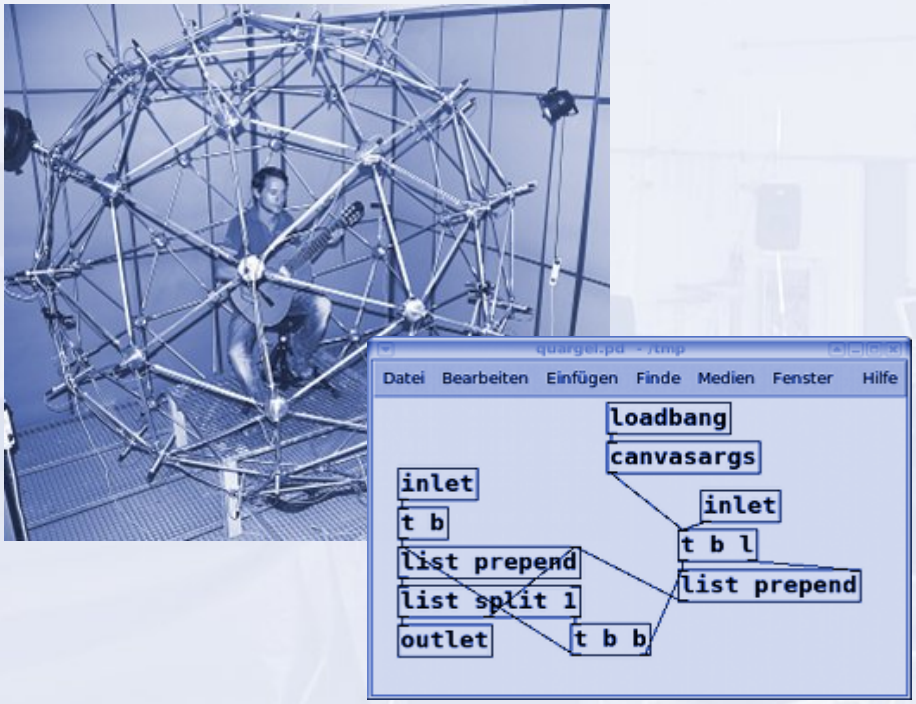
Computermusiksysteme VO

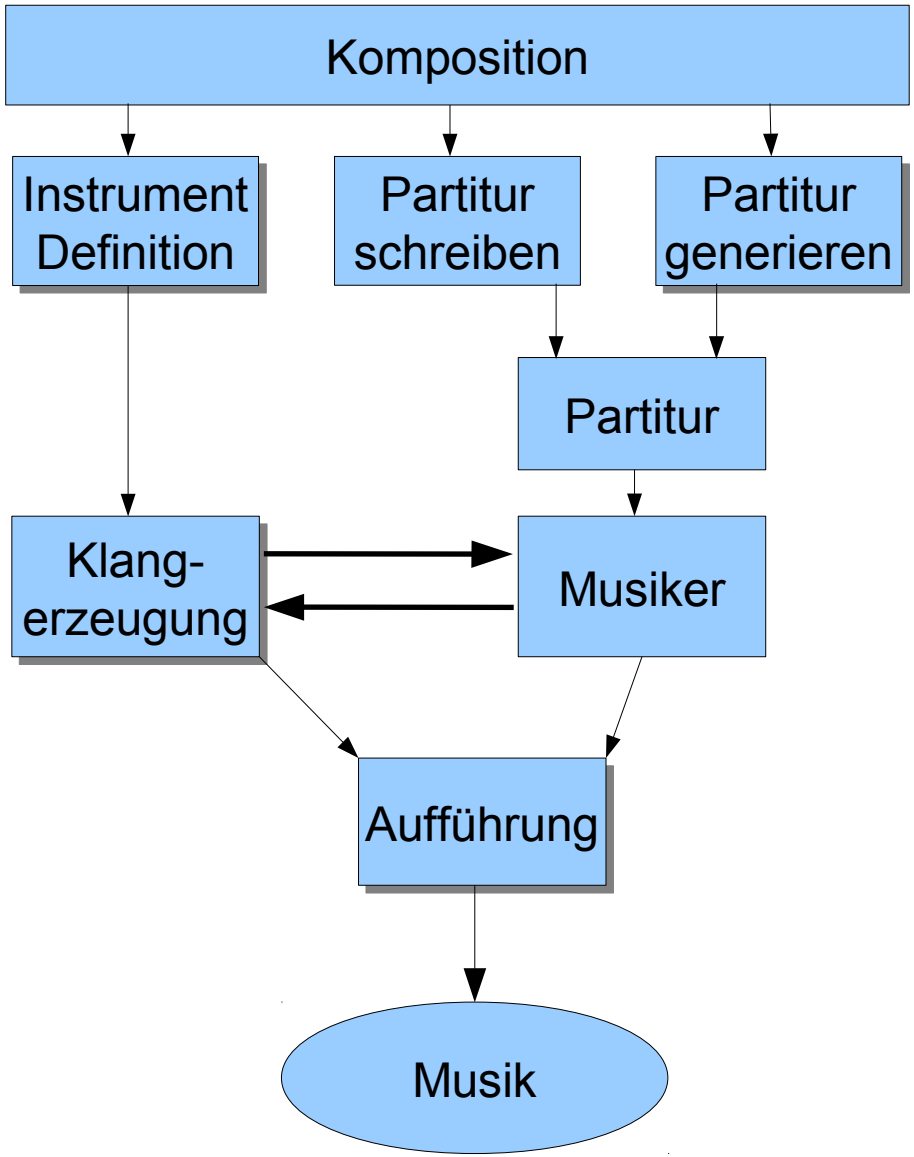


IOhannes m zmölnig

Grundlagen

- Einsatzgebiete von CMS
- Kompositionsprozess
- Repräsentation von Musik
  - Kontinuitätsmodell
  - Ereignismodell
  - Musik als Prozess
- historischer Überblick

Einsatzgebiete von Computermusiksystemen



historische Entwicklung

- Komposition
  - Illiac Suite (1956)
- Klangsynthese
  - MUSIC (1957)
- Steuerung
- Echtzeit
  - X4, ISPW (IRCAM, 80er Jahre)

Illiac Suite (1956)

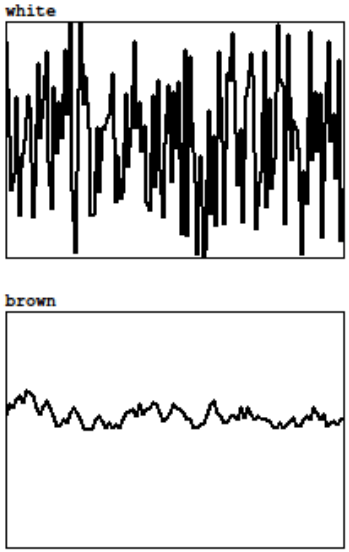
- Lejaren Hiller + Leonard Isaacson
- für Streichquartett
- 4 Sätze
  - cantus firmi
  - 4-stimmige Segmente
  - Rhythmus, Dynamik
  - Markoff-Ketten

Komposition am Computer

- Partitursynthese
  - Computer Assisted Composition
- Algorithmische Komposition
  - „Automatische Komposition“
  - Musikalisches Würfelspiel (18.Jhdt)
  - Muhammed Ibn Musa Al-Chorezmi (Bagdad, 9.Jhdt): „Regeln der Wiedereinsetzung und Reduktion“
  - Ein Algorithmus ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen

Algorithmische Komposition

- Deterministik
  - Bearbeitung großer Datenmengen
    - Wiederholungen, Variationen
- Aleatorik
  - stochastische Musik
  - Fokus: Zufallsprozess
  - Zufallsgeneratoren



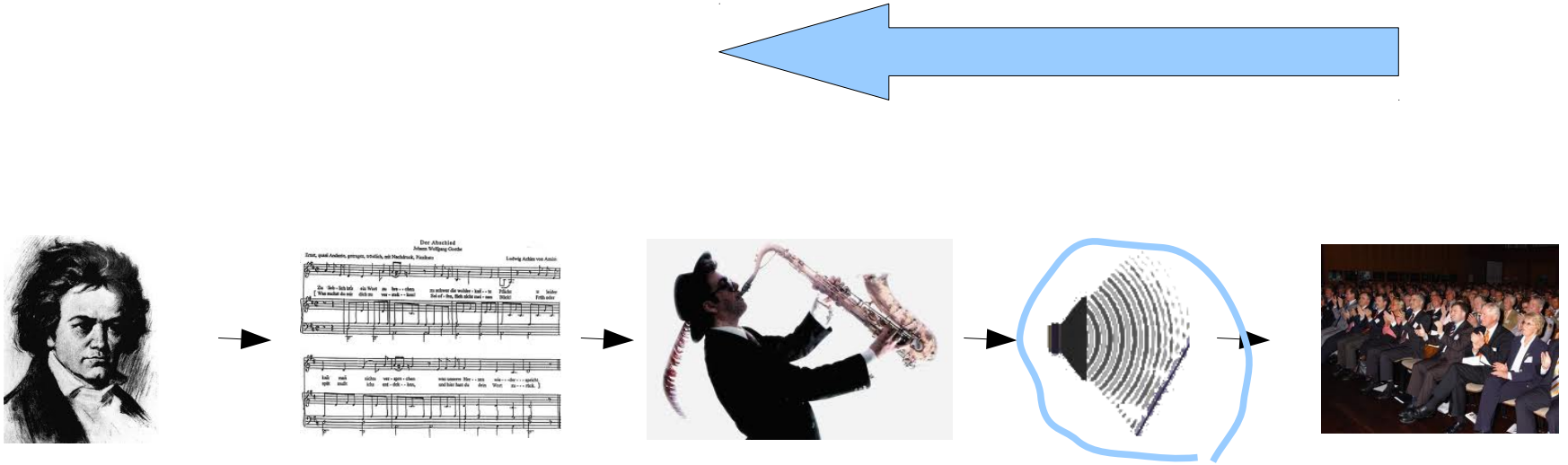
Werkzeuge in der Algorithmischen Komposition

- Markov-Ketten
- Neuronale Netzwerke
- State Machines,... Cellular Automata
- Fuzzy Logic

- → „Algorithmische Komposition VO“ (Nierhaus)

Musik am Computer

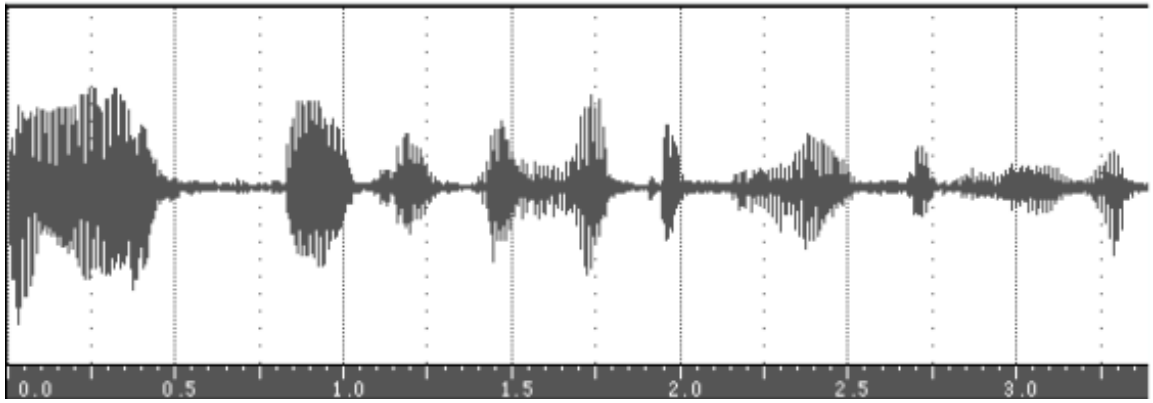
- Grundproblem: mathematische Repräsentation



Zeit-Y Diagramme

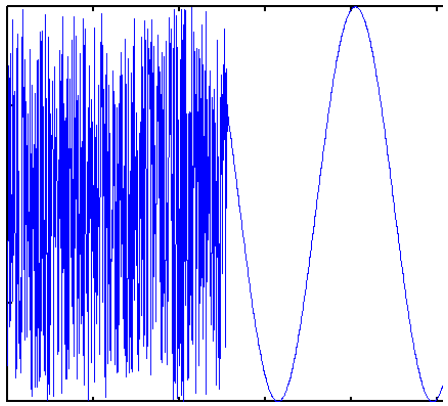
- Zeit-Amplituden Repräsentation
- Zeit-Frequenz Repräsentation

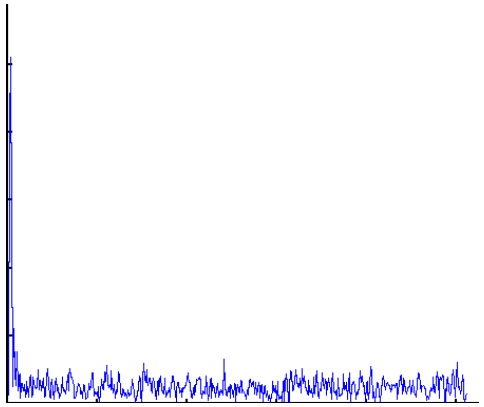
Zeit-Amplituden Diagramme

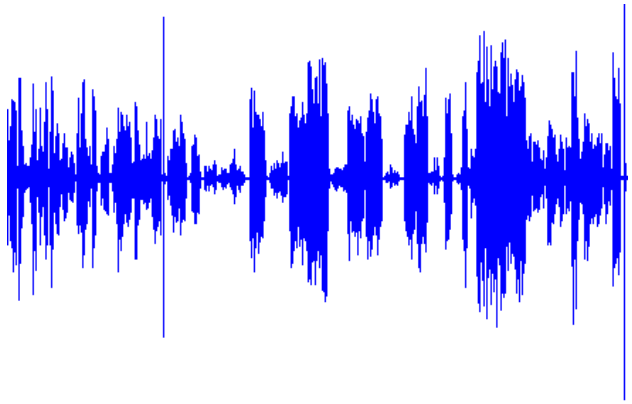


- Gesamte Musikalische Information in einer Dimension zusammengefasst
- Repräsentation
  - Zeitverhalten (Hüllkurven)
  - Orientierung ↔ Analyse

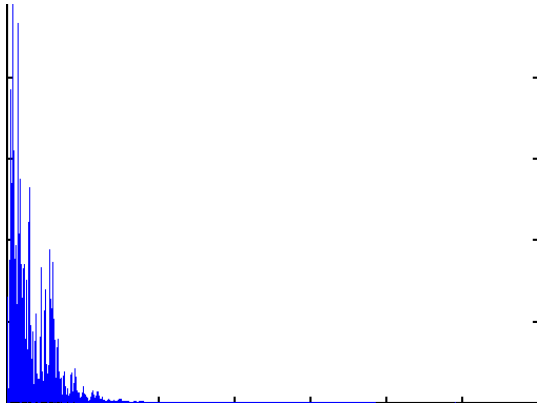
Frequenz Diagramme



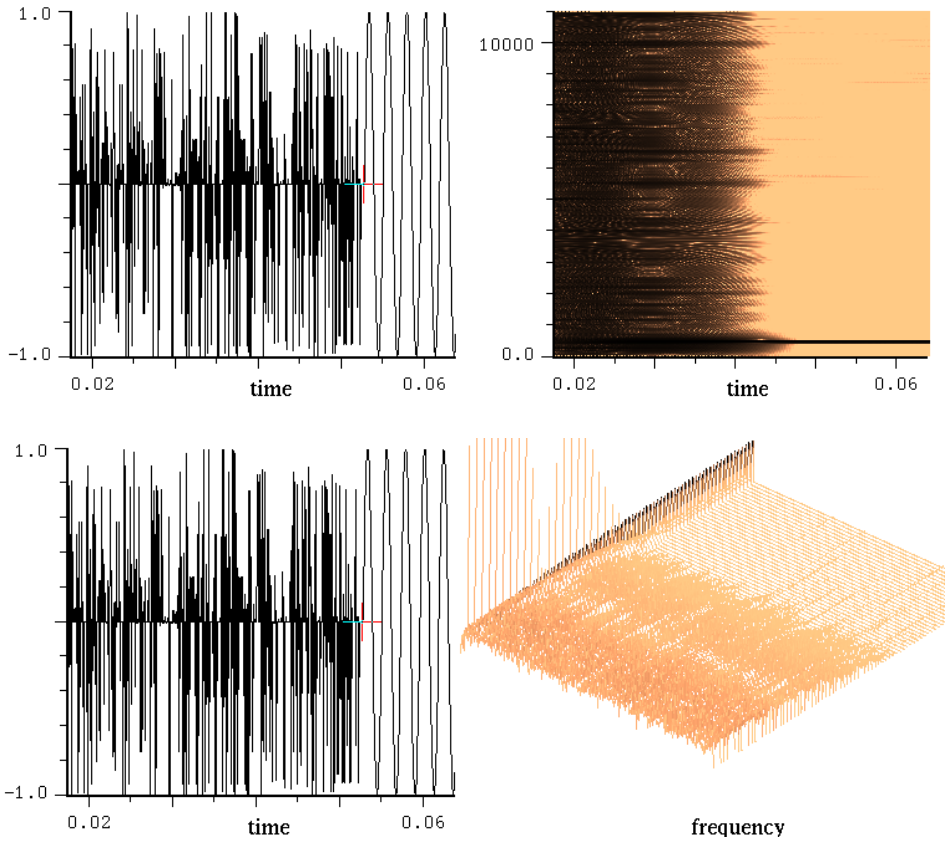




?

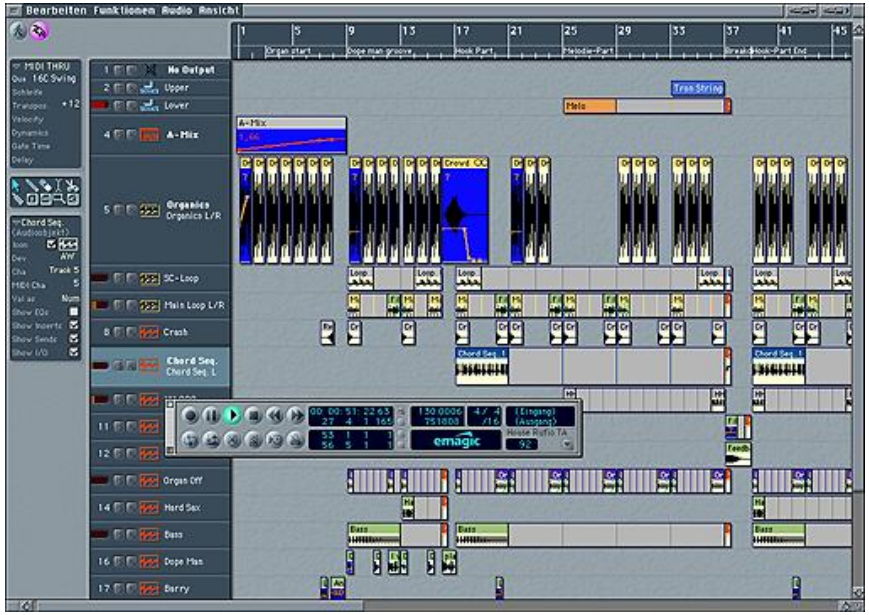


Zeit-Frequenz Diagramme

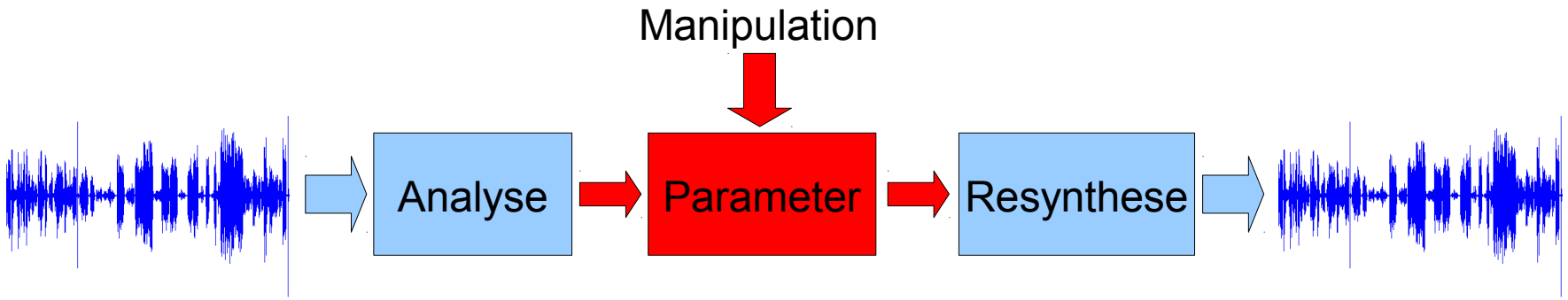


Organisation

- Organisation in perzeptive Einheiten
  - Stimmen / „Instrumente“
  - Musikalische Ereignisse / „Töne“
- Bearbeitung – Multitrack-Editor
  - entlang der Zeitachse
    - Schneiden
  - Track-basierte Bearbeitung
    - Filter/Effekte
    - Soundeditor
  - Vertikale-Bearbeitung
    - Mixing

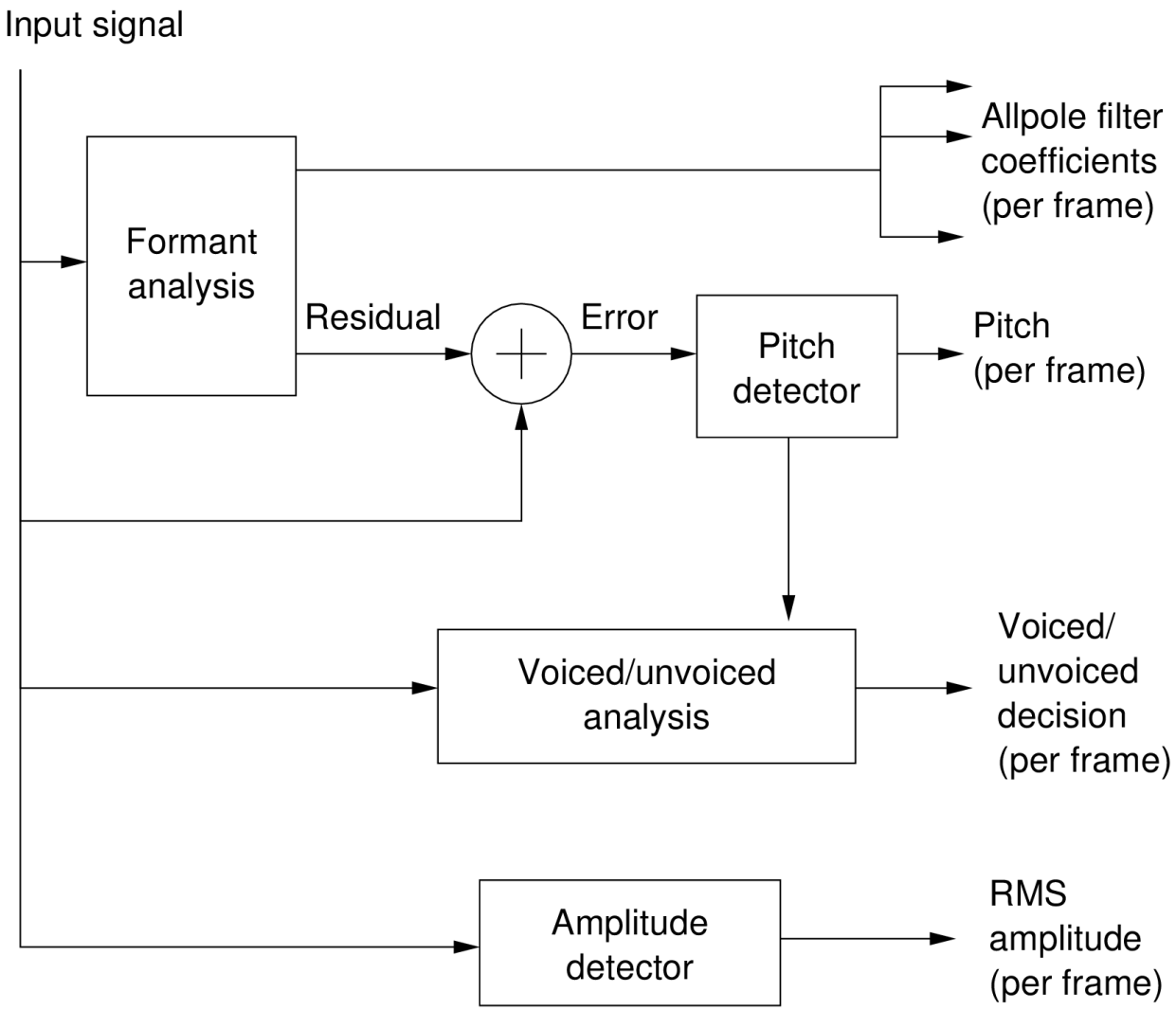


Parametrische Darstellung von Musiksignalen

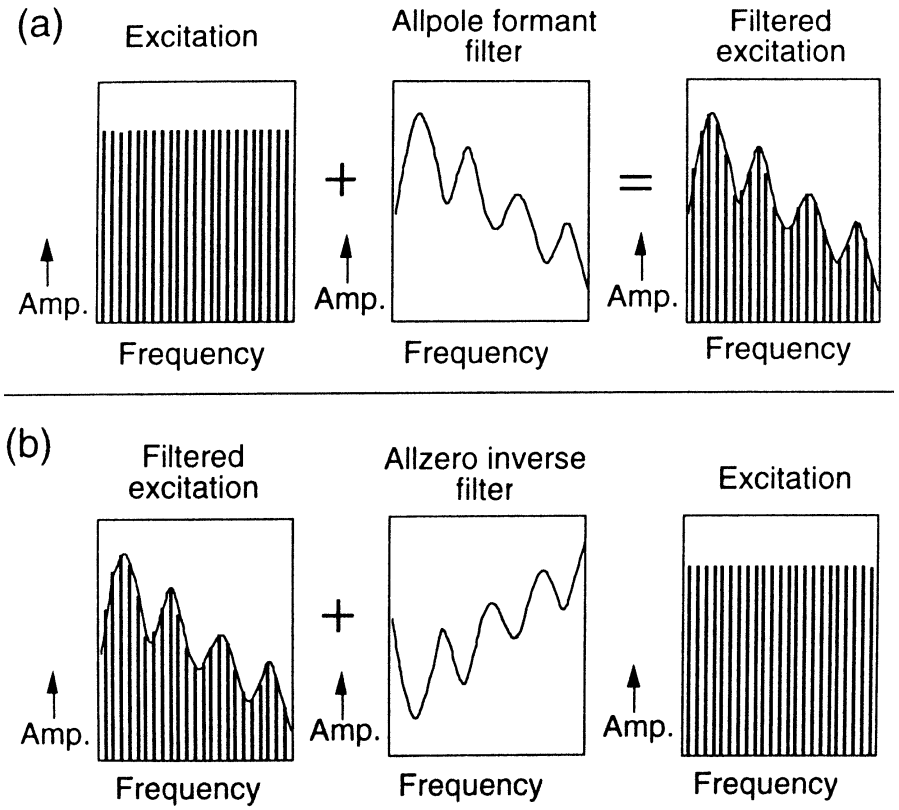


- Additive Synthese
  - Teiltonanalyse
- Linear Predictive Coding
  - Formantenanalyse
- ...

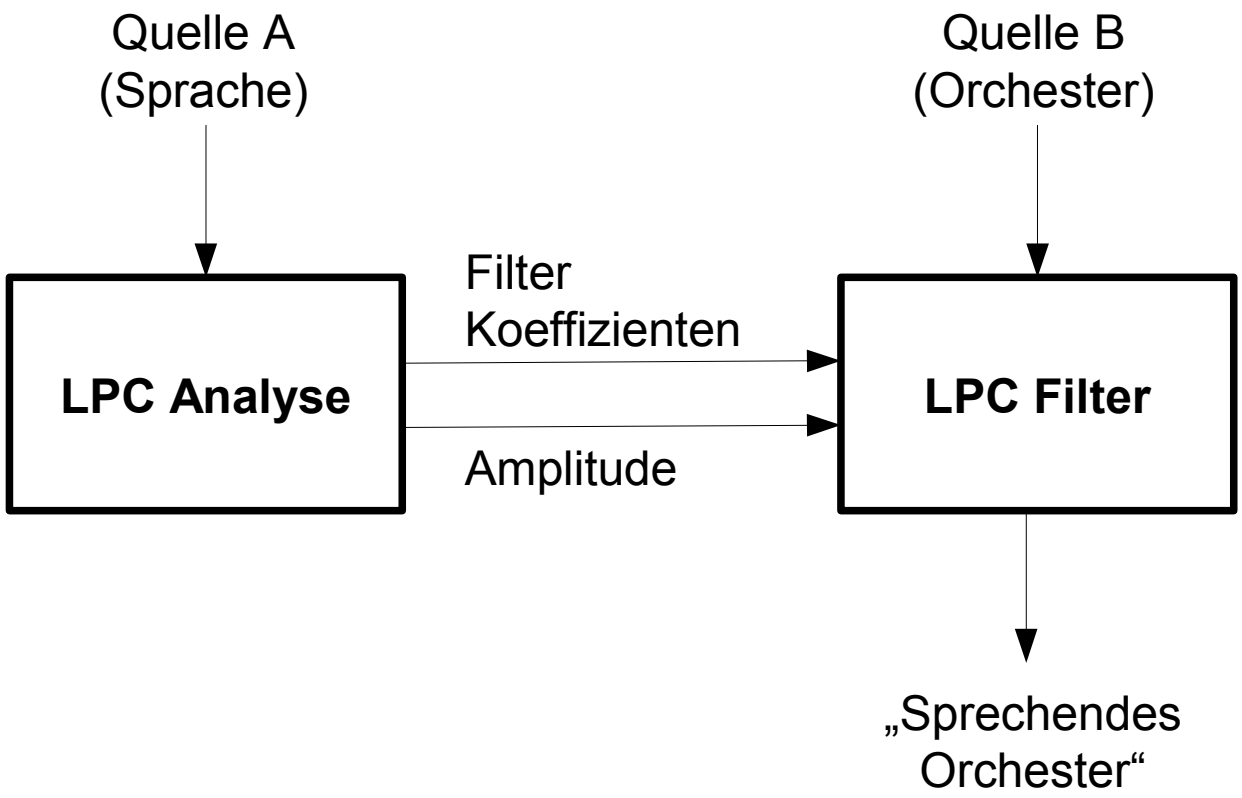
Linear Predictive Coding (Analyse)



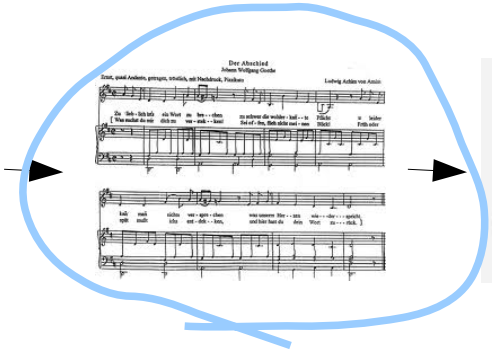
Linear Predictive Coding

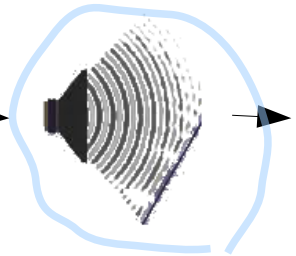


LPC Cross Synthesis

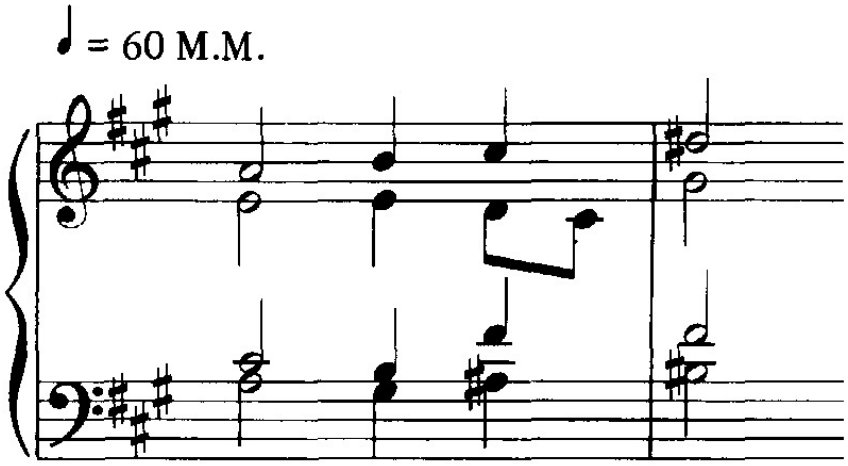


Musik am Computer





symbolische Darstellung



„Es ist genug“ – J.S. Bach

```
48,2  50,1  52,1    54,2          (soprano)

43,2  43,1  40,.5   40,.5  47,2   (alto)

40,2  38,1  45,1    45,2          (tennor)
36,2  35,1  37,1    39,2          (bass)
```

CMusic notation

```
struct {
     int pitch ;
     float dur ;
} chorale[4][6] = {
 48,2., 50,1., 52,.1, 54,2., -1,0., -1,0. ,
 43,2., 43,1., 40,.5, 40,.5, 47,2., -1,0. ,
 40,2., 38,1., 45,1., 45,2., -1,0., -1,0. ,
 36,2., 35,1., 37,1., 39,2., -1,0., -1,0.
};
```

C Code

Steuerdaten: Datenflussmodell

```
<Samplerate>:<Wert>,<Wert>,<Wert>
```

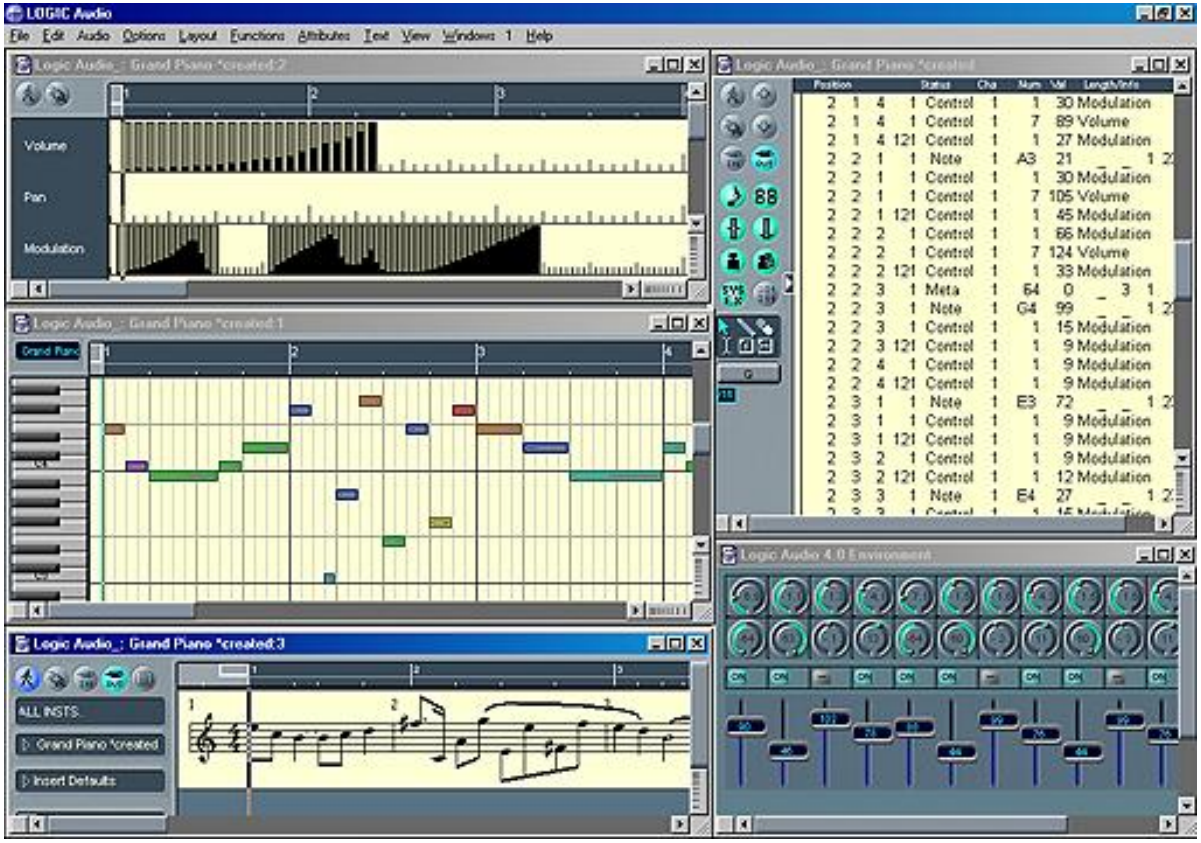
- Vorteile
  - Zeit implizit
  - Werte (zeitlich) geordnet
- Nachteile
  - große Datenmengen (auch bei wenigen Änderungen)

Steuerdaten: Ereignismodell

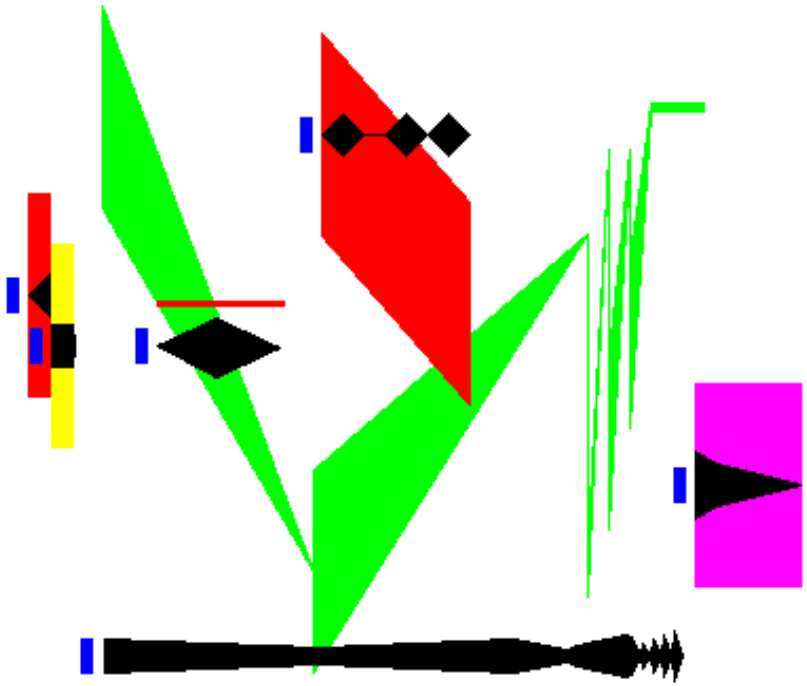
```
<Zeit><Wert>,<Zeit><Wert>,<Zeit><Wert>
```

- Vorteile
  - Datenmenge proportional zu Information
  - komplexe Strukturen abbildbar
- Nachteile
  - komplexe Interpretation
  - sehr große Datenmengen bei viel Information
    - großer Overhead

Ereignismodell: MIDI-Sequencer



Ereignismodell: mehrdimensionale Parameter

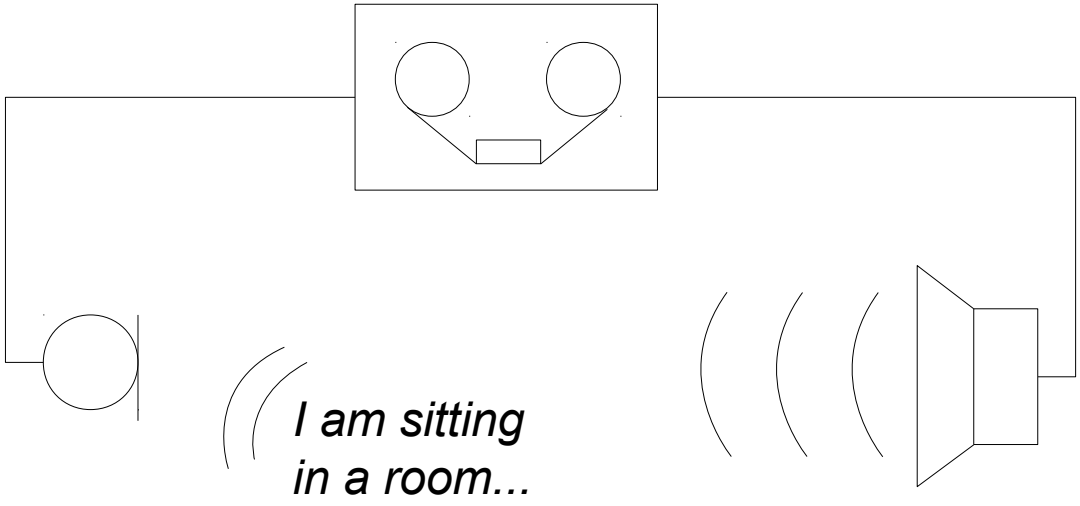


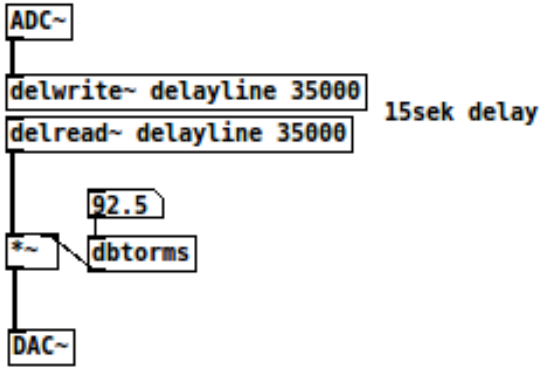
Prozessdenken

- „I am sitting in a room“ (A.Lucier 1969)

„I am sitting in a room different from the one you are in now. I am recording the sound of my speaking voice and I am going to play it back into the room again and again until the resonant frequencies of the room reinforce themselves so that any semblance of my speech, with perhaps the exception of rhythm, is destroyed. What you will hear, then, are the natural resonant frequencies of the room articulated by speech. I regard this activity not so much as a demonstration of a physical fact, but more as a way to smooth out any irregularities my speech might have.“

I am sitting in a room





Steuerungstechniken

- Analogsteuerung
- MIDI
  - Hardware
  - Protokoll
  - Files

Analogsteuerung

- Hybride Systeme
  - GROOVE (Mathews & Moore 1970)
  - HYBRID (Kobrin 1977)
- Digitale Kontrolle
- Analoge Synthesizer

- Spannungssteuerung (VC)
  - Spannungsanpassung
    - hochohmiger Eingang
    - niederohmiger Ausgang

Analogsteuerungsnormen

- de-facto Standard: Moog

- 0..10V
  - deckt „gesamten“ Bereich eines Parameters ab
- Amplitudensteuerung
- Frequenzsteuerung
- Filter-Güte,...
  - „regelbarer Bereich“

Amplitudensteuerung

- Abdeckung des gesamten musikalisch-verwendbaren Hörbereichs
  - 0..10V → 0..100dB: 10 dB/V
- JND
  - 0.2 – 0.4dB (optimale Bedingungen)
  - 1dB praktisch ausreichend

$$\frac{1\ dB}{100\ dB} = 100\ Schritte$$

- D/A-Umsetzer
  - 8bit (256 Werte) ausreichend
    - auch noch bei ½ Bit Rauschen

Frequenzsteuerung

- Abdeckung des gesamten Hörbereichs
  - 1V/Oktave (Moog)
  - 0V := 20Hz

$$f = 20 \cdot 2^{U} [Hz]$$

$$U = \frac{\log(f/20)}{\log(2)} [V]$$

U [V]	f [Hz]
0	20
1	40
2	80
...	...
10	20.480

- JND
  - 1%
  - Δ1Hz @ 100Hz

Frequenzsteuerung (Auflösung)

$$\Delta f = 101[Hz] - 100[Hz]$$

$$\Delta U = \frac{\log(101/20) - \log(100/20)}{\log(2)} \approx 0.0144 \approx \frac{10}{700}[V]$$

- D/A-Umsetzer
  - 11bit ausreichend (700 Schritte)
  - Praxis: 12bit

MIDI

- Musical Instrument Digital Interface
- Amerikanisch-japanisches Hersteller-Konsortium (1983)
- Problemstellung
  - viele digitale/hybride Synthesizer am Markt
  - Kompatibilität!
- Lösung
  - einfaches, hardwarenahes Protokoll
- Zielpublikum
  - (westlicher) Massenmarkt

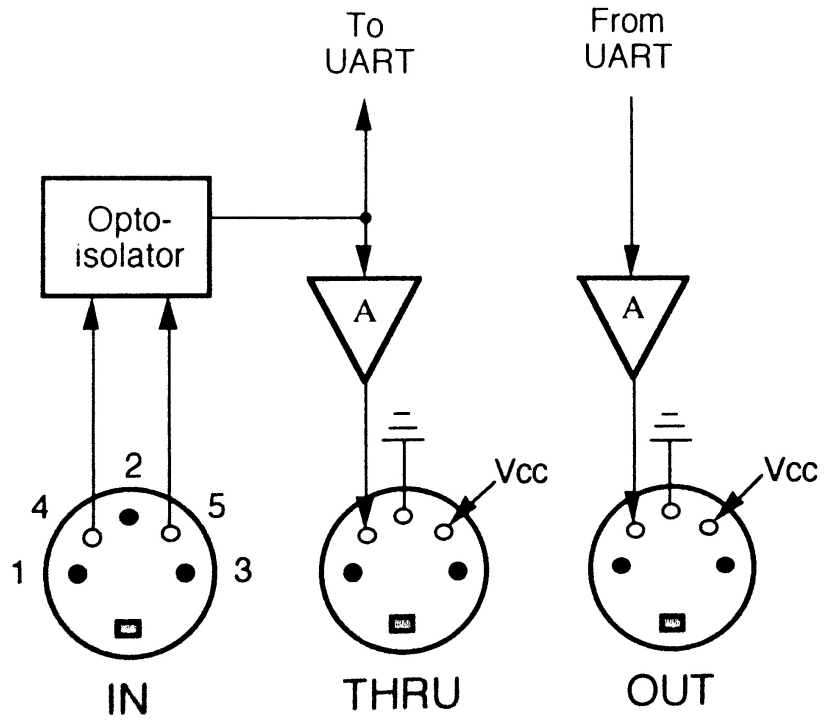
Möglichkeiten durch MIDI

- Einsatzgebiete
  - Live-Aufführung
  - später: Studiobereich

- Trennung von Controller (Keyboard) und Klanggenerator (Synth)
  - 1 Controller → viele Synths
  - vielfältige Controller
- digitale Repräsentation
  - Software-Controller
  - generische Software

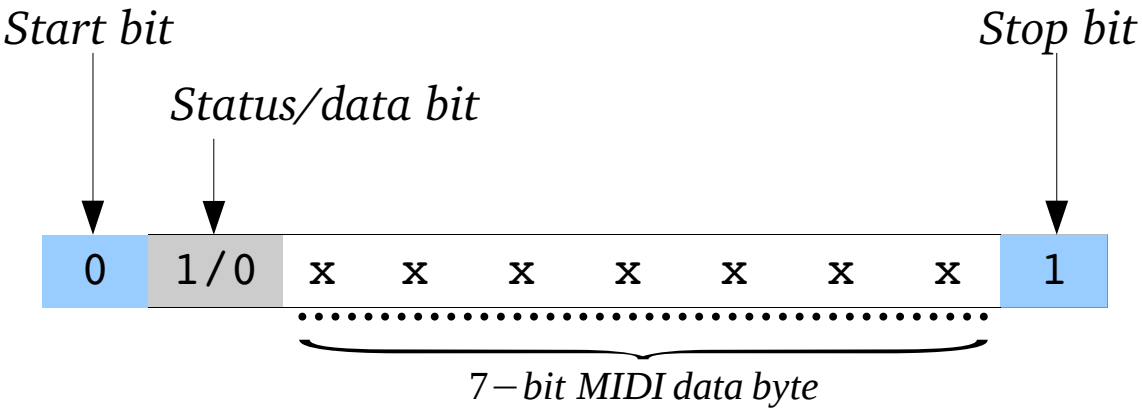
MIDI hardware

- Stromschleife (5mA)
- galvanische Trennung
- Übertragungsstrecken
  - 2-poliges geschirmtes Kabel
  - 5-poliger DIN Stecker
- IN/THRU/OUT

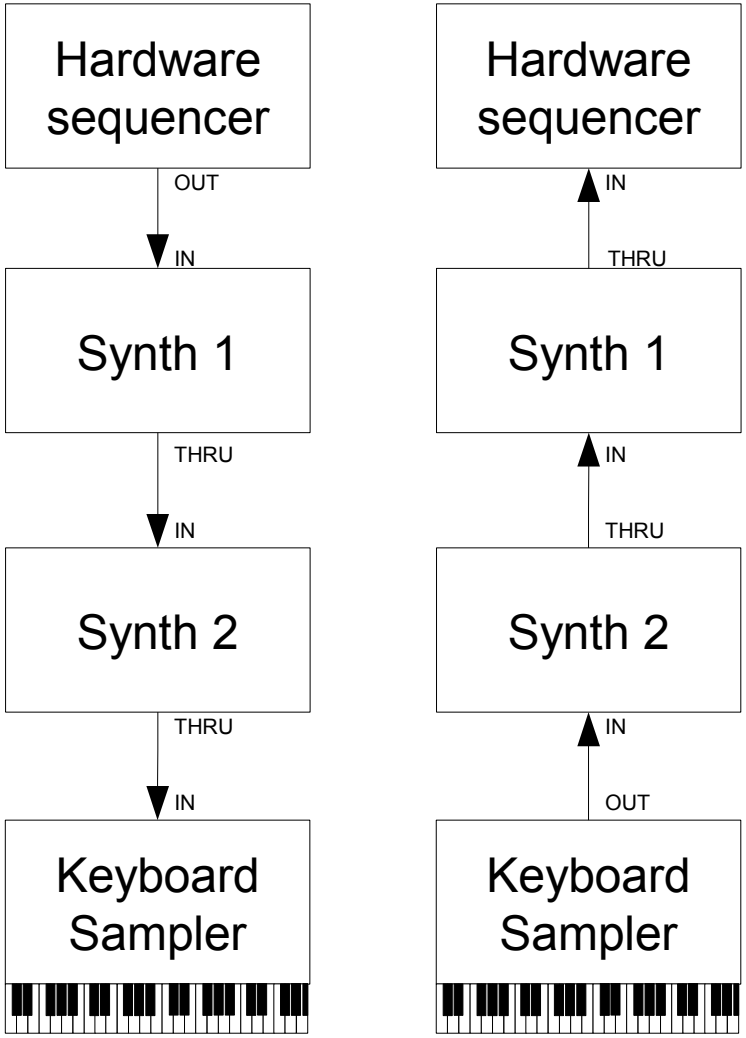


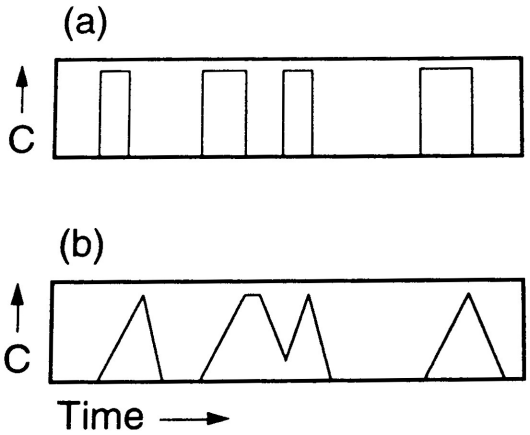
Serielle Schnittstelle

- UART
  - (Universal Asynchronous Receiver/Transmitter)
- 31250 Baud
  - 1MHz/32
  - Toleranz ±1%
- Inverse Logik (hi=0, lo=1)
- 10bit Frames



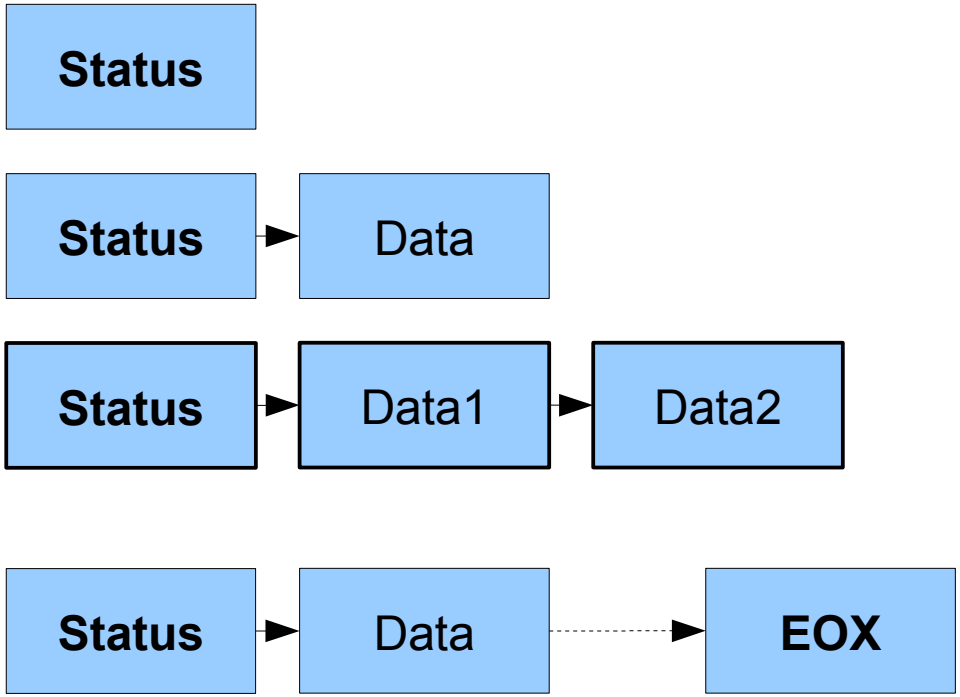
MIDI THRU: Daisy chaining





MIDI messages

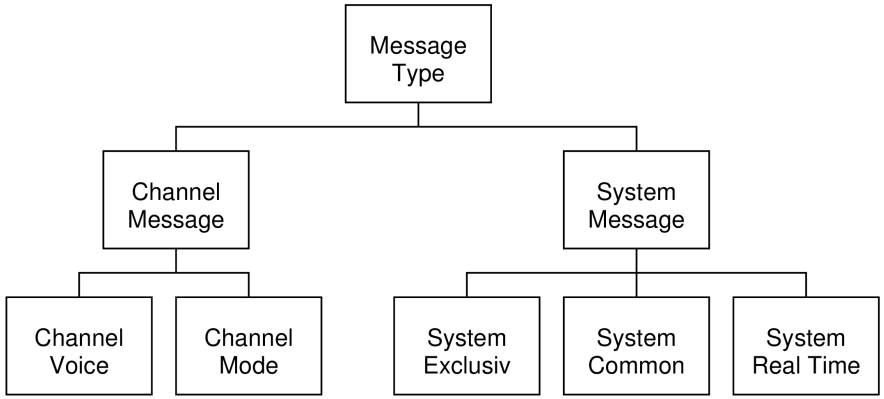
- Status Byte
  - Message Typ und Adresse
- Data Byte
  - Payload für Befehl
  - evtl. Multibyte



- Zeit
  - implizit codiert („jetzt“)

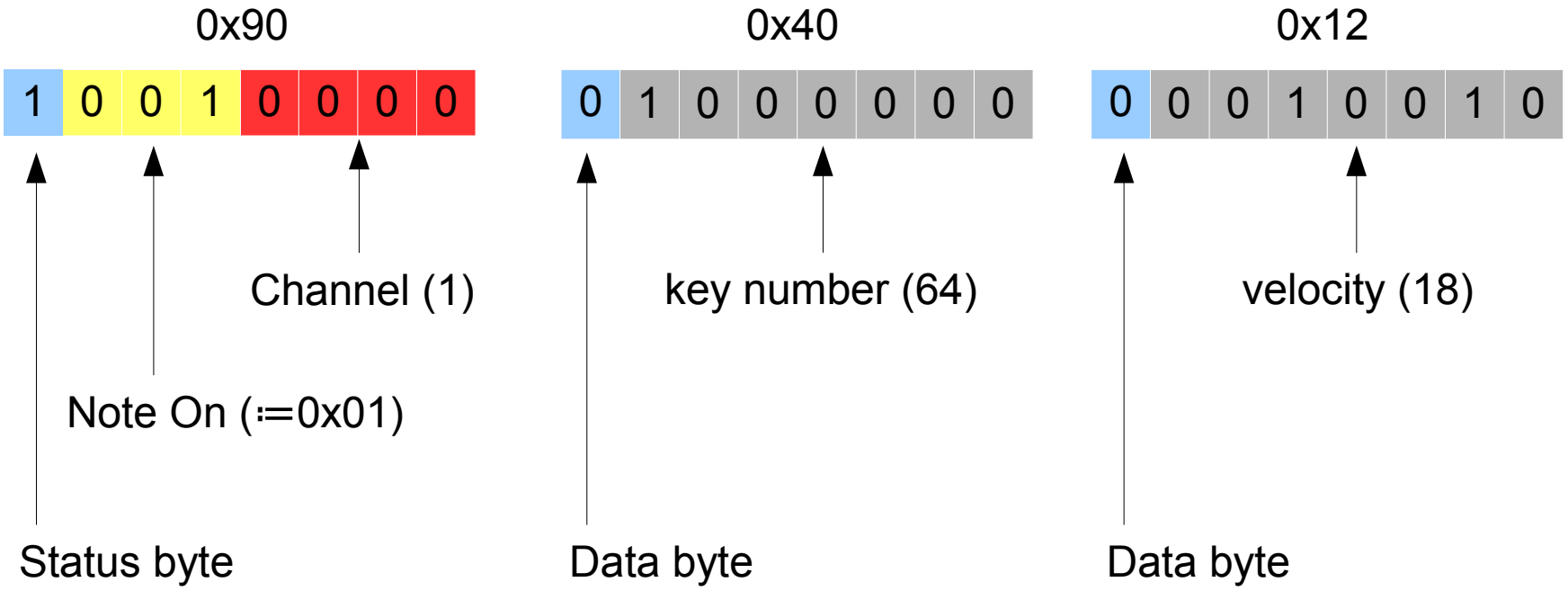
Typen von MIDI Messages

- Channel Voice
  - Note On/Off
- Channel Mode
  - All Notes Off
- System Common
  - Tune Request
- System Real Time
  - Active Sensing
- System Exclusive



Channel Message: Note On/Off

- 3 byte Message



- Note Off
  - 0x80
  - alternativ: *Note On* mit velocity≔0

Channel Messages: Control Change

- kontinuierliche Parameter per Channel
- Syntax: **0xB**<*n*> <*controlnum*> <*value*>
- Gruppen:
  - 0... 31: MSB
  - 32... 63: LSB für 0..31
  - 64... 95: Single Byte Controller
  - 96, 97: Increment/Decrement Controller
  - ...
- Auflösung
  - mit MSB+LSB: 14bit
  - Kontinuitätsprobleme!

Running Status

- Übertragungsgeschwindigkeit
  - ca. 1ms / NoteOn (3 bytes)
  - evtl. Timing-Probleme
- Strategie: Redundanzreduktion
  - Status Byte muss nicht permanent übertragen werden
  - → Datenreduktion um ca. 1/3

Δt	Status	key	vel	
0	0x90	0x34	0x35	NoteOn (channel=1), key=52, vel=53
120		0x37	0x2f	RS (NoteOn, ch=1), key=55, vel=47
5		0x34	0x00	RS (NoteOn, ch=1), key=52, vel=0 → NoteOff
100		0x37	0x00	RS (NoteOn, ch=1), key=55, vel=0 → NoteOff

Channel Voice Messages...

Musikalische Events (kanalspezifisch)

- Program Change
  - Mapping: Device-abhängig
- Aftertouch
- Pitch Bend
  - per channel!

Channel Mode Messages...

Kanal-Konfiguration

- All Notes Off
- Local/Remote Keyboard Control
- Mode Select

Mode Select

- Omni-on polyphonic („Omni“)
  - <Channel> wird ignoriert
  - Testen
- Omni-on mono
  - <Channel> wird ignoriert
  - monophonic Synthesizer
- **Omni-off „polyphonic“ („Poly“)**
  - Channel ↔ Instrument (polyphon)
- Omni-off („Multi“)
  - multi-timbre Synthesizer
  - Channel ↔ Timbre („Stimme“ einer Gruppe)

- General MIDI (1990)
  - Standardisierte „ähnliche“ Setups

System Messages

- System Common

  Allgemeine Einstellungen

  - Tune Request
  - Song Select / Song Position Pointer
  - MIDI Time Code

- System Real Time

  (kurze) Befehle zur Steuerung des Zeitverlaufs

  - Start/Stop/Continue
  - MIDI Clock
  - Active Sensing

Timing: MIDI Clock

- Synchronisation von Drum Machines,...
  - Relative Zeit
- Status: **0xF8**
- 24 Ticks pro Viertelnote
- Audio ↔ MIDI-Clock Konverter
  - Speicherung als Audio-Daten

- zusätzliche System Real Time messages
  - Start
  - Stop
  - Continue
  - Active Sensing

Timing: MIDI Time Code (1987)

- Synchronisation in Video/Film-Studios
  - Absolute Zeit
- SMPTE timecode `hh:mm:ss:ff`
- Übertragung in Quarter-Frames
  - 120 quarter-frames/sec
  - Status: **0xF1**
  - Data: 0nnndddd
    - n: Typ
    - d: Value-Nibble
- Zeitauflösung (1-2ms)
  - Zuspieltrigger

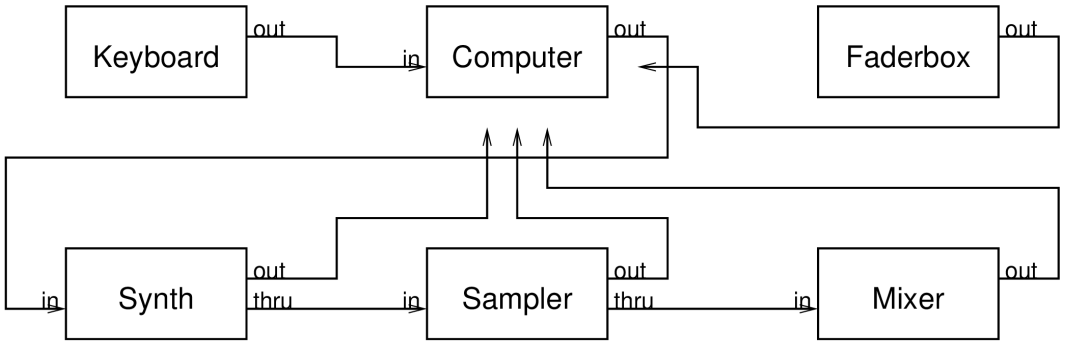
SysEx

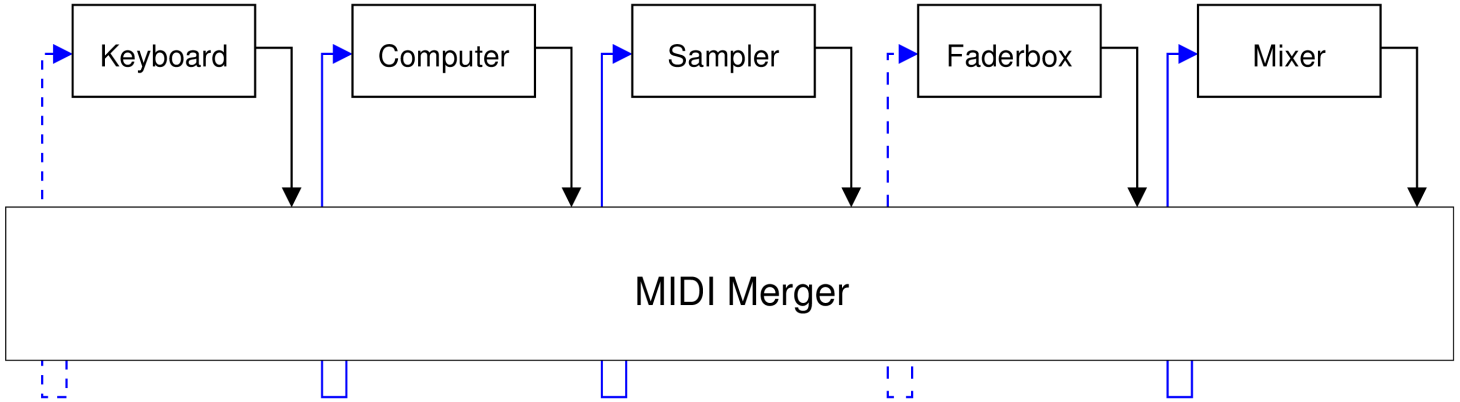
- Herstellerspezifische Messages
- 0xF0 <ID> …. 0xF7
  - Abbruch bei Status-Byte (vor 0xF7), ausser System Real Time messages
- <ID>
  - manufacturer ID
  - reserviert für zukünftige Protokoll-Erweiterungen:
    - 0x7E Non-Real Time
    - 0x7F Real Time

MIDI Einschränkungen

- Technische Einschränkungen
  - Routing
  - Bandbreite
- Repräsentations-Einschränkungen
  - sozio-kulturelle Annahmen über „Musik“
    - Metrisch
    - Tonal

Routing





- Bandbreite
  - Priorisieren von Messages (Note Off)
- Running State

Bandbreite

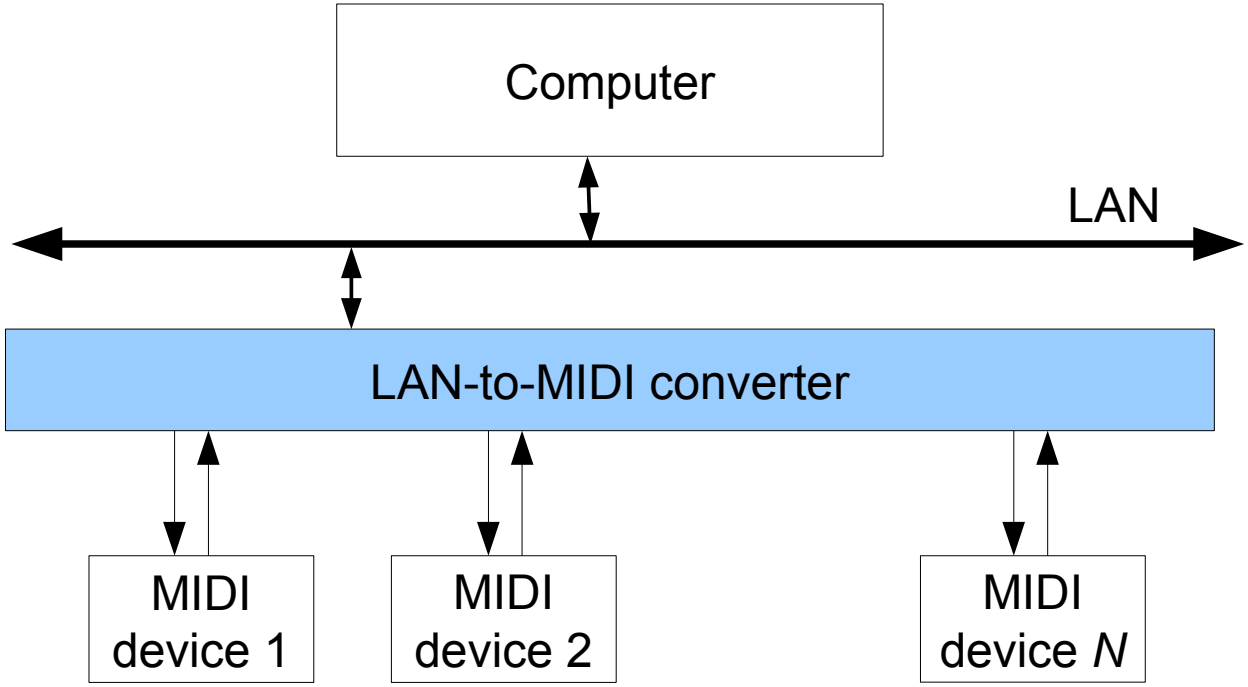
- 31.25 kBaud
- ca. 1ms / Message (3-byte)
- 16 channels Volllast
  → 50-150 Messages/channels/sek
- kontinuierliche Controller
  - permanent Messages solange Controller aktiv
- MIDI choke
  - Timing ungenau (Jitter)

Bandbreite: Multiplexing

- Aufteilen der Daten auf parallele Pfade
  - separate Kabel
- Vorteil
  - innerhalb der MIDI-Spezifikationen
  - Dezidierte Leitung für „wichtige“ Messages
- Nachteil
  - Hardwareaufwand
  - Devices mit mehreren MIDI ports

Bandbreite: Transportprotokoll

- (teilweise) Ersetzen des Physikalischen/Datalink Layer
- schnellere serielle Verbindungen
- Kapselung in Netzwerkprotokolle



Steuerungstechniken

- MIDI
  - Files
- Netzwerk
  - OSI-Model
  - FUDI
  - OSC

MIDI files

- Serielles Protokoll enthält Zeit implizit

- Standard MIDI File (SMF) - 1988
- Erweiterung um Timing-Information
  - Timestamps in „ticks“
  - 8-32bit integer
  - relative Länge, bezogen auf Tempo

MIDI File Types

- Type-0
  - single Track
  - Mehrere Channels im Track
- Type-1
  - Parallele Tracks
  - Timing Information für alle Tracks ident
  - Timing Information im Track#1 (TempoMap)
- Type-2
  - Sequentielle Tracks
  - Timing Information pro Track
  - „Drum machine Format“

MIDI File Structure

MThd <len=6> <fmt> <NumTracks> <ZeitTyp>

MTrk <len> <trackevent> <trackevent> … <trackevent>

<dtime> <event> | <dtime> <event> | ... | <dtime> <EOT-event>

MTrk <len> <trackevent> <trackevent> … <trackevent>

<dtime> <event> | <dtime> <event> | ... | <dtime> <EOT-event>

⋮

SMD Header

MThd <len[4]> <fmt[2]> <tracks[2]> <time[2]>

- len: immer 6
- time: Ticklänge in Pulses/Quarternote
  - Default: 120bpm
  - time[0]<0 →
    - -time[0] = SMPTE fps
    - time[1] = Ticklänge in Pulses/Frame

SMF MTrk events

- dtime (in Ticks)
  - bis zu 32bit
  - Enkodiert in 7bit Wörter (variable Anzahl)
  - MSB=0 signalisiert letztes Wort
- MIDI event
  - Status+Databytes
- SYSEX event
  - *0xF0 <len> <data>*
  - *0xF7 <len> <data>*
- META event
  - 0xFF <type>...
  - End-Of-Track: 0xFF 0x2F 0x00
  - Tempo: 0xFF 0x51 0x03 tt tt tt
    - in µsec/Schlag

Netzwerk-Kommunikation

- verfügbare Infrastruktur
  - Standardkomponenten
  - Reliabilität: Insellösung
- Beziehung
  - Client
  - Server

OSI-Netzwerkmodell

Beispiel

Daten

```
GET /
```

MIME

Verschlüsselung

```
afad...1530
```

TCP/IP

```
TCP | afad...1530
```

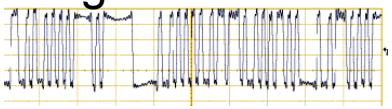
IP

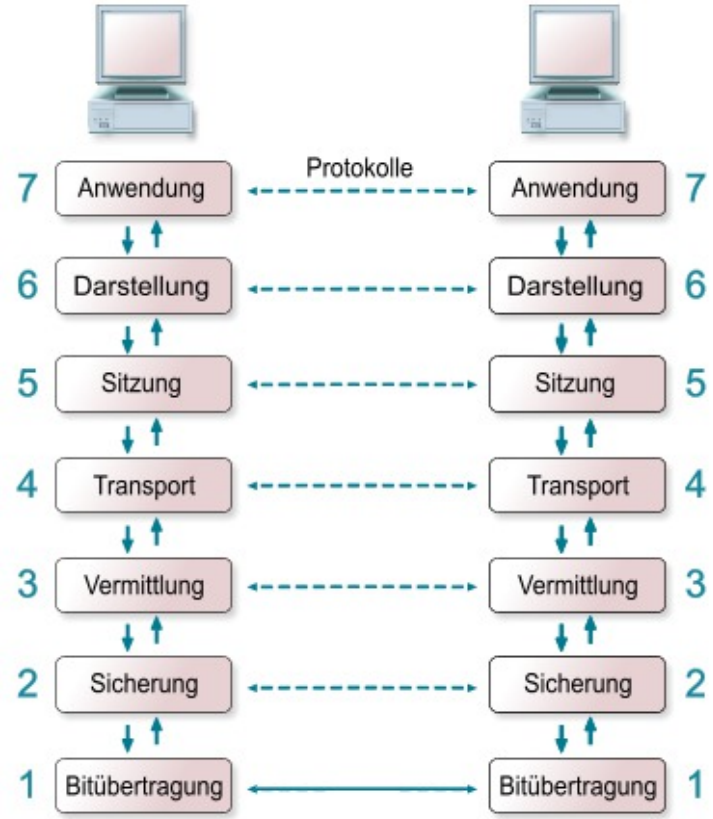
```
IP | TCP | afad...1530
```

ethernet

```
eth | IP | TCP | afad...1530
```

elektr.Signale





typische Implementierung auf Ebene:

„Applikation“

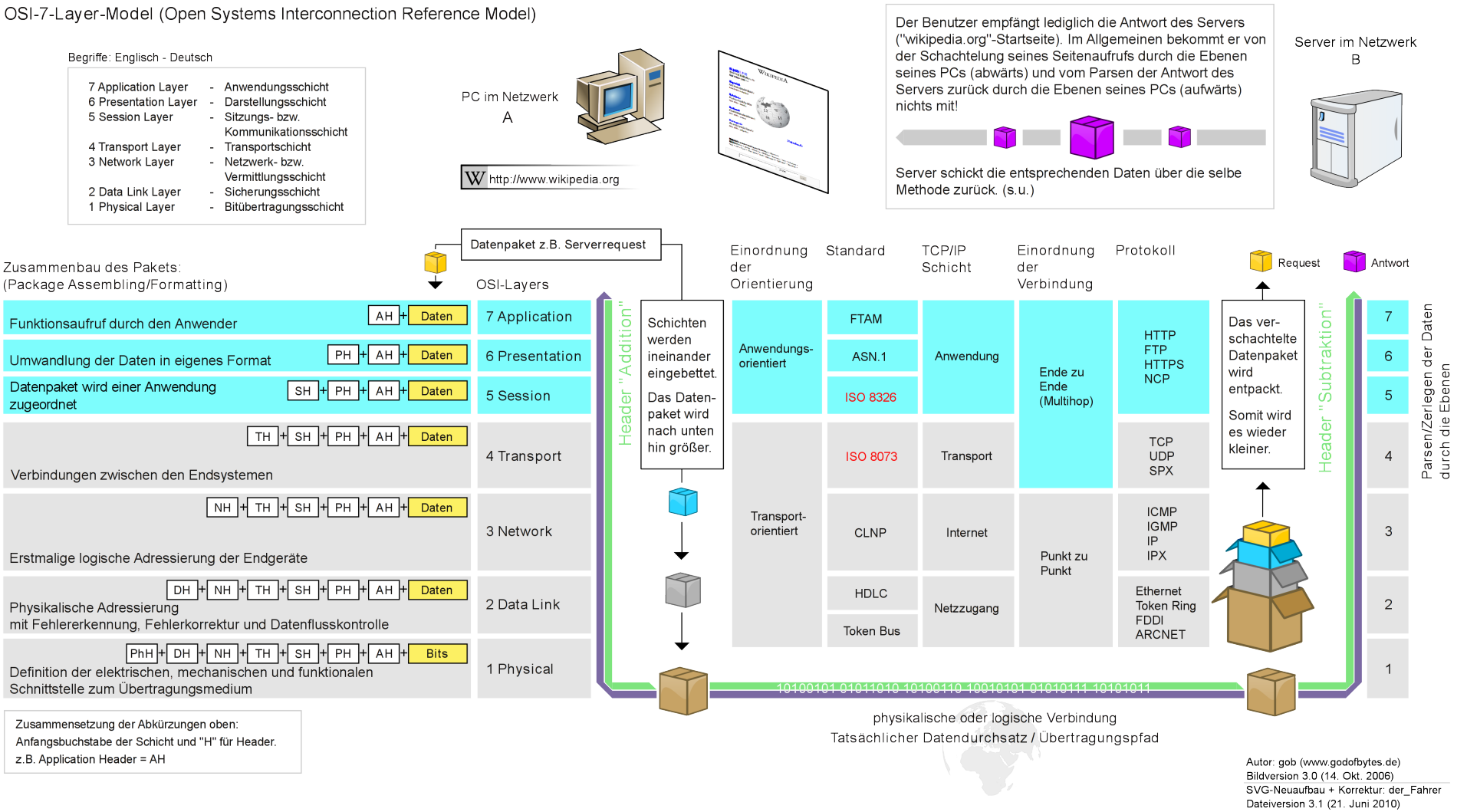
„Betriebssystem“

„Router“

„Hub“

„Kabel“

OSI-Netzwerkmodell

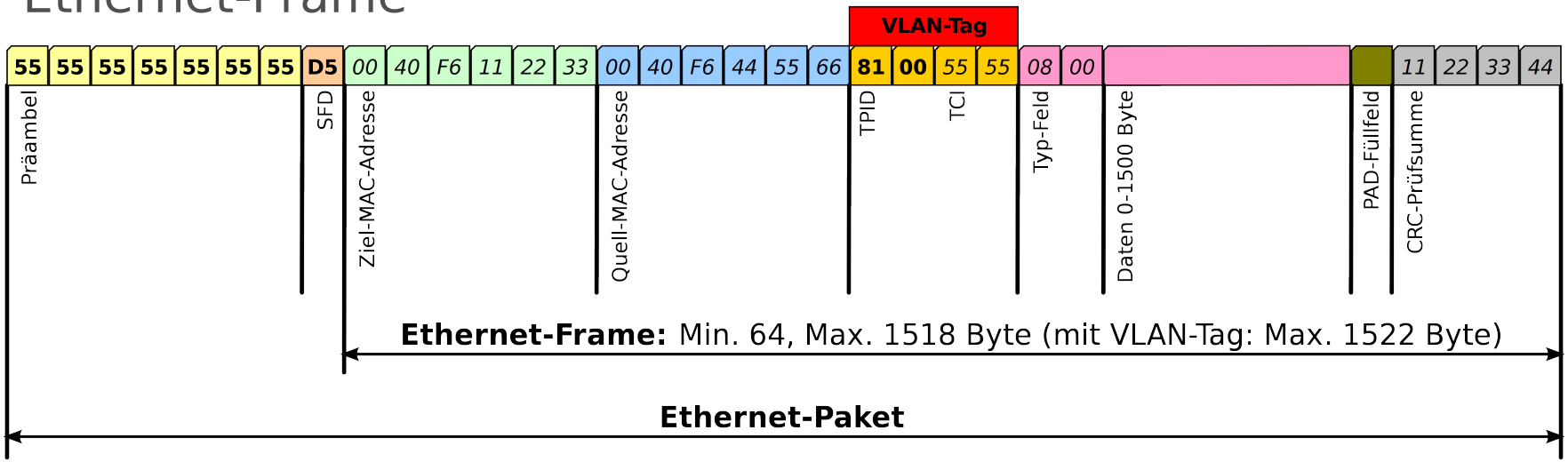


Data Link Layer: Ethernet

- Punkt ↔ Punkt Verbindung
  im Lokalen Netzwerk
  (kein Routing)
- MAC-Adressierung
  - 6-Byte „unique“ Adresse
  - Broadcast / Layer2-switches

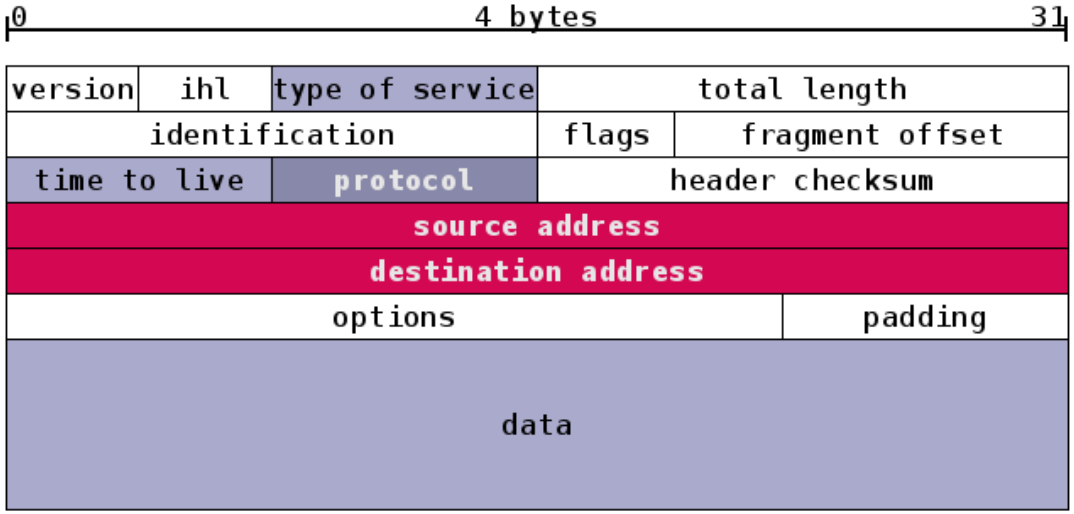
- datalink+physical layer
  - 10BASE-T
  - 100BASE-TX
  - 100BASE-FX
    - optisch
  - 1000BASE-T
  - 1000BASE-X
    - optisch

- Ethernet-Frame



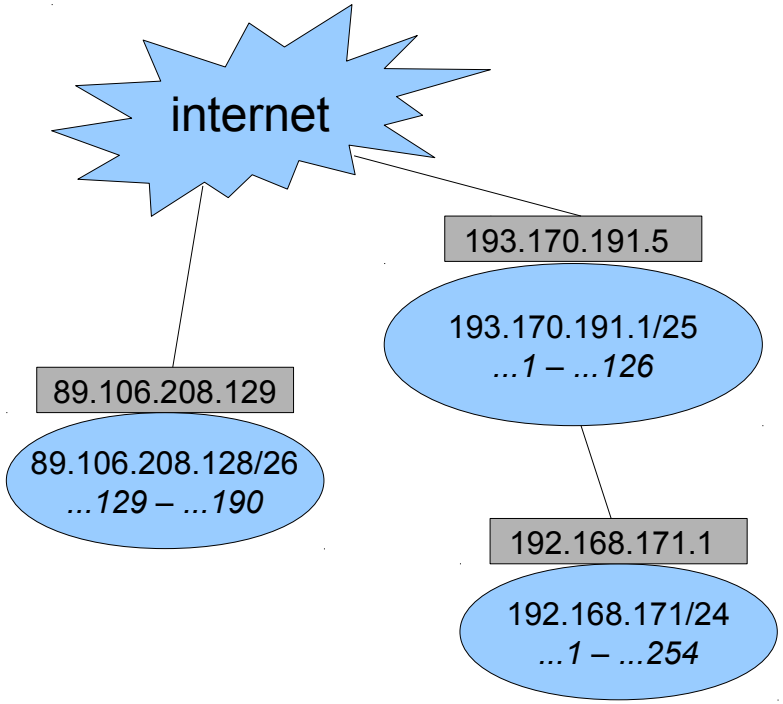
Network Layer: IP (Internet Protokoll)

- Endpoint ↔ Endpoint Verbindung
  - Routing → Komplexe Netzwerk-Strukturen (Internet)
- Routing
  - Paket-Verlust
  - Mehrfachübertragung
  - Alternative Pfade
  - Fragmentierung



IPv4 Netzwerke

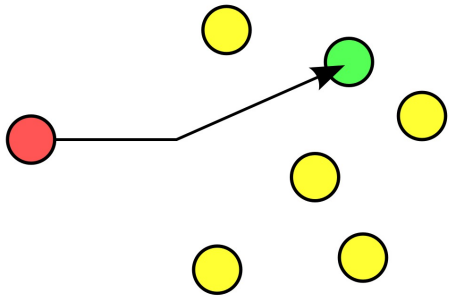
- IPv4
  - 32bit: 4.294.967.296 Adressen
  - *3.Feber 2011*
  - (IPv6: 128bit – $3.4*10^{38}$)
- Segmentierung:



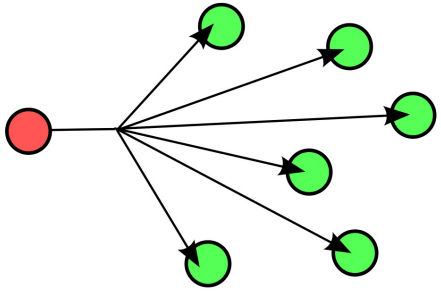
- Private Netzwerke
  - reservierte IP-Adressen
    - 10.0.0.0/255.0.0.0
    - 172.16.0.0 – 172.31.255.255
    - 192.168.0.0/16
  - NAT (Network Address Translation)

Network Layer: IP

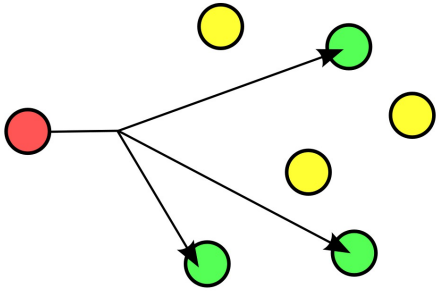
- Unicast
  - 1 → 1
  - 192.168.7.63



- Broadcast
  - 1 → all
  - 192.168.7.255

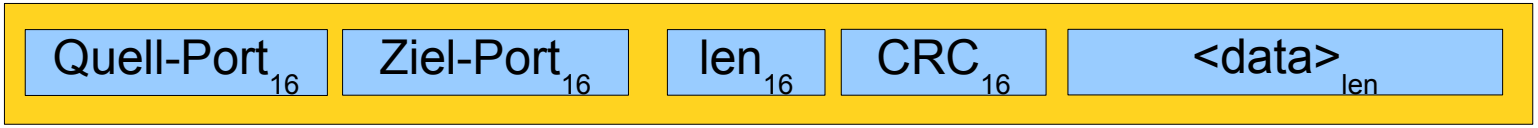


- Multicast
  - 1 → some
  - 224.0.0.90



Transport Layer: UDP

- „User Datagram Protocol“
- Paketorientiert
- „verbindungslos“, unsicher
- Stream-Multiplexing
  - Ports



- Anwendung
  - Datenintegrität < Übertragungsgeschwindigkeit
  - Robust gegen Empfänger-Ausfall

Transport Layer: TCP/IP

- „Transmission Control Protocol“
- Stream-orientiert („seriell“, keine Pakete)
- Verbindungs-orientiert

- Handshake
  - SYN – SYN/ACK – ACK
- Paket-Reihenfolge
  - Sequenz-Nummer

- Re-Transmission
  - verlorene/fehlerhafte Pakete
- Flow Control

- Anwendung
  - Datenintegrität > Übertragungsgeschwindigkeit

FUDI

- „Fast Universal Digital Interface“
- Netzwerk Protokoll
  - TCP/IP
  - UDP
- Text-basiert (ASCII)

FUDI messages

- „<symbol>{␣<atom>},“
- atom: <number>|<symbol>
- number
  - numerischer Wert
  - Dezimalschreibweise „3.14“
- symbol
  - Text (nicht-numerischer Wert)
  - Sonderzeichen (␣, ;, ,)
- Trennzeichen
  - Atom-Trenner: „␣“ (whitespace)
  - Message-Trenner: „,“
  - End-of-Message: „;“

FUDI

- Vorteile
  - sehr einfach
  - human readable
  - erweiterbar
- Nachteile
  - nicht hardware-optimiert
  - kein Timestamping
  - keine komplexen Datentypen
  - Symbole
    - viele verbotene Sonderzeichen
  - Zahlen
    - schlecht abgebildet (Genauigkeit)
  - geringe Verbreitung

Steuerungstechniken

- Netzwerk-Kommunikation (cont.)
  - OSC
- Computermusik-Sprachen
  - Sprachparadigmen
    - prozedural vs funktional

Open Sound Control

- Generisches Kommunikationsprotokoll
- CNMAT (1997)
- Nachfolger von ZIPI
  - *Zeta Instrument Processor Interface*
  - CNMAT/Zeta (1994)
  - Music Parameter Description Language (MPDL)
  - MIDI Nachfolge?

Open Sound Control (OSC)

- Message
  - <Adresse> {<Parameter>}
- Adressen
  - symbolischer Namensraum
    - offen
    - hierarchisch
    - dynamisch
- Parameter
  - numerische, symbolische,... Typen
- mehrere Ziele (Adressen) in einer Nachricht
- synchrone Nachrichten („atomic commits“)
- Timing
  - genaues Timestamping

OSC Transport Layer

- Transport *agnostisch*
- Paket-orientiert
  - ursprünglich: UDP
- später stream-orientierte Transport-Protokolle
  - TCP/IP
  - serial-over-USB (Microcontroller)

OSC über Stream-basierten Transport

- SLIP-encoding
  - *Serial Line Internet Protocol*
  - Pakete werden mit 0xC0 umschlossen
  - 0xC0 → 0xDB 0xDC
  - 0xDB → 0xDB 0xDD

  - 0xA0 0xB0 0xC0 → 0xC0 0xA0 0xB0 0xDB 0xDC 0xC0

OSC messages

- <address>,<typetags>{args}
- Bsp:
  - /filter/coeffs,ff 0.4 0.7

- <address> ↔ Methode am OSC-Server

  filter::coeffs(1.7, 376.2231);

OSC Adressen

'/synthA/channel2/gain'

- Hierarchischer Adressraum
  („Gain vom Kanal#2 im SynthA“)
  - Trennsymbol: '/'
- reservierte Buchstaben
  - ␣
  - #,/
  - *?[]{}
- gefolgt von Typetag-String
  - beginnt mit ','

OSC Typen

Typ	TypeTag	Beschreibung
int32	i	32bit Integer
float32	f	32bit Float
OSC-string	s	\0-Terminierter String (Zero-Padding auf 4bytes)
OSC-blob	b	len(int32)+bytes[len] +Zero-Padding
…	…	…

- Alle Typen sind 32bit aligned
- alle Strings sind OSC-Strings (Zero-padding!)

OSC messages (Beispiel)

- „/note,i 64“

- am Server:
  - ::note(64);

/note								(i)				64			
/	n	o	t	e				,	i						@
47	110	111	116	101	0	0	0	44	105	0	0	0	0	0	64
0x2F	0x6E	0x6F	0x74	0x65	0x00	0x00	0x00	0x2C	0x69	0x00	0x00	0x00	0x00	0x00	0x40

OSC Address-Patterns

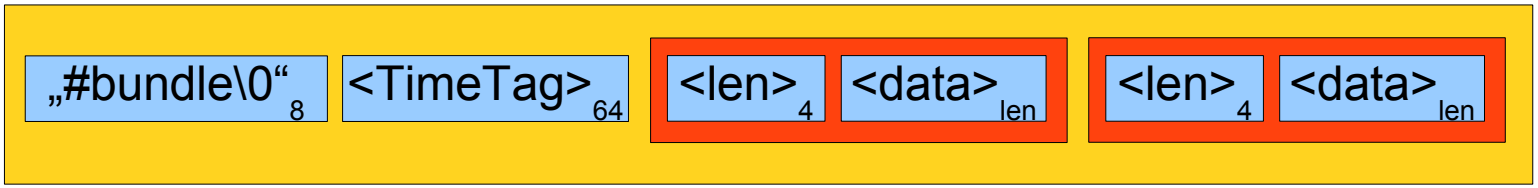
- '/synthA/channel*/gain'

- erlaubte Patterns
  - /neil*/jack
  - /neil?/jack
  - /neil[12a]/jack
  - /neil[1-5]/jack
  - /neil[!9]/jack
  - /neil/{jack,me}

- Reihenfolge der Abarbeitung undefiniert

OSC bundle

- Gruppierung
  - enthält beliebige Anzahl von Messages und Bundles
- TimeTags
- atomic
  - alle Messages passieren zum gleichen logischen Zeitpunkt
  - Reihenfolge muss erhalten bleiben



OSC bundle (Beispiel)

```
{@ 2012-11-08 14:34:56.432151
        /synthA/note,s C#
        /synthB/freq/interpoltime,f 1.0
        /synthB/freq,f 443.0
}
```

- zum Zeitpunkt „ 2012-11-08 14:34:56.432151“
  - spiele am synthA eine Note
  - setze am synthB die Frequenz-Interpolationszeit
  - setze am synthB die Frequenz (evtl. mit Interpolation)

OSC TimeTags

- 64bit absolute Zeit (NTP)
  - 32bit Sekunden seit 1900-01-01 (136 Jahre)
    - rollover 2036!
  - 32bit Sekundenbruchteil (233 Pikosekunde)
  - 00000...0001 ↔ „immediately“
- simpel
  - Ignorieren des TimeTags und „sofort“ ausführen
- OSC1.0
  - Client scheduled Events in der Zukunft
  - Server führt Events zum richtigen Zeitpunkt aus
- auch:
  - Events werden zum Zeitpunkt des Auftretens im Client getaggt
  - Server kann Jitter eliminieren

OSC Evaluierung

- „Content-Format“ aber kein „Protokoll“
  - Format-Beschreibung
  - keine semantische Beschreibung
- Interoperabilität
  - benötigt Higher-Level Protokolle
  - Service Discovery(?)
- Hardware
  - Außeruniversitäre „Produkte“?
- Software
  - praktisch jede CM-Umgebung
  - OSC-Fernsteuerung via „Touch-Devices“

Programmieren

"Programs must be written for people to read, and only incidentally for machines to execute."
(Abelson & Sussman)

Programmieren

- Programmieren = Problemlösen

- Kriterien
  - Korrektheit
  - Geschwindigkeit
  - Wartbarkeit
  - ...

Programmieren in der Computermusik

- „Korrektheit“
  - benötigt klare Problemstellung

- Herangehensweisen
  - bottom-up
    - spezifische Lösung für „Stück“
  - top-down
    - generische Lösung, z.B. „Kompositionsmaschine“
    - praktisch unmöglich
  - → Frameworks

Sprachparadigmen

- imperativ
  - prozedural
- deklarativ
  - funktional
  - logisch
- objekt-orientiert

Imperative/Prozedurale Sprachen

- ProgrammiererIn legt fest, „wie“ ein Problem gelöst werden muss
- Programm
  - Folge von Handlungsanweisungen

```
unsigned int factorial(unsigned int n) {
  unsigned int result=1;
  for(unsigned int i=1; i<n; i++)
    result=result*i;
  }
  return result;
}
```

Deklarative Sprachen

- Programm beschreibt „was“ berechnet werden soll
- System muss sich selbst darum kümmern, „wie“ es zum Ergebnis kommt

- z.B. SQL
  - `SELECT * FROM events WHERE timestamp<7843;`

Funktionale Sprachen

- basierend auf Mathematischen Funktionen
- Inputs → Output
  - *f*(x, y, z) → a
- kein Kontext!
  - keine Seiteneffekte
  - alle Funktions-Argumente können parallel berechnet werden

Funktionale Sprachen

- Lambda-Kalkül
  - Anonyme Funktionen

```
sqrsum(x, y) = x*x+y*y
(x,y) ↦ x*x+y*y
```

  - Currying
    - Funktionen liefern Funktionen zurück

$$x \mapsto (y \mapsto x*x+y*y)$$

$$f(X, Y) \rightarrow (f(Y))(X)$$

- Rekursive Strukturen
  - loops

Beispiel: LISP

```
(defun append (list1 list2)
      (cond ((null list1) list2)
            ((null list2) list1)
            (t (cons (car list1) (append (cdr list1) list2))))))
```

- defun
  - Funktionsdefinition
- cond
  - Condition (if/elseif/else)
- Konstanten
  - t = TRUE
  - null = überprüfe auf leere Liste

- Listen
  - **cons**: Listenelement, bestehend aus
  - **car**: Kopf der Liste
  - **cdr**: Rest der Liste

Computermusik-Sprachen

- Sprachparadigmen
  - deklarativ: logisches Programmiersprachen
- Entwicklung von CM-Sprachen
- Non-Realtime CM-Systeme
  - CDP

Imperative vs Deklarativ Sprachen

- Imperatives Programmieren
  - ProgrammiererIn legt fest, „wie“ ein Problem gelöst werden muss
- Deklaratives Programmieren
  - Programm beschreibt „was“ berechnet werden soll
  - funktional
  - logisch

Funktionale Sprachen

- basierend auf mathematischen Funktionen
- Inputs → Output
  - *f*(x, y, z) → a
- kein Kontext!
  - keine Seiteneffekte
  - alle Funktions-Argumente können parallel berechnet werden

Logische Sprachen

- Beschreibung von mathematischen Theoremen
- Prädikat-Kalkül
- Prädikat
  - Funktion die Wahr/Falsch liefert
- Programmieren
  - Anlegen von Datenbasis
  - Anfrage (Query)
    - Antwort Ja/Nein
    - System versucht Query zu beweisen
      - *„Automatischer Theorem-Beweis“*

Beispiel: PROLOG

```
append([], X, X).
append(X, [], X).
append([X1 | X], Y, [X1 | Z]) :- append(X, Y, Z).
```

- Syntax
  - **[]** : (leere) Liste
  - **:-** : Implikation (←)
  - **|** : Listen-Kopf, Listen-Rest
  - XXX: Variable
  - xxx : Konstante/Funktion

Objekt-Orientierte Sprachen

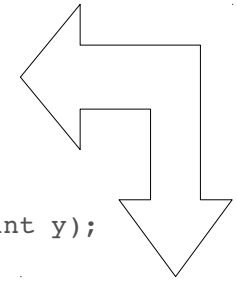
- Kapselung von zusammengehörigen Daten/Funktionen in logische Einheit
- Datentyp: Klasse
- Datenelement: Objekt
  - Daten (Members)
  - Operatoren (Methods)
    - prozedural

```
class Shape {
  int X, Y;
  void draw(void);
  void move(int x, int y) {
     X=x; Y=y;
  }
};
```

Vererbung (Inheritance)

- Neue Klassen als Spezialisierungen/ Varianten von existierenden Klassen
  - Beibehaltung
  - Erweiterung
  - Modifikation

```
class Shape {
  int X, Y;
  void draw(void);
  void move(int x, int y);
};
```



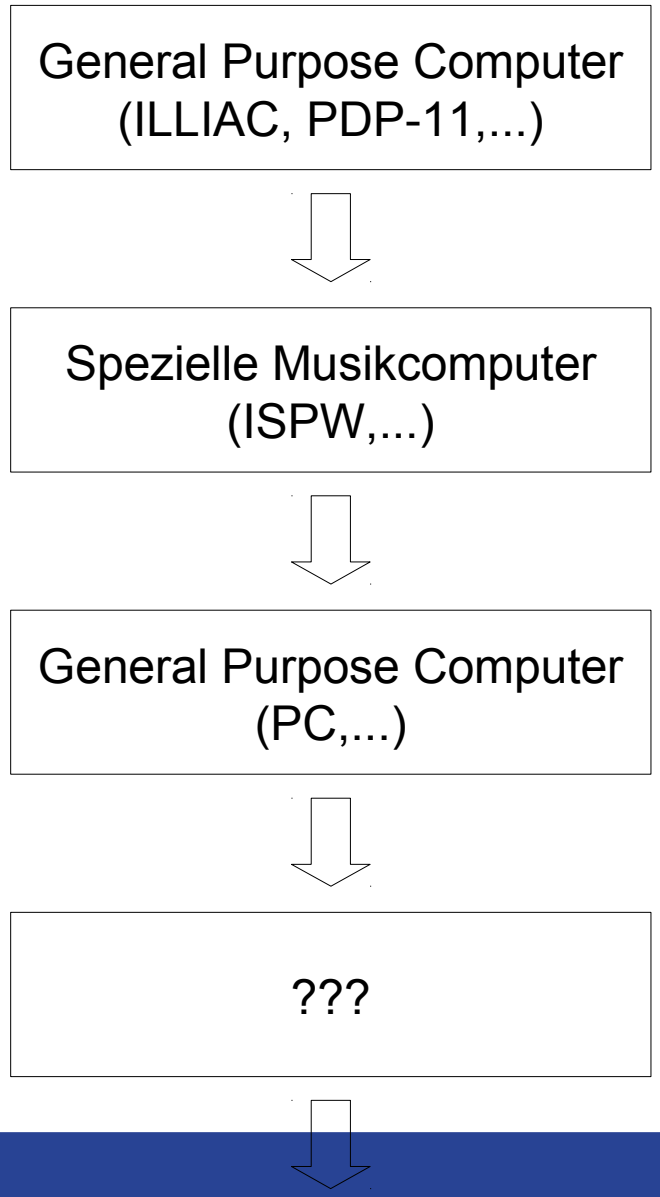
```
class Square : Shape {
  void size(float f);
  void draw(void);
}
```

Sprachen in der Computermusik

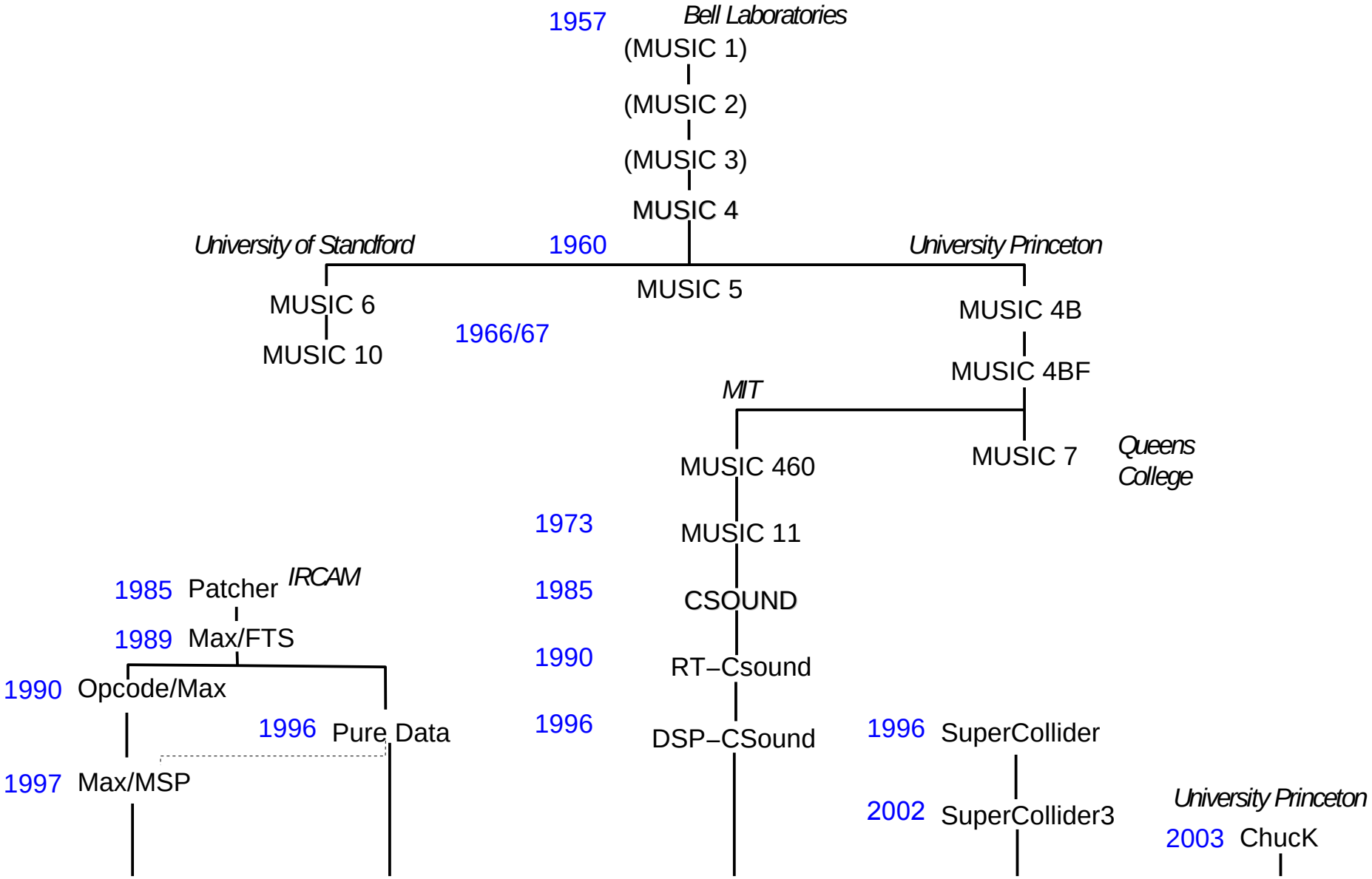
- Geschichtlicher Überblick
- Unit Generator
- Offline Systeme

Sprachen in der Computermusik

- Ziel-Platform
  - abhängig von
    - Verfügbarkeit
    - Leistungsfähigkeit



Stammbaum der Computermusiksprachen

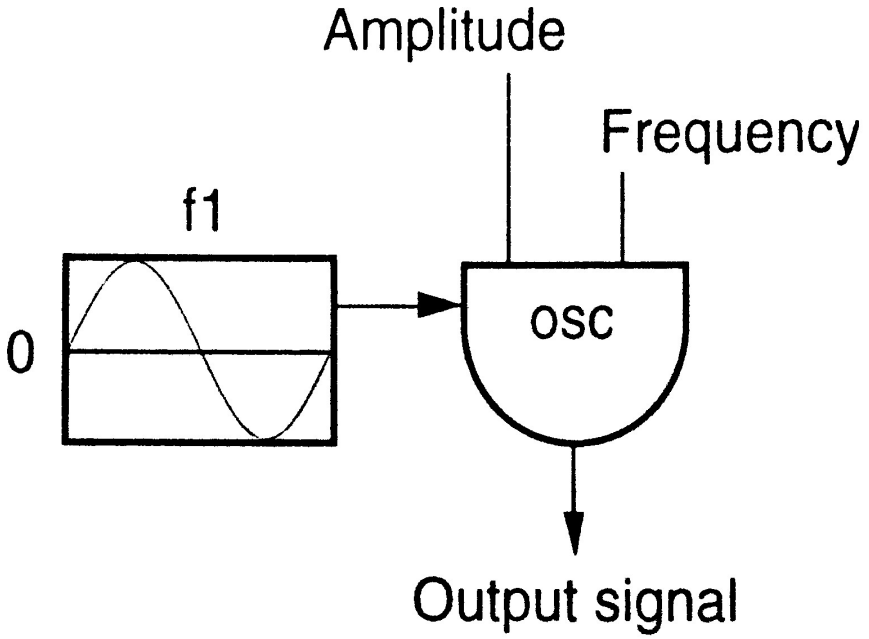


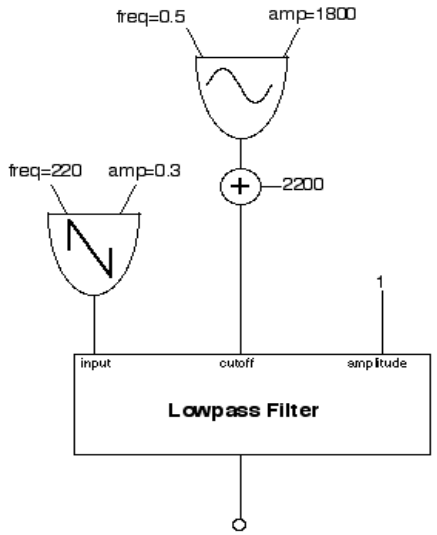
Music-N

- *Musiksynthese*
- MUSIC 1-4: Max Mathews
- MUSIC-4B
  - IBM Mainframe
- MUSIC-11
  - PDP-11
- MUSIC5, MUSIC-4BF
  - FORTRAN

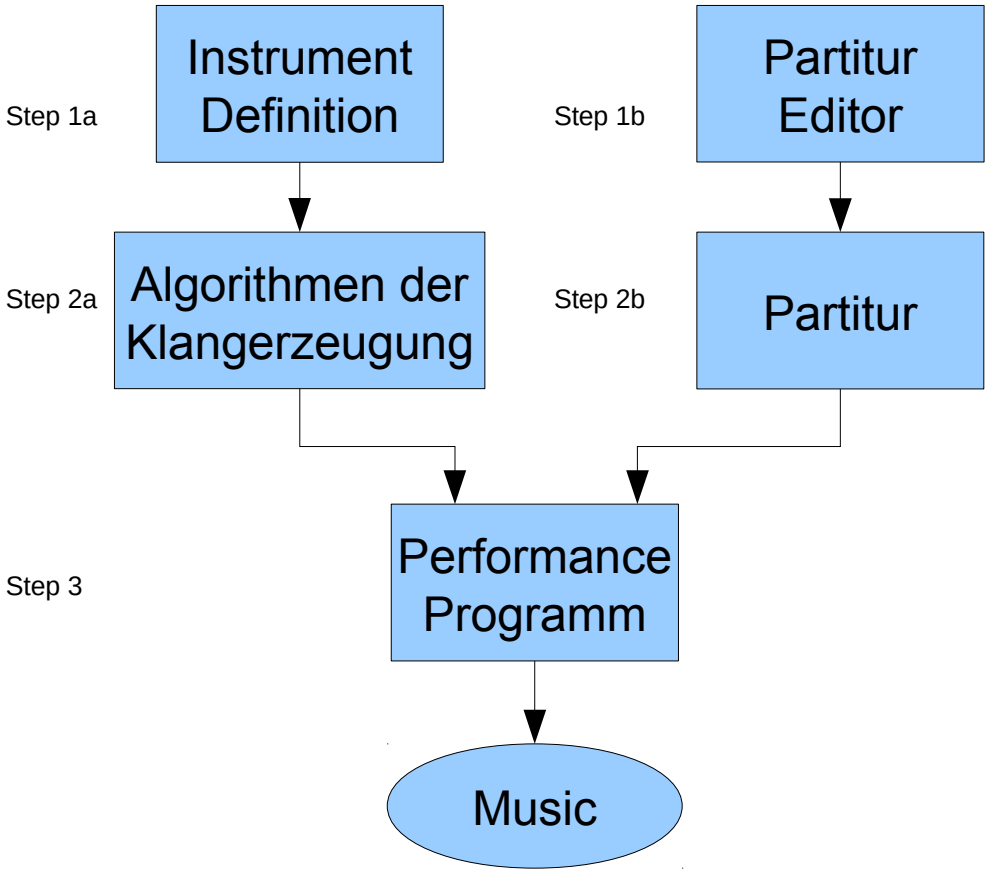
Unit Generator (UGen)

- Music III (1960)
- Signalverarbeitungs-modul
  - Oszillator
  - Filter
  - Verstärker
  - ...
- sind verschaltbar
  → Patches
  (Instrumente))





MUSIC-4


Step 1a
Instrument Definition
Step 1b
Partitur Editor
Step 2a
Algorithmen der Klangerzeugung
Step 2b
Partitur
Step 3
Performance Programm
Music

Computermusik Systeme

- Offline-Systeme
- Online-Systeme
  - Real-Time Systeme

CDP – Composers' Desktop Project

- University of York
- Mainframe → Personal Computer

- 1986
  - AtariST
- 1994
  - PC

CDP Prinzip

- Klänge als Material
  - „Musique concrète“
- Bearbeitung von ganzen Soundfiles
- Sammlung von kleinen spezialisierten Programmen
  - „Intruments“
- Batch-Verarbeitung

CDP Instruments

- Transforming the waveform of the sound
  - extracting and transforming the loudness contour
  - various time-variable banks of filters
  - different types of 'waveset' distortion
- Segmenting the sound, and reassembling segments
  - 'harmonizer' type time-stretch or pitch-shift
  - sound-shredding
  - reverberation
  - regular, to random, brassage - from spectral modification to event-scale montage (similar to granular synthesis)
- Transforming the time-varying spectrum of the sound
  - pitch and formant extraction and manipulation
  - morphing from one sound to another
- Building new events from the whole sound
  - multi-dimensional time-varying texture generation (aka: granular synthesis)
  - mixing, e.g. with attack synchronisation

CDP Example

- „Make a strong bar of sound then a glissando down and a glissando up from a recorded material and Mixing these down.“

```
specchord input1 out1 chord.txt -p12
specglis out1 final1 -2 -p12
specglis input2 final2 4 -p12
splice final final1 final2
```

Computermusik-Sprachen

- CSound
  - Score/Orchestra
  - Scheduling

CSound

- Music-11 Nachfolger
  - 1985
  - MIT (Barry Vercoe)

- 2-teilige Programme
  - Orchestra
    - Sammlung von Instrumentendefinitionen
  - Score
    - steuert Instrumente des Orchesters

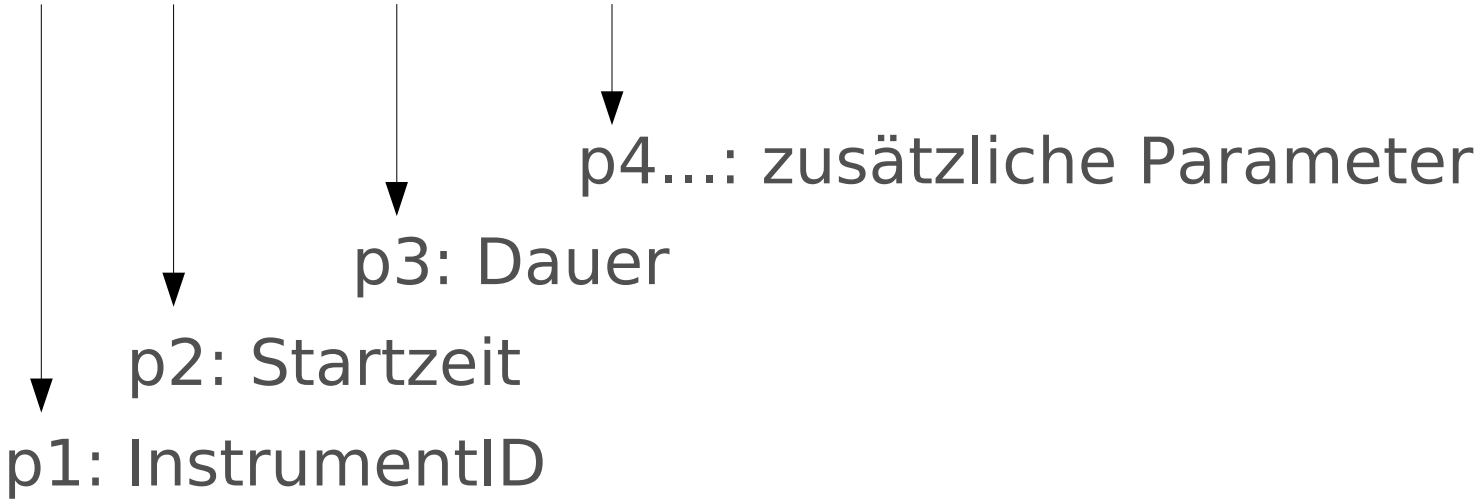
CSound Score

- *i*-Anweisungen
  - Spielen eines Instruments (Parameter)
- *f*-Anweisungen
  - Datenfelder
- *s*- und *e*-Anweisungen
  - Partitur-Abschnitte

```
f 1 0 512 9 1 1 0
i 1 0.000 0.150 424 20000 1000 0.010
i 1 0.150 0.300 727 20000 1000 0.010
s ; section#1 ends
i 1 0.000 0.150 621 20000 1000 0.010
i 1 0.150 0.300 326 20000 1000 0.010
e ; score ends
```

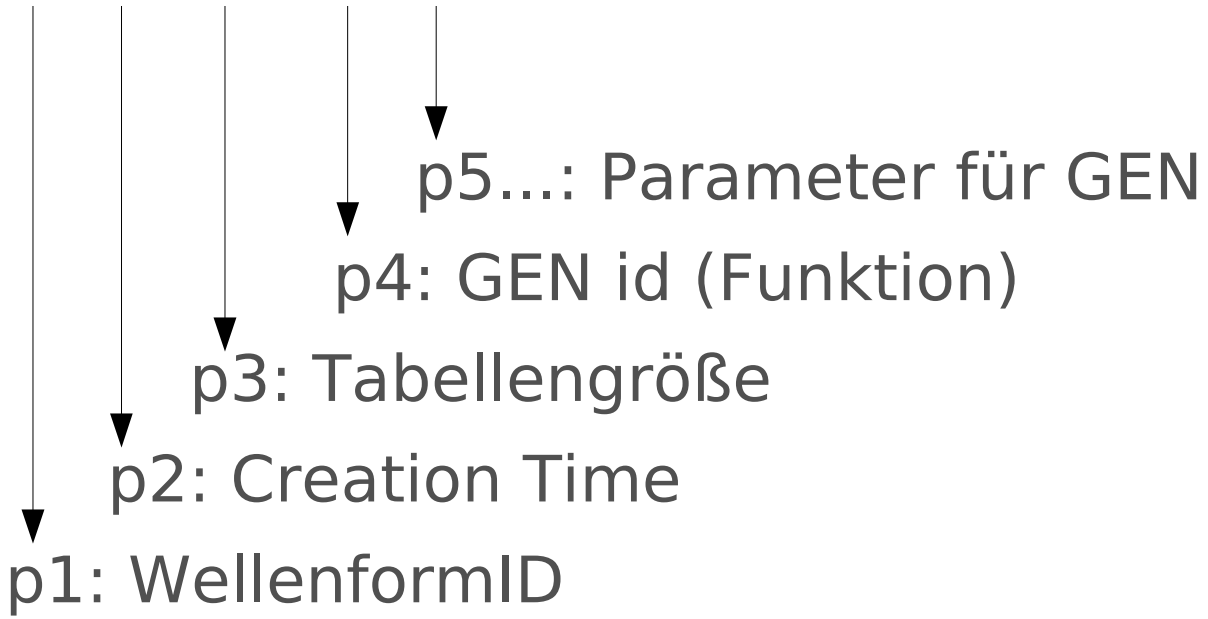
CSound Score: Instrumentenaufruf

```
i1 0.450 0.150 424 20000 1000 0.010
```



CSound Score: Function Tables

```
f2  0  512  7  0  43  1  171  1  84  -1  171  -1  43  0
```

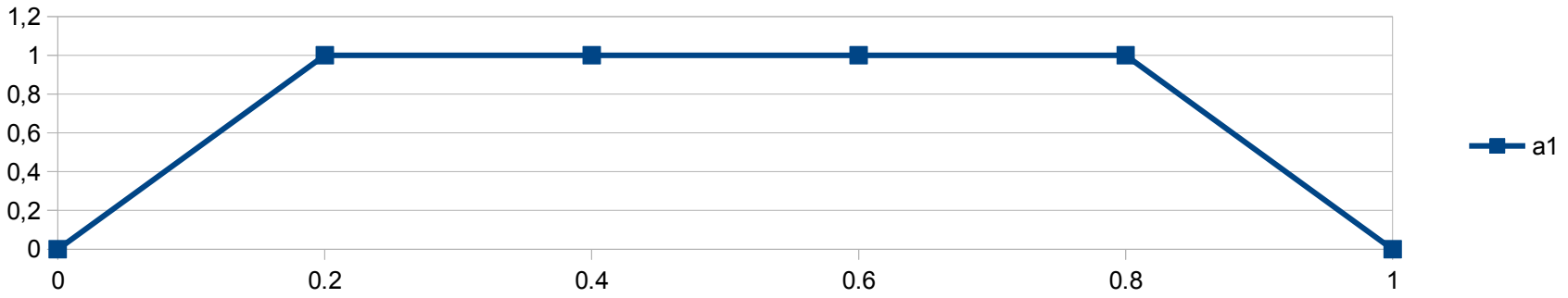


CSound Orchestra

- Definition von Instrumenten
  - Signalverarbeitung
  - jedes Instrument hat ID

```
[variable] <opcode> [parameter1],...
```

```
a1 linseg 0, 0.2, 1, 0.8, 1, 0.2, 0
```



CSound Scheduling

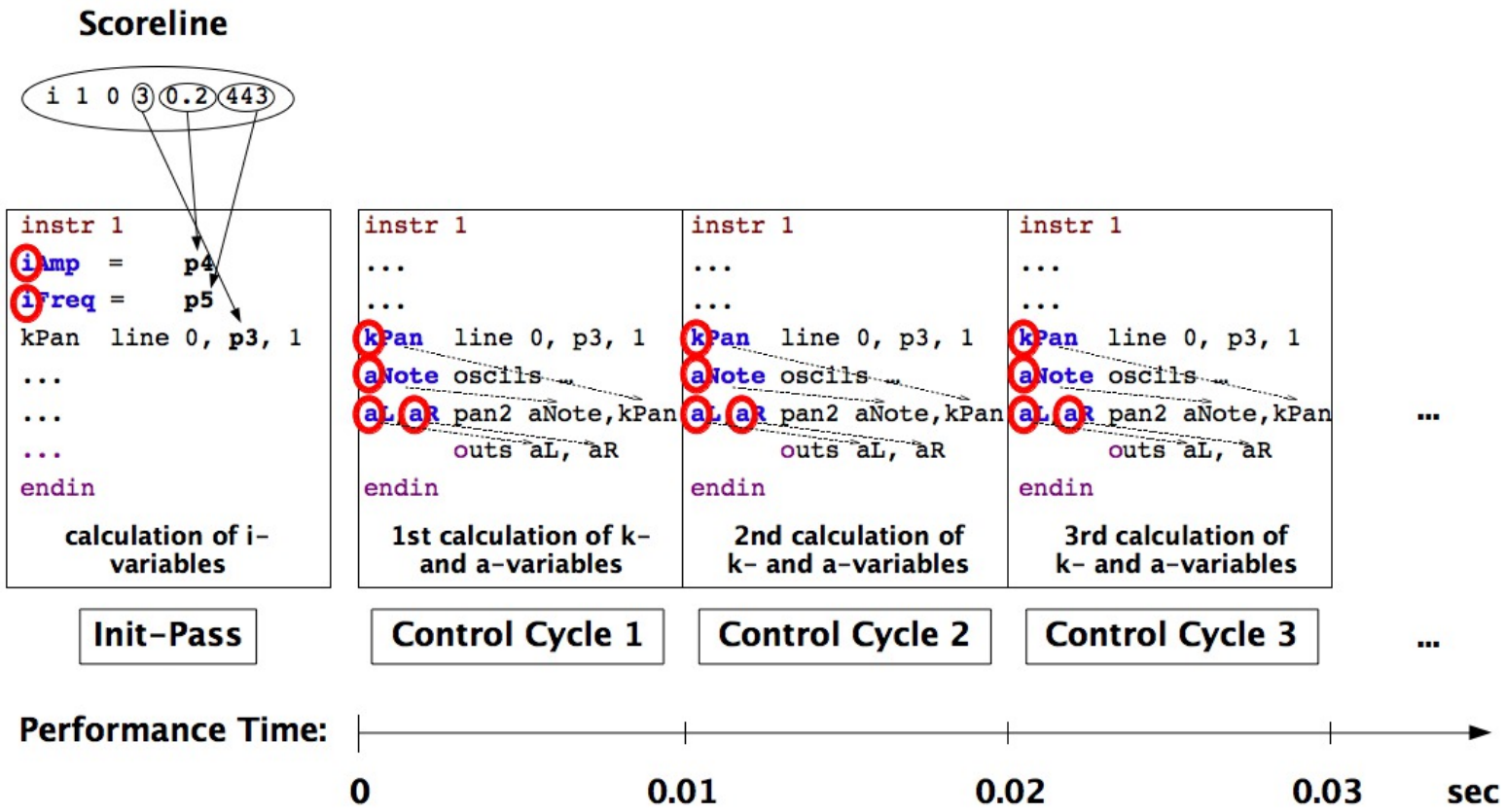
- Init-Zeit
  - *i-time*
  - Initialisation zu Beginn jedes Instrumenten-Aufrufs („Note Time)
- Performance-Zeit
  - Sample-Rate (*srate*)
  - Control Rate (*krate*)
    - Parameter Updates
    - Effizienz: niedriger als Sample-Rate
    - Praktikabilität: srate/krate ganzzahlig

CSound Namenskonventionen

- Variablen-Namen
  - typenabhängige Prefixe

Variablen-Typ	Update-Zeitpunkt	Prefix
Score Paramenter	i-time	p
Init Variable	i-time	i
Control Signal	k-rate	k
Audio Signal	k-rate	a

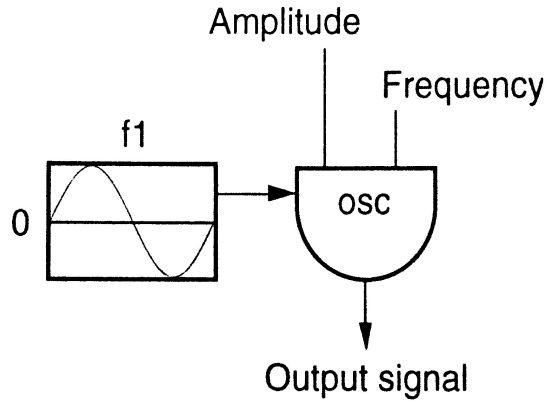
CSound Scheduling



- audio
  - Signale werden als Samplearrays zu den k-times abgearbeitet

CSound Opcodes

- Unit Generators



- User Defined Opcodes (UDO)

- evtl. eigene k-rate
  - **setksmps**

```
opcode Foo, k,k
  ka xin
  kb = ka * 2
  xout kb
endop
```

Echtzeit-Systeme

- Anforderungen
  - Latenz
- Echtzeit Datenfluss Systeme
  - Graph
  - Scheduling
  - Rekursion
  - Signalblöcke

Zeit

- Echtzeit
  - Zeit von Abläufen in der „realen Welt“
- Modellzeit
  - „logische“ Zeit
  - von Software selbstverwaltete Laufzeit
- Echtzeitfähigkeit
  - Modellzeit synchron zur Echtzeit

Echtzeit-Systeme

- Ergebnis wird garantiert innerhalb eines definierten Zeitintervalls berechnet
- „rechtzeitiges“ Reagieren
- Zeitintervalle
  - Stunden/Minuten
  - µSekunden

Echtzeit-Anforderungen

- harte Echtzeitanforderung
  - Überschreitung der Reaktionszeit → maximaler Schaden
- weiche Echtzeitanforderung
  - Verletzung der Deadline → keine katastrophalen Auswirkungen

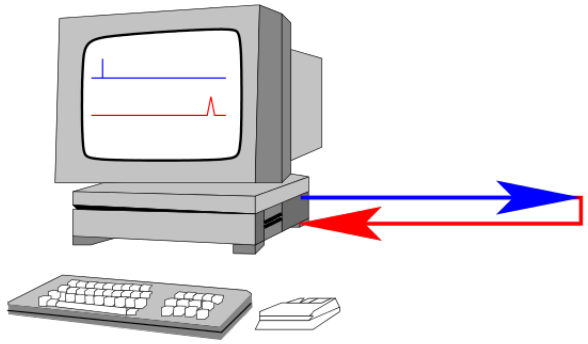
- Computermusik
  - Anforderung abhängig von Anwendungszweck
  - z.B. Experimente
    - Überschreitung „ärgerlich“
  - z.B. große Lautsprchersysteme
    - Überschreitung evtl. „gesundheitsgefährdend“
    - harte Echtzeit angestrebt

Strategien

- Ereignissteuerung („IRQ“)
  - externes Ereignis startet Verarbeitung sofort
    - evtl. Unterbrechung bereits laufender Verarbeitung
  - sehr schnelle Reaktion
  - bei vielen Ereingissen → evtl. Überlastung
    - Verlust von Ergebnissen
    - Überschreitung des Zeitlimits
- Zeitsteuerung („polling“)
  - Verarbeitung wird durch Zeitplan gesteuert
  - Rechenaufwand planbar

Latenz

- *lat.* „latens“: verborgen
- Latenzzeit
  - Zeitraum zwischen Aktion und Reaktion
  - Audiosysteme
    - Verzögerung eines Audiosignals, das vom Eingang zum Ausgang durchgereicht wird
- Synchronizität von verschiedene Datentypen
  - Audio & Events wie MIDI sollten gleiche Durchlaufzeiten haben

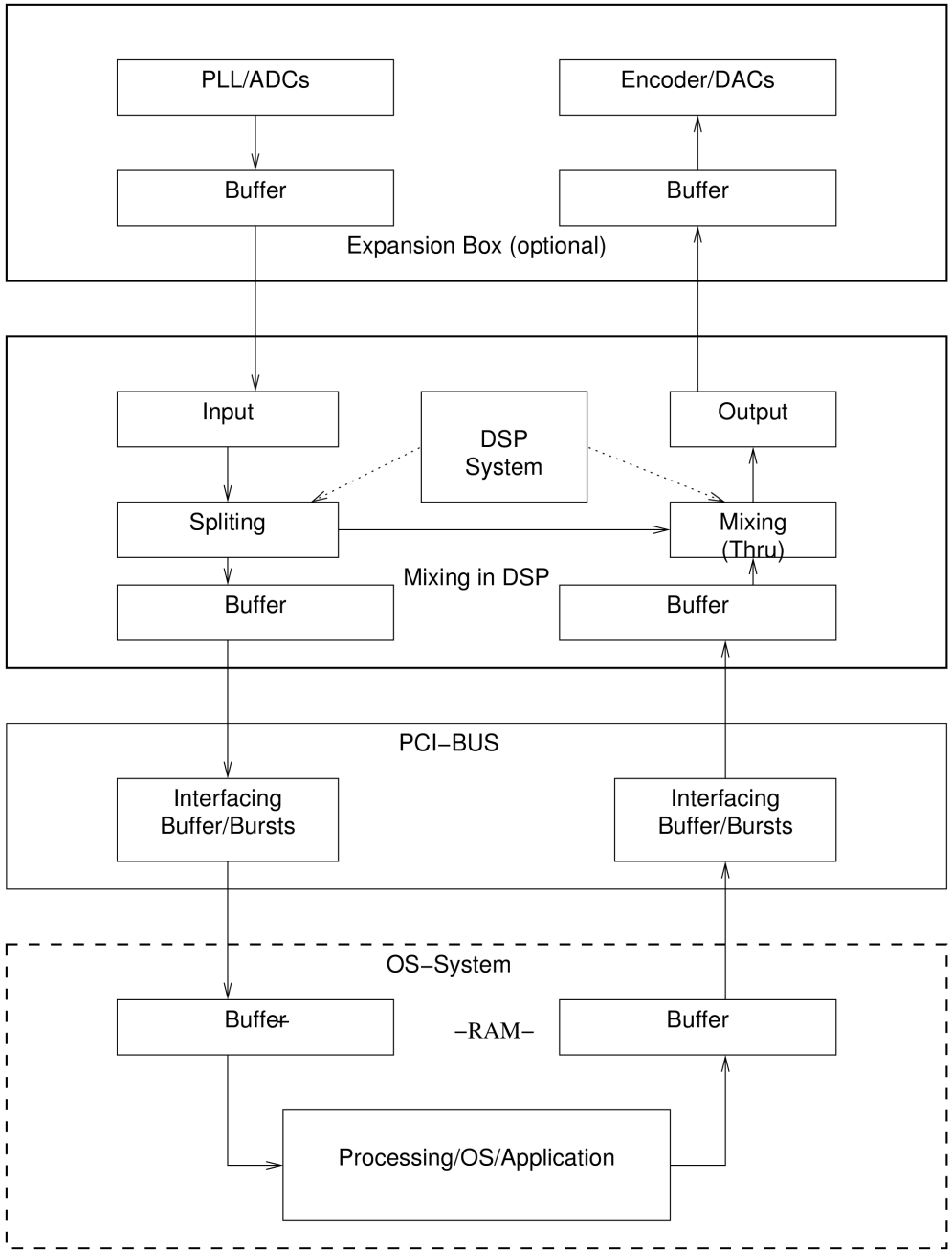


Latenz

- Hardware-Latenz
- Betriebssystem-Latenz
- Applikations-Latenz

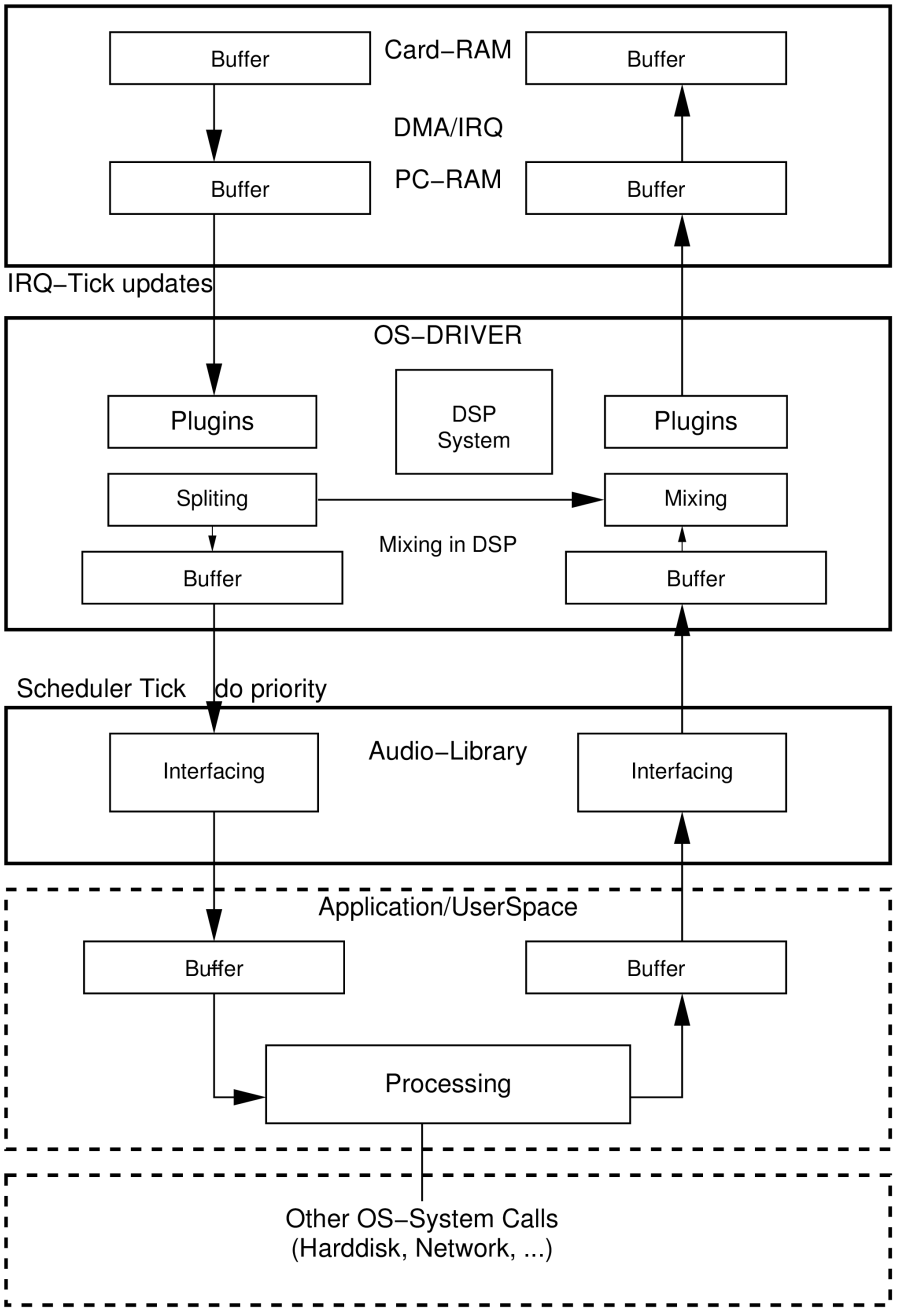
Hardware Latenz

- AD/DA-Umsetzer
  - ~1 sample
- Expansion Board
  - IEEE1394, USB,...
  - Bulk-Übertragung
  - evtl. shared ressource
- PCI-Bus
  - Bulk-Übertragung (mehrere kByte)



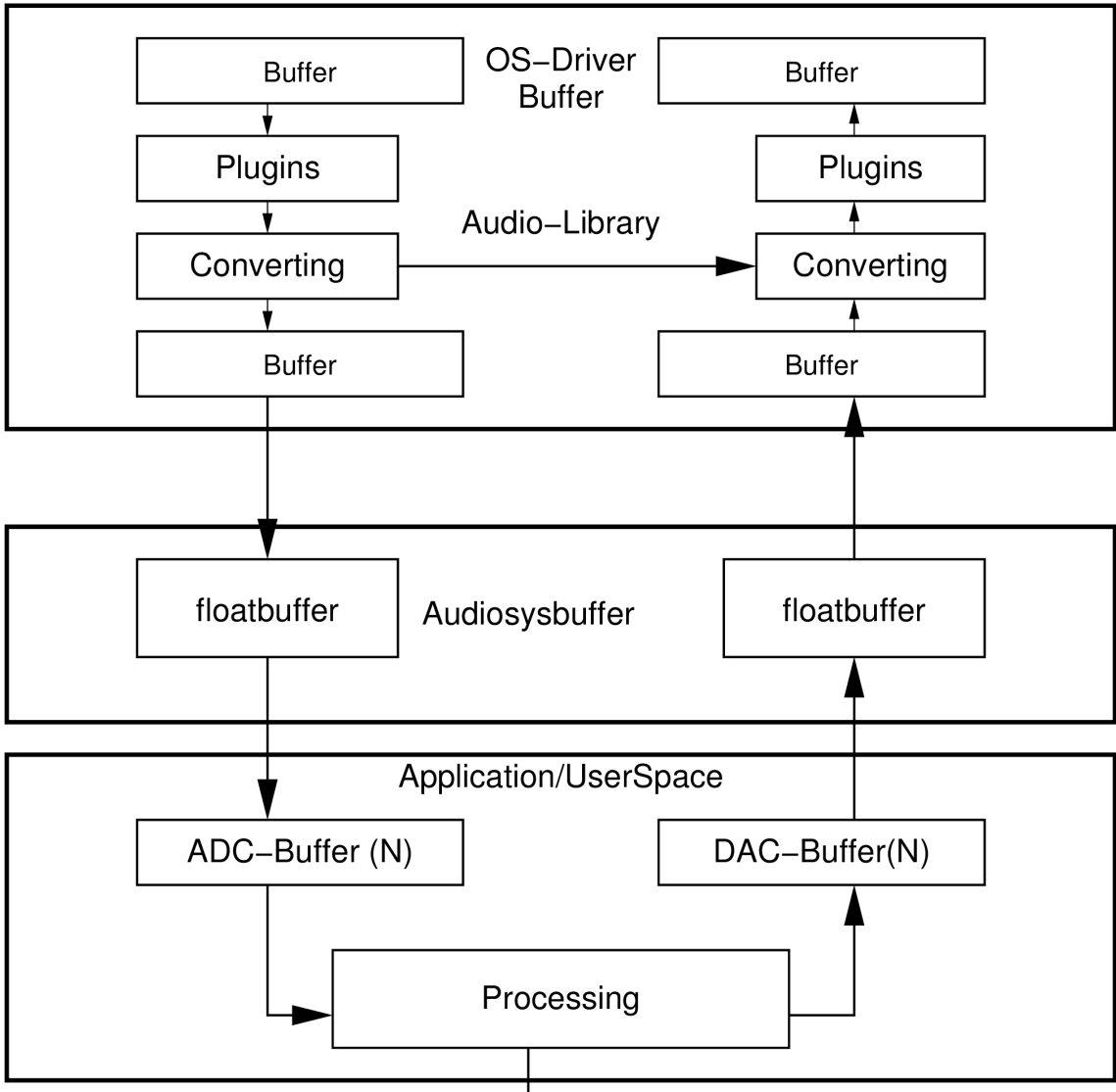
OS Latenz

- System macht evtl. noch Processing
  - z.B. DspFuncLib (OSX)
- Audio-Library
  - generisches Interface (hardware-unabhängig)
  - evtl. Konversion von Samplerate / Sampleformat
  - Interrupt→RingBuffer
  - Interrupt→Callback



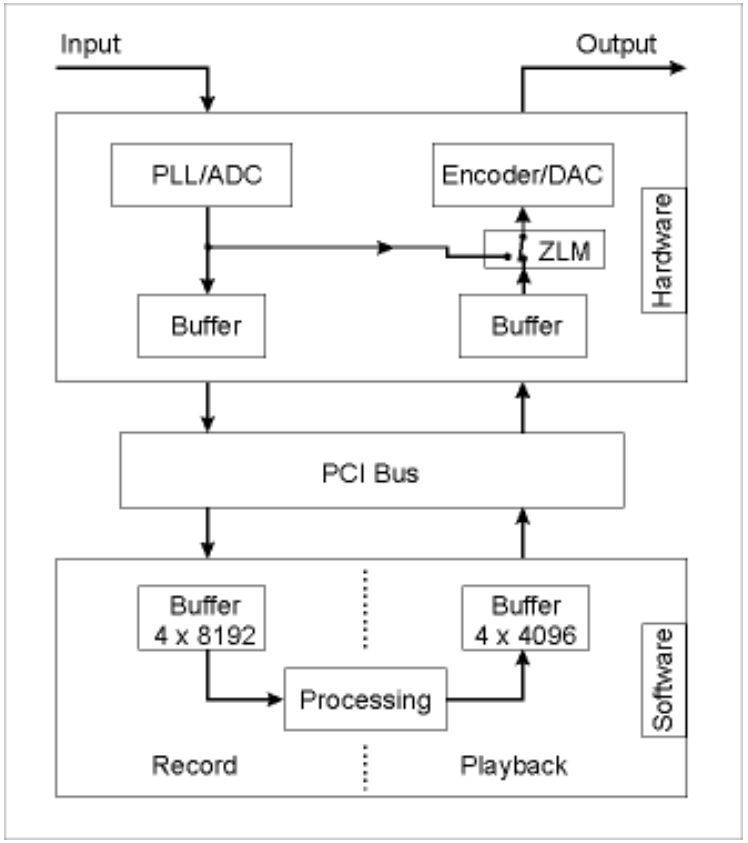
Software Latenz

- Polling
  - App. frägt regelmäßig nach neuen Daten
- Callback
  - OS ruft Funktion in App. auf sobald neue Daten angekommen sind
- Buffer
  - App. entkoppelt Lesen/Schreiben um Peaks abzufangen



Zero-Latency Monitoring

- Monitoring auf Soundkarte
  - Minimallatenz
    - wenige Samples
      - *~1 Sample ADC*
      - *~1 Sample DAC*
  - Umgehung der Latenz von Bus & Software



Signalverarbeitende Echtzeit-Systeme

- Präferenz für Datenfluss-Systeme
- keine Datenfluss-Hardware
  - Implementierung in *von Neumann* Architektur

Data Flow Programming

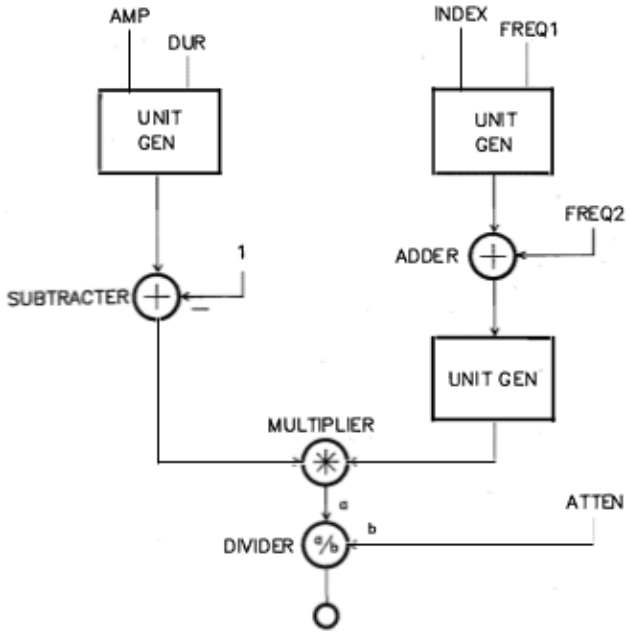
- Daten triggern Verarbeitung

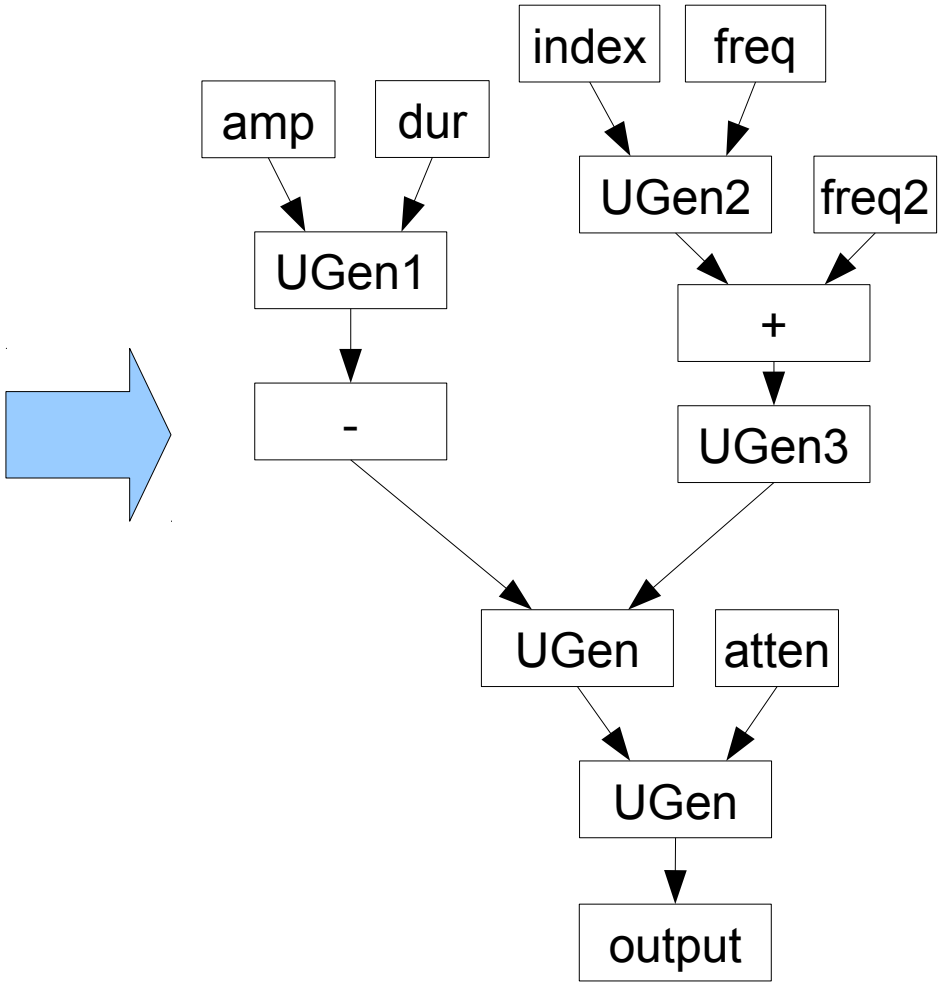
- „Control Flow“
  - Anweisungen werden sequentiell abgearbeitet
  - Operatoren holen sich die benötigten Daten

- Abarbeitungsbedingung
  - alle Eingänge eines Knotens müssen befüllt sein
  - Eingänge ohne Verbindung
    - „Default Value“

Graph

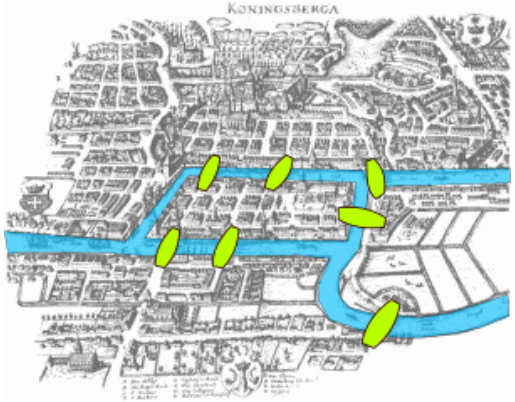
- Abstraktion von UGen-Netzwerken in gerichtete Graphen



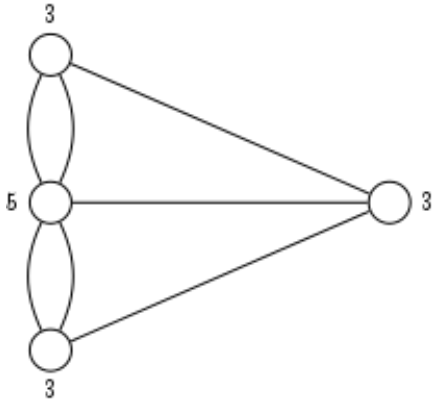


Königsberger Brückenproblem

- Euler (1736)
  - gibt es einen Weg, bei dem man alle sieben Brücken über den Pregel genau einmal überquert, und wenn ja, gibt es einen Rundweg?

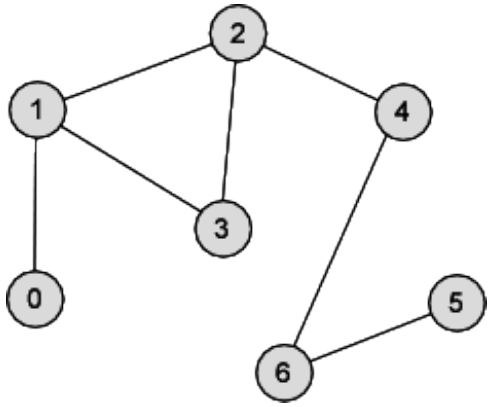


- Topologisches Problem
  - kein Weg möglich, da alle (>2) Knoten eine ungerade Anzahl von Kanten besitzen, aber eine gerade Anzahl benötigt wird, um den Knoten wieder verlassen zu können

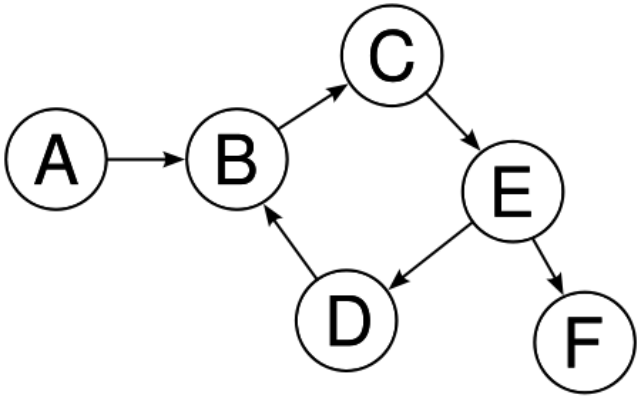


Graphen

- Topologie
  - Knoten
  - Kanten



- gerichteter Graph
  - Kanten haben Orientierung



Data-flow Graphs

- Graph
  - gerichtet
  - azyklisch
- Knoten
  - *actors*
- Kanten
  - *FIFO-Channels*
- Daten
  - *tokens*
  - z.B. Samples, (Sample Arrays)
  - z.B. Spektren (Spectral Arrays)

Sychroner Datenfluss

- Daten fließen synchron
  - alle Daten (im Graphen) werden mit der gleichen Rate abgearbeitet
- Scheduling
- Events
  - per Definition *asynchron*
  - → paralleles System

Aufgabenstellungen

- Data
  - Bereitstellung von Buffern für FIFO-Channels
- Operation
  - process()
- Graph handling
  - Scheduler/Dispatcher

Dataflow Buffer: Optimierung

- Blockverarbeitung
  - Array von „Samples“
    - meist 2^n Samples
    - Default: 64
  - Loop-Unrolling (parallele Verarbeitung)
  - Kontrollparameter müssen nur selten upgedatet/berechnet werden
- Nachteil
  - höhere Latenz
- Vorteil
  - auch für andere Datentypen verwendbar (z.B. Spektren)

Dataflow Buffer: Optimierung

- Cache
  - L3 : ~4MB
  - L1-L2: ~256kB

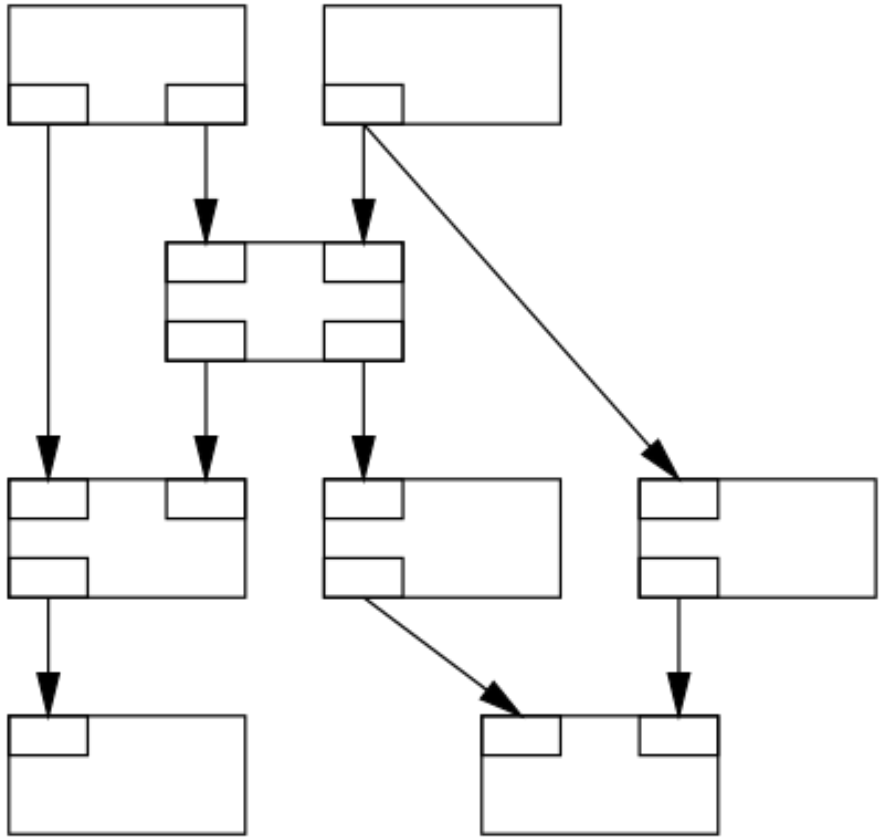
  - nach Möglichkeit Cache-Verletzungen vermeiden
    - alle Daten innerhalb eines Durchgangs sollten in den Cache passen
    - → kurze Buffer
    - → Buffer wiederverwenden

Scheduler/Dispatcher

- Scheduler
  - Graphen-Analyse
    - Rekursion, Verbindungskonsistenz
  - Erzeugung des Execution Graph
  - Buffer-Optimierung
- Dispatcher
  - Verwaltung der Execution Loop
  - start/run/stop

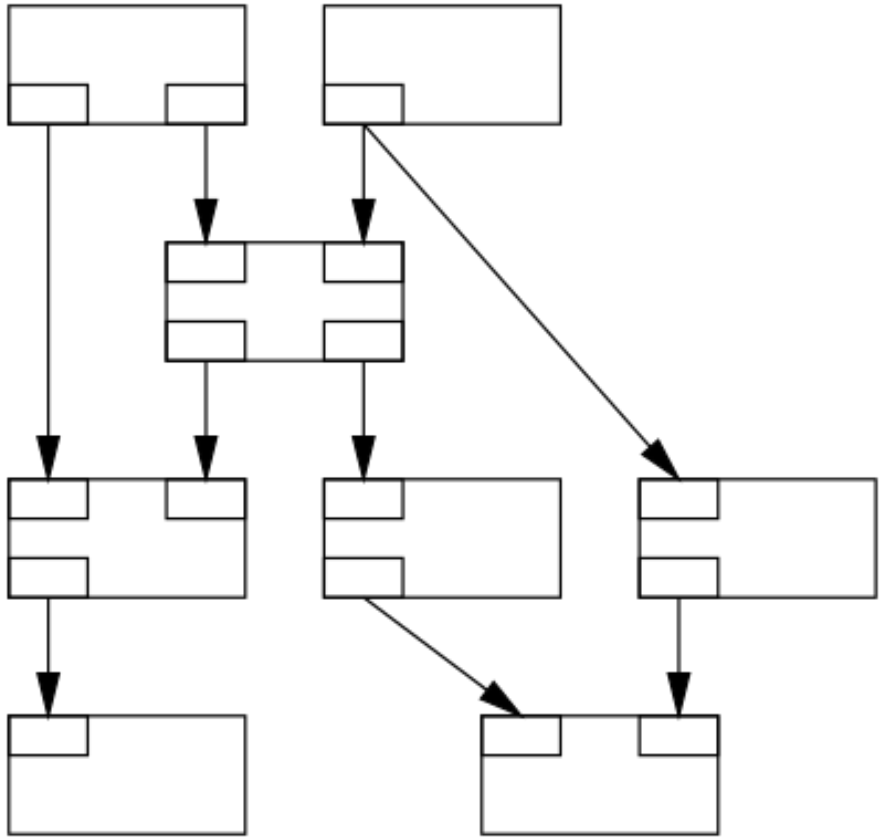
Execution Graph

- gesucht
  - minimale Anzahl an Übergabebuffern
  - → $OUT_i = IN_j$
- top-down approach
  - ausgehend von Quellen (UGens wo alle Eingänge gefüllt sind)
  - „Signal fließt von oben nach unten“
- bottom-up approach
  - ausgehend von den Senken
- Rekursion
  - Teile des Graphen werden vom Scheduler ausgelassen, da Dataflow-bedingung nicht erfüllbar ist



Execution Graph

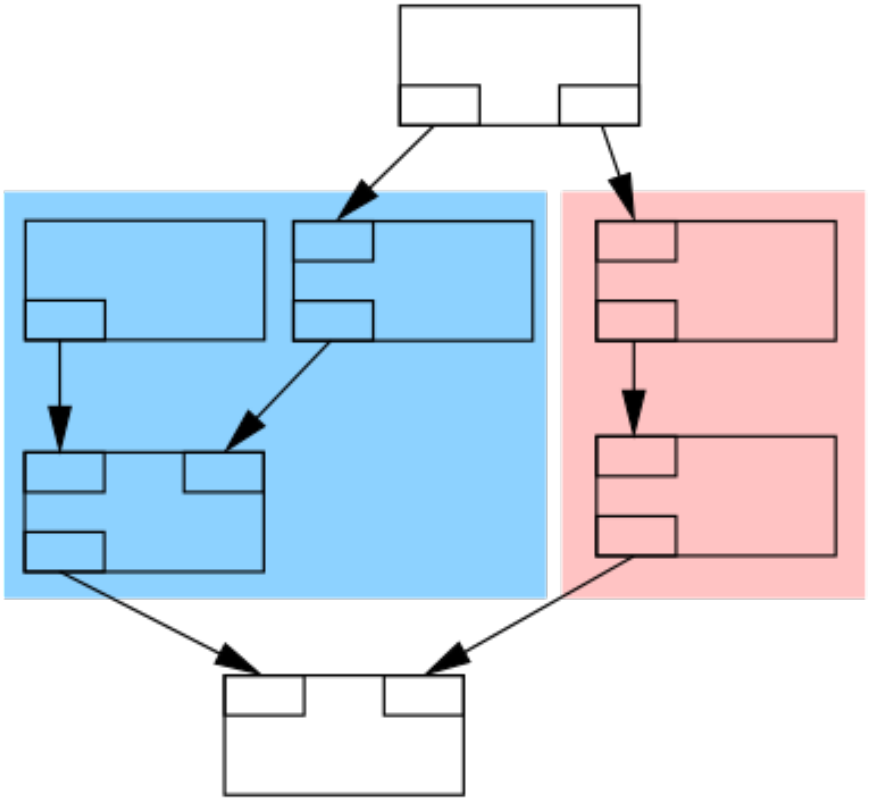
- gesucht
  - minimale Anzahl an Übergabebuffern
  - → $OUT_i = IN_j$
- top-down approach
  - ausgehend von Quellen (UGens wo alle Eingänge gefüllt sind)
  - „Signal fließt von oben nach unten“
- bottom-up approach
  - ausgehend von den Senken
- Rekursion
  - Teile des Graphen werden vom Scheduler ausgelassen, da Dataflow-bedingung nicht erfüllbar ist



Parallelverarbeitung

- Probleme
  - Balance
  - Synchronisation
  - Knoten-Isolation

  → schlecht automatisierbar

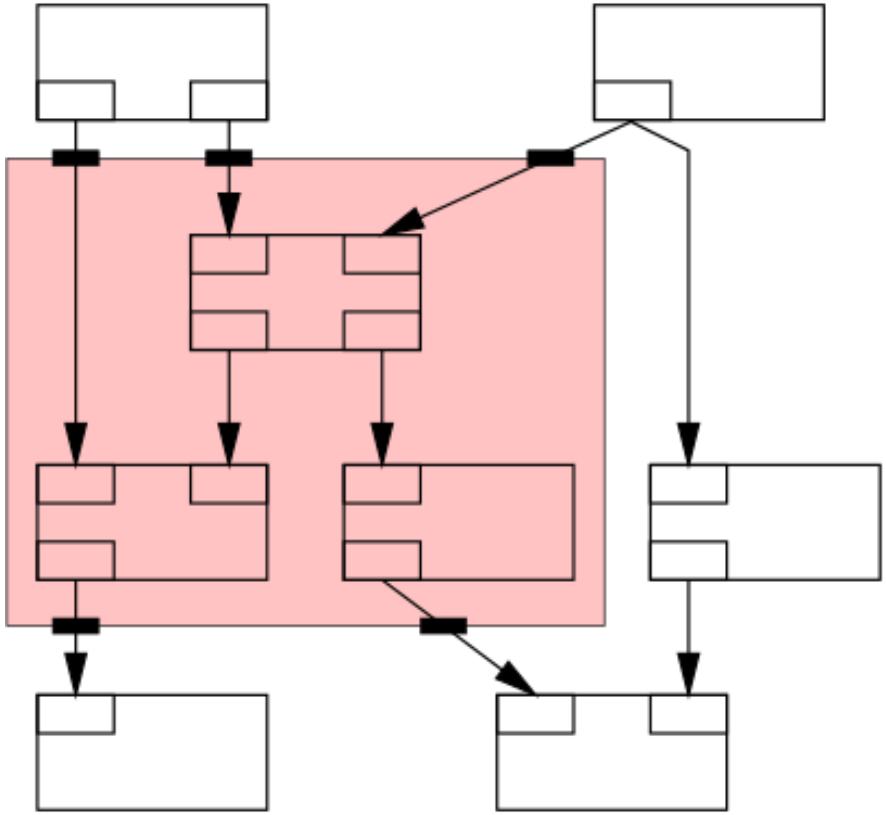


Scheduler/Dispatcher

- Scheduler
  - erstellt Execution-Graph
  - <UGen1> <UGen2> <UGen3>...
  - Buffer-Zuweisung an UGens: *a*, *b*, *c*, ...
- Dispatcher
  - führt Schedule (Execution-Graph) zyklisch aus
    - synchroner „Tick“
  - UGen[1].process(OUT:*a*, OUT:*b*)
    UGen[2].process(OUT:*c*)
    UGen[3].process(IN:*b*, IN:*c*, OUT:*b*, OUT:*c*)
    ...

Re-Blocking

- Block-Eigenschaften gelten für gesamten Graphen
- Sub-Graphen
  - z.B. FFT
- Re-Blocking
  - wo Kanten in Sub-Graph wechseln

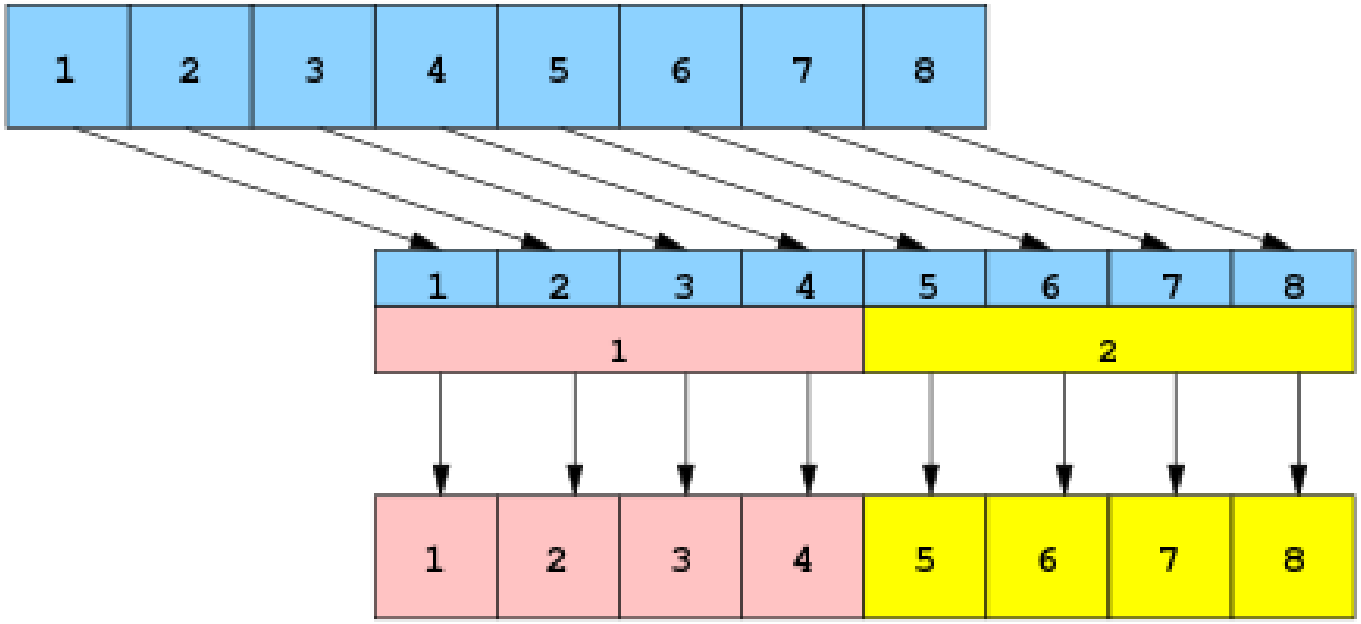


Re-Blocking: 64 → 256 Samples

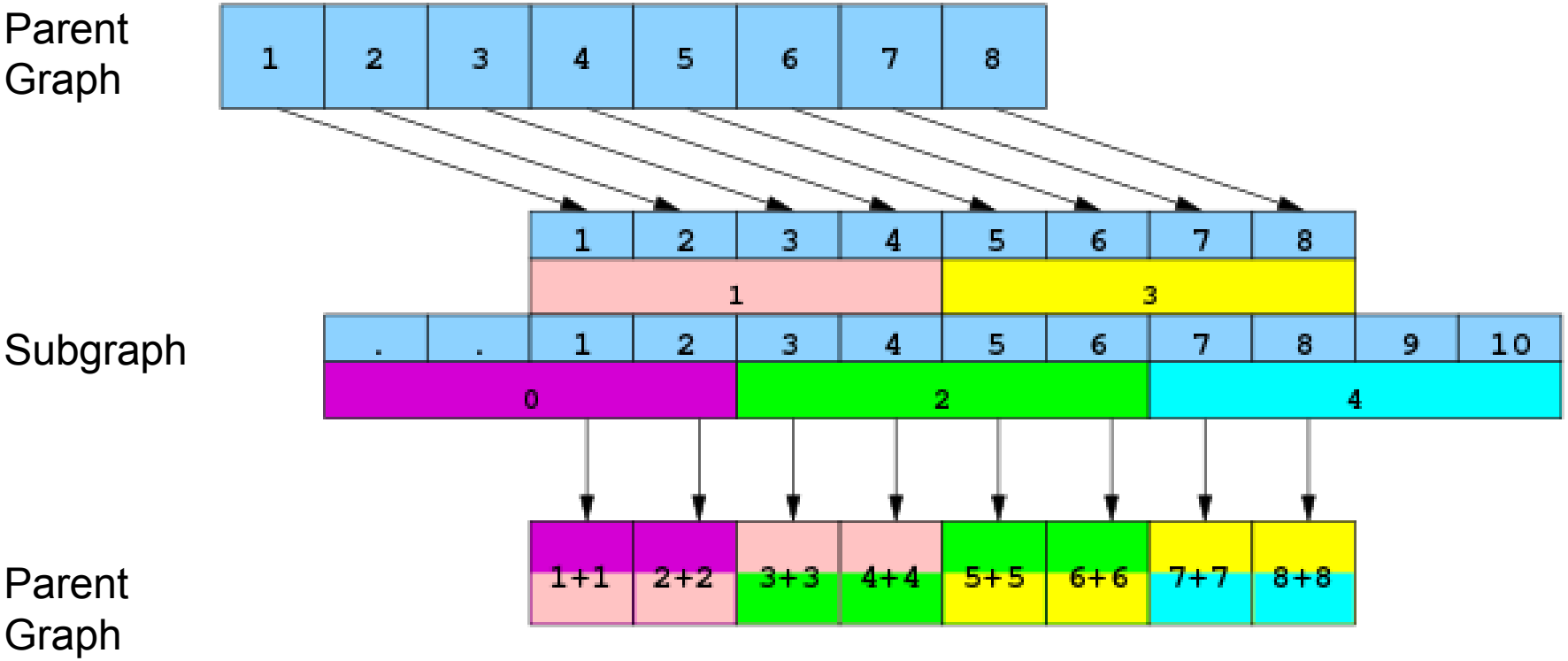
Parent Graph

Subgraph

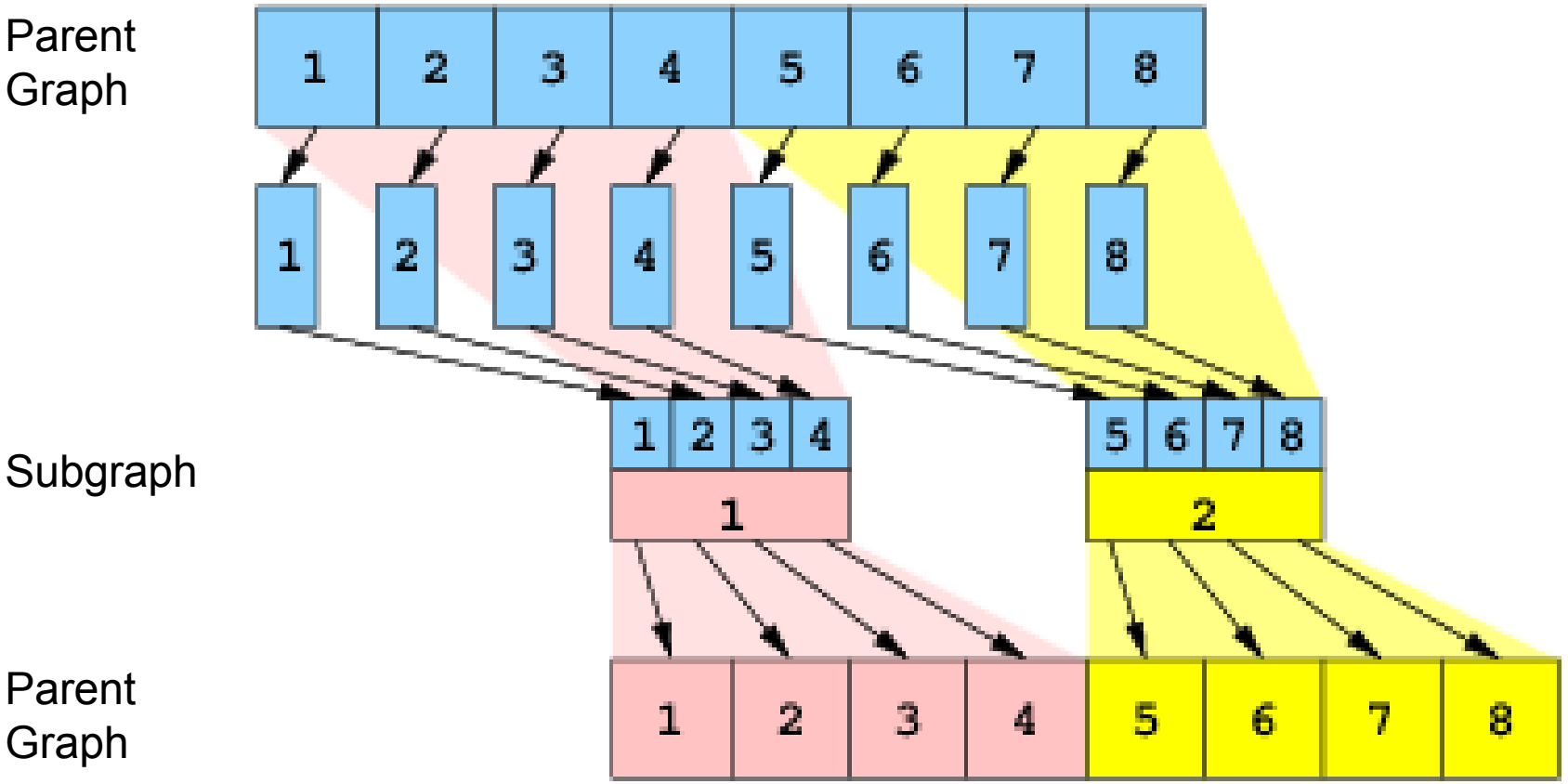
Parent Graph



Re-Blocking + Overlap



Re-Blocking + Downsampling



Event-orientierte Systeme

- asynchrone Abarbeitung
  - „beim Auftreten“
- Scheduling: Interrupt
  - mit Unterbrechen der aktuellen Berechnung
  - (oder über FIFO)
- Scheduling: Polling
  - Events werden nur zu gewissen Zeiten aus FIFO abgeholt

Event- & sychrone Systeme

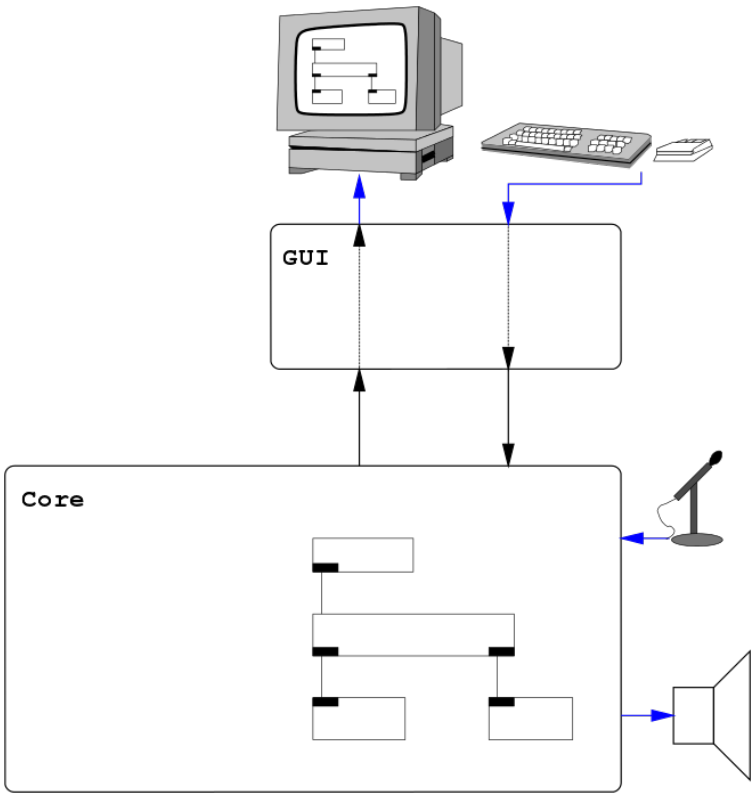
- Kommunikation
  - Serialisierung
    - Abarbeiten von Events zwischen den Synchronen Ticks
  - Parallelisierung
    - Kapselung von Zuständen der beiden Systeme
    - Datenaustausch an Thread barriers
    - IPC
      - *shared memory*
      - *lock-free*

Echtzeit-Systeme

- Pure Data
  - Architektur
- SuperCollider
  - Architektur
  - Language ↔ Synth
- ChucK

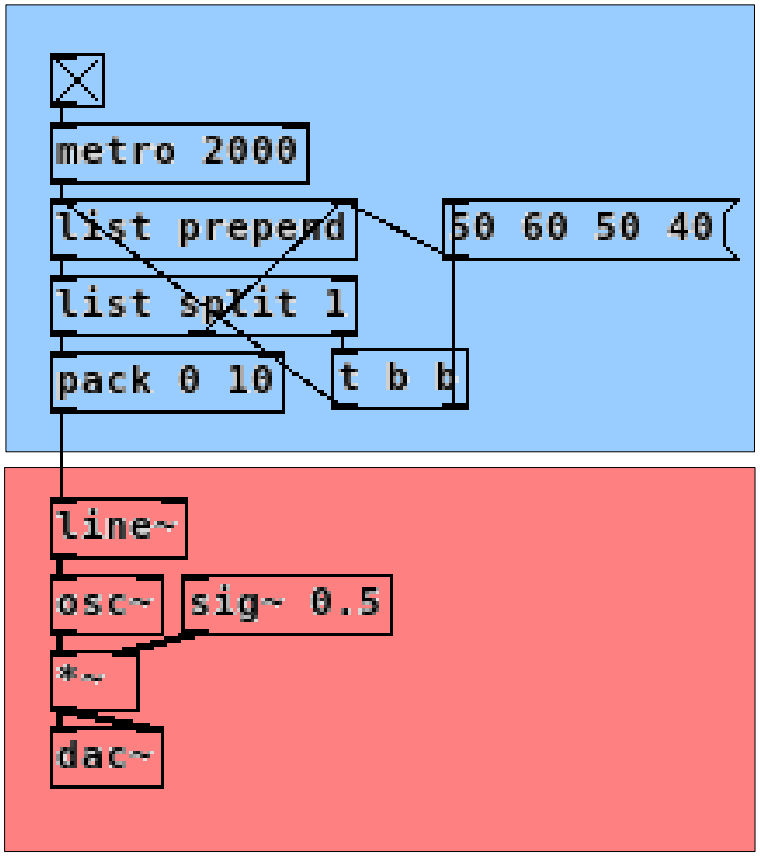
Pure Data: Architektur

- Core-Prozess
  - Dataflow-Graphen Verwaltung
  - Audio und Event-Verarbeitung
  - gesamte Programmlogik
- GUI-Prozess
  - „Slave“
    - Kommunikation über TCP/IP
  - Pd→GUI
    - Zeichenanweisungen
  - GUI→Pd
    - Userinput



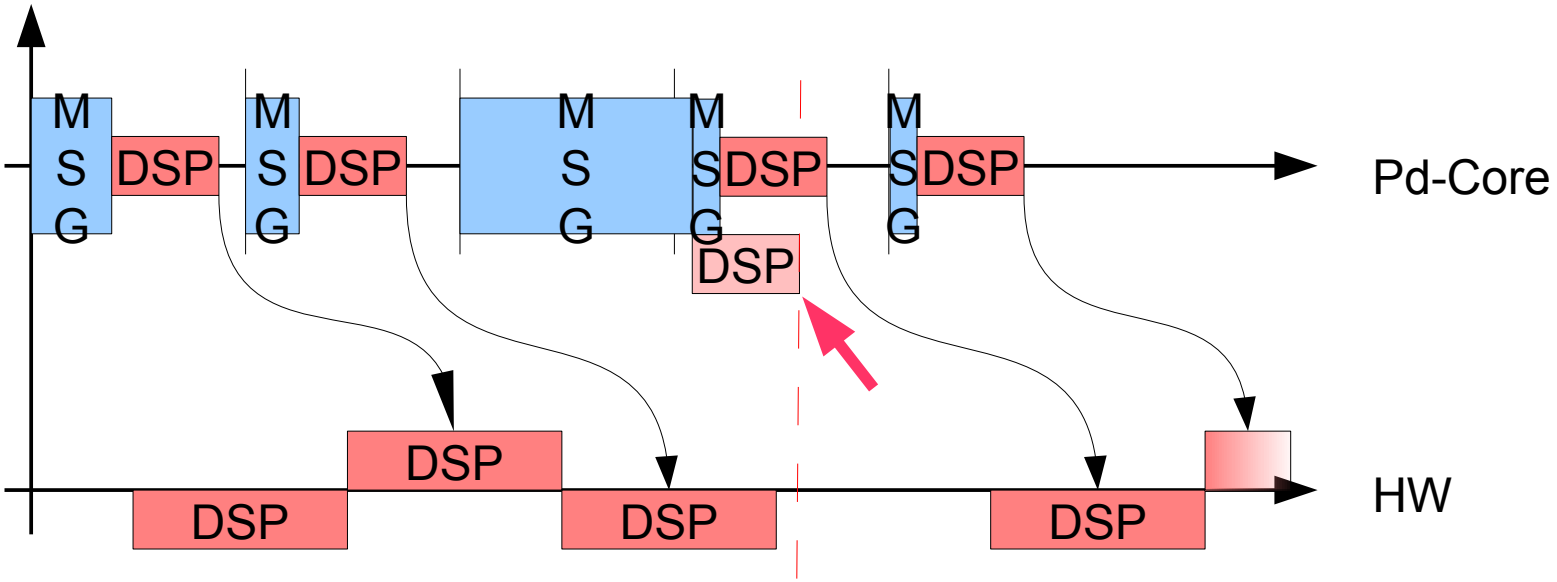
Pd: Patcher

- Dataflow-Graph == Programm
- Metapher
  - auf asynchronen Dataflow erweitert
- Tokens
  - Signale (Zahlenarrays)
  - Listen
    - Zahlen
    - Symbole (Strings)
    - ...



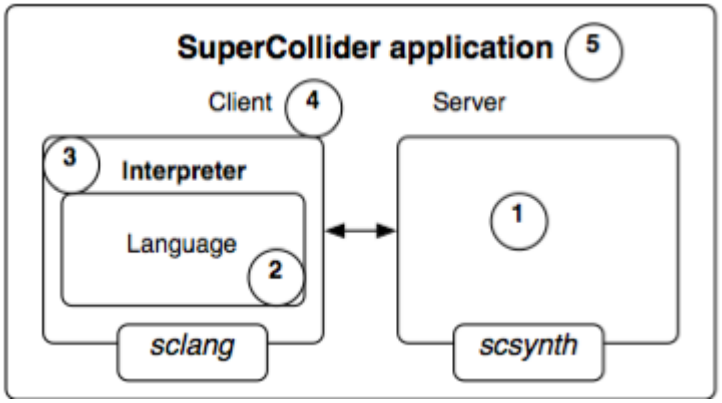
Pd: Scheduler

- Loop
  - Events (unbekannte Dauer)
  - DSP (rel. konstante Dauer)
  - sleep



SuperCollider: Architektur

- Client-Server Modell
- Server
  - Audio-Processing
  - UGen-Netzwerk
- Client
  - „SC-Language“
  - steuert Server
    - einfügen neuer UGens
    - Kontroll-Daten



(1)Audio Server
(2)Audio Programming Language
(3)Language Interpreter
(4)Interpreter Programm as Client
(5)Application

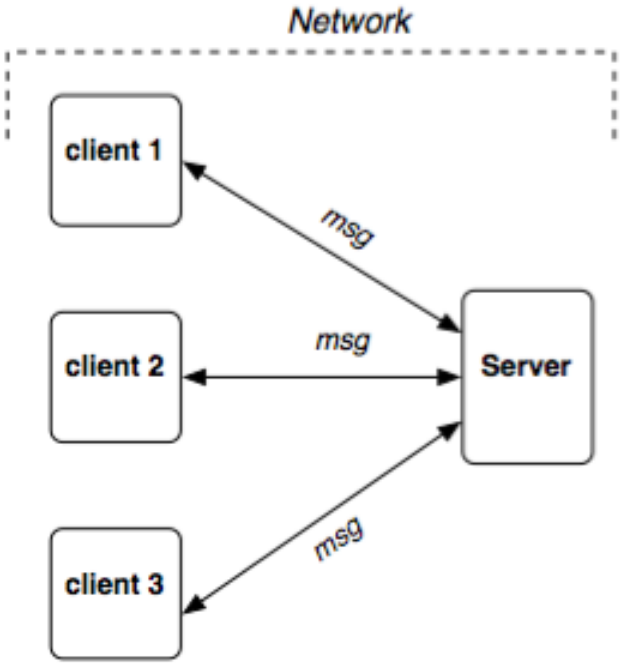
SC3: Architektur

PROs

- Lastverteilung
- Stabilität
  - gegenüber Absturz
- Modularität
  - mehrere Clients
  - alternative Clients
- örtlich ungebunden

CONs

- Latenz
- Asychronizität
  - nicht-deterministisch



SC3: Kommunikation

- OpenSoundControl
  - neuer UGen
    - /s_new 'synthname' <id> ...
  - Kontroll-Daten
    - /n_set <id> 'parm' <value>
  - Kontroll-Bus
    - /c_set <busID> <value>

- Datentypen
  - (Strings)
  - Zahlen [Arrays]

SC3: scsynth

- Audio Server
- verwaltet Dataflow-Graphen
- zusätzlich
  - Signal-Busse
  - Kontrol-Busse

SC3: sclang

- SmallTalk-basierte Sprache
- Objektorientiert
  - „alles ist ein Objekt“
  - Self-Contained
- Read-Eval-Print Loop (REPL)

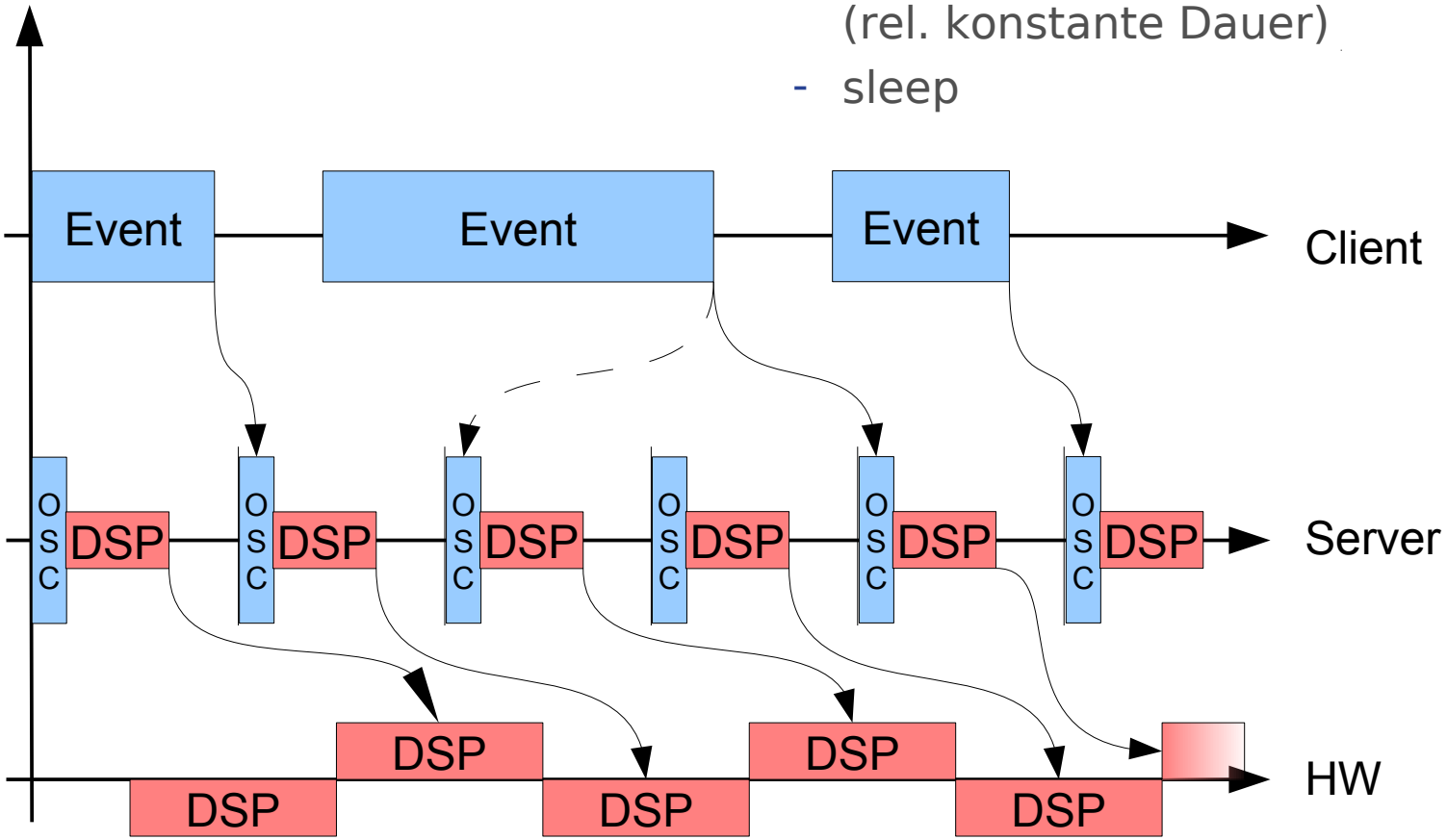
- eigene (parallele) Zeitverwaltung

SC3: Scheduler

- Server-Loop
  - OSC-receiver
    (minimale konst. Dauer)
  - DSP
    (rel. konstante Dauer)
  - sleep

SC3: Scheduler (vereinfacht)

- Client-Loop (REPL)
  - Events (unbekannte Dauer)
- Server-Loop
  - OSC-receiver (minimale konst. Dauer)
  - DSP (rel. konstante Dauer)
  - sleep



intro to chucK

What is chucK?

- general-purpose audio programming language

- strongly typed

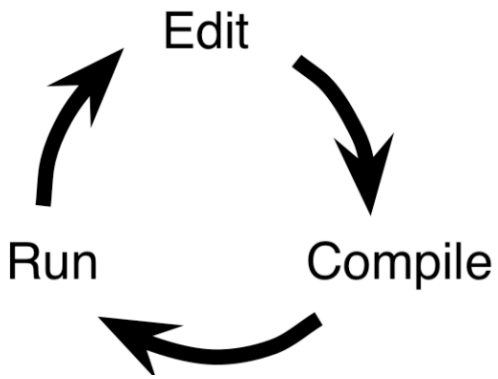
- strongly timed

- object oriented

- platform independent

on-the-fly programming

editing and compiling during runtime



syntax

chuck operator => direction of data flow

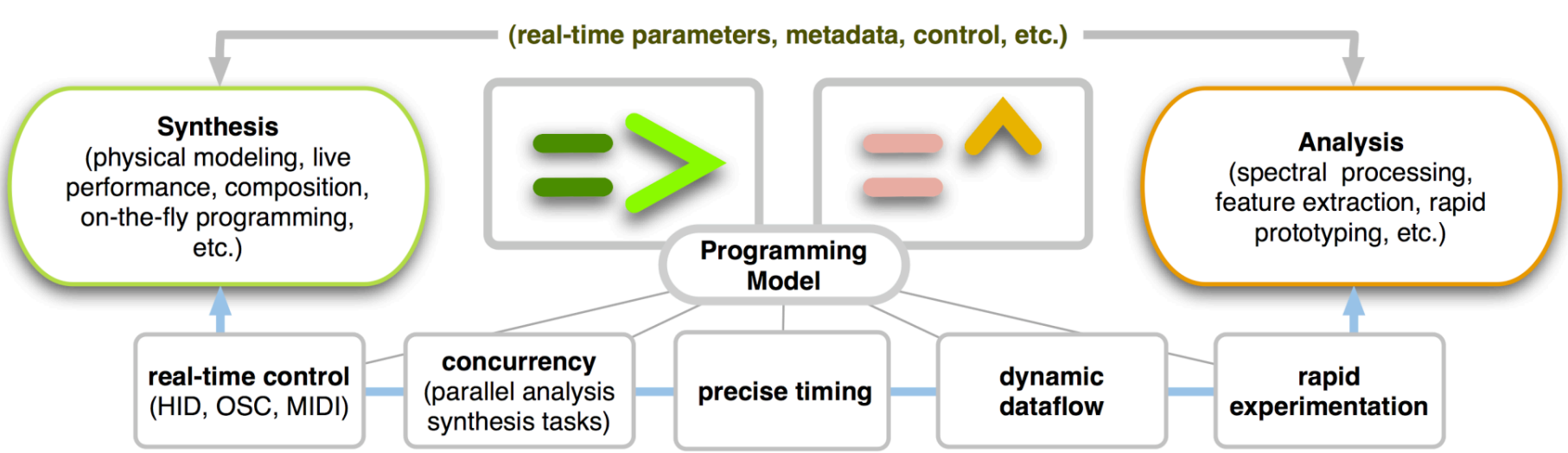
upchuck

- (Windowing, FFT, IFFT, DCT, IDCT)

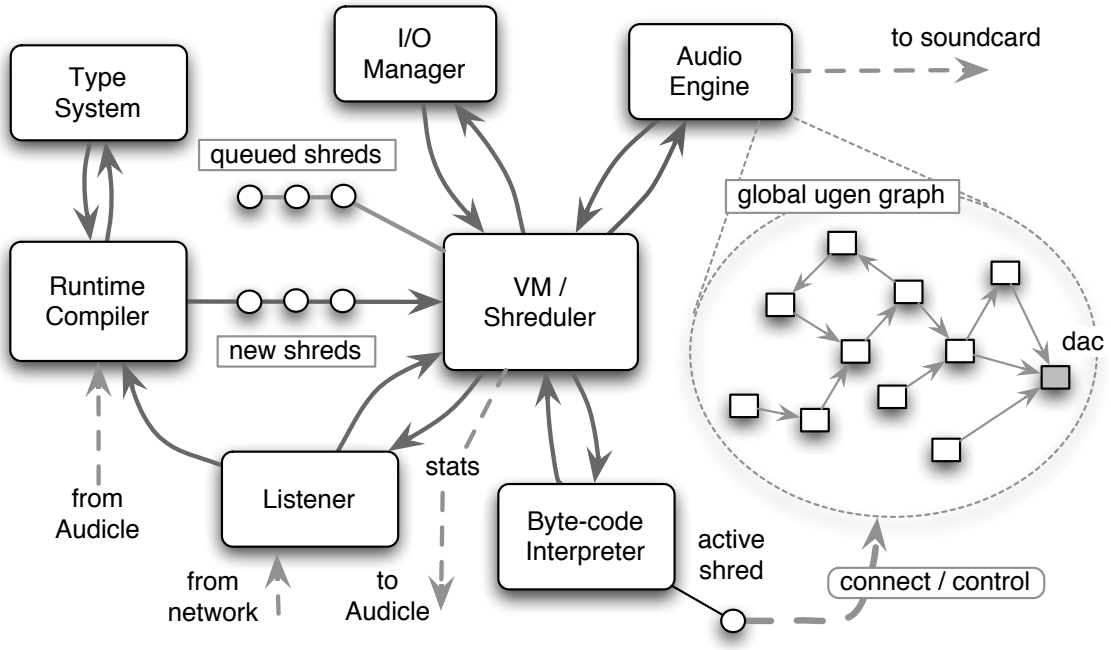
- data passed between UAnae is not audio samples

- the relationship of UAna computation to time fundamentally different (computes on blocks of sample, or on metadata)

combining analysis and synthesis in chucK



chucK architecture



passage of time is data-driven, and this guarantees that the timing in the shreds is bound to the audio output and not to any other clocks.

some chucK facts

- chucK does not sleep. He waits.

- chucK doesn't wear a watch, HE decides what time it is

- chucK has two speeds. Walk, and Kill.

- chucK can divide by zero.

Instrumentalmusik und Live-Elektronik

LVA#17.0078