

# Multimedia 2012

Konrad Froitzheim

# Programm

## 0. Medien

### 1. Algorithmen

- Daten
- Medien

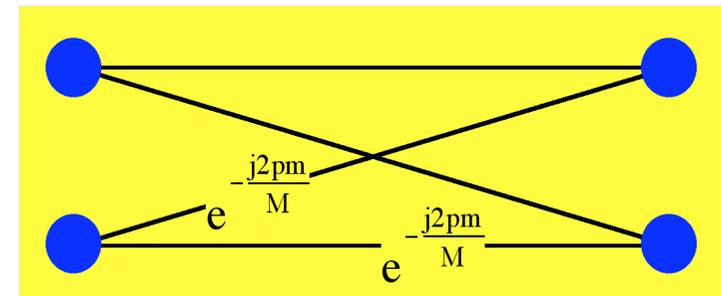
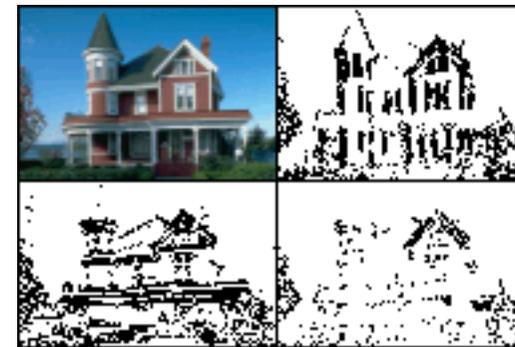
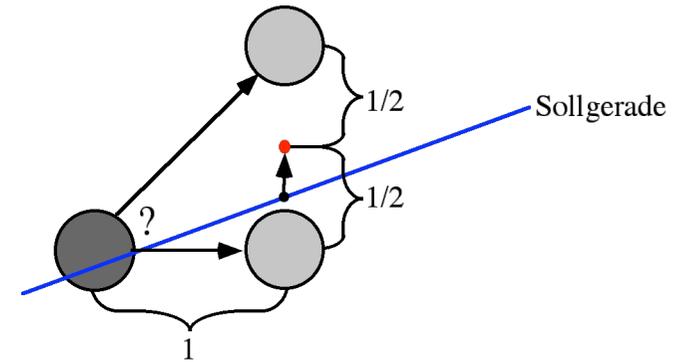
### 2. Hardware II

- Prozessorerweiterungen
- Signalprozessoren
- Bussysteme
- Plattenspeicher

### 3. Quicktime

- Toolbox
- Dateiformat
- Programmieren Movie Toolbox
- Programmieren Component Manager

### 4. Synchronisation



## Literatur

- Apple Computer: Inside Quicktime.
- Chapman, Chapman: Digital Multimedia, 2000.
- Froitzheim, K.: Multimediale Kommunikationsdienste, 1996.
- Halsall, F.: Multimedia Communications, 2001.
- JPEG, MPEG, H.261; Comm. of the ACM, April 1991.
- Koegel Buford, J.F.: Multimedia Systems, Addison Wesley, 1994.
- Li, Z., Drew, M.: Fundamentals of Multimedia, 2004.
- Marven, C., Ewers, G.: Digital Signal Processing, 1993.
- Multimedia in the Workplace; Comm. of the ACM, January 1993.
- Rabbani, M., Jones, P.: Digital Image Compression Techniques, 1991.
- Steinmetz, Nahrstedt: Multimedia - Computing, Comm., App. 1995.
- Steinmetz, R.: Multimedia - Technologie, Springer, 1993.

# Formales

## Termine:

Vorlesung: Dienstag, 11:00-12:30, Woche 2, 3409  
Mittwoch, 16:00-17:30, 3409

## Dramatis Personae:

Prof. Dr. Konrad Froitzheim

03731/393939

[frz <at> informatik.tu-freiberg.de](mailto:frz@informatik.tu-freiberg.de)

Mathias Buhr

[mathias.Buhr <at> informatik.tu-freiberg.de](mailto:mathias.Buhr@informatik.tu-freiberg.de)



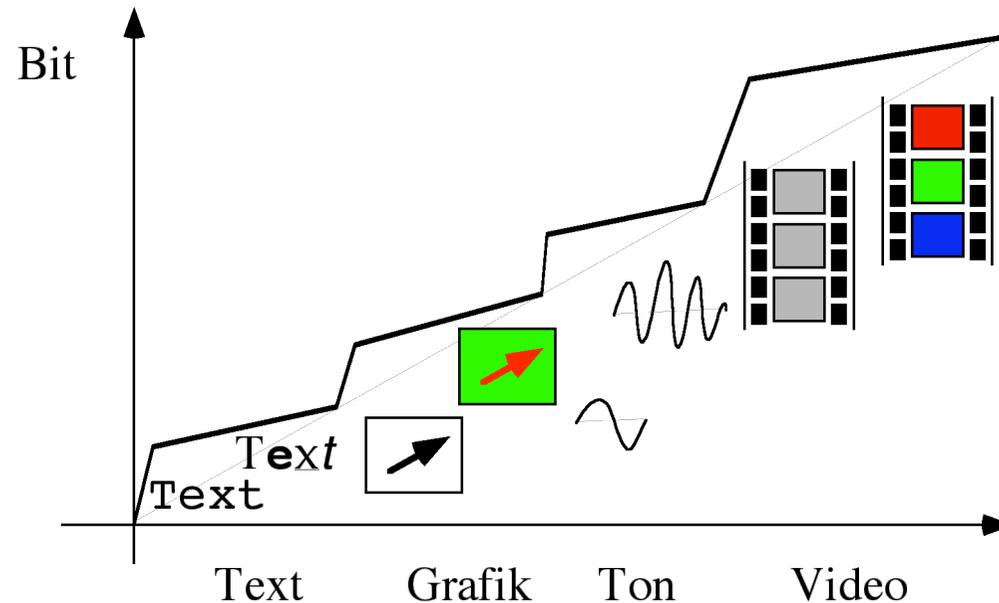
## Vorlesungsarchiv (kein Scriptum):

- <http://ara.informatik.tu-freiberg.de/vorlesungen/MM2012.doc>

Prüfung: Klausur in der vorlesungsfreien Zeit

# 0. Medien

## 0.1 Medien und Wahrnehmung



- Nutzlast (Bit/bit)

- Information wird in Bit gemessen, bit = Anzahl {0,1}

- ASCII-Text            10 byte

- Bitmap                1000 Punkte \* 1 byte

- Telefon                8.000 byte

- Audio-CD             44.100 Samples \* 2 byte \* 2

- Video                 25 Bilder \* 256 Spalten \* 192 Zeilen \* 3 byte/Punkt = 3.686.400 byte

- TV                     25 Bilder \* 704 Spalten \* 625 Zeilen \* 3 byte/Punkt = 33.000.000 byte

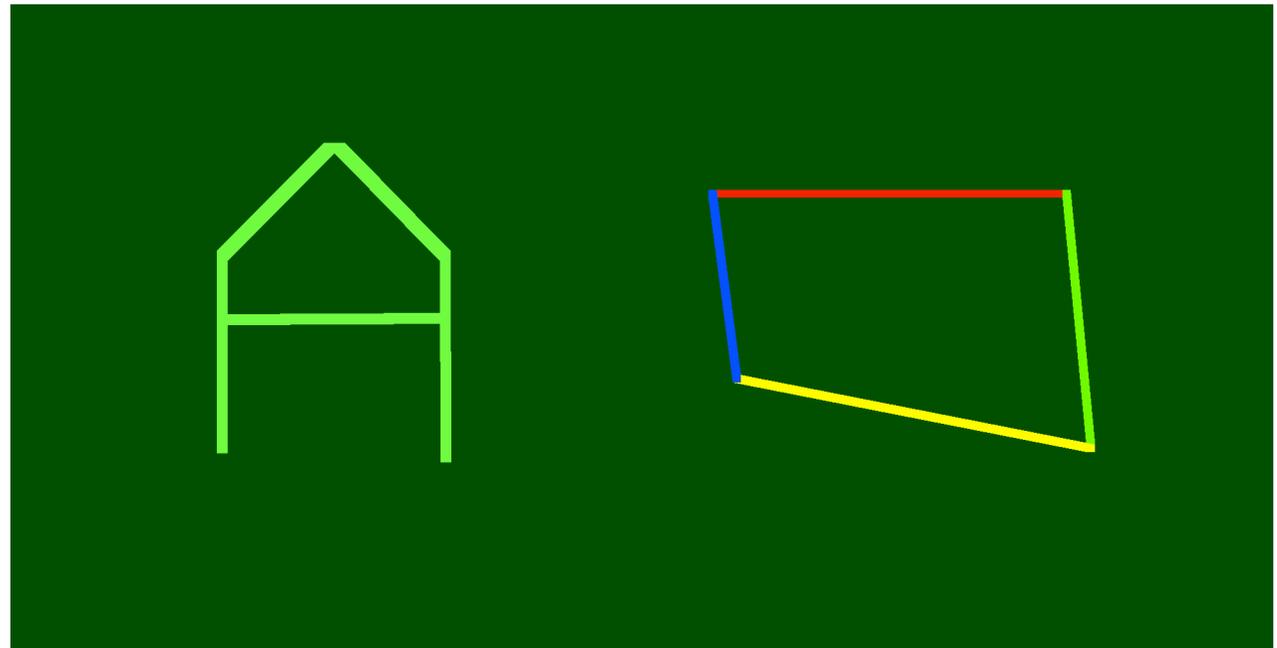
- Aufnahmevermögen und Bitrate
- Dimensionen, räumliche Effekte
  - Menschen haben räumliches Empfinden (Sehen, Hören, Gleichgew.)
  - Raum und Zeit
  - Dimensionen werden vielfältig ausgewertet
- Diskrete und kontinuierliche Medien
  - Klassifikation entsprechend Auflösungsvermögen der Wahrnehmung
  - Im Raum Punkte oder Verläufe: Pixelmaps oder Photographien
  - In der Zeit Stilleben oder Bewegung
    - Grafik oder Animation
    - Bilder oder Video
  - Audio
    - physikalisch immer kontinuierlich
    - Psychisch auch diskret:
      - Sprache oder Musik
- Abschattungseffekte
  - in einem Medium
  - zwischen Medien

Spracherkennung



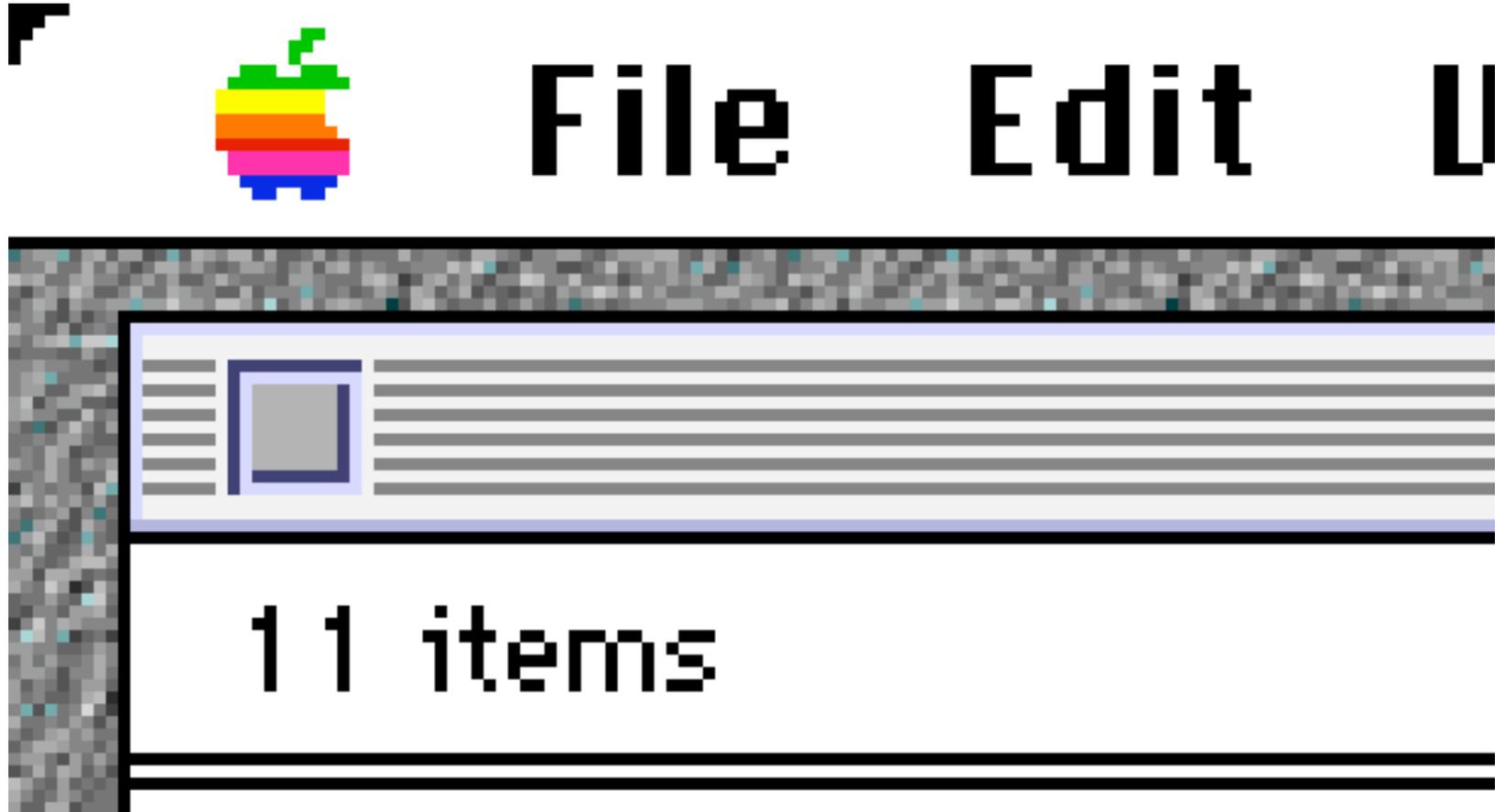
## 0.2 Computergrafik

- Darstellung visueller Objekte
  - Buchstaben und Zahlen,
  - geometrische Objekte (Gerade, Kreis, Rechteck, ...)
  - Attribute (Farbe, Muster, Font, ...).
- Bildspeicher
  - Hauptspeicher oder im Adapter,
  - eventuell mehrere Ebenen (Farbe, Graustufen, räumliche Position).
- Buchstabenbildschirme
- Vektorgrafik
  - alle Objekte aus einzelnen Strecken (Vektoren) aufgebaut



- Rasterbildschirm

- jeder Punkt einzeln ansprechbar
- uneingeschränkt grafikfähig

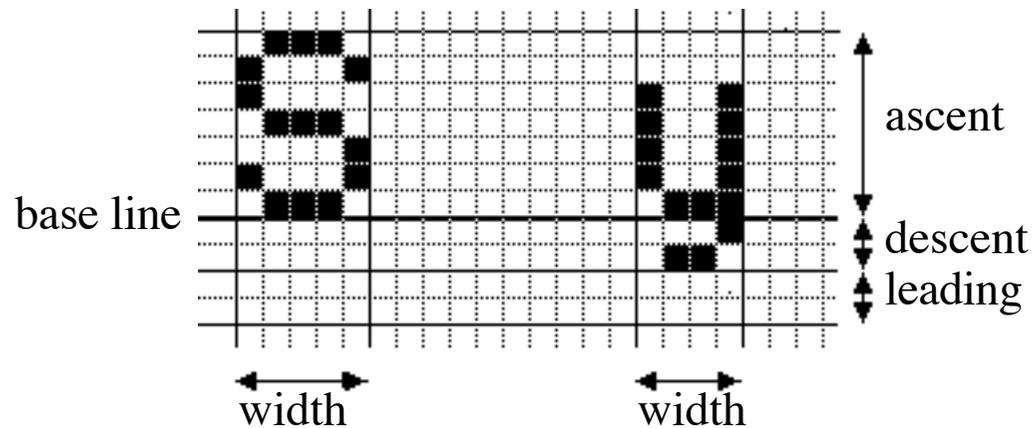


- Punkteanzahl typisch 512\*342 bis 2560\*1600
- 72, 80 bis 100 Punkte/Zoll (dpi)
- 1920\*1080\*24 bit für 24" Farbmonitor -> 6.220.800 Byte

## 0.2.1 Text

- Zeichensatz: kompakte Kodierung für Buchstabe ( $\neq$  Glyph)
  - ASCII: American Standard Code for Information Interchange
    - 0 .. 31 Druckersteuerzeichen
    - 32 .. 127 druckbare Zeichen
    - 128 .. 255 nichtstandardisierte Erweiterungen
  - EBCDIC: Extended Binary Coded Decimal Interchange Code
  - ISO 8859-X
    - Erweiterung von ASCII um länderspezifische Zeichen
    - 1, 2, 3, 4 und 9 für lateinische Zeichensätze, weitere für Euro ...
    - 5 kyrillisch, 6 arabisch, 7 griechisch und 8 hebräisch
  - Unicode und UTF-8, UTF-16, UTF-32
    - Codes für **alle** Schriftzeichen der Welt: 0 .. 1,114,111
    - heute auch alte, nicht mehr verwendete Zeichen
    - mehr Zeichen -> mehr Information/Zeichen
  - UTF-8: variable Länge 7, 11, 16, 21, 26, 31 Bit Codes
    - ASCII: 0xxx xxxx,
    - 11 Bit: 110xxxxx 10xxxxxx
    - 31 Bit: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

- Schriftattribute
  - **fett**, *kursiv*, Umriss, **schattiert**, ...
  - Zeichengröße und -breite
  - Kerning und Ligaturen: fl statt fl
- Fontmetrik
  - beschreibt Laufeigenschaften des Textes
  - monospace vs. proportional

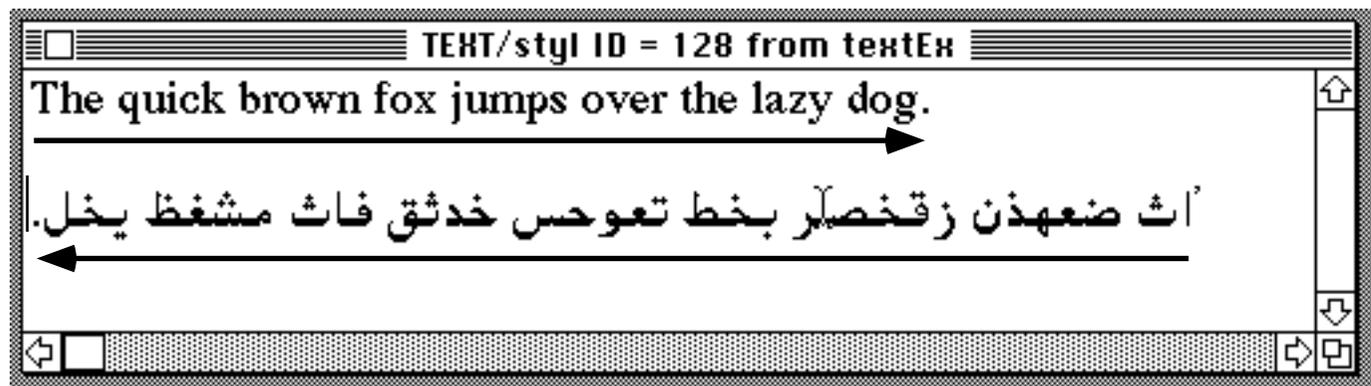


- Fontomania

- tausende verschiedene Zeichensätze
- Font-Beschreibungsalgorithmen siehe Kapitel 3
- Times-Roman
- Helvetica
- Palatino
- Σψμβολ
- ❁❁☐❁ ❁❁■❁❁❁❁▼▲ ,.- (Zapf Dingbats)

- nicht-lateinische Schriften

- andere Fonts
- Hebräisch, Arabisch, Chinesisch, ...
- Schreibrichtung rechts -> links, vertikal



## 0.2.1.1 Zeichendarstellung

- Bitmap-Fonts

- werden entworfen, gezeichnet, gespeichert und fertig verteilt ...
- in verschiedenen Größen (z.B. 6 Punkte bis 127 Punkte)



- und Formen

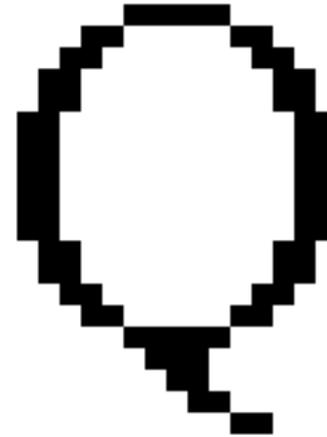
Chicago, Monaco, New York, Geneva

- Zeichensätze liegen in Rasterform auf der Festplatte

"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2
"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2
3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	[	\	]	^	_	`	a	b	c	d	e
U	V	W	X	Y	Z	[	\	]	^	_	`	a	b	c	d	e
f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
w	x	y	z	{		}	~	**	0uA	S0a	S0c	0eE	0nN	0u0	0uU	0ea
W	X	Y	Z	{		}	~		Ä	Å	Ç	É	Ñ	Ö	Ü	á
0`a	0ia	0ua	0na	0a	0c	0ee	0`e	0ie	0ue	0ei	0`i	0ii	0ui	0nn	0eo	0`o
à	â	ã	ä	å	ç	é	è	ê	ë	í	ì	î	ï	ñ	ó	ò
0io	0uo	0no	0eu	0`u	0iu	0uu	0t	S08	04	03	06	08	07	0s	0r	0g
ô	ö	õ	ú	ù	û	ü	†	°	¢	£	§	•	¶	ß	®	©

- Werden bei Bedarf in den Speicher geladen.
- Größe 24 Punkt (Vergrößerung \*8)

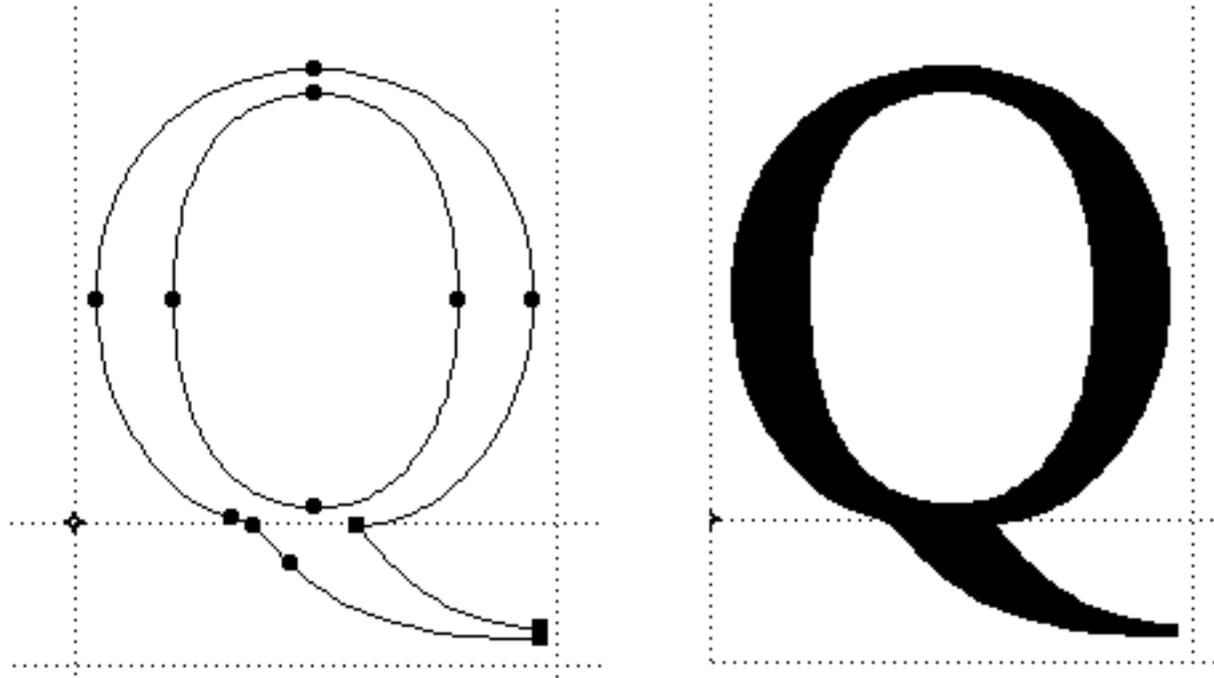
Q



- Auflösungsabhängig, schlecht skalierbar
- Bitmap-Fonts werden bei zunehmender Zeichengröße Speicherfresser
- Bold, Italic, ... müssen separat gespeichert werden

## 0.2.1.2 Kurven zur Beschreibung von Fonts

- Die Umrisse der Zeichen werden als Kurvenzug angegeben
- Zur Darstellung wird dieser Kurvenzug ausgefüllt

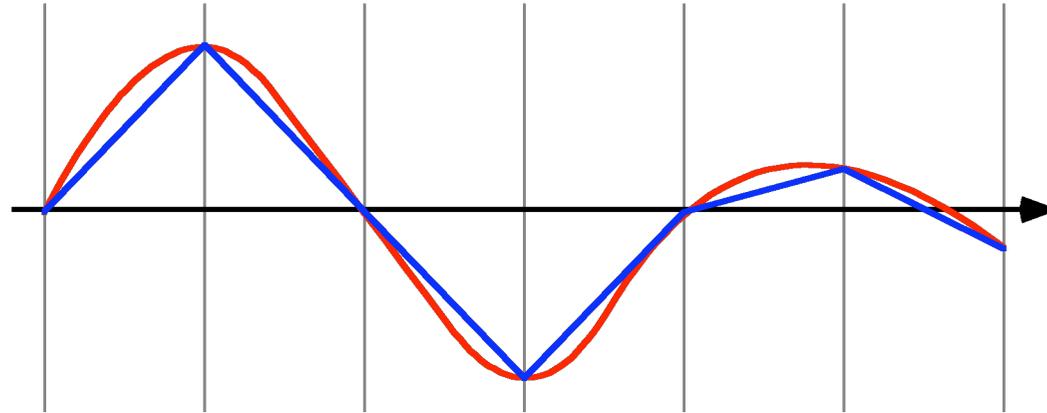


- unabhängig vom Koordinatensystem
  - affine Invarianz
  - möglichst einfach berechenbar
- > Stützpunkte und Interpolation

- Ähnlich Interpolation und Approximation mit Splines

- stückweise linear:  $f_i(x) = a_i x + b_i$

- An den Stützpunkten stetig:  $f_i(x) = f_{i+1}(x)$



- stückweise kubisch:  $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$

- An den Stützpunkten:

- a) stetig:  $f_i(x_k) = s_k, f_i(x_{k+1}) = s_{k+1} \Rightarrow 2n$  Gleichungen

- b) 'glatt':  $f'_i(x) = f'_{i+1}(x) \Rightarrow 2(n-1)$  Gleichungen

- $\Rightarrow$  Gleichungssystem  $4n$  Unbekannte,  $2n + 2n - 2$  Gleichungen

- je nach Randbedingungen versch. Approximationseigenschaften

## • Bézier-Kurven

### - Beispiel 2. Ordnung

gegeben  $b_0, b_1, b_2$

$$b_0^1(t) = (1-t)b_0 + t b_1$$

$$b_1^1(t) = (1-t)b_1 + t b_2$$

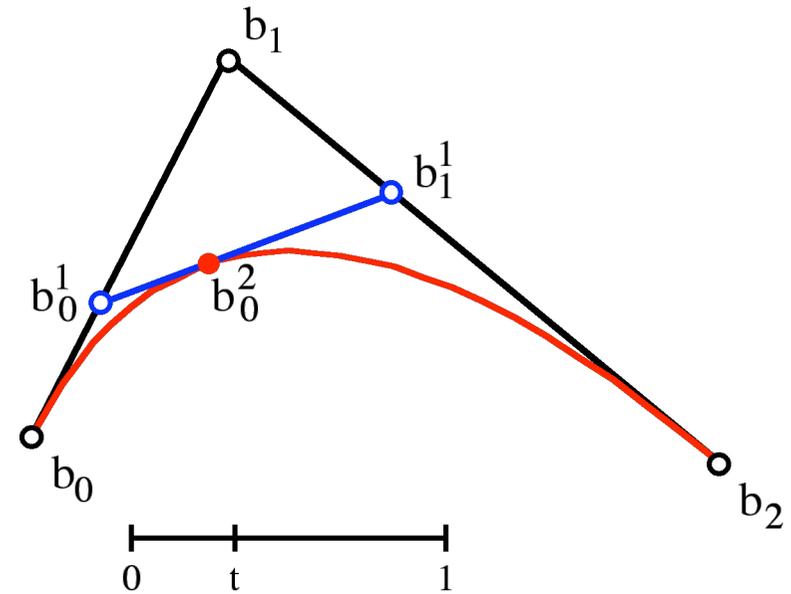
$$b_0^2(t) = (1-t)b_0^1(t) + t b_1^1(t)$$

### - Algorithmus von de Casteljau

gegeben  $b_0, b_1, \dots, b_n$

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + t b_{i+1}^{r-1}(t)$$

$$r = 1, \dots, n; i = 0, \dots, n-r$$



$$B_i^n = \binom{n}{i} t^i (1-t)^{n-i}$$

$$b_0^n = \sum_{j=0}^n b_j B_j^n(t)$$



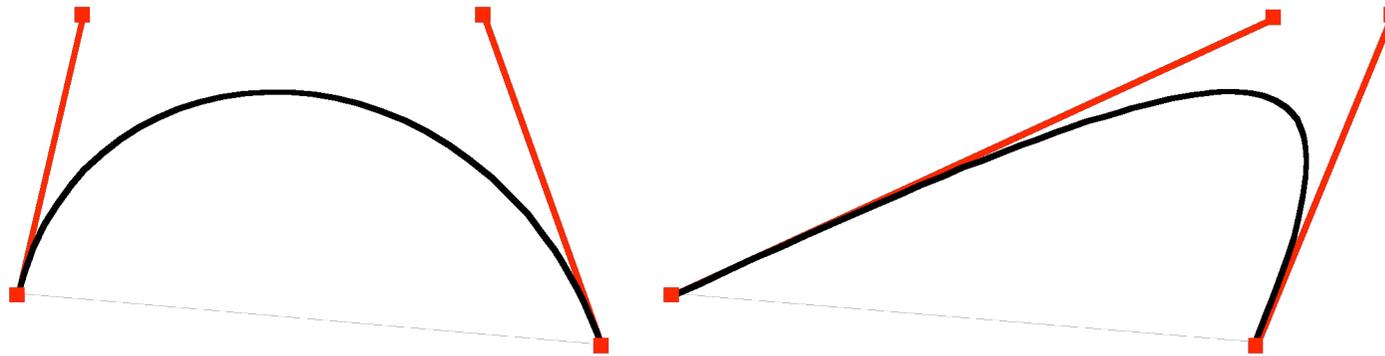
- Explizite Darstellung mit Bernsteinpolynomen

- Bézier-Kurven 3. Ordnung

Kontrollpolygon durch vier Punkte:

Anfangspunkt ( $b_0$ ) und Endpunkt ( $b_3$ )

2 Kontrollpunkte ( $b_1, b_2$ )

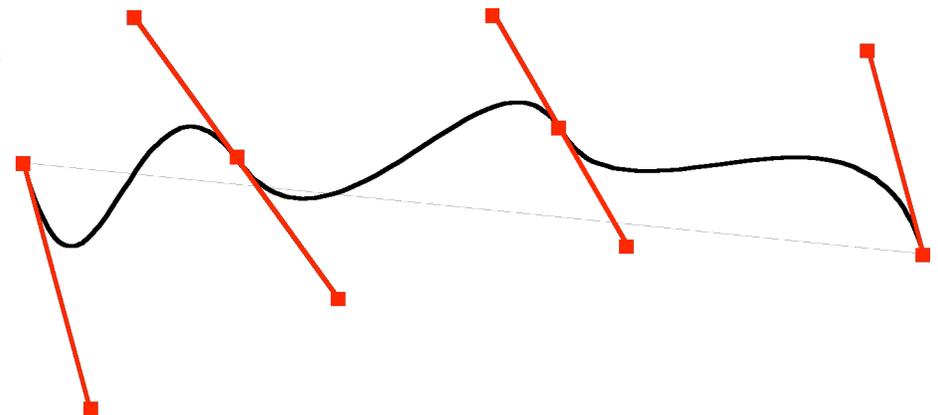


• zusammengesetzte Kurve

- mehrere Bézier-Splines zur Darstellung einer Kurve

- Interpolationseigenschaft

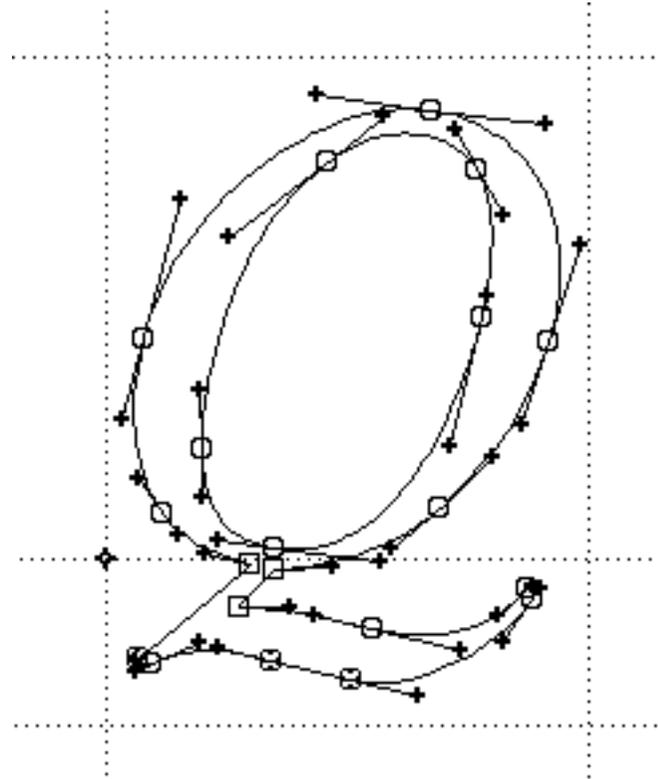
- Kontrollpunkte so legen, daß die Kurve  
glatt wird





- PostScript Type-1 Fonts

- Fontparameter
- Zeichenparameter
- Bézier Kurven zur Beschreibung des Umrisses
- 'Hints' zur Detailverbesserung



- TrueType oder andere Outline-Fonts benutzen ähnliche Kurven

## 0.2.2 Geometrische Objekte

- geometrische Figuren

- Strecken:



- Rechtecke:



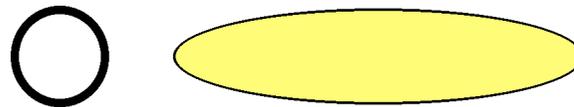
- Rechtecke mit abgerundeten Ecken:



- Polygone und geglättete Polygone:



- Kreise und Ellipsen:

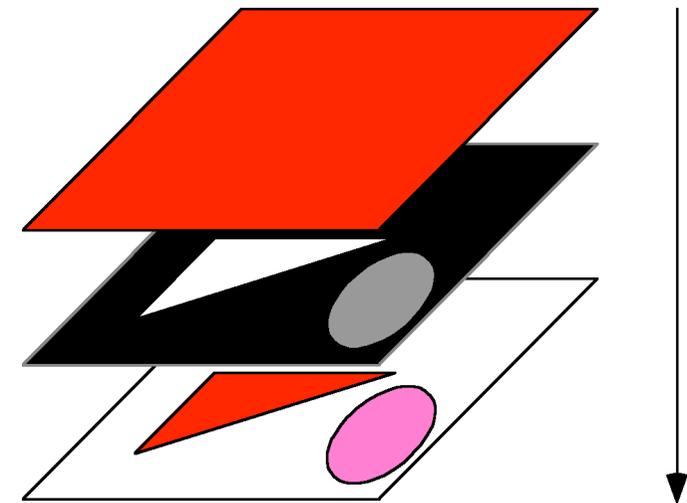


- Polygone  $\neq$  Rechtecke

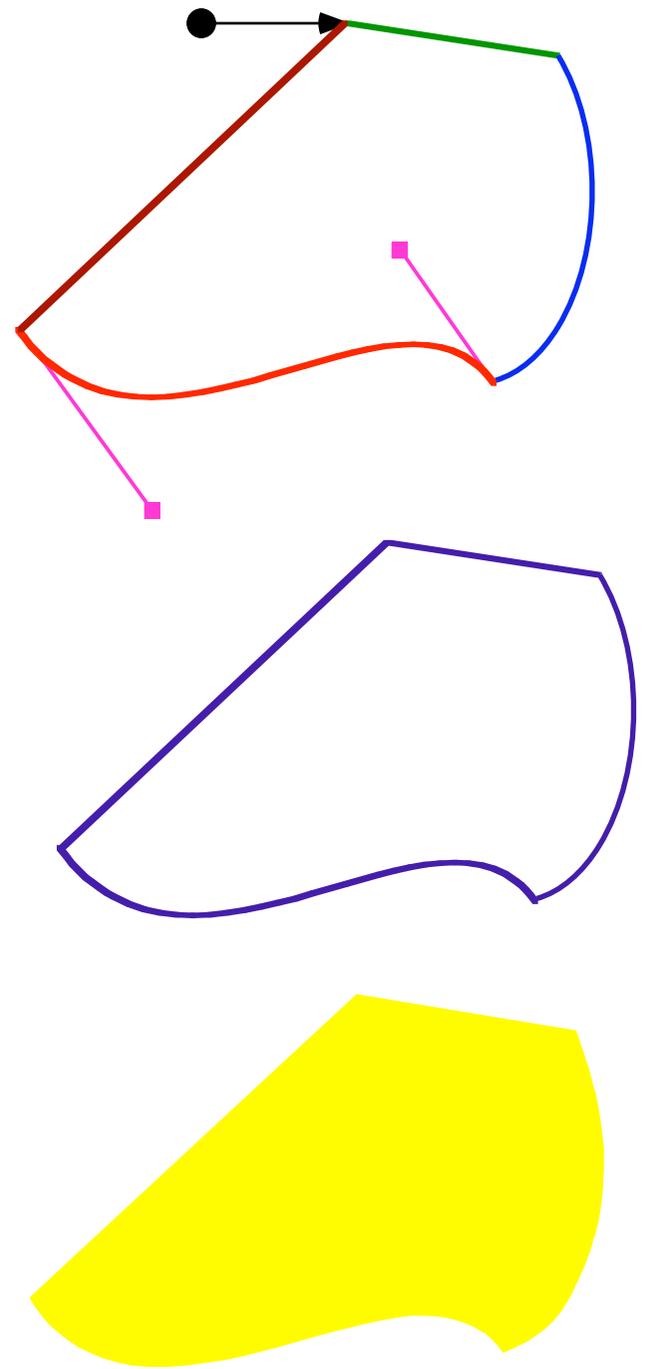
- Begrenzte Anzahl geometrische Objekte vs. Semantikerhaltung

- > effiziente Implementierung

- Beispiel Cairo
  - 2D Graphik-Bibliothek
  - mehrere Zielgrafiksysteme (X, Win32, Postscript, PDF, Quartz, ...)
  - "auflösungsunabhängig"
  - <http://www.tortall.net/mu/wiki/CairoTutorial>
- Koordinatensystem
  - Ursprung links oben
  - Koordinateneinheiten ~ Pixelgröße
- Grafikkontext
  - Zustand des Zeichensystemes
  - Stiftdicke, -farbe, -muster, und -position,
  - Füllfarbe und -muster,
  - Textattribute (Font, Typeface, Größe, ...)
- Farbtransfer
  - Quelle der Grafikoperationen: Farben, Muster, ...
  - Maske kontrolliert Transfer von Quelle nach Ziel
  - Ziel der Grafikoperationen: Fläche (surface)

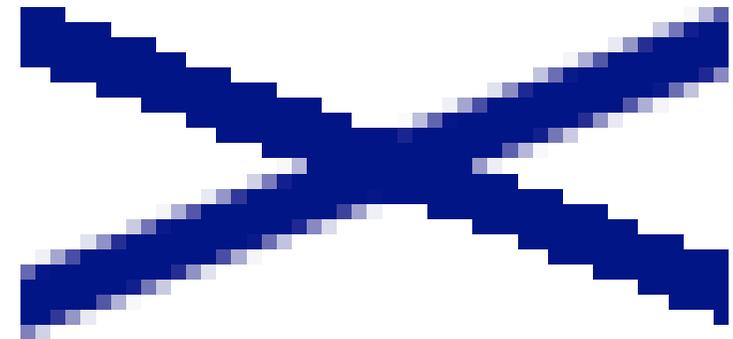
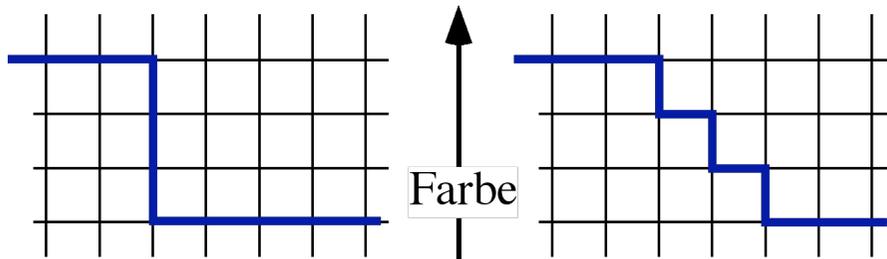


- Pfad
  - Kette von kubischen Bézier Splines
  - offen oder geschlossen
  - durch Pfadoperationen erzeugt
- Pfadoperationen (path verbs)
  - `move_to`
  - `line_to(x,y)`: Strecke vom aktuellen Punkt zu x,y
  - `arc(mp,r,w1,w2)` : Kreisbogen
  - `curve_to(kp1,kp2,ep)`: Bézier Spline explizit
  - `close_path`
- Objekte zeichnen: Pfad färben oder füllen
  - `stroke`: Stift bewegt sich entlang des Pfades
  - `fill`: malt vom Pfad umschlossenes Gebiet aus
  - im Hintergrund wird Maske erstellt
- Text
  - `text_path("etwas Text")`: string->Pfad
  - `show_text("blibliablubb")`: `text_path` + `fill`
  - natürlich mit Fonts: `select_font_face()`, ...



## 0.2.3 Aliasing

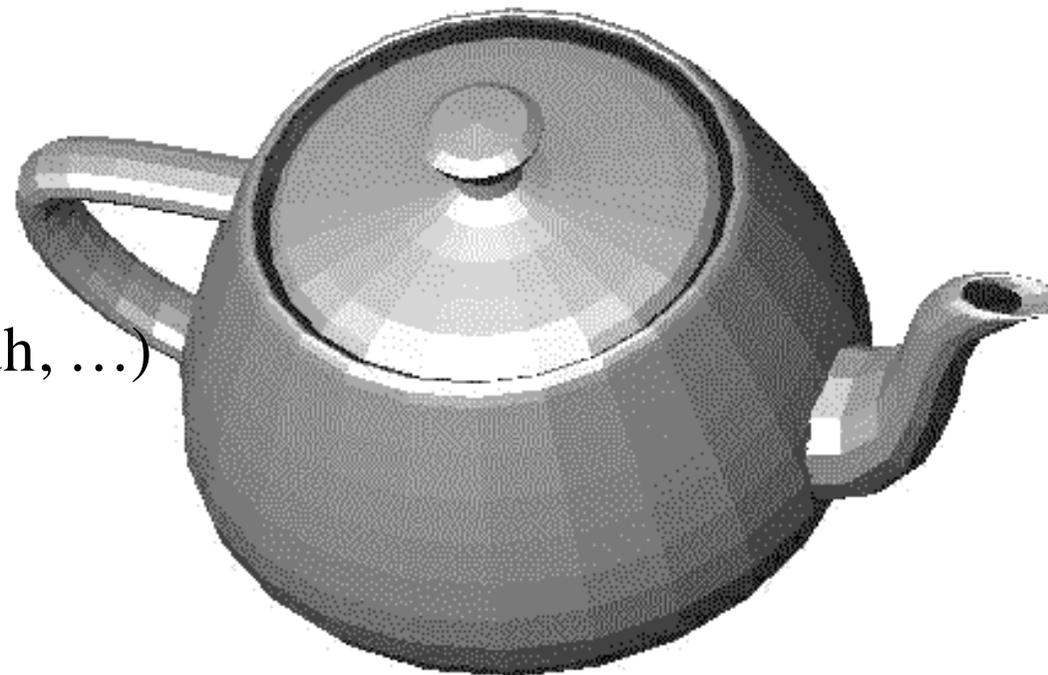
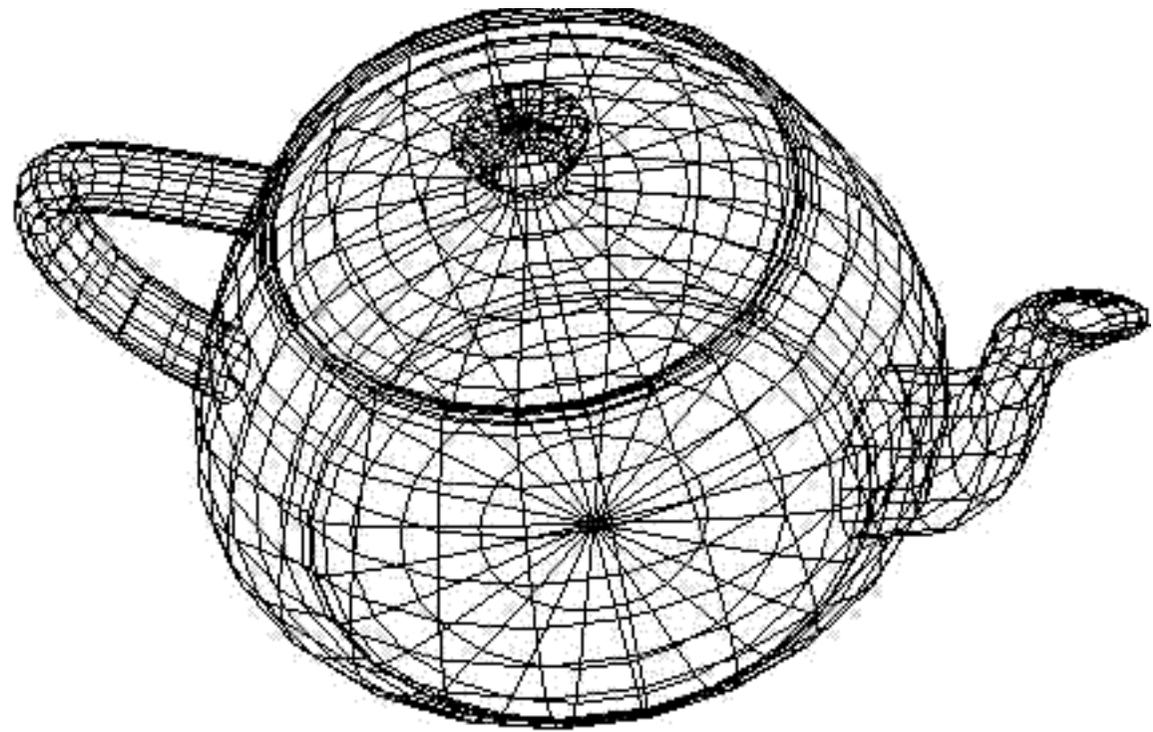
- Aliasing und grafische Objekte
  - scharfe Kanten -> hohe Frequenz
  - (Unter-)Abtastung durch Pixeldarstellung
- Antialiasing
  - Kanten glätten
  - im Grafikprogramm



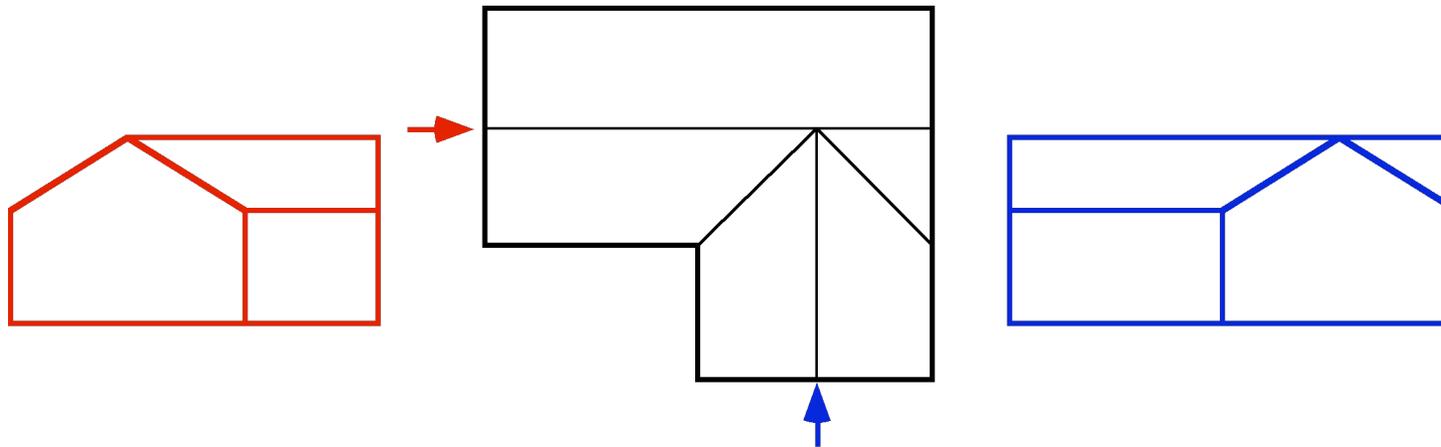
- Filter
- Auch bei Fonts
  - Adobe Type Manager
  - Systemkomponenten

## 0.2.4 3-D Grafik

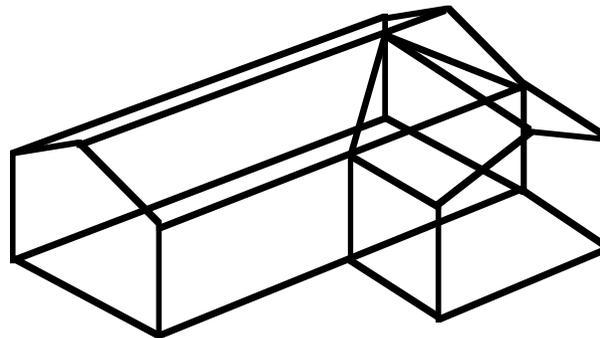
- Modellieren
  - Topologie und Geometrie
  - geometrische Objekte erzeugen und anordnen
  - Attribute festlegen (Glanz, Farbe, Durchsichtigkeit)
  - Texturen bestimmen
  - Lichtquellen anordnen
- Rendering
  - Kameratyp und -position
  - Renderer wählen
  - Abbild berechnen
- Interagieren
  - Zeigemittel (Spacemouse, Handschuh, ...)
  - Auswählen (picking)
  - Navigieren



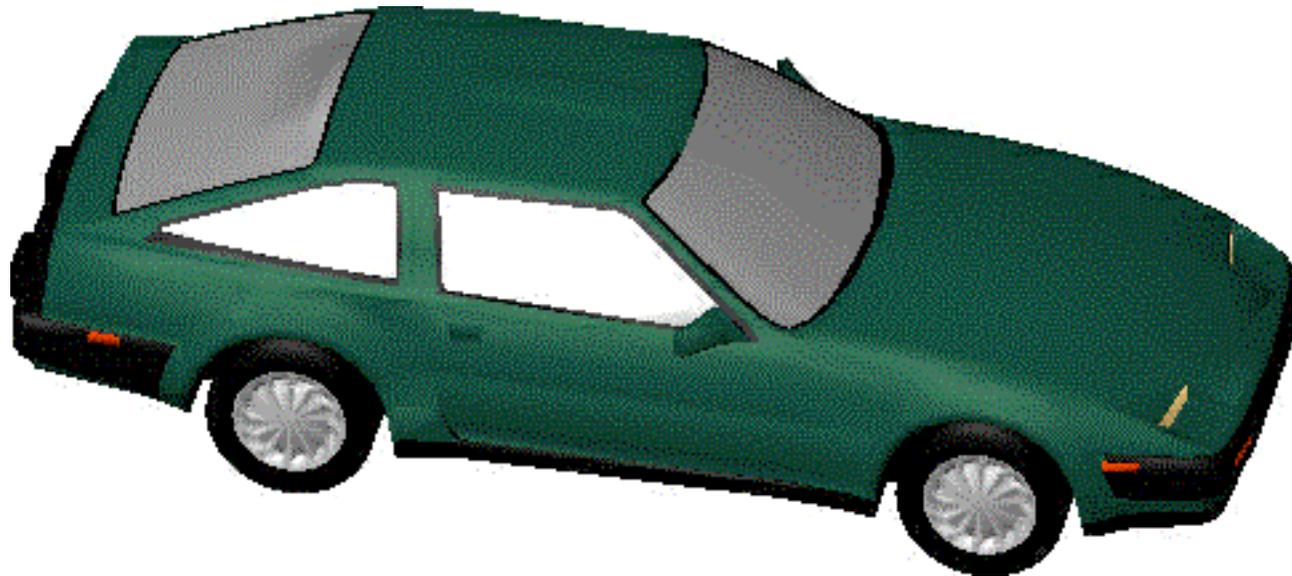
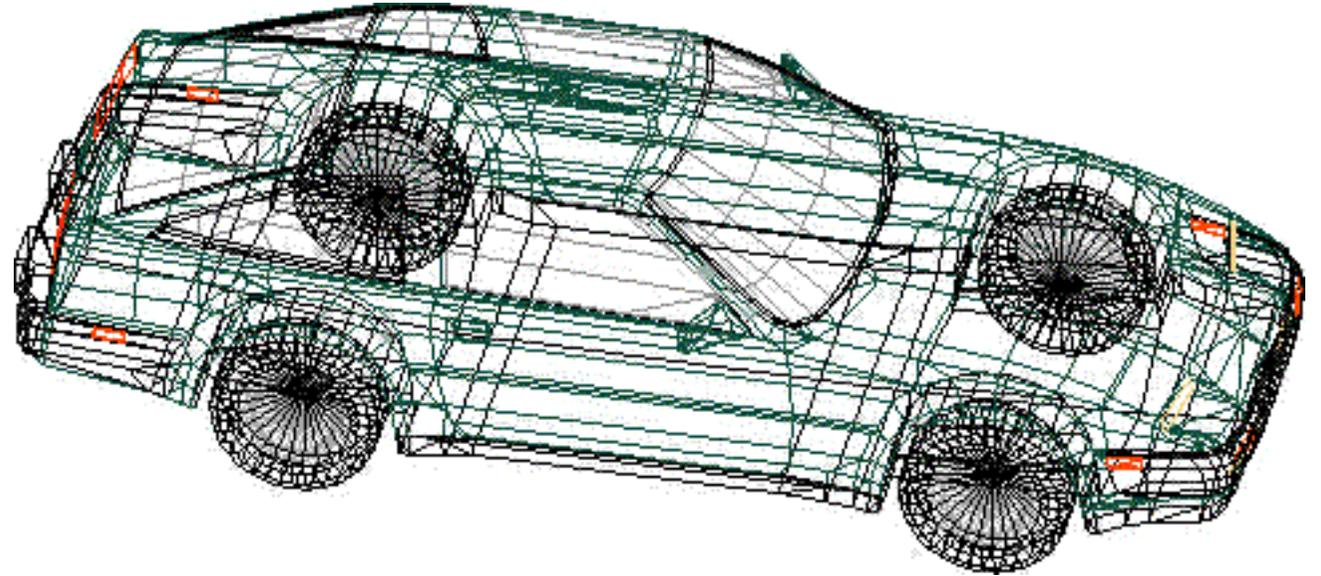
- Präsentation meist zweidimensional
  - Leinwand, Bildschirm, Papier
  - Projektion von 3-D Szenen auf 2-D Ebene
  - Tiefenhinweise gehen teilweise verloren
- Ansicht und Aufsicht



- Projektion und Drahtmodell

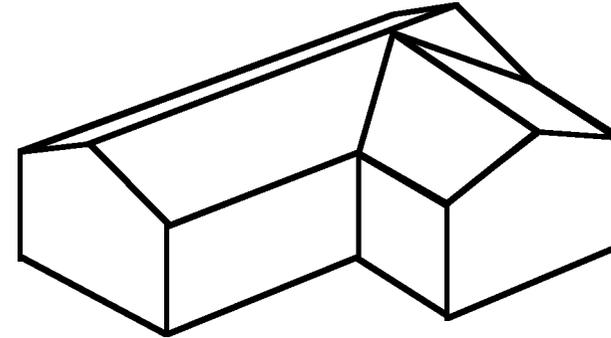
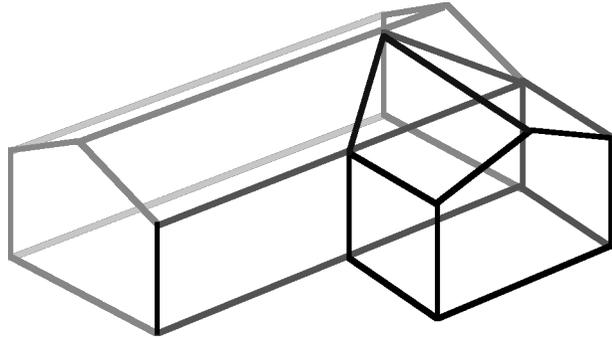


- Drahtmodell (wire frame)
  - (fast) ohne Tiefeneindruck



- Depth-Cueing

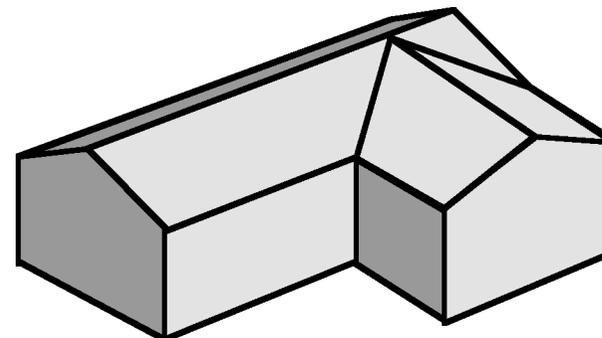
- Linien 'vorne' hervorheben



- Animation: Drehen um eine Achse
- Entfernen verdeckter Linien

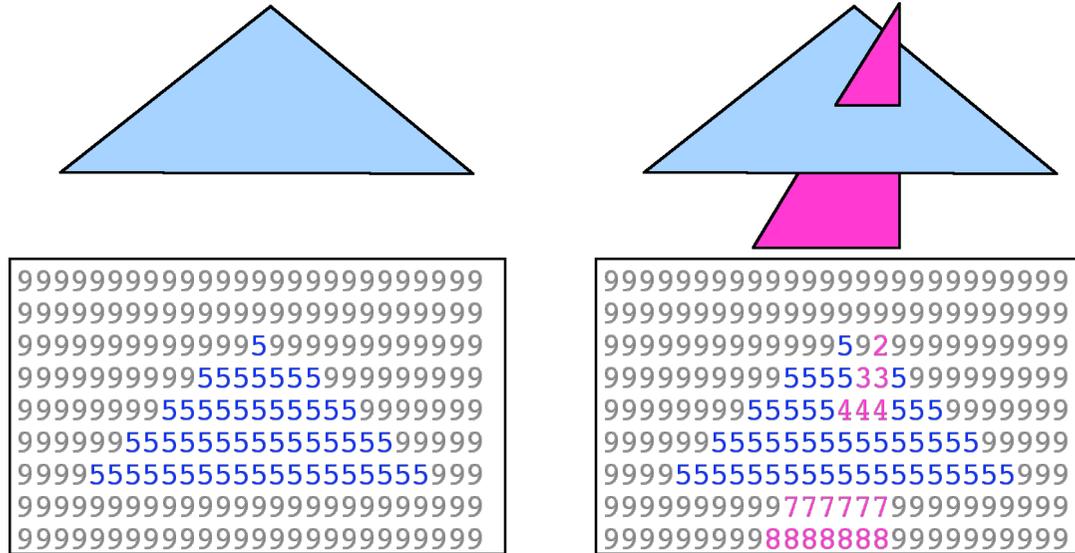
- Verbesserung der Darstellung

- Füllen der Flächen

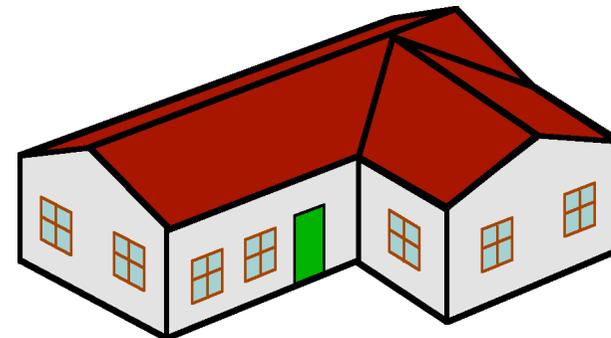


- Entfernen verdeckter Flächen (siehe auch J. Warnock)
- Tiefenpufferalgorithmus (z-buffer) von Ed Catmull
- Pixel = (R, G, B, Z)

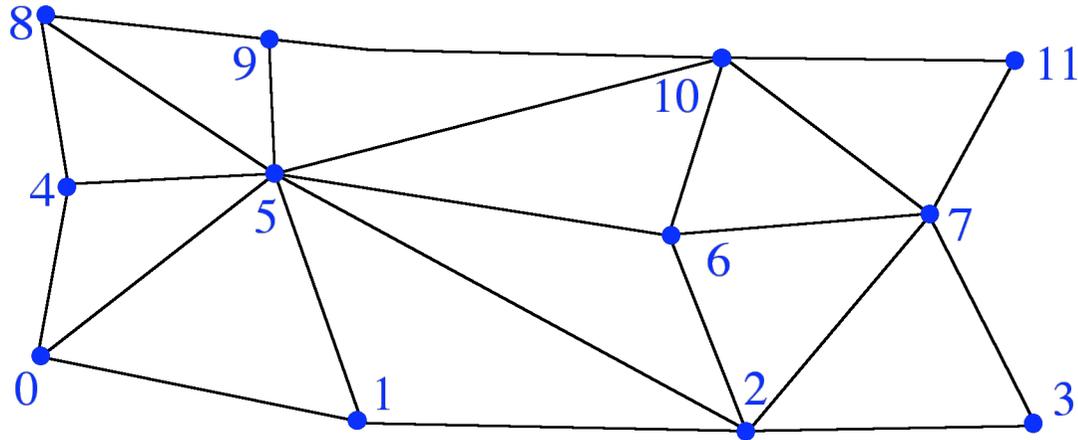
**IF** newpix.z < pixmap[x,y].z **THEN** pixmap[x,y] := newpix;



- Schattierungen simulieren Lichteinfall
- realistische Farben, Detail

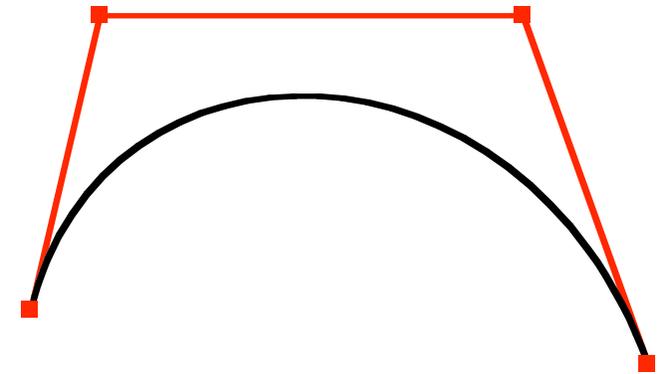


- TriGrid: Gruppe von Dreieck-Facetten
  - vereinfachte Oberflächenbeschreibung



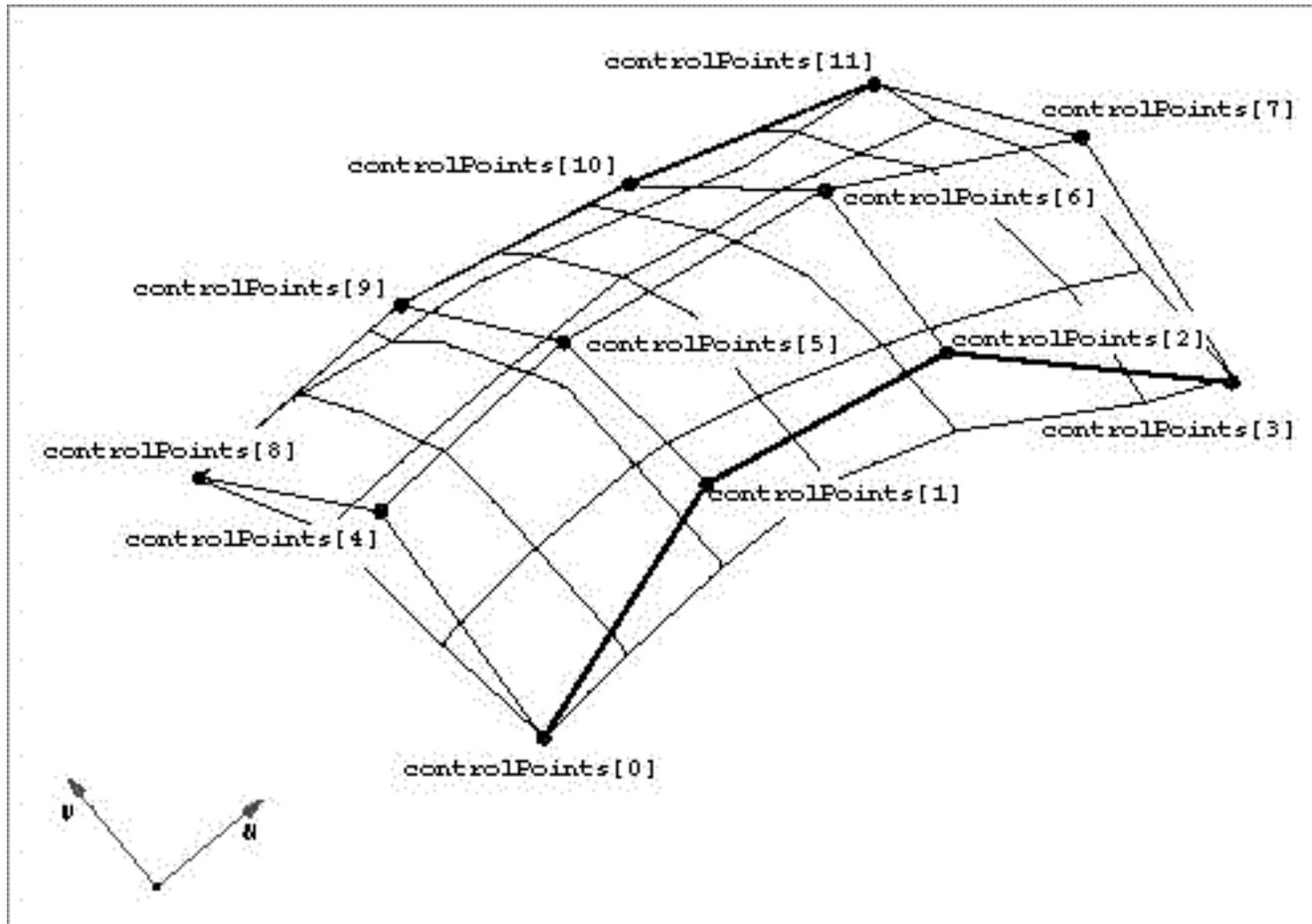
- Splines

- stückweise definierte Kurve
- Anpassung an vorgegebene Kurve
- viele Spline-Typen mit besonderen Eigenschaften
- Bézier, kubische Splines, deBoor
- NURB: nonuniform rational B-spline

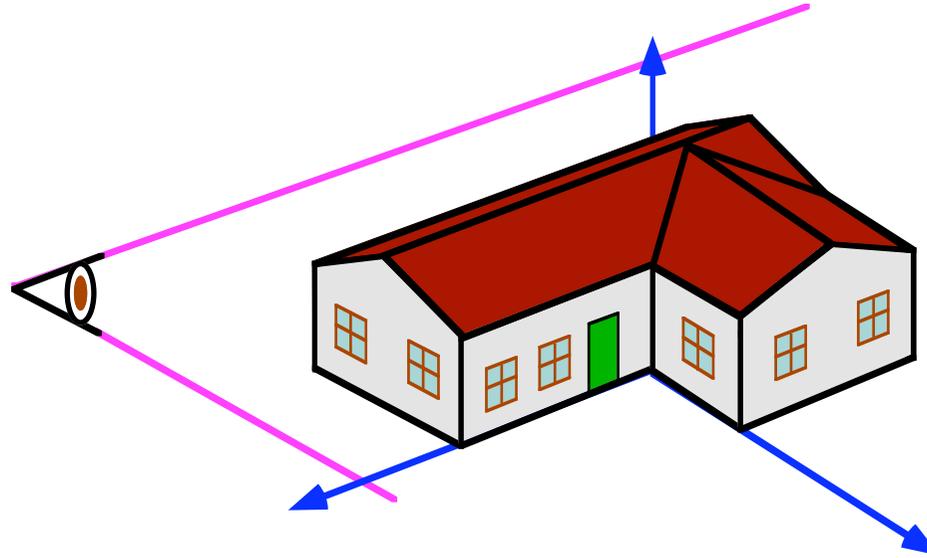


## • Spline-Patches

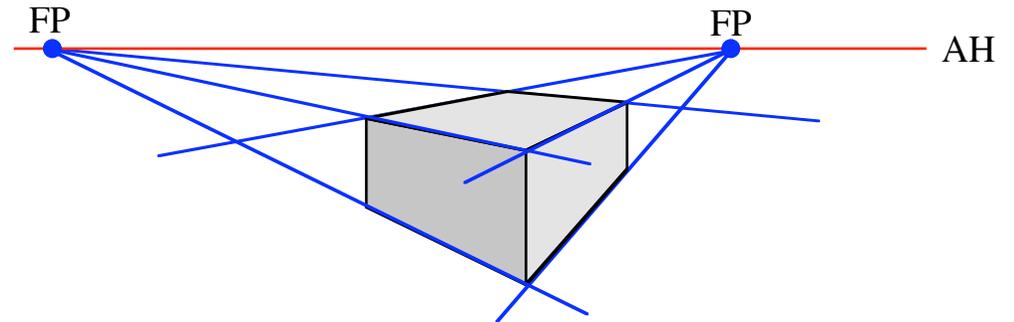
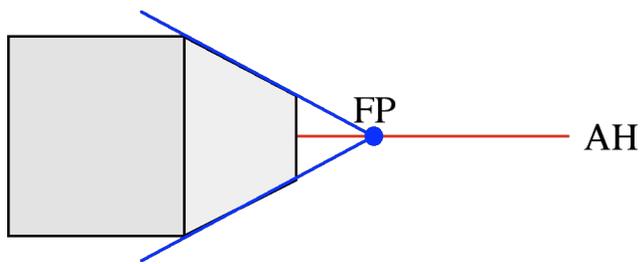
- stückweise Beschreibung von Oberflächen (patches)
- Flächen als 3-dimensionales Analogon von Splines
- Facetten sind Vierecke mit Splines als Kanten
- NURB-patches



- Kamera: Betrachtungsort, Blickwinkel, Öffnungswinkel



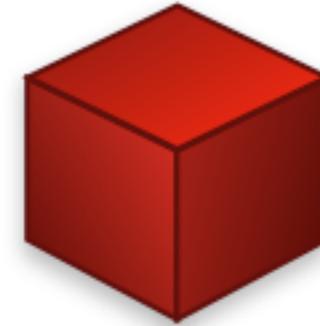
- Perspektivische Projektion
  - Fluchtpunkt(e)



- Maße nicht korrekt ablesbar

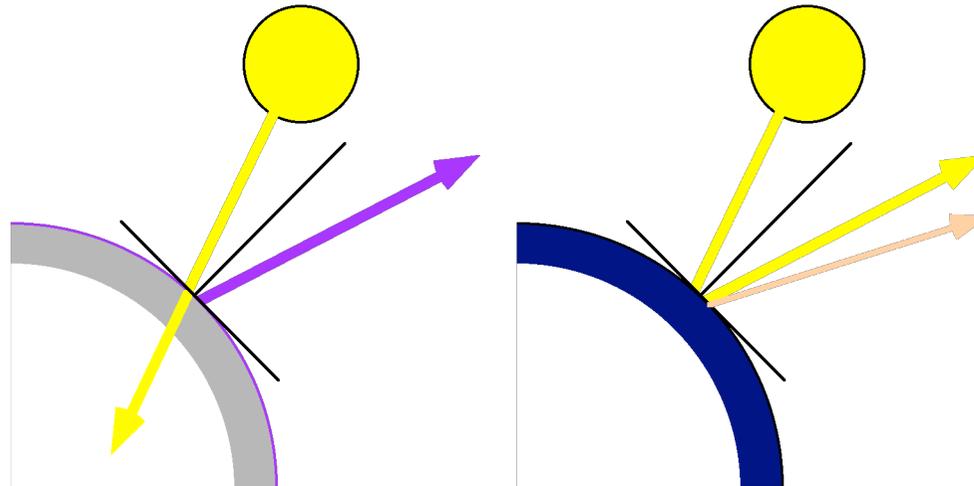
- Parallelprojektion

- orthographische Projektion: Grundriß, Aufriß
- schiefe (axonometrische) Projektion
- isometrische Projektion



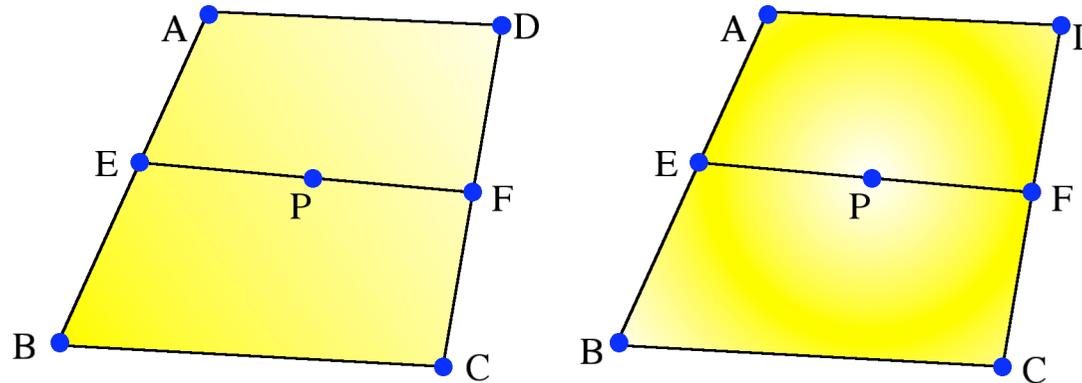
- Beleuchtung

- Umgebungslicht (ambient), Punkt-Licht
- diffuse Reflektion
- Objekte werden von Lichtquelle angestrahlt
- Licht wird teilweise reflektiert, teilweise durchgelassen

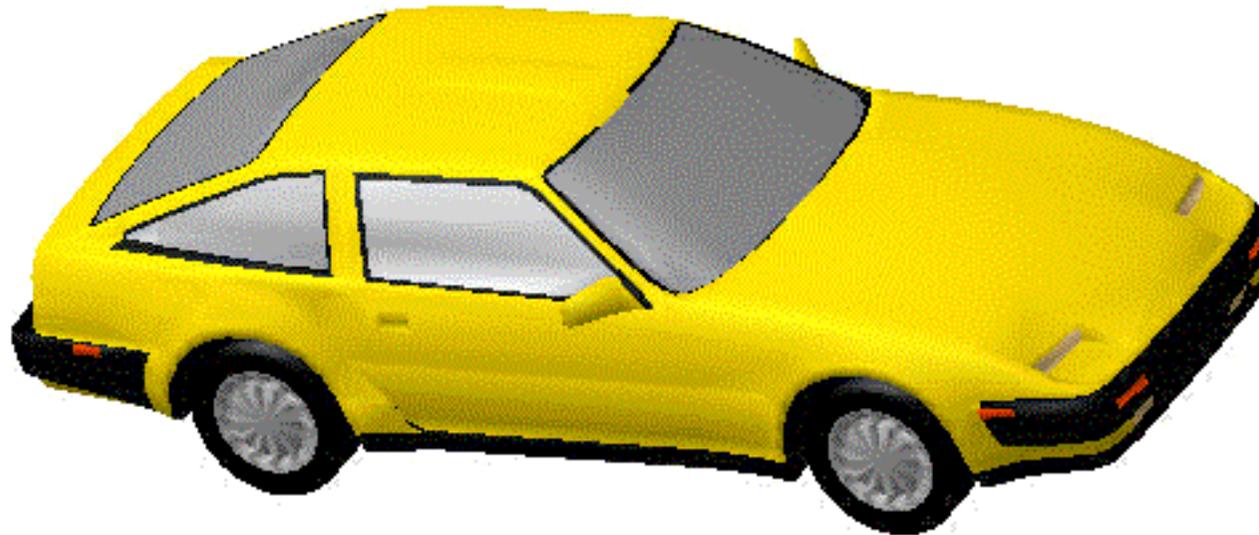


- spiegelnde Reflektion
- imitierendes Modell von Bui-Tung Phong, 1975

- Flat Shading: ein Farbwert pro Oberflächen-Facette
- Smooth Shading
  - Farbverlauf auf den Facetten
  - Gouraud-Shading: Interpolation zwischen Farbwerte an Eckpunkten

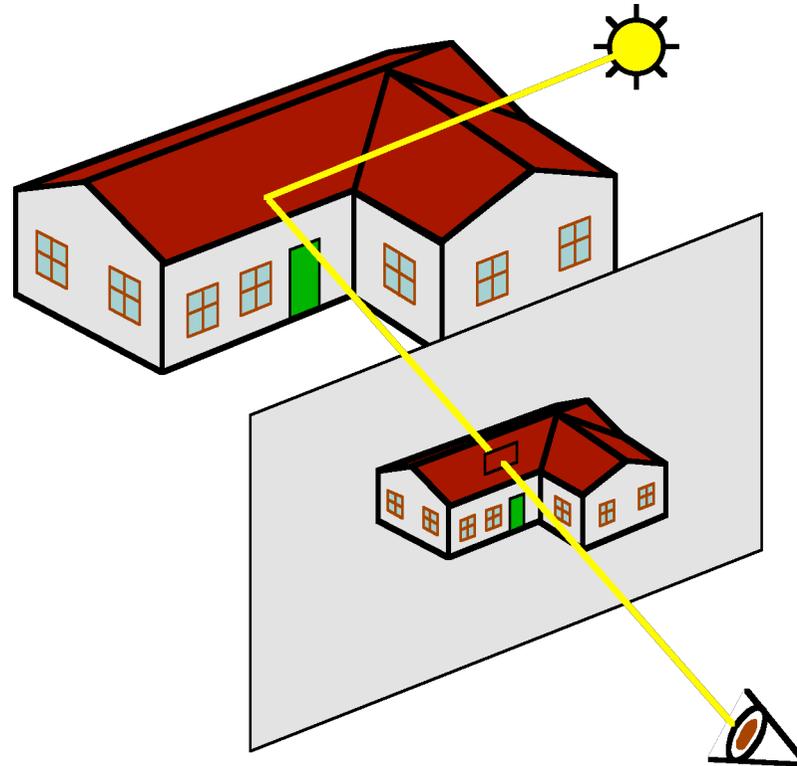


- Phong-Shading: individuelle Intensitätsberechnung für Flächenpixel



## • Ray-Tracing

- Reflektionen von Reflektionen, mehrfache Spiegelung
- Strahlpfad berechnen
- vorwärts und rückwärts (vom Auge zum Licht)

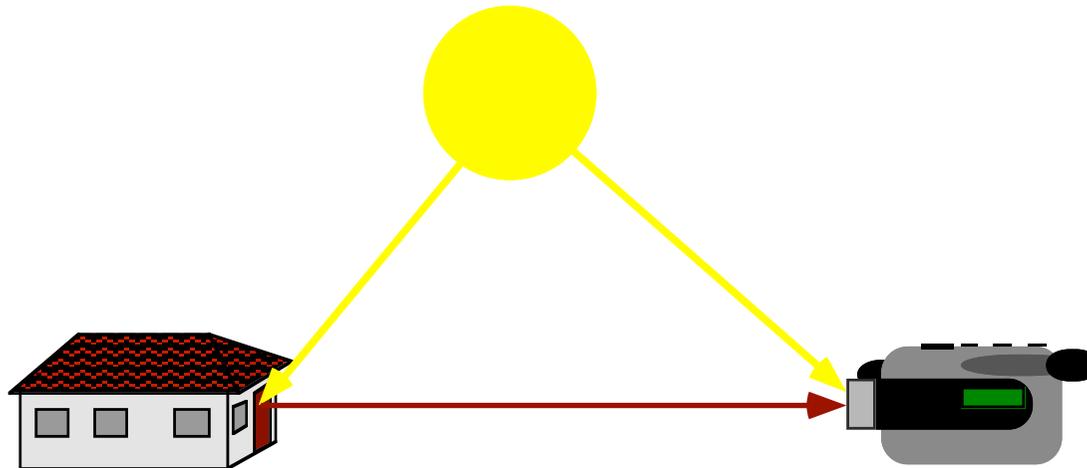


- rechenintensiv
- Berechnung blickpunktabhängig

- Radiosity
  - sichtunabhängige Berechnung
  - Einteilung der Oberflächen in patches
  - Emitter und Reflektor
  - Beleuchtungseinfluß auf alle anderen patches berechnen
  - Formeln aus der Wärmelehre
  - Abbrechen der Berechnung unter einem Grenzwert
- Textures
  - Oberflächenstruktur (Holz: Maserung)
  - 'Bekleben' der Oberflächen mit Muster
  - Bilder und Filme als Texturen
- VRML: Virtual Reality Markup Language
  - textuelle Beschreibung von 3D-Objekten und Szenen
  - primitive Objekte (cylinder, ...)
  - Transformation, Gruppierung, Oberflächeneigenschaften
  - Texturen (MPEG-Filme)
  - Objekte und Hyperlinks
  - Sensoren erzeugen Events für andere Objekte

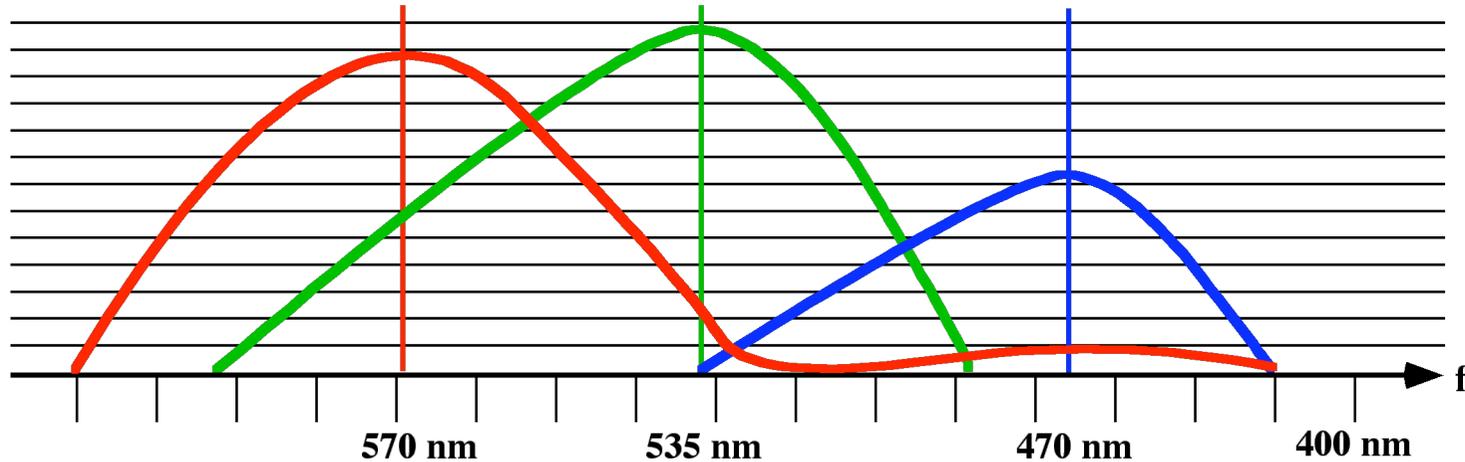
## 0.3 Standbilder

- Kontinuierliche Verläufe
  - Film hat höhere Auflösung als Auge
  - Abzüge, Bücher
  - Guter Druck typisch 2500 dpi
- Farbe
  - Lichtquelle (, Reflektion), Auge/Kamera/...:



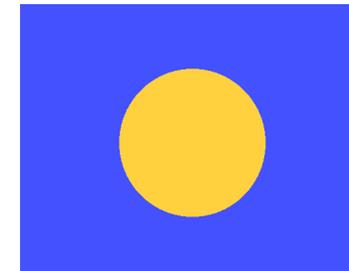
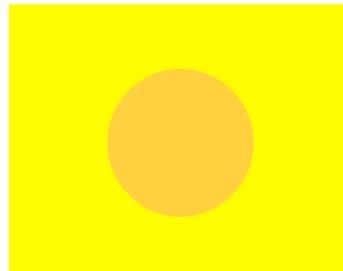
- Reflektiertes Licht = Licht - absorbiertes Licht = Oberflächenfarbe

- Spektrum und Empfindlichkeit des menschliche Sehapparates
  - 120 M Stäbchenzellen für Helligkeit in der Peripherie



- 7 M Zapfenzellen für Farbe (570, 535, 455 nm)

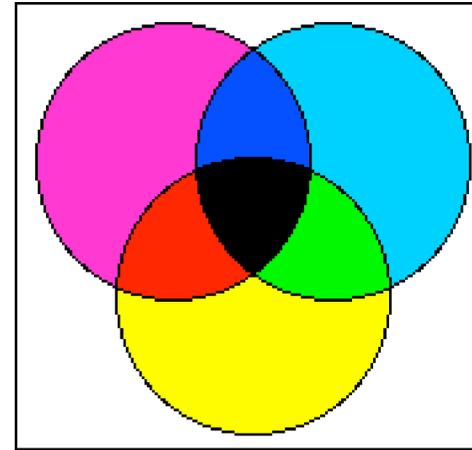
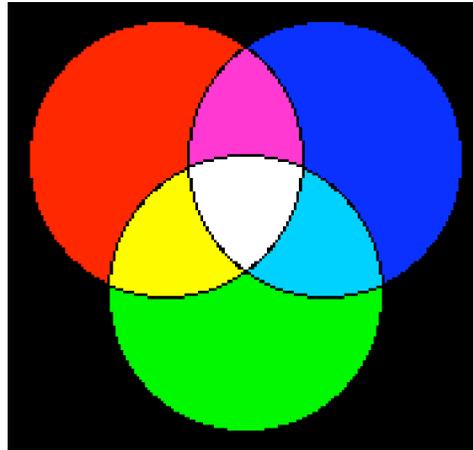
- Mensch sieht bis zu 350.000 Farbnuancen
- Abschattung



• Farbmischen:

additiv

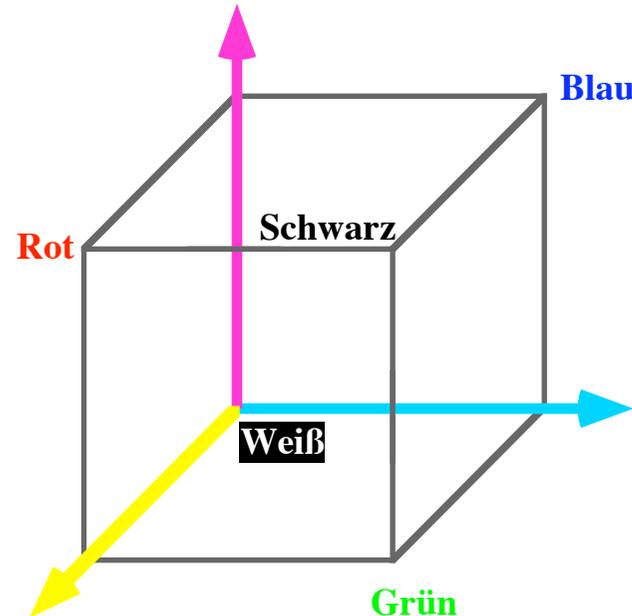
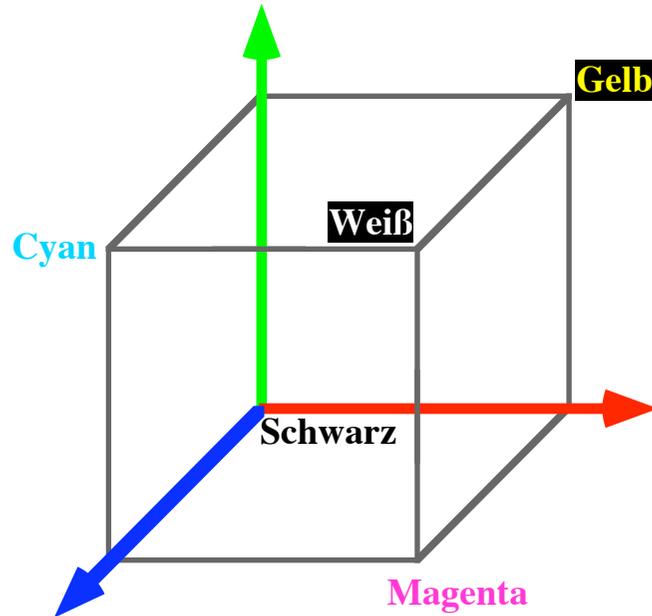
subtraktiv



• Farbmodelle

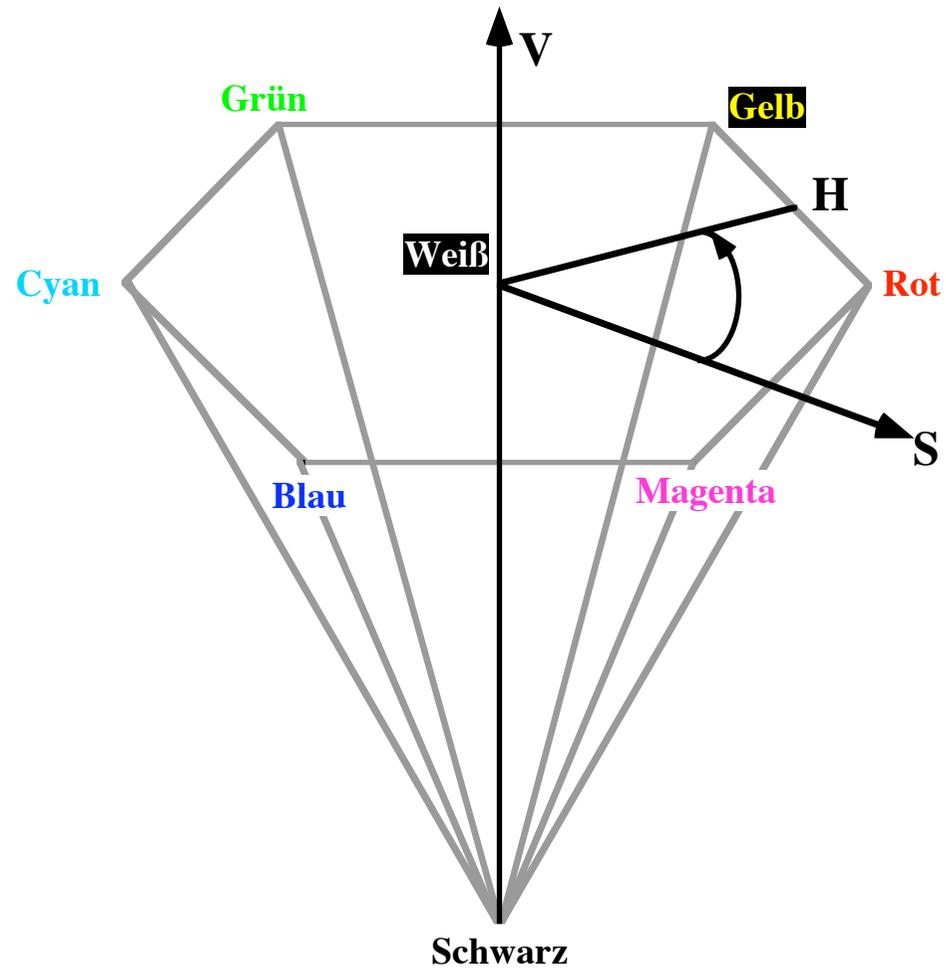
RGB

CMY



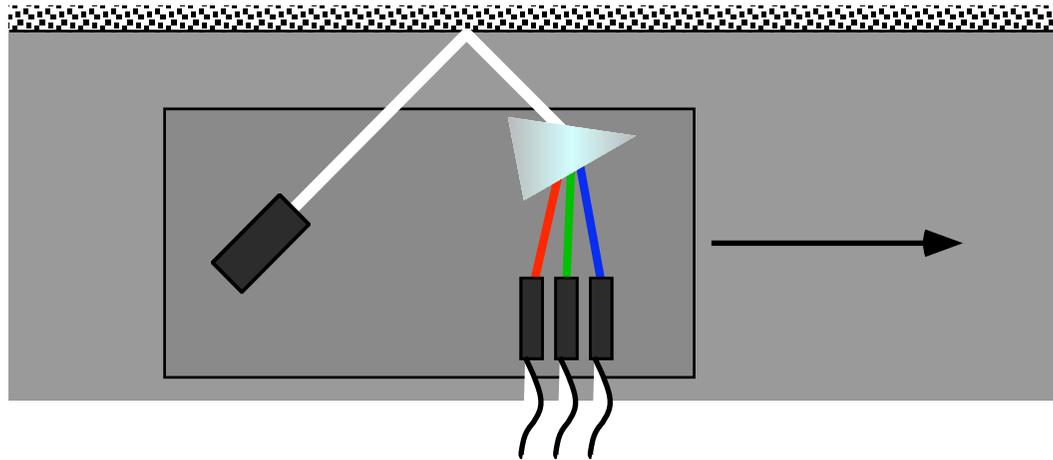
$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} \text{Weiß} \\ \text{Weiß} \\ \text{Weiß} \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

# - HSV (Ton, Sättigung, Helligkeit)

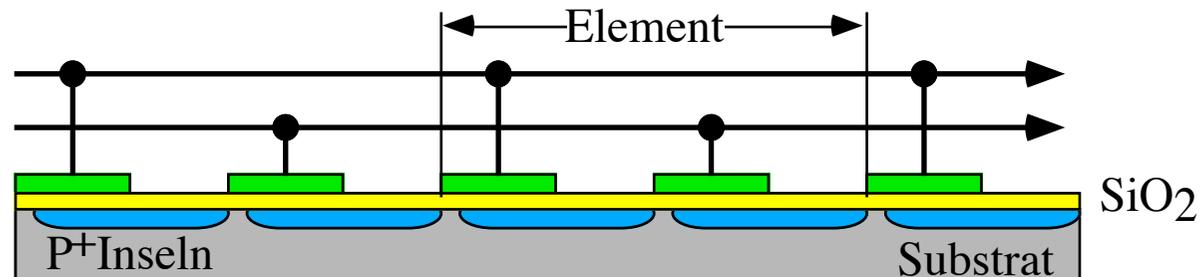


- Digitalisierung

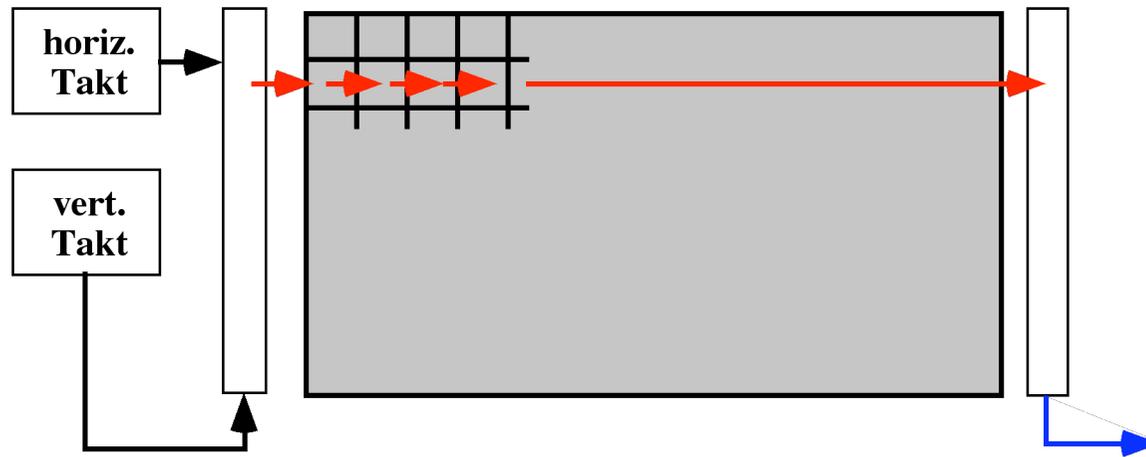
- Horizontale und vertikale Diskretisierung (Zerlegen in Pixel)
- Diskretisierungsschritt entspricht Auflösung: 72 bis 6000 dpi
- Bild wird angeleuchtet und Licht auf Detektor reflektiert
- Quantisierung: 8 oder 12 Bit für Graustufen  
8, 16 oder 24 Bit für Farbe  
eventuell mit Farbpalette



- CCD-Zeile



- Digitale Kameras benutzen CCD-Matrix

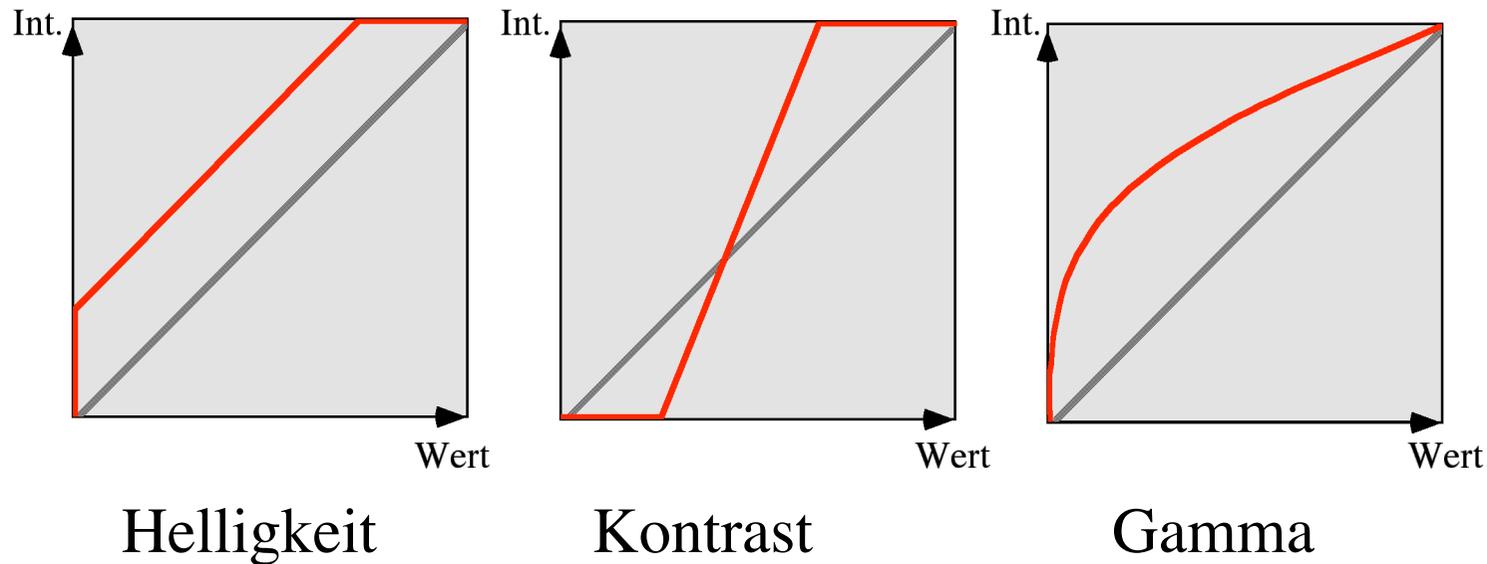


- ca 300.000 Zellen pro  $\text{cm}^2$
- Unterschiede zwischen Zellen
- nur 50 - 80% der Chipfläche ist mit aktiven Elementen bedeckt

- Bildkodierung

- RGB wird meist bei Computermonitoren verwendet
- CMYK (Cyan, Magenta, Yellow, Schwarz) besonders für Druck
- HSV für Fernsehen

- Aufbereitung nach der Digitalisierung
  - Kalibrierung der Farbwerte
  - Helligkeitsregelung, Kontrastverstärkung und 'Gamma' pro Farbkanal
  - Vorsicht: Color-Matching verwendet auch  $\gamma$



- Datenmenge kann groß werden
  - Auflösung für Weiterverarbeitung wichtig (Druckgewerbe)
  - $(200 * 4 \text{ [inch]}) * (200 * 6 \text{ [inch]}) * 3 \text{ Bytes} = 2.880.000 \text{ Bytes}$
  - => Kompression

## 0.4 Video

- S/W Fernsehen (eigentlich Graustufen)
- Auflösung wesentlich geringer als bei Standbildern

	Zeilen	Punkte/Zeile	Bilder/s	
CCIR 601	486	720	30	59,94 Hz
	586	720	25	50 Hz
CIF	288	352	25	Common Intermediate Format
QCIF	144	176	25	Quarter CIF
SIF	240	352	30	Standard Intermediate Format

- Fernsehnormen

- Halbbilder (Felder, gerade/ungerade Zeilen) mit doppelter Frequenz
- Farbfernsehen:

PAL, SECAM: 50 Hz

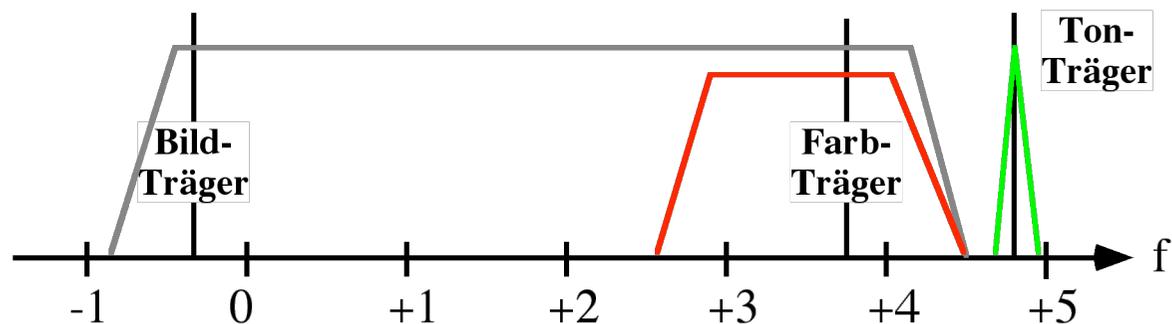
NTSC: 59,94 Hz (in Europa 50 Hz)

Bildwiederholrate = Übertragungsrate

- HDTV (1280-2048 \* 720-1152), Bildwiederholrate 50 oder 59,94 Hz

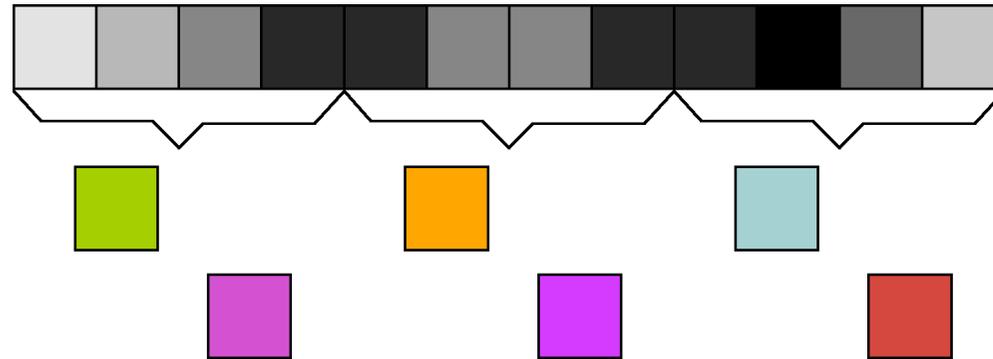
Bildwiederholrate  $\neq$  Übertragungsrate (24 Hz, 36 Hz, 72 Hz)

- Kameras produzieren RGB
  - drei Übertragungskanäle
  - Synchronisation?
  - => Mischsignal
- Composite
  - NTSC (National Television Systems Committee, ...)
  - PAL (Phase Alternating Line)
  - SECAM (Sequentiel Couleur avec Memoire)
  - Grundidee: SW-Fernsehen + irgendwas = Farbe
  - Farbraum mit Luminance und Chrominanz
  - Luminance := SW-Signal
- Farbraum HSV
  - Chrominanzsignal mit niedrigerer Bandbreite
  - auf Subcarrier (3,58 MHz)





- Farbauflösung des Auges schlechter -> Unterabtastung  
z.B.: 4:1:1 (YUV, PAL), 15:5:2 (YIQ, NTSC)



- Koeffizienten entsprechen Farbempfindlichkeit des Auges
- NTSC: YIQ (In-phase and Quadrature, I: 1,3 MHz, Q: 0,45 MHz)
  - $Y = 0,30 R + 0,59 G + 0,11 B;$
  - $I = 0,60 R - 0,27 G - 0,32 B;$
  - $Q = 0,21 R - 0,52 G + 0,31 B;$
- PAL: YUV (U, V: 1,3 Mhz)
  - $Y = 0,30 R + 0,59 G + 0,11 B;$
  - $U = (B-Y) * 0,493 = -0,15 R - 0,29 G + 0,44 B;$
  - $V = (R-Y) * 0,877 = 0,61 R - 0,52 G - 0,10 B;$
- VHS noch stärker analog komprimiert

- Konvertierungsmatrix YIQ - > RGB

$$\frac{1}{4096} \begin{pmatrix} 4788 & 0 & 6563 \\ 4788 & -1611 & -3343 \\ 4788 & 8295 & 0 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- Farbraumkonvertierung teuer:

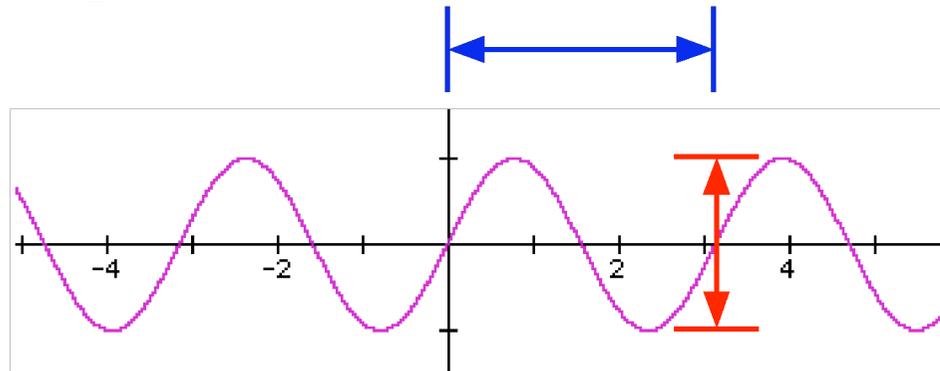
- 7 (5) Multiplikationen, 4 Additionen
- 8 Laden, 3 Speichern, 3 Shifts
- pro Pixel 23 Instruktionen plus Adressmanipulation ~ 30 Inst.
- CIF:  $288*352*25*30 = 76.032.000$  Instruktionen/sec
- zzgl. Upsampling
- Verbesserungen: R und B und Teil von G aus Tabelle:
  - 3 Laden, 1 Mult., 1 Add.; 8 Laden, 1 Shift, 3 Speichern = 17 + Adr
- oder Multiply-Accumulate Instruktion

- Digitale Kompression zwingend

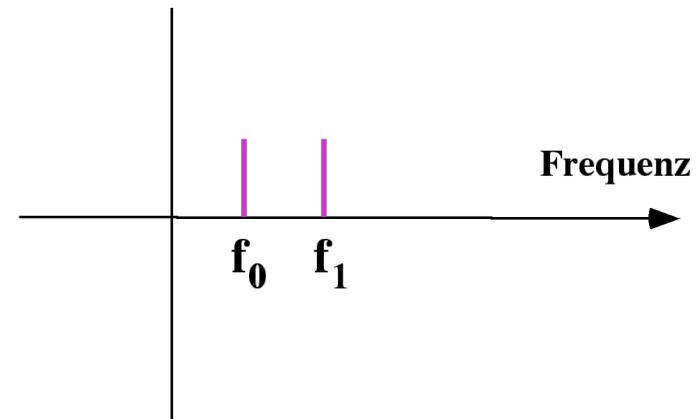
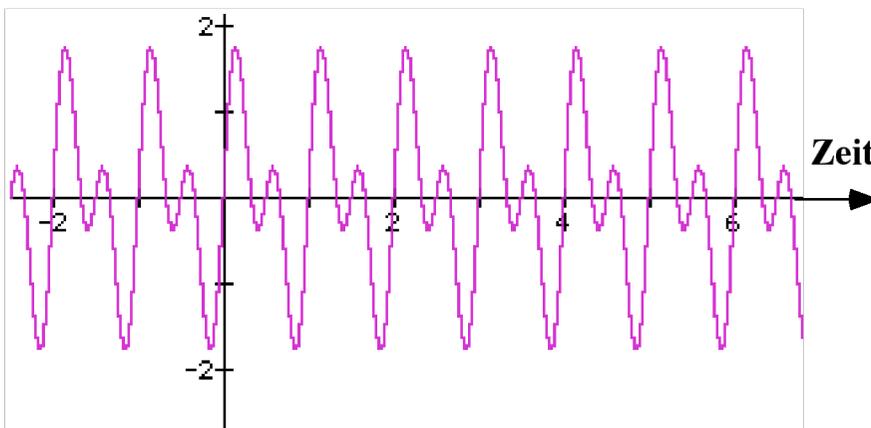
## 0.5 Audio

### 0.5.1 Audio-Eigenschaften

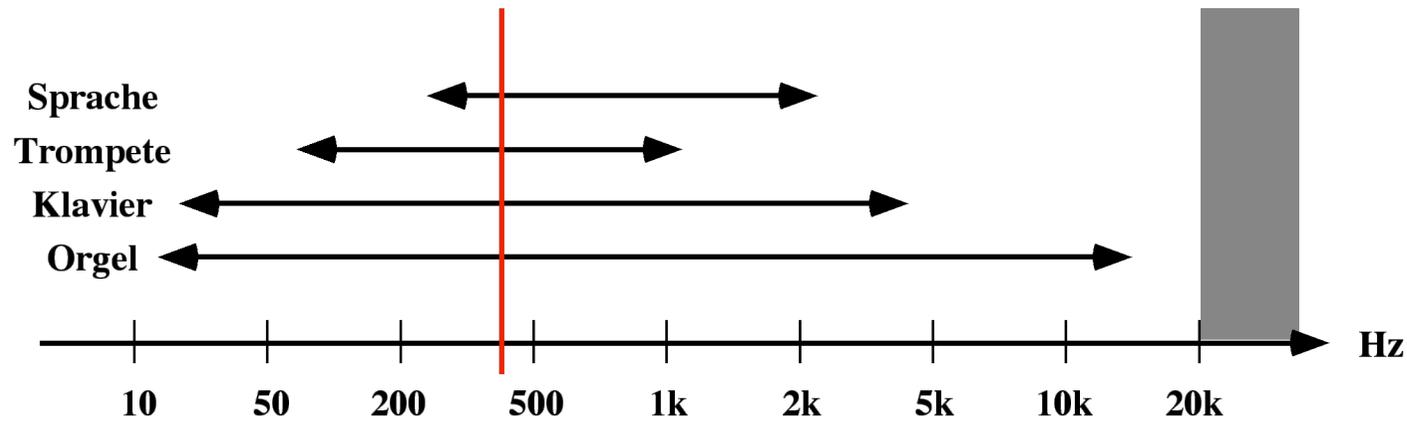
- Frequenz und Amplitude



- Amplitude -> Lautstärke (gemessen in dB)
- Frequenz (1m/Wellenlänge) -> Tonhöhe
- Fourier: Jede Schwingung kann als Summe von Sinusschwingungen dargestellt werden:  $f(x) = \sin 2\pi x + \sin 4\pi x$



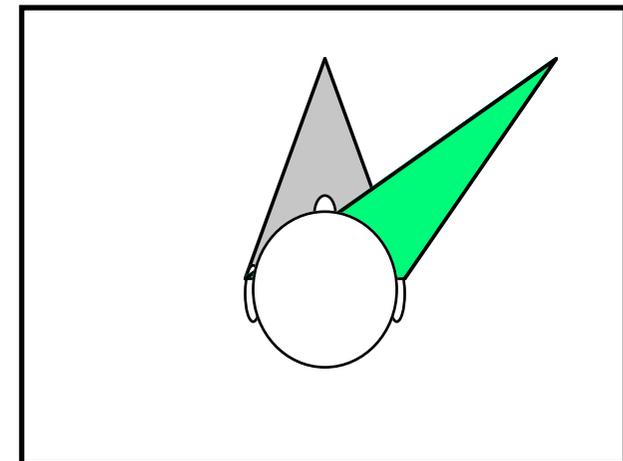
## • Typische Frequenzbereiche



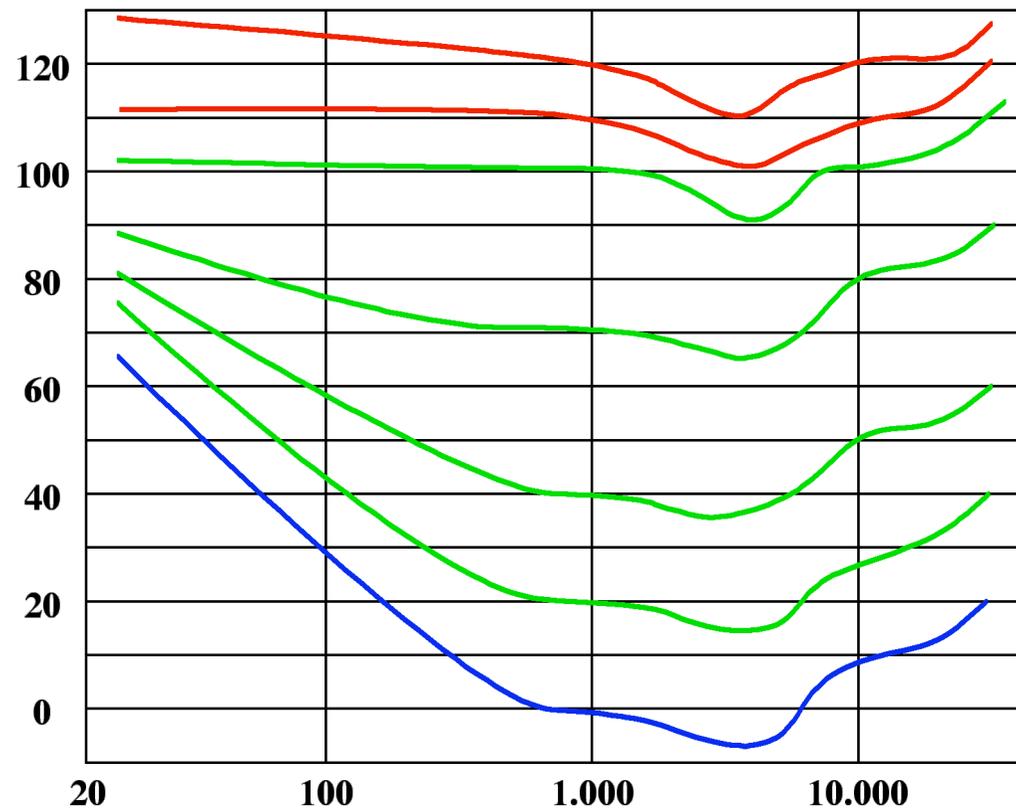
- Telefon 300Hz - 3.400 Hz
- Heimstereo 20 Hz - 20.000 Hz
- UKW (FM) 20 Hz - 15.000 Hz

## • Räumliches Hören

- Lautstärke
- Laufzeitunterschiede zu den Ohren
- Spektrale Analyse nach Ohrposition
- Filterfunktionen durch Außenohr
- Echos

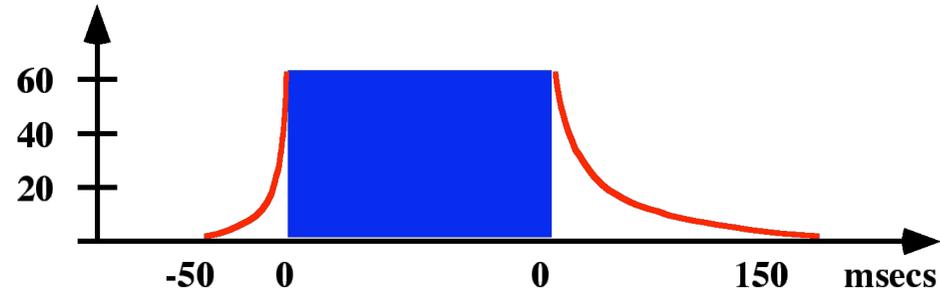


- Menschliches Hörvermögen
  - 20 - 20.000 Hz
  - hohes zeitliches Auflösungsvermögen
  - logarithmisch bezüglich Amplitude
- Lautstärkeempfinden nach Fletcher und Munson

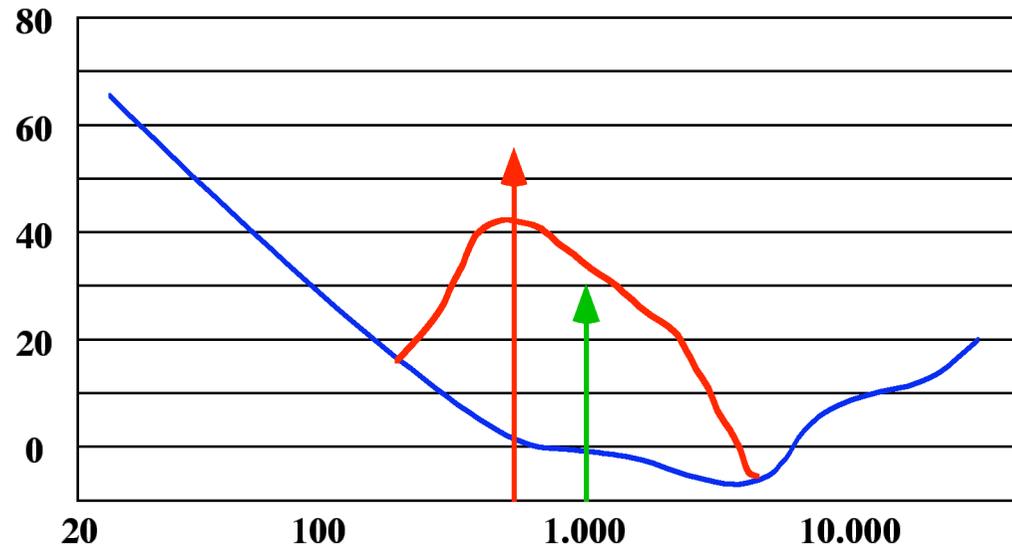


- Abschattung

- Zeit



- Frequenz



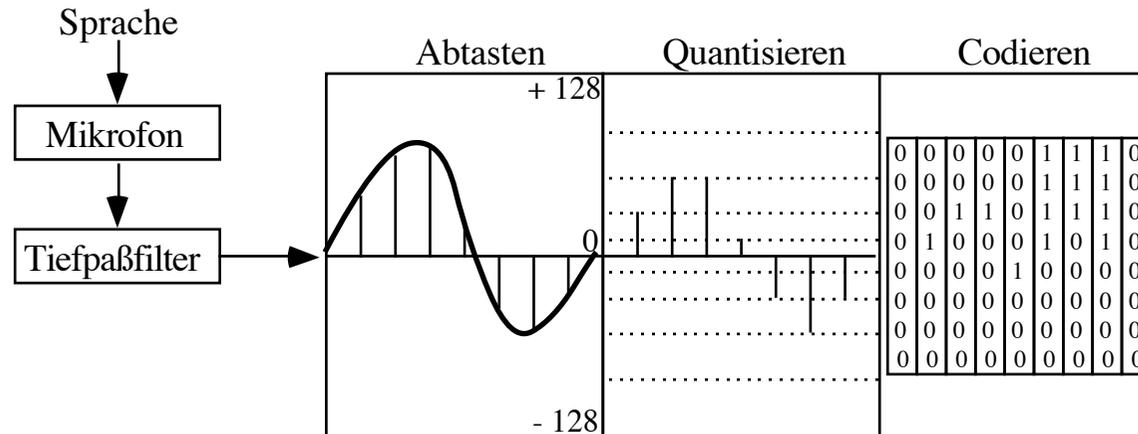
- Phase

- zwei gleiche, phasenversetzte Schwingungen können sich auslöschen:

$$y = \sin x + \sin (x + \pi)$$

## 0.5.2 Digitale Repräsentationen (PCM, CD-Audio, DAT, ...)

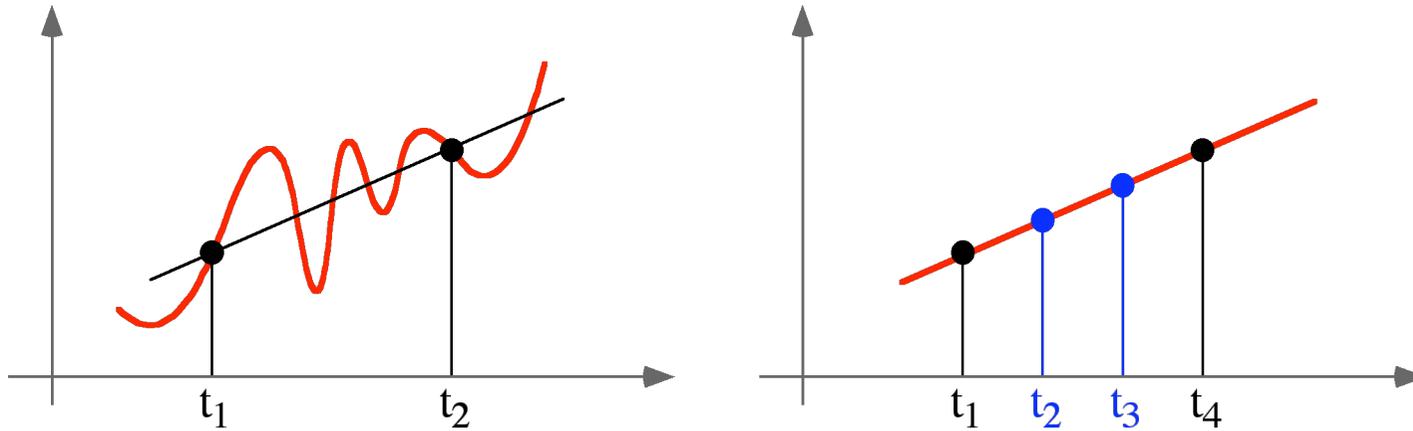
- Digitalisierung am Beispiel Telefon



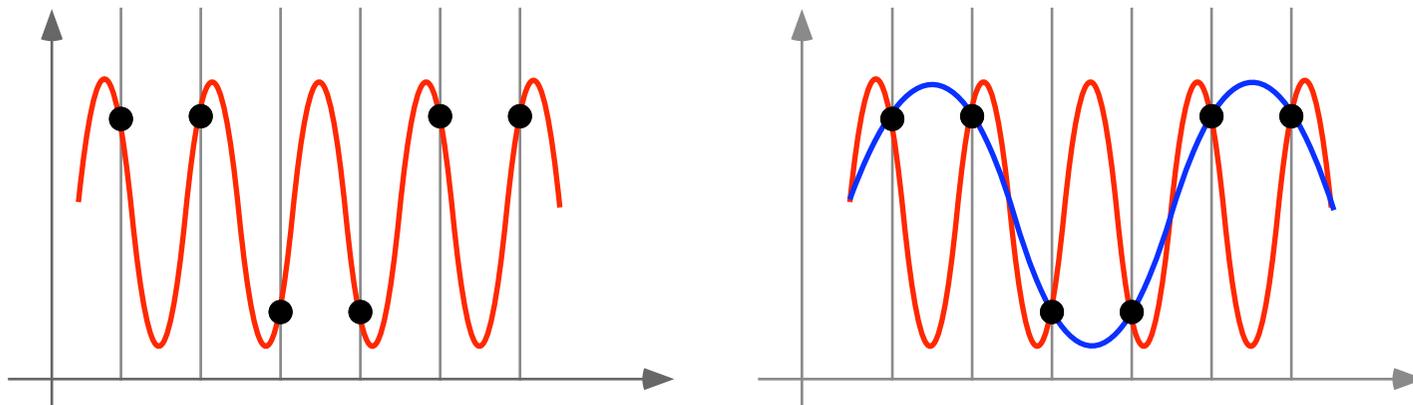
- Allgemein

- zeitliche Diskretisierung (Abtasten, Sampling)  
Einteilung der Zeitachse in einzelne Stücke
- Wert-Diskretisierung (Quantisierung)  
digitalen Näherungswert finden  
Reelle Zahl vs. Real/Integer

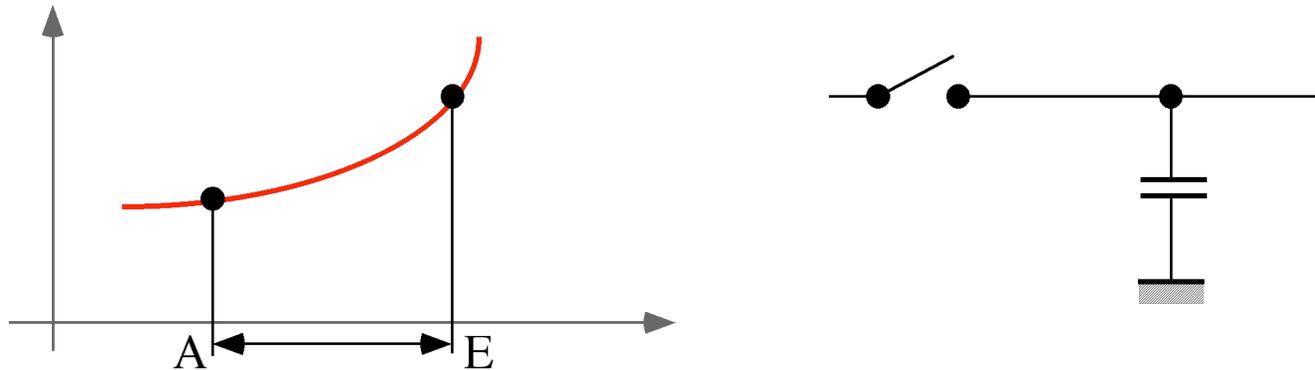
# • Abtasttheorem



- Anzahl Abtastwerte pro Zeiteinheit?
- Abtastfrequenz  $> 2 \cdot$  (höchste Frequenz)
- [Whittaker 1915/1929, Borel 1897]
- Aliasing bei zu niedrigen Abtastraten

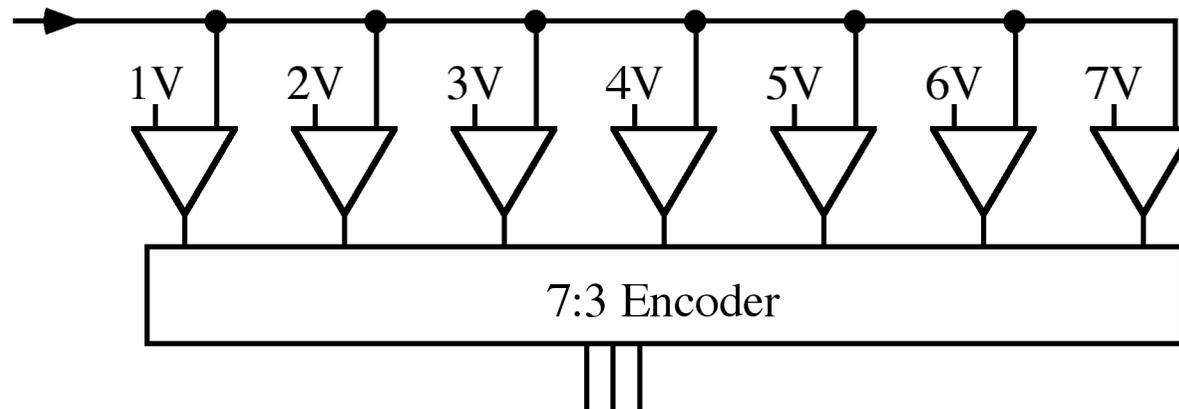


- Tiefpaß-Filter gegen Aliasing (siehe DSP-Kapitel)
- Sample-and-Hold



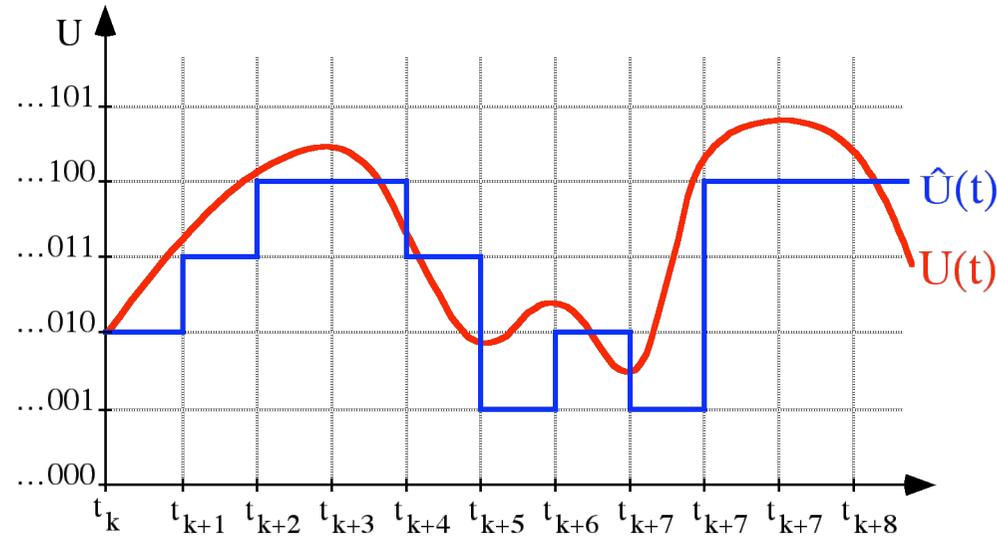
- Quantisierung (ADC)

- Wandlung des analogen Wertes in diskreten (digitalen) Wert



- Quantisierungsfehler
- 6 dB pro Bit => 96 dB bei 16 bit (CD-A)

- Diskretisierung und Quantisierung ergeben Treppenfunktion



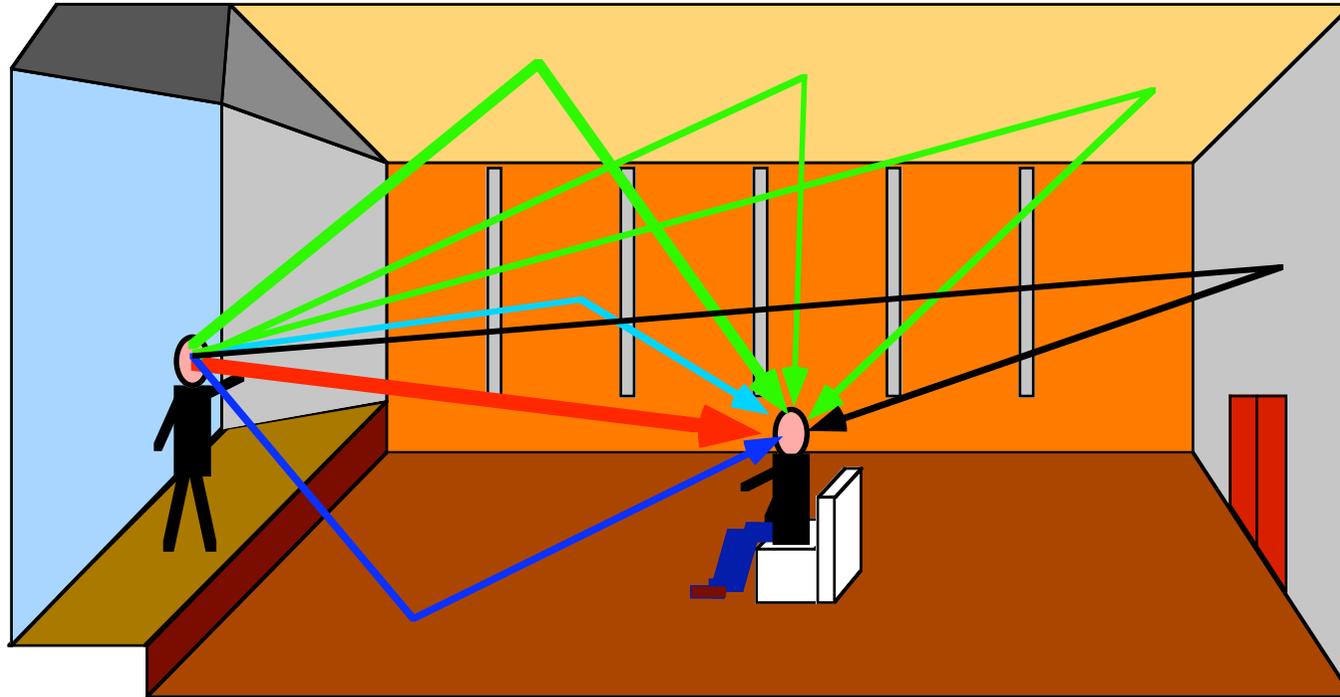
- Codierung

- als Integerzahl (CD: 16 bit)
- als Pseudo-Real (A-law,  $\mu$ -law: 8 bit)
- als Differenzen

±	E	x	p	M	a	n	t
---	---	---	---	---	---	---	---

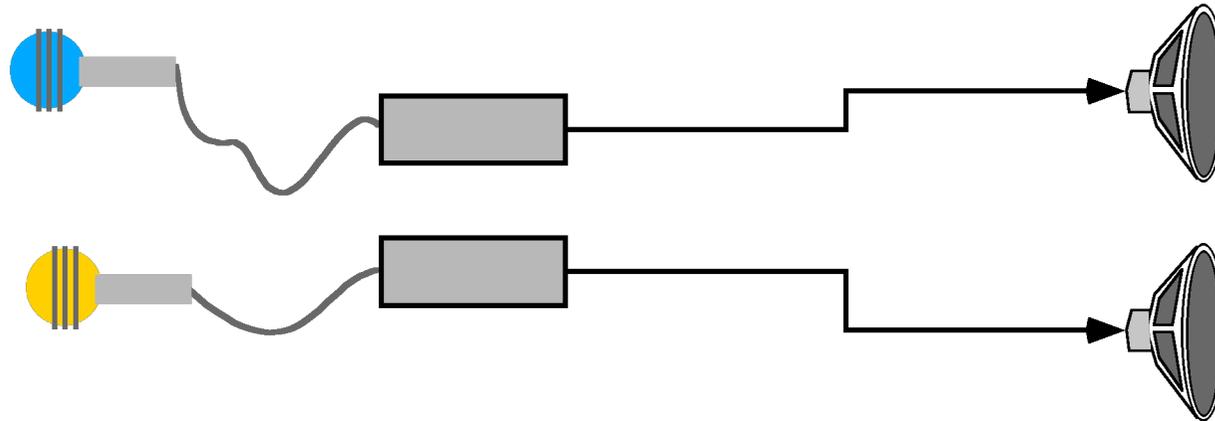
• typische Verfahren	bit	samples/sec	Kanäle	Datenstrom
CD-Audio	16	44.100	2	~1.4 Mbit/s
A-law (ISDN)	13 -> 8	8.000	1	64 kbit/s
ADPCM (Telefon)	13 -> 2	8.000	1	16 kbit/s

## 0.5.3 Raumton



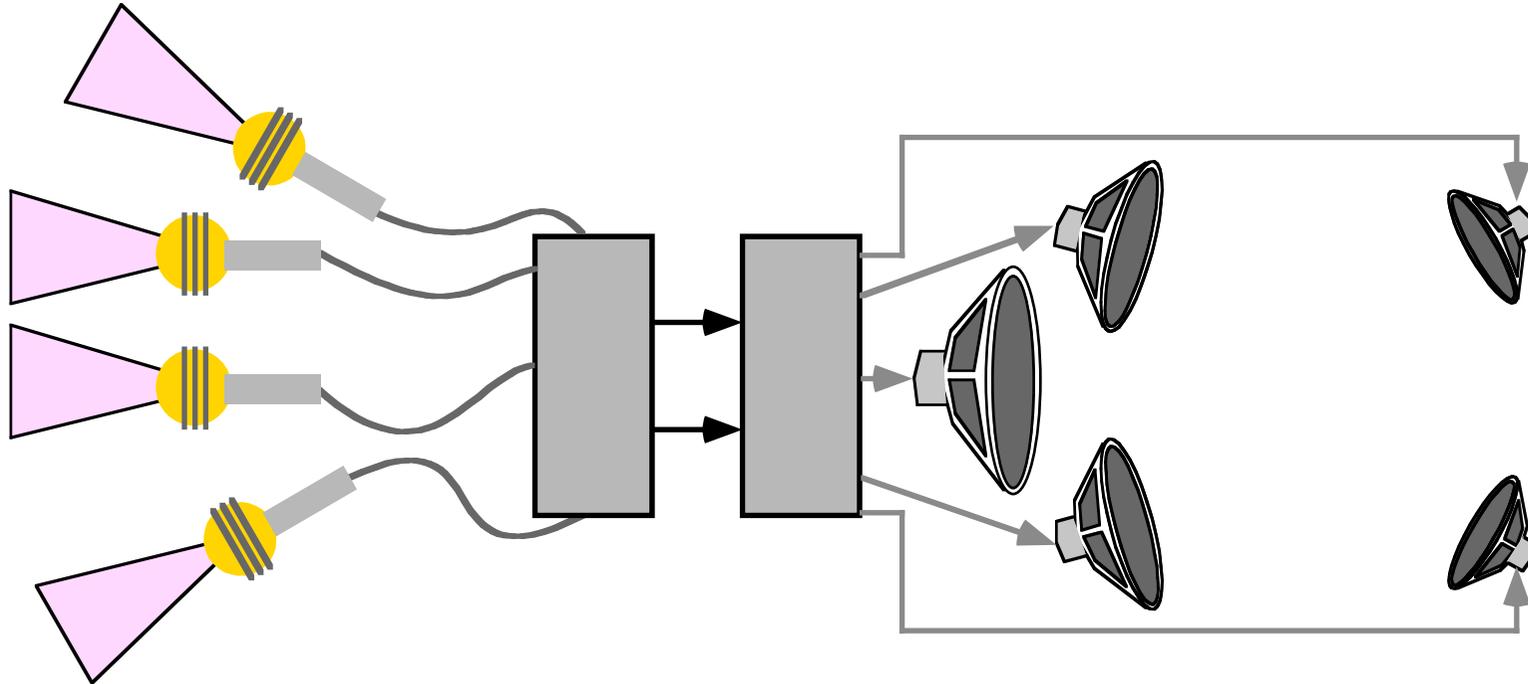
- Reflexionen von Wänden, Decke, Boden, Gegenständen
- Wahrnehmung der
  - Signalstärke
  - Richtung der Quelle
  - Dämpfung durch Kopf in höheren Frequenzen
  - Laufzeitunterschiede ( $650 \mu\text{sec}$  hörbar)
- Simulation der Reflexionen durch Laufzeitunterschiede

- Kopfhörer
  - kontrollierte Umgebung
  - Bewegungssensor: Kopfdrehung, Ortsveränderung
  - keine Richtungsortung
- Stereo: zwei Kanäle, links und rechts

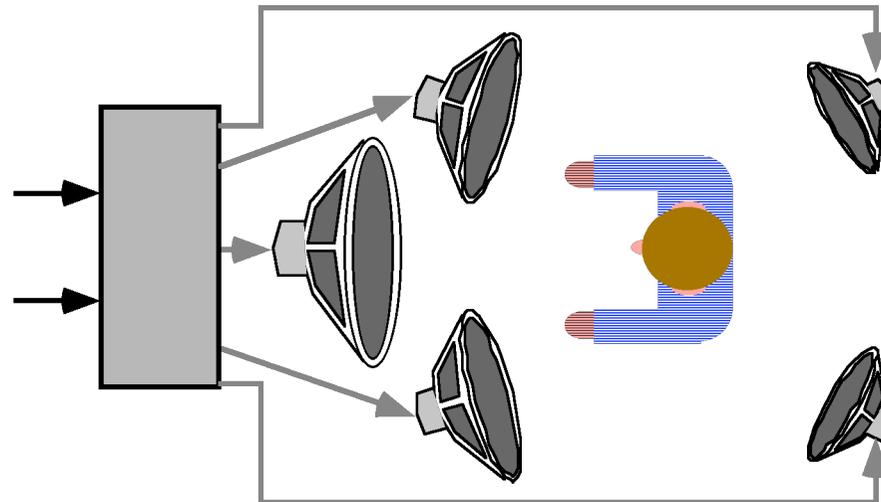


- Simulierter Raumklang
  - Reflektionen durch Verzögerung simulieren (-> Hall)

- Aufwendige Lautsprecheranordnung
  - Surround Sound (Dolby, DTS)
  - Raumaufnahme oder Simulation
  - Richtungsortung möglich



- Verbesserung des räumlichen Klangs
    - stereophon: 2 Kanäle links, rechts
    - quadrophon: 4 Kanäle VL, VR, HL, HR
    - Surround Sound: 4 Kanäle VL, VR, Center, Surround
- Raumgefühl auch außerhalb des akustischen Zentrums
- Links-Rechts wie bei Stereo
- Dialog 'aus dem Bild'
- Umgebungslautsprecher erzeugen Rundumeffekt
- Surround-Kanal < 7 kHz



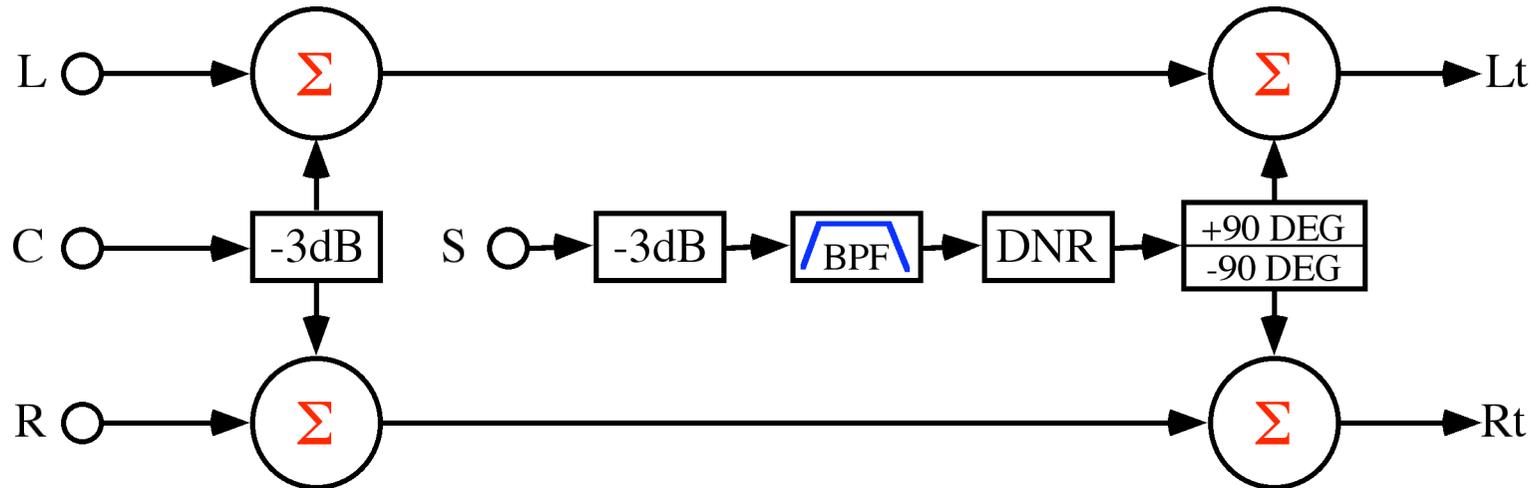
## • Kodierung in zwei Kanälen

- Einpassung in Verteilungsweg (Film, TV, Rundfunk, Band, CD)
- Kompatibilität mit normalen Stereoanlagen und TV
- Kodierung

$$L_t := V_L + 0,7 * C + 0,7 * S'$$

$$R_t := V_R + 0,7 * C + 0,7 * S''$$

Faktor 0,7 entspricht -3 dB



## - S-Kanal

Bandpass 100 Hz - 7 kHz, Dolby-B

$\pm 90^\circ$  phasenverschoben in Lt bzw. Rt

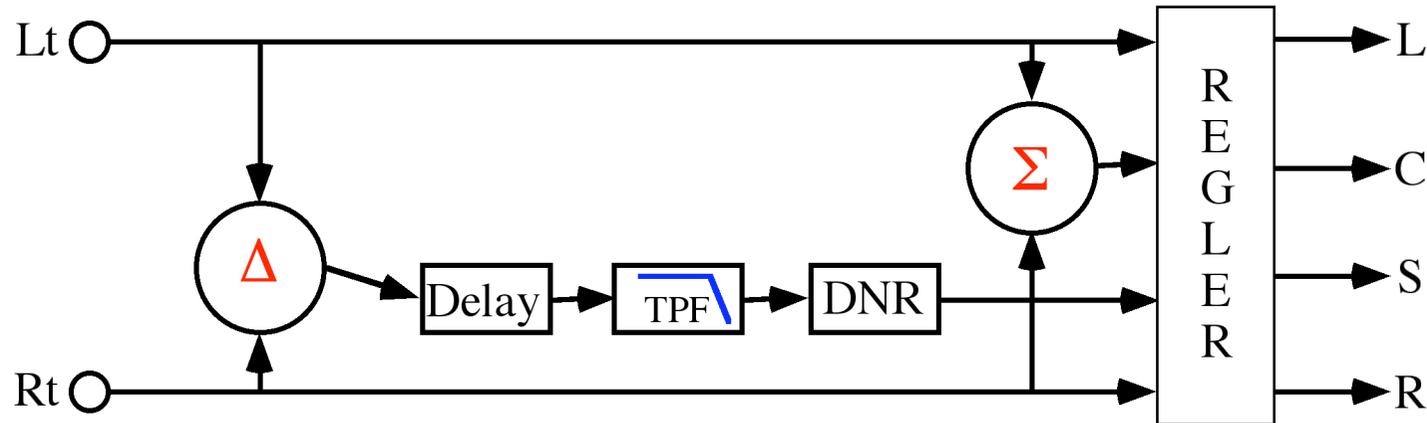
- Dekodierung

$$VL := Lt$$

$$VR := Rt$$

$$C := Lt + Rt = VL + VR + 1,4 * C (+ 3dB)$$

$$S := Rt - Lt = VR - VL + C - C + S' - S'' = VR - VL + 1.4 * S$$



- S wird mit Verzögerung abgespielt

20 - 25 ms für Mittenposition

15 für vorn

30 für hinten

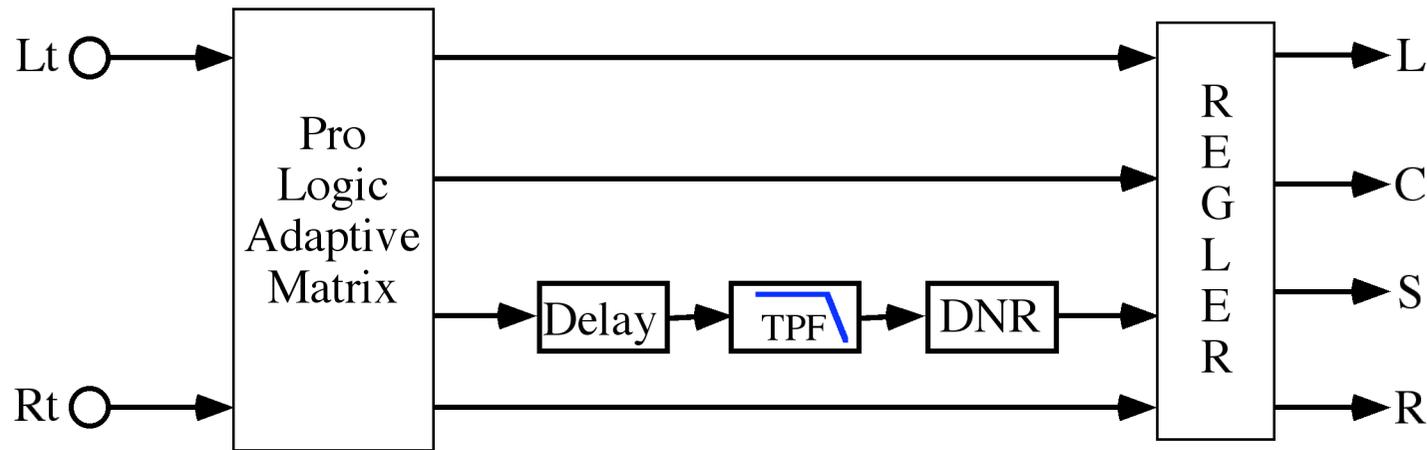
- Dolby Labs Pro-Logic

- Signalverarbeitung erhöht die Kanaltrennung

- dominante Schallquellen erscheinen reduziert in falschen Kanälen

- => Signaldominanzvektor (Kanäle logarithmisch vergleichen)

- => dominante Signale verstärken



- 6000 Filme (<http://www.dolby.com/film.html>)

- Fernsehshows (StarTrek Voyager, Dave Letterman, ...)

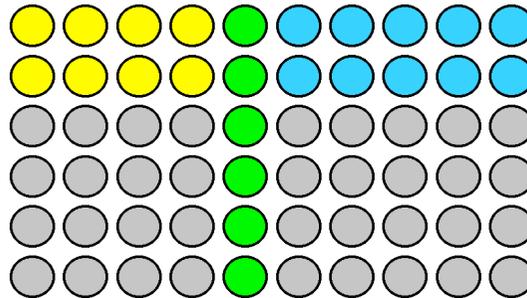
- Musik CDs

- THX: Qualitäts-Richtlinien

# 1. Algorithmen

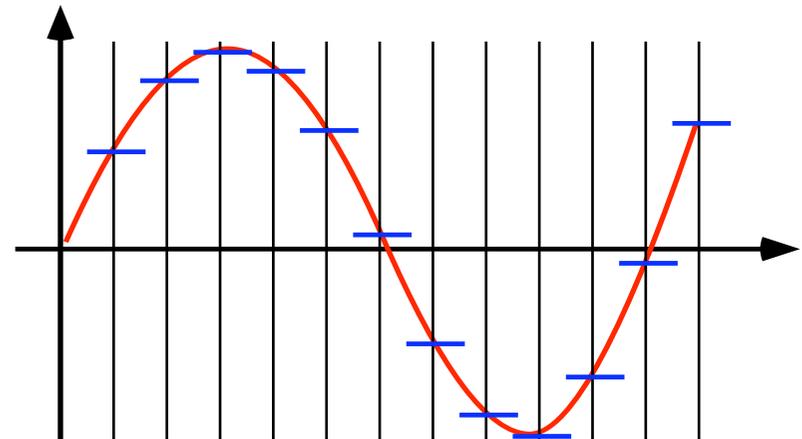
## 1.1 Kompression

- Fast alle Daten enthalten Redundanz:
  - "eins, zwei, drei"  $\leftrightarrow$  "1,2,3"

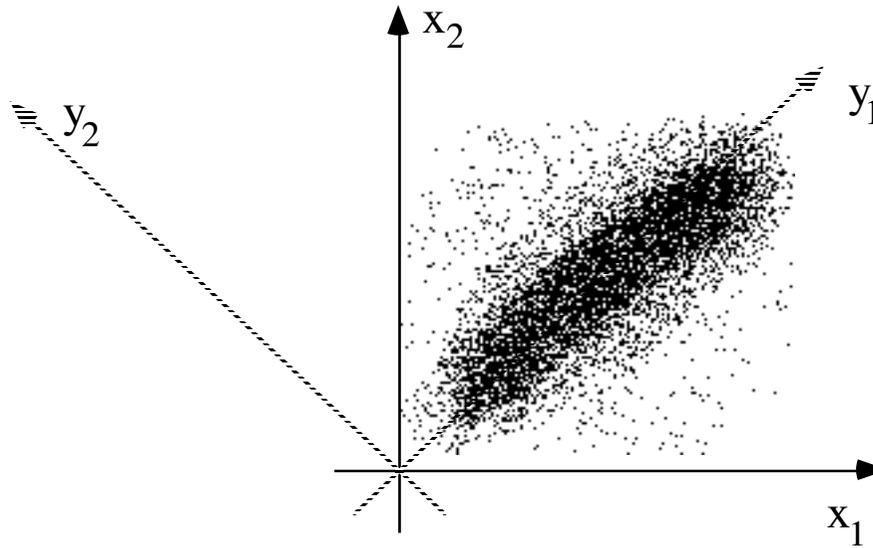


- Unterdrückung von Redundanz erhöht Informationsrate
  - Speicherung
  - Übertragung
- Einführung von nicht wahrnehmbarem Verlust
- Probleme
  - Rechenaufwand
  - Fehleranfälligkeit (Bitfehler, Paketverlust, etc)
  - Informationsverlust ( $\leftrightarrow$  Weiterverarbeitung)

- Entropie-Kodierung
  - Korrelationen finden und ausnutzen
  - Nullunterdrückung, Lauflängen-Kodierung (Problem: Rauschen)
  - Huffman Coding
  - Lempel-Ziv (LZW, etc)
  - verlustlos
  - Weiterverarbeitung möglich
- Differenzen-Kodierung
  - in Raum und Zeit
  - $d_i = |val_i - val_{i+1}| < \varepsilon$
  - Delta-Modulation
  - Vorhersage
- Truncation Coding
  - Grenzen der menschlichen Wahrnehmung
  - z.B. Runden: Gleitpunkt-Zahlen
  - Audio, Video
  - verlustbehaftet



- Transformationskodierung
  - 'Daten aus einem anderen Blickwinkel betrachten'

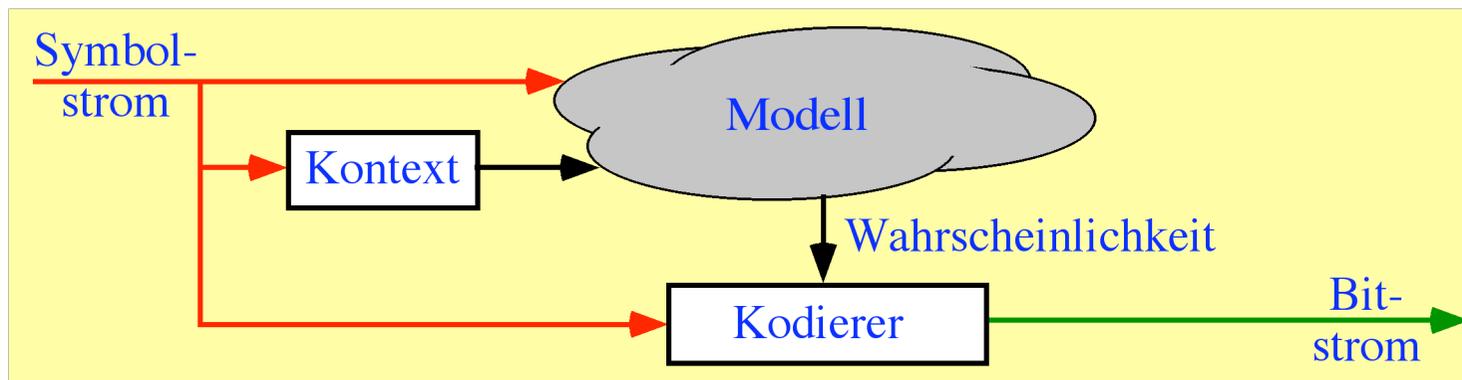


- Koordinatentransformation
- z.B. für Bilder
- Modellbasierte Kodierung
  - Decoder = eine Maschine die einen Medienstrom erzeugt
  - Coder findet die richtigen Eingaben für diese Maschine
  - Audio, Video mit sehr niedriger Bitrate

## 1.1.1 Entropiekodierung

- Shannon, 1948
- Nachrichtenquelle
  - Symbolfolgen
  - Wahrscheinlichkeit vs. Informationsgehalt
- Entropie eines Symboles  $s_i$  mit Wahrscheinlichkeit  $p_i$ 
  - $H_i = -\log_2 p_i$  bits
  - auch Selbstinformation oder Informationsgehalt genannt
  - $\neq$  durchschnittliche Entropie aller Symbole = Entropie der Quelle
- Wahrscheinlichkeit einer Symbolfolge  $s_1 s_2 \dots s_n$ ?
  - Symbole haben Wahrscheinlichkeit  $p(s_i) = p_i$
  - $p = p(s_1) * p(s_2) * \dots * p(s_n)$
  - $p(\text{"the"}) = 0,076 * 0,042 * 0,102 = 3,25584 * 10^{-4}$
  - $H(\text{"the"}) = 11,6$  bits
  - einfaches Modell
- Shannon: Mittlere Kodelänge kann Entropie annähern
- Shannon-Fano-Code tut das aber nicht

- Modelle mit endlichem Kontext
  - Wahrscheinlichkeit  $p(s_i)$  allein  $\neq$  im Kontext
  - $p(s_i='h') = 0,042 \Rightarrow H('the') = 11,6$  bits
  - $p(s_i='h' | s_{i-1} = 't') = 0,307 \Rightarrow H('the') = 6,5$  bits
- Brown'scher Korpus [W.N. Francis, H. Kucera; 1961]
  - Buchstaben, Digramme, Trigramme, Tetragramme
  - Wort-Delimiter ist auch ein Zeichen
- Ordnung des Modelles
  - Anzahl Vorgängerzeichen
- Modell versorgt Kodierer mit "Wissen"



### 1.1.1.1 Nullunterdrückung und Lauflängenkodierung

- viele Daten enthalten Läufe

- Nullunterdrückung

- Bsp.: BA 83 00 00 00 00 00 00 00 AF FE FF 00 00 00 03 00 05



- Problem: Marker+Zähler im Datenstrom

- => Präfix für Problemzeichen

- komprimiert: BA 83 06 AF FE FF 03 00 03 01 00 05

- Läufe beliebiger Zeichen

- Daten (Zeichen + Kennzeichen + Anzahl) Daten

- Kennzeichen im Datenstrom => Kennzeichen Kennzeichen

- Bsp.: Hallo AAAAAA, wieviel \$ kostet ein Auto?

- komprimiert: Hallo A\$6, wieviel \$0 kostet ein Auto?

- Vereinfachungen in Sonderfällen

- 7-bit Zeichen => höchstes Bit markiert Zähler

- 8-bit Zeichen => 7 bit + Präfix für seltene Zeichen mit 8 Bit

- FF xx bereits Marker im Datenstrom (JPEG)

### 1.1.1.2 Huffman Kompression

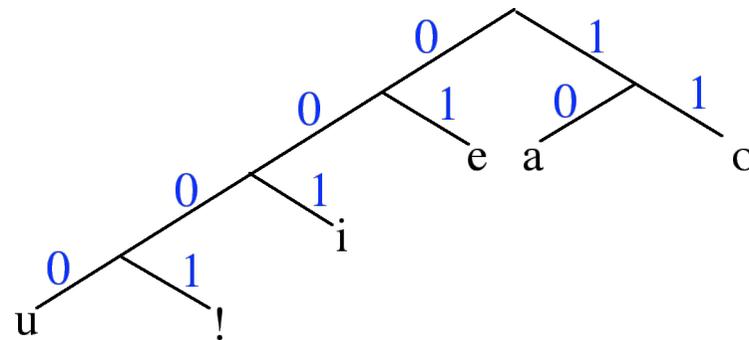
- $p(s_1) > p(s_2)$  dann sei  $\text{Länge}(c(s_1)) < \text{Länge}(c(s_2))$
- Idee: Baum erzeugen
  1. alle Nachrichten  $s_i$  in Tabelle aufschreiben
  2. Wahrscheinlichkeiten  $p(s_i)$  ermitteln
  3. 2 Nachrichten mit kleinster Wahrscheinlichkeit finden
  4. Zusammenfassen und Wahrscheinlichkeiten addieren
  5. Weiter mit 3 bis nur noch eine Nachricht
- Baum traversieren von Wurzel bis Blättern
  - linke Kante markiert mit 0, rechte mit 1
  - in den Blättern Bitkette der vorangehenden Kanten eintragen
  - es entstehen verschieden Lange Bitketten
- Anstelle der Symbole Bitketten aus dem Baum übertragen
- Redundanz = mittlere Codelänge - Entropie  $< p_{\max} + 0,086$

- Beispiel

- Nachrichten: (a,0.2), (e,0.3), (i,0.1), (o,0.2), (u,0.1), (!,0.1)
- iterativ zusammenfassen

e	0.3	e	0.3	e	0.3	(a,o)	0.4	((u,!),i),e	0.6	((u,!),i),e), (a,o)	1
a	0.2	a	0.2	((u,!),i)	0.3	e	0.3	(a,o)	0.4		
o	0.2	o	0.2	a	0.2	((u,!),i)	0.3				
i	0.1	(u,!)	0.2	o	0.2						
u	0.1	i	0.1								
!	0.1										

- Baum erzeugen und attributieren

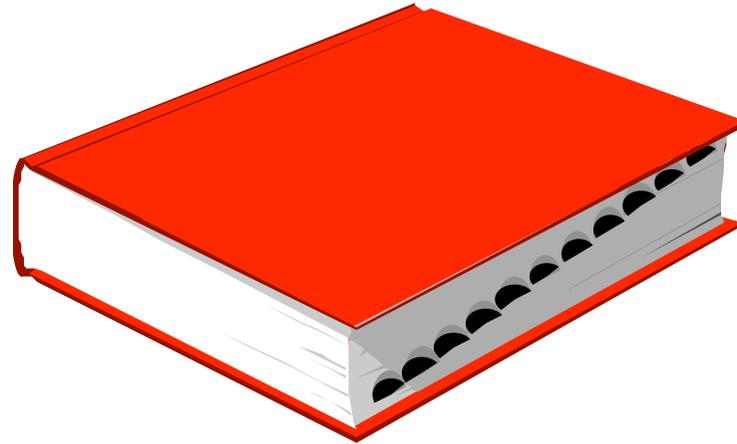


- Codetabelle: (a,10), (e,01), (i,001), (o,11), (u,0000), (!,0001)

- a-priori Wissen: Codebaum bzw. Eingabezeichen-Verteilung
- *Huffman* nur optimal falls  $p(a_i) = 1/2^n$  ;  $\forall i$  bei binärer Kodierung

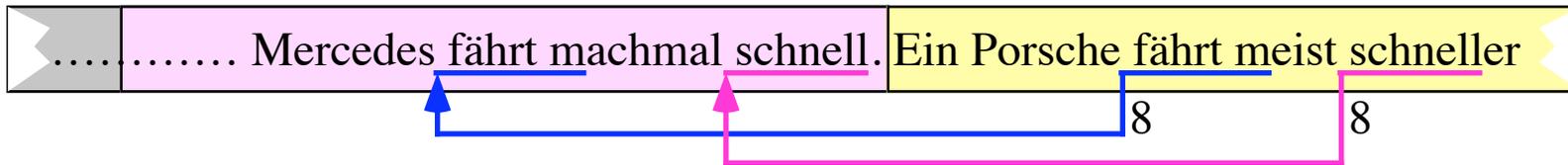
### 1.1.1.3 Wörterbuchkompression (Lempel-Ziv)

- Symbolgröße und Kodelänge
  - Konstante Symbolgröße : variable Kodelänge
  - Variable Symbolgröße : konstante Kodelänge



- Wörterbuch mit Tupeln (Phrase, Kode)
- Phrasen sind Folgen von Eingabesymbolen
- Erzeugung des Wörterbuches?
- Jacob Ziv und Abraham Lempel; 1977, 1978

- LZ77: alte Daten als Kodebuch
  - Fenster in der Vergangenheit absuchen
  - Zeiger + Länge in dieses Fenster übertragen



- Heute: LZH kombiniert LZSS mit Huffman

- LZ78: Faktorisierung des Eingabestroms

- Eingabe:	aaabbabaabaabab						
- Faktorisierung:	a	aa	b	ba	baa	baaa	bab
- Phrasen#:	1	2	3	4	5	6	7
- Ausgabe:	(0,a)	(1,a)	(0,b)	(3,a)	(4,a)	(5,a)	(4,b)

- Automatisch adaptiv
- Optimal falls Tabelle beliebig groß und Eingabe unendlich lang
- Datenstruktur für das Wörterbuch kritisch
  - Liste, Hash oder Mehrwegbaum

- Praktikabel seit Terry Welch, 1984: LZW  $\Rightarrow$  Patent  $\Leftrightarrow$  GIF
  - nur Codes (Pointer) übertragen, keine Symbole
  - Tabelle mit allen Symbolen initialisieren
  - neue Codes nicht sofort verwenden
  - letztes Symbol des neuen Codes  $\rightarrow$  erstes Symbol des nächsten Codes

- LZW-Algorithmus

```

WHILE getNextChar(theChar) DO BEGIN
  index:=FindEntry(currCode, theChar);
  IF dict[index].code <> empty THEN
    currCode:= dict[index].code
  ELSE BEGIN
    dict[index].code:=nextCode; inc(nextCode);
    dict[index].parentCode:=currCode;
    dict[index].character :=theChar
    Out(currCode);
    currCode:=ORD(theChar)
  END BEGIN
END END;

```

- Aufwand Kodierung:  $O(n_S)$ 
  - Schleife über Eingabestrom  $\rightarrow n_S$  Durchläufe
  - FindEntry
  - Hash max. 80% belegt, double hashing  $\rightarrow$  im Mittel 2 Versuche

- Aufwand Dekodierung:  $O(n_c)$ 
  - Schleife über Codestrom  $\rightarrow n_c$  Durchläufe
  - Arrayzugriff, Ausgabe
  - Wartung des Modelles (wie im Kodierer)
- Experimentell: 60 Instruktionen/Zeichen
- Eingabedatenströme oft heterogen
  - Keine neuen Tabelleneinträge frei
  - LZW: Codes verlängern
- Tabellen und Bäume müssen klein bleiben
  - Suchzeit, Speicherplatz
  - maximale Tabellengröße
  - Tabelle initialisieren falls voll
  - LRU wiederverwenden
- Kompressionsleistung messen
  - Kodierungsbasis initialisieren
  - Modell neu aufbauen

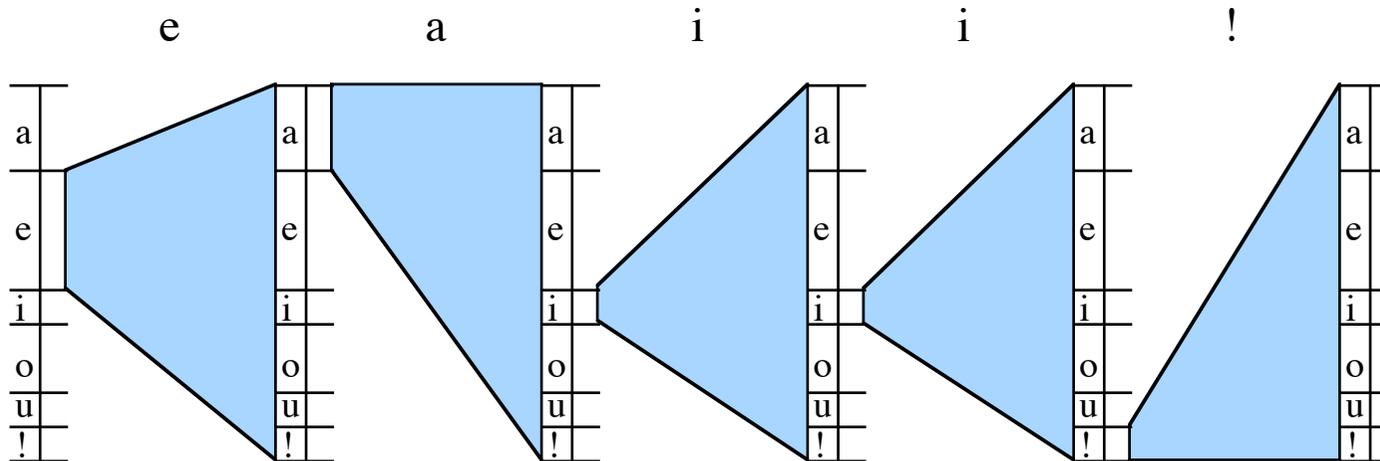
## 1.1.1.4 Arithmetische Kompression

- Nachricht durch Teilintervall von  $[0,1)$  repräsentieren
- Intervalltabelle

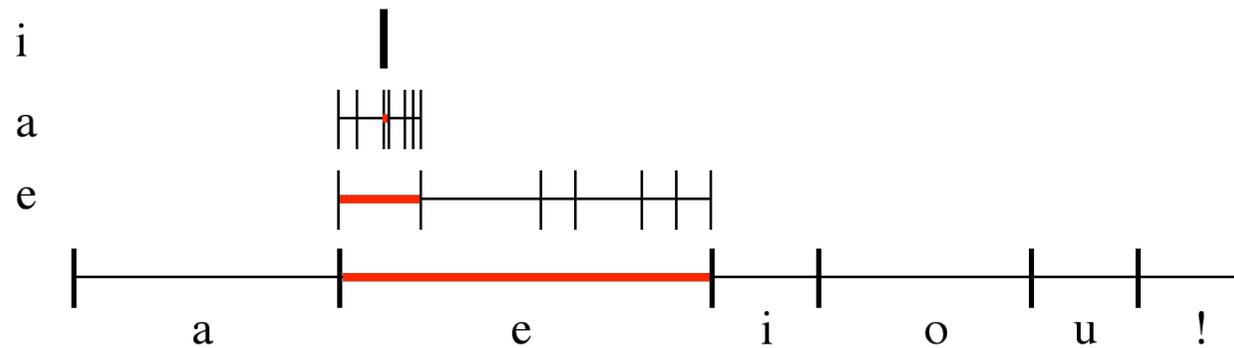
a	e	i	o	u	!
0,2	0,3	0,1	0,2	0,1	0,1
$[0 ; 0,2)$	$[0,2 ; 0,5)$	$[0,5 ; 0,6)$	$[0,6 ; 0,8)$	$[0,8 ; 0,9)$	$[0,9 ; 1)$

- Intervallentwicklung bei Eingabe eaii!

-	e	a	i	i	!
$[0 ; 1)$	$[0,2 ; 0,5)$	$[0,2 ; 0,26)$	$[0,23 ; 0,236)$	$[0,233 ; 0,2336)$	$[0,23354 ; 0,2336)$



- Jedes Eingabesymbol verkleinert und verschiebt Intervall
  - entsprechend  $p(s_i)$



```

WHILE GetSymbol(theSymb) DO BEGIN
  range:=high-low;
  high:=low+range*theSymb.high;
  low:=low+range*theSymb.low;
END;

```

- Decoder sucht Symbol zu Intervall
  - in Intervalltabelle
  - etwas schwieriger

- Finden des richtigen Intervalls
  - Anordnen der Symbole entsprechend  $p(s_i)$
  - Bisektion

```

WHILE theSymb <> EOF DO BEGIN
  theSymb:=FindSymb(value,range);
  PutSymbol(theSymb);
  range:=theSymb.high-theSymb.low;
  value:=value-theSymb.low;
  value:=value/range;
END;

```

- Länge der Gleitpunktzahl hängt von der Eingabelänge ab
- Inkrementelle Übertragung und Dekodierung
  - Senden eines Kodes, sobald er feststeht
  - im binären Fall: Intervall  $\in [0 ; 0,5) \Rightarrow 0$   
 Intervall  $\in [0,5 ; 1) \Rightarrow 1$
  - solange ausgeben, bis  $0,5 \in$  Intervall
  - Intervall-Skalierung: Bsp:  $[0,23 ; 0,236) \rightarrow [0 ; 0,6)$
  - nach jeder Ausgabe Intervall skalieren  $\Rightarrow$  im binären Fall shiften

- Kodierfenster

- Symbol senden falls entscheidbar
- Fenster verschieben

low: 0,1001000111011010

high: 0,1001101111101010

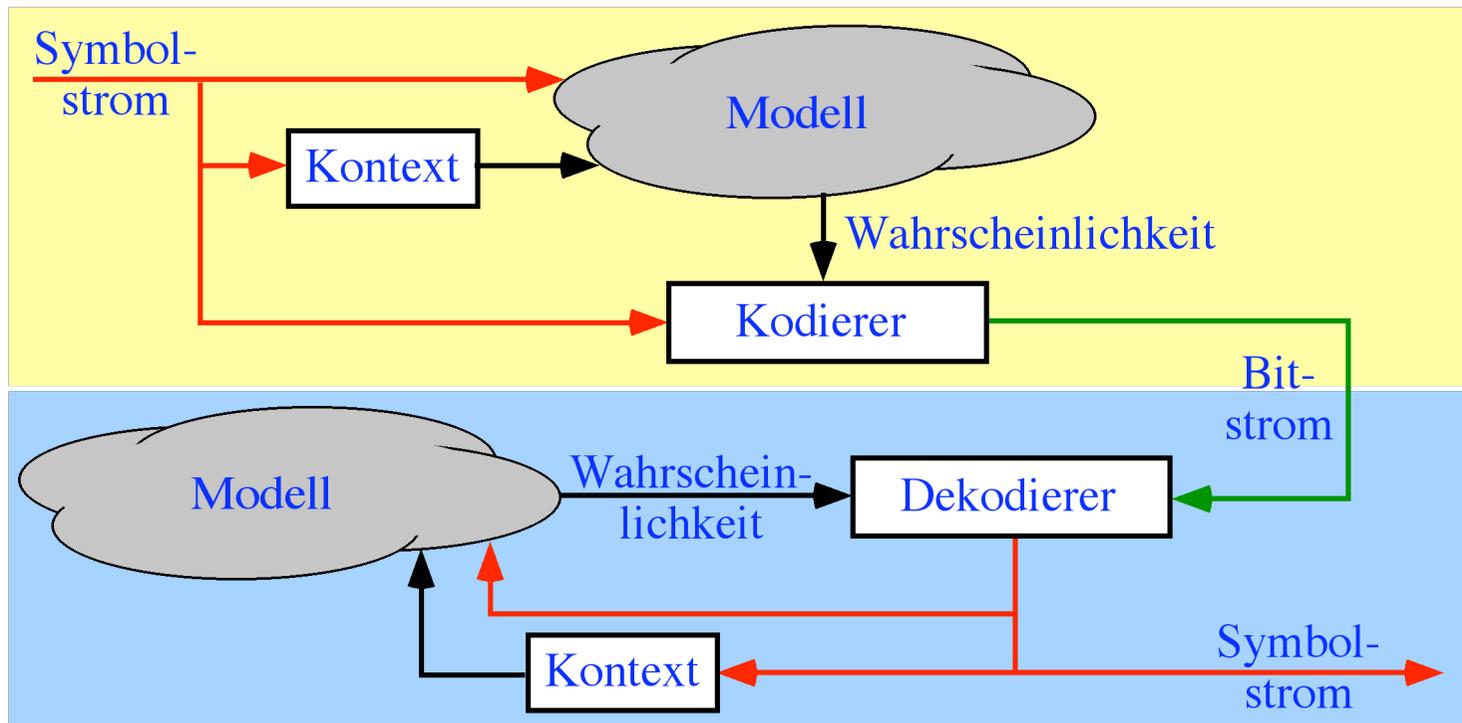
low: 0,10010001110110100

high: 0,10011011111010101

low: 0,10010001110110100000

high: 0,10011011111010101111

- Modell erneuern
  - cumFreq[theSymb] inkrementieren
  - cumFreq[theSymb] evtl. tauschen
  - alle Höheren inkrementieren
- Modell auch im Dekodierer mitführen



- Modelle höherer Ordnung möglich
  - Ordnung 2:  $256 \cdot 256 \cdot 256$  Einträge
  - Teil-Tabellen nur bei Bedarf dynamisch anlegen

- Adaptive Modellbildung möglich
- Kodelänge
  - Intervallgröße =  $p(\text{Nachricht})$
  - Repräsentant des Intervalls hat  $\log(p)$  Stellen =  $H(\text{Nachricht})$
  - Endesymbol+Auffüllen: 9 bit / Nachricht
  - Rundungsfehler  $< 10^{-4}$  bits/Symbol
  - Symbolzähler im Modell ist beschränkt: 0,25% / MByte
- $O(n_s)$  für Kodierung:
  - Zeichen lesen, low und high berechnen
  - c mal Bit ausgeben, shiften
- $O(n_b)$  für Dekodierung:
  - Bit in Code lesen, Code an Modell anpassen
  - Zeichen finden ( $O(m)$ )
  - low und high berechnen
- $O(m)$  für Verwaltung des Modells
- experimentell: 500 Instruktionen/Zeichen
- Kodieren:  $\sim 100$  MC68020 Instruktionen / Byte
- Dekodieren:  $\sim 120$  MC68020 Instruktionen / Byte

### 1.1.1.5 Bildkompression mit Farbtabelle

- Übertragung und Speicherung der Bilder im Index-Format
  - Indexlänge typisch 8 Bit
  - CLUT enthält echte Farbwerte, gehört zum Bild
- Kompression 2:1 oder 3:1 bei RGB
- Klassische Redundanzunterdrückung wieder sinnvoll (-> GIF: LZW)
- Originalbild eventuell "dithern":
  - Jeder Punkt wird auf eine Farbe in der Tabelle abgebildet
  - Fehler in einem Pixel mit umliegenden Pixeln ausgleichen
  - Gewichtsfunktion für Fehlerverteilung um ein Pixel

.	.	.	.	.	.	.
.	1	2	4	2	1	.
.	2	4	8	4	2	.
.	4	8	(i,j)	8	4	.
.	2	4	8	4	2	.
.	1	2	4	2	1	.
.	.	.	.	.	.	.

- Iterativer Prozess, rechenintensiv

- Algorithmus zur Tabellenermittlung für RGB-Bilder
  - Ausgangsbild z.B. 24 bit RGB
  - Abbildung "ähnlicher" RGB-Farben auf CLUT-Index
  - bestmögliche Auswahl der Farben
  - Parametrisierbar (Anzahl Indices, Abstand der Farben, ...)

- Octree Quantisierung

```

TYPE Clut = ARRAY [0..maxTabEntries] OF LongInt;
FUNCTION PixelsLeft : BOOLEAN; ...
FUNCTION GetPixel : LongInt; ...
PROCEDURE FindClosestTwo(VAR c:Clut; VAR i,j:INTEGER);
FUNCTION ReduceTable(VAR theClut : Clut):INTEGER;
    VAR i,j : INTEGER;
    BEGIN FindClosestTwo(theClut,i,j);
        theClut[i]:= FindMixCol(theClut[i],theClut[j]);
        ReduceTable:= j;
    END {ReduceTable};

BEGIN
nextfree:=0;tablefull:=false; ...
WHILE PixelsLeft DO BEGIN
    myClut[nextfree]:= GetPixel;
    IF nextfree >= maxTabEntries THEN tablefull:=true;
    IF not tablefull THEN nextfree:= nextfree+1
    ELSE nextfree:= ReduceTable(myClut);
END {while};
  
```

- Median Cut

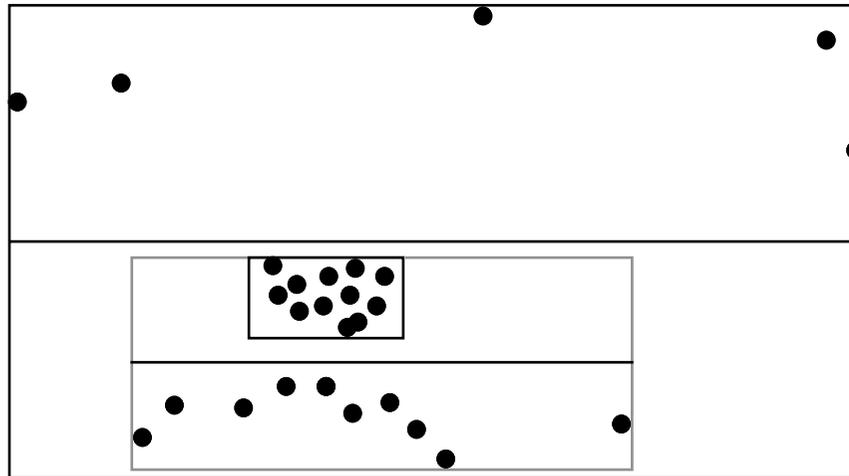
- Farbpunkte im dreidimensionalen Raum (Achsen R,G,B)

- Einteilung des Farbraumes in  $m$  Kästchen

- die alle Punkte enthalten

- Kästchen enthalten ungefähr gleichviele Punkte

- das jeweils vollste Kästchen wird entlang der längsten Achse halbiert



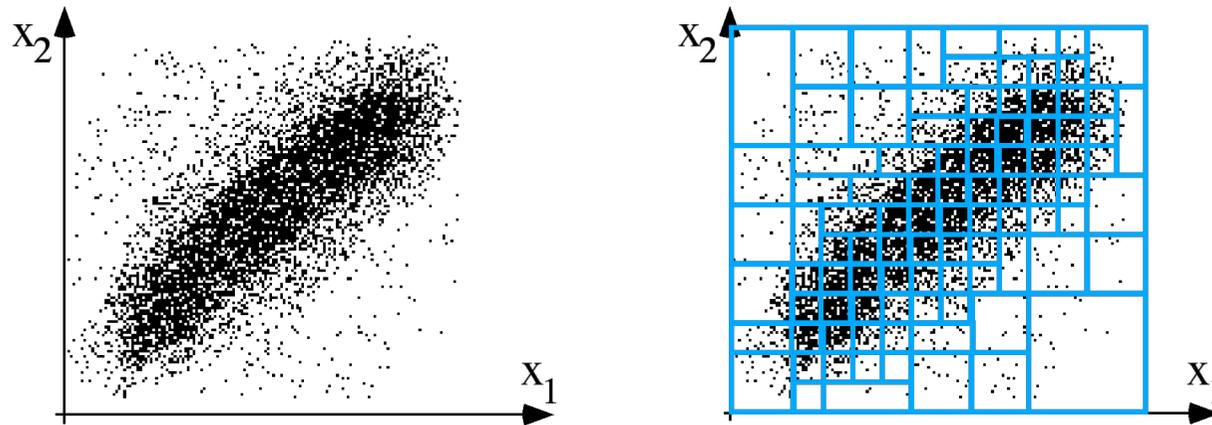
- ein Wert repräsentiert Kästchen (also "ähnliche" Farben)

- Nachbehandlung

- $m+n$  Tabelleneinträge bilden

- Reduzierung um  $n$  Tabelleneinträge nach besonderen Kriterien

- Verallgemeinert: Codebuch ( $\Rightarrow$  Vektorquantisierung)
  - Wertetupel  $(x_1, x_2, \dots, x_n)$  (Vektoren)
  - auf  $k$  Repräsentanten abbilden (quantisieren)
  - $\text{card}(x^n) \gg k \Rightarrow$  Kompression



- Bewegtbildkompression mit Farbtabelle(n): asymmetrisch
  - Film wird mit Farbtabelle(n) gespeichert
  - einfache Dekompression im Anzeigegerät: CLUT laden
- Drei Strategien zur Tabellenverwaltung:
  - feste Farbtabelle für gesamte Sequenz: einfach, Ergebnisse schlecht
  - neue Tabelle für jedes Bild: Farbtabellenumschaltung problematisch
  - Adaptive Farbtabelle mit kleinen Veränderungen von Bild zu Bild

## 1.1.2 Standbildkompression

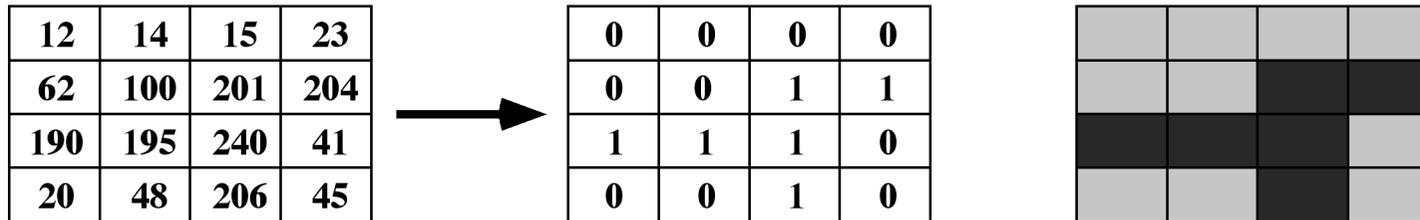
### 1.1.2.1 Blockcodiertechniken

- Einfache Verfahren ohne Floating-Point
- Kompromiß Leistung vs. Systembelastung
- Subsampling: 4:2:2, 4:1:1
- Ausnutzen vorhandener (Grafik-) Beschleuniger
  - z.B. Cell-A, Cell-B (Sun)
  - Idee: Bitmaptransfer mit Vordergrund und Hintergrundfarbe
- Erhalten gewisser Bildeigenschaften:
  - Mittelwert
  - Standardabweichung
  - Grobstrukturen

## 1.1.2.1.1 Block Truncation Coding

(Mitchell, Delp, Carlton; Kishimoto, Mitsuya, Hoshida; 1978)

- Für Graustufenbilder
- Einteilen eines Bildes in 4\*4 Pixel Blöcke



- Mittelwert  $m = \frac{1}{n} \sum_{i=0}^{15} p_i$  (Im Beispiel 101)
- Mittelwerte  $m_{low} = \frac{1}{\text{card}(p_i | p_i < m)} \sum_{i=0, p_i < m}^{15} p_i$  (Bsp. 38)  
 und  $m_{high} = \frac{1}{\text{card}(p_j | p_j \geq m)} \sum_{j=0, p_j \geq m}^{15} p_j$  (Bsp. 206)
- Eventuell andere Blockgrößen und nur ein Grauwert  
 - Varianz berechnen

- Code für Block (bitmap, mlow, mhigh)
  - 32 bit
  - 2 bit / Pixel
  - Bitmap beschreibt Form
- Auswahl von m, mlow, mhigh optimieren: Momente
  - Kontur-erhaltend
  - Helligkeit und Kontrast erhalten
  - andere Wahl des 'Unterscheidungswertes' m
- Mittelwert und Varianz übertragen
  - gemeinsame Kodierung von Mittelwert und Varianz in 6 bit
  - => 22 bit pro Block: 1,375 bit/Pixel



Original



BTC



Differenz

## 1.1.2.1.2 Color Cell Compression (Campbell, 1986)

- Verallgemeinerung auf Farbbilder
- 4\*4 Blöcke
- Luminanz-Mittelwert als Quantisierer
- 2 24-bit Werte für 0 und 1
  - RGB Komponenten der Repräsentanten aus Farbebenen ermitteln
  - Code für Block (bitmap, vlow, vhigh) => 64 bit
  - 4 bit / Pixel
- 24 bit Werte auf Farbtabelleindizes abbilden
  - Median-Cut wie oben
  - Code für Block (bitmap, ilow, ihigh) => 32 bit
  - 2 bit / Pixel
- Farbtabelle pro Bild nötig
- Weitere Kompression siehe SMP (Software Motion Pictures)
  - subsampling
  - Monocolor Blocks

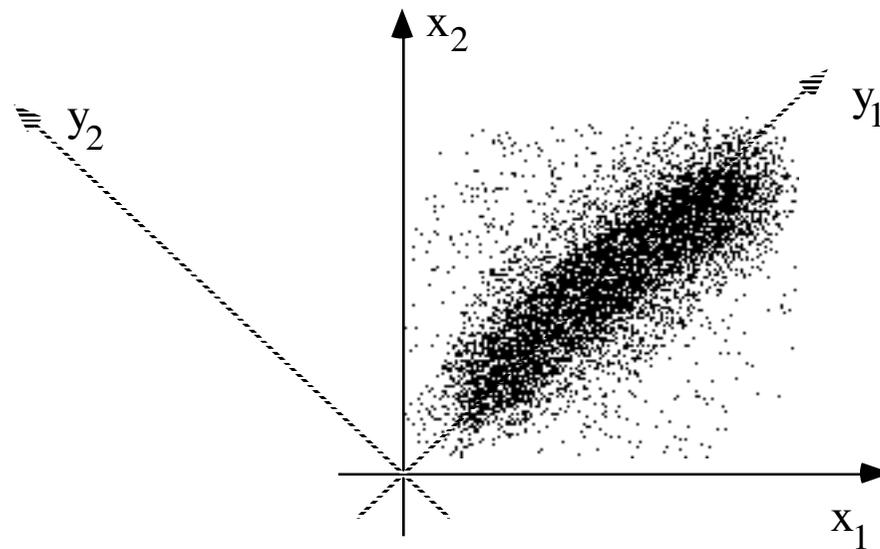
## 1.1.2.2 Transformationskodierung

- Interpretation der Transformation

- Drehung

- asymmetrische Varianz bezüglich Achsen

- Gesamtvarianz bleibt erhalten



- Darstellung mit Basisfunktionen

- Synthetisierung eines Punkteblocks als Linearkombination von BF  
z.B. Fourier-Transformation (FFT)

- Diskrete Cosinus Transformation
  - zweidimensional für  $n \times n$  Block

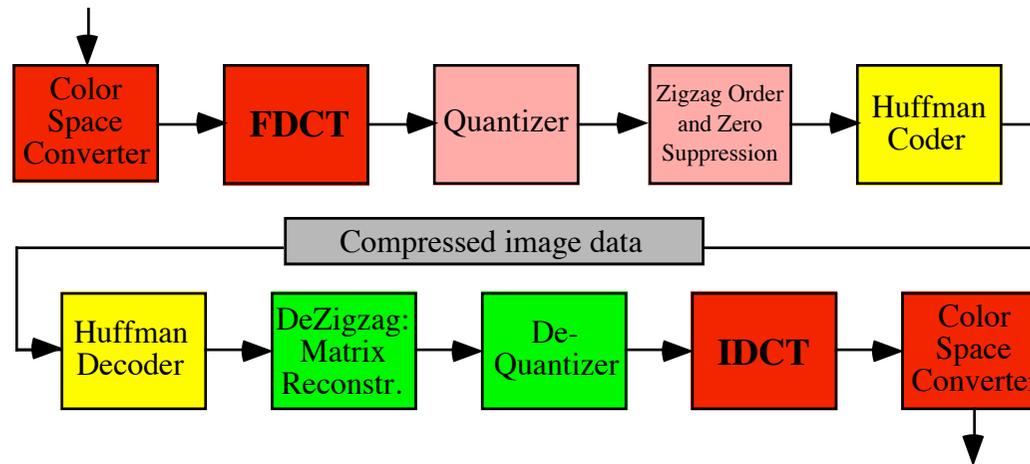
$$F(u, v) = \frac{2}{n} C(u) C(v) \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} f(j, k) \cos\left(\frac{(2j+1)u\pi}{2n}\right) \cos\left(\frac{(2k+1)v\pi}{2n}\right)$$

$$f(j, k) = \frac{2}{n} \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} C(u) C(v) F(u, v) \cos\left(\frac{(2j+1)u\pi}{2n}\right) \cos\left(\frac{(2k+1)v\pi}{2n}\right)$$

$$C(w) = \begin{cases} \frac{1}{\sqrt{2}} & w = 0 \\ 1 & w = 1, 2, \dots, n-1 \end{cases}$$

- DCT ist Orthogonale Transformation:  $C = Q^T P Q$ 
  - $2 * 64 * (8 \text{ Mult} + 7 \text{ Add})$
  - Schnelle Algorithmen ähnlich FFT existieren
    - 2n FFT
    - Generalized Chen Transform
    - Spalten/Zeilenweise Ausführung: 1D-DCTs

# 1.1.2.2.1 JPEG Joint Photographic Expert Group



- JPEG-DCT

- 8\*8 Punktematrix
- Pro 'Farbebene' (RGB, CMYK, YUV)
- Wert links/oben wird als Gleichstrom-Koeffizient bezeichnet
- Andere Werte 'Wechselstromkoeffizienten'

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

DCT →

<b>1260</b>	-1	-12	-5	2	-2	-3	1
-23	-17	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	1	1	0	0	0
-1	-1	1	2	0	-1	1	1
2	0	2	0	-1	1	1	-1
-1	0	0	-1	0	2	1	-1
-3	2	-4	-2	2	1	-1	0

- Umquantisierung -> Kompression, Verlust an Genauigkeit
  - Normalisierung
  - Quantisierungsschritt nimmt mit hohem Index zu

1260	-1	-12	-5	2	-2	-3	1
-23	-17	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	1	1	0	0	0
-1	-1	1	2	0	-1	1	1
2	0	2	0	-1	1	1	-1
-1	0	0	-1	0	2	1	-1
-3	2	-4	-2	2	1	-1	0

Quant →

79	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- Quantisierungsmatrizen [Lohscheller, 1984]:

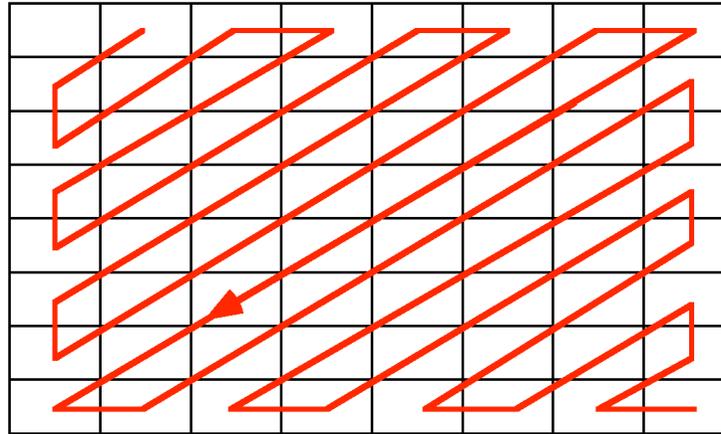
Luminanz (Y)

16	11	10	16	24	40	51	61
14	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Chrominanz (U,V)

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

- Linearisierung



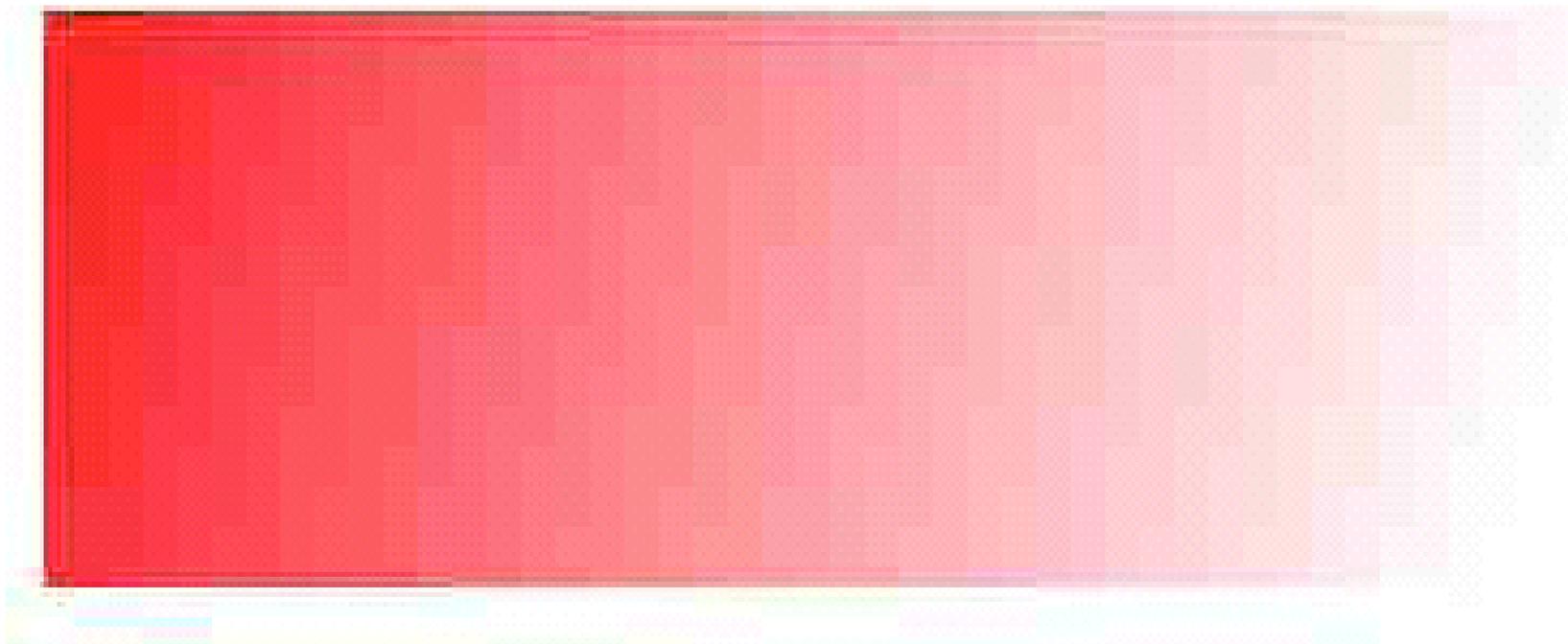
79; 0 -2 -1 -1 -1 0 0 -1 Blockende

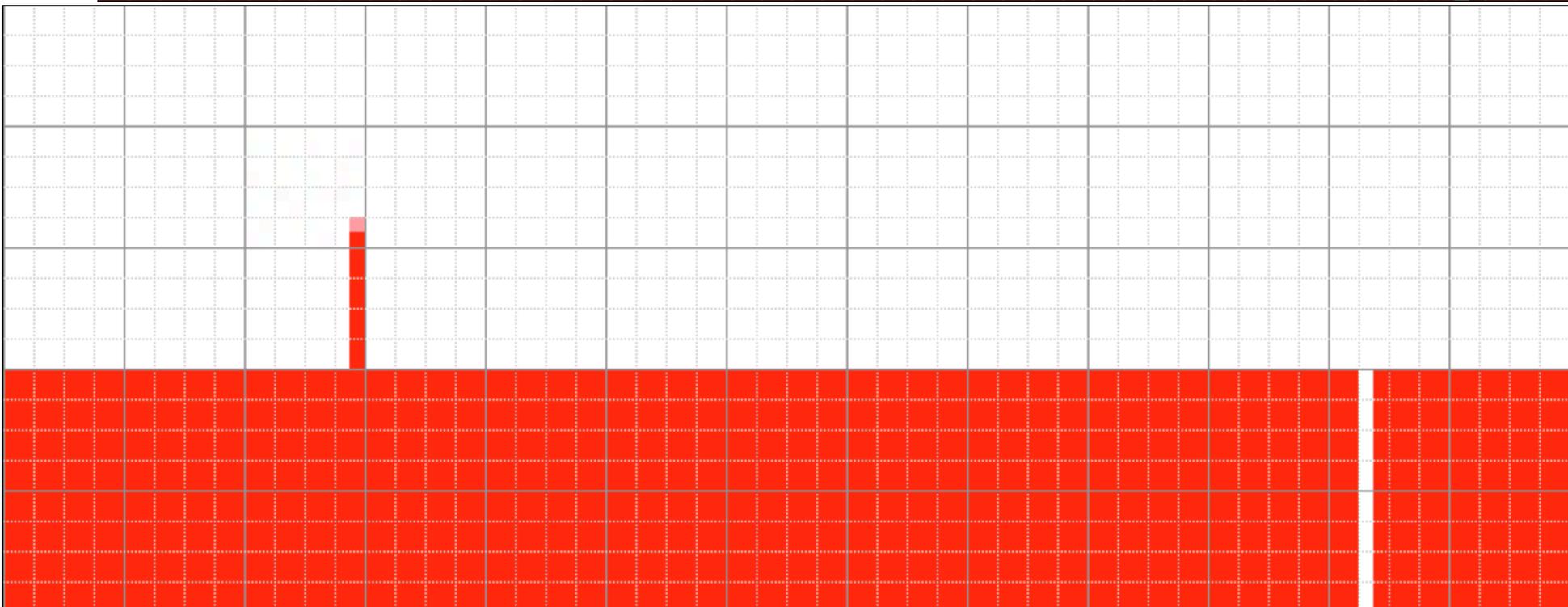
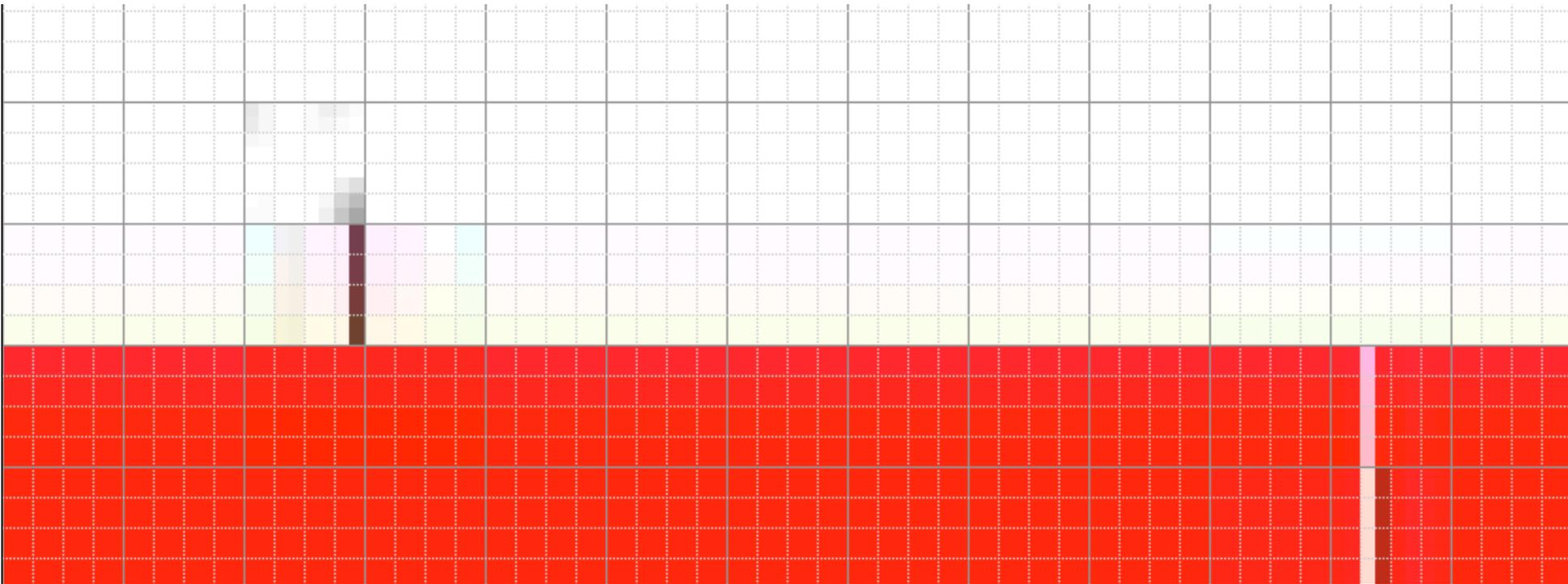
- Lauflängenkodierung + Huffman-Kodierung zur weiteren Kompression
- JFIF - JPEG File Interchange Format
  - optimierte Quantisierungsmatrizen
  - optimierte Huffman-Tabellen
  - YUV 4:1:1 oder 4:2:2
- JPEG/M zur Bewegtbildkompression
  - jedes Bild selbständig JPEG-komprimiert

- Bit/Pixel 0.3 Qualität    mittel, Blöcke sichtbar  
0.5                            ordentlich  
0.75                           sehr gut  
1.5                            hervorragend  
2.0                            kein Unterschied zum Original erkennbar

- Typische JPEG-Artefakte

- Blockbildung (nur eine Mischfarbe pro Block)
- 'Ausschwingen' bei scharfen Farbgrenzen
- Beispiel Rotkeil 1: 104 komprimiert





## 1.1.2.2.1.1 JPEG-Varianten

- Sequential Mode
- Progressive Mode (Expanded Lossy DCT-based Mode)
  - sukzessive Bildverfeinerung
  - Änderung der Quantisierung:
    - Einteilung in Koeffizientenklassen (spektrale Auswahl)
    - Bitklassen in Koeffizienten (successive approximation)
- Arithmetische Kodierung
  - statt Huffman-Tabellen
  - 5% - 10% kompakter
- 12 bit/Pixel
- Lossless Mode: Prädiktion aus Umgebungspixeln + Differenz

0	keine Prädiktion
1,2,3	$X=A, X=B, X=C$
4	$X=(A+B+C)/3$
5,6	$X=(A+(B+C)/2)/2, X=((A+B)/2+C)/2$
7	$X=(A+B)/2$

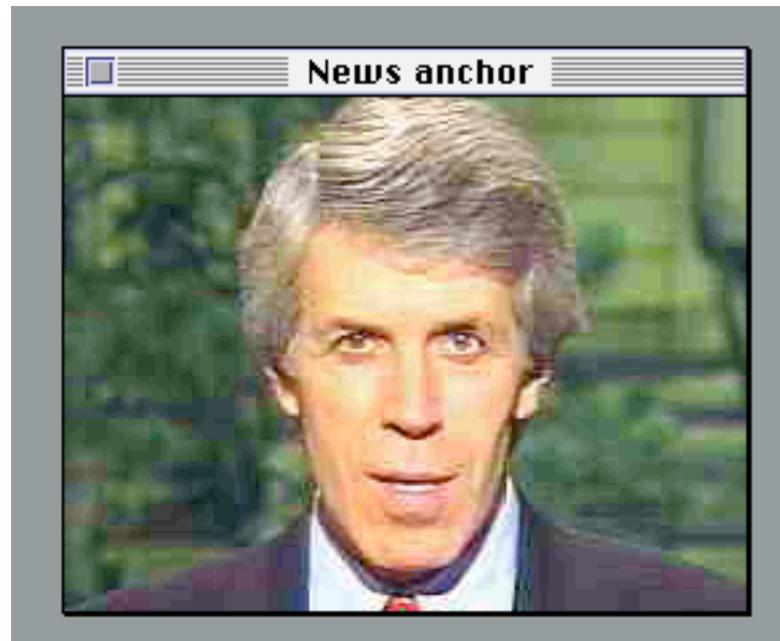
	C	B	
	A	X	

- Hierarchischer Modus: sequentielle Auflösungsverfeinerung

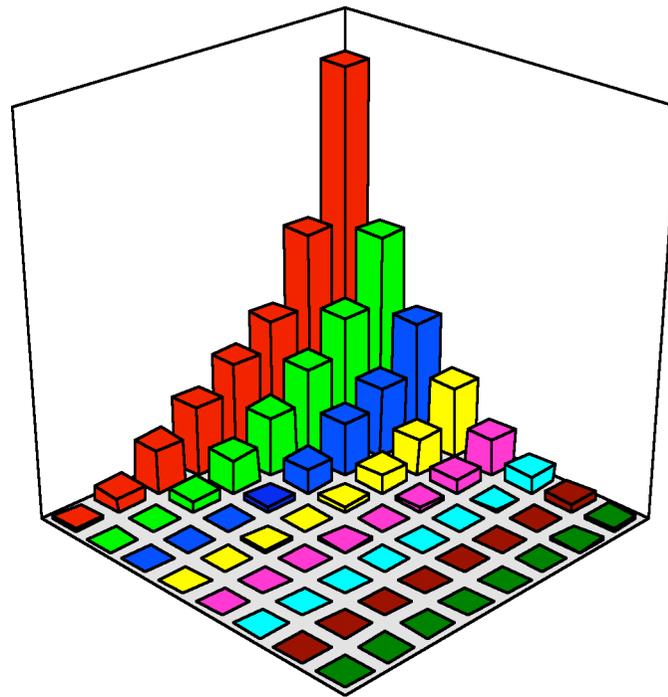
## 1.1.2.2.1.2 Optimierung

- Statistische Betrachtungen
- Hohe Kompression  $\Leftrightarrow$  Viele Elemente = 0

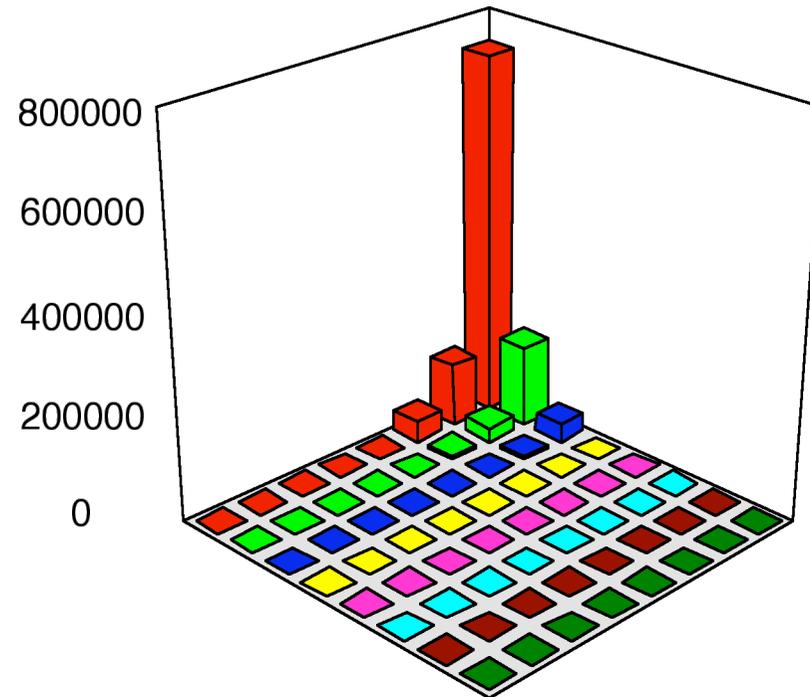
	News 1	News 2	Office	Movie	Text
Y	7.8 %	9.7 %	8.0 %	12.6 %	22.4 %
U/V	2.2 %	2.3 %	2.4 %	2.1 %	1.5 %



- Elemente  $\neq 0$  in der linken oberen Ecke

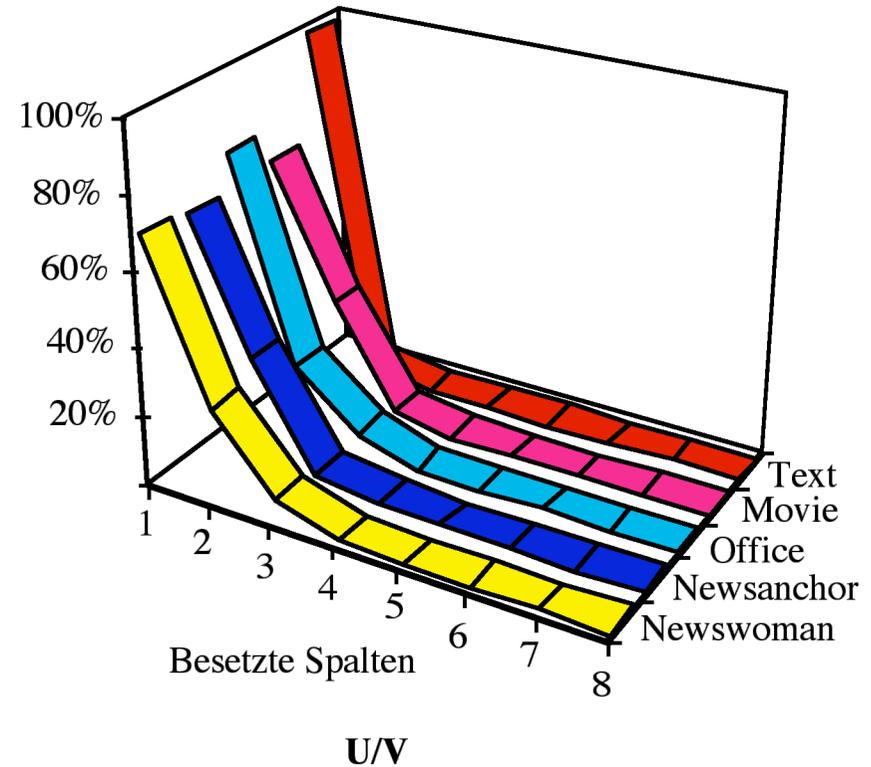
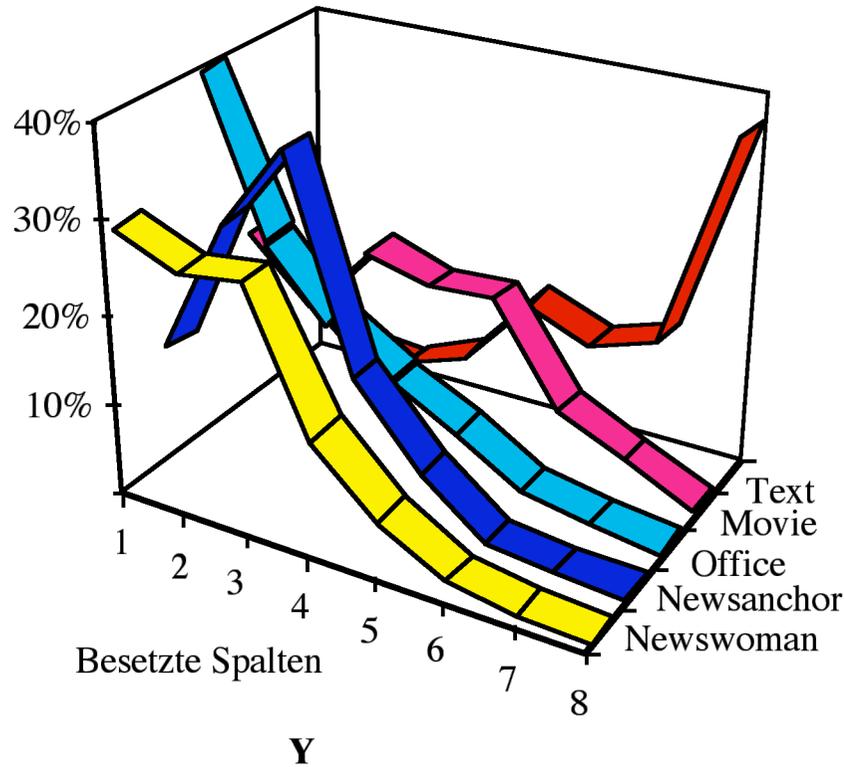


Luminanz (Y)



Chrominanz (U, V)

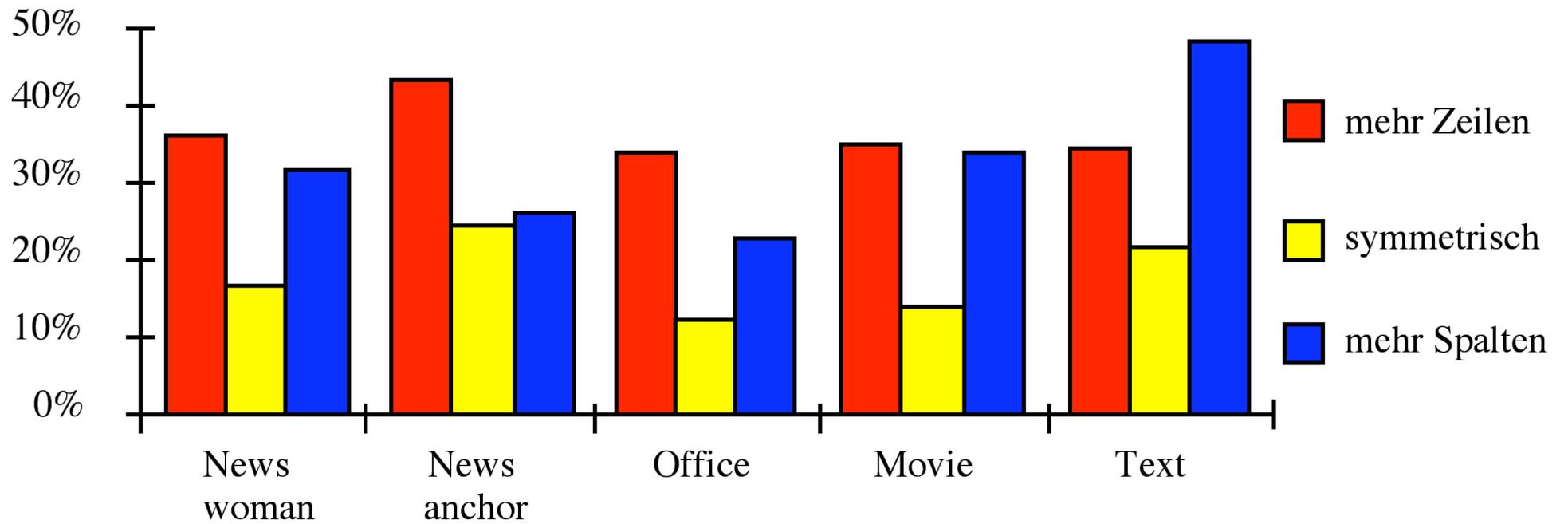
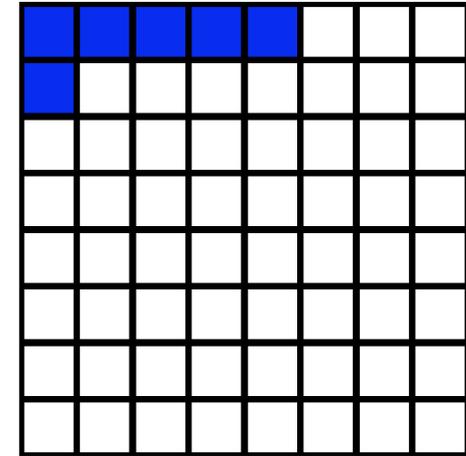
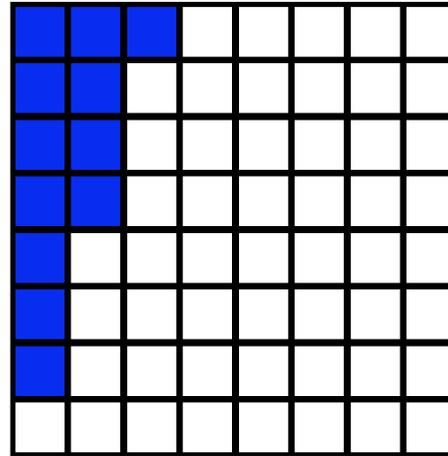
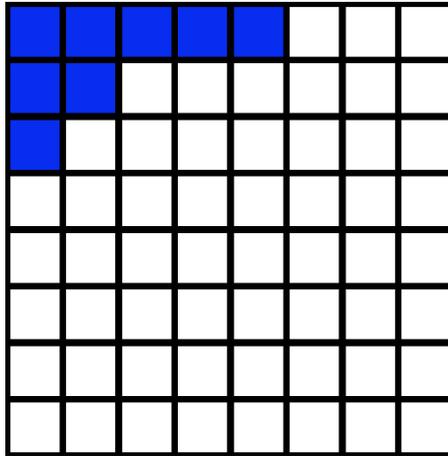
- Leere Zeilen und Spalten



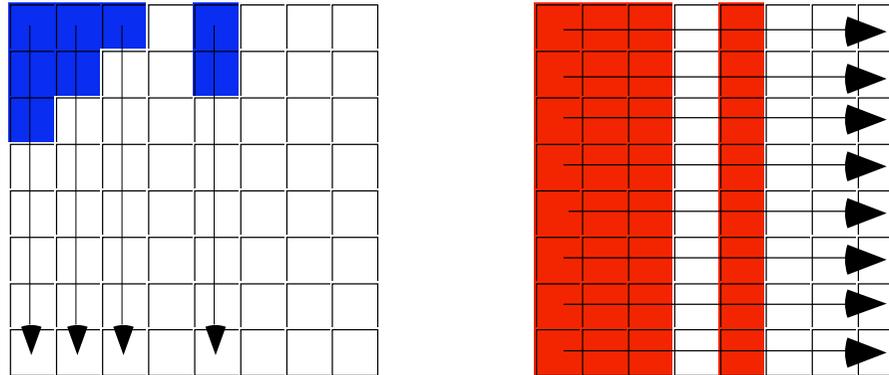
- Beispiele

	Bit/Pixel	Y	U/V
Telekonferenz	0,14	8	2
Video	0,23	10	3
Standbild	0,5	15	4

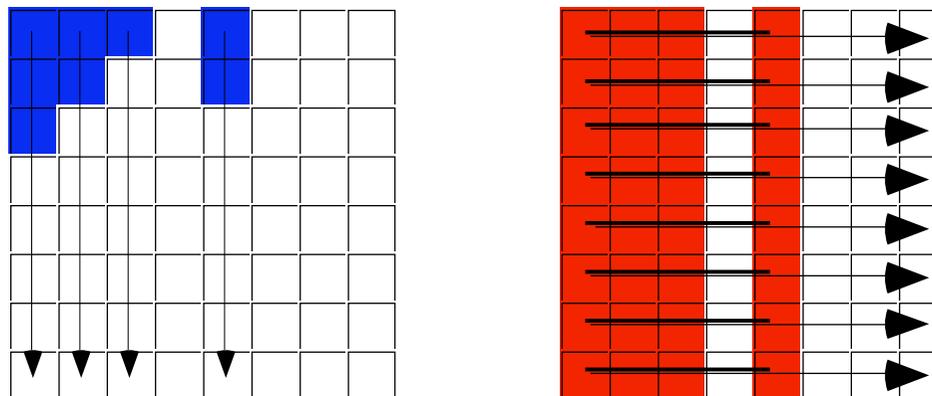
- Asymmetrien



- IDCT algorithm:
  - 1D (Chen)
  - Multiplikationen  $\sim$  Additionen
- Erste Optimierung: IDCT nur in nichtleeren Spalten/Zeilen

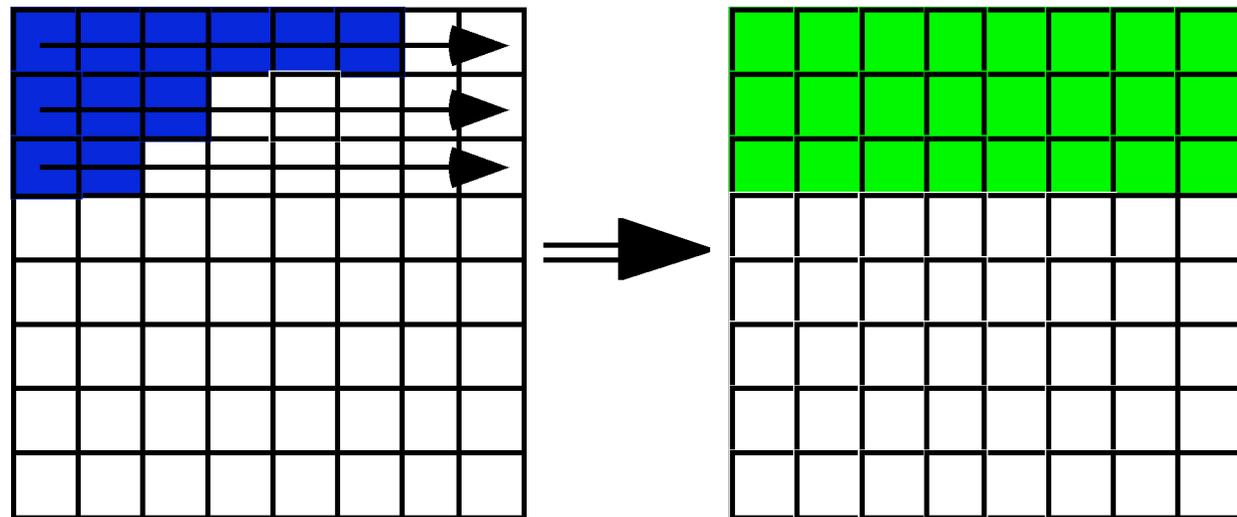
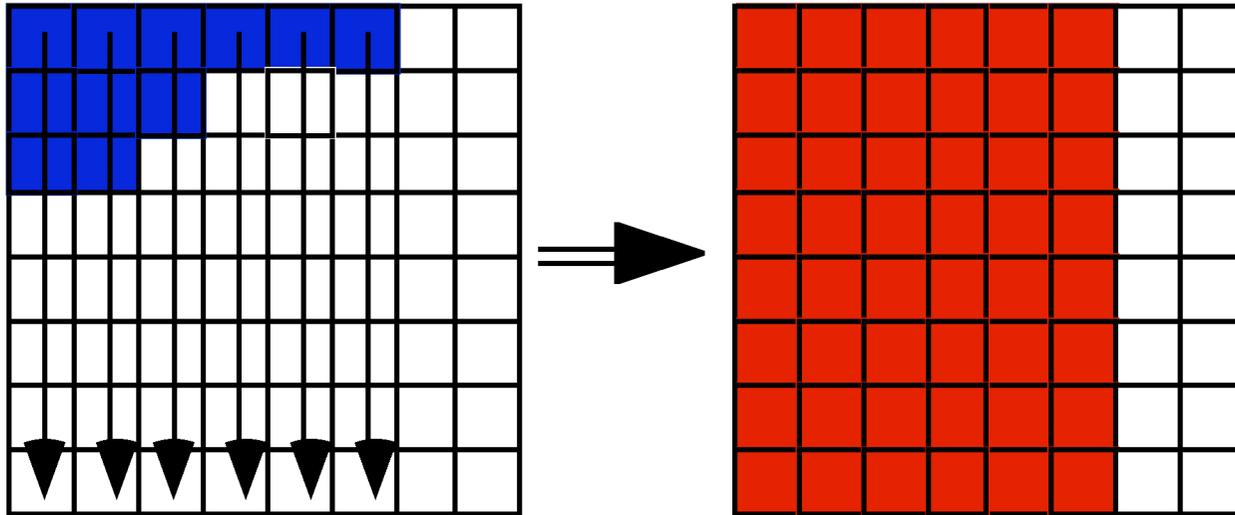


- Zweite Optimierung : 1D IDCT mit beschränkter Länge
  - Grad  $r$  einer Transformation: Index des letzten Elementes  $\neq 0$

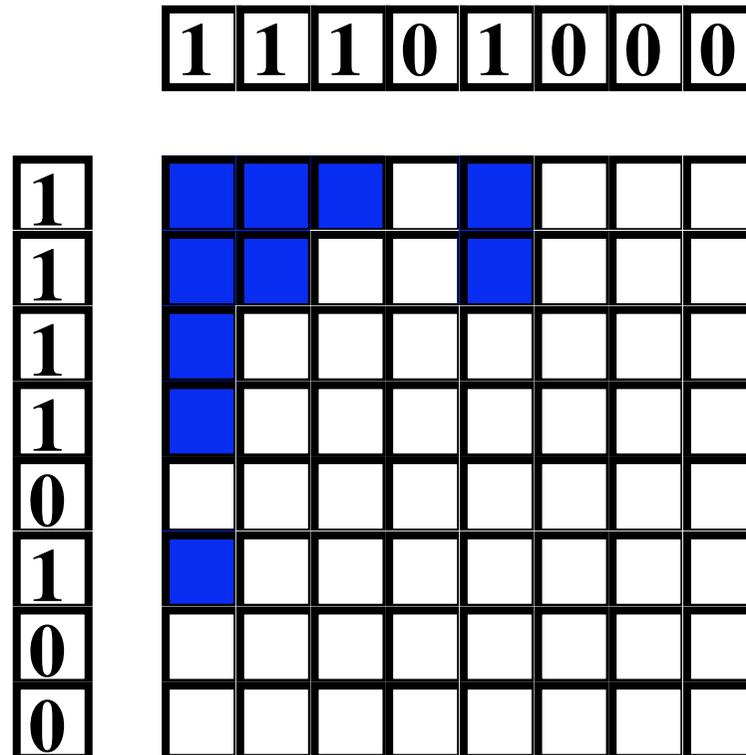


- Dritte Optimierungsstufe: Matrix dünnbesiedelt halten

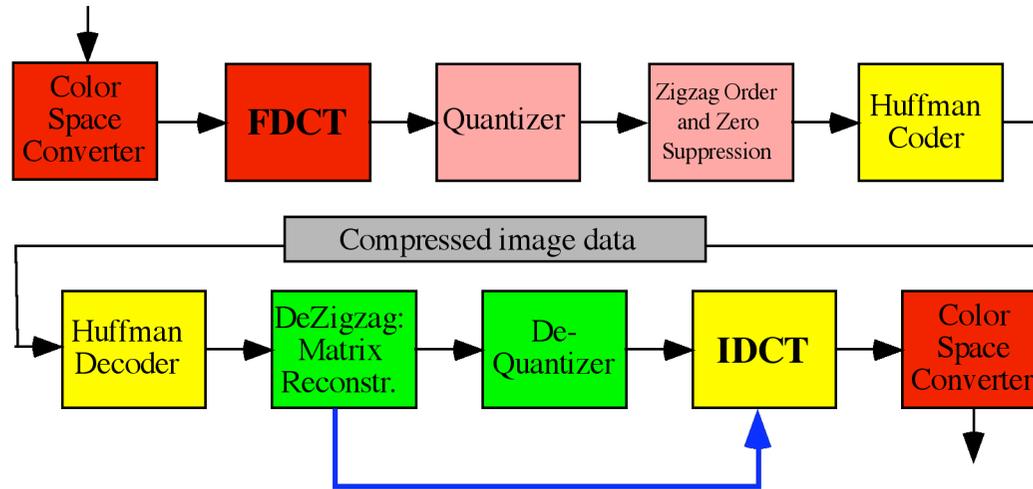
$$P = Q^T C Q = (Q^T C) Q = Q^T (C Q)$$



- Wie erhält man die Struktur-Information?
  - Nullen in der Matrix zählen ist teuer
  - Compare kostet genauso wie Compute
  - Konstanter Overhead
- Effizient: Huffman decode / matrix reconstruction
  - Zeilen- und Spalten-Bitsets
  - Overhead skaliert mit Kompression



- revidierter Prozeß



- MC68040 und DSP AT&T 3210, heute: PPC 601

- JPEG Strombearbeitung in Pascal auf dem Hauptprozessor,
- Huffman, Matrix Rekonstruktion, und
- IDCT in 3210 Assembler auf 66 MHz AT&T 3210,
- YUV -> RGB in 68040 Assembler

- DOnly: 64 ops um vollständige Matrix zu erzeugen
- 1D IDCT auf leeren Spalten einsparen
- 1D IDCT mit Grad r implementieren

Grad	1	2	3	4	5	6	7	8
3210-Ops	12	27	33	37	42	44	46	48
PPC-Ops	17	28	33	38	40	43	46	49

- Information ist während der Matrix Rekonstruktion vorhanden:

```

C[i,j] := coefficient;
Set(rows[i]);
Set(cols[j]);

```

- skaliert mit Kompressionsfaktor

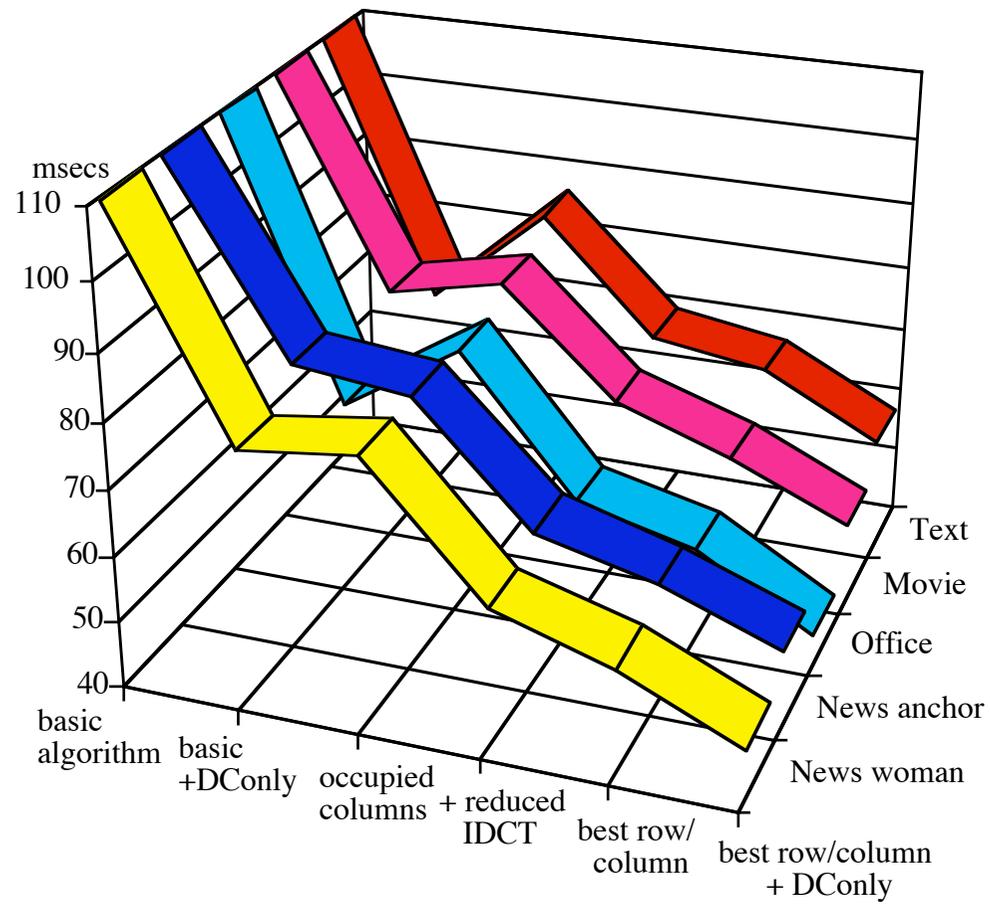
- Erste Richtung wählen

```

IF rows < cols THEN BEGIN
  rowtransform; coltransform
END
ELSE BEGIN
  coltransform; rowtransform
END;

```

# • Beschleunigung der IDCT



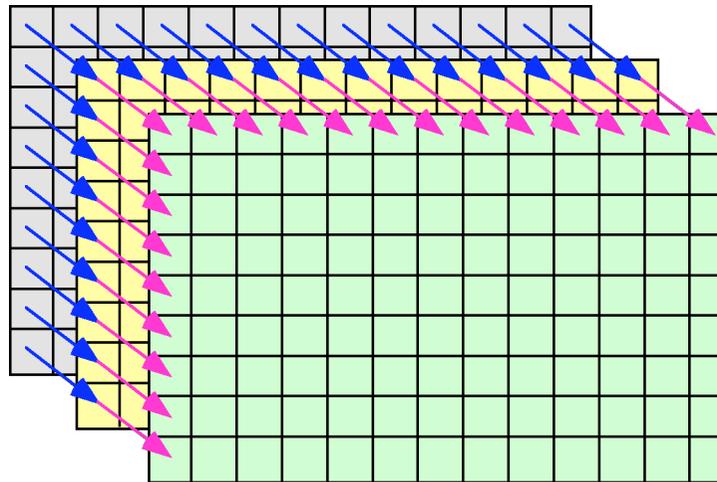
- Gesamtbeschleunigung

Framerate [frames/s]	News 1	News 2	Office	Movie	Text
Basis Algorithmus	7.6	7.3	7.5	7.0	6.1
Basis + DOnly	9.9	9.0	10.8	8.9	7.9
Nur besetzte Spalten	9.7	9.2	9.6	8.6	7.0
Nur besetzte Spalten + reduziert IDCT	11.8	10.9	12.0	9.9	8.0
Auswahl Zeile/Spalte	12.6	11.5	12.6	10.6	8.1
Auswahl Zeile/Spalte + DOnly	14.1 (86%)	12.5 (70%)	14.5 (92%)	11.5 (65%)	8.8 (45%)

- Auf 80 MHz PowerPC > 18 fps

### 1.1.2.2.1.3 Heuristische Beschleunigung der FDCT

- Reduzierte IDCT wird aus Entropiedekodierung abgeleitet
  - Bitmap beschreibt Blockstruktur
  - Blockstruktur als Ergebnis der FDCT
- Woher bekommt man Blockstrukturinformation bei der FDCT?
  - garnicht bei der Einzelbildkodierung
  - bei M-JPEG vom vorhergehenden Bild
  - nicht immer zuverlässig

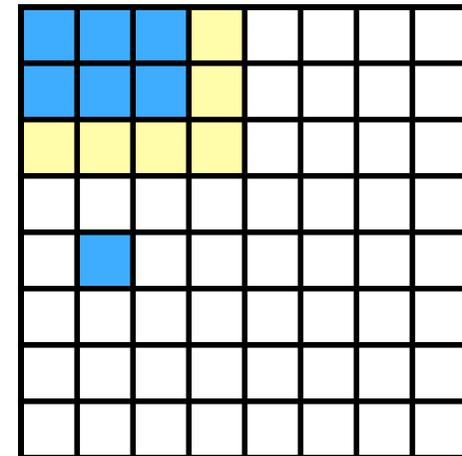
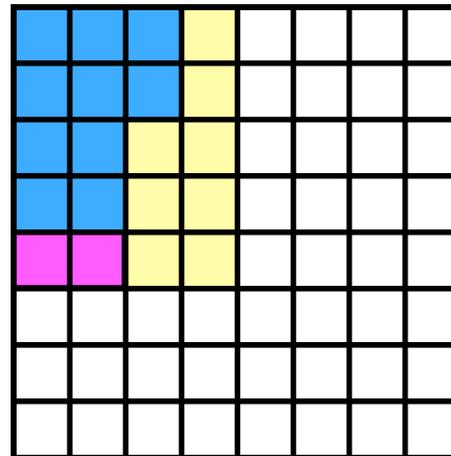
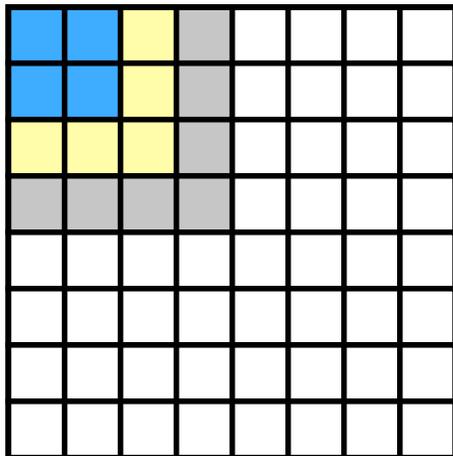
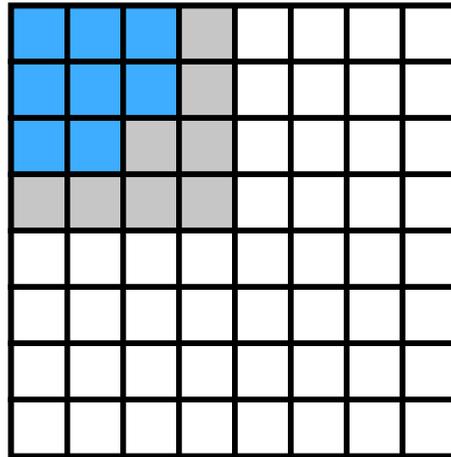


- Einsparpotential
  - FDCT (siehe IDCT)
  - Quantisierung: ca. 50%
  - Zig-Zag und Huffman geringfügig
- Experimentelle Ergebnisse
  - Beschleunigung der gesamten Kompression (inkl. RGB -> YUV)

	Mittel	Min	Max
Graustufen/Video	45%	28%	
Graustufen/Konferenz	52%		77%
RGB/Video	27%	22%	
RGB/Konferenz	30%		35%

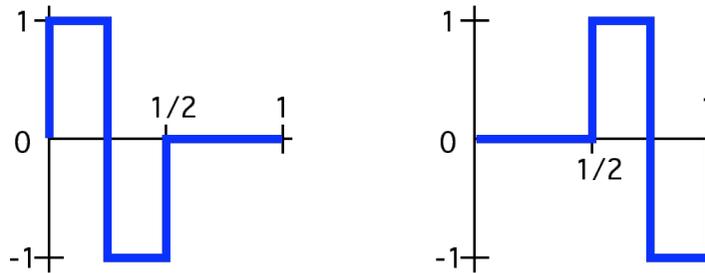
- erzeugtes Rauschen < 2dB
- Blockstruktur
  - Blockstruktur kann sich ändern
  - Block zu klein => Bild verfälscht
  - Block zu groß => Rechenzeit vergeudet
  - Überwachung einbetten in Entropiekodierphase

- Überwachung mit Pufferzone: ein oder zwei Zeilen bzw. Spalten
  - Block erweitern falls Pufferzone  $Z_i, S_i$  nicht leer
  - Block schrumpfen, falls  $Z_{i-1}, S_{i-1}$  leer
  - Singuläre Koeffizienten?



## 1.1.2.2.2 Wavelet-Kompression

- DCT modelliert Pixelwertkurve mit Cosinus-Summen
  - feste Blockstruktur lästig
  - isolierte Spitzen erfordern lange Summen
  - DFT/DCT eigentlich unendlich
- Wavelets verallgemeinern Ansatz
  - Funktionenraum mit besonderen Eigenschaften
  - Haar



- Daubechies, Shannon, Meyer, Koifman, Morlet
- Splines (biorthogonal, semiorthogonal), Maxflat, Binary biorth.
- Multiresolution

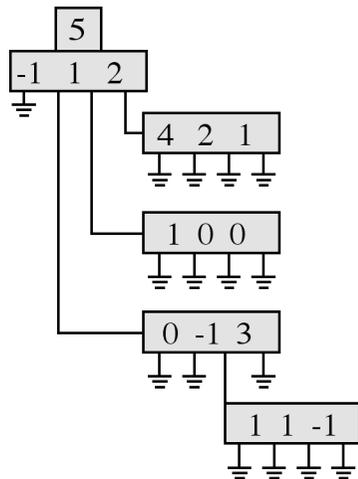
- Wavelet-Analyse mit Haar-Wavelets

- Mittelwerte bilden und speichern  $(6,4) + (-3,3,1,-1) \Rightarrow (6,4) + (-3,1)$
- Differenzen (Detail) in freien Plätzen speichern

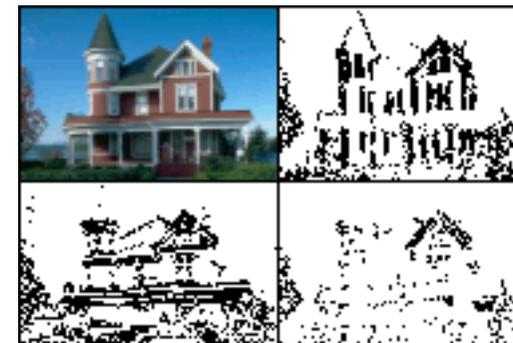
$$\begin{bmatrix} 3 & 9 \\ 5 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 6 & -3 \\ 4 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & -1 \\ 1 & -2 \end{bmatrix}$$

- Differenzen quantisieren bzw. mit weniger Bits kodieren

- Wavelet-Baum



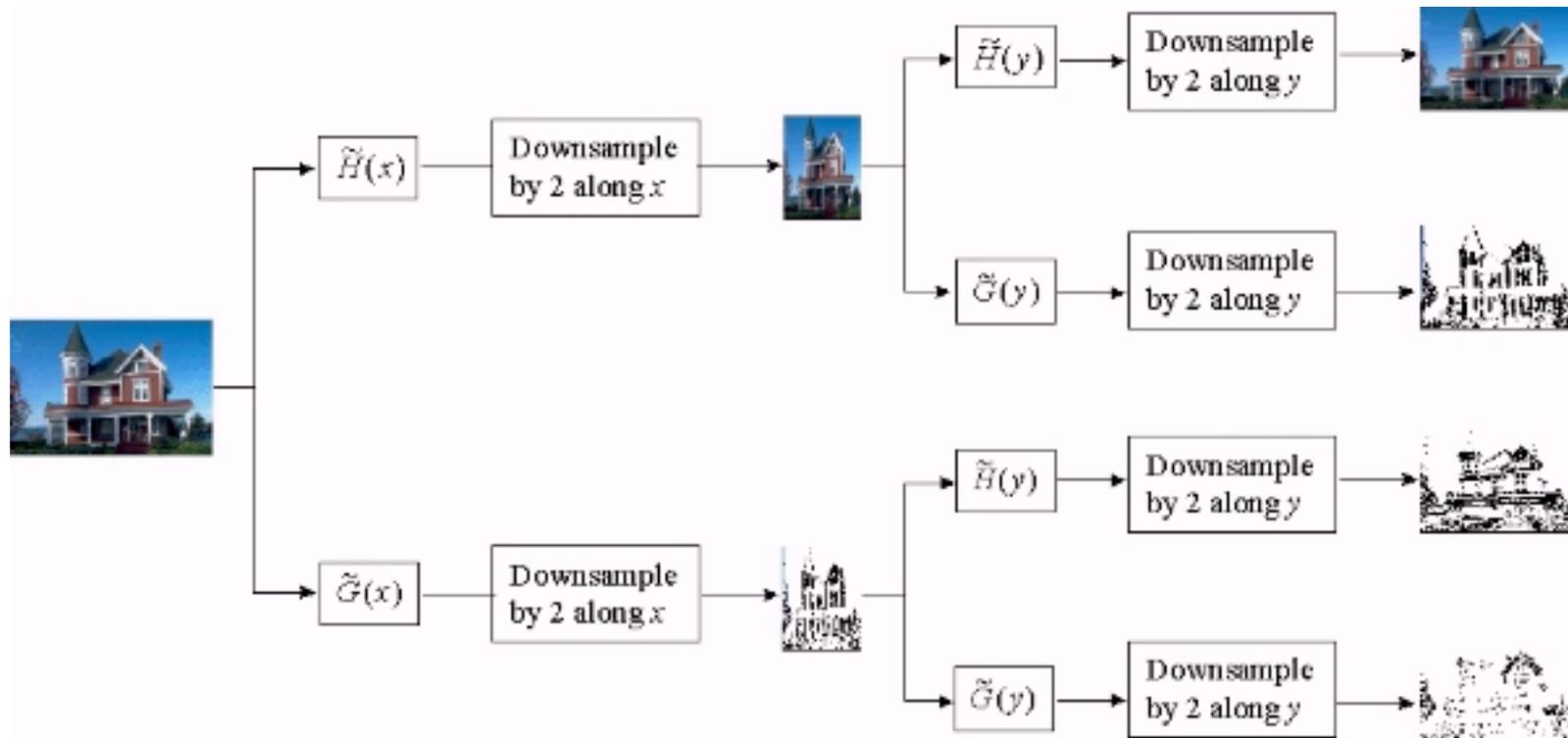
3		11		5
		8	8	13
		8	4	
6	4	10	0	
6	4	4	-2	



- Abschneiden = Verlust einführen
- Anpassen des Abschneidens an Randbedingungen
- unterschiedliche Auflösungen möglich

- Filter
  - Tiefpassfilter erzeugt Durchschnitt  $(6,6,4,4)$ : Scaling
  - Hochpassfilter erzeugt Differenzen  $(-3,3,1,-1)$ : Wavelet
  - Unterabtastung mit Faktor 2  $(6,4) + (-3,1)$
- Filterbank
  - Tiefpass und Hochpass
  - teilt Signal in Frequenzbänder
  - vervielfachen Information
  - downsampling
- Synthese umgekehrt
  - upsampling durch Einfügen von 0  $(6,4)+(-3,1) \rightarrow (6,0,4,0)+(-3,0,1,0)$
  - Additions-Filter und Subtraktions-Filter
  - Orthogonale Transformation  $\Rightarrow$  Synthese = Analyse<sup>T</sup>
- Transformationen
  - neue Repräsentation des Signals
  - verlustfrei: orthogonal
  - invertierbar: biorthogonal

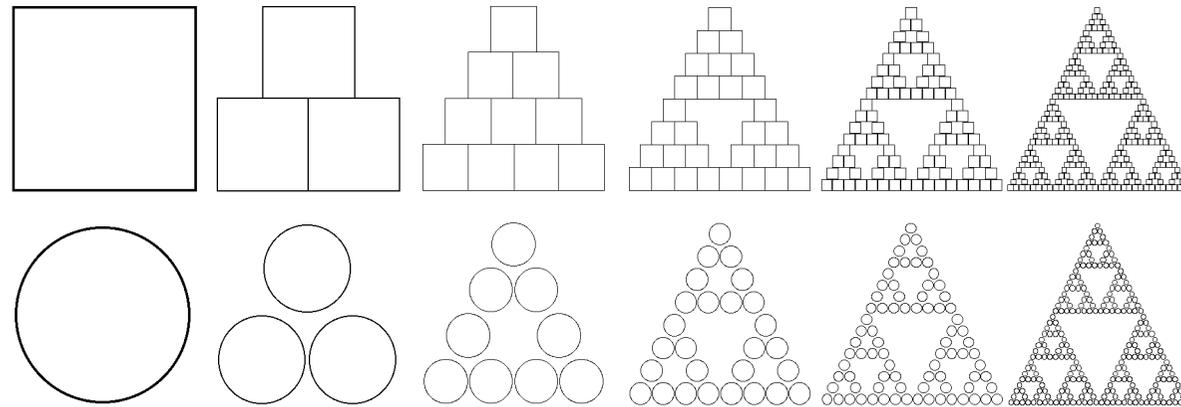
- Kompression: Iteration Lowpass/Highpass+Downsample



- dynamische Bitzuordnung an Bänder
- RLE + Huffman

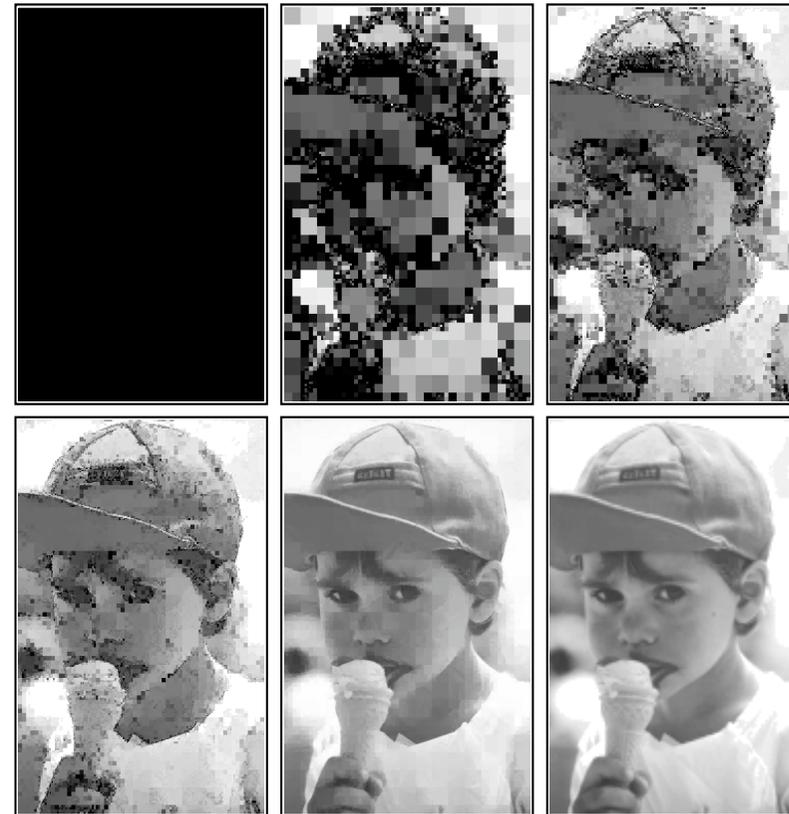
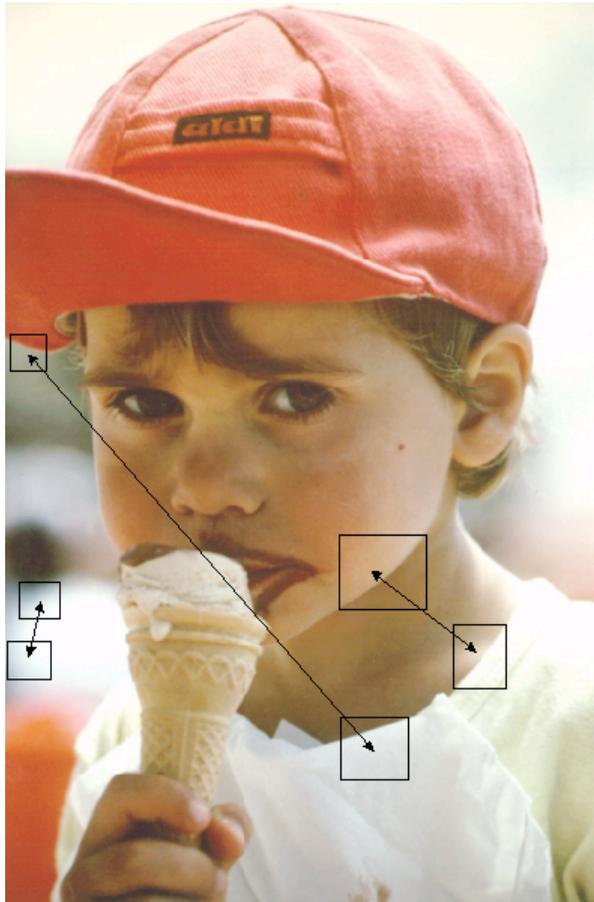
### 1.1.2.2.3 Fraktale Bildkompression

- fraktale Geometrie
  - iterierte Funktionensysteme
  - erzeugen beliebige Bilder
  - Konvergenz gegen Attraktor
  - Selbstähnlichkeit



- Kompression = inverses Problem
  - Funktionensystem finden, auf Bild anwenden
  - Verfahren konvergieren oft nicht
  - Bild erkennbar, aber mit hohem Rauschen
- Abkehr von der reinen Lehre der Fraktale

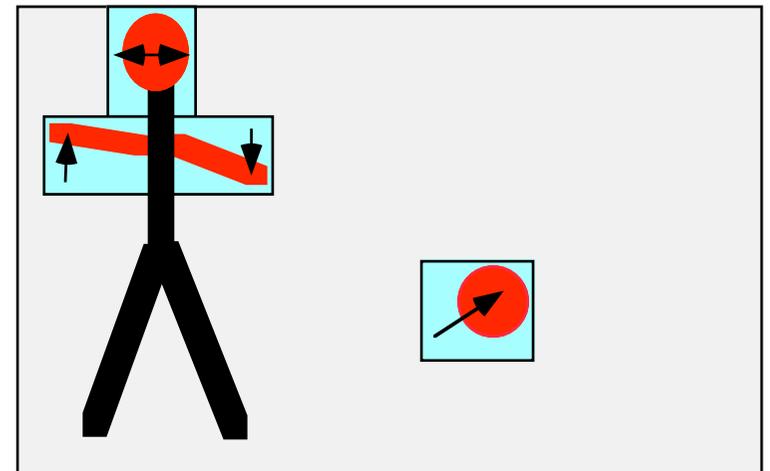
- Kompression: Ähnlichkeit zwischen Bildteilen suchen
  - Domain-Blöcke auf Range-Blöcke verkleinern
  - Helligkeit, Kontrast, Größe, Winkel verändern



- Dekompression: Iteration der Abbildung
  - Anwendung auf beliebiges Ausgangsbild

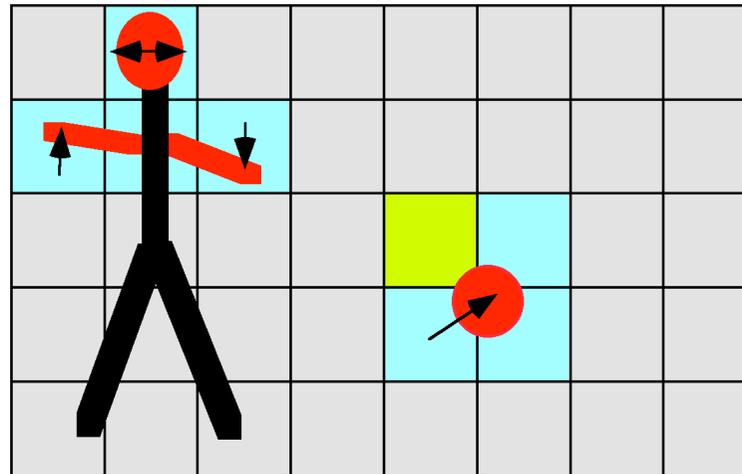
## 1.1.3 Videokompression

- Videostrom besteht aus Einzelbildern
  - Bilder einzeln komprimieren
  - z.B. JPEG/M, X-Movie, SMP
  - stärkere Kompression als bei Einzelbildern
  - Kompression 20:1 -> 120 MBit/s:6 MBit/s
- Ähnlichkeit der Bilder in zeitlicher Sequenz
  - hohe Ähnlichkeit in Szenen zwischen Schnitten
  - nur Änderungen übertragen
  - stationäre Objekte, Hintergrund
  - Blockbildung erforderlich
  - Bewegung im Raum = Transformationen
- Differenzerkennung
  - Differenzmaß, Blockgröße
  - exakt oder näherungsweise?
- Technische Probleme
  - Änderung verloren/verfälscht?
  - später Einstieg in den Strom



### 1.1.3.1 Conditional Replenishment [Frederick]

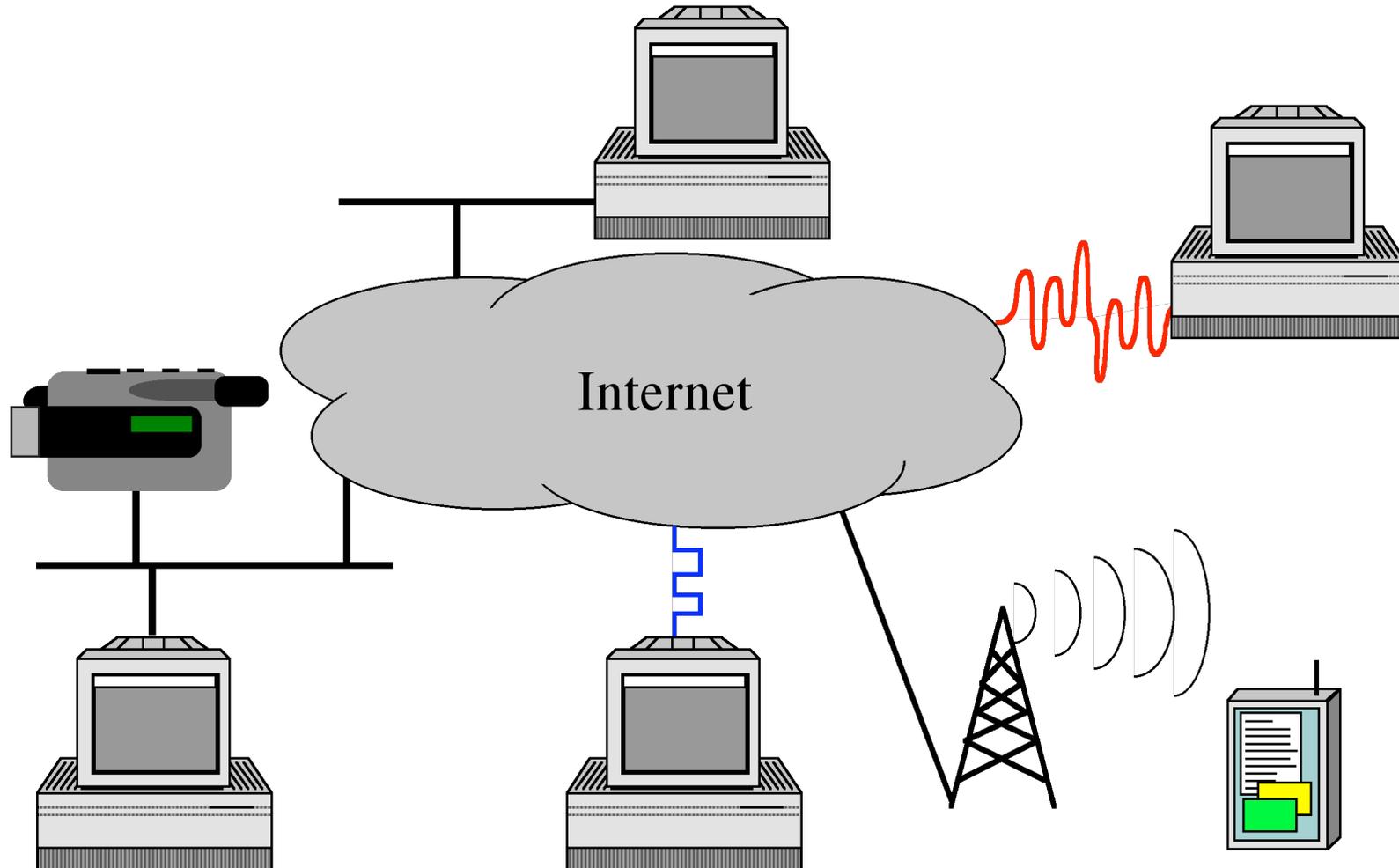
- Anfangsbild komprimiert vollständig übertragen
- Einteilung der Folge-Bilder in Blöcke
  - blockweise Differenz berechnen
  - Differenz > Schwellwert -> Block 'nachfüllen'
  - Blöcke vollständig oder als Differenz übertragen



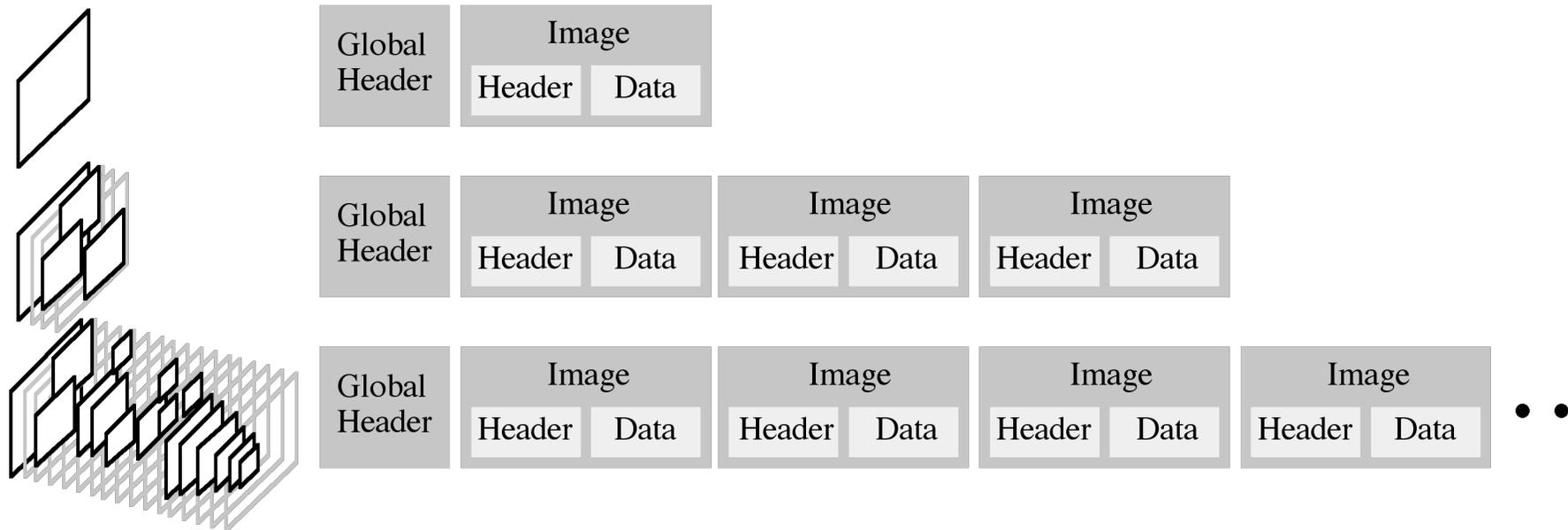
- Formate für Conditional Replenishment
  - selbstdefiniert: mbone-Tool nv
  - GIF, Delta-JPEG [Wolf, Froitzheim]
  - mbone-Tool vic: Intra-H.261 [McCanne, Jacobsen]

## 1.1.3.2 Streaming

- Unterschiedlicher Durchsatz in Mehrpunkttopologien
  - LAN, ISDN, Modem, GSM, ...

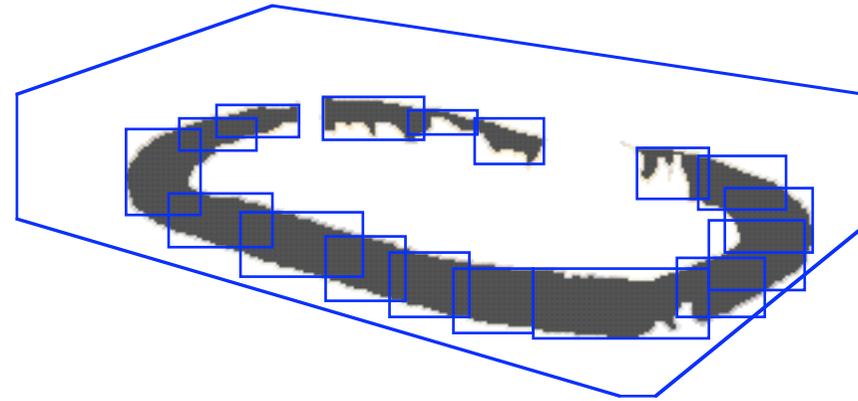
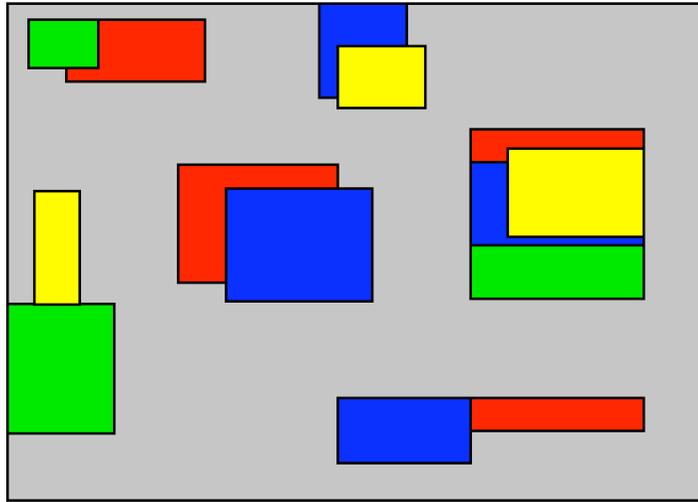


- GIF aus dem Bildschirmtext-Zeitalter
  - Header: globales Rechteck und Farbtabelle
  - Bilder als Teilrechtecke, Farbindices, evtl. mit eigener Farbtabelle

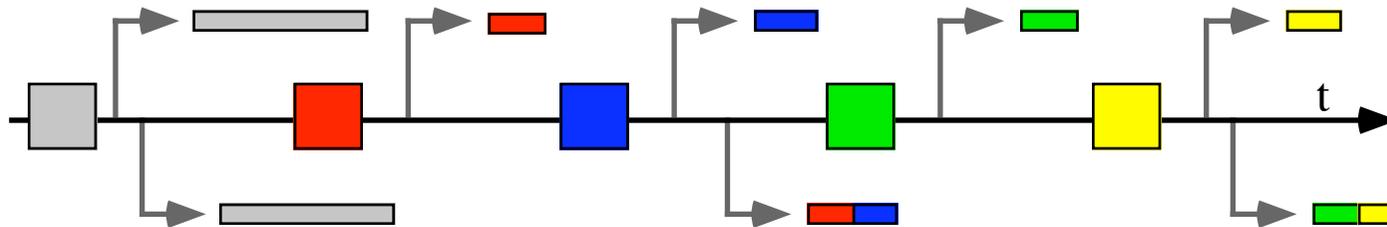


- LZW zur Indexkompression
- animated GIF im WWW
- conditional replenishment möglich
- Δ-JPEG ähnlich
  - JPEG-Vollbild
  - Änderungen als Sub-Images: pos, kleines JPEG-Bild
  - auf Raster oder beliebige Position

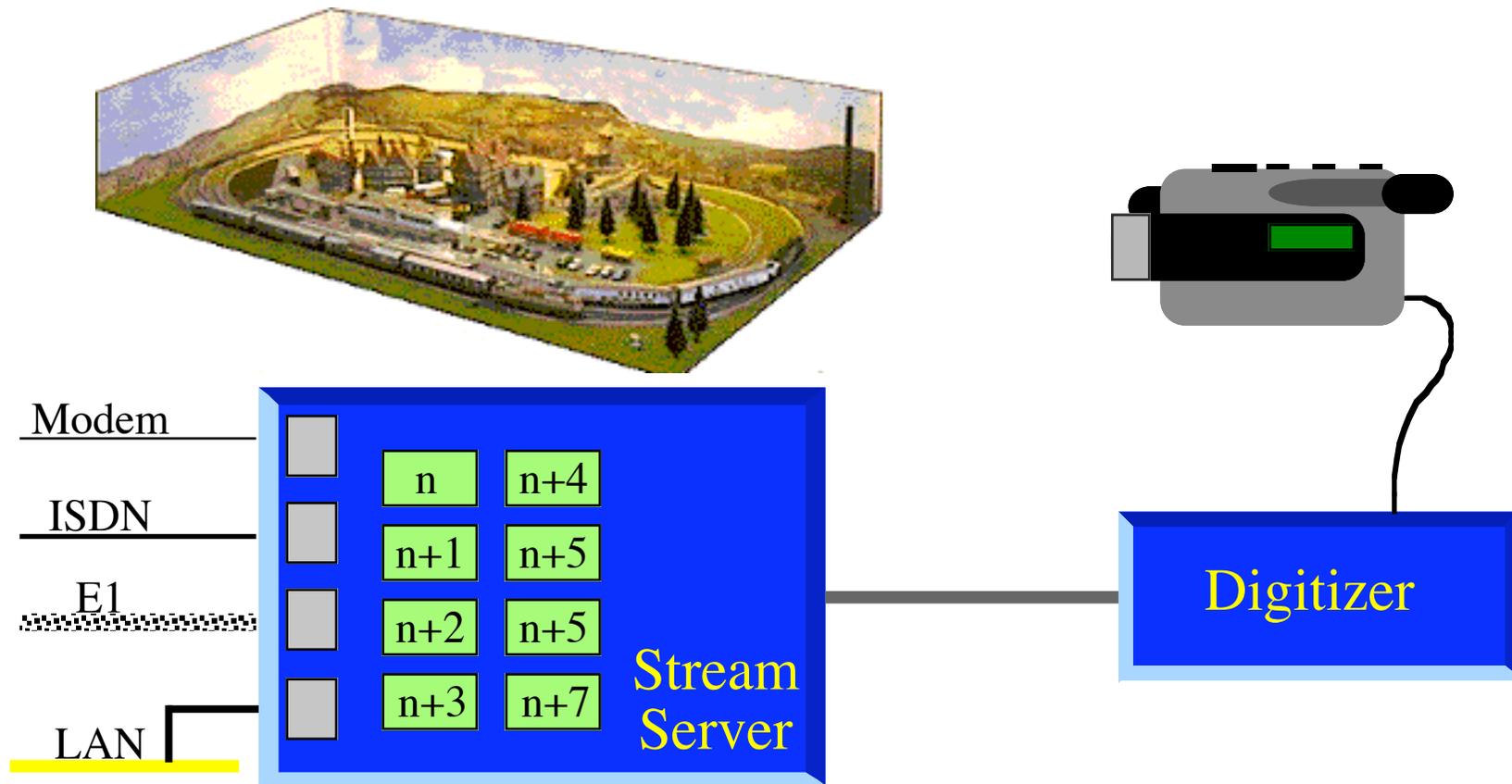
- Bestimmung der Differenz: Änderungsinformation (action-block)



- Updateinformation beim Bildabruf erzeugen



- Webvideo

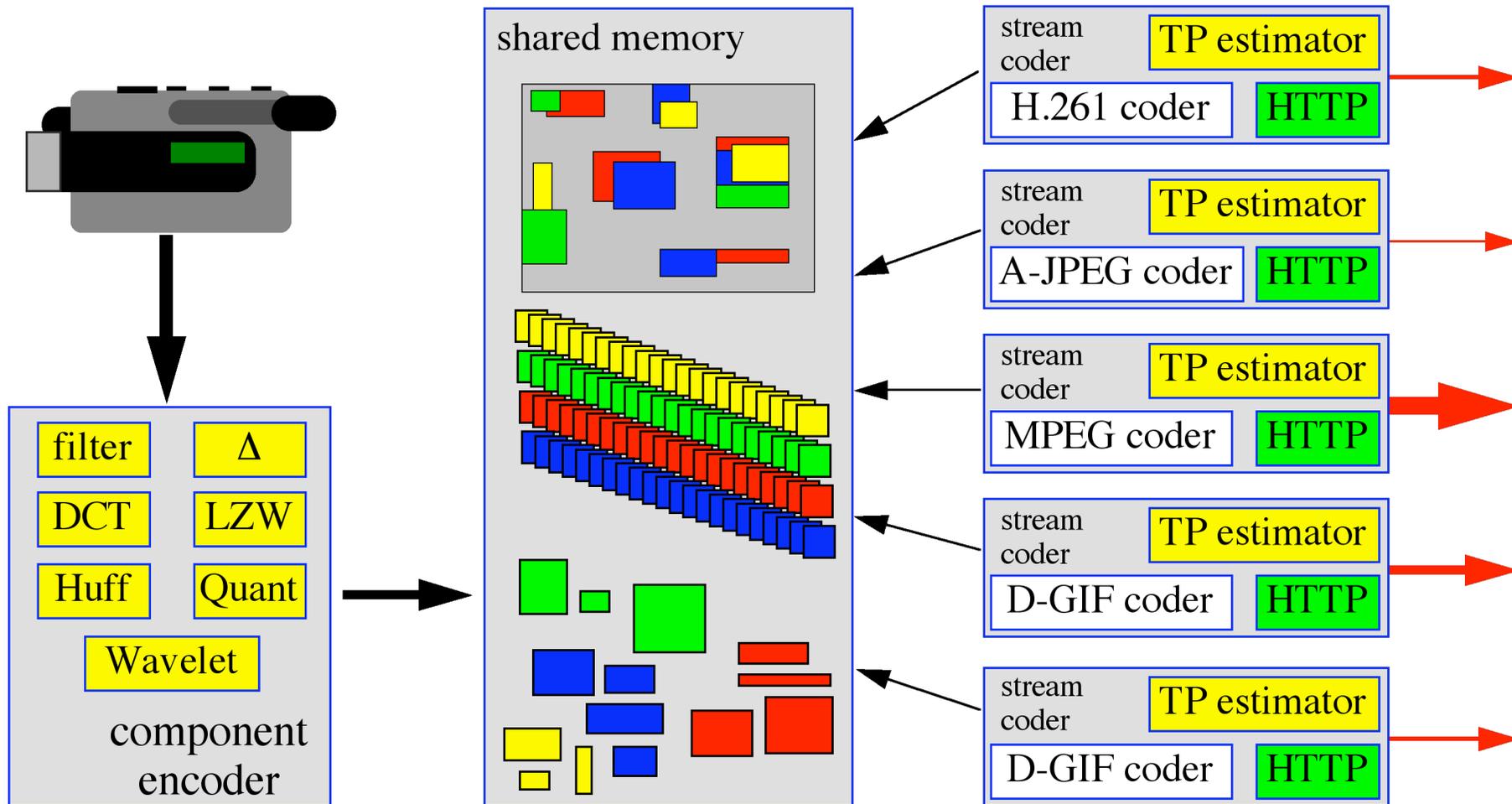


- Streamserver

- erzeugt Differenzinformation
- konstruiert Datenstrom beim Absenden eines Bildes

- GIF-Strom aus Anfangsbild und Teilbildern

- CESC - Component encoding, Stream Construction
  - Kanalmessung, just in time
  - Wiederverwendung von vorverarbeiteten Stücken



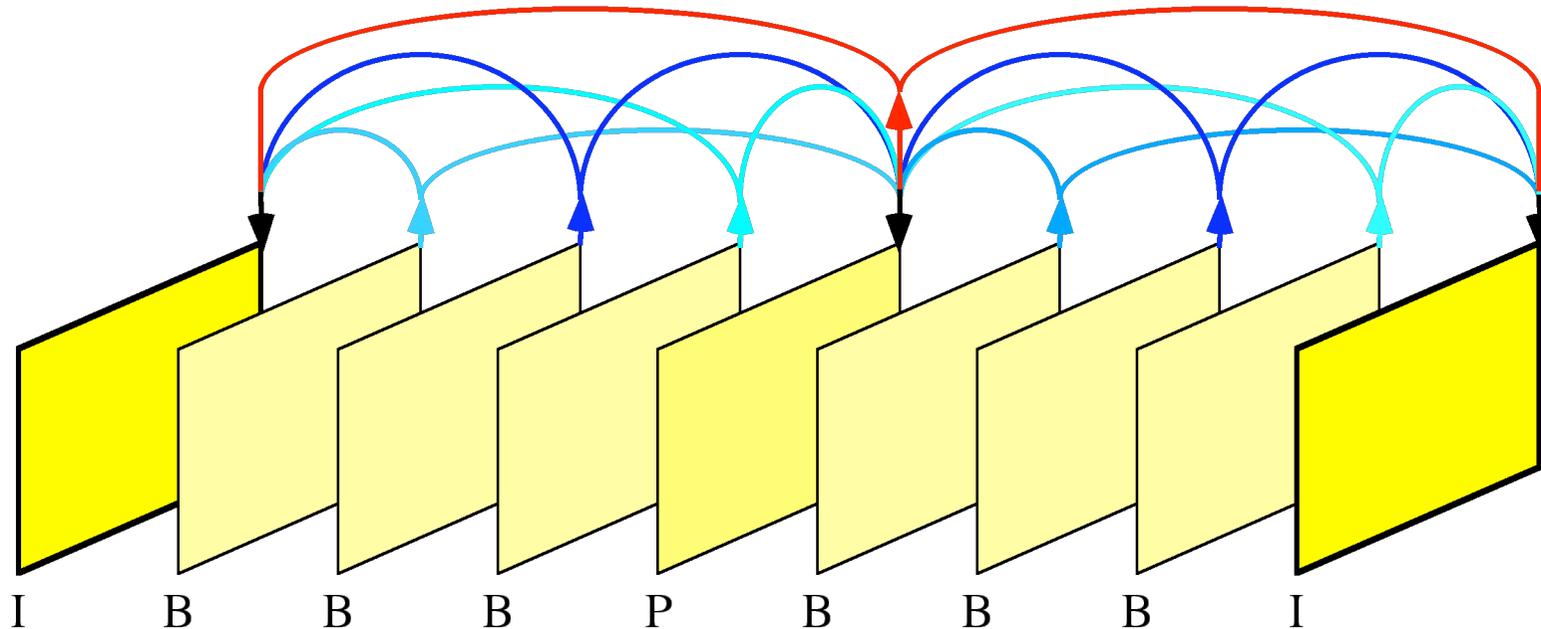
- Wiederverwendung
  - DCT+Huff: JPEG,  $\Delta$ -JPEG, H.261, MPEG
  - LZW:GIF
- Differenzerkennung
  - Grenzwerte von der Eingabe abhängig: RGB, JPEG, ...
  - Filter über mehrere Blöcke
  - für alle Stromformate
- Änderungsblock
  - evtl. variable Größe
  - Gitter oder beliebige Position
- Bandbreitenvorhersage
  - kleiner TCP-Puffer => buffer empty callback
  - Verlorene Zeit während 'back-trip delay' und Stromkonstruktion
  - nächsten callback vorhersagen und Kanal vorsorglich füllen
- Garbage collection

### 1.1.3.3 MPEG (Moving Pictures Expert Group)

- Kompression von Bewegtbildern und Ton
  - Auflösung SIF oder CIF, 30/25 Hz (MPEG I)
  - Audio in CD-Qualität
  - Datenrate 1,5 MBit/S
- Farbraumkonvertierung RGB -> YUV
  - U und V werden jeweils auf die Hälfte reduziert (4:2:2)
- JPEG zur Kompression von "Stützbildern"
  - Resynchronisation nach Datenverlust
  - Später Einstieg in den Datenstrom
  - Vermeidet 'wegdriften' vom Original
- Ausnutzen von Beziehungen zwischen Frames: Bewegungskompensation
  - mehrere, unterschiedliche Bewegungen im Bild
  - Bewegungsinterpolation, Bewegungsvorhersage
- Synthese von Zwischenbildern aus Stützbild-Elementen

### 1.1.3.3.1 MPEG-Frametypen

- verschiedene Frametypen
  - Intrapictures (JPEG-kodiert, I)
  - Predicted Pictures (Vorwärts-Prädiktion, P)
  - Interpolated Pictures (bidirektionale Prädiktion, B)



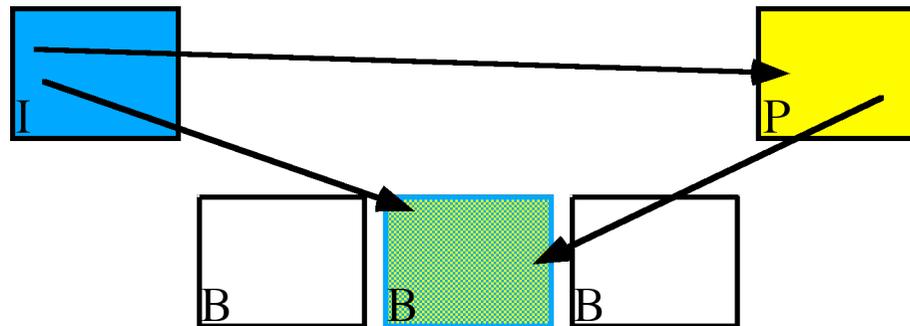
- Macroblöcke 16\*16 im Y-Bild

- Intrapictures (I-Frame)
  - JPEG-kodiert
- Vorwärts-Prädiktion (P-Frame)
  - Suchen wohin sich Macroblöcke bewegt haben

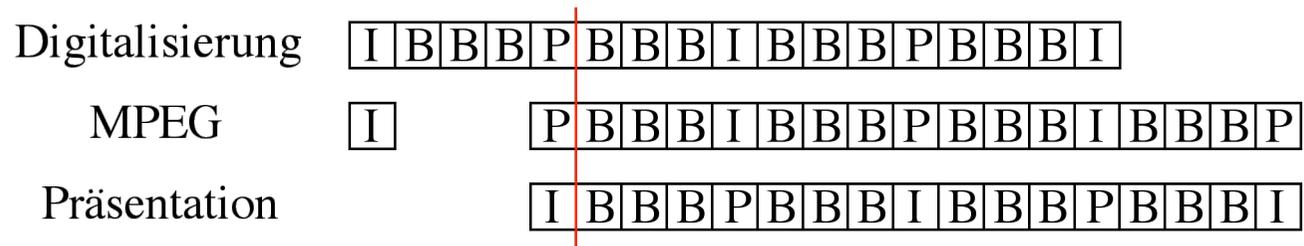
$$\min(x,y) \sum_{i=0}^{15} \sum_{j=0}^{15} |I_1(i,j) - I_2(i+x,j+y)|$$

- Zwei Möglichkeiten zum Übertragen der Macroblöcke
  - Bewegungsvektor + Fehlerinformation (JPEG-codiert)
  - voller JPEG-Block
- leichte Unterschiede zu JPEG
- Bewegungsvektoren differenzkodiert

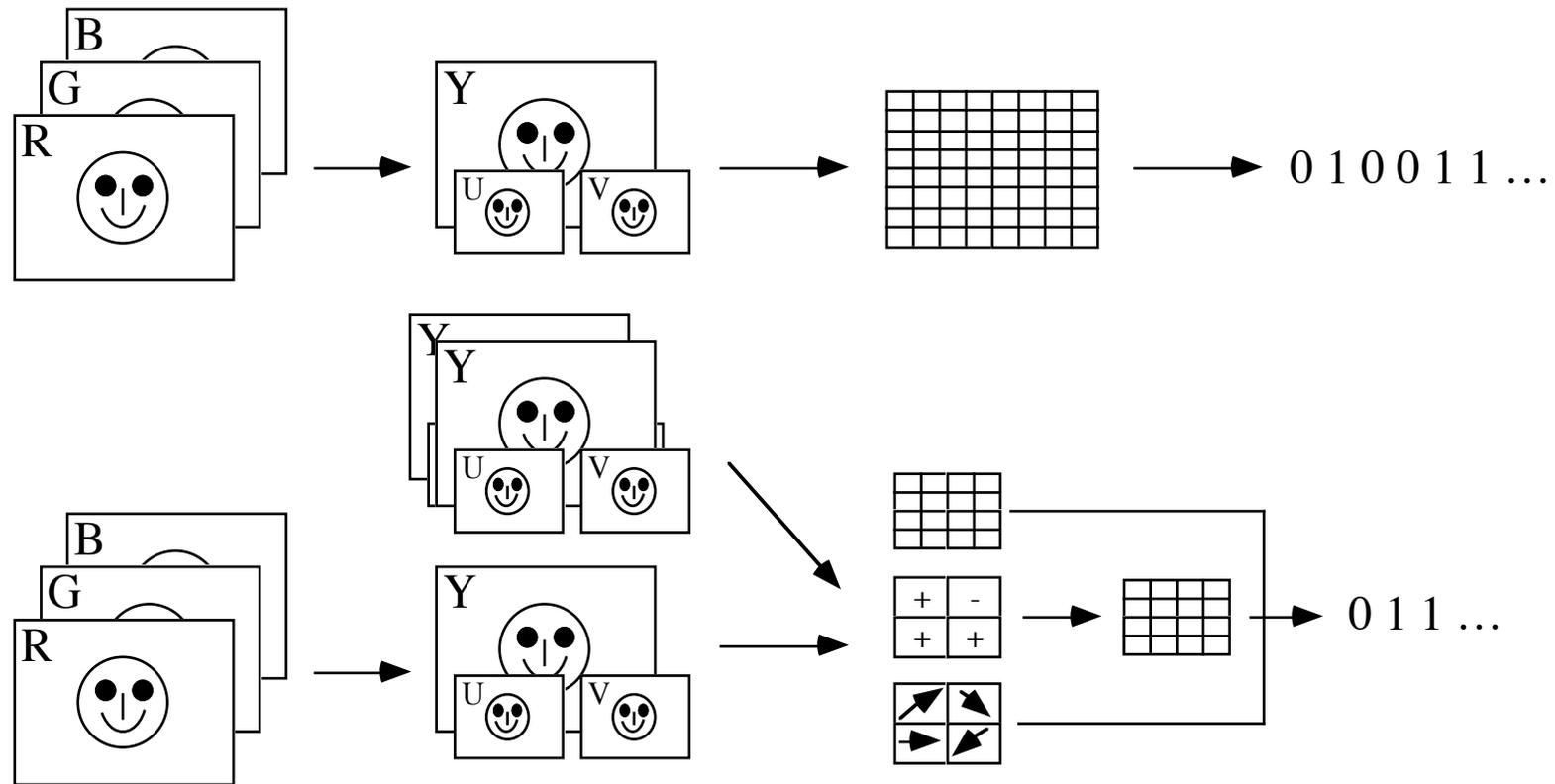
- Bidirektionale Prädiktion (B-Frame)
  - zwei Typen Pixelinformation: Objekt + auftauchender Hintergrund
- bei der Codierung
  - Zwischenrahmen blockweise mit Interpolation ermitteln
  - Unterschied zum tatsächlichen Block ermitteln (error block)
  - Korrekturterm mit JPEG behandeln
  - Bidirektional für Behandlung von auftauchendem Hintergrund
- Drei Optionen pro Macroblock
  - Fehlerblock zur Korrektur der Interpolation
  - Bewegungsvektor (vorwärts, rückwärts, beide)
  - Codiertes Original



- Bei der Decodierung
  - Zwischenrahmen interpolieren
  - Korrekturterme anwenden
  - Blöcke bewegen
  - Volle Blöcke einsetzen
- Probleme der B-Frame Kodierung
  - Erhöhte Speicheranforderungen (3 Bilder)
  - Verzögerungen (Pipeline) von 1/6 Sekunde
  - Übertragungsreihenfolge: IPBBIBBB PBBIBBB ...



- Modeldecoder-Prinzip
  - Sicht vom aktuellen Bild zu Bezugsbildern
  - Coder sorgt für vollständige Überdeckung:
  - Block (x,y) kommt von Position (v,w). Nicht Block (a,b) nach (c,d)!



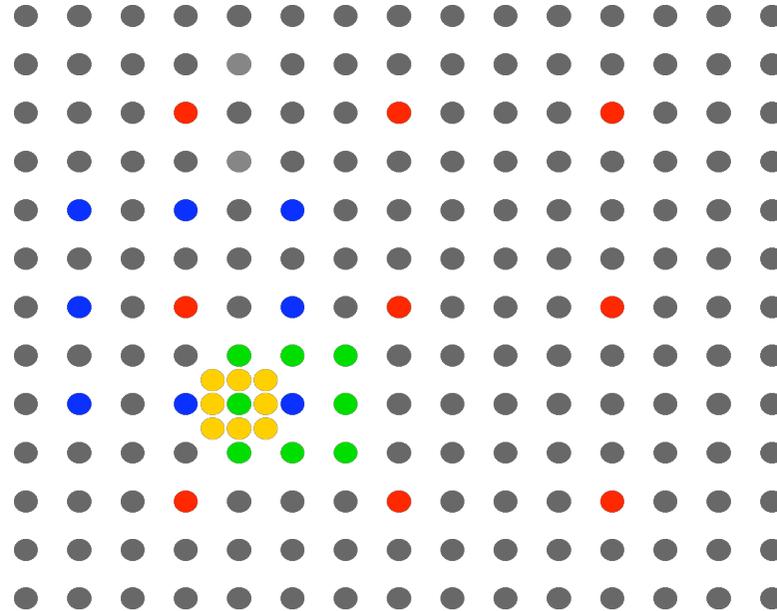
- Eigenschaften

- Bewegungsvektoren suchen ist sehr rechenintensiv
- Je nach Szenario in Bewegungsvorhersage investieren
- Decoder wesentlich einfacher als Coder
- Einstieg alle 0,4 Sekunden möglich
- Variable Bitrate

### 1.1.3.3.2 Bewegungsvektorsuche

- Nicht standardisiert, Ideen im Standard
- Makroblock 16\*16
- Im Y-Bild
- Reichweite
  - 7 Bereiche: -8 .. +7.5 bis -1024 .. +1023
  - pro Bild einheitlich
  - bestimmt Codierung des Vektors
- Differenzmaß (Kostenfunktion)
  - Summe der Pixeldifferenzen  $\sum |a_{ij} - b_{ij}|$  (mindestens 511 Operationen)
  - Fehlerquadrate
  - Halbpunktgenauigkeit => Interpolation
- Suchstrategie
  - Vollsuche mit Reichweite n:  $511 * \sum_{i=0}^n (2i+1)^2 + (2i)^2$
  - Vollsuche mit ganzen Vektoren, Minimum als Zentrum der Suche mit halben Vektoren

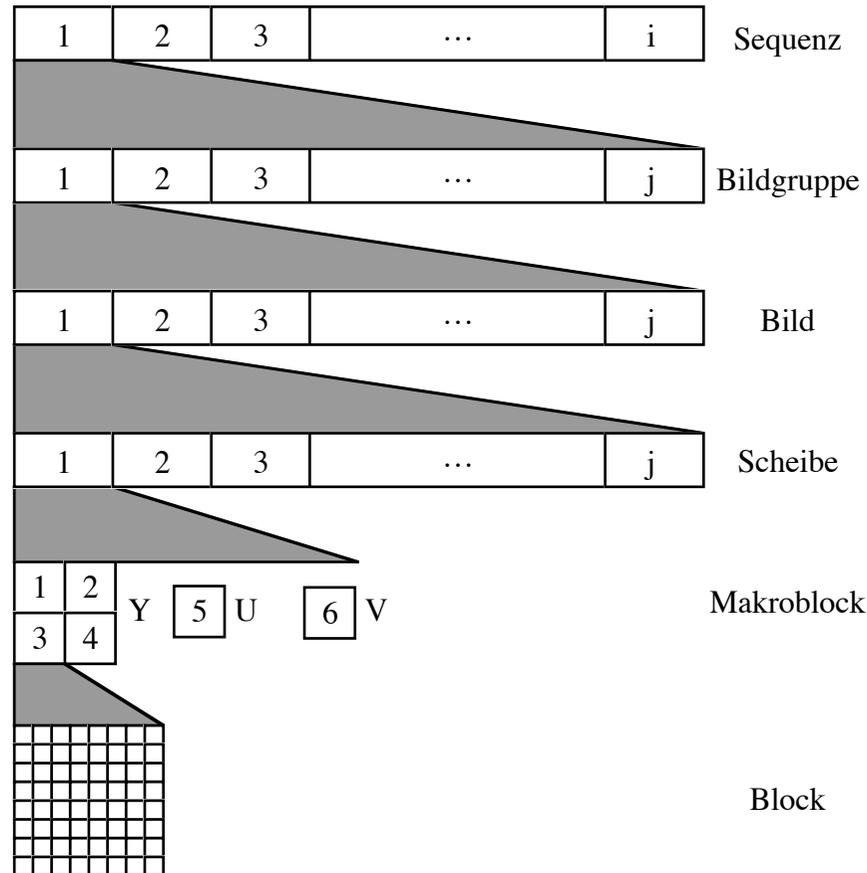
## - Logarithmisch



- Ausnutzen bekannter Vektoren aus den vorangegangenen Bildern
- Suche im Originalbild gibt bessere Vektoren
- Suche im decodierten Bild gibt kleinere Fehlerterme
- Bewegungsvektoren:
  - gut für Kammerschwenk und linear bewegte Objekte
  - schlecht für Zoom und rotierende Objekte

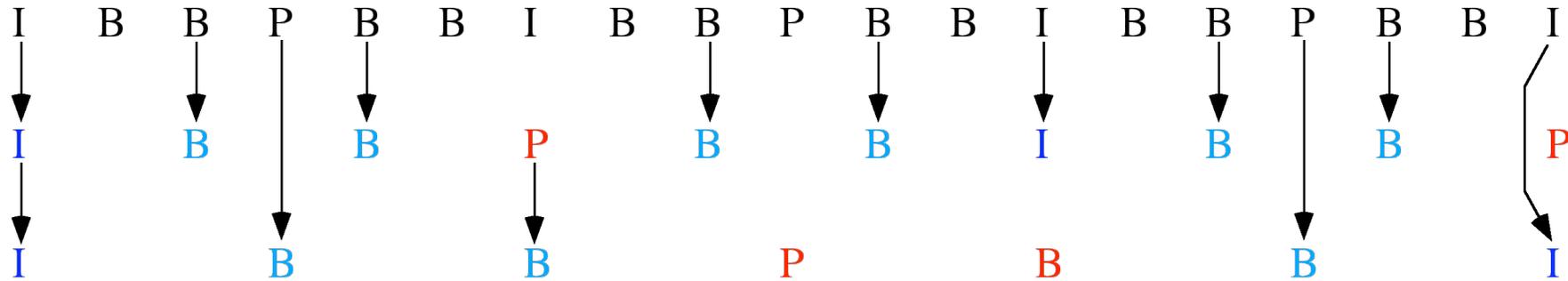
### 1.1.3.3.3 MPEG-Datenstrom

- Synchronisationspunkte auf Teilbildebene
  - Aufsetzen der Dekompression nach einem Übertragungsfehler
- Datenstrom



- MPEG-Bitstrom hat sechs Ebenen (Layer)
  - Sequence: ein Videostrom
  - Bildgruppe (GOP, group of pictures): Teil eines Videos  
 IBBBPBBB(I), IBBPBBPBB(I), IPPPPP(I), IIIIIII, ...
  - Bild (I, P, B)
  - Scheibe (slice): Bildausschnitt mit wählbarer Größe zur Resynchronisation nach einem Fehler
  - Makroblock (macroblock): 16\*16 Punkte, auf der die Bewegungskompensation aufbaut
  - Blöcke: 8\*8 Punkte für DCT, Quantisierung und Linearisierung
- Variable Bitrate
  - Szenenwechsel, freiwerdender Hintergrund -> Anstieg
  - Datenrate begrenzt durch CD-ROM bzw. Kanal
- Video Buffering Verifier (VBV)
  - simuliert Puffer im Empfänger
  - Buffer-Underflow und -Overflow
  - Steuerung der Quantisierung/Vektorsuche
  - glättet Datenstrom
  - Datenrate aber nicht konstant

- MPEG-Datenstrom schwer skalierbar
  - Wiederholrate 7 statt 25 reduziert Datenrate um 20%
- Hierarchische Kompression?



- Wiederverwendung von Bewegungsvektoren
  - filmabhängiger Mehraufwand
  - höhere Wiederverwendbarkeit → schlechtere Kompression
- Kompression → Bereitstellung der Kodierungsbasis
    - jedes Bild als komprimiertes Vollbild
    - Differenzmarkierung
    - Bewegungsvektoren

- MPEG-2 (ISO 13818) mit CCIR-601 Bildern
  - 4 MBit/s, random access, VCR-Funktion
  - program stream (ähnlich MPEG-1) + Verschlüsselung, Prioritäten, ...
  - transport stream: 188-byte Pakete
  - besondere Vorkehrungen für Interlaced Betrieb
  - Skalierbarkeit
  - Layer (base, middle, enhancement)

Levels	Profile	Simple	Main	SNR Scale	Spatial Scale	High 4:2:0, 4:2:2
High	1920*1152*30		80			25,80,100
High 1440	1440*1152*30		60		15,40,60	20,60,80
Main	720*576*30	15	15	10,15		20
Low	352*288*30		4	3,4		

- Levels: Bildauflösung
- Profiles: Techniken und Komplexität der Kodierung
  - SNR-scalability: zusätzliche DCT-Koeffizienten im enhancement-layer
  - spatial scale mit zwei räumlichen Auflösungen
 => "MP@ML"

### 1.1.3.4 ITU-Videokompression H.261

- Verteilung auf mehrere ISDN-Kanäle ( $p \times 64$ )
  - $p < 6$ : QCIF (Kompression 50:1 bei 10 fps für  $p=1$ )
  - $p \geq 6$ : CIF (optional)
- Bildwiederholrate wählbar
- JPEG für Stützbilder (intraframe)
- Vorwärts-Prädiktion für Zwischenbilder
  - Macroblockvergleich im dekodierten Stützbild
  - Bewegungsvektor
  - Differenz-DCT mit linearer Quantisierung
- bitratenkonstant
  - Quantisierungsmatrix für DCT wird zur Laufzeit verändert
  - Bildqualität schwankt während Verbindung
  - 'Quantisierschleife' für Übertragungspuffer
- Datenstrom mit vier Ebenen
  - Picture, Group of Blocks
  - Macro Blocks, Block

## 1.1.4 MPEG-4

- MPEG

- JPEG + Differenzkodierung + mpn-Audio + Multiplexer + ...

- MPEG-1

- Kompressionstechniken für Video und Audio
- Stromformat
- 'implizite' Synchronisation

- MPEG-2

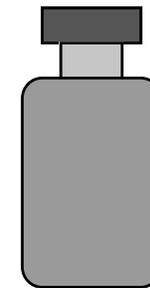
- flexibleres Stromformat und Profile
- Synchronisation durch einfache Zeitachsen

- Wie gut ist die 'Bewegungskompensation'?

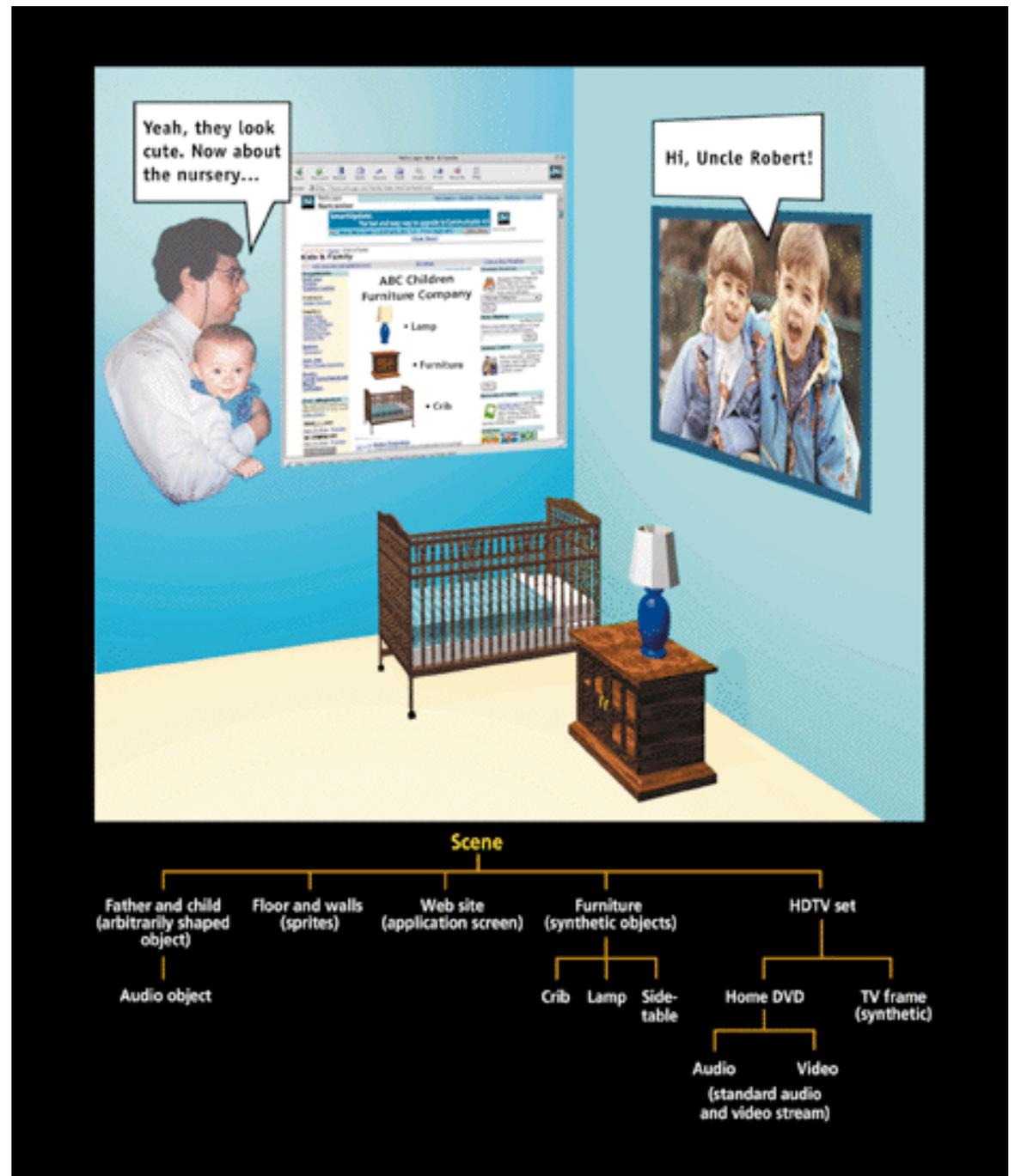
- Objekte bewegen sich Ansichtswinkel-konstant zur Kamera
- akzeptabel für Kameraschwenk+Landschaft
- gleiche Behandlung für langsame und schnelle Bildteile

- Kompressionsleistung

- Objektzerlegung und Semantikwissen
- verbesserte Kodierung (Prädiktion ...)

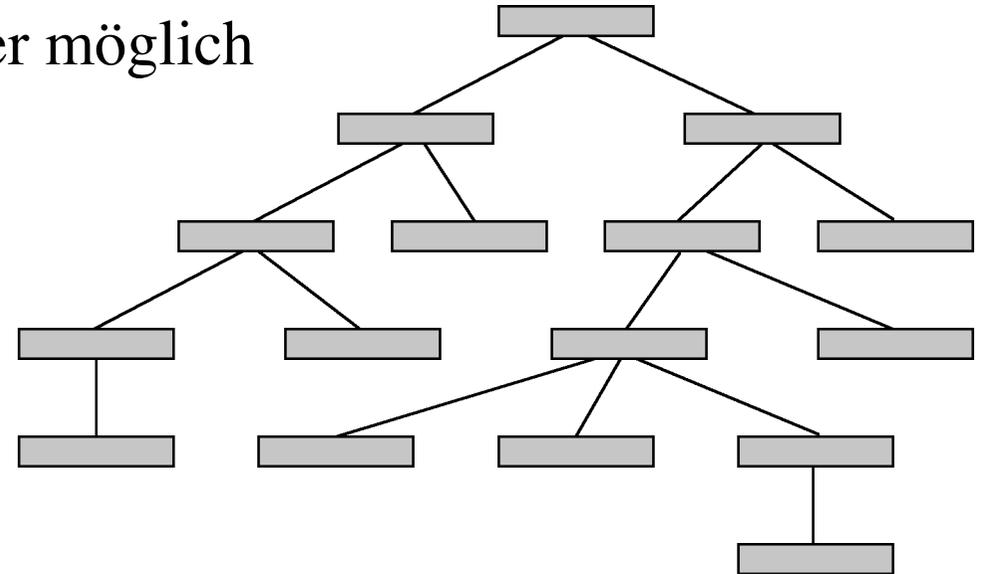


- Szenen im *Dekoder*
  - Menge von Objekten
  - künstliche Objekte
  - Video-Objekte
  - Bilder
  - Grafiken
  - Audioströme
  - Interaktionselemente
- Komposition
  - 4D-Position
  - Transformation
  - Programm
- Kodierung
  - fehlertolerant
  - skalierbar



- Szene
  - Objekthierarchie
  - Hintergrund, mehrere Vordergrund-Objekte
  - 'composition information'
  - Objektanordnung sogar durch Benutzer möglich

- Media objects (audio-visual objects)
  - zeitliche und räumliche Ausdehnung
  - hörbar und sichtbar
  - Text und Graphik
  - synthetisches Audio
  - zusammengesetzte Objekte



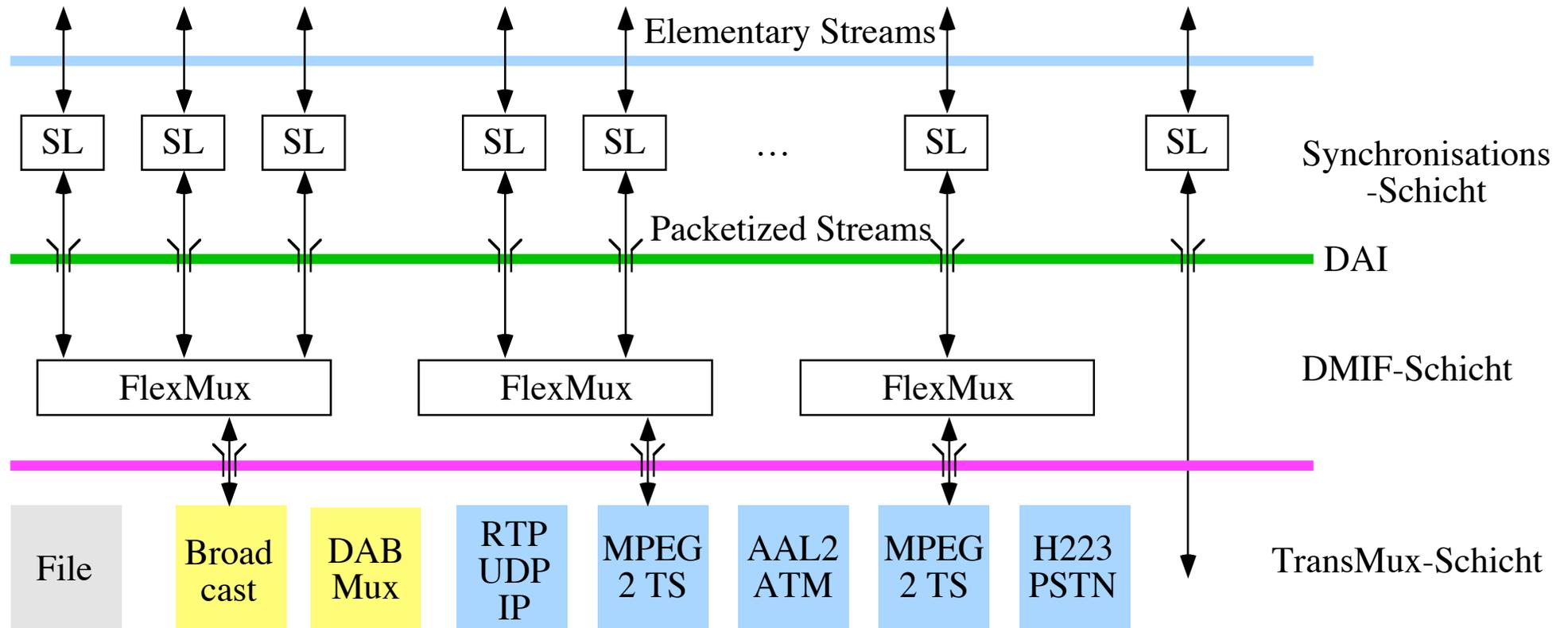
- Object descriptor
  - Objectbeschreibung: ES, Semantik, Zugang (URL ...)
  - Strombeschreibung: Decoder, QoS, ...
  - SDL: Parserbeschreibungs-Sprache  
`align bit(32) picture_start_code=0x00000100`

- Elementary Streams (ES)
  - Datenströme für Objekte (Samples, Animation, Text, Szenen, ...)
  - Zeitstempel

## 1.1.4.1 Transportmodell

- Synchronisation Layer

- Zeitstempel und Individual Access Units



- Elementary Streams

- Medien-Daten

- Kommando-Transport (Play, Pause, ...)

- Delivery Multimedia Integration Framework (DMIF)
  - **Abstraktion** für Netzwerke, Disks, Broadcast
  - DAI: DMIF (Delivery) Application Interface
  - Session-Protokoll mit Local-DMIF, Remote-DMIF, Remote-App
  - Remote-Komponenten evtl. local simuliert (Disk, Broadcast)
  - FlexMux-tool, andere Tools möglich
  - interleaving von ES's zu sinnvollen Gruppen
  - z.B. QoS-gesteuert
  - wenige Netzwerkverbindungen
- TransMux-Schicht
  - erbringt **Transportleistung**
  - nur das Interface ist in MPEG-4 spezifiziert
  - leer, leicht- oder schwergewichtig
- Sync Layer
  - liefert **Datenelemente (Elementary Streams)** zur richtigen Zeit
  - Zeitstempleauswertung
  - Füllen der Decoder-Buffer und Buffer Management

- Service

- `DA_ServiceAttach(ParentSessionId, URL, DataIn, SessionId, ...)`
- `DA_ServiceAttachCallback()`
- `DA_ServiceDetach*()`

- Channel

- `DA_ChannelAdd(SessionId, ChannelDesc, channelHandle, ...)`
- `DA_ChannelAddCallback()`
- `DA_ChannelDelete*()`

- QoS Monitoring

- `DA_ChannelMonitor(channelHandle, ...)`
- `DA_ChannelEvent(qosReport)`

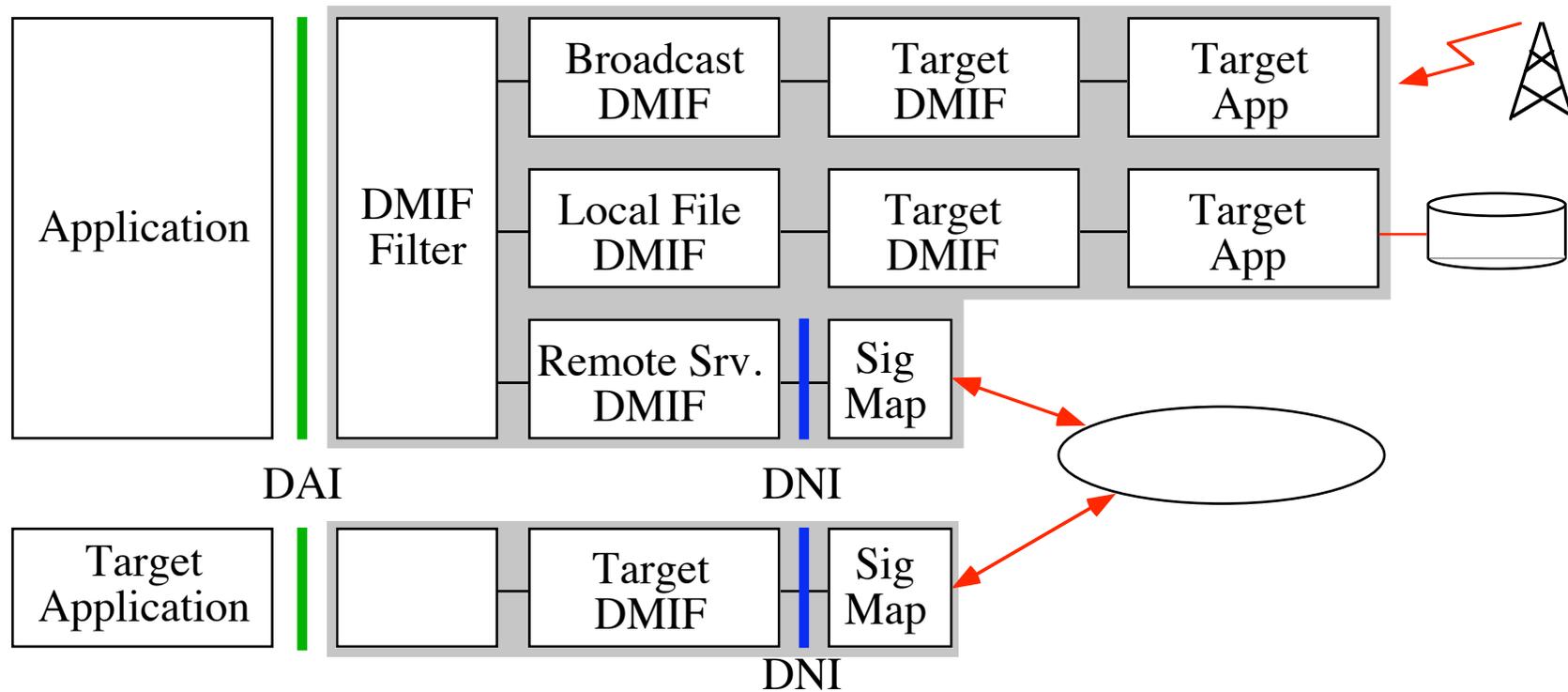
- User Command

- `DA_UserCommand(channelHandle, ...)`
- `DA_UserCommandCallback()`
- `DA_UserCommandAck()`
- `DA_UserCommandAckCallback()`

- Data

- `DA_Data(channelHandle, streamDataBuffer, streamDataLen)`
- `DA_DataCallback(channelHandle, streamDataBuffer, streamDataLen)`
- `DA_Data(channelHandle, ..., appDataBuffer, ...)`
- `DA_DataCallback(channelHandle, ..., appDataBuffer, ...)`

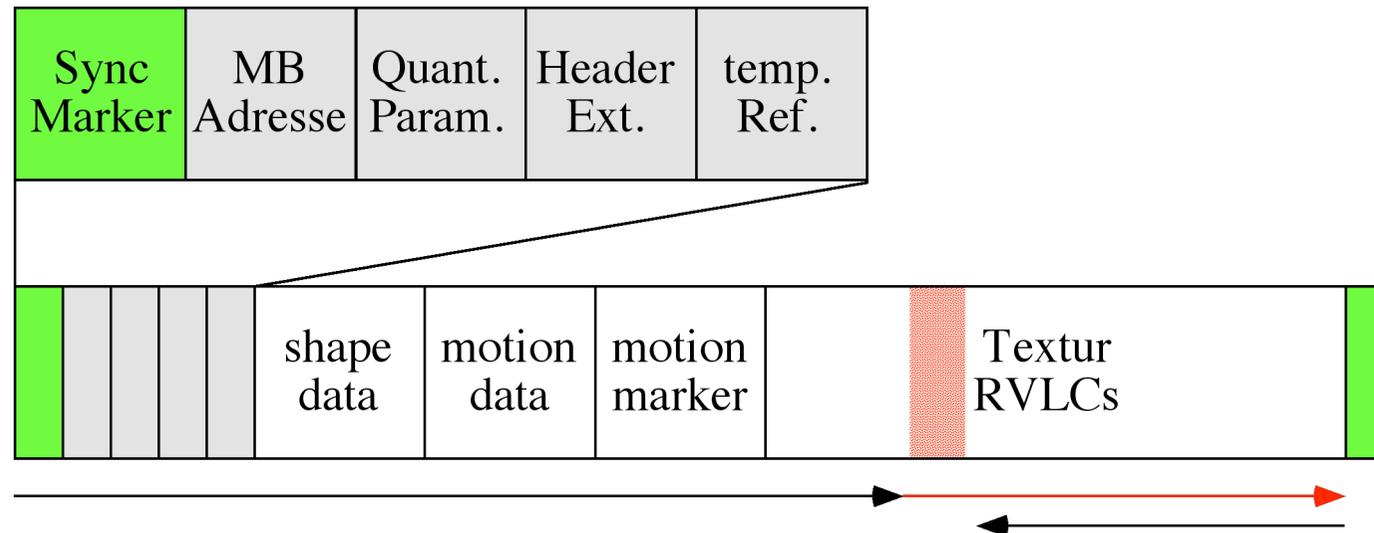
## • DMIF Network Interface



## • Dienstelemente (primitives)

- Session (setup, release)
- Service (attach, detach)
- TransMux (setup, release, config)
- Channel (add, added, delete)
- User command (command, ack)

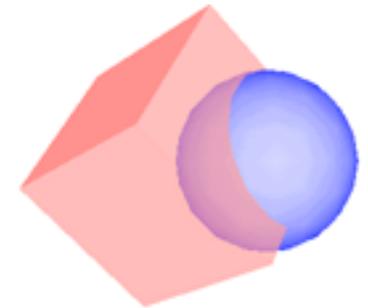
- Methoden zur Fehlererholung
  - Marker zur Resynchronisation
  - Reversible VLCs
  - Interleaving
  - Kodierung von Schichten (layered coding)



- Richtungsunabhängig dekodierbare Symbole (RVLC)
  - VLC: variable length codes
  - VLCodes mit fester Anzahl '1' (Hamming-Gewicht)
  - RVLC = Präfix+VLC+Suffix
  - Vorwärts dekodieren bis Bitfehler
  - zum Marker scannen
  - Rückwärts dekodieren bis Bitfehler

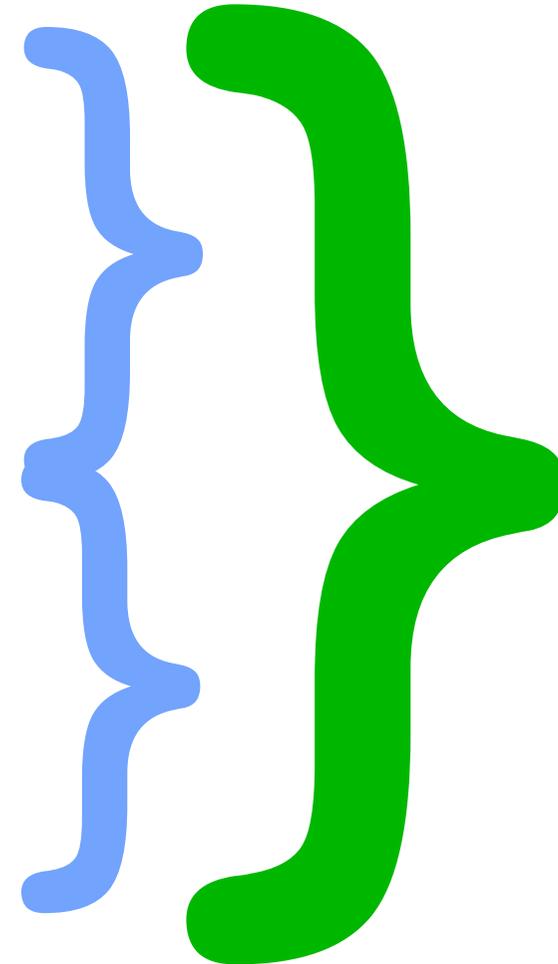
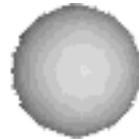
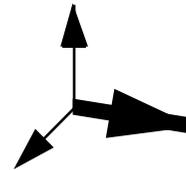
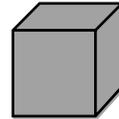
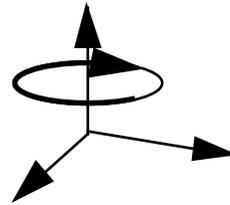
## 1.1.4.2 Szenenbeschreibung

- Szenenbeschreibung
  - "Programmiersprache"
  - Objekte definieren
  - Eigenschaften ändern (Farbe, Leuchtkraft, ...)
  - Objektgruppen mit relativer Position zueinander
  - Kamera, Beleuchtung
- VRML bzw. Web3D bzw. X3D
- Virtual Reality Markup Language
- Einfache ASCII-basierte Syntax
- VRML 97 (2.0)
  - dynamische Welten, Interaktive Kontrolle
- External Authoring Interface (EAI)
  - 3D-Welt aus externem Programm fernsteuern
  - Sensoren zur Kommunikation mit Programm



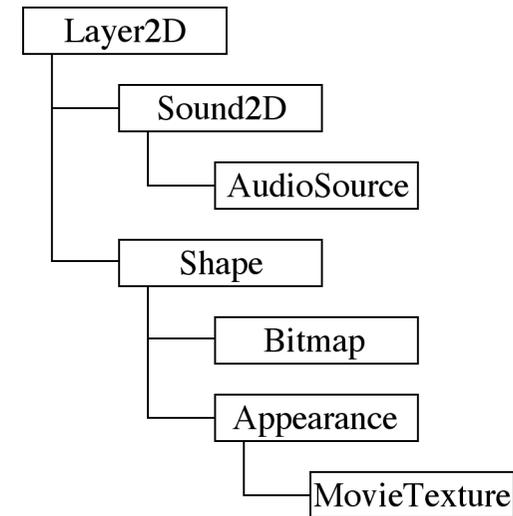
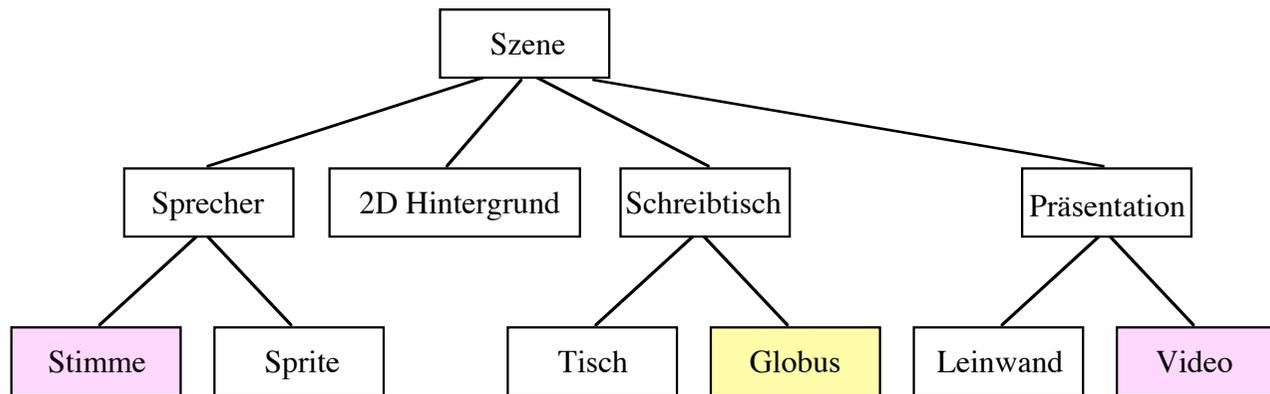
- Eine einfache VRML-Welt

```
#VRML V1.0 ascii
Separator {
  Separator {
    Transform {
      rotation 0 1 1 1.2
    }
    Material {
      emissiveColor 1 0 0
      transparency 0.3
    }
    Cube {}
  }
  Separator {
    Transform {
      translation 1 0 0
    }
    Material {
      emissiveColor 0 0 1
    }
    Sphere {}
  }
}
```



- BIFS: Binary Format for Scenes

- 3D-Szenenbeschreibung
- binär kodiert statt VRML-ASCII-Text
- dynamische Szenen: Objekte hinzufügen und entfernen
- Positionsveränderung  $\neq$  motion vector



- Objektposition

- globales und lokale Koordinatensysteme
- Ausdehnung in Zeit und Raum

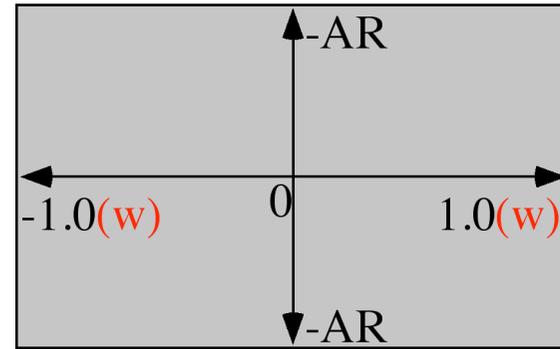
- Veränderungen der Objektattribute

- MPEG-J und Information im Elementary Stream

- XMT: eXtensible MPEG-4 Textual format: VRML/X3D bzw SMIL

- Rendering-Kontext

- Layer2D: root-Objekt für 2D-Szenen
- Layer3D: root-Objekt für 3D-Szenen
- CompositeTexture2D (...3D) off-screen
- Transform2D, Transform
- Koordinaten metrisch oder pixel



- Sound, Sound2D Knoten

- Shape-Knoten

- geometry-Feld: Rectangle, Circle, Box, Sphere, Bitmap
- appearance-Feld: Material, ImageTexture, MovieTexture

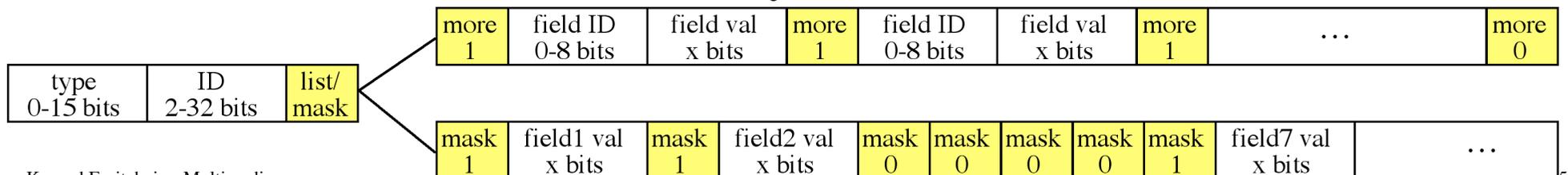
- Sensoren, TimeSensor, Interpolatoren

- BIFS-Kommandos

- Insert/Delete Knoten oder Felder in Knoten
- Replace Szenen, Knoten, Felder, Routen

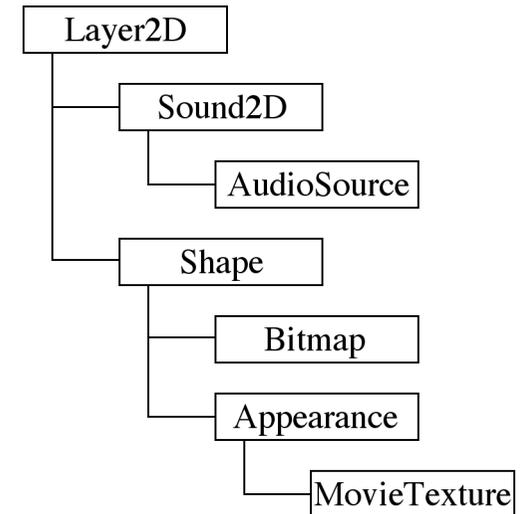
- Binärkodierung

- Command Frames in Elementary Stream Access Units



## • Einfacher Szenengraph in XMT-Form

1	<code>&lt;Replace&gt; &lt;Scene&gt;</code>	10
2	<code>&lt;Layer2D&gt; &lt;children&gt;</code>	12
3	<code>&lt;Sound2D&gt; &lt;source&gt;</code>	11
4	<code>&lt;AudioSource url='od:3"</code>	
5	<code>startTime="0.0" stopTime="-1.0"/&gt;</code>	94
6	<code>&lt;/source&gt; &lt;/Sound2D&gt;</code>	1
7	<code>&lt;Shape&gt;</code>	8
9	<code>&lt;geometry&gt; &lt;Bitmap/&gt; &lt;/geometry&gt;</code>	7
10	<code>&lt;appearance&gt;</code>	1
11	<code>&lt;Appearance&gt; &lt;texture&gt;</code>	6
12	<code>&lt;MovieTexture url="od:4" loop="false"</code>	
13	<code>&lt;startTime="0.0" stopTime="-1.0"/&gt;</code>	93
14	<code>&lt;/texture&gt; &lt;/Appearance&gt;</code>	2
15	<code>&lt;/appearance&gt;</code>	1
16	<code>&lt;/Shape&gt;</code>	1
17	<code>&lt;/children&gt; &lt;/Layer2D&gt;</code>	1
18	<code>&lt;/Scene&gt; &lt;/Replace&gt;</code>	2

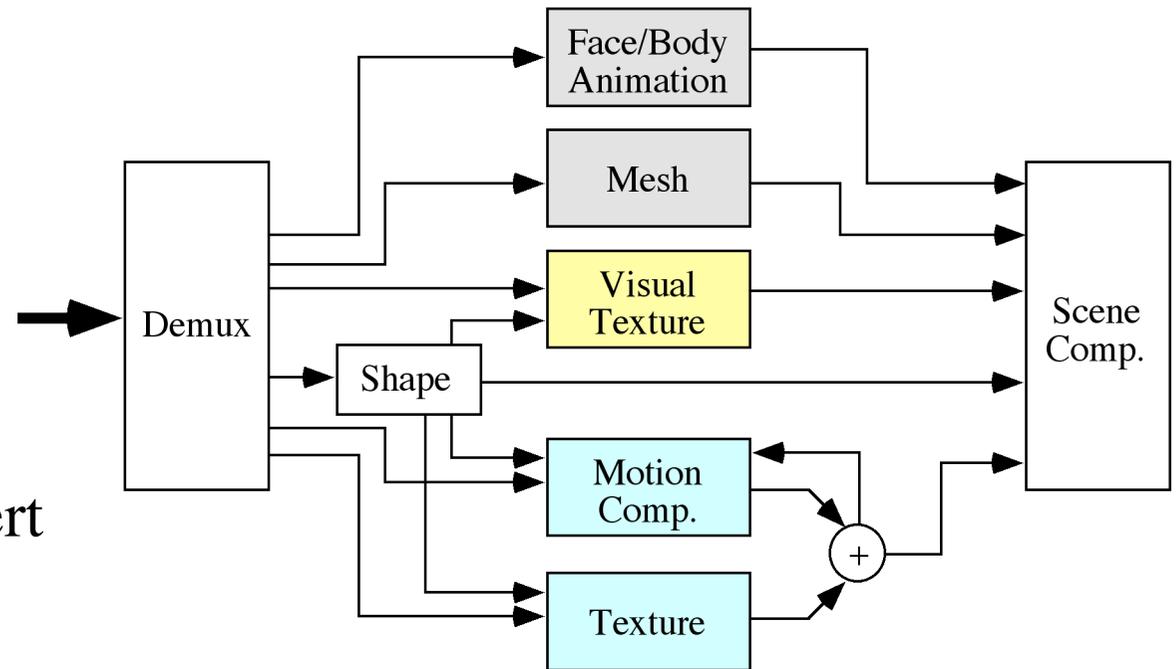


- BIFS-Grösse 250 Bit - 32 Byte

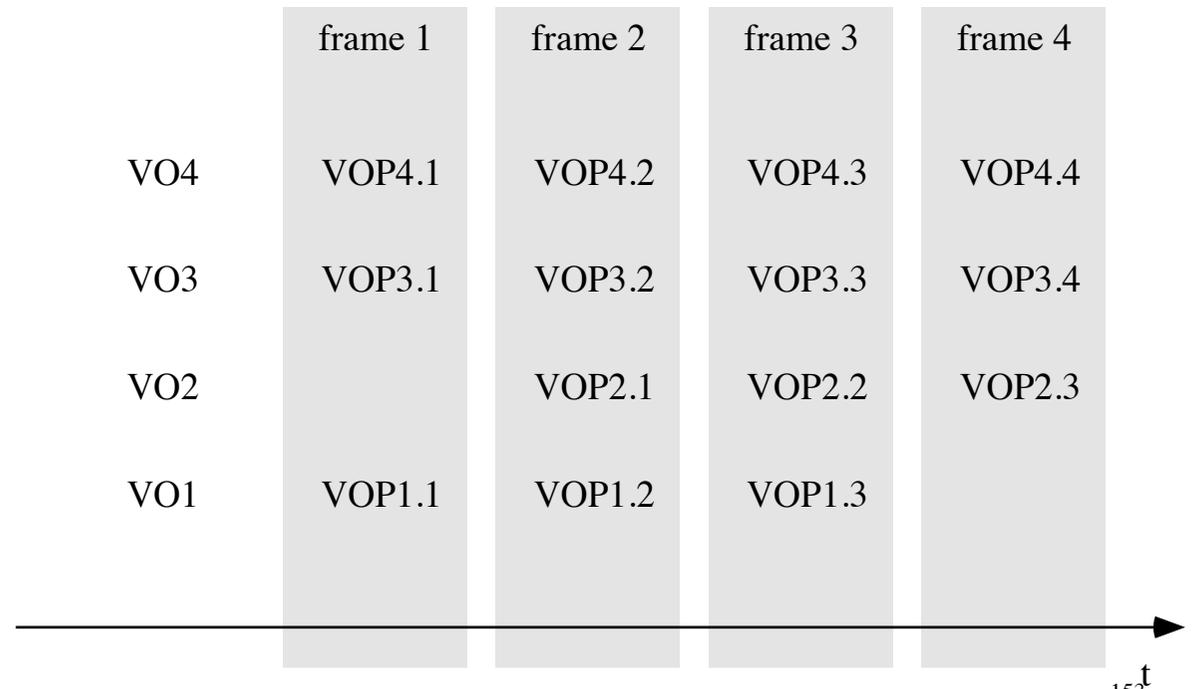
- Objektsdeskriptoren 3 und 4 enthalten Medienströme

## 1.1.4.3 Visual Objects

- Szenen-Komponente
  - individueller Zugang
  - einzeln manipulierbar
- Texture = Pixelinformation
- Video object
  - Texture wie MPEG1/2 kodiert
  - Bewegung (motion)
  - Form (shape)
  - triviale und komplexe Form



- Framekonstruktion
  - aus Video-Object-Planes
  - 'Prioritäten'
  - Hintergrund

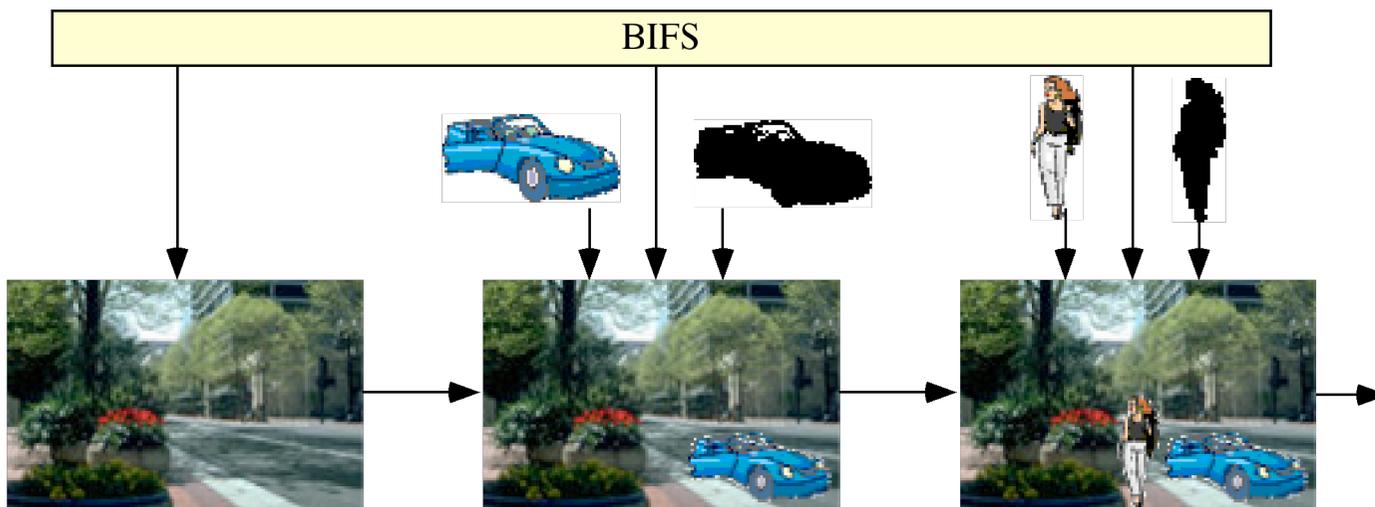
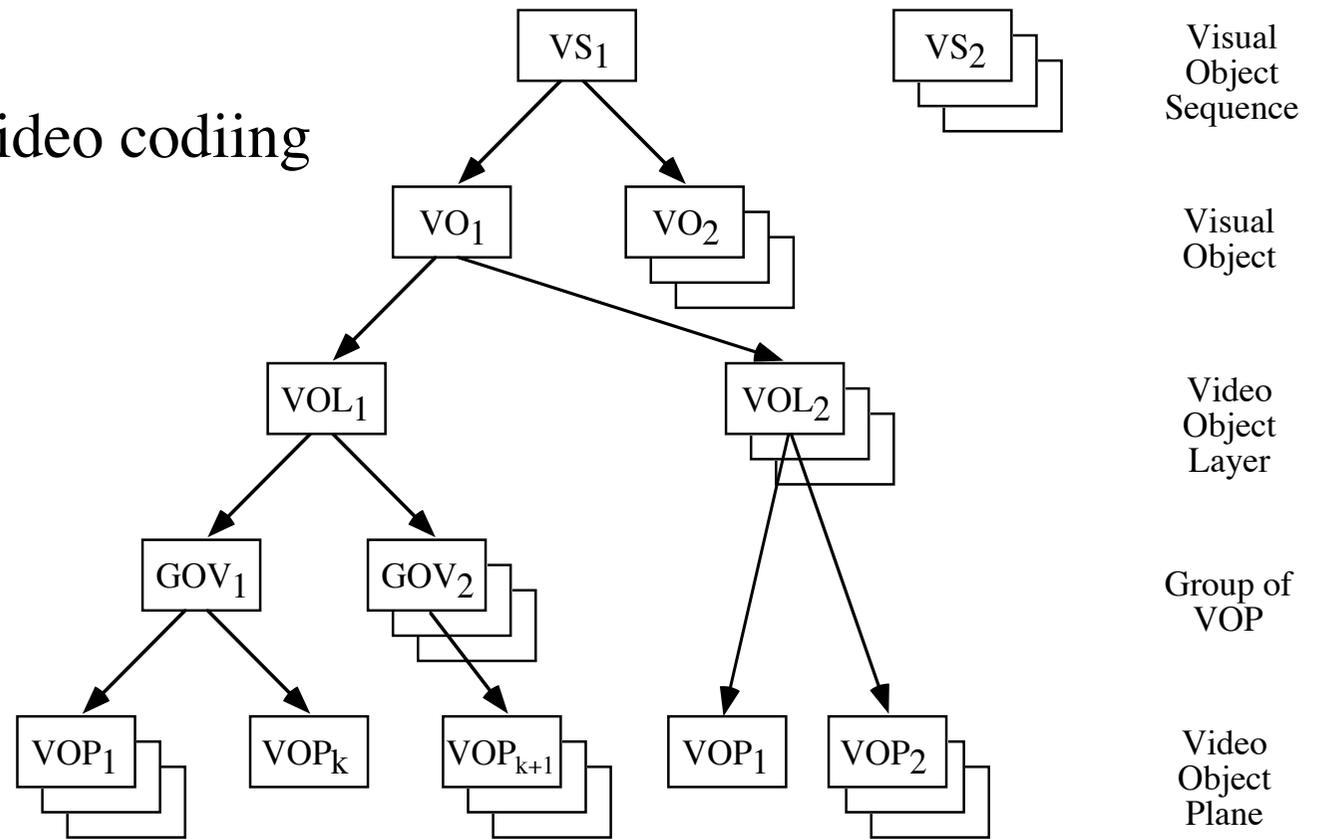


- Video Object Layer

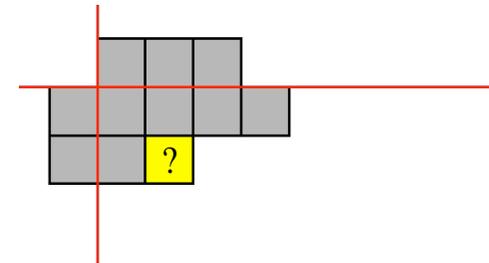
- mehrere bei scalable video coding
- Auflösung
- Bildqualität

- GOP

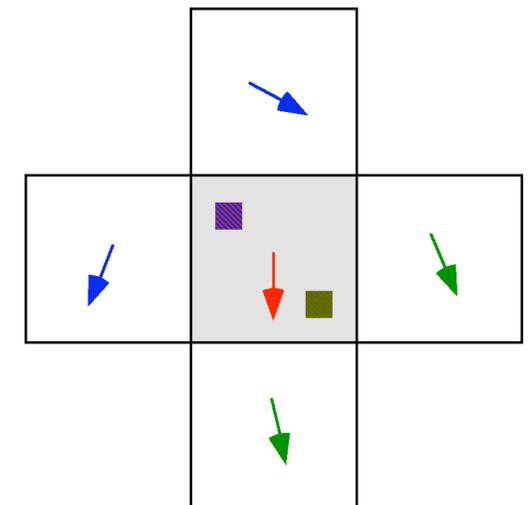
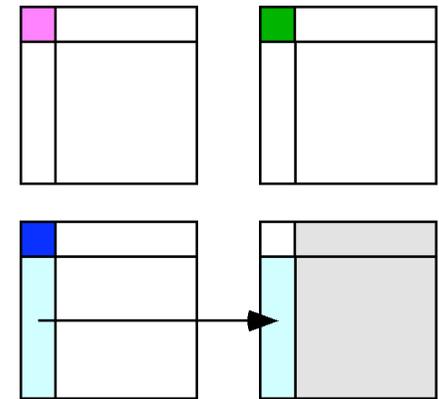
- Random Access
- Re-Synchronisation



- Alpha-Kanal (shape)
  - kontrolliert Mischen der Videoobjekte
  - einfache Maske als Bitmap: Pixel einsetzen oder nicht
  - oder Transparenzwert
  - oder Funktion zur Pixelverknüpfung
- Shape-Kodierung
  - alpha-plane als Bitmap
  - umfassendes Rechteck und Position im Frame
  - evtl. globaler, nicht-binärer alpha-Wert
  - transparente und deckende Makroblöcke
  - Rand-Makroblöcke: binary alpha blocks
  - bab mit Praediktionstechnik vorhergesagt
  - Wahrscheinlichkeit wird vom ArithKoder verwendet
- Kodierung für 'echten' alpha-Kanal
  - binäre Hilfsmaske wie Bitmap-Maske kodiert
  - Texture-Coding für alpha-Blöcke
  - Rand-Makroblöcke mit besonderen Techniken



- Texture-Kodierung
  - I-VOP, B-VOP, P-VOP
  - GOV: Group of VOPs
  - inter- und intrakodiert
- Prädiktion in Intra-Bildern
  - für eine AC-Koeffizienten Spalte/Zeile
  - Auswahl durch DC-Differenzen (Gradient)
  - Prädiktionsfehler für Spalte/Zeile kodiert
  - Rest vom Block DCT-kodiert
  - verschiedene Linearisierungs-Sequenzen
- 1 oder 4 Motion-Vektoren pro Makroblock
  - VOP-global motion compensation: 4 globale MVs
- Overlapped Block Motion Compensation
  - Pixelprädiktion und Bewegungsvektoren
  - Pixelprädiktion mit Nachbarvektoren
  - $\text{Pixel} := a_i * p[\text{nvec}_1] + b_i * p[\text{nvec}_2] + c_i * p[\text{blockvec}] - \text{Rest}$
  - Nachbarvektorenwahl abhängig vom Quadranten
  - DCT über Reste



- Still texture object
  - Wavelet-Kodierung möglich
  - layered bitstreams: base, enhancement(s)

- Sprites

- Sprite ändert sich nicht in der Zeit
- übertragen in I-VOP
- Ausschnitt im Bild sichtbar
- static video object planes (S-VOP): 3-D-Vektoren für Sprite
- Schwenk, Zoom, ...
- ->Hintergrund verändert sich scheinbar



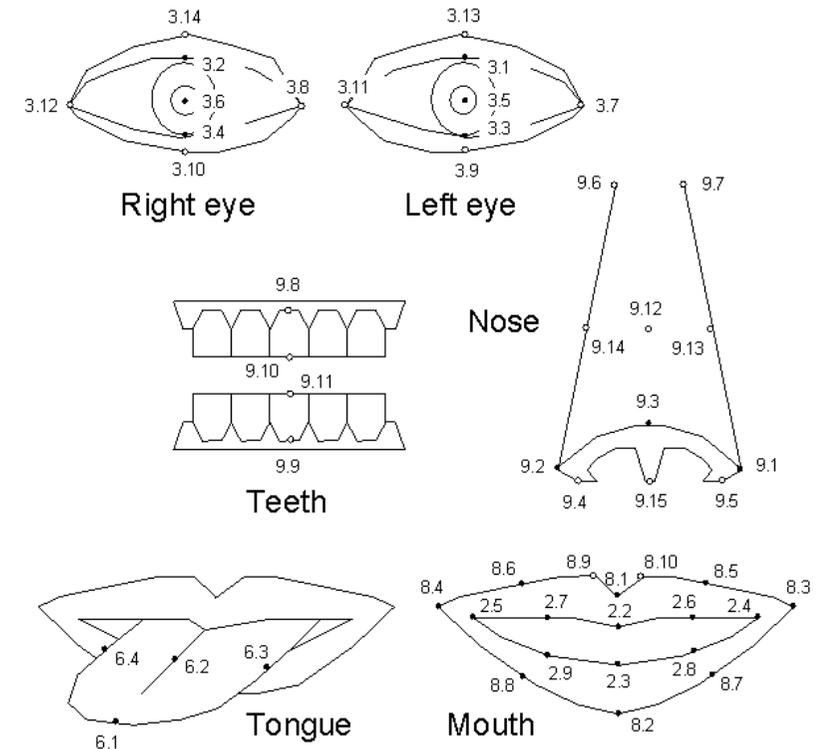
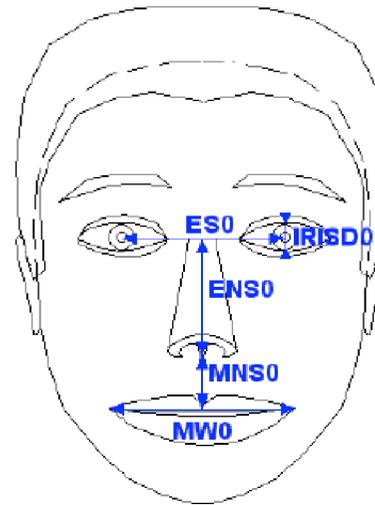
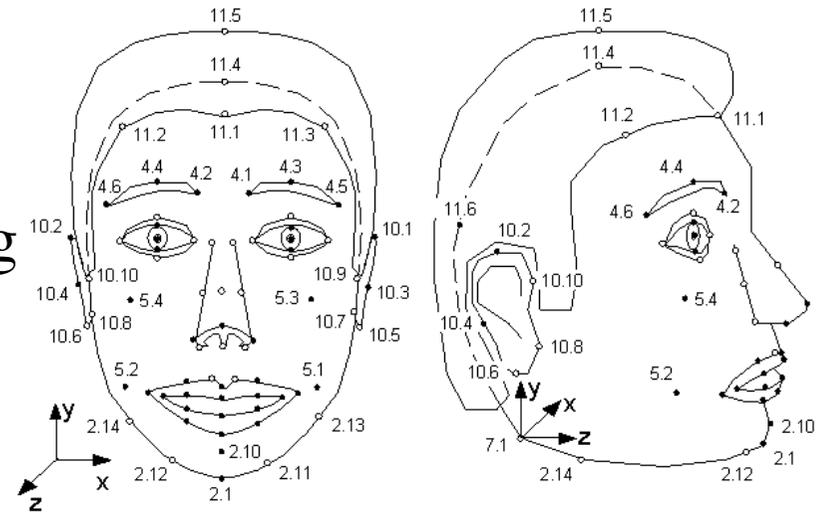
- 2-dimensionale Gitter (2D-Mesh)

- Dreiecks-Netz
- Vektoren für Gitterpunkte -> verzerrtes Gitter
- Prädiktion für Gitterpunktvektoren ...
- I-VOP-Textur wird mit dem Gitter verzerrt
- P-VOP-Fehler-Textur addiert

- 3D-Meshes siehe BIFS/VRML

- Face animation object

- Modell mit generischem Gesicht
- FDP: Face Definition Parameters
- BIFS-Mechanismen zur Modellübertragung
- Default Modell im Decoder
- FAP: Face Animation Parameters (68)
- Übertragung der Parameter in FAPU



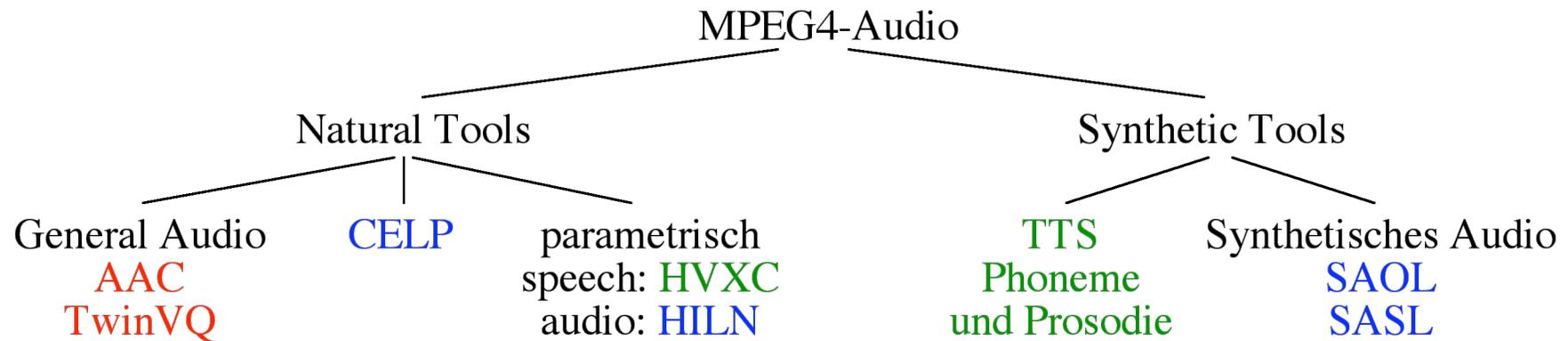
• Feature points affected by FAPs  
 ○ Other feature points

- Body animation object

- 3D-Polygonnetze
- BDP: Oberfläche, Grösse, Textur
- Body Animation Parameters (168)
- Standardposition frontal stehend
- Mechanismus zur Übertragung der Parameter

## 1.1.4.4 MPEG-4 Audio (Überblick)

- AudioBIFS (siehe 2.4.6.1)
- Coding Tools
  - De-Kompressionsverfahren: parametrisch oder psychoakustisch
  - Synthese



- Bitraten pro Kanal : 2-4, 4-16, 16-64 [kbit/s]
- Siehe 1.1.4.6

## 1.1.4.5 Profile in MPEG-4

- Untermengen der Kodiermöglichkeiten (Tools)
  - Endgeräte-Klassen und Szenarien
  - Conformance-Tests
- Visual Profiles für Natural Video
  - simple: rechteckig, fehlertolerant (z.B. Mobilfunk)
  - simple scalable: Auflösung (Raum, Zeit) skalierbar
  - Advanced simple: B-Frames+1/4 pel+GMC -> Internet-Video bis TV
  - advanced real-time simple (ARTS): Videotelefonie, Fehlertoleranz
  - core: Objekte beliebiger Form, sprites, Interaktivität
  - core scalable
  - main: interlaced, transparent, Sprites; Broadcast und DVD
  - advanced coding efficiency: core+1/4 pel+GMC -> 'mobile broadcast'
  - n-bit: andere Bit-Tiefen für Überwachungskameras etc.
  - simple studio (nur I-Frames, 180-1800 Mbit/s)
  - core studio (900 Mbit/s)

- Visual Profiles für Synthetic Video
  - simple face animation, simple face and body animation
  - scalable texture (advanced scalable texture)
  - basic animated 2-D texture
  - hybrid: natural und synth. Video
- Audio Profiles
- General Audio
  - Mobile Audio Interworking: AAC, TwinVQ
- Speech und general Audio
  - scalable
  - high quality (AAC LTP, CELP),
  - natural: alle natural-tools
- Synthetic:
  - SAOL, midi, wavetable, TTSI
- Hybrid natural/synthetic audio
  - speech: 1-20 Objekte, CELP, HVXC, TTSI
  - main: alle tools
  - low delay audio: HVXC, CELP, AAC, TTSI

- Graphics Profiles
  - simple 2-D, complete 2D, basic 2D, core 2D, advanced 2D
  - complete: volle BIFS-Funktionalität
  - X3D interactive
  - 3D audio
- Scene Graph Profiles
  - simple facial animation
  - scalable texture
  - simple face and body animation
- MPEG-J Profiles
  - personal: PDAs, gameboys, etc.; nur network, scene, resource API
  - main: personal+decoder+ decoder function+... APIs
- Levels für Profiles: Auflösung

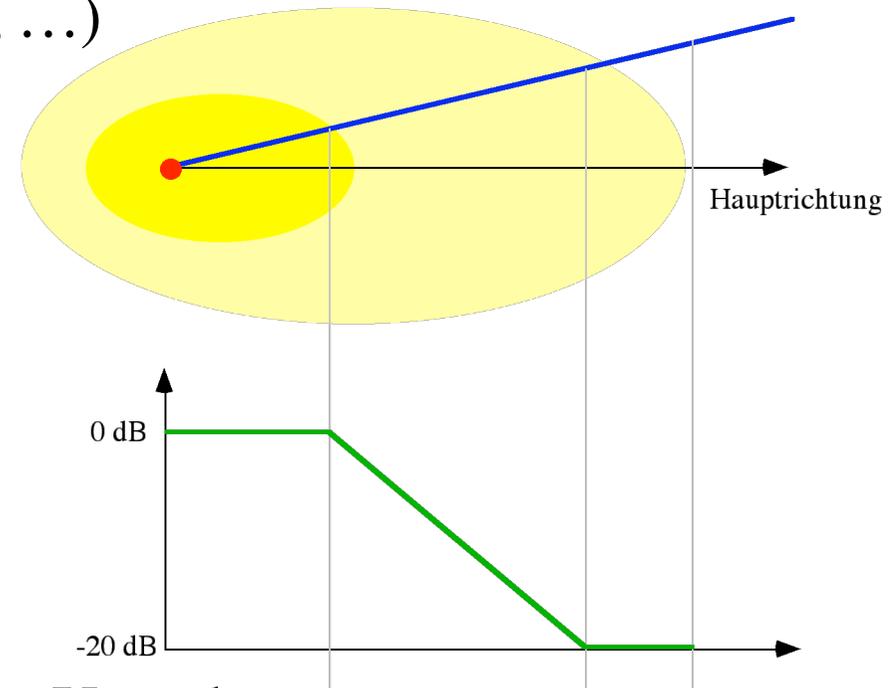
## 1.1.4.6. Audio Objects (siehe [Scheirer et al., 1999])

- AudioBIFS

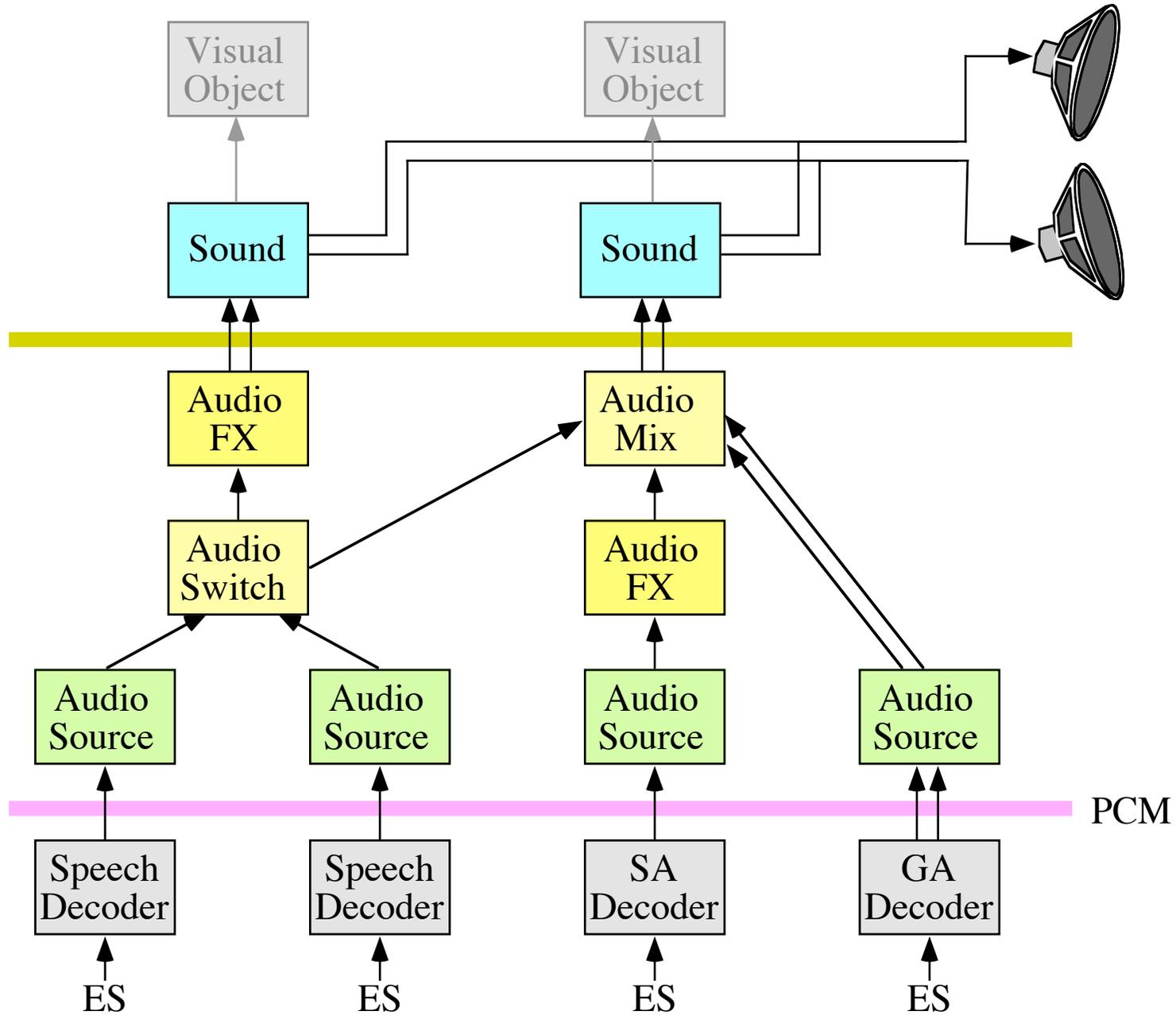
- Subgraph mit Signalverarbeitungsfluss
- Objekt in der Szene (3D-Position, Ellipse, ...)
- AudioSource: Decoder, URL, startTime, stopTime, numChan, ...
- AudioMix und AudioSwitch: m Kanäle -> n Kanäle
- AudioFX (SAOL-Programm)
- Group, Transform, ...
- ListeningPoint

- MPEG-4V2: Virtuelle Akustik

- Modellierung der (künstlichen) akustischen Umgebung
- Modelle für Quelle, Umgebung, Wiedergabeumgebung
- Dopplereffekt
- AcousticScene
- AcousticMaterial: Transferfunktionen Durchlässigkeit und Reflektion
- DirectiveSound: spektrale Auswirkungen der Position



- Beispiel für Audio-Szenengraph



- Beispiel AudioBIFS [Scheirer et al., 99]

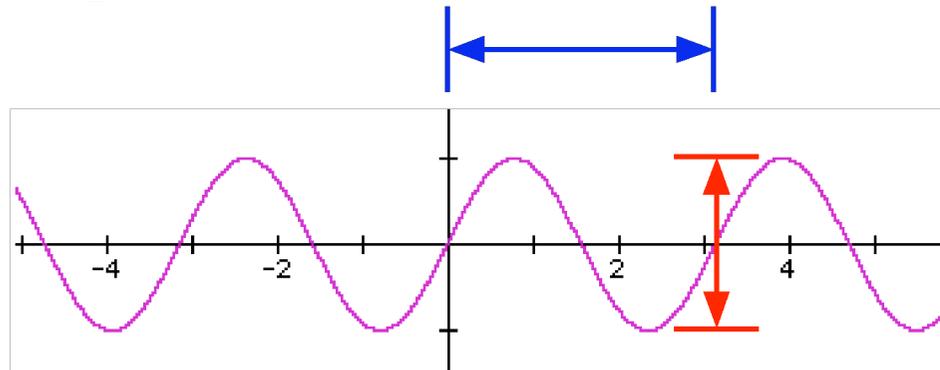
```

Group {
  children [
    sound {
      spatialize FALSE
      source [
        AudioMix {
          numChan 2 // 4 Eing., 2 Ausg.
          phaseGroup [1,1]
          matrix [0.8 0.4 1.0 0.1 0.4 0.8 0.1 1.0]
          children [
            AudioSource {
              url "<string>"
              numChan 2 // 2 Eing.
              phaseGroup [1,1]
            },
            AudioFX {
              orch "<SAOL bytecode>" // mono -> stereo
              numChan 2 // 2 Ausg.
              children [
                AudioSource {
                  url "<string>"
                  numChan 1 // 1 Eing.
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}

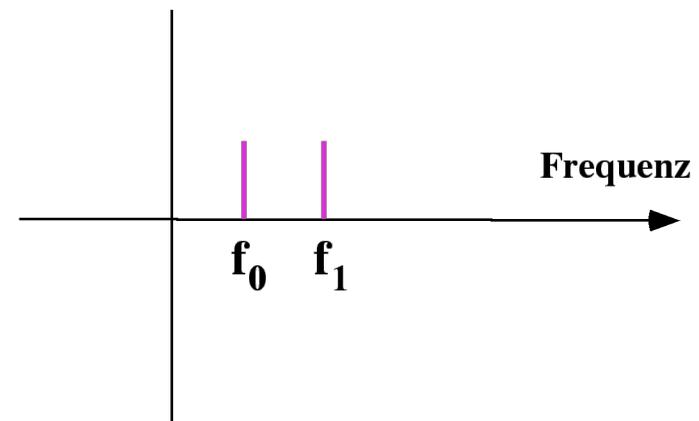
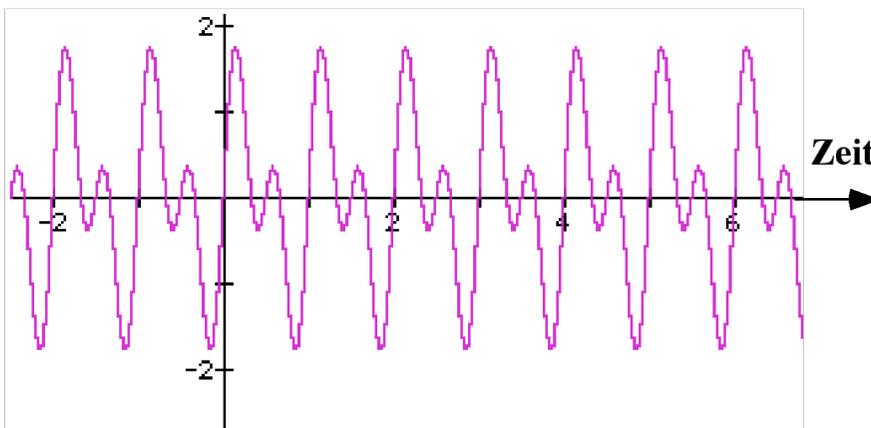
```

## 1.1.5 Audio-Kompression

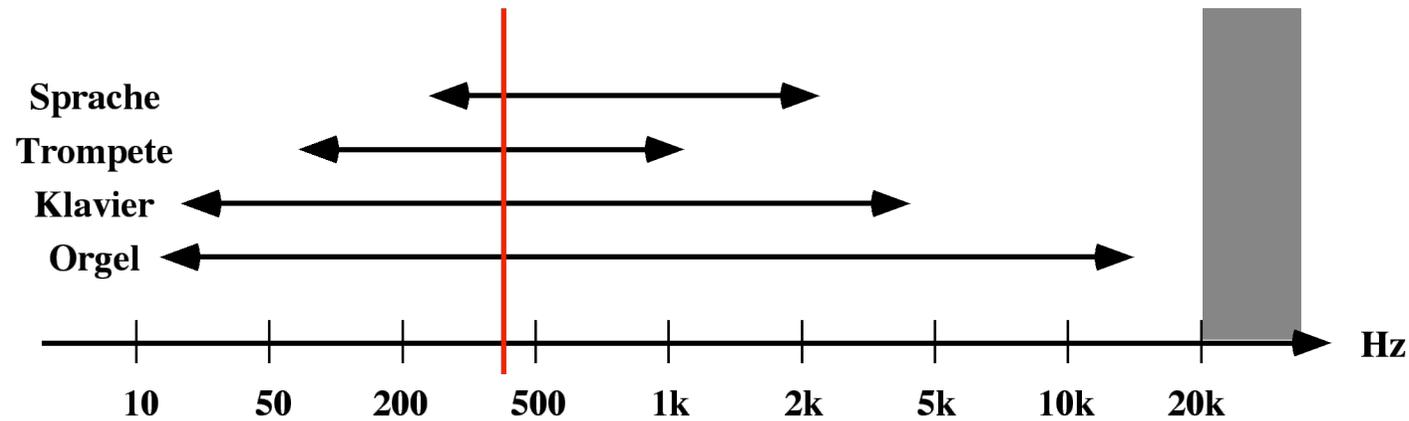
- Wiederholung: Audioeigenschaften
- Frequenz und Amplitude



- Amplitude -> Lautstärke (gemessen in dB)
- Frequenz (1m/Wellenlänge) -> Tonhöhe
- Fourier: Jede Schwingung kann als Summe von Sinusschwingungen dargestellt werden:  $f(x) = \sin 2\pi x + \sin 4\pi x$



- Typische Frequenzbereiche

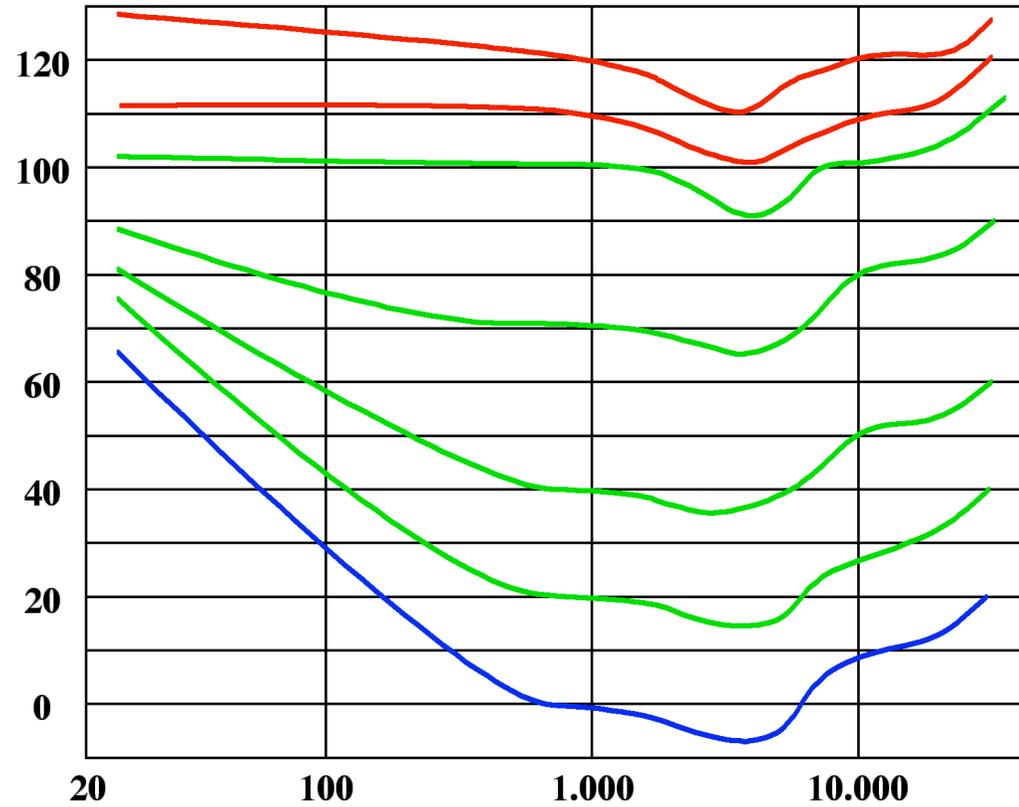


- Telefon 300Hz - 3.400 Hz
- Heimstereo 20 Hz - 20.000 Hz
- UKW (FM) 20 Hz - 15.000 Hz

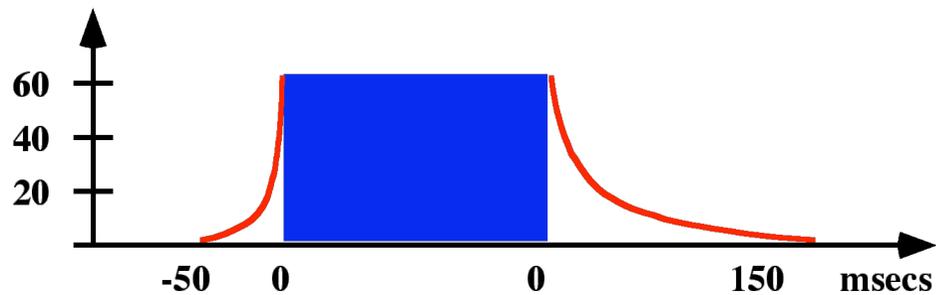
- Menschliches Hörvermögen

- 20 - 20.000 Hz
- hohes zeitliches Auflösungsvermögen
- logarithmisch bezüglich Amplitude

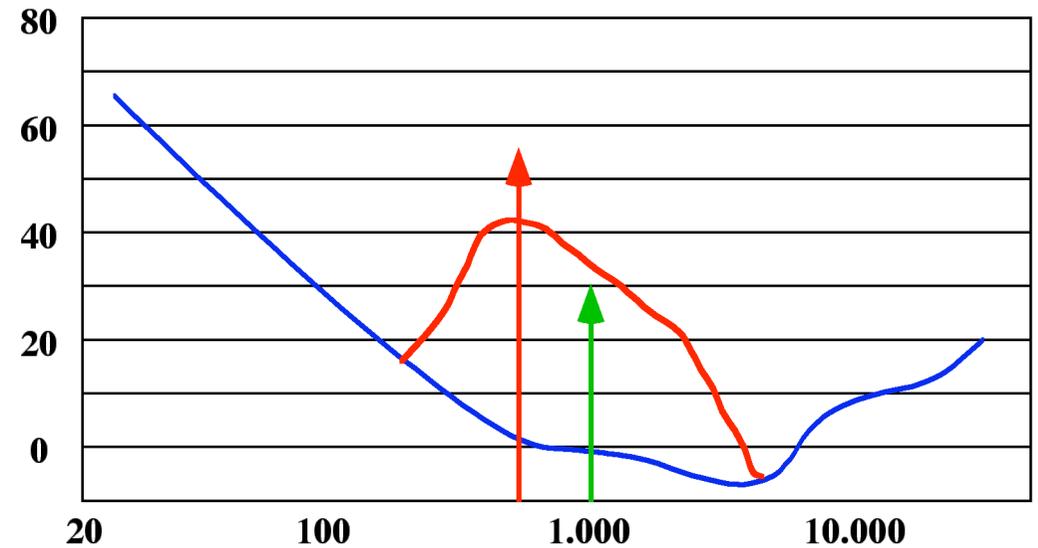
- Lautstärkeempfinden nach Fletcher und Munson



- Abschattung  
- Zeit



## Frequenz

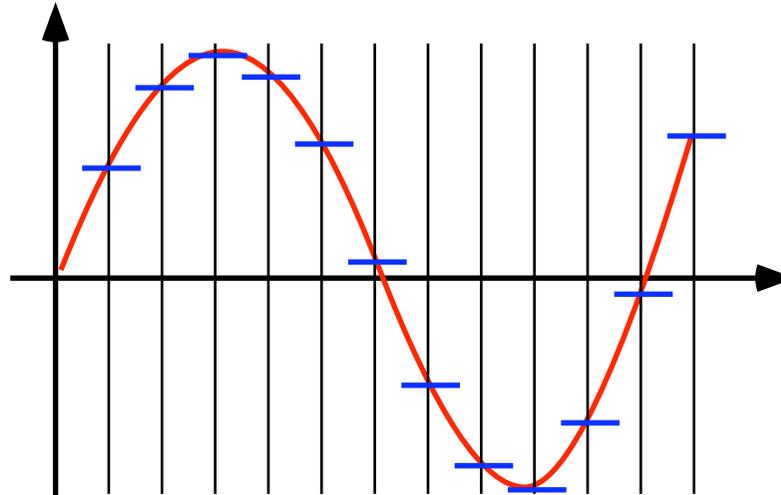


- Perceptual Coding

- Filterbänke zerlegen in Bänder
- Maskierung im Band und zwischen Bändern
- Wahrnehmungsmodell liefert Quantisierung für Bänder
- Transformation temporale in spektrale Darstellung
- Entropiekodierung

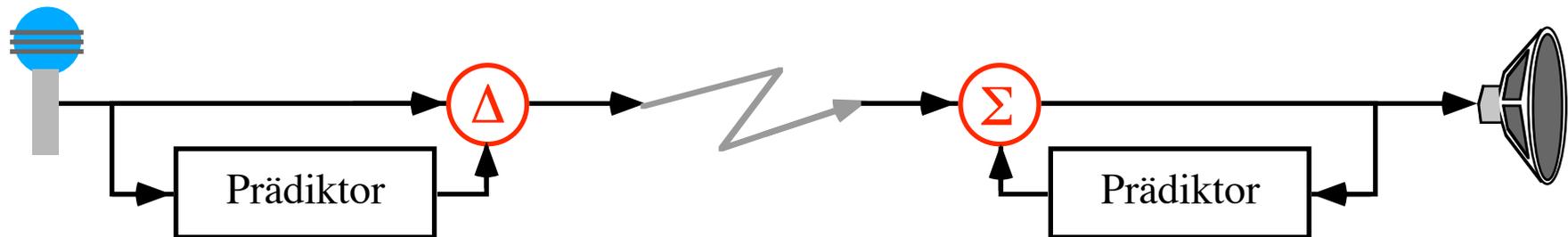
## 1.1.5.1 Predictive Coding

- Stetige, glatte Kurve => nächstes Sample 'in der Nähe'
  - $E(s_i - s_{i-1}) \ll E(s_i)$
  - Schluß von  $s_{i-1}$  auf  $s_i$



- Übertragung der Differenzen zwischen Samples
- Variable Bitlänge des Codes
- Delta-Modulation
- eventuell nur  $\pm 1$  pro Sample

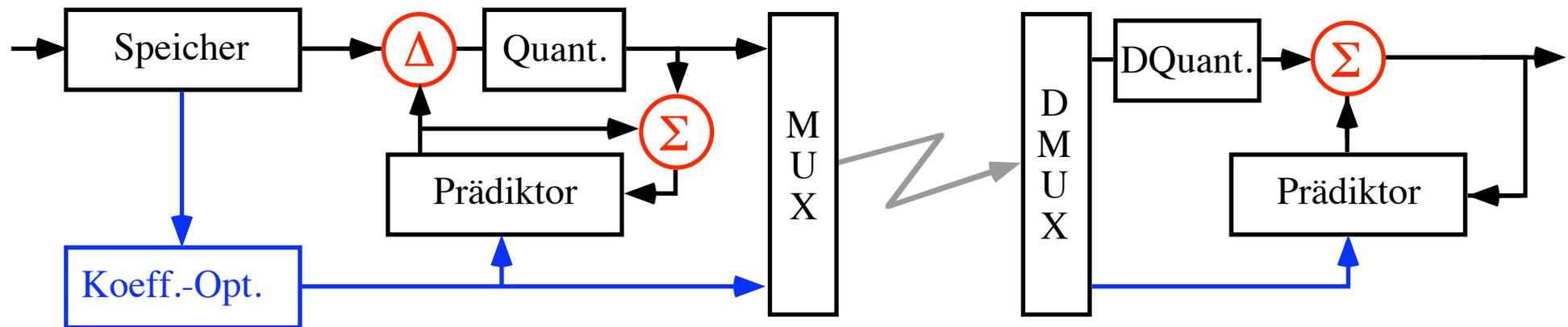
- Nächstes Sample in ähnlicher Richtung
  - $E(f'(s_i) - f'(s_{i-1})) < \varepsilon$
  - Schluß von  $(s_{i-n}, \dots, s_{i-1}, s_i)$  auf  $s_{i+1}$
  - Vorhersage des nächsten Samples
  - Übertragung des Fehlers der Vorhersage
  - Differential PCM (DPCM)



- Algorithmus der Prädiktion
  - Extrapolation
  - vorhergehende n Samples mit Gewichten  $a_i, i=1..n$
  - Prädiktorkoeffizienten

- Adaptive DPCM (ADPCM)

- Optimierung der Prädiktionskoeffizienten:  $d_i = s_i - s_i^*$  minimal
- Intervallweise Optimierung
- Übertragung der Koeffizienten zum Empfänger
- CD-Interactive
- ITU G.721, G.722



- Rekonstruierte Werte zu Prädiktorberechnung

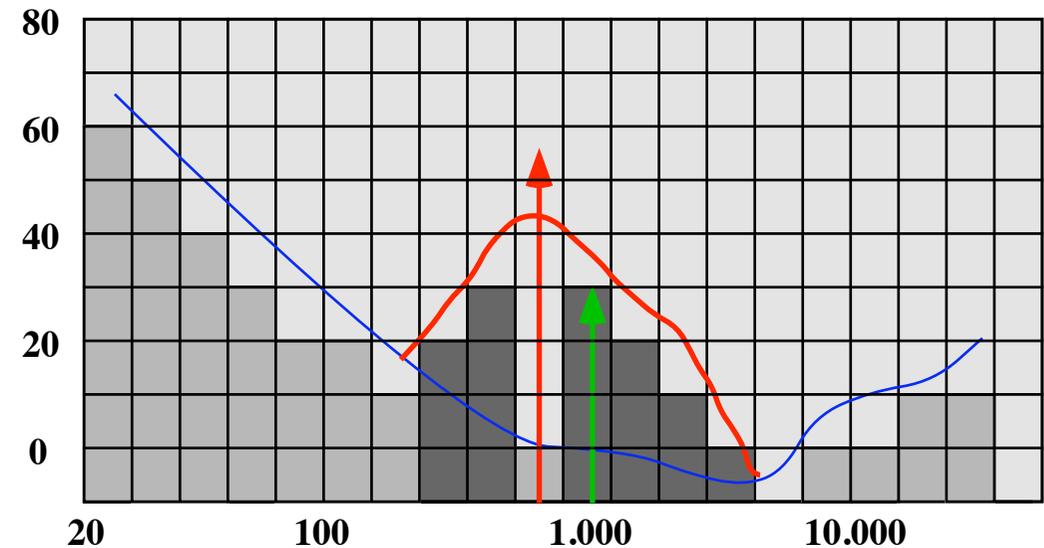
## 1.1.5.2 MP3

- MP3

- MPEG - Moving Pictures Expert Group
- Video
- Audio-Layers 1, 2, 3 => mp3

- Idee

- Subband-Kodierung
- psychakustische Effekte ausnutzen



- Frequenzspektrums in Bänder zerlegen

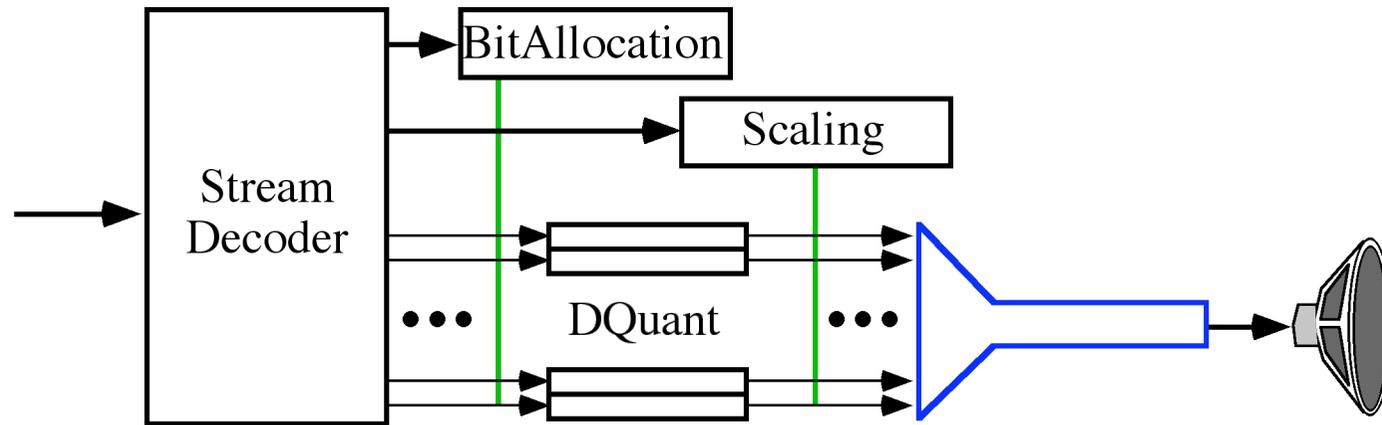
- Filterbank mit n Filtern
- Bandbreite  $F/n$
- Bsp.: DAT 48 kSamples/s, 32 Bänder (MPEG), Bandbreite = 750 Hz

- Nur hörbare Bänder übertragen

- normale Fletcher-Munson-Kurve
- Maskierung durch andere Bänder

- MPEG

- Modelldekker vorgeschrieben
- Datenstromformat vorgeschrieben
- Beispielkoder im Standard

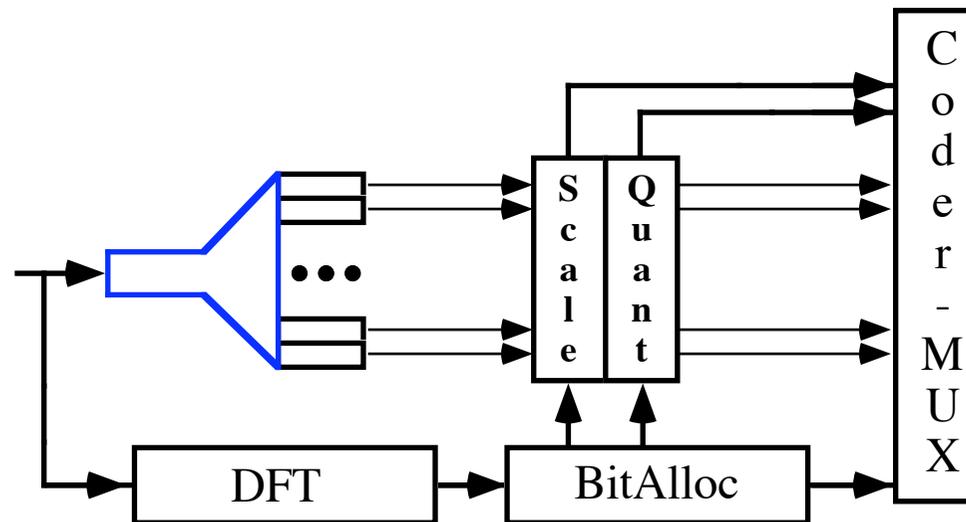


- Synthese-Subband-Filter

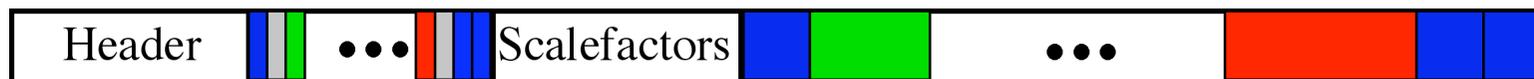
Siehe z.B. N. Fliege: Multiraten-Signalverarbeitung, S. 219 ff.  
Aufwärtsabtastung, Filtern, Summieren

## • Beispielkoder

- Bandaufteilung mit Analyse-Subband-Filter
- Rahmen (Frame) mit  $384 = 12 * 32$  Samples  $\Rightarrow$  12 Samples pro Subband
- Maskierungswerte (Hörschwelle, andere Bänder) berechnen
- Frequenzbänder aussondern
- Skalieren (pro Frame und Band)
- Quantisieren
- Iterative Bitzuweisung an Bänder, konstante Bitrate

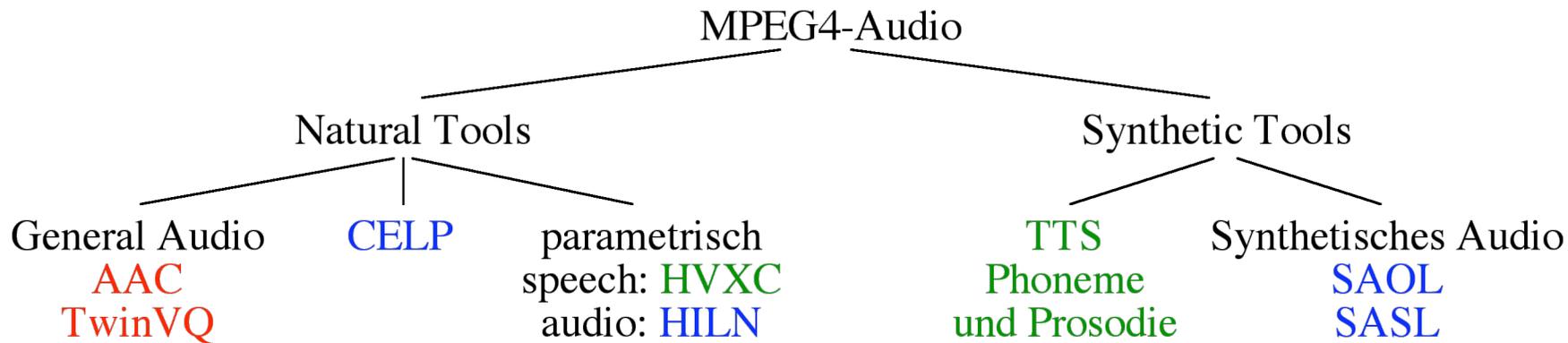


- Analyse-Subband-Filter
  - 32 neue Werte in  $X[0..511]$  shiften (Modulationsvektor)
  - Mit Fenster gewichten
  - Mit Modulationsmatrix multiplizieren
  - 1 Wert pro Band, 32 Bänder
- Psychoakustische Modell
  - Fourier Transformation
  - Schalldruck in jedem Band bestimmen
  - Masken pro Band bestimmen
  - Signal-to-Mask-Ratio SMR berechnen
- Datenstrom



- MPEG Audio Layers
  - I: mit einfacher Filterbank
  - II: mehr Samples, feinere Samplekodierungsmöglichkeiten
  - III: Hybridfilter (Subband, MDCT), adaptive Segmentierung  
MDCT: Modified DCT for Audio Layer III
- Unterstützte Abtastraten: 32, 44.1 und 48 kSamples/s
- Stereo
- MPEG-Audio Datenstrom
  - Layer I:  $n \cdot 32$  kbit/s,  $n = 1, \dots, 14$
  - Layer II: 32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384 kbit/s
  - Layer III: 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320 kbit/s
- Ab 192 kbit/s kein Unterschied zum Original hörbar

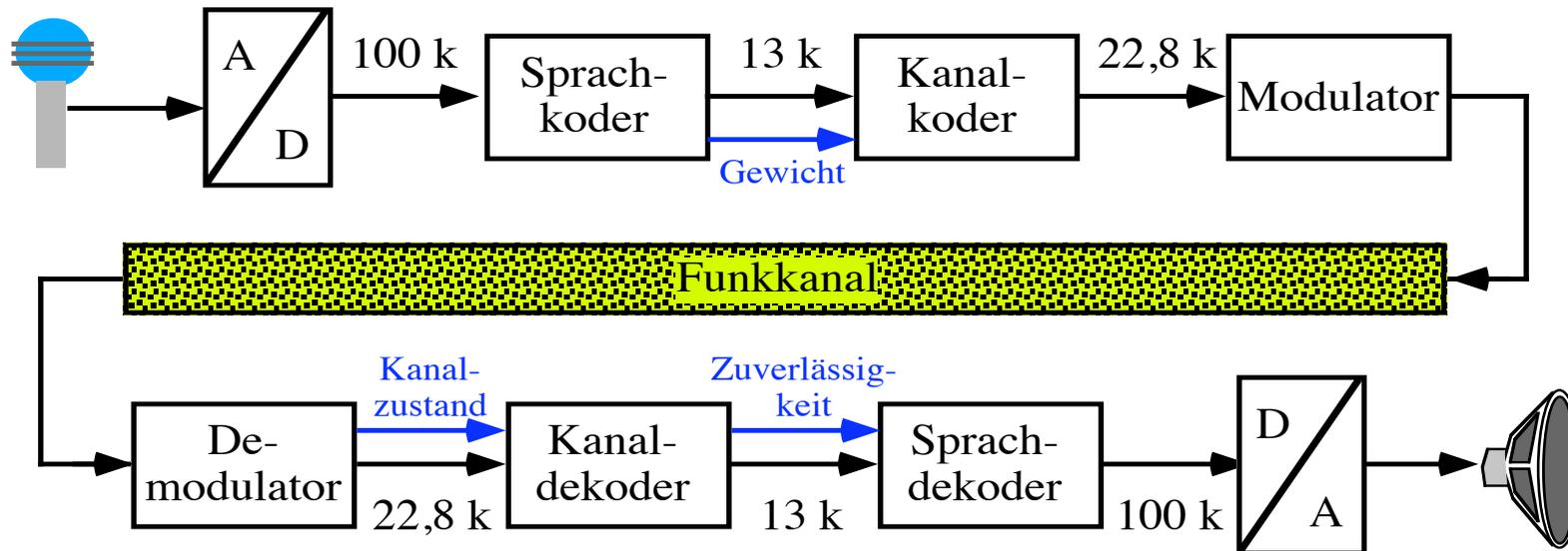
- MPEG-4



- verbessertes AAC
  - bandweise Noise Substitution
  - Long Term Prediction (LTP) und Differenzkodierung
- TwinVQ: Transform-domain Weighted Interleave Vector Quantization
  - alternative Kodierung der spektralen Koeffizienten
  - Normalisierung und Vektorquantisierung (inkl. Codebuchauswahl)
  - guten Ergebnisse bis zu 6 kbit/s
- Low Delay Audio Coding abgeleitet von AAC (Qualität vgl. mp3)
  - MPEG-4 GA bei 24 kbit/s: 110+210 msec Verzögerung
  - Low Delay AC: 20 msec
  - halbe Framelänge: 512/480 Abtastwerte und halbe Fenster
  - kleines oder kein Bit-Reservior

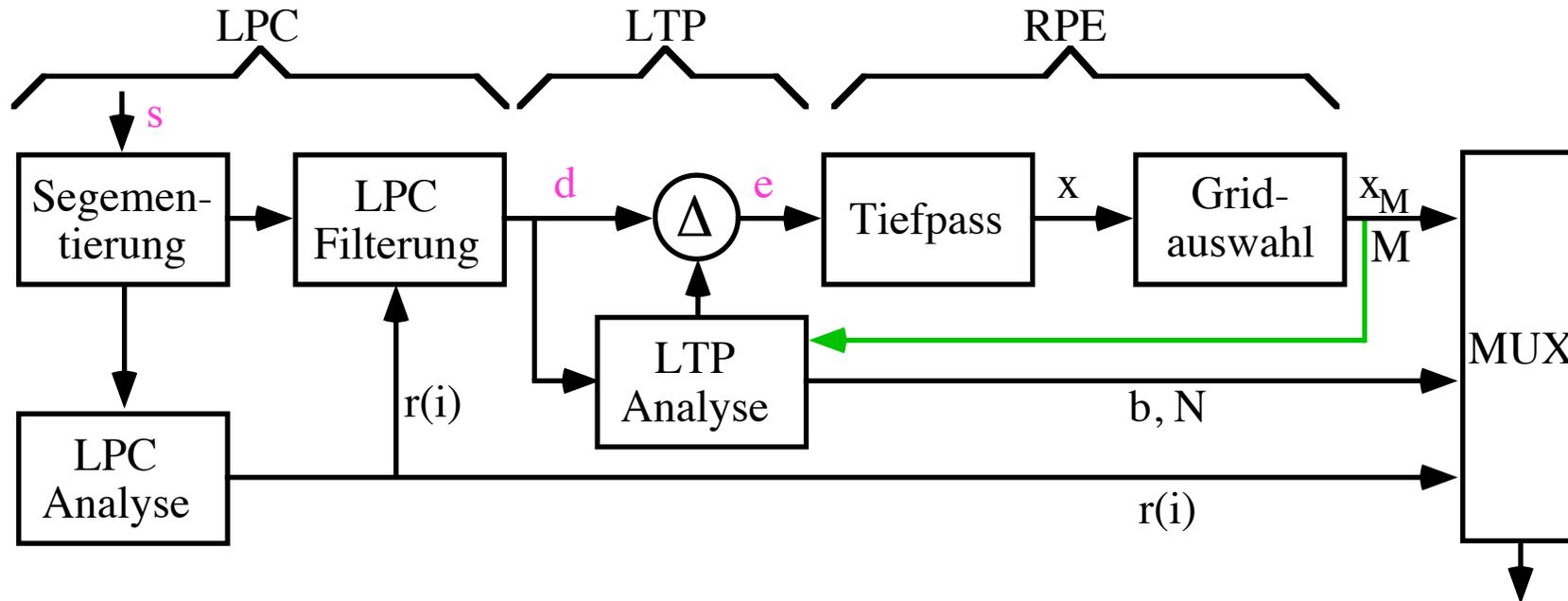
## 1.1.5.3 GSM

- Nachempfinden des menschlichen Sprachapparates
- Integrierte Quell- und Kanalkodierung
  - Einteilung in geschützte und ungeschützte Information
  - Signal-Qualitätsinformation für den Dekoder



- ca. 2 DSP-Mips

- Sprachkoder

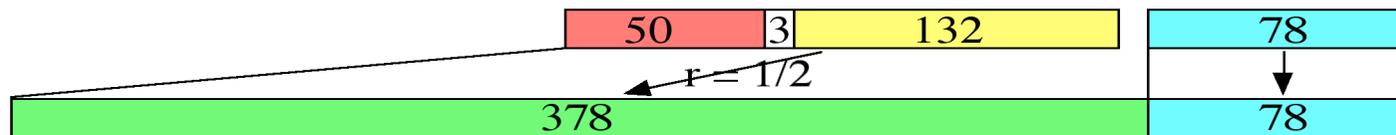


- 8 Linear Predictive Coding (LPC) Koeffizienten aus 160 Samples
- LPC-Koeffizienten Filter laden
- LPC-Filter transformiert 160 PCM-Samples
- Differenz zu Vorhersagesignal (Long-Term-Prediction LTP) bilden
- 3 (4) verschränkte Folgen von Differenzsamples bilden (Gitter)
- Folge mit maximaler Energie auswählen
- Quantisieren
- Multiplexen

- Kodierung

Parameter	Bit/Block	Bitrate
- LPC-Filterkoeffizienten 6,6,5,5,4,4,3,336		1,8 kbit/s
- LTP Delay	28	1,4 kbit/s
- LTP Gain	8	0,4 kbit/s
- RPE-Grid	8	0,4 kbit/s
- RPE Skalierung	24	1,2 kbit/s
- RPE Werte	156	7,8 kbit/s

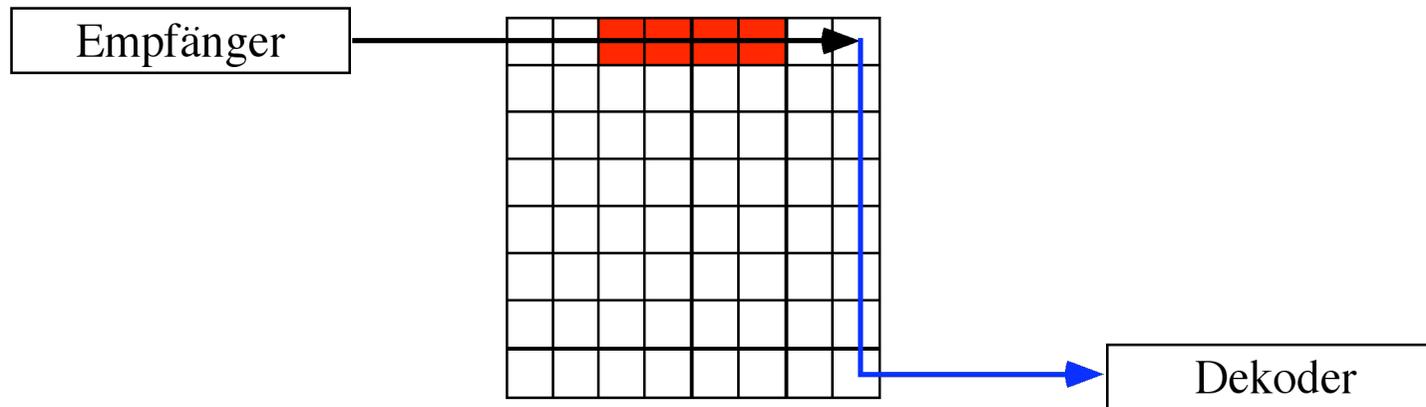
- Kanalcodierung



- Klasse Ia: LPC-Parameter (MSBs), LTP-Parameter
- Klasse Ib: LPC-Parameter, Gitter-Auswahl, RPE-Daten (MSBs)
- Klasse II: LPC-Parameter (LSBs), RPE-Daten (LSB)

- Interleaving

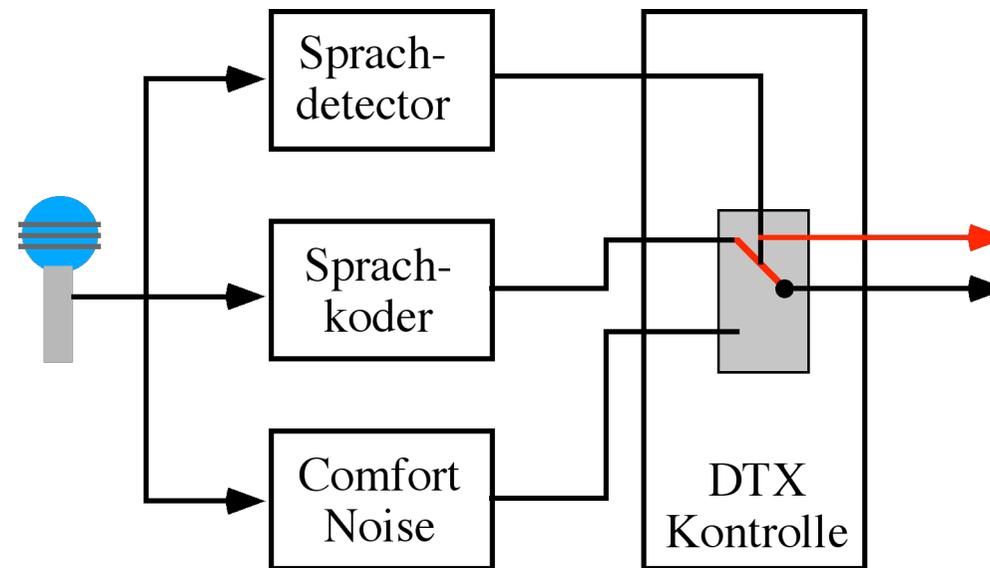
- Burstfehler verteilen
- Faktor 8



- Faltungscodes gut bei "gestreuten" Fehlern

- Discontinuous Transmission (DTX)

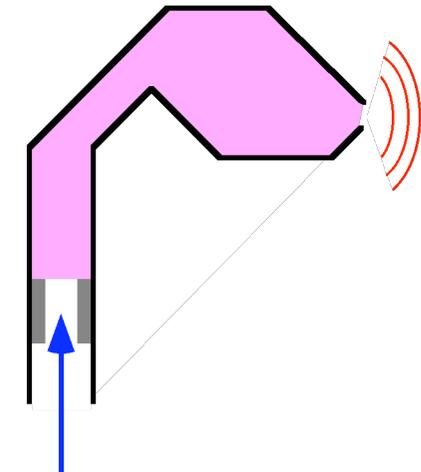
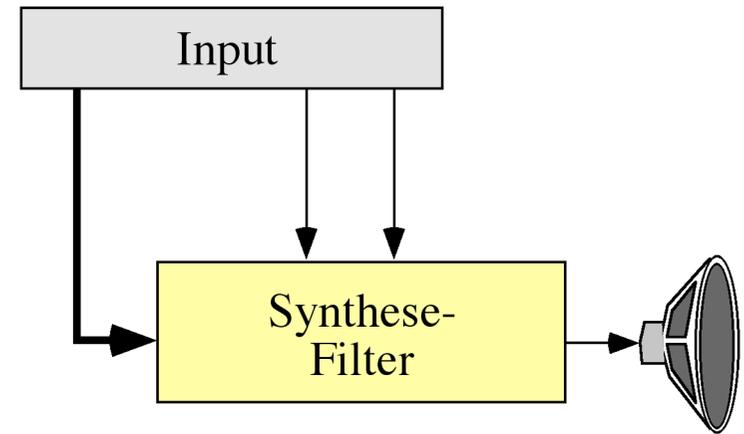
- Voice Activity Detection
- In Sprachpausen keine GSM-Bursts senden
- Periodisch Ruhegeräusch senden
- Ruhegeräusch im Empfänger abspielen



- reduziert Interferenzen zu anderen Mobiltelefonen
- spart Batterie

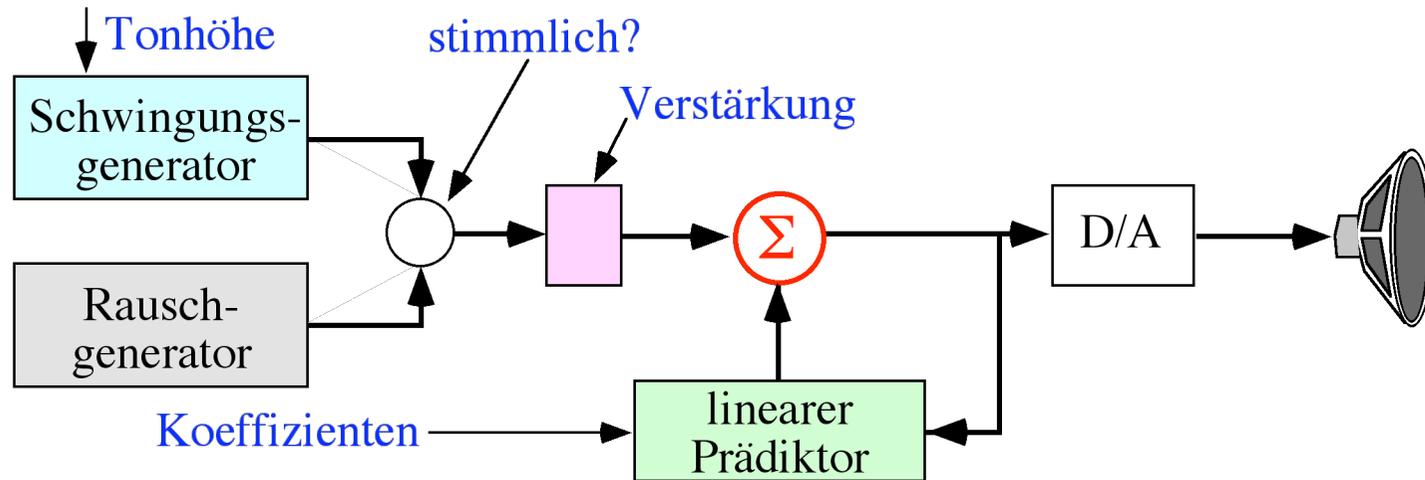
## 1.1.5.4 Weitere Ideen

- Parametrische Kodierung
  - niedrige Bitraten von 1.2 – 18 kbit/s
  - Decoder: Filter zur Sprachsynthese
  - Filterparameter beschreiben das Modell
  - Bitstrom mit Anregungsdaten für Filter
- Modell für Sprachapparat
  - Luftproduzent (Lunge)
  - Schwingungserzeugung (Stimmbänder)
  - Lautformung in 'Röhre' (Mund, Lippen, Nase)
  - stimmliche Laute durch Schwingung und Formung
  - stimmlose Laute mit Rauschen und Formung
- Sprachkodierung
  - Unterscheidung stimmlos / stimmlich
  - Tonhöhe für stimmliche Laute
  - Formanten bestimmen (-> Parameter)
  - Residuum kodieren (-> Anregungsdaten)

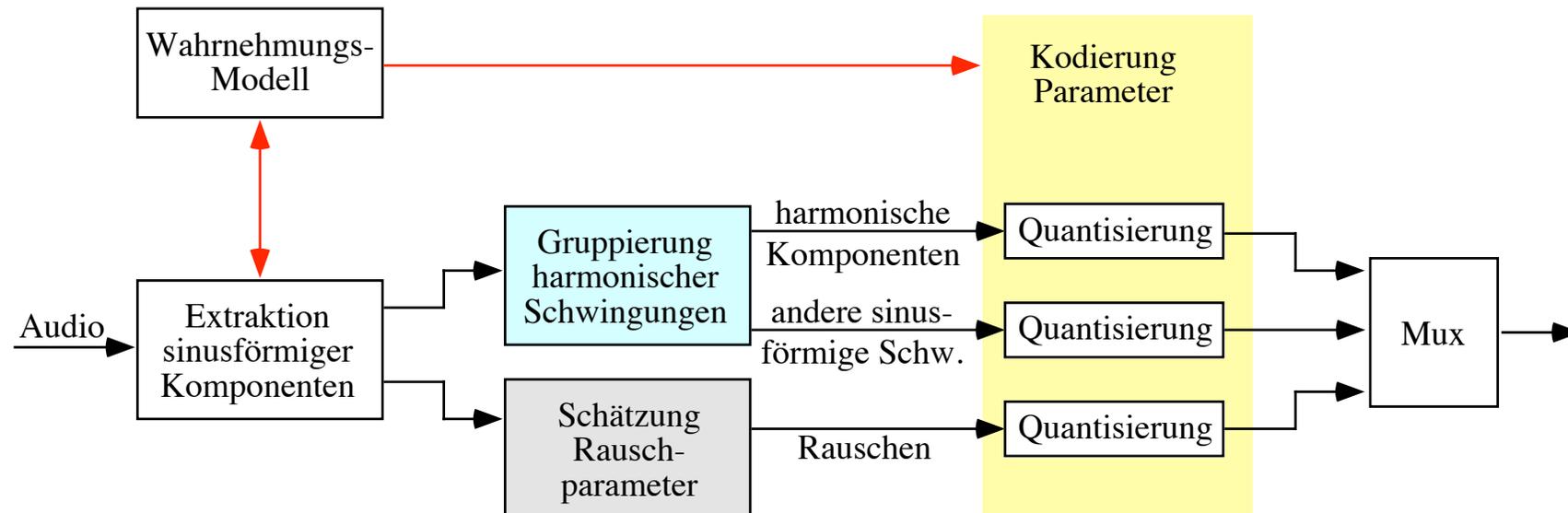


- CELP: Code Excited Linear Prediction
  - 6, 8.3, 12, 18 kbit/s
  - Analyse durch 'Synthese'
  - Erregungssignal ergänzt Generatorsignale
  - wird durch Ausprobieren bestimmt
  - Codebuch für Erregungssignal
  - adaptive Codebuchergänzung
  - Silence Insertion Descriptor und Comfort Noise
  - Wideband-CELP: 16 kHz
- HVXC: Harmonic Vector Excitation
  - 2 und 4 kbit/s; VBR: 1.2-1.7 kbit/s
  - Erregung für stimmliche Laute: Spektrum-Hüllkurve vektorquantisiert
  - Erregung für stimmlose Laute: Codebuch ähnlich CELP

- Linear Predictive Coding (LPC)
  - 'ADPCM ohne Prädiktionsfehlerübertragung'



- HILN: Harmonic and Individual Lines plus Noise
  - general audio bei 4 kbit/s
  - Zerlegung in Komponenten
  - Rauschen: Amplitude und spektrale Hüllkurve
  - sinusförmige Anteile: Frequenz und Amplitude
  - harmonische Anteile: Grundfrequenz und spektrale Hüllkurve

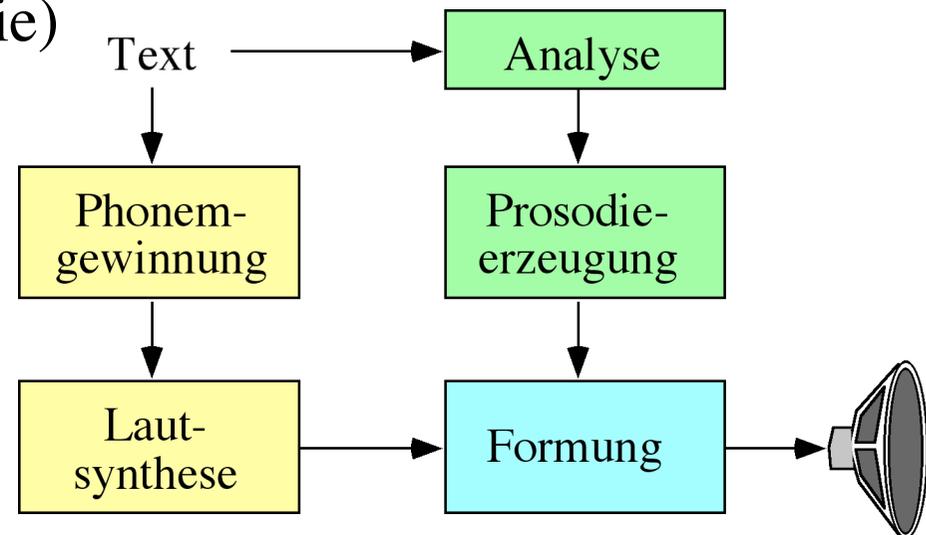


- Synthesized Sound

- Structured Audio Tool
- SASL: Structured Audio Score Language (-> 3.3)
- SAOL: Structured Audio Orchestra Language (-> 3.3)
- TTS: text-to-speech: ASCII + 'Ausspracheinformation'
- TTS zusammen mit Gesichtsanimation

- Text-to-Speech

- Text als Buchstaben-Strom
- Identifikation der Phoneme, phonembasierte Synthese
- Face-Animation-Parameter aus Phonemen gewinnen
- Parameter für das Synthesemodell
- Prosodie-Information (Satzmelodie)



## 1.1.5.5 Audio und Paketisierung

- Qualität und Kompression

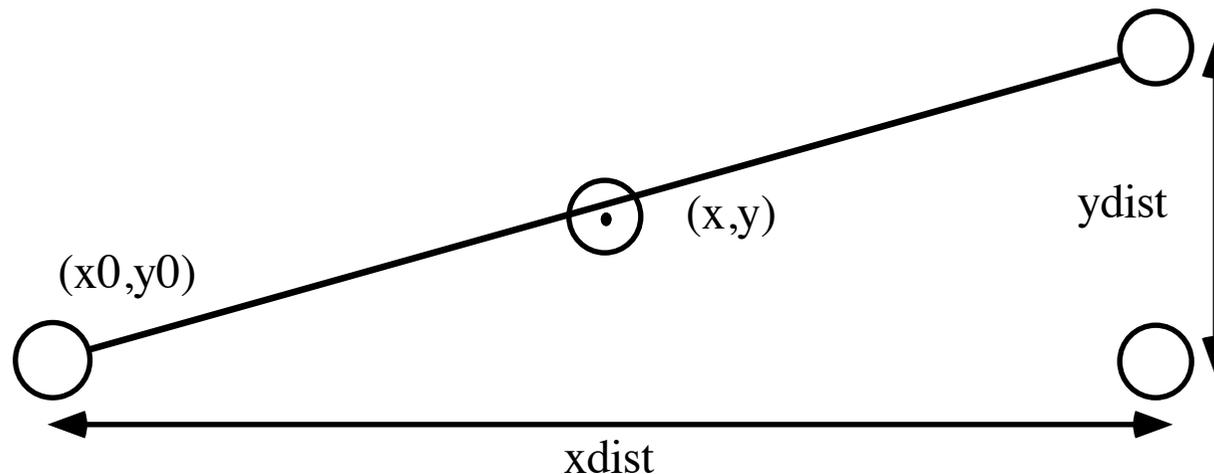
			500 byte	20 msec
- sehr gut:	MPEG	192 kbit/s	21 msec	500 byte
- mittel:	ADPCM	32 kbit/s	125 msec	80 byte
- Telefon:	ADPCM	16 kbit/s	250 msec	40 byte
- mäßig:	GSM	13 kbit/s	307 msec	32.5 byte
- schlecht:	hrGSM	4 kbit/s	1 sec	10 byte
- Kosten für kurze und lange Pakete gleich
- Konferenzdienste (Telefon, ...)
  - niedriges Delay → mittlere und hohe Bitraten
  - schlechte Bandbreitennutzung im Internet
  - Teilnehmeranschlußleitung limitiert
  - ADPCM oder GSM
- niedrige Bitraten
  - Delay unvermeidbar
  - geeignet für Verteildienste

## 1.2 Grafische Algorithmen

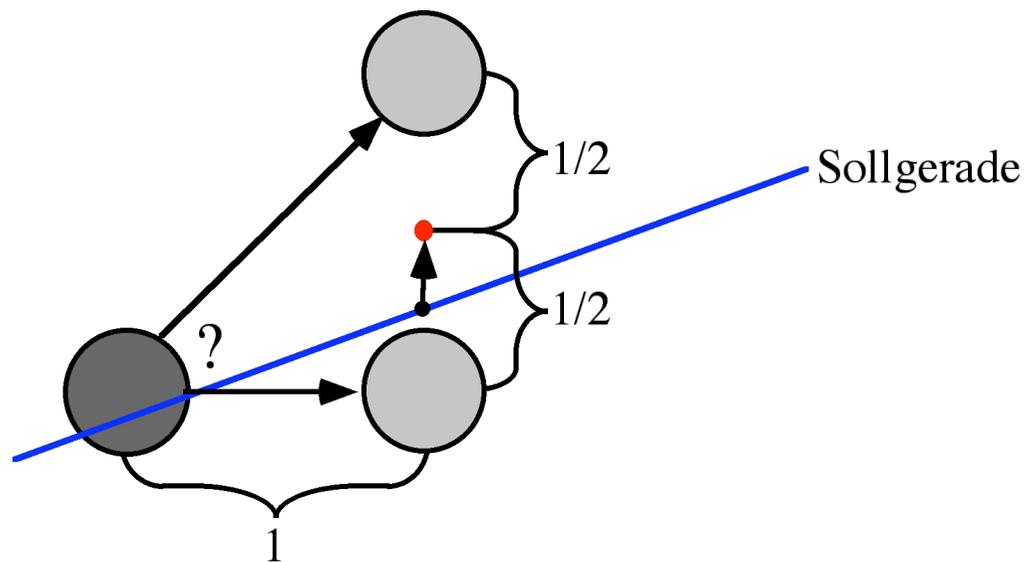
- Abbildung der geometrischen Objekte auf das Punkteraster
- Inkrementelle Algorithmen
- Integer-Arithmetik

### 1.2.1 Bresenham Algorithmus für Geraden

- Algorithmus nach Bresenham (IBM Syst. Journal, Vol. 4, No. 1, 1965).



- Gleichmäßige Schritte in Hauptrichtung,
- nach Bedarf Schritte in Nebenrichtung.



- Oberen Punkt wählen, falls Abstand vom Mittelpunkt  $< 0$ .

$$\text{cntl} = (y + 1/2) - y_{\text{soll}}$$

$$= (y + 1/2) - (x - x_0) * y_{\text{dist}} / x_{\text{dist}} - y_0$$

$$\text{cntl} * x_{\text{dist}} = (y - y_0) * x_{\text{dist}} + (x_0 - x) * y_{\text{dist}} + x_{\text{dist}} / 2$$

- Fortlaufendes Aufsummieren eliminiert die Multiplikationen:

$$\text{pro } x\text{-Schritt: } \text{cntl} := \text{cntl} - y_{\text{dist}};$$

$$\text{pro } y\text{-Schritt: } \text{cntl} := \text{cntl} + x_{\text{dist}};$$

$$\text{Initialisierung: } \text{cntl} := x_{\text{dist}} \text{ DIV } 2$$

- Der Fehler ist immer  $\leq 0,5 * \text{Rastermaß}$ .

- Rahmenprogramm für Grafikbeispiele:

```
PROCEDURE drawThings (input, output);  
CONST scale= 30;  
VAR a, b, globx, globy: LongInt;  
  
    PROCEDURE setpixel;  
    BEGIN moveto(globx, globy);  
            lineto(globx, globy)  
    END;  
  
    PROCEDURE Bresenham ...  
  
    PROCEDURE Ellipse ...  
  
BEGIN (* main *)  
    globx:= ...; globy:= ...;  
    a := 4; b:=?;  
    FOR b:=1 to 7 DO BEGIN  
        ellipse(globx, globy, a*scale, b*scale);  
        globx := 0;  
        bresenham(globx, globy, a*scale, b*scale);  
    END  
END (* drawThings *);
```

```

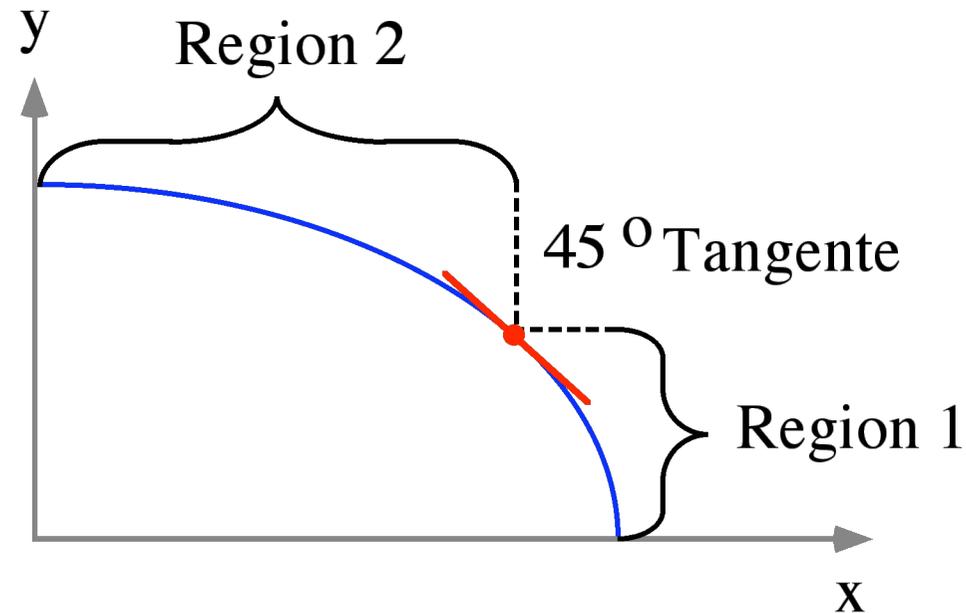
PROCEDURE Bresenham(VAR x, y: LongInt;
                    xdist, ydist : LongInt);
VAR xsign, ysign, cntl, i: LongInt;

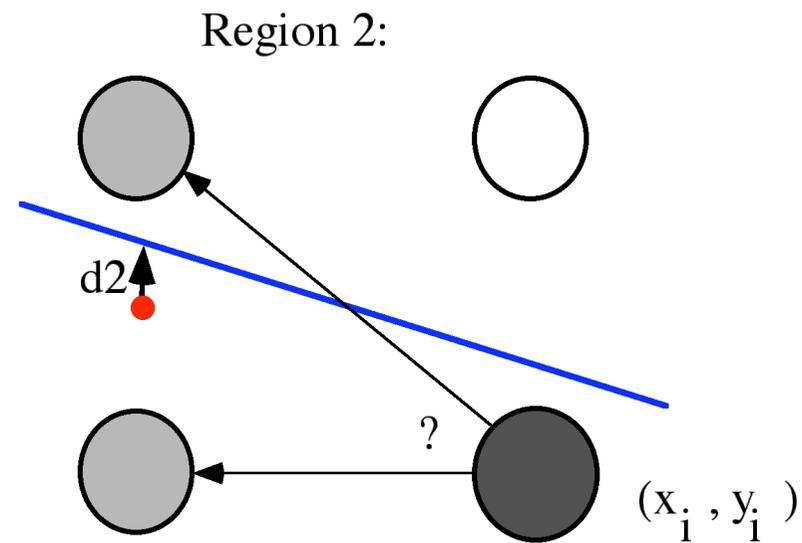
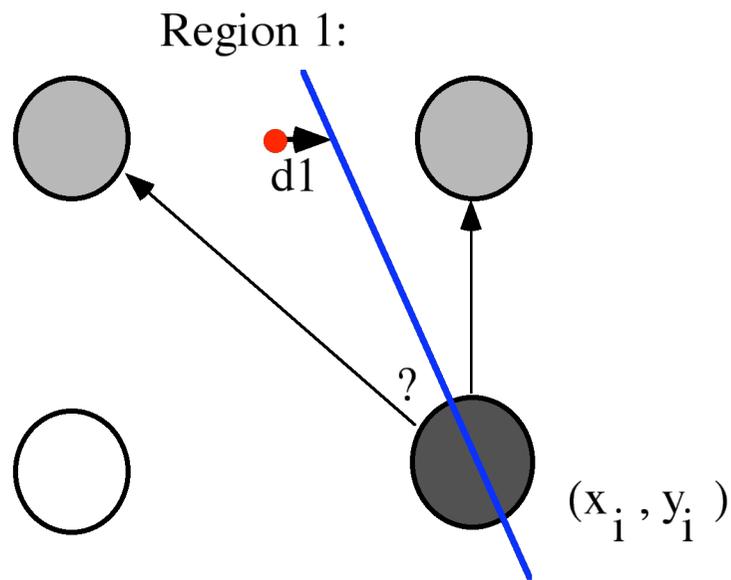
BEGIN
  IF xdist > 0 THEN xsign:= 1
  ELSE BEGIN
    xsign := -1; xdist := -xdist
  END;
  IF ydist > 0 THEN ysign:= 1
  ELSE BEGIN
    ysign := -1; ydist := -ydist
  END;
  (* Ende der Initialisierung. *)
  IF xdist >= ydist THEN BEGIN (*flach*)
    Setpixel;
    cntl := xdist div 2;
    FOR i := 1 to xdist DO BEGIN
      x := x + xsign;
      cntl := cntl - ydist;
      IF cntl < 0 THEN BEGIN
        y := y + ysign;
        cntl := cntl + xdist
      END;
      Setpixel
    END END (* flach, FOR xdist *)
    ELSE Bresenham(y, x, ydist*ysign, xdist * xsign)
  END (* Bresenham *);

```

## 1.2.2 Mittelpunktalgorithmus für Kreise und Ellipsen

- nach J. Van Aken, (IEEE C. Graph. & Appl., Sept. 84, pp 24-35.)
- Unterscheiden zwischen zwei Regionen:
  - Dazwischen 45 Grad Tangentenpunkt.
  - Umschalten der Strategie bei 45 Grad.





- Koordinaten der Mittelpunkte:

$$d1 : (x_i - 1/2, y_i + 1)$$

$$d2 : (x_i - 1, y_i + 1/2)$$

- Abstände  $d1$ ,  $d2$  von den Mittelpunkten als Kriterium für Folgepunkte
- Einsetzen in Ellipsengleichung:

$$0 = b^2 x^2 + a^2 y^2 - a^2 b^2$$

$$b^2(x_i^2 - x_i + 1/4) + a^2(y_i^2 + 2y_i + 1) - a^2 b^2 = d1 = cnt1 / 4$$

$$b^2(x_i^2 - 2x_i + 1) + a^2(y_i^2 + y_i + 1/4) - a^2 b^2 = d2 = cnt2 / 4$$

```

PROCEDURE ellipse(VAR x,y: LongInt; a,b: LongInt);
VAR cnt11, cnt12: LongInt;
BEGIN x:= a; y:= 0;
    REPEAT setpixel;
        cnt11:=    b*b * (4 * x*x - 4 * x + 1)
                + a*a * (4 * y*y + 8 * y + 4)
                - 4 * a*a * b*b;
        cnt12:=    b*b * (4 * x*x - 8 * x + 4)
                + a*a * (4 * y*y + 4 * y + 1)
                - 4 * a*a * b*b;
        IF cnt11 <0 THEN y:= y + 1
        ELSE IF cnt12 <0 THEN BEGIN
            y := y + 1; x := x-1
        END
        ELSE x := x-1;
    UNTIL x <0
END (* Ellipse *);

```

- Verfeinerungen:
  - cnt11 nur berechnen solange in Region 1
  - Multiplikationen durch sukzessive Additionen ersetzen
- Fehler kleiner als  $0.5 * \text{Rastermaß}$
- Vorsicht bei Zahlenbereichsüberschreitungen
- Verallgemeinern für Hyperbeln, Parabeln etc.

## 2. Multimedia-Hardware

### 2.1. Prozessorerweiterungen

#### 2.1.1. Beispielproblem

- YUV-Konvertierung
- UpSampling: 4:1:1 -> 4:4:4
- Konvertierungsmatrix YIQ -> RGB

$$\frac{1}{4096} \begin{pmatrix} 4788 & 0 & 6563 \\ 4788 & -1611 & -3343 \\ 4788 & 8295 & 0 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- CIF:  $288*352*25*30 = 76.032.000$  Instruktionen/sec
  - 7 (5) Multiplikationen, 4 Additionen
  - 8 Laden, 3 Speichern, 3 Shifts
  - pro Pixel 23 Instruktionen plus Adressmanipulation  $\sim 30$  Inst.
- Tabelle  $8 * 3$  MB groß

```
display^[i] := rgb[BSL(y,16)+BSL(u,8)+v];
```

- Programmbeispiel

```

FOR i := 0 TO LineLength-1 DO BEGIN (* 2 *)
  tmpy := 4788 * y^[i]; (* 3 *)
  r:=BSR((tmpy + 6563 * v^[vi]), 12); (* 4 *)
  IF r < 0 THEN r := 0 ELSE IF r > 255 THEN r := 255; (* 6 *)
    (* Sättigungs-Clipping *)

  g:=BSR((tmpy -1611*u^[ui] -3343*v^[vi]),12); (* 7 *)
  IF g < 0 THEN g := 0 ELSE IF g > 255 THEN g := 255; (* 6 *)

  b:=BSR((tmpy +8295*u^[ui]),12); (* 4 *)
  IF b < 0 THEN b := 0 ELSE IF b > 255 THEN b := 255; (* 6 *)

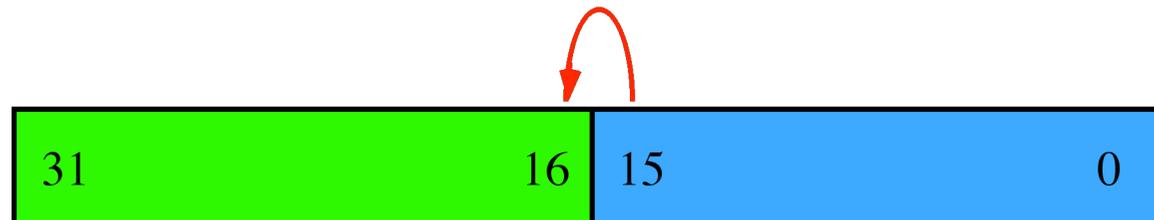
  display^[i] := BSL(r, 16) + BSL(g, 8) + b; (* 5 *)
  IF i mod 4 = 3 THEN BEGIN ui:=ui+1; vi:=vi+1 END; (* 5 *)
END;

```

- 48 Instruktionen

- Zahlen nur 16 bit groß

- 2 Pixel parallel addieren/multiplizieren <-> Carry



- Mittelwertbildung
- Farbtabelleberechnung mit Octree

```
theClut[i] := Average(theClut[i], theClut[j]);
```

- Elementweise Addition und Division / 2

```
theClut[i].r := (theClut[i].r + theClut[j].r) SHR 1; (*5*)
```

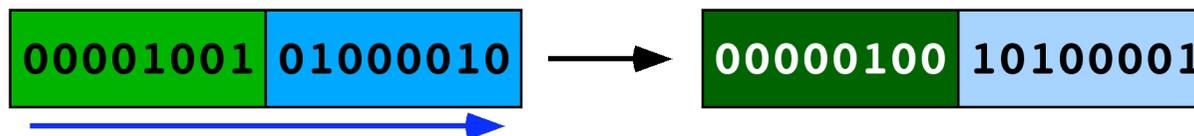
```
theClut[i].g := (theClut[i].g + theClut[j].g) SHR 1; (*5*)
```

```
theClut[i].b := (theClut[i].b + theClut[j].b) SHR 1; (*5*)
```

- Problem bei Parallelausführung

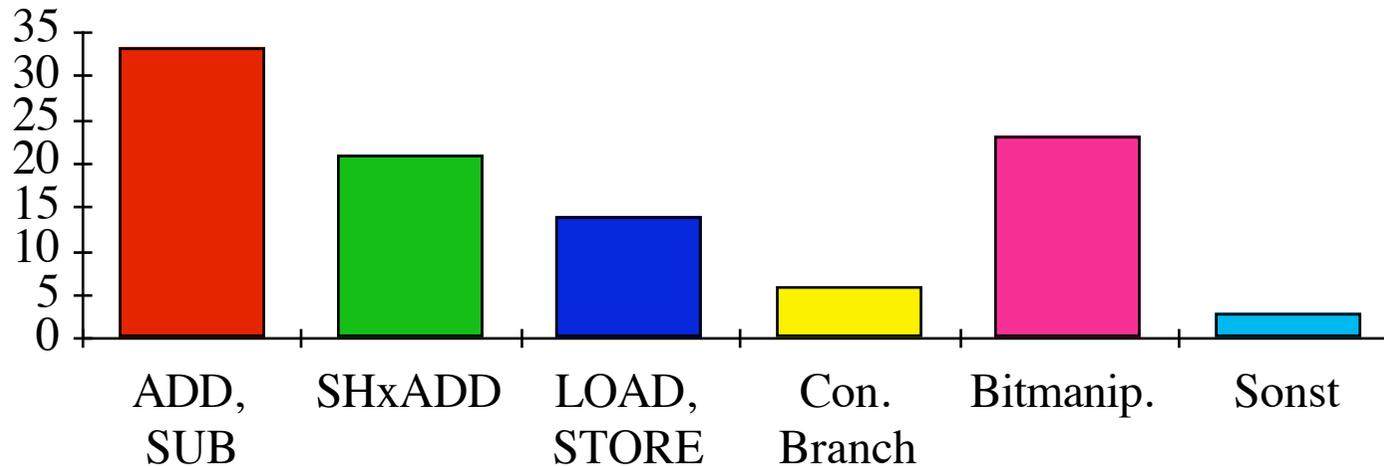
- ungerade + gerade => ungerade

- ungerade SHR 1 => niedrigerer Wert + 128 (bzw. 32768)



## 2.1.2. Spezialinstruktionen

- Codeanalyse IDCT



- SIMD-Instruktionen

- Werte im Speicher nebeneinander => 1 Bustransfer für 2/4 Werte
- kleine Zahlen, große Register (32, 64 bit)

$$\begin{array}{r} 00010010 \ 01001001 \\ + 01100010 \ 00100000 \\ \hline 01110100 \ 01101001 \end{array}$$

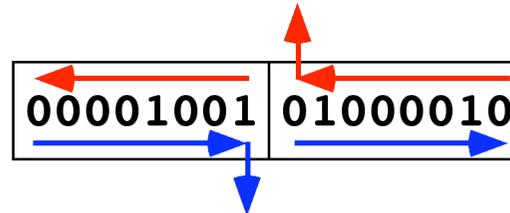
$$\begin{array}{r} 00010010 \ 11001001 \\ + 01100010 \ 01100000 \\ \hline 01110101 \ 00101001 \end{array}$$

- Konfigurierbare ALU

- Paralleles Addieren/Subtrahieren

$$\begin{array}{r} 00010010 \ 11001001 \\ + \underline{01100010 \ 01100000} \\ \hline 01110100 \ 00101001 \end{array}$$

- Carry Unterbrechung konfigurieren
- Paralleles Shiften



- Paralleles Multiplizieren  $16 * 16$

- Operation ShiftxAdd

- RISC ohne Integermultiplikation
- besonders für Multiplikation mit (kleinen) Konstanten

- Sättigungsarithmetik

$$\begin{array}{r}
 00010010 \ 11001001 \\
 + \ 01100010 \ 01100000 \\
 \hline
 01110100 \ 00101001
 \end{array}$$

$$\begin{array}{r}
 00010010 \ 11001001 \\
 + \ 01100010 \ 01100000 \\
 \hline
 01110100 \ 11111111
 \end{array}$$

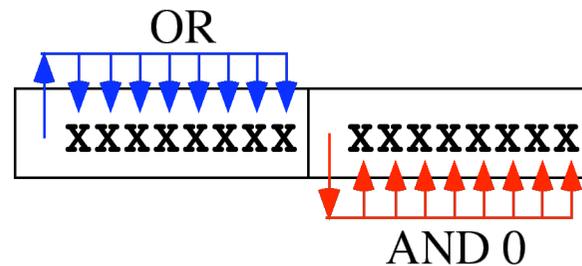
`erg := a + b;`

`IF erg < 0 THEN erg:= 0 ELSE IF erg > max THEN erg:= max;`

- besondere Carry/Borrow Behandlung

`IF borrow THEN regPart := 0;`

`IF carry THEN regPart := maxPartInt;`



- Auch für 2-seitigen Begrenzer (Clipping)

- YUV mit Parallel-Add/Mult und Sättigungsarithmetik

**YRGB** = BSL(**299**,16)+**299**; **VR** = BSL(**410**,16)+**410**; **UG,VG,UB** ...  
**i:=0; ui:=0; vi:=0;**

**REPEAT**

```

thisV := LoadHigh(v^[vi])+v^[vi];          (* 2 *)
thisU := LoadHigh(u^[ui])+u^[ui];          (* 2 *)

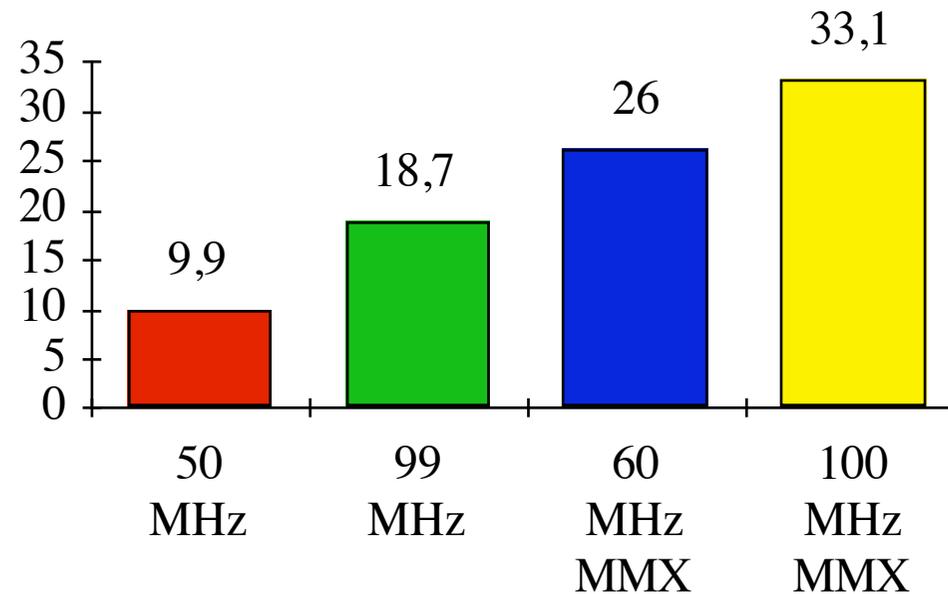
tmpy := YRGB * Load32(y^[i]);              (* 2 *)
r:=PBSR(tmpy + VR * thisV,8);              (* 3 *)
g:=PBSR(tmpy + UG * thisU + VG * thisV,8); (* 5 *)
b:=PBSR(tmpy + UB * thisU,8);              (* 3 *)

display^[i]:=          BSL(High(r),16)      (* 2 *)
          +          BSL(High(g),8) + High(b); (* 5 *)
i := i+1;              (* 1 *)
display^[i]:=          BSL(Low(r),16)      (* 2 *)
          +          BSL(Low(g),8) + Low(b)  (* 5 *)
i := i+1;              (* 1 *)
...                    (* 29 *)
ui:=ui+1; vi:=vi+1;  (* 2 *)
UNTIL i>=LineLength; (* 2 *)

```

- 66 Instruktionen für 4 Pixel: 16,5 Inst/Pixel

- HP PA-RISC 7100LC, 8000
  - HADD,ss ra, rb, rt
  - HADD,us ra, rb, rt
  - HSUB, HAVE (Mittelwert)
  - HSHLADD, HSHRADD als Teil der Multiplikation
- Bsp: MPEG Video Dekompression (352\*240 Pixel)

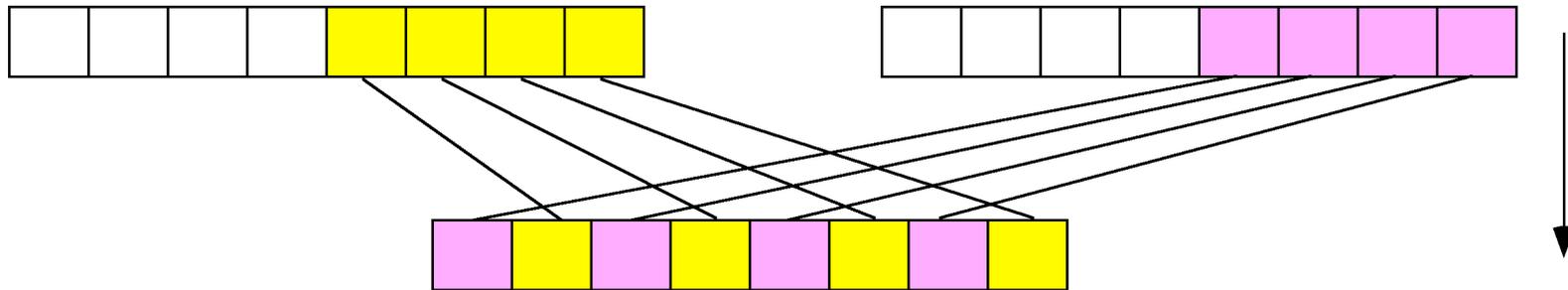


- Intel Pentium MMX
  - 8 64-bit-Register, mm0 .. mm7
  - entweder die FP's nach Umschaltung oder zusätzlich
  - $64 = 2*32 = 4*16 = 8*8$
  - 57 'neue' Instruktionen
- Parallele Operationen
  - paddw dest, source ; add words
  - paddusb dest, source ; add bytes saturated
  - psrlw dest, count ; shift word right
- Parallele Multiplikation und Multiply-Add
  - pmulhw dest, source ; multiply word:  $16*16 \Rightarrow 32$   
; 4 high-words  $\rightarrow$  dest (oder low words)
  - 3 Prozessorzyklen, aber Pipeline
  - pmaddwd dest, source ; multiply and accumulate word
- Testen umständlich
  - pcmpeq[b,w,d] reg, reg/mm ; auch gt
  - Ergebnisse als 00/11 gepackt in Register dest

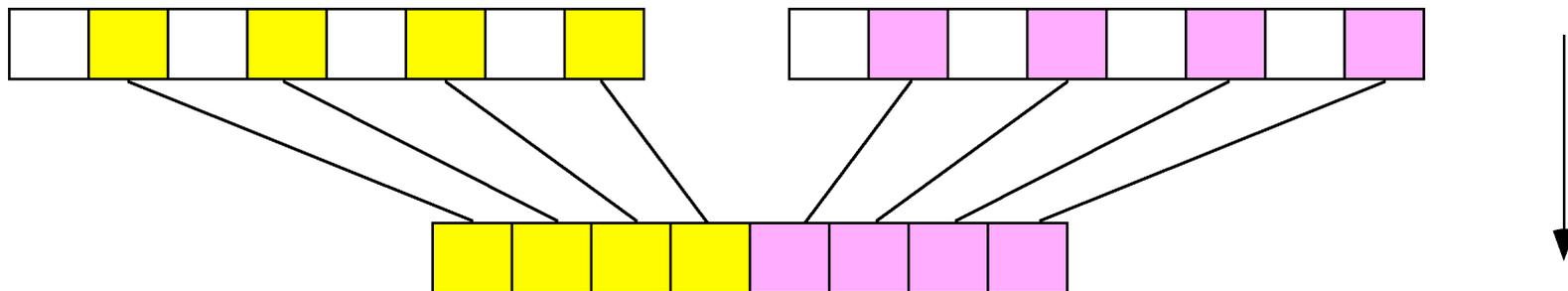
## • Laden und Packen

- movq dest, source ; 8 byte

- punpcklbw dest, source ; unpack low bytes to word



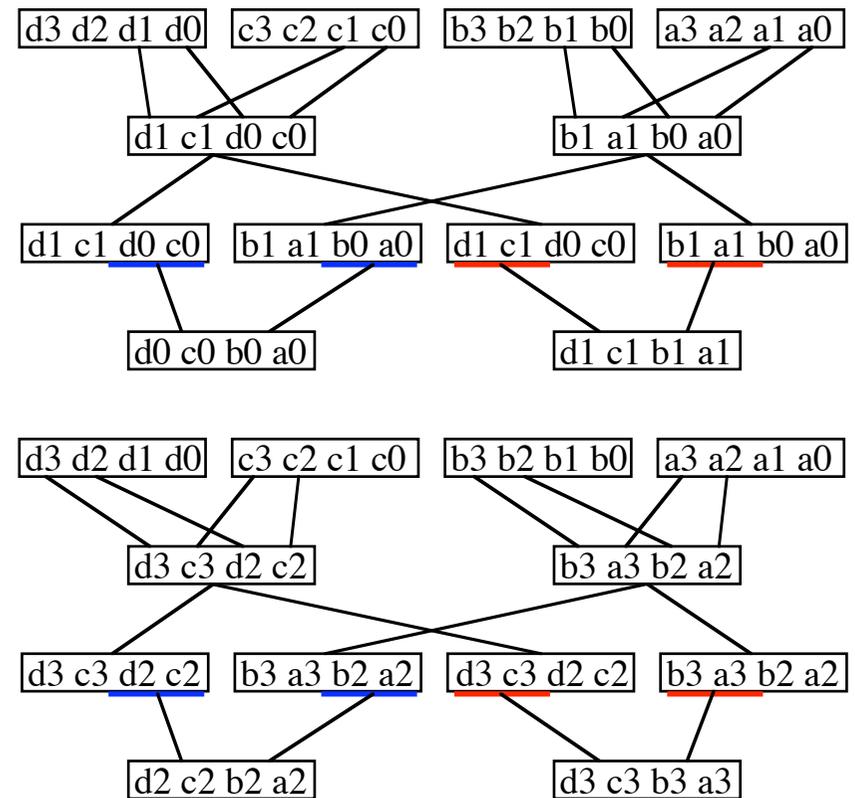
- packuswb dest, source ; dest:= d6 d4 d2 d0 s6 s4 s2 s0  
; unsigned saturation



- Matrix-Transposition: 4\*4

```

moveq      mm1, row1
moveq      mm2, row2
moveq      mm3, row3
moveq      mm4, row4
punpcklwd  mm1, mm2
punpcklwd  mm3, mm4
moveq      mm6, mm1
punpckldq  mm1, mm3   ;1. Zeile
punpckhdq  mm6, mm3   ;2. Zeile
moveq      mm5, mm1
moveq      mm1, row1
moveq      mm3, row3
punpckhwd  mm1, mm2
punpckhwd  mm3, mm4
moveq      mm7, mm1
punpckldq  mm1, mm3   ;3. Zeile
punpckhdq  mm7, mm3   ;4. Zeile
  
```



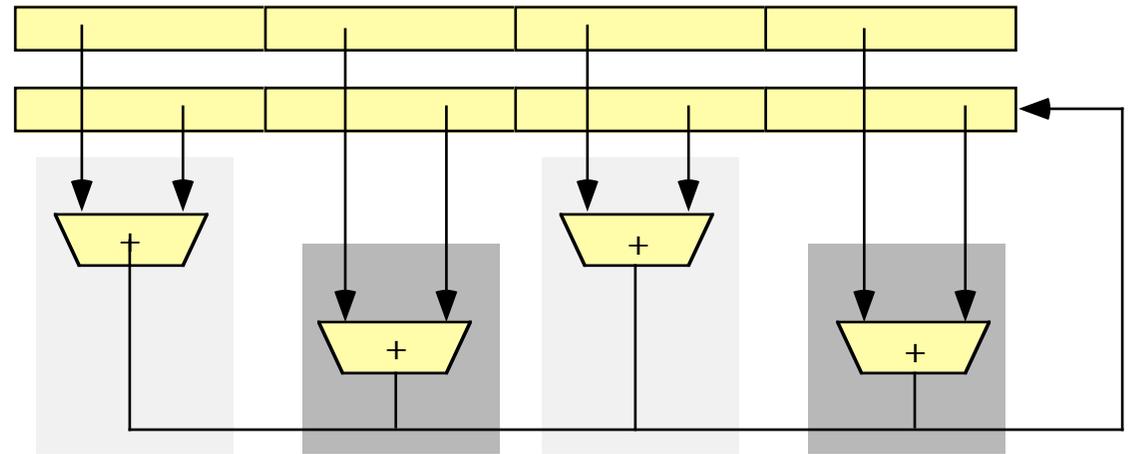
- Jim Blinn: I don't think compilers will automatically generate MMX code anytime soon, leaving a need for human assembly language crafters. This makes me happy.

## 2.1.3 SIMD in Mikroprozessoren

- Motorola AltiVec
  - G4-Kern, FPU
  - AltiVec-Erweiterung
  - 32 Vektor-Register je 128 Bit
  - Vector ALU: Integer und Floating Point SIMD:
    - 16\*8 Integer
    - 8\*16 Integer
    - 4\*32 Integer und Float
  - Vector Permute Unit: pack, unpack, load, store
- Intel MMX
  - Pentium-Erweiterung
  - Integer SIMD
  - Wiederverwendung von Pentium-Einheiten
  - 8 Vektor-Register je 64 Bit (= 8 FPU-Register)
  - Umschaltung FPU-MMX

- 3DNow! und SSE

- MMX und Floating Point SIMD (32 bit Float)
- 8 Vektor-Register je 128 Bit
- Mischverwendung möglich ...
- FP für grafische Algorithmen
- Ausführung von 2 \* 2-SIMD
- 4-MAC pipelined
- Produkt (4x4)\*(4) aufwendig



- Bsp.:  $(a*b)+(c*d)+(e*f)+(g*h)$  in 5 Zyklen

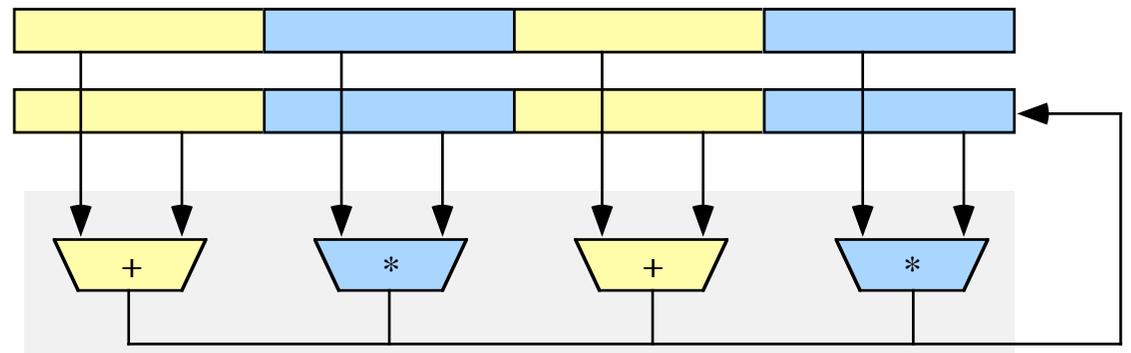
- Streaming SIMD Extensions

- Vektor-Reg  $\leftrightarrow$  FP-Reg
- neuer Prozessor-State

- SSE2

- double Precision Float

- SSE3, SSE4, SSE5

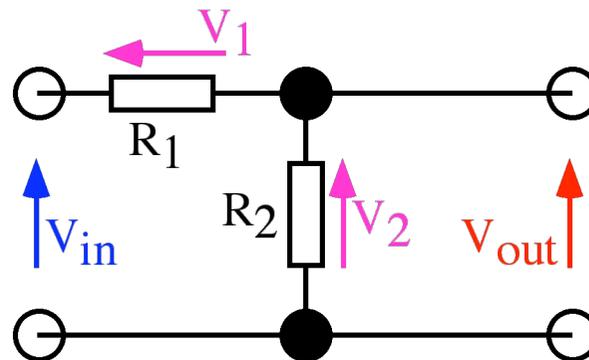


## 2.2. Signalprozessoren

### 2.2.1. Signalverarbeitungsalgorithmen

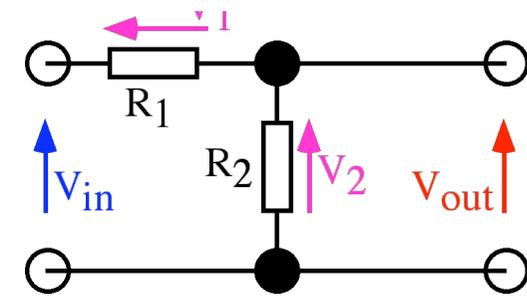
#### 2.2.1.1. Filter

- Anwendungen
  - Anti-Aliasing
  - MPEG-Audio Analyse
  - Bildverarbeitung
  - GSM ...
- Spannungsteiler
  - $U = I * R$

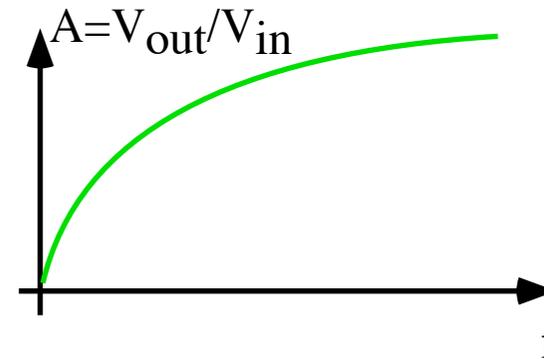
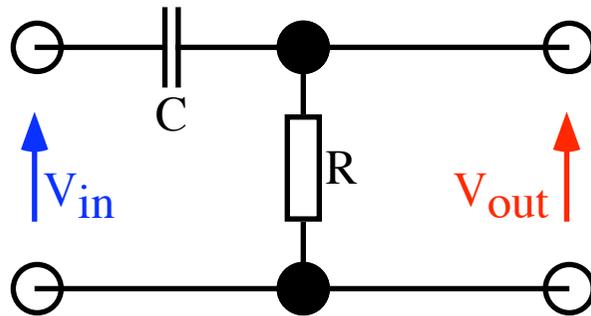


$$\frac{V_1}{V_2} = \frac{IR_1}{IR_2} = \frac{R_1}{R_2}, V_{in} = V_1 + V_2 \Rightarrow V_{out} = V_2 = \frac{V_{in}R_2}{R_1 + R_2}$$

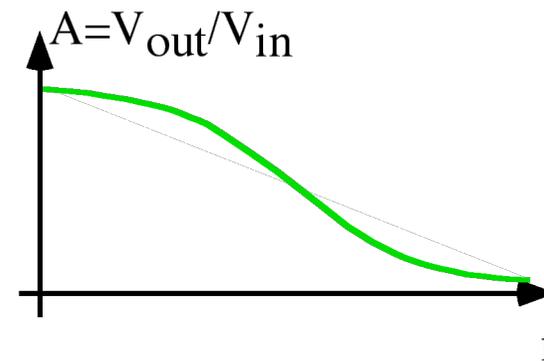
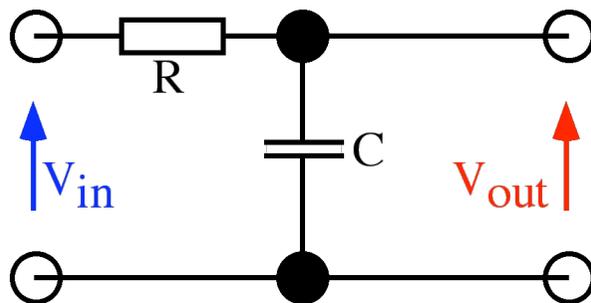
$$V_{out} = V_2 = \frac{V_{in} R_2}{R_1 + R_2}$$



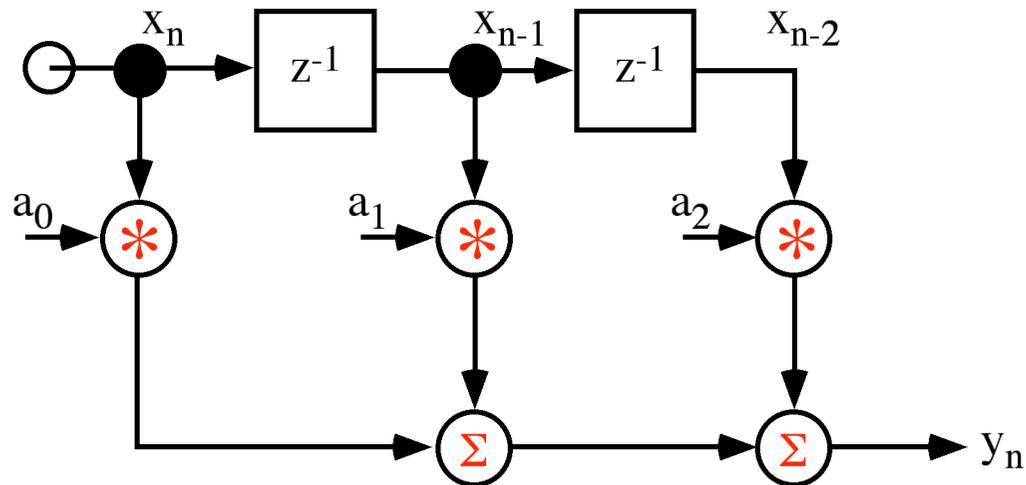
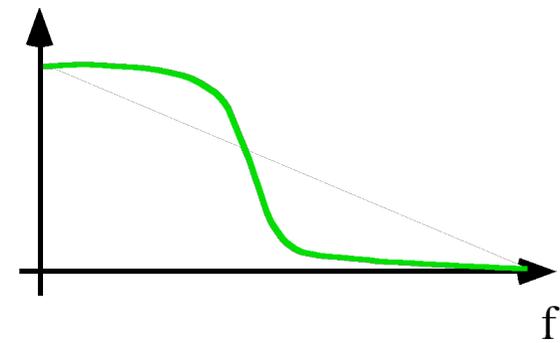
- Frequenzabhängiger Spannungsteiler  
- Hochpass



- Tiefpaß als RC-Filter



- Steile Flanke von Pass-Band zum Stop-Band
  - mehrstufige Filter
  - Tschebyschev
  - weitere Filtereigenschaften: Phasenantwort, ...
- Filter mit endlicher Impulsantwort: FIR-Filter



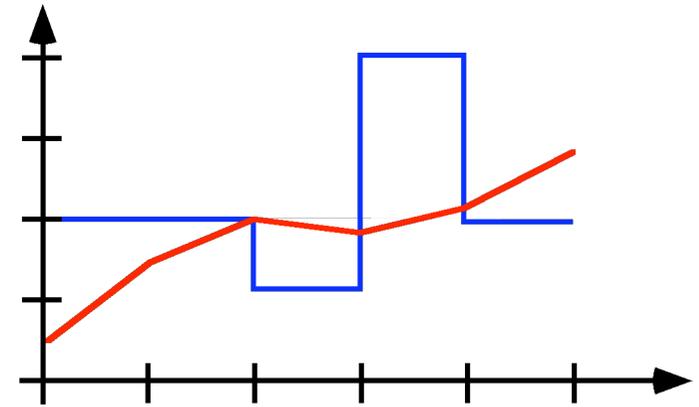
- $y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2}$
- Lineare Phasenantwort: Koeffizienten symmetrisch um Mitte

- Beispiel:

$$a_0 = 0,25; \quad a_1 = 0,5; \quad a_2 = 0,25$$

$$x_0 = 20; \quad x_1 = 20; \quad x_2 = 20;$$

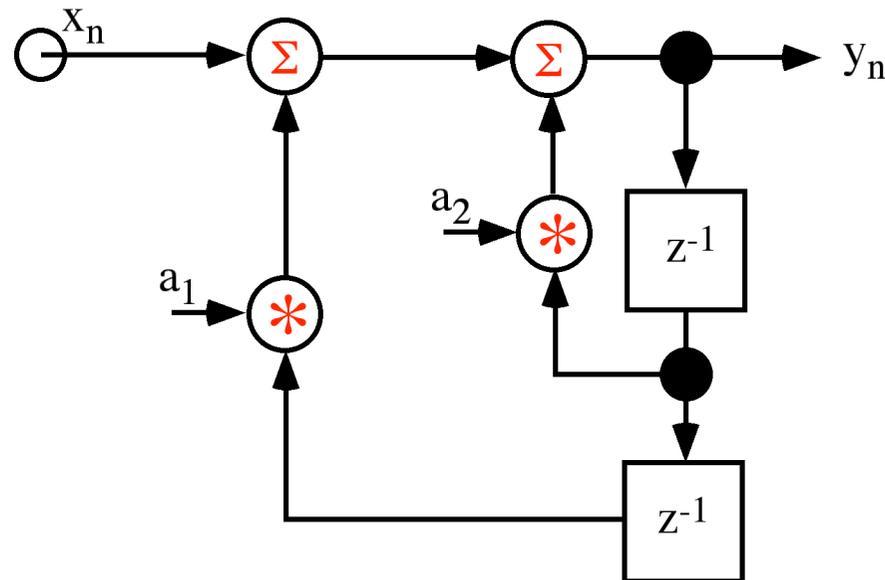
$$x_3 = 12; \quad x_4 = 40; \quad x_5 = 20$$



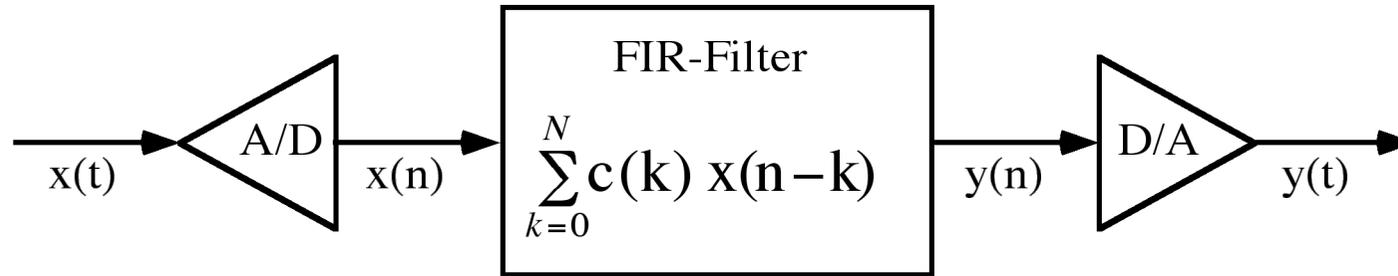
- Allgemeines FIR-Filter

$$- y(n) = \sum_{k=0}^{N-1} c(k)x(n-k)$$

- Unendliche Impulsantwort: IIR



- Digitales Filter



- Algorithmische Bausteine

```

repeat
    sum := 0;
    for i:=0 to n do begin
        hilf := x[i] * coeff[i];
        sum := sum + hilf;
    end;
    PORT(DAC) := sum;
    for i:=n-1 downto 0 do
        x[i+1] := x[i];
        x[0] := PORT(ADC);
until feierabend;

```

(\* 1 \*)  
 (\* 3\*(n+1) \*)  
 (\* n+1 \*)  
 (\* n+1 \*)  
 (\* 1 \*)  
 (\* 3\*n \*)  
 (\* n \*)  
 (\* 1 \*)  
 (\* 2 \*)

- $9n + 10$  Instruktionen

- Verbesserungen:

- Ringpuffer
- Multiply-Accumulate

## 2.2.1.2. Fouriertransformation

- Spektrale Analyse eines Signales

- Fourier, 1822:

- jedes periodische Signal kann dargestellt werden durch:

$$x(t) = \sum_{k=-n}^n C_k e^{i(\omega_k t)}$$

- Fourier-Reihe

- Diskrete Fourier-Reihe  $x(t) \rightarrow x(n)$

- Nicht-periodisches Signal

- Kontinuierliche Fourier Transformation

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi f t} dt$$

- Diskrete Fourier Transformation (DFT)

- endliche Berechnung  $\Rightarrow$  'Fenstern' (Konvolution mit Fensterfunktion)

- Endform:

$$X_N(k) = \sum_{n=0}^{N-1} x(n) e^{-i\left(\frac{2\pi kn}{N}\right)}$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_N(k) e^{i\left(\frac{2\pi kn}{N}\right)}$$

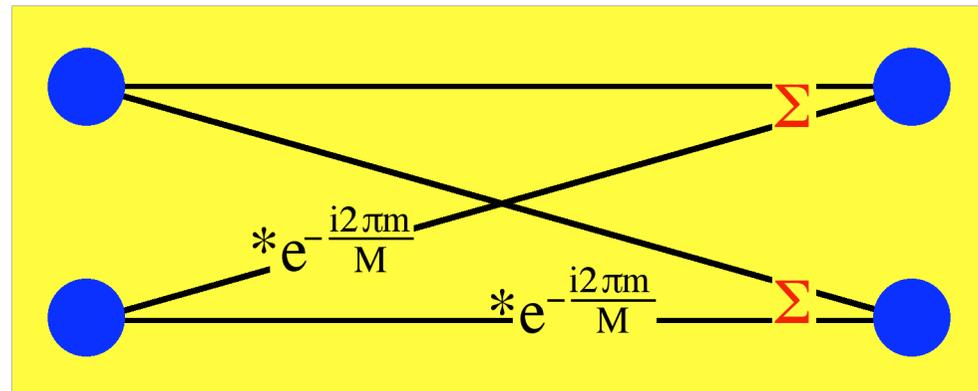
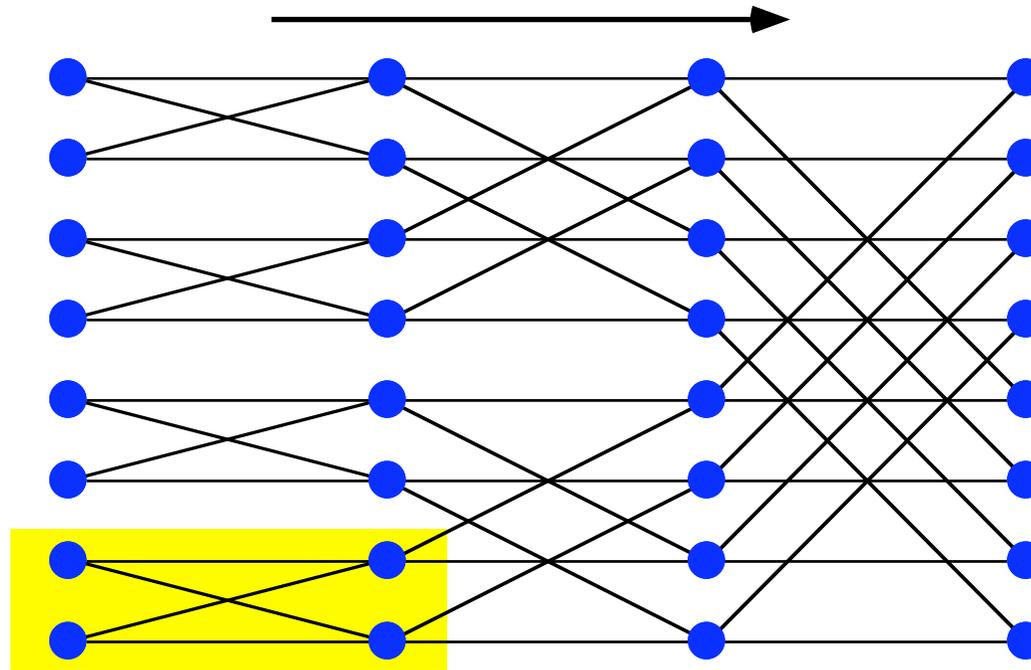
- DFT entspricht Multiplikation  $\text{Matrix}_{N,N}/\text{Vektor}_N$ 
  - extrem rechenintensiv:  $O(N^2)$
  - $N = 1000 \Rightarrow 1$  Million Instruktionen
  - 50 Mips DSP  $\Rightarrow$  Analyse eines 25 Hz Signales
- FFT nach J.W. Cooley, J.W Tuckey, 1965
  - Zerlegung einer N-DFT in Folge von  $\log_2(N)$  2-DFTs

$$\begin{aligned}
 X_N(k) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) e^{-j\left(\frac{2\pi k 2r}{N}\right)} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) e^{-j\left(\frac{2\pi k(2r+1)}{N}\right)} \\
 &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) e^{-j\left(\frac{2\pi k 2r}{N}\right)} + e^{-j\left(\frac{2\pi k}{N}\right)} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) e^{-j\left(\frac{2\pi k 2r}{N}\right)}
 \end{aligned}$$

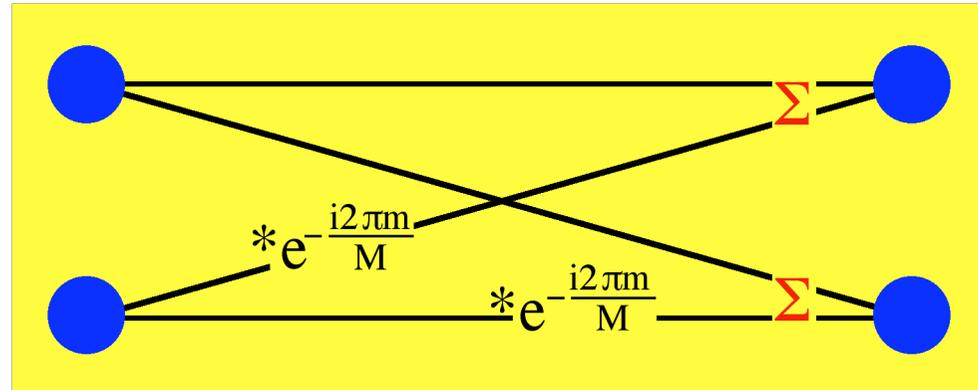
- N Zweierpotenz

- Rekursion
- $O(N \log_2 N)$

- Berechnungsfluß



- Butterfly



Multiplikation mit  $\exp(\dots)$

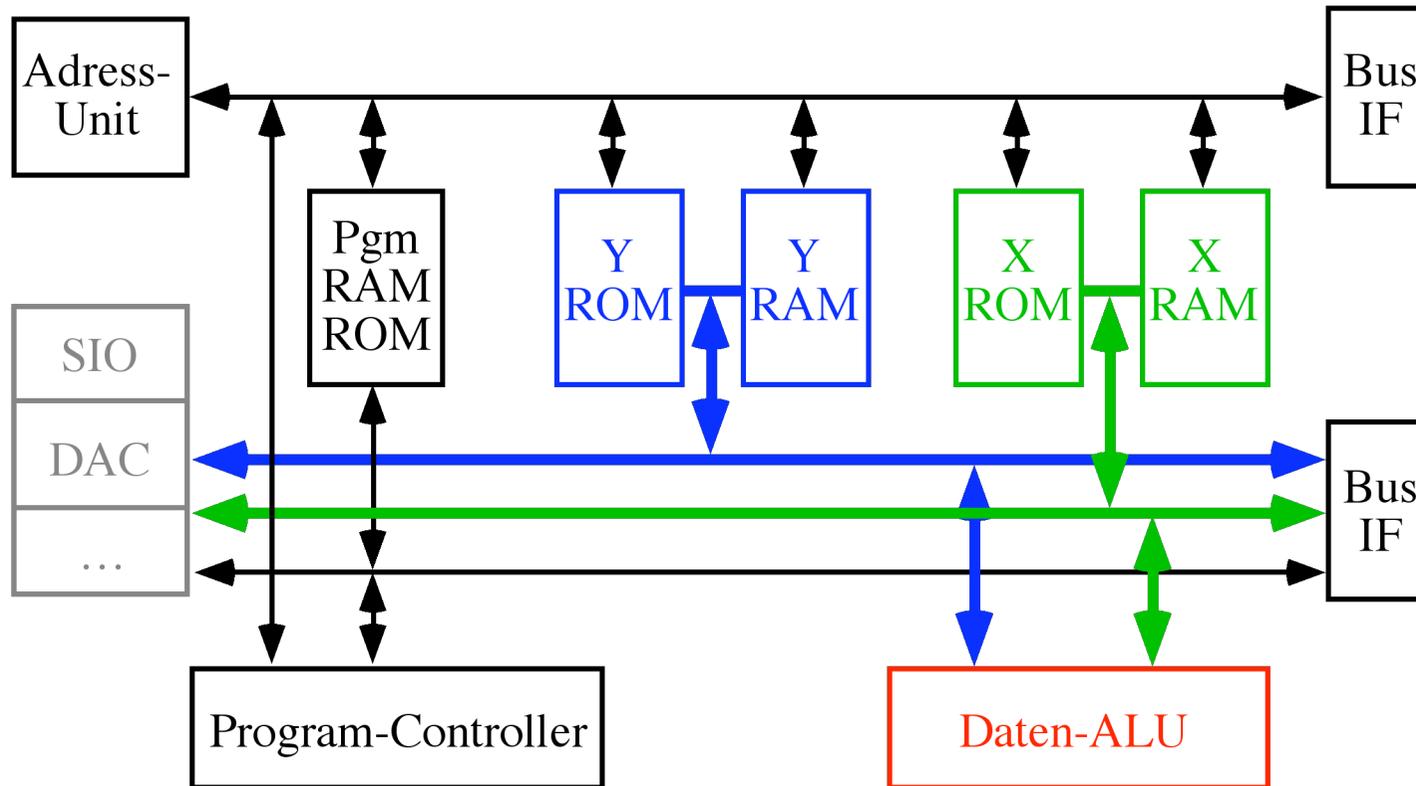
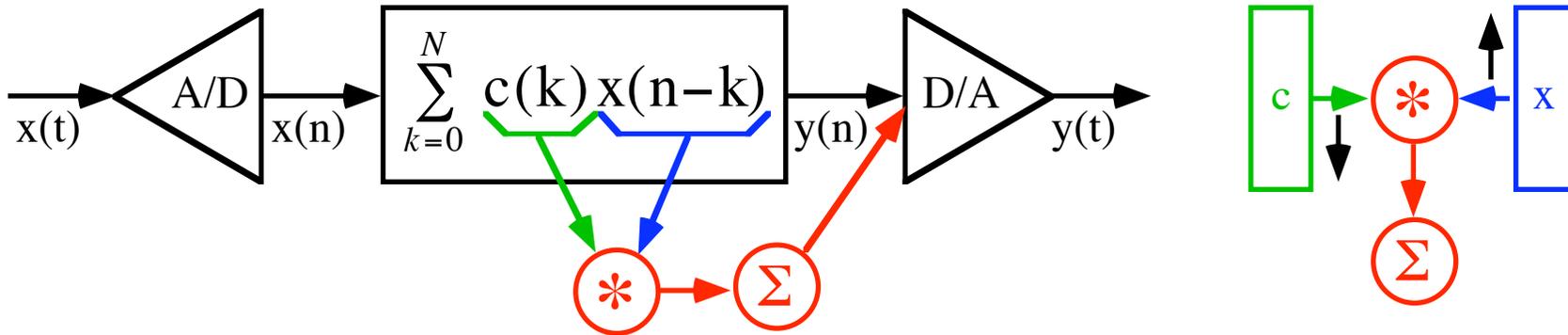
Vor Beginn umsortieren:

000	001	010	011	100	101	110	111
$x(0)$	$x(1)$	$x(2)$	$x(3)$	$x(4)$	$x(5)$	$x(6)$	$x(7)$
$x(0)$	$x(4)$	$x(2)$	$x(6)$	$x(1)$	$x(5)$	$x(3)$	$x(7)$
000	100	010	110	001	101	011	111

=> Bitreversal-Adressierung

- Goertzel-Algorithmus für DTMF-Dekodierung
- Chen-Transform für DCT

## 2.2.2 DSP-Architektur



- Processing Units
  - Control, Instruction Decode, ...
  - Adressverarbeitung
  - Multiplikation
  - evtl. reduzierte FPU
  - keine MMU
  - zusätzliche Funktionen auf dem Chip:  
ADC, DAC, SIO, Timer, ...
- Speicher im DSP
  - on Chip RAM, 512 bis 4096 Worte
  - Wort 16/24 Bit Integer oder 24/32 Bit Float
  - typisch Aufteilung in Bänke
  - ROM für besonders häufig gebrauchte Konstanten (sin, cos, e, ...)

- Instruktionen (AT&T 3210)

- Multiply-Accumulate-Store

```
*r11++r17 = a0 = a0 + *r3++ x *r4--
```

- reduzierte Bitverarbeitung

- Spezialbefehle: PCM <-> Linear

- Bedingte Befehle:

```
erg := ifcond (reg)
```

```
TwoSideLimiter:    a2 = -*r1           ; a2 := - limit
                    a1 = -a0 +*r1       ; if a0 > limit
                    a0 = ifalt(*r1)     ; then a0:=limit
                    a1 = a0 + *r1       ; if a0 < limit
                    a0 = ifalt(a2)      ; then a0:= -limit
```

- Schleifen:

```
DO instCount, times
DO instCount, reg
```

- Adressierung

- weniger allgemeine Adressierungsmodi als CISC
- Modulo-Adressierung (=> Ringpuffer)

```
if (index++ > base+blockSize) index = base;
```

- Vorkehrungen für Bit-Reversal:

000	001	010	011	100	101	110	111
↓	↓	↓	↓	↓	↓	↓	↓
000	100	010	110	001	101	011	111

- Durchlauf des Vektors: Bits Spiegeln, 1 Addieren, Bits Spiegeln
- Address-Mode Reverse-Carry-Add:

$2^{M-1}$  addieren, dabei Carry von links nach rechts bewegen

Bsp 8-FFT:  $N = 2^3 = 2^M$

$$\begin{array}{r}
 000 \\
 + 100 = 100 \\
 \quad + 100 = 010 \\
 \quad \quad + 100 = 110 \\
 \quad \quad \quad + 100 = 001 \\
 \quad \quad \quad \quad + 100 = 101 \\
 \quad \quad \quad \quad \quad + 100 = 011 \\
 \quad \quad \quad \quad \quad \quad + 100 = 111
 \end{array}$$

- Sortieren des Eingabe-Vektors bei der FFT (3210)

```

// r18 = M/2
// r16 = M/2
// r1, r2 begin of input vector
// r3  "swap-space"

```

```

Loop:      r1 - r2                                // exchange?
           if(ge) pcgoto Bitrev
           nop
           *r3++ = a0 = *r1++                    // yes!
           *r3-- = a0 = *r1--
           *r1++ = a0 = *r2++                    // real
           *r1-- = a0 = *r2--                    // imaginär
           *r2++ = a0 = *r3++
           *r2-- = a0 = *r3--

Bitrev:    r2 = r2 # r16
           r1 = r1 + 8
           r18 = r18 - 1
           if(ge) pcgoto Loop
           nop

```

- Optimierungen

- Ausnutzen der Pipeline-Latenz
- Jump-Slot

```
// r18 = M/2  
// r16 = M/2  
// r1, r2 begin of input vector
```

```
Loop :   r1 - r2  
        if(ge) pcgoto Bitrev  
        nop  
        *r1    = a0 = *r2  
        *r2++  = a0 = *r1++  
        *r1    = a0 = *r2  
        *r2--  = a0 = *r1--  
Bitrev:  r1 = r1 + 8  
        r18 = r18 - 1  
        if(ge) pcgoto Loop  
        r2 = r2 # r16
```

- Beispiel: FIR Filter

$$y(n) = \sum_{k=0}^{N-1} c(k)x(n-k)$$

```

repeat
    sum := 0;
    for i:=0 to n-1 do begin
        sum :=sum + coeff[i]*
            x[(curr+i) MOD(n-1)];
    end;
    curr:= (curr+1) AND (n-1)
    DAC:= sum;
    x[curr]:= ADC;
until feierabend;

```

(\* 1 \*)  
 (\* 3(n+1) \*)  
 (\* MAC \*)  
 (\* n+1 \*)  
 (\* 2; empty pipeline \*)  
 (\* 1 \*)  
 (\* 1 \*)  
 (\* 2 \*)

- 4n + 7 Instruktionen

- TMS 320 (TI 320C30)

- mit FPU

- \* **Init**

- \*

- LDI**    **N, BK**
        - LDI**    **Coeff, AR0**
        - LDI**    **X, AR1**

- \* Blocksize**

- \*

- \* **Get sample**

- \*

- Repeat**    **LDF**    **IN, R3**
        - STF**    **R3, \*AR1++%**
        - LDF**    **0, R0**
        - LDF**    **0, R2**

- \* **Filter**

- RPTS**    **N**
        - MPYF3** **\*AR0++%, \*AR1++%, R0**    ||
        - ADDF**    **R0, R2**

- \* RepCount:= N**
          - ADDF3** **R0, R2, R2**

- STF**    **R2, Y**
        - B**        **Repeat**

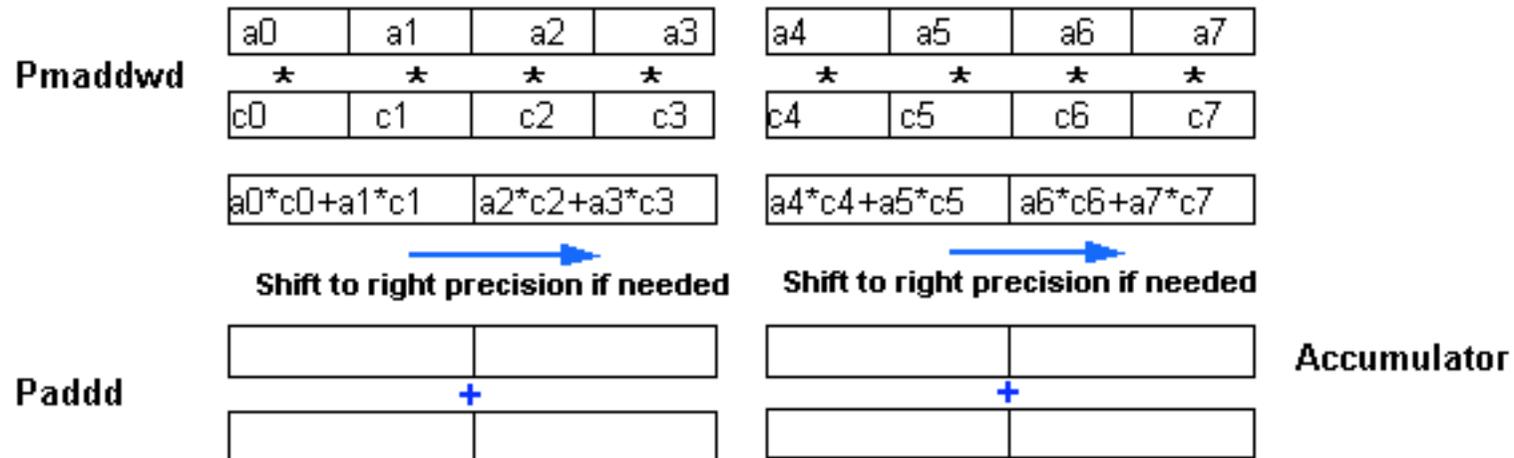
- 1\*FilterOrder + 8 Instruktionen

- 200 MHz DSP => 100 Mips => 100 MSamples/Filterlänge

- Filterlänge 256 => 200 kHz Signal filterbar

- oder mit MMX [Intel]

$$x = \sum a(i) * c(i)$$

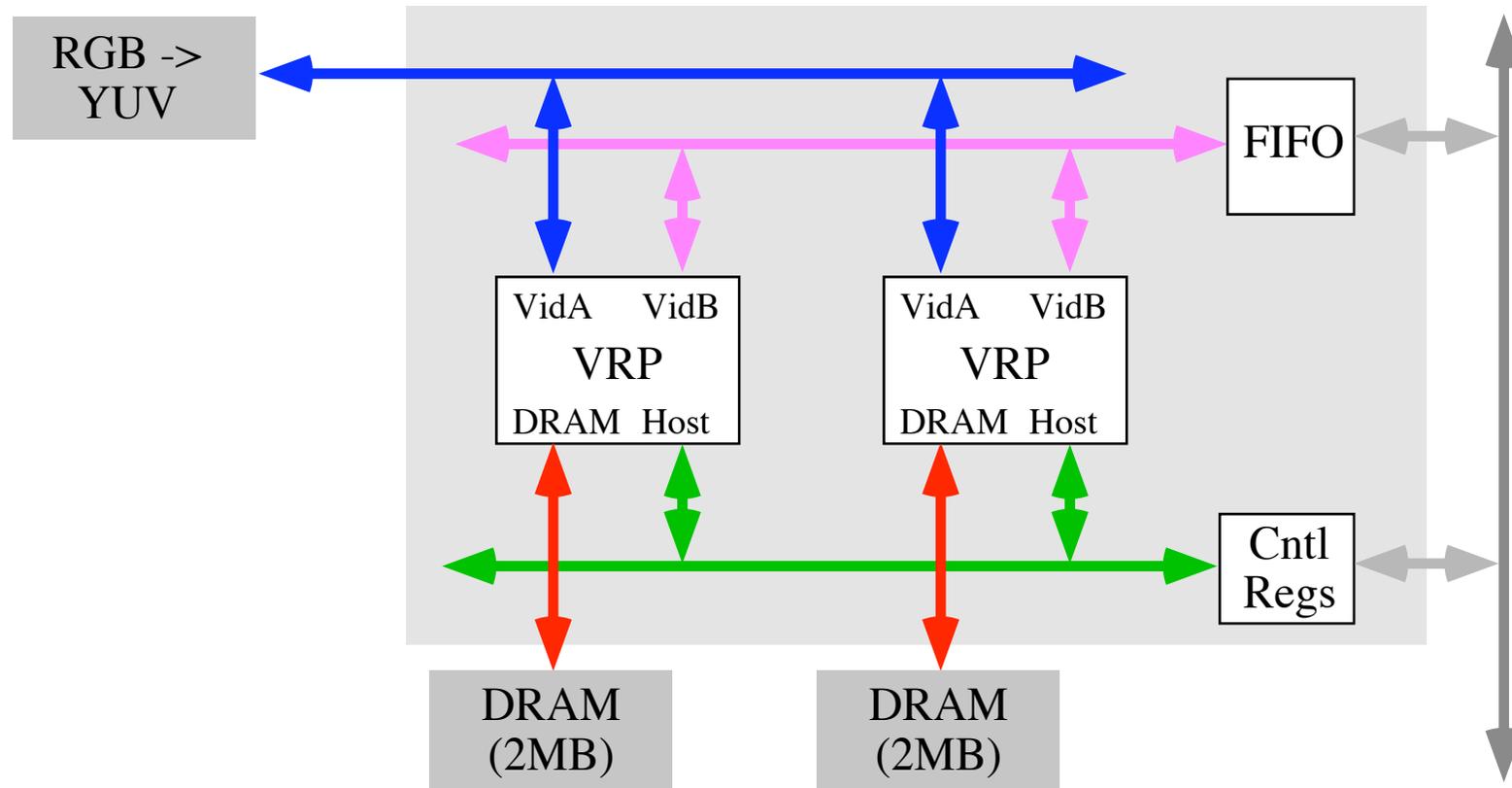


**Note:** Input data and coefficients are 16-bit precision.  
If not, first unpack to 16 bit.

- Filterdesign
  - Koeffizientenwahl
  - MatLab Skripte von den Herstellern
  - Simulationspakete (z.B. TI.com)

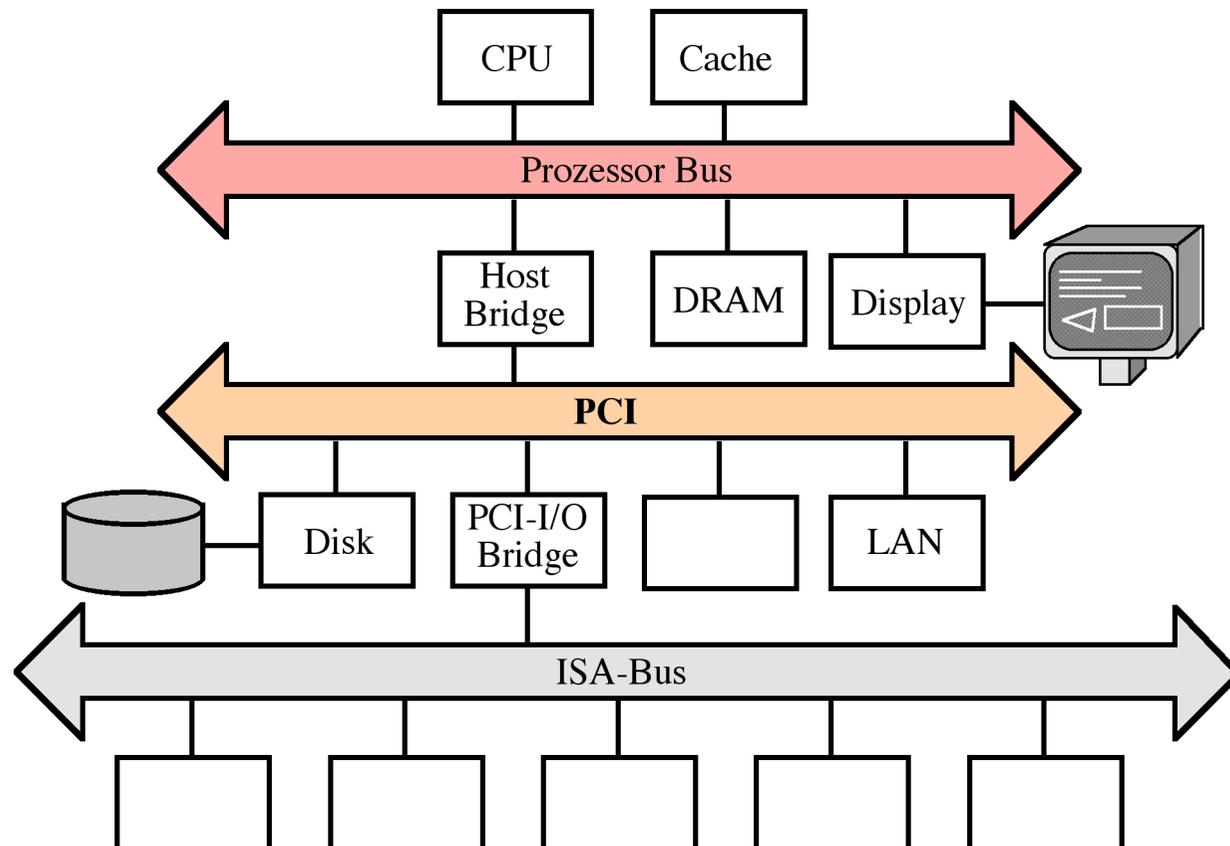


- Beispiel C-Cube CLM4550 "MPEG-1 Video Encoder"

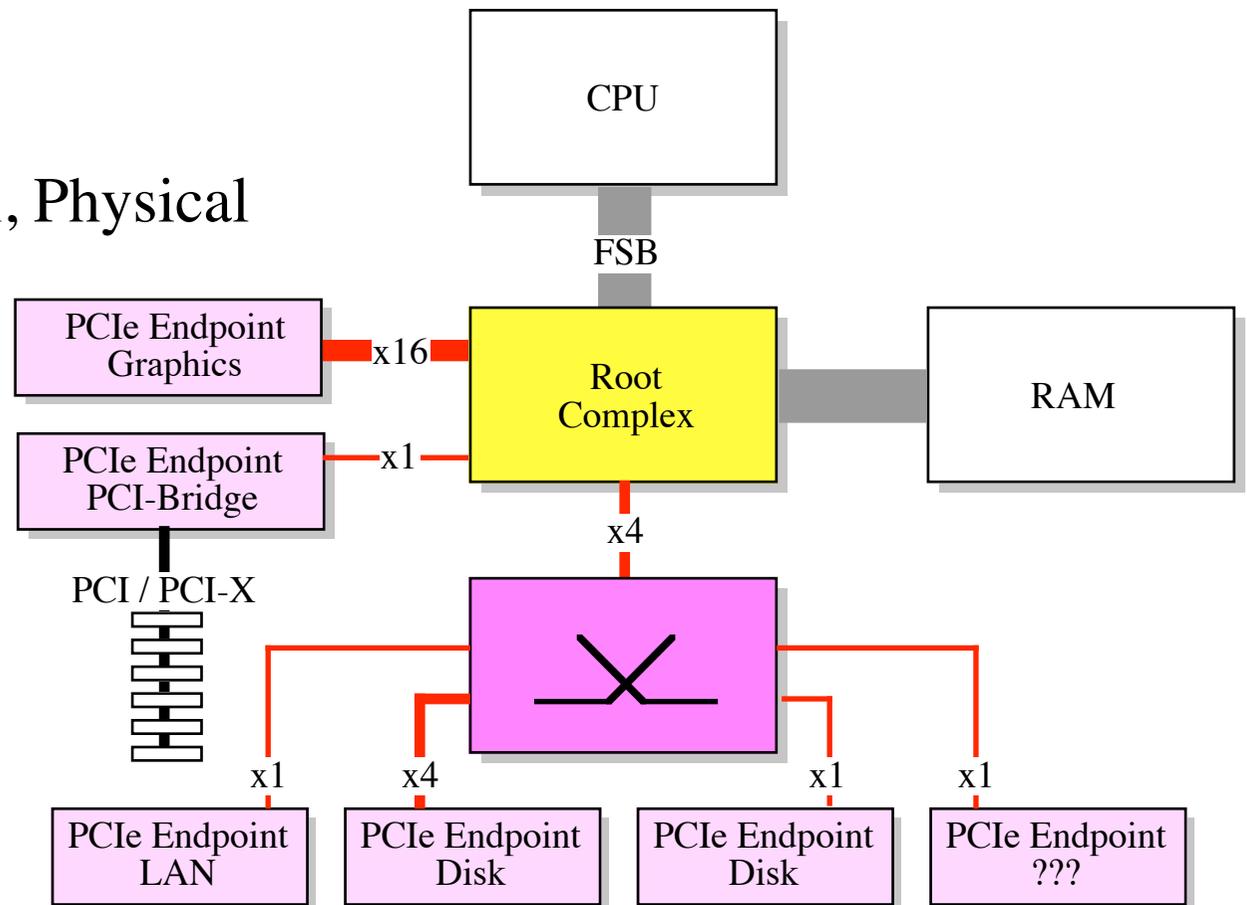


## 2.4 Hochgeschwindigkeitsbussysteme

- Peripheral Component Interconnect (PCI)
  - vier Slots
  - Adress/Datenmultiplex, 32 bit; 64 bit Erweiterung
  - vollständiger Bus mit Arbitrierung und Interrupts
  - 30 ns -> 66 MByte/s
  - Burst bis 133 MByte/s
  - CPU unabhängig: 80x86, Alpha, PowerPC

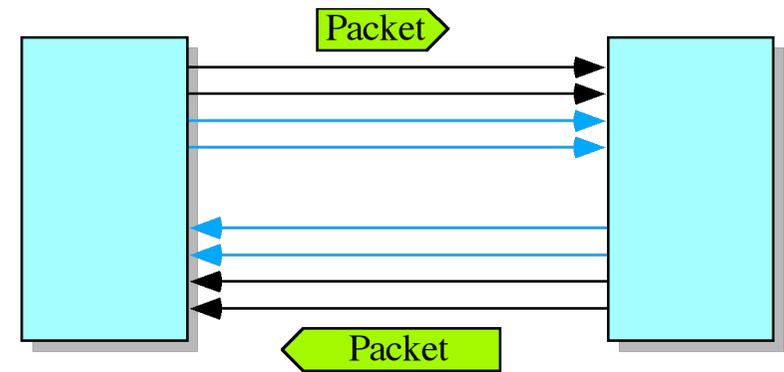


- PCI-Skalierung problematisch: Takt begrenzt
  - Skew vs. Leitungslänge und Leitungsführung
  - Adressauswertung/Chip Select auf den Karten
- PCI Express
  - serielle Links übertragen Bytes
  - Switch möglich
  - 1, 2 - 32 Links
  - Transaktionen in Paketen
  - Schichten: Transfer, Link, Physical
- PCIe Konfigurationen
  - PCs, Server
  - Netzwerk-Geräte
  - Mehrprozessorsysteme



- Links

- 2,5 Gbit/s pro Link
- 250 MByte/s für Daten+Overhead
- Taktgewinnung aus Signal
- 1, 2, 4, 8, 12, 16, 32 'Lanes'
- Bytestrom byteweise auf mehreren 'Lanes'



- Transaction Layer

- Flusskontrolle mit Krediten
- Virtuelle Kanäle: Multiplex zwischen Geräten
- Differentiated Services, Traffic Class Field
- Transaction Layer Packet (TLP): Header+Data+T-CRC

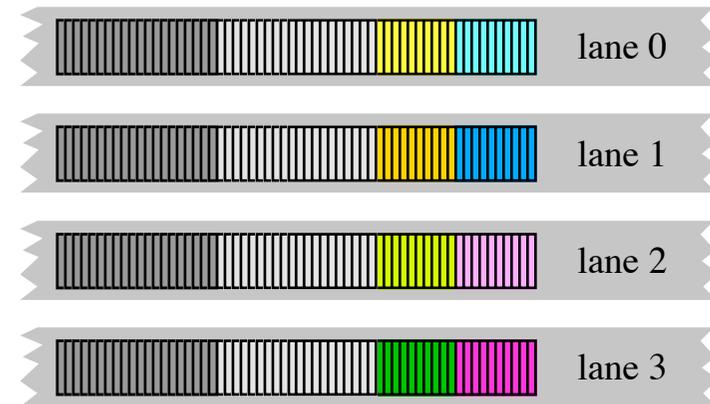
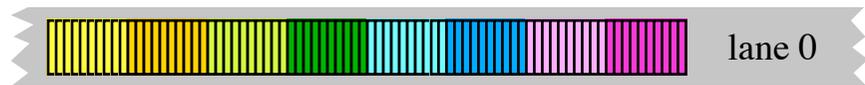


- Link Layer

- zuverlässige Übertragung
- Sequenznummern und L-CRC
- ACK, Retransmission
- Data Link Layer Packet (DLLP)

- Physical Layer

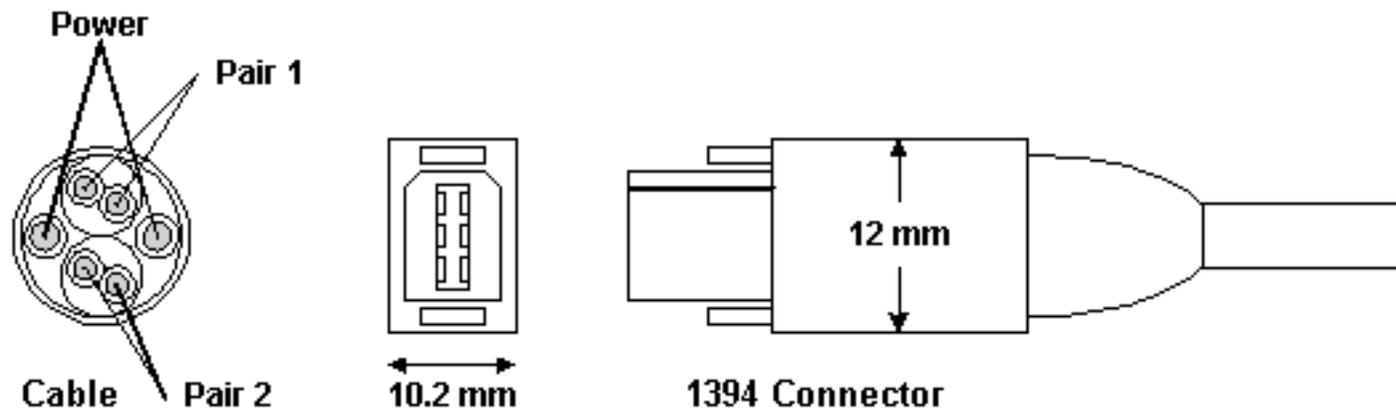
- differentielle Übertragung: RX- und TX-Paare
- Kodierung 8b/10b (= Gigabit Ethernet)
- Sondersymbole für Frame-Bildung
- 2.5 Gbit/s - 10 Gbit/s pro Link pro Richtung
- 'de-skew' in Trainingsphase
- byteweise Verteilung auf Lanes



- PCI Express x16 Graphics 150W-ATX

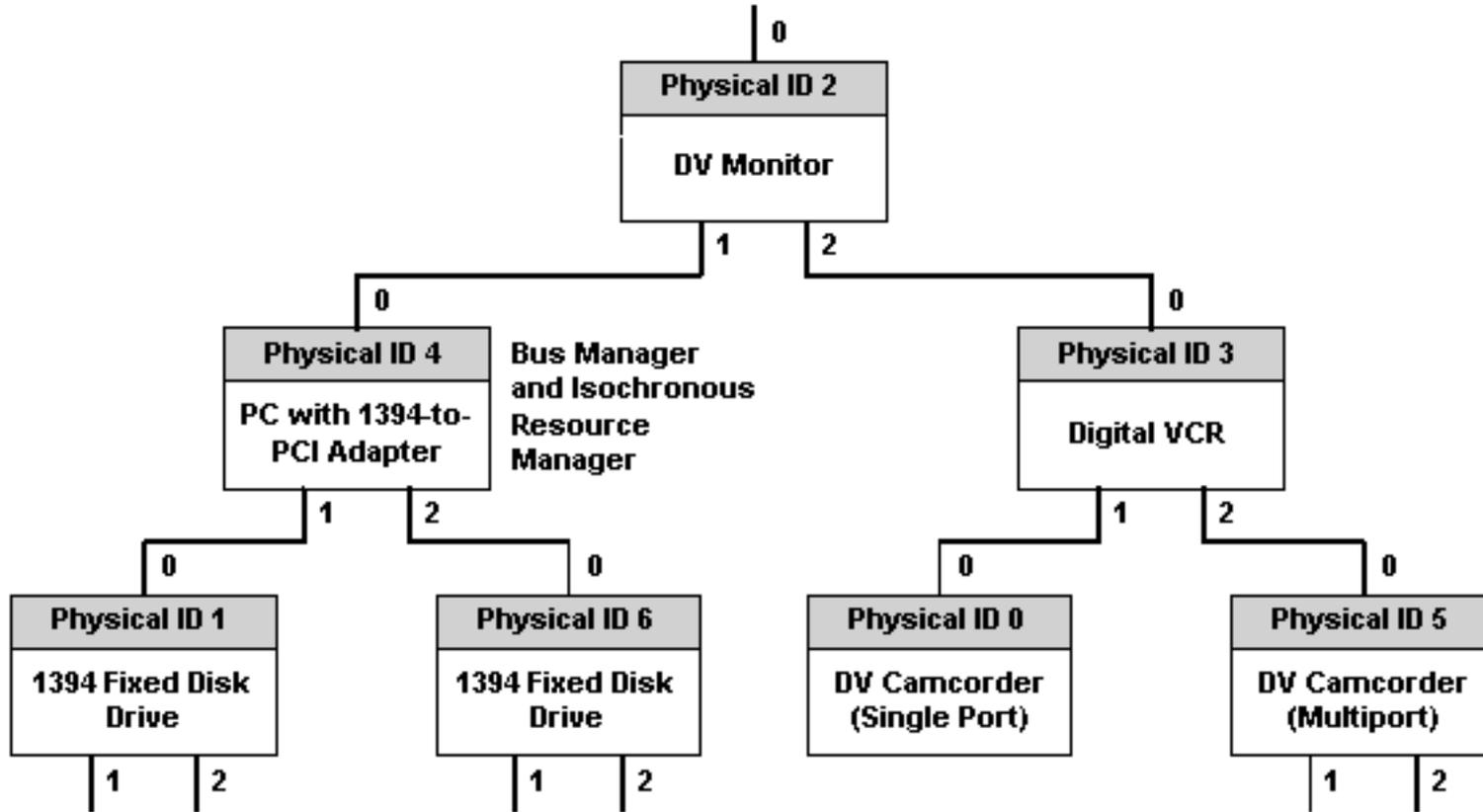
- 16 PCIe-Links
- **150W** Stromversorgung

- IEEE 1394: Firewire
  - desktop-LAN
  - Apple, Intel, TI, Adaptec, Sony (iLink), ...
  - auf der Hauptplatine oder kabelbasiert
- High speed serial bus IEEE-Working Group seit 1988
  - günstige Verkabelung, automatische Konfiguration

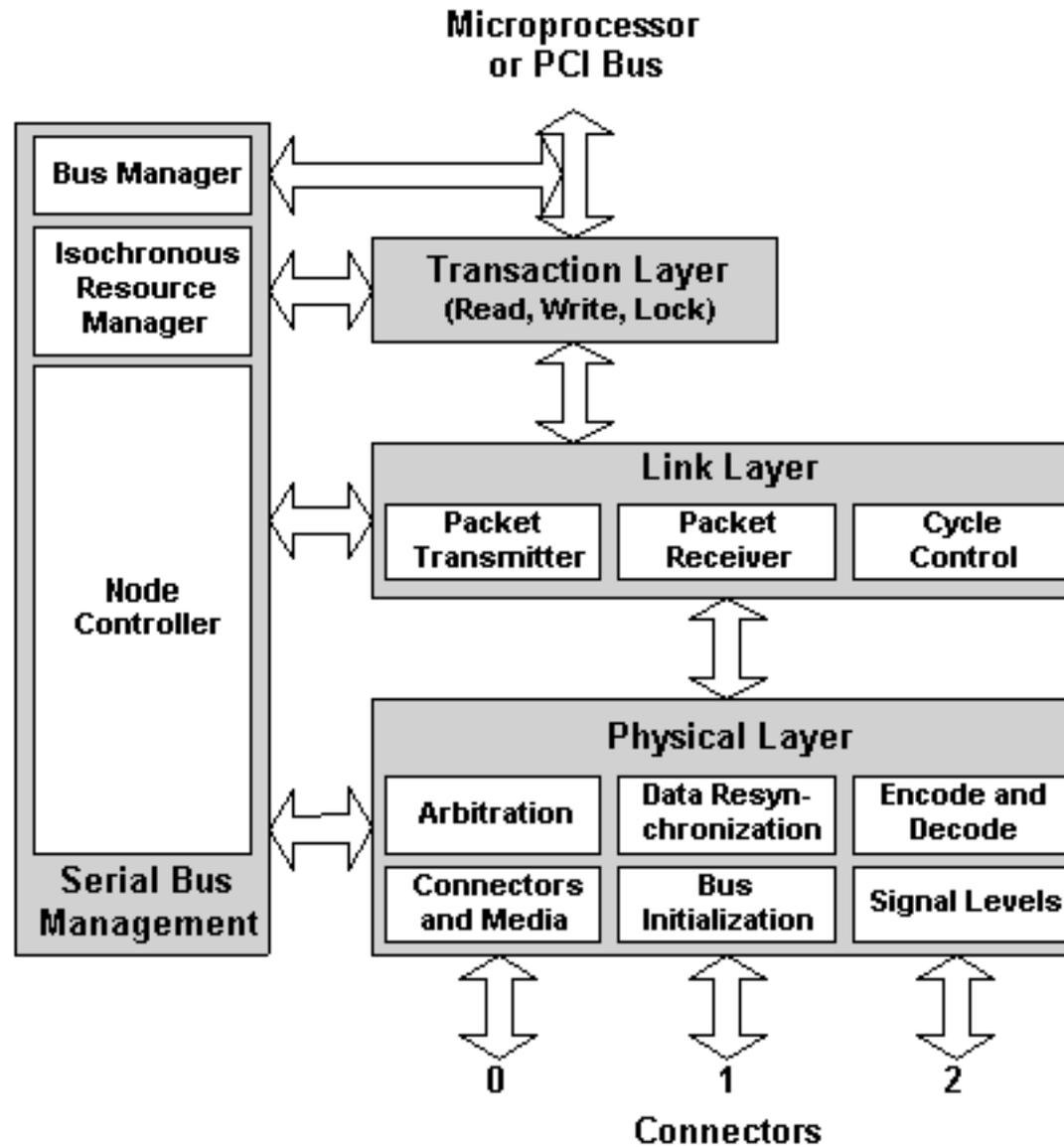


- 100, 200 und 400 MBit/s
- IEEE 1394b: 800, 1800, 3200 MBit/s
- Anschluß interner und externer Digitizer und Speichermedien
- Überspielkabel für Digital Video (DV)
- Reservierter Durchsatz und asynchrone Transfers
- Paar 1 für Daten, Paar 2 für Strobe zur Taktrückgewinnung

- Typische Konfiguration der Standardisierung
  - nichtlineares Video-Editieren
  - root = cycle master
  - Manager: Bus und Isochronous Resources



- Software-Instanzen

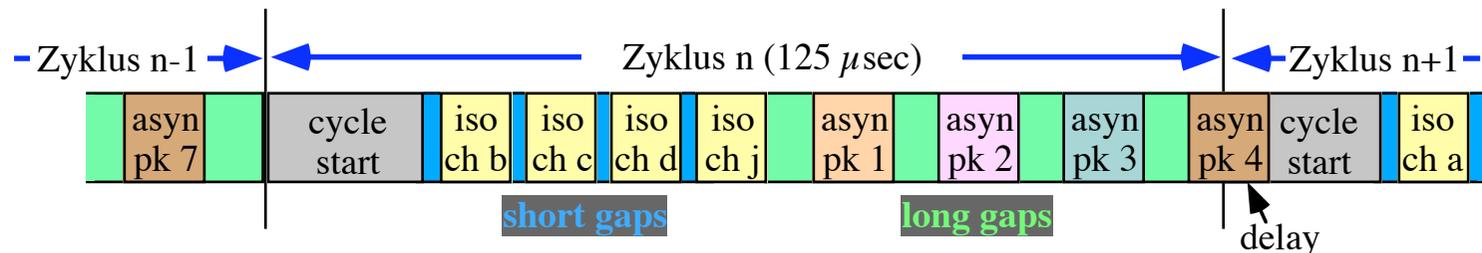


- Bus-Arbitrierung - Isochrone Phase

- wiederholt in den Short Gaps, bis sich keiner mehr meldet

```
FOR i:=1 TO AdrLen DO BEGIN
  others:=0;
  IF myAdr[i]=1 Then Send('1')
  ELSE read(others);
  IF others = 1 THEN EXIT {lost arbitration}
END;
```

- Datenpaket mit vereinbarter Länge
- isochrone Sender können auch Zyklus auslassen

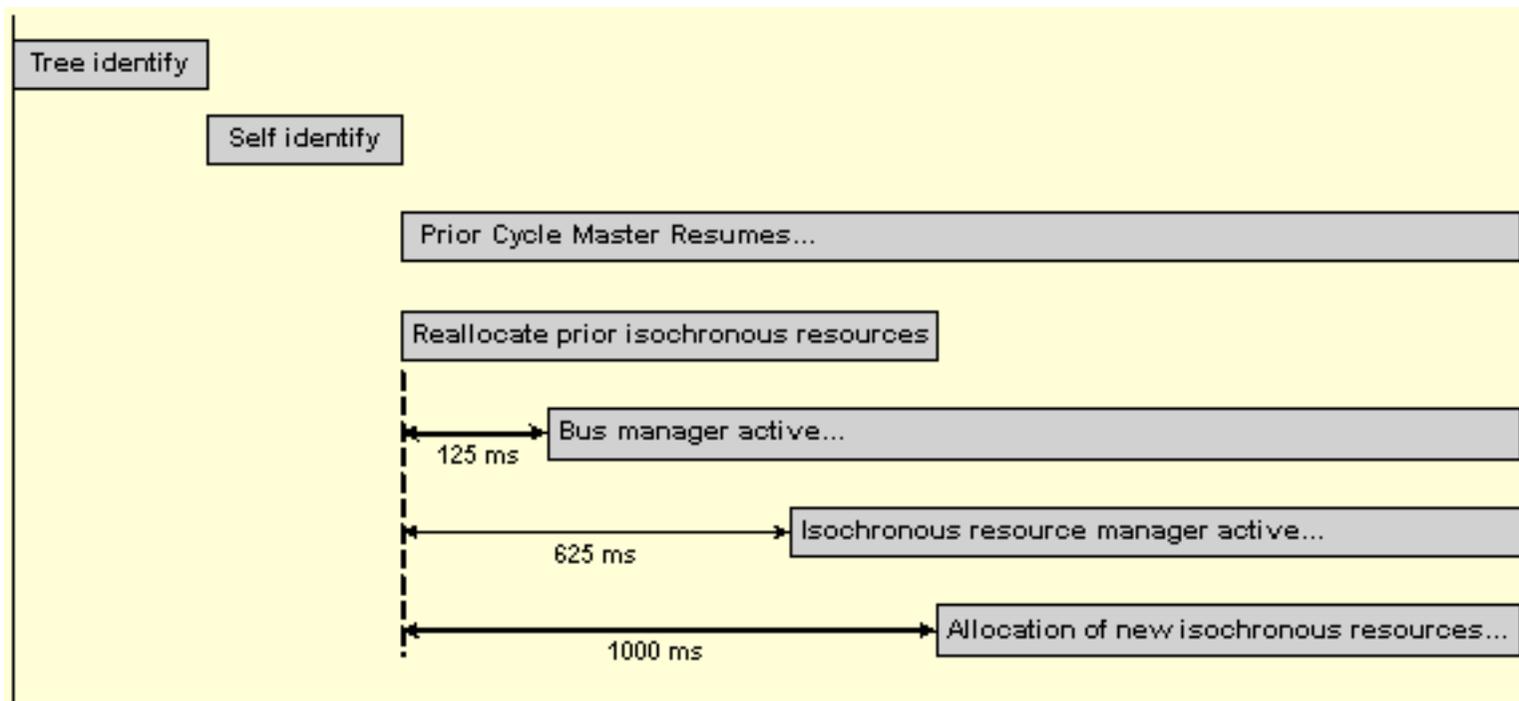


- Bus-Arbitrierung - Asynchrone Phase

- Request von allen Sendewilligen
- arbeitet sich durch Nodes nach oben: 1. Request weiter, Rest: Deny
- Root sendet Grant an Auserwählten und Deny an andere Nodes
- Fairness-Intervall

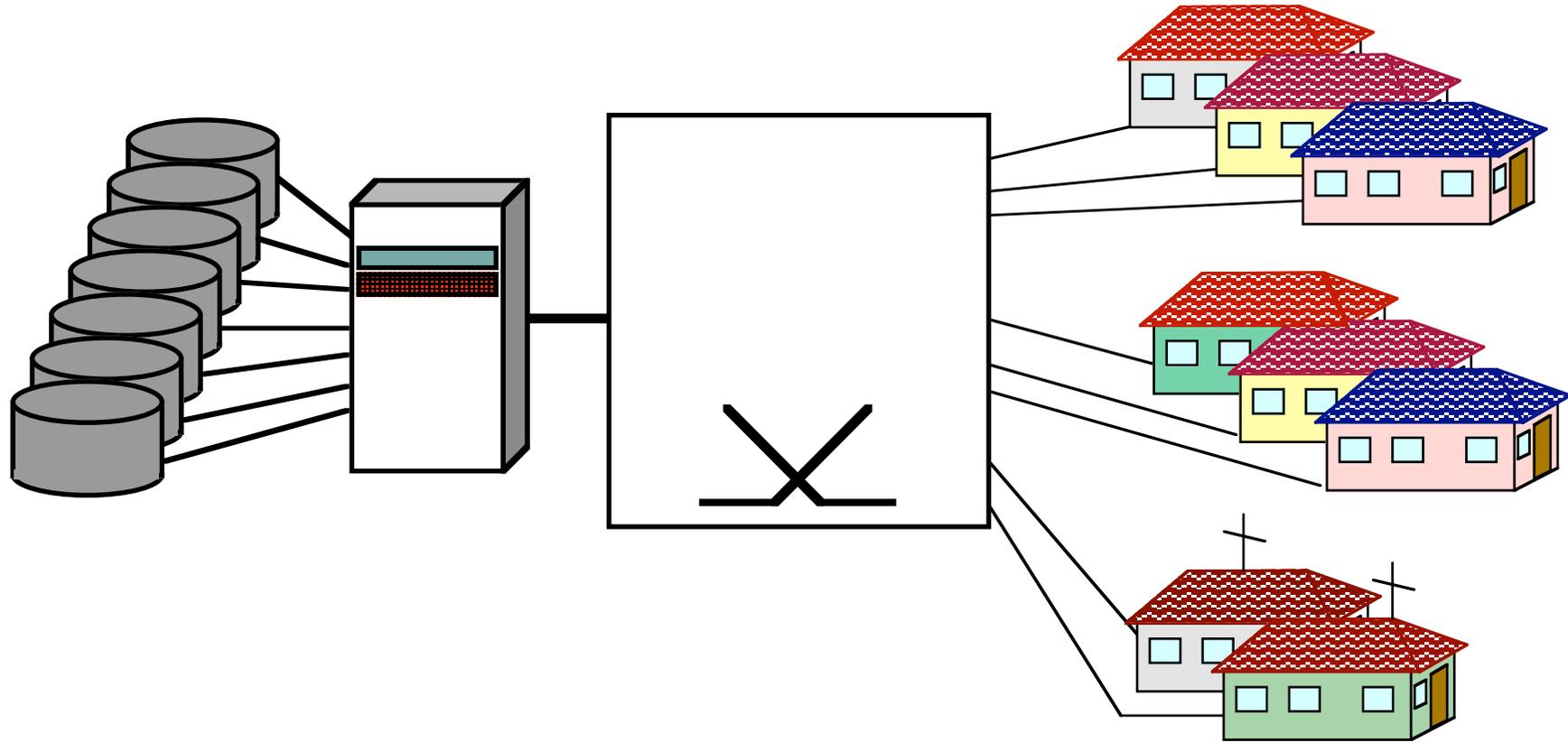
## • Bus-Reset

- primäres Managementinstrument
- hot-plugging
- Verteilen von IDs
- Zuteilung von isochronen Ressourcen
- mindestens 20% für asynchronen Dienst



## 2.5 Speichersysteme

- Beispiel 'video on demand'

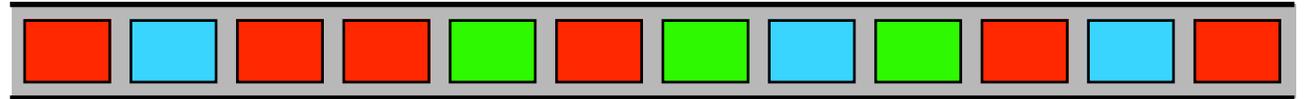


- viele Klienten (1.000, 10.000, 100.000)
- Gigantische Datenmengen (100 - 10.000 Spielfilme, 120 min)  
TV-Qualität (MPEG-2):  $4 \text{ MBit/s} * 60 * 120 / 8 = 3,6 \text{ GByte}$
- Echtzeitbedingungen

## 2.5.1 Verbesserte Platzierung von Dateien

- Kopfpositionierung
- Rotationslatenz
- Kapazitätsoptimierung
- Verstreut

- natürliche Entstehung

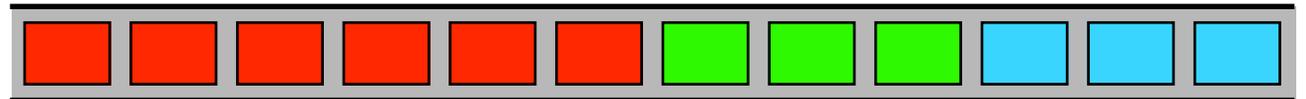


- viel Positionierung

- fehlerhafte Sektoren abbilden auf Reservespuren

- Zusammenhängend

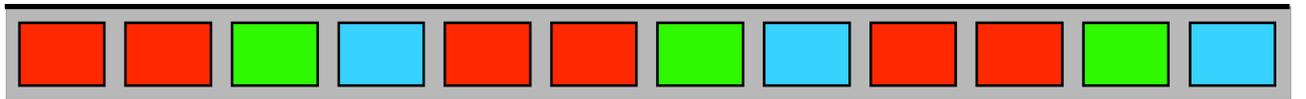
- wenig Positionierung



- Verschnitt wächst bei Änderungen -> Sortierläufe

- zeitlich korrekter Zugriff nicht leichter

- Interleaved



- zeitlich korrekter Zugriff vereinfacht

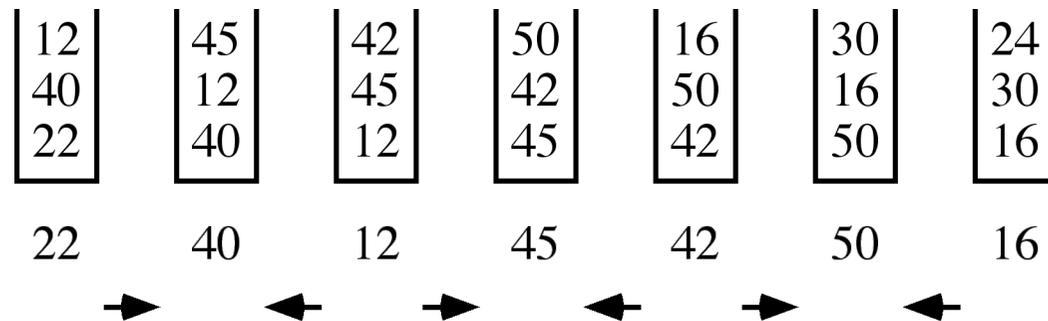
- Einfügen und Löschen schwierig

- schwer planbar

## 2.5.2 Disk Scheduling

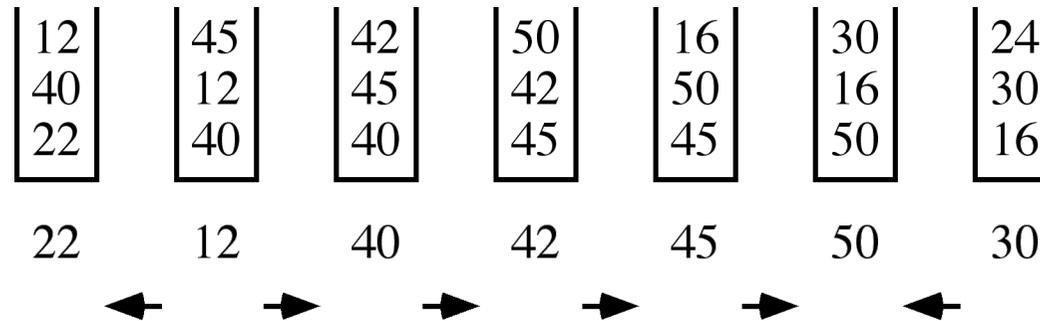
### 2.5.2.1 Klassische Methoden

- Kriterien
  - Antwortzeiten
  - Durchsatz
  - Fairness (Plattenränder und Plattenmitte)
  - Positionierungszeit und Rotationsverzögerung
- First Come First Serve (FCFS)
  - einfache Planung
  - überflüssige Kopfbewegungen



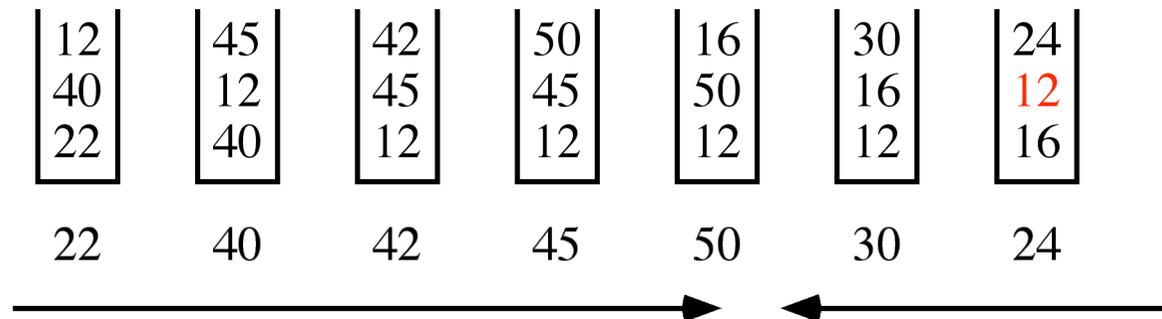
- Shortest Seek Time First (SSTF)

- weniger Positionierung
- höhere Antwortzeiten
- unfair



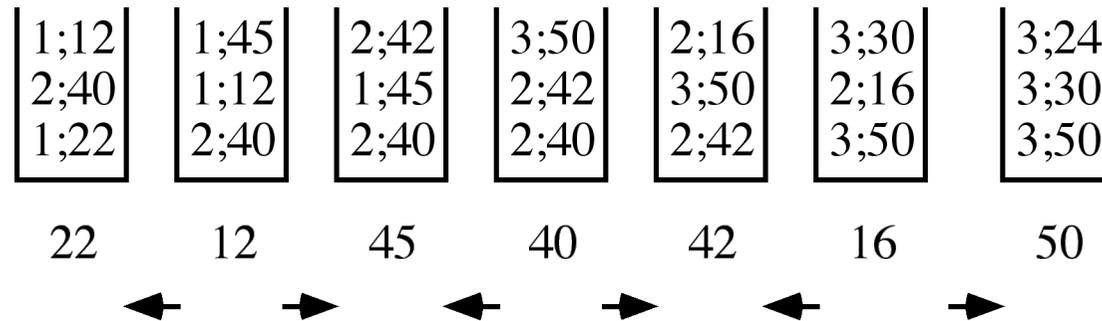
- SCAN

- Nächster Block in Bewegungsrichtung
- Kopf ändert Richtung nur an Plattenenden
- wenig Positionierung
- hohe Antwortzeiten



## 2.5.2.2 Integration kontinuierlicher Medien

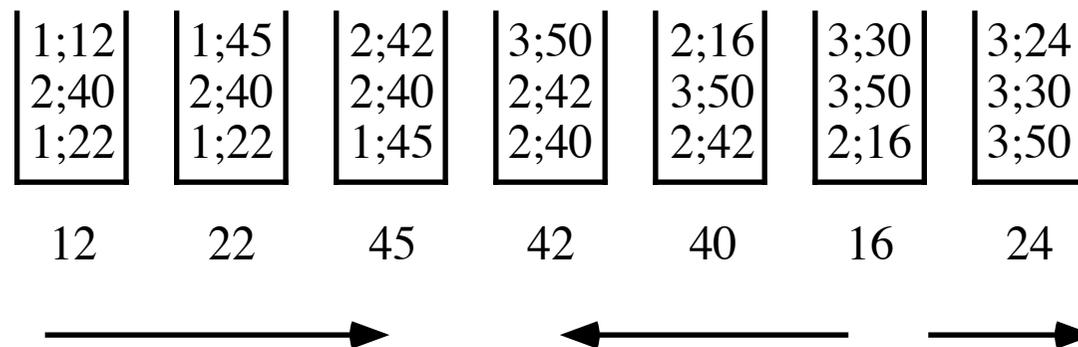
- Maßnahmen zur Einhaltung von Zeitgrenzen
- EDF (Earliest Deadline First)?
  - Block mit nächster Deadline lesen



- viel Positionieren
- wenig Durchsatz

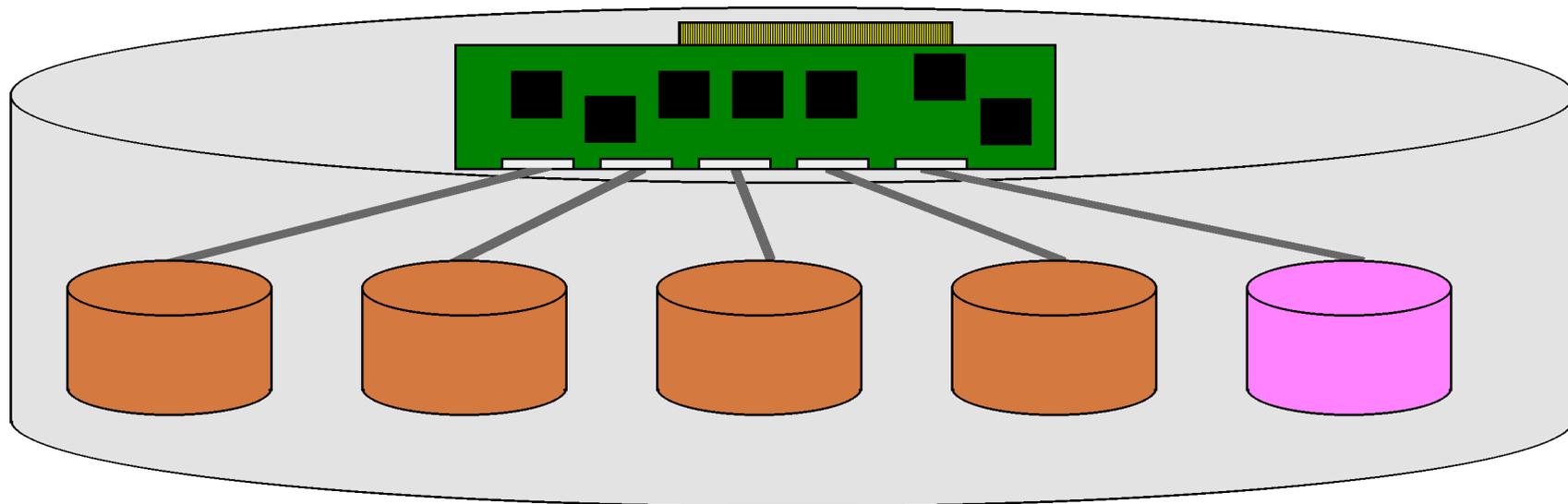
### • SCAN + EDF

- bei gleicher Deadline Bewegungsrichtung beibehalten



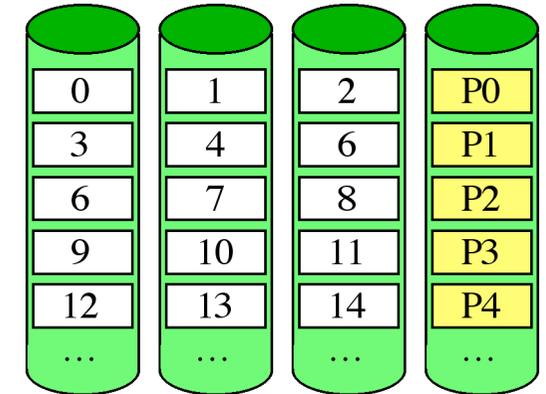
## 2.5.3 RAID - Redundant Array of Independent Disks

- A Case for Redundant Arrays of Inexpensive Disks [1987, UCB]
  - Datenintegrität
  - I/O-Durchsatz durch parallele Interfaces
  - niedrigere Kosten im Vergleich zu Hochleistungslaufwerken
  - 'striping' + Prüfsumme



- RAID-Taxonomie
  - 5 Ebenen (layers)
  - ansteigende Komplexität
  - zusätzliche Layer (6, ...) für verbesserte Geschwindigkeit
  - Kombinationen

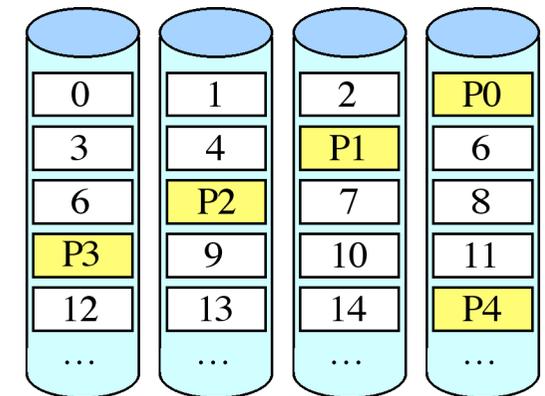
- RAID 0: mehre Platten mit striping ohne Redundanz
- RAID 1: Schreiboperationen auf 2 Festplatten gleich (Plattenspiegeln)
- RAID 3: stromorientierte Anwendungen)



- byte striping: Daten byteweise auf Platten verteilt
- separate Platte(n) mit Parity
- Obermenge von RAID 2
- Platten synchronisiert

- RAID 4

- block striping
- einzelne Blöcke erfordern nur einen Zugriff
- Blockgruppen von n Platten
- **separate Platte(n) mit Parity <-> Parallel Schreiben?**



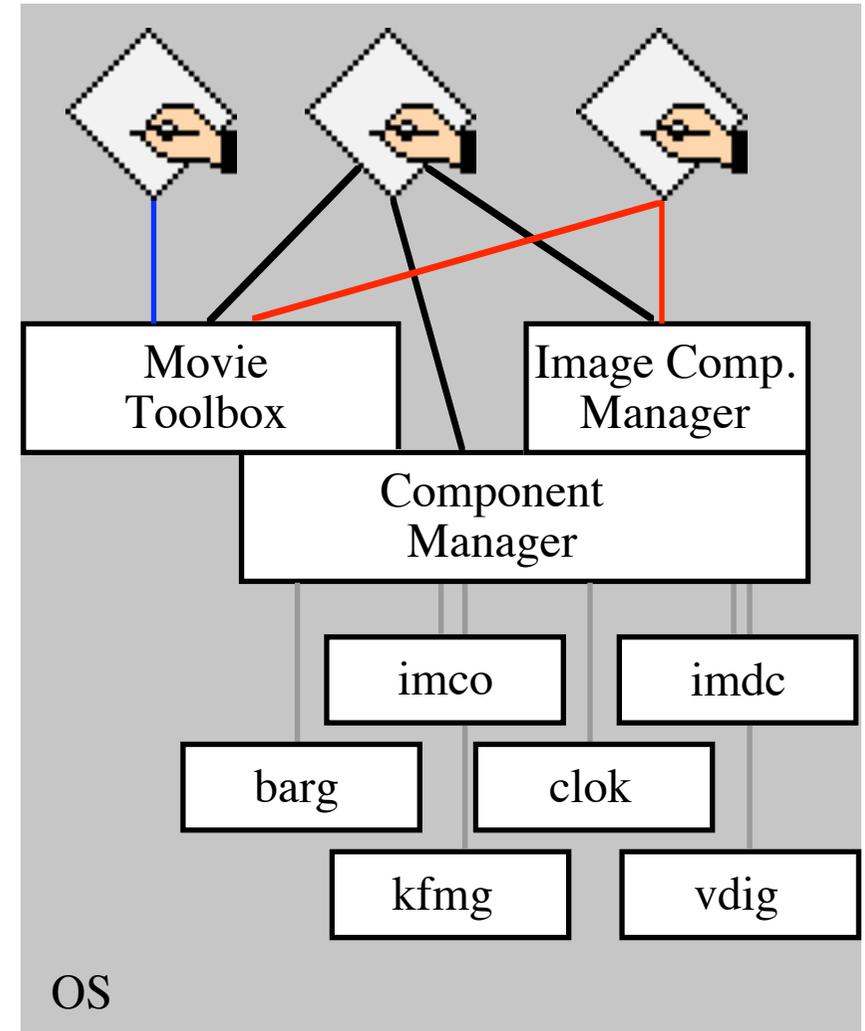
- RAID 5: transaktionsorientierte Anwendungen)

- block striping mit **verteilter Parity**
- Flaschenhals Parity reduziert
- Platten nicht synchronisiert

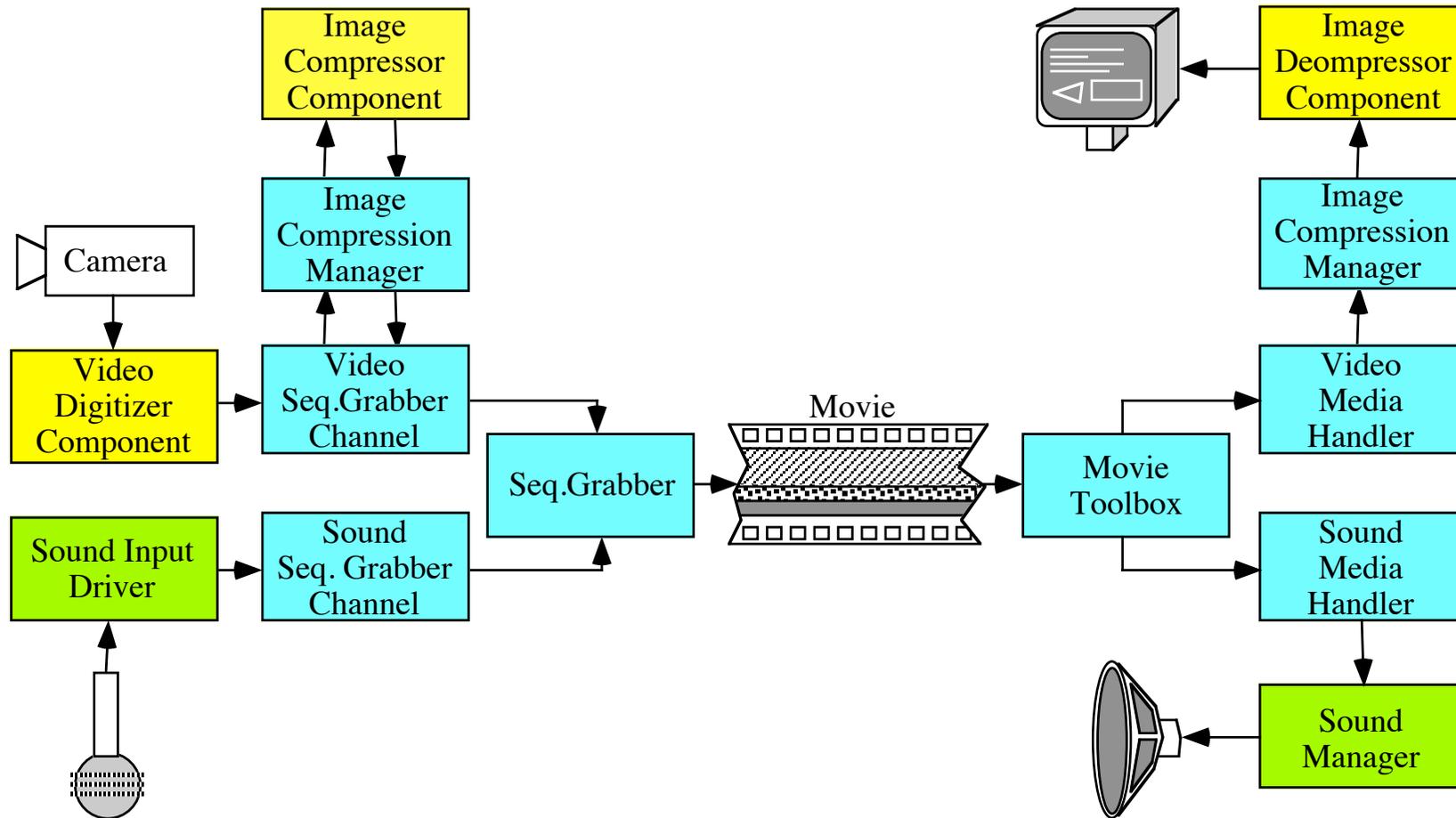
- RAID 6, RAID 7, RAID 35

### 3. Multimedia-Programmierung am Beispiel: Quicktime

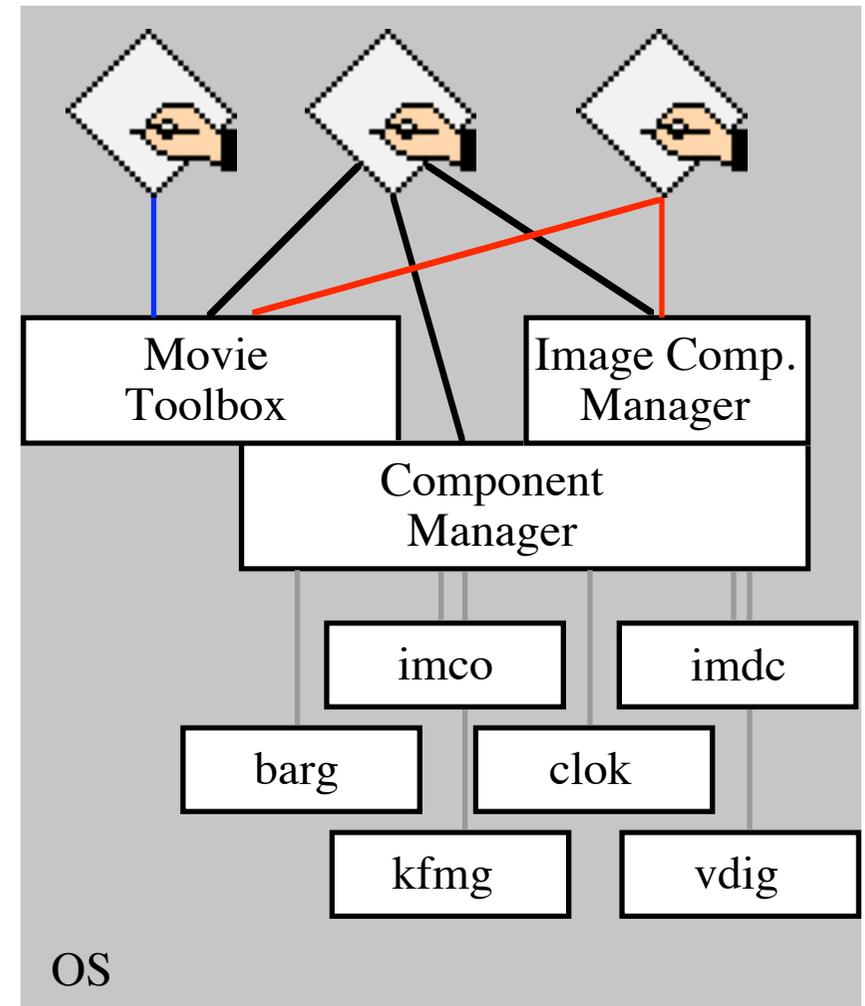
- Multimedia Toolbox
- System Software
- Rahmen für Multimedia Komponenten
  - Abstraktion: Hardware, Software
  - Management
- Application Programmers Interface
- Dateiformat
- Std-Applikationen -> Movie Toolbox
- Multimedia-Applikationen
  - > Component-Manager
  - > Components
- ICM -> Image Codecs
- Sequencegrabber
  - > Component-Manager
  - > video-Digitizer



# • Funktionale Komposition



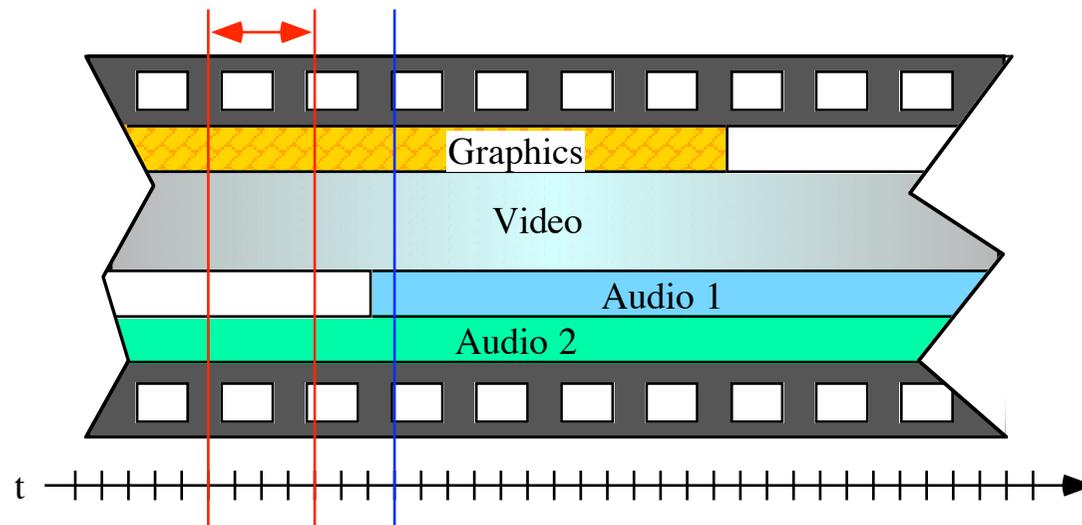
- Movie Toolbox zur Filmverarbeitung
  - Suchen, Öffnen und Schließen
  - Abspielen
  - Aufnehmen und Editieren
- Image Compression Manager
- Component Manager
  - verwaltet Komponenten
  - 'Datenbank'
  - Interface zu den Komponenten
- Komponenten
  - Device Driver
  - von Apple und anderen Herstellern
  - Standard Interface für Hardware
  - Sequence Grabber (barg)
  - Video Digitizer (vdig)
  - Image Compressor (imco),
  - Image Decompressor (imdc)



## 3.1 Movies

- Movie

- Zeit: Skala und Dauer
- 1..n Tracks
- Preview: Startzeit, Dauer, Tracks
- Poster: Zeitpunkt im Movie
- in derselben Datei oder verstreut in anderen Dateien

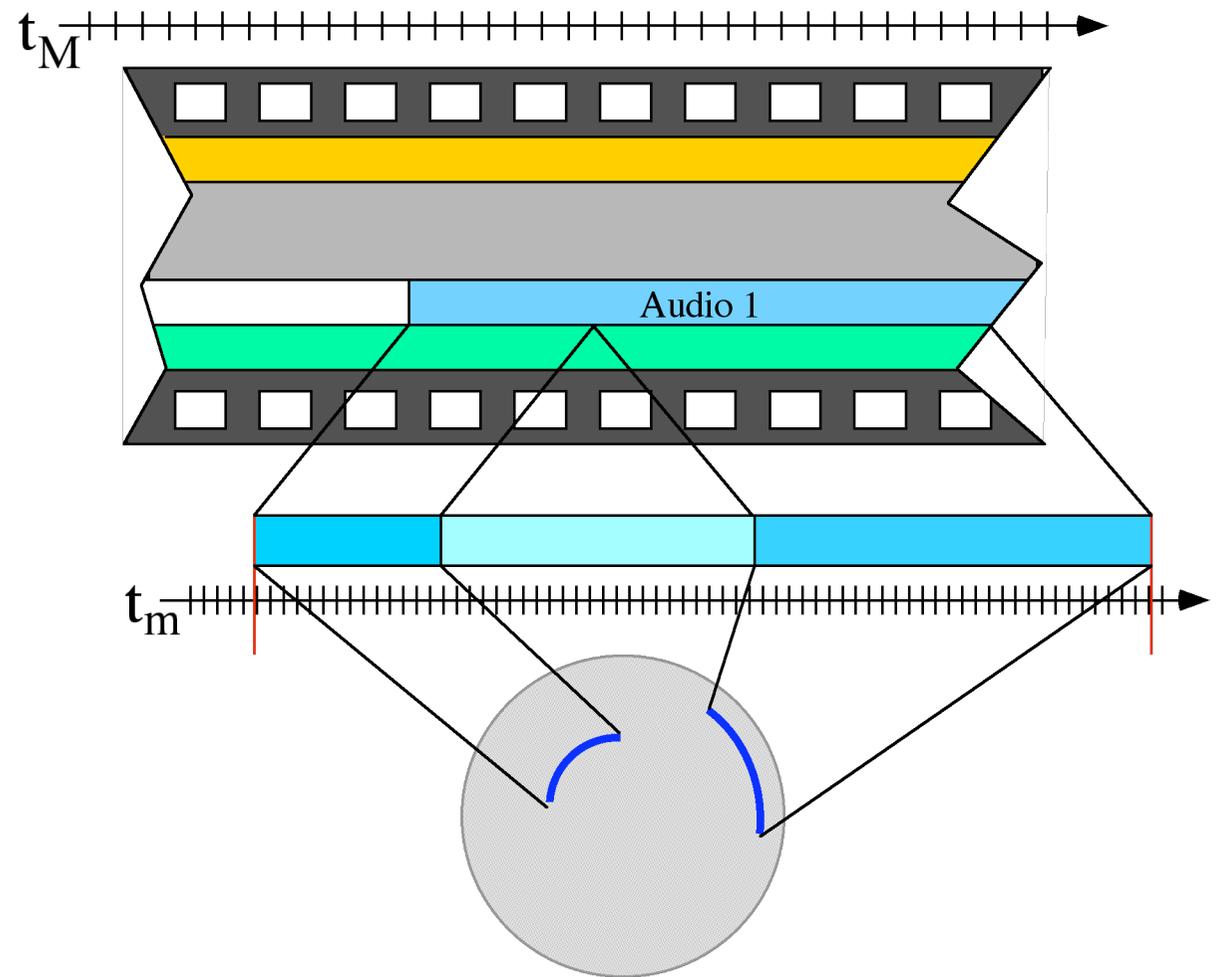


- Track

- Einzelner Medienstrom
- Zeitinformation: Dauer, Offset
- Liste von Referenzen auf Medium (Edit-Liste)
- Präsentationseigenschaften
  - Transformationsmatrix,
  - Clipping-Rgn, ...
  - Lautstärke, ...

- Media

- Trackdaten
- Zeitkoordinatensystem
- Media-Handler



### 3.1.1 QMF - QuickTime Movie Format

- Dokumentenformat Apple's QuickTime
  - Multimedia-Filme
  - Video, Audio, Text, Grafik, ...
  - Präsentationsinformation (Transformationen, Zeitbezüge, ...)
- MPEG-4 Dateiformat
  - Speicherung auf dem Server
  - Austausch zwischen Servern
  - ≠ Stromformat
- Movie ist eine Datenstruktur
  - Metadaten
  - Anzahl Tracks, Kompressionsverfahren, Zeitbezüge
  - Verweise auf Mediendaten (Samples, Video-Bilder)
  - Mediendaten im File oder in anderen Dateien

- Information in Atomen

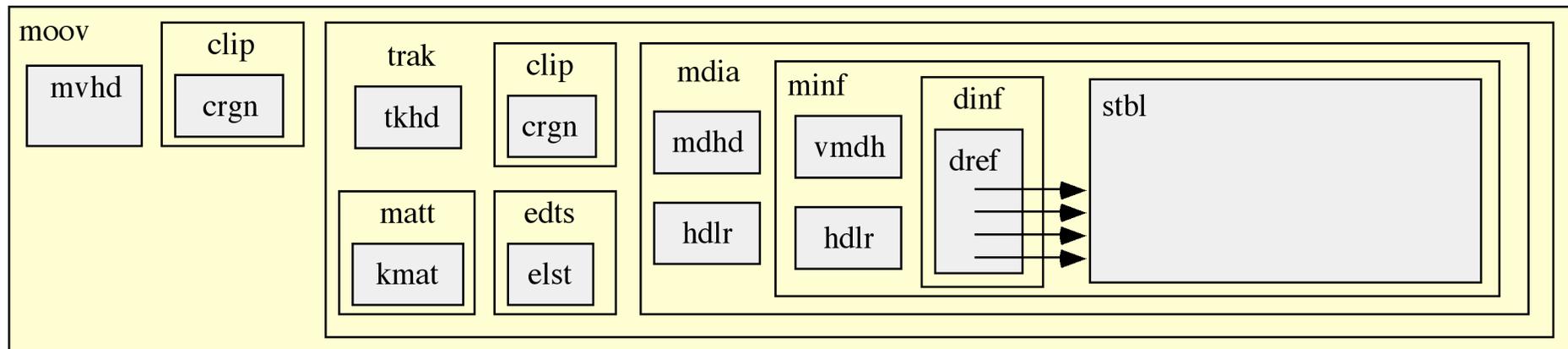
- Behälter-Atom, Blatt-Atom



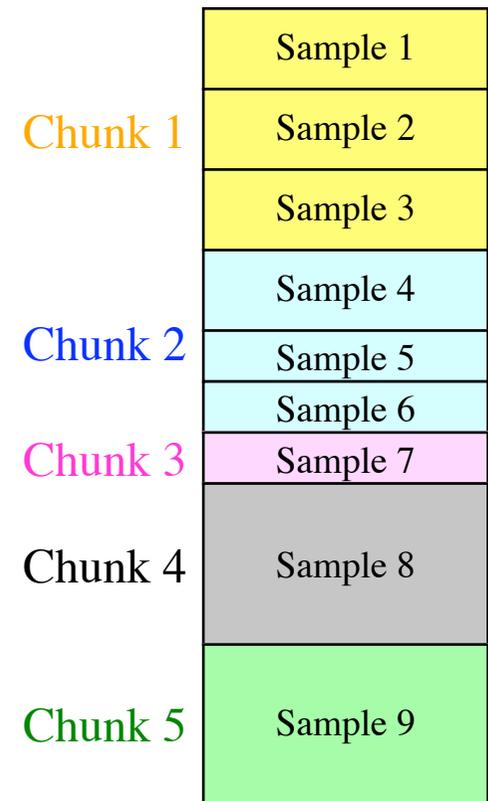
- Grösse in 32 bit
- Typ 4 Zeichen ('clip', 'trak', 'moov', ..., 'free', 'skip')
- auch komplexere und flexible QT-Atome möglich
- Nummer: 32 bit
- Anzahl Kinder: 16 bit

- Atome unsortiert in der Datei

- Strukturinformation im Data- oder Resource-Fork
- Sample-Daten im Data-Fork

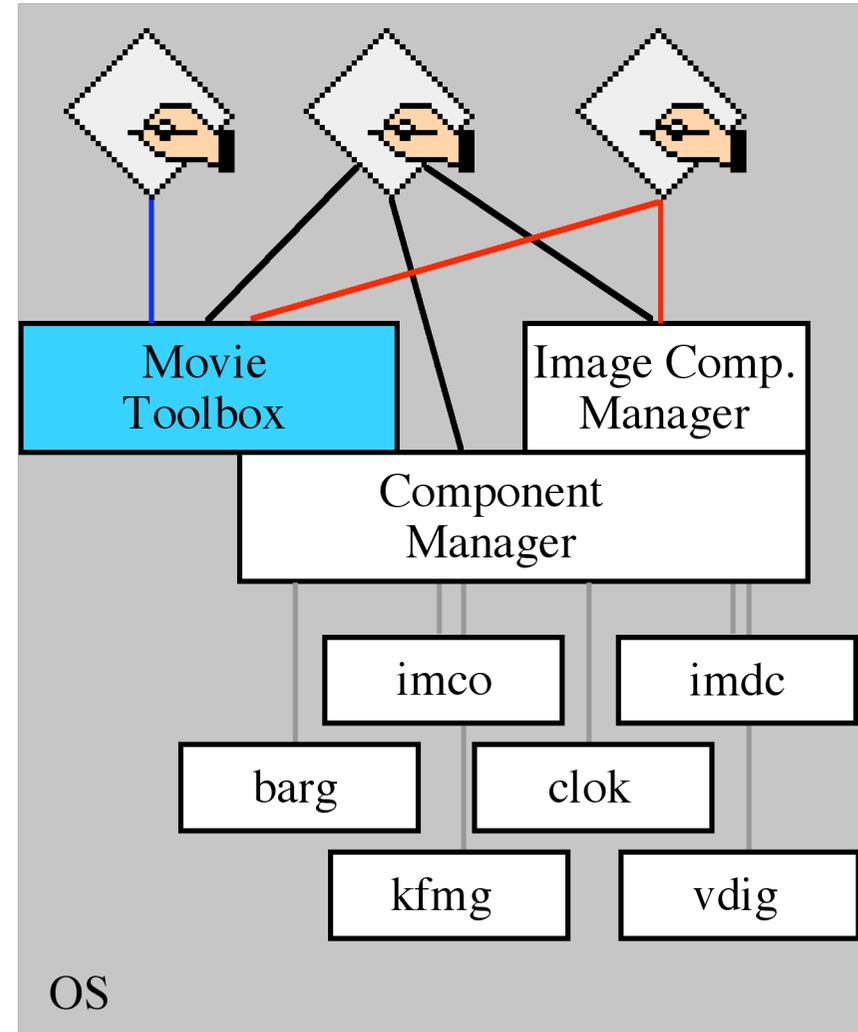


- Movie Header Atom
  - version, flags, creation time, modification time
  - duration, preferred rate, preferred volume, matrix
  - preview
  - Variablen: selection, current time, next track
- Edit List
  - Medienteile eines Tracks
  - Dauer und Zeiger (offset) in Quelle
- Data Reference Atoms ('dref')
  - Anzahl Einträge, Referenzen
  - Referenz: 'alis'/'rsrc', flags
  - Pfadname oder Res-Typ und Res-Id
- Sample Table Atom ('stbl')
  - time-to-sample (Anzahl Samples, Sample-Dauer)
  - sync sample (Indices der Keyframes)
  - sample-to-chunk
  - sample size (Tabelle mit Sample-Größen)
  - chunk offset (Tabelle mit chunk-Anfang in Bytes)

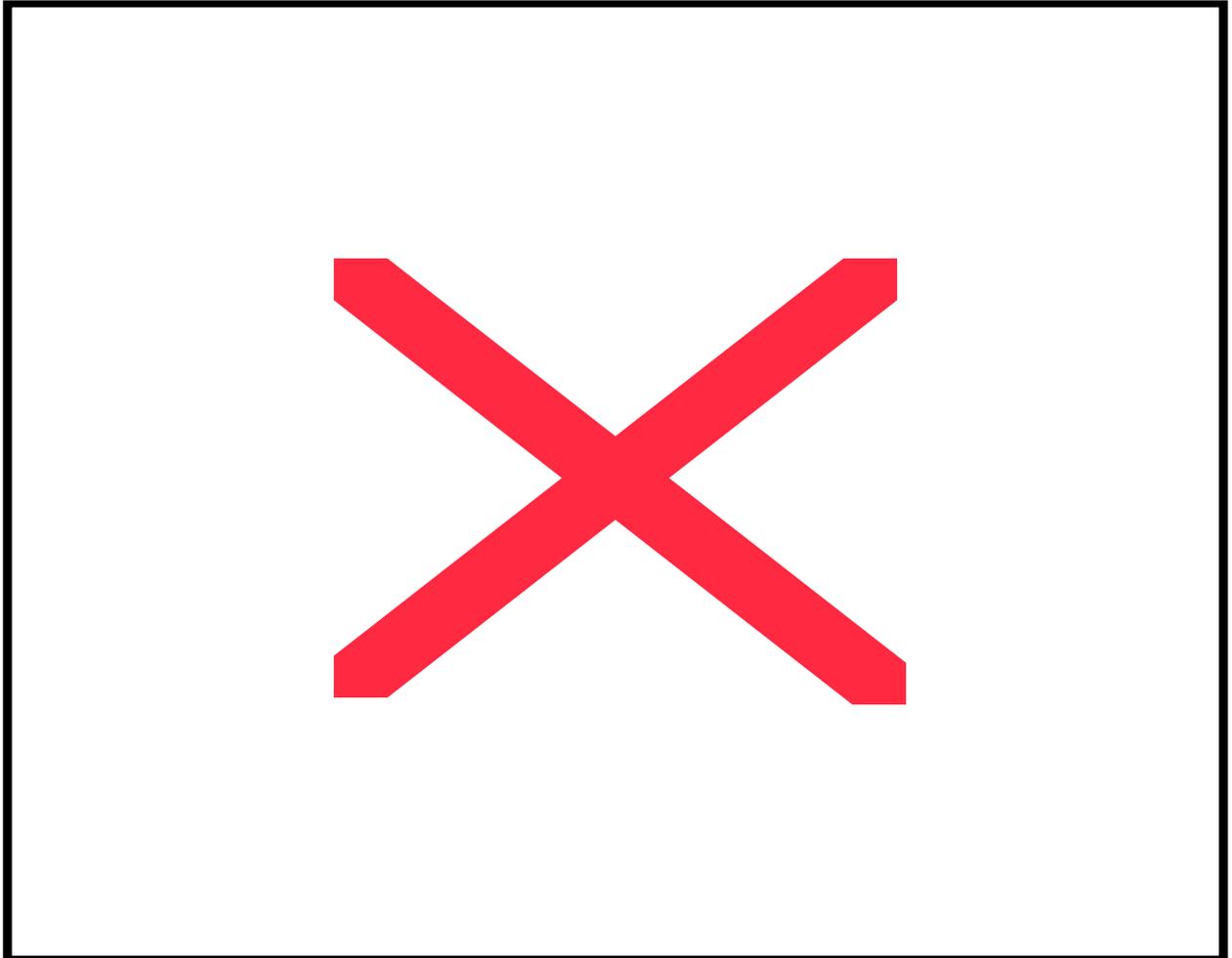


## 3.2 Movie Toolbox

- Movie erzeugen
  - NewMovie
  - SetMovieBox, SetMovieGWorld
  - NewMovieTrack
  - NewTrackMedia
  - FlattenMovie
- Einfache Erweiterung von Apps
  - Abspielfunktion
  - NewMovieFromFile
  - StartMovie, StopMovie
- Bearbeitung von Movies
  - Edit-Standardoperationen:  
Cut, copy, paste, clear, undo
  - Clipboard



- Beispielprogramm Movie Toolbox
  - Annotation von Filmen (Untertitel, Erklärungen)
  - Rechteck mit Text unter dem Film-Rechteck
  - Gesteuert durch Film-Zeit
- Struktur
  - Initialisieren
  - GetOurMovie
  - MovieMenu
  - DisplayAnnotations
  - Carbon-Interface



- GetOurMovie

```
void GetOurMovie(void)
{ SFReply    aReply;
  OSErr      myerr;
  short      rRef;
  short      resId = 0;

  mySFGetFile('Moov', &aReply);
  if (aReply.sfGood)
  { if (OpenMovieFile(&aReply.sfFile, &rRef, fsRdPerm) == 0)
    { SetPort(GetWindowPort(window));
      myerr=NewMovieFromFile(&myMovie, ..., newMovieActive, NULL);
      myadjustWindow();
      CloseMovieFile(rRef);
      mySFGetFile('TEXT', &aReply);
      if (aReply.sfGood)
        { if (FSpOpenDF(&aReply.sfFile, fsRdPerm, &rRef) == 0)
          { ReadAnnotations(rRef); myerr=FSClose(rRef);
        } } }
    } } }
```

- DoMenuCommand

```
pascal OSStatus MovieMenu( EventHandlerCallRef next,
                          EventRef theEvent, void* uD)
{ HCommand cmd;

  GetEventParameter( theEvent, ..., typeHCommand, ..., NULL, &cmd );

  if (cmd.commandID==kCmdStop)      { StopMovie(myMovie);
                                     return noErr; }
  if (cmd.commandID==kCmdPlay)     { StartMovie(myMovie);
                                     nextAnnoStop=FALSE;
                                     return noErr; }
  if (cmd.commandID==kCmdNextAnno) { StartMovie(myMovie);
                                     nextAnnoStop=TRUE;
                                     return noErr; }
  if (cmd.commandID==kCmdRewind)   { StopMovie(myMovie);
                                     GoToBeginningOfMovie(myMovie);
                                     currAnn=0; return noErr; }
  return CallNextEventHandler(next, theEvent);
}
```

- DisplayAnnotations

```
pascal void DisplayAnno(EventLoopTimerRef theTimer, void* uD)
{ Str255      currTime;
  TimeRecord  timeRec;
  long        theTime;
  CGrafPtr    savePort;

  MoviesTask(NULL,0);    // give QT time to display the video

  GetPort(&savePort);
  SetPort(GetWindowPort(window));
  EraseRect(&timeRect);
  theTime=GetMovieTime(myMovie,&timeRec);
  NumToString(theTime,currTime);
  MoveTo(timeRect.left,timeRect.bottom);
  DrawString(currTime);
  if (theTime >= timeArr[currAnn])
  { EraseRect(&annoRect);
    MoveTo(annoRect.left,annoRect.bottom-3);
    DrawString(annotations[currAnn]);
    currAnn=currAnn+1;
    if (nextAnnoStop) StopMovie(myMovie);
  }
  SetPort(savePort);
}
```

- Hauptprogramm und Events

```
void InstallHandlers()  
{ EventTypeSpec myEvents =  
    {kEventClassCommand, kEventCommandProcess};  
  
    InstallApplicationEventHandler(NewEventHandlerUPP(MovieMenu),  
    1, &myEvents, 0, NULL);  
    InstallEventLoopTimer(..., kEventDurationSecond/30,  
    NewEventLoopTimerUPP(DisplayAnno), ...);  
}
```

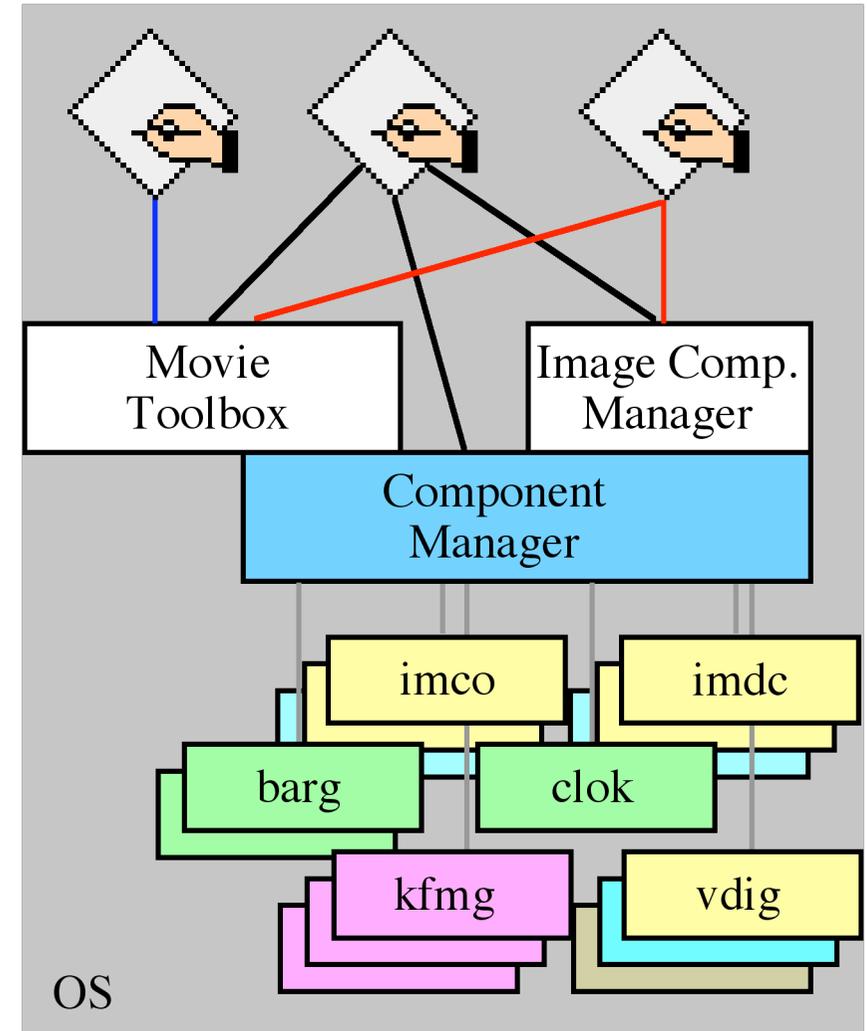
```
int main(int argc, char* argv[])  
{ OSStatus err;  
  
    err = initialize(); // make window etc.  
    if (err==noErr) {  
        GetOurMovie();  
        InstallHandlers(); // two: MovieMenu and a timertask  
        RunApplicationEventLoop(); // Let carbon do the loop  
    }  
    return err;  
}
```

## 3.3 Component Manager und Komponenten

- Objektorientiert
  - Klassen
  - mehrere Instanzen
  - Vererbung
- Suchen von Componenten

```
desc... = ...;
co=NULL;
done=FALSE; i=0;
max = CountComponents (&desc);
do
{co=FindNextComponent (co, &desc);
 GetComponentInfo (co, &cdesc, inf);
 done= (inf... == ...);
 i=i+1;
} while (!done && i<max);
```
- Öffnen und Schließen

```
cInst=OpenComponent (co);
```
- Benutzen
  - komponentenspezifisch
  - Standardmechanismus zum Aufruf



## Wichtige Komponenten

- Sequence Grabber: 'barg'
  - Audio und Video
  - **SGSetDataOutput**(grabComp, fileSpec, ...);
  - **SGNewChannel**(grabComp, **SoundMediaType**, channelId);
  - **SGStartRecord()**, **SGStop()**, **SGPause()**,
  - **SGSettingsDialog**(grabComp, channelId, ...)
- Sequence Grabber Channels: 'sgch'
- Kompression: 'imco' bzw. 'imdc'
  - CDBandCompress / CDBandDecompress
  - Standard Image Compression Dialog Component ('scdi')
- Video Digitizer: 'vdig'
  - **VDSetInput**, **VDSetInputFormat**, **VDSetInputStandard**
  - **VDSetActiveSrcRect**
  - **VDSetPlayThroughDestination**
  - **VDGrabOneFrame**, **VDGrabOneFrameAsync**(,,CompProc,,)
  - eventuell mit Kompression
  - Kontrolle des AD Wandlers (Farbe, Helligkeit, ...)

- Beispiel: MovieController
  - Videorecorderfunktion
  - Editieren von Movies: Cut, Copy, Paste und Undo

```

pascal OSStatus DoMovieEdit( EventHandlerCallRef next,
                             EventRef theEvent)
{
  ComponentResult err;
  HICommand cmd;

  GetEventParameter(theEvent, ..., sizeof(HICommand), NULL, &cmd);

  if (cmd.commandID==iUndo)  {err = MUndo(contComp);
                             return noErr;}
  if (cmd.commandID==iCut)   {clipMovie = MCCut(contComp);
                             return noErr;}
  if (cmd.commandID==iCopy)  {clipMovie = MCCopy(contComp);
                             return noErr;}
  if (cmd.commandID==iPaste) {err = MCPaste(contComp, clipMovie);
                             return noErr;}
  if (cmd.commandID==iClear) {err = MCClear(contComp);
                             return noErr;}

  return -1;
}

```

```

pascal OSStatus ControllerMenu( EventHandlerCallRef next,
                                EventRef theEvent, void* uD)

    HICommand cmd;
    Rect aRect; Point aPoint;
    ComponentResult cErr;

    GetEventParameter(theEvent, ..., sizeof(HICommand), NULL, &cmd);
    if (cmd.commandID == kCmdCOpen) {
        contComp = OpenDefaultComponent('play', 0);
        if (contComp != NULL) {
            GetMovieBox(myMovie, &aRect); aPoint.v=aRect.top; ...;
            cErr=MCSetMovie(contComp, myMovie, window, aPoint);
            cErr=MCSetVisible(contComp, TRUE);
            cErr=MCDraw(contComp, window); }
        return noErr;}
    if (cmd.commandID == kCmdCClose) {
        cErr=MCSetVisible(contComp, FALSE);
        cErr=CloseComponent(contComp); contComp=NULL;
        return noErr;}
    if (cmd.commandID == kCmdCEdit) {
        if (MCIsEditingEnabled(contComp) == 0) {
            cErr=MCEnableEditing(contComp, TRUE); AdjustEditMenu(TRUE); }
        else { cErr=MCEnableEditing(contComp, FALSE);
            AdjustEditMenu(FALSE); }
        return noErr;}
    if (DoMovieEdit(next, theEvent)==noErr) return noErr;
    return CallNextEventHandler(next, theEvent);}

```

```

pascal OSStatus WindowClick(...)
{ ComponentResult compRes;
  EventRecord theEvent;

  if (contComp!=NULL) {
    ConvertEventRefToEventRecord(evtRef,&theEvent);
    compRes= MCIsPlayerEvent(contComp,&theEvent);
    if (compRes=1) return 0;
  }
  return CallNextEventHandler(next,evtRef);
}

pascal void time4QT(EventLoopTimerRef theTimer, void* uD)
{ MCIdle(contComp); // give QT time to display the video }

void InstallHandlers()
{ EventTypeSpec myEvt = {kEventClassCommand,kEventCommandProcess};

  InstallApplicationEventHandler(NewEventHandlerUPP(ControllerMenu),
                                1,&myEvt,0,NULL);

  myEvt.eventClass=kEventClassMouse;myEvt.eventKind=kEventMouseDown;
  InstallApplicationEventHandler(NewEventHandlerUPP(WindowClick),
                                1,&myEvt,0,NULL);

  InstallEventLoopTimer(GetMainEventLoop(),0,kEventDurationSecond/30,New
  EventLoopTimerUPP(time4QT),NULL,&t4QTimer);}

```

## Beispiel-Komponente: cool

```
typedef struct {...} CoolRecord, **globHandle;

pascal ComponentResult CoolOpen(globHandle globs,
                                ComponentInstance self)
{globs = (globHandle)Newhandle(sizeof(CoolRecord));
  if (!globs) return MemError;
  SetComponentInstanceStorage((Handle)globs);
  Return noErr; }

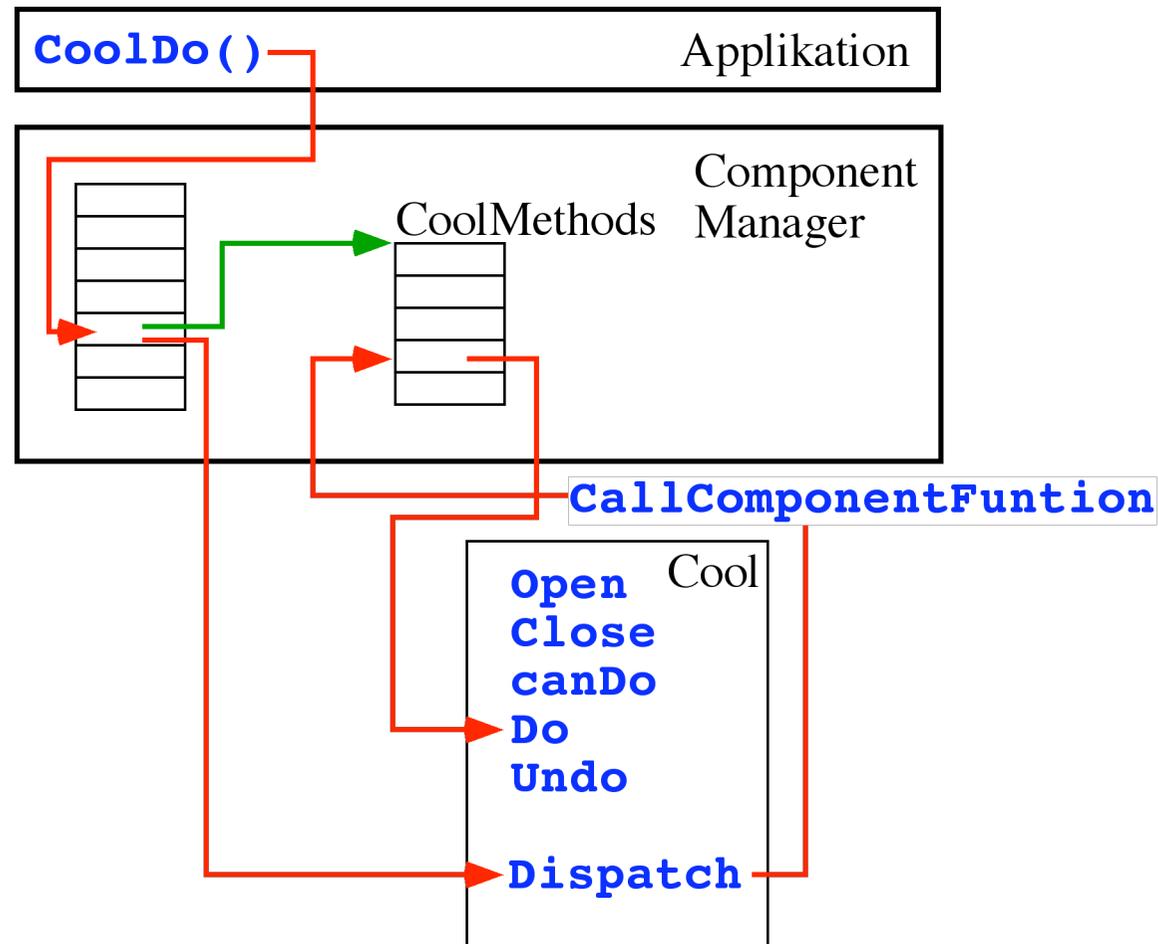
pascal ComponentResult CoolClose(GlobHandle globs,
                                  ComponentInstance self)
{if (globs) Disposehandle((handle)globs); return noErr;}

pascal ComponentResult CoolCanDo(short selector)
{;}

pascal ComponentResult CoolDo(globHandle globs)
{;}

pascal ComponentResult CoolUndo(GlobHandle globs)
{;}
```

- Objektorientierter Aufruf durch ComponentManager und Component
  - Componenttabelle, Methodentabellen
  - Dispatchprozedur in der Component



```
// CallComponentFunctionwithStorage    -> CallCFStore
// CallComponentFunction                -> CallCF
```

```
pascal ComponentResult CoolDispatch(ComponentParameters *parms,
                                     Handle store)
{
  switch (parms->what) {
    case kComponentOpenSelect: return CallCF(parms, &CoolOpen);

    case kComponentCloseSelect:
      return CallCFStore(store, parms, &CoolClose);

    case kComponentCanDoSelect: return CallCF(parms, &CoolCanDo);

    case kComponentVersionSelect: return 0;

    case coolDoSelect: return CallCFStore(store, parms, &CoolDo);

    case coolUndoSelect: return CallCFStore(store, parms, &CoolUndo);

    default return badComponentSelector;
  } }
}
```

- Aufruf: `erg = CoolDo(...);`

## 4. Synchronisation

- Multimedia = Mischung mehrerer Medien

- Bezüge zwischen Medien

- inhaltlich: Grafik und Tabelle: gemeinsame Daten

- räumlich: Layout, Benutzungsschnittstelle, Stereo, ...

- zeitlich: gleichzeitig, unabhängig, hintereinander

- Erschwerte Bedingungen durch kontinuierliche Medien

- Synchronität

Einhaltung der inhaltlichen, räumlichen und/oder zeitlichen Bezüge bei der Präsentation mehrerer, gemischter Medienströme.

- Synchronisation

- Meyers Lexikon:

- Die Herstellung des Gleichlaufs zwischen zwei Vorgängen, Maschinen oder Geräten bzw. -teilen.

- Herstellung der Synchronität

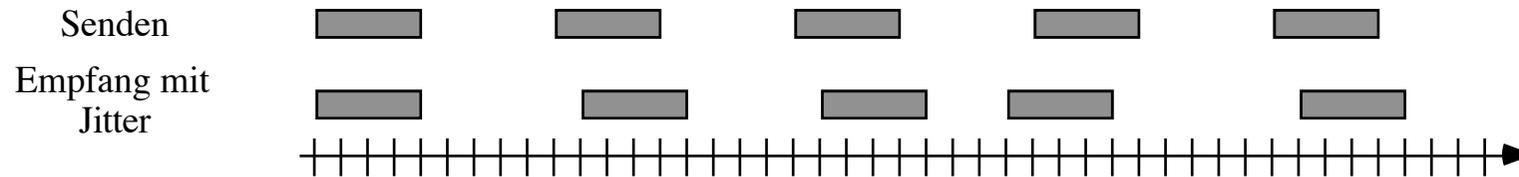
- Orchestrierung (Orchestration)

- Koordinierte Präsentation von verschiedenen Datenströmen
  - Audio, Video, Grafik, Text
  - Steuerinformation der Applikation (Benutzungsschnittstelle)
- Weiche Synchronitätsanforderungen
  - Gegensatz zu Prozesskommunikation
- Lokale Synchronität (Punkt-zu-Punkt)
  - Audio  $\Leftrightarrow$  Video (Lippensynchron)
  - Untertitel (Text/Grafik)  $\Leftrightarrow$  Video
  - Audio  $\Leftrightarrow$  Grafik/Animation
  - Grafik  $\Leftrightarrow$  Grafik
- Globale Synchronität (Mehrpunkt)
  - Präsentation an verschiedenen Arbeitsplätzen
  - Ströme werden zur gleichen Zeit präsentiert
  - essentiell in Gruppenarbeitsszenarien
    - Tokenverteilung
    - Globaler Jitter < 200 msec für Sprachkonferenzen

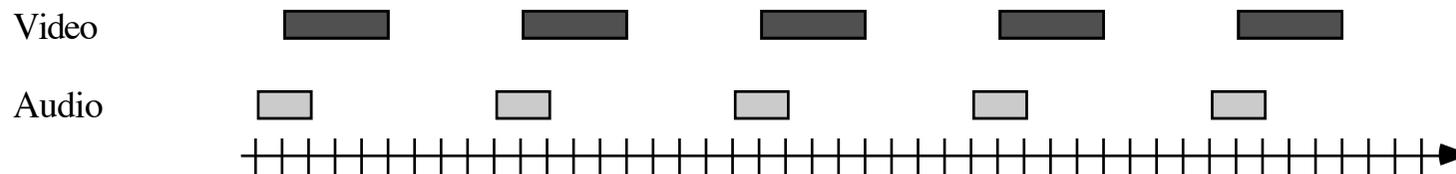
- Natürliche Zeitbezüge durch parallele Aufnahme
  - Live-Übertragung
  - Telepräsenz
  - implizit
- Synthetische Zeitbezüge durch Anordnung (Schnitt)
  - gespeicherte Multimediadaten
  - Filme
  - Überblendungen und Einspielungen gespeicherter Daten in Live-Multimedia
  - explizit
  - Beschreibungsmodell
    - parallel, sequentiell, unabhängig

## 4.1 Quellen der Asynchronität

- Verzögerung (Delay) auf dem Weg vom Produzenten zum Konsumenten
  - wegababhängig
  - stört besonders globale Synchronisation
- Jitter: Variation der Verzögerung



- Skew (Divergenz) zwischen Datenströmen



- Ankunft zu verschiedenen Zeiten
  - Präsentation zur gleichen Zeit
- Auch die Synchronisationsinformation selbst kann betroffen sein

- Entstehung im Endgerät:

- Betriebssystem

- Prozess-Scheduling

- Kritische Bereiche

- Interruptsperr

- Warteschlangen

- Latenz des Speichermediums (Festplatten)

- Verarbeitungspipelines (Kompression, Rendering)

- Busbelegung

- Virtueller Speicher

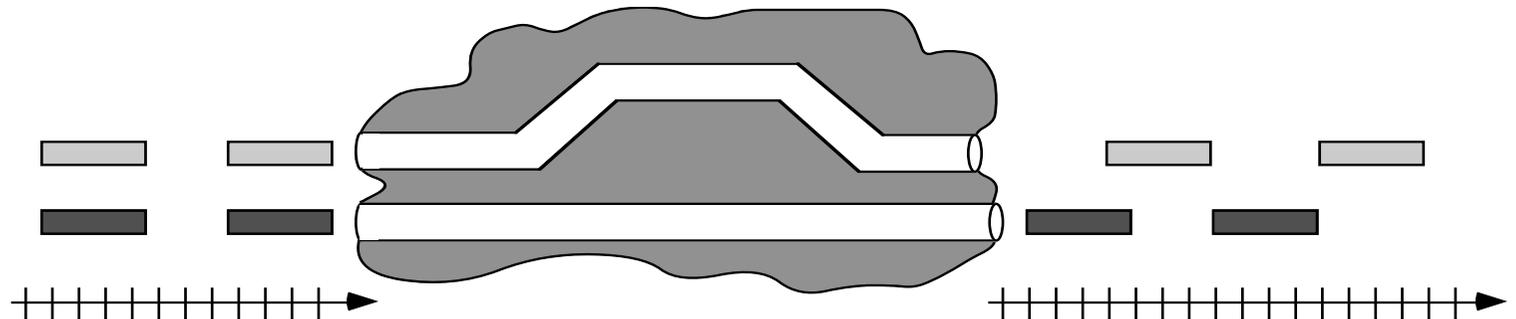
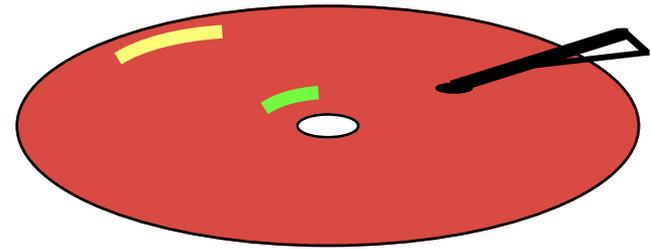
- Auf dem Übertragungsweg

- Netzknoten mit Store and Forward

- Paketverlust und Retransmission

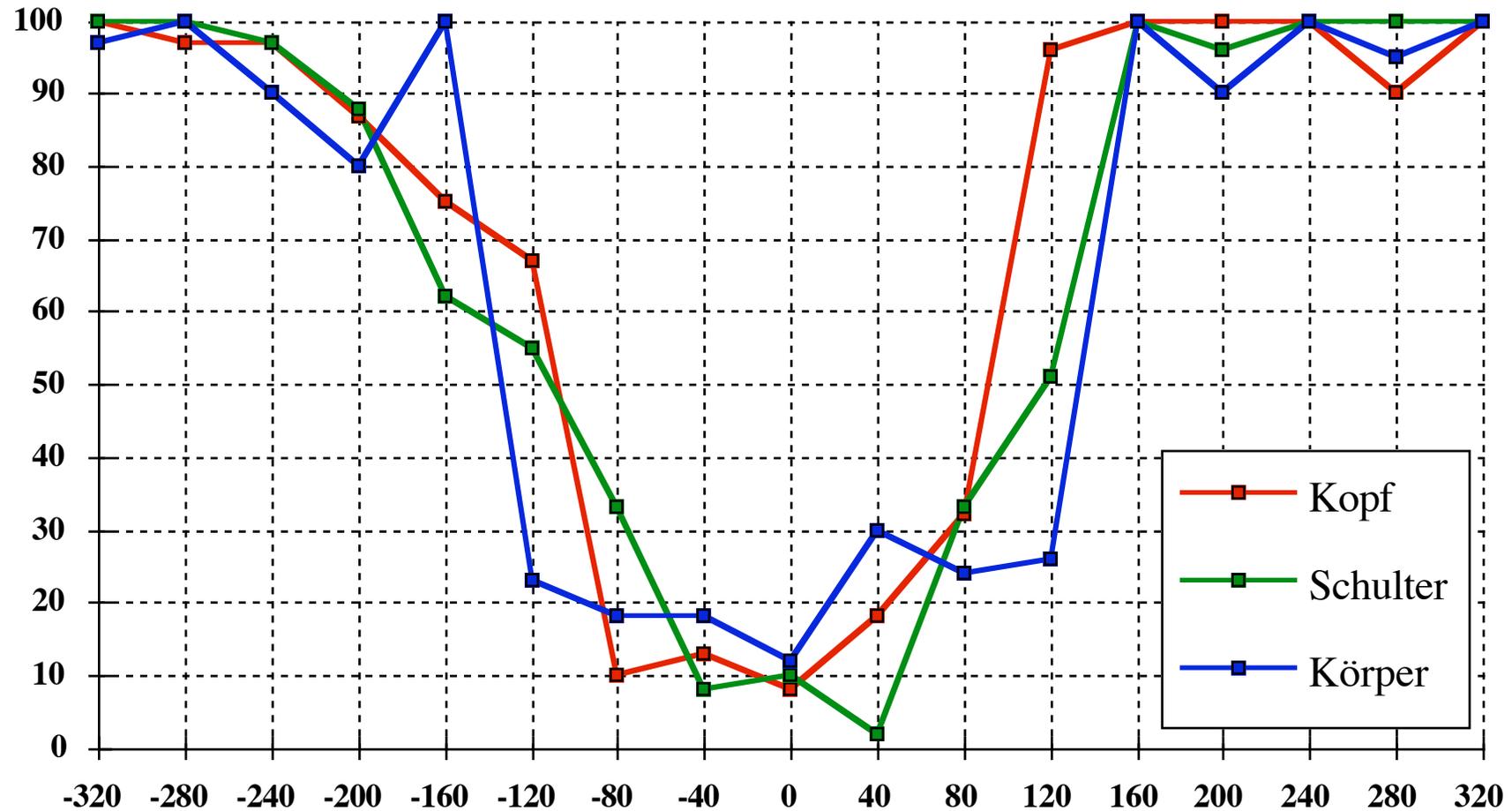
- Leitungen unterschiedlicher Laufzeit (ISDN ...)

- Bearbeitungsverzögerung in den Protokollen

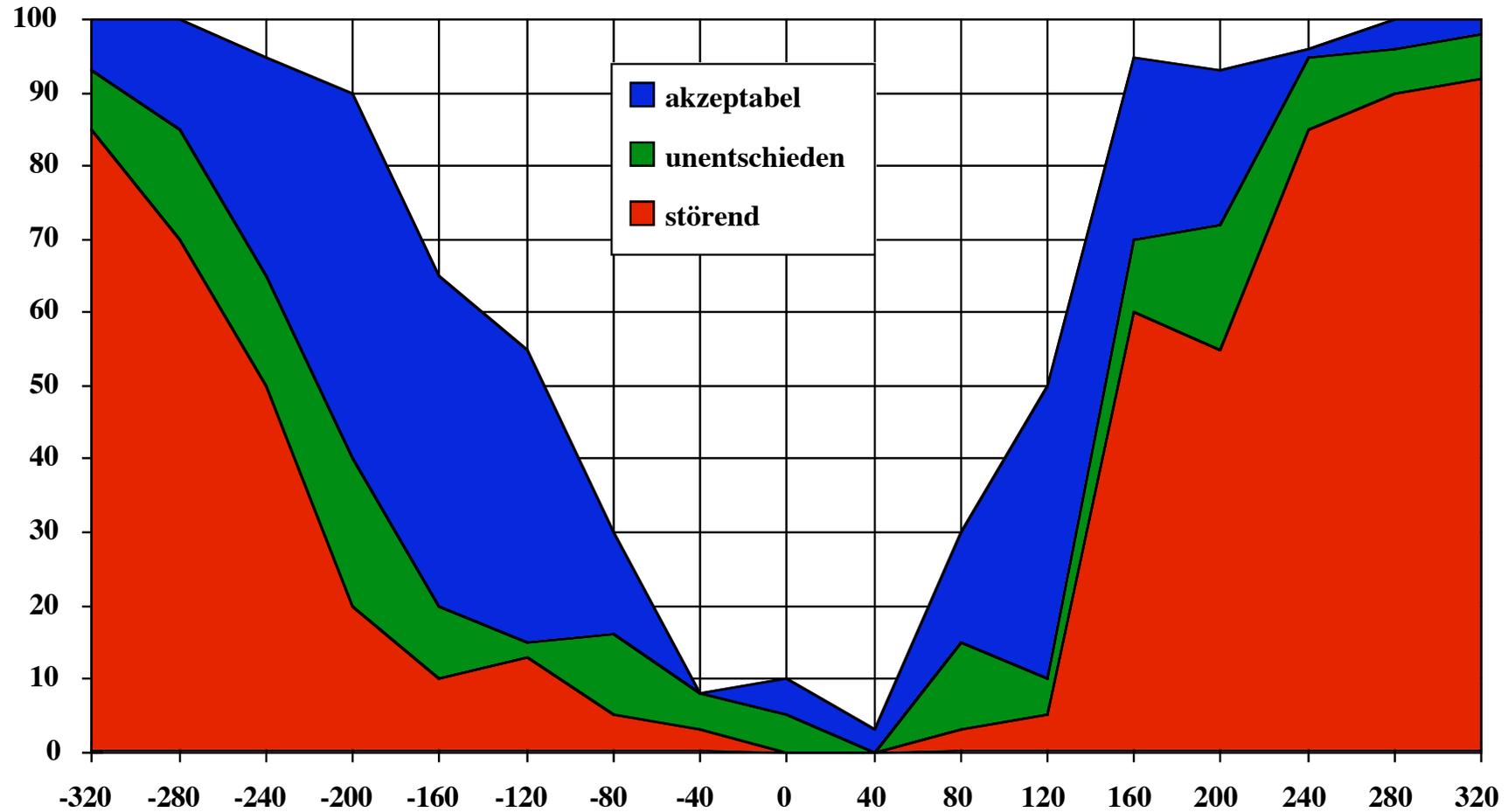


## 4.2 Wahrnehmung von Synchronitätsfehlern

- Empirische Bestimmung (IBM-ENC, R. Steinmetz, 1993)
  - ca. 100 Probanden
- Lippensynchronisation: Audio-Video

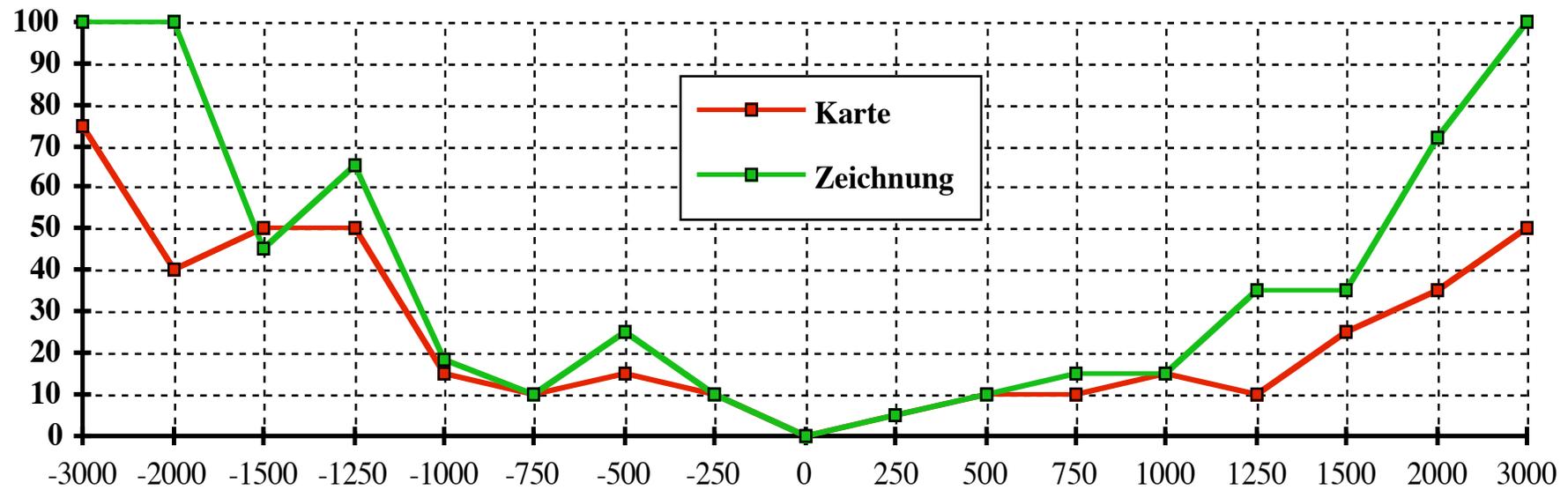
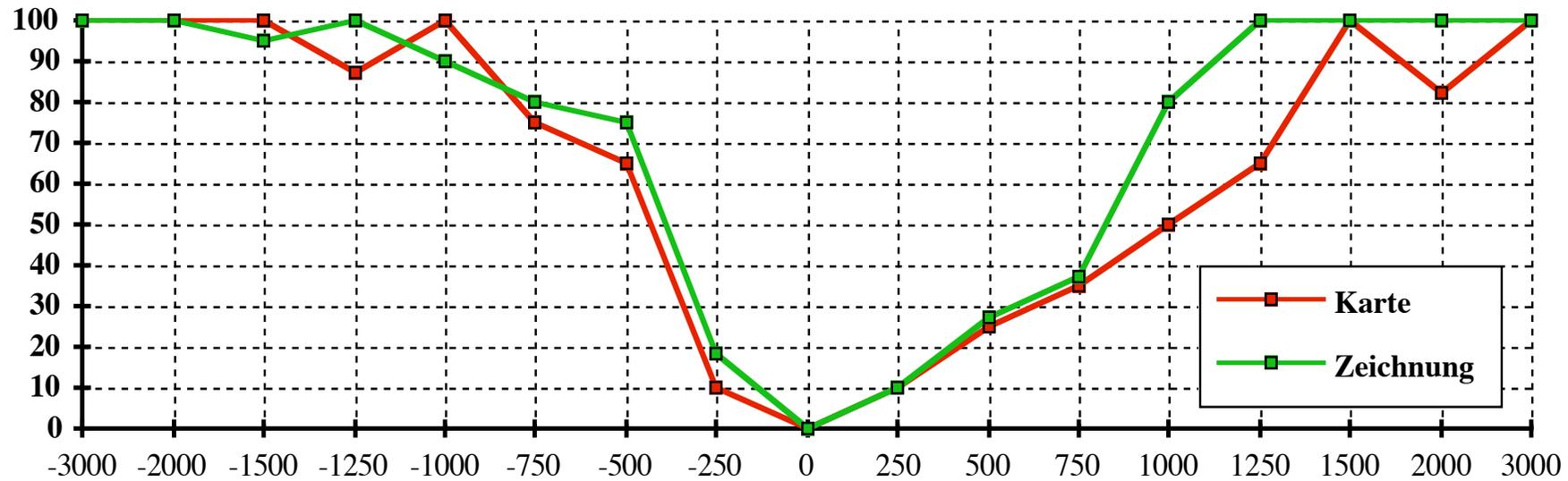


# • Akzeptanz



- $\pm 80$  ms unmerklich
- $-160$  ms  $<$  skew  $<$   $120$  ms: nicht störend
- Audio-Synchronität unabhängig von der Sprache

- Synchronisation Audio - Zeiger auf Grafik



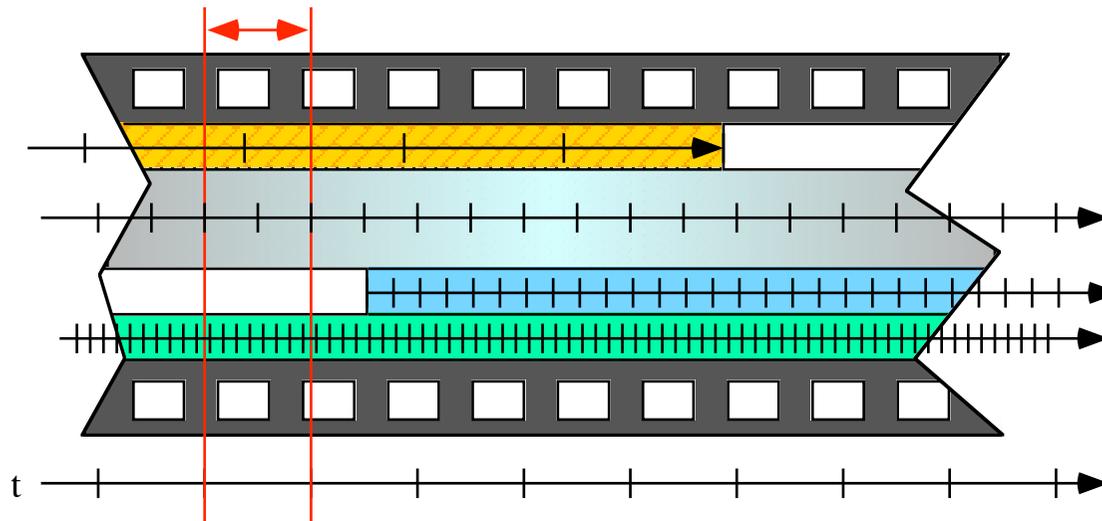
- Überblendung kritischer als Parallelpräsentation bei Text/Grafik und Video
- Grenzen der Wahrnehmung / Störung

Video	Animation	korreliert	-120	120
	Audio	lippensynchron	-80	80
	Grafik	überblendet	-240	240
		nebeneinander	-500	500
Text	überblendet	-240	240	
	nebeneinander	-500	500	
Audio	Animation	korreliert	-80	80
	Audio	stereophon	-11	11
		Dialog	-120	120
		Hintergrund	-500	500
	Grafik	eng (Musik mit Noten)	-5	5
		lose (Dia-show)	-500	500
	Text	korreliert	-240	240
Zeiger	korreliert	-500	750	

- Jitter auch kritisch in einem Datenstrom
  - Audio: >5 ms

## 4.3 Synchronisationsinformation

- Zeitangaben
  - Sample-bezogen
  - SMPTE (Society of Motion Pictures and Television Engineers)  
entwickelt von NASA  
Stunde:Minute:Sekunde:Frame:Bits (80 Bits/Frame)  
Linear Time Code (LTC) auf besonderer Spur  
Vertical Interval TC im vertical blanking  
Puls pro Halbbild, keine Werte
  - eventuell auf andere Spur (Medium) bezogen
- Synchronität zwischen Medienströmen

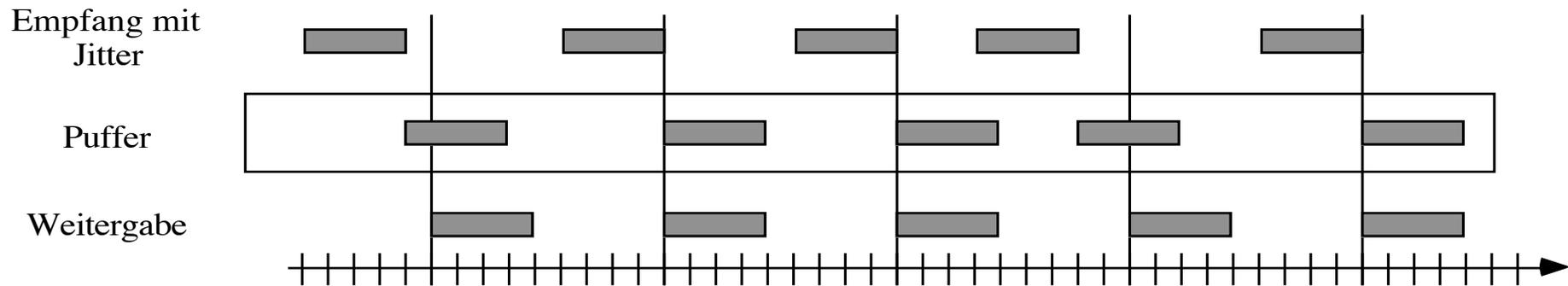


- Zeitkritische Dateneinheit  $m$  als Tripel
  - $m = (\text{Daten}, \text{Start}, \text{Dauer})$
  - Gültigkeit =  $[\text{Start}, \text{Start}+\text{Dauer})$
  - Medienstrom ist Folge zeitkritischer Dateneinheiten
- Zeitachsen
  - Auflösung 1 sec bis  $10 \mu\text{sec}$
  - Individuelle Zeitachsen pro Spur
    - Schnitt und Kombination verschiedener Spuren
    - Zeitangaben relativ zum Beginn einer Spur
    - Abbildung auf globale Zeitachse bei der Wiedergabe
- Koordinationsspur ( $\sim$ verbindung)
  - Zeiger, die in die Spuren zeigen
  - Kleine Datenmenge, Übertragung leicht
- Abhängigkeitsgraphen
- siehe QuickTime Movie-Format

- Zeit in verteilten Systemen
  - Muß nicht absolut richtig sein
  - Gleicher Wert bei allen beteiligten Systemen
  - Drift im  $\mu\text{sec}$  Bereich
  - Quarze arbeiten mit Genauigkeit  $10^{-6}$  => pro Stunde bis zu 4 msec
- Uhrensynchronisationsprotokoll
  - Network Time Protocol [Mills, RFC 1305]
- DCF77 (Physikalisch technische Bundesanstalt)
  - Auflösung 1 Sekunde
  - Fehler  $\ll 1\mu\text{sec}$
  - Laufzeit des Signales vom Sender zum Empfänger bekannt
- NAVSTAR GPS
  - Positionsermittlung in mobilen Einheiten
  - 22 Satelliten strahlen Zeit und Satelliten-Position aus
  - Endgerät ermittelt eigene Position aus Laufzeitunterschieden
  - Unabhängig von der Position des Endgerätes
  - Zeitfehler  $< 363$  ns

## 4.4 Mittel zur Synchronisation

- Verzögerungsvermeidung
  - auf dem Übertragungsweg
    - Resource-Reservierung: Bandbreite, Puffer
    - Protokolloptimierung
  - im Endgerät
    - Reservierung: Puffer, Zeit (CPU, ...)
    - keine Datenkopien (DMA, Zeigerübergabe)
- Jitterausgleich (Schadensbegrenzung)
  - Einfügen einer Basis-Verzögerung

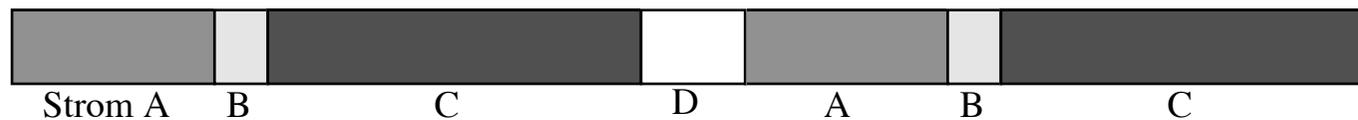


Berechnung der Delaylänge:  $\text{Max}(\text{Üb. Verzögerung}) + \text{Paketdauer}$

- Extrapolation bei fehlenden Samples kontinuierlicher Medien
  - Interleaving -> Interpolation möglich, MPEG ...
  - zu spät eintreffende Samples wegwerfen

- Multiplexen aller Medienströme in einen Datenstrom
  - Synchronisation auf Stromebene
  - Blöcke mit gleicher 'Zeitdauer'
  - Sender arrangiert Daten für optimale Präsentation

Empfängereigenschaften ausnutzen  
Zeitaufwand ...



- alle Ströme sind gleichermaßen von Verzögerungen betroffen
- Verschleiert Eigenschaften von Datenströmen

Burstdaten stören Isochrone Daten

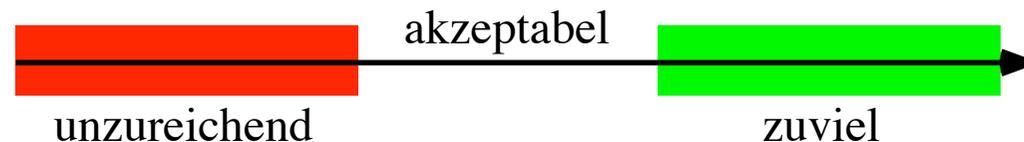
Nur Summen-QoS Anforderung möglich

Netzwerk kann keine vernünftige Resourcereservierung vornehmen

- Strategie zur Synchronisation
  - Resource Reservation , Scheduling

## 4.5 Resource Reservation

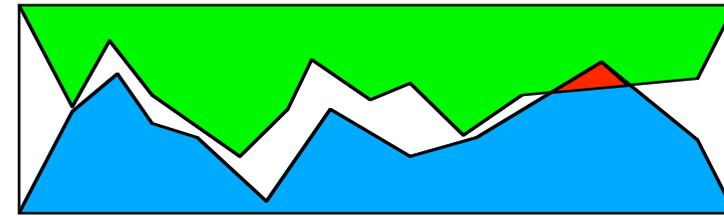
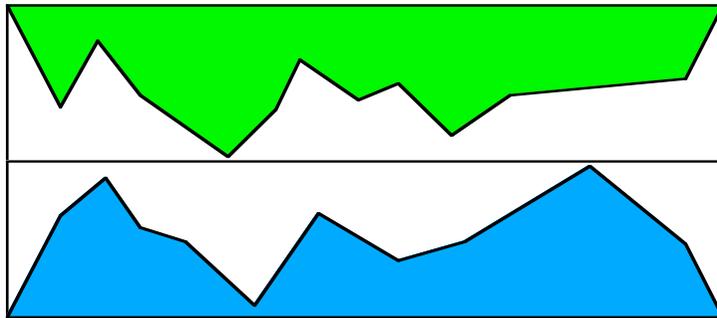
- Bearbeitungseinheiten für Medien
  - Prozessor
  - Speicher (RAM, Magnetplatten, CD, CD-ROM)
  - Spezialhardware (Kompression, Dekompression)
  - Präsentationsgeräte
  - Digitalisierer
  - Netzwerk und Netzwerkadapter
- Dienstegüte (Quality of Service, QoS)
  - Applikationsanforderungen <-> Systemkapazität
  - Best Effort oder Garantie
  - Garantie -> Reservierung, Planung
  - Multiplexeffekte verbessern Kosten/Nutzen



- Spezifikation: Interval gewünscht und minimal  
required QoS, desired QoS

- Garantierte Dienstegüte

- Obere Grenze für Last
- Pessimistische Annahmen Systemverhalten
- Sehr zuverlässig
- Überreservierung
- Schlechte Ressourcennutzung



- Statistische Dienstegüte

- Oberer Grenze für Last
- Mittlere Systembelastung
- Stochastische Ausnutzung der angeforderten Dienstegüte
- keine Garantie ...

- Ressourcen-Verwaltung

- Verwaltung des Mangels
- Optimierung des Systemdurchsatzes

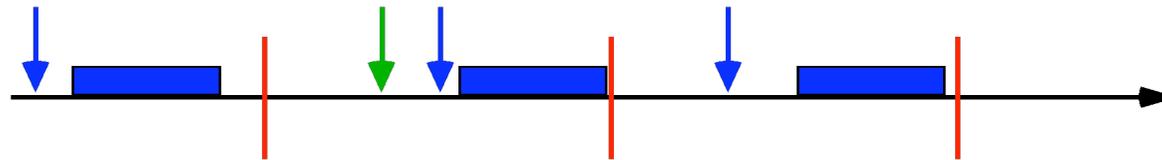
- Resource-Planung
  - Berechnung der nötigen Ressourcen abhängig von QoS
  - statische Zuteilung
  - entlang des Datenpfades
- Verhandlung
  - mit der Applikation
  - mit anderen Komponenten
  - Modifikation zur Laufzeit
- Dynamische Resource-Zuteilung (Resource Scheduling)
  - für jede Resource
  - steuert Bearbeitungsreihenfolge
  - Ereignisgesteuerter Prozesswechsel
  - Unterbrechung
  - überwacht Zuteilung (Policing)
- Prioritäten
  1. Multimediadaten mit garantierter Dienstegüte
  2. Multimediadaten mit statistischer Dienstegüte
  3. Andere Prozesse

## 4.6 Echtzeit und Scheduling

- Erweiterte Prozesskennzeichen:
  - Priorität
  - Zeitkritisch
  - Intervall des regelmäßigen Aufrufens (Gültigkeit)
- Interruptkennzeichen:
  - Maximale Verzögerung bis zur Bearbeitung
  - Minimale Bearbeitungsdauer
- Leichtgewichtige Prozesse
  - teilweiser Kontextwechsel beim Aufruf
  - besonders für ISR und Call-Backs
  - Einschränkung der aufrufbaren Betriebssystemdienste
  - Cache-Invalidierung bei MMU Umschaltung
- Problem Prioritäts-Umkehr
  - nicht unterbrechbare Prozesse mit niedriger Priorität
  - Prozess mit höherer Priorität wird gestartet
  - Priorität anderer Prozesse steigt während der Bearbeitungszeit
  - Unterbrechung des Ersten, Thrashing

- Scheduler mit garantierter 'Quality of Service'
  - behandelt zeitkritische Prozesse bevorzugt
  - garantiert Interruptbearbeitung innerhalb der verlangten Zeit
  - normale Betriebssystemprozesse haben niedrige Priorität
  - Zeitgarantien für Prozesse

Deadline = späteste Ausführungszeit  
 = erwartete Ankunft + max. Delay



höchste Priorität für Prozesse mit:  
 Daten präsent und Deadline erreicht

- Periodische (isochrone) Prozesse
- Feste Einplanung
- Jitter -> Resourcevergeudung

- Non-Preemptive Scheduling

- Prozesse nicht unterbrechbar



- Inhärent für viele Ressourcen
- Prioritätsumkehr möglich
- Weniger Prozesswechsel

- Preemptive Scheduling

- Prozesse durch andere Prozesse mit höherer Priorität unterbrechbar

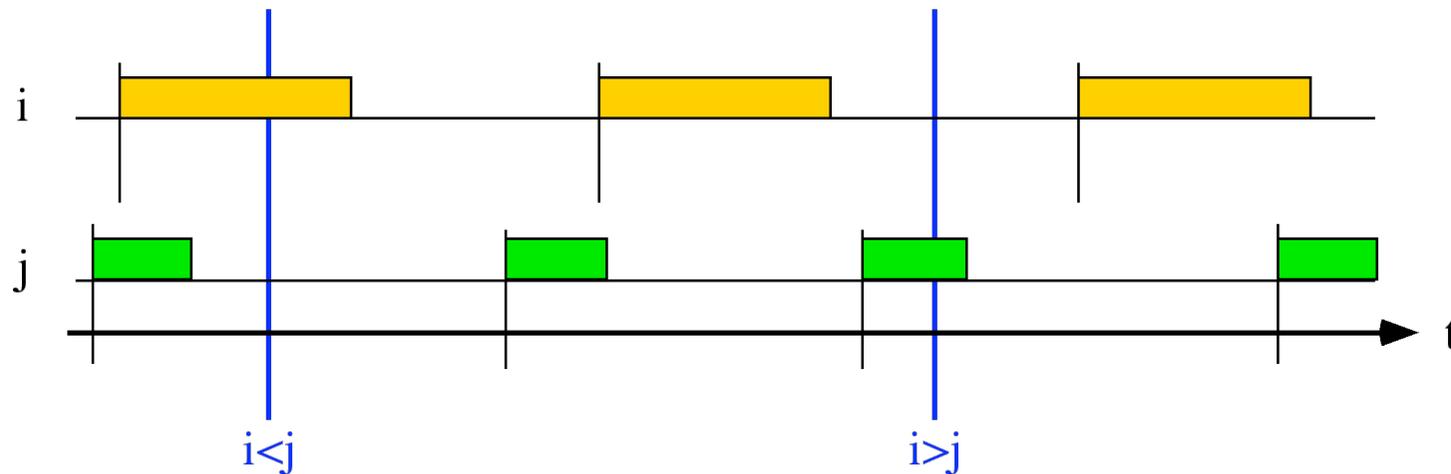


- Oft in OS vorhanden für CPU
- Prioritätsumkehr möglich
- Prozesswechsel häufig und teuer



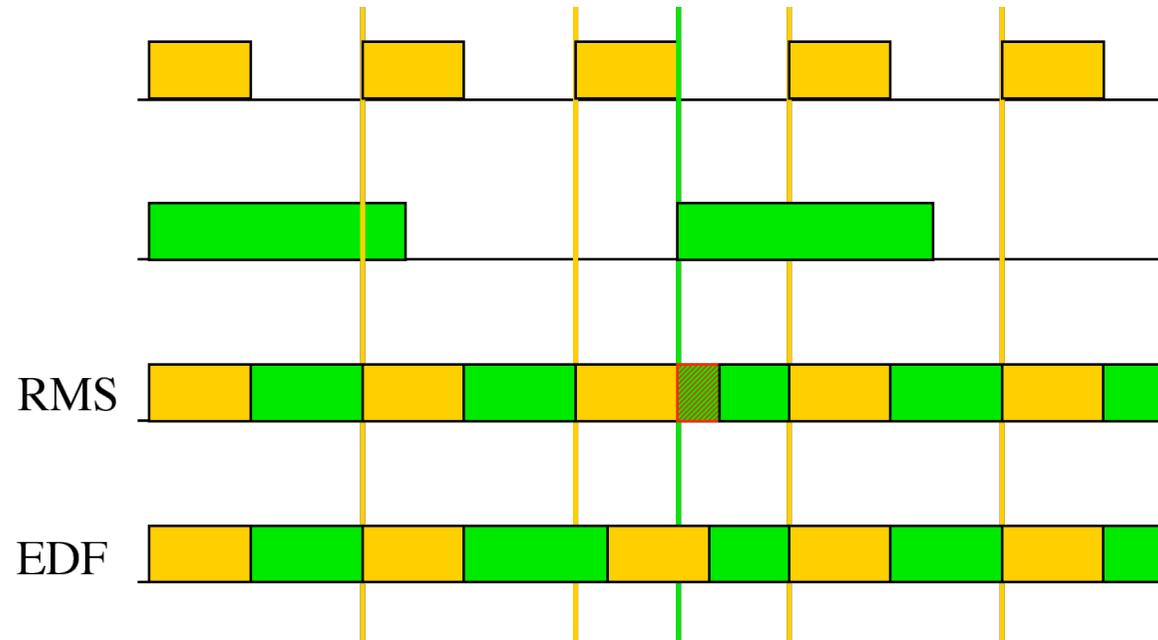
## • Deadline Based Scheduling

- dynamische Planung
- EDF: Earliest Deadline First
- Prozess mit kürzester Frist wird aktiviert
- neuer Prozess bereit -> Unterbrechung + neue Fristenauswertung
- Unterbrechbarkeit nötig



- Prioritäten ändern sich
- $O(n^2)$
- QoS-Planung wie raten-monotones Verfahren
- Variante für non-preemptive möglich, Planbarkeit schlechter

- Vergleich



- EDF reagiert flexibler: Jitter, ...
- auch in nicht-preemptiven Systemen
  - Prozesse Echtzeit-fähig
  - Sprachmittel z.B. in RTC++

```
within (deadline_duration) do  
  { do_something  
  };
```
- Real-Time Mach (CMU)

## 5. Ausblick

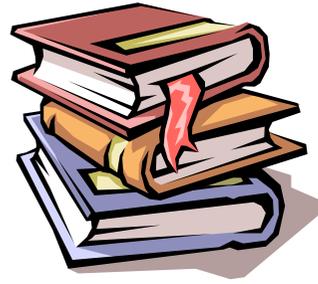
- Status quo
- Kommunikation in der realen Welt
  - sehen-und-kommunizieren
  - Reichweite der Sinne
- Soziale Protokolle
  - Aufdringlichkeit
  - Unterbrechung
- Verbindungskontrolle
  - anschauen und ...
  - implizites Filtern
- Dialogsteuerung
  - CSMA/CD
  - warten bis man dran ist
  - Wirkung beobachten



- Telephonie verändert sich
  - mobil und allgegenwärtig
  - flat rate
  - ständige Unterbrechung
- Empfangsbereitschaft
  - Ort
  - Tätigkeit
  - Gemütslage
- Verbindungssteuerung
  - Zahlen, Zahlen, ...
  - Telefonbuch, Anrufliste
  - lange Startzeit
  - explizit

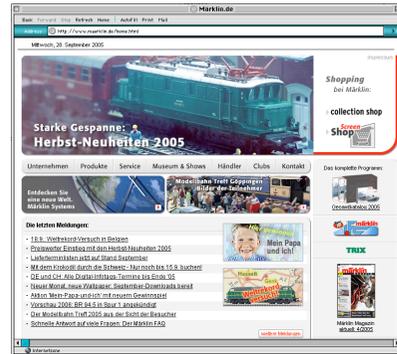
- Informationsdienste

- sehen-und-konsumieren
- anreizgetrieben
- Papier extrem billig
- Radio und TV



- Medien

- Text
- Diagramme und Bilder
- Audio, Video



- Suche: Mustererkennung

- Steuerung

- physisch
- Nehmen-Lesen-Weglegen
- Programmwahl, Zap



- Broadcast

- Konsumtive Multimediadienste

- computerbasiert
- 'ubiquitär'
- integriert in alle Tätigkeiten
- ständig eingeschaltet

- Steuerung = Suche

- >1000 Programme?
- Links, Linklisten
- Programmzeitung ~ Yahoo
- Suchmaschinen

- Bewertung

- Bsp: blogs
- Vertrauenswürdigkeit
- Popularität

- Spiele in der realen Welt
  - teilnehmen
  - evtl. bewegen und anstreng
  - Reichweite der Sinne
- Interagieren
  - Brettspiele, Quiz
  - Wettkampf
  - Mannschaftssport
- Steuerung
  - verabreden, teilnehmen
  - Regeln



- Computerspiel
  - virtuelle Welten
  - jederzeit, mühelos
  - online-Spiele
- Verfügbarkeit
  - Ort, Zeit
  - Tätigkeit
  - Gemütslage
- Anonyme Interaktion
- Steuerung
  - Zahlen, Zahlen, ...
  - Chatroom, ...
  - lange Startzeit
  - explizit

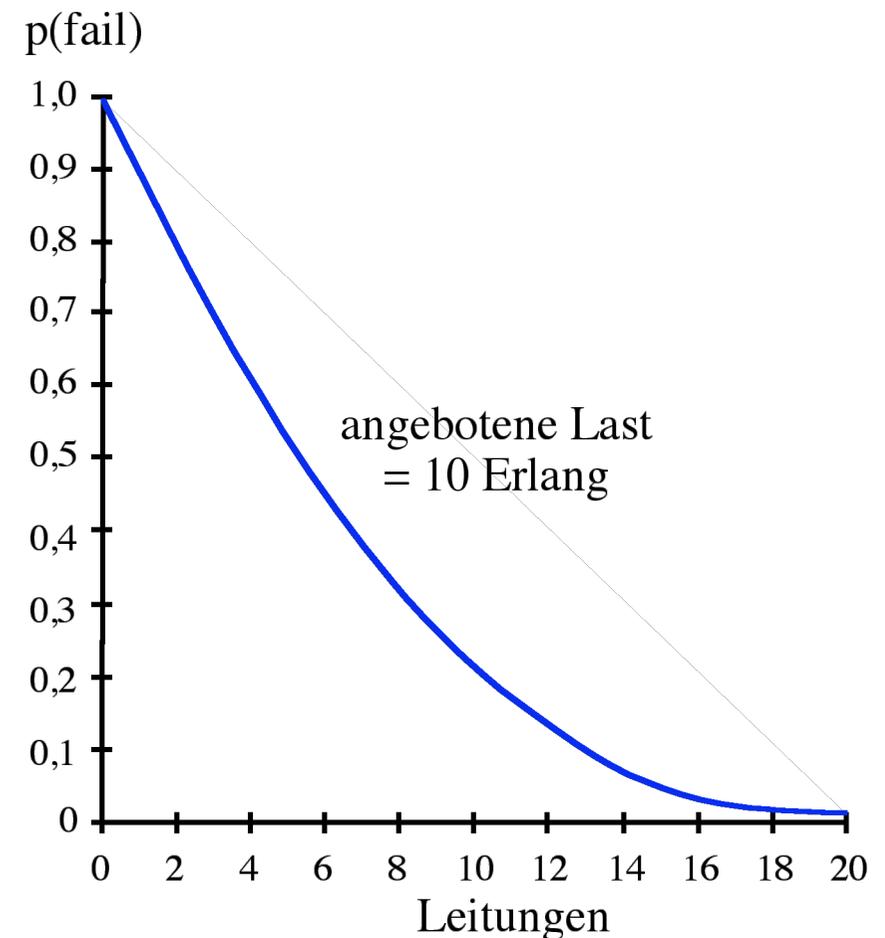
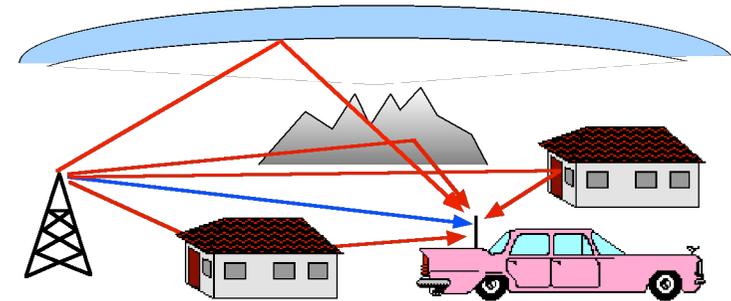
- Öffentliche Vision
- Größer
  - höhere Auflösung (HDTV)
- Überall
  - UMTS, 4G, ...
  - W-LAN, Hotspots, ...
- Videoiger
  - Videotelefonie, Video-Mail
  - Handy-TV
- Kontrollierter
  - Firewalls, Inhaltsfilter, ...
  - im Namen der Sicherheit
- Konsumorientiert
- Gewinnorientiert
  - Bezahlung pro Konsumvorgang
  - Detailabrechnung vs. flat-rate



- Videotelefon: Live-Bild selten präsentabel
  - Beleuchtung, Kameraführung
  - Kleidung, Aufräumen, Schminken, ...
- Nischen-Märkte
  - Reisezeit >> Sprechzeit
  - Großeltern-Abwehr
  - Konferenzstudios (Polycom etc)
- Mobiles Videotelefon
  - 'animierte Briefmarken'
  - Display+Kamera+Mikro+Lautsprecher
- Produktionskosten hoch
  - Drehbuch
  - Aufnahme, Schnitt
  - Kompression



- Funkkanal stör anfällig
  - zelluläres System begrenzt Sendeleistung
  - mehr Teilnehmer => mehr Störungen
- Durchsatz der Funksysteme beschränkt
  - raffinierte Fehlerkorrektur nötig
  - geteilte Kapazität
  - multimediale Inhalte stark komprimiert
- Realistisches Nutzungsmuster
  - viele Benutzer
  - kurze Benutzungsdauer
- Handy-TV
  - realistisch als Broadcast-Dienst
  - Satelliten (DMB) und DVB-H



- Flat rate und Überall-Telefonie
  - Telefon klingelt ständig
  - Verbindungen werden länger
  - neue Wege beim Anruf-Management
- Erweiterung der 'lokalen' Mechanismen
  - 'Hans ---- wann fahren wir los?
  - 'ich möchte mit dem Verkäufer vom Montag sprechen'
- Telefone
  - finden Nummern wie Bookmarks und Google
  - nehmen Anrufe entsprechend Situation an
  - Dauerverbindungen mit Aktivierung/Deaktivierung
- 3D-Szenen mit Avataren
  - Blickkontakt
  - Video kann Illusion verbessern
  - Anordnung der Stimmen im Raum

- Präsentations-Hardware

- Projektion, Audio
- HUD, auch portabel
- 3D-Video, 3D-Sound



- Störungen managen

- Präsenz in Kommunikationsräumen
- Bewertung und Kommunikation von Anrufparametern

- Assoziation von Namen/Konzepten und Nummern

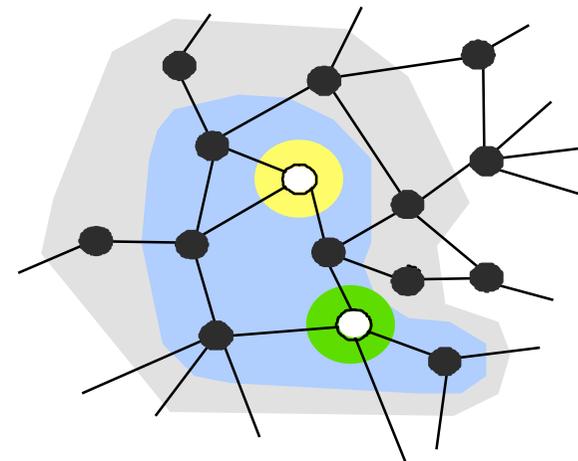
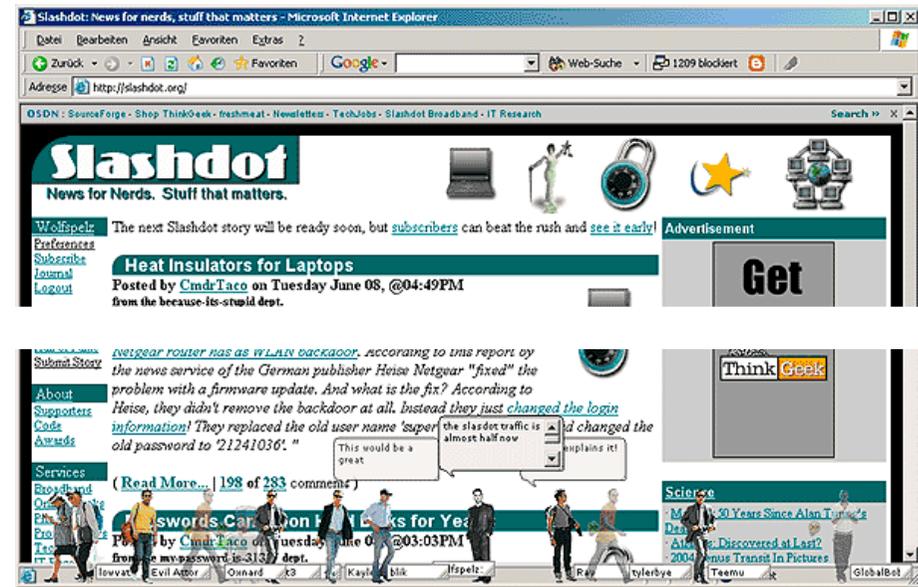
- Spracherkennung
- Durchsuchen aller Kommunikationsvorgänge [Gray: Personal Petabyte]
- Suchen im Web und anderen Verzeichnissen [Bush: Memex]

- Verbindungen besser managen

- Präsenzinformation kommunizieren und visualisieren
- Dauerverbindungen



- Reale Welt
  - Entfernung [Meter]
  - Soziale Metriken ...
- WebSpace [CoBrow, 1996]
  - Zeit + Ort: Link-Distanz
  - keywords und content
  - Project BuddySpace
- Präsenz = Ort + Ansprechbarkeit
- Telefon
  - Kommunikationssituation
  - Arbeitssituation
  - Kalender in PC und PDA
- Mobiltelefonie
  - fahren, gehen, rennen, ...
  - physischer and mentaler Stress



- Integration Dialog- und Konsumdienste
  - Dialog im Web => CoBrow
  - Suche statt Telefonbuch
  - Commpage statt Homepage
- Kommunikationsraum Holodeck
  - Ziele als Avatare
  - natürliche Sprachsteuerung
  - Dialogsteuerung wie in der realen Welt
  - Blickkontakt, Ansprechen, ...
  - Secondlife
- Kommunikationswohnung
  - Büro, Spiel, Unterhaltung, Bibliothek, ...
  - Metaverse [Neal Stephenson: Snow Crash]

