Technische Informatik 2012

Konrad Froitzheim

Cables and Connectors - that is what computer architecture is about
[C.Gordon Bell]



Formales

Termine:

Vorlesung:	Montag	11:00 - 12:30, HUM-1115
	Donnerstag	11:00 - 12:30, HUM-1115, Woche 1
	Übungen	

Dramatis Personae:



Prof. Dr. Konrad Froitzheim

Professur Betriebssysteme und Kommunikationstechnologien

frz (at) informatik.tu-freiberg.de

Frank Gommlich: gommlich(at)informatik.tu-freiberg.de

- Vorlesungsunterlagen (kein Scriptum): http://ara.informatik.tu-freiberg.de/Vorlesungen/TI2012.doc
- Prüfung: Klausur in der vorlesungsfreien Zeit

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

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

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

Zur Prüfungsvorbereitung ist die intensive Lektüre der Fachliteratur unumgänglich.

Literatur

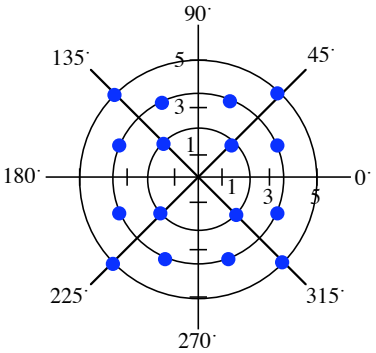
50 Years of Computing; IEEE Computer, 10/96
Black, U.: ATM: Foundation for Broadband Networks; Prentice Hall, 1995.
Brinkschulte, U., Ungerer, T.: Mikrocontroller und Microprozessoren; 2002.
Carne, E.B.: Telecommunications Primer; Prentice Hall, 1999.
Foster, C.: Computer Architecture; Academic Press, New York
Froitzheim, K.: Multimediale Kommunikationsdienste, 1996.
Häckelmann, H.: Kommunikationssysteme; Springer, 2000.
Halsall, F.: Computer Networks …; Addison-Wesley, 2005.
Patterson, D., Hennessy, J.: Computer Organization and Design, 2008.
Rauschmayer, D.J.: ADSL/VDSL Principles; Indianapolis, 1999.
Schiller, J.: Mobilkommunikation; München, 2000.
Stokes, J.: Inside the Machine; San Francisco, 2006.
tecCHANNEL: Prozessortechnologie; München, 2004.

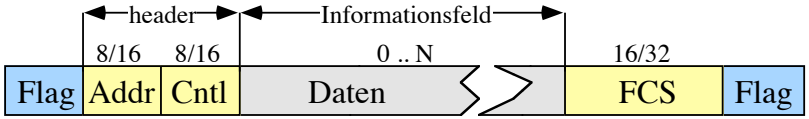
Inhaltsübersicht

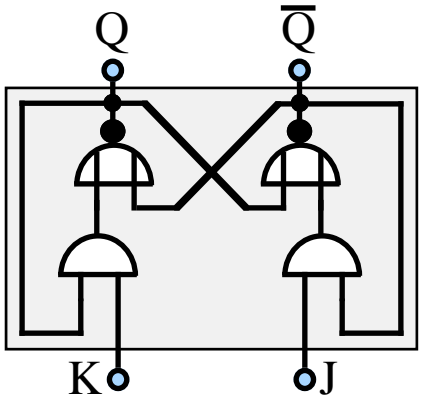
0. Wiederholung: Nachrichten und Zahlen
   - 0.1 Informationsbegriff
   - 0.2 Zahldarstellung
   - 0.3 Rechnen

1. Leiten
   - 1.1 Leitungseigenschaften
   - 1.2 Übertragungsmedien
   - 1.3 Signalbildung und Modulation
   - 1.4 Multiplex
   - 1.5 Digitale Kanäle
   - 1.6 Asynchrone Rahmenübertragung
   - 1.7 Lokale Netzwerke

2. Schalten (WH)
   - 2.1 Transistoren, Gates
   - 2.2 Rechnen und Speichern







3. Rechneraufbau

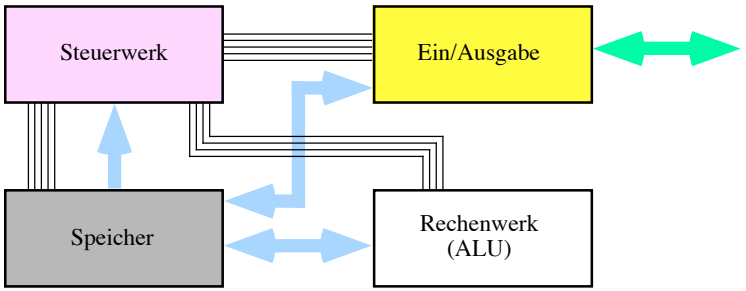
- 3.1 Funktionseinheiten: Speicher, Prozessor, Bus, …
- 3.2 Registertransfer
- 3.3 Hardware/Software-Interface und Timing
- 3.4 Compiler
- 3.5 Assembler
- 3.6 Taxonomien

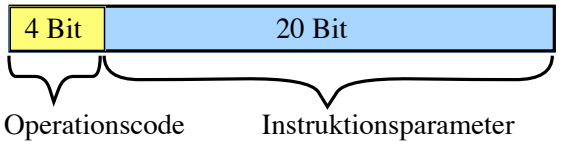
4. Instruktionssätze

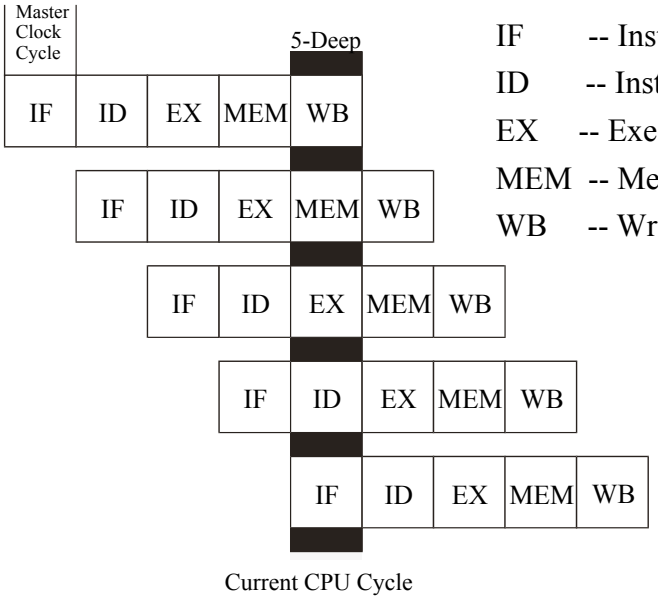
- 4.1 Adressierung und Operanden
- 4.2 Operationen
- 4.3 CISC
- 4.4 RISC

5. Pipelines

- 5.1 Instruktionen laden
- 5.2 Branch prediction und EPIC
- 3.2 Ausführung
- 5.4 Superskalare Architekturen



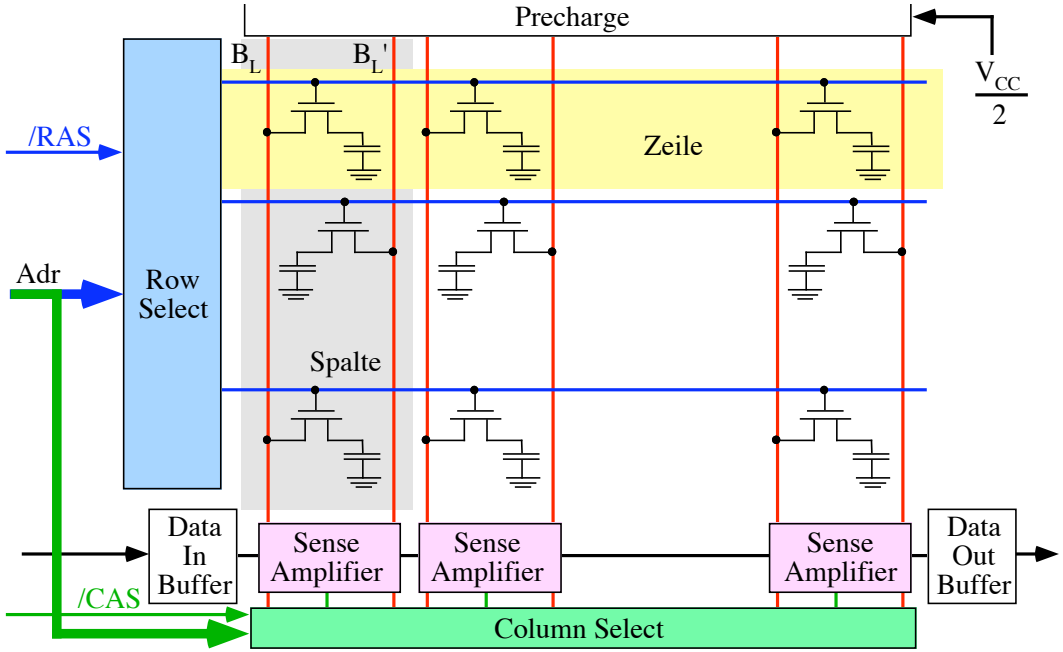


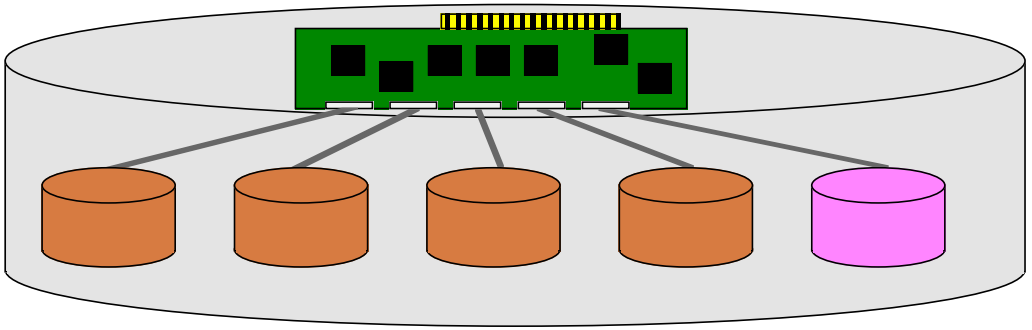


IF -- Instruction Fetch
ID -- Instruction Decode/Register Fetch
EX -- Execute/Address Calculation
MEM -- Memory Access
WB -- Write Back

6. Speicherarchitektur

- 6.1 Hauptspeicher
- 6.2 Bridges
- 6.3 Caches
- 6.4 Virtueller Speicher
- 6.5 Plattenspeicher





0. Nachrichten und Zahlen

0.1 Informationsbegriff

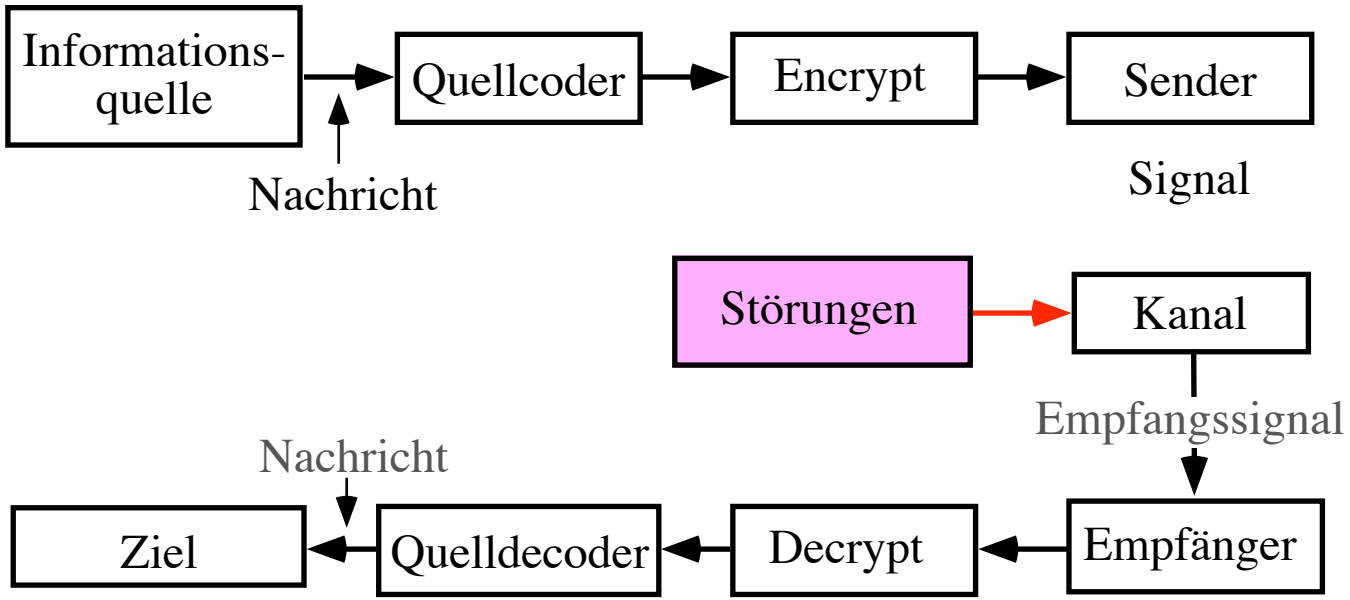
- Shannon, 1948:
  - Definition als mathematische Größe
- Nachrichtenquelle
  - Nachrichten mit Wahrscheinlichkeit p
  - Wahrscheinlichkeit bestimmt Informationsgehalt
- Definition: Entropie H = -ld p bit
- Beispiel Münzwurf
  - Ereignisse Kopf und Zahl
  - p(Kopf) = 1/2, p(Zahl) = 1/2

  => H(Kopf) = -ld 1/2 bit = ld 2 = 1 bit
- Beispiel Würfel
  - Ereignisse 1, 2, 3, 4, 5, 6
  - p(i) = 1/6

  => H("3") = -ld 1/6 bit = 2,58… bit

- Wahrscheinlichkeit einer Symbolfolge $s_1 s_2 \ldots s_n$
  - Symbole haben Wahrscheinlichkeit $p(s_i) = p_i$
  - $p = p(s_1) * p(s_2) * \ldots p(s_n)$
  - p("the") = 0,076 * 0,042 * 0,102 = 3,25584 * 10^{-4}
- Entropie einer Nachricht: H = - ld p [bit]
  - H("the") = 11,6 bits
- Wahrscheinlichkeit $p(s_i)$
  - allein ≠ im Kontext
  - $p(s_i = 'h') = 0{,}042$ => H('the') = 11,6 bits
  - $p(s_i = 'h' \mid s_{i-1} = 't') = 0{,}307$ => H('the') = 6,5 bits
- Informationsrate in
  - englische Buchstaben: 4,76 bit/Buchstabe
  - englischem Text: 4,14 bit/Buchstabe
  - englischem Text: 9,7 bit/Wort, ~ 2 bit/Buchstabe

- Satz von der rauschfreien Quellkodierung
  - H(N) = x bit => ∃ Kode K mit Länge$_K$(N) = x Bits
  - Mittlere Kodelänge kann Entropie annähern
- Kode = Nachricht + Redundanz
- Übertragung



- Quellkodierung reduziert ungewollte Redundanz
- Sicherheitskodierung verhindert Mithören
- Kanalkodierung (Fehlerkontrollkodes) gegen Störungen

- Kanalkapazität

$$C = Bandbreite \log_2(1 + \frac{Signalleistung}{Rauschleistung})$$

- Satz [Shannon]

> Für jeden Kanal mit Kanalkapazität C existieren Kodes, so daß in diesem Kanal Informationen mit einer Rate kleiner C übertragen werden können mit beliebig kleinem Fehler.

  - Existenzbeweis
- Coderate = Anzahl Nachrichten / Anzahl Symbole im Code
- Hamming: Error Detecting and Error Correcting Codes [1950]
  - konstruktiver Ansatz
  - kombinatorische Codes
  - z.B. Parity
- Hamming-Abstand
  - Anzahl Elemente, in denen Codeworte v und w unterschiedlich

- Kommunikation [Shannon, Weaver]:
  - Lebewesen oder Maschine beeinlußt anderes Lebewesen oder Maschine
  - technisches Problem
  - semantisches Problem (Symbole und ihre Bedeutung)
  - Effektivitäts-Problem (Einfluß der Bedeutung)
- Speziell: Datenaustausch
  - über immaterielle Träger
  - größere Distanzen
  - zwischen Mensch und/oder Maschine
- Immaterielle Träger
  - Energieflüsse, z.B. elektromagnetische Wellen
  - kein CD- oder Diskettentransport

0.1 Zahldarstellung

0.1.1 Ganze Zahlen

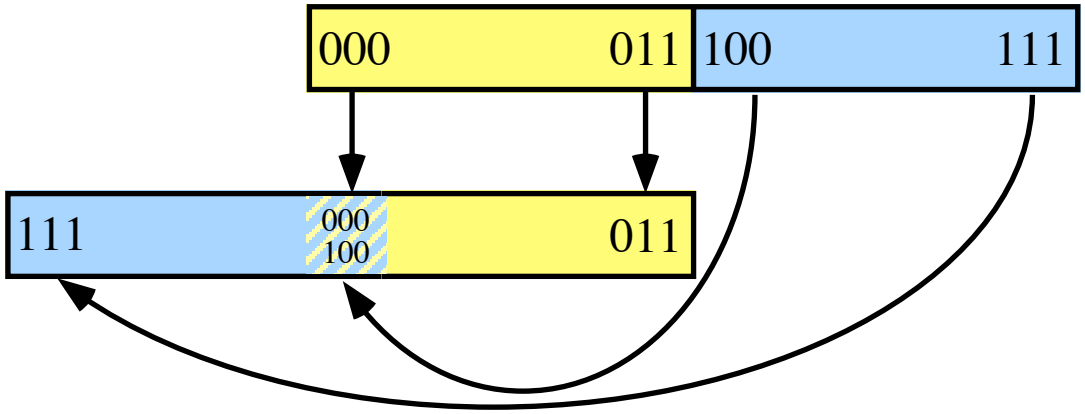
- Polyadisches Zahlsystem
  - $z = \sum_{i=0}^{n-1} a_i B^i$
  - $0 \le a_i < B$
  - Basis 10: $1492 = 2*10^0 + 9*10 + 4*100 + 1*1000$
  - Basis 2: $1492 = 1*1024+0*512+1*256+1*128+1*64$
    $+0*32+1*16+0*8+1*4+0*2+0*1 = 10111010100$
  - Basis 16: $1492 = \$5D4 = 5*256 + 13 * 16 + 4 * 1$
- Zahlenbereich
  - 10 Bit => 0..1023 = 1 'Kilo' -1
  - 20 Bit => 0..1024*1024-1 = 1 'Mega' -1
  - 32 Bit => 0..4 294 967 295 (2^{32}-1) = 4 'Giga' - 1
- negative Zahlen?

$$2^{10} = 1024$$

- Vorzeichen-Betrag (sign-magnitude)
  - 1 Bit Vorzeichen
  - höchstes Bit

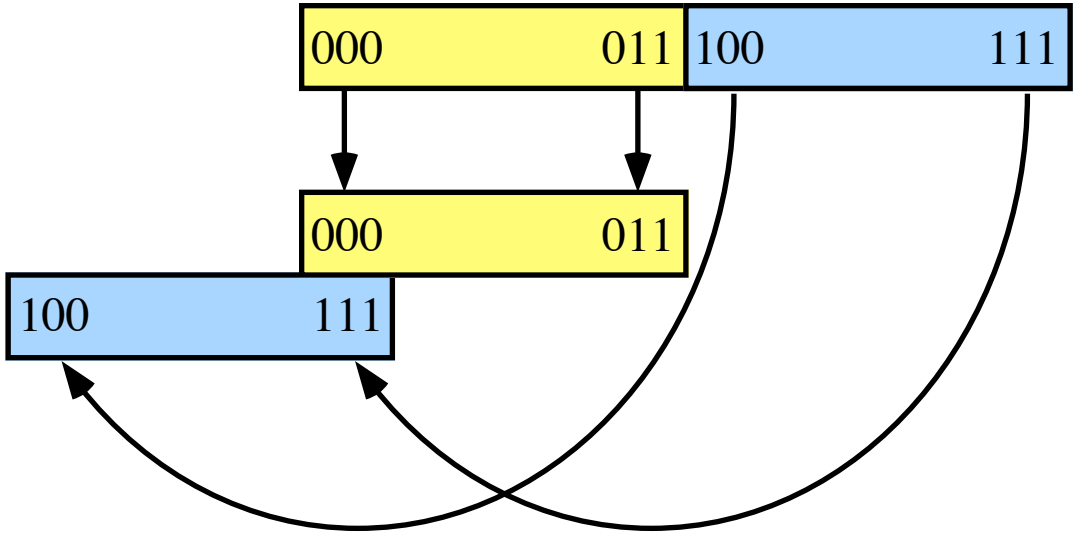


  - 8 Bit (256 Werte): -127..127



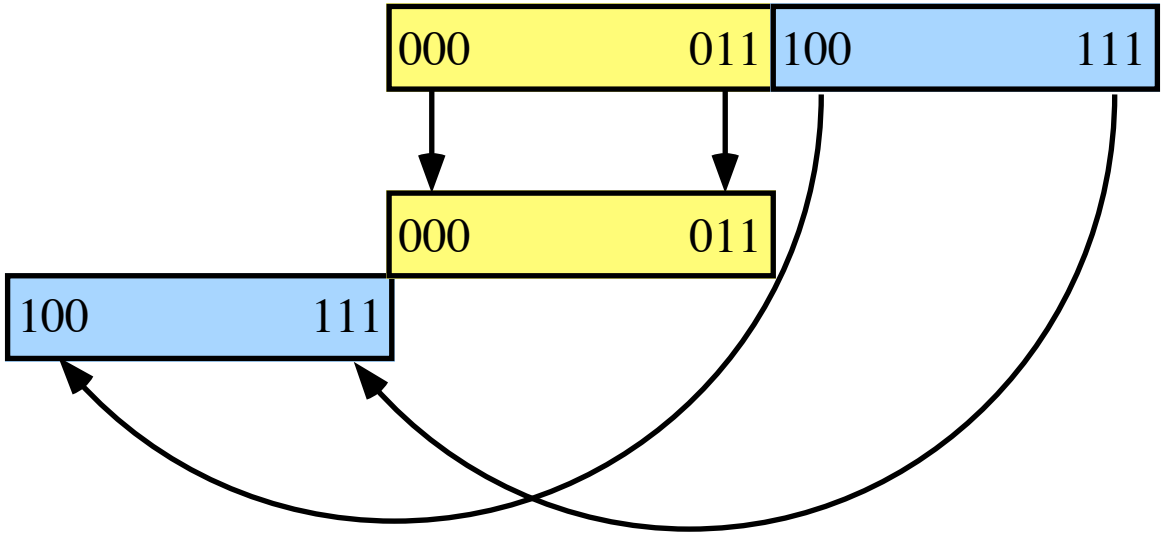
  - komplexe Rechenregeln

- 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

0.1.2 Rationale und Reelle Zahlen

- Rationale Zahlen
  - Brüche: 1/2, 1/3, …
  - Dezimalschreibweise ggg,ddd: 0,5; 12,625
  - evtl unendlich viele Stellen: 16/3
  - ggggg,ddddd… : 1,33333333333333333333...
  - ggg,ddd = ggg + 0,ddd
  - ganzzahliger Teil + Bruchteil

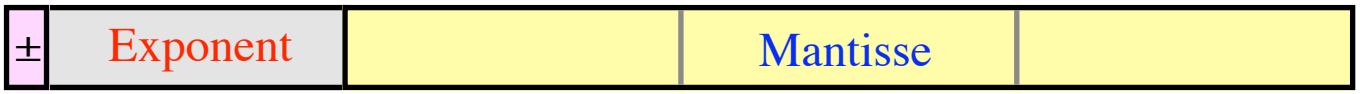
±	64	32	16	8	4	2	1

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

- Näherungsdarstellung von reellen Zahlen
  - $\pi \approx 3{,}1415926$
  - $\sqrt{2} \approx 1{,}414213562$
  - wann sind diese Fehler nicht katastrophal?

  => numerische Mathematik

- Normalisieren
  - ggg,ddd = 0,gggddd * 10^{3}
  - 1,34567 = 0,134567 * 10^{1}
  - 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

0.2 Rechnen

- Addieren im Zehnersystem
  - stellenweise addieren

```
    1513
+   2112
    3625
```

  Übertrag

```
    1523
+   2192
    3715
```

  - Rechenregeln für Übertrag
    - 7+8 = 5, Übertrag 1
    - 5+5 = 0, Übertrag 1
- Binärer Fall
  - stellenweise addieren

```
    01001010
+   00101111
    01111001
```

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

- Beschränkte Stellenzahl
  - größte darstellbare Zahl
  - Ergebnis grösser: Überlauf

```
    11001010
+   10101111
    ---------
    01111001
```

  - eventuell negative Zahl

```
    01001010
+   01101111
    ---------
    10111001
```

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

- Multiplikation
- Primitives Verfahren

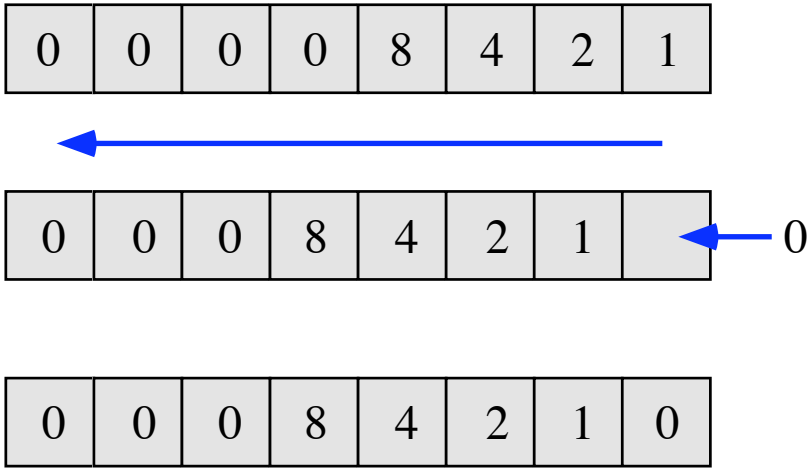
  - Multiplikand * Multiplikator

  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 ≤ Stellenzahl Multiplikator
  - im Zehnersystem:

$$
\begin{array}{rr}
1214 * 211 & \\
\hline
 & 1214 \\
 & 12140 \\
+ & 242800 \\
\hline
 & 256154
\end{array}
$$

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

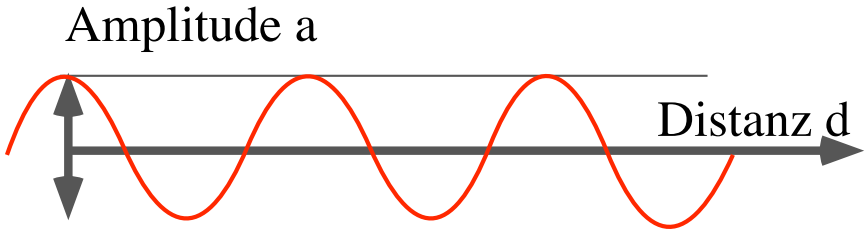
- binär Multiplizieren
  - ShiftLeft = Multiplikation mit 2
  - Stellen nur 0 oder 1 => Multiplikand addieren oder nicht
- Verfahren
  1. Ergebnis = 0; i = 0; a = Multiplikand; b = Multiplikator
  2. falls letzte Stelle von b = 1: Ergebnis = Ergebnis + a
  3. ShiftLeft(a), ShiftRight(b)
  4. Falls b>0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

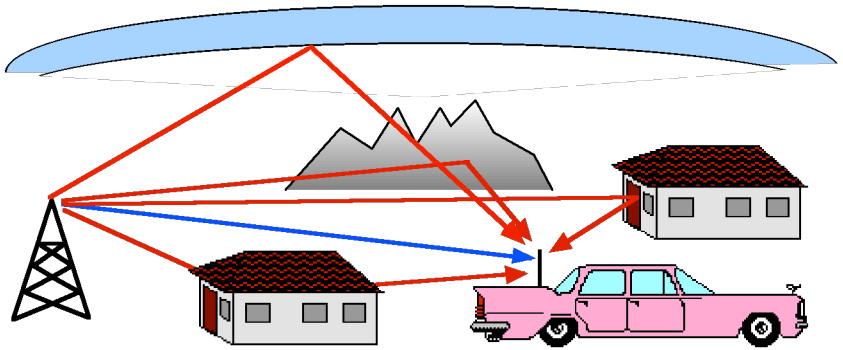
Iteration	a	b	erg
0	0000 1100	0000 0101	0000 0000
			+ 0000 1100
1	0001 1000	0000 0010	0000 1100
			nix tun
2	0011 0000	0000 0001	0000 1100
			+ 0011 0000
3	0110 0000	0000 0000	0011 1100

1. Leiten

1.1 Leitungseigenschaften

- Physikalisches Signal zur Darstellung von Datensymbolen:
  - Schallwellen
  - Licht
  - elektrischer Strom & Spannung
  - Mikrowellen
- Transport über einen Übertragungskanal
  - Luft
  - Draht, Telefonleitung
  - Lichtleiter für Lichtimpulse
  - Funkkanal für Radiosignale
- Eigenschaften des Kanales
  - Wellenwiderstand und Reflexion
  - Laufzeit
  - Wellengleichung und Phasengang
  - Dämpfung
  - Bandbreite

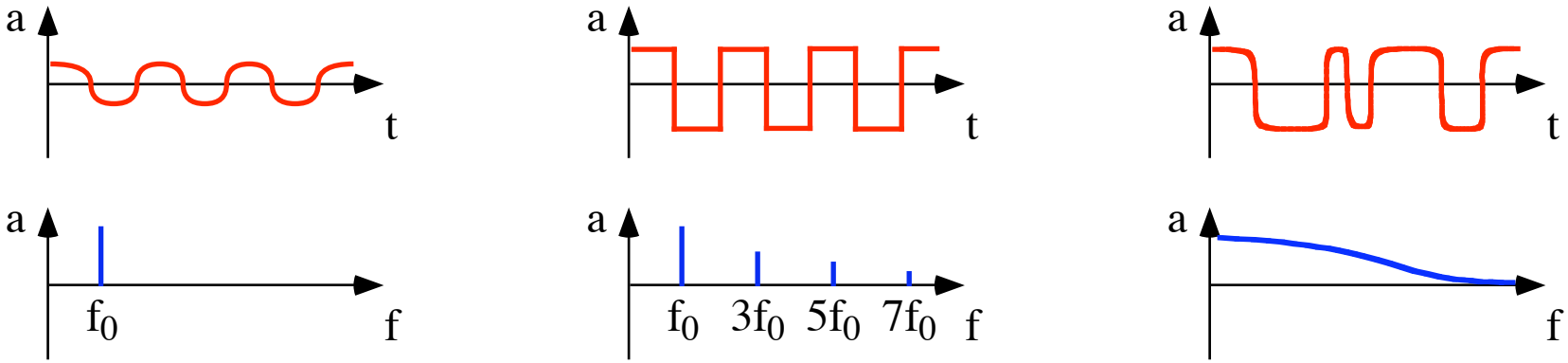




- Jedes periodische Signal kann dargestellt werden als
  - Summe von Sinus und Cosinus [Joseph Fourier, 1822]

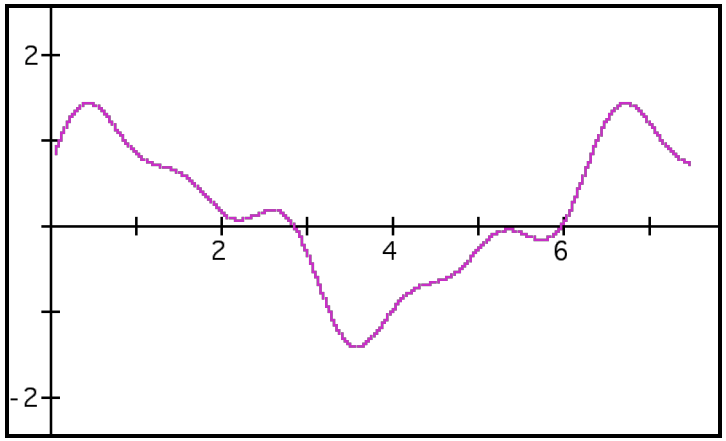
$$v(t) = a_0 + \sum_{n=1}^{\infty} a_n \sin n\omega t + \sum_{n=1}^{\infty} b_n \cos n\omega t$$

  - a_0, a_n, b_n sind die Fourierkoeffizienten
  - Berechnung mittels Fourieranalyse
  - a_0 ist der Gleichstromanteil
- Signal und Frequenzspektrum

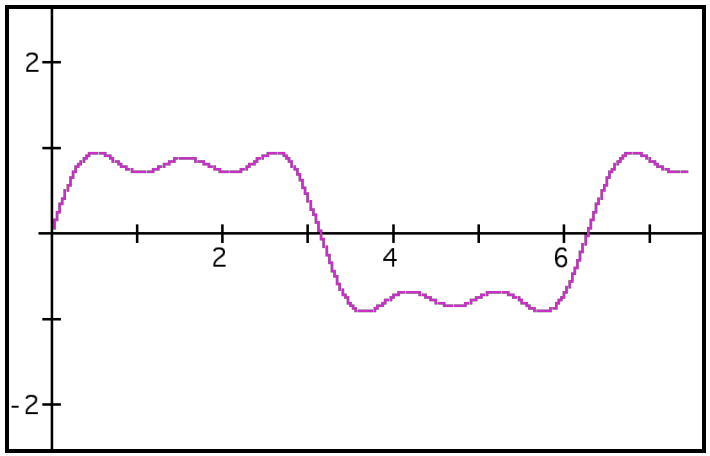


- Übertragungsleitung als Filter
  - dämpft unterschiedliche Frequenzen verschieden
  - Hochpass, Tiefpass, Bandpass
  - Verzögerungswirkung auf Frequenzen
  - Reaktion auf Phasen
- Lineare Schaltkreise und Sinuswellen
  - verändern Frequenzen nicht
  - können relative Amplituden ändern
  - können relative Phase verschieben

$\sin(x+\pi/4) + 1/3 \sin(3x) + 1/5 \sin(5x)$



$\sin(x) + 1/3 \sin(3x) + 1/5 \sin(5x)$



- Effekte bei der Übertragung

0 1 0 0 0 0 1 0 0

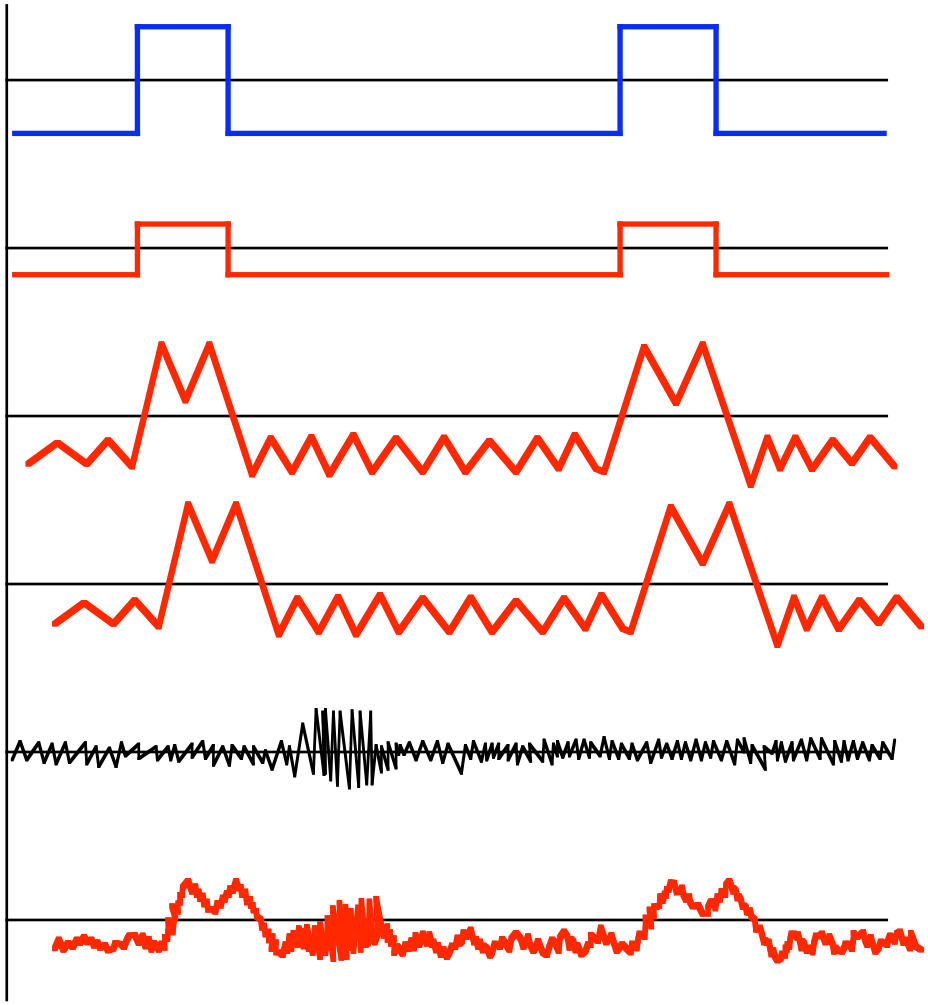
Originalsignal

Dämpfung

Bandbreitenbeschränkung

Verzögerung

Rauschen



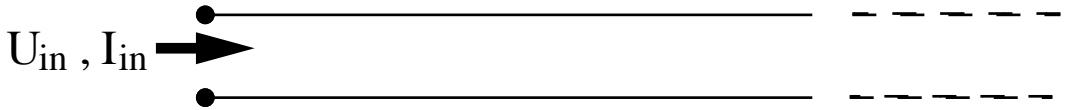
Übertragungsfehler!

0 1 1 0 0 0 1 0 0

1.1.1 Wellenwiderstand und Reflektionen

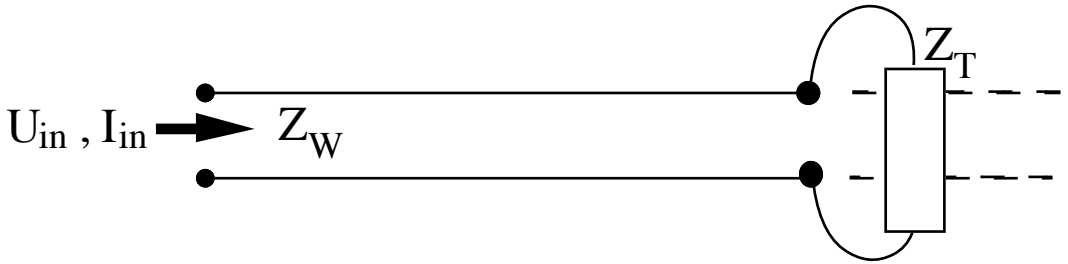
- Wellenwiderstand
  - Ohmsches Gesetz: R = U/I
  - Widerstand R nur wenn U und I sich 'langsam' ändern
  - Impedanz Z = U/I
  - frequenzspezifisch
  - Eigenschaft des Übertragungsmediums
- Unendlich lange Leitung bietet am Eingang Wellenwiderstand Z_w

Leitungseigenschaft $Z_W = U_{in}/I_{in}$



- Reflektion durch abrupte Änderung des Wellenwiderstandes
  - Bsp. Schall an der Grenze Luft-Wasser
  - Bsp. Koaxialkabel mit verschiedenen Geometrien

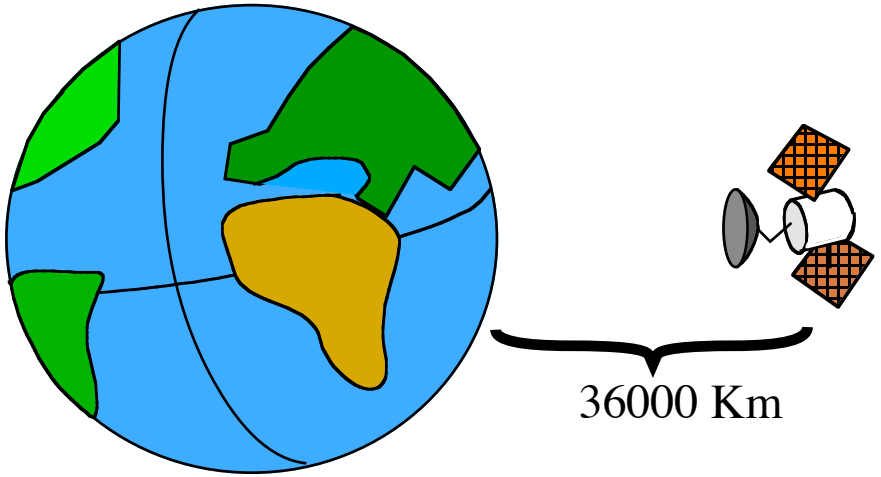
- Leitung abschneiden
  - abgeschnittenen Teil ersetzen durch einen Widerstand Z_T
  - Termination (Abschluss)



- Z_T <> Z_W Rückwirkungen auf den Eingang, sogenannte Reflektionen
- Abschlußwiderstand Z_T
  - absorbiert gesamte Energie <=> Z_T=Z_W
  - z.B. Thinnet 50Ω

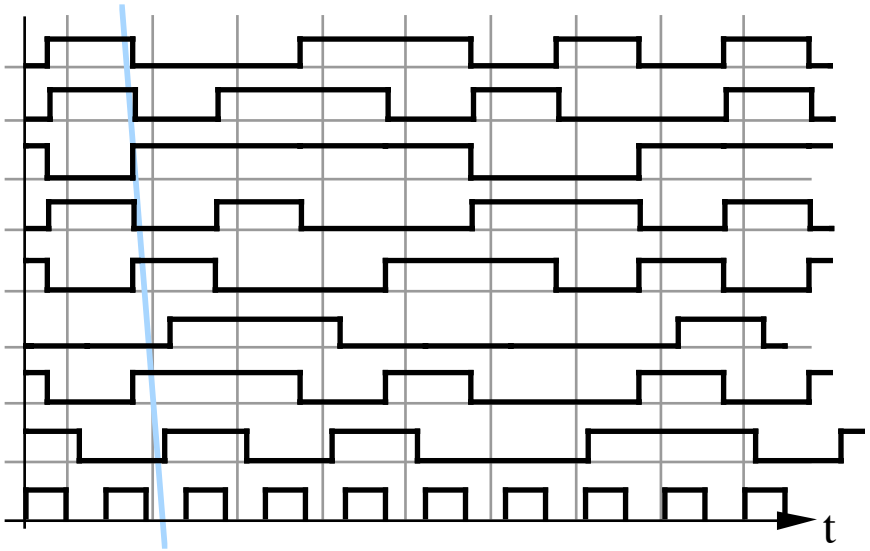
1.1.2 Laufzeit

- Lichtgeschwindigkeit c : obere Grenze ~ 299.792,458 km/sec
  - Ausbreitungsgeschwindigkeit auf Leitungen < c
  - Signal im Kabel ca. 200.000 km/sec
  - Schall 0,343 km/sec, Erdbebenwellen 8 km/sec
- Signalverzögerung beim Telefonieren über Satelliten:



- Zusätzliche Verzögerungen im Vermittlungsrechner
- Medium wird Speicher
  - 5000 km => Laufzeit 25 ms
  - 64 kbit/s => 1600 bit
  - 2 Mbit/s => 50000 bit

- Probleme bei paralleler Datenübertragung
  - Adern gleich lang und gleich schnell?
  - 20 cm Draht = 1 Nanosekunde
  - Skewing: störende Laufzeitunterschiede

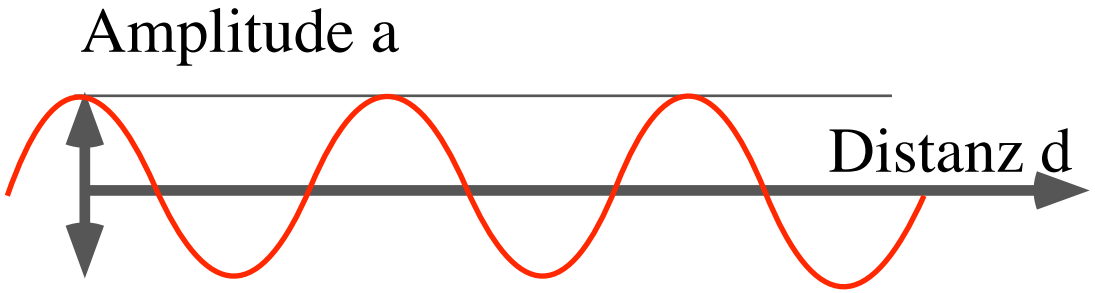


  - Taktflanke trifft eventuell nicht immer die Mitte des Datenbits
  - siehe z.B. PCIe

1.1.3 Wellengleichung und Phasengang

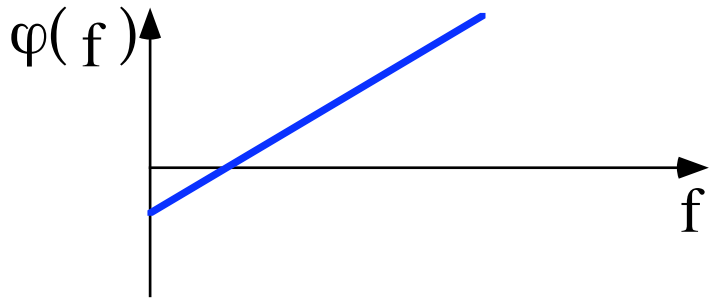
- Signalamplitude zur Zeit t am Ort d:

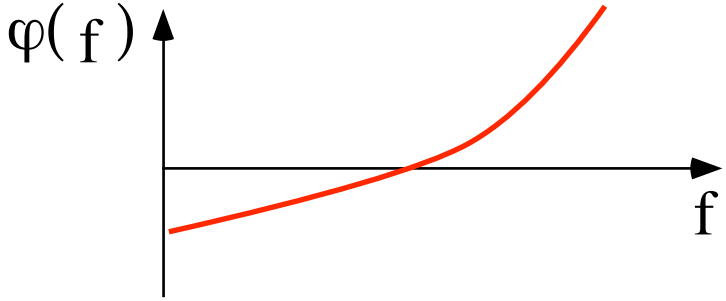
  $a(t,d) = A*\cos(2\pi(f*t - d/\lambda)) = A*\cos(\varphi)$



- Phase φ für ein bestimmtes t_o und d_o auf einer Leitung:
  - φ = Phase = $2\pi (f*t - d/\lambda) = \varphi(f,t,d)$
  - Ausbreitungsgeschwindigkeit c, Wellenlänge λ: $f = c/\lambda$
- Phasengang linear: frequenzunabhängiger Einfluß auf
  - Wellenlänge, Geschwindigkeit

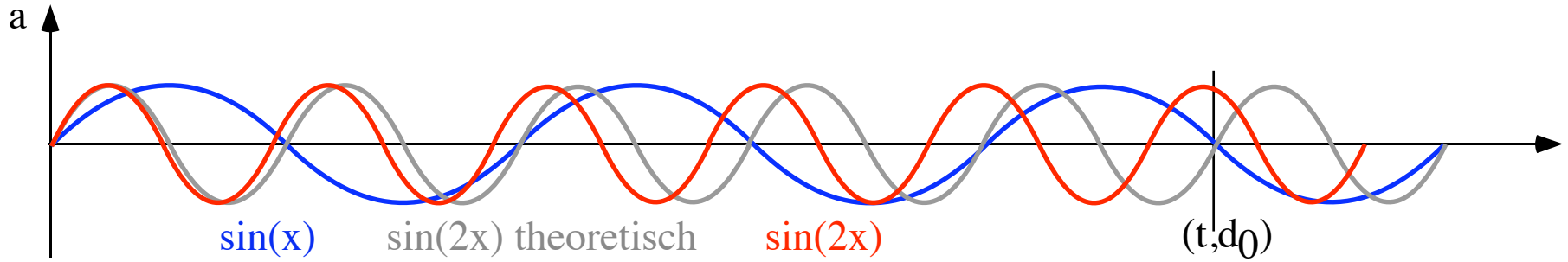
- Linear in f: $\varphi(f) = 2\pi(f \cdot t_0 - {}^{d_0}\!/_{\lambda}) = 2\pi(f \cdot t_0 - d_0 \cdot {}^{f}\!/_{c})$





- Nicht linear in f: $\varphi(f) = 2\pi f(t_0 - {}^{d_0}\!/_{c(f)})$

  - höhere Frequenzen breiten sich langsamer aus (λ wird kleiner)

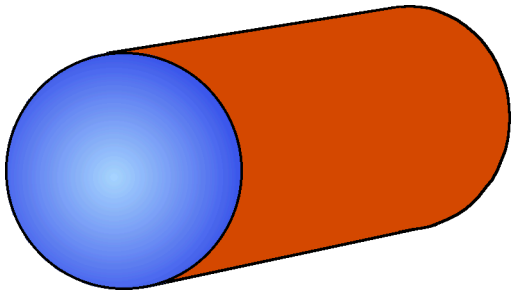


- Frequenzabhängige Laufzeit (Dispersion)
  - führt zu Verzerrungen und Intersymbolinterferenz
  - Phasengeschwindigkeit und Gruppengeschwindigkeit
  - Kompensierung (Equalization)

1.1.4 Dämpfung

- Abschwächung des Signals
- Ohmscher Verluste
- Skin Effekt

  - höhere Frequenz => Selbstinduktion
  - höhere Impedanz im Kern
  - Strom fließt auf der Oberfläche (=> weniger leitende Fläche)
  - erhöhte Abstrahlung



- Dielektrischer Verluste

  - isolierte Drähte bilden 'Kondensator'
  - Energieverlust durch den Isolator

- Abstrahlungsverluste
- Reflexionen

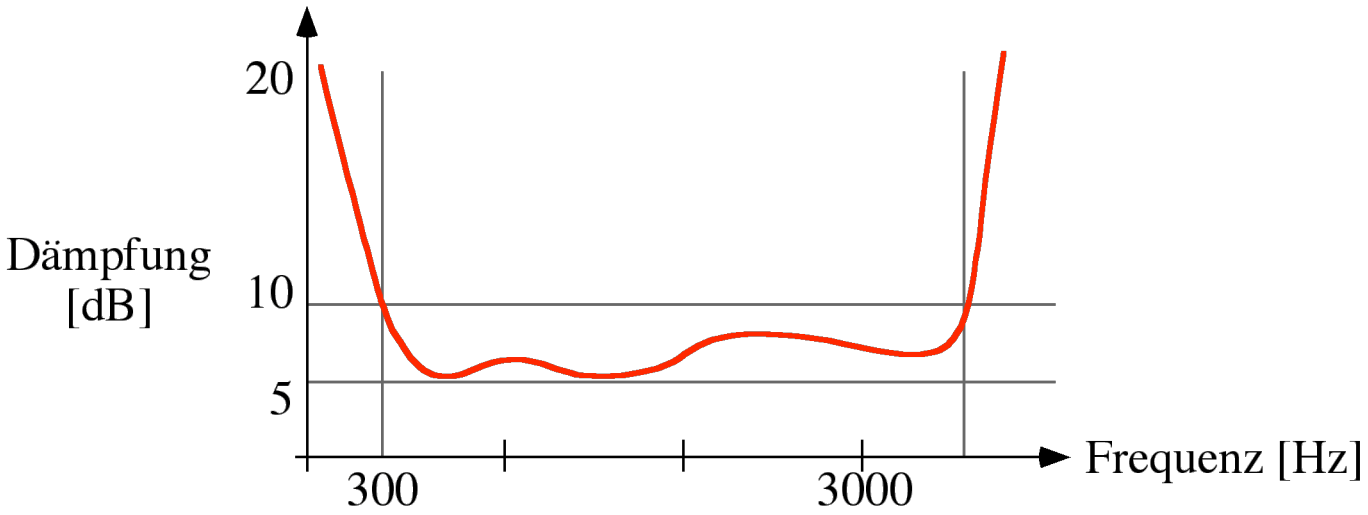
- Dämpfung $G = 10 \log(P_{in}/P_{out}) = 20 \log(U_{in}/U_{out})$ [dB]
  - additive Eigenschaft einzelner Leitungsabschnitte
  - exponentiell in der Entfernung
  - Referenzpegel von 1mV



- Frequenzabhängige Dämpfung => Filter, lineare Verzerrung
  - Korrektur mit Equalizer

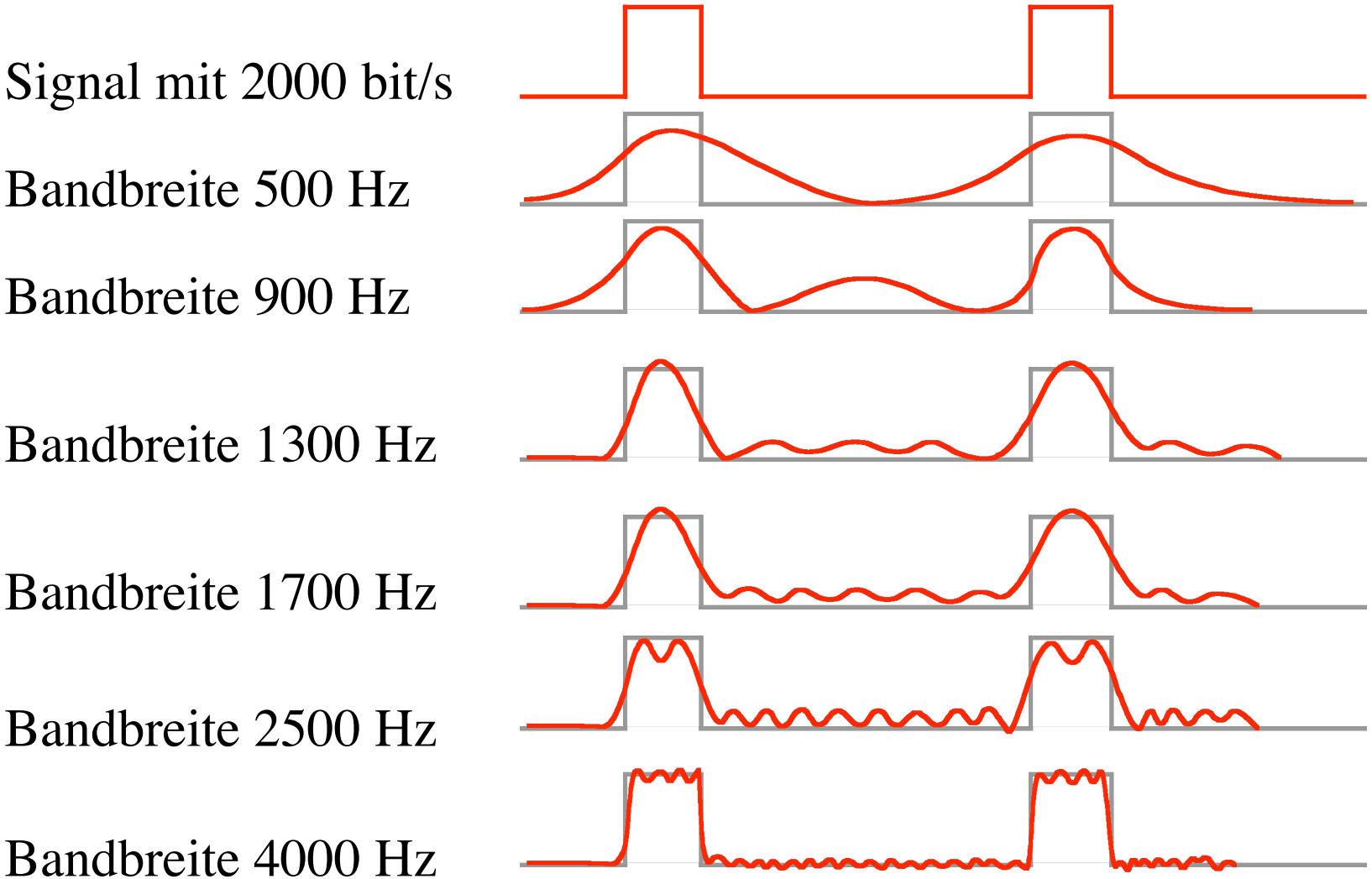
1.1.5 Bandbreite

- Band: Intervall zwischen unterer und oberer Grenzfrequenz
  - dazwischen einigermaßen gleichmäßige Dämpfung
  - linearer Phasengang
- Beispiel Telefon*verbindung*:



- Begrenzung nicht ideal
  - keine vertikalen Flanken
  - Abschneidefrequenzen festlegen
  - breiteres Frequenzband reservieren

- Einfluß der Bandbreite auf digitales Signal



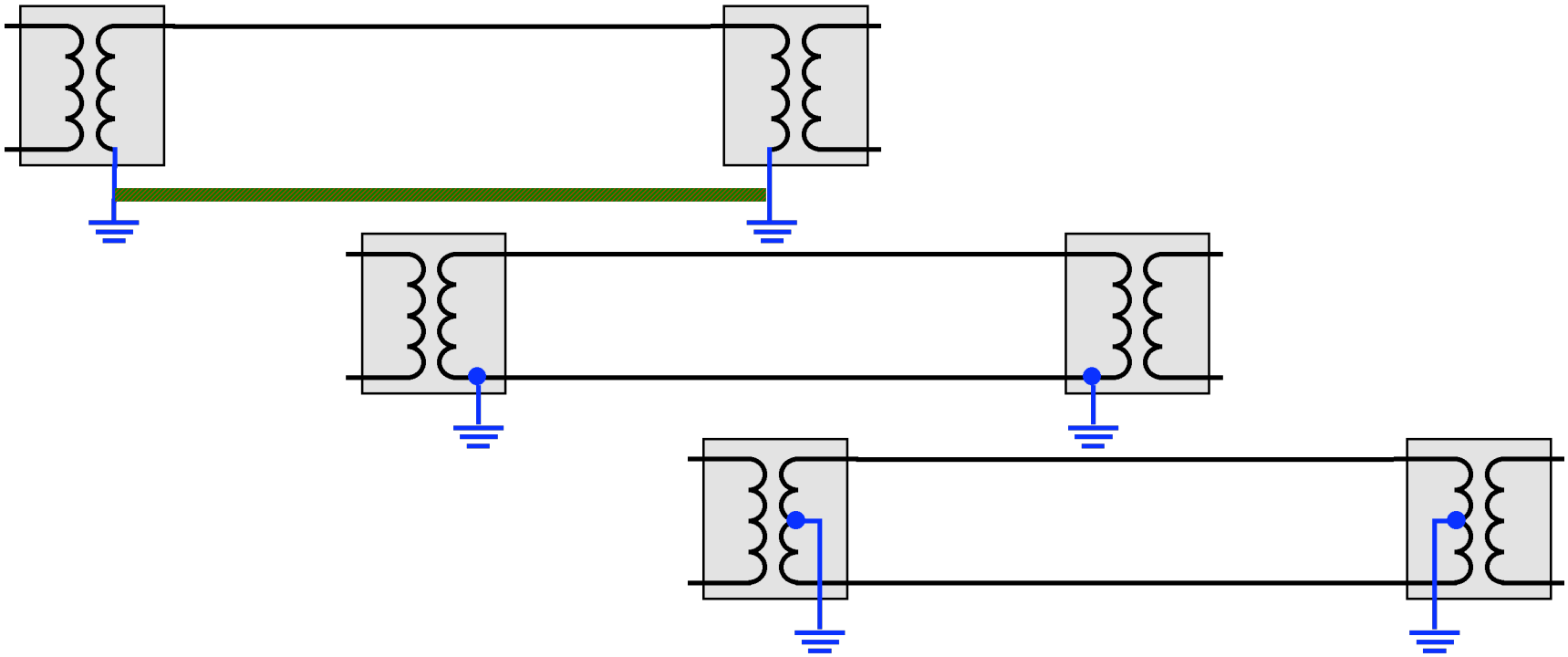
1.1.6 Störspannung (Rauschen)

- Einschaltspitzen
  - Elektrogeräte, Funkenbildung
- Übersprechen
  - zwischen Kabeln in einem Kabelkanal
  - zwischen Adern im Flachbandkabel
  - Funkfrequenzen
- Thermisches Rauschen
  - Kühlen von Leitungen
  - Halbleiterrauschen
- Reflexionen
- Störpegel gemessen in dBmV
- Signalstörung
  - transient: stochastischer Prozeß
  - permanent: weisses Rauschen
  - Impulsstörungen

1.2 Übertragungsmedien

1.2.1 Stromleiter = Kupferkabel

- Leiter meist aus Kupfer
  - früher teuer

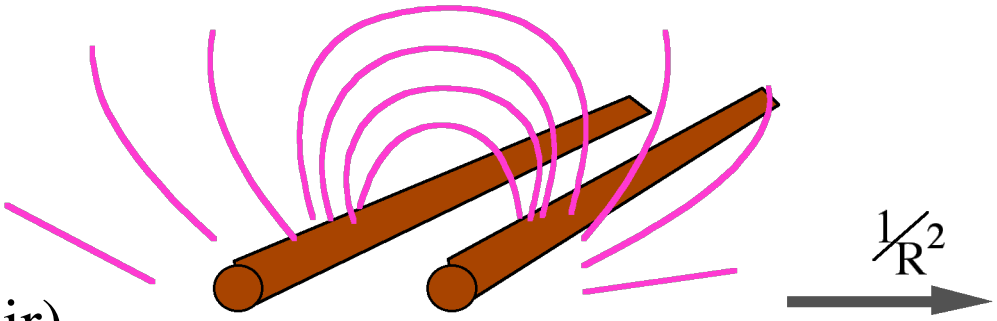


- Isolation aus Polyäthylen oder PVC
  - früher Papier oder Holzmark
  - Luftleitungen an Telegraphenmasten

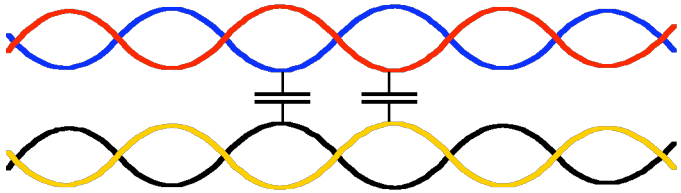
- Preisgünstig für kurze Entfernungen oder kleine Datenraten
- Relativ hoher Wellenwiderstand

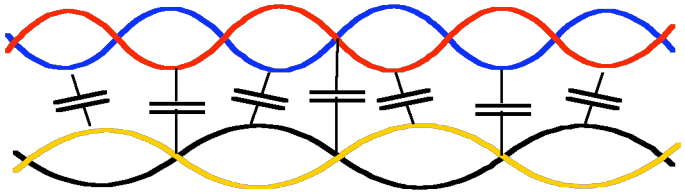
- Twisted Pair	0,9 mm	120 Ω	???? dB/mile
- Telefonfreileitung	2,7 mm	600 Ω	1 dB/mile
- FM Bandkabel		300 Ω	

  - angenehm für den Leitungstreiber
- Vieladrig für grössere Installationen
- Wenig oder keine Abschirmung
- Verdrillt
  - Reduktion der Abstrahlung (twisted pair)
  - Störungen wirken auf beide Adern gleich
  - Feld der Doppelader nimmt ab mit $1/R^2$
- Geschwindigkeit 130 - 200 Mm/s



- Besonderheiten der Verdrillung
  - Verdrillungslänge << Wellenlänge
  - in Mehrfachkabeln unterschiedliche Verdrillungslänge



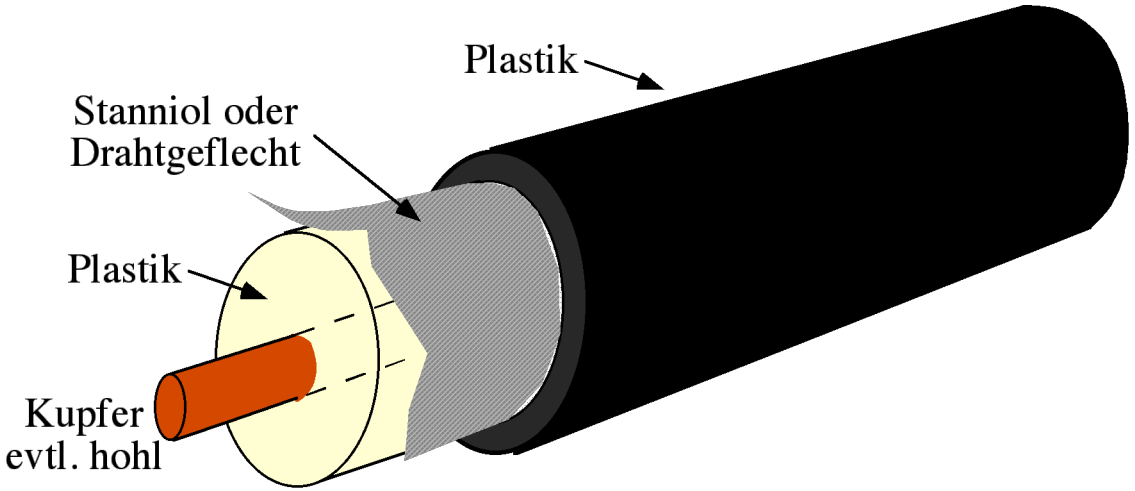


- Unshielded Twisted Pair (UTP)
  - 26 American Wire Gauge ~ 0,4 mm
  - 19 AWG ~ 0,9 mm
- Kategorieeinteilung in USA

CAT 1	Sprache, langsame Daten	bis 56 kbit/s
CAT 2	Daten bis 1 Mbit/s	
CAT 3	bis 16 MHz	10BaseT, 4 Mbit/s Tokenring
CAT 4	bis 20 MHz	16 Mbit/s Tokenring
CAT 5	bis 100 MHz	100BaseT, ATM
CAT 6	bis 250 MHz	Gigabit EN

Koaxial-Kabel

- Antennenkabel, Ethernet



- Wellenwiderstand 50 Ω oder 75 Ω, 200 - 240 Mm/s
- Kennzeichnung: Verhältnis Innenleiter/Außenleiter 2,6/9,5 [mm]
- Widerstandsverluste wachsen mit Wurzel der Frequenz => 2 bis 10 GHz
- Signalausbreitung im Dielektrikum zwischen den Leitern
- metallische Ummantelung gegen Störung und Abstrahlung
  - Aussenfeld vernachlässigbar
  - reduziert Überprechen drastisch
  - unter 100 kHz unwirksam

Hohlleiter (Waveguide)

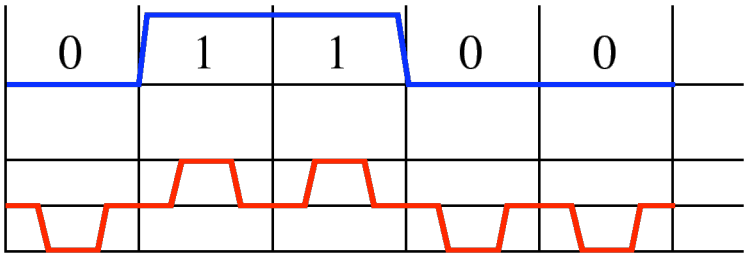
- vgl. 'Gartenschlauch-Telefon'
- Metallischer Hohlkörper
  - gefüllt mit Luft oder Stickstoff
  - rund, elliptisch oder rechteckig

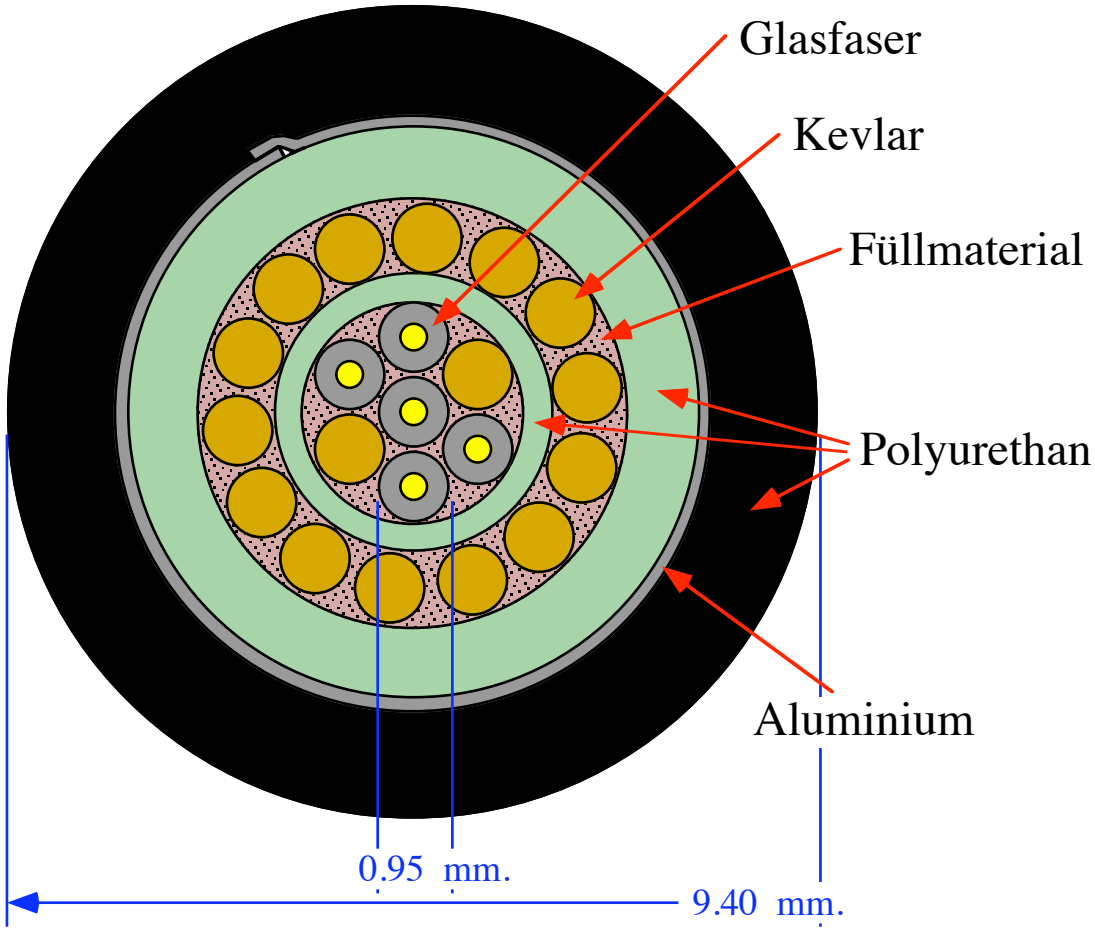


- Geführte Ausbreitung
  - elektromagnetische Wellen
  - sehr hohe Frequenzen (Mikrowellen)
  - fortlaufende Reflektion
  - hohe Energien
- Auch als Zuleitungen für Mastantennen
- 2GHz - 110 GHz

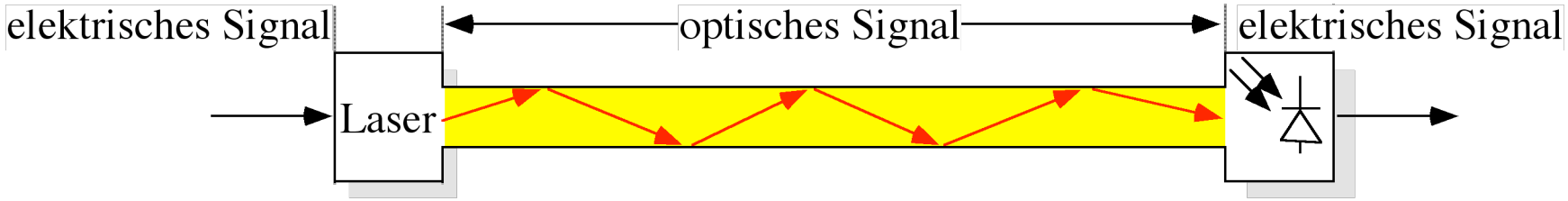
1.2.2 Lichtwellenleiter

- Licht als Datenträger
- Intensitätsmodulation
  - meist dreiwertig
- Glas oder Plastik als Leiter
- Komplexe Handhabung
  - spleißen
  - Endstücke konfektionieren
  - biegen und brechen
- Gute Störsicherheit
- Sender
  - LEDs oder Laserdioden
- Empfänger
  - Photodioden
  - Phototransistor
- Erbium-Glas verstärkt

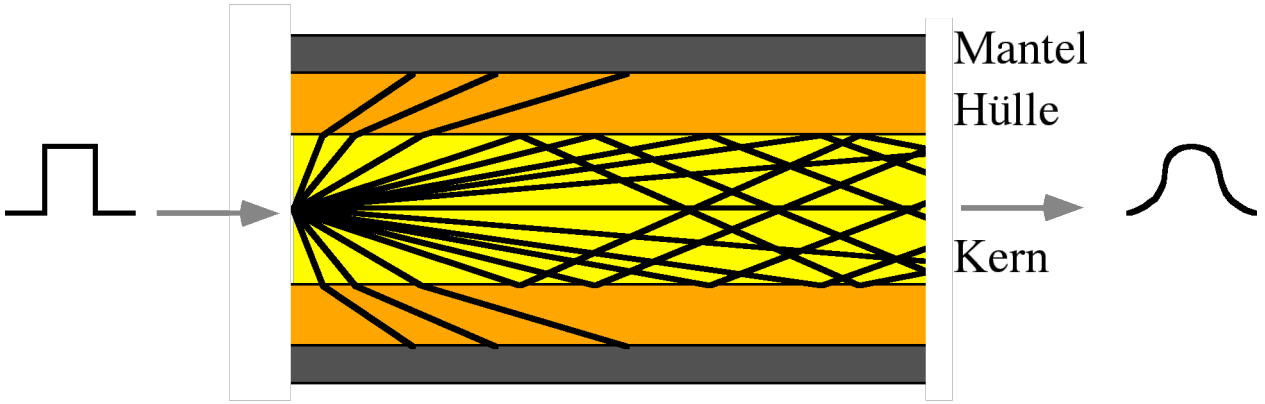




- Allgemeine Funktion

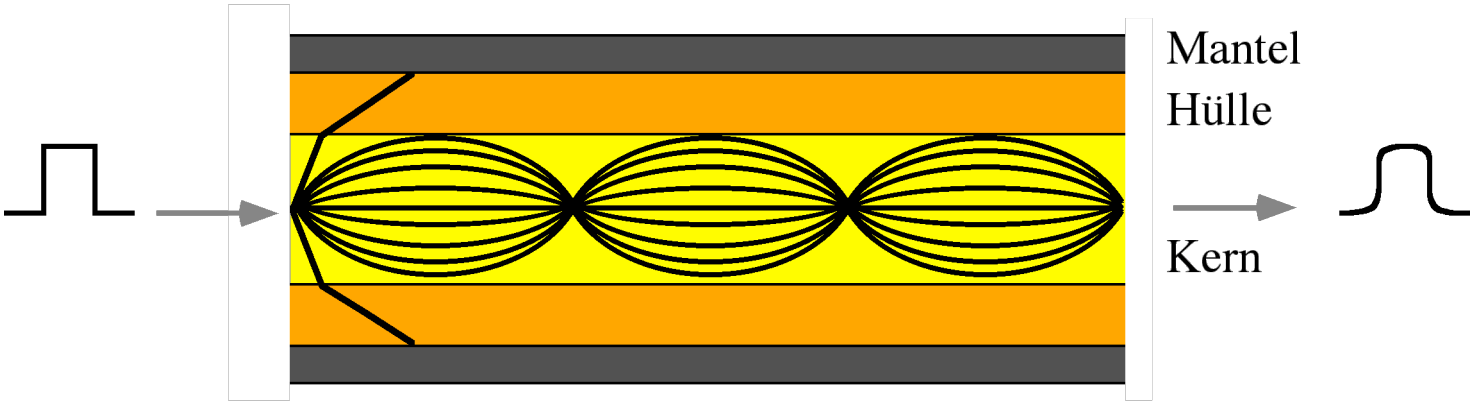


- Multimode Glasfaser mit Stufenindex
  - Brechung oder Reflexion beim Übergang Kern zu Hülle
  - Kern und Hülle aus Glas
  - Hülle mit niedrigerem Brechungsindex

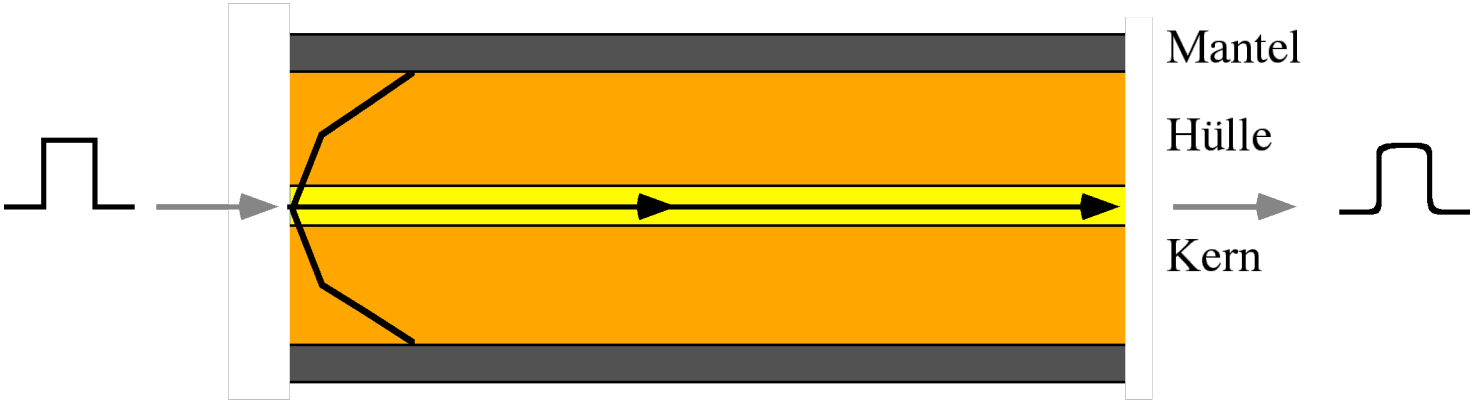


  - Absorption durch Mantel.
  - Unterschiedliche Pfadlänge.
  - Kern 50–125 μm, Hülle 125–500 μm

- Multimode Glasfaser mit Gradientenindex

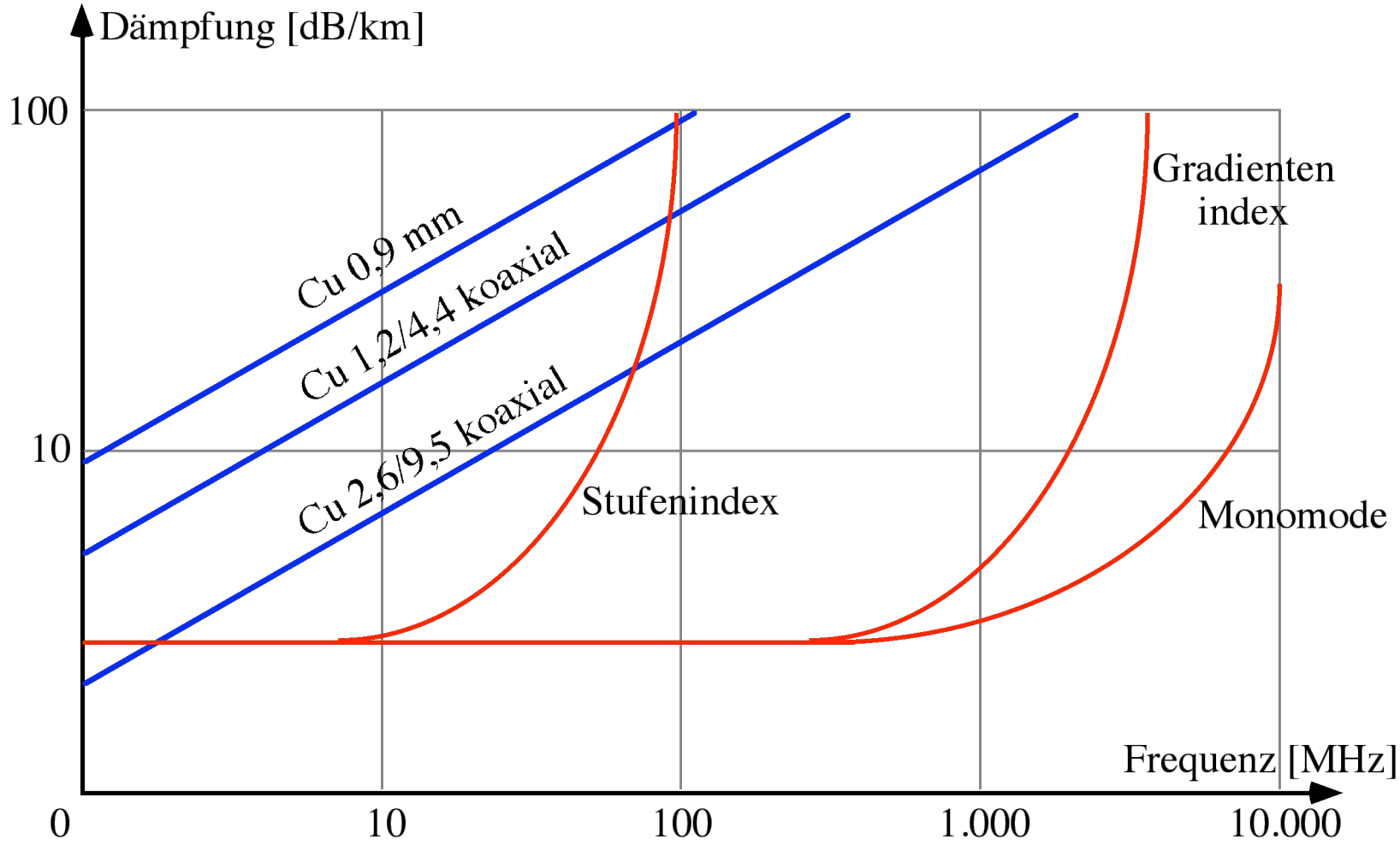


- Monomode Glasfaser (single mode)
  - Fernleitungen für Telefongespräche …



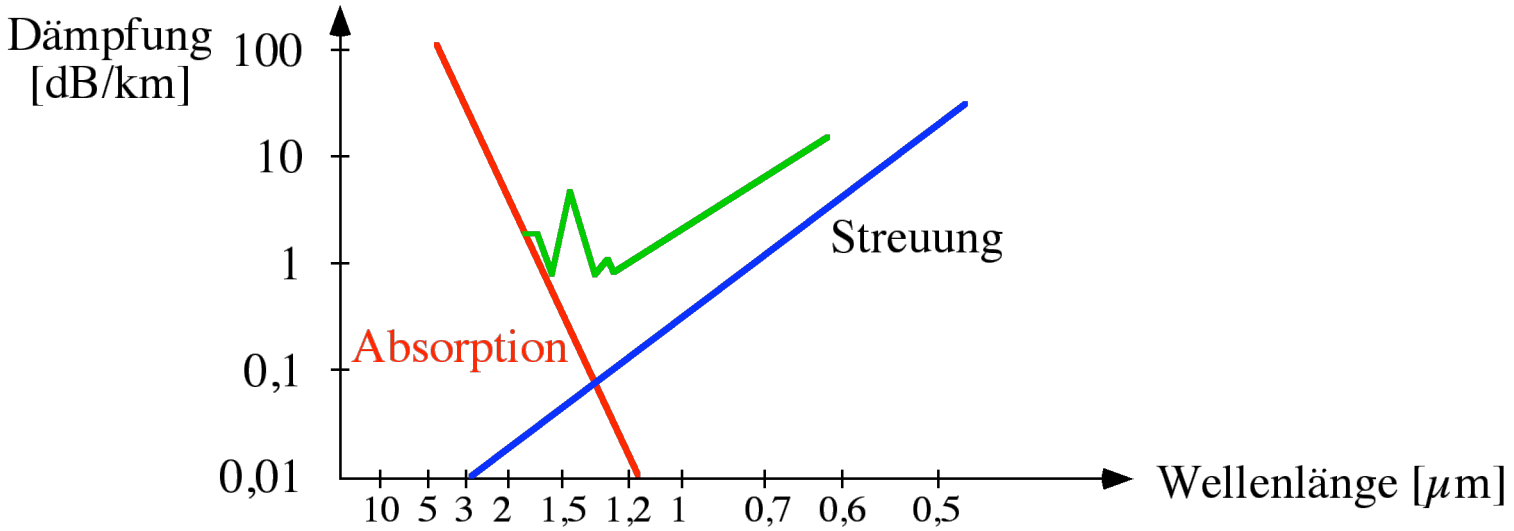
  - Kernradius ≈ Wellenlänge, 3–10 μ
  - gleiche Pfadlänge für alle Photonen
  - (Amplituden-)Modulation mit bis zu 50 GHz
  - Wellenlängen 850, 1300 oder 1500 nm.

- Dämpfungsvergleich

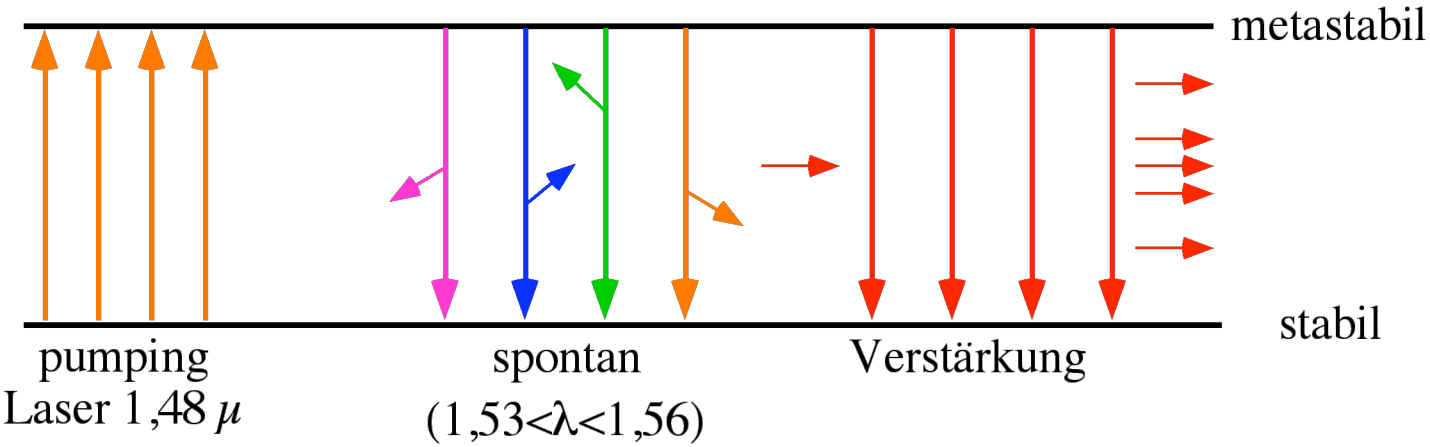


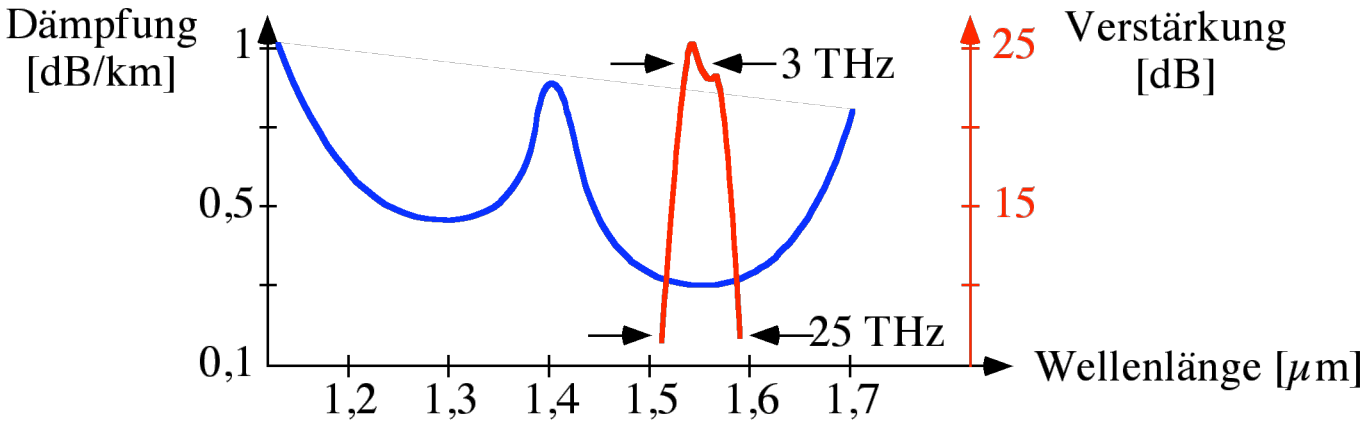
- Glasfaserdämpfung komplexer

- Dämpfung in Glasfasern
  - Absorption (thermisches Rauschen, Verunreinigungen)
  - Streuung (lokale Änderungen im Brechungsindex)
    (vgl. blauer Himmel bei Sonnenschein)
    Rayleigh Streuungs-Grenze abhängig von λ
  - Spleißstellen und Biegungen
- Absorption und Streuung können unterschiedliche Vorzeichen haben
  - gegenseitiges Aufheben
  - Lokale Minima (1,3 μm und 1,45 μm)
  - besondere Profile des Brechungsindex



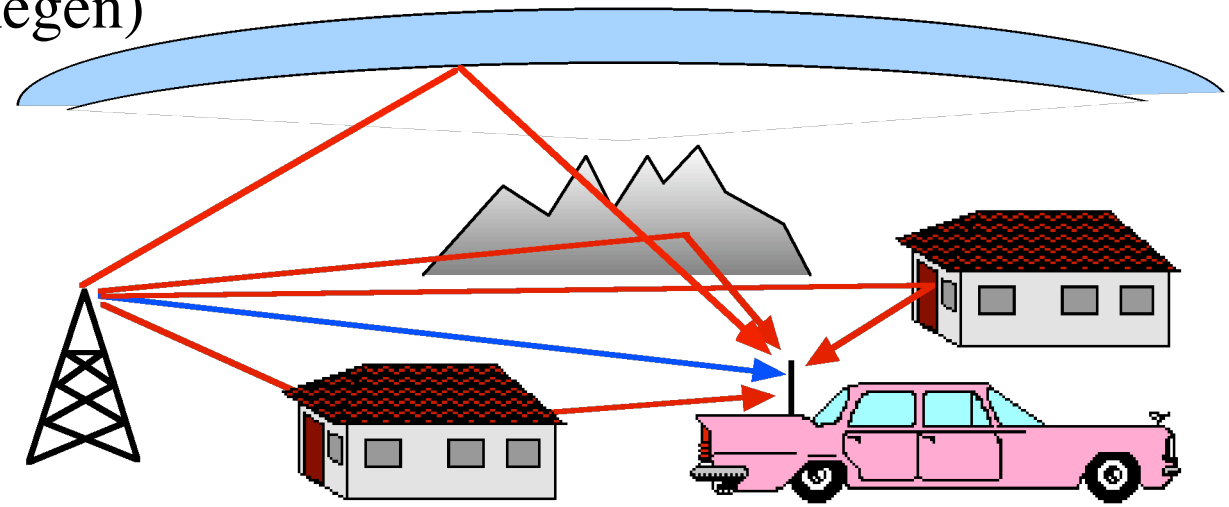
- Optischer Verstärker
  - 1987 Erbium-dotierte Glasfaser [Univ. Southampton]
  - Erbium-Ionen mit Laser (1,48 μm) in metastabilen Zustand bringen
  - spontane Rückkehr mit diffuser Photonenabgabe
  - Signalphotonen regen sofortige Rückkehr an
  - Photonenabgabe in Richtung des Signalphotons



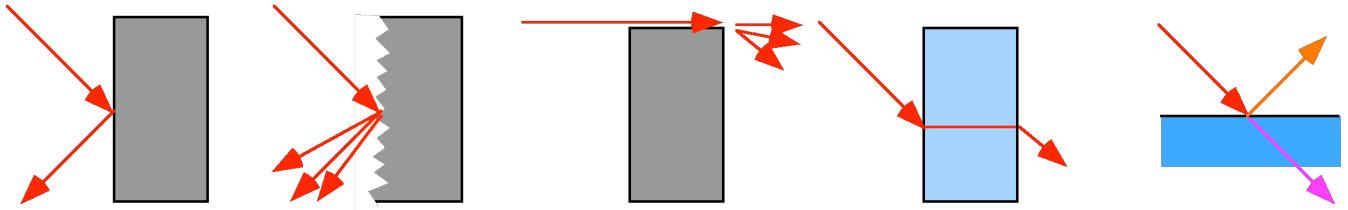


1.2.3 Funk

- Reduzierte Zuverlässigkeit:
  - Wetterverhältnisse (Schnee, Regen)
  - Atmosphärische Störungen
  - Mehrwegausbreitung
  - Abschattung

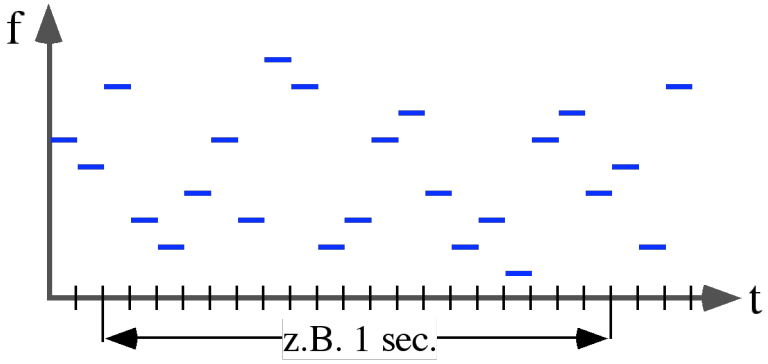


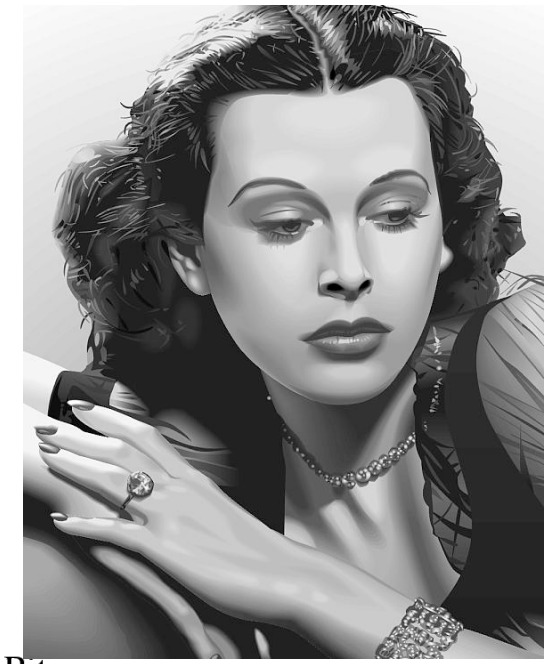
- Störungen bei der Ausbreitung des Funksignales
  - Reflexion, Beugung

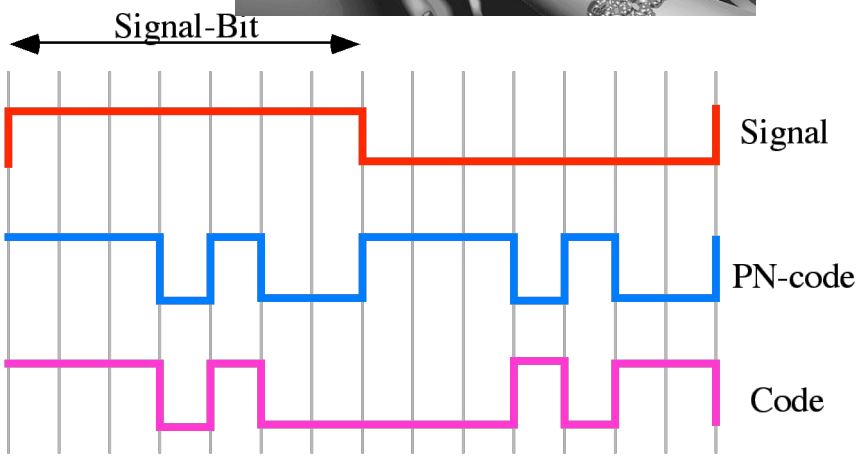


- Problem
  - Mehrwegausbreitung => Signal verwaschen
  - andere Funksignale überlagern dauerhaft

- Spread Spectrum [Hedy Lamarr, 1941]
  - Mehrfaches Frequenzband verwenden
  - Signal schmalbandig kodieren
  - auf Trägerfrequenz aufmodulieren
  - Trägerfrequenz in Sekundenbruchteilen wechseln
  - Frequenz-Sequenz aus Pseudozufallsfolge
  - Spreizen des Signales
- Frequency Hopping (FHSS)

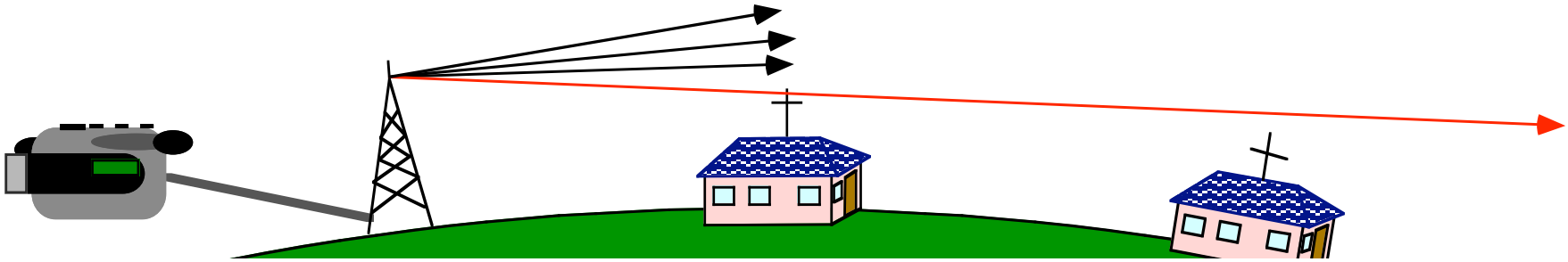




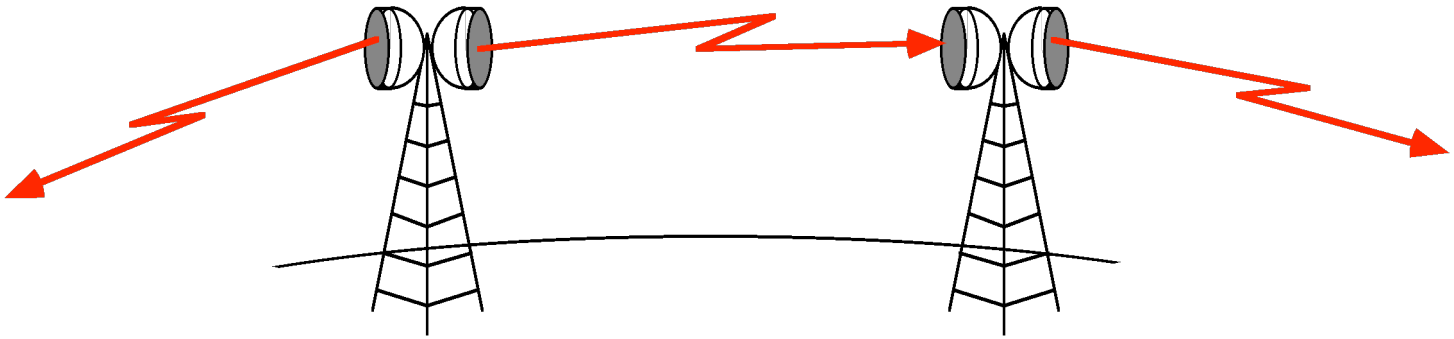


- Direct Sequence Spread Spectrum (DSSS)
  - digitale (pseudo-Zufall) Bit-Sequenz pro Datenbit
  - Empfänger 'sucht' Bit Sequenzen

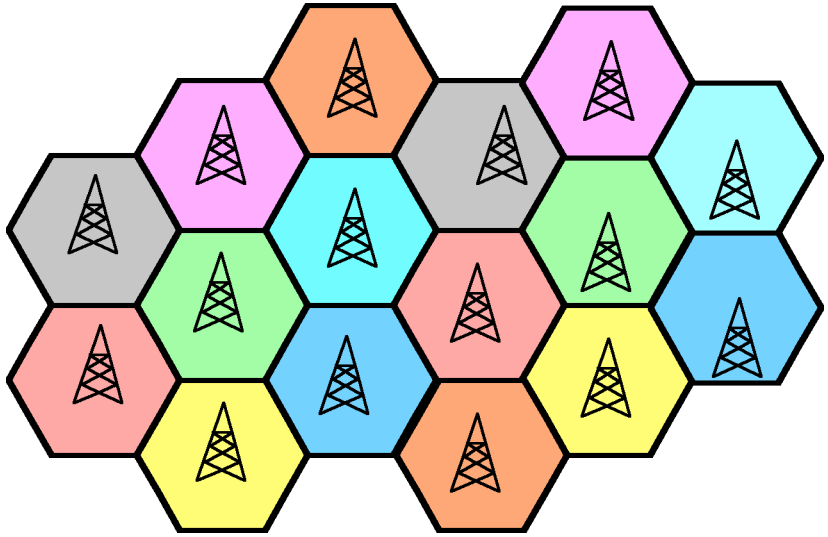
- Terrestrischer Funk
  - Betriebsfunk
  - Öffentliche Mobilfunknetze (GSM 900 und 1800 MHz)
  - Rundfunk, Fernsehen



- Ausbreitung
  - entlang der Erdoberfläche bis 700 km
  - reflektiert an Erde und Ionosphäre bis 7000 km
- Mikrowellenrichtfunk im Fernmeldenetz
  - point - to point, line-of-sight
  - AR6A: 6 GHz, 6.000 Telefonate, AT&T 1980: 800.000 km

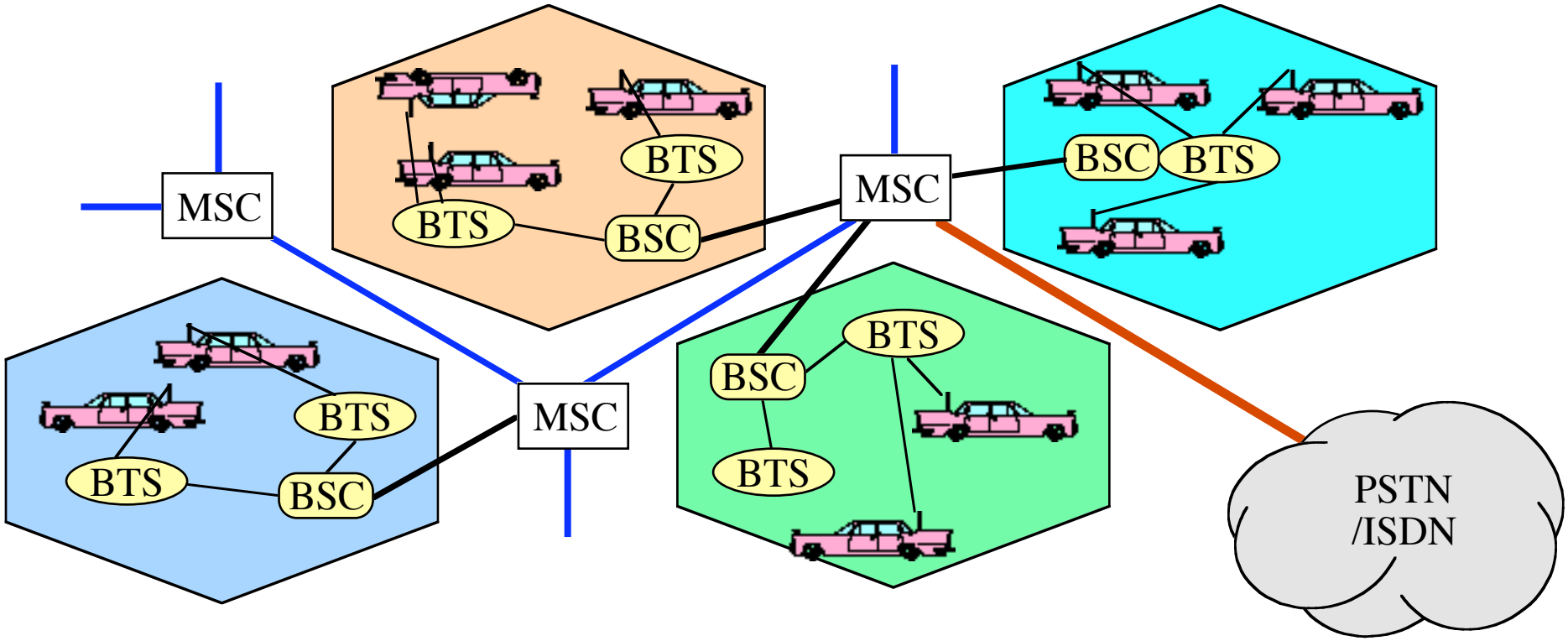


- Frequenz-Wiederbenutzung durch Zellstruktur
  - hexagonale Struktur theoretisch
  - Landschaft bestimmt Zelltopologie
  - Gerätedichte bestimmt Zellgröße
  - Antennen mit Richtcharakteristik
  - Simulationssysteme und Messungen



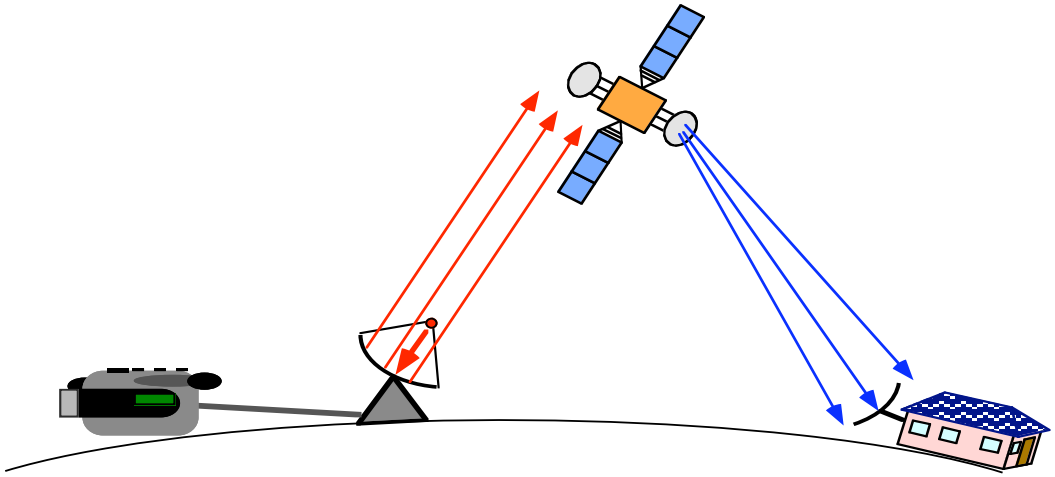
- Mobile Einheit wählt Zelle
  - Signalstärkemessung
  - komplexe Adressbindung
  - Handover während der Verbindung?
  - Spezialantennen zur 'Verfolgung' der Mobilstation

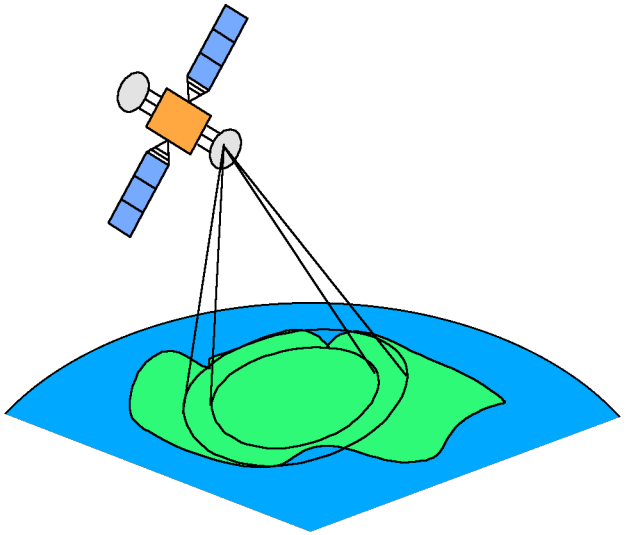
- GSM-Organisation



- Kürzelinflation
  - MS - Mobile Station
  - BSS - Base Station Subsystem
    BTS - Base Transceiver Station, BSC - Base Station Controller
  - MSC - Mobile Switching Center
  - IWF - InterWorking Function

- Direktstrahlende Satelliten
  - TV-Satelliten (Astra, 10-12 GHz; Eutelsat, …)
  - VSAT - Very Small Aperture Terminal: Telefonie
  - Inmarsat





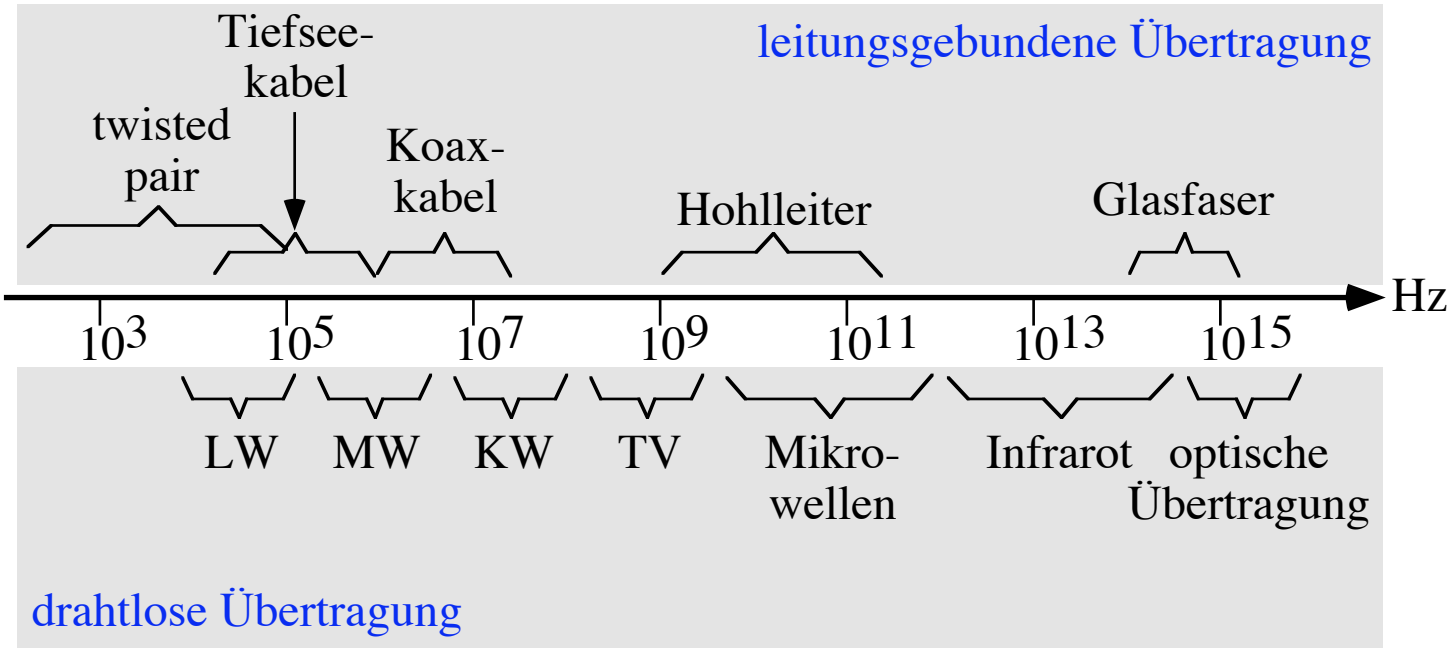
  - Iridium, Teledesic …
- Transponder
  - Empfänger und Demodulator
  - Regenerator
  - Modulation und Sender
  - Strom kein Problem

- Satelliten als Relaisstationen
  - Telstar, Intelsat, …
  - ähnlich Richtfunk
  - Fernsehrelaisstrecken, Telefonie im Multiplex
- Satellitentypen
  - GEO - Geostationary Orbiter
  - MEO: Medium Earth Orbiter
  - LEO: Low Earth Orbiter
  - Umlaufbahn bestimmt Eigenschaften und Startkosten

	GEO	MEO	LEO
Flughöhe	35.786 km	6.000 -12.000 km	700 - 2.000 km
Umlaufdauer	24 h	4 - 12 h	1,5 - 2 h
Satelliten	3 - 4	10 - 15	> 40
Zellen	< 800	~ 800	> 3.000
Verzögerung	> 300 ms	150 ms	< 50 ms

- Iridium: 66, 6 Bahnen, 780 km, 10 min sichtbar, 4.000 Kanäle
- Teledesic: 840, 21 Bahnen, 700 km, 2.500 Kanäle

- Einteilung des Spektrums



- Regulatorische Vorgaben
  - bei der leitungsgebundenen Übertragung frei
  - Abstrahlung reguliert (EMV)
  - bei drahtloser Übertragung reguliert durch Behörden

1.3 Signalbildung und Modulation

- a) Analoge Daten, Analoges Signal
  - klassische Technik
  - direkte Übertragung
  - Modulation zur Mehrfachausnutzung
  - nicht in dieser Vorlesung diskutiert
- b) Digitale Daten, Digitales Signal
  - Kanal ohne wesentliche Frequenzbeschränkung
- c) Analoge Daten, Digitales Signal
  - Bsp. Telefonie im ISDN
  - Kostenvorteile digitaler Vermittlungstechnik
  - Multimedia-Kommunikation
  - Transport mit b) oder d)
- d) Digitale Daten, Analoges Signal
  - Kanal mit echter Frequenzbeschränkung
  - Medien, die nur analoge Signale übertragen
  - Telefon, Glasfaser, Funk, …

- Digitale Daten, Analoges Signal
  - Möglichst viele Datenbits übertragen
  - Übertragungskanal hat eingeschränkte Bandbreite
  - Telefonleitung 300 .. 3400 Hz
  - Fernsehkanal 7 MHz
- Energie auf Frequenzen außerhalb des übertragenen Bandes
  - verloren und stört andere
  - Energiespektrum des Signals muß Übertragungskanal entsprechen
- Ausgestrahlte Leistung > Grundrauschen im Kanal
- Taktrückgewinnung
  - separate Leitung
  - aus dem Signal
- Basisband
  - (fast) unendliches Spektrum bei digitalen Signalen
- Modulation
  - bandbreitenbeschränkte Kanäle, Mehrfachausnutzung
  - Trägerfrequenz

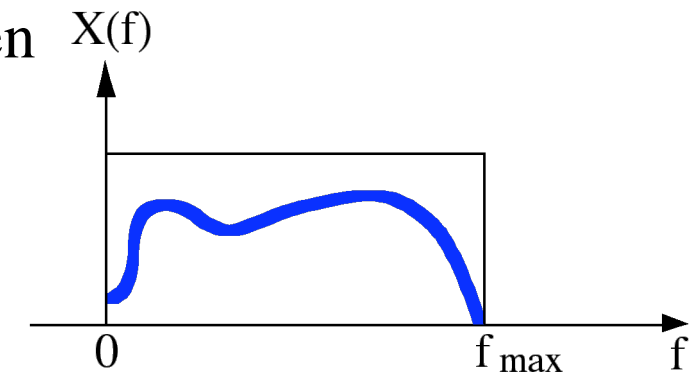
1.3.1 Abtasttheorem und Kanalkapazität

- Abtast-Theorem [Whittaker, 1915/1929, Borel 1897]
  - wie häufig muss ein Signal abgetastet werden?
  - eindeutige Identifikation/Rekonstruktion
- Minimale Abtastfrequenz > 2 f_{max} Abtastwerte pro Sekunde
  - Bsp.: $\sin(2\pi t)$: 1 Hz, 2 Abtastwerte: $\sin(0) = 0$, $\sin(\pi) = 0$
- Wieviele Symbole/sec über nach oben frequenzbeschränkten Kanal?

$$v(t) = a_0 + \sum_{i=1}^{n} a_i \sin i\omega t + \sum_{j=1}^{n} b_j \cos j\omega t$$

- Kanal überträgt alle durch f_{max} beschränkten Fourierspektren
- Beschreibung durch je einen Satz Fourierkoeffizienten
  - endliche Menge bei periodischer Funktion
  - unendliche Menge bei aperiodischer Funktion
  - mehr Spektren werden nicht übertragen
  - mehr Koeffizienten werden nicht übertragen

=>Kanal kann maximal $2f_{max}$ Symbole/sec übertragen

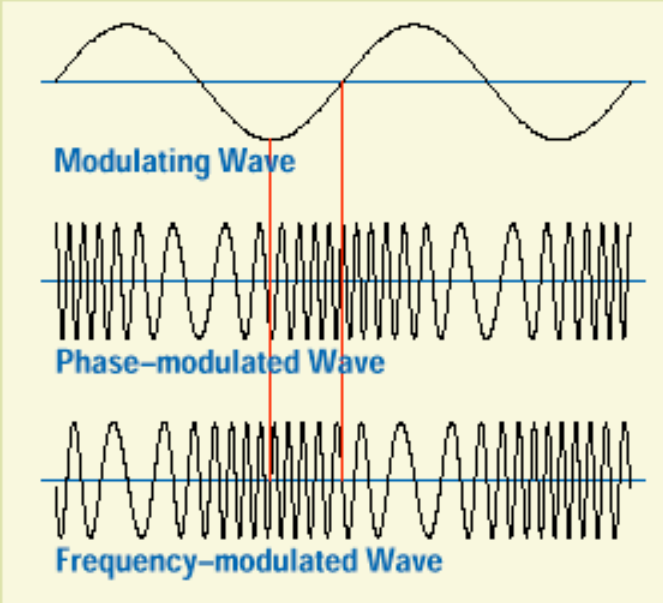


- Annahmen
  - mehrwertige Symbole bzw. Abtastwerte: 2, 3 Bit, 1 Byte, 12 Bit …
  - Symbolrate bzw. Bitrate
  - Symbole mit mehreren Bits erhöhen die Menge der Information
  - Abtastrate gemessen in Baud (Symbolen pro Sekunde)
- H. Nyquist, 1924
  - rauschfreier Kanal
  - V= Anzahl diskrete Signalstufen
  - max. Datenrate = $2 * f_{max} * \log_2 V$ [bit/s]
- Beispiel
  - f_{max} = 3.000 Hz, 4 Signalstufen
  - D_{max} = 12.000 bit/s
- => Viele Signalstufen erhöhen Datenrate
- C. Shannon, 1948
  - Kanal mit zufälligem Rauschen
  - wieviele Signalstufen kann man noch sicher unterscheiden?

- Signal/Rauschverhältnis S/N begrenzt die Anzahl der Bits pro Symbol
- Anzahl übertragbare Bits pro Sekunde:
  - Shannon Limit = $f_{max} * \log_2 (1 + S/N)$ [bit/s]
  - S/N = 0 : keine Information
  - S/N = 1 : ~ f_{max} Bit/sec
- Telefonleitung:
  - analog: 24 dB => S/N ~ 250; f_{max} = 3000 Hz => 24.000 bit/s
  - heute begrenzt Quantisierungsrauschen von G.711 (μ-law)
  - theoretisch: 39,5 dB, praktisch: 35 dB => S/N = 3162
  - => 34.882 bit/s (theoretisch 39.365 bit/s)
- Agenda
  - wie moduliert und codiert man auf einer vorgegebenen Leitung?
  - Modems: wie kommt man auf Telefonleitungen nahe an 35.000 bit/s?
  - 56 kbit/s?
  - wie kann man G.711 umgehen, um höhere Datenraten zu erreichen?

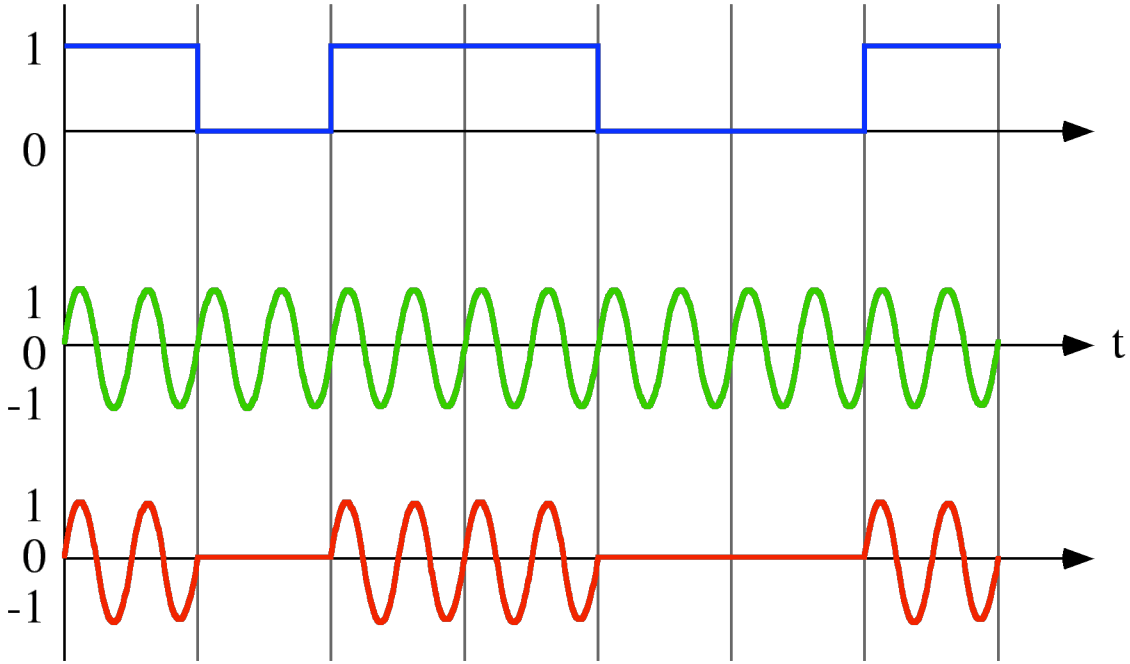
1.3.2 Modulation und Modems

- Modulation
  - Charakteristika einer Trägerfrequenz ändern
  - proportional zur Amplitude des Basisbandsignales
  - Empfänger mißt Charakteristika
- Amplitude und Winkelposition (Frequenz, Phase)
  - Wellengleichung $e(t) = A \cos(\omega t + \varphi)$
  - Frequenz: Abweichung von der Trägerfrequenz [Armstrong, 1933]
  - Phase bezogen auf t=0

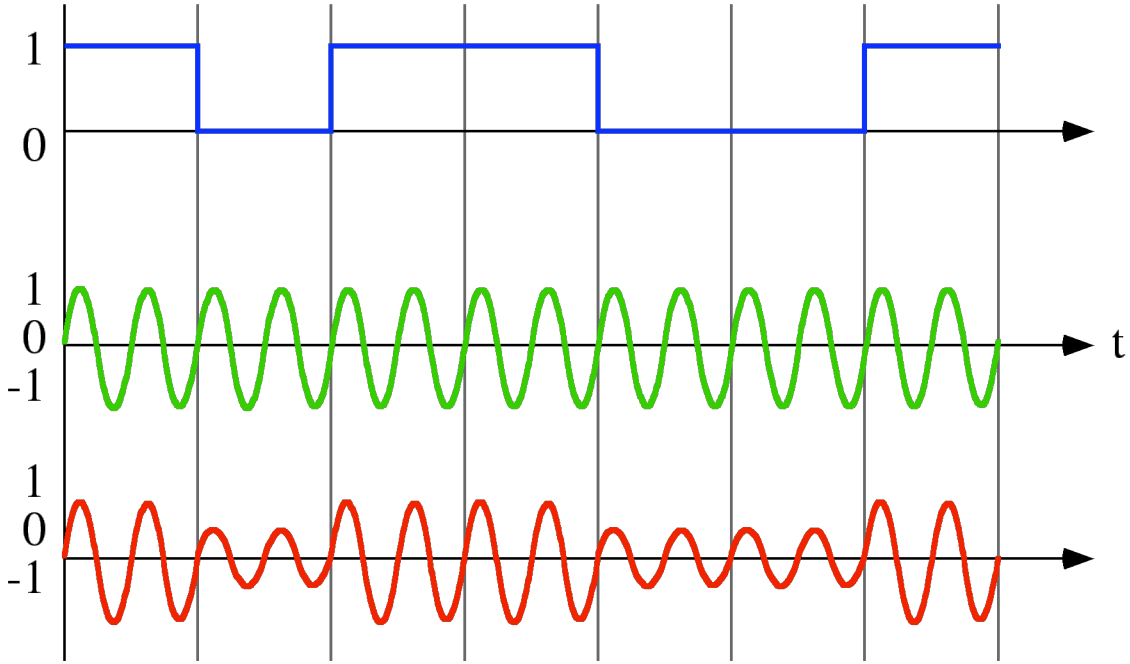




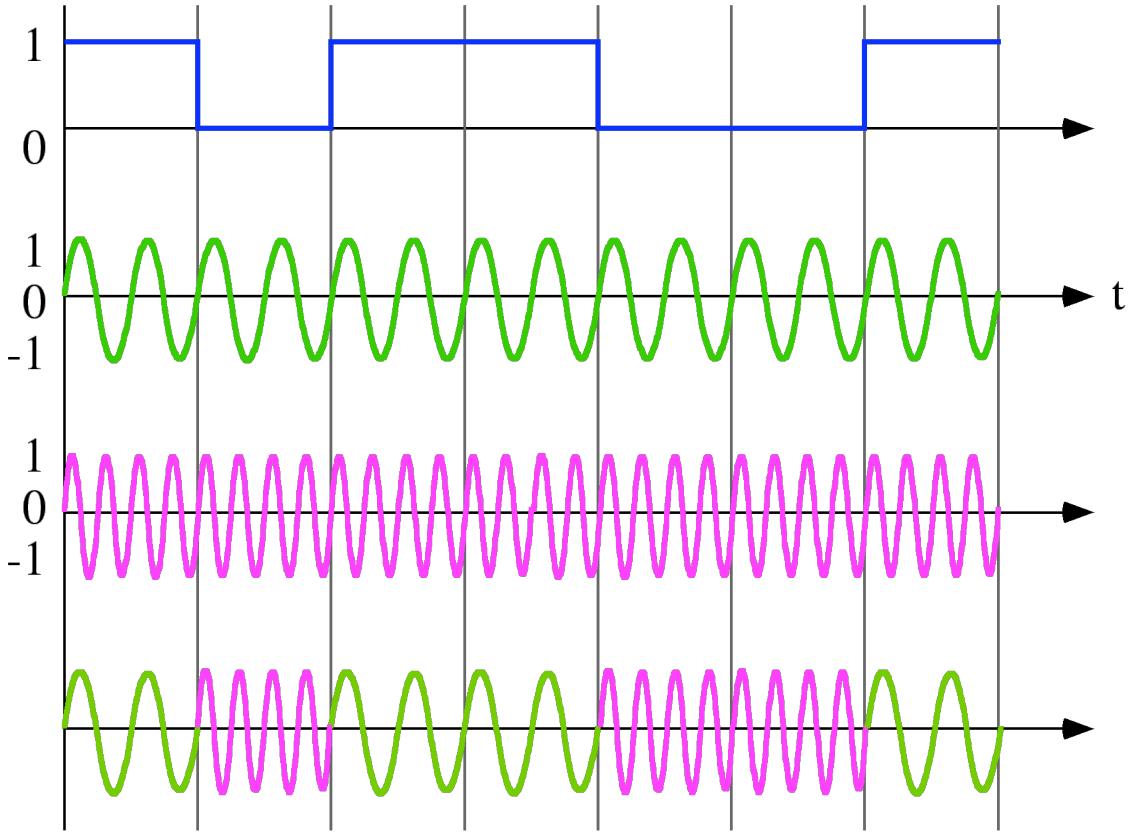
- 'on-off' Keying (ooK)
  - keying: Sender ein/ausschalten bzw. umschalten
  - Umschaltrate = Symbolrate = Bitrate



- Amplitude Shift Keying (ASK, Amplitudenmodulation)
  - Amplitude umschalten (Intensitätsmodulation)
  - für analoge und digitale Daten
  - ständig Trägerfrequenz senden/empfangen
  - Umschaltrate = Symbolrate



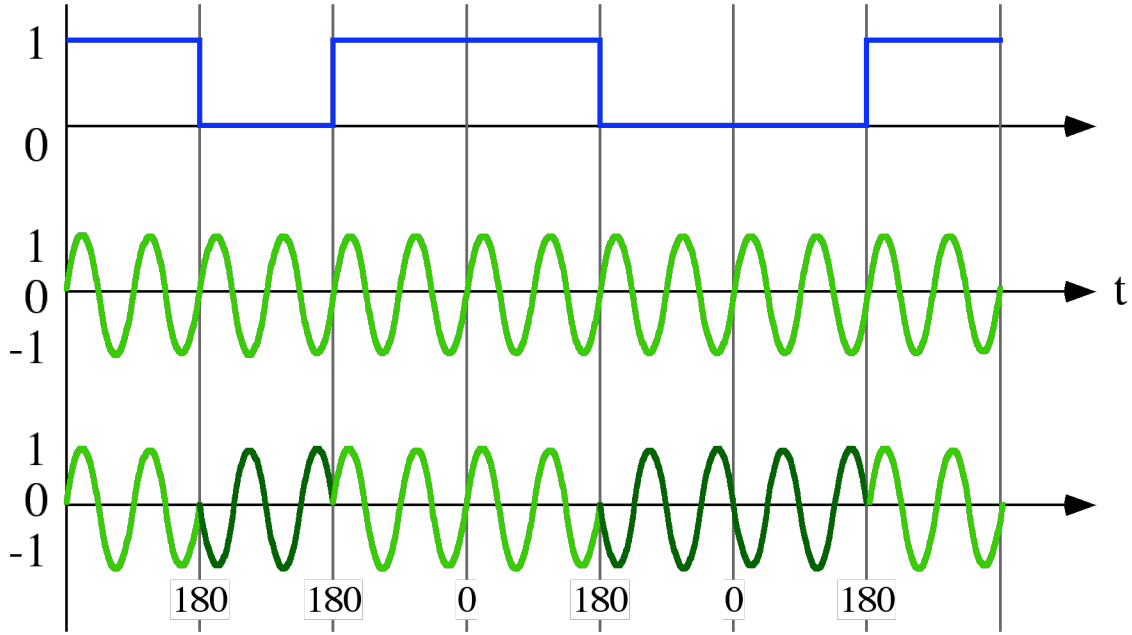
- Frequency Shift Keying (FSK)



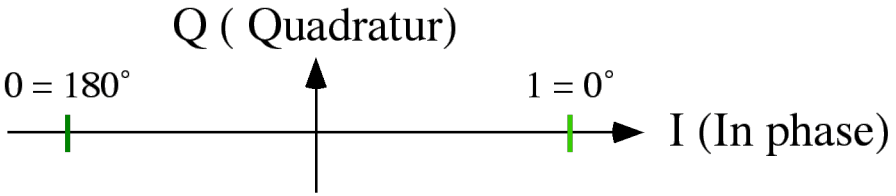
- V.21-Modem: Space (0) and Mark (1)
  - 980 und 1180 für Originate-Seite (Bell 103: 1070 und 1270)
  - 1650 und 1850 für Answer-Seite (Bell 103: 2025 und 2225)
  - Echo-Unterdrückung abschalten
  - 300 bps vollduplex

- Zwischenfrage zu FSK:
  - 2 gleichbreite Bänder: A und B
  - A: (x Hz, x+n Hz),
  - B: (mx Hz, mx + n Hz)
  - Behauptung: in B höhere Symbolrate möglich (bzw. warum nicht?)
  - Argument: Extremwerte zählen geht in B schneller
- Nyquist: max. Datenrate = $2 * f_{max} * \log_2 V$ [bit/s]
  - $2 f_{max}$ ist die maximale Symbolrate
  - unabhängig vom Frequenzbereich
- Signal moduliert Abweichung von der Trägerfrequenz
  - Bandbreite modulierendes Signal ≤ Frequenzband
  - Unterscheidung 10.100Hz und 10.200Hz ~ 100Hz und 200Hz

- Phase Shift Keying (PSK)
  - 180° = Space (0), 0° = Mark (1)
  - erfordert Referenzsignal



- Phasendiagramm (Länge = Amplitude)



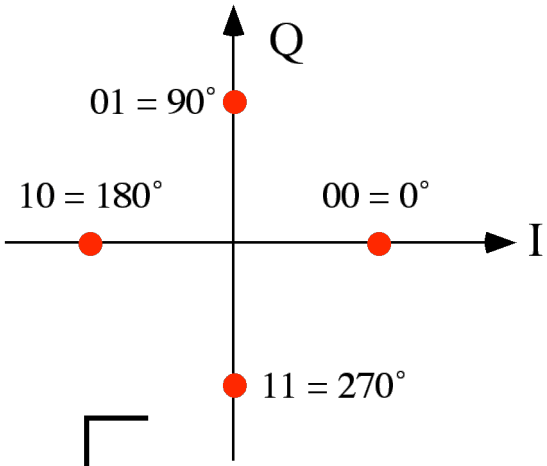
- Differentielles PSK: 'Taktgewinn' aus ständigem Phasenshift

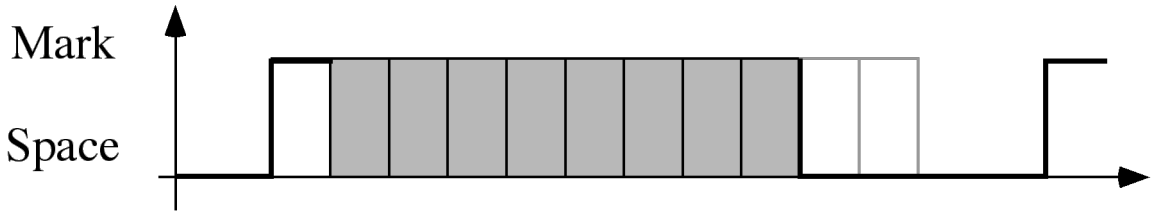
Telefon Modems

- V.21 / Bell 103 siehe oben
- DPSK (V.22)
  - DiBit PSK (auch: QPSK)

DiBit	00	01	10	11
Phasen-Shift	90°	0°	180°	270°

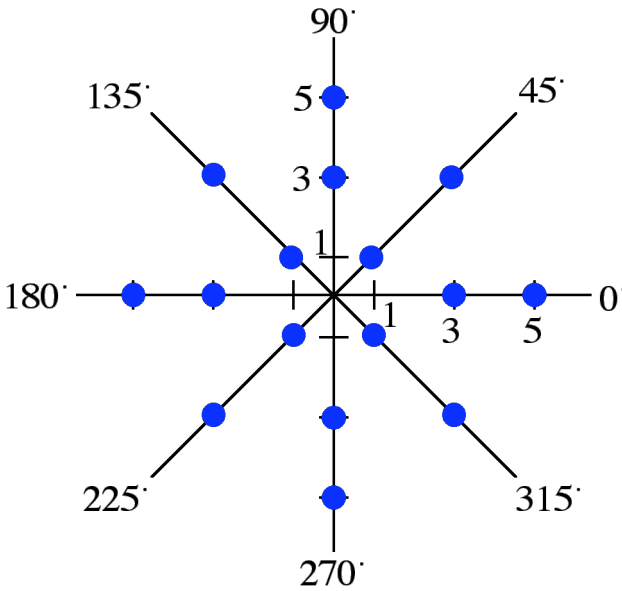
  - 600 baud, 1200 bit/s
  - Bell 212A: 600 bit/s
  - Fallback nach FSK
  - V.22bis mit 2400 bit/s
- Asynchroner Betrieb
  - Kennzeichnung der Bytes wie V.24
  - Startbit + Stopbit(s)

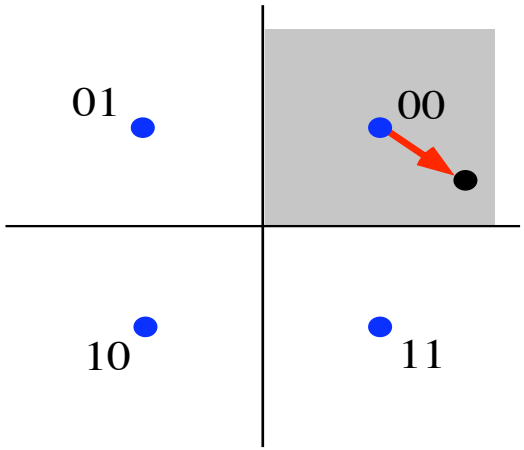




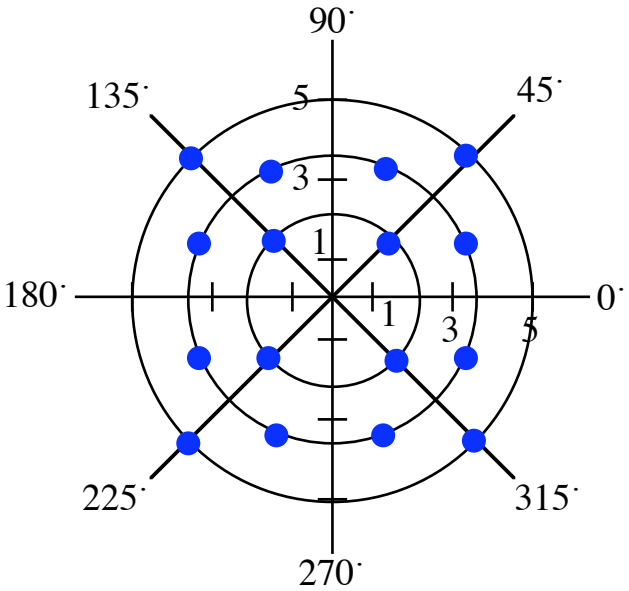
- Funktionen des Senders
  - Takt erzeugen
  - Scrambler (genug 0/1 Übergänge)
  - Modulator
  - D/A-Wandler und Tiefpass
  - Equalizer präkompensiert für Kanal (Amplitude und Delay)
- Funktionen des Empfängers
  - adaptiver Equalizer gesteuert vom Demodulator
  - Taktrückgewinnung
  - Demodulator
  - Descrambler

- Digitales Backbone-Netz
  - Signal auf weiten Strecken als PCM übertragen
  - nur Vermittlung und Anschlußleitung analog
- Quadrature Amplitude Modulation QAM
  - Phasenmodulation und Amplitudenmodulation
- Amplitude und Phase ergeben Vektor
  - Konstellation
  - Enscheidungsregionen
  - Fehlervektor minimieren

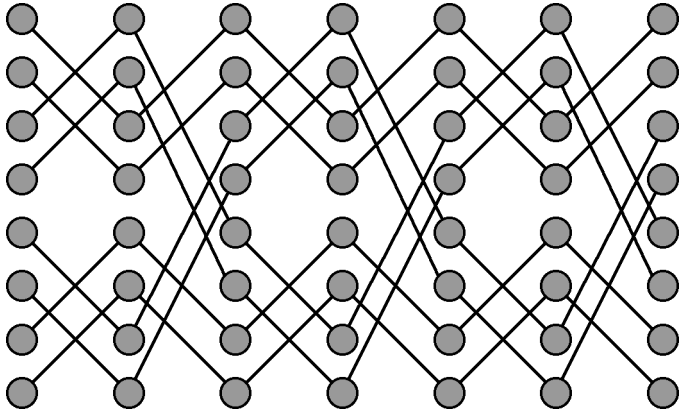




- z.B. V.29 (9600 bit/s) halbduplex, z.B. in Fax-Geräten
  - 4 Bit pro Symbol
  - 1. Bit bestimmt Amplitude, 2,3,4 wählen Phase
  - Hilfskanal moduliert Fehlervektor
- V.32: Daten mit 9.600 bit/s
  - 2400 Baud, 4 bit pro Symbol
  - 2400 Hz, 1800 Hz Trägerfrequenz => 600 - 3000 Hz
  - Echounterdrückung für Vollduplexbetrieb
  - 16-Konstellation
  - 32-Konstellation mit TCM

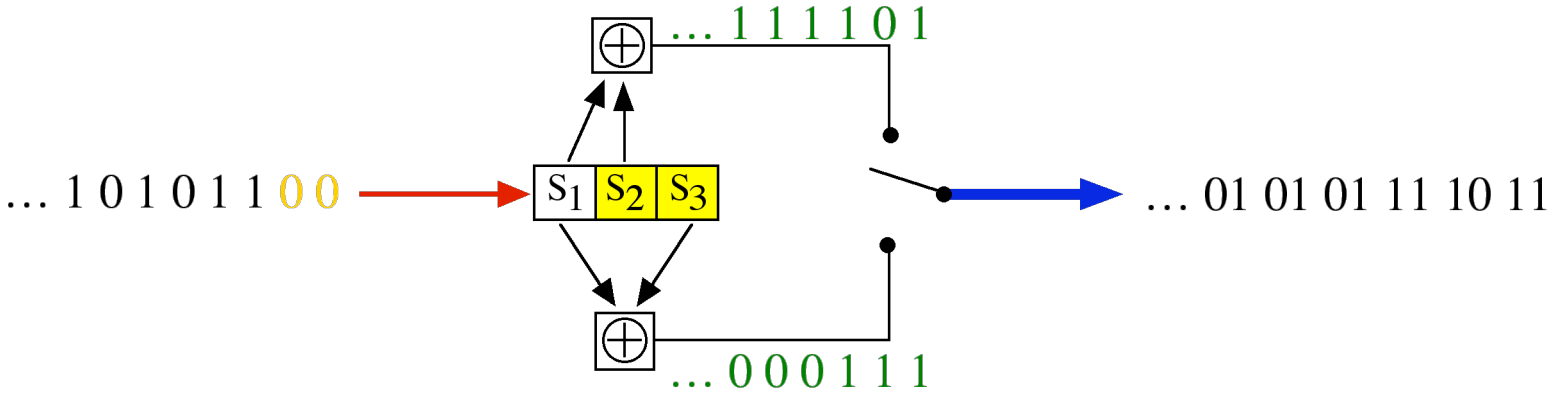


- Trellis Coded Modulation [Gottfried Ungerböck, IBM Rüschlikon]
  - 6 dB Gewinn durch Kodierung
  - 3 dB Verlust durch 1 Bit mehr pro Symbol
  - trellis = dt. Spalier

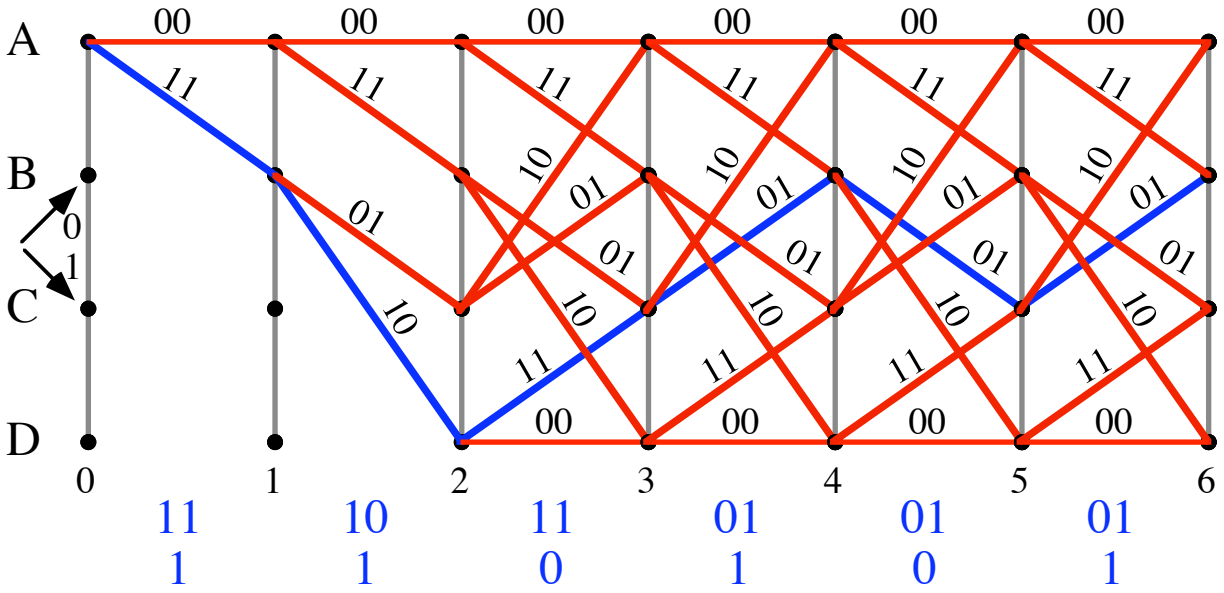


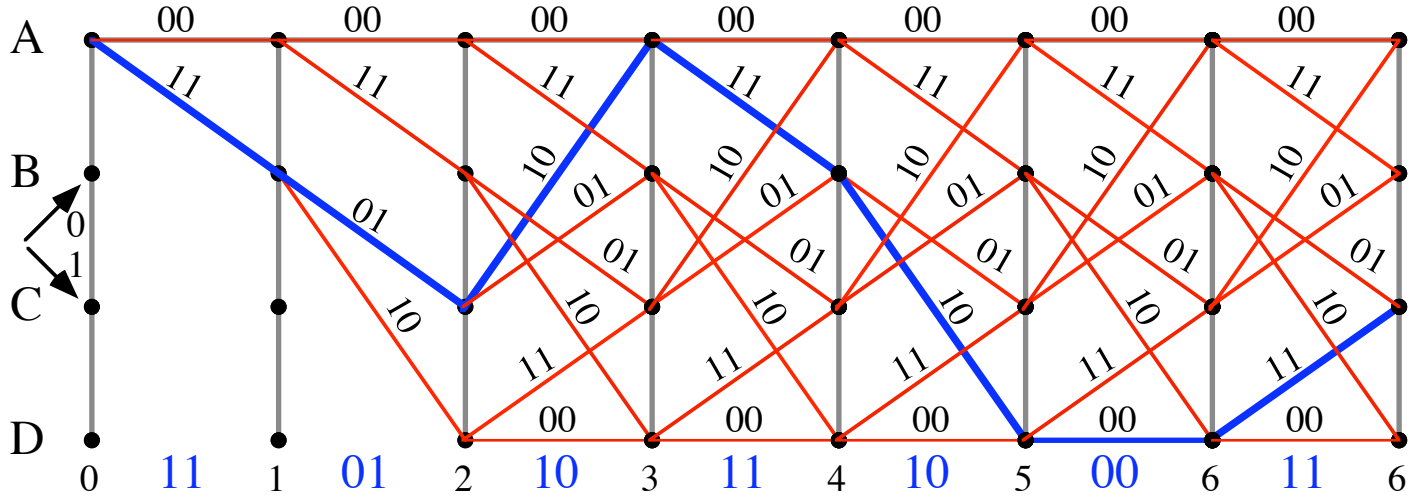
- Zustandsautomat
  - Zustand -> Folgesymbol
  - mehr Punkte in der Konstellation als gültige Symbole
  - Zustand definiert Untermenge der Folgepunkte
  - Bsp: 4 Punkte, Eingabe (0 / 1) wählt aus 2 Folgepunkten einen aus
  - V.32: 32 Punkte (5 Bit), 16 Codeworte (4 Bit), r = 4/5

- Faltungskodierer (hier r = 0,5)

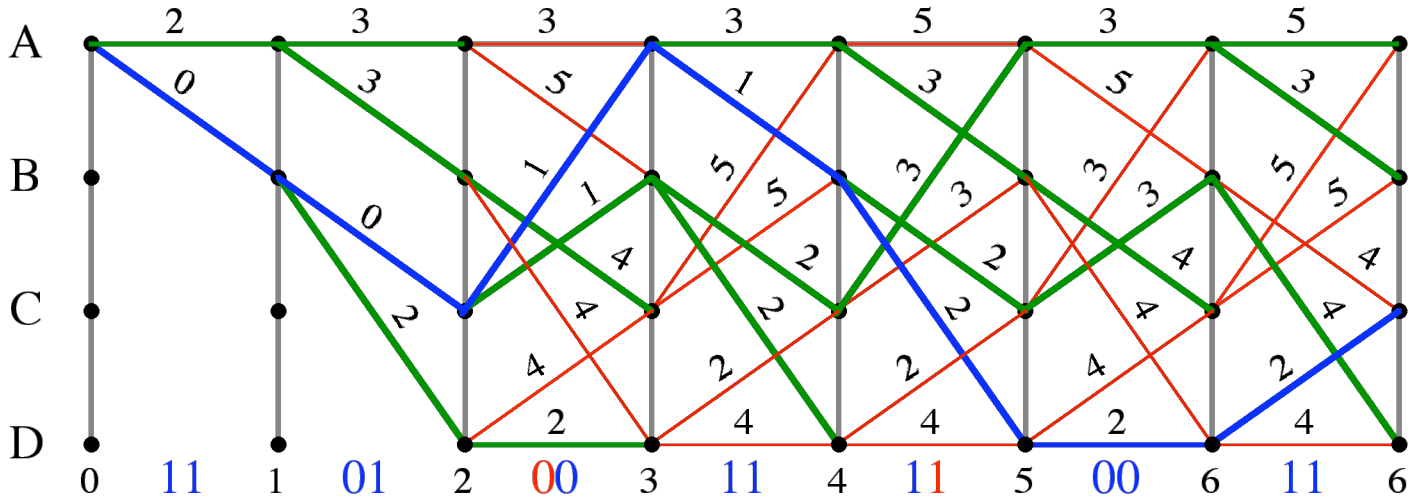


- Trellis-Diagramm

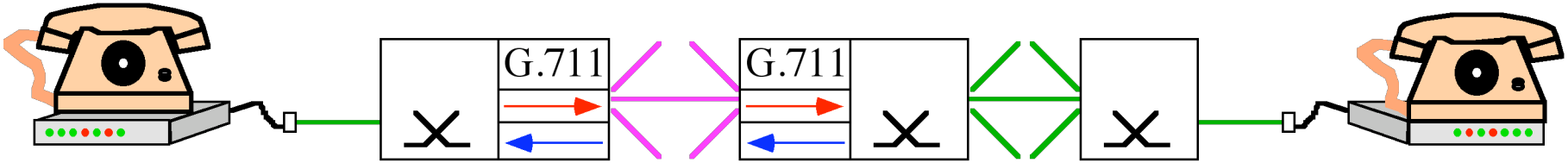




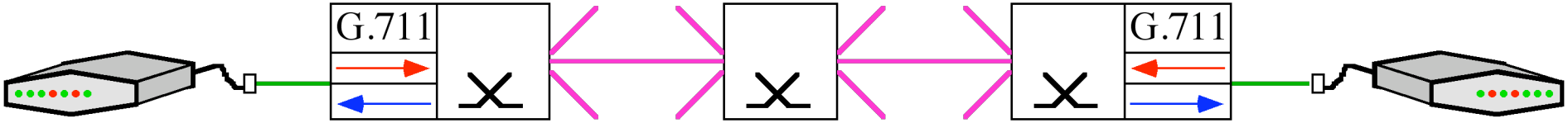
- 11 01 10 11 10 00 11 Störung: -> 11 01 00 11 11 00 11 …
- Empfänger: entscheidet rückwärts
  - minimiert Distanz zum nächsten Punkt in der Konstellation
  - Viterbi: suche Pfad mit minimalem Gewicht



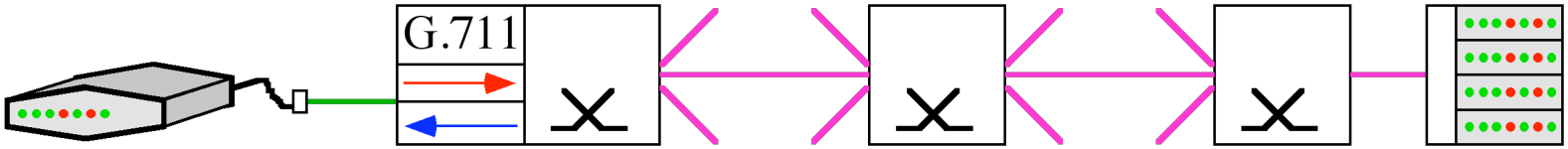
- V.32
  - (Q1,Q2, Q3, Q4) -> (Y0, Y1, Y2, Q3, Q4)
  - Viterbi-Pfadlänge ≈ 5*Länge des Faltungskodierers = 15 ≈ 16
- Telefonleitungen werden besser
  - digitale Fernleitungen
  - digitale Vermittlungen
- Telefonnetzwerk klassisch
  - Rauschen in analogen Vermittlungen und Leitungen
  - G.711 Quantisierungsrauschen



- Telefonnetzwerk im ISDN-Zeitalter

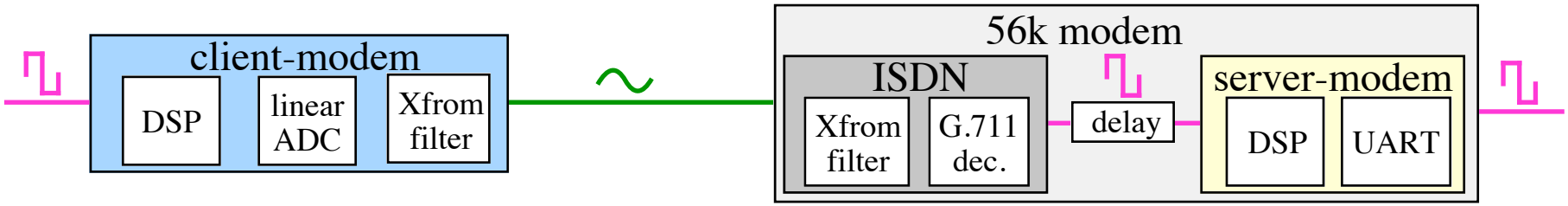


- V.34 (V.fast): 28.800 bzw. 33.600 bit/s
  - 3429 Baud, 9 bit, 2048 Konstellation
  - dreidimensionale Trellis-Codierung
  - Trägerfrequenz 1959 Hz
- Wie umgeht man den G.711-Quantisierer?
- Telefonnetzwerk für ISP



  - eine Richtung ohne G.711 A/D (ISP -> Subscriber)
  - G.711 D/A ohne Quantisierungsrauschen
- Übertragungsweg fast vollständig digital
  - PCM von der 1. Vermittlung bis zum ISP
  - nur Teilnehmeranschlußleitung analog
- G.711 D/A an der Teilnehmerleitung 'Teil' des 56k-Modems
  - analoge Pulsform vorhersagbar ähnlich D/A im V.34 Modem
  - Codec-Eigenheiten, Filter vor der Leitung, …
  - Vorschriften (BAPT, FCC, …)

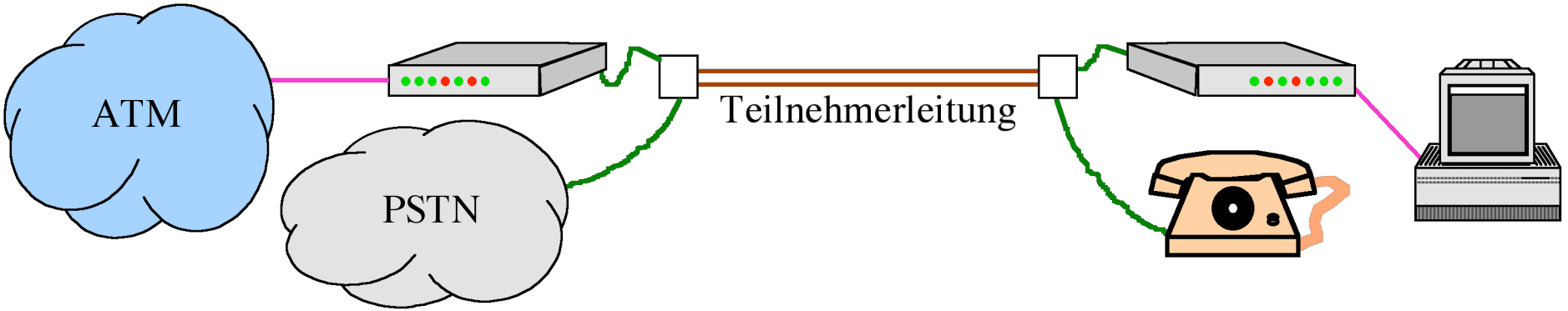
- 56 kbit/s Modems (V.90)
  - spezielle Netzwerkkonfiguration
  - 56 kbit/s downstream, max. 33.600 upstream
- Leitung aus der 'analogen' Sicht des 56k Modems



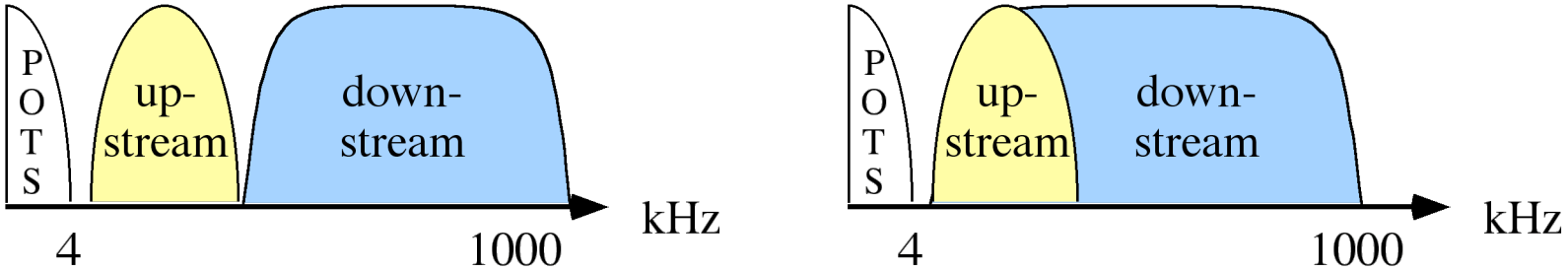
- Puls-Amplituden-Modulation PAM
  - vorgegeben durch 'Mitbenutzung' des G.711-DAC
  - 56.000 kbit/s, 8.000 samples/s => 128 Signalwerte
  - G.711 erzeugt Treppenfunktion aus diskreten PCM-Werten
  - Filter in der Vermittlung glättet Treppenfunktion
  - linearer ADC sampled glatte Treppenfunktion
  - DSP sucht und interpretiert Treppenstufen als diskrete Werte
- SNR auf Teilnehmerleitung und im client-modem
  - 14 bit ADC: 86 dB
  - Shannon(56 kbit/s, 3800 Hz) = 45 dB

Modems für die Teilnehmeranschlußleitung

- Kupferkabel von der Vermittlung zum Teilnehmer
  - meist keine aktiven Komponenten, evtl. Spule
  - 75% < 2 km; 98% < 8 km
- HDSL - High Speed Digital Subscriber Loop
  - 2 MBit/s (high: > ISDN)
  - ETSI TM3
- ADSL - Asymmetric Digital Subscriber Loop
  - viele Applikationen sind asymmetrisch: Upstream vs. Downstream
  - POTS auf demselben Kabel
  - 5,5 km / 0,5 mm Kabel / 1,5 Mbit/s / 16 kbit/s
  - 2,7 km / 0,4 mm Kabel / 6,1 Mbit/s / 576 kbit/s

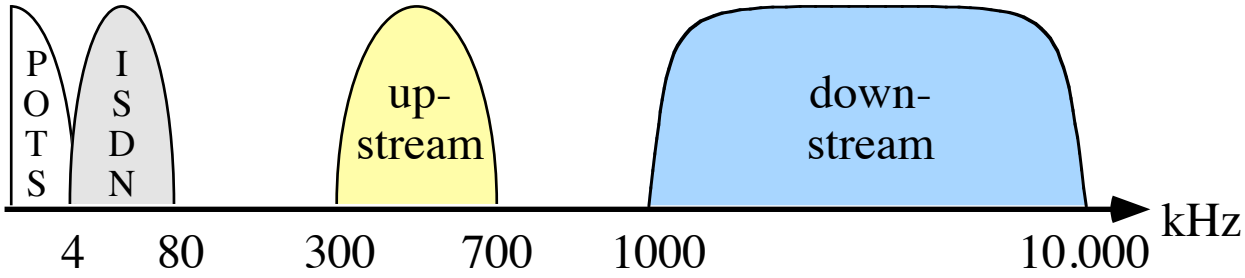


- downstream n * 1536 bzw. n * 2048 kbit/s
- upstream 16, 64, 160, 384, 544, 576 kbps
- Twisted Pair
  - 90 dB bei 1 MHz und 5 km
  - typische Telefonleitung mit 22 Spleiss-Stellen [Bellcore]
  - FEC Fehlerkorrektur
  - DMT: Discrete Multitone ~240 * 4 kHz Sub-Kanäle
  - pro Kanal QAM
- Frequenzmultiplex oder Echo-Unterdrückung



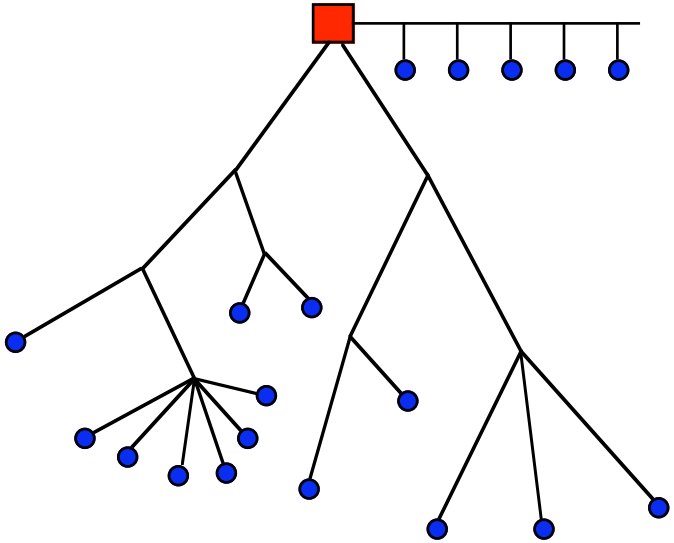
- Erfolgreich
  - ADSL-Forum
  - Standardisierung in ANSI T1E1.4: T1.413
  - Hersteller z.B. Orckit, TuT, Amati, NetSpeed…

- VDSL - Very high rate Digital Subscriber Loop
  - Bruchteil der ATM-Rate 155.52 MBit/s
  - 1500 m: 12.96 MBit/s, 1000 m: 25.92 MBit/s, 300 m: 51.84 MBit/s
  - asymmetrisch: upstream 1.6, 2.3, 19.2 MBit/s
  - symmetrisch: echo-cancellation
  - Platz für Telefon und ISDN

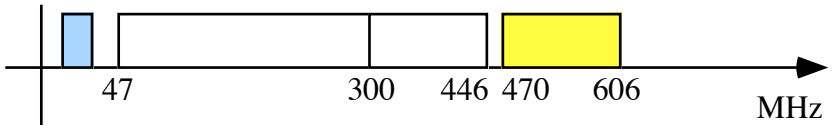


- Übertragungscode
  - CAP Carrierless AM/PM (QAM, QPSK upstream)
  - DMT Discrete Multitone basiert auf DFT von Einzel-Carrieren
  - DWMT Discrete Wavelet Multitone
  - SLC Simple Line Code: Basisband gefiltert
  - FEC: Reed Solomon
- Mehrgerätekonfigurationen
  - NT aktiv: Ethernet-Hub, NT passiv: FDM

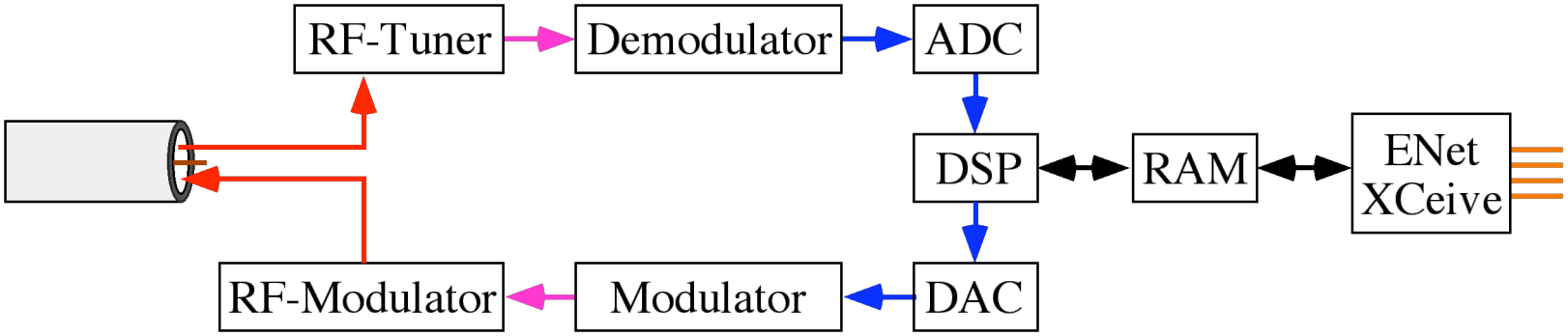
- Kabelverteilnetze (CATV)
  - Umbau zum bidirektionalen Netz



- Cable-Modems
  - downstream in einem TV-Kanal: 6 MHz
  - upstream 2 MHz Band im Bereich 10 - 32 MHz
  - alternativ Kanäle (blau, gelb) außerhalb des TV-Spektrums



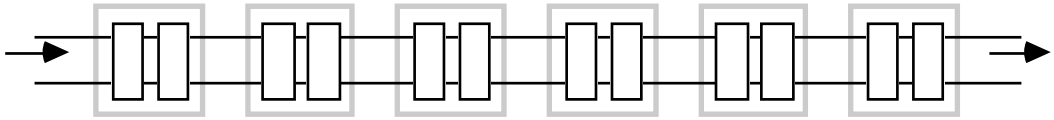
- Ähnlich konventionelle Modems
  - höhere Symbolrate
  - downstream QAM mit 64/256 Konstellation: 43 MBit/s
  - upstream QPSK 600 kbit/s - 10 MBit/s



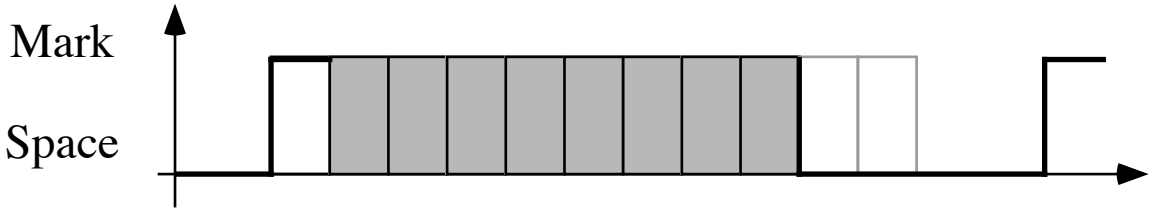
  - Schnittstelle 10BaseT, 100BaseT
- Gemeinsames Medium
  - mit anderen Teilnehmern geteilt
  - downstream durch Router verteilt
  - mehrere TV-Kanäle verwendbar
  - Sequentialisierung im upstream-Kanal: S-CDMA
  - 10% Protokolloverhead
  - Verschlüsselung

1.3.3 Digitale Signalübertragung

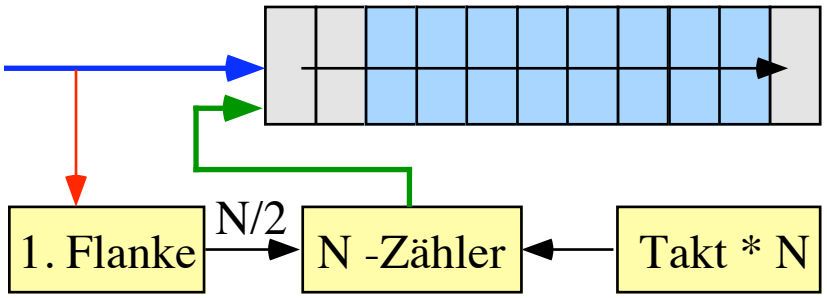
- Aufgaben
  - sichere Dekodierung der Daten (Rauschabstand)
  - Fehlerkorrektur
  - Bandbreitenbeschränkung
  - Taktableitung und Nulldurchgänge
  - Gleichstromanteil minimieren

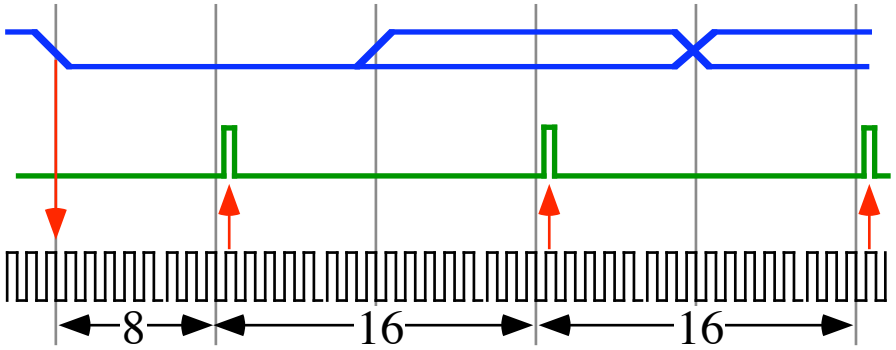


- Asynchrone Übertragung
  - Unterscheidung Leerlauf gegen Signal

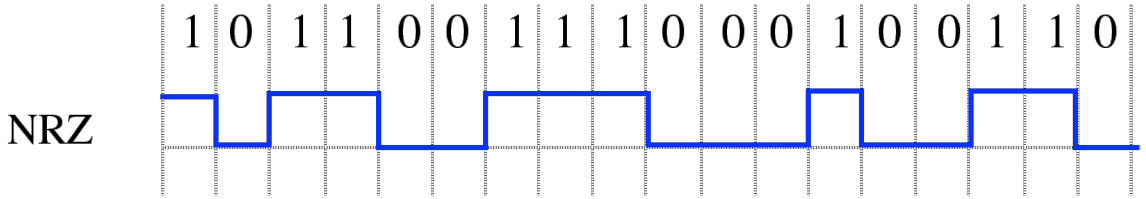


  - Takt-Resynchronisation mit jedem Startbit

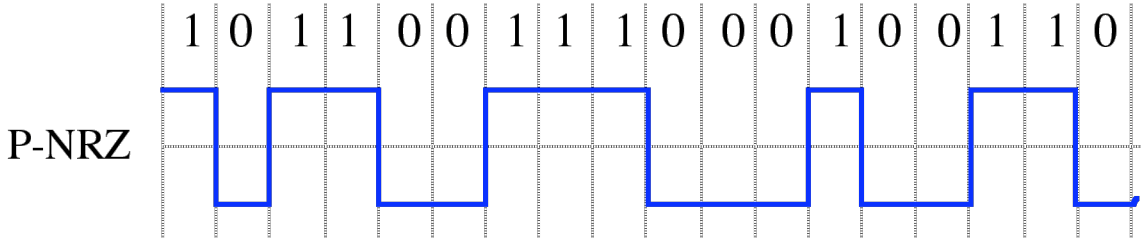




- Synchrone Übertragung
  - weniger Durchsatz-Verschwendung
  - separate Taktleitung
  - evtl. auch für Signalisierung
  - oder Füllen von Lücken mit Sonderzeichen
- NRZ: Non-Return-to-Zero

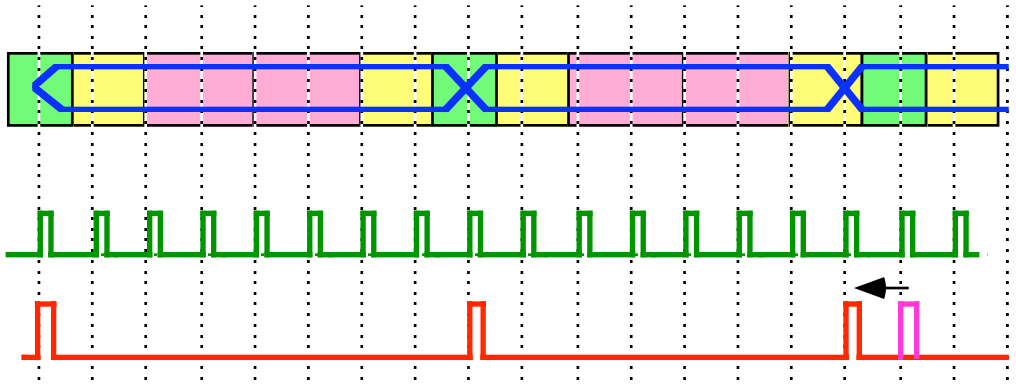


  - Gleichstromreduktion: PNRZ

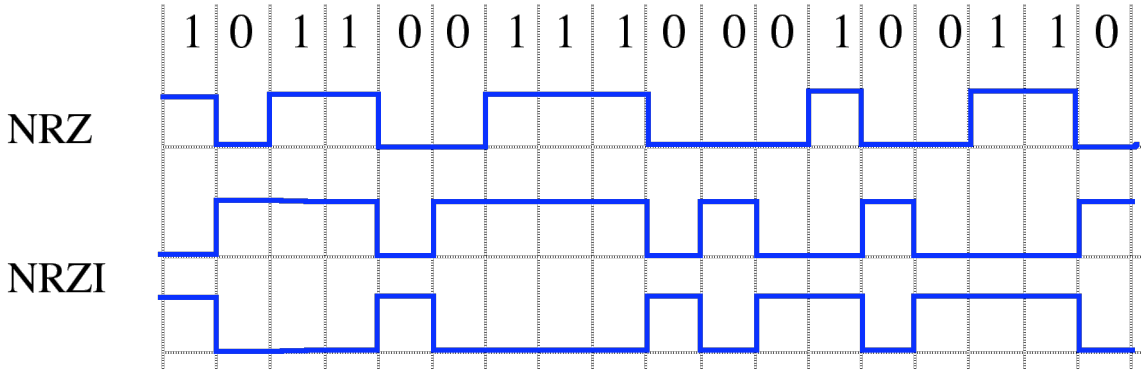


  - Gleichstrom bei ungleicher Anzahl 1 und 0

- Digital phase-locked-loop (DPLL)
  - genauer Taktgeber
  - justiert aus dem Empfangssignal

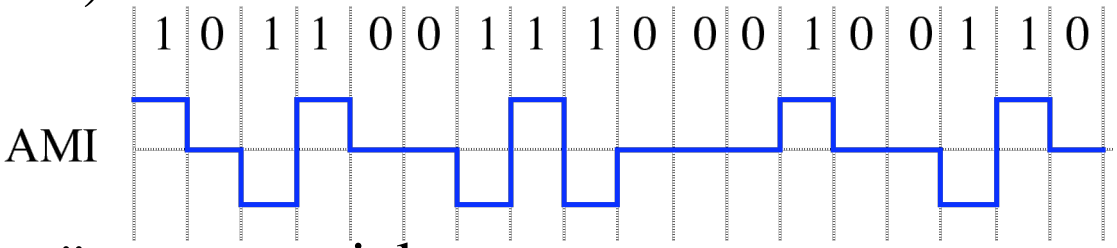


- NRZI: non return - zero inverted
  - '0' wird mit Übergang kodiert

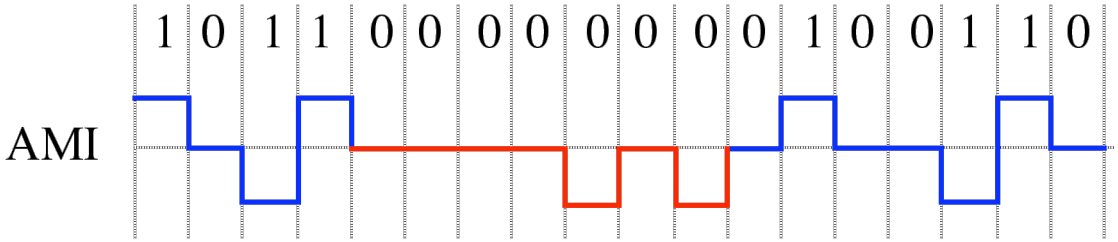


  - DPLL funktioniert nur bei genügend Übergängen
  - HDLC/LAP-B etc. haben höchsten 6 '1' nacheinander

- AMI Alternate Mark Inversion (z.B. T1)
  - return to zero

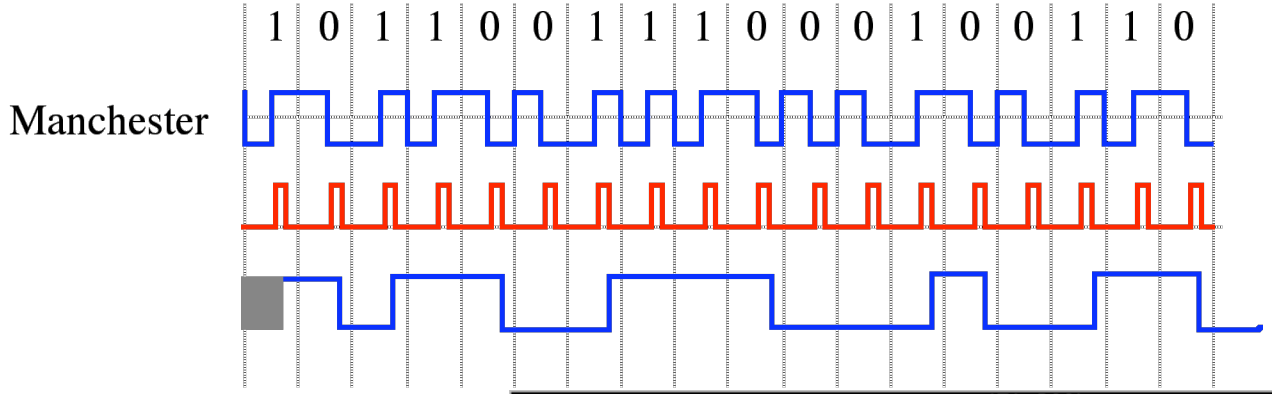


  - Takt im Signal
  - Einfügen von '1' um genügend Übergänge zu erzielen
  - Scrambler
  - Codebedingung: '1' alterniert
- Verletzung der Codebedingung
  - Fehlersituation
  - Überlagerung bei Mehrfachzugriff
  - Taktübertragung trotz '0'-Ketten (DDS, B8ZS für T1)

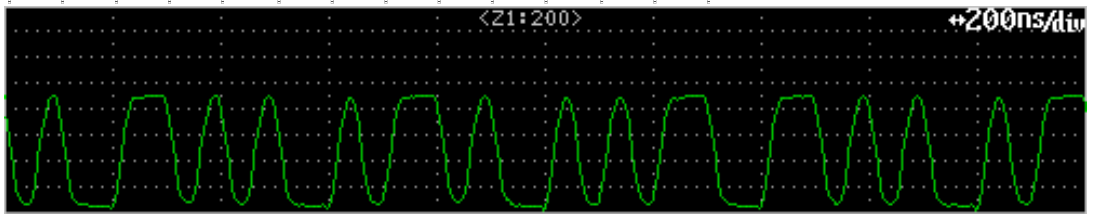


  - Taktübertragung im idle-Zustand

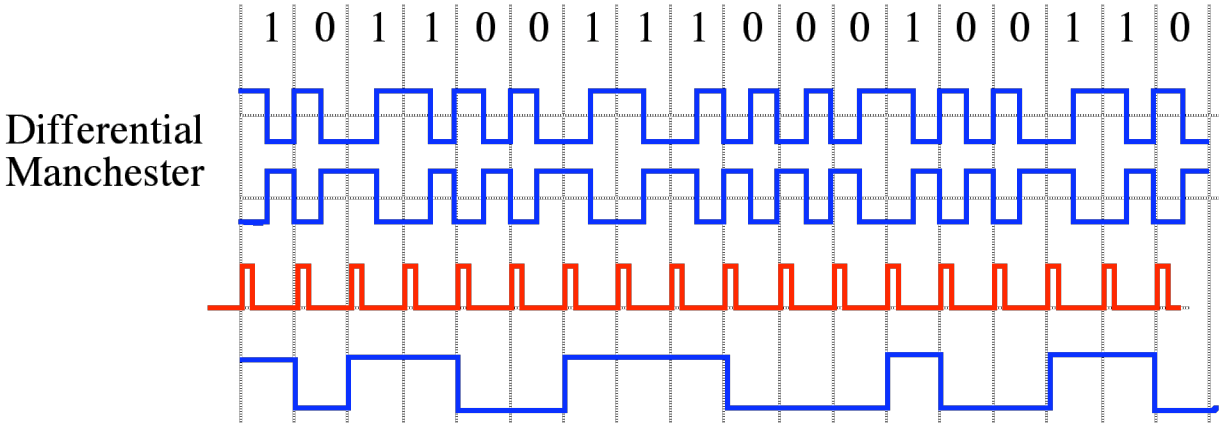
- Diphase-Codierung (Manchester) => Ethernet



  - DC-balanced



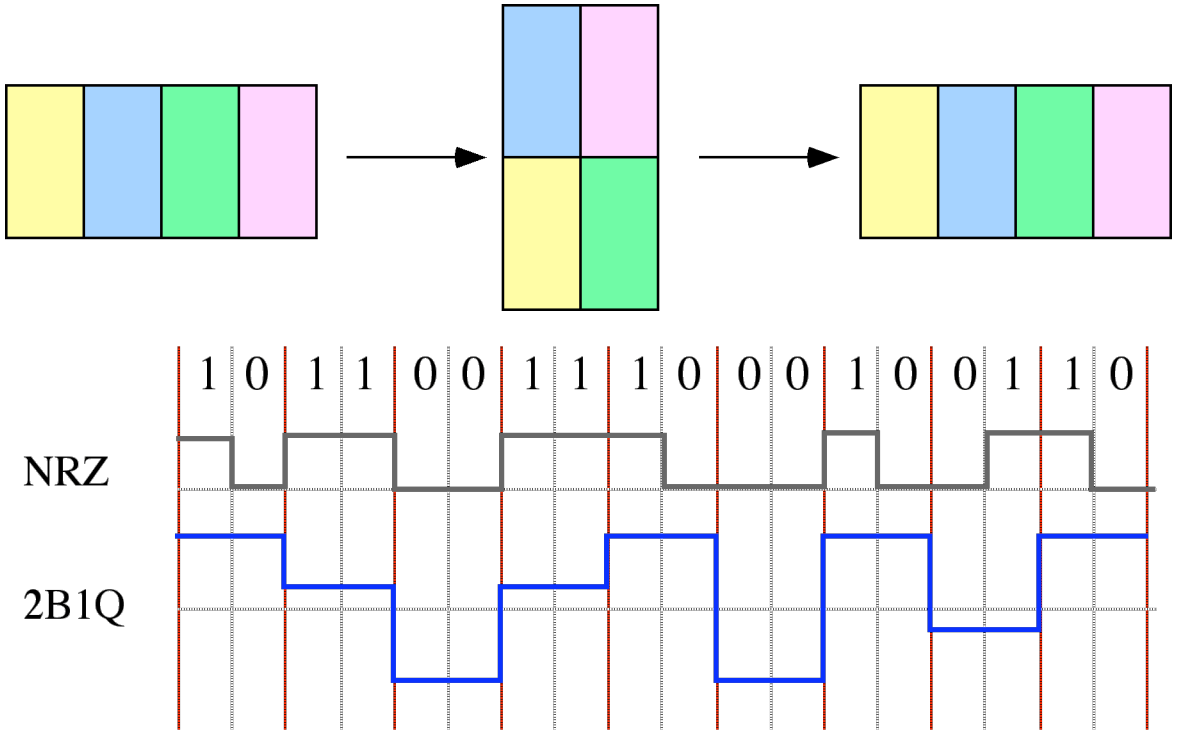
- Differential Manchester
  - Übergang am Anfang nur bei 0-Bit



  - differentieller Empfänger => Polarität irrelevant

Gruppencodierungen

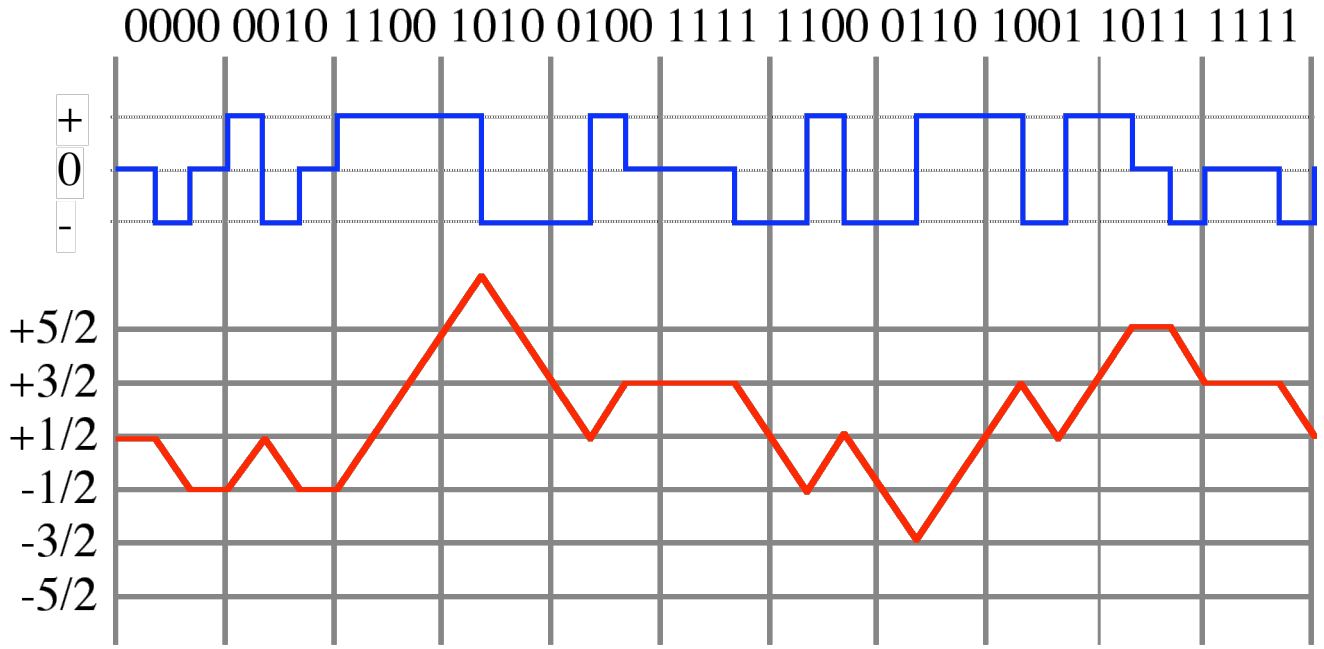
- Baudratenreduzierung mBnL
  - m B-Zeichen werden auf n L-Codesymbolen abgebildet
  - mehrere Signalstufen pro Codesymbol L
- 2B1Q Codierung
  - 2 Binärstellen codieren
  - 1 quaternäres Zeichen (quat) übertragen



- 4B3T Codierung
  - 4 Binärstellen = 16 Werte, 3 Ternärstellen = 27

	Code	nächster Zustand	Code	nächster Zustand	Code	nächster Zustand	Code	nächster Zustand
0001	0 - +	schwarz	0 - +	rot	0 - +	blau	0 - +	magenta
0111	- 0 +	schwarz	- 0 +	rot	- 0 +	blau	- 0 +	magenta
0100	- + 0	schwarz	- + 0	rot	- + 0	blau	- + 0	magenta
0010	+ - 0	schwarz	+ - 0	rot	+ - 0	blau	+ - 0	magenta
0011	+ 0 -	schwarz	+ 0 -	rot	+ 0 -	blau	+ 0 -	magenta
1110	0 + -	schwarz	0 + -	rot	0 + -	blau	0 + -	magenta
1001	+ - +	rot	+ - +	blau	+ - +	magenta	- - -	schwarz
0011	0 0 +	rot	0 0 +	blau	0 0 +	magenta	- 0 0	rot
1101	0 + 0	rot	0 + 0	blau	0 + 0	magenta	- 0 -	rot
1000	+ 0 0	rot	+ 0 0	blau	+ 0 0	magenta	0 - -	rot
0110	- + +	rot	- + +	blau	- - +	rot	- - +	blau
1010	+ + -	rot	+ + -	blau	+ - -	rot	+ - -	blau
1111	+ + 0	blau	0 0 -	schwarz	0 0 -	rot	0 0 -	blau
0000	+ 0 +	blau	0 - 0	schwarz	0 - 0	rot	0 - 0	blau
0101	0 + +	blau	- 0 0	schwarz	- 0 0	rot	- 0 0	blau
1100	+ + +	magenta	- + -	schwarz	- + -	rot	- + -	blau

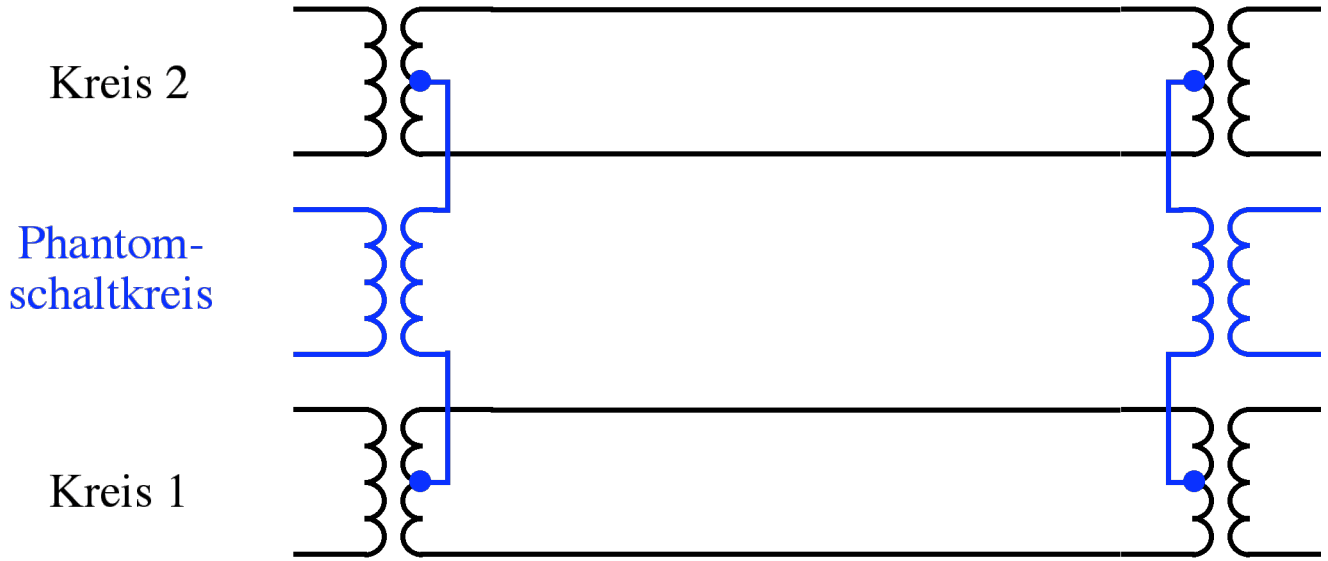
- Baudrate 3/4 der Bitrate
- Gleichstromkompensation:



- Mehrere Codierungen können dasselbe Symbol darstellen
  - stromgesteuerte Auswahl erlaubt DC-Kompensation
- ungültig Codierungen erlauben Synchronisierung auf Gruppenanfang
  - auch bei Festplatten oder Audio-CDs

1.4 Multiplex

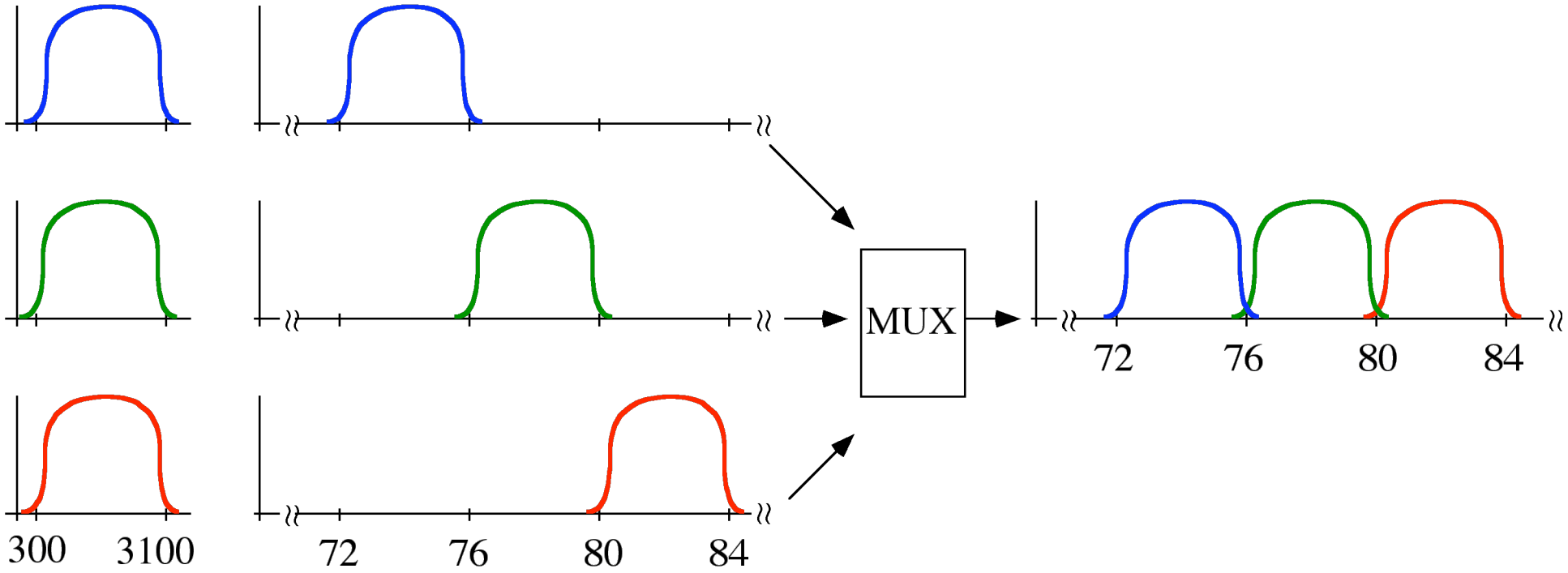
- Viele Adern pro Kabel
- Phantomschaltung
  - zusätzliches Signal aufgeteilt auf jeweils zwei Drähte
  - wenig Übersprechen



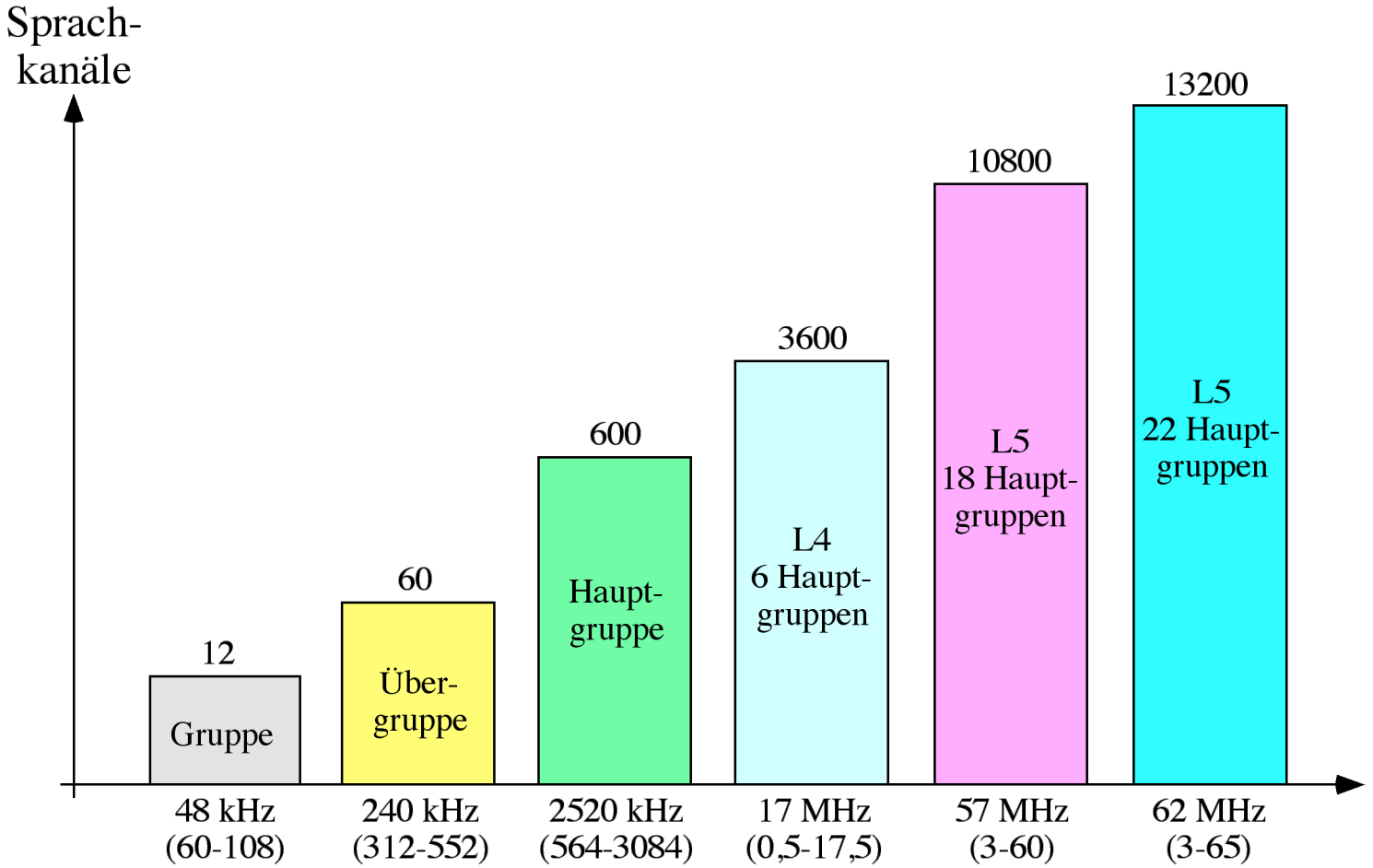
  - auch 4+3

1.4.1 Frequenzmultiplex

- Frequency Division Multiplex (FDM)
- Amplitudenmodulation
  - Sinusschwingung mit Signal modulieren
  - unteres Seitenband ausfiltern
  - Frequenzbandumsetzung
- Wiederholung der Frequenzbandumsetzung

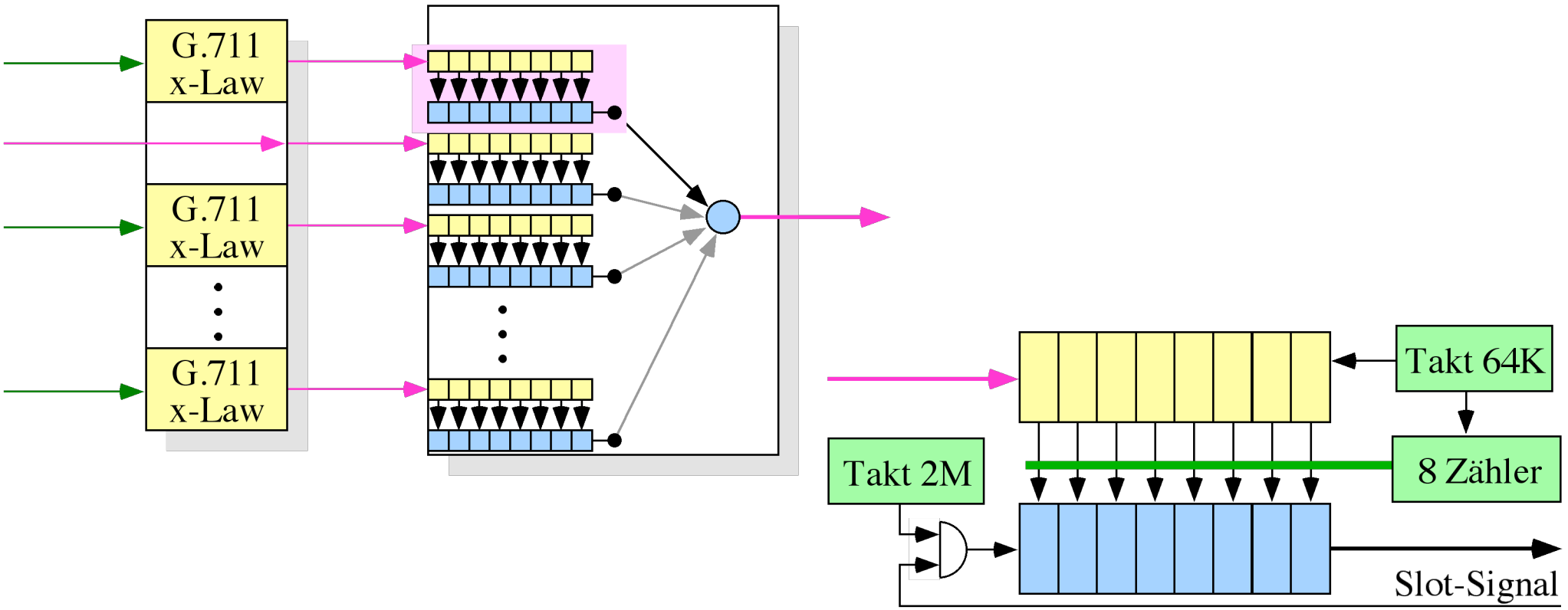


- Analoge Übertragungshierarchie
  - LE5 auf Koaxialkabel, Verstärker nach 1,5 km: 13200 Telefonate
  - TAT-7: Verstärker nach 10 km, 4200 Telefonate
  - Gegenkopplungsverstärker



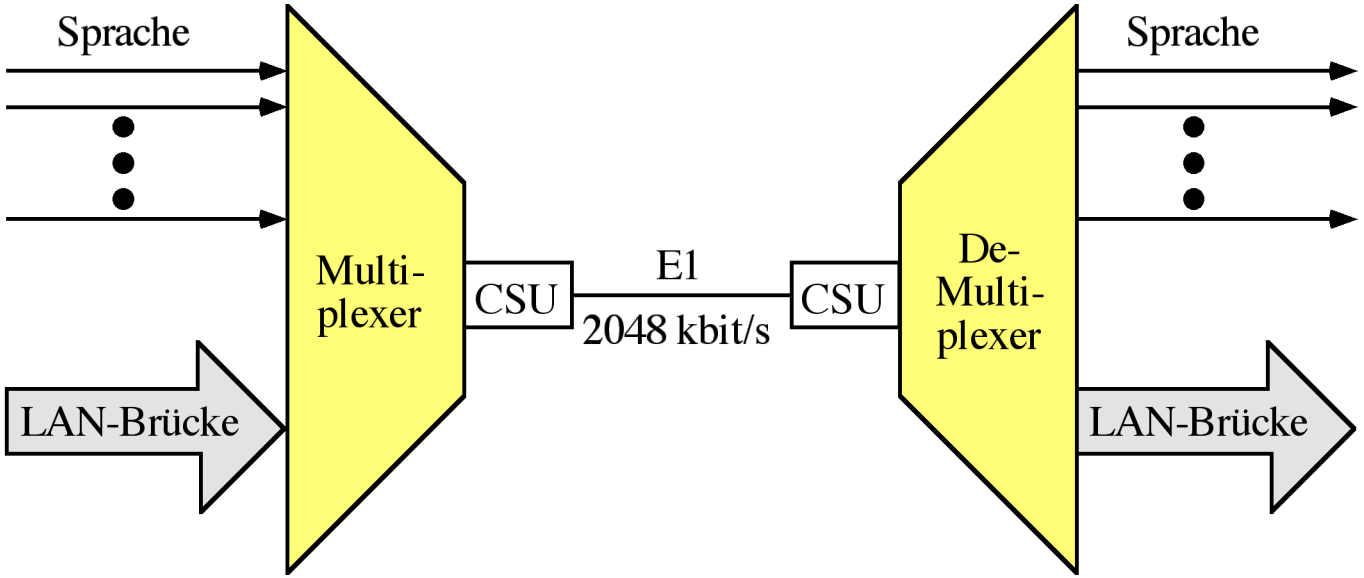
1.4.2 Zeitmultiplex

- Time Division Multiplexing (TDM)
- Einteilung der Zeit in Bereiche (Zeitschlitze)
  - Leitung mit hohem Bittakt
  - Eingangssignale mit niedrigem Bittakt
  - Zuweisung von Zeitschlitzen an Eingangsleitungen
  - Einfügen und Entnehmen des langsamen Signales mit hohem Takt

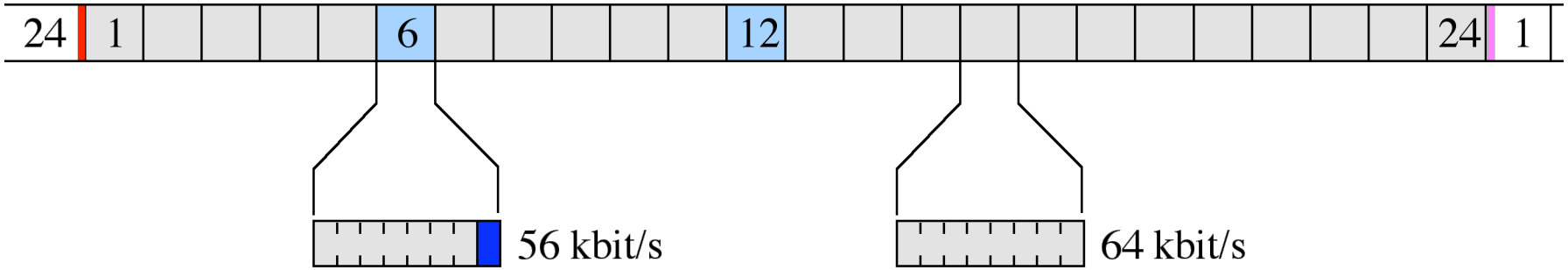


T1/E1

- Übertragung von PCM-Samples zwischen Vermittlungen
  - DS0: 64 kbit/s
  - T1 (DS1): 24 Telefonkanäle* 64 kbit/s = 1.544 MBit/s
  - E1: 2.048 MBit/s: 30 Kanäle + Signalisierungskanal + Sync-Kanal
  - Pulsamplitudenmodulation AMI
  - verdrillte Kupferadern, digitale Repeater nach 900 m
  - ab 1962
  - vermietet an Kunden, heute auch für Daten
  - CSU: Channel Service Unit



- Synchrone Übertragung mit Rahmenstruktur

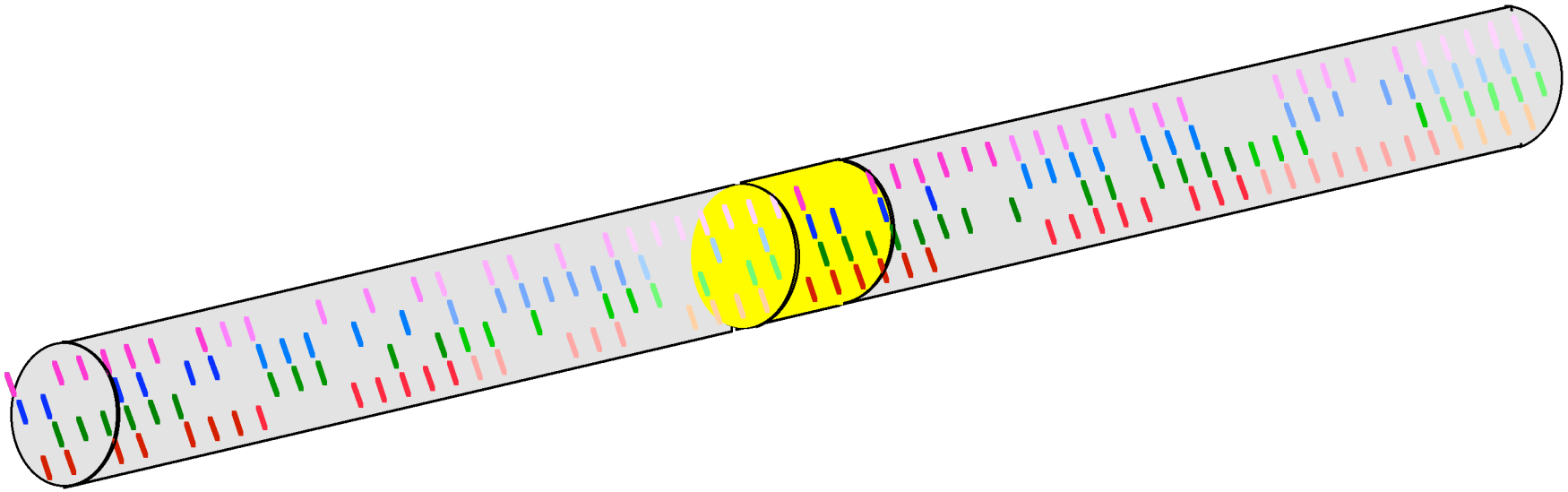


  - 22 * 8 bit + 2 * 7 bit + 2 bit Signalisierung + 1 bit Frame = 193 bit
  - 8.000 * 193 = 1.544.000
  - B7ZS (64R), B8ZS siehe oben
- DS-Hierarchie, Plesiochronous Digital Hierarchy (PDH)

Mbit/s	Telefonkanäle	US&CA	ITU-T	MBit/s
0,064	1	DS0		0,064
1,544	24	DS1	E1	2,048
6,312	96	DS2	E2	8,448
44,736	672	DS3	E3	34,368
139,264	1920	DS4E	E4	139,264
274,176	4032	DS4	E5	565,148

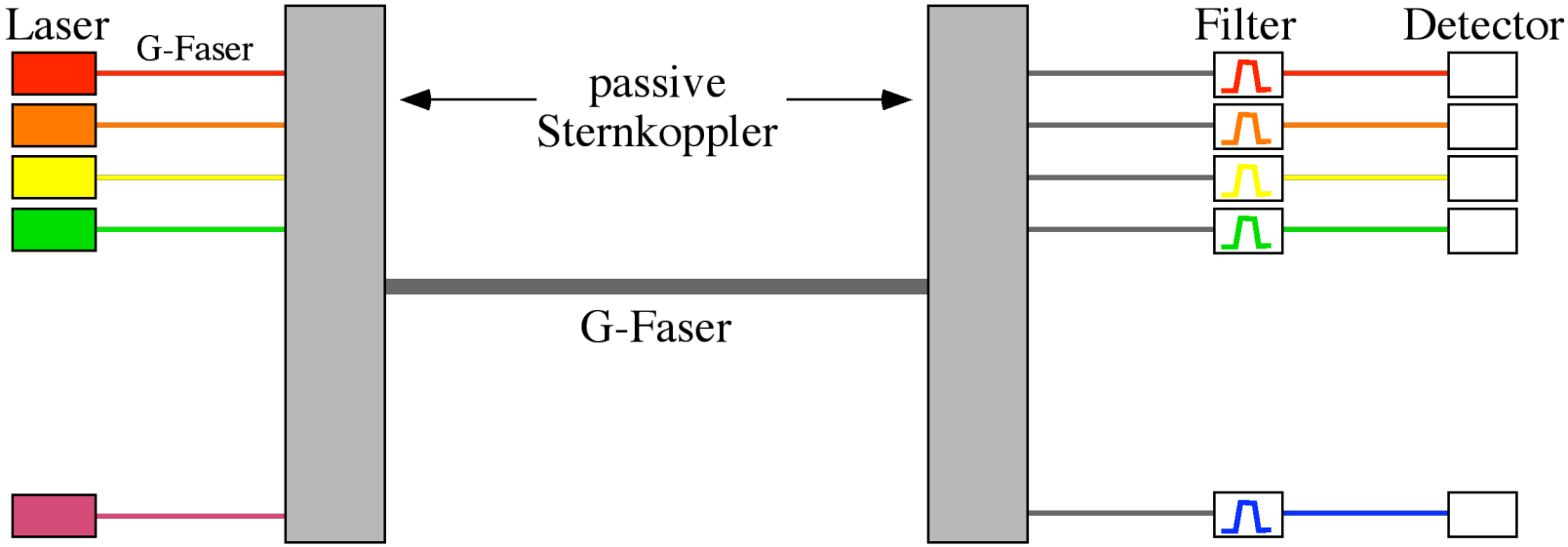
1.4.3 Wavelength Division Multiplex (WDM)

- Typische Glasfaser: 1,55μ, 0,2 dB/km, 250 MBit/s
- Injection Laser Diodes
  - sehr schmales Spektrum
  - verschiedene Wellenlängen
- Mischen (multiplexen) mehrerer Lichtwellen auf einer Glasfaser
  - theoretisch 25 THz zur Verfügung
  - 1022 Kanäle

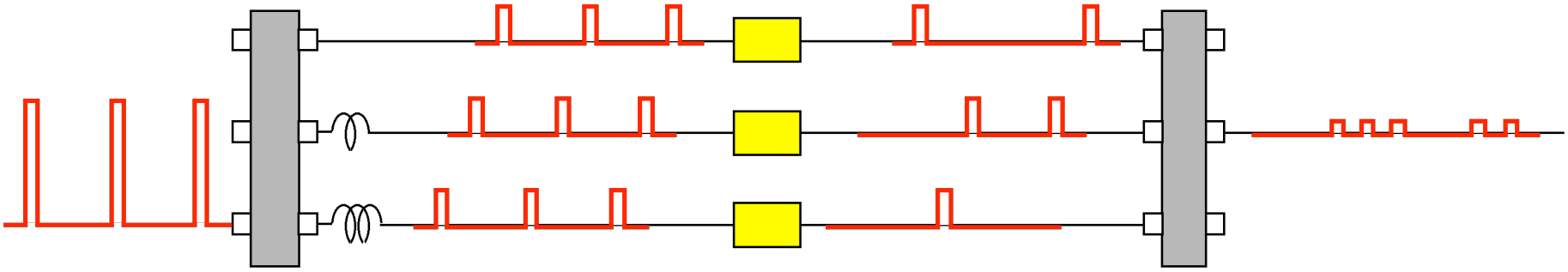


- elektronische Regeneratoren nur für ein Signal, langsam
- optische Verstärker

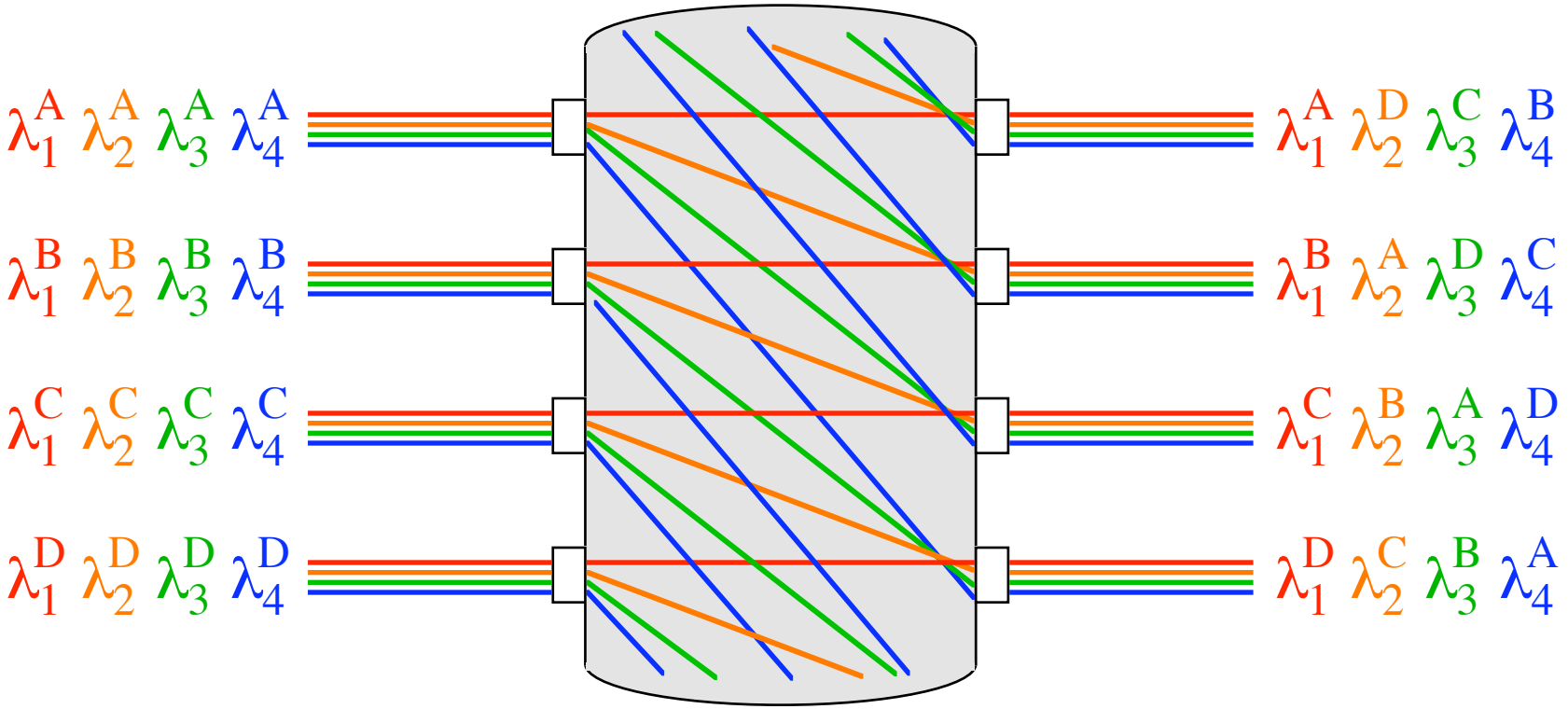
- WDM-Schema
  - Kanal-Zwischenraum 50 - 500 GHz (0,4 - 4 nm)



- Delay-Line Multiplexing
  - Datenraten >100 GBit/s erwünscht, Laser mit hohem Pulstakt schwer
  - 400 GBit/s pro Kanal

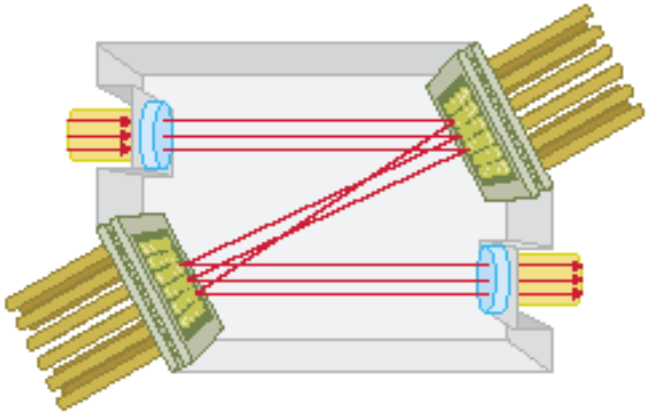


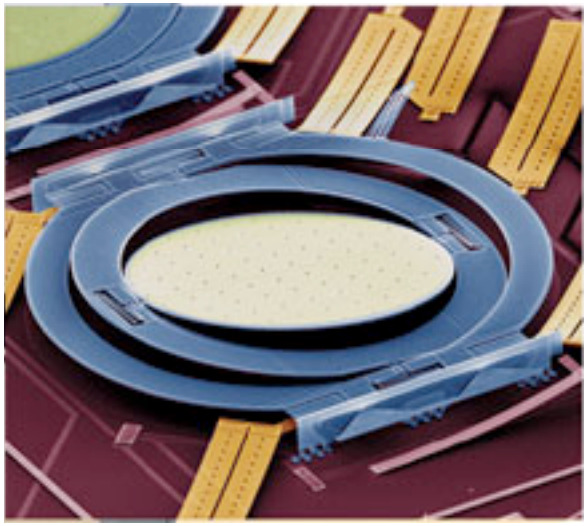
- Passiver Wellenlängen Router

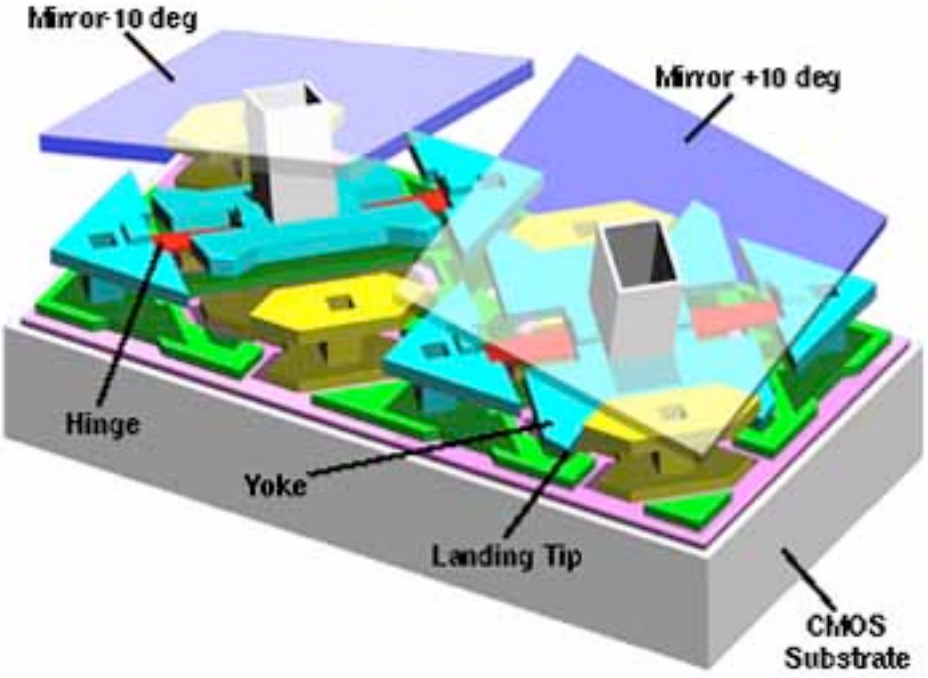


- Experimentelle Resultate
  - 1,1 TB - 150 km, 'normale' Faser, 55 Kanäle á 20 GB [1996]
  - 2,6 TB, 126 Kanäle á 20 GB [1997]
  - 0,1 TB - 9000 km
- Kommerziell z.B. 160 Kanäle, 10-40 GBit/s pro Kanal
- Zu lösen: Vermittlungstechnik, Signalisierung

- Prismen zerlegen in Einzel-Wellenlängen
- Optical Cross Connect
  - viele kleine Spiegel auf einem Chip
  - MEMS: microelectromechanical systems
  - jeder Spiegel individuell ansteuerbar
  - pro Datenstrahl ein Spiegel







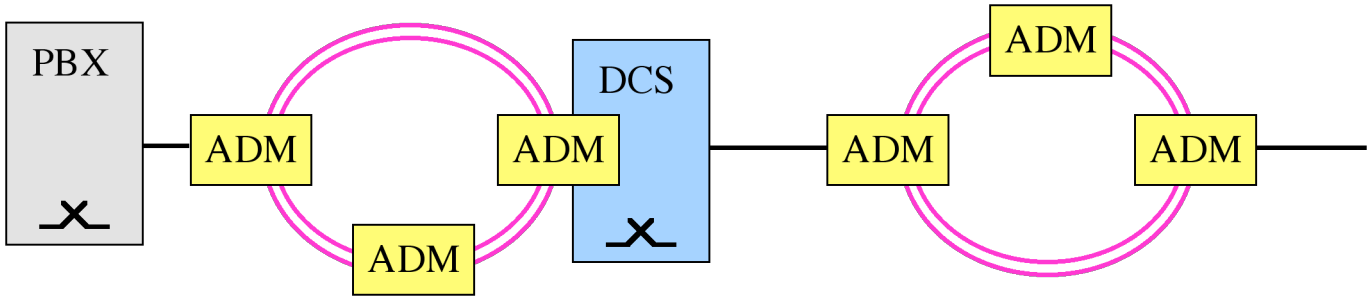
- evtl. Frequenzumsetzung

http://www.bell-labs.com/org/physicalsciences/projects/mems/mems.html

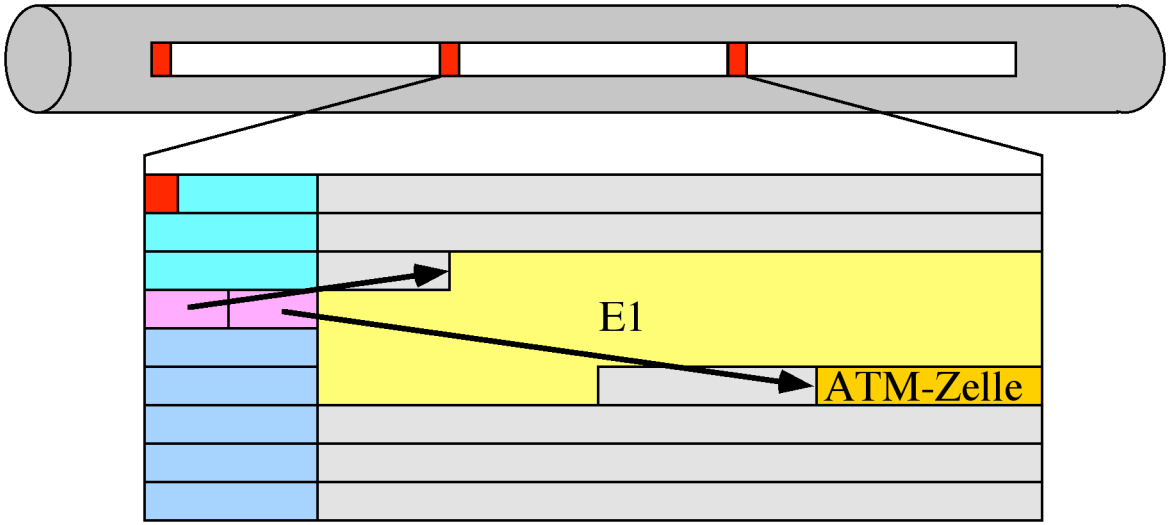
1.5 Digitale Kanäle

1.5.1 SONET/ SDH

- Synchronous Optical Network [Bellcore]
- Europa: SDH synchronous digital hierarchy
- Vorher: T1, …, DS3 = 28 * T1
  - einzeln synchron
  - 3 gemischte Rahmensignale
  - vollständiges Demultiplexen um Unterkanäle zu extrahieren
- Einfaches (De-)Multiplexen
  - add and drop
  - einzelne Kanäle bis auf 64 kbit/s
- Ringstruktur mit Add/Drop Multiplexern (ADM)
  - doppel- oder vierfach-Ring
  - Ringe werden mit Digital Cross-Connects verbunden (DCS)



- Sonet-Rahmen = Synchronous Transport Signal - Level 1
- Synchronous Transport Module - level 1: STM-1 (STS-3)
  - 155.52 MBit/s
- 8000 STM-1 Rahmen /Sekunde
  - 9 Zeilen á 270 Bytes (9 Byte Overhead, 261 Byte Nutzlast)
  - 2016 DS0 Kanäle (64 kbit/s)



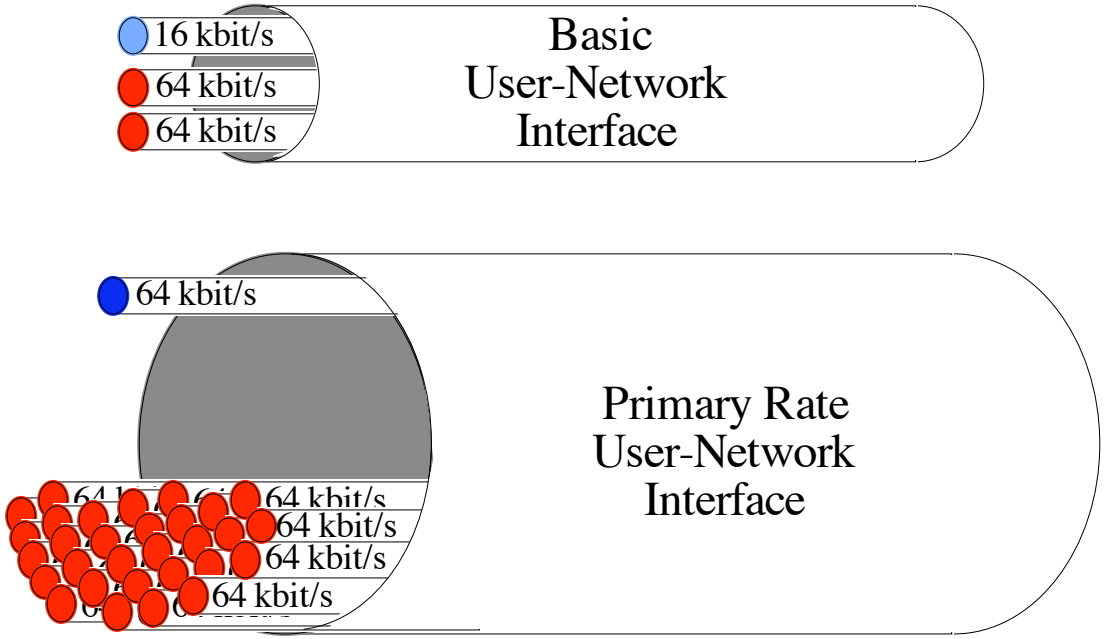
- Management (overhead)
  - Framing, Parity, Signalisierung, Bitfehlertest, Schleifen, Statistik
  - Zeiger auf Virtual Container (VC)
  - Sprachkanäle für Personal

OC-Kanal-Hierarchie

- Standards und Schnittstellen von Glasfasern
  - zwischen Vermittlungsstellen
  - Verbindungen im synchronen optischen Netzwerk
- Optical Carrier: 256 Interfaces

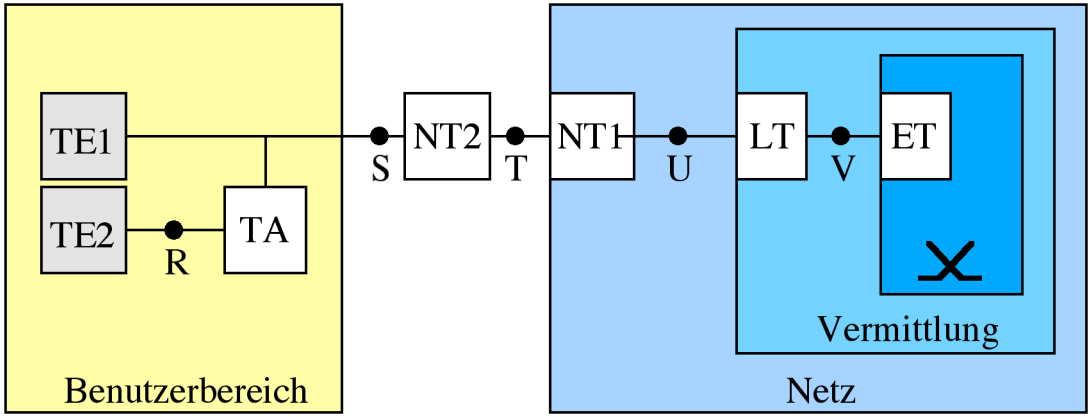
Datenrate (Mbit/s)	OC-Ebene	SONET	SDH
51,84	OC-1	STS-1	-
155,52	OC-3	STS-3	STM-1
466,56	OC-9	STS-9	STM-3
622,08	OC-12	STS-12	STM-4
933,12	OC-18	STS-18	STM-6
1244,16	OC-24	STS-24	STM-8
1866,24	OC-36	STS-36	STM-12
2488,32	OC-48	STS-48	STM-16

1.5.2 ISDN-Leitungen

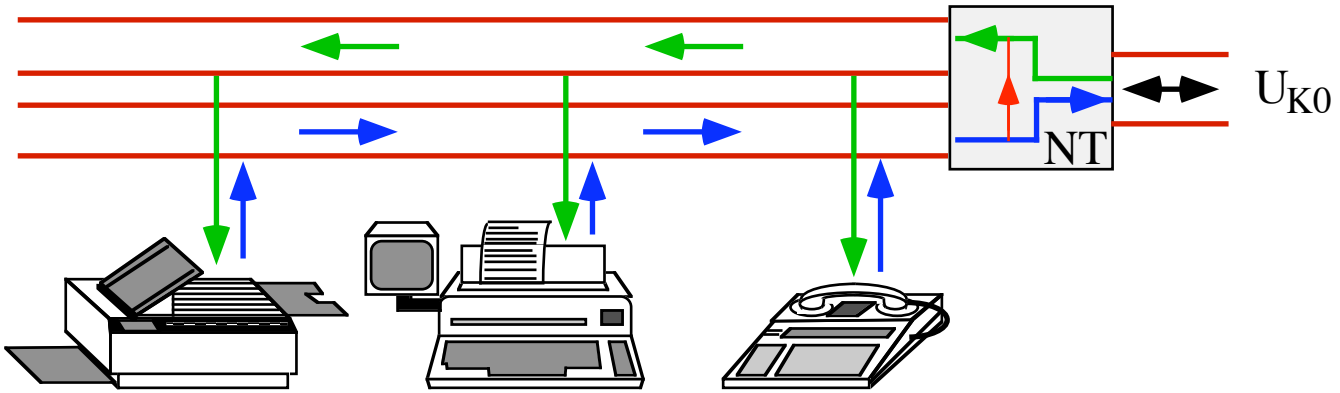


- 64-Kbps, unrestricted, 8-kHz structured
  - 8000 Bytes pro Sekunde
  - transparent, unrestricted
- 64-Kbps, 8-kHz structured, usable for speech information transfer
  - PCM-Samples
  - wird A-Law nach μ-Law oder entgegengesetzt konvertiert
  - Sprache kann auch weiter komprimiert werden

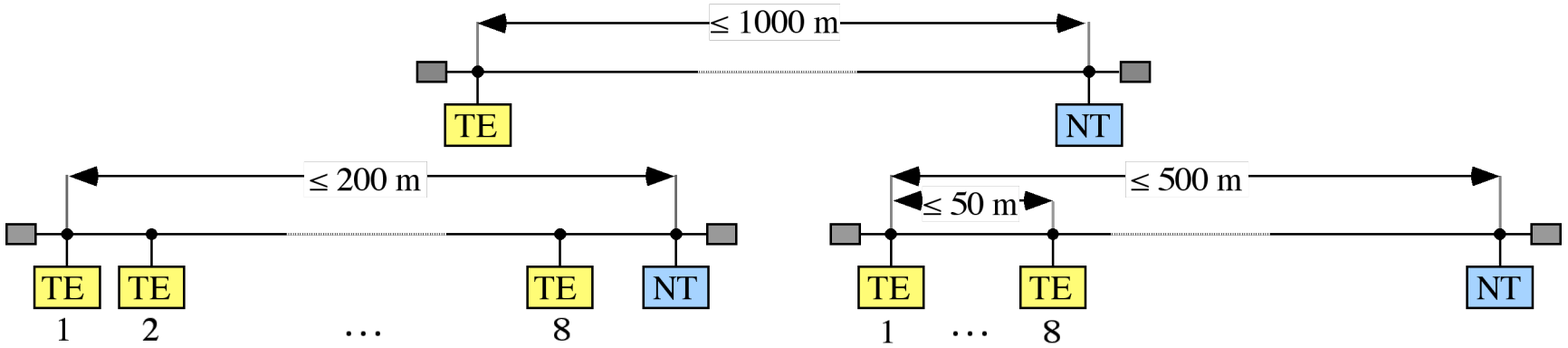
- 64-Kbps, 8-kHz structured, for 3.1 kHz audio information transfer
  - PCM-samples
  - A-Law nach μ-Law
  - keine weitere Kompression, für Modemsignale geeignet
- Alternate speech/64-Kbps, unrestricted, 8-kHz structured
  - Umschaltung zwischen dem ersten und dem zweiten Modus
- 2x64-Kbps, 384-Kbps, 1,536-Kbps, 1920-Kbps unrestricted
  - 8-kHz structured
  - Simultanverbindung
  - phasengleicher Schaltung von 2, 6, 24 bzw. 30 B-Kanälen
  - H-Kanäle
- ISDN Schnittstellen und Leitungen



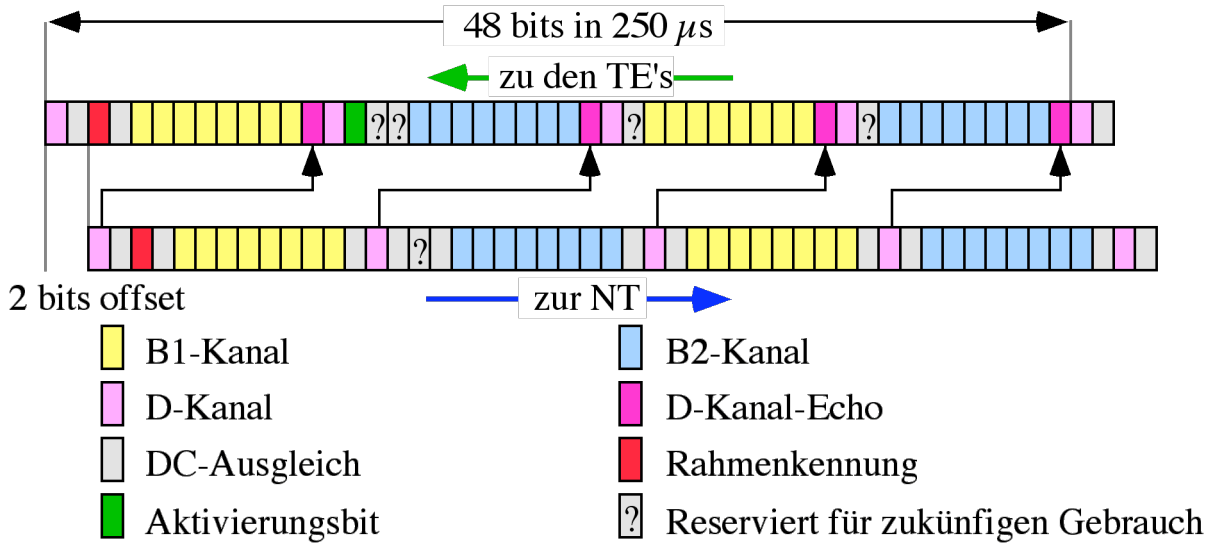
- Teilnehmerschnittstelle: S_0
  - Netzabschluss (NT) wandelt 2 auf 4-adrigen hausinternen Bus
  - eingeschränkte Stromversorgung
  - Bis zu 8 Endgeräte am So-Bus



- Für kurze Anschlussdistanzen
  - 500 Meter für Busbetrieb
  - 1000 Meter für Einzelanschluss



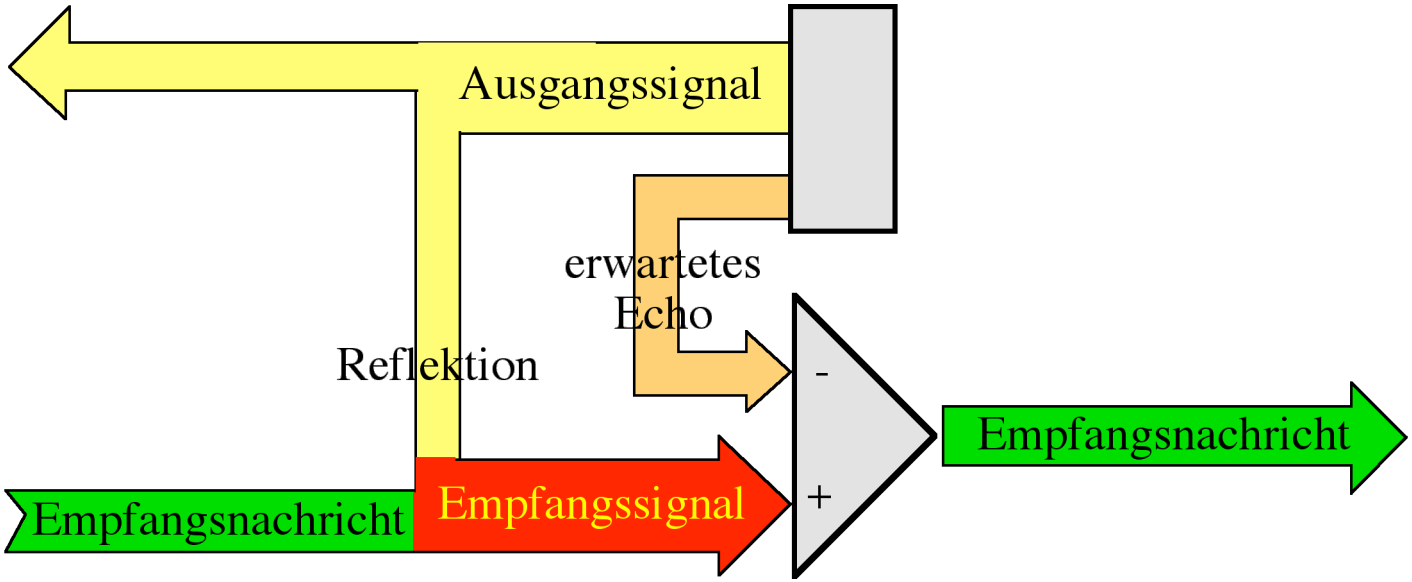
- 2 B-Kanäle à 64 KB:
  - für gleichzeitigen Betrieb mehrer Geräte
  - Mehrtelefonbetrieb im Haus
- D-Kanal (-Protokoll) für alle gleichzeitig:
  - Signalisierung beim Verbindungsaufbau
  - Behelfsmässiger Datenverkehr



- S_0-Rahmen für 2B+D auf dem 4-adrigen Bus
  - Doppelader "inbound"
  - Doppelader "outbound"

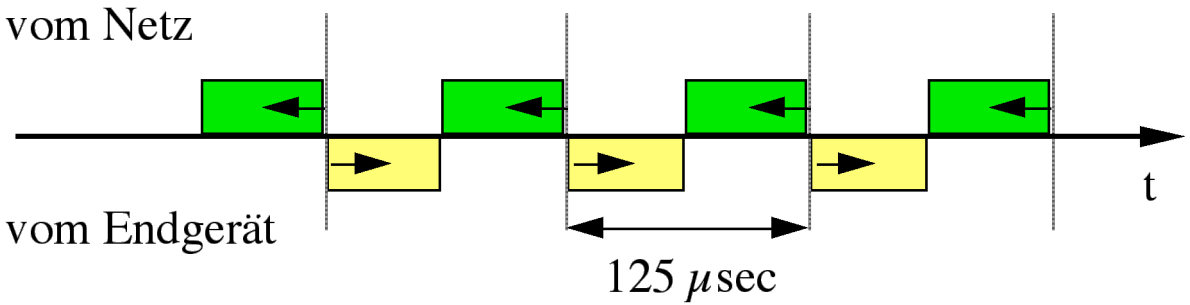
- Netto 144 kbit/s, brutto 192 kbit/s pro Richtung
  - 2* 64 KBits/sec (B-Kanäle)
  - 1* 16 KBits/sec (D-Kanal)
  - outbound 16 KBits/sec D-Kanalecho
  - Rahmenbildung ...
- Kollisionsfreie Zugriffsregelung für den D-Kanal mit Echobits
  - ähnlich CSMA/CD, aber zerstörungsfrei
  - Sender hört im Echo-D-Kanal mit
  - senden falls Kanal frei
  - evtl. Kollision
  - Kontrolle im Echo-Kanal: Sendestop falls Empfang ≠ Sendesignal
  - '0'-Bits stärker als '1'
  - Adressfeld des Rahmens unterschiedlich
- S_{2m} wie T1/E1

- Teilnehmeranschlußleitung: U_{K0}
  - MMS43 (Modify Monitor Sum, 4B3T-Code)
  - Rahmen 1 ms: 11 Sync-Symbole, 1 Meldesymbol, 4*27 Nutzbauds
  - 36 Bits: 2 Byte B1-Kanal, 2 Byte B2-Kanal, vier Bit D-Kanal
  - 2 Drähte mit Echokompensation
  - 8 km

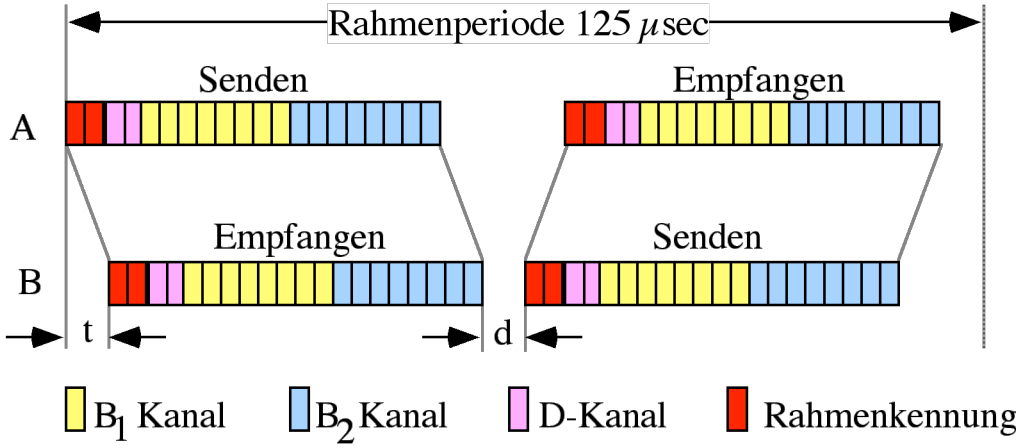


  - EuroISDN auch 2B1Q

- Teilnehmeranschlußleitung: U_{P0}
  - 2B1Q
  - 2 Drähte mit Ping-Pong



  - Distanz < 2 km
  - höhere Übertragungsrate: 384 kbit/s
- evtl. proprietäte Verfahren
  - 1 B-Kanal und 160 kbit/s
  - einfacheres Rahmenformat
  - 512 kbit/s, …

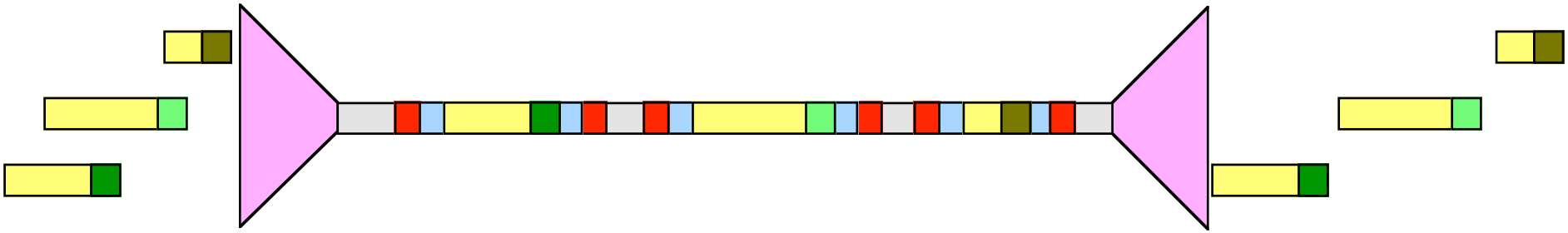


1.6 Asynchrone Rahmenübertragung

- Synchrone Leitung - asynchroner Datenstrom
  - Rahmenkennzeichen



  - Füllsignale
    - Synchronisation, …
  - beliebig Bitanzahl
  - statistischer Multiplex

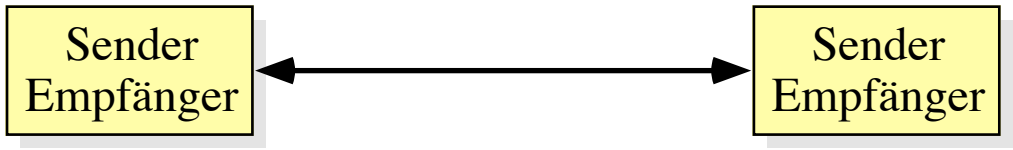


  - Verbindungskontrolle
  - Fehlererkennung
    - Wahrscheinlichkeit unentdeckter Fehler, CRC
  - Fehlerkorrektur: Wiederholung, FEC

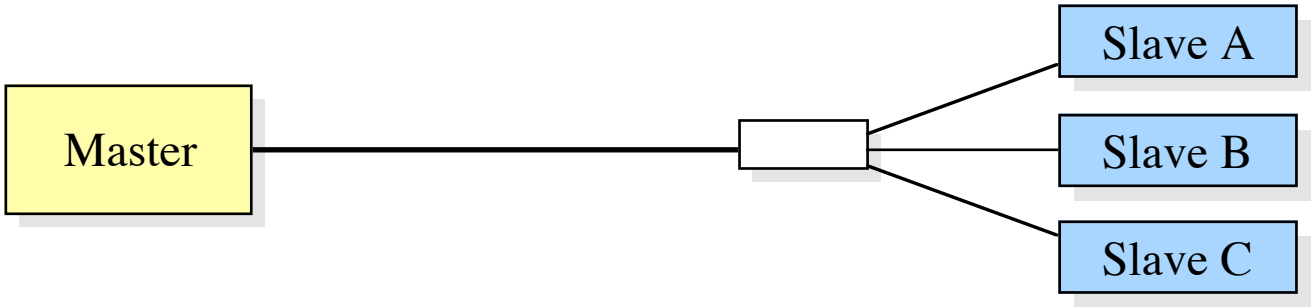
Andere Namen: Zellen, Pakete

Topologien

- Punkt-zu-Punkt

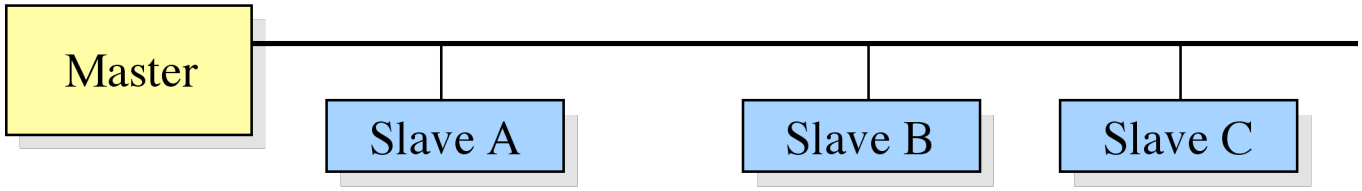


- Multipoint



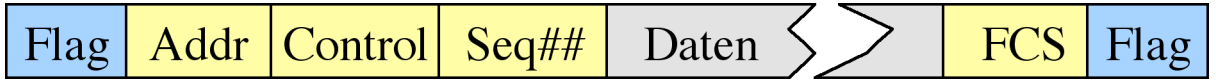
  - Teilstrecke auf einem Übertragungskanal mit Multiplex
  - Adressierung

- Multidrop



  - ein Übertragungskanal, mehrere Zuhörer
  - Adressierung

- Exemplarisches Rahmenformat
  - "Link-Layer Protocol Data Unit"
  - bit- oder byteorientiert



- Flag: Oktettsynchronisierung und Rahmenbegrenzung
- Addr: Adressierung für LAN- und Mehrpunktkonfigurationen
- Control Art der Meldung
  - Open, Close
  - Bestätigung, Flusskontrolle
  - Informationstransport ...
- Seq##: Laufende Nummerierung
  - evtl. verlorene Meldungen festzustellen
  - Bestätigungen ebenfalls mit Sequenznummer.
- Daten: Von der Sicherungsschicht nicht weiter interpretiert
- CRC: Prüfsumme für Bitfehler (Cyclic Redundancy Check)

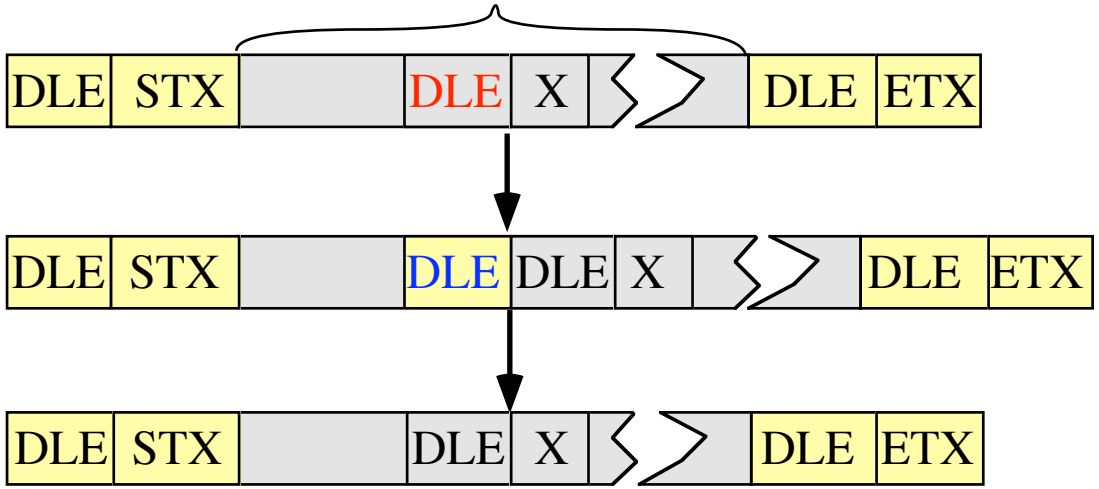
1.6.1 Zeichenorientierte Rahmenübertragung: BSC (Bisync)

- IBM
  - Terminalansteuerung
  - EBCDIC
  - ISO basic mode mit ASCII
- Spezialbuchstaben für Rahmenbildung und Kontrolle
  - SYN als Lückenfüller
  - SOH: start of header
  - STX: start of text
  - ETX: end of text
  - EOT: end of transmission
  - ACK/NAK: (negative) acknowledge
  - DLE: data link escape
- Zeichenorientierter Rahmen
  - Daten durch (DLE, STX) und (DLE, ETX) eingerahmt
  - Paketanfang (DLE, SOH, Identifier, Stationsadresse)

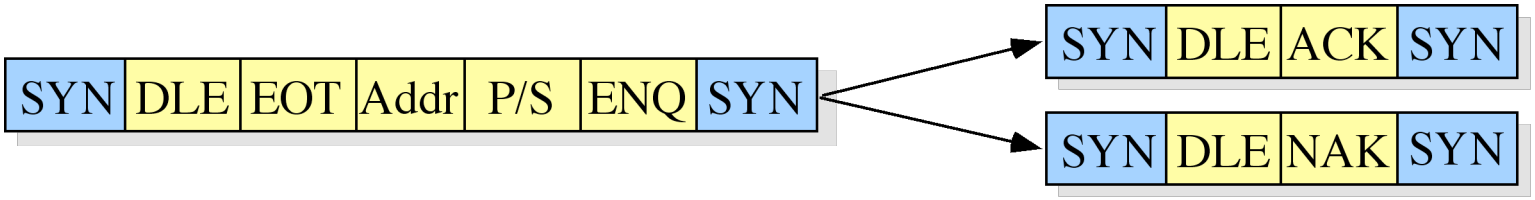


- Transparenz der Datenübertragung:

- Daten eine (DLE, . . .) Gruppe als Inhalt => Missverständnis
- "Zeichenstopfen": ein zusätzliches DLE-Zeichen wird eingefügt
- vom Empfänger wieder entfernt

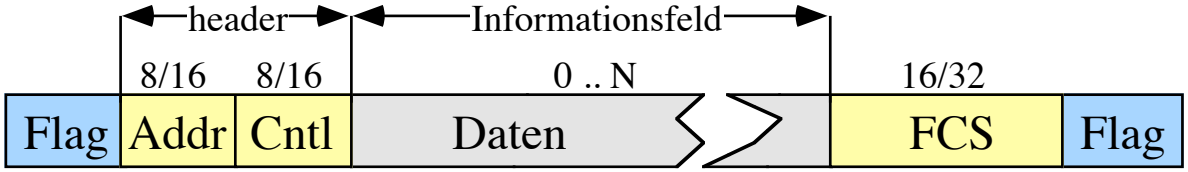


- Poll+Adresse: Master bietet Slave die Möglichkeit, Daten zu senden
- Select + Adresse: Master hat Daten für den Slave



1.6.2 Bitorientierte Rahmenübertragung: HDLC

- High Level Data Link Control [IBM]
  - LAP-B (ISO 4335)
  - LAP-D (ITU Q.921) Link-Protokoll für ISDN-Signalisierung
  - LLC (Logical Link Control, IEEE 802.2) für LANs



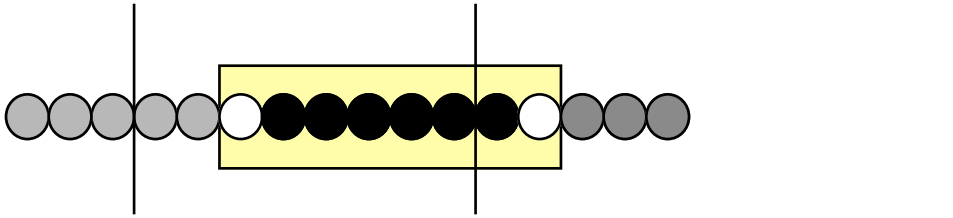
- HDLC-Rahmen
  - beliebige Bitanzahl im Rahmen (Paket)
  - Flags 'rahmen' Daten ein
  - Kontrollinformation am Paketanfang (header)
  - CRC-Prüfsumme am Paketende ($x^{16}+x^{12}+x^5+1$)

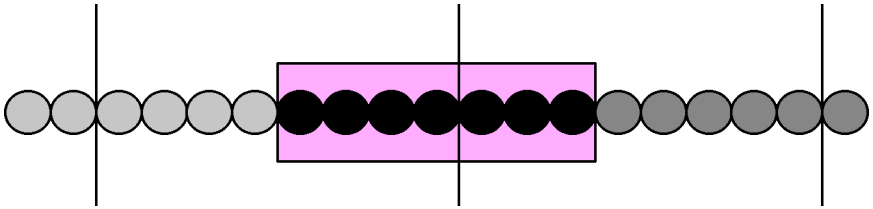
- Besondere Bitmuster

Flags, ' 0111 1110 '

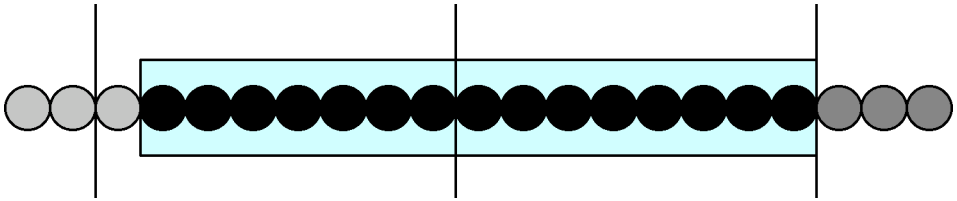
- evtl. auch als Idle-Muster
- Bitstuffing

Abbruchmuster (abort), 7 mal '1'



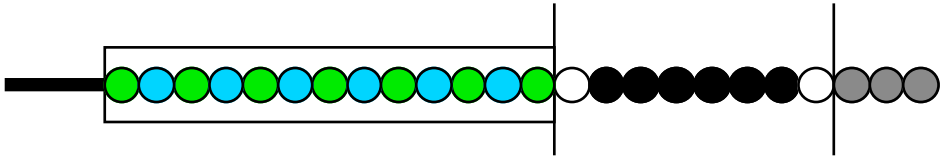


Idle-Muster, 15 mal '1'



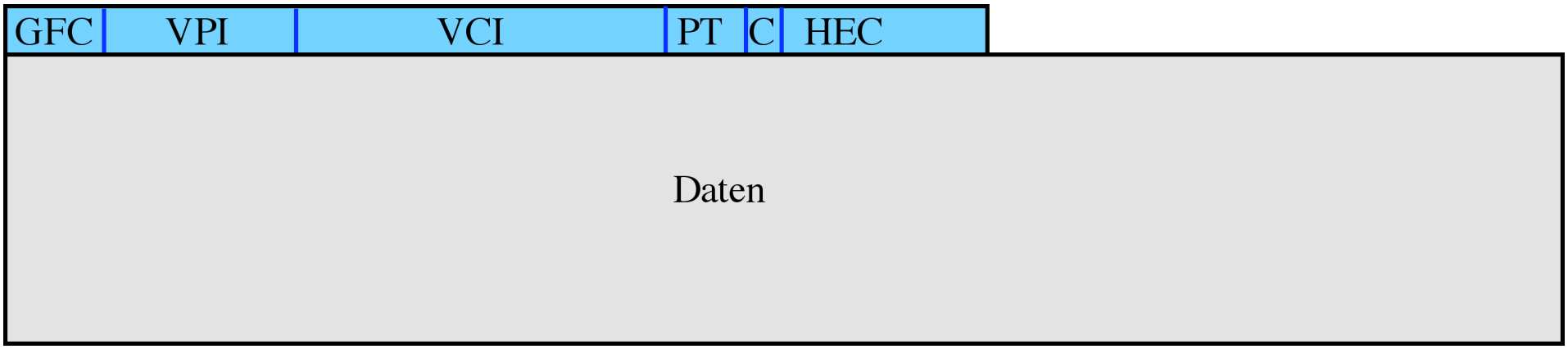
Taktsynchronisation

- zusätzliche Präambel
- falls kein Takt vom Modem
- evtl. >100 Bits,
- gehört zur physikalischen Ebene



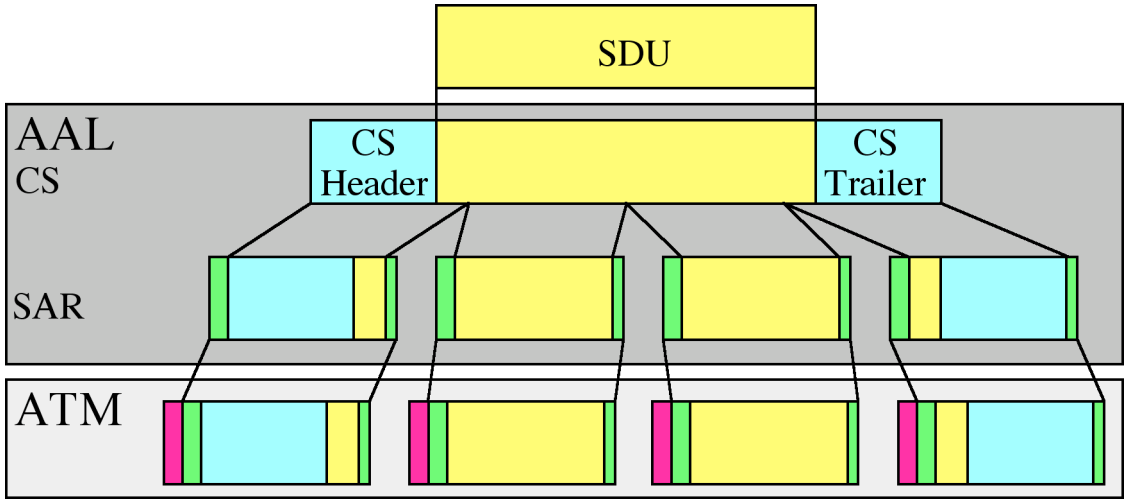
1.6.3 Übertragung mit fester Rahmengröße: ATM-Zellen

- 53 Bytes = 48 Byte Nutzlast + 5 Byte Verwaltung



- Problem: Aufteilung der Nutzdaten auf Zellen
  - Anwendungsdienst (=> application layer framing)
  - zwischengeschaltete Übertragungsdienste (AAL)
  - Ströme: Ton, Video
  - SDUs: Datenpakete
- AAL: ATM Adaptation Layer
  - Verteilung größerer SDUs auf den Nutzlastbereich der ATM-Zelle
  - evtl. Fehlerkorrektur, Framing, etc.

- Convergence Sublayer: Rahmenbildung

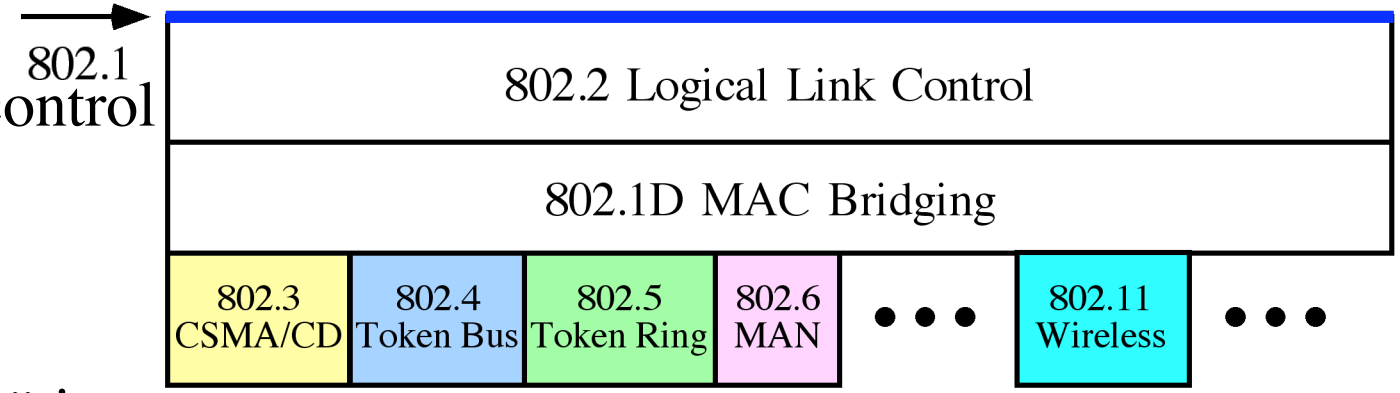


- Segmentation and Reassembly Sublayer
  - bei 140 MBit/s aufwendig

	AAL 1	AAL 2	AAL 3/4	AAL 5
Verkehrstyp	CBR	VBR	ABR/UBR	ABR/UBR
Medien	H.261, G.711	MPEG	Daten	Daten
Nutzlast	46 oder 47	45	44	48
Header in Zelle	Seq#	Seq#		-
Fehler	CRC, FEC?	CRC, FEC?	CRC, Segmentinfo	
CS-Header			Puffergröße	Mngt, CRC

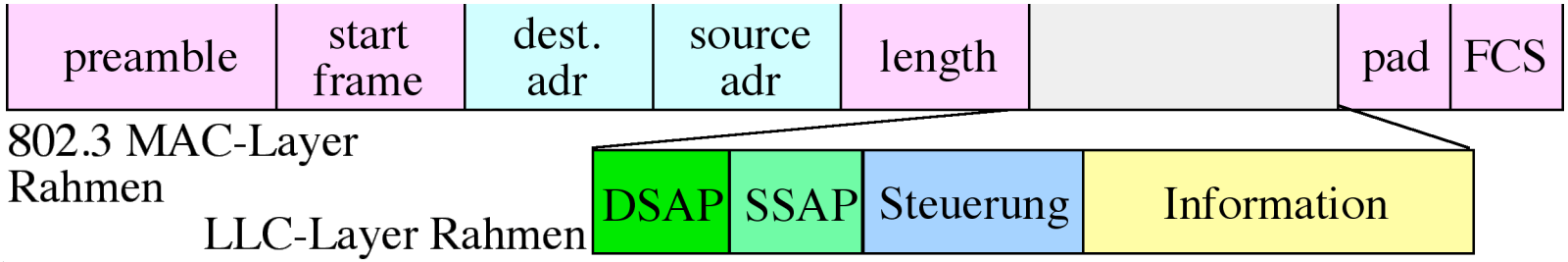
1.7 Lokale Netzwerke

- Broadcast-Topologie
  - einer sendet, alle hören zu
  - Adressauswertung im *Empfänger* entscheidet über Speicherung
- IEEE Standards 802.x
  - MAC: Medium Access control



- 802.1 Interface
  - "verbindungslos", unbestätigt
    - L_DATA.request, L_DATA.indication (auch DL_UNITDATA)
  - "verbindungslos", bestätigt
    - L_DATA_ACK.[*], L_DATA_ACK_STATUS.indication
  - "verbindungsorientiert"
    - L_CONNECT, L_DATA_CONNECT, L_DISCONNECT

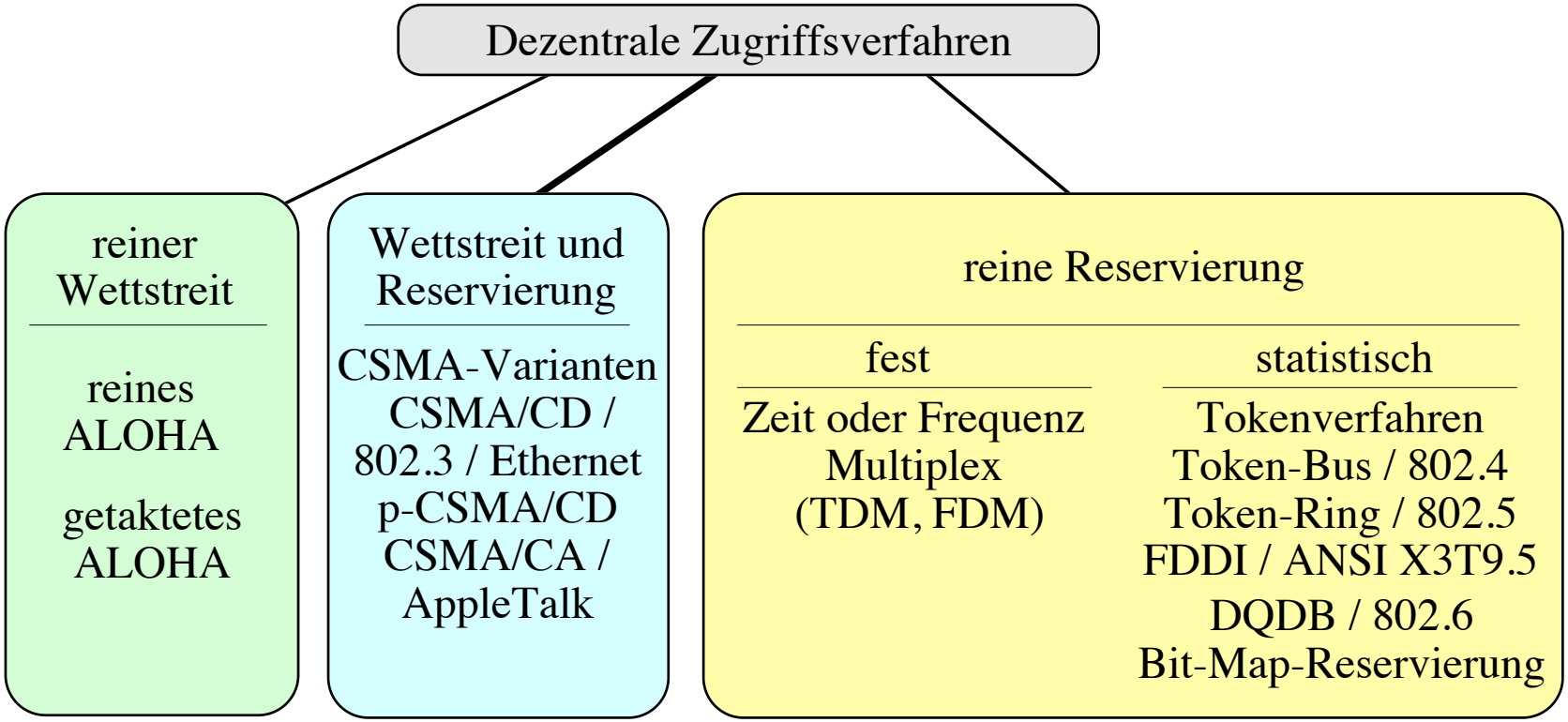
- LLC-Rahmen



- Steuerfeld ähnlich HDLC, LAPB, LAPD …
  - Quell- und Zieladresse
  - 1 oder 2 Byte Steuerfeld
- Verbindungsorientierte Protokoll-Varianten mit 7 Bit Sequenznummern
- Rahmentypen ähnlich HDLC
  - nur symmetrische Konfigurationen
  - I, RR, RNR, REJ, UI, UA, DISC
  - SABMF, XID, TEST, DM, FRMR
  - UI-Frame für typischen Datagramm-Transfer
- MAC-Services
  - MA_UNITDATA.[requestlindication]
  - MA_UNITDATA.confirm bestätigt Übertragung
  - in machen LANs auch Empfang

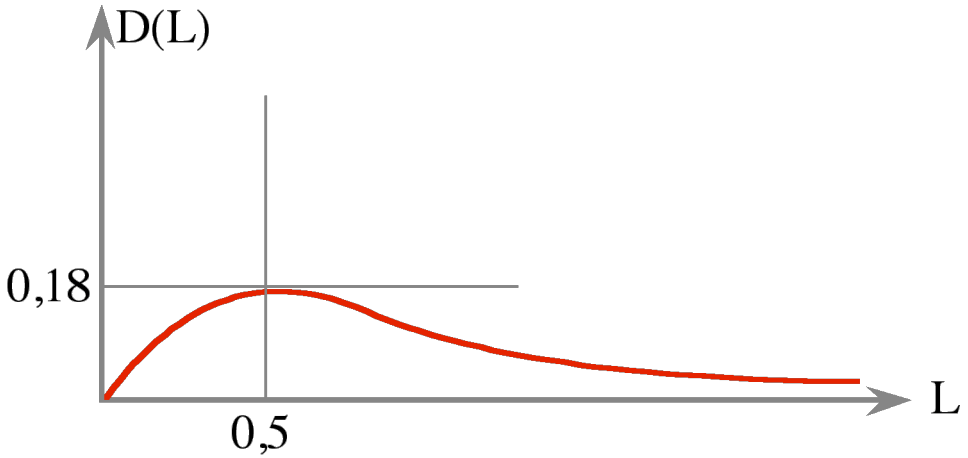
1.7.1 Zugriffssequentialisierung

- Reservierungsverfahren
  - Anmeldung im Reservierungsrahmen
  - Stationen senden in aufsteigender Folge
  - Verhältnis Reservierung/Datenübertragung ungünstig bei vielen Teilnehmern und niedriger Aktivität
- konzeptuell dezentrale Verfahren

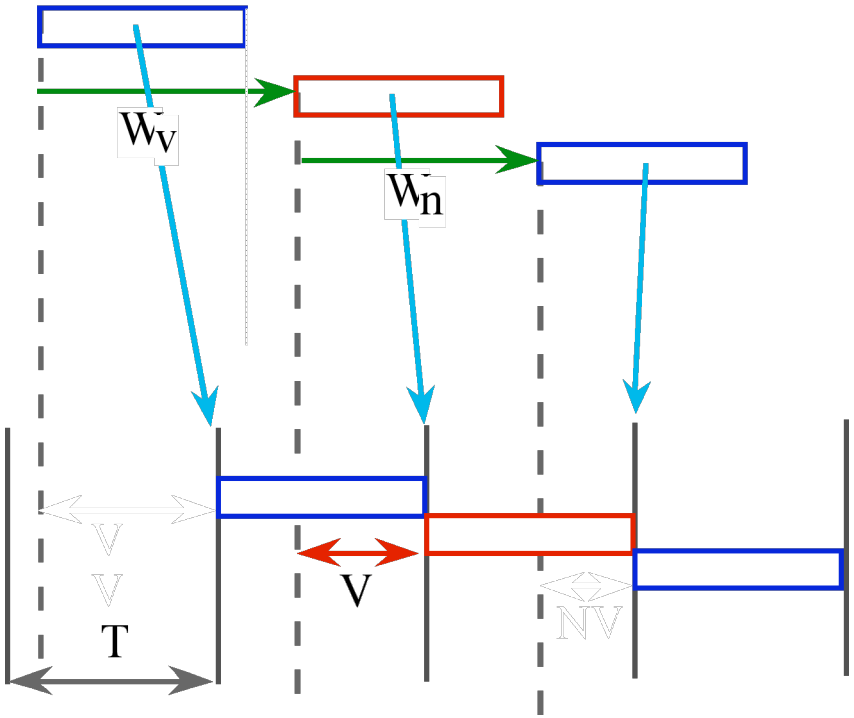


1.7.1.1 Statistische Verfahren

- Aloha
  - Paket sofort senden, wenn es bereit ist
  - Überlagerung/Kollision wird nicht bemerkt
  - höhere Schichten warten auf Bestätigung
  - falls Timeout erneute Übertragung
- Unter Annahme der Poissonverteilung D< 18%

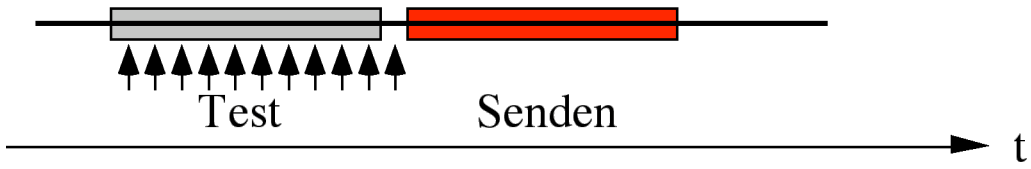


- slotted Aloha
  - Zeitschlitze (Slots) der Dauer ~T
  - Paketlänge fest
  - Übertragung nur zu Beginn eines Slots

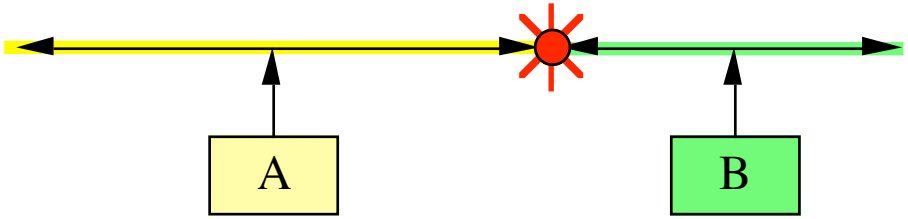


  - entweder komplette Überlagerung oder keine
- Annahme Exponentialverteilung: D < 37%

- CSMA: Carrier Sense Multiple Access
- Vor Übertragung hören, ob Medium belegt
  - wenn frei: Senden
  - wenn belegt: warten und nochmal versuchen



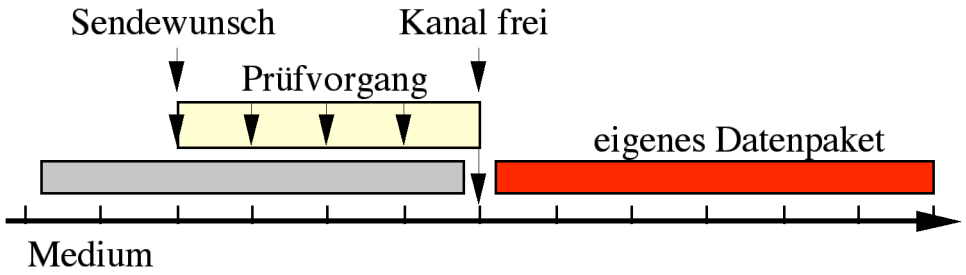
- Problem falls zwei (fast) gleichzeitig testen und dann senden
  - Signallaufzeit bis zu allen anderen ist Risikoperiode



=> p(Kollision) für CSMA ist proportional zur Signallaufzeit
- langes Kabel erhöht Kollisionsgefahr

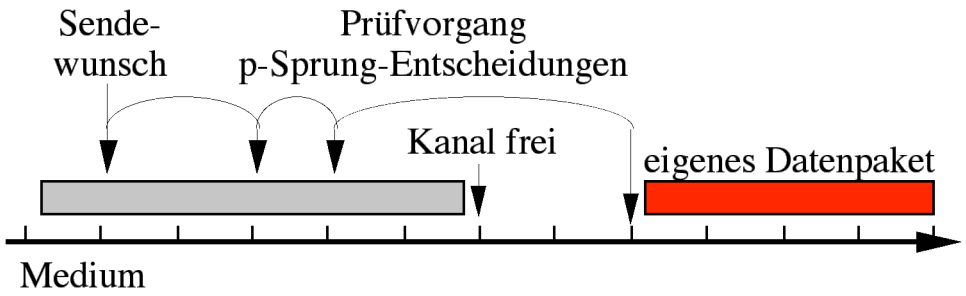
- Persistent CSMA
  - sofort senden, wenn der Kanal frei ist

```
while carrier
      do  { persist };
Send;
```



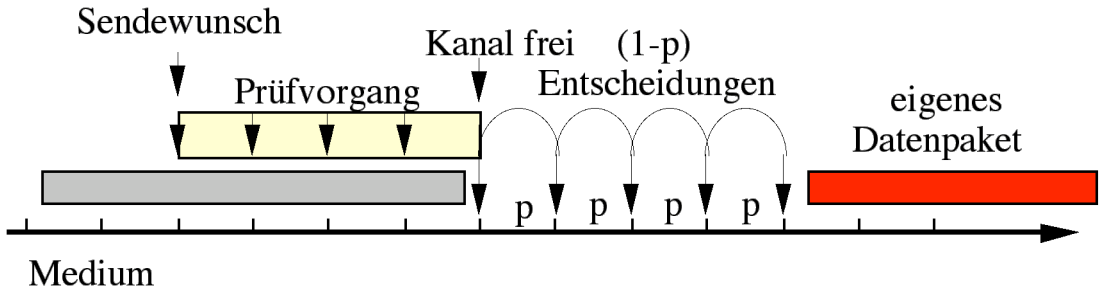
- Non-persistent CSMA:

```
while carrier
      do Wait(random);
Send;
```

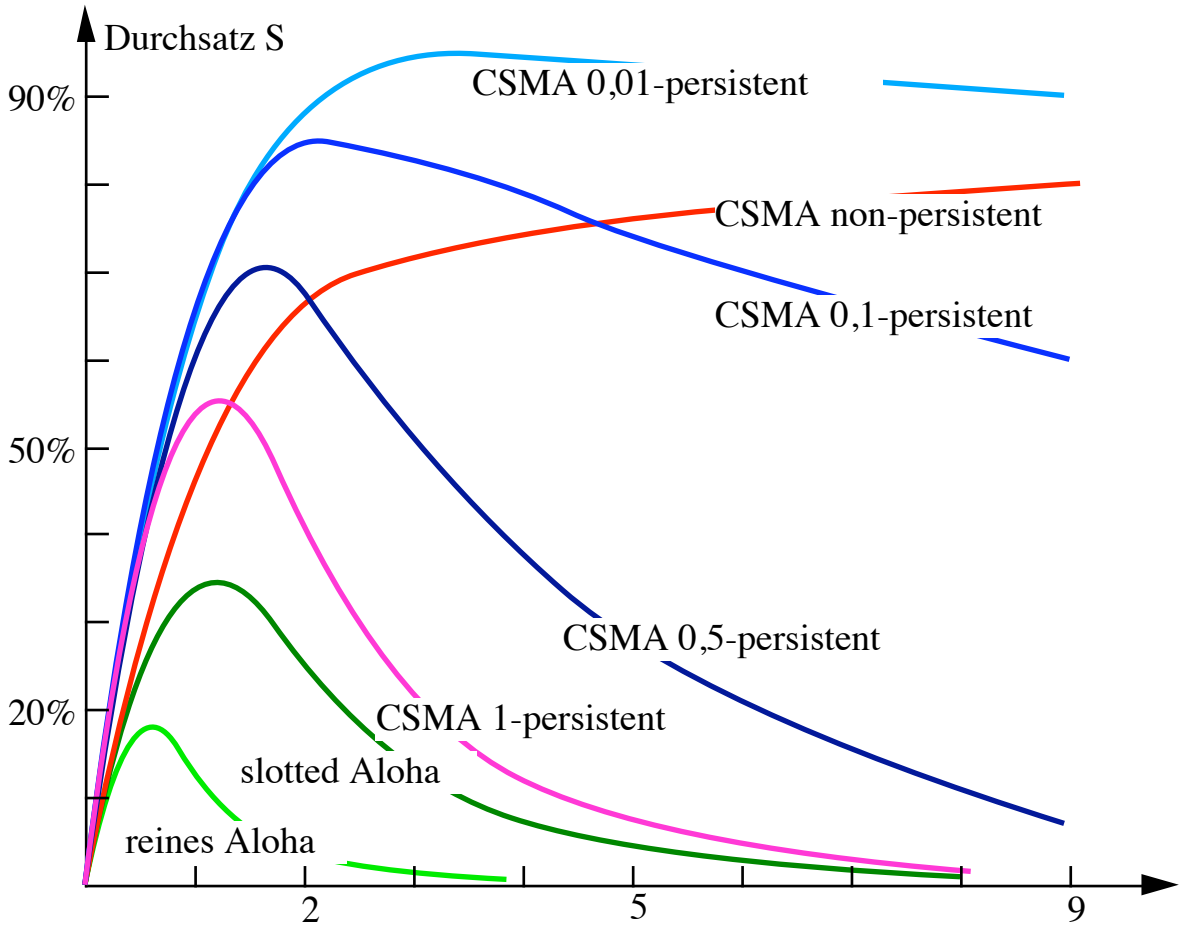


- p-persistent CSMA
  - Medium wird dauernd beobachtet
  - nach dem Freiwerden des Mediums Zufallsentscheidung
  - nur mit Wahrscheinlichkeit p wird sofort gesendet
  - sonst wieder von vorn

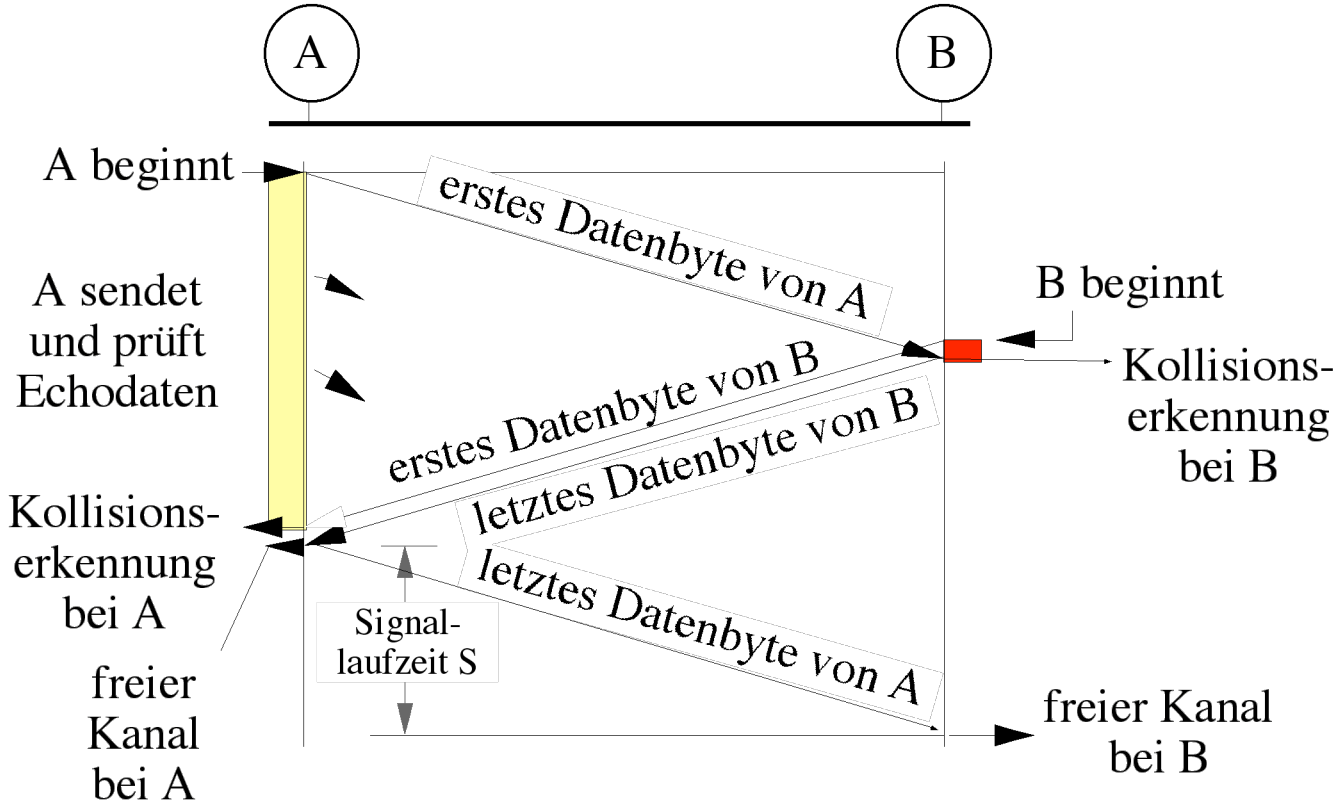
```
while (not p or carrier)
      do Wait(1);
send;
```



- Poisson-verteilte Last



- CSMA/CD: Carrier Sense Multiple Access with Collision Detection
  - Kollision frühzeitig erkennen.
  - Transceiver beobachtet Leitung
  - Beobachten des Mediums während der Übertragung
  - Kollision: Senden abbrechen und Störimpuls senden (Jamming)



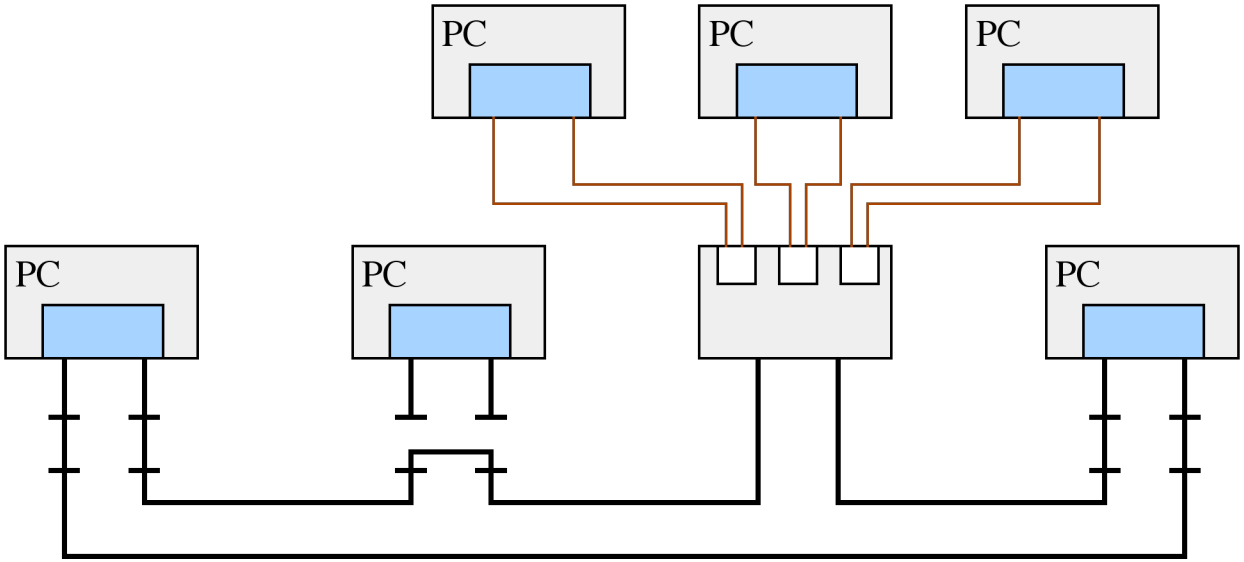
- Kollisionsfenster: slot-time = 2 * Laufzeit + Sicherheitszeit

- Exponential Backoff
  - vor Wiederholung nach einer Kollision wird die Wartezeit verdoppelt
  - Begrenzung auf 2^{10} Slots
  - Anpassung an die Netzlast
  - trotzdem oft lange Wartezeiten
- CSMA/CA: Carrier Sense Multiple Access with Collision Avoidance
  - Probepaket an Empfänger senden (RTS)
  - Auf Antwort warten (CTS)
  - kommt CTS ungestört: Informationspaket senden
  - sichert auch, daß Empfänger bereit
  - RTS/CTS + Wartezeiten verhindert Kollision der Datenpakete
  - RTS kann mit anderem RTS kollidieren

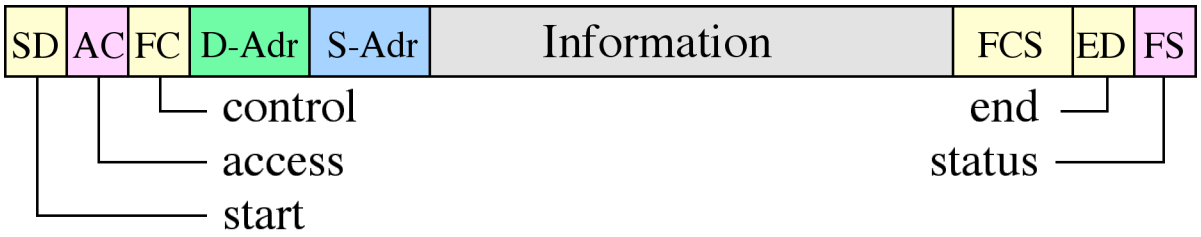
  -> Sender muß nicht mithören, weniger Hardwareaufwand
  - siehe LocalTalk

1.7.1.2 Deterministische Verfahren

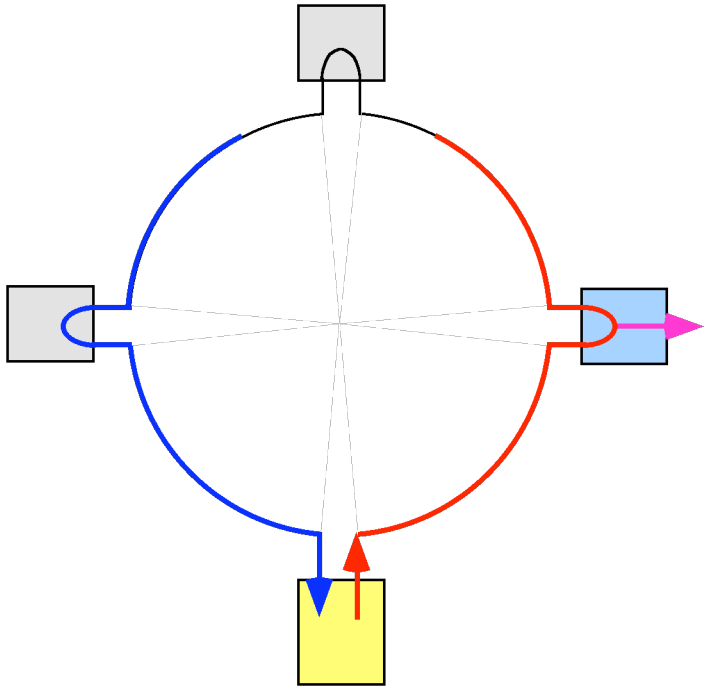
- Token-Ring
  - Interface ist Repeater
  - passiv: mechanische Durchschaltung (Relais)
  - aktiv: elektronische Durchschaltung (Schieberegister)



- Rahmenformat



- Sendeerlaubnis/Token zirkuliert im Ring
  - im access-Feld
  - T-Bit = 0 => Frame ist Token
  - T-Bit = 1 => Frame ist normales Datenpaket
- Empfangen:
  - im Ring zuhören und Adresse erkennen
  - Bitweise vom Repeater kopieren
  - Bestätigung am Paketende anbringen
- Senden
  - Token abwarten
  - Token vom Ring entfernen
  - Ring auftrennen
  - Paket senden
  - Token senden
  - Auf den eigenen Paketanfang warten
  - Bestätigung + Paketende empfangen
  - Ring schließen

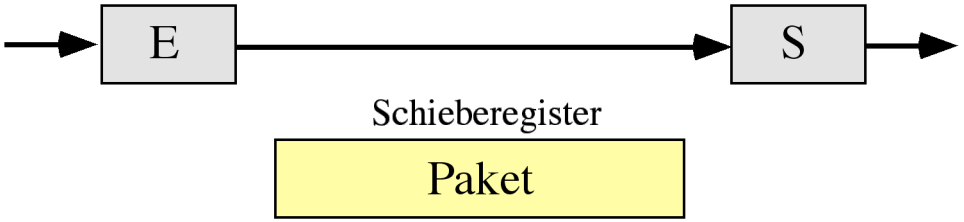


- Problem:
  - Token erkennen und vom Ring nehmen
  - Token und StartOfFrame in einem Bit unterschiedlich
  - ein Bit im Repeater zwischenspeichern,
  - Bit testen und verändern.

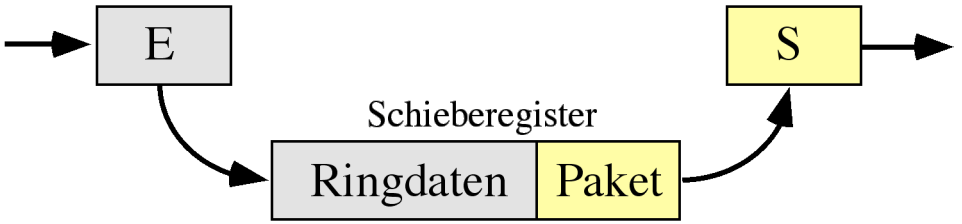
  => mindestens 1 Bit Verzögerung pro Station.
- Kollisionsfrei, Durchsatz nahe bei 100%
- Wartezeit bis zum Senden?
  - Tokenlaufzeit (TRT) + Pakete der Vorgänger
  - Vollast: n-1 Stationen kommen vorher

- Leerlauf:
  - viele Stationen im Ring => auch im Leerlauf hohe Wartezeiten.
  - Pro Station >1 Bit zusätzliche Verzögerung.
  - CSMA gibt im Leerlauf kurze Wartezeit.
- Protokoll recht umfangreich (Prioritäten etc.)
- TokenBus ist logisch ein Ring, Stationen reichen Token herum

- Registereinfügung
  - Schieberegister in jeder Station
  - Paketlänge = Schieberegisterlänge
  - Senden durch 'Vordrängeln'
- Station sendebereit
  - Paket im Schieberegister
  - Warten auf Paketanfang



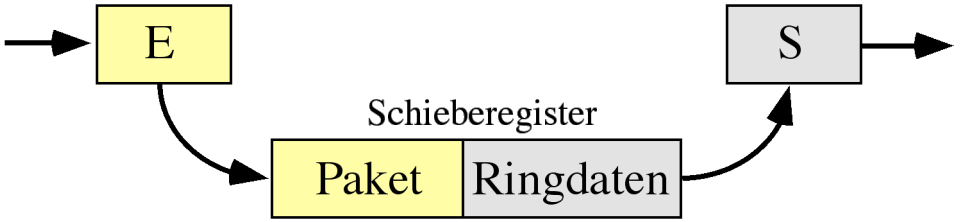
- Senden
  - Ringinhalt in Register schieben
  - Paket 'im Ring'



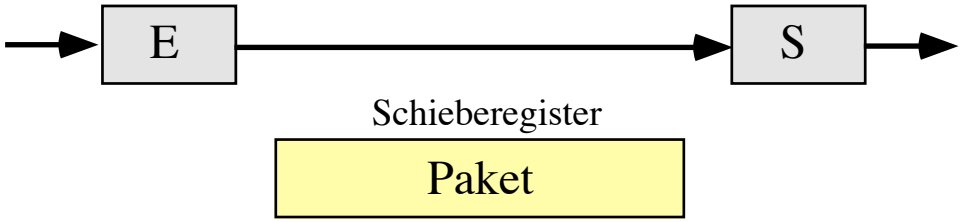
- Paket vollständig gesendet



- Eigenes Paket wieder ins Schieberegister

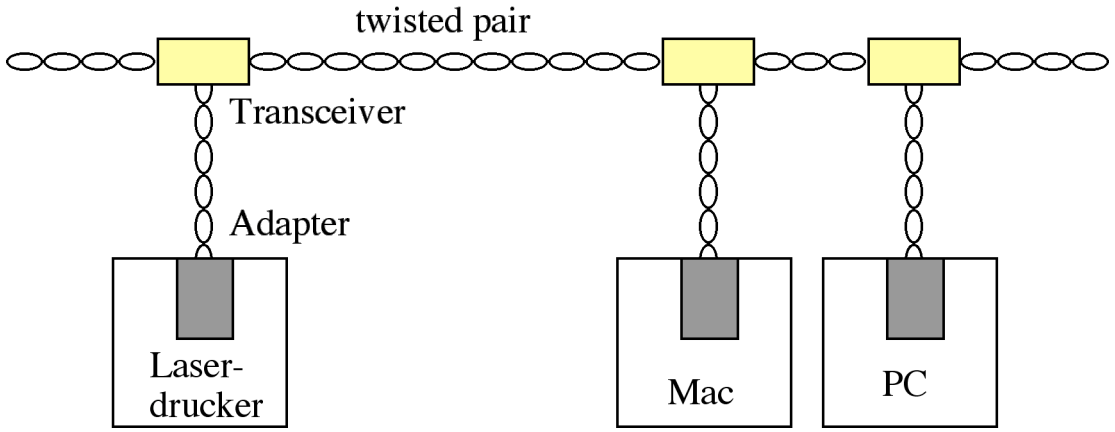


- Eigenes Paket vollständig im Register
  - Ring kurzschließen = Paket vom Ring nehmen



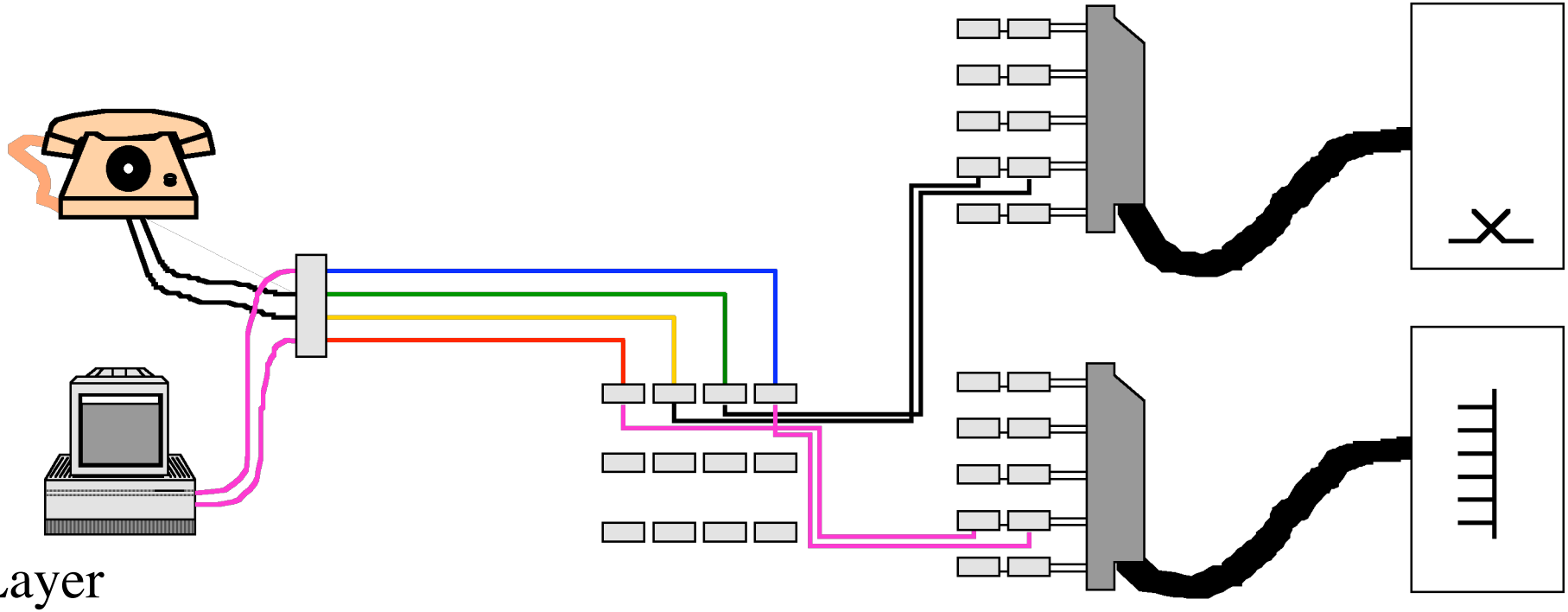
1.7.2 LocalTalk

- Teil der AppleTalk Architektur
  - physical layer und link layer
- basiert auf USART
  - Zilog 8530 SCC
  - HDLC inkl. CRC
  - asynchrone und synchrone Funktion
  - bis 230 400 bit/s
  - zwei Ports



- LocalTalk-Verkabelung
  - trivialer Transceiver mit Transformatoren als Übertrager
  - einfaches twisted pair Kabel

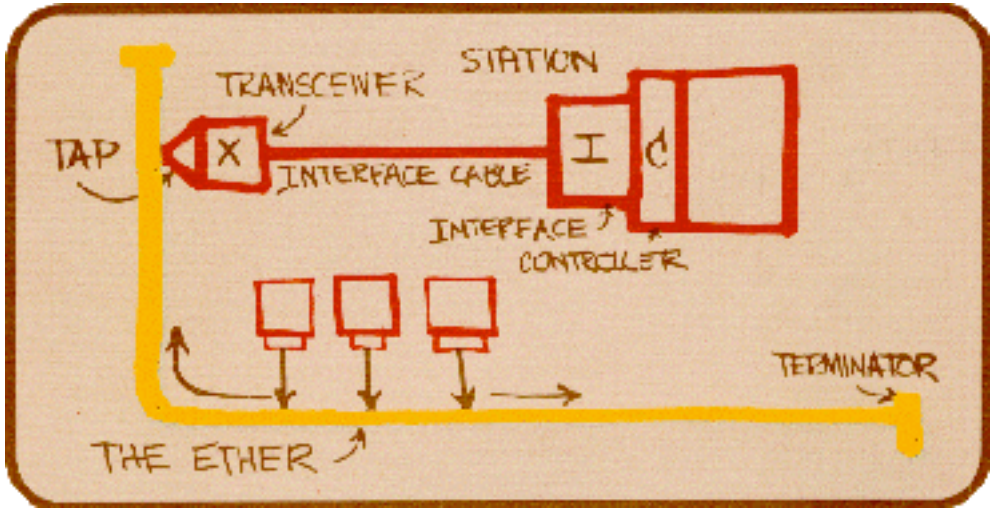
- PhoneNet auf 2 Telefondrähten
  - äußeres Paar der US-Vierdrahtverkabelung
  - vorhandene Kabel mit RJ11-Stecker
  - Sterntopologie
- Starcontroller



- Link Layer
  - HDLC-artiger Rahmen
  - dynamische Vergabe der Knotenadressen

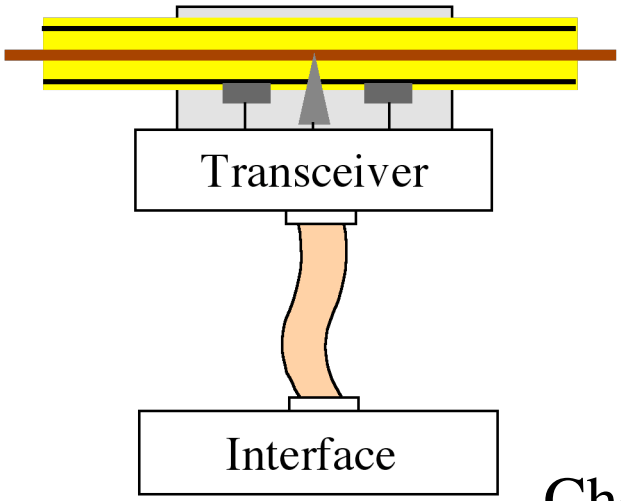
1.7.3 Ethernet

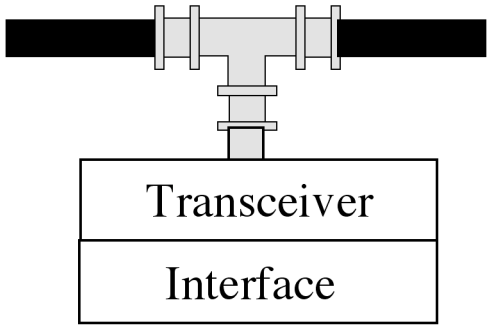
- Bob Metcalfe, David Boggs, Xerox PARC



- CSMA/CD, Kollisionserkennung
  - zwei Pakete auf dem Medium = Signalüberlagerung
  => Manchester - Code - Verletzung
  - |Gleichspannungsanteil| > U_{th} => Kollision
  - DC ohne Kollision > -1.293 V => -1,448V ≤U_{th} ≤ -1,59V
- Größenbeschränkung des Kollisionsgebietes
  - 2500 m => Signallaufzeit maximal 23 μsec
  - nach round-trip von 46,4 μsec (= 464 bit) keine Kollision mehr
  - Paketlänge max. 12.144 bit

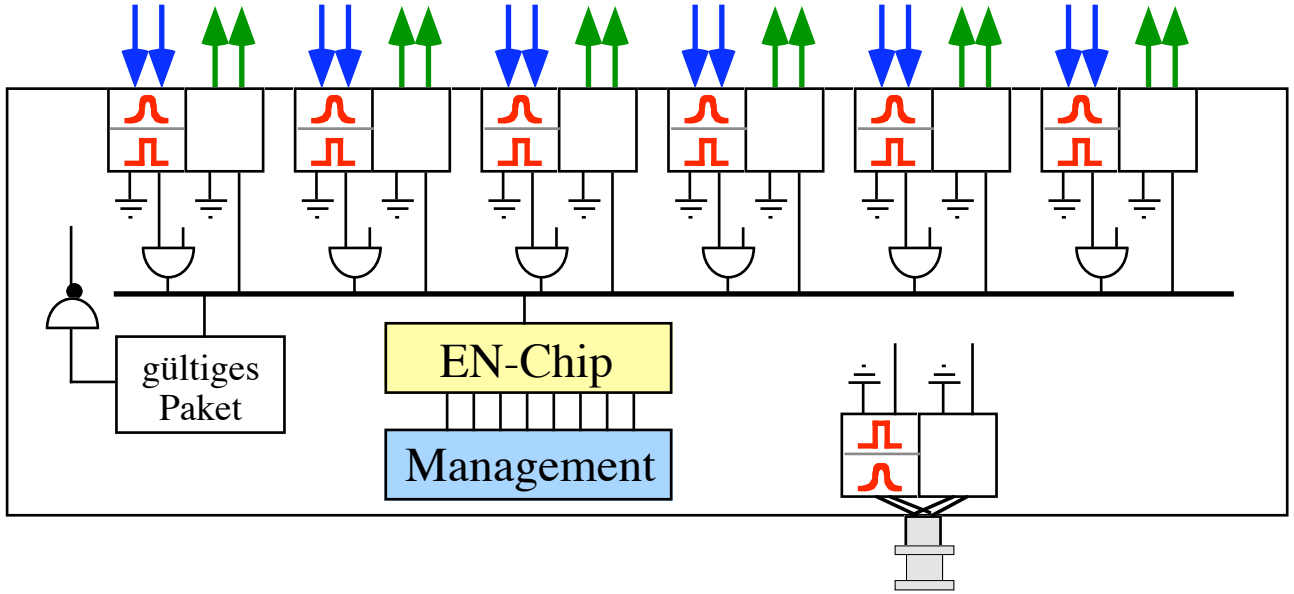
- AUI
  - Interface am Computer mit AUI-Stecker
  - Transceiver am Kabel mit Tap
- 10Base5 (ThickNet, Yellow Cable)
  - 50 Ω Koaxialkabel, 10 mm, starr
  - 500 m





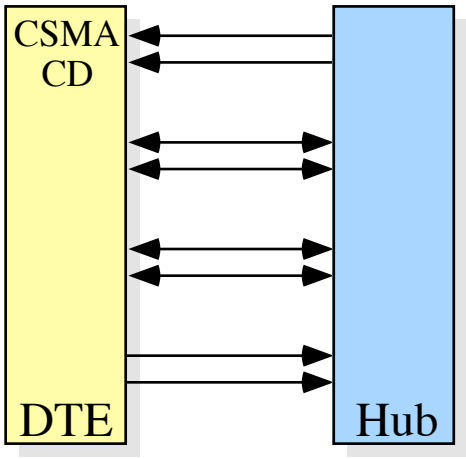
- 10Base2 (ThinNet, Cheapernet)
  - 50 Ω Koaxialkabel, 5 mm, RG58A/U, BNC-Stecker
  - 200 m
- 10Broad36 [Sytec]
  - baumstrukturiertes Kabelverteilnetz, Koaxialkabel 75 Ω
- 10BaseT
  - 4 Drähte twisted pair: 2 senden, 2 empfangen

- Verkabelung wie Telefonsystem
- Sternkoppler im 'wiring cabinet'

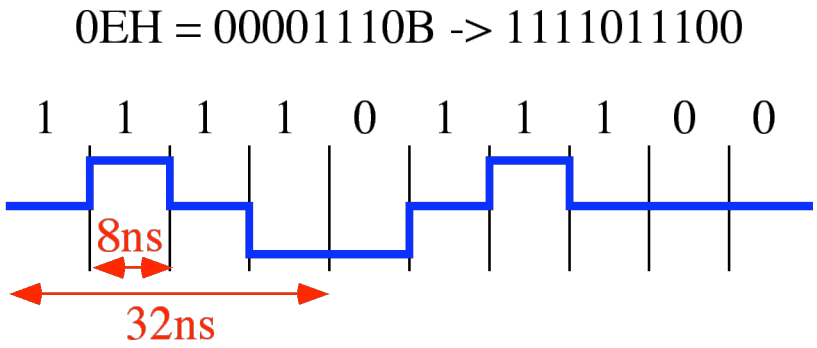


- Gültiges Paket => andere Rx-Leitungen abschalten
- Kollisionen = Carrier-Sense auf dem Empfangspaar AND Senden

- Fast Ethernet (802.3μ)
  - 100 Mbit/s und GigaBit
- CSMA/CD funktioniert bei dieser Geschwindigkeit nicht mehr dezentral
  - Sterntopologie
  - zentrales CSMA/CD
  - Sternkoppler bietet Möglichkeit der Vermittlung
- 100VGAnyLAN (IEEE 802.12)
  - Demand Priority statt CSMA/CD
  - Baumstruktur
- 100BaseT4
  - 4 Adernpaare bis zu 100 Meter, cat 3, 4, 5
  - 8B6T, Symbolrate 25 MHz

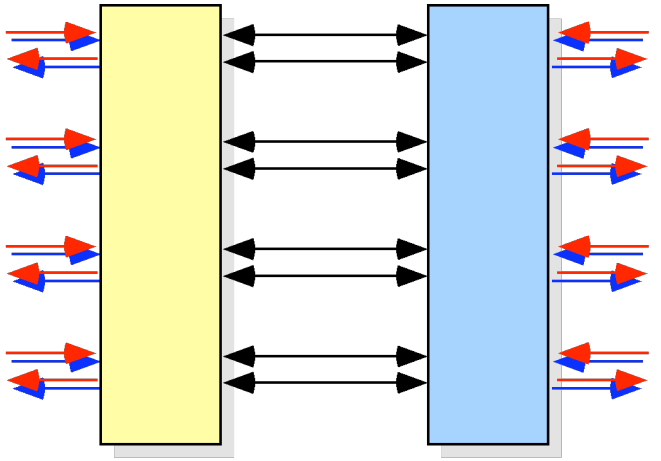


- 100BaseTX
  - 2 Adernpaare, cat 5
  - 4B5B
  - manchesterkodiert 125 MHz: nicht bei cat 5
- Leitungskodierung MLT-3 (NRZI-3)
  - reduziert Frequenz
  - reduziert Gleichstromanteil
  - 11111 -> 32 ns = 31,25 MHz



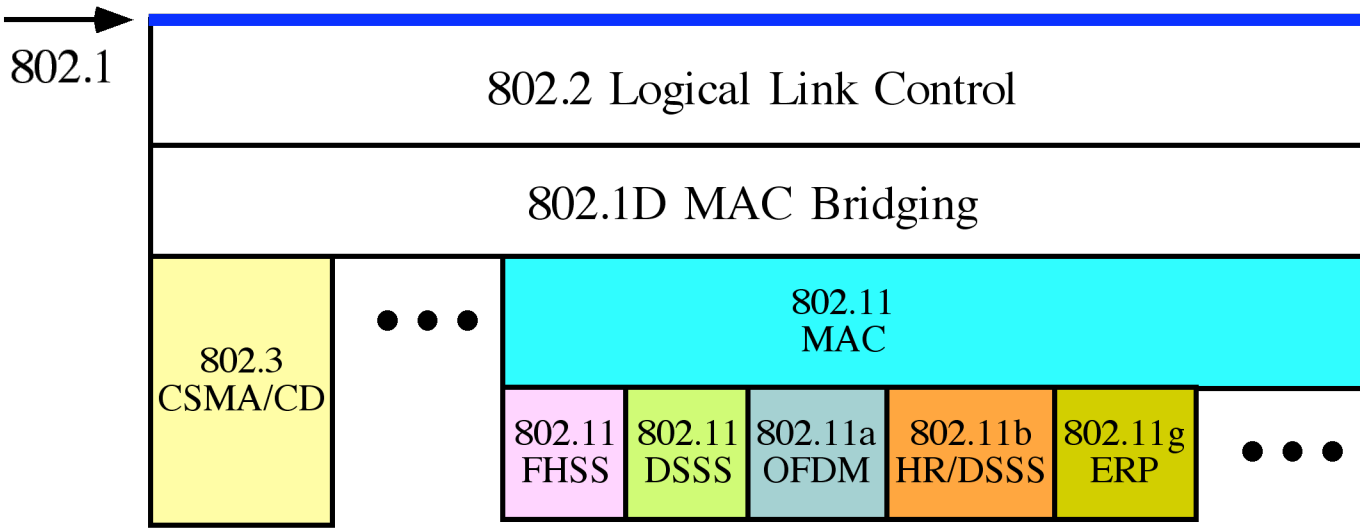
- 4B5B
  - 32 mögliche Kombinationen
  - 16 für Nutzdaten mit mind. 2 Levelwechseln pro Symbol -> Taktableitung
  - Start_of_Frame, End_of _Frame, Idle

- Gigabit-Ethernet (802.3z)
  - T: UTP, cat 5, 100m
  - Gradienten-Multimode Glasfaser 500m
  - SX: (850 nm, max. 5km); LX: (1300nm, max. 500 m)
  - CX: 25 m, "TwinAX"
- 1000Base-X
  - kleine Pakete vs. Collision-Domain (64 bytes ~ 20 m)
  - minimale Frame-Größe 512 bytes
  - Padding, Extension, Frame Bursting (folgende kleine Frames sofort senden)
- 1000Base-T
  - 4 Paare vollduplex
  - Echounterdrückung
  - 8 Bit vom Interface = 4*2 Bit
  - 125 MSymbole/s pro Paar (125 Mbaud)
- 4D-PAM5 (8B/10B)
  - 2 Bit –> 5 Werte
  - 256 Werte in 625 Symbolwerten kodieren
  - 512 Werte für Trellis-Kodierung, 113 für Kontrolle
  - Viterbi-Decoder



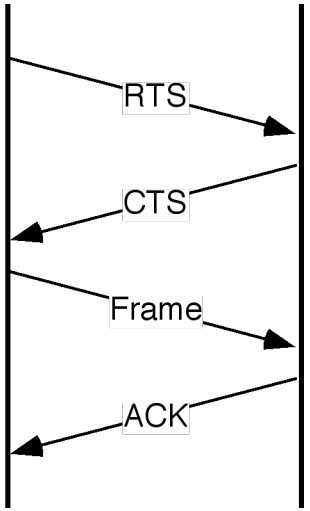
1.7.4 Wireless LANs - 802.11

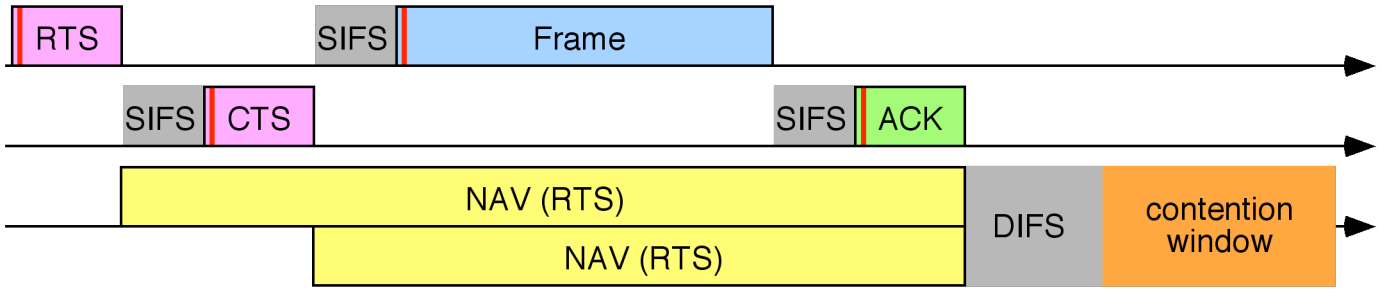
- Integration von lokalen Funknetzwerken in IEEE 802
  - neuer MAC-Layer 802.11
  - drahtlose PHY-Layer



- ISM-Band (industrial, scientific, and medical): 2.4 GHz
- Modulationsverfahren
  - FHSS: frequency hopping spread spectrum
  - DSSS: direct sequence spread spectrum
  - OFDM: orthogonal frequency divison multiplexing (5 GHz)
  - HR/DSSS: higher rate direct sequence spread spectrum
  - ERP: Extended Rate Physical (ERP-DSSS und ERP-OFDM)

- Netzwerkstruktur
  - Access Points
  - zellulärer Charakter
- MAC-Layer 802.11
  - Kanal unzuverlässig: Frames bestätigt übertragen
  - atomic operation: Frame+ACK
  - DCF - Distributed Coordination Function: CSMA/CA
  - Point CF, Hybrid CF
- Virtual Carrier Sense: Network Allocation Vector
  - physical CS teuer und hidden node Problem
  - NAV: Feld *duration* im Frame [μsec]
  - Stationen beobachten duration: NAV-counter

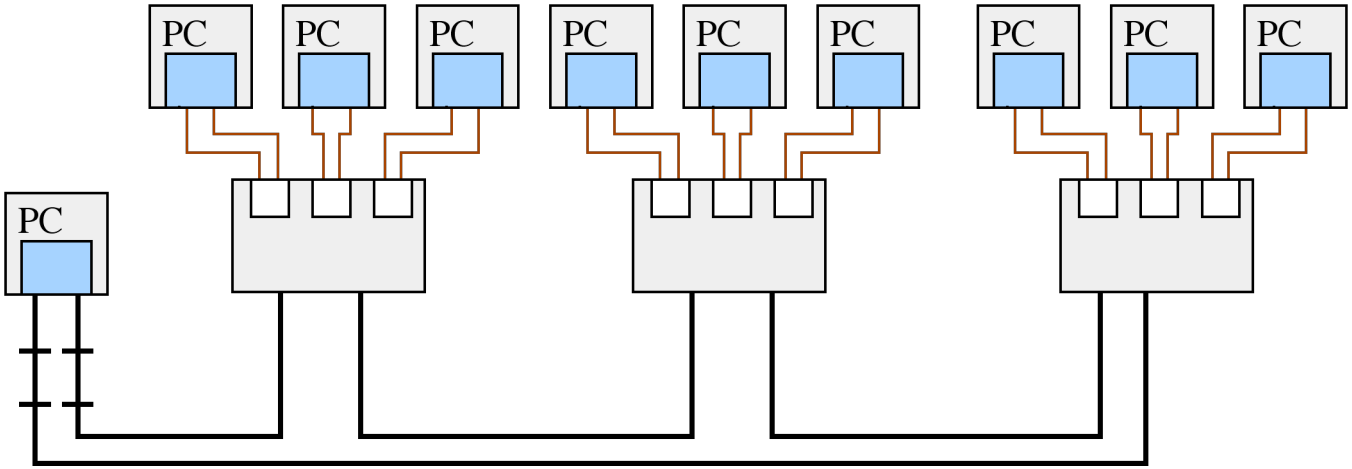




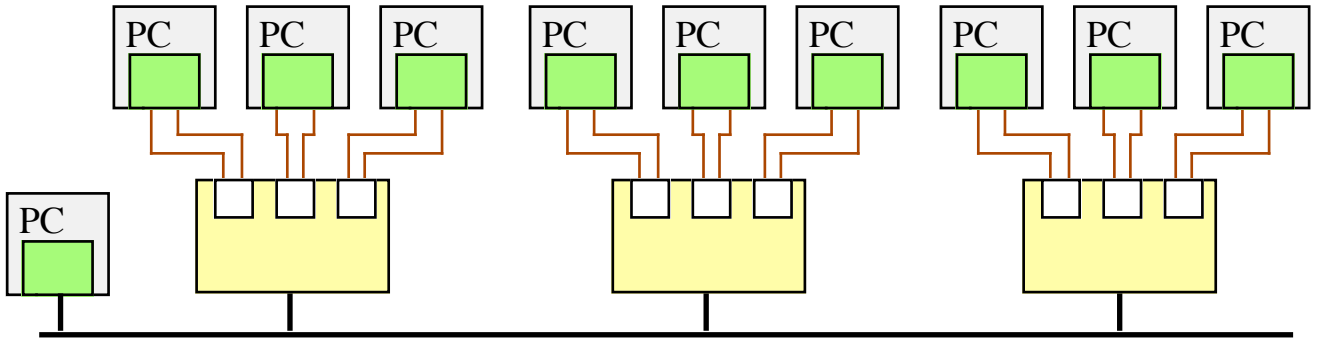
  - short interframe space, DCF interframe space

1.7.5 Token Ring

- IBM Zürich
- 4 MBit/s oder 16 MBit/s
- Verkabelung im wesentlichen sternförmig

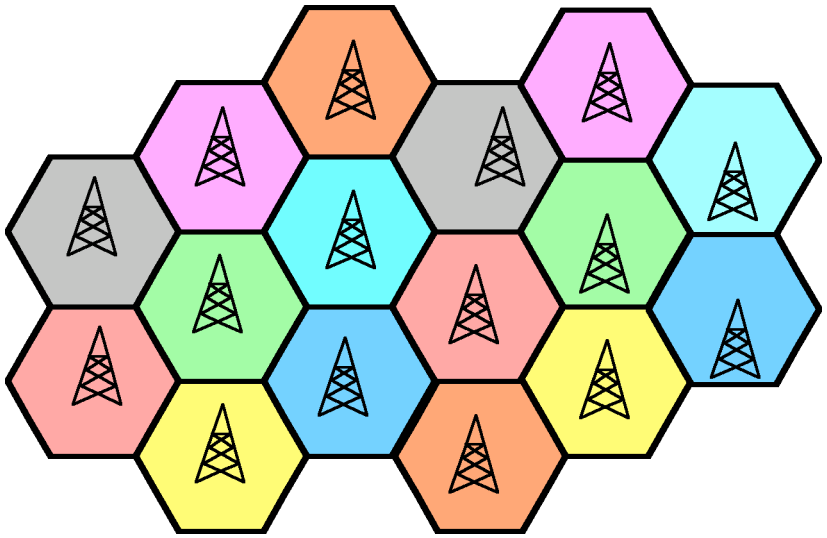


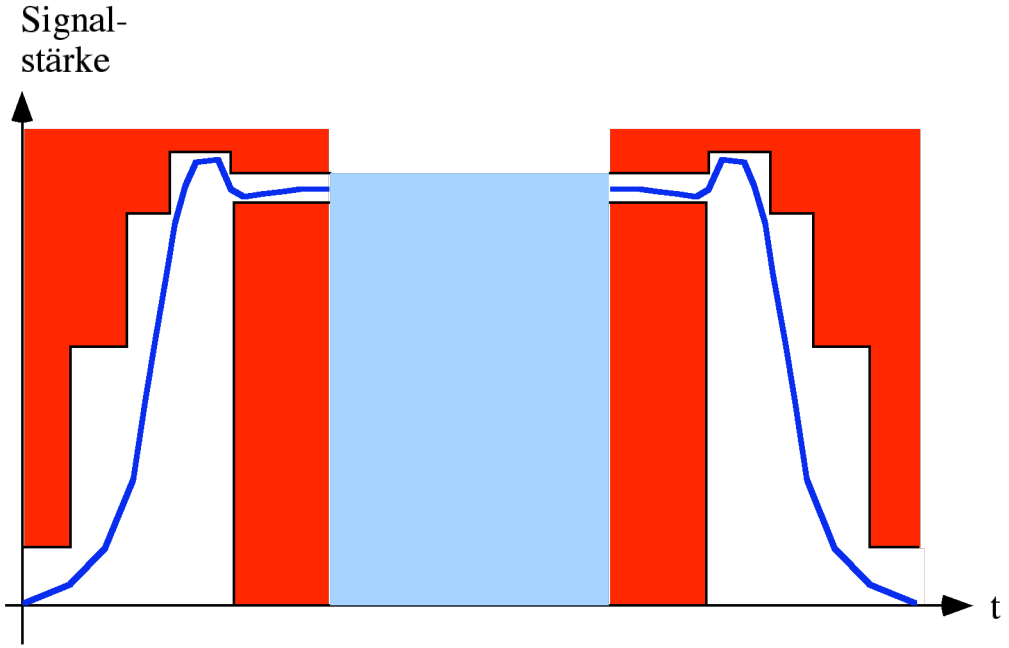
- Synoptics LatticeNet



1.8 Technik digitaler Mobilfunknetze

- Zelluläre Struktur: Frequenzwiederverwendung, Handover



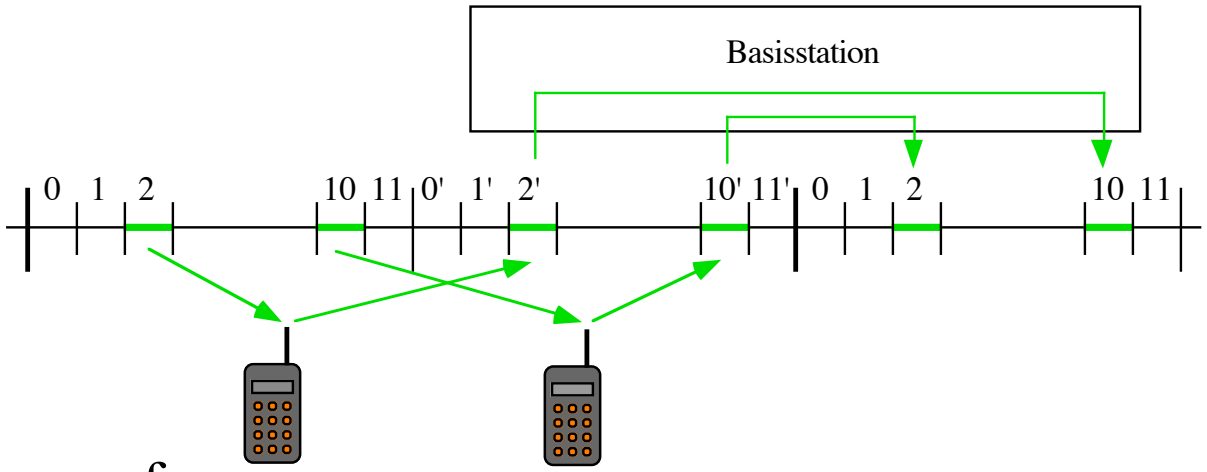


- Zeitmultiplex auf Trägerfrequenz (TDMA)
  - Rahmen, Multirahmen, Superrahmen
  - Timeslots in Rahmen -> Funkburst
  - Schutzzeiten um die Timeslots

1.8.1 DECT (Digital European Cordless Telecommunications)

- Schnurlose Telefonie im Gebäude bzw. Campus
  - evtl. Zellstruktur
  - automatische Zellanordnung
  - Nebenstellenanlagen
- 1880 - 1900 MHz
- 10 Trägerfrequenzen, je 100 Rahmen/sec
- 24 Slots (12 downlink, 12 uplink)

  480 bit - Sync - Schutzzone - Header = 320 bit/slot

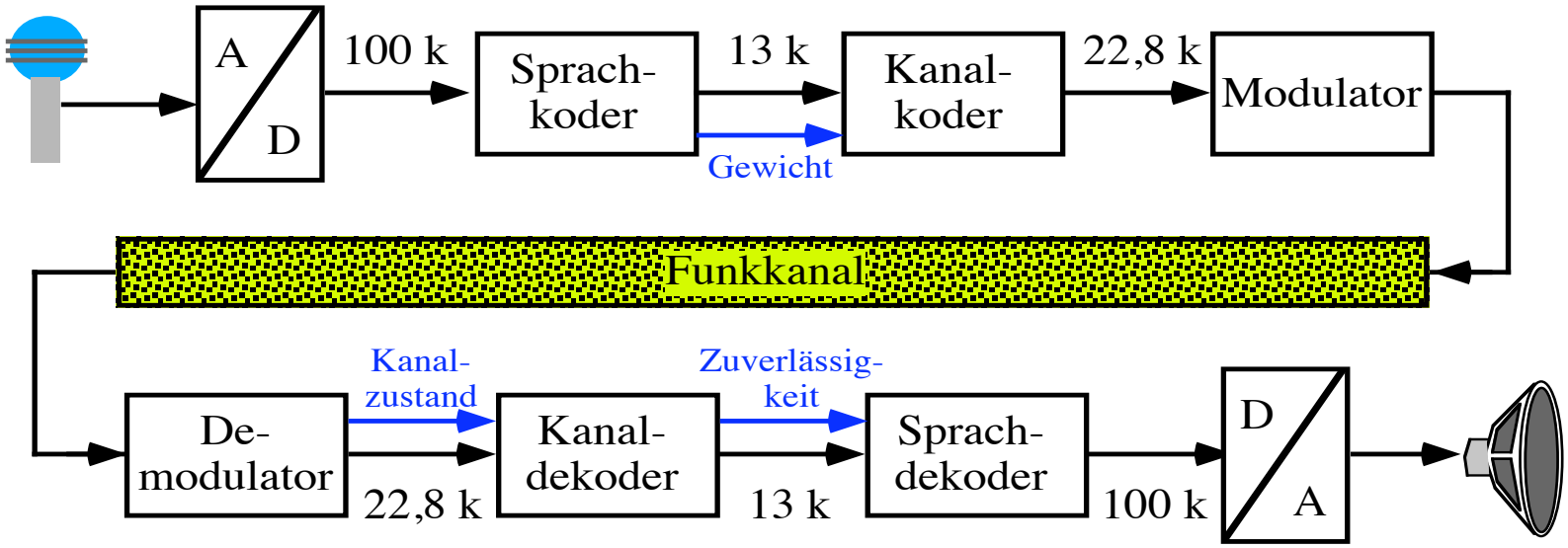


- Sprachkompression auf 32 kbit/s nach G.721 (ADPCM)

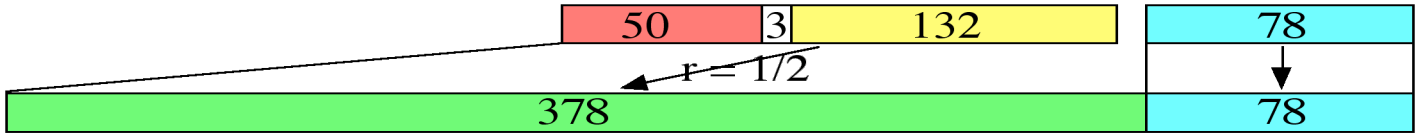
1.8.2 GSM: Global System for Mobile Communications

- Group Spéciale Mobile (GSM)
- später: Global System for Mobile Communication
- 890 - 915 MHz, 935 - 960 MHz, 1800 MHz, 1900 MHz
  - 125 Bänder, max. 15 pro Zelle
  - 217 Rahmen/sec mit 8 Bursts (= Timeslots)
  - 24,7 kbit/s pro Kanal
  - Gaussian Minimum Shift Keying
- Signalisierungskanäle
  - LAPDm
  - Datenrate 950 bit/s
  - Verbindungskontrolle ähnlich Q.930/931 (ISDN)
  - Mobilitätsmanagement

- komplexe Kanalstruktur: TCH, SACCH, FACCH, SDCCH, RACH…
  - Traffic Channel, Slow (Fast) Associated Control Channel
  - Access Grant Channel, Random Access Channel, …
  - Mapping der Kontroll-Kanäle im Slot 0
- Integrierte Quell- und Kanalkodierung
  - Einteilung in geschützte und ungeschützte Information
  - Signal-Qualitätsinformation für den Dekoder

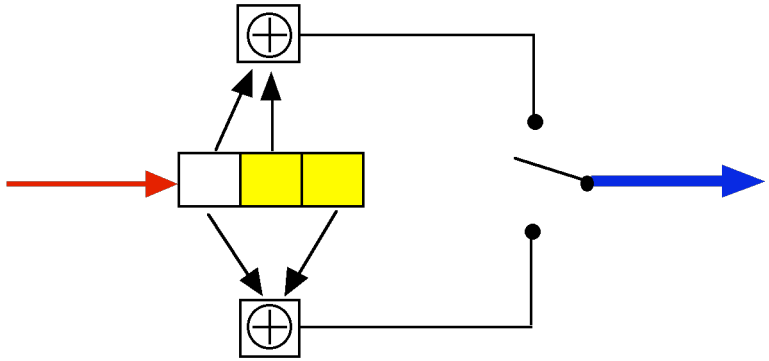


- Nutzkanal TCH
  - Bruttodatenrate 22800 bit/s
  - Sprache 13000 bit/s (Code 260 aus 456)

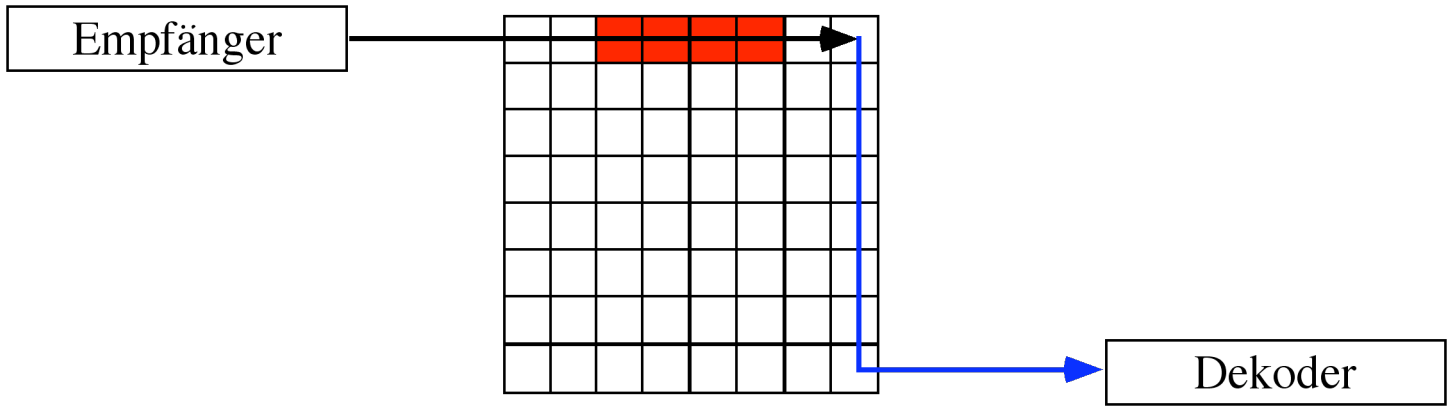


  - Daten 12000 bit/s (Code 240 aus 456)
  - Bitratenadaption …
- GSM-Sprachkoder LPC/RPE
  - Linear Predictive Coding, Long Term Prediction, Regular Pulse Excitation
  - Klasse Ia: LPC-Parameter (MSBs), LTP-Parameter
  - Klasse Ib: LPC-Parameter, Gitter-Auswahl, RPE-Daten (MSBs)
  - Klasse II: LPC-Parameter (LSBs), RPE-Daten (LSB)

- Kanal mit hoher Fehlerwahrscheinlichkeit (10^{-3})
  - Faltungscodes zur Fehlerkorrektur

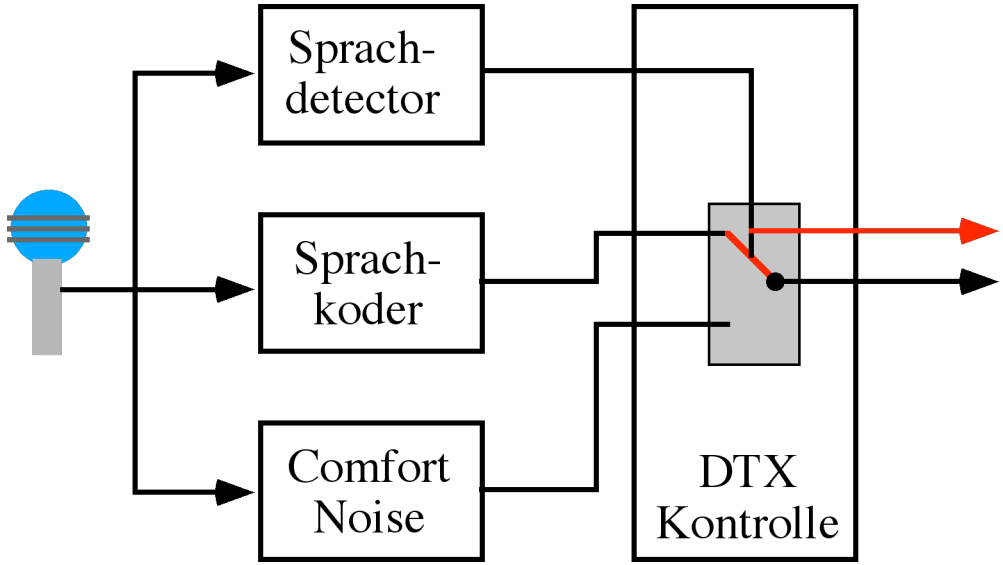


- Interleaving
  - Burstfehler verteilten
  - I = 8 bei Sprache, I = 19 bei Daten



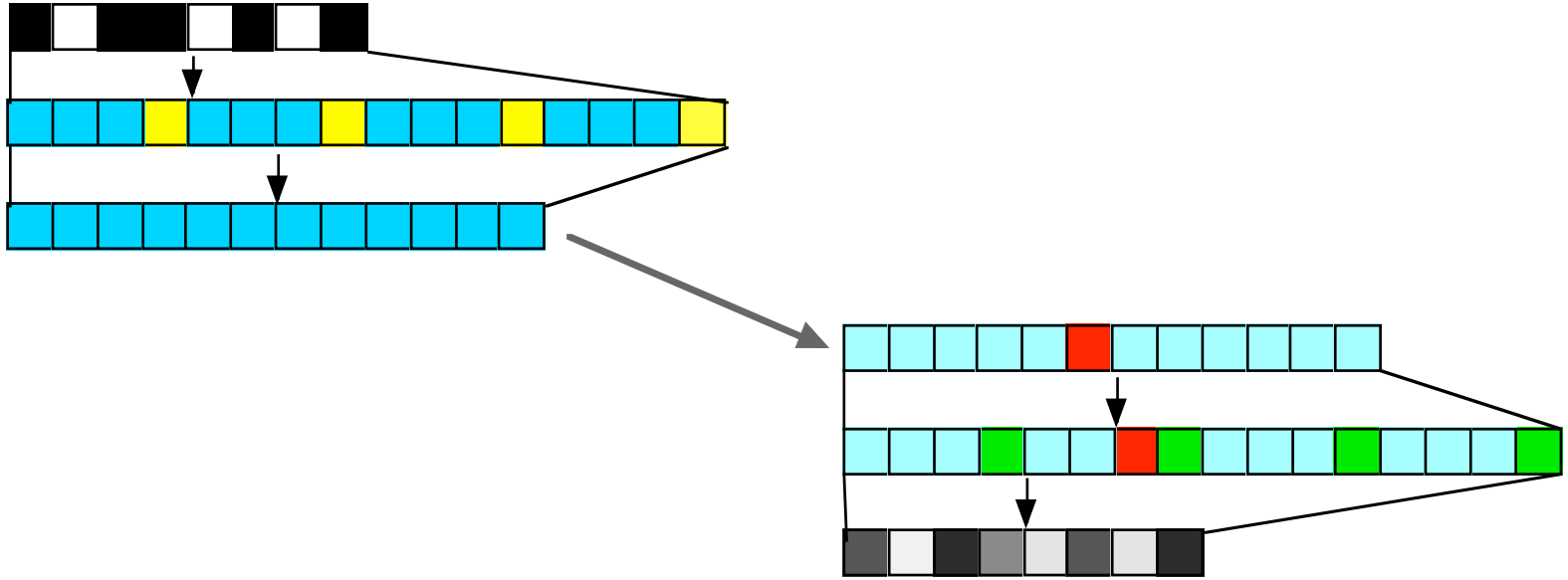
  - Faltungscodes gut bei "gestreuten" Fehlern

- Discontinous Transmission (DTX)
  - Voice Activity Detection
  - In Sprachpausen keine GSM-Bursts senden
  - Periodisch Ruhegeräusch senden
  - Ruhegeräusch im Empfänger abspielen



  - reduziert Interferenzen zu anderen Mobiltelefonen
  - spart Batterie

- Datenübertragung
  - Faltungscode mit Rate 1/2
  - 'Punktierung' = Bits weglassen



  - Dekoder würfelt punktierte Bits
  - Faltungsdekoder korrigiert 'falsch' gewürfelte Bits
- Radio Link Control Protocol
  - ähnlich HDLC / LAP

- Radio Resource Management
  - Messung der Kanalgüte
  - Minimierung der Sendeleistung
  - Frequenzhüpfen
  - Zellwechsel
- Mobilitätsmanagement
  - Roaming
  - MS-ISDN: CountryCode + NatDestCode + SubscriberNumber
- 49 172 7333563
  - Mobile Station Roaming Number (MSRN)
  - Home Location Registry: MS-ISDN -> MSRN
  - Zellwechsel => MSRN im HLR ändern
  - Visitor Location Registry für abgehende Rufe

  => Mobilstation hinterläßt “Spur”

  - Struktur in der SubscriberNumber durch Vertrieb
  - IMEI: International Mobile Equipment Identity
  - IMSI: International Mobile Subscriber Identifier

2. Schalten

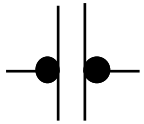
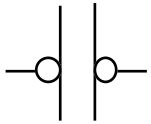
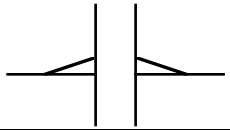
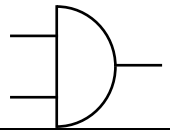
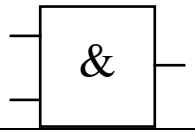
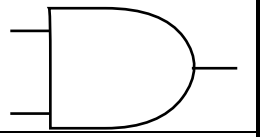
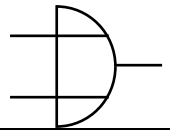
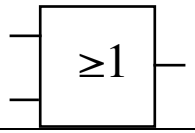
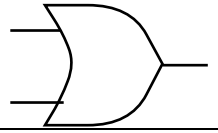
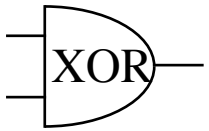
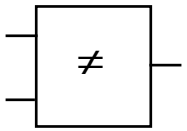
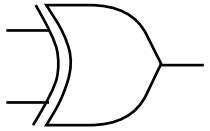
2.1 Transistoren, Gates

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

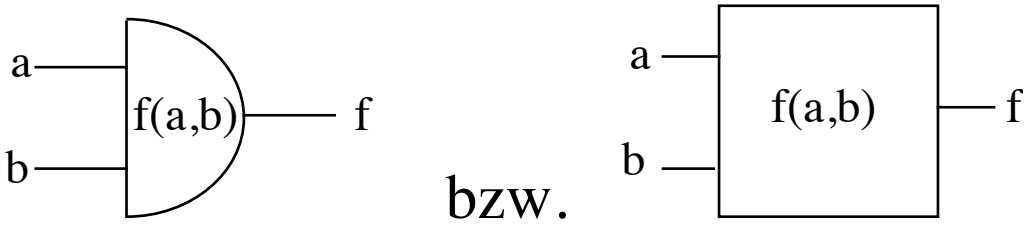
V_1	V_2	V_3	…	V_n	f_1	f_2	…	f_{max}
1	1	1		1	0	0		1
0	1	1		1	0	0		1
0	0	1		1	0	0		1
0	0	0		0	0	1		1

		a=1, b=1	a=1, b=0	a=0, b=1	a=0, b=0
f_0		0	0	0	0
f_1		0	0	0	1
…		…	…	…	…
f_8	AND	1	0	0	0
f_{14}	OR	1	1	1	0
f_{15}		1	1	1	1

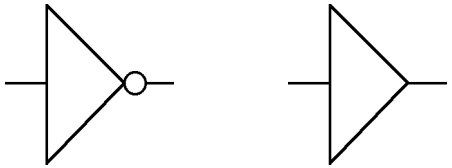
- Symbolische Darstellung von Schaltkreisen
  - Schaltplan
  - DIN 40700, IEEE/ANSI Y32.14, IEC

Funktion	Operator	graphisches Symbol		
		DIN	IEEE	IEC
Negation	$\neg$, $\overline{}$			
Und	$\cdot$,$\wedge$,*			
Oder	+,$\vee$			
Exklusiv Oder	$\oplus$,$\neq$			

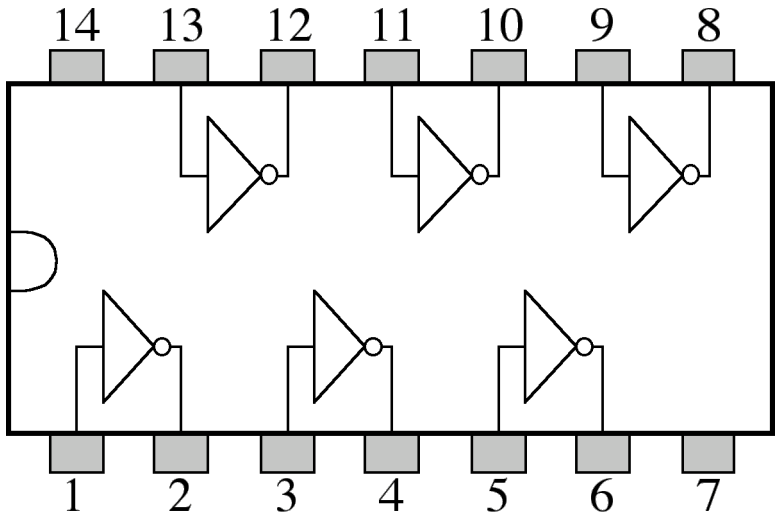
- Allgemeine Schaltfunktionen:



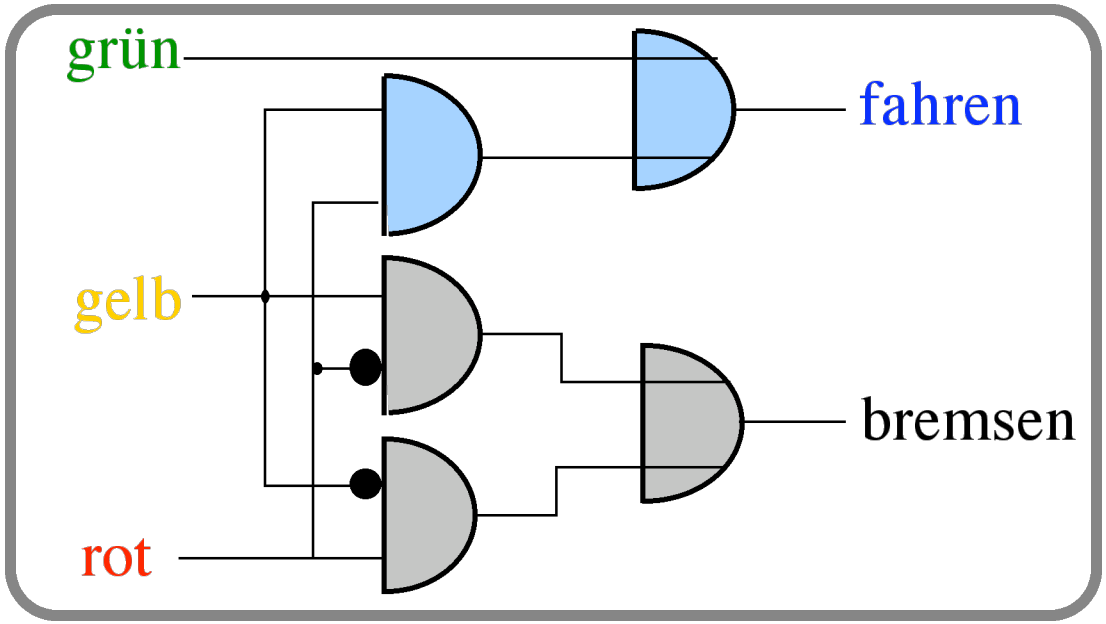
- Inverter, Verstärker:



- Hex Inverter 7404 als Chip:
  - Stift #7: 0 Volt, Erde, GND
  - Stift #14: z.B. +5 Volt:

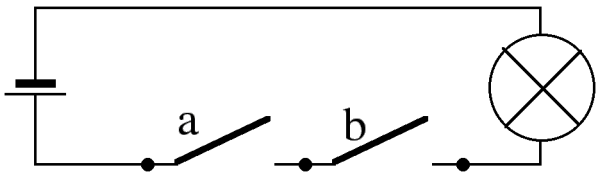


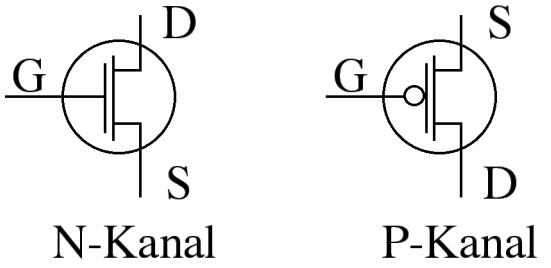
- Ampel-Reaktion



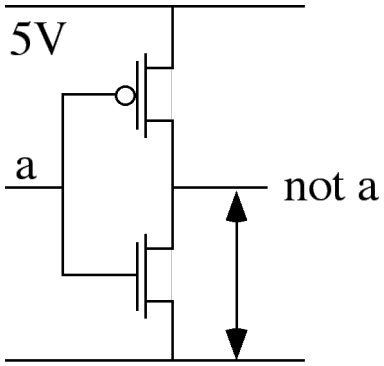
- Schaltalgebra
  - Schaltfunktionen
  - Rechenregeln
  - Normalform
  - Minimierung

- 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 -> I_{DS} steigt exponentiell

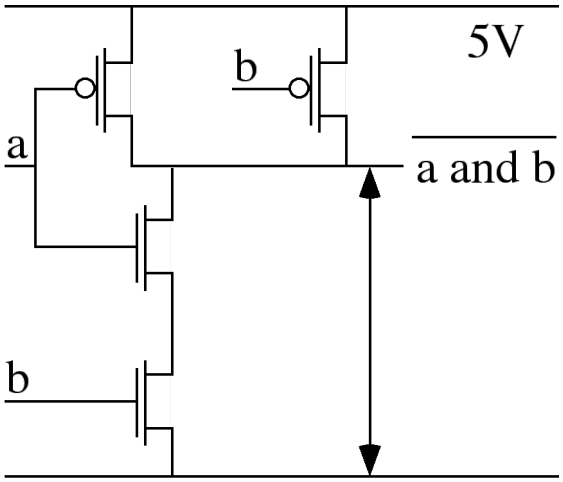


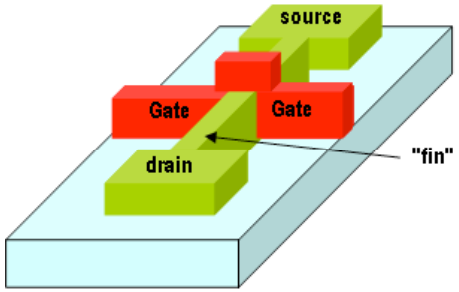


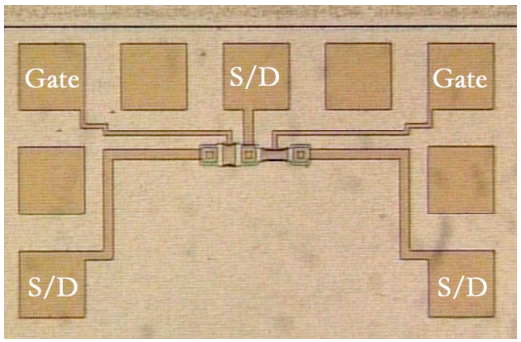
- Negation (NOT)



NAND

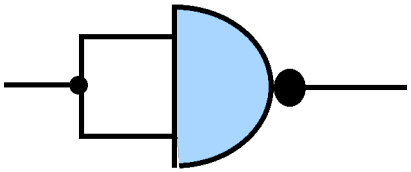




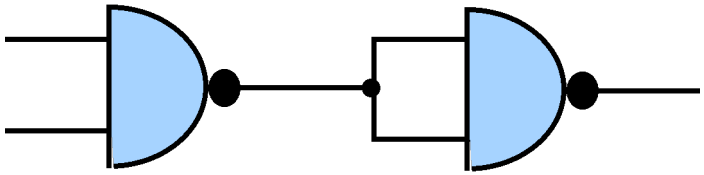


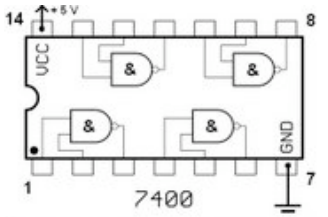
- Boolsche Algebra
- Kombination von NICHT mit UND / ODER
  - $\overline{A \wedge B} = \overline{A} \vee \overline{B}$
  - $A \vee B = \overline{\overline{A} \wedge \overline{B}}$
  - $A \wedge B = \overline{\overline{A} \wedge \overline{B}}$
- NAND und NOR einfacher zu bauen
  - CMOS: complementary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

Inverter aus 1 NAND

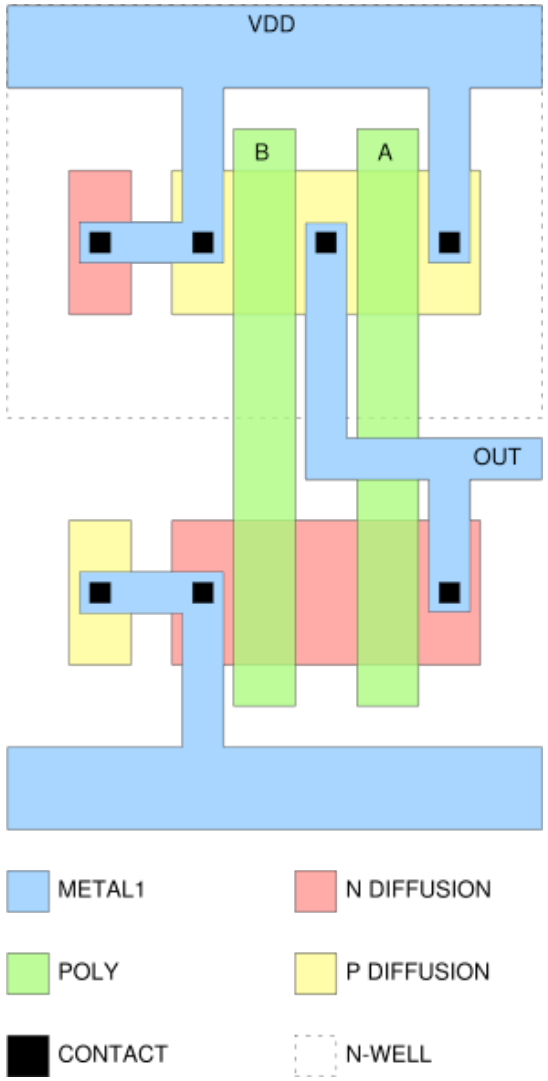


UND aus 2 NANDs





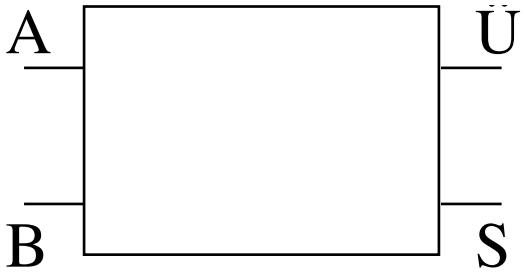




- 74xx, 74HCTxx, …
  - Einfache Logik-Chips
  - 7404: 6 Inverter
  - 7400: 4 NAND-Gatter
  - 7402: 4 NOR-Gatter
  - 7420: 2 NAND-Gatter mit je 4 Eingängen

2.2 Rechnen und Speichern

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



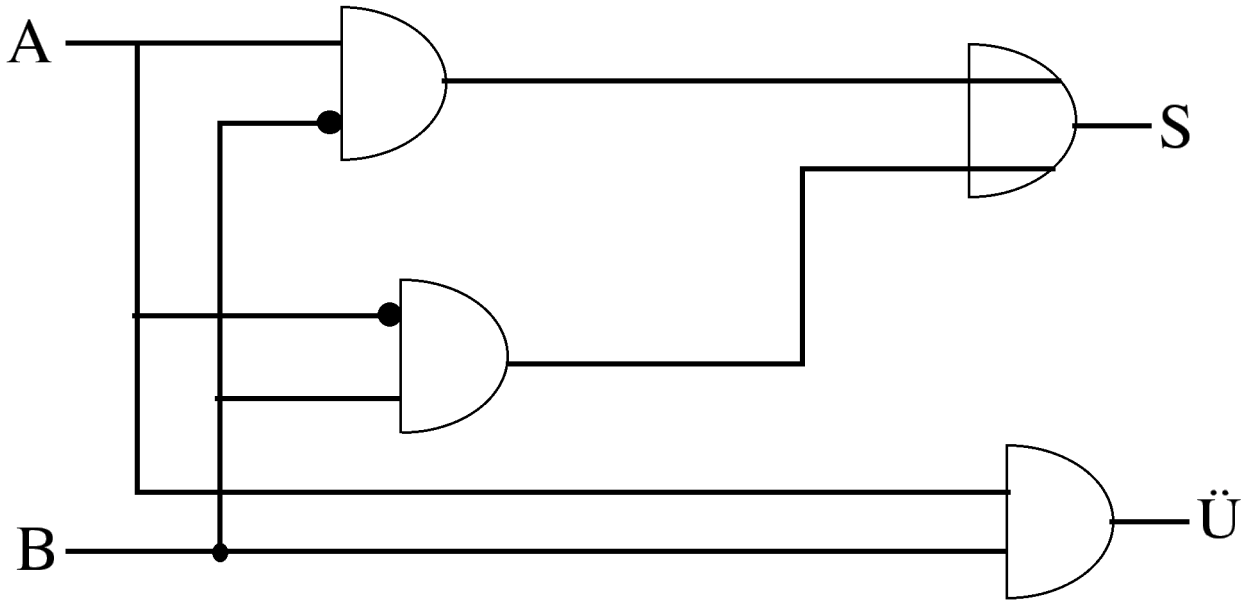
- Wahrheitstafel

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

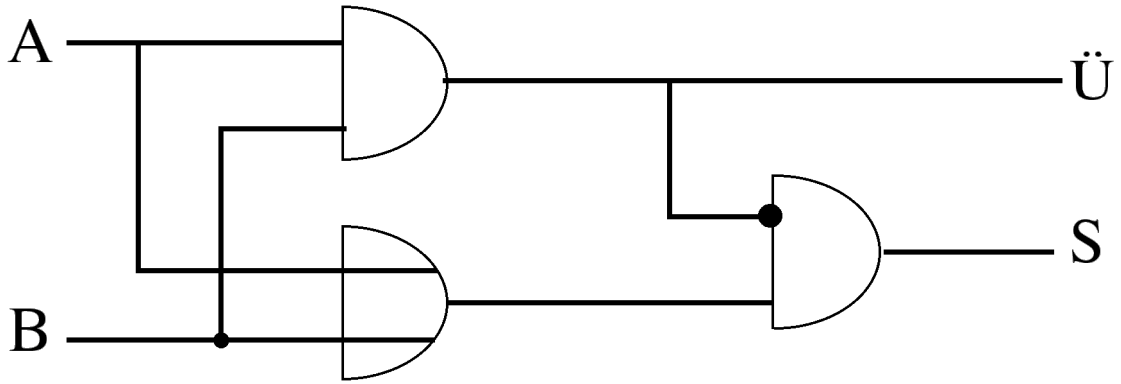
- Addierer kann aus sogenannten Halbaddieren zusammengesetzt werden
- Halbaddierer berücksichtigt nicht den Übertrag der früheren Stufe

- Halbaddierer

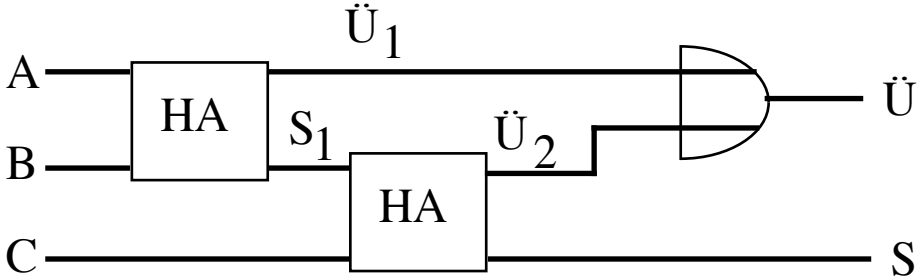
$S = A\bar{B} \vee \bar{A}B \qquad Ü = AB$



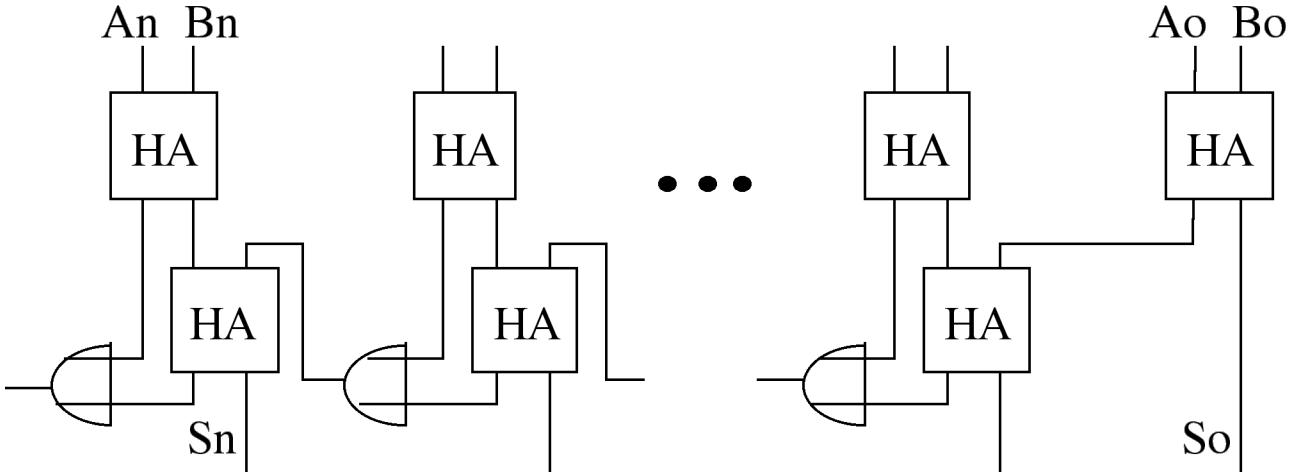
- Vereinfacht



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

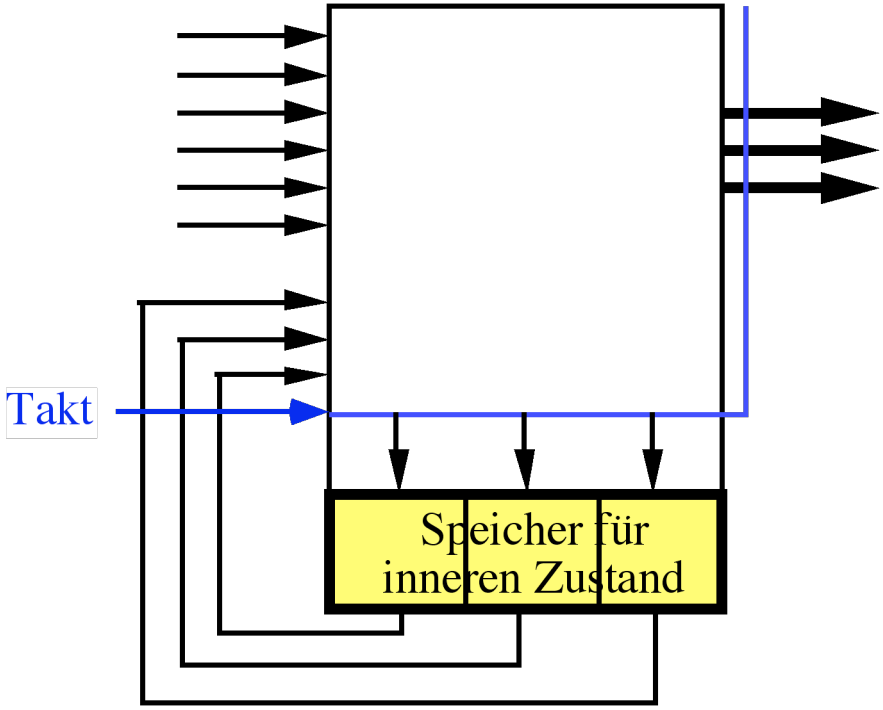


- Paralleladdierer
  - für die Addition eines Binärwortes
  - die Summen der jeweiligen Binärstellen parallel bilden
  - Übertrag durch die Stufen fortpflanzen lassen (Delay!)



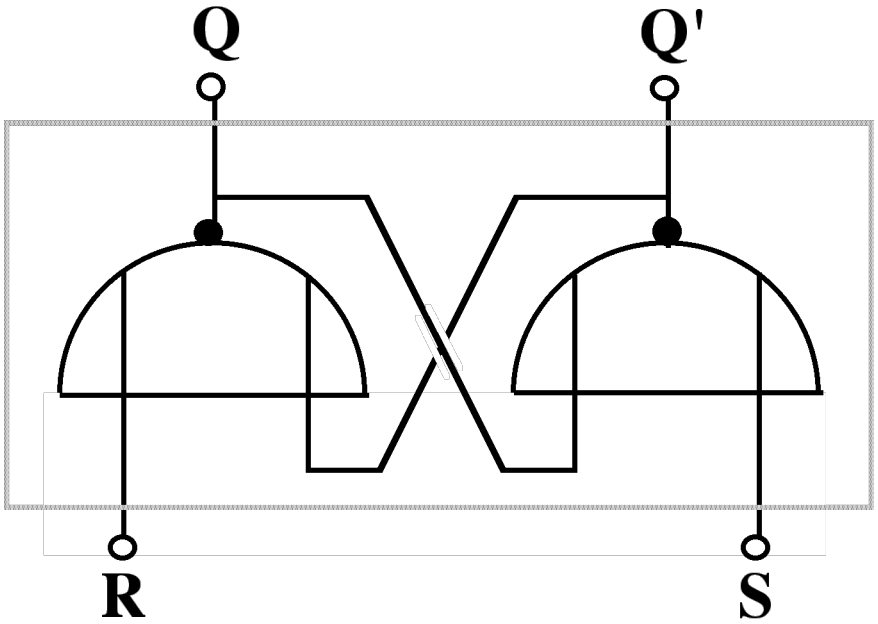
  -> Carry Lookahead

- Sequentielle Schaltungen
  - mit inneren Zuständen
  - beeinflussen Ausgangsfunktionen

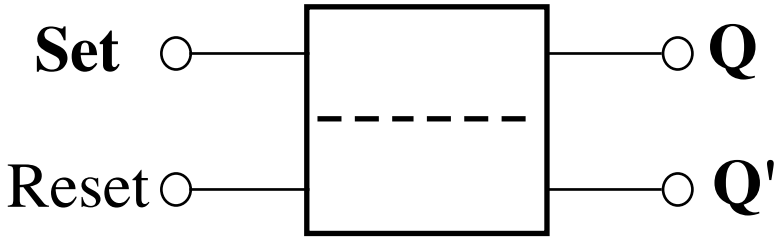


- Takt sorgt für die Sequenzierung
  - Eingänge und Zustand -> Ausgänge und neuen Zustand
  - Neuen Zustand übernehmen und für nächste Taktperiode speichern

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



- sog. RS-Flip-Flop:
  - **S**etzen ("Set"),
  - **R**ücksetzen ("Reset")

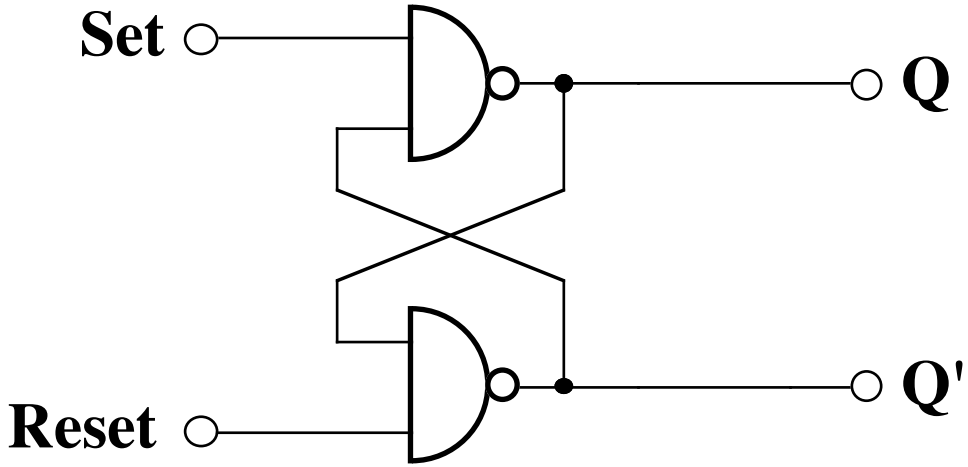


- Schaltfunktion für RS-Flip-Flop:
  - $\mathbf{Q = \overline{R \vee Q'} = \overline{R \vee \overline{(S \vee Q)}} = \overline{R} \wedge (S \vee Q)}$
  - $\mathbf{Q' = \overline{(S \vee Q)} = \overline{S} \wedge (R \vee Q')}$
- Wahrheitstafel:

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

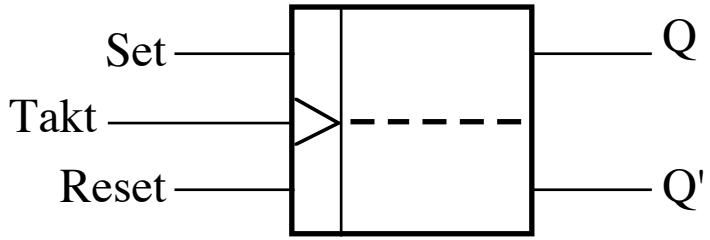
- RS-FF aus Nand-Gattern:



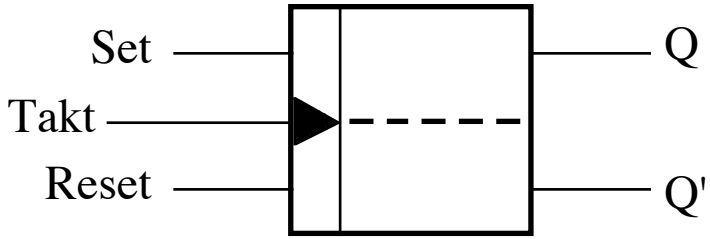
- => modifizierte Wahrheitstabelle:

S	R	Q	Q'	
1	1	Q_{alt}	Q'_{alt}	Speichern
0	1	1	0	Setzen
1	0	0	1	Rücksetzen
0	0	1	1	Unzulässig

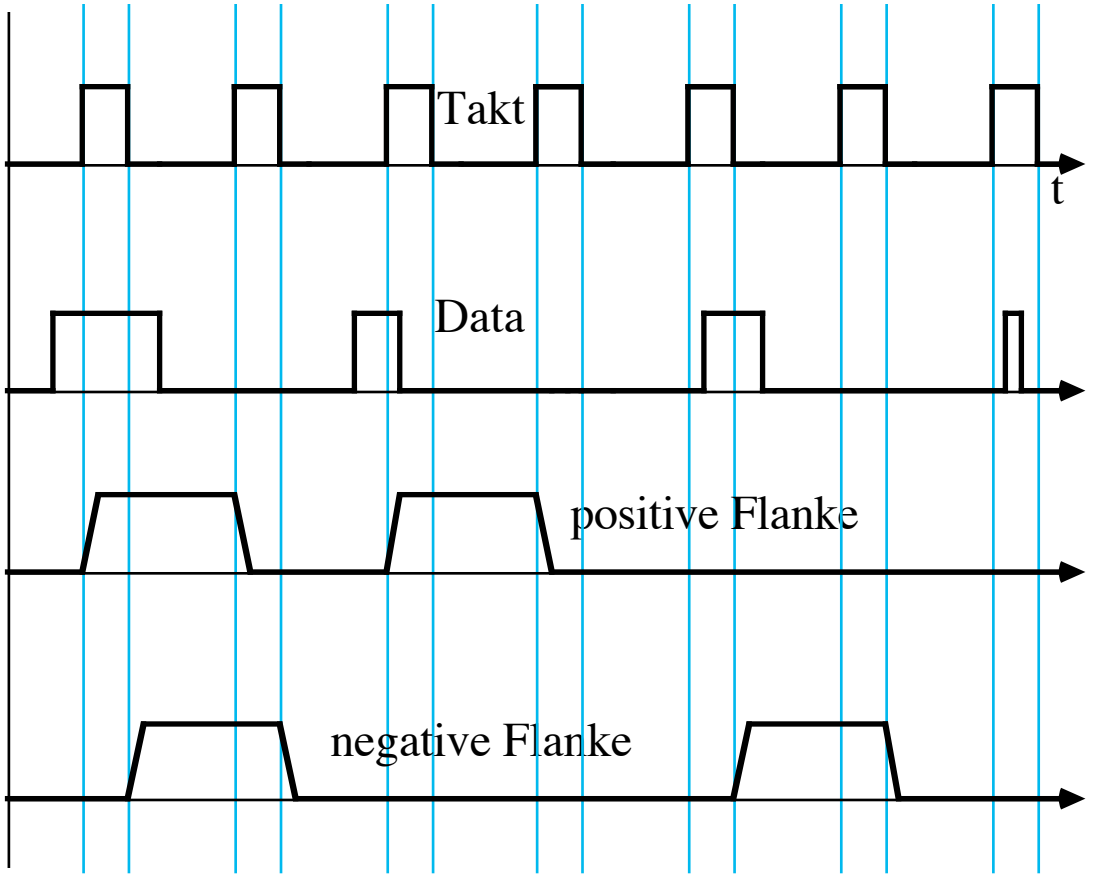
- Taktsteuerung
  - Eingangswerte stabilisieren lassen
  - dann Übernahmepuls
  - Eingangswerte werden nur zum Taktzeitpunkt berücksichtigt
- Flankengesteuertes Flipflop (abfallend):



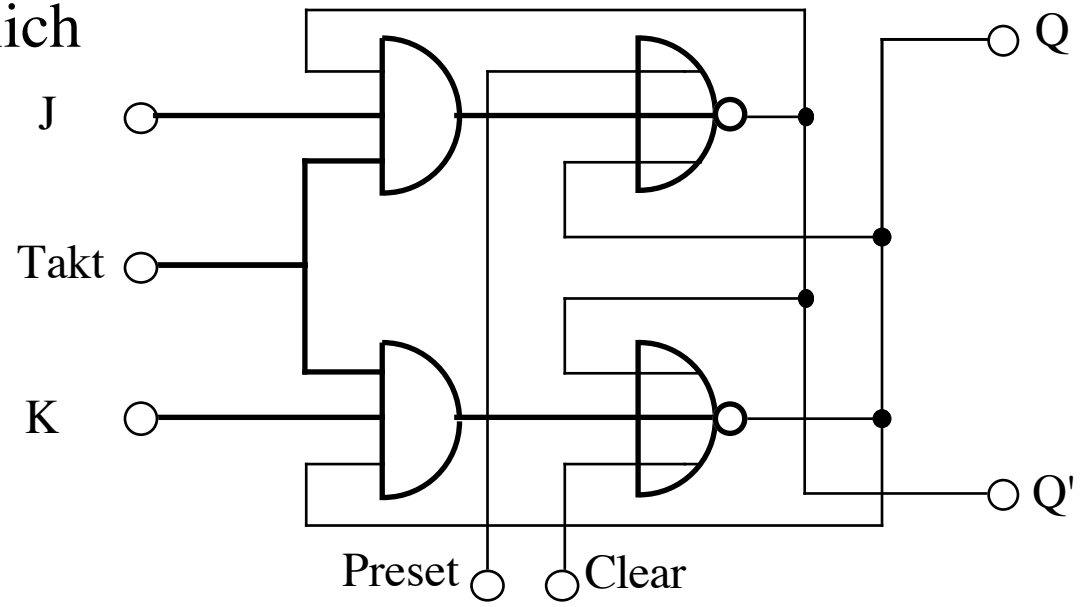
- Flankengesteuertes Flipflop (ansteigend):



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



- JK-Flip-Flop
  - Eingänge vom Ausgang her verriegeln
  - Preset- & Clear-Eingänge möglich



- Ablauf

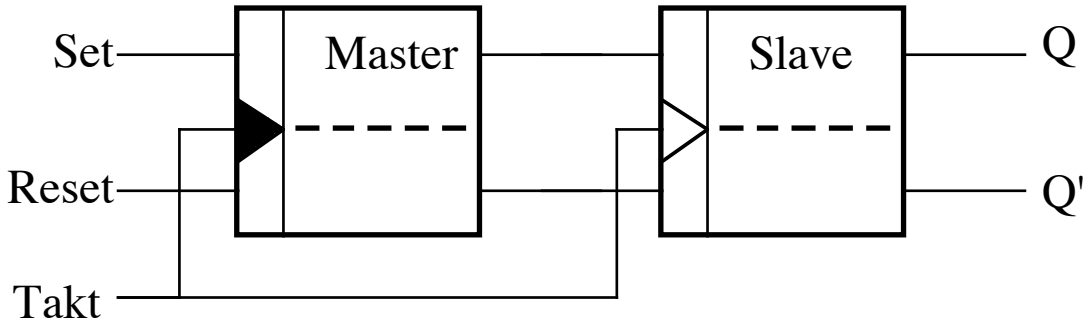
a) J=x, K=x, T=0, Q=1, Q'=0
   {Anfangskonfiguration}

b) J=1, K=0, T=1 -> Q=1, Q'=0 {Eingang J unwirksam}.

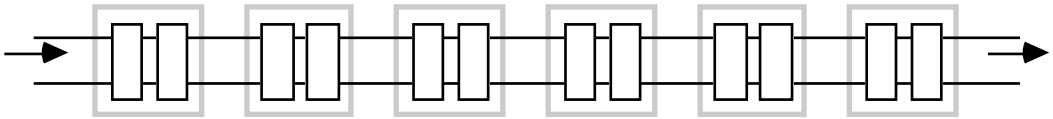
c) J=0, K=0, T=0 -> Q=1, Q'=0 {Takt aus, J&K unwirksam }

d) J=0, K=1, T=1 -> Q=0, Q'=1 {Umschalten nur mit Takt }
   J=1, K=1, T=1 -> Q=1, Q'=0

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

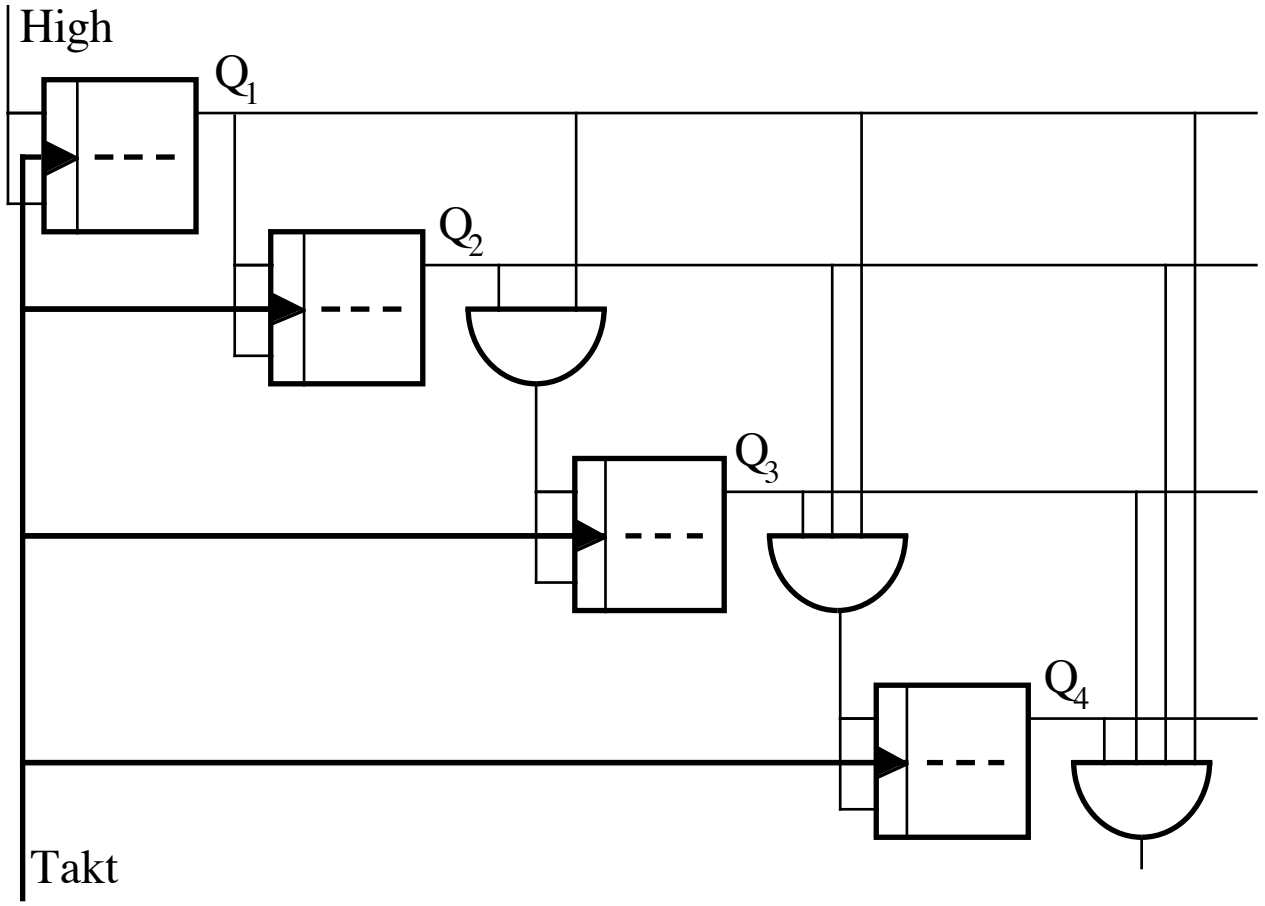


- Für Schaltungen mit heiklem Timing
  - Registertransfers
  - Pufferregistern mit MS-FFs
- MS-Schieberegister

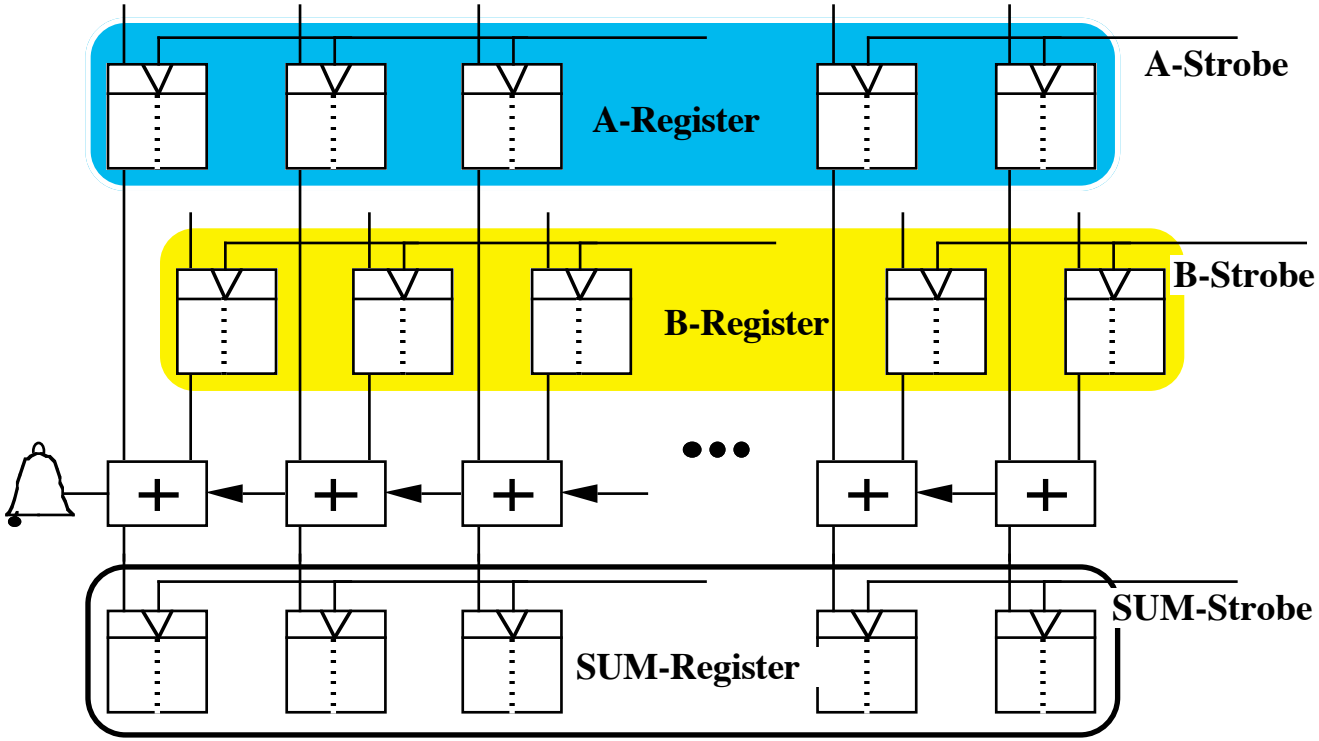


  - als digitale Verzögerungsleitung
  - für digitale Filter
  - als Rechenoperation

- Zähler
  - Jede Stufe erhält Takt
  - schaltet in Abhängigkeit vom Zustand aller vorherigen Stufen



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

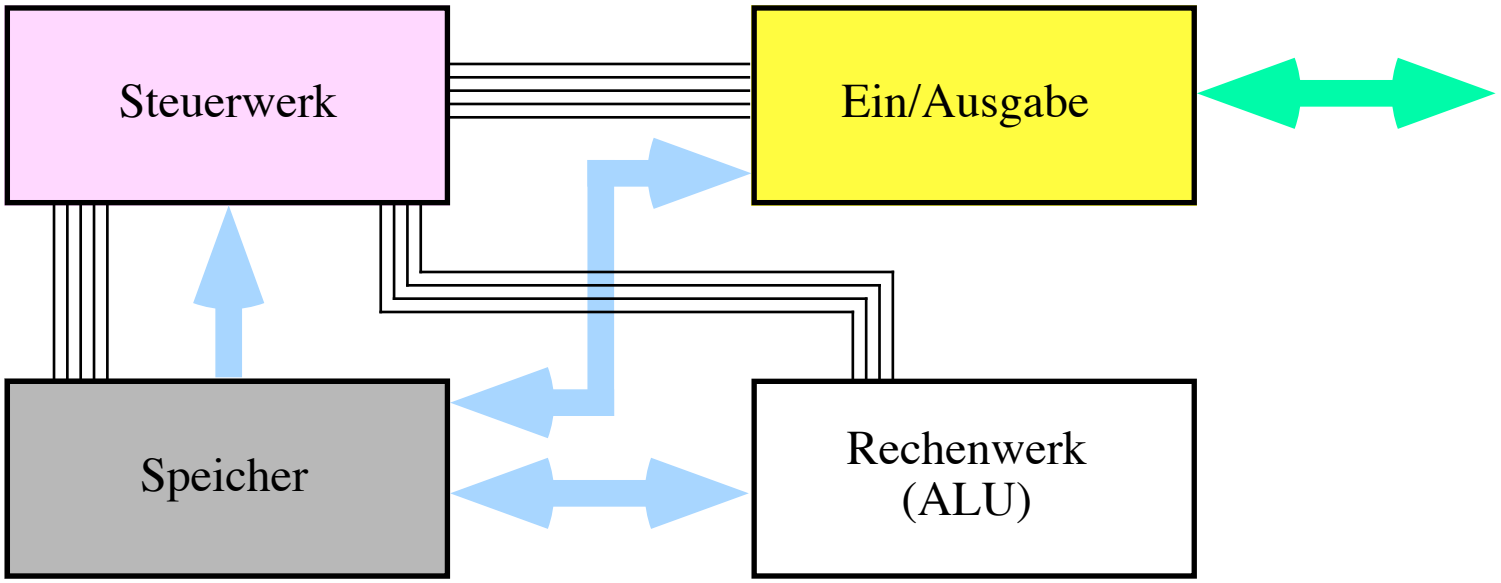


- Ablauf
  - A-Strobe zum Füllen des Registers A
  - B-Strobe zum Füllen des Registers B
  - Addition bzw. Übertrag abwarten,
  - Summe abholen mit SUM-Strobe
  - Überlauf?

3. Rechnerarchitektur

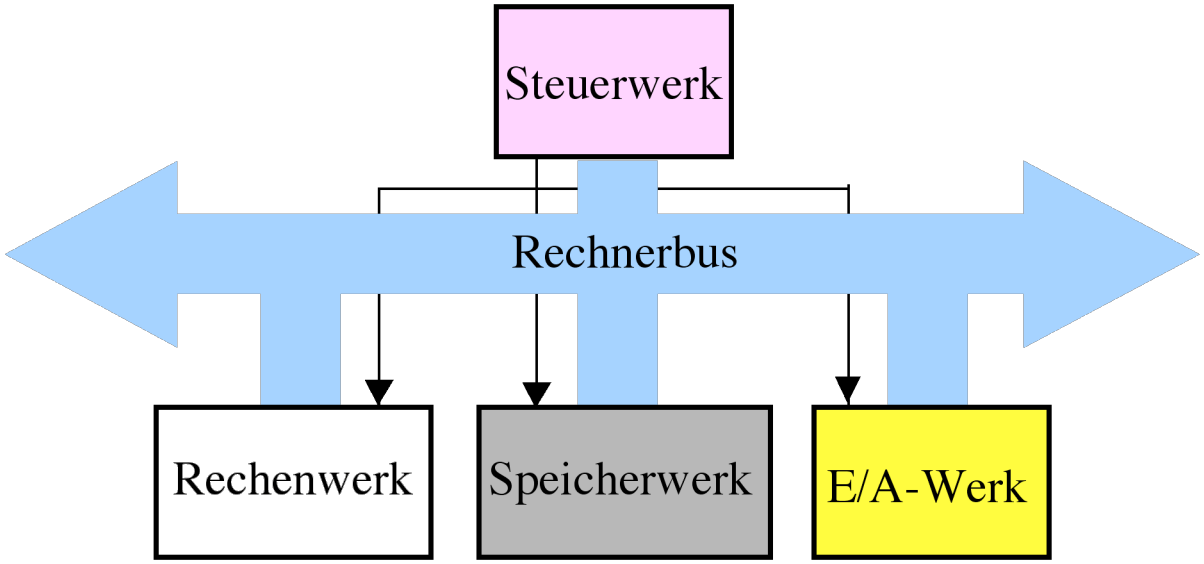
3.1 Funktionseinheiten: Speicher, Prozessor, Bus, ...

- klassischer Rechner nach J. v. Neumann
  - Rechnerwerk für arithmetische und logische Verknüpfungen
  - Universalspeicher für Daten und Programme

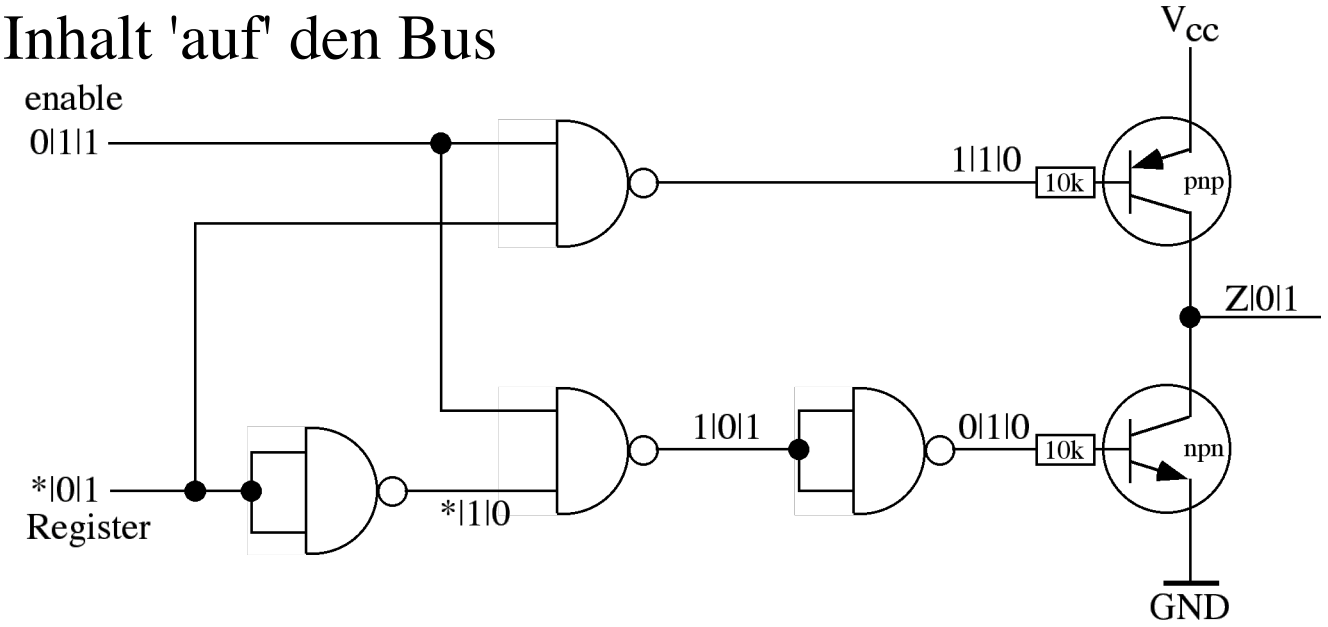


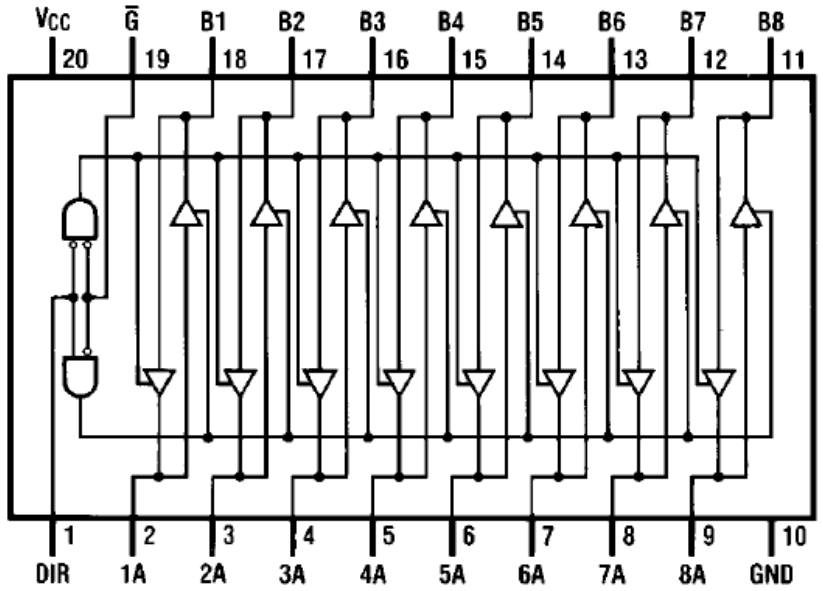
- Steuerwerk
  - sequentieller Ablauf des Programmes
  - Interpretation der Instruktionen => Steuerbefehle

- Busorientierter Rechner
  - universell verbunden
  - Schwerpunkt Transport und Verteilung
  - weniger Verbindungen

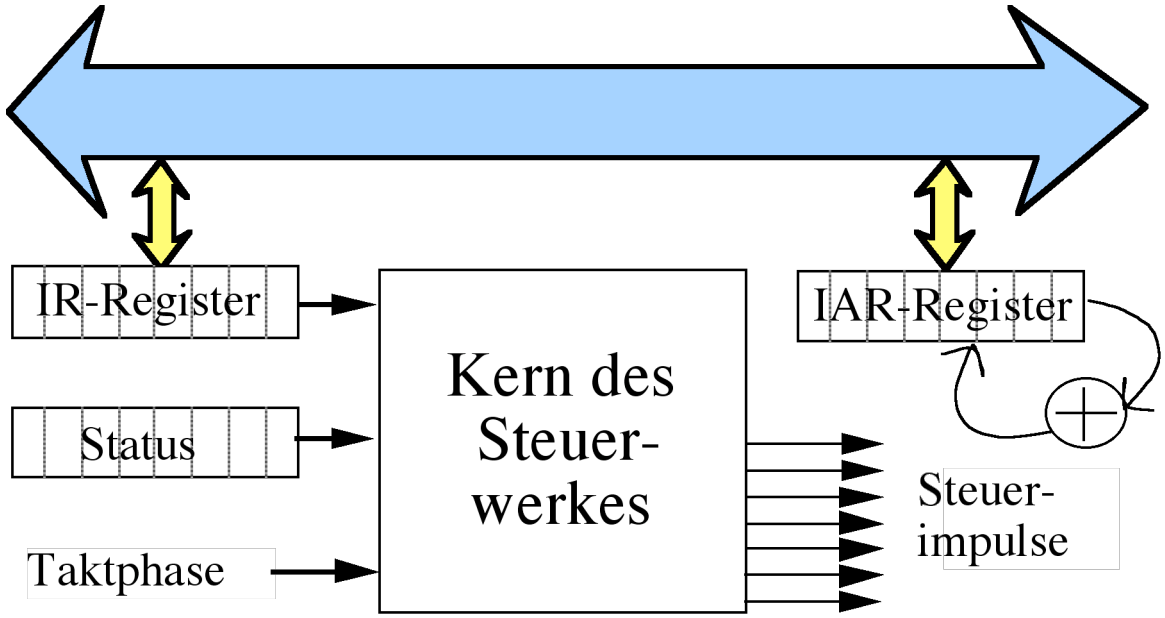


- Bustreiber
  - Register liegt nicht 'direkt' am Bus
  - Steuersignal legt Register-Inhalt 'auf' den Bus
- Tri-state-Schaltung
  - 3 Zustände
  - hochohmig
  - 0, 1
- 74xx251
  - 8 Bit
  - Transceiver
  - bidirektional
  - Richtungseingang
  - Strobe



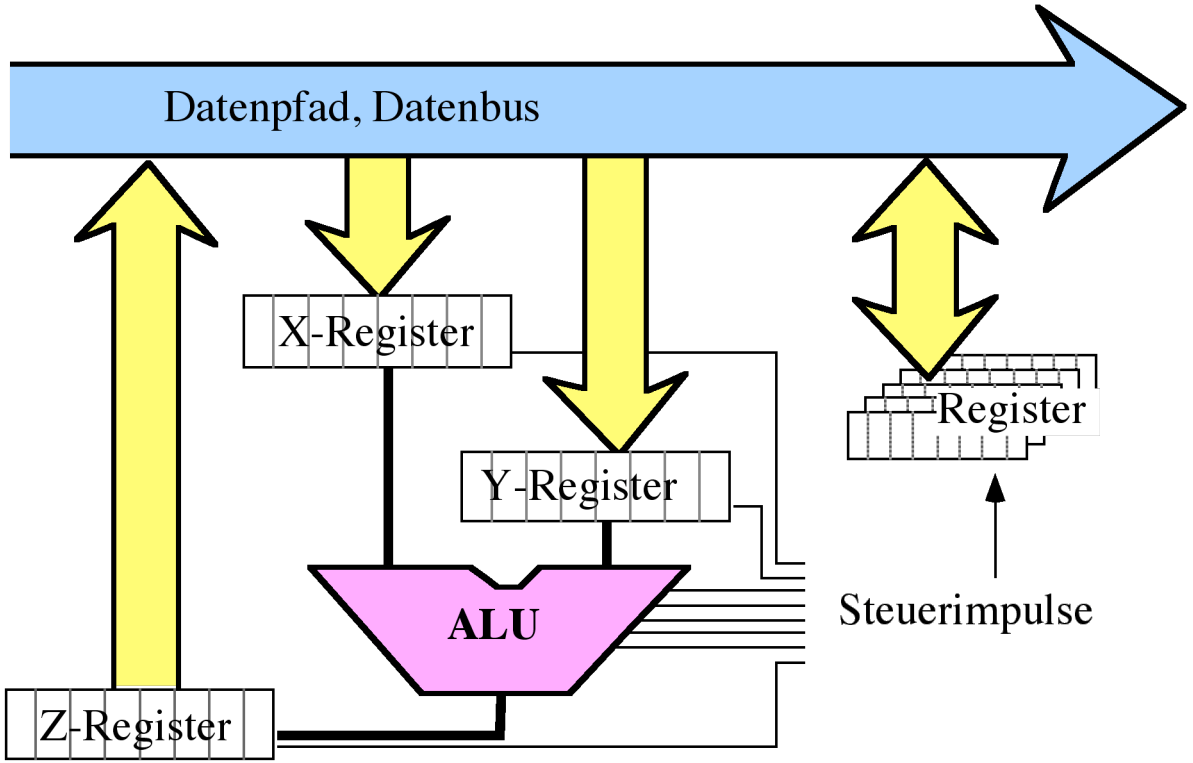


- Steuerwerk erzeugt Signale, die
  - Datentransfer auslösen
  - ALU-Operation auswählen
  - Speicheroperation auslöst



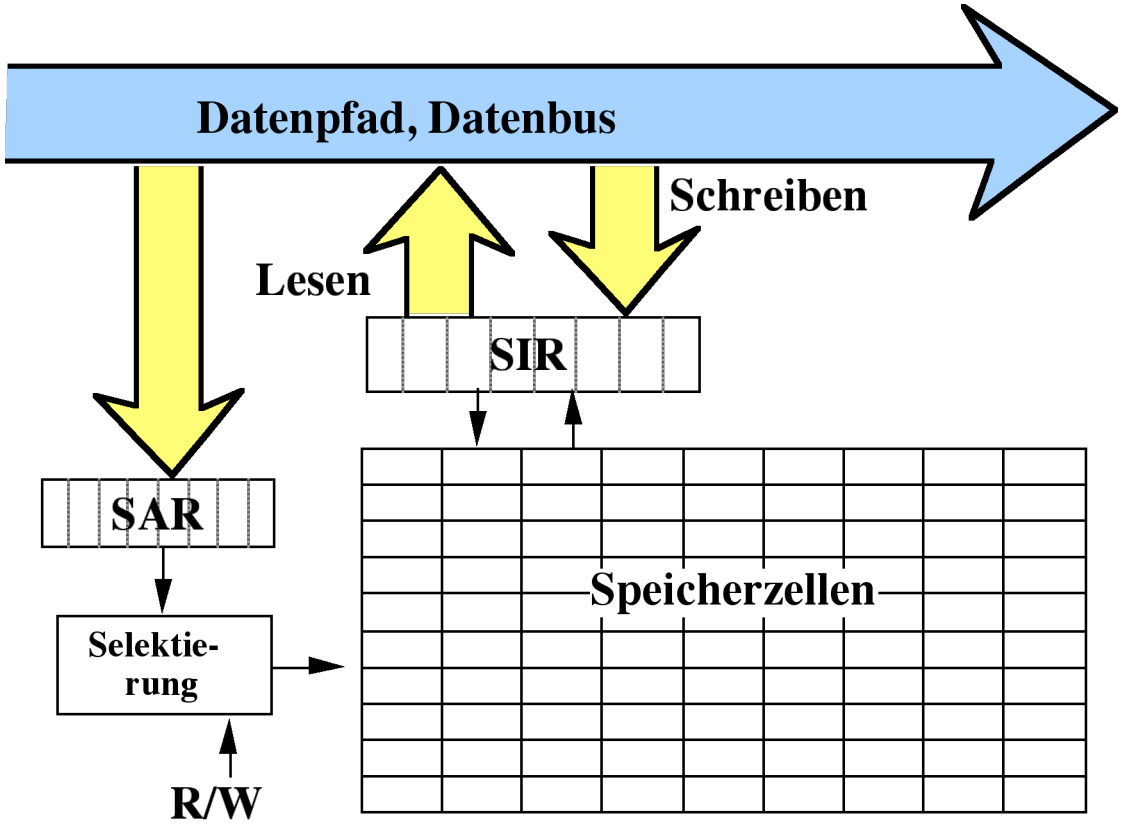
- Befehl wird in Sequenz von Kernzuständen umgesetzt
  - Kernzustand bestimmt Steuersignale
- Register
  - Instruktionsregister (IR)
  - Instruktionsadressregister (IAR) bzw. "Program Counter" (PC)
  - eventuell eigener Addierer für IAR

- 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

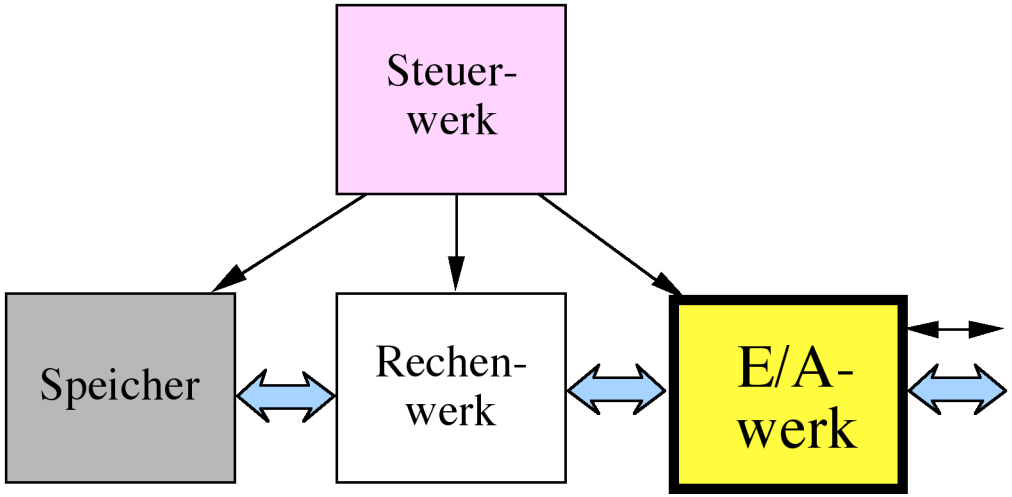
- Random Access Memory (RAM)

  - Halbleiterspeicher halten Inhalt nur wenn sie "unter Strom" stehen
  - dynamische Speicher (DRAM) müssen periodisch aufgefrischt werden

- ROM: Festwertspeicher

- Speicherhierarchie aus Kostengründen
- Register
  - 4-256 Wörter,< 1 Nanosekunde
  - kein separater Zyklus für die Adresse.
- Cache, Prozessorpufferspeicher
  - 8 KWörter bis 1024 KBytes, < 10 Nanosekunden
  - häufig gebrauchte Daten und Codeteile
  - für Programmierer unsichtbar
- Hauptspeicher
  - 64 - 16000 Megabytes, ~ 7 - 10 Nanosekunden
  - evtl. inklusive Adressübersetzung
  - separater Zyklus für die Speicheradresse
- Hintergrundspeicher
  - Festplatte, Netz,
  - 80 Gbytes -2000 Gbytes, ~ 10 Millisekunden
  - Zugriff über Betriebssystem
  - blockweise übertragen

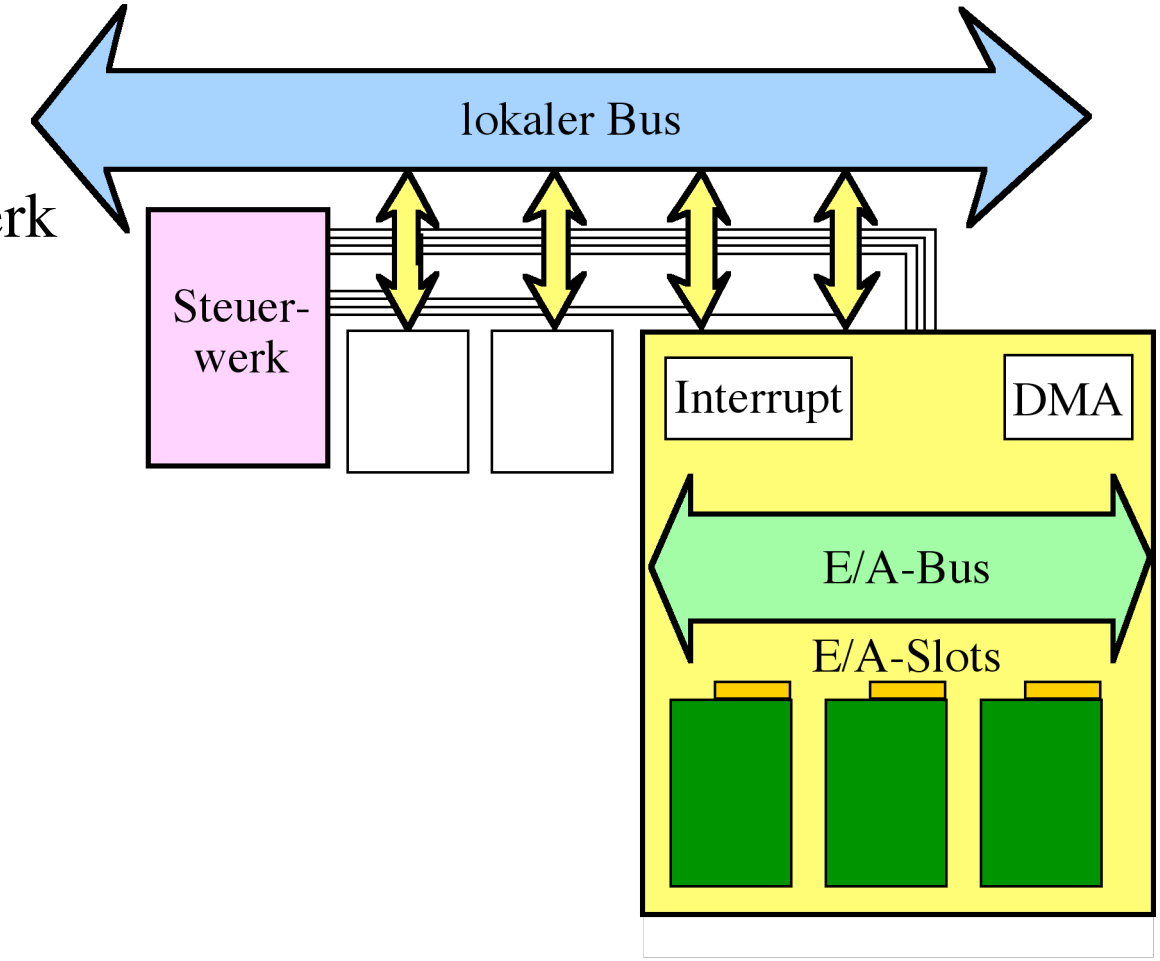
- Ein-/Ausgabewerk
  - kontrolliert durch Steuerwerk
  - Transport via Rechenwerk in den Speicher
  - Bedienungsleitungen zum Peripheriegerät
  - Keine Parallelarbeit von Rechenwerk & E/A
  - Missbrauch der Rechenregister
  - Wartezyklen der CPU auf Peripheriegeräte



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

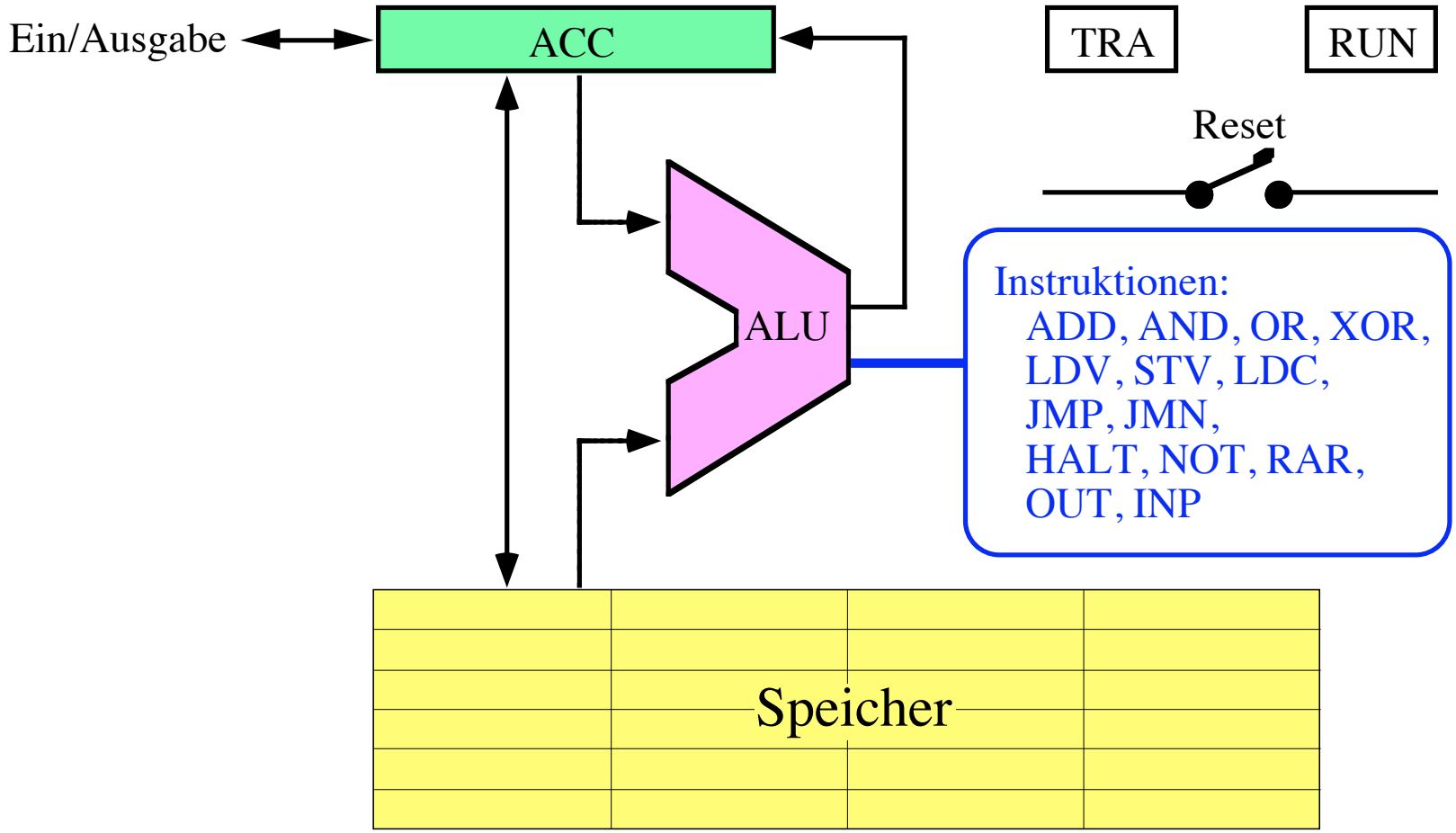
- Moderne Rechner
  - besonderer, standardisierter E/A-Bus
  - getrennt vom Prozessorbus
- autonomere E/A-Schnittstelle
  - externer Bus,
  - Abläufe parallel zum Rechenwerk
  - Interrupt-Funktion
  - DMA-Kanäle

  (direct memory access)
  - Display-Kontroller
  - Adapterplatinen
  - VLSI-Chips
- Grafik evtl. Extrabus
  - AGP, PCIe



3.2 Registertransfer

3.2.1 Beispielarchitektur Mima



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

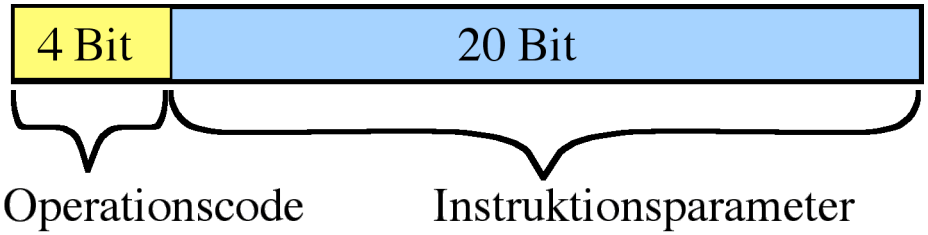
- Einfacher Instruktionssatz
- Befehle zur Datenmanipulation

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 ← ACC + Speicher[a]
AND	a	ACC ← ACC ∧ Speicher[a]
OR	a	ACC ← ACC ∨ Speicher[a]
XOR	a	ACC ← ACC ⊕ Speicher[a]
EQL	a	ACC<>Speicher[a]: ACC← 0
		ACC=Speicher[a]: ACC← 11..1
LDV	a	ACC ← Speicher[a]
STV	a	Speicher[a] ← ACC
LDC	c	ACC ← c

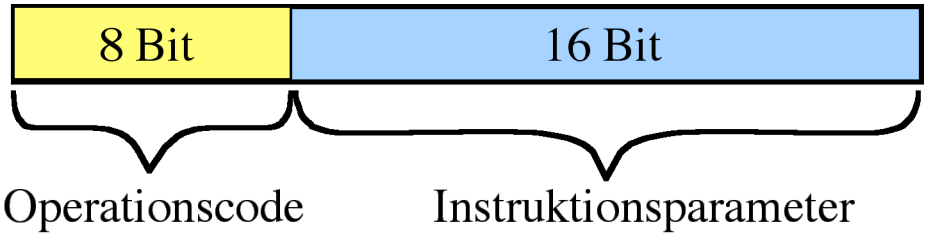
- Kontrollfluß

JMP	p	Nächste Instruktion in Speicher[p]
JMN	p	Sprung nach p falls ACC negativ
HALT		RUN ← 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 reserviert
- Erweitertes Format
  - mehr als 16 Instruktionen möglich,
  - 16 Bit Parameter & nicht immer benützt

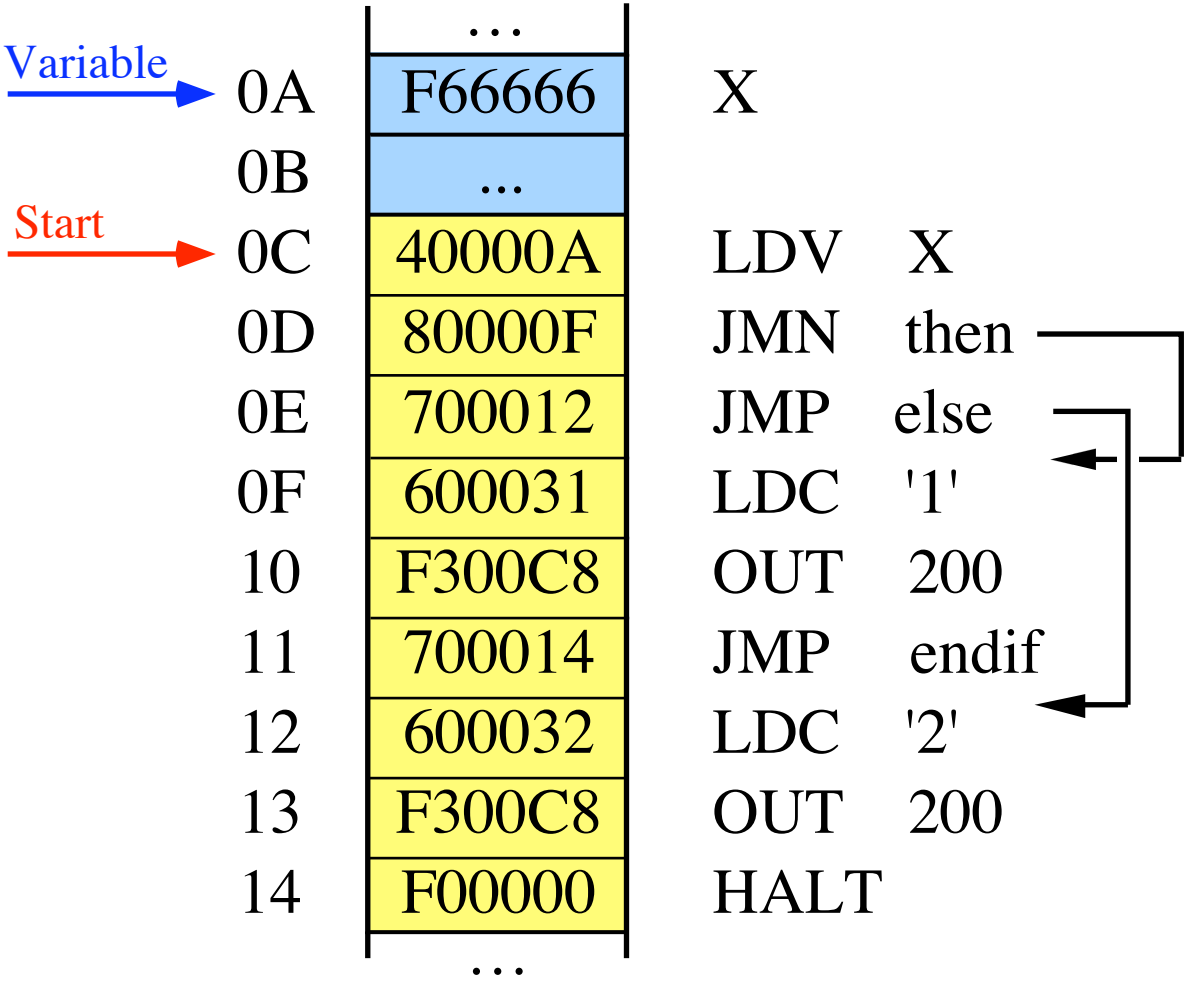


  - F0, F1, F2 HALT, NOT, RAR
  - F3, F4,… OUT, INP, …

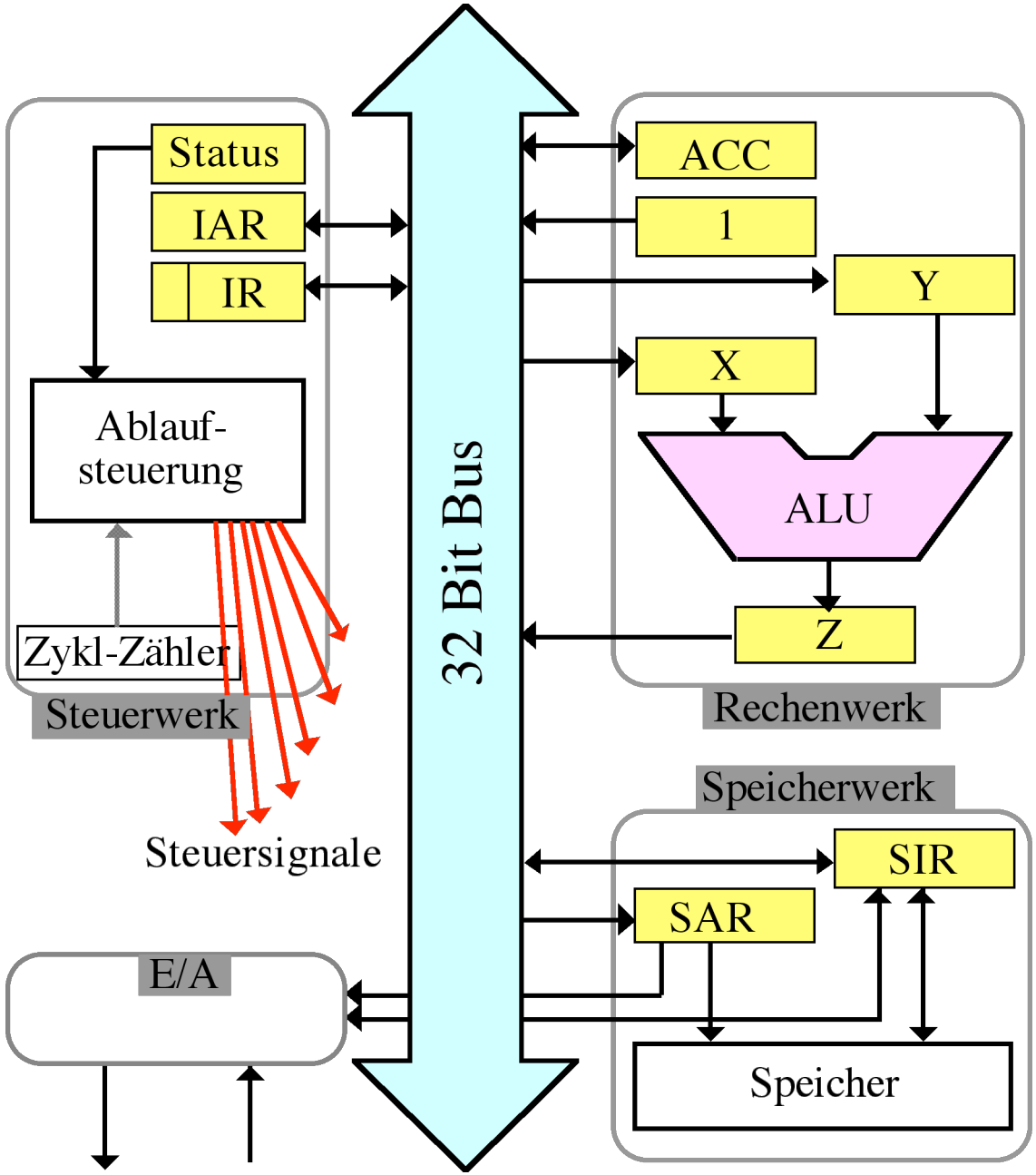
- Assemblerprogrammieren im Schnelldurchgang

```
if (X < 0)     printf("1")
               else printf("2");
```

- Hexadezimaler MiMa-Speicherauszug



- Speicher
  - statisch
  - 24 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



3.2.2 Ablauf der Instruktionen

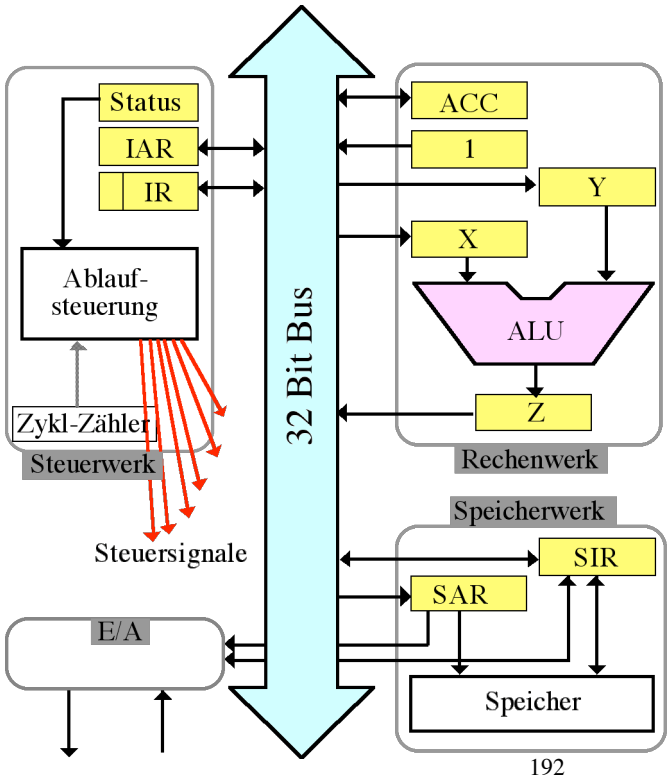
- Zweiteilung der Befehle
  - Fetch-Zyklus holt Befehl
  - Execute Zyklus führt Befehl aus
- Unterzyklen im Steuerwerk
  - Mikrozyklus, "Minor Cycle", Taktphase
  - andere Steuerimpulse in jedem Unterzyklus
  - Registertransferebene
- Mima-Instruktionen
  - 12 Unterzyklen
  - 5 Unterzyklen Fetch
  - 7 Unterzyklen Execute
- http://mimasimulator.appspot.com/

- 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
```

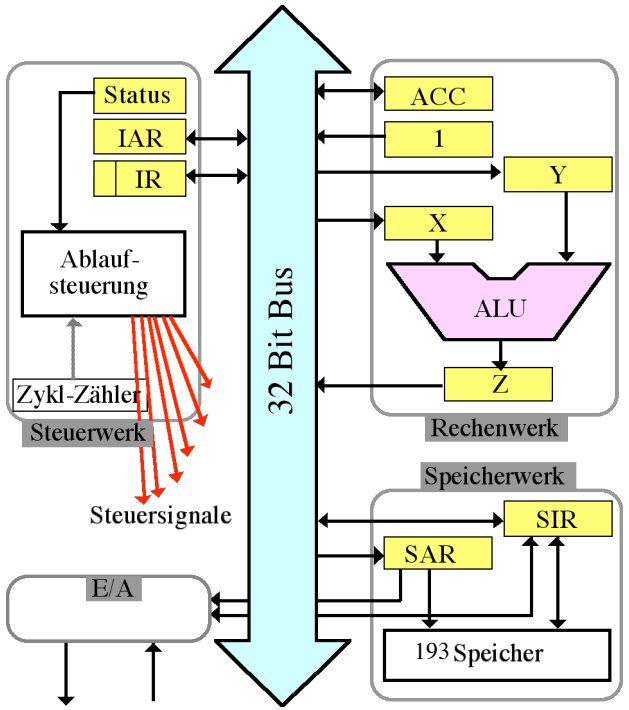


- ADDa ; addieren von Inhalt der Speicherzelle a zum ACC

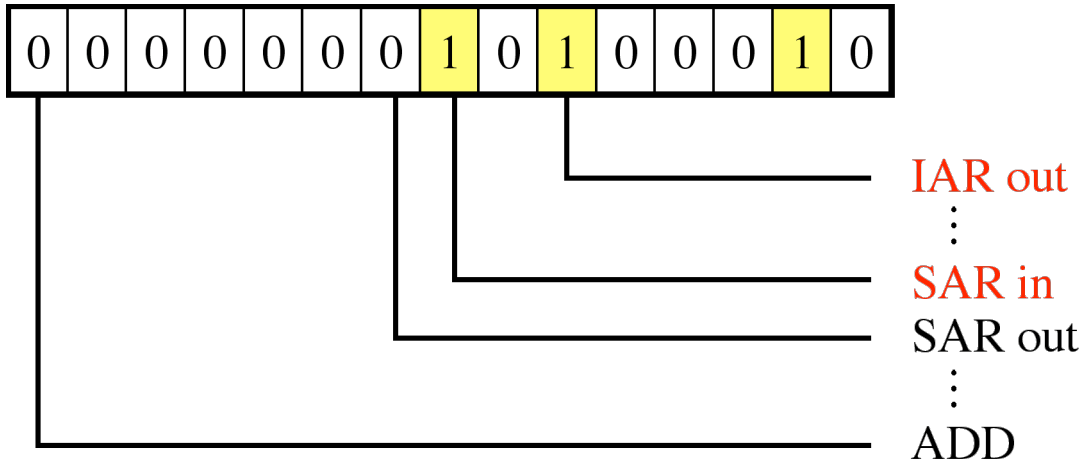
```
1.-5.     Fetch-Zyklus wie oben
    ;Fetch-Ende, jetzt Execute-Zyklus
6.        IR  -> SAR,  Leseimpuls an Speicher
7.        ACC  -> X,   Warten auf Speicher
8., 9.    Warten auf Speicher
10.       SIR -> Y,  ALU auf Addition schalten
11.       warten auf ALU
12.       Z  -> ACC
    ;Execute-Ende, nächste Instruktion
```

- JMN p - Jump if Negative
  - Bedingter Sprung zur Instruktion im Speicherwort[p]
  - negativ bedeutet, das oberste Bit im Akkumulator ist 1

```
1.-5.     Fetch-Zyklus wie oben
    ;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
```



- Befehle liegen im Speicher
  - werden geholt wie Daten
  - kommen in Befehlsregister
  - Befehlszähler
- Befehl ⊗ Status => 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



- 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

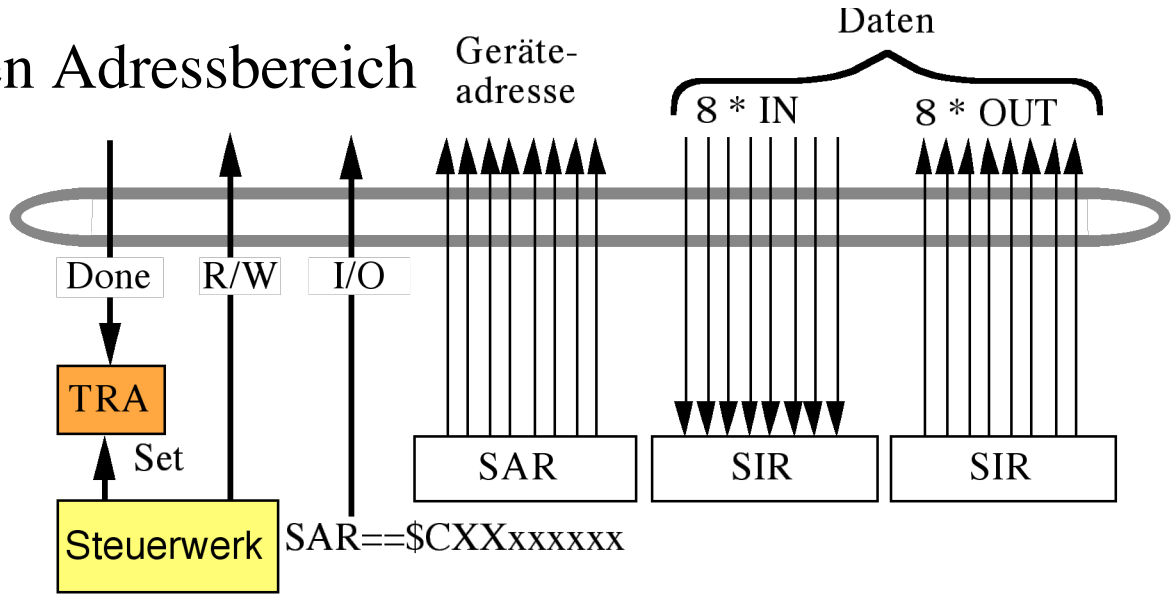
- Steuerwerk kann realisiert werden:
  - als 20 Schaltfunktionen mit 15 Variablen
  - als ROM mit 2^{15} Wörtern à 20 Bit

- Start der Mima
  - Drücken der Reset-Taste

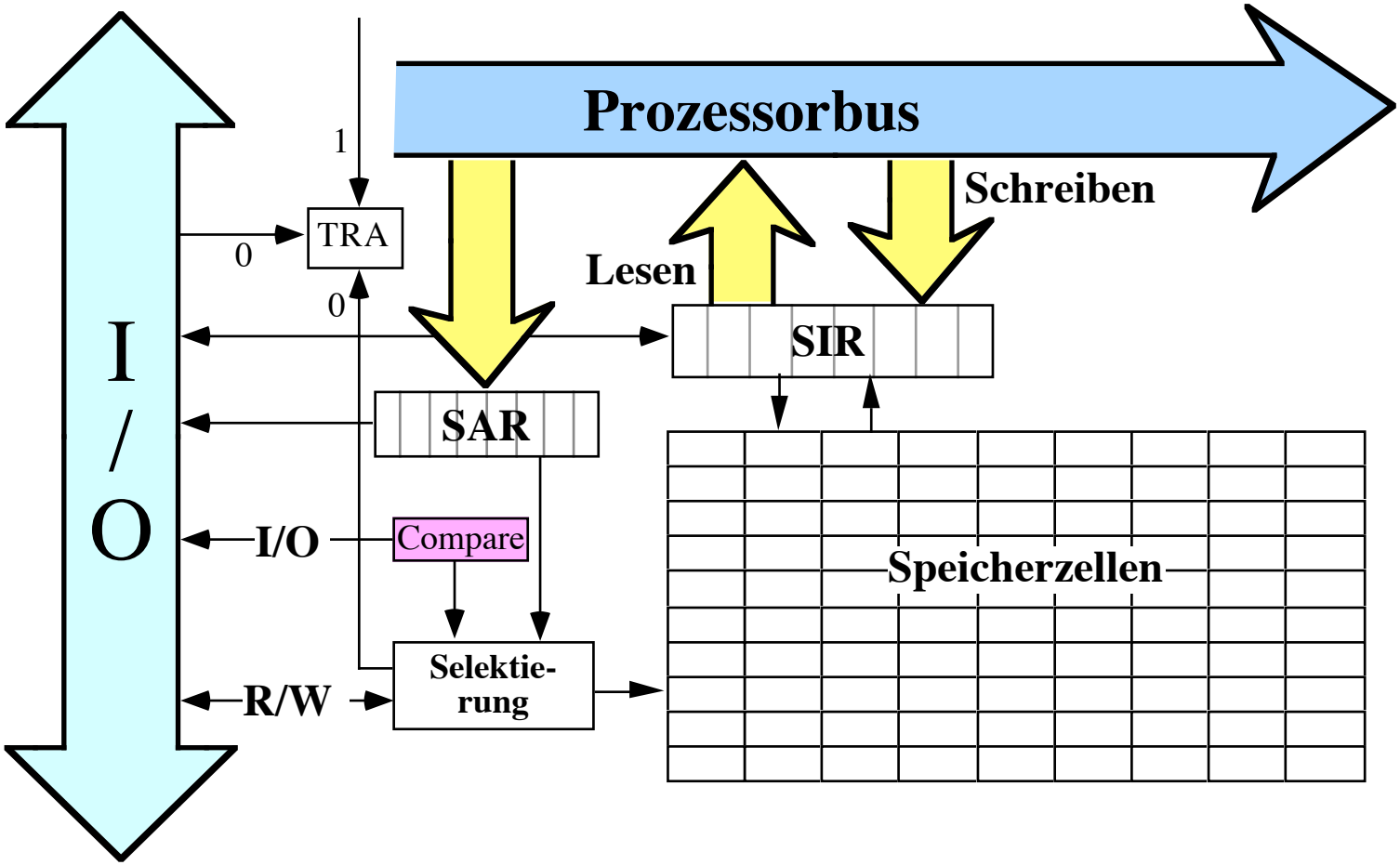
```
0 -> IAR
0 -> TRA
1 -> RUN
```

- 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
  - Adr == $CXXxxxxxx => E/A
  - Adr <> $CXXxxxxxx => Speicher
- TRA-Bit
  - vom Steuerwerk gesetzt
  - von Speichercontroller oder Peripherie zurückgesetzt
  - Steuerwerk wartet auf TRA=0
- Geräte beobachten die Geräteadresse im EA-Werk
- Als Bus ausgeführte E/A-Schnittstelle



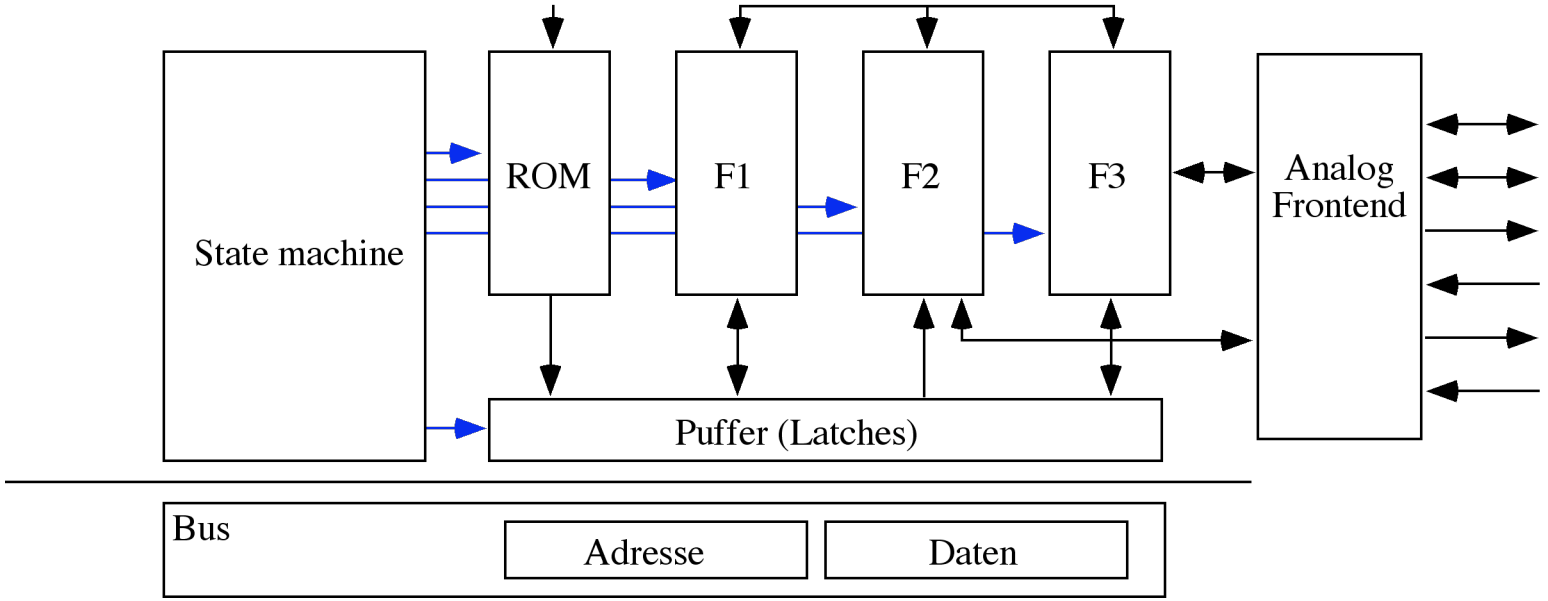
- Speicher mit Memory-mapped I/O
  - TRA Bit wird vom Prozessor gesetzt
  - von der Selektionsschaltung zurückgesetzt
  - oder vom Gerätebus zurückgesetzt



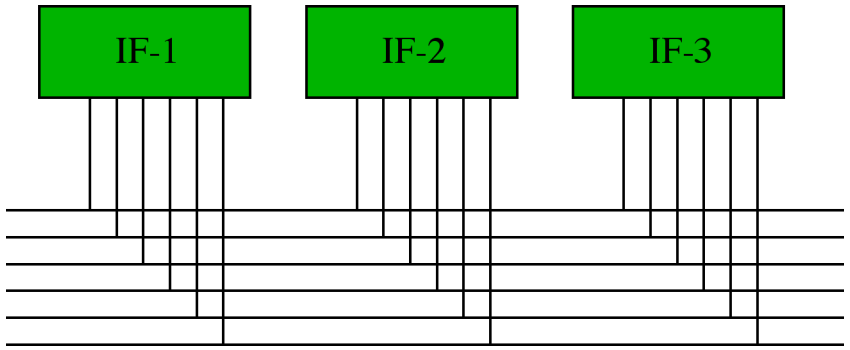
SAR = Speicheradressregister, SIR = Speicherinhaltsregister

3.3 Adressierung, Timing, Hardware/Software-Interface

- Funktionseinheiten (Chips) an Bus anschliessen
  - Prozessorbus ('memory-mapped') oder I/O Bus
  - Adressen und Daten
  - Steuerleitungen: Read/Write, Strobes, …
- Businterface
  - Busphasen, Adressdekodierung (state machine)
  - Puffer
  - Register
  - Identifizierung



- Bus
  - Adressbus und Datenbus oft 'gemultiplext'
  - Steuerleitungen: Adress-Strobe, DataStrobe
  - Takt
  - Card-Select



- Puffer
  - entkoppeln Bustakt und Adapter-Takt
  - evtl. bidirektional
  - Bustreiber: active-high, active-low, offen (tri-state)
- State-Machine
  - Chipselect (C/S) erzeugen
  - Adress-Dekodierung für Datenpuffer
  - Adress-Dekodierung für Chip
  - niedrige Adressbits zur Registerauswahl im Chip
- Chip Select: CS
  - aus Adresse abgeleitet
  - z.B. mit kombinatorischer Schaltung

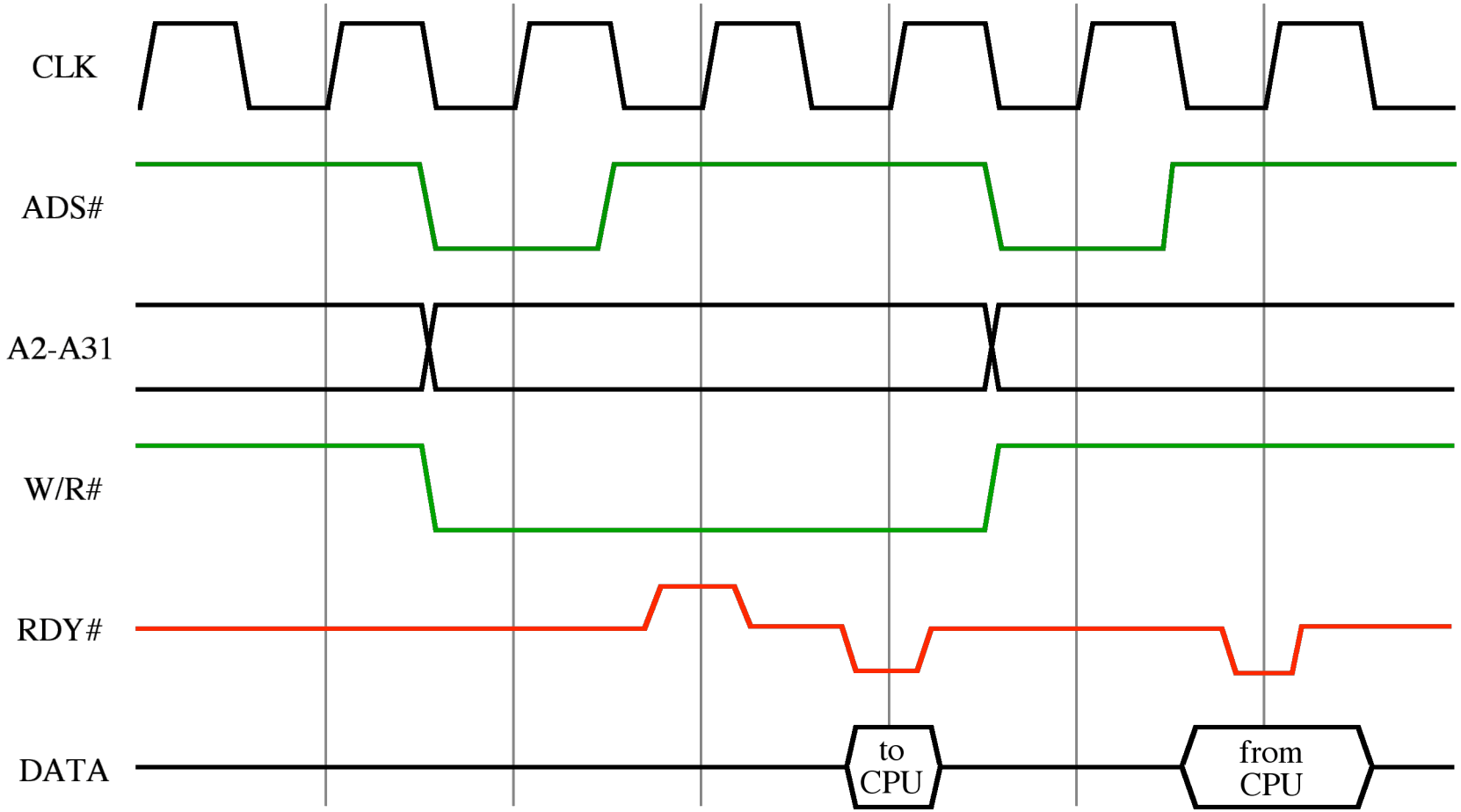
```
if (((unsigned) theAddress>>24) == 0x0FC)
```

- Programmable Logic
  - z.B. GAL 16V8, 22V10, …
  - Eingabe-Pins und Ausgabe-Pins
  - manche mit Flip-Flops
  - Und, Oder, Not
  - Enwicklungssysteme (Abel, …)

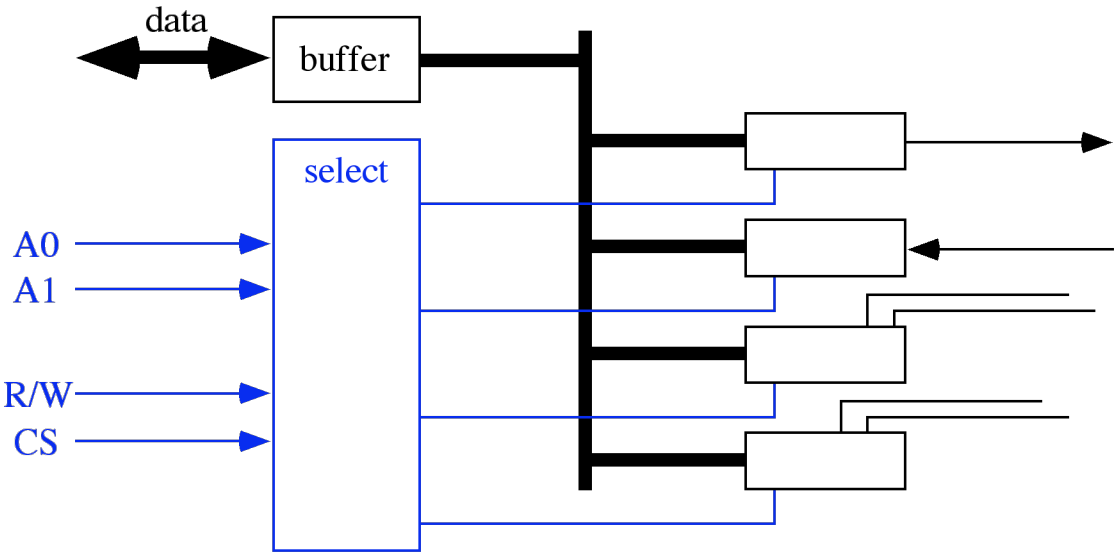
```
"/**  Inputs  **/
A24       Pin 1          ;      "/* address bus                    */
A25       Pin 2          ;      "/*                                 */
A26       Pin 3          ;      "/*                                 */
A27       Pin 4          ;      "/*                                 */
A28       Pin 5          ;      "/*                                 */
A29       Pin 6          ;      "/*                                 */
A30       Pin 7          ;      "/*                                 */
A31       Pin 8          ;      "/*                                 */
"/**  Outputs  **/
MYCHIP    Pin 13         ;

Equations;
MYCHIP = !A24 & !A25 & A26 & A27 & A28 & A29 & A30 & A31
                  "/* (Adresse = FC XX XX XX) */
```

- Timing-Diagramm
  - Buszyklen von Intel 486
  - mit eingefügten Wait-State beim Read
  - # für invertierte Logik



- Instruktion zum Ansprechen von HW
  - Abstraktion: Register
  - LDV/STV mit Adressen
  - Adresse wählt Chip und Register (Chip-Select siehe oben)
  - evtl. A0 und A1 nicht auf dem Bus -> Registernummer*4



- Register für Kommandos
  - Bitketten
  - Bitkette lesen: Statusinformation
  - Bitkette schreiben: Kommando
  - Registerauswahl aus Subadresse im Chip

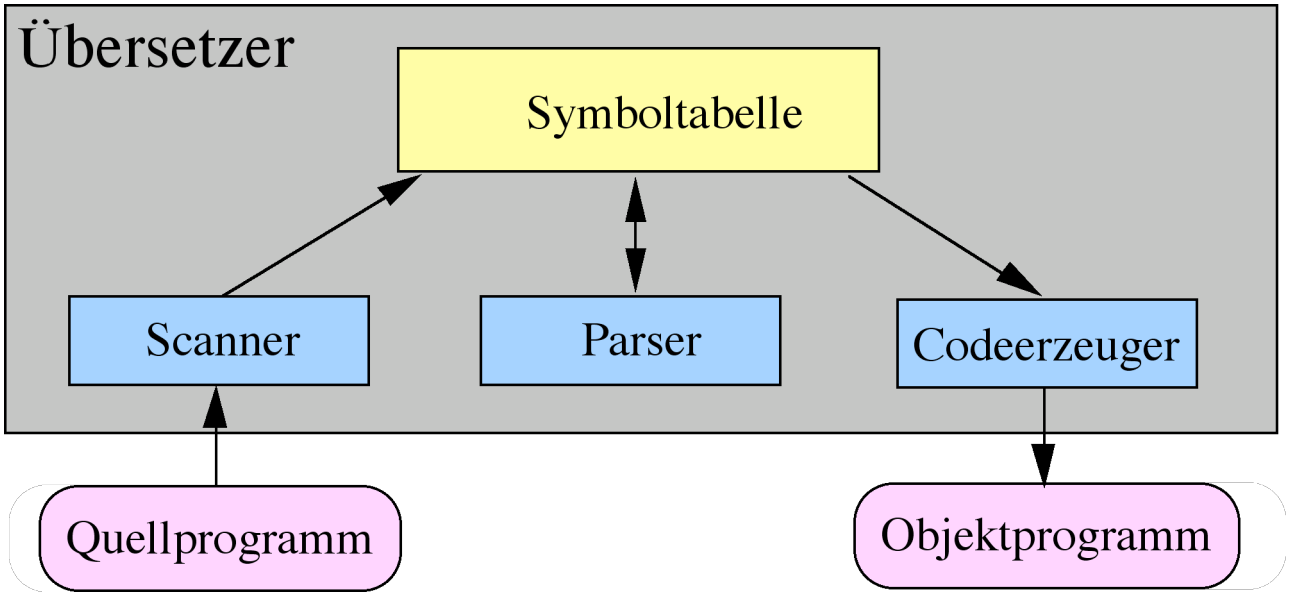
- Datenübergabe
  - Byte bzw. Wort
  - evtl. mehrere Byte lesen

```
int i;
volatile unsigned char *status = 0xFFA00;
volatile unsigned char *datareg = 0xFFA04;
unsigned char packet[1500]
…
i = 0;
while ((*status & $04) && i < 1500)
   packet[i++]= *chipdatareg;
```

- Kontrollblock
  - Kommando/Status und Daten als Record im Speicher
  - verkettete Liste, Ringpuffer , etc.

3.4 Compiler

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



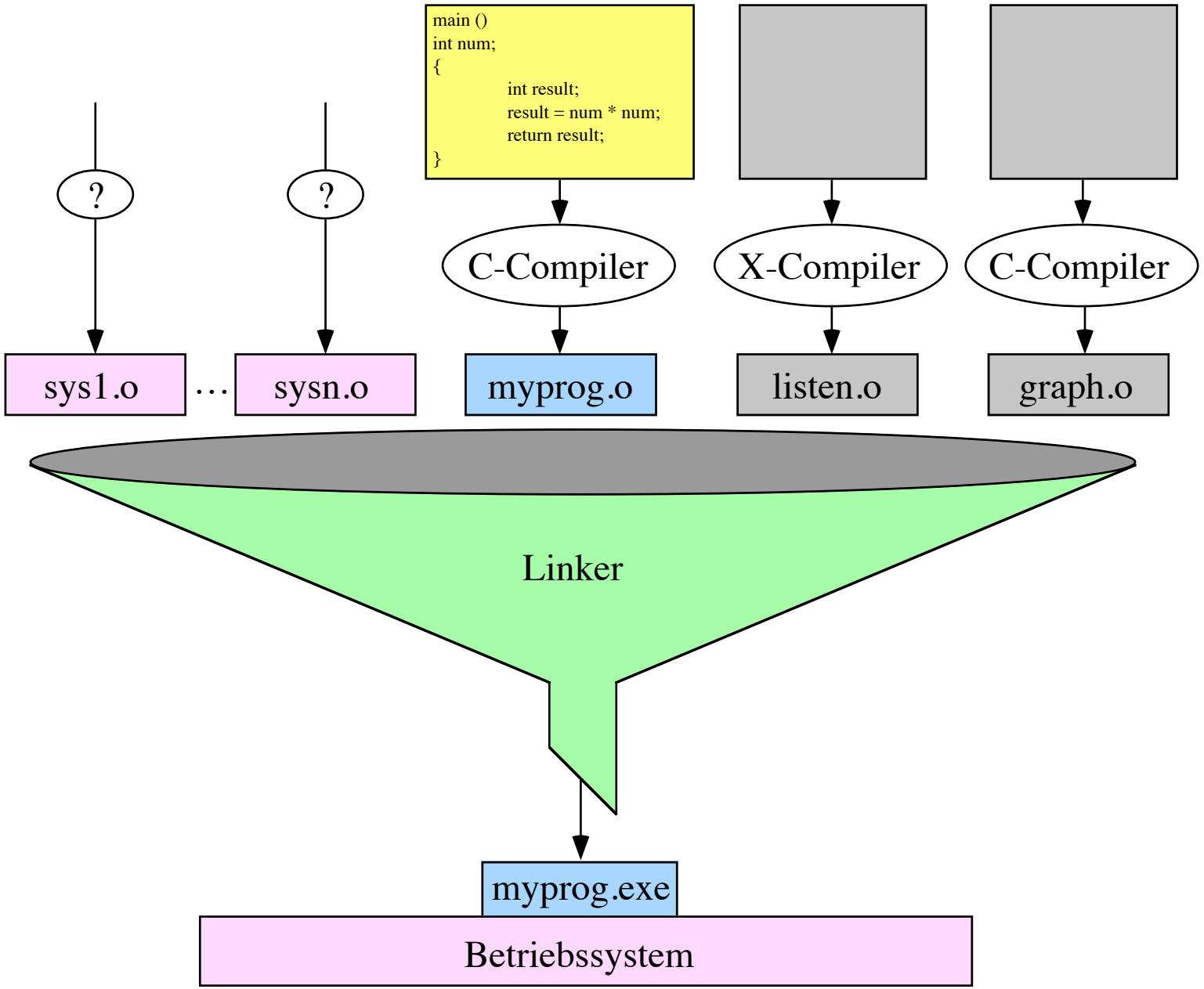
- Einlesen des Programmes (Scanner)
  - findet Symbole
  - Identifier, Konstanten, …
- Syntaktische Analyse
  - zulässige Symbole werden verarbeitet ("Parsing")
  - für unzulässige Symbole Fehlermeldungen erzeugen
  - über "Look-Ahead" entschieden, welcher Pfad gewählt werden soll
  - bei schwierigen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

- Beispiel: Ausdruck übersetzen

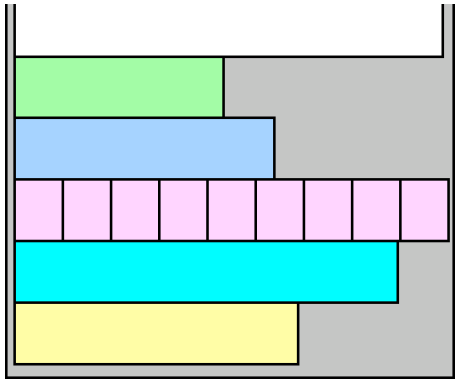
a = b-(c-((d*e)-g/h))

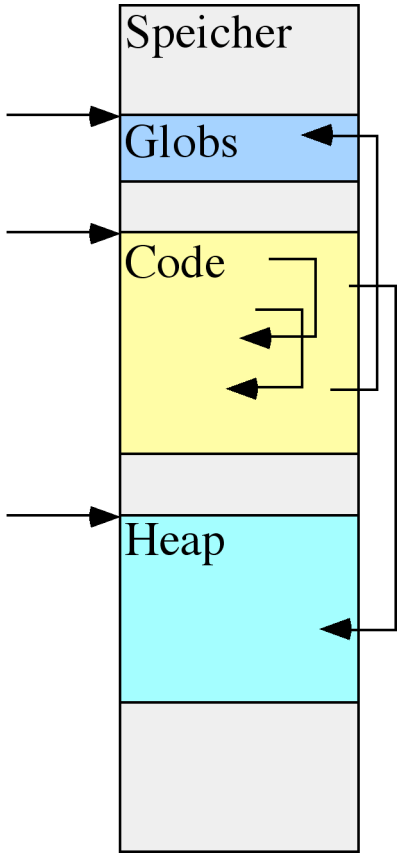
```
LDV    g
DIV    h          ; g/h
STV    hilf       ; optimiert wird nicht
LDV    d
MUL    e          ; (d*e)
SUB    hilf       ; -
STV    hilf
LDV    c
SUB    hilf
STV    hilf
LDV    b
SUB    hilf
STV       a          ;   a:= ...
```

- Kompletter Programmierablauf

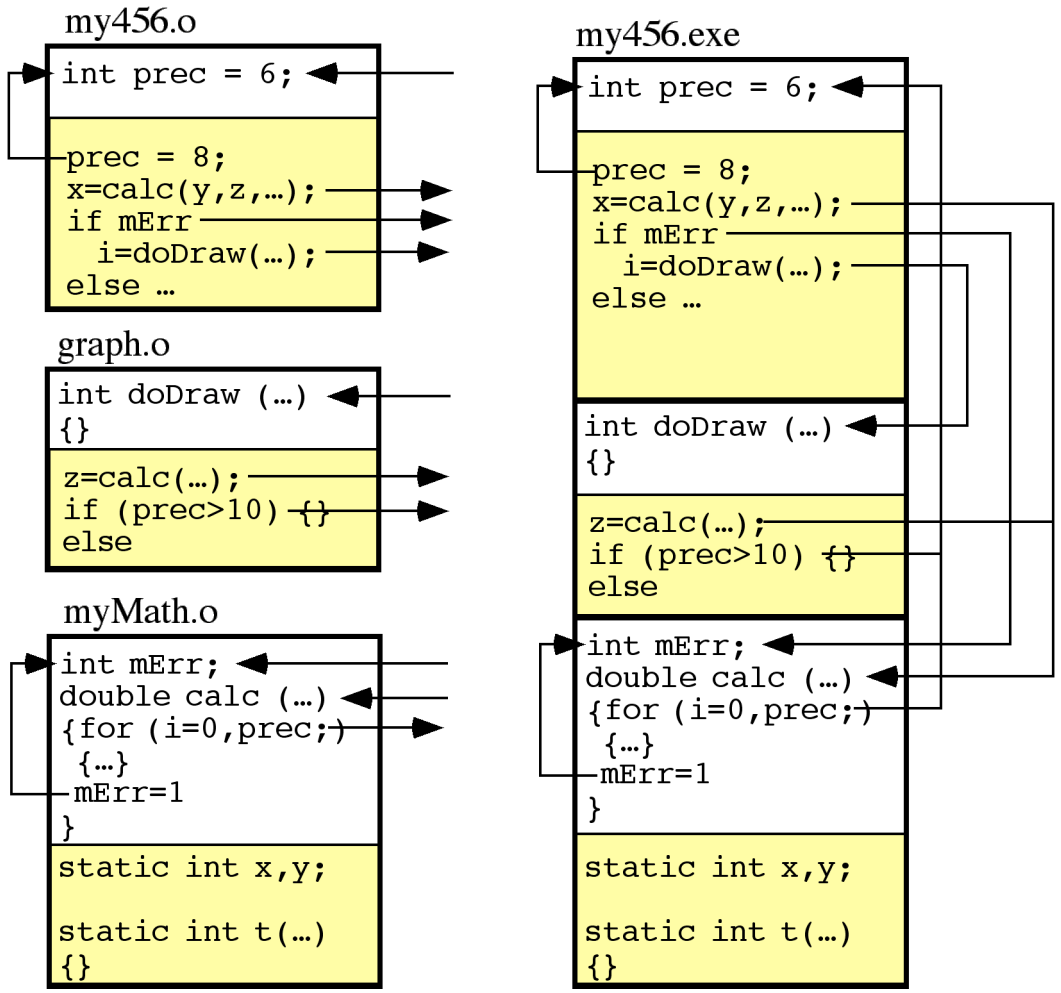


- Daten im Speicher
  - globale Variablen
  - lokale Variablen
  - Datenstrukturen
  - dynamische Datenstrukturen
  - Instanzvariablen => dynamische Datenstrukturen
- Adressierung
  - Befehle brauchen Adressen
  - LDV, STV, …
  - JMP, JMN
- Lage der Ziele zur Laufzeit?
  - Ladepunkt im Speicher
  - relative Adressierung der Sprünge
  - relative Adressierung der Daten mit Basisregister
  - besonderer Code für dynamische Bibliotheken (DLLs))





- Adressen in anderen Modulen?
  - Auflösung durch Linker
  - Auflösung durch Lader



3.5 Assembler

- Assemblersprache
  - menschenverständliche Form der Maschinenbefehle
  - Mnemonics: LDV, ADD, JMP, …
  - Variablen, Datenstrukturen
  - Makros
- Programm zum assemblieren
  - transformiert Assembler-Programme in Objektcode
  - Modularisierung
  - Speicheraufteilung
  - Sprünge berechnen
- Cross-Assembler
  - läuft auf Maschine A
  - erzeugt Code für Maschine B
  - für neue Computer oder kleine Architekturen
- Vorteile
  - Geschwindigkeit
  - kompakter Code
  - Zugang zu Spezialbefehlen

3.5.1 Elemente der Assemblersprache

- Vordefinierte Namen für die Instruktionen
  - aussagekräftig
  - instruction mnemonics"
- Namen für Speicheradressen:
  - frei wählbar
  - Instruktions- und Datenadressen
  - Sprungmarken
  - Variablen
- Literalkonstanten:
  - Dezimalzahlen
  - Hexadezimalzahlen
  - Zeichenkonstanten in Hochkommas
  - bedeuten Werte oder Adressen
- Kommentar nach Strichpunkt
- gegenwärtige Speicheradresse : *

3.5.2 Anweisungen der Assemblersprache

- Konstantenvereinbarung:

```
< name >      =    < wert >                    [; < kommentar >]
; acht       =    8
```

- Ladepunktanweisung:

```
*             =    < Adresse >                 [; < Kommentar >]
; *          =    start                       ; Anweisung für Assembler
```

- Speicherdeklarationsanweisung:

```
[< Name >]    DS   [< Wert >]                  [; Kommentar]
; index       DS   0                          ; ein Wort, nicht typisiert
```

  - Datentypen z.B. DL, DW, …

- Instruktionsanweisung:

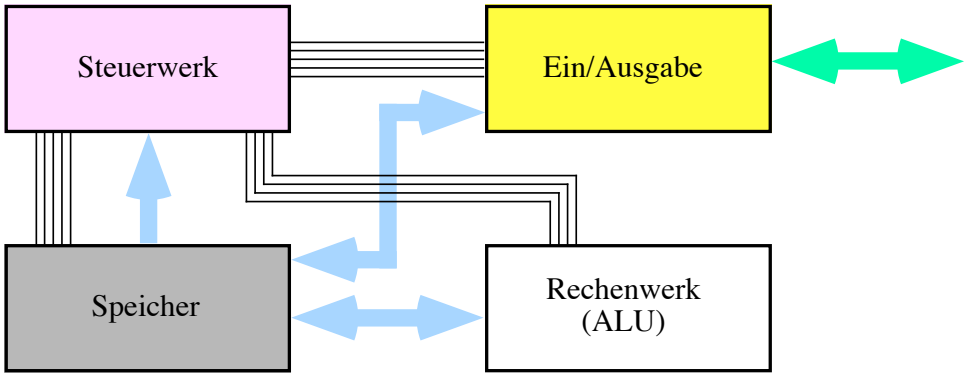
```
 [Name]       Instruktionsname    [Operand [± Wert] ]  [; Kommentar]
; start       LDV                 index
```

  - **[± Wert]** Modifikation des Operandenwertes (-adresse) durch Assembler

- Bessere Assembler bieten auch Module, Scopes etc.

3.6 Taxonmie von Rechnerarchitekturen

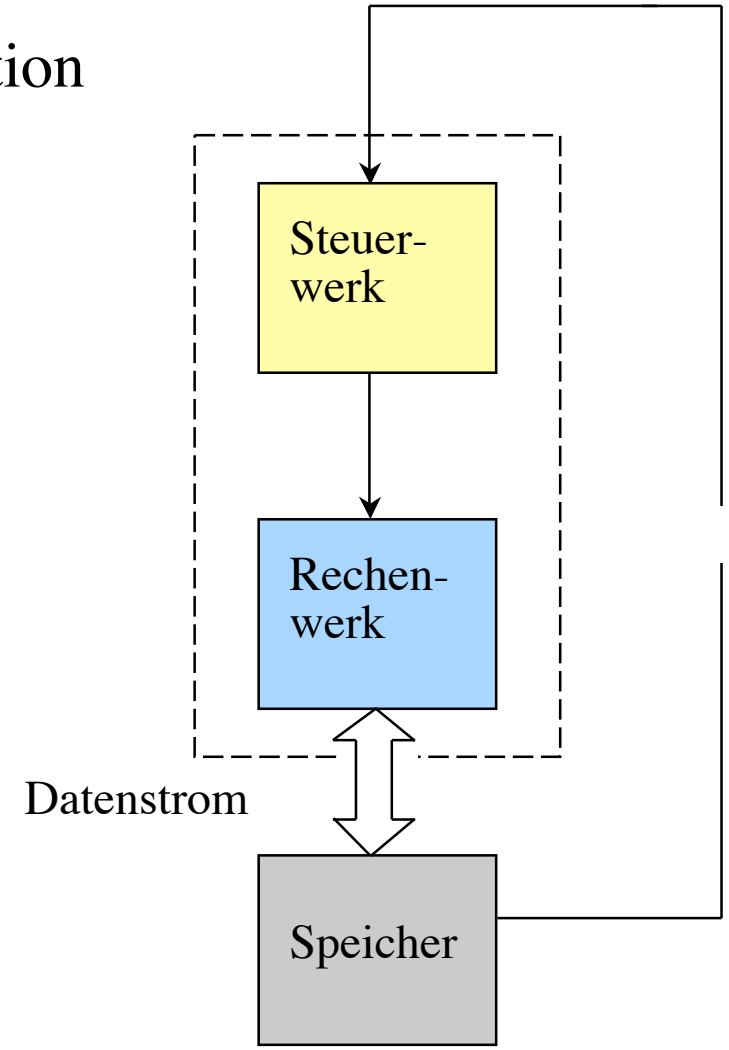
3.6.1 von Neumann



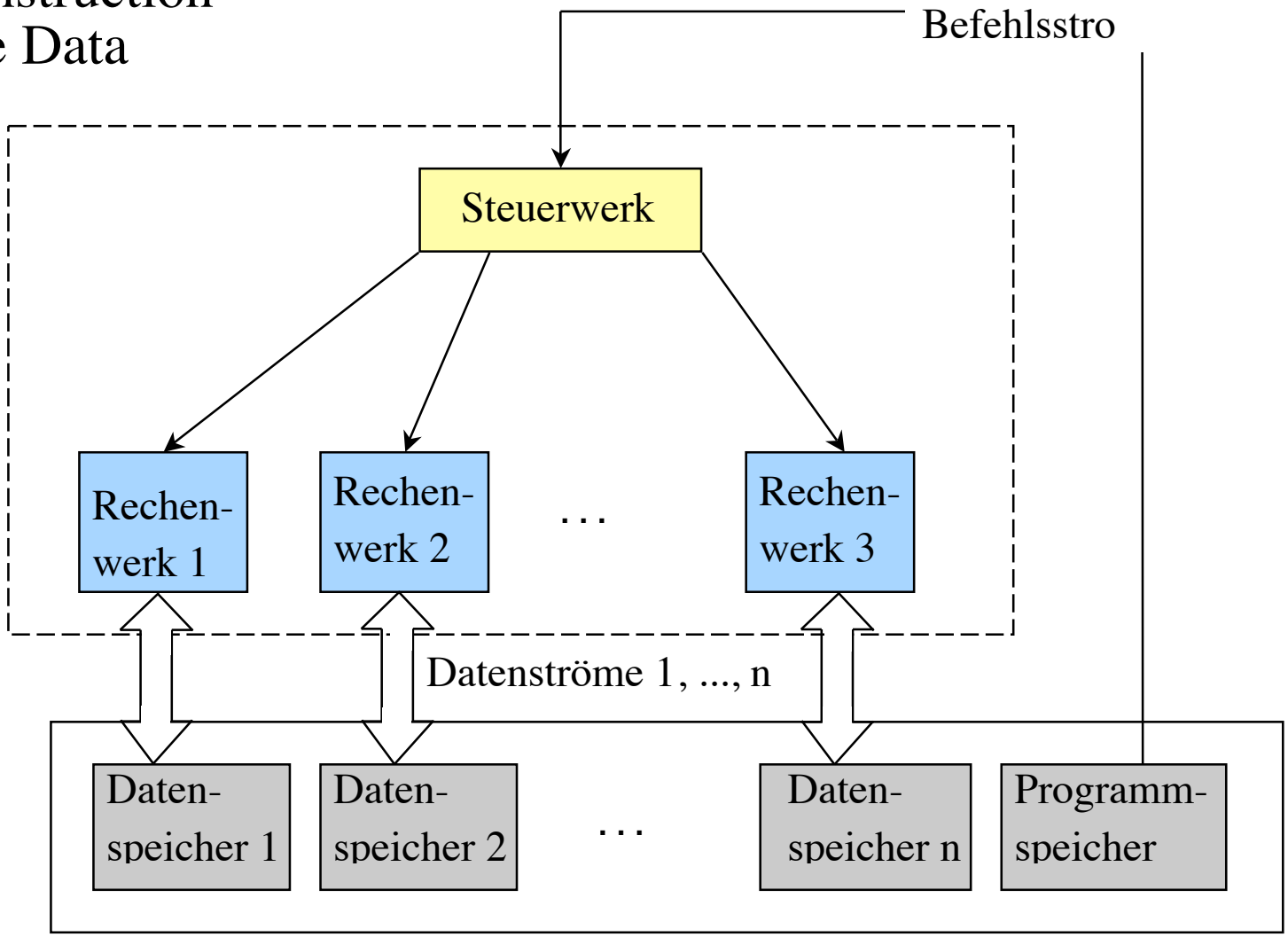
- Komponenten-Modell
- Befehle
  - ein Datenwert
  - seriell abgearbeitet
- Speicher
  - Code und Daten ohne Unterschied
  - kein eingebauter Zugriffsschutz
  - unstrukturiert und nicht typisiert: Semantik im Programm
  - linear adressiert
- Verbindung Speicher-CPU: von-Neumann Flaschenhals

3.6.2 Klassifikation nach Flynn

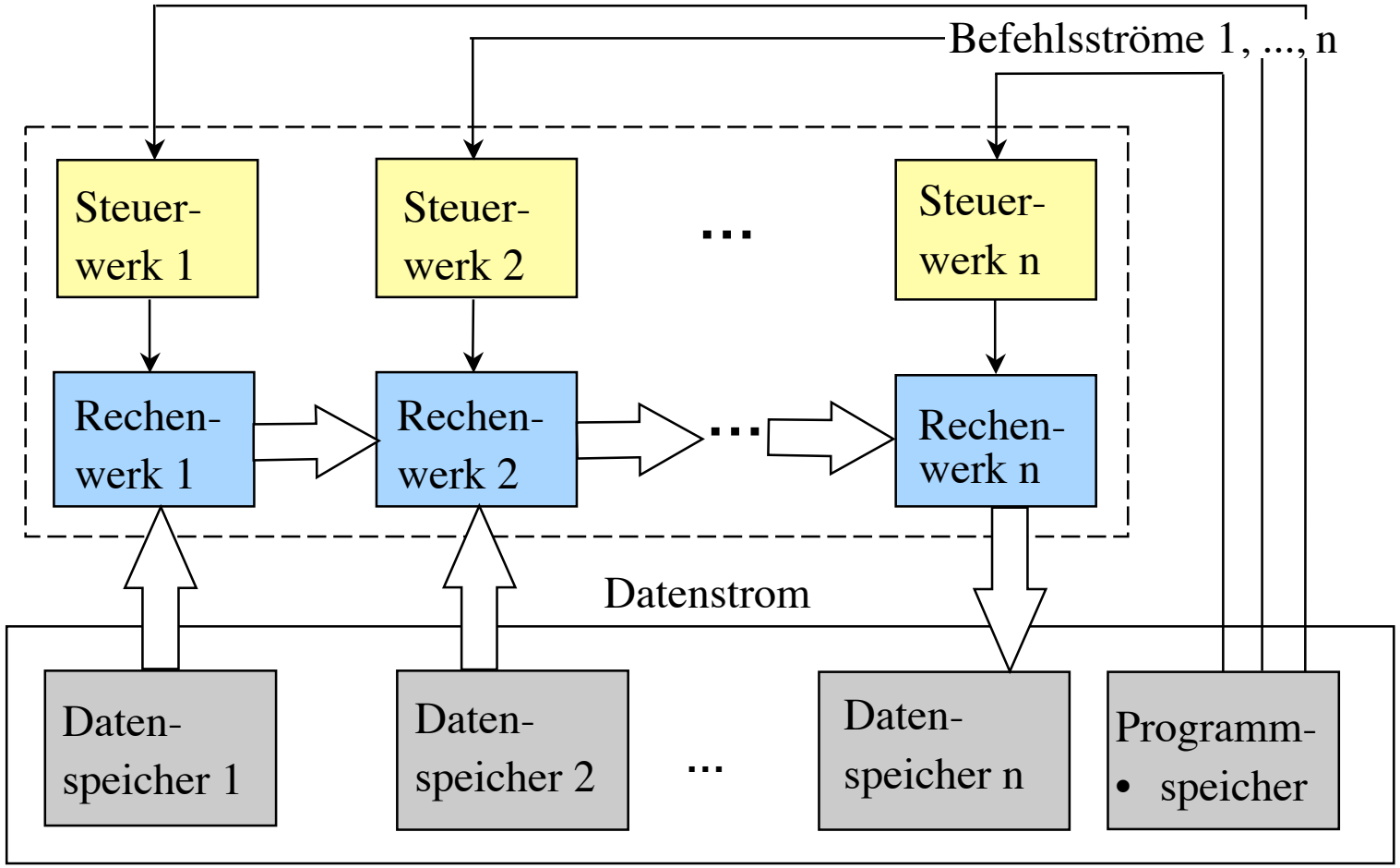
- SISD
  - single Instruction
  - single Data



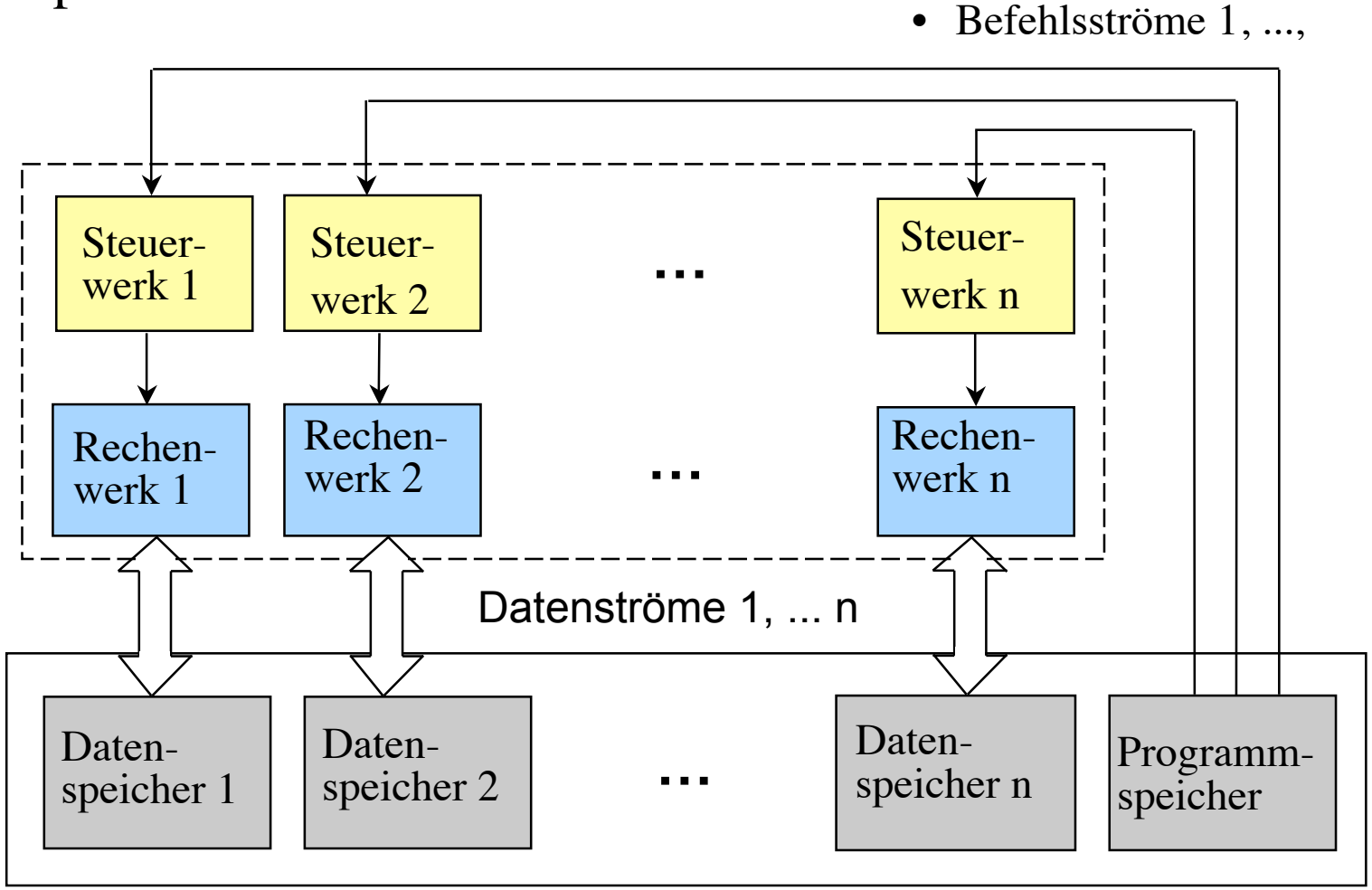
- SIMD
  - single Instruction
  - multiple Data



- MISD
  - multiple Instruction
  - single Data



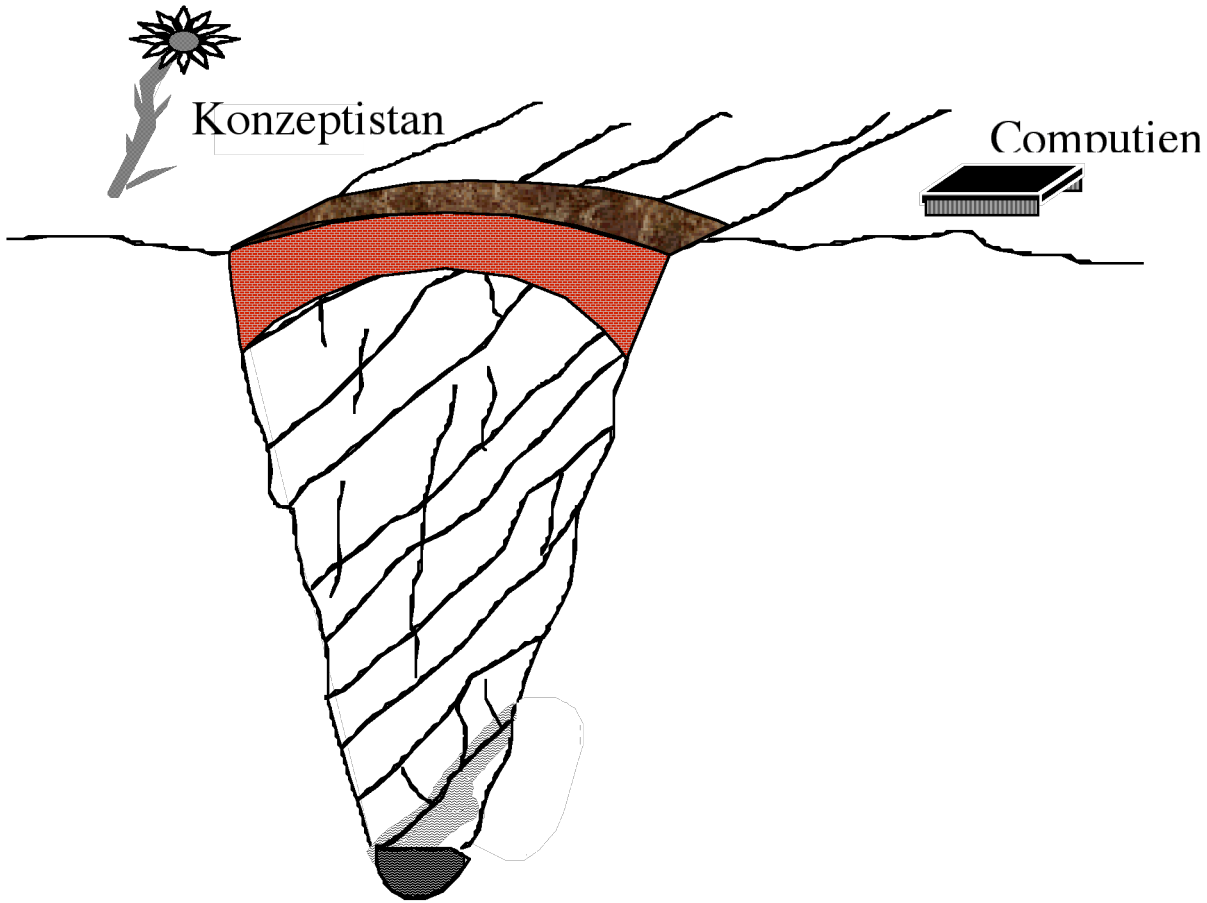
- MIMD
  - multiple Instruction
  - multiple Data



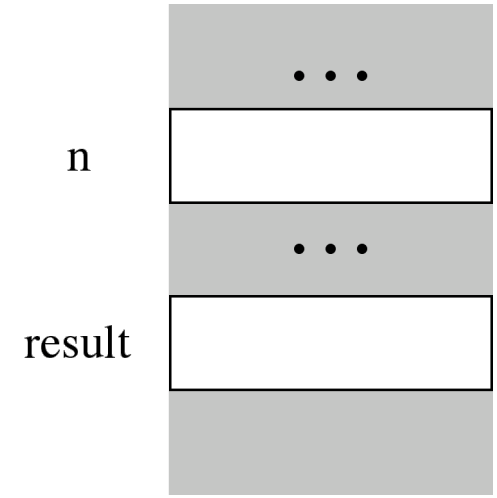
4. Instruktionssätze

- Programmiermodell
- Semantische Lücke
  - Datentypen
  - Objekte
  - Prozeduren
  - Formeln

  - RAM
  - Register
  - Operationen
- Datenfluss-Maschinen
- Mengen von Befehlen

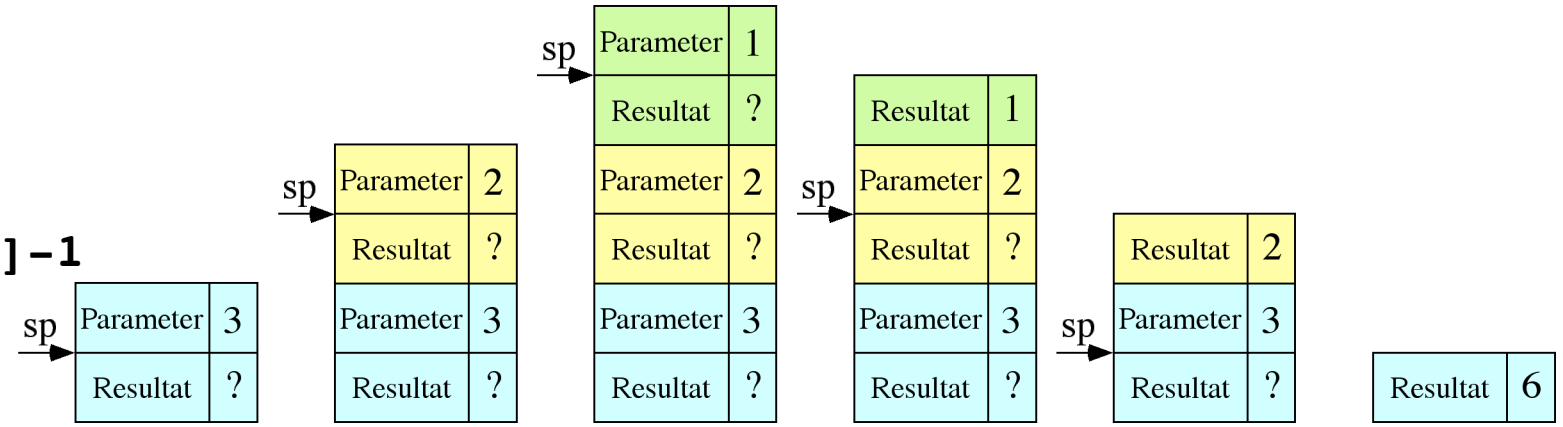


- Prozeduren und Parameter
  - Aufrufender: Platz für Parameter und Resultate
  - Gerufener: Platz für lokale Variablen
  - naiv: statische Allokation durch Compiler
  - Beispiel: fac(3)=1?

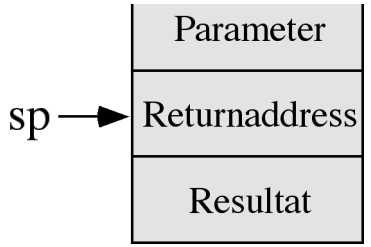



```
int fac(int n)
{ if (n==1) return 1;
  else return fac(n-1)*n;
}
…
fac(3);

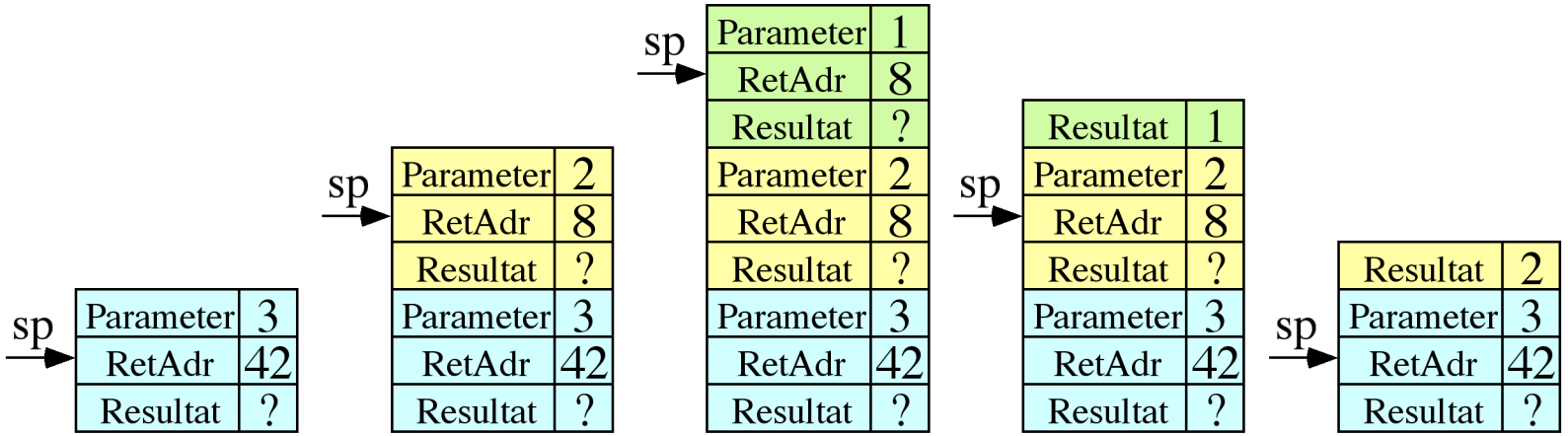
if (mem[sp]==1)
{   mem[sp-1]=1
    "return" }
mem[sp+2]=mem[sp]-1
sp=sp+2
"call"
sp=sp-2
mem[sp-1]=mem[sp+1]*mem[sp]
"return"
```



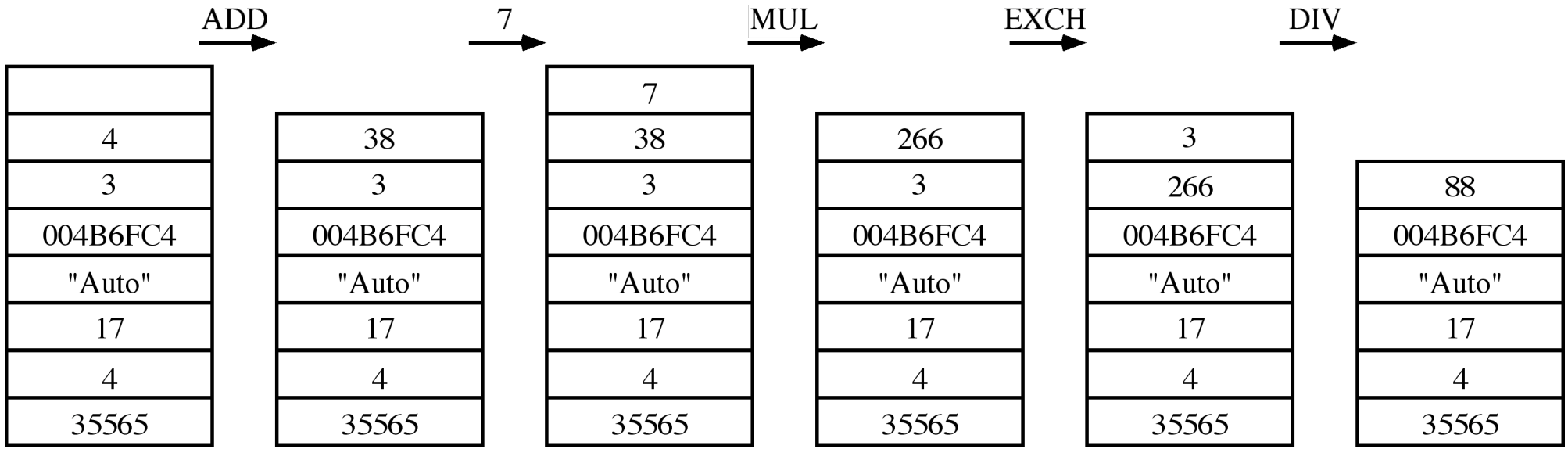
- Integration der Return-Adresse
  - Stackframe
  - z.B. Aufrufer verwaltet Stackframe




```
                                        int fac(int n)
01 if (mem[sp+1]>1) jump 04             { if (n<=1)
02 mem[sp-1] = 1                           return 1;
03 jump 10
04 mem[sp+4] = mem[sp+1] - 1
05 sp=sp+3
06 mem[sp] = 08
07 jump 01
08 sp=sp-3
09 mem[sp-1] = mem[sp+1] * mem[sp+2]       return fac(n-1)*n;
10 jump mem[sp]                         }
…
38 mem[sp+4]=3
39 sp=sp+3
40 mem[sp]=42
41 jump 1
42 sp=sp-3
```



- Stack als Maschinenmodell
  - Java Virtual Machine
  - p-code, Interpreter
  - PostScript



- Speicher-Speicher
- Register-Modelle
  - Akkumulator (siehe Mima)
  - GPR: General Purpose Register
- GPRs
  - Register-Speicher
  - Load-Store
  - viele Register -> viele Bits in der Instruktion (Länge!)
  - wenige Register -> viele Speicherzugriffe
- Vergleich für `c = a+b;`

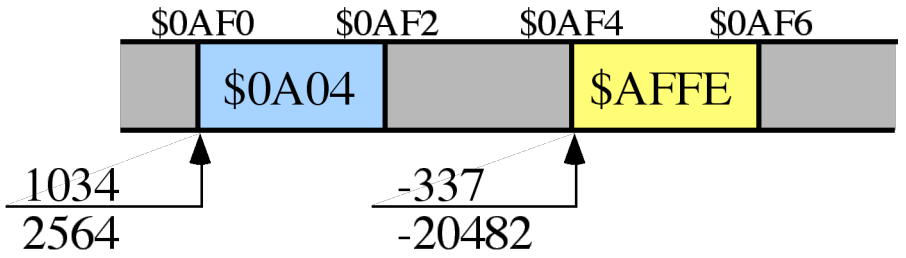
Stack	Akkumulator	Register-Speicher	load-store
`Push a`	`LDV a`	`Move a,R1`	`Load R1,a`
`Push b`	`ADD b`	`Add b,R1`	`Load R2,b`
`Add`	`STV c`	`Move R1,c`	`Add R3,R1,R2`
`Pop c`			`Store c, R3`

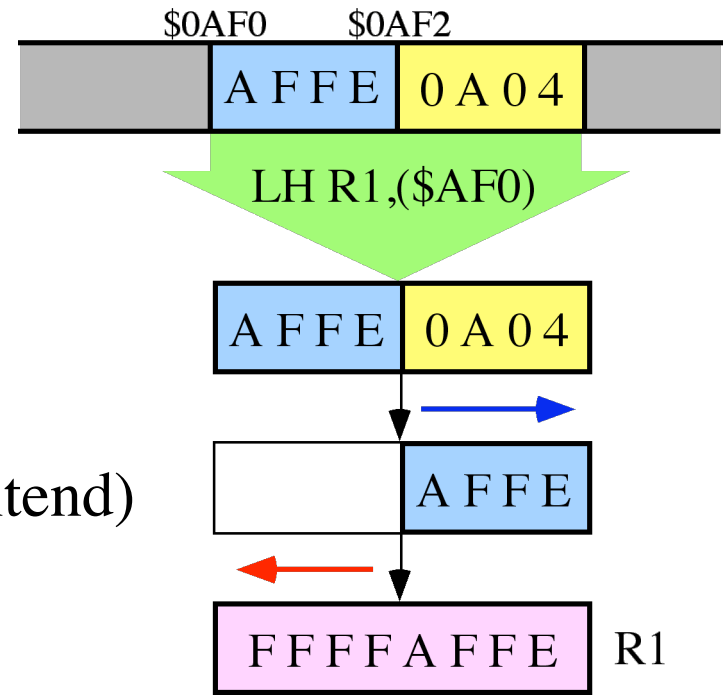
- Konventionen für GPRs
  - Pointer: Stack, Procedure-Frame, Globals, With, …
  - Procedure-Parameter
  - Temporäre Variablen, Expression-Eval, …
- Anzahl Operanden
  - 2: Operation Op1, Op2/Ziel
  - 3: Operation Ziel, Op1, Op2
  - alle Register/Memory?
- GPR-Maschinen

Typ	Beispiel	Vorteile	Nachteile
Reg-Reg (0,3)	SPARC, PPC, Mips, Alpha, HP-PA	Feste Befehlslänge, einfache Code-Generierung, feste Taktlänge	Mehr Instruktionen, evtl. Bits verschwendet
Reg-mem (1,2)	80x86, 68000	gute Codedichte, einfach zu codieren	unterschiedliche Befehlslänge, Operand verloren (Erg.)
Mem-Mem (3,3)	VAX	sehr kompakt	Befehle unterschiedlich lang, Speicherflaschenhals, unterschiedliche Taktanzahl

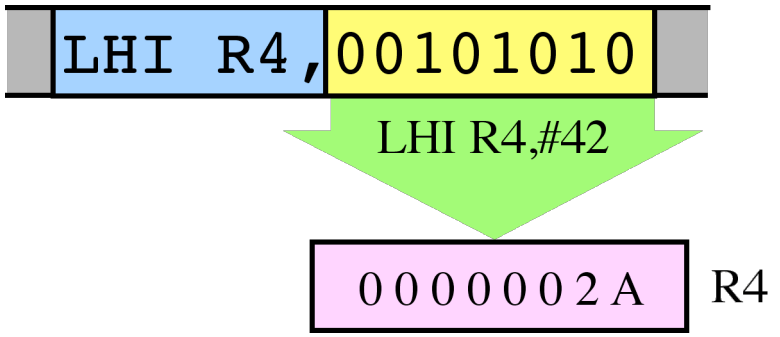
4.1 Adressierung und Operanden

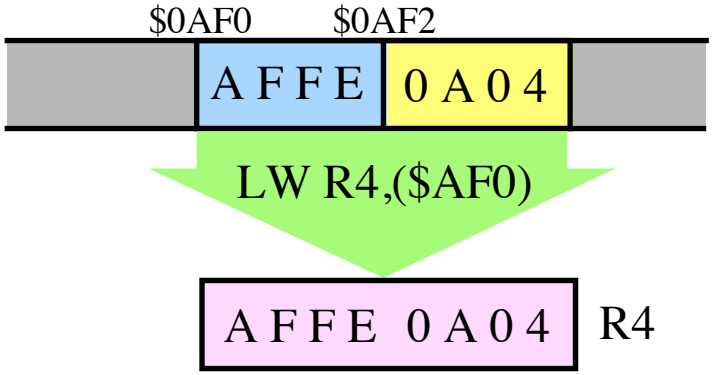
- Operandenlänge
  - 8, 16, 32, 64 Bit
  - Bitzugriff meist Bytezugriff
- Little-Endian
  - niederwertigstes Bit/Byte an der Adresse
  - Intel, Alpha
- Big Endian
  - höchstwertigstes Bit/Byte an der Adresse
  - 68000, PPC, …
- Alignment
  - Halbwort-Adressen durch 2 teilbar
  - Wortadressen durch 4 teilbar
  - 8, …
  - Gesamtes Speicherwort wird geholt
  - Schaltung zum Ausrichten (Shift + Sign-Extend)
  - Auswirkung auf Rest-Register?



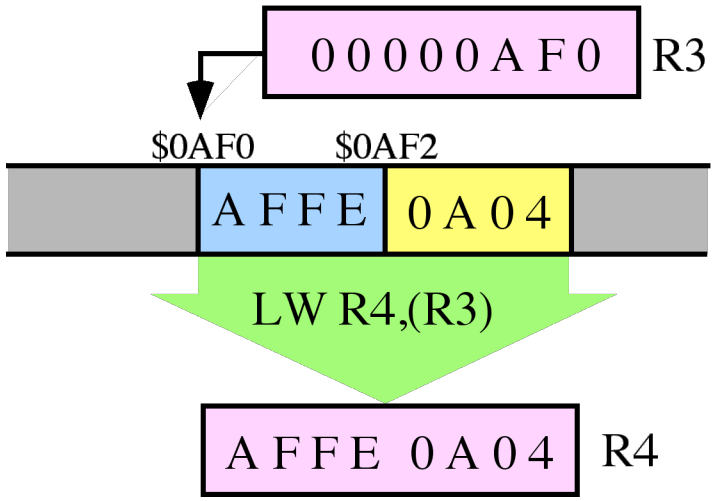


- Adressierungsmodi
  - Mima: Adresse in der Instruktion (einfache Variable)
  - Mima: Operand in der Instruktion (Konstente)
  - weitere Modi: indirekte Adressierung (Pointer, …)
- Immediate
  - Operand im Befehl
  - `LHI R4,#42`
  - Konstante laden
  - Shift, ALU-Ops, Compare, …
- Direkt (absolut)
  - EffektiveAdresse = Instruktion AND 007FFFFF
  - `LW R4,($0AF0)`
  - Adresselänge beschränkt
  - oder Instruktion 2 Worte

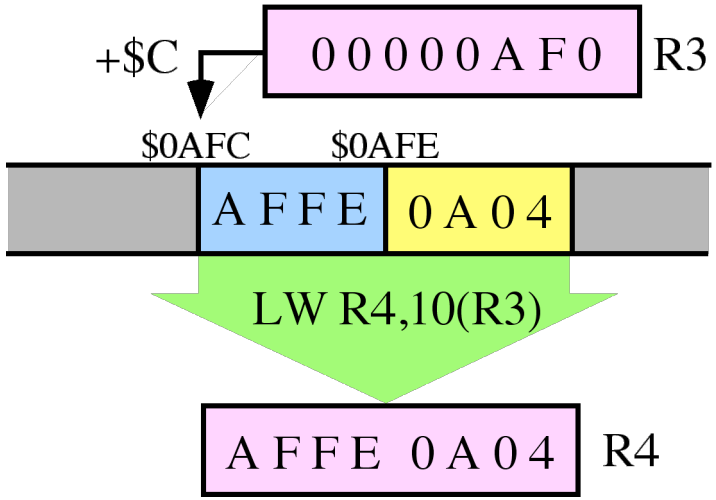




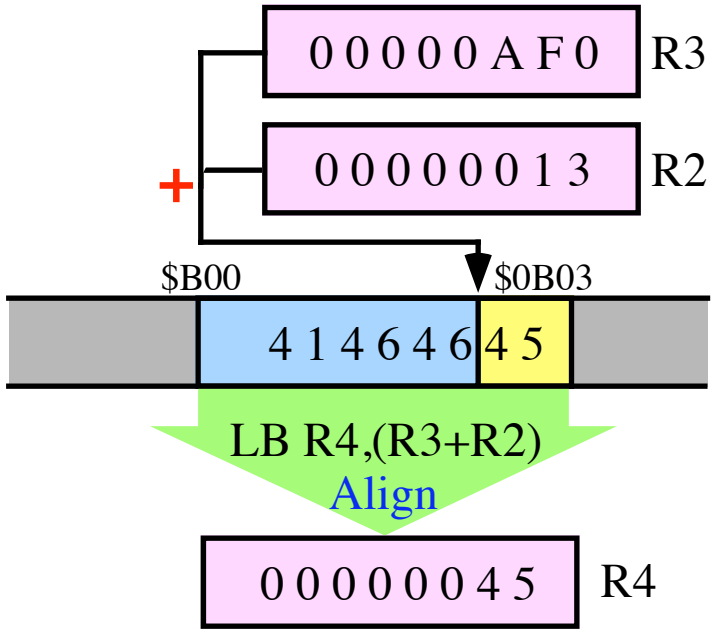
- Register Indirekt
  - EA = Rn
  - `LW R4,(R3)`
  - Pointer
  - Adressrechnung
  - komplexe Arrays



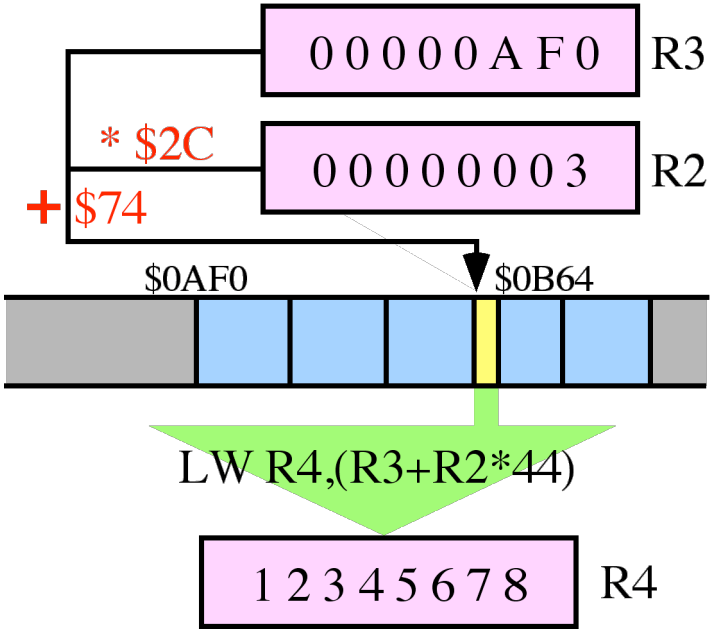
- Register Indirekt mit Displacement
  - EA = Rn+Disp
  - `LW R4,12(R3)`
  - Feld im Record
  - lokale Variablen (Stackframe), …



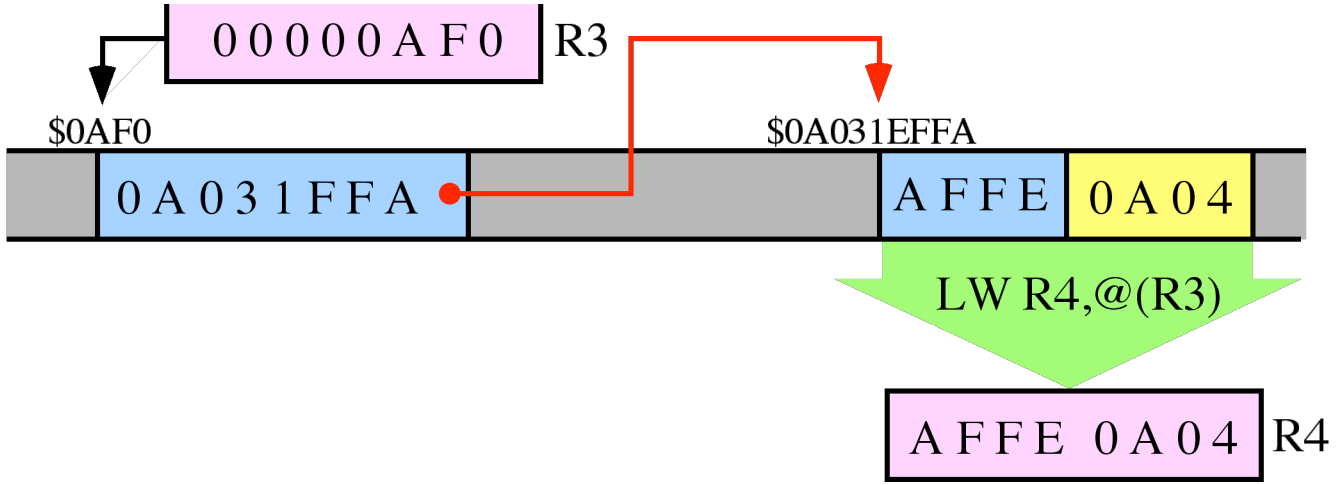
- Register Indirekt mit Index
  - EA = Rn + Ri
  - `LB R4,(R3+R2)`
  - z.B. Char-Array
  - R3 Zeiger auf Vektor
  - R2 z.B. Schleifenindex



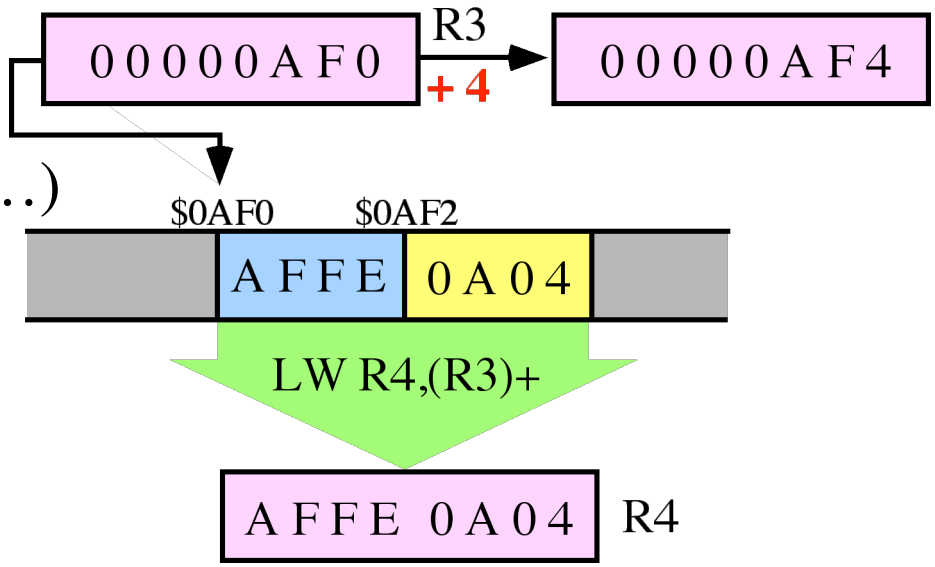
- Register Indirekt mit Index*Scale
  - EA = Rn + Ri*scale
  - `LW R4,(R3+R2*44)`
  - Vektor mit Elementen > Byte



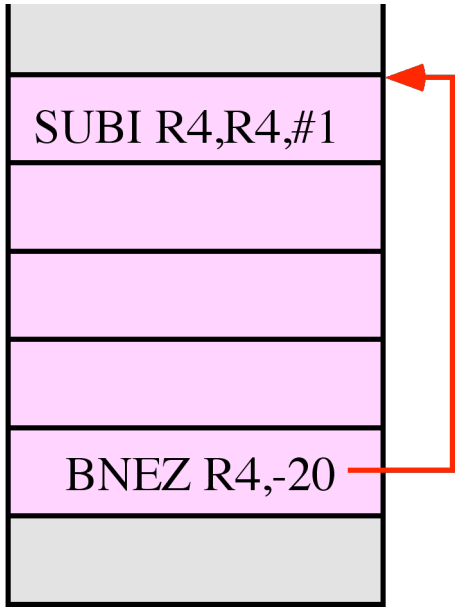
- Speicher Indirekt
  - EA = Speicher[Rn]
  - `LW R4,@(R3)`
  - Handles dereferenzieren



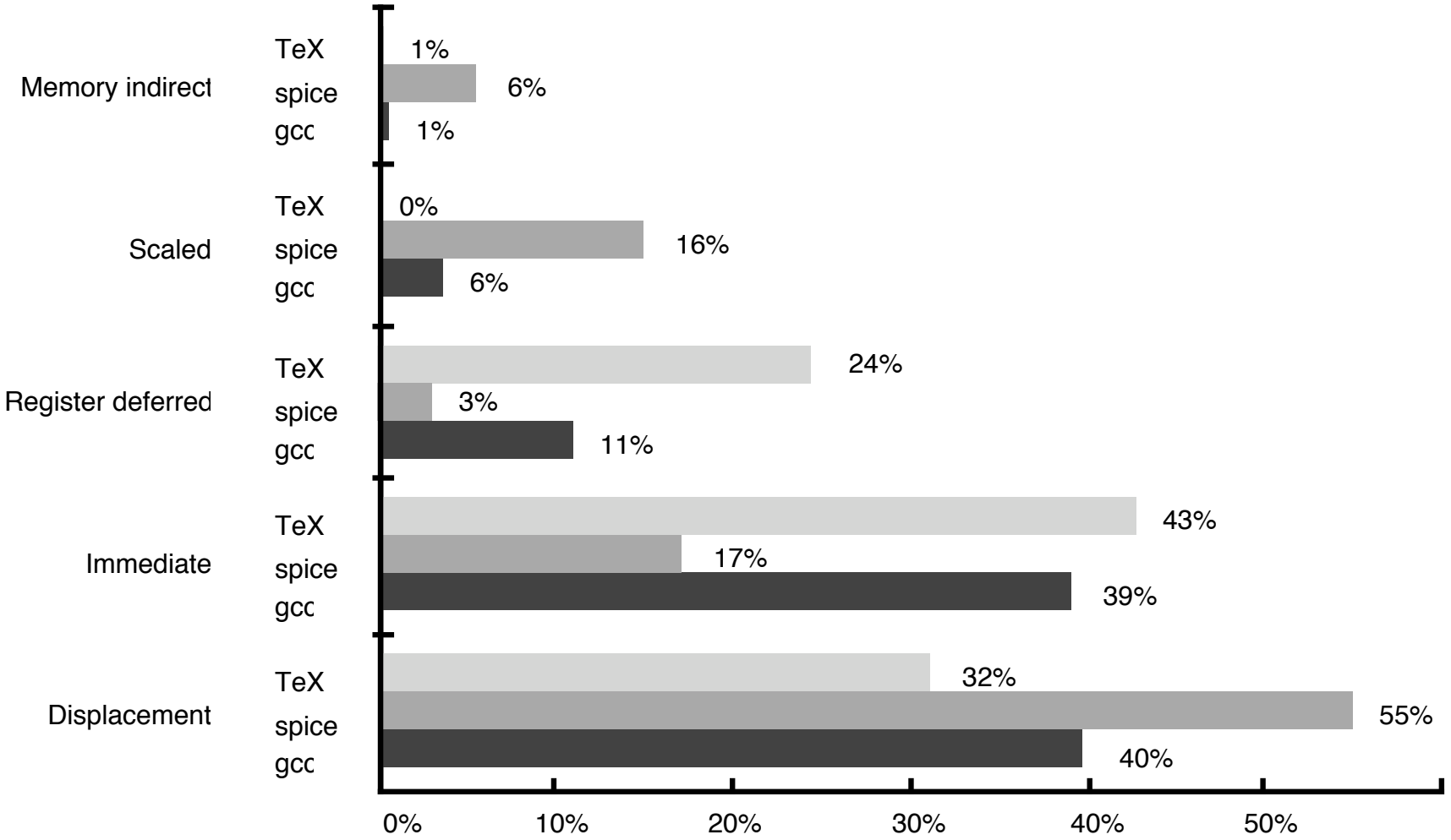
- Increment und Dekrement
  - post-operation: EA = Rn; Rn = Rn ± size
  - pre-operation: Rn = Rn ± size; EA = Rn;
  - Operandentyp ergibt Increment-Wert
  - `LW R4,(R2)+`
- Architekturspezialitäten
  - segmentbasiert (Datensegment bei 8086, …)
  - PC-relativ (strings von Prozeduren)



- Sprungadressen: PC-relativ
  - Ziel = PC+displacement
  - `if-then-else` etc.
  - `for, while, …`
  - positionsunabhängiger Code
- Sprungadressen: Register-indirekt
  - Ziel = Rn, Rn+Ri…
  - ähnlich Datenadressmodi
  - case/switch, DLLs, virtuelle Funktionen, Sprungtabellen, …
- Absolute Sprünge
  - relozieren beim Laden
  - Standard-Adressraum?



- Häufigkeit der Adressierungsarten (VAX)



=> LW R4,@(R3) ersetzen durch: LW R2,(R3) und LW R4,(R2)

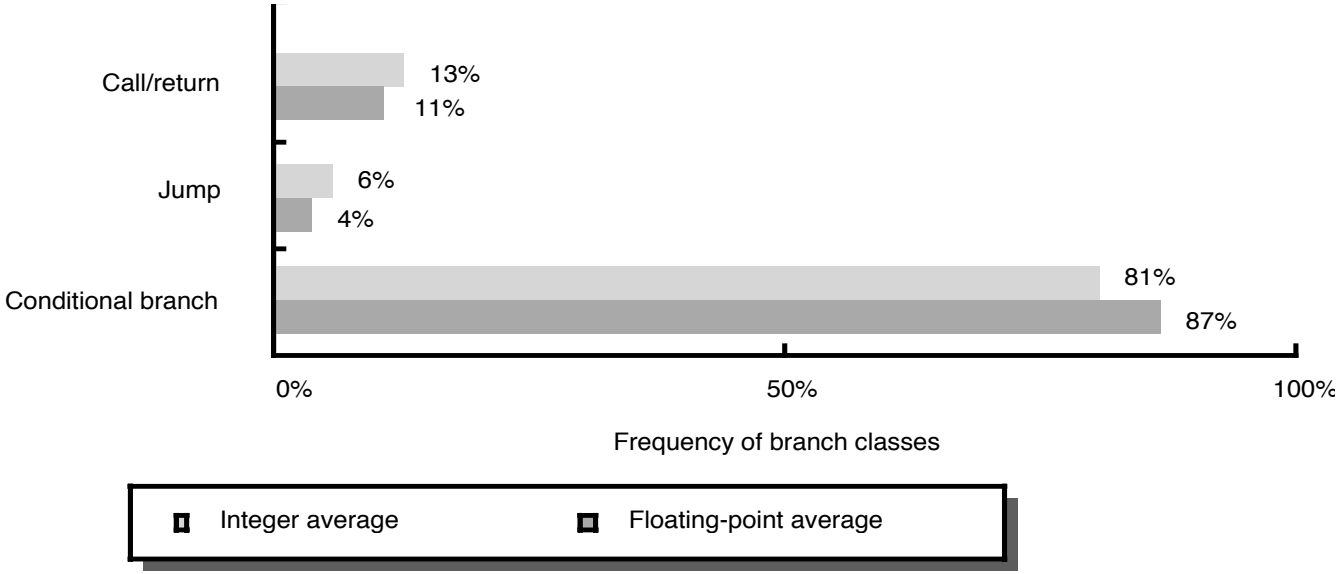
- Instruktionssatzentwurf
  - nur compilererzeugbare Adressmodi
  - seltene Adressierungsarten ersetzen
- Wichtige Adressierungsarten
  - Indirekt mit und ohne Displacement
  - Immediate
  - mehr als 75%
- Displacement-Feldgröße
  - 12 - 16 Bit
  - mehr als 75%
  - Displacement zu groß => Adressrechnung
- Immediate-Feldgröße
  - 8 - 16 Bit
  - 50%-80%
  - Immediate zu groß => Konstante im Speicher

4.2 Operationen

- Instruktionsgruppen
  - Kontrollfluß
  - Datentransfer
  - Arithmetische und logische Operationen
  - Vektorbefehle (SIMD), Multiply-Accumulate (MISD), …
  - Floating Point
  - Systemaufrufe
  - Dezimal, String
- Lokale Verzweigung
  - aus Programmiersprachenkonstrukten
  - J, JR; B, …
  - geringes Displacement: -128 .. 127
- Sprung
  - größere Distanz
  - J, JR; JMP, J, …
  - evtl. tabellisiert
  - typisch register- oder speicherindirekt

80x86 Befehlssatz

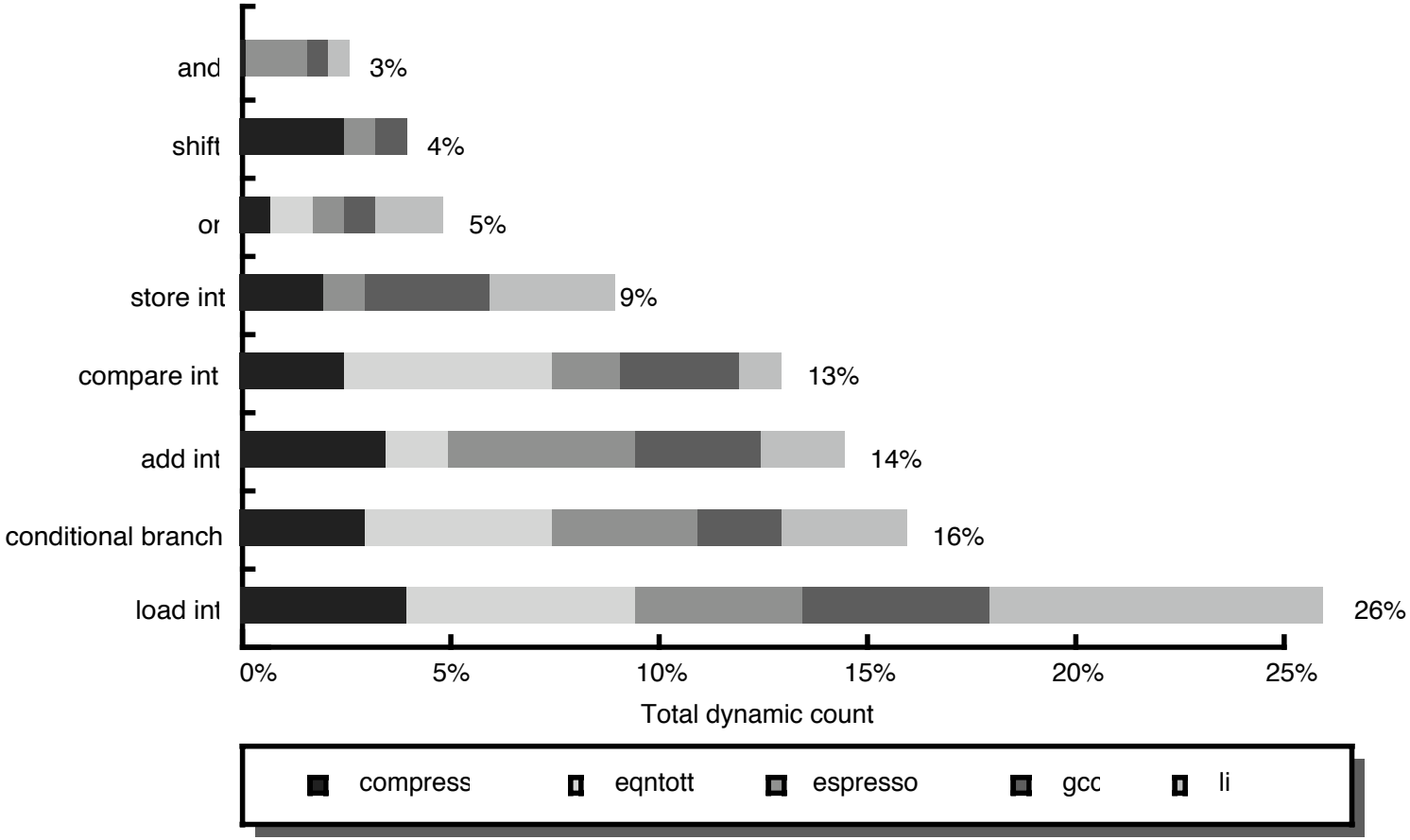
load	22%
conditional branch	20%
compare	16%
store	12%
add	8%
and	6%
sub	5%
move register-register	4%
call	1%
return	1%
(SpecInt92)	95%



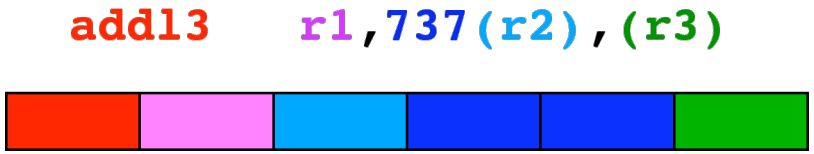
- Bedingte Verzweigung
  - Condition Code
  - von ALU gesetzt
  - evtl. beim Laden gesetzt
  - Zero, Negative, Carry, Borrow, …
  - Compare-Befehl
- Prozeduraufrufe
  - Hin-Sprung (JAL, JALR)
  - lokal: mittleres Displacement
  - auch entfernt
  - evtl. tabellisiert (DLL, Interruptvektor,)
  - Rücksprung (JR, ; RET)
  - Register retten

- Arithmetische und logische Operationen
  - Integerarithmetik: Add, Sub, Multiplikation, Division
  - And, Or, …
  - Compare
  - Bitfeldoperationen
- Floating Point
  - Koprozessor-Konzept
  - Hardware und evtl. Software
  - eigene Register, Transferbefehle
  - CPU muss trotzdem eff. Adresse generieren
  - evtl. nur Transfer aus GP-Registern
  - Datenformate: IEEE 754
- Konfiguration und Management
  - Interruptsperren
  - Version
  - Prozessorstatuswort
  - Test-and-Set
  - Interrupt (TRAP, RFE; IRET)

- Datentransfer
  - Load und Store bzw. Move
  - Schwerpunkt Adressierung
  - Register-Spec
  - Speicher-Adresse
- Relative Häufigkeit (DLX, SpecInt92)



- Effiziente Kodierung der Befehle
  - Operation und Register
  - Adressen und Immediates
  - kompakt: Kodedichte und schnelles Laden
  - Platz für Adressen und Immediates
  - einfach dekodierbar
  - leicht generierbar
- Opcode
- Adress-Spezifikation
  - Adressierungsmodus
  - eigenes Feld
  - Register
- Instruktion soll in 1 Wort passen
  - 32 Bit, 64 Bit
  - Verschnitt bei vielen Befehlen
- Pipeline-Effizienz



- Amdahls Gesetz
- Den häufigen Fall beschleunigen
  - tatsächlich ausgeführte Instruktionen

$$Speedup = \frac{1}{(1 - schnellerer_Anteil) + \frac{schnellerer_Anteil}{Beschleunigung}}$$

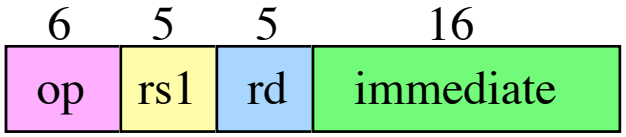
  - Bsp: 90% Anteil 10-fache Beschleunigung

$$Speedup = \frac{1}{0,1 + \frac{0,9}{10}} = \frac{1}{0,19} = 5,2631$$

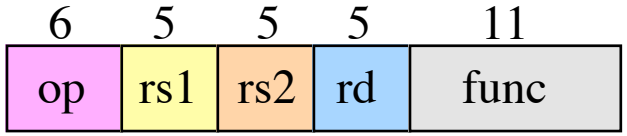
- Beispielapplet
  - http://www.cs.iastate.edu/~prabhu/Tutorial/CACHE/amdahl.html

$$CPU - Zeit = Anzahl_Inst * \frac{Zeit}{Zyklus} * \frac{Zyklen}{Befehl}$$

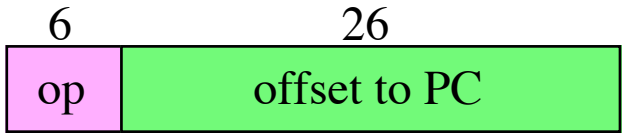
- DLX-Architektur
  - Load/Store, 32 GPRs, R0=0
  - 32 Bit, Big Endian
  - 8, 16, 32 Bit Integer
  - Adressierung indirekt mit 16 Bit Displacement
  - Adressierung immediate mit 16 Bit
  - 32 Floating Point Register (32 Bit) = 16 * 64 Bit
- DLX-Instruktionskodierung
  - 32 Bit Befehle
  - 6 Bit Opcode inkl. Adressierung
  - ALU-Funktion 11 Bit
- I(mmediate)-Typ
  - load, store, …
  - conditional branch, jump register, …



- R(egister)-Typ
  - ALU rd := rs1 func rs2



- J(ump)-Typ
  - Jump, Jump&Link, Trap, RTE



- Datentransfer
  - zwischen Registern und Speicher
  - zwischen Registern und FP-Registern
  - Adressierung 16-bit displacement + contents of a GPR

LB, LBU, SB	Load byte, load byte unsigned, store byte
LH, LHU, SH	Load halfword, load halfword unsigned, store halfword
LW, SW	Load word, store word (to/from integer regs)
LF, LD, SF, SD	Load SP float, load DPF, store SPF, store DPF
MOVI2S, MOVS2I	Move from/to GPR to/from a special register
MOVF, MOVD	Copy FP reg or a DP pair to another reg or pair
MOVFP2I, MOVI2FP	Move 32 bits from/to FP reg to/from integer reg

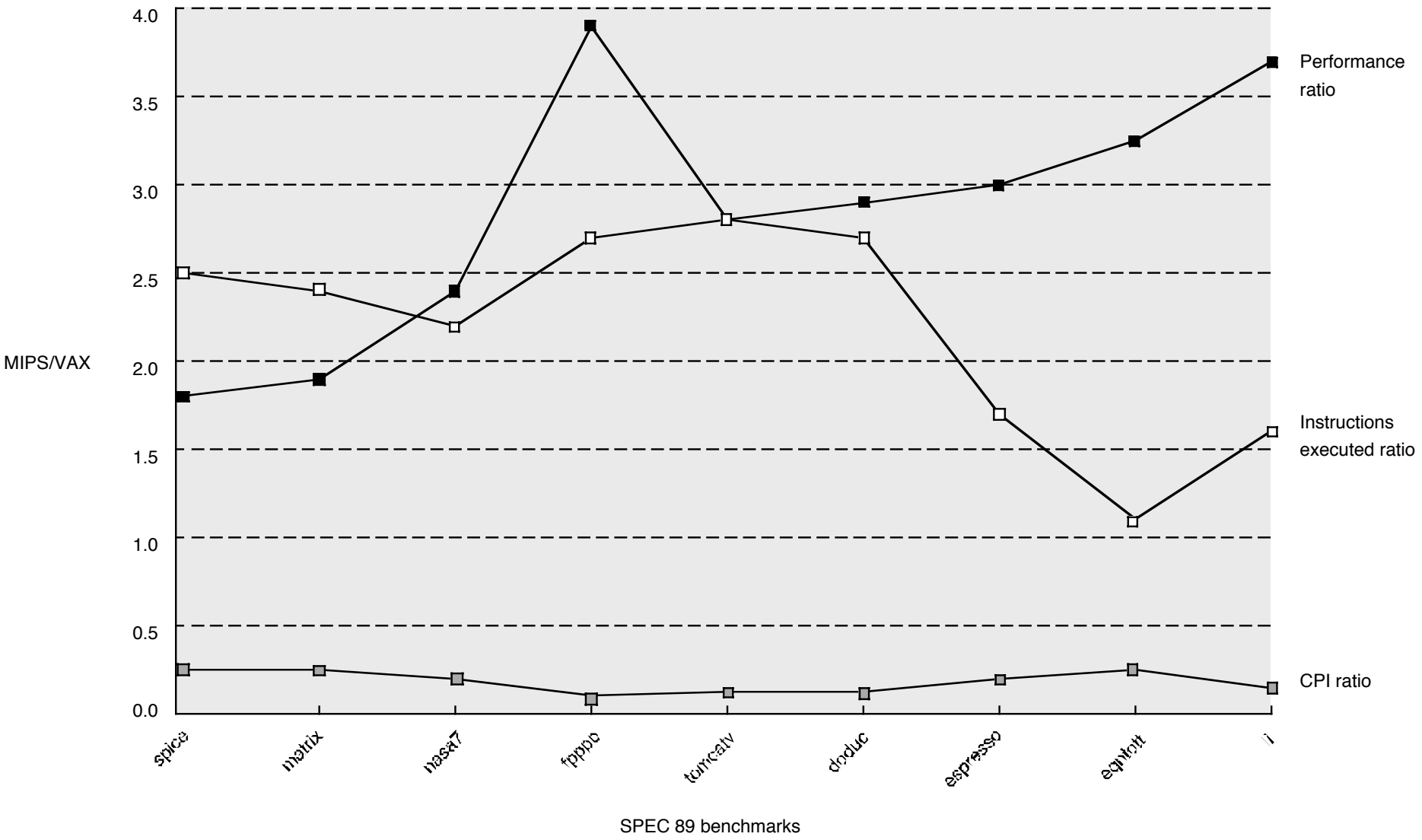
- Arithmetic / Logical
  - Integer Bzw Bitketten in Registern
  - Overflow => Trap

ADD, ADDI, ADDU, ADDUI	signed and unsigned Add, add immediate (all immediates are 16-bits);
SUB, SUBI, SUBU, SUBUI	Subtract, subtract immediate; signed and unsigned
MULT, MULTU, DIV, DIVU	* und /, signed bzw.unsigned Operanden floating-point register
AND, ANDI	Register oder immeadiate
OR, ORI, XOP, XOPI	Register oder immediate
LHI	Load high immediate
SLL, SRL, SRA, SLLI, SRLI, SRAI	Shifts
Sxx, SxxI	Set conditional: xx: LT, GT, LE, GE, EQ, NE

- Control
  - Conditional branches and jumps
  - PC-relativ oder Register-indirekt
  - 16-bit offset from PC

BEQZ, BNEZ	Branch GPR =/≠ zero
BFPT, BFPF	Test compare bit in FP status register and branch;
J, JR	26-bit Offset vom PC(J) oder Ziel in Register (JR)
JAL, JALR	Jump and link: R31:= PC+4;
	Ziel PC-relative (JAL) oder Register (JALR)
TRAP	Transfer to operating system at a vectored address
RFE	Return to user code from an exception; restore user code

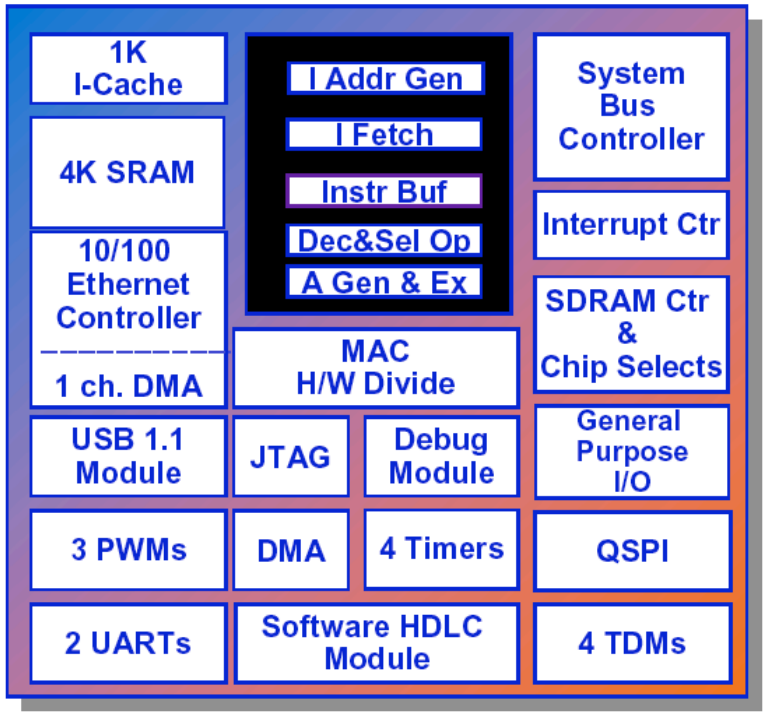
- Pro Befehl ein Maschinenzyklus
  - Vergleich VAX- MIPS



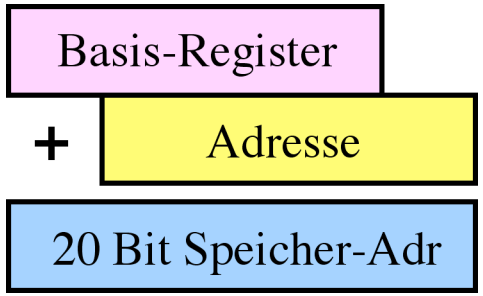
4.3 Stichwort: CISC

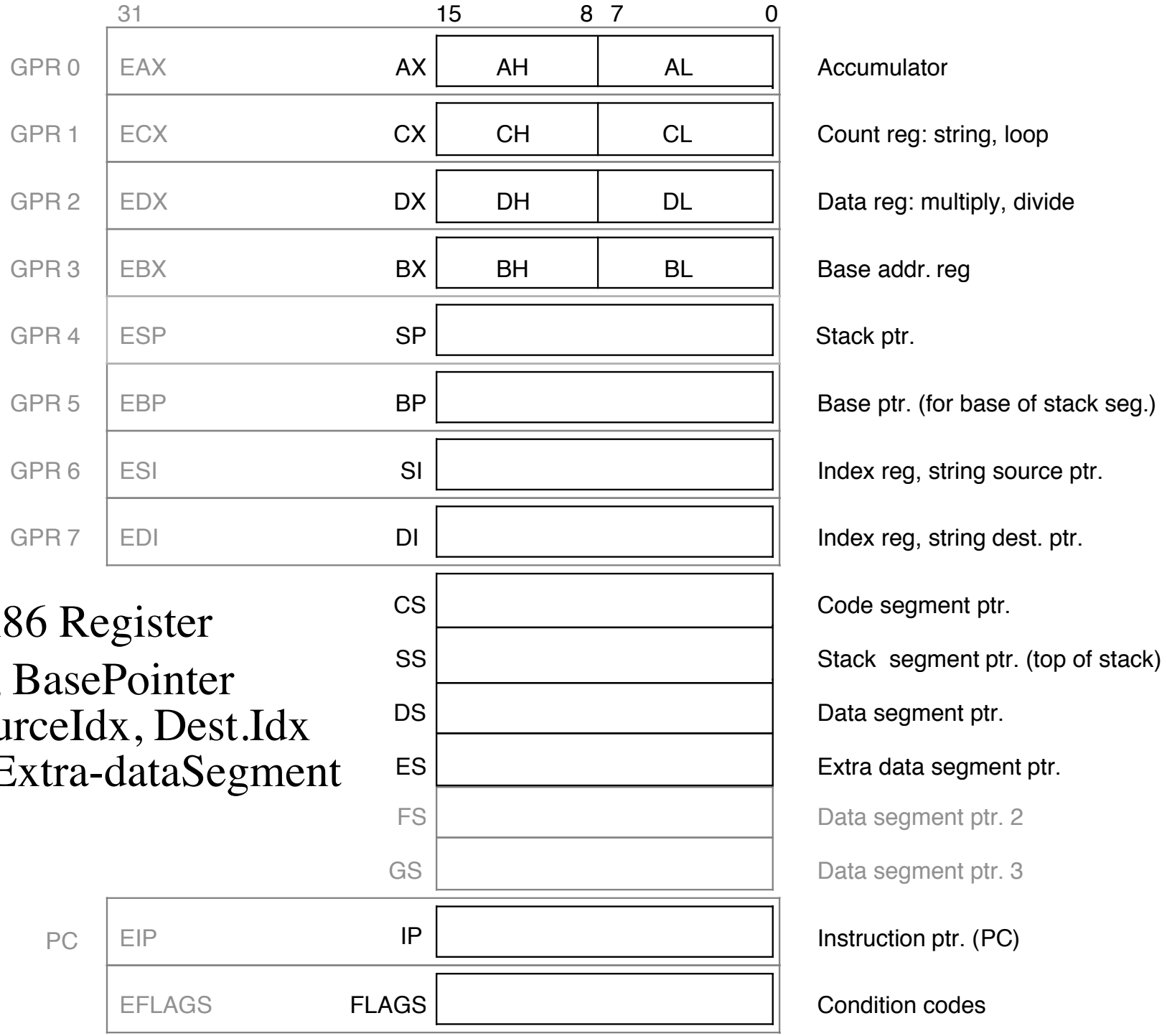
- Complex Instruktion Set Computer
  - individuelle Instruktionslänge: Speicher und Takte
  - evtl. dynamisch veränderliche Anzahl Takte
  - aufwendige Dekodierung
- Befehle mit viel Funktionalität
  - Anzahl Operanden
  - MISD-Instruktionen (Multiply-Accumulate, etc.)
- Mythos Programmiersprachen-Konstrukte
  - Schleifen => Schleifenbefehle (decrement & branch)
  - Funktionsaufrufbefehle mit Registerrettung
  - Case-Befehle
- Codedichte
  - viele Längen für Adress-Felder
  - Selektoren für Operandenzahl

- VAX - die ultimative CISC
  - 32 Bit Architektur, 32 Bit Bus
  - 16 Register, 12-15: AP, FP, SP, PC
  - leistungsstarke Befehle: CALL inkl. Register retten
  - INDEX-Instruktion: 45% schneller mit anderen VAX-Inst.
  - MicroVAX: 175 von 304 Instruktionen+Emulation
  - Nachfolger: Alpha
- Motorola 680x0 [1979]
  - orthogonaler Befehlssatz
  - 16 Register: A0-A7, D0-D7
  - 32 Bit Architektur, linearer Speicher
  - viele Adressmodi
  - 24 Bit externe Adressen
  - 68008: 8 Bit externer Daten-Bus
  - Multiplex von Adressen und Daten
  - 68020 [1984]: 3-stufige Pipeline,
    256 byte cache, 32 Bit extern
  - Coldfire ohne komplizierte 680x0 Instruktionen
  - Konfigurationen mit RAM, ROM und Interfaces



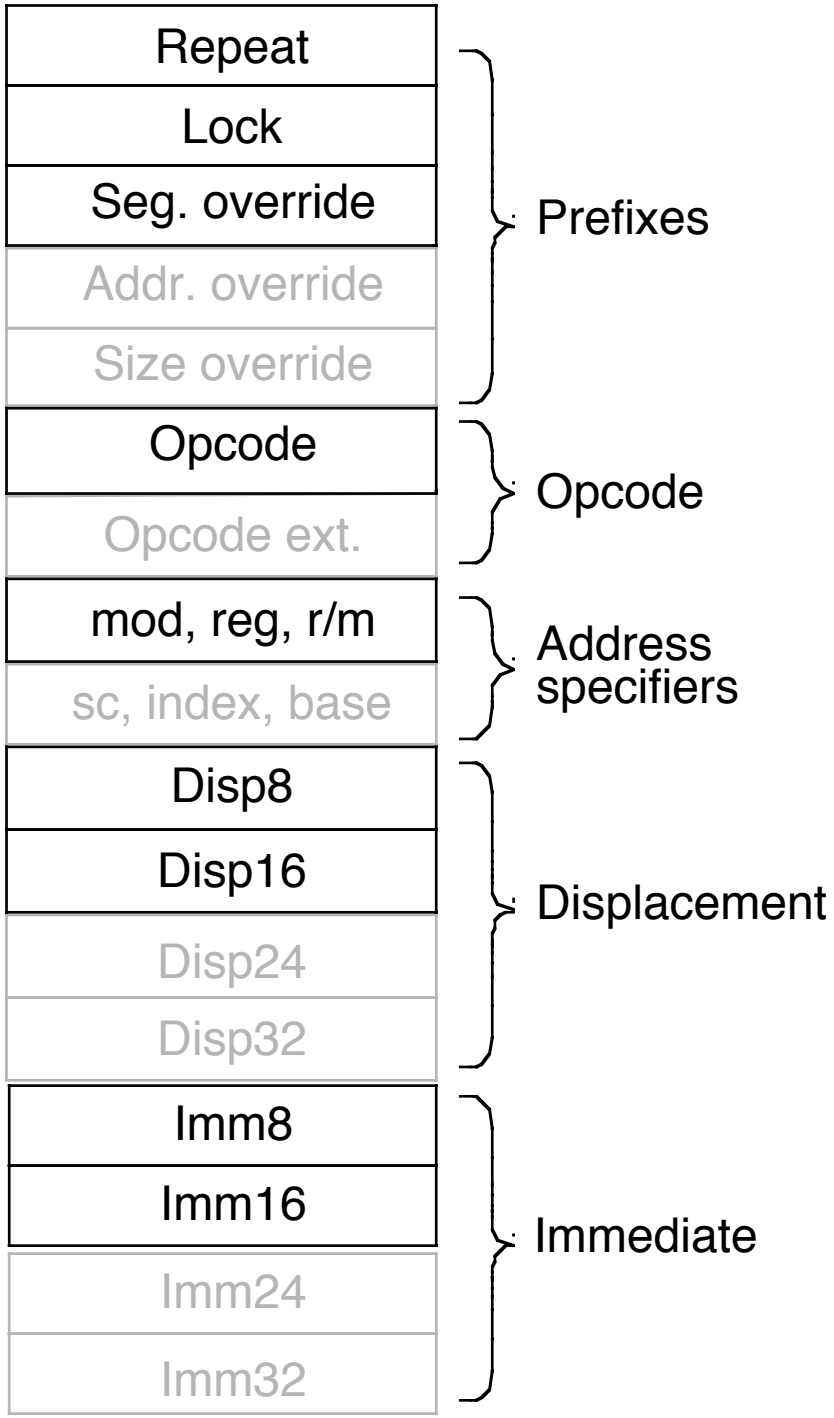
- IA: Intel Architecture
- Intel 8086
  - wenige Register: AX, BX, CX, DX, …
  - komplizierte Architektur
  - basiert auf 4004, 8008, 8080
  - segmentierter Speicher: 16 Bit Adressen + 16 Bit Basisregister
  - near/far Pointer und Jumps
- Evolution des 80x86
  - 8088, 8086, 80188, 80186, 80286
  - 80386DX, 80386SX
  - 80486DX, 80486DX2
  - Pentium
- Pentium Pro, PII, PIII, P4
  - RISC-Kern mit Übersetzung für 80x86-Befehle
  - Pipeline 14 und mehr Stufen
- 'Megahertz sells'
  - extreme Pipeline => extreme Taktzahlen
  - hohes Pipelinerisiko => EPIC
  - Befehle evtl. mehrere Takte lang





- Besondere 80x86 Register
  - StackPointer, BasePointer
  - Stringops SourceIdx, Dest.Idx
  - CS, DS, SS, Extra-dataSegment

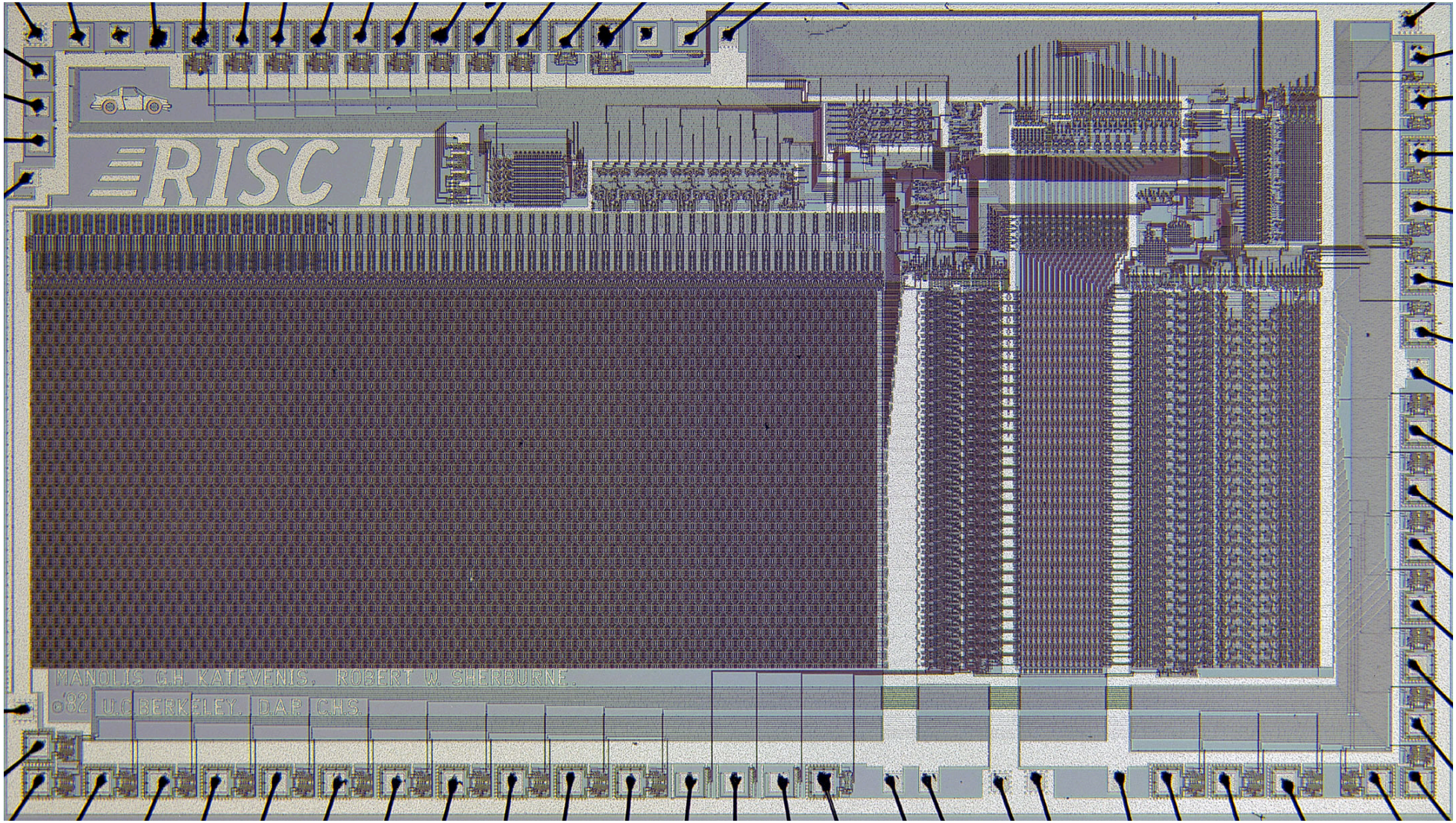
- Sprünge near und far
  - JMP, JMPF, JNZ, JZ
  - CALL, CALLF
  - RET, RETF
  - LOOP: CX--, JNZ 8 Bit Displacement
- Datentransfer
  - MOV reg-reg, reg-mem
  - PUSH, POP
- Arithmetik
  - ADD; SUB, CMP, …
  - SHx, RCR, INC, DEC, …
- Stringoperationen
  - MOVS
  - LODS
- Floating Point
  - Stackbasiert
  - 8 Stack-'Register'



4.4 Stichwort: RISC

- IBM 801
- A Case for the Reduced Instruktion Set Computer
  - David Patterson, UCB, 1980
  - pro Takt ein Befehl
  - Kompromisse bei den Befehlen
  - Load-Store Architektur
- Make the general case fast
  - Instruktionssatz-Optimierung
  - ein Befehl - ein Zyklus (≠Takt)
  - Pipeline
  - Branch-Delay-Slot(s) (siehe Pipeline)
- Einfache Konstruktion
  - reguläre Struktur
  - Registerbänke, Cache, ALUs, Multiplexer
  - wenig Verwaltung
  - leichte Migration auf neue Verfahren

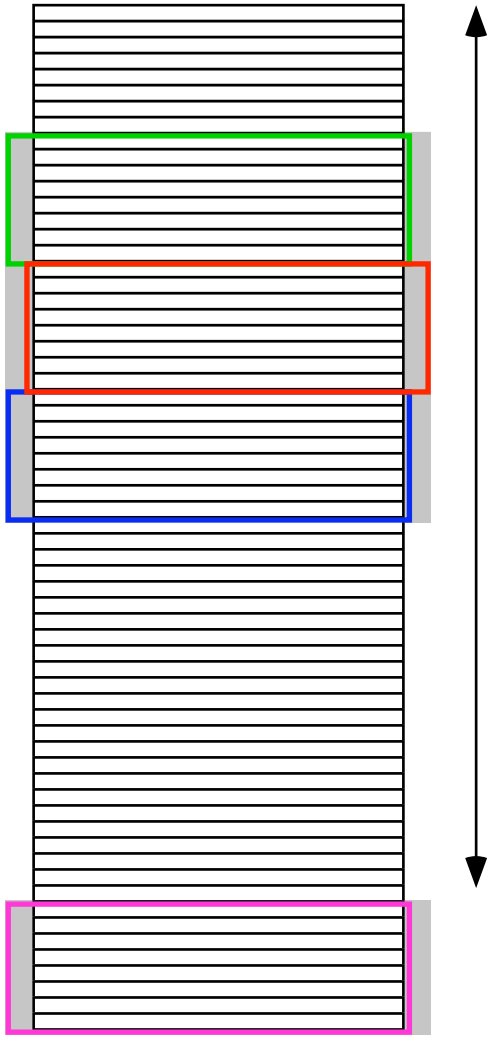
- RISC I/II [1982, 1983, Patterson et al, UCB]
  - 40,760 Transistoren, 3 micron NMOS, 60 mm^2, 3 MHz
  - Basis für Sun's SPARC



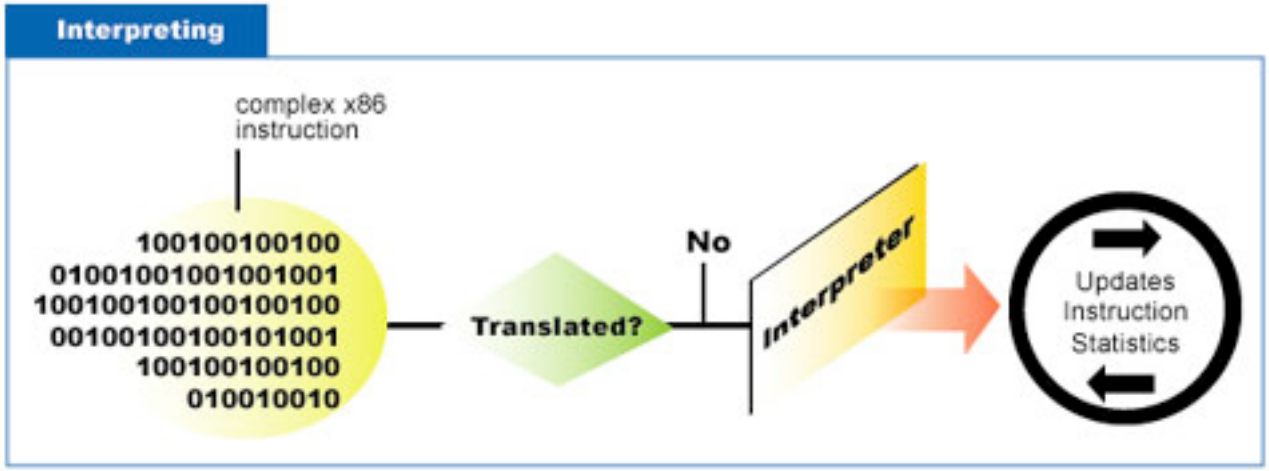
- MIPS [John Hennessy]
  - Universitätsprojekt, Stanford U
  - Microprocessor without Interlocked Pipeline Stages
  - später Mips Inc, dann SGI
  - R2000, …
  - SGI, Nintendo 64, Playstation, PS2
  - Drucker, Cisco-Router
- MIPS Kennzahlen
  - 32 Bit: Adresseraum, Register, Instruktionen
  - 31 GPRs und R0
  - 32*32 FPRegs (oder 16*64)
- MIPS IV Vergleich zu DLX
  - Indizierte Adressierung
  - Instruktionen für nicht-alignierte Daten
  - Multiply-Add
  - Unteilbares SWAP
  - 64 Bit Datentransfer und Arithmetik

- PowerPC
  - Motorola, IBM, Apple
  - Ideen der 801: Power 1, Power 2
  - PPC 601 (G1), 603 (G2), 604, 620 (64 Bit, MP)
  - G3, G4 < 1 GHz; G4 mit Vektoreinheit AltiVec
  - G5 > 1GHz
  - IBM-Gekko: Nintendo GameCube (Dolphin)
- PowerPC Kennzahlen
  - 32 Bit: Adresseraum, Register, Instruktionen
  - 32 GPRs
  - 32*32 FPRegs (oder 32*64)
- PowerPC Vergleich zu DLX
  - Fast alle Adressierungsarten
  - nicht-alignierte Daten möglich
  - Multiply-Add
  - Unteilbares SWAP
  - 8*4 Bit Condition-Codes (ähnlich Registerkonzept)
  - 64 Bit Instruktionen für 64 Bit Chips

- HP-Precision Architecture
  - erste RISC mit SIMD Multimediaerweiterungen
  - ShiftAdd statt Multiplikation, DS für Division
  - viele Conditional Branches
  - 32/48 Bit Adressen mit Segmentierung
- SPARC
  - max. 32*16 Register, klassisch 128
  - Register Windows für Prozeduren
  - Window-Größe 8+8+8
  - 8 für lokale Variable
  - 16 für Parameter: 8 In, 8 Out
  - 8 für globale Variablen
  - typisch 8 Register für Variablen und Parameter
  - Save/Restore um Registerfenster zu verschieben
  - V8: Tagged-Instruktionen für LISP und Smalltalk

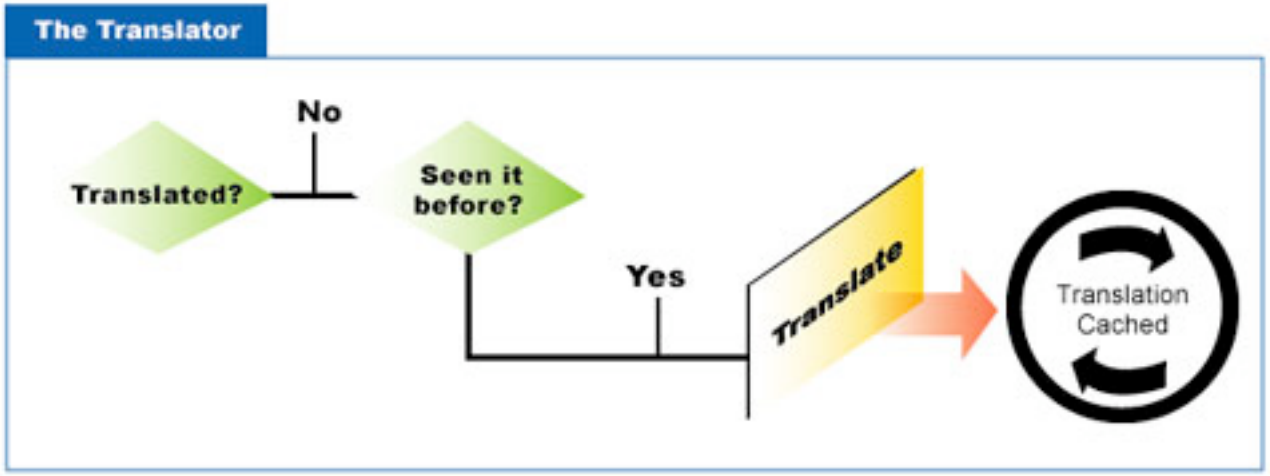


- Transmeta
- Apples Emulatorprojekt
  - Interpretation der 68LC040-Befehle
  - 40 Mhz 68040 auf 66 MHz PPC601
  - Übersetzung häufiger Instruktionssequenzen
- IBM-Projekt zur Emulation des 80386

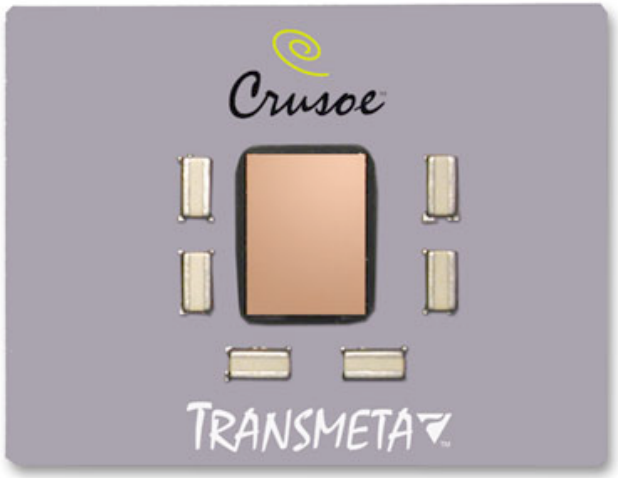


- VLIW: Very Large Instruktion Word
  - mehrere Instruktionen pro Instruktion Fetch
  - Unabhängigkeit?

- Häufig ausgeführte Sequenzen besonders behandeln
  - Instruktions-Statistik
  - Code Profiling => Optimierung
  - dynamischer Übersetzer (code morphing)
  - übersetzte Blöcke cachen



- Crusoe
  - 64 Register a 32 Bit
  - 4 Funktionseinheiten zur parallelen Befehlsausführung
  - Java Mode



- Intel-Code

```
addl  %eax,(%esp) // load data from stack, add to %eax
addl  %ebx,(%esp) // ditto, for %ebx
movl  %esi,(%ebp) // load %esi from memory
subl  %ecx,5      // subtract 5 from %ecx register
```

- 1. Übersetzungslauf

```
ld    %r30,[%esp]    // load from stack, into temporary
add.c %eax,%eax,%r30 // add to %eax, set condition codes.
ld    %r31,[%esp]
add.c %ebx,%ebx,%r31
ld    %esi,[%ebp]
sub.c %ecx,%ecx,5
```

- 2. Übersetzungslauf (klassische Compilertechnik)
  - gemeinsame Teilausdrücke
  - Schleifeninvarianten
  - unbenutzer Code

```
ld    %r30,[%esp]    // load from stack only once
add   %eax,%eax,%r30
add   %ebx,%ebx,%r30 // reuse data loaded earlier
ld    %esi,[%ebp]
sub.c %ecx,%ecx,5    // only this last condition code needed
```

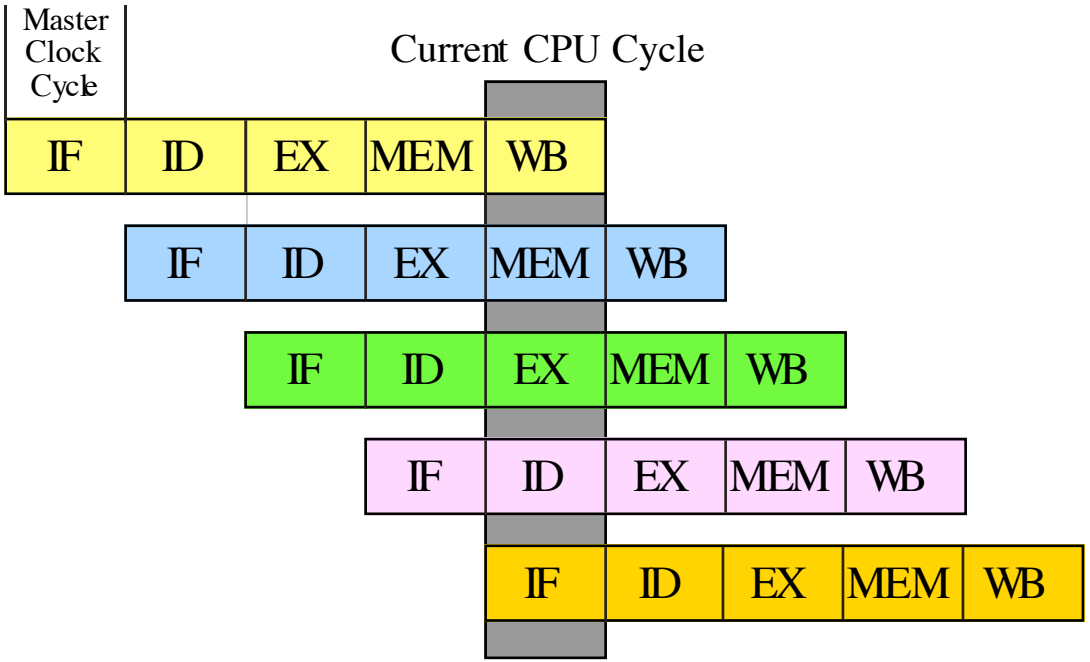
- 3. Übersetzungslauf
  - Befehle umordnen
  - Befehle auf parallele Funktionseinheiten verteilen
  - Software

```
1. ld %r30,[%esp];   sub.c %ecx,%ecx,5
2. ld %esi,[%ebp];   add %eax,%eax,%r30;  add %ebx,%ebx,%r30
```

- ARM (Advanced RISC Machine, eigentlich Acorn RISC Machine)
  - 32 Bit
  - 2007: 98% aller verkauften Mobiltelefone enthalten ARM-Variante
  - 2009: 90% aller embedded Prozessoren
  - IP: Qualcomm, TI, Intel, Freescale, Apple, STM, …
  - aktuell: Cortex A8, A9, A9 MPCore
- Eigener Instruktionssatz
  - 16 Register
  - Instruktionen 32 bit
  - conditional execution: SUBLT R5, R5, R3
  - shift und rotate an Instruktionen "anhängen": ADD R3, R3, R5, LSL #1
- Thumb-mode
  - 16 bit Instruktionsworte
  - weniger Optionen im Befehl
  - nicht alle Register spezifizierbar
  - höhere Codedichte und/oder nur 16 Bit Speicherinterface
  - z.B. Game Boy Advance

5. Pipelines

- Fließbandarbeit
  - Instruktionen haben Teilschritte
  - parallele Ausführung der Teilschritte verschiedener Instruktionen
  - Weitergabe der Zwischenergebniss an nächste Stufe (≠MIMA)

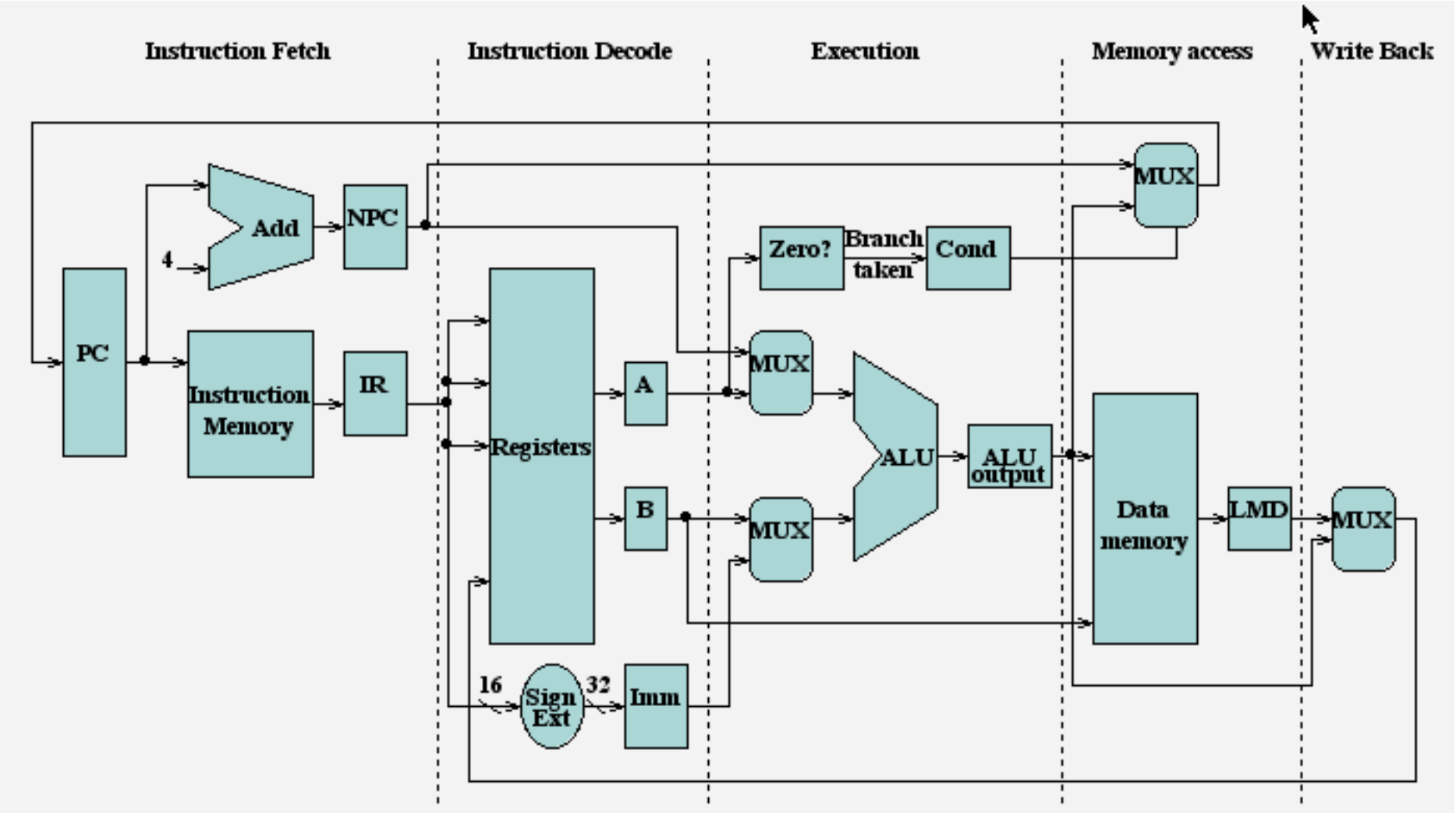


- 1 Instruktion
  - n Maschinen-Zyklen insgesamt
  - pro Maschinen-Zyklus 1 Instruktion fertig
  - m Takte = 1 Maschinen-Zyklen; m<=4

- Unabhängigkeit der Einzelschritte in der Pipeline
  - Entkopplung der Stufen
  - genug Verarbeitungseinheiten (ALU + 1 Addierer für Adressen)
- Pipelinetakt
  - bestimmt durch längsten Teilschritt
  - z.B. Speicherzugriff oder Alu-Operation
  - Länge der Einzelschritte ausbalancieren
- Pipeline-Register entkoppeln Stufen
  - Pipeline-Latches
  - Stufe n-1 schreibt Resultat spät im Zyklus
  - Stufe n nimmt Input früh im Zyklus
- Speicher
  - Resultate in einem Zyklus
  - getrennte Speicher/Caches für Instruktion und Daten?
- Pipeline-Risiken
  - Verzweigung, Sprung
  - Operandenbezüge
  - Resource-Mangel (z.B. ALU, Registerbank)

  => Pipeline-Stall

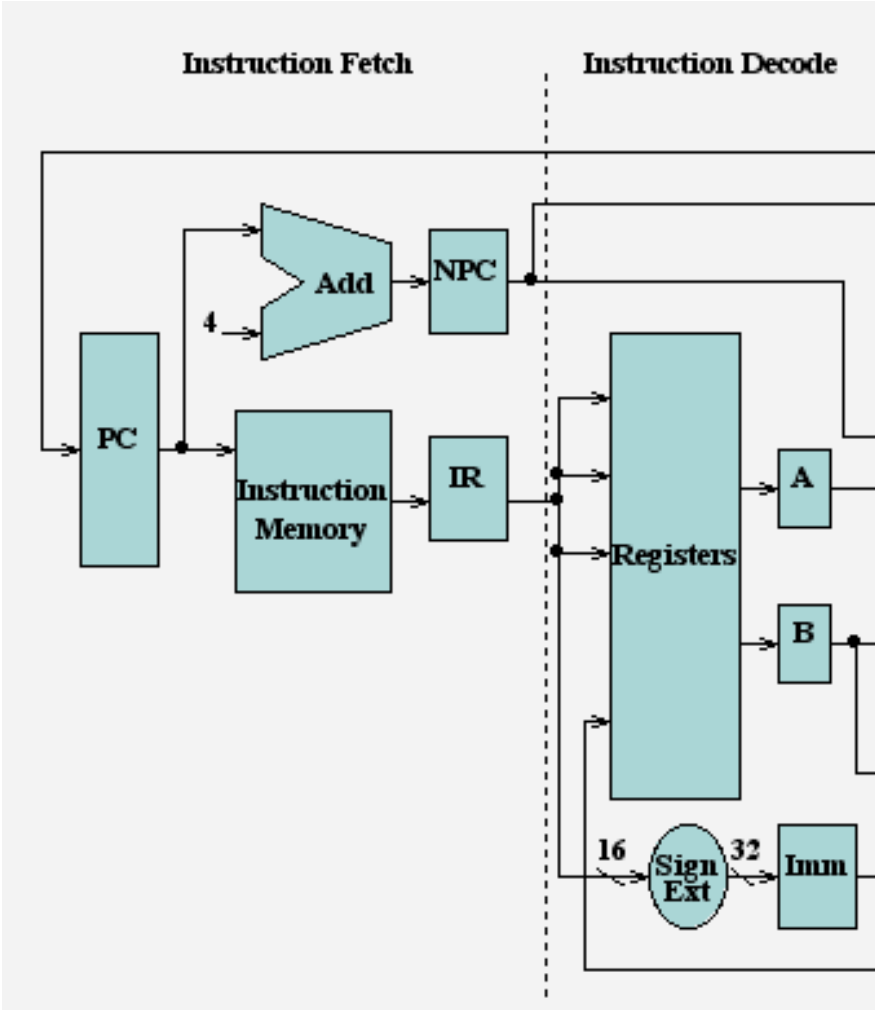
- DLX-Pipeline



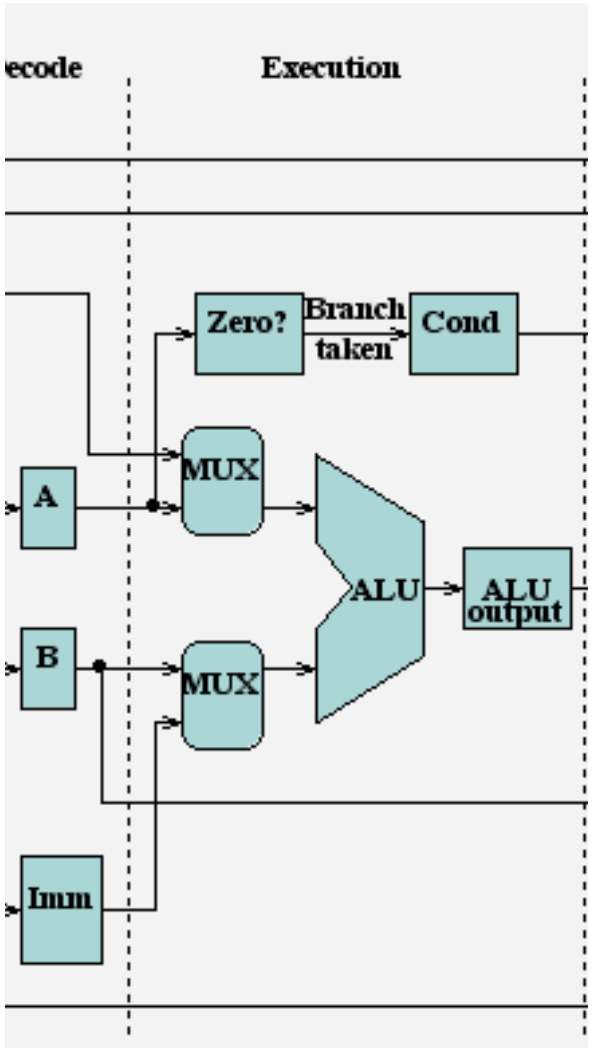
- Steuerung sorgt für unabhängige Ausführung
  - 'Durchreichen' des Opcodes und der Operanden

5.1 Instruktionen bearbeiten und ausführen

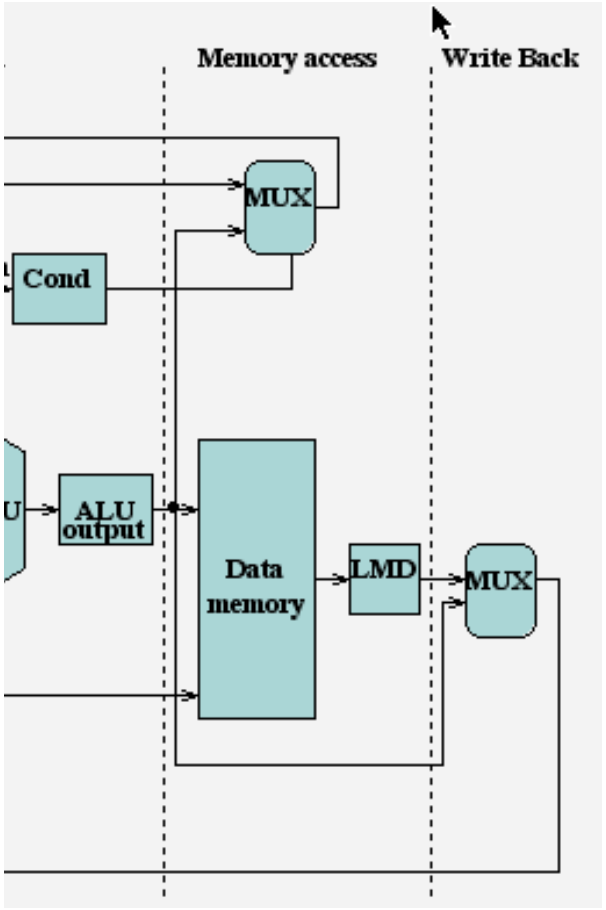
- Schönes Java-Applet
  http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/DLXimplem.html
- Instruction fetch cycle (**IF**)
  - `IR := Mem[PC]`
  - `NewPC := PC+4`
- Instruction decode/register fetch (**ID**)
  - Dekodierung: welcher Befehl
  - Operanden aus der Register-Bank in die Eingangsregister der ALU
  - $A := Regs[IR_{6..10}]$
  - $B := Regs[IR_{11..15}]$
  - $Imm := (IR_{16})^{16} \,\#\#\, IR_{16..31}$
  - Festes Instruktionsformat: Parallelität



- Execution/Effective address cycle (**EX**)
  - Berechnung
  - Ausgangsregister der ALU setzen (+ CC)
  - Load/Store: EA im ALU-Output-Reg
  - `ALUOut := A + Imm`
  - Register-Register: Resultat der Operation
  - `ALUOut := A` *`func`* `B`
  - Register-Immediate: Resultat der Operation
  - `ALUOut := A` *`func`* `Imm`
  - Control-Xfer: ALU berechnet Sprungziel
  - `ALUOut := NPC + Imm`
  - `Cond := A` *`op`* `0`



- Memory access/branch completion cycle (**MEM**)
  - nur bei Load/Store ausgeführt
  - Load: Daten von MEM[ALU-Output] holen
  - `LMD := MEM[ALUOut]`
  - Store: Daten nach MEM[ALU-Output] schreiben
  - `MEM[ALUOut]:= B`
  - Control-Xfer: bei ausgeführten Sprüngen PC schreiben
  - `if Cond then PC:=ALUOut`
    `else PC:=NPC`
- Write-back cycle (**WB**)
  - leer falls Control-Xfer
  - Resultat in die Registerbank übertragen
  - Load
  - $\texttt{Regs[IR}_{11..15}\texttt{]} := \texttt{LMD}$
  - Register-Immediate
  - $\texttt{Regs[IR}_{11..15}\texttt{]} := \texttt{ALUOut}$
  - Register-Register
  - $\texttt{Regs[IR}_{16..20}\texttt{]} := \texttt{ALUOut}$



5.2 Pipeline Hazards

- Konflikte in Architektur oder Abhängigkeiten
- Cache-Miss
  - Pipeline anhalten
  - Warten bis Daten angekommen sind
- Hazard
  - konkurrierender Zugriff auf Ressourcen oder Daten
  - keine neuen Instruktionen starten
- Pipeline-Bubble (oder Stall)
  - Beispiel nur ein Instruktions/Datencache

	1	2	3	4	5	6	7	8	9	10
Instr i	IF	ID	EX	MEM	WB					
Instr i+1		IF	ID	EX	MEM	WB				
Instr i+2			IF	ID	EX	MEM	WB			
Stall				bubble	bubble	bubble	bubble	bubble		
Instr i+3					IF	ID	EX	MEM	WB	
Instr i+4						IF	ID	EX	MEM	WB

- Alternative Darstellung

Instr	1	2	3	4	5	6	7	8	9	10
Instr i	IF	ID	EX	MEM	WB					
Instr i+1		IF	ID	EX	MEM	WB				
Instr i+2			IF	ID	EX	MEM	WB			
Instr i+3				stall	IF	ID	EX	MEM	WB	
Instr i+4						IF	ID	EX	MEM	WB

- Anderer Fall

Instr	1	2	3	4	5	6	7	8	9	10
Instr i	IF	ID	EX	MEM	WB					
Instr i+1		IF	ID	stall	EX	MEM	WB			
Instr i+2			IF	stall	ID	EX	MEM	WB		
Instr i+3				stall	IF	ID	EX	MEM	WB	
Instr i+4						IF	ID	EX	MEM	WB

5.2.1 Structural Hazards

- Nur ein Schreibeingang in das Register-file
  - 2 Schreibzugriffe in einem Zyklus
- Nur ein Cache (Bus) für Daten und Befehle
  - Load.MEM kollidiert mit Inst3.IF

Instr	1	2	3	4	5	6	7	8
Load	IF	ID	EX	MEM	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				IF	ID	EX	MEM	WB

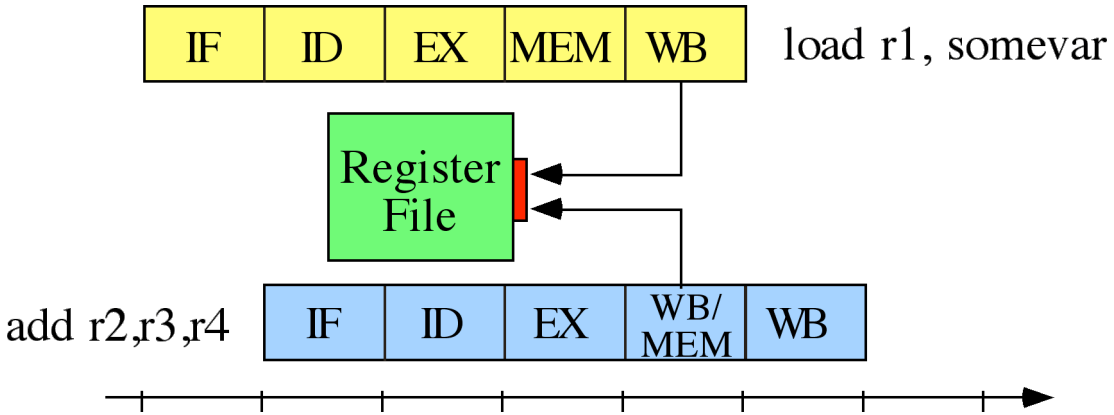
  - Inst3.IF verschieben

Instr	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	**MEM**	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Stall				bubble	bubble	bubble	bubble	bubble	
Instr 3					**IF**	ID	EX	MEM	WB

- Annahme:
  - MEM-Stufe kann ALU-Resultat ins Register-File schreiben
  - nur ein Write-Port

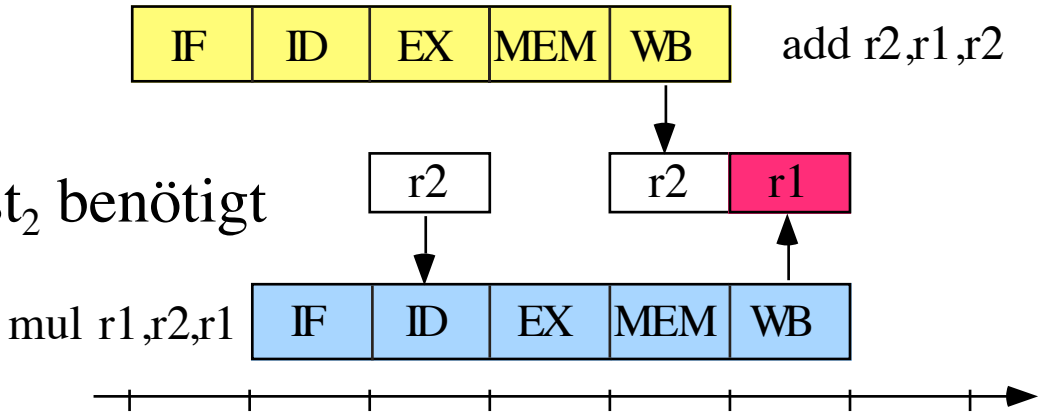
```
load    r1, somevar
add     r2,r3,r4
```

  - Daten für r1 und r2 kommen gleichzeitig am RegisterFile an



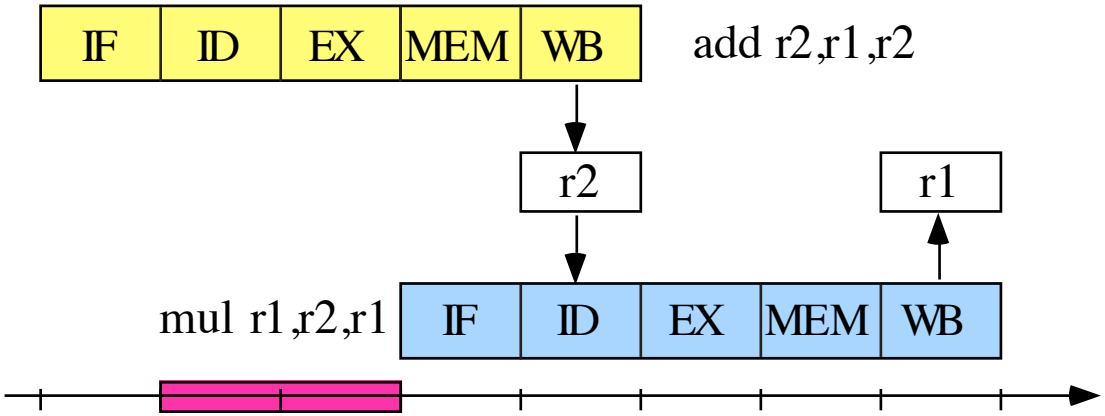
- Hardware entdeckt Konflikt und fügt bubble ein
- Resource Replication: 2 write ports
  - weitere Abhängigkeiten erzeugt?
  - weitere Überwachung und Interlocking

5.2.2 Data Hazards

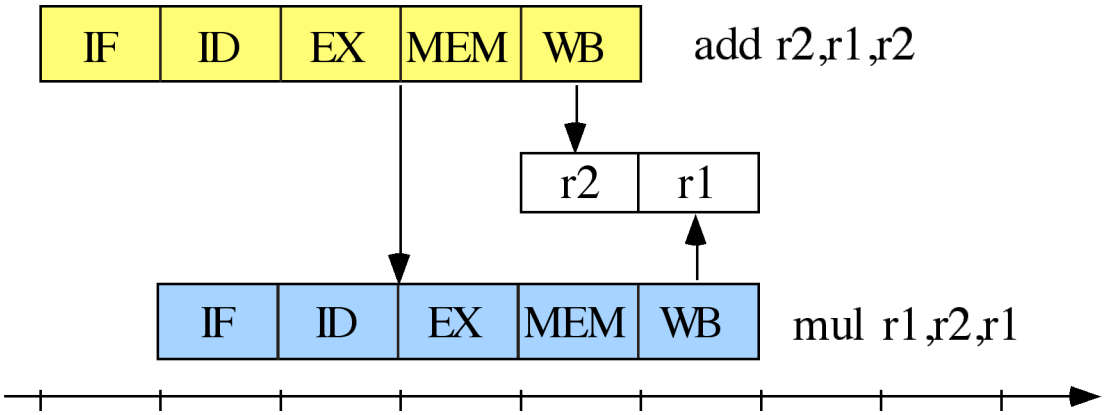


- Echte Abhängigkeit
  - $Inst_1$ schreibt in das Register, das $Inst_2$ benötigt
  - auch im Speicher!
- Read After Write (RAW)
  - $Instr_2$ liest Operand, bevor $Instr_1$ ihn schreibt
- Write After Read (WAR)
  - $Instr_2$ schreibt Operand, bevor $Inst_1$ ihn liest
  - nur bei superskalaren Architekturen
- Write After Write (WAW)
  - $Instr_2$ schreibt Operand, bevor $Inst_1$ ihn schreibt
  - in komplexeren Pipelines als DLX
- Namens-Abhängigkeit
  - anti-abhängig: $Inst_1$ liest Register, das $Inst_2$ später überschreibt
  - Ausgabe-abhängig: $Inst_1$ schreibt Reg., das $Inst_2$ danach überschreibt
  - keine Daten von $Inst_1$ and $Inst_2$ übergeben
  - Implementierungsproblem

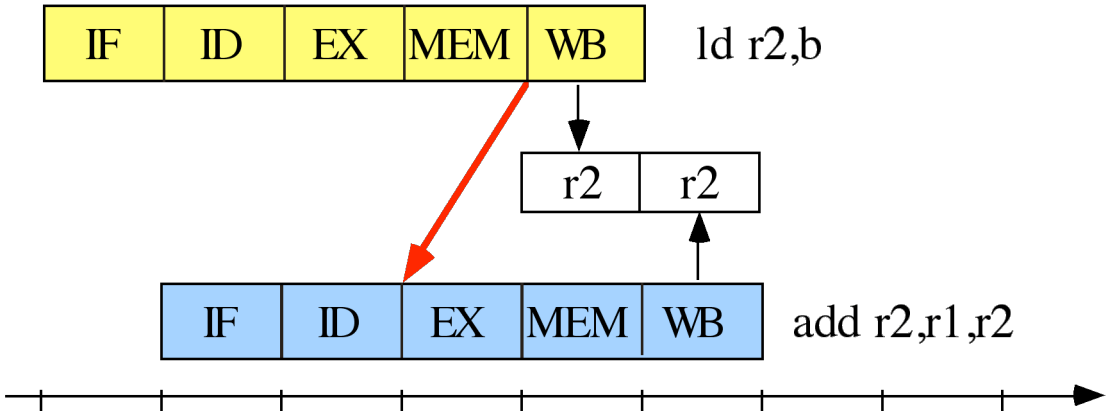
- Lösungen
- Compiler erzeugt geeigneten Code
  - NOP(s) einfügen (Software bubble)
  - Instruktionen umordnen
- Hardware
  - Schaltung um Konflikt zu erkennen
  - Pipelinesteuerung
- Interlocking
  - Pipeline anhalten (stall)
  - evtl. mehrere Zyklen



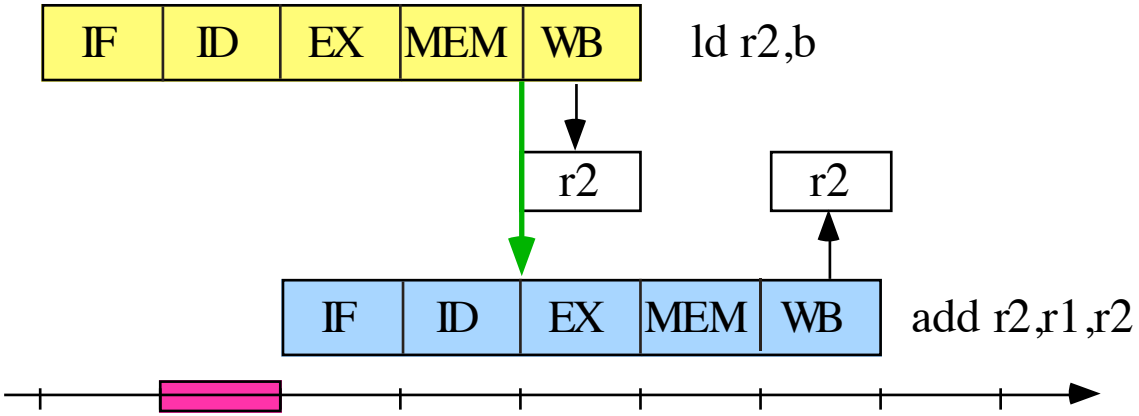
- Forwarding
  - Abkürzungen in der Pipeline
  - ALU-Resultat von $Inst_{EX}$ direkt an ALU-Input für $Inst_{ID}$ kopieren



  - LMDR von $Inst_{MEM}$ auch an ALU-Input für $Inst_{ID}$ kopieren
- Forwarding nicht immer möglich



- Forwarding mit Interlocking
  - $Inst_2$ datenabhängig (RAW) von $Inst_1$



  - bubble kann nicht ganz aufgelöst werden
- Scoreboard entdeckt Abhängigkeiten

5.2.3 Control Hazards

- Sprung-Instruktionen (bedingt und fest)
  - Spungziel muss berechnet werden
  - Bubbles einfügen bis Sprung MEM ausgeführt hat
  - Sprung erst in ID festgestellt
  - IF während Sprung-ID oft wertlos

IF	ID:Sprung	EX	MEM	WB					
	IF->stall	stall	stall	IF	ID	EX	MEM	WB	
					ID	EX	EX	MEM	WB

- 3 Zyklen pro Spung verloren
  - Verzweigungen ca 20%

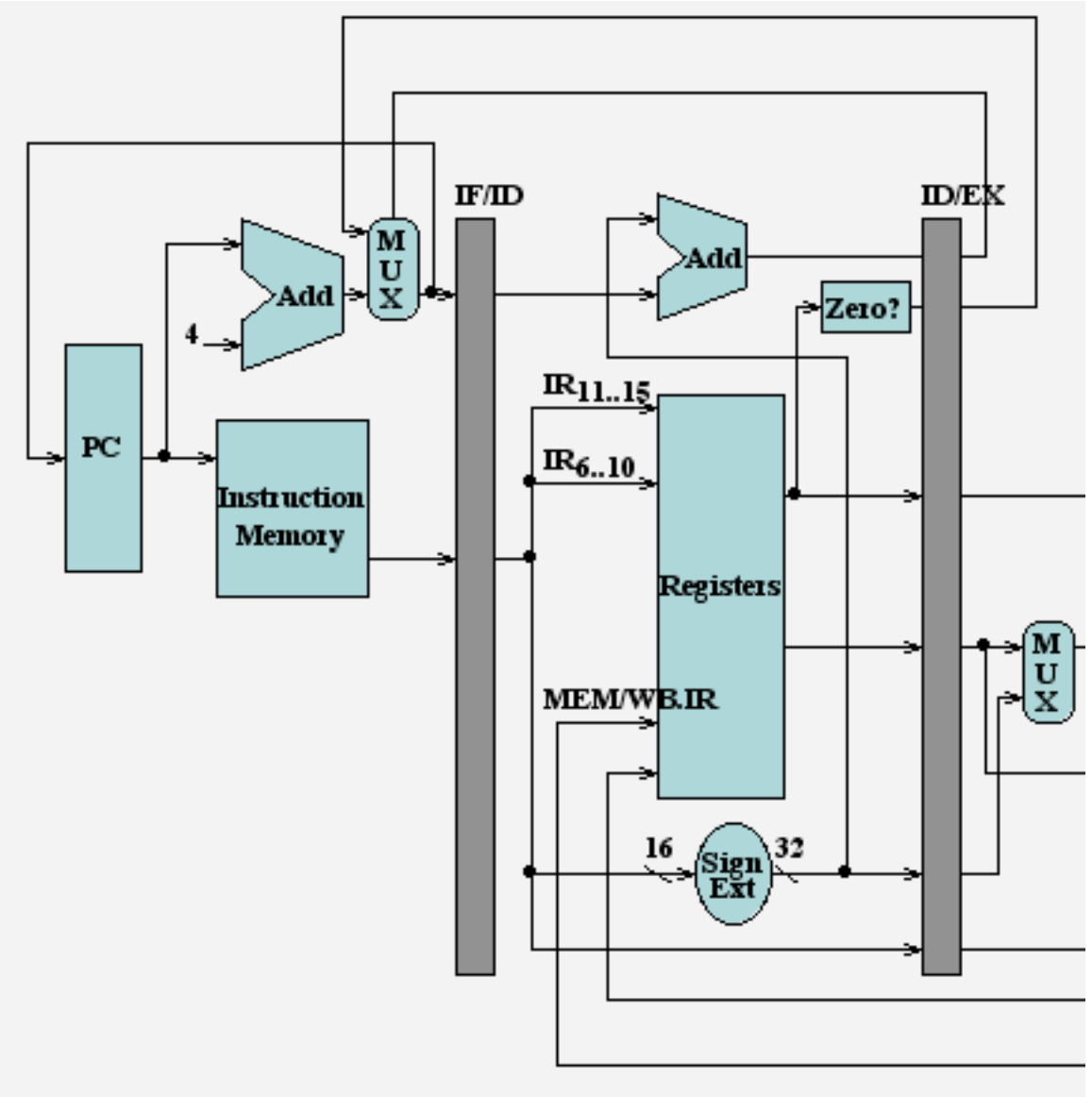
  => 1,6 CPI (cycles per instruction)
- Abhilfe
  - schon in IF feststellen ob gesprungen wird
  - neuen PC früher berechnen

- Pipeline umbauen
  - Addierer für Branch in ID
  - Zugriff auf Reg früh
  - Addieren spät
  - Condition auswerten spät
  - nur ein Stall

IF	ID	EX
	IF->stall	IF

- Bedingte Sprünge
  - einfacher Test auf 0
  - einfach ausführbar
  - Data hazard?

```
SUBI   r3,r3,1
BNEZ   r3,loop1
```



- Wie oft werden Sprünge ausgeführt?
  - 17% bedingte Sprünge
  - 85% der Rückwärts-Sprünge ausgeführt (Schleifen)
  - 60% der Vorwärts-Sprünge ausgeführt (if/then/else)
  - 67% aller Sprünge werden ausgeführt
- Triviale Vorhersage: Sprung wird nicht genommen
  - predict-not-taken
  - einfach weitermachen, als ob nicht gesprungen wird
  - permanenten Zustand (Register!) nicht verändern
  - falls doch gesprungen wird: IF/ID latches löschen
  - immerhin 33% Gewinn

IF	ID:Sprung	EX	MEM	WB					
	IF	ID/leer	EX/leer	MEM/leer	WB/leer				
		IF/IF	ID	EX	MEM	WB			
			IF	ID	EX	MEM	WB		
				IF	ID	EX	MEM	WB	
					ID	EX	EX	MEM	WB

5.2.4 Exceptions

- I/O, OS-Call, …
- Fehler, Page-Fault, …
- Breakpoints
- Beispiel: Page Fault bemerkt in der MEM-Stufe
  - Betriebssystem tauscht Seite
  - Wiederaufnahme
  - was passiert mit der Instruktion in der WB-Stufe?
- Unterbrechung in DLX
  - TRAP-Instruktion in IF/ID latches schreiben
  - Schreiboperationen der (Folge-)Instruktion unterbinden
  - Unterbrechung rettet PC
  - PC restaurieren mit RFE
- Problem: Delay Slot

5.3 Pipelinebeschleunigung

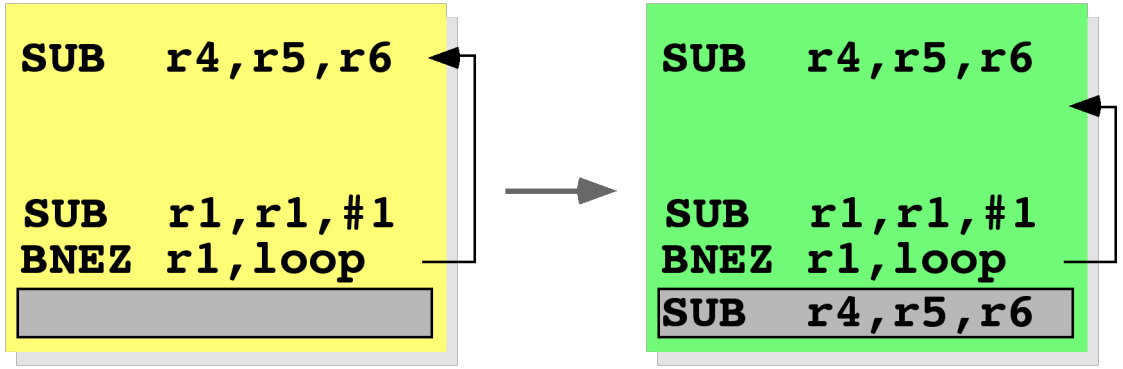
5.3.1 Softwarelösungen

- m Instruktionen nach dem Branch werden immer ausgeführt
  - kann der Compiler oft sinnvoll füllen
  - Programmierer finden fast immer etwas

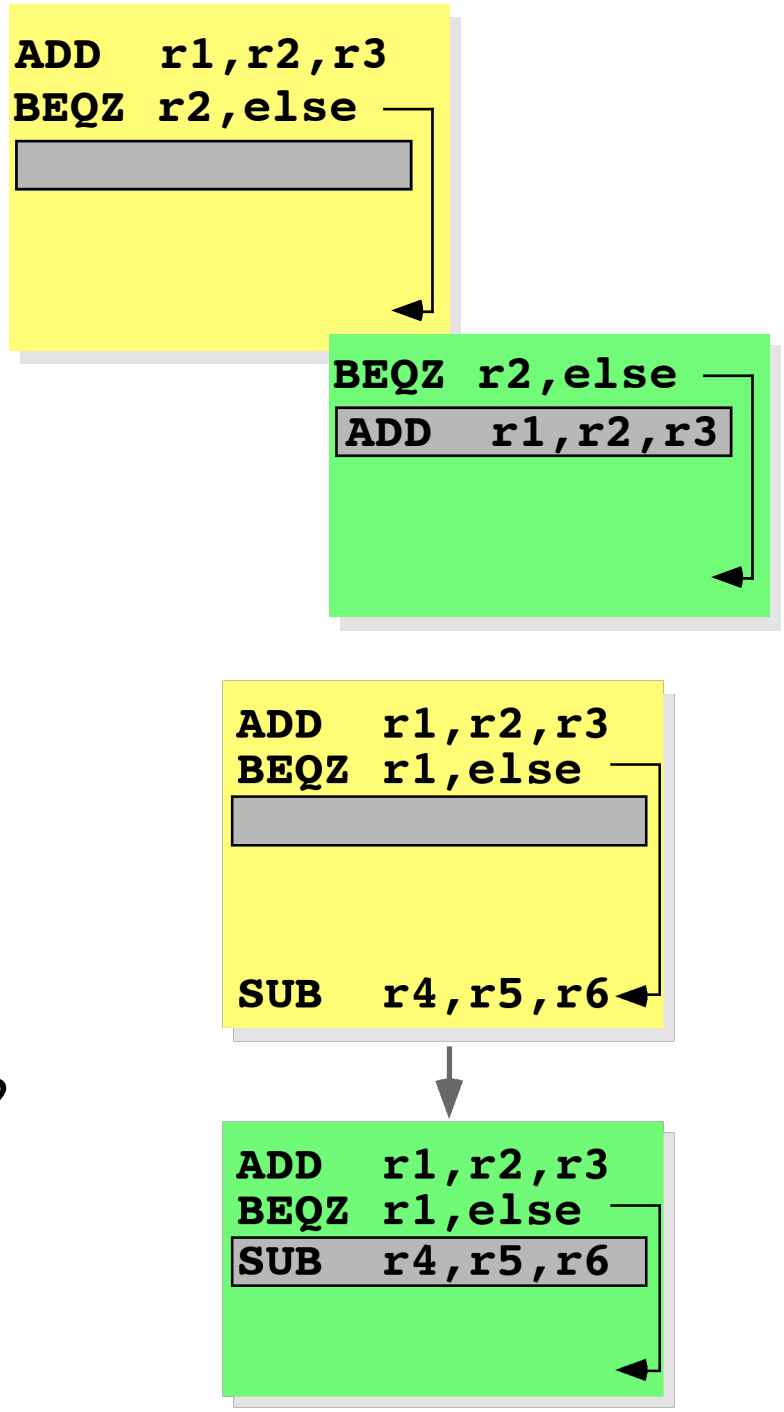
Inst i	IF	ID	EX	MEM	WB				
Inst i+1 (delay-slot)		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EX	MEM	WB	
Inst i+4					IF	ID	EX	MEM	WB

Inst i	IF	ID	EX	MEM	WB				
Inst i+1 (delay-slot)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target+1				IF	ID	EX	MEM	WB	
Branch target+2					IF	ID	EX	MEM	WB

- Compiler plant für den Delay-Slot
- Instruktion von vorn in den slot schieben
  - Sprung unabhängig von der Instruktion
- Instruktion vom Ziel vorziehen
  - muss evtl. kopiert werden



  - einmal zuviel berechnet
  - Ergebnis korrigieren?
  - Ergebnis hinter der Schleife nicht benutzen
- Nachfolge-Instruktion vorziehen
  - Register in der Sprung-Region unverändert?
- Auch dynamisch in Hardware



- Loop Unrolling
  - Spünge sind Pipeline-Feinde
  - häufig am Ende von Schleifen
  - Anzahl Sprünge pro Durchlauf reduzieren

```
for (i=0;i<1000;i++)
x[i]=x[i] + y [i];
```

```
i=0;
while (i<1000) {
x[i]=x[i] + y [i];
x[i+1]=x[i+1] + y [i+1];
x[i+2]=x[i+2] + y [i+2];
x[i+3]=x[i+3] + y [i+3];
i+=4;}
```

  - reduziert auch Branch-Delay-Slots
- Schleifenzähler
  - n Schleifendurchläufe
  - Ausrollfaktor k
  - Hauptschleife n DIV k
  - Aufräumschleife n MOD k
  - Autoinkrement?

Pipeline Scheduling im Compiler

- Code wird für Prozessor optimiert
  - Eigenschaften der Prozessor-Pipeline bekannt (exposed)
  - delay-slot(s)
  - Resource Hazards
  - eigentlich schlecht: spätere Verbesserungen nicht nützlich
- Instruktionen umsortieren
  - besser, je mehr Instruktionen zur Auswahl
  - Schleifen ausrollen
  - datenabhängige Instruktionen auseinanderziehen
  - Branch-Slots füllen
- Datenabhänigigkeit
  - Datenfluß zwischen Instruktionen: RAW
  - statische Abhängigkeit
- Namensabhängigkeit
  - Antiabhängigkeit kann zu WAR führen
  - Ausgabe-Abhängigkeit kann zu WAW führen
  - Register-Renaming (andere Register verwenden)

- Kontroll-Abhängigkeit
  - Codeblock durch IF bewacht
  - welche Instruktionen herausziehen bzw. hineinschieben?
- Exception-Verhalten

```
      BEQZ    R2, L1
      LW      R1, 0(R2)
L1:
```

  - Exception ignorieren falls Sprung gemacht wird
- Datenfluß
  - dynamische Abhängigkeit
  - Spekulation

Datenfluß-Abhängigkeit (R1)	R4 später nicht verwendet	R4 spekulativ früh berechnen
`ADD R1,R2,R3`	`ADD R1,R2,R3`	`SUB R4,R5,R6`
`BEQZ R4,L`	`BEQZ R12,L`	`ADD R1,R2,R3`
`SUB R1,R5,R6`	`SUB R4,R5,R6`	`BEQZ R12,L`
`;stall`	`;stall`	`ADD R5,R4,R9`
`L: OR R7,R1,R8`	`ADD R5,R4,R9`	`L: OR R7,R8,R9`
	`L: OR R7,R8,R9`	

- Beispiel: for (i=0;i<1000;i++) x[i]=x[i] +s;
- DLX-Code einfach

```
loop:   LD     F0,0(R1)
        ADDD   F4,F0,F2      ; s in F2
        SD     0(R1),F4
        SUBI   R1,R1,#8      ; double word
        BNEZ   R1,loop
```

- Ausführung mit DLX-Pipeline

```
loop:   LD     F0,0(R1)
        stall
        ADDD   F4,F0,F2      ; 3 Zyklen!
        stall
        stall
        SD     0(R1),F4
        SUBI   R1,R1,#8      ; double word
        stall
        BNEZ   R1,loop
        stall
```

- Instruktionsreihenfolge ändern

```
loop:  LD     F0,0(R1)
       SUBI   R1,R1,#8       ; double word
       ADDD   F4,F0,F2       ; 2 Zyklen
       stall
       BNEZ   R1,loop
       SD     8(R1),F4       ; Instruktion verändert
```

  - SD eigentlich abhängig von SUBI
  - SD-Instruktion verändern
  - 6 Zyklen pro Durchlauf statt 10

- Loop unrolling

```
loop:  LD     F0,0(R1)
       ADDD   F4,F0,F2
       SD     0(R1),F4
       LD     F0,-8(R1)
       ADDD   F4,F0,F2
       SD     -8(R1),F4
       LD     F0,-16(R1)
       ADDD   F4,F0,F2
       SD     -16(R1),F4
       LD     F0,-24(R1)
       ADDD   F4,F0,F2
       SD     -24(R1),F4
       SUBI   R1,R1,#32
       BNEZ   R1,loop
```

  - Adressen in Instruktionen verändert
  - 28 Zyklen = 7 Zyklen pro Element
  - 14 Inst + 4 LD-stall + 1 SUBI -stall, 8 ADDD-stalls + branch-delay

- Schleife planen (scheduling)

```
loop:  LD     F0,0(R1)
       LD     F6,-8(R1)
       LD     F10,-16(R1)
       LD     F14,-24(R1)
       ADDD   F4,F0,F2
       ADDD   F8,F6,F2
       ADDD   F12,F10,F2
       ADDD   F16,F14,F2
       SD     0(R1),F4
       SD     -8(R1),F8
       SUBI   R1,R1,#32
       SD     -16(R1),F12   ;R1 noch nicht geändert
       BNEZ   R1,loop
       SD     8(R1),F16     ;R1 fertig: 8=32-24
```

  - 14 Zyklen = 3.5 Zyklen pro Element
  - ohne stalls

5.3.2 Hardwarelösungen

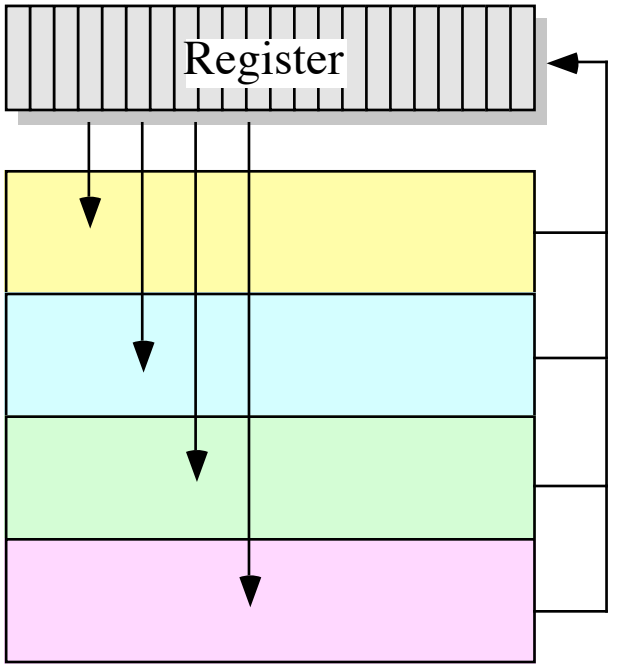
- Mehrere Ausführungseinheiten
- Pipeline-Eigenschaften nicht in das Programm einbauen
  - Prozessor kann verbessert werden
  - schnelle Ausführung alten Codes
  - CISC auf RISC
- Beispiel Pentium Pro und später
  - 80x86-Instruktion auf eine oder mehrere µOperationen aufteilen

1 µOperation	`mov eax,[es:0x10]`	`Wert laden`
2 µOperationen	`mov [es:0x10],eax`	`berechne Zieladresse`
		`speichern`
4 µOperationen	`add [es:0x10],eax`	`Wert laden`
		`addieren`
		`berechne Zieladresse`
		`speichern`

  - µOperationen im Instruktionspool abgelegt
  - dynamisch planen

5.3.2.1 Dynamic Scheduling

- Instruction-Decode entscheidet über Ausführung
  - Warteschlange mit Befehlen
  - structural Hazards auswerten
  - nächsten möglichen Befehl starten
  - auf Functional-Units verteilen
  - evtl. dynamisch umordnen
- Instruktions-Resultat-Reihenfolge geändert
  - WAR, WAW können entstehen
  - Unprecise Exceptions?
  - Planung abhängig von Registern
  - Instuktionen können lange in FU hängen
- Modifikation der DLX-Pipeline
  - DLX-ID prüft structural H. und data H.
  - Neu: ID aufteilen in *Issue* und *Read Operands*
  - Warteschlange zwischen Fetch und Issue
  - DLX: Nur bei Floating Point Unit
  - 1 Integer, 2 FP-MULs, 1 FP-DIV, 1 FP-ADD



- Beispielproblem

```
DIVD   F0,F2,F4      ; viele Zyklen
ADDD   F10,F0,F8
SUBD   F12,F8,F14    ; WAW: SUBD  F10,F8,F14
                     ; WAR: SUBD  F8,F8,F14
```

  - SUBD vorziehen
- Registerverwaltung
  - Auflösung der data hazards
  - Warten auf Daten
  - Warten beim Speichern in Register
- Scoreboard
  - startet Instruktionen (issue) in EX[i]
  - gibt Register als Operanden frei (RAW)
  - stellt fest, dass EX[i] ist fertig
  - überprüft ob WB Register schreiben kann (WAR, WAW)
- Scoreboard-Datenstrukturen
  - Instruktions-Status: issued, read, execution complete, write result
  - Funktionseinheit-Status
  - Register Resultat Status

- Scoreboard-Tabellen [CDC 6600, 1964]
  - Schnappschuss für DLX-Pipeline

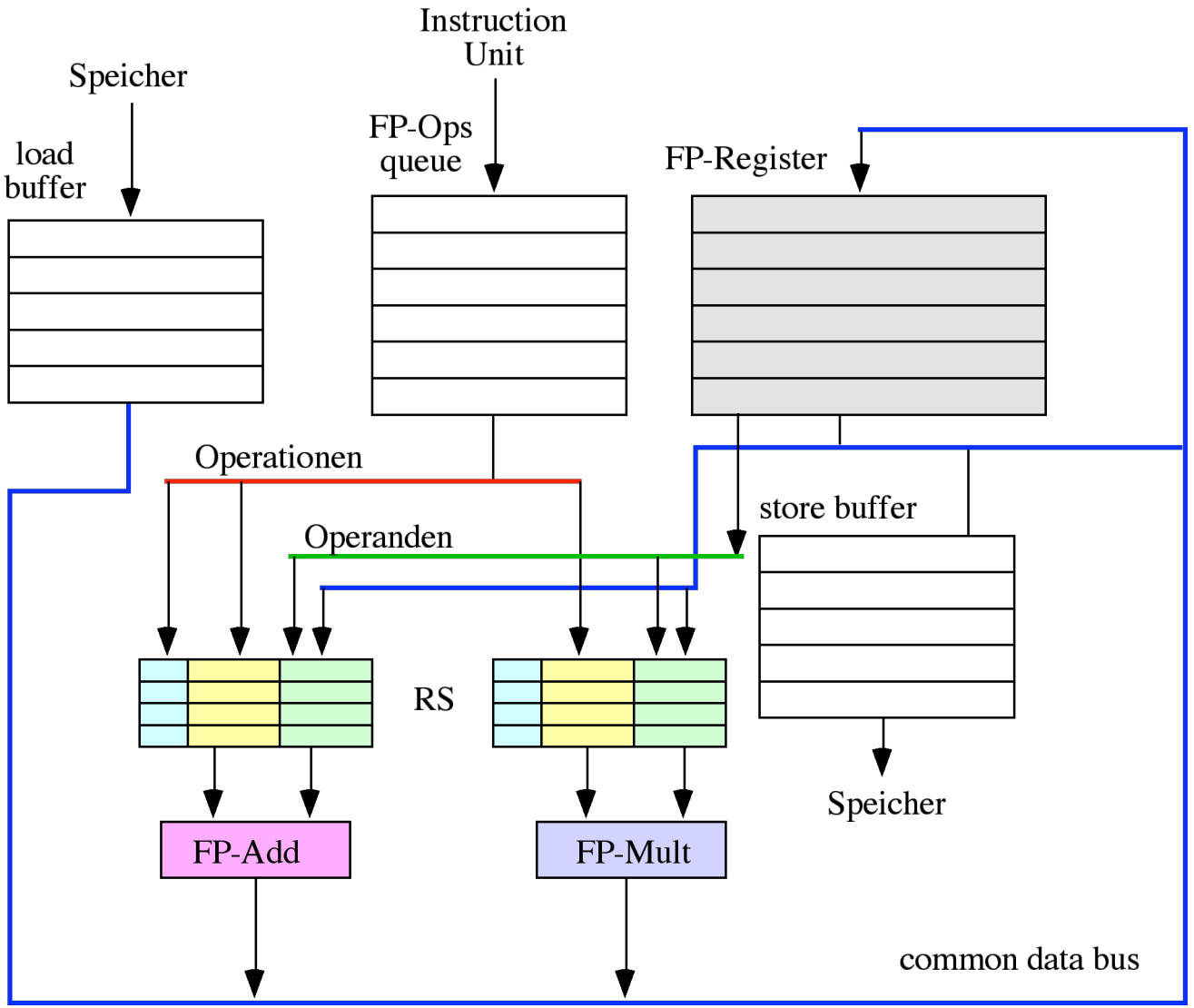
Instruktion	issued	read ops	exec. comp.	write back
`LD F6,32(R2)`	true	true	true	true
`LD F2,48(R3)`	true	true	true	
`MULTD F0,F2,F4`	true			
`SUBD F8,F6,F2`	true			
`DIVD F10,F0,F6`	true			
`ADDD F6,F8,F2`				

Funktionseinheit	busy	Op	RegD	RegS1	RegS2	FU1	FU2	rdy1	rdy2
Integer	true	Load (2)	F2	R3				false	
FMult1	true	Mult	F0	F2	F4	Integer		false	true
FMult2	false								
FAdd	true	Sub	F8	F6	F2		Integer	true	false
FDiv	true	Div	F10	F0	F6	FMult1		false	true

Register	F0	F2	F4	F6	F8	F10	F12	…	F30
Funktionseinheit	FMult1	Integer			FAdd	FDiv			

- Register Renaming
  - eigentlich: weitere Register verwenden
  - Register nicht unbedingt im Programmiermodell
  - vermeidet 'unechte Hazards': WAR und WAW
  - wirkt nicht bei Datenfluss-Hazards
- Tomasulu [Robert Tomasulu, 1967]
  - FPU für IBM 360/91
  - 'verteiltes' Scoreboard
  - Common Data Bus (CDB)
  - implizites Register-Renaming mit 'Reservation Stations'
  - aktive Register
  - Load und Store Puffer
- Reservation Station (RS)
  - Puffer vor Funktionseinheit
  - Operation und Operanden
  - Register beim Eintrag in RS mit anderer RS verbinden
  - verhindert WAR und WAW
  - aktiver Puffer: besorgt Operanden vom CDB (ähnl. forwarding)
  - ohne Register-'Umweg'

- Tomasulu-DLX-FPU



5.3.2.2 Spekulative Ausführung

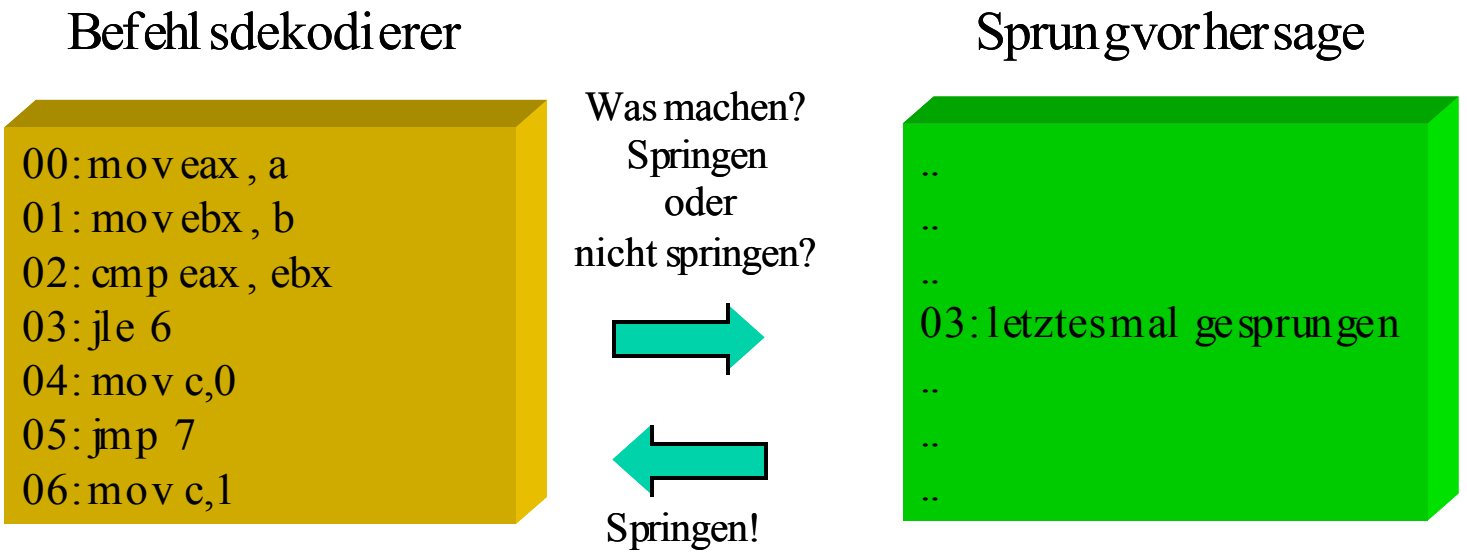
- Kontroll-Abhängigkeiten
  - Schleifen, Verzweigungen, Assertions, …
  - Branch-Delay
  - ILP: Anzahl-Instruktionen in Verarbeitung steigt
- Branch Prediction
  - Sprungentscheidung dynamisch vorhersagen
  - ausgewählte Alternative starten
  - rückgängig machen falls Fehlentscheidung
- Branch Prediction Buffer (branch history table)
  - Speicher, adressiert mit niederwertigen Adressbits
  - enthält letzte Entscheidung für diese Adresse (nicht Befehl!)
  - Verbesserung: n-Zähler
  - Entscheidung > n/2 > Sprung
  - Sprung inkrementiert, nicht springen dekrementiert
- Beispiel 4096 Puffereinträge á 2 Bit
  - 82% - 99% richtige Vorhersage bei SPEC89
- Korrelierte Prädiktoren für abhängige Sprünge

- Instruktionspool sollte immer gefüllt sein

```
        cmp a,b
        jle @else
        mov ecx, 0
        jmp @end
@else:  mov ecx, 1
@end:
```

- Problem
  - welcher Befehl soll als nächstes dekodiert werden?
  - mov ecx,0 oder mov ecx,1
  - Prozessor führt Code spekulativ aus
- Realisierung im PII & PIII
  - Puffer mit 512 Einträgen
  - Adresse des Sprungbefehls
  - Sprung ausgeführt: Ja oder Nein
  - jeder Sprung (ausgeführt oder nicht) wird vermerkt

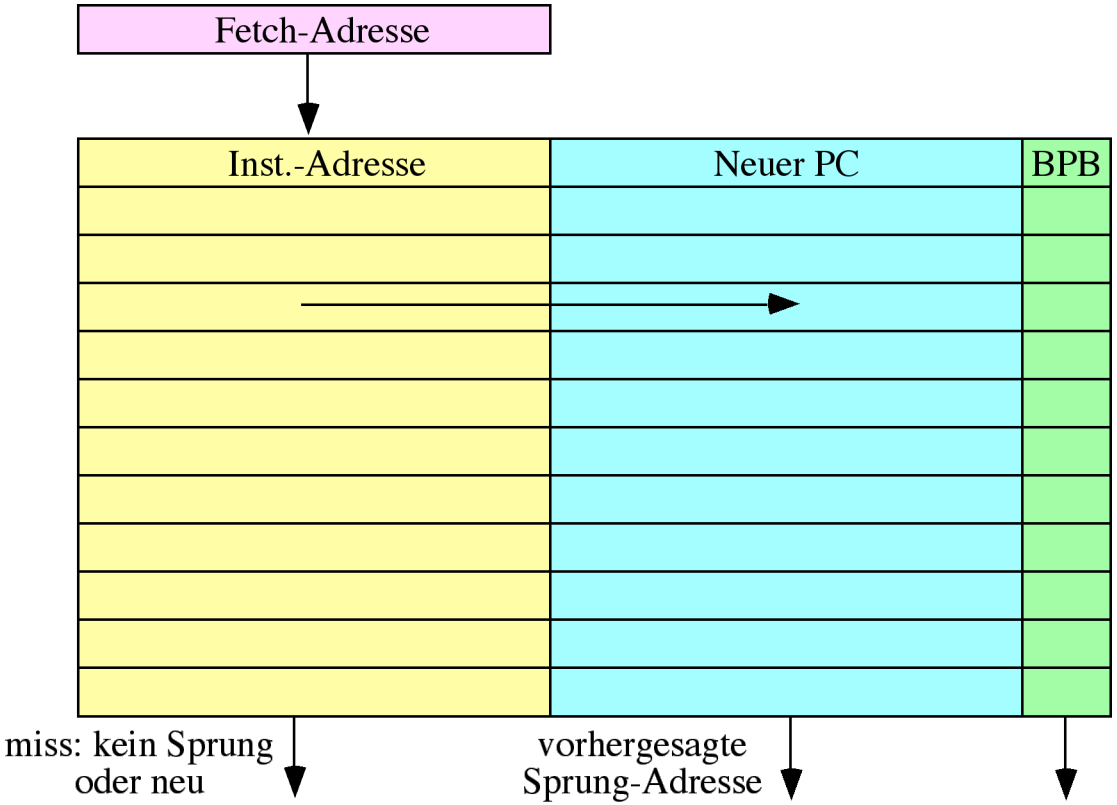
- Beispiel der dynamischen Sprungvorhersage



- Statische Sprungvorhersage
  - falls Sprung der Sprungvorhersage nicht bekannt
  - bedingte Rückwärtssprünge: Sprung ausführen
  - bedingte Vorwärtssprünge: Sprung nicht ausführen
- Strafzyklen (Penalties)
  - keine: korrekt vorhergesagt, dass der Sprung nicht ausgeführt wird
  - 1 Zyklus: korrekt vorhergesagt, dass Sprung ausgeführt wird
  - 9 - 26 Zyklen bei falscher Vorhersage

- Branch Target Buffer
  - nicht Entscheidung vorhersagen, sondern Ziel
  - Test beim Laden *jeder* Instruktion
  - neue Sprünge eintragen
  - evtl. Ziel ändern

```
if (PC in BTB) nextPC = item[PC].predictedPC
```



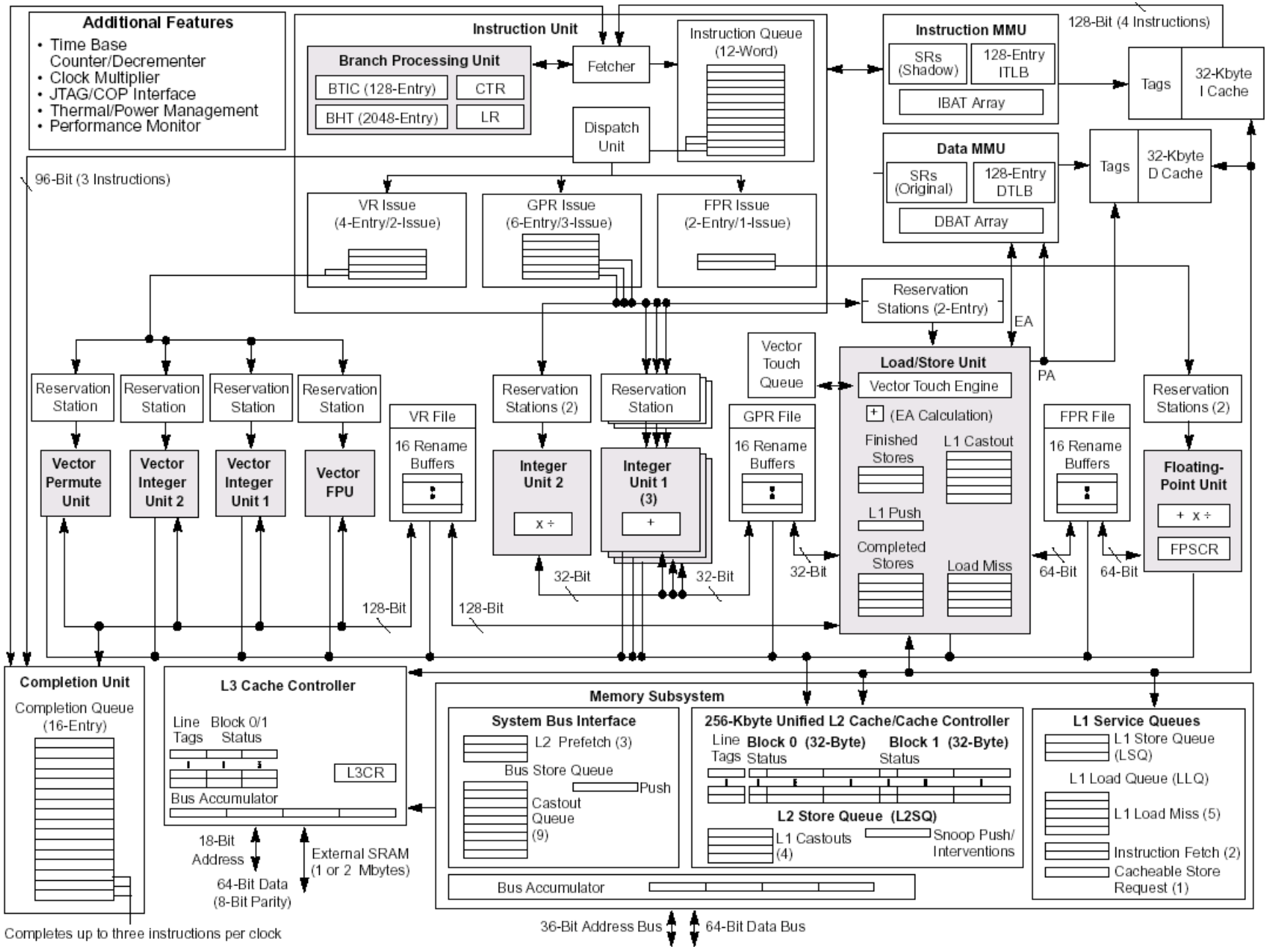
5.3.2.3 Stichwort: Superskalare Architektur

- Multiple Instruction Issue
  - an mehreren Instruktionen parallel arbeiten
  - parallele Pipelines
  - aber evtl. Abhängigkeiten
- Very Long Instruction Word (VLIW)
  - mehrere Befehle pro Instruktions-Adresse
  - statisch Funktionseinheiten zugeordnet
  - siehe Crusoe (oben) und Signalprozessoren
- EPIC: Explicitly Parallel Instruction Computing [IA64]
  - jede Instruktion trägt Branch-Zweig-Prädikat
  - spekulative Ausführung aller Zweige
- Superskalare Ausführung
  - 2 bis 8 Befehle gleichzeitig starten
  - abhängig von Hazards
  - mehrere Befehle gleichzeitig holen: 64/128 Bit
  - einfachste Variante: 1* Integer + 1 * FP
  - dynamisch geplant: Scoreboard oder Tomasulu

Zusammenfassung Instruktionsausführung

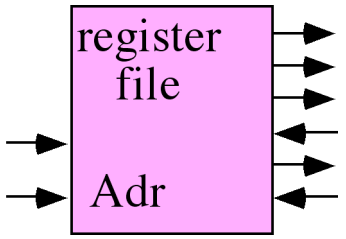
- Pipeline steigert Durchsatz
  - Phasen der Verarbeitung durch Puffer entkoppelt
- Probleme: Hazards
  - echte Datenabhängigkeit und Namensabhängigkeit
  - Ressource-Engpässe
  - Control-Hazards
- Statische Techniken zur Optimierung
  - Delay-Slot füllen
  - Schleifen ausrollen
  -> Instruktionsplanung
- Dynamische Techniken
  - Instruktionen bedingt starten: Hazard-Entdeckung
- Leistungssteigerung: Instruction Level Parallelism (ILP)
  - parallele Funktionseinheiten
  - Scoreboard, Tomasulu
  - completion unit bewertet spekulative Ausführung
  - VLIW oder multiple Issue

- PowerPC 7450

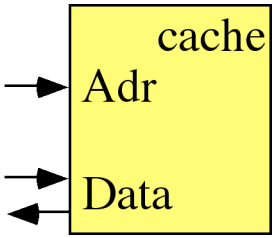


6. Speicherarchitektur

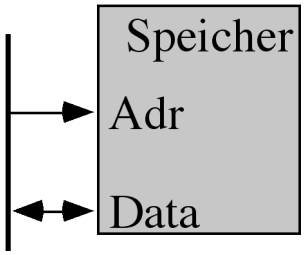
- Read/Write Speichertypen
  - statisches RAM
  - dynamisches RAM
  - EEPROM: Schreiben sehr langsam
  - Flash: Schreiben langsam
  - Magnetspeicher: langsamer Zugriff



- Hierarchie
  - Register (Flip-Flops u.ä.)
  - Level 1 Cache integriert in Prozessor (8-64 K, *2?)
  - on-chip Cache auf dem Prozessor-Die (L2, z.B. 128-1024 K)
  - externer/Backside Cache (L3, z.B. 2/4 MB)
  - Hauptspeicher
  - Virtueller Speicher: Auslagern von Teilen auf Magnetspeicher

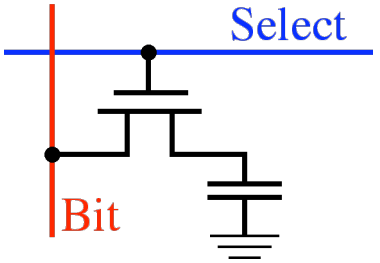


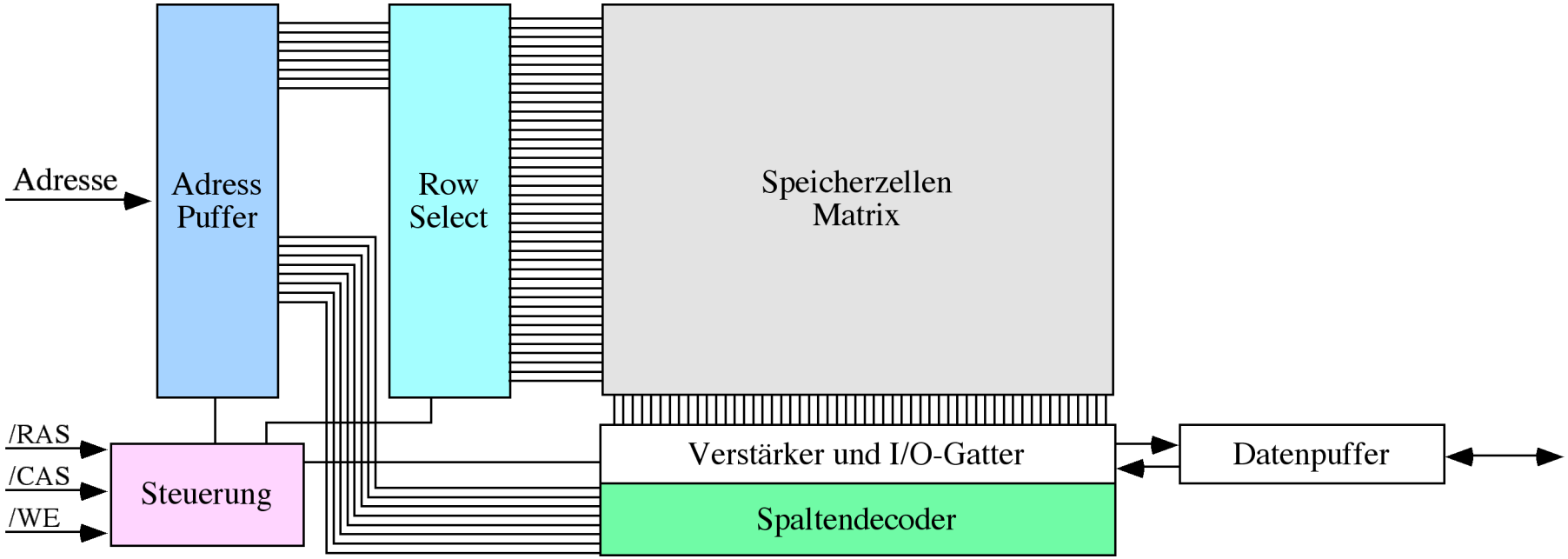
- (Flash-)ROM
  - Urlader
  - Parameter
  - Systemkomponenten: BIOS, Toolbox, etc.
  - Programme für Embedded Systems



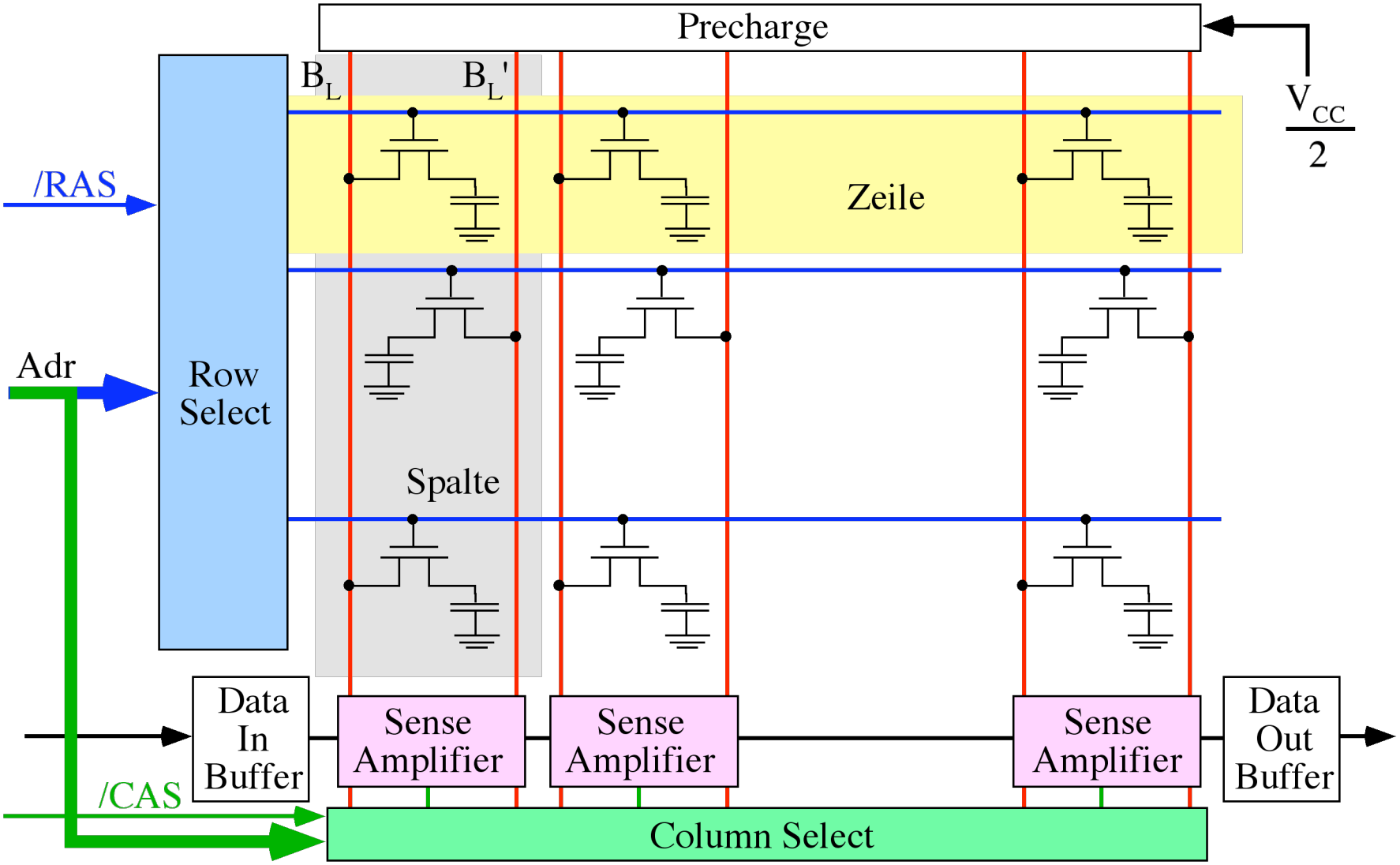
6.1 Hauptspeicher

- Dynamisches RAM (DRAM)
  - 1 Kondensator + 1 Transistor pro Speicherzelle (Bit)
  - Refresh nach 8 - 64 msec
- DRAM-Zugriff
  - Adressleitungen sparen: Multiplex
  - Anordnung als Matrix in Zeilen und Spalten
  - Row Address, danach Column Address
  - Strobes wenn gültiger Teil anliegt: RAS und CAS

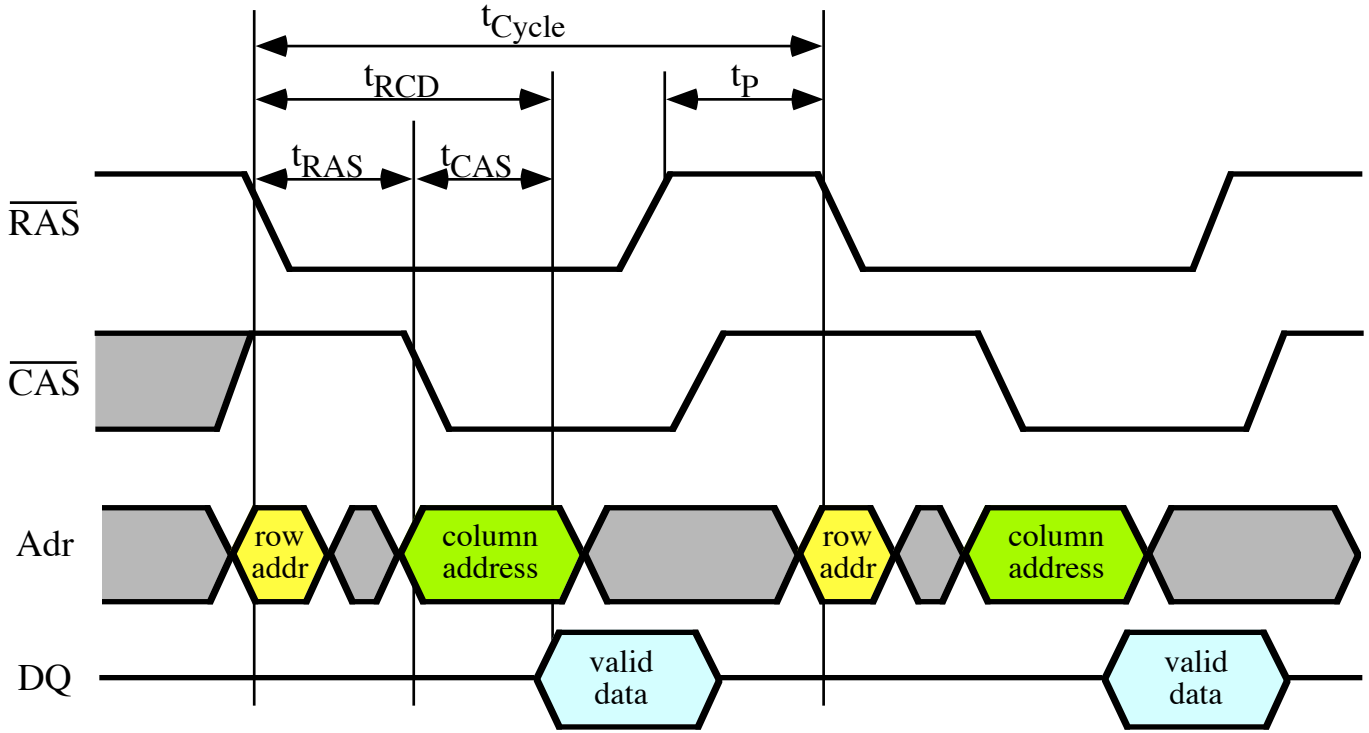




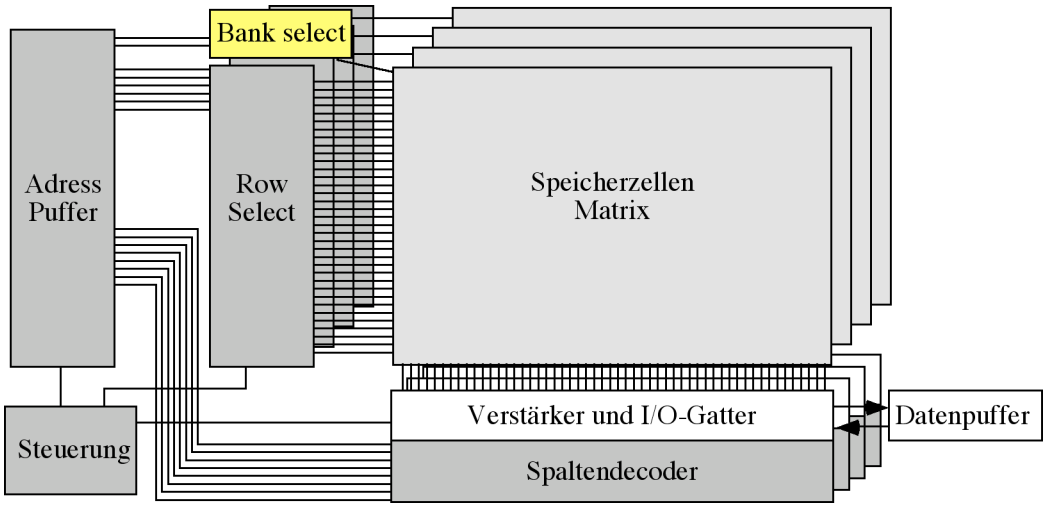
- Dynamisch?
  - Lesen zerstört Bit => Zurückschreiben (Refresh)
  - Auswerten geringster Ladungen (pF Kondensatoren)



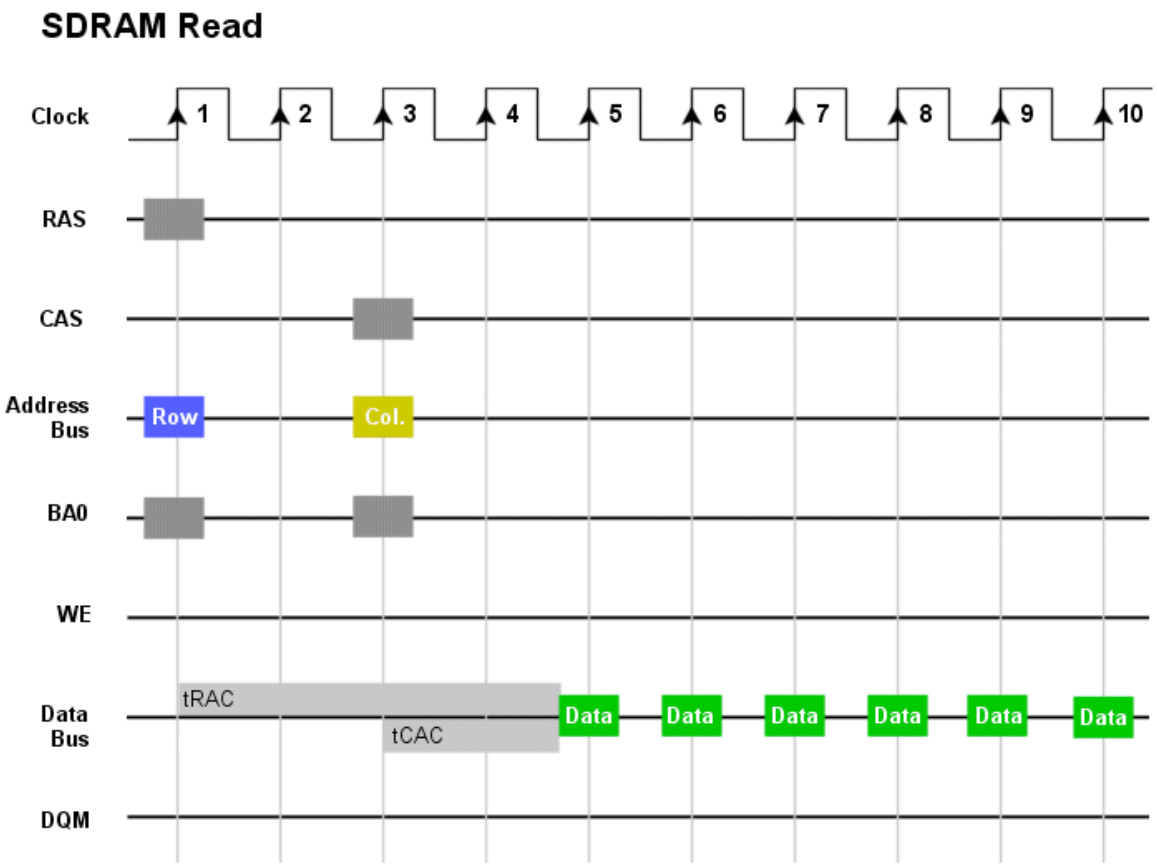
- Ablauf
  - R/W anlegen
  - MSB Adresse anlegen
  - /RAS
  - R-Adr. dekodieren
  - Zeile ansteuern
  - Zeile verstärken
  - LSB Adresse anlegen
  - /CAS
  - Bits auswählen
  - Daten liegen an
  - Kondensatoren aufladen
  - Precharge auf Vcc/2
- Zugriffszeit 50 ns (SDRAM)
- Zykluszeit 70 ns (SDRAM)
  - vor nächstem Lesen
  - Ladung wiederherstellen
  - Precharge abwarten



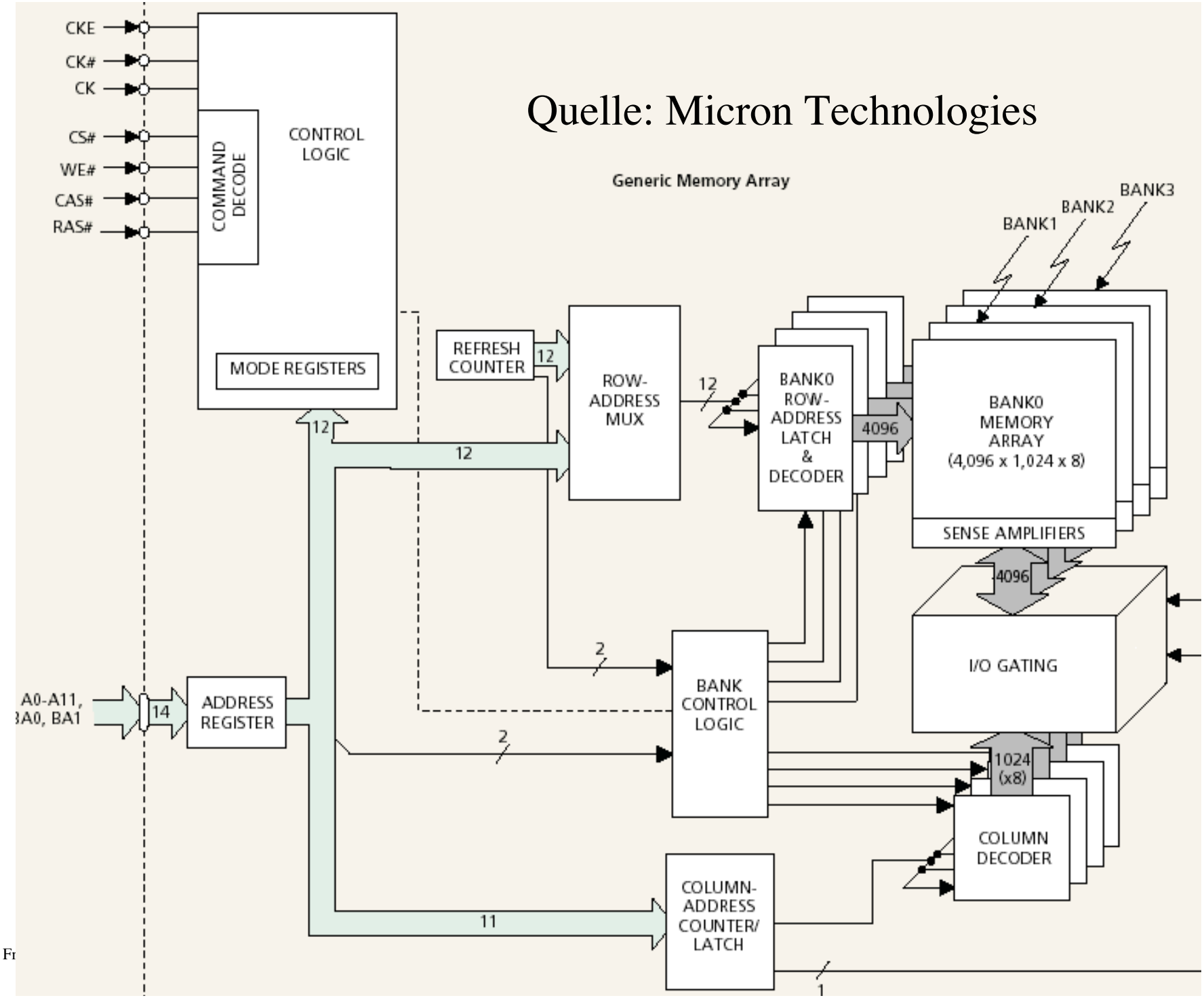
- Refresh
  - über Zeit geht Ladung verloren, falls kein R/W-Zugriff
  - 8-64 msec
  - Steuerung erzeugt Lese-Zyklen für Zeilen
  - extern durch 'Zeilenleser'
  - evtl. transparent für Prozessor (hidden refresh)
- Refresh zeilenweise
  - ~ 20 / sec
  - 2k, 4k, 8k Zeilen
  - pro Zeile T_{RC}-T_{CAC} => 40 ns
  - 20 * 4096 * 40 ns = 3.3 msec
- Anordnung
  - Bit-parallel: nMx8 etc.
  - 2 Bänke reduzieren Precharge-Wahrscheinlichkeit
  - entspr. Zugriffsmuster optimieren



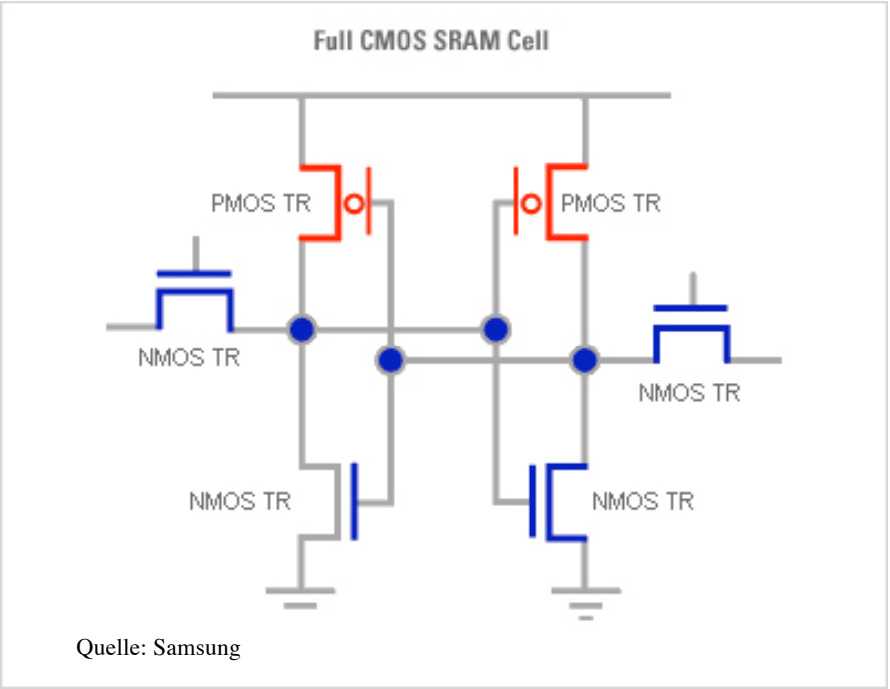
- DRAM asynchron
  - Adresse anlegen, RAS, CAS
  - CPU wartet bis Daten bereit
- FPM: (RAS - CAS-Daten) - CAS-Daten - CAS-Daten - CAS-Daten
  - 6-3-3-3 Bus-Zyklen
- EDO 5-2-2-2@66 MHz
- SDRAM
  - synchron zum Speicherbus
  - 100, 133, 266 MHz
  - CPU kann anderes tun
  - timing z.B. 3-2-2
  - CAS-Latenz
  - RAS-CAS Delay
  - RAS precharge
- DDR-RAM
  - Datenaustausch an beiden Flanken
  - DDR2: Transfertakt = 2 * Speichertakt, prefetch buffer 4 bit
  - DDR3: Transfertakt = 4 * Speichertakt, prefetch buffer 8 bit



Quelle: Micron Technologies

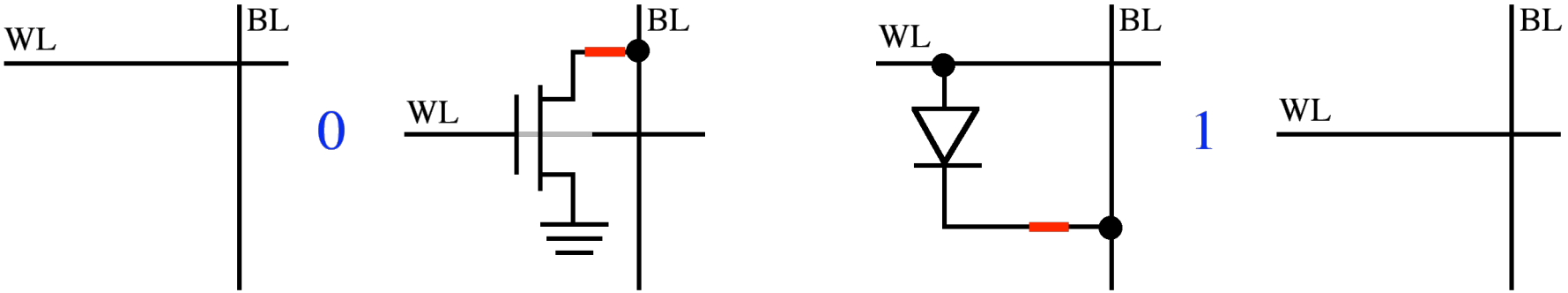


- DRAM-Probleme
  - Adressmultiplex (Row-Column) kostet Zeit
  - Precharge bestraft Mehrfachzugriff auf Chip hintereinander
  - FP-RAM, EDO, SD-RAM, … beschleunigen sequentiellen Zugriff
  - 50 ns = 50 Prozessorzyklen bei moderner CPU
- Speichersystem-Optionen
  - n*Wortbreite => n*Speicherdurchsatz (Alpha AXP 21064: 256 bit)
  - Interleaving
  - Unabhängige Speicherbänke
- Statisches RAM (SRAM)
  - 4 - 6 Transistoren pro Speicherzelle (Bit)
  - alle Adressleitungen zum Chip
  - kein Refresh
  - Zugriffszeit = Zykluszeit
  - Kapazität ca. 1/4 - 1/8 von DRAM
  - 8 mal teurer
  - 8 - 16 mal so schnell



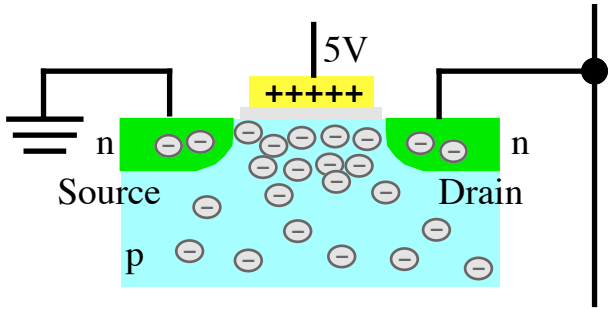
Quelle: Samsung

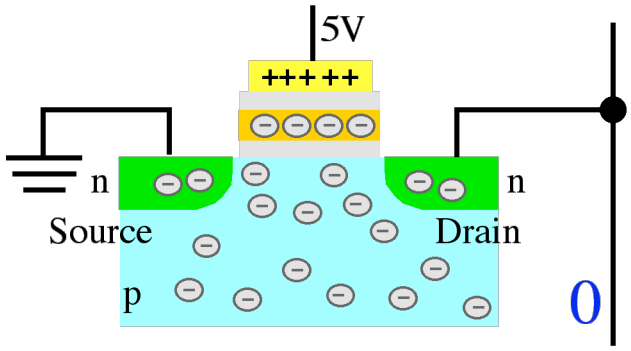
- ROM - Read Only Memory
  - Dioden-Matrix
  - Zeilenleitung aktivieren
  - Diode -> 1; fehlende Diode -> 0
  - CMOS: Transistoren an den verbundenen Kreuzungen
  - maskenprogrammiert

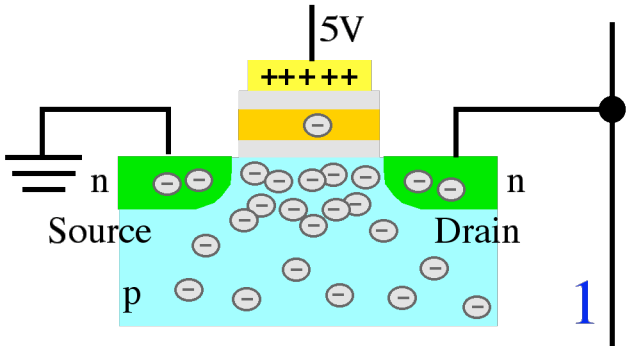


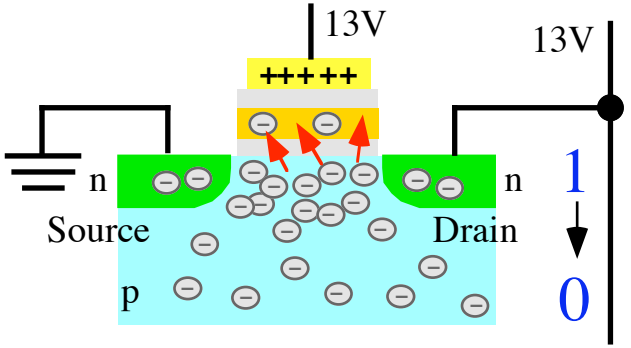
- PROM - Programmable ROM
  - alle Kreuzungen verbunden
  - Fuse an jeder Verbindung
  - Fuse mit Programmierspannung 'durchschmelzen'

- EPROM - Erasable PROM
  - besonderer Transistor mit 'Floating Gate': FAMOS
  - Fowler-Nordheim Tunneleffekt
  - Löschen mit UV-Licht

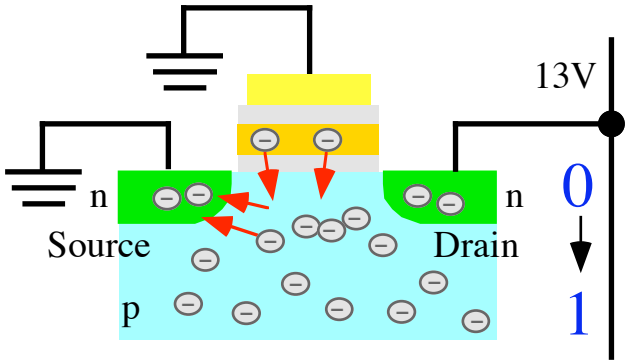








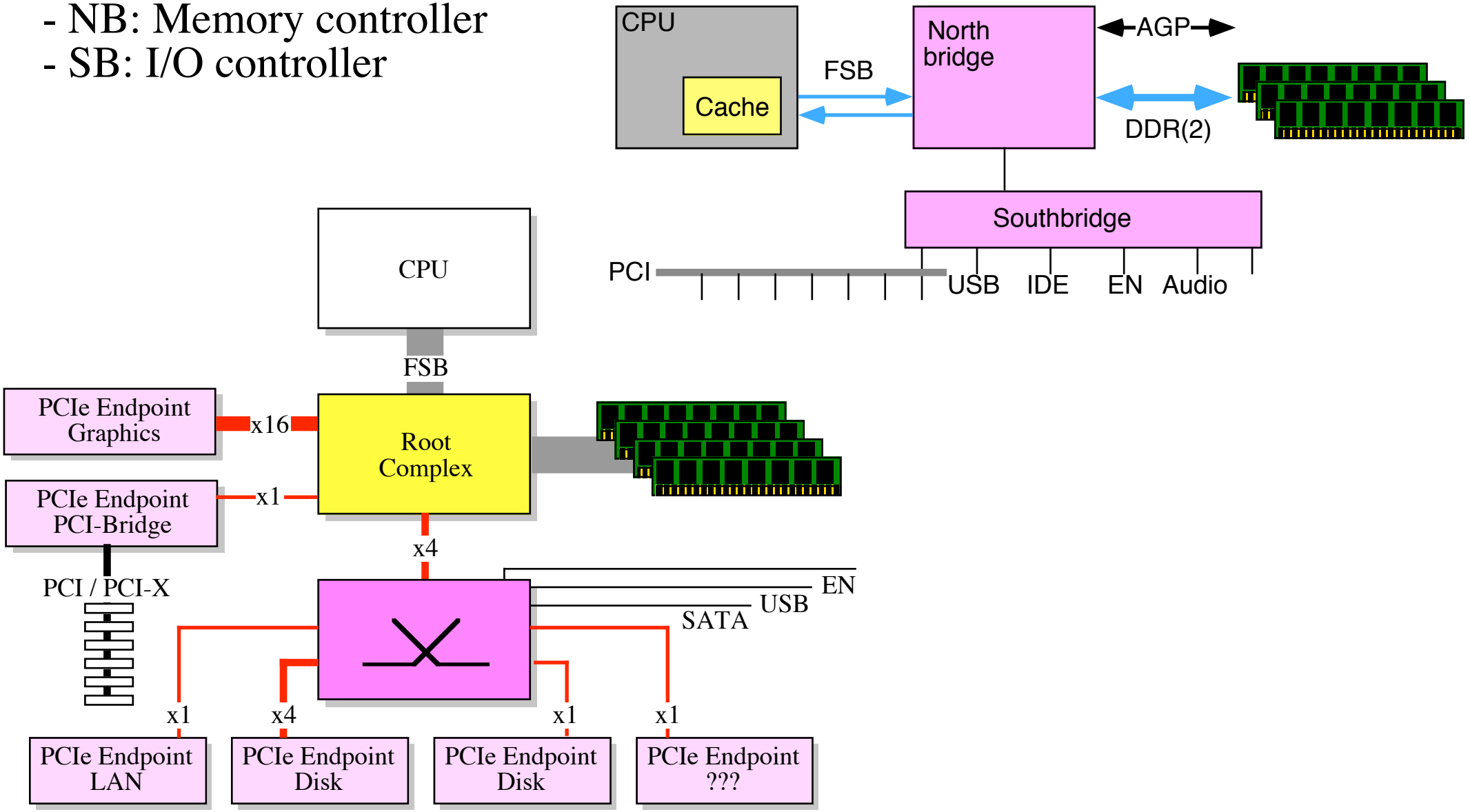
- EEPROM - Electrically EPROM
  - floating gate Transistoren elektrisch löschbar
  - Zellen 'einzeln' löschen
- Flash Memory
  - ganze Bänke löschen
  - modern: mehrwertige Logik



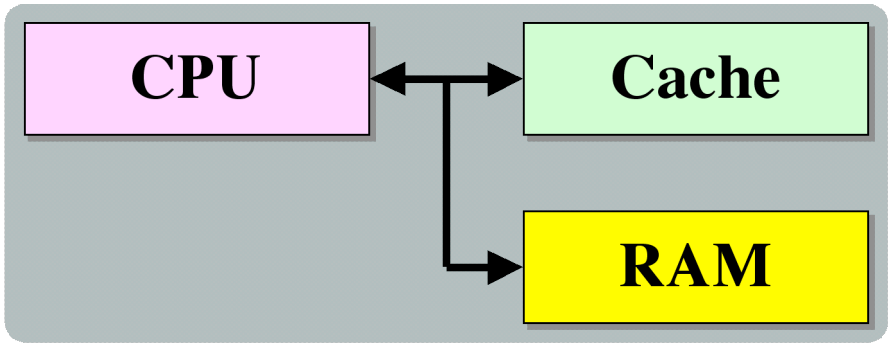
6.2 Bridges: Interface CPU/Cache - Speicher - Bus

- DDR/DDR2 Speicher
  - 64 bit pro Modul, Dual Channel: 2 Kanäle á 64 Bit
  - Adressbits nach Kapazität
  - CAS, RAS-CAS, Gesamtzyklus
  - DDR: z.B. 2-2-6 bei 100 - 200 MHz
  - DDR2: z.B. 4-4-12 bei 100 - 300 MHz
  - DDR3: z.B. 5-5-15 bei 100 - 200 MHz
- CPU
  - Datenzugriff zunächst im Cache, evtl. Cache Miss
  - Cache-Line 4 oder 8 Worte = 128/256 bit
  - Speicherzugriff transportiert Zeile
  - Bsp: PPC 970FX: ADIN(0:43), ADOUT(0:43)
  - Multiplex: Adresse-Daten-Daten-Daten-Daten
  - P6-Bus: 64 bit 'split-transaction': ReadRequest, MemoryReply
- Test [Bahmann, 2005]
  - sequentiell 75 MTrans/s
  - entspricht 11 CPU-Takten

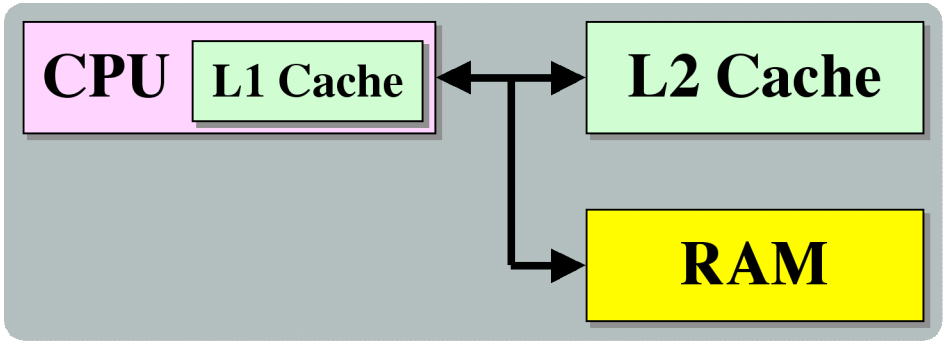
- Systemarchitektur mit Bridges
  - NB: Memory controller
  - SB: I/O controller



6.3 Cache



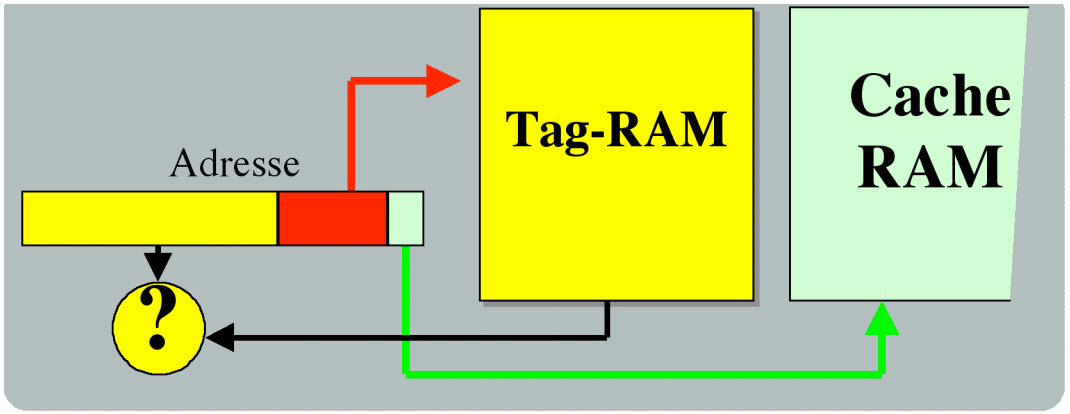
- Problem
  - Pipelinestufe MEM
  - Daten in 1 Zyklus, sonst stall(s)
  - DRAM zu langsam
  - SD-RAM 100 MHz => 50 Stalls!
- Schneller Pufferspeicher
  - zwischen Pipeline und Speicher
  - hält bereits benutzte Speicherinhalte
  - Befehle und Daten
  - eventuell getrennt für Befehle

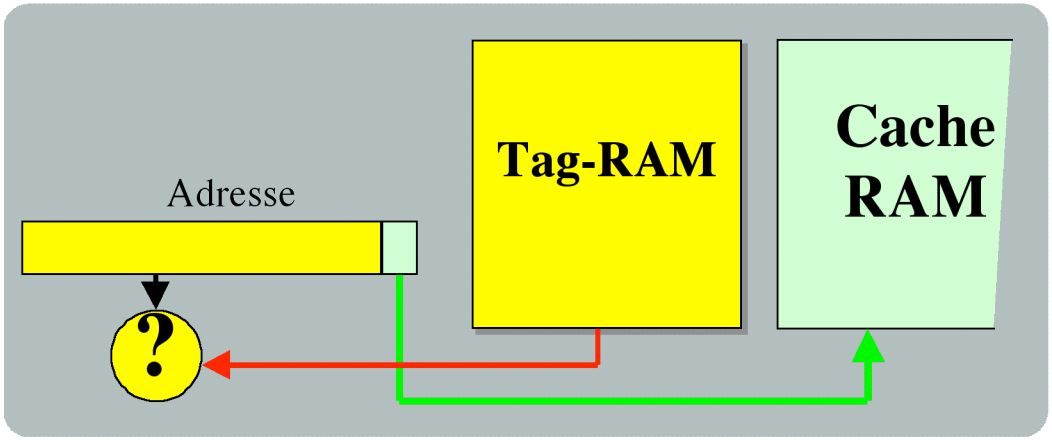


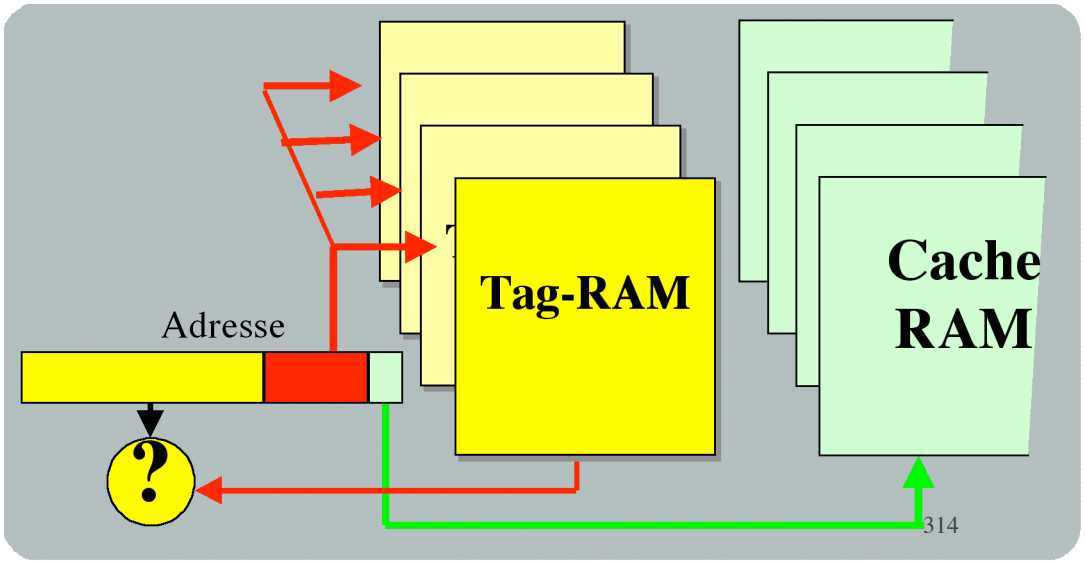
- Beschleunigung?
  - beim *ersten* Lesen einer Adresse Speicherwort puffern (langsam)
  - bei weiteren Lesevorgängen von Adresse Wort sofort liefern
  - beim Schreiben puffern und evtl. asynchron schreiben
  - Durchschreibestrategie
- Blockstrukturiert (Cachelines)
  - Cache speichert größere Speicherstücke
  -> Lokalitätsprinzip

- Schreibzugriffe 7% der DLX-Speicherzugriffe
  - 25% der datenorientierten Speicherzugriffe
- Write-Through (leicht implementierbar)
  - Cache-Inhalt sofort in unterer Hierarchiestufe nachführen
  - erhebliche Verzögerungen über den Speicherbus
  - ->Block-Write-Buffer reduziert Write-Stall
- Write-Back
  - Modifikationen im Cache durchführen
  - Cache-Inhalt erst beim Ersetzen der Zeile zurückschreiben
  - Dirty-Bit markiert beschriebene Cache-Zeile
  - Konsistenzproblem zwischen Cache & Hauptspeicher
  - Multiprozessor-Architekturen?
- Sonderfall:Write-Miss
  - Write-allocate: Zeile in den Cache laden und modifizieren
  - No-write-allocate: nur auf niedriger Stufe modifizieren

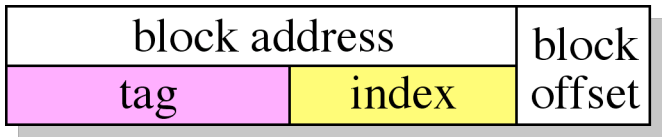
- Cachezugriff
  - Speicheradresse
  - Cacheadresse
  - ist das Wort im Cache?
  - wo liegt das Wort?
- Direct Mapped Cache
  - Abb. Adresse->Cacheeintrag
  - Vergleich ob Treffer
- Voll assoziativ
  - Vergleich Adresse - Tag-RAM
  - kostet evtl. Zeit
  - Hardware aufwendig
- Weg assoziativ
  - Untermenge (Set, Blockgruppe)
  - Set-Auswahl direct mapped
  - in der Untermenge assoziativ
  - n Wege: n Blöcke im Set
  - n-way set associative





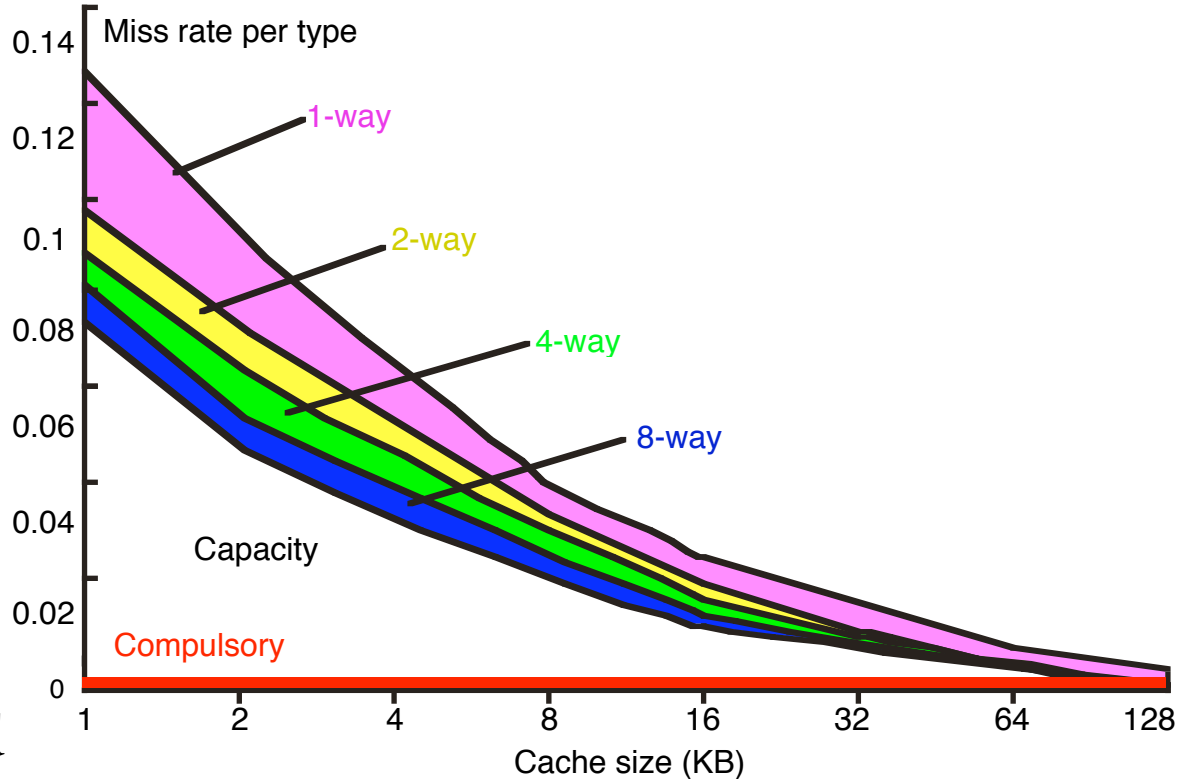


- Zugriffsstruktur Block-Tag
  - stückweise Interpretation der Adresse
  - Test der Blöcke im Set: tag
  - Index des Sets im Cache
  - Block-offset führt zum Datenwort
  - Gültigkeits-Bit (vaild-bit)



- Tag-Vergleich parallel
  - z.B. wired-XOR
  - parallel Daten aus dem Cache holen
  - n-Wege => n:1 Multiplexer
- Austausch mit dem Hauptspeicher
  - immer ganze Cache-Zeilen (z.B. 16 oder 32 Bytes)
  - als Burst-Zugriff transportiert
  - Cache-Zeile überschreiben: alten Inhalt vorher zurückschreiben?
- Block-Austausch-Strategie
  - Zufall
  - LRU: Least Recently Used

- Ursachen für Cache-Miss
  - unvermeidliche (compulsory)
  - größenbedingt (capacity)
  - konfliktbedingt (conflict)
- Miss-Rate reduzieren
  - größere Blöcke => größere(r) Miss-penalty
  - mehr Assoziativität => Kosten, Zeit für Vergleich?
  - 'Opfer'-Cache (z.B. 4 Einträge)
  - Prefetch bei Miss: Block[i] und Block[i+1]
  - Prefetch-Anweisungen vom Compiler
  - Compiler erhöht Lokalität des Programmes (z.B. Arrays mischen)
- Cache-Miss-Penalty reduzieren
- Cache-Hit-Zeit reduzieren
  - virtual Cache vs. physical Cache (aber: flush beim Prozesswechsel)

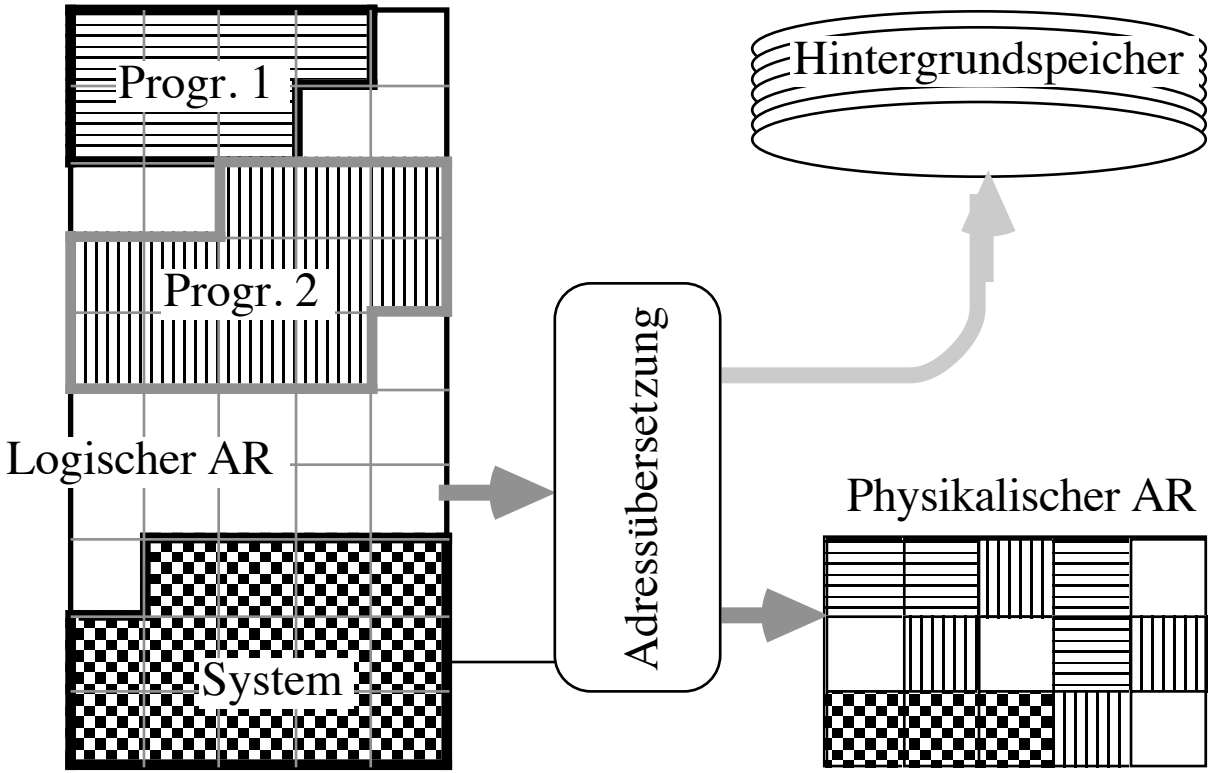


6.4 Virtueller Speicher

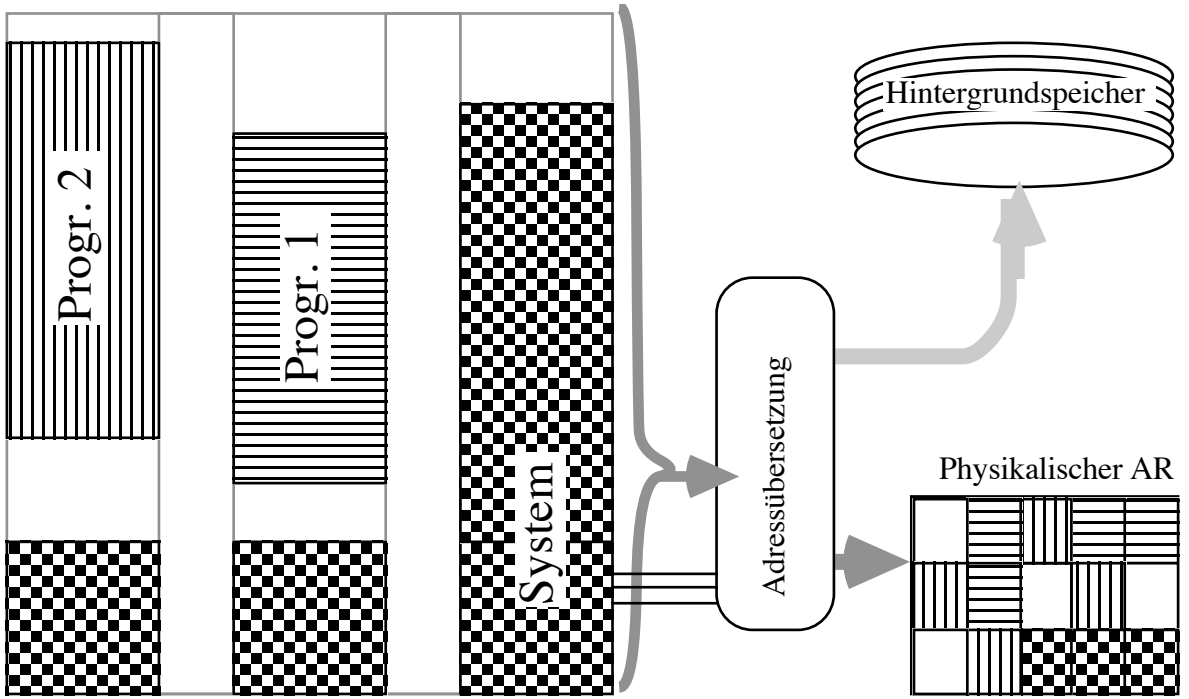
- RAM kleiner als Adressraum
  - 512 MB = 2^{29}
  - Programme wollen konzeptuell den ganzen Adressraum benutzen
  - Programme zu groß => Overlays
- Ladepunktabhängigkeit: Relozieren des Codes beim Laden
- Multitasking
  - mehrere Prozesse laufen im Zeitmultiplex
  - komplettes Auslagern (Swappen) beim Prozesswechsel zu teuer
  => gleichzeitig im RAM
  - Größenproblem verschärft
  - Schutz vor anderen Prozessen?
- Virtueller Speicher
  - Prozess (Programm) sieht nur logische Adressen
  - flacher Adressraum so groß wie Adressbits erlauben
  - Prozessor sieht virtuelle Adresse
  - Memory Management übersetzt in physische Adressen

- Physischer Adresseraum
  - RAM beschränkt
  - Teil des Prozessspeichers ausgelagert auf Hintergrundspeicher
  - z.B. Festplatte
- Seiten-Kachel Konzept
  - virtueller Speicher in Seiten eingeteilt
  - physischer Speicher in Kacheln eingeteilt
  - Verbindungs-Tabelle kann sehr groß werden
- Memory Management Unit
  - übersetzt virtuelle Adresse in physische Adresse
  - überprüft Präsenz der Seite in Kachel
- Page Fault
  - Seite liegt nicht im physischen Speicher => Interrupt
  - Betriebssystem speichert eine Kachel
  - lädt Kachel mit benötigter Seite vom Hintergrundspeicher
  - ändert Seiten/Kacheltabelle
- Auslagerungsstrategie LRU

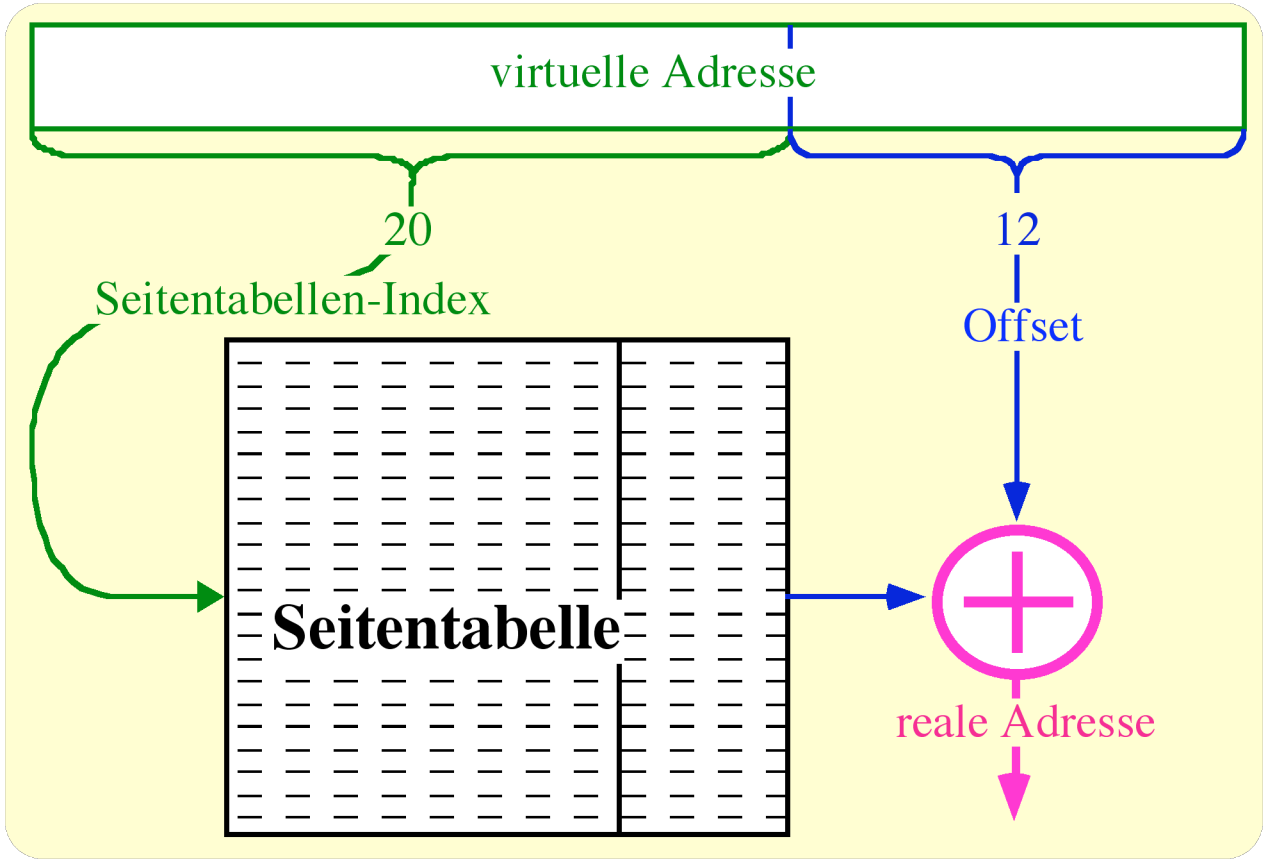
- Einfach virtualisierter Speicher
  - mehrere Programme teilen sich einen virtuellen Adressraum



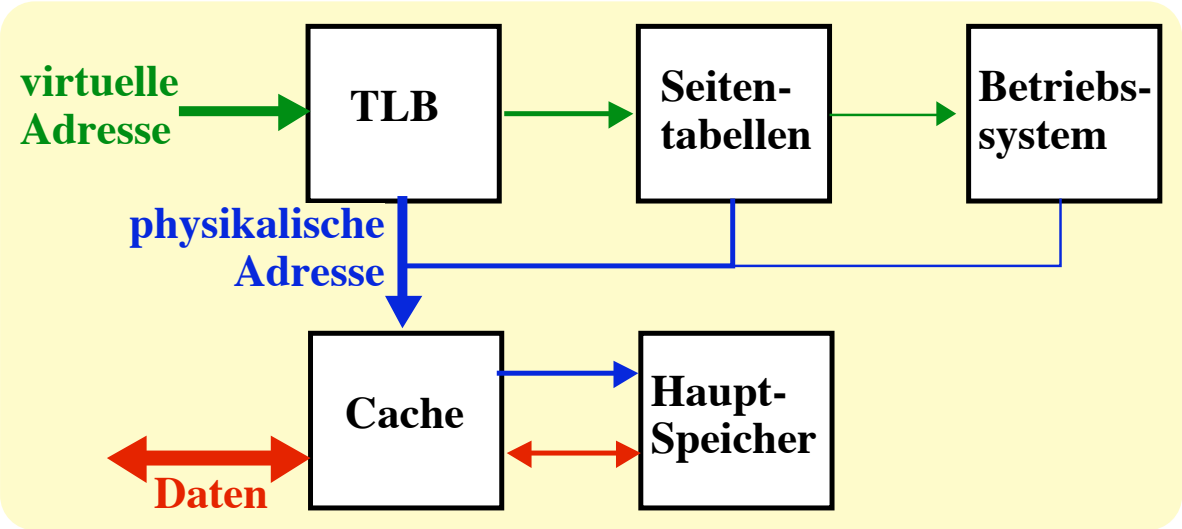
- Mehrfach virtualisierter Speicher
  - jedes Programm hat eigenen virtuellen Adressraum
  - Adressübersetzungstabelle beim Prozesswechsel umgeschalten
  - Teile des OS-Adressraumes auch für Anwendungsprogramme zugreifbar
  - Hardwareeinrichtung zur Adressübersetzung.



- Abbildung virtuelle Adresse -> physische Adresse
  - virtuell: Seitenadresse + Offset
  - Seiten/Kacheltabelle: Seitenadresse -> Kacheladresse
  - physische Adresse:= Tabelle.Kacheladresse + virtuelle_Adresse.Offset
  - Translation Lookaside Buffer: Cache für Adressübersetzung

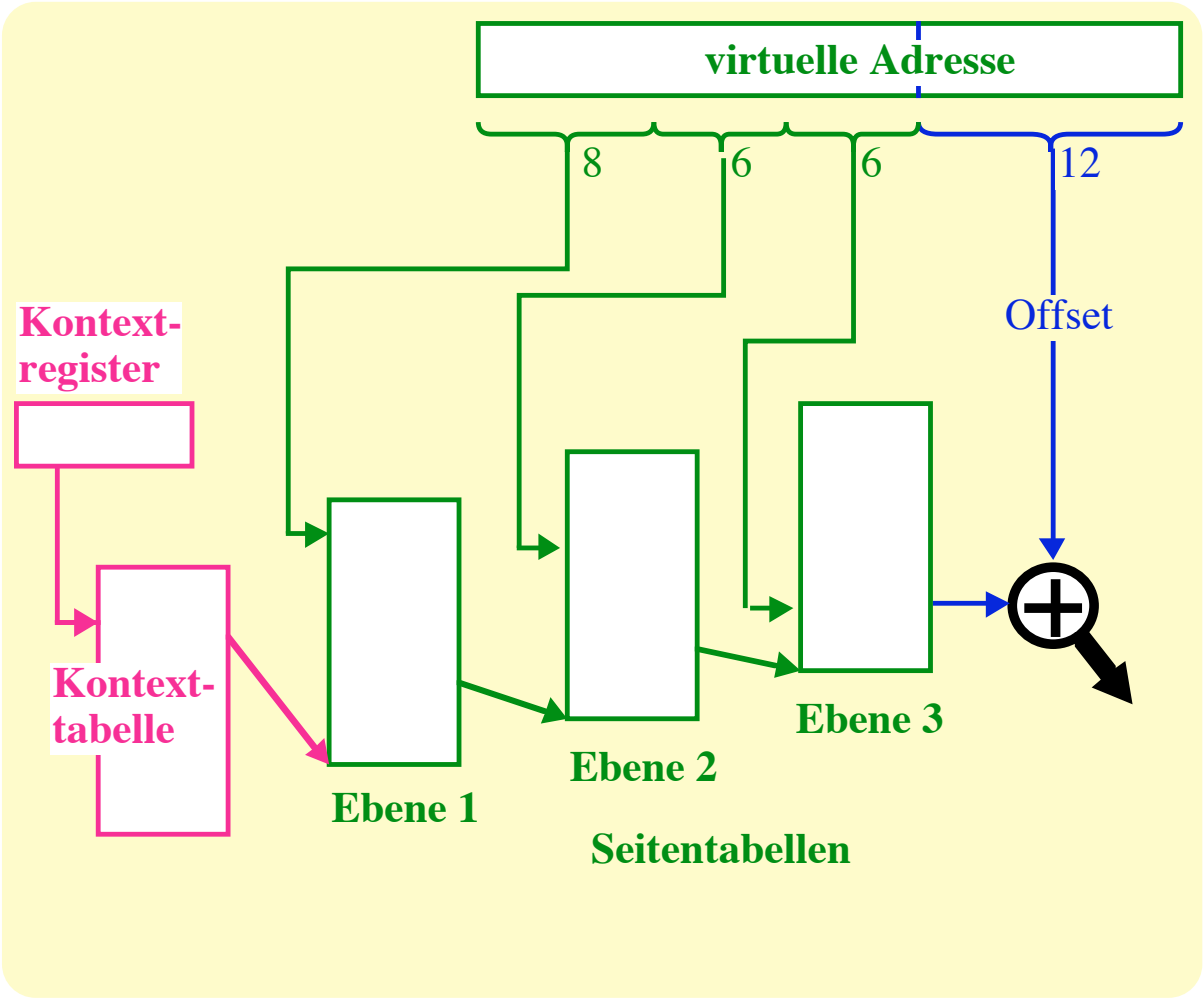


- Übersetzungspuffer (TLB)
  - TLB = "Translation Lookaside Buffer"
  - puffert schon früher übersetzte Adressen
  - kein Treffer im TLB
    => Hardware greift auf Seitentabellen im Hauptspeicher zu
  - Hohe Trefferrate (Hit ratio) wichtig

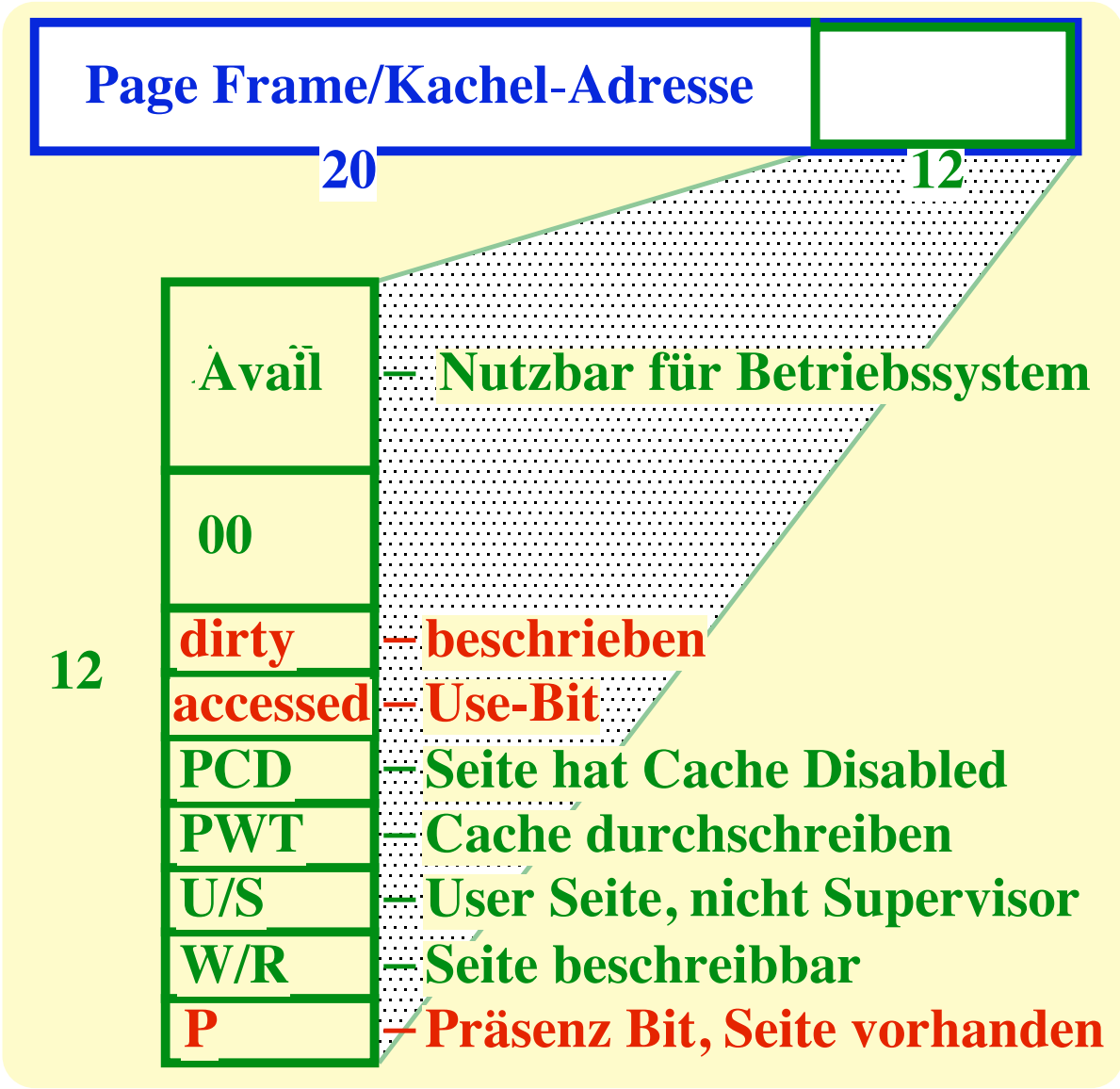


- Cache benutzt evtl. physische Adressen
  - nicht sichtbar für die Software
  - physische Adresse -> Eindeutigkeit

- Mehrstufige Adressübersetzung (IA)
  - eine Übersetzungstabelle pro Prozeß
  - Adresskontext umschalten



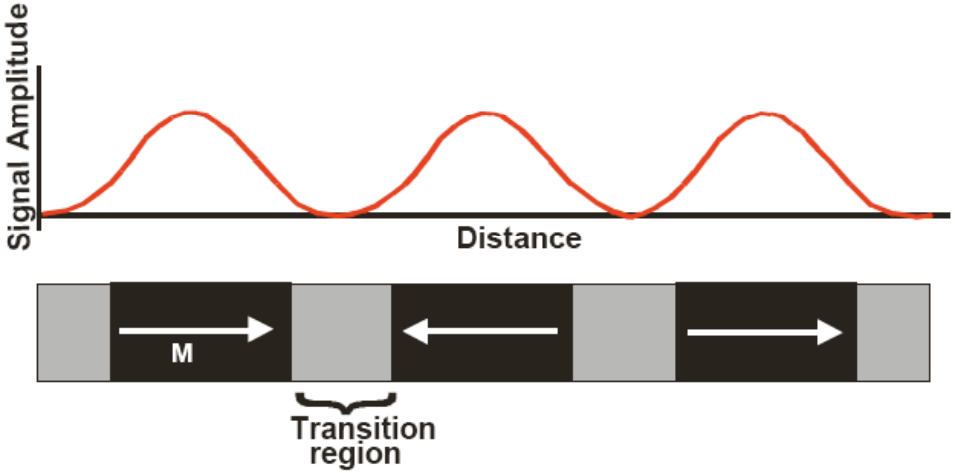
- Beispiel Seitentabelleneintrag Intel Architektur



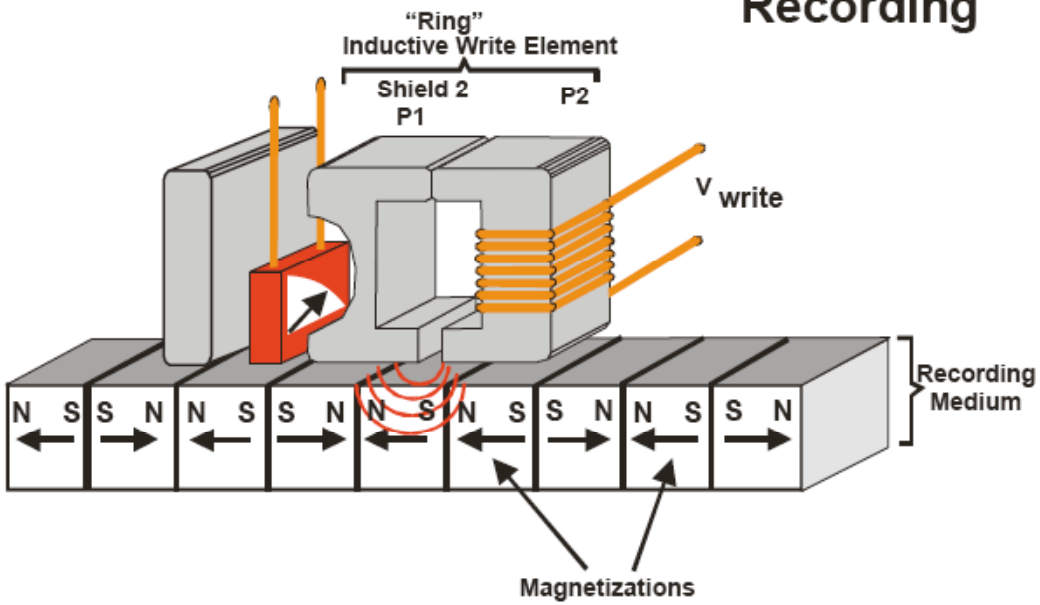
6.5 Plattenspeicher

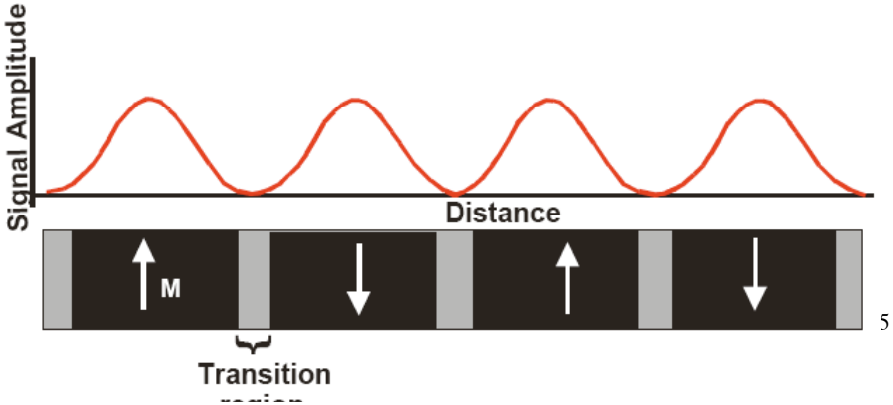
6.5.1 Hardware

- Magnetische Datenspeicherung
  - Kobalt Legierung
  - Träger Glas oder Aluminium
  - magnetische Regionen
  - longitudinal recording
  - Bilder von E. Grochowski,Hitachi
- Schreib/Leseköpfe
  - 'fliegen' über der Platte
  - einige Nanometer über der Oberfläche
  - Schreibspule, Lesespule
- Platten
  - Kreisscheiben
  - rotieren um Achse
  - z.B. 5400 U/min
- Perpendicular Recording
  - Bilder von E. Grochowski, Hitachi

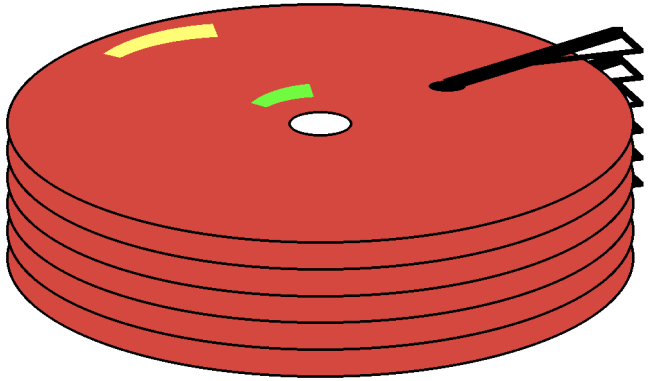


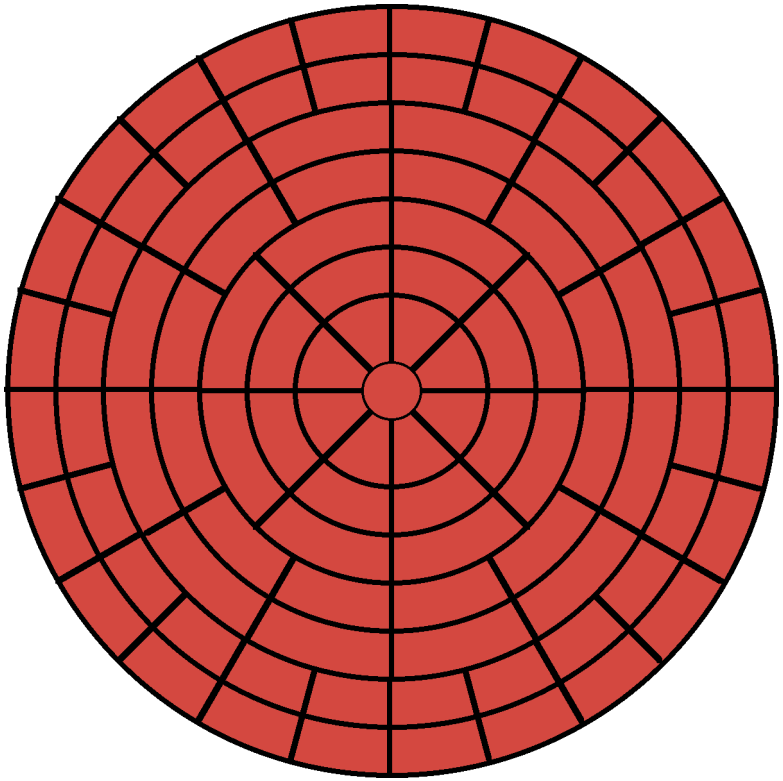
ngitudinal Recording





- Geometrie
  - n Platten auf einer Spindel (z.B. 5)
  - 2n Oberflächen -> 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)
- Zoned Bit Recording
  - CAV: constant angular velocity
  - unterschiedliche Sektoranzahl innen/außen
- Hardwareadresse
  - Zylinder
  - Kopf (~Plattenoberfläche)
  - Sektor
- LBA: Logical Block Address
  - Firmware bildet LBA auf HW-Adr. ab
  - mapping von defekten Sektoren
- Landing Zone, Servo Spuren

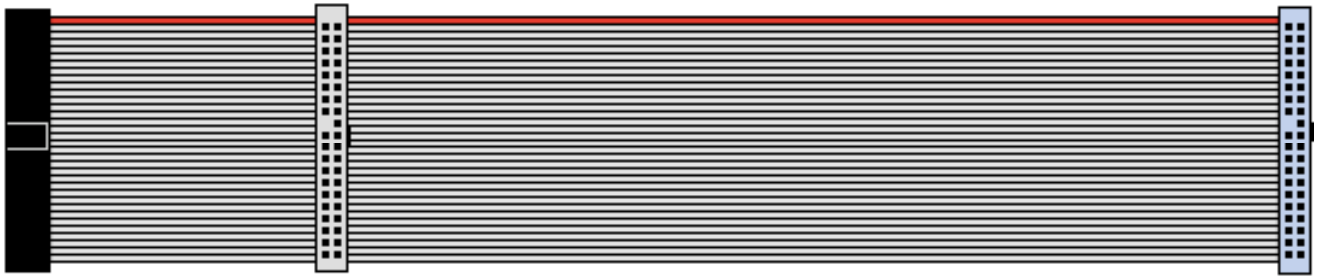




6.5.2 Interfaces

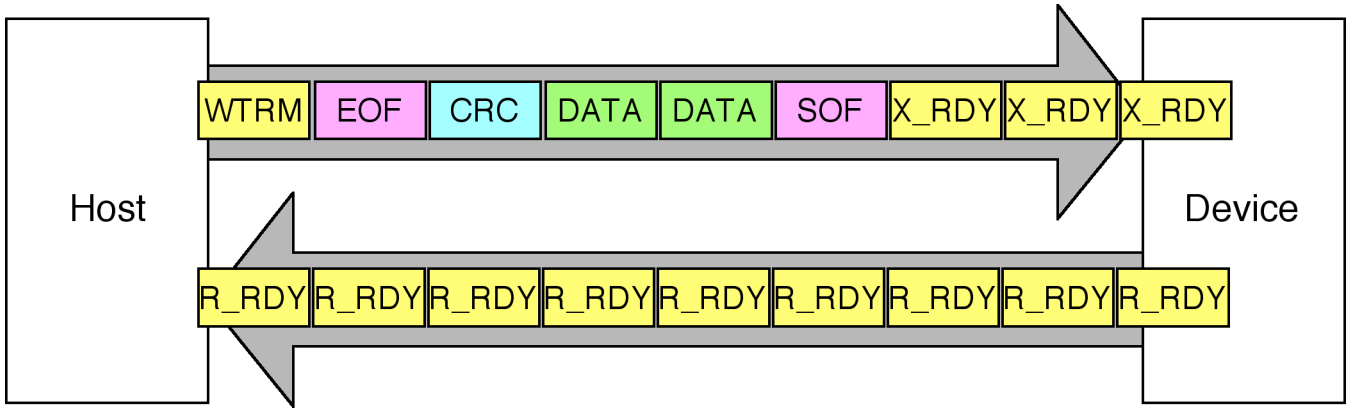
- ATA: AT Attachment
  - IDE: Integrated Drive Electronics
  - EIDE: Enhanced Integrated Drive Electronics
- Programmed I/O
  - wortweise mit CPU-Instruktionen transferieren
  - CPU generiert Adressen
- DMA-Mode
  - Hardware generiert Adressen
  - Wort belegt Bus nur einmal

Parallel ATA
- 40 Leitungen
- 16 Datenbits
- Kommandos: Read, Write
- Kontrollbits: HRDY, DDACK, C/S, Cable Select, IRQ
- Adressen multiplexen: Addr0, Addr1, Addr2
- ATA-4, …:overlapped commands
- 2 Geräte pro Interface: Master/Slave



- SATA: Serial ATA
  - 4-Draht Interface, 2 differentielle Paare
  - 3 Ground-Drähte
  - 8b/10b Code (siehe GB-Ethernet, …)
  - 0/1 Übergänge sorgen für Taktableitung
  - Punkt-zu-Punkt Verbindungen
- SATA Frames
  - SOF, FIS, CRC, EOF
  - FIS: Frame Information Structure
  - transportiert Daten, ATA-Kommandos, Shadow Register
  - Bsp. Kommando: 27h, command, cylinder, head, sector, sector count
  - Bsp. Daten: 46h, 1-2048 Datenworte

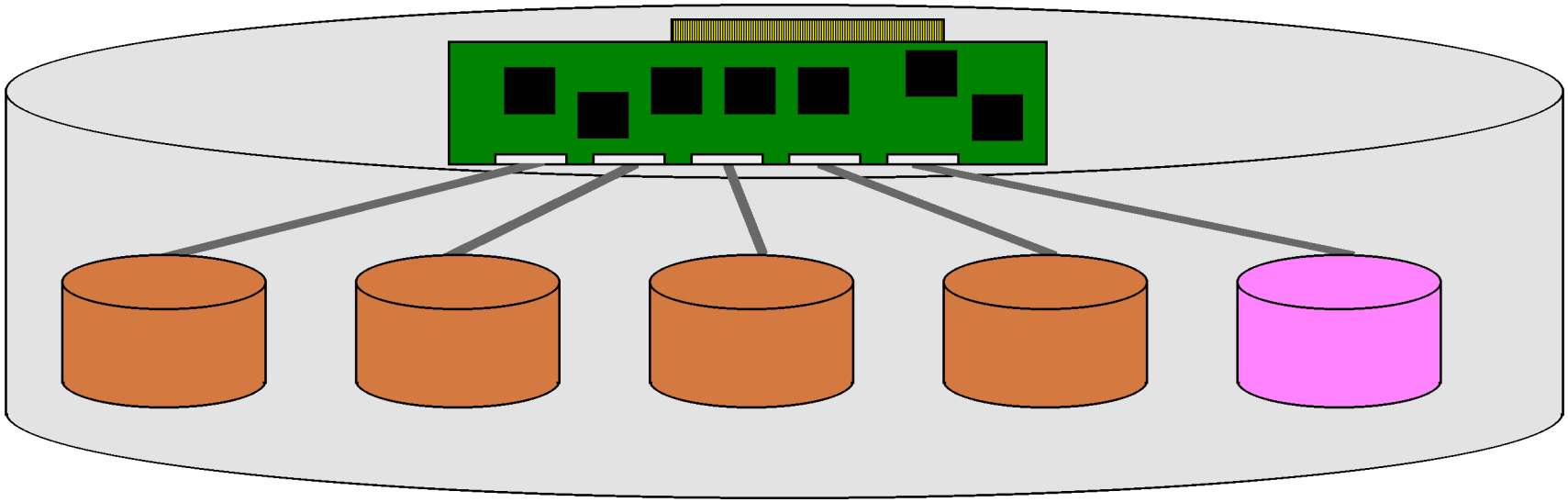




- Native Command Queuing
  - Controller optimiert Kopfbewegungen
  - shortest seek time, Scan, …
- SCSI
  - 8 Geräte an einem Bus
  - 8 Bit 'breit'
- SCSI command protocol
  - Kommandos in einem Byte
  - Read, Write, Format, …
  - Test unit ready, start/stop, mode sense, …
  - command desc. block mit Parametern, z.B. Adresse, …

6.5.3 RAID- Redundant Array of Independent Disks

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



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

- RAID 0: mehre Platten mit striping ohne Redundanz
- RAID 1: Schreiboperationen auf 2 Festplatten gleich (Plattenspiegeln)
- RAID 3 (für stromorientierte Anwendungen)
  - byte striping: Daten byteweise auf Platten verteilt
  - separate Platte(n) mit Parity
  - Obermenge von RAID 2
  - Platten synchronisiert
- RAID 4
  - block striping
  - einzelne Blöcke erfordern nur einen Zugriff
  - Blockgruppen von n Platten
  - separate Platte(n) mit Parity <-> Parallel Schreiben?
- RAID 5 (für transaktionsorientierte Anwendungen)
  - block striping mit verteilter Parity
  - Flaschenhals Parity reduziert
  - Platten nicht synchronisiert
- RAID 6, RAID 7, RAID 35

