

# Rechnerarchitektur und Betriebssysteme

*Robin Rausch, Florian Maslowski, Ozan Akzebe*

*20. Mai 2023*

## Inhaltsverzeichnis

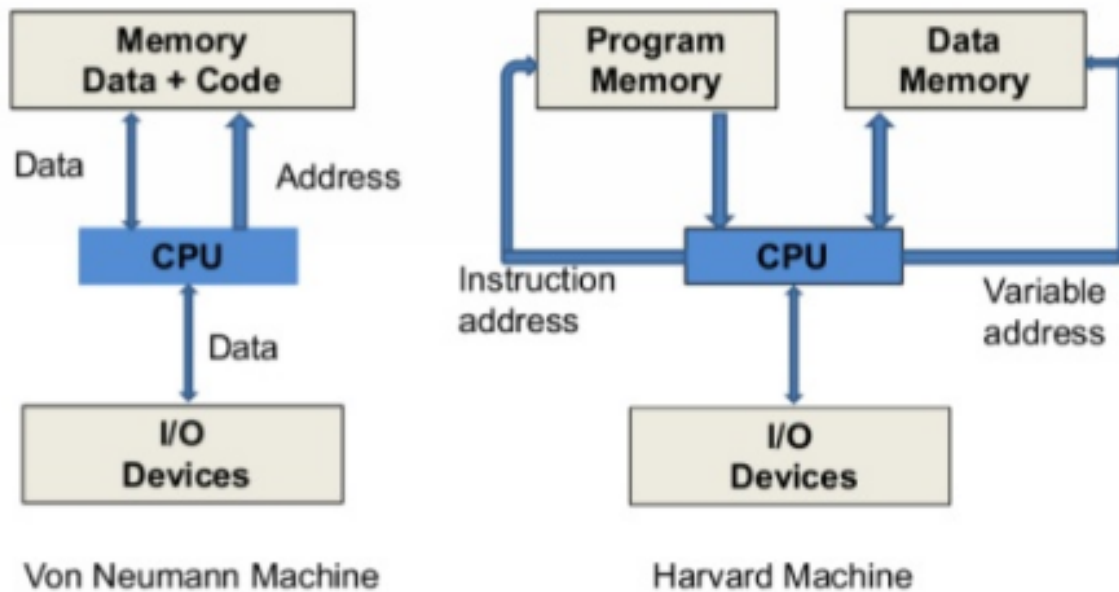
<b>1 Computerarchitekturen</b>	<b>4</b>
1.1 Neumann-Architektur . . . . .	4
1.2 Harvard-Architektur . . . . .	4
<b>2 Betriebssysteme Grundlagen</b>	<b>4</b>
2.1 Architekturen von Betriebssystemen . . . . .	4
<b>3 Binärrechnen</b>	<b>5</b>
3.1 Komplementbildung . . . . .	5
3.1.1 Vorzeichenbetragszahl . . . . .	5
3.1.2 Zweierkomplement . . . . .	5
3.1.3 Kommazahlen . . . . .	5
<b>4 Addierer</b>	<b>6</b>
4.1 Halb-Addierer . . . . .	7
4.2 Volladdierer . . . . .	7
4.3 N-bit Addierer . . . . .	8
4.4 Serien Addierer . . . . .	8
<b>5 Subtrahierer</b>	<b>9</b>
<b>6 Multiplizieren</b>	<b>9</b>
<b>7 Dividieren</b>	<b>9</b>
<b>8 Rechnen mit Komplementzahlen</b>	<b>9</b>
8.1 BCD(=Binary coded decimals) . . . . .	9
<b>9 Schaltungen</b>	<b>10</b>
9.1 Junktoren/binäre Operatoren . . . . .	10
9.1.1 KV-Diagramm . . . . .	10
9.2 Verbindungen: . . . . .	10
<b>10 Bussystem</b>	<b>10</b>
<b>11 Interrupt</b>	<b>11</b>
11.1 Interrupt Controller . . . . .	11

<b>12 Direct Memory Access - DMA</b>	<b>11</b>
<b>13 Fließband/Pipeline</b>	<b>12</b>
<b>14 Betriebssysteme</b>	<b>12</b>
14.1 Architekturen von Betriebssystemen . . . . .	13
<b>15 Speicheraufbau</b>	<b>13</b>
<b>16 Speicherverwaltung und Dateisysteme</b>	<b>14</b>
16.1 Speicherhierarchie . . . . .	14
16.2 Direkte Speicherverwaltung . . . . .	14
16.3 Swapping . . . . .	14
16.4 virtueller Speicher . . . . .	15
16.5 segmentorientierter Speicher . . . . .	16
16.6 seitenorientierter Speicher - <i>Paging</i> . . . . .	18
16.7 Demand Paging . . . . .	18
16.8 MMU . . . . .	18
16.9 Backups . . . . .	18
16.10 RAID . . . . .	18
16.11 FAT . . . . .	19
16.12 NTFS . . . . .	19
16.12.1 Datenträger . . . . .	20
16.13 Extended Filesystem . . . . .	20
<b>17 Kernel</b>	<b>20</b>
17.1 Threading . . . . .	21
17.2 Kernel . . . . .	21
17.2.1 Monolithischer Kernel . . . . .	21
17.2.2 Mikrokern . . . . .	21
17.3 Geschichtetes System . . . . .	21
17.4 Modulares System . . . . .	21
17.4.1 Hybridkernel . . . . .	21
<b>18 Booten</b>	<b>22</b>
18.1 Master Boot Record (MBR) . . . . .	22
18.2 GPT . . . . .	22
18.3 UEFI und BIOS . . . . .	23
<b>19 UNIX</b>	<b>23</b>
<b>20 Prozessverwaltung</b>	<b>23</b>
20.1 Prozesse . . . . .	23
20.1.1 Threads . . . . .	24
20.1.2 Prozesszustände . . . . .	24
20.2 Prozesszustände . . . . .	24
20.3 Mehrprozessbetrieb . . . . .	25
20.4 Multitasking . . . . .	25
20.4.1 Kontext . . . . .	25
20.5 Scheduling . . . . .	26
20.6 Prozess Scheduling Strategien . . . . .	26

---

20.7 Scheduling Strategien . . . . .	26
20.7.1 First Come First Serve . . . . .	26
20.7.2 Shortest remaining time . . . . .	26
20.7.3 Round Robin . . . . .	27
20.7.4 Priority . . . . .	27
<b>21 Virtualisierung</b>	<b>27</b>
21.1 Hypervisor . . . . .	27
<b>22 Powershell</b>	<b>27</b>

# 1 Computerarchitekturen



## 1.1 Neumann-Architektur

1. Ein Adressraum

## 1.2 Harvard-Architektur

1. Getrennte Adressen für Daten und Code
2. Getrennte Bussysteme
3. Kompiliertes Programm nicht ausführbar
4. gleichzeitige Kommunikation

# 2 Betriebssysteme Grundlagen

## 2.1 Architekturen von Betriebssystemen

Schnittstelle zwischen der Anwendersoftware und der Speicher- und Prozessverwaltung.

**Monolithisches System:** sehr schnell, da alles im Kernel geschieht und es kaum Schnittstellen hat. wenn eine Komponente defekt ist, ist der Kernel nichtmehr funktionsfähig. Updates erfordern Gesamtes Update => Größer, zeitintensiver

**Geschichtetes System:**

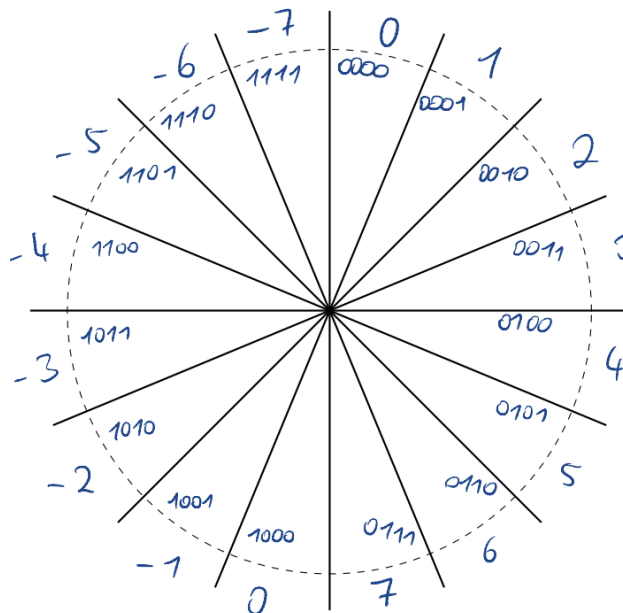
**Modulares System:**

**Mirkoarchitektur:** Prozesse dauern länger, da auch Prozesse/Treiber ausgelagert sind

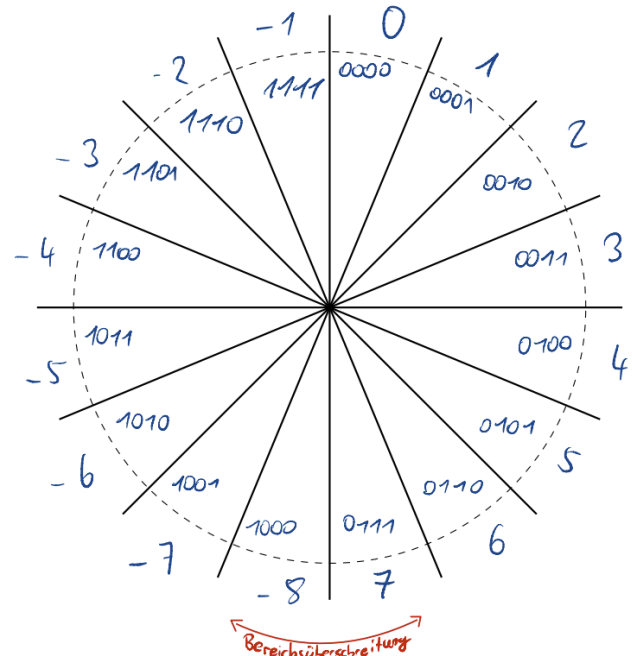
## 3 Binärrechnen

### 3.1 Komplementbildung

#### 3.1.1 Vorzeichenbetragszahl



#### 3.1.2 Zweierkomplement



#### 3.1.3 Kommazahlen

Vorkommastellen      Nachkommastellen

$2^n \dots 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$  ,  $2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \dots 2^{-n}$

am Beispiel einer 8-Bit Dualzahl 4 Vorkomma- und 4 Nachkommastellen

8   4   2   1   ,    $\frac{1}{2}$     $\frac{1}{4}$     $\frac{1}{8}$     $\frac{1}{16}$    Hauptnenner  $16 = 2^4$

## 4 Addierer

### A) keine Bereichsüberschreitung

★ positiv + positiv

$$\begin{array}{r} 2 \\ +3 \\ \hline 5 \end{array}$$

$$\begin{array}{r} 0010 \\ + 0011 \\ \hline 0101 \end{array} = \checkmark$$

$$\begin{array}{l} C_{n-1} = 0 \\ C_{n-2} = 0 \end{array}$$

★ negativ + positiv

$$\begin{array}{r} -2 \\ +3 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 1110 \\ + 0011 \\ \hline 0001 \end{array} = \checkmark$$

$$\begin{array}{l} C_{n-1} = 1 \\ C_{n-2} = 1 \end{array}$$

### B) mit Bereichsüberschreitung

★ positiv + positiv

$$\begin{array}{r} 4 \\ +5 \\ \hline 9 \end{array}$$

$$\begin{array}{r} 0100 \\ + 0101 \\ \hline 1001 \end{array} \neq \checkmark$$

$$C_{n-1} = 0$$

$$C_{n-2} = 1$$

★ negativ + negativ

$$\begin{array}{r} -6 \\ + (-5) \\ \hline -11 \end{array}$$

$$\begin{array}{r} 1010 \\ + 1011 \\ \hline 0101 \end{array} \neq \checkmark$$

$$C_{n-1} = 1$$

$$C_{n-2} = 0$$

## 4.1 Halb-Addierer

Formel

$$x + y = c | s$$

Wahrheitstafel

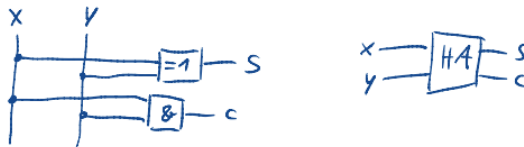
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Boolesche Gleichung

$$s = (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

$$c = x \wedge y$$

Schaltung / Symbol



Laufzeit

Sei  $\Delta T :=$  Laufzeit eines Gatter (1, v)  
dann folgt  $t_{HA} = 2 \Delta T$

## 4.2 Volladdierer

Formel

$$x + y + u = c | s$$

Wahrheitstafel

	x	y	u	c	s
	0	0	0	0	0
	0	1	0	0	1
	1	0	0	0	1
	1	1	0	1	0
	0	0	1	0	1
	0	1	1	1	0
	1	0	1	1	0
	1	1	1	1	1

Boolesche Gleichungen

$$(x \wedge y) \vee ((x \vee y) \wedge u)$$

Symbol



Schaltung(s-Varianten)

· zweistufiges Netz

(selber zeichnen)

· Laufzeit

· aus HA

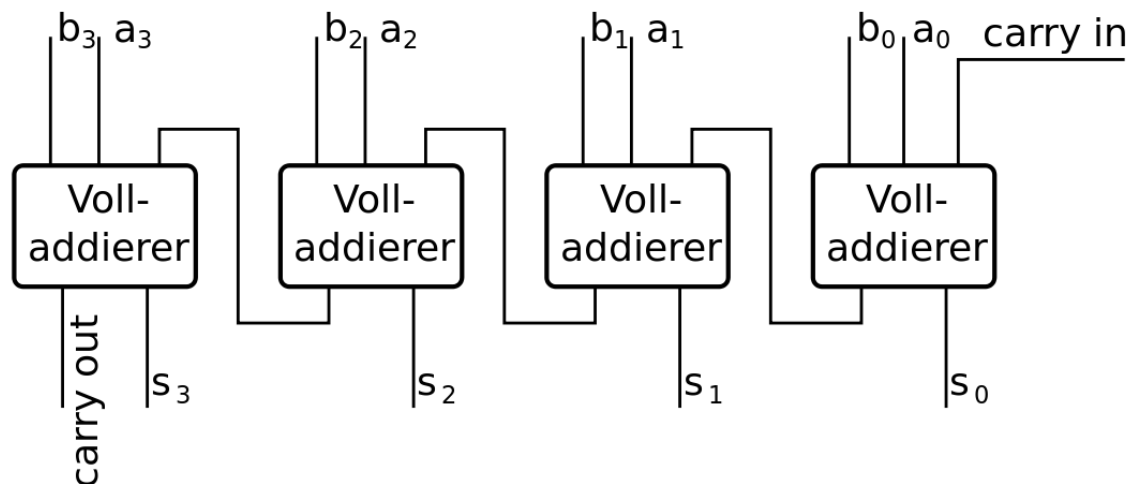
· Schaltung aus HA



· Laufzeit

$$t_{VA \text{ aus HA}} = 5 \Delta T$$

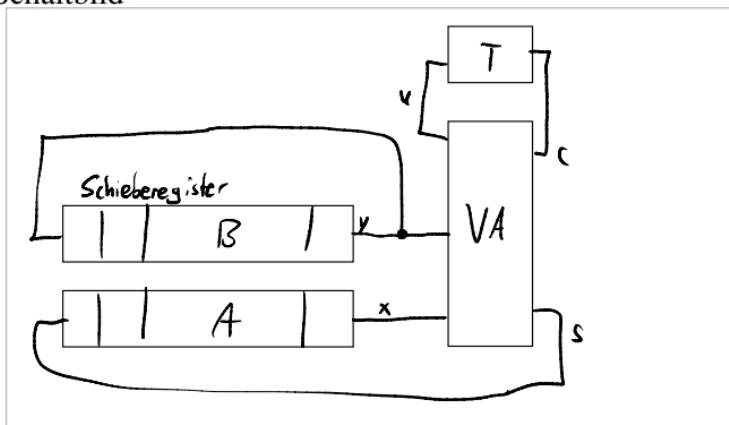
## 4.3 N-bit Addierer



## 4.4 Serien Addierer

### Serien Addierer

- › Addition mit Speicher
- › z.B. 1 Ein-Bit-Addierer für N-Bit Zahl
- $\vec{a} + \vec{b} = c | \vec{e}$
- › Schaltbild



- › Ablauf / Flussdiagramm

- init: Schiebereg. füllen  
→ Speicher(T) löschen

- Rechnen:  
→ VA:  $x + y + u = c | s$



## 5 Subtrahierer

Das (Zweier-)Komplement der negativen Zahl bilden und mit der anderen addieren.  
 Die entehende Zahl ist das (Zweier-)Komplement des Ergebnisses.

<i>OhneKomplement :</i>	<i>MitKomplement :</i>
01101	01101
−01110	+10010
<hr/>	
=11111	=11111
⇒ − 00001	⇒ − 00001

## 6 Multiplizieren

$$1010 * 1011 = 111110$$

$$\begin{array}{r}
 1010 \\
 +1010 \\
 +0000 \\
 +1010 \\
 \hline
 =1101110
 \end{array}$$

## 7 Dividieren

$$\begin{array}{r}
 10/5=2 \\
 1010/101=0010 \\
 \begin{array}{r}
 \underline{-0} \\
 10 \\
 \underline{-0} \\
 101 \\
 \underline{-101} \\
 0 \\
 \underline{-0} \\
 0
 \end{array}
 \end{array}$$

## 8 Rechnen mit Komplementzahlen

### 8.1 BCD(=Binary coded decimals)

Dezimalziffer wird mit jeweils 4 Bits dargestellt, die zusammen als Tetrade bezeichnet werden. Tetraden im Bereich  $9 < x < 16$ , werden Pseudotetraden genannt.  
 Fällt das Ergebnis in den Bereich der Pseudotetraden muss es mit 6 addiert werden.

$$\begin{array}{r}
 1 \quad 1 \quad 1 \quad 1 \\
 0 \quad 1 \quad 0 \quad 1 \\
 + \quad 0 \quad 0 \quad 1 \quad 1 \\
 \hline
 1 \quad 0 \quad 0 \quad 1
 \end{array}
 \quad
 \begin{array}{r}
 0 \quad 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad 1 \quad 1
 \end{array}$$
  

$$\begin{array}{r}
 1 \quad 1 \quad 1 \\
 1 \quad 1 \quad 1 \quad 1 \\
 + \quad 0 \quad 1 \quad 1 \quad 0 \\
 \hline
 0 \quad 1 \quad 0 \quad 1
 \end{array}$$

## 9 Schaltungen

### 9.1 Junktoren/binäre Operatoren

Nach bindungsstärke sortiert. Klammern binden am stärksten.

$\neg$  NICHT

$\wedge$  UND

$\vee$  ODER

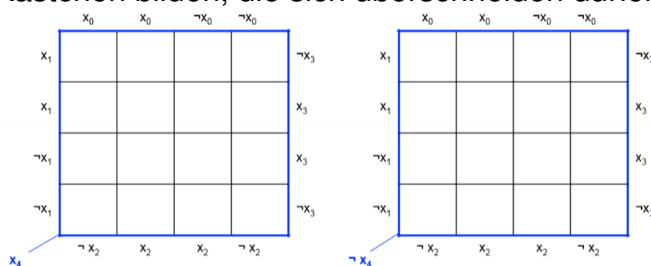
$\rightarrow$  Implikation

$\leftrightarrow$  Äquivalenz

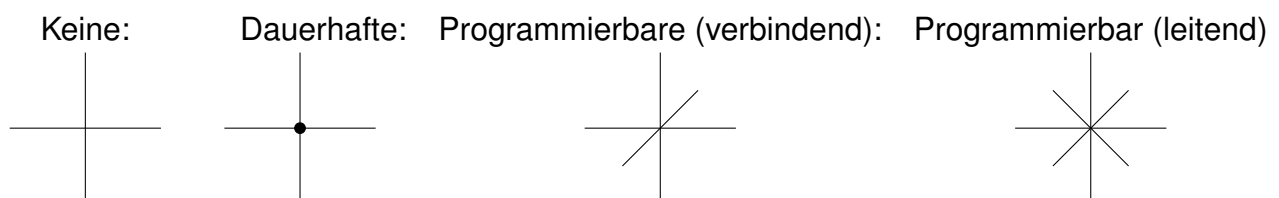
$\mapsto$  Zuweisung

#### 9.1.1 KV-Diagramm

Kästchen bilden, die sich überschneiden dürfen.



### 9.2 Verbindungen:



## 10 Bussystem

Hier muss noch Text rein

## 11 Interrupt

Die Interrupt-Funktion in Rechnerarchitekturen ermöglicht die Unterbrechung der normalen Programmausführung, um auf Ereignisse zu reagieren, die eine sofortige Aufmerksamkeit erfordern. Ein Interrupt kann beispielsweise durch eine externe Hardwarekomponente wie eine Tastatur oder ein Timer ausgelöst werden. Sobald ein Interrupt auftritt, unterbricht der Prozessor das aktuell ausgeführte Programm und springt zu einer spezifischen Interrupt-Service-Routine (ISR), die den Interrupt behandelt.

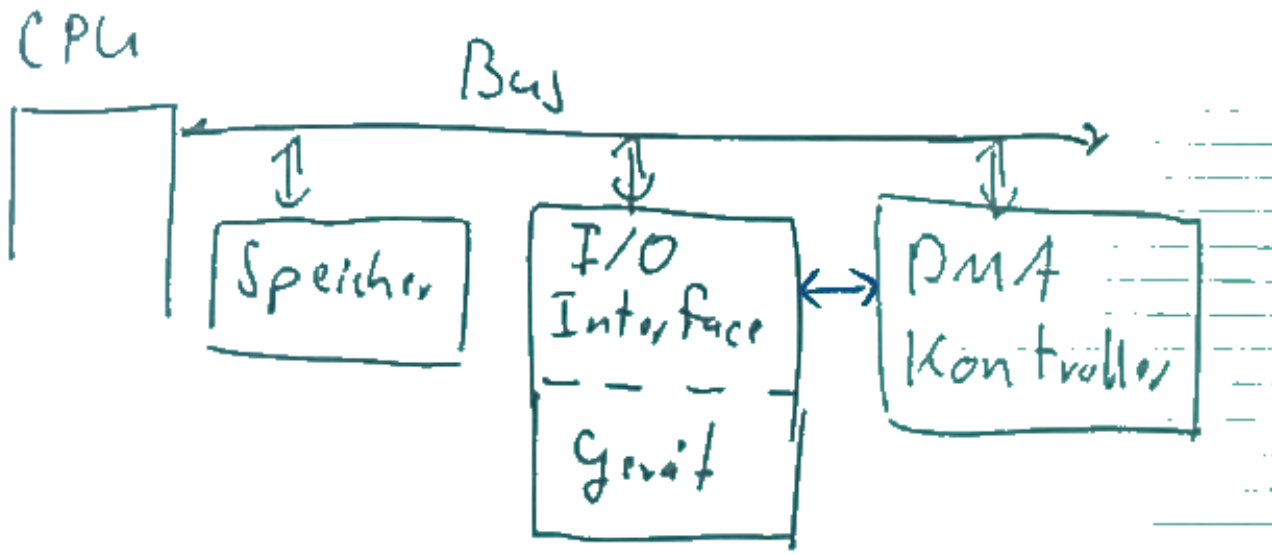
### 11.1 Interrupt Controller

Der Interrupt-Controller ist eine Hardwarekomponente, die für die Verwaltung von Interrupts zuständig ist. Er empfängt Interrupt-Signale von verschiedenen Quellen und priorisiert sie entsprechend ihrer Wichtigkeit. Der Interrupt-Controller weist jedem Interrupt eine Priorität zu und leitet den Interrupt an den Prozessor weiter, wenn dieser für die Behandlung bereit ist.

Der Interrupt-Controller spielt eine entscheidende Rolle bei der effizienten Verarbeitung von Interrupts in einem Computersystem. Er ermöglicht es, mehrere Interrupts gleichzeitig zu behandeln und sicherzustellen, dass die Interrupts entsprechend ihrer Priorität bearbeitet werden. Durch die Verwendung eines Interrupt-Controllers können Ressourcen effektiv genutzt und Konflikte zwischen verschiedenen Interrupts vermieden werden.

## 12 Direct Memory Access - DMA

Direct Memory Access (DMA) ist ein Verfahren in Computersystemen, das es bestimmten Geräten ermöglicht, Daten direkt zwischen dem Arbeitsspeicher (RAM) und ihren internen Speicherbereichen auszutauschen, ohne die direkte Beteiligung der CPU.



### Ablauf ohne DMA:

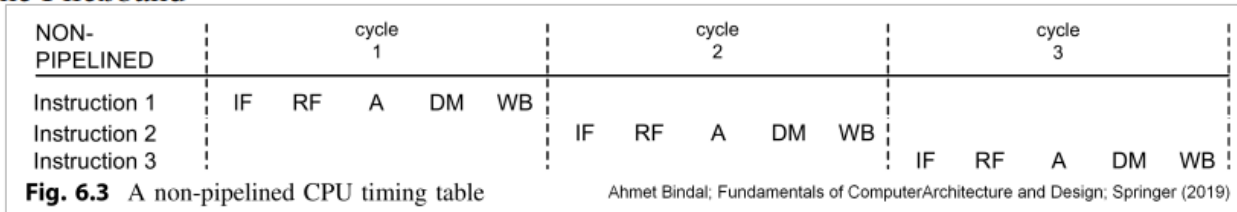
1. Quelladresse bestimmen/berechnen
2. CPU liest von Gerät
3. Zieladresse bestimmen
4. CPU schreibt in Speicher
5. Zähler erhöhen
6. Prüfen ob Alles übertragen wurde, ansonsten wiederhole

### Ablauf mit DMA:

1. Kontroller initialisieren
2. Transfer anstoßen(programmgesteuert/bedarfsgesteuert)
3. Übergabe (Kontroller übernimmt)
4. Transfer (Kontroller steuert(direkt, indirekt))
5. Kontroller gibt ab (CPU übernimmt Buskontrolle)

## 13 Fließband/Pipeline

### ohne Fließband



### Mit Fließband:

Takt	Stufe				
	CF	D	OF	E	W
1	Bef 1				
2	Bef 2	Bef 1			
3	Bef 3	Bef 2	Bef 1		
4	Bef 4	Bef 3	Bef 2	Bef 1	
5	Bef 5	Bef 4	Bef 3	Bef 2	Bef 1

## 14 Betriebssysteme

Schnittstelle zwischen der Anwendersoftware und der Speicher- und Prozessverwaltung.

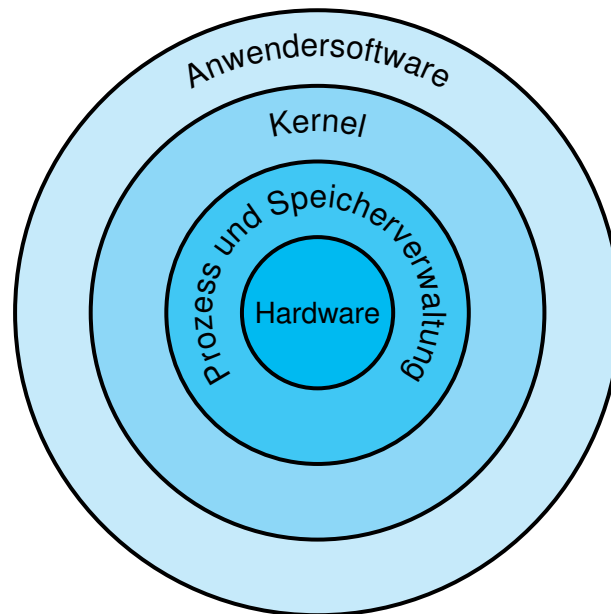


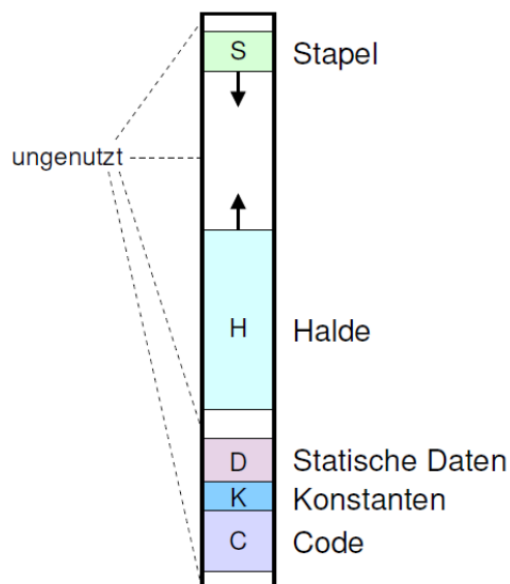
Abbildung 1: Aufbau eines Betriebssystems

## 14.1 Architekturen von Betriebssystemen

## 15 Speicheraufbau

**Speichermedien** nach Schnelligkeit sortiert:

Prozessorregister => Prozessorcachel => Hauptspeicher => Festplatte



**Stapel/Stack:** Stapelartige Datenstruktur für Variablen. Nur oberstes Glied kann gelesen oder hinzugefügt/gelöscht(push & pop) werden.

**Halde/Heap:** Auf Binärbaum basierte Datenstruktur. Dynamische Speicherung von Objekten. Schnell, Effiziente Speichernutzung, simple Verwaltung. Mit malloc Speicher allozieren/reservieren

**Statische Daten:**

**Konstanten:****Code:**

## 16 Speicherverwaltung und Dateisysteme

### 16.1 Speicherhierarchie

Speicher werden nach ihrer Schnelligkeit in eine Hierarchie eingeordnet, von kurzer zu langer Zugriffszeit.

1. Prozessorregister
2. Prozessorcaché
3. Hauptspeicher
4. Festplatte
5. Wechseldatenträger

### 16.2 Direkte Speicherverwaltung

Beim Einprogrammbetrieb verwaltet das Betriebssystem den Hauptspeicher als einen großen Speicherblock. Dieser besteht aus System und Benutzerbereich.

Betriebssystem	Programm	freier Speicher
----------------	----------	-----------------

*Problem:*

Die Programmgröße ist durch die Speichergröße begrenzt. Der Hauptspeicher ist somit zu klein für alle aktiven Prozesse.

*Lösung: Swapping*

### 16.3 Swapping

Beim Swapping wird ein Bereich der Festplatte als sogenannter 'Swap-Space' deklariert. In diesen können Prozesse ausgelagert werden. Daten und Programmcode des Prozesses werden vollständig in den Swap Space ausgelagert.

**Ein ausgelagerter Prozess kann nicht ausgeführt werden.**

**Verschiedene Strategien:**

**Not resently used**

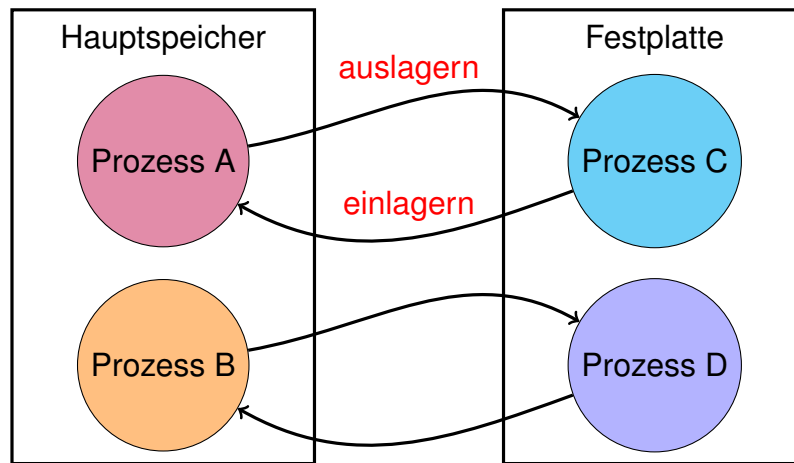
**First in, first out**

**Least frequently used**

Demnach wären hier nur Prozess A und Prozess B ausführbar.

Ein Prozess wird ausgelagert wenn:

- Ein Prozess, der neu entsteht nicht im Hauptspeicher untergebracht werden kann.
- Ein Prozess, seinen Hauptspeicherplatz erweitern möchte, jedoch kein zusätzlicher Speicher verfügbar ist.
- Ein *wichtigerer* Prozess aus dem Swap-Space eingelagert werden muss und für ihn kein Hauptspeicherbereich verfügbar ist.



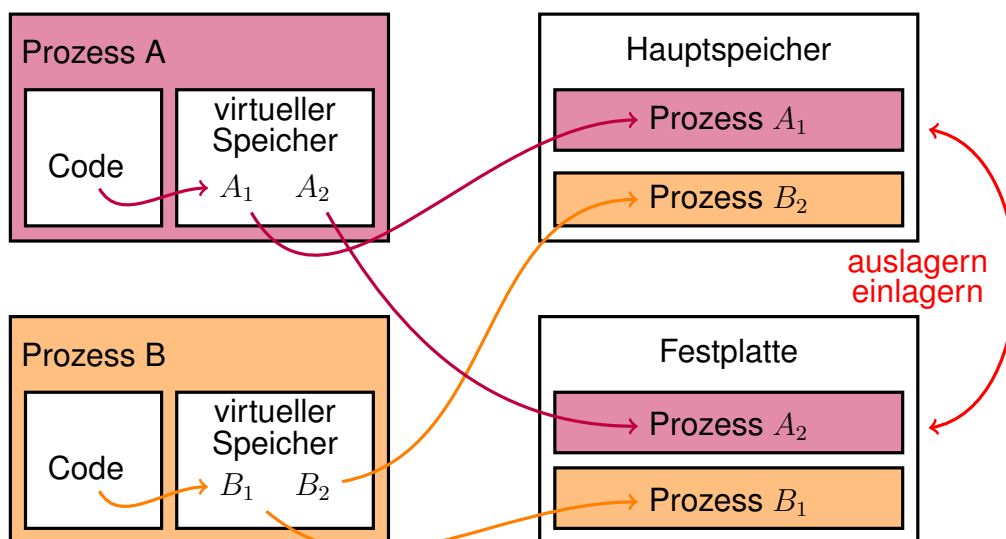
## 16.4 virtueller Speicher

Festplattenspeicher und Hauptspeicher werden zu einem Speicherblock vereint. Dadurch stehen mehr Speicheradressen zur Verfügung, wie tatsächlich im Hauptspeicher vorhanden sind = virtueller Speicher.

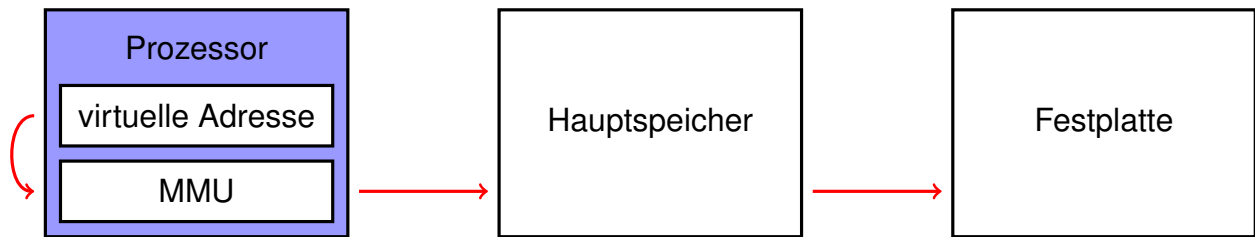
Im Gegensatz zum Swapping können beim virtuellen Speicher Prozesse auch nur teilweise im Hauptspeicher stehen, während der Rest meist auf dem Festplattenspeicher liegt.

Jedem Prozess wird ein virtueller Speicher geboten, in den alles außer arbeitenden Code und Daten (bleiben im Hauptspeicher) eingelagert wird. Wird von einem Prozess auf den Speicher zugegriffen, verweist die virtuelle Adresse auf eine reelle Adresse des Hauptspeichers. Dieser Verweis geschieht durch die *MMU (Memory Management Unit)*. Befinden sich die Daten in einem ausgelagerten Festplattenblock, wird dieser in den Hauptspeicher eingelagert.

Durch den Einsatz eines Virtuellen Speichers ergeben sich folgende Vorteile:

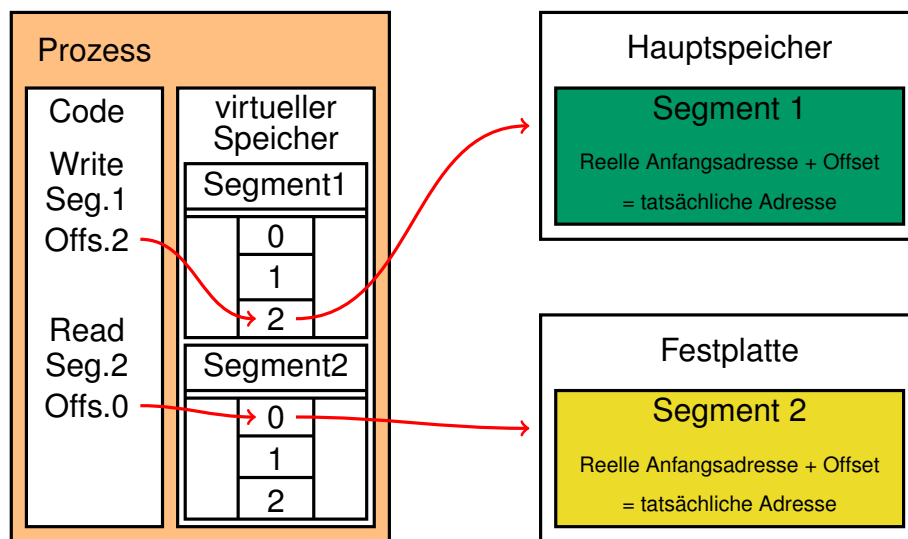


- Programmbereiche und Datenbereiche sind in ihrer Länge nicht durch die reelle Größe des Hauptspeichers begrenzt.
- Es können gleichzeitig mehrere Programme ausgeführt werden, deren Gesamtlänge die Hauptspeichergöße überschreitet.



## 16.5 segmentorientierter Speicher

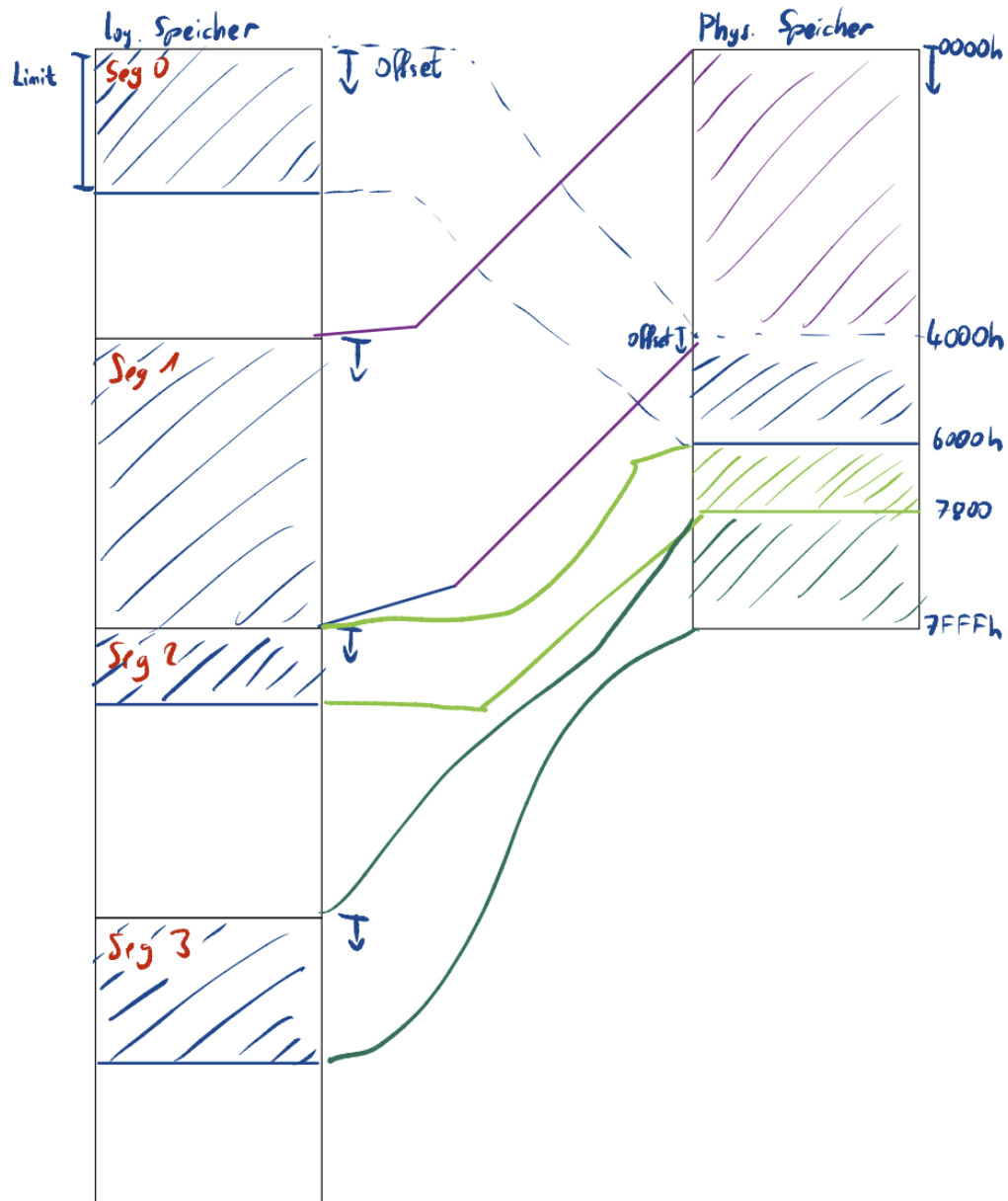
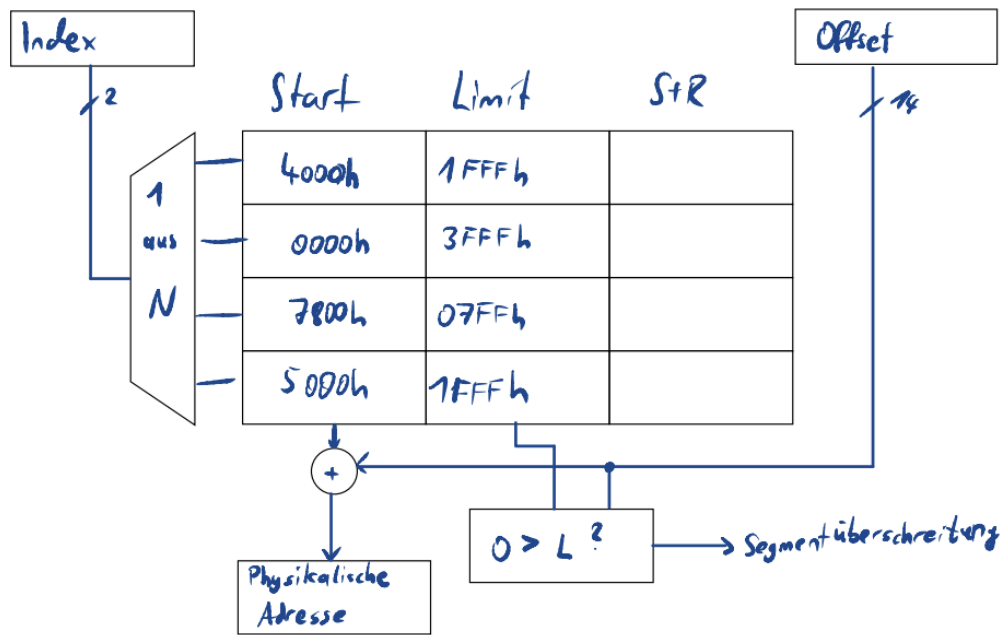
Der Adressraum wird in Segmente variabler Größe entsprechend der der logischen Einheiten des Programms unterteilt. Jedem Segment ist ein Speicherbereich im Haupt oder Festplattenspeicher zugeteilt. Die Virtuelle Adresse besteht aus zwei Teilen, der Segmentnummer und dem Offset. Dieser gibt die Position innerhalb des Segments an.



Der Prozess gibt mit der virtuellen Adresse an, auf welches seiner Segmente und auf welche Position innerhalb dieses Segmentes er zugreifen möchte. Die Umrechnung von virtuellen in reelle Adressen geschieht mit der Segmenttabelle. Jeder Prozess besitzt eine eigene Tabelle. Die Segmenttabelle eines Prozesses enthält für jedes Segment einen Eintrag mit folgenden Informationen:

- Reelle Anfangsadresse des Segments
- Valid-Bit (gibt an, ob sich das Segment auf dem Hauptspeicher oder auf der Festplatte befindet)
- Länge des Segments
- Zugriffsrechte



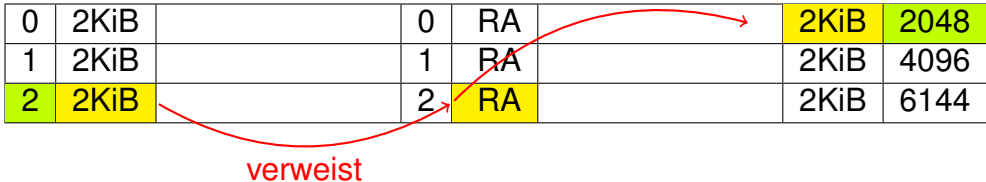


## 16.6 seitenorientierter Speicher - Paging

Strategie zur Speichereinteilung  
 Speicher wird in gleich große Blöcke/Seiten geteilt  
 Seitentabelle übersetzt virtuelle in reelle Adressen.  
 Erlaubt es Memory dynamisch zu reservieren.

virtuelle Adresse			Seitentabelle			reelle Adresse	
0	2KiB		0	RA		2KiB	2048
1	2KiB		1	RA		2KiB	4096
2	2KiB		2	RA		2KiB	6144

verweist



## 16.7 Demand Paging

Hybrid aus Paging und Swapping.  
 Nicht alle Teile des Prozesses müssen nicht im Memory sein.  
 Selten genutzte Pages werden ausgelagert.

## 16.8 MMU

Die Memory Management Unit übersetzt virtuelle Adressen in Physikalische.  
 Kontrolliert Speicherzugriffe.  
 Jeder Prozessor hat eigene MMU.  
 Für Betriebssysteme mit virtueller Speicherverwaltung (Paging)

## 16.9 Backups

Physikalisch getrennte Kopie aktueller Daten.  
 Hilfreich bei Datenverlust durch zum Beispiel einen System-Virus.  
 Hier wird auf ein Backup zugegriffen, welches vor dem Angriff gemacht wurde.

## 16.10 RAID

RAID  $\neq$  Backup  
 Für Ausfallsicherheit gibt es verschiedene RAID Systeme, um Daten ausfallsicher zur Verfügung zu stellen.

**RAID 0:** Speicherung der Daten auf 2 Festplatten für schnellere Zugriffszeiten

**RAID 1:** Redundante Speicherung der Daten auf 2 Festplatten für Ausfallsicherheit

**RAID 5:** Speicherung der Daten mit Paritätsbits auf mehreren Festplatten für Kompromiss aus Ausfallsicherheit und verbesserten Zugriffszeiten

**RAID 10:** Beide Systeme(1 & 0) vereinen

**RAID 50:** Beide Systeme(5 & 0) vereinen

...

## 16.11 FAT

Die FAT (*File Allocation Table, Dateizuordnungstabelle*) stellt ein Inhaltsverzeichnis der Festplatte dar und ist notwendig, um die auf dem Datenträger bereitstehenden Daten zu finden. In einem 16-Bit System (FAT 16) sind genau  $2^{16} = 65536$  Adressenverwaltungseinträge in der FAT möglich. Bei einer Clustergröße von 512 Byte beträgt die maximale Partitionsgröße 32 MiB ( $2^{16} \cdot 512$  Byte).

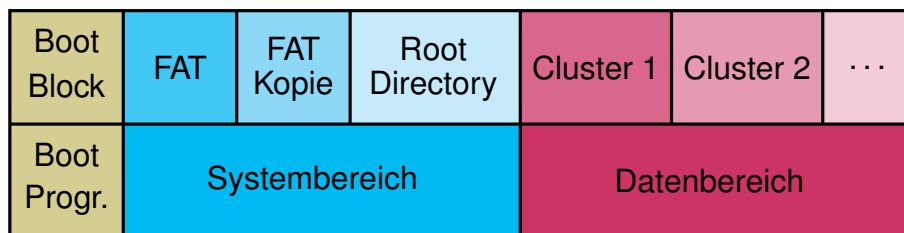


Abbildung 2: Aufbau des FAT Dateisystems

Durch die geringe Clustergröße können Dateien meist nur fragmentiert abgelegt werden. In einer Dateizuordnungstabelle werden Dateipfade gespeichert und können ausgelesen werden. Das richtige Zusammensetzen der Fragmente ist so möglich.

Ein Nachteil des FAT Systemes liegt darin, dass die FAT an einer festen Position auf dem Datenträger gespeichert ist. Bei Festplatten (Speichermedien mit rotierender Disk und Lesekopf) muss der Lesekopf bei jedem Lese/Schreib Vorgang zur FAT und zurück bewegt werden. Dann muss gewartet werden, bis die FAT den Lesekopf passiert. Das kostet viel Zeit. Heutzutage findet man das FAT System in der Flash Drive Technik.

## 16.12 NTFS

Das NTFS (*New Technology File System*) Dateisystem wurde erstmals mit Windows NT (*New Technology*) eingeführt und ist seither das preferierte Dateisystem von Windows. NTFS verwaltet Dateien anders als bei FAT in der MFT (*Master File Table*). Diese enthält die sogenannten Records. Zu jeder abgelegten Datei existiert also ein Record in der MFT der folgende Informationen enthält: Header, Informationen (z.B. Erstellungsdatum), den Dateinamen, Zugriffsrechte und die eigentlichen Daten bzw. die Position der Daten im Datenbereich. Dateien kleiner 1.5 KiB können direkt in den Record in der MFT gespeichert werden, wohingegen Dateien größer 1.5 KiB in den Datenbereich des Datenträgers gespeichert werden. Der Record verweist dann auf die entsprechende Stelle. Der Datenbereich ist bei NTFS variabel. Daraus resultieren kürzere Lese/Schreib Zeiten. Außerdem ist die MFT zwei mal zwischen den Datenbereichen enthalten.

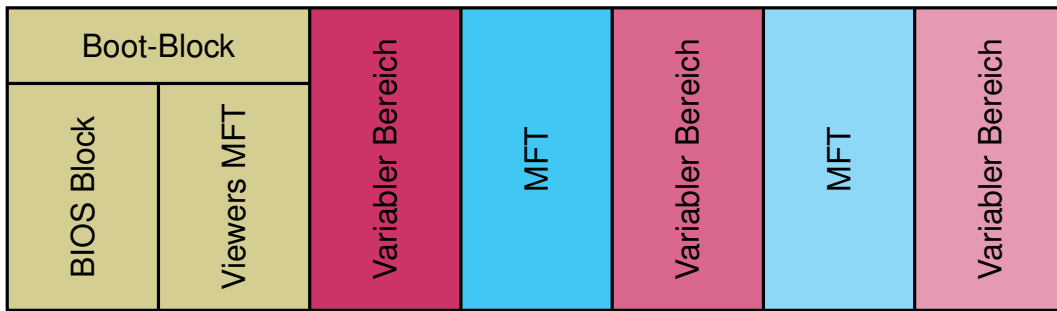


Abbildung 3: Aufbau einer NTFS Partition

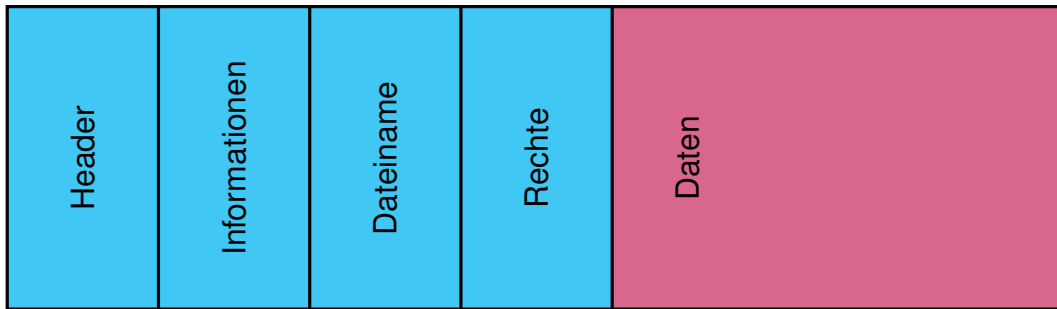


Abbildung 4: NTFS Record Dateien kleiner 1.5KiB

NTFS verfügt über eine hohe Fehlertoleranz. Eine geschriebene Datei wird mit der sich noch im RAM befindender Original Datei verglichen, um Fehler zu vermeiden. Wird ein Fehler erkannt, wird der betreffende Block als fehlerhaft markiert. Der Schreibvorgang wird dann an einem Anderen versucht. Dieses Prinzip nennt man Hot-Fixing. NTFS verfügt zudem über eine sogenannte LOGFILE. Alle Transaktionen werden in diese eingetragen. So kann nach einem Absturz verlorenes Datengut wiederhergestellt werden. NTFS Dateinamen können bis zu 255 Unicode Zeichen lang sein.

### 16.12.1 Dataruns

Ist ein Dateneintrag bei NTFS zu groß, muss dieser in den Datenbereich ausgelagert werden, liegt eine sehr große Datei vor, so muss diese zusätzlich fragmentiert werden. Diese Fragmentierung wird mit Hilfe des B-Tree Prinzipes in den Datenbereich des Records eingetragen. Mittles LCN-VCN Mapping ist es nun möglich die fragmentierte Datei auf dem Datenträger richtig und vollständig zusammen zu setzen.

## 16.13 Extended Filesystem

Speicherplatz wird in Blöcke zerlegt. Auf diese Blöcke zeigen Inodes. Inodes speichern auch die Attribute und Zugriffsrechte. Alle Informationen zum Dateisystem stehen in einem Superblock. Es gibt mehrere Kopien der Superblöcke.

## 17 Kernel

Der Kernel ist die Schnittstelle zwischen der Anwendersoftware und der Speicher- und Prozessverwaltung. Er kann auch als Betriebssystem-Kern bezeichnet werden.

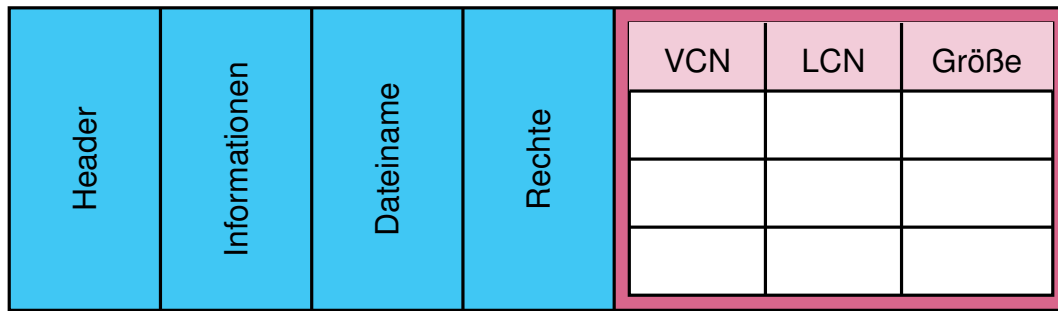


Abbildung 5: NTFS Record Dateien größer 1.5KiB

## 17.1 Threading

**Multi** Bearbeitet mehrere Threads simultan. Ein Prozess kann mehrere Threads enthalten.

**Hyper** Unterteilt 1 Prozessor in 2 virtuelle Prozessoren.

## 17.2 Kernel

### 17.2.1 Monolithischer Kernel

- + sehr schnell, da alles im Kernel geschieht und es kaum Schnittstellen hat
- wenn eine Komponente defekt ist oder einen Fehler hat, ist der Kernel nichtmehr funktionsfähig
- Updates müssen im Gesamten direkt erfolgen(z.B. Treiberupdates)

Der Monolitische Kernel wird bei Betriebssystemen wie z.B. DOS, Linux und Android verwendet.

### 17.2.2 Mirkokernel

- + wenn eine Komponente defekt ist oder einen Fehler hat, kann man den Kernel trotzdem weiterhin verwenden
- Prozesse dauern länger, da auch Prozesse/Treiber ausgelagert sind

Der Mirkokernel wird bei Betriebssystemen wie z.B. Echtzeitsbetriebssystemen oder RISC OS verwendet.

## 17.3 Geschichtetes System

## 17.4 Modulares System

### 17.4.1 Hybridkernel

- + Nutzung beider Vorteile
- Nutzung beider Nachteile

Der Hybridkernel wird bei Betriebssystemen wie z.B. Windows oder MAC OS verwendet.

## 18 Booten

Mehrstufiges Starten des Computers. Prüfen ob Hardware passt. OS laden.

**Kaltstart** Bei Reset oder Stromausfall. Alles laden und prüfen.

**Warmstart** Normal. Hardware wird nicht erneut geprüft.

### 18.1 Master Boot Record (MBR)

Der Master Boot Record (kurz MBR) enthält ein Startprogramm für BIOS-basierte Computer und eine Partitionstabelle. Außerdem werden durch den MBR die Partitionen einer Festplatte eingeteilt. Zudem kann man die Betriebssysteme und Dateisysteme der einzelnen Partitionen herauslesen.

Erster Sektor (512 Byte) auf Festplatte.

Gibt an wie und wo Betriebssystem gespeichert ist, um es in den RAM zu laden/booten (Startprogramm).

Sektor besteht aus: Bootloader  $\Rightarrow$  Disk-Signatur  $\Rightarrow$  Nullbytes  $\Rightarrow$  Partitionstabelle  $\Rightarrow$  Magic Number (55 AA)

Tabelle 1: Partitionstabelleneinträge

Adresse	Inhalt
00	80hex = bootfähige Partition 00hex = nicht bootfähige Partition
01	CHS-Eintrag des ersten Sektors
04	Typ der Partition (Dateisystem/Partitionsytp)
05	CHS-Eintrag des letzten Sektors
08	Startsektor nach LBA
0C	Anzahl der Sektoren in der Partition

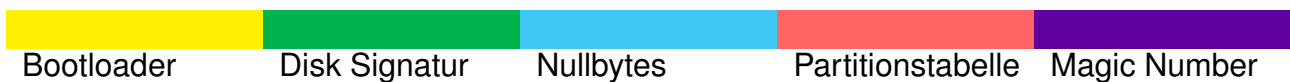
### 18.2 GPT

Die Guide Partition Table ist der bessere MBR und hat folgende Vor- und Nachteile gegenüber dem MBR:

- + Funktioniert auch bei 64-Bit Systemen
- + beliebig viele primäre Partitionen
- + verfügt über Schutz- und Sicherheitsmerkmale(Backups & Prüfsummen)
- Nur mit UEFI kompatibel

Tabelle 2: Master Boot Record

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	eb	48	90	10	8e	d0	bc	00	b0	b8	00	00	8e	d8	8e	c0
0010	fb	be	00	7c	bf	00	06	b9	00	02	f3	a4	ea	21	06	00
...																
01a0	10	ac	3c	00	75	f4	c3	00	00	00	00	00	00	00	00	00
01b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	80	01
01c0	01	00	83	fe	ff	ff	3f	00	00	00	41	29	54	02	00	fe
01d0	ff	ff	82	fe	ff	ff	80	29	54	02	fa	e7	1d	00	00	fe
01e0	ff	ff	83	fe	ff	ff	7a	11	72	02	fa	e7	1d	00	00	fe
01f0	ff	ff	05	fe	ff	ff	74	f9	8f	02	0c	83	6c	04	55	aa



## 18.3 UEFI und BIOS

UEFI ist das modernere und bessere BIOS. UEFI verfügt gegenüber BIOS folgende Vorteile:

- schnelleres Starten durch paralleles Laden der Treiber
- Datenträger können auch mit mehr als 2TB booten
- Funktioniert auch bei modernen 64-Bit Systemen
- Netzwerke können auch gebootet werden

## 19 UNIX

Mehrbenutzerbetriebssystem.

Arbeitet nach "Everything is a file" ⇒ Alle Daten liegen als files for.

## 20 Prozessverwaltung

Die Betriebsmittel des Computersystems müssen zwischen den verschiedenen laufenden Programmen und Systemaufgaben verteilt werden. Zu diesem Zweck werden die einzelnen Aufgaben als Prozesse ausgeführt, die vom Betriebssystem verwaltet werden.

### 20.1 Prozesse

Ein Prozess besteht aus dem Programmcode und dem Prozesskontext. Der Prozesskontext wiederum setzt sich zusammen aus den Registerinhalten des Prozessors, Speicherbereichen für die Daten und weiteren Betriebsmitteln.

Ein Programm kann sich aus mehreren Prozessen (Windows: Tasks) zusammensetzen. Ein Prozess kann Unterprozesse besitzen. So ist hier B ein Unterprozess von A, D E und F wiederum sind Unterprozesse von B.

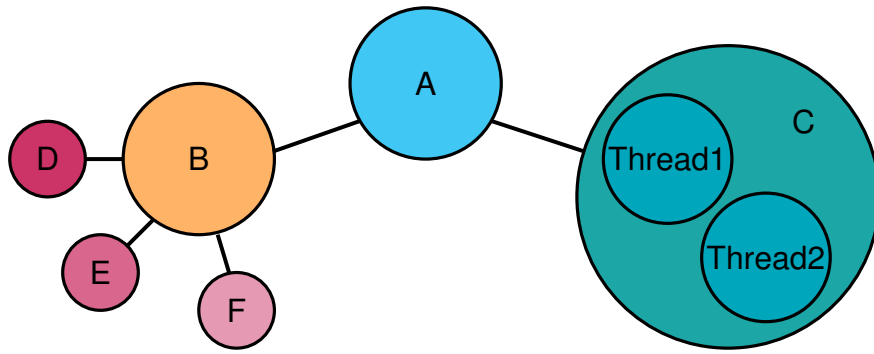


Abbildung 6: Prozesse, Unterprozesse und Threads

### 20.1.1 Threads

Prozesse können neben eigenständigen Unterprozessen auch noch Threads haben. Threads sind leichtgewichtige Prozesse innerhalb eines übergeordneten Prozesses. Threads teilen sich mit anderen Threads verschiedene Betriebsmittel. Threads, die dem selben Prozess zugeordnet sind, verwenden den gleichen Adressraum. Dadurch ist eine Kommunikation zwischen diesen Threads möglich.

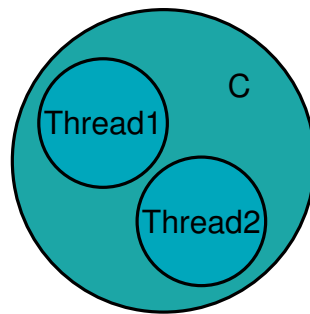


Abbildung 7: Threads in Prozess C

**Multi** Bearbeitet mehrere Threads simultan. Ein Prozess kann mehrere Threads enthalten.

**Hyper** Unterteilt 1 Prozessor in 2 virtuelle Prozessoren.

### 20.1.2 Prozesszustände

Ein Prozess kann folgende Zustände annehmen:

## 20.2 Prozesszustände

1. Rechnend
2. Blockiert
3. Bereit
4. Gestartet
5. Beendet (Suspendiert)



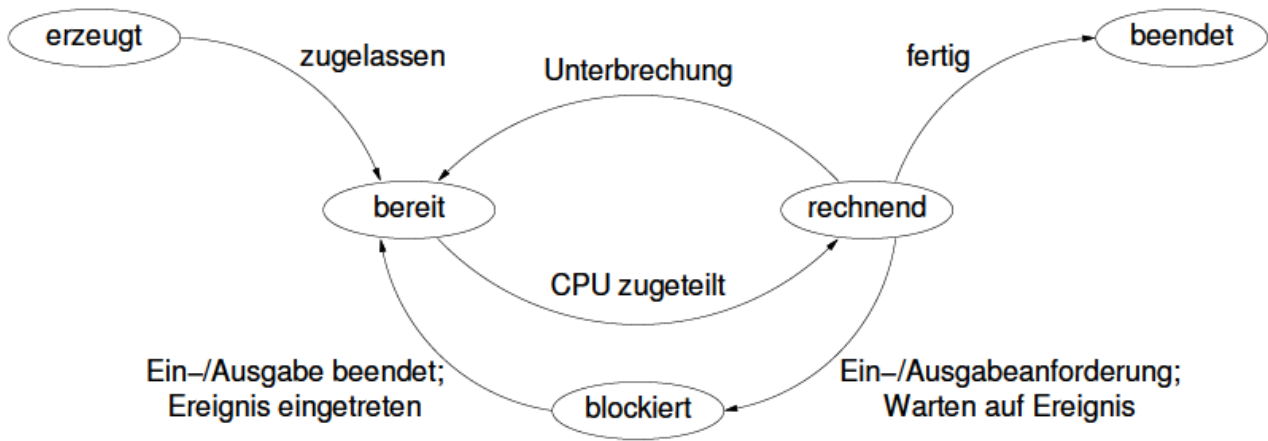
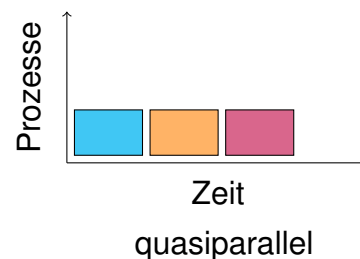
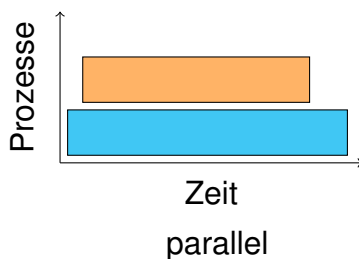


Abbildung 8: Prozessmodell

## 20.3 Mehrprozessbetrieb

Der Kern eines Prozessors kann immer nur einen Prozess bearbeiten. Prozesse nutzen die zur Verfügung stehende Rechenzeit allerdings nicht optimal, da viele Prozesse auf langsamere äußere Einflüsse warten müssen (bsp. Tastatureingabe). Die Wartezeit eines Prozesses kann effizient genutzt werden, indem diese Wartezeit einem anderen Prozess zugeteilt wird. Man unterscheidet hierbei zwischen zwei Prinzipien:

1. parallel, wenn jeder Kern einer CPU genau einen Prozess bearbeitet
2. quasiparallel, wenn ein Kern einer CPU mehrere Prozesse nacheinander bearbeitet.



## 20.4 Multitasking

Es wird zwischen zwei Arten von Multitasking unterschieden:

1. Kooperatives Multitasking, hier ist es jedem Prozess selbst überlassen, wann er die Kontrolle an das Betriebssystem zurück gibt. Geringer Verwaltungsaufwand, aber es besteht die Gefahr, dass ein unkooperativer Prozess das System blockiert.
2. Präemptives Multitasking, hier steuert das Betriebssystem die Vergabe der Rechenzeit. Der Scheduler kann einem Prozess Zeit oder Ereignis gesteuert Rechenzeit entziehen. Somit ist die Bearbeitung wichtigerer Prozesse zu jedem Zeitpunkt möglich, ein unkooperativer Prozess legt das System nicht lahm.

### 20.4.1 Kontext

Um einen Prozess unterbrechen zu können, muss dieser nach der Unterbrechung dieselbe Umgebung vorfinden können, welche der Prozess hatte, bevor er unterbrochen wurde.

Der Zustand des Prozesses wird also vor der Unterbrechung gespeichert, und bleibt das so lange, bis ihm wieder Rechenzeit zugeteilt wird. Unmittelbar bevor er wieder aktiv wird, wird der gespeicherte Zustand geladen. Aus Sicht des Prozesses scheint es so, als wäre er nie angehalten worden.

Wichtig für dieses Verhalten ist, dass Prozesse abgeschirmt voneinander sein müssen. Prozess A darf den Kontext von Prozess B nicht kennen. Prozess A erfährt immer nur seinen eigenen Kontext.

## 20.5 Scheduling

Konkurrieren mehrere Prozesse um die Rechenzeit, teilt der Scheduler ihnen gemäß folgender Kriterien die Rechenzeit zu:

1. **Fairness**, jeder Prozess erhält einen gerechten Anteil der Rechenzeit
2. **Effizienz**, der Prozessor arbeitet möglichst effizient
3. **Antwortzeit**, diese wird für interaktiv arbeitende Benutzer minimiert
4. **Verweilzeit**, die Wartezeit für die Ausgabe von Stapelaufträgen wird minimiert
5. **Durchsatz**, Maximierung der Auftragszahl für ein bestimmtes Zeitintervall

## 20.6 Prozess Scheduling Strategien

Ziel ist möglichst gute Fairness, Effizienz, Antwortzeit, Verweilzeit, Durchsatz.

## 20.7 Scheduling Strategien

### Präemptiv vs. Nicht-Präemptiv

**Präemptiv:** Betriebssystem verwaltet die Prozesse. => Betriebssystem kann dem Prozess Ressourcen bereits vor der Fertigstellung wieder entziehen.

**Nicht-Präemptiv:** Prozess kann selbst entscheiden wann er Ressourcen freigibt.

### 20.7.1 First Come First Serve

Dem Prozess, der als erstes Betriebsmittel und Rechenzeit anfordert, werden diese zugeteilt.

—> nicht präemptiv

### 20.7.2 Shortest remaining time

Prozesse, welche am schnellsten beendet werden können, werden zuerst bearbeitet.

—> präemptiv

### 20.7.3 Round Robin

Jedem Prozess wird gleich viel Rechenzeit in Form von Zeitscheiben zugeteilt. Ist die Zeit abgelaufen, wird der Prozess in den Prozesszustand *bereit* gesetzt, der nächste Prozess wird aufgerufen. Nachdem ein Prozess bearbeitet wurde, und nicht vollständig abgearbeitet werden konnte, wird dieser hinten in der Warteschlange eingereiht, und muss warten, bis er erneut an der Reihe ist. Ein Prozess, der die zugeteilte Rechenzeit nicht in vollem Maße ausschöpfen kann, weil er schon früher fertig ist, blockiert das ganze System, bis die Zeitscheibe abgelaufen ist. Eine effiziente Nutzung der Ressourcen ist daher nicht möglich.

→ präemptiv

### 20.7.4 Priority

Es wird zwischen zwei Priority verfahren unterschieden:

1. **Präemptiv/ Highest Priority First**

Ein *bereit* stehender Prozess höherer Priorität verdrängt einen *rechnenden* Prozess, geringer Priorität.

2. **Nicht Präemptiv**

Prozess mit bester/ Priorität wird zuerst abgearbeitet

Bei Prozessen mit gleicher Priorität erfolgt die Abarbeitung nach zum Beispiel Round Robin.

## 21 Virtualisierung

Erstellen einer virtuellen Umgebung.

Unabhängig von tatsächlicher Hardware (solange Hardwareleistung nicht überschritten wird).

### 21.1 Hypervisor

Schicht zwischen Betriebssystemen und Hardware. Möglichkeit mehrere Gastsysteme auf einem Hostsystem zu verwalten/hosten. Aufgaben: VMs erstellen und ausführen, Ressourcen flexibel verwalten, Abgrenzung von VM-Hardware für Sicherheit.

**Typ1-Hypervisor:** Bare Metall Hypervisor. Es muss ein physischer Hypervisor gekauft werden.

**Typ2-Hypervisor:** Auf Softwareebene. Ist als Software für Rechner zu erhalten.

## 22 Powershell

### Funktionen

```
function addieren ($z1, $z2){
    return $z1 + $z2
}
function subtrahieren ($z1, $z2){
    return $z1 - $z2
}
function multiplizieren ($z1, $z2){
    return $z1 * $z2
}
function dividieren ($z1, $z2){
    return $z1 / $z2
}

$input = Read-Host "What do you want to do? addieren(0), subtrahieren(1), m

$inputnumber1 = Read-Host "Geben sie ihre erste Zahl ein"
$inputnumber2 = Read-Host "Geben sie ihre zweite Zahl ein"

switch($input){
    0 {$ergebnis = addieren $inputnumber1 $inputnumber2}
    1 {$ergebnis = subtrahieren $inputnumber1 $inputnumber2}
    2 {$ergebnis = multiplizieren $inputnumber1 $inputnumber2}
    3 {$ergebnis = dividieren $inputnumber1 $inputnumber2}
}
Write-Host $ergebnis
```

## Externe Klassen