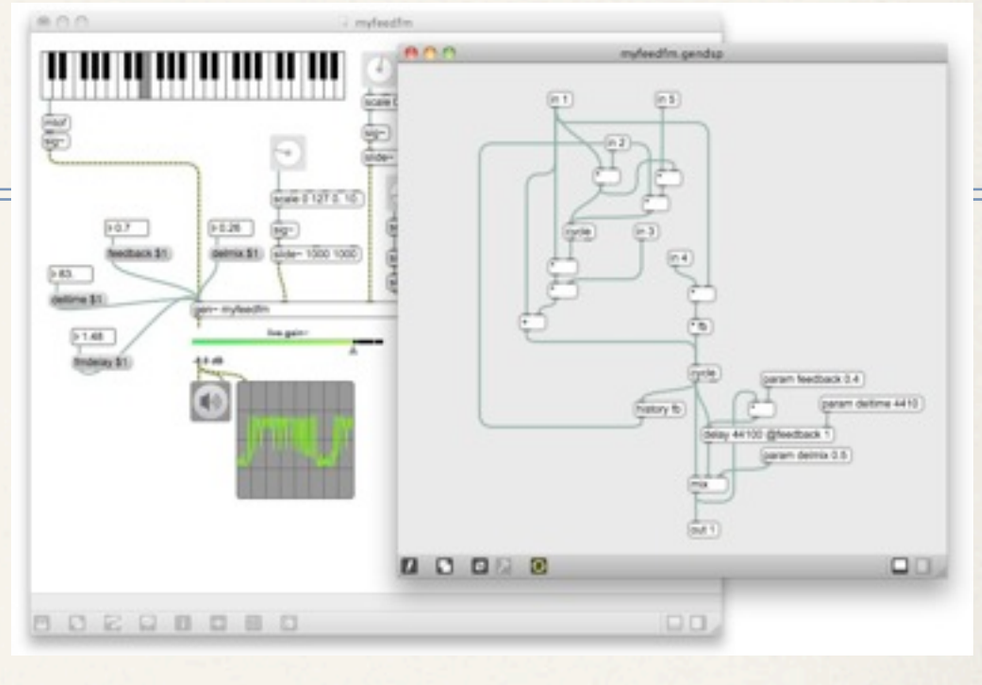
Super Collider 3 für Einsteiger

Dr. Chikashi Miyama

Jan. 2012

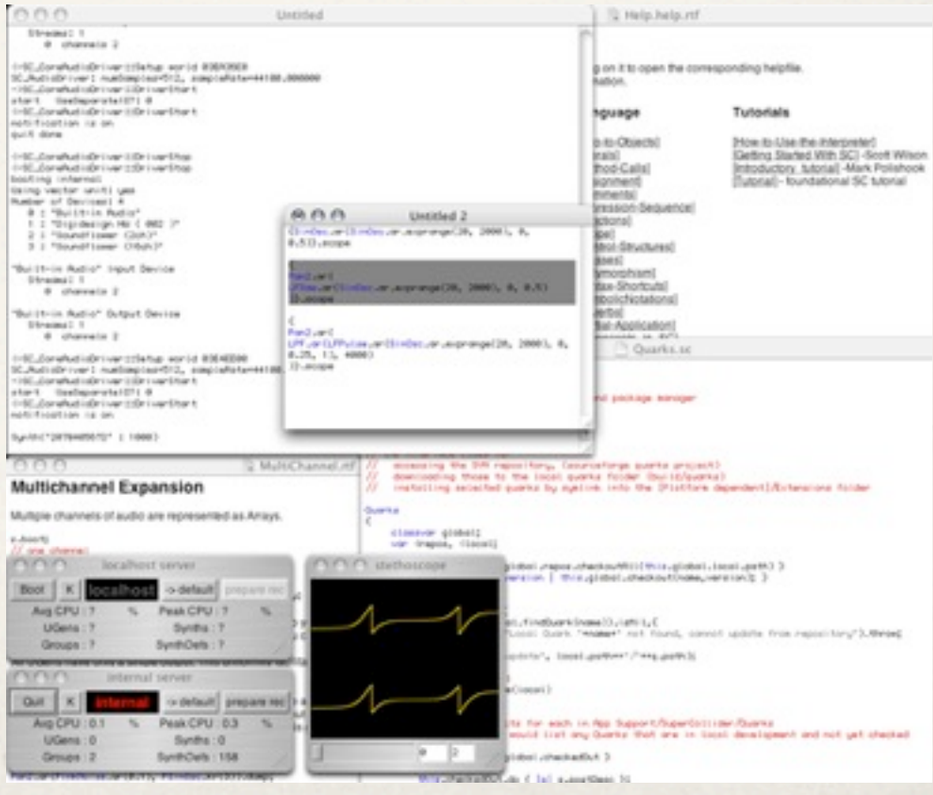
Super Collider 3



- SC3 (Abkürzung)
  - Programmiersprache (wie C, C++, Java, Javascript, Lisp usw.)

- Mit SC3 kann man...
  - eigene Klangeffekte entwickeln
  - verschiedene Klänge synthetisieren
  - Generative Musik
  - Live-Coding

Demo.2



Installation

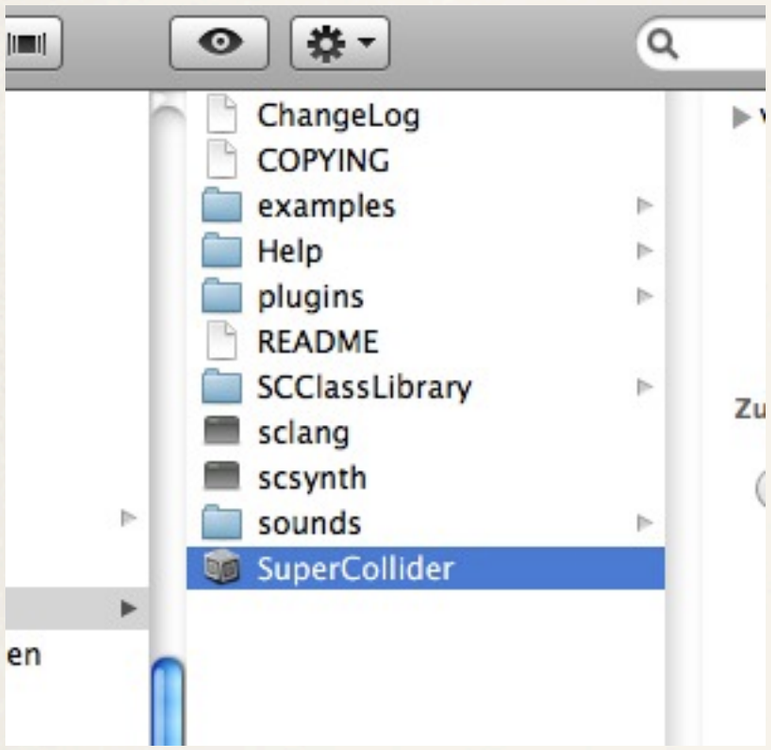
SuperCollider Website

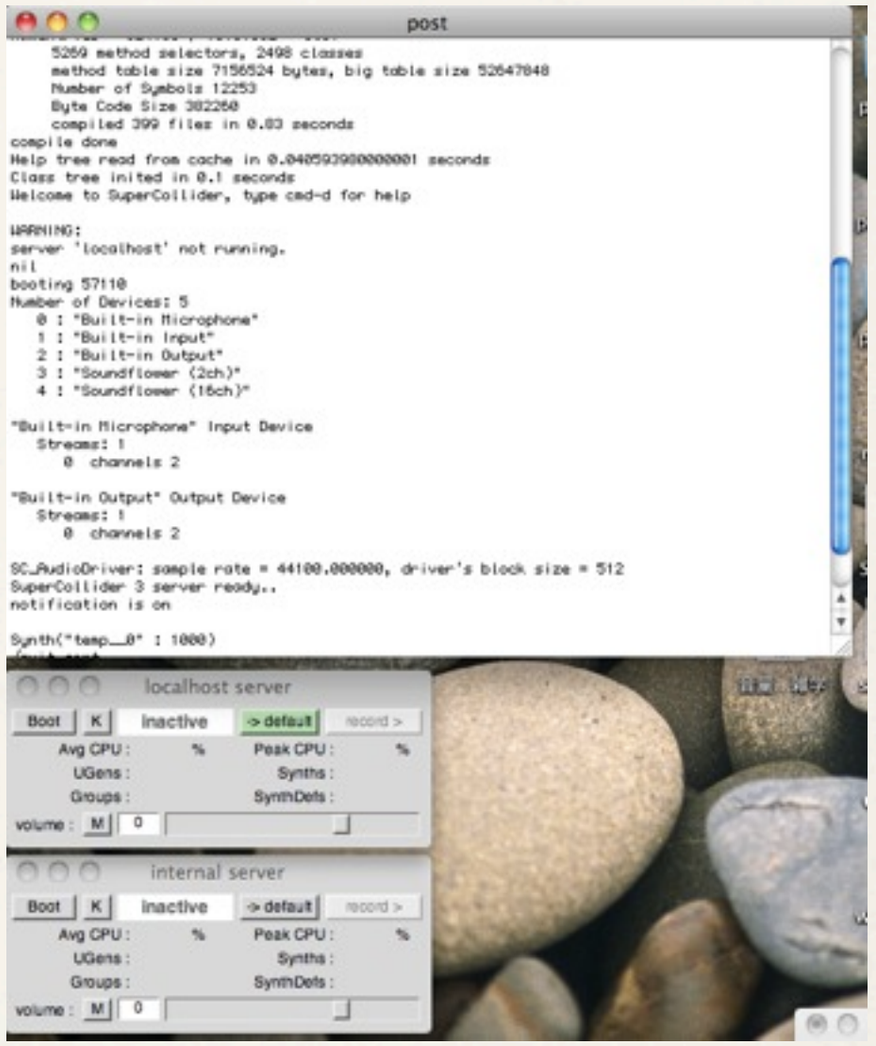
- http://supercollider.sourceforge.net/
  - SC3 für Mac, Windows und Linux
  - Beispiele
  - Videos
  - Tutorials
  - Wiki

Installation (Mac)

- Kopiere den SuperCollider Ordner in Programme (Applications) Ordner

Start SC3





- Im „Programme“ Ordner, das SuperCollider Ikon doppelklicken.

1 großes Fenster mit Text
2 kleines Fenster mit verschiedenen Knöpfen

SC3 besteht aus zwei Software

SCLang

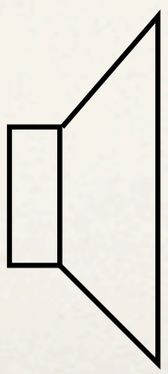
Programmieren /
Synthesizer oder
Kompositionsalgorithmus entwickeln

OSC messages

Andere Software

z.B.
Max/MSP, TouchOSC,
PureData, Quartz Composer, Ardour,
Chuck, Logic Pro, Processing

SC Server
(local host)



Lautsprecher

Projekt 1 : Synthesizer

1.a Sinuswelle

```
{
  SinOsc.ar(440);
}.play
```

1. Wähle File -> New aus
2. Drücke „Boot“ auf dem *localhost server* Panel
3. Schreib den Code in „Untitled“ Fenster ab
4. Wähle alle 3 Zeilen aus (oder Cmd + a)
5. Drücke ENTER (nicht Return)

↓

Fn + Return oder Ctrl + Return

6. Drücke Cmd + . (um den Klang zu stoppen)

1.a Tonhöhe / Frequenz

```
{
  SinOsc.ar(2200);
}.play
```


muss zwischen 40 - 22000 Hz. sein

1.b Lautstärke / Amplitude

Frequenz

Phase

```
{
  SinOsc.ar(440, 0.0, 0.2);
}.play
```

Amplitude

0.0 (Minimum) - 1.0 (Maximum)

1.c Klangfarbe

```
{
   Saw.ar(440);
}.play
```

Statt SinOsc.ar(440);

- Pulse.ar(440);
- Saw.ar(440);
- Blip.ar(440);
- LFTri.ar(440);
- Formant.ar(440);

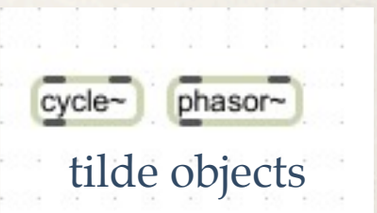
Tipp / Unit Generator

UGen... Unit Generator

Ugen funktioniert wie **„Tilde Object "** in Max/MSP. Ein Ugen hat eine eigene Funktion um einen Klang zu generieren, analysieren oder prozessieren. Ein Ugen kann z.B. ein Oscillator, ein Filter, ein Pitch Tracker, sein.

z.B. : SinOsc, Pulse, Saw, Formant, PitchShift, LPF, sind Ugen.

- Wie viel Ugens gibt es in SC3?
  - ca. 250 !!!
    - SuperCollider Help > Tour of Ugens.
    - Die Datei stellt die wichtigsten Ugens (nicht alle) vor.



tilde objects

Tipp / HELP!

- Helpfiles
  - Zu erst, wähle ein Wort aus.

```
{SinOsc.ar(440.0)}.play
```

  - Dann drück „Command + D“

SinOsc **interpolating sine wavetable oscillator**

SinOsc.ar(freq, phase, mul, add)

Sinusoidal oscillator; a sine tone.

Note: This is the same as Osc except that the table has already been fixed as a sine table of 8192 entries.

freq - frequency in Hertz
phase - phase offset or modulator in radians

```
{ SinOsc.ar(200, 0, 0.5) }.play;


// modulate freq
{ SinOsc.ar(XLine.kr(2000, 200), 0, 0.5) }.play;
```

2.a Harmonie

- Man kann alle Klänge benennen.

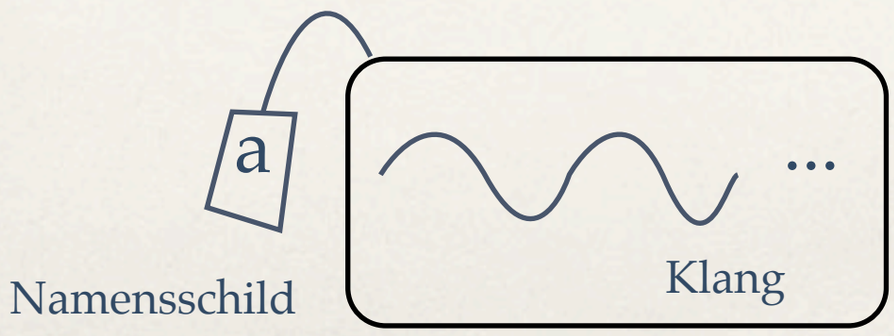
```
{
  a = SinOsc.ar(440);
  b = Pulse.ar(660);

  a
}.play
```

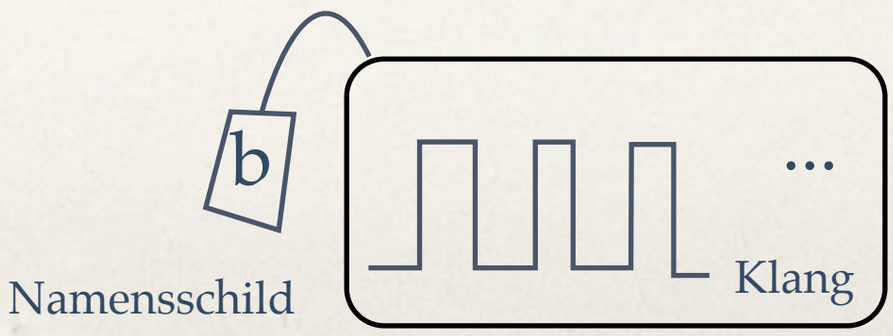
Dieser Klang heißt „a“

Dieser Klang heißt „b“

Wir hören „a“, wenn „a“ auf der letzten Zeile steht.

Probiere mal bitte mit „b“





2.b Harmonie

- Akkord: zwei oder drei Töne gleichzeitig spielen

```
{
  a = SinOsc.ar(440);
  b = Pulse.ar(660);

  a + b;
}.play
```

```
{
  a = SinOsc.ar(440);
  b = Pulse.ar(660);
  c = LFTri.ar(990);

  a + b + c;
}.play
```

addieren!

2.c Harmonie

- a = a + *etwas* ist OK auf SC3-Sprache

```
{
  a = SinOsc.ar(440);
  b = Pulse.ar(660);

  a + b;
}.play
```

```
{
  a = SinOsc.ar(440);
  a = a + Pulse.ar(660);
  a
}.play
```

ich benenne *a + Pulse.ar(660)* wieder *a*

Tipp / Kommentar

- Man kann immer im Programm Kommentare schreiben.
- Ein Kommentar muss // (doppelten Schrägstrich) anfangen.
- SC3 ignoriert Zeichen, nach dem doppelten Schrägstrich.

```
{
  SinOsc.ar(440);
  //Mine schöne Sinuswelle!
}.play
```

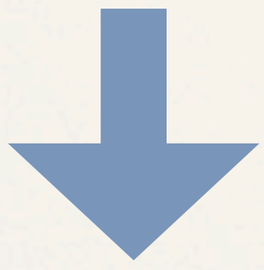
Kommentar :
SC3 ignoriert diese Zeile

Tipp / Syntax Colorize

- Cmd + ‘ (Apostroph)

  mit *Syntax Colorize*, koloriert SC3 das Programm automatisch!

```
//Crescendo
{
    SinOsc.ar(440) * XLine.ar(0.001, 1, 1);
}.play;
```



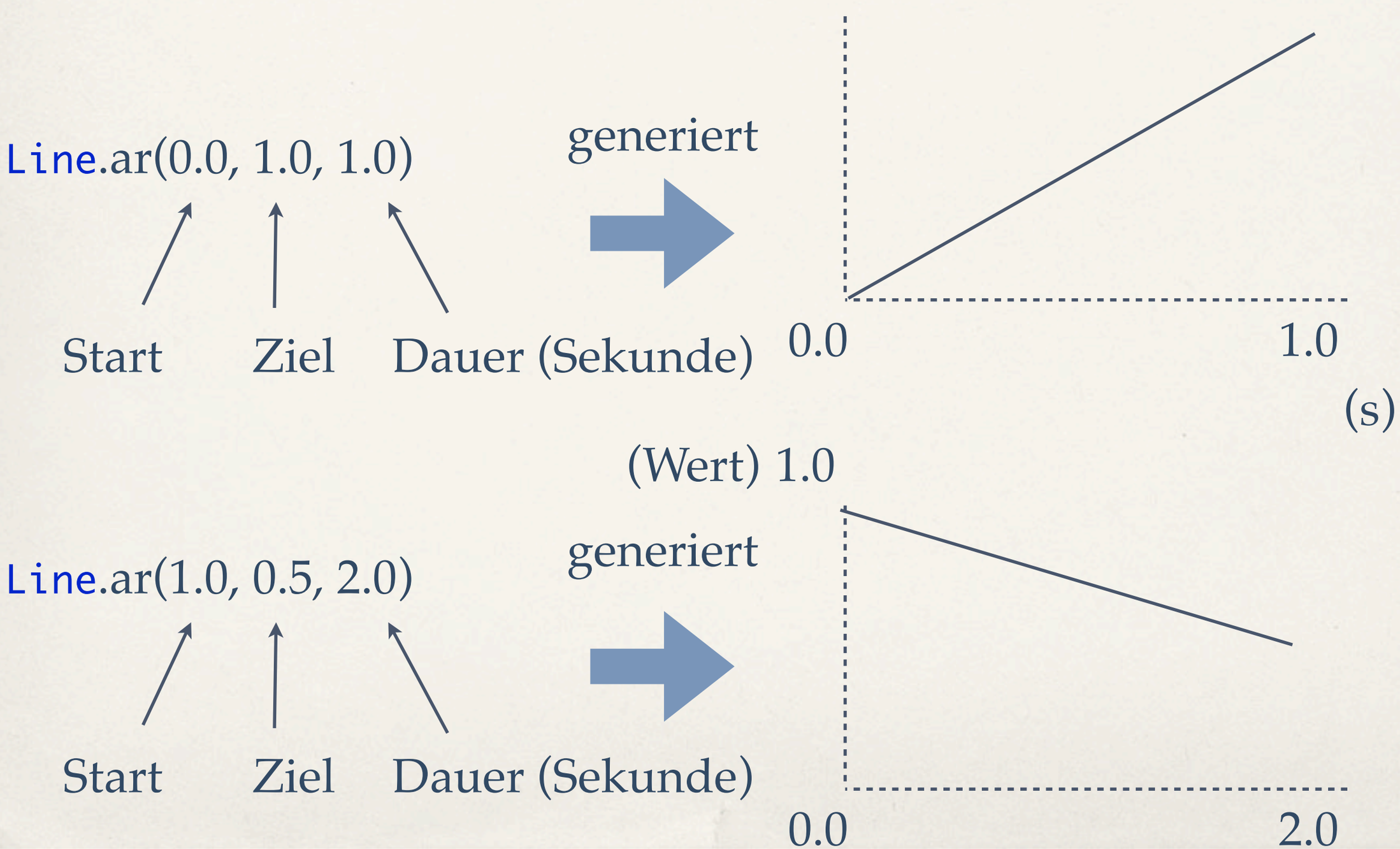
```
//Crescendo
{
    SinOsc.ar(440) * XLine.ar(0.001, 1, 1);
}.play;
```

Rot ... Comment

Bleu ... Ugens

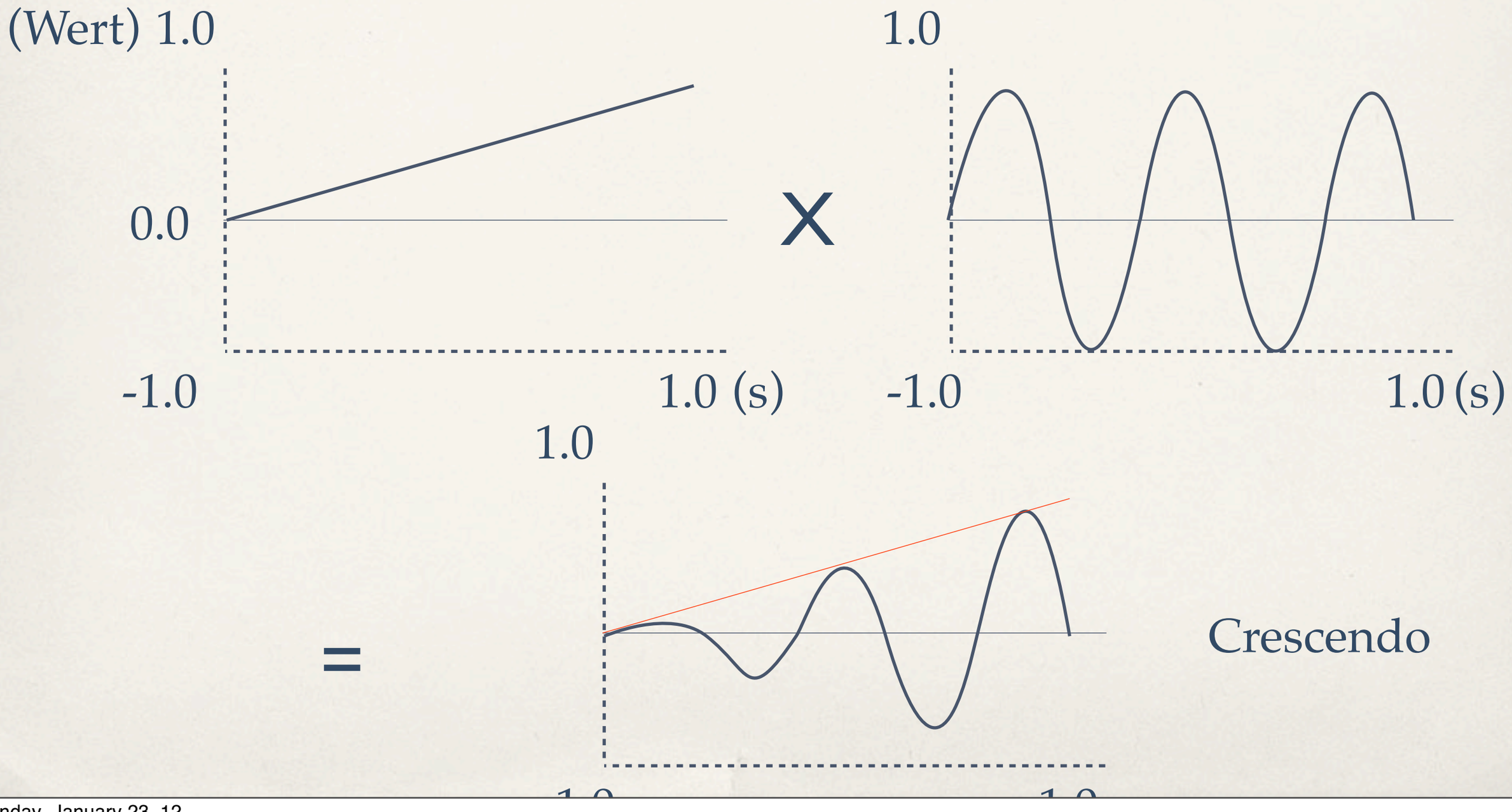
usw...

3. Crescendo / Decrescendo



3. Crescendo / Decrescendo

- Wenn man **Line.ar** und **SinOsc.ar** Multipliziert....



3.a Crescendo / Decrescendo

```
{
  SinOsc.ar(440) * Line.ar(0, 1, 1);
}.play
```

Crescendo

```
{
  SinOsc.ar(440) * Line.ar(1, 0, 1);
}.play
```

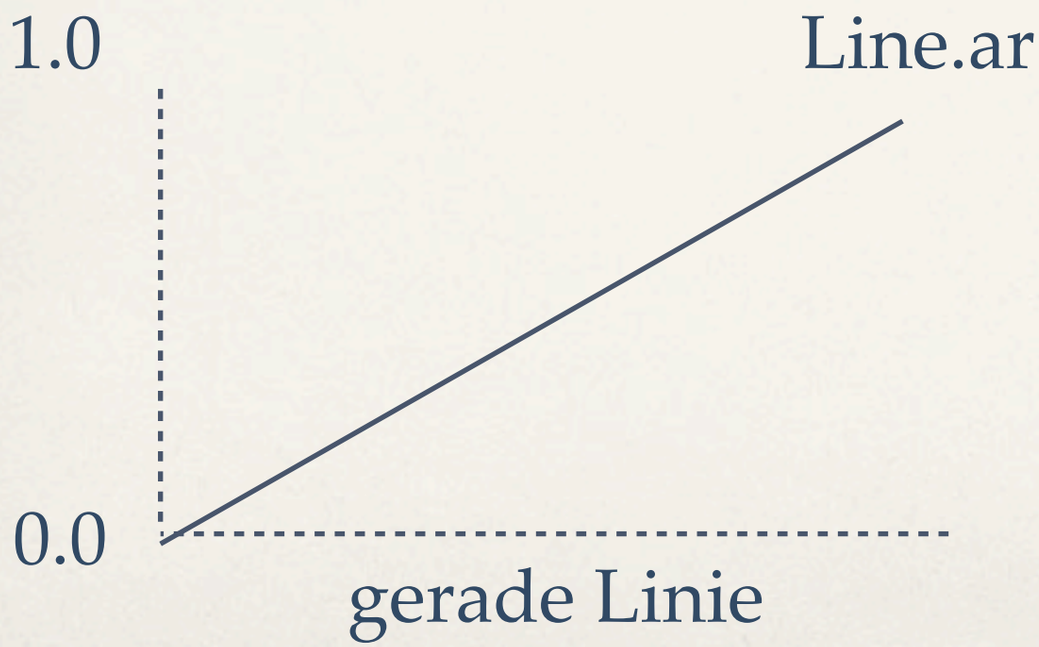
Decrescendo

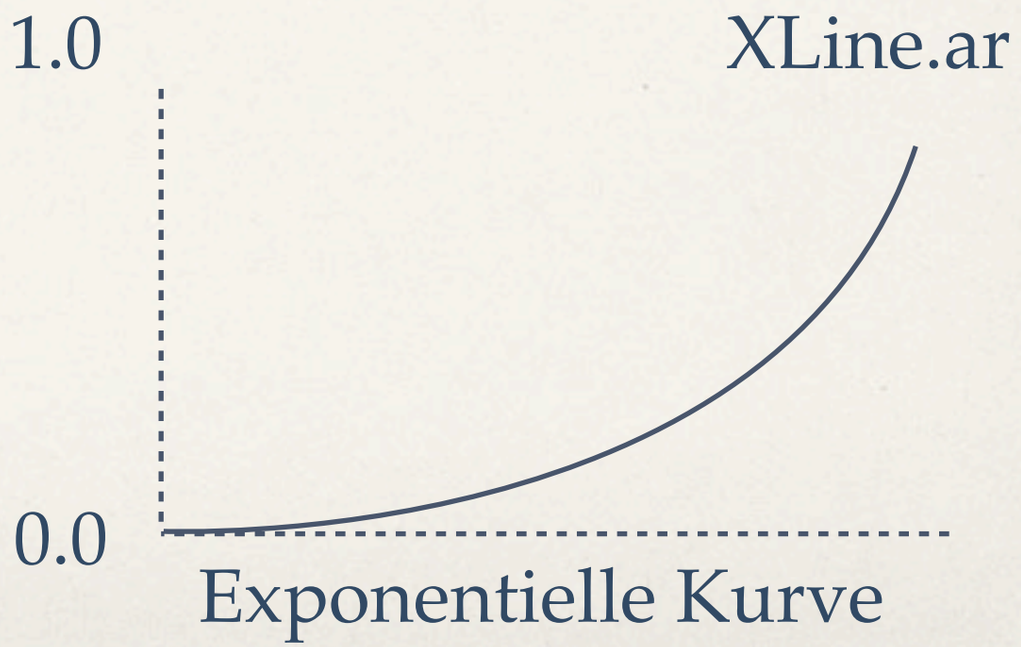
3.b Crescendo / Decrescendo

- Besseres Crescendo

darf nicht mit 0.0 anfangen

```
{
  SinOsc.ar(440) * XLine.ar(0.001, 1, 1);
}.play
```





4.a Glissando

```
{
  SinOsc.ar(Line.ar(220, 880, 1)) ;
}.play
```

Start (Hz.) Ziel (Hz.) Dauer (Sekunde)

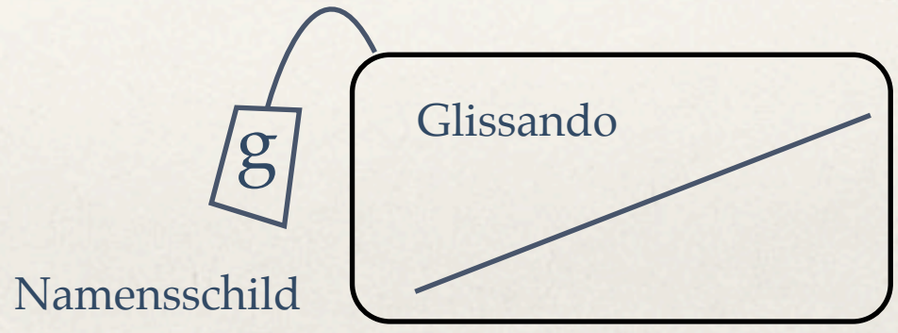
```
{
  SinOsc.ar(XLine.ar(220, 880, 1)) ;
}.play
```

4.b Glissando

- Man kann sowohl Klänge als auch Glissandi oder Crescendi benennen

```
{
  g = Line.ar(440, 880, 1); // mein Glissando heißt „g“
  SinOsc.ar(g) ;
}.play
```

```
{
  c = XLine.ar(0.001, 1, 1); // mein Crescendo heißt „c“
  SinOsc.ar(440, 0, c) ;
}.play
```



4.c Glissando

- Wenn man einen längeren Namen benutzen möchte...

```
{
  mein_glissando = Line.ar(440, 880, 1); // das geht nicht
  SinOsc.ar(mein_glissando) ;
}.play
```

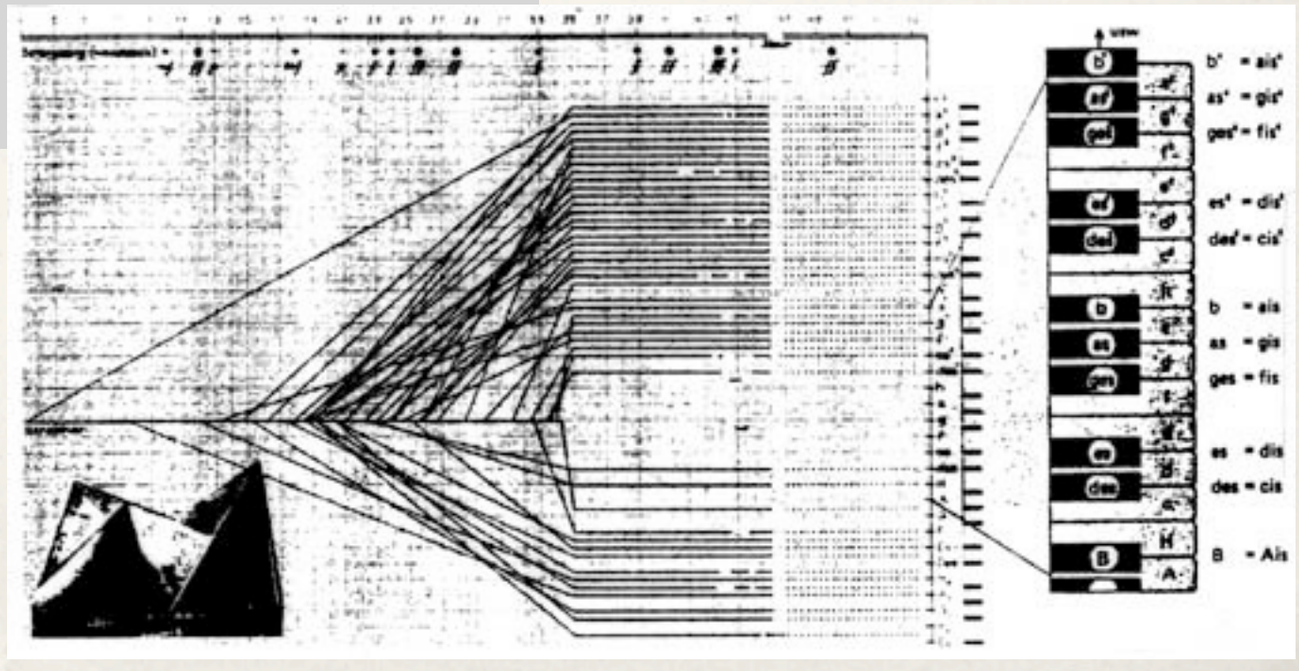
- Man muss zuerst den Namen mit **var** deklarieren.

```
{
  var mein_glissando; // Deklaration: Ich will „mein_glissando“ als ein Name benutzen.
  mein_glissando = Line.ar(440, 880, 1);
  SinOsc.ar(mein_glissando) ;
}.play
```

4.d Glissando (fakultativ)

- *Mini - Metastasis?*

```
{
  a = SinOsc.ar(220);
  b = SinOsc.ar(Line.ar(220, 330, 15));
  c = SinOsc.ar(Line.ar(220, 440, 15));
  d = SinOsc.ar(Line.ar(220, 550, 15));
  e = SinOsc.ar(Line.ar(220, 660, 15));
  (a + b + c + d + e)* 0.05;
}.play
```



5.a Kontrolle mit der Maus

- Unser erstes interaktives Programm

```
{
  x = MouseX.kr(100, 1000);
  SinOsc.ar(x);
}.play
```

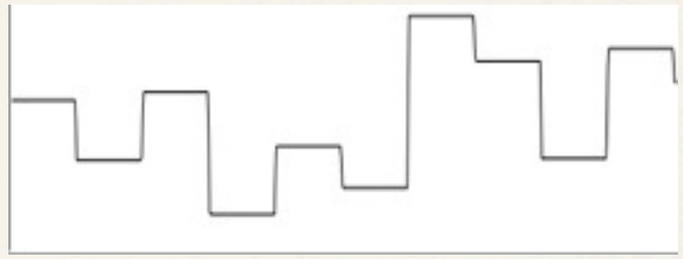
Bereich

5.b Kontrolle mit der Maus

- Randomisierung mit LFNoise

Frequenz Amplitude

```
{
  SinOsc.ar(440+LFNoise0.ar(5, 20));
}.play
```



```
{
  SinOsc.ar(440+LFNoise1.ar(5, 20));
}.play
```

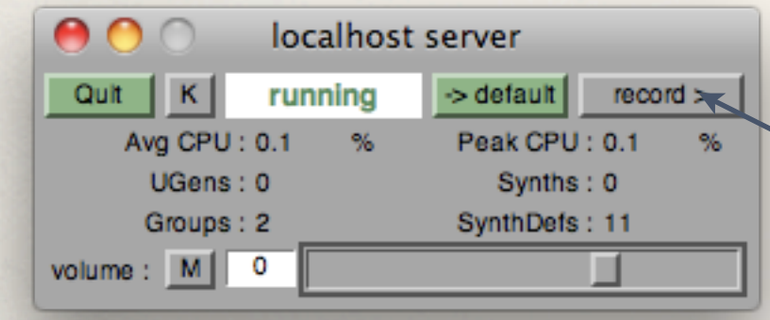
```
{
  SinOsc.ar(440+LFNoise2.ar(5, 20));
}.play
```

5.c Kontrolle mit der Maus

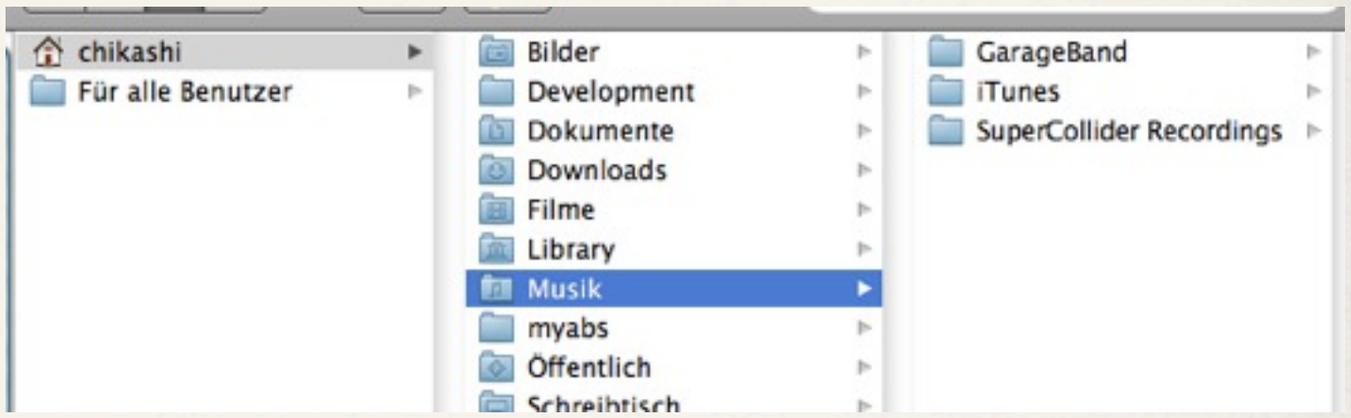
```
{
  x = MouseX.kr(100, 1000);
  y = MouseY.kr(0, 400);
  SinOsc.ar(x + LFNoise2.kr(30, y));
}.play
```

Die Amplitude von LFNoise2 wird per Maus kontrolliert.

Tipp : Aufnahme



Klick „record“



- Aufnahme
  - 1) Führe die *Synth* aus
  - 2) Klick „record“
  - 3) Klick „stop“
  - 4) man kann die Soundfile in diesem Ordner finden:
  - /Benutzer/<dein Name>/Musik/SuperCollider Recordings

6. Tonleiter

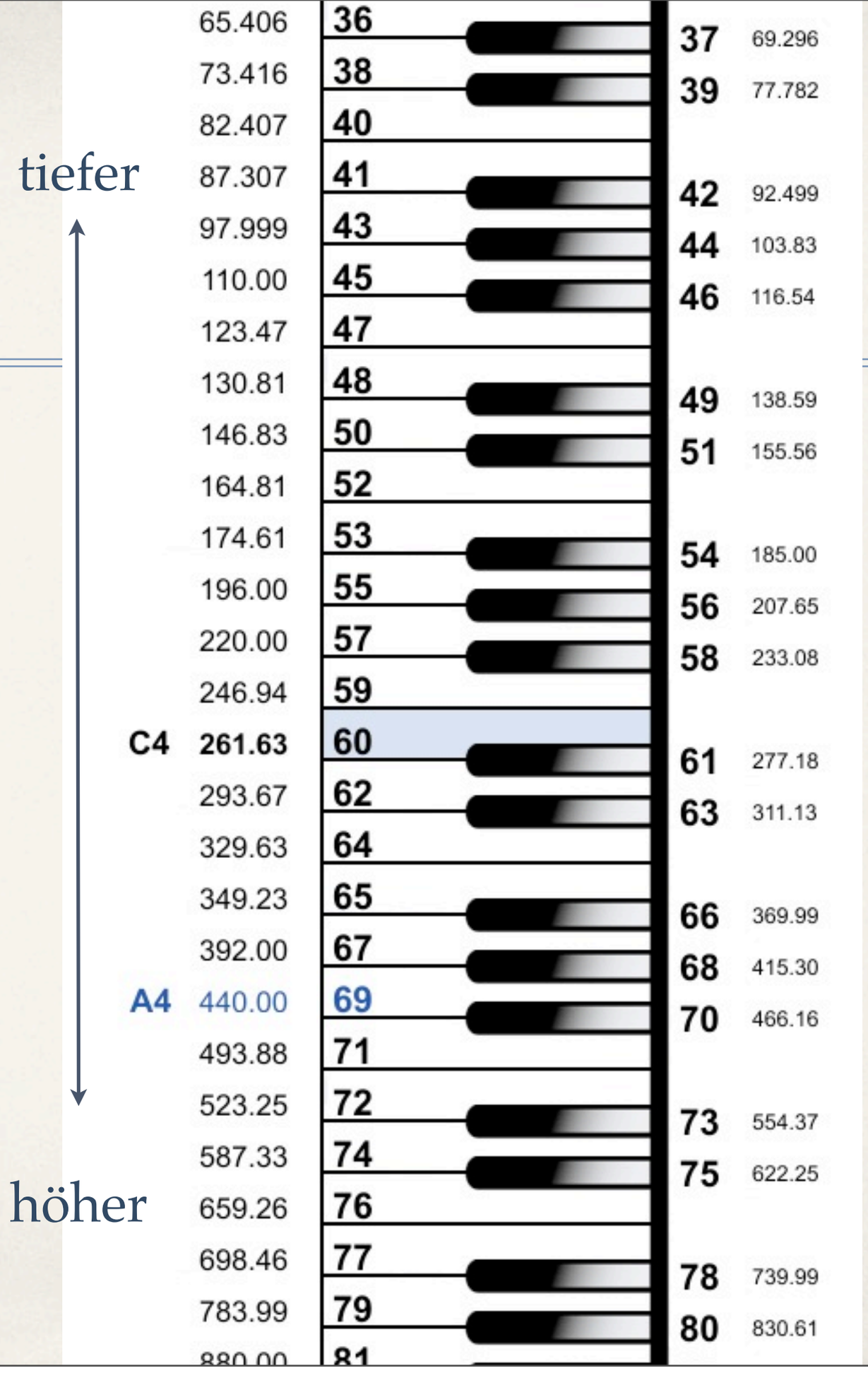
- MIDI note number

In SC 3 werden Tonhöhen oft auf MIDI note number geschrieben.

A4 = 440 Hz. = 69

C4 = 261 Hz. = 60

usw.



6.a Tonleiter

- Von MIDI Note number zu Frequenz Konversion

```
69.midicps ↵
```
... 440

```
440.cpsmidi ↵
```
... 69

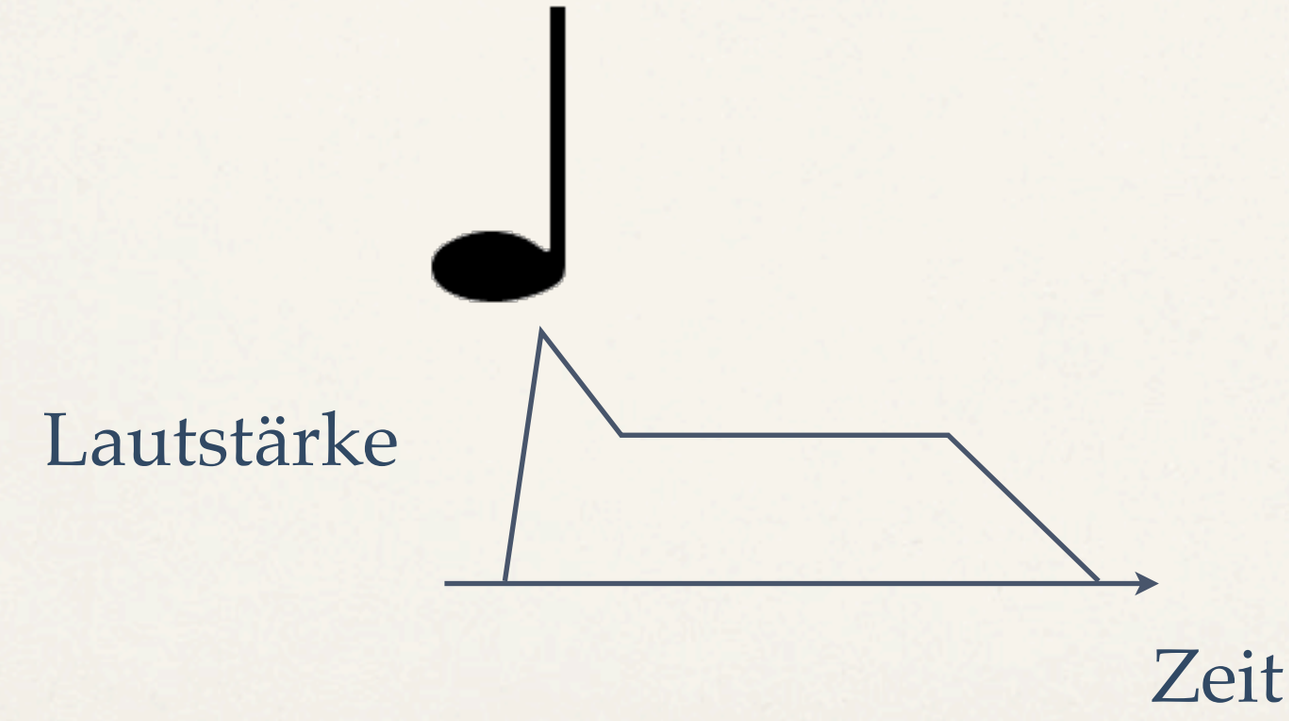
cps ... cycle per second

6.b Tonleiter

```
{
  c = SinOsc.ar(60.midicps); // c4
  e = SinOsc.ar(64.midicps); // e4
  g = SinOsc.ar(67.midicps); // g4
  (c + e + g) * 0.05; // C Dur Akkord
}.play
```

7. Hüllkurve (Envelope)

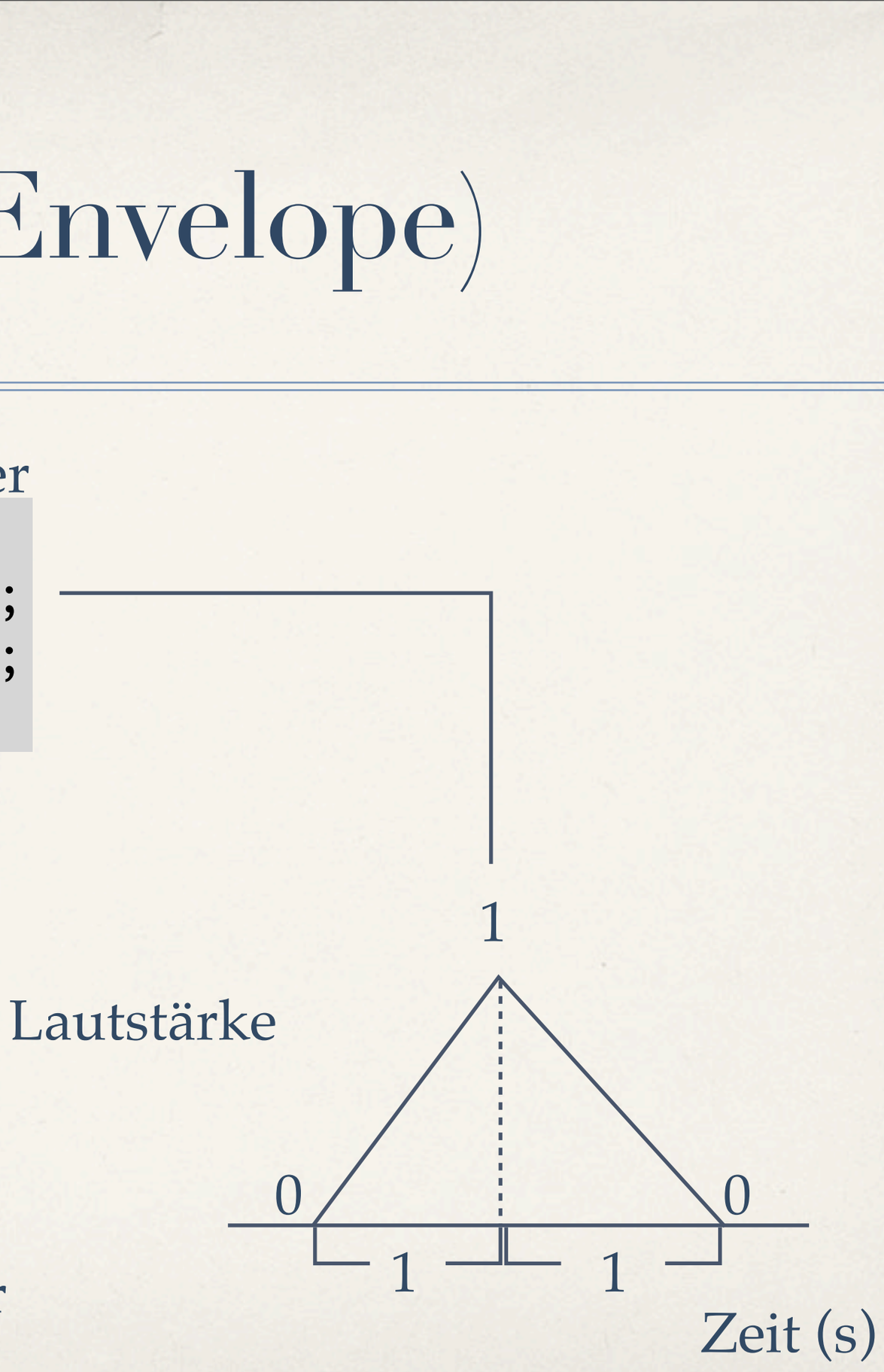
Hüllkurve ist crescendo und descrescendo in einer musikalischen Note



7.a Hüllkurve (Envelope)

Lautstärke Dauer

```
{                    |       |
  e = Env.new([0,1,0],[1,1]);
  EnvGen.ar(e) * Saw.ar(880);
}.play
```

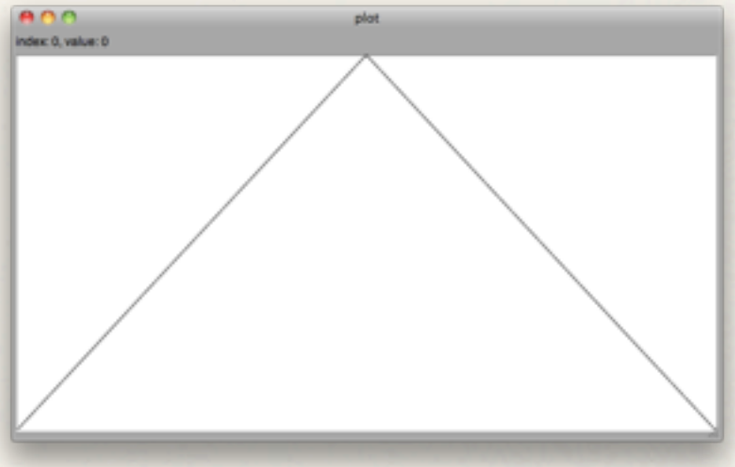


Env ... Entwurf der Hüllkurve
EnvGen ... Hüllkurvegenerator

7.b Hüllkurve (Envelope)

- Man kann die Hüllkurve schauen...

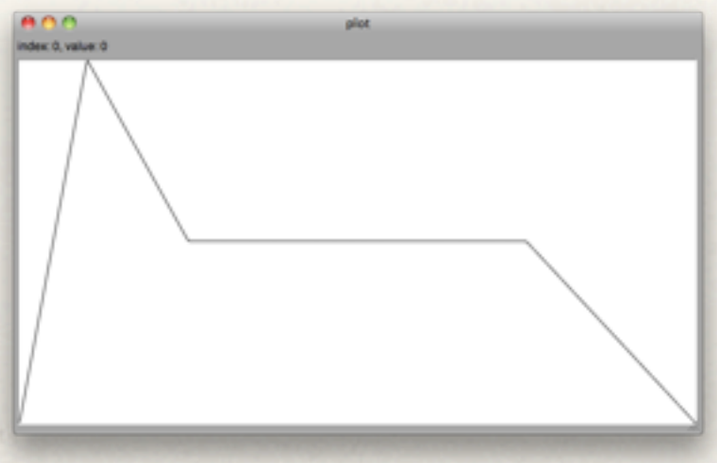
```
Env.new([0,1,0],[1,1]).plot
```



- ...und hören

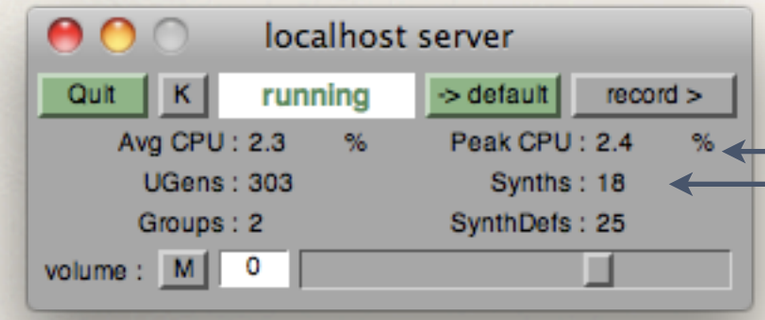
```
Env.new([0,1,0],[1,1]).test
```

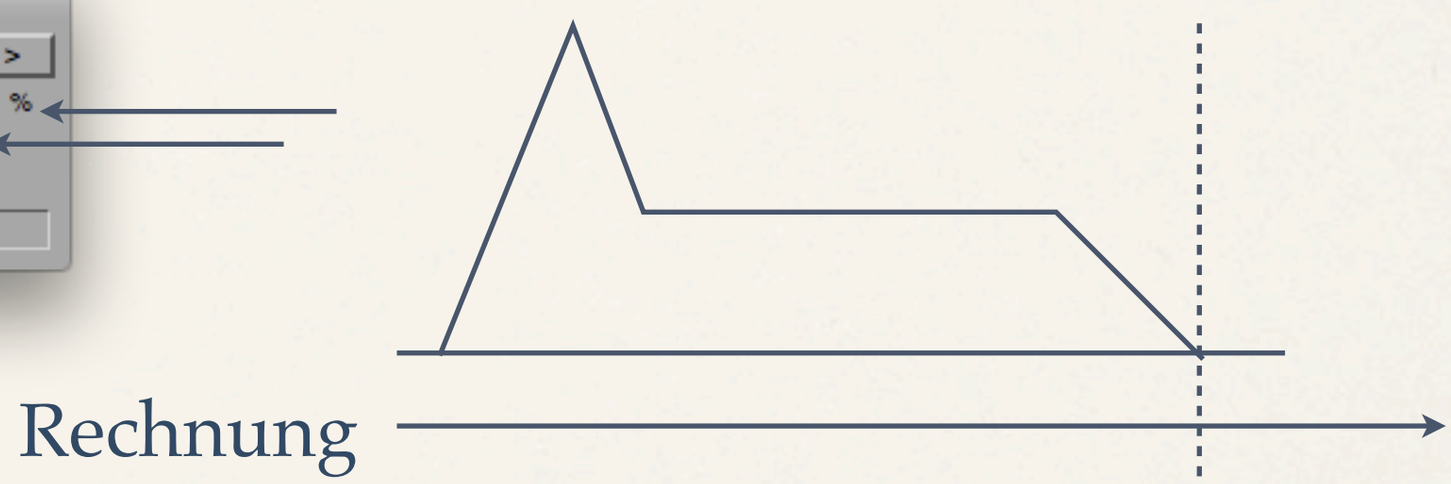
```
Env.new([0,1,0.5,0.5,0],[0.2,0.3,1,0.5]).plot
```



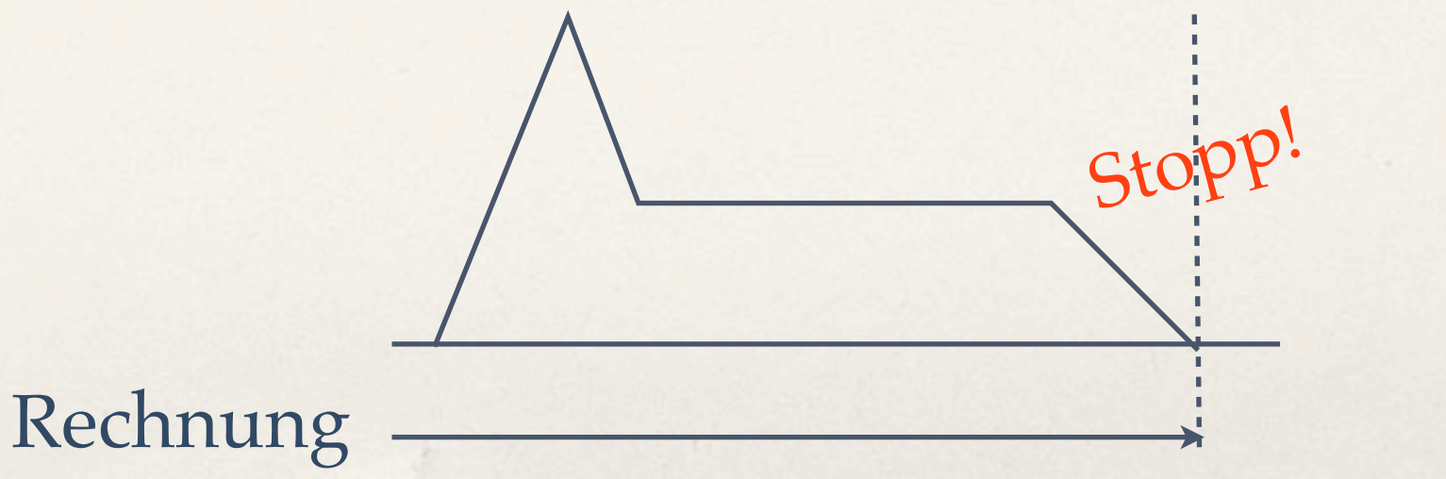
7.c Hüllkurve (Envelope)

- die CPU rechnet noch für die beendet Töne





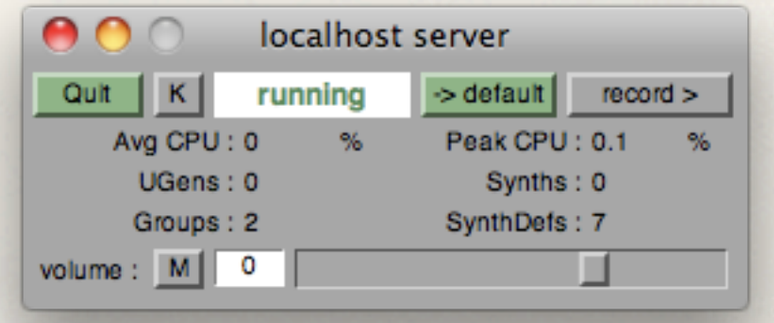
- die Rechnung soll automatisch beenden, wenn der Ton endet.



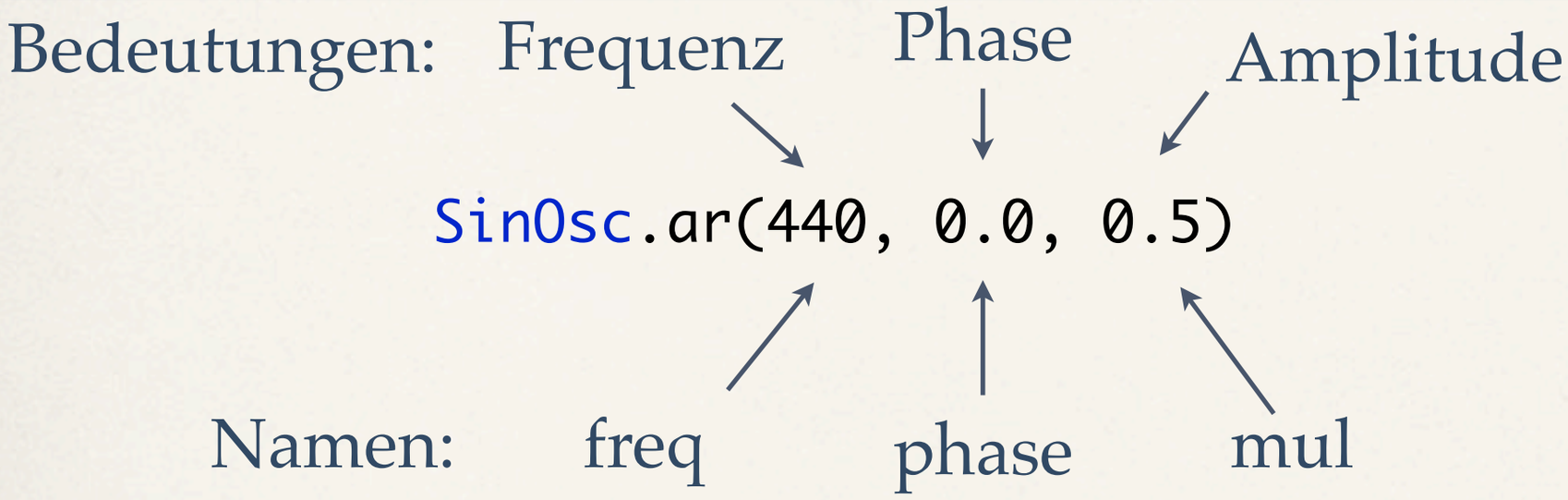
7.c Hüllkurve (Envelope)

```
{
  e = Env.new([0,1,0],[1,1]);
  EnvGen.ar(e, doneAction:2) * Saw.ar(880);
}.play
```

Die Syntheseberechnung wird automatisch beendet, wenn der Ton endet.



Tipp : Argumente



Frequenz : 440
Phase : 0.0
Amplitude: 0.5

```
SinOsc.ar(440, mul: 0.5)
```

Man kann auch mit einem Namen und einem Doppelpunkt Parameter kontrollieren.

```
SinOsc.ar(mul: 0.5, freq: 440)
```

Wenn man mit einem Namen allen Parameter kontrolliert , spielt die Reihenfolge des Parameters keine Rolle.

Tipp : Argumente

```
EnvGen.ar(envelope, gate, levelScale, levelBias,
timeScale, doneAction)
```

z.B EnvGen hat 6 Argumente.
Wenn man nur zwei Parameter bzw. *envelope* und *doneAction* eingeben möchte, kann man

```
EnvGen.ar(e, doneAction:2)
```

schreiben statt

```
EnvGen.ar(e, 1, 1, 0.0, 1, 2)
```

8.a Filter

- LPF.ar ... Low Pass Filter
- HPF.ar ... High Pass Filter
- BPF.ar ... Band Pass Filter

```
{
  a = Pulse.ar(100);
  LPF.ar(a,550);
  //HPF.ar(a,1550);
  //BPF.ar(a,400, 15);
}.play
```

```
{
  a = WhiteNoise.ar;
  LPF.ar(a,550);
  //HPF.ar(a,1550);
  //BPF.ar(a,400, 15);
}.play
```

8.b Filter

Beispiele mit Filtern

```
{
  a = Pulse.ar(100);
  BPF.ar(a,MouseX.kr(100, 2000), 1);
}.play
```

Die Cutoff Frequenz wird per Maus kontrolliert.

```
{
  a = Saw.ar(200);
  BPF.ar(a, SinOsc.kr(1) * 150 + 200  , 0.5);
}.play
```

Die Cutoff Frequenz wird durch SinOsc kontrolliert.

8.c Filter

Techno Lead?

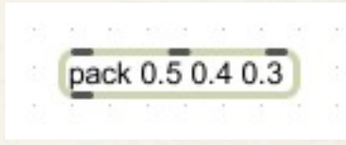
```
{
  a = Pulse.ar(100);
  LPF.ar(a, XLine.ar(2000, 8000, 0.9));
}.play
```

mit Hüllkurve

```
{
  a = Pulse.ar(100);
  l = LPF.ar(a, XLine.ar(2000, 8000, 0.3));
  e = Env.new([0.0, 1.0, 0.0],[0.03, 0.4]);
  EnvGen.ar(e, doneAction: 2) * l
}.play
```

Tipp : Klammern

- Drei Arten von Klammern
  - () Runde Klammern
    - SinOsc.ar(440) ... Argumente von Ugen
  - [] Eckige Klammern



    - [0.5, 0.4, 0.3] ... „Array“, funktioniert wie „pack“ in Max/MSP
      Man kann mehrere Zahlen zusammenstellen.
  - {} Geschweifte Klammern
    ... „Function“ eine Sequenz von Instruktionen

Tipp / .kr oder .ar

Wenn man ein Ugen benutzt, man
<Name eines Ugens>.**ar** oder
<Name eines Ugens>.**kr**
schreiben muss.

z.B. SinOsc.ar / SinOsc.kr

Tipp / .kr oder .ar

- .ar bedeutet **audio rate**
  - Mit einem **.ar** Ugen kann man einen Klang generieren oder andere Ugen kontrollieren. Aber **.ar** braucht mehr CPU-Kraft.
- .kr bedeutet **control rate**
  - Mit einem **.kr** Ugen kann man nur andere Ugens kontrollieren. **.kr** benötigt weniger CPU-Kraft.

Tipp / .kr oder .ar

- z.B.

```
{
   l = Line.ar(0, 440, 5);
   k = SinOsc.ar(880+l);
}.play
```

```
{
   l = Line.kr(0, 440, 5);
   k = SinOsc.ar(880+l);
}.play
```

... das geht auch.

... weniger CPU Verbruch

```
{
   l = Line.kr(0, 440, 5);
   k = SinOsc.kr(880+l);
}.play
```

... das geht nicht

9.a SynthDef

- Ich möchte mit „Techno Lead“ eine Melodie spielen.

- Mit **SynthDef** kann man eine eigene *Synth*(Instrument) definieren.

Der Name meiner *Synth*

```
SynthDef("MyLead",{
   a = Pulse.ar(100);
   l = LPF.ar(a, XLine.ar(2000, 8000, 0.3));
   e = Env.new([0.0, 1.0, 0.0],[0.03, 0.4]);
   x = EnvGen.ar(e, doneAction: 2) * l;
   Out.ar(0, x);
}).load(s)
```

0 bedeutet Linkslautsprecher. 1 ist Rechtslautsprecher

9.b SynthDef

danach kann man mit dem Name die Synth verwenden

```
Synth("MyLead")
```

Die Definition ist **permanent**. Nachdem Neustart des Computers kann man auch “MyLead” verwenden.

Wie kann man musikalische Parameter
(z.B. Frequenz oder Amplitude) der Synth verändern?

9.c SynthDef

```
SynthDef("MyLead",{
   arg freq;
   a = Pulse.ar(freq);
   l = LPF.ar(a, XLine.ar(2000, 8000, 0.3));
   e = Env.new([0.0, 1.0, 0.0],[0.03, 0.4]);
   x = EnvGen.ar(e, doneAction: 2) * l;
   Out.ar(0, x);
}).load(s)
```

Deklaration von Argument „freq“

mit Parameter

```
Synth("MyLead", ["freq", 340]);
```
← 340 Hz

```
Synth("MyLead", ["freq", 880]);
```
← 880 Hz

```
Synth("MyLead", ["freq", 69.midicps]);
```
← 440 Hz

9.d SynthDef

```
SynthDef("MyLead",{
   arg freq, amp;
   a = Pulse.ar(freq) * amp;
   l = LPF.ar(a, XLine.ar(2000, 8000, 0.3));
   e = Env.new([0.0, 1.0, 0.0],[0.03, 0.4]);
   x = EnvGen.ar(e, doneAction: 2) * l;
   Out.ar(0, x);
}).load(s)
```

Zweites Argument „amp“

Einstellung der Lautstärke

```
Synth("MyLead",["freq", 220, "amp", 0.1])
```

Frequenz = 220

Lautstärke = 0.1

9.e SynthDef

mit “default” Argumente

```
SynthDef("MyLead",{
   arg freq = 440, amp = 0.5;
   a = Pulse.ar(freq) * amp;
   l = LPF.ar(a, XLine.ar(2000, 8000, 0.3));
   e = Env.new([0.0, 1.0, 0.0],[0.03, 0.4]);
   x = EnvGen.ar(e, doneAction: 2) * l;
   Out.ar(0, x);
}).load(s)
```

```
Synth("MyLead",["freq", 220, "amp", 0.1])
```

Frequenz = 220
Lautstärke = 0.1

```
Synth("MyLead",["amp", 0.1])
```

Frequenz = 440
Lautstärke = 0.1

```
Synth("MyLead")
```

Frequenz = 440
Lautstärke = 0.5

10.a Der leichteste Sequencer

- Scheduling

```
Routine{
   Synth("MyLead",["freq",60.midicps]); // spiele C4
   1.0.wait; //warte 1 Sekunde
   Synth("MyLead",["freq",64.midicps]); // spiele E4
   1.0.wait; //warte 1 Sekunde
   Synth("MyLead",["freq",67.midicps]); // spiele G4
}.play;
```

10.b Der leichteste Sequencer

```
Routine{
   loop({ //unendlicher Loop
      Synth("MyLead",["freq",60.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",72.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",67.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",68.midicps]);
      0.2.wait;
   });
}.play
```

10.c Der leichteste Sequencer

Rhythmisch?

```
Routine{
   loop({ //unendlicher Loop
      Synth("MyLead",["freq",60.midicps]);
      0.3.wait;
      Synth("MyLead",["freq",72.midicps]);
      0.3.wait;
      Synth("MyLead",["freq",67.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",68.midicps]);
      0.2.wait;
   });
}.play;
```

10.d Der leichteste Sequencer

`12.rand` ⟵ generiert eine Zahl zwischen 0 und 11

```
Routine{
   loop({ //unendlicher Loop mit Randomisierung
      Synth("MyLead",["freq",(12.rand+60).midicps]);
      0.3.wait;
      Synth("MyLead",["freq",72.midicps]);
      0.3.wait;
      Synth("MyLead",["freq",67.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",68.midicps]);
      0.2.wait;
   });
}.play
```

10.e Der leichteste Sequencer

Beschränkte Zufälligkeit

`[60, 62, 63].choose` ⟵ generiert 60, 62 oder 63

```
Routine{
   loop({ //unendlicher Loop mit Randomisierung
      Synth("MyLead",["freq",[60, 62, 63].choose.midicps]);
      0.3.wait;
      Synth("MyLead",["freq",72.midicps]);
      0.3.wait;
      Synth("MyLead",["freq",67.midicps]);
      0.2.wait;
      Synth("MyLead",["freq",68.midicps]);
      0.2.wait;
   });
}.play
```

Tipp / Array

Array funktioniert wie „pack“ Objekt in Max/MSP.
Man kann viele Informationen zusammenstellen.

```
[2, 304, 15]; // 3 Zahlen
["Musik", "Hochschule", "Koeln"]; // 3 Wörter
[2, "Hochuschule", 15]; // gemischt ... auch OK
```

ein Array akzeptiert verschiedene „messages“

```
[60, 62, 64, 67, 71].at(2); //nimm zweiten Inhalt
[60, 62, 64, 67, 71].choose; //wählt ein Zahl aus.
[60, 62, 64, 67, 71].scramble; //randomisierte Reihenfolge.
[60, 62, 64, 67, 71] + 12; //jede Zahl + 12
[60, 62, 64, 67, 71] + [5,3,1,-2,-6];
[60, 62, 64, 67, 71].mirror;
[60, 62, 64, 67, 71].rotate(2);
[60, 62, 64, 67, 71].stutter(2);
```

Tipp / Array -
Musikalische Anwendung

```
Routine({
   [60, 61, 62, 67].do{
   arg pitch;
   Synth("MyLead",["freq", pitch.midicps]);
   0.1.wait
   }
}).play;
```

```
Routine({
   [60, 61, 62, 67].scramble.do{
   arg pitch;
   Synth("MyLead",["freq", pitch.midicps]);
   0.1.wait
   }
}).play;
```

```
Routine({
   [60, 61, 62, 67].mirror.do{
   arg pitch;
   Synth("MyLead",["freq", pitch.midicps]);
   0.1.wait
   }
```

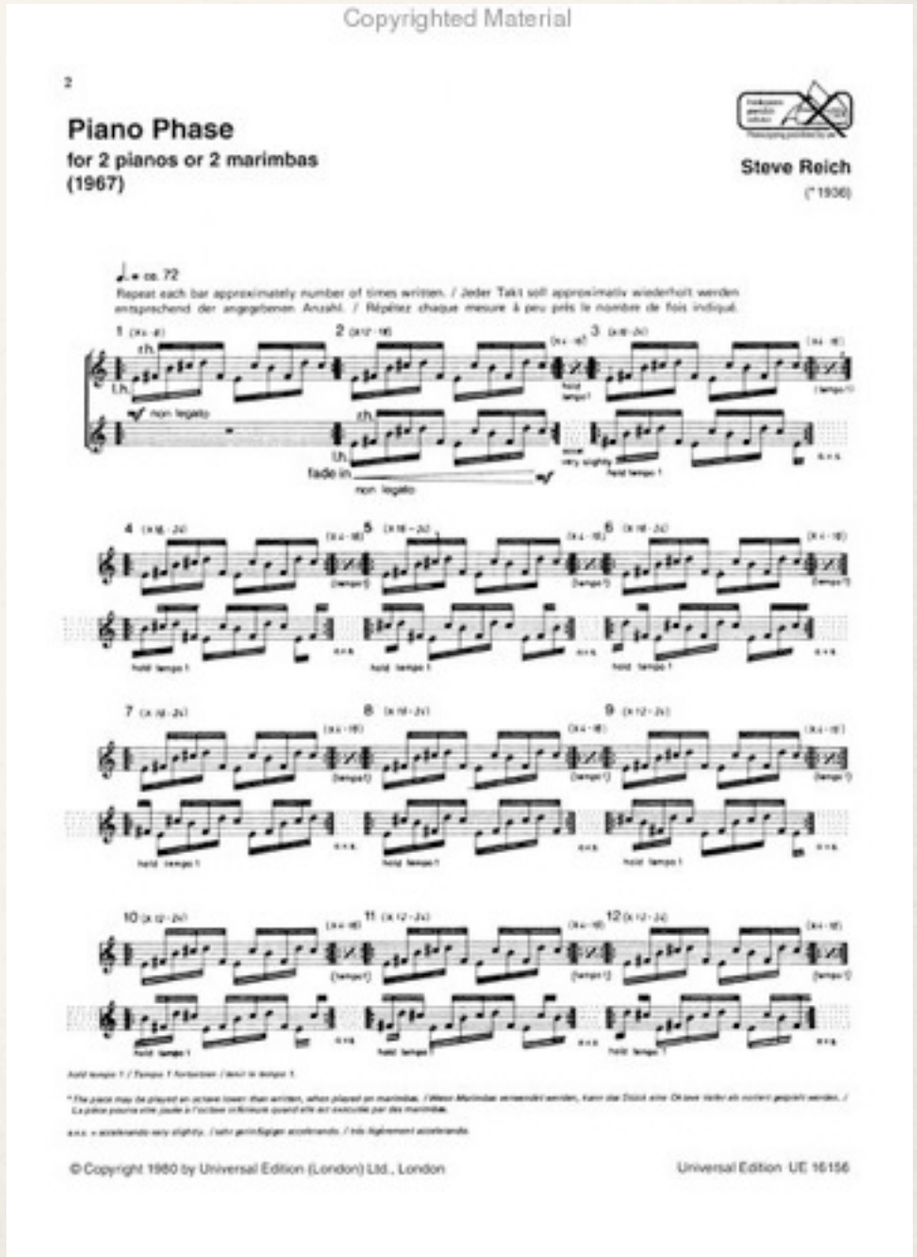
Tipp / Array -
Musikalische Anwendung

```
Routine({
   [60, 61, 62, 67].scramble.do{
   arg pitch;
   Synth("MyLead",["freq", pitch.midicps]);
   0.1.wait
   }
}).play;
```

```
Routine({
   ([60, 61, 62, 67]+7).do{
   arg pitch;
   Synth("MyLead",["freq", pitch.midicps]);
   0.1.wait
   }
}).play;
```

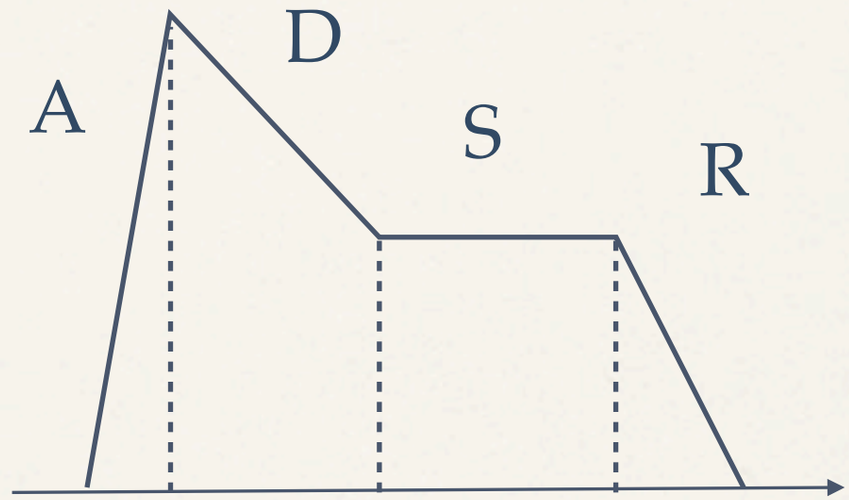
Tipp / Array -
Musikalische Anwendung

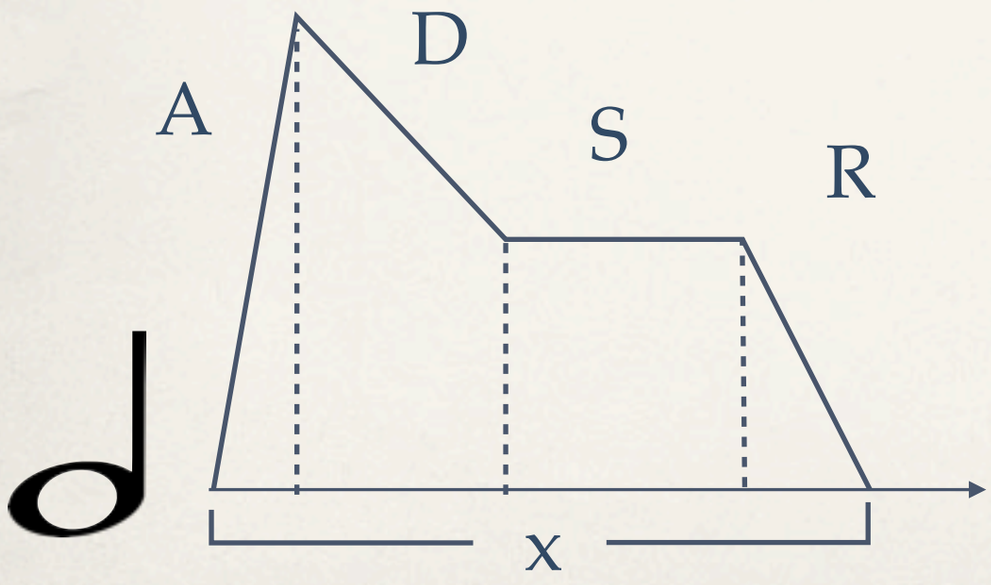
```
// Steve Reichs Stück = 16 Zeilen in SC3!
Routine({
   a = [64,66,71,73,74,66,64,73,71,66,74,73];
   x = 0;
   loop({
      b = a.rotate(x);
      4.do{
         12.do{
            arg idx;
            Synth("MyLead",["freq", a.at(idx).midicps]);
            Synth("MyLead",["freq", b.at(idx).midicps]);
            0.13.wait
         };
      };
      x = x + 1;
   });
}).play;
```

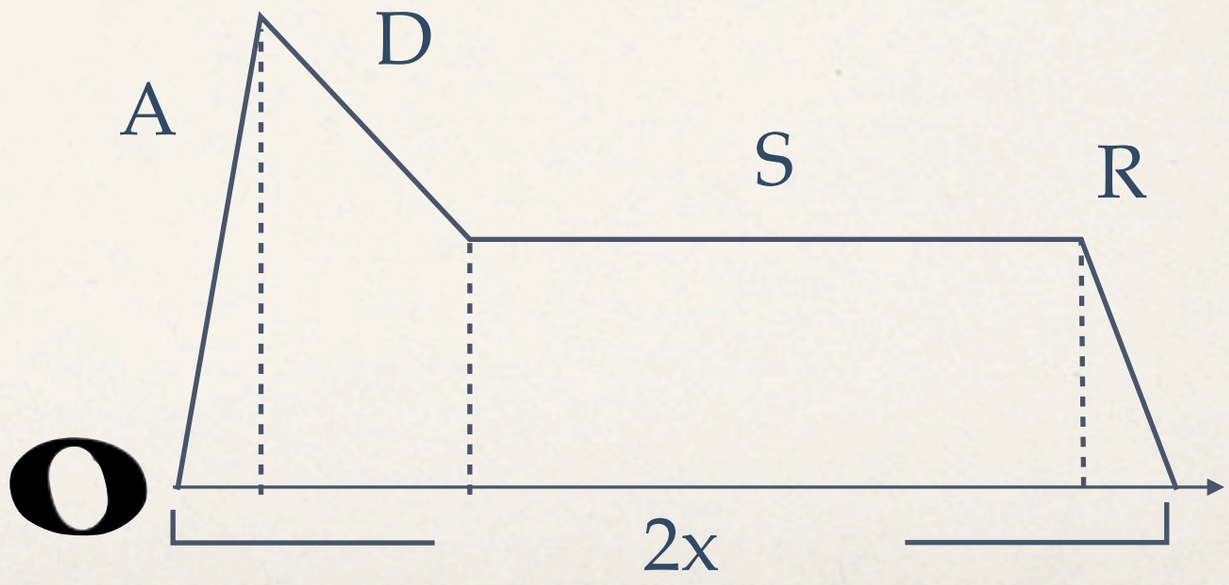


11. Hüllkurve mit unbestimmter Dauer

- A ... Attack
- D... Decay
- S ... Sustain
- R ... Release







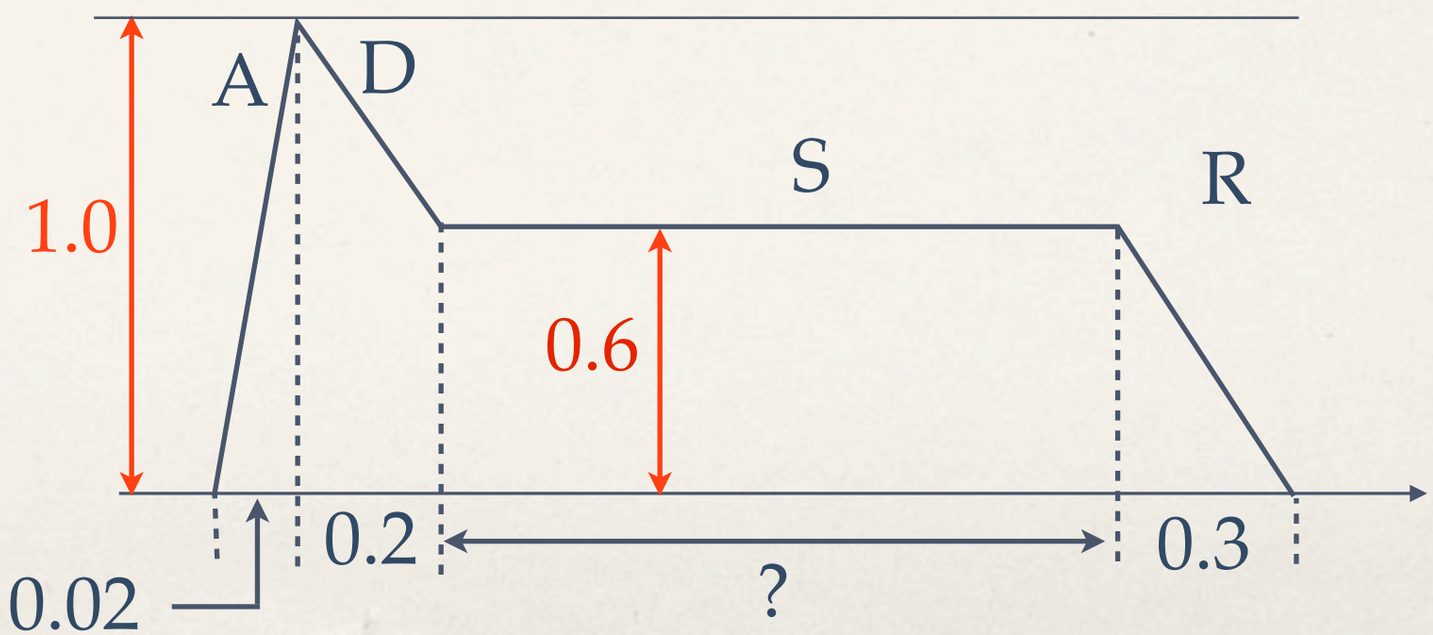
11. Hüllkurve mit unbestimmter Dauer

Hüllkurve mit bestimmter Dauer

```
Env.new([0.0, 1.0, 0.0],[0.5, 0.5]);
```

Hüllkurve mit unbestimmter Dauer

```
Env.adsr(0.02, 0.2, 0.6, 0.3);
```

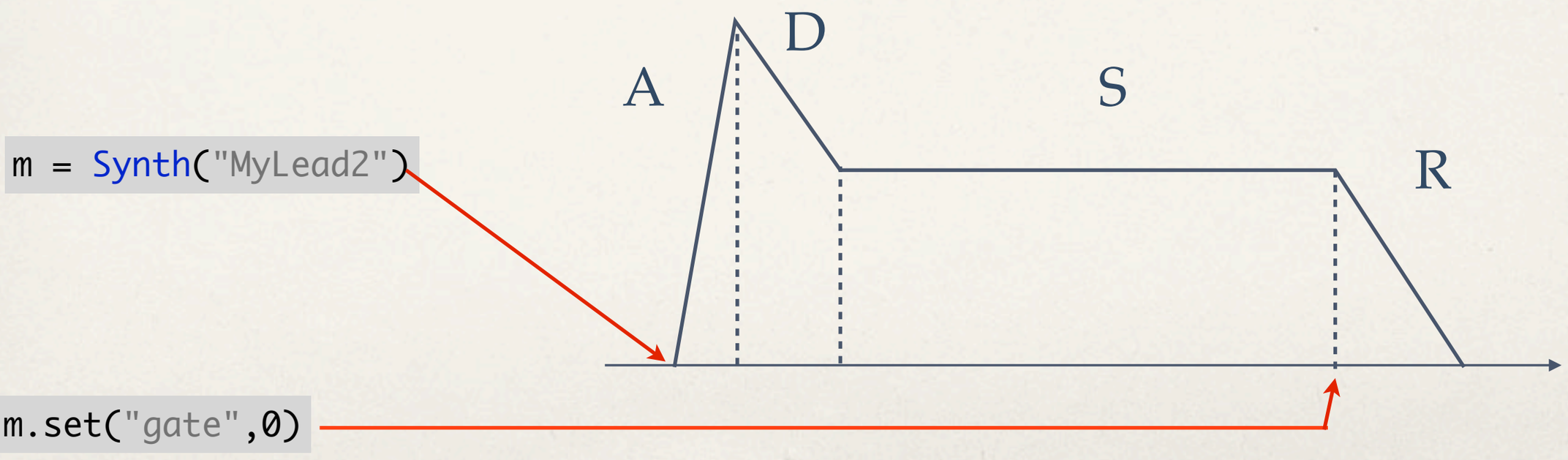


11. Hüllkurve mit umbestimmter Dauer

```
SynthDef("MyLead2",{
   arg freq = 440, amp = 0.5,  gate = 1;
   a = LFPulse.ar(freq) * amp;
   l = LPF.ar(a, XLine.ar(120, 4080, 1.0));
   e = Env.adsr(0.02, 0.2, 0.6, 0.3);
   x = EnvGen.ar(e, gate, doneAction: 2) * l;
   Out.ar(0, p);
}).load(s)
```

wenn *gate* größer als 0 ist,
wird der Ton gespielt.

wenn *gate* 0 ist,
wird der Ton gestoppt.



Demo: Mit MIDI Keyboard eine *Synth* spielen

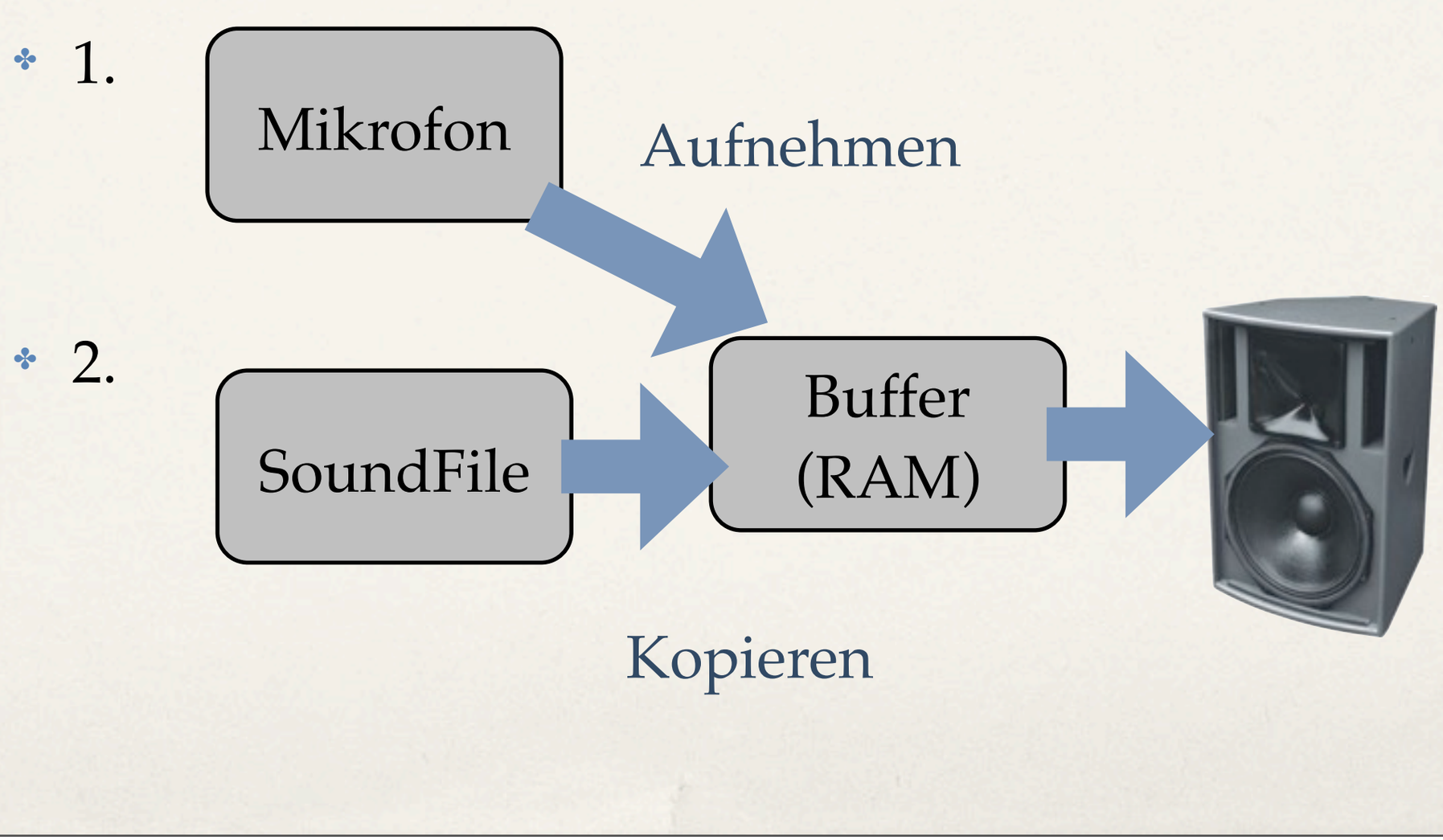
```
(
   MIDIIn.connect;
   MIDIIn.noteOn  = { arg src, chan, num, vel;    [chan,num,vel / 127].postln; };
   MIDIIn.noteOff = { arg src, chan, num, vel;    [chan,num,vel / 127].postln; };
)
```

```
(
   var keys= Array.newClear(127);
   MIDIIn.connect;
   MIDIIn.noteOn = { arg src, chan, num, vel;
      keys[num] = Synth("MyLead2", ["freq", num.midicps]);
   };

   MIDIIn.noteOff = { arg src, chan, num, vel;
      keys[num].set("gate",0);
   };
)
```

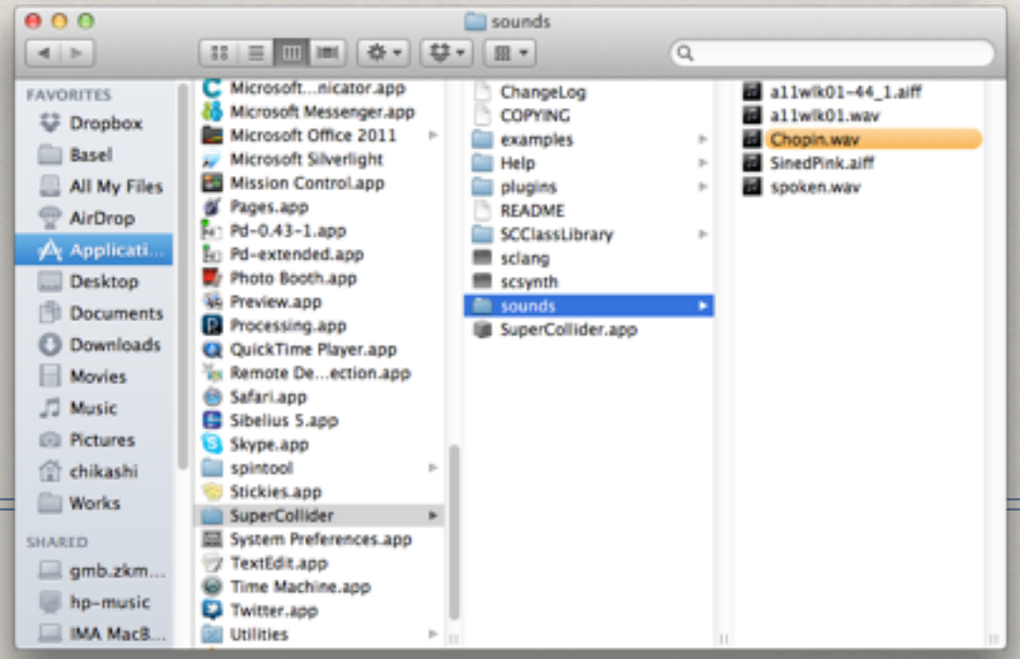
Projekt 2 : Sampler

1. Sample und Buffer

- Wie kann man einen Sample mit SC3 spielen?



1.a Buffer und Sample



- Kopiere deine Soundfiles in
  - /Programme/SuperCollider/sounds/

```
{
    b = Buffer.read(s, "sounds/chopin.wav");
    PlayBuf.ar(1, b, 1 , doneAction:2);
}.play
```

Pfad von SC3 Programme

der Buffer heißt *b*

Anzahl der Kanäle

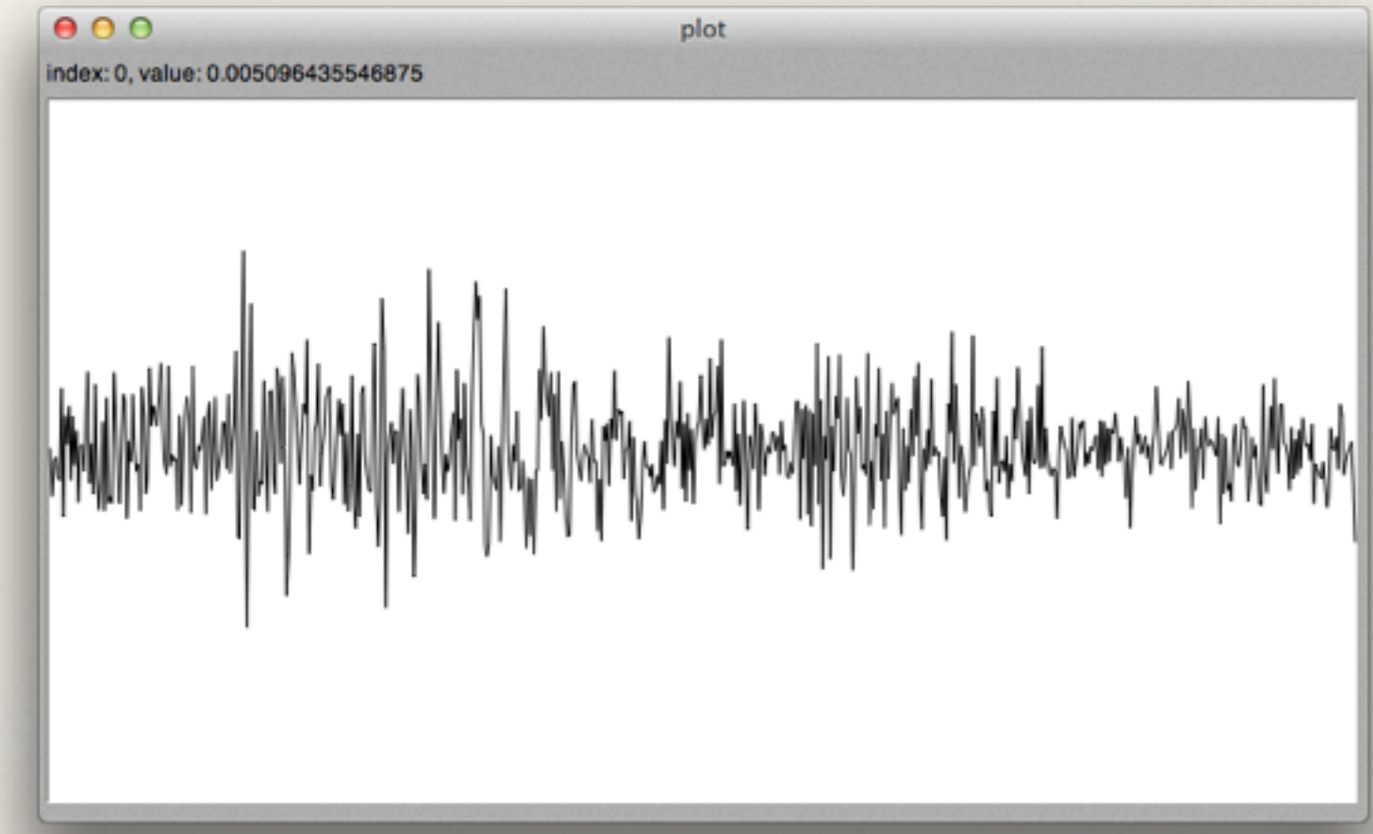
Geschwindigkeit

Schneller oder Langsamer

```
{
    b = Buffer.read(s, "sounds/chopin.wav");
    PlayBuf.ar(1, b, 0.5, doneAction:2);
    //PlayBuf.ar(1, b, 1.5, doneAction:2);
}.play
```

1.b Buffer und Sample

```
b = Buffer.read(s, "sounds/chopin.wav");
b.plot
```

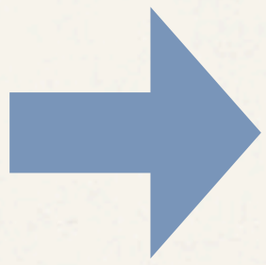


2.a Aufnahme mit Buffer

ACHTUNG: bitte benutze einen Kopfhörer

```
{
   SoundIn.ar(0); // Klang vom Mikrofon
}.play;
```

direkt

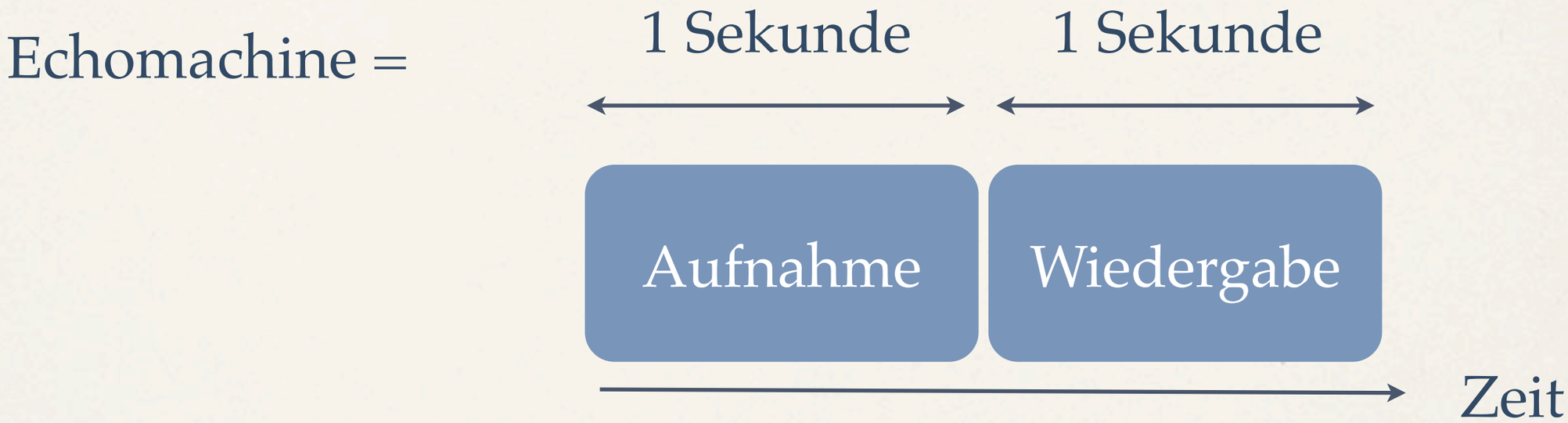


2.b Aufnahme mit Buffer

```
{
   b = Buffer.alloc(s, 44100); // einen Buffer zuweisen
   a = SoundIn.ar(0); // Klang von Mikrophon des Computers
   RecordBuf.ar(a, b, doneAction:2, loop: 0); // nimm auf
   0; // um Feedback-Problem zu vermeiden
}.play;

{
   a = PlayBuf.ar(1, b, doneAction:2 ); //spiele den Buffer
   Out.ar(0, a);
}.play
```

```
{
   a = PlayBuf.ar(1, b, 1.5); //schneller
   Out.ar(0, a);
}.play
```

3.a Echomachine



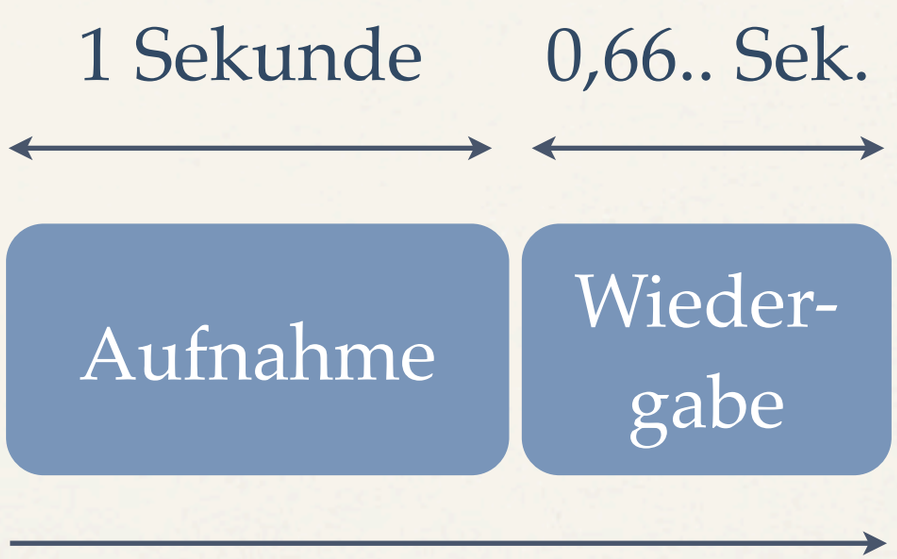
3.a Echomachine

```
(
   b = Buffer.alloc(s, 44100); //  einen Buffer zuweisen
   SynthDef("recording",{
         a = SoundIn.ar(0);
         RecordBuf.ar(a, b, doneAction:2, loop:0);
   }).load(s);

   SynthDef("playback",{
      arg rate = 1.0; // Geschwindigkeit
      a = PlayBuf.ar(1, b, rate, doneAction:2);
      Out.ar(0, a);
   }).load(s);
)

(
   Routine({
      Synth("recording"); // Aufnahme
      1.0.wait; //warte 1.0 Sekunde
      Synth("playback");  // Wiedergabe
   }).play;
)
```

3.b Echomachine

```
Routine({
   Synth("recording");
   1.0.wait;
   Synth("playback",["rate", 1.5]);
}).play;
```

schneller



3.c Echomachine

- Echo harmonizer

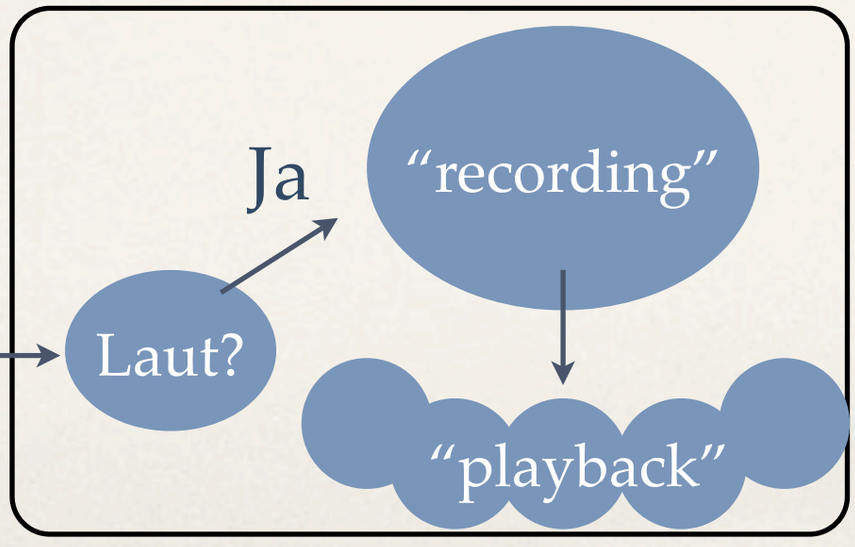
```
Routine({
   Synth("recording");
   1.0.wait;
   Synth("playback",["rate", 1.0]);
   Synth("playback",["rate", 64.midicps / 60.midicps]);
   Synth("playback",["rate", 67.midicps / 60.midicps]);
}).play;
```

Dur-Dreiklang

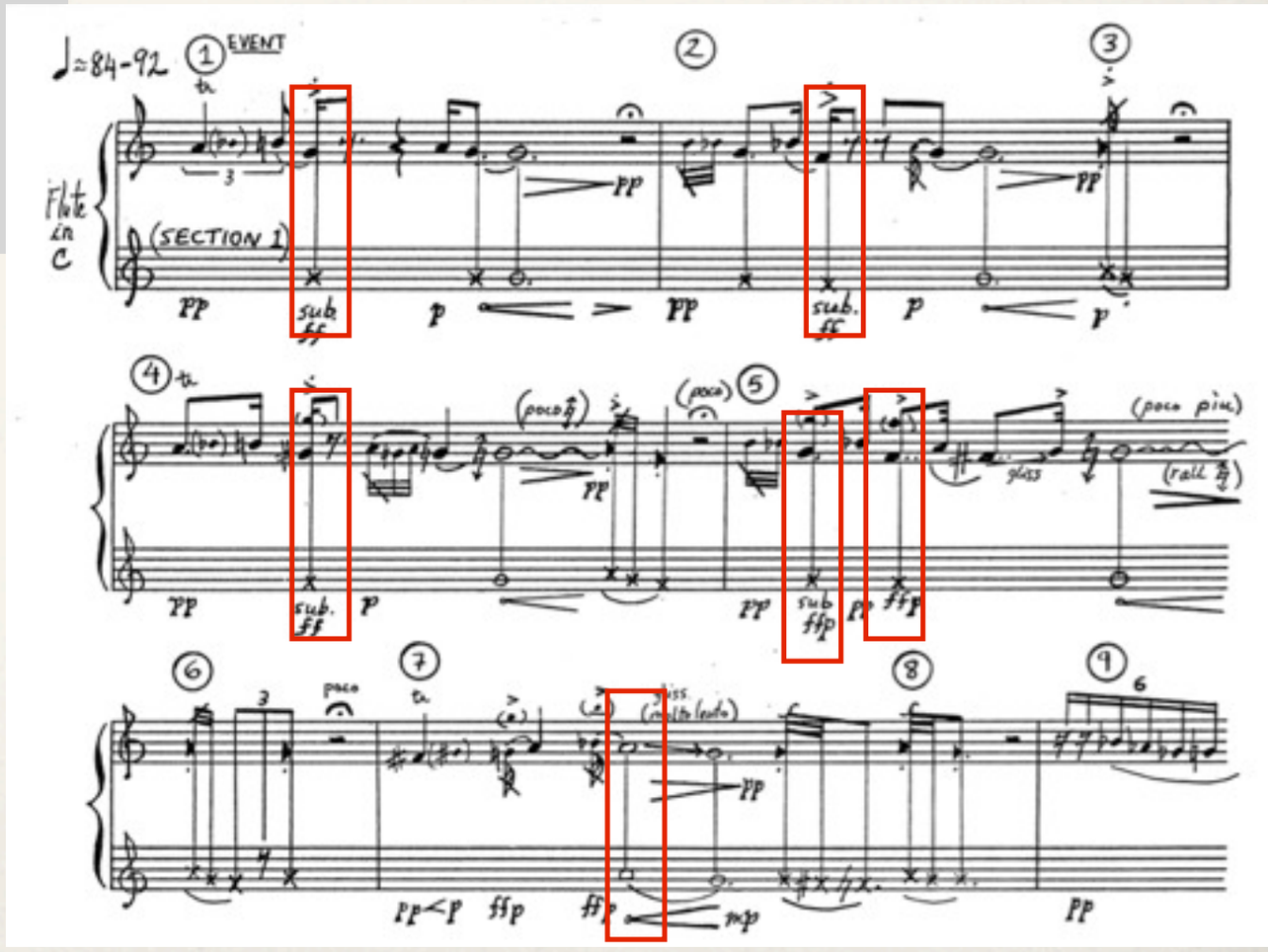
3.d Echomachine

```
Routine({
   Synth("recording");
   1.0.wait;
   Synth("playback",["rate", 0.5]);
   Synth("playback",["rate", 1]);
   Synth("playback",["rate", 2]);
   Synth("playback",["rate", 3]);
   Synth("playback",["rate", 4]);
}).play;
```

Flöte



Live-Elektronik Technik von Cort Lippe



4.a Loop

```
(
   b = Buffer.read(s,"sounds/Chopin.wav");
)

{
   PlayBuf.ar(1, b, 1 , loop:1); //loop mode
}.play

{
   PlayBuf.ar(1, b, Line.kr(1,2,15) , loop:1); //graduell schneller
}.play

{
   PlayBuf.ar(1, b, SinOsc.kr(1,0,0.3)+1.0, loop:1); //schneller und langsamer
}.play

{
   PlayBuf.ar(1, b, LFNoise2.kr(3,0.5)+1.0, loop:1); //Randomisierung
}.play
```

4.b Loop

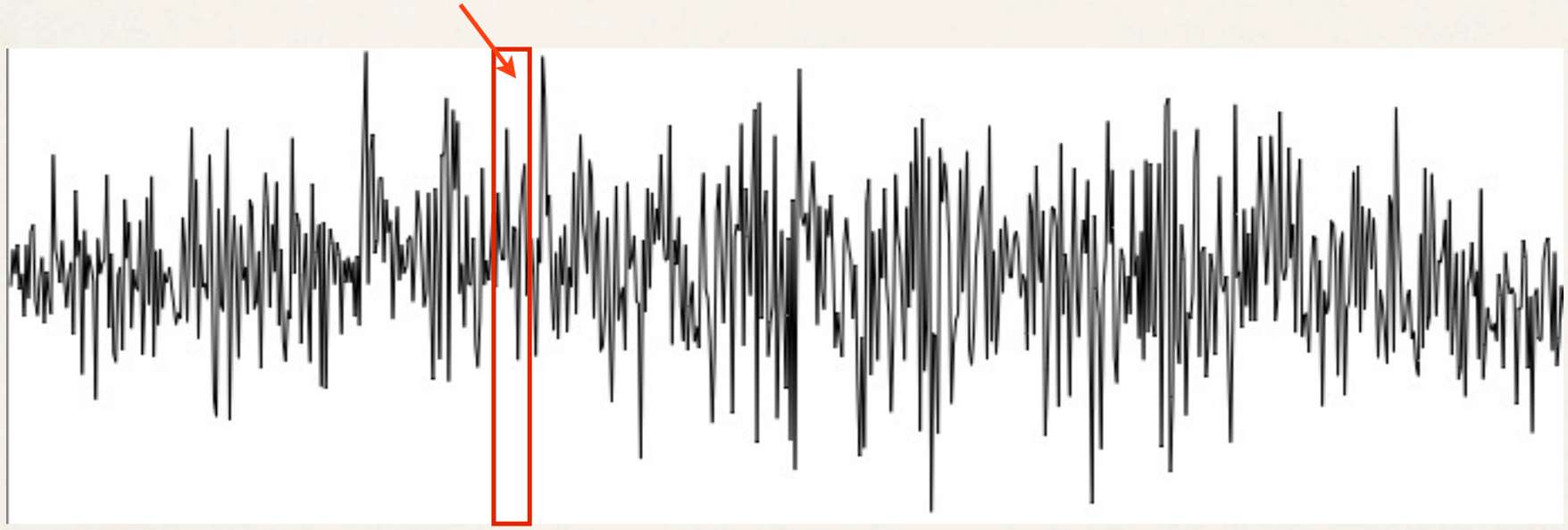
- Steve Reich “Come Out”

```
{
   b = Buffer.alloc(s, 44100);
   a = SoundIn.ar(0);
   RecordBuf.ar(a, b, 1 , doneAction: 2, loop:0);
   0;
}.play

{
   a = PlayBuf.ar(1, b, 1 , loop:1);
   b = PlayBuf.ar(1, b, 1.01 , loop:1);
   a + b;
}.play
```

5. Granular

- Was ist Granular Sampling ?

ca’ 20 - 100 msec: *grain (Getreide)*

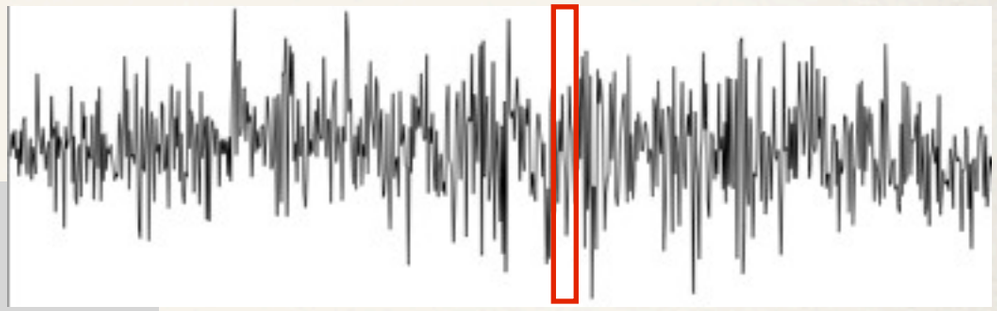


Wiederholung

Sound File

5.a Granular

```
(
   b = Buffer.read(s,"sounds/Chopin.wav");
)
//Granular Sampling mit 3 Zeilen
{
   TGrains.ar(2, Impulse.ar(50), b, 1.0, MouseX.kr(0, 8));
}.play
```

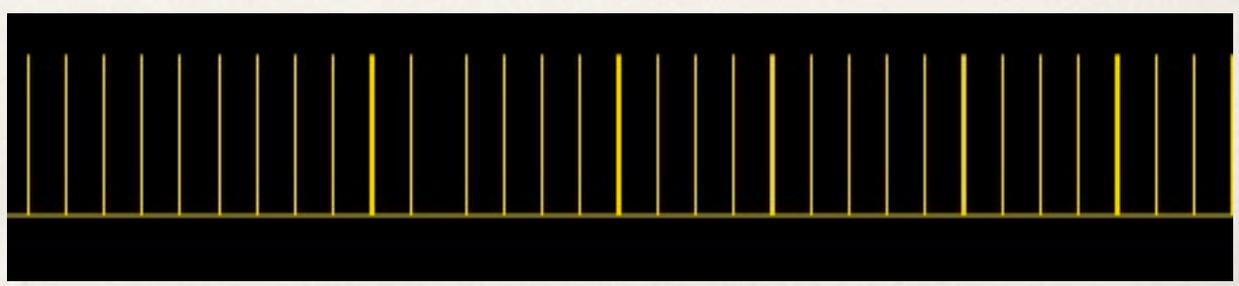


Anzahl der Kanäle

"Trigger"
Frequenz von
Grain-Produktion

Buffer

Geschwindigkeit



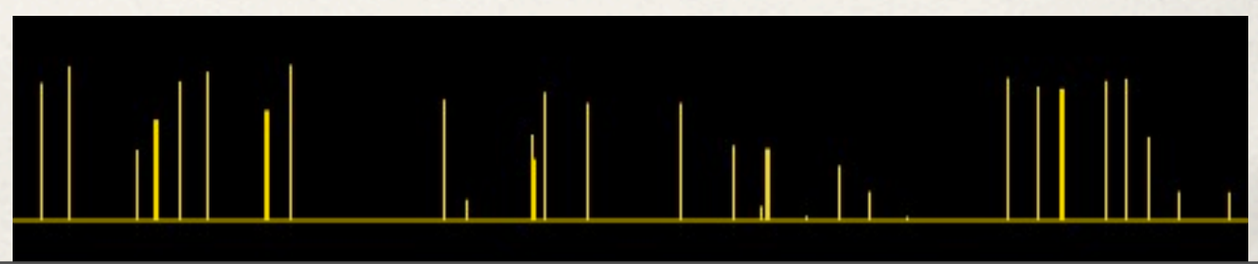
5.b Granular

```
//Mit Pitch Shift
{
   TGrains.ar(2, Impulse.ar(50), b, MouseY.kr(0.5, 2.0), MouseX.kr(0, 8) );
}.play
```

```
//MouseY kontrolliert die Frequenz der Grains.
{
   TGrains.ar(2, Impulse.ar(MouseY.kr(1, 50)), b, 1.0, MouseX.kr(0, 8) );
}.play
```

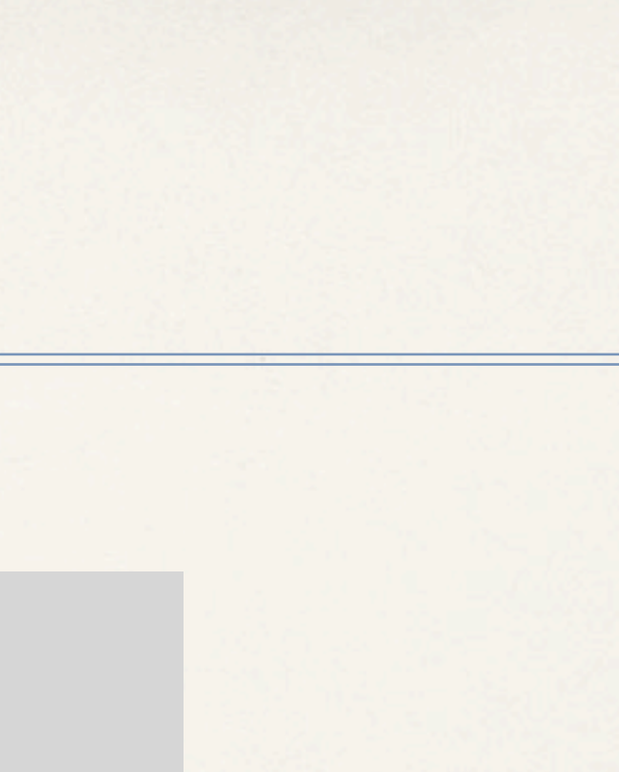
```
//Dust = unregelmäßige Impulse
{
   TGrains.ar(2, Dust.ar(MouseY.kr(1, 100)), b, 1.0, MouseX.kr(0, 8) );
}.play
```

Dust:



5.c Granular

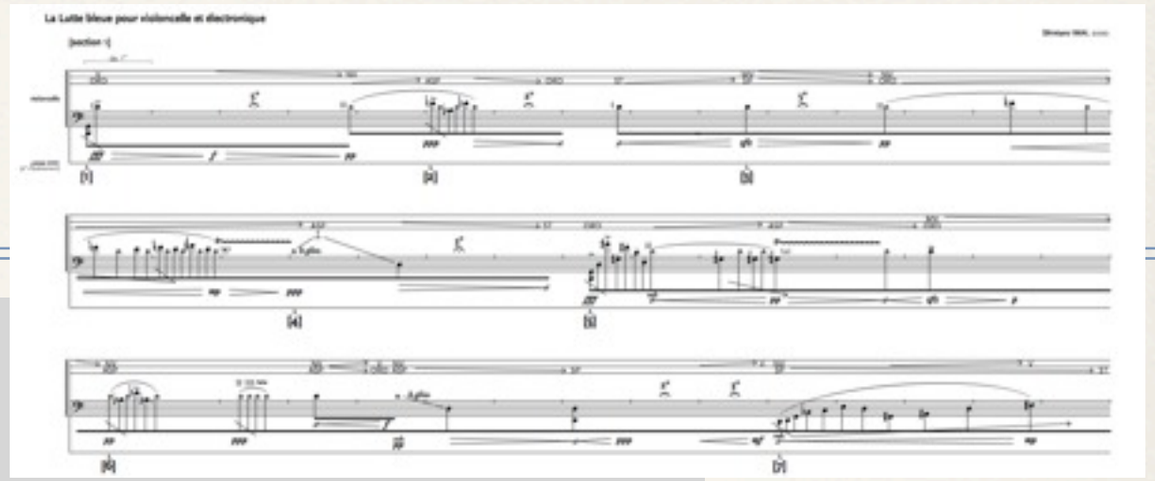
```
//MouseX kontrolliert Randombereich
{
   t = Impulse.ar(MouseY.kr(1, 50));
   TGrains.ar(2, t, b, 1.0, TRand.ar(0, MouseX.kr(0,10), t));
}.play
```



Trevor Wischart .. Fabulous Paris

5.d Granular

DSP Technik von Shintaro Imai



La Lutte Blaue für Cello und Live-Elektronik

```
(
   b = Buffer.read(s,"sounds/Cello.wav");
)

{
   i = Impulse.ar(50);
   r = MouseY.kr(0.5, 1.5);
   p = MouseX.kr(0, 10);
   TGrains.ar(2, i, b, r, p);
}.play;

{
   i = Impulse.ar(50);
   p = MouseX.kr(0, 10);
   r = MouseY.kr(0.5, 1.5);
   g = TGrains.ar(2, i, b, r, p);
   g = TGrains.ar(2, i, b, r * 2, p) +g; // Oberton 1
   g = TGrains.ar(2, i, b, r * 3, p) +g; // Oberton 2
   g = TGrains.ar(2, i, b, r * 4, p) +g; // Oberton 3
   g = TGrains.ar(2, i, b, r * 5, p) +g; // usw..
}.play
```

5.e Granular

Quasi-echtzeit Granular

Density for Harp

```
(
   b = Buffer.alloc(s, 44100);

   SynthDef("recording",{
      a = SoundIn.ar(0);
      RecordBuf.ar(a, b, doneAction:2, loop: 0);
   }).load(s);

   SynthDef("granular",{
      arg rate = 1.0;
      i = Impulse.kr(30);
      l = Line.kr(0.0,1.0,10.0 ,doneAction:2);
      g = TGrains.ar(2, i, b, rate ,TRand.kr(0, 0.01, i) + l);
      Out.ar(0,g);
   }).load(s);
)

Routine({
   Synth("recording");
   1.0.wait;
   Synth("granular");
}).play;
```

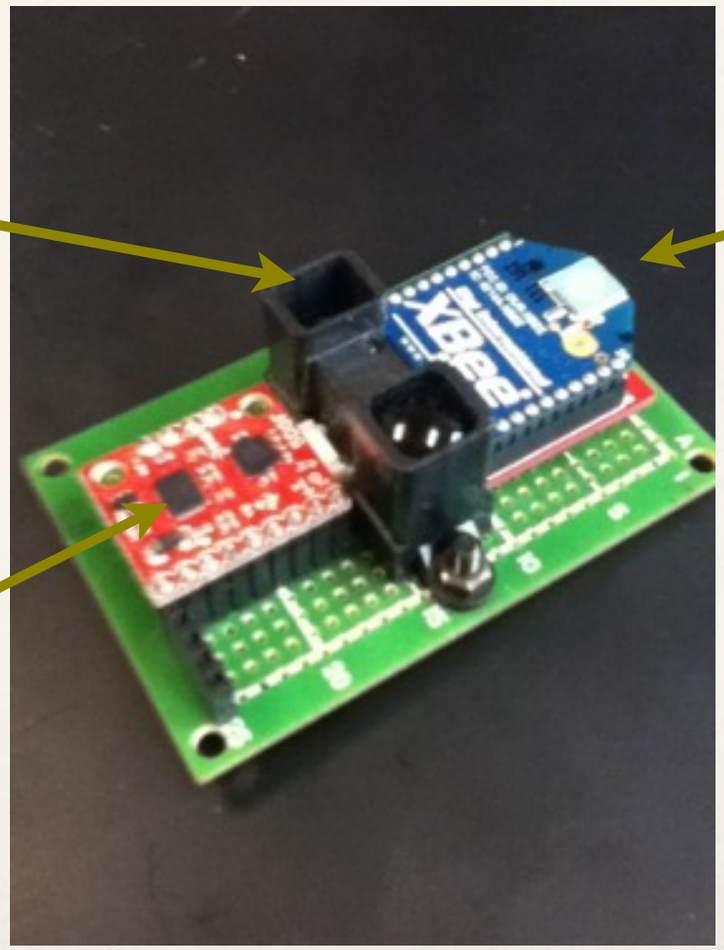
5.f Granular

Dur-Dreiklang

```
Routine({
   Synth("recording");
   1.0.wait;
   Synth("granular");
   Synth("granular", ["rate", 64.midicps / 60.midicps]);
   Synth("granular", ["rate", 67.midicps / 60.midicps]);
}).play;
```

Demo

- Granular mit Sensoreninstrument Qgo2



Infrarotsensor

Xbee Antenna

5 Degrees of Freedom

Beschleunigungssensor + Gyrosensor

ACHTUNG: bitte benutze einen Kopfhörer

Projekt 3 : Effekt und Interaktivität

1. SoundIn

ACHTUNG: bitte benutze einen Kopfhörer.

```
{
    SoundIn.ar(0);
}.play
```

der linke Kanal des Mikrofons

Direkte Verbindung

2.a Delay

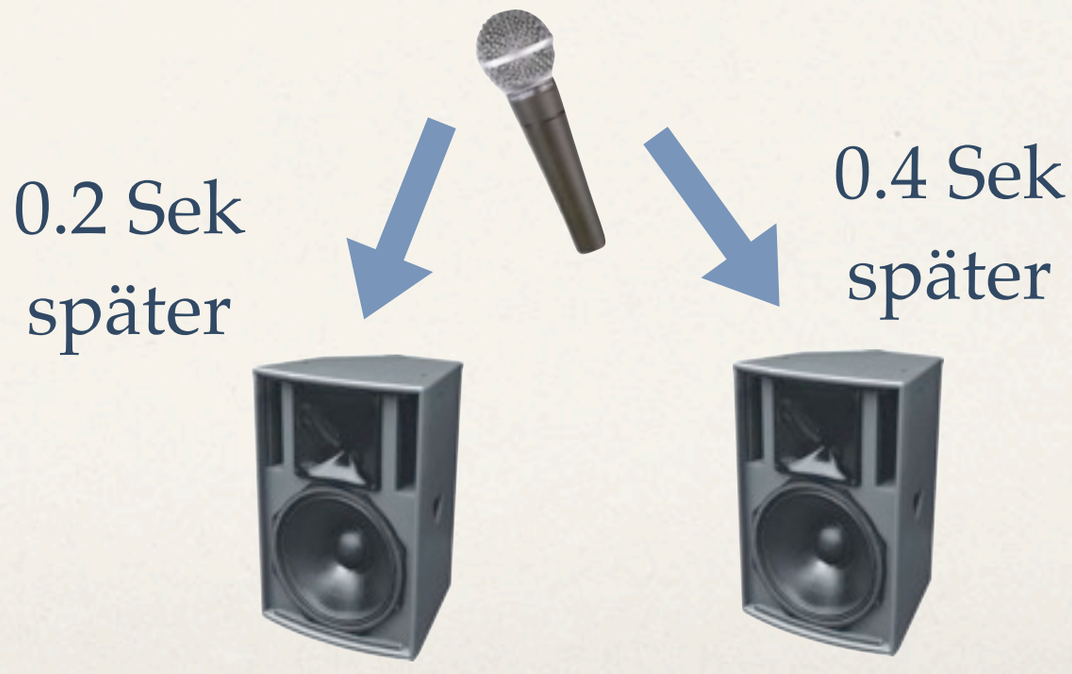
Delay

```
{
   a = SoundIn.ar(0);
   DelayN.ar(a, 0.5, 0.5);
}.play
```

DelayN.ar(*in, maxdelaytime, delaytime* ...)

Ping Pong Delay (L->R)

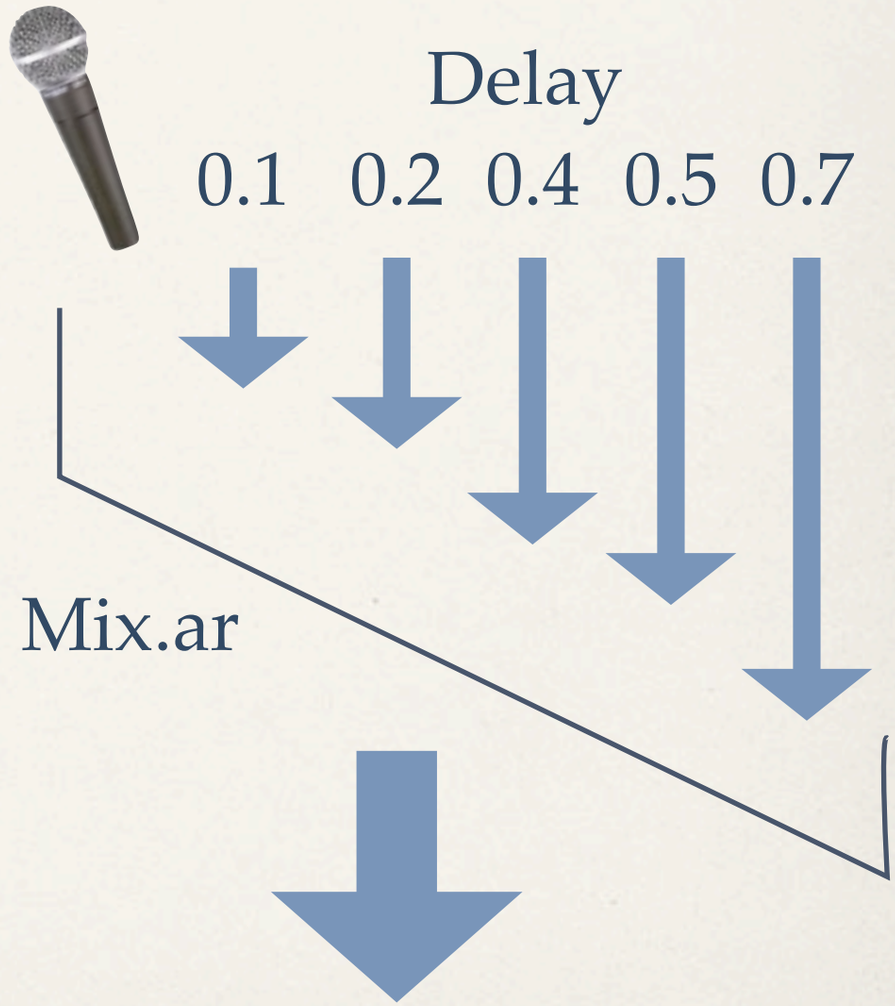
```
{
   a = SoundIn.ar(0);
   l = DelayN.ar(a, 0.2, 0.2);
   r = DelayN.ar(a, 0.4, 0.4);
   [l,r];
}.play
```



2.b Delay

Multi tap Delay

```
{
   a = SoundIn.ar(0);
   b = DelayN.ar(a, 0.7, 0.1);
   b = b + DelayN.ar(a, 0.7, 0.2);
   b = b + DelayN.ar(a, 0.7, 0.4);
   b = b + DelayN.ar(a, 0.7, 0.5);
   b = b + DelayN.ar(a, 0.7, 0.7);
}.play
```



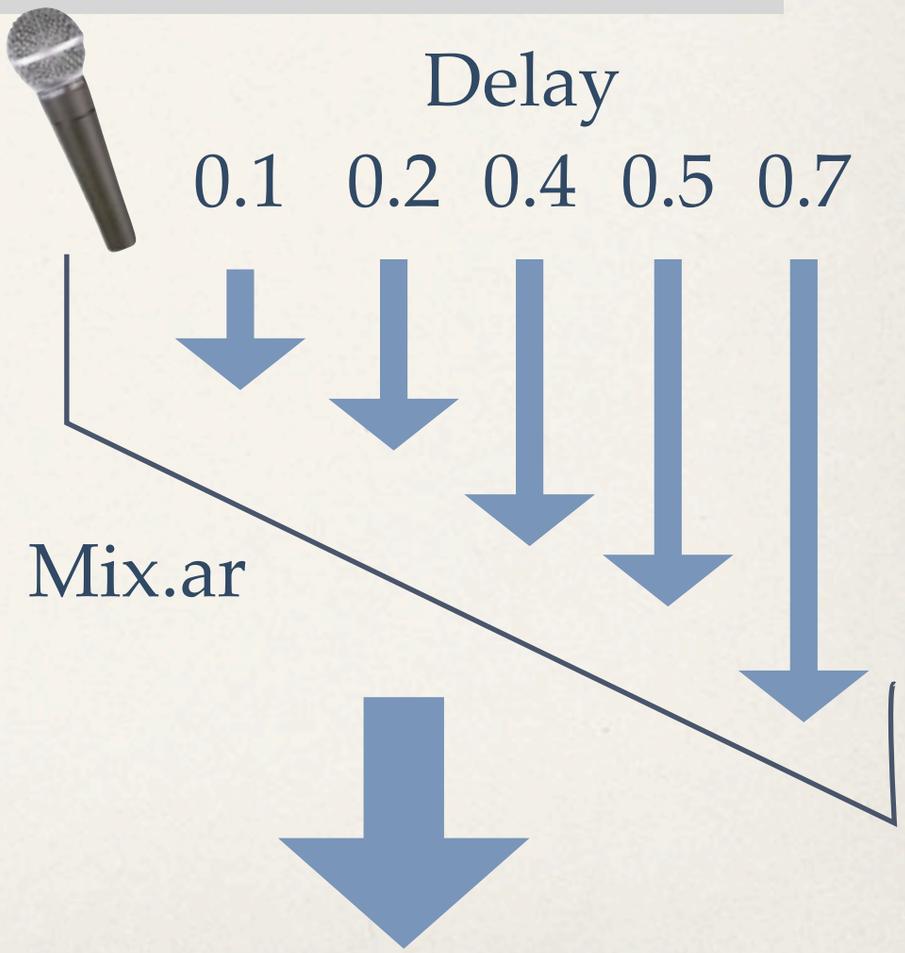
2.c Delay

Mix.ar

```
{
   //Drei Klänge in einem Array bzw. SinOsc, Saw und WhiteNoise
   a = [SinOsc.ar(465,0.0,0.4), Saw.ar(120,0.2), WhiteNoise.ar(0.05)];
   Mix.ar(a); //Mische drei Klänge
}.play
```

Mit DelayN

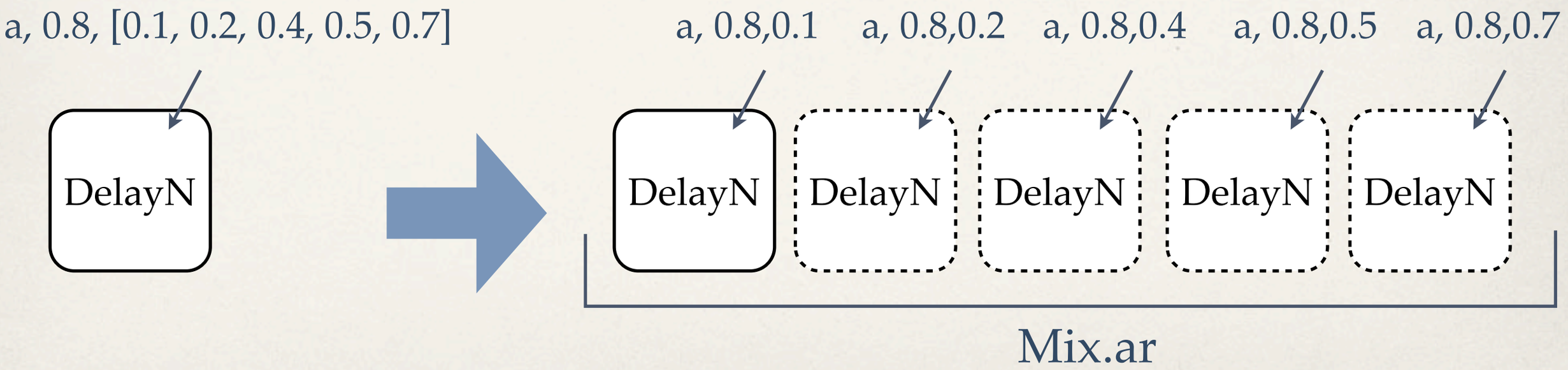
```
{
   a = SoundIn.ar(0);
   //5 Delay Lines
   b = [
      DelayN.ar(a, 0.7, 0.1),
      DelayN.ar(a, 0.7, 0.2),
      DelayN.ar(a, 0.7, 0.4),
      DelayN.ar(a, 0.7, 0.5),
      DelayN.ar(a, 0.7, 0.7)];
   Mix.ar(b);//Mische alle
}.play
```



2.d Delay

```
{
   a = SoundIn.ar(0);
   Mix.ar(DelayN.ar(a, 0.8, [0.1, 0.2, 0.4, 0.5, 0.7]));
}.play
```

Wenn man einen Array als Argument gibt, verfünffacht SC3 automatisch die Ugen.



Tipp: Array-Mix.ar Methode

- Man kann die *Array-Mix.ar* Methode auf andere Programme anwenden.

```
{
   Mix.ar(SinOsc.ar([440,660,880]));
}.play
```

```
{
   n = WhiteNoise.ar(5);
   Mix.ar(BPF.ar(n, [440,660,880], 0.01));
}.play
```

```
{
   Mix.ar(SinOsc.ar(XLine.kr(220,[330,440,550,660], 15))) * 0.1;
}.play
```

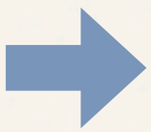
Tipp: Array.fill

- Array.fill(x, y) generiert einen Array mit *x* Räume und füllt alle Räume mit *y*.

```
Array.fill(3, 5)
```

```
[ 5, 5, 5 ]
```

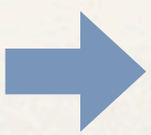
```
Array.fill(6, 2)
```



```
[ 2, 2, 2, 2, 2, 2 ]
```

- Man kann auch eine Funktion {} als das zweite Argument verwenden.

```
Array.fill(3, {10.rand})
```



```
[ 6, 1, 8 ]
```

Tipp: Array.fill

- Mix.ar-Array Methode

```
{
   Mix.ar(SinOsc.ar([440,660,880]));
}.play
```

- Mix.ar-Array.fill

```
{
   Mix.ar(SinOsc.ar(Array.fill(3, {400.rand + 400})));
}.play
```

Drei Sinuswellen mit randomisierten Frequenzen (Frequenzbereich : 400 - 800)

```
{
   Mix.ar(SinOsc.ar(Array.fill(40, {1000.rand + 200}))) * 0.1;
}.play
```

3. Ring Modulation

Stockhausen *Mantra*

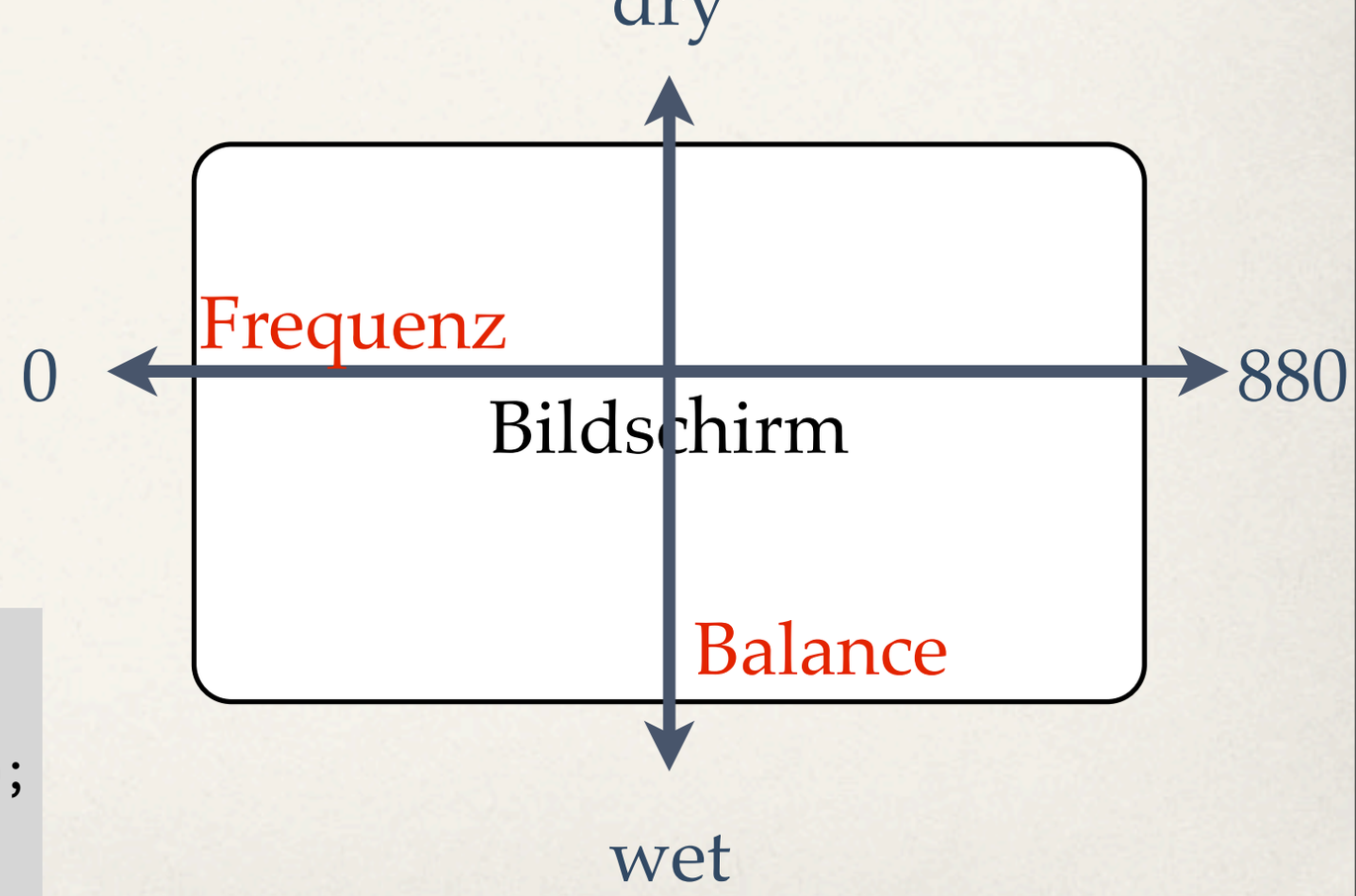
```
{
   a = SoundIn.ar(0);
   a * SinOsc.ar(220);
}.play
```

per Maus kontrollieren

```
{
   a = SoundIn.ar(0);
   a * SinOsc.ar(MouseX.kr(0, 880));
}.play
```

Dry-Wet Balance

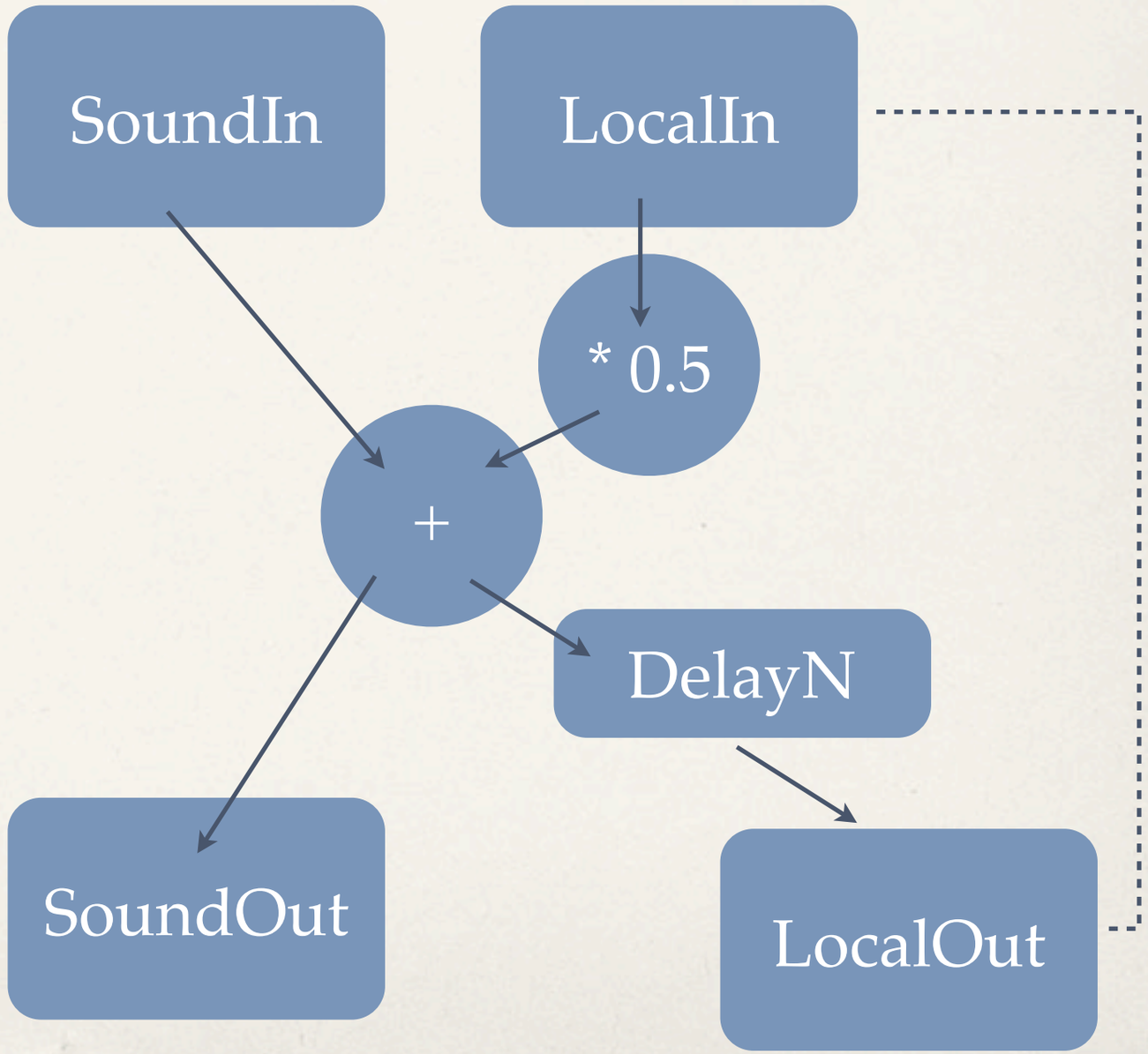
```
{
   a = SoundIn.ar(0);
   b = a * SinOsc.ar(MouseX.kr(0, 880));
   XFade2.ar(a, b, MouseY.kr(-1,1));
}.play
```



4.a Feedback / Flanger

Feedback Delay

```
{
   a = SoundIn.ar(0);
   l = LocalIn.ar(1) * 0.5;
   m = a + l;
   d = DelayN.ar(m, 0.2, 0.2);
   LocalOut.ar(d);
   Out.ar(0, m);
}.play;
```



4.b Feedback / Flanger

Flanger

```
{
   a = SoundIn.ar(0);
   l = LocalIn.ar(1) * 0.95;
   m = a + l;
   d = DelayC.ar(m, 0.5, SinOsc.kr(0.05, 0.0, 0.003) + 0.006);
   LocalOut.ar(d);
   Out.ar(0, m);
}.play;
```

Chorus

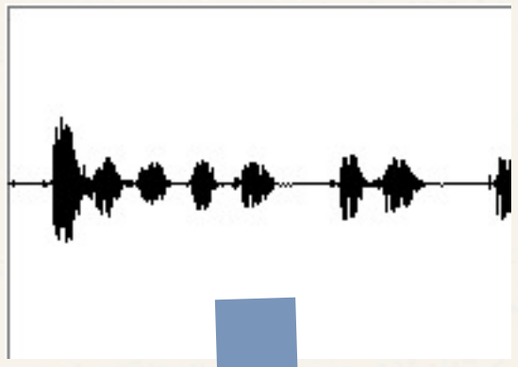
```
{
   a = SoundIn.ar(0);
   l = LocalIn.ar(1) * 0.1;
   m = a + l;
   f = Array.fill(4, {0.05.rand + 0.05});
   d = Mix.ar(DelayC.ar(m, 0.5, SinOsc.kr(f, 0.0, 0.05) + 0.15));
   LocalOut.ar(d);
   Out.ar(0, m);
}.play;
```

5. Distortion

```
{
   a = SoundIn.ar(0);
   (a * 40).clip2(0.5);
}.play
```

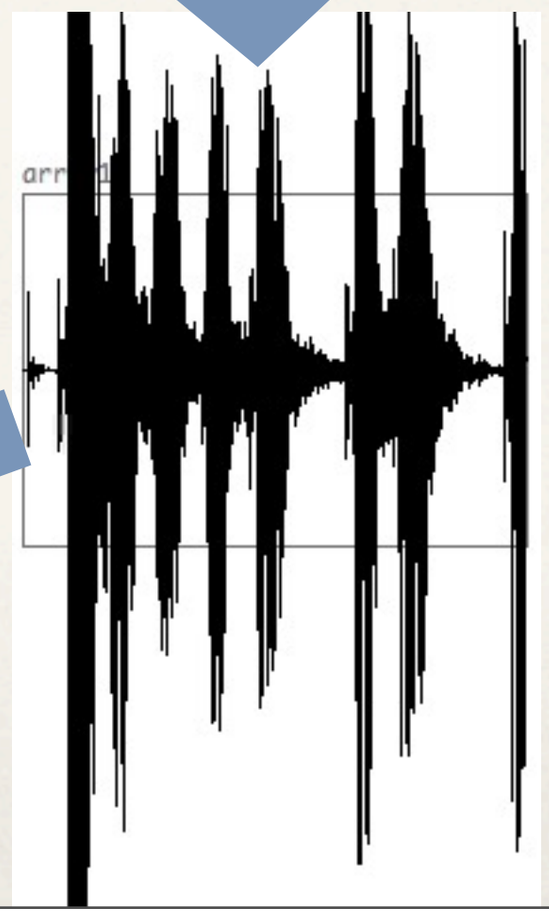
Clipping

Verstärkung



Verstärkung

x 40



Clipping

0.5

-0.5

6. Auto Panner

- Panning in SC3

```
{
   w = WhiteNoise.ar;
   Pan2.ar(w, 1.0); // -1.0 Links, 1.0 Rechts, 0.0 Mitte
}.play
```

- von links nach rechts

```
{
   w = WhiteNoise.ar;
   Pan2.ar(w, Line.kr(-1.0, 1.0, 1.0)); // von Links nach Rechts. dauert 1 Sekunde.
}.play
```

- Auto Panning

```
{
   w = WhiteNoise.ar;
   Pan2.ar(w, SinOsc.kr(1.0)); // 1 Hz.
}.play
```

6. Auto Panner

- Auto panning

```
{
   a = SoundIn.ar(0);
   Pan2.ar(a, SinOsc.kr(1.0)); // 1 Hz.
}.play
```

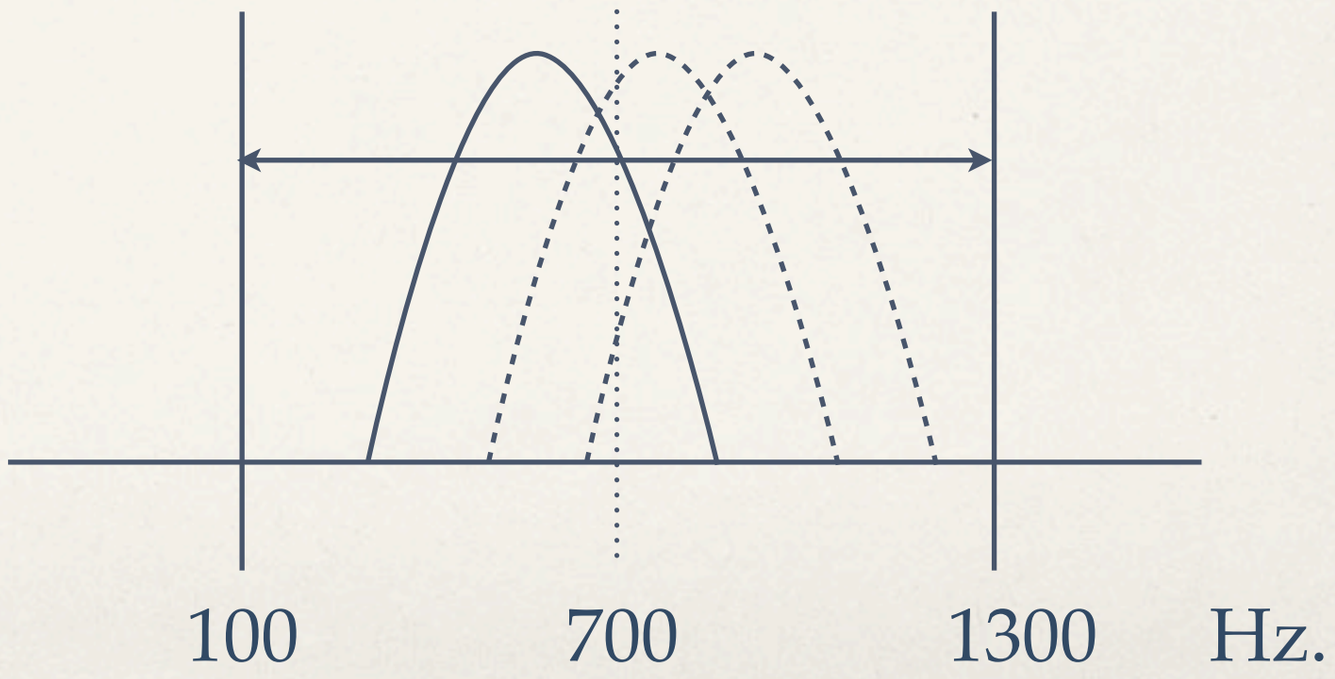
- Random panning

```
{
   a = SoundIn.ar(0);
   Pan2.ar(a, LFNoise1.kr(1.0)); // 1 Hz.
}.play
```

7. WahWah

- Wahwah

```
{
  a = SoundIn.ar(0);
  BPF.ar(a, SinOsc.kr(3, 0.0, 600) + 700, 0.4);
}.play;
```



8.a Pitchshifter / Harmonizer

```
{
   m = SoundIn.ar(0);
   PitchShift.ar(m, 0.2, 0.5); //  eine Oktave tiefer
}.play
```

window size

rate

```
{
   m = SoundIn.ar(0);
   PitchShift.ar(m, 0.2, 0.5, 0.5); // mit pitch deviation
}.play
```

8.b Pitchshifter / Harmonizer

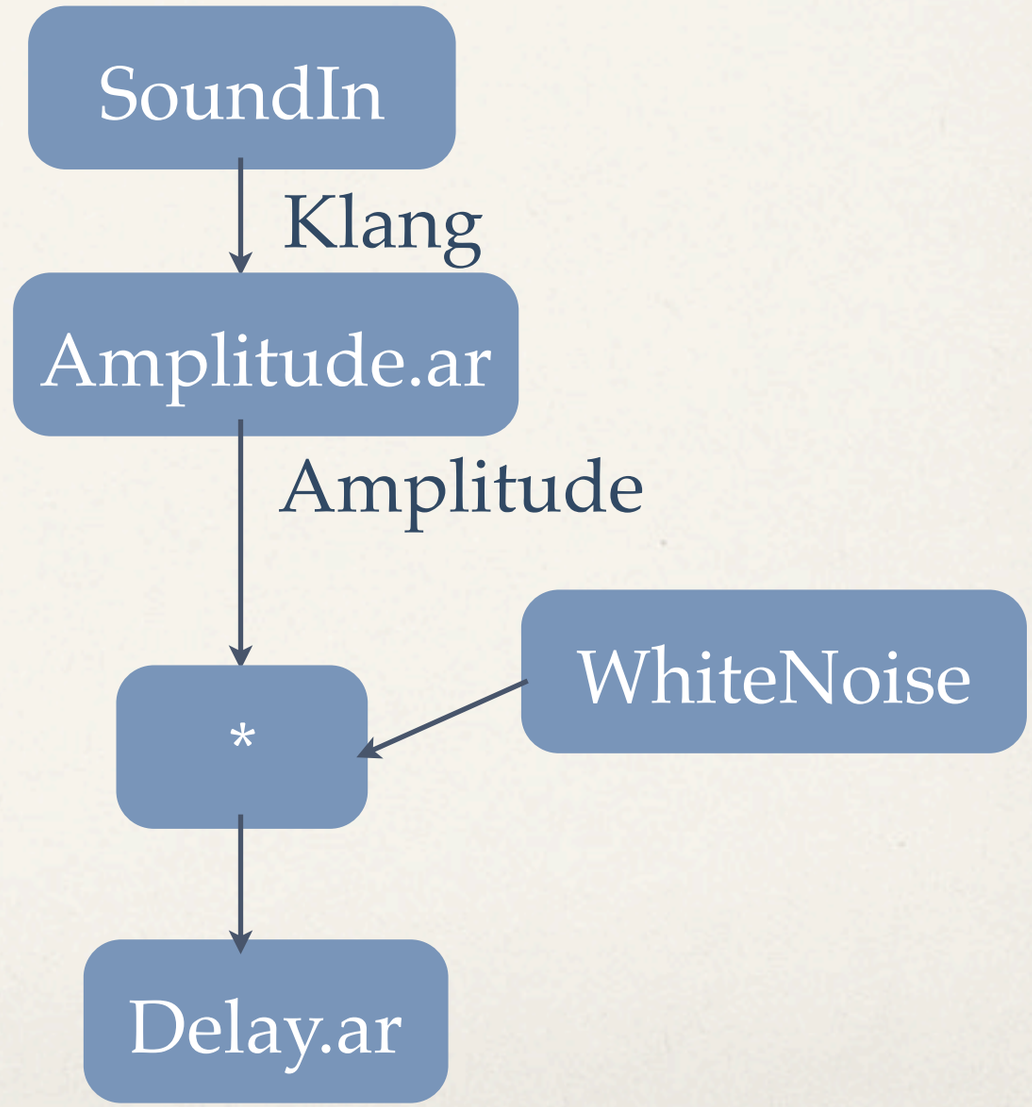
- Harmonizer

```
{
  m = SoundIn.ar(0);
  a = PitchShift.ar(m, 0.2, 57.midicps / 60.midicps );       // terz
  a = a + PitchShift.ar(m, 0.2, 53.midicps / 60.midicps);  // quinte
}.play
```

9.a Amplitude Follower

- Man kann mit Amplitude.ar die Lautstärke analysieren.

```
{
   a = SoundIn.ar(0);
   b = Amplitude.ar(a);
   c = WhiteNoise.ar * b;
   d = DelayN.ar(c, 1, 1);
}.play
```

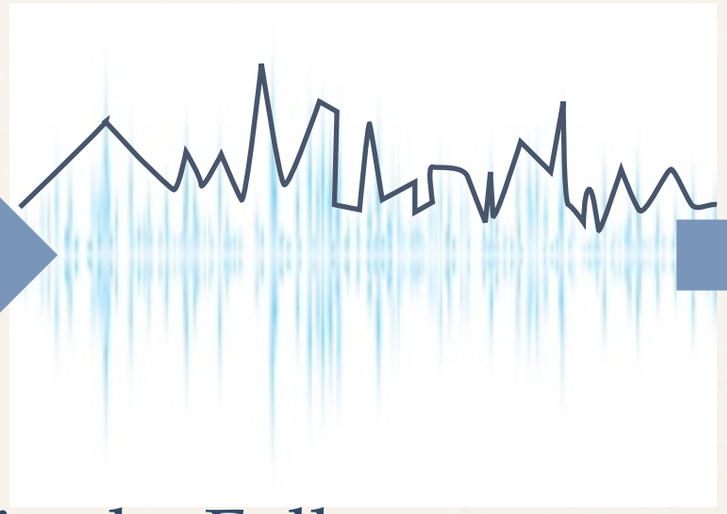


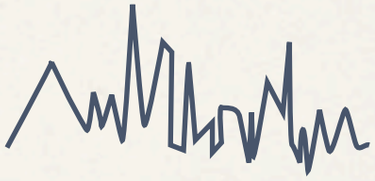
Die Amplitude vom eingegebenen Klang kontrolliert die Amplitude von WhiteNoise.

9.b Amplitude Follower

Oboe Control von Thomas Kessler

H.Holliger gewidmet





Stauchen x 0,5

Strecken x 1,2

mit Amplitude Follower

Amplitude Kontur analysieren

* Soundfiles

Die Lautstärke der anderen Klänge (Sound Files) damit kontrollieren

9.b Amplitude Follower

```
{
   p = Buffer.alloc(s, 88200);
   a = SoundIn.ar(0);
   b = Amplitude.ar(a);
   RecordBuf.ar(b,p, doneAction:2, loop:0);
}.play

{
   PlayBuf.ar(1, p, doneAction:2) * WhiteNoise.ar;
}.play

{
   PlayBuf.ar(1, p, 0.5, doneAction:2) * WhiteNoise.ar;
}.play
```

← Langsamer

10. Pitch Tracker

- Man kann mit Pitch.kr die Tonhöhe analysieren.

```
{
   a = SoundIn.ar(0);
   #b, c = Pitch.kr(a, ampThreshold: 0.1);
   d = Pulse.ar(b) * c;
   DelayN.ar(d, 1, 1);
}.play
```

Pitch.kr gibt zwei Daten zurück.

b ... Frequenz
c ... hat Tonhöhe? (nicht still oder Geräusch?)

Man muss vor b und c # schreiben, wenn ein Ugen zwei Daten zurückgibt.

11.a Reverb

```
{
   a = SoundIn.ar(0);
   FreeVerb.ar(a, 0.9, 0.5, 0.5);
}.play
```

dry / wet
(0 - 1)

Raum
(0 - 1)

HP damp

```
{
   a = SinOsc.ar(LFNoise2.kr(10, 520)+840) * 0.1;
   FreeVerb.ar(a, MouseX.kr(0,1), 0.5, 0.5);
}.play
```

11.b Reverb

GVerb ... Reverb mit mehreren Parametern

```
{
   var roomsize, revtime, damping, inputbw, spread, drylevel, earlylevel,taillevel;
   roomsize = 15; // Quadratmeter
   revtime = 0.6; // Sekunde
   damping = 0.5; // HF(Hochfrequenz) Damping <0 - 1>
   inputbw = 0.5; // Input Damping <0 - 1>
   spread = 15;
   drylevel = -6;
   earlylevel = -15;
   taillevel = -17;

   a = SinOsc.ar(LFNoise2.kr(10, 520)+840) * 0.005;
   GVerb.ar(a, roomsize, revtime, damping, inputbw, spread, drylevel, earlylevel, taillevel
}.play
```

12.a Kombination

- Ping Pong Multitap Delay

```
{
   a = SoundIn.ar(0);
   l = DelayN.ar(a, 1.0, [0.1, 0.3, 0.5]);
   r = DelayN.ar(a, 1.0, [0.2, 0.4, 0.6]);
   [l,r];
}.play
```

```
{
   a = SoundIn.ar(0);
   l = DelayN.ar(a, 1.0, [0.1, 0.5, 0.7]);
   r = DelayN.ar(a, 1.0, [0.2, 0.4, 0.8]);
   [l,r];
}.play
```

12.b Kombination

- Amplitude kontrolliert die Frequenz des Pannings

```
{
   a = SoundIn.ar(0);
   t = Amplitude.kr(a) * 100 + 3;
   Pan2.ar(a, SinOsc.kr(t));
}.play
```

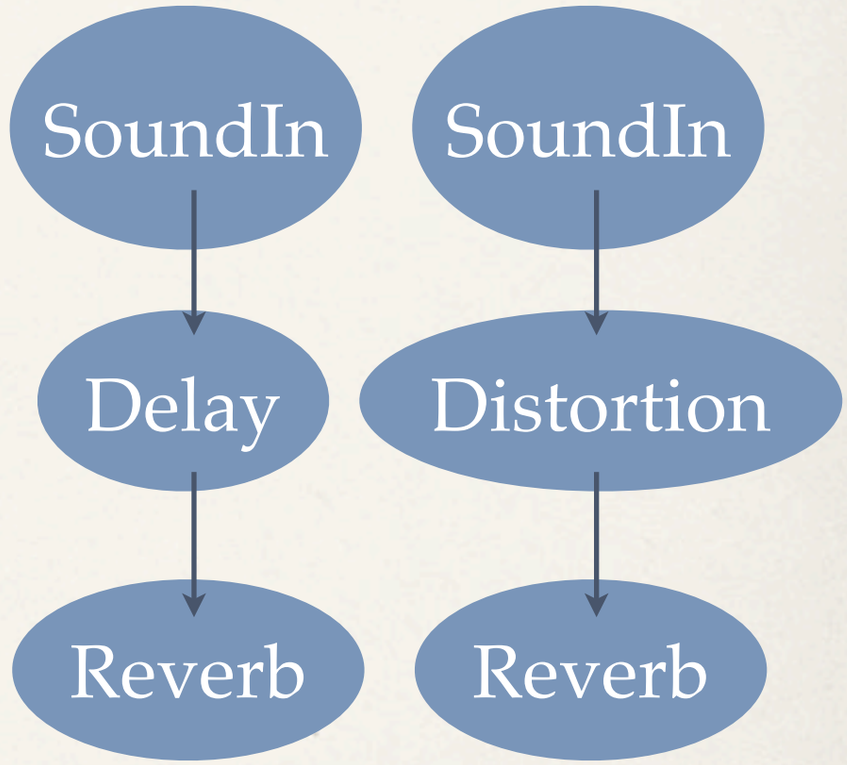
- Amplitude kontrolliert die Frequenz von FM

```
{
   a = SoundIn.ar(0);
   t = Amplitude.kr(a) * 100;
   SinOsc.ar(440 + SinOsc.kr(t,0.0,200));
}.play
```

12.c Kombination

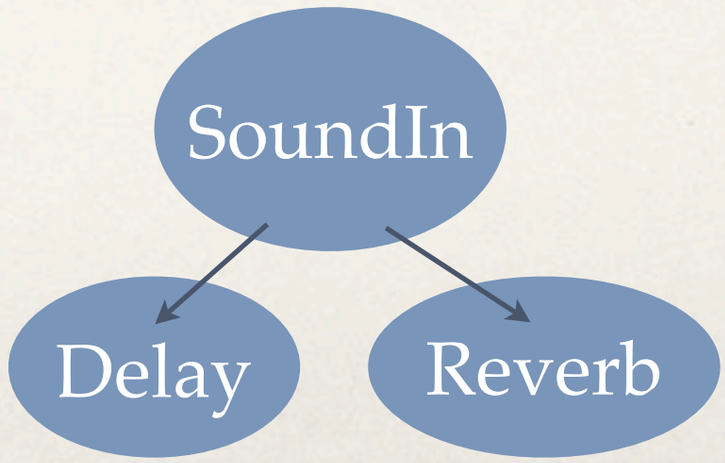
- Serielle Verbindung

```
{
   a = SoundIn.ar(0);
   d = DelayN.ar(a, 0.5, 0.5);
   r = FreeVerb.ar(d);
}.play
```



- Parallele Verbindung

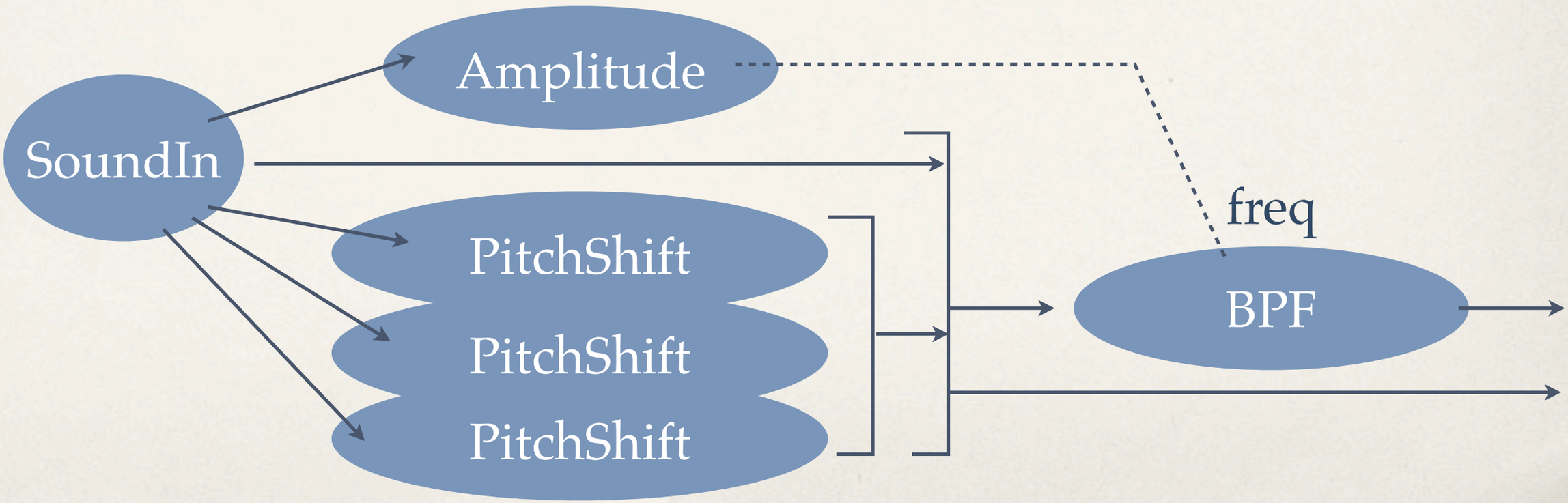
```
{
   a = SoundIn.ar(0);
   d = DelayN.ar(a, 0.5, 0.5);
   r = FreeVerb.ar(a);
   d + r;
}.play
```



12.d Kombination

- Fat Lead Effect

```
{
  i = SoundIn.ar(0);
  a = Amplitude.kr(i) ;
  i = i + Mix.ar(PitchShift.ar(i, 0.2, [0.25, 0.5, 2.0]));
  i = i + BPF.ar(i , LFCub.kr(a * 20,0,800) + 880,0.5);
}.play;
```



12.e Kombination

- Eingegebener Klang kontrolliert Granular

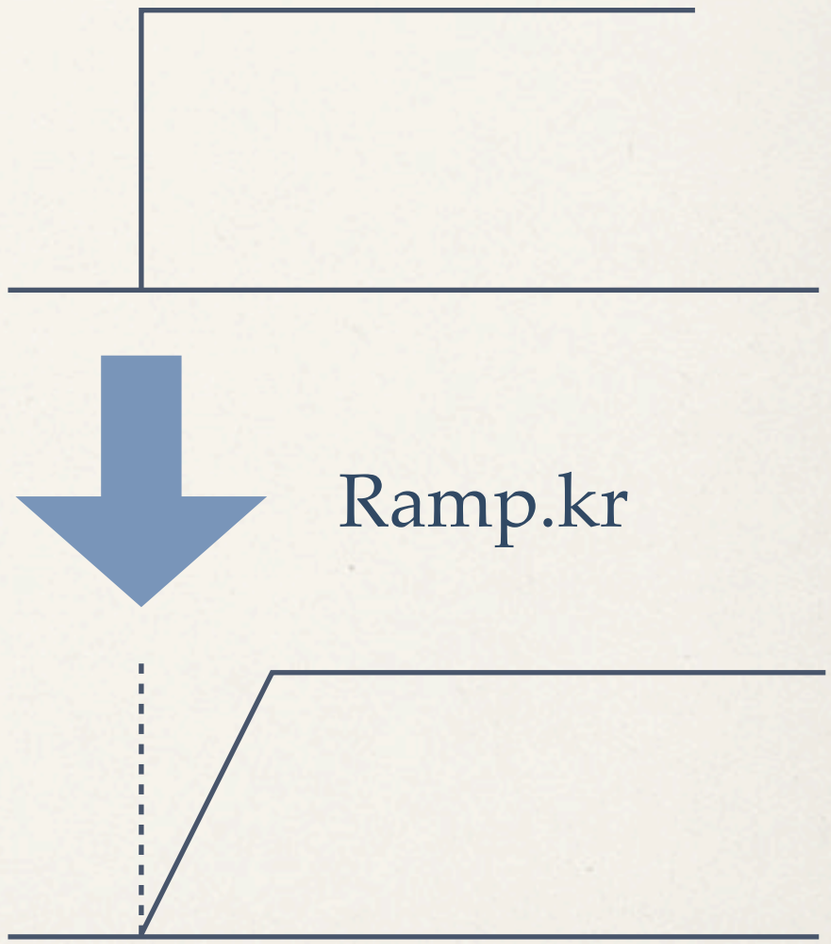
```
(
   b = Buffer.read(s,"sounds/Chopin.wav");
)

{
   a = SoundIn.ar(0);
   x = Amplitude.kr(a);
   #y, z = Pitch.kr(a);
   TGrains.ar(2, Impulse.ar(x * 100), b, y * z / 400 + 1.0, MouseX.kr(0, 8) );
}.play
```

Tipp: Glättung

- Ramp.kr

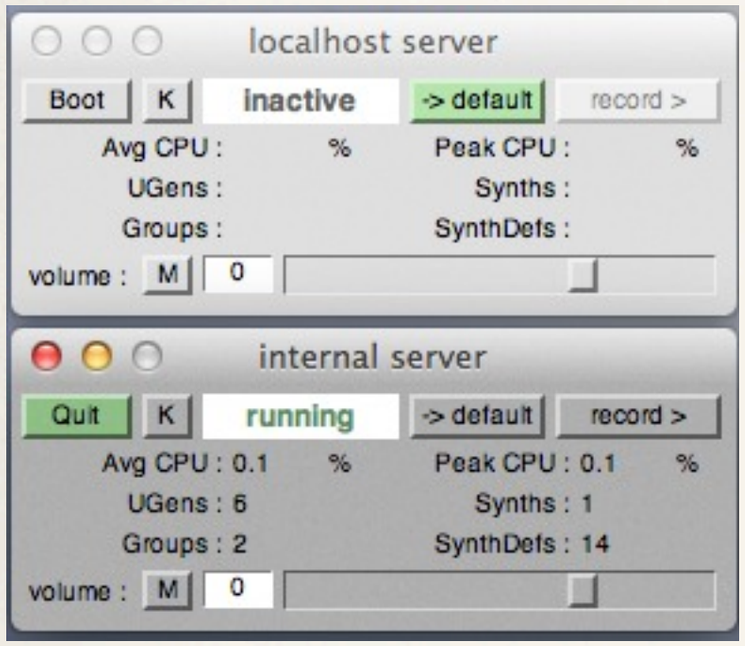
```
{
   a = SoundIn.ar(0);
   #b, c = Pitch.kr(a, ampThreshold: 0.1);
   d = Pulse.ar(Ramp.kr(b, 0.05)) * c;
   DelayN.ar(d, 1, 1);
}.play
```

LagTime



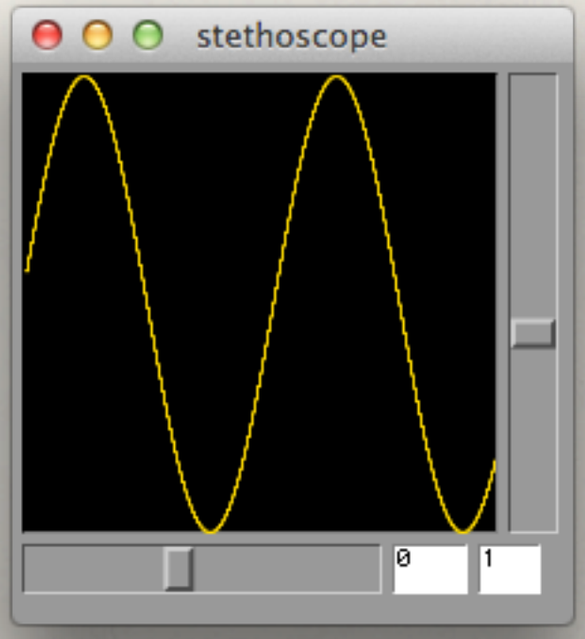
Graphische Oberfläche

1.a Oszilloskop



Schalte *internal server* statt *localhost server* ein

```
{
  SinOsc.ar(440);
}.scope
```

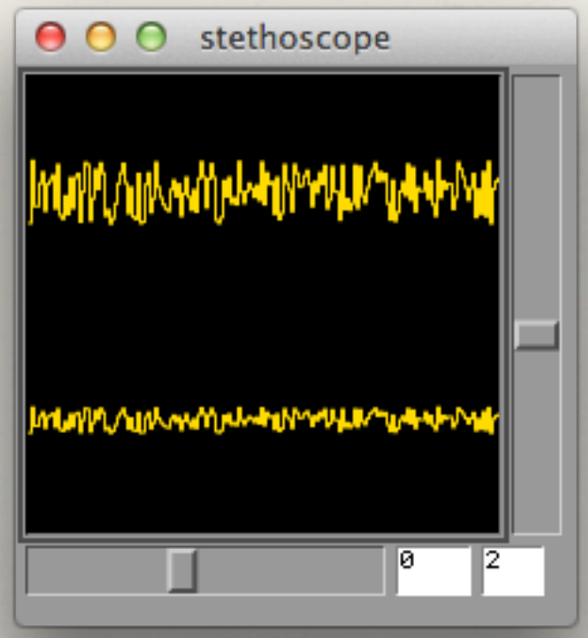


1.b Oszilloskop

```
{
  a = WhiteNoise.ar(0.3);
  Pan2.ar(a, MouseX.kr(-1,1))
}.scope
```

Linker Kanal

Rechter Kanal

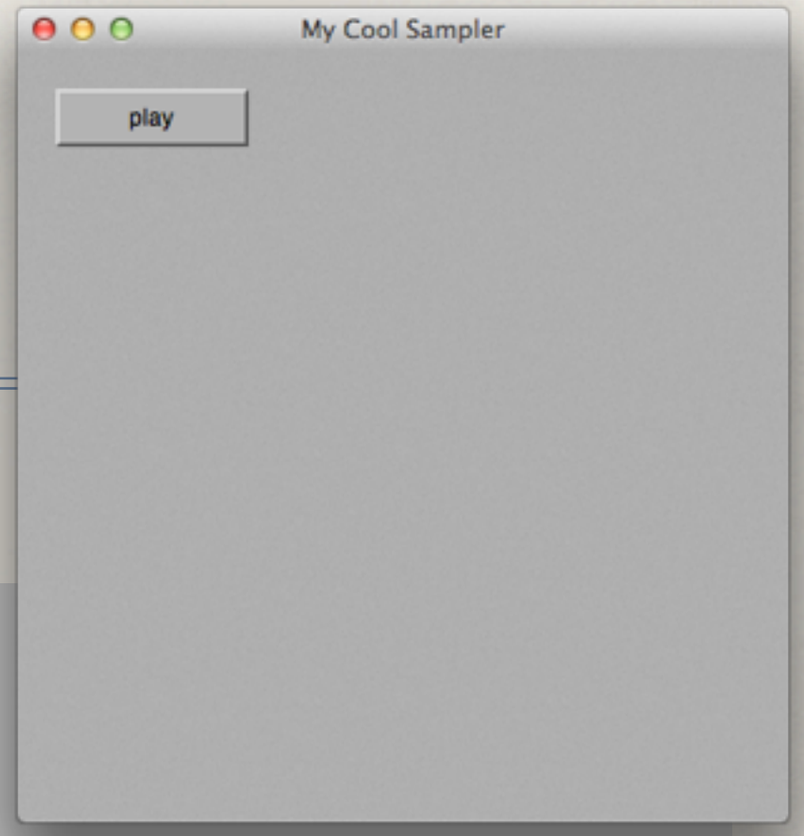


2. Fenster


```
(
  w = Window.new("My Cool Window");
  w.front;
)
```

```
(
  w = Window.new("My Cool Window", Rect(300, 200, 200, 300));
  // Position X,Y, Höhe und Breite
  w.front;
)
```

3. Knopf



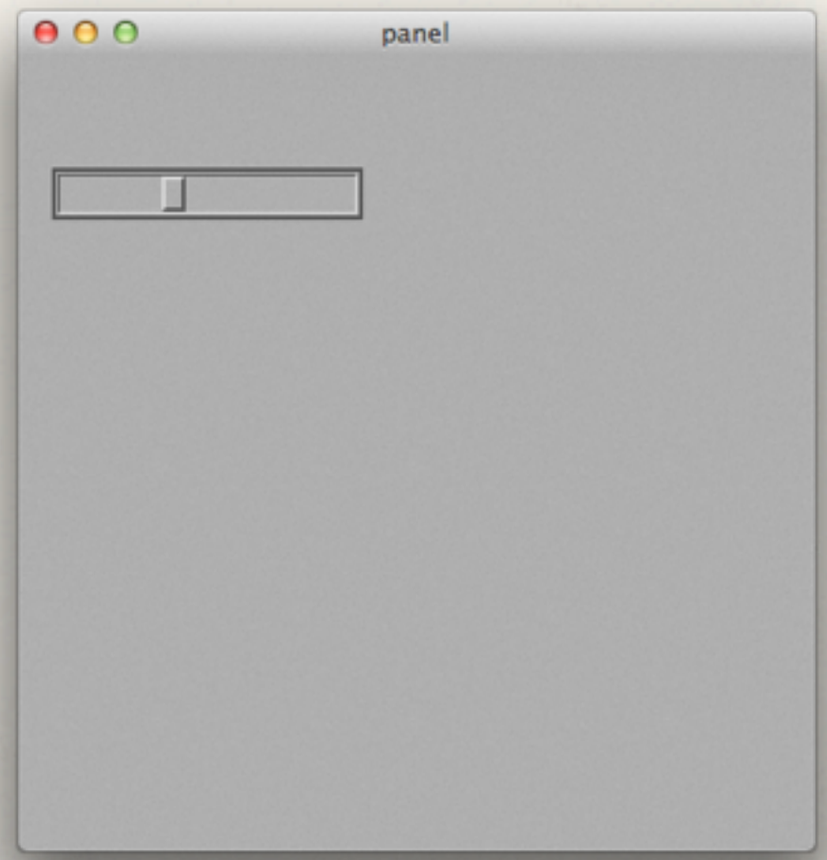
Sampler mit Oberflächen

```
(
   a = Buffer.read(s,"sounds/a11wlk01.wav");

   SynthDef("mySound",{
      Out.ar(0, PlayBuf.ar(1,a));
   }).load(s);
)
```

```
(
   w = Window.new("My Cool Sampler");
   b = Button.new(w, Rect(20, 20, 100, 30)); //x,y,Breite,Höhe
   b.states_([["play"]]);
   b.action_({
      Synth("mySound");
   });
   w.front;
)
```

4. Slider

```
(
   w = Window.new.front;
   a = Slider(w, Rect(20, 60, 150, 20));
   a.action = { a.value.postln };
)
```



4.b Slider

```
(
  SynthDef("MySynth", {
    arg freq;
    Out.ar(0,Saw.ar(freq),0.3);
  }).load(s);
)

(
  x = Synth("MySynth");
  w = Window.new.front;
  a = Slider(w, Rect(20, 60, 150, 20));
  a.action = {
    x.set("freq",a.value * 880 + 200);
  };
)
```

5. Mischpult

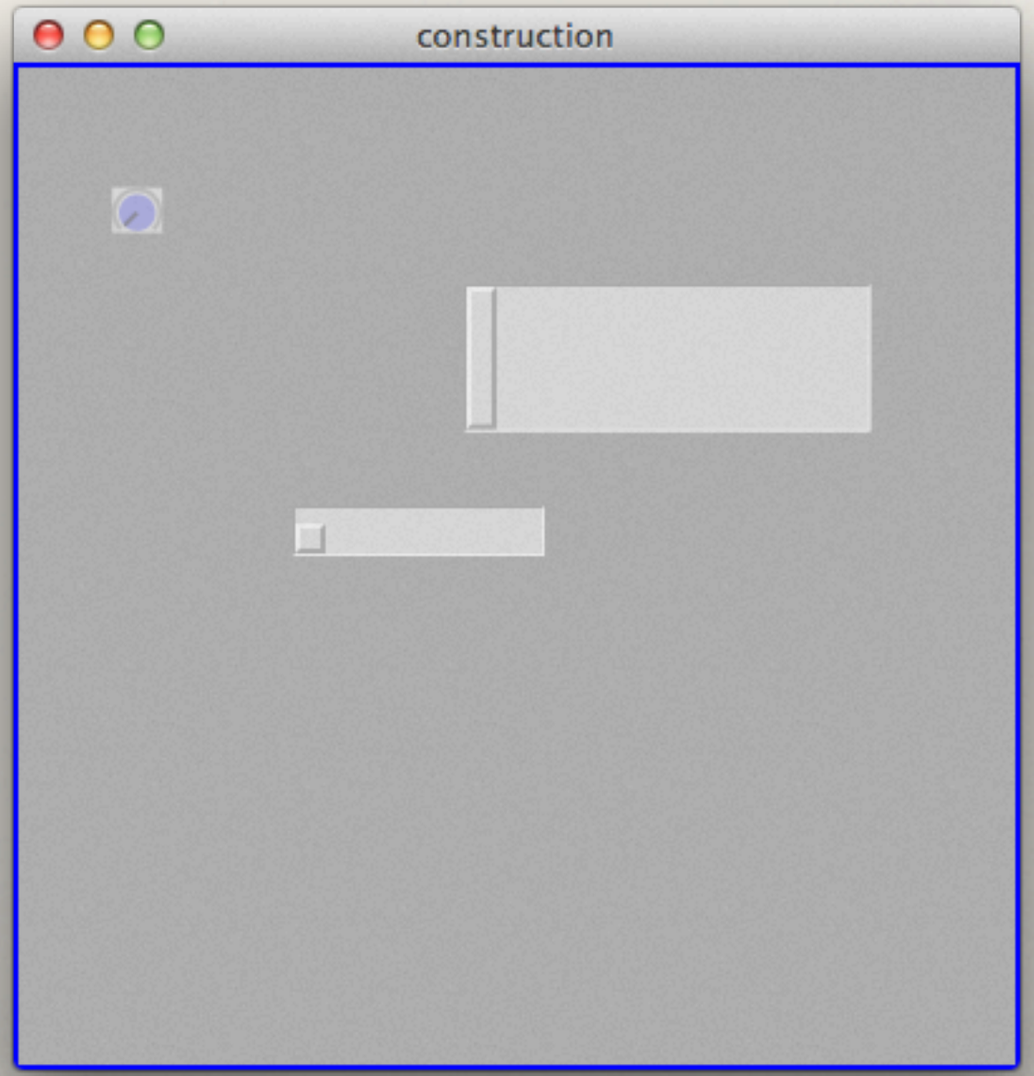

```
SynthDef("test",{
  arg noise_volume = 0, sine_volume = 0;
  x = WhiteNoise.ar(noise_volume);
  y =SinOsc.ar(440) * sine_volume;
  Out.ar(0,  x + y);
}).load(s);


(
  t = Synth("test");

  w = Window.new.front;
  a = Slider.new(w, Rect(20, 20, 150, 20));
  a.action = {t.set("noise_volume", a.value)};
  b = Slider.new(w, Rect(20, 60, 150, 20));
  b.action = {t.set("sine_volume", b.value)};
)
```

Zu kompliziert?

- UI -> New SCWindow

  Drag & Drop Knöpfe,
  Sliders, TextField usw.

- Drück “-> Code”



Was wir im Sommer studieren werden

Was wir im Sommer studieren werden

- 1. Spektrale Effekte in SC3
  - Spectral Freeze
  - Spectral Shift
  - Spectral Stretch
  - Spectral Noise Gate
  - Spectral Randomization
  - Cross Synthesis

Demo.1

Was wir im Sommer studieren werden

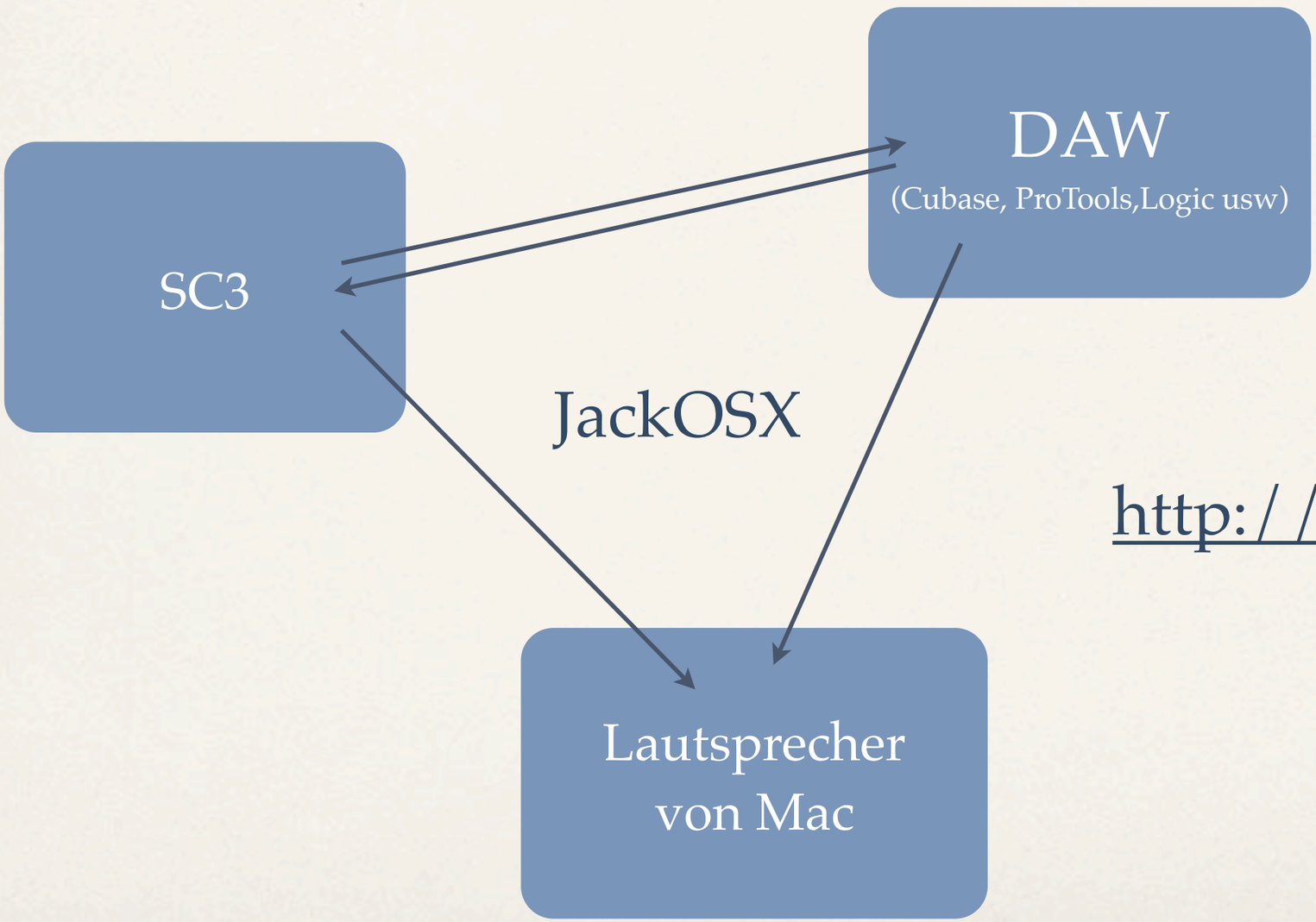
- 2. Stream, Pattern und Events

Diese Techniken sind im Prinzip für generative Musik aber wir können diese Techniken auf interaktive Musik anwenden.

Demo.2

Was wir im Sommer studieren werden

- 3. DAW und SC3

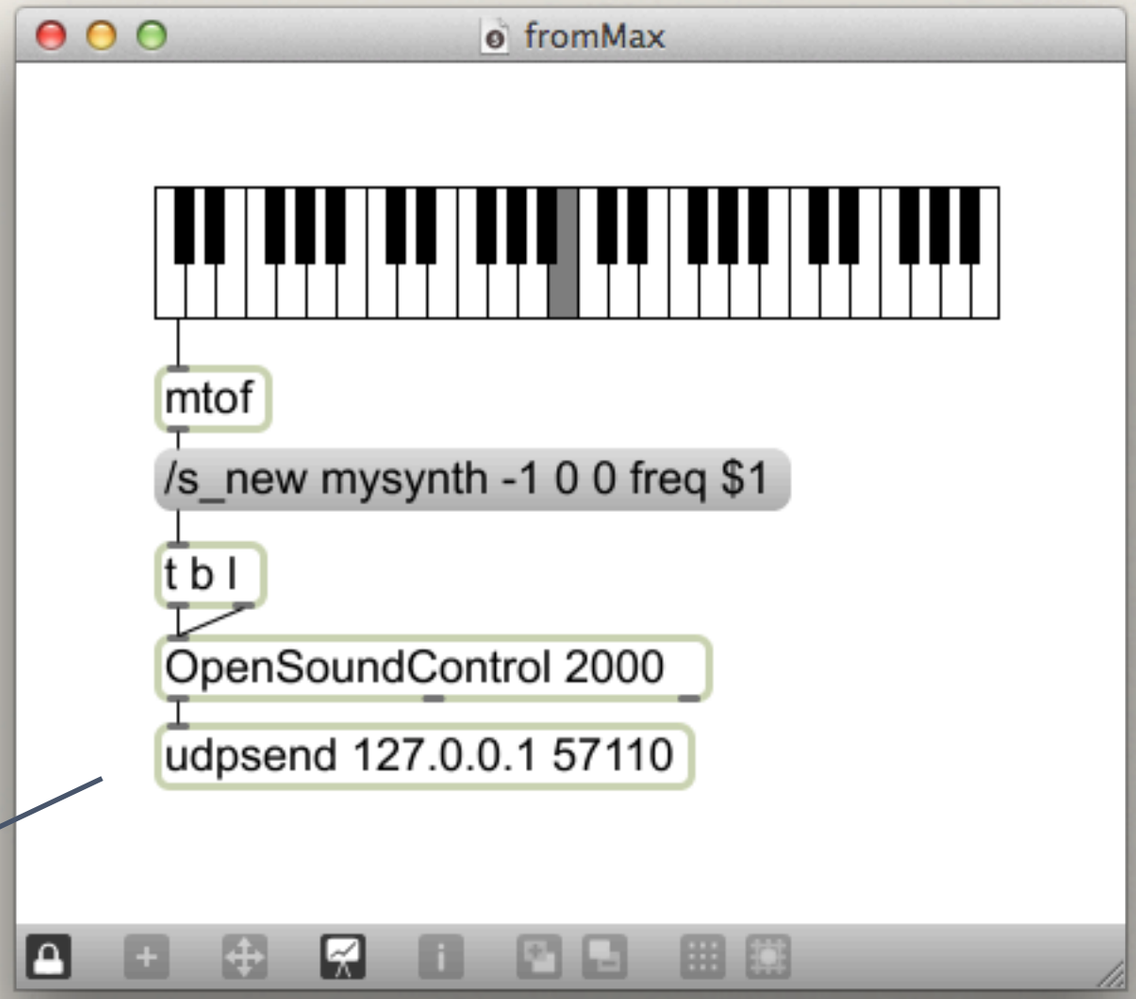


http://jackosx.com/

Was werden wir im Sommer studieren?

- 4. Max/MSP + SC3

```
SynthDef("MySynth",{
   arg freq = 220;
   a = Saw.ar(freq);
   Out.ar(0,a);
}).load(s);
```



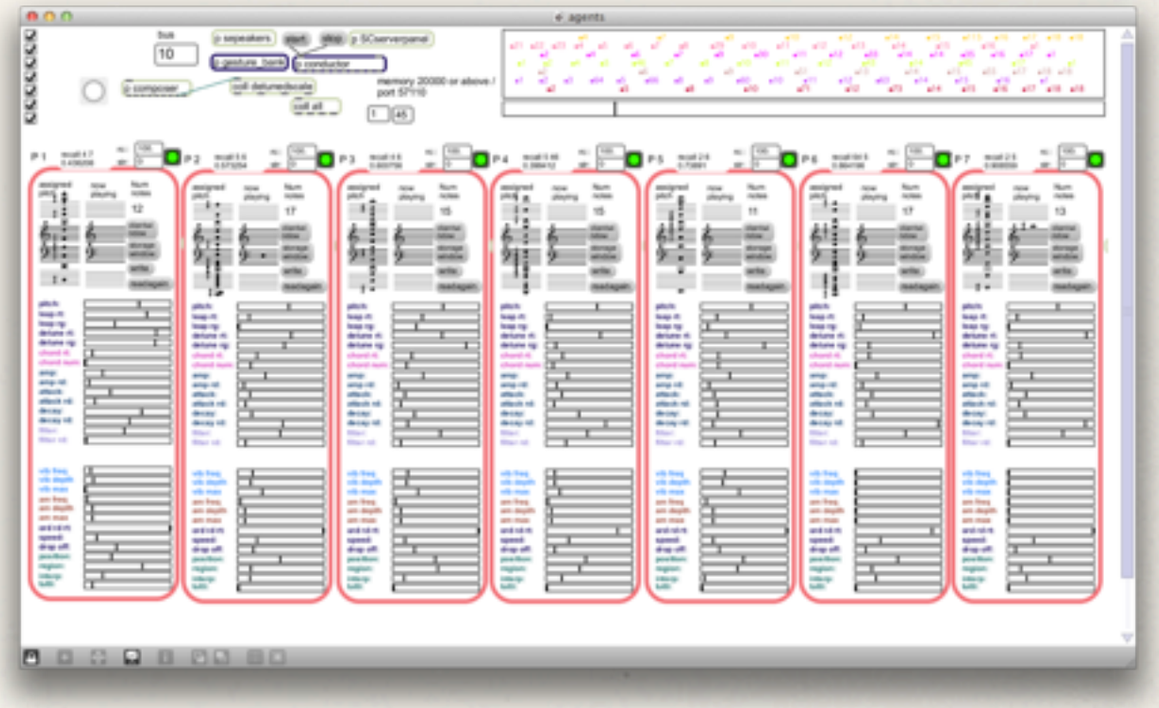
SC3 Server

Gleich wie

```
Synth("mysynth", ["freq", 800]);
```

Was wir im Sommer studieren werden

Thrum for fixed media (8 ch)

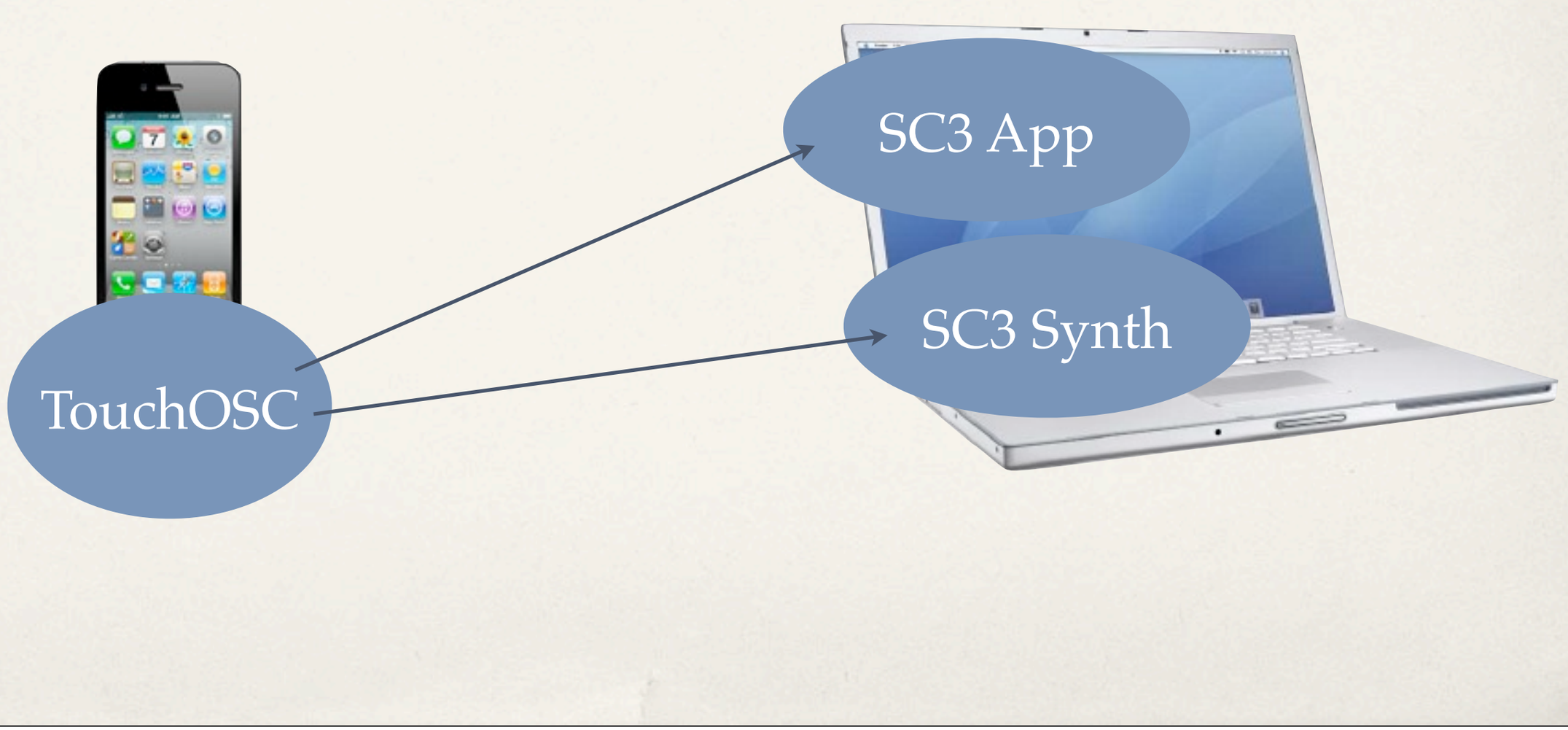


- 4. Max/MSP + SC3
- Algorithmische Komposition
- Karplus-Strong Synthesis

```
(
    {Pluck.ar(WhiteNoise.ar(0.1), Impulse.kr(2), 440.reciprocal, 440.reciprocal, 10,
        coef:MouseX.kr(-0.999, 0.999))
    }.play(s)
)
```

Was wir im Sommer studieren werden

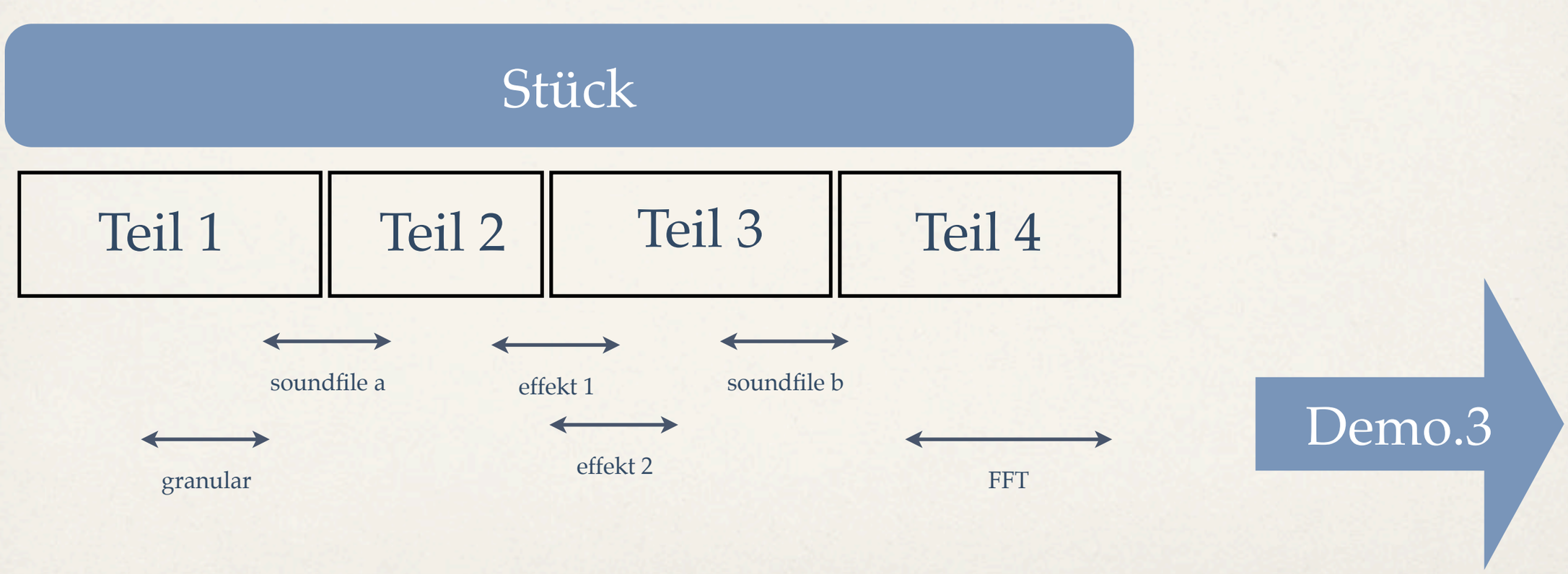
- 5. TouchOSC ... eine iPhone / iPodTouch App, die Daten von Beschleunigungssensor und Gyrosensor sammelt und als OSC Messages schickt.



Was wir im Sommer studieren werden

- 6. **Qlist** in SC3

Wie man SC3 auf live-elektronische Stücke anwendet.



Was wir im Sommer studieren werden

- 7. SC3 Plug-ins (Quarks)
  - Liste von Quarks
    - http://quarks.sourceforge.net/
- 8. Super Collider für iOS
  - damit kann man auf iPhone oder iPad Super Collider verwenden