IT-Grundlagen
WS 2013 / 14

Studiengang Scientific Programming

0 – Organisatorisches

Termine, Motivation, Inhalte der Vorlesung, Literatur, …

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Termine
Wann muss ich wo sein?

Regelmäßige Termine

- Vorlesung: Donnerstags 08:15 – 09:15 im Hörsaal
- Übung: Ist in die Java-Übung integriert

Klausur (vorläufig)

- Freitag, 07. Februar 2014 um 09:00

Motivation
Warum sind wir hier?

Inhalte der Vorlesung
Wo IT-Grundlagen draufsteht…

1. **Programmentwicklung**
2. **Zahlendarstellung**
    1. Ganze Zahlen
    2. Gleitkommazahlen
3. **Codierung**
4. **Syntax, Semantik, Grammatik**
    1. Formale Sprachen
    2. Compiler, Linker, …
5. **Rechnerorganisation**
    1. Von Neumann Architektur
    2. Maschinensprache
6. **Ein- und Ausgabegeräte**
    6. Speicher
    7. Caches
    8. Tastatur, Bildschirm, …
7. **Betriebssysteme**
    6. Speicherverwaltung
    7. Dateisysteme
8. **Kryptologie**
9. **Datensicherheit**
    6. Definition und Risiken
    7. Gesetzgebung

Literatur
Wenn ich den Typ vorne mal nicht verstehe…

Literatur

- Computerarchitektur: Strukturen – Konzepte – Grundlagen Andrew Tanenbaum / James Goodman, Prentice-Hall ISBN 3-8272-9573-4
- Wikipedia und dort genannte Quellen oder z.B. Homepages der Urheber (insbesondere, wenn nichts explizit angegeben ist)
- Sollten wir an irgendeiner Stelle die Rechte Dritter verletzt haben, bitten wir um konkrete Hinweise. Diese Zusammenstellung wird nicht kommerziell verwertet.
- Generell gilt: Viele Themen werden nur angeschnitten, die Vorlesung verstehen wir als Startpunkt für die eigene Beschäftigung mit den entsprechenden Themen.

Grundlage Folien

- U. Detert, B. Küppers, B. Kraft, B. Magrean, H. Pflug, M. Sczimarowsky, C. Terboven, A. Voß

Über mich
Wer bin ich – und wenn ja wie viele?

- Lebenslauf
  - MATSE-Ausbildung / Bachelor-Studium: 2007 – 2010
  - Master-Studium: 2010 – 2012
  - Webentwickler: 2012 – 2013
  - Dozent in der MATSE-Abteilung: seit 2013
- Interessen
  - Kryptographie, Künstliche Intelligenz, Zahlentheorie
- Sonstige Veranstaltungen
  - IT-Systeme, Parallelprogrammierung, Robotik (Uni Maastricht)
- Kontakt
  - Telefon: 0241 / 80 – 24770
  - Email: kueppers@rz.rwth-aachen.de

1 – Programmentwicklung

Entwurfsstrategien, Notationen

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

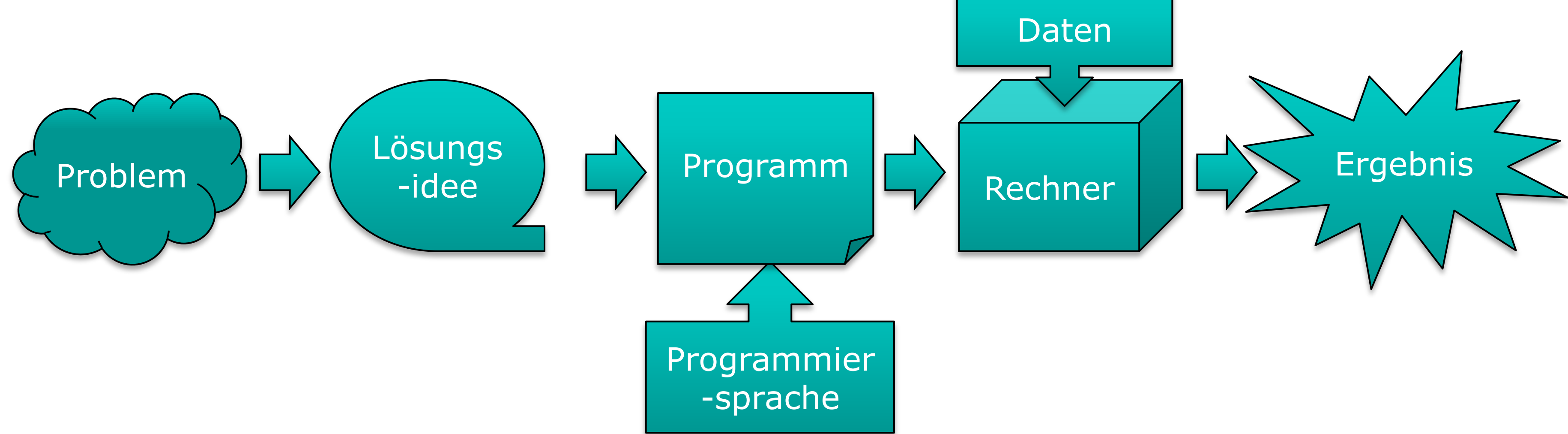
In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Programmkonzeption
  - Wasserfallmodell
  - Anforderungsanalyse
  - Entwurf
- Notationen
  - Flussdiagramme
  - Struktogramme

Vom Problem zur Lösung
Man nehme einen Hammer…

Definition: Programmieren

- Programmieren ist die exakte Beschreibung einer Lösungsidee in einer formalen Sprache



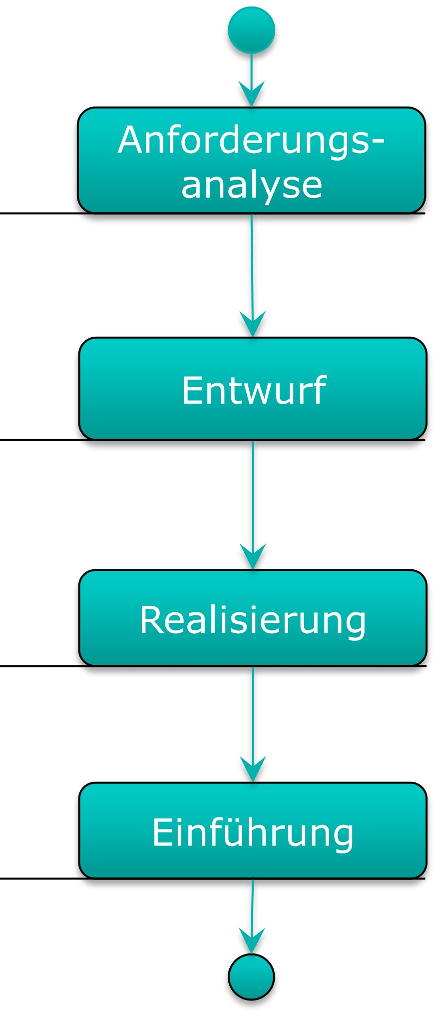
Drei Schritte

1. Analyse des Problems
2. Ausarbeitung eines Lösungs-Konzeptes
3. Formulierung der Lösung in einem Programm

Einfaches Wasserfallmodell
Wasser marsch!

Vom Problem zur Lösung

- Anforderungen abklären
- Schrittweises Ausarbeiten
- Algorithmen und Datenstrukturen entwickeln
- Installation und Inbetriebnahme



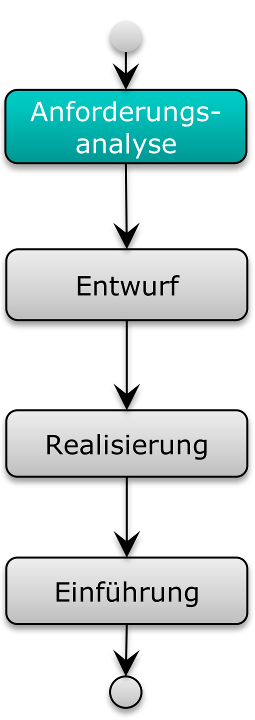
Anforderungsanalyse
Was will mein Kunde von mir?!

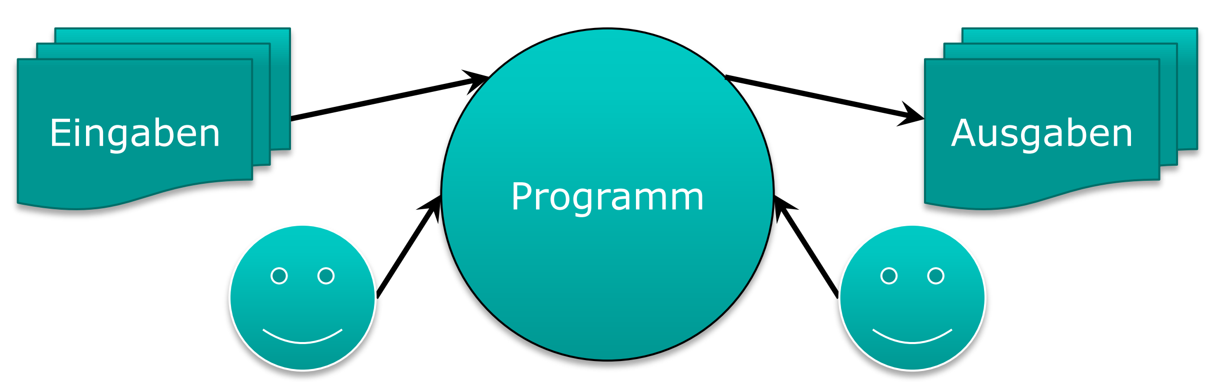
Problem genau analysieren und umfassend beschreiben

- Was soll genau entwickelt werden?

Vorgehensweise zur Analyse

- Datenorientiert
  - > Welche Daten werden eingegeben?
  - > Welche Daten werden ausgegeben?
- Aktivitätsorientiert
  - > Welche Benutzer interagieren wie mit dem System?





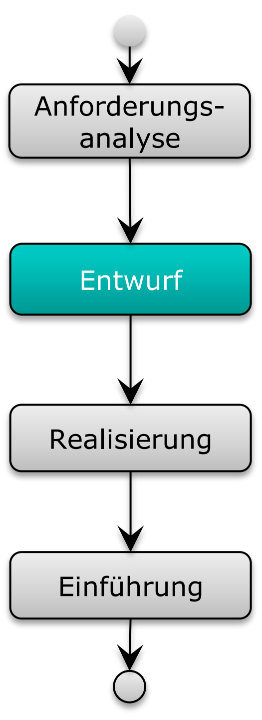
Entwurf
Aus Groß mach Klein

Formulierung des Gesamtproblems

- Überblick über System und relevanten Kontext
  - Ohne Details
  - Aus technischer Sicht

Zerlegung des Gesamtproblems in Teilprobleme

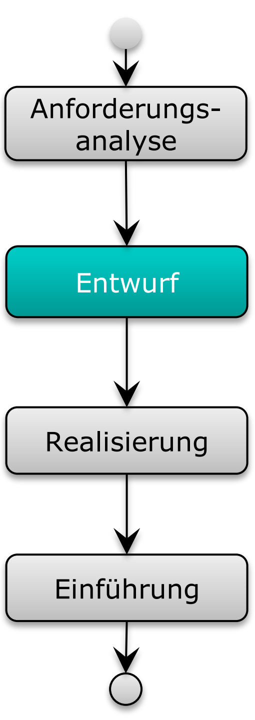
- Reduzierung der Komplexität
- Entwurfsstrategien
  - Top-Down
  - Bottom-Up
  - Mischformen



Top-Down
Hinab in die Untiefen

Vom Ganzen zu seinen Teilen

- Schrittweise Detailierung und Verfeinerung des Problems
- Formulierung von Teilproblemen (Modularisierung)
- Problemlösung erst nach Erreichen eines bestimmten Detail-grades



Beispiel: Hausbau

- „Ja, so mit vier Wänden halt"

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

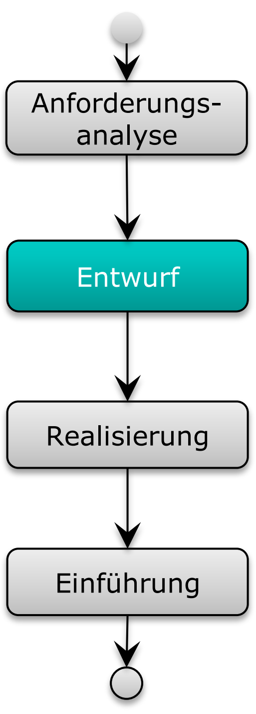
Bottom-Up
Hoch hinaus

Aus einzelnen Teilen ein Ganzes zusammenbauen

- Identifikation von elementaren Bausteinen
- Schrittweises Zusammensetzen der Bausteine
- Gesamtlösung entsteht zuletzt

Beispiel: Hausbau

- „Wie auch immer die Räume aussehen: Ich brauche LAN-Buchsen in jedem Raum"



Vor- und Nachteile
Top-Down vs. Bottom-Up

	Top-Down	Bottom-Up
Vorteile	• Kleine Teilprobleme leichter zu lösen • Einschätzung des Arbeitsfortschritts	• Gute Möglichkeit zur Wiederverwendung bestehender Bausteine • Technische Realisierung an der Basis früh betrachtet
Nachteile	• Bei einer großen Zahl von Modulen erschwerte Wirkungskontrolle • Gefahr technische Basis zu spät zu betrachten	• Gesamtproblem erst spät betrachtet • Sichtweise im Entwurf teilweise unvollständig

Notationen für den Entwurf
Anforderungsanalyse und jetzt?

Beschreibung der Analyseergebnisse im Entwicklungsprozess

- In standardisierter Form
- Für jede Phase eigene Notationen verfügbar
- Meistens grafisch
- Möglichst kompakt, lesbar und eindeutig

Motivation

- Dokumentation der Analysearbeit
  - > Anforderungen
  - > Schnittstellen
  - > Arbeitsschritte
  - > Standard- und Sonderfälle
- Abstimmung mit Teammitgliedern und Kunden
- Grundlagen für folgende Phasen

Notationen für den Entwurf
Grundlegende Grundlagen

Beispiele für Notationen

- Textuell
  - > Pseudocode:

Zuweisung:	Setze x auf den Wert von a	**$x \leftarrow a$**
Vergleich:	Ist y gleich 0?	**$y = 0$**
	Ist y ungleich 0?	**$y \neq 0$**

- Grafisch
  - > Flussdiagramme
  - > Struktogramme
  - > Unified Modeling Language (UML)

Pseudocode

Wie war das nochmal mit dem ggT?

Es soll ggT(a, b) berechnet werden

1. $x \leftarrow a$
   $y \leftarrow b$

2. Wenn $y \neq 0$, dann
   I. $r \leftarrow x - \left\lfloor \frac{x}{y} \right\rfloor * y$
   II. $x \leftarrow y$
   III. $y \leftarrow r$
   IV. Wiederhole Schritt 2

3. $ggT \leftarrow x$

Flussdiagramme (DIN 66001)
Alles ist im Fluss

Flussdiagramme verwenden im Hauptsächlichen folgende Elemente

Symbol	Bedeutung
Start / Stop	Kontrollpunkt
(Pfeil / Linie)	Verbindung zum nächsten Element
Operation	Operation / Tätigkeit
Unter-programm	Unterprogramm ausführen
Ein- und Ausgabe	Ein- und Ausgabe
Verzweigung	Verzweigung / Entscheidung

Flussdiagramme

Und wieder der ggT

Es soll ggT(a, b) berechnet werden

START

x ← a

y ← b

y ≠ 0

Wahr

r ← x − [x / y] * y

x ← y

y ← r

Falsch

ggT = x

ENDE

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Flussdiagramme
Ein anschauliches Beispiel

Struktogramme (DIN 66 261)
Nassi-Shneiderman-Diagramme

Motivation

- Flussdiagramme sind fehleranfällig und unübersichtlich
- Beschränkung der Verzweigung und Reduzierung des Umfangs

Konventionen

- Ein Eingang, ein Ausgang
- Umfang maximal eine Seite
- Rücksprünge werden vermieden
- Pro Modul ein Diagramm

Struktogramme
Anweisungen

Darstellung *eines* Verarbeitungsschrittes

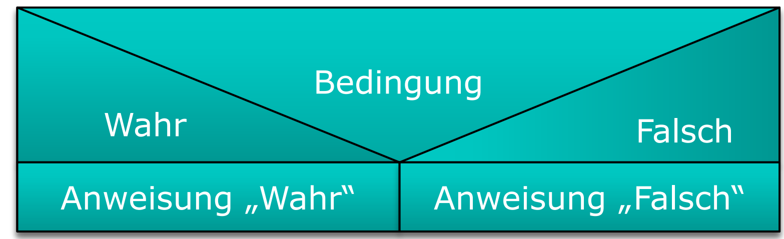
- Beispiel:
  - Wertzuweisung
  - Ein- und Ausgabeanweisungen
  - Unterprogrammaufrufe

Anweisung

Struktogramme
Bedingung und Auswahl

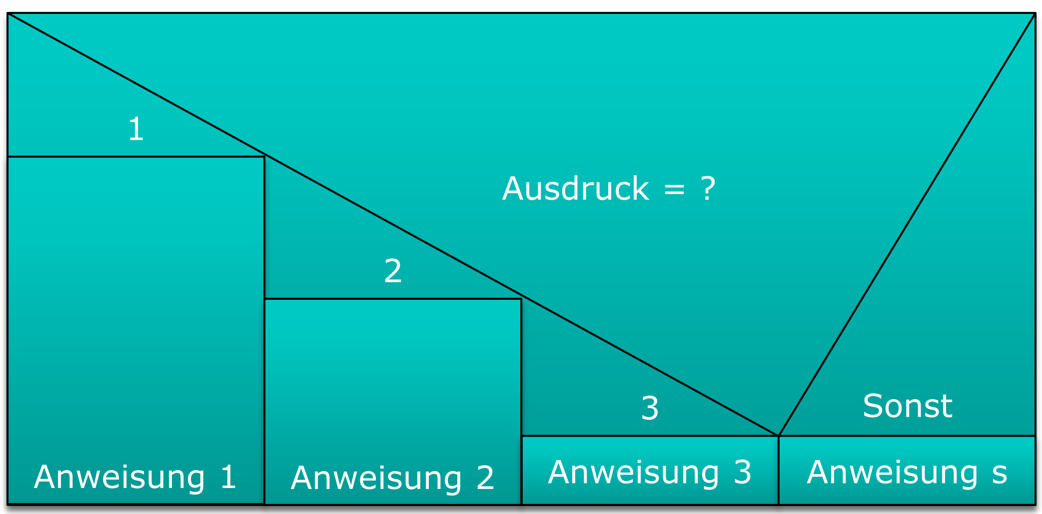
Verzweigungen in Programmen

- „If-Else" in Java



Mehrfachverzweigungen

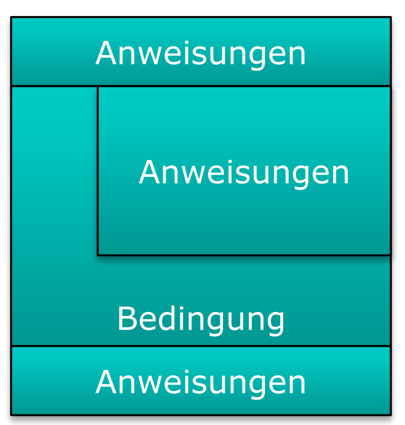
- „Switch-Case" in Java



Struktogramme
Schleifen

Fußgesteuerte Schleifen

- Anweisungen werden mindestens einmal ausgeführt
- „Do-While" in Java

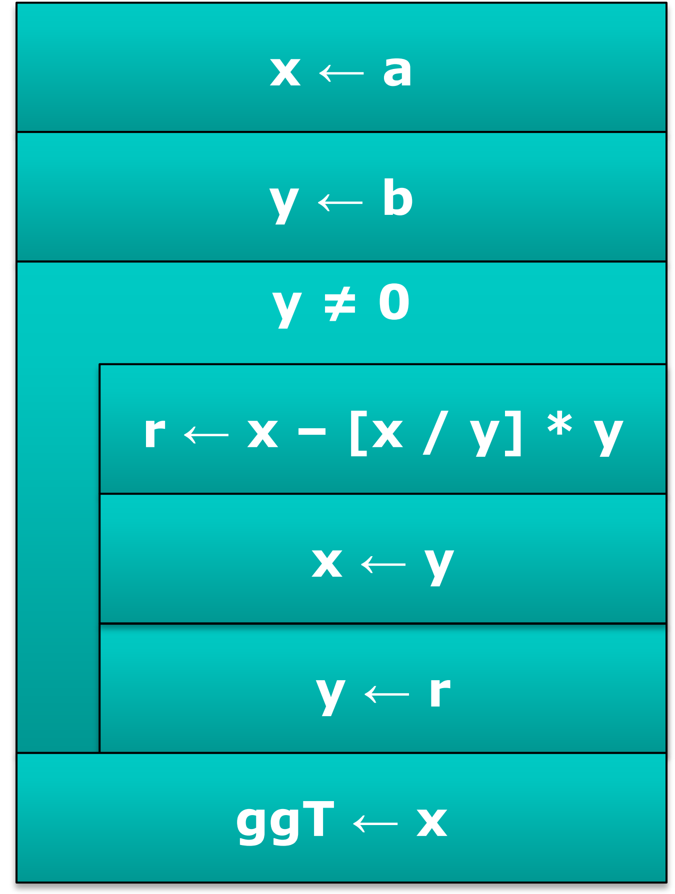


Kopfgesteuerte Schleifen

- Anweisungen werden auch beim ersten Mal erst nach Prüfung der Bedingung ausgeführt
- „While" oder „For" in Java

Struktogramme
Überraschung: der ggT

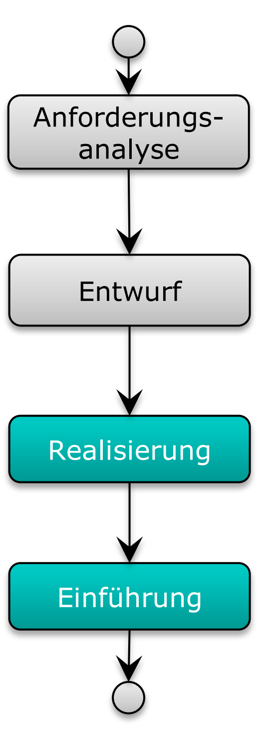
Es soll ggT(a, b) berechnet werden



Wasserfallmodell
Realisierung & Einführung

Realisierung

- Umsetzung des Entwurfs in einer konkreten Programmiersprache
- Gewählte Programmiersprache abhängig vom Einsatzzweck
  - > z.B. Java & C# für Desktopapplikationen
  - > z.B. Assembler & C++ für hardwarenahe Applikationen
  - > z.B. PHP & ASP für Webanwendungen
  - > …



Einführung

- Auslieferung, Installation und Inbetriebnahme der Software beim Kunden
- Genauer Umfang vom Vertrag abhängig

2 – Zahlendarstellungen

Ganze Zahlen, Negative ganze Zahlen, Gleitkommazahlen

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Zahlensysteme
  - > Dezimal-, Binär, Hexadezimalsystem
  - > Umrechnungen zwischen diesen Zahlensystemen
- Zahlendarstellungen
  - > Darstellung ganzer Zahlen
  - > Darstellung von und Rechnen mit negativen ganzen Zahlen
  - > Darstellung von und Rechnen mit Gleitkommazahlen

Zahlen
Ein anschauliches Beispiel

Zahlen
Herr Ober, Zahlen, bitte!

243

1001001

2201

73

133

49

111

1021

3G

4D

Zahlen

Herr Ober, Zahlen, bitte!

243_5 1001001_2 2201_3

73_{10} 133_7 49_{16}

111_8 102_4 $3G_{19}$ $4D_{15}$

Zahlen
Grundlegende Grundlagen

Jede natürliche Zahl lässt sich in eindeutiger Weise darstellen

$$n_B = \sum_{i=0}^{r} B^i * d_i$$

Dabei ist…

- B die **Basis** ($B > 1$)
  - Dezimalsystem: $B = 10$
  - Binärsystem: $B = 2$

- d_i die Ziffer an einer **wertigen Stelle** ($0 \leq d_i < B, d_r > 0$)
  - Im Dezimalsystem: Einer, Zehner, Hunderter, …

- r die „Länge" der Zahl ($r \in \mathbb{N}$)
  - r gibt die höchste wertige Stelle an (Anzahl Stellen = $r + 1$)

Zahlen
Ein Beispiel

Es soll die Zahl 73_{10} näher betrachtet werden.

Informelle Zerlegung der Zahl

- 7 Zehner, 3 Einer

Ein wenig formeller:

- $n_B = B^0 * d_0 + B^1 * d_1$
- $B = 10$
- $d = \{d_0 = 3, d_1 = 7\}$
- $73_B = 10^0 * 3 + 10^1 * 7$
- Damit ist $r = 1$

$$n_B = \sum_{i=0}^{r} B^i * d_i$$

Zahlensysteme
Einige Beispiele

Zahlensystem	Basis	Wertebereich (Ziffern)	Beispiel
dezimal	10	{0,1,2,...,8,9}	73_{10}
binär	2	{0,1}	$1001001_2 = 73_{10}$
hexadezimal	16	{0,...,9,A,...,F}	$49_{16} = 73_{10}$

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Zahlensysteme
Motivation

Warum verwenden Computer andere Zahlensysteme (Binär-, Hexadezimalsystem) und nicht das Dezimalsystem?

- Interne Repräsentation des Binärsystems sehr einfach
  - > Vereinfacht: Strom an = 1, Strom aus = 0
- Binärsystem für den Menschen schnell unlesbar
  - > Hexadezimalsystem guter Kompromiss aus begrenztem Ziffernvorrat und Zahlenlänge
  - > Hexadezimalsystem und Binärsystem sind sehr leicht ineinander umzurechnen
- Beispiel:
  - > Binär: 1101010100010011000100_2
  - > Hexadezimal: $3544C4_{16}$

Zahlensysteme
There and back again

Umrechnung: Ins Dezimalsystem

- Einfaches Aufaddieren der einzelnen Stellen unter Berücksichtigung der jeweiligen Wertigkeit

Beispiele

- Binärsystem
  > $1001001_2 = 1_{10} * (2_{10})^0 + 1_{10} * (2_{10})^3 + 1_{10} * (2_{10})^6 = 1_{10} + 8_{10} + 64_{10} = 73_{10}$

- Hexadezimalsystem
  > $49_{16} = 9 * (16_{10})^0 + 4_{10} * (16_{10})^1 = 9_{10} + 64_{10} = 73_{10}$

Zahlensysteme
There and back again

Umrechnung: Dezimalsystem ins Binärsystem – Methode 1

- Informeller Ansatz:
  1. Suche größte Stelle im Binärsystem, die in die Dezimalzahl „hineinpasst", und subtrahiere diese
  2. Wiederhole das Vorgehen solange mit dem Rest der Subtraktion, bis sich kein Rest mehr ergibt
  3. Fülle - falls nötig - Lücken und niederwertige Stellen mit Nullen auf

- Beispiel: Umrechnung der 73_{10}
  > $73 - 2^6 = 9 \quad \rightarrow 1$
  $9 - 2^5 < 0 \quad \rightarrow 0$
  $9 - 2^4 < 0 \quad \rightarrow 0$
  $9 - 2^3 = 1 \quad \rightarrow 1$
  $1 - 2^2 < 0 \quad \rightarrow 0$
  $1 - 2^1 < 0 \quad \rightarrow 0$
  $1 - 2^0 = 0 \quad \rightarrow 1$

$$73_{10} = 1001001_2$$

Zahlensysteme
There and back again

Umrechnung: Dezimalsystem ins Binärsystem – Methode 2

- Informeller Ansatz:
  1. Teile die Dezimalzahl durch 2 unter Berücksichtigung des Rests
  2. Wiederhole das Vorgehen solange mit dem Ergebnis der Division, bis sich 0 als Ergebnis ergibt

- Beispiel: Umrechnung der 73_{10}
  > $73 : 2 = 36 \quad Rest\ 1$
  $36 : 2 = 18 \quad Rest\ 0$
  $18 : 2 = 9 \quad Rest\ 0$
  $9 : 2 = 4 \quad Rest\ 1$
  $4 : 2 = 2 \quad Rest\ 0$
  $2 : 2 = 1 \quad Rest\ 0$
  $1 : 2 = 0 \quad Rest\ 1$

$$73_{10} = 1001001_2$$

Zahlensysteme
There and back again

Umrechnung: Dezimalsystem ins Hexadezimalsystem

- Umrechnung funktioniert prinzipiell genau, wie die Umrechnung ins Binärsystem
- Umrechnung mit Subtraktion allerdings deutlich komplizierter, deswegen hier Beschränkung auf Umrechnung mit Division
- Beispiel: Umrechnung der 73_{10}
  - > $73 : 16 = 4 \quad Rest\ 9$
    $4 : 16 = 0 \quad Rest\ 4$ ↑

$$73_{10} = 49_{16}$$

Zahlensysteme
There and back again

Umrechnung: Binärsystem ins Hexadezimalsystem

- Vorüberlegung
  - 16 ist selbst zur Basis 2 darstellbar: $\mathbf{16 = 2^4}$
  - Umrechnung nutzt diesen Zusammenhang aus
- Die Umrechnung
  - Die Binärzahl wird von rechts beginnend in Gruppen von 4 Ziffern gruppiert; nötigenfalls mit führenden Nullen auffüllen
  - Jede Gruppe kann direkt in eine hexadezimale Ziffer umgerechnet werden
- Beispiel: Umrechnung von 1001001_2
  - Gruppierung: 0100_2 1001_2
  - Umrechnung: 4_{16} 9_{16}
  - Ergebnis: 49_{16}

Zahlensysteme
There and back again

Umrechnung: Hexadezimalsystem ins Binärsystem

- Die Umrechnung macht sich denselben Zusammenhang zu Nutze, wie die Umrechnung in die Gegenrichtung

- Beispiel: Umrechnung von 49_{16}
  - Betrachtung einzelner Ziffern: 4_{16} 9_{16}
  - Umrechnung: 0100_2 1001_2
  - Ergebnis: 1001001_2

Zahlensysteme
Übungen

Binär	Dezimal	Hexadezimal
101010		
	37	
		AC

Zahlensysteme
Übungen

Binär	Dezimal	Hexadezimal
101010	42	2A
100101	37	25
10101100	172	AC

Negative Zahlen

Warum denn so negativ?

Positive ganze Zahlen lassen sich in einem Computer sehr einfach im Binärsystem darstellen, aber ...

Wie lassen sich negative ganze Zahlen in einem Computer darstellen?

Negative Zahlen
Warum denn so negativ?

Mögliche Lösungen

- Darstellung durch Betrag und Vorzeichen
- Darstellung im 1-Komplement
- Darstellung im 2-Komplement

Negative Zahlen
Betrag und Vorzeichen

Eine Stelle im Dualsystem wird für das Vorzeichen verwendet

- Das höchstwertige Bit (= das Bit ganz links) wird zur Darstellung des Vorzeichens verwendet

- Wert des Bits für eine …
  - … positive Zahl: 0
  - … negative Zahl: 1

- Beispiel: Darstellung von $+73_{10}$ und -73_{10}
  - $+73_{10} = 01001001_2$
  - $-73_{10} = 11001001_2$

Negative Zahlen
Betrag und Vorzeichen

Die Darstellung negativer Zahlen mit einem Vorzeichenbit hat mehrere Nachteile

- Es gibt zwei Darstellungen der Null
  - $+0 = (0, 0, \ldots, 0)$ und $-0 = (1, 0, \ldots, 0)$
  - Es wird Platz für eine Zahl verschwendet

- Das Rechnen mit diesen Zahlen funktioniert nicht gut
  - Die Subtraktion lässt sich nicht auf die Addition zurückführen
  - Zusätzlicher Aufwand für eine eigene Subtraktion
  - Beispiel: $0001_2 - 0001_2 \quad \stackrel{?}{=} \quad 0001_2 + (-0001_2)$

$$\begin{array}{ll} & 0001_2 \\ - & 0001_2 \\ = & 0000_2 \end{array} \qquad\qquad \begin{array}{ll} & 0001_2 \\ + & 1001_2 \\ = & 1010_2 \end{array}$$

Negative Zahlen
Das 1-Komplement

Das 1-Komplement („logische Komplementbildung“) einer Binärzahl wird durch stellenweises invertieren gebildet

- Die Subtraktion kann in diesem Fall auf die Addition zurückgeführt werden
- Beispiel: $0001_2 - 0001_2 \stackrel{?}{=} 0001_2 + (-0001_2)$

$$\begin{array}{cr} & 0001_2 \\ - & 0001_2 \\ = & 0000_2 \end{array} \qquad \begin{array}{cr} & 0001_2 \\ + & 1110_2 \\ = & 1111_2 \end{array}$$

- Auch in diesem Fall gibt es wieder zwei Nullen
  - $+0 = (0, 0, \dots, 0)$ und $-0 = (1, 1, \dots, 1)$

Negative Zahlen
Das 1-Komplement

Aber auch das Rechnen mit dem 1-Komplement ist manchmal umständlich

- Tritt bei einer Subtraktion ein Überlauf auf, so muss dieser ignoriert werden und stattdessen eine „1“ addiert werden
- Beispiel: Es soll $12_{10} - 8_{10}$ berechnet werden
  - Umrechnung
    - $12_{10} = 1100_2$
    - $-8_{10} = 0111_2$
  - Berechnung
    - $$\begin{array}{rr} & 1100_2 \\ + & 0111_2 \\ = & 10011_2 \end{array}$$
  - Beachtung des Sonderfalls
    - Überlauf ignorieren: $10011_2 \rightarrow 0011_2$
    - „1“ addieren: $0011_2 + 0001_2 = 0100_2$

Negative Zahlen
Das 2-Komplement

Das 2-Komplement („arithmetische Komplementbildung“) einer Binärzahl wird durch stellenweises invertieren und anschließendes addieren einer „1“ gebildet

- Die Subtraktion kann auch in diesem Fall auf die Addition zurückgeführt werden

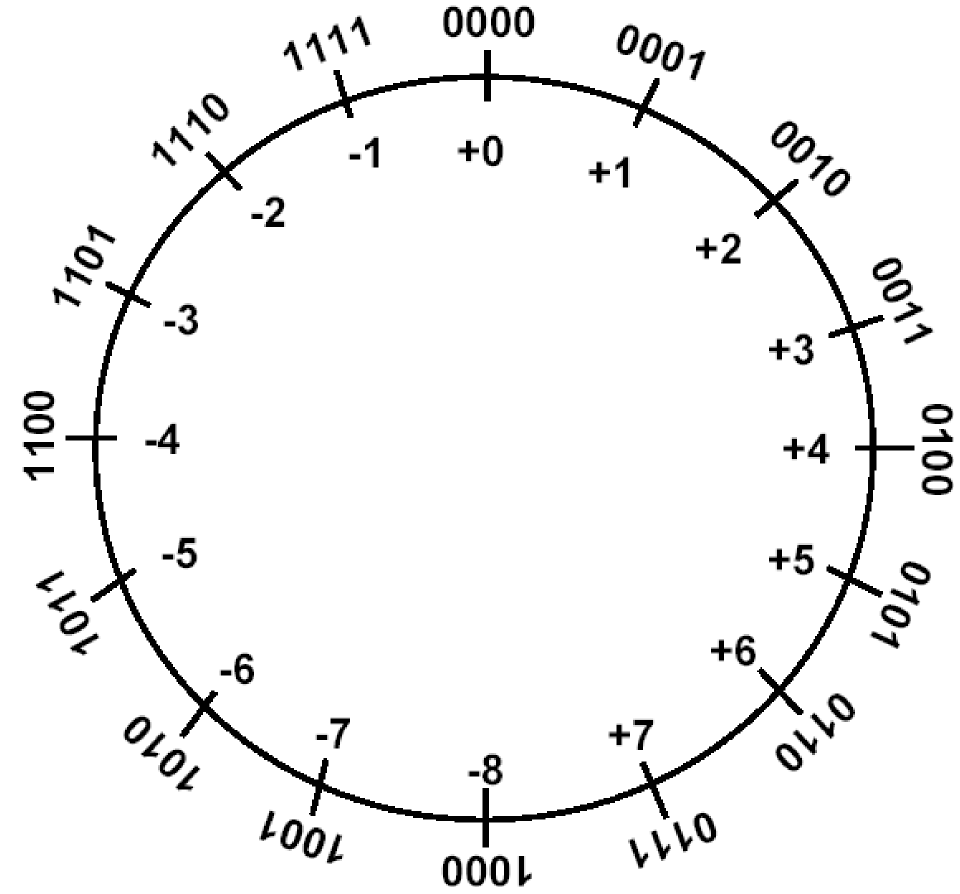
- Beispiel: $0001_2 - 0001_2 \stackrel{?}{=} 0001_2 + (-0001_2)$

$$\begin{array}{cr} & 0001_2 \\ - & 0001_2 \\ = & 0000_2 \end{array} \qquad \begin{array}{cr} & 0001_2 \\ + & 1111_2 \\ = & 10000_2 \end{array}$$

- Anmerkungen
  - > Es existiert nur noch eine Null
  - > Das Vorzeichen ist am führenden Bit erkennbar
  - > Der Überlauf wird ignoriert

Negative Zahlen
Das 2-Komplement

Das 2-Komplement kann wie folgt grafisch dargestellt werden



Negative Zahlen
Übung

Betrag & Vorzeichen	1-Komplement	2-Komplement
10001		
	1000	
		1011

Negative Zahlen
Übung

Betrag & Vorzeichen	1-Komplement	2-Komplement
10001	1110	1111
10111	1000	1001
10101	1010	1011

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Kommazahlen
Bart, komma her! – Komma Du her! – Achja?!

Es gibt prinzipiell zwei verschiedene Arten von Kommazahlen in einem Computersystem

- Festkommazahlen
  - > Zahlen mit einer festen Anzahl von Stellen vor und hinter dem Komma bei fester Anzahl von Bits
  - > Darstellung über negative Zweierpotenzen
  - > Addition und Subtraktion funktioniert analog zur Addition und Subtraktion ganzer Zahlen
  - > Beispiel: $2.5_{10} + 0.25_{10} = 10.10_2 + 00.01_2 = 10.11_2$

- Gleitkommazahlen
  - > Zahlen mit variabler Genauigkeit, d.h. Stellen hinter dem Komma, bei fester Anzahl von Bits
  - > Addition und Subtraktion müssen speziell definiert werden

Kommazahlen
Gleitkommazahlen und IEEE

Die Darstellung von Gleitkommazahlen mit Bits und Bytes wird über die IEEE 754 Norm geregelt

- Es gibt einfache Genauigkeit mit 32 Bits und doppelte Genauigkeit mit 64 Bits
- Die Bits werden für verschiedene Teile der Zahl verwendet
  - Vorzeichen S
  - Exponent E
  - Mantisse M
- Die Darstellung erfolgt immer zur Basis 2 in folgender Form

$$x = s * m * 2^e$$

Kommazahlen
Gleitkommazahlen und IEEE

Aus technischen Gründen werden nicht die Werte gespeichert, aus denen die Zahl tatsächlich berechnet wird

- Die „echten" Werte müssen aus den gespeicherten Werten $\boldsymbol{S}$, $\boldsymbol{E}$ und $\boldsymbol{M}$ und einem festen Wert $\boldsymbol{b}$ berechnet werden

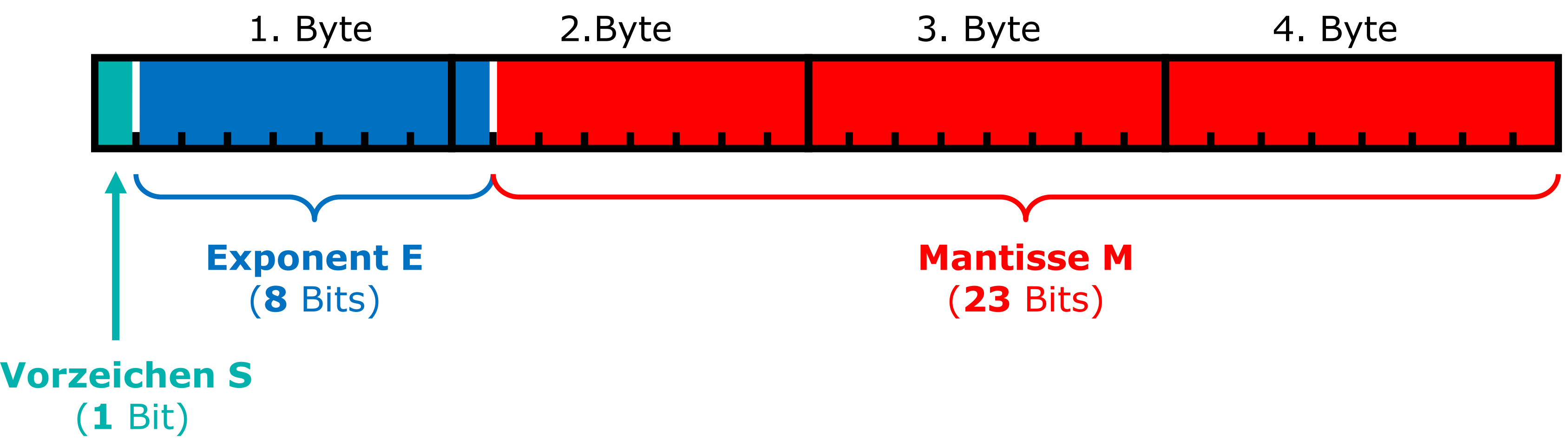
$$x = s * m * 2^{e}$$

$$\boldsymbol{s} = (-\boldsymbol{1})^{\boldsymbol{S}}$$

$$\boldsymbol{e} = \boldsymbol{E} - \boldsymbol{b}$$

$$\boldsymbol{m} = \boldsymbol{1},\boldsymbol{M}$$

Gleitkommazahlen
Einfache Genauigkeit



- Der Exponent kann Werte von -126 bis +127 annehmen
  - Die Werte -127 und +128 sind für Spezialfälle reserviert
- Der Biaswert b ist 127

Gleitkommazahlen
Doppelte Genauigkeit

1. Byte 2.Byte 3. Byte 4. Byte . . . 8. Byte

Exponent E (**11** Bits)

Mantisse M (**52** Bits)

Vorzeichen S (**1** Bit)

- Der Exponent kann Werte von -1022 bis + 1023 annehmen
  - > Die Werte -1023 und + 1024 sind für Spezialfälle reserviert
- Der Biaswert b ist 1023

Gleitkommazahlen
Normalisierung

Normalisierung bringt die Mantisse einer Gleitkommazahl in ein bestimmtes Format

- Vorüberlegungen
  - Die Mantisse selbst ist eigentlich auch eine Gleitkommazahl
  - Ein Format zur Speicherung von Gleitkommazahlen zu definieren, welches selbst eine Gleitkommzahl speichern muss, ist unsinnig
- Idee
  - Das Komma der Mantisse kann beliebig verschoben werden, sofern der Exponent der Gleitkommazahl entsprechend angepasst wird
- Beispiel aus dem Dezimalsystem
  - $579_{10} * (10_{10})^0 = 57{,}9_{10} * (10_{10})^1 = 5{,}79_{10} * (10_{10})^2$ usw.

Gleitkommazahlen
Normalisierung

Das Komma der Mantisse wird so verschoben, dass nur noch eine einzelne 1 direkt vor dem Komma steht

$$\boldsymbol{m} = 1,\boldsymbol{M}$$

- Da dies (zunächst) für alle Gleitkommazahlen nach der IEEE 754 Norm gilt, muss die 1 vor dem Komma der Mantisse nicht mehr explizit gespeichert werden

- Es werden in den entsprechenden Bits der Mantisse also nur die Nachkommastellen $\boldsymbol{M}$ der Mantisse gespeichert

- Der Exponent muss bei der Normalisierung der Mantisse entsprechend angepasst werden

Gleitkommazahlen
Umwandlung

Prinzipiell wird bei der Umwandlung einer Dezimalzahl in das IEEE 754 Format wie folgt vorgegangen

1. Umwandlung der Dezimalzahl in eine binäre Festkommazahl ohne Vorzeichen

2. Normalisieren und Bestimmen des Exponenten

3. Vorzeichen-Bit bestimmen

4. Gleitkommazahl zusammensetzen

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Gleitkommazahlen
Umwandlung – Beispiel

Die Zahl 73.5_{10} soll in eine Gleitkommazahl nach IEEE 754 Norm in einfacher Genauigkeit umgewandelt werden

1. Umwandlung der Dezimalzahl in eine binäre Festkommazahl ohne Vorzeichen
   - $\mathbf{73.5_{10} = 1001001.1_2}$
2. Normalisieren und Bestimmen des Exponenten
   - $\mathbf{1001001.1_2 * 2^0 = 1.0010011_2 * 2^6}$
   - $\boldsymbol{M} = \mathbf{00100110000000000000000_2}$
   - $\boldsymbol{E} = \mathbf{6_{10} + 127_{10} = 110_2 + 1111111_2 = 10000101_2}$
3. Vorzeichen-Bit bestimmen
   - $\boldsymbol{S} = \mathbf{0}$
4. Gleitkommazahl zusammensetzen
   - **01000010 10010011 00000000 00000000**

Gleitkommazahlen
Spezialfälle

Anmerkung

- r ist die Anzahl der Exponenten-Bits

Exponent E	Mantisse M	Bedeutung
$E = 0$	$M = 0$	$+/- 0$
$E = 0$	$M \neq 0$	$+/- 0.M * 2^{1-b}$
$0 < E < 2^r-1$	$M \neq 0$	$+/- 1.M * 2^{E-b}$
$E = 2^r-1$	$M = 0$	$+/- \infty$
$E = 2^r-1$	$M \neq 0$	NaN

Gleitkommazahlen
Addition und Subtraktion

Addition und Subtraktion zweier Gleitkommazahlen im IEEE 754 Format

- Beide Zahlen müssen denselben Exponenten haben, ansonsten muss zunächst eine Zahl umgerechnet werden
  - Die betragsmäßig kleinere Zahl wird umgerechnet

- Haben beide Zahlen denselben Exponenten kann die Mantisse einfach addiert oder subtrahiert werden

- Die resultierende Zahl muss – wenn möglich und nötig – wieder normalisiert werden

Gleitkommazahlen

Ein beispielhaftes Beispiel – Addition

Es seien zwei Gleitkommazahlen a und b gegeben

- a = |0|10000001|(1)10010000000000000000000
- b = |0|01111111|(1)10110001000100000000000

Die Zahl b ist kleiner und muss daher umgerechnet werden

- b = |0|10000001|(0)01101100010001000000000

Addition

- a + b = |0|10000001|(1)11111100010001000000000

Gleitkommazahlen
Bezug zur Programmiersprache

Datentyp	Größe	Wrapper-Klasse	Wertebereich	Beschreibung
byte	8 Bit	java.lang.Byte	−128 ... +127	2-Komplement
short	16 Bit	java.lang.Short	−32.768 ... +32.767	2-Komplement
int	32 Bit	java.lang.Integer	−2.147.483.648 ... +2.147.483.647	2-Komplement
long	64 Bit	java.lang.Long	−9.223.372.036.854.775.808 ... +9.223.372.036.854.775.807	2-Komplement
float	32 Bit	java.lang.Float	±1,4E−45 ... ±3,4E+38	Gleitkommazahl (IEEE 754)
double	64 Bit	java.lang.Double	±4,9E−324 ... ±1,7E+308	Gleitkommazahl (IEEE 754)

Gleitkommazahlen
Übung

Festkommazahl	Gleitkommazahl
01011001.010	
	1 10000110 10110010010000000000000

Gleitkommazahlen
Übung

Festkommazahl	Gleitkommazahl
01011001.010	0 10000101 01100101000000000000000
-11011001.001	1 10000110 10110010010000000000000

3 – Codierung

Einfache Codes, Spezialisierte Codes

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Einfache Codes
  - Darstellung von Ziffern und Zeichen im Rechner
  - Gängige Standards

- Spezialisierte Codes
  - Motivation für spezielle Codierungen
  - Optimale Codes
  - Robuste Codes

Codes
Ein anschauliches Beispiel

Was ist ein Code?

Und was hat dieser Da Vinci damit zu tun?

Morsecode

The Da Vinci Code

ASCII-Code

Zugangscode

Barcode

Genetischer Code

Sourcecode

QR-Code

Geheimcode

Codes
Ein anschauliches Beispiel

Was ist ein Code?
Eigenschaften

Ein Code ist …

- … eine Formulierung von **Information**
- … eine Vereinbarung über einen **Satz von Symbolen**
- … eine **Abbildung**, die Symbolen eines *Alphabets* eindeutige Symbole eines eventuell anderen *Alphabets* zuordnet
  - Ist die Abbildung **eineindeutig**, so ist der Code **entzifferbar**

Was ist ein Code?
Sinntragende Zeichen / Symbole

Code dient dazu Informationen zu transportieren

- Es gibt sinntragende Zeichen, aus denen die Information vom Empfänger des Codes gewonnen werden kann
  - > Werden auch Symbole genannt
- Ein Code bildet zwischen zwei sogenannten Alphabeten aus sinntragenden Zeichen (eindeutig) ab
- Beispiele
  - > Ziffern, Großbuchstaben, Kleinbuchstaben, Sonderzeichen, Hieroglyphen, Kanjis, …

ASCII
Ein einfacher Code

Der ASCII-Code stellt Buchstaben, Ziffern und Sonderzeichen anhand einer binären Zahl mit 7 Bits dar

- Vorteile
  - Einfach zu Codieren und Decodieren
  - Kurze Darstellung
  - Wenig Speicherplatz
- Nachteile
  - Menge der darstellbaren Zeichen (zu) klein

ASCII
Codierungstabelle

Anmerkung

- 128 Zeichen darstellbar: 00 – 7F

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ASCII
Ein beispielhaftes Beispiel

Es soll der Ausdruck „MILKY GREEN“ codiert werden:

- Es muss zu jedem Buchstaben und jedem Sonderzeichen (= Leerzeichen) die entsprechende Zahl aus der Codierungstabelle gelesen werden

- In diesem Fall sollen die Zahlen zur Basis 10 angegeben werden

M	I	L	K	Y	␣	G	R	E	E	N
77	73	76	75	89	32	71	82	69	69	78

ASCII
Übung

String	ASCII Binär	ASCII Hexadezimal
MATSE		
	1000101 1000100 1010110	
		43 2B 2B

ASCII
Übung

String	ASCII Binär	ASCII Hexadezimal
MATSE	1001101 1000001 1010100 1010011 1000101	4D 41 54 53 45
EDV	1000101 1000100 1010110	45 44 56
C++	1000011 0101011 0101011	43 2B 2B

Unicode
Motivation

Mit den 7 Bits von ASCII kann man nur sehr wenige Zeichen darstellen.

Was ist z.B. mit Umlauten?

Unicode
Ein erweiterter Code

Unicode ist ein Zeichensatz mit 32 Bits, der als weltweiter Standard geplant ist

- Ziel ist ein eindeutiger, digitaler Code für jedes sinntragende Zeichen weltweit
  - > Globaler Informationsaustausch wird möglich
- Die digitalen Codes für die Zeichen des ASCII-Codes sind weiterhin dieselben
  - > Führenden Nullen werden hinzugefügt
- Aufbau des Codes
  - > Zunächst 16-Bit Codierung mit fester Länge, später 32 Bit
  - > Maximal **65.536** bzw. **4.294.967.296** Zeichen darstellbar
  - > Notation im Hexadezimalsystem **U+(XX)XXXXXX**
    - > Dabei geben die letzten 2 Byte eine Position in einer Plane (Ebene) an
    - > Die Angabe der Plane ist dabei optional (Standard = 00)

Unicode
Basic Multilingual Plane

Anmerkung

- Das erste Byte adressiert den Block innerhalb der Plane, das zweite Byte ein Zeichen innerhalb des Blocks

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

- Lateinische Schriften und Symbole
- Lautschriften
- Andere europäische Schriften
- Nahost- und Südwestasiatische Schriften
- Afrikanische Schriften
- Südasiatische Schriften
- Südostasiatische Schriften
- Ostasiatische Schriften
- CJK-Ideogramme
- Kanadische Silben
- Symbole
- Diakritika
- UTF-16-Surrogates und privater Nutzungsbereich
- Verschiedene Zeichen
- Nicht belegte Codebereiche

Unicode
Anmerkungen

- Zeichen, die in unterschiedlichen Kontexten mehrfach vorkommen werden im Unicode auch mehrfach codiert
  - z.B. gibt es das „A" im lateinischen und im griechischen Alphabet
- Es gibt nur 17 Planes
  - Die 32 Bit werden also (noch) nicht vollständig zur Zeichencodierung benutzt
- Näheres zu ASCII, Unicode und UTF-8
  - http://www.youtube.com/watch?v=MijmeoH9LT4

Unicode Transformation Format
Ein Code mit variabler Länge

Um die Unicode-Zeichen im Rechner zur Weiterverarbeitung darstellen zu können gibt es die UTF-Codierung

Es gibt mehrere UTF-Standards

- **UTF-32** je Zeichen 4 Bytes (32 Bit)
  - feste Länge, einfache Codierung, hoher Speicherbedarf
- **UTF-16** je Zeichen 2 oder 4 Bytes (16 – 32 Bit)
  - Variable Länge, hoher Speicherbedarf
- **UTF-8** je Zeichcen 1 bis 4 Byte (8 – 32 Bit)
  - Variable Länge, niedriger Speicherbedarf

UTF-8
Ein Code mit variabler Länge

UTF-8 ist dazu geeignet, den Speicherplatz eines Zeichens den Anforderungen anzupassen

- Handelt es sich bei einem Zeichen um ein ASCII-Zeichen wird ein Byte zur Codierung benötig
  - > (0 – 127) → 0xxx xxx
- Handelt es sich um ein anderes Zeichen werden 2 bis 4 Bytes zur Codierung benötigt
  - > Das erste Byte beginnt mit einer Anzahl von Einsen, die der Anzahl von zu verwendenden Bytes entspricht, gefolgt von einer 0
  - > Jedes folgende Byte beginnt mit **10**
  - > Beispiel
    - > Ein 3 Byte-Code sieht wie folgt aus: **1110 xxxx | 10xx xxxx | 10xx xxxx**
    - > Damit verbleiben 16 Bits zur Darstellung eines Unicode Zeichens

UTF-8
Übung

Zeichen / Dezimalwert	Unicode	UTF-8
A / 65		
ä / 257		
Ў / 1038		

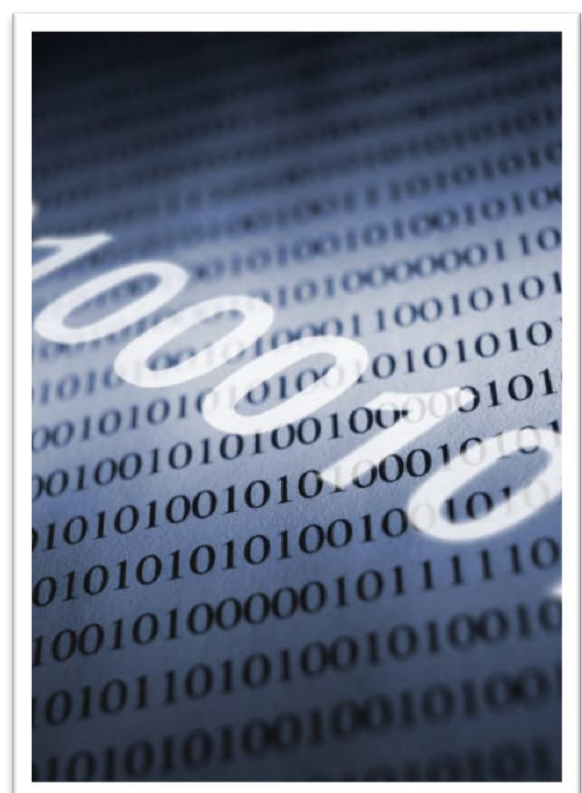
UTF-8
Übung

Zeichen / Dezimalwert	Unicode	UTF-8
A / 65	U+0041	41
ä / 257	U+0101	C4 81
Ў / 1038	U+040E	D0 8E

Spezielle Codes
Wozu überhaupt?

Es gibt verschiedene Gründe für spezielle Codierungen

- Technische Randbedingungen
- Speicheroptimierung
- Robuste Codierung
  - > Fehler erkennen & korrigieren



Spezielle Codes
Technische Randbedingungen

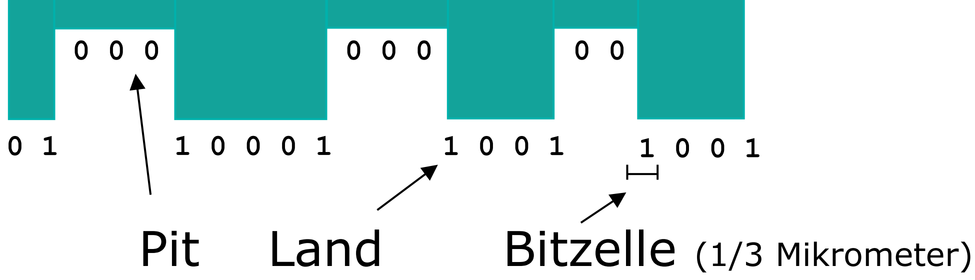
Es kann technische Randbedingungen geben, die die Verwendung spezieller Codes nötig machen

- Möglichkeiten zur Optimierung
  - z.B. Binärcode auf dem Computer
- Ungenauigkeiten
  - z.B. Daten auf einer CD / DVD / Blu-ray
- Bestimmte Formate
  - z.B. QR-Code zur Erkennung mit Smartphones

Spezielle Codes
Technische Randbedingungen – Die CD

Technische Realisierung einer CD

- Eine CD hat eine reflektierende Oberfläche
  - > Pits = (eingebrannte) Vertiefungen
  - > Lands = Erhebungen
- Codierung
  - > Pits & Lands = 0
  - > Übergänge = 1



© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Spezielle Codes

Technische Randbedingungen – Die CD

Technische Randbedingungen

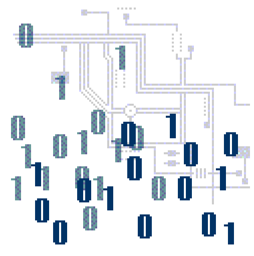
- Es müssen mindestens 2 Nullen und höchsten 11 Nullen zwischen zwei Einsen folgen
  - 8-Bit-Codes sind nicht verwendbar
  - Stattdessen wird ein 14-Bit-Code benutzt
    - EFM = Eight-To-Fourteen-Modulation
    - Jedem 8-Bit-Code wird anhand einer Tabelle ein 14-Bit-Code zugewiesen

Dezimal	Binar	EFM
0	0	1001000100000
1	1	10000100000000
2	10	10010000100000
3	11	10001000100000
4	100	1000100000000
5	101	100010000
6	110	10000100000
7	111	100100000000
8	1000	1001001000000
9	1001	10000001000000
10	1010	10010001000000
11	1011	10001001000000
12	1100	1000001000000
13	1101	1000000
14	1110	10001000000
15	1111	100001000000
16	10000	10000000100000

Spezielle Codes
Speicheroptimierung

Codes zur Speicheroptimierung komprimieren Daten

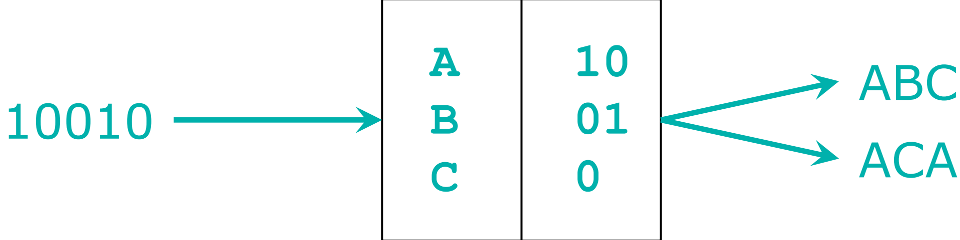
- Ziele der Kompression
  - > Möglichst Kompakte Darstellung
  - > Verlustfreie Codierung (nicht immer)
- Motivation
  - > Codes mit fester Länge (z.B. ASCII) ineffizient
- Lösungsansatz
  - > Kurzer Code für häufig vorkommende Zeichen
  - > Langer Code für seltene Zeichen



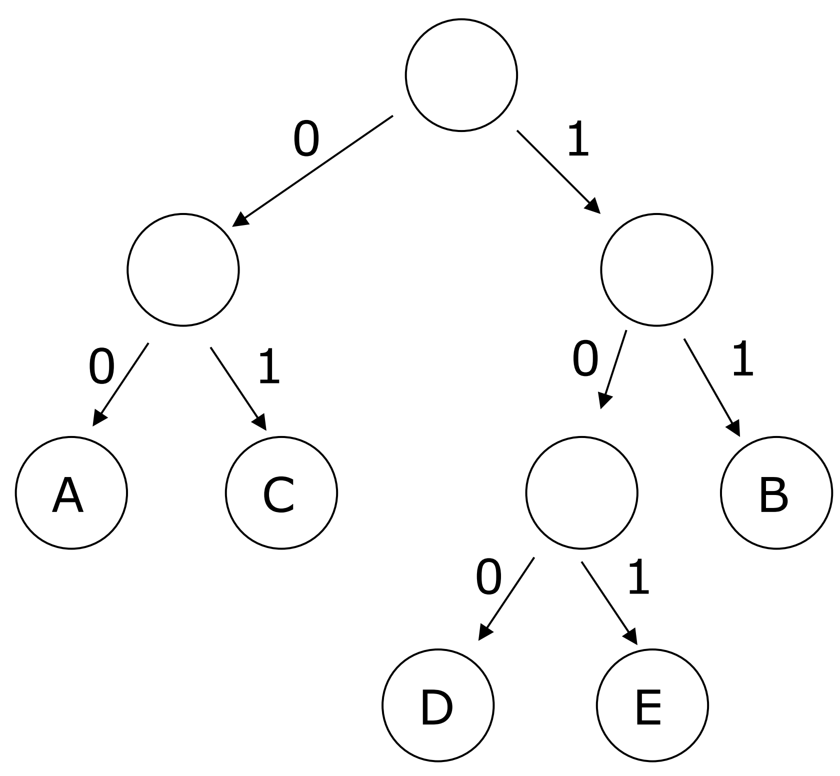
Spezielle Codes
Speicheroptimierung – Präfixfreiheit

Wenn Codes unterschiedliche Länge haben, müssen diese präfixfrei sein

- Kein Codewort ist der Anfang eines anderen Codewortes
- Beispiel für nicht-präfixfreie Codierung



Spezielle Codes
Speicheroptimierung – Präfixfreiheit



A	B	C	D	E
00	11	01	100	101

Spezielle Codes
Speicheroptimierung – Huffman-Code

Der Huffman-Code ist eine Variante eines präfixfreien Codes

- Eigenschaften
  - > Code möglicherweise für jeden Anwendungsfall anders
  - > Basiert auf den Auftrittswahrscheinlichkeiten eines Zeichens
  - > Die mittlere Codelänge wird minimiert

- Algorithmus
  1. Schreibe Symbole mit Wahrscheinlichkeiten als Wald
  2. Fasse die beiden Bäume mit der geringsten Wahrscheinlichkeit in einem neuen Baum zusammen
     - > Die Wahrscheinlichkeit für den neuen Zweig ergibt sich durch Addition
  3. Wiederhole bis nur noch ein Baum existiert

Spezielle Codes
Speicheroptimierung – Huffman-Code

Beispiel

7 6 3 5 15
A B C D E

7 6 8 15
A B C D E

13 8 15
A B C D E

21 15
A B C D E

36
A B C D E

Spezielle Codes
Übung

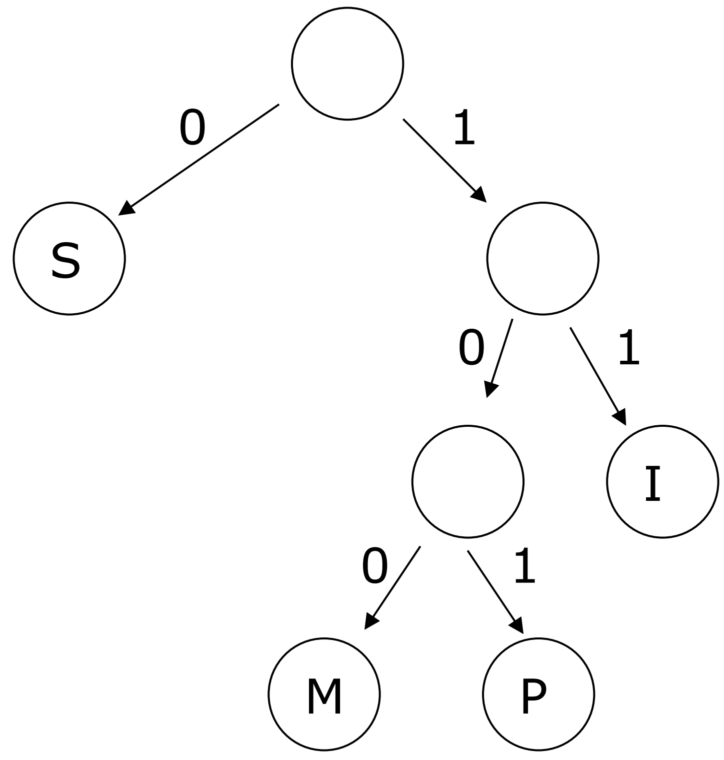
Codieren Sie das folgende Wort nach dem Huffmann-Code:

MISSISSIPPI

Spezielle Codes
Übung

Codieren Sie das folgende Wort nach dem Huffmann-Code:

MISSISSIPPI



Spezielle Codes
Robuste Codierung

Manche Übertragungsmedien sind fehleranfällig

- Fehler müssen erkannt und im Optimalfall korrigiert werden können

- Lösungsvarianten
  - Mehrfache Übertragung der Daten
    - Sehr einfach zu realisieren
    - Es werden sehr viele Daten übertragen

  - Prüfsummenverfahren
    - Es werden zusätzliche Informationen übertragen, die von den zu übertragenden Daten abhängig sind

Spezielle Codes
Robuste Codierung – Prüfsummen

Ein einfaches Prüfsummenverfahren

- Ein einfaches Prüfsummenverfahren ist die sogenannte **Parität**
  - > Die zusätzliche Information bezieht sich auf die Anzahl der zu übertragenden Einsen im Binärcode
  - > Es wird nur ein Bit zusätzlich benötigt: Das **Paritätsbit**
    - > Ist die Anzahl der Einsen ungerade ist das Paritätsbit 1, sonst 0
  - > Es kann allerdings nur **ein** Fehler gefunden und **nicht** korrigiert werden
  - > Beispiel
    - > Binärdaten: 1111 0000
    - > Prüfsumme: 0 (da vier Einsen)

Spezielle Codes
Robuste Codierung – Hamming-Code

Um Fehler besser erkennen und vor allem korrigieren zu können kann der Hamming-Code verwendet werden

- Ein Codewort im Hamming-Code ist immer $N = 2^k - 1$ Bits groß
  - Davon sind k Bits für die Parität und werden immer an Positionen, deren Nummer eine 2er-Potenzen ist, gesetzt
  - Die restlichen $n = N - k$ Bits können Informationen enthalten
  - Beispiel

Bitposition	1	2	3	4	5	6	7
	Parität	Parität	Daten	Parität	Daten	Daten	Daten

Spezielle Codes
Robuste Codierung – Hamming Code

Berechnung der Paritätsbits

- Ein Paritätsbit verknüpft $\mathbf{2^{k-1}-1}$ der Datenbits mit einer XOR-Verknüpfung („die Hälfte der Bits" = Verknüpfte Datenbits + Paritätsbit)
  - Das bedeutet, dass die Anzahl der Einsen in der „Hälfte der Bits" gerade ist
- Welche Datenbits werden in einem Paritätsbit verknüpft?
  - Alle Bits die eine „1" im entsprechenden Polynomialcode haben
- Was bedeutet das?
  - Der Polynomialcode des Paritätsbits an Stelle 2^l besteht aus einer Abfolge von 2^l Einsen, danach folgen 2^l Nullen, danach wieder 2^l Einsen usw.
  - Der Polynomialcode beginnt erst am zugehörigen Bit, vorher stehen Nullen

Spezielle Codes
Robuste Codierung – Hamming Code

Beispiel für $N = 7$

- 3 Paritätsbits p_n an Stellen 1, 2, 4
- 4 Datenbits d_n an Stellen 3, 5, 6, 7

Bitposition	1	2	3	4	5	6	7
Aufteilung	Parität	Parität	Daten	Parität	Daten	Daten	Daten
Daten	p_1	p_2	**0**	p_3	**1**	**1**	**0**
Polynomialcode l=0	1	0	1	0	1	0	1
Polynomialcode l=1	0	1	1	0	0	1	1
Polynomialcode l=2	0	0	0	1	1	1	1

- Berechnung der Paritätsbits
  - $p_1 = 0 \oplus 1 \oplus 0 = 1$
  - $p_2 = 0 \oplus 1 \oplus 0 = 1$
  - $p_3 = 1 \oplus 1 \oplus 0 = 0$

Spezielle Codes
Robuste Codierung – Hamming Code

Erkennung von Fehlern

- Zum Erkennen von Fehlern muss der Empfänger einer codierten Nachricht selbst noch einmal die Paritätsbits berechnen

- Anschließend werden die übertragenen Paritätsbits mit den berechneten verglichen
  - Alle Bits sind gleich
    - Die Information ist korrekt übertragen worden
  - Es ist ein Bit unterschiedlich
    - Das Paritätsbit ist falsch übertragen worden
  - Es sind zwei oder mehr Bits unterschiedlich
    - Anhand der Kombinationen aus Datenbits und Paritätsbits kann das defekte Datenbit erkannt werden

Spezielle Codes
Robuste Codierung – Hamming Code

Beispiel für Fehlerkorrektur – 1

Bitposition	1	2	3	4	5	6	7
Aufteilung	Parität	Parität	Daten	Parität	Daten	Daten	Daten
Übertragene Daten	**1**	**1**	**0**	**0**	**1**	**1**	**0**
Empfangene Daten	**1**	**1**	**0**	**0**	**1**	**1**	**1**
Berechnete Bits	**0**	**0**	-	**1**	-	-	-
Polynomialcode l=0	1	0	1	0	1	0	1
Polynomialcode l=1	0	1	1	0	0	1	1
Polynomialcode l=2	0	0	0	1	1	1	1

- Berechnung der Paritätsbits
  - > $p_1 = 0 \oplus 1 \oplus 1 = 0$ ungleich
  - > $p_2 = 0 \oplus 1 \oplus 1 = 0$ ungleich
  - > $p_3 = 1 \oplus 1 \oplus 1 = 1$ ungleich

Bit 7 ist fehlerhaft

Spezielle Codes
Robuste Codierung – Hamming Code

Beispiel für Fehlerkorrektur – 2

Bitposition	1	2	3	4	5	6	7
Aufteilung	Parität	Parität	Daten	Parität	Daten	Daten	Daten
Übertragene Daten	**1**	**1**	**0**	**0**	**1**	**1**	**0**
Empfangene Daten	**1**	**1**	**0**	**0**	**0**	**1**	**0**
Berechnete Bits	**0**	**1**	**-**	**1**	**-**	**-**	**-**
Polynomialcode l=0	1	0	1	0	1	0	1
Polynomialcode l=1	0	1	1	0	0	1	1
Polynomialcode l=2	0	0	0	1	1	1	1

- Berechnung der Paritätsbits
  - > $p_1 = 0 \oplus 0 \oplus 0 = 0$ ungleich
  - > $p_2 = 0 \oplus 1 \oplus 0 = 1$ gleich
  - > $p_3 = 0 \oplus 1 \oplus 0 = 1$ ungleich

Bit 5 ist fehlerhaft

Spezielle Codes
Übung

Wie lautet der Hamming-Code zu den 4 Datenbits 1001?

Wo ist der Fehler im Code 1010000?

Wo ist der Fehler im Code 0011000?

Spezielle Codes
Übung

Wie lautet der Hamming-Code zu den 4 Datenbits 1001?

- 0011001

Wo ist der Fehler im Code 1010000?

- Das zweite Paritätsbit ist falsch

Wo ist der Fehler im Code 0011000?

- Das letzte Datenbit ist falsch

4.1 – Formale Sprachen

Syntax, Semantik, Grammatik

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Aufbau und Anwendung von formalen Grammatiken
- Bezug zu Programmiersprachen

Sprachen
Ein anschauliches Beispiel

Was ist eine Sprache?
Und was hat dieser Da Vinci damit zu tun?

Fremdsprache

Formale Sprache

Muttersprache

Bildersprache

Umgangssprache

Gebärdensprache

Programmiersprache

Körpersprache

Konstruierte Sprache

Geheimsprache

Lautsprache

Weltsprache

Formale Sprachen
Grundlegende Grundlagen

Eine **formale Sprache** …

- … dient nicht zur Kommunikation, sondern zur eindeutigen Darstellung von Informationen
  - z.B. Programmiersprachen
- … hat eine Grammatik, die den Aufbau der Sprache regelt und deren Syntax festlegt
- … hat ein Alphabet erlaubter Zeichen
- … besteht aus Worten, die aus den Zeichen des Alphabets aufgebaut sind

Formale Sprachen
Beispiel einer Grammatik

```
1.  Satz          := Subjekt " " Verb " " Präposition
                       " " Nomen Ende

2.  Subjekt       := Nomen

3.  Nomen         := Artikel Substantiv

4.  Artikel       := "der" | "die" | "das"

5.  Substantiv    := "mann" | "frau" | "haus"

6.  Verb          := "geht"

7.  Präposition   := "in"

8.  Ende          := "."
```

Formale Sprachen
Grammatik – Backus-Naur-Form (BNF)

Eine Grammatik beschreibt bzw. regelt die Syntax einer formalen Sprache

- Terminale Symbole
  - > Elemente aus Zeichen des Alphabets der Sprache
- Nicht-terminale Symbole
  - > Symbole, die nach den Syntaxregeln aus anderen Symbolen zusammengesetzt sind
- Grundidee
  - > Angabe der terminalen Symbole
  - > Rekursive Definition der nicht-terminalen Symbole

Formale Sprachen
Grammatik –BNF

- Symbole werden durch eine Zuweisung [:=] definiert
  > Non-terminale Symbole beziehen sich auf mindestens ein anderes Symbole
  > Terminale Symbole beziehen sich auf kein anderes Symbol

- Verknüpfungen bei einer Zuweisung
  > Ein Zwischenraum bedeutet „**und**" und legt die Reihenfolge der Symbole fest
  > Ein senkrechter Strich [|] bedeutet „**oder**"

- Wiederholungen
  > Mit {} geklammerte Ausdrücke können beliebig oft wiederholt werden
  > Mit [] geklammerte Ausdrücke können 0 oder 1 mal auftreten

- Klammerung mit () dient zur logischen Gliederung

Formale Sprachen
Produktionen

Eine Produktion ist eine syntaktisch mögliche Kombination von terminalen Symbolen

- In einer Produktion kommen keine nicht-terminalen Symbole vor
  - Die Regeln der Grammatik müssen „bis zum Schluss" angewendet werden

- Beispiel einer Produktion
  - `der mann geht in das haus.`
  - `der mann geht in der haus.`
  - `die frau geht in das mann.`

Formale Sprachen
Semantik

Die Semantik beschreibt die Bedeutung der Zeichen

- Syntaktisch mögliche Produktionen sind unter Umständen „sinnlos"
  - Der Grammatik müssen weitere Regeln hinzugefügt werden, damit alle syntaktisch möglichen Produktionen auch einen Sinn ergeben

- Beispiel
  - In der bekannten Grammatik ist es sinnvoll, dass eine Produktion von `Nomen` nur aus Artikel und Substantiv des gleichen „Geschlechtes" bestehen darf

```
Nomen          :=    ( wArtikel wSubstantiv ) |
                     ( mArtikel mSubstantiv )
wArtikel       :=    "die"
mArtikel       :=    "der"
wSubstantiv    :=    "frau"
mSubstantiv    :=    "mann"
```

Formale Sprachen

Beispiel einer Grammatik in BNF

```
Programm :=    'PROGRAM' Bezeichner
               'BEGIN'
               { Zuweisung [";"] }
               'END' "." ;

Bezeichner := Buchstabe { ( Buchstabe | Ziffer ) } ;
Zahl := [ "-" ] Ziffer { Ziffer } ;
String := '"' { AlleZeichen } '"' ;
Zuweisung := Bezeichner "=" ( Zahl |
                              Bezeichner |
                              String ) ;
Buchstabe := "A" | "B" | "C" | "D" | "E" | "F" | "G"
           | "H" | "I" | "J" | "K" | "L" | "M" | "N"
           | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
           | "V" | "W" | "X" | "Y" | "Z" ;
Ziffer := "0" | "1" | "2" | "3" | "4" | "5" | "6"
        | "7" | "8" | "9" ;
AlleZeichen := ? alle sichtbaren Zeichen ? ;
```

Formale Sprachen
Beispiel einer Produktion

```
PROGRAM DEMO1
BEGIN
    A0=3;
    B=45;
    H=-100023;
    C=A;
    D123=B34A;
    ESEL=GIRAFFE;
    TEXTZEILE="Hallo, Welt!";
END.
```

Formale Sprachen
Formale Definition einer Grammatik

Die Grammatik einer Sprache L(G) ist ein 4-Tupel mit:

$$G = \{ N, T, \Sigma, P \}$$

- N = Menge der nicht-terminalen Symbole
- T = Menge der terminalen Symbole
- Σ = Startsymbol
- P = Menge der Produktionen
  (die sich aus den Regeln der Syntax und der Semantik ergeben)

Formale Definition einer Grammatik

Anmerkungen

- Die Menge der terminalen und nicht-terminalen Symbole ist disjunkt
- Es gibt nur endlich viele Produktionsregeln
- Das Startsymbol verhindert die Bildung nicht gewollter „Teil-"Produktionen
  - > Σ = {Satz} garantiert z.B. vollständige Sätze in der Produktion
- Man sagt eine Grammatik „erzeugt" die zugehörige Sprache
  - > Das bedeutet, dass alle Produktionen der Sprache der Grammatik gehorchen

Formale Sprachen
Übungen

Welche Sprache L erzeugt folgende Grammatik ?

$$G = \{ N, T, \Sigma, P \}$$

- $N = \{ X \}$
- $T = \{ a , b \}$
- $\Sigma = \{ X \}$
- $P = \{ X \rightarrow ab , X \rightarrow aXb \}$

L =

Formale Sprachen
Übungen

Welche Sprache L erzeugt folgende Grammatik ?

$$G = \{ N, T, \Sigma, P \}$$

- $N = \{ X \}$
- $T = \{ a , b \}$
- $\Sigma = \{ X \}$
- $P = \{ X \rightarrow ab , \; X \rightarrow aXb \}$

$$L = \{ a^n b^n \mid n \in \mathbb{N} \}$$

4.2 – Programmiersprachen

Compiler, Linker & the Rest

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Bezug zu formalen Sprachen
- Ablauf der Erstellung von Programmen

Compiler
Wozu das Ganze?

Als Compiler wird ein Programm bezeichnet, das Quellcode in ein lauffähiges Programm überführt

- Ein Compiler übersetzt nach Bytecode oder Maschinensprache
- Es gibt verschiedene Arten von Compilern
  - > Ahead-Of-Time Compiler
    - > „normaler Compiler"
  - > Just-in-Time Compiler
    - > wird erst aktiv, wenn ein bestimmter Teil des Programm gebraucht wird
  - > [Interpreter]
    - > Führt Quellcode direkt und Schritt für Schritt aus
    - > Kein Compiler im eigentlichen Sinne

Compiler
Ablauf der Kompilierung

Die Kompilierung besteht aus zwei Phasen

- Analysephase – Der Code wird analysiert und auf Fehler überprüft
  - Lexikalische Analyse
  - Syntaktische Analyse
  - Semantische Analyse

- Synthesephase – Der Programmcode wird erzeugt
  - Zwischencode-Erzeugung
  - Programmoptimierung
  - Codegenerierung

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Compiler
Analysephase – Lexikalische Analyse

Die lexikalische Analyse wird vom lexikalischen Scanner (kurz: *Lexer*) ausgeführt

- Der Quellcode wird in einzelne Teile, sog. Tokens, zerlegt
- Die Tokens werden klassifiziert (Schlüsselwort, Bezeichner, …)
- Kommentare und überflüssige Whitespaces werden entfernt

Compiler
Analysephase – Syntaktische Analyse

Die syntaktische Analyse wird vom Parser ausgeführt

- Die einzelnen Tokens, die der Lexer erzeugt hat, werden in einen Syntaxbaum umgesetzt

- Der Sourcecode wird auf syntaktische Fehler überprüft

Compiler
Analysephase – Semantische Analyse

Bei der semantischen Analyse werden die einzelnen Anweisungen überprüft

- Sind alle Variablen vor dem ersten Zugriff deklariert worden?
- Stimmen Quell- und Zieltyp einer Zuweisung überein?
- Sind alle aufgerufenen Funktionen deklariert?
- …

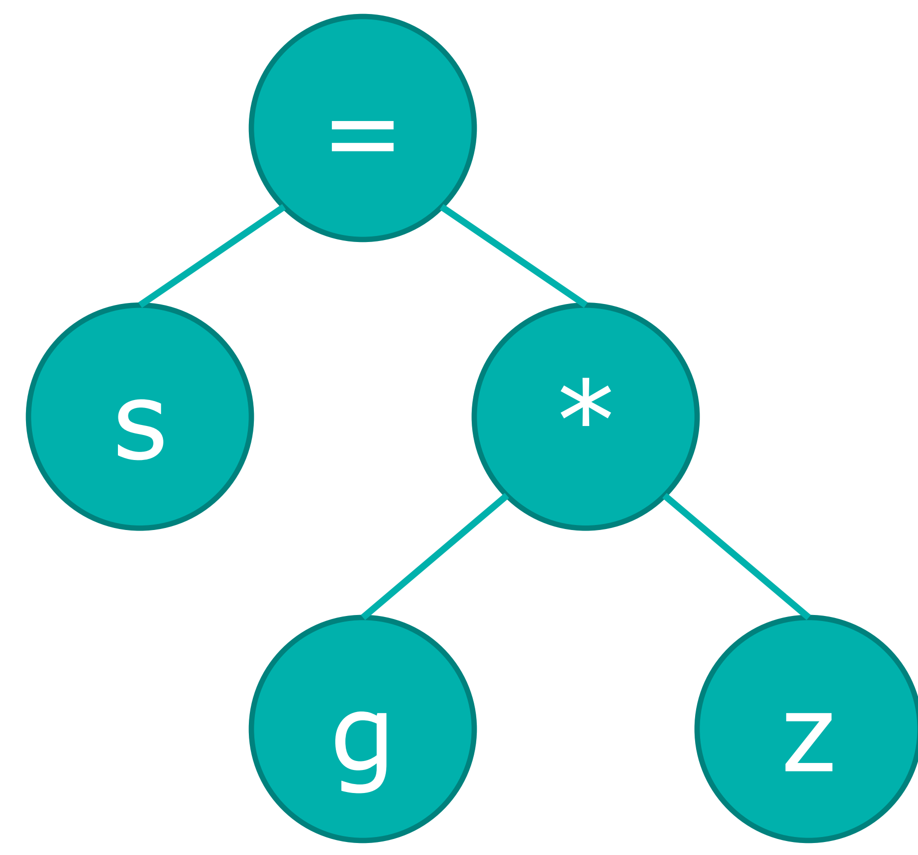
Compiler
Analysephase – Beispiel

strecke = geschwindigkeit * zeit;

- Bezeichner „strecke“
- Operator (Zuweisung) „=“
- Bezeichner „geschwindigkeit“
- Operator (Multiplikation) „*“
- Bezeichner „zeit“
- Trennzeichen (Befehlsende) „;“

Compiler
Analysephase – Beispiel

```
strecke = geschwindigkeit * zeit;
```



Compiler
Synthesephase

Die Synthesephase besteht aus …

- … Zwischencode-Erzeugung
  - > Generiert einen (plattformunabhängigen) Zwischencode
    - > Portabilität, Optimierung einfacher, …
- … Programmoptimierung
  - > Anpassung an Zielplattform und –hardware
  - > Optimierung der Befehlsstruktur
- … Codegenerierung
  - > Aus dem (optimierten) Zwischencode wird der Programmcode erzeugt
    - > Entweder lauffähiges Programm …
    - > … oder Objektdatei (→ Linker)

Compiler
Synthesephase – Programmoptimierung

Viele verschiedene Optimierungen möglich

- Einsparung von Maschinenbefehlen
- Umsetzung statischer Formeln
- Elimination von „Dead Code"
- Entfernung unbenutzter Variablen
- Integration von Funktionsaufrufen
- …

Compiler
Einsparung von Maschinenbefehlen

Dreieckstausch von Variablen

Programmiersprache	Maschinencode	
	Ohne Optimierung	Mit Optimierung
temp = a	a → Register 1 Register 1 → temp	a → Register 1
a = b	b → Register 1 Register 1 → a	b → Register 2 Register 2 → a
b = temp	temp → Register 1 Register 1 → b	Register 1 → b

Compiler
Umsetzung statischer Formeln

Als statische Formeln werden Formeln bezeichnet, welche (mehrere) Konstanten enthalten

- Alle zur Kompilierungszeit bekannten Konstanten werden zusammengefasst, um Operationen zur Laufzeit einzusparen
- „constant folding"

Beispiel

- Vorher

```
> pi = 3.141592653; u = 2 * pi * r;
```

- Nachher

```
> pi = 3.141592653; u = 6.28318531 * r;
```

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Compiler
Elimination von „Dead Code“ …

… und unbenutzten Variablen

- Nicht verwendeter Code kann beim Kompilieren entfernt werden

- Beispiel
  - Vorher
    - ```
      int addiere(int a, int b) {
              int c;
              return (a + b);
              c = a + b;
              return c;
      }
      ```

  - Nachher
    - ```
      int addiere(int a, int b) {
              return (a + b);
      }
      ```

Compiler
Integration von Funktionsaufrufen

Bei kleinen Funktionen ist der Aufwand der Adressumsetzung und des Sprungbefehls oft größer, als der Nutzen des strukturierten Codes

- Beispiel
  - Vorher
    - ```
      int addiere(int a, int b)
      {
          return a + b;
      }

      int c = addiere(3, 4);
      ```
  - Nachher
    - ```
      int c = 3 + 4;
      ```

Compiler
Expertenmeinung

Donald Knuth:

- "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil"

Michael A. Jackson

- *“The First Rule of Program Optimization: Don't do it.*
- *The Second Rule of Program Optimization (for experts only!): Don't do it yet.”*

Linker
Generelle Informationen

Ein Linker erstellt aus verschiedenen Programmodulen (Objektdateien) ein lauffähiges Programm

- Die Dateien werden „zusammengebunden“ (gelinkt)
  - Eigene Objektdateien
  - Programmbibliotheken
- Umsetzung symbolischer Adressen (Funktionen, Variablen) der verschiedenen Module in Speicheradressen
- Unterscheidung zwischen …
  - Statischem Linken
  - Dynamischem Linken

Linker
Statisches Linken

Alle verfügbaren Objektdateien werden zu einer einzigen, ausführbaren Programmdatei gelinkt

- Einfacher Austausch von Programmteilen nicht möglich
  - Bei Änderungen am Programm muss der Linkvorgang jeweils wiederholt werden
- Wird heute nur noch in Mischform mit dynamischem Linken betrieben

Linker
Dynamisches Linken

Funktions- und Variablenadressen werden erst zur Laufzeit des Programms aufgelöst

- Einfacher Austausch von Programmteilen möglich
- DLL (Dynamically Linked Library), Shared Library
- Vorteile
  - Von mehreren Programmen verwendete DLLs müssen nur einmal im System verfügbar sein
  - Die fertigen Programm werden kleiner
- Nachteile
  - Es muss sichergestellt sein, dass die richtige DLL in der richtigen Version vorliegt

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Linker
Programmbibliotheken

Als Programmbibliothek wird eine Sammlung von Funktionen bezeichnet

- Oft thematisch zusammenhängend
  - z.B. Grafikbibliotheken, Mathematikbibliotheken, …
- Programmbibliotheken sind keine eigenständigen Programme, sondern nur Hilfsmodule
- Unterscheidung zwischen …
  - Statischen Bibliotheken (statisches Linken)
  - Dynamischen Bibliotheken (dynamisches Linken)

Linker
Interpreter

Quellcode wird nicht kompiliert, sondern bei der Programmausführung eingelesen und Schritt für Schritt ausgeführt

- z.B. in Webabwendungen verwendet (PHP, JavaScript, …)
- Nachteile gegenüber Compilersprachen
  - > Geringere Ausführungsgeschwindigkeit
- Vorteile gegenüber Compilersprachen
  - > Quellcode ist einfach portabel, sofern für die Zielplattform eine Interpreter-Implementierung vorhanden ist
  - > Bei Änderungen des Quellcodes muss nicht das ganze Programm aufwändig neukompiliert werden

5 - Rechnerorganisation

Von Neumann Architektur

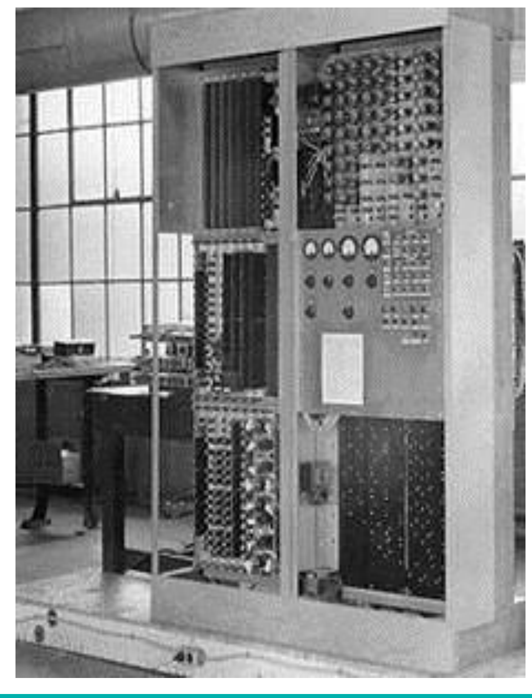
© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Von Neumann`sches Rechnerkonzept
- Speicherhierarchie zur Befehlsverarbeitung
- Typische Prozessorarchitekturen

Der von Neumann-Rechner
EDVAC



John von Neumann:
First Draft of a Report on the EDVAC
(**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) [1]
30. 6. 1945

[1] http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf

Der von Neumann-Rechner
Komponenten

Übersicht

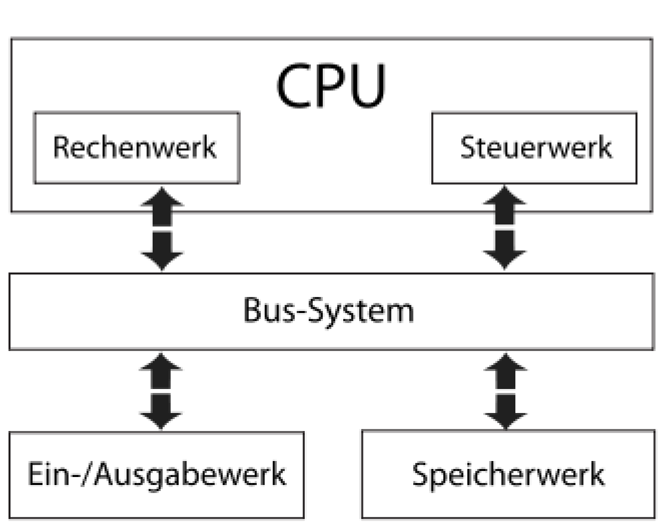
Der von Neumann-Rechner
Komponenten

Übersicht

CPU:

- ALU (Arithmetic Logic Unit) – Rechenwerk
  - > Führt Rechenoperationen und logische Verknüpfungen durch.
- Control Unit – Steuerwerk oder Leitwerk
  - > Interpretiert Anweisungen eines Programms
  - > Verschaltet Daten und notwendige ALU-Komponenten

- Memory – Speicherwerk
  - > Speichert Programme und Daten, RAM (Rnd. Access Mem.), ROM (Read-Only Mem.)
- Bus – Datenleitung
  - > Datenbus und Adressbus
- I/O Unit – Eingabe-/Ausgabewerk
  - > Steuert Ein- und Ausgabe von Daten



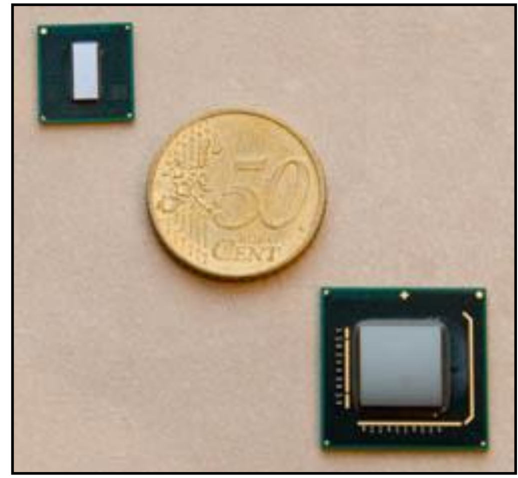
Der von Neumann-Rechner
Komponenten – CPU

Übersicht

Der von Neumann-Rechner
CPU

Die CPU (Central Processing Unit)

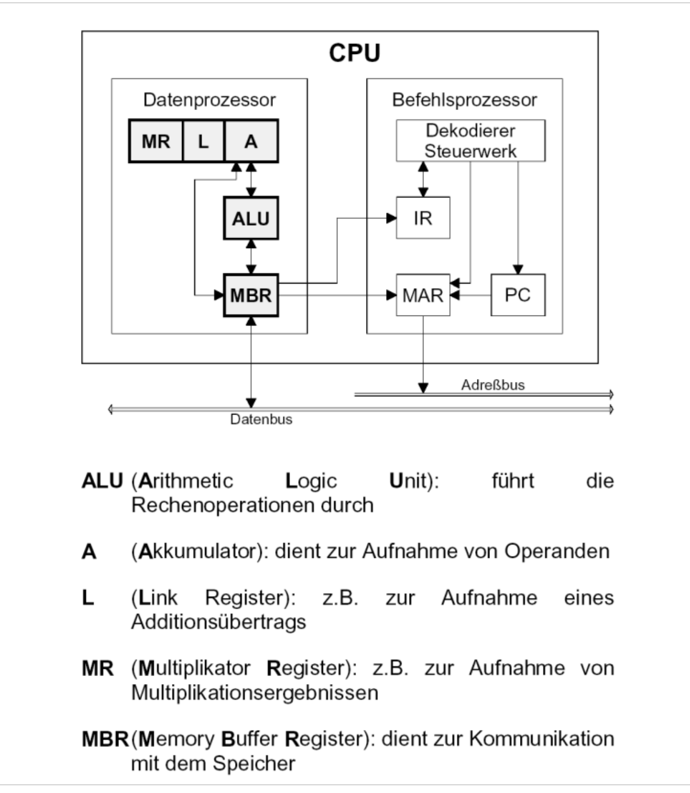
- Auch Prozessor genannt
- Besteht aus Leitwerk und Rechenwerk
- Beispiel: Intel Pentium 4, Intel Atom, AMD Phenom, …



Der von Neumann-Rechner
CPU

Rechenwerk (Datenprozessor)

- Ausführung von arithmetischen und logischen Operationen **(ALU)**
- Zwischenspeicherung mittels Registern:
  - Akkumulator-Register **(A)**
  - Operanden-Register **(MBR)**
  - Übertrags-Register **(L)**



ALU (**A**rithmetic **L**ogic **U**nit): führt die Rechenoperationen durch

A (**A**kkumulator): dient zur Aufnahme von Operanden

L (**L**ink Register): z.B. zur Aufnahme eines Additionsübertrags

MR (**M**ultiplikator **R**egister): z.B. zur Aufnahme von Multiplikationsergebnissen

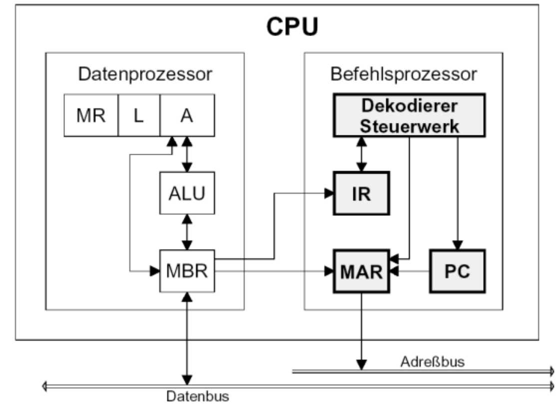
MBR (**M**emory **B**uffer **R**egister): dient zur Kommunikation mit dem Speicher

Der von Neumann-Rechner
CPU

Leitwerk (Befehlsprozessor)

- Überwachung der Programm durchführung, also der im ge-speicherten Programm festgelegten Arbeitsanweisungen

1. Adresse des ersten Befehls in den Befehls- Zähler laden **(PC)**
2. Leitwerk holt den Befehl in das Befehls-Register **(IR)**
3. Befehls- Zähler (PC) um 1 erhöht.
4. Ausführung des Befehls
5. Weiter bei 2.



IR (**I**nstruction **R**egister): enthält den jeweils aktuell bearbeiteten Befehl

MAR (**M**emory **A**ddress **R**egister): enthält die Adresse des als nächsten anzusprechenden Speicherplatzes

PC (**P**rogram **C**ounter): beinhaltet die Adresse des als nächsten auszuführenden Befehls

Dekodierer/Steuerwerk: Dekodierung von Befehlen und Steuerung der Ausführung

Der von Neumann-Rechner
CPU

Leitwerk

- Befehlsarten
  - > Arithmetische Befehle (Grundrechenarten)
  - > Logische Befehle (Vergleichsoperationen, …)
  - > Transferbefehle (Speicherung, …)
  - > Steuerbefehle (Sprungbefehle, …)
  - > …

- Befehlsaufbau
  - > Jeder Befehle bestehet aus zwei Abschnitten:
    - > Operationsteil (Welcher Befehl?)
    - > Argumente (Womit, Wohin, …?)

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Der von Neumann-Rechner
CPU

Befehlsabarbeitung im Leitwerk

- Jeder Befehl durchläuft drei Komponenten des Leitwerks
  - **Befehlsregister** (***fetch cycle***)
    - Der Befehl wird in das Befehlsregister des Steuerwerks geladen
  - **Dekodierer** (***decode cycle***)
    - Der Befehl wird dekodiert und der Befehlszähler neu gesetzt
  - **Steuerwerk** (***execute cycle***)
    - Der Befehl wird ausgeführt

Es gibt zwei grundsätzliche Typen von CPUs

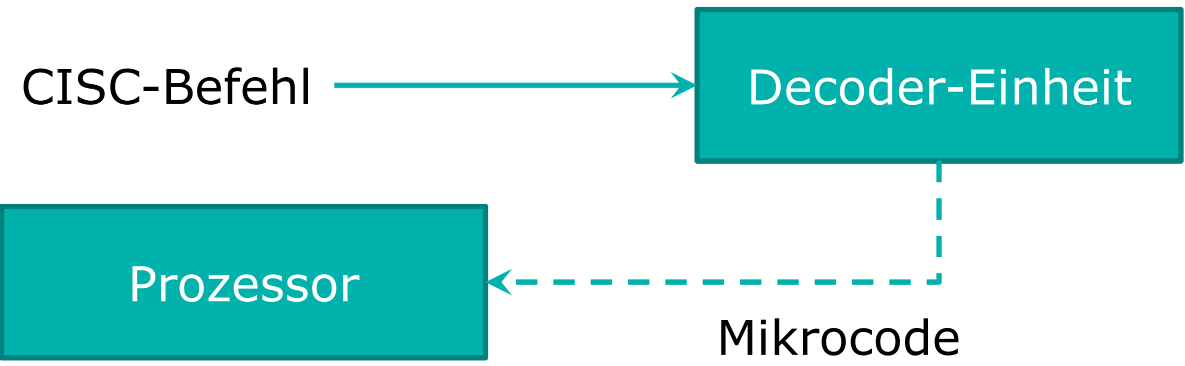
- CISC (Complex Instruction Set Computer)
  - Komplexe Befehle, Bsp. Division
  - Befehlssatz meist in Form von Microcode
  - Wenig Register, Bsp. x86 (9 Register)

- RISC (Reduced Instruction Set Computer)
  - Verzichtet auf komplexe Befehle
  - Einzelnen Befehle sind fest verdrahtet
  - Viele Register, Bsp. PowerPC (32 GPR Register, Playstation, Lockheed Martin F-22)

Der von Neumann-Rechner
CISC CPUs

Eine CISC CPU hat folgende Eigenschaften

- Die CPU stellt eine große Anzahl zum Teil sehr komplexer Befehle zur Verfügung
- Die Abarbeitungszeit verschiedener Befehle ist unterschiedlich lang
- Die Befehle haben unterschiedliche „Größen“ (Speicherplatz)
- CISC-CPUs sind mikroprogrammiert



Der von Neumann-Rechner
CISC CPUs

Argumente…

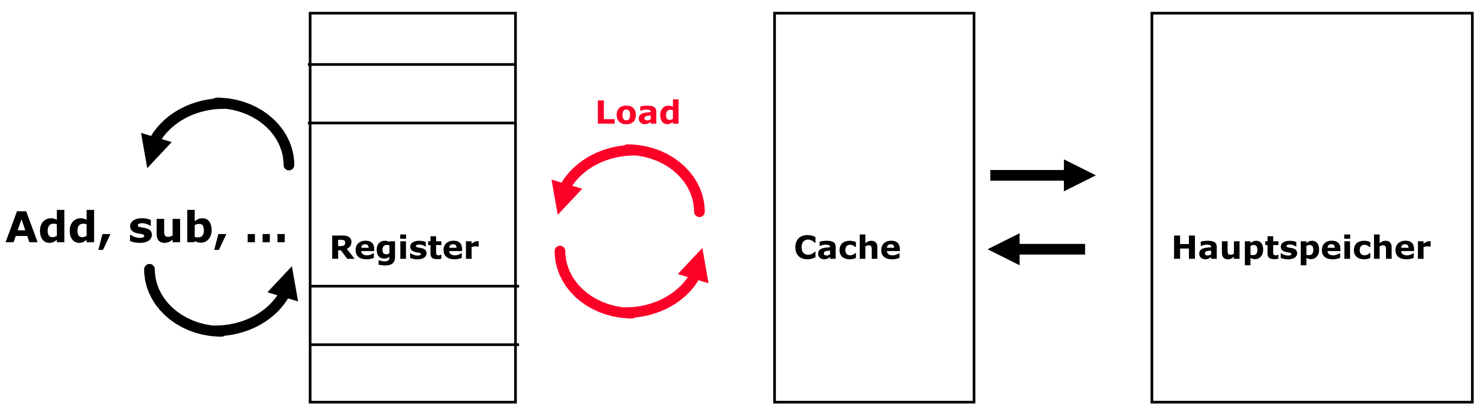
- … für CISC
  - Komplexe Befehle verringern den Hauptspeicherbedarf für Programmcode
  - Komplexe Befehle vereinfachen den Compiler

- … gegen CISC
  - Die komplexen Befehle, die eigentlich die Compiler vereinfachen sollen, werden kaum von den Compilern benutzt
  - Caching verringert die Anzahl der Hauptspeicherzugriffe ebenfalls effizient

Der von Neumann-Rechner
RISC CPUs

Eine RISC CPU hat folgende Eigenschaften

- Die CPU stellt wenige, elementare Funktionen zur Verfügung
- Alle Befehle haben die gleiche „Größe"
- Dekodieraufwand kleiner
- Befehle sind „fest verdrahtet"



Der von Neumann-Rechner
RISC CPUs

Nach D. Tabak ("RISC Architecture", 1995) müssen mindestens 5 der folgenden Kriterien erfüllt sein, damit eine RISC-Architektur vorliegt

- Kriterien für RISC CPUs:
  - Mindestens 80% der Befehle werden in einem Taktzyklus ausgeführt
  - Alle Befehle werden mit einem Maschinenwort codiert
  - weniger als 128 Maschinenbefehle
  - weniger als 4 Adressierungsarten
  - weniger als 4 Befehlsformate
  - Speicherzugriffe nur über Load/Store-Befehle
  - Register-Register Architektur (ALU)
  - mehr als 32 Prozessorregister
  - festverdrahtete Maschinenbefehle (keine Mikroprogrammierung)
  - höhere Programmiersprachen werden durch optimierende Compiler unterstützt

Der von Neumann-Rechner
RISC vs. CISC

	RISC	CISC
Ausführungszeit	1 Datenpfadzyklus (fetch, decode, execute)	≥ 1 Datenpfadzyklus
Instruktionsanzahl	Klein	Groß
Instruktionsformat	Einfach / einheitlich	Variabel
Steuerung über…	Hardware	Mikroprogramm
Hauptspeicherzugriffe	LOAD/STORE-Architektur	Keine Einschränkungen
Pipelining	Möglich	Nicht möglich
Verlagerung der Komplexität	Compiler	Hardware

Der von Neumann-Rechner
Komponenten – Speicher

Übersicht

Datenbus

CPU

Speicher

RAM | ROM

I / O

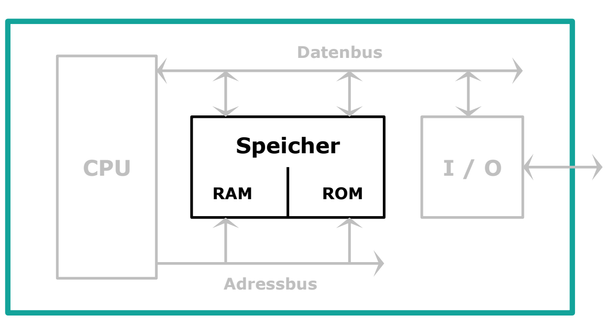
Adressbus

Speicher
Ein anschauliches Beispiel

Der von Neumann-Rechner
Speicher

Es gibt im Wesentlichen zwei Arten von Speicher

- ROM (Read Only Memory)
  - Festwertspeicher
    - Daten fest eingebrannt, nicht mehr veränderbar
  - Es gibt mehrere Arten
    - ROM, PROM, EPROM, EEPROM
- RAM (Random Access Memory)
  - Speicher mit wahlfreiem Zugriff
    - Speichert Befehle **und** Daten
  - Es gibt zwei verschiedene Arten
    - SRAM (Static RAM)
    - DRAM (Dynamic RAM)

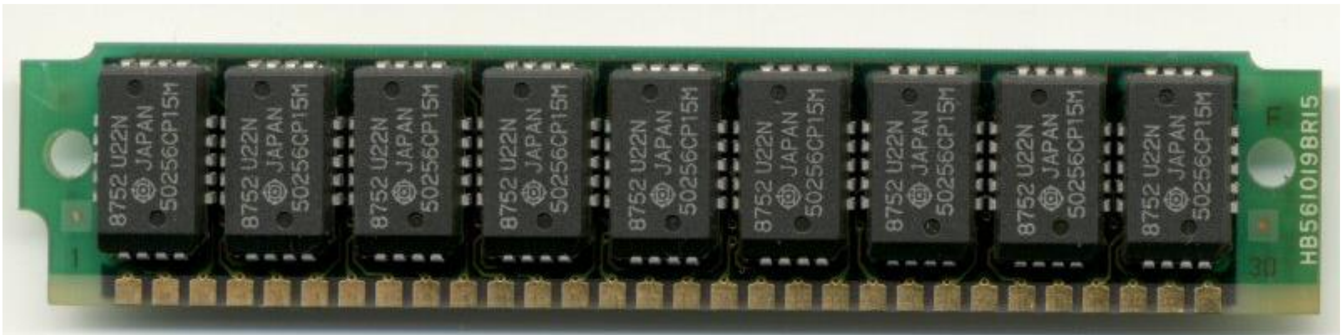


Der von Neumann-Rechner
Speicher – RAM

RAM ist ein flüchtiger Speicher, der bei der Abschaltung der Betriebsspannung verloren geht

- Im RAM liegen Informationen, die ein Programm während der Laufzeit braucht
  - Soll eine Information über das Ende des Programms hinaus gespeichert werden, muss sie auf der Festplatte (o.Ä.) gespeichert werden

- Der RAM besteht aus gleichartigen Speicherplätzen, die jeweils eine Adresse haben



Single Inline Memory Modul (SIMM)

Der von Neumann-Rechner
Speicher – DRAM vs. SRAM

Es gibt zwei verschiedene Arten von RAM in einem Computer

- Der Hauptspeicher eines Computers ist aus DRAM aufgebaut
  - Der Inhalt von DRAM-Modulen ist flüchtig (volatil)
    - Die Information muss ständig aufgefrischt werden
      - Auch nach dem Auslesen der Information

- Der Cache eines Computers ist aus SRAM aufgebaut
  - Der Inhalt von SRAM-Modulen ist nicht flüchtig

SRAM	**ist**	**schneller**	**aber**	**teurer**	**und**	**größer**
DRAM		**langsamer**		**billiger**		**kleiner**

Der von Neumann-Rechner
Speicher – Hierarchie

Kosten / Bit ↑ (von unten nach oben)

Kapazität, Zykluszeit, Zugriffszeit ↓ (von oben nach unten)

Ebene	Kapaz.	Geschw.
Register / On Chip Cache	wenige Bytes bis 8 KB	1 -10 ns 1-10 GB/s
Cache (SRAM)	8 KB bis 8 MB	1 - 20 ns 1 - 5 GB/s
EDO RAM, FPM RAM, SDRAM, Rambus DRAM	16 MB b. 64 GB	10 -100 ns 100 MB/s - 1.6 GB/s
Peripheriegeräte	1 GB bis 1 TB	30 MB/s
Externe Speicher	1 TB bis 1 PB	ms - sec. 1 - 40 MB/s

Der von Neumann-Rechner
Expertenmeinung

„640 KBytes [Arbeitsspeicher] sollten für jeden genug sein…“

(*Bill Gates, 1981****)***

Der von Neumann-Rechner
Komponenten – Busse

Übersicht

Datenbus

CPU

Speicher

RAM | ROM

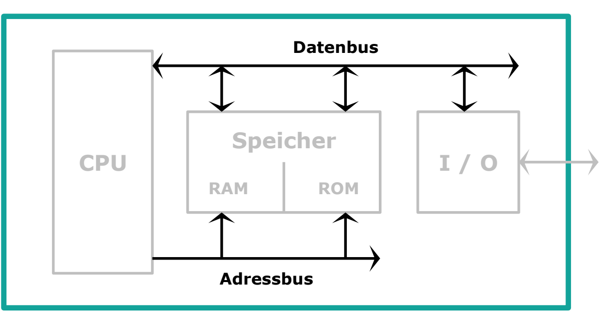
I / O

Adressbus

Der von Neumann-Rechner
Busse

Busse ermöglichen die Kommunikation zwischen Speicher, CPU und I/O-Einheit

- Man unterscheidet zwischen Daten- und Adressbus
  - Der Datenbus ist bidirektional
  - Der Adressbus ist unidirektional



Der von Neumann-Rechner
Komponenten – I / O

Übersicht

Der von Neumann-Rechner
Eigenschaften

Der Von-Neumann-Flaschenhals

- Vorüberlegungen
  - Die Ausführungszeit eines Befehls ist meistens sehr kurz
  - Die Zugriffszeit auf den Speicher ist im Vergleich sehr lang
- Die CPU wartet, während Dinge aus dem Speicher geladen oder in den Speicher geschrieben werden
  - Die Ausführungszeit eines Programms hängt (auch) von der Anzahl der Speicherzugriffe ab
  - Es gibt viele Ideen und Strategien, um Teilbereich zu parallelisieren bzw. zu beschleunigen

Der von Neumann-Rechner
Pipelining

Idee: „Never waste time"

- Sie kommen aus dem Urlaub und es ist viel schmutzige Wäsche zu waschen!

- Zur Verfügung stehen:
  - > Eine Waschmaschine (0.5 Std. Laufzeit)
  - > Ein Trockner (0.5 Std. Laufzeit)
  - > Eine Bügelmaschine (0.5 Std. Arbeit zum Bügeln)
  - > Ein Wäscheschrank (0.5 Std. Arbeit zum Einräumen)

- Jede Person (A, B, C, D) wäscht seine Wäsche selbst.

Der von Neumann-Rechner
Pipelining

Dauer: 8 Stunden

Time 6 PM 7 8 9 10 11 12 1 2 AM

Task order A B C D

mit Pipelining

Time 6 PM 7 8 9 10 11 12 1 2 AM

Task order A B C D

Dauer: 3,5 Stunden

Der von Neumann-Rechner
Pipelining

Pipelining ist nur auf RISC-Architekturen möglich

- Die Befehlsausführung funktioniert immer nach demselben Prinzip
- Es gibt unabhängige Teile einer Befehlsausführung
  - > Diese Können parallelisiert werden
  - > Diese benutzen im Allgemeinen unabhängige Hardwarekomponenten

- Beispiel
  - > 5 Stage Pipeline
    - > Instruction Fetch (**IF**): Befehl lesen
    - > Instruction Decode (**ID**): Befehl decodieren
    - > Execute (**EX**): Ausführen
    - > Memory Access (**MEM**): Speicherzugriff
    - > Writeback (**WB**): Ergebnis schreiben

Instr. No.	Pipeline Stage					
1	IF	ID	EX	MEM	WB	
2		IF	ID	EX	MEM	WB
3			IF	ID	EX	MEM
4				IF	ID	EX
5					IF	ID
Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**

Der von Neumann-Rechner
Pipelining – Hazards

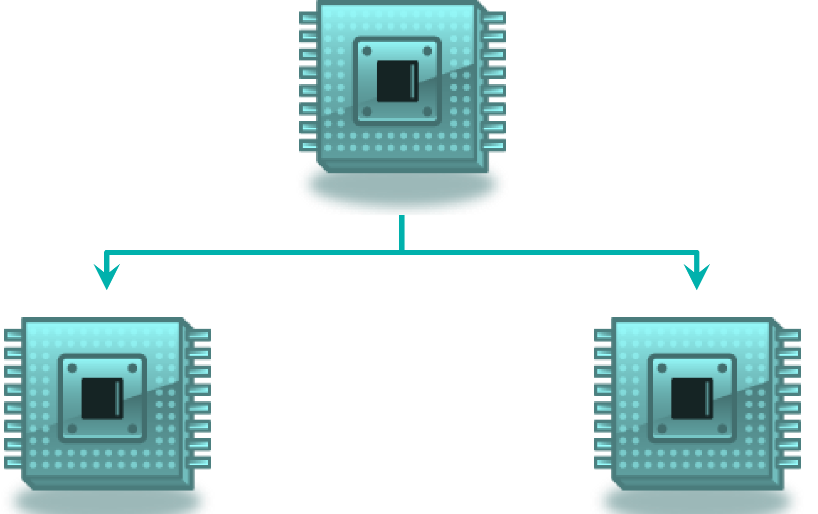
Hazards sind Situationen, in denen die Abarbeitung einer Pipeline unterbrochen werden muss

- Structural-Hazards
  - Instruktionen benötigen dieselbe Hardware und können nicht überlappend ausgeführt werden
- Data-Hazards
  - Datenabhängigkeiten zwischen aufeinander folgenden Instruktionen
- Control-Hazards
  - Änderungen der sequentiellen Abarbeitung von Befehlsfolgen durch bedingte Sprünge

Der von Neumann-Rechner
Pipelining – Hazards

Structural-Hazards

- Problem
  - > Instruktionen benötigen dieselbe Hardware
- Lösung
  - > Vermeidung typischerweise durch Replizieren der Hardware



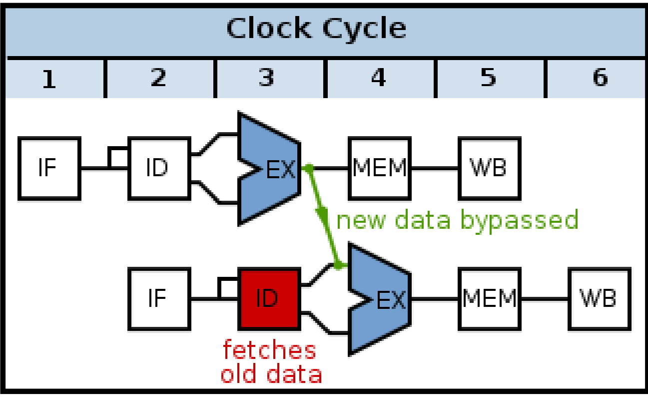
Der von Neumann-Rechner
Pipelining – Hazards

Data-Hazards

- Problem
  - > Datenabhängigkeiten

Pipeline Stage	Clock Cycle					
	1	2	3	4	5	6
Fetch	SUB	AND				
Decode		SUB	AND			
Execute			SUB	AND		
Access				SUB	AND	
Write-Back					SUB	AND

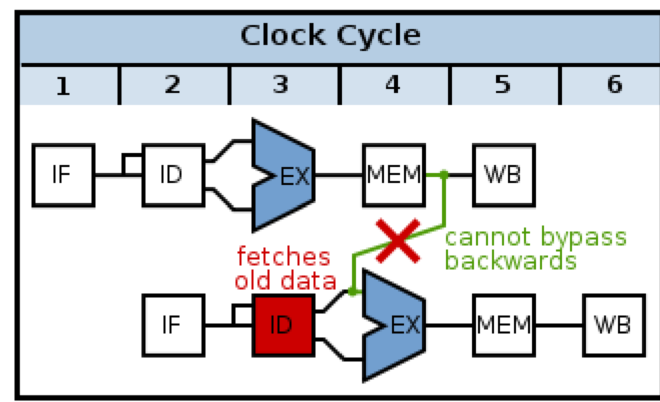
- Lösung
  - > Data Bypassing
    - > Eingangsdaten für Befehle stammen aus Registern oder „Zwischenergebnissen"



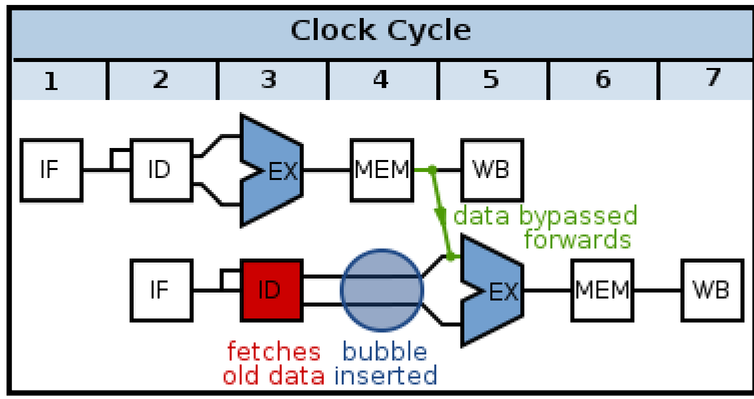
Der von Neumann-Rechner
Pipelining – Hazards

Data-Hazards – Data Bypassing

- Problem
  - > Es kann trotzdem sein, dass Informationen nicht rechtzeitig zur Verfügung stehen



- Lösung
  - > NOP
    - > NOP-Befehl (No Operation) einfügen



Der von Neumann-Rechner
Pipelining – Hazards

Control-Hazards

- Bedingte Sprungbefehle ändern den als nächstes auszuführenden Befehl
  - Nächster Befehl steht erst am Ende des vorherigen Befehls fest
  - Die Pipeline hat bereits begonnen unnötige Befehle abzuarbeiten
- Lösung:
  - Stall / Freeze
  - Predict Taken
  - Predict Not Taken
  - Branch Delay Slot

6 - Speicher

Arbeitsspeicher und Cache

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Technische Realisierungen von Speicher
  - Arbeitsspeicher
  - Massenspeicher
- Funktionweise und Aufbau eines Caches
- Funktionsweise eines RAID-Systems

Speicher
Fotografischer Speicher

Speicherung der Information als Bild

- Speichern der Information durch chemischen Vorgang
- Lesen der Informationen durch simples Betrachten des Bildes
  - Formatunabhängig
  - Kein spezielles Lesegerät notwendig
- Informationen können bewegt oder unbewegt abgelegt werden
- Beispiele
  - Filme
  - Fotos

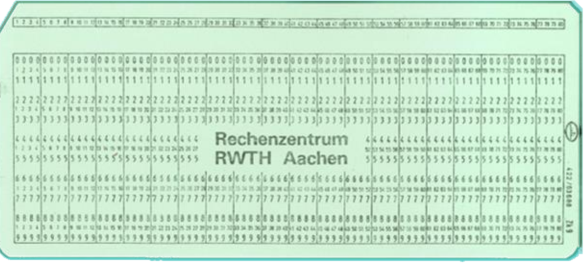
Speicher
Mechanischer Speicher

Daten werden durch mechanische Einwirkung auf das Speichermedium geschrieben

- Daten können analog oder digital gespeichert werden
- Daten können mechanisch oder optisch ausgelesen werden

Beispiele

- Mechanischer Lesevorgang
  - > LP (analog)
  - > Lochkarte (digital)
- Optischer Lesevorgang
  - > Laserdisc (analog)
  - > CD, DVD, Blu-ray, … (digital)



Speicher
Halbleiterspeicher

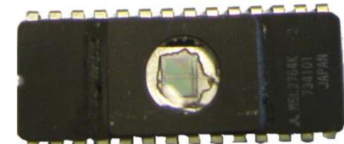
Die Daten werden auf Siliziumbasis mittels integrierter Schaltkreise gespeichert

Verschiedene Arten der Speicherung werden unterschieden

- Flüchtige Speicherung
  - > DRAM, SRAM

- Permanente Speicherung
  - > ROM, PROM

- Semi-permanente Speicherung
  - > EPROM, EEPROM, Flash-Speicher, …



Speicher
Magnetischer Speicher

Die Informationen werden durch gezielte, partielle Magnetisierung des Speichermediums abgelegt

- Zugriff auf die Informationen mittels eines Schreib-/Lesekopfes

- Magnetisches Material als …
  - … Plattenstapel (z.B. bei Festplatte)
    - Beweglicher Lesekopf
  - … Band (z.B. bei MCs)
    - Feststehender Lesekopf
- Speicherung der Informationen analog oder digital

Beispiele

- Festplatte, Diskette, … (digital, beweglicher Lesekopf)
- Magnetstreifen, Magnetkarte, … (digital, feststehender Lesekopf)
- MC, VHS, … (analog, feststehender Lesekopf)

Speicher
Kenngrößen

Typ und Geschwindigkeit

GeIL DIMM 1 GB DDR-400 Kit (GE1GB3200BDC)

Art.-Nr.: IAID81

€ 53,90*

In den Warenkorb legen

Durchschnittliche Produktbewertung (bei 177 Bewertungen):

» Produkt bewerten » alle Bewertungen ansehen

Typ

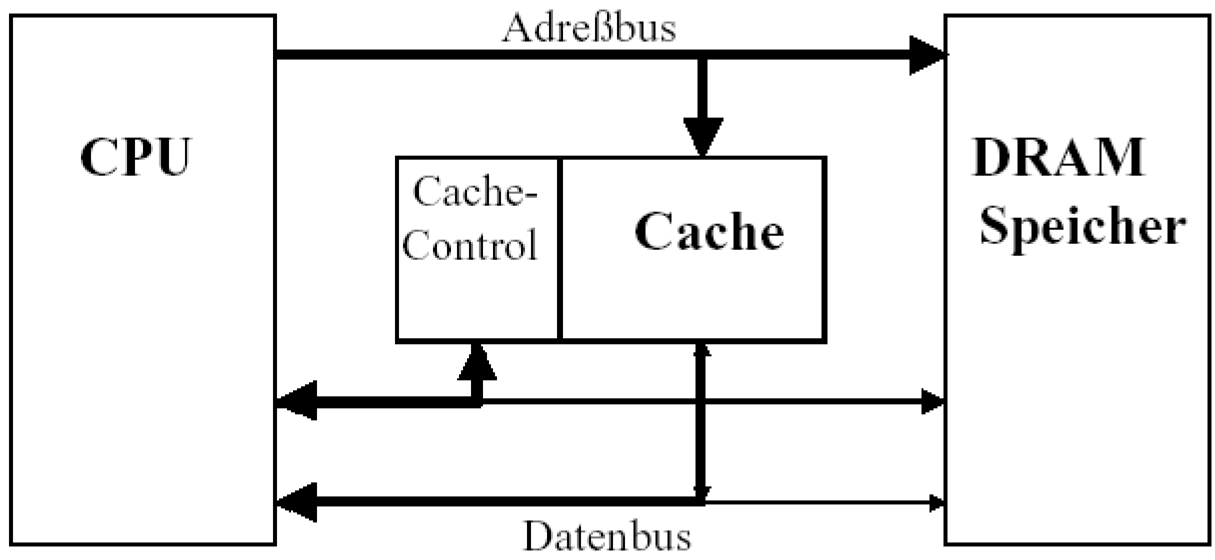
Größe

Problem

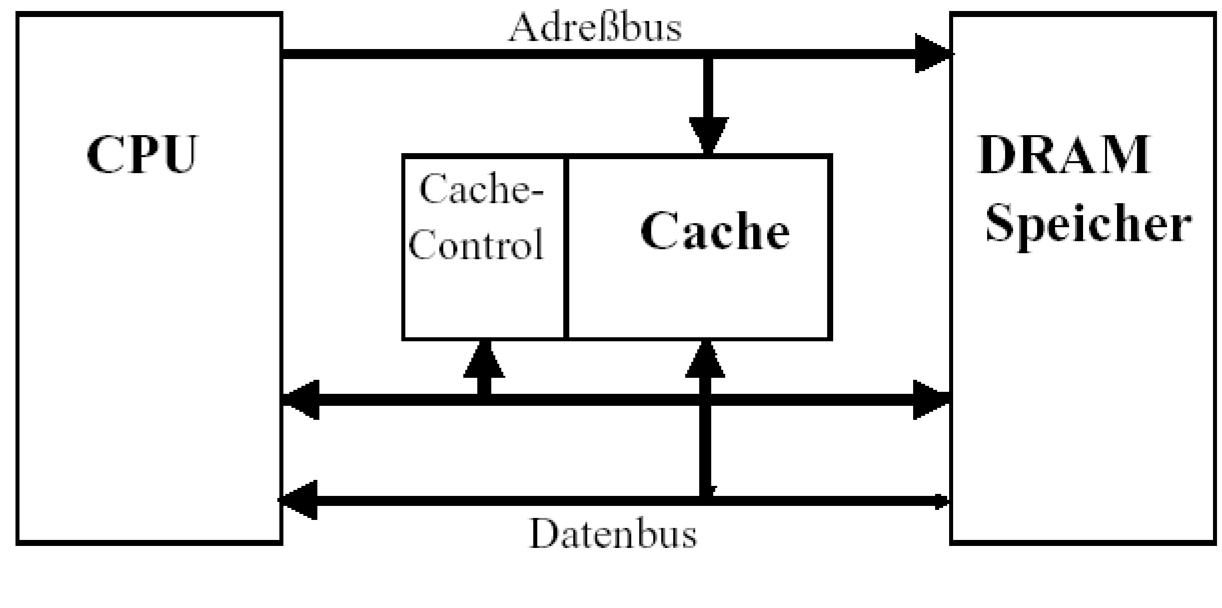
- Schnelle SRAM-Bausteine sind teuer
- Schnelle SRAM-Bausteine nehmen viel Platz in Anspruch

→ In Größenordnung von mehreren GB Speicher schwer zu realisieren

Caches
Idee: Cache-Hit



Caches
Idee: Cache-Miss



... einige Wartezykeln

Caches
Allgemeines

Grundlegendes zu Caches

- Cache ist aus SRAM-Bausteinen aufgebaut
  - Wenig Speicher, im Vergleich zum Hauptspeicher
  - Schneller Speicher, im Vergleich zum Hauptspeicher
- Ein Cache enthält nur kleine Datenmengen (*Cacheblocks*) aus dem Hauptspeicher
  - Problem: Welche Daten sollen aus dem Hauptspeicher geladen werden?
- Ist im Cache keinen Platz mehr für einen neuen Block, muss zuerst ein Block verdrängt werden – **Wie?**

Caches
Lokalitätsprinzip

Es gibt zwei Arten von sogenannter Lokalität

- Zeitliche Lokalität
  - Es ist, bei entsprechender Programmierung, sehr wahrscheinlich, dass auf eine Speicherzelle nicht nur einmal, sondern zeitlich „nah" mehrmals zugegriffen wird

- Örtliche Lokalität
  - Es ist, bei entsprechender Programmierung, sehr wahrscheinlich, dass nach dem Zugriff auf eine bestimmte Speicherzelle auch ein Zugriff in deren unmittelbarer „Nachbarschaft" stattfindet

Caches
Hierarchie auf einem modernen System

CPU-Kern

L1-Cache | L1-Cache

L2-Cache

L3-Cache

Hauptspeicher

Daten- und Befehlszugriffe getrennt

SRAM (8–64 kByte, 1–3 Takte Zugriffszeit)

SRAM (256–16386 kByte, 3–6 Takte)

SRAM (4–64 MByte, 12–15 Takte)

DRAM (64 MByte–8 GByte, 25–40 Takte)

Caches
Hierarchie auf einem modernen System

Die wichtigsten Cache-Level haben folgende Eigenschaften

- L1-Cache
  - Im Prozessor integriert
  - Trennung in Befehls- und Datencache
  - Cache-Blöcke häufig nicht länger als 32 Bytes
- L2-Cache
  - Heute auch schon im Prozessor integriert
  - Gemeinsame Speicherung von Befehlen und Daten
  - Cache-Blöcke bis zu 128 Bytes

Caches
Aufbau

Ein Cache funktioniert wie folgt

- Cache ist ein Assoziativspeicher für Blöcke des Hauptspeichers
  - Tag (*Key/Schlüssel*) entspricht der Speicheradresse
- Ungültige Einträge werden nicht aus dem Cache gelöscht
  - Markierung ungültiger Blocks durch Löschen des Valid-Flags (*V-Flag*)
- Aufbau eines Cacheeintrags
  - Die Daten selbst
  - Tag (Teil der Adresse)
  - Statusbits, u.a.
    - Valid-Flag (gültige Daten)
    - Modified bzw. Dirty (wurde geändert, wichtig für Write-Back)

Caches
Platzierung

Mikroprozessor

Steuerbus

Daten

Cache Controller

Adressbus

Cache

Hauptspeicher

Caches

Aufbau

Steuerung (CPU)

Adresse

Daten (CPU)

Tag-RAM

Daten Cache

Komparator

Daten (Hauptspeicher)

Caches
Arten von Caches

Es gibt drei Arten von Caches

- Direct Mapped Cache
  - > Direkte Abbildung auf den Hauptspeicher
  - > Jeder Block kann nur an genau einem Platz im Cache eingefügt werden, der sich aus der Hauptspeicheradresse ergibt
- Fully associative Cache
  - > Ein Block kann an jeder Stelle des Caches eingefügt werden
  - > Identifizierung über Tag
- Set associative Cache
  - > Mehrere fully associative Caches arbeiten parallel
  - > Identifizierung des „richtigen“ Caches über Tag

Caches
Direct Mapped Cache

Die Stelle, an der ein Block im Cache eingefügt wird, berechnet sich direkt aus der Adresse im Hauptspeicher durch eine Modulo-Operation

- Beispiel
  - Cache besteht aus 8 Blöcken
  - Speicherblock 13 soll in den Cache aufgenommen werden
  - Cacheblock = $13 \, mod \, 8 = 5$

Caches
Fully Associative Cache

Ein Block kann in jeder Stelle in den Cache eingefügt werden

- Beim Einfügen in den Cache wird die Adresse des Blocks in das Tag-RAM geschrieben
- Bei der Überprüfung, ob ein Block im Cache liegt, wird das Tag-RAM durchsucht, ob es die entsprechende Speicheradresse enthält

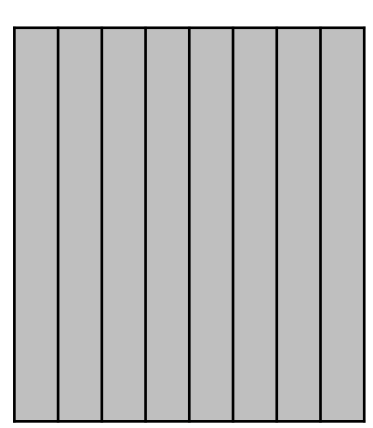
Caches
Set-Associative Cache

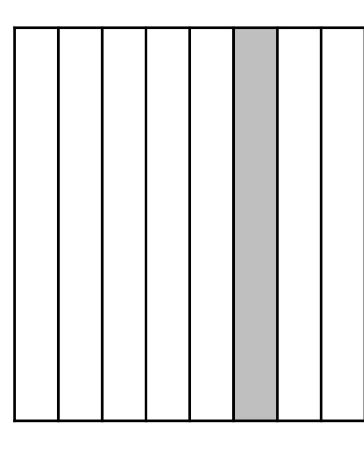
Mehrere fully associative Caches arbeiten parallel

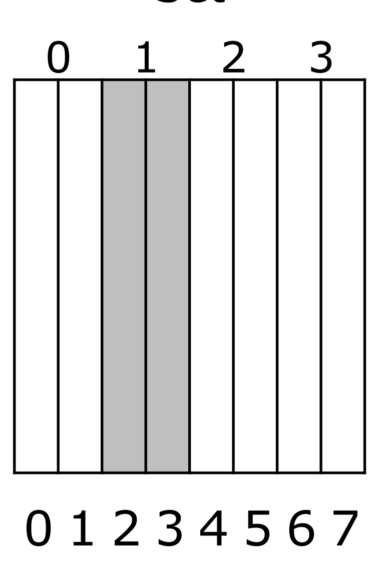
- Der fully associative Cache, in den ein Block eingefügt wird, wird anhand der Speicheradressen und Modulo-Operationen bestimmt

- Beispiel
  - Es gibt 8 fully associative Caches zu je 10 Blöcken
  - Speicherblock 13 soll in den Cache aufgenommen werden
  - Bestimmung des fully associative Caches: $13 \, mod \, 8 = 5$
    - Der fünfte Cache wird verwendet
  - Einfügen des Blocks in diesen Cache wie bei fully associative Caches üblich

Caches
Blockplatzierung







Es soll der Block mit der Speicheradresse 13 eingefügt werden.

- $13 \, mod \, 8 = 5$
- $13 \, mod \, 4 = 1$

Caches
Lesezugriffe

Findet ein Lesezugriff auf Speicherzelle A statt …

- … wird zunächst geprüft, ob Speicherzelle A bereits im Cache liegt.
  1. A liegt im Cache (***cache hit***)
     - Der Datensatz kann direkt aus dem Cache gelesen werden.
  2. A liegt nicht im Cache (***cache miss***)
     - Der Datensatz muss aus dem Hauptspeicher in den Cache geladen werden.
     - Aus dem Cache wird das Datum in die CPU geladen.

Caches
Schreibzugriff

Findet ein Schreibzugriff auf Speicherzelle A statt …

- … wird zunächst geprüft, ob Speicherzelle A bereits im Cache liegt.
  1. A liegt im Cache (***cache hit***)
     - Der Datensatz wird im Cache aktualisiert
     - (Der Datensatz wird im Hauptspeicher aktualisiert)

  2. A liegt nicht im Cache (***cache miss***)
     - Der Datensatz wird direkt in den Hauptspeicher geschrieben.
     - Der Inhalt des Caches wird nicht verändert.

Caches
Schreibzugriff

Es gibt zwei Verfahren zum Aktualisieren von Datensätzen im Cache

- Write-Through
  - > Der Datensatz wird im Cache und direkt anschließend auch im Hauptspeicher aktualisiert
  - > Keine Probleme mit der Datenkonsistenz im Hauptspeicher
- Write-Back
  - > Der Datensatz wird im Cache aktualisiert und erst in den Hauptspeicher zurückgeschrieben, wenn der entsprechende Cacheblock aus dem Cache verdrängt wird
  - > Niedrige Belastung der Systembusse und keine Wartezyklen

Caches
Cache Misses

Es gibt drei Arten von Cache Misses

- *Compulsory (zwangsweise, zwangsläufig)*
  > erstmaliger Zugriff auf eine Adresse, dazugehörigen Daten noch nicht im Cache ⇨ Mgl. Lsg.: Prefetch Einheiten laden selbständig Cache
- *Capacity*
  > Cache ist zu klein, Daten waren im Cache vorrätig, wurden aber wieder aus dem Cache entfernt ⇨ Mgl. Lsg.: größerer Cache
- *Conflict*
  > in einem Satz ist nicht mehr genug Platz und Block wird entfernt, obwohl Platz in anderen Sätzen wäre; erneuter Zugriff auf entfernten Block ist ein „Conflict Miss"⇨ Mgl. Lsg.: Erhöhung der Cacheblocks pro Satz – also eine Erhöhung der Assoziativität.

Caches
Verdrängungsstrategien

Wenn der Cache voll ist muss Platz geschaffen werden

- FIFO (First In, First Out)
  - Der Block, welcher chronologisch gesehen zuerst in den Cache geladen wurde, wird aus dem Cache verdrängt
- LRU (Least Recently Used)
  - Der Block, auf den am längste nicht mehr zugegriffen wurde, wird aus dem Cache verdrängt
- Optimale Ersetzungsstrategie
  - Der Block, der chronologisch gesehen am spätesten wieder gebraucht wird, wird aus dem Cache verdrängt
  - Nachteil: Diese Information steht meistens nicht zur Verfügung
- Varianten der genannten Strategien

Massenspeicher
RAID

RAID steht für “Redundant Array of Independent Disks“

- Ehemals „Redundant Array of Inexpensive Disks“
- Dient zur Organisation mehrerer physikalischer Festplatten in einem Verbund
- Kann hardware- oder softwareseitig verwaltet werden
- Es werden gezielt redundante Daten erzeugt, um beim Ausfall einer Festplatte die Integrität und Funktionalität des Gesamtsystems trotzdem gewährleisten zu können
  - RAID0 Sonderfall

Massenspeicher
Aufbau eines RAID

Es werden mindestens 2 Festplatten benötigt

- Alle in das RAID eingebundenen Festplatten werden gemeinsam betrieben und zielen auf die Verbesserung einer Eigenschaft ab
  - > Erhöhung der Ausfallsicherheit
  - > Steigerung der Datentransferrate
  - > Erweiterung der Speicherkapazität
  - > Möglichkeit des Austauschs von Festplatten im laufenden Betrieb
  - > Kostenreduktion durch Einsatz mehrerer kostengünstiger Festplatten

- Die genau Art des Betriebs wird durch das sog. *RAID-Level* angegeben

Massenspeicher
Hardware-RAID vs. Software-RAID

Es gibt zwei verschiedenen Arten ein RAID zu verwalten

- In einem Hardware-RAID werden alle beteiligten Festplatten durch den sogenannten RAID-Controller angesprochen und verwaltet
- In einem Software-RAID wird die Verwaltung der Festplatten im RAID komplett von einer Software gesteuert
- Pro und Contra beim Software-RAID
  - Pro
    - Es wird kein zusätzliches Hardwarebauteil benötigt
    - Der RAID-Controller selbst kann keinen Schaden nehmen (wie beim Hardware-RAID möglich)
  - Contra
    - CPU und Systembusse werden stärker belastet
    - Nur der RAM kann als Cache benutzt werden

Massenspeicher
Probleme mit RAID 1 / 2

Es können mehrere Problem beim Betrieb eines RAID auftreten

- Größenänderungen
  - > Es ist nicht ohne Weiteres möglich die Kapazität eines RAID im Nachhinein durch einfaches Zuschalten einer weiteren Festplatte zu erhöhen
- Austausch von Datenträgern
  - > Muss eine Festplatte im RAID ausgetauscht werden, so muss sie durch eine Platte gleicher oder größerer Kapazität ausgetauscht werden
- Defekte Controller
  - > Defekte RAID-Controller können Datenverluste erzeugen
  - > Defekte RAID-Controller können i.A. nur durch identische Controller ersetzt werden

Massenspeicher
Probleme mit RAID 2 / 2

Es können mehrere Problem beim Betrieb eines RAID auftreten

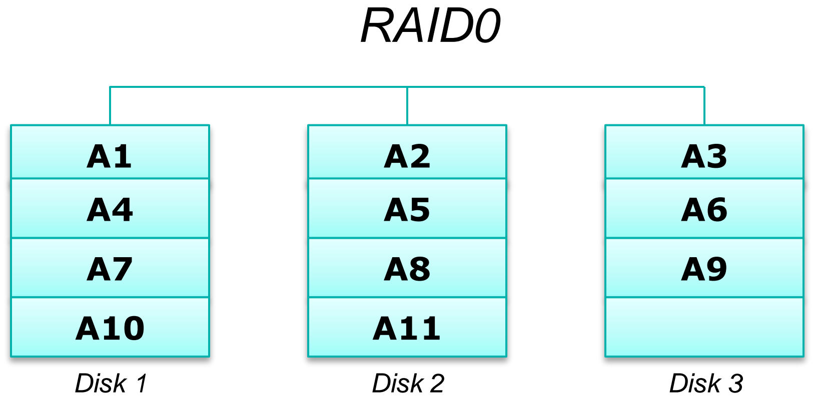
- Fehlerhaft produzierte Datenträgerserien
  - > Kommen mehrere Festplatten einer fehlerhaften Charge im selben RAID zum Einsatz können sich deren Fehler multiplizieren

- Statistische Fehlerrate bei großen Festplatten
  - > Die vom Hersteller angegebenen Fehlerraten können sich beim Betrieb der Platte in einem RAID akkumulieren

Massenspeicher
RAID 0

RAID 0 ist kein RAID im herkömmlichen Sinne, da keine redundanten Daten erzeugt werden

- Bietet höhere Transferraten durch *Striping*
  - Alle beteiligten Festplatten werden in gleich große Blöcke unterteilt
    - Häufig 64 kB pro Block
  - Diese Blöcke werden im Reißverschlussverfahren zu einer großen Festplatte angeordnet
  - Zugriff auf Blöcke kann in gewissem Maße parallel geschehen



Massenspeicher
RAID 0

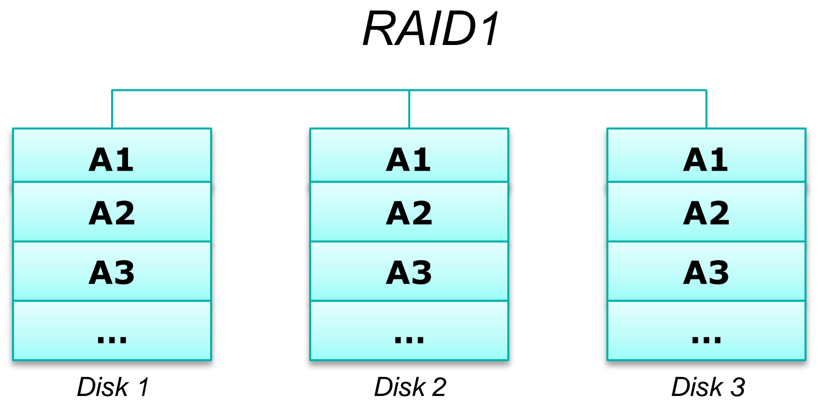
Es gibt mehrere Nachteile bei RAID 0

- Ist eine Festplatte im RAID 0 defekt, so können die Nutzdaten nicht mehr vollständig rekonstruiert werden
  - Kleine Dateien können möglicherweise gerettet werden, wenn keine Datenblöcke auf der kaputten Festplatte lagen
- Der laufende Betrieb muss bei Ausfall einer Festplatte zwangsläufig unterbrochen werden
- Die Ausfallwahrscheinlichkeit eines RAID 0 mit n Festplatten beträgt $1 - (1 - p)^n$
  - Die Ausfallwahrscheinlichkeiten p einer einzelnen Festplatte müssen statistisch unabhängig voneinander sein

Massenspeicher
RAID 1

RAID 1 bietet hohe Datenredundanz durch *Mirroring*

- Alle Daten werden identisch auf alle verfügbaren Festplatten geschrieben
  > Lesezugriffe können, ähnlich wie bei RAID0, parallel ablaufen



Massenspeicher
RAID 1

Auch RAID 1 hat einige Nachteile

- Das RAID kann maximal so groß sein, wie die kleinste beteiligte Festplatte

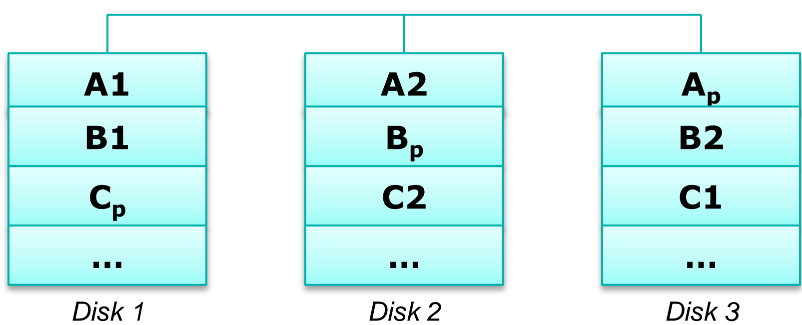
- RAID1 ist KEIN Ersatz für Datensicherung, da alle Schreib-operationen auch auf allen Festplatten vorgenommen werden
  - > Versehentlich gelöschte oder geänderte Daten sind trotzdem verloren

Massenspeicher
RAID 5

RAID 5 bietet höhere Transferraten und hohe Datenredundanz

- Daten werden, wie bei RAID0, in gleich großen Blöcken auf die beteiligten Platten gelegt
- Jeweils eine Platte enthält statt eines Datenblocks einen Paritätsblock
  - > Die Berechnung des Paritätsblocks erfolgt durch XOR-Verknüpfung aller zugehöriger Datenblöcke
  - > Datenintegrität ist bei Ausfall von maximal einer Platte gewährleistet (Rekonstruktion möglich)

RAID5



Massenspeicher
RAID 5

Redundanz bei RAID 5 wird mit XOR-Operationen erreicht

- Original bei drei Platten

D1	D2	D3	par
1	1	0	0
0	1	1	0
0	1	0	1

- Wiederherstellung bei Ausfall durch Auswertung der Parität

D1	D2	D3	par
1	?	0	0
?	1	1	0
0	1	?	1

Massenspeicher
RAID 5

Die Speicherkapazität eines RAID 5 mit n Festplatten errechnet sich wie folgt

- $(n-1) * (Kapazität\ der\ kleinsten\ Festplatte)$

- Zweiphasiges Schreibverfahren
  - Schreiben der Daten
  - Schreiben / Aktualisieren des Paritätsblocks

Massenspeicher
RAID Kombinationen

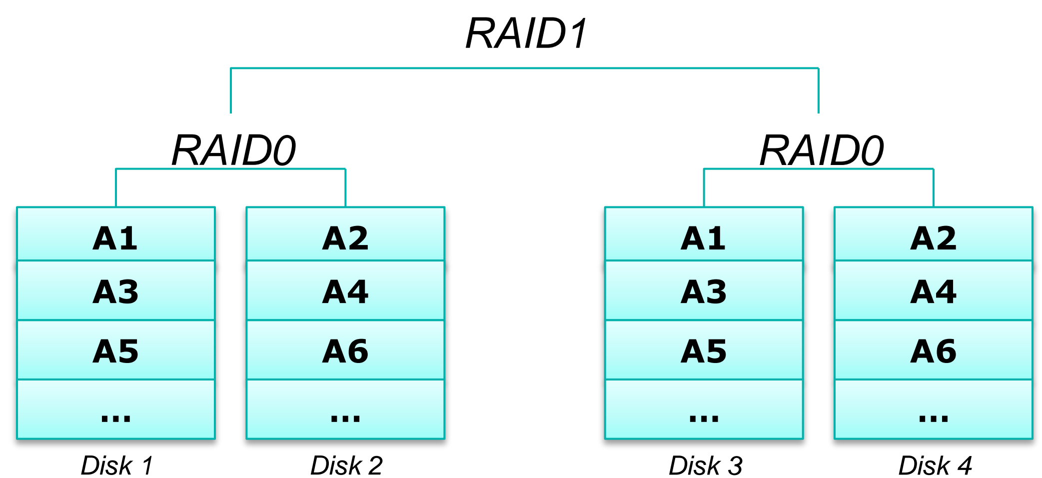
Kombinationen von RAID-Systemen sind möglich

- Mehrere RAID-Systeme eines bestimmten Typs werden zu einem RAID-System zusammengefasst
  - > z.B. RAID 10, RAID 01, …
- RAID-Kombinationen sind theoretisch n-stufig möglich
  - > z.B. RAID 100

Massenspeicher
Beispiele für RAID Kombinationen

RAID 01

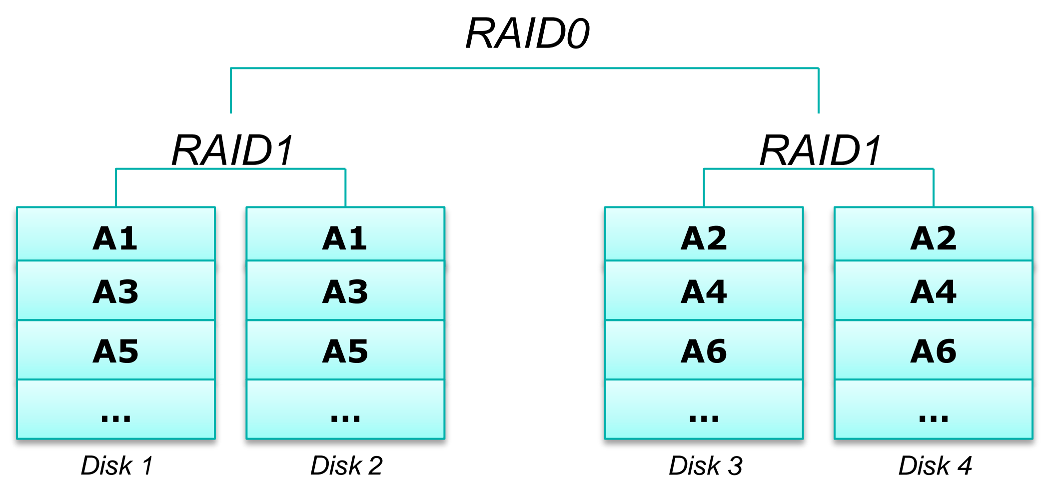
- Mehrere RAID0 werden zu einem RAID1 zusammengefasst



Massenspeicher
Beispiele für RAID Kombinationen

RAID 10

- Mehrere RAID1 werden zu einem RAID0 zusammengefasst



© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Massenspeicher
Festplatten

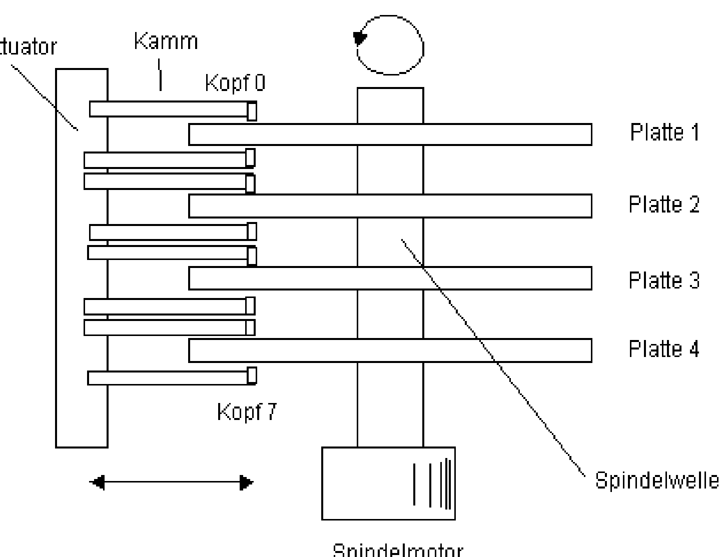
Festplatte, Hard Disk, HD, Hard Disk Drive, HDD, …

- Information wird durch Magnetisierung abgelegt
  - Gehäuse der Festplatte beinhaltet mehrere, auf einer Achse übereinander montierten, runden Platten, welche mit einer magnetisierbaren Schicht überzogen sind
  - Enthaltene Platten haben alle denselben Durchmesser (3,5‟, 2,5‟, 1,8‟ Formfaktor)
- Heutzutage etwa 1-3 TB Speicherkapazität üblich

Massenspeicher
Aufbau einer Festplatte

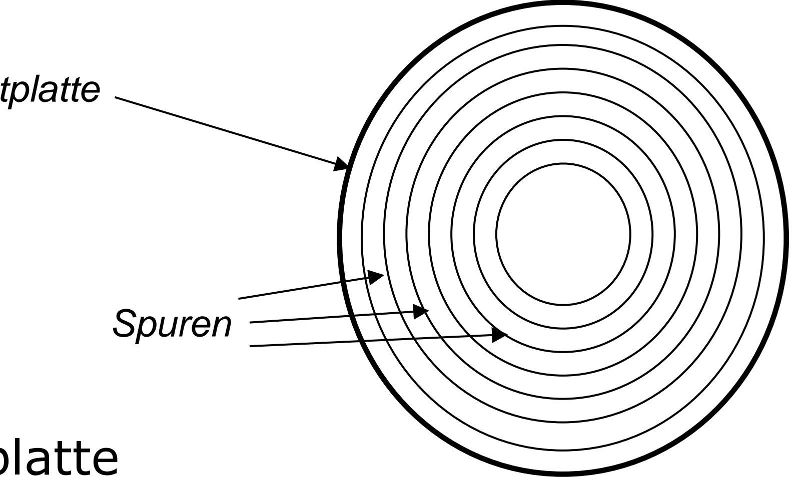
Eine Festplatte funktioniert wie folgt:

- Schreib-/Leseköpfe werden durch einen zentralen Kamm über die Platten bewegt
  - > Etwa 1 µm Abstand zwischen Platte und Kopf
- „Landing Zone" zum Parken der Köpfe
  - > Kommen die Köpfe mit einem Plattenbereich in Berührung, welcher Daten enthält, werden diese zerstört
- Platten rotieren mit konstanter Um-drehungszahl (5400 – 15000 rpm)



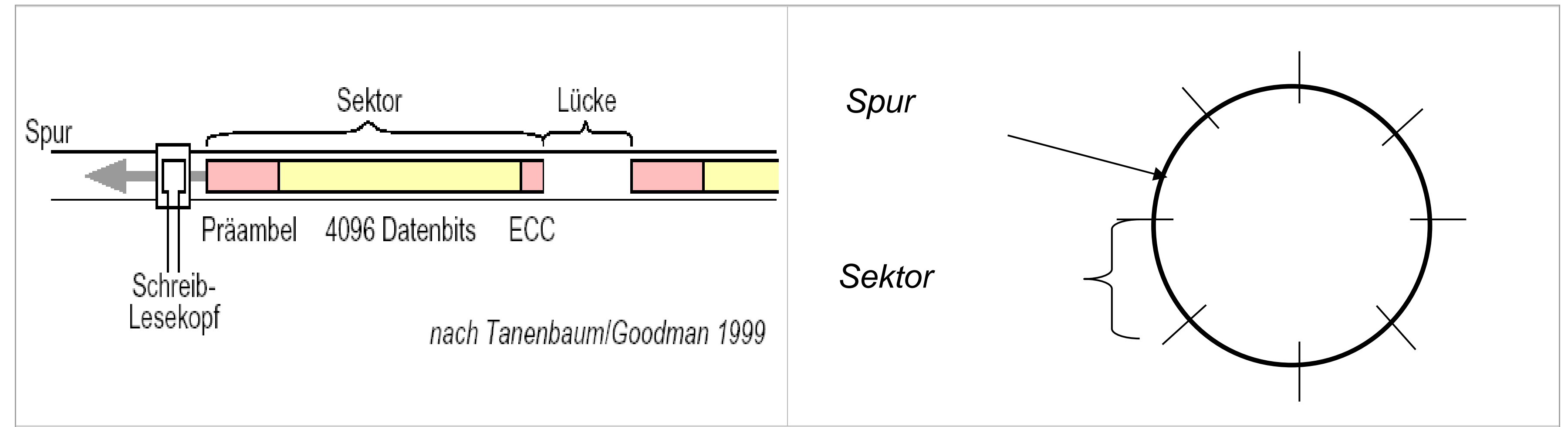
Massenspeicher
Spuren und Sektoren

Die Oberfläche der Platten ist in konzentrische Kreise, die Spuren, eingeteilt



Die Spuren sind in Abschnitte fester Speichergröße, die Sektoren, unterteilt

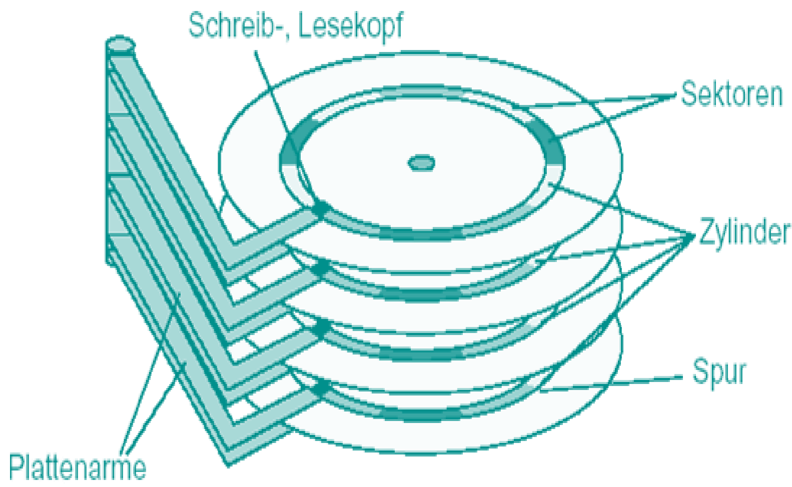
- Kleinste adressierbare Einheit einer Festplatte



Massenspeicher
Zylinder

Als Zylinder werden die übereinander liegenden Spuren der Platten bezeichnet

- Alle Zylinder können von den entsprechenden Schreib-/Leseköpfen gleichzeitig zugegriffen werden



Massenspeicher
Kenngrößen

Charakteristische Größen

- **kontinuierliche Übertragungsrate** *(sustained data rate)*
  > Datenmenge, die die Festplatte beim Lesen aufeinander folgender Sektoren im Mittel pro Sekunde überträgt (Schreiben ähnlich)

- **mittlere Zugriffszeit ([data] access time)**
  > der Spurwechselzeit *(seek time*, Mittelwert beim Wechsel von einer zufälligen zu einer anderen zufälligen Spur)

  > der Latenzzeit *(latency, i*m Mittel dauert es eine halbe Umdrehung, bis ein bestimmter Sektor unter dem Kopf vorbeikommt)

  > die Kommando-Latenz *(controller overhead,* Zeit, die der Festplattencontroller damit verbringt, das Kommando zu interpretieren)

Massenspeicher
Kenngrößen

Charakteristische Größen

Exemplarische Entwicklung der Plattengeschwindigkeit über der Zeit

Kategorie	Jahr	Modell	Größe	Drehzahl	Datenrate	Spurwechsel	Latenz	mittlere Zugriffszeit
Server	1993	IBM 0662	1 GB	5.400 min^{-1}	ca. 5 MB/s	8,5 ms	5,6 ms	15,4 ms
Server	2002	Seagate Cheetah X15 36LP	18–36 GB	15.000 min^{-1}	52–68 MB/s	3,6 ms	2,0 ms	5,8 ms
Server	2007	Seagate Cheetah 15k.6	146–450 GB	15.000 min^{-1}	112–171 MB/s	3,4 ms	2,0 ms	5,6 ms
Desktop	1989	Seagate ST296N	80 MB	3.600 min^{-1}	0,5 MB/s	28 ms	8,3 ms	40 ms
Desktop	1993	Seagate Marathon 235	64–210 MB	3.450 min^{-1}	?	16 ms	8,7 ms	24 ms
Desktop	1998	Seagate Medalist 2510–10240	2,5–10 GB	5.400 min^{-1}	?	10,5 ms	5,6 ms	16,3 ms
Desktop	2000	IBM Deskstar 75GXP	20–40 GB	5.400 min^{-1}	32 MB/s	9,5 ms	5,6 ms	15,3 ms
Desktop	2009	Seagate Barracuda 7200.12	160–1000 GB	7.200 min^{-1}	125 MB/s	8,5 ms	4,2 ms	12,9 ms
Notebook	1998	Hitachi DK238A	3,2–4,3 GB	4.200 min^{-1}	8,7–13,5 MB/s	12 ms	7,1 ms	19,3 ms
Notebook	2008	Seagate Momentus 5400.6	120–500 GB	5.400 min^{-1}	39–83 MB/s	14 ms	5,6 ms	18 ms

Massenspeicher
Kenngrößen

Festplatten - SATA - 2,5 Zoll (26)

Produkt	Kapazität	ms/Cache/U	Preis pro GB	Bewertungen	Preis
Hitachi HTS547564A9E384 640 GB SATA 300, Travelstar 5K750	640 GB	12/8/5400	€ 0,09*	★★★★★ 1 Bewertung	€ 59,90*
Hitachi HTS543232A7A384 320 GB SATA 300, Travelstar Z5K320, 7 mm	320 GB	5,5/8/5400	€ 0,13*	» jetzt bewerten	€ 41,49*
Hitachi HTS545025B9A300 250 GB SATA 300, Travelstar 5K500.B	250 GB	12/8/5400	€ 0,15*	★★★★★ 2 Bewertungen	€ 36,49*
Hitachi HTS545032B9A300 320 GB SATA 300, Travelstar 5K500.B	320 GB	12/8/5400	€ 0,12*	» jetzt bewerten	€ 39,99*
Hitachi HTS545050B9A300 500 GB SATA 300, Travelstar 5K500.B	500 GB	12/8/5400	€ 0,10*	★★★★☆ 22 Bewertungen	€ 51,90*
Hitachi HTS547575A9E384 750 GB SATA 300, Travelstar	750 GB	12/8/5400	€ 0,11*	★★★★★ 2 Bewertungen	€ 84,90*

Massenspeicher
Solid State Drives (SSD)

SSDs sind die neueste Entwicklung im Festplattensektor

- Vorteile
  - SSDs haben im Vergleich zu herkömmlichen Festplatte keine mechanischen Bauteile mehr
    - Keine Stoßanfälligkeit
    - Wesentlich kürzere Zugriffszeit, da keine mechanischen Teile in Position gebracht werden müssen
    - Keine Geräuschentwicklung
  - SSDs verbrauchen weniger Energie als herkömmliche Festplatten
  - SSDs bieten besseren Datendurchsatz
- Nachteil
  - SSDs haben einen wesentlich höheren Preis pro Speichereinheit

Massenspeicher
Solid State Drives (SSD)

SSDs

Kingston

Solid State Drive - SATA - 1,8 Zoll (3)

Produkt	Kapazität	lesen / schr.	Preis pro GB	Bewertungen	Preis
Kingston SSDNow V+ Series 1,8" SSD 128 GB SATA 300, SVP180S2/128G	128 GB	230 / 180 MB/s	€ 1,64*	» jetzt bewerten	€ 209,90*
Kingston SSDNow V+ Series 1,8" SSD 64 GB SATA 300, SVP180S2/64G	64 GB	230 / 180 MB/s	€ 2,-*	» jetzt bewerten	€ 127,90*
Kingston SSDNow V+ Series 1,8" SSD 256 GB SATA 300, SVP180S2/256G	256 GB	230 / 180 MB/s	€ 1,77*	» jetzt bewerten	€ 454,-*

» nach oben

Massenspeicher
CDs / DVDs / Blu-rays

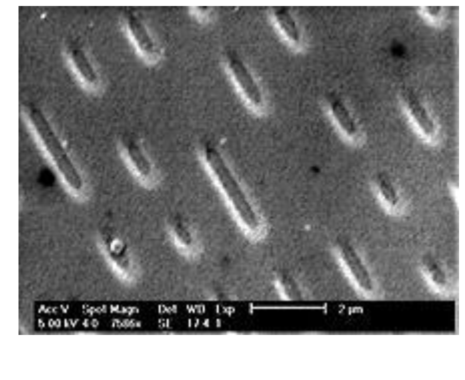
Der Datenträger besteht aus einer Kunststoffscheibe mit 12 cm Durchmesser, welche mit einer reflektierenden Aluminiumschicht bedampft wurde

- Kapazitäten
  - CD: 540 MB – 900 MB (SL)
  - DVD: 4,7 GB (SL), 8,5 GB (DL)
  - Blu-ray: 25 GB (SL), 50 GB (DL)

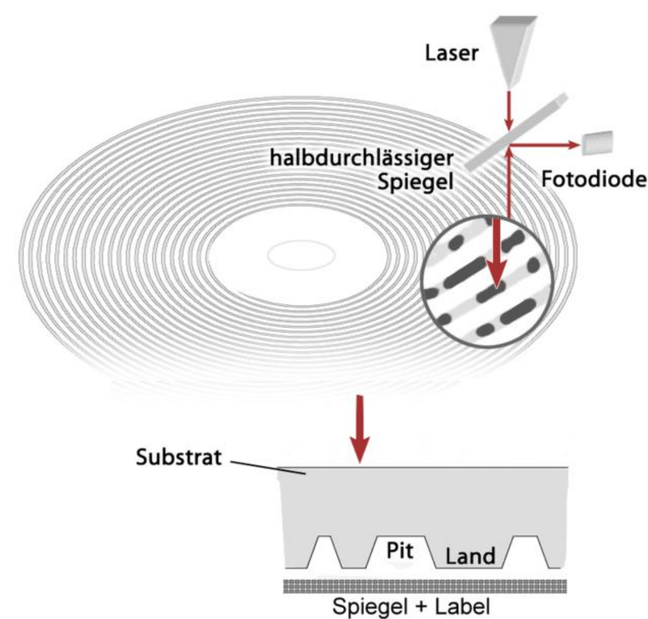
- Datenraten
  - CD: 1,228 MBit/s
  - DVD: 11,08 MBit/s
  - Blu-ray: 36,00 MBit/s
- Optische Speicherung der Daten durch sog. *Pits* und *Lands*

Massenspeicher
Prinzipielle Arbeitsweise

Ein fein gebündelter Laserstrahl wird von einer Laserdiode über Optik mit kippbarem Spiegel auf die Oberfläche der optischen Platte fokussiert



- Die Intensität des reflektierten Strahls wird von einem Sensor erfasst und liefert das Bitsignal
  - > Helligkeitsschwankungen aufgrund destruktiver Interferenz
  - > Teilwellen mit leicht unterschiedlichem Laufweg
  - > Pits erzeugen reduzierte Helligkeit
- Wellenlängen der Laserstrahlen
  - > CD: 780 nm (Infrarot)
  - > DVD: 650 nm (Rot)
  - > Blu-ray: 405 nm (Violett)
- Abtasten von unten, Pressung von oben (Pits sind Vertiefungen)



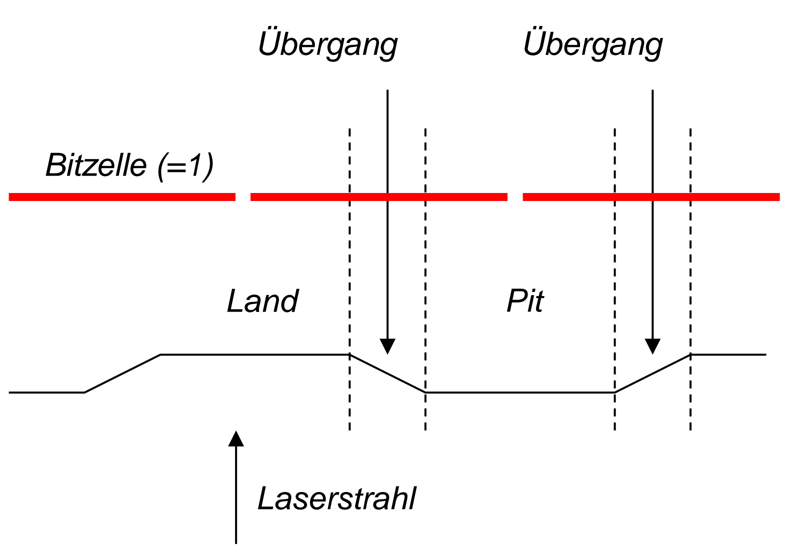
Massenspeicher
Pits und Lands

Pits und Lands sind kleine Vertiefungen und Erhebungen auf der Oberfläche des Datenträgers

- Ein Abschnitt einer Spur von ca. 0.3 µm Länge ist eine Bitzelle
- NRZ-M-Codierung (non-return-to-zero-mark – 1 sind Zustandsänderungen)
  - > Bitzelle mit Übergang stellt eine „1" dar
  - > Bitzelle ohne Übergang stellt ein „0" dar

Auslesen der Information schwierig

- Modulation (EFM=Eight-to-Fourteen Mod.)
  - > CD: EFM-Modulation
  - > DVD: EFMplus-Modulation
  - > Blu-ray: 17PP-Modulation



7 – Eingabe und Ausgabegeräte

Tastatur, Maus, Touchscreen, Bildschirm

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Wichtige Ein- und Ausgabegeräte kennen
- Technische Realisierungen grundlegend verstehen
- Kriterien zur Bewertung und Auswahl kennen

Eingabe- und Ausgabegeräte
Eingabegeräte

HID – human interface device

Eingabe von Hand	Lesegeräte	A/V-Eingabegeräte
- **Tastaturen**	- **Kartenleser**	- **Mikrofon**
- **Zeigeeinrichtungen** für Bildschirme (Joystick, Maus)	- **Belegleser**	- **Telefon**
- **Schreibstifteingabe** Touchpanels, Smartboards	- **Scanner** (Bildabtaster)	- **Abspielgeräte**
		- **Kameras**

Eingabe- und Ausgabegeräte
Ausgabegeräte

Bildschirme

- **Kathodenstrahl-Monitore**
- **Flachbildschirme**
  - LCD
  - Plasma
- **Projektoren**

Schreib- und Druckgeräte

- **Zeichendrucker**
- **Zeilendrucker**
- **Seitendrucker**
- **Plotter**

Audioausgabegeräte

- **Lautsprecher**
- **Kopfhörer**

Eingabe- und Ausgabegeräte
Tastaturen

Eingabe von Text

Spezialfunktionen

Eingabe- und Ausgabegeräte
Tastaturen

Nachteile von Tastaturen

- Zeitaufwendige Dateneingabe
  - Tippen einzelner Zeichen
  - Cursorpositionierung in einem Text
- Nur für Sprachen mit wenigen Zeichen geeignet
- Unterschiedliche Yeichens’tye & Tastenbelegungen
- Relativ lange “Übungszeit”
- Kein Eingabe bildlicher Daten möglich

Eingabe- und Ausgabegeräte
Tastaturen

Eingabe- und Ausgabegeräte
Tastaturen

Für einige Anwendungsfälle gibt es spezielle Tastaturen, die die Arbeit erleichtern

Eingabe- und Ausgabegeräte
Mäuse

Zeigen & Klicken

Markieren

Scrollen

Eingabe- und Ausgabegeräte

Mäuse

Das Bild kann zurzeit nicht angezeigt werden.

Eingabe- und Ausgabegeräte
Mäuse

Die Bewegung einer Maus

- Die Bewegungen werden optisch oder mechanisch erfasst
- Die Steuerimpulse werden über Kabel, Funk oder Infrarotlicht übertragen
- Beispiel: Optische Maus
  - > Leuchtdiode oder Laser, Kamera, Rechnereinheit
  - > Ablauf
    - > Die Lichtquelle beleuchtet den Untergrund
    - > Die Kamera macht Bilder vom Untergrund
    - > Die Rechnereinheit vergleich aufeinanderfolgende Bilder und gibt den entsprechenden Steuerimpuls aus

Eingabe- und Ausgabegeräte
Touchscreens

Touchscreens sind berührungsempfindliche Bildschirme

- Interaktion geschieht direkt mit dem Touchscreen

- Es gibt verschiedene technische Ansätze, die wichtigsten sind …
  - Resistive Touchscreens
    - Der Touchscreen reagiert auf mechanischen Druck
    - Als Eingabemedium kann alles möglich verwendet werden (Finger, Stift, …)
  - Kapazitive Touchscreens
    - Der Touchscreen reagiert auf elektrische Spannung
    - Als Eingabemedium können nur elektrisch leitende Dinge verwendet werden (Finger, spezielle Stifte, …)

RWTHAACHEN UNIVERSITY
MATSE Ausbildung
JÜLICH FORSCHUNGSZENTRUM
FH AACHEN UNIVERSITY OF APPLIED SCIENCES

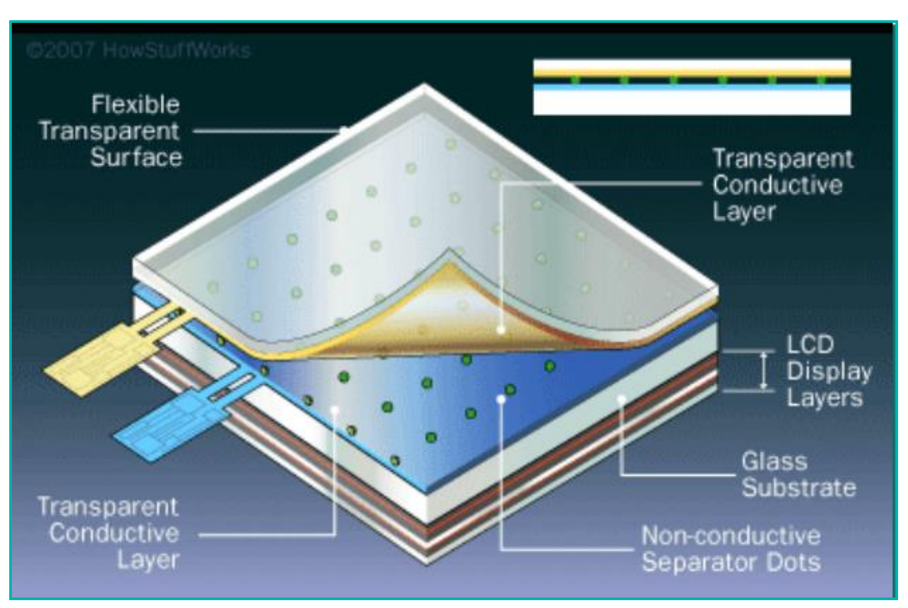
Eingabe- und Ausgabegeräte
Touchscreens

Beispiel: Kapazitiver Touchscreen

- Der Touchscreen besteht aus zwei Glasplatten
  - > Jeweils beschichtet mit transparentem (leitendem) Metalloxyd
  - > Beschichtung in parallelen Leiterbahnen pro Glasplatten
    - > Ausrichtung der Leiterbahnen auf den beiden Glasplatten ist senkrecht zueinander

- Elektrisches Feld im Normalzustand exakt definiert und messbar

- Berührung des Touchscreens mit einem leitenden Gegenstand verändert das elektrische Feld
  - > Punkt der Veränderung kann bestimmt werden



Eingabe- und Ausgabegeräte
Bildschirme

Eingabe- und Ausgabegeräte
Bildschirme

Grundlegendes zu Bildschirmen …

- Die Größe eines Bildschirmes wird in Zoll angegeben
  - Typische Größen
    - Smartphones: 3" – 6"
    - Tablets: 7", 8.9", 10.1"
    - Netbooks 10" – 13"
    - Laptops: 13" – 17"
    - Desktop: ab 19"
- Die Auflösung eines Bildschirmes wird in Pixeln angegeben
  - Pixel sind Bildpunkte, welche die „Schärfe" des Bildschirms bestimmen
  - Ein Pixel besteht in der Regel aus roten, grünen und blauen Subpixeln
  - Typische Formate
    - 1024x768 (4:3), 1280x1024 (5:4), 1366x768 (16:9), 1920x1080 (16:9)

Eingabe- und Ausgabegeräte
Bildschirme

Es gibt verschiedene Arten von Bildschirmen, die wichtigsten sind…

- CRT-Bildschirme *(CRT = Cathode Ray Tube → Kathodenstrahlröhre)*
  - Ein Elektronenstrahl bringt Phosphor-Pixel zum Leuchten
    - Helligkeit des Pixels ist abhängig von der Intensität des Elektronenstrahls
- Plasma-Bildschirme
    - „Anzünden" eines Plasma in winzigen Zellen bringt entsprechende Pixel zum Leuchten
    - Helligkeit des Pixels ist abhängig von der Leuchtdauer des Plasmas
- LCD-Bildschirme *(LCD = Liquid Cristal Display → Flüssigkristalle)*
  - Gleichmäßige Hintergrundbeleuchtung wird durch Flüssigkristalle polarisiert
    - Helligkeit des Pixels ist abhängig von der Polarisation

Eingabe- und Ausgabegeräte
Alles zusammen

8 - Betriebssysteme

Allgemeines, Speicherverwaltung

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Aufgaben eines Betriebssystemen
  - Scheduling
  - Speicherverwaltung
- Bekannte Betriebssysteme benennen

Betriebssysteme
Windows, oder was?

Was ist ein Betriebssystem?

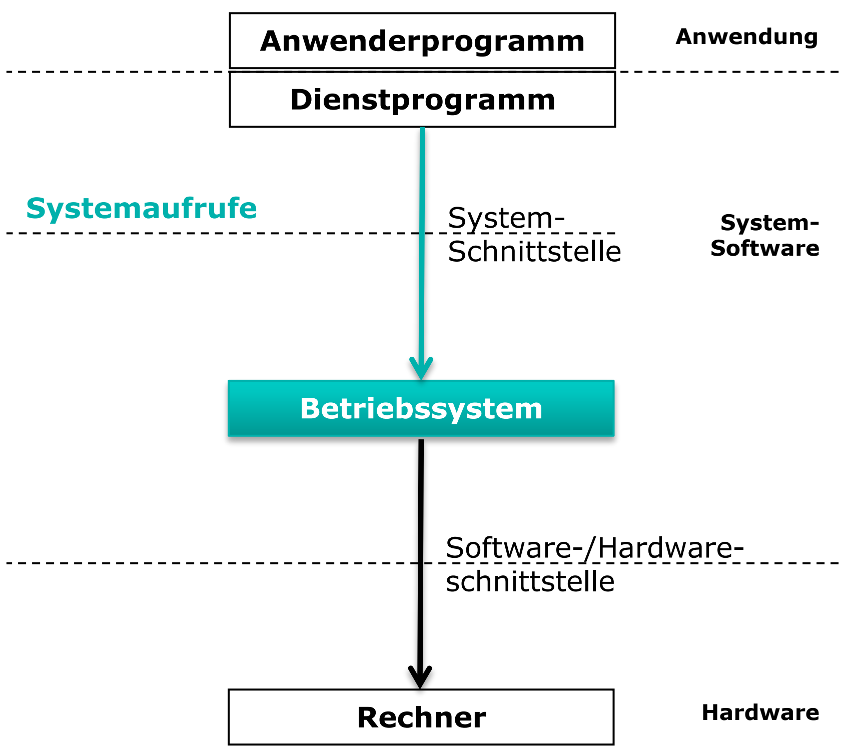
- Versuch einer Definition
  - *„Als Betriebssystem bezeichnet man die Software, die den Ablauf von Programmen auf der Hardware steuert und die vorhandenen Betriebsmittel verwaltet."*
- Zu einem Betriebssystem gehört ggf. auch die Bereitstellung von Dienstprogrammen
  - Einfacher Editor, Übersetzer, Datenfernverarbeitung, Netzverwaltung, …
- Betriebssysteme stellen für den Anwender die Schnittstelle zur Maschine dar

Betriebssysteme
Eigenschaften eines Betriebssystems

Das Betriebssystem ist ein grundlegendes Systemprogramm

- Das Betriebssystem liegt als Softwareschicht zwischen Hardware und Anwendungen

- Die Komplexität der Hardware wird durch das Betriebssystem vor dem Anwender verborgen
  - > Hardware Abstraction Layer (HAL)



Betriebssysteme
Anforderungen an ein Betriebssystems

Hohe Zuverlässigkeit

- Korrektheit
- Sicherheit
- Verfügbarkeit
- Robustheit
- Schutz von Benutzerdaten

Hohe Leistung

- Gute Auslastung der Systemressourcen
- Kleiner Verwaltungsaufwand
- Hoher Durchsatz
- Kurze Reaktionszeit

Hohe Benutzerfreundlichkeit

- Angepasste Funktionalität
- Einfache Benutzerschnittstelle
- Hilfestellung

Einfache Wartbarkeit

- Einfache Upgrades
- Einfacher Erweiterbarkeit
- Portierbarkeit

... und geringe Kosten!

Betriebssysteme
Stapelverarbeitungssysteme

Das Betriebssystem arbeitet nicht interaktiv, sondern nimmt nur Jobs von Benutzern an

- Jobs werden in eine Job-Queue gelegt und nacheinander abgearbeitet
- Jobs erfordern keinerlei Benutzerinteraktion, d.h. laufen vollständig autonom ab
- Ausgabe der Ergebnisse eines Jobs erfolgt über Dateien oder sonstige Ausgabegeräte (Drucker, …)
  - Ausgabe über den Monitor ist nicht sinnvoll…

Betriebssysteme
Interaktive Systeme / Dialogsysteme

Das Betriebssystem arbeitet interaktiv

- Programme werden durch Benutzereingaben gesteuert
  - Tastatureingaben
  - Mauseingaben
  - …
- Ausgabe der Ergebnisse eines Programms kann über jedes beliebige Ausgabegerät erfolgen
  - Im Gegensatz zu Stapelverarbeitungssystemen, sind auch Ausgaben auf dem Monitor sinnvoll

Betriebssysteme
Echtzeitsysteme

Echtzeitsysteme sind reaktive Systeme, welche über Sensoren und Aktoren mit der Umwelt interagieren

- Aktionen erfolgen als Reaktion auf eine Änderung von Sensorwerten

- Es gibt zeitliche Anforderungen an das System
  - Harte Zeitanforderungen
    - Reaktionen **müssen** innerhalb eines definierten Zeitfensters erfolgen
    - Das Überschreiten des Zeitfensters macht das Ausführen / Beenden der Reaktion überflüssig
  - Weiche Zeitanforderungen
    - Überschreitungen des Zeitfensters werden toleriert, **können** aber zur Abnahme der Ergebnisqualität führen

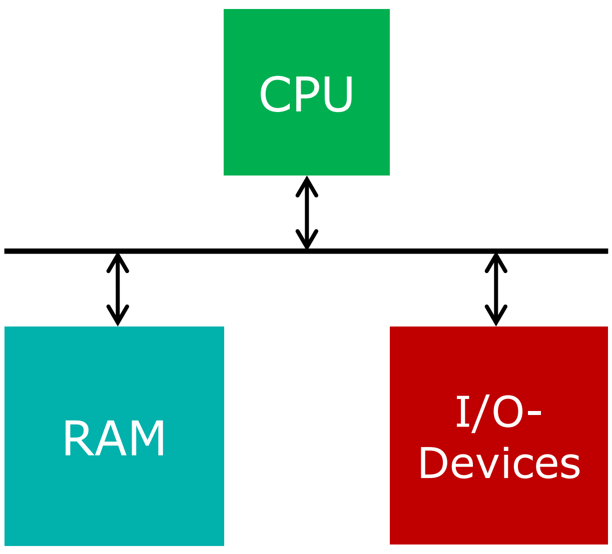
Ein Betriebssystem hat viele Aufgaben …

- Betriebsmittelverwaltung
- Prozessverwaltung
- Speicherverwaltung
- Dateiverwaltung
- Rechteverwaltung
- …

Betriebssysteme
Betriebsmittel

Betriebsmittel sind …

- Aktive, zeitlich aufteilbare
- Passive, nur exklusiv nutzbare
- Passive, räumlich aufteilbare



Betriebssysteme
Betriebsmittelverwaltung

Die Betriebsmittelverwaltung besteht im wesentlich aus der …

- … Zuteilung von Ein- und Ausgabegeräten
  > Regelung des Zugriffs einzelner Programme auf nur exklusiv nutzbare Betriebsmittel

  > Konfliktvermeidung

- … Verwaltung des Dateisystems
  > Bearbeitung von Dateizugriffen aus Programmen heraus
    > Existiert die angeforderte Datei?

    > Darf der ausführende User diese Datei öffnen?

    > …

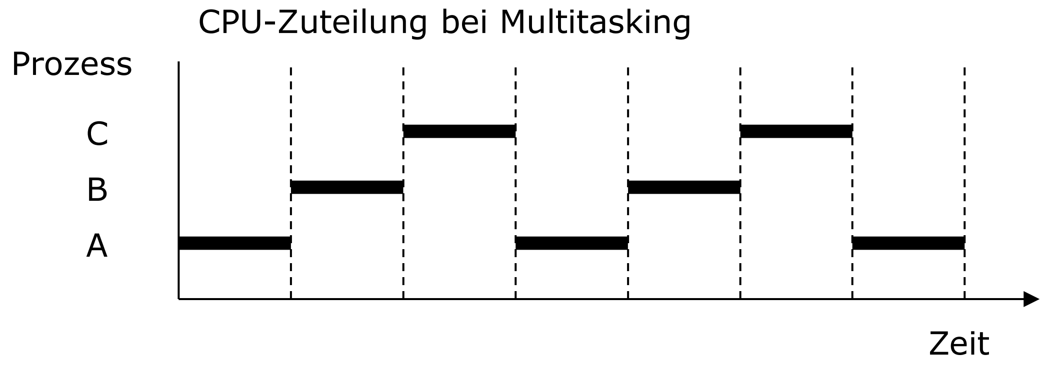
  } Rechteverwaltung

Ein Prozess …

- … ist die Abstraktion eines sich in Ausführung befindlichen Programms
  - Ein Prozess besteht aus den Programmbefehlen und dem Prozesskontext
  - Der Prozesskontext besteht aus dem privaten Adressraum des Programms, geöffnete Streams (Dateien, Sockets, …) und abhängigen Prozessen
- … wird über die Prozessverwaltung gesteuert
  - Die Prozessverwaltung steuert die Zuteilung von Betriebsmitteln an den Prozess
    - CPU
    - Speicher
    - Ein- und Ausgabegeräte
    - …

Der Prozessor kann zwischen mehreren Prozessen hin und her geschaltet werden

- Im Allgemeinen wird sogenanntes preemptives Multitasking verwendet
  - Das Betriebssystem entscheidet, wann welcher Prozess zur Ausführung kommt
  - Jeder Prozess wird für einige Millisekunden (Zeitscheibe, timeslice) ausgeführt
  - Der Benutzer erhält dadurch den Eindruck von Parallelität



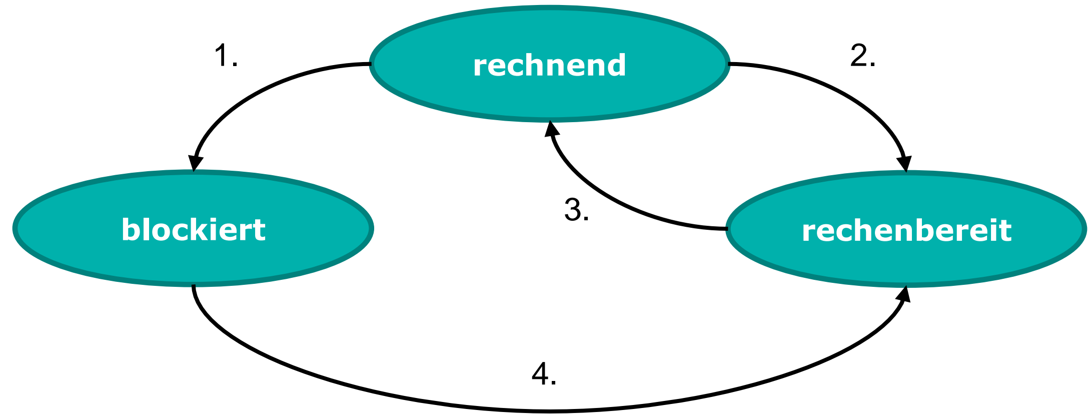
Im Gegensatz zum preemptiven Multitasking bestimmt ein Prozess selbst, wann er den Prozessor an andere Prozesse abgibt

- Nachteile
  - Ein einzelner Prozess kann bei bestimmten Fehlern, z.B. Endlosschleifen, das gesamte System zum Absturz bringen
  - Selbst das Betriebssystem kann nicht mehr rechnen, wenn ein Prozess den Prozessor nicht wieder freigibt

Betriebssysteme
Prozesszustände

Ein Prozess kann sich in folgenden Zuständen befinden

- Rechnen: Der Prozessor ist dem Prozess zugeteilt
- Blockiert: Der Prozess kann nicht ausgeführt werden, bis ein externes Ereignis auftritt
- Rechenbereit: Der Prozess ist ausführbar, aber der Prozessor ist einem anderen Prozess zugeteilt

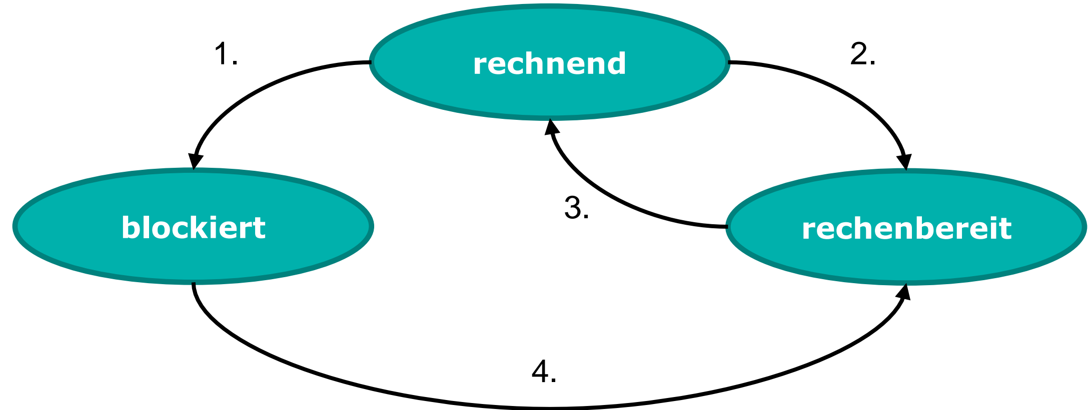


© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Betriebssysteme
Prozesszustände

Der Prozessscheduler organisiert die Übergänge der Prozesse zwischen den einzelnen Zuständen

1. Der Prozess wartet, z.B. auf Eingabe des Benutzers
2. Die Zeitscheibe des Prozesses ist abgelaufen oder ein höher priorisierter Prozess muss ausgeführt werden
3. Der Prozess bekommt eine neue Zeitscheibe zugeteilt
4. Das Ereignis, auf welches ein Prozess nach 1. gewartet hat, ist eingetreten



Betriebssysteme
Scheduling

Ein guter Scheduling-Algorithmus muss einige Anforderungen erfüllen

- Fairness: Jeder Prozess erhält einen gerechten Anteil der CPU-Zeit
- Effizienz: Die CPU und andere Ressourcen sind möglichst ausgelastet
- Einhaltung der Systemregeln
- Weitere Anforderungen hängen vom Einsatzgebiet ab
  - Stapelverarbeitungssysteme
  - Dialogsysteme
  - Echtzeitsysteme

Betriebssysteme
Scheduling auf Stapelverarbeitungssystemen

Zusätzlich Anforderungen an den Scheduler auf einem Stapelverarbeitungssystem

- Möglichst hohe CPU-Auslastung
  - Es sollte nicht vorkommen, dass die CPU Leertakte hat
- Job-Durchsatz
  - Die Anzahl der bearbeiteten Aufgaben pro Zeiteinheit sollte maximal sein
- Minimale Durchlaufzeit
  - Die Zeit, welche ein Job von Ankunft in der Job-Queue bis zur Fertigstellung des Jobs benötigt, sollte minimal sein

Betriebssysteme
Scheduling auf Dialogsystemen

Zusätzlich Anforderungen an den Scheduler auf einem Dialogsystem

- Kurze Antwortzeiten
  - Der Benutzer sollte nicht das Gefühl haben, auf Reaktionen des Systems, z.B. auf Mausklicks oder Tastatureingaben, warten zu müssen
  - Prozesse, die Interaktion mit dem Benutzer erfordern, sollten vor anderen Prozessen bevorzugt werden
- Proportionalität
  - Die Antwortzeit verschiedener Prozesse sollte mit der Benutzererwartung übereinstimmen
  - Aus Benutzersicht „einfache" Aufgaben sollten schneller erledigt werden, als aus Benutzersicht „schwierige" Aufgaben

Zusätzlich Anforderungen an den Scheduler auf einem Echtzeitsystem

- Einhaltung von Zeitfenstern
  - Der Scheduler muss einen Überblick über die Zeitfenster verschiedener Prozesse haben und diesen entsprechend Rechenzeit zuweisen
- Vorhersagbarkeit
  - Das System muss deterministisch agieren / reagieren

Betriebssysteme
Schedulingstrategien

Einige bekannte Scheduling-Strategien sind…

- First-Come, First Serve (Stapelverarbeitungssysteme)
- Shortest Job First (Stapelverarbeitungssysteme)
- Prioritätsscheduling (alle Systeme)
- Round Robin (alle Systeme)
  - Alle laufenden Prozesse bekommen der Reihe nach eine (gleichgroße) Zeitscheibe auf der CPU zugeteilt

Betriebssysteme
First Come, First Serve

Die Prozesse werden nach der Reihenfolge ihres Einfügens in die Job-Queue eines Stapelverarbeitungssystems bearbeitet

- Die Zuteilung des Prozessors an andere Prozesse findet nur statt, wenn ein laufender Prozess zu warten beginnt oder sich beendet

- Jeder Prozess kommt garantiert an die Reihe

- Kurze Prozesse müssen unter Umständen sehr lange warten, bis sie ausgeführt werden
  - > Unverhältnismäßig langes Warten auf das Ergebnis

A	B	C	D

Betriebssysteme
Shortest Job First

Die Prozesse werden aufsteigend nach ihrer geschätzten Ausführungszeit bearbeitet

- Große Prozesse kommen möglicherweise nie an die Reihe, wenn immer wieder kleinere Prozesse in die Job-Queue eingefügt werden

- Die Wartezeit auf das Ergebnis eines Prozesses verhält sich in etwa proportional zur Ausführungszeit des Prozesses

C	D	A	B

Betriebssysteme
Beispiel: FCFS vs. SJF

Gegeben sind 4 Prozesse mit den folgenden Ausführungszeiten

- Prozess A: 6 Minuten
- Prozess B: 10 Minuten
- Prozess C: 2 Minuten
- Prozess D: 2 Minuten

Betriebssysteme
Beispiel: FCFS vs. SJF

Die mittlere Verweilzeit berechnet sich dann wie folgt

- First Come, First Serve
  - > Prozess A: 6 Minuten
  - > Prozess B: 16 Minuten
  - > Prozess C: 18 Minuten
  - > Prozess D: 20 Minuten
  - > Mittlere Verweilzeit: $\frac{6+16+18+20}{4} = 15$ Minuten
- Shortest Job First
  - > Prozess A: 10 Minuten
  - > Prozess B: 20 Minuten
  - > Prozess C: 2 Minuten
  - > Prozess D: 4 Minuten
  - > Mittlere Verweilzeit: $\frac{10+20+2+4}{4} = 9$ Minuten

Betriebssysteme
Prioritätsscheduling auf Dialogsystemen

Jedem laufenden Prozess wird eine Priorität zugewiesen

- Zuweisung der Priorität wird vom Betriebssystem gesteuert
  - Es gibt dynamische und statische Zuweisung
- Es wird immer dem Prozess mit der höchsten Priorität eine Zeitscheibe zugeteilt
- Neu hinzukommende Prozesse hoher Priorität verdrängen rechnende Prozesse niedriger Priorität
  - Auch der Übergang nach Rechenbereit eines hoch priorisierten Prozesses verdrängt niedrig priorisierte Prozesse

Betriebssysteme
Statische vs. Dynamische Priorität

Es gibt zwei Arten von Prioritätszuweisung

- Statische Priorität
  - > Jeder Prozess erhält bei seinem Start eine feste Priorität
  - > Der Prozess mit der höchsten Priorität bekommt als nächstes eine Zeitscheibe zugeteilt
    - > Gibt es mehrere Prozesse mit der gleichen Priorität werden diese im Round-Robin-Verfahren bearbeitet
  - > Wird oft in Echtzeitsystemen verwendet

- Dynamische Priorität
  - > Jedem Prozess wird eine Anfangspriorität zugeordnet
  - > Der Prozess mit der höchsten Priorität bekommt als nächstes eine Zeitscheibe zugeteilt
    - > Die Prioritäten der Prozesse werden dynamisch geändert
    - > Es gibt verschiedene Verfahren der Realisierung

Betriebssysteme
Multilevel-Feedback-Queue

Die MFQ ist ein Beispiel für dynamische Prioritätszuweisung

- Die Prozesse werden anhand ihres bisherigen Ressourcenverbrauchs priorisiert
- Es werden mehrere FIFO-Queues benutzt
  - Neue Prozesse werden in der höchst priorisierten Queue eingefügt
- Prioritäten werden abhängig vom Verhalten des Prozesses geändert
  - Der Prozess gibt den Prozessor freiwillig ab
    - Der Prozess wird in dieselbe Queue wieder eingefügt
  - Der Prozess verbraucht seine gesamte Zeitscheibe
    - Der Prozess wird in der nächst niedriger priorisierten Queue wieder eingefügt

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Betriebssysteme
Speicherverwaltung

Die Speicherverwaltung eines Betriebssystems ermöglicht den Prozessen den Zugriff auf den Arbeitsspeicher des Computers

- Der Zugriff soll effizient, komfortabel und sicher sein
  - Der Zugriff auf den Speicherbereich eines anderen Prozesses muss durch das Betriebssystem explizit unterbunden werden

- Es wird zwischen realer und virtueller Speicherverwaltung unterschieden
  - Welche der beiden Varianten verwendet wird hängt vom Einsatzgebiet des Systems ab

Betriebssysteme
Reale Speicherverwaltung

Der Arbeitsspeicher wird aus den Prozessen heraus direkt adressiert

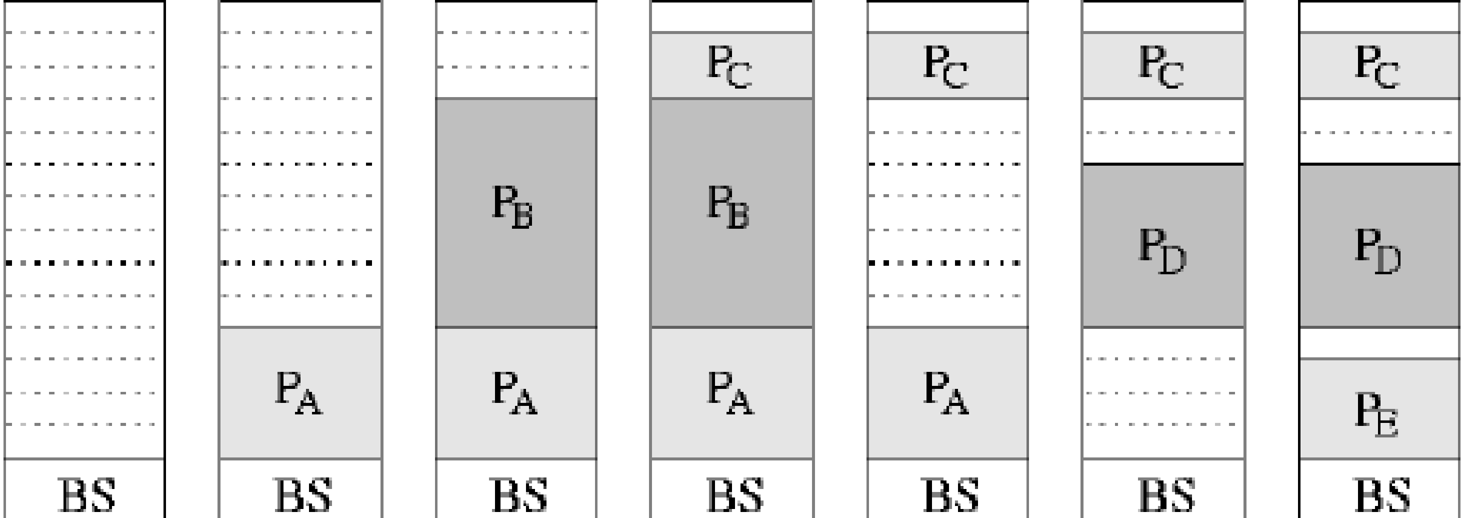
- Die Größe des physikalischen Speichers begrenzt die Anzahl der gleichzeitig ausführbaren Prozessen
  - > Nutzung von Swapping, um mehr Prozesse „parallel“ betreiben zu können

- Probleme realer Speicherverwaltung
  - > Fragmentierung des Speichers
  - > Suche nach freien Speicherblöcken mittels Belegungstabelle sehr aufwändig

Betriebssysteme
Fragmentierung des Speichers

Beim Ersetzen von Speicherblöcken können viele kleine, freie Bereiche im Speicher entstehen

- Neue Prozesse finden keine ausreichend großen, zusammenhängenden Bereiche mehr



Betriebssysteme
Reale Speicherverwaltung

Beim Starten eines Prozesses wird dieser in einen freien Speicherblock geladen

- First Fit
  - > Wählt den ersten, freien Speicherbereich, welcher groß genug ist
  - > Diese Variante ist schnell, verschwendet aber möglicherweise große Blöcke an kleine Prozesse
- Next Fit
  - > Funktioniert wie First Fit, startet aber nicht am Anfang des Speichers, sondern an der Stelle, wo der letzte Prozesse eingefügt wurde
- Best Fit
  - > Wählt den kleinstmöglichen, freien Speicherbereich
  - > Verteilt Speicher gut, aber die Suche nach freien Speicherblöcken ist sehr aufwändig

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Betriebssysteme
Reale Speicherverwaltung

Nachteile reale Speicherverwaltung

- Nutzung des Speicherplatzes
  > Es muss Platz für das gesamte Programm und die Daten gefunden werden, obwohl diese wahrscheinlich nicht alle gleichzeitig benötigt werden

- Beschränkung des Speicherplatzes
  > Es kann insgesamt nicht mehr Speicher genutzt werden, als physikalisch vorhanden ist

- Belegung des Speichers
  > Die Anforderung zusammenhängende Speicherblöcke für Prozesse zu finden, verschärft das Problem der Fragmentierung

Virtuelle Speicherverwaltung behebt die Nachteile realer Speicherverwaltung

- Jedem Prozess wird ein scheinbar zusammenhängender Speicherbereich der Größe n zur Verfügung gestellt
  - Tatsächlich besteht der Speicher des Prozesses aus nicht zwangsläufig zusammenhängenden virtuellen Pages
  - Der Prozess kann seinen Speicher mit den virtuellen Adressen 0 bis $(n-1)$ adressieren
- Die Gesamtheit aller virtuellen Adressen wir als virtueller Adressraum bezeichnet

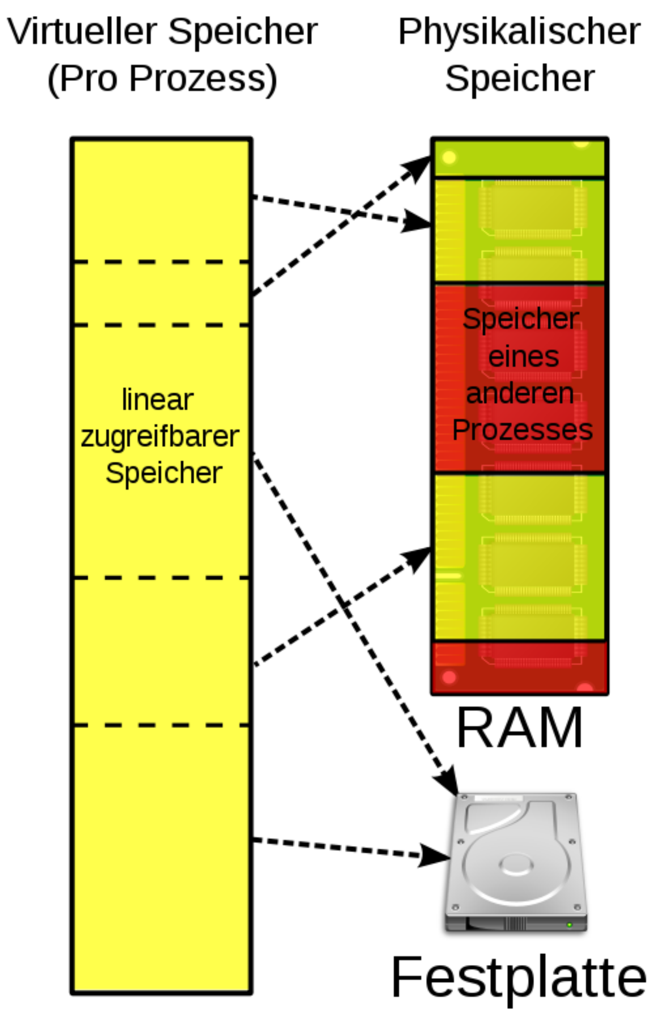
Betriebssysteme
Virtuelle Pages

Virtuelle Pages werden rechnerintern auf physikalisch vorhandene Pages gleicher Größe abgebildet

- Die physikalischen Pages können irgendwo im Arbeitsspeicher oder in einer Auslagerungsdatei auf der Festplatte liegen

- Beim Zugriff eines Prozesses auf eine virtuelle Speicheradresse…
  - … wird zunächst die zu dieser Adresse gehörige virtuelle Page ermittelt
  - Anschließend wird die zu dieser virtuellen Page gehörige physikalische Page ermittelt
    - Die Zuordnung von virtuellen und physikalischen Pages und deren Ablageort werden in der sogenannten Pagetable gespeichert
  - Die relative Speicheradresse innerhalb einer Page ist in virtuellen und physikalischen Pages dieselbe

Betriebssysteme
Virtuelle Pages



Quelle: http://de.wikipedia.org/wiki/Virtuelle_Speicherverwaltung

Berechnung der physikalischen Speicheradresse

n Bit
m Bit
Virtuelle Adresse
Seitennummer
Offset
Basisregister
1-stufige Seitentabelle (max 2^n Zeilen)
Statusbits
Basisadresse
Reale Adresse
Basisadresse der realen Seite
Offset

Quelle: http://de.wikipedia.org/wiki/Paging

Berechnung der physikalischen Speicheradresse:

- Beispiel Adresslänge 16 Bit, 8 Bit Offset (low) und 8 Bit Seitennummer (high)
  - > Seitentabelle:

Eintrag	Gültig	Seitenrahmen
00	Nein	-
01	Ja	0x17
02	Ja	0x20
03	Ja	0x08
04	Nein	-
05	Ja	0x10

  - > Zu übersetzen:

virtuelle Adresse	physikalische Adresse
0x083A	ungültig (Seite 8 ex. nicht)
0x01FF	0x17FF (Seite 1, Rahmen 0x17)
0x0505	0x1005 (Seite 5, Rahmen 0x10)
0x043A	ungültig (Seite 4 ungültig)

Betriebssysteme
Paging on Demand

Zu einem Prozess gehörige, jedoch aktuell nicht genutzte Pages werden auf den Hintergrundspeicher, z.B. Festplatte, ausgelagert

- Es wird Arbeitsspeicher für andere Prozesse freigegeben
- Braucht der Prozess zu einem späteren Zeitpunkt wieder, muss sie erneut in den Arbeitsspeicher geladen werden
  - Es muss eventuell eine andere Page aus dem Arbeitsspeicher verdrängt werden
  - Es gibt mehrere Möglichkeiten zu bestimmen, welche Page aus dem Arbeitsspeicher verdrängt wird

Betriebssysteme
Verdrängungsstrategien

Ähnlich wie beim Caching gibt es auch hier verschiedene Verdrängungsstrategien

- FIFO (First-In, First-Out)
  - > Die Page, welche bereits am längsten im Speicher liegt, wird verdrängt
- Least recently / frequently used (LRU / LFU)
  - > Die Page welche am längst nicht mehr bzw. am seltensten zugegriffen wurde, wird verdrängt
- Unversehrtheit der Speicherseite
  - > Es werden Pages ausgelagert, die sich im Arbeitsspeicher nicht geändert haben, sodass Schreiboperationen auf die Festplatte entfallen

Betriebssysteme
Verdrängungsstrategien

Ähnlich wie beim Caching gibt es auch hier verschiedene Verdrängungsstrategien

- FIFO (First-In, First-Out)
  - Die Page, welche bereits am längsten im Speicher liegt, wird verdrängt
- Least recently / frequently used (LRU / LFU)
  - Die Page welche am längst nicht mehr bzw. am seltensten zugegriffen wurde, wird verdrängt
- Unversehrtheit der Speicherseite
  - Es werden Pages ausgelagert, die sich im Arbeitsspeicher nicht geändert haben, sodass Schreiboperationen auf die Festplatte entfallen

Betriebssysteme
Verdrängungsstrategien

FIFO-Strategie

- Seitenanforderungen: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Arbeitsspeicher	Page 1	1	1	1	4	4	4	5	5	5	5	5	5
	Page 2		2	2	2	1	1	1	1	1	3	3	3
	Page 3			3	3	3	2	2	2	2	2	4	4
Kontrollzustand (wie lange im Speicher in Zyklen)	Page 1	0	1	2	0	1	2	0	1	2	3	4	5
	Page 2	-	0	1	2	0	1	2	3	4	0	1	2
	Page 3	-	-	0	1	2	0	1	2	3	4	0	1

Anzahl Einlagerungen: 9

Betriebssysteme
Verdrängungsstrategien

LRU-Strategie

- Seitenanforderungen: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Arbeitsspeicher	Page 1	1	1	1	4	4	4	5	5	5	3	3	3
	Page 2		2	2	2	1	1	1	1	1	1	4	4
	Page 3			3	3	3	2	2	2	2	2	2	5
Kontrollzustand (wie lange nicht drauf zugegriffen)	Page 1	0	1	2	0	1	2	0	1	2	0	1	2
	Page 2	-	0	1	2	0	1	2	0	1	2	0	1
	Page 3	-	-	0	1	2	0	1	2	0	1	2	0

Anzahl Einlagerungen: 10

„Future"-Strategie

- Seitenanforderungen: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Arbeitsspeicher	Page 1	1	1	1	1	1	1	1	1	1	3	4	4
	Page 2		2	2	2	2	2	2	2	2	2	2	2
	Page 3			3	4	4	4	5	5	5	5	5	5
Kontrollzustand (in wie viel Zyklen wird zugegriffen)	Page 1	4	3	2	1	3	2	1	-	-	-	-	-
	Page 2	-	4	3	2	1	3	2	1	-	-	-	-
	Page 3	-	-	7	7	6	5	5	4	3	2	1	-

Anzahl Einlagerungen: 7

Übung - Strategie: FIFO

- Seitenanforderungen: 1, 4, 2, 5, 3, 2, 1, 4, 2, 5

Referenzfolge		1	4	2	5	3	2	1	4	2	5
Arbeitsspeicher	Page 1										
	Page 2										
	Page 3										
Kontrollzustand (Wie lange schon im Speicher)	Page 1										
	Page 2										
	Page 3										

Anzahl Einlagerungen:

Betriebssysteme
Verdrängungsstrategien

Übung - Strategie: FIFO

- Seitenanforderungen: 1, 4, 2, 5, 3, 2, 1, 4, 2, 5

Referenzfolge		1	4	2	5	3	2	1	4	2	5
Arbeitsspeicher	Page 1	1	1	1	5	5	5	5	4	4	4
	Page 2	-	4	4	4	3	3	3	3	2	2
	Page 3	-	-	2	2	2	2	1	1	1	5
Kontrollzustand (Wie lange schon im Speicher)	Page 1	0	1	2	0	1	2	3	0	1	2
	Page 2	-	0	1	2	0	1	2	3	0	1
	Page 3	-	-	0	1	2	3	0	1	2	0

Anzahl Einlagerungen: 9

Betriebssysteme
Pageflattering

Wenn einem Prozess nicht genügend Pages zur Verfügung stehen, kann es passieren, dass sehr oft Pages nachgeladen / ersetzt werden müssen

- Der Prozess verbringt mehr Zeit mit dem Warten auf den Speicher, als mit der eigentlichen Ausführung

- Ursachen
  - Prozesse ohne Lokalität: Random Access auf große Speicherbereiche
  - Zu viele Prozesse
  - Schlechter Ersetzungsstrategie

- Lösung
  - Zuteilung einer genügend großen Anzahl von Pages
  - Begrenzung der Prozessanzahl
  - Codeoptimierung, sodass der Prozess lokaler arbeitet

Betriebssysteme
Lokale vs. Globale Ersetzung

Es gibt zwei mögliche Varianten Pages zu ersetzen

- Lokale Ersetzung
  - > Ein Prozess ersetzt immer nur seine eigenen Pages
    - > Statische Zuteilung von Pages an Prozesse
    - > Nachladen / Ersetzen von Pages liegt in der Verantwortung der Prozesse
- Globale Ersetzung
  - > Ein Prozess ersetzt auch Pages anderer Prozesse
    - > Dynamisches Verhalten der Prozesse kann berücksichtigt werden
    - > Im Schnitt bessere Effizienz, da ungenutzte Pages von anderen Prozessen verwendet werden können

Virtuelle Speicherverwaltung sorgt implizit dafür, dass mehrere Prozesse nicht gegenseitig auf ihre Speicherbereiche zugreifen können

- Versucht ein Prozess auf eine Adresse zuzugreifen, für welche er selbst keinen Speicher allokiert hat, wird die dazugehörige Page nicht gefunden
  - Es tritt ein Segmentation Fault auf

- Gezielt auf den Speicher anderer Prozesse kann nicht zugegriffen werden, da erst bei der Umsetzung der virtuellen Adresse der physikalische Speicher referenziert wird

Betriebssysteme
Speicherbereinigung

Wird in einem Prozess Speicher dynamisch zur Laufzeit allokiert, muss dieser auch irgendwann wieder freigegeben werden

- Wird der Speicher nicht freigegeben, steht dem Prozess irgendwann kein Speicher mehr zur Verfügung

- Der Prozess kann dies selbst tun
  - z.B. in C/C++ mit free() oder delete()
  - Volle Kontrolle über belegten Speicher, aber kompliziert

- Es kann eine automatische Speicherbereinigung (engl. Garbage Collection) verwendet werden
  - z.B. in JAVA
  - Ist bequem, entzieht dem Programmierer aber die Kontrolle über den Speicher

Vorteile der Speicherbereinigung

- Einige Programmierfehler im Umgang mit Speicher (De-)Allokation können vermieden werden
  - Doppelte Freigabe von Speicher
  - Zugriff auf bereits freigegebene Objekte

- Das Programm läuft möglicherweise schneller, da Speicher gezielt und gesammelt freigegeben werden kann

Nachteile der Speicherbereinigung

- Der Zeitpunkt der Garbage Collection ist nicht deterministisch und hängt von diversen Faktoren ab
  - z.B. Belegung des Speichers
  - Der Algorithmus zur Berechnung des Startzeitpunkts variiert von Plattform zu Plattform (z.B. von C# von JAVA)

- Aufgrund des non-Determinismus ist eine Garbage Collection nicht ohne Weiteres für Echtzeitsysteme geeignet

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

DOS = Disk Operating System

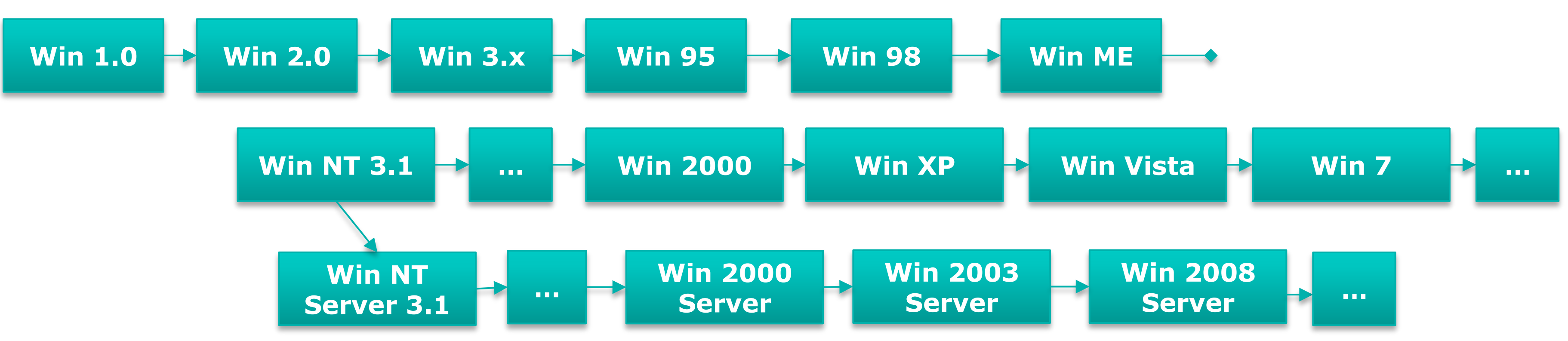
- Reale Speicherverwaltung
- Kooperatives Scheduling
- Kein Multitasking
- Kommandozeilen-basiertes Dialogsystem

```
Starten von MS-DOS...


HIMEM testet den erweiterten Speicher...beendet.



This driver is provided by Oak Technology, Inc..
OTI-91X ATAPI CD-ROM device driver, Rev D91XV352
(C)Copyright Oak Technology Inc. 1987-1997
  Device Name          : CDROM
  Transfer Mode        : Programmed I/O
  Number of drives     : 1


C:\>C:\DOS\SMARTDRV.EXE /X
MSCDEX Version 2.23
Copyright (C) Microsoft Corp. 1986-1993. Alle Rechte vorbehalten.
       Laufwerk D: = Treiber CDROM Gerät 0
C:\>_
```

Betriebssysteme
Windows

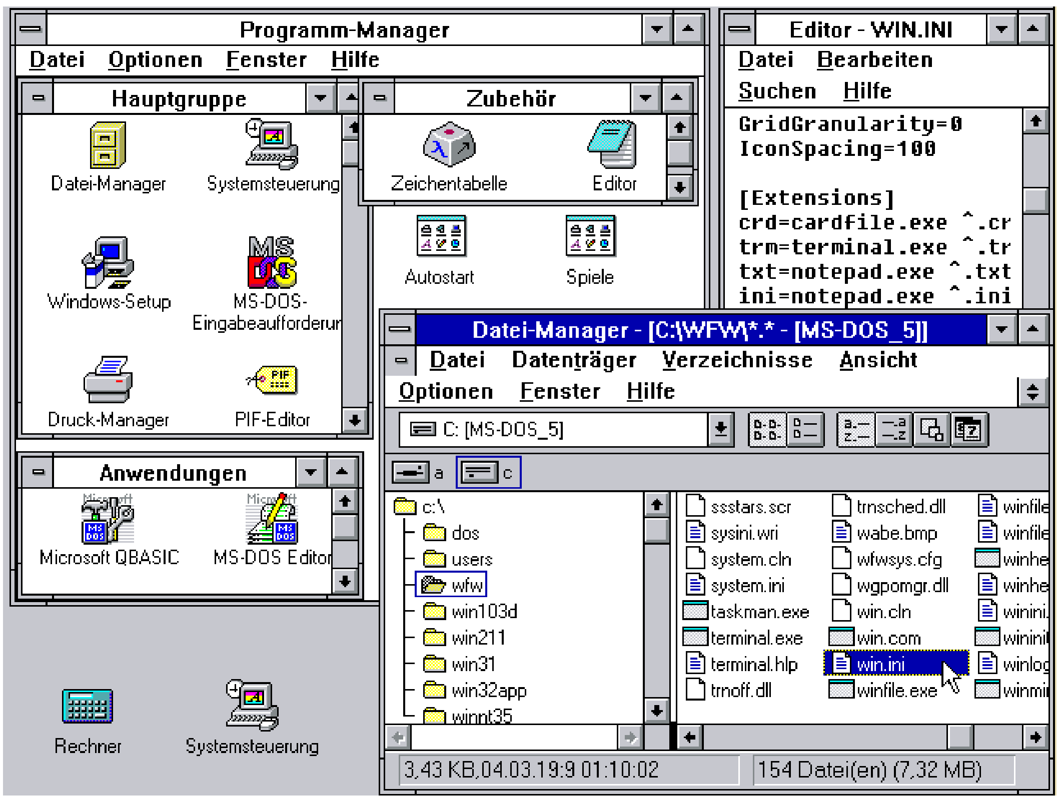
Es gibt nicht DAS Windows, sondern …

- Windows 1.x, 2.x, 3.x — 16 Bit
- Windows 9x — 32 Bit
- Windows {NT 3.x, NT 4.x, 2000, ab XP} — 32 / 64 Bit



DOS-Linie 16 Bit

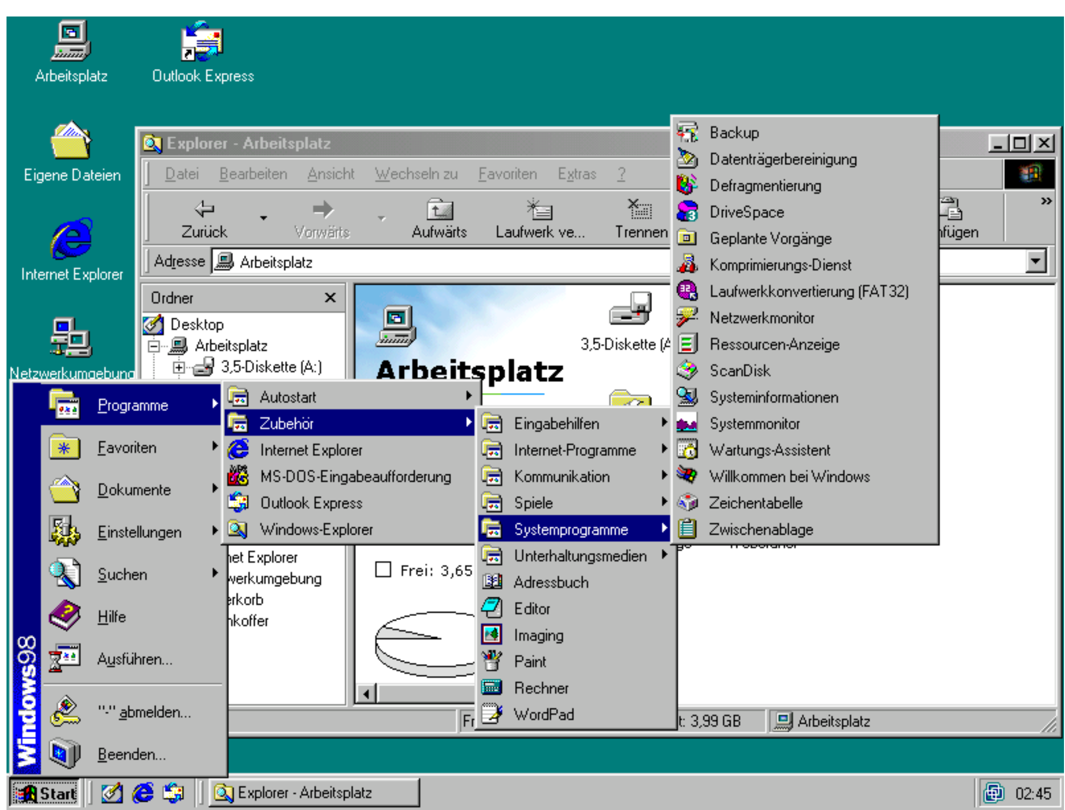
- Reale Speicherverwaltung
- Kooperatives Scheduling
- Multitasking möglich
- Graphisches Dialogsystem



Betriebssysteme
Windows

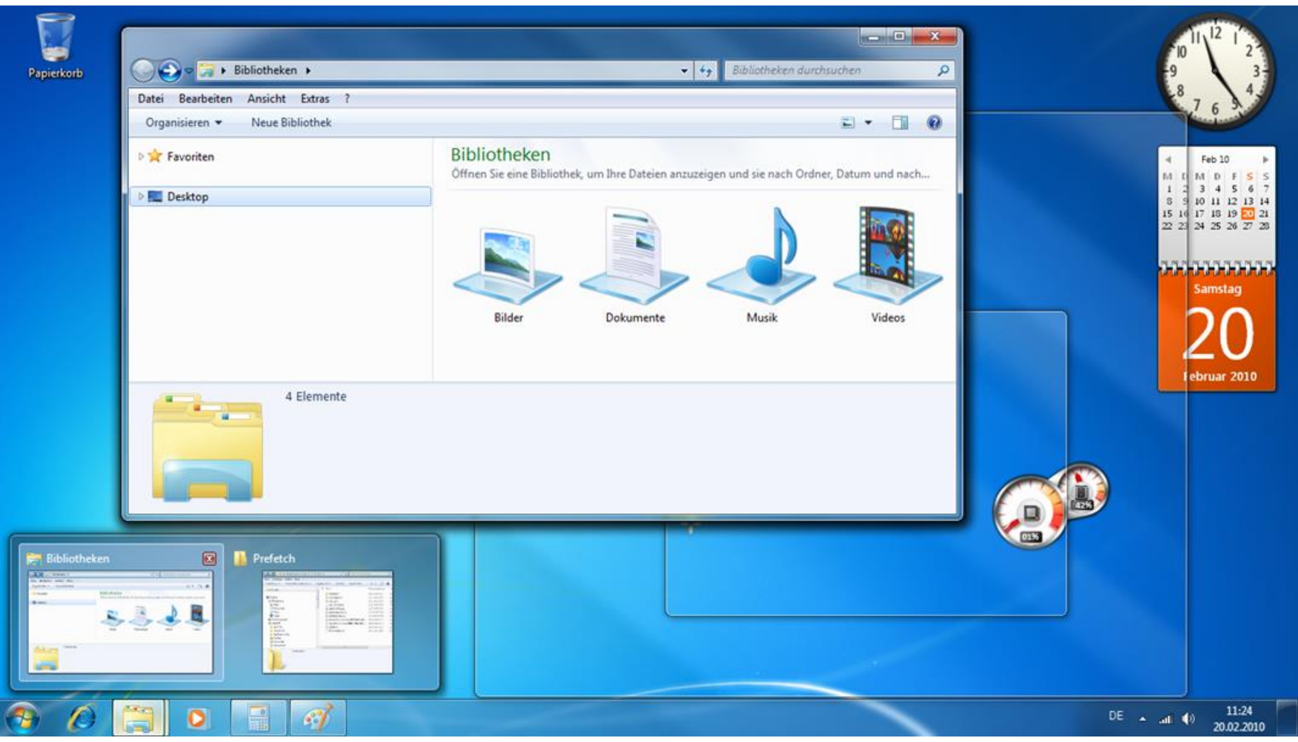
DOS-Linie 32 Bit

- Virtuelle Speicherverwaltung
- Preemptives Scheduling für 32 Bit-Programme, kooperatives Scheduling für 16-Bit Programme
- Multitasking möglich
- Graphisches Dialogsystem



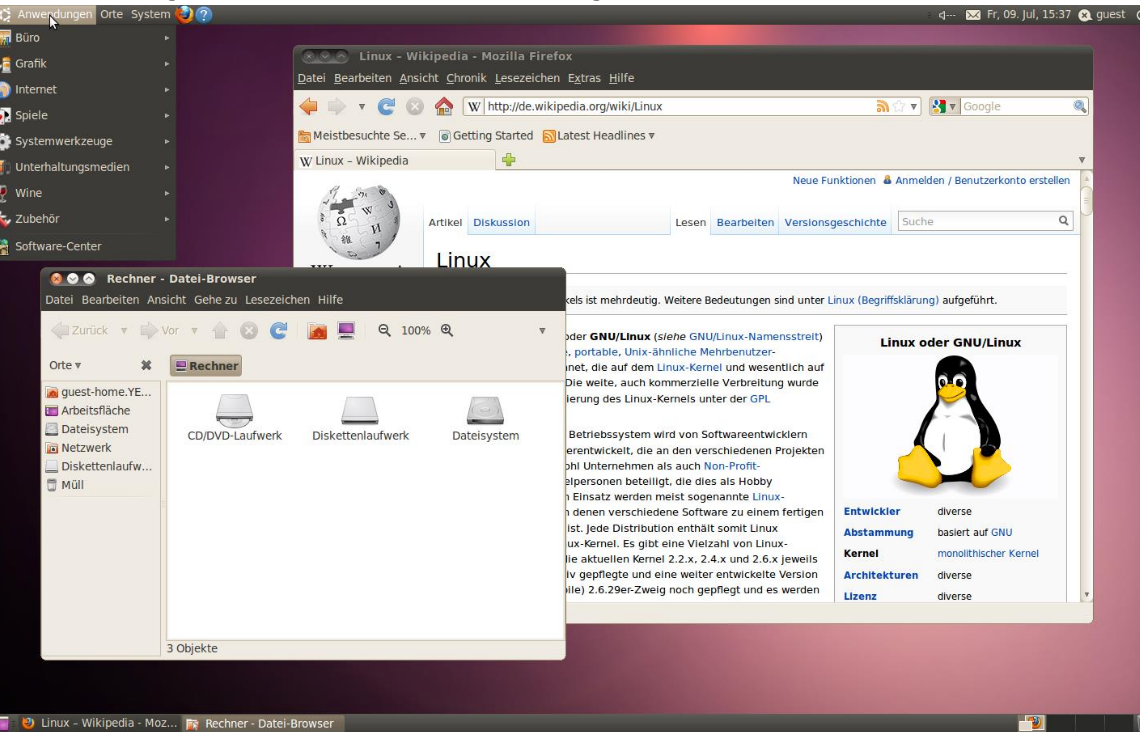
NT-Linie 32/64 Bit

- Virtuelle Speicherverwaltung
- Preemptives Scheduling
- Multitasking möglich
- Graphisches Dialogsystem / Batchsystem (in der Server-Variante)



Linux

- Virtuelle Speicherverwaltung
- Preemptives Scheduling
- Multitasking möglich
- Kommandozeilen-basiertes bzw. graphisches Dialogsystem / Batchsystem



Betriebssysteme
Mac OS X

Mac OS X

- Virtuelle Speicherverwaltung
- Preemptives Scheduling
- Multitasking möglich
- Kommandozeilen-basiertes bzw. graphisches Dialogsystem

9 - Datensicherheit

Kryptographie, Kryptoanalyse, Datenschutz

© **FH AACHEN** UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Lessons Learned
Muss ich mir das alles merken?

In diesem Kapitel geht es darum folgende Dinge zu verstehen und zu können

- Die Begriffe Kryptographie und Kryptoanalyse kennen und voneinander angrenzen können
- Algorithmen, Klassische und moderne Verfahren kennen
- Digitale Signatur kennen

Kryptologie
Allgemeines

Kryptologie setzt sich zusammen aus …

- Kryptographie
  - Beschäftigt sich mit der Verschlüsselung von Informationen
- Kryptoanalyse
  - Beschäftigt sich mit dem Entschlüsseln von Informationen, ohne den passenden Schlüssel zu kennen

Kryptologie
Grundbegriffe

Oft verwendete Begriffe in diesem Themenfeld sind …

- Klartext M
  - Der Klartext ist die unverschlüsselte Nachricht
- Geheimtext / Chiffretext C
  - Der Geheimtext ist die verschlüsselte Nachricht
- Chiffrierung / Dechiffrierung
  - Das Ver- bzw. Entschlüsseln wird Chiffrierung bzw. Dechiffrierung genannt
- Schlüssel
  - Die Geheime Information zum Entschlüsseln ist der Schlüssel

Kryptologie
Kryptographie – Aber wozu?

Kryptographie hat vier Hauptziele

- Sichere Kommunikation über ein unsicheres Medium
  - Netzwerke (speziell Internet), Briefe, …
- Integrität von Nachrichten
  - Ist die vorliegende Nachricht verändert worden?
- Authentifizierung von Kommunikationspartnern
  - Rede ich wirklich mit der richtigen Person?
- Verbindlichkeit
  - Hat der Absender die Nachricht wirklich selbst verschickt?

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Kryptologie
Kryptographie – Rahmenbedingungen

Welche Anforderungen muss Kryptographie erfüllen?

- Kerckhoffs Prinzip (1883)
  - > Das System muss unentzifferbar sein
  - > Das System selbst darf keine Geheimhaltung erfordern
  - > Der Algorithmus muss leicht zu übermitteln sein und ein Mensch muss sich den Schlüssel ohne schriftliche Aufzeichnung merken können
  - > Das System sollte (muss) mit telegraphischer Kommunikation kompatibel sein (Telefon, eMail, etc.)
  - > Das System muss transportabel sein und die Bedienung darf nicht mehr als eine Person erfordern
  - > Das System muss einfach anwendbar sein

Kryptologie
Kryptographie – Algorithmen

- Kryptographie
  - Klassische Verfahren ← Buchstaben
    - Transpositions-chiffren ← Vertauschung von Buchstaben
    - Substitutions-chiffren ← Ersetzung von Buchstaben
  - Moderne Verfahren ← Bits
    - Symmetrische Verfahren ← Gleicher Schlüssel zum Ver- und Entschlüsseln
    - Asymmetrische Verfahren ← Unterschiedliche Schlüssel zum Ver- und Entschlüsseln

Kryptologie
Transpositionschiffren – Skytale

Zur Verschlüsselung wird ein Pergamentstreifen um einen Holzstab gewickelt und beschriftet

- Klartext: Um den Holzstab gewickelter Pergamentstreifen
- Geheimtext: Pergamentstreifen ohne Holzstab
- Schlüssel: Holzstab mit bestimmtem Durchmesser

Quelle: http://de.wikipedia.org/wiki/Skytale

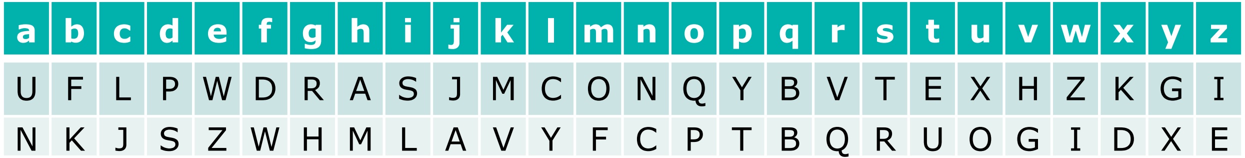
Kryptologie
Substitutionschiffren

Es wird zwischen zwei Arten von Substitutionschiffren unterschieden

- Monoalphabetische Substitutionschiffren
  - > Einem Klartextalphabet wird ein Geheimtextalphabet zugeordnet

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
U	F	L	P	W	D	R	A	S	J	M	C	O	N	Q	Y	B	V	T	E	X	H	Z	K	G	I

- Polyalphabetische Substitutionschiffren
  - > Einem Klartextalphabet werden mehreren Geheimtextalphabete zugeordnet



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
U	F	L	P	W	D	R	A	S	J	M	C	O	N	Q	Y	B	V	T	E	X	H	Z	K	G	I
N	K	J	S	Z	W	H	M	L	A	V	Y	F	C	P	T	B	Q	R	U	O	G	I	D	X	E

Kryptologie
Substitutionschiffren – Cäsar-Chiffre

Ein Beispiel für eine monoalphabetische Substitutionschiffre ist die Cäsar-Chiffre

- Klar- und Geheimtextalphabet sind das lateinische Alphabet
  - Verschiebung der Alphabete zueinander um eine Bestimmte Anzahl von Buchstaben
  - Die Anzahl der Buchstaben, um die verschoben wird, ist der Schlüssel der Cäsar-Chiffre

- Beispiel
  - **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
  - **J K L M N O P Q R S T U V W X Y Z A B C D E F G H I**
  - **HALLO** wird zu **QJUUX**

Kryptologie

Substitutionschiffren – Vigenère-Verschlüsselung

Ein Beispiel für eine polyalphabetische Substitutionschiffre ist die Vigenère-Verschlüsselung

- Ein Schlüsselwort bestimmt, wie viele und welche Geheimtextalphabete benutzt werden
  - Die einzelnen Geheimtextalphabete leiten sich aus der Cäsar-Chiffre ab

- Beispiel
  - Schlüsselwort: **EDV**
  - **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
  - **E F G H I J K L M N O P Q R S T U V W X Y Z A B C D**
  - **D E F G H I J K L M N O P Q R S T U V W X Y Z A B C**
  - **V W X Y Z A B C D E F G H I J K L M N O P Q R S T U**
  - **MATSE** wird zu **QDOWH**

Kryptologie
Moderne Verfahren

Moderne Verfahren unterscheiden zwischen verschiedenen Varianten

- Symmetrische Chiffren
  - Blockchiffren
  - Stromchiffren
- Asymmetrische Chiffren
- Hybride Chiffren

Kryptologie
Symmetrische Chiffren

Bei symmetrischen Chiffren wird zum Ver- und Entschlüsseln derselbe Schlüssel benutzt

- Symmetrische Chiffren unterscheiden zwischen Block- und Stromchiffren
- Vorteile
  - Geringer Rechenaufwand
- Nachteile
  - Geheimer Schlüssel muss über ein unsicheres Medium transportiert werden
- Klassische Verfahren sind prinzipiell auch symmetrische Chiffren
- Bekannte Verfahren sind z.B. DES, 3DES, AES, Blowfish, …

Kryptologie
Symmetrische Chiffren

Alice

Bob

writes

make up key:

reads

Hello Bob, how are you? This is the secret message:
...
...
...

Hello Bob, how are you? This is the secret message:
...
...
...

encrypts

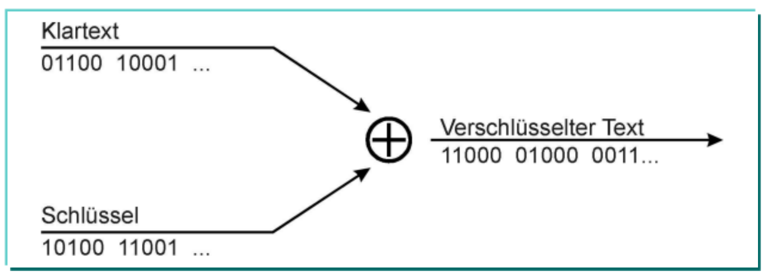
decrypts

insecure channel

Symmetrische Chiffren unterscheiden zwischen …

- …Blockchiffren
  - > Der Klartext wird in Blöcke gleicher Länge eingeteilt und blockweise verschlüsselt
  - > Es wird derselbe Schlüssel für alle Klartextblöcke verwendet
  - > Der letzte Klartextblock wird mit Nullen aufgefüllt

- … Stromchiffren
  - > Der Klartext wird bitweise anhand eines Schlüsselstroms verschlüsselt
  - > Der Schlüsselstrom und der Geheimtext haben dieselbe Länge wie der Klartext



© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Kryptologie
Asymmetrische Chiffren

Bei asymmetrischen Chiffren werden unterschiedliche Schlüssel zum Ver- und Entschlüsseln benutzt

- Zwei Schlüssel bilden ein Schlüsselpaar, bestehend aus…
  - Privater Schlüssel („private key“) zum Entschlüsseln
  - Öffentlicher Schlüssel („public key“) zum Verschlüsseln
- Vorteile
  - Privater Schlüssel muss nicht übertragen werden
- Nachteile
  - Rechenaufwand deutlich höher als bei symmetrischen Chiffren (~ Faktor 1000)

Kryptologie
Asymmetrische Chiffren

Alice

writes

Hello Bob, how are you? This is the secret message:
...
...
...

encrypts

provides

public

insecure channel

Bob

reads

Hello Bob, how are you? This is the secret message:
...
...
...

private

decrypts

Kryptologie
Asymmetrische Chiffren

Das meist genutzte asymmetrische Verfahren ist RSA

- RSA ...
  - ... wurde 1977 am MIT von R. Rivest, A. Shamir und L. Adleman entwickelt
  - ... basiert auf Primfaktorzerlegung
- Der RSA-Algorithmus funktioniert wie folgt
  - Verschlüsselung: $C = M^e \bmod N$
  - Entschlüsselung: $M = C^d \bmod N$
- Die entsprechenden Schlüsselpaare haben folgende Gestalt
  - Privater Schlüssel: (d, N)
  - Öffentlicher Schlüssel: (e, N)

Ein Beispiel zu RSA

- Gegeben
  - Algorithmen: $C = M^e \bmod N, \; M = C^d \bmod N$
  - Privater Schlüssel: $(d, N) = (47, 143)$
  - Öffentlicher Schlüssel: $(e, N) = (23, 143)$
  - Klartext: $M = 97$ (a)

- Berechnen des Geheimtextes
  - $C = M^e \bmod N = 97^{23} \bmod 143 = 102$

- Berechnen des Klartextes
  - $M = C^d \bmod N = 102^{47} \bmod 143 = 97$

© FH AACHEN UNIVERSITY OF APPLIED SCIENCES, FORSCHUNGSZENTRUM JÜLICH, RWTH AACHEN UNIVERSITY

Kryptologie
Digitale Signatur

Eine digitale Signatur dient zur Sicherung der …

- … Identität des Autors einer Nachricht
- … Integrität der Nachricht selbst
- Anforderungen an eine digitale Signatur
  - Es darf nur dem echten Sender einer Nachricht möglich sein, eine digitale Signatur zu erzeugen
  - Die Signatur muss vom Empfänger überprüft werden können
  - Eine Signatur muss fest zu einer Nachricht gehören und darf nur in Verbindung mit dieser Nachricht gültig sein

Kryptologie
Exkurs: Hashfunktion

Eine Hashfunktion f(x) hat folgende Eigenschaften

- f(x) bildet eine beliebige, aber wohldefinierte, Eingabe auf einen Wert fester Länge (= Hashwert) ab
- f(x) sollte injektiv sein, ist es aber meist nicht
- f(x) ist nicht umkehrbar (speziell für kryptologische Hashfunktionen)
- Beispiel: MD5 (md5() in PHP)
  - **MATSE** → *0241739d4c2596e4f12a2fd531de6163*

Kryptologie
Digitale Signatur

Wie funktioniert eine digitale Signatur?

- Signieren
  1. Berechnung einer Prüfsumme der Nachricht mittels einer Hashfunktion
  2. Verschlüsselung der Prüfsumme mit dem privaten Schlüssel

- Prüfen der Signatur
  1. Entschlüsselung der Nachricht mit dem öffentlichen Schlüssel des Senders (= Identität des Autors)
  2. Eigene Berechnung einer Prüfsumme über die Nachricht
  3. Vergleich der selbst berechneten Prüfsumme mit der entschlüsselten Prüfsumme (= Integrität der Nachricht)

Kryptologie
Hybride Chiffren

Die Kombination von symmetrischen und asymmetrischen Chiffren nennt man hybride Chiffren

- Der verhältnismäßig kurze Schlüssel einer Blockchiffre wird asymmetrisch verschlüsselt und zum Empfänge einer Nachricht transportiert

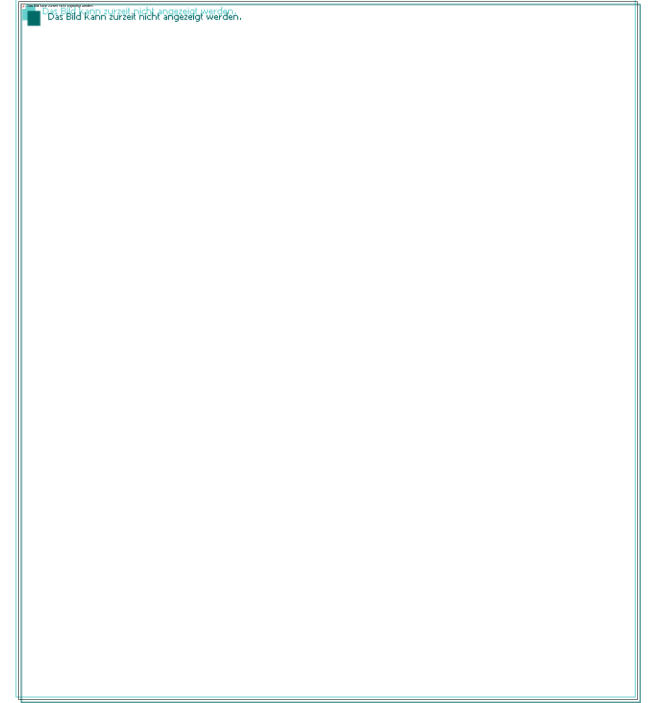
- Die Kommunikation selbst läuft symmetrisch verschlüsselt ab

- Durch Kombination der beiden Verfahren wird ein guter Kompromiss zwischen Sicherheit und Rechenaufwand erzielt

- Anwendungsgebiete
  - HTTPS (SSL im Browser)
  - E-Mailverschlüsselung
  - …

Wozu Kryptoanalyse?

- Oft krimineller Hintergrund
  - z.B. Auslesen von Kreditkartennummern bei Online-Einkäufen
- Bekannte Beispiele für Kryptoanalyse
  - Die Enigma im 2. Weltkrieg
  - Der „NSA-Skandal"



Es gibt mehre Arten eine Kryptoanalyse durchzuführen

- Brute-Force
  - > Ausprobieren aller möglichen Schlüssel
  - > Ergebnis pro Schlüssel muss jeweils inhaltlich überprüft werden
- Wörterbuchangriff
  - > Geht davon aus, dass der Schlüssel ein existierendes Wort ist
  - > Schneller als Brute-Force, jedoch wirkungslos wenn ein zufälliger Schlüssel verwendet wird
- Häufigkeitsanalyse
  - > Nur bei klassischen Chiffren
  - > Die Verteilung der Buchstaben in einer Sprache wird ausgenutzt
- „Fortgeschrittene" Methoden

Datenschutz
Sicherheitsrisiken

Daten können nicht nur in falsche Hände gelangen, sondern auch verloren gehen

- Daten können…
  - > … gestohlen werden
  - > … verbrennen
  - > … gelöscht werden
  - > … durch Hardwaredefekte verloren gehen
  - > … durch Malware / Viren unbrauchbar gemacht werden
  - > …

Datenschutz
Sicherheitsrisiken

Daher hat Datenschutz nicht nur etwas mit Kryptographie zu tun, sondern auch mit anderen Aspekten

- Gebäude
  - Bauliche und organisatorische Maßnahmen gegen höhere Gewalt und Sabotage
- Geräte
  - Geräte redundant betreiben
- Programme und Daten
  - Backup der Daten erstellen
- Dienste
  - Maßnahmen gegen Missbrauch und Attacken

Mit zunehmender Digitalisierung wird der korrekte Umgang mit personenbezogenen Daten immer wichtiger

- Jeder hat das Recht auf „informationelle Selbstbestimmung“
  - Soll heißen: Jeder darf selbst entscheiden, welche Informationen über sie / ihn öffentlich zugänglich sind

- Es werden jedoch zunehmend Daten erfasst und gesammelt
  - Im öffentlichen Raum
    - Verbrechensaufklärung, Verbrechensprävention, …
  - Im privaten Raum
    - Kontrolle von Mitarbeitern, Erfassung von Kundenprofilen, …

Datenschutz
Personenbezogene Daten

Gesetzlicher Rahmen stellt sicher, dass…

- … nur relevante Daten verwendet werden
- … ein Betroffener Auskunft über verwendete Daten erhalten kann
- … ein Betroffener Anspruch auf Korrektur von falschen Daten hat
- … eine Weitergabe der Daten nur unter bestimmten Bedingungen möglich ist

Datenschutz
Personenbezogene Daten

Stellen, die mit sensiblen Daten umgehen (Ämter, Firmen, …) müssen sicherstellen, dass verarbeitete Daten „sicher“ sind, durch …

- … bauliche Maßnahmen (z.B. Zutrittsbeschränkung)
- … Benutzer- und Rechteverwaltung in der IT
- … Kryptographie