

Max Fundamentals

Juan P Bello

Max

- Max is a high-level graphic programming language, written in C
- It provides an interactive programming environment with extensive libraries of pre-compiled and pre-tested algorithms
- It automatically generates code from graphical configurations
- The programmer can be oblivious to the details of writing code
- That allows rapid prototyping by (non)programmers
- On the other side it makes the language less flexible, and usually performance is slower.
- Max is specifically intended for real-time computer music applications
- It allows the creation of standalone applications
- It can be expanded by including so-called external objects written in C and Java

A brief history

- First developed on the mid 80s by Miller Puckette, then at IRCAM
- The name Max honors Max V. Matthews



- Originally commercialized by Opcode Systems in 1990
- In the mid 90s, Dave Zicarelli (then at Opcode) started adding DSP functionalities into Max to match those in the new PD software created by Puckette (now at UCSD). That resulted in Max/MSP
- From 1999 onwards Max has been commercialized by Zicarelli's own company Cycling'74
- In 2003 a major new release, Jitter, empowered Max with matrix processing, real-time video processing and 3D capabilities.
- It is the most widely-used tool for interactive music systems.



Object-oriented programming

- Allow programmers to build digital systems as collections of interacting objects, each with a specific function (see Pope, 1991)
- Basically akin to the process of analysis: break down a whole into its constituent (functional) parts
- Objects communicate with each other through messages containing data or orders
- Popular programming languages (C for example) now feature extensive object-oriented capabilities
- Max shares many of the attributes of fully-developed object-oriented languages: modularity, nesting, reusability of objects.

Objects in MAX

- Objects are algorithms that execute an action
- Max programs are modular, composed of a number of these objects, each executing an action that contributes towards the overall desired musical result
- Three types of objects:
 - Standard: Existing objects that come with Max's library
 - Custom: User-created objects that result from the combination of standard ones
 - External: new objects programmed in C and Java
- Objects can receive messages from: computer peripherals, MIDI devices and other MAX objects
- Each object has a number of parameters that regulate its behavior.
- They are often variables that change during execution time. If not specified they might default to a "logical value".
- Some parameters can be typed directly into an object box as arguments (e.g. when they are not expected to change or as initial values)
- Incoming data overrides arguments.

Objects in MAX

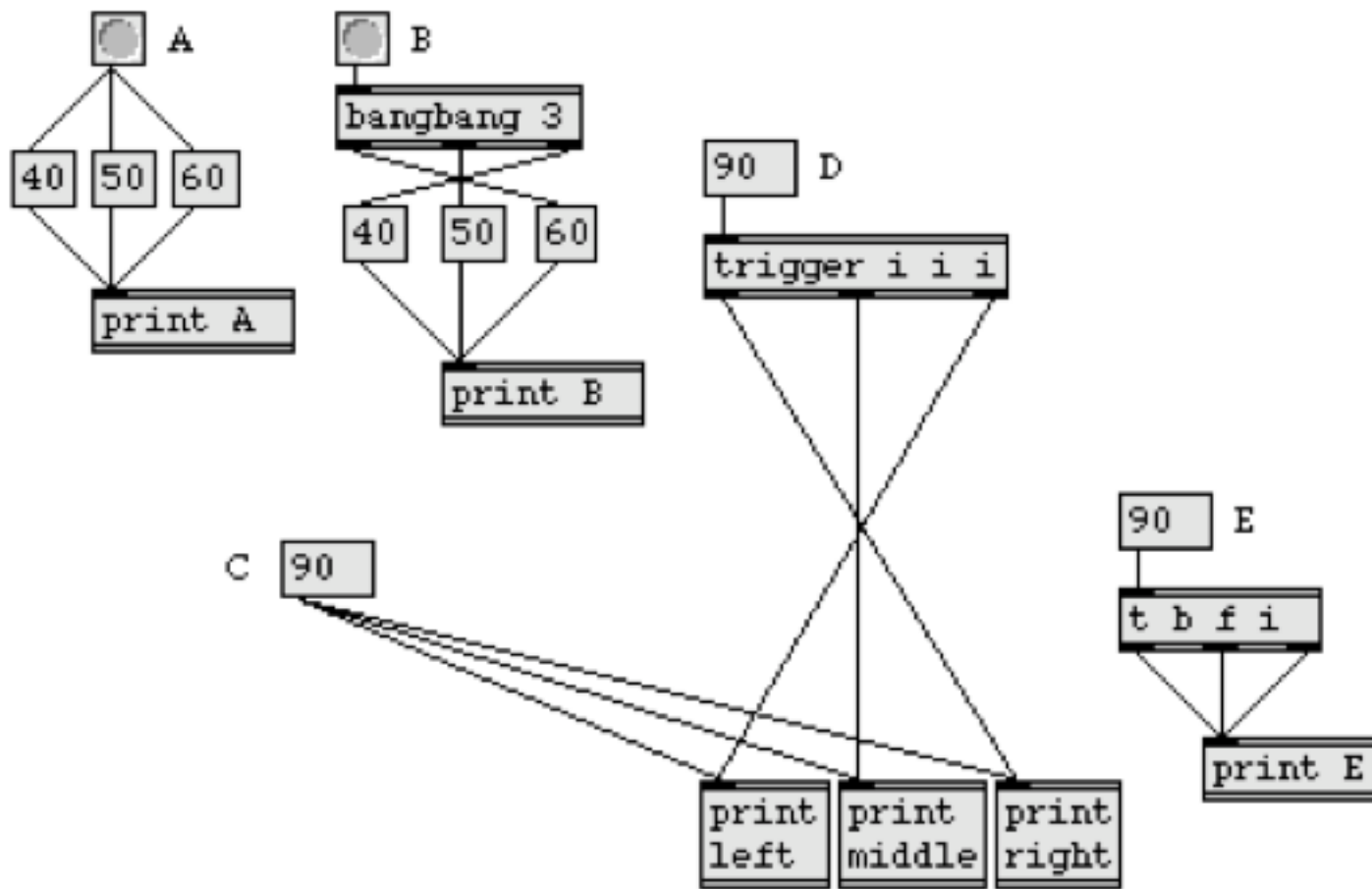
- Data transmission between objects is achieved by graphically wiring them
- We create these wires by dragging the buttoned-down mouse between the outlets and inlets of different objects.
- Because of the wiring process, the resulting programs are known as patches
- Patches can be modified in Edit mode
- To access Edit mode, we can: ctrl-click (or -click) on the patch window, or key ctrl-E (or -E).
- In Edit mode we can access the object Inspector (Get Info) and see/modify the object properties.
- You can also create your own objects
- Order of execution: Top to bottom, right to left; nothing (usually) happens until data arrives at the leftmost inlet (trigger); the *buddy* object forcefully synchronizes inputs.

Messages in Max

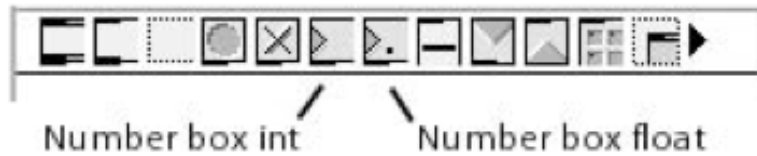
- Max communicates through information messages which are passed on between objects
- Max understands four main types of messages:
 - Bangs: A special type of message that means "Do it"
 - Numbers: numbers in Max can be integers (int) or decimal numbers (float) and can be expressed in a variety of formats (decimal, hex, MIDI number, note)
 - Words: or symbols, can be used as control data, i.e. to send orders to objects (e.g. read, stop, set)
 - Lists: Consists of a group of space-separated numbers or symbols.

Messages: Bangs

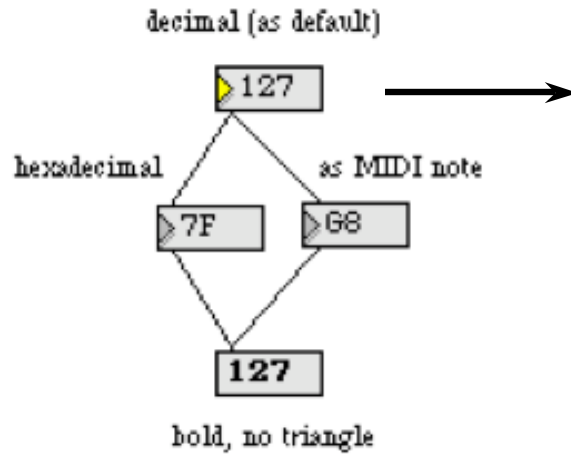
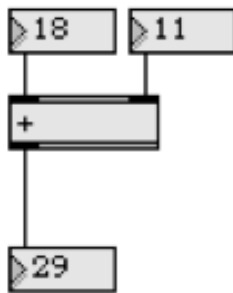
- The bang message triggers the object's action (Tutorial 7)



Messages: Numbers



number box can be used as a slider



Get Info

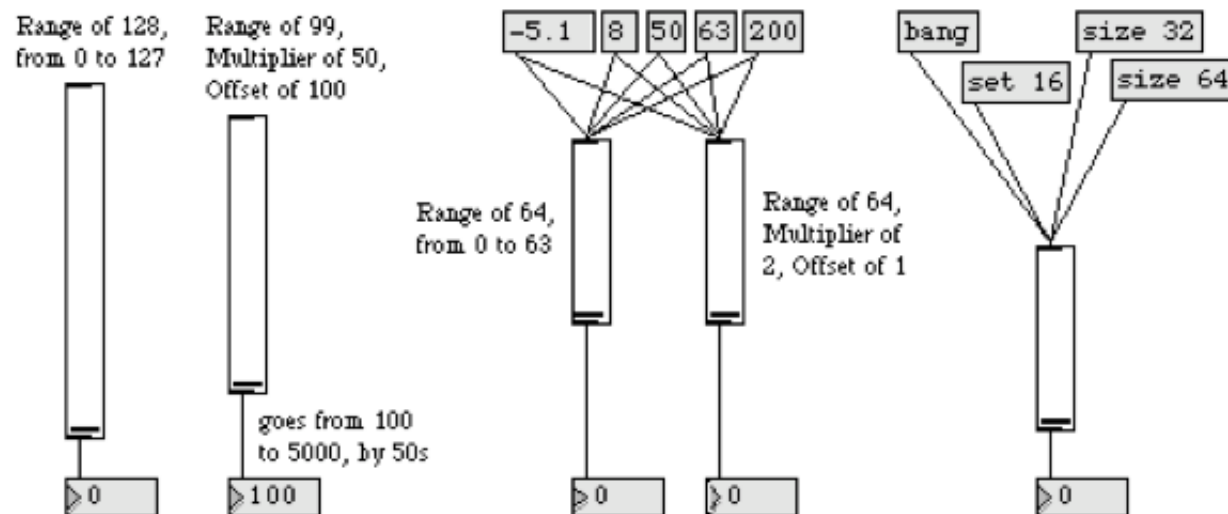
Range	Min. <input type="text" value="None"/>	Max. <input type="text" value="None"/>
	<input checked="" type="checkbox"/> No Min	<input checked="" type="checkbox"/> No Max
Options	<input type="checkbox"/> Bold <input type="checkbox"/> Draw Triangle <input type="checkbox"/> Output Only on Mouse-Up <input type="checkbox"/> Can't Change <input type="checkbox"/> Transparent	
Display Style	<input type="text" value="Decimal"/>	
Colors	<input type="text" value="Number"/> <input type="text" value="0"/> <input type="text" value="0"/> <input type="text" value="0"/>	
<input type="button" value="Revert"/>		

Inspector for a number box

Decimal (the default)	Hex	Roland octal
<input type="text" value="61"/>	<input type="text" value="3D"/>	<input type="text" value="86"/>
Binary	MIDI Note Names (C3)	Note Names C4
<input type="text" value="111101"/>	<input type="text" value="C#3"/>	<input type="text" value="C#4"/>

Messages: Numbers

- Sliders are number boxes with a UI designed for on-screen control

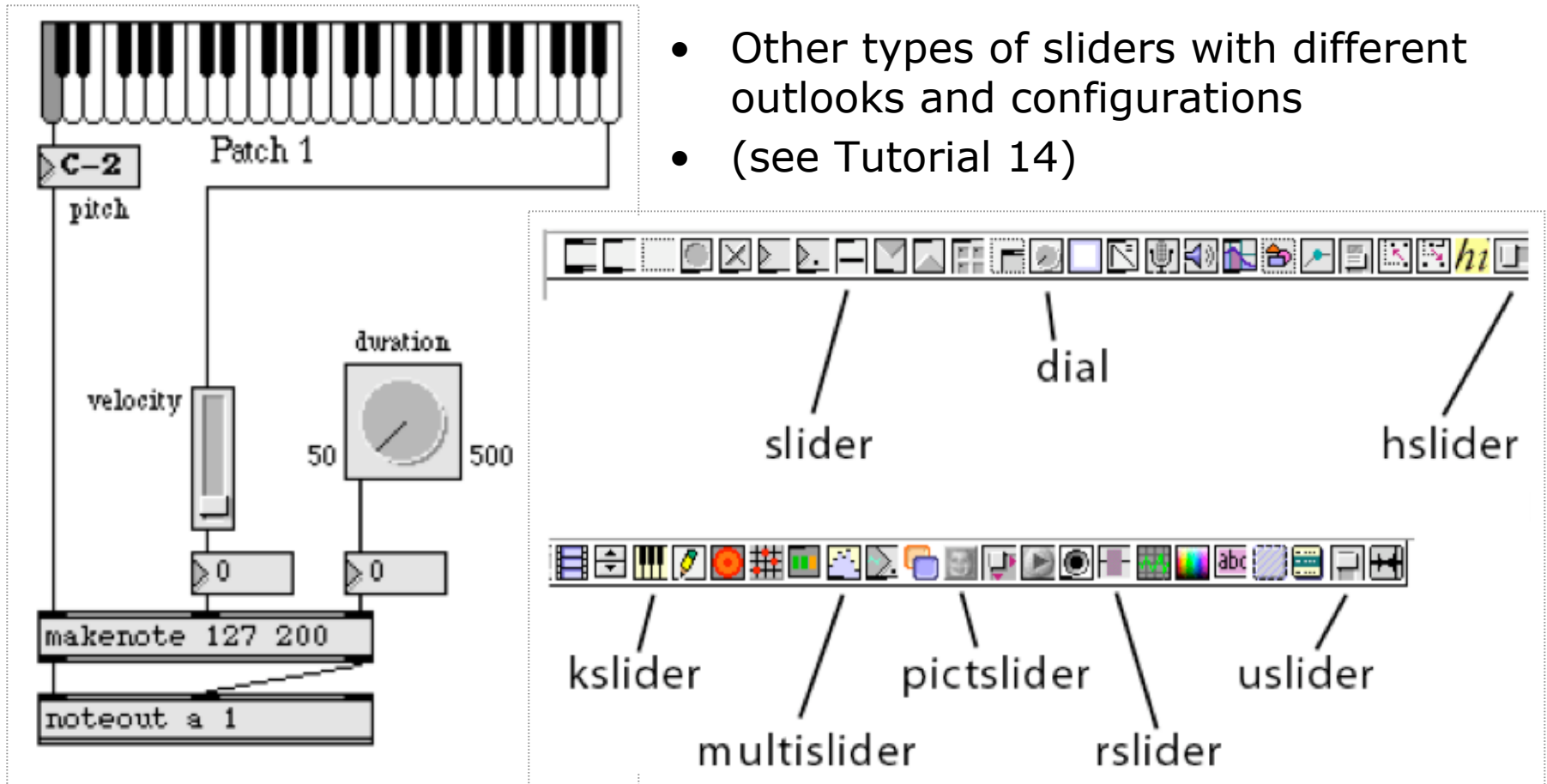


- They also allow for changes on range, resolution and offset

Slider Range	<input type="text" value="128"/>
Offset	<input type="text" value="0"/>
Multiplier	<input type="text" value="1"/>
<input type="button" value="Revert"/>	

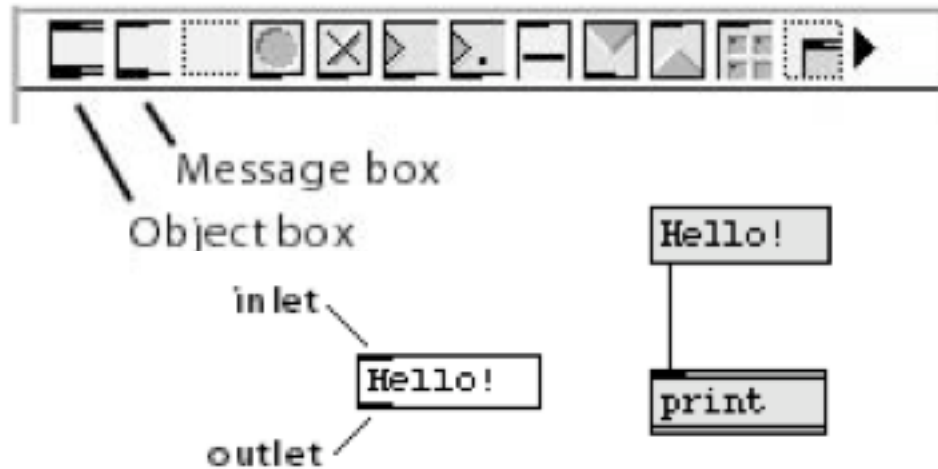
Messages: Numbers

- Other types of sliders with different outlooks and configurations
- (see Tutorial 14)

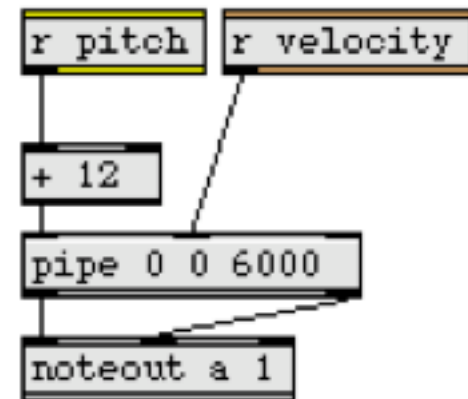
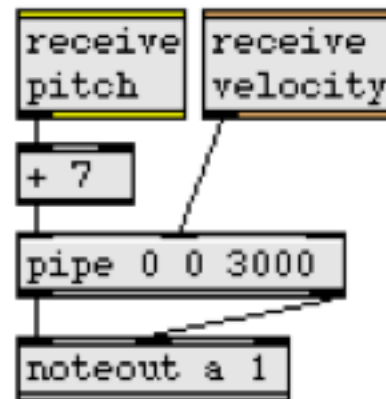
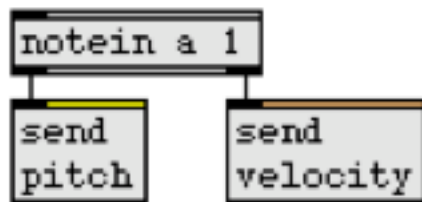
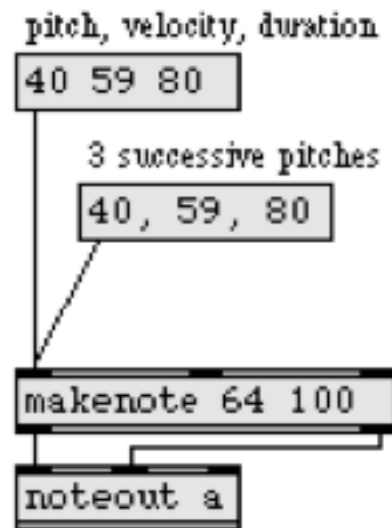


Messages: Words and lists

The message box

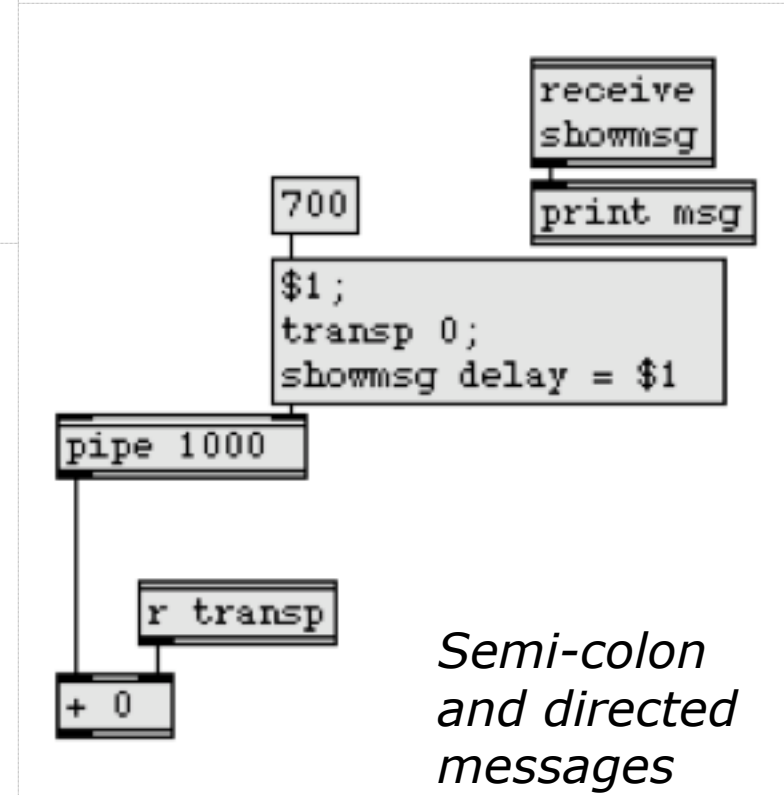
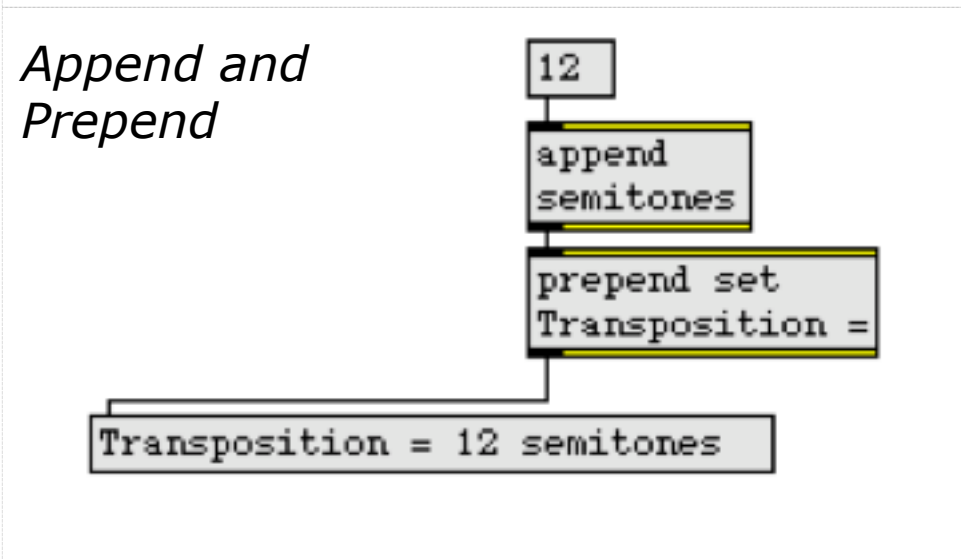
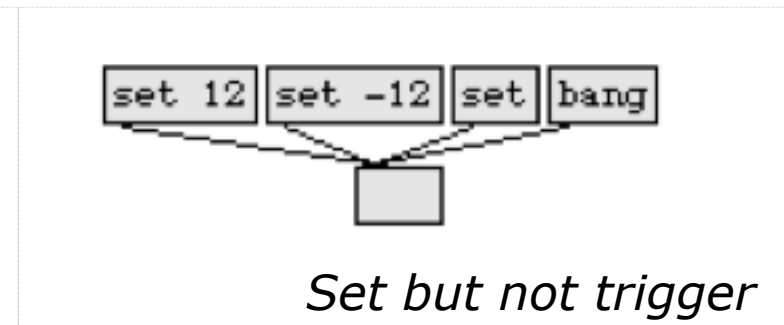
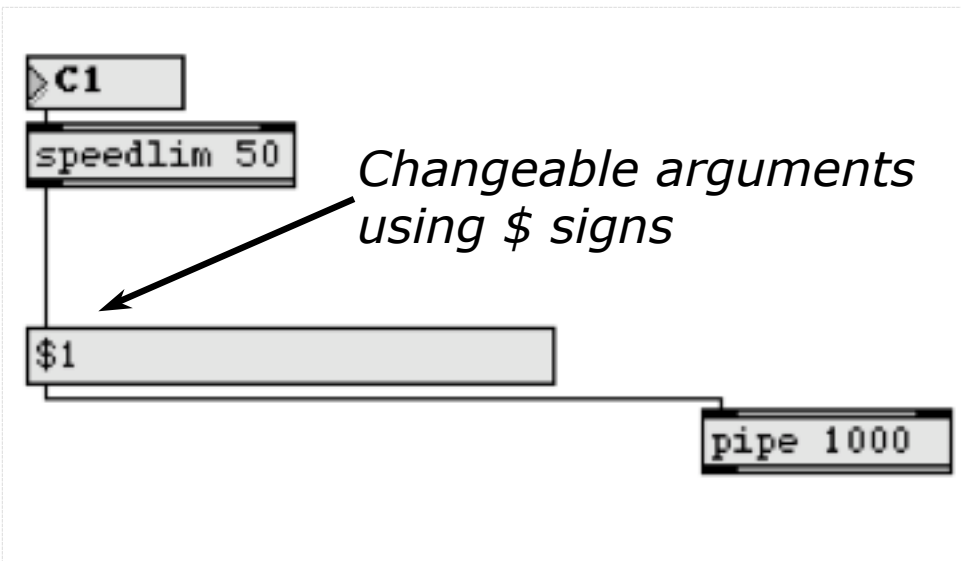


Lists and commas



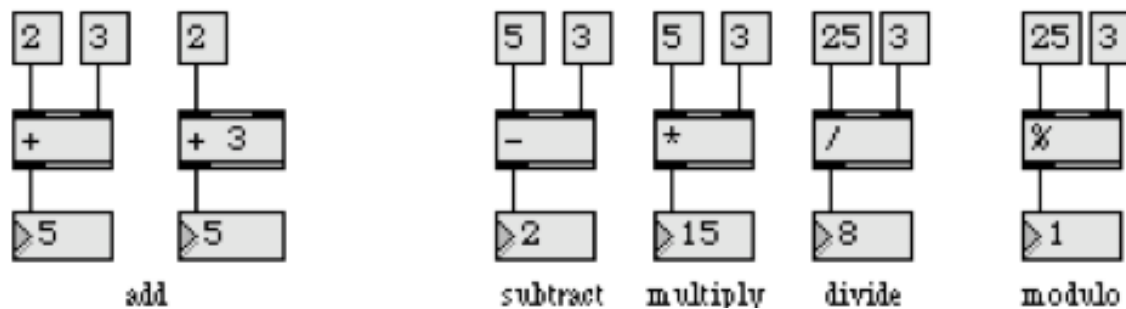
Sending and receiving

Messages: Words and lists

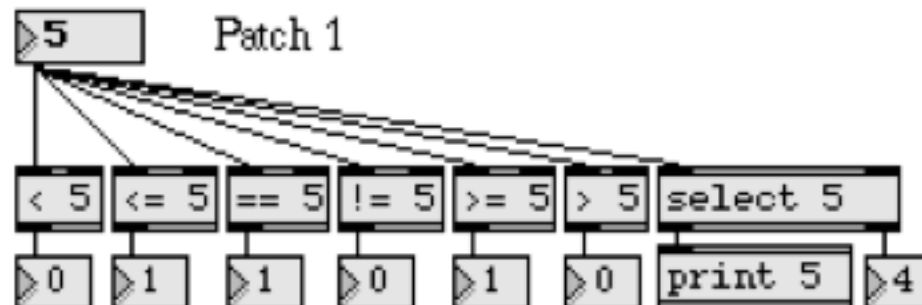


Common objects: Operators

- Arithmetic operators

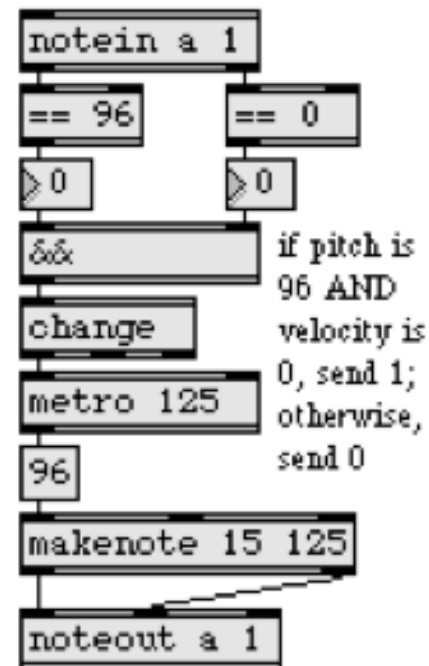
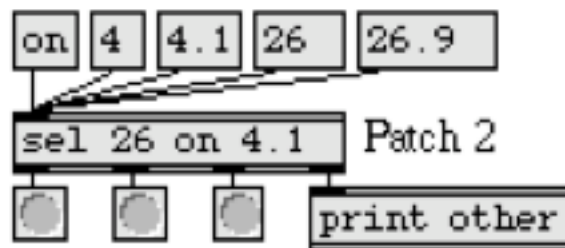


- Operations can be float if specified by the argument
- A List on the left inlet can specify all arguments
- Relational operators

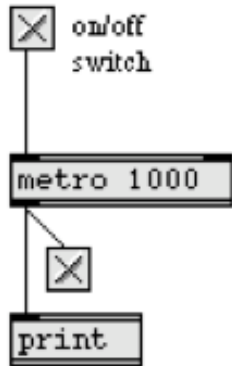


Common objects: Comparisons

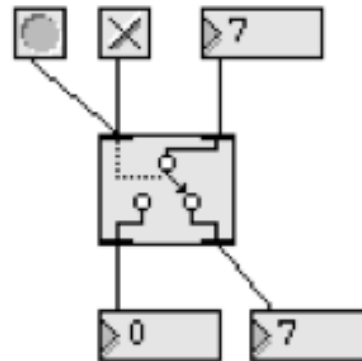
- Other objects for comparison include
- Logical operands: and (&&), or (||)
- select: which identifies specific values on a data stream
- Change: which identifies change (filters out repetitions)
- Split: bandpass filter where arguments specify range
- Speedlim: limits the speed of passing through



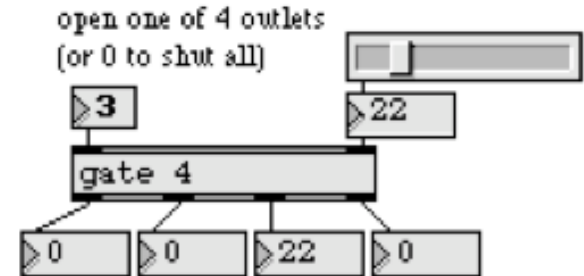
Common objects: Gates/switches



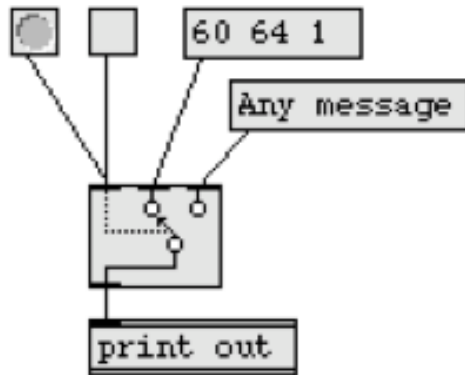
Toggle



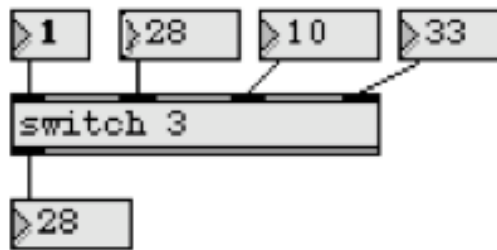
Ggate



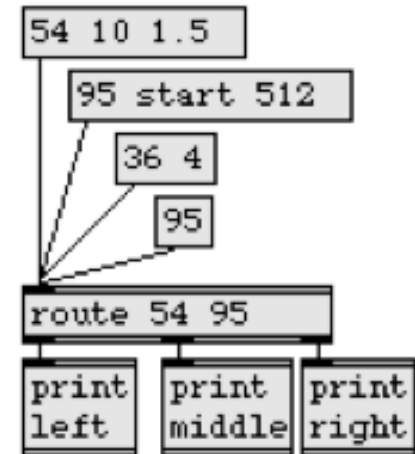
Gate



Gswitch

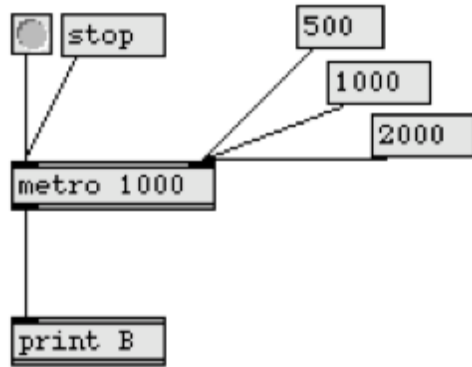


Switch

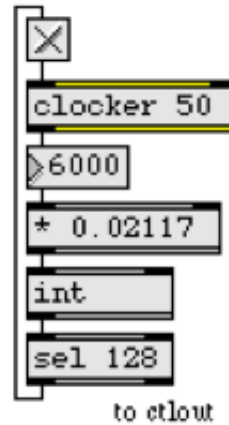


Route

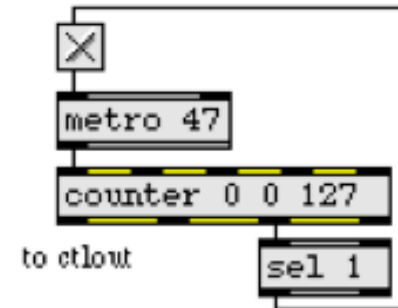
Common objects: Timers



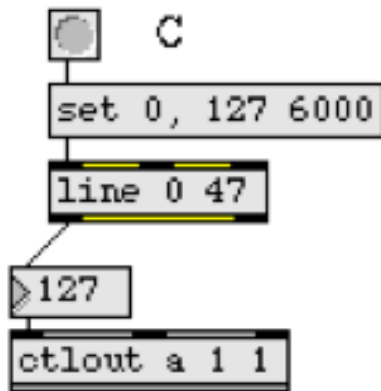
Metro



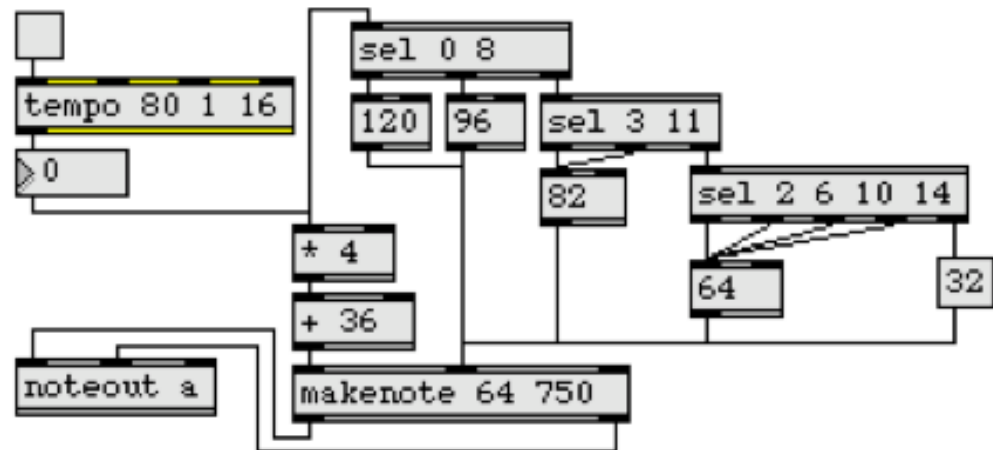
Clocker



Counter



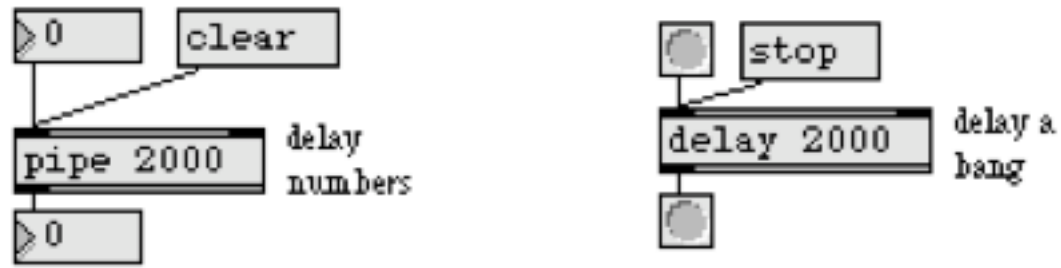
Line



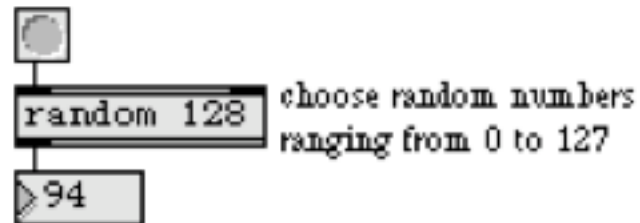
Tempo

Common objects: Delays/Random

- Other objects include delays:

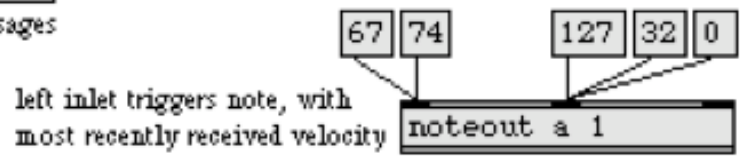
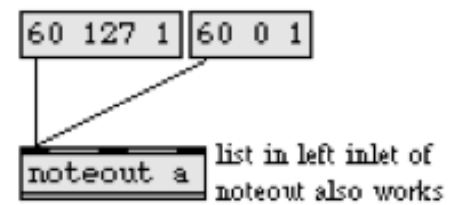
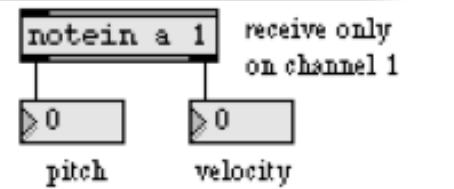
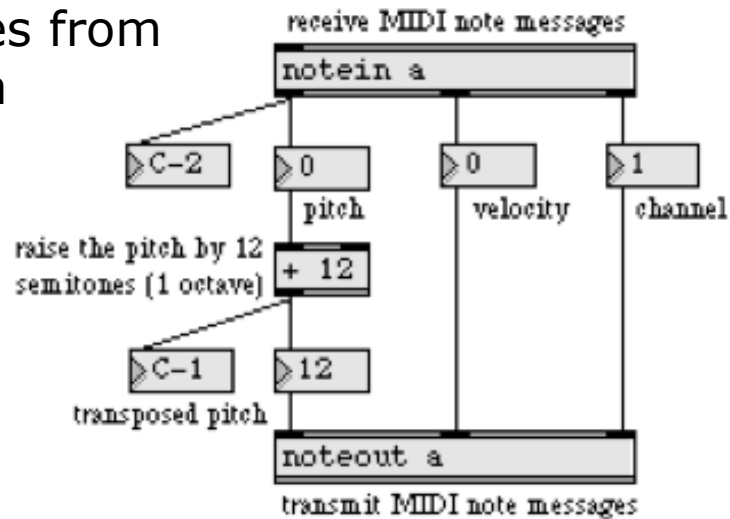
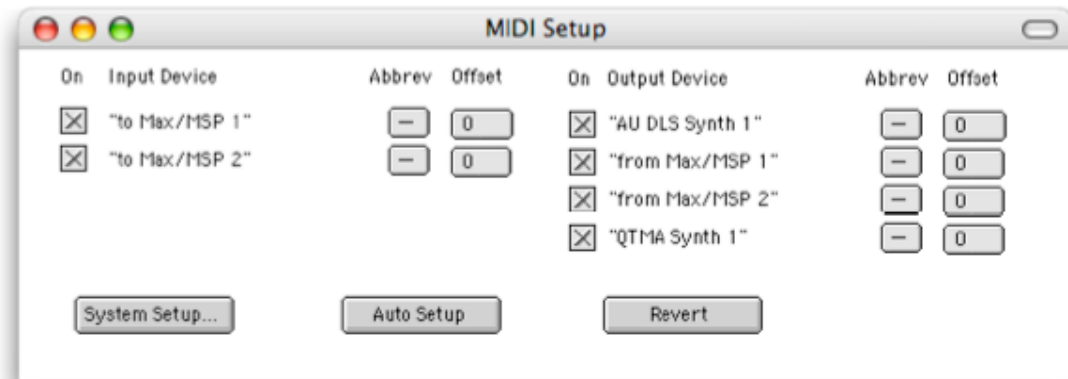


- Random number generators



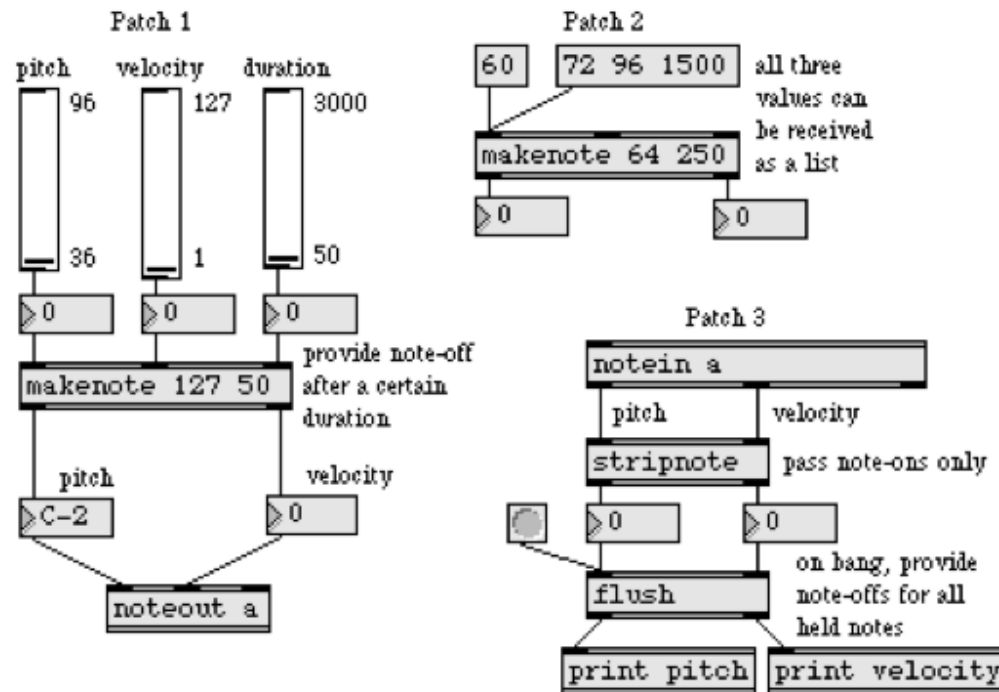
Max and MIDI

- First, we need to define instrument and channel naming conventions
- Max has objects that retrieve specific message types from the MIDI data stream
- Tutorial 12



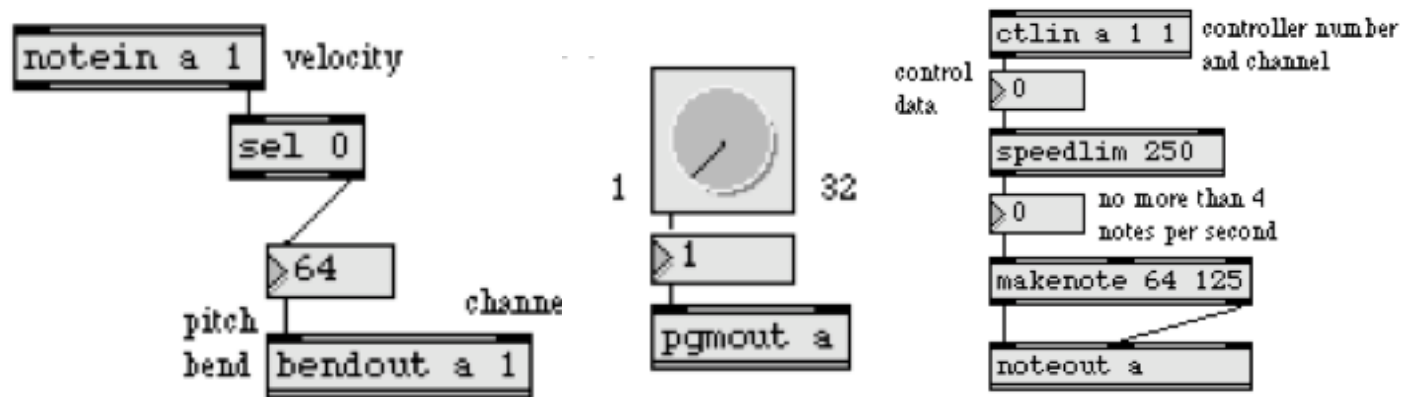
Max and MIDI

- Notes are characterized by notein as [pitch,velocity] pairs
- Note off messages are thus represented using velocity 0
- If notein data is used to control further processes in the Max patch, then those 0 values can be very disrupting
- Stripnote, makenote and flush offer solutions to this
- xnotein and xnoteout can handle note off velocities

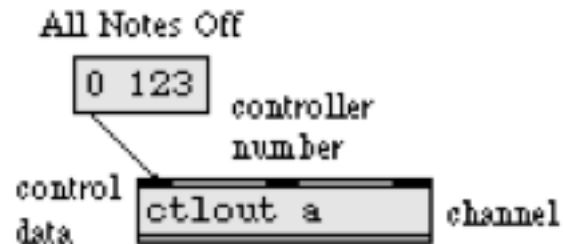


Max and MIDI

- Other objects include: bendin, bendout, ctlin, ctout, pgmin, pgmout, (and for more precision) xbendin, xbendout

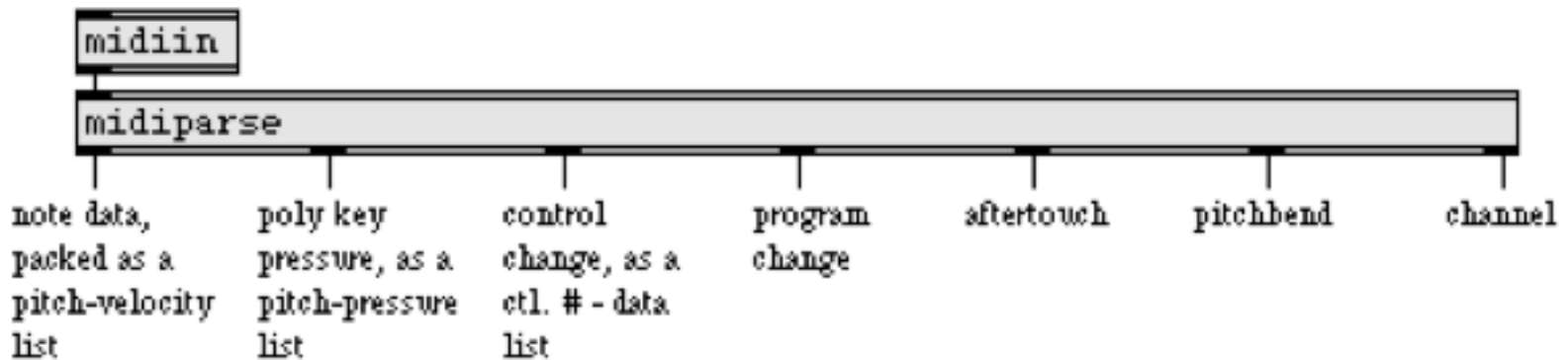


- This data can be re-assigned to other controls (Tutorial 16)
- Ctl objects can also send/receive channel mode messages

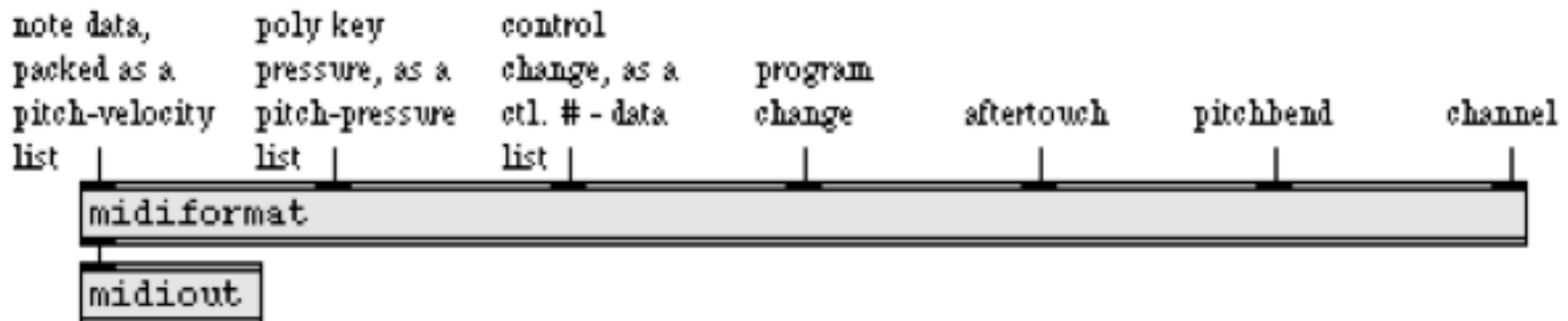


Max and MIDI

- Max is also able to handle raw MIDI data using midiin and midiout
- Midiin can be used in conjunction with midiparse, which decomposes the data into specific messages

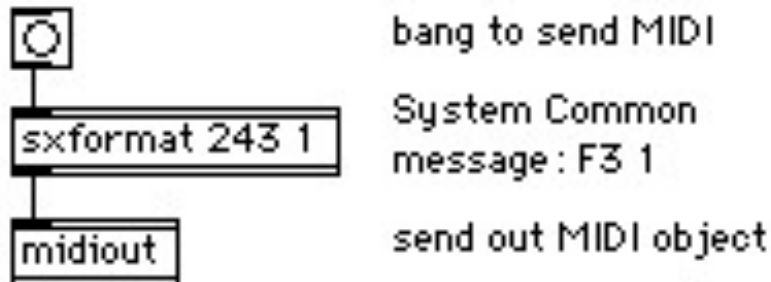
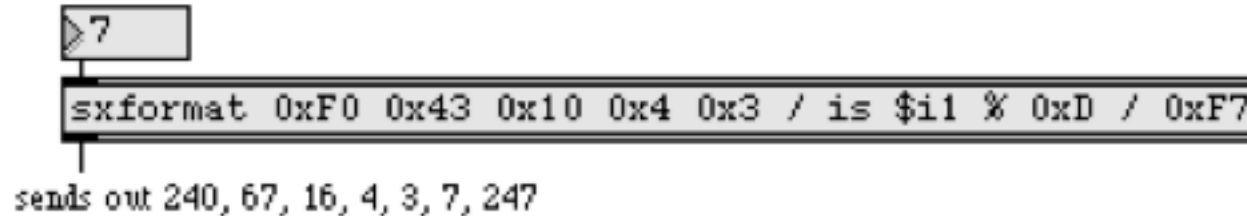


- In the opposing end, midiformat recombines these messages for transmission through midiout



Max and MIDI

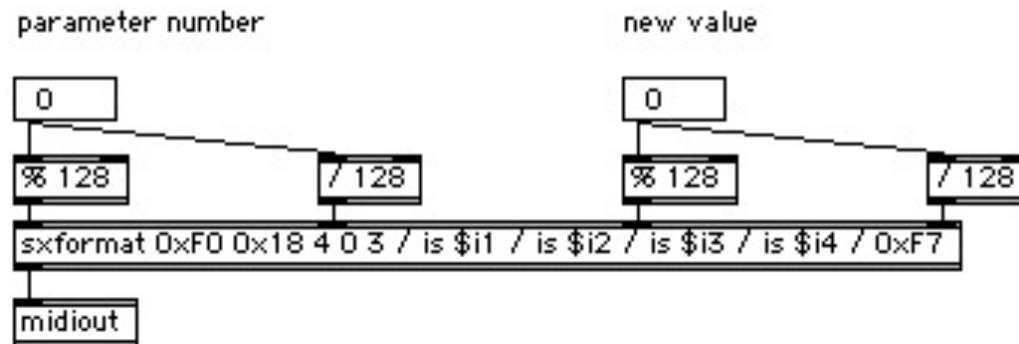
- Incoming SysEx messages are handled by sysexin
- However, outgoing SysEx messages need to be properly formatted by the user before being sent through midiout.
- Sxformat helps the user in this task
- The prefix '0x' identifies a number as hex.
- The message /is \$i1/ handles changeable inputs



Max and MIDI

- An example: changing the parameters of an E-mu Proteus
- To change a parameter on the Proteus, we indicate the parameter number in two bytes (there is a list in the manual) followed by the parameter value in two bytes. Such that the message is:
- F0 18 04 00 03 [LS par#] [MS par#] [LS val] [MS val] F7

Proteus Parameter Changer

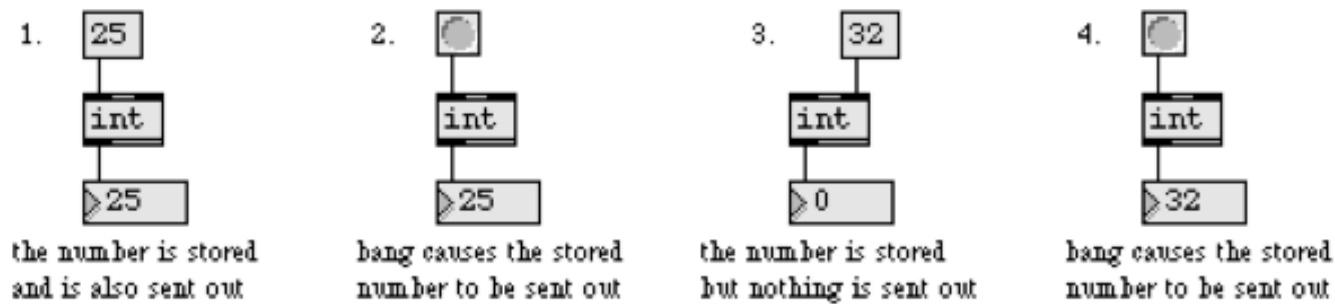


F0	system exclusive status byte
18	E-mu ID byte
04	product ID byte
dd	device ID byte
cc	command byte
...	data bytes (256 bytes/preset)
F7	EOX

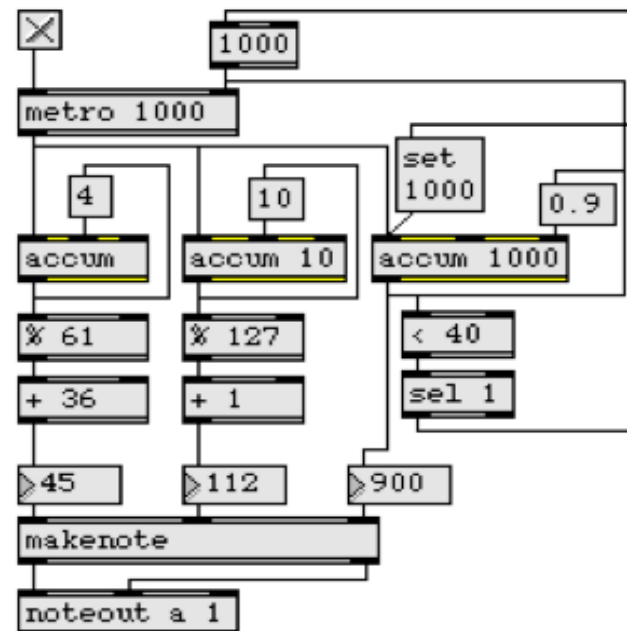
MIDI Implementation chart

Storing data

- The basic objects for data storage are int and float. Both work on the same way:



- Accum is similar but has extra functionalities for adding and multiplying to the incoming number
- See Tutorial 21

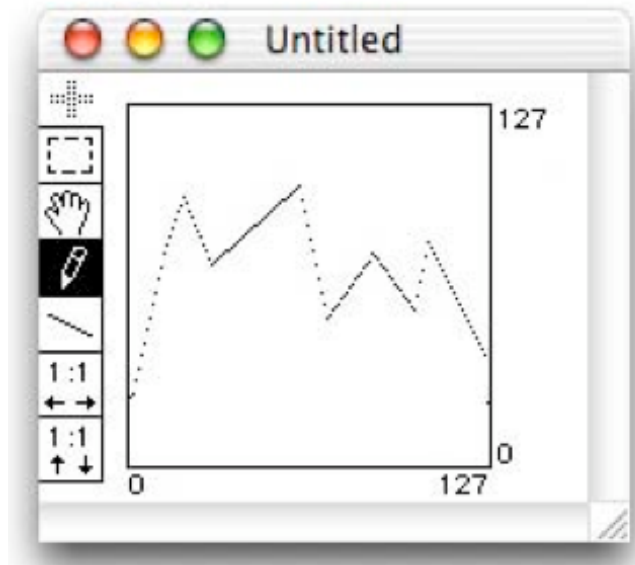


Storing data

- However it is much more powerful to be able to store and index arrays of data
- Functions like funbuff and table (Tutorial 32) can do this

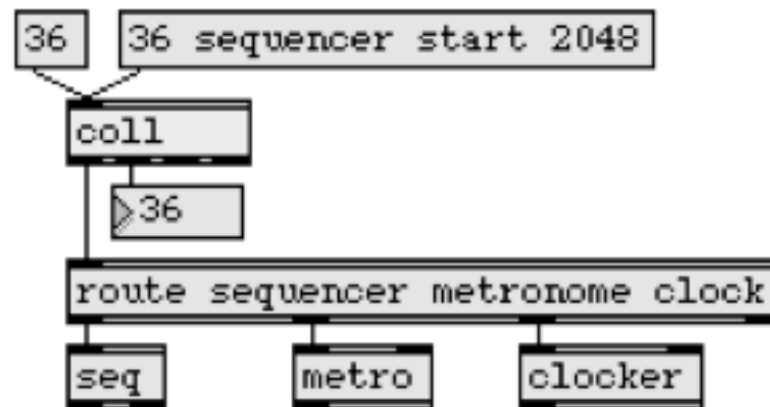


Table Size	<input type="text" value="128"/>
Table Range	<input type="text" value="128"/>
Options	<input type="checkbox"/> Save Table With Patcher
	<input type="checkbox"/> Don't Save
	<input type="checkbox"/> Use Note Name Legend
	<input type="checkbox"/> Signed Values
<input type="button" value="Revert"/>	



Storing data

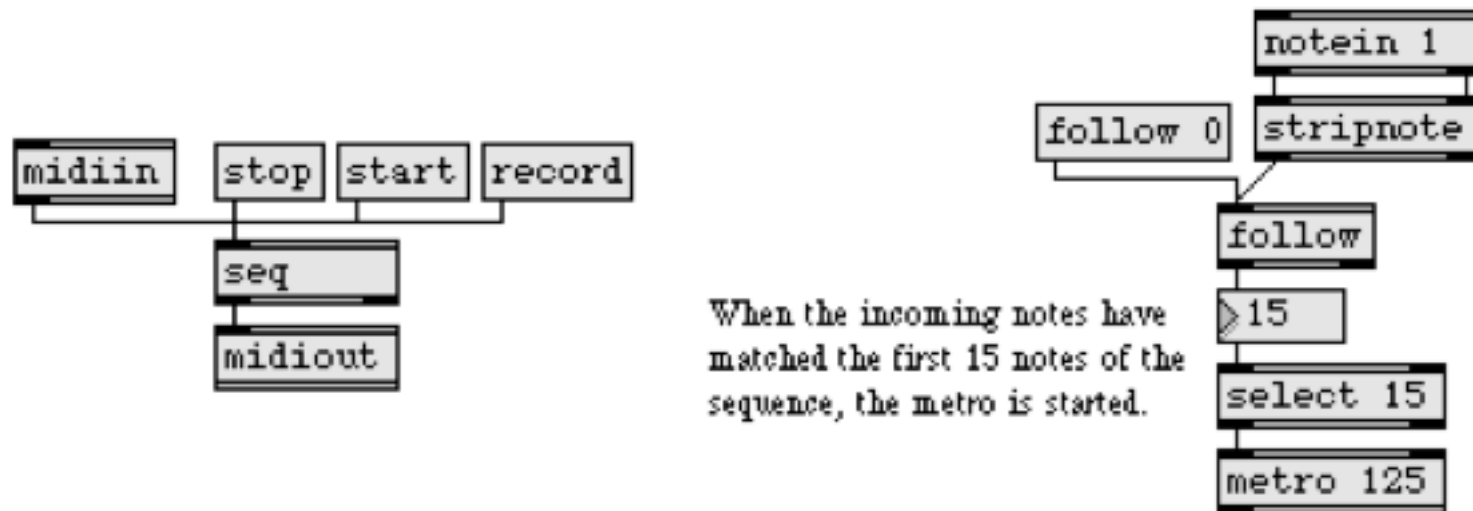
- The most versatile data structure in Max is coll (collection).
- It can store anything, indexed by any number or word



- Another useful data structures include Menu and Preset.
- Preset is able to store and recall the settings of other objects in the window
- See tutorial 37

Sequencing

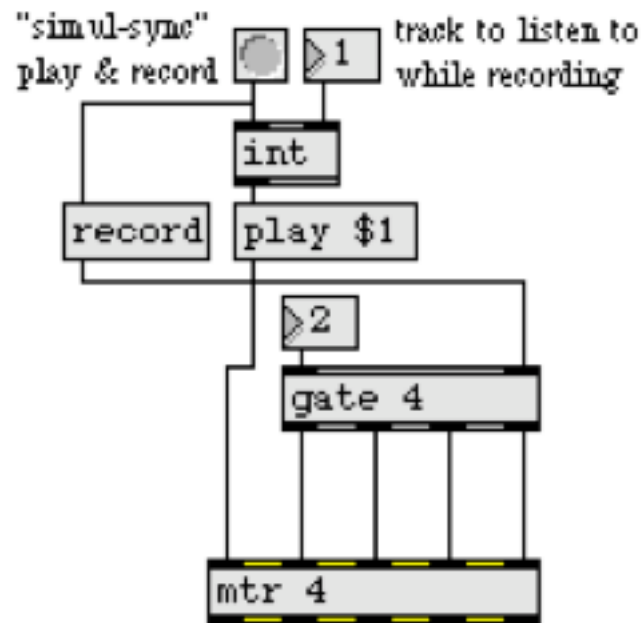
- Max uses several objects (seq, follow, mtr and detonate) for recording and playback of MIDI sequences



- Seq can record and playback a sequence, and navigate through the events of that sequence (using next, prev, etc)
- Follow is effectively a score-follower, able to follow a live rendering of the sequence that it stores (an even tollerate some mismatch)

Sequencing

- Mtr goes even further by letting you record and playback a multi-track MIDI sequence (see tutorial 36)



- (Timeline is the all powerful Max sequencer, not limited to MIDI, but able to record and playback all sorts of complex Max messages - see Tutorial 41 for more info)

References

- Cycling'74 page: <http://www.cycling74.com/>
- Max documentation: [http://www.cycling74.com/twiki/bin/view/ProductDocumentation/WebHome#Max MSP 4 6](http://www.cycling74.com/twiki/bin/view/ProductDocumentation/WebHome#Max_MSP_4_6)
- Downloads: <http://www.cycling74.com/downloads>
- Max in Wikipedia: <http://en.wikipedia.org/wiki/Max/MSP>
- Winkler, T. *Composing Interactive Music: Techniques and Ideas using Max*. MIT Press (1999)
- Pope, S.T. (Ed). *The Well-Tempered Object: Musical Applications of Object-Oriented Technology*. MIT Press (1991)