Music 209
Advanced Topics in Computer Music
Lecture 6 – Real-Time Control

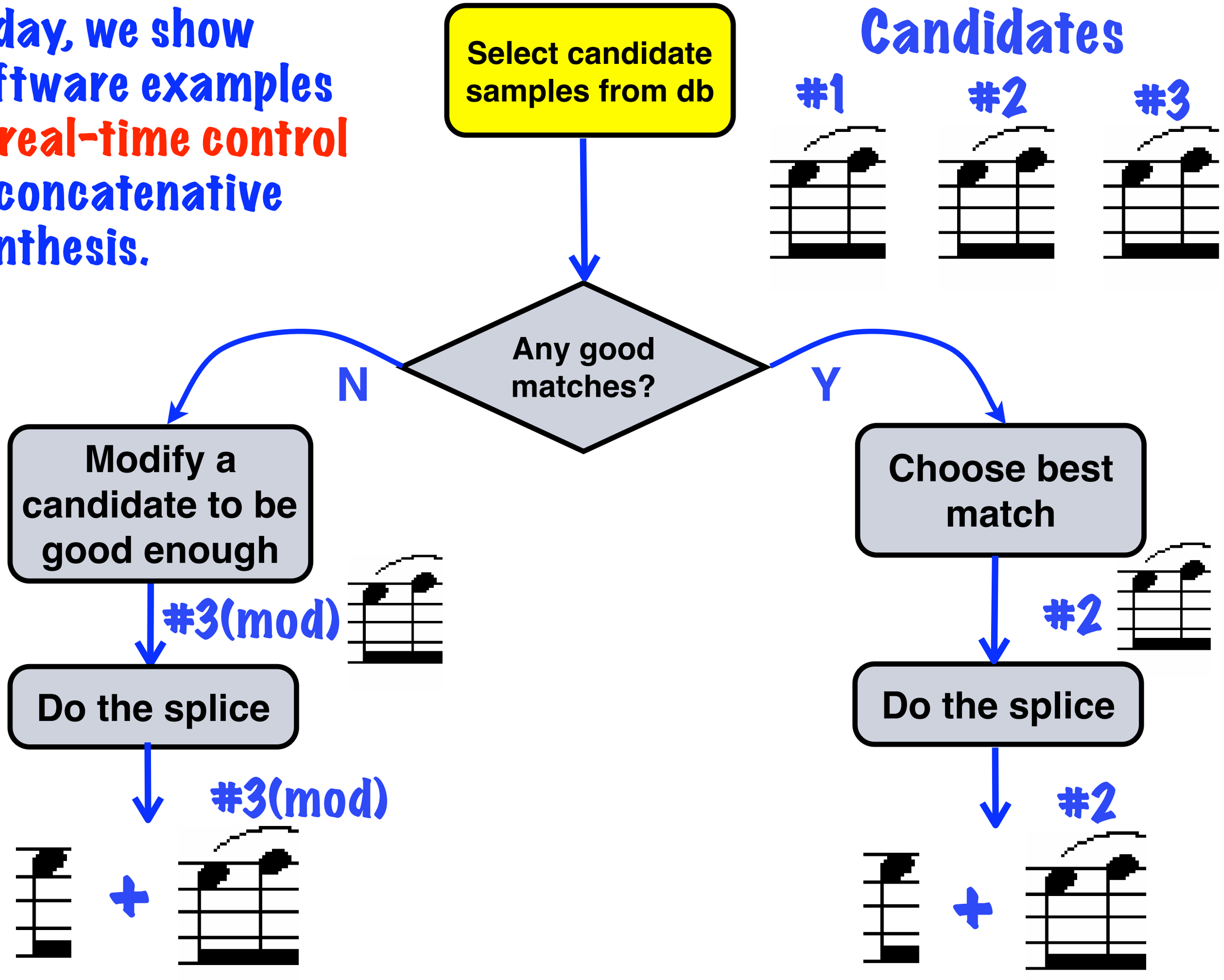
2006-2-23

Professor David Wessel (with John Lazzaro)

(cnmat.berkeley.edu/~wessel, www.cs.berkeley.edu/~lazzaro)

www.cs.berkeley.edu/~lazzaro/class/music209

Today, we show software examples of real-time control in concatenative synthesis.



Topics for today ...

- **Systems architecture**
- **Structured Audio tutorial**
- **Example: A 185 MB piano**
- **Concatenative coding techniques**

Piano Samples

MIDI cable

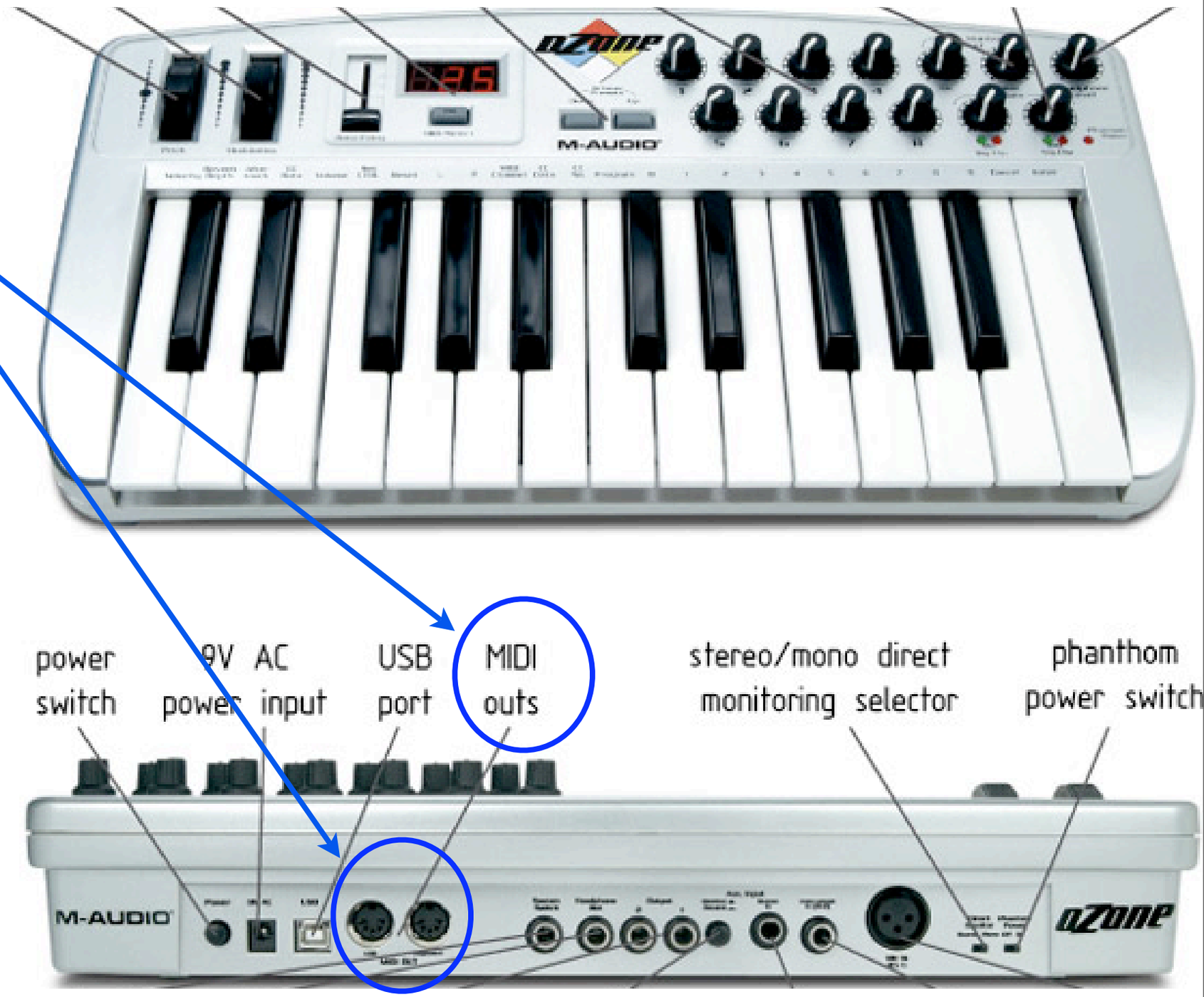
Stereo Audio Out

Dave Smith, NY AES
Convention, 1981

MIDI : A network protocol for musical instrument control

Unidirectional serial link - 31,250 Hz

Sends commands for key press, key release, knobs.



MIDI: Commands sent on a wire

Command sent on one of 16 voice channels

128 notes, 60 = Middle C

Channel Voice Messages	Bitfield Pattern
NoteOff (end a note)	1000cccc 0nnnnnnn 0vvvvvvv
NoteOn (start a note)	1001cccc 0nnnnnnn 0vvvvvvv
PTouch (Polyphonic Aftertouch)	1010cccc 0nnnnnnn 0aaaaaaa
CControl (Controller Change)	1011cccc 0xxxxxxx 0yyyyyyy
PChange (Program Change)	1100cccc 0ppppppp
CTouch (Channel Aftertouch)	1101cccc 0aaaaaaa
PWheel (Pitch Wheel)	1110cccc 0xxxxxxx 0yyyyyyy

127 strike velocities, 0 = NoteOff

Controllers send 7-bit values. Ex: controller 7 is channel volume.

Program change: 7-bit number maps channel to a timbre.

Key milestones for the project appear below.

Title	Due Date	Description	Percent of Grade
Project Abstract	March 1, 11:59 PM	A short (one or two page) description of the project. PDF or plain text format is fine -- please, no .doc files. Collaborative projects should include information on how the work will be split between team members. Email this abstract to the instructors (wessel [at] cnmat [dot] berkeley [dot] edu, lazzaro [at] eecs [dot] berkeley [dot] edu).	5 percent

Due Next Weds!

You are free to propose a project topic of your own creation. Alternatively, you may choose one of the project ideas below (click on the link for a complete description).

- Drum-related Projects
  - Creating Electronic Drum Samples from Acoustic Drum Samples
  - Tools for Automating Drum Track Arrangements
  - Timbre-Space Browsers for Drum Loops and Individual Hits
  - Realistic Retuning of Drum Sounds
  - Real-Time Performance by Retiming Drum Loops
  - Fusing Multiple Drum Hits into a Single Percept
- Wind Instrument Projects
  - Playing Horns from a Keyboard with Improved Articulation
  - Automatic Horn Phase Selection to Match a Track
  - Real-time Timbre Selection with a Wind Controller
- Computer Systems Projects
  - CoreSample: Kernel Database Services for Concatenative Synthesis
- Vocal Projects
  - Synthesis, Analysis, and Algorithmic Composition of Glossolalia Vocals
  - Lyric Design for Phrase-Based Vocal Synthesis

Structured Audio

SAOL (pronounced ”sail”)

ISO/IEC JTC 1/SC 29/WG 11 **N2503-sec5**

Date: 1999-3-10

ISO/IEC FDIS 14496-3 sec5

ISO/IEC JTC1/SC29 WG11

MPEG

MOVING PICTURE EXPERTS GROUP

ISO/IEC JTC 1/SC 29/WG11

Secretariat: Narumi Hirose

Information technology - Coding of audio-visual objects

Part 3: Audio

Section 5: Structured audio

Standardized language

Many implementations ...

Eric Scheirer (MIT Media Lab)

MPEG-4 Structured Audio: Developer Tools

By John Lazzaro and John Wawrzynek, CS Division, UC Berkeley.

MPEG-4 Structured Audio

MPEG-4 Structured Audio (MP4-SA) is an ISO/IEC standard (edited by Eric Scheirer) that specifies sound not as sampled data, but as a computer program that generates

The MP4-SA Book

We wrote an online book to show how to create audio content for MPEG 4 Structured Audio.

Links

Introductory Example

COMPILING MPEG 4 STRUCTURED AUDIO INTO C

John Lazzaro and John Wawrzynek

CS Division
UC Berkeley
Berkeley, CA, 94720
{lazzaro, johnw}@ cs.berkeley.edu

ABSTRACT

Structured Audio (SA) is an MPEG 4 Audio standard for algorithmic sound encoding, using the programming language SAOL. The paper describes a SA decoder, sfront, that translates a SAOL program into a C program, which is then compiled and executed to create audio. Performance data shows a 7.6x to 20.4x speedup compared to the SA reference MPEG decoder.

sfront

Download the latest version of sfront, a translator that converts MP4-SA files into efficient C programs that generate audio for rendering, interactive and network applications.

Sfront is written by John Lazzaro and John Wawrzynek, and is freely redistributable under the terms of the GNU Public License.

The sfront reference manual describes how to install and use the program. Developers can learn how to add control and audio drivers to sfront, as well as learn about the internals of sfront and the C programs it creates.

Where MIDI meets SAOL

Each MIDI NoteOn launches a new instance of the instr bound to the program number.

A SAOL "Instr"

```
instr sine (pitch, vel) preset 0 {
  // code goes here
}
```

Instance passed in NoteOn number and velocity.

MIDI program number

NoteOff schedules instance to terminate

Sets program number for a MIDI channel

Channel Voice Messages	Bitfield Pattern
NoteOff (end a note)	1000cccc 0nnnnnnn 0vvvvvvv
NoteOn (start a note)	1001cccc 0nnnnnnn 0vvvvvvv
PChange (Program Change)	1100cccc 0ppppppp

Benefits

The language run-time does real-time scheduling for you.
All you do is supply behavior code.

```
instr sine (pitch, vel) preset 0 {

  // code goes here

}
```

No extra code for polyphony.

The language makes parallelism explicit:
SAOL code is multi-core ready.

Channel Voice Messages	Bitfield Pattern
NoteOff (end a note)	1000cccc 0nnnnnnn 0vvvvvvv
NoteOn (start a note)	1001cccc 0nnnnnnn 0vvvvvvv
PChange (Program Change)	1100cccc 0ppppppp

Execution model

```
Time (s)    Cycle      Pass      X-#


0.990000    k-cycle
            a-cycle
0.990025    a-cycle
0.990050    a-cycle
0.990075    a-cycle
  ...

0.999925    a-cycle
0.999950    a-cycle
0.999975    a-cycle

1.000000    k-cycle    i-pass    --
                       k-pass      |
            a-cycle    a-pass      |
1.000025    a-cycle    a-pass      | 1
1.000050    a-cycle    a-pass      |
1.000075    a-cycle    a-pass      |
  ...         ...        ...       |
1.009975    a-cycle    a-pass    --


1.010000    k-cycle    k-pass    --
            a-cycle    a-pass      |
1.010025    a-cycle    a-pass      |
1.010050    a-cycle    a-pass      | 2
1.010075    a-cycle    a-pass      |
  ...         ...        ...       |
1.019975    a-cycle    a-pass    --


1.020000    k-cycle    k-pass    --
            a-cycle    a-pass      |
1.020025    a-cycle    a-pass      |
1.020050    a-cycle    a-pass      | 3
1.020075    a-cycle    a-pass      |
  ...         ...        ...       |
1.029975    a-cycle    a-pass    --
```

Birth

No ipass

```
global {
  srate 44100;
  krate 1050;
  }
```

Global block sets audio and control sample rates.

```
instr sine (pitch, vel) preset 0 {

  // Variable declarations
```

Benefit: Keeps code for all timescales in one place.

```
  // Code that runs once, at
  // instantiation.
```

I-RATE
I-PASS

"spoken words"

```
  // Code that runs at the start
  // of each control cycle.
```

"phonemes"

K-RATE
K-PASS

```
  // Code that runs at the audio
  // sample rate.
```

"waveform"

A-RATE
A-PASS

```
}
```

Example 1: One Note Fits All ...

Plays one piano sample across entire keyboard.
Each key plays same pitch.

Samples read from disk and locked into RAM during startup.

```
global {
    srate 44100;
    krate 1050;

    table right_060_mf(sample, -1,
      "060_C3KM56_M.wav");
}
```

Variable name used in SAOL program code.

Piano sample file on disk.

Instr declaration + i-rate code

```
instr piano_proto (pitch, vel) preset 0 {
```

How global {} variables are made visible in an instr.

```
  imports exports table right_060_mf;
  ivar rel_time, full_scale, volume;
  ksig k, rel;
  asig i;

  // **********************
  // computed during i-pass
  // **********************


  rel_time = 0.250;
  full_scale = 0.25;
  volume = full_scale*(vel/127);   // NoteOn velocity
                                   // scales volume
```

Assignments to a variable happen at the rate indicated by these keywords (ivar, ksig, asig).

"vel" is the velocity value of the NoteOn that created the instance.

Instr k-rate code

A "standard name" (built-in variable).

```
// ************************
// computed during k-pass
// ************************
```

released is 1 if instance is slated for termination before next k-pass.

```
if (released && !rel)     // Add release time when NoteOff occurs
  {
    rel = 1;
    extend(rel_time);
  }
```

We postpone termination so that we can fade note out.

extend() is a SAOL command.

```
if (!rel &&
 (k > ftlen(right_060_mf) - rel_time*s_rate - 2*(s_rate/k_rate)))
  {
    turnoff; // Force NoteOff before we run out of samples
  }

k = k + (s_rate/k_rate);
```

Is the sample ready to run out?

If so, we "force" a NoteOff by using the turnoffcommand. Next kpass, released will be 1.

Instr a-rate code

Demo

```
// **************************
// computed during a-pass
// **************************

if (!rel)     // Attack and sustain portion of note
  {
    output(volume*tableread(right_060_mf, i));
  }

else          // Release envelope after NoteOff
  {
    output(aline(volume, rel_time, 0)*
           tableread(right_060_mf, i));
  }

i = i + 1;
```

output Sum audio sample value onto the instr's "output bus".

NoteOn: Scale by velocity constant.

NoteOff: Scale by fadeout envelope.

tableread(..., i) Interpolated read of the i'th sample of the table.

Sample Databases

Sample database

But how do we conveniently access 100s of samples in a SAOL program?

```
global {

  table left_024_mf
    (sample, -1, "024_C0KM56_M.wav");

  table right_024_mf
    (sample, -1, "024_C0KM56_M.wav");

table left_031_mf
  (sample, -1, "031_G0KM56_M.wav");

table right_031_mf
  (sample, -1, "031_G0KM56_M.wav");

            [...]
```

Tablemaps

```
instr full (pitch, vel) preset 0 {

imports exports table low;
imports exports table mid;
imports exports table hi;

                0    1    2
tablemap set(low, mid, high);

output(tableread(set[1], i));

}
```

Index values

Reads from "mid" table