

# **Voice Allocation with poly~**

**and the *bach* Library for Computer-Assisted Composition (CAC)**

**A MaxMSP Patching Odyssey**

**ENT3320 — *Interactive VR, Music, Sound, and Memory***

**New York College of Technology, CUNY City Tech  
Spring 2020**

**Professor Louis Goldford**  
[LGoldford@citytech.cuny.edu](mailto:LGoldford@citytech.cuny.edu)



*Computer Music Center, Columbia University*



*Creative Commons  
Attribution-ShareAlike 4.0 International*

# CONTENTS

<b>Voice Allocation with poly~</b>	<b>1</b>
<b>1. Proportional Notation GUI</b>	<b>3</b>
1.1 Introducing bach.roll	3
1.2 Composing with bach.roll	4
1.3 Simple playback with bach.ezmidisplay	5
<b>2. Synthesis with the bach Library</b>	<b>6</b>
2.1 Monophonic Synthesizer	6
2.1.1 Monophonic Synth Subpatcher	7
2.2 Polyphonic Synth with poly~	8
2.2.2 poly~ Subpatcher	9
2.2.3 Monitoring the Mute-/Busy-States	10
2.2.4 poly~ Subpatch Folder Structure	11
2.3 Adding Control Over Synth Parameters	12
2.3.1 poly~ Subpatch With Ring Modulation	13
<b>3. A Boiler For poly~ Instruments</b>	<b>14</b>
3.1 poly~ Subpatch Scaffolding	15
3.2 Voice Trigger Simulation	16
3.3 Data Stream Simulation	17
3.4 Example Data Inside Coll	18
<b>4. What Is A Note? — Part 1</b>	<b>19</b>
<b>5. What Is A Chord?</b>	<b>20</b>
<b>6. MIDI vs. Midicents (Microtonality)</b>	<b>21</b>
<b>7. Playing With Beat Frequencies</b>	<b>22</b>
<b>8. What Is a Note — Part 2</b>	<b>23</b>
8.1 bach “Slot Machines” Tutorial — Part A	24
<b>9. Controlling poly~ With bach Slots</b>	<b>25</b>
9.1 poly~ Subpatch	26
<b>10. Sketch Vs. Score</b>	<b>27</b>
<b>11. How to Record Audio from Max</b>	<b>28</b>

# 1. Proportional Notation GUI

# 1.1 Introducing bach.roll

**1. Say Hello to Your New Friend: `bach.roll`**

`<live.tab>`

- `read` <= load a MIDI or text file with musical sequence below
- `writetxt` <= write your music to a MIDI (.mid) or text (.txt) file
- `dump` <= send everything out of its 1st outlet (on the left)
- `play` <= play the sequence! (but notice that you don't hear anything...)
- `stop` <= stop it (duh)
- `clear` <= erase the sequence

`<attrui>`

`vzoom` <= adjust the vertical zoom (enter a %)  
`zoom` <= adjust the horizontal zoom (drag up or down)

`<bach.roll>`

`bach.roll` is what we might call a "score editor" that uses "unmetered, proportional" notation. That means you generally don't have to worry about rhythm; you can visualize your sequence of musical ideas using nothing but absolute, "chronocentric" time (minutes and seconds).

You can use `bach.roll` to display a MIDI file, such as the one inside the `bach.roll` object below, or you can use it to notate your OWN...

You can also use it to PLAY your sequence of notes. For example, press the "play" button on the left (4th down in the list inside the `<live.tab>` object). Notice that a green scrollbar wipes across the screen and highlights each "note" as it plays. But you don't hear anything!... We need something to "synthesize" our notes as they are played back by `bach.roll`...

<Also, you can instantiate your own `bach.roll` object if you type "`bach.roll`" into an empty object box like this. But it won't look exactly like the one above; that's because we're using a `bach.roll` object that has been specially formatted for you to use. Therefore, it might be easiest to simply COPY and PASTE the `bach.roll` objects you find in these tutorial patches, clear their contents, and recycle them over and over in your own patches...>

## 1.2 Composing with bach.roll

bach.poly.02.compose

100% ▾

2. Write your OWN notes in bach.roll

<live.tab>

- read <= load a MIDI or text file with musical sequence below
- writetxt <= write your music to a MIDI (.mid) or text (.txt) file
- dump <= send everything out of its 1st outlet (on the left)
- play <= play the sequence! (you won't hear anything...)
- stop <= stop it (duh)
- clear <= erase the sequence

Below is the same bach.roll object but missing the notes from before. I've simply pressed "clear" on the left to erase its contents.

CREATE SOME NOTES OF YOUR OWN:  
In your locked patch, cmd. + click inside bach.roll to create a single note. Do this a couple of times...

ALTER THE LENGTH OF A NOTE:  
See those long black bars that extend from each circle? They're called "tails" and they show the note's length — its duration. Click + drag across the right-hand side of a tail to select it. Now, shift + click + drag on it to adjust horizontally the note's duration.

COPY + PASTE SOME NOTES:  
Click + drag over some notes to select them. They turn green. Now, cmd. + C (copy), click anywhere in bach.roll, and cmd. + V (paste).

TRANSPOSE (MOVE PITCH UP OR DOWN):  
Click + drag over some notes to select them. They turn green. Use your computer's arrow keys to move them up or down in pitch.

<bach.roll>

...But, notice we STILL can't hear anything...

CMC CC BY SA

louis.goldford@columbia.edu — 2019, NYC

# 1.3 Simple playback with bach.ezmidisplay

bach.poly.03.ezmidisplay

3. Simple MIDI Playback Engine

<live.tab>

- read <= load a MIDI or text file with musical sequence below
- writetxt <= write your music to a MIDI (.mid) or text (.txt) file
- dump <= send everything out of its 1st outlet (on the left)
- play <= play the sequence! (you won't hear anything...)
- stop <= stop it (duh)
- clear <= erase the sequence

<attrui>

vzoom	Auto
zoom	100.

<= adjust the vertical zoom (enter a %)  
<= adjust the horizontal zoom (drag up or down)

<bach.roll>

We can listen to our musical ideas by connecting `bach.roll` to this object, `bach.ezmidisplay`. Notice that we make this connection using `bach.roll`'s 7th outlet below, called the "playout."

Press "play" on the left to hear some music. Press stop to, well... stop.

What we actually hear is MIDI data from the `bach.roll` being sent to `bach.ezmidisplay`, played back on our computer's internal MIDI synthesizer.

But these simple sounds are mostly boring! If we want to build our own sounds, or use more interesting waveforms, we need to make our own synthesizers...

bach.portal @out t <= a simple way to "monitor" the data coming out of `bach.roll`

done

CMC

louis.goldford@columbia.edu — 2019, NYC

## **2. Synthesis with the *bach* Library**

## 2.1 Monophonic Synthesizer

**4. A Dirty Monophonic Synth**

<attrui>

vzoom	Auto
zoom	100.

<live.tab>

Turn audio on (click the <ezdac> below).  
Play the bach.roll.  
Double-click on the subpatch called "monophonic.synth" (highlighted in RED) to see notes on how this cheap synthesizer works...

<bach.roll>

Onset 3.654s Cents 9971.4

CMC BY SA

bach.print <= another way to monitor; watch the Max console (shift + cmd. + M) as you play the sequence for note data!  
(Also, bach.print is super helpful for DEBUGGING!)

p monophonic.synth <= Double-click to see contents.

live.gain~

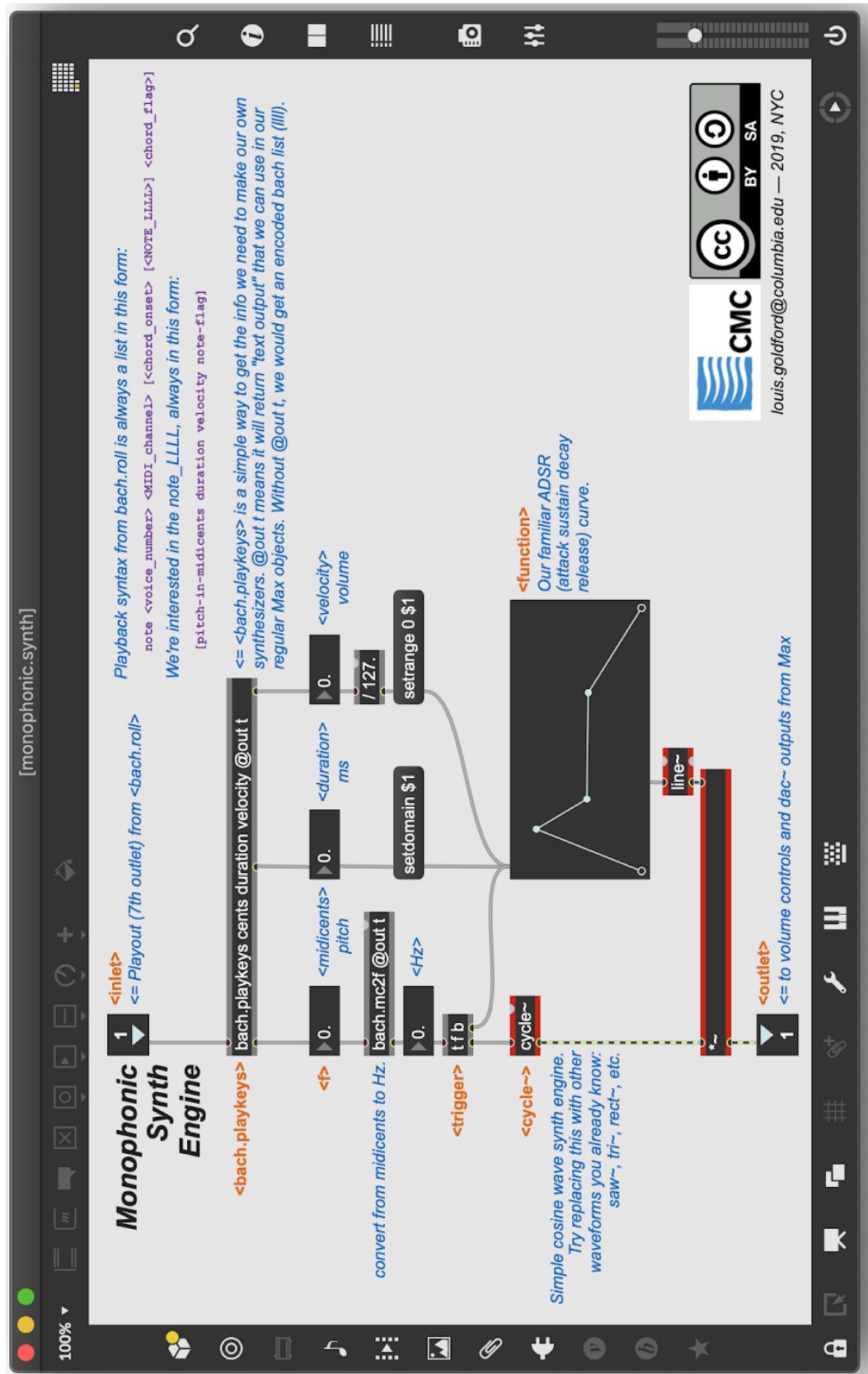
-20 dB

<ezdac~>

<= Make sure audio is "on" by clicking this before you play the bach.roll.

...But, there are 2 problems with this:  
1. Hear those "clicks"? We need a "cleaner" synth without clicks; and  
2. Notice we have TWO PARTS above... We should be hearing 2 notes at the same time, and other notes that overlap, but we only get 1 of those notes when we should hear two (this is also called "monophonic").  
We need a version that will play more than one note at the same time...

## 2.1.1 Monophonic Synth Subpatcher



## 2.2 Polyphonic Synth with poly~

bach.poly.05.polyphony

5. Polyphony Using poly~

In music, POLYPHONY == more than one note at the same time.  
In synthesis, VOICE ALLOCATION is what we use to achieve polyphony.

<bach.roll>

<live.tab>

read  
writetxt  
dump  
play  
stop  
clear

0'00" 0'01" 0'02" 0'03" 0'04" 0'05" 0'06"

Onset 3.191s Cents 4414.3

Monitor the poly~-'s mute and busy states:

p mute.busy prepnote <= poly~ eats "note" msgs.

Arguments to poly~ object:  
poly~ <name of patcher> <max. # voices>  
<= Double-click to see contents.

poly~ simple.voice.polyCore.v01 4

live.gain~

live.gain~ 0 0 0 <= "busymap" for each voice

0 0 0 <= "mutemap" for each voice

-20 dB

<ezdac~>

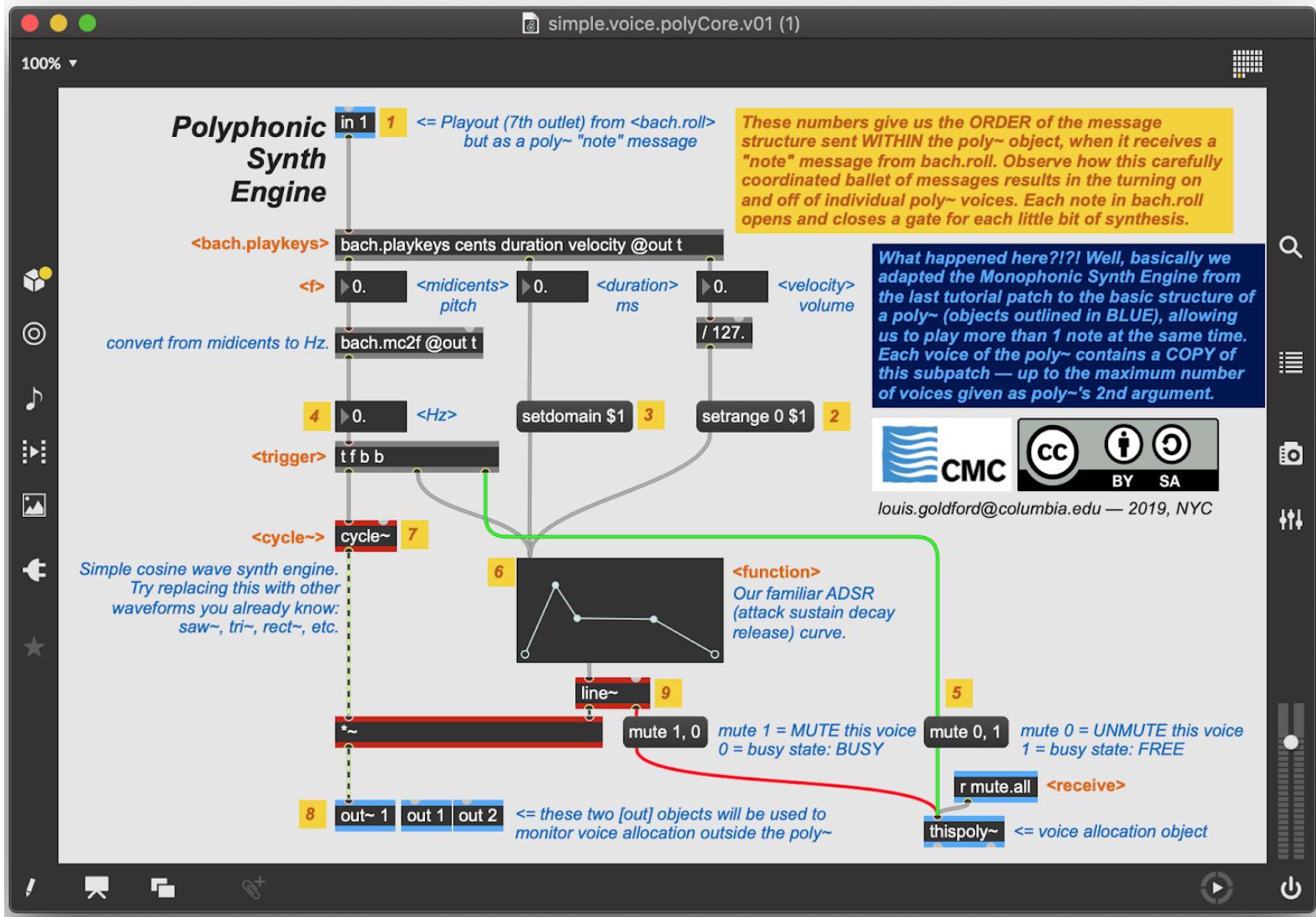
INITIALIZE by muting everything from outside of the poly~.

mute 1, 0 mute 1 = MUTE this voice  
0 = busy state: BUSY

s mute.all

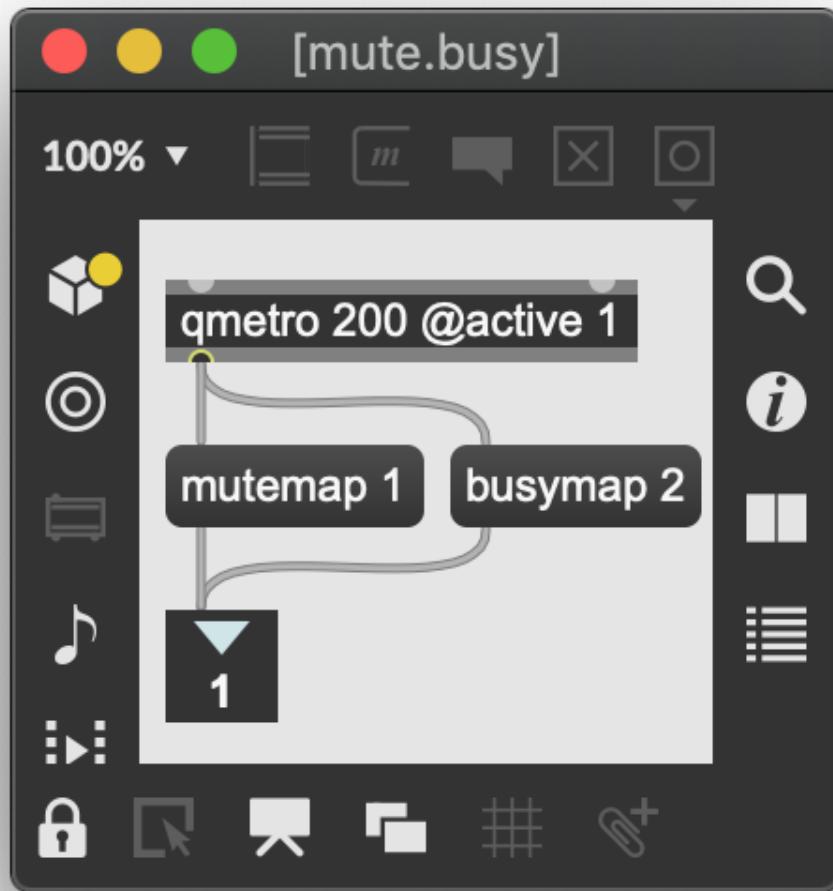
Each voice in a poly~ is an internal copy of the subpatch loaded into it (its first argument). MUTE and BUSY states tell us which of the voices are currently being used. When we send the correct messages in order to thispoly~ (inside the poly~ subpatch), the system turns voices on and off at the right time, saving us much-needed computational power!

## 2.2.2 poly~ Subpatcher



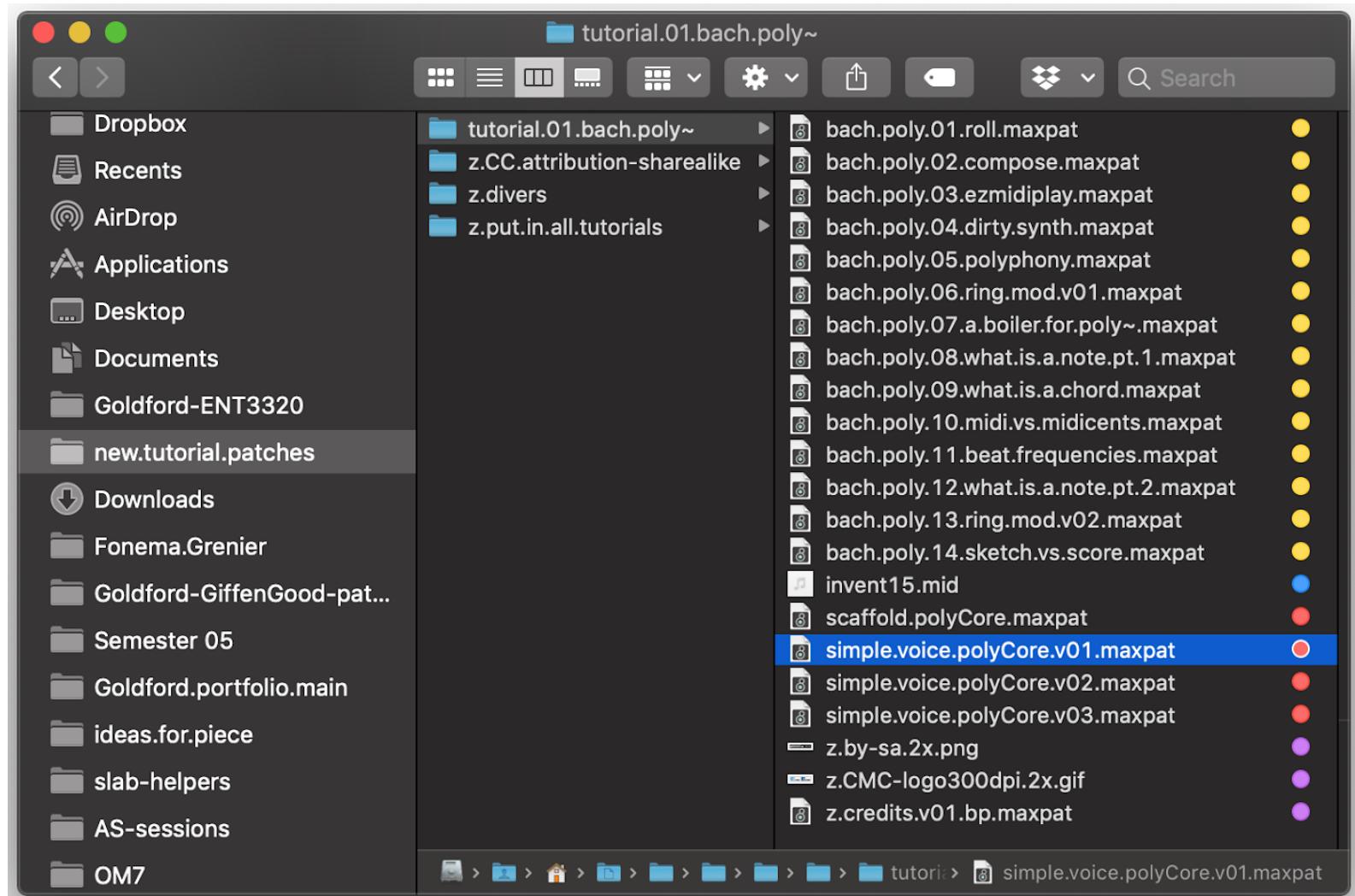
### 2.2.3 Monitoring the Mute-/Busy-States

By adding this subpatcher (below) to your main patch and connecting it to your poly~ (see 2.2 above), and an additional two extra outlets to the poly~ subpatcher itself (see 2.2.2 above), one can easily monitor the “mute” and “busy” map for all voices of the poly~, the total number of which was given as the 2nd argument to the poly~ object in the main patch.



## 2.2.4 poly~ Subpatch Folder Structure

The main patch containing your poly~ object (see 2.2 above) and the subpatch loaded by the poly~ object and given as its 1st argument (see 2.2.2 above), must both be contained in the same directory, at the same level:



## 2.3 Adding Control Over Synth Parameters

**6. Let's Make Our Synth A Bit More Interesting...** ...Add just a pinch of ring modulation to spice things up... <bach.roll>

**Monitor the poly~'s mute and busy states:**

- p mute.busy**: Initialize by muting everything from outside of the poly~.
- mute 1, 0**: mute 1 = MUTE this voice  
0 = busy state: BUSY
- s mute.all**

**<live.tab>**

**<live.gain~>**

**<ezdac~>**

**<bach.roll>**

**poly~ simple.voice.polyCore.v02 4** (Double-click to see contents.)

**target 0**: Change the number here by dragging up/down. This is the amount by which we ring modulate each note.

**t fb**: Each time you change the value above, (1) send poly~ a "target 0" message so that each voice receives this msg. in its 2nd inlet when you (2) send it.

**0 0 0**: busymap for each voice

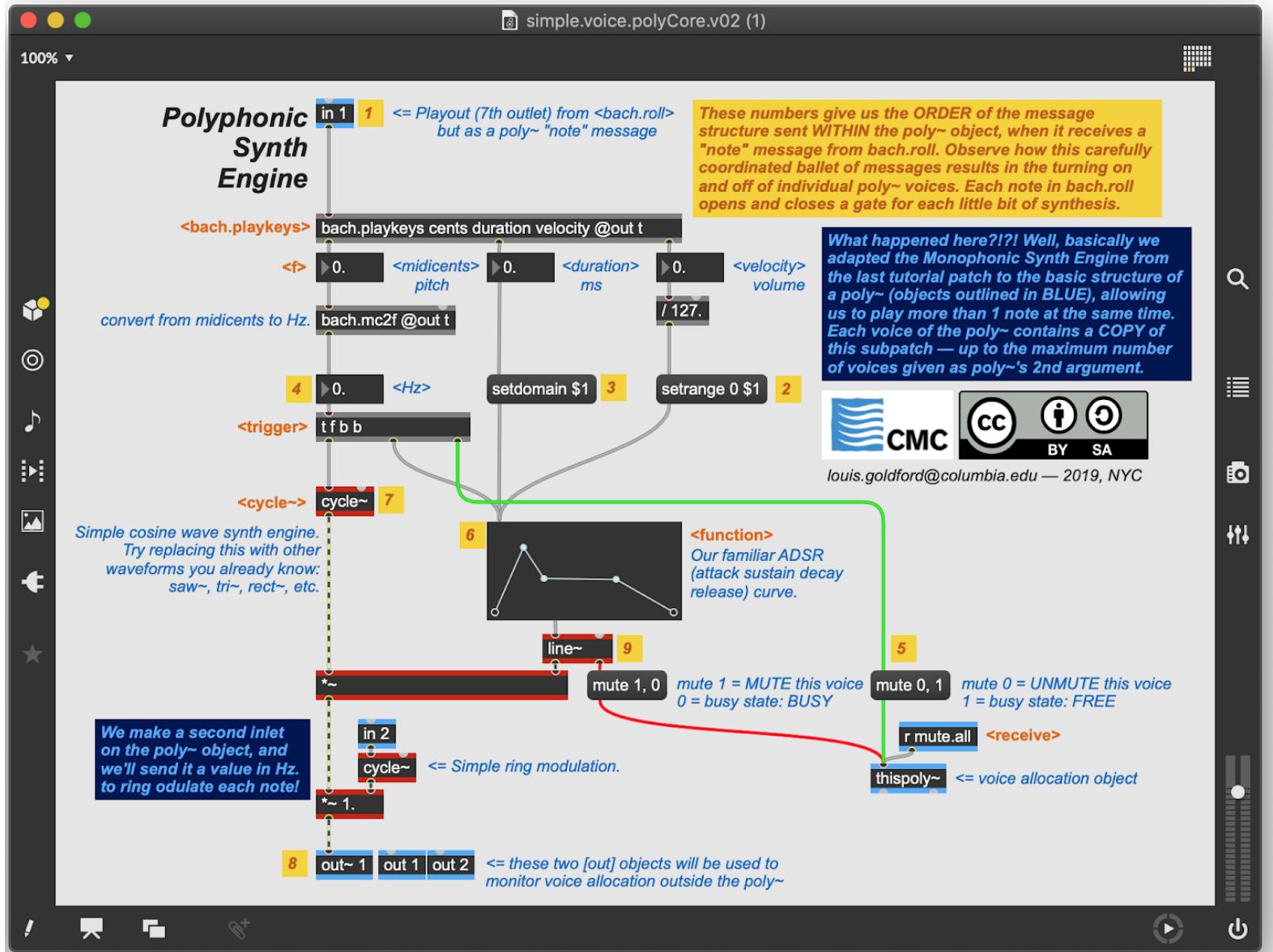
**0 0 0**: mutemap for each voice

-20 dB

Each voice in a poly~ is an internal copy of the subpatch loaded into it (its first argument). MUTE and BUSY states tell us which of the voices are currently being used. When we send the correct messages in order to thispoly~ (inside the poly~ subpatch), the system turns voices on and off at the right time, saving us much-needed computational power!

**CMC** **BY SA**  
louis.goldford@columbia.edu — 2019, NYC

## 2.3.1 poly~ Subpatch With Ring Modulation



# 3. A Boiler For poly~ Instruments

**7. A Boiler For poly~ Instruments**

We can use poly~ to create all sorts of fun polyphonic instruments in Max. We'll be making a few instruments in this class using poly~. We can even use poly~ to create things that are not so instrument-like — For example, some people use poly~ just to turn signals on and off in their patch. Some use a poly~ to compute the spatial movement of 3D sound objects. Some design elaborate reverbs with a poly~.

In general though, a poly~ is designed to make multiple copies of the same thing, permitting simultaneous instances of the same DSP (digital signal processing) algorithms. As such, a well-designed poly~, no matter what kind of signal it's producing, should have a few of the same elements:

A "panic" method" in case all voices need to be suddenly turned off.  
**s mute.all**

**p mute.busy** A way to monitor the mute and busy states; to know which voices are free and which are occupied.

**p data.stream.simulation** <coll> 6 6400  
**coll data.inside.all.poly.voices** A method of getting streams of data into a poly~. Here, the numbers in the <coll> object are constantly being updated, but they can all be referenced inside the poly~ by including an instance of the same <coll> (see inside)

**duration** 500.00 **attack** 100.00 **decay** 230.00 **sustain** 320.00 **release** 950.00  
**<live.dial>** **s dur** **s attack** **s decay** **s sustain** **s release** A way to look inside any voice of the poly~. This method opens up a window for each instance.  
**p voice.trigger.simulation**

**r to.poly**

**poly~ scaffold.polyCore 16** <= Double-click to see contents.  
mutemap  
busymap

louis.goldford@columbia.edu — 2019, NYC

## 3.1 poly~ Subpatch Scaffolding

in 1 <= Use the <in>, <out> and <in~> and <out~> objects in place of normal Max inlets / outlets.

<trigger>

Your "synthesis engine," a.k.a. your DSP process (digital signal processing) goes here.

CHALLENGE: Use this basic logic, this "scaffold" of a poly~ to build a ring modulator, an AM synth, or a sampler; or any other "classic synthesis" process that you may have encountered in Max or another program.

HINT: We already built a ring modulator earlier in this tutorial...

t l b 1  
r dur  
del 500  
r mute.all  
t 0  
adsr~ 500 0 1 500  
r attack  
r decay  
r sustain  
r release  
thispoly~  
1  
p get.this.index.value  
coll data.inside.all.poly.voices  
4100  
out~ 1 out 1 out 2

<= Scales the volume of each note "event" played back by this poly~ voice.

<= Used for mute & busy states

<= A system that manages the opening and closing of the signal gate; before we used a <function> object and sent separate "mute 0, 1" and "mute 1, 0" messages. The <adsr~> object does all of this at the same time. (option + click on <adsr~> to see its help file.)

<= A tool for voice allocation.

<= voice number of this poly~ instance

<= repeat that # over and over again...

<= all <coll> data inside every voice

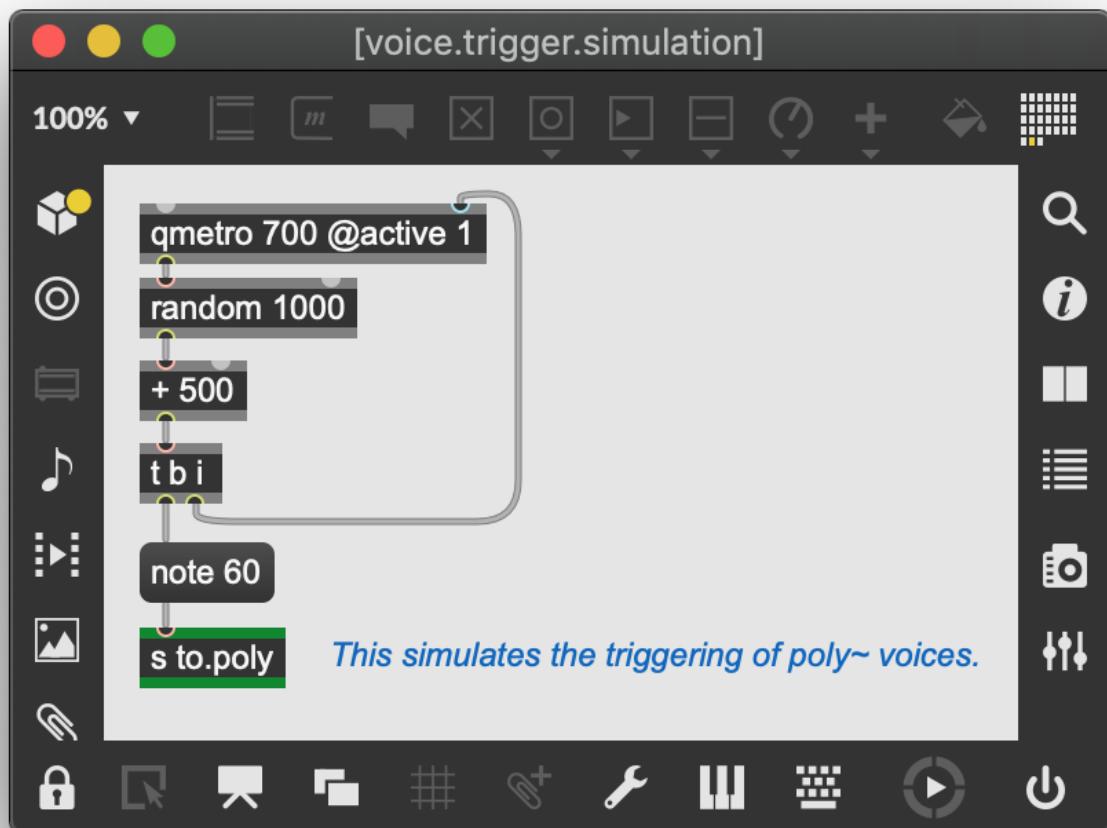
<= Current value for this voice number. Data updating "live" inside the poly~!

CMC CC BY SA

louis.goldford@columbia.edu — 2019, NYC

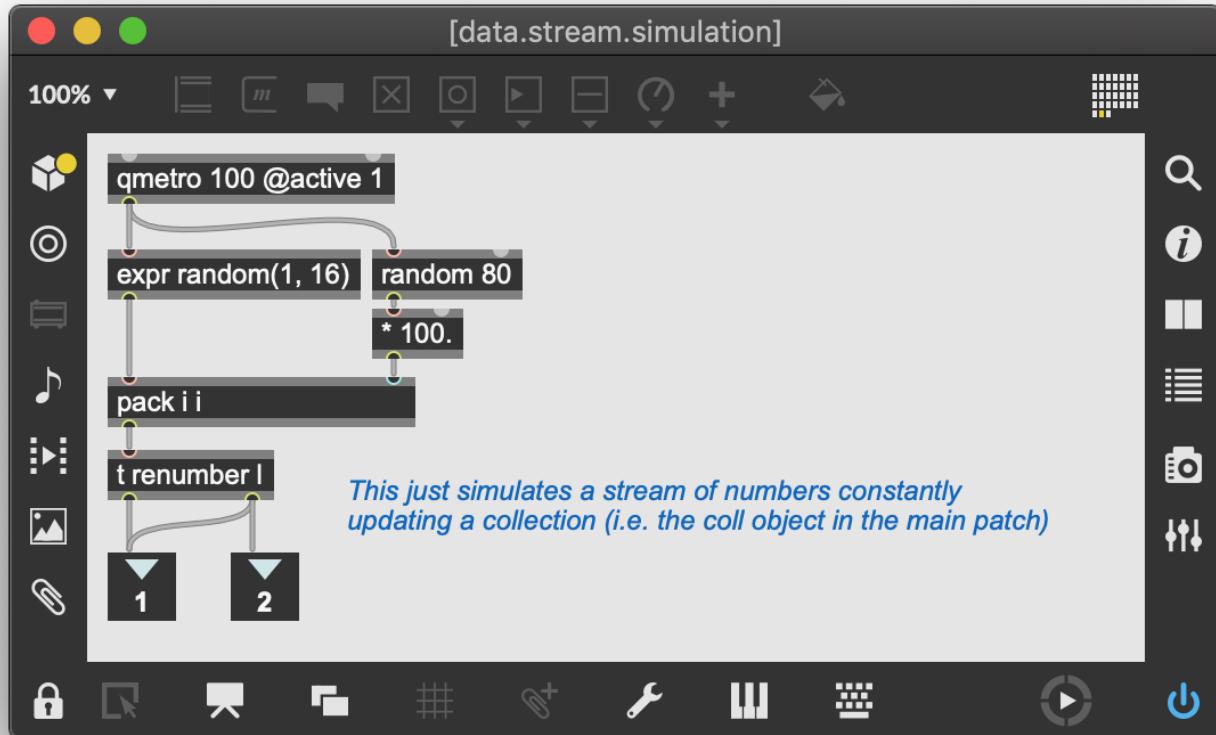
## 3.2 Voice Trigger Simulation

Use a *MIDI keyboard* or a *bach.roll* in place of this:

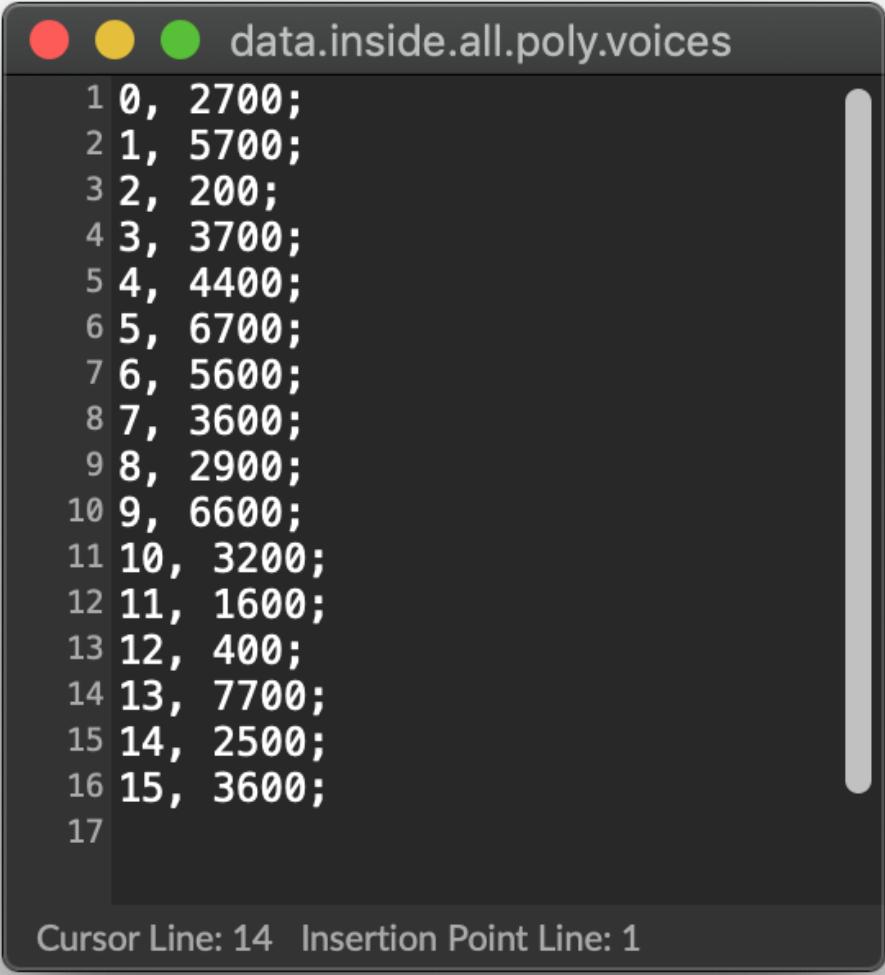


### 3.3 Data Stream Simulation

This could be a stream of data updating itself live from a webpage, or it could be algorithmically-generated from within your main patch somewhere. The idea is to shuttle a lot of numbers at the same time into your poly~ and to be able to use them all, or individually based on the voice number (as in this example, see 3.1 above).



## 3.4 Example Data Inside Coll



The image shows a screenshot of a code editor window. At the top, there are three colored circular icons: red, yellow, and green. To the right of these icons, the text "data.inside.all.poly.voices" is displayed. Below this, a list of 17 numbered entries (1 through 17) is shown, each consisting of two numbers separated by a comma. The entries represent voice data, likely pitch or frequency values. The cursor is positioned at line 14, and the insertion point is at line 1.

```
1 0, 2700;
2 1, 5700;
3 2, 200;
4 3, 3700;
5 4, 4400;
6 5, 6700;
7 6, 5600;
8 7, 3600;
9 8, 2900;
10 9, 6600;
11 10, 3200;
12 11, 1600;
13 12, 400;
14 13, 7700;
15 14, 2500;
16 15, 3600;
17
```

Cursor Line: 14 Insertion Point Line: 1

## **4. What Is A Note? — Part 1**

bach.poly.08.what.is.a.note.pt.1

100% ▾

In a locked patch, click on the note below so that it turns purple... <bach.roll>

**8. What Is A Note? — Part 1**

Now that we're getting into things a little bit, let's reexamine what we call a "note," from the perspective of a synthesizer...

...Notice how this text appears in `bach.roll` above the note:  
Onset 0.910s Cents 7200.0 Duration 2.212s Velocity 100

From this perspective, a single "note" is specified by 4 parameters:

`$\Delta t = t2 - t1$`    **ONSET** time ( $\Delta t$  wrt  $t=0$ , the start of the sequence)

`$t1=0 \xrightarrow{\Delta t} t2$`    **PITCH (in midicents)**

`$t1=\text{onset} \xrightarrow{\Delta t} t2$`    **DURATION** ( $\Delta t$  wrt  $t1=\text{note onset}$ )

**VELOCITY** (a MIDI value of 0 — 127)  
used to express relative "loudness" of each note

In `bach.roll`, each "note event" from `Playout` (7th outlet) is expressed as a LIST consisting of these basic 4 parameters:

```
note <voice_number> <MIDI_channel> ( <chord_onset> ( <pitch> <duration> <velocity> <note-flag> ) <chord_flag> )
```

# 5. What Is A Chord?

bach.poly.09.what.is.a.chord

100% ▾

9. What Is A Chord?

So far we've mostly been working with notes that appear one after another. At least, until we added POLYPHONY in the last patch. Chords, as it turns out, are a special case of polyphony...

A chord is just a collection of notes that all sound at the same time... In other words, they all share the same ONSET value.

In traditional music notation, we show this by placing all members of the same chord on one "STEM," i.e. the line that points up here:

In bach, all notes in a chord can have their own duration:

<bach.roll>

<bach.ezmidisplay>

These 4 notes all have the same ONSET, and bach interprets this by placing them all on the same stem, just like in music.

You can listen to the whole chord or to each note of the chord individually. With the patch locked, click on bach.roll and press the spacebar to hear the whole chord. Now, click + drag only over the notes you want to hear, highlighting only those notes in purple. Now, shift + spacebar to hear only your selection.

ADD NOTES TO THIS CHORD:  
With the patch locked, hover over a note in the chord until you see the cross.  
Hold down option + shift. The green + (plus) sign appears.  
With these keys held down and the green + sign visible, click on a note.  
It turns purple as usual, but...  
Now DRAG and it creates a COPY of the note, still on the same stem.  
Drag up or down to the pitch you want, and then release everything.

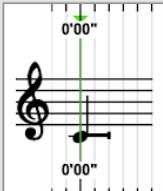
Congrats... now you have 5 notes in your chord!  
Try dragging the notes up or down, rearranging them as you please.  
Listen to them (see the text next to bach.ezmidisplay), and make changes in pitch and in the number of notes until you find a chord that you like.

In bach.roll, each "chord" from Playout (7th outlet) is expressed as a LIST of LISTS consisting of the different notes. Each list is wrapped in parentheses or brackets. Here is a chord with 3 notes in it:

```
chord <voice_number> <MIDI_channel> (<chord_onset> ( <pitch> <duration> <velocity> <note-flag> ) ( <pitch> <duration> <velocity> <note-flag> ) ( <pitch> <duration> <velocity> <note-flag> ) <chord_flag> )
```

# 6. MIDI vs. Midicents (Microtonality)

bach.poly.10.midi.vs.midicents

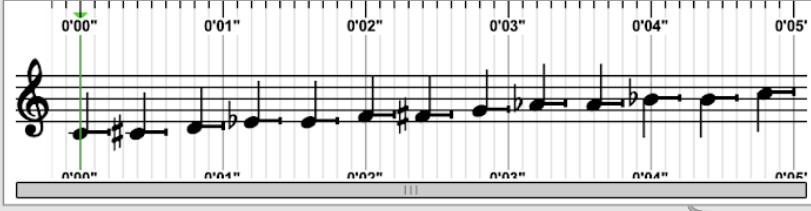
100%  On a MIDI keyboard, "middle C" is arbitrarily given as MIDI note number 60:



louis.goldford@columbia.edu — 2019, NYC

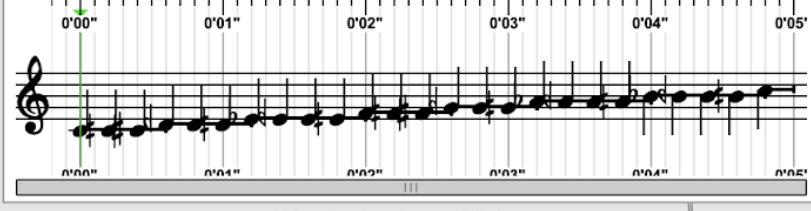
<= But if you click on the note and type cmd. + I (to bring up bach's "inspector" patch), press enter and notice that bach calls this pitch 6000, not 60.

In normal MIDI note numbering, adjacent integers are given for each half step between the notes of the keyboard. For example, below are all the notes between middle C and the next C above it; MIDI notes 60, 61, 62, ... 70.



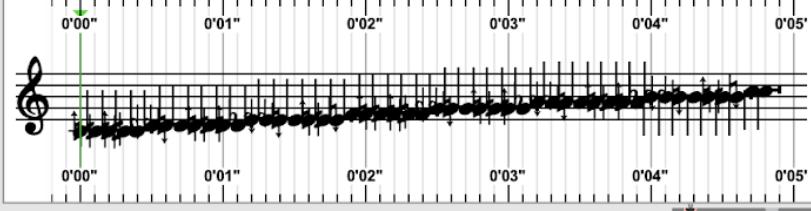
In bach, we use MIDICENT values (MIDI note number \* 100). So 60 becomes 6000, and this gives us 100 "cents" between each half step to express smaller degradations of pitch "between the keys of the piano." We call these kinds of smaller tones "microtones."

For example, if we divide all half steps on the piano in half, we obtain "quarter tones" — the half steps between half steps: 6000, 6050, etc...

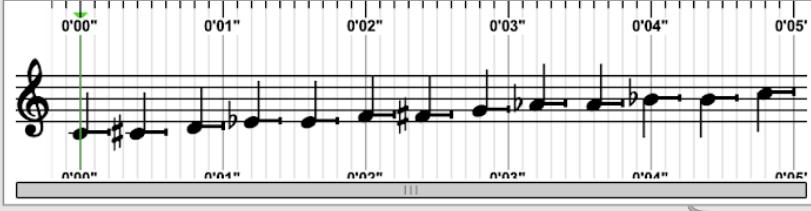


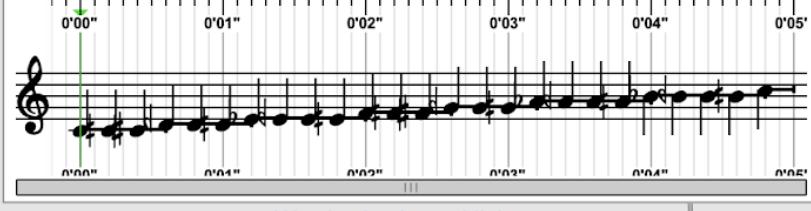
We give bach.ezmidisplay an argument of 4 so it plays back quarter tones. => bach.ezmidisplay 4

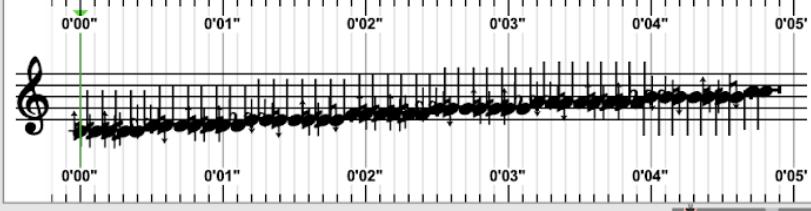
If we further subdivide quarter tones, we get the half steps of half steps of half steps — or what we simply call "eighth tones" (6000, 6025, 6050, etc...)



bach.ezmidisplay 8







# 7. Playing With Beat Frequencies

bach.poly.11.beat.frequencies

100% ▾

11. Beat Frequencies

We can use microtonality to achieve some seriously cool timbral effects...

louis.goldford@columbia.edu — 2019, NYC

For example, here are 2 pitches spaced one 8th-tone apart:

Play the bach.roll and observe that when the 2nd pitch enters, we hear a "beating" between the two frequencies...

6700. 6725. <= midicent values for the 2 notes above (pitch)

bach.mc2f @out t <= conversion in Hertz

391.995436  
397.697141 <= their values in Hertz (frequency)

expr \$f2 - \$f1 <= calculate the difference between the 2 frequencies

5.701705 <= difference in Hertz

translate hz ms <= conversion in milliseconds

175.386137 <= difference value in milliseconds

metro 175.386137 @active 1

click~

beat.freq

This is the frequency at which the 2 pitches above beat — the difference between their 2 frequencies values.

The beating creates subtle shifts in timbre that change the sound over time & make the notes sound more "alive."

-inf dB

bach.ezmidisplay 8

Search icon

Information icon

Volume icon

Mute icon

Speaker icon

File icon

Edit icon

Tools icon

Help icon

Power icon

## 8. What Is a Note — Part 2

bach.poly.12.what.is.a.note.pt.2

100% □ m □ X □ □ □ □ □ +

### 12. What Is A Note? — Part 2 see also bach tutorial #5 — Slot Machines (a through x)

Previously we'd seen that a note can be defined by its pitch (midicent value), duration, onset, and velocity. But more parameters are available to us. In LISP code, a "slot" is an additional field for storing data, contained in an object. In bach, these objects are notes themselves, and many additional data fields can be contained in a single note. These slots can represent anything we want — in music, they most commonly refer to lyrics, amplitude envelopes or other synthesis parameters, or even entire scores themselves. We'll use slot #1 to control our ring modulation...

First, we define some "slotinfo" about slot #1, and then we show it in the bach.roll (bgslots 1):

```
(slotinfo (1 (name ringmod) (key f) (type function) (range 0 10000) (representation Hz))), bgslots 1
```

You can see the function curve drawn in slot #1 (red), associated with the note. In a locked patch, click on the note and press "1" to bring up a "slot window" for the first slot. Click and drag to add and adjust additional points. Cmd. + double-click on a point to delete it. When finished, click outside the red slot window to make it disappear. Watch what happens to the slot when you adjust the length of the note... <bach.roll>

<live.tab>

The patcher interface shows a musical score with a note highlighted. Below the score is a graph with a red line representing a function curve. To the left of the graph is a vertical stack of buttons: read, writetext, dump, play, stop, clear. To the right of the graph is a horizontal timeline with time markers at 0'00", 0'01", 0'02", 0'03", and 0'04". Below the graph is a section labeled <live.tab>. Further down is a section labeled <bach.playkeys> containing a message box with the text: "bach.playkeys cents duration velocity (slot 1) @out t". Below this are four controls labeled <f> 0., <midicents> pitch, <duration> ms, and <velocity> volume. A large callout box points to this section with the text: "<= Now, play the bach.roll. When the note is triggered, we obtain pitch and other values, but we also obtain the contents of slot #1. It's a 'function' slot, so it contains breakpoints in groups of 3 floating-point numbers, wrapped in parens. In the next patch we'll make a few adjustments to our ring modulation poly~ from the previous tutorial patch...".

CMC CC BY SA

louis.goldford@columbia.edu — 2019, NYC

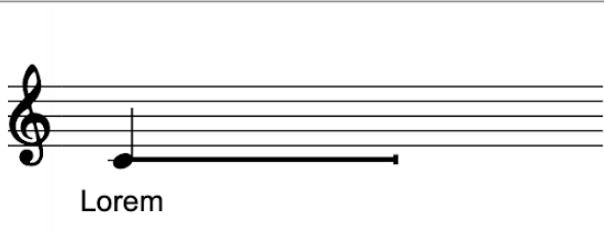
# 8.1 bach “Slot Machines” Tutorial — Part A

Slot Machines (a) - *behind each notehead, a brand new world*

One of the strong suit of the bach library is the concept of slot. To understand it, we have to think about a classical score, having pitches, durations, dynamics, or a proportional scores, having onsets, pitches and dynamics. This can be enough in some cases. But it won't be sufficient in many other situations.

Dealing with computer-assisted composition might also mean having the need to characterize notes more specifically; either because you want to assign a playing style, or because you need to attach some lyrics, or because you have built a sequencer with bach and now you need to handle amplitude envelopes or panning variations.

This is just a tiny tiny part of what one can have in mind which goes beyond the classic note representation. And if you think about this, all sums up in having some more data attached to each note. That's it. Nothing more than this: slots are additional data assigned to each note. All the rest is understanding the right type of data, and to operate accordingly.



You can define up to 30 slots, each one having name, type and features you desire. A slot is a customizable container: it is something that you can adapt at your needs. It can contains text, numbers, breakpoint functions, files, and much more. You can give each slot a name, and map it on a different key, in order to have its content popped up in the so-called slot window, when a note is selected and the specific key is pressed on the computer keyboard.

By default you can see first 10 slots at work by selecting notes and pressing numbers 1 to 0

When a slot window is open, press Shift+TAB to switch to the next slot, and Shift+Alt+TAB to step back to the previous one. Use ESC (or click outside the window) to close it.

Another interesting and standard way to use slots is to associate them to some important visual or graphical content. For instance, changing note colors will be possible via associating a dedicate slot to it. The same is true for noteheads and for lyrics. In such a sense, slots are flexible enough to alter some graphical or musical parameters (via the family of attributes @link....toslot). That's why some slots come ready to contain dynamics, lyrics, articulations, noteheads, etc (popping up while using the 'd', 'l', 'a', 'h' keys). In this tutorial we won't get through this specific slot usage, we'll just stick with the classical usage of slots. But let's introduce things little by little!

*Love the tutorial name! I'll just drop it and go play online poker...*

[Continue to section \(b\)](#)

The interface includes a toolbar with various icons for file operations, a zoom slider (100%), and a status bar with playback controls (play, stop, volume).

# 9. Controlling poly~ With bach Slots

bach.poly.13.ring.mod.v02

100% ▾

13. Our Ring Modulation poly~ — Controlled by a Function Slot

Using slots, we can control many synth params beyond the typical pitch, onset, duration, and velocity data fields. Double-click on the red poly~ below to see a few adjustments we can make to control our ring modulation frequency using the contents of slot #1: We added [bach.slot2curve] to translate the slot data into useful values for a [curve~] object. The smoothly-moving frequency value from [curve~] is then used to adjust the frequency of [cycle~], which is then multiplied against the note frequency to obtain a ring modulation effect. Notice each note has its own function slot data (i.e. each has its own shape), and therefore controls its own shifting timbre.

**<bach.roll>**

**<live.tab>**

Monitor the poly~'s mute and busy states:

INITIALIZE by muting everything from outside of the poly~.

mute 1, 0      mute 1 = MUTE this voice  
0 = busy state: BUSY

s mute.all

**<live.gain>**

**<ezdac>**

**<Double-click to see contents.**

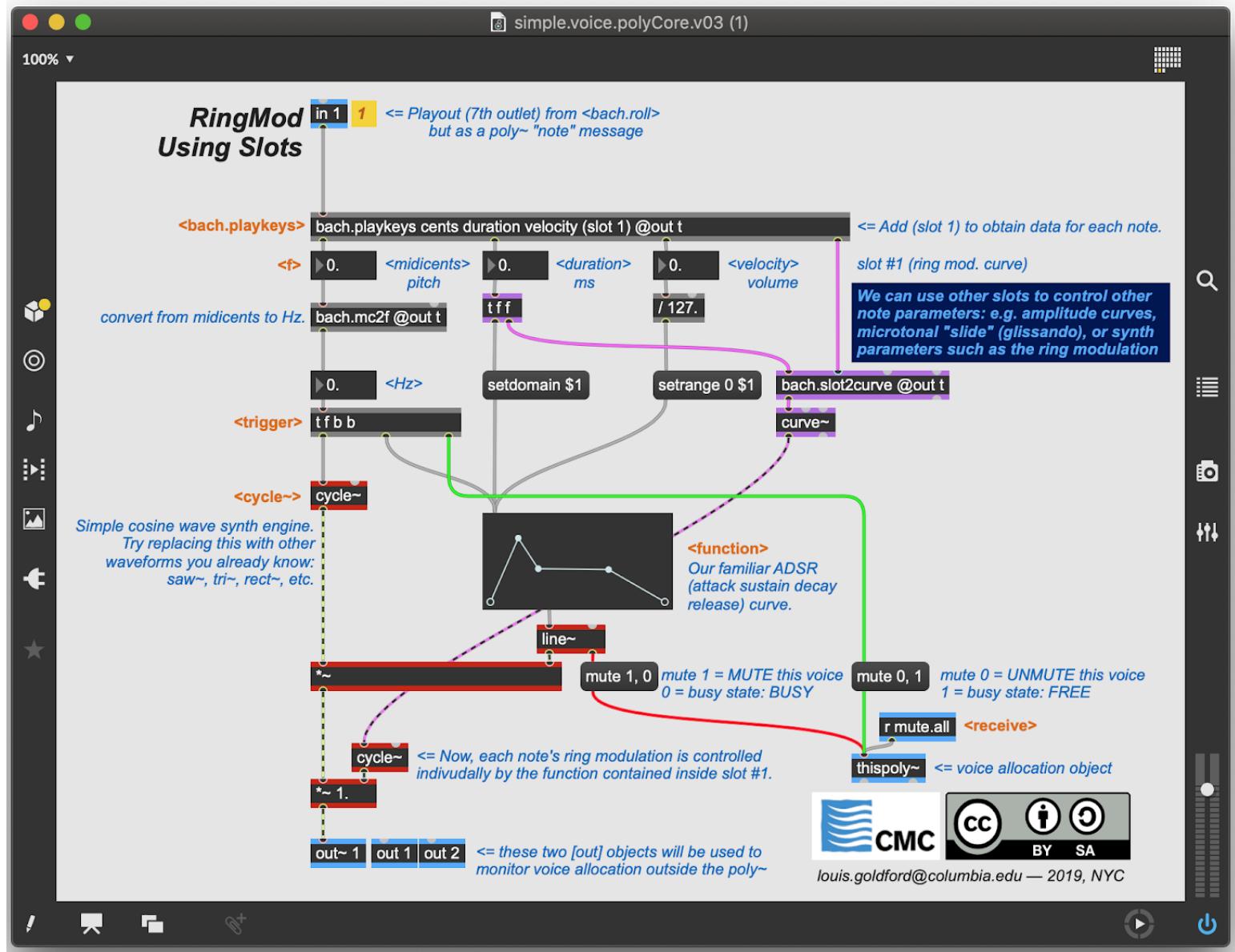
**<= "busymap" for each voice**

**<= "mutemap" for each voice**

CMC CC BY SA

louis.goldford@columbia.edu — 2019, NYC

## 9.1 poly~ Subpatch



## **10. Sketch Vs. Score**

**14. Sketch vs. Score — A Space For Free Play**

Just like composers of the classical era, today's musical creators need a space to experiment, improvise, and play freely with musical materials; and a separate space to develop and to "mock up" ideas as they begin to crystallize...

a musical "sketch"      a final notated "score"

We can record, sketch out, and play back with our ideas using [bach.transcribe] and a MIDI keyboard of our choice:

The patch includes the following components and instructions:

- Top Section:** A `loadbang` triggers a `midiinfo` object. The `AU DLS Synth 1` dropdown is set to `1 <= select MIDI keyboard`.
- MIDI Input:** A `notein` object receives notes from the keyboard. It has two outputs: `note #` and `velocity`.
- Recording and Transcription:** A `record` button starts recording, and `allnotesoff` stops it. The recorded notes are processed by `* 100.` and then sent to `pack i i 1`, which outputs to `bach.transcribe`.
- Output:** The `bach.transcribe` object outputs to a `<live.text>` object, which displays the transcribed notes.
- Score Generation:** The patch includes a Creative Commons Attribution-ShareAlike license logo (`CMC BY SA`) and the email `louis.goldford@columbia.edu — 2019, NYC`.
- Sketch vs. Score Comparison:**
  - bach.roll:** Shows a musical staff with a green vertical line at the start. The x-axis is time in seconds from 0'00" to 0'06".
  - bach.poly.14.sketch.vs.score:** Shows a musical staff with a green vertical line at the start. The x-axis is time in seconds from 0'00" to 0'06".
- Instructions:**
  - Click "record," start playing keyboard, stop when finished.
  - In a locked patch, select and COPY the notes you want to keep from the first `bach.roll` below:
  - Treat the `bach.roll` above as if it were a SKETCH; treat the one below more like a SCORE; that is, PASTE only the ideas you want to keep to the score below...

# 11. How to Record Audio from Max

You can quickly record any signal generated in Max.

Any sound coming into a `dac~` object can be captured using the **Quickrecord** subpatch:

