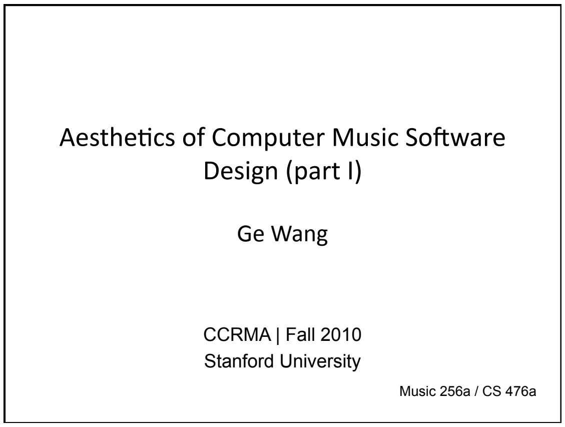
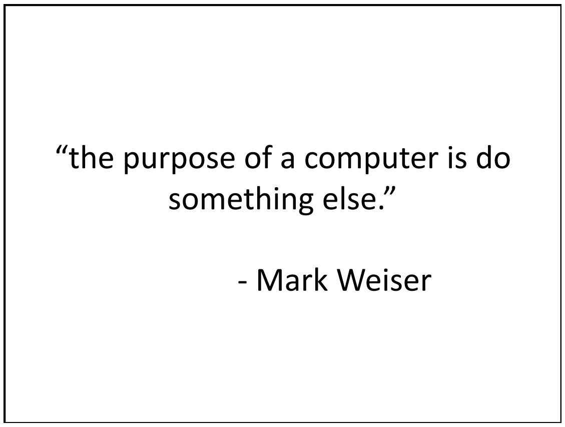


Aesthetics of Computer Music Software Design (part I)

Ge Wang

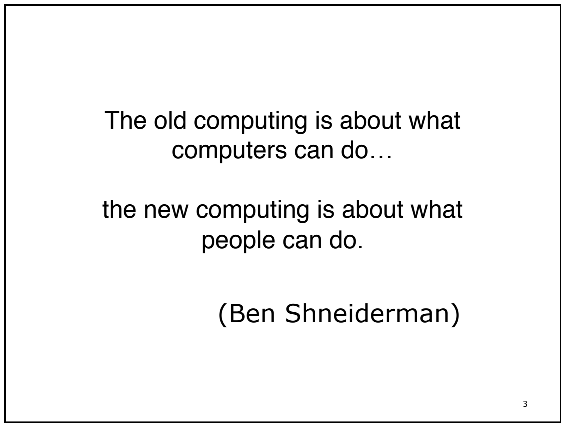
CCRMA | Fall 2010
Stanford University

Music 256a / CS 476a



“the purpose of a computer is do something else.”

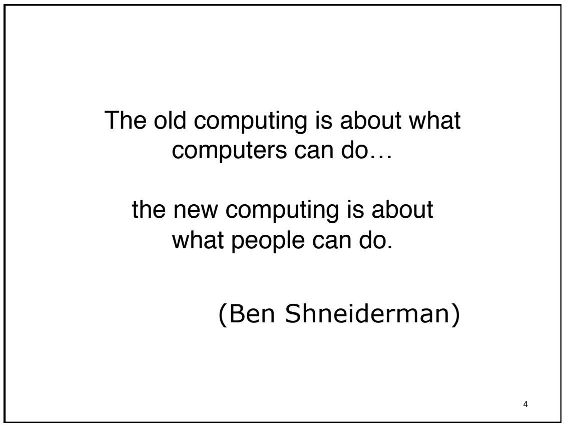
- Mark Weiser



The old computing is about what computers can do…

the new computing is about what people can do.

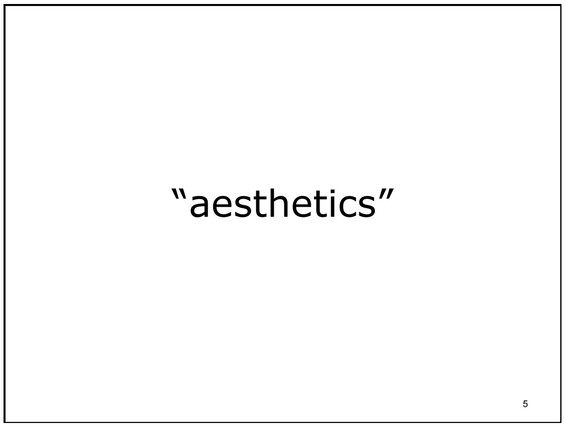
(Ben Shneiderman)



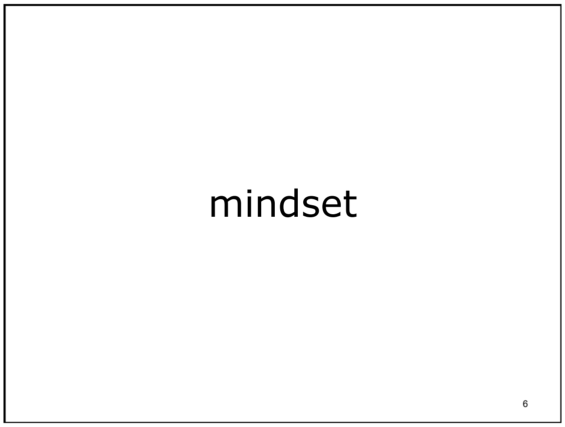
The old computing is about what computers can do…

the new computing is about what people can do.

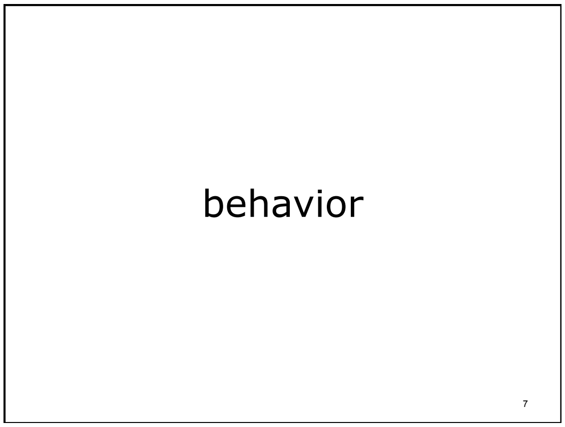
(Ben Shneiderman)



“aesthetics”

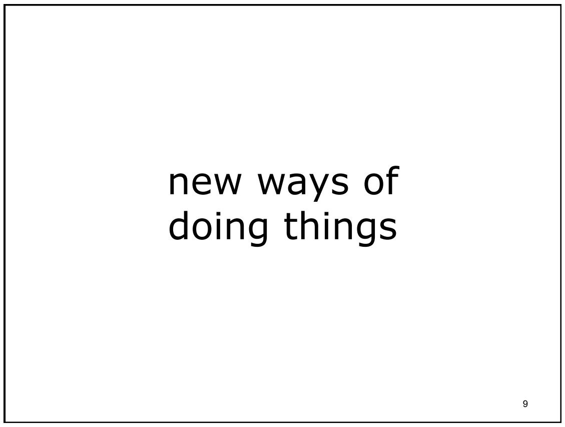


mindset

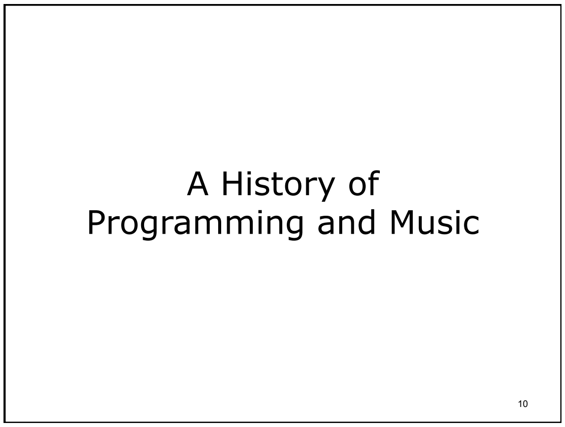


behavior

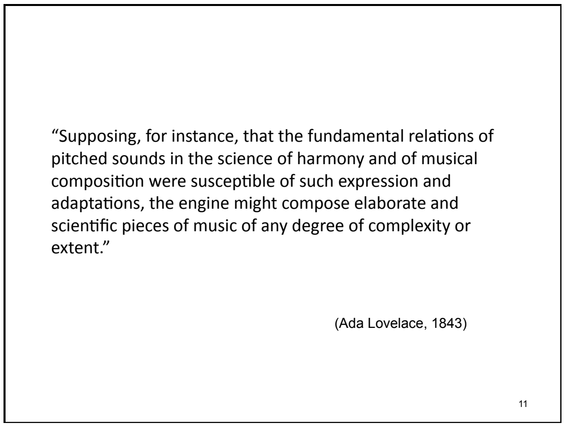
feel



new ways of doing things

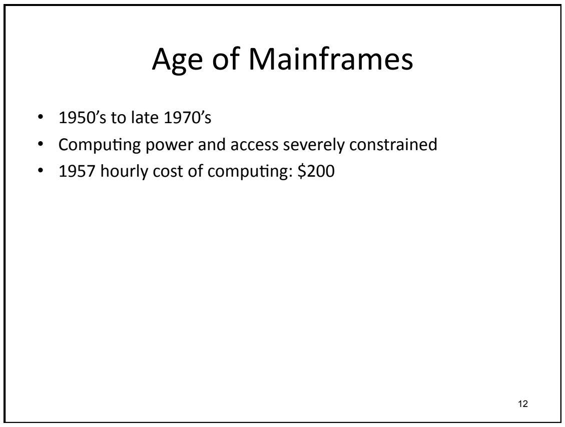


A History of Programming and Music



"Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."

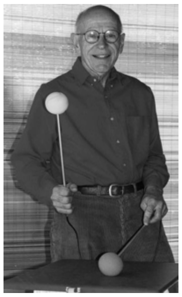
(Ada Lovelace, 1843)



Age of Mainframes

- 1950's to late 1970's
- Computing power and access severely constrained
- 1957 hourly cost of computing: $200

Rise of MUSIC-N



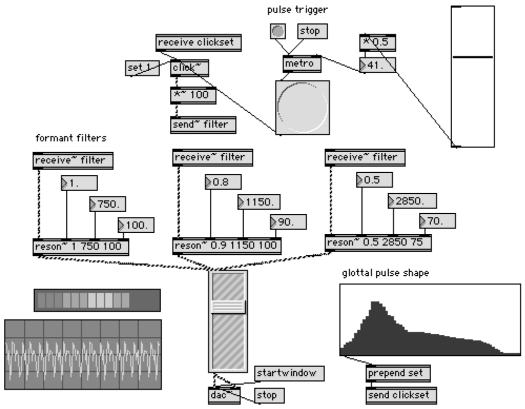
- Max Mathews
- Unit Generator
- Patches
- Orchestra vs. Score

Early MUSIC-N

- MUSIC I to V (Mathews)
- MUSIC IV-B (Winham and Howe)
- MUSIC-10 (Chowning, Moore)
- MUSIC 11 (PDP-11)
- MUSIC 360 (Vercoe)
- Cmusic (F. Richard Moore)

Modern Environments

- Max/MSP (Puckett, Zicarrelli)
  - graphical patching
- Nyquist (Dannenberg)
  - LisP, combines orc + sco
- SuperCollider (McCartney)
  - Smalltalk/C, client/server
- Csound (Vercoe et al.)
- Common Lisp Music (Schottstaedt)



Distro's and Libraries

- CARL (Computer Audio Research Lab)
  - "UNIX for Music", open-source
- Cmix (Lansky)
  - Flexible library for mixing audio, DSP, MINC
- Synthesis Toolkit (Cook and Scavone)
  - C++, Physical Modeling, Real-time

Languages for Music

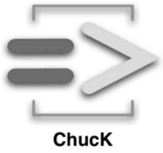
- Formula (Anderson and Quivila)
  - Forth Music Language
  - Control signals
  - Warpable time-mechanism
- Haskore (Hudak et al)
  - Modules in Haskell
  - For describing *music* (mostly western), not *sound*

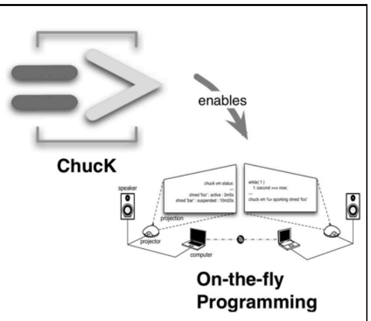
Post-modern Environments

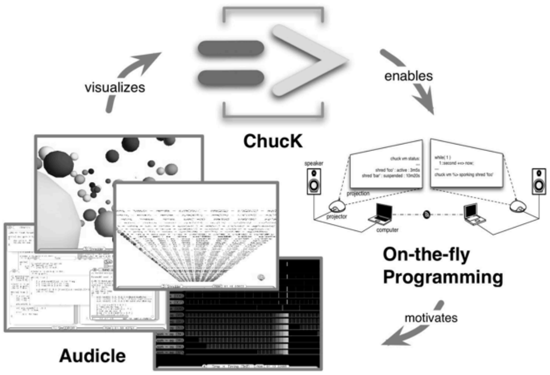
(rise of homebrew software, live coding)

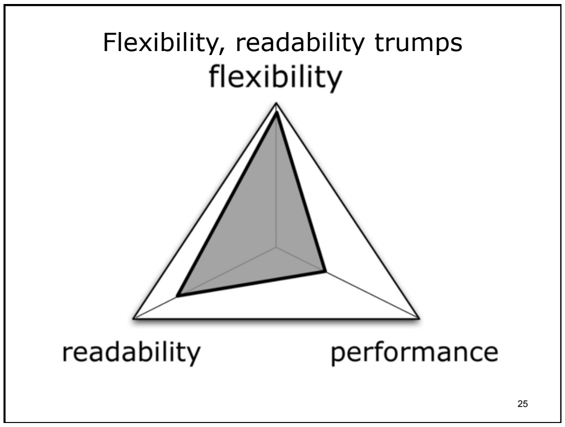
- Proliferation of programming environments
  - Lower barriers of entry
  - End users **=>** developers **=>** end users
- TOPLAP
  - Temporary organization for proliferation of live audio programming
- JITLIB, Impromptu, feedback.pl, Fluxus…
  - Many more

ChucK

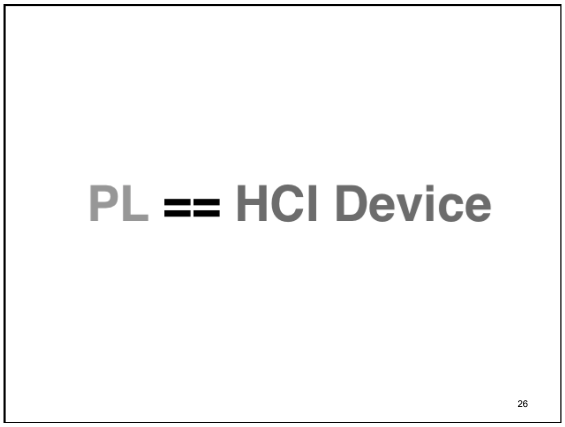




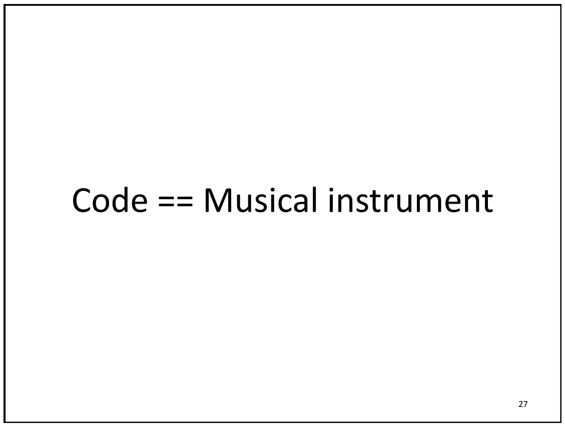




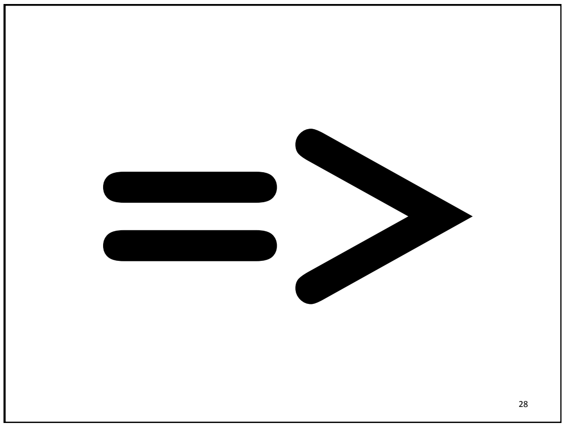
Flexibility, readability trumps



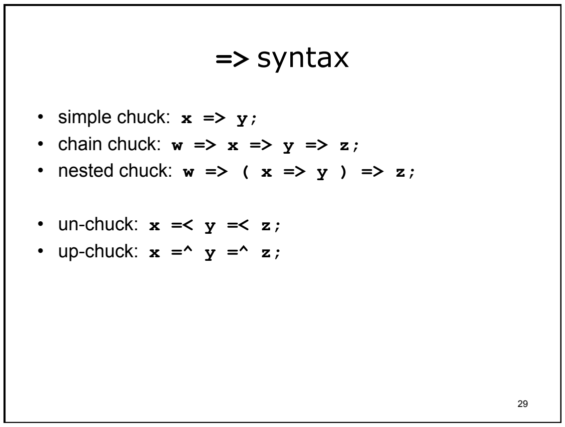
PL == HCI Device



Code == Musical instrument



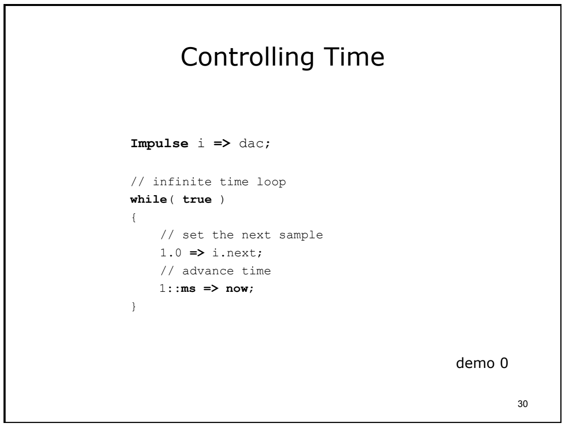
=>



=> syntax

- simple chuck: `x => y;`
- chain chuck: `w => x => y => z;`
- nested chuck: `w => ( x => y ) => z;`

- un-chuck: `x =< y =< z;`
- up-chuck: `x =^ y =^ z;`



Controlling Time

```
Impulse i => dac;

// infinite time loop
while( true )
{
    // set the next sample
    1.0 => i.next;
    // advance time
    1::ms => now;
}
```

demo 0

ChucK Timing Constructs

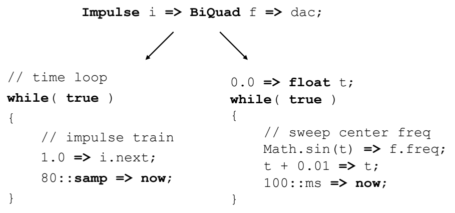
- **`dur`** is a native type
  - units:
    `samp, ms, second, minute, hour, day, week`
  - arithmetic:
    `1::second + 200::ms => dur quarter;`
- **`time`** is a native type
  - **`now`** keyword holds current chuck time
  - arithmetic:

```
5::second + now => time later;
while( now < later ) {... }
```

Advancing Time

- ChucK time stands still until you “advance” it
- two semantics for advancing time
  - chuck to now
    `1::second => now;`
  - wait on event
    `event => now;`
- you are responsible for keeping up with time
- timing embedded in program flow
- time == sound

Concurrent Audio Programming



```
Impulse i => BiQuad f => dac;
```

```
// time loop
while( true )
{
    // impulse train
    1.0 => i.next;
    80::samp => now;
}
```

```
0.0 => float t;
while( true )
{
    // sweep center freq
    Math.sin(t) => f.freq;
    t + 0.01 => t;
    100::ms => now;
}
```

Concurrency

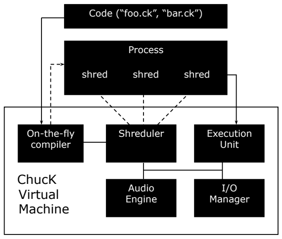
- implemented using “shreds”
  - resemble non-preemptive threads
- automatically synchronized by time!
- possible to easily write truly parallel, sample-synchronous audio code
- can work at low and high level
  - fine granularity == power and control
  - arbitrary granularity == flexibility and efficiency
- a solution to the control-rate issue

ChucK Concurrency + Timing

- “strongly-timed”
- no loss of generality (any ugen any time)
- staying “in the language”
  - express more from within the language
  - greatly reduce need for externals
- provides natural modularity for on-the-fly programs

ChucK Virtual Machine

ChucK Virtual Machine



Code ("foo.ck", "bar.ck")

Process

shred shred shred

On-the-fly compiler

Shreduler

Execution Unit

ChucK Virtual Machine

Audio Engine

I/O Manager

Audio Computation

- controlled by shreds
- computes audio outside of shreds
  - traverses the global UGen graph from well-known sinks, such as 'dac'
- UGens and UAnae cache the latest computation

On-the-fly Programming

(coding while running with scissors)

on-the-fly programming:

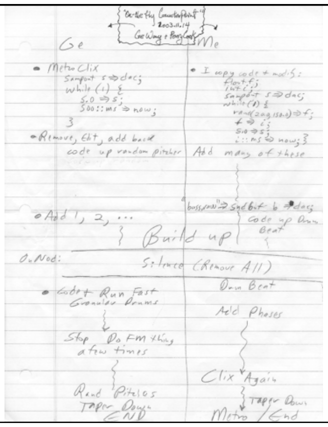
(n.) the act of modifying the logic and structure of a program during runtime, for the purpose of rapid experimentation, and exerting expressive control. (also live coding)

The League of Automatic Composers (1974)

On-the-fly programmers (2004)

On-the-fly programmers (2008)

http://chuck.cs.princeton.edu/



"On-the-fly Counterpoint"
2003.11.14
Ge Wang + Perry Cook

Ge

- MetroClix
```
Sampout s => dac;
while (1) {
    5.0 => s;
    500::ms => now;
}
```
- Remove, Edit, add back
  code up random pitcher

- Add 1, 2, ...

} Build up

On Nod: Silence (Remove All)

- Code + Run Fast
  Granular Drums

Stop Do FM thing
a few times

Rand Pitches
Taper Down
END

Me

- I copy code + modify:
```
float f;
int i;
Sampout s => dac;
while (1) {
    rand(20.0, 150.0) => f;
    f => i;
    5.0 => s;
    i::ms => now; }
```
Add many of these

"bass.raw" => sndbuf b => dac;
code up Drum
Beat

Drum Beat

Add Phases

Clix Again

Taper Down

Metro / End

Power Tools Can Maim

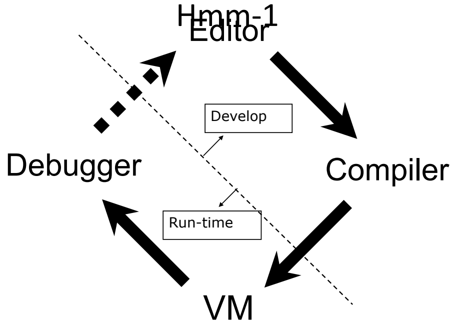
- power to spork many, many, many shreds
- power to precisely synchronize shreds
- edit and re-spork
- query for status…

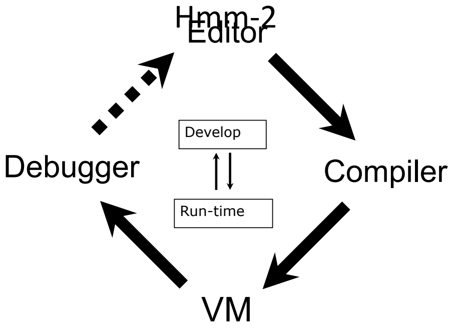
But, Oops…

- which shred is which?
- which version of the edited code did I save?
- who is using all the processor cycles?
- what is the relative timing of the shreds?
- who is clipping?

The Audicle

Hmm-1





Hmm-2
Editor

Develop

Run-time

Debugger

Compiler

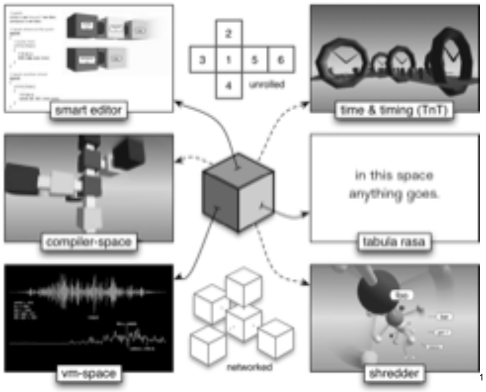
VM

49

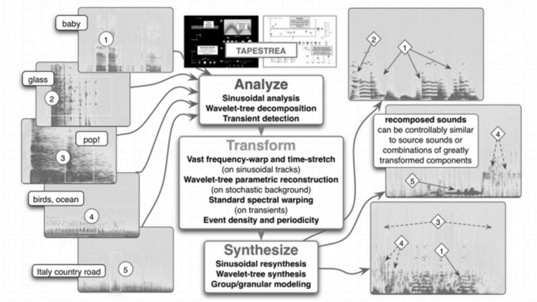
The Audicle

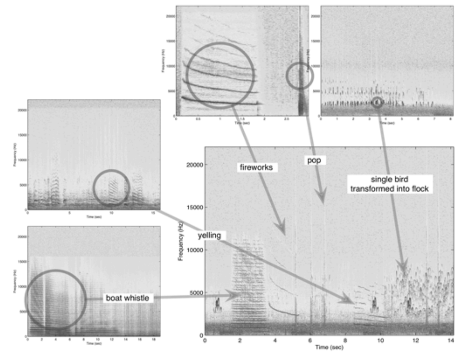
- visualization (audio, runtime stats, shreduling, etc.)
- insight into real-time, live programs
- different views of programs
  - syntax (code, objects)
  - concurrency (shreds)
  - time and timing (time, timing)
  - semantics (type, coming soon)
- different view of programming process
  - "Program monitoring as performance art" - Andrew Appel
- new way of thinking about real-time and live audio programming

50



1


baby
1
glass
2
pop!
3
birds, ocean
4
Italy country road
5
TAPESTREA
Analyze
Sinusoidal analysis
Wavelet-tree decomposition
Transient detection
Transform
Vast frequency-warp and time-stretch
(on sinusoidal tracks)
Wavelet-tree parametric reconstruction
(on stochastic background)
Standard spectral warping
(on transients)
Event density and periodicity
Synthesize
Sinusoidal resynthesis
Wavelet-tree synthesis
Group/granular modeling
2
1
recomposed sounds
can be controllably similar
to source sounds or
combinations of greatly
transformed components
4
5
3
4
1


fireworks
pop
single bird
transformed into flock
yelling
boat whistle
Frequency (Hz)
Time (sec)