https://matse.paddel.xyz/spicker

IT-Grundlagen

Patrick Gustav Blaneck

Letzte Änderung: 13. Juni 2021

In dieser Zusammenfassung werden Inhalte aus dem ITG-Skript von Bastian Küppers verwendet.

Inhaltsverzeichnis

1 Codierung

1.1 Stellenwertsysteme

Definition: Stellenwertsystem

Allgemein lässt sich der Wert einer Zahl in einem Stellenwertsystem zur Basis B wie folgt ausdrücken (d_i ist der Wert der i-ten Stelle, r der Exponent der höchtswertigen Stelle):

$$n_b = \sum_{i=0}^{r} B^i \cdot d_i$$

Algorithmus: Dezimal $\rightarrow$ Binär, Hexadezimal, ...

Sei $B = 2$ für Umrechnung ins Binärsystem, bzw. $B = 16$ für das Hexadezimalsystem.

Es gilt für eine umzurechnende Zahl z:

$$\begin{aligned}
z : B &= z_0 && \text{Rest } r_0 \\
z_0 : B &= z_1 && \text{Rest } r_1 \\
z_1 : B &= z_2 && \text{Rest } r_2 \\
&\dots \\
z_{n-2} : B &= z_{n-1} && \text{Rest } r_{n-1} \\
z_{n-1} : B &= 0 && \text{Rest } r_n
\end{aligned}$$

Damit gilt dann: $(z)_{10} = (r_n r_{n-1} \dots r_2 r_1 r_0)_B$ (also gelesen von *unten nach oben*). □

Algorithmus: Binär $\rightarrow$ Hexadezimal

Sei eine Binärzahl b gegeben mit $n \in 4\mathbb{N}$ Bits.

Dann kann b wie folgt in eine Hexadezimalzahl h mit $m = \frac{n}{4}$ Zeichen umgeformt werden:

$$\underbrace{b_{n-1}b_{n-2}b_{n-3}b_{n-4}}_{h_{m-1}} \dots \underbrace{b_7 b_6 b_5 b_4}_{h_1} \underbrace{b_3 b_2 b_1 b_0}_{h_0}$$

Erinnerung: 4 Bits können binär nur 16 mögliche Werte annehmen!

1.2 Zahlendarstellungen

Definition: Einerkomplement

Das *Einerkomplement* einer Binärzahl wird gebildet, indem man alle Bits negiert.

Das erste Bit gibt dabei das Vorzeichen an.

Nachteile: Doppelte Darstellung der Null, Subtraktion lässt sich nicht auf Addition mit einer negativen Zahl zurückführen

Definition: Zweierkomplement

Das *Zweierkomplement* einer Binärzahl wird gebildet, indem man das Einerkomplement bildet und zusätzlich 1 addiert.

Das erste Bit gibt dabei das Vorzeichen an.

Vorteile: Subtraktion entspricht Addition mit einer negativen Zahl, keine doppelte Null

Definition: Binäre Festkommazahlen

Die Umwandlung des ganzzahligen Anteils einer (dezimalen) Festkommazahl erfolgt analog zu ganzen Zahlen.

Zusätzlich werden aber die Nachkommastellen mit entsprechend negativen Exponenten weiter „verrechnet“.

Definition: Gleitkommazahlen nach *IEEE 754*

Die Darstellung einer Gleitkommazahl

$$x = s \cdot m \cdot b^e$$

besteht aus

- Vorzeichen s (1 Bit)
- Mantisse m (p Bits, $p_{\text{float}} = 23$, $p_{\text{double}} = 52$)
- Basis b (bei normalisierten Gleitkommazahlen nach IEEE ist $b = 2$)
- Exponent e (r Bits, $r_{\text{float}} = 8$, $r_{\text{double}} = 11$)

Algorithmus: Umrechnung in *IEEE 754*

- Umwandeln einer Dezimalzahl in eine *binäre Festkommazahl* ohne Vorzeichen.
- Normalisieren der *Mantisse*:
  Das Komma wird n Stellen nach links verschoben, so dass dort nur noch eine 1 als ganzzahliger Anteil vorhanden ist. (n ist bei Verschieben nach rechts negativ!)
  M ist dann der *Nachkommateil*.
- Bestimmen des *Exponenten*: $E = (\text{bias } + n)_2$ (bias ist meist 127, wenn nicht anders gegeben)
- Bestimmen des *Vorzeichenbits* s
- Zusammensetzen der Gleitkommazahl

$$s \mid E \mid M$$

1.3 Zeichendarstellungen

Definition: UTF-8 Codierung

Unicode-Bereich	UTF-8-Codierung	Möglichkeiten
0000 0000 - 0000 007F	0xxx xxxx	128 (7 Bits)
0000 0080 - 0000 07FF	110x xxxx 10xx xxxx	2.048 (11 Bits)
0000 0800 - 0000 FFFF	1110 xxxx 10xx xxxx 10xx xxxx	65.536 (16 Bits)
0001 0000 - 0010 FFFF	1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx	2.097.152 (21 Bits)

1.4 Spezielle Codierungen

Algorithmus: Huffman-Codierung

1. Schreibe Buchstaben mit Auftrittshäufigkeiten als „Wald“.
2. Fasse die beiden Bäume mit der geringsten Auftrittshäufigkeiten zu einem neuen Baum zusammen, dabei werden die Auftrittshäufigkeiten addiert.
3. Wiederhole, bis nur noch ein Baum existiert.
4. Codierung eines Buchstaben entspricht dann dem *Pfad* zum entsprechenden Blatt mit „links“ $\rightarrow 0$, „rechts“ $\rightarrow 1$

Mittlere Codelänge:

$$L(C) = \frac{\text{\#Bits der verschlüsselten Nachricht}}{\text{\#Zeichen der verschlüsselten Nachricht}}$$

Definition: Hamming-Codierung

Bei der Hamming-Codierung werden Paritätsinformationen zu Daten hinzugefügt, um so mögliche Übertragungsfehler zu erkennen.

Hamming-Codewörter haben die Länge $N = 2^k - 1$, wobei k Paritätsbits enthalten sind. Die Bits werden der Einfachheit halber bei Eins beginnend durchnummeriert. Die Paritätsbits stehen an den Stellen, deren Index eine 2er-Potenz ist.

Sind $p_1, p_2, \ldots, p_k$ Paritätsbits, $d_1, d_2, \ldots, d_{N-k}$ Bits des Datenwortes und $c_1, c_2, \ldots, c_N$ die Bits des zu bildenden Codewortes, hat ein Codewort des so konstruierten Hamming-Codes die folgende Form:

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	$\ldots$
p_0	p_1	d_1	p_2	d_2	d_3	d_4	p_3	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	p_4	d_{12}	$\ldots$

Dabei wird jedem Paritätsbit eine spezielle Bitmaske zugewiesen:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	$\ldots$
Bitmaske p_0	p_0	p_1	1	p_2	1	0	1	p_3	1	0	1	0	1	0	1	$\ldots$
Bitmaske p_1	p_0	p_1	1	p_2	0	1	1	p_3	0	1	1	0	0	1	1	$\ldots$
Bitmaske p_2	p_0	p_1	0	p_2	1	1	1	p_3	0	0	0	1	1	1	1	$\ldots$
Bitmaske p_3	p_0	p_1	0	p_2	0	0	0	p_3	1	1	1	1	1	1	1	$\ldots$

Damit gilt:

$$c_1 = p_0 = c_3 \oplus c_5 \oplus c_7 \oplus c_9 \oplus c_{11} \oplus c_{13} \oplus c_{15} \oplus \ldots$$
$$\iff \text{jedes ungerade Datenbit}$$
$$c_2 = p_1 = c_3 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{11} \oplus c_{14} \oplus c_{15} \oplus \ldots$$
$$\iff \text{ein Datenbit rechts von } p_1\text{, zwei überspringen, zwei einberechnen, } \ldots$$
$$c_4 = p_2 = c_5 \oplus c_6 \oplus c_7 \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15} \oplus \ldots$$
$$\iff \text{drei Datenbit rechts von } p_2\text{, vier überspringen, vier einberechnen, } \ldots$$

Algorithmus: Fehlererkennung beim Hamming-Code

Situation: Empfangen eines Hamming-codierten Datensatzes

1. Erneutes Berechnen der *Paritätsbits*
2. Erkennen, welche neu berechneten Paritätsbits p_i' *verschieden* sind zu den empfangenen Paritätsbits p_i
3. Bitfehler ist in dem Datenbit passiert, in dem gilt:
   $p_i = p_i' \implies$ Bitmaske für p_i ist 0
   $p_i \neq p_i' \implies$ Bitmaske für p_i ist 1

Es wird angenommen, dass nur ein Bitfehler in den Datenbits passiert ist.

Anmerkung: Ist lediglich ein Paritätsbit verschieden, dann ist ein Übertragungsfehler in dem betreffenden Paritätsbit selbst aufgetreten, da alle Datenbits zur Berechnung von mindestens zwei Paritätsbits verwendet werden.

2 Formale Sprachen

2.1 Backus-Naur-Form

Definition: (kontextfreie) Grammatik

Eine (kontextfreie) Grammatik G ist ein 4-Tupel $\{N, T, \Sigma, P\}$ mit folgenden Eigenschaften:

- N ist eine endliche Menge von *Nichtterminalsymbolen*,
- T ist eine endliche Menge von *Terminalsymbolen*,
- ein *Startsymbol* $\Sigma \in N$,
- eine endliche Menge an Produktionsregeln $P \subset N \times T^*$.

Es gilt $N \cap M = \emptyset$. * bezeichnet die Kleensche Hülle.

Anmerkung: Die Notation verhält sich in der Vorlesung sehr anders als in den meisten Quellen.

Definition: Backus-Naur-Form

In der *Backus-Naur-Form* werden Symbole durch eine Zuweisung definiert.
Diese wird durch den Operator `:=` dargestellt.

- Dabei gilt weiterhin, dass in der Zuweisung eines *Nichtterminalsymbols* mindestens ein weiteres Symbol auftauchen muss und
- bei der Zuweisung eines *Terminalsymbols* nur Zeichen des Alphabets.
- Bei einer Zuweisung können mehrere Symbole oder Zeichen(-ketten) des Alphabets verbunden werden.
- Durch ein Leerzeichen („ “, `space`) wird eine `und`-Verknüpfung definiert,
- durch einen senkrechten Stricht („ | “, `pipe`) wird eine `oder`-Zuweisung definiert.
- Zur logischen Gliederung der Verknüpfungen können Klammern verwendet werden.
- Es ist ebenfalls möglich Kardinalitäten für Symbole zu definieren:
  Geschweifte Klammern {} definieren dabei eine *beliebige Wiederholung* und
  eckige Klammern `[]` definieren ein *einmaliges Auftreten*.
  In beiden Fällen ist das Weglassen des Symbols ebenfalls möglich.

Beispiel: Backus-Naur-Form

```
        Satz  := Subjekt Verb Präposition Nomen Ende
     Subjekt  := Nomen
       Nomen  := Artikel Substantiv
     Artikel  := "der"| "die"| "das"
  Substantiv  := "Mann"| "Frau"| "Haus"
       Verb   := "geht"
 Präposition  := "in"
       Ende   := "."
```

2.2 Programmiersprachen

Definition: Compiler

Das Programm, welches den menschenlesbaren *Quellcode* anhand der Regeln der zugrundeliegenden Grammatik interpretiert und in maschinenlesbare Anweisungen überführt, wird *Compiler* genannt.

Die maschinenlesbaren Anweisungen können dabei entweder direkt in eine maschinenspezifische Binärfolge umgewandelt werden, oder in einen sogenannten *Bytecode*.

Im zweiten Fall kann der Bytecode, auch Zwischencode genannt, noch an beliebigen Prozessortypen angepasst werden, muss dafür aber vor der Ausführung erneut bearbeitet werden.

Bei der Kompilierung eines Programms werden zwei Phasen durchlaufen:

1. **Analysephase**, in der der Quellcode analysiert und auf Fehler geprüft wird:
   Lexer (lexikalischer Scanner):
   Quellcode wird in einzelne Teile (*Tokens*) zerlegt und klassifiziert (z.B. als *Schlüsselwörter* oder *Bezeichner*).
   Der Lexer erkennt hier z.B. falsch benannte Variablen.
   Parser (syntaktische Analyse):
   Vom Lexer erzeugte *Tokens* werden dem *Parser* übergeben. Der Parser überprüft den Quellcode auf Fehler und setzt ihn bei Fehlerfreiheit in einen *Syntaxbaum* (AST) um.
   Hier werden Fehler wie fehlende Semikolons oder falsche genutzte Operatoren erkannt.
   Semantische Analyse:
   Hier wird z.B. kontrolliert, ob eine verwendete Variable auch vorher deklariert wurde, oder ob Quell- und Zieltyp einer Anweisung übereinstimmen. Dabei erzeugte Metadaten werden in den AST integriert. In diesem Schritt werden generell semantische Fehler erkannt, d.h. Anweisungen, die korrekt aussehen, aber fehlerhaft sind.

2. **Synthesephase**, in der Binär- bzw. Bytecode erzeugt wird:
   Zwischencodeerzeugung:
   Hier wird plattformunabhängiger Bytecode erzeugt, der als Grundlage für den nächsten Schritt dient.
   Optimierung:
   Während der Optimierung wird versucht die Performanz des erzeugten Programms zu steigern.
   Codegenerierung:
   Hier wird aus dem optimierten Zwischencode der *Binärcode* erzeugt.

Bonus: Ahead-of-Time Compiler

Bei dieser Variante wird der gesamte Quellcode *in einem Rutsch* in Binär- oder Bytecode übersetzt. Das kann mitunter dazu führen, dass die Kompilierung sehr lange dauert.

Bonus: Just-in-Time Compiler

Daher haben sich mit der Zeit auch sogenannte *Just-in- Time Compiler* etablieren können, die nur einen kleinen Teil des Quellcodes direkt übersetzen und weitere Teile des Quellcodes erst dann übersetzen, *wenn sie bei der Programmausführung benötigt werden*.

Definition: Linker

Nachdem der Compiler den Binärcode erzeugt hat, muss der *Linker* daraus noch ein lauffähiges Programm erstellen. Das ist insbesondere der Fall, wenn es mehrere Dateien mit Quellcode gibt.

Dies geschieht indem der Binärcode zu den verschiedenen Quellcodedateien, der nun in sogenannten *Objektdateien* vorliegt, zusammengefügt wird. Dabei werden beispielsweise symbolische Adressen aus mehreren Quellcodemodulen und externen Bibliotheken so angepasst, dass sie zusammen passen.

Dabei wird zwischen *statischem Linken* und *dynamischen Linken* unterschieden:

Beim **statischen Linken** werden alle verfügbaren Objektdateien zu *einer einzigen, ausführbaren Programmdatei* gelinkt.

- Vorteil: keine externen Abhängigkeiten, Programm auf jedem geeigneten System ohne Weiteres lauffähig
- Nachteil: nicht mehr möglich, einzelne Programmteile auszutauschen ohne den Linkvorgang vollständig zu wiederholen,

Beim **dynamischen Linken** werden Funktions- und Variablenadressen erst zur Laufzeit aufgelöst, sodass externe Bibliotheken einfach in Form von *Dynamically Linked Libraries* (DLLs) bzw. *Shared Objects* (SOs) angesprochen werden können. Da DLLs bzw. SOs als existierend vorausgesetzt werden, wird das fertige Programm bei dynamischem Linken kleiner.

- Vorteil: mehrere Programme können dieselbe externe Bibliothek verwenden, ohne dass der benötigte Code mehrfach in einzelne Programme integriert werden muss.

Definition: Interpreter

Interpreter zeichnen sich dadurch aus, dass der Quellcode nicht einmalig in Binärcode übersetzt wird, sondern bei jeder Programmausführung schrittweise abgearbeitet wird.

- Vorteile: einfache Portabilität, dynamischere Quellcodeverwaltung
- Nachteile: deutlich langsamere Ausführungsgeschwindigkeit, keine Optimierungen an Programmstruktur möglich

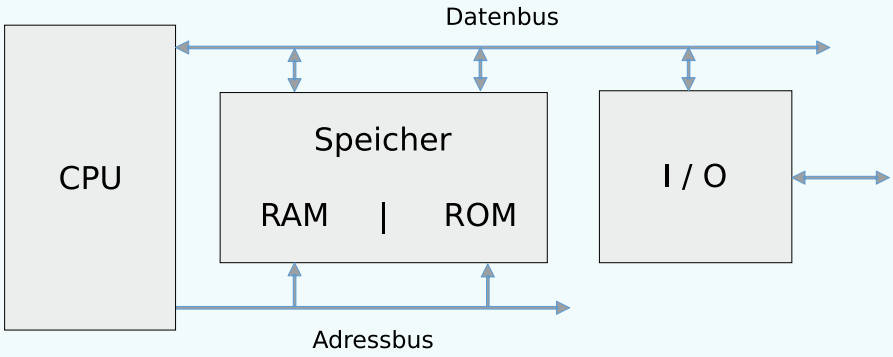
3 Rechnerarchitekturen

3.1 Von-Neumann-Architektur

Definition: Von-Neumann-Architektur

Im Grundsatz besteht die Architektur aus drei Komponenten: *CPU*, *Speicher* und *I/O Einheit*.

Die einzelnen Komponenten sind über Datenleitungen, sogenannte *Busse*, verbunden.



Definition: CPU

Die *CPU* (*Central Processing Unit*, Prozessor) ist sozusagen das Gehirn des Computers. Die CPU besteht im Wesentlichen aus drei Teilen, dem *Leitwerk* und dem *Rechenwerk* und den *Registern*, welche direkt zur Abarbeitung von Befehlen benötigte Daten und berechnete Ergebnisse aufnehmen können.

Das *Leitwerk* steuert die Ausführung des Binärcodes, das *Rechenwerk* führt anfallende Rechenoperationen aus.

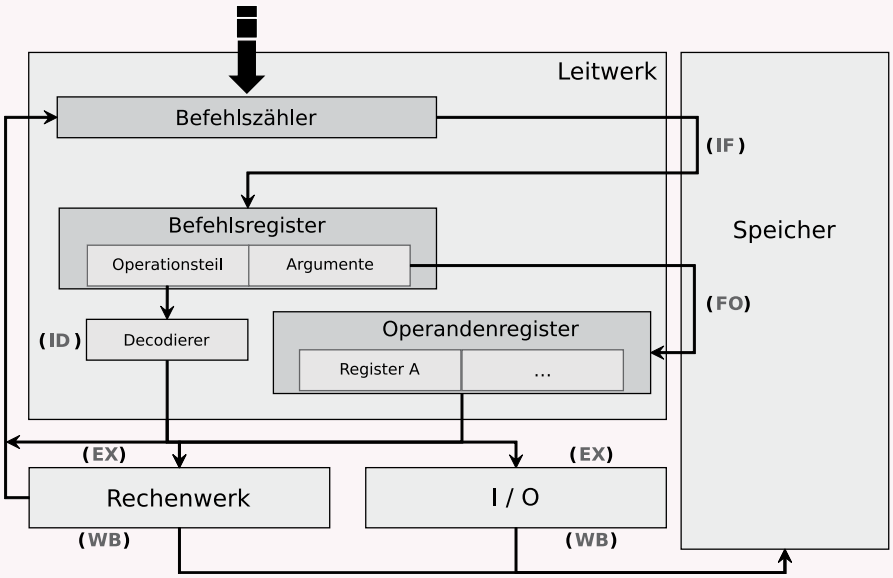
Bonus: Aufbau Rechenoperationen

- *Operationsteil*: codiert konkreten Befehl
- *Operanden*: stellen z.B. Summanden einer Operation, oder Adresse einer Variablen dar

Bonus: Abarbeitung von Befehlen

Jeder Befehl durchläuft bei der Abarbeitung in der CPU folgende Schritte:

- *Instruction Fetch (IF)* : Befehl lesen
- *Instruction Decode (ID)* : Befehl decodieren
- *Fetch Operands (FO)* : Operanden laden
- *Execute (EX)* : Befehl ausführen
- *Writeback (WB)* : Ergebnis schreiben



Definition: Prozessorarchitekturen

CISC (*Complex Instruction Set Computer*):
Eine CISC CPU zeichnet sich durch einen *komplexen Befehlssatz* in Form von *Microcode* und dem Vorhandensein nur *weniger Register* aus.

RISC (*Reduced Instruction Set Computer*):
eine RISC CPU nur über *wenige, in Hardware realisierte Befehle* und *viele Prozessorregister*.

Definition: Pipelining

In einer *RISC CPU*, die nur wenige und elementare Befehle verwendet, kann dafür gesorgt werden, dass alle Teilschritte, deren *parallele Verarbeitung* das Pipelining ermöglicht, gleich lange dauern. Nur deswegen kann das Konzept des Pipelinings erfolgreich umgesetzt werden.

Das ist bei einer *CISC CPU* aufgrund der vielen und teils sehr komplexen Befehle *nicht* möglich.

Beispiel: Pipelining

5-Stage-Pipeline:

- Instruction Fetch (**IF**): Befehl lesen
- Instruction Decode (**ID**) : Befehl decodieren
- Execute (**EX**) : Ausführen
- Memory Access (**MEM**) : Ausführen
- Writeback (**WB**) : Ergebnis schreiben

Instr. 1	IF	ID	EX	MEM	WB	
Instr. 2		IF	ID	EX	MEM	WB
Instr. 3			IF	ID	EX	MEM
Instr. 4				IF	ID	EX
Instr. 5					IF	ID
Clock Cycle	1	2	3	4	5	6

Definition: ROM

Der *ROM* ist ein Festwertspeicher, der - prinzipiell - *nur gelesen* werden kann und die Firmware des Computers gespeichert hat.

Im ROM liegt das sogenannte *BIOS* (*basic input output system*) bzw. moderner *UEFI* (*unified extensible firmware interface*). Die im ROM abgelegten Informationen stellen die Firmware des Rechners dar. Diese Firmware sorgt dafür, dass der Computer nach dem Einschalten in die Lage versetzt wird, grundlegende Hardwarekomponenten zu verwalten.

Definition: RAM

Der *RAM*, auch *Hauptspeicher* genannt, ist ein Speicher mit *wahlfreiem Zugriff*, der seinen Inhalt jedoch bei Verlust der Betriebsspannung *verliert*. Im RAM werden Informationen abgelegt, die ein *Programm zur Laufzeit* benötigt.

Sollen Daten über das Ende des Programms hinaus gespeichert werden, müssen sie über die *I/O-Einheit* auf einen anderen Speicher geschrieben werden.

Definition: Cache

Cache ist *schneller* als RAM, kann aber nur *weniger Speicherkapazität* zur Verfügung stellen.

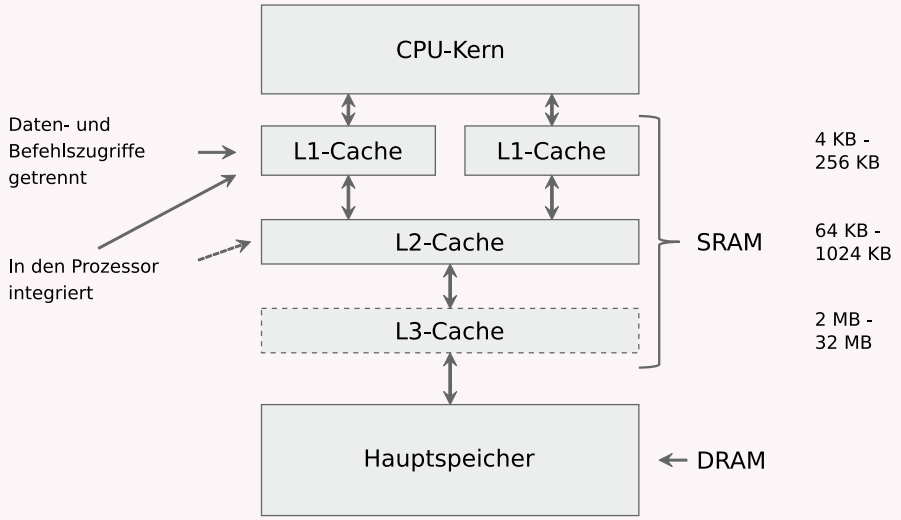
Das bedeutet, dass der Cache nur kleine Datenmengen, sogenannte *Cacheblocks*, aus dem Hauptspeicher vorhalten kann. Diese haben eine definierte Größe und können dann schneller in die CPU geldaden werden, als das aus dem RAM möglich wäre.

Definition: Lokalität

Zeitliche Lokalität:
Es ist, bei entsprechender Programmierung, sehr wahrscheinlich, dass auf eine Speicherzelle nicht nur einmal, sondern in kurzer Zeit *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.

Bonus: Aufbau eines Caches



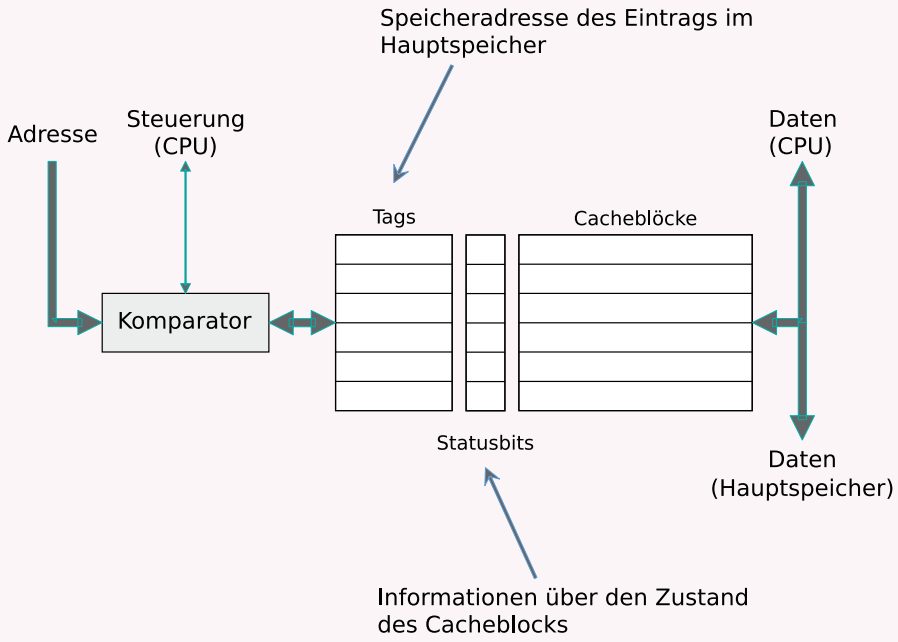
Aus der Abbildung ist ersichtlich, dass der Cache selbst ebenenweise organisiert ist.

Im Regelfall sind mindestens die Ebenen L1 und L2 vorhanden, oftmals sogar noch eine dritte Ebene L3. Dabei ist in jedem Fall der L1-Cache in die CPU integriert, häufig auch noch der L2-Cache.

Beginnend beim L3-Cache werden die Cachelevel mit größerer Nähe zur CPU jeweils kleiner und schneller. Dieser Aufbau soll die Frage nach der Auswahl der Daten, die im Cache vorgehalten werden, vereinfachen. Es ist also möglich einen relativ großen Datenbestand im L3-Cache vorzuhalten, der schneller ist als der Hauptspeicher. Von dort aus kann dann wiederum eine Teilmenge der Daten im noch schnelleren L2-Cache vorgehalten werden, usw.

Liegen benötigte Daten nicht im L1-Cache, welcher Daten bzw. Befehle schlussendlich an die CPU liefert, ist aufgrund der Lokalität und des Aufbaus des Caches die Wahrscheinlichkeit hoch, dass die benötigten Daten nicht aus dem langsamen Hauptspeicher geladen werden müssen, sondern sich in einem der niedrigeren Cache-Level finden und damit immer noch vergleichsweise schnell zur Verfügung gestellt werden können.

Bonus: Interner Aufbau eines Cachelevels



Zu jedem Cacheblock wird die Startadresse des Blocks im RAM gespeichert, diese Information wird *Tag* genannt.

Wird von der CPU eine Anfrage an den Speicher gestellt, wird zunächst anhand des Tags unter Zuhilfenahme des Komparators geprüft, ob der angefragte Datensatz im Cache liegt. Dabei wird auch geprüft, ob eine angefragte Speicheradresse innerhalb eines Cacheblocks liegt. Das ist möglich, da der Tag sowie die Größe des Cacheblocks bekannt sind.

Ist dies nicht der Fall, wird die Anfrage an ein niedrigeres Cachelevel bzw. den RAM weitergereicht.

Außerdem werden zu jedem Cacheblock Statusbits gespeichert, die beispielsweise angeben ob sich der Cacheblock verändert hat (*dirty bit*), seit er in den Cache geladen wurden, oder ob ein Platz im Cache mit einem gültigen Cacheblock belegt ist (*invalid bit*).

Definition: Organisation der Tags

Die *Blöcke* (Cache-Lines) eines Caches können in so genannte *Sätze* zusammengefasst werden. Für eine bestimmte Adresse ist dann immer nur einer der Sätze zuständig. Innerhalb eines *Satzes* bedienen alle Blöcke also nur einen *Teil* aller vorhandenen Adressen.

Im Folgenden stehe die Variable m für die *Gesamtanzahl der Cacheblöcke* und n für die *Anzahl der Blöcke pro Satz*, die so genannte *Assoziativität*. Dann besteht der Cache also aus $\frac{m}{n}$ Sätzen.

Direkt abgebildet (*DM*, *direct mapped*, $n = 1$):
Es gibt pro Cacheblock nur eine einzige Möglichkeit, wo dieser platziert werden kann.

Daher kann es allerdings vorkommen, dass ein Cacheblock nicht platziert werden kann, obwohl noch Platz im Cache wäre.

Vollassoziativ (*FA*, *fully associative*, $n = m$):
Ein Cacheblock kann beliebige auf freie Plätze im Cache zugeordnet werden.

Bei einer Speicheranfrage müssen allerdings alle gespeicherten Tags durchsucht werden.

Satzassoziativ (*SA*, *set associative*, $2 \leq n \leq \frac{m}{2}$):
Der verfügbare Platz wird in Gruppen unterteilt. Wie bei einem DA Cache gibt es nur eine Gruppe, in der ein Cacheblock platziert werden kann; wie bei einem VA Cache kann der Cacheblock innerhalb dieser Gruppe frei platziert werden.

Definition: Cache-Lesezugriffe

Findet ein Lesezugriff auf Speicherzelle A statt, wird geprüft, ob Speicherzelle A bereits im Cache liegt.

- A **liegt im Cache** (*cache hit*):
  Datensatz kann direkt aus dem Cache gelesen werden.
- A **liegt nicht im Cache** (*cache miss*):
  Datensatz muss aus dem Hauptspeicher in den Cache geladen werden, danach wird der Datensatz gelesen.

Definition: Schreibmodi eines Cache

- **write-through** (*WT*):
  Datensatz wird im Cache und direkt im Hauptspeicher aktualisiert.
  Vorteile: keine Probleme mit Datenkonsistenz im Hauptspeicher
  Nachteile: hoher Aufwand für Schreiboperationen
- **write-back** (*WB*):
  Datensatz wird im Cache aktualisiert und erst dann in den Hauptspeicher geschrieben, wenn der entsprechende Cacheblock aus dem Cache verdrängt wird.
  Vorteile: niedrige Belastund der Systembusse, keine Wartezyklen
  Nachteile: fehlende Datenkonsistenz

Definition: Cache-Schreibzugriffe

Findet ein Schreibzugriff auf Speicherzelle A statt, wird geprüft, ob Speicherzelle A bereits im Cache liegt.

- A **liegt im Cache** (*cache hit*):
  Datensatz wird im Cache (und im Hauptspeicher) aktualisiert.
- A **liegt nicht im Cache** (*cache miss*):
  Datensatz wird im Hauptspeicher geschrieben, Inhalt des Caches wird nicht verändert.

Definition: Cache Misses

- **Capacity Miss**:
  - tritt auf, wenn Datensatz bereits im Cache war, aber *bereits verdrängt* wurde (aufgrund mangelnder Kapazität)
  - hauptsächlich bei VA Caches
- **Compulsory Miss**:
  - tritt auf, wenn ein Datensatz das *erste Mal* verwendet wird
  - unabhängig vom Typ des Caches
- **Conflict Miss**:
  - tritt auf, wenn Datensatz bereits im Cache war, aber *bereits verdrängt* wurde (weil ein anderer Cacheblock an entsprechende Stelle gelagert werden sollte)
  - vor allem bei DA Caches

Definition: RAID

RAID: **R**edundant **A**rray of **I**ndependent **D**isks

Ein *RAID* besteht aus mindestens zwei Festplatten und zielt 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

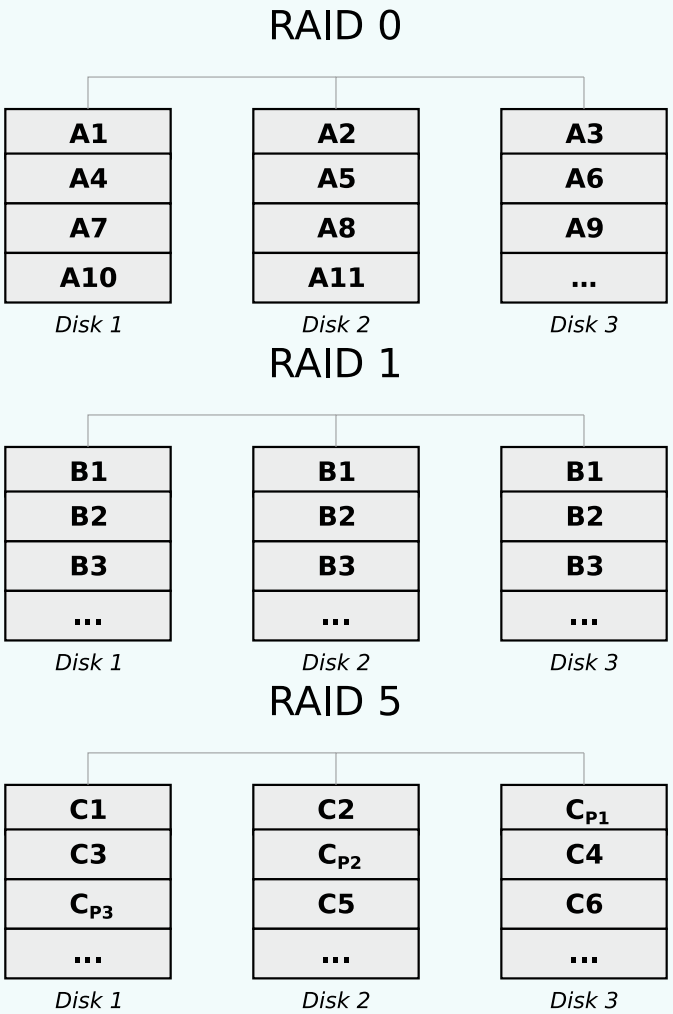
Definition: RAID-Level

Die genaue Funktionsweise des RAID wird durch das sogenannte *RAID-Level* angegeben.

- **RAID 0**:
  - höhere Transferraten durch *Striping*
  - Daten werden auf mehrere Festplatten verteilt
  - beim Lesen und Schreiben können mehrere Festplatten parallel verwendet werden

  Nachteil: fällt eine Festplatte aus, sind meist alle Daten verloren
- **RAID 1**:
  - erhöhte Ausfallsicherheit durch *Mirroring*
  - Daten in gleicher Weise auf mehrere Festplatten gleichzeitig abgelegt
  - einzelne Daten *können* auch parallel von mehreren Festplatten gelesen werden

  Nachteil: wird eine Datei gelöscht, wird sie auf allen Platten gelöscht (kein Backup!)
- **RAID 5**:
  - versucht Vorteile von RAID 0 und RAID 1 zu vereinen
  - höhere Ausfallsicherheit bei höherer Datentransferrate
  - besteht aus mindestens drei Festplatten
  - Verwendet Variante von *Striping*:
    - nicht auf alle n Festplatten verteilt
    - auf allen Festplatten Paritätsinformationen zu Daten auf anderen $n-1$ Platten
    - kann Ausfall einer Festplatte kompensieren



Merhere RAID-Systeme eines Typs können auch zu einem RAID-System zusammengefasst werden (z.B. *RAID 100*, *RAID 01*, *RAID 10*, ...).

Bonus: HDD, SSD

HDD (Hard Disk Drive, *Festplatte*):

- Gehäuse der Festplatte beinhaltet mehrere, auf einer Achse übereinander montierten, runden Platten, welche mit einer magnetisierbaren Schicht überzogen sind
- Schreib-/Leseköpfe werden durch einen zentralen Kamm über die Platten bewegt
- „Landing Zone“ zum Parken der Köpfe (Berührung → Datenverlust)
- Platten rotieren mit konstanter Umdrehungszahl (5400-15000 rpm)
- Kenngrößen:
  - kontinuierliche Übertragungsrate (*sustained data rate*)
  - mittlere Zugriffszeit (*(data) access time*), bestehend aus:
    - Spurwechselzeit (*seek time*)
    - Latenzzeit (*latency*)
    - Kommando-Latenz (*controller overhead*)

SSD (Solid State Drive):

- keine mechanischen Bauteile
- niedriger Energieverbrauch
- hoher Datendurchsatz
- hoher Preis pro Speichereinheit

Bonus: Daten- und Adressbus

Einzelne Komponenten sind über Leitungen, sogenannte *Busse* verbunden.

- *Datenbus* (bi-direktional)
- *Adressbus* (uni-direktional, leitet Adressanfragen der CPU an RAM oder Cache weiter)

3.2 Parallele Rechnerarchitekturen

Definition: Flynn'sche Klassifikation (Flynn'sche Taxonomie)

	Befehlsströme (*Data Streams*)		
	Einfach (*Single*)	Mehrfach (*Multi*)	
Datenströme (*Data Streams*)	**SISD**	**MISD**	Einfach (*Single*)
	SIMD	**MIMD**	Mehrfach (*Multi*)

- *SISD* entspricht der Von-Neumann-Architektur
- *MIMD* entspricht dem heutigen Mehrprozessorsystem
- *SIMD* entspricht dem Aufbau einer Grafikkarte (genutzt in HPC)
- *MISD* eher ungebräuchlich.

Definition: Shared-Memory Systeme

Ein *Shared-Memory System* teilt den vorhandenen RAM unter den verfügbaren Prozessorenkernen auf. Das bedeutet, dass Daten zwischen den einzelnen Prozessorkernen *implizit über den RAM* verteilt werden können, da jeder Kern Zugriff auf den RAM hat.

SMP (*symmetric multi processing*) skaliert vergleichweise schlecht. Das liegt daran, dass an die vorhandene Basis der Von-Neumann-Architektur einfach weitere Prozessorkerne angeschlossen werden. Da sich diese Prozessorkerne nun aber das vorhandene Bus-System teilen müssen, entsteht an dieser Stelle ein *Flaschenhals*.

ccNUMA (*cache-coherent non-uniform memory architecture*) soll dieses Problem, speziell im Bereich HPC, beheben. Dabei wird der vorhandene Hauptspeicher auf mehrere Memory-Controller aufgeteilt. Jeder Prozessor ist dann an einen eigenen Memory-Controller angeschlossen. Dabei kann grundsätzlich weiterhin jeder Kern auf den gesamten RAM zugreifen. Es kann nur sein, dass der Zugriff länger dauert, wenn der betreffende Teil des RAMs von einem anderen Memory-Controller verwaltet wird.

Definition: Distributed-Memory System

Ein *Distributed-Memory System* verbindet mehrere *unabhängige Recheneinheiten*, sodass Daten *explizit* über eine Netzwerkverbindung zwischen dieses Recheneinheiten verteilt werden müssen.

Dieser Ansatz skaliert sehr gut, d.h. es ist ohne Weiteres möglich weitere Recheneinheiten anzuschließen, ohne die Gesamtperformance des Systems zu beeinträchtigen

Bonus: Speedup und Effizienz

Speed Up und *Effizienz* beurteilen die Güte paralleler Programmausführung, indem sie *Zeitersparnis* und die *Anzahl der verwendeten Prozessorkerne* in Relation setzen.

Sei $T(p)$ die Zeit zur Programmausführung bei Verwendung von p CPUs. Dann sind der *Speed Up* $S(p)$ und die Effizienz $E(p)$ definiert wie folgt:

$$S(p) = \frac{T(1)}{T(p)} \qquad E(p) = \frac{S(p)}{p}$$

Der *Speed Up* gibt an, wieviel schneller die Programmausführung ist. Die Effizienz gibt an, wie gut die verwendeten Prozessorkerne genutzt worden sind.

Im Idealfall ist $S(p) = p$ und $E(p) = 1$.

Bonus: Amdahl's Law

Nach Amdahl wird der Geschwindigkeitszuwachs vor allem durch den sequentiellen Anteil des Problems beschränkt, da sich dessen Ausführungszeit durch Parallelisierung nicht verringern lässt.

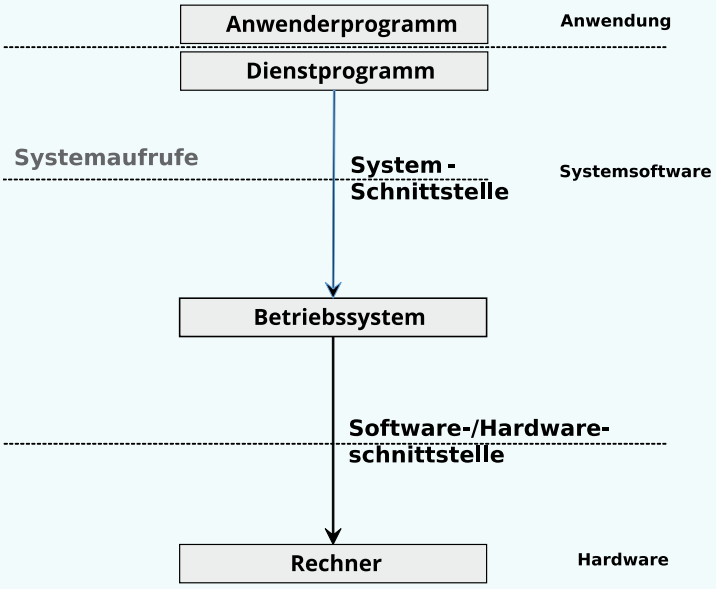
Der *Speed Up* nach Amdahl ist wie folgt definiert ($f \in (0, 1]$: serieller Teil des Programms):

$$S(p) = \frac{T(1)}{f \cdot T(1) + (1-f) \cdot \frac{T(1)}{p}} = \frac{1}{f + \frac{1-f}{p}}$$

4 Betriebssysteme

Definition: Betriebssystem

Das *Betriebssystem* liegt als Softwareschicht zwischen dem Rechner bzw. der *Software-Hardwareschnittstelle*, die das BIOS zur Verfügung stellt, und den Anwenderprogrammen. Das heißt, dass ein Endnutzer nur mit dem Betriebssystem, den vom Betriebssystem bereitgestellten Dienstprogrammen und den Anwenderprogrammen in Kontakt kommt.



Bonus: Anforderungen an ein Betriebssystem

- Hohe Zuverlässigkeit und Leistung
- Einfache Bedienbarkeit und Wartbarkeit
- Niedrige Kosten

Bonus: Batchsysteme

Batchsysteme sind dazu gedacht, Rechenaufgaben ohne Nutzereingabe abzuarbeiten. Dazu gibt es eine *Job Queue*, in welche Aufgaben eingestellt werden. Diese Aufgaben werden dann bearbeitet und die Ergebnisse an den Nutzer ausgegeben.

Bonus: Dialogsysteme

Dialogsysteme sind auf eine Interaktion mit dem Benutzer ausgelegt. Sie sind die wohl geläufigste Form von Betriebssystemen, da diese Form auf z.B. Desktop-Computern eingesetzt wird.

Dialogsysteme werden noch einmal unterteilt in *Single User-* und *Multi-User-Systeme*.

Bonus: Echtzeitsysteme

Echtzeitsysteme sind reaktive Systeme, die mit Hilfe von Sensoren Ereignisse registrieren und anhand von Aktoren darauf reagieren. Dabei ist die zeitliche Abfolge bzw. die Dauer der Ausführung von Interesse.

4.1 Prozess

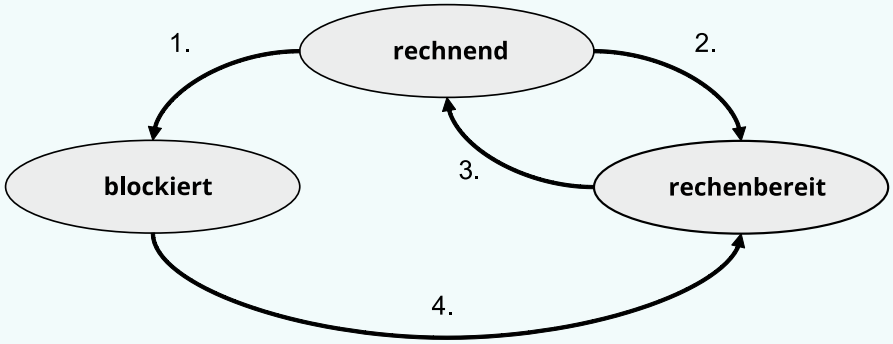
Definition: Prozess

Ein *Prozess* ist die Abstraktion eines in Ausführung befindelichen Programms.

Er besteht aus den *Programmbefehlen* und dem *Prozesskontext*.

Der *Prozesskontext* besteht aus dem privaten Adressraum des Prozessors, geöffneten Streams und abhängigen Prozessen.

Definition: Prozesszustände



1. Der Prozess muss auf ein externes Ereignis warten.
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 externe Ereignis, auf das der Prozess gewartet hat, ist eingetreten.

Bonus: Multitasking

Preemptives Multitasking:

- Betriebssystem entscheidet, wann welcher Prozess zur Ausführung kommt
- Benutzer erhält Eindruck von Parallelität

Kooperatives Multitasking:

- Prozess bestimmt selbst, wann er den Prozessor abgibt
- *Nachteile*
  - z.B. Endlosschleifen können das gesamte System zum Absturz bringen
  - das Betriebssystem kann nicht berechnen, wann der Prozessor wieder frei ist

Definition: Scheduling

Die Zuteilung von Zeitscheiben wird *Scheduling* genannt und ist der Kern der Prozessverwaltung. Das Scheduling sollte dabei jederzeit die folgenden Eigenschaften erfüllen:

- *Fairness*: Jeder Prozess erhält einen gerechten Anteil der CPU-Zeit.
- *Effizienz*: Die CPU und andere Ressourcen sind möglichst vollständig ausgelastet.

Bonus: Priorität

Statische Priorität:

- Jeder Prozess erhält beim Start eine *feste Priorität*
- Prozess mit *höchster Priorität* bekommt als nächstes eine Zeitscheibe zugeteilt
- Oft in Echtzeitsystemen verwendet

Dynamische Priorität:

- Jeder Prozess erhält beim Start eine *Anfangspriorität*
- Prozess mit *höchster Priorität* bekommt als nächstes eine Zeitscheibe zugeteilt
- Prioritäten der Prozesse werden *dynamisch geändert*

Algorithmus: Scheduling: FIFO (First In First Out)

Prozesse werden nach *Reihenfolge* ihres Einfügens in die *Job-Queue* bearbeitet.

- Zuteilung der CPU findet nur statt, wenn laufender Prozess wartet oder sich beendet
- Jeder Prozess kommt garantiert an die Reihe
- Kurze Prozesse müssen unter Umständen sehr lange warten, bis sie ausgeführt werden

Algorithmus: Scheduling: SJF (Shortest Job First)

Prozesse werden aufsteigend nach ihrer *geschätzten Ausführungszeit* bearbeitet.

- Große Prozesse kommen möglicherweise nie an die Reihe, wenn stets kleinere dazukommen
- Wartezeit auf das Ergebnis eines Prozesses sich in etwa proportional zur Ausführungszeit

Algorithmus: Scheduling: MLFQ (Multilevel Feedback Queue)

Bei diesem Ansatz gibt *mehrere FIFO-Queues*, denen jeweils eine *Priorität* zugeordnet ist.

Ein neuer Prozess wird immer in der Queue mit *höchster Priorität* eingeordnet.

Wird der Prozess während der ersten Zeitscheibe fertig, so verlässt er das System.

Gibt er die CPU freiwillig ab, beispielsweise weil er durch das Warten auf ein externes Ereignis blockiert wird, wird er, sobald er wieder bereit ist, in *dieselbe Queue wieder einsortiert* und dort weiter ausgeführt.

Verbraucht der Prozess seine Zeitscheibe vollständig, so wird er in die *nächst-niedriger priorisierte FIFO-Queue* eingereiht. Dort gelten wieder dieselben Regeln wie vorher.

Verbraucht der Prozess immer weiter seine Zeitscheiben vollständig, kommt er schließich in der *am niedrigsten priorisierten Queue* an. Dort verweilt er, bis er abgearbeitet wurde, d.h. es gibt *keine Möglichkeit* wieder in höher priorisierte Queues eingestuft zu werden.

Wieviele FIFO-Queues es gibt, ist vom konkreten Einsatzszenario abhängig.

4.2 Speicherverwaltung

Definition: Reale Speicherverwaltung

Jedem Prozess wird ein zusammenhängener Block im Hauptspeicher zugeteilt. Wird in diesem Kontext der Arbeitsspeicher direkt aus den Prozessen heraus adressiert, spricht man von *realer Speicherverwaltung*.

Das bedeutet auch, dass die Größe des physikalisch vorhandenen Hauptspeichers die Anzahl der gleichzeitig ausführbaren Prozesse begrenzt.

Nachteile:

- Es muss Platz für das *gesamte Programm und die Daten* gefunden werden.
- Es kannt nicht mehr Speicher genutzt werden, als *physikalisch vorhanden*.
- Anforderung, zusammenhängender Speicherblöcke zu finden, verschärft Problem der *Fragmentierung*.

Bonus: Fragmentierung

Fragmentierung passiert dann, wenn mehrere, kleine Blöcke im Hauptspeicher frei sind und unter der Prämisse, dass einem Prozess ein zusammenhängender Block im Hauptspeicher zugeordnet werden muss, dies eventuell zu einer Situation führt, in der kein neuer Prozess gestartet werden kann, obwohl in Summe genügend Hauptspeicher frei wäre.

Definition: Swapping

Beim *Swapping* wird der Hauptspeicherinhalt eines Prozesses auf den *Hintergrundspeicher*, beispielsweise eine Festplatte (HDD), *ausgelagert*, um Platz für andere Prozesse zu schaffen.

Bekommt dann der Prozess, dessen Daten gerade auf dem Hintergrundspeicher liegen, die CPU zugeteilt, müssen seine Daten *erneut in den Hauptspeicher geladen werden*, wahrscheinlich nachdem die Daten eines anderen Prozesses ausgelagert wurden.

Definition: Virtuelle Speicherverwaltung

Jedem Prozess wird ein *scheinbar zusammenhängender Speicherbereich* zur Verfügung gestellt. Tatsächlich besteht der Speicher des Prozesses aus nicht zwangsläufig zusammenhängenden *virtuelle Pages*.

Der Prozess kann seinen Speicher mit *virtuellen Adressen* beginnend bei 0 adressieren.

Die Gesamtheit aller virtuellen Adressen wird als *virtueller Adressraum* bezeichnet.

Definition: Virtuelle Pages

Virtuelle Pages werden auf Blöcke im Hauptspeicher gleicher Größe abgebildet.

Hier kann auch *Swapping* genutzt werden. In diesem Fall aber für einzelne Pages, nicht für den gesamten Hauptspeicherinhalt des Prozessors.

Definition: Pagetable

Beim Zugriff auf eine virtuelle Speicheradresse durch einen Prozess muss diese Adresse in eine physikalische Adresse umgewandelt werden. Das geschieht anhand der *Pagetable*, die das Betriebssystem *für jeden Prozess* erstellt und aktualisiert.

Da die Pagetable virtuelle Pages auf physikalische Pages gleicher Größe abbildet, gibt es einen Teil der Adresse, der sogenannte *Offset*, der die Position der Daten innerhalb der Page angibt.

Abhängig von der Größe der Pages besteht das Offset aus m Bits. Für eine Pagegröße von 1MB werden beispielsweise 20 Bits als Offset benötigt.

Der Rest der Adresse, die Seitennummer, muss dann anhand der Pagetable in die Basisadresse umgesetzt werden, um die Adresse im physikalischen Speicher zu erhalten. Da die Seitennummer aus n Bits besteht, kann die Pagetable maximal 2^n Einträge enthalten.

Beispiel: Pagetable

Die Länge einer Adresse sei 16 Bit, aufgeteilt in je 8 Bit für Offset und Seitennummer.

Es sei außerdem folgende Seitentabelle gegeben:

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

Dann können virtuelle Adressen anhand dieser Pagetable wie folgt umgesetzt werden:

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

Hinweis: Ist eine Adresse ungültig, wurde die dazugehörige Page in den Hintergrundspeicher ausgelagert. In diesem Fall muss die physikalische Page in den RAM geladen und die Pagetable aktualisiert werden.

Bonus: Paging on Demand

Das Vorgehen, aktuell unbenutzte Pages aus dem Hauptspeicher auf den Hintergrundspeicher auszulagern wird auch als *Paging on Demand* bezeichnet. Das Ziel dabei ist, Arbeitsspeicher für andere Prozesse freizugeben.

Dabei kann ein Prozess entweder Platz für eine bestimmte Anzahl von physikalischen Pages zugewiesen bekommen, die sich im Laufe der Prozessabarbeitung nicht ändert, oder es wird dynamisch anhand der aktuellen Speicherauslastung entschieden, wieviel Platz ein Prozess belegen darf.

Algorithmus: Speicherverwaltung: FIFO (First In First Out)

Beim *FIFO*-Verfahren wird diejenige Page ausgelagert, welche sich schon am längsten im Hauptspeicher befindet. Dazu muss in der Pagetable festgehalten werden, wann welche Page in den Hauptspeicher geladen wurde.

Algorithmus: Speicherverwaltung: LRU/LFU (Least Recently / Frequently Used)

Bei *LRU* wird mitgehalten, wieviele Ladevorgänge seit der letzten Benutzung einer Page vorgenommen wurden. Das heißt, dass im Gegensatz zu FIFO, der Kontrollzustand bei der Verwendung einer Page wieder auf „0“gesetzt wird.

Beispiel: Speicherverwaltung

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

FIFO-Strategie:

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	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

9 Einlagerungen

LRU-Strategie:

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	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

10 Einlagerungen

4.3 Dateisystemverwaltung

Definition: BIOS (Basic Input/Output System)

Das *BIOS*:

- ist die *Firmware* bei x86-PCs
- liegt im *nichtflüchtigen Speicher* auf der Hauptplatine des PCs
- leitet den *Start* des *Betriebssystems* ein

Definition: UEFI (Unified Extensible Firmware Interface)

UEFI ist die zentrale Schnittstelle zwischen:

- der *Firmware*
- den *einzelnen Komponenten* eines Rechners
- und dem *Betriebssystem*

Bonus: MBR (Master Boot Record)

Der *MBR* besteht aus insgesamt 512 Byte, die sich auf 446 Byte für einen (optionalen) Bootloader, 64 Byte für die *Partitionstabelle* und 2 Byte für eine (0xAA55) aufteilen. Die Magic Number dient dazu, einen gültigen MBR zu identifizieren.

In der Partitionstabelle können maximal 4 Partitionen definiert werden, d.h. die Festplatte kann in maximal 4 logische Einheiten aufgeteilt werden.

Bonus: GPT (GUID Partition Table)

Mit der Einführung von UEFI wurden auch die Limitierungen des MBR aufgehoben und die GPT als Nachfolger definiert.

Die *GPT* beinhaltet zu Beginn aus Kompatibilitätsgründen einen MBR, sodass ein hybrider Betrieb möglich ist. In der GPT können bis zu 128 Partitionen abgelegt werden.

Zur Absicherung der GPT wird eine exakte Kopie der GPT am Ende des Datenträgers abgelegt.

Definition: Dateisystem

Ein Dateisystem ist im Prinzip eine Ablageorganisation für Daten auf einem Datenträger des Computers. Das Dateisystem muss sicherstellen, dass Dateien *lesend und schreibend geöffnet* und auch wieder *geschlossen* werden können. Das bedeutet, dass Dateinamen auf physikalische Adressen auf dem Datenträger abgebildet werden müssen.

Spezielle Eigenschaften des Datenträgers (Festplatte, USB-Stick, ...) müssen berücksichtigt werden.

Generell bieten alle (modernen) Dateisysteme folgende Attribute:

- Dateiname
- Ablageort (Ordner bzw. Verzeichnis)
- Dateigröße
- Zugriffsrecht

Definition: Lineare Dateisysteme

Bei linearen Dateisystemen werden Daten direkt hintereinander auf den Datenträger geschrieben. Das bedeutet, dass wahlfreier Zugriff nicht möglich ist. Daher finden diese Dateiysteme heutzutage nur noch Anwendung in Bereichen, in denen es nicht primär auf Geschwindigkeit ankommt.

Definition: Hierarchische Dateisysteme

Daten werden auf hierarchischen Dateisystemen in einer Verzeichnisstruktur abgelegt.

Diese Art von Dateisystem ist die wohl verbreiteste auf modernen Computern und kann auf Festplatten, SSDs, USB-Sticks, SD-Karten und sonstigen herkömmlichen Datenträgern verwendet werden.

Definition: Netzwerkdateisysteme

In Netzwerkdateisystemen wird entfernter Speicher auf einem Server wie ein lokales Medium behandelt. Das Betriebssystem muss dann die Zugriffe auf Dateien in Netzwerkkommunikation umwandeln. Für den Nutzer eines Betriebsystems stellt sich Netzwerkspeicher allerdings in der Regel wie ein hierarchisches Dateisystem dar.

Bonus: Sicherheitsaspekte

Paralleler Zugriff im Multitasking:

- Bereitstellung von Locks für den Dateizugriff

Stromausfall während einer Schreiboperation:

- Es muss Datenkonsistenz gewährleistet werden
- Atomare Operationen, welche entweder abgeschlossen oder ausstehend sind
  → Journalingdateisysteme

Definition: Journalingdateisysteme

Bei einem Journalingdateisystem werden alle Aktionen auf der Festplatte protokolliert und erst als gültig aufgefasst, nachdem das Beenden der Aktion auf dem Dateisystem im *Journal* (Protokoll) vermerkt wurde.

- *Metadatenjournaling*:
  - Konsistenz des Dateisystems
- *Fulljournaling*:
  - Konsistenz des Dateisystems
  - Konsistenz der Dateiinhalte

5 Virtualisierung

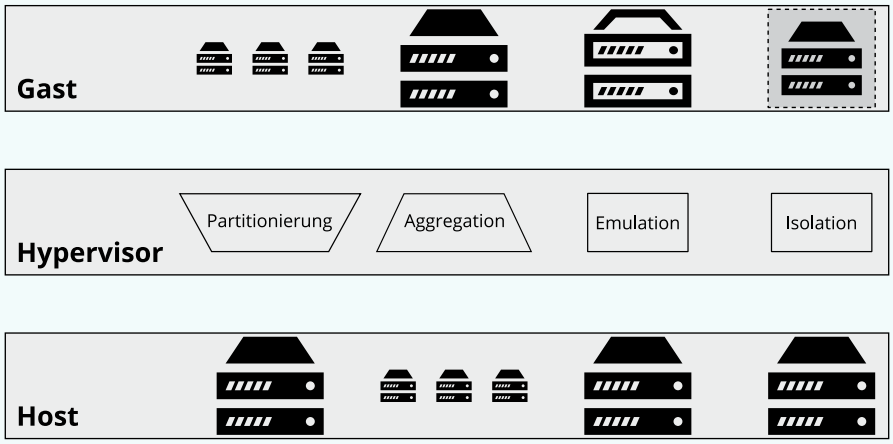
Definition: Virtualisierung

Virtualisierung bezeichnet Methoden, die es erlauben, Ressourcen eines Computers *zusammenzufassen oder aufzuteilen*.

Dies wird erreicht, indem real existierende Hardware unter Zuhilfenahme einer Softwareschicht zu virtueller Hardware *abstrahiert* wird.

Dabei können mehrere Szenarien unterschieden werden:

- Partitionierung
- Aggregation
- Emulation
- Isolation



Definition: Hypervisor / Virtual Machine Monitor (VMM)

Der *Hypervisor* ist ein Stück Soft- oder Hardware, das die Umsetzung zwischen der *virtuellen Maschine* und der *physikalischen Hardware* vornimmt.

Typ-1-Hypervisor:

- läuft direkt auf physikalischer Hardware

Typ-2-Hypervisor:

- läuft als Anwendung auf dem Hostsystem

5.1 Virtualisierungskonzepte

Definition: Paravirtualisierung

- Funktionalitäten des Gast BS werden gezielt verändert (Kernel Anpassungen)
- Gast-BS „weiß“, dass es sich in einer virtuellen Umgebung befindet
- Gast-BS kann direkt mit dem Hypervisor interagieren, benötigt keine Hardware-Emulation

Vorteile: gute Performance

Nachteile: Gastsysteme nicht beliebig wählbar, hoher Aufwand für Kernel-Entwickler

Definition: Hardware-unterstützte Virtualisierung

- Neue Prozessortechnologien: CPUs besitzen Befehlssatz, der Virtualisierung unterstützt
- Modifikation des Gast-BS soll vermieden werden, direkt durch Hardware gelöst
- Hypervisor soll durch hardwarebasierte Speicherverwaltung entlastet werden

Vorteile: Gast-BS muss nicht modifiziert werden, Gastsysteme frei wählbar

Nachteile: kein gemeinsamer Standard, Virtualisierungsplattform muss Technologie unterstützen

Definition: Hardware-Emulation

- Innerhalb einer VM wird Standardhardware eines Rechners komplett oder teilweise simuliert
- Emulator erzeugt Softwareschnittstellen, die vom Gast-BS angesprochen werden können
- Emulator sorgt dafür, dass Befehle, die an simulierte Hardware gerichtet sind, für die physische Hardware des Hostsystems umgewandelt werden

Vorteile: flexible Wahl der Gast-BS

Nachteile: Performanceverlust durch hohen Virtualisierungsaufwand

Definition: Betriebssystemvirtualisierung

- Innerhalb des Host-BS werden Virtual Environments (VE) / Container erzeugt
- In VE ist kein eigenständiges Betriebssystem installiert
- Kernel-Bibliotheken und Geräte-Treiber des Hostsystems genutzt
- Einige Individualdaten müssen für Container definiert werden (z.B. Dateisystem, IP Adresse, Hostname)

Vorteile: gute Performance, wenig Speicherbedarf

Nachteile: keine freie Wahl des Gast-BS (gebunden an Hostsystem)

5.2 Cloud Computing

Definition: Cloud Computing

Cloud Computing ist im Wesentlichen ein Model um einen allgegenwärtigen, bequemen, bedarfsgesteuerten Netzwerkzugang zu einem gemeinsamen Pool konfigurierbarer Computer-Ressourcen zur Verfügung zu stellen. Zudem soll das Ganze schnell und mit minimalem Verwaltungsaufwand und Interaktion mit dem Provider funktionieren.

Definition: Servicemodelle

Infrastructure as a Service (IaaS):

- vom Anbieter verwaltete Infrastruktur
- Nutzer muss verwendetes Betriebssystem und Software vollständig selbst verwalten

Platform as a Service (PaaS):

- virtuelle Plattform zur Verfügung gestellt (Betriebssystem, Entwicklungsplattform)
- alles „unterhalb" der Plattform ist aber *nicht* unter der Kontrolle des Nutzers

Software as a Service (SaaS):

- nur eine einzelne Anwendung zur Verfügung gestellt
- alles „unterhalb" der Anwendung ist aber *nicht* unter der Kontrolle des Nutzers

Bonus: Charakterisken von Cloud Computing

- Selbstverwaltung (On-demand self-service)
- Breitband Internetzugang
- Ressourcenbündelung
- Elastizität
- Leistungsmessung

Definition: Bereitstellungsmodelle

Private Cloud:

- für eine ganz spezielle Nutzergruppe betrieben
- kann auch von der Firma selbst verwaltet werden

Community Cloud:

- wird für verschiedene Nutzergruppen in einem bestimmten Kontext betrieben
- in der Regel von einer der teilnehmenden Gruppen oder von externem Dienstleister bereitgestellt

Public Cloud:

- für beliebige Nutzer zur Verfügung steht
- von beliebigem Anbieter betrieben

Hybrid Cloud:

- kombiniert mehrere der vorhergehenden Bereitstellungsmodelle
- einzelnen Teile unabhängig voneinander, aber durch standardisierte Schnittstellen verbunden

6 Datenschutz und Sicherheit

6.1 Datensicherheit

Definition: Kryptographie

Kryptographie ist die Wissenschaft der Verschlüsselung von Informationen. Sie wird genutzt um Informationen auf eine Art und Weise zu verändern, sodass ein Unbefugter die Informationen nicht mehr lesen kann. Es soll nur dem tatsächlichen Adressaten einer Nachricht möglich sein, diese auch zu lesen. Es geht also explizit nicht darum, den Diebstahl von Informationen zu verhindern, sondern es geht darum, dass entwendete Informationen nicht gelesen werden können.

Definition: Transpositionschiffren

Die Verschlüsselung wird durch Umordnung der Zeichen im Klartext umgesetzt.

Definition: Substitutionschiffren

Die Verschlüsselung wird durch Ersetzung der Zeichen im Klartext realisiert.

Monoalphabetische Substitution:

- Jedes Zeichen des Klartexts wird durch genau ein eindeutiges Zeichen aus dem Geheimtextalphabet ersetzt, das für einen gewählten Schlüssel immer gleich ist.

Homophone Substitution:

- Jedes Zeichen des Klartexts wird durch genau ein anderes Zeichen aus einer eindeutigen Menge von Zeichen ersetzt, die für einen gewählten Schlüssel immer gleich ist.

Polyalphabetische Substitution:

- Jedes Zeichen wird durch ein eindeutiges Zeichen aus einem von mehreren geheimen Alphabeten ersetzt, die für einen gewählten Schlüssel immer gleich sind. Die Alphabete werden dabei immer der Reihe nach verwendet.

Beispiel: Cäsar Chiffre

Im einfachsten Fall beinhalten $\mathfrak{P}$ und $\mathfrak{S}$ dieselben Zeichen und sind nur gegeneinander um k Zeichen verschoben. Diese Variante einer *monoalphabetischen Substitutionschiffre* wird *Cäsar Chiffre* genannt.

Alice wählt für die Verschlüsselung $k = 10$ und erhält damit die folgenden Alphabete:

$\mathfrak{P}$: 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
$\mathfrak{S}$: 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

Da Alice $k = 10$ gewählt hat, beginnt das Geheimtextalphabet mit J, dem zehnten Buchstaben des Alphabets. Die Verschlüsselung führt Alice durch, indem sie jeden Buchstaben ihres Klartexts durch den entsprechenden Buchstaben aus dem Geheimtextalphabet ersetzt.Werden die beiden Alphabete wie weiter oben aufgeschrieben, ist das einfach der Buchstabe im Geheimtextalphabet der direkt unter dem Buchstaben im Klartextalphabet steht. Damit ergibt sich für die Nachricht HALLO der Geheimtext QJUUX. Bob kann den Geheimtext dann entschlüsseln, indem er wiederum beide Alphabete untereinander schreibt und jedem Buchstaben aus dem Geheimtext den zugehörigen Buchstaben aus dem Klartextalphabet zuorndet, der dann genau darüber steht.

Beispiel: Vigenère Chiffre

Eine bekannte *polyalphabetische Substitutionschiffre* ist die *Vigenère Chiffre*. Sie funktioniert im Prinzip wie die Cäsar-Chiffre, verwendet also verschobene lateinische Alphabete zur Verschlüsselung. Wie viele Alphabete und wie jedes von ihnen verschoben, wird durch den Schlüssel k bestimmt, der in diesem Fall ein Wort ist. Es gibt also soviele geheime Alphabete, wie es Zeichen in k gibt und jedes dieser Alphabete ist verschoben, so dass das i-te Alphabet mit dem i-ten Zeichen von k beginnt. Alice wählt das Schlüsselwort $k = EDV$. Damit ergeben sich Klartext- und Geheimalphabete wie folgt:

$\mathfrak{P}$: 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
$\mathfrak{S}_1$: 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
$\mathfrak{S}_2$: 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
$\mathfrak{S}_3$: 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

Damit kann Alice den Klartext $M =$ HALLOHALLO verschlüsseln, indem Sie die Buchstaben des Klartexts wie bei der Cäsar Chiffre zuordnet und dabei die drei Geheimtextalphabete reihum verwendet. Da der Klartext mehr Zeichen enthält, als es Geheimtextalphabete gibt, fängt sie nach jeweils drei Zeichen wieder beim ersten Geheimtextalphabet an. Somit ergibt sich der Geheimtext $C =$ LDGPRCEOGS.

Definition: Moderne Verschlüsselungsverfahren

Moderne Verschlüsselungsverfahren werden, im Gegensatz zu klassischen Verfahren, nicht mehr auf Zeichen einer natürlichen Sprache angewandt. Stattdessen werden Zahlen bzw. Bits als Grundlage der Operationen verwendet. Damit können beliebige Daten verschlüsselt werden, nicht mehr nur Text. Damit sind moderne Verschlüsselungsverfahren den Anforderungen der heutigen digitalen Gesellschaft gewachsen.

Symmetrische Verfahren:

- denselben Schlüssel zur Ver- und Entschlüsselung
- Problem des Schlüsselaustauschs
- Vertraulichkeit, aber keine zur Sicherstellung von Authentifizierung und Integrität

→ Blockchiffren
- zu verschlüsselnde Daten in m Blöcke derselben Größe aufgeteilt
- einfachster Fall: jeder der zuvor erzeugten Blöcke m_i mit Schlüssel k zu verschlüsseltem Block c_i derselben Größe übersetzt

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

Asymmetrische Verfahren:

- Die meisten asymmetrischen Verfahren basieren auf mathematischen Problemen, die nicht effizient zu lösen sind.
-

Algorithmus: RSA Verschlüsselung

RSA basiert auf der Faktorisierung ganzes Zahlen, für die es keinen bekannten effizienten Algorithmus gibt. Dabei geht es darum, eine Zahl in ihre Primfaktoren zu zerlegen.

Da beide Schlüssel des Schlüsselpaars zusammen funktionieren sollen, müssen sie nach einer festen Vorschrift erzeugt werden:

1. Wähle zwei Primzahlen p, q mit $p \neq q$
2. Berechne $N = p \cdot q$
3. Berechne $\varphi(N) = (p-1) \cdot (q-1)$
4. Wähle ein e, das teilerfremd zu $\varphi(N)$ ist mit $1 < e < \varphi(N)$
5. Berechne d, sodass $e \cdot d \equiv 1 \mod \varphi(N)$

Nach dieser Prozedur ist (e, N) der *public key* und (d, N) der *private Key*. Damit kann nun jeder Nachricht $M \in \mathbb{N}, 1 < M < N$ anhand folgender Formel verschlüsselt werden:

$$C = M^e \mod N$$

Die Entschlüsselung der verschlüsselten Nachricht C kann dann anhand einer ähnlichen Formel durchgeführt werden:

$$M = C^d \mod N$$

Beispiel: RSA Verschlüsselung

Damit Alice Bob eine Nachricht schreiben kann, muss Bob zunächst seinen *public Key* zur Vefügung stellen.

Zur Erzeugung seines Schlüsselpaars wählt Bob die Primzahlen $p = 13$ und $q = 17$. Damit ergibt sich $N = 221$ und $\varphi(N) = 192$. Anschließend wählt Bob $e = 5$ und berechnet damit $d = 77$. Damit kann Bob $(5, 221)$ als seinen *public Key* an Alice geben und $(77, 221)$ behält er als *private Key* für sich.

Alice hat nun Bobs *public Key* und möchte damit die Nachricht $M = 42$ verschlüsseln. Dazu berechnet Sie $C = 42^5 \mod 221 = 9$ und schickt die verschlüsselte Nachricht anschließend an Bob.

Bob kann nun mit seinem *private Key* die erhaltene Nachricht entschlüsseln und erhält $M = 9^7 7 \mod 221 = 42$.

Definition: Digitale Signatur

Um auch *Authentifizierung* und *Integritat* herstellen zu können, muss eine digitale Signatur verwendet werden. Dazu braucht es zunachst eine *Hashfunktion*, die eine Art Fingerabdruck der Nachricht erzeugt.

Nachdem mit der Hashfunktion der Hashwert der Nachricht berechnet wurde, wird dieser mit dem *private Key* des Senders verschlüsselt.

Danach berechnet der Empfänger selbst den Hashwert der empfangen Nachricht. Stimmen der selbst berechnete Hashwert und der entschlüsselte Hashwert überein ist die *Integrität* der Nachricht sichergestellt. Da zudem das Entschlüsseln des Hashwertes mit dem öffentlichen Schlüssel des Senders funktioniert hat, ist der Sender authentifiziert, da dessen privater Schlüssel zur Erstellung der digitalen Signatur verwendet wurde.

Es ist also notwendig, dass sich Sender und Empfanger vorab auf eine Hashfunktion einigen. Zusatzlich ist wichtig zu beachten, dass eine Digitale Signatur *keine Vertraulichkeit* herstellt, dazu muss die Nachricht zusatzlich verschlüsselt werden.

Definition: Hybride Chiffren

Die Kombination von symmetrischen und asymmetrischen Chiffren nennt man *hybride Chiffren*.

- verhältnismäßig kurzer Schlüssel einer Blockchiffre wird asymmetrisch verschlüsselt und zum Empfänger einer Nachricht transportiert
- Kommunikation selbst läuft symmetrisch verschlüsselt ab
- Kombination erzielt guten Kompromiss zwischen Sicherheit und Rechenaufwand

Index

Beispiele