  
the number of notes with that  
duration.)



# Creating rhythms /3

By default, every note in a sequencer is triggered using MIDI note number 60 (Middle C, or C4). The duration of each note is calculated as *sequencer size / number of trigs*, with a gate time of 0.8.

We can define arrays of note durations and/or gate times. These arrays don't need to have the same length as the number of trigs, which allows us to create polymetric sequencers—sequencers where each parameter can have its own rhythmic cycle.

For example:

16 semiquavers notes: '38, 41 , 43' ; durations: '0.1 , 0.3' ; gateTime: '0.6, 0.5 , 0.4 , 0.1'

# Creating rhythms/4



- #(1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0) asSeq
- #rumba asRhythm
- '9128' hexBeat



- #(1 0 1 0 1 0 1 0 1 0 1 1 1) asSeq
- 12 quavers, 4 semiquavers
- 'AAAF' hexBeat

# Hexbeats

- 4 bits = 1 Nibble / 8 bits = 1 Byte.
- 1 Bar, 16 Steps (1/16th quantisation).
- If every step corresponds to a Bit of Information, in a Bar we find 16 Bits, (2 Bytes).
- In eve)ry Bar there are 4 Beats, so in each Beat there are 4 Bits, i.e. 1 Nibble.
- Every Hexadecimal symbol represents a Nibble (4 steps

0	0	0	0	0	
1	0	0	0	1	Upbeats
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	Downbeats
9	1	0	0	1	
A	1	0	1	0	Quavers
B	1	0	1	1	
C	1	1	0	0	
D	1	1	0	1	
E	1	1	1	0	
F	1	1	1	1	Semiquavers

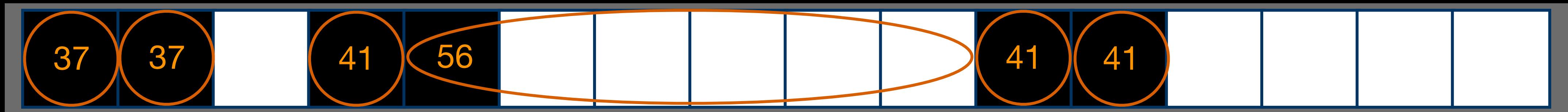
# Random rhythms and melodies

```
16 randomRhythm.  
32 randomTrigs.  
64 randomTrigWithProbability: 70.  
24 plena randomNotesIn: Scale istrian + 46.  
(32 semiquavers randomNotesFrom: #(48 60 63 56)).  
(64 semiquavers randomWalksOn: Scale sakura octaves: 3 root: 48).
```

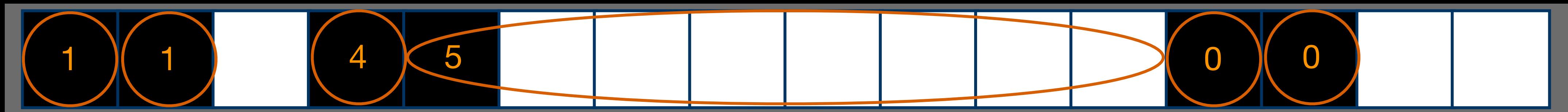
```
gen := MarkovJazzMelodyGenerator new.  
melody := gen generateMelodyFromScaleDegrees: Scale bebop root: 60 startDegree: 0 length: 32.
```

# The ‘musky’ notation

A Tidal Cycles ‘string notation’ to create Sequencers with different notes or different samples



- '37 \* 2 , ~ , 41 , 56 / 8 , 41\* 2 , ~ \* 4 ' asDirtNotes



- '1 \* 2 , ~ , 4 , 5 / 8 , 0\* 2 , ~ \* 2 ' asDirtIndex

# Scales?

**Scale sakura**

**Scale sakura + 60**

**Scale sakura root: 60 octave: 2**

# Sequnecer operations

,

\*

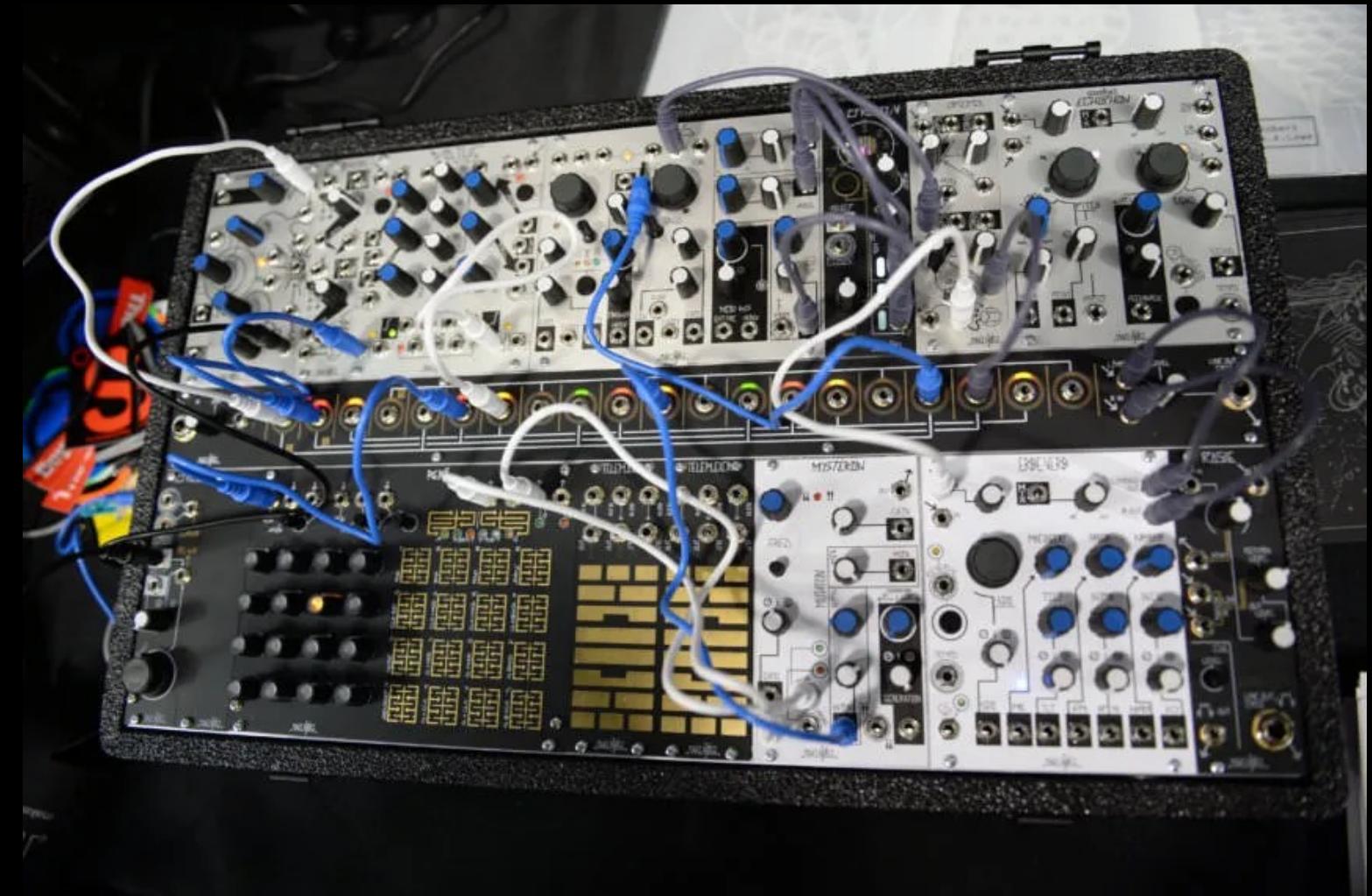
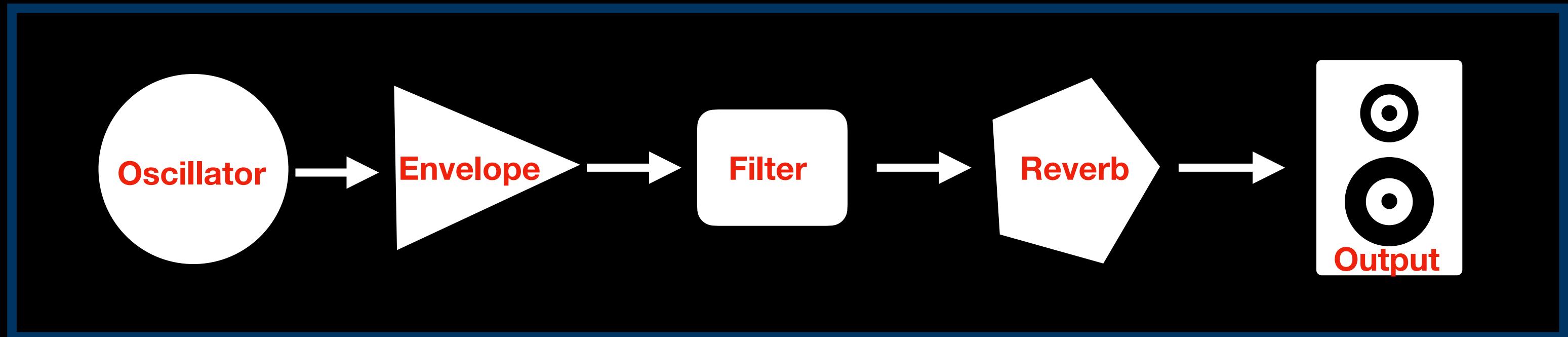
+

# PART 3: SOUND DESIGN WITH PHAUUSTO



# Start your engine

- Phausto is a library and API for sound generation and DSP programming within the Pharo IDE, powered by an embedded Faust compiler.
- Phausto brings a modular-synth-inspired approach to designing synths and effects.
- Unit Generators are connected by setting their members value or using the **ChucK** operator => .



# Let's build a synth

```
1 oscillator := Pulse0sc new label: 'MySynth'.
2 env := ADSREnv new label: 'MySynth'.
3 filter := Korg35LPF new label: 'MySynth'.
4 effect := GreyHoleDW new label: 'MySynth'.
5
6 synth := oscillator => env => filter => effect.
7 dsp := synth stereo asDsp.
8
9 dsp traceAllParams.
10 dsp init.
11 dsp start.
12 dsp playNote: 60 prefix: 'MySynth' dur: 1.
```

# Drive Phausto with Coypu

```
1  
2  
3 p := Performance uniqueInstance .  
4 p forDsp: dsp.  
5  
6 '38, 56, 65, ~ , 42 , 76 , 81 , 45' asDirtNotes to: #MySynth.  
7  
8 p freq: 129 bpm.  
9 p playFor: 16 bars.  
10  
11  
12
```

# Sum of synths

```
1  dsp := (Kick new + Clap new + CombString new + Hat new + PopFilterDrum new + SawTrombone new + DubDub new) stereo asDsp.  
2  
3  dsp init.  
4  dsp start.  
5  
6  p := Performance uniqueInstance .  
7  p forDsp: dsp.  
8  
9  '8080' hexBeat to: #Kick.  
10 '38, 56, 65, ~ , 42 , 76 , 81 , 45' asDirtNotes to: #DubDub.  
11 #rumba asRhythm to: #Hat.  
12 32 randomTrigs to: #Clap.  
13 '60/32 , 63/16 ' asDirtNotes to: #SawTrombone.  
14 #( 7 16 euclidean) to: #PopFilterDrum.  
15  
16 p playFor: 16 bars.
```



# Turbo/EcoPhausto

- TurboPhausto is set of synthesisers and effects, *made with Phausto*, crafted for programming music on-the-fly with Coypu.
- Inspired by the SuperDirt audio engine for SuperCollider, it turns Pharo into a powerful environment for live-coded music and algoraves.
- Packed with 50 MB of algorave-ready royalty free audio samples, made by *Lucretio*, *The Analogue Cops* and the legendary dutch electro producer *Legowelt*.



# Ponle la gasolina!



```
1 TurboPhausto start.  
2 tp := TurboPhausto new.  
3 tp bpm: 167.  
4  
5 #junglekick' asRhythm to: #kick.  
6 16 cumbiaClave to: #marimba.  
7 #(5 16) euclidean to: #conga.  
8  
9 tp play.
```

```
1 EcoPhausto start.  
2 ep := EcoPhausto new.  
3 ep bpm: 137.  
4  
5 '9090' hexBeat to: #kick.  
6 16 quavers to: #ch.  
7 16 randomTrigs to: #bongo.  
8  
9 tp play.
```

- The **start** method loads all instruments and effect and create a **DSP** that is assign to a class variable named **tpDsp**.
- It also initialise a new **Performance** (also stored in a class variable) and connect it to the **DSP**.

# Build your own library



# PART 4: PERFORMANCE



# Tuning Phausto for Coypu

Organising instruments into performance-ready classes

Assigning instruments to Coypu performers

# Nibbling things up

p := Performance uniqueInstance .

“Any Performer can be used”

p freq: 129 bpm.

p playFor: 128 bars.

p mute: #kick.

p mute: #(#clap #bass).

p solo: #bongo.

p unsolo: #bongo.

p backup.

p muteAll.

‘AAAAAAAF’ hexBeat to: #snare.

p restore

# Nibbling things up/2

EcoPhausto start.

ep := EcoPhausto new.

ep playFor: 128 bars.

downbeats #asRhythm to: #kick.

rumba #asRhythm to: #kick.

#kick swapWith: #snare.

ep solo: #bongo.

ep solo: #bongo.

ep unsolo: #bongo.

ep backup.

#(5 16 euclidean) to: #bongo.

(#(5 16 euclidean) offset: 2)to: #conga. “Also try offsetLeft or offsetRight!”

ep restore.

# PART 5: VISUAL INTERFACE



# Bloc

- **Bloc** is a powerful low-level graphical framework for Pharo, created by *Alain Plantec* and enhanced by the *Feenk* team for *GToolkit*. These combined efforts are now merging back into Pharo, significantly advancing its graphical capabilities.
- **Bloc** is poised to become the primary graphical framework for Pharo, gradually replacing the well-established but aging Morphic framework.
- **Bloc** is under active development, so features and behavior may change as it matures.

# Widgets for Phausto

**Building control interfaces for instruments**

**Building control Adding mixer controls and visual feedback for instruments**

**Linking UI elements to musical parameters**

# CoypulDE

**CoypulDE is a set of widgets for the Coypu and Phausto music and DSP libraries,  
implemented in Bloc and Toplo**

# Join the community



## Pharonauts on Discord