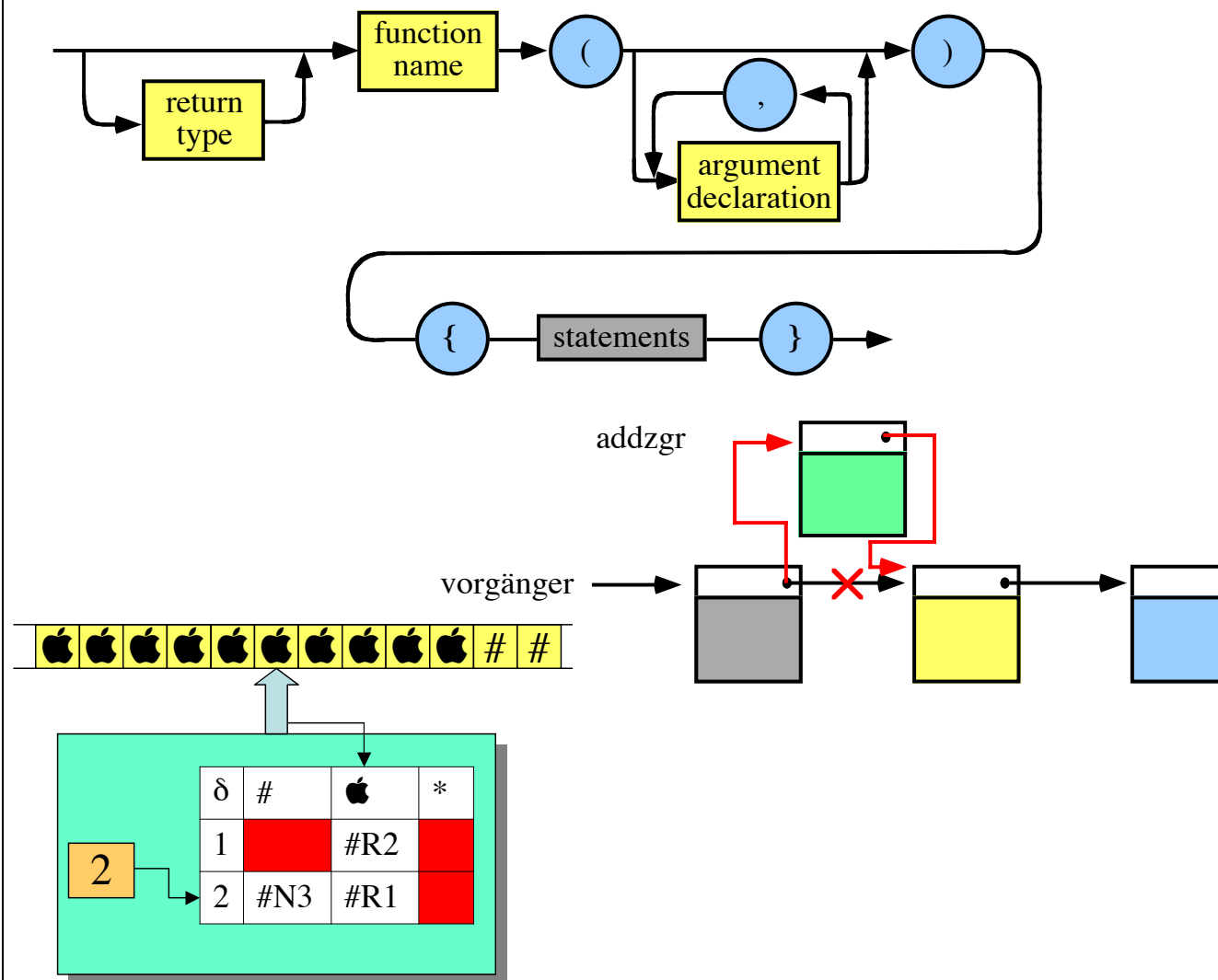


Grundlagen der Informatik



KAPITEL 1

Gebiet und Vorlesung

- Informatik - die Wissenschaft des Kommunikationszeitalters
- Vision
- Gebiete
- Formales zur Vorlesung
- End User License Agreement
- Literatur
- Inhaltsübersicht

*I think there is a world market for maybe five computers.
[T.J. Watson, Chairman, IBM, 1943]*

Informatik = Information + Automatik

Die Wissenschaft von der automatischen Verarbeitung von Informationen

- Information
 - lateinisch: Auskunft, Belehrung, Benachrichtigung
 - umgangssprachlich: Kenntnis über Sachverhalte und Vorgänge
 - inhaltliche Bedeutung eines Sachverhaltes (Semantik)
- Daten
 - Informationen, die zum Zweck der Verarbeitung eine bestimmte, vorher vereinbarte Darstellungsform (Syntax) haben
 - kontinuierliche Funktionen: "analoge Daten"
 - oder Zeichen: "digitale Daten"
- Automatik (griech.): selbsttätiger Ablauf
- Rechnen?

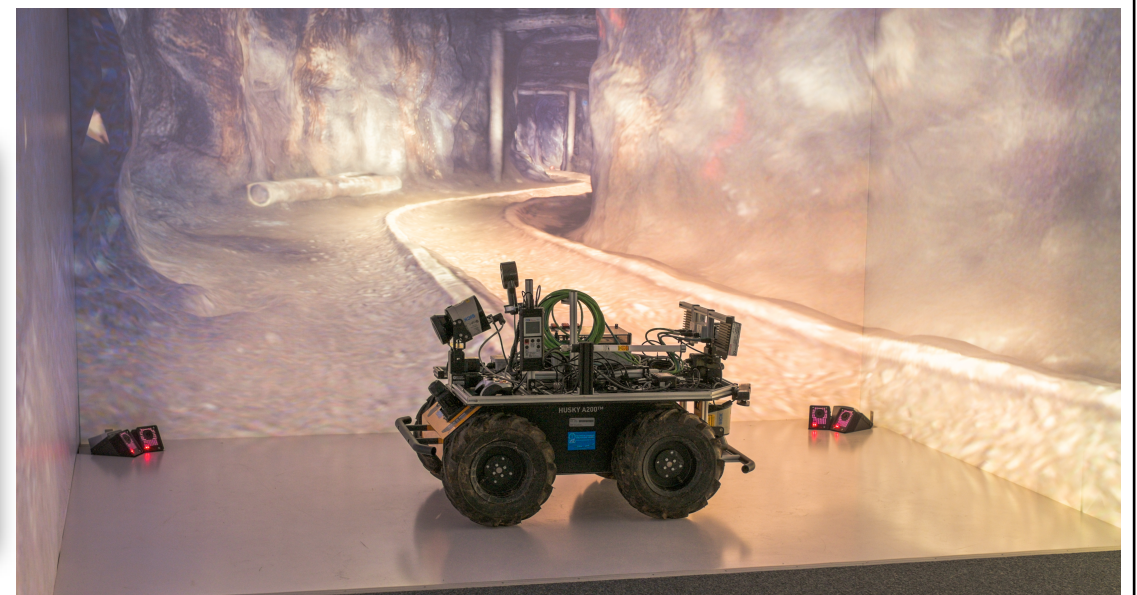
- Aufgabe der Informatik
 - Erforschung der grundsätzlichen Verfahrensweise der Informationsverarbeitung
 - Entwicklung allgemeiner Methoden
 - Anwendung in den verschiedensten Bereichen
- Informatik als Strukturwissenschaft
 - Struktur und Eigenschaften von Informationen
 - Struktur und Eigenschaften von Informationsverarbeitungssystemen
 - heute elektronische Datenverarbeitungsanlagen
 - Methoden: Analyse, formale Beschreibung, Konstruktion
- Informatik ist Ingenieurwissenschaft
 - Computer und Zubehör konstruieren
 - Verfahren und Algorithmen entwerfen
 - Programmieren
 - Pflegen und warten
 - aber: D. Knuth: The *Art* of Computer Programming

Vision

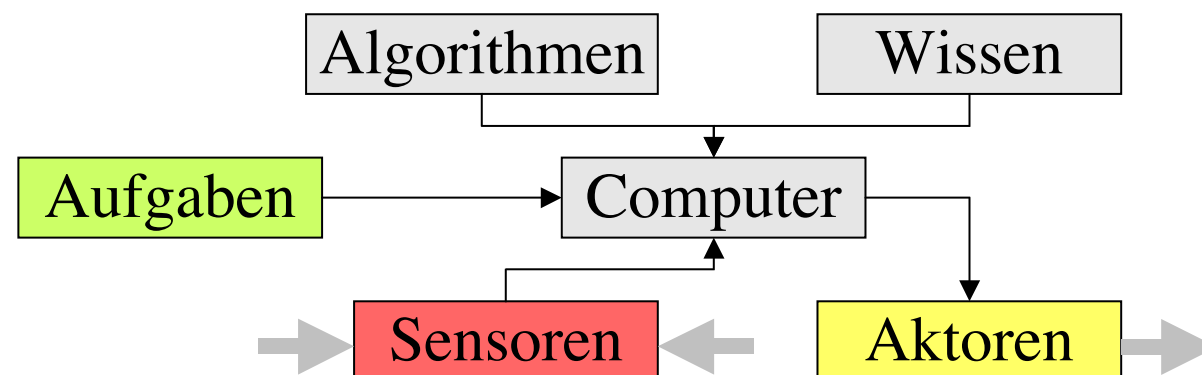
- Memex
 - Internet + Web
 - Apple, 1988: Knowledge Navigator
- Holodeck
 - ca. 1990: Vision in Star Trek: The Next Generation
 - Neal Stephenson, 2000: Snowcrash
 - ab 2004: 3D-Welten WoW, SecondLife, ...
 - immersive 3D-Interfaces noch teuer -> X-Site
- Künstliche Intelligenz (AI)
 - Google
 - IBM: Deep Blue, Watson

Vannevar Bush, 1945: Memex - a device in which one stores all his books, records, and communications, and which is mechanized so that it can be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.

Alan Turing, 1950: At the end of the century, the use of words and general educated opinion will have changed so much that one will be able to speak of "machines thinking" without expecting to be contradicted.

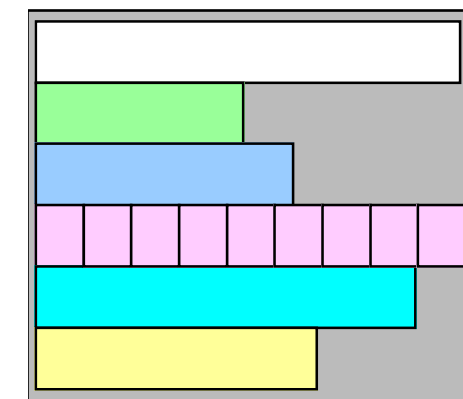
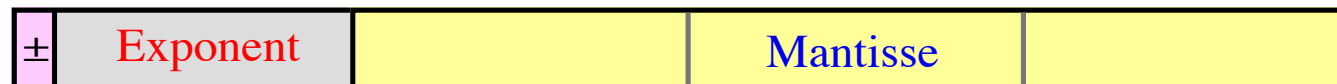
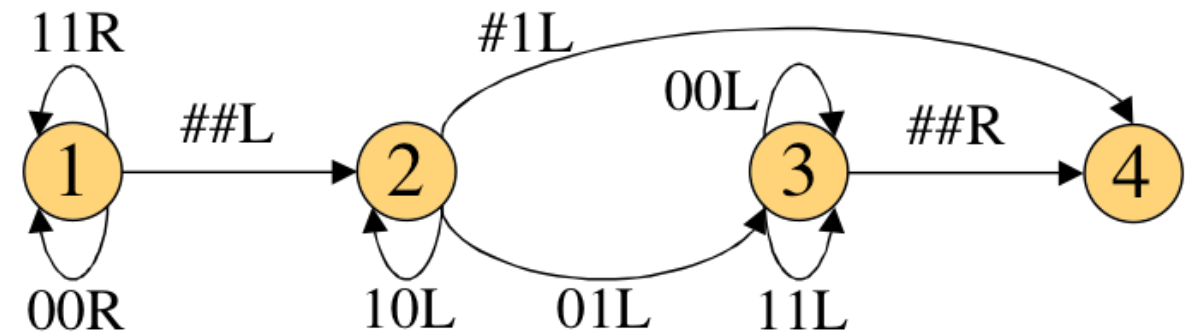


- Automobile
 - Verteilter Computer mit Antrieb und Rädern
 - Transportleistung
 - Navigation
- Google Car
 - ... bauen kein Fahrzeug sondern den Fahrer [Dolgov]
 - Lernen im Verkehr
 - Bildverarbeitung, Klassifikation
 - Statistische Verfahren
 - Neuronale Netze
- Roboter - "Freund und Helfer"
 - Actoren, Sensoren, Energieversorgung
 - Computer
 - co-operieren, schwieriges Gelände, ...
 - Autonomie



Gebiete der Informatik

- Theoretische Informatik
 - Berechenbarkeit
 - Aufwand für Berechnung
 - Beweisbarkeit von Programmen
 - Entscheidbarkeit
 - formale Methoden
 - Sprach-Klassen
- Algorithmen und Datenstrukturen
 - Formeln, Ablauf, Sequentialisierung
 - Daten darstellen und speichern
 - Zahldarstellung



- Programmieren
 - vom Problem zum Programm
 - ingenieurmässiges Gestalten (kreativ und systematisch)
 - Software Engineering
 - Programmiersprachen und Compiler
 - Objekte, Patterns und Frameworks
- Betriebssysteme
 - Abstraktion von Geräten und Computer-Teilen ("APIs")
 - Koordination der Programme
 - gerechte Verteilung der Ressourcen (Zeit, Platz, ...)
 - Schutz und Sicherheit
- Rechnerarchitektur
 - vom Transistor zum Maschinenbefehl
 - Speicher, Festplatte, Eingabe, Ausgabe, ...
 - Parallelrechner

```
enum Sys {case none, Elektro, Diesel, Dampf}

class Lok : Fahrzeug {
  var leistung = 0, last = 0
  var prinzip = Sys.none

  init(aName : String, pwr : Int, load : Int) {
    prinzip = Sys.none
    leistung = pwr; last = load
    super.init(aName: aName, typ: Fz.Lok)
  }

  func add (zug : Fahrzeug) -> Bool{
    if last < zug.cmpwght() {return false}
    else {super.add(zug); return true}
  }
}
```

There is no substitute for speed [E. McCreight, 1988]

- Technische Kommunikation

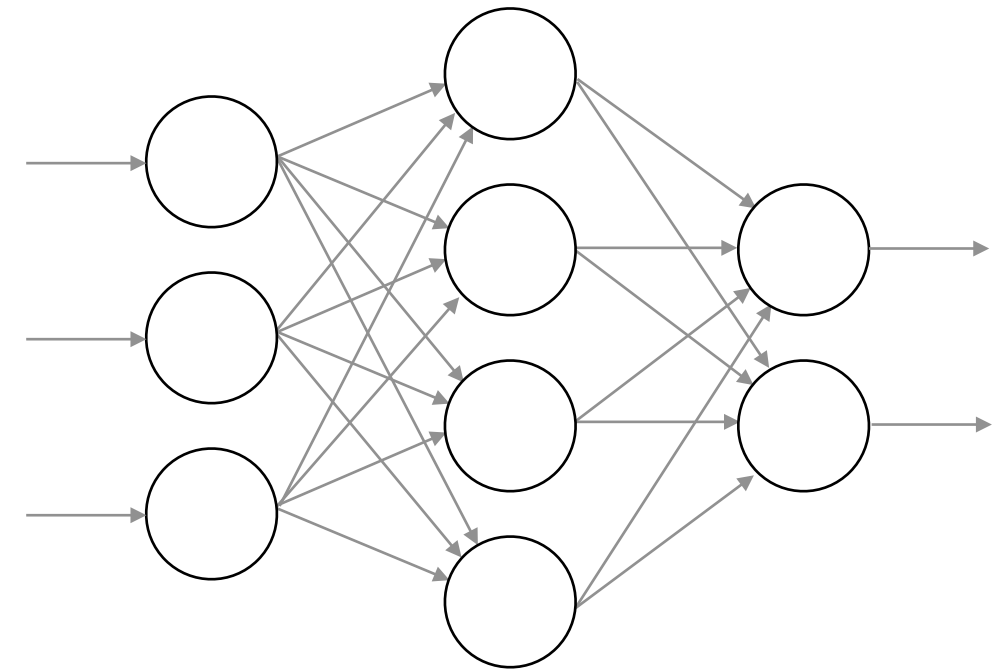
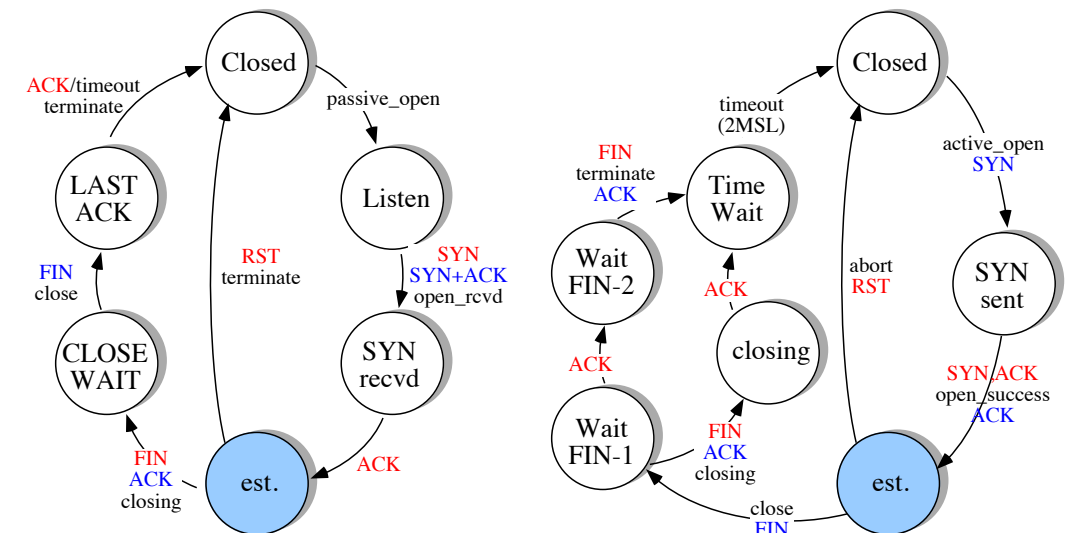
- Signalübertragung und -Vermittlung
- Protokolle
- Kompression
- Rechnernetze, Internet, ...

- Künstliche Intelligenz

- Lernen und Inferenz
- z.B. neuronale Netze, Support Vector Machine, ...

- Anwendungen

- Bürosoftware
- Datenbanken und Informationssysteme
- Multimedia
- Kommunikationsdienste (Telefon, TV, WWW, ...)
- CAD - Computer Aided Design
- Simulation
- Spiele
- Steuerung
- Robotik



Formales

Dramatis Personae:

Prof. Dr. Konrad Froitzheim: frz@tu-freiberg.de

Professur Betriebssysteme und Kommunikationstechnologien

MSc. NC Ben Lorenz

Dipl.-Math. Martin Reinhard

MSc. AI Jonas Treumer

Vorlesungsunterlagen (kein Skriptum):

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

kostenlos im itunes store: Konrad Froitzheim suchen



Diese Unterlagen zur Vorlesung sind weder ein Skriptum noch sind sie zur Prüfungsvorbereitung ausreichend.

Ohne das gesprochene Wort in der Vorlesung sind sie unvollständig und können sich in Einzelfällen sogar als irreführend erweisen.

Sie dienen dem Vortragenden als Gedächtnisstütze und den Hörern als Gerüst für ihre Vorlesungsnotizen.

Zur Prüfungsvorbereitung ist die intensive Lektüre der Fachliteratur unumgänglich.

Die Unterlagen werden während des Semesters noch gändert oder ergänzt.

- Übungen
 - Übungsleiter Assistenten
 - Vertiefungen und praktische Aufgaben
 - Fragen vorbereiten
 - fakultative Programmieraufgaben
- Klausur
 - ca. 100 Punkte
 - 6-8 Aufgaben
 - Wissen und Fähigkeiten
 - bestanden mit ca. 40 Punkten
 - 1.0 ab ca 95 Punkte

Klausur Grundlagen der Informatik

- Bearbeitungszeit: 120 Minuten
- Es sind **keine** Unterlagen (Bücher, Vorlesungsmitschriften, Übungen), Taschenrechner, Mobiltelefone (ausschalten und in die Tasche legen) sowie Computer jeglicher Art erlaubt
- Legen Sie beschriebene Lösungsblätter abgedeckt ab
- Zwischenschritte für die Berechnungen/Umstellungen sind mit anzugeben
- Die Klausur ist mit Aufgabenzetteln abzugeben
- Achten Sie bei Programmieraufgaben auf Sauberkeit und Sicherheit des Quellcodes (Fehlerüberprüfungen etc.).
- Lassen Sie ausreichend Platz bei Programmieraufgaben. Oftmals müssen Zeilen eingefügt werden. Größere (Teil-)Aufgaben sollten unbedingt auf einzelne Seiten platziert werden.

Lösen Sie alle Aufgaben!

Aufgabe 0 (1 Punkt)
Schreiben Sie Ihre Matrikelnummer und Studiengang auf jedes Blatt das in die Bewertung einbezogen werden soll! Ergänzen Sie diese Angaben auch auf der letzten Seite dieser Aufgabenzettel.

Aufgabe 1 - Zahlensysteme (12 Punkte)

- (1 Punkt) Was ist 2^7 in dezimaler Schreibweise?
- (4 Punkte) Addieren Sie die folgenden Zahlen $6F_h$, 01001101_b , 25_d schriftlich binär. Geben Sie das Ergebnis in hexadezimaler Form an.
- (2 Punkte) Multiplizieren Sie die Dualzahlen 101101 und 110 schriftlich, d.h. mit Lösungsweg und im Binärsystem.
- (5 Punkte) Geben Sie für die Dezimalzahlen 0 und -27 alle zugehörigen 8-Bit-Darstellungen im Einer- und im Zweierkomplement an.

Aufgabe 2 - C Programmierung (20 Punkte)

2.1 (2 Punkte) Im Quellcodefragment rechts ist eine Funktion gegeben. Beschreiben Sie in 1-2 Sätzen die Wirkung und den Rückgabewert der Funktion.

2.2 (3 Punkte) Ändern Sie die Funktion so ab, dass eine gültige Liste (A->B->C->NULL) umgekehrt (C->B->A->NULL) und als Ergebnis geliefert wird.

2.3 (3 Punkte) Ändern Sie die Struktur `node` so ab, dass Messwerte einer Geothermiebohrung gespeichert werden können. Ein Messwert besteht aus den Attributen Bohrtiefe, Temperatur und dem Datum der Messwerterfassung.

2.4 (12 Punkte) Implementieren Sie die Funktion `node_t* read_node(FILE *f)`. Sie soll einen einzelnen Messwert erzeugen und die Daten für die Strukturattribute aus der übergebenen Datei `f` auslesen. Die Datei ist bereits geöffnet und gültig. Sie können davon ausgehen, dass die Datensätze zeilenweise (mit max. 256 Zeichen Länge) und die Attribute kommasapariert in der

```
typedef struct node {
    struct node *next;
} node_t;
node_t* foo(node_t *head)
{
    node_t *prev = NULL;
    node_t *cur = head;
    while(cur->next != NULL) {
        node_t *tmp = cur->next;
        cur->next = prev;
        prev = tmp;
    }
    return cur;
}
```

Literatur

Conway, J., Hillegass, A.: iOS Programming; 2012.

Eernisse, M.: Build your own AJAX web application; Sitepoint, 2006.

Ferguson, N., Schneier, B., Kohno, T.: Cryptography Engineering, Wiley, 2010.

Gamma, E. et al: Design Patterns; 1994.

Goll, Bröckl, Dausmann: C als erste Programmiersprache; Teubner, 2003.

Kernighan, B., Ritchie, D.: The C Programming Language; 1988.

Knuth, D., E.: The Art of Computer Programming; 1973.

Post, E.: Real Programmers Don't Use PASCAL; Datamation, Vol. 29, Nr., 1983

Schöning, U.: Theoretische Informatik - kurzgefaßt; Spektrum, 1995.

Sedgewick, R.: Algorithms in C; Addison Wesley, 1998 ...

Wirth, N.: Algorithmen und Datenstrukturen; Teubner, 1983 ...

Inhaltsübersicht

1. Einleitung

...

2. Information und Daten

2.1 Informationsbegriff

2.2 Zahldarstellung

2.3 Rechnen

2.4 Datentypen

3. Vom Problem zum Programm

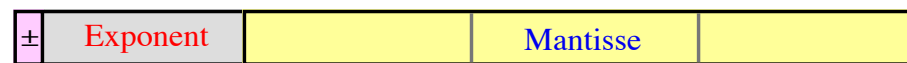
3.1 Idee und Analyse

3.2 Sequentialisierung

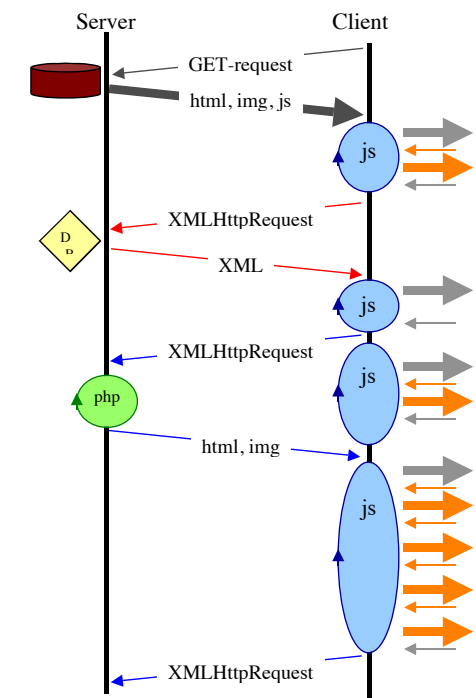
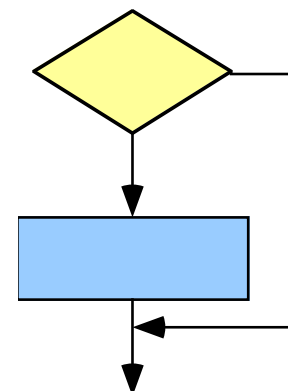
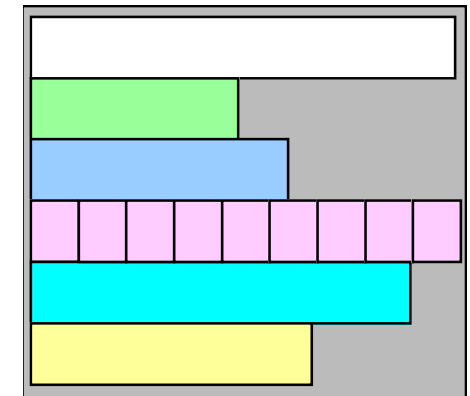
3.3 Programmieren

3.4 Systematische Fehlersuche

3.5 Reaktives Programmieren



```
int square (int num)
{
    int result;
    result = num * num;
    return result;
}
```



4. Algorithmische Komponenten

4.1 Sortieren

4.2 Listen

4.3 Speicherverwaltung

4.4 Rekursion

4.5 Objekte, Patterns und Frameworks

5. Betriebssystem: Abstrahieren und Koordinieren

5.1 Verteilung der Ressourcen

5.2 Struktureller Aufbau am Beispiel iOS

5.3 App Programmieren

6. Rechnerarchitektur: Vom Transistor zum Programm

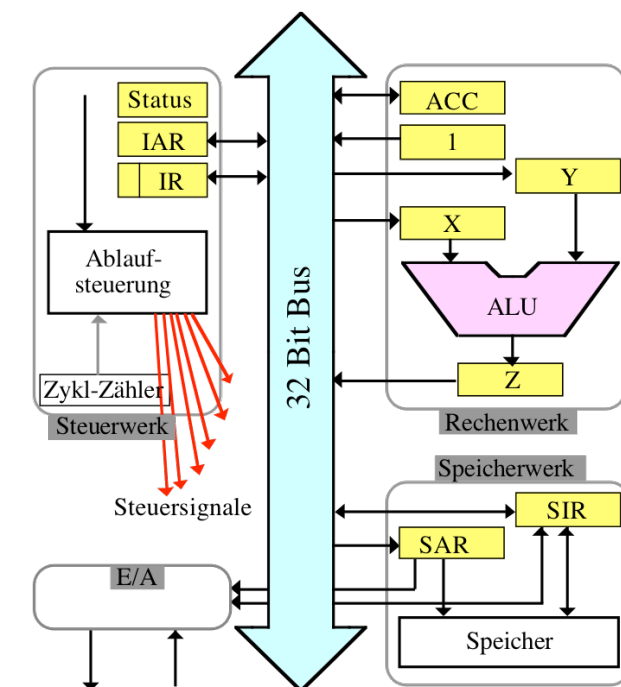
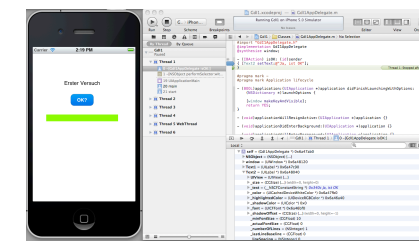
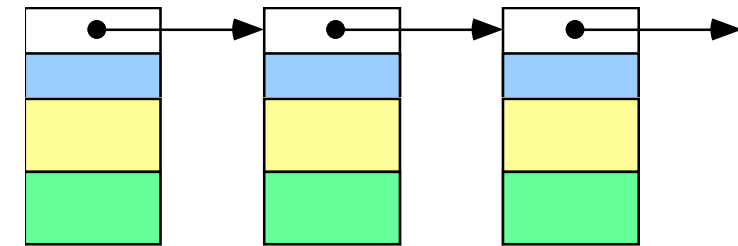
6.1 Transistoren, Gates

6.2 Rechnen und Speichern

6.3 Funktionseinheiten: Speicher, Prozessor, Bus, ...

6.4 Maschinenbefehle

6.5 Compiler



7. Computer, Internet und Sicherheit

7.1 Szenarien

7.2 Werkzeuge

8. Theoretische Informatik

8.1 Automaten und formale Sprachen

8.2 Berechenbarkeit

8.3 Komplexitätstheorie

9. Anwendungsprogramme

9.1 Datenbanken

9.2 Bürosoftware

9.3 Medienbearbeitung

10. Medien

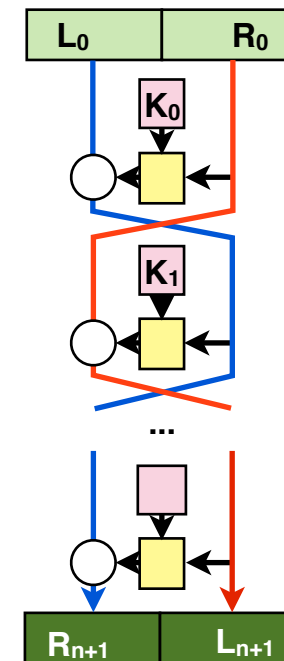
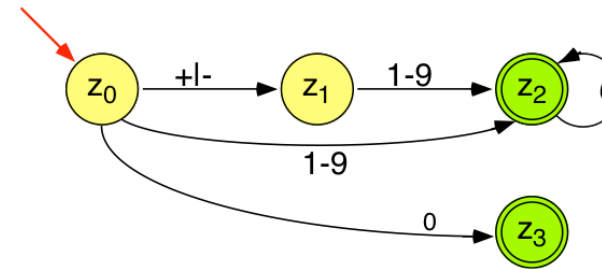
10.1 Medien und Wahrnehmung

10.2 Computergrafik

10.3 Standbilder

10.4 Video

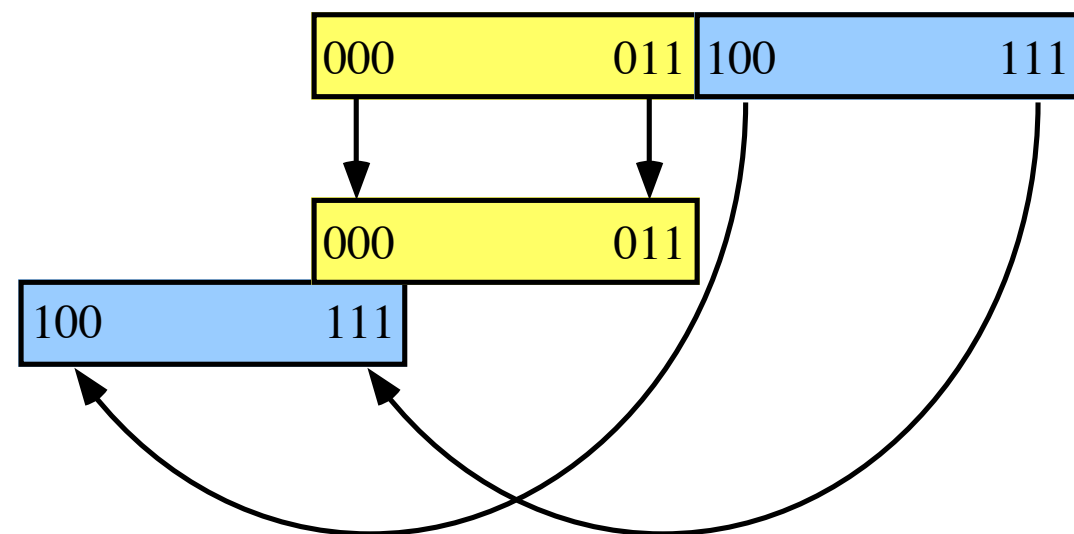
10.5 Audio



Information und Daten

- Information als mathematische Größe
- Zahlssysteme und Zahldarstellung
- Grundlegende Datentypen
- Ideen des maschinellen Rechnens

$$H = \sum_i p(s_i) \cdot \lg \frac{1}{p(s_i)}$$



Informationsbegriff

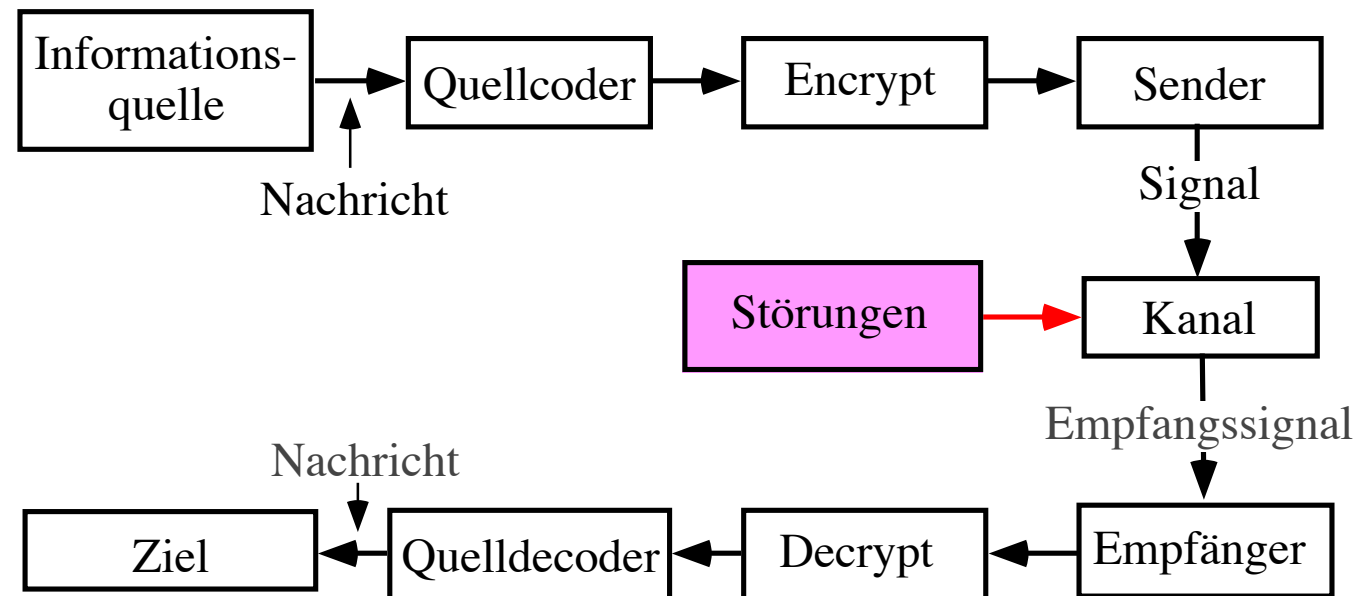
- Shannon, 1948: Definition als mathematische Größe
- Nachrichtenquelle
 - Nachrichten mit Wahrscheinlichkeit $p(s_i)$
 - Wahrscheinlichkeit bestimmt Informationsgehalt
- Informationsgehalt eines Symbols: (Selbstinformation) $I_i = -\lg p(s_i)$ bit

$$H = \sum_i p(s_i) \cdot \lg \frac{1}{p(s_i)}$$

- Entropie einer Nachrichtenquelle
- Beispiel Münzwurf
 - Ereignisse Kopf und Zahl
 - $p(\text{Kopf}) = 1/2$, $p(\text{Zahl}) = 1/2$
 - $\Rightarrow H(\text{Kopf}) = -\lg 1/2 \text{ bit} = \lg 2 = 1 \text{ bit}$
- Beispiel Würfel
 - Ereignisse 1, 2, 3, 4, 5, 6
 - $p(i) = 1/6$
 - $\Rightarrow H("3") = -\lg 1/6 \text{ bit} = 2,58... \text{ bit}$

- Wahrscheinlichkeit einer Symbolfolge $s_1s_2\dots s_n$
 - Symbole haben Wahrscheinlichkeit $p(s_i) = p_i$
 - $p = p(s_1) \cdot p(s_2) \cdot \dots \cdot p(s_n)$
 - $p(\text{"the"}) = 0,076 \cdot 0,042 \cdot 0,102 = 3,25584 \cdot 10^{-4}$
- Wahrscheinlichkeit $p(s_i)$
 - allein $\neq$ im Kontext
 - $p(s_i = 'h') = 0,042 \quad \Rightarrow H('the') = 11,6 \text{ bits}$
 - $p(s_i = 'h' \mid s_{i-1} = 't') = 0,307 \quad \Rightarrow H('the') = 6,5 \text{ bits}$
- Informationsrate in
 - englische Buchstaben: 4,76 bit / Buchstabe
 - englischem Text: 4,14 bit / Buchstabe
 - englischem Text: 9,7 bit / Wort, ~ 2 bit / Buchstabe
- Modell entscheidend
- Satz von der rauschfreien Quellkodierung
 - $H(N) = x \text{ bit} \Rightarrow \exists \text{ Kode } K \text{ mit } \text{Länge}_K(N) = x + \varepsilon \text{ Bits}$
 - Mittlere Kodelänge kann Entropie annähern
- Kode = Nachricht + Redundanz

- Übertragung
 - zwischen Computern
 - im Computer
 - Speicherung



- Kommunikation [Shannon, Weaver]:
 - Lebewesen oder Maschine beeinflusst anderes Lebewesen oder Maschine
 - technisches Problem
 - semantisches Problem (Symbole und ihre Bedeutung)
 - Effektivitäts-Problem (Einfluß der Bedeutung)

Zahldarstellung

- Ganze Zahlen
- Polyadisches Zahlssystem

$$z = \sum_{i=0}^{n-1} a_i B^i$$

$$- 0 \leq a_i < B$$

$$\text{- Basis 10: } 1492 = 2 \cdot 10^0 + 9 \cdot 10 + 4 \cdot 100 + 1 \cdot 1000$$

$$\begin{aligned} \text{- Basis 2: } 1492 &= 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 \\ &\quad + 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 = 10111010100 \end{aligned}$$

$$\text{- Basis 16: } 1492 = \$5D4 = 5 \cdot 256 + 13 \cdot 16 + 4 \cdot 1$$

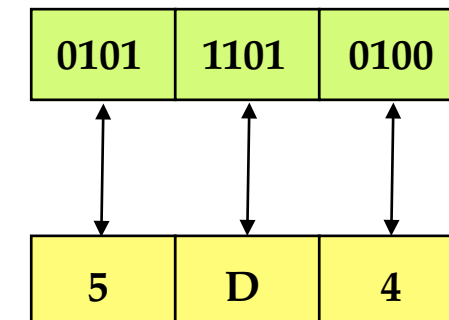
- Zahlenbereich

$$\text{- 10 Bit} \Rightarrow 0..1023 = 1 \text{ 'Kilo' -1}$$

$$\text{- 20 Bit} \Rightarrow 0..1024 \cdot 1024 - 1 = 1 \text{ 'Mega' -1}$$

$$\text{- 32 Bit} \Rightarrow 0..4\,294\,967\,295 \text{ (} 2^{32} - 1 \text{)} = 4 \text{ 'Giga' -1}$$

- negative Zahlen?



$$2^{10} = 1024$$

- Vorzeichen-Betrag (sign-magnitude)

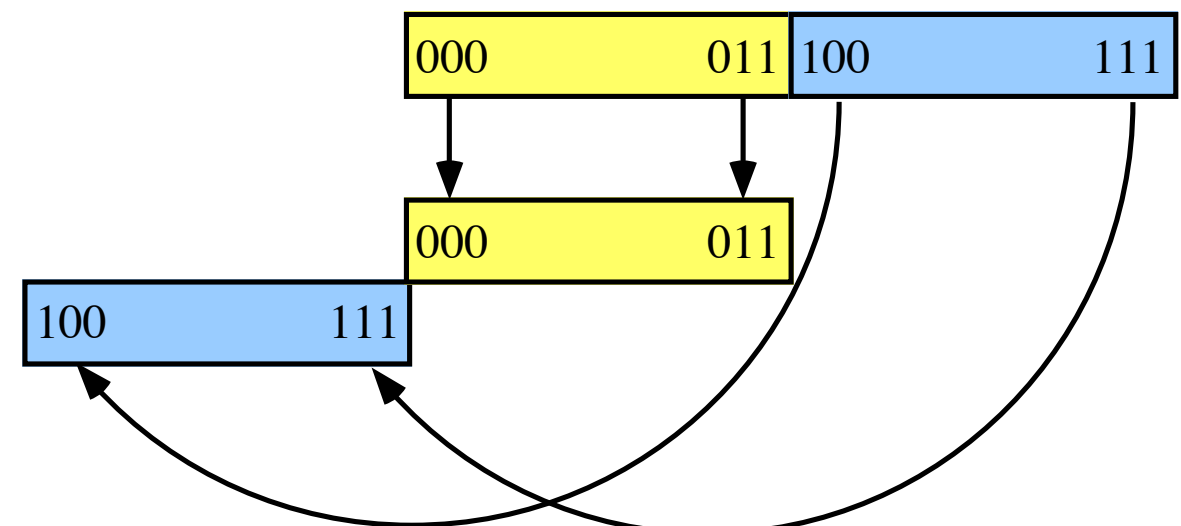
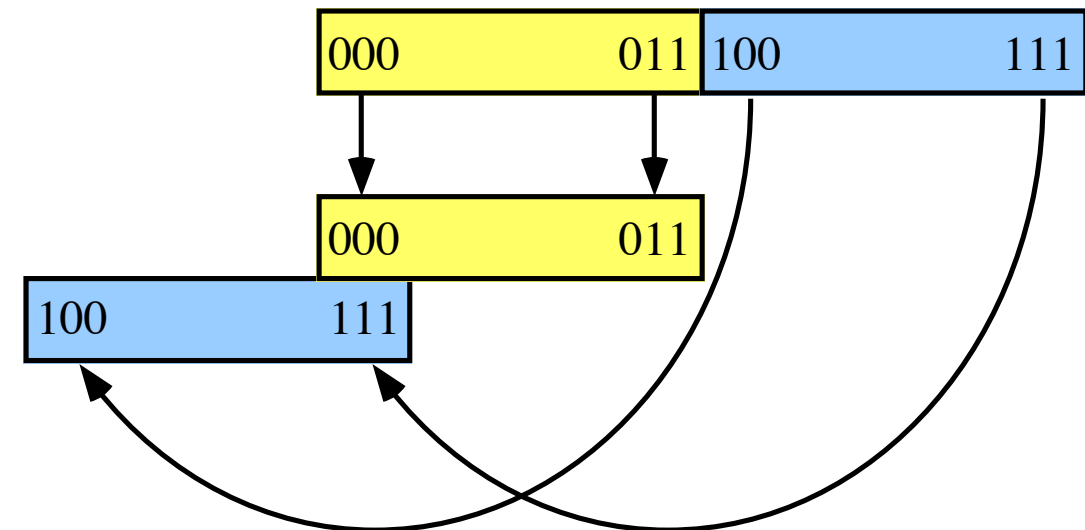
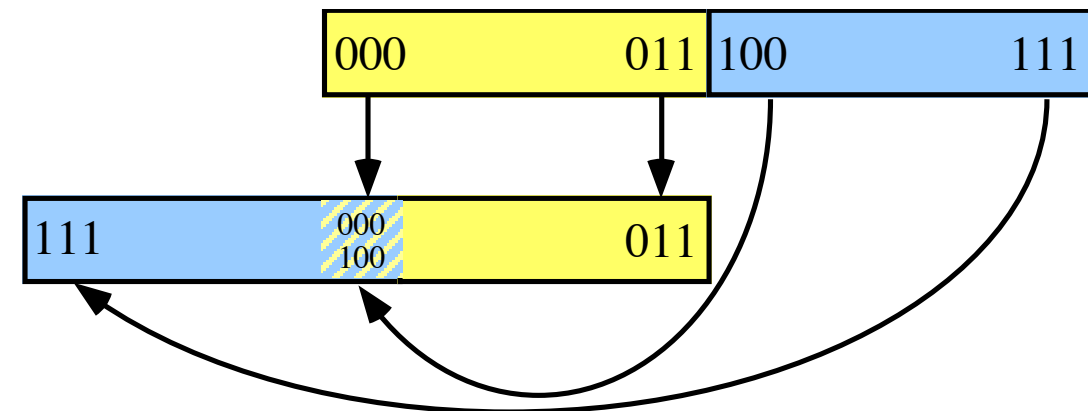
- 1 Bit Vorzeichen
- 8 Bit (256 Werte): -127..127
- komplexe Rechenregeln

- Stellenkomplement

- jede Stelle 'negieren': 0->1, 1->0
- 00111111 = 63
- 11000000 = -63
- negative Null: 00000000 und 11111111
- -127..0,0..127
- besondere Rechenregeln

- Basiskomplement

- jede Stelle negieren, 1 addieren
- 00111111 = 63
- $11000000 + 1 = 11000001 = -63$
- nur eine Null
- Umrechnung macht mehr Arbeit
- Rechenregeln einfach



- Rationale Zahlen

- Brüche: $1/2$, $1/3$, ...
- Dezimalschreibweise ggg,ddd: 0,5; 12,625
- evtl unendlich viele Stellen: $16/3$
- ggggg,dddd... : $1,3333333333333333333333333333...$
- ggg,ddd = ggg + 0,ddd
- ganzzahliger Teil + Bruchteil

- Näherungsdarstellung von reellen Zahlen

- $\pi \approx 3,1415926$
 - $\sqrt{2} \approx 1,414213562$
 - wann sind diese Fehler nicht katastrophal?
- => numerische Mathematik

±	64	32	16	8	4	2	1
---	----	----	----	---	---	---	---

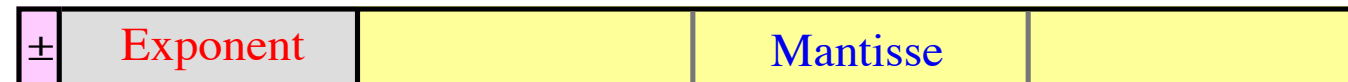
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$
---------------	---------------	---------------	----------------	----------------	----------------	-----------------	-----------------

- Normalisieren

- ggg,ddd = 0,gggddd * 10³
- 1,34567 = 0,134567 * 10¹
- 0,012345 = 0,12345 * 10⁻¹
- 0,0000001 = 0,1 * 10⁻⁶

- Floating-Point Zahlen

- 0 < Mantisse < 1
- 10^{exp} "Exponent"
- Vorzeichen



- Trick

- normalisieren => 0,10111010101010 * 2^{exp}
- erstes Bit immer = 1 => weglassen

- typische Formate

- ANSI/IEEE 754-1985
- single: 32 bit - 23 bit Mantisse, 8 bit Exponent
- double: 64 bit - 52 bit Mantisse, 11 bit Exponent
- extended: 80 bit

Rechnen

- Addieren im Zehnersystem

- stellenweise addieren

$$\begin{array}{r} 1513 \\ + 2112 \\ \hline 3625 \end{array}$$

- Übertrag

$$\begin{array}{r} 15\textcolor{red}{2}3 \\ + 21\textcolor{red}{9}2 \\ \hline 3\textcolor{blue}{7}15 \end{array}$$

- Rechenregeln für Übertrag $7+8 = 5 + \text{Übertrag } 1$

- Binärer Fall

- stellenweise addieren

$$\begin{array}{r} 01001010 \\ + 00101111 \\ \hline 011\textcolor{blue}{1}001 \end{array}$$

- Rechenregeln einfach: $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$ Übertrag 1

- Beschränkte Stellenzahl

- größte darstellbare Zahl
- Ergebnis grösser: Überlauf

$$\begin{array}{r} 11001010 \\ + 10101111 \\ \hline 01111001 \end{array}$$

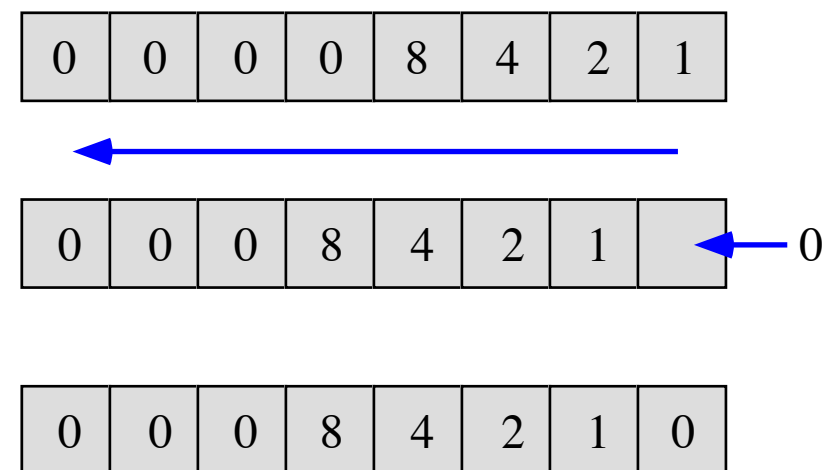
- eventuell negative Zahl

$$\begin{array}{r} 01001010 \\ + 01101111 \\ \hline 10111001 \end{array}$$

- Subtrahieren

- Komplement addieren: $a-b = a+(-b)$
- besondere Regeln zur Ergebnisinterpretation

- Multiplikation
- Primitives Verfahren
 - Multiplikand * Multiplikator
 - 1. erg auf Null setzen
 - 2. erg = erg + Multiplikand
 - 3. Multiplikator um 1 herunterzählen
 - 4. falls Multiplikator > 0: weiter mit 2
- Sonderfall
 - Verschieben um eine Stelle, Null nachziehen
=> Multiplikation mit Stellenwert



- Multiplizieren

- **Multiplikand** * **Multiplikator**

1. i auf Eins setzen

2. Addieren von (**Multiplikand** * Stelle i des **Multiplikators**)

3. Verschieben des **Multiplikanden** (mit Stellenwert multiplizieren)

4. i hochzählen

5. weiter mit 2, falls $i \leq \text{Stellenzahl Multiplikator}$

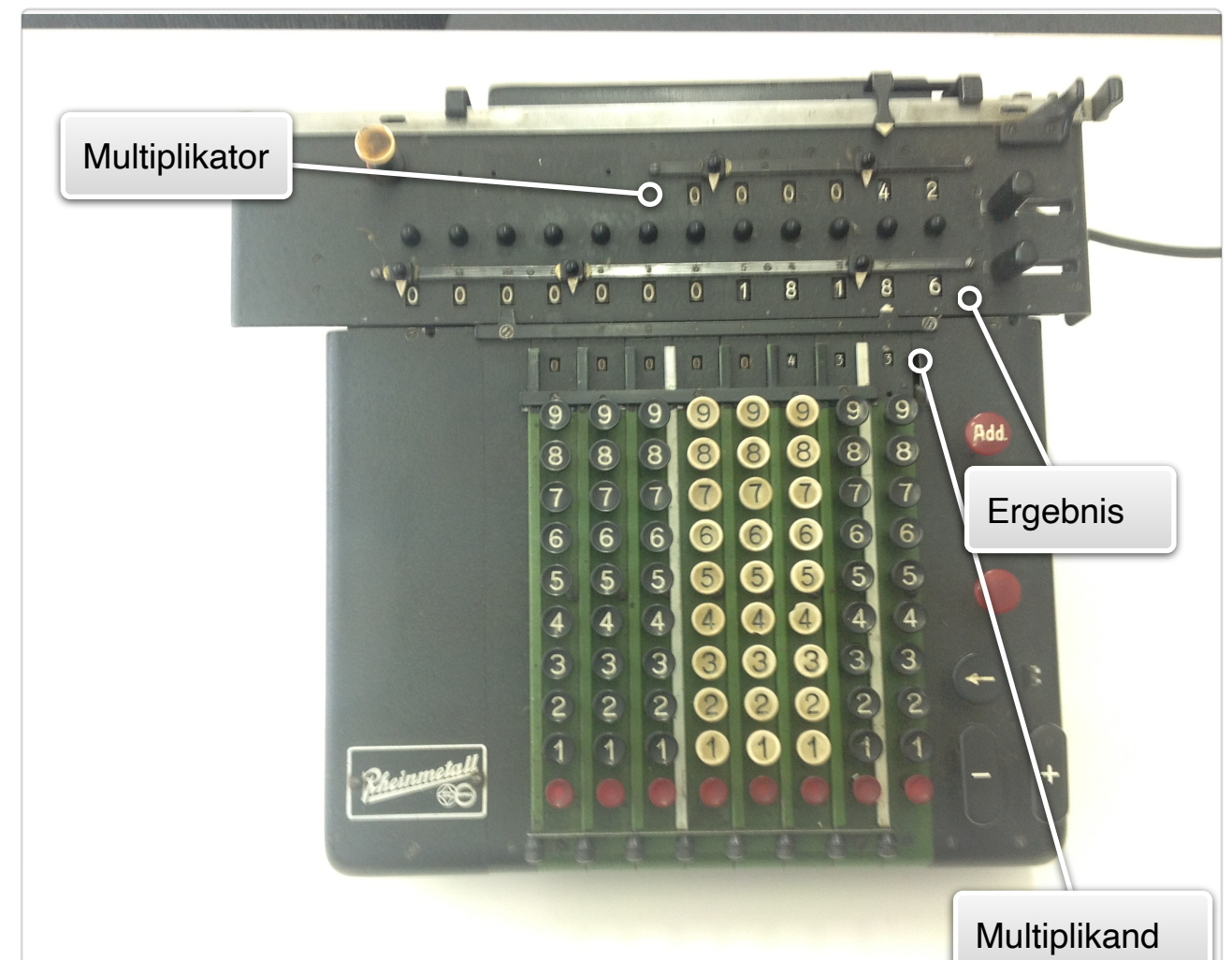
- im Zehnersystem:

$$\begin{array}{r}
 1214 \quad * \quad 211 \\
 \hline
 1214 \\
 12140 \\
 242800 \\
 \hline
 256154
 \end{array}$$

- Trick:

- **Multiplikator** in jedem Schritt eine Stelle nach rechts schieben
 - letzte Stelle in Schritt 2 verwenden

Interactive 2.1 Rechenmaschine ca. 1938



- binär Multiplizieren
 - ShiftLeft = Multiplikation mit 2
 - Stellen nur 0 oder 1 => Multiplikand addieren oder nicht
- Verfahren
 1. Ergebnis = 0; **a** = Multiplikand; **b** = Multiplikator
 2. falls letzte Stelle von **b** = 1: Ergebnis = Ergebnis + **a**
 3. ShiftLeft(**a**)
 4. ShiftRight(**b**)
 5. Falls **b**>0 : weiter mit 2
- Beispiel: 12 * 5

Iteration	a	b	erg
0	0000 1100	0000 0101	0000 0000
			+ 0000 1100
1	0001 1000	0000 0010	0000 1100
			nix tun
2	0011 0000	0000 0001	0000 1100
			+ 0011 0000
3	0110 0000	0000 0000	0011 1100

Datentypen

- Buchstaben

- 'A', 'B', ..., 'Z', 'a', 'b', ..., 'z', '.', ',', ';', ..., '0', ..., '9'
- 65, 66, ...
- ASCII, EBCDIC
- Sonderzeichen: ., ! " \$ % & / () = ? @ " # ^ \ ~ . - * # ; : _ - < >
- unsichtbare Zeichen: Space / Blank, CR

char <Name>; A

- Zeichenketten (strings)

- zusammengesetzt aus Buchstaben
- 'Auto', 'Bergakademie', 'Klaus Dieter'

char <Name> [n];

K	l	a	u	s	␣	D	i	e	t	e	r		
---	---	---	---	---	---	---	---	---	---	---	---	--	--

- Wie lang ist ein string?

- Länge(string) = Anzahl Buchstaben
- Länge('Auto') = 4
- Länge('Klaus Dieter') = 12
- Längelfeld oder besonderes Zeichen am Ende

- Boolesche Typen
 - 2 Werte: true, false
- Aufzählungen
 - frei gewählte Bezeichner
 - (rot, gruen, blau)
 - (Mercedes, BMW, VW, Ford, Opel, Audi, Porsche)
 - werden auf Zahlen abgebildet
 - `enum {red,green,blue} color`
- Zahlen (typische Länge ...)
 - `short, unsigned short` (16 bit)
 - `int, unsigned int` (16, meist 32 bit)
 - `long, unsigned long` (32 oder 64 bit)
 - `float` (32 bit)
 - `double` (64 bit: 2.22 e-308 bis 1.79 e+308)

- Zusammengesetzte Datentypen
 - aus mehreren elementaren Datentypen zusammengesetzt
 - gefühlter Sonderfall Zeichenkette siehe oben
- Vektoren und Matrizen
 - gleiche Datentypen

7	12	73	0	0	0	123	456	13	13	12	9	12	9
---	----	----	---	---	---	-----	-----	----	----	----	---	----	---

`float vielstufig [n][m]...[k]`

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 7 & 8 \\ 0 & 1 & 9 & 2 \\ 12 & 13 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 7 & 2 & 5 \\ 1 & 2 & 4 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 4 \\ 4 \\ 8 \end{pmatrix}$$

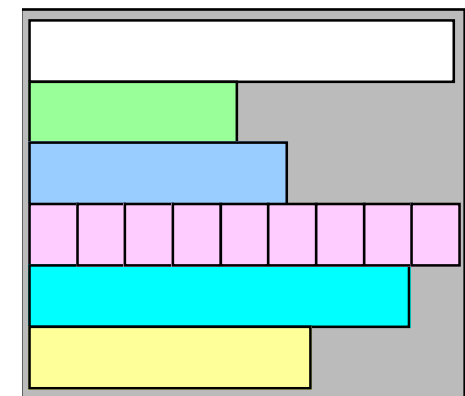
- Datenstrukturen (Records, zusammengesetzte Daten, ...)
 - unterschiedliche Typen

```

struct Wagenbeschr
{
    char    name[32];
    enum    {Schlaf,Abteil, ...} Art ;
    int     Baujahr;
};

```

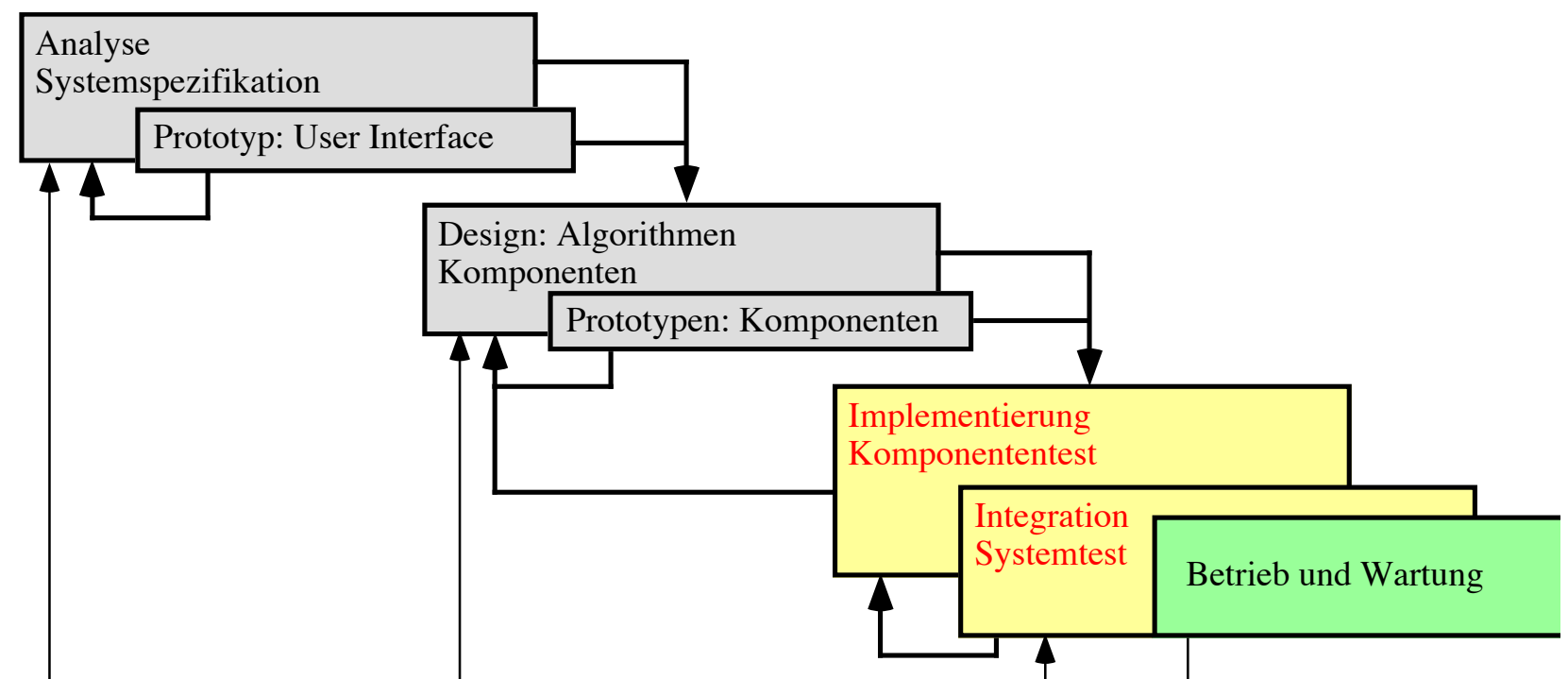
- beschreiben Gegenstände, Personen, ...



KAPITEL 3

Vom Problem zum Programm

- The Art of Computer Programming [[D. Knuth](#), 1968]
- Programmieren ist Ingenieurtätigkeit
 - systematisch, planbar
 - reproduzierbar
 - dokumentiert
- Theoretischer Ablauf



- Computer Aided Software Engineering (CASE)

- Computerprogramm zum Programmieren
- grafische Hilfsmittel
- formale Spezifikation
- Versionsverwaltung

- Rollen

- Projektleiter
- Analytiker
- Programmierer
- Tester

- Phasenmodell

- globale Anforderungen, Struktur der Arbeit
- Untersuchung eines Geschäftsgebietes, Anforderungen
- Systementwurf aus Benutzersicht
- Technische Design
- Codierung, Datenbank-Generierung
- Systemeinführung, Test
- Verwendung (Produktion)

- Agile software development

- kleine Gruppen
- Selbstanordnung
- ständige Kommunikation
- intensive Reviews
- Kunden integriert in Zyklus

- Extreme Programming

- Imperatives Programmieren

- Programm entsprechend Datenfluss
- Funktionen zur Daten- und Zustandänderung
- Datenstrukturen
- Pascal, C, Modula, Cobol, Fortran, ...

```
fac = 1; i = 1;  
do  
    {fac = fac * i;  
      i = i+1;}  
while (i <= n);
```

- Objektorientiertes Programmieren

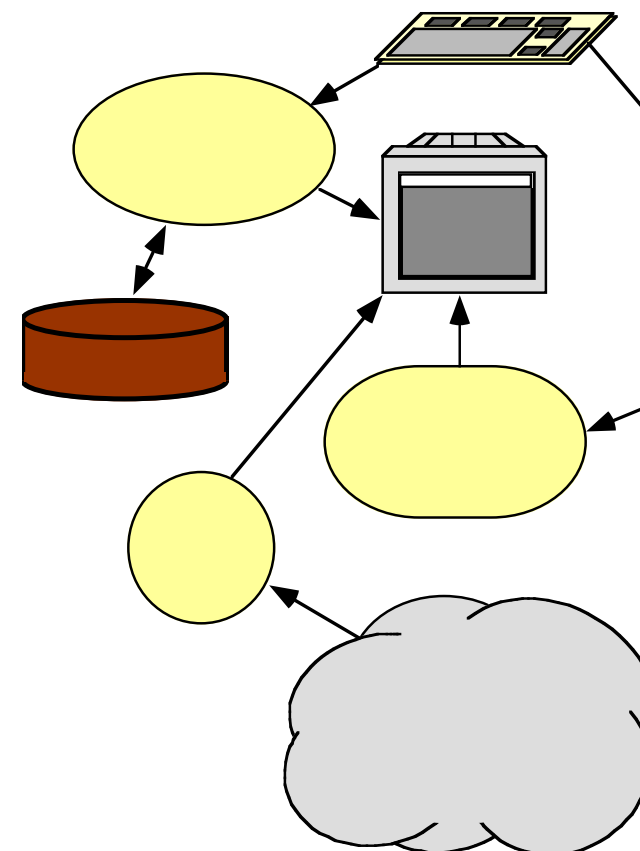
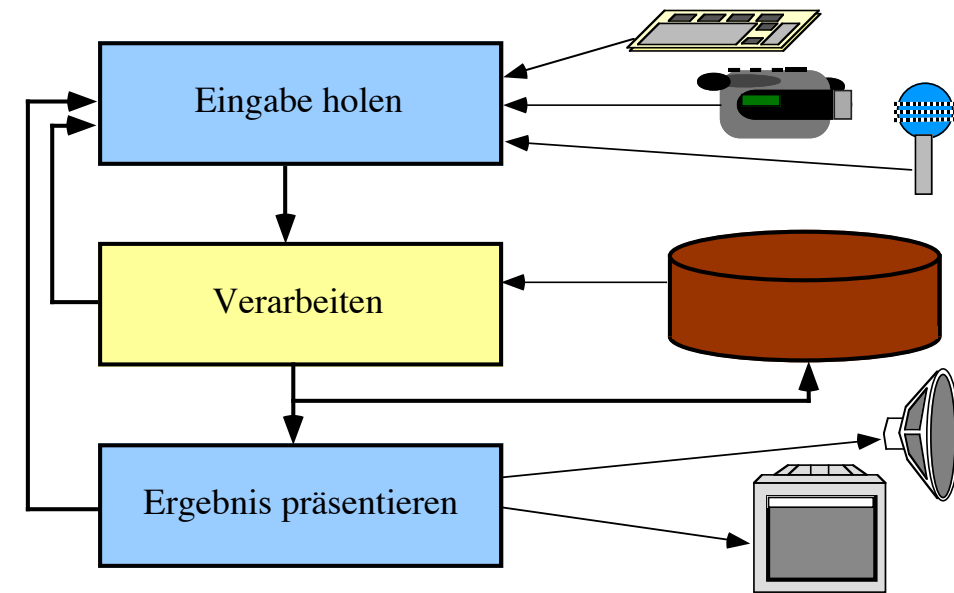
- Datenstrukturen mit Transformationsfunktionen
- Zustände in den Datenstrukturen
- ereignisorientiertes Programmieren
- Smalltalk, Java, C++, Oberon, ...

- Funktionales Programmieren

- Folgen mathematisch/logischer Funktionen
- Lambda-Kalkül
- ohne Zustände und änderbare Daten
- APL, ML, **Haskell**, Lisp, Prolog, ...

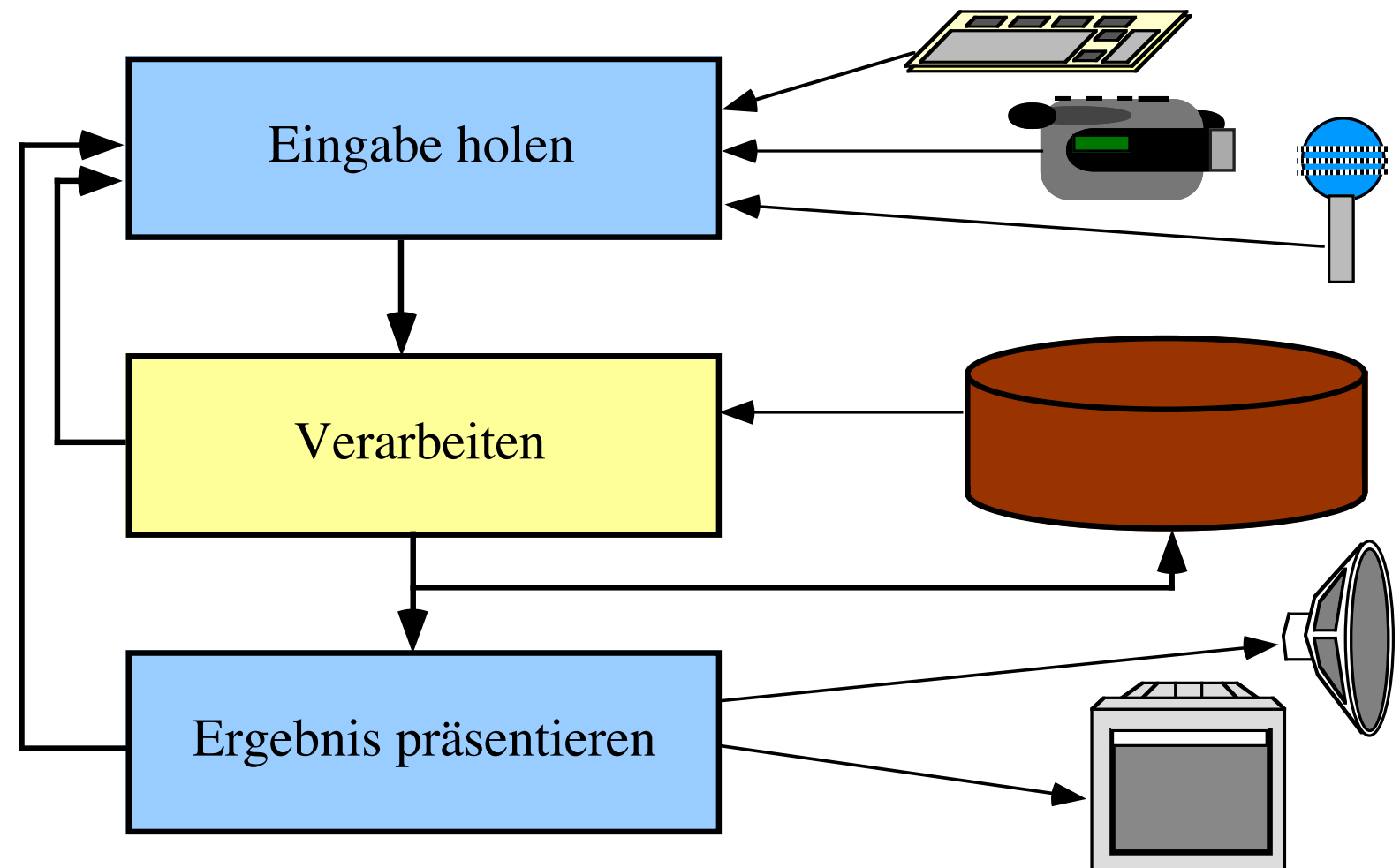
```
fac :: Integer -> Integer  
fac 0 = 1  
fac n | n > 0 = n * fac (n-1)
```

- Aktives Programmieren
 - Ablauf kontrollieren
 - Beginn,
 - {Eingabe, Verarbeitung, Ausgabe},
 - Ende
 - Programmierer plant Programmablauf
 - Programmiersprachen siehe oben
- Reaktives Programmieren
 - Komponenten bereitstellen
 - gekapseltes Verhalten
 - ereignisgesteuert
 - Anreiz -> Transformation -> Antwort
 - AJAX
 - php, Ruby, ...



Idee und Analyse

- Aufgaben
 - Berechnen
 - Bearbeiten
 - Organisieren
 - Kommunizieren
 - Messen und Steuern
- Programmstruktur



- **Problem** definieren
 - intuitive Beschreibung
 - Eingabe detailliert aufschreiben
 - Resultate definieren
 - Ausgabe genau aufschreiben

PROBLEM	VERFAHREN	TYPISCHE OPERATIONEN
Pullover anfertigen	Strickmuster	rechte Masche, linke Masche
Kuchen backen	Rezept	Mehl wiegen, rühren
Schrank aufstellen	Bauanleitung	schrauben, stecken, fluchen

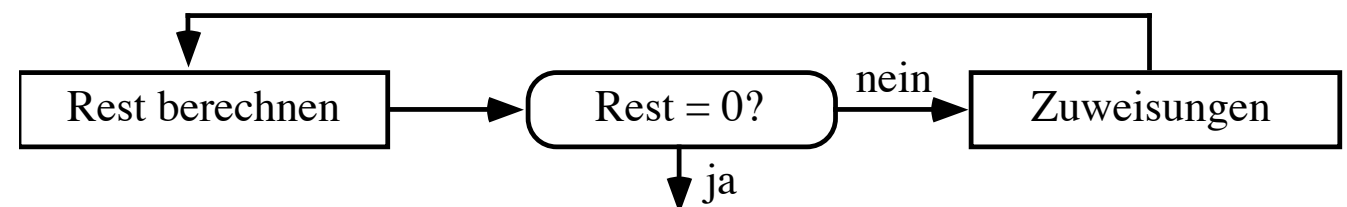
- Wissen im Gebiet sammeln
 - Erfahrung der Mitarbeiter bzw. Kunden nutzen
 - klassische Bearbeitung genau studieren
 - ähnliche Programme untersuchen
 - Gesetze bzw. Normen beachten
- **Verfahren** (er)finden
 - Formeln und Regeln
 - Menge von **Operationen**

- Berechenbarkeit prüfen
 - Eingabewerte erfassbar?
 - Ergebnis ausdrückbar?
 - Laufzeit (Komplexität) des Verfahrens
 - Ressourcen (Speicherplatz)
- Datenstrukturen bilden
 - Abbildung der Eingabe
 - Transformation in Ausgaben
 - Vollständigkeit
 - Speicherplatz
 - optimierter Zugriff (schnell und / oder einfach)
- Werkzeuge auswählen
 - Computertyp- und Ausstattung, Betriebssystem
 - Programmierwerkzeuge (Sprache, Prototyping-System, ...)
 - Basissoftware (Datenbank, ...)
- Kosten ermitteln
 - Arbeitszeit
 - Geräte und Material

Sequentialisierung

- Algorithmus

- Abu Ja'far Mohammed ibn Mûsâ *al-Khowârizm*
- Abfolge von Schritten
- Rechenvorschrift
- bis ca. 1950 nur im Zusammenhang mit Euklidischem Algorithmus



- Euklidischer Algorithmus

- größten gemeinsamen Teiler **ggt** von (m,n) finden

```
1. setze rest = (m DIV n)
2. rest = 0? setze ggt = n; fertig
3. setze m = n und n = rest; weiter mit 1.
```

- Schrittweise Verfeinerung (step-wise refinement)

- Zerlegung in Teilprobleme
- Verfahren finden oder entwickeln

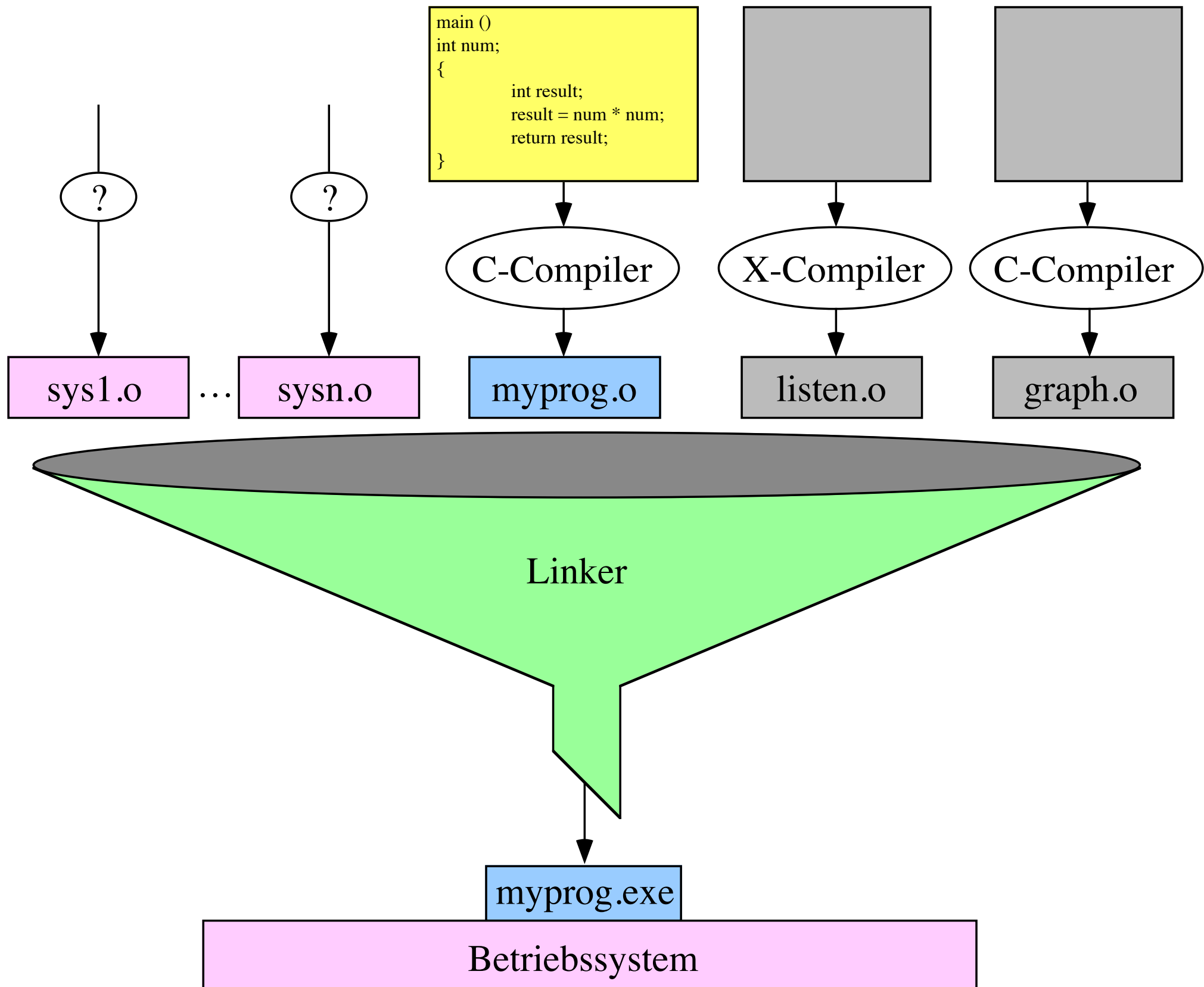
- Beispiel: Kaffee machen (Pseudocode)
- Grobverfahren
 1. Wasser kochen
 2. Nescafe in die Tasse
 3. Wasser in die Tasse
- Einzelschritte verfeinern
 - 1.1 Kessel mit Wasser **füllen**
 - 1.2 Auf die Herdplatte stellen
 - 1.3 Herdplatte anstellen
 - 1.4 **Warten bis** Wasser **kocht**
 - 1.5 Herdplatte abstellen
 - 2.1 Glas öffnen
 - 2.2 **Menge** Kaffeepulver auf den Löffel laden
 - 2.3 Löffel in die Tasse leeren
 - 3.1 Kessel von der Herdplatte nehmen
 - 3.2 Tasse mit Wasser **füllen**
 - 3.3 Umrühren
- Weiter verfeinern

- Methoden zur Strukturbeschreibung
 - Algorithmus beschreiben
 - Verfeinerungsstufen
- Grafische Darstellungsformen
 - Flußdiagramme (DIN 66 001)
 - Struktogramm (Nassi-Shneiderman-Diagramm, DIN 66 261)
 - Datenflußplan
- Sprachliche Beschreibung
 - Pseudocode
 - spezielle Entwurfssprachen
 - Spezifikationssprache
 - Elemente einer Programmiersprache
- Wesentliche Verarbeitungselemente
 - Sequenz
 - Selektion, Iteration
 - Gruppierung, Parallelität
 - Ein- und Ausgabe

Programmieren

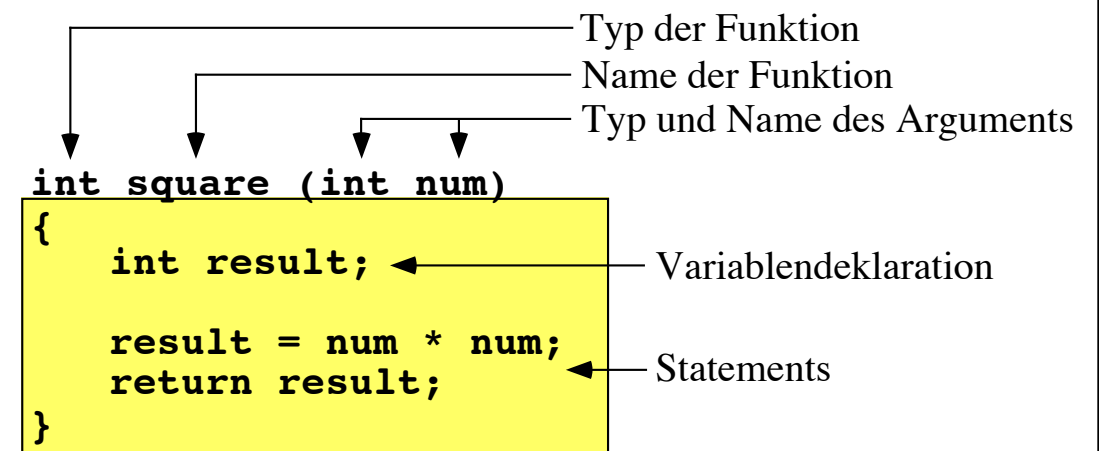
- Viele Programmiersprachen
 - Allzweckssprachen: C, Pascal, Modula, Oberon, Ada
 - Spezialisiert: COBOL, FORTRAN
 - BASIC
 - objektorientiert: SmallTalk, Java, C++, Oberon
- Arbeitszyklus
 1. Editieren
 2. Übersetzen (Compiler)
 3. Zusammensetzen mit Standardteilen (Linker)
 4. Ausführen und Fehlersuchen
 5. if Fehler then 1.
- Fehlersuche
 - wesentlicher Bestandteil des Programmierens
 - Debugger
 - schrittweise ausführen
 - Werte und Zwischenergebnisse anzeigen

It is practically impossible to teach good programming style to students that have had prior exposure to BASIC. As potential programmers, they are mentally mutilated beyond hope of regeneration. [E. W. Dijkstra]



- C
 - von Dennis Ritchie und Ken Thompson
 - weit verbreitet, sehr portabel
 - eng verbunden mit dem Erfolg von UNIX
 - ANSI-Standard X3.159-1989, C89 / C90
 - ISO-Standard ISO 9899:1999 (C99)
 - hat vielfältige Ausdrucksmöglichkeiten
 - effizientes Programmieren möglich
 - verleitet zum Tricksen
- Zentrale Abstraktion: Funktionen
- Funktionskopf (Deklaration)
 - beschreibt die Schnittstelle zu den Benutzern
 - Parameteranzahl und -typ (Argumente)
 - Ergebnistyp
- Funktionsrumpf (Definition)
 - Variablen-Deklaration (Speicherplatz-Bestellung)
 - Verarbeitung
 - benutzt eventuell andere Funktionen

```
int square (int num)
{
    int result;
    result = num * num;
    return result;
}
```



- 3 Funktionen die man immer braucht
- Einlesen von Eingabewerten: `scanf()`
 - beliebig viele Parameter
 - int, float, char, string...
 - von stdio - Standard-Eingabe
 - meist Tastatur

```
scanf("%d %d", &ersteZahl, &zweiteZahl);
```
- Eingabesteuerung
 - Formatstring mit Beschreibung der Eingabe
 - %d, %i integer
 - %e, %f, %g float
 - %c char
 - %s string
- Besondere Zeichen
 - space trennt und wird übersprungen
 - Zeilenende beendet Eingabe

```
123__456__<CR> => ersteZahl=123; zweiteZahl=456;
```
- Eingabeanleitung muß mit `printf()` erzeugt werden

- Ausgabe: printf()

```
printf("Hello World\n ");
```

- beliebig viele Parameter
- Formatsteuerung wie scanf()
- auf "stdout" - Standard Ausgabe
- z.B. Bildschirm
- mischt Formatstring und Variablen

```
printf("hier sind 3 Zahlen: %f %d %f", fl, gz, my);
```

- Kombinierte Ein/Ausgabe

=> Benutzungsoberfläche

```
printf("Buchstaben eingeben:"); /* scanf flushed */
```

```
scanf("%c",&zeichen); /* stdin */
```

```
printf("Der Zeichenwert ist: %d\n", zeichen)
```

```
Buchstaben eingeben:A
Der Zeichenwert ist: 65
—
```


- Spezial-Funktion main()
 - dahinter steckt das selbstgeschriebene Programm
 - gesamte Software ist eine Menge von Funktionen
 - Anwendungsprogramm ist auch eine Funktion

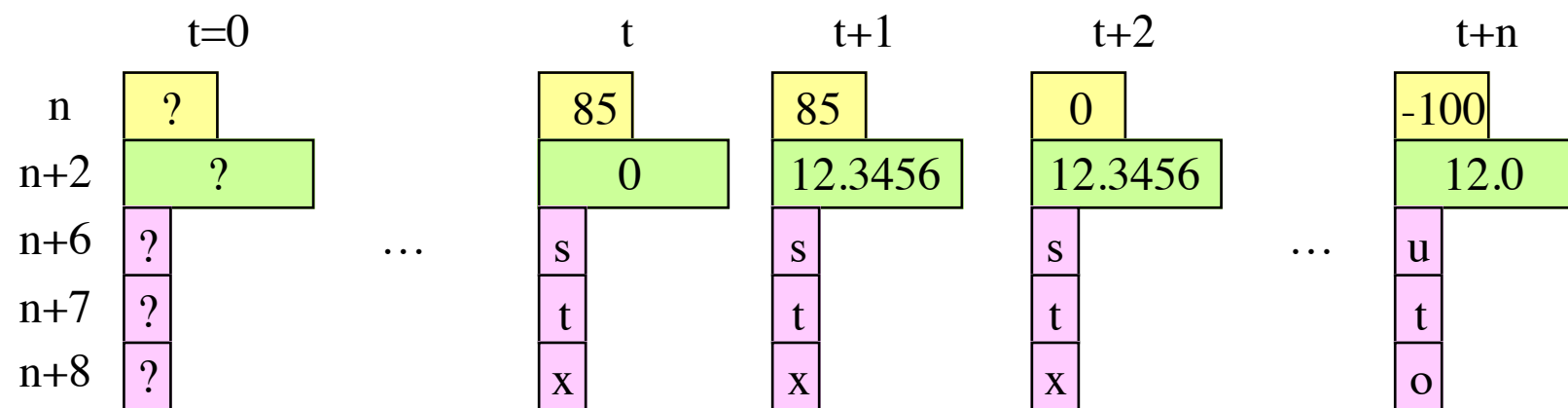
```
#include <stdio.h> /*das Noetigste importieren*/  
int main(void)  
{  
    printf("kuckuck\n"); /* Verarbeitung */  
    return 0; /* mit Rueckgabewert "kein Fehler" */  
}
```

- Programm eintippen mit irgendeinem Editor
 - Notepad, TextEdit, emacs, vi, xedit, ...
- Übersetzen und ausführen
 - cc ist der compiler, linkt auch

```
>cc -Wall -o myprogram myprogram.c  
>./myprogram  
kuckuck  
>
```

Variablen-Deklaration: `int result;`

- Identifier
 - Namen, die der Programmierer erfindet
 - Buchstaben, Ziffern, "_", keine Ziffer am Anfang
 - **case-sensitive**
- Manche Zeichenketten bereits besetzt
 - Schlüsselworte: `if`, `switch`, `while`, ...
 - vordefinierte Typen: `int`, `float`, ...
- Variablen
 - Container zum Aufbewahren von Werten
 - werden vom Compiler als Speicherplatz angelegt
 - nach dem Programmende (...) verloren



- Mit Identifiern bezeichnet
 - Verwendung in Formeln, ...
 - Speicherplatzadresse wird manchmal benötigt
 - &<ident> liefert Adresse
- Aussagekräftige Namen
 - dokumentieren Programm
 - lassen Typ erkennen
 - ungarische Notation [Symoni, 1999]
- Deklaration
 - Typ <Name>, ..., <Name>;
int i, j;
float pi, psi, karl_otto;
char zeichen;
char name [32], name2 [16], _2tername [31];

- Literale

- Wert fest (Konstante?)
- für Vergleiche, Initialisierung
- Zahlen, Buchstaben, String

```
index = index + 1;
```

```
if (zeichen > 'Z') ...;
```

```
zeichen = zeichen + 32; /* macht Kleinbuchst. */
```

```
pi = 3.1415926;
```

```
mpi = -3.1415926;
```

- Variablendeklaration mit Vorbesetzung

- Typ <Name> = <Ausdruck>;

```
int        i = 12;
```

```
double    pi = 3.1415926;
```

```
double    pi1 = 4*atan(1);
```

```
double    pi2 = -mpi;
```

```
char       space = ' ';
```

- Typkonvertierung
 - zur Anpassung von Parametern
 - Verwendung in Ausdrücken; Bsp: `erg = i * pi;`
 - eventuell mit echter 'Umrechnung'
 - manche Konvertierung nur 'formal'
- Explizite Konvertierung (typecast)
 - `(<typename>) <ausdruck>`
 - `int n = 3; ...; erg = sin((double) n);`
 - guter Stil
- Implizite Konvertierung
 - vom Compiler eingebaut
 - typisch in Ausdrücken und Funktionsaufrufen
 - Rückgabewerte
 - Typhierarchie
- Vorsicht mit der impliziten Konvertierung

```
i = 1.5;    /* i enthaelt nun 1 */
j = -5.8;   /* j enthaelt nun -5 */
f = 3;      /* f enthaelt nun 3.0 */
```

- Gültigkeit der Variablen
 - Ort der Vereinbarung
 - im 'Programm' => globale Variable
 - im Block => lokale Variable
 - in der Funktion => lokale Variable
 - entscheidet über Benutzbarkeit
- Lebenszeit von Variablen
 - **lokal**: in der Funktion - nur während des Funktionsaufrufes (Block)
 - **global**: während der gesamten Programmausführung
 - **static** macht auch lokale Variable permanent
 - siehe auch Abschnitt Funktionen

Operatoren, Ausdrücke und Anweisungen

- Formeln

`dreiecksflaeche = grundlinie * hoehe / 2;`

`kreisflaeche = radius*radius*3.1415926;`

`umfang = 2*radius*pi;`

- Unäre (einstellige) Operatoren

- Binäre (zweistellige) Operatoren

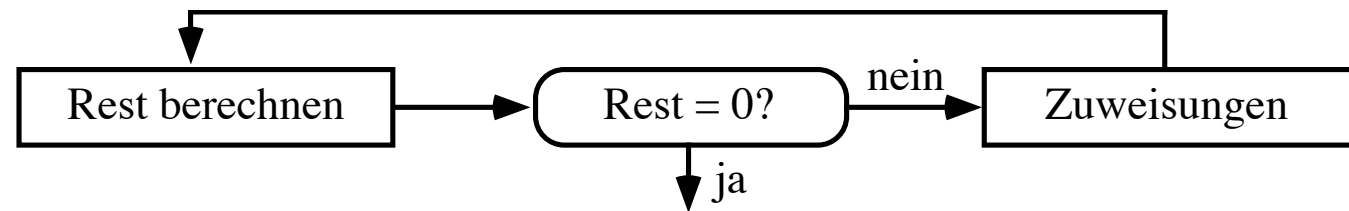
Vorzeichen	+, -
Inkrement und Dekrement	++, --
Adresse	&
Negation	!

Multiplikation	*, /, % (Rest der int-Division)
Addition	+, -
Relation	<, <=, >, >=
Gleichheit	==, !=
logisches UND	&&
logisches ODER	
Zuweisung	=, +=, -=, *=, ...

- %

do

```
{ rest = m % n;
  if (rest != 0)
  { m=n; n=rest;
  }
} while (rest>0);
```



- Vergleiche <, <=, >, >=

- Zahlen 0 oder 1 als Ergebnis
 - 0 ist false, alles "ungleich 0" ist true
- ```
rest>0; m!=n;
```

- Gleich oder ungleich: ==, !=

- geht oft nicht bei Gleitpunktzahlen
- liefert 0 oder 1

```
-1 < 0, 0 == 0, 1 != -1 /* true */
0>1, 1>10 /* 0 */
```



- Logische Operatoren: `&&`, `||`  
`((a>b)&&(f>g)) /* (5>3) UND (7.3>-1.2) => true */`  
`((a>=b)|| (f<=g)) && i) /* i entspricht i!=0)*/`
- Zuweisung =  
`links = <Ausdruck>;`  
 - legt ausgewerteten Ausdruck in Variable links  
 - kann auch als boolsches Resultat verwendet werden  
`if (i=0) printf("ich werde nie gedruckt");`
- Arithmetische Zuweisung: `+=`, `-=`, `*=`, `/=`, `%=`  
`<variable> op= <ausdruck>;`  
 - entspricht `<variable> = <variable> op <ausdruck>;`  
 - natürlich auch mit Ergebnis
- Inkrement: `++`, `--`  
 - addiert bzw subtrahiert 1  
 - liefert Inhalt von j vor dem Inkrement: `j++`  
 - liefert Wert von j nach dem Inkrement: `++j`  
 - liefert auch Ergebnis: `do ... while (j--)`

- Klammerregeln
  - Klammer hat höchsten Vorrang
  - gruppiert Formelteile
$$(a+b)*c \neq a+b*c$$

$$1 + ((3+1)/(8-4)-5)$$
- Vorrang-Regeln (precedence)

| Typ                      | Assoziativität  |
|--------------------------|-----------------|
| Klammern                 | links -> rechts |
| unäre Ops                | rechts -> links |
| Multiplikation, Addition | links -> rechts |
| relational               | links -> rechts |
| bitweises UND, ODER      | links -> rechts |
| logisches UND, ODER      | links -> rechts |
| Zuweisung                | rechts -> links |

- Wirkung auf verschiedene Typen

```
int j=5;
float f=5;
einint = j / 2; /* einint == 2 */
einfloat = f/2; /* einfloat == 2.5 */
```

- Typ des Ausdrucks wird durch Elemente des Ausdrucks bestimmt

```
einfloat = j/2; /* einfloat == 2.0 */
```

- Elegant

```
aktuelles_zeichen = name[index++];
summe += element;
```

- Etwas kryptisch:

```
a = b = c;
if (a=b) ...;
```

- Bitte nicht:

```
x = j * j++;
einfloat = einint = 3.5;
einint = einfloat = 3.5;
```

- Anweisungen (Statements)

- <Ausdruck>;

- Selektion, Iteration, Sprung

- Increment

- mehrere geklammerte Statements

```
{ a=7;
 anz_gedr_zchn=printf("auch printf hat Ergebnis");
 gesZeichen += anz_gedr_zchn;
 if (gesZeichen > 80) printf("\n");
}
```

- {} macht aus mehreren Statements ein Statement (Block)

- {...;...;...;}

- macht Syntax-Beschreibung einfacher

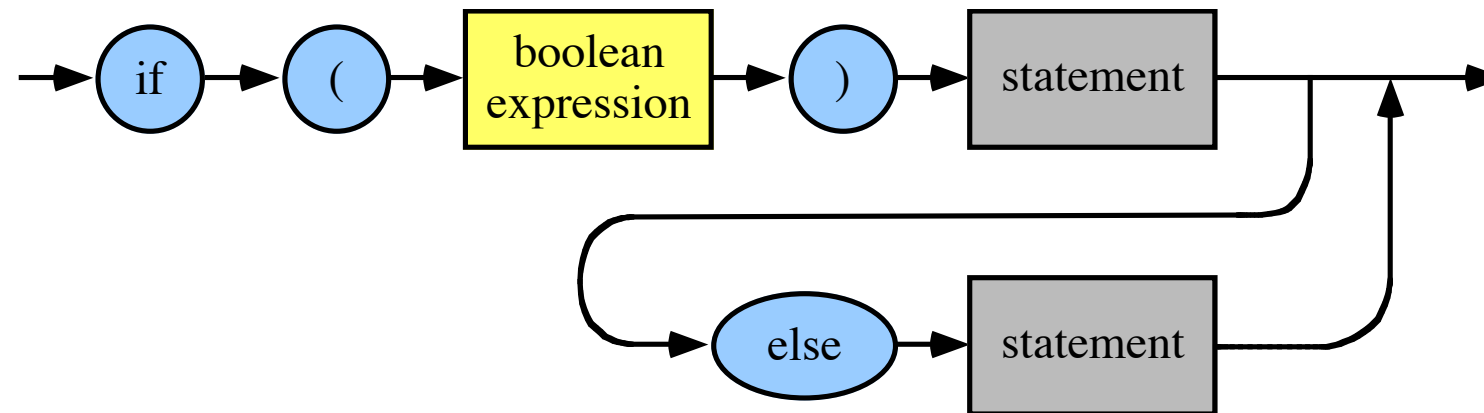
- Variablenvereinbarung möglich

```
if (a>b) <statement>;
```

```
else <statement>;
```

## Kontrollfluß

- Verzweigung
  - then implizit, immer vorhanden



- else optional

- Beispiele

```
if (a>b) max = a;
else max = b;
```

```
if (b>a) /* Werte tauschen */
 { swap = a; a = b; b = swap; }
```

```
if (divisor==0) printf("Fehler\n");
else quotient=dividend/divisor;
```

if-Schleife: goto fail

- Boolean Expression

- eigentlich normaler Ausdruck mit Zahl als Resultat

- true falls Resultat  $\neq 0$ !

- ```
if (a-b) printf("a ist ungleich b");
```

- klassischer Fehler: $a=0$

- ```
if (a=0) ...; /* immer falsch */
```

- ```
if (b=5) ...; /* immer wahr */
```

- geschachtelte Verzweigung

```
int min(a,b,c)
```

```
int a,b,c;
```

```
{ if (a<b)
```

```
    if (a<c) return a;
```

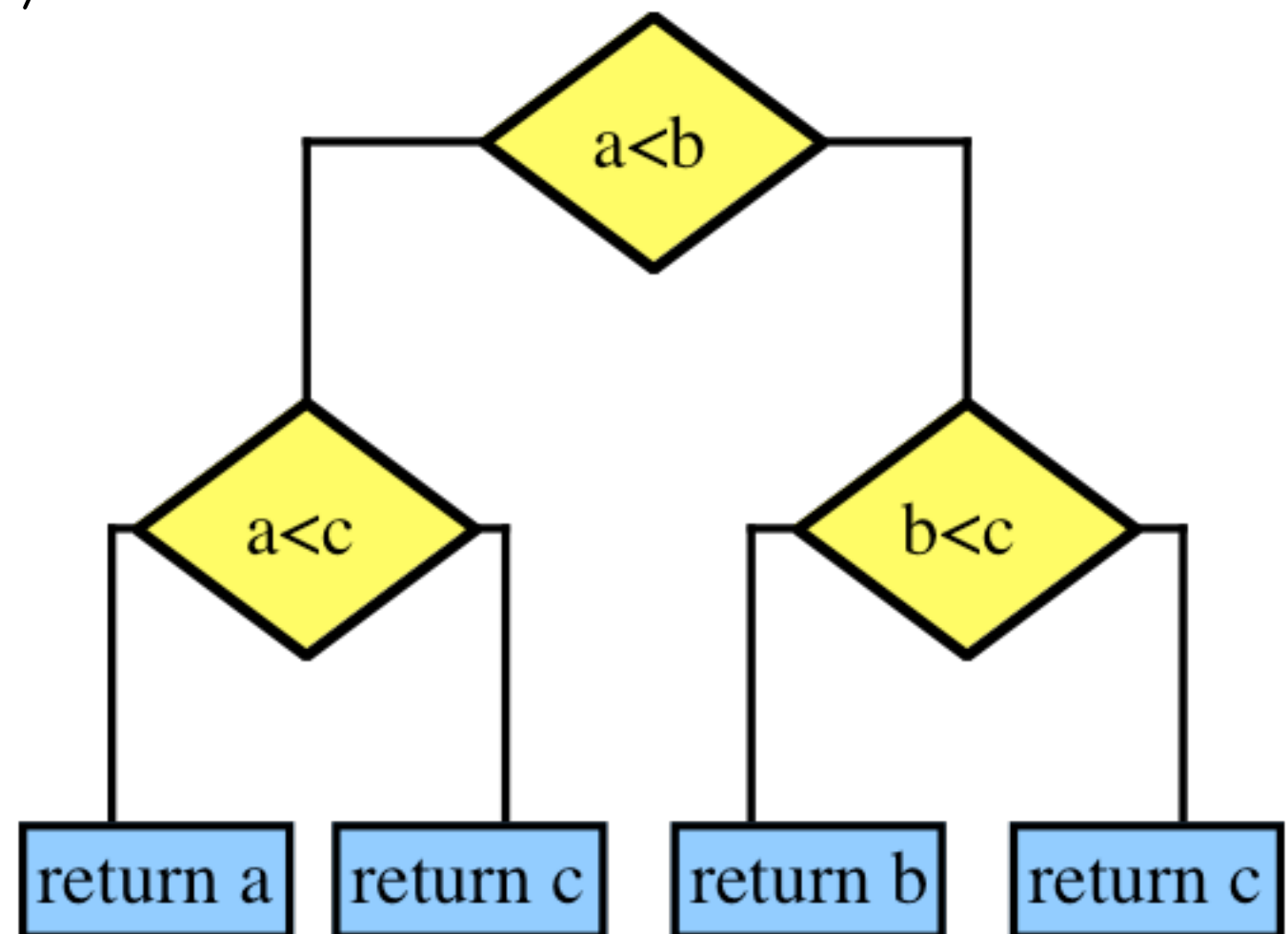
```
    else return c;
```

```
else if (b<c)
```

```
    return b;
```

```
    else return c;
```

```
}
```



- Kaskadierte Verzweigung

- mehr als 2 Fälle

- nacheinander abfragen

```
int fallunterscheidung (eingabe)
```

```
char eingabe;
```

```
{
```

```
    if (eingabe == 'A')
```

```
        return 1;
```

```
    else if (eingabe == 'B')
```

```
        return 2;
```

```
    else if (eingabe == 'C')
```

```
        return 3;
```

```
    else if (eingabe == 'D')
```

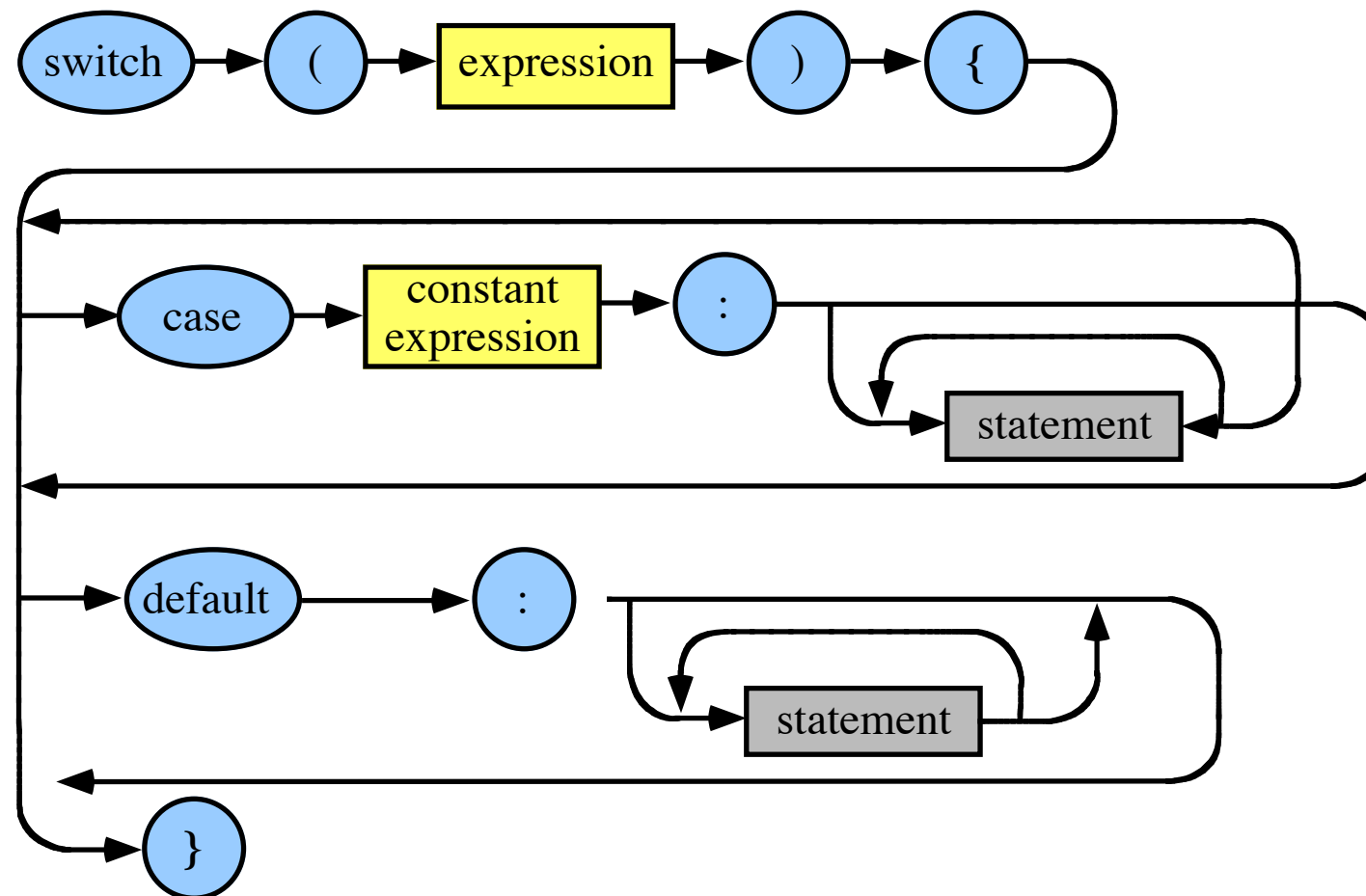
```
        return 4;
```

```
    else
```

```
        return -1;
```

```
}
```

- Mehrfach-Verzweigung



- expression wird ausgewertet
- Resultat wird mit constant expressions verglichen
- Ausführung wird bei 'true'-Zweig fortgesetzt

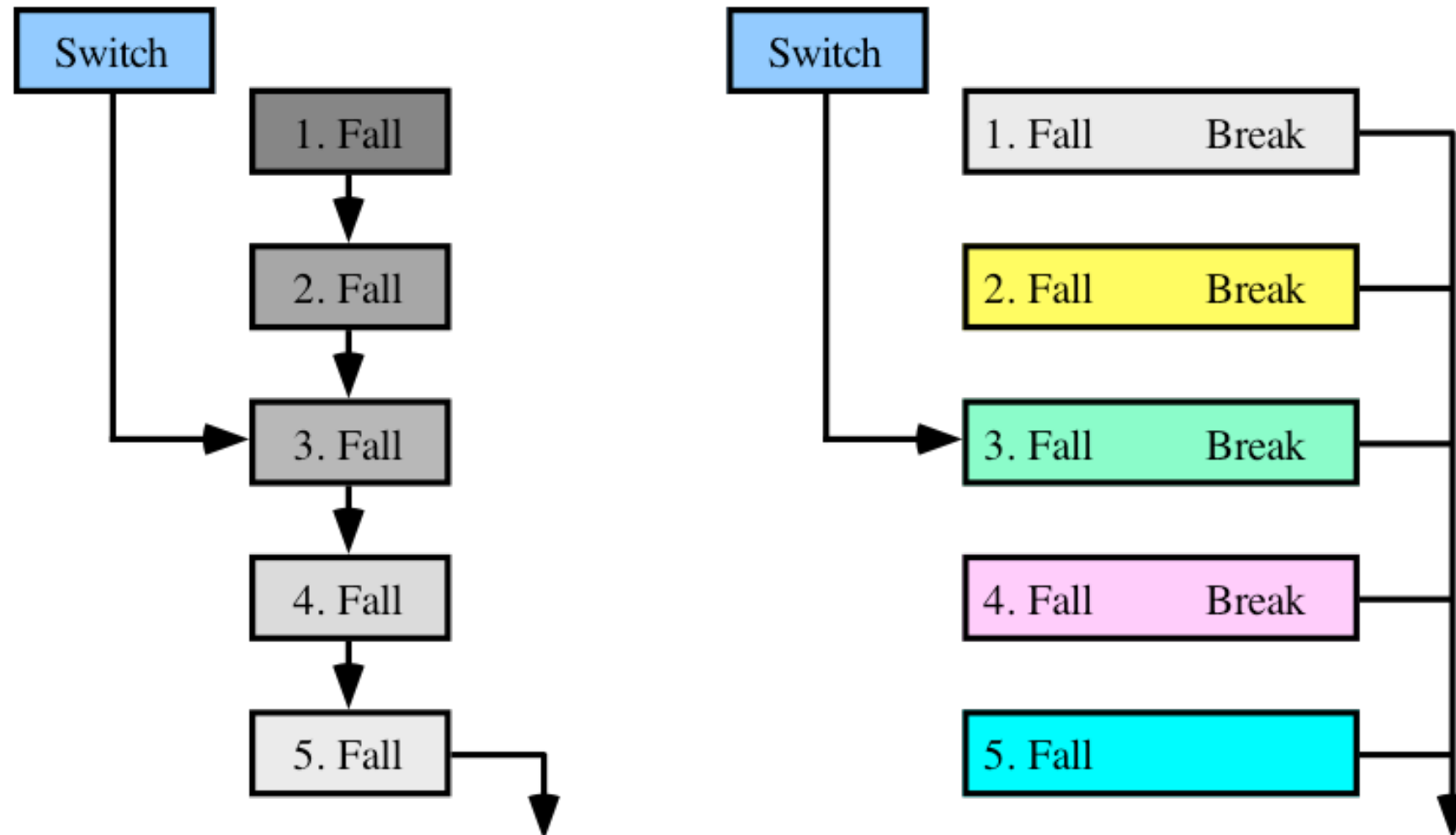
- Beispiel: Rechnen

```
double evaluate(double op1,char operator,double op2)
{  switch (operator)
    {  case '+':  return op1 + op2;
      case '-':  return op1 - op2;
      case '*':  return op1 * op2;
      case '/':  return op1 / op2;
      default:  printf("Aetsch, Fehler!!!");
    }  return 0; }
```

- schlecht: alle folgenden Statements werden auch ausgeführt

```
double evaluate(double op1,char operator,double op2)
{  double erg=0;
   switch (operator)
   {  case '+':  erg = op1 + op2;
     case '-':  erg = op1 - op2;
     case '*':  erg = op1 * op2;
     case '/':  erg = op1 / op2;
     default:  printf("Aetsch, Fehler!!!");
   }  return erg ; /* ist immer op1/op2 oder 0 */ }
```

- **break** ist wichtiges Element von **switch**
 - verhindert Ausführung der restlichen Fälle



```
switch (eingabe)
{
  case 'A':  ausfuehren(); break;
  case 'B':  beenden();   break;
  case 'D':  drucken();   break;
  default :  printf("Fehlerhafte Eingabe");
}
```

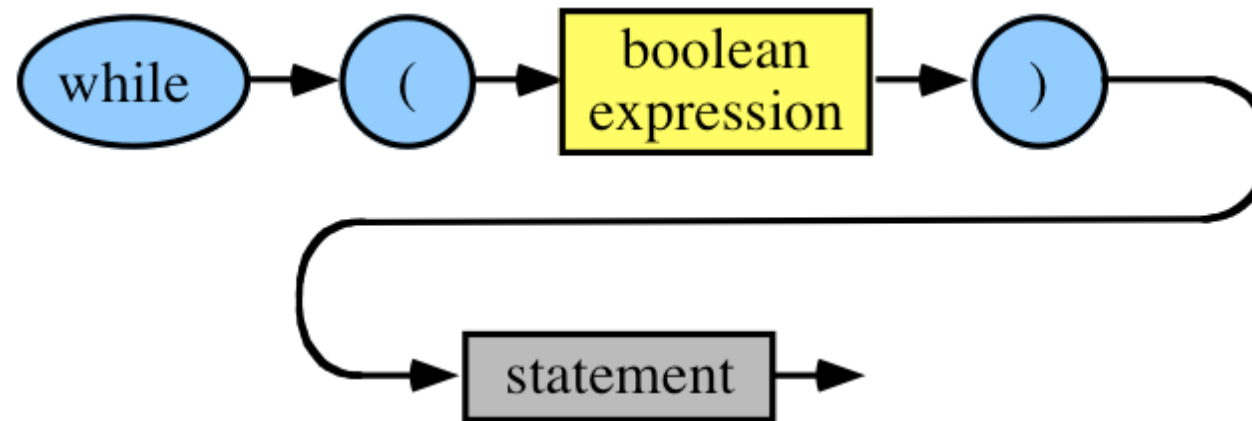
- Verbessertes Beispiel

```
double evaluate(double op1,char operator,double op2)
{ double erg=0;
  switch (operator)
  { case '+': erg = op1 + op2; break;
    case '-': erg = op1 - op2; break;
    case '*': erg = op1 * op2; break;
    case '/': erg = op1 / op2; break;
    default: erg = 0; printf("Aetsch, Fehler!!!");
  } }
```

- Beispiel: einfaches Menue

```
printf("Bitte waehlen Sie:\n S = Sichern\n \
      L = Laden\n N = Neu Anlegen");
scanf("%c",&eingabezeichen);
switch (eingabezeichen)
{ case 'S': case 's': savefile(); break;
  case 'L': case 'l': loadfile(); break;
  case 'N': case 'n': newfile();
}
```

- Schleife



- Bedingung wird am Anfang und nach jedem Durchlauf geprüft

- Beispiel n! (Fakultät, $2*3*4*5*...*(n-1)*n$)

```
fak = 1; i = 1;
while (i <= n)
    {fak = fak * i; i = i + 1;}
```

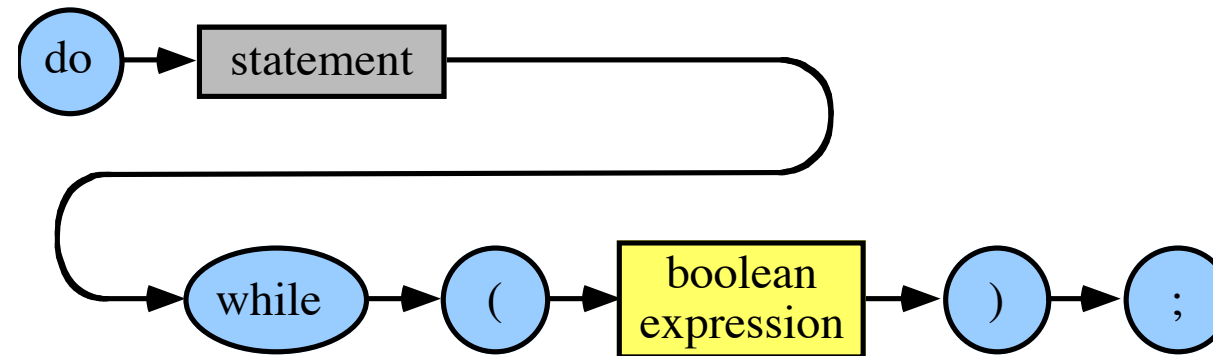
- Beispiel: Leerstellen in der Eingabe zählen

```
printf("Satz eingeben: \n");
ch = getchar();
while (ch != '\n')
{   if (ch == ' ') num_of_spaces++;
    ch = getchar();}
printf("Anzahl Leerstellen: %d", num_of_spaces);
```

- Schleife mit Test am Ende

- erst ausführen, dann testen

- => 'statement' wird mindestens einmal ausgeführt



- Beispiel: nochmal n!

- fak = 1; i = 1;

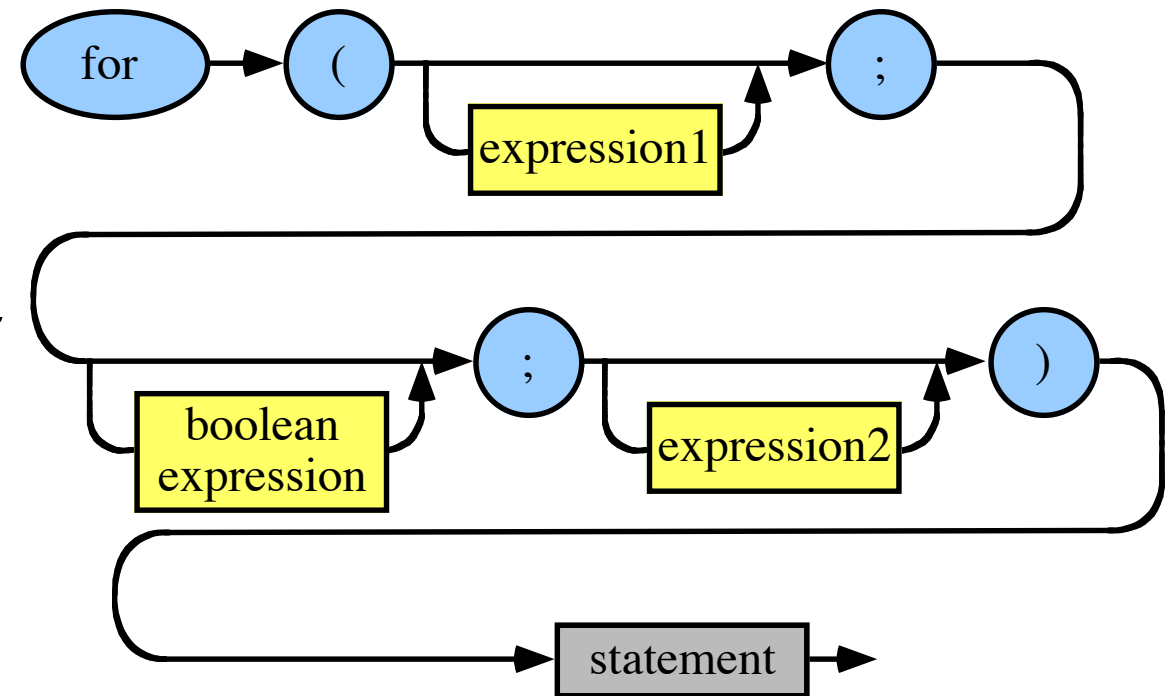
- do

- { fak *= i; i ++;

- } while (i <= n);

- Zählschleife

- expression1 initialisiert Schleifenzähler und andere Variable
- boolean expression entscheidet über Ausführung
- expression2 wird **nach** der Ausführung von Statement ausgeführt
- sollte Schleifenzähler inkrementieren



- Beispiel: schon wieder n!

```
fak = 1;
```

```
for (i = 1; i <= n; i+=1) fak = fak * i;
```

```
/* oder */
```

```
for (fak=1,i = 1; i <= n; ) fak = fak * i++;
```

- 3 Expressions bieten viele Möglichkeiten

```
for (fak = i = 1; n-i++; fak *= i );  
for (sum = i = 0; n-i++; sum += i );  
/* hacker's paradise */
```

- Vorsicht mit der Anzahl

```
for (i=0; i<n; i++) /* n-mal ausgeführt*/  
for (i=0; i<=n; i++) /* (n+1)-mal ausgeführt*/
```

- Leerzeichen überspringen

```
void skipspaces ()  
{ int ch = ' '; /* kleiner Trick */  
  for (; ch==' '; ch=getchar())  
    ; /* Null Statement */  
  ungetc (ch, stdin);  
}  
/* wollen wir nicht doch lieber while verwenden? */
```

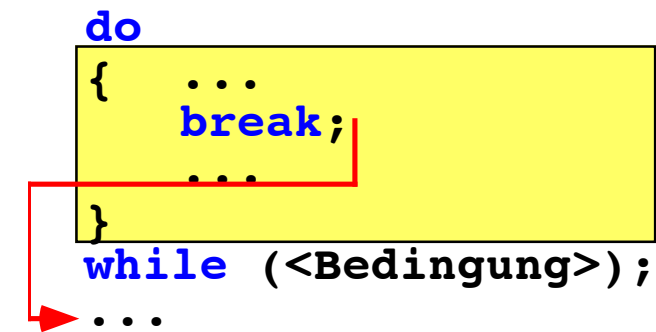
- Geschachtelte Schleifen

- innere Schleife vollständig ausgeführt

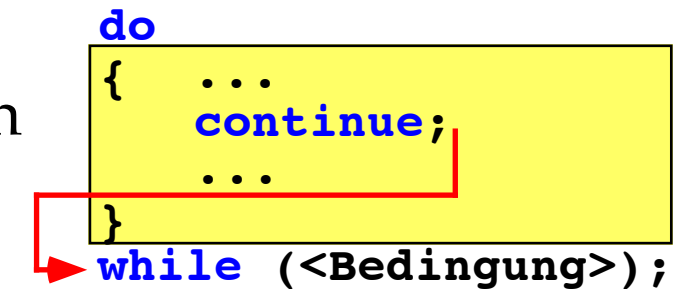
- mehrstufig

```
int main ()
{
    int j,k;
    printf("      1   2   3   4   5   6   7   8   9   10\n");
    printf("      -----\n");
    for (j=1; j <= 10; j++)
    {
        printf("%5d|",j);
        for (k=1; k<=10; k++)
            printf("%4d", j*k);
        printf("\n");
    }
    return 0;
}
```

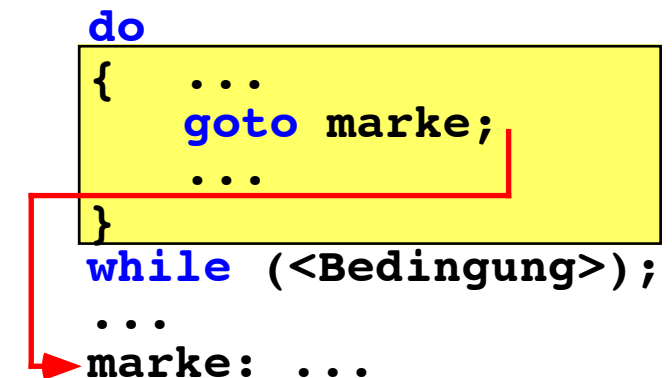

- Sprünge: the good, the bad and the ugly
 - break, continue, goto <label>
 - direkte Einflussnahme auf den Kontrollfluss
 - Programm kann auch anders formuliert werden



- **break** bricht ab
 - umfassende Schleife
 - switch
 - Ausführung mit Statement nach Schleife / switch fortsetzen
 - immer nur die jeweils innerste Schleife (bzw. switch)



- **continue** setzt fort
 - bleibt in der Schleife
 - Schleifentest + evtl. nächste Iteration
- **goto** springt zu einer Marke
 - Marke definiert: `ich_bin_ein_marke: <statement>`
 - `goto ich_bin_ein_marke;`
 - "goto fail"

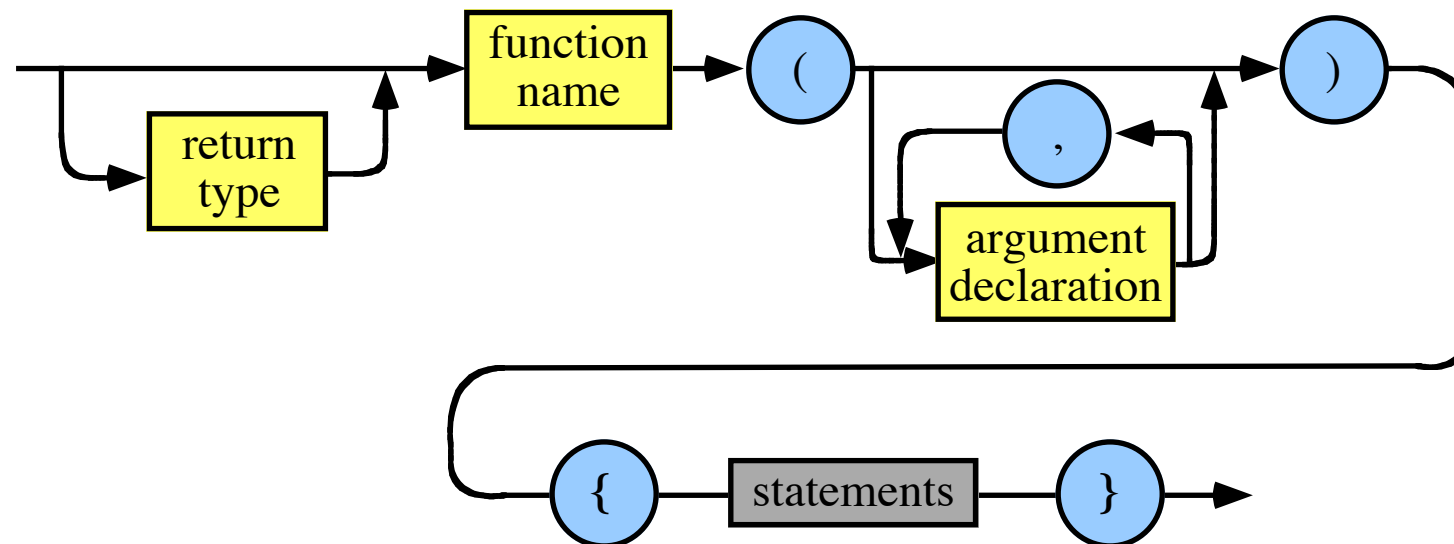


*Go To Statement Considered Harmful;
E. Dijkstra; CACM, März 1968.*

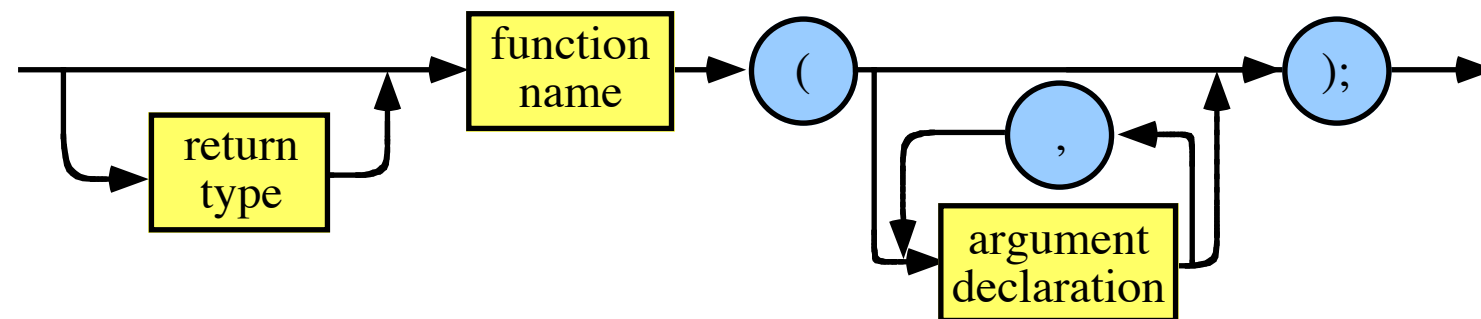
Funktionen im Detail

- Definierte Aufgaben gesondert programmieren
 - stepwise refinement
 - Problem zerlegen
 - häufig gebrauchte Komponenten
 - Werkzeugkasten
 - nur aktuelle Werte unterscheiden sich
 - Abstraktion
- Definition
 - Anzahl und Typ der Argumente
 - Statements im Rumpf der Prozedur

```
int max (int a, int b)
{
    if (a>b) return a;
    else return b;
}
```



- Keine Funktionen in Funktionen
- Deklaration vor der ersten Verwendung
 - falls Funktion vor der Deklaration aufgerufen wird
 - oder in Header-Dateien



```
int f2 (int m, int n, int p); /* Deklaration */
```

```
...
```

```
int f1 (int a, int b)
```

```
{ ...
```

```
    f2(a, 13, b+3);
```

```
    ... }
```

```
...
```

```
int f2 (int m, int n, int p) /* Definition */
```

```
{ ...
```

```
    f1(m, 13);
```

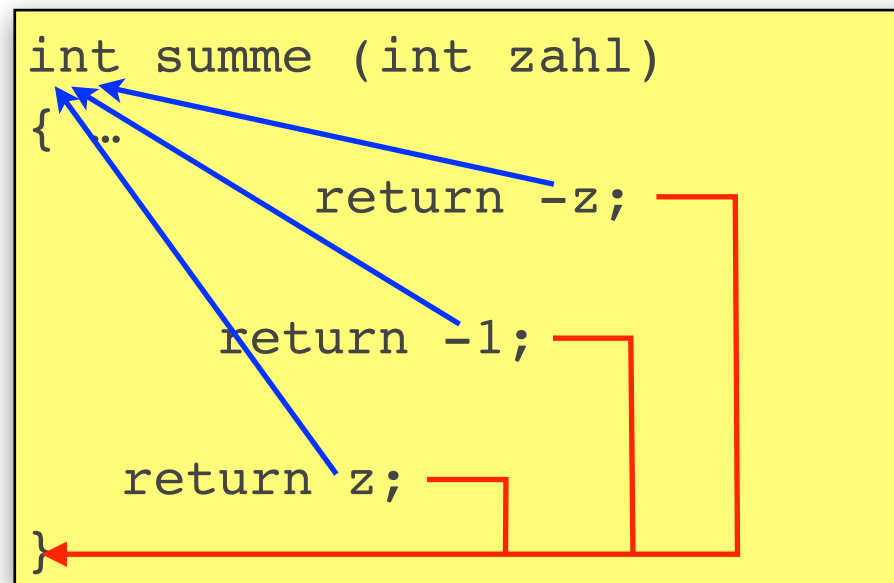
```
    ... }
```

- Parameter
 - automatische Konvertierung ähnlich wie bei Zuweisung
 - es werden Werte übergeben
 - keine Variablen!
 - Werte werden in 'neue' Variablen gelegt
 - **Trennung Wert-Variable**
- Beispiel Werte-Parameter

```
void f(int argument)
{ argument = 3;}

int main ()
{  int a = 2;
   f(a);
   printf("%d\n",a);  /*wetten,dass 2 gedruckt wird?*/
   return 0;
}
```

- Rückgabewert
 - mit Funktionsresultat
 - return-Statement
 - Funktion wird sofort verlassen
 - mit Ergebnissrückgabe



- Resultate mit Seiteneffekt erzeugen
 - Parameter als Zeiger auf die Variablen
 - zeigen auf 'Behälter' für Rückgabewerte

- Beispiel: Vertauschen von 2 Variablen

```
void swap (int *x,int *y)
```

```
{  int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

```
main()
```

```
{  int a=2, b=3;
   swap(&a,&b);
   printf("a ist %d\t b ist %d\n",a,b);
}
```

- Siehe scanf()

```
int f (int z)
{  z = 42;
}
```

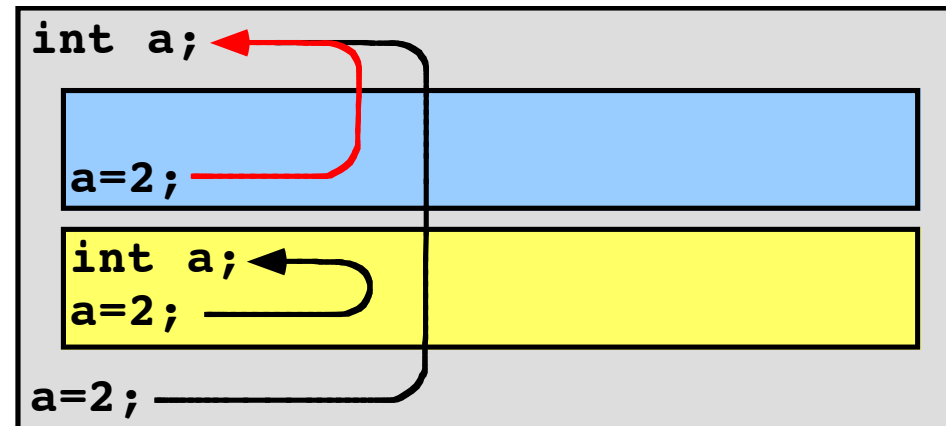
z 42

```
int f (int *z)
{  *z = 42;
}
```

```
int main ()
{  int platz;
   ...
   f(&platz);
   ...
}
```

platz 42

- Gültigkeit der Namen
 - Parameter und lokale Variablen
 - lokal gültig
 - überdecken weiter aussen definierte Namen



- void
 - Funktion ohne Resultat
 - Spezialtyp als Füllelement
- ```
void licht_an(int welches)
{...}
```

- Ein komplettes Programm

```
#include <stdio.h>
```

```
int summe (int zahl)
```

```
{ if (zahl >0) return zahl + summe(zahl-1);
```

```
 else return 0;
```

```
}
```

```
void allesummen (int max)
```

```
{ int lauf = 0;
```

```
 for (lauf = 1; lauf <= max; lauf++)
```

```
 printf("%3d: %6d\n",lauf,summe(lauf));
```

```
}
```

```
int einlesen()
```

```
{ int zahl;
```

```
 printf("Bitte Obergrenze eingeben:");
```

```
 scanf("%d",&zahl);
```

```
 return zahl;
```

```
}
```

```
int main ()
```

```
{ allesummen(einlesen()); return 0;}
```



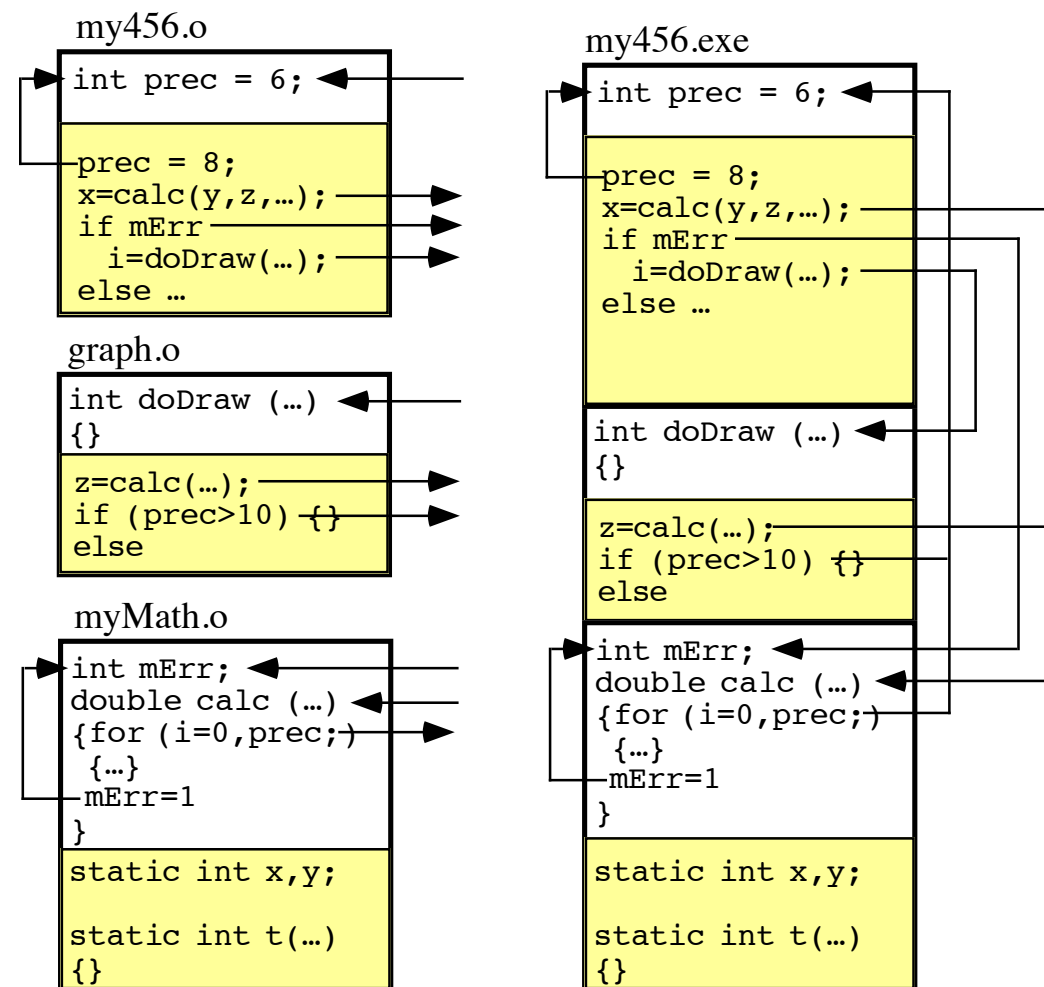
## Module

- Programme auf mehrere Dateien verteilen
  - Strukturierung des Problemes
  - Gruppenarbeit
  - getrennte Übersetzung
- Schnittstelle definiert
  - Leistungen des Modules
  - nach außen sichtbar
  - Typen und Variablen
  - Funktionen
  - Implementierung und Details versteckt
- Bekanntgabe an andere Module
  - Datei `regler.h` enthält Schnittstelle
  - Datei `regler.c` enthält Implementierung
  - Import mit include-Pseudobefehl: `#include <modul.h>`
- Übersetzung von Modulen
  - Parameter überprüfen,
  - Modul-Code und Referenzlisten erzeugen

- Zusammensetzen (Linken)
  - Zusammenkopieren der Module
  - Auflösen externer Referenzen: Einsetzen von Speicheradressen

Compiler

Linker

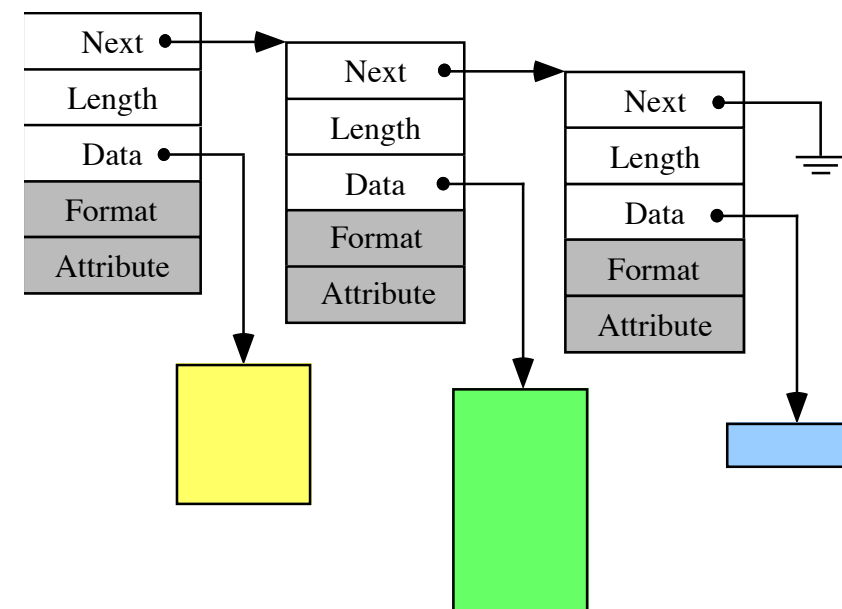
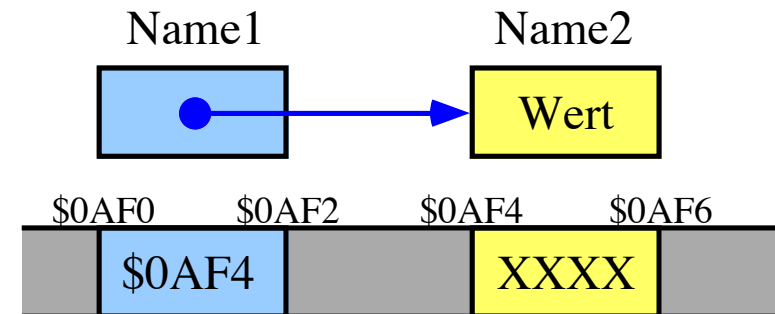


- Laufzeitumgebung: besondere Module
  - kommen mit dem Compiler
  - gehören oft zum Betriebssystem
  - standardisiert: ANSI, ISO, POSIX, ...
- Vorgefertigte Funktionen
  - Standardwerkzeuge
  - Mathematische Funktionen
  - Ein / Ausgabe
  - String-Verarbeitung
  - Datum und Zeit
- Mathematische Funktionen: math.h  
sqrt(), log(), exp(), sin(), cos (), ...
- Ein / Ausgabe: stdio.h
  - scanf(), printf(), getchar()
  - Datentyp FILE => Dateien
- String-Verarbeitung siehe unten

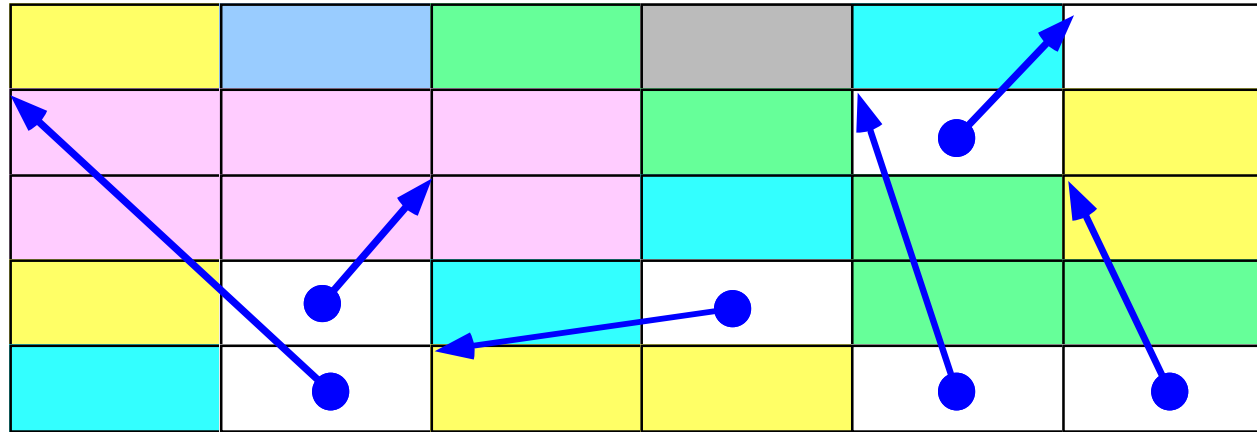
# Arrays und Pointer

## Zeiger (Pointer)

- Namen
  - identifizieren Objekte
  - von Menschen benutzbar
  - DNS: [www.froitzheim.com](http://www.froitzheim.com)
- Adressen
  - von Maschinen manipulierbar
  - schwer von Menschen verständlich
  - IP-Nummer: 134.60.77.64
  - 415-653-9516
- Referenzen
  - Verweis (Hinweis) auf andere Elemente
  - Bsp: Stellvertreter, Anrufumleitung
  - Hypertext-Referenzen
  - Listen, Bäume, ...
  - Strukturinformation im Textprogramm
  - Alias / Verknüpfung (z.B. Windows 95, 98)



- Zeigervariablen
  - zeigen auf beliebige Datenobjekte



- '\*' in Deklarationen und Definitionen

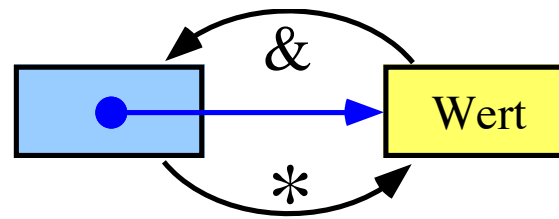
```
int *p; /* kein Speicherplatz für int! */
char ach, bch, *pch;
```

...

```
p = (int*)98; /* Wo mag das wohl sein? */
p = p+1; /* nun zeigt p auf Speicher 100 */
 /* oder auf 102? */
 /* Antwort: 98 + sizeof(int) */
```

- Address-of Operator '&'

```
char ach, bch, *pch;
ach = 'X';
pch = &ach;
scanf("%f", &messwert)
```



- Dereferenzierung mit '\*'

```
printf("pch zeigt auf: %d", *pch); /* druckt X */
pch = (char*)100;
printf("Speicherzelle 100: %d", *pch); /* druckt ? */
```

- Bedeutung von '\*' entgegengesetzt in:

- Definition / Deklaration: 'pointer-to'
- Expression: 'content-of'

- Leerer Zeiger == NULL

```
if (next == NULL) printf("Fertig");
```

- Pointer als Parameter

- ermöglichen Rückgabe von Resultaten

```
int raten(int *wert, int min, int max)
{ int try = 0;
 do { printf("\nBitte Eingabe: ");
 scanf("%d", wert);
 try = try + 1;}
 while (*wert < min || *wert > max);
 return try;
}
int eingabe;
/* Aufruf */
punkte = punkte + raten(&eingabe,17,99);
printf("der Kandidat hat %d geraten", eingabe);
```

- Übergabe großer Datenmengen ohne Kopie

- Kopien kosten Zeit und Platz
- aber: Veränderung in der Funktion evtl. unerwünscht

# Arrays

- Arrays sind Vektoren, Matrizen,

|   |    |    |   |   |   |     |     |    |    |    |   |    |   |     |
|---|----|----|---|---|---|-----|-----|----|----|----|---|----|---|-----|
| 7 | 12 | 73 | 0 | 0 | 0 | 123 | 456 | 13 | 13 | 12 | 9 | 12 | 9 | ... |
|---|----|----|---|---|---|-----|-----|----|----|----|---|----|---|-----|

- Definition

```
<typ> <name> "[" <dimension> "]"
 {"[" <dimension> "]"};
```

- Beginn immer bei 0
- Dimension ist Längenangabe

```
int vektor [100];
float matrix [zeilen][spalten];
```

- Initialisierung

```
int rot [3] = {255,0,0};
char name [32] = "Hallo"; /* 32 byte */
char name2 [] = "Hallo"; /* 6 byte */
```

- Verwendung

- meist elementweise

```
vektor[index]
matrix[zeilenindex][spaltenindex]
```



- Dualität Array - Pointer

- Array-Ident zeigt auf erstes Element
- zunächst das Erste (array[0])
- abgekürzte Schreibweise

vektor[0]      entspricht:    \*vektor

&vektor[0]    entspricht:    vektor

vektor + <expr> entspricht: &vektor[<expr>]

\*(vektor + <expr>) entspricht: vektor[<expr>]

vektor + index

&vektor[5]-2

/\* aequivalent \*/

matrix[i][j]

\*(matrix[i] + j)

\*(\*(matrix + i) + j)

- Länge des Grundtyps wird beachtet

- Als Parameter für Prozeduren
  - Prozedurdefinition ohne Wissen über Länge
  - Länge fest oder als Parameter

```
void feldlesen(char *dasFeld, int feldlength)
{
 int index;
 for (index=0; index < feldlength; index++)
 {printf("Element %d eingeben: ",index);
 scanf("%c", &dasFeld[index]);}
}
```

```
char einVektor[100];
```

```
int main()
{
 feldlesen(einVektor,100);
 drucken(einVektor,100); /* noch programmieren */
 return 0;
}
```

- Strings

- char-Array: `char name[32];`
- Adressierung wie bei Arrays
- Ende des Strings durch `'\0'` ('null terminated')

- Stringkonstanten in "..."

```
* "Konrad" entspricht 'K'
* "Konrad"+2 entspricht 'M'
* ("Konrad"+2) entspricht 'n'
"Konrad"[6] entspricht '\0'
"Konrad" /* Adresse */
```

- Zuweisungen nur elementweise

```
name[0]='A'; name[1]='l'; name[2]='f';
name[3]='\0';
printf("%s\n",name);
```

```
...
Alf
```

- Funktionen zur Stringverarbeitung in string.h

- `strncpy`, `strlen`, `strcat`, `strcmp`, ...

- `strncpy` (`strcpy` gefährlich ...)

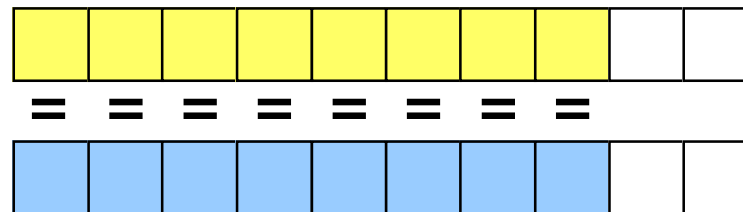
```
char name [32];
printf("%s",name);
strncpy(name, "Jeremias Jammermeier", sizeof(name)-1);
printf("%s\n",name+8);
```

```
...
Alf Jammermeier
```

- Idee für strlen

```
int strlen(char *theStr)
{ int length = 0;
 while (*theStr != '\0') /* 0 reicht auch */
 {length++; theStr++;}
 return length;
}
```

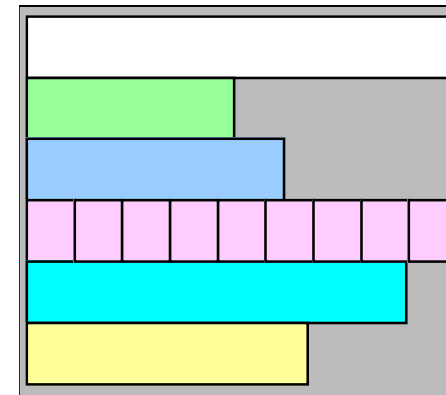
- Idee für strcmp



```
int strcmp(char *aStr, char *bStr)
{ if (strlen(aStr) != strlen(bStr)) return 0;
 while ((*aStr == *bStr) && *aStr)
 {aStr++; bStr++;}
 if (*aStr == '\0') return 1;
 else return 0;
}
```

## Structures

- C-Form von Records
  - Objektbeschreibung, Personaldaten
  - heterogene Datenelemente
  - besonders in Datenbanken
  - verschiedene Felder
  - evtl. Zeiger



- Typ `struct`

- Strukturdeklaration

```
struct Wagenbeschr
{
 char name[32];
 enum helptype {Schlaf, Abteil, ..., Kessel, ...} art;
 int Baujahr;
 struct Wagenbeschr *next;
};
```

- Variablendefinition

```
struct Wagenbeschr ShellWagen, BPWagen, Aralwagen;
```

- Zuweisung

```
strncpy(ShellWagen.name, "2-17 85003201", 31);
ShellWagen.art = Kessel;
ShellWagen.Baujahr = 1972;
ShellWagen.next = NULL;
Aralwagen = {"", Kessel, 1988, NULL};

BPWagen = Shellwagen;
```

- Vergleich nicht als Einheit

```
if (BPWagen == Shellwagen) /* falsch */
/* illegal structure operation */
```

- typedef macht eine bestimmte structure zum Typ

```
typedef struct Wagenbeschr WagenType;

WagenType EzzoWagen;

/* natuerlich auch fuer Standardtypen */

typedef double Gewicht;
Gewicht maximal_zulaessiges_Gesamtgewicht;
```

- Initialisierung

```
WagenType SpeiWa = {"", Speise, 1978, NULL};
WagenType PersWa = {.art = Personen,
 .baujahr = 1979}; /* rest 0 */
```

- Funktionsaufruf

- {} Auf Parameterposition möglich

- mit Typecast

```
typedef struct {char *name; int number} X;
```

```
void fn1(const X *x);
{...;}
```

```
fn1(&a); /* benannte Variable */
fn1(&(X){ "name", 42 }); /* anonyme Variable */
fn1(&(X){ .name="name", .number=42 });
fn1(&(X){ "name", random() });
fn1(&(const X){ .name="name", .number=42 });
/* anonyme Konstante */
```

- Pointer auf structures

```
WagenType *wagenPtr;
wagenPtr = &EsoWagen;
*wagenPtr = BPWagen;
```

- Zugriff auf Komponenten

- Problem: '.' bindet stärker als '\*'

```
/* falsch: printf("%s", *wagenPtr.name) */
(*wagenPtr).Baujahr = 1990;
```

- neuer Operator:

```
wagenPtr -> Baujahr = 1990;
printf("%s", wagenPtr -> name);
```

- Parameter und Resultat von Funktionen

```
WagenType *newcopy (Wagentype *derWagen)
{ WagenType *help;
 help = malloc(sizeof(Wagentype));
 if (help == NULL) printf("%s", "Out of memory");
 else *help = *derWagen;
 return help;
}
```



# Systematische Fehlersuche

- Software hat Fehler
  - komplexe logische Systeme
  - Programme mit 1 Million Zeilen und mehr
  - Interaktion mit anderen Programmen
  - unscharfe Eingabe
- Verifikation schwer evtl. unmöglich
  - unentscheidbare Probleme ( $\Rightarrow$  Goedel)
  - vollständige Aufzählung der Randbedingungen
- Programm ordentlich aufschreiben
  - systematisch einrücken
  - aussagekräftige Namen
- Programm oft ausführen
  - mit verschiedenen Parametern
  - auf verschiedenen Rechnern
  - andere Personen testen lassen
- "Debugging is great fun" [Anonymous]

*Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.*  
[B. Kernighan]

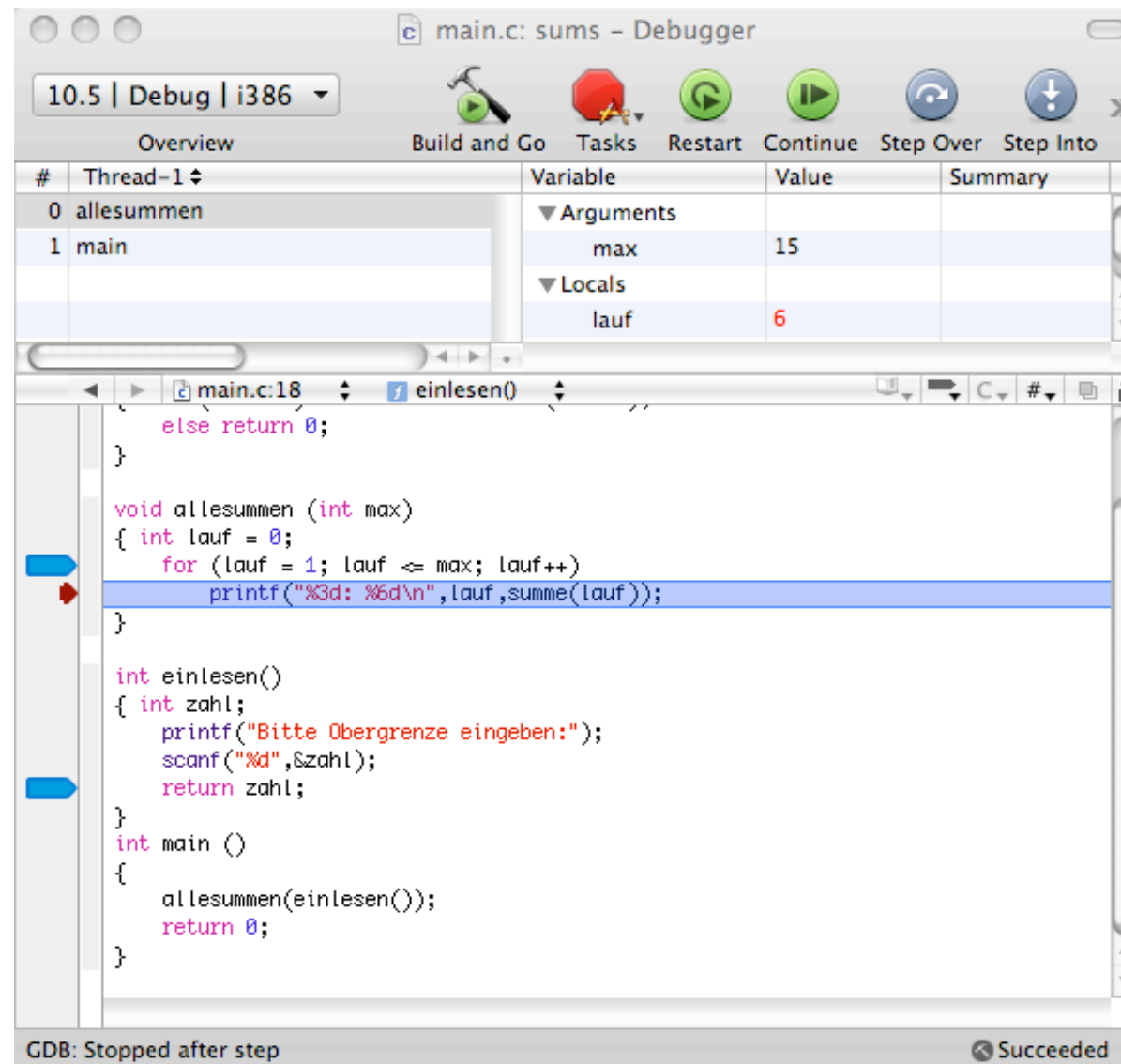
```
while (x == y) {
 something();
 somethingelse();
}
finalthing();
```

- Typische Probleme
- Bedingungen
  - Test auf Gleichheit: (a <= 0) oft besser als (a == 0)
  - "echt größer" (>) oder "größer-gleich" (>=)?
- while-Schleife
  - Endebedingung falsch
  - Seiteneffekte auf Endebedingung
- Variable nicht vorbesetzt
  - häufig bei erratischen Fehlern
 

```
int teiler;
scanf("%d",b);
a = b/teiler; /* Was steht wohl in teiler? */
```
  - Programm läuft nur einmal
  - Programm läuft nur nach bestimmten anderen Programmen
  - Programm läuft, wenn Variable eingefügt wird
- Pointer zeigt irgendwohin
  - NULL-Pointer, ...
  - Speicher wird überschrieben

- Systematische Vorgehensweise
  - Beobachtung des Problem (reproduzieren!)
  - Hypothesenbildung (warum?)
  - Messen (wenn - dann)
  - Beseitigen (programmieren)
- Wolfs-Zaun
  - Fehler eingrenzen
  - schrittweise eingrenzen mit printf()
- Haben die Variablen den richtigen Wert?
  - am Ende von Funktionen
  - nach jedem Statement
- Besondere Abfragen
  - Annahme: Variable sollte  $\neq 0$  sein  
`if (<variable>==0) printf("Annahme verletzt\n");`  
oder: `assert(<variable>!=0); /* aus assert.h */`
  - Fehlermeldung und evtl. Abbruch
  - assertion (Zusicherung)

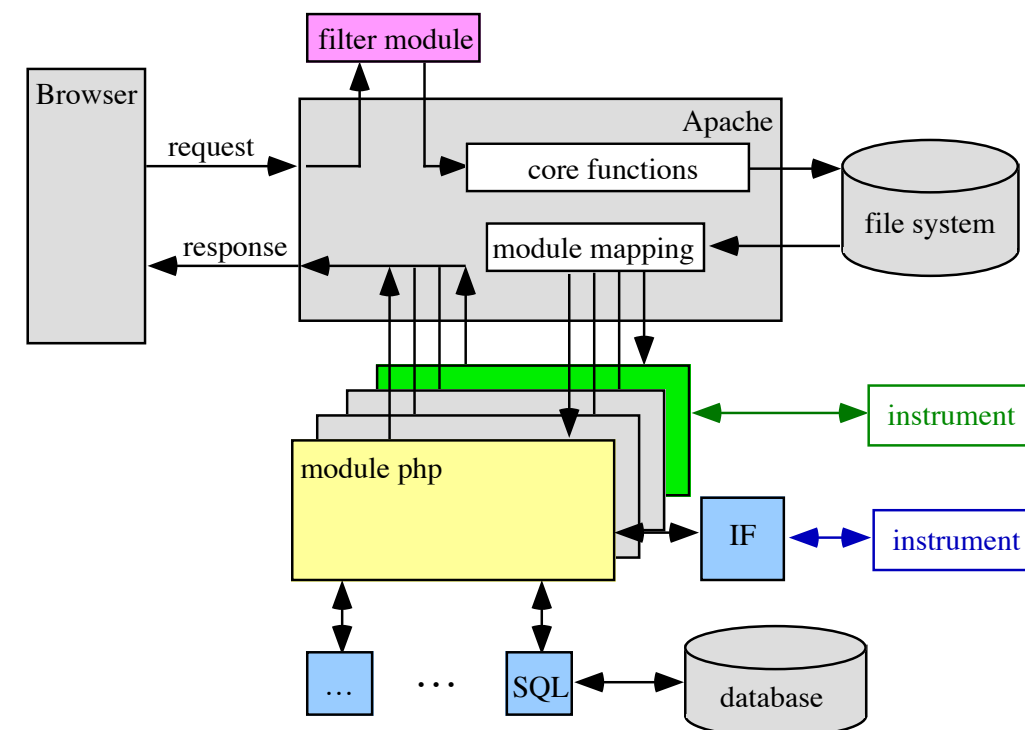
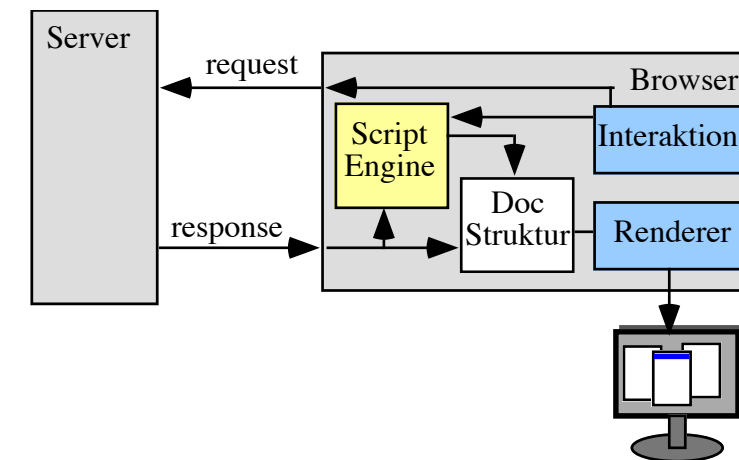
- Debugger
  - Variablen:Inspektion , Werte setzen
  - Aufrufkette
  - Breakpoints
  - trace
  - step



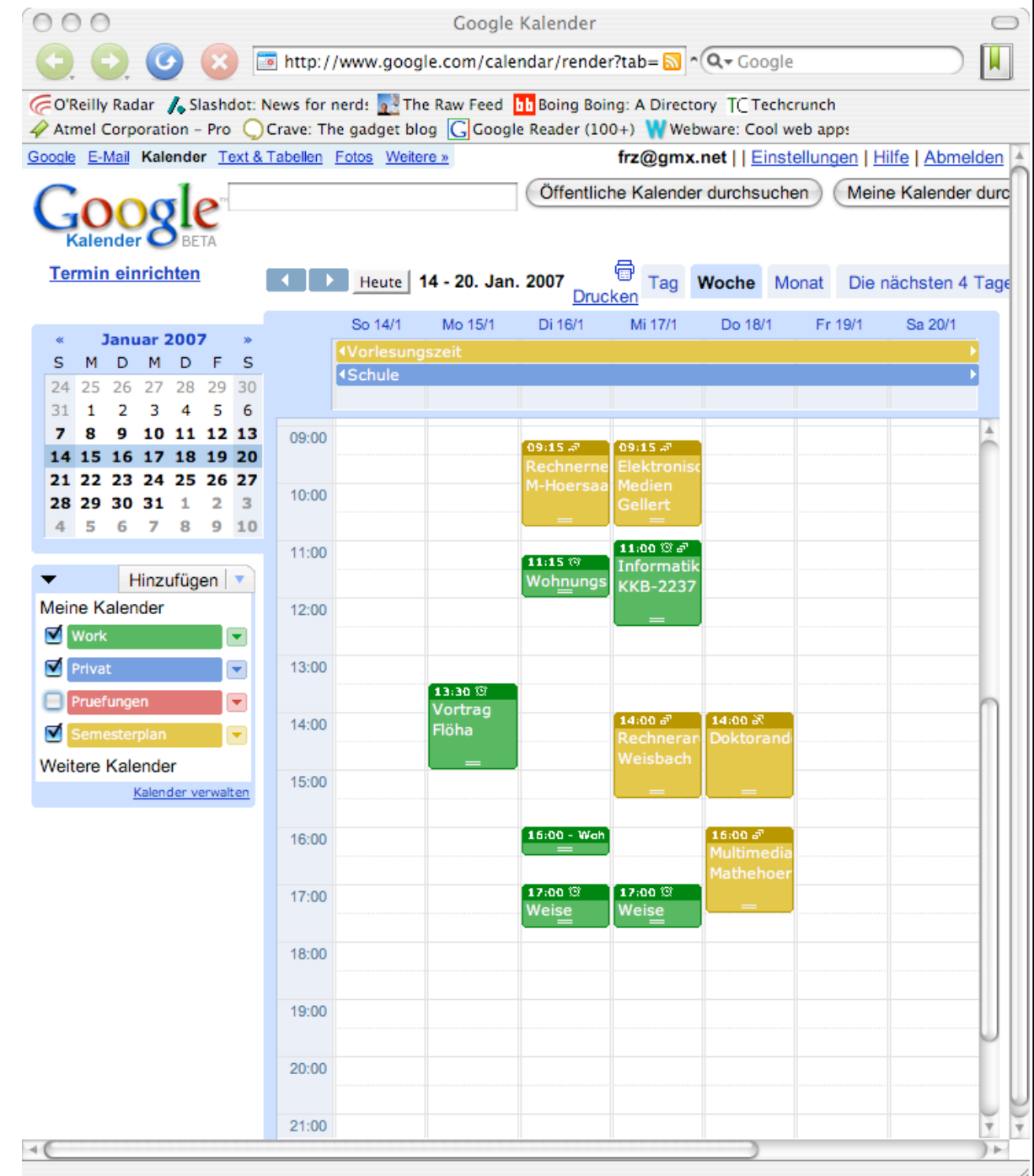
# Reaktives Programmieren

## Ausführungsmodell WebApplikation

- Mikro-interaktiv
  - Browser
  - html mit Grafiken als Substrat
  - individuelles Verhalten: JavaScript
- Server
  - Datenbank + Geräte
  - Apache und Apache-Module
- Scripte
  - client-side: Javascript, Java Applets
  - server-side: ASP, php, Java und Beans
- Beispiele
  - <http://rr.informatik.tu-freiberg.de>
  - Shopping-Seiten

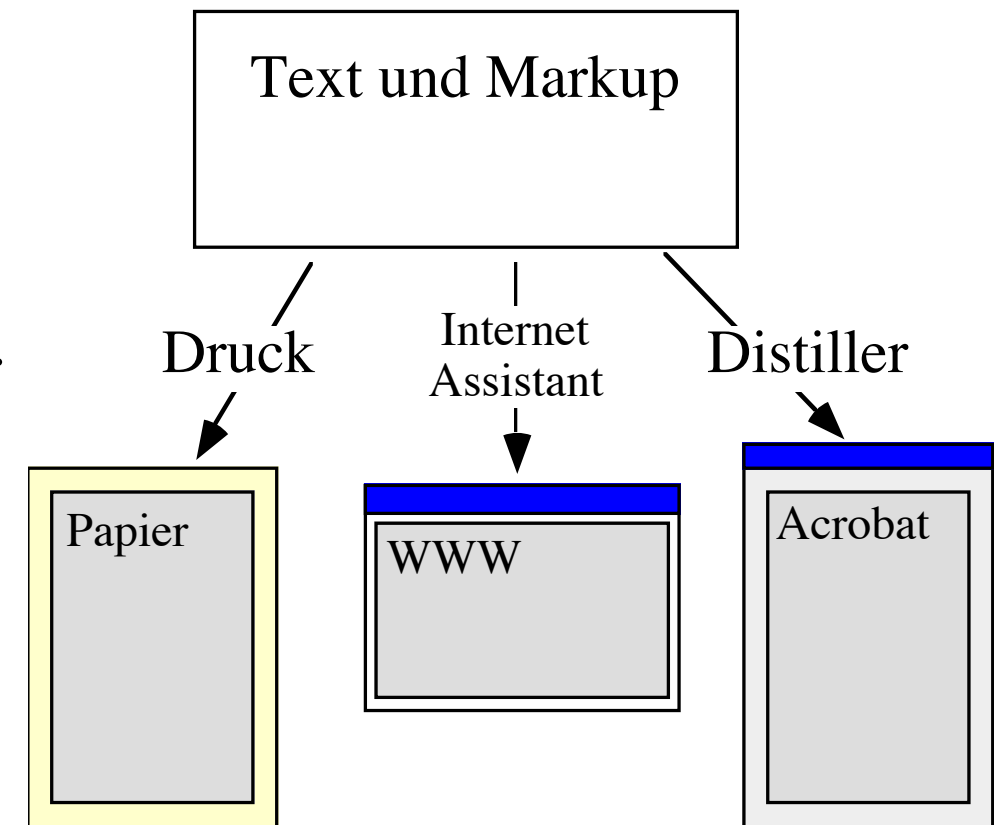


- Programmierparadigma
- Infrastruktur
  - Server:(Apache+php | JavaBeans)
  - Server:Datenbank
  - Client:(InternetExplorer | Firefox | Safari)
- Krümelware
  - kleine, vorgefertigte Bauteile (-> Objekte)
  - abgeleitete (erweiterte) Objekt
  - inkrementelles Programmieren
  - Einfügen und Erweitern statt Bauen
  - Gesamtkonzept? Architektur?
  - Datenfluß designen
- AJAX
  - Asynchronous JavaScript And XML
  - maps.google.com
  - flickr.com
  - docs.google.com, google calendar



## HTML, die 'Sprache' des WWW

- Hypertext Markup Language
  - Berners-Lee 1989
  - Zweck: Verknüpfung von Dokumentation der Hochenergiephysik
  - keine Bilder - textbasierte Klienten
  - Standard Generalized Markup Language
  - Document Type Definition (DTD) von SGML
  - Hypertext-Referenzen: URLs eingebettet in das Dokument
  - <http://www.w3.org/TR/REC-html40/>
- Markup
  - logische Struktur für Text
  - Überschrift, normaler Paragraph, Zitat, ...
  - Fußnote, Literaturverweis, Bildunterschrift, ...
- Zuordnung der Attribute beim Satz
  - Autor produziert Inhalt und Struktur
  - Drucker setzt
  - Corporate Identity ...
- HTML: ASCII-Text + <A>-Tag



- Beispieltext:

```
<HTML> <HEAD>
<TITLE> Ein HTML-Beispiel </TITLE>
</HEAD> <BODY>
Dies ist ein Hypertext Dokument.
<P>Mit einem Bild:
 und einem
 Hyperlink </P>
</BODY> </HTML>
```

**Dies** ist ein Hypertext Dokument.

Mit einem Bild:  
und einem [Hyperlink](#)



- Weitere Elemente siehe z.B. <http://www.selfhtml.org/>
  - Listen, Stile
  - Formatierung



## JavaScript

- Programmfragmente in HTML
  - Verbesserung von HTML-Seiten auf der Klienten-Seite
  - Fenstergröße und -Gestaltung
  - Menus, Effekte, ...
  - Beispiel: <http://maps.google.com>
- Interpreter im Browser
- Eingebettet in HTML
  - script-Tag

```
<html><head><title>Test</title>
<script language="JavaScript">
<!--
 alert("Hallo Welt!");
//-->
</script>
</head><body>
</body></html>
```

- Oder in anderen HTML-Tags

```
<html> <head>
 <title>JavaScript-Test</title>
 <script language="JavaScript">
 <!--
 function Quadrat(Zahl)
 {var Erg = Zahl * Zahl;
 alert("Quadrat von " + Zahl + " = " + Erg); }
 //-->
 </script> </head>
 <body> <form>
 <input type=button value="Quadrat von 6 errechnen"
 onclick="Quadrat(6)">
 </form> </body>
</html>
```

- Eventhandler

- Attribut in html-Tags
- beschreiben Ausführungsbedingung
- Aufruf einer JavaScript-Funktion
- onload, onclick, onmouseover, onkeydown, ...

- Sprache
  - Notation ähnlich Java
- Anweisungen
  - Zuweisungen

```
zahl = 0; zahl++; zahl+=1;
```
  - Bedingte Anweisungen und Schleifen

```
if (zahl<0) zahl = 0;
while (idx<100) {...; idx++;}
for(i = 1; i <= 100; i++)
 {...}
```
  - Funktionsaufrufe

```
alert("Und hier ein kleiner Hinweis");
```
  - Klammern mit {}

```
if (Ergebnis > 100)
{ Ergebnis =0; Neustart(); }
```

- Variablen: kein ordentliches Typenkonzept
  - Vereinbarung mit 'var'
  - Typ Zahl oder String
  - wird bei der ersten Zuweisung festgelegt

```
var Antwort = 42;
var Frage = "The Question for god ..."
```
  - global oder in Funktion lokal
- Objekte: Eigenschaften und Methoden
  - selbstdefiniert oder vordefiniert in Umgebung
  - window, document, images, links, array, ...
  - Objekt erzeugen mit 

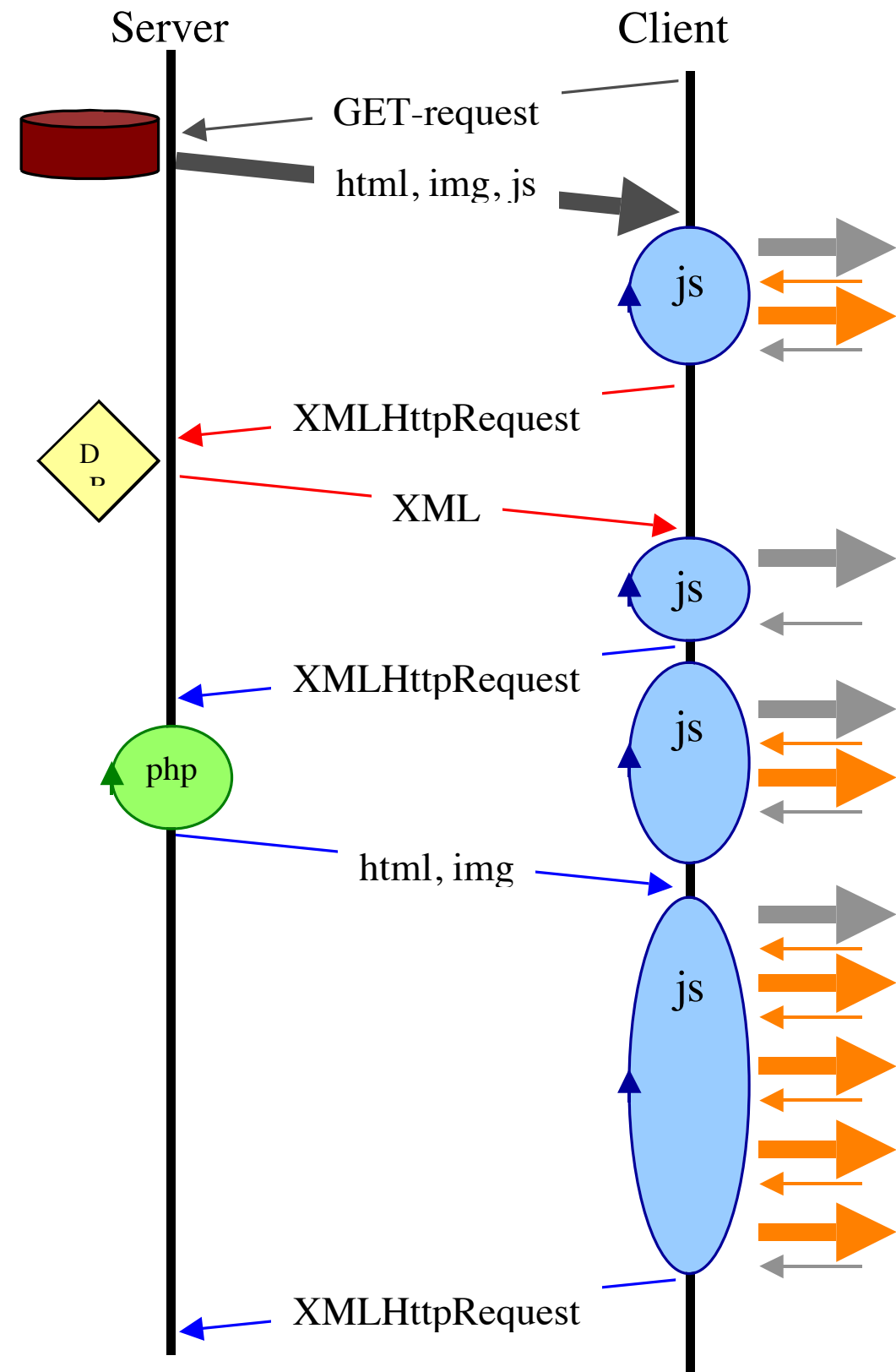
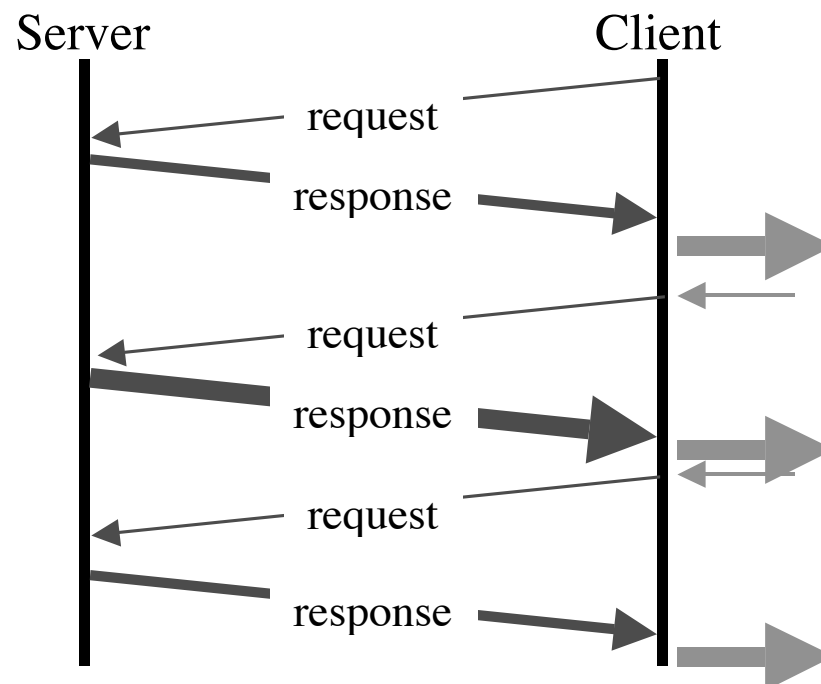
```
var einobjekt = new <object>;
var einFenster = window.open(<ref>,<titel>,<parms>);
```
  - Datenstruktur (Felder = Eigenschaften)

```
einFenster.name = "Lustig";
```
  - Methoden zur Manipulation: print, blur, moveTo, scrollBy, ...

```
einFenster.resizeTo(500,400);
```
- Funktionen
  - Anweisungsblock mit Variablen
  - Aufruf aus anderen Funktionen und in Eventhandlern

# AJAX

- Architektur von AJAX-Applikationen
- Programm im Browser (Client)
  - JavaScript
  - AJAX Libraries
- Server
  - Webserver, Datenbank
  - Serverside Scripte
- Leichtgewichtige Kommunikation
  - XMLHttpRequest
  - **synchron** und **asynchron**



- XMLHttpRequest
- JavaScript Klasse
  - erstmals in IE 5
  - Interface für Http
  - ohne Benutzerinteraktion
  - synchron und asynchron
- Properties
  - readyState, status
  - onreadystatechange
  - responseText
- Methoden
  - open
  - send

```
function createXMLHttpRequest() {
 try {return new ActiveXObject(
 "Msxml2.XMLHTTP"); } catch(e) {}
 try {return new ActiveXObject(
 "Microsoft.XMLHTTP"); } catch(e) {}
 try {return new XMLHttpRequest(); } catch(e) {}
 alert("XMLHttpRequest not supported");
 return null;}

```

```
var xhReq = createXMLHttpRequest();
xhReq.open("get", "sumget.phtml?num1=10&num2=20", true);
xhReq.onreadystatechange = function() {
 if (xhReq.readyState != 4) { return; }
 var serverResponse = xhReq.responseText;
 ...};
xhReq.send(null);

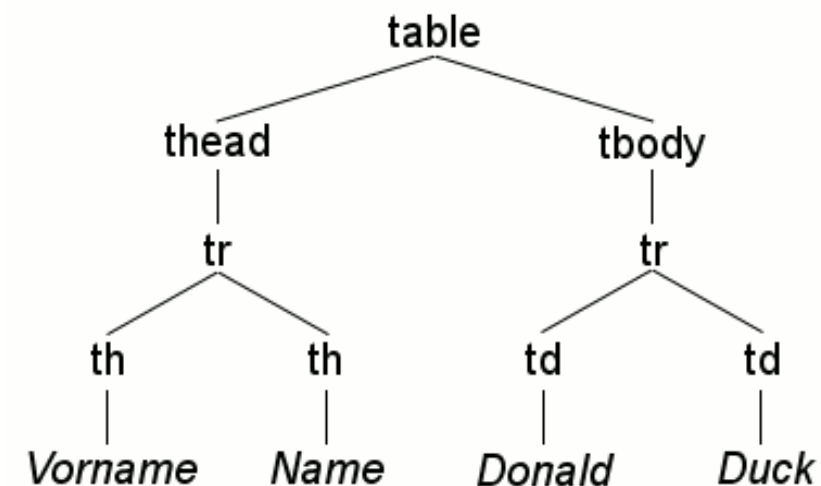
```

- Das X in AJAX
- Markup
  - Markup: Trennung Struktur - Inhalt
  - logische Struktur der Seite
  - Bsp: Überschriften, Absätze, Zitate, ...
- XML: eXtensible Markup Language
  - Syntax für Markup
  - Semantik in XSL oder CSS
- Document Object Model DOM
  - baumartige Struktur der Dokumente
  - Zugriff auf Dokumenteninhalte (=Objekte)
  - Inhalt, Struktur, Stil
- AJAX
  - XML als ein Transfer-Format für Inhalt
  - Manipuliert DOM-Knoten
  - Einfügen, Löschen, Ändern
  - Browser 'rendert' Dokument

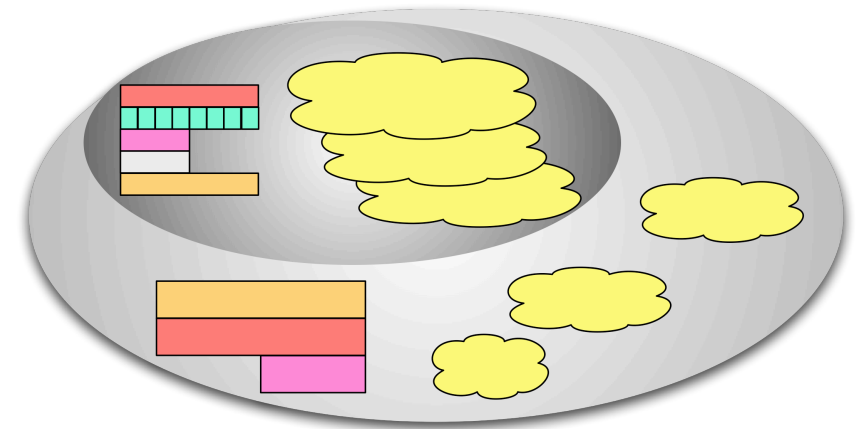
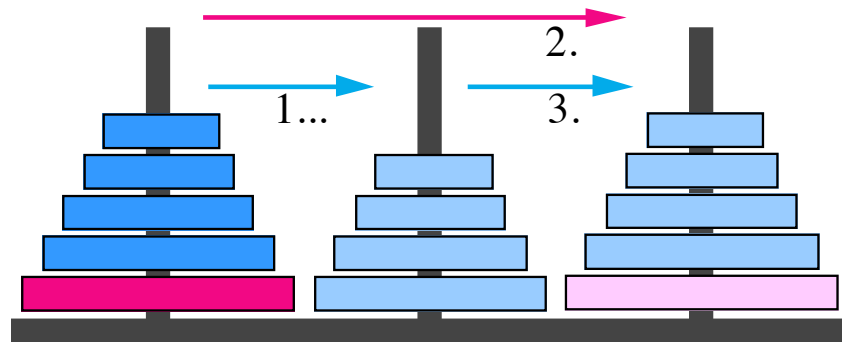
```

<table>
 <thead>
 <tr>
 <th>Vorname</th>
 <th>Name</th>
 </tr>
 </thead>
 <tbody>
 <tr>
 <td>Donald</td>
 <td>Duck</td>
 </tr>
 </tbody>
</table>

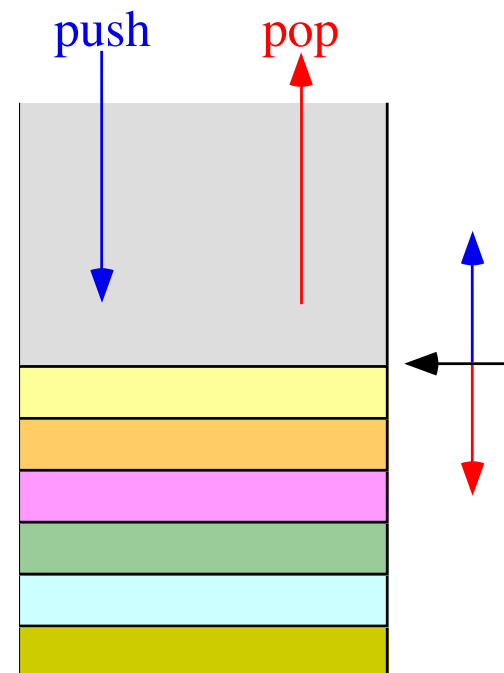
```



# Algorithmische Komponenten



- Sortieren
- Listen
- Bäume
- Speicherverwaltung
- Objekte und Patterns

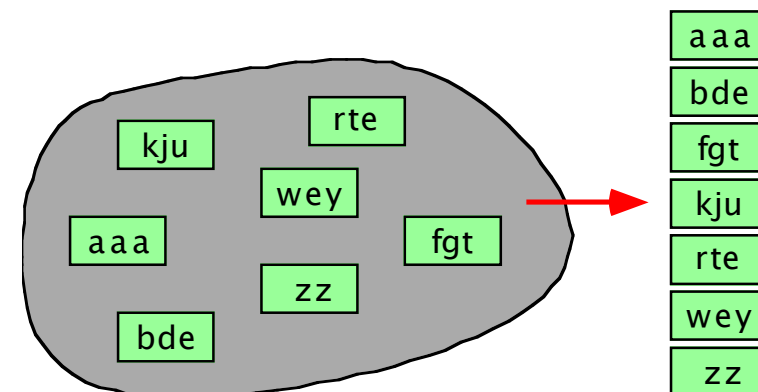
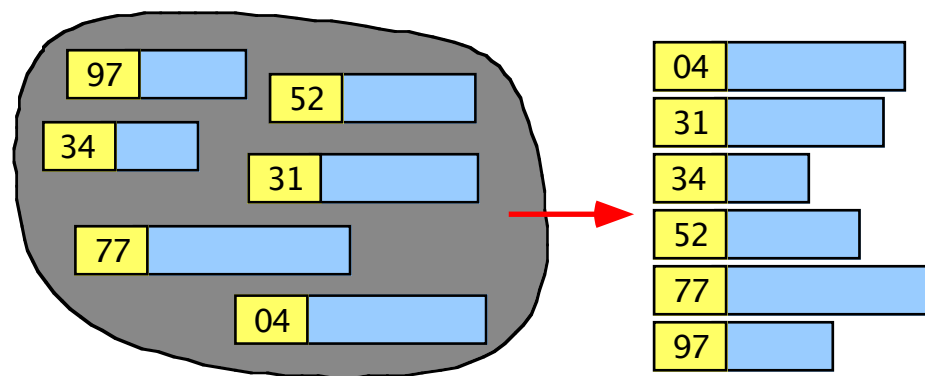




# Sortieren

## Traditionelles EDV-Verfahren

- algorithmische Fingerübungen
- Abschätzungen für Rechenaufwand
- Einfluss des Datenvolumens
- Problemstellung
  - ungeordnete Menge von Elementen,
  - evtl. nur sequentiell zugreifbar
  - Ordnung der Schlüssel gesucht

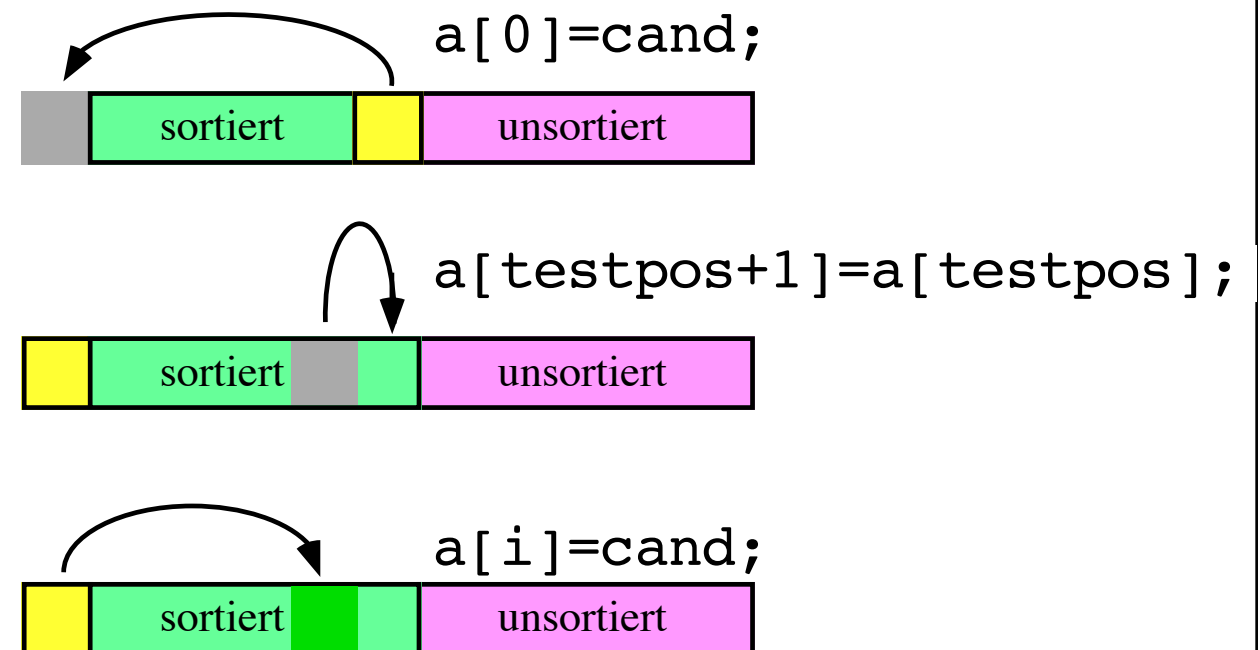


- Sortieren durch Einfügen
- Zu sortieren: item menge[N];
  - Kandidat im linken Teil einsortieren
  - entstandene Lücke mit schon sortierten Elementen auffüllen

```

void DirektesEinfügen(item *a, int N)
{
 item cand;
 int candpos, testpos;
 for (candpos=2, candpos<=N, candpos++)
 {
 cand= a[candpos]; a[0]= cand;
 testpos=candpos-1;
 while (cand < a[testpos])
 {
 a[testpos+1]=a[testpos];
 testpos= testpos-1;
 }
 a[testpos+1]=cand;
 }
}

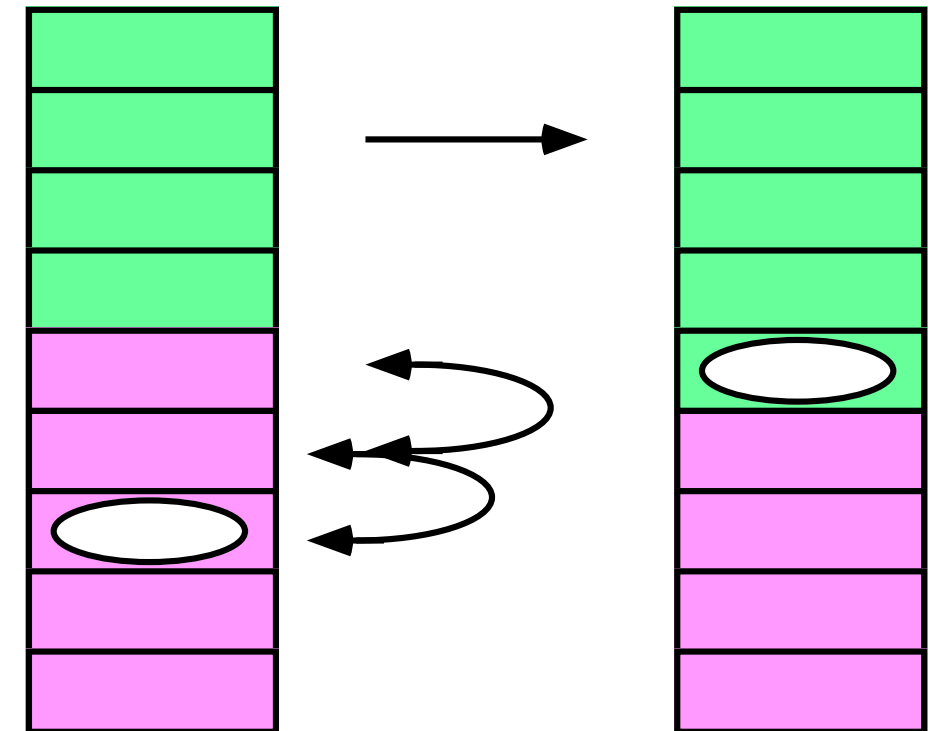
```



- Sortieren durch Vertauschen ("Bubble Sort")

- Benachbarte Elemente paarweise vertauschen, falls  $a[i] < a[i+1]$
- Analogie zu einer aufsteigenden Blase
- bis zu der ihrem 'Gewicht' entsprechenden Höhe

```
void bubblesort(item *a, int N) {
 item zwischen;
 int fertig, cand;
 for (fertig=2, fertig<N, fertig++) {
 for (cand=N, cand>=fertig, cand--) {
 if (a[cand-1]>a[cand])
 { zwischen=a[cand-1];
 a[cand-1]=a[cand];
 a[cand]=zwischen;
 }
 }
 }
}
```



- Verhalten der Elemente

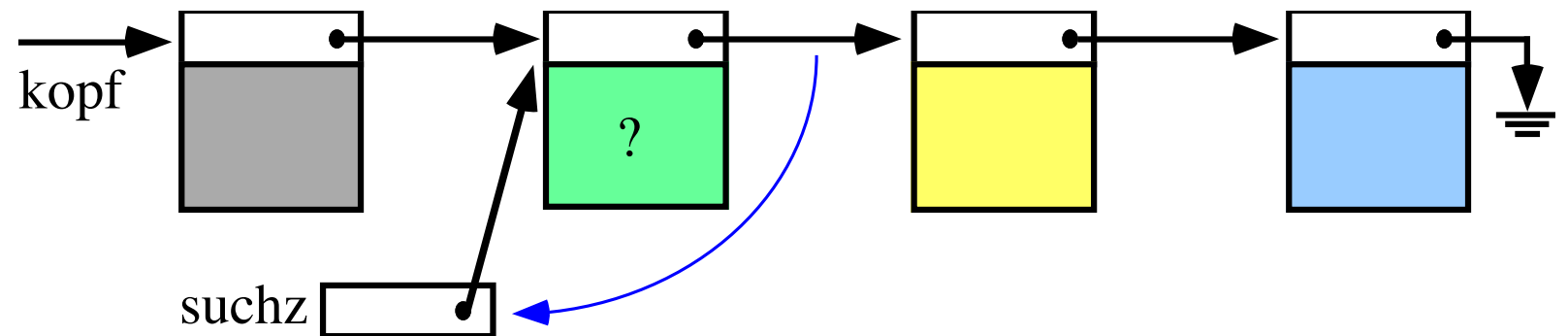
- leichtes Teilchen in einem Durchlauf ganz nach oben
- schweres Teilchen sinkt mit jedem Durchlauf nur um eine Stelle
- nur solange sortieren, bis kein Austausch mehr stattfindet
- "Shakersort" als Variante mit abwechselnder Durchlaufrichtung
- auch Bubblesort bewegt Elemente nur über eine kleine Distanz

# Listen

- Liste: Verkettete Records
  - linear: Liste
  - mehrweg: Baum

- Durchsuchen einer Liste

```
gefunden = NULL;
suchz = kopf;
while (suchz != NULL)
{ if (suchz->data==gesucht)
 {
 gefunden = suchz;
 break;
 }
 else suchz = suchz->nxt;
}
```

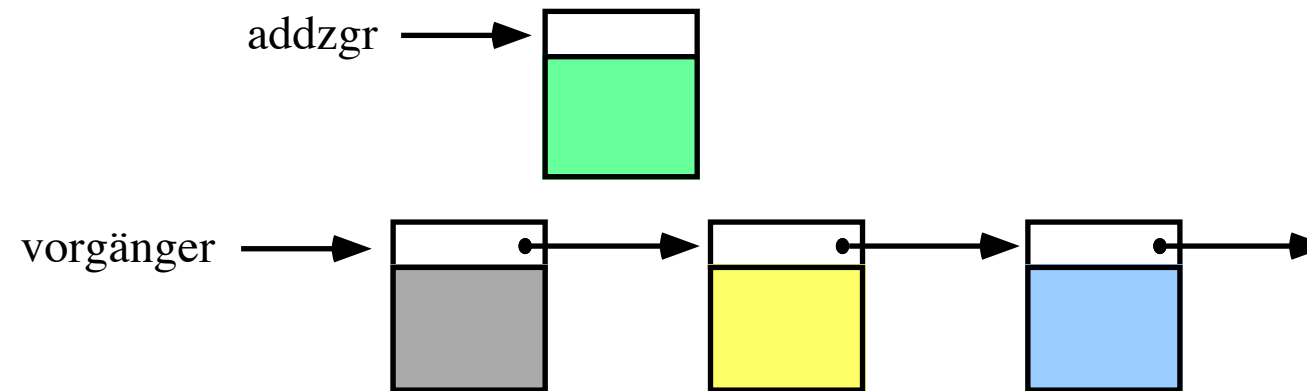


- Aufbau einer neuen Liste

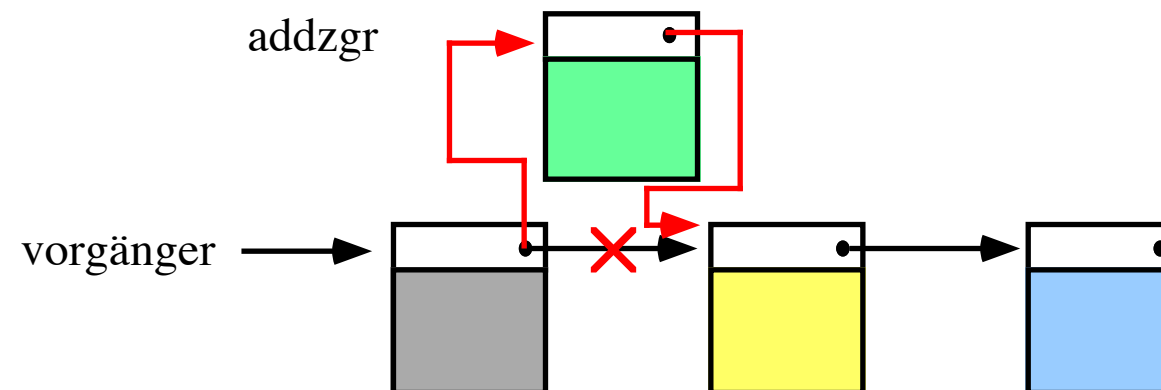
- Einlesen von content-Records
- Einfügen am Kopf der Liste

```
listelem *kopf, *addzgr;
content eingabe;
...
kopf = NULL; /* Liste leer */
while (elem_lesen(&eingabe))
{
 addzgr = elem_anlegen();
 addzgr->data = eingabe;
 addzgr->nxt = kopf;
 kopf = addzgr;
}
...
```

- Unmittelbar nach irgendeinem Vorgänger einfügen

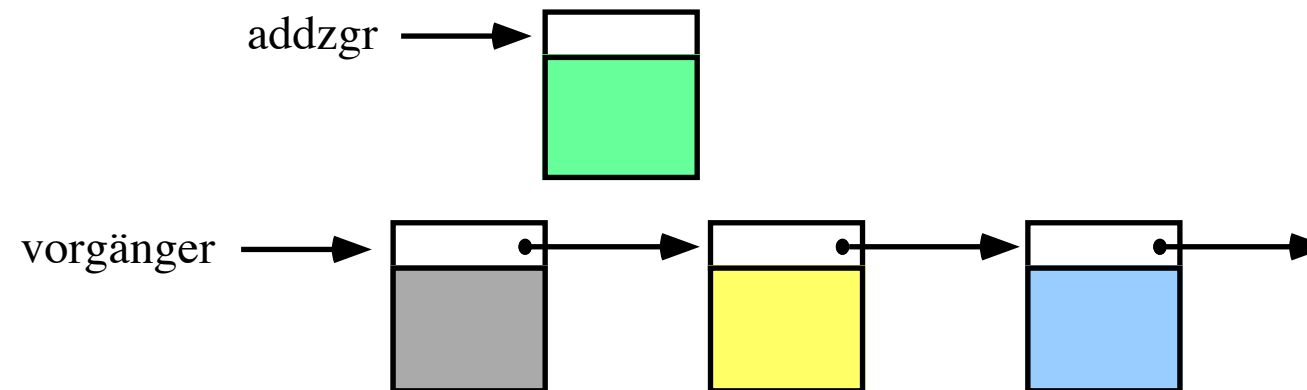


```
addzgr->nxt = vorgänger->nxt;
vorgänger->nxt = addzgr;
```

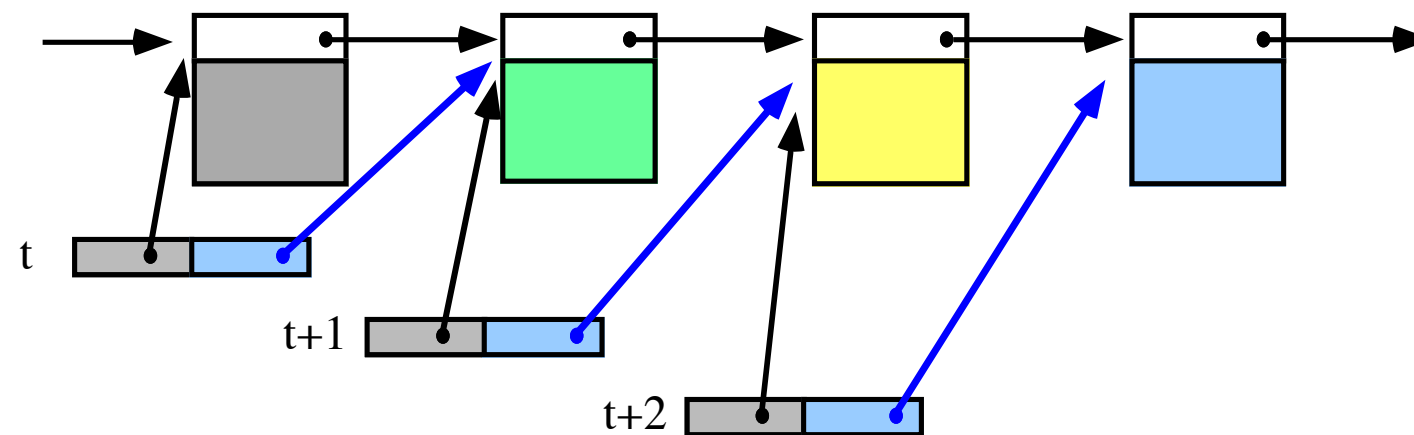


- Zeiger auf Vorgänger wird gebraucht

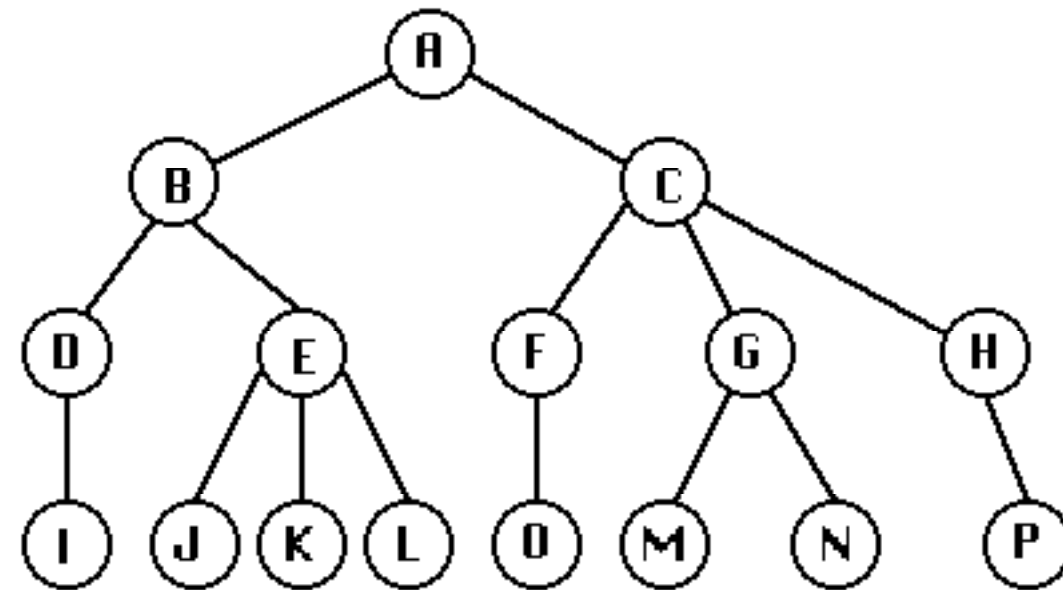
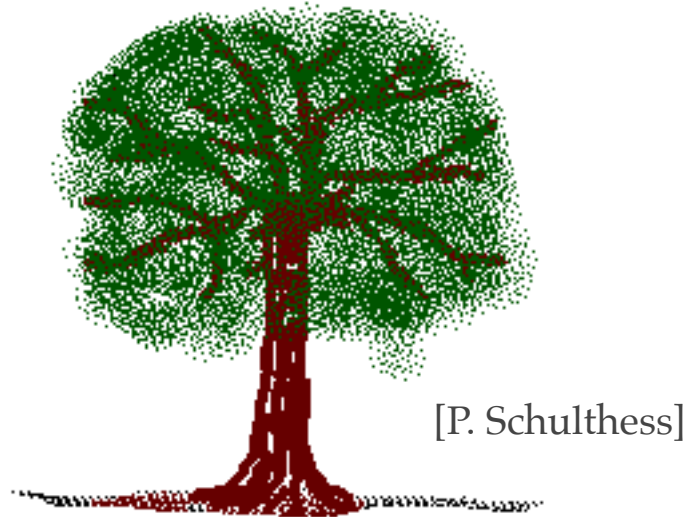
- Einfügen *nach* einem Element ist leicht
- *vor* einem Element schwierig  
=> Trick: Nachher einfügen und altes Element nach vorne kopieren.



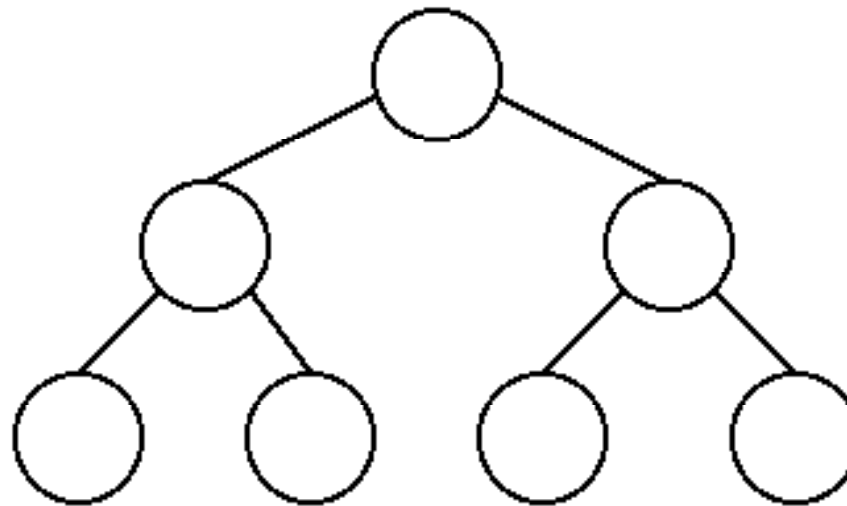
- Andere Lösung  
Schleppzeiger auf Vorgänger immer mitführen



- Listen mit mehr als einem Nachfolger: Bäume



- Binärer Baum





- Durchsuchen von Bäumen

- linker Folge-Teilbaum
- Knoten testen
- rechter Folge-Teilbaum
- => in-order
- Aufhören bei Erfolg

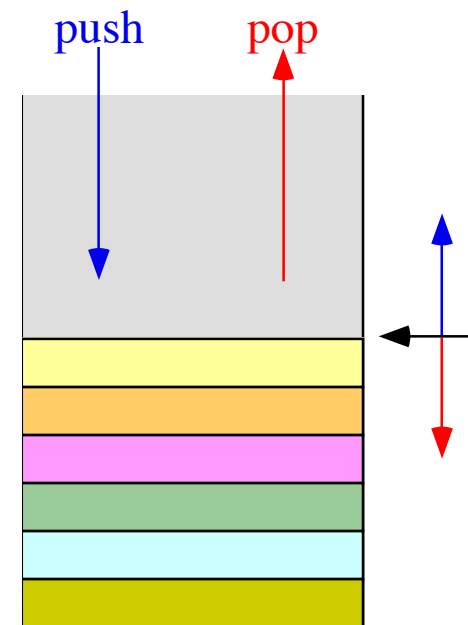
```
knoten *teste(knoten *mitte, elt such)
{
 knoten *help;
 if (mitte == NULL) return NULL;
 else
 { help = teste(mitte->links, elt);
 if (help != NULL) return help;
 if (mitte->inhalt == such) return mitte;
 return teste(mitte->rechts, elt);
 }
}
```

- Andere Suchreihenfolge

- per-order: Knoten, links, rechts
- post-order: links, rechts, Knoten

# Speicherverwaltung

- Platz für dynamisch eintreffende Daten
- Verwaltung
  - freien Platz finden
  - gebrauchten markieren (lock)
  - nicht mehr gebrauchten freigeben
  - Fehlerrouinen
- Bearbeitungssemantik
  - Stack: last in - first out
  - Warteschlange: first in - first out
- Stack (Stapel)
  - Stack\_Push legt Element in den Stack
  - Stack\_Pop holt Element ab / heraus
  - Stack\_Create
  - Implementierung mit Array oder mit Liste



```

#include <stdlib.h>
#include "item.h"

Item *stack;
int level, size;
Item badItem = {.text="stack_underrun"};
enum stack_error {stack_noError, stack_overflow};

void stack_Create(int maxElts)
 {stack=malloc(maxElts*sizeof(Item));
 level=0; size = maxElts;}

int stack_Push(Item theItem)
 {if (level<size)
 {stack[level++]=theItem; return stack_noError;}
 else return stack_overflow;}

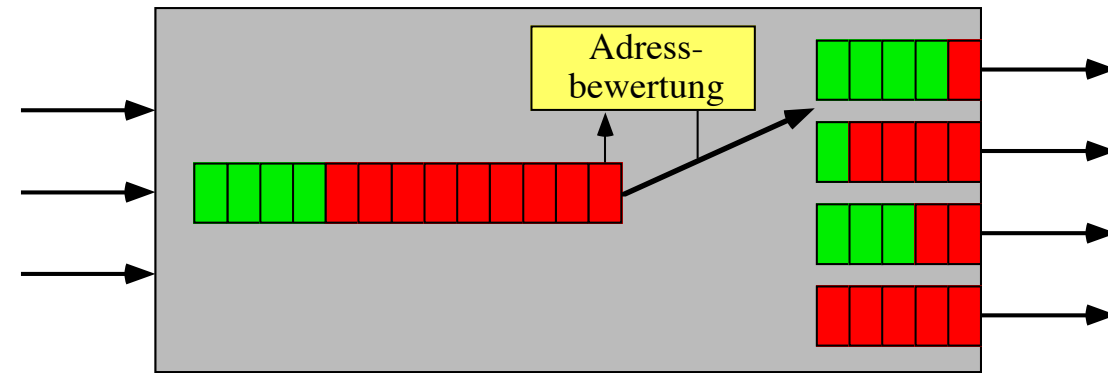
Item stack_Pop()
 {if (level>0) return stack[--level];
 else return badItem;}

int main () {stack_Create(42);};

```

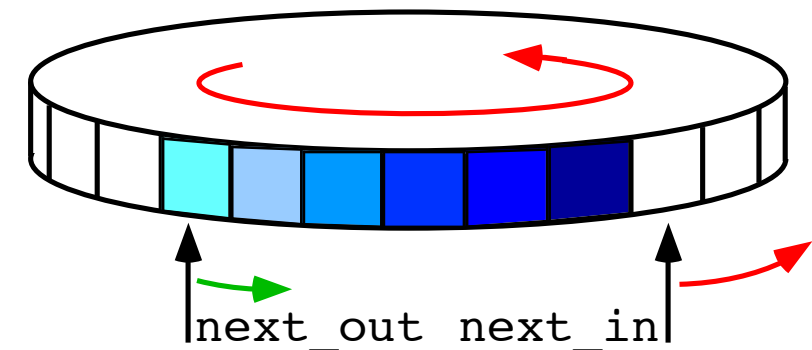
- Queue (Warteschlange)

- FIFO
- z.B. Netzwerkadapter, Router
- in Array oder Liste



- Ringpuffer

- 2 Zeiger: next\_in, next\_out
- Schreiben inkrementiert next\_in
- Lesen inkrementiert next\_out
- inkrementieren modulo buffersize
- am besten 8, 16, 32, 64, ...



```
#include <stdlib.h> /* nach Sedgewick */
#include "Item.h"
```

```
typedef struct QUEUEnode* link;
struct QUEUEnode { Item item; link next; };
link head, tail;
```

```
void QUEUEinit() { head = NULL; }
int QUEUEempty() { return head == NULL; }
```

```

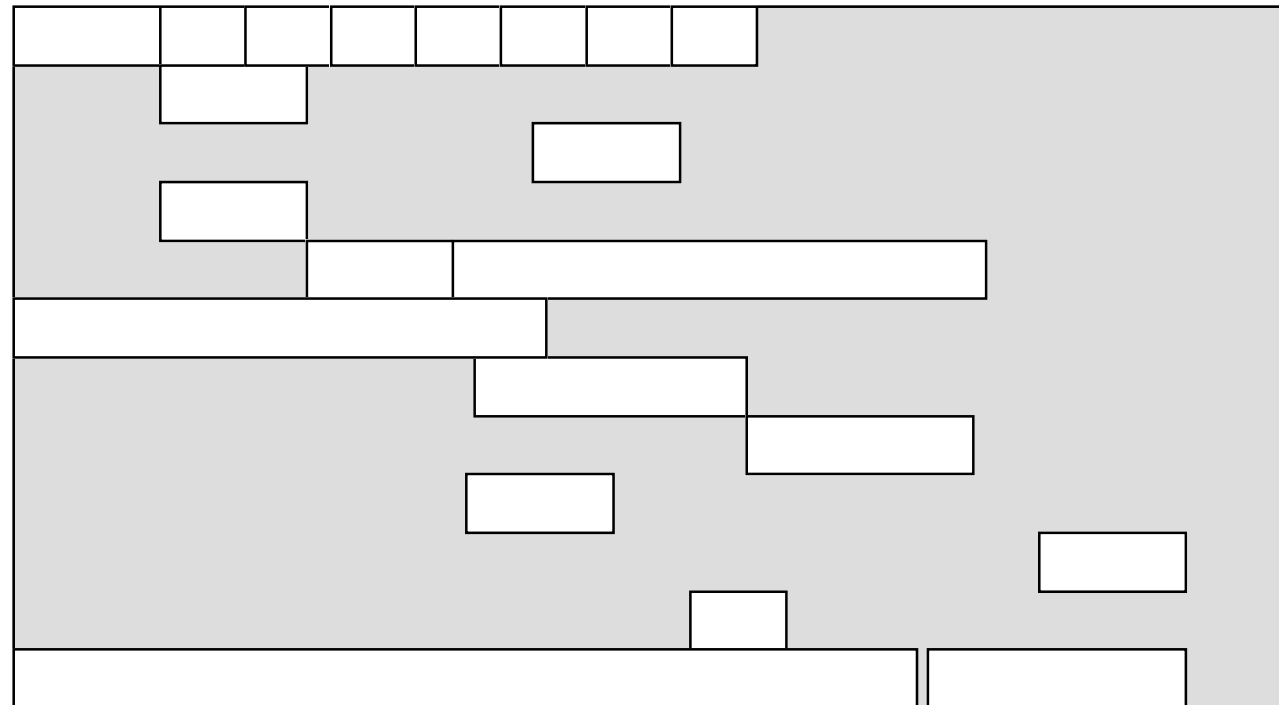
link NEW(Item item, link next)
{ link x = malloc(sizeof *x);
 x->item = item; x->next = next;
 return x;
}

enqueue(Item item)
{ if (head == NULL)
 { head = (tail = NEW(item, head)); return; }
 tail->next = NEW(item, tail->next);
 tail = tail->next;
}

Item dequeue()
{ Item item = head->item;
 link t = head->next;
 free(head); head = t;
 return item;
}

```

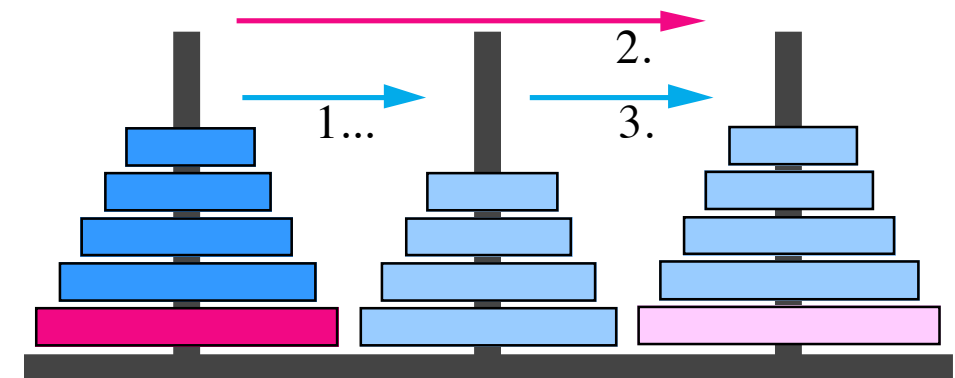
- Heap (Halde)
  - beliebige Speicherbereiche



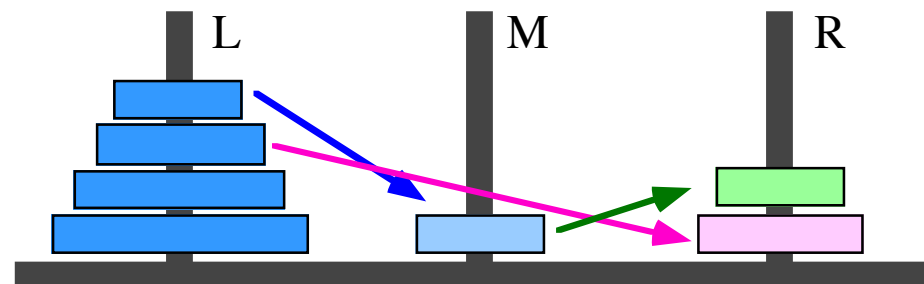
- anlegen mit `void *malloc(size_t size)` aus `stdlib.h`
  - freigeben mit `free(void *pointer)`
- Implementierung
  - großer Bereich des Hauptspeichers
  - Liste mit belegten Blöcken
  - Freispeicherliste
  - Fragmentierung sorgfältig managen

# Rekursion

- Funktion wird in sich wieder aufgerufen
  - direkt oder indirekt
- Problem auf einfacheres zurückführen
  - wiederholen bis zum trivialen Problem
  - in jedem Schritt etwas leichtes machen
- wo haben wir das schon benutzt?
- Beispiel: Türme von Hanoi
- Regeln:
  - Scheibenturm von links nach rechts bringen
  - immer nur eine Scheibe bewegen
  - nur kleinere Scheiben auf größere legen



- Lösungsansatz:
  - Bewegung des Turmes mit Höhe N auf Turm mit Höhe N-1 zurückführen,
  - Turm( N-1 ) auf mittleren Stock bringen
  - verbleibende Scheibe nach rechts
  - Turm( N-1 ) nach rechts
- Rekursive Strategie:
  - triviale Lösung für Turm( 0 )
  - Lösung zu Turm(1) = Lösung zu Turm(0) + eine Scheibe bewegen
  - Lösung zu Turm(N) = Lösung zu Turm(N-1) + Scheibe N bewegen



- Züge für eine Turmhöhe von 4  
 $L \rightarrow M, L \rightarrow R, M \rightarrow R, L \rightarrow M, R \rightarrow L, R \rightarrow M, L \rightarrow M, L \rightarrow R,$   
 $M \rightarrow R, M \rightarrow L, R \rightarrow L, M \rightarrow R, L \rightarrow M, L \rightarrow R, M \rightarrow R,$



```

#include <stdio.h>

typedef char Stock; /* 3 Stöcke: 'L', 'M', 'R' */

void BewegeScheibe(Stock von, Stock nach)
{ Printf("%c", von); Printf("->%c", nach);
 Printf("%c", ',');
}

void BewegeTurm(int restHoehe, Stock von,
 Stock zwisch, Stock nach)
{ if (restHoehe<=0) return;
 BewegeTurm(restHoehe-1, von, nach, zwisch);
 BewegeScheibe(von, nach);
 BewegeTurm(restHoehe-1, zwisch, von, nach);
 printf("\n");
}

main()
{ BewegeTurm(4, 'L', 'M', 'R'); }

```

# Objekte, Patterns und Frameworks

## Grundzüge des objektorientierten Programmierens

- Kay, Goldberg, Ingalls: Smalltalk
- Datenstrukturen
  - Inhalt als Attribute
  - Funktionen zur Inhaltsmanipulation
  - Semantik steckt in den Funktionen

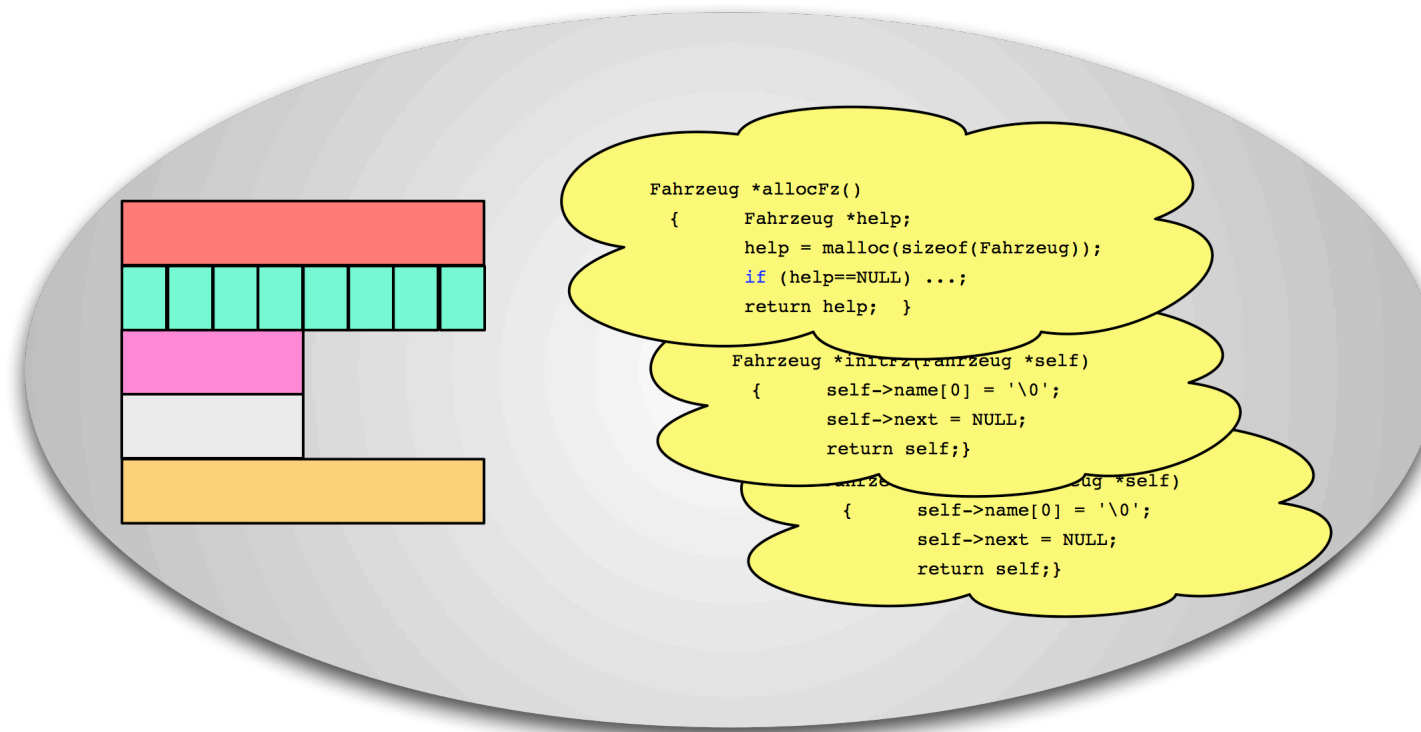
```
typedef struct fzstruct
{ char name[32];
 enum {Lok,Perswg,Gueterwg} art;
 int Baujahr;
 struct Fahrzeug *next;
} Fahrzeug;
```

```
Fahrzeug *lok;
...
lok = initFz(allocFz());
lok->next = initFz(allocFz());
```

```
Fahrzeug *allocFz()
{ Fahrzeug *help;
 help = malloc(sizeof(Fahrzeug));
 if (help==NULL) ...;
 return help;
}

Fahrzeug *initFz(Fahrzeug *self)
{ self->name[0] = '\0';
 self->next = NULL;
 return self;
}
```

- Objekte: Struktur+Funktionen
  - als eigenständige Einheit des Programmierens
  - Klassiker: Initialisierung einer Struktur, Vergleich, ...
  - datenstruktur-orientierte Modularisierung



- Beispiel Swift
  - Apple 2014
  - NeXTStep, OpenStep, GNUStep, Cocoa, iPhone, iOS
  - typesafe: optionals

- Klasse = Strukturdeklaration + Funktionen

- Funktionen zum:

- Initialisieren, Löschen, ...

- evtl. Felder zugreifen

- Listenoperationen

- heißen Methoden, Selektoren, Messages

=> Konzept

```
enum Fz {case none, Lok, PersWg, Gueterwg}

class Fahrzeug {
 var name: String
 var art : Fz
 var baujahr = 2014, gewicht = 0
 var next: Fahrzeug? = nil

 init(aName : String, typ : Fz) {
 art = typ; name = aName
 }
 func add(fz : Fahrzeug) { next = fz }
 func setweight (wght : Int) {gewicht = wght}
 func getweight () -> Int {return gewicht}
 func getNext () -> Fahrzeug {return next!}
 func cmpwght() -> Int {
 if next==nil {return gewicht}
 else {return gewicht + next!.cmpwght()}}
}
```

- Instanzen / Instantiierung

- dynamisches Anlegen `<class>()`

- Nachrichten senden

- entspricht Funktionsaufruf

`<obj>.<message>()`

- mit Parametern:

`<obj>.<message>(<parm>:<wert>)`

`<obj>.<message>(<parm1>: <wert1>, <parm2>: <wert2>, ...)`

```
var zug = Fahrzeug(aName:"V200 002", typ: Fz.Lok)
zug.baujahr = 1953; zug.gewicht=80;

zug.add(Fahrzeug(aName: "BDm",typ:Fz.Perswg))
```

- Accessor-Methoden

- in den Methoden der Klasse mit Feldname

- klassisch: `<instance>.<property>`

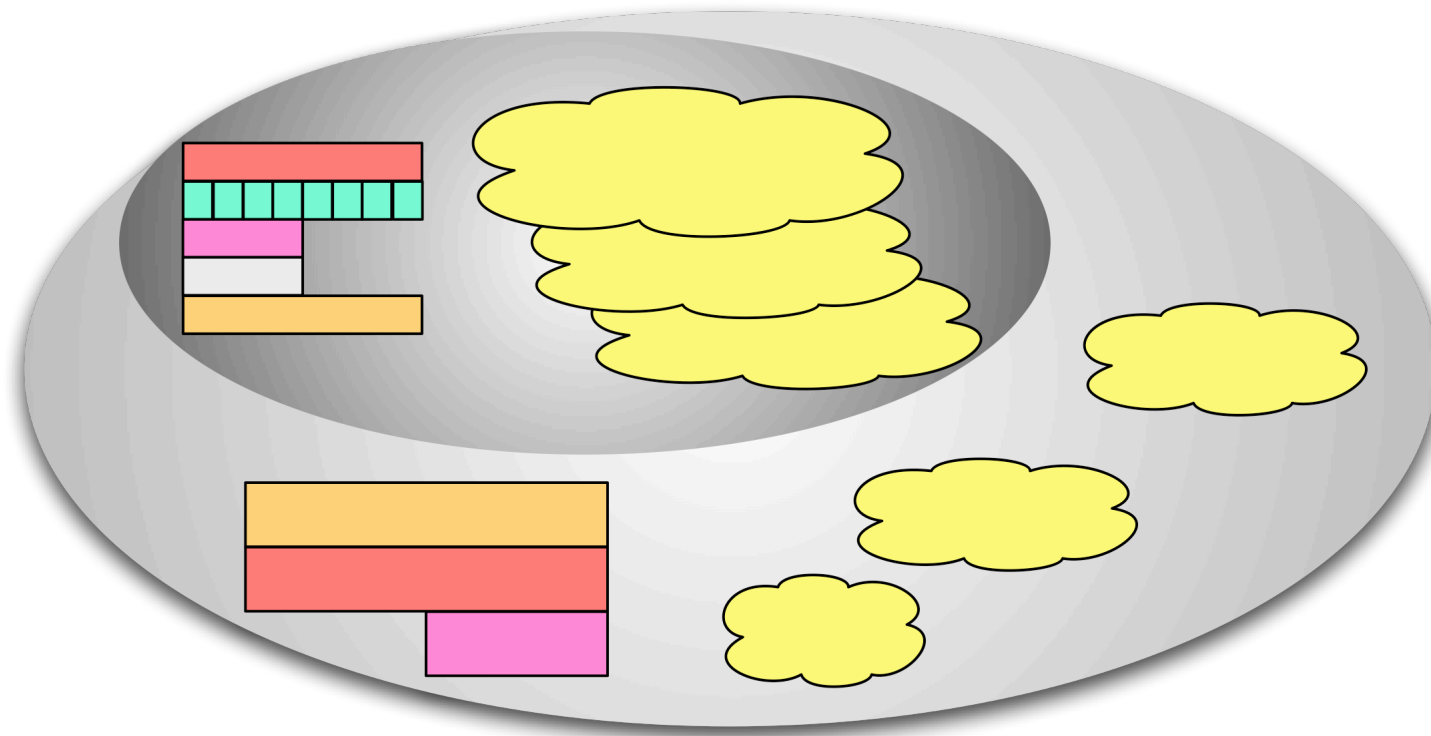
- Accessor `<object>.<feldzugriff>()`

- Namenskonvention für Methode `get<feld>` bzw. `set<feld>`

```
init(aName : String, typ : Fz) {
 art = typ
 name = aName }
```

```
func setweight(wght: Int)
func getweight() -> Int
```

- Vererbung
  - Klassen erweitern
  - Basisklasse + neue Felder und Methoden



- Felder und Methoden der Basisklasse weiter verwendbar
- Begriff: Superklasse
- self und super

- Methoden

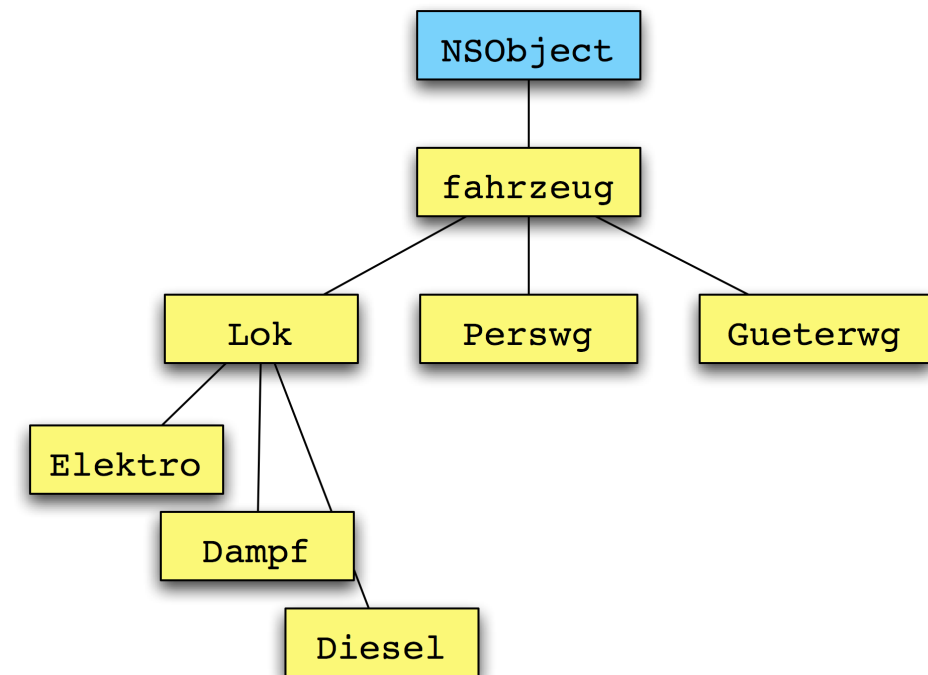
- erben
- neue hinzufügen
- manche überschreiben
- Scope: Blatt zur Wurzel

```
enum Sys {case none, Elektro, Diesel, Dampf}

class Lok : Fahrzeug {
 var leistung = 0, last = 0
 var prinzip = Sys.none

 init(aName : String, pwr : Int, load : Int) {
 prinzip = Sys.none
 leistung = pwr; last = load
 super.init(aName: aName, typ: Fz.Lok)
 }
 func add (zug : Fahrzeug) -> Bool{
 if last < zug.cmpwght() {return false}
 else {super.add(zug); return true}
 }
}
```

- Klassenbaum
  - Alle Klassen erben von einer Wurzelklasse, hier: NSObject
  - möglichst viele Methoden erben
- Klassenmethoden
  - zum allgemeinen Klassenmanagement
  - z.B. Konstruktor
  - `class func <name>() ...`
  - self bezieht sich hier auf Klasse
  - auch Klassenvariablen
- Mehrfachvererbung?
  - von mehreren Klassen erben
  - nicht wirklich notwendig
  - nicht in allen OO-Programmiersprachen
  - in manchen Hilfskonstrukte





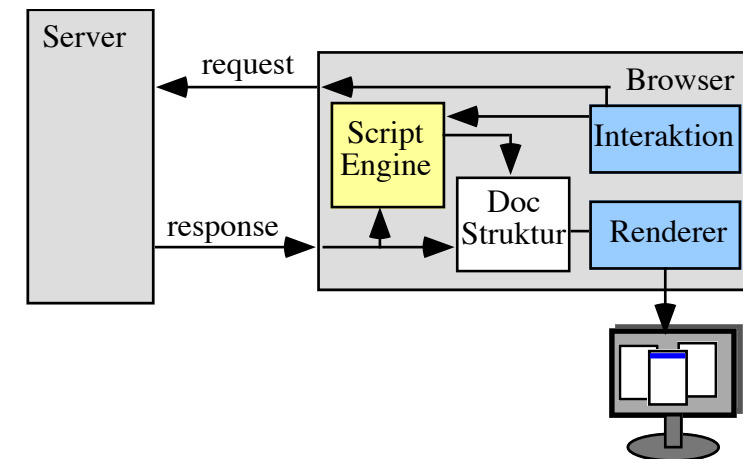
## Frameworks am Beispiel

- Funktionen und Strukturen in Libraries
  - häufig verwendet
  - Listen, Dateizugriff, Ein/ Ausgabe, Fenster, ...
  - auch API genannt
  - kann zur Abstraktion verwendet werden
  - Portabilität
- Framework
  - Menge von Klassen
  - Wiederverwendbarkeit
  - Erweiterbarkeit: überschreiben, spezialisieren, ...
  - gibt Programmstruktur vor
  - "Klasse Programm"
- Kontrollumkehr
  - Programm wird zur Funktionssammlung
  - Funktionen im Programm werden von außen gerufen
  - Programmierer *füllt* vorgegebene Funktionen

- Frameworks in iOS und MacOS
  - WatchKit, HealthKit, Media Player, ...
  - Core Image, GLKit, ...
- Beispiel Foundation
  - aus NeXTStep: Namen NS\*
  - NSDate, NSError
  - NSPreferences, NSPort, ...
  - NSNotification, NSNotificationCenter
  - NSString: Unicode
- Operating System Services
  - NSFileManager, NSPathUtilities
  - NSThread, NSProcessInfo
- NSURL

```
var myurl=NSURL(string:"http://ara.informatik.tu-freiberg.de/index.html")
let hostname = myurl?.host
```

- Patterns
- Gang of four: "Design Patterns: Elements of Reusable ..." [1994]
- Lösungen für typische Problemstrukturen
  - generalisiert, wiederverwendbar
  - 'best practice'
  - "über" Datenstrukturen und Algorithmen
  - auch zur Prozesskommunikation
- Ein berühmtes architectural pattern: MVC
- Model
  - Verhalten und Daten
  - application domain
- View
  - Darstellung des Modells zur Interaktion
  - Display, Rendern
- Controller: reagiert auf Events
  - ändert Modell
  - auch nachrichtenbasiert



- Creational Patterns
  - Objekte erzeugen, Singleton
  - Factory, Builder, Object Pool, Prototype
- Structural Patterns
  - Beziehungen zwischen Elementen
  - Bridge, Adapter
  - Composite und Aggregate
  - Proxy, Facade, Extensibility (Framework pattern)
- Behavioural Patterns
  - Chain of responsibility: Events erhalten und weitergeben
  - Observer (publish / subscribe): für Events registrieren
  - Iterator greift auf Felder nacheinander zu
- Concurrency Patterns
  - konzeptuell parallele Ausführung (threads)
  - Active Object, Monitor, Leader Follower, ...
  - Reactor
  - Scheduler
  - Thread Pool

# Betriebssysteme

- Abstrahieren und Koordinieren
- Sammlung häufig gebrachter Softwarekomponenten
  - Hardware-Abstraktion
  - essentielle Software
- Zuteilung der Ressourcen
  - CPU, Speicher, Bildschirm, ...
- Oft mit Kommandointerpreter verwechselt
  - Shell, Command.COM, ...
  - Explorer, Finder, ...
- Dateisystem
  - ebenfalls nur Teil des Betriebssystems
  - Verwaltung von Plattenplatz (Sektoren)
  - Zuordnung und Wiederverwendung
  - Abstraktion Sektor - Bytestrom

```
printf("%c", char);
```

```
MOVE char,$A7823A
```

```
MOV AX,[BP]
OUT $3F8,AX
```

# Verteilung der Ressourcen

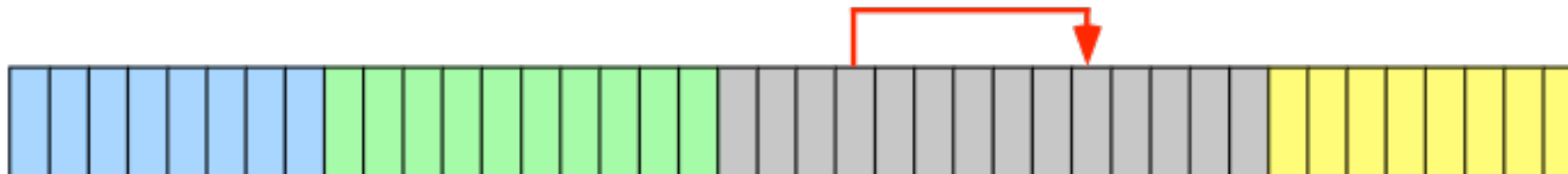
- Prozessor
  - Verteilung der Arbeit auf Prozessoren
  - Unterbrechungen
  - wartende Prozesse
- Speicher (RAM, Festplatte)
- Ein/ Ausgabe
  - Bildschirm, Drucker
  - Audio-Ausgabe
- Verteilung des Mangels
  - Bildschirmfläche endlich => Fensterkonzept
  - Tastatur, Maus
  - Audio-Ausgang => Mixer
  - Prozessorzeit => Zeitscheiben, Zeitüberwachung

## Prozesse und Scheduling

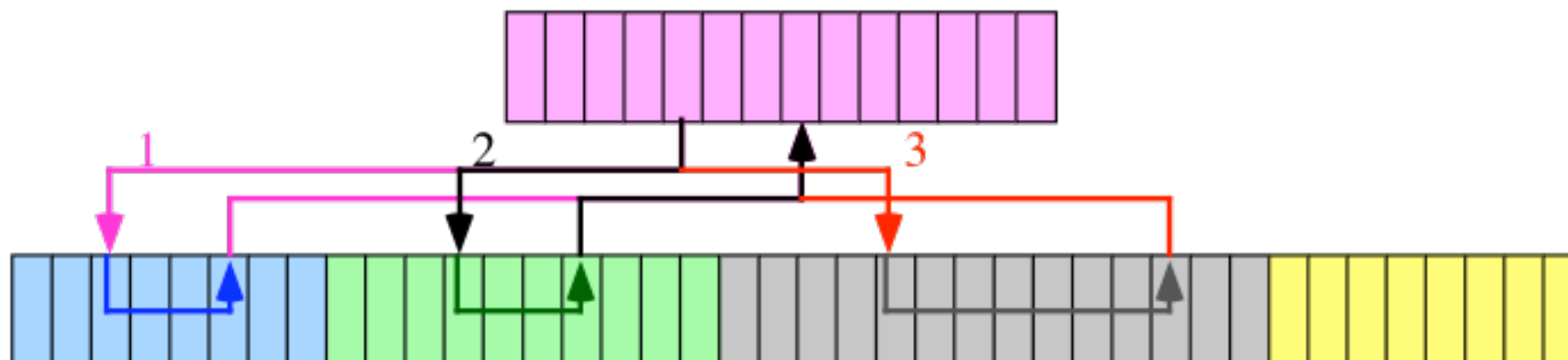
- Instruktionen werden sequentiell ausgeführt
  - in der Regel nicht gleichzeitig
  - ein Befehl nach dem anderen
  - Sprünge verändern nur die Reihenfolge



- Programm = ausführbares Objekt = Prozeß
  - mehrere Prozesse liegen im Speicher



- Benutzerprozesse und ein besonderer Steuerprozeß



- Besonderer Steuerprozeß
  - gibt Ausführungsrecht befristet ab an Benutzerprozesse
  - verteilt Ausführungsrecht 'gerecht'
  - hat besondere Rechte
  - kennt alle anderen Prozesse
- Rückwechsel vom Benutzerprozeß zum Scheduler
  - Sprung in den Scheduler
  - freiwillig
  - erzwungen nach Fristablauf (=> Unterbrechung)
- Prozesseigenschaften
  - Wichtigkeit (Priorität)
  - benutzte Ressourcen (Festplatte, Drucker, ...)
  - verbrauchte Zeit
  - zugeordneter Speicher
- Auslagern (Swapping)
  - falls Hauptspeicher zu klein
  - exaktes Prozeßabbild auf Festplatte schreiben



- Prozesse

- selbständige Codeeinheiten
- Object-Code
- Speicher
- Attribute

- Multi-Tasking

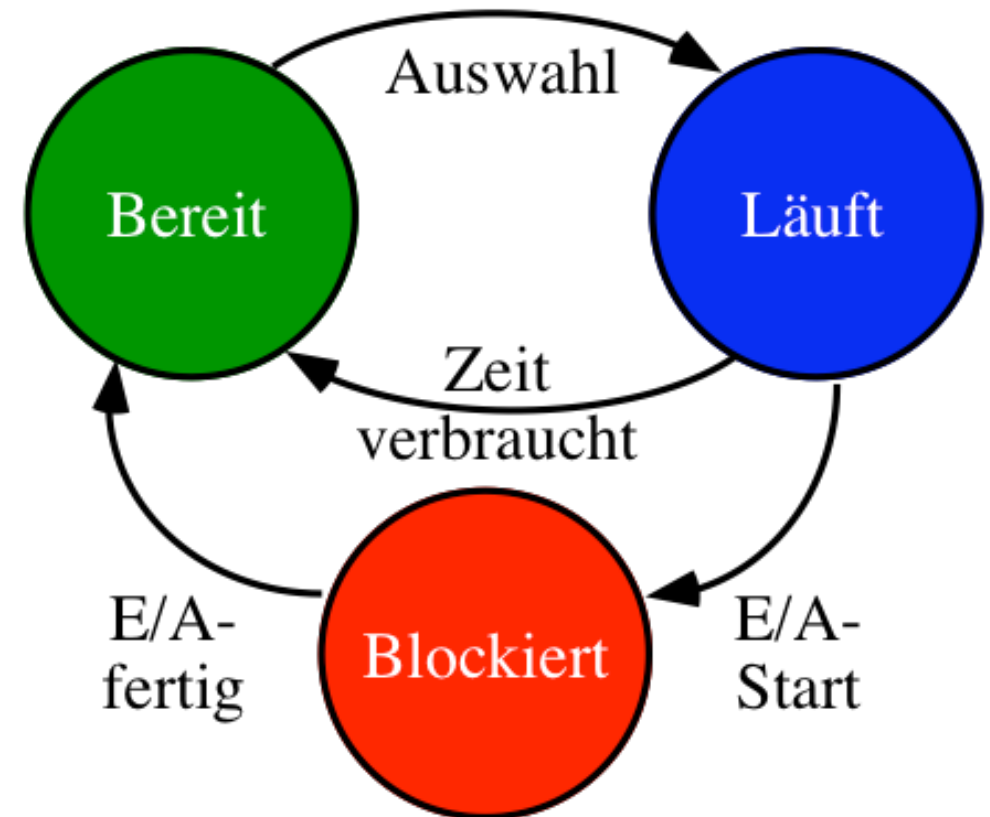
- mehrere Prozesse laufen 'verschachtelt'
- "Zeitmultiplex"
- für den Benutzer gleichzeitig
- verschiedene Strategien des Wechsels



- Prozesswechsel (Umschalten zwischen Programmen)

- Anhalten eines Prozesses
- Speichern der Status-Information (PC, Register etc.)
- Laden der Status-Information des neuen Prozesses
- Wiederaufnahme der Ausführung bei der nächsten Instruktion
- z.B. nach Zeit  $t$ , wenn gewartet werden muß, ...

- Einplanung  $\neq$  Scheduling



- Non-Preemptive Scheduling

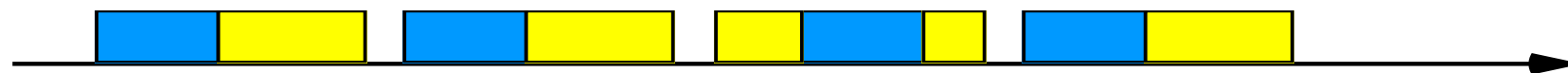
- Prozesse nicht unterbrechbar



- Weniger Prozesswechsel
- Kritische Prozesse können nicht unterbrochen werden

- Preemptive Scheduling

- Prozesse durch andere Prozesse mit höherer Priorität unterbrechbar

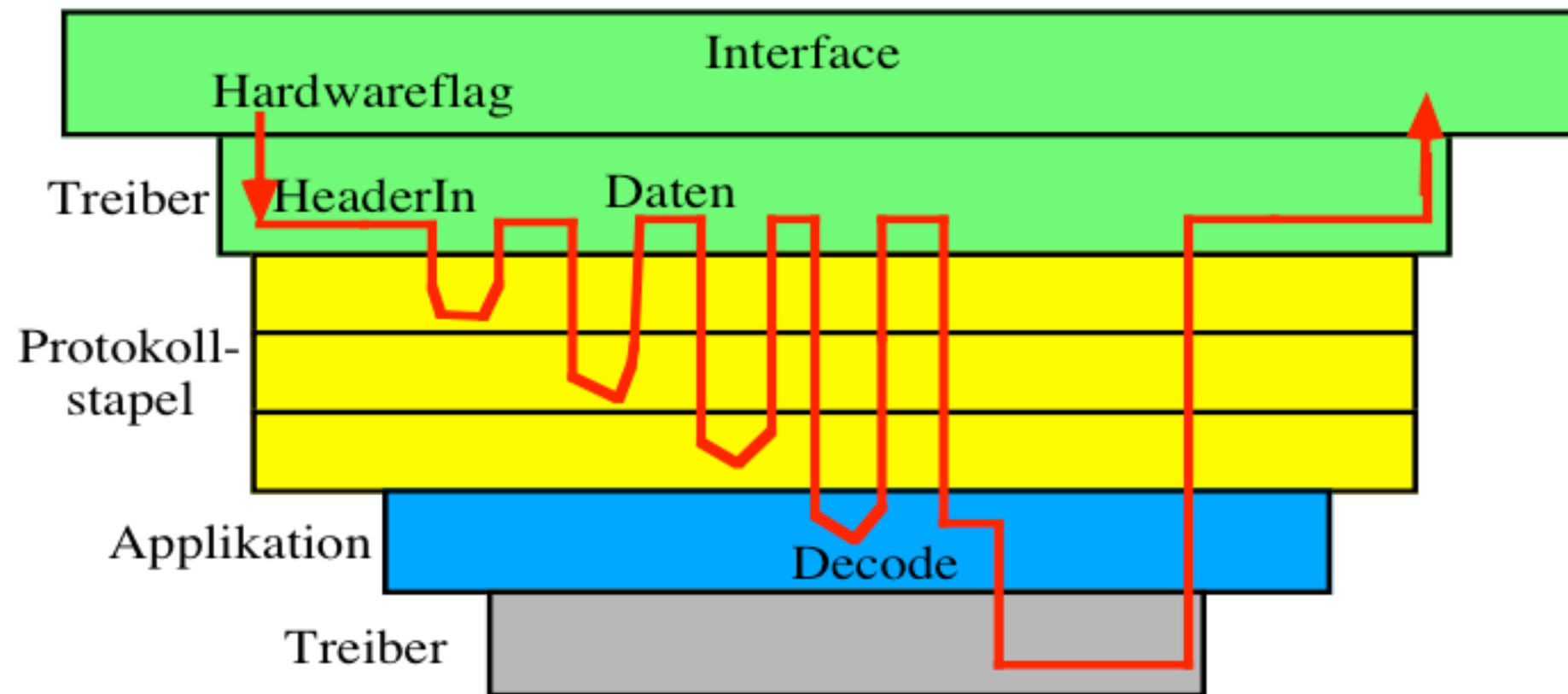


- Oft in Betriebssystemen vorhanden für CPU
- Prozesswechsel häufig und teuer

- Prioritätsfestlegung

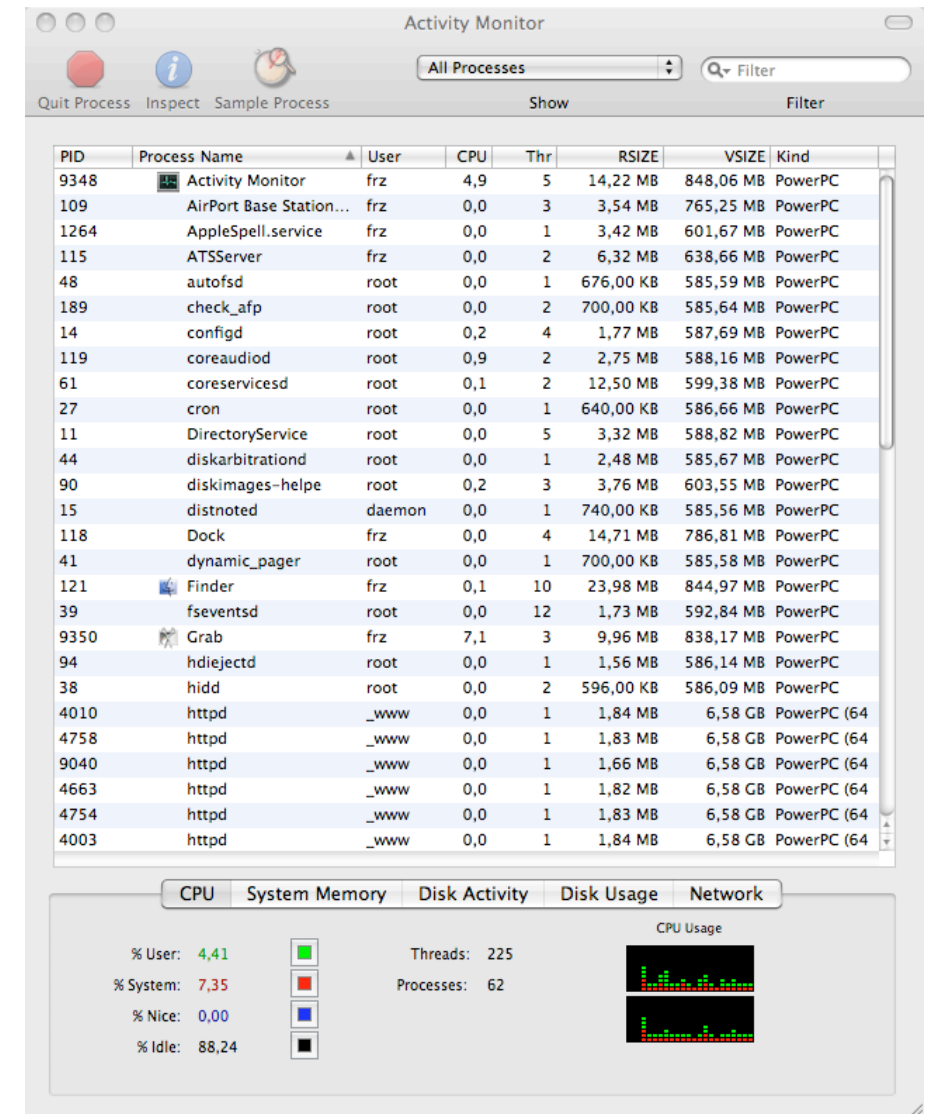
- Wichtigkeit
- nahe 'Abgabe'-Termine
- lange gelaufen => Priorität sinkt

- Unterbrechungsmanagement (Interrupts)
  - Interrupt-Service-Routine im Netzwerktreiber
  - Completion-Routine auf höheren Schichten (Call Back Routine)



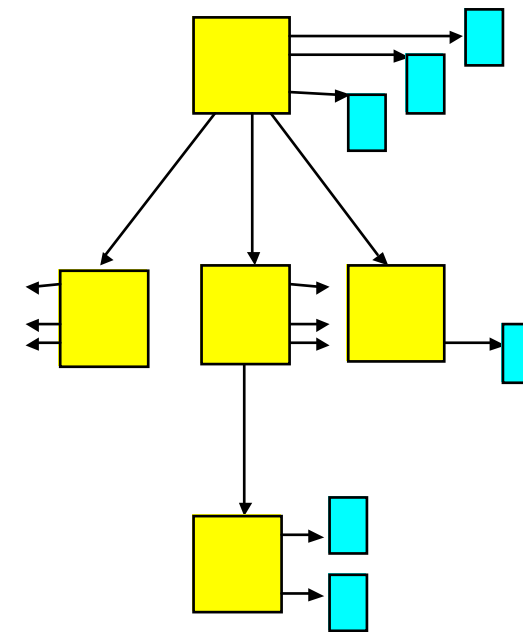
- Interrupt-Latenz durch nicht unterbrechbare Prozesse
- Interruptsperre, z.B. UNIX-Kern

- Steuerung der Prozesse
  - besonderes Steuerprogramm ('Shell')
  - kennt Scheduler
  - Starten eines Programmes => neuer Prozeß
  - Zwangs-Beenden entfernt Prozeß
  - besonders bekannt bei UNIX bzw. DOS
- Grafische Benutzeroberflächen für Steuerprogramm
  - integriert in File-System-Browser
  - Menubefehl 'Open'
  - Doppelklick als Abkürzung
  - Cntl-Alt-Del (Strg-Alt-Entf) zeigt Prozeßliste
  - Windows: Alt-Tab wechselt zwischen Programmen
- Eingebaute Kommandos
  - Prozessliste anzeigen
  - Geräteverwaltung
  - auch Dateiverwaltung auf der Festplatte



## Verwaltung des Plattenplatzes

- Zentrale Abstraktion: Datei
  - Menge von Bytes auf der Platte
  - wird vom Betriebssystem zusammengehalten
  - für Programme und Daten
  - Daten: Briefe, Tabellen, Bilder, Datenbank, ...
- Verzeichnisse strukturieren Dateimenge
  - Verzeichnis enthält Dateien
  - Verzeichnis kann andere Verzeichnisse enthalten
  - Wurzelverzeichnis
- Pfad
  - Wurzel / Verzeichnis / Verzeichnis / Verzeichnis / Datei
  - c:\benutzer\froitzheim\daten\vorlesung.doc

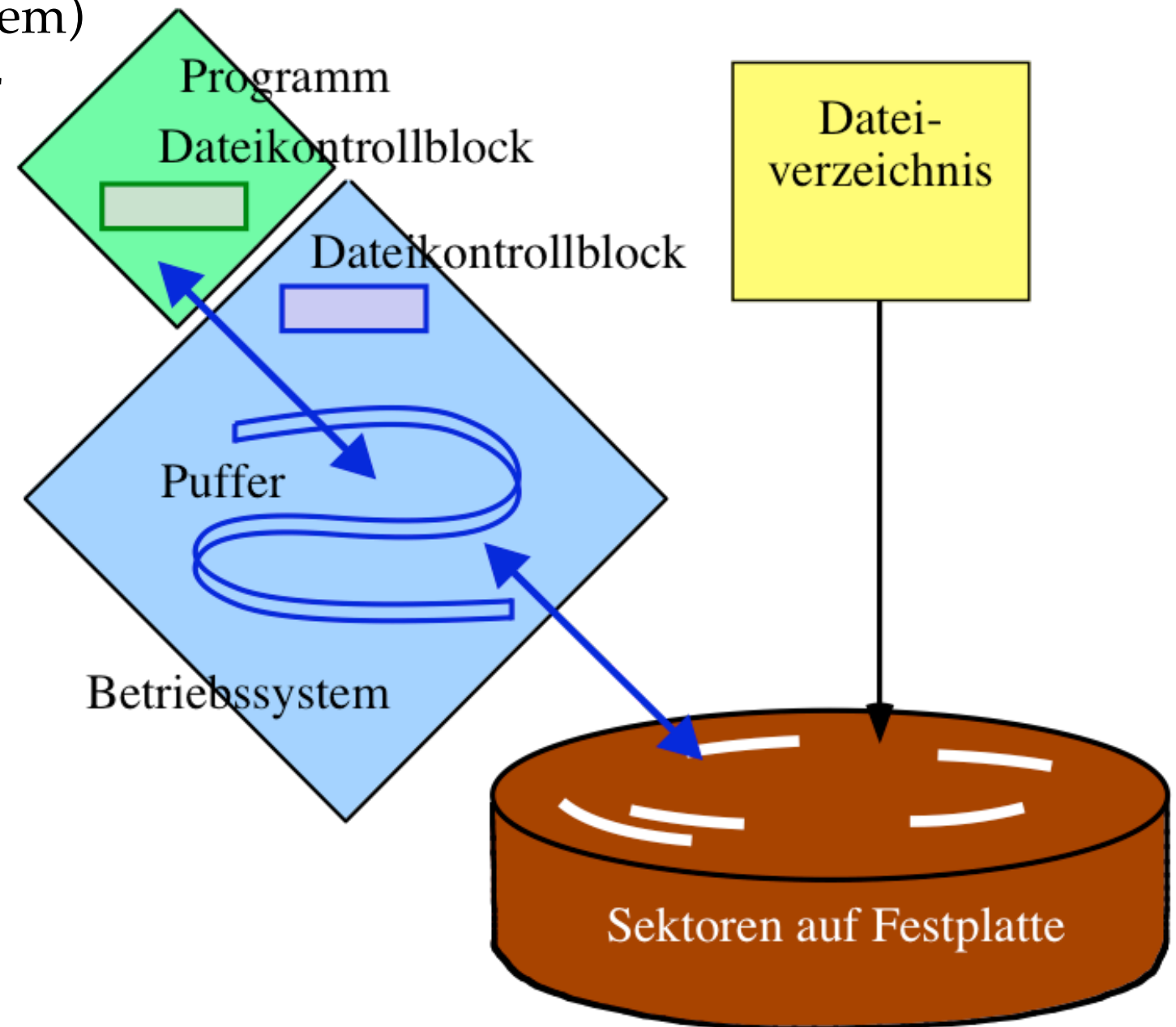


- Historisch gesehen ein Magnetband
  - mit einem Schreib- / Lesekopf
  - Datenblöcke sequentiell aneinanderreihen
  - wahlfreie Zugriffe teuer bzw. langsam
  - Einfügeoperationen sehr teuer
  - beinahe beliebiges Größenwachstum
- Aus der Sicht der Programmiersprache
  - Von Dateimodulen exportierter Typ "File"
  - sequentiell Lesen und Schreiben
  - Öffnen und Schliessen, Zugriffsrechte
- Aus der Sicht des Betriebssystems
  - benanntes Objekt ("WS1996PI.TXT")
  - Dateikontrollblock mit Puffer im RAM
  - Assoziation einer Dateivariablen mit externer Datenregion
  - Abbildung:  
blockweise adressierte Objekte auf Platte  
=> byteweise adressierte Objekte im Hauptspeicher



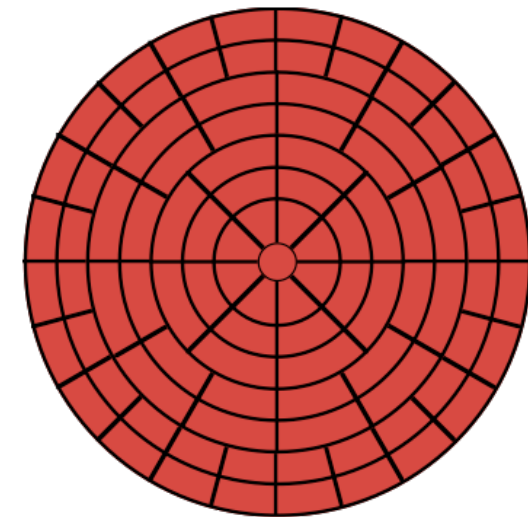
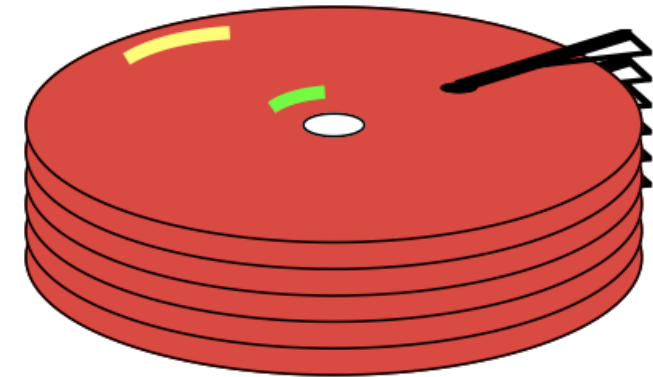
- Implementierung

- Programm mit Zugriffsroutinen
- Treiberrouinen im Betriebssystem
- Dateivariable im Programm (file)
- Dateikontrollblock (User & System)
- Pufferbereich im Hauptspeicher
- Dateiverzeichnis für Festplatte
- Sektorzuordnung auf Festplatte



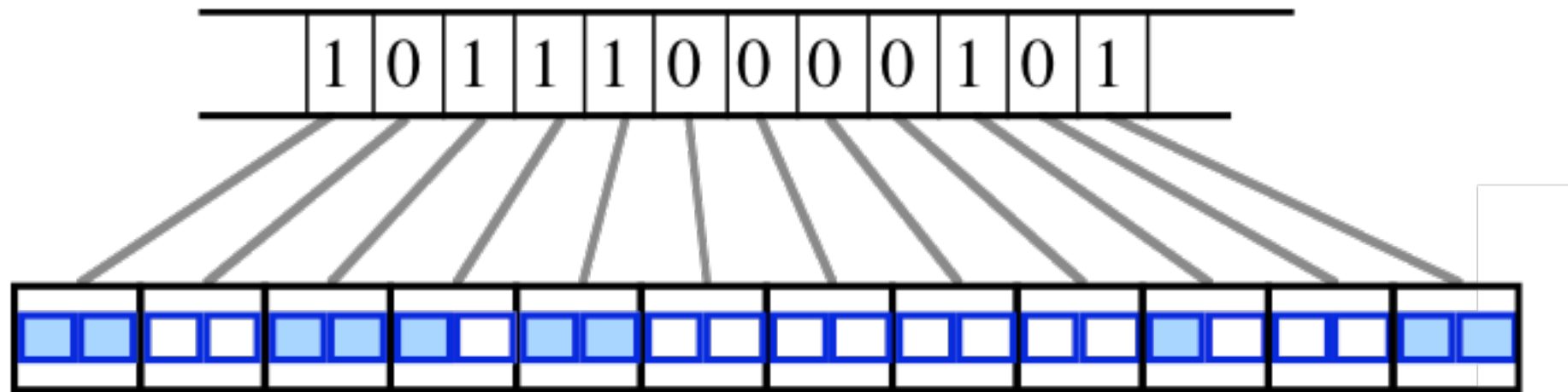


- Plattenformat
  - $n$  Platten auf einer Spindel (z.B. 5)
  - $2n$  Oberflächen  $\rightarrow 2n$  Köpfe
  - viele Spuren pro Platte (z.B. 40.000)
  - Zylinder: Menge von Spuren im vertikalen Schnitt
  - viele Sektoren pro Spur (z.B. 1000 - 6000)
- Mehrere Festplatten-Partitionen
  - $m$  Gbytes pro Platte
  - $d$  Dateinamen pro Platte
  - $s$  Dateistücke pro Platte
  - bessere Nutzung für kleine Dateien ...
- SSD (Solid State Drive) auch blockorientiert
- Verschiedene Datei- & Betriebssysteme auf verschiedenen Partitionen
  - MacOS, MS-DOS
  - Unix, Linux
  - Windows (NTFS)
- Umschalten zwischen Betriebssystemen
  - Parameter RAM
  - Bootmanager





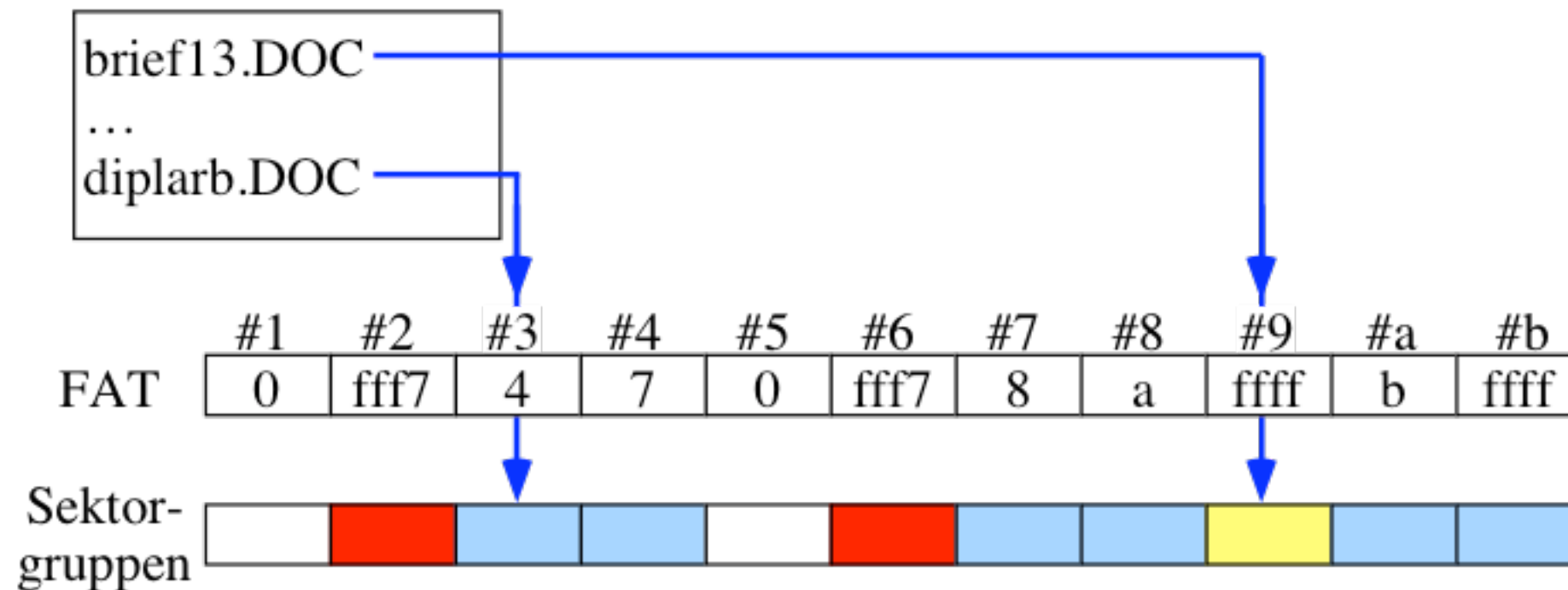
- Plattenspeichervergabe: Wo ist noch Plattenplatz frei?
  - Absuchen der FAT
  - kompakter mit Allokierungs-Bittabelle
  - Bitplätze entsprechen Platz auf der Platte



- Enthält ein Bit für jede Belegungseinheit (Allocation Block)
  - teilweise im Hauptspeicher
  - gesetzt, falls der Block belegt ist
  - z.B. 65536 Bit in der Tabelle
  - ab 64 Mbyte mehr als 2 logische Blöcke pro Bit
- Große Platten
  - viele Blöcke pro Eintrag (Cluster)
  - größere Bitmap
- Keine Aussage über Dateizugehörigkeit

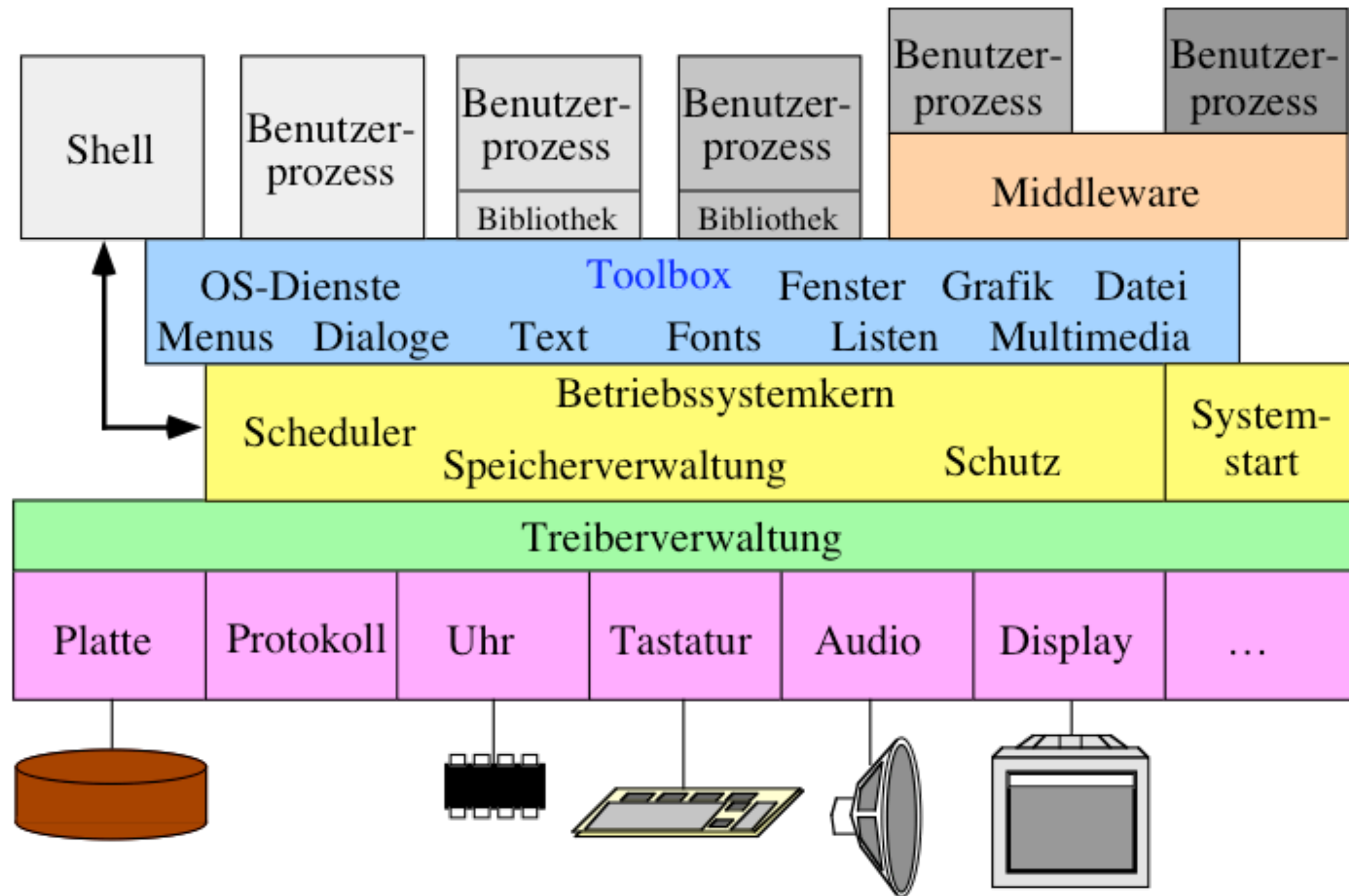
- File Allocation Table (FAT)

- Dateizuordnungstabelle auf den ersten Spuren einer Platte
- Belegungseinheiten (Blöcke) der Datei über FAT verkettet
- der letzte Block ist mit \$FFFF markiert
- schadhafte Blöcke sind mit \$FFF7 markiert



- Verzeichnis zeigt auf den ersten Block einer Datei
- Andere Dateisysteme
  - Windows NTFS komplexer
  - UNIX-NFS netzwerkfähig
- Virtual File System als Abstraktion

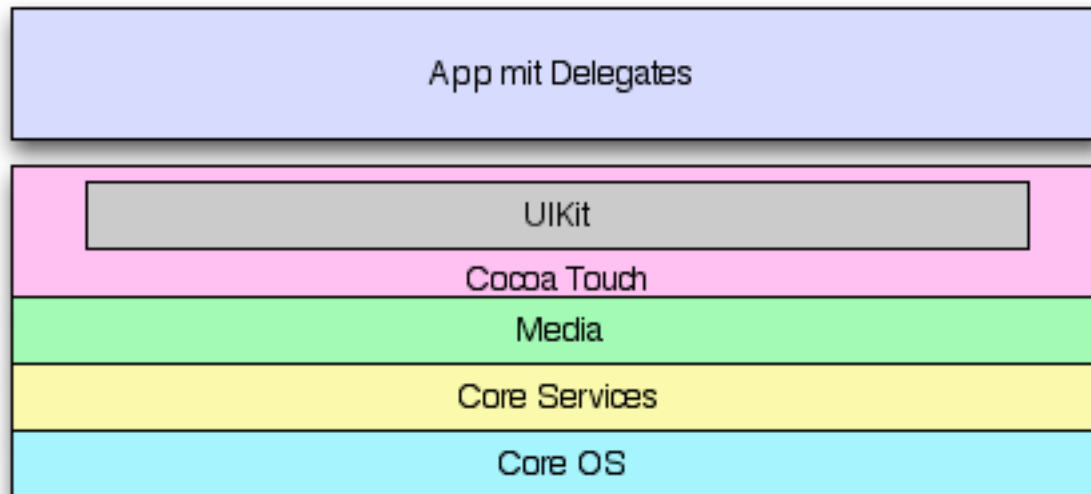
# Struktureller Aufbau am Beispiel iOS



- Abstrakter Zugriff auf Hardware
  - Präsentation: Bildschirm, Drucker, Audio
  - permanenter Speicher: Festplatte, Diskette, CD/DVD
  - Systemressourcen: Uhr, FPU, Sensoren, ...
  - Kommunikation: V.24, Centronics, SCSI, Ethernet, USB, SATA
  - Software: Dateien, Fonts, Menus, Fenster, ...
- Verbergen von:
  - Speicheradressen
  - Registern
  - Kommandos
- Benutzung fertiger Komponenten
- Austausch der Implementierung
  - Beschleunigung
  - neuer Standard
  - Portabilität (WindowsNT auf IA, PowerPC, Alpha, ...)
  - anderes Netzwerk (Bsp: Ethernet statt Token Ring)

- iOS

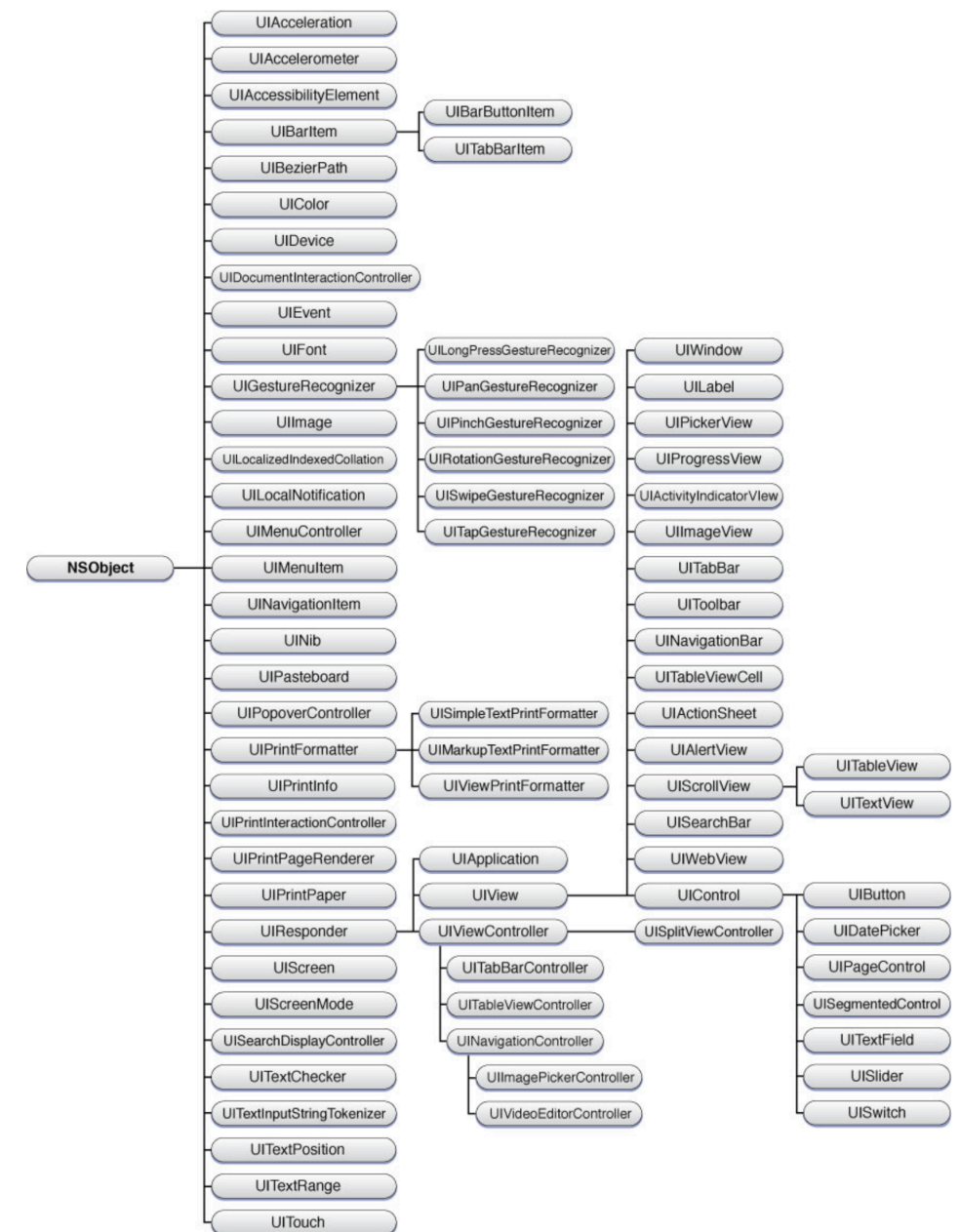
- für iPhone, iPod touch, iPad



- LCD, touch-Sensor-Overlay, nur Flash-Speicher
- UIKit

- Komponenten des UIKit

- UIApplication
- UIWindow
- UILabel
- UITextView, UITableView
- UIImageView
- UISearchBar, UIWebView
- UISlider, UISwitch



## **Beispiel: Applikation Framework UIKit**

- App wird in iOS-Umgebung eingepasst
  - Events (Ereignisse) von Cocoa / UIKit angenommen
  - als Messages an App-Methoden geschickt
- Delegation
  - App soll auf Messages reagieren
  - klassisch: Callbacks bzw. Listeners, z.B. Eventhandler
  - iOS-Pattern 'Delegate'
  - Methoden erweitern, überschreiben oder hinzufügen
- Delegate Protocol
  - spezifiziert Vorgaben
  - Methoden und Parameter
  - manche müssen vorhanden sein, andere optional
  - UIApplicationDelegate

- Struktur einer App

- AppDelegate mit mehreren Protokollen

- `class AppDelegate : UIResponder, UIApplicationDelegate { ...`

- Delegate

- Interface zum Scheduler

- `func application:`

- `func applicationWillResignActive:`

- `func applicationDidEnterBackground:`

- `func applicationWillEnterForeground:`

- `func applicationDidBecomeActive:`

- `func applicationWillTerminate:`

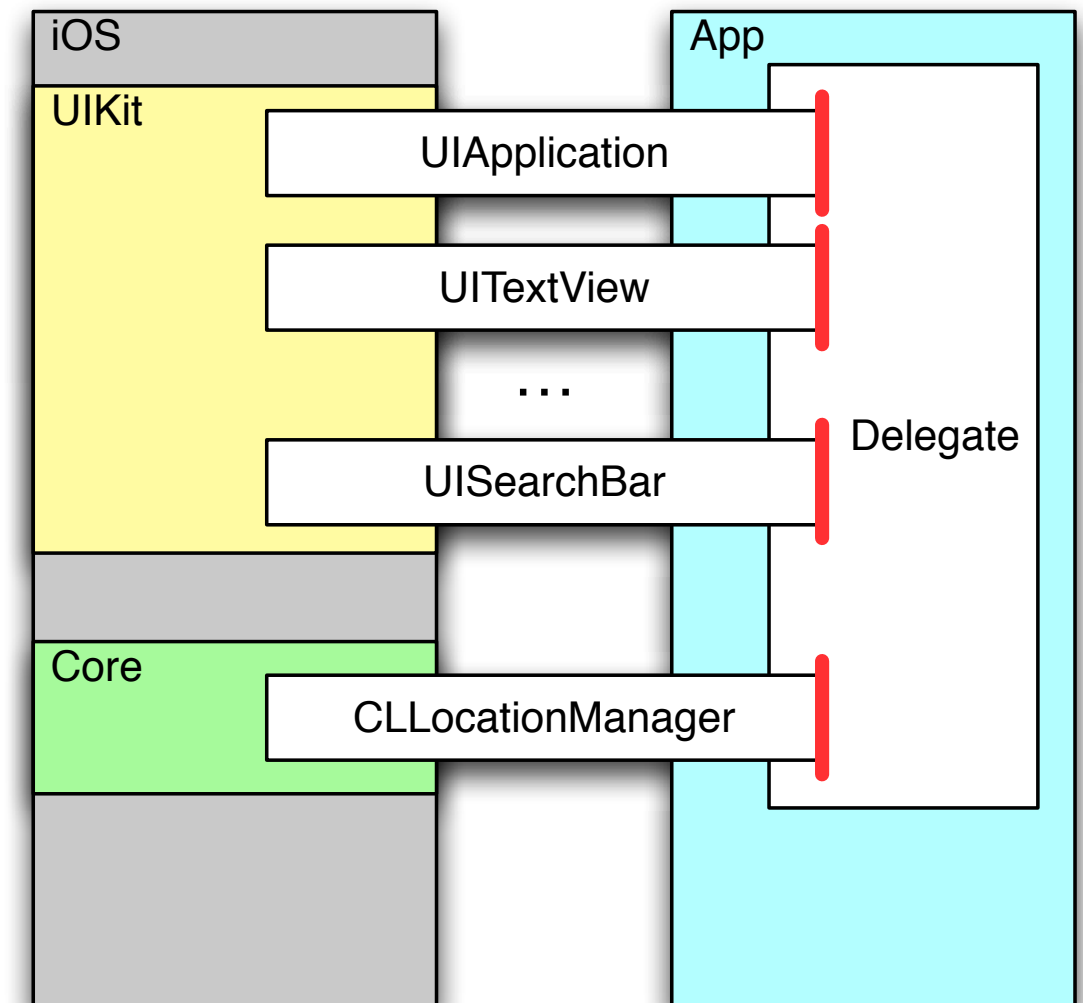
- `func applicationDidReceiveMemoryWarning:`

- UI-Responder

- `@IBAction func buttonPressed(AnyObject);`

- weitere (eigene) Methoden

- importiert weitere (eigene) Klassen



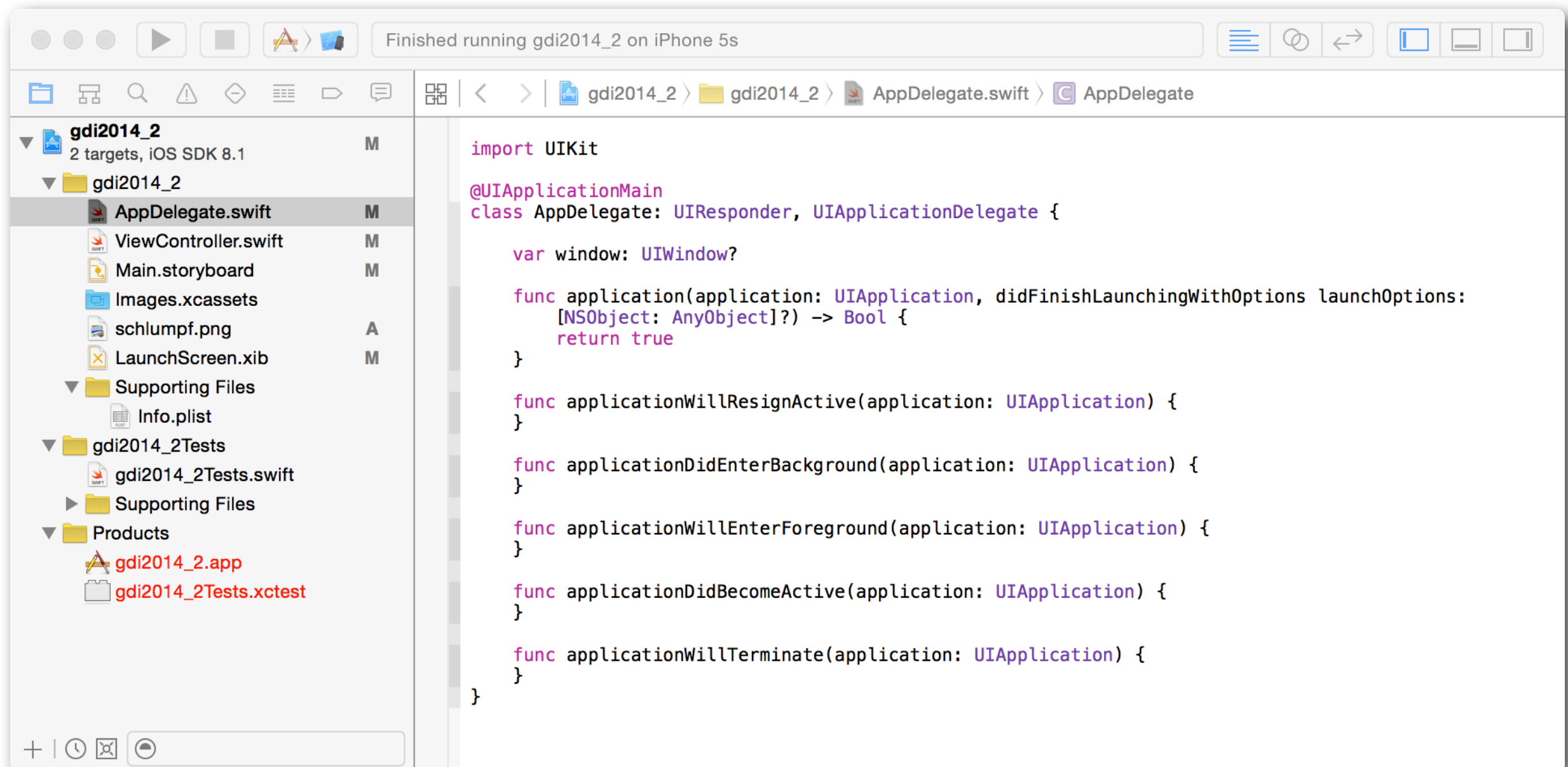
# App Programmieren

- Xcode
  - nur auf Intel-Macs
  - Cross-Compiler für ARM
  - Debugger gdb
  - besondere Lizenz ...
  - Mac mini im Pool
- Interface Builder
  - graphischer Entwurf der Oberfläche
  - Zuordnung UI-Elemente zu Methoden
- Simulator
  - auf dem Mac
  - interagiert mit Debugger: Breakpoints etc
  - simuliert multitouch
- Device
  - iPod Touch, iPad, iPhone
  - mit USB-Dock



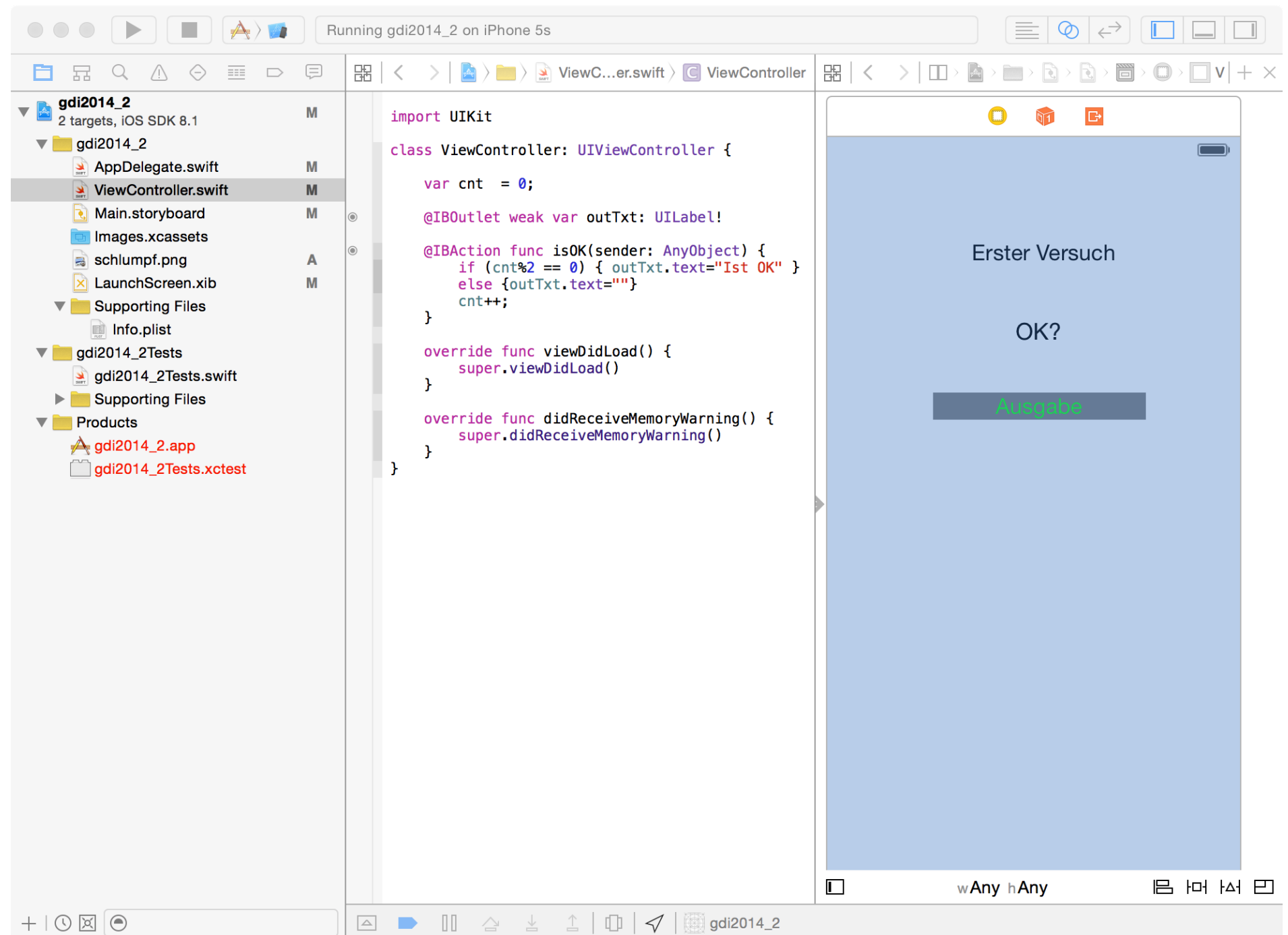
- Xcode

- Neues Projekt aus Projektvorlagen
- generiert Delegate-Klasse und View Controller Klasse
- Storyboard
- Resources: Dateien, Bilder, ...



- Storyboard

- Tool zur grafischen Design der User Interface
- viele Klassen in Library
- Label
- Button
- Text-Feld
- ScrollView
- imageView
- Searchbar
- ...
- Eigenschaften
- Verbindungen



- AppDelegate Interface

- Ausgabeelemente: IBOutlet
- Methoden für Eingabe-Elemente
- (IBAction) als Resultat
- evtl. weitere DelegateIF

```
import UIKit

class ViewController: UIViewController{
 var cnt = 0;

 @IBOutlet weak var outTxt : UILabel!

 @IBAction func isOk(sender: AnyObject){
 if (cnt%2 == 0) {outTxt.text = "Ist OK"}
 else {outTxt.text = ""}
 cnt++
 }
}
```

- AppDelegate Implementation

- Ausgabe: Properties von IB-Elementen setzen
- Methoden für Eingaben implementieren
- Methoden überschreiben

```
func applicationDidBecomeActive(application: UIApplication) { ... }
```

## • Debugger und Simulator

The screenshot displays the Xcode IDE with the following components:

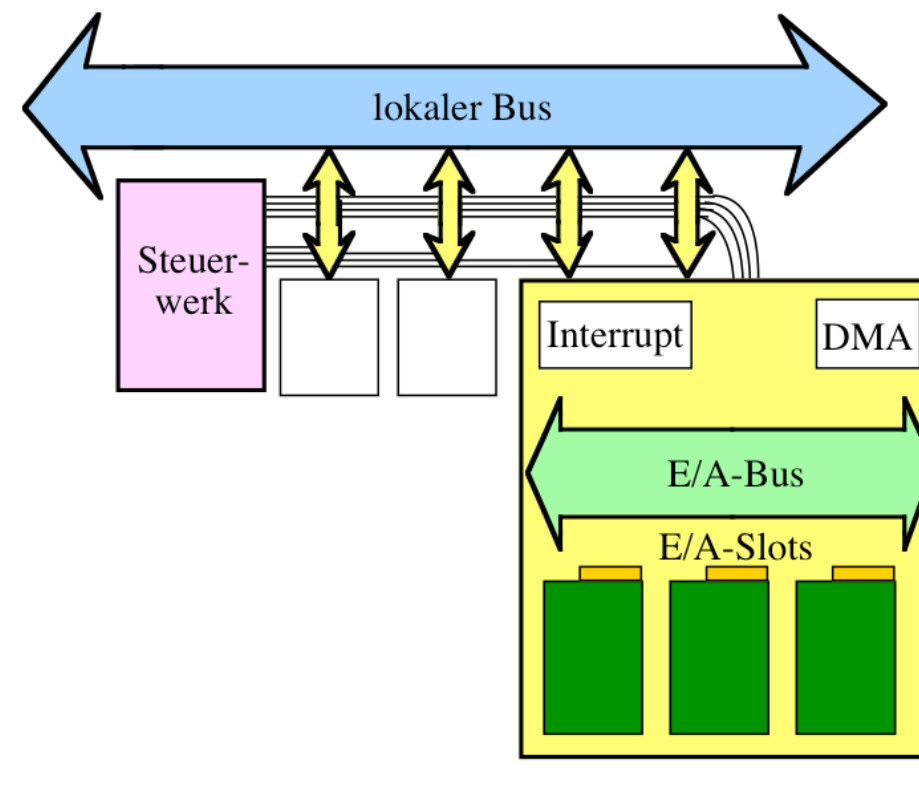
- Top Bar:** Shows the application is running on an iPhone 5s simulator.
- Left Sidebar:** Contains the 'gdi2014\_2' target, which is paused (PID 1804). Below it are system resource monitors for CPU (0%), Memory (19.1 MB), Disk (Zero KB/s), and Network (Zero KB/s). The 'Thread 1' section shows a stack of calls, with the current thread being '0 gdi2014\_2.ViewController.isO...'. Other threads (Thread 2, Thread 6) are also listed.
- Source Editor:** Displays the Swift code for 'ViewController.swift'. A breakpoint is set at line 1.1, which is currently active. The code includes an import for UIKit, a class declaration for 'ViewController' inheriting from 'UIViewController', and several methods: 'isOK' (which toggles the text of 'outTxt' based on a counter), 'viewDidLoad', and 'didReceiveMemoryWarning'.
- Bottom Panel:** The 'lldb' (LLDB) window shows the current state of the program. It lists variables: 'sender' (UIButton), 'self' (ViewController), 'cnt' (Int, value 3), and 'outTxt' (UILabel). The 'Some' variable is also shown as a UILabel.
- iOS Simulator:** A separate window on the right shows the simulated iPhone 5s screen. It displays the text 'Erster Versuch' and 'OK?' with a green button labeled 'Ausgabe'.

# Rechnerarchitektur

- Vom Transistor zum Programm
- Schaltfunktionen
- Hardwarestrukturen
- Transport, Speicherung, Rechnen

*Software wird schneller langsam als Hardware schneller wird [M. Reiser]*

*Hardware: The parts of a computer system that can be kicked. [Jeff Pesis]*



# Transistoren, Gates

- Schaltfunktionen
  - 1..n Eingänge, 1..m Ausgänge
  - $2^n$  Argumentkombinationen
  - $2^{2^n}$  mögliche Funktionen
- 2-stellige Schaltfunktion

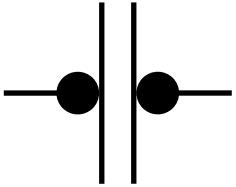
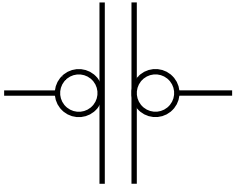
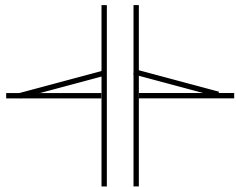
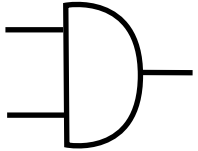
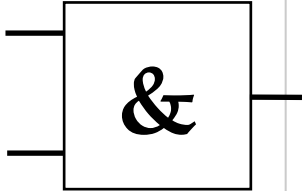
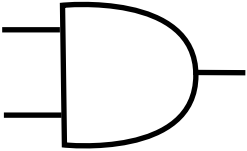
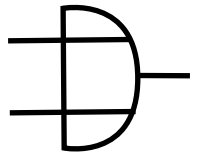
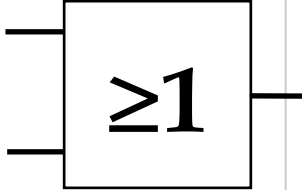
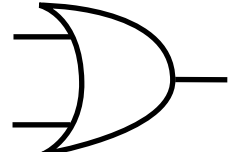
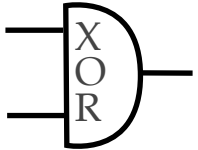
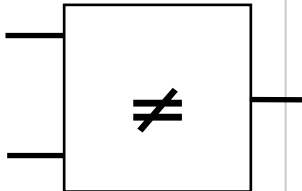
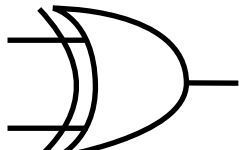
$V_1$	$V_2$	$V_3$	...	$V_n$	$f_1$	$f_2$	...	$f_{\max}$
1	1	1		1	0	0		1
0	1	1		1	0	0		1
0	0	1		1	0	0		1
0	0	0		0	0	1		1

		A=1, B=1	A=1, B=0	A=0, B=1	A=0, B=0
f0		0	0	0	0
f1		0	0	0	1
...		...	...	...	...
f8	AND	1	0	0	0
f14	OR	1	1	1	0
f15		1	1	1	1

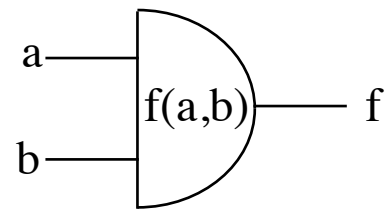
- Symbolische Darstellung von Schaltkreisen

- Schaltplan

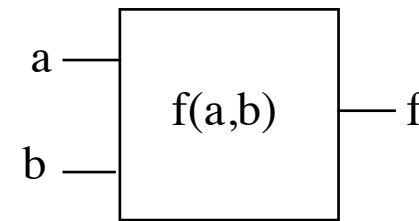
- DIN 40700, IEEE / ANSI Y32.14, IEC

Funktion	Operator	DIN	IEEE	IEC
Negation	$\neg$ , $-$			
Und	$\cdot$ , $\wedge$ , $*$			
Oder	$\vee$ , $+$			
Exklusiv Oder	$\oplus$ , $\neq$			

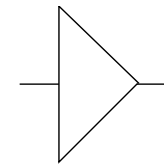
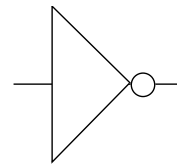
- Allgemeine Schaltfunktionen:



bzw.

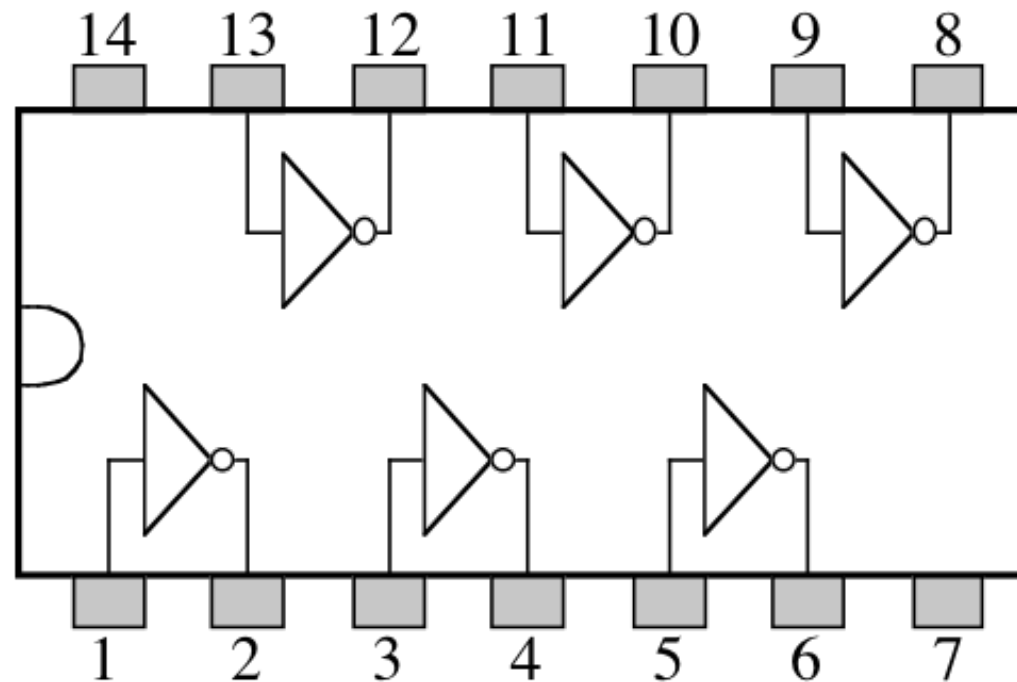


- Inverter, Verstärker:



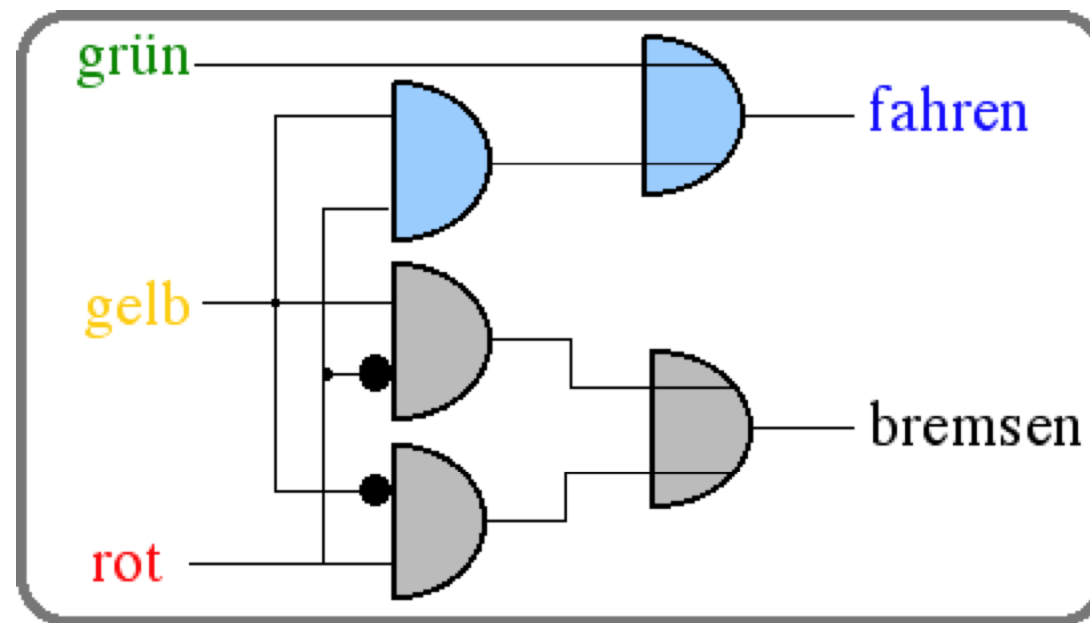
- Hex Inverter 7404 als Chip:

- Stift #7: 0 Volt, Erde, GND
- Stift #14: z.B. +5 Volt:





- Ampel-Reaktion



- Schaltalgebra
  - Schaltfunktionen
  - Rechenregeln
  - Normalform
  - Minimierung

- ODER (OR) Schaltung

- $a \cup b$

- Disjunktion, logische Summe, Vereinigung

- UND (AND) Schaltung

- $a \cap b$

- Konjunktion, logisches Produkt, Durchschnitt

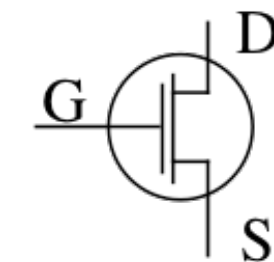
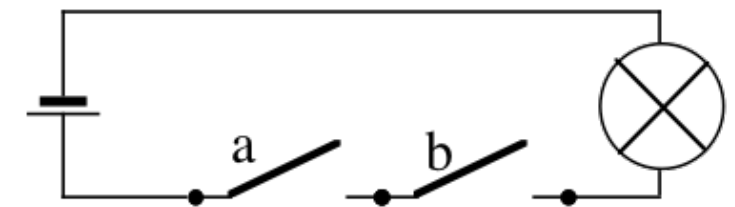
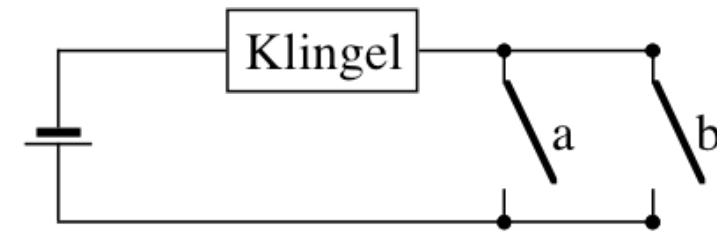
- Feldeffekttransistor (**FET**)

- Gate-Source-Spannung erzeugt Feld

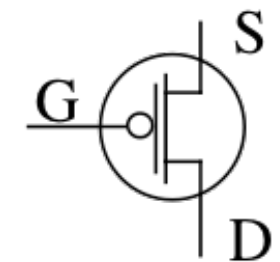
- Feld kontrolliert Stromfluss im Drain-Source-Kanal

- $U_{GS}$  steigt  $\rightarrow I_{DS}$  steigt exponentiell

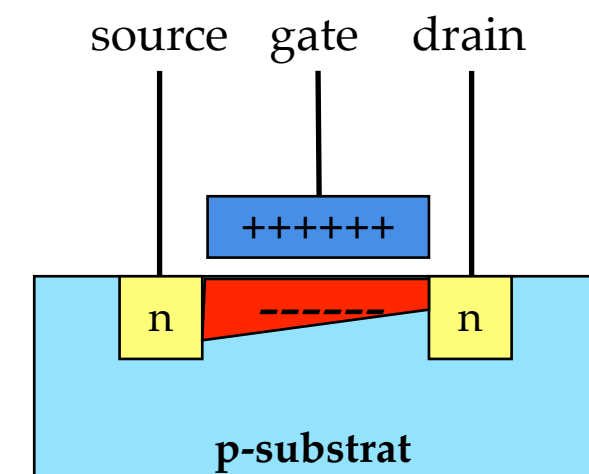
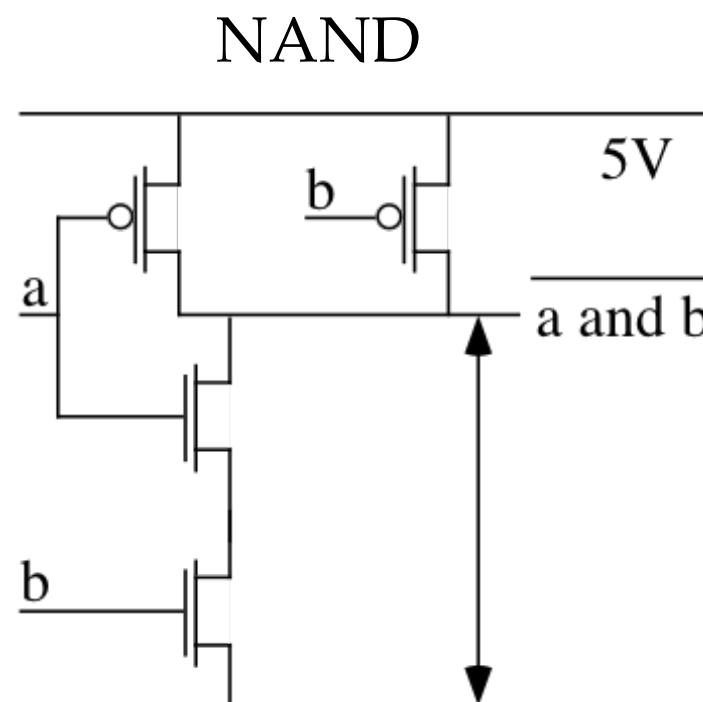
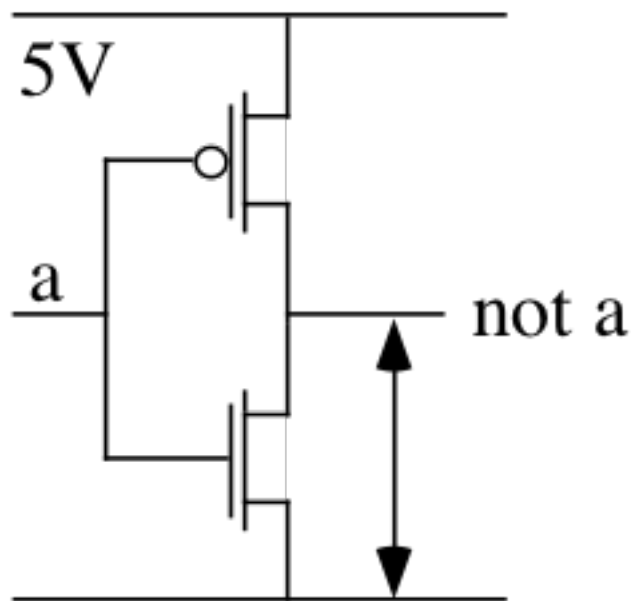
- Negation (NOT)



N-Kanal

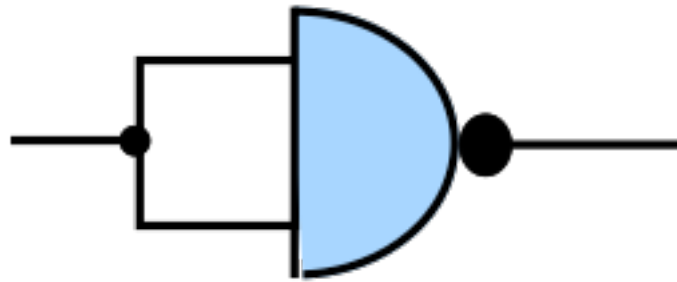


P-Kanal

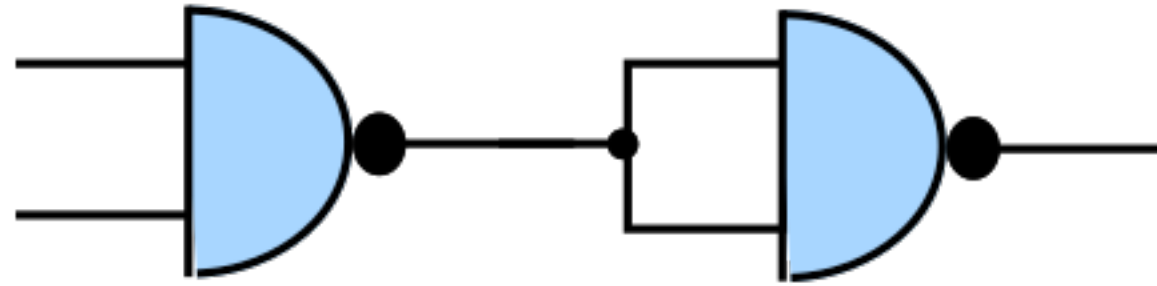


- Boolesche Algebra
- Kombination von NICHT mit UND / ODER
  - $\overline{A \wedge B} = \overline{A} \vee \overline{B}$
  - $A \vee B = \overline{\overline{A} \wedge \overline{B}}$
  - $A \wedge B = \overline{\overline{A} \vee \overline{B}}$
- NAND und NOR einfacher zu bauen
  - CMOS: complementary Metal-Oxide-Silicon
  - 4 Transistoren in CMOS-Technik

Inverter aus 1 NAND



UND aus 2 NANDs



- 74xx, 74HCTxx, ...
  - Einfache Logik-Chips
  - 7404: 6 Inverter
  - 7400: 4 NAND-Gatter
  - 7402: 4 NOR-Gatter
  - 7420: 2 NAND-Gatter mit je 4 Eingängen

# Rechnen und Speichern

- Addition von zwei einstelligen Binärzahlen
  - Eingangsvariablen
  - Summenausgang
  - Übertrag zur nächsten Stufe

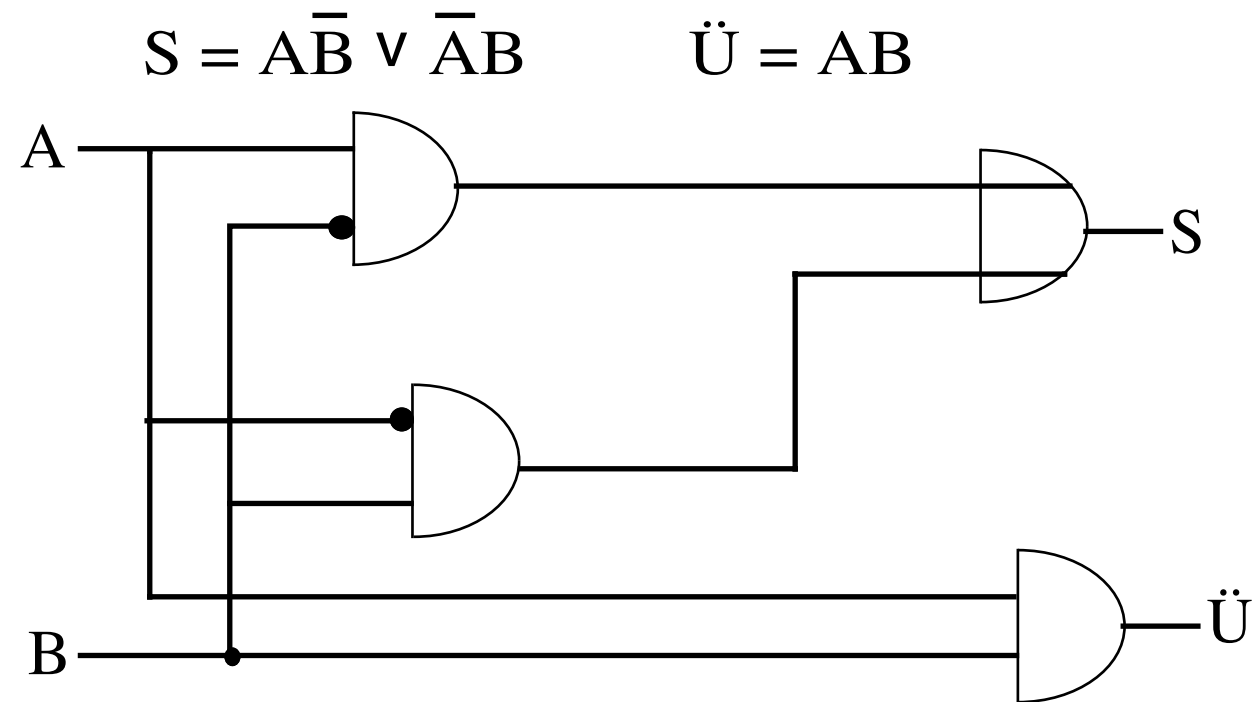


- Wahrheitstafel

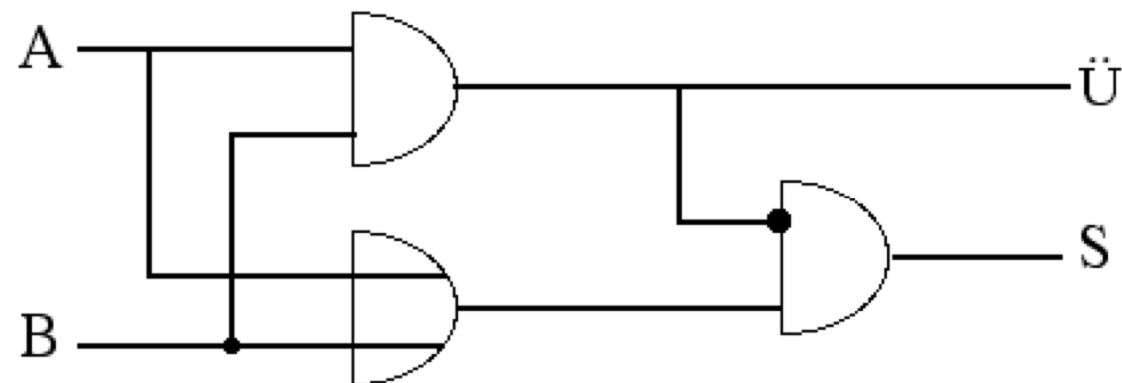
A	B	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Addierer kann aus sogenannten Halbaddieren zusammengesetzt werden
- Halbaddierer berücksichtigt nicht den Übertrag der früheren Stufe

- Halbaddierer

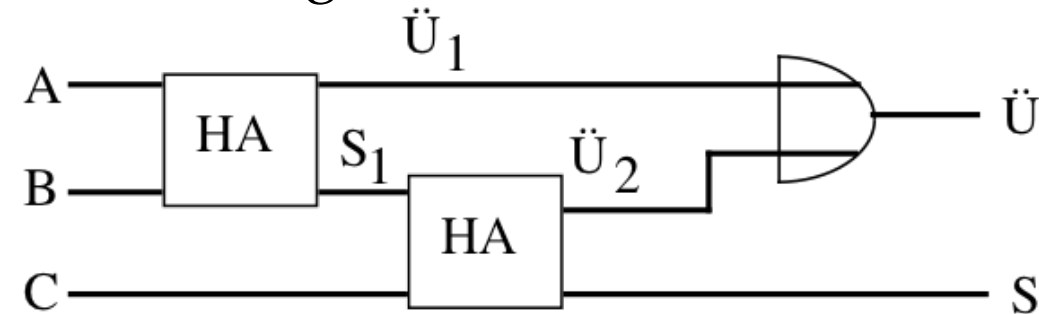


- Vereinfacht



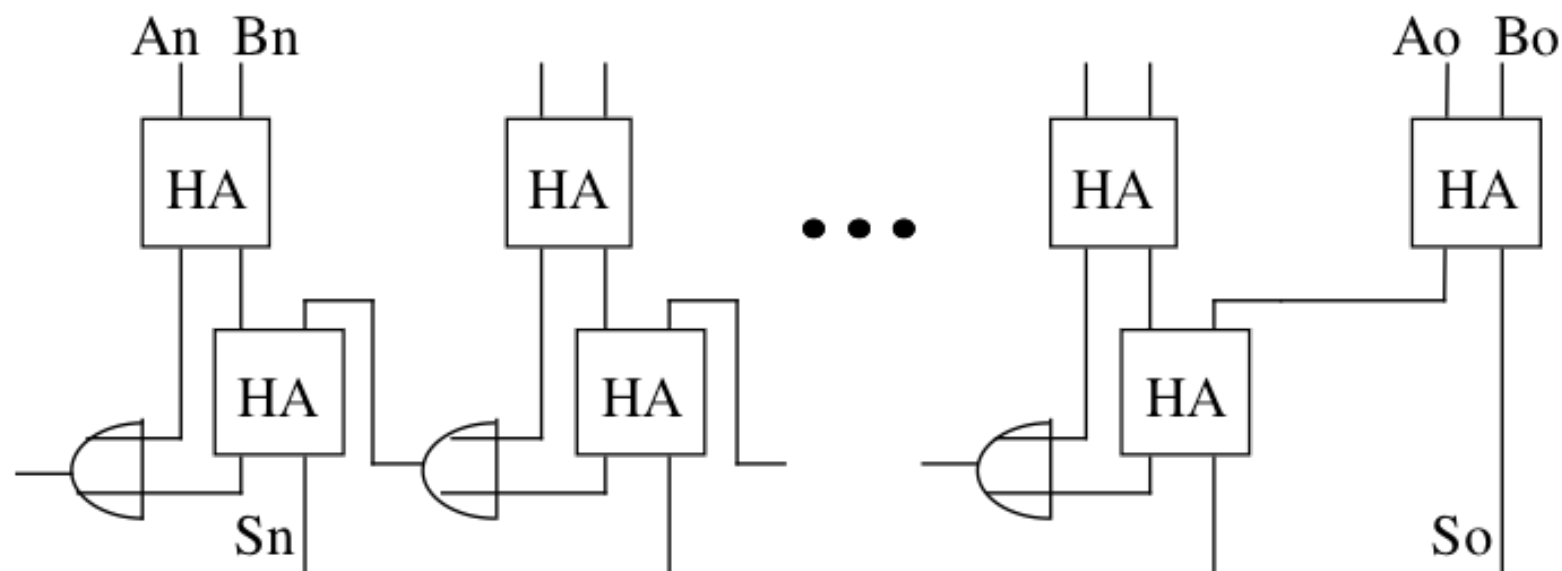
- Volladdierer

- für eine Binärstelle
- berücksichtigt auch den Übertrag einer früheren Stufe



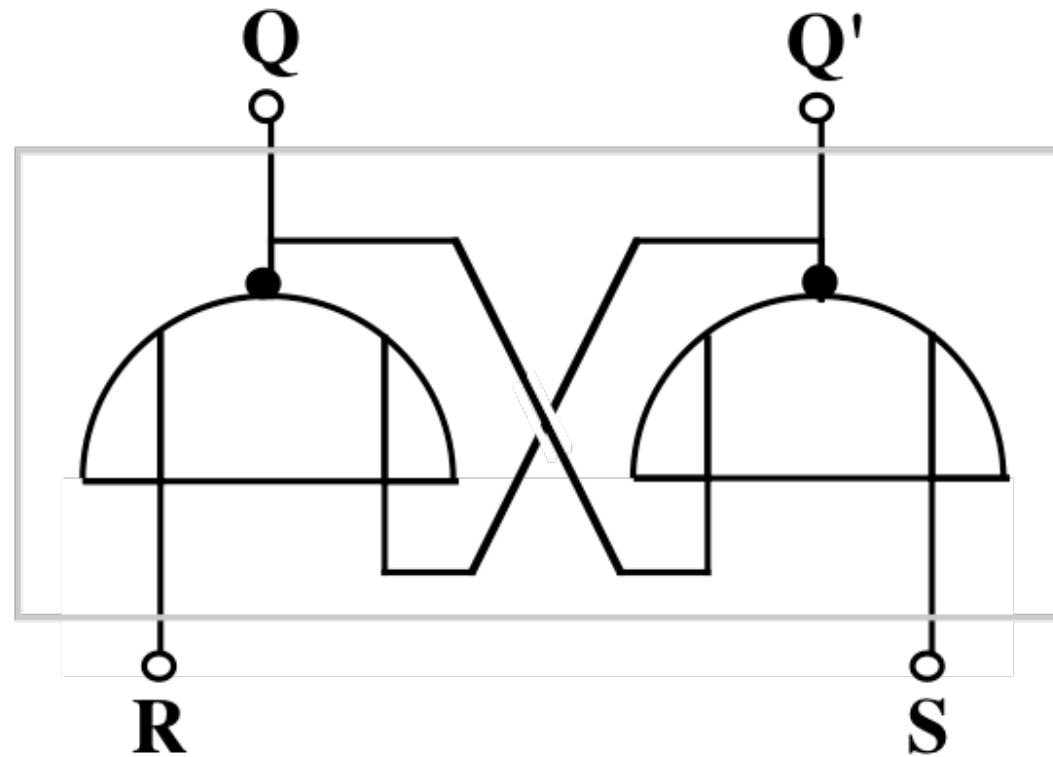
- Paralleladdierer

- für die Addition eines Binärwortes
- die Summen der jeweiligen Binärstellen parallel bilden
- Übertrag durch die Stufen fortpflanzen lassen (Delay!)

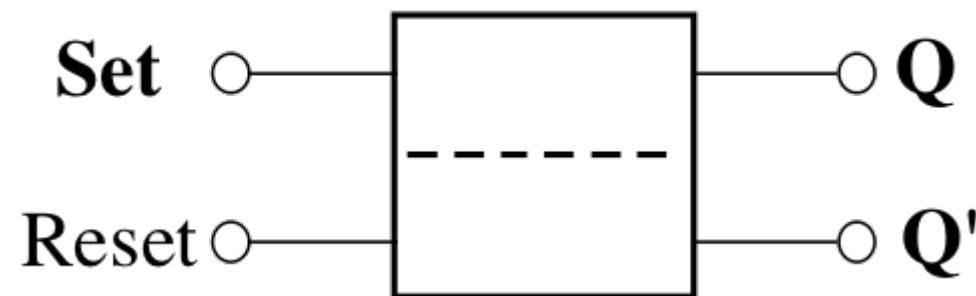


-> Carry Lookahead

- Einfache Speicherzelle: Flip-Flop
  - Speichern einer Binärstelle
  - die beiden Hälften halten sich gegenseitig



- sog. RS-Flip-Flop:
  - Setzen ("Set"),
  - Rücksetzen ("Reset")



- Schaltfunktion für RS-Flip-Flop:

- $Q = \overline{R \vee Q'} = \overline{R \vee (\overline{S \vee Q})} = \overline{R} \wedge (S \vee Q)$

- $Q' = \overline{(\overline{S \vee Q})} = \overline{S} \wedge (R \vee Q')$

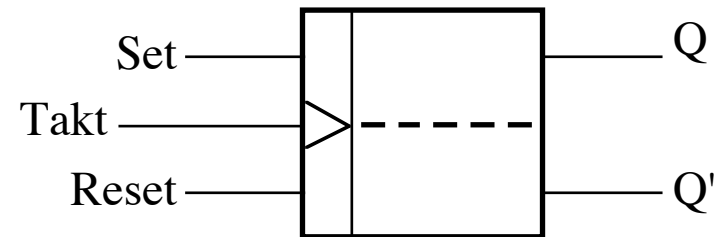
- Wahrheitstafel:

R	S	$Q_n$	$Q'_n$
0	0	$Q_{n-1}$	$Q'_{n-1}$
0	1	1	0
1	0	0	1
1	1	0	0

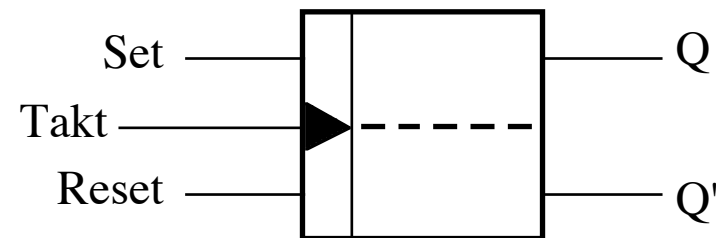
- undefinierter Folgezustand für  $R=1; S=1$
- Zwei Speicherungs-Zustände mit  $R=0; S=0$ :
  - $Q = 0; Q' = 1$
  - $Q = 1; Q' = 0$
  - fast immer  $Q \neq Q' !$
- Zwischenzustände während Umschaltung



- Taktsteuerung
  - Eingangswerte stabilisieren lassen
  - dann Übernahmeimpuls
  - Eingangswerte werden nur zum Taktzeitpunkt berücksichtigt:
- Flankengesteuertes Flipflop (abfallend):

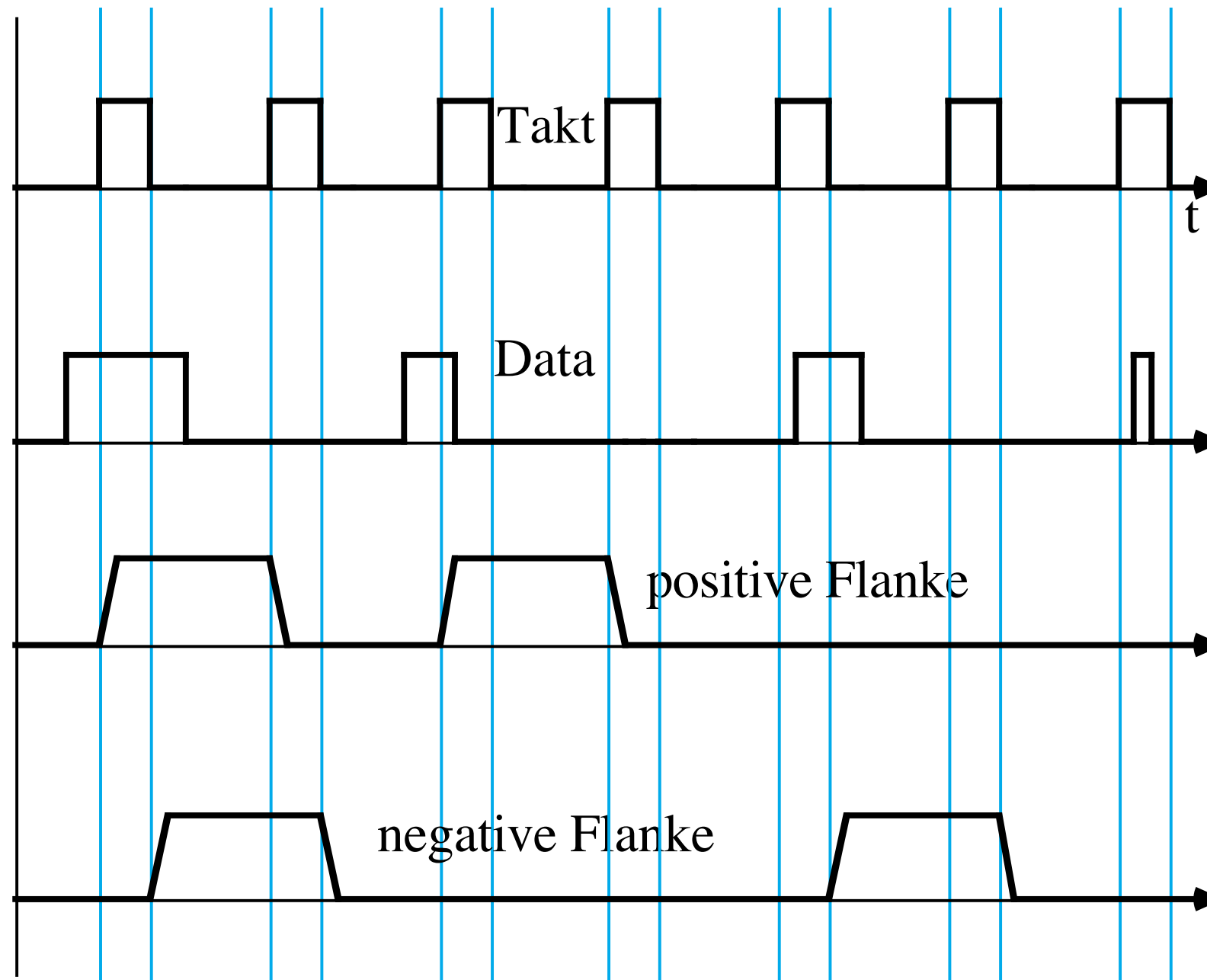


- Flankengesteuertes Flipflop (ansteigend):



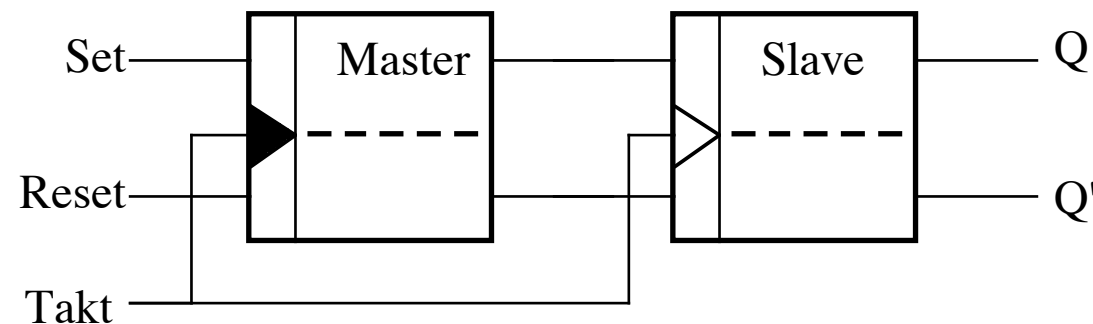
- Signalverlauf

- erst bei entsprechender Taktflanke Eingabe einlesen
- evtl. unterschiedliche Ergebnisse



- Master-Slave Flip-Flop

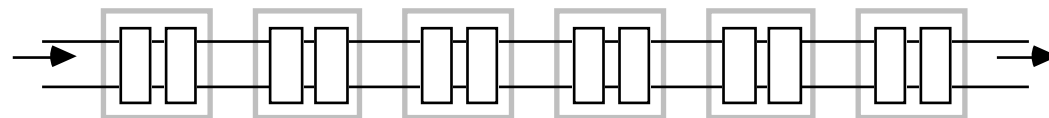
- Zwei FF-Stufen arbeiten phasenverschoben
- Master übernimmt Eingangswerte mit ansteigender Flanke
- Slave gibt mit abfallenden Flanke Werte an Ausgang



- Für Schaltungen mit heiklem Timing

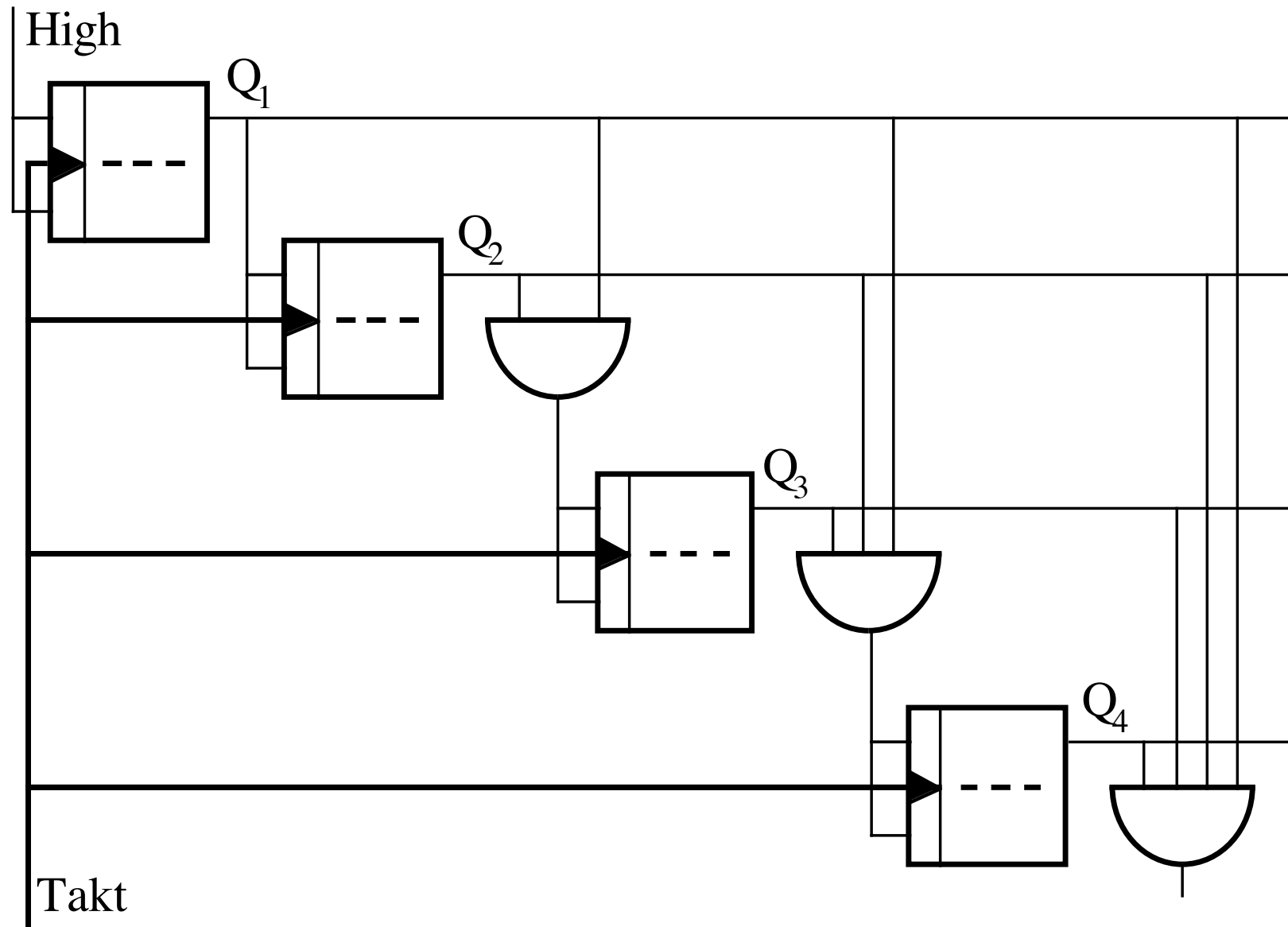
- Registertransfers.
- Pufferregistern mit MS-FFs.

- MS-Schieberegister

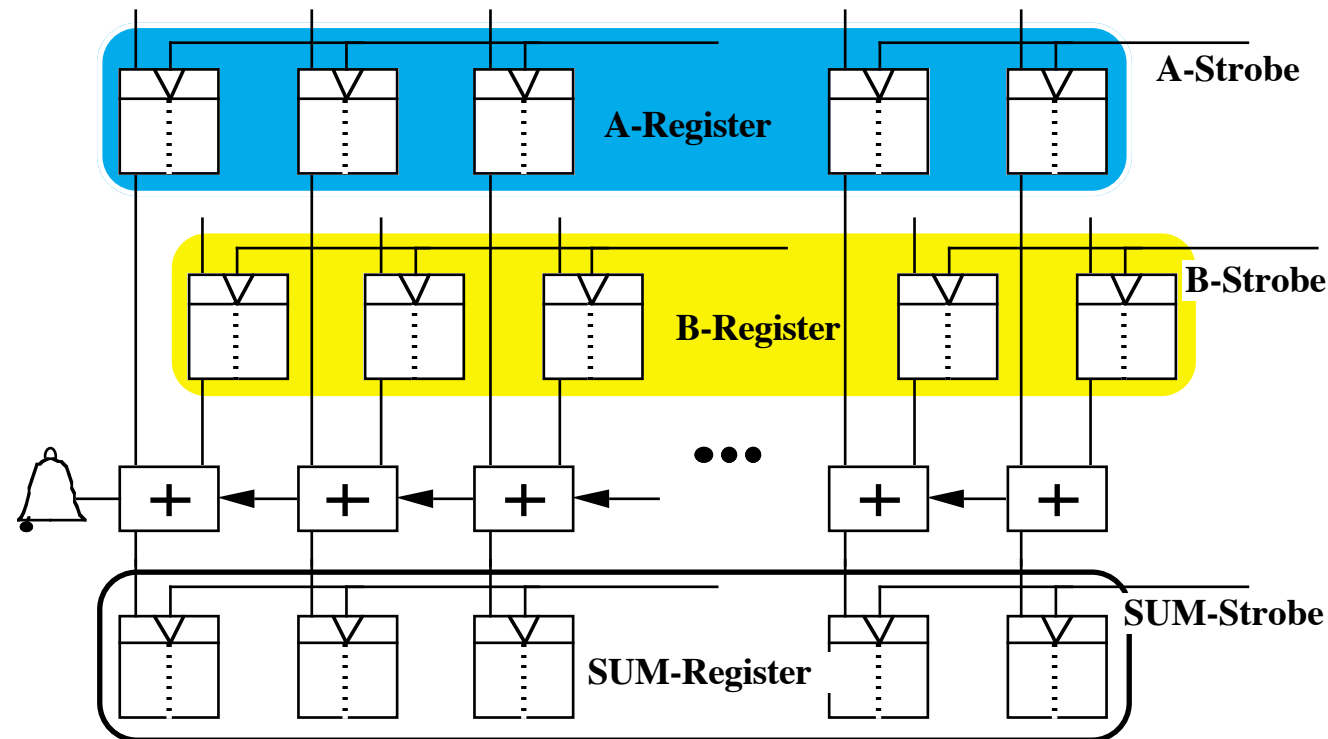


- als digitale Verzögerungsleitung
- für digitale Filter
- als Rechenoperation

- Zähler: D-Flip-Flops
  - Jede Stufe erhält Takt
  - schaltet in Abhängigkeit vom Zustand aller vorherigen Stufen



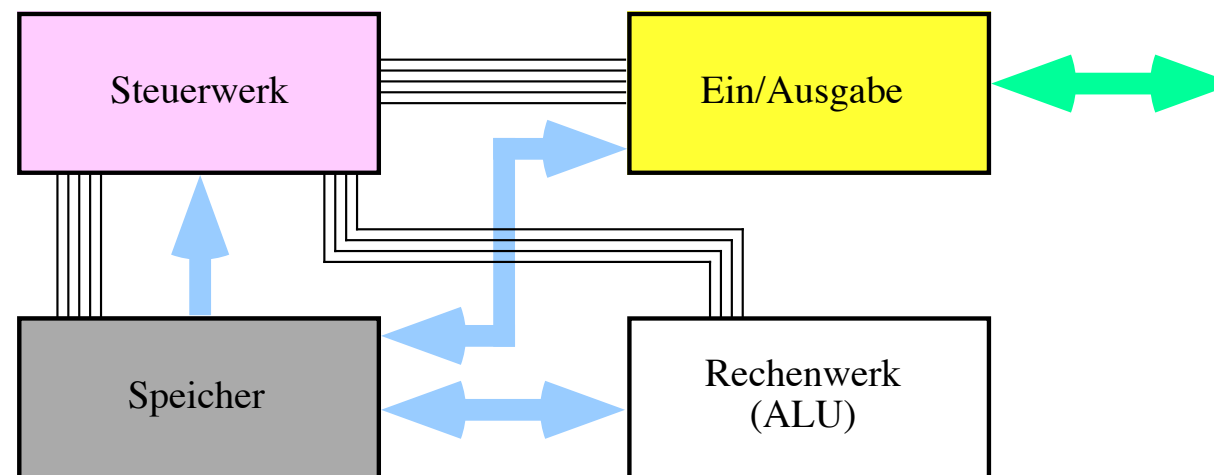
- Register speichert Zahlen
- Register in Verbindung mit Addierschaltung bzw. ALU



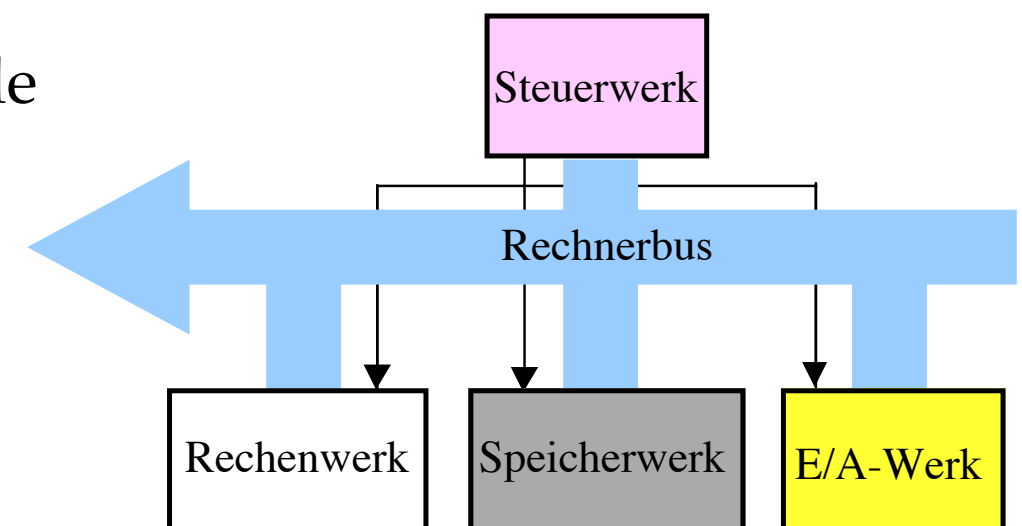
- Ablauf
  - A-Strobe zum Füllen des Registers A
  - B-Strobe zum Füllen des Registers B
  - Addition bzw. Übertrag abwarten,
  - Summe abholen mit SUM-Strobe
  - Überlauf

# Funktionseinheiten: Speicher, Prozessor, Bus, ...

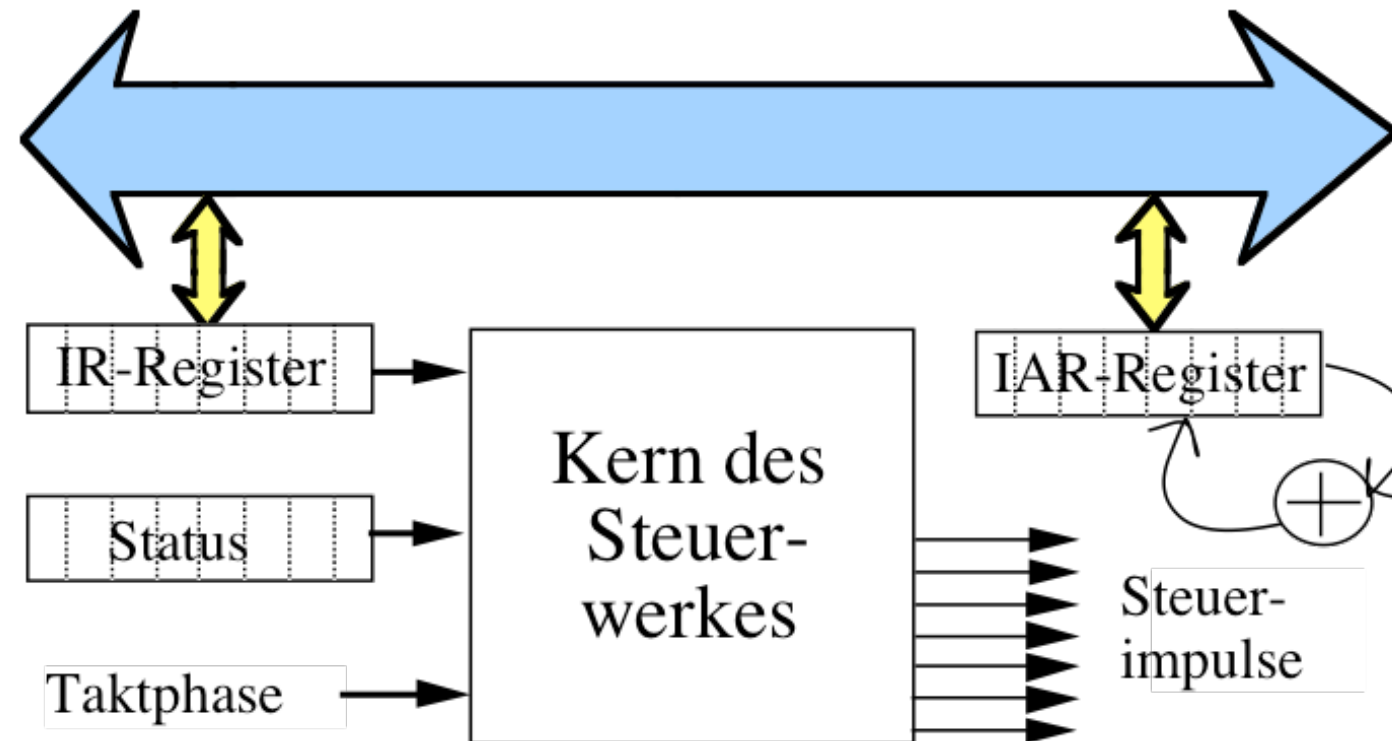
- Klassischer Rechner nach J. v. Neumann
  - Rechenwerk für arithmetische und logische Verknüpfungen
  - Universalspeicher für Daten und Programme



- Steuerwerk
  - sequentieller Ablauf des Programmes
  - Interpretation der Instruktionen => Steuerbefehle
- Busorientierter Rechner
  - universell verbunden
  - Schwerpunkt Transport und Verteilung
  - weniger Verbindungen



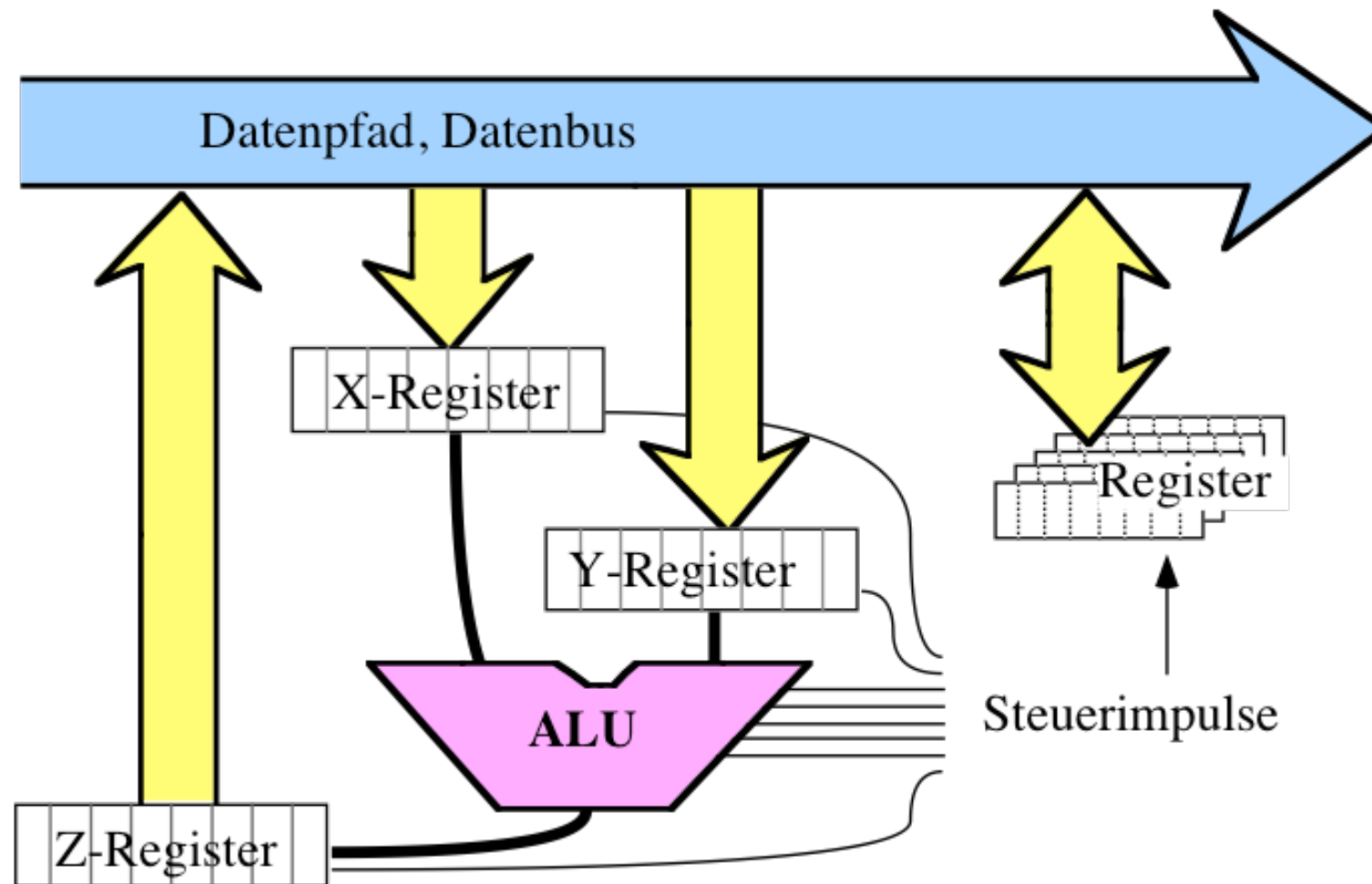
- Steuerwerk erzeugt Signale, die
  - Datentransfer auslösen
  - ALU-Operation auswählen
  - Speicheroperation auslösen



- Befehl wird in Sequenz von Kernzuständen umgesetzt
  - Kernzustand bestimmt Steuersignale
- Register
  - Instruktionsregister (IR)
  - Instruktionsadressregister (IAR) bzw. "Program Counter" (PC)
  - eventuell eigener Addierer für IAR

- Rechenwerk

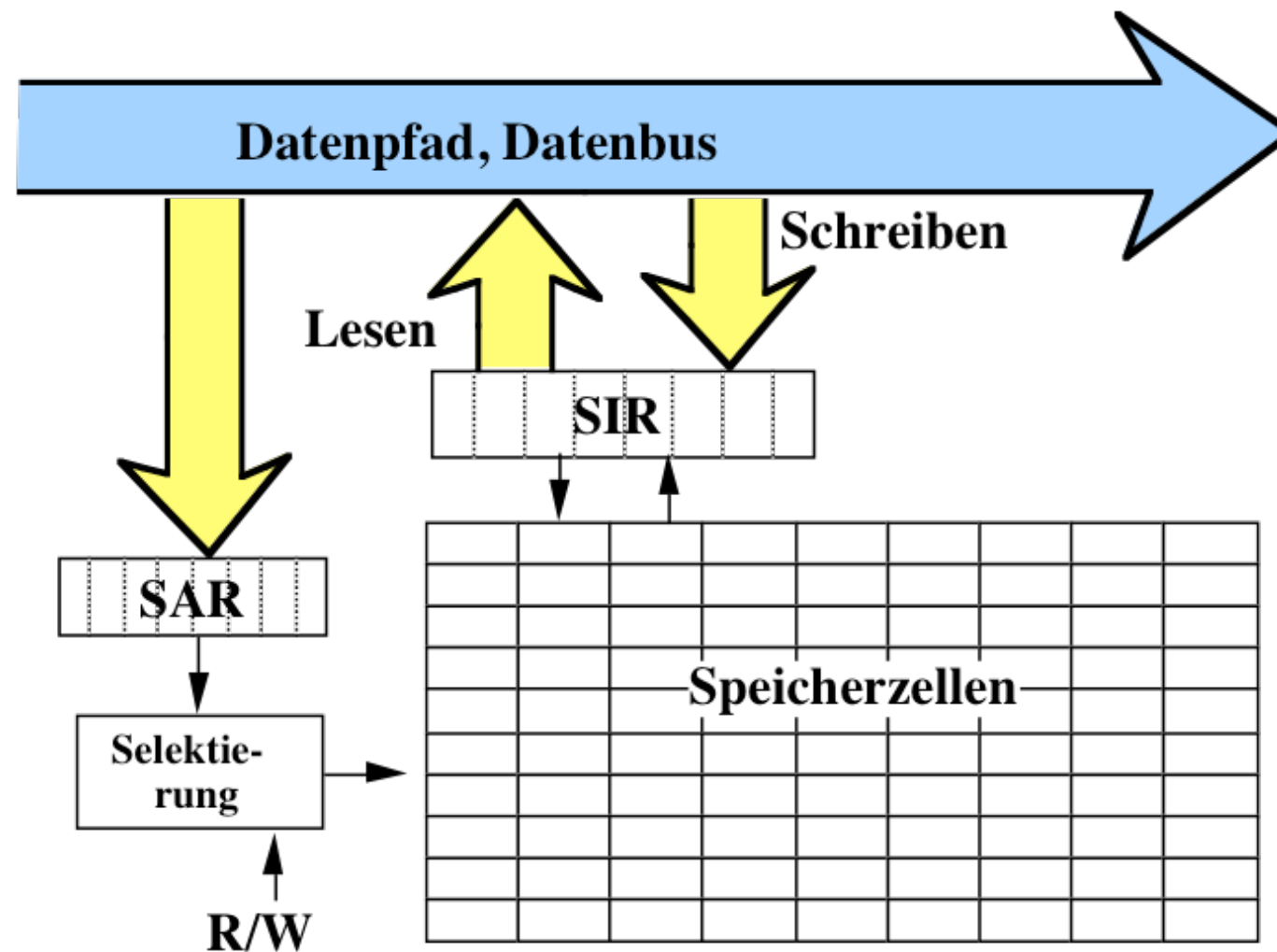
- kombinatorisches Schaltnetz
- Addieren, Multiplizieren, UND, ODER, Vergleich, Shift, ...
- Ausgang der ALU liegt dauernd am Z-Register an
- X-Register und Y-Register liegen dauernd am ALU-Eingang an



- Zwischenresultate in Zusatzregistern
- Steuerleitungen bewirken Datenübernahme = Transport



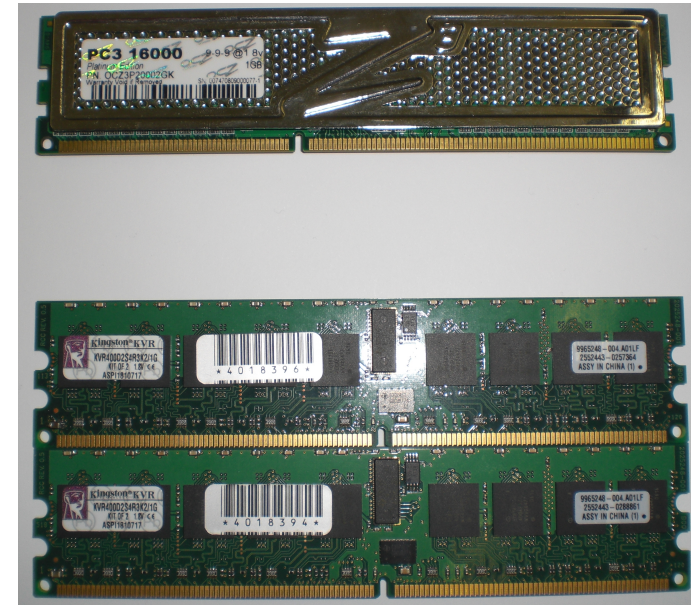
- Speicher



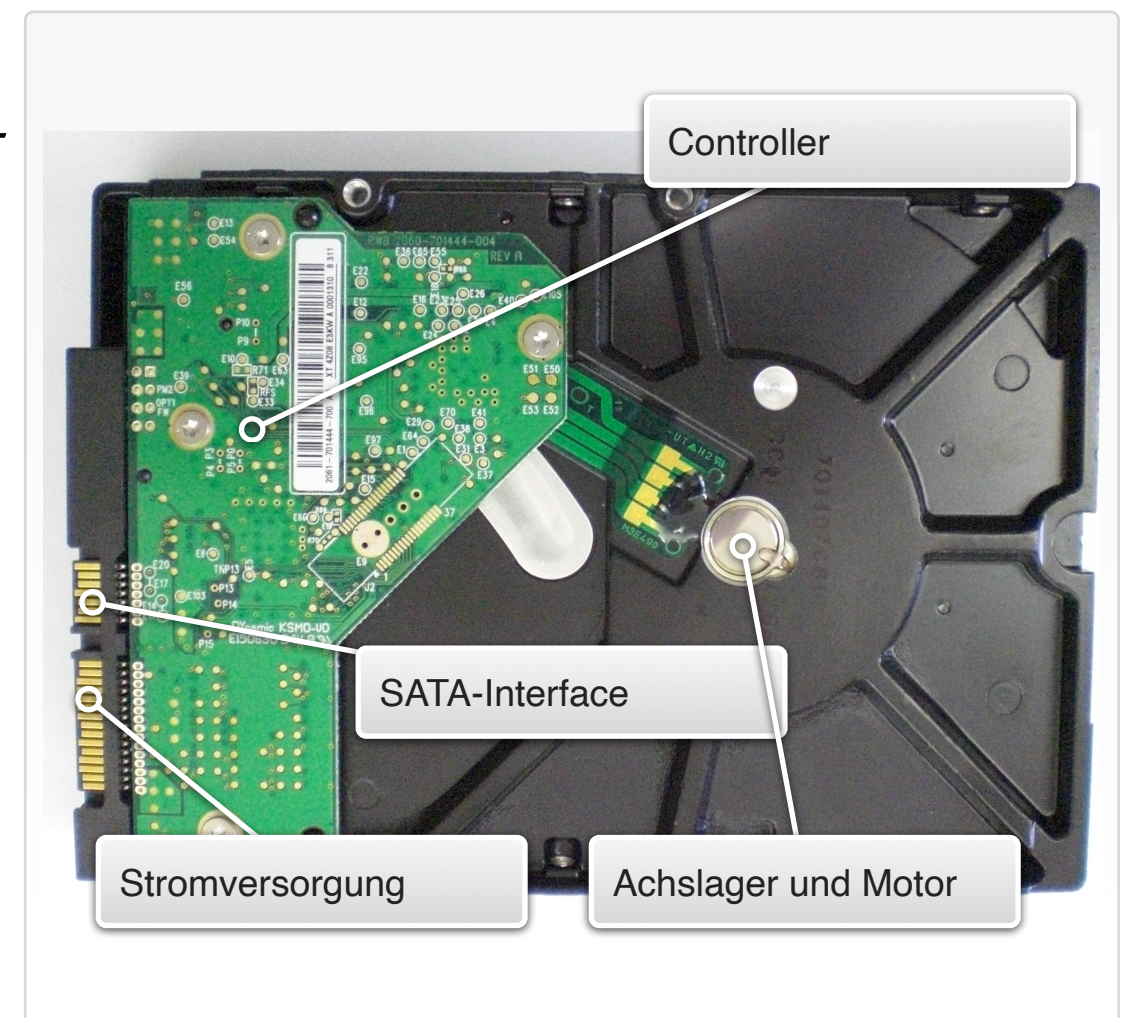
SAR = Speicheradressregister, SIR = Speicherinhaltsregister

- Random Access Memory (RAM)
  - Halbleiterspeicher halten Inhalt nur wenn sie "unter Strom" stehen
  - dynamische Speicher (DRAM) müssen periodisch aufgefrischt werden
- ROM: Festwertspeicher

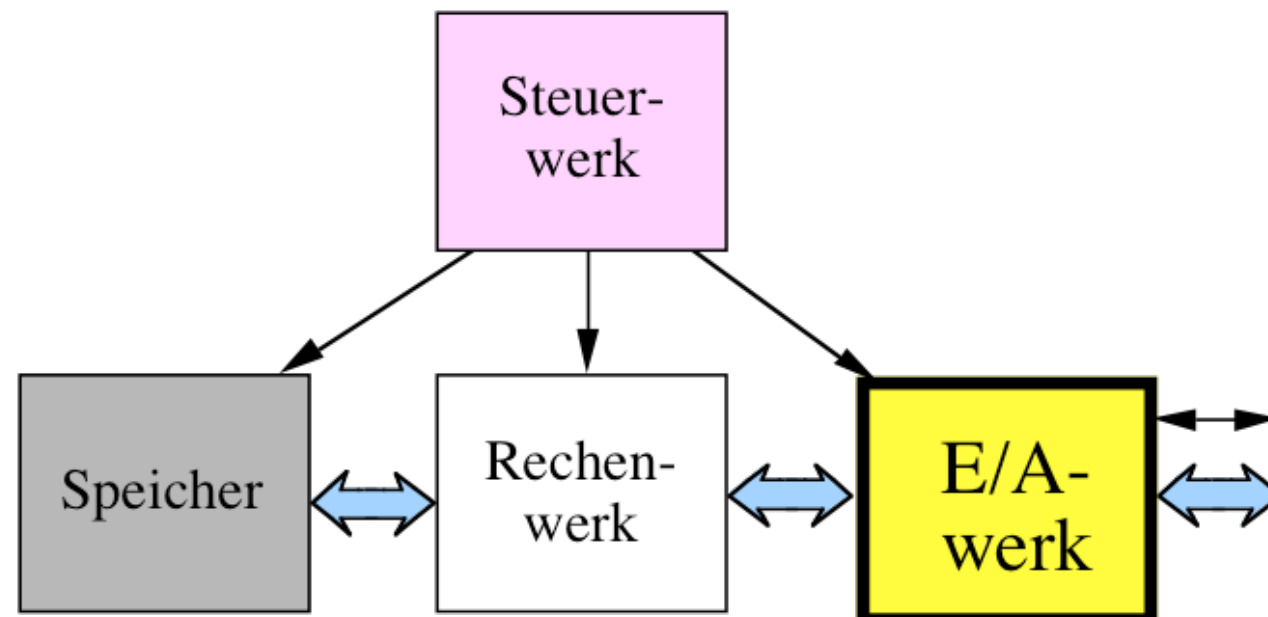
- Speicherhierarchie aus Kostengründen
- Register
  - 10-100 Wörter, < 1 Nanosekunde
  - kein separater Zyklus für die Adresse.
- Cache, Prozessorpufferspeicher
  - 8 KWörter bis 8 MBytes, < 10 Nanosekunden
  - häufig gebrauchte Daten und Codeteile
  - für Programmierer unsichtbar
- Hauptspeicher
  - 256 - 16384 Megabytes, ~ 10 - 50 Nanosekunden
  - evtl. inklusive Adressübersetzung
  - separater Zyklus für die Speicheradresse
- Hintergrundspeicher
  - Festplatte, Netz,
  - 40 - 3000 Gbytes, ~ Millisekunden
  - Zugriff über Betriebssystem
  - blockweise übertragen



**Interactive 6.1** Sata-Festplatte 500 GB

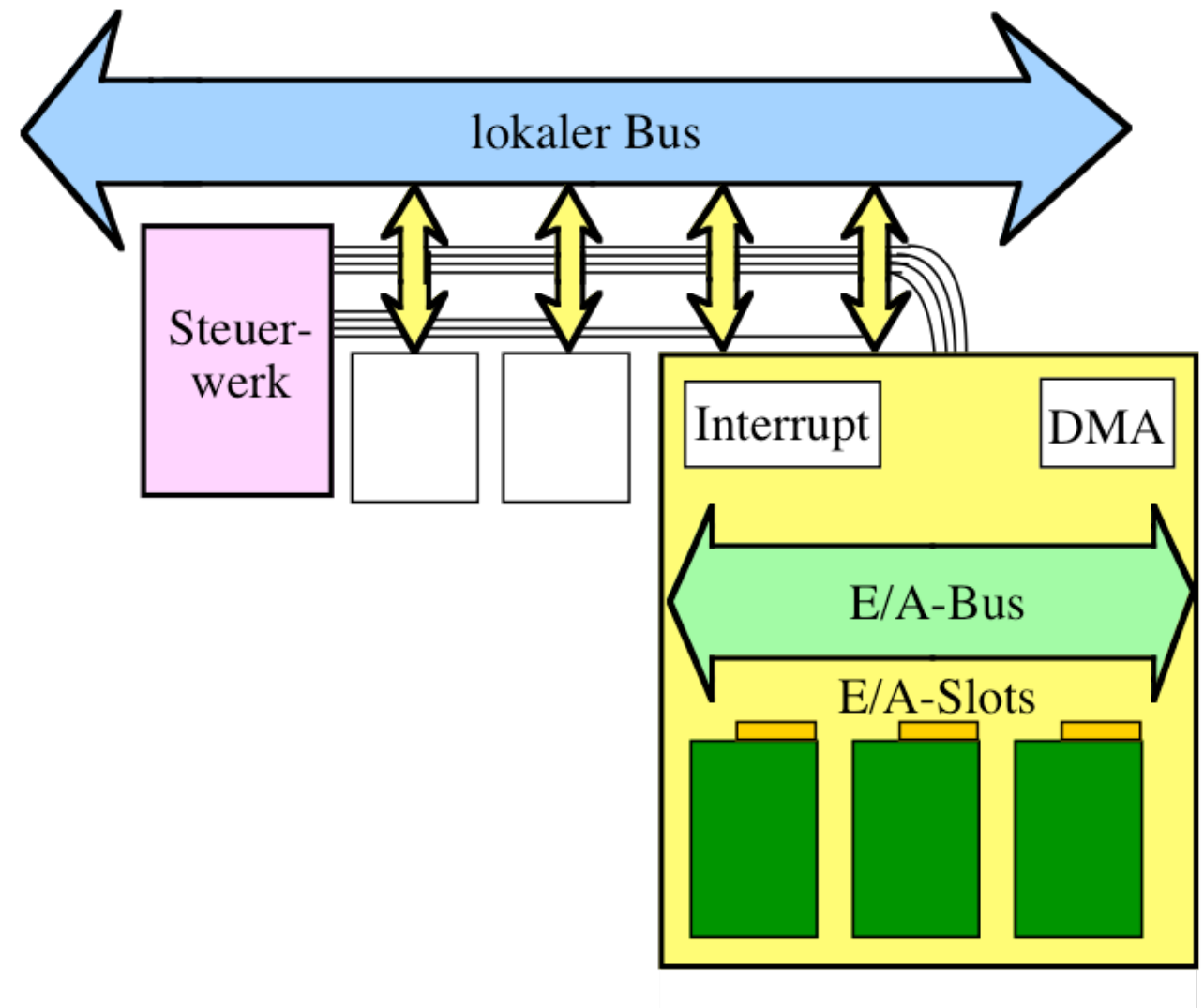


- Ein- / Ausgabewerk
  - kontrolliert durch Steuerwerk
  - Transport via Rechenwerk in den Speicher
  - Bedienungsleitungen zum Peripheriegerät
  - Keine Parallelarbeit von Rechenwerk & E / A
  - Missbrauch der Rechenregister
  - Wartezyklen der CPU auf Peripheriegeräte



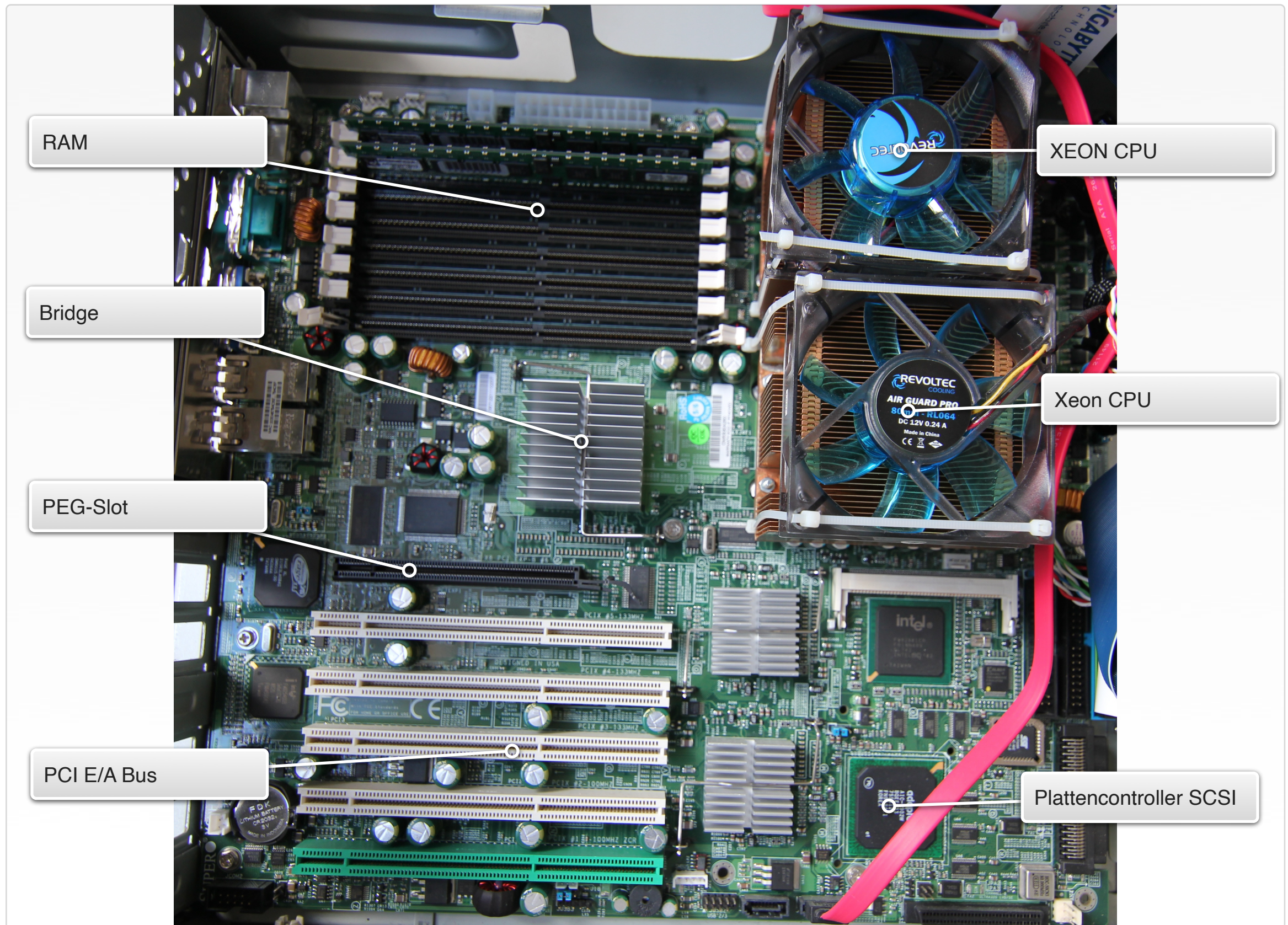
- Pfad in den Speicher als Flaschenhals
  - Resultate von Berechnungen
  - E / A-Übertragungen
  - Instruktionen

- Moderne Rechner
  - besonderer, standardisierter E / A-Bus
  - getrennt vom Prozessorbus
- autonomere E / A-Schnittstelle
  - externer Bus,
  - Abläufe parallel zum Rechenwerk
  - Interrupt-Funktion
  - DMA-Kanäle  
(direct memory access)
  - Display-Kontroller
  - Adapterplatinen
  - VLSI-Chips
- Grafik evtl. Extrabus
  - AGP
  - PCIe, PEG





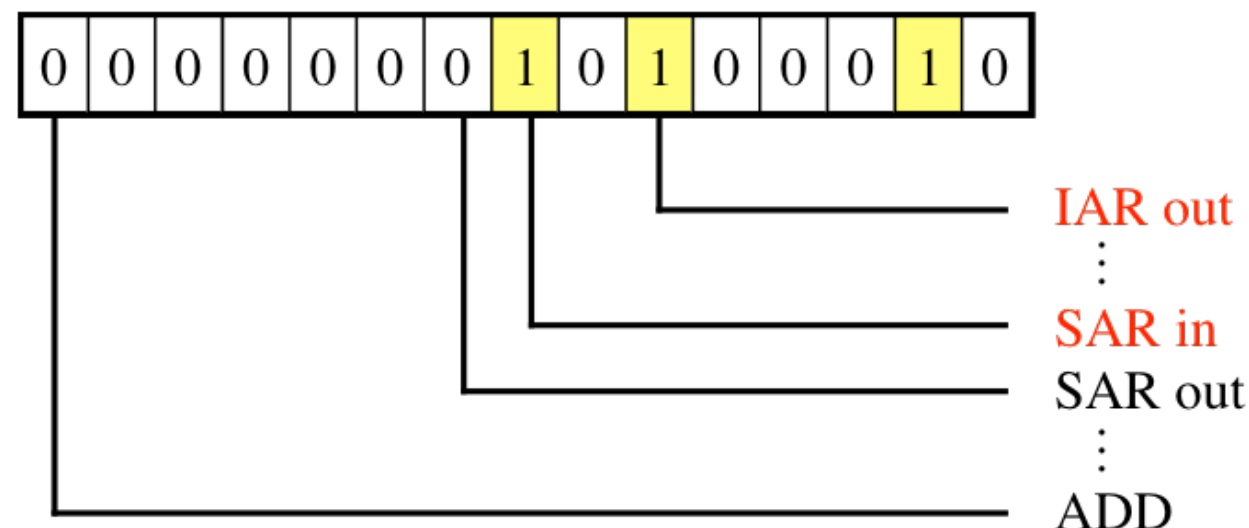
## Interactive 6.2 Severboard Intel Xeon ca. 2009





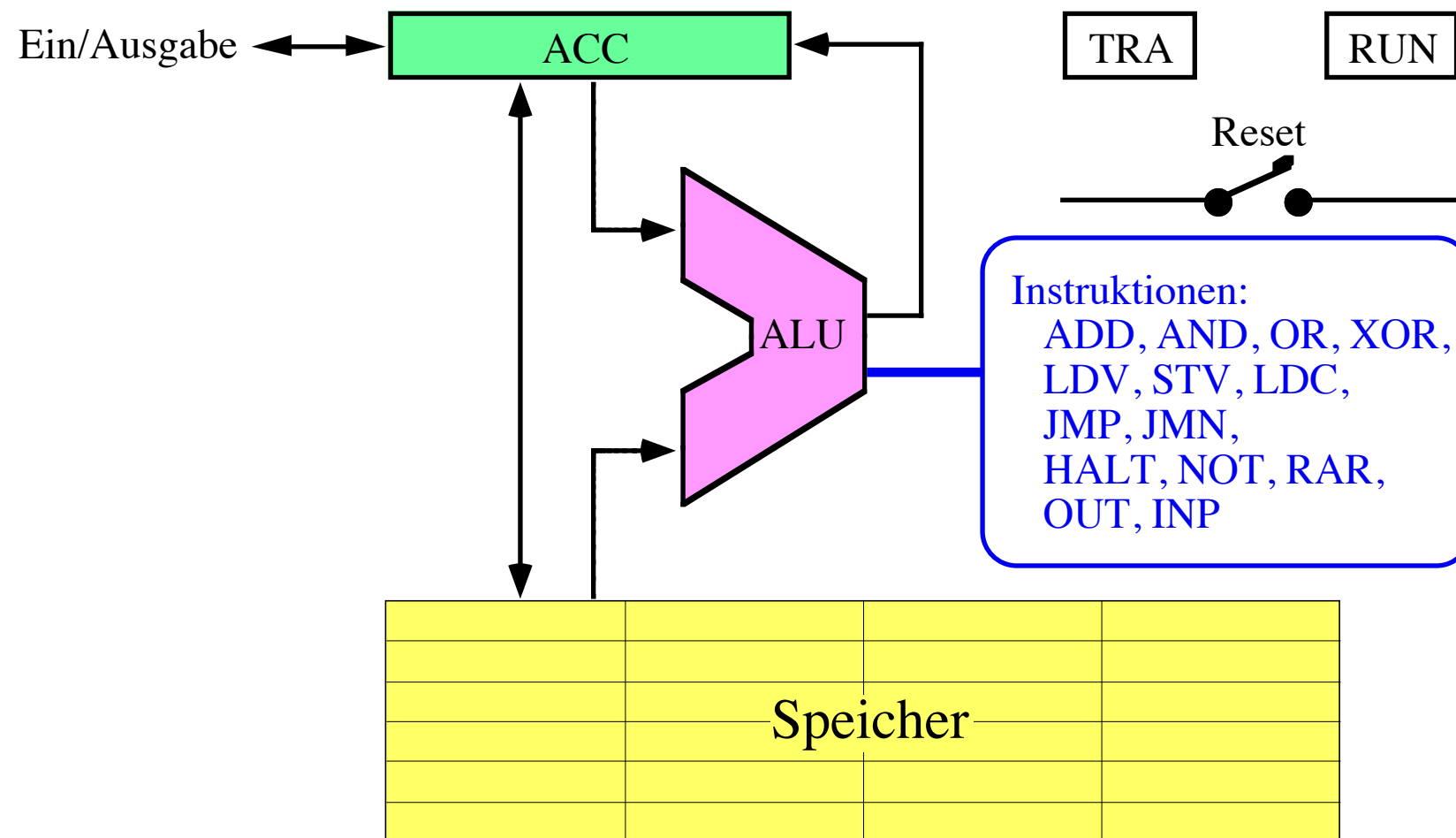
# Maschinenbefehle

- Befehle liegen im Speicher
  - werden geholt wie Daten
  - kommen in Befehlsregister
  - Befehlszähler
- Befehl  $\otimes$  Status  $\Rightarrow$  Sequenz von Steuerworten
  - Bits des Steuerwortes an Steuerleitungen anlegen
  - Speicher, E/A, ALU, ... reagieren auf Steuerleitungen
  - in jedem Takt ein Steuerwort
  - Befehle können mehrere Steuerworte enthalten
- Mikrobefehle



# Minimalmaschine

- ein Register
- nur wenige, einfache Instruktionen
- TRA: Warten auf E / A
- 32-Bit Architektur



- Befehle zur Datenmanipulation

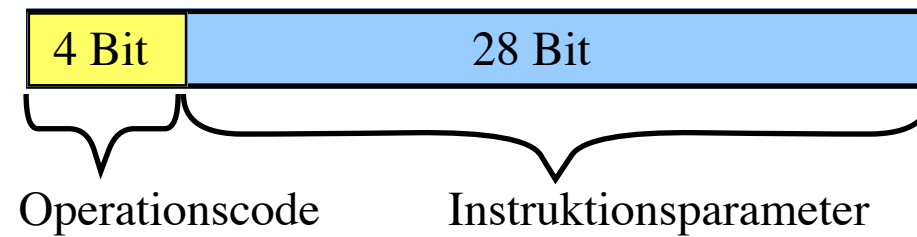
NOT		Komplementiere den Akkumulator, B-1 Komplement
RAR		Rotiere den Akkumulator um 1 Bit nach rechts
ADD	a	Addiere den Inhalt der Speicherzelle a zum Akkumulator $ACC \leftarrow ACC + \text{Speicher}[a]$
AND	a	$ACC \leftarrow ACC \wedge \text{Speicher}[a]$
OR	a	$ACC \leftarrow ACC \vee \text{Speicher}[a]$
XOR	a	$ACC \leftarrow ACC \otimes \text{Speicher}[a]$
EQL	a	$ACC \neq \text{Speicher}[a]: ACC \leftarrow 0$ $ACC = \text{Speicher}[a]: ACC \leftarrow 11..1$
LDV	a	$ACC \leftarrow \text{Speicher}[a]$
STV	a	$\text{Speicher}[a] \leftarrow ACC$
LDC	c	$ACC \leftarrow c$

- Kontrollfluß

JMP	p	Nächste Instruktion in Speicher[p]
JMN	p	Sprung nach p falls ACC negativ
HALT		$RUN \leftarrow 0$ Instruktionsausführung hält an Später Restart nur aus Speicher[0] möglich



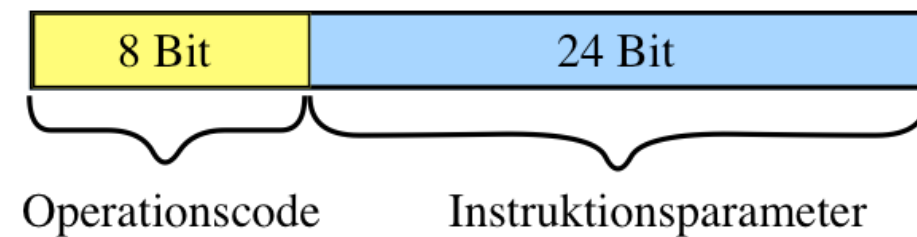
- Basisformat



- 0,1,2,3      ADD, AND, OR, XOR
- 4, 5, 6      LDV, STV, LDC
- 7, 8, 9      JMP, JMN, EQL
- A .. E      future use
- 28 Bit für Speicheradressen

- Erweitertes Format

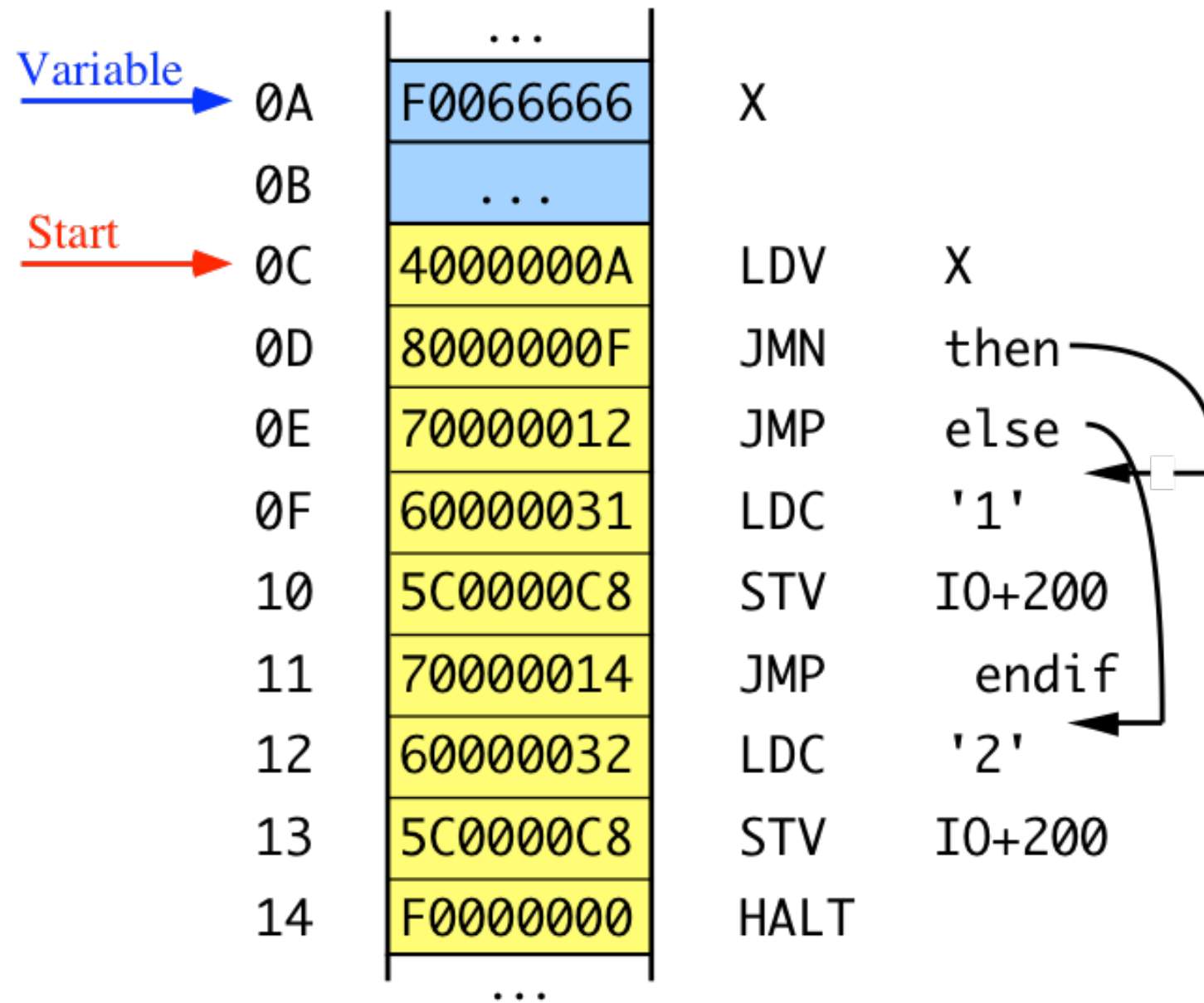
- mehr als 16 Instruktionen möglich,
- 24 Bit Parameter & nicht immer benützt



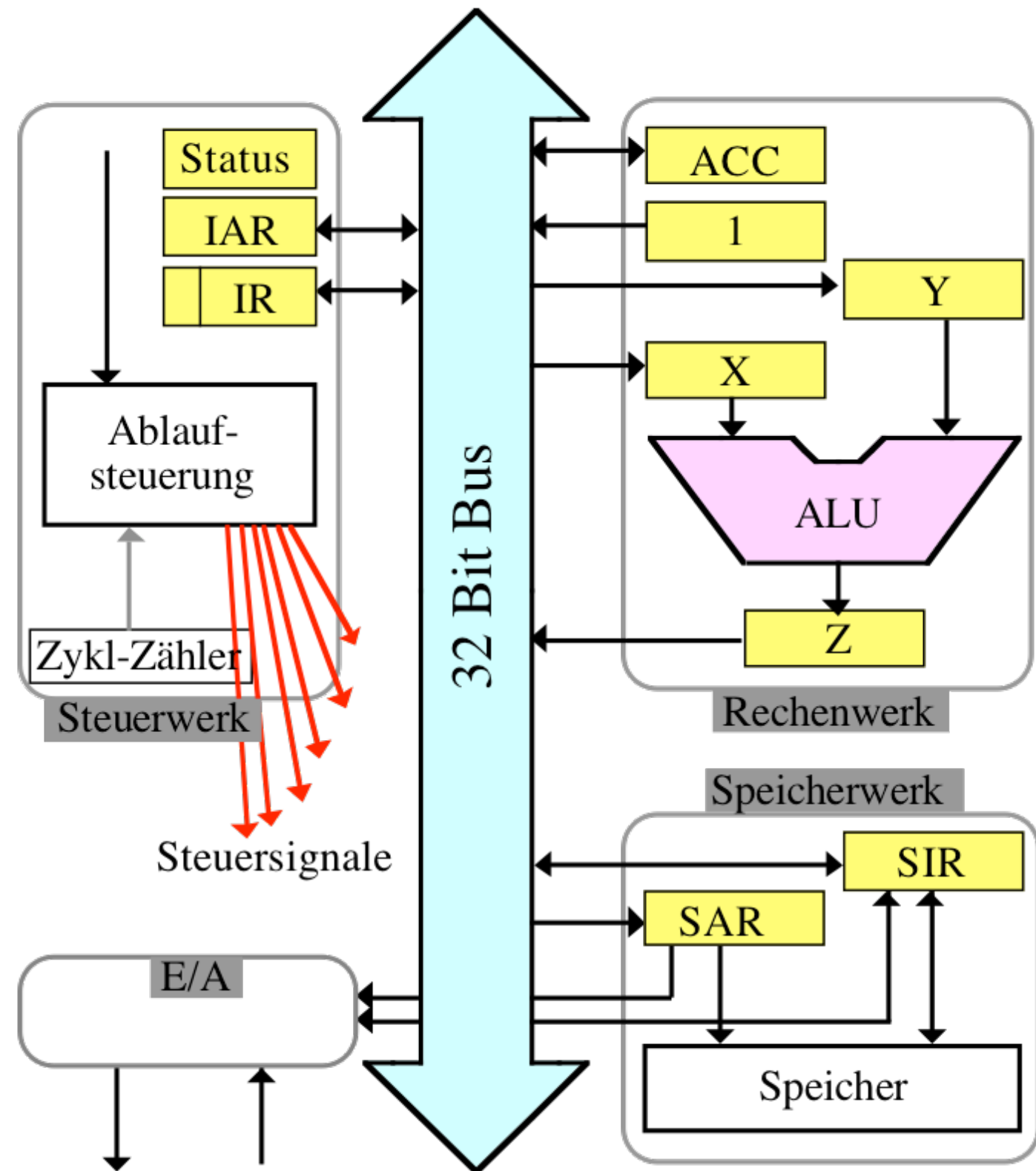
- F0, F1, F2      HALT, NOT, RAR
- F3, F4,...      future use

- Assemblerprogrammieren im Schnelldurchgang  

```
if (X < 0) printf("1")
 else printf("2");
```
- Hexadezimaler MiMa-Speicherauszug

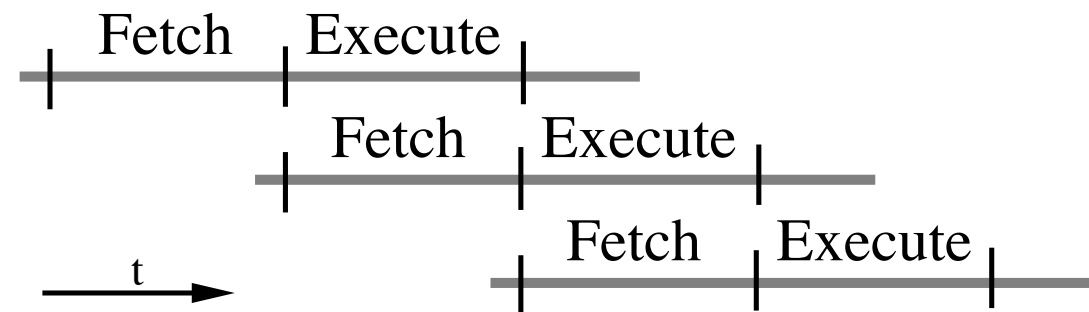


- Speicher
  - statisch
  - 32 Bit/Wort
- 1-Register
  - z.B. Inkrement
- Steuerwerkregister
  - Instruktionen
  - Instruktionsadresse (PC)
  - Status
- ALU
  - Eingang X,Y
  - Ergebnis Z
  - ADD, AND, XOR, OR
- Registertransfer
  - Quell-Register an Bus
  - Zielregister übernimmt
  - Steuerleitungen



## Ablauf der Instruktionen

- Zweiteilung der Befehle
  - Fetch-Zyklus holt Befehl
  - Execute Zyklus führt Befehl aus
  - eventuell überlappend



- Unterzyklen im Steuerwerk
  - Mikrozyklus, "Minor Cycle", Taktphase
  - andere Steuerimpulse in jedem Unterzyklus
  - Registertransferebene
- Mima-Instruktionen
  - 12 Unterzyklen
  - 5 Unterzyklen Fetch
  - 7 Unterzyklen Execute

- Ablauf der Lade-Instruktion

LDV     a   ; laden von Inhalt der Speicherzelle a in ACC  
              ; ACC <= Speicher[a]

**;Fetch-Zyklus**

1.            IAR -> (SAR, X), Leseimpuls an Speicher
2.            1 -> Y,        ALU auf Addition schalten,  
                                   Warten auf Speicher
3.                            Warten auf ALU, Warten auf Speicher
4.            Z -> IAR,        Warten auf Speicher
5.            SIR -> IR

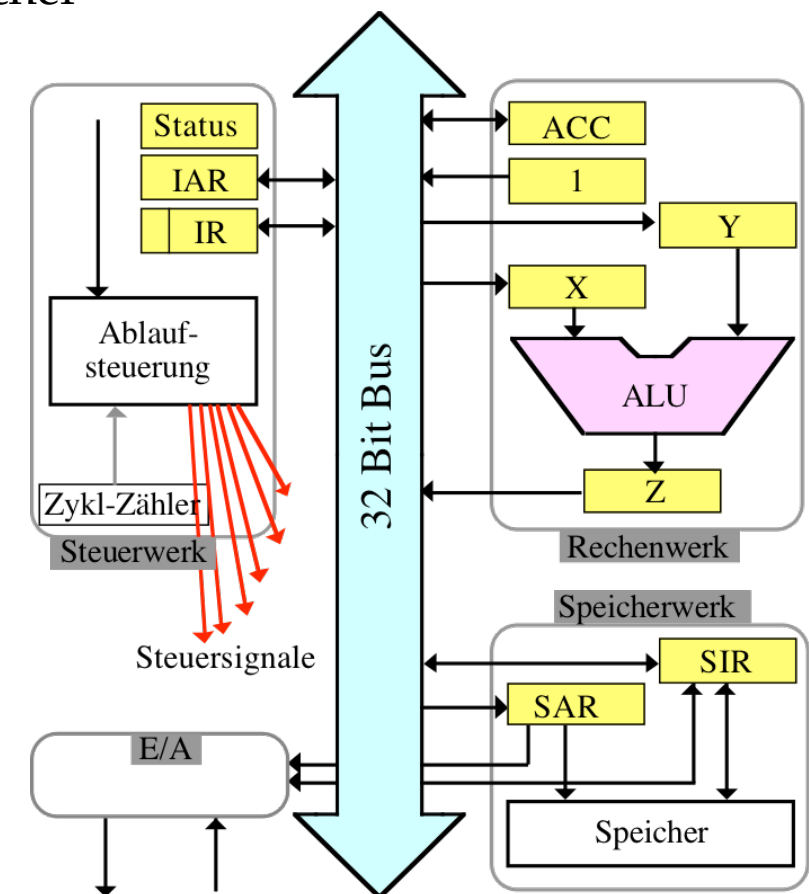
**;Fetch-Ende, jetzt Execute-Zyklus**

6.            IR -> SAR,        Leseimpuls an Speicher
- 7.,8.,9.                    Warten auf Speicher
10.           SIR -> ACC
- 11.,12.                    leere Unterzyklen

**;Execute-Ende, nächste Instruktion**

- IAR -> (SAR, X)

- IAR auf den Bus legen
- SAR und X mit Steuerimpuls vom Bus lesen lassen



- Ablauf der ADD-Instruktion

ADD a ; addieren von Inhalt der Speicherzelle a zum ACC

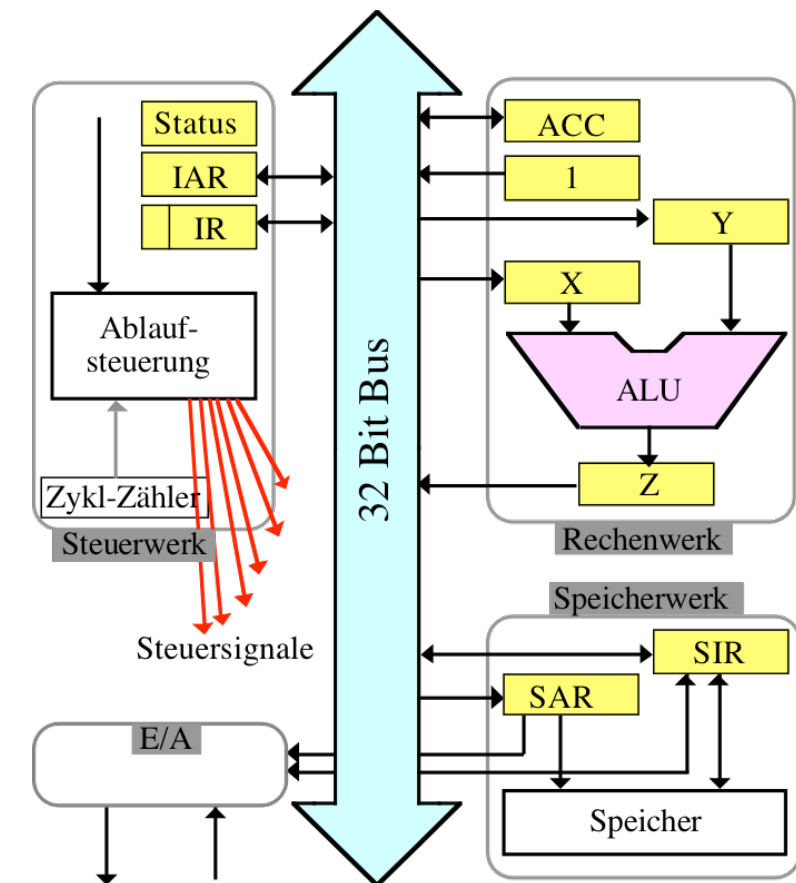
**;Fetch-Zyklus**

1. IAR -> (SAR, X), Leseimpuls an Speicher
2. 1 -> Y, ALU auf Addition schalten,  
Warten auf Speicher
3. Warten auf ALU, Warten auf Speicher
4. Z -> IAR, Warten auf Speicher
5. SIR -> IR

**;Fetch-Ende, jetzt Execute-Zyklus**

6. IR -> SAR, Leseimpuls an Speicher
7. ACC -> X, Warten auf Speicher
- 8., 9. Warten auf Speicher
10. SIR -> Y, ALU auf Addition schalten
11. warten auf ALU
12. Z -> ACC

**;Execute-Ende, nächste Instruktion**



- JMN p - Jump if Negative

- Bedingter Sprung zur Instruktion im Speicherwort[ p ]
- negativ bedeutet, das oberste Bit im Akkumulator ist 1
- X, Y-Register und ALU werden ignoriert
- kein Datenzugriff auf Speicher

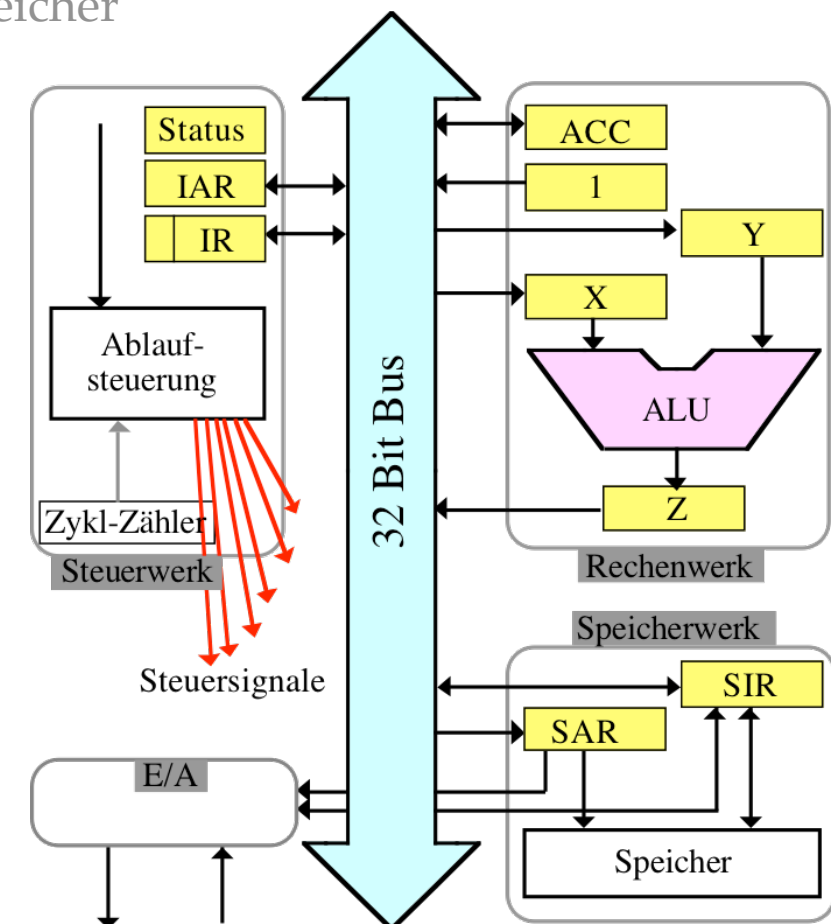
### ;Fetch-Zyklus

1. IAR -> (SAR, X), Leseimpuls an Speicher
2. 1 -> Y, ALU auf Addition schalten,  
Warten auf Speicher
3. Warten auf ALU, Warten auf Speicher
4. Z -> IAR, Warten auf Speicher
5. SIR -> IR

### ;Fetch-Ende, jetzt Execute-Zyklus

- 6a. falls vorderstes Bit in ACC = 1:  
IR -> IAR
  - 6b. falls vorderstes Bit in ACC = 0:  
leerer Unterzyklus
7. ... 12. leere Unterzyklen

### ;Execute-Ende, nächste Instruktion



- Start der Mima
  - Drücken der Reset-Taste
    - 0 -> IAR
    - 0 -> TRA
    - 1 -> RUN
- Maschine beginnt ab Adresse 0 Instruktionen auszuführen
  - evtl. andere, festgelegte Adresse
- Urlader
  - einige Speicherzellen ab Adresse 0 sind Festwertspeicher
  - enthalten einen Urlader ("Boot-Strap Loader")
  - liest Programm von einem bestimmten Gerät in den Speicher
  - beginnt mit dessen Ausführung
- Schaltpult mit Schaltern und Lämpchen zum
  - Modifizieren und Inspizieren von Registern und Speicherinhalten
  - bei älteren Rechnern



- Integration der E / A in den normalen Adressbereich

- memory mapped I / O
- In - LDV, Out - STV
- Geräte langsamer als Speicher
- variable Geschwindigkeit
- => Handshake

- Speicherwerk wertet Adresse aus

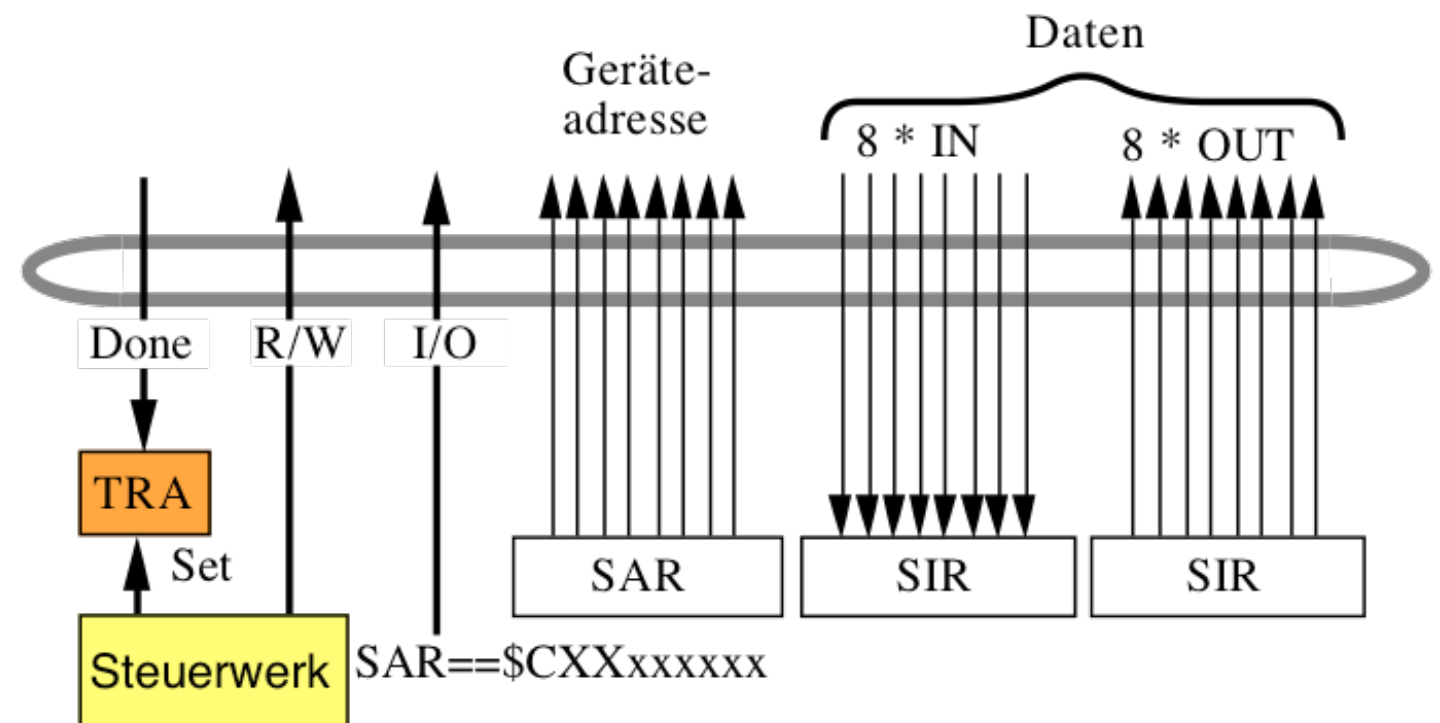
- $\text{Adr}_{[26..27]} == 11 \Rightarrow \text{E / A}$
- $\text{Adr}_{[26..27]} \neq 11 \Rightarrow \text{Speicher}$
- $3 \cdot 2^{26}$  Worte ( $3 \cdot 2^{28}$  Byte)

- TRA-Bit

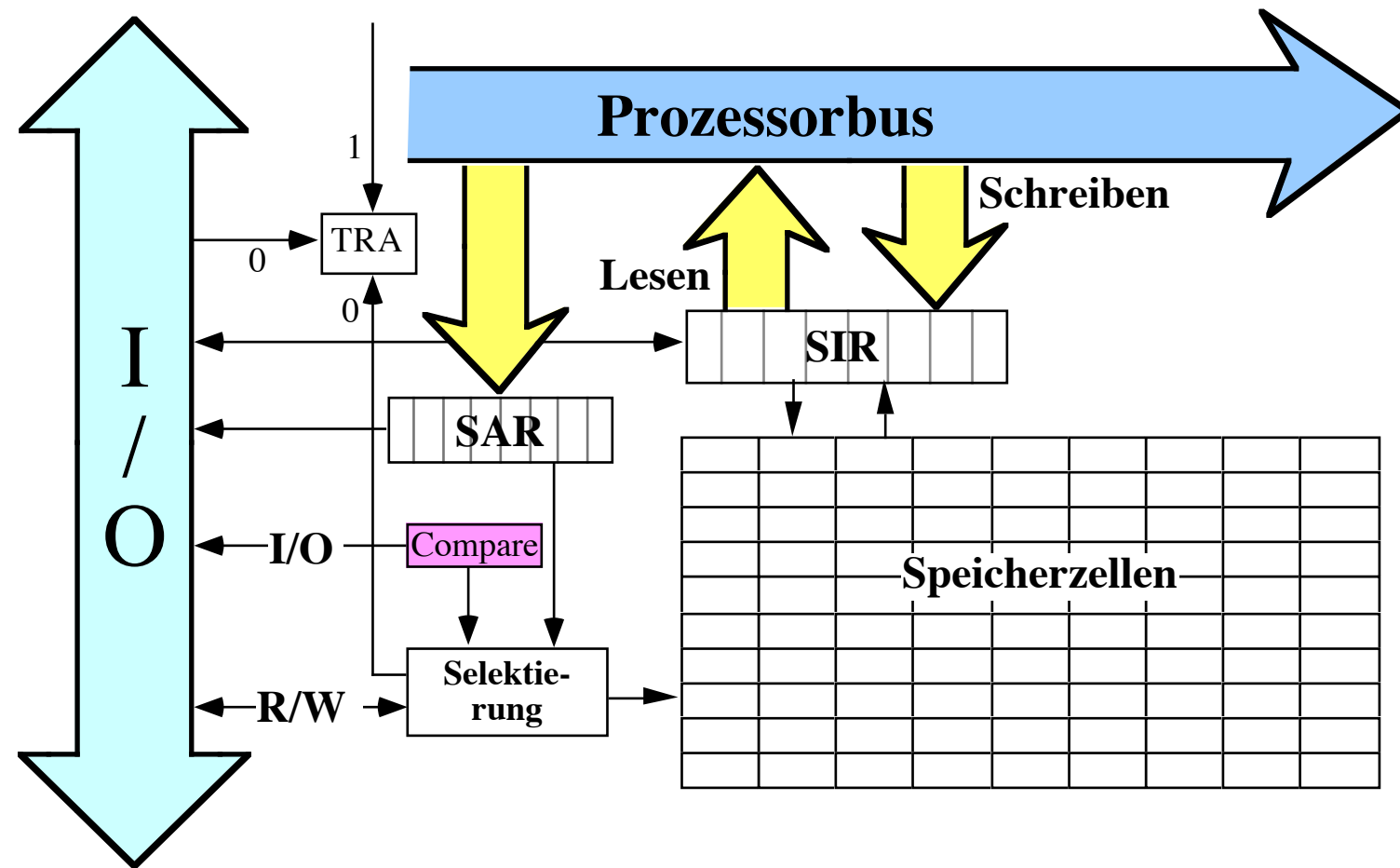
- vom Steuerwerk gesetzt
- von Speichercontroller oder Peripherie zurückgesetzt
- Steuerwerk wartet auf  $\text{TRA}=0$

- Geräte beobachten die Geräteadresse im EA-Werk

- Als Bus ausgeführte E / A-Schnittstelle



- Speicher mit Memory-mapped I/O
  - TRA Bit wird vom Prozessor gesetzt
  - von der Selektionsschaltung zurückgesetzt
  - oder vom Gerätebus zurückgesetzt



SAR = Speicheradressregister, SIR = Speicherinhaltsregister

## Erzeugung der Steuersignale

- Eingangssignale zum Steuerwerk:
  - Operationscode aus IR-Register (4 / 8 Bit)
  - Vorzeichenbit im Akkumulator
  - Mikrozyklus 1..12 aus externem Zähler
  - Status-Register = (RUN, TRA)

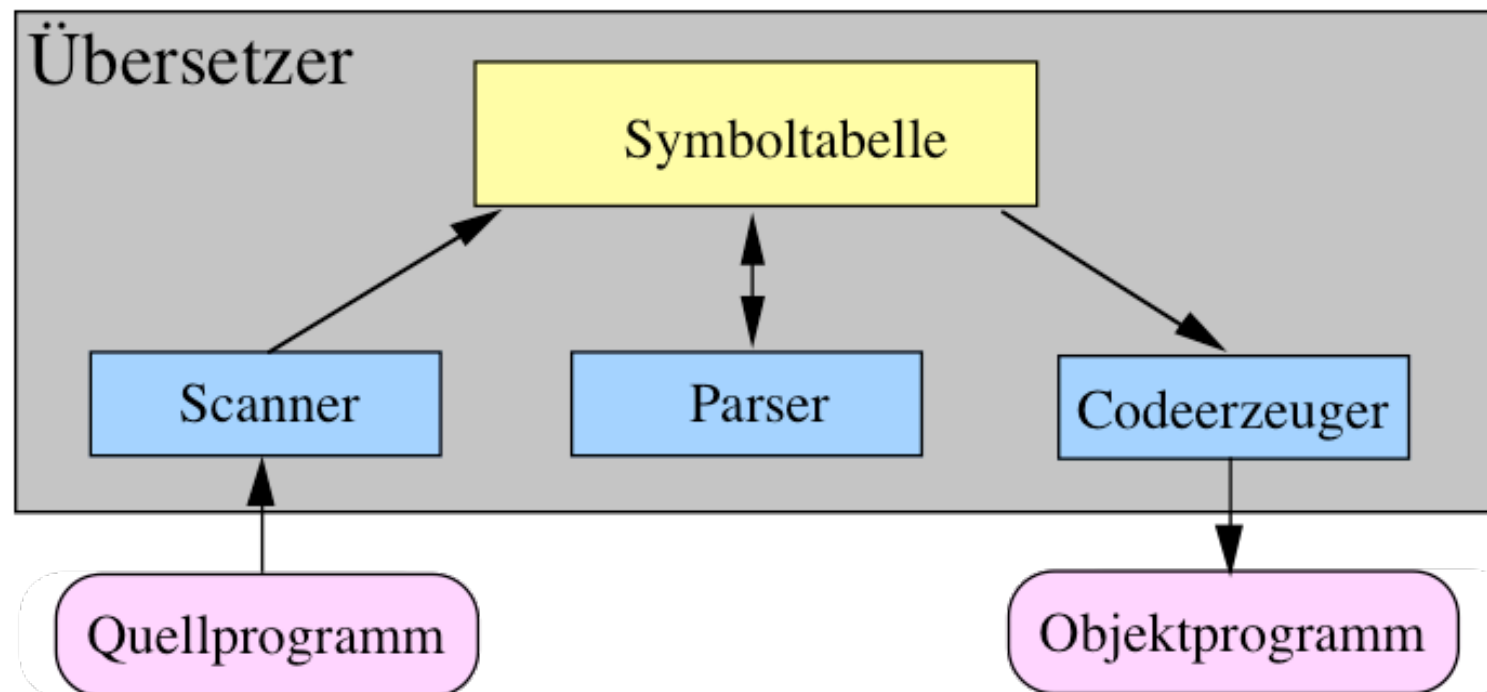
=>  $8+1+4+2 = 15$  Eingangsleitungen

- Ausgangssignale vom Steuerwerk
  - Rechteckige Steuer-Impulse
  - evtl. statische Steuerung (z.B. für ALU)
  - Takteingang für Flip-Flops
  - Out-Enable für Tristate-Ausgang am Bus

- RUN: 1 Leitung zum Clear-Eingang
  - TRA: 1 Leitung zum Preset-Eingang
  - ALU: 3 statische Pegel für die unterschiedlichen ALU-Funktionen
  - Speicher: 2 Leitungen für Read & Write
  - je 1 Input-Enable für Register
    - X, Y, ACC
    - IR, IAR, SIR, SAR
  - zusätzlich Output-Enable für Register
    - ACC, 1, Z
    - IAR, SIR, IR (nur Bit [0..27])
- =>  $1+1+3+2+7+6 = 20$  Steuersignale
- Steuerwerk kann realisiert werden:
    - als 20 Schaltfunktionen mit 15 Variablen
    - als ROM mit  $2^{15}$  Wörtern à 20 Bit

# Compiler

- Brücke Programmiersprache - Objektcode
  - C, Pascal, Modula, Fortran,
  - IA, 68000, PowerPC, 8051, Z80, DSPs, ...
  - Name => Adresse
  - Statement => Instruktionen
  - Prozeduraufruf => Sprung und Rücksprung



- Einlesen des Programmes (Scanner)
  - findet Symbole
  - Identifier, Konstanten, ...
- Syntaktische Analyse
  - zulässige Symbole werden verarbeitet ("Parsing")
  - für unzulässige Symbole Fehlermeldungen erzeugen
  - über "Look-Ahead" entschieden, welcher Pfad gewählt werden soll
  - bei schwierigen Programmiersprachen sehr weit vorausschauen
  - LL1 Programmiersprachen => maximal 1 Symbol Look-Ahead.
- Erzeugen der Maschinenbefehle (Codegenerierung)
  - syntaktische Prozeduren können auch die Instruktionen erzeugen
- Strategien
  - rekursive descent
  - bottom-up
  - top-down
  - Übersetzung für virtuelle Maschine besonders einfach
  - zeilenweise Übersetzung

- Beispiel: Ausdruck übersetzen

$a = b - (c - ((d * e) - g / h));$

LDV	g	
DIV	h	; g/h
STV	hilf	; optimiert wird später
LDV	d	
MUL	e	; (d*e)
SUB	hilf	; -
STV	hilf	
LDV	c	
SUB	hilf	
STV	hilf	
LDV	b	
SUB	hilf	
STV	a	; a= ...

## CHAPTER 7

# Computer, Internet und Sicherheit

- Attacken: Viren, Mithören, Fälschen, ...
- Werkzeuge
  - Codes, Verschlüsselung
  - Authentisierung, Signaturen
  - Zertifikate
  - Key exchange
- Dienste: chat, mail, https, ...
- Systemaspekt: Schloss ... Tür ... Haus; Schlüssel unter dem Blumentopf
- Inhalte und Metadaten
- Schneier, B.: Applied Cryptography; Indianapolis; 2015.
- Ferguson, Schneier, Kohno: Cryptography engineering; Indianapolis; 2010.

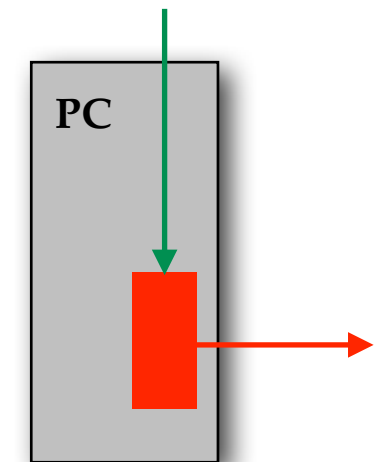
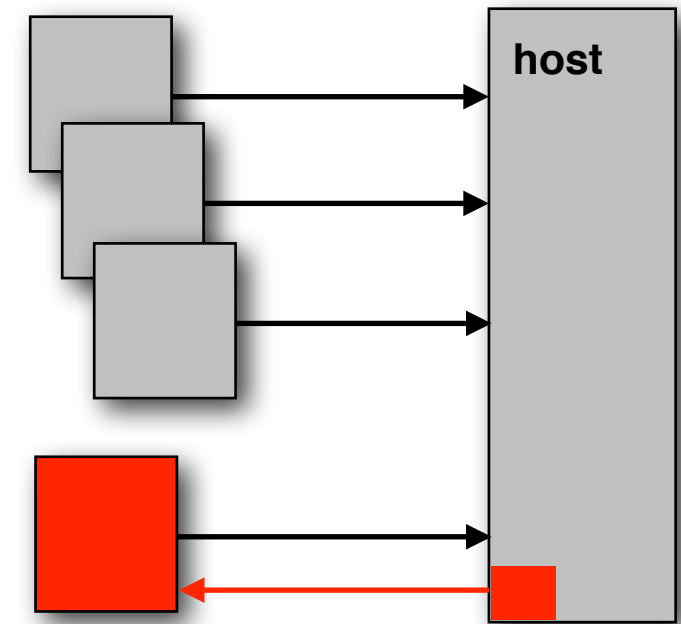
*Schneier's Law: Any person can invent a security system so clever that he or she can't imagine a way of breaking it.*



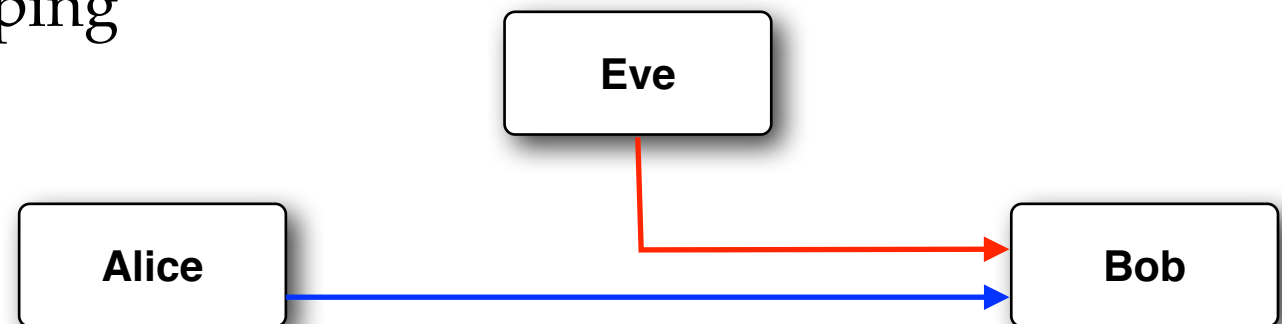
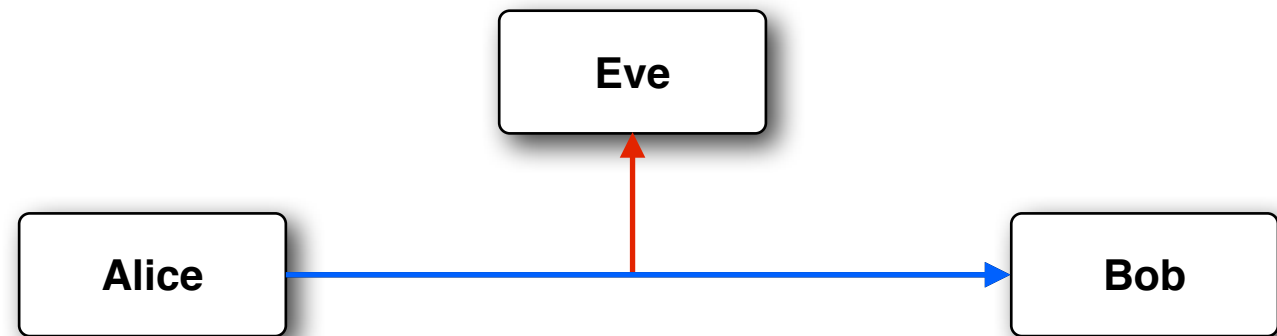
# Szenarien

- Angreifer
  - schlau und ruchlos
  - hohe Belohnung
  - Risiko kann minimiert werden
- Kommunikation: Attacken
  - abhören, verfälschen, fälschen
  - 'man in the middle'
  - Metadaten aufzeichnen -> social graph
- Computer: Eindringen, Infizieren
  - Viren, Trojaner, ...
  - Informationen stehlen, Erpressung
- Social engineering, z.B. Phishing, 419, ...

- Invasion
- Server: Einloggen mit erlangten Credentials
  - Tupel (Username, Passwort, ...)
  - erraten/erschliessen: (e-mail adr, leichte pwds)
  - mithören, man-in-the-middle
  - woanders stehlen:  $n * (\text{e-mail adr, std-pwd})$
  - social engineering
- Sever: Systemlücken
  - z.B. buffer overrun
  - Admin Rechte erlangen
  - eigene Software installieren
- PC & Telefon: Software installieren lassen
  - App download
  - Bilder, zip-archives, ...
- Drive-by Attacke
  - Webseite führt Code im Browser aus
  - e-mail mit Anhang
  - Erweiterungsmechanismen der Clients, helper-applications

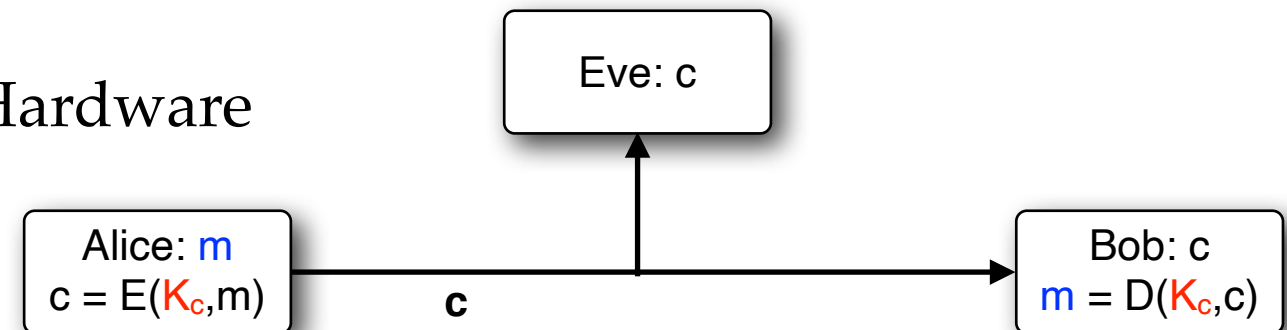


- Kommunikation attackieren
  - alte 'Kunst'
  - Caesar (ROT13), Enigma, ...
- Mitlesen
  - Codes, Codebücher, Keys
  - Brute-Force-Attacke vs. Dauer
  - Fehler im Algorithmus
- Stören
  - Kanalcodierung, z.B. Frequency Hopping
- Fälschen
  - Signaturen / Authentifizierung
  - ähnlich Verschlüsselung
- Man in the middle
  - Access Point
  - GSM-Basisstation (Stringray)



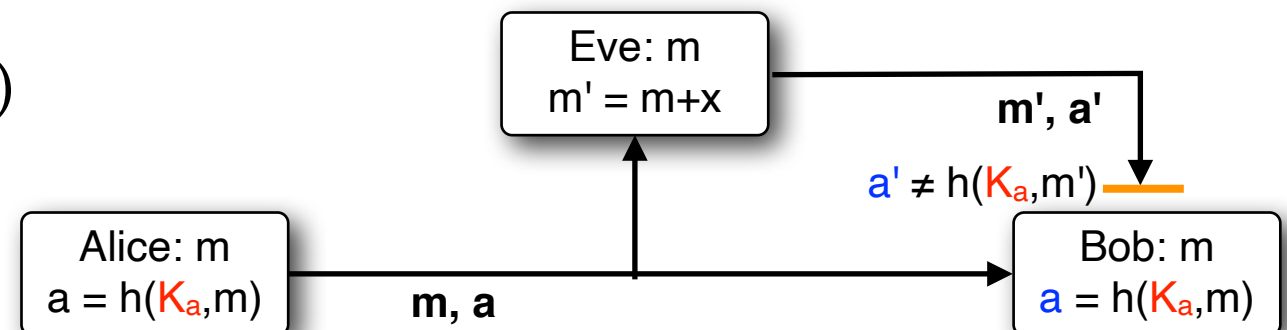
- Kryptographie

- Nachrichten transformieren: codieren
- Verschlüsseln und Entschlüsseln
- diskrete Mathematik, Algorithmen, Hardware
- geheimer Schlüssel (key)
- Problem Schlüsselverteilung



- Authentication von Nachrichten

- Echtheit der Nachricht verifizieren
- Nachricht unverschlüsselt
- Message Authentication Code (MAC)
- erzeugt mit Schlüssel



- Authentication mit Credentials

- (username, password)
- verschlüsselter Kanal
- Passwort als Hash im Host speichern
- alternativ: Frage -> verschlüsselte Antwort

- Komponenten

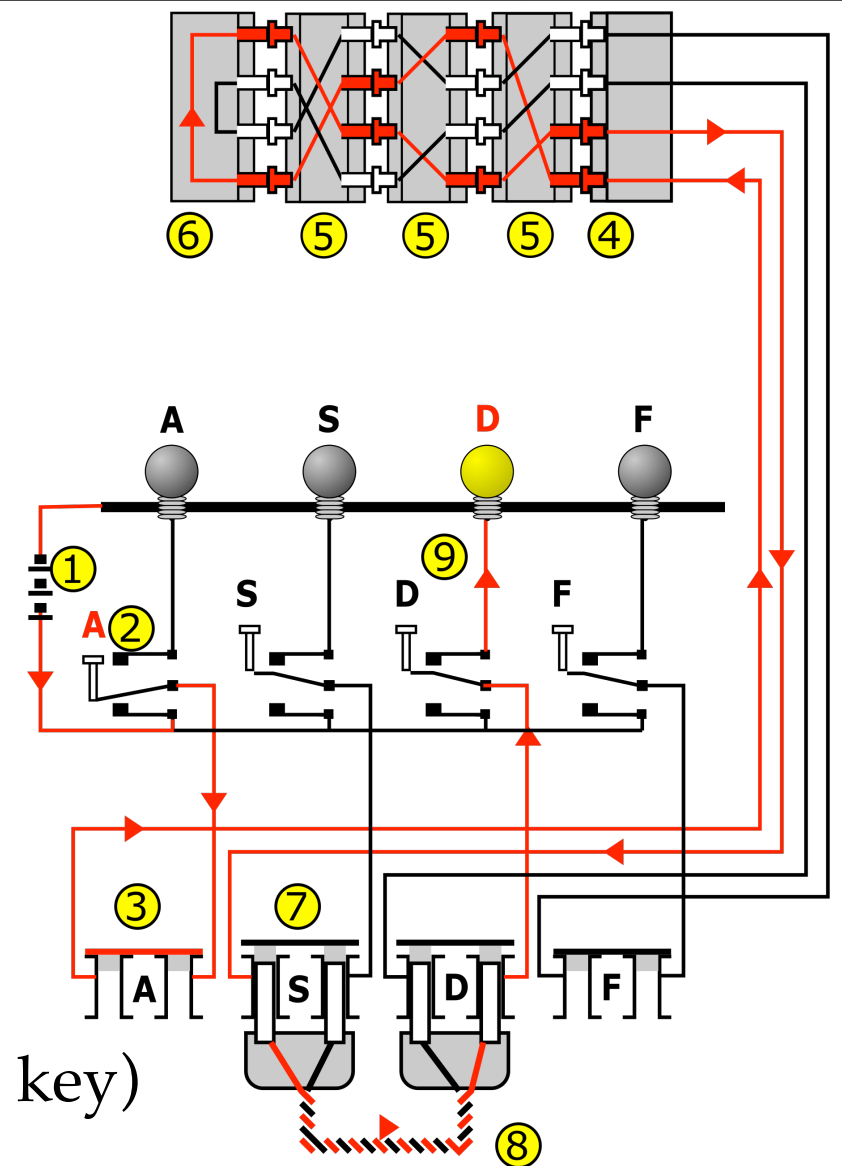
- Chiffre (cipher) und Schlüssel (key), Algorithmen, Schlüsselverteilung
- Einwegfunktionen (hash), Zufallszahlen (random)

- Brute force attack
  - Nachricht  $c(m)$ , key  $K_c$  unbekannt,  $\text{len}(K_c) = n$  Bits
  - Algorithmus bekannt
  - $2^n$  verschiedene Schlüssel  $K_x$ , ausprobieren +  $D(K_x, c)$  bewerten
  - Rechenzeit  $T = (t_{\text{decode}} + t_{\text{test}}) * 2^n$
- Geburtstagsparadoxon
  - Vorlesung: 23 Studierende
  - $p(\text{zwei mit gleichem Geburtstag}) > 0.5$
  - zwei Nachrichten mit dem gleichen Hash?

Bit	$10^8$ keys/sec	$10^{12}$ keys/sec
32	0,0000013619	0,000000000014
56	22,849313178	0,00228493132
128	$1,07903E+23$	$1,07903E+19$
256	$3,67174E+61$	$3,67174E+57$

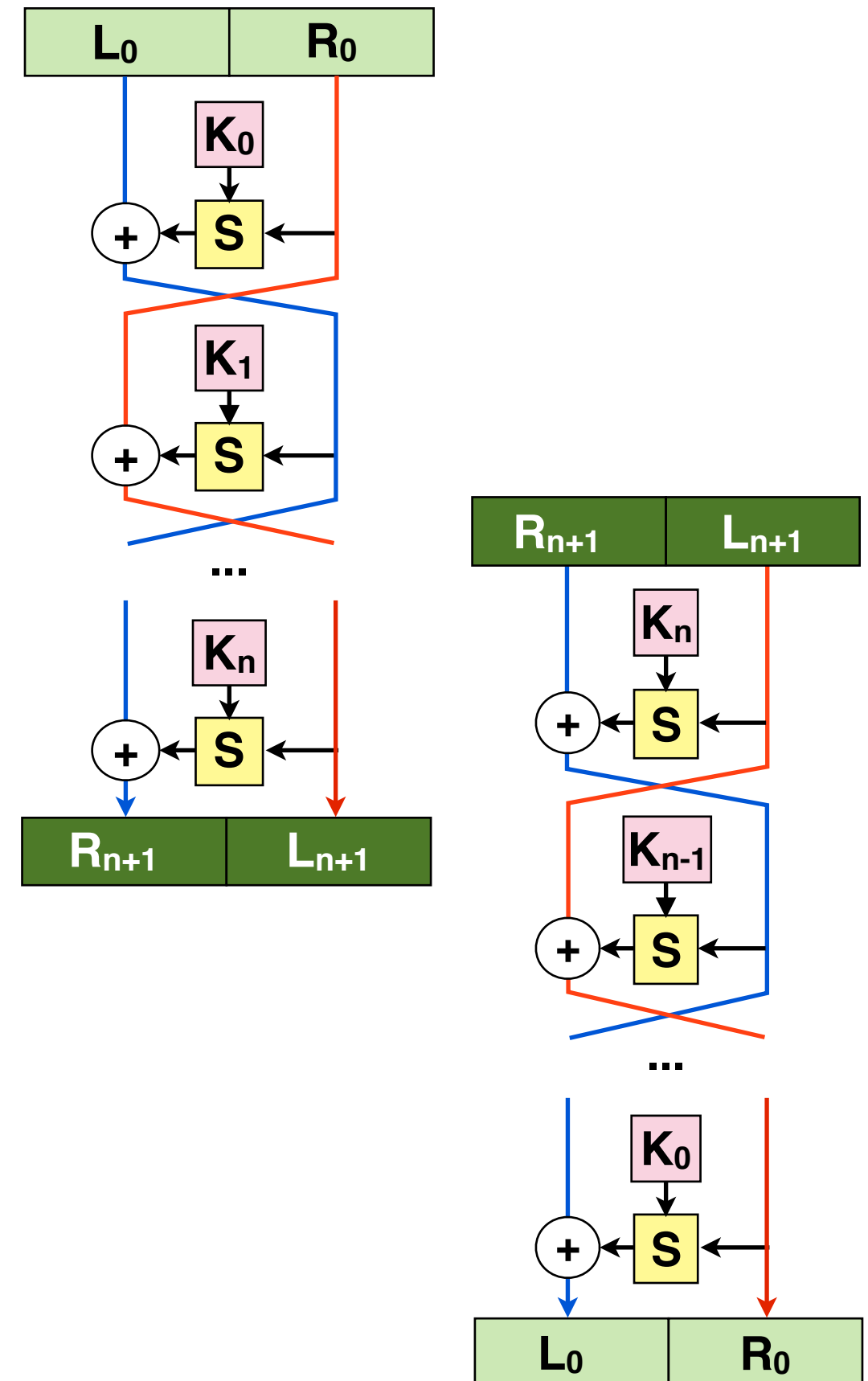
# Werkzeuge

- Ciphers
  - block
  - stream
- Hash (fingerprint)
- Zufallsgeneratoren, Einmalzahlen
- Schlüssel
  - symmetrisch
  - asymmetrisch: verschlüsseln  $\neq$  entschlüsseln (->public key)
- Monoalphabetische Verschlüsselung
  - Ersetzungstabelle  $C_{xmit,i} = code[C_{m,i}]$
  - Kryptoanalyse trivial: Häufigkeiten
  - klassische Version: Julius Caesar
  - bereits verschlüsselte Zeichen verändern Tabelle: Enigma



[de.wikipedia.org: MesserWoland]

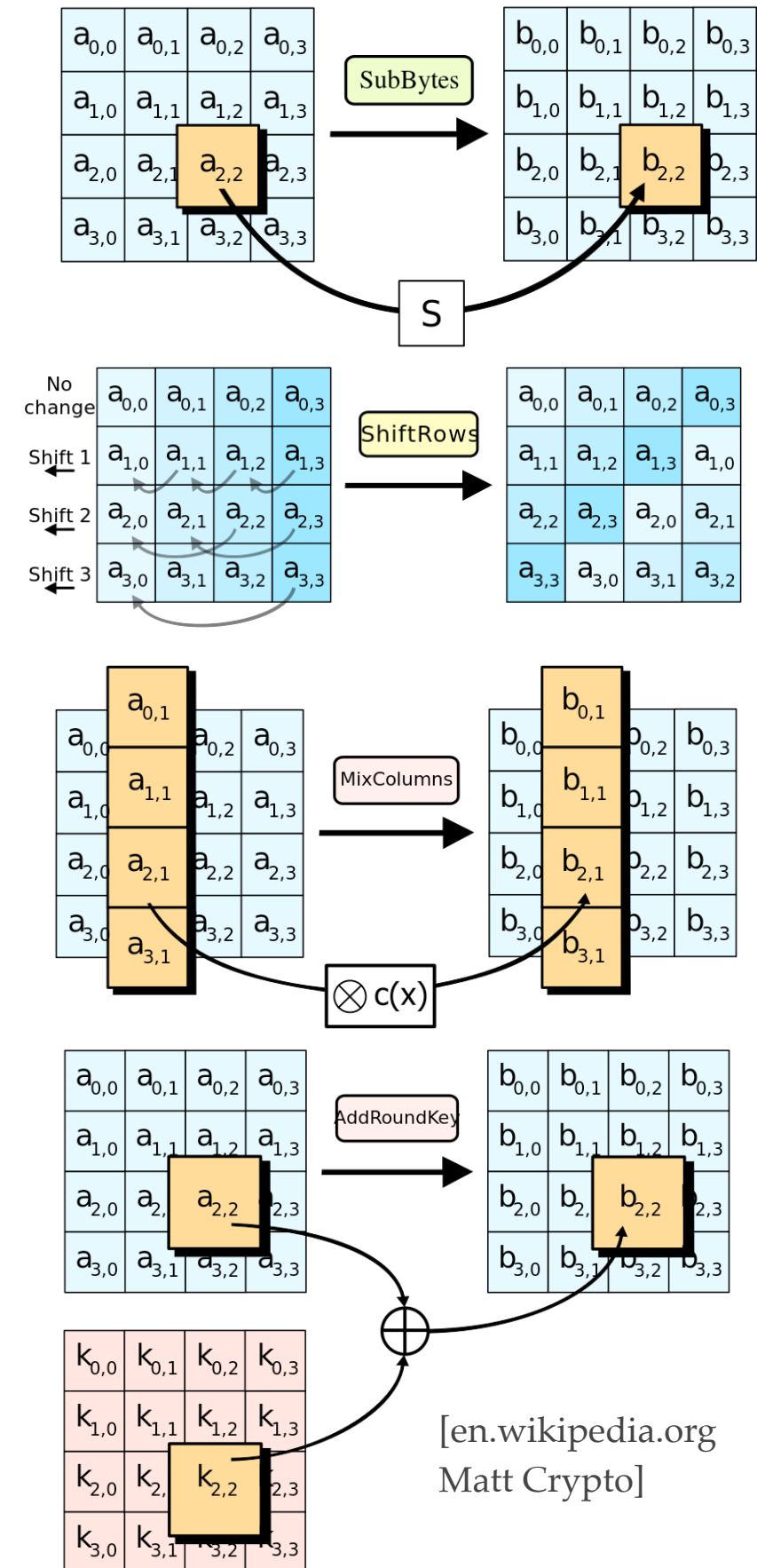
- Konfusion:
  - siehe Caesar, Enigma
  - Substitution (Buchstabenersetzung)
  - nichtlinear, lookup table
- Diffusion
  - Redundanz verteilen
  - z.B. Permutation der Bits
- XOR ( $\vee$ ,  $\wedge$ ,  $\oplus$ )
  - $a \oplus b = (a \vee b) \wedge \neg(a \wedge b)$
  - $a \oplus a = 0$
  - $a \oplus b \oplus a = b$
  - $c := a \oplus b: a = c \oplus b, b = c \oplus a$
- Feistelchiffre
  - Substitutions-Permutations-Netzwerk (SPN)
  - Bitkette (Block) in zwei Teile spalten
  - S-Box: Funktion  $f(\text{bits}, \text{key})$
  - Permutation:  $L_{i+1} = L_i \oplus S(R_i, K_i)$
  - n Runden
  - DES, Blowfish, Twofish, ...



- Block Cipher
  - Nachricht auffüllen auf  $n \cdot \text{Blockgrösse}$
  - $n$  Blöcke verschlüsseln mit SPN
  - Entschlüsselung mit inversem Schema und Key
- Cipher Block Chaining:  $C_i := E(K, P_i \oplus C_{i-1})$ 
  - Initialization Vector  $C_0$ : Random
- Stream Cipher, z.B. OFB, CTR
  - $K_i := \text{KeyStreamGenerator}(i)$ ;
  - $C_i := P_i \oplus K_i$



- Advanced Encryption Standard
  - AES, Rijndael, [NIST,2001]
  - symmetric keys 128, 192, 256 bit
  - 10, 12 bzw. 14 Runden
  - pro Runde
    - SubBytes
    - ShiftRows
    - MixColumns: Matrixmultiplikation über  $GF(2^8)$
    - AddRoundKey
  - Rundenschlüssel vom key ableiten (key expansion)



- Hash

- $H: M \rightarrow S$
- $|M| \gg |S|$
- $m \in M$  (unterschiedlich) lang
- $h \in S$  kurz, meist gleich kurz
- Kollisionen:  $m \neq m'$  und  $h(m) = h(m')$
- Prüfsummen, thumbnails, ...
- Keys in Datenbanken
- Kollisionen: Liste, double Hash, ...

```
char key[1000];
int hash = 42;
printf("string eingeben: ");
scanf("%s",&key);
for (int i=0; i< strlen(key);i++)
 {hash = (hash ^ key[i])*2; /* XOR */
 if (hash>255) {
 hash = (hash & 255)+1;}}
```



- Kryptografischer Hash (one way hash)
  - $H(m)$  leicht
  - Einwegfunktion: finde  $m$  mit  $h = H(m)$  schwer
  - kollisionsarm: finde  $m'$  mit  $H(m') = H(m)$  schwer
  - MD5 (Message Digest 5) [Rivest]
  - SHA (Secure Hash Algorithm) [NIST, NSA]
  - SHA-256, SHA-384, SHA-512
  - RIPE-MD: modifizierter MD4, 128, 256, 320 bit
- Message Authentication Code
  - auch verschlüsselte Nachricht  $m$  kann modifiziert werden
  - kryptographische Prüfsumme
  - $MAC(K, m)$  feste Länge,  $\text{len}(MAC) \ll \text{len}(m)$
  - $(m, MAC)$  senden
  - $K_m \neq K_{MAC}$

- RNG

- Computer deterministisch
- z.B. Kongruenzgenerator
- Startwert  $P_0$
- pseudo-random

$$P_{n+1} = (aP_n + b) \bmod m$$

- random() und andere

- Startwert  $P_0$  setzen mit `srandom(int seed)`
- aus physikalischem Prozess besorgen
- Uhrzeit
- Timer-Jitter
- Rauschen (CCD, Antenne, andere Sensoren)
- Mausbewegung, ...

- nonce: number only used once

- gegen Replay-Attacken
- Initialisierungs-Vektor
- (pseudo)random+timestamp

```
var start = get_usec()
sleep(100)
srandom(get_usec()-start)
let rnd = random()
```

- Problem Schlüsselverteilung

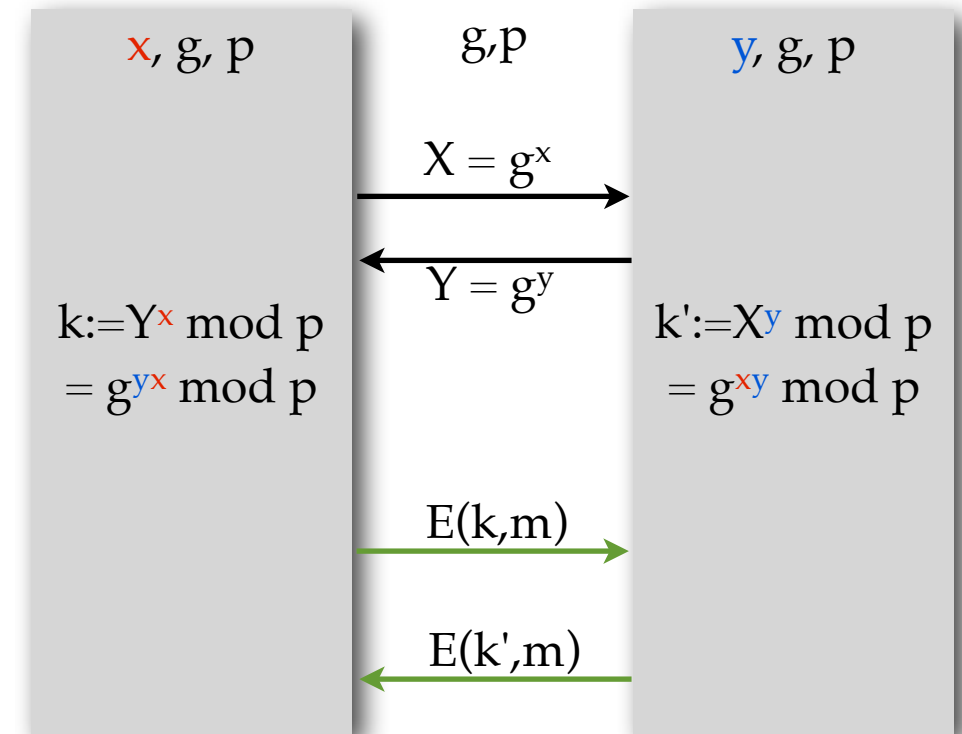
- shared *secret* keys
- out-of-band Verteilung der Schlüssel
- $1:n \Rightarrow n$  Schlüssel
- $n:n \Rightarrow n*(n-1)/2$  Schlüssel

- Diffie-Hellman, 1976

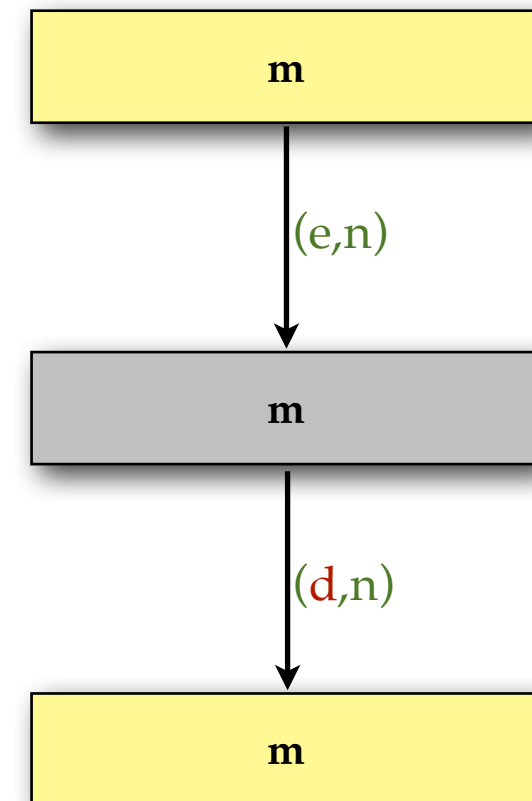
- zyklische Gruppe der Ordnung  $p$  erzeugt von  $g$
- Alice wählt Geheimnis  $x \in \{1, \dots, p-1\}$
- Bob wählt Geheimnis  $y \in \{1, \dots, p-1\}$
- Eve: finde  $a \mid A = g^a \bmod p$ ;  $g, p$  und  $A$  bekannt
- dafür braucht(?) man den diskreten Logarithmus, viel Rechenleistung
- sorgfältige Auswahl von  $g$  und  $p$
- $p$  groß, 4096 bit sicher bis ca. 2050

- Schlüssel viel länger als z.B. AES

- sicheren Kanal mit DH aufbauen
- Austausch von AES-Schlüsseln
- AES-Kanal benutzen für Bulk-Daten

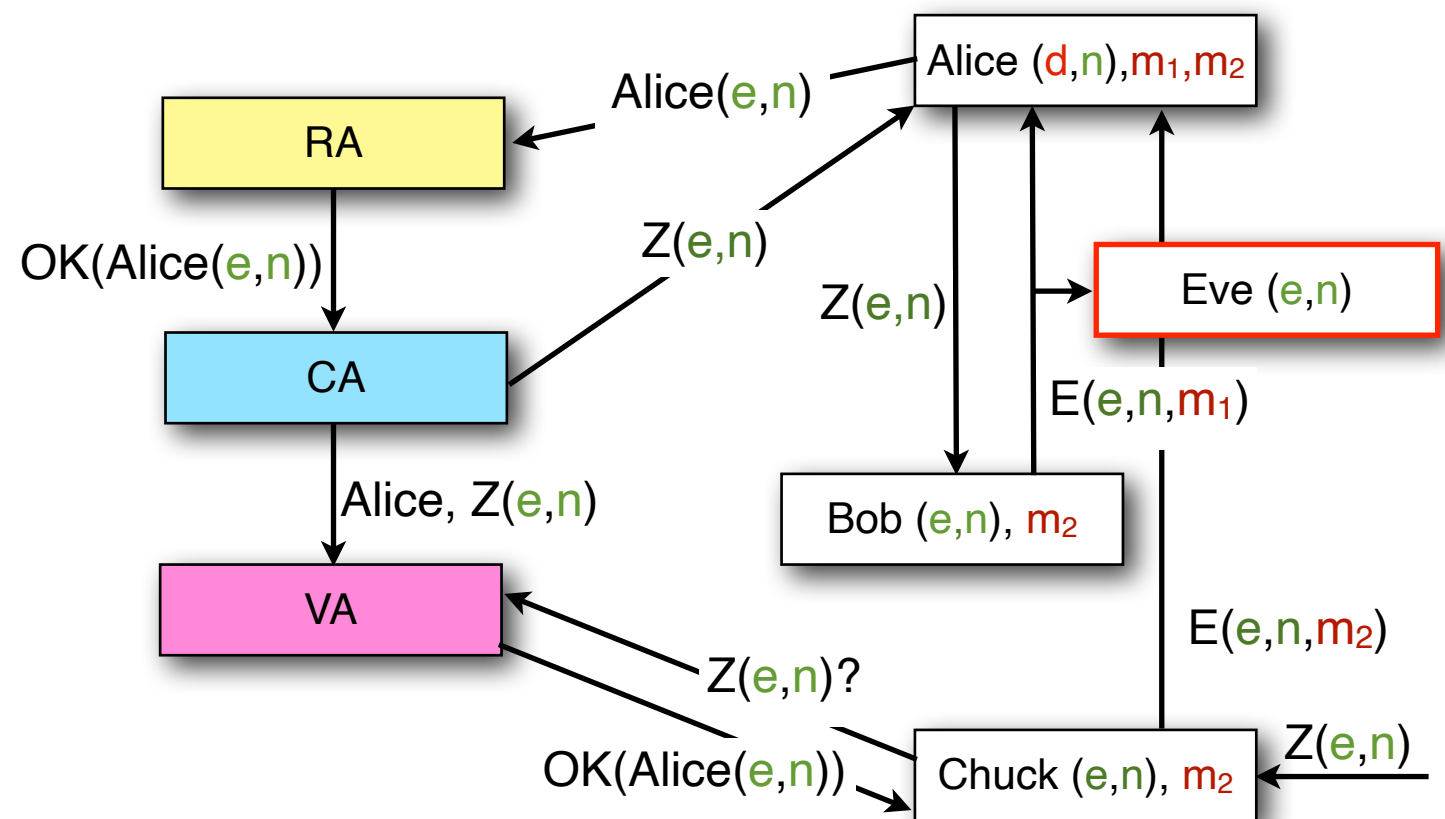


- Faktorisieren großer Zahlen
  - $n$  gegeben; finde  $p_1 \dots p_m \mid n = p_1 * \dots * p_m$ .
  - bzw:  $n$  gegeben; finde  $p \mid n = p * m, p \neq n, p \neq 1, m \in \mathbb{Z}$
  - Problem ist in NP
  - $n = p * q, p$  und  $q$  prim
- Rivest, Shamir, Adleman (RSA), 1978
  - asymmetrisches Verfahren: public key, private key
  - $n = p * q; n$  2000 - 4000 bit
  - wähle Schlüssel  $e \mid e$  und  $(p-1)*(q-1)$  teilerfremd
  - berechne Schlüssel  $d$  mit  $e * d = 1 \bmod (p-1)(q-1)$
  - public key:  $(e, n)$ , private key  $d$
  - $c_i := m_i^e \bmod n$
  - $c_i^d = m_i^{ed} = m_i^{k*(p-1)(q-1)+1} = m_i m_i^{k*(p-1)(q-1)} = m_i \bmod n$
  - $m_i^{k*(p-1)(q-1)} = 1 \bmod n$  Beweis mit kleinem Fermat  $a^{p-1} = 1 \bmod p$
- Kryptoanalyse
  - nur bei schlechter Wahl von  $p, q, e, d$  sind Angriffe bekannt
  - $d$  hinreichend groß wählen
  - Faktorisierung von  $n$  bei 4096 und 8192 bit noch schwierig
  - Konzepte für Angriff mit Quantencomputern





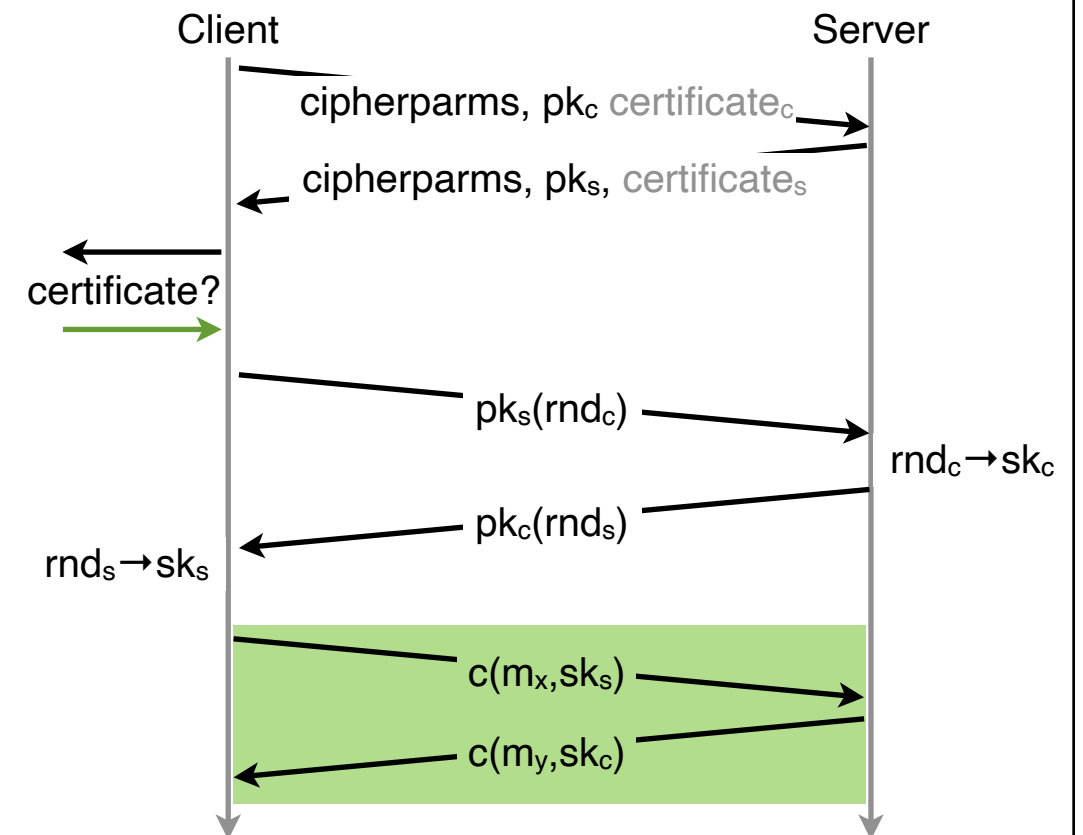
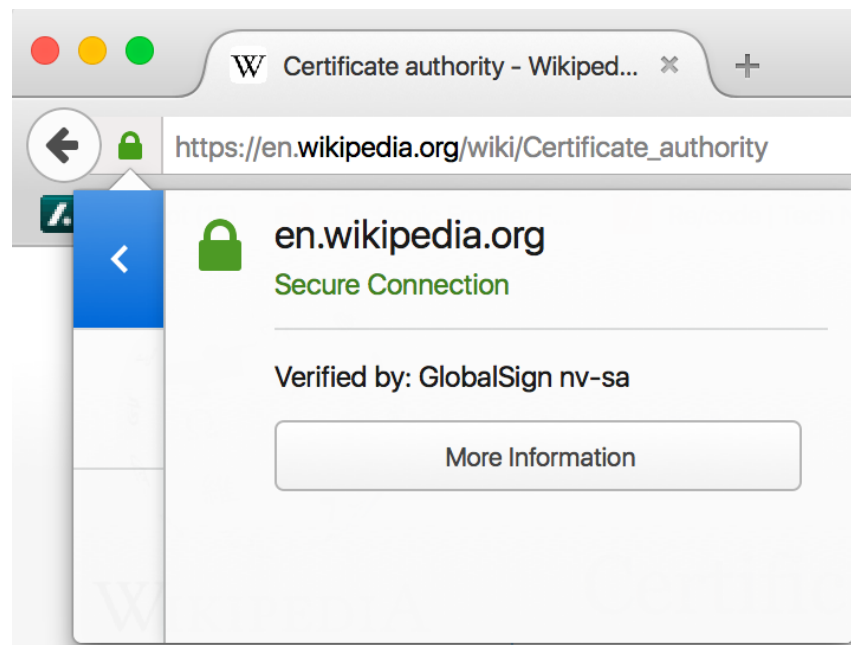
- Public Key Infrastructure (PKI)
  - Empfänger (z.B. Server) bietet public key an
  - ist das wirklich der public key des Empfängers?
  - Zertifikate ausstellen, verteilen und überprüfen
  - Zertifikate von Webservern, ...
- Registration Authority (RA)
  - erhält public key
  - prüft Identität
  - trusted
- Certification Authority (CA)
  - erhält (Identität, public key)
  - erstellt Zertifikat
  - verteilt Zertifikate
  - trusted
- Verification Authority (VA)
  - überprüft Zertifikat
  - erhält Zertifikate von CA
  - trusted



- Zertifikat-Kette
  - Verweis auf anderes Zertifikat
  - root certificate: Anker der Zertifikat-Kette
- Certifikate Authorities
  - gut bekannt => Vertrauen
  - kommerzielle, Behörden, usw.
  - z.B. im Browser eingetragen
  - Nachfrage beim Benutzer bei 'unbekannten' CAs ...
  - CA kann kompromittiert werden
- Let's Encrypt
  - OpenSource und kostenlos
  - automatischer Prozess
- Forward Secrecy
  - symmetric keys zu Beginn austauschen
  - RSA zum Schlüsselaustausch
  - Schlüsselaustausch+Nachrichten aufzeichnen
  - Langzeitschlüssel wird bekannt → alte Kommunikation entschlüsselbar
  - Abhilfe: ephemeral Diffie-Hellman zum Schlüsselaustausch

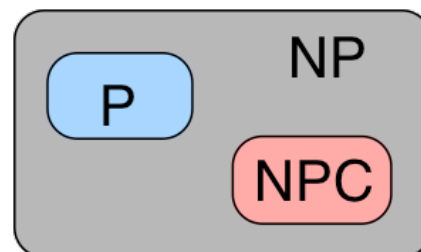
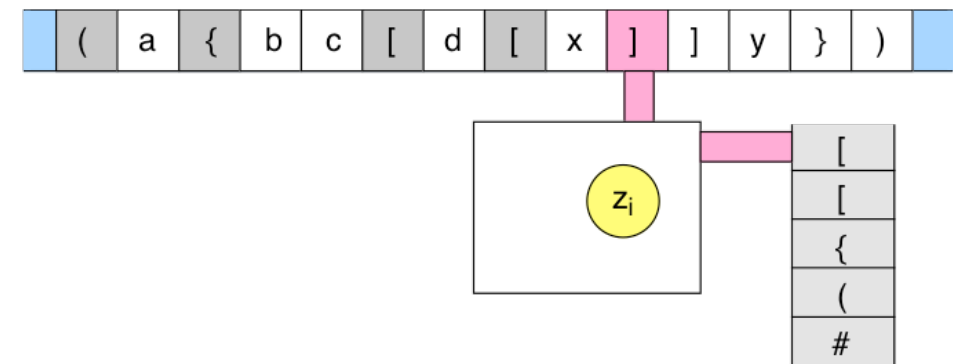


- Transport Layer Security TLS (RFC5246)
  - Basis: Secure Socket Layer (SSL)
  - symmetrische Verschlüsselung
  - Sessionkeys vereinbaren: public key (RSA)
  - oder ephemeral-DH für perfect secrecy
- https: HTTP mit SSL / TLS Verbindung
  - <https://en.wikipedia.org>



# Theoretische Informatik

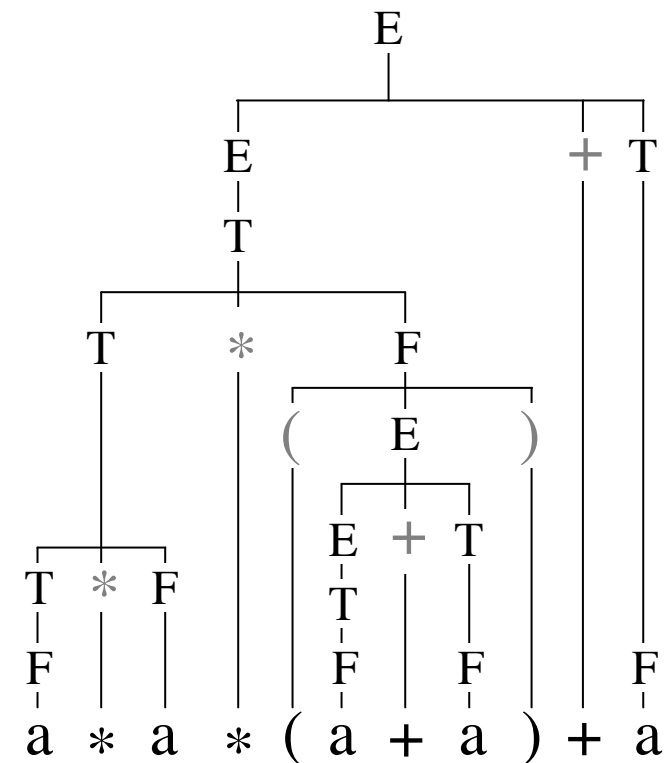
- Grundlagen
- Erkenntnisse zur Berechenbarkeit
- Formale Sprachen
- Automaten
- Notation



# Automaten und formale Sprachen

- Formale Sprache
  - Alphabet  $\Sigma$
  - $\Sigma^*$  Menge aller Worte über  $\Sigma$
  - Formale Sprache ist Teilmenge von  $\Sigma^*$
- Beispiel EXPR: *korrekt* geklammerte arithmetische Ausdrücke
  - $\Sigma = \{ (, ), +, -, *, /, a \}$  ; **a kann Variable oder Konstante sein**
  - $(a-a)*a+a/(a+a)-a \in \text{EXPR}$
  - $((((a+(a)))))) \in \text{EXPR}$
  - $((a+)-a( \notin \text{EXPR}$
  - wann ist ein Ausdruck  $w$  korrekt geklammert, also  $w \in \text{EXPR}$ ?
- Grammatik ist 4-Tupel  $G=(V, \Sigma, P, S)$ 
  - $V$  endliche Menge von **Variablen**
  - $\Sigma$  endliche Menge von **Symbolen** (Terminalsymbole)
  - $V \cap \Sigma = \emptyset$
  - **P Produktionen** (Regeln)
  - $S \in V$  ist die Startvariable

- Produktionen
  - $u, v \in (\mathbf{V} \cup \Sigma)^*$
  - Relation  $u \Rightarrow_G v$
  - $u = xyz, v = xy'z ; x, z \in (\mathbf{V} \cup \Sigma)^*$
  - $y \rightarrow y' \in \mathbf{P}$
- Sprache  $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$
- Ableitung von  $w_n$ 
  - Folge  $(w_0, w_1, w_2, \dots, w_n)$
  - $w_0 = S, w_n \in \Sigma^*$
  - $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$
  - nichtdeterministisch
- Ausdruck korrekt geklammert?
  - Ableitung existiert für Grammatik  $G$
  - $G = (\{\mathbf{E}, \mathbf{T}, \mathbf{F}\}, \{(\,), \mathbf{a}, +, *\}, \mathbf{P}, \mathbf{E})$
  - $\mathbf{P} = \{ \begin{array}{ll} \mathbf{E} \rightarrow \mathbf{T}, & \mathbf{E} \rightarrow \mathbf{E} + \mathbf{T}, \\ \mathbf{T} \rightarrow \mathbf{F}, & \mathbf{T} \rightarrow \mathbf{T} * \mathbf{F}, \\ \mathbf{F} \rightarrow \mathbf{a}, & \mathbf{F} \rightarrow (\mathbf{E}) \end{array} \}$
  - $a * a * (a + a) + a \in L(G) ?$



- Grammatik-Typen

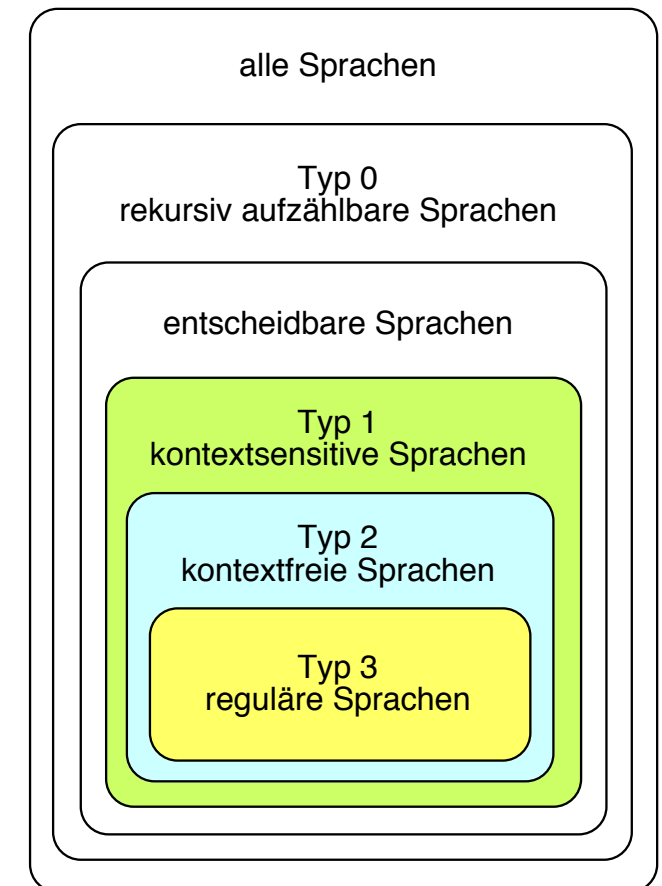
- Definition: jede Grammatik vom Typ 0
- Typ 1 kontextsensitiv:  $\forall w_1 \rightarrow w_2 \in P$  gilt  $|w_1| \leq |w_2|$
- Typ 2 kontextfrei:  $\forall w_1 \rightarrow w_2 \in P$  gilt  $w_1 \in V$
- Typ 3 regulär: Typ 2 und  $w_2 \in \Sigma \cup \Sigma V$   
( $w_2$  Terminalsymbol(+Variable))

- Chomsky-Hierarchie

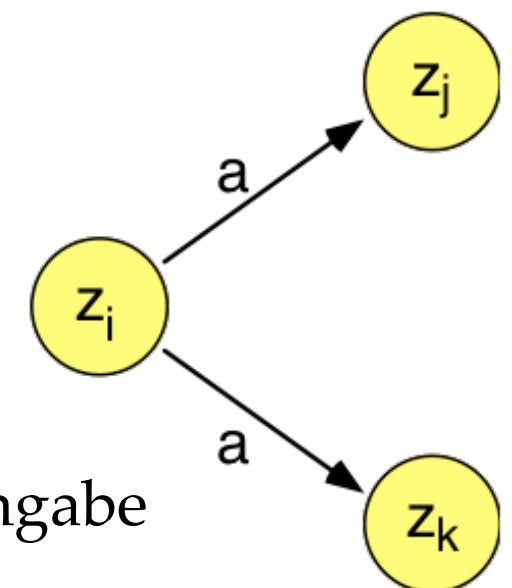
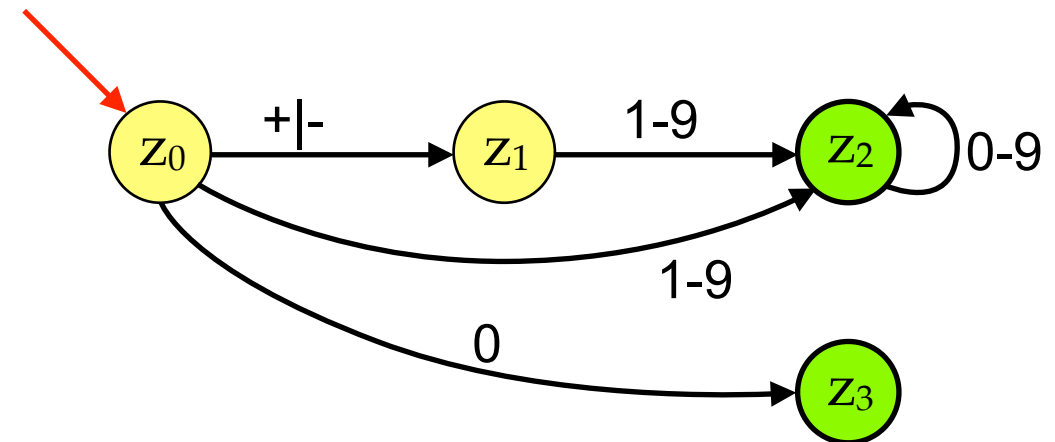
- Sprache L vom Typ x  
 $\exists$  Grammatik G vom Typ x mit  $L(G) = L$
- kontextsensitiv:  $uAv \rightarrow uxv$
- kontextfrei: auch ohne 'passenden' Kontext ersetzen
- Syntaxanalyse für Programmiersprachen
- Spezialklassen zwischen Typ 2 und 3:  $LL(k)$  und  $LR(k)$

- Entscheidbarkeit

- $\exists$  Algorithmus, der in endlicher Zeit feststellt, ob  $w \in L(G)$
- Typ 1,2,3 entscheidbar
- Es gibt Typ 0 Sprachen, die nicht entscheidbar sind



- Automaten
  - akzeptieren ein Wort
  - Menge aller akzeptierten Wörter: Sprache
- Deterministischer endlicher Automat (DFA)
  - $M = (Z, \Sigma, \delta, z_0, E)$
  - $Z$  endliche Menge der Zustände
  - $\Sigma$  endliches Eingabealphabet,  $Z \cap \Sigma = \emptyset$
  - Überföhrungsfunktion  $\delta: Z \times \Sigma \rightarrow Z$
  - $z_0 \in Z$  ist Startzustand,  $E \in Z$  Endzustände
- Darstellung typisch als Graph
  - gerichtet, beschriftet
  - Knoten sind Zustände
  - Pfeil auf Eingangsknoten
  - akzeptierende Endknoten als Doppelkreis
  - Kanten von  $z_1$  nach  $z_2$  mit  $a$  beschriftet:  $\delta(z_0, a) = z_1$
- Durch DFA erkennbare Sprachen sind regulär (Typ 3)
- Nichtdeterministischer endlicher Automat (NFA)
  - Zustandsübergang in verschiedene Folgezustände bei gleicher Eingabe



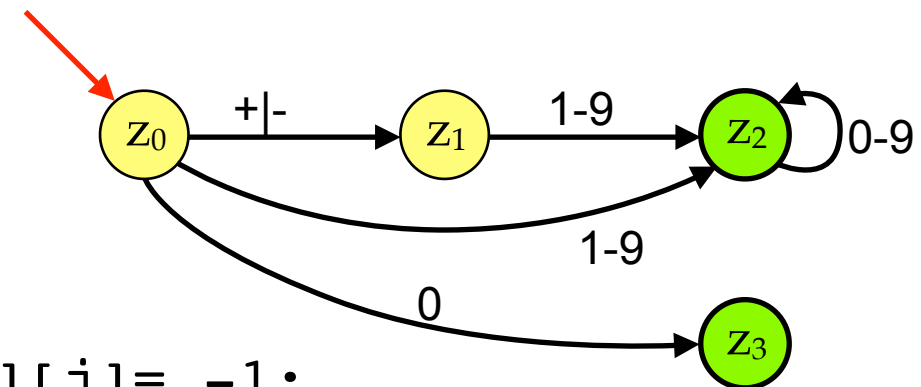
- Tabellenbasierte Implementierung von Automaten

z	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	'+'	'-'	...
0	3	2	2	2	2	2	2	2	2	2	1	1	-1
1	-1	2	2	2	2	2	2	2	2	2	-1	-1	-1
2	2	2	2	2	2	2	2	2	2	2	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

- Initialisierung der Tabelle

```
void tabinit(int tab[4][128])
{
 int i,j = 0;
 for (i=0; i< 4; i++)
 for (j=0; j< 128; j++) tab[i][j]= -1;
 for (i='0'; i<='9'; i++) tab[0][i] = 2;
 tab[0]['+']= 1; tab[0]['-']= 1;
 tab[0]['0'] = 3;
 for (i='1'; i<='9'; i++) tab[1][i] = 2;
 for (i='0'; i<='9'; i++) tab[2][i] = 2;}

```



```

#include <stdio.h>

/* function tabinit einsetzen */

int istEndzustand(int zustand)
 {return zustand ==2 || zustand ==3;}

int main(void)
{
 int zeichen, zustand = 0;
 int tabelle[4][128];

 tabinit (tabelle);

 while (zeichen=getchar()!='\n')
 if ((zustand = tabelle[zustand][zeichen])<0)
 break;

 if (istEndzustand(zustand))
 printf("akzeptiert\n");
 else printf("nicht akzeptiert\n");
 return 0;
}

```



- EBNF: erweiterte Backus-Naur-Form (ISO 14977)
  - kompakte Notation für kontextfrei Grammatiken
  - Regeln mit derselben linken Seite zusammenfassen

$$A \rightarrow \beta_1$$

$$A \rightarrow \beta_2$$

...

$$A \rightarrow \beta_n$$

wird zu:  $A ::= \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

- $A ::= \alpha[\beta]\gamma$  steht für

$$A ::= \alpha\gamma \mid \alpha\beta\gamma$$

$\beta$  kann einmal zwischen  $\alpha$  und  $\gamma$  eingefügt werden

- $A ::= \alpha\{\beta\}\gamma$  steht für

$$A ::= \alpha\gamma \mid \alpha\beta\gamma$$

$$B ::= \beta \mid \beta B$$

$\beta$  kann beliebig oft eingefügt werden

- manchmal Minimum und Maximum durch Index

$$A ::= \alpha\{\beta\}^4\gamma$$

Gruppierung mit ()

$$P = \{ \begin{array}{l} E \rightarrow T, \\ E \rightarrow E+T, \\ T \rightarrow F, \\ T \rightarrow T^*F, \\ F \rightarrow a, \\ F \rightarrow (E) \end{array} \}$$

$$P = \{ \begin{array}{l} E \rightarrow T \mid E+T, \\ T \rightarrow F \mid T^*F, \\ F \rightarrow a \mid (E) \end{array} \}$$

- Beispiel einfache Programmiersprache [Wikipedia]

```

program ::= 'PROGRAM' ,
 wsp , ident , wsp ,
 'BEGIN' , wsp ,
 {assignment, ";", wsp} ,
 'END.'

ident ::= alphachar , { alphachar | digit }
num ::= ["-"] , digit , { digit }
string ::= '"' , { anychar } , '"'
assignment ::= ident , " :=" , (num | ident | string)
alphachar ::= "A" | "B" | "C" | "D" | "E" | "F" | "G"
 | "H" | "I" | "J" | "K" | "L" | "M" | "N"
 | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
 | "V" | "W" | "X" | "Y" | "Z" ;
digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
wsp ::= ? ASCII Character 32 ?
anychar ::= ? all visible characters minus " ?

```

```

PROGRAM DEMO1
BEGIN
 A0:=3;
 B:=45;
 C:=A;
 D123:=B34A;
 HANS:=WURST;
 TEXTZEILE:="Moin Moin";
END.

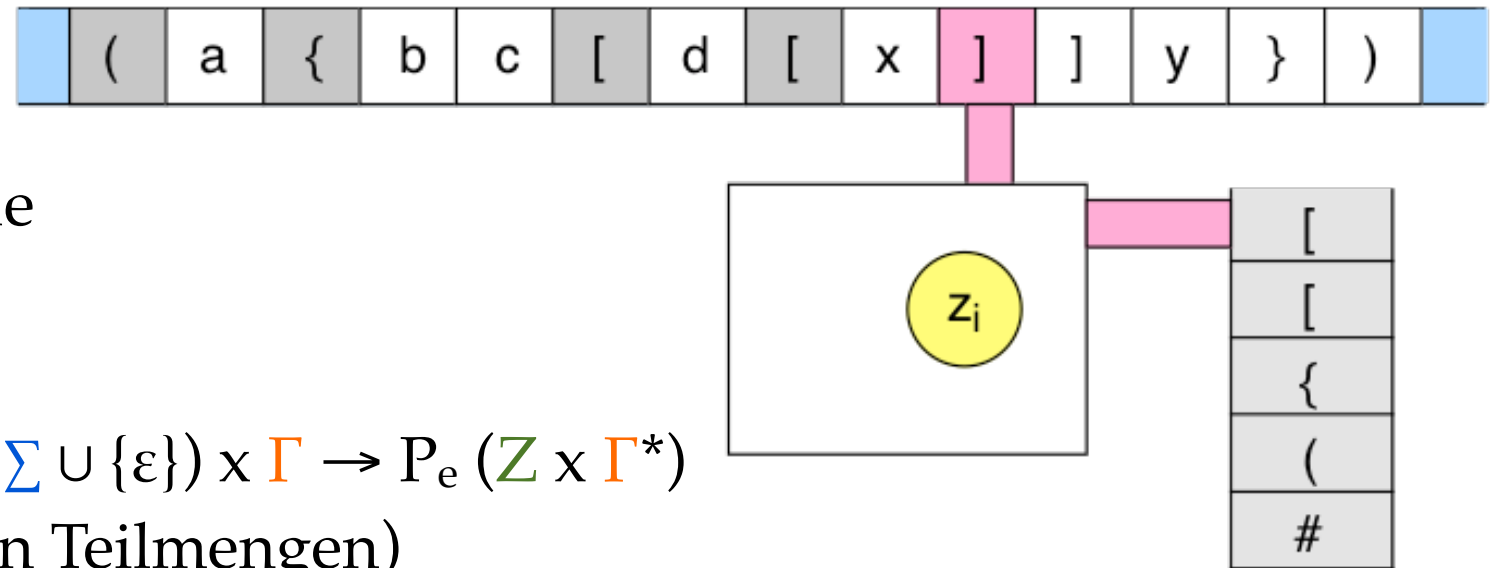
```

- Kontextfreie Sprachen (Typ 2)

- werden nicht durch endlichen Automaten akzeptiert
- Beispiel Klammerausdrücke: Zählen der Klammern
- Automat mit mehr 'Gedächtnis' als nur Zustand nötig

- Kellerautomat

- $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$
- $Z$  endliche Menge der Zustände
- $\Sigma$  Eingabealphabet
- $\Gamma$  Kelleralphabet
- Überföhrungsfunktion  $\delta : Z \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P_e(Z \times \Gamma^*)$   
( $P_e$  ist Menge aller endlichen Teilmengen)
- $z_0 \in Z$  ist Startzustand
- $\# \in \Gamma$  das unterste Kellerzeichen



- PDA - Push Down Automaton

- Keller ist Zusatzgedächtnis
- nichtdeterministisch
- akzeptiert genau die kontextfreien Sprachen
- können zum Beispiel korrekt geklammerte Ausdrücke erkennen

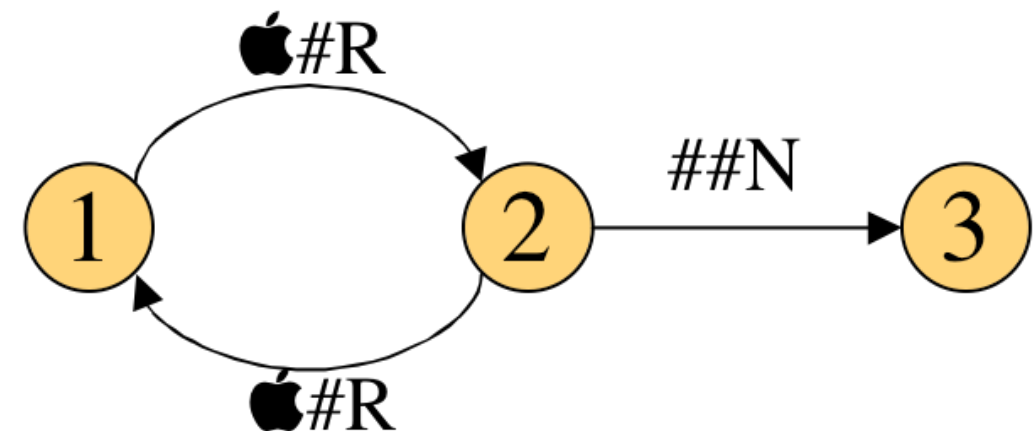
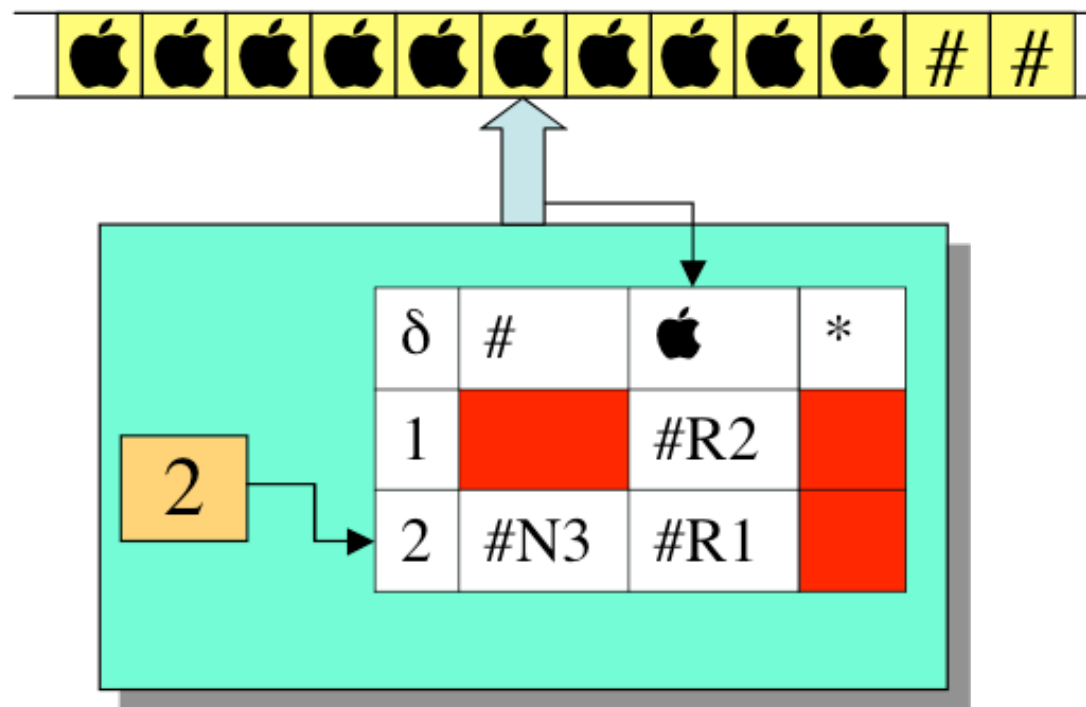
# Berechenbarkeit

- Intuitive Berechenbarkeit
  - formale Definition von 'intuitiv berechenbar' schwer

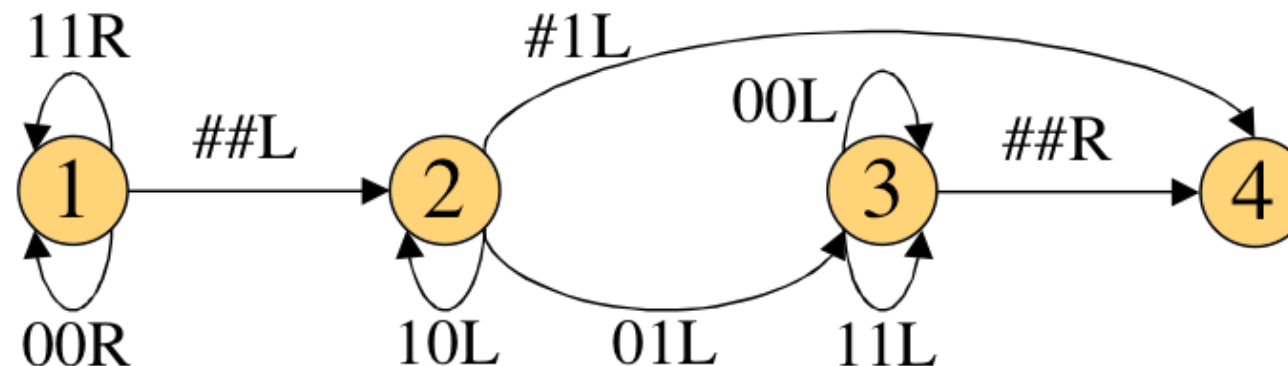
Eine Funktion  $f$  heisst berechenbar, falls ein Algorithmus (Programm) existiert, der (das) ausgehend von der Eingabe  $(x_1, \dots, x_n)$  in endlich vielen Schritte  $f(x_1, \dots, x_n)$  berechnet.

- Beispiele
  - $f(n) = 2 \cdot n$
  - $f_\pi(n) = 1$  falls  $n$  Anfang der Dezimalbruchentwicklung von  $\pi$  ist  
sonst 0
- Gibt es nicht berechenbare Funktionen?
  - ist für jede reelle Zahl  $r$   $f_r(n)$  berechenbar?
  - Nein: überabzählbar viele  $r$  (Cantorsches Diagonalschema ...)
  - nur abzählbar viele Algorithmen
  - oder das Cantorsche Abzählschema modifizieren

- Turingmaschine [[Alan Turing](#), 1936]
  - Automat
  - unbegrenztes Band mit Zeichen eines Alphabets
  - Schreib / Lesekopf
  - Bandbewegung ein Feld nach rechts oder links
  - Zustand, gelesenes Zeichen und Regel
  - Zustandsübergang, Zeichen schreiben, Band bewegen



- Formale Beschreibung
  - $Z$  endliche Menge von Zuständen
  - $\Gamma$  Arbeitsalphabet
  - $\Sigma \subset \Gamma$  Eingabealphabet
  - Überföhrungsfunktion  $\delta$ 
    - $\delta: Z \times \Gamma \rightarrow Z \times \Gamma \times \{L,R,N\}$  (deterministische TM)
    - $\delta: Z \times \Gamma \rightarrow P(Z \times \Gamma \times \{L,R,N\})$  (nicht-deterministische TM)
  - $z_0$  Startzustand
  - $\# \in \Gamma - \Sigma$ : Blank
  - $E \subseteq Z$  Menge der Endzustände
- Beispiel: Binäre Addition von 1
- Turing-Vollständigkeit
  - ein Computer, der alle mit Turing-Maschinen berechenbaren Funktionen berechnen kann (Bsp: zellulärer Automat Rule 110)



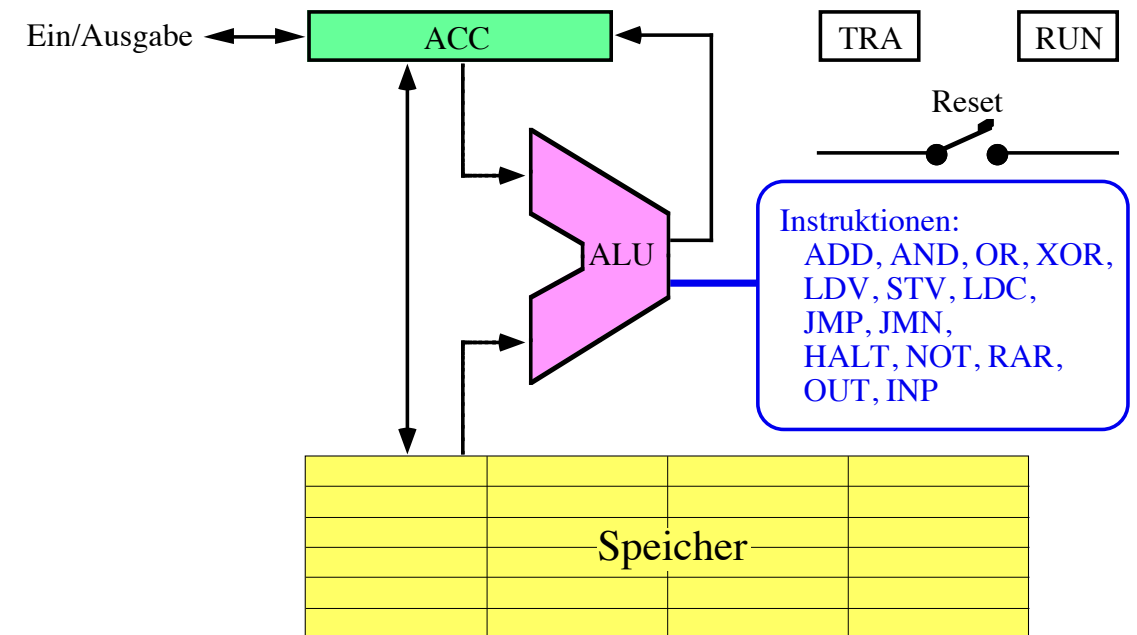
- Turingmaschinen akzeptierten Sprachen
  - $T(M) = \{x \in \Sigma^* \mid z_0x \rightarrow^* \alpha z \beta; \alpha, \beta \in \Gamma^*; z \in E\}$
  - allgemeine TM akzeptieren Typ 0 Sprachen
- Linear beschränkte Turing-Maschine: endliches Band
  - akzeptieren kontextsensitive Sprachen (Typ 1)
- Turing-Berechenbarkeit
  - $f : N \rightarrow N$  Turing-berechenbar falls  $\exists$  TM
  - $\forall n_1, \dots, n_k, m \in N$  gilt
  - $f(n_1, \dots, n_k) = m$  **genau dann wenn**  $z_0n_1bn_2b\dots n_kb \rightarrow^* \# \dots \# z_em_b\# \# \#$
- Churchsche These [Alonzo Church, 1936]

Die Klasse der Turing-berechenbaren Funktionen ist genau die Klasse der intuitiv berechenbaren Funktionen.

- eine These, von deren Richtigkeit fast alle Informatiker überzeugt sind
- wahrscheinlich, hilfreich, 'funktioniert' gut

- Registermaschinen

- RM oder auch RAM
- unendlich viele Registerzellen (~Speicher)
- Akkumulator und Einadressbefehle
- Load, Store, Arithmetik, Sprünge
- Adressierung: Konstante, direkt, indirekt
- ähnlich MIMA
- kann Turingmaschinen simulieren
- intuitiv: kann alle intuitiv berechenbaren Funktionen berechnen



- While Berechenbarkeit

- kann Turingmaschine simulieren und umgekehrt

```

Prog ::= id := ausdruck
 | IF bed THEN Prog ELSE Prog
 | WHILE bed DO Prog DONE
 | Prog; Prog

bed ::= ausdruck <= ausdruck
ausdruck ::= id | 0 | succ(ausdruck)

```

- LOOP Berechenbarkeit schwächer



- Primitiv rekursive Funktionen
  - konstante Funktionen sind primitiv rekursiv
  - Identität und Nachfolgerfunktion primitiv rekursiv
  - $f(0, \dots) = g(\dots)$
  - $f(n+1, \dots) = h(f(n, \dots), \dots)$
  - äquivalent zu LOOP-Berechenbarkeit
- Arithmetik als primitiv rekursive Funktionen
  - $\text{add}(0, x) = x$  /\* Identität \*/
  - $\text{add}(n+1, x) = \text{succ}(\text{add}(n, x))$
  - $\text{mult}(0, x) = 0$
  - $\text{mult}(n+1, x) = \text{add}(\text{mult}(n, x), x)$
- $\mu$ -Rekursion
  - $g(x_1, \dots, x_k) = \min\{n \mid f(n, x_1, \dots, x_k) = 0, m < n \text{ } f(m, x_1, \dots, x_k) \text{ definiert}\}$
  - terminiert evtl. nicht

Die Klasse der  $\mu$ -rekursiven Funktionen stimmt genau mit der Klasse der While-, RAM- und Turing-berechenbaren Funktionen überein.

- Ackermann-Funktion

- berechenbar, nicht primitiv rekursiv
- $\text{ack}(0,n) = n+1$
- $\text{ack}(m,0) = a(m-1,1) ; m>0$
- $\text{ack}(m,n) = \text{ack}(m-1,a(m,n-1)); m,n>0$

m/n	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	5	7	9	11
3	5	13	29	61	125
4	13	65533	$2^{65536}-3$	$a(3,(4,2))$	$a(3,a(4,3))$

- Halteproblem

- existiert ein Programm, das für jedes Paar von TM und Eingabe berechnet, ob die TM auf der Eingabe anhält?
- Gegenbeispiel von Turing, 1936
- semi-entscheidbar: TM hält entweder oder läuft endlos

Jedes Beweissystem für die Menge der wahren arithmetischen Formeln ist notwendigerweise unvollständig

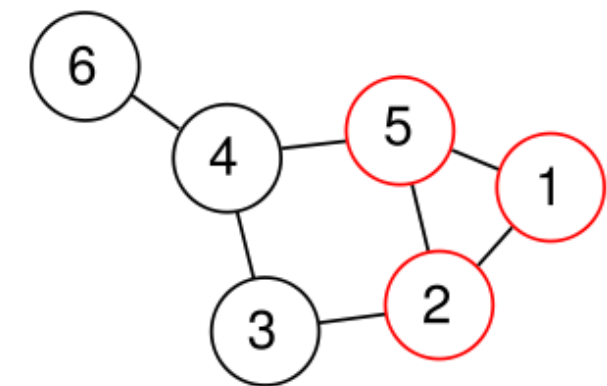
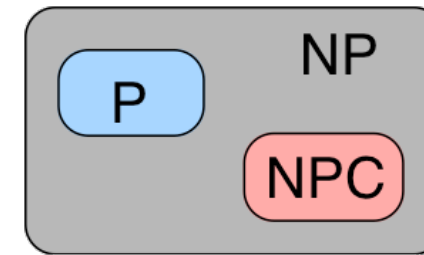
- Gödelscher Unvollständigkeitssatz [K. Gödel, 1931]

- es bleiben immer wahre arithmetische Formeln, die nicht beweisbar sind
- in jedem logischen System existieren Aussagen, die weder bewiesen noch widerlegt werden können

# Komplexitätstheorie

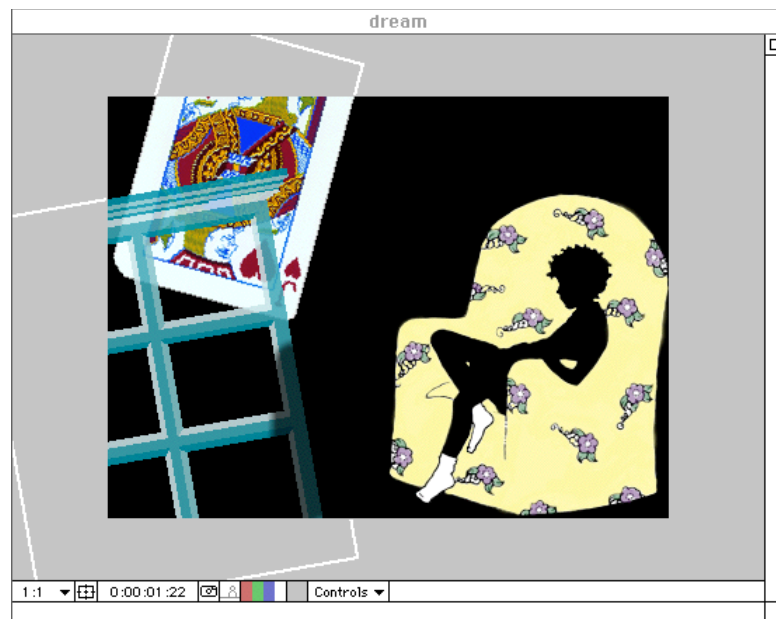
- Berechenbar ist nicht unbedingt 'praktisch' lösbar
  - Ressourcen-Verbrauch: Rechenzeit, Speicher
  - stark wachsende Probleme
- Problem der Klasse P
  - in polynomialer Zeit lösbar
  - Suchen in Listen mit  $n$  Elementen  $O(n)$
  - Sortieren  $O(n \log n)$
  - Sieb des Erathosthenes  $O(n^2)$
- SAT-Problem (Satisfiability)
  - gegeben boolesche Gleichung  $g$
  - SAT findet eine Variablenbesetzung  $x$ , so daß  $g(x) = \text{true}$
  - $O(2^n)$
- Klasse NP
  - nichtdeterministisch polynomielle Zeit
  - polynomial entscheidbar, ob Kandidat Lösung ist
  - exponentielle Komplexität
  - jedes Problem in NP lässt sich auf SAT-Problem polynomial zurückführen

- NP-vollständig (NPComplete)
  - p ist NPC, falls SAT auf p zurückführbar ist
  - alle 'gleich schwer'
  - Tausende NPC-Problem bekannt
  - $P=NP$ ???
- Cliques-Problem
  - Clique = Teilmenge von Knoten in Graphen paarweise verbunden
  - existiert k-Clique in einem Graphen?
- Hamilton-Problem
  - ein Weg der jeden Knoten genau einmal berührt?
  - Bsp: Springerproblem im Schach
- TSP - Travelling Salesman Problem
  - kürzester Hamilton-Kreis?
- Rucksack-Problem
- Anwendungen
  - Kryptographie, Optimierung
  - Stundenplanproblem, Verdrahtung in Chips und Platinen

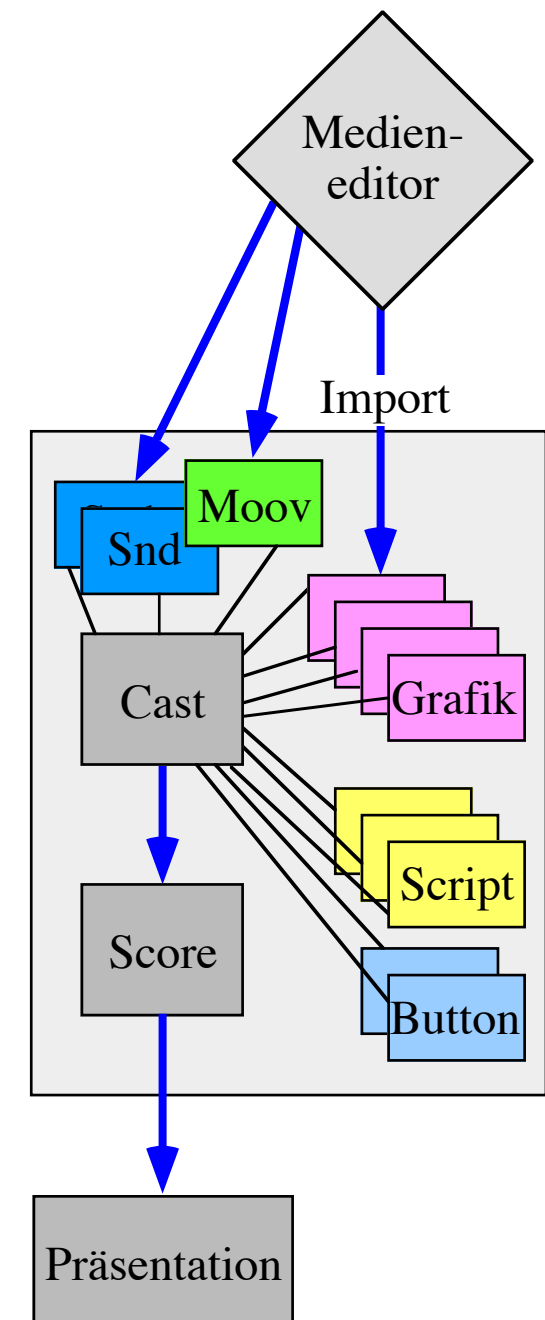


# Anwendungen

- |            |  |  |  |  |
|------------|--|--|--|--|
|            |  |  |  |  |
|            |  |  |  |  |
|            |  |  |  |  |
|            |  |  |  |  |
| =Y7+Y11-Z9 |  |  |  |  |
|            |  |  |  |  |
|            |  |  |  |  |
|            |  |  |  |  |



```
select Name
from Professor
where not exists
 (select *
 from Vorlesung
 where Leser=Persnr);
```

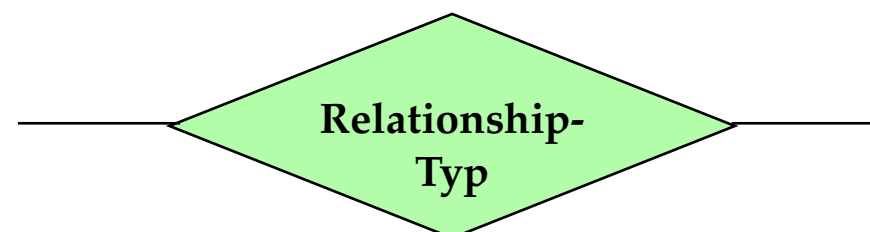
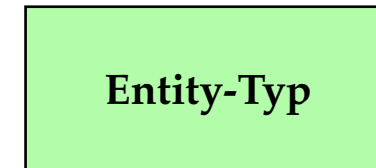


# Datenbanken

- Nach Oertel 1998, insbesondere Beispiele
- Datenbanksystem (DBS)
  - Datenbank (DB)
  - Datenbankmanagementsystem (DBMS)
- Datenbank
  - Datenbestände
  - Beschreibung
  - im Jargon: Hardware, Software und Daten
- Datenbankmanagementsystem
  - Softwaresystem
  - Verwaltung und Bereitstellung der Daten
- Datenbanksprache
  - Definitionssprache (DDL)
  - Manipulation und Abfrage (DML)

- Datenmodell
  - Diskursbereich (e.g. Lagerhaltung, Versandhandel, Personal, ...)
  - Abbildung Diskursbereich auf Datenbank
  - Strukturen, Operationen, Integritätsbedingungen
- Diskursbereich
  - anwendungsrelevanter Ausschnitt der Realität
  - Handelsunternehmen: Artikel, Lager, Kunden, Lieferanten, Konten ...
  - Universität: Studenten, Assistenten, Professoren, Lehrveranstaltungen, ...
- Entity-Relationship-Model (ERM)
  - Ableitung des Diskursbereiches
  - Gegebenheiten
  - Beziehungen
- Entwurfsziele ERM
  - schematisch, zeitunabhängiges Modell
  - Unterstützung der Datenbank-Entwurfsphasen
  - Visualisierung von Zusammenhängen
  - Kommunikation: Anwender, Feldingenieur, Entwickler
  - Abbildbarkeit in Datenmodelle (insbesondere in das Relationenmodell)

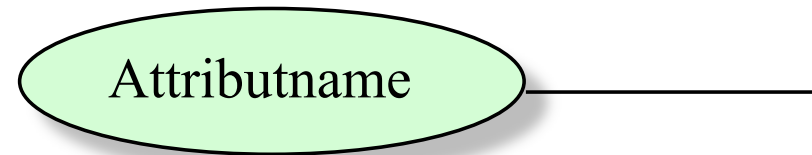
- Elemente des ERM und graphische Darstellung
- Entity
  - diskrete Abstraktion
  - Gegenstand der Anschauung oder des Denkens
- Entity-Typ
  - Verallgemeinerung (Klassifikation) gleichartiger Entities
  - graphisch: Rechteck mit Namen
  - im Beispiel: Student, Assistant, Professor, Lehrveranstaltung, Prüfung
- Relationships
  - zwischen je einer Entity von mehreren Entity-Typen
  - nicht notwendig verschiedene Entity Typen
- Relationship-Typ
  - Verallgemeinerung (Klassifikation) gleichartiger / ähnlicher Relationen
  - graphisch: Raute mit Namen
  - ungerichtete Kanten zu Entity-Typen
  - im Beispiel: lesen, hören, prüfen, voraussetzen, arbeiten\_für





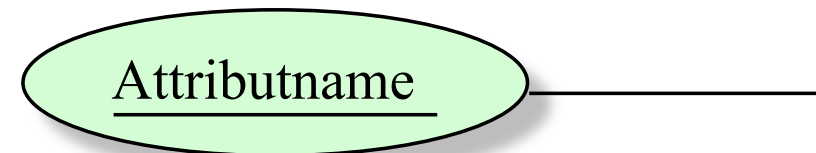
- Attribut

- Merkmal zur Beschreibung
- Entities und Relationships
- Name, Merkmalswertemenge (Domäne)
- graphisch: Kreis / Ellipse mit Attributnamen
- Kante zum Entity / Relationship-Typ
- im Beispiel: Namen, Vorname, Adresse, Fachgebiet, Note, ...



- Schlüssel

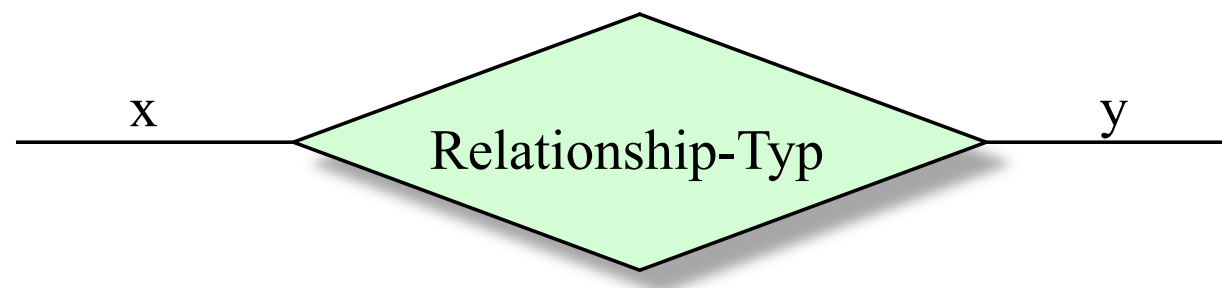
- Teilmenge von Attributen
- identifizieren von Entity / Relationshiptypen
- minimal bzgl. Identifikation
- graphisch: Kreis / Ellipse mit unterstrichenem Attributnamen
- Kante zum Entity / Relationship-Typ
- im Beispiel: Personalnummer, Matrikelnummer, ...



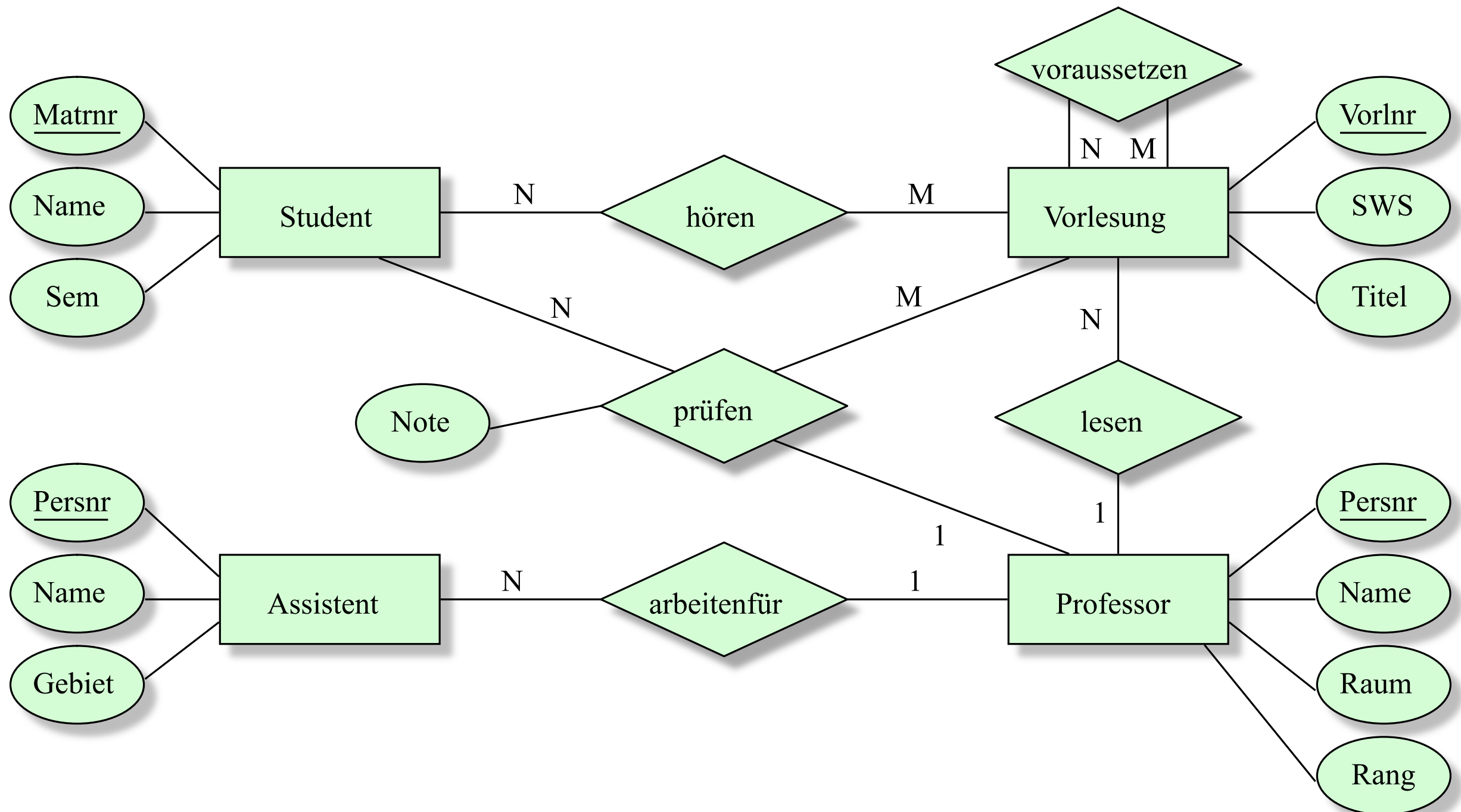
- Primärschlüssel

- pragmatisch ausgewählter Schlüssel

- Art des Relationship-Typs R zwischen Entity-Typen E1 und E2
  - 1:1 In R kann eine Entity aus E2 nur mit einer Entity aus E1 in Bez. stehen  
In R kann eine Entity aus E1 nur mit einer Entity aus E2 in Bez. stehen
  - 1:n In R kann eine Entity aus E2 nur mit einer Entity aus E1 in Bez. stehen
  - m:n mehrere Entities aus E1 können zu mehreren in E2 Bez. haben
- Zuordnungscharakteristik
  - Erweiterung auf mehrstellige Relationship-Typen möglich
  - grafisch: Beschriftung der Kanten
  - Beispiel: lesen (1:n), hören (m:n), prüfen (m:n:1)



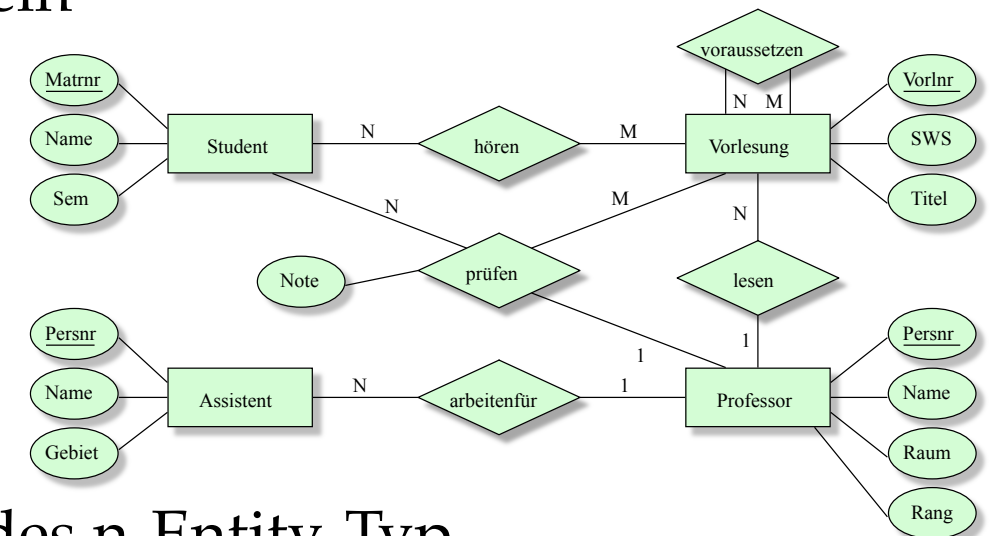
- ERM-Diagramm des Diskursbereichs Universität



- Relationales Datenmodell [Codd, 1970]
  - Relationenalgebra
  - Tabellen: Spalten (Attribute), Zeilen (Entities)
  - Integritätsbedingungen
- Relation R
  - Wertemengen  $V_i$ , Attribute  $A_i$
  - Teilmenge des kartesischen Produktes der  $V_j$
  - $R = R(A_1, A_2, \dots, A_n) \mid V_1 \leftrightarrow V_2 \leftrightarrow \dots \leftrightarrow V_n$
  - Tupel:  $r = (v_1, v_2, \dots, v_n)$  aus R
  - Relation: Tabelle
- Im Beispiel
  - Student(Matnr, Name, Semester)
  - Professor (Persnr, Name, Grad, Raum)
  - Vorlesung (Vorlnr, Titel, SWS, Leser)
- Integrität
  - Schlüssel-, Dömänen-, Entity-, referentielle Integrität

- Operationen auf Relationen
- Vereinigung  $R \cup S$
- Durchschnitt  $R \cap S$
- Differenz  $R - S$ 
  - alle Tupel aus  $R$ , die nicht in  $S$  sind
- Projektion  $\pi_L$ 
  - alle Spalten zu den Attributen aus Liste  $L$
- Selektion  $\sigma_B : R \rightarrow R$ 
  - alle Tupel, die Bedingung  $B$  erfüllen
- Produkt  $R \times S$ 
  - kartesisches Produkt aus 2 Tabellen
  - neue Tabelle mit Kombination aller Zeilen
- Verbund  $\bowtie$
- Division  $/$
- Umbenennung  $\rho$

- Überführung ERM -> RDM
- Entity-Typ
  - Relationsschema mit Namen, Attributen, Schlüsseln
- Relationship m:n
  - Relationschema mit Namen des R-typ
  - eigenen Attributen
  - Primärschlüssel der beteiligten Entity-Typen
- Relationship 1:n
  - Integration des 1-Entity-Typs in Relationschema des n-Entity-Typ
  - Primärschlüssels des 1-E-Typs als Fremdschlüssel
  - Aufnahme der Attribute des Relationship-Typs
- Relationship 1:1
  - Integration des 1-Entity-Typs in RS des anderen 1-Entity-Typ
  - Primärschlüssels des einen 1-E-Typs als Fremdschlüssel
  - Aufnahme der Attribute des Relationship-Typs
- Mehrstellige Relationship Typen
  - analog zu Relationship-Typ m:n

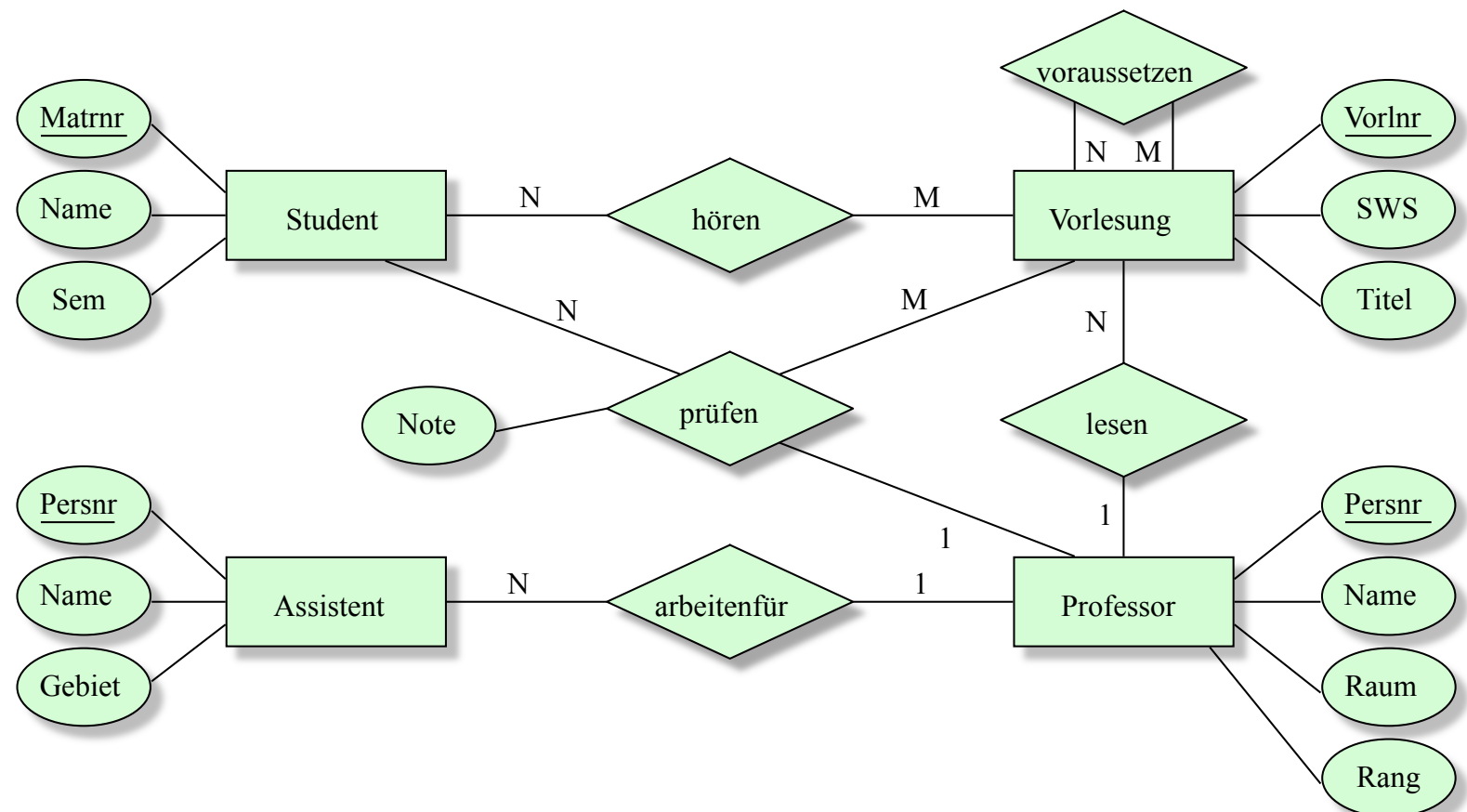


- Beispielrelationen

Assistent			
<u>Persnr</u>	Name	Gebiet	Chef
3002	Peters	Fische	2125
3003	Abel	Blumen	2125
3004	Werter	Insekten	2126
3005	Reiers	Bäume	2127
3006	Nebel	Vögel	2127

Professor			
<u>Persnr</u>	Name	Rang	Raum
2125	Schmidt	W3	212
2126	Rödel	W3	413
2127	Klopfer	W2	421
2133	Paul	W2	235
2134	Antons	W2	177
2136	Croll	W3	166
2137	Kasper	W3	402

Student		
<u>Matnr</u>	Name	Sem
24002	Turms	18
25403	Jasper	12
26120	Fuchs	10
26830	Anders	8
27550	Schubert	6
28106	Claas	3
29120	Tappers	2
29555	Fritz	1

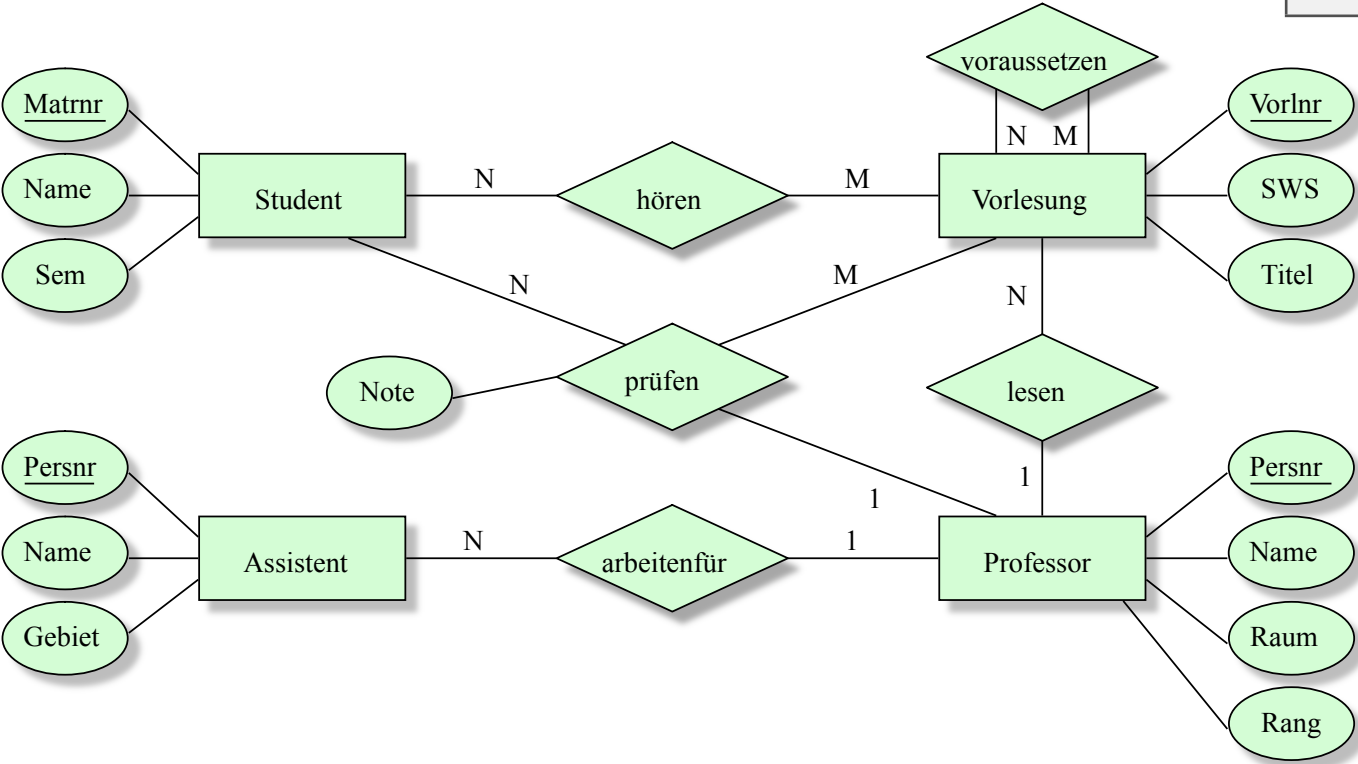


Vorlesungen			
<u>Vorlnr</u>	Titel	SWS	Leser
5001	Grundlagen	4	2137
5041	Ernährung	4	2125
5043	Eulenvögel	3	2126
5049	Gartenbau	2	2125
4052	Laufvögel	4	2125
5052	Wasserpflanzen	3	2126
5216	Biologie	2	2126
5259	Mimikri	2	2133

voraussetzen	
<u>Vorgaenger</u>	<u>Nachfolger</u>
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052

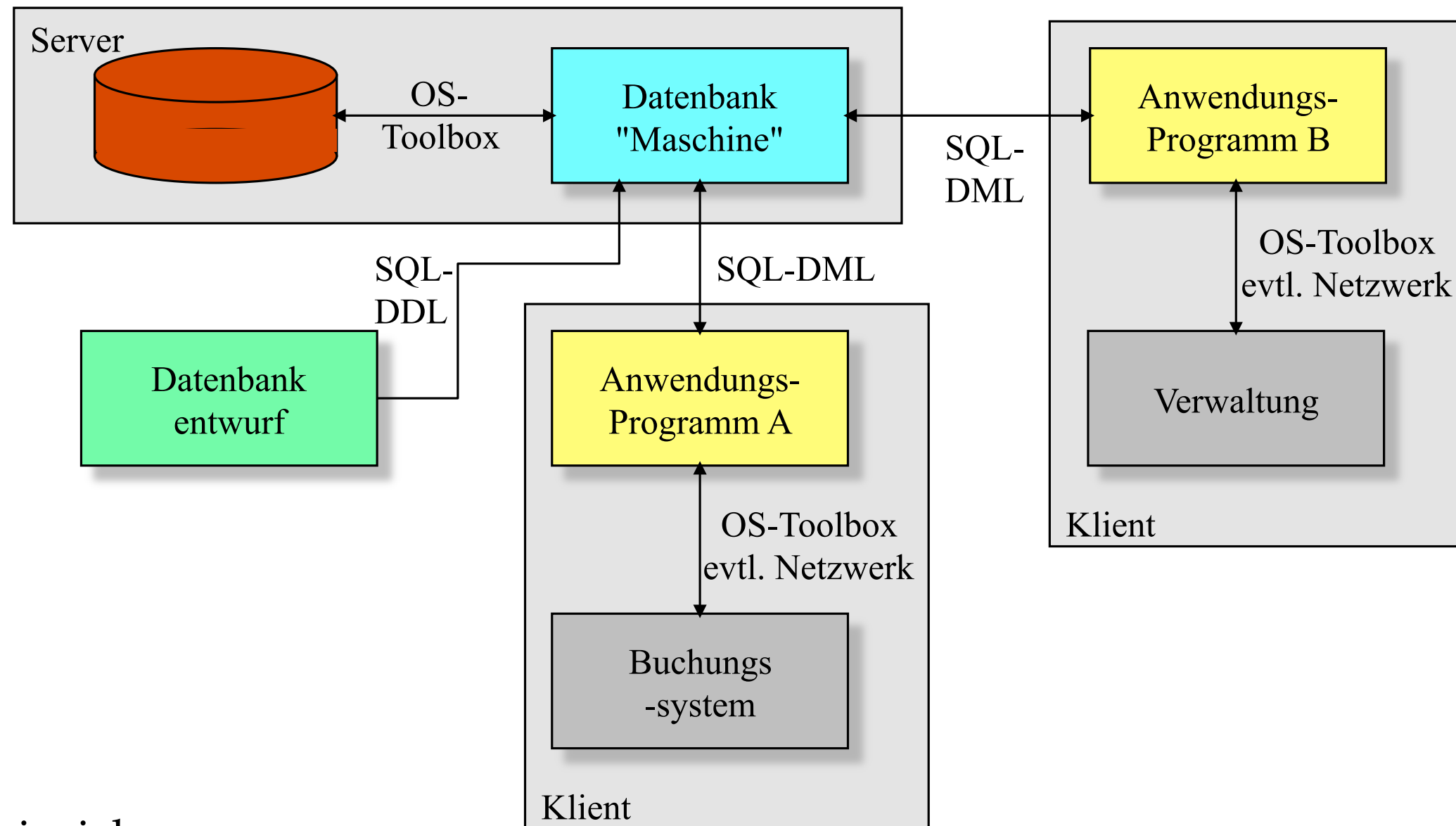
hoeren	
<u>Matnr</u>	<u>Vorlnr</u>
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen			
<u>Matnr</u>	<u>Vorlnr</u>	<u>Persnr</u>	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2





- Datenbanksystem



- Beispiele

- Microsoft Access, dbase, ...
- Oracle, Ingres, Informix, Adabas, Sybase
- mySQL, PostgreSQL, DB2

- SQL: Structured Query Language (SEQUEL, Structured English Query Language)
  - eine relationale Datenbanksprache
  - Datendefinitionssprache DDL
  - Datenmanipulationssprache DML
    - Änderungssprache
    - Abfragesprache
- Datenmanipulation / Abfrage

STANDANFRAGEN	
<code>select</code>	Projektion
<code>from</code>	Relation, Produkt, Join
<code>where</code>	Selektion, Differenz, Durchschnitt
<code>union</code>	Vereinigung
<code>all / any / in / exists</code>	Divison, Quantifizierung
<code>group by</code>	Aggregation
<code>order by</code>	Sortierung

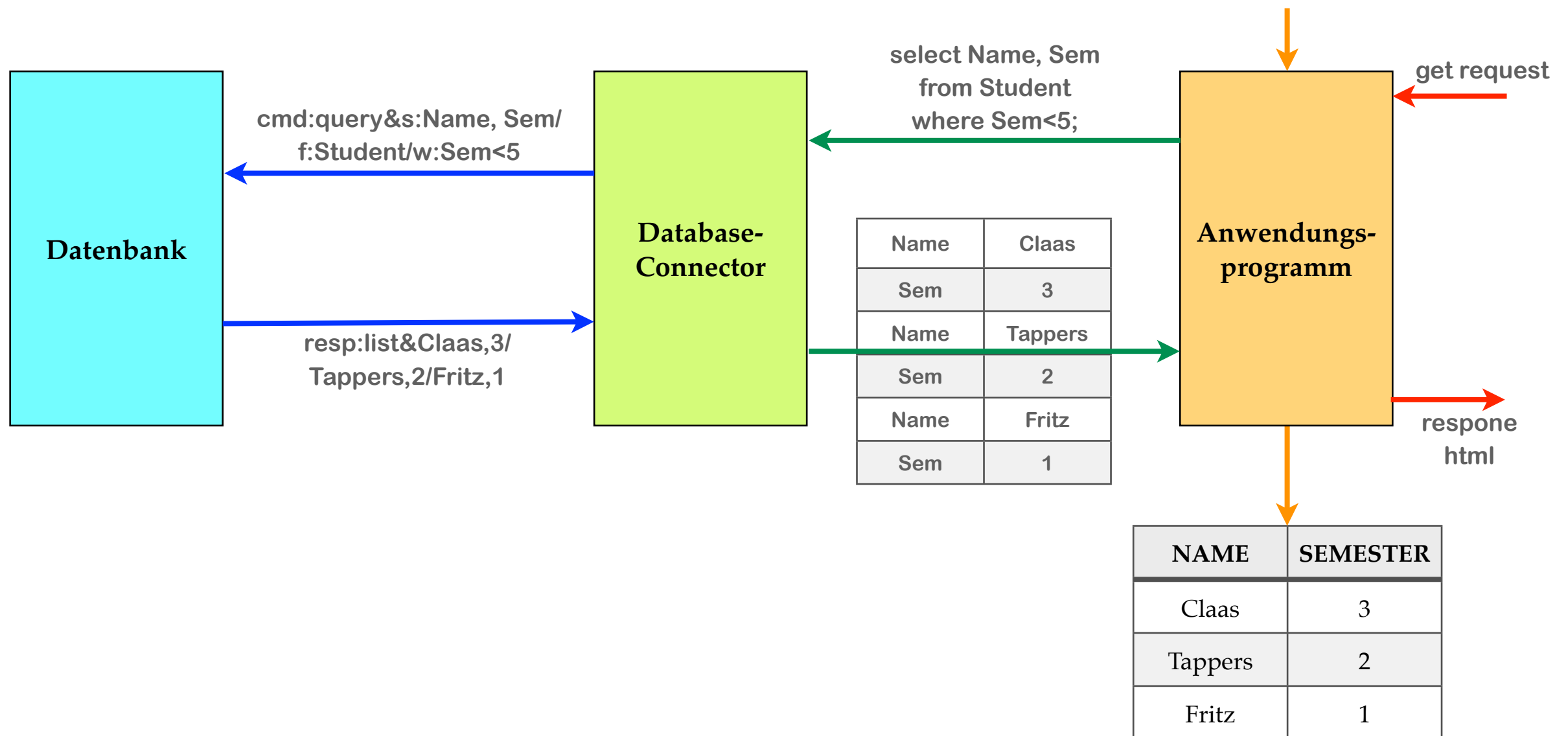
- Einfache Anfrage
  - `select` Spalte(n) `from` Tabelle(n) `where` Bedingung

- Abfrage aus einer Tabelle

```
select Matr, Name, Sem
from Student
where Sem<5;
```

```
select *
from Assistent;
```

```
select distinct Sws
from Vorlesung
where Leser=2125 and not Titel='Ernährung';
```



- Abfrage über mehrere Tabellen

select Name, Titel from Professor, Vorlesung where Persnr=Leser and Titel = 'Laufvögel';	select Name, Titel from Student, hören, Vorlesung where Student.Matnr = hören.Matnr and hören.Vorlnr = Vorlesung.Vorlnr;	select s.Name, v.Titel from Student s, hören h, Vorlesung v where s.Matnr=h.Matnr and h.Vorlnr = v.Vorlnr;
---------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------

- Spaltenauswahl

- Spaltenname, Konstante
- \* alle Spalten, **distinct** Duplikatbeseitigung

- Zeilenauswahl

- Spaltenname, Konstante
- Vergleichsoperatoren: =, !=, <>, >, >=, <, <=
- logische Verknüpfungen **and**, **or**, **not**
- Bereichsangaben **between** ... **and**, Muster **like** '...%...'
- Leertest **is null**, **is not null**

select Name from Student where Sem>= all (select Sem from Student);	select Name from Professor where not exists (select * from Vorlesung where Leser=Persnr);	select * from Vorlesung where Vorlnr in (5001, 5041);
---------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------	----------------------------------------------------------------

- Geschachtelte Abfragen

```
select * from Assistent
where Chef =
 (select Persnr from Professor
 where Name = 'Schmidt');
```

```
select * from prüfen
where Note = (select avg(Note) from prüfen);
```

- Mengenoperationen

- Vereinigung: **union**
- Durchschnitt: **intersect**
- Differenz: **except**

```
(select Name from Assistent)
union
(select Name from Professor);
```

```
(select Vorlnr from Vorlesung)
except
(select Vorlnr from hören);
```

```
select Vorlnr from Vorlesung
where Vorlnr not in
(select Vorlnr from hören);
```

- Quantifizierung

- **all, any, none** (für alle, ...)
- **in, not in** (Element von)
- **exists, not exists**: subquery

```
select Persnr, Name, Raum
from Professor
where Rang = 'W3'
order by Rang desc, Name asc;
```

```
select Name
from Student
order by Matnr;
```

- Gruppierung (ähnlich Sortieren)
  - **group by** Spalte(n) Zusammenfassung in Ergebnistabelle
  - **having** Bedingung Kriterium

<b>select Leser, sum(SWS) from Vorlesung group by Leser;</b>	<b>select Leser, sum(SWS) from Vorlesung group by Leser having avg(SWS)&gt;3;</b>	<b>select Leser, Name, sum(SWS) from Vorlesung, Professor where Leser=Persnr and Rang ='W3' group by Leser, Name having avg(SWS)&gt;3;</b>
----------------------------------------------------------------------	-----------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------

- Elementfunktionen
  - arithmetische Funktionen: +, -, \*, /, ...
  - Strings: char\_length, substring, ||, ...
  - Datum: current\_time, current\_date, +, -, \*, ...

<b>select Matnr, Note-1 from prüfen where Vorlnr=5001;</b>	<b>select Rang    '-Professor' from Professor;</b>	<b>select Matnr, Sem, current_date from Student;</b>
--------------------------------------------------------------------	--------------------------------------------------------	----------------------------------------------------------

- Sortieren
  - order by Spalte(n)
  - asc/desc

- Datenmanipulationssprache
- Einfügen
  - **insert into** Tabelle **values** Tupel;
  - **insert into** Tabelle Anfrage;

<b>insert into Professor values (2136,'Carl','W3',null);</b>	<b>insert into Student(Matnr,Name) values(25000, 'Gärtner');</b>	<b>insert into Assistent(Persnr,Name) select Matnr,Name from Student where Sem&gt;15;</b>
------------------------------------------------------------------	----------------------------------------------------------------------	---------------------------------------------------------------------------------------------------

- Ändern
  - **update** Tabelle **set** Werte **where** Bedingung;

<b>update Vorlesung set Sws = 2;</b>	<b>update Student set Sem = Sem+1;</b>	<b>update Professor set Raum = 213 where Name = 'Rödel';</b>	<b>update Vorlesung set Title='Grundlagen', SWS=3, Leser=2125 where Vorlnr = 5001;</b>
------------------------------------------	--------------------------------------------	----------------------------------------------------------------------	----------------------------------------------------------------------------------------------------

- Löschen
  - **delete from** Tabelle **where** Bedingung;

<b>delete from Vorlesung where Vorlnr=5001;</b>	<b>delete from Student where Sem&gt;=15;</b>	<b>delete from voraussetzen where Vorgänger in (select Nachfolger from voraussetzen);</b>
-----------------------------------------------------	--------------------------------------------------	---------------------------------------------------------------------------------------------------

- Datendefinitionssprache
  - create, alter, drop
  - in table, view
  - mit Rechte Vergabe (grant) und Entzug (revoke)
- Datentypen
  - char(n), varchar(n)
  - number(d), number(d,s)
  - date, long, blob
  - Namen: alphachar , { alphachar | digit }
- Tabellen
  - **create** definiert Tabelle
  - Spalten und Integritätsbedingungen

```
create table Tabellen-Name
(Spalten-Name Typ,
 Spalten-Name Typ not null,
 Spalten-Name Typ not null unique,
 ...)
```

```
create table Professor
(Persnr number(5) not null unique primary key,
 Name varchar(20) not null,
 Rang char(2),
 Raum number(4));
```

```
create table voraussetzen
(Vorgänger number(5),
 Nachfolge number(5));
```



# Bürosoftware

- Typische Büroarbeiten
  - Briefe schreiben
  - einfache und mittlere Berechnungen (Angebote, einfache Modelle)
  - Präsentation (Vorträge)
- Dokumentenverarbeitung
  - Integration verschiedener Medien
  - Text, Grafik, Bild, Tabellen
  - Unterstützungsfunktionen: Rechtschreibung, ...
  - Gruppenarbeit und Versionskontrolle
  - dynamische Komposition
- Office-Pakete
  - MS-Office
  - Staroffice, OpenOffice, LibreOffice
  - ...

## Text

- vi, WordStar, Edit, Edlin, Notepad, ...
  - konzeptuell eine lange Zeichenkette mit Space zwischen Wörtern
  - 'File' = Magnetbandkonzept
- Zeichen: ASCII, EBCDIC, ISO 8859-X, Unicode (siehe [7.2](#))
  - Buchstaben, Ziffern, Sonderzeichen
  - A, B, C, D, ..., Z, Ä, Ö, Ü
  - a, b, c, d, ..., z, ä, ö, ü, ß
  - 0 ... 9
  - ., # + \* ! " § \$ % & / ( ) = ? ' ^ < > ^
  - Steuerzeichen für Ausgabegerät: [LF](#), [CR](#), [FF](#), [NL](#), [VT](#), [HT](#), ...
- Tastatur als Eingabegerät
  - Tasten für die wichtigsten Buchstaben
  - Umschalttasten
  - evtl. nicht alle Zeichen eingebbar

- Zeilen

- Einheit des Darstellungsmediums (Papier, Bildschirm)
- Breite bestimmt Anzahl Zeichen in der Zeile

Der durchschlagende Erfolg von Internet, WWW und der modernen digitalen Kommunikationstechnologien hat einen neuen Wirtschaftszweig rund um Kommunikation, Information und elektronischen Handel (E-Commerce) geschaffen. Diese neue Industrie hat sehr hohen Bedarf an einem neuen Typus Ingenieur, dessen Spezialität Softwareanwendungen in Netzwerken (Network Computing) sind. Die starke betriebswirtschaftliche Orientierung eines solchen Fachmanns ist eine weitere unverzichtbare Qualifikation.

- Absatz

- logische Struktur
- prinzipiell beliebig lange Zeichenkette
- Objekt vieler Funktionen der Verarbeitung

Der durchschlagende Erfolg von Internet, WWW und der modernen digitalen Kommunikationstechnologien hat einen neuen Wirtschaftszweig rund um Kommunikation, Information und elektronischen Handel (E-Commerce) geschaffen. Diese neue Industrie hat sehr hohen Bedarf an einem neuen Typus Ingenieur, dessen Spezialität Softwareanwendungen in Netzwerken (Network Computing) sind. Die starke betriebswirtschaftliche Orientierung eines solchen Fachmanns ist eine weitere unverzichtbare Qualifikation.

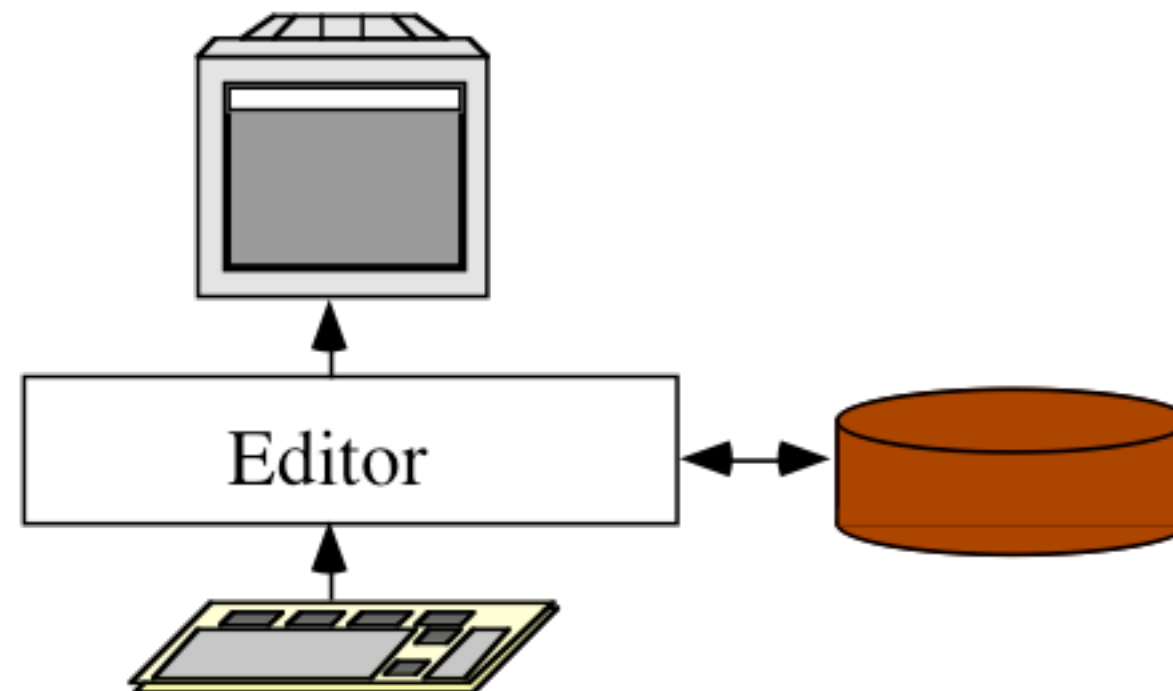
Die Fakultät für Mathematik und Informatik der Technischen Universität - Bergakademie Freiberg hat als erste deutsche Universität einen Bachelor-Studiengang, der genau solche Ingenieure schnell und gründlich ausbildet, entwickelt. Wir hoffen, so unseren Studenten einen sicheren und zukunftsorientierten Beruf zu eröffnen.

- Umbruch

- Abbildung von Absätzen auf Zeilen
- Abbildung der Zeilenmenge auf Seiten

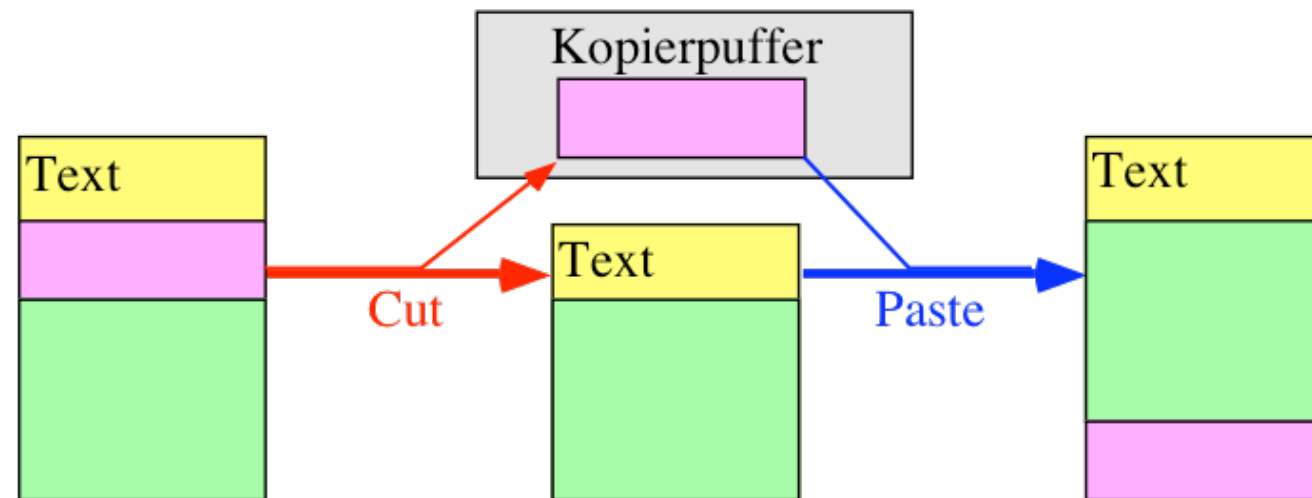
- Zeilenwechsel  $\leftrightarrow$  Absatzwechsel

- Editor: einfaches Textverarbeitungsprogramm
  - Zeichen eintippen
  - Zeichenkette anzeigen
  - Cursor: Bearbeitungsposition: einfügen, löschen
  - Cursor positionieren mit Spezialtasten oder Maus
  - Speichern, Drucken, ...



- Markieren
  - Cursor setzen, Umschalttaste + Cursor neu setzen
  - Markieranfangs-Zeichen, Cursor bewegen, Markierende-Zeichen
  - auch ohne Maus oder Touchpanel

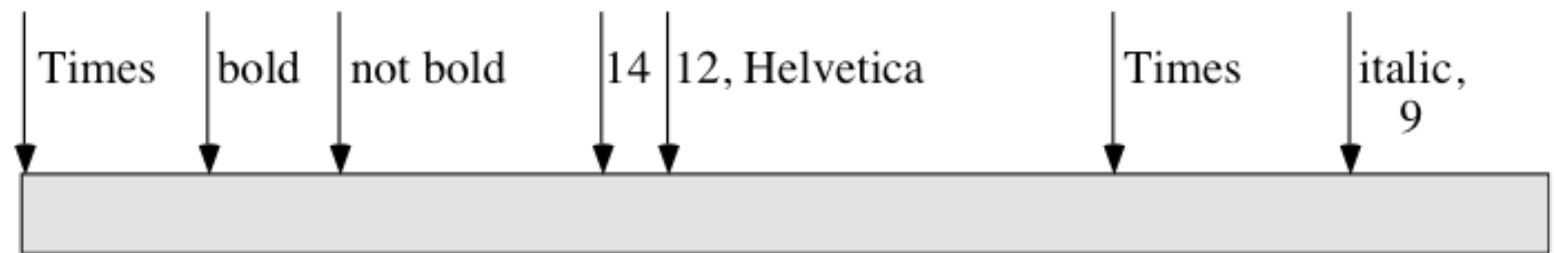
- Kopierpuffer
  - Text markieren
  - Kommando: ausschneiden oder kopieren
  - Zielposition setzen
  - Kommando: einsetzen
  - Zwischenablage, Clipboard



- Einfache Satzmethoden
  - Tabulator
  - Leertaste (Space, Blank)
  - Leerzeilen
  - aber Vorsicht!

## Formatierung

- Satzregeln aus der Buchdruckkunst
  - Ziel: verbesserte Lesbarkeit und "Schönheit"
  - widows and orphans
  - unterstreichen ist unschön
  - kein Fontsalat



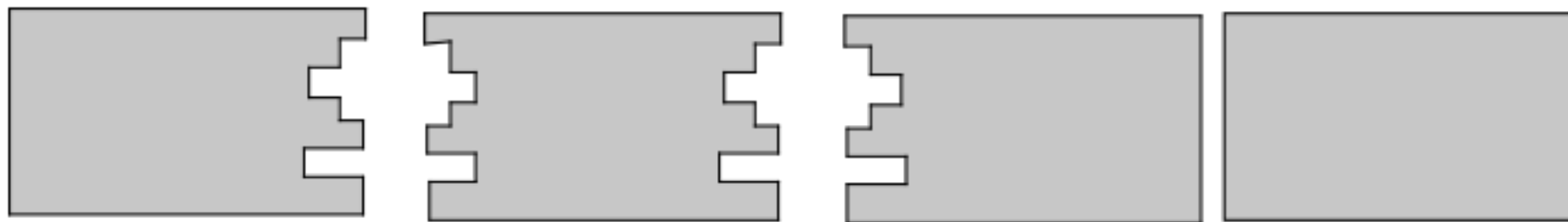
- Text bekommt Attribute
- Text immer noch sequentiell
  - Attribute gelten für alle folgenden Zeichen
  - bis anderes Attribut der selben Kategorie
- Buchstabenbild und Zeichensatz (Font) siehe Kapitel Medien
- Zeichenformat bezieht sich auf einzelne Zeichen

Der **durchschlagende** Erfolg von *Internet*, WWW und der **modernen** digitalen Kommunikationstechnologien hat einen neuen Wirtschaftszweig rund um Kommunikation, Information und **e**lektronischen **Handel** (E-Commerce) geschaffen. Diese neue Industrie hat sehr hohen Bedarf an einem neuen Typus Ingenieur, dessen Spezialität Softwareanwendungen in Netzwerken (Network Computing) sind. Die starke **betriebswirtschaftliche**

**Orientierung** eines solchen Fachmanns ist eine weitere **un**verzichtbare Qualifikation.

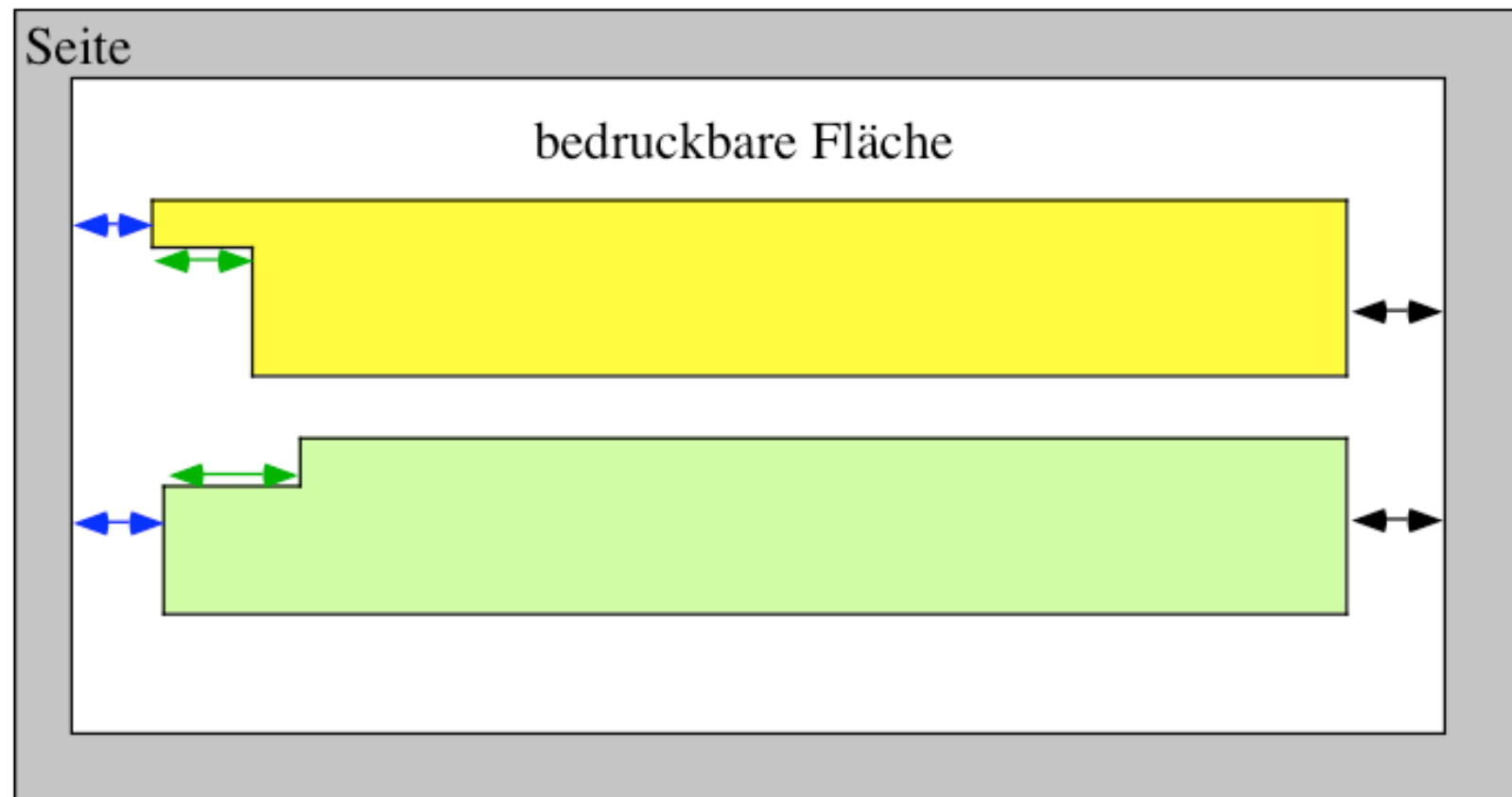
- Kerning und Ligaturen

- Tabulatoren: horizontale Marken
  - stabile Formatierung
  - Sprung zur Marke
  - linksbündig, rechtsbündig, zentriert, Zahl, Währung
- Absatz
  - inhaltlich eng zusammengehörender Text
  - semantische Einheit
  - von anderen Absätzen abgesetzt
  - automatische Absatzformatierung aus Formatvorlage (Stile)
- Form des Absatzes
  - Flattersatz: linksbündig, rechtsbündig, zentriert



- Blocksatz
- Absatzzeilen zusammenhalten
- Mit nächstem Absatz zusammenhalten

- Einzug = Abstand vom horizontalen Rand
  - links, rechts
  - **hängend** (= Sonderfall erste Zeile)

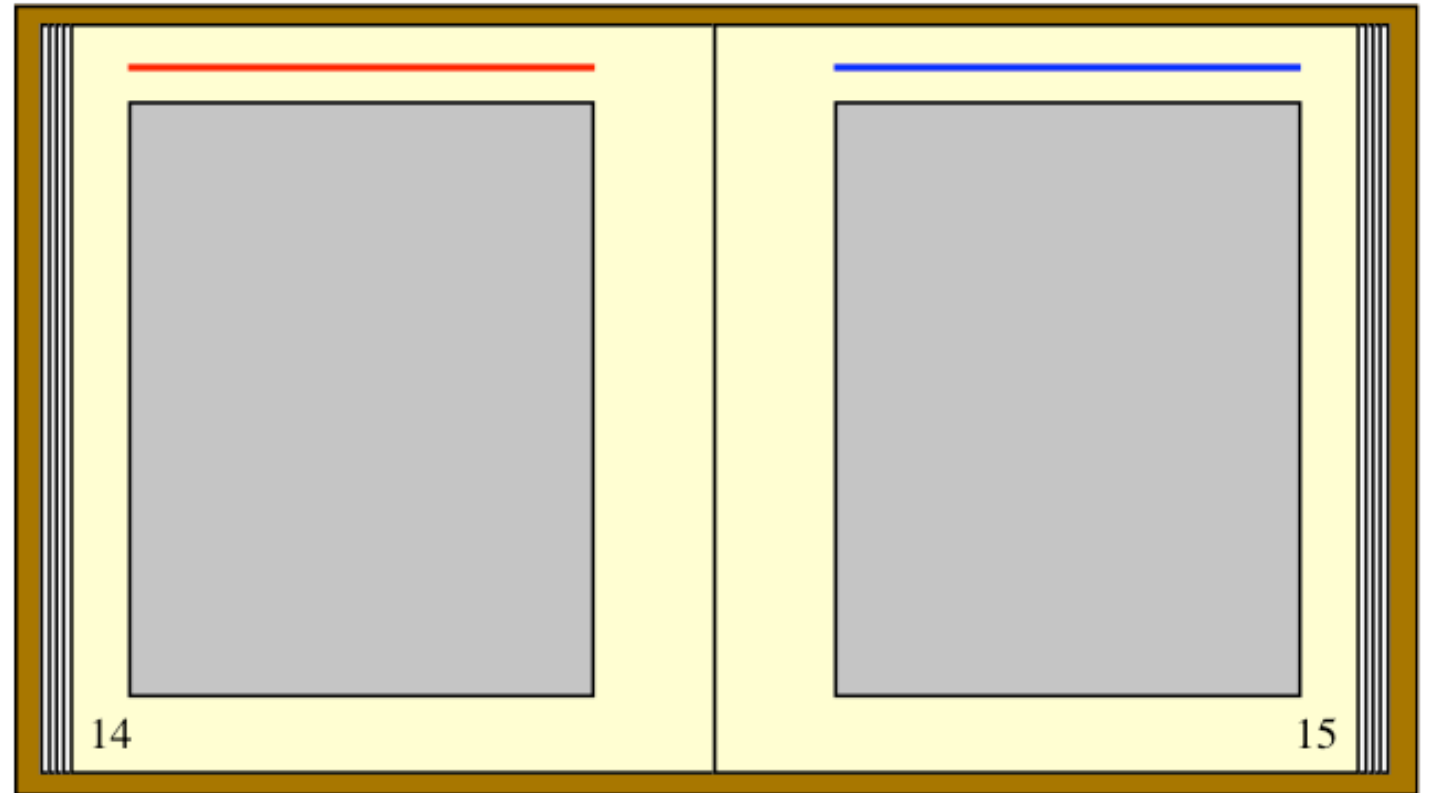


- Zeilenabstand (Durchschuß, leading)
- Abstand zu andern Absätzen
  - vorher, hinterher
  - Sonderfall oberer Rand bzw. unterer Rand
  - Leerzeilen sind "schlechter Stil" bzw. gefährlich



- Maßeinheiten

- 1 Inch = 2,54 Zentimeter
- 1 Zeile = 1 Pica = 1 / 6 Inch
- 1 Punkt = 1 pt = 0,035 cm
- 12 Punkt = 1 Zeile = 1 / 6 Inch

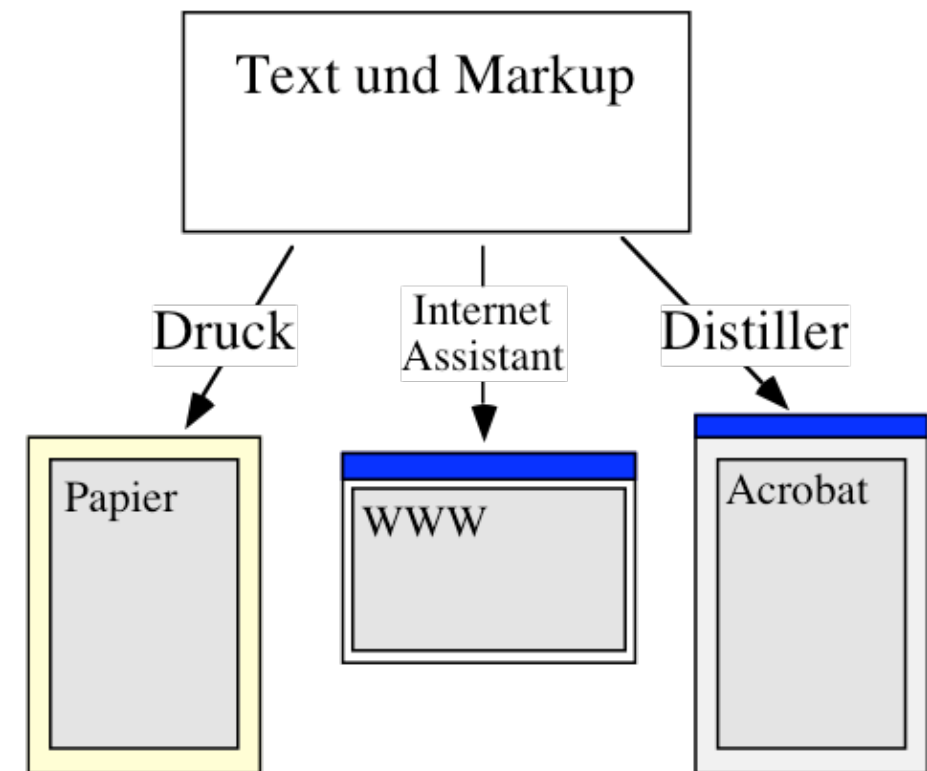


- Seitenmaße

- bedruckbare Fläche kleiner als Papier
- Abstände: links, rechts, oben, unten
- Bücher: gespiegelt für gerade und ungerade Seiten
- besondere Seitenbereiche: Kopfzeilen, Fußzeilen
- Seitennummern

## Text und Struktur

- Markup
  - logische Struktur für Text
  - Überschrift, normaler Paragraph, Zitat, ...
  - Fußnote, Literaturverweis, Bildunterschrift, ...
- Zuordnung der Attribute beim Satz
  - Autor produziert Inhalt und Struktur
  - Drucker setzt
  - Corporate Identity ...
- Tex
- Formatvorlagen (Stile, Styles)
  - Mehr als 2 Seiten => Formatvorlagen
  - Menge von Attributen für Zeichen
  - Menge von Attributen für Absatz
  - Analogie zur Programmiersprache: Prozeduren für Satz
- Besondere Formatvorlagen
  - Überschriften => automatisches Inhaltsverzeichnis
  - Abbildungsverzeichnis, ...

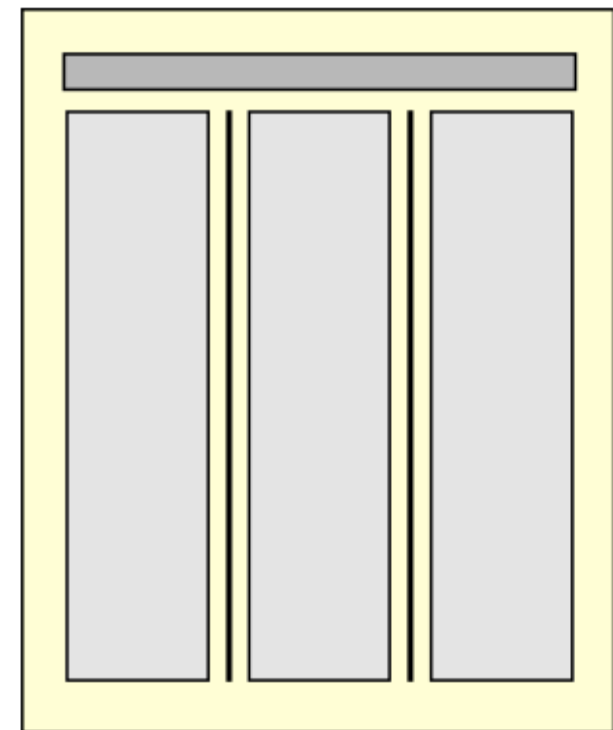
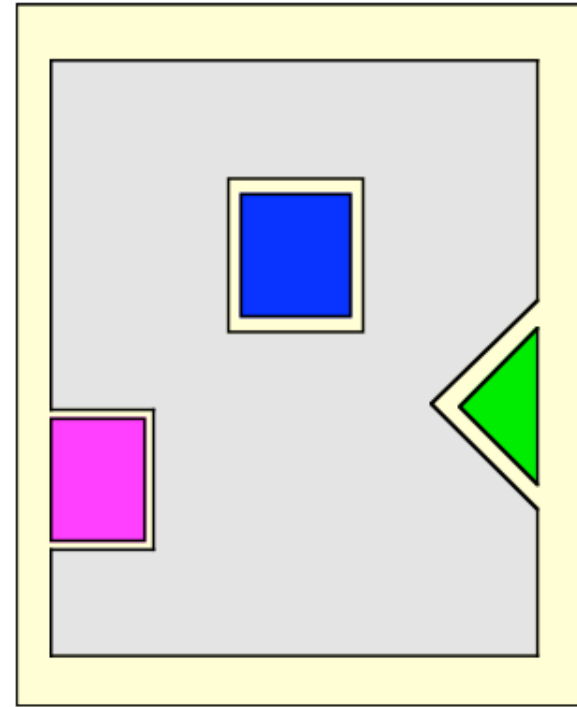


## Spezialfunktionen für Text

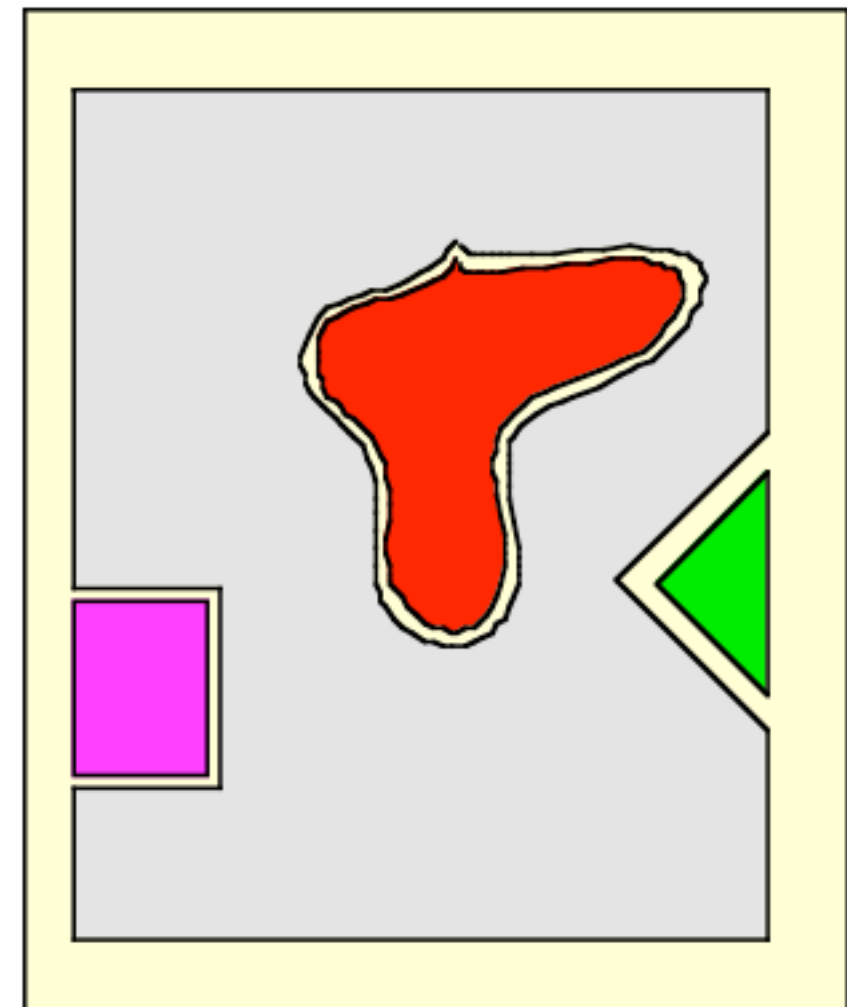
- Trennung
  - manuell mit Sonderzeichen oder am Bindestrich
  - automatische Silbentrennung
  - semi-manuell
- Rechtschreibprüfung
  - manuell oder automatisch (aber: Intervall  $\rightarrow$  Interval  $\Rightarrow$  Interwal)
  - Wörterbücher für Sprachen
  - neues Textattribut: Sprache
  - persönliche Wörterbücher
- Grammatik
  - einfache Regeln
  - stereotype Ratschläge
  - funktioniert in Englisch besser
- Thesaurus
  - Synonyme und Antonyme finden
- Zählfunktionen: Wörter, Buchstaben, ...
- Tabellen, Listen, grafische Rahmen, ...

# Dokumentenverarbeitung

- Komponenten
  - Text
  - Grafik
  - Bild
  - Objekte von anderen Programmen
- Mischen der Komponenten
  - Textfluß
  - Objekt läuft mit Text
  - Objekt fixiert, Text läuft herum
- Spalten
- Positionsrahmen
- Dynamisches Einbetten von Objekten (OLE)

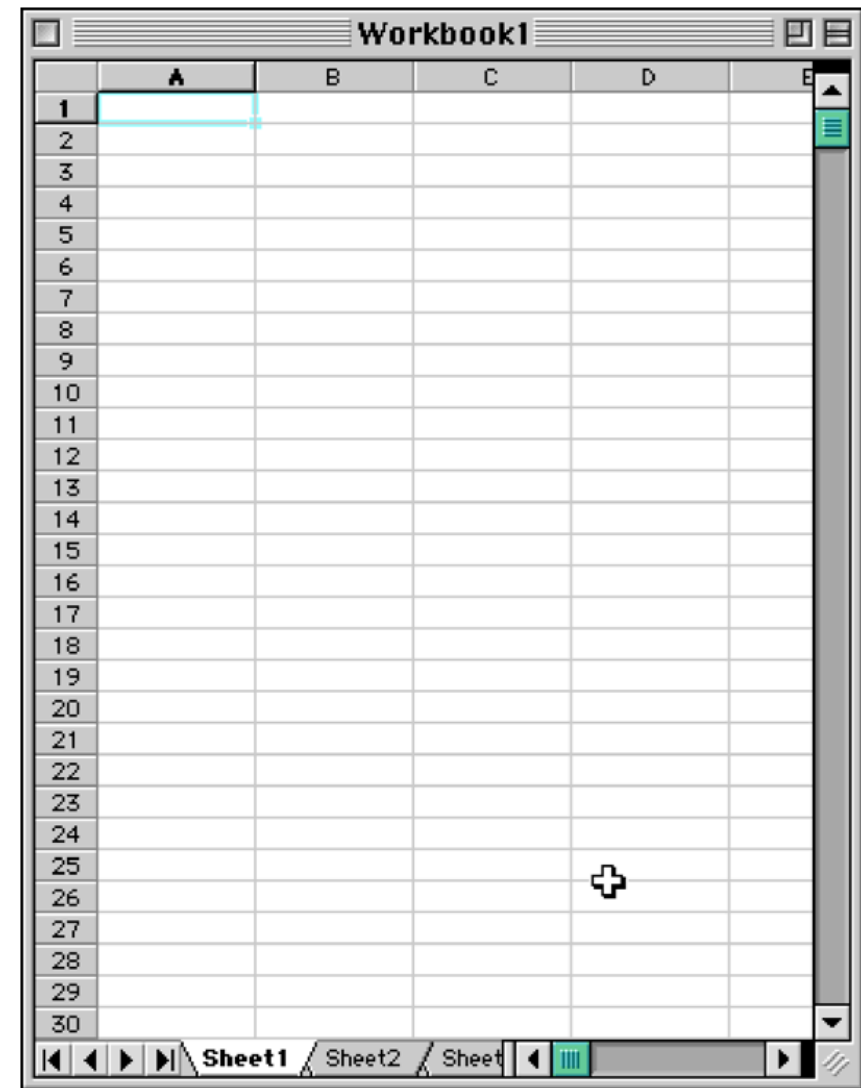


- Besondere Programme für Computersatz
  - InDesign, Quark Xpress
  - Schnittstelle zu Druckmaschinen (Farbseparation, ...)
- Seitenorientiert
- Textfluß zentrales Element
  - präzise Angabe der Textfläche
  - Text fließt um Objekte herum
- Objekte
  - Position auf der Seite fest
  - beliebige Kontur der Objekte
  - Text folgt der Kontur
- Viele weitere Eigenschaften
  - Inhaltsverwaltung, Bearbeitungshilfen
  - Farbkontrolle
  - hairlines ...



## Tabellenkalkulation

- Berechnungen
  - Zahlenkolonnen addieren
  - Angebote:  $(10 * 1,53 + 12 * 12,5 + \dots + 15,73) * 1,16$
  - einfache Modelle: Gehalt berechnen, Steuerformel, ...
  - Kontostand projizieren
- Visicalc [Bricklin, 1978] für Apple II
- Lotus 1-2-3 [Kapor, 1982] für IBM-PC
- Aktuell: Excel, Lotus, Quattro
- Grundidee: Rechenblatt
  - Spreadsheet
  - Zeilen und Spalten
  - Formeln
  - Variablen
  - einfache Programmierkonstrukte



- Felder

- enthalten ein 'Datum'
- Typ
- Adresse = (Spalte,Zeile)

- Datentypen

- Zeichenketten
- Zahlen
- Währung
- Datum, Zeit
- Postleitzahl, Telefonnummer, Social Security Number

- Formatieren

- Zeichenattribute ähnlich Textprogramm
- Datentyp-Formate (Währung, Zeit, ...)
- Dezimalstellen
- 0,13 = 13%

Debitor	Rechnung vom	Betrag
Müller	12-Jan-99	1713,12
Meier	13-Jan-99	5312,5
Schmidt	17-Jan-99	1800,67
Hartmann	03-Feb-99	782
Meier-Möllenc	09-Feb-99	1190,78

## Formeln

- Elemente
  - Konstante
  - Zellinhalte
  - Operatoren
  - Funktionen
- Arithmetik
  - +, -, \*, /
  - Punkt-vor-Strich
  - Klammern
  - auch mit Datum rechnen
- Bezüge
  - Tupel (Spalte, Zeile)
  - ähnlich Adressen in der Programmiersprache
  - auch Zellbereiche A1:B6 (Rechteck!)
- Konstante
  - Zahlen, Datum, Prozent
  - Text



- Relative Bezüge

- Normalfall, automatisch
- (Spalte-n, Zeile-n)
- oder absolut **angezeigt**
- A1, B17, AA123, ...

- Absolute Bezüge

- Ausnahme
- Adresse (Spalte, Zeile) fest
- Kennzeichen: \$
- Bsp: \$A\$3
- absolute Zeilenadresse: A\$3
- absolute Spaltenadresse \$A3

- Bezüge und Editieren

- automatische Anpassung relativer Bezüge
- beim Kopieren, nicht beim Verschieben
- auch beim Einfügen / Löschen 'dazwischen'
- auch bei Bereich füllen

=Y7+Y11-Z9				

- Funktionen
  - <name>(<argument1>;<argument2>;...;<argumentn>)
  - Summe(), Max(), Min(), Anzahl(), ...
  - Zellmenge ist ein Argument: Summe(A3:B10)
  - Schachtelung möglich
- Funktionsgebiete
  - Mathematik, Trigonometrie, Statistik
  - Finanzmathematik (Annuität, ...)
  - Datum und Zeit, Text
  - Logische Funktionen, Feldinformation
- WENN(Bedingung;<then-val>;<else-val>)
  - WENN(B17<>0;B16 / B17;0)
  - Bedingung mit UND(), ODER(), NICHT()
- Funktionsassistent
  - kennt Funktionsnamen
  - hilft bei Argumenten

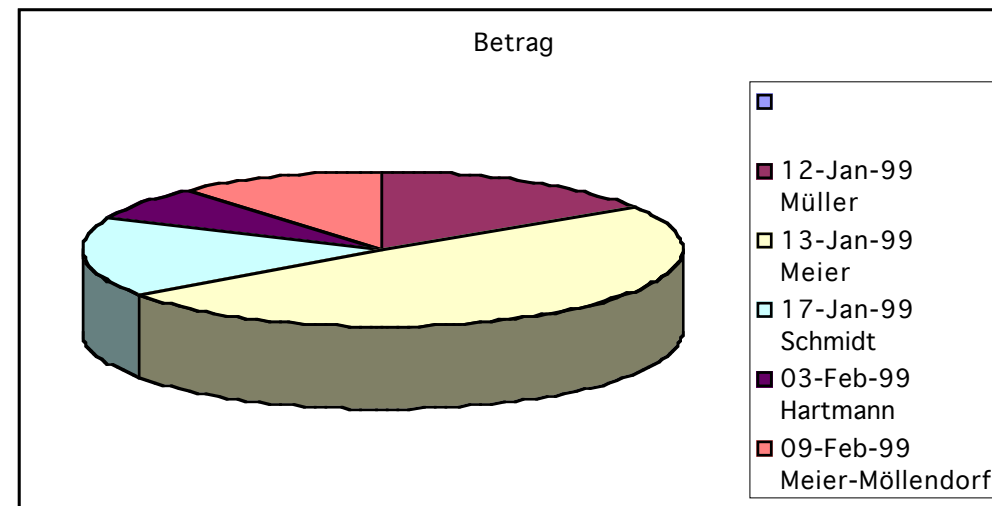
## Namen

- Tabellenkalkulation ist Programm!
  - dokumentieren der Formeln
  - Sicherheit bei späteren Änderungen
- Zellbezüge unübersichtlich
  - Bezug auf Prozentsatz etc.?
  - dokumentiert nicht
- Zellen können benannt werden
  - einzeln oder Bereich
  - Namensregeln ähnlich Programmiersprachen
  - benannte absolute Bezüge
  - Indirektion: Name -> Zelle
- Namen in Formeln einsetzen
  - verändern sich nicht beim Kopieren
  - Auswahl aus Bereich

# Diagramme

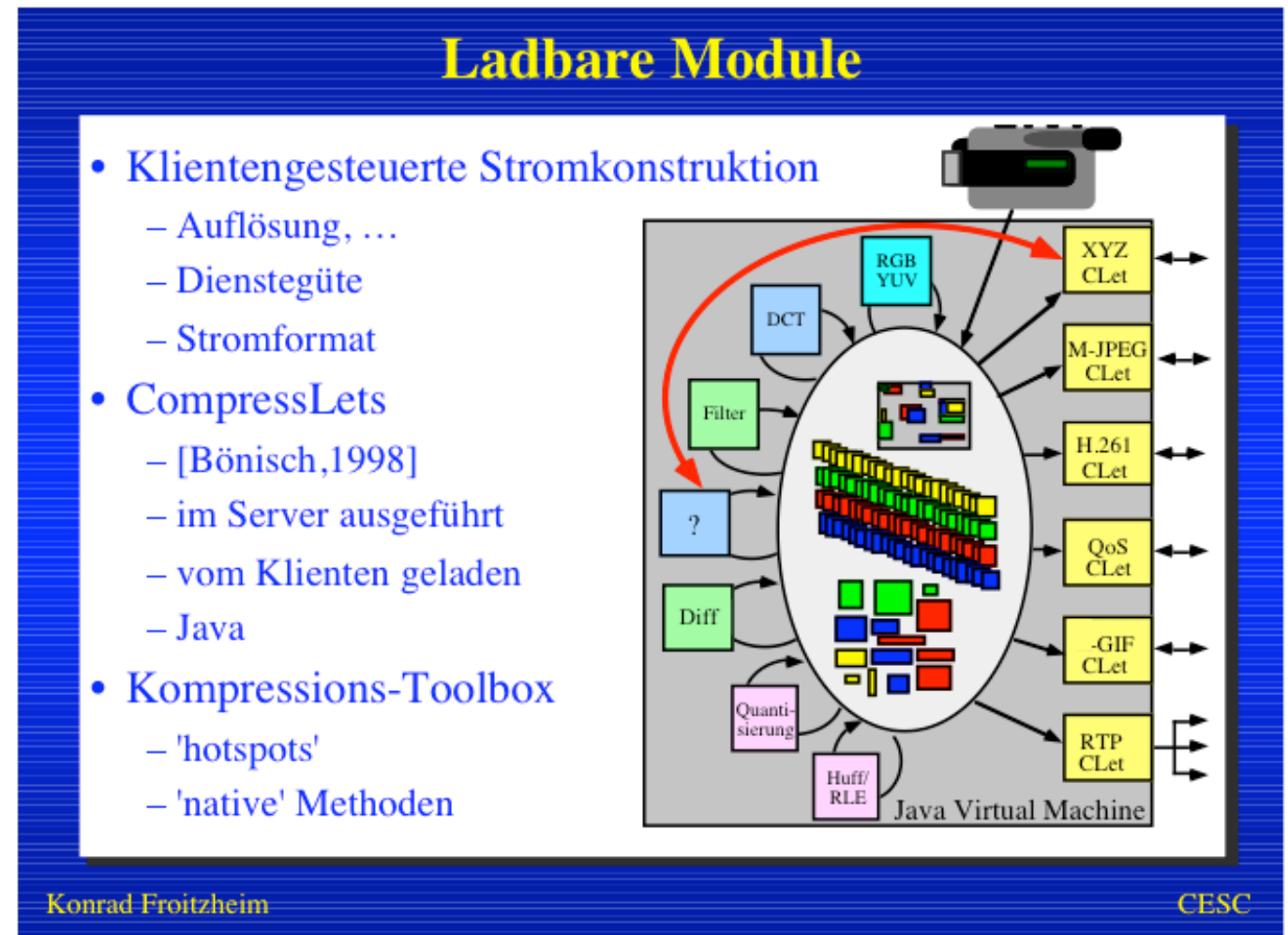
- Visualisierung einer Tabelle
  - Anteile
  - Trends
- Tabelle selektieren
  - Achsen
  - Werte
- Diagramm-Art wählen
  - Kuchen
  - Linie, Punkte
  - Pseudo 3D
- Formatieren
  - Achsen, Beschriftung
  - Legende

Debitor	Rechnung vom	Betrag
Müller	12-Jan-99	1713,12
Meier	13-Jan-99	5312,5
Schmidt	17-Jan-99	1800,67
Hartmann	03-Feb-99	782
Meier-Möllenc	09-Feb-99	1190,78

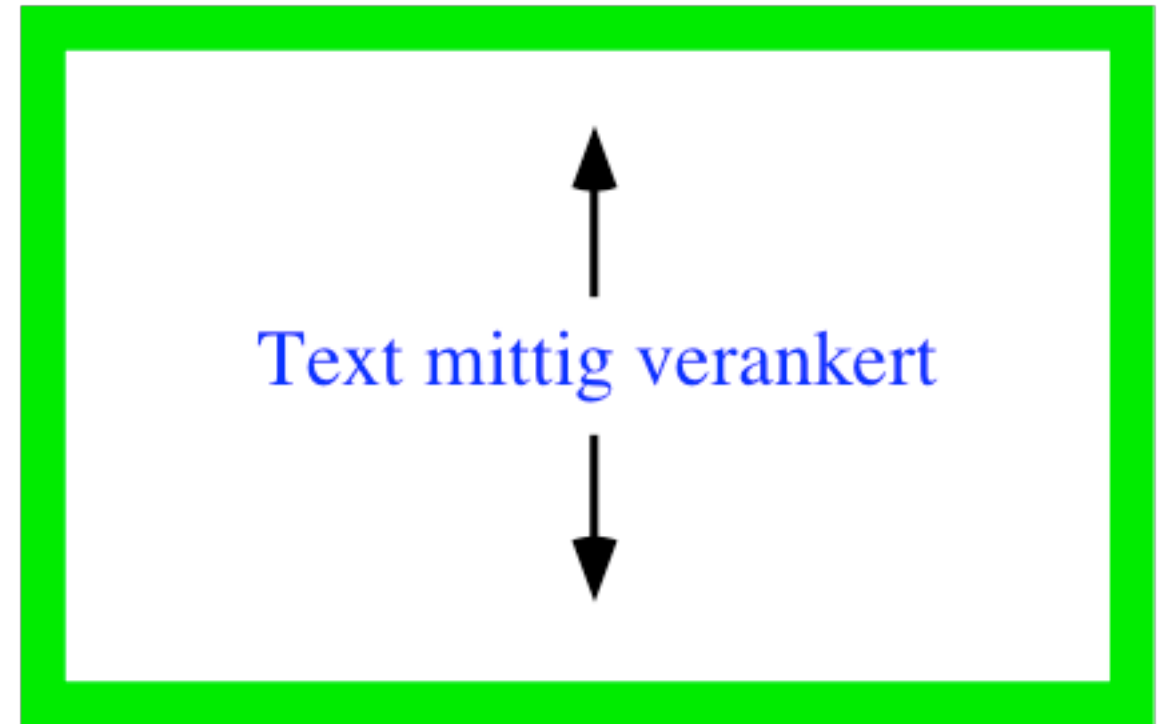


## Präsentationen und Vorträge

- Elektronische Folien
  - Text (Punkte)
  - Grafiken
  - Animation
- Didaktische Aufbereitung
  - sequentielle Präsentation
  - Hauptpunkte - Details
  - Präsentationskontrolle
- Dramaturgie
  - Übergänge zwischen Kapiteln
  - Einzelpunkte animieren
  - Überblendungen
- Powerpoint, Impact, Keynote, ...



- Textformat
  - vertikaler Text-Anker
  - Zeichenformate wie gewohnt
- Zeichenprogramm eingebaut
  - fertige Elemente
  - Clip-Art
  - geometrische Grafik
- Animierte Grafik
  - Komponenten einblenden lassen
  - Art der Einblendung wählbar
  - Anlaß wählbar
  - Geräusch möglich
  - Bewegung = ausblenden, verschoben einblenden



- Entwurfshilfen
  - Grunddesign mit 'Wizard'
  - Outline mit Überschriften, Hauptpunkten, Unterpunkten
  - Einzelfolie
  - Notiz-Seiten
  - 'Lichtkasten' zum Sortieren
- Ausgabeoptionen
  - elektronische (Präsentation, Arbeitsplatz, Kiosk)
  - Folien
  - Papier: 'Handouts', Notiz-Seiten
  - WWW leider nur Bitmap
- Präsentationshilfen
  - Generalprobe mit Zeitnahme
  - zeitgesteuerte Präsentation

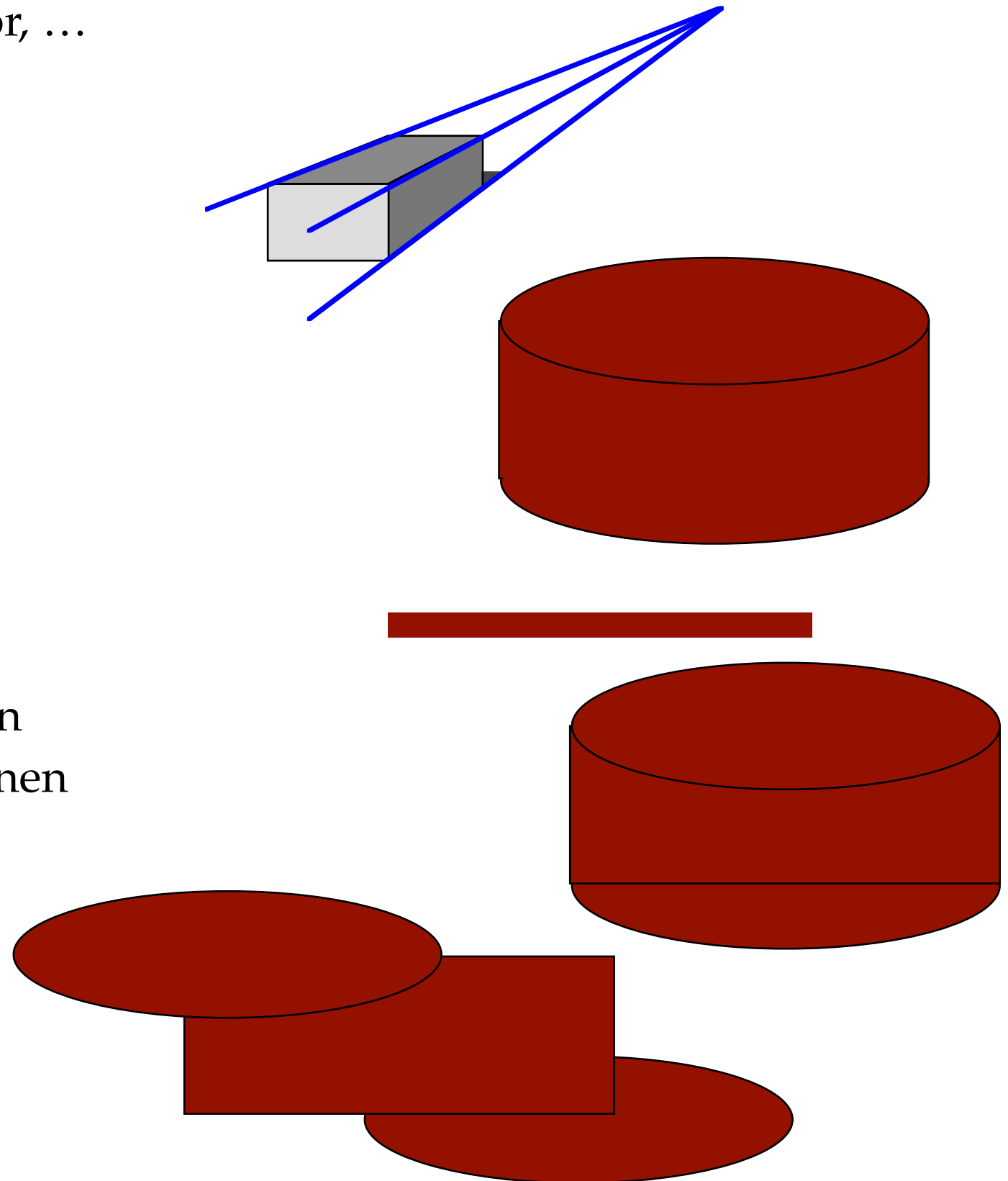
# Medien bearbeiten

- Grafiken: Line-Art
  - MacDraw, Corel, Illustrator, ...
  - Strecken, Linienzüge
  - Rechtecke, Polygone
  - Kreise, Ellipsen
- Pixelmanipulation
  - Photoshop, Gimp, ...
  - Photo retouch
  - Photomontage
- Schneiden und Kombinieren
  - Premiere, FinalCut, After Effects
  - Spurkonzept
  - Orchestrierung



## Objektgrafik

- MacDraw, Corel, Canvas, Illustrator, ...
- Einfaches CAD
- Graphische Objekte
  - Strecken, Linienzüge
  - Rechtecke, Polygone
  - Kreise, Ellipsen
- Attribute
  - Farbe, Muster, Stiftform, ...
  - Linien, Fläche
- Anordnung der Objekte
  - zusätzliche geometrische Figuren
  - Verdecken von Hilfskonstruktionen
- Pseudo-3D
  - Perspektive
  - Schattenwurf

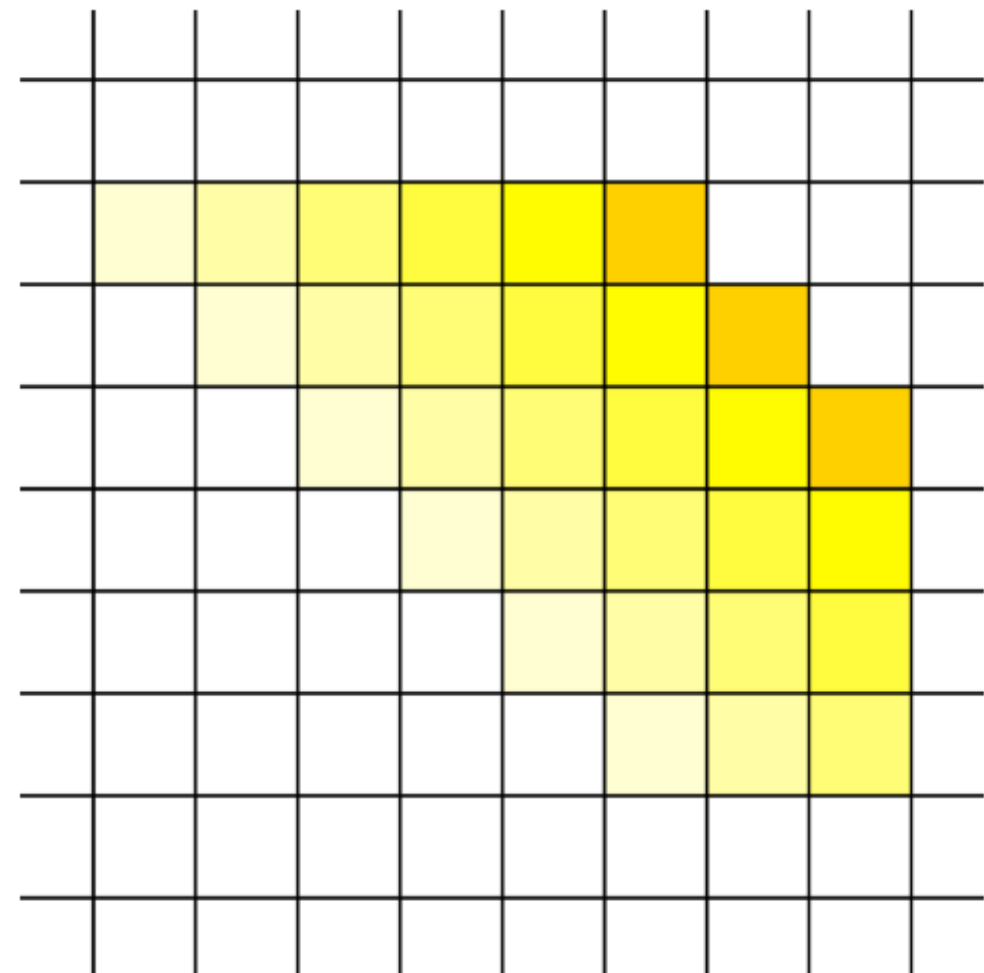


## Pixelgrafik

- MacPaint, PaintShop, ...
- Photos und Bilder
  - Pixel
  - 'glatte' Übergänge
  - Farben
- Werkzeuge
  - Stift: Punkte setzen
  - Radiergummi: Punkte entfernen
  - Farbeimer: ausfüllen
  - Spraydose
- Attribute
  - Farbe, Muster, Stiftform, ...
  - Linien, Fläche



[P. Schulthess]



## Photoshop, GIMP etc

- Bildverbesserung
  - Retouch
  - Kontrast, Helligkeit, Farbsättigung, Histogramm-Equalization
  - Kanten bearbeiten (Schärfe, ...)
- Selektion
  - Rechteck, Ellipse und Lasso
  - 'magnetisches' Lasso
  - "Zauberstab"
  - farbgesteuert
- Filter
  - Störungen entfernen (z.B. Durchschnitt mit Nachbarn)
  - Effekte
- Zeichenwerkzeuge
  - Stift, Pinsel, Farbeimer, Radiergummi

- Digitalisierte Bilder sind verfälscht
  - CCD-Rauschen, unterschiedlich stark in den Farbkanälen
  - Helligkeit und Kontrast
  - Farben schlecht (Farbtemperatur des Lichtes ...)
- Rauschen entfernen
  - Lab Farbraum
  - blauer Kanal besonders verrauscht
- Nachschärfen
- Kurven (Helligkeit, Kontrast, Farbkanäle)
  - Histogramme
  - Farbkorrektur
- Effekte
  - als Filter implementiert
  - Verwischen (blur, PSF), Kanten schärfen, ...
- Kai's Powertools
  - lokale und globale Transformationen



0.01	0.1	0.25	0.1	0.01
0.1	0.35	0.5	0.35	0.1
0.25	0.5	1.0	0.5	0.25
0.1	0.35	0.5	0.35	0.1
0.01	0.1	0.25	0.1	0.01

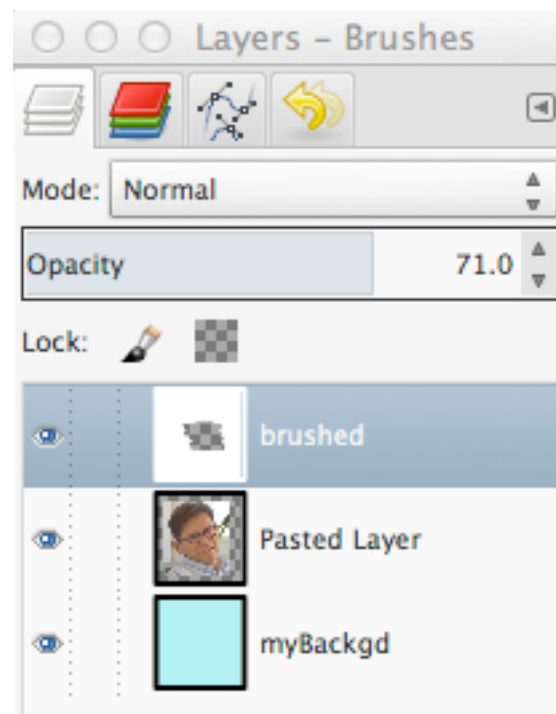
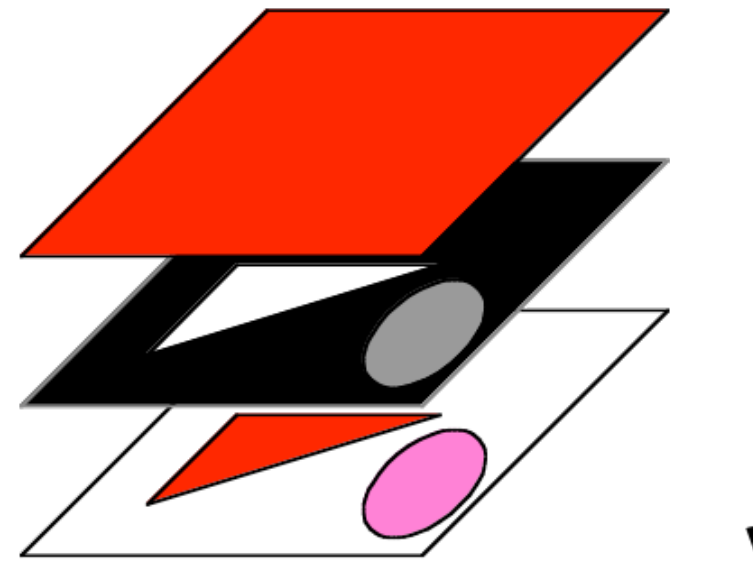
- Räumliche Anordnung der Bildelemente

- Layer
- Organisation des Bildes
- wichtige Elemente in Layern
- Transparenz
- arithmetische Verknüpfung

- Farbkalibrierung

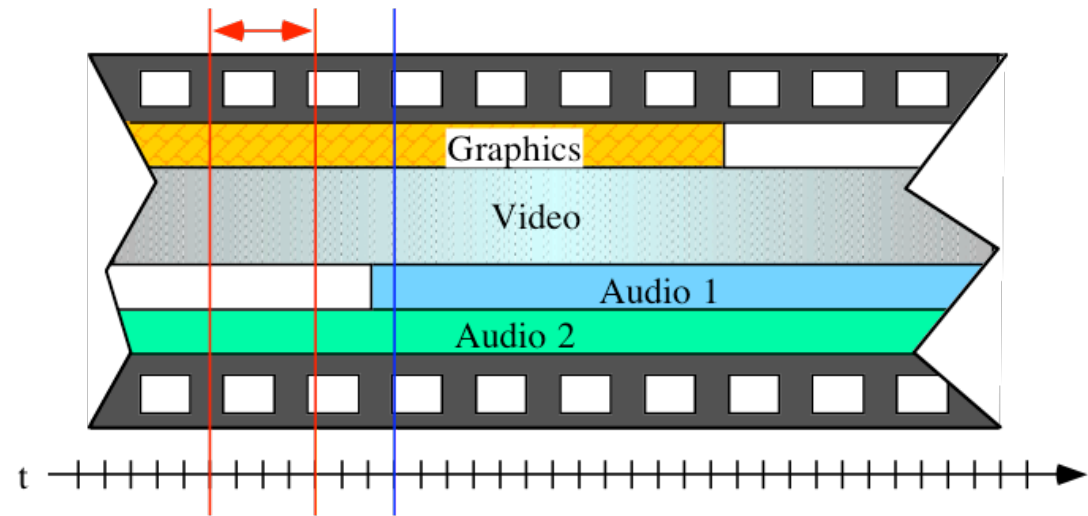
- Druck-Ausgabe

- Farbanpassung an Drucker
- Farbseparationen

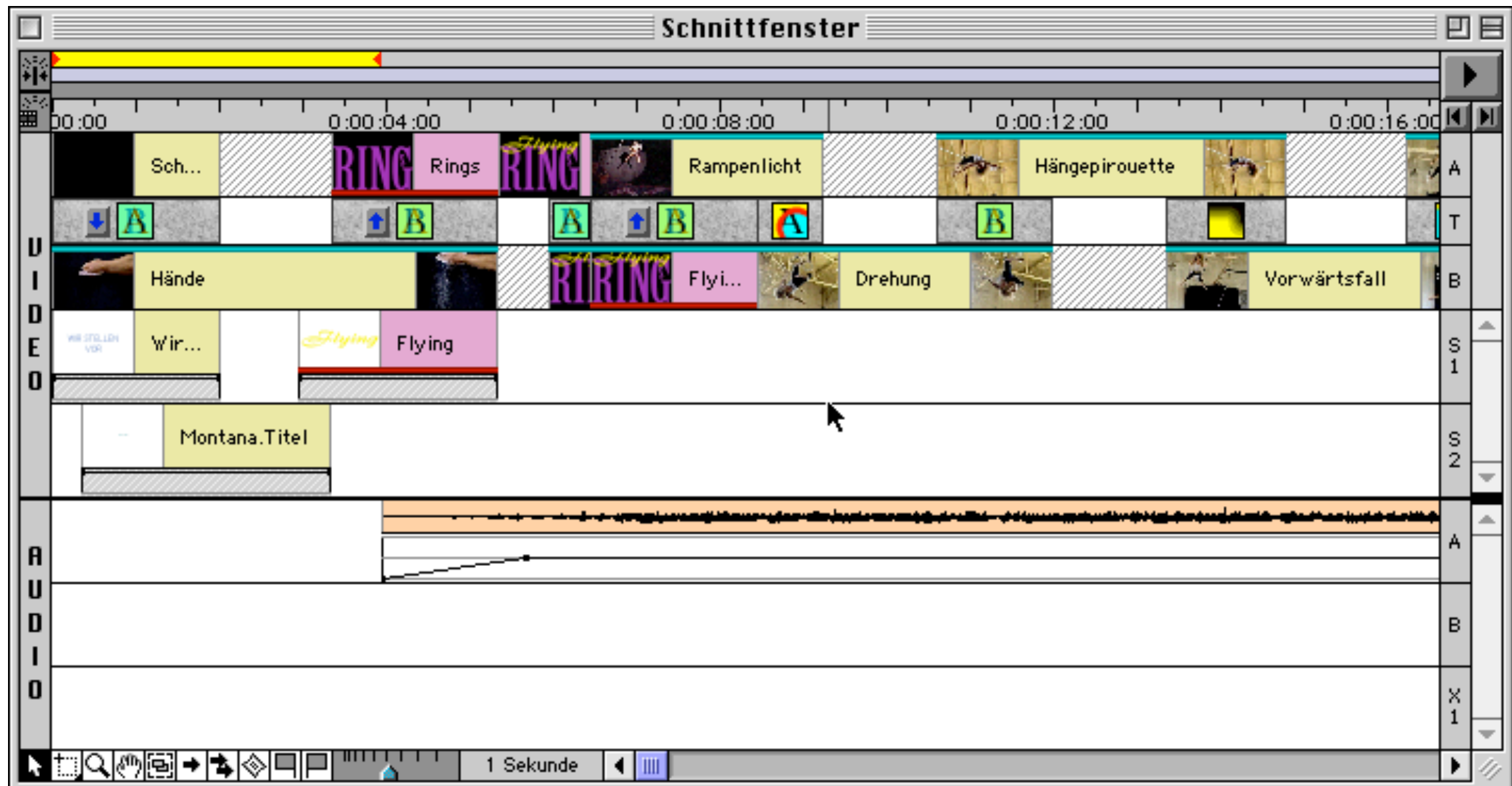


## 8.2.6 Digitaler Videoschnitt

- Adobe Premiere, Apple Final Cut Pro
- Projekt
  - technische Parameter
  - Menge von Komponenten: Audio, Video, Titel
  - Digitalisierung
  - Import
- Medienströme bearbeiten
  - editieren
  - Präsentationseigenschaften ändern
  - Filter
  - einfache Effekte: Bewegung, Zoom, Drehung
- Produktion des Filmes
  - Movie-Datei in vielen Formaten
  - Edit Decision List für Schnittsysteme

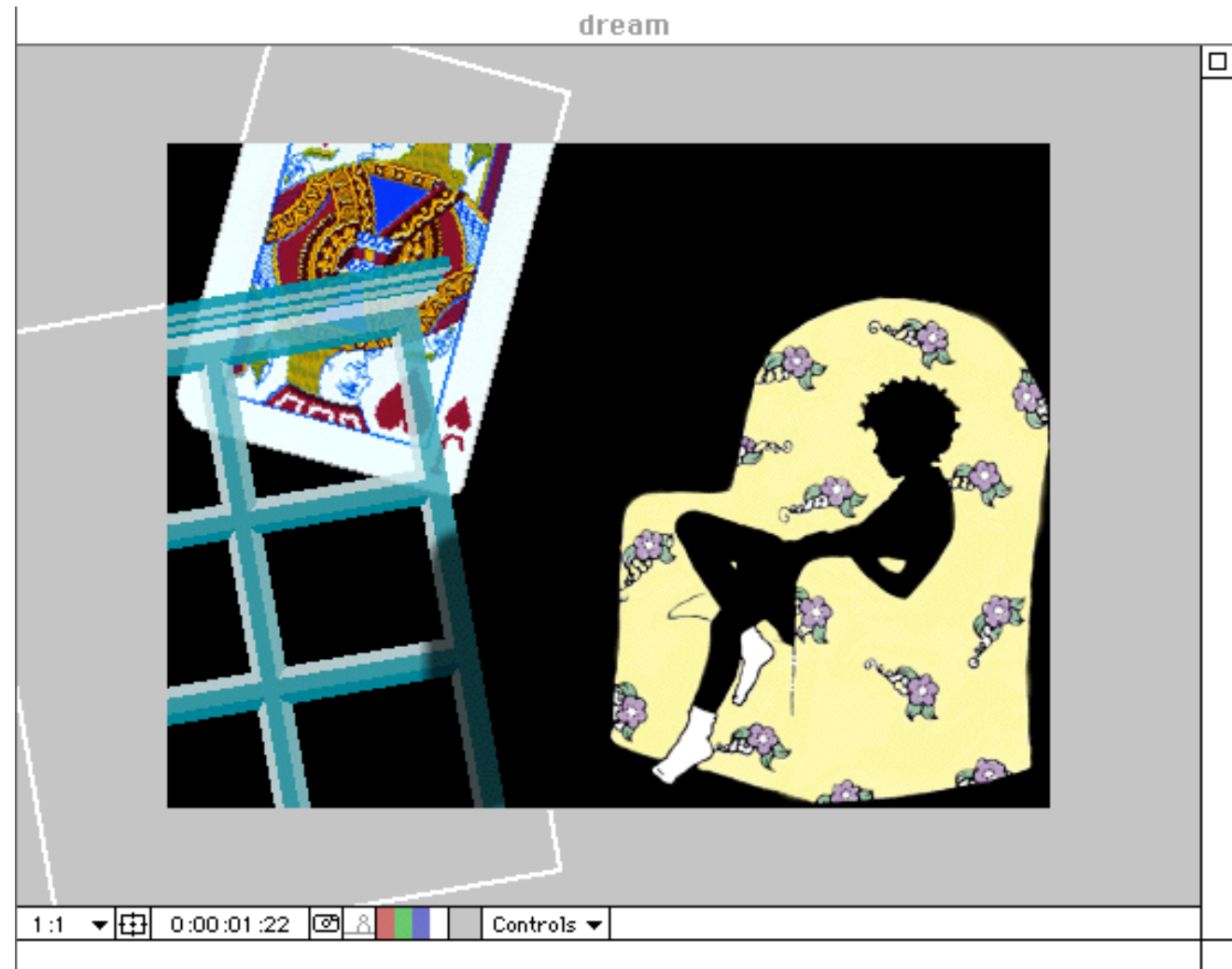


- Mehrere Spuren
  - zeitliche Anordnung
  - Übergänge gestalten



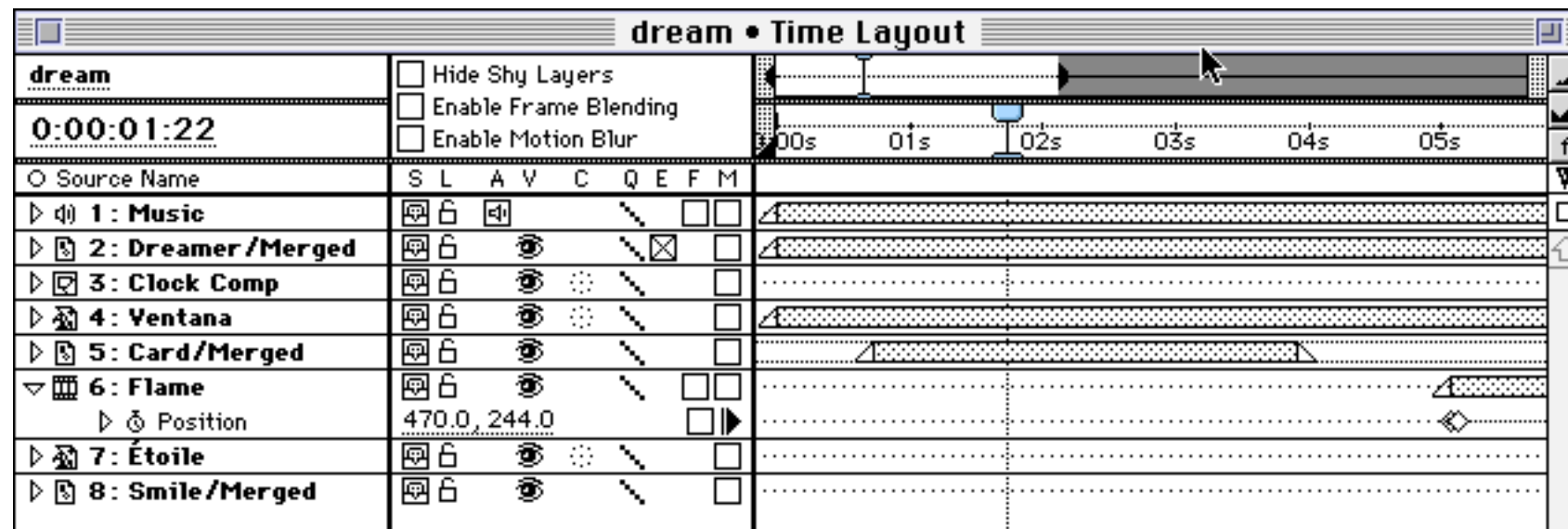
## Montage und Animation

- z.B. After Effects
- Bilder lernen Laufen
  - Animationen
  - Zeichnungen, Text
- Export in viele Filmformate
- Objekte anordnen
- Änderungen in der Zeit
  - Eigenschaften
  - Anordnung
- $f(\text{Objekt}, t)$





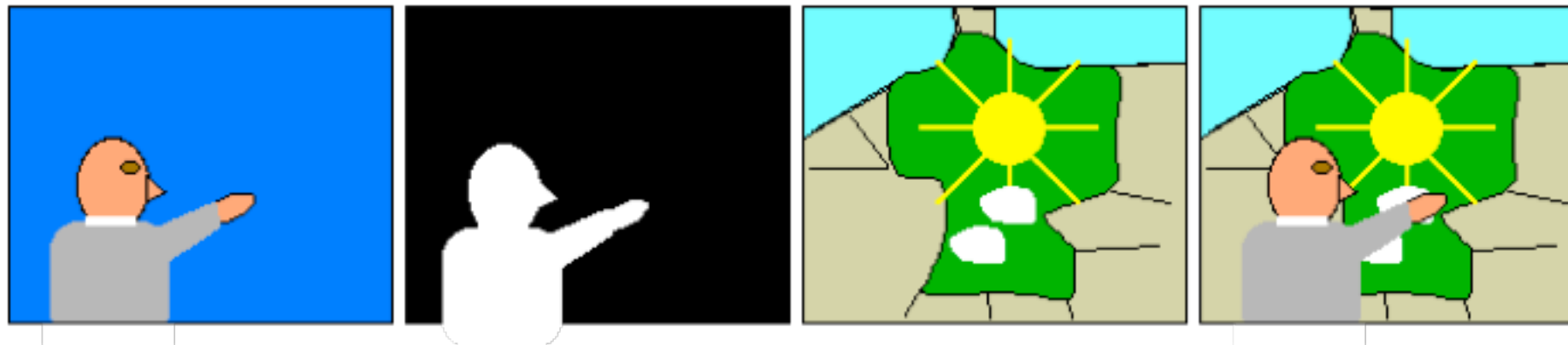
- Schichten (Layer) überlagern
  - Darstellung zeitlich begrenzt (In, Out)
  - Maske
- Zeitachsen und Keyframes



- Keyframes haben Schichteigenschaften als Attribute
  - Übergang zwischen Keyframes
    - => gradueller Übergang von A1 nach A2
  - lineare oder komplexe Übergangsfunktion

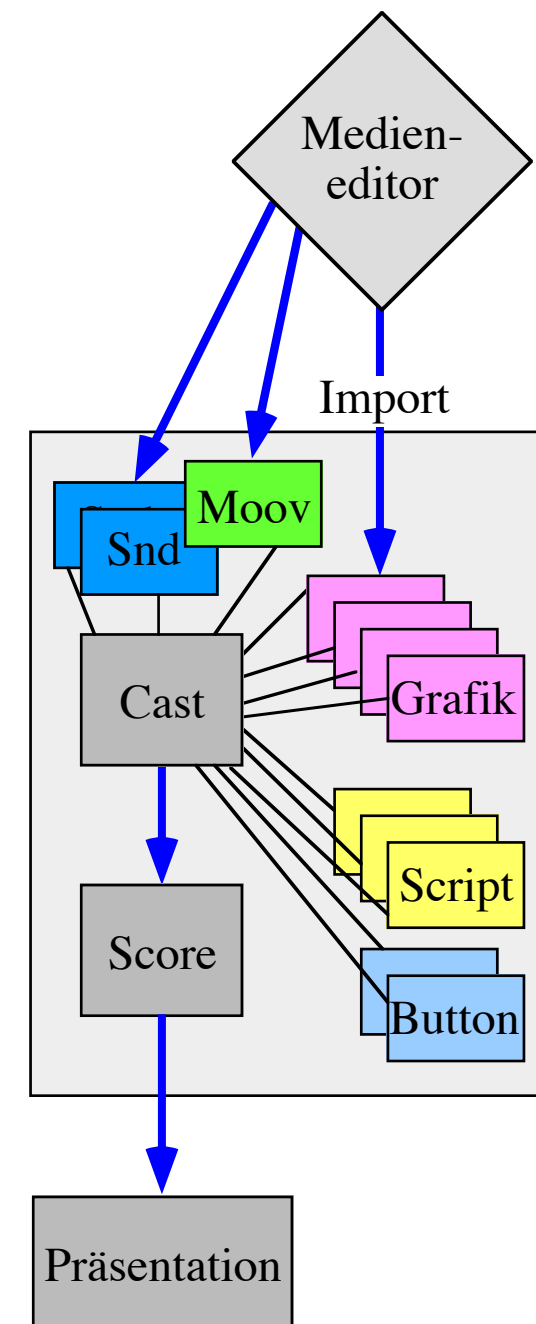
- Geometrische Attribute der Schichten in der Zeit
  - Größe
  - Bewegung entlang eines Pfades (Gerade, Bezier-Spline)
  - Rotation, Ankerpunkt
  - Bewegungsverzerrung
  - Durchsichtigkeit (Ein- und Ausblenden)
- Farb-Attribute
- Filter in der Zeit
  - Verschwimmen
  - Schatten
  - Textur
  - Erhebung, 3D, ...
- Überblenden zwischen Schichten
  - Transparenz
  - Überblendeffekte (Dissolve, Wipe, Vorhang, ...)

- Motion Pack
  - Identifizieren bewegter Elemente
  - Bewegungen glätten
  - Bewegungen 2. Ordnung
- Keying Pack
  - Wetterbericht
  - Teile des Originalbildes werden 'transparent'
  - einfache Keys: Farbe, Helligkeit
  - scharfe Grenzen
  - Wertebereich der Transparenz

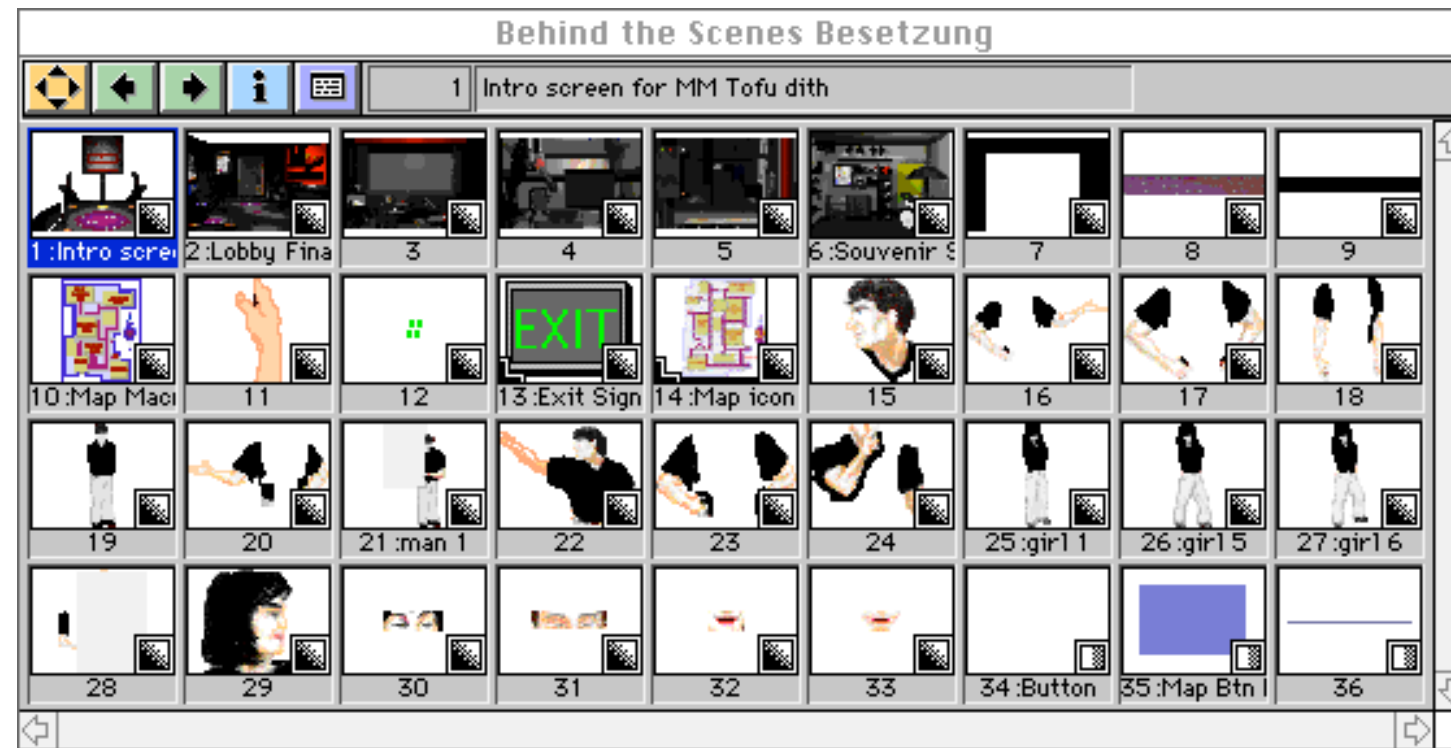


## MacroMedia/Adobe Director

- Präsentationen
  - Animationen (nicht extrem anspruchsvoll)
  - Abspielen von Video und Sound
  - ShockWave
- Grafische Elemente
  - Text, Grafik, Video, Sound
  - 16/32 bit Farben
  - 1,2,4,8 bit Farben mit verschiedenen Paletten
- Benutzungsschnittstelle
  - simple Navigation mit Knöpfen
- Begriffe
  - Darsteller (actors) und Besetzung (cast)
  - Drehbuch (score)
  - Programmiersprache Lingo (scripts)
  - Sprite - Behälter für actor



- Darsteller (actors) und Besetzung (cast)

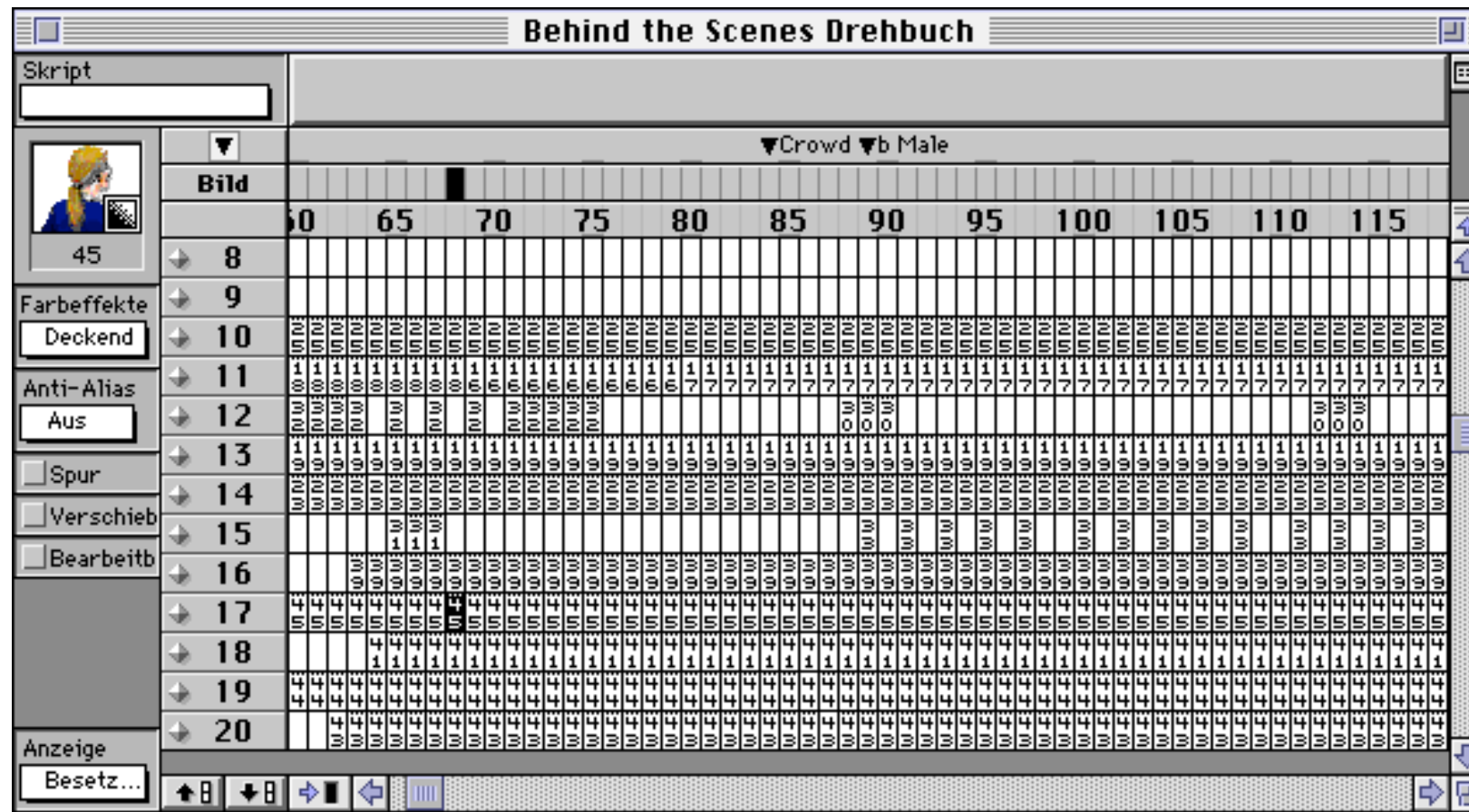


- Grafische Objekte (PICT, PICT-Sequenz, Bitmap)
- QuickTime Movie
- Sound
- Button, Menu, ...
- Script

- Zeichenwerkzeug integriert
  - Nachbearbeitung von importierten Bildern
  - Grafiken können auch Skript haben



- Drehbuch (score)



- Zeit horizontal (Frames)
- vertikal 'Spuren'
- Nummern aus cast
- Programmiersprache Lingo (scripte)
  - erweitertes HyperTalk
  - Objektorientierung als Nachgedanke

- Basis-Lingo

- **on** <Event> ... **end** <Event>
- **set** <Variable> **of** <Objekt> **to** <Wert>
- **set** <Variable> **=** <Wert>
- **if** <condition> **then** <clause> **else** <clause> **end if**
- **go** <Identifier>
- **repeat with** <Variable> **=** <Wert> **to** <Wert>
- ...
- end repeat**
- <Identifier> ruft Prozedur
- globale Variable

- Handler

- mouseUp, mouseDown, startMovie, stopMovie
- selbstdefiniert

```
on mouseUp
 go to frame "start"
end mouseUp
```



- Objektorientierung

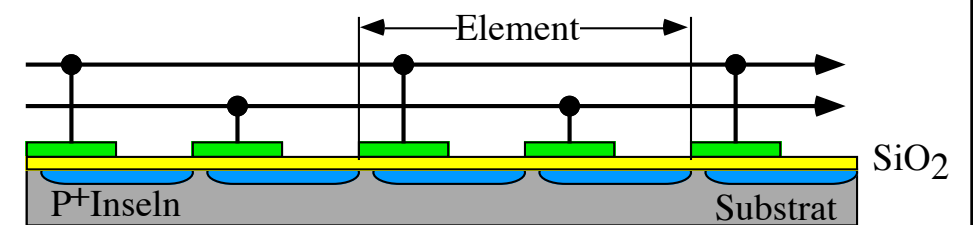
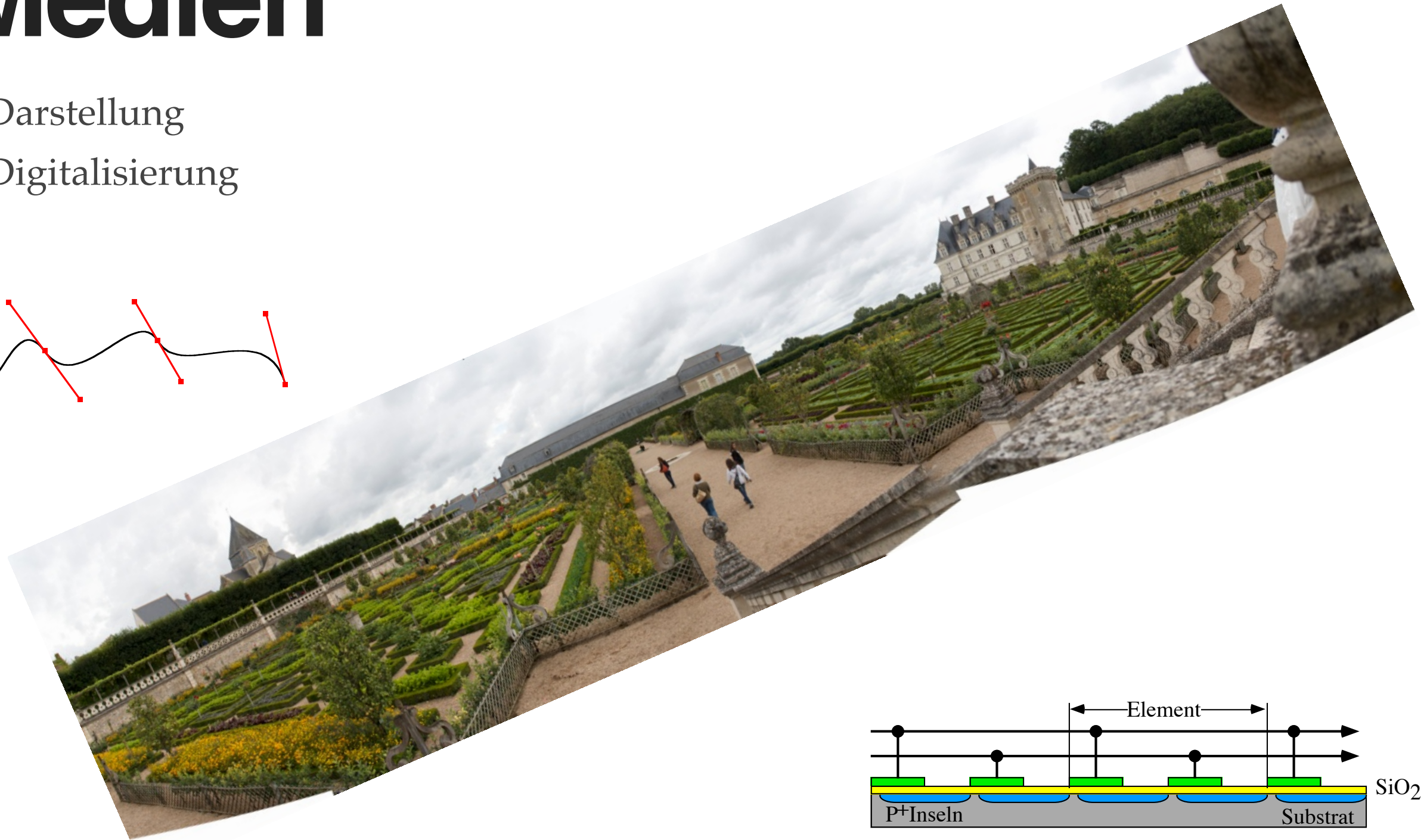
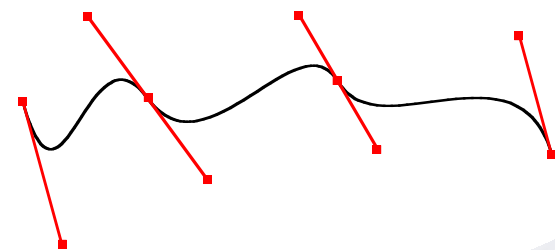
- Darstellerscript = Klasse
- property-variable = Instanzvariable
- ancestor-variable = Inheritance
- birth ist Konstruktor (on birth)

```
MECH , 76
global gMyPegBoard,gConstrainSprite,gIsRunning,gHandCursor,gSpeedCntrSprite
property PEG_columns,PEG_rows,myHPegs ,myVPegs,myAnimator,myButtons
on birth me
 set PEG_columns = 16 -- we have 16 columns of pegs
 ...
 -- build the vert peg list
 set myVPegs = [] -- empty list
 set count = 2
 setAt myVPegs,1,50 -- set first array elt
 repeat while count < PEG_rows
 set pos = 50 + ((count - 1) * 30)
 setAT myVPegs , count,pos
 set count = count + 1
 end repeat
 ...
 return me
end birth
```

## KAPITEL 7

# Medien

- Darstellung
- Digitalisierung



# Medien und Wahrnehmung

- Nutzlast (Bit/bit)

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

ASCII-Text 10 byte

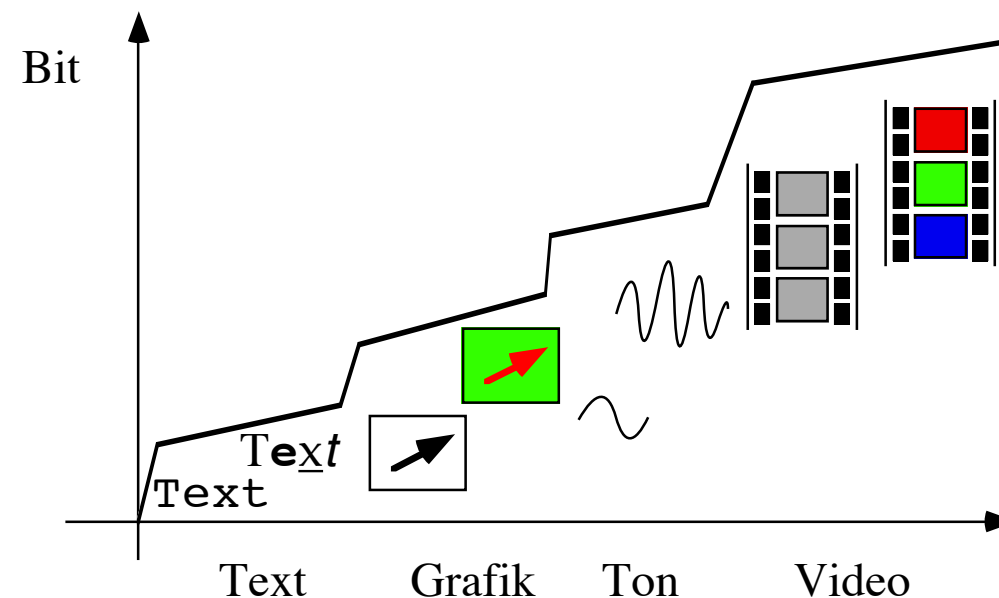
Bitmap 1000 Punkte \* 1 byte

Telefon 8.000 byte

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

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

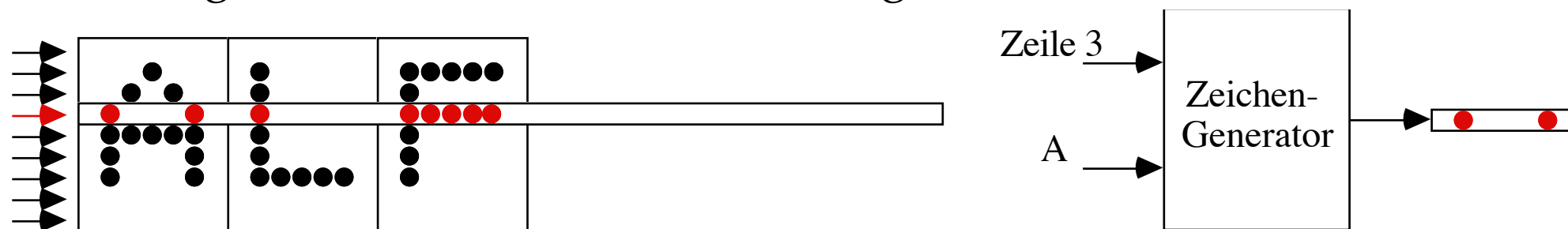
full HD 25 Bilder \* 1920 Spalten \* 1080 Zeilen \* 3 byte/Punkt = 155.520.000 byte



- 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: Spracherkennung
    - Sprache oder Musik
- Abschattungseffekte
  - in einem Medium
  - zwischen Medien

# Computergrafik

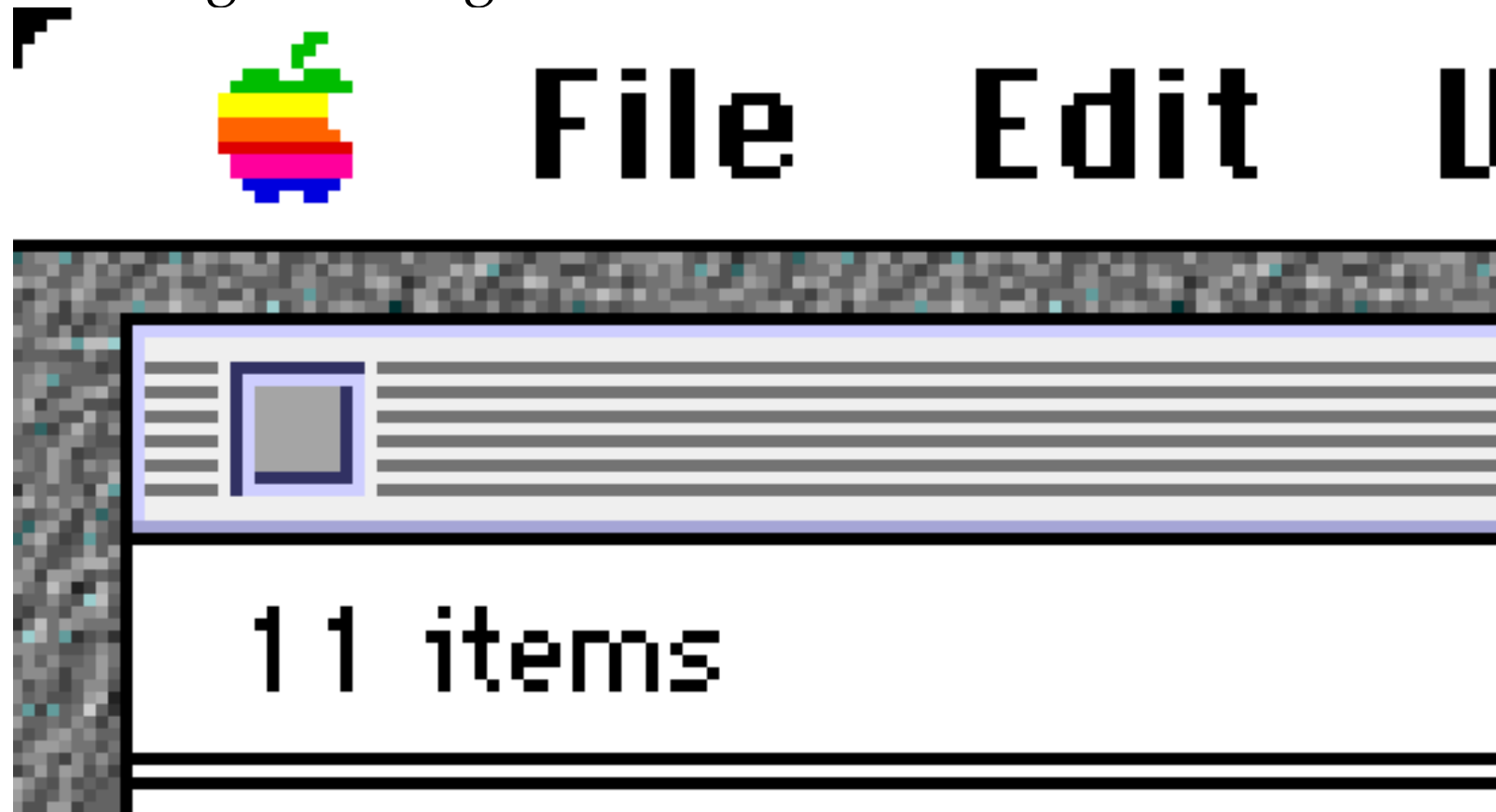
- Darstellung visueller Objekte
  - Buchstaben und Zahlen,
  - geometrische Objekte (Gerade, Kreis, Rechteck, ...)
  - Attribute (Farbe, Muster, Font, ...).
- Bildspeicher
  - Hauptspeicher oder im Adapter,
  - eventuell mehrere Ebenen (Farbe, Graustufen, räumliche Position).
- Buchstabenbildschirme
  - nur Buchstaben darstellbar
  - oft als Rasterbildschirm, aber Punkte nur in Gruppen ansprechbar
  - Zeichengenerator: ROM zur Abbildung der Buchstaben auf Raste



- Vektorgrafik ...

- Rasterbildschirm

- jeder Punkt einzeln ansprechbar
- uneingeschränkt grafikfähig



- Punkteanzahl typisch 1024\*768 bis 2560\*1600
- 72, 80 bis 240 Punkte / Zoll (dpi)
- 2560\*1600\*24 bit für 30" Farbmonitor -> 12.188.000 Byte
- Bildänderungsrate (Framerate, >25 Hz) und Bildwiederholrate (>70Hz)
- Transport über Monitorinterface: Pixeldaten\*Framerate

## Text

- Zeichensatz

- 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

- 5 kyrillisch, 6 arabisch, 7 griechisch und 8 hebräisch

- Unicode

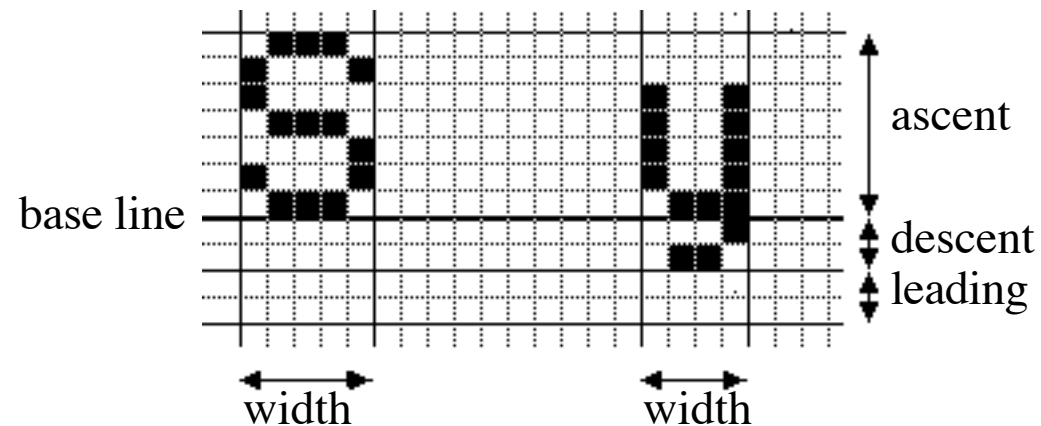
- Codes für **alle** Schriftzeichen der Welt

- 16 Bit / Zeichen

- 28.000 Codes für Ideographen (China, Korea, Japan)

- mehr Zeichen -> mehr Information / Zeichen: ae -> ä, ss -> ß

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





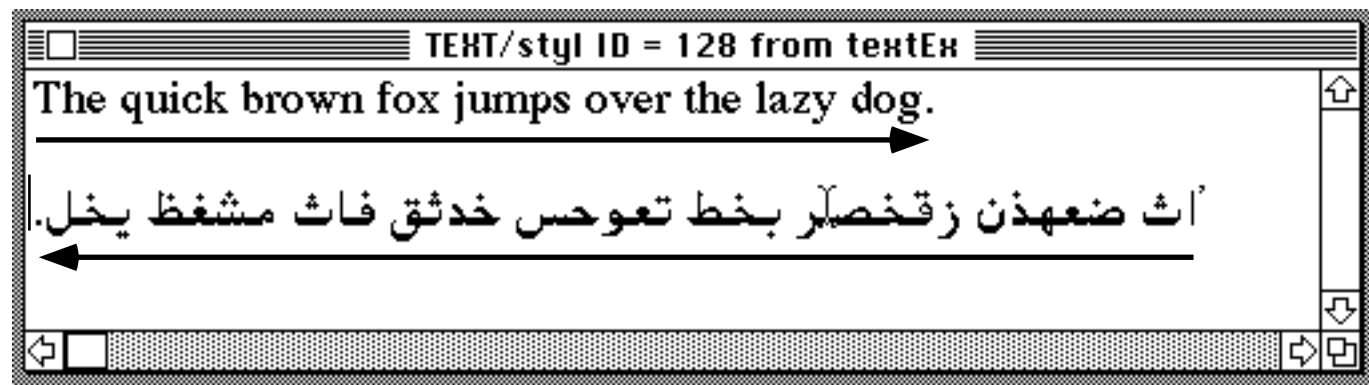
- Fontomania

- tausende verschiedene Zeichensätze
- Font-Beschreibungsalgorithmen siehe Kapitel 3
- Times New Roman
- Helvetica
- Palatino

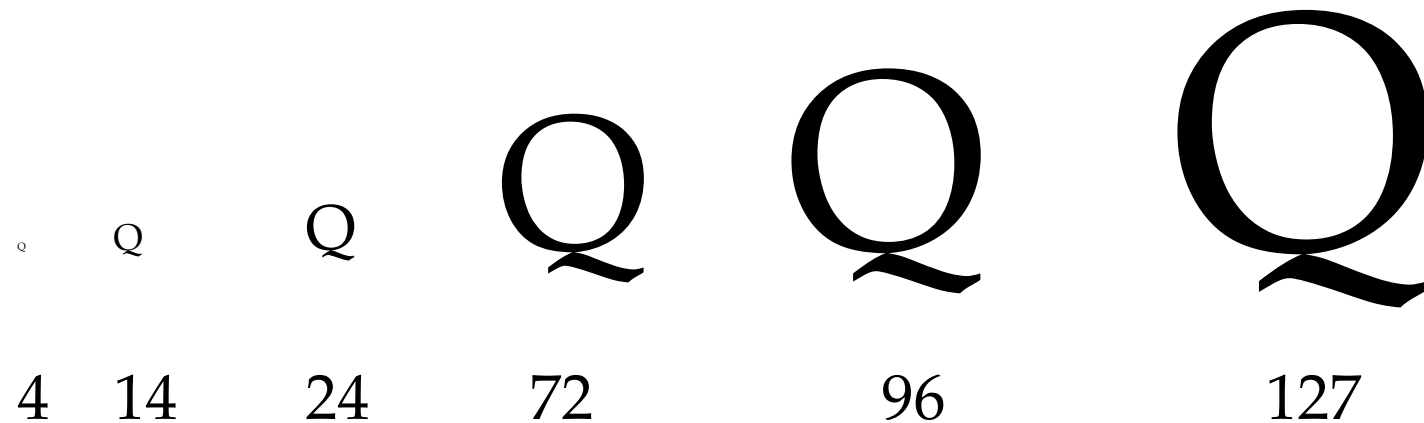
- *Zapfino*

- nicht-lateinische Schriften

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



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



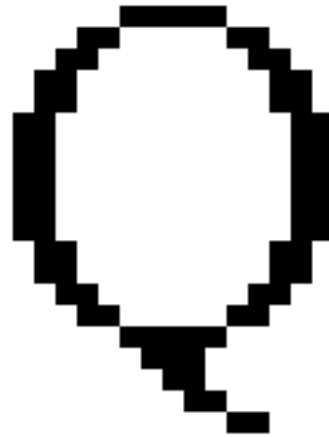
- und Formen

- Zeichensätze als Raster und Outline im System

Font - Book Antiqua [/Users/kf/Library/Fonts/Book Antiqua]																											
0000	---	0002	---	---	---	---	---	---	0009	000A	---	---	000D	---	---	---	---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F	0030	0031	0032	0033		
							!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3		
0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D		
4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M		
004E	004F	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F	0060	0061	0062	0063	0064	0065	0066	0067		
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	`	a	b	c	d	e	f	g		
0068	0069	006A	006B	006C	006D	006E	006F	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	---	00C4	00C5		
h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~		Ä	Å		
00C7	00C9	00D1	00D6	00DC	00E1	00E0	00E2	00E4	00E3	00E5	00E7	00E9	00E8	00EA	00EB	00ED	00EC	00EE	00EF	00F1	00F3	00F2	00F4	00F6	00F5		
Ç	É	Ñ	Ö	Ü	á	à	â	ä	ã	å	ç	é	è	ê	ë	í	ì	î	ï	ñ	ó	ò	ô	ö	õ		
00FA	00F9	00FB	00FC	2020	00B0	00A2	00A3	00A7	2022	00B6	00DF	00AE	00A9	2122	00B4	00A8	2260	00C6	00D8	221E	00B1	2264	2265	00A5	00B5		
ú	ù	û	ü	†	°	¢	£	§	•	¶	ß	®	©	™	´	¨	≠	Æ	Ø	∞	±	≤	≥	¥	μ		
2202	2211	220F	03C0	222B	00AA	00BA	03A9	00E6	00F8	00BF	00A1	00AC	221A	0192	2248	2206	00AB	00BB	2026	00A0	00C0	00C3	00D5	0152	0153		
∂	Σ	Π	π	∫	ª	º	Ω	æ	ø	¿	¡	¬	√	ƒ	≈	Δ	«	»	...		À	Ã	Õ	Œ	œ		
2013	2014	201C	201D	2018	2019	00F7	25CA	00FF	0178	2044	20AC	2039	203A	FB01	FB02	2021	00B7	201A	201E	2030	00C2	00CA	00C1	00CB	00C8		
-	—	“	”	‘	’	÷	◊	ÿ	Ÿ	/	€	<	>	fi	fl	‡	·	,	„	‰	Â	Ê	Á	Ë	È		
00CD	00CE	00CF	00CC	00D3	00D4	F8FF	00D2	00DA	00DB	00D9	0131	02C6	02DC	00AF	02D8	02D9	02DA	00B8	02DD	02DB	02C7	---	---	---	---		
Í	Î	Ï	Ì	Ó	Ô		Ò	Ú	Û	Ü	₁	²	³	⁴	⁵	⁶	⁷	⁸	⁹	ˆ	˜	˘	˙	˚	π		
---	00A4	00A6	00AD	00B2	00B3	00B9	00BC	00BD	00BE	00D0	00D7	00DD	00DE	00F0	00FD	00FE	0160	0161	02C9	2126	2219	F001	F002				
4	⊠		-	2	3	1	¼	½	¾	Ð	×	Ý	Þ	ð	ý	þ	Š	š	-	Ω	.	fi	fl				

Size: [dropdown] | [font icon] | [font name] | Pages mode: [dropdown] | MacOS Roman | Glyph: A [0041] | Selected: 1 / 245

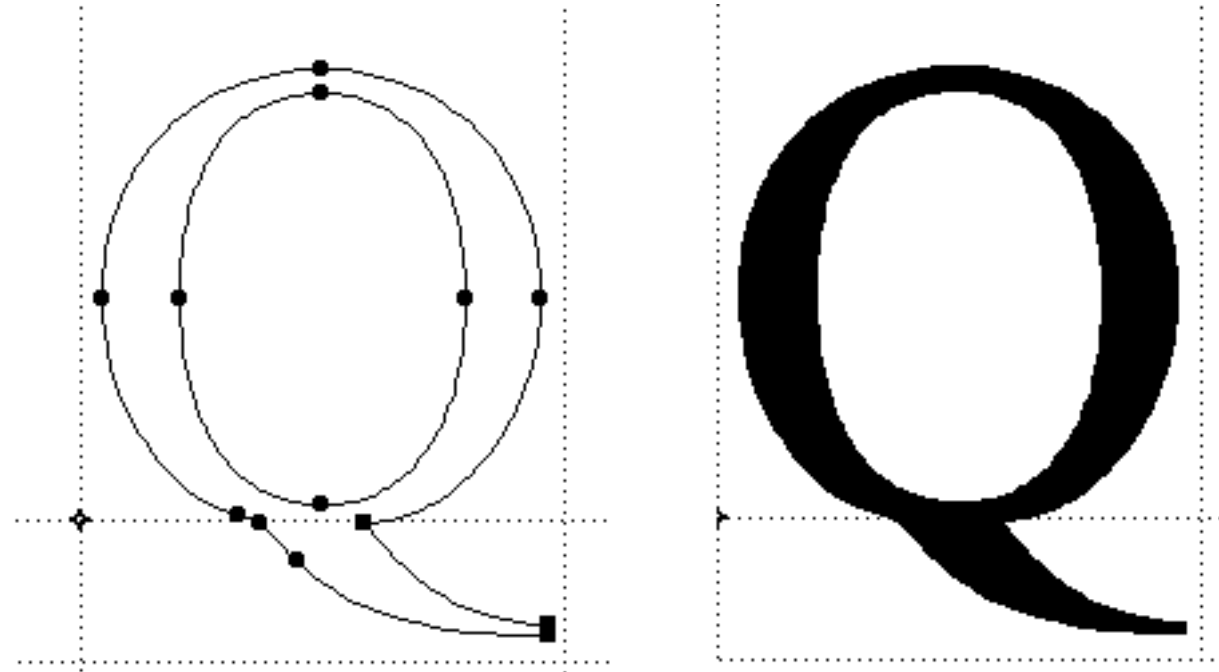
- Werden bei Bedarf in den Speicher geladen.
- Größe 24 Punkt (Vergrößerung \*8)



Q

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

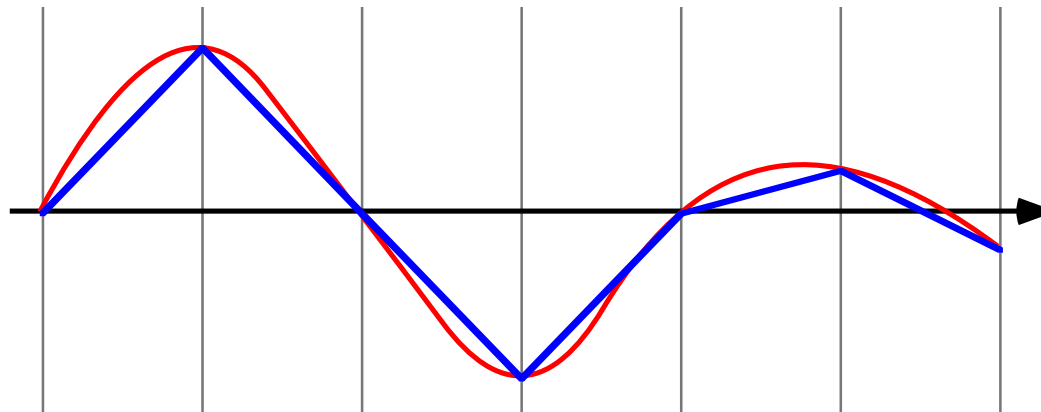
- Kurven zur Beschreibung von Fonts
- Die Umrisse der Zeichen werden als Kurvenzug angegeben



- 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

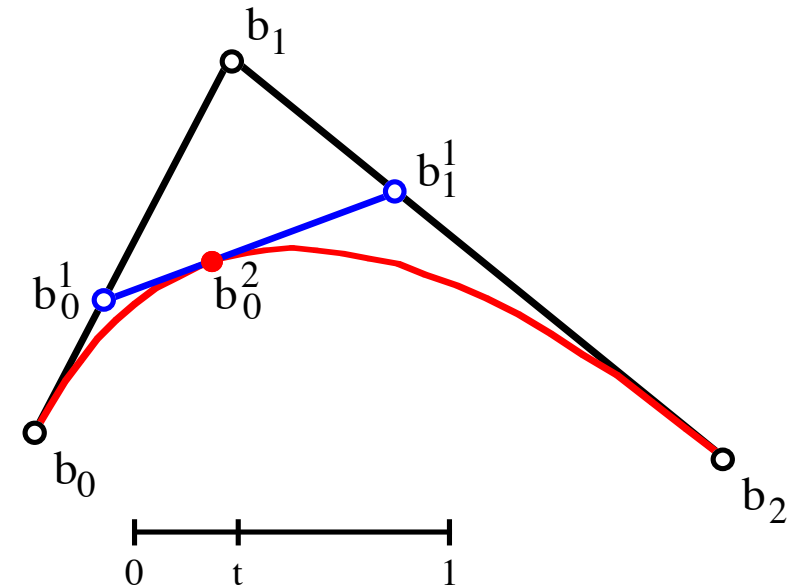
- Beispiel 2. Ordnung

gegeben  $b_0, b_1, b_2$

$$b_{1,0}(t) = (1 - t) b_0 + t b_1$$

$$b_{1,1}(t) = (1 - t) b_1 + t b_2$$

$$b_{2,0}(t) = (1 - t) b_{1,0}(t) + t b_{1,1}(t)$$



- Algorithmus von de Casteljau

gegeben  $b_0, b_1, \dots, b_n$

$$b_i^r(t) = (1 - t) b_i^{r-1}(t) + t b_{i+1}^{r-1}(t) \quad r = 1, \dots, n; i = 0, \dots, n-r$$

$$B_i^n = \binom{n}{i} t^i (1 - t)^{n-i} \quad b_0^n = \sum_{j=0}^n b_j B_j^n(t)$$

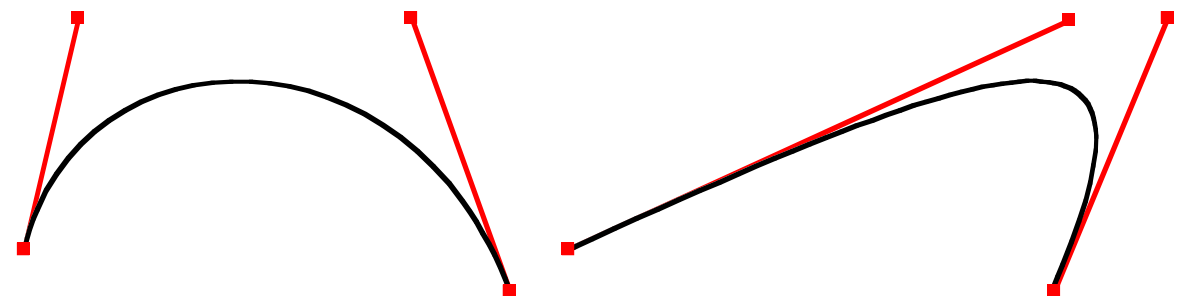
- Explizite Darstellung mit Bernsteinpolynomen

- Bézier-Kurven 3. Ordnung

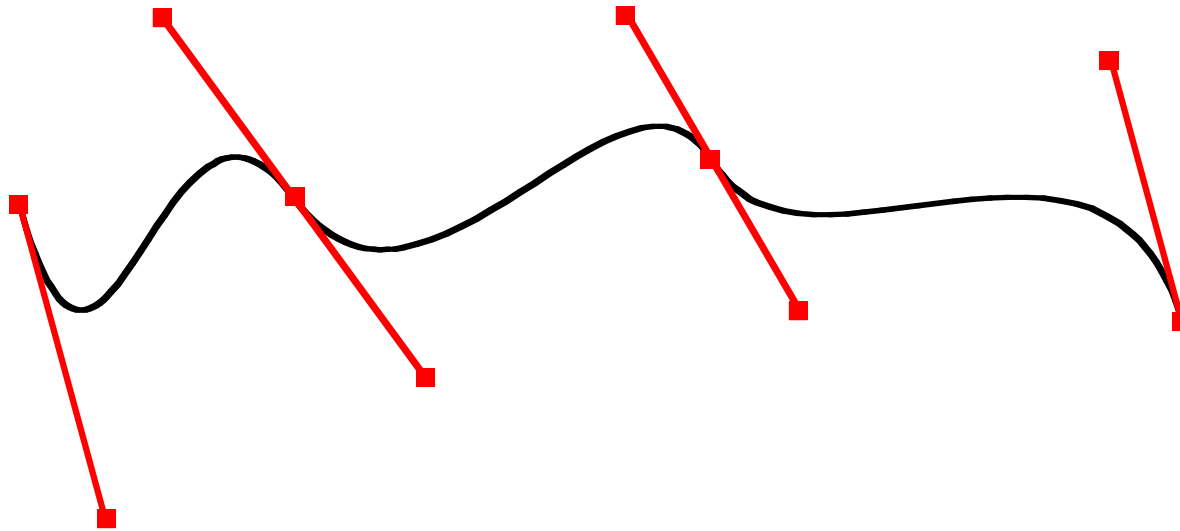
Kontrollpolygon durch vier Punkte:

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

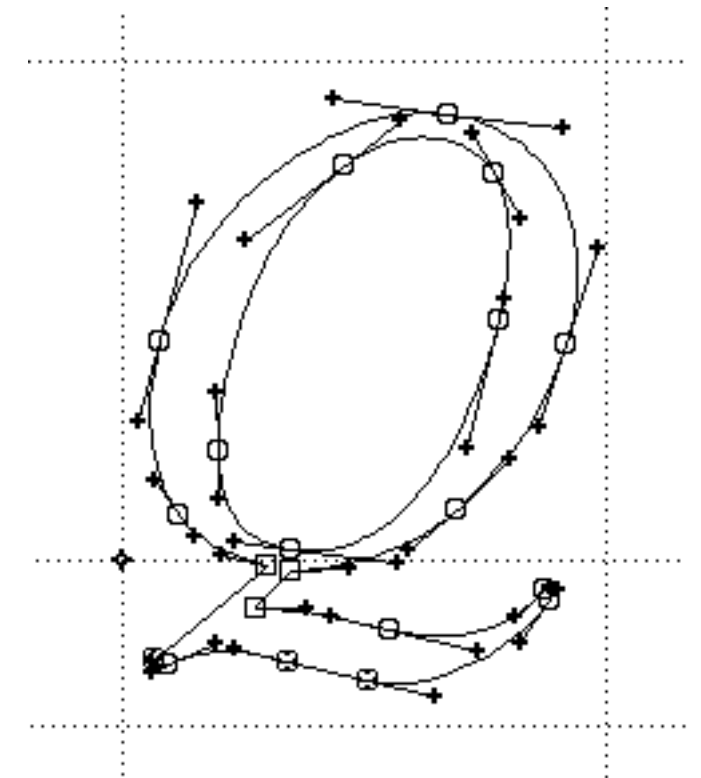
2 Kontrollpunkte ( $b_1, b_2$ )



- 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





## 3-D Grafik

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

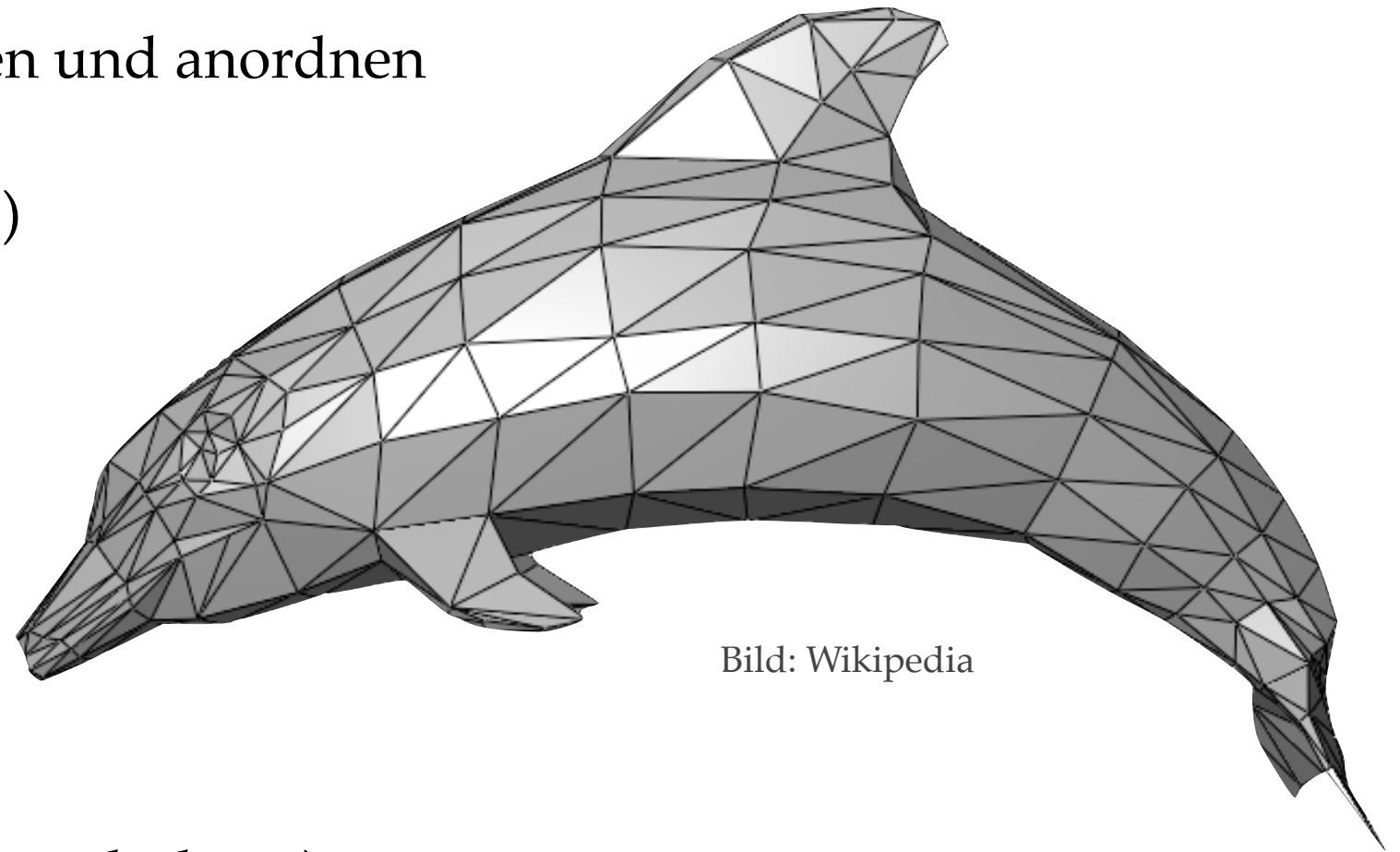
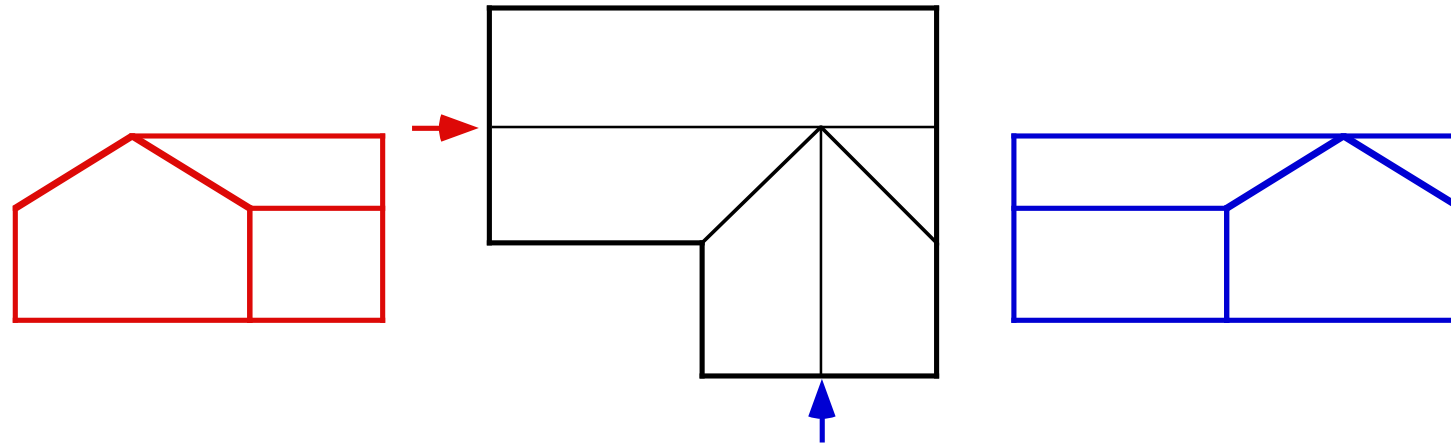
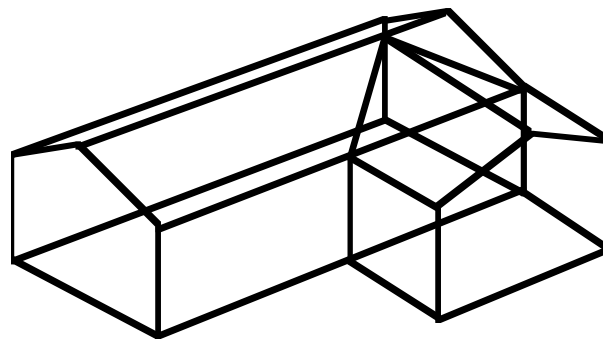


Bild: Wikipedia

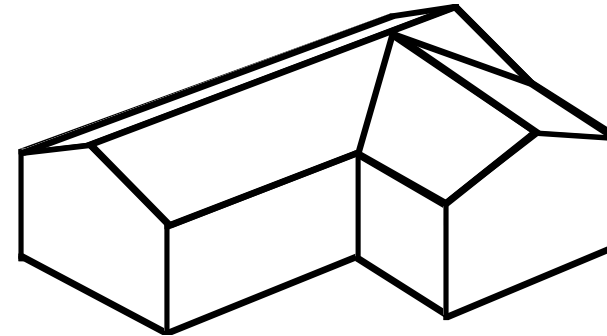
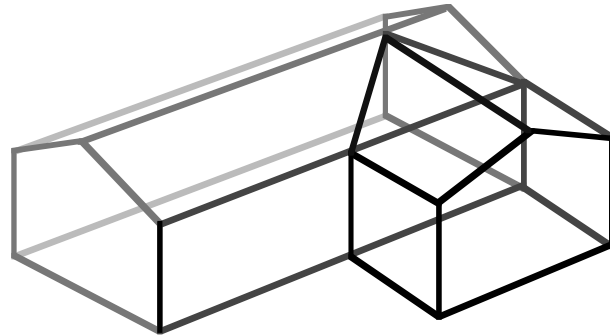
- Präsentation meist zweidimensional
  - Leinwand, Bildschirm, Papier
  - Projektion von 3-D Szenen auf 2-D Ebene
  - Tiefenhinweise gehen teilweise verloren
- Ansicht und Aufsicht



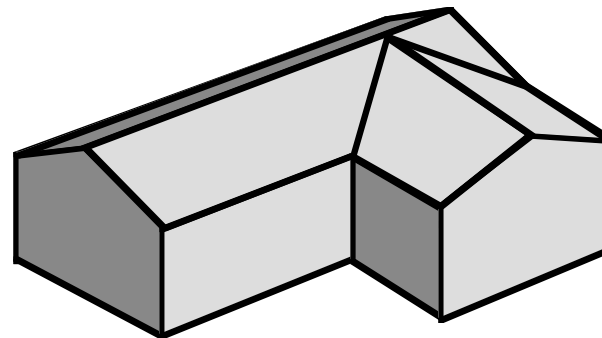
- Projektion und Drahtmodell
  - (fast) ohne Tiefeneindruck



- Depth-Cueing
  - Linien 'vorne' hervorheben



- Animation: Drehen um eine Achse
  - Entfernen verdeckter Linien
- Verbesserung der Darstellung
  - Füllen der Flächen

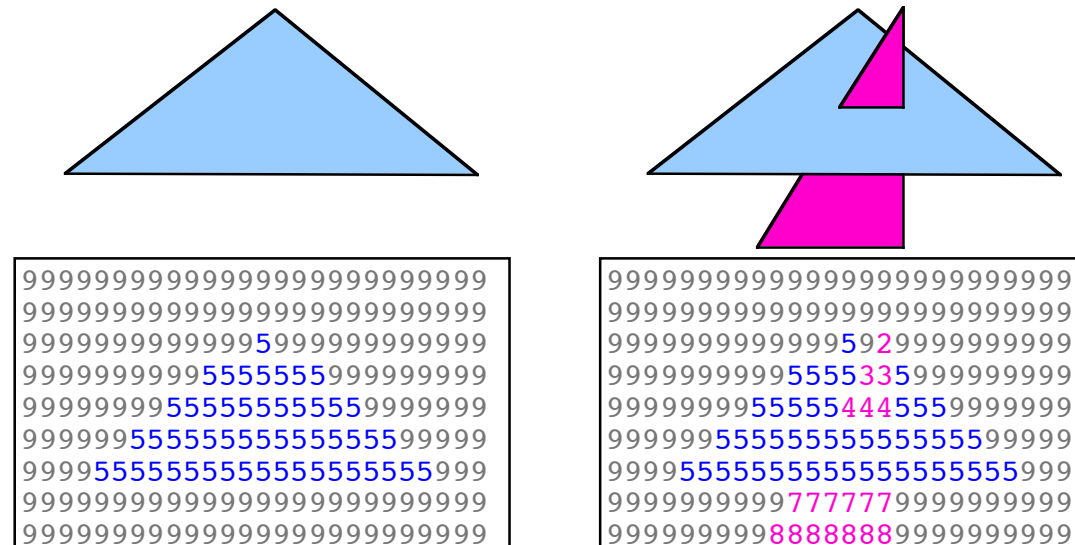


- Entfernen verdeckter Flächen

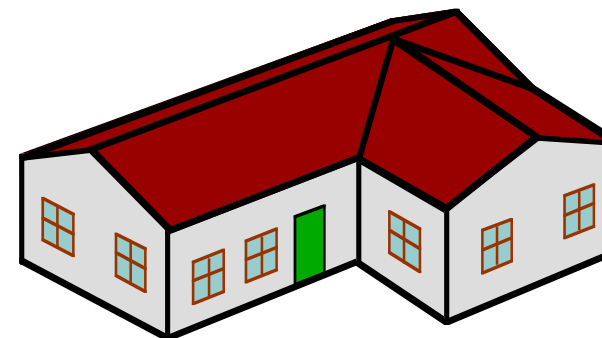
Algorithmus von E. Catmull

Tiefenpufferalgorithmus (z-buffer) Pixel = (R, G, B, Z)

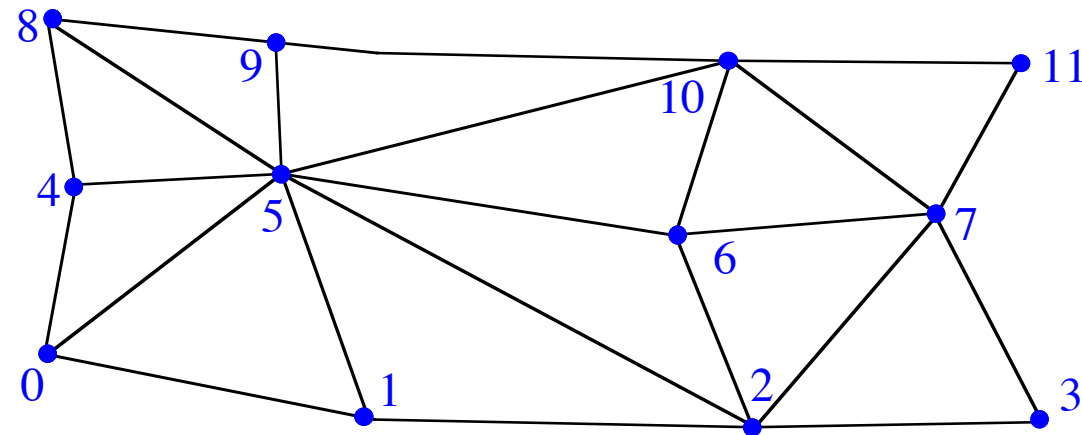
**IF** newpix.z < pixmap[x,y].z **THEN** pixmap[x,y] := newpix;



- Schattierungen simulieren Lichteinfall
- realistische Farben, Detail



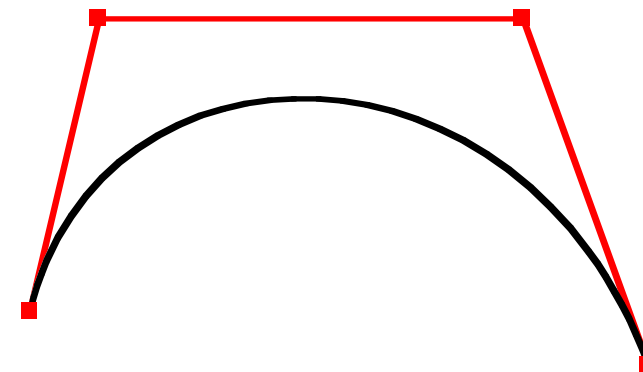
- TriGrid: Gruppe von Dreieck-Facetten



- 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

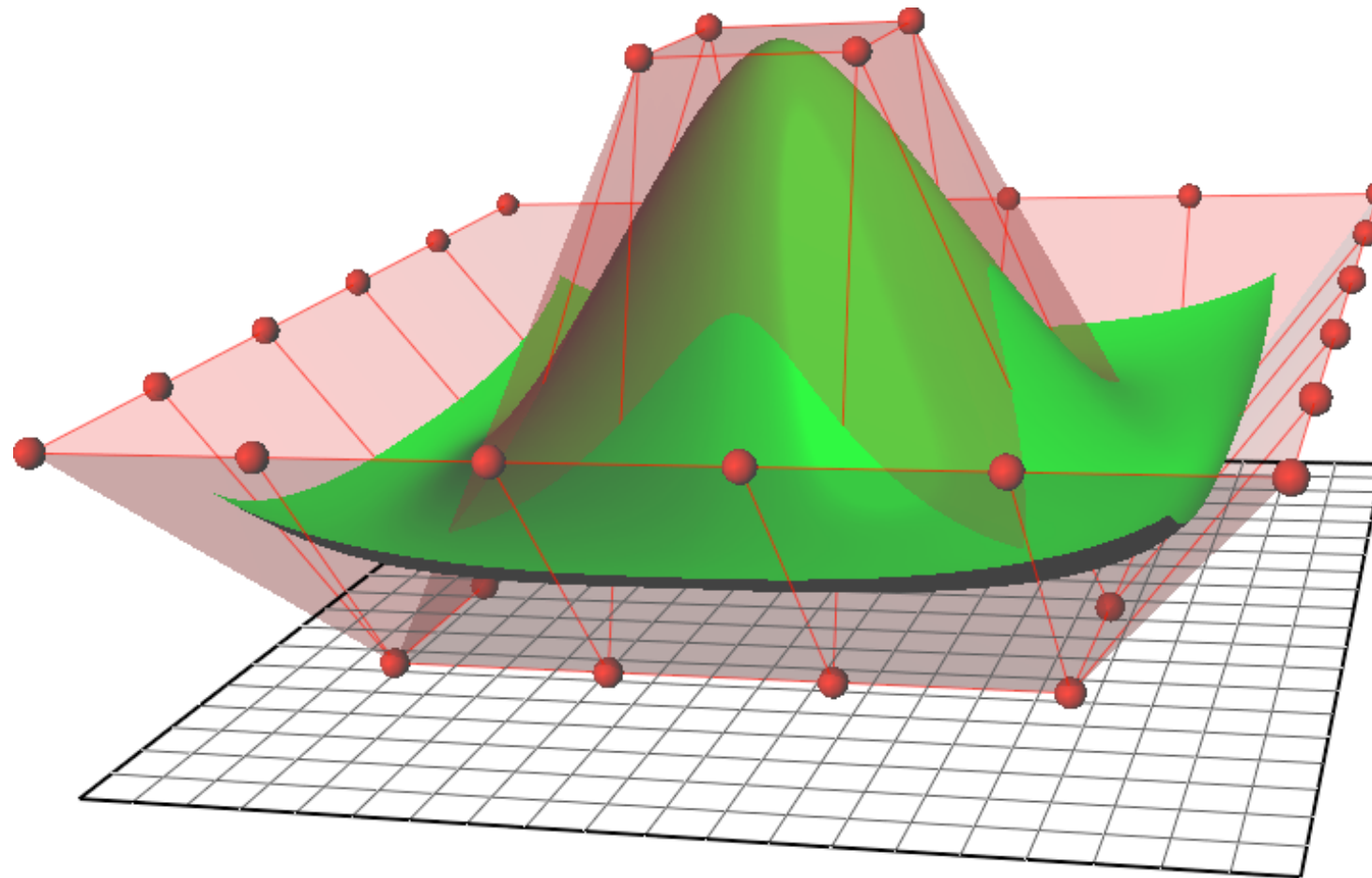
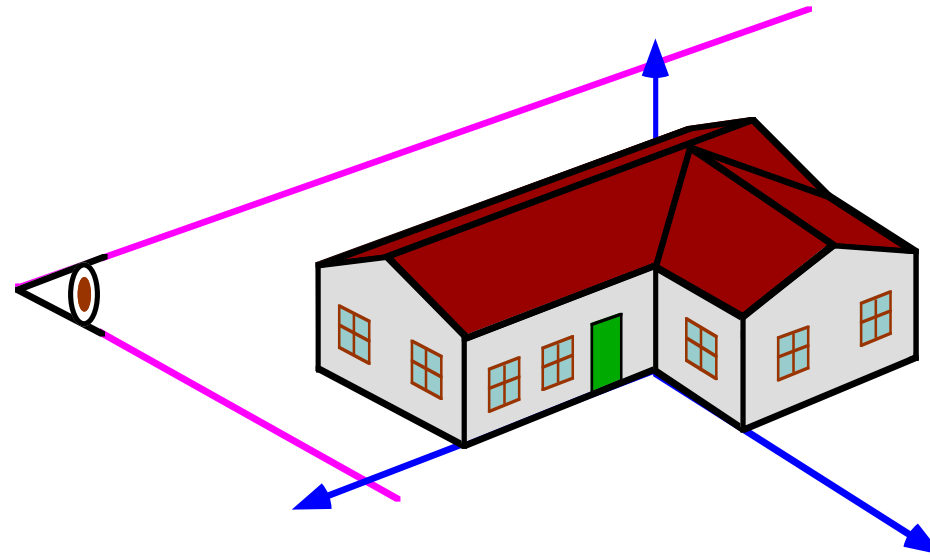
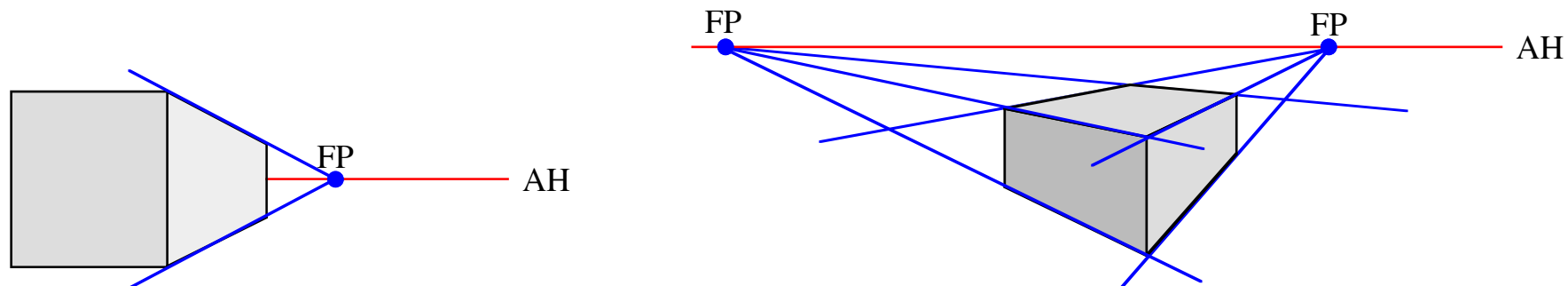


Bild: Wikipedia

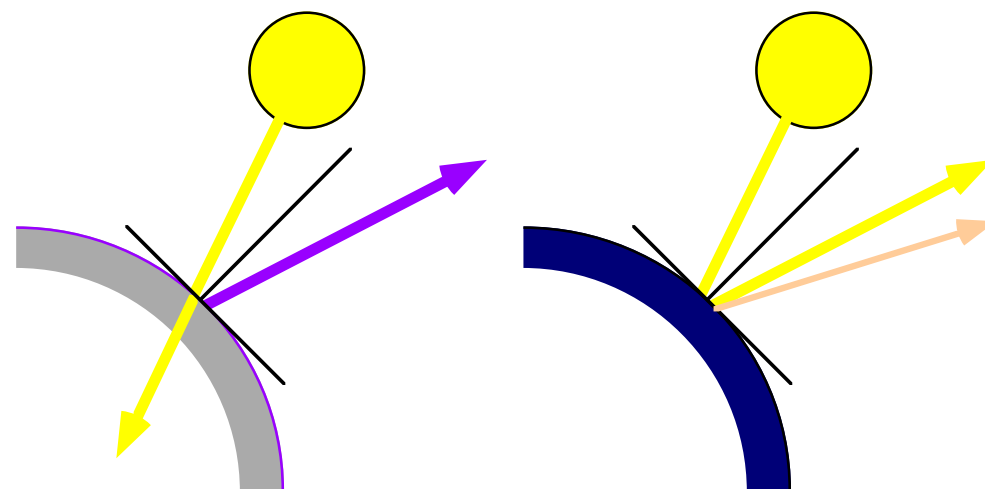
- Kamera: Betrachtungsort, Blickwinkel, Öffnungswinkel
- Perspektivische Projektion



- Fluchtpunkt(e)
- Maße nicht korrekt ablesbar

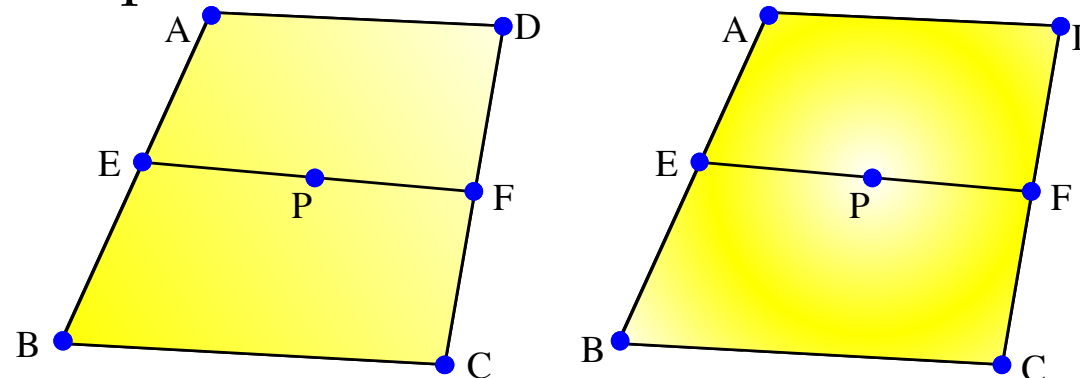


- Parallelprojektion
  - orthographische Projektion: Grundriß, Aufriß
  - schiefe (axonometrische) Projektion
  - isometrische Projektion
- Beleuchtung
  - Umgebungslicht (ambient), Punkt-Licht
  - diffuse Reflektion
  - Objekte werden von Lichtquelle angestrahlt
  - Licht wird teilweise reflektiert, teilweise durchgelassen
  - spiegelnde Reflektion
  - imitierendes Modell von Bui-Tung Phong, 1975

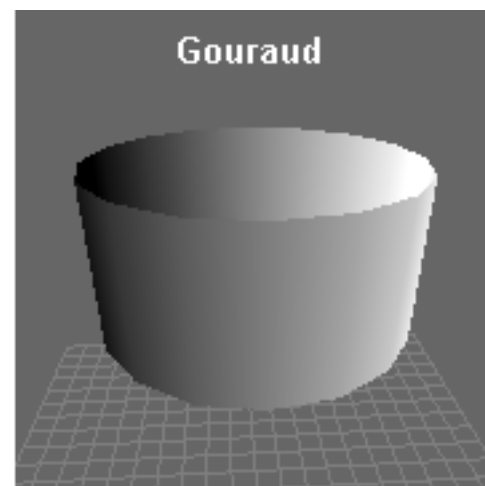
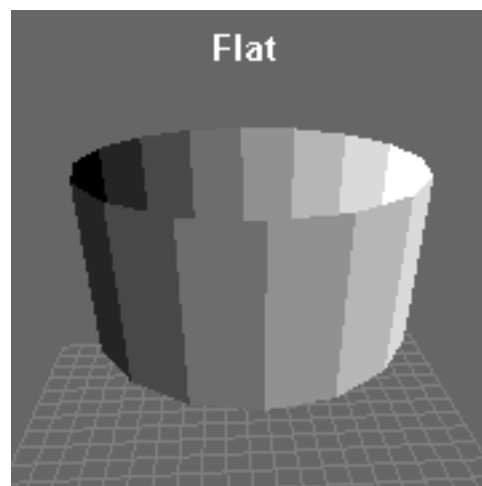




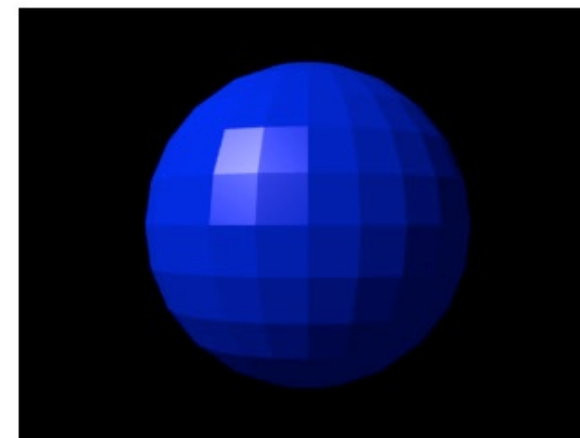
- Flat Shading: ein Farbwert pro Oberflächen-Facette
- Smooth Shading
  - Farbverlauf auf den Facetten
  - Gouraud-Shading: Interpolation zwischen Farbwerten an Eckpunkten



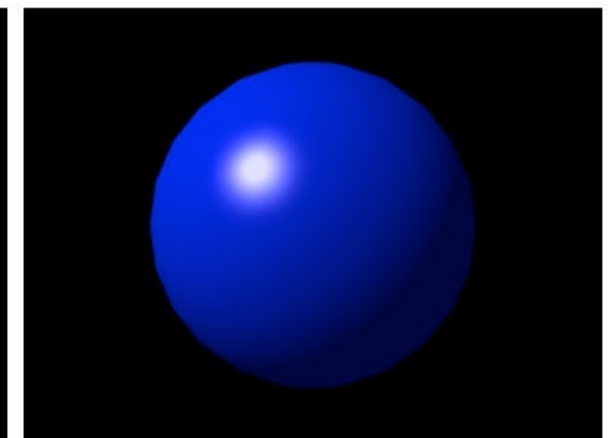
- Phong-Shading: individuelle Intensitätsberechnung für Flächenpixel



Bilder: Wikipedia

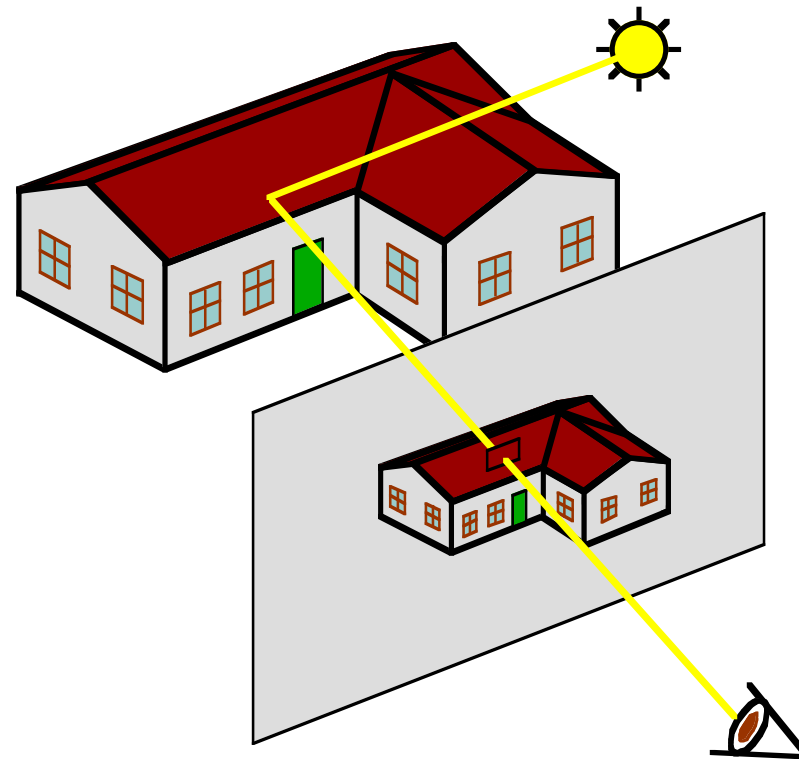


FLAT SHADING



PHONG SHADING

- Ray-Tracing
  - Reflektionen von Reflektionen, mehrfache Spiegelung
  - Strahlpfad berechnen

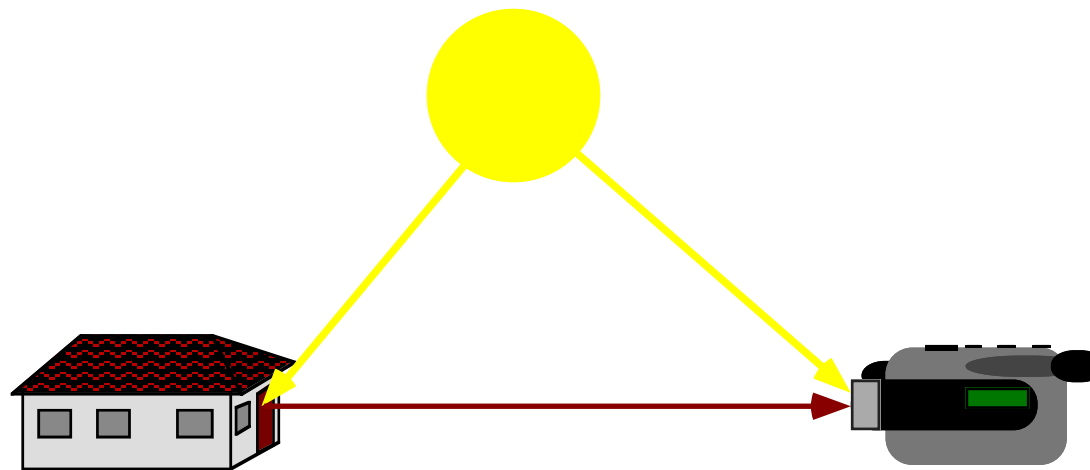


- vorwärts und rückwärts (vom Auge zum Licht)
- rechenintensiv
- Berechnung blickpunktabhängig

- 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

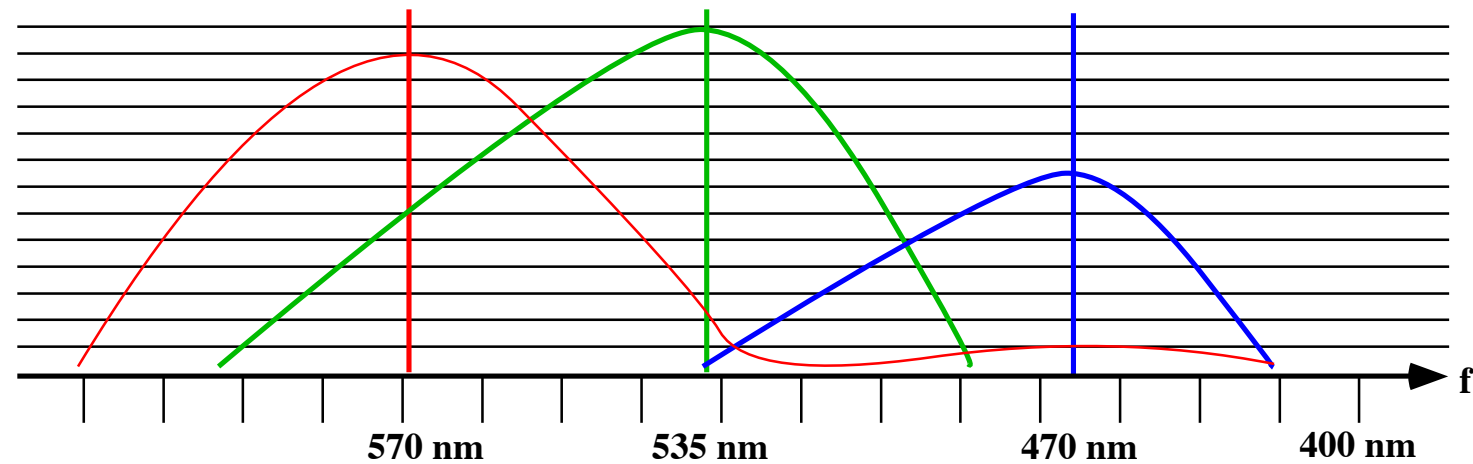
# Standbilder

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

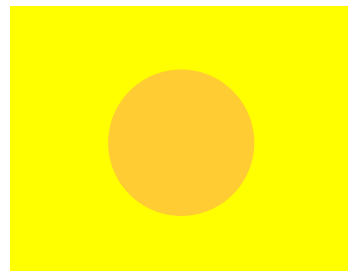


- $\text{Reflektiertes Licht} = \text{Licht} - \text{absorbiertes Licht} = \text{Oberflächenfarbe}$

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

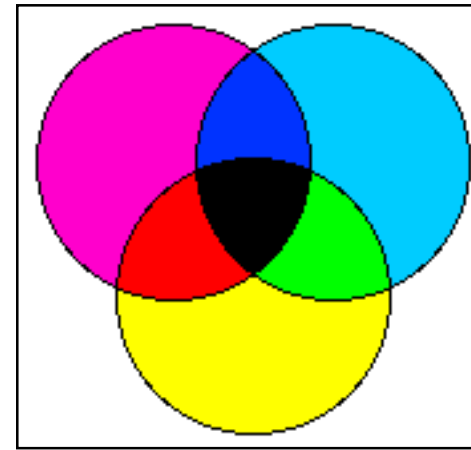
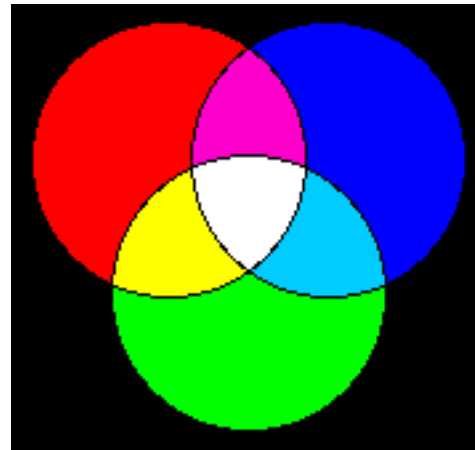


- 7 M Zapfenzellen für Farbe (570, 535, 455 nm)
- Mensch sieht bis zu 350.000 Farbnuancen

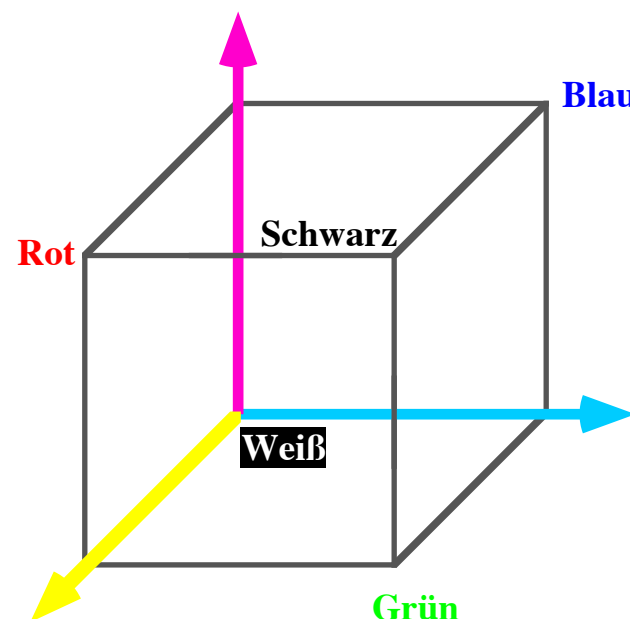
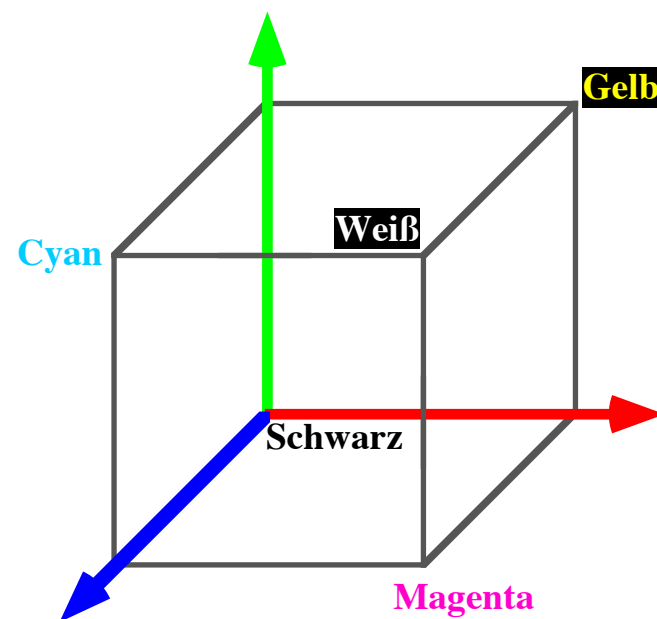


- Abschattung

- Farbmischen:      additiv      subtraktiv

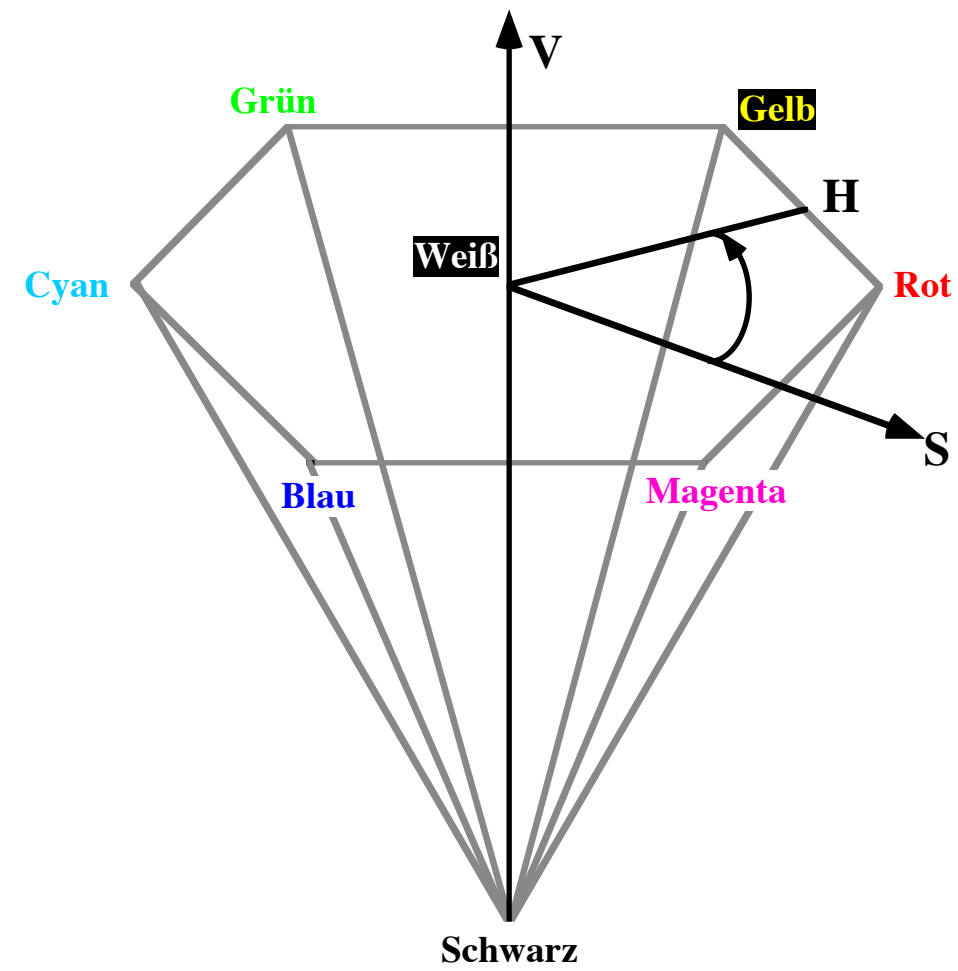


- Farbmodelle      RGB      CMY



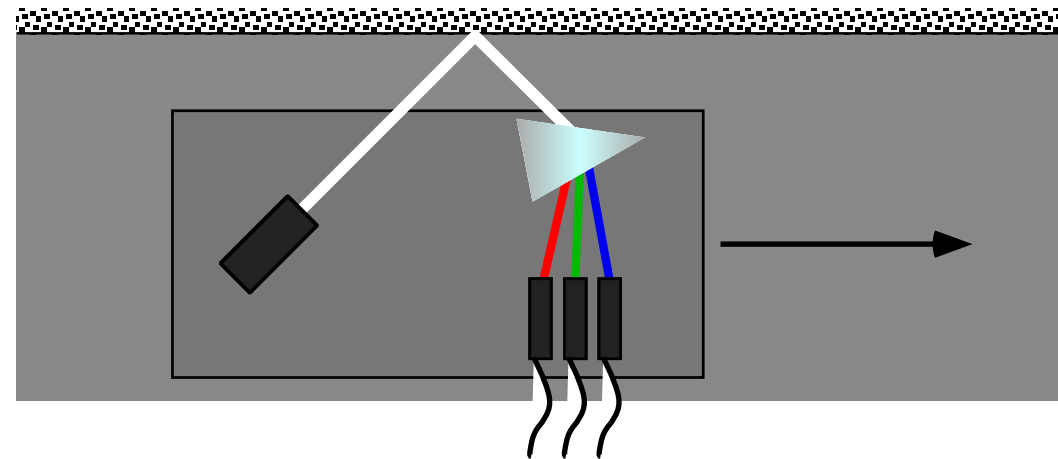
$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} Weiß \\ Weiß \\ Weiß \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- HSV (Ton, Sättigung, Helligkeit)



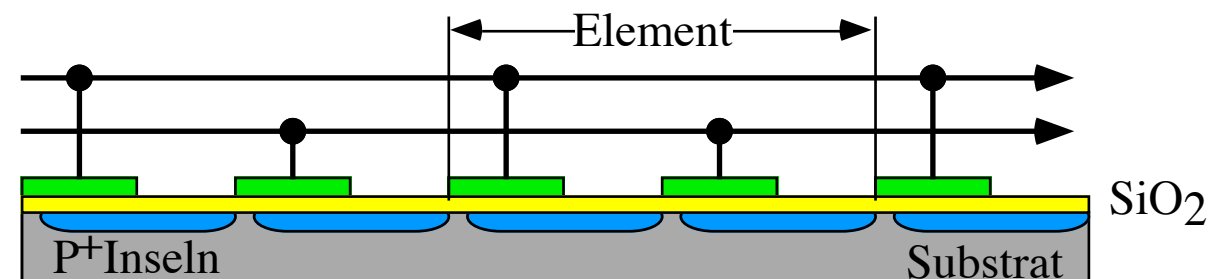
- Digitalisierung

- 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



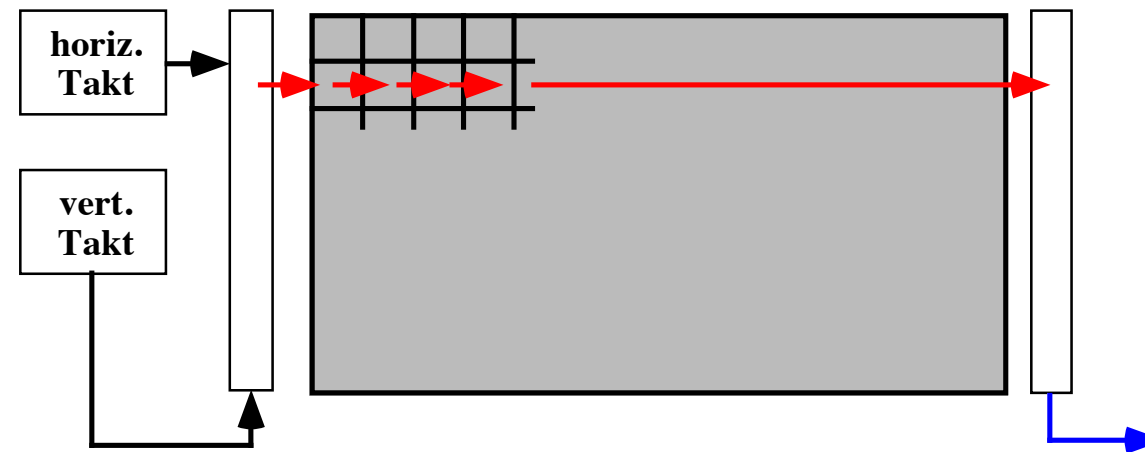
8, 16 oder 24 Bit für Farbe  
eventuell mit Farbpalette

- CCD-Zeile



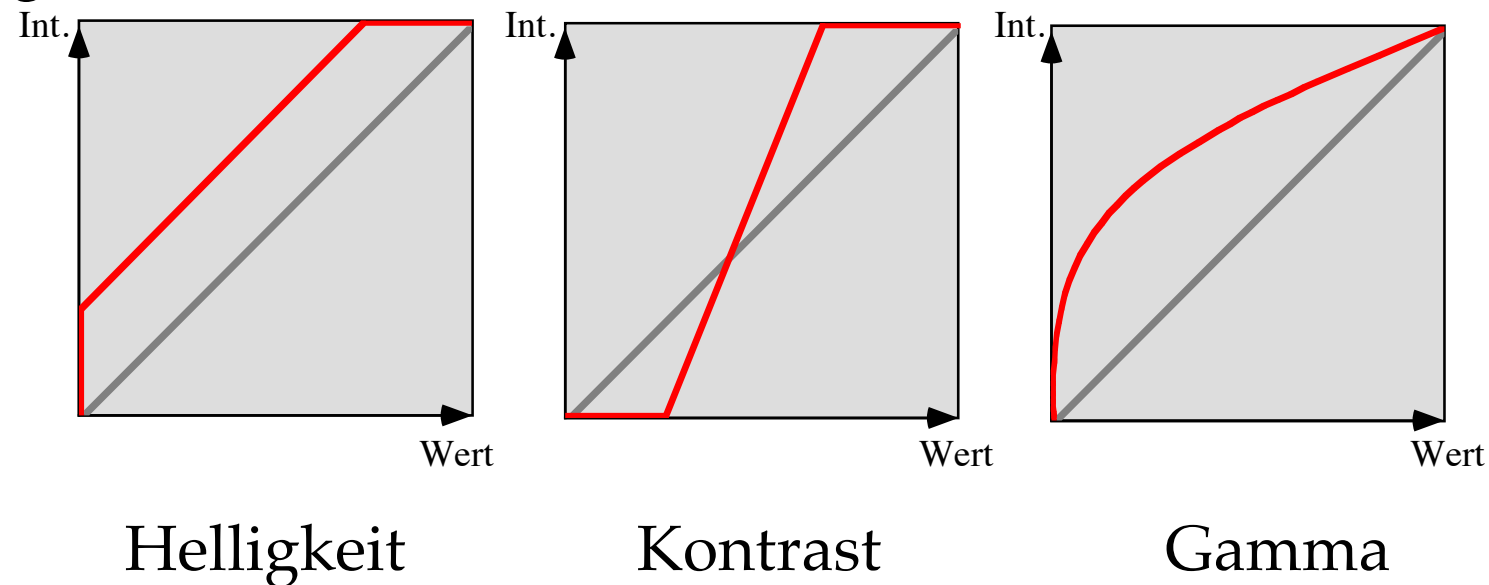


- Digitale Kameras benutzen CCD-Matrix
  - Auslesen spaltenweise



- ca 3000-4000 ppi
  - Unterschiede zwischen Zellen
  - nur 50 - 80% der Chipfläche ist mit aktiven Elementen bedeckt
- Active Pixel Sensor (APS, CMOS)
  - Photodiode + 3 Transistoren (buffer / amplify, reset, select)
  - Auslesen zeilenweise wie DRAM
- 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
  - Color-Matching verwendet auch 'Gamma'



- Datenmenge kann groß werden
  - Auflösung für Weiterverarbeitung wichtig (Druckgewerbe)
  - $(300 * 4 \text{ [inch]}) * (300 * 6 \text{ [inch]}) * 3 \text{ Bytes} = 6.480.000 \text{ Bytes}$
  - => Kompression

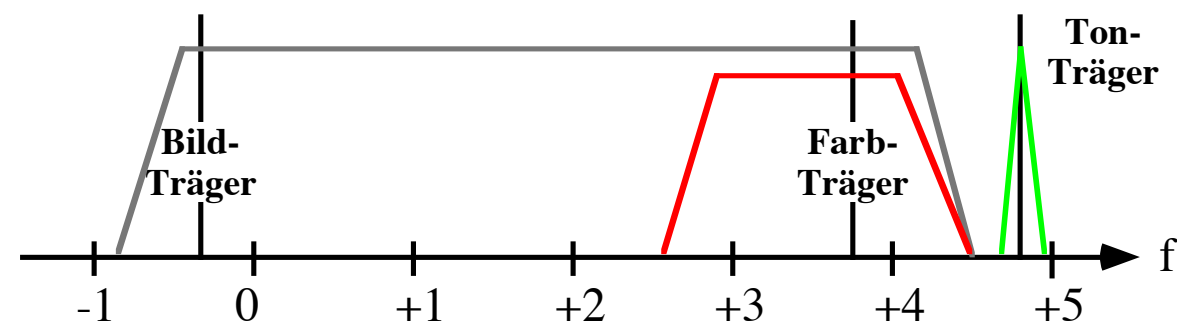
# Video

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

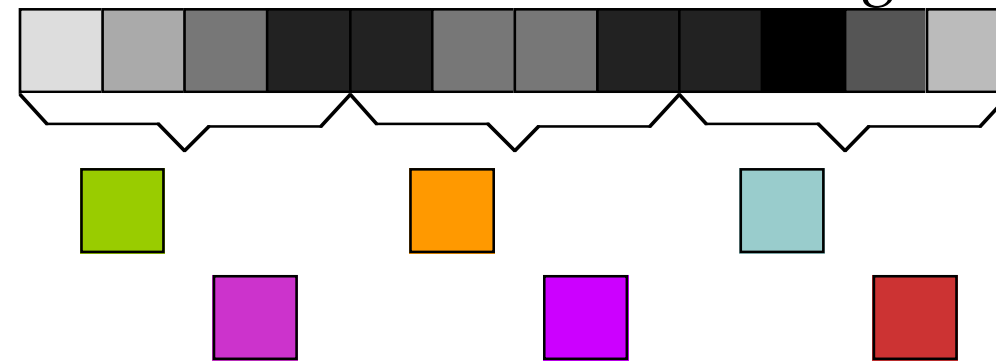
	Zeilen	Punkte/Zeile	Bilder/s	
CCIR 601	486	720	30	59,94 Hz
	586	720	25	50 Hz
CIF	288	352	25	Common Intermediate Format
QCIF	144	176	25	Quarter CIF
SIF	240	352	30	Standard Intermediate Format

- 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: High Definition TV
  - 1366-1920 \* 720-1080; (Bildwiederholrate 50 oder 59,94 Hz)
  - Bildwiederholrate  $\neq$  Übertragungsrate (24 Hz, 36 Hz, 72 Hz)
  - MPEG-2/4 zur Übertragung

- 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)

$$Y = 0,30 R + 0,59 G + 0,11 B;$$

$$U = (B-Y) * 0,493 = -0,15 R - 0,29 G + 0,44 B;$$

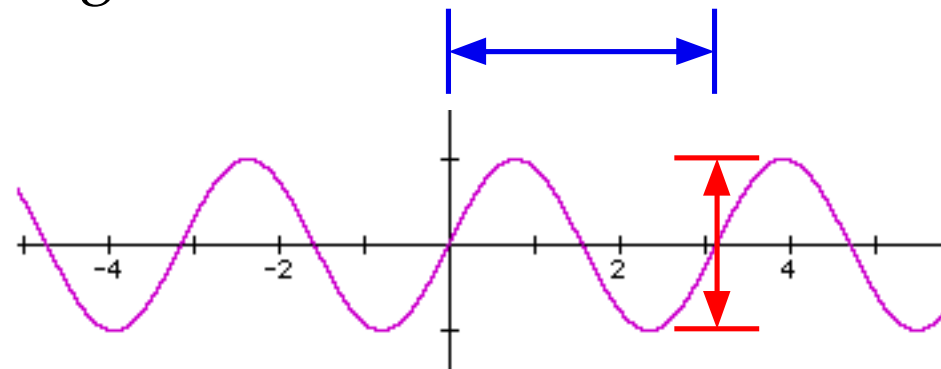
$$V = (R-Y) * 0,877 = 0,61 R - 0,52 G - 0,10 B;$$

- VHS noch stärker analog komprimiert

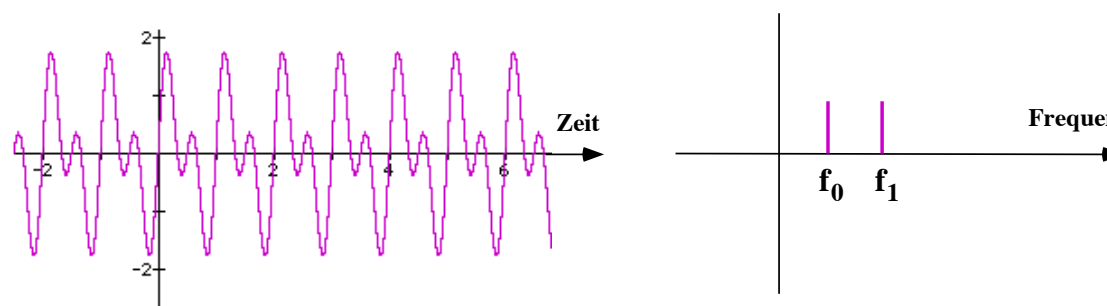
# Audio

## Audio-Eigenschaften

- Frequenz und Amplitude
  - Amplitude -> Lautstärke (gemessen in dB)
  - Frequenz (1m / Wellenlänge) -> Tonhöhe

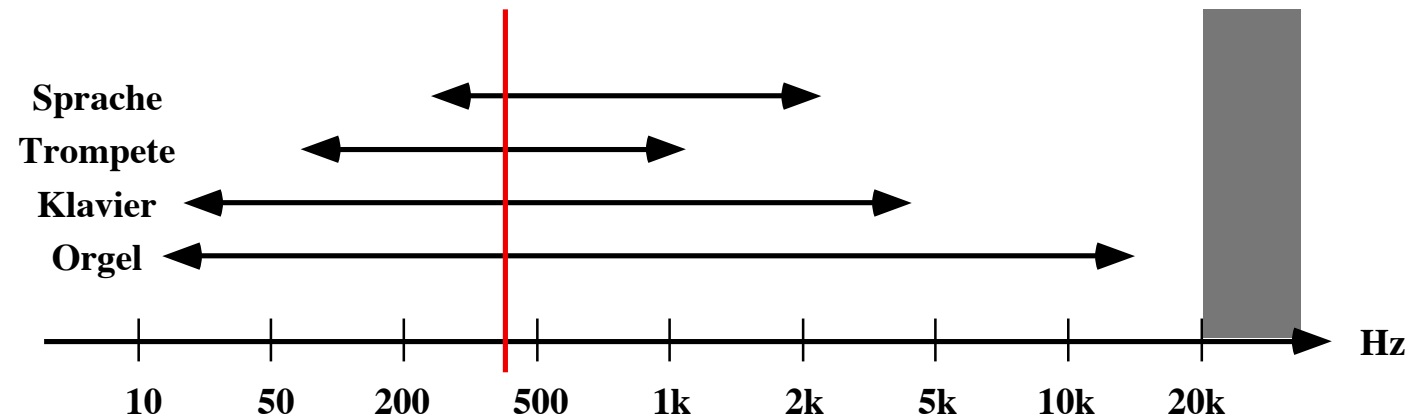


- Fourier: Jede Schwingung kann als Summe von Sinusschwingungen dargestellt werden:



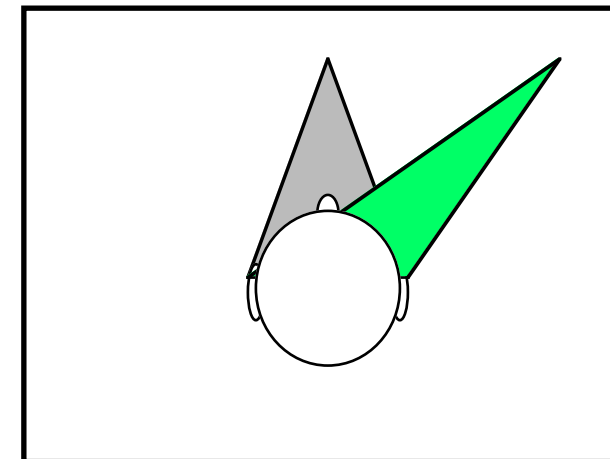
- Typische Frequenzbereiche

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

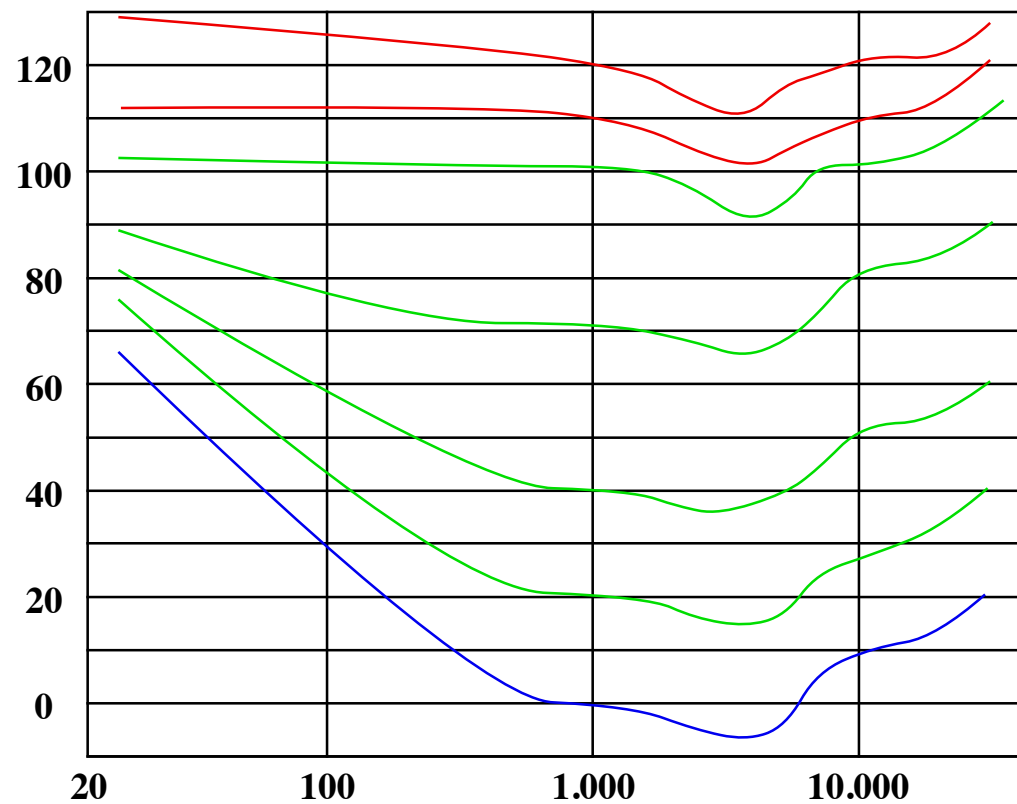


- Räumliches Hören

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

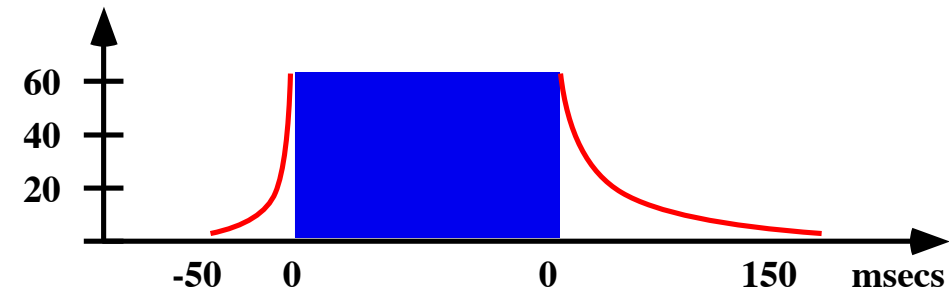


- Menschliches Hörvermögen
  - 20 - 20.000 Hz
  - hohes zeitliches Auflösungsvermögen
  - logarithmisch bezüglich Amplitude
- Lautstärkeempfinden nach Fletcher und Munson

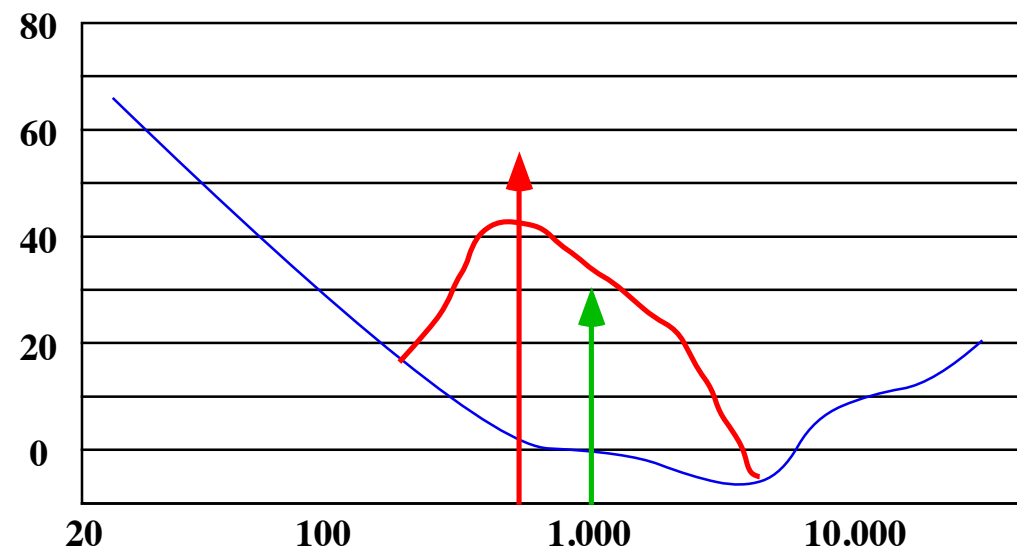




- Abschattung
  - Zeit



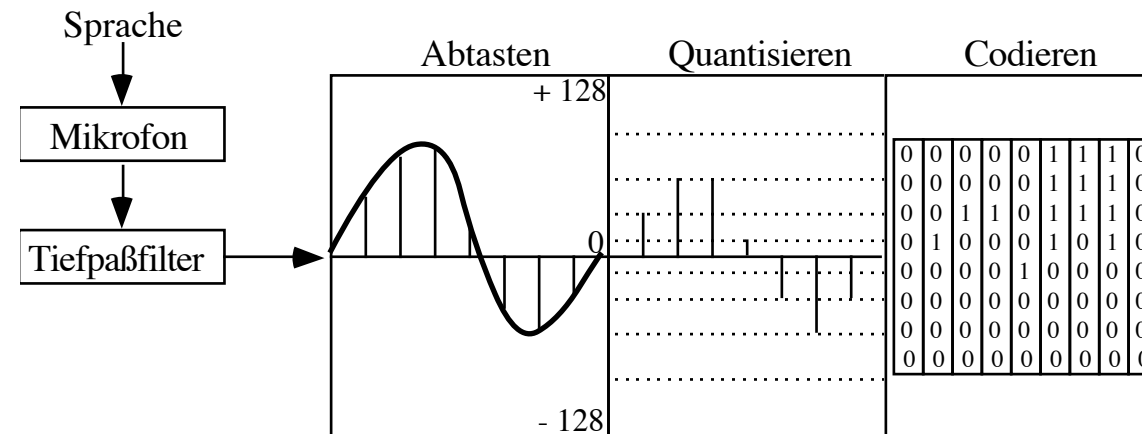
- Frequenz



- $\text{Phase} = \sin x + \sin (x + \pi)$
- zwei gleiche, phasenversetzte Schwingungen können sich auslöschen:

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

- Digitalisierung am Beispiel Telefon

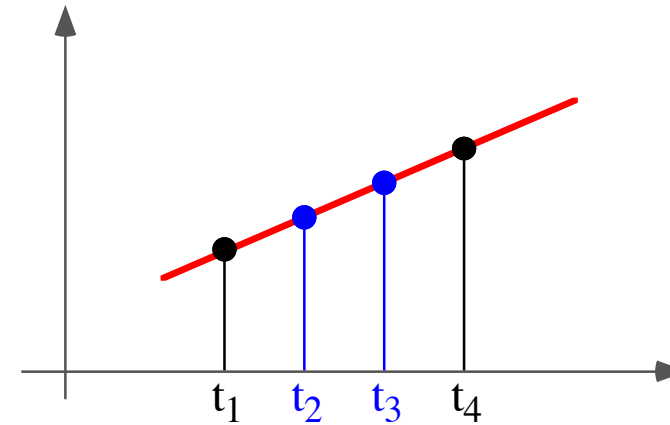
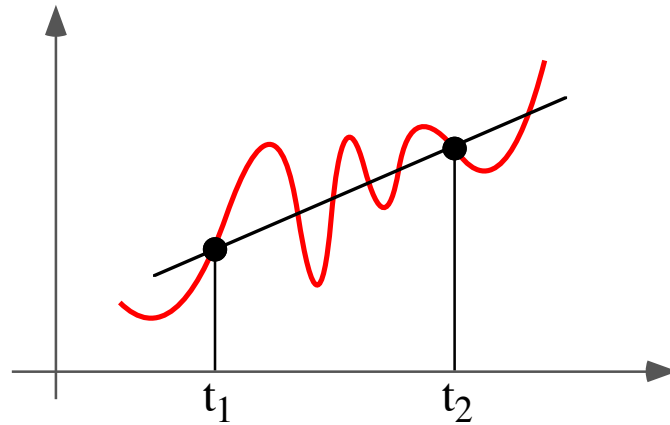


- Allgemein

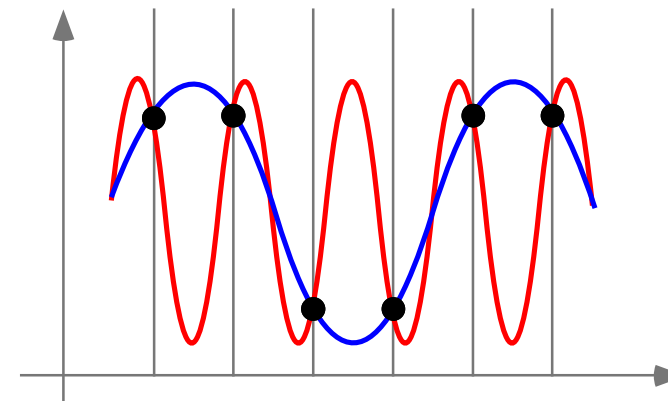
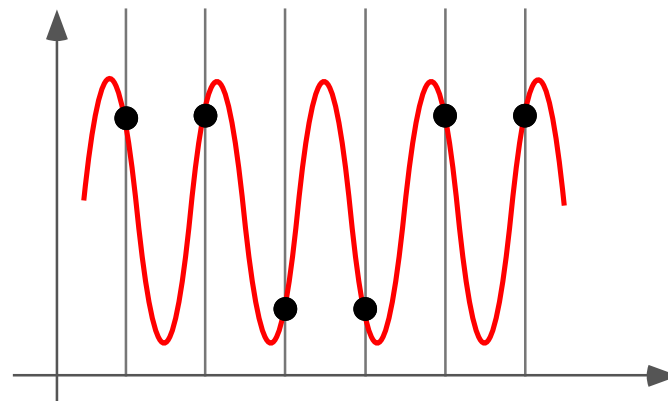
- zeitliche Diskretisierung (Abtasten, Sampling)  
Einteilung der Zeitachse in einzelne Stücke
- Wert-Diskretisierung (Quantisierung)  
digitalen Näherungswert finden  
Reelle Zahl vs. Real / Integer

- Abtasttheorem

- Anzahl Abtastwerte pro Zeiteinheit?

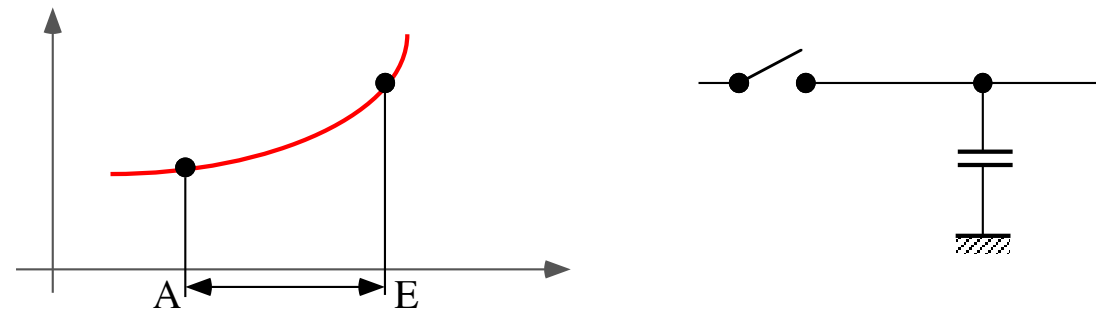


- Abtastfrequenz  $> 2 \cdot$  (höchste Frequenz)
- [Whittaker 1915/1929, Borel 1897]
- Aliasing bei zu niedrigen Abtastraten



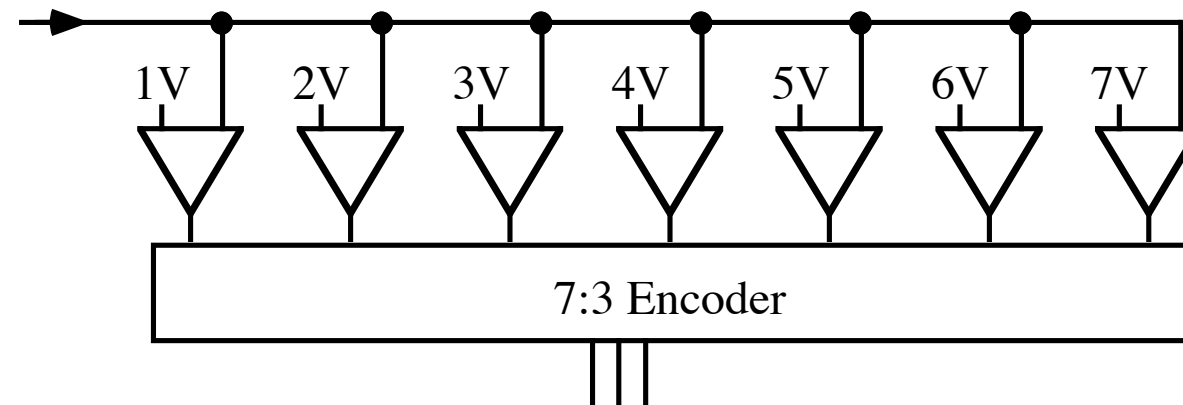
- Tiefpaß-Filter gegen Aliasing

- Sample-and-Hold

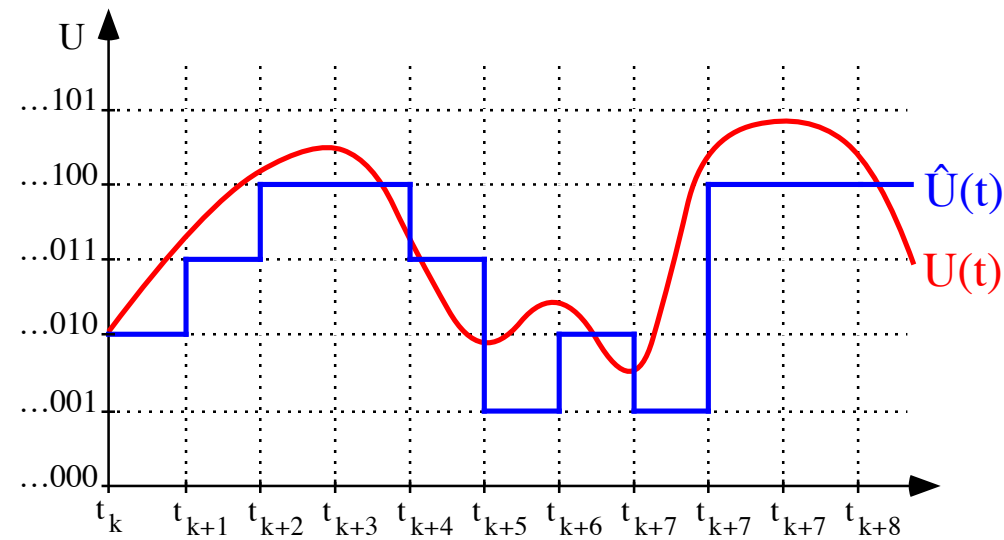


- Quantisierung (ADC)

- 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

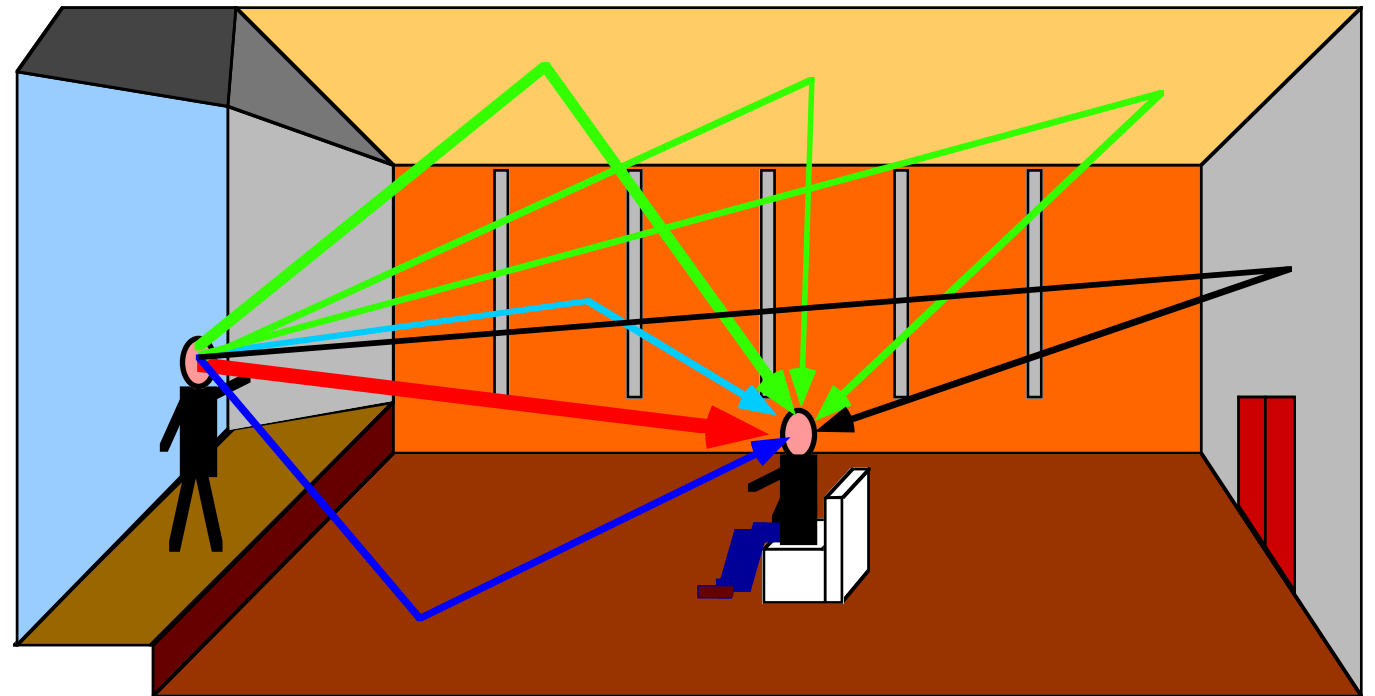
- als Integerzahl (CD: 16 bit)
- als Pseudo-Real (A-law,  $\mu$ -law: 8 bit)
- als Differenzen

$\pm$	E	x	p	M	a	n	t
-------	---	---	---	---	---	---	---

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)	3 -> 2	8,000	1	16 kbit/s

## Raumton

- Reflexionen von Wänden, Decke, Boden, Gegenständen
- Wahrnehmung der
  - Signalstärke
  - Richtung der Quelle
  - Dämpfung durch Kopf in höheren Frequenzen
  - Laufzeitunterschiede (650  $\mu$ 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

