

# Betriebssysteme

Rechnerarchitekturen und Betriebssysteme Teil 2

Im Studiengang 5CS, Informatik

Referent: Hendrik Siegmund

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - Speicherverwaltung
  - Dateisysteme
  - Ein- und Ausgabe
  - Multiprozessorsysteme
  - Virtualisierung

Praxis: Umgang mit Linux (Prof. Brunner) und Windows Server 2019

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Das Prozessmodell

- Mit dem Prozessmodell stellt das Betriebssystem eine Abstraktion zentraler PC-Ressourcen zur Nutzung durch Programme bereit
- Ein **Prozess** ist die **laufende Instanz eines Programms** mit allen zur Ausführung benötigten Daten, Befehlen und sonstigen Informationen
- Ein alternativer Begriff für Prozess ist **Task**
- Der Prozess kann als **Umgebung für eine Aktivität** verstanden werden, **der** vom Betriebssystem **CPU-Zeit** zur Ausführung **zugeteilt wird**

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Das Prozessmodell

- Ziel des Prozessmodells ist es, das „gefühlt“ gleichzeitige Ausführen von Programmen in einer einzelnen CPU zu ermöglichen:
- **Multitasking, Multiprocessing oder Multiprogramming**

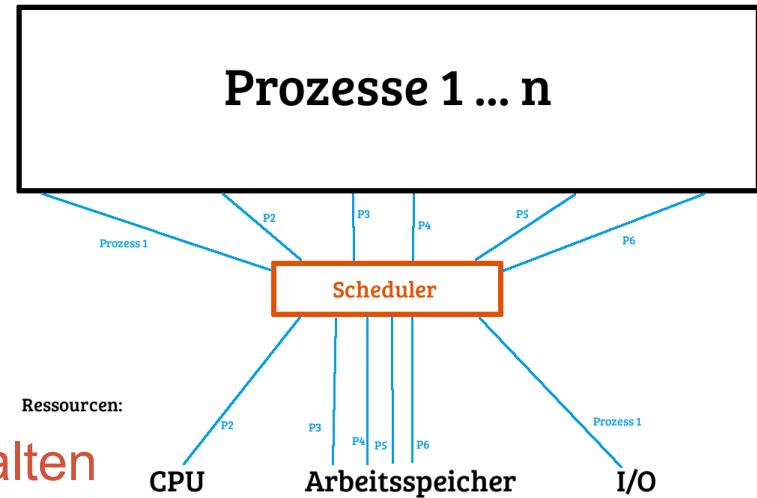
Name	8% CPU	50% Arbeitsspe...
Apps (4)		
➤ Microsoft Outlook (32 Bit)	0,1%	42,8 MB
➤ Microsoft PowerPoint (32 Bit)	0,2%	102,4 MB
➤ Task-Manager	2,3%	19,6 MB
➤ Windows-Explorer	0,1%	25,1 MB

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Das Prozessmodell

- Sämtliche auf einem Rechner ausführbare Software wird als **Menge sequenzieller Prozesse** verstanden
- Das Betriebssystem hat die Aufgabe, die CPU-Zeit gerecht, effizient und zuverlässig an die konkurrierenden Prozesse zu verteilen
- Diese Steuerungsaufgabe übernimmt ein **Scheduler** und erledigt sie nach betriebssystemabhängig verschiedenen Regeln
- Der Scheduler nutzt dazu eine **Prozesstabelle** (Process Control Block) mit allen relevanten Metadaten zu den Prozessen



# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Das Prozessmodell

- Der Scheduler nutzt die Prozesstabellen, um den Prozessen CPU-Zeit und damit verbundene Ressourcen zuzuteilen

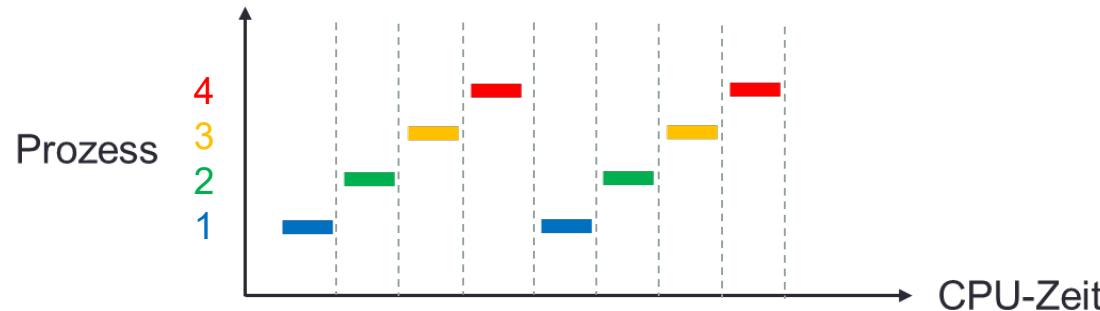
Name	PID ^	Benutzername...	CPU	Arbeitsspeicher...
Systemunterbrechung	-	SYSTEM	01	0 K
Leerlaufprozess	0	SYSTEM	80	8 K
System	4	SYSTEM	00	20 K
CCXProcess.exe	68	m5000035	00	324 K
node.exe	76	m5000035	00	4.144 K
Registry	96	SYSTEM	00	9.008 K
svchost.exe	344		00	14.208 K
smss.exe	424		00	152 K
fontdrvhost.exe	484		00	240 K
csrss.exe	660		00	688 K
dptf_helper.exe	720	m5000035	00	400 K
wininit.exe	796		00	360 K
services.exe	868		00	3.040 K
lsass.exe	880		00	6.028 K
svchost.exe	888		00	2.312 K
svchost.exe	924		00	8.232 K
svchost.exe	944		00	1.128 K
svchost.exe	1016		00	240 K
svchost.exe	1060		00	1.180 K
SearchIndexer.exe	1072		00	13.352 K
audiogd.exe	1128		00	4.068 K
RuntimeBroker.exe	1188	m5000035	00	548 K
svchost.exe	1240		00	816 K
svchost.exe	1280		00	2.180 K

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Das Prozessmodell

- Die CPU-Zeit kann auf mehrere Prozesse verteilt werden, indem diese (in schneller Folge) **abwechselnd ausgeführt** werden



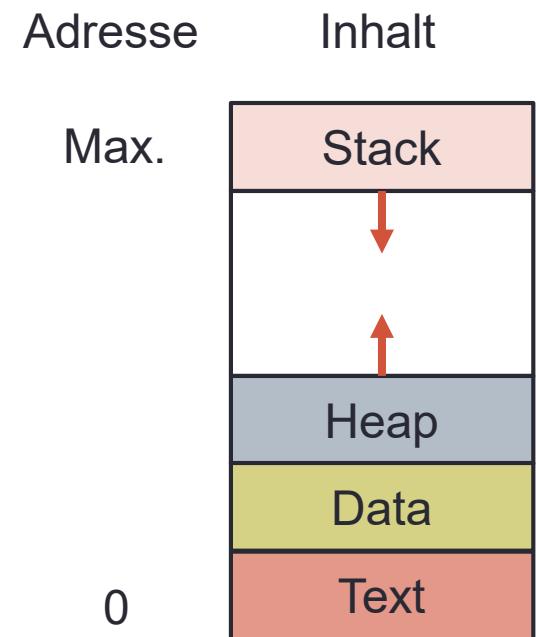
- Beim Wechsel zwischen den Prozessen werden **Prozessbestandteile** gesichert und für die Wiederaufnahme bereitgehalten

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozessbestandteile

- Prozesse besitzen ihren **privaten** Adressraum im Arbeitsspeicher mit folgenden Inhalten:
  - **Programmcode** (Text)
  - **Daten** (Variablen)
  - **Heap** als frei zuweisbarer Speicher
  - **Stack** (LIFO-Speicher, z.B. für Unterprogramme)

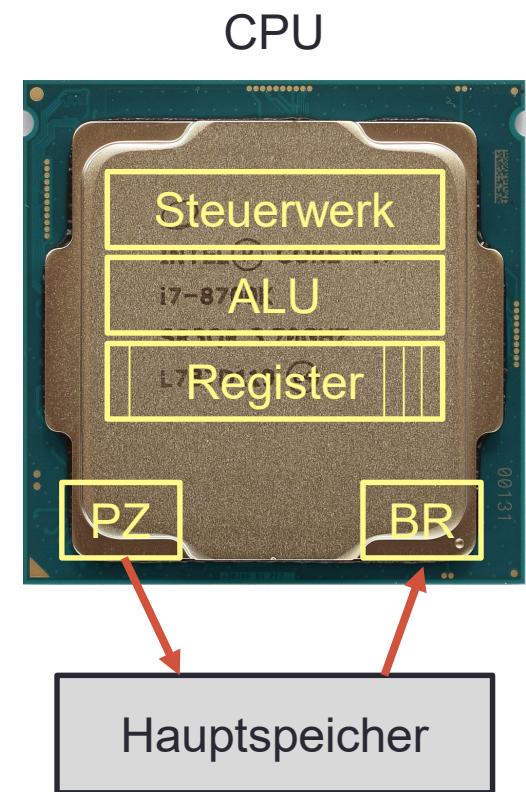


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozessbestandteile

- Zum Prozess gehören außerdem:
- Der Wert des **Befehls-** oder **Programmzählers** (PZ) der CPU
- Inhalt des **Befehlsregisters** (BR) der CPU
- Inhalte der **Prozesstabellen** des Betriebssystems wie Prozess-ID, User-ID, Sicherheitsattribute, Prioritäten usw.



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesserzeugung

- Prozesse müssen erzeugt, verwaltet und beendet werden
- Neue Prozesse werden aus folgenden Anlässen erzeugt:
  1. Systemstart
  2. Anforderung durch einen anderen Prozess
  3. Anforderung durch einen Benutzer
  4. Auf Großrechnern durch das Betriebssystem zur Ausführung eines neuen Stapelverarbeitungs-Jobs



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesserzeugung

- Technisch wird ein Prozess stets durch einen **Systemaufruf** erzeugt
- Unter UNIX durch **Fork**
- Unter Windows durch **CreateProcess**
- Beim Systemaufruf werden alle erforderlichen Parameter gesammelt bzw. generiert und angegeben

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesserzeugung – Prozesstypen

- Abhängig vom Anlass der Erzeugung eines Prozesses und dessen Aufgabe unterscheiden Betriebssysteme zwischen Prozessen ohne und mit Benutzerinteraktion
- Prozesse ohne Benutzerinteraktion werden oft automatisch gestartet und laufen vom Benutzer unbemerkt
- Solche Hintergrundprozesse heißen unter UNIX **Deamons** und unter Windows **Dienste**
- Zur Einsparung von Ressourcen können Hintergrundprozesse auch erst dann gestartet werden, wenn ein anderer Prozess sie benötigt

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesserzeugung – Prozesstypen

- Dienste oder Hintergrundprozesse können unter Windows mit zwei Werkzeugen verwaltet werden
- **Task Manager**
- **Konsole Dienste**
- Dienste ist nur für Administratoren zugänglich

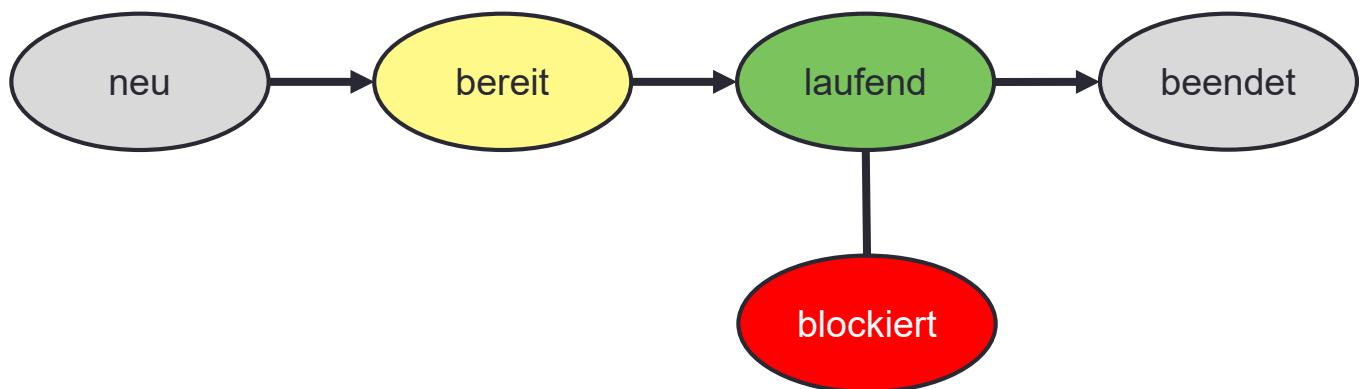
Prozesse	Leistung	App-Verlauf	Autostart	Benutzer	Details	Dienste
Name	11% CPU	58% Arbeitsspe...				
<b>Apps (5)</b>						
›  Firefox (3)	0,1%	226,4 MB				
›  Microsoft PowerPoint (32 Bit)	0,3%	61,7 MB				
›  Paint	0,2%	9,7 MB				
›  Task-Manager	2,5%	26,2 MB				
›  Windows-Explorer	0,7%	34,6 MB				
<b>Hintergrundprozesse (92)</b>						
›  AcroTray (32 Bit)	0%	0,9 MB				
›  Adobe Acrobat Update Service	0%	0,4 MB				
›  Adobe CEF Helper	0%	5,9 MB				
›  Adobe CEF Helper	0%	14,7 MB				
›  Adobe CEF Helper	0,1%	27,4 MB				
›  Adobe CEF Helper	0%	5,0 MB				

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesszustände

- Nach ihrer Erzeugung können sich Prozesse in einem von fünf Zuständen befinden:
  - Neu
  - Bereit
  - Laufend
  - Blockiert
  - Beendet



# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesszustände

- Prozesse können sich in einem von fünf Zuständen befinden:

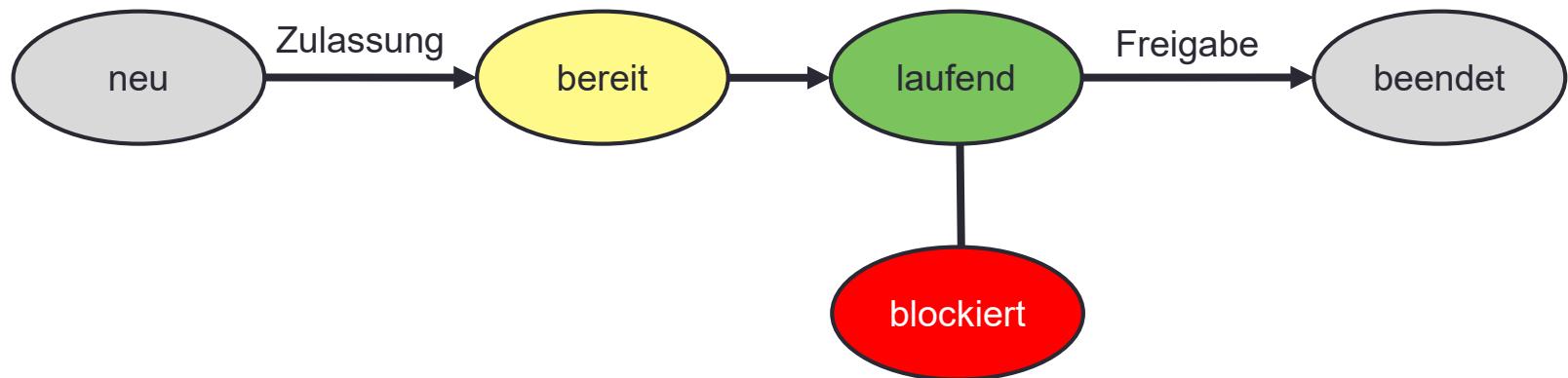
Zustand	Beschreibung	Englisch
Neu	Ein neuer Prozess ist fertig vorbereitet und wartet auf Zulassung	new
Bereit	Der Prozess ist zur Ausführung fertig	ready
Laufend	Der Prozess läuft, er wird durch die CPU ausgeführt	running
Blockiert	Der Prozess wartet auf ein Ereignis, etwa eine Eingabe oder das Ergebnis der Ausführung eines anderen Prozesses	blocked, waiting
Beendet	Der Prozess ist beendet, die wesentlichen Ressourcen sind wieder freigegeben	terminated

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesszustände

- Neue Prozesse müssen zur Bearbeitung erst zugelassen werden
- Beendete Prozesse nutzen keine Ressourcen mehr



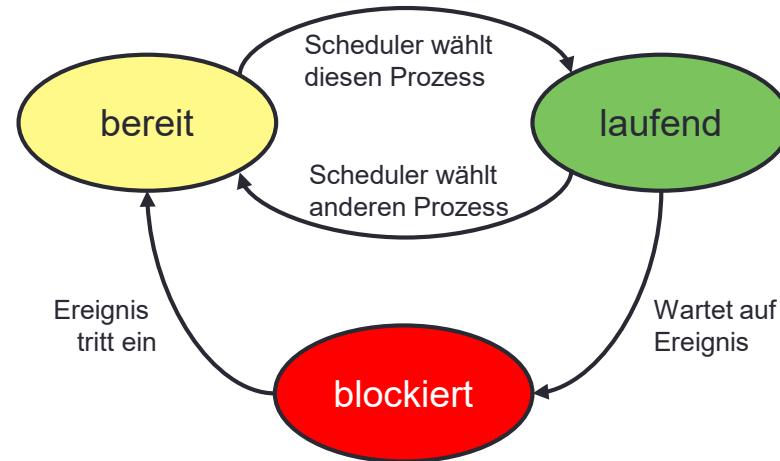
- Nur bereite, laufende und blockierte Prozesse werden hier betrachtet

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesszustände

- Prozesszustände können ineinander übergehen
- Die Übergänge sind definiert und werden vom Scheduler gesteuert

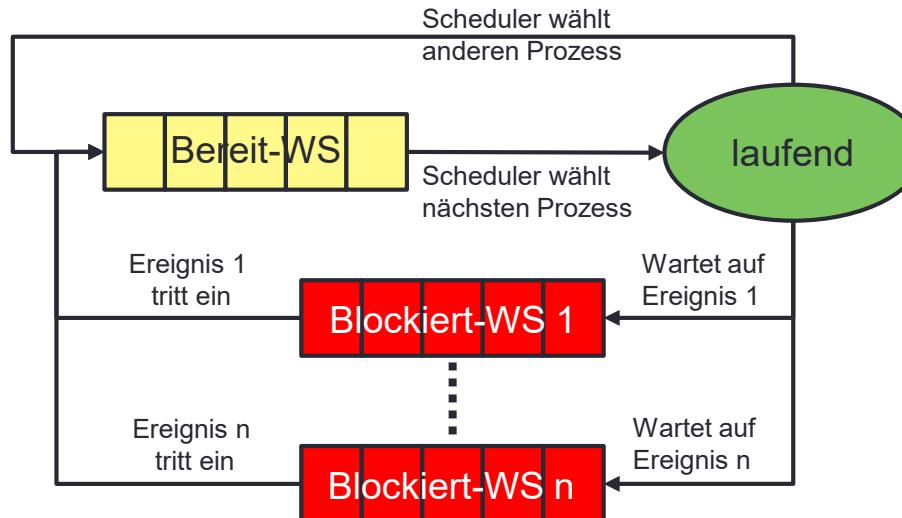


# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesszustände

- Normalerweise ist eine Vielzahl von Prozessen gleichzeitig aktiv
- Der Scheduler arbeitet deshalb mit Warteschlangen für jeden Zustand



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesszustände wechseln

- Der Zustand von Prozessen kann aus verschiedenen Gründen verändert werden:
- Der **Scheduler aktiviert** nach Ablauf des Zeitfensters eines laufenden Prozesses **einen anderen Prozess**
- Das **Programm ruft explizit einen anderen Prozess auf** (Supervisor Call)
- **Trap**: Eine aktuelle Instruktion beendet den Prozess, z.B. ein Fehler
- Ein **Interrupt** geht ein, der Prozess muss beendet werden, um dem Interrupt Handler die CPU zur Verfügung stellen zu können

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesse beenden

- Prozesse enden unter bestimmten Bedingungen
- Die häufigsten Gründe für das Beenden eines Prozesses sind:
  - **Normales Beenden (freiwillig)**
  - **Beenden nach einem Fehler (freiwillig)**
  - **Beenden nach einem schwerwiegenden Fehler (unfreiwillig)**
  - **Beenden durch einen anderen Prozess (unfreiwillig)**

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesse beenden

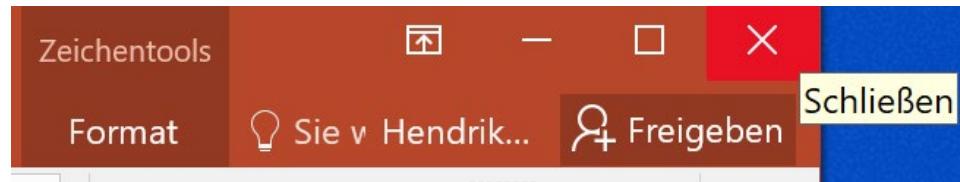
- Im UNIX werden Prozesse durch den Systemaufruf `exit` beendet
  - Unter Windows ist der entsprechende Systemaufruf `exitProcess`
  - Diese Systemaufrufe erzeugen die Prozesse im Normalfall selbst, um sich nach getaner Arbeit zu beenden
- 
- Das **Anhalten** eines Prozesse durch den Scheduler **ist kein Beenden**

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesse beenden

- Die meisten Prozesse beenden sich also selbst, wenn sie ihre Aufgabe erfüllt haben
- Prozesse mit Benutzerinteraktion können meist auch interaktiv durch den Benutzer beendet werden: In einer GUI durch Klicken auf das X



- Normales Beenden führt zur Löschung aller Einträge in der Prozesstabellen und zur Freigabe aller genutzten Ressourcen

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesse beenden

- Stellen Prozesse selbst einen Fehler fest, können sie sich unterschiedlich verhalten:
- Im vielen Fällen (nicht Benutzerfreundlich) beenden sie sich **ohne Rückmeldung**
- Benutzerfreundlicher sind Rückfragen in der Kommandozeile
- Prozesse mit grafischer Benutzerinteraktion können ein Dialogfenster öffnen und nach zusätzlichen Eingaben fragen oder den Benutzer eine Entscheidung zum Beenden treffen lassen

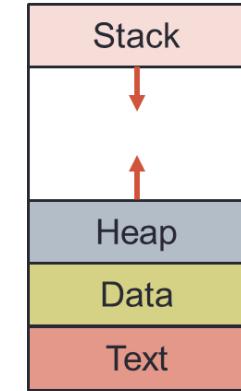
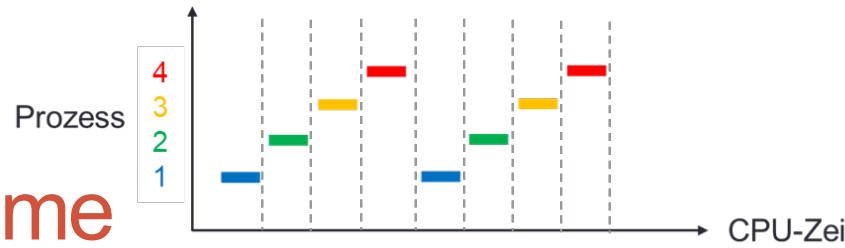
# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesse beenden

- Treten schwerwiegende Fehler auf, etwa wegen Programmierfehlern, kann ein Prozess mit gleichzeitiger Fehlermeldung unterbrochen werden
- Dann ist es möglicherweise erforderlich, den Prozess durch einen Systemaufruf zu beenden, unter UNIX `kill` und unter Windows `TerminateProcess`
- Diese Systemaufrufe werden zum Teil auch beim Herunterfahren des Rechners verwendet, es können Datenverluste auftreten

# Betriebssysteme



## Aufgaben – Prozesse und Threads verwalten

### Threads

- Prozesse bieten die Möglichkeit, mehrere Programme schon auf einer CPU quasi gleichzeitig auszuführen
- Jeder Prozess besitzt einen privaten Arbeitsspeicherbereich, der gegen Zugriff anderer Prozesse gekapselt ist
- Prozesse können deshalb zwar z.B. über eine **Pipe** miteinander kommunizieren, bei der die Ausgabedaten eines Prozesses als Eingabedaten an einen anderen Prozess weitergegeben werden, aber die gleichzeitige gemeinsame Datennutzung ist „**by Design**“ nicht möglich

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads

- Um dennoch gleichzeitigen mehreren Verarbeitungsvorgängen die gemeinsame Nutzung des Arbeitsspeichers eines Prozesses zu ermöglichen, wurde das Konzept der **Threads** entwickelt
- Threads wurden bereits im Einführungsteil der LV als **parallele Verarbeitungsstränge innerhalb eines Prozesses** (Fäden) definiert, die auf **gemeinsame Daten** des Prozesses zugreifen

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads

- Threads ermöglichen folglich einem Programm, das als Prozess in einem Rechner ausgeführt wird, gleichzeitig mehrere Operationen an denselben Daten auszuführen
- In den meisten Programmen bzw. deren Prozessen laufen im Hintergrund mehrere Threads

Details						Dienste
Name	PID	Benutzerna...	CPU	Arbeitsspeicher (ak...	Threads	
openvpnserve.exe	4504		00	0 K	2	
OUTLOOK.EXE	13792	m5000035	00	23.244 K	40	
POWERPNT.EXE	12256	m5000035	02	151.340 K	26	
ptim.exe	8852	m5000035	00	488 K	4	

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads

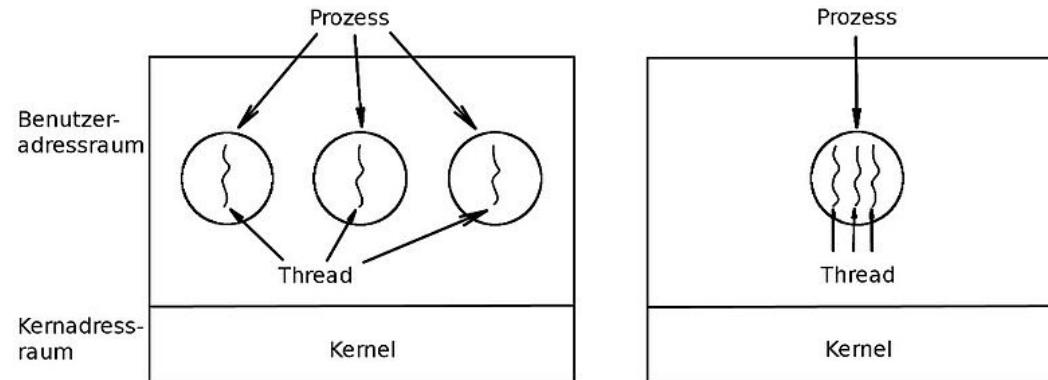
- Threads sind besonders in komplexen Programmen unentbehrlich für eine gute Benutzererfahrung:
- Ein Textverarbeitungsprogramm, das geöffnete Dokumente während der Bearbeitung regelmäßig automatisch auf die Festplatte sichert
- Ein E-Mail-Client, der parallel zum Editieren der E-Mails den Server auf neue Nachrichten prüft, Kalenderdaten synchronisiert und Instant Messaging- oder Newsdienste abfragt
- Da in diesen Fällen auf Daten des gleichen Prozesses zugegriffen wird, wäre diese Arbeitsweise mit Prozessen allein nicht möglich

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads

- Drei parallele Verarbeitungsvorgänge können als **drei Prozesse** oder **drei Threads in einem Prozess** ausgeführt werden
- Drei Threads in einem Prozess teilen sich dessen Adressraum



Quelle: Tanenbaum und Bos (2016), S. 147

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads

- Threads teilen sich neben dem Adressraum auch weitere Daten des übergeordneten Prozesses, u.a.:
  - Globale Variablen
  - Geöffnete Dateien
- Jedoch besitzen Threads auch eigene Informationen:
  - Stack
  - Befehlszähler
  - Register
  - Zustand

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Threads – Ebenen

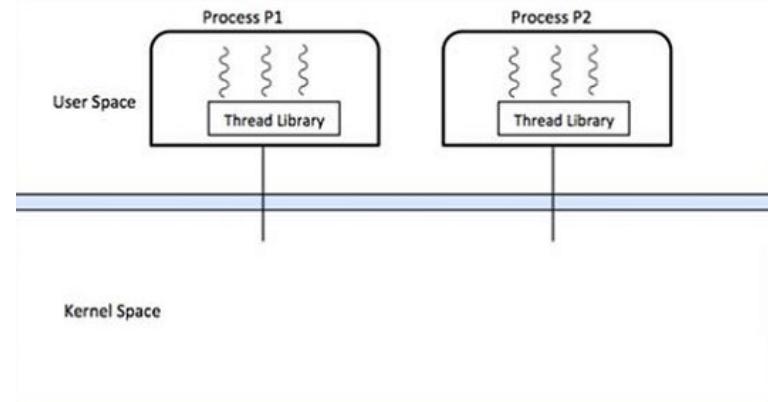
- Threads können prinzipiell entweder im Benutzer-Adressraum oder im Kernel-Adressraum laufen:
- **User-Level Threads ULT**
- **Kernel-level Threads KLT**

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Threads – Ebenen

- ULTs werden nur vom Prozess verwaltet, der dazu eine **Threads Library** besitzt: Code mit allen Verwaltungsfunktionen
- ULTs sind für den Kernel unsichtbar, der Kernel kooperiert nur mit dem zugehörigen Prozess



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Threads – Ebenen

- ULTs sind einfacher zu handhaben, aber es gibt den wesentlichen Nachteil, dass schon das **Blockieren eines Threads** dazu führt, dass **der ganze Prozess blockiert**
- KLTs können dagegen nicht mit einer Thread Library auf Prozessebene verwaltet werden, sondern diese Aufgaben muss der Kernel mit übernehmen
- Der Verwaltungsaufwand, die „Kosten“, ist dadurch deutlich höher, allerdings ergibt sich der Vorteil der Unabhängigkeit von Threads auf Kernelebene

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Threads – Vergleich mit Prozessen

- Aufgrund der Vergleichbarkeit mit Prozessen werden Threads auch als **leichtgewichtige Prozesse** oder **lightweight processes** bezeichnet
- Wie beim Prozess kann die CPU quasi gleichzeitig mehrere Threads ausführen: Multithreading
- Threads besitzen daher die von Prozessen bekannten Zustände



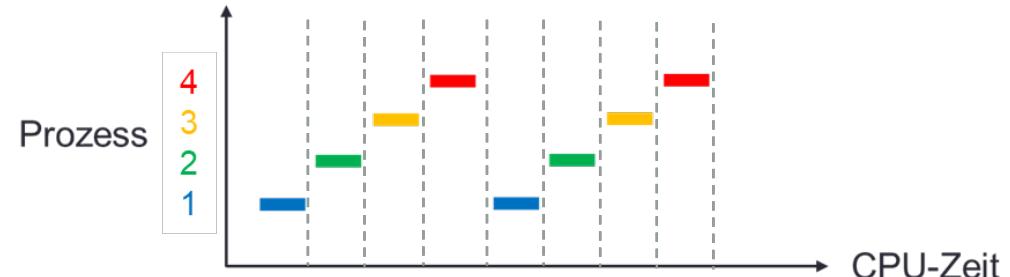
# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Threads – Vergleich mit Prozessen

- Da sich Threads viele Parameter mit ihrem Prozess teilen, kann der Prozess selbst seine Threads verwalten: Erzeugen, Umschalten, Beenden
- Das Umschalten zwischen Threads ist damit einfacher und schneller als zwischen Prozessen
- Allerdings müssen die Threads eines Prozesses untereinander synchronisiert werden um Fehler zu vermeiden
- Wird ein Prozess beendet, terminieren auch alle seine Threads

# Betriebssysteme



Aufgaben – Prozesse und Threads verwalten

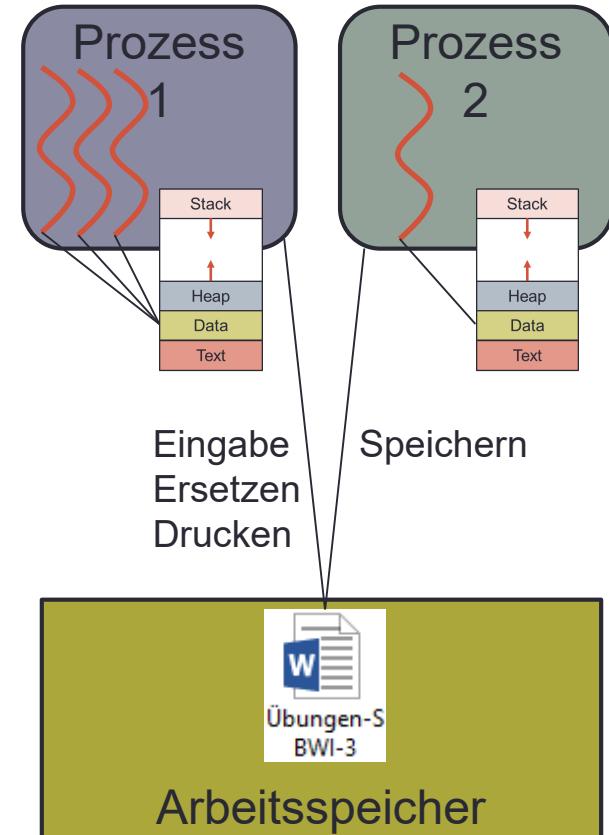
Interprozesskommunikation und Scheduling

- In einem Multitasking-Betriebssystem können mehrere Prozesse und Threads **nebenläufig** ausgeführt werden
- Solange diese Prozesse und Threads unabhängig voneinander sind und keine Ressourcen gemeinsam nutzen, entstehen bei der Unterbrechung durch den Scheduler keine Probleme
- Prozesse bemerken die Unterbrechung nicht und setzen die Arbeit stets da fort, wo sie zuvor unterbrochen wurden
- Das ändert sich jedoch, wenn Prozesse oder Threads gemeinsame Ressourcen benutzen müssen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten  
Interprozesskommunikation und Scheduling

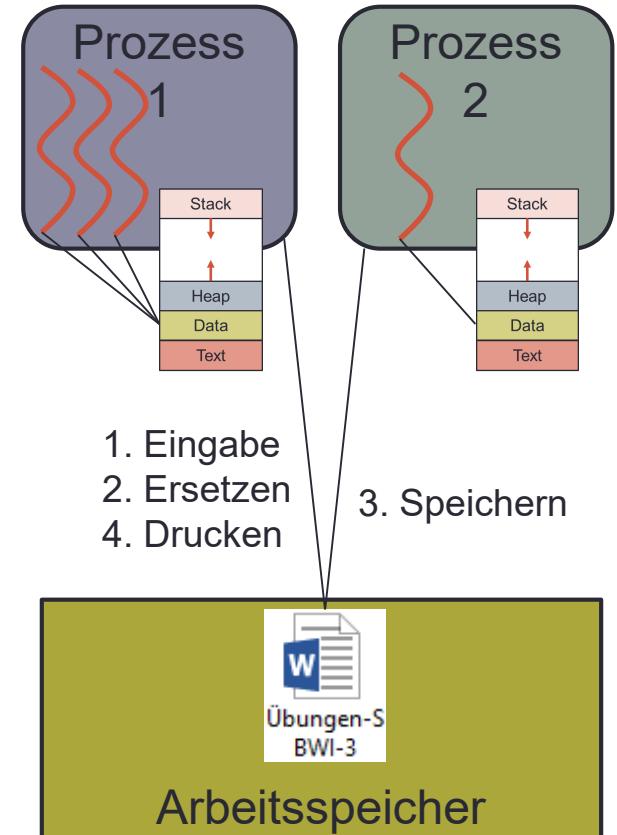
- In der Praxis gibt es dafür viele Beispiele:
- Threads greifen auf den gemeinsamen Speicher und **globale Variablen** ihres Prozesses zu
- Prozesse können zwar nicht den Speicher anderer Prozesse benutzen, aber mehrere Prozesse können sich **Daten in einem externen Speicherbereich** teilen



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten  
Interprozesskommunikation und Scheduling

- Bei der **Nebenläufigkeit** von Prozessen und Threads **mit gemeinsamen Daten** können Zustände eintreten, bei denen das Ergebnis der Verarbeitung von der zeitlichen Abfolge der Arbeitsschritte abhängt
- Solche Zustände werden als **Race Conditions**, **Wettlaufsituationen** oder **zeitkritische Abläufe** bezeichnet, die zu Fehlern führen und unbedingt vermieden werden müssen



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Race Conditions

- Bedingungen für das Auftreten von Race Conditions sind
- A: **Nebenläufige Prozesse** oder **Threads** werden ausgeführt
- B: Diese nutzen **gemeinsame Daten** wie z.B. globale Prozessvariablen

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Race Conditions

- Beispiel: Ein Java-Programm mit zwei unterschiedlich arbeitenden Threads, die eine gemeinsame globale Variable verwenden
- Der Scheduler kann zufällig genau dann zwischen den Threads wechseln, wenn der erste die Variable nach seiner Anweisung berechnet und gesetzt hat
- Ist der zweite Thread zufällig genausoweit, übernimmt nach dem Wechsel diesen Wert und gibt ihn aus, was jedoch ein falsches Ergebnis liefert
- <https://youtu.be/dIOg4Dz-bgM>

# Betriebssysteme

Aufgaben – Prozesse  
und Threads verwalten  
Race Conditions

```
1. public class Beispiel_Race_Conditions {
2.
3.     static int counter = 0;
4.
5.     public static class Counter_Thread_A extends Thread {
6.         public void run() {
7.             counter = 10;
8.             counter++;
9.             counter++;
10.            System.out.println("A-Counter: " + counter);
11.        }
12.    }
13.
14.    public static class Counter_Thread_B extends Thread {
15.        public void run() {
16.            counter = 20;
17.            counter++;
18.            counter++;
19.            counter++;
20.            counter++;
21.            counter++;
22.            counter++;
23.            System.out.println("B-Counter: " + counter);
24.        }
25.    }
26.
27.    public static void main(String[] args) {
28.        Thread a = new Counter_Thread_A();
29.        Thread b = new Counter_Thread_B();
30.        a.start();
31.        b.start();
32.    }
33.
34. }
```

Quelle: vfhcab.oncampus.de/loop/Race\_Conditions#Aufgabe\_1

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Race Conditions

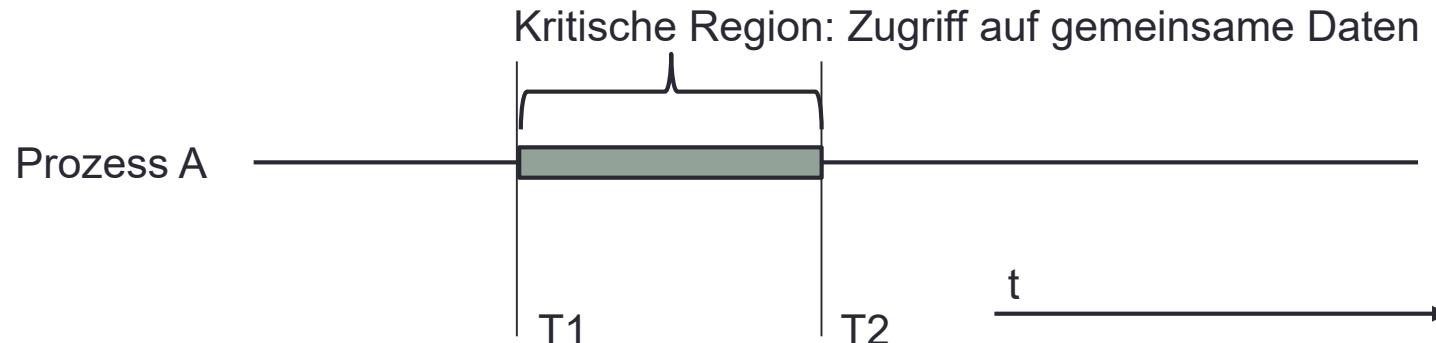
- Das Problem entstand, weil zwei **nebenläufige Threads** die **gleiche Variable genutzt und verändert** haben und der Scheduler davon unbeeindruckt nach eigenem Ermessen die Wechsel durchführt
- Um Race Conditions zu vermeiden, müssen sich die Threads oder Prozesse  **gegenseitig ausschließen** (mutual exclusion):
- **Während ein Prozess auf gemeinsame Daten zugreift, darf dies kein anderer tun, bis der erste seine Arbeit erledigt hat**
- Allerdings ist es nicht nötig, dass die Prozesse oder Threads **ihre gesamte Arbeit einstellen**...

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Es reicht, wenn sich Prozesse und Threads nur während eines Zugriffs auf gemeinsame Daten gegenseitig ausschließen
- Diese Zeiten heißen kritische Region oder kritischer Abschnitt (critical region, critical section)

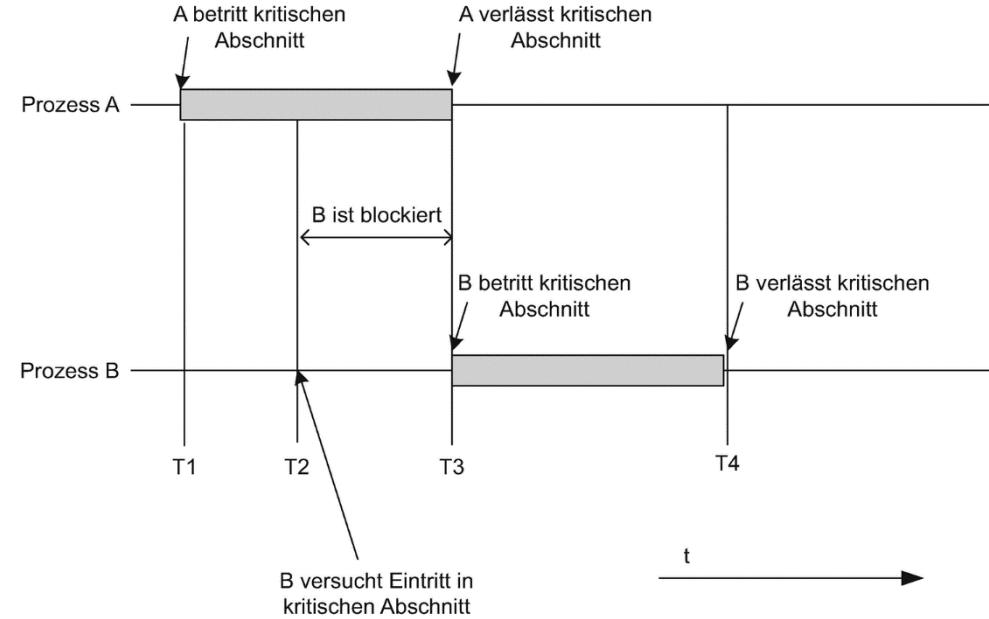


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Ein Prozess muss nur daran gehindert werden, **in seine kritische Region einzutreten, solange ein anderer Prozess noch in seiner eigenen kritischen Region arbeitet**
- Dann sind Race Conditions prinzipiell nicht möglich



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Dafür gibt es verschiedene Ansätze:
- Vorübergehend **alle Interrupts ausschalten**
- Synchronisation: Prozesse versuchen selbst, ihren Ablauf zeitlich abzustimmen: **Signalisieren**, z.B. mit Sperrvariablen und aktivem Warten oder TSL-Befehl
- Aktive Einbeziehung des Betriebssystems: **Semaphore**

Für die folgenden Überlegungen umfasst der Begriff Prozess zur Vereinfachung sowohl Prozesse als auch Threads, denn beide unterliegen den gleichen Regeln

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

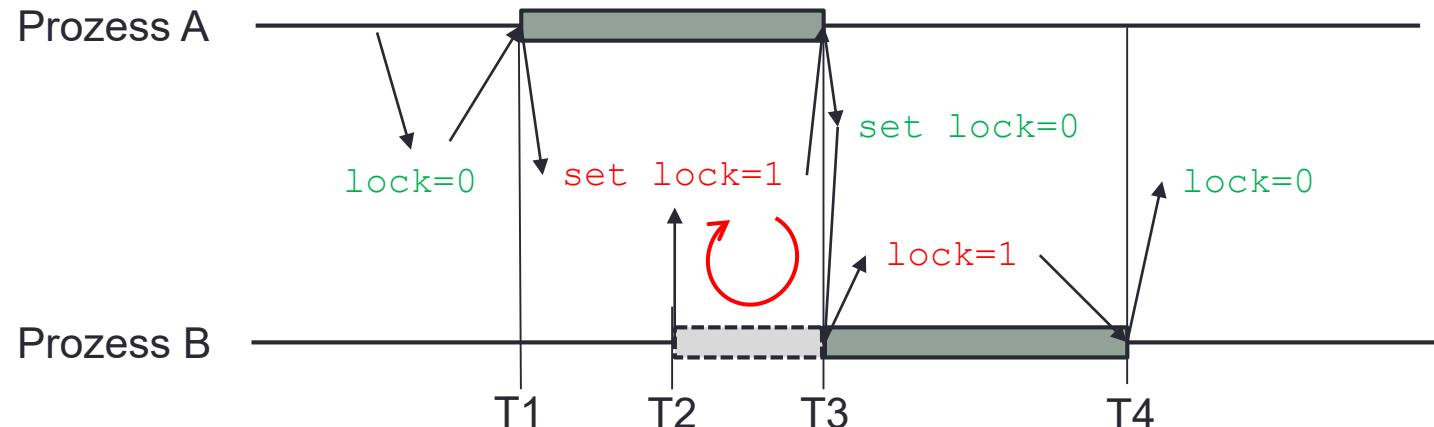
- Wird ein Scheduler angewiesen, alle Interrupts auszuschalten solange sich ein Prozess im kritischen Bereich befindet, kann damit eine Unterbrechung dieses Prozesses sicher verhindert werden
- **Aber:**
- Wenn der Prozess fehlerhaft ist und seine Arbeit nicht beendet, wird der Interrupt möglicherweise nie wieder eingeschaltet und der Scheduler wäre blockiert: Ende des Multitaskings
- Auf Multiprozessormaschinen könnte der Prozess auch auf mehrere CPUs verteilt sein, für die sicher nicht alle Interrupts deaktiviert sind

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Signalisieren, z.B. mit Sperrvariablen und aktivem Warten
- Gemeinsame Variable für alle Prozesse: `int lock;`
- Prozesse führen **Test, Set and Lock** aus: Sperre testen und setzen

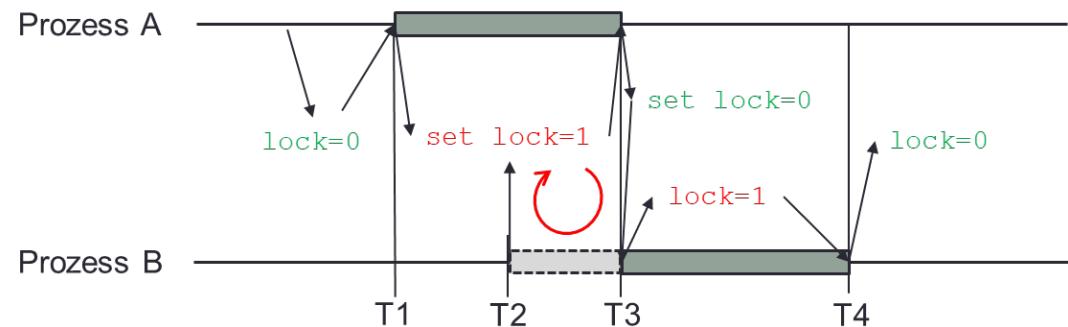


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Problem: Auch während der Operation Test, Set and Lock kann ein Prozess vom Scheduler abgeschaltet und die Werte der Variablen können überschrieben werden
- Außerdem verschwendet der Prozess in der Warteschleife CPU-Zeit



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Synchronisierung

- Alternativ kann diese Funktion aber auch die CPU durchführen
- **TSL-Maschinenbefehl** in der CPU: Das **Prüfen, Setzen und Sperren** wird nicht vom Prozess sondern auf Hardwareebene ausgeführt
- Auch der **XCHG-Befehl** in x86-CPUs kann zum Ändern von Variablen verwendet werden
- Beide Maschinenbefehle sind **atomar**, also **nicht unterbrechbar**
- Es bleibt aber dabei, das Prozesse aktiv warten müssen, um irgendwann in ihre kritische Region eintreten zu können
- Besser wäre eine Lösung mit externer Steuerung und ohne Warten

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore



- **Semaphor** bedeutet **Signalgeber**, abgeleitet von Formsignalen bei der Eisenbahn, in Analogie zur Steuerung der Streckenabschnitte
- Prozesse müssen sich nicht mehr selbst prüfen und benachrichtigen, sondern dies übernimmt eine außerhalb der Prozesse angeordnete Instanz, etwa der Scheduler
- Prozesse müssen nicht mehr aktiv warten, sondern blockieren, solange andere im kritischen Abstand arbeiten
- Das Konzept der Semaphore wurde 1965 von **Edsger W. Dijkstra** vorgestellt

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore

- Ein **Semaphor** ist eine Datenstruktur, die einen **Zähler** und eine **Warteschlange** enthält
- Der Zähler kann von allen Prozessen gelesen werden
- Die Datenstruktur darf nur durch zwei **atomare Operationen** verändert werden, die genau definiert sind: P (down) und V (up)
- **Atomar** bedeutet wieder **nicht unterbrechbar**
- Für den gegenseitigen Ausschluss von Prozessen wird ein **binärer Semaphor** verwendet: Es werden nur die Zustände 0 und 1 ausgewertet

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore

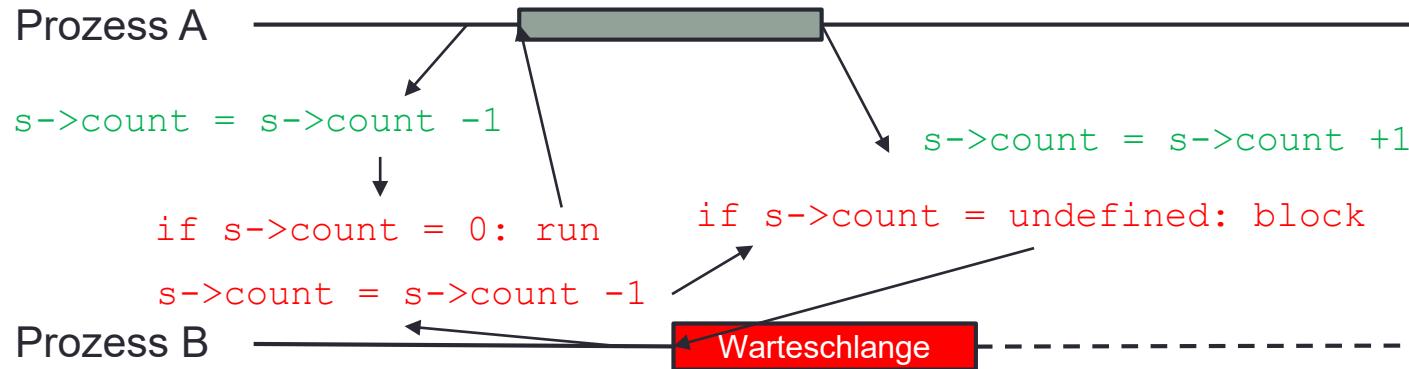
- Binäre Semaphoren oder **Mutex-Locks (mutual exclusion)** funktionieren wie folgt:
- Ein Prozess, der in seine kritische Region eintreten möchte, dekrementiert den Zähler `count` um 1 (Operation `down`)
- Ist der Zähler jetzt = 0, setzt der Prozess seine Arbeit fort
- Durch reduzieren auf 0 wird anderen Prozessen signalisiert, dass die gemeinsam genutzte Ressource (variable...) **gesperrt** ist: **Mutex lock**
- Ist der Zähler bereits 0, ist das Dekrementieren nicht möglich, der Prozess blockiert und wird in die Warteschlange eingereiht

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesssteuerung – Semaphore

- Am Ende der Arbeit in der kritischen Region wird der Zähler durch die Operation `up` wieder um 1 erhöht und damit die gemeinsame Ressource für andere Prozesse wieder freigegeben



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore

- Damit entfällt das aktive Warten und CPU-Zeit wird nicht mehr verschwendet
- Im Übrigen ist dies aber eigentlich auch nur eine Sperrvariable mit den gleichen Nachteilen...

Wo ist der entscheidende Unterschied?

- Die down- und up-Operationen werden nicht von den Prozessen selbst ausgeführt, sondern **als Systemaufrufe vom Betriebssystem**
- Das Betriebssystem kann sicherstellen, dass diese Operationen **atomar** sind, etwa indem es **währenddessen Interrupts deaktiviert**

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore

- Es wird keine CPU-Zeit mehr durch aktives Warten verschwendet
- Semaphore übertragen die Kontrolle der kritischen Abschnitte dem Betriebssystem, die Prozesse müssen nur noch Systemaufrufe starten:
- Beispielsweise unter Windows
  - `s1 = CreateSemaphore (... , Initialwert, „Name“)`
  - `s1 = OpenSemaphore (... , „Name“)`
  - **Down-Operation:** `WaitForSingleObject (s1,...)`
  - **Up-Operation:** `ReleaseSemaphore (s1,...)`

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Semaphore

- Es gibt mehrere Arten von Semaphoren:

Initialwert	Name	Beschreibung
N	Zählsemaphor	Verwendung bei mehrteiligen Variablen, etwa RAM-bereichen, löst Erzeuger-Verbraucher-Problem
1	Binärsemaphor	Lösung des Wechselseitigen Ausschlusses
0	Schlafen/Wecken	Aufrufender Prozess wird blockiert, bis er durch einen anderen Prozess wieder geweckt wird

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

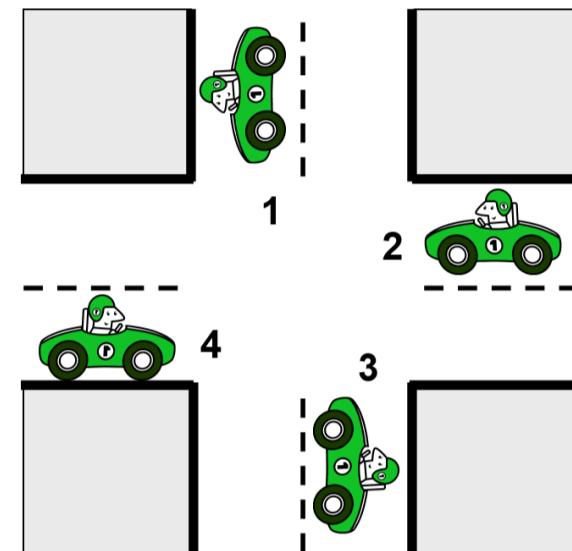
- Mit Semaphoren lassen sich Prozesse synchronisieren und das Auftreten von Race Conditions verhindern
- Allerdings ist gerade der exklusive Zugriff von Prozessen auf gemeinsam genutzte Ressourcen ein Faktor, der zu einem weiteren Problem führt: **Deadlocks** oder **Verklemmung**
- In Synchronisierten und geregelten Prozessabläufen kann die Situation eintreten, dass Prozesse sich gegenseitig Ressourcen entziehen und aufgrund der Regeln nicht wieder freigeben
- Dann kann ein Prozess mehr arbeiten, das System steht still

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

- Kreuzung ohne Vorfahrtsregel und vier Fahrzeuge
- Jedes Fahrzeug muss einem anderen die Vorfahrt gewähren, kein Fahrzeug darf fahren



Quelle: [ais.informatik.uni-freiburg.de](http://ais.informatik.uni-freiburg.de)

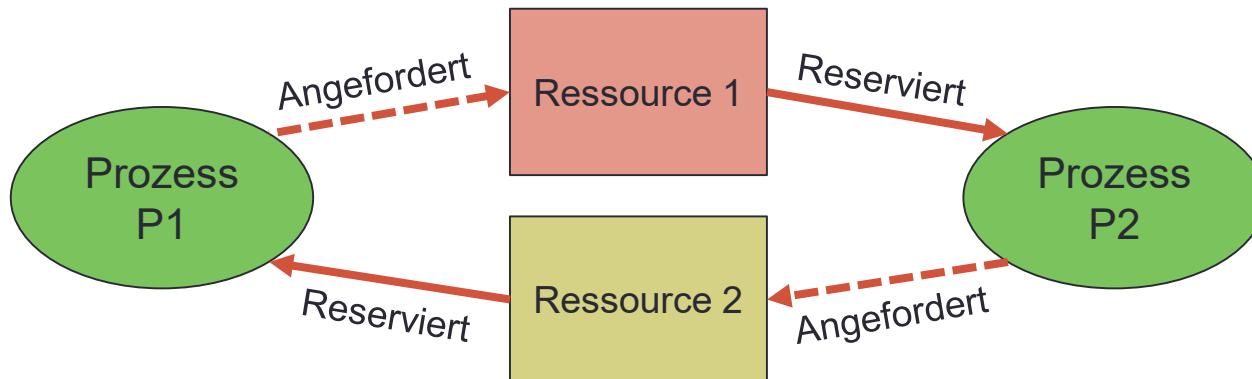
# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

Definition

- Eine Menge von Prozessen befindet sich im Deadlock, wenn jeder Prozess auf eine Ressource wartet, die nur von einem Prozess derselben Menge freigegeben werden kann



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

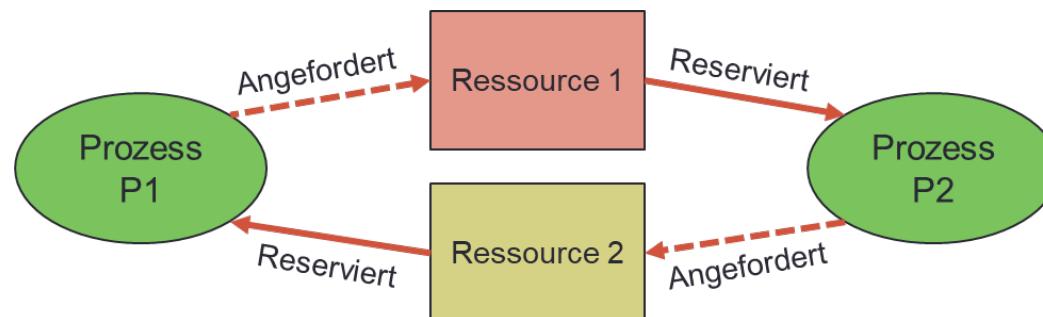
- Ein Deadlock entsteht unter den folgenden vier Bedingungen:
  1. **Mutual Exclusion Condition:** Eine Ressource steht einem Prozess nur exklusiv zu und kann nicht mehrfach genutzt werden
  2. **Wait for Condition (Hold and Wait):** Prozesse warten bis sie alle nötigen Ressourcen besitzen und behalten dabei die Kontrolle über Ressourcen, die ihnen bereits zugewiesen wurden
  3. **No Preemption Condition:** Keinem Prozess kann eine Ressource „gewaltsam“ entzogen werden

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

- Ein Deadlock entsteht unter den folgenden vier Bedingungen:
  4. **Circular Wait Condition:** Es gibt eine zyklische Kette von Prozessen, die bereits mindestens eine Ressource belegen und gleichzeitig auf weitere Ressourcen warten, die bereits andere Prozesse belegt haben



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

- Ein Deadlock entsteht genau dann, wenn das **Circular Wait** nicht aufzulösen ist
- Das ist der Fall, wenn Bedingungen 1-3 gelten
- Diese Bedingungen gelten in der Prozesssteuerung, die wir bisher betrachtet haben

Bedingungen 1-4 sind folglich **notwendig und hinreichend** für das Eintreten eines Deadlocks

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

- Deadlocks werden üblicherweise auf drei verschiedenen Wegen behandelt:
- Deadlocks **verhindern (Prevention)**
  - **Indirekt** durch Verhindern mindestens einer der Bedingungen 1-3
  - **Direkt** durch Verhindern des Circular Wait
- Deadlocks **vermeiden (Avoidance)**
  - Das Betriebssystem wird so konstruiert, dass Deadlocks nicht auftreten können
- Deadlocks **erkennen und beheben**
  - Wenn Deadlocks aufgetreten sind, werden Maßnahmen zur Beseitigung ergriffen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks

- ...oder die Möglichkeit von Deadlocks wird schlicht **ignoriert**
- Das tun UNIX und Windows, „Vogel-Strauß-Algorithmus“

Warum?

- Deadlocks können entstehen, müssen aber nicht
- Das Verhindern oder Vermeiden von Deadlocks kann sehr aufwändig und damit teuer sein
- Seltene Deadlocks werden durch Reboot kuriert, sie sind akzeptabel

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Deadlocks zu verhindern ist möglich, indem mindestens eine der vier zu seiner Entstehung nötigen Bedingungen verhindert wird:
  - Mutual Exclusion Condition
  - Wait for Condition (Hold and Wait)
  - No Preemption Condition
  - Circular Wait Condition
- 
- Nur: Welche?

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Mutual Exclusion Condition auflösen
- Es gibt unteilbare Ressourcen, die folglich nur von einem Prozess zur Zeit genutzt werden können: Einzelne CPU, Drucker, Festplatte...
- Der gegenseitige Ausschluss sollte ja durch Semaphore gerade bewirkt werden, weil er dringend benötigt wird
- Diese Bedingung kann deshalb nicht aufgehoben werden

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Wait for Condition (Hold and Wait) auflösen
- Das Betriebssystem könnte von allen Prozessen verlangen, dass sie alle benötigten Ressourcen **nur auf einmal reservieren** dürfen
- Das ist problematisch, weil Ressourcen sehr lange ungenutzt brach liegen würden, nur weil ein Prozess sie irgendwann brauchen wird
- Andere Prozesse müssten dann ewig auf häufig genutzte Ressourcen warten oder **verhungern**
- Kann deshalb praktisch nicht umgesetzt werden

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- No Preemption Condition auflösen
- Prozesse müssen bereits reservierte Ressourcen wieder freigeben, wenn sie eine weitere nötige Ressource nicht bekommen; später dürfen sie alle Ressourcen erneut anfordern
- Prozesse müssen ihre Ressourcen freigeben, wenn andere sie brauchen
- Kann nur teilweise umgesetzt werden, für Ressourcen deren Zustand sich leicht sichern lässt, z.B. Register und Speicher, aber nicht Drucker

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Circular Wait Condition auflösen
- Eine Lösung ist, eine strikte lineare **Ordnung** zu etablieren
- Alle Ressourcen durchnummernieren und eine Liste erstellen: R1, R2, R3....Rn
- Dabei gilt  $R1 < R2 < R3 < \dots < Rn$
- Ein Prozess, der  $R_i$  besitzt, darf nur  $R_j$  zugeteilt bekommen, wenn  $R_j > R_i$  ist ...

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Circular Wait Condition auflösen
  - ...Besitzt der Prozess bereits eine Ressource  $R_k$  mit  $R_k > R_j$ , bekommt er  $R_j$  nur, nachdem er zuerst  $R_k$  abgegeben hat
  - Nach  $R_j$  muss er dann  $R_k$  erneut reservieren
- 
- Das funktioniert und ist umsetzbar, führt aber möglicherweise zu schlechter Systemperformanz: Ein Prozess bekommt eine Ressource nur der Ordnung halber nicht und muss warten

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks verhindern (Prevention)

- Welche Bedingung also verhindern?

Bedingung	Konsequenz	möglich?
Mutual Exclusion	Aufgabe der exklusiven Ressourcennutzung	nein
Wait for (Hold and Wait)	Geringe Effizienz, Prozesse können verhungern	nein
No Preemption	Geht nur für manche Ressourcen	nein
Circular Wait	Komplexe Scheduler-Regeln, Performanz eingeschränkt	(ja)

- Einzig brauchbare Option ist Ausschließen der **Circular Wait Condition** – zu einem evtl. erheblichen Performanz-Preis

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesssteuerung – Deadlocks

- Deadlocks werden üblicherweise auf drei verschiedenen Wegen behandelt:
- Deadlocks verhindern (Prevention)
  - Indirekt durch Verhindern mindestens einer der Bedingungen 1-3
  - Direkt durch Verhindern des Circular Wait
- Deadlocks **vermeiden (Avoidance)**
  - Das Betriebssystem wird so konstruiert, dass Deadlocks nicht auftreten können
- Deadlocks **erkennen und beheben**
  - Wenn Deadlocks aufgetreten sind, werden Maßnahmen zur Beseitigung ergriffen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

- Das Betriebssystem wird so konstruiert, dass Deadlocks nicht auftreten können
- Umsetzung: Das System prüft, ob das Starten weiterer paralleler Prozesse bzw. die Vergabe weiterer Ressourcen zu einem Deadlock führen könnte
- Nur wenn das nicht der Fall ist, werden neue Prozesse gestartet oder weitere Ressourcen vergeben
- **Voraussetzung:** Die Anzahl der Prozesse sowie verfügbarer und maximal benötigter Ressourcen muss bekannt sein

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

- Zunächst werden drei Zustände definiert:
- **Deadlock:** Das System ist verklemmt, kein Fortschritt mehr möglich
- **Unsafe:** Es könnte sein, dass ein Deadlock auftritt
- **Safe:** Es gibt eine mindestens teilweise parallele Abfolge von Prozessen P<sub>1</sub>-P<sub>n</sub>, die jedem Prozess auch **mit maximal benötigten Ressourcen** den erfolgreichen Abschluss ermöglicht
- Wenn es mehrere Ressourcentypen gibt, muss diese Bedingung für jeden Ressourcentyp erfüllt sein

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

- Es gilt die Regel, dass Prozesse und Ressourcen nur dann gestattet werden, wenn dadurch nicht der Unsafe State erreicht wird
- Konservativer Ansatz, denn:
- Alle Deadlocks sind Unsafe, aber Unsafe ist nicht automatisch Deadlock



- Es muss also im Unsafe State gar nicht zum Deadlock kommen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

- Für die Prüfung hat Dijkstra 1965 den **Bankier-Algorithmus** vorgeschlagen
- Der Bankier-Algorithmus benötigt folgende Informationen:
- **R** als Vektor der insgesamt vorhandenen Ressourcen jedes Typs
- **Z** als Vektor der aktuell zugewiesenen Ressourcen jedes Typs
- **V** als Vektor der verfügbaren Ressourcen jedes Typs
- Je eine Matrix aller Prozesse und der ihnen **aktuell zugewiesenen** und der **noch maximal benötigten** Ressourcen jedes Typs

	Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed

$$\begin{aligned}R &= (6342) \\Z &= (5322) \\V &= (1020)\end{aligned}$$

Quelle: Geändert nach  
Tanenbaum und Bos (2016)

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

Der Algorithmus erledigt folgendes:

1. Suche einen Prozess, dessen maximaler Ressourcenbedarf  $\leq V$  ist
2. Weise diesem Prozess alle noch benötigten Ressourcen zu und lasse ihn seine Arbeit abschließen. Markiere den Prozess als beendet und füge die freien Ressourcen zu  $V$  hinzu
3. Wiederhole 1. und 2. für alle noch verbleibenden Prozesse, bis alle Prozesse als beendet markiert sind; ist das möglich, bleibt das System im Zustand Safe, andernfalls erreicht es Unsafe

	Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still needed

$$\begin{aligned}R &= (6342) \\Z &= (5322) \\V &= (1020)\end{aligned}$$

Quelle: Geändert nach  
Tanenbaum und Bos (2016)

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks vermeiden (Avoidance)

- Der Bankier-Algorithmus ist theoretisch funktionsfähig, wird aber in Betriebssystemen offenbar trotzdem nur selten benutzt
- Gründe dafür sind:
  - Kaum ein Prozess weiß im Vorhinein, welche Ressourcen er benötigen wird
  - Die Anzahl der Prozesse ändert sich im laufenden Betrieb ständig
  - Ressourcen können unbemerkt ausfallen
- In der Praxis ist Deadlock Avoidance oft ebenfalls nicht einsetzbar

	Process	Tape drives	Plotters	Printers	CD ROMs
Resources assigned	A   3 0 1 1	B   0 1 0 0	C   1 1 1 0	D   1 1 0 1	E   0 0 0 0
Resources still needed	A   1 1 0 0	B   0 1 1 2	C   3 1 0 0	D   0 0 1 0	E   2 1 1 0

	Process	Tape drives	Plotters	Printers	CD ROMs
Resources assigned	A   3 0 1 1	B   0 1 0 0	C   1 1 1 0	D   1 1 0 1	E   0 0 0 0
Resources still needed	A   1 1 0 0	B   0 1 1 2	C   3 1 0 0	D   0 0 1 0	E   2 1 1 0

$$\begin{aligned}R &= (6342) \\Z &= (5322) \\V &= (1020)\end{aligned}$$

Quelle: Geändert nach  
Tanenbaum und Bos (2016)

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesssteuerung – Deadlocks

- Deadlocks werden üblicherweise auf drei verschiedenen Wegen behandelt:
- Deadlocks verhindern (Prevention)
  - Indirekt durch Verhindern mindestens einer der Bedingungen 1-3
  - Direkt durch Verhindern des Circular Wait
- Deadlocks vermeiden (Avoidance)
  - Das Betriebssystem wird so konstruiert, dass Deadlocks nicht auftreten können
- **Deadlocks erkennen und beheben**
  - Wenn Deadlocks aufgetreten sind, werden Maßnahmen zur Beseitigung ergriffen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

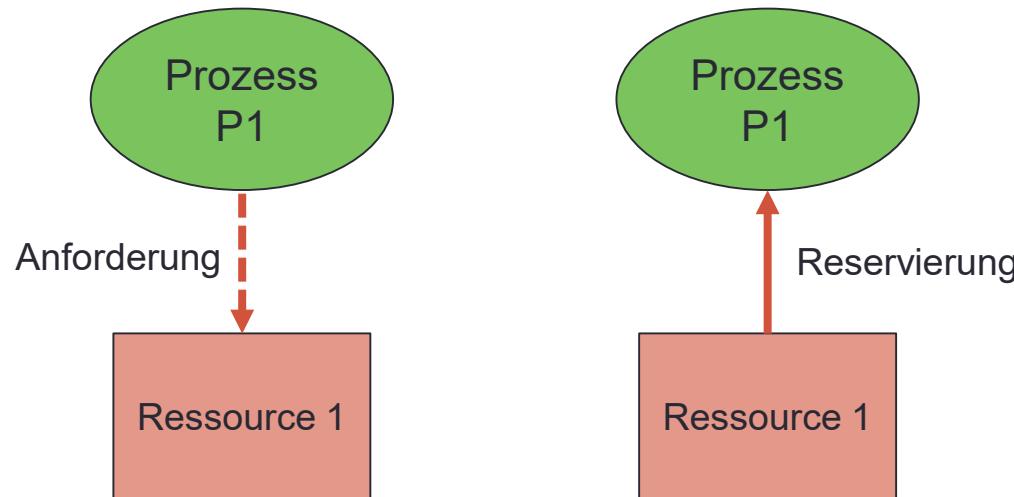
- Betriebssysteme können Deadlocks zulassen, indem sie alle Ressourcen gewähren, solange diese freie vorhanden sind
- Die Systeme müssen dann jedoch Maßnahmen zu deren Erkennung und Auflösung treffen
- Die Erkennung funktioniert unterschiedlich, abhängig davon, ob es von jedem Ressourcentyp eine oder mehrere Instanzen gibt

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Für Systeme mit einer Instanz pro Ressourcentyp, z.B. einer CPU, kann ein **Ressourcenbelegungsgraph** Deadlocks anzeigen

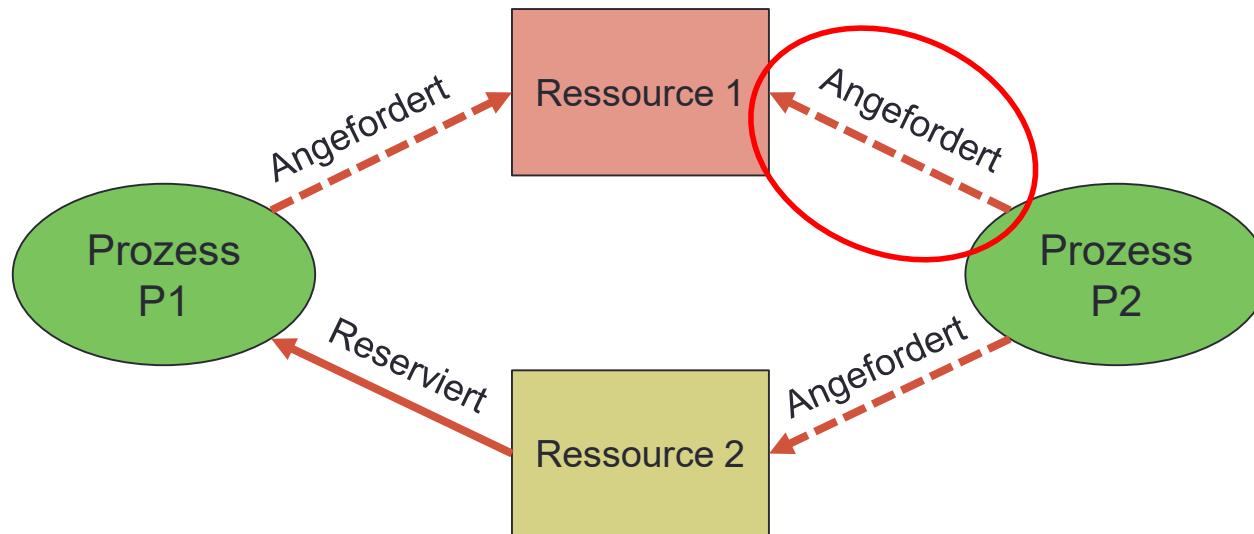


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Jede Anforderung von Ressourcen durch Prozesse wird aufgezeichnet

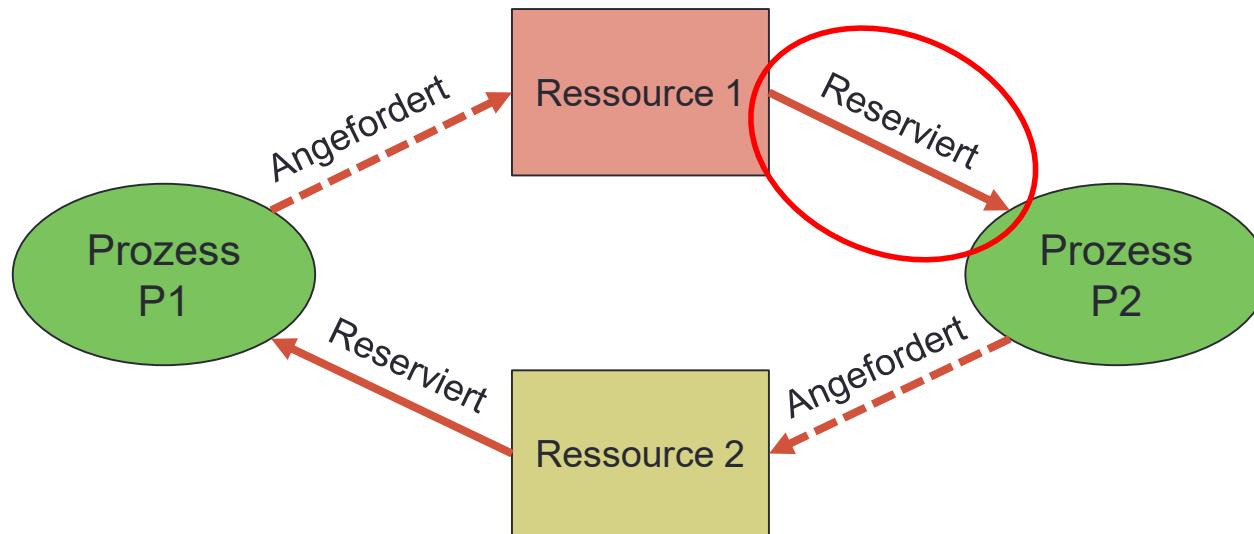


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Erteilte Reservierungen werden durch Vertauschen der Pfeilrichtung registriert

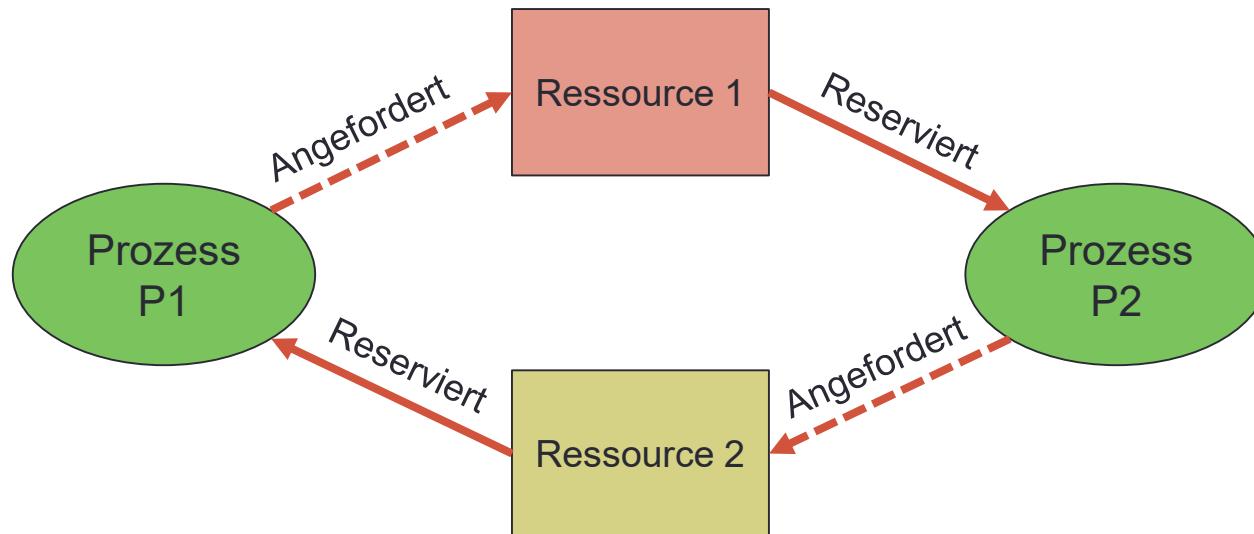


# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Zeigt sich in einem solchen Graphen an einer beliebigen Stelle ein Zyklus, dann liegt ein Deadlock vor (vgl. Folie 59)



# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Für Systeme mit mehreren Instanzen je Ressourcentyp, etwa Speicher oder mehrere CPUs, kann der Bankier-Algorithmus verwendet werden:
- Suche einen Prozess, der mit den aktuell nicht zugewiesenen Ressourcen beendet werden kann
- Beende diesen, markiere ihn als beendet und füge seine frei gewordenen Ressourcen den nicht zugewiesenen Ressourcen hinzu
- Ist dies für alle Prozesse möglich, gibt es keinen Deadlock
- Bleiben ein oder mehrere Prozesse übrig, besteht ein Deadlock

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Hat ein System ein Deadlock erkannt, besteht die Wahl zwischen verschiedenen Methoden der Auflösung
- Abbrechen aller beteiligten Prozesse
- Abbrechen einzelner beteiligter Prozesse, wenn nötig Schrittweise
- Rollback, sofern es eine Form des Protokolls gibt (Transaktionsprotokoll, Checkpointdatei)
- Zeitweiliges Entziehen einer Ressource eines Prozesses (preemption)

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Deadlocks erkennen und auflösen

- Entscheidende Frage beim Abbruch einzelner Prozesse ist:

Wer wird das Opfer?

- Vorzugsweise Prozesse, die ohne gravierende Nebenwirkungen neu gestartet werden können: Lesende Prozesse wie Programmstarts oder Aufrufe von Websites
- Bitte keine komplexen Datenbankaktionen oder Printjobs!

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Zusammenfassung

- Prozesse und Threads laufen quasi parallel, indem allen jeweils nur ein **kurzes Zeitfenster der CPU** zugeteilt wird
- Vor und nach diesem Zeitfenster sind die Prozesse inaktiv
- Um Fehler bei gemeinsam genutzten Ressourcen zu vermeiden, gibt es verschiedene Techniken wie die Nutzung von **Sperrvariablen** oder **Semaphoren**
- **Deadlocks** können ignoriert, verhindert, vermieden oder erkannt und behoben werden
- Die Prozesssteuerung übernimmt der **Scheduler** im Betriebssystem

# Betriebssysteme

## Aufgaben – Prozesse und Threads verwalten

### Prozesssteuerung – Scheduling

- Für die wirklich sichere und effiziente Zusammenarbeit von Prozessen haben Tanenbaum und Bos (2016) vier Bedingungen definiert:
  - Keine zwei Prozesse dürfen gleichzeitig in ihren kritischen Regionen sein
  - Es dürfen keine Annahmen über Geschwindigkeit und Anzahl der CPUs gemacht werden
  - Kein Prozess, der selbst nicht in einer kritischen Region ist, darf einen anderen Prozess blockieren
  - Kein Prozess soll unendlich lange warten müssen bis er in seine kritische Region eintreten darf
- Es ist die Aufgabe des Schedulers, diese Bedingungen zu sichern

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling



© Can Stock Photo

- Diese Bedingungen können auf verschiedene Weise erfüllt werden:
- Wie beschrieben per Interprozesskommunikation (IPC)
  - **Synchronisierung** von Prozessen zur Vermeidung von Race Conditions und daraus entstehenden Fehlern
  - **Kommunikation** zwischen Prozessen
  - **Deadlock-Behandlung**
- Verschiedene **Instanzen, Regeln und Strategien** für das Scheduling
  - Entscheidung, welchem Prozess als nächsten Ressourcen zugeteilt werden

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Instanzen im System

- Abhängig davon, mit welchen Zeitfenstern ein Scheduler operiert, werden drei Scheduler-Typen unterschieden:
- Long-term Scheduler
- Medium-term Scheduler
- Short-term Scheduler

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Instanzen im System

- **Long-term Scheduler** entscheiden, welchen neuen Prozess sie zum Eintritt in die Bereit-Warteschlange zulassen
- Dabei wird zwischen CPU-lastigen und I/O-lastigen Prozessen unterschieden
- Aufgabe ist es, eine gute Balance zwischen beiden Prozesstypen zu finden und die Prozesse entsprechend zuzulassen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Instanzen im System

- Medium-term Scheduler kümmern sich darum, welche Prozesse sich vorübergehend aus dem Hauptspeicher entfernen lassen und in langsamere Teile des virtuellen Speicher übertragen werden können
- Sie übernehmen das **Swapping** oder **Paging**
- Short-term Scheduler weisen Prozesse aus der Bereit-Warteschlange die Prozessorzeit zu
- Die folgenden Erörterungen beziehen sich vorwiegend auf Short-term Scheduling

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Ein Scheduler hat grundsätzlich zwei Möglichkeiten, Prozesse zu steuern:
- **Nicht unterbrechend:** ein einmal gestarteter Prozess läuft so lange bis er beendet ist oder blockiert, der Scheduler greift nicht ein; blockierte oder beendete Prozesse werden sofort durch neue ersetzt
- **Unterbrechend:** Der Scheduler unterbricht laufende Prozesse nach bestimmten Regeln, um z.B. die Durchlaufzeit zu optimieren

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Das Scheduling hat je nach Art des Betriebssystems verschiedene Schwerpunkte

Alle Betriebssysteme	Stapelverarbeitungs-systeme	Interaktive Systeme	Echtzeitsysteme
<ul style="list-style-type: none"><li><b>Fairness:</b> Jeder Prozess ist mal dran</li><li><b>Strategien werden durchgesetzt</b></li><li><b>Balance:</b> Alle Systemkomponenten sind ausgelastet</li></ul>	<ul style="list-style-type: none"><li><b>Durchsatz:</b> So viele Jobs pro Stunde wie möglich</li><li><b>Minimale Durchlaufzeit</b> für jeden Job</li><li><b>Maximale CPU-Auslastung</b></li></ul>	<ul style="list-style-type: none"><li><b>Antwortzeit:</b> schnelle Antwort auf Eingaben</li><li><b>Proportionalität:</b> Erwartungen des Benutzers erfüllen</li></ul>	<ul style="list-style-type: none"><li><b>Deadlines einhalten:</b> Datenverlust vermeiden</li><li><b>Vorhersagbarkeit:</b> keine Qualitäts-einbußen beim Streaming</li></ul>

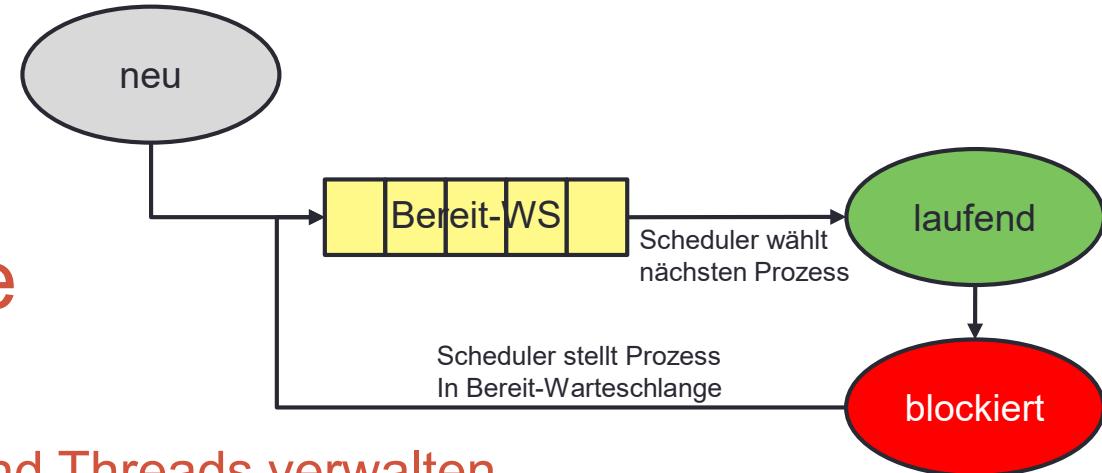
# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- In Stapelverarbeitungssystemen sind folgende Strategien üblich:
  - First Come First Serve (FCFS), auch bekannt als First in First Out (FIFO)
  - Shortest Job First bzw. Shortest Job Next
  - Shortest Remaining Time Next

# Betriebssysteme



Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- First Come First Serve ist eine nicht unterbrechende Strategie
- Alle neuen Prozesse werden in eine Warteschlange gestellt und der Reihe nach abgearbeitet
- Ein beendeter Prozess wird durch einen neuen ersetzt
- Ein blockierter Prozess wird wieder hinten in die Warteschlange eingereiht und durch den nächsten in der Warteschlange ersetzt
- Vorteil: sehr einfach, wenig Overhead
- Nachteil: Lange dauernde Prozesse bremsen kurze aus

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Shortest Job First bzw. Shortest Job Next ist ebenfalls nicht unterbrechend, sortiert aber die Warteschlange nach der angenommenen Bearbeitungszeit eines Prozesses
- Kurze Prozesse kommen zuerst dran:



- Durchschnittliche Durchlaufzeit links:  $(10+15+20+25)/4=17,5$
- Durchschnittliche Durchlaufzeit rechts:  $(5+10+15+25)/4=13,75$

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Shortest Job First (SJF) kann also die Durchlaufzeit optimieren, wenn:
  - alle Prozesse gleichzeitig verfügbar sind
  - die Durchlaufzeit jedes Prozesses vorher bekannt ist
- Shortest Remaining Time Next ist ein unterbrechendes SJF
- Jeder neue Prozess wird an der verbleibenden Laufzeit des aktuellen Prozesses gemessen
- Ist der neue Prozess kürzer als die Restlaufzeit des aktuellen Prozesses, wird dieser unterbrochen und der neue zuerst abgearbeitet
- Bevorzugt neue, kurze Prozesse

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- In **interaktiven Betriebssystemen** werden andere Strategien verwendet (Auswahl):
  - Round-robin Scheduling
  - Priority-Scheduling
  - Fair-Share-Scheduling

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Round-robin-Scheduling teilt reihum jedem Prozess ein festes Zeitfenster zu
- Das Ergebnis ist ein ausgewogenes Verhältnis zwischen den Prozessen bezüglich Durchsatz und Durchlaufzeit
- Erzeugen relativ viel Overhead, weil dazu keine Optimierung vorgenommen wird
- Berücksichtigen nicht die Bedeutung von Prozessen

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Priority-Scheduling vergibt Prioritäten an Prozesse und weist den Prozessen entsprechend ihrer Priorität mehr oder weniger Ressourcen zu
- Problem: Wie wird **Verhungern** verhindert, also dafür gesorgt, dass Prozesse mit niedriger Priorität überhaupt abgearbeitet werden?
- Mögliche Verfahren: **Dynamische Prioritäten**, Kombination von **Prioritätsklassen** und **Round-robin** in einer Klasse

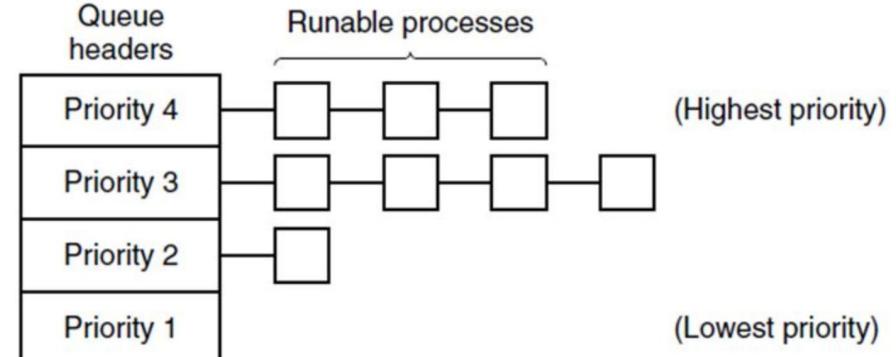
# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Priority-Scheduling mit dynamischen Prioritäten erteilt neuen Prozessen eine prozesstypabhängige Priorität, etwa höher für interaktive Prozesse und niedriger für Hintergrundprozesse
- Bekommt ein Prozess erstmals eine Ressource zugewiesen, wird er bearbeitet und seine Priorität dann herabgesetzt
- Im Ergebnis „überholen“ Prozesse mit niedrigerer Priorität irgendwann die ursprünglich höher priorisierten Prozesse und werden dann ausgeführt

# Betriebssysteme



Quelle: Tanenbaum und Bos (2016)

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Priority-Scheduling mit Prioritätsklassen kann z.B. mehrere Warteschlangen verwalten: **Multilevel Queue Scheduling**
- Scheduling findet dann nur zwischen den Klassen statt, innerhalb einer Klasse wird z.B. Round-robin eingesetzt

Strategie:

- Lasse zuerst alle Prozesse der höchsten Priorität per Round-robin laufen, bis diese Warteschlange leer ist
- Bediene dann ebenso die folgenden Warteschlangen
- Prozesse wechseln nicht zwischen Klassen, die Prioritäten müssen dynamisch angepasst werden, damit kein Verhungern auftritt

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Um möglichst flexible und gerechte Ressourcenverteilung zu erreichen, kann Multilevel Queue Scheduling um eine **Bewertung des vorherigen Verhaltens eines Prozesses** erweitert werden:
  - Das Scheduling verschiebt Prozesse z.B. nach bereits verwendeter CPU-zeit oder nach I/O-Aktivität zwischen den Prioritätsklassen
  - Prozesse, die zuviel CPU-Zeit nutzen, werden herabgestuft, lange wartende Prozesse heraufgestuft
- Diese Strategie heißt **Multilevel Feedback Queue Scheduling**, sie ist äußerst flexibel und sehr komplex
- Windows, MacOS und Linux bis 2.6.0 verwenden diese Strategie

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling-Regeln und Strategien

- Fair Share Scheduling berücksichtigt den Besitzer eines Prozesses und bewertet dessen Anteil an der gesamten Systemleistung
- Um Benutzer mit vielen offenen Prozessen nicht überproportional zu bevorzugen, kann jedem Benutzer ein proportionaler CPU-Anteil gegeben werden
- Der Scheduler prüft vor der Zuteilung von Ressourcen, wem der Prozess gehört und teilt nur dann zu, wenn dieser Besitzer seinen Anteil noch nicht mit anderen Prozessen ausgeschöpft hat
- Linux ab 2.6.3 nutzt eine sehr komplexe Variante, das Completely Fair Scheduling

# Betriebssysteme

Aufgaben – Prozesse und Threads verwalten

Prozesssteuerung – Scheduling: Zusammenfassung

- Scheduler nutzen bestimmte Strategien zur Entscheidung darüber, welcher Prozess eine Ressource erhält
- Je nach Betriebssystemtyp werden andere Ziele verfolgt:
- Maximaler Durchsatz, den Erwartungen entsprechende Antwortzeiten oder harte und sichere Einhaltung von Deadlines (Echtzeitbetriebssysteme)
- Viele Betriebssysteme nutzen eine universelle und sehr komplexe Strategie, die sich Multilevel Feedback Queue nennt und für die meisten Anwendungsfälle gute Ergebnisse liefert.

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - **Speicherverwaltung**
  - Dateisysteme
  - Ein- und Ausgabe
  - Multiprozessorsysteme
  - Virtualisierung

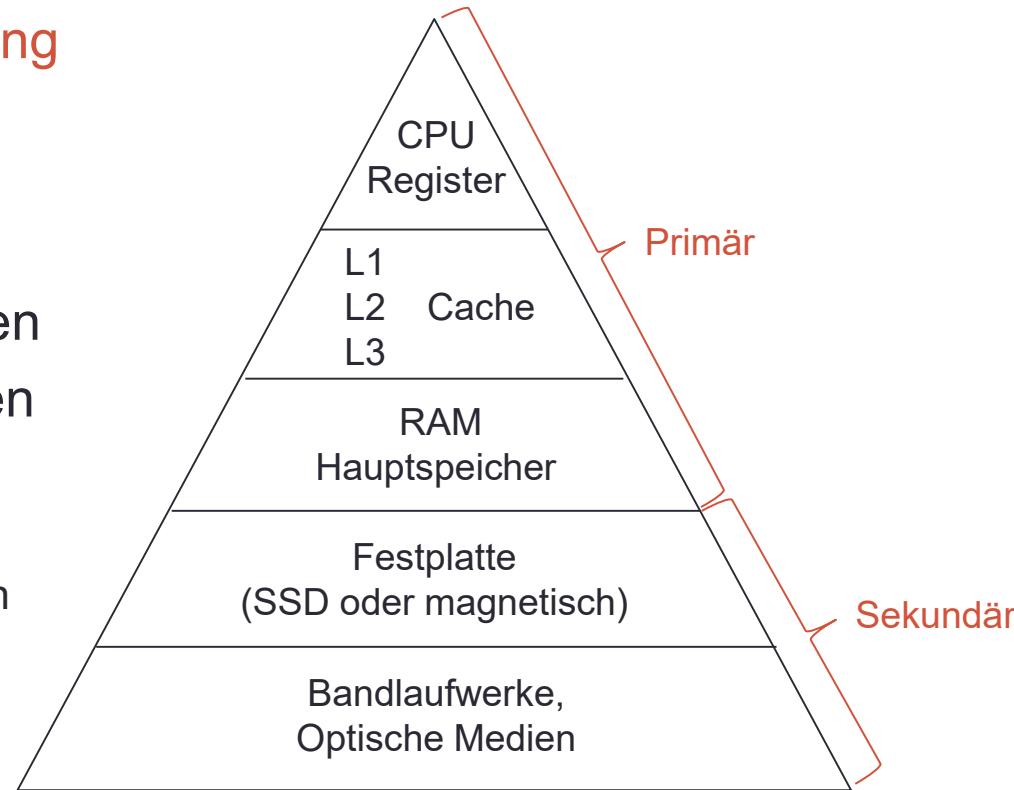
Praxis: Umgang mit Linux (Prof. Brunner) und Windows Server 2019

# Betriebssysteme

## Aufgaben – Speicherverwaltung

### Speicherhierarchie

- Computer besitzen mehrere Speicherebenen mit verschiedenen Eigenschaften
- Die Speicherelemente lassen sich in einer hierarchischen Struktur zusammenfassen
  - **Primärer Speicher:** schnell, klein bis mittel, preiswert bis teuer
  - **Sekundärer Speicher:** langsam, groß bis sehr groß, preiswert

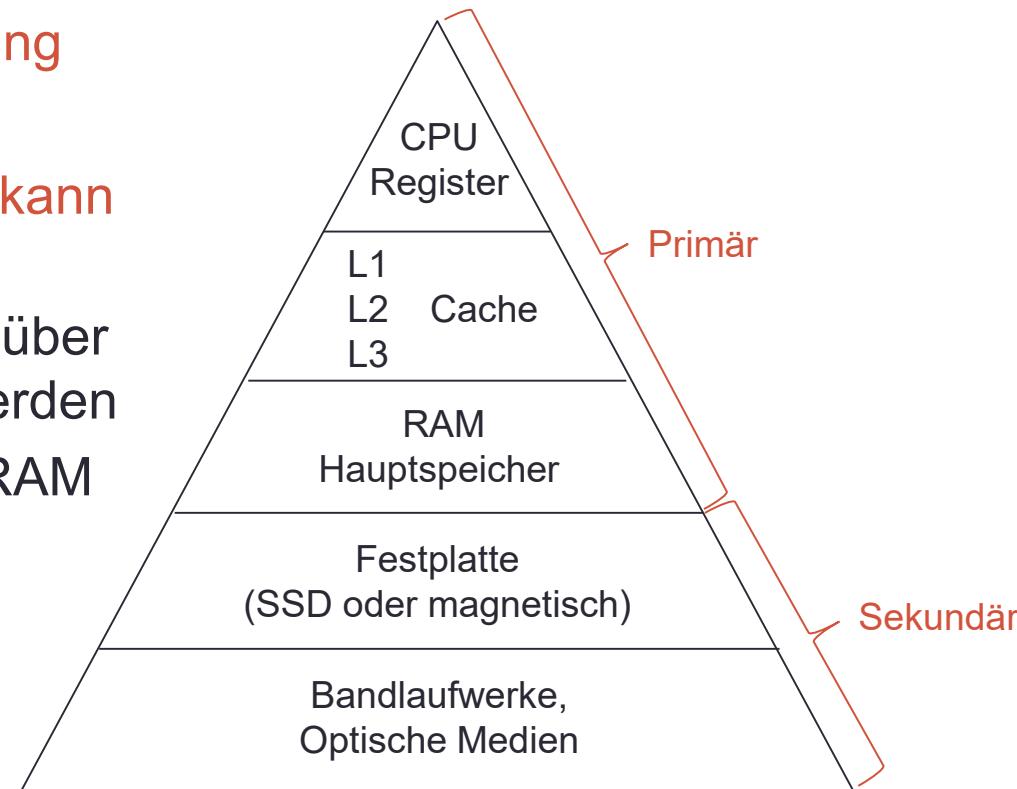


# Betriebssysteme

## Aufgaben – Speicherverwaltung

### Speicherhierarchie

- Nur den primären Speicher kann die CPU direkt ansprechen
- Sekundärer Speicher muss über Controller angesprochen werden
- Primärer Speicher ist als SRAM und DRAM ausgeführt und flüchtig
- Sekundärer Speicher ist nichtflüchtig

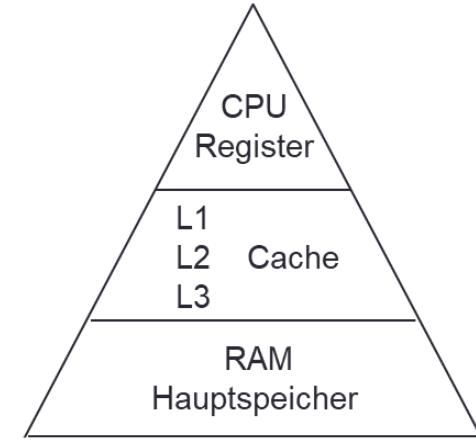


# Betriebssysteme

## Aufgaben – Speicherverwaltung

### Speicherhierarchie

- Die **Speicherverwaltung** des Betriebssystems betrifft logisch den **Primären Speicher**, kann sich aber physikalisch auf den Sekundären Speicher ausdehnen
- Moderne Betriebssysteme **abstrahieren und virtualisieren** den Primären Speicher
- Wir betrachten hier die Methoden der Speicherverwaltung



# Betriebssysteme

## Aufgaben – Speicherverwaltung

### Ziele

- In aktuellen Betriebssystemen laufen gleichzeitig z. B. etwa 100 Prozesse, die alle um Arbeitsspeicher konkurrieren
- Die **Speicherverwaltung** soll verhindern, dass...
- Prozesse Speicherbereiche des **Kernels verwenden**
- Prozesse auf Speicherbereiche anderer Prozesse zugreifen
- Alle laufenden Prozesse genügend Speicher zur Verfügung haben
  - Ohne Verschwendungen von Speicher
  - unabhängig vom tatsächlich physikalisch vorhandenen RAM

# Betriebssysteme

## Aufgaben – Speicherverwaltung

### Methoden

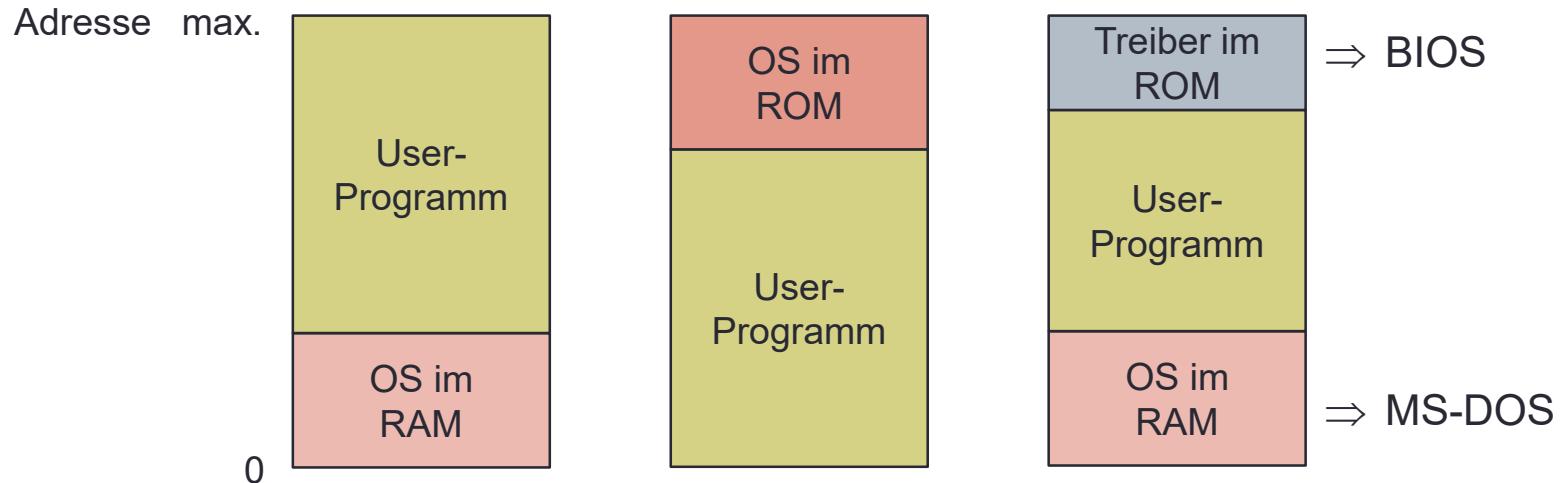
- Das Betriebssystem arbeitet eng mit spezieller Hardware zur Speicherverwaltung zusammen
- Dazu wurden mehrere Werkzeuge entwickelt, z.B.:
  - **Basis- und Limit-Register** (v.a. historisch, einige 8/16 Bit Prozessoren)
  - Segmentierung
  - Swapping
  - Virtueller Speicher mit Paging
- Alle diese Methoden **abstrahieren den physikalischen Speicher**

# Betriebssysteme

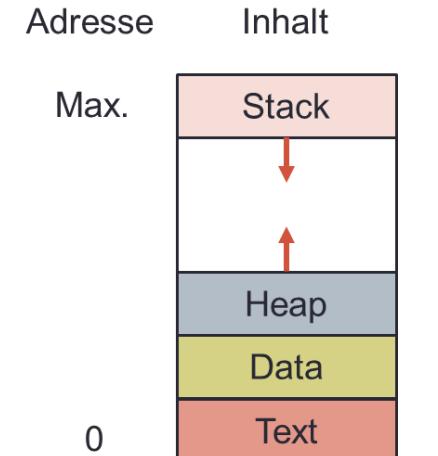
Aufgaben – Speicherverwaltung

Methoden: Monoprogrammierung und physikalischer Speicher

- Manche Handhelds, eingebettete Systeme und MS-DOS führ(t)en nur einen Prozess gleichzeitig aus und abstrahier(t)en den Speicher nicht



# Betriebssysteme



Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion, logischer und physikalischer Speicher

- Speicherabstraktion führt die Idee aus dem Prozessmodell fort, jedem Prozess seinen eigenen privaten Speicherbereich zuzustehen
- Dieser **Adressraum** beginnt für jeden Prozess bei 0, ist **zusammenhängend** und endet bei einer höchsten Adresse
- **Vorteil:** Der Prozess (und der Programmierer) arbeitet intern mit logischen, innerhalb des Prozesse eindeutigen Adressen, egal auf welcher Maschine der Prozess läuft
- **Nachteil:** Mehrere parallele Prozesse verwenden für sich die gleichen logischen Adressen

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion, logischer und physikalischer Speicher

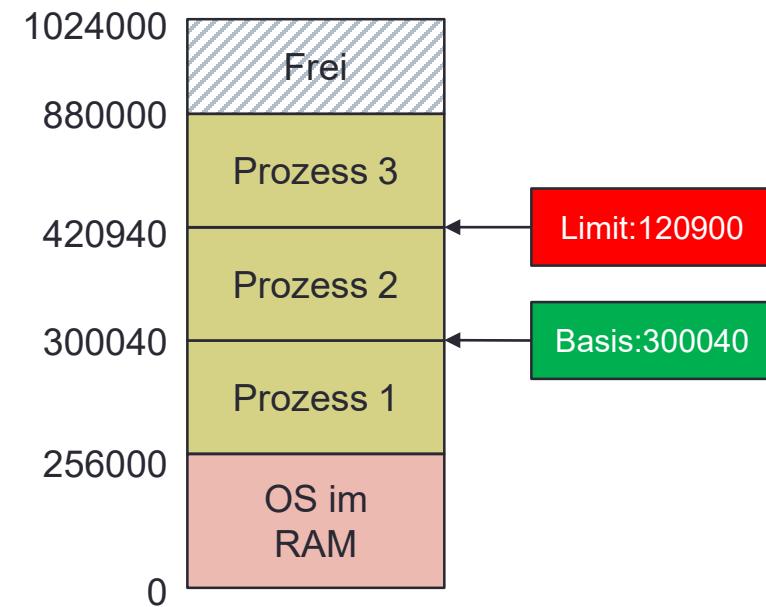
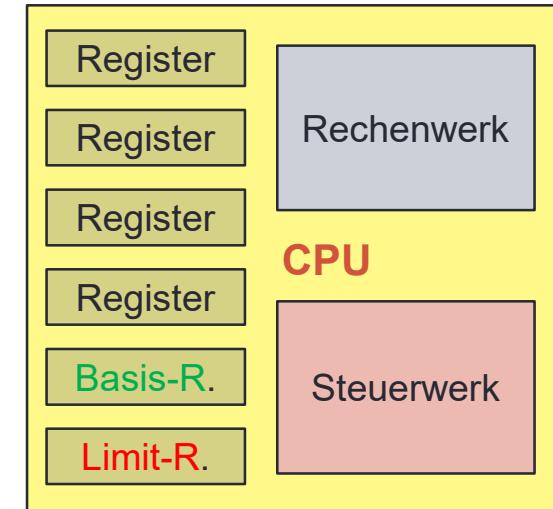
- Die Speicherabstraktion muss dafür sorgen, dass die gleichen logischen Adressen verschiedener paralleler Prozesse zuverlässig und eindeutig auf verschiedene physikalische Adressen im RAM abgebildet werden: Speicherabbildung
- Eine frühe, einfache Lösung dafür war, der CPU zwei zusätzliche Register zu geben:
  - Basis-Register
  - Limit-Register
- Diese Idee ermöglicht Nebenläufigkeit für Prozesse im phys. Speicher

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion,  
logischer und physikalischer  
Speicher

- Das **Basis-Register** enthält einen **individuellen Adress-Offset** für den Prozess, der jeder Speicheranforderung hinzugaddiert wird
- Das **Limit-Register** enthält einen **relativen maximalen Adresswert** für jeden Prozess oberhalb von dessen Basisadresse

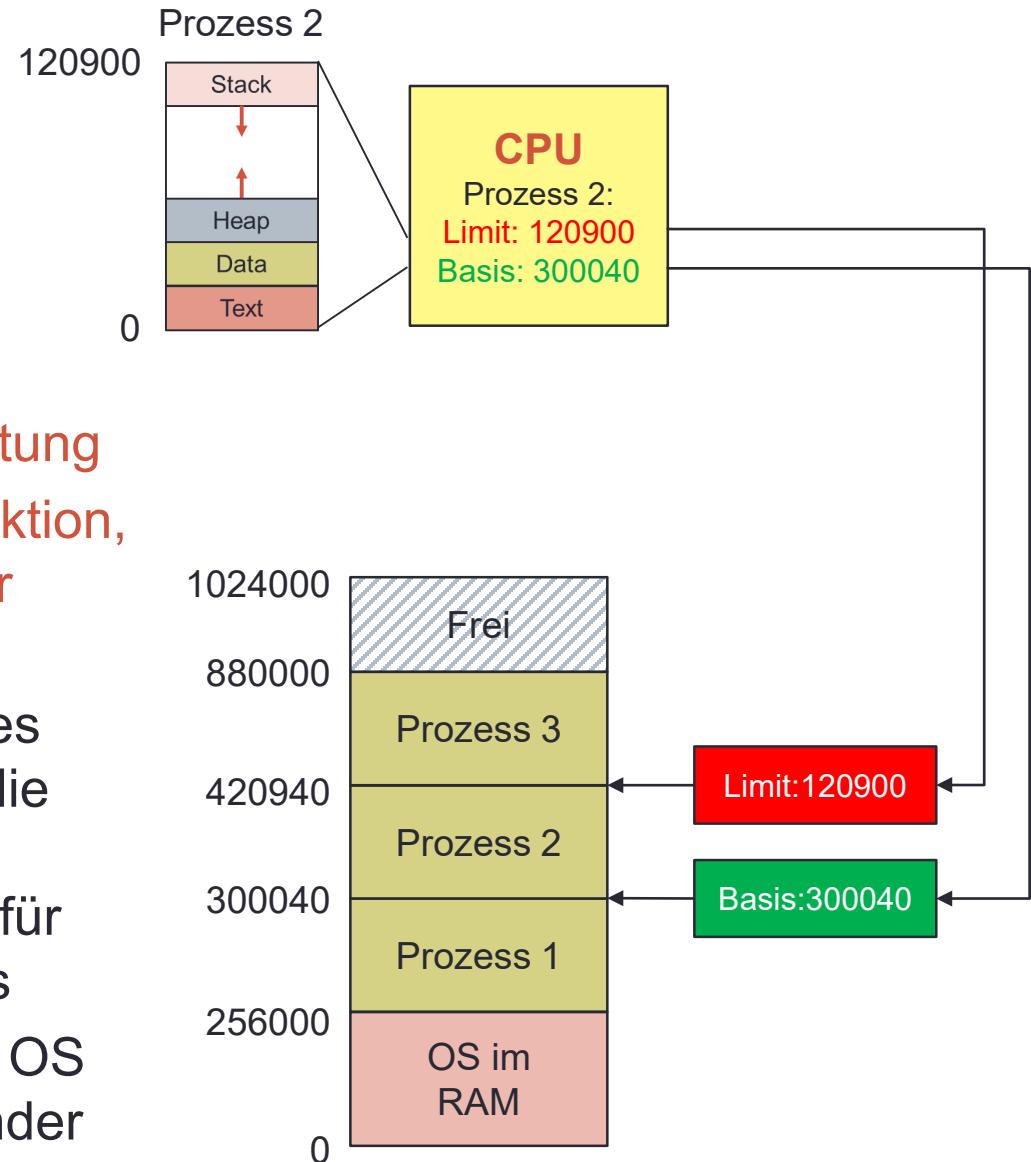


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion,  
logischer und physikalischer  
Speicher

- Bei jeder Anforderung eines Prozesses hebt die CPU die geforderte Adresse in das richtige Speichersegment für den anfordernden Prozess
- Die Speicherbereiche von OS und Prozessen untereinander sind so wirksam getrennt



# Betriebssysteme

Aufgaben – Speicherverwaltung

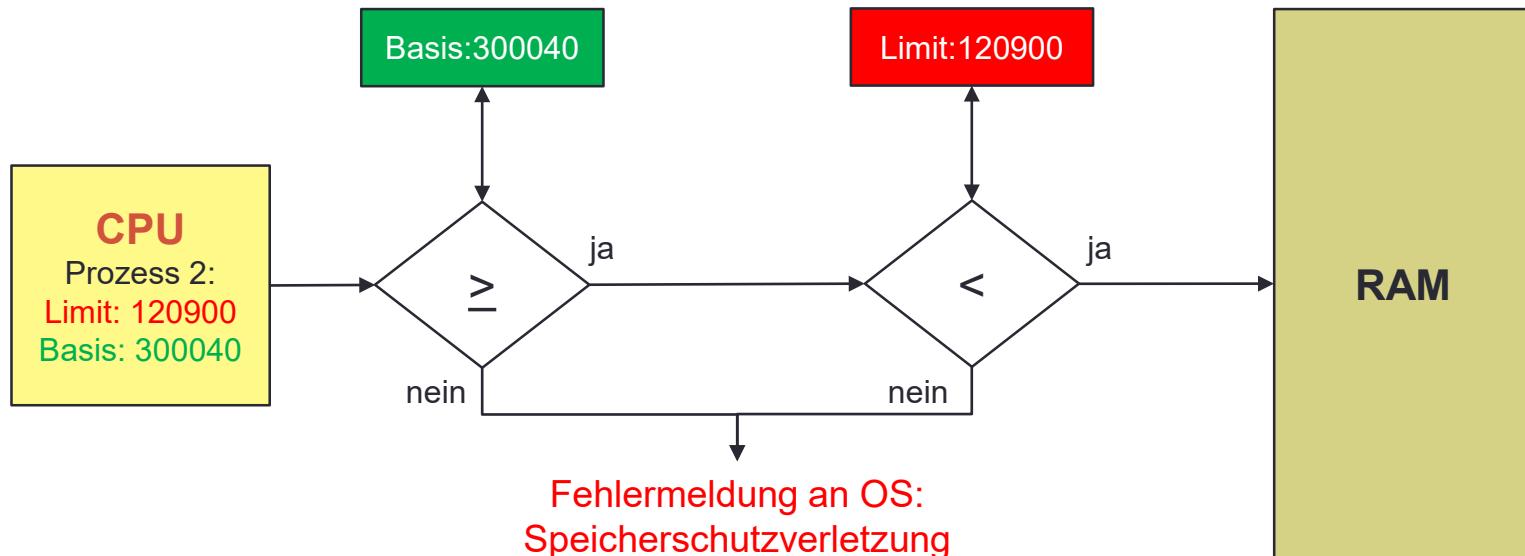
Methoden – Speicherabstraktion, logischer und physikalischer Speicher

- Diese **Segmentierung** des physikalischen Speichers wurde zusätzlich mit einem **Speicherschutz** versehen:
- Die CPU wird angewiesen, jeden Speicherzugriff eines Prozesses zu prüfen
- Nur wenn die vom Prozess angeforderten Adressen sich nach Addition des Basisregisterwertes noch im Speicherbereich des Prozesses befinden (Limit), wird die Anforderung ausgeführt
- Andernfalls wird eine Fehlermeldung ausgegeben und der Prozess beendet

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion, logischer und physikalischer Speicher



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion in x86-Prozessoren

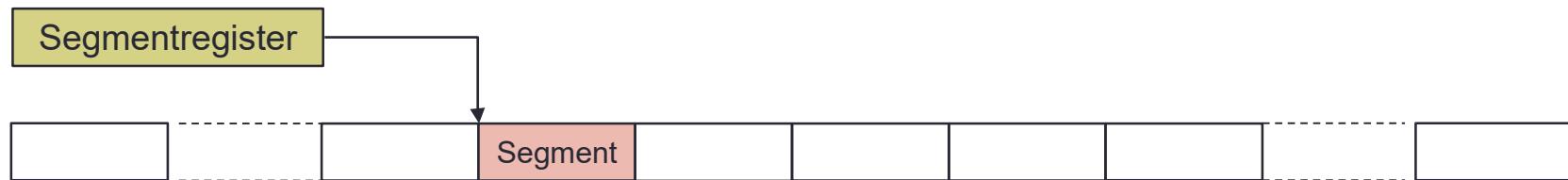
- Die x86-Prozessorfamilie verwendete zunächst nur physikalischen Speicher und adressierte diesen im **Real Mode**
- Der Real Mode realisierte eine Segmentierung mit festen Abständen
- Mit 20 Bit Adressbusbreite konnten der 8086/8088 und 80186 1024000 Bit oder 1 MB adressieren
- Die einzelne Adresse wurde in zwei Teilen angegeben, einer **Segmentadresse** und einer **Offsetadresse** für jedes Segment

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion in x86-Prozessoren

- Die **Segmentadresse** wird in einem 16-Bit-Register gehalten und zeigt auf den Beginn eines Speichersegments



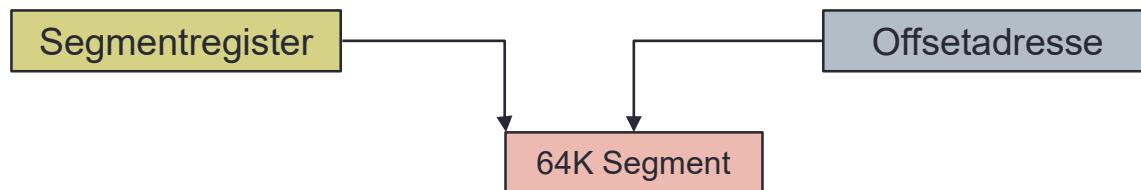
- Von den 16 Bit im Registers haben aber nur die 12 höchstwertigen Bits eine Bedeutung, die vier LSB sind 0 und werden ignoriert (entspricht Multiplikation mit 16)

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion in x86-Prozessoren

- Die **Offsetadresse** zeigt auf eine Adresse innerhalb des jeweiligen Segments



- Die gesamte Speicheradresse wird aus den ersten 12 Bits des Segmentzeigers und der 8 Bit langen Offsetadresse gebildet
- Damit ergeben sich Segmente mit je 64K Größe und 20 Bit Adressen
- Jede 32-Bit x86 CPU startet im Real Mode

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion in x86-Prozessoren

- Im Real Mode bestand **kein** Speicherschutz
- Über einen unglücklichen Zwischenschritt mit dem 16-Bit-Prozessor 80286 wurde 1985 mit dem 80386 der **Protected Mode** eingeführt
- Er bot Speicherschutz und 32-Bit-Adressraum, also 4 GB, und weitere Vorteile des noch zu besprechenden **Virtuellen Speichers**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Prozesse auslagern: Swapping

- Die Segmentierung mit z.B. Basis- und Limit-Register erlaubte die Ausführung mehrerer Prozesse innerhalb des vorhandenen physikalischen Arbeitsspeichers
- Übersteigt die Summe des Speicherbedarfs allerdings diesen Speicherplatz, können keine weiteren Programme ausgeführt werden
- Lösungsansatz: Die nächste Ebene der Speicherhierarchie in den verfügbaren logischen Adressraum einbinden, auch wenn sie deutlich langsamer ist: **Swapping**
- **Auslagern aktuell nicht aktiver Prozesse auf die Festplatte**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

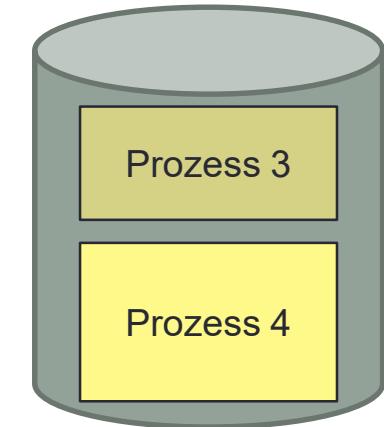
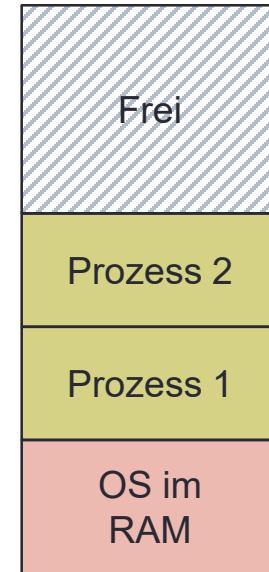
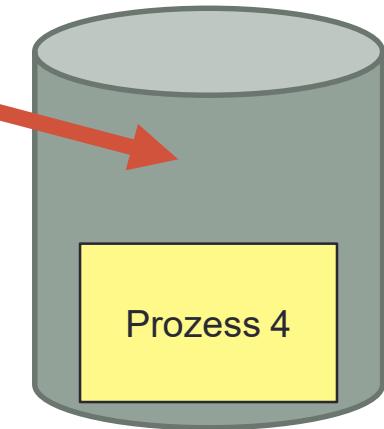
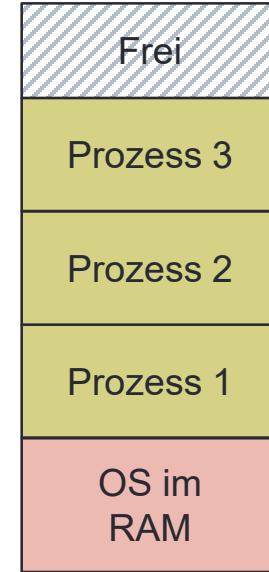
- Der Scheduler im Betriebssystem kennt den Aktivitätsstatus aller Prozesse im System
- Er kann entscheiden, ob und wie lange ein Prozess nicht benötigt wird
- Wird ein Prozess einige Zeit nicht benötigt, kann er zeitweise komplett auf die Festplatte ausgelagert werden
- Der frei werdende Platz im RAM kann von einem anderen Prozess belegt werden

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

1. Swap Out: Prozess 3 aus dem RAM auf die Festplatte auslagern

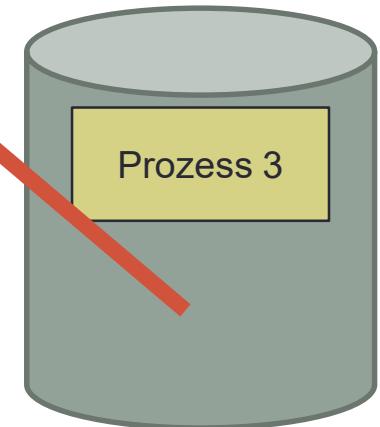
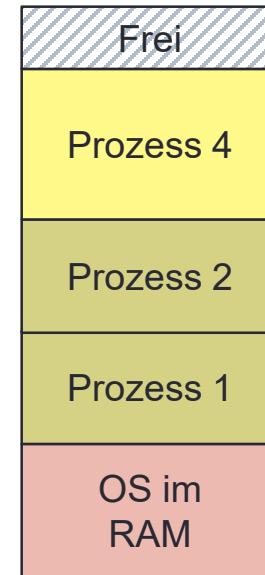
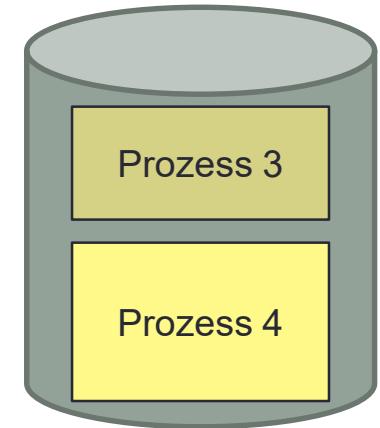
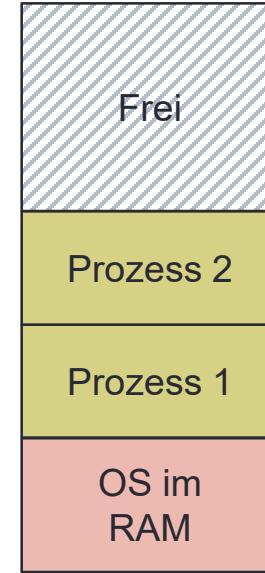


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

1. Swap Out: Prozess 3 aus dem RAM auf die Festplatte auslagern
  2. **Swap In:** Prozess 4 von der Festplatte in den RAM einlagern
- 
- Anschließend ist Prozess 4 zur Ausführung fertig

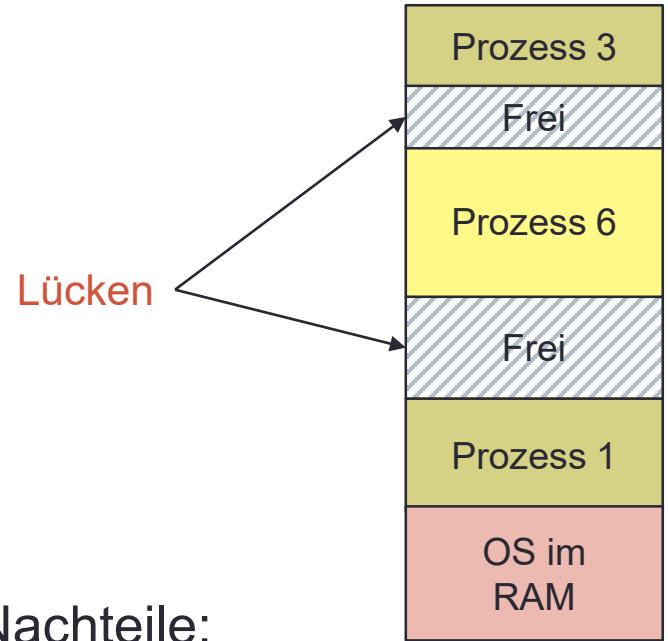


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

- Swapping hat jedoch zwei wesentliche Nachteile:
- **Fragmentierung des Arbeitsspeichers:** Durch Aus- und Einlagerung von Prozessen mit unterschiedlichem und veränderlichem Speichervolumen entstehen verschieden große ungenutzte Lücken im Arbeitsspeicher
  - **Externe Fragmentierung** zwischen den Prozessen
  - **Interne Fragmentierung** innerhalb der Prozesse (Heap/Stack-Zwischenraum)
- **Ineffizienz durch hohe Datenmengen und lange Wartezeiten** bei Aus- und Einlagerung (Festplatten sind 100.000 mal langsamer als RAM)



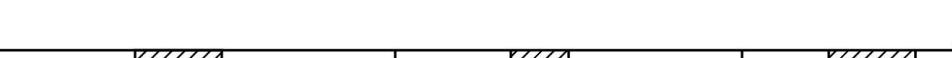
# Betriebssysteme

# Aufgaben – Speicherverwaltung

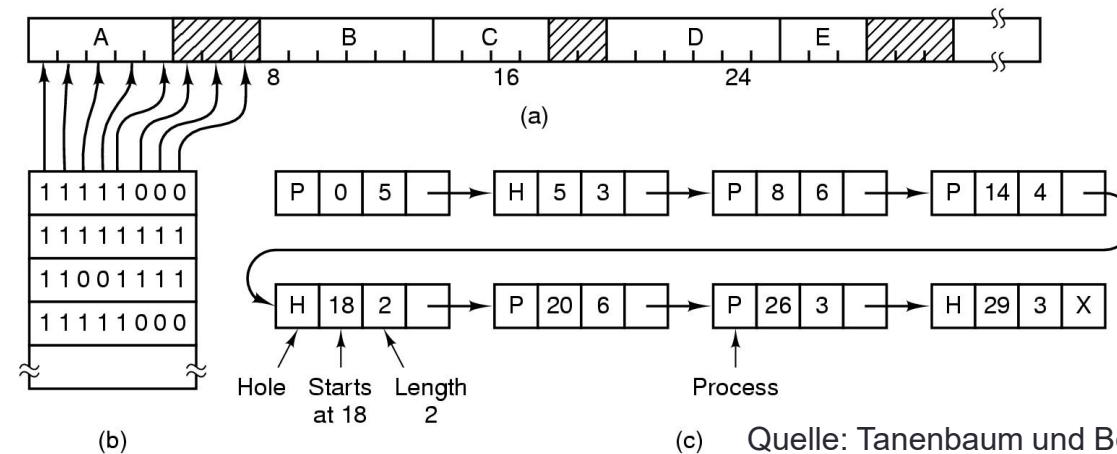
# Methoden – Swapping

- Die Lücken zwischen den Prozessen werden vom Betriebssystem verwaltet und bestmöglich wieder aufgefüllt
  - Die Verwaltung verwendet entweder:
  - **Bitmaps** (b)
  - oder
  - **Listen** (c)

(a)



(b)



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

- Vor der Einlagerung von Prozessen von der Festplatte in den RAM sucht das Betriebssystem eine passende Lücke
- Dafür können verschiedene Algorithmen verwendet werden
- **First Fit:** Sucht die **erste** ausreichend große Lücke
- **Best Fit:** Sucht die **am besten passende** Lücke
- **Worst Fit:** Sucht die **größte** Lücke und lässt den größten Restbereich
- **Quick Fit:** Findet häufige Größen (4K, 8K, 16K, 20K...) sehr schnell
- Beste Leistung haben First Fit und Quick Fit, **aber...**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Swapping

- ...die Fragmentierung kann nicht völlig vermieden werden
- Nach einiger Zeit sind bis zu 1/3 des Speichers in so kleine Fragmente aufgeteilt, dass kein ganzer Prozess mehr hineinpasst
- Dieses Problem ist nur mit einer echten **Virtualisierung des Speichers** lösbar, die nicht mehr ganze Prozesse sondern nur noch einzelne Speicherbereiche definierter, einheitlicher Größe auslagert:
- **Paging**
- Dafür ist allerdings **leistungsfähige Hardwareunterstützung** nötig

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion per Hardware: Die MMU

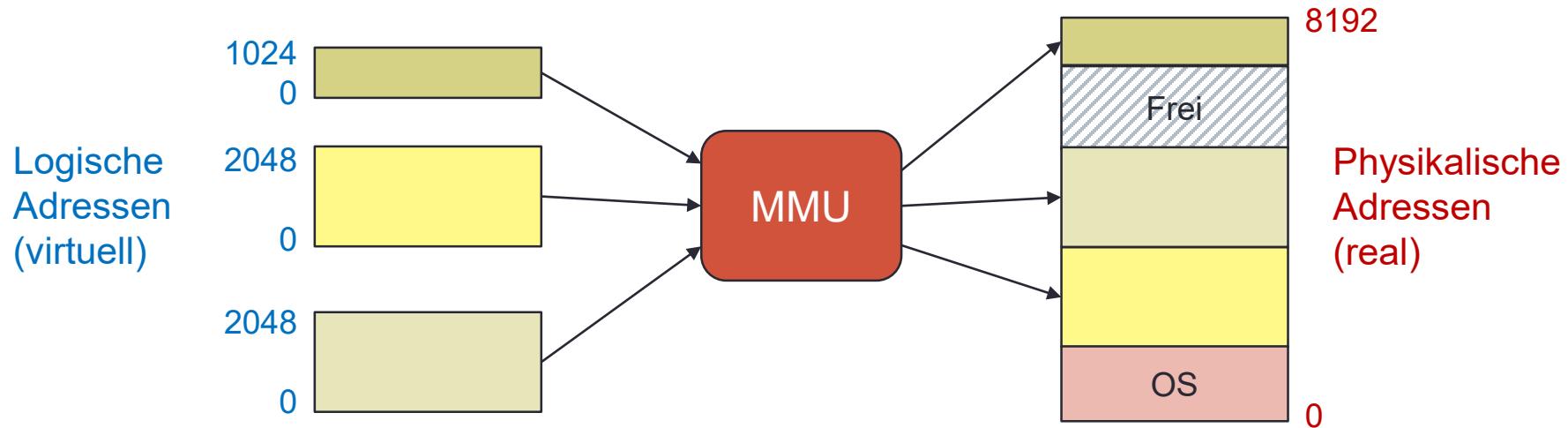
- Mit größer werdendem logischen Adressraum (32 Bit), steigender Anzahl nebenläufiger Prozesse und Fragmentierung durch Swapping wurde für die Abbildung logischer auf physikalische Adressen eine **Hardwarelösung** etabliert
- Die **Memory Management Unit** oder **MMU**
- Die MMU war zunächst eine externe Komponente zwischen CPU und Speicher
- Sie wurde erst im Laufe der Entwicklung in die CPU überführt
- In aktuellen CPUs sind mehrere MMUs auf dem Chip integriert

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion per Hardware: Die MMU

- Die MMU übernimmt die gesamte Adressübersetzung aus den **logischen Adressen** der Prozesse in die **physikalischen Adressen** des Speichers im Computer

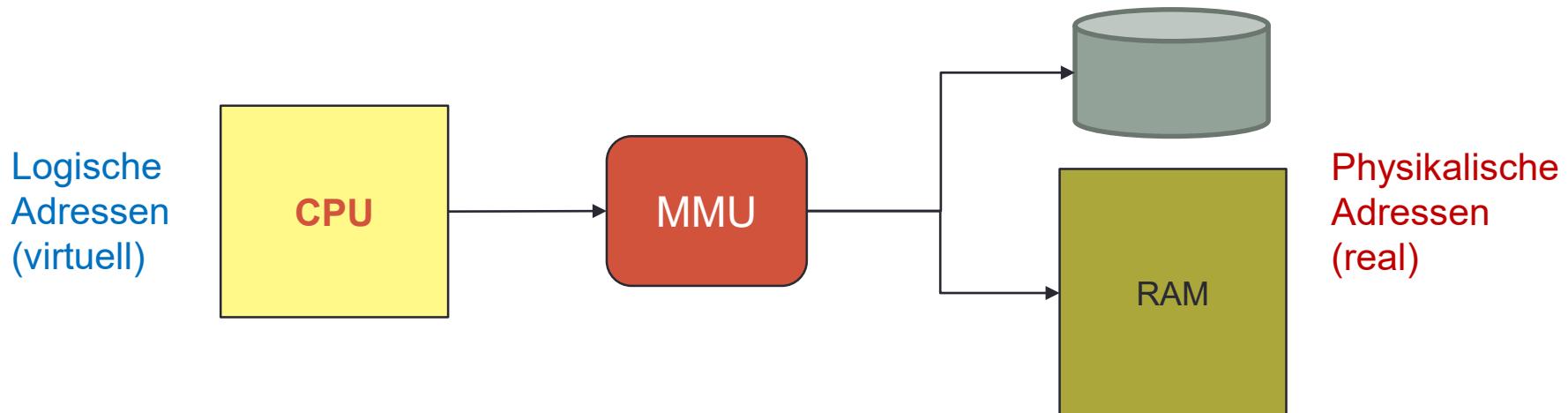


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion per Hardware: Die MMU

- Dabei bezieht die MMU auch Speicherbereiche auf der Festplatte mit ein, die über Vermittlung des Betriebssystems angesprochen werden

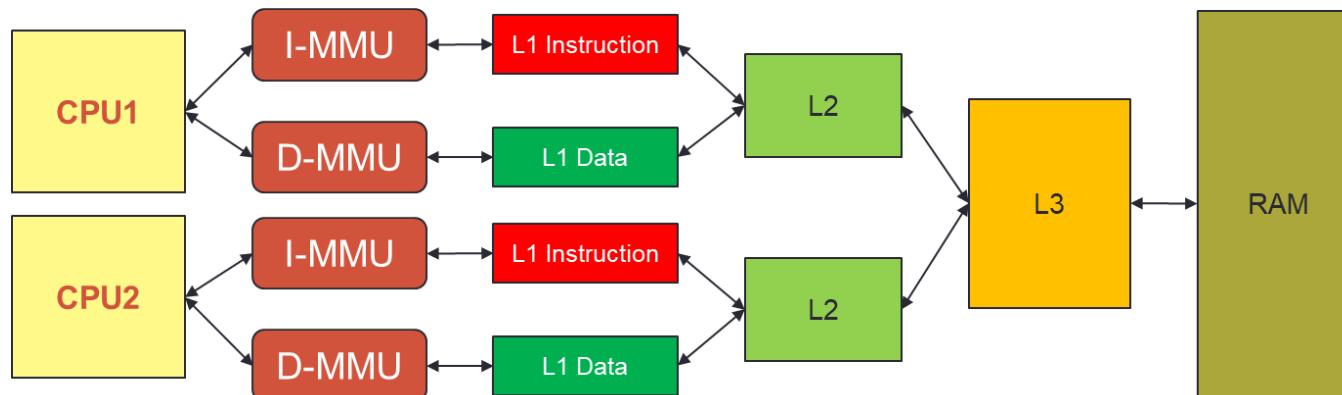


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion per Hardware: Die MMU

- Physikalisch ist die MMU meist zwischen der CPU und dem L1-Cache angeordnet
- In der CPU werden nur noch logische (virtuelle) Adressen verarbeitet



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Speicherabstraktion per Hardware: Die MMU

- Die MMU entkoppelt den logischen Adressraum der Prozesse endgültig vom physikalischen Speicher
- Für CPU, Prozesse und Programmierer ist diese Entkopplung in erster Näherung völlig transparent, es gibt nur noch **virtuelle Adressen**
- Das Resultat aus Sicht der Prozesse ist ein rein **virtueller Speicher**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

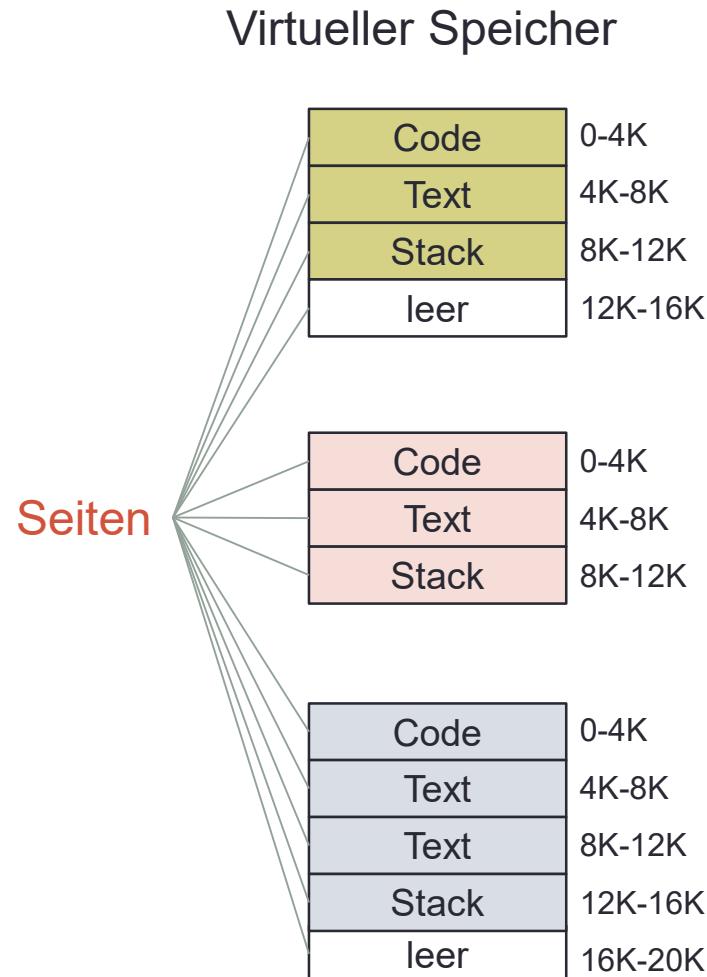
- Prozesse erwarten einen **zusammenhängenden Adressraum**
- Swapping hat deshalb ganze Prozesse ausgelagert – mit langen Wartezeiten und dem Problem der Fragmentierung
- Tatsächlich ist aber zu jedem Zeitpunkt selbst bei laufendem Prozess nicht dessen gesamter Arbeitsspeicherinhalt in Benutzung
- Könnten nur inaktive Teilbereiche des Adressraums eines Prozesses ausgelagert werden, ließen sich Wartezeiten reduzieren und die Fragmentierung besser steuern
- Genau diese Lösung bietet **Paging**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Virtueller Speicher ist untrennbar mit **Paging** verbunden
- Der gesamte virtuelle Adressraum für Prozesse und CPU wird in **Pages** oder **Seiten** eingeteilt
- **Seiten** sind **zusammenhängende Adressblöcke** von fester Größe
- Jeder Prozess belegt eine seinem Speicherbedarf entsprechende Anzahl an Seiten

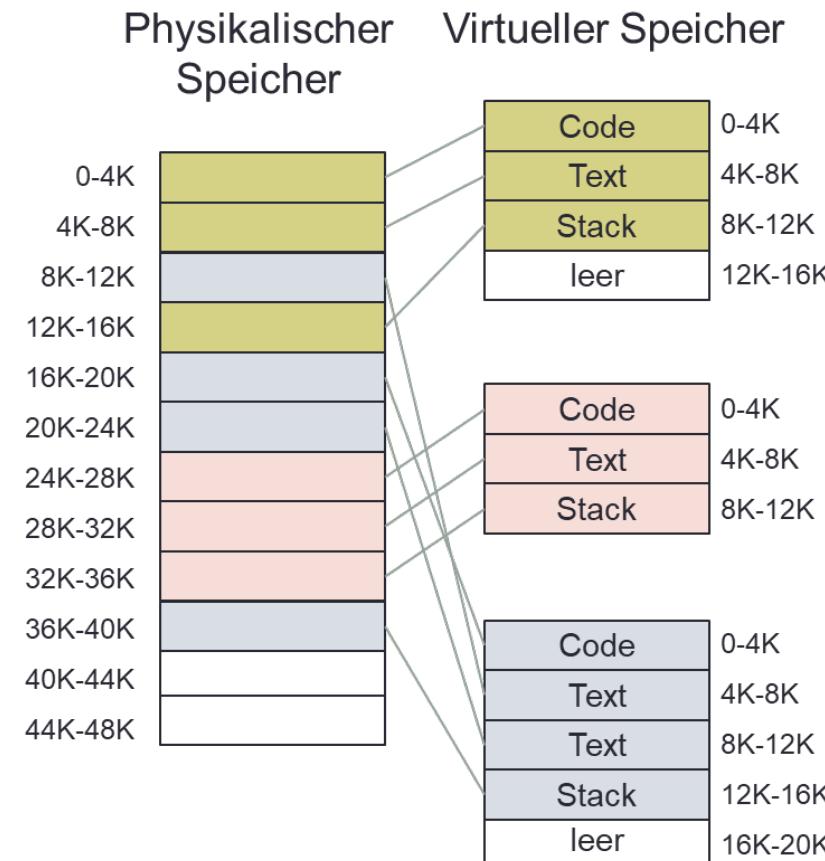


# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Seiten besitzen z.B. 4K Größe
- Aktuelle Prozessoren unterstützen allerdings mehrere Seitengrößen zwischen 4K und z.B. 16M
- Seiten des virtuellen Adressraums werden auf **Page Frames** oder **Seitenrahmen** im physikalischen Speicher abgebildet
- Seitenrahmen und Seiten sind genau gleich groß



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Die Auslagerung von nicht benötigten Seiten wird als **Paging** bezeichnet
- Mit nicht benötigten Seiten können statt ganzer Prozesse auch nur aktuell nicht benötigte **Prozessteile** ausgelagert werden
- Das Aus- und Einlagern einzelner Seiten ist wesentlich schneller als bei ganzen Prozessen

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Die Abbildung mit Seiten fester Größe unterbindet externe Fragmentierung, weil beim Aus- und Einlagern nur passende Lücken entstehen
- Allerdings ergibt sich eine **versteckte** oder **interne Fragmentierung**, wenn der benötigte Speicherplatz kein ganzzahliges Vielfaches der Seitengröße ist
- Dann müssen teilgefüllte Seiten genutzt werden, ungenutzter Bereich innerhalb von Seiten muss in Kauf genommen werden

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

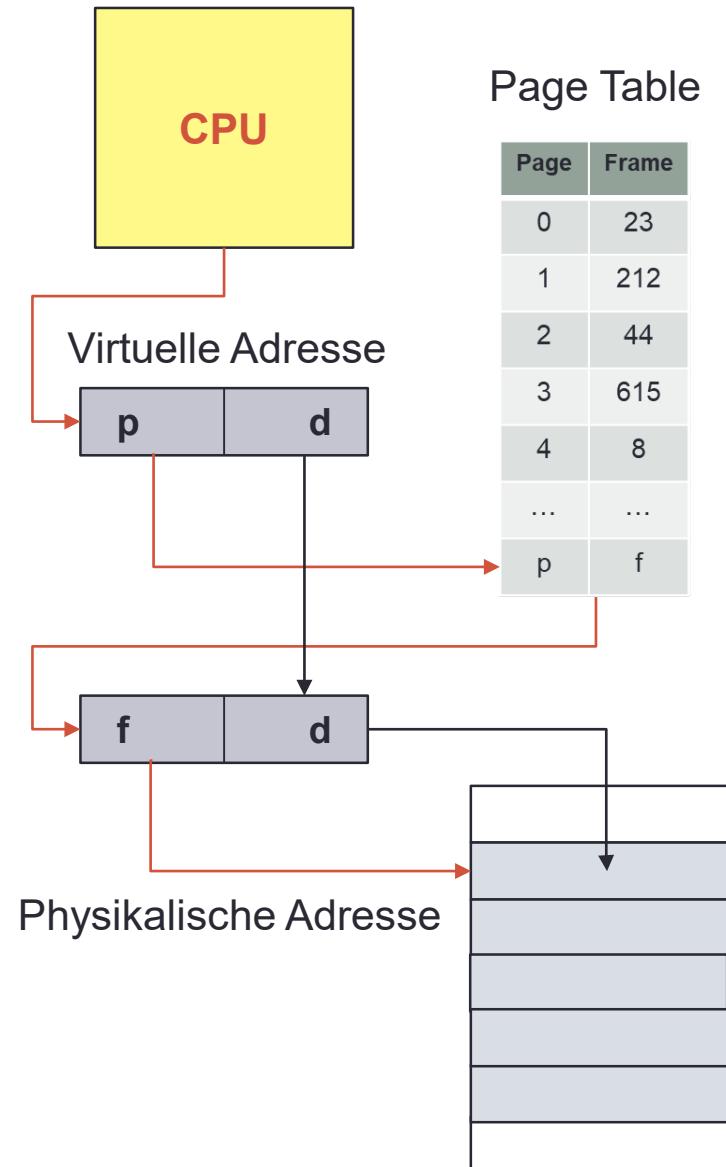
- Die MMU benötigt eine stets aktuelle Übersicht über die genutzten und ungenutzten Seiten sowie die Beziehung zwischen virtuellem und physikalischen Speicherort
- Diese Übersicht ist die **Page Table** oder **Seitentabelle**
- In der Seitentabelle werden virtuelle **Seitennummer** und zugehörige physikalische **Rahmennummer** gespeichert
- Die **exakte Speicheradresse innerhalb einer Seite** braucht bei Abbildung nicht festgehalten zu werden, sie **kann unverändert als Offset weitergegeben werden**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Von der CPU kommt die virtuelle Adresse einer Seite im Format **Seitenummer (p) – Offset (d)**
- Die MMU schreibt **p** in die Page Table, ordnet eine **Framenummer f** zu und reicht diese mit dem **Offset d** an den RAM weiter
- In der Seitentabelle werden weitere Daten gespeichert, u.a. zur Nutzung einer Seite oder Berechtigungen



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

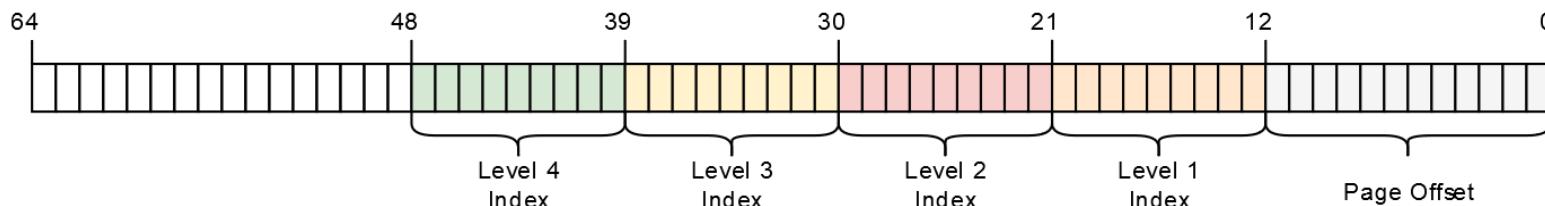
- Seitentabellen können aufgrund ihrer Größe kaum in Registern untergebracht werden und liegen daher im Hauptspeicher
- Bei großen Speicherbereichen wird die Seitentabelle sehr groß
- Große Tabellen sind schwer durchsuchbar und belegen viel Speicherplatz
- Aktuelle Prozessoren verwenden **Multi Level Page Tables**, also verkettete Seitentabellen in mehreren Ebenen

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Die von AMD für 64-Bit-Prozessoren eingeführte und später von Intel übernommene „Page Map Level 4“ -Technologie besitzt **vier Seitentabellen mit je 512 ( $2^9$ ) Einträgen**
- Die **Seitennummer p** umfasst alle  $4 \times 9 = 36$  Bit der vier Tabellenebenen
- Mit dem **Offset d** von 12 Bit ergibt sich ein 48-Bit Adressraum und damit 256 TB adressierbarer Speicher



Quelle: os.phil-opp.com/paging-introduction

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

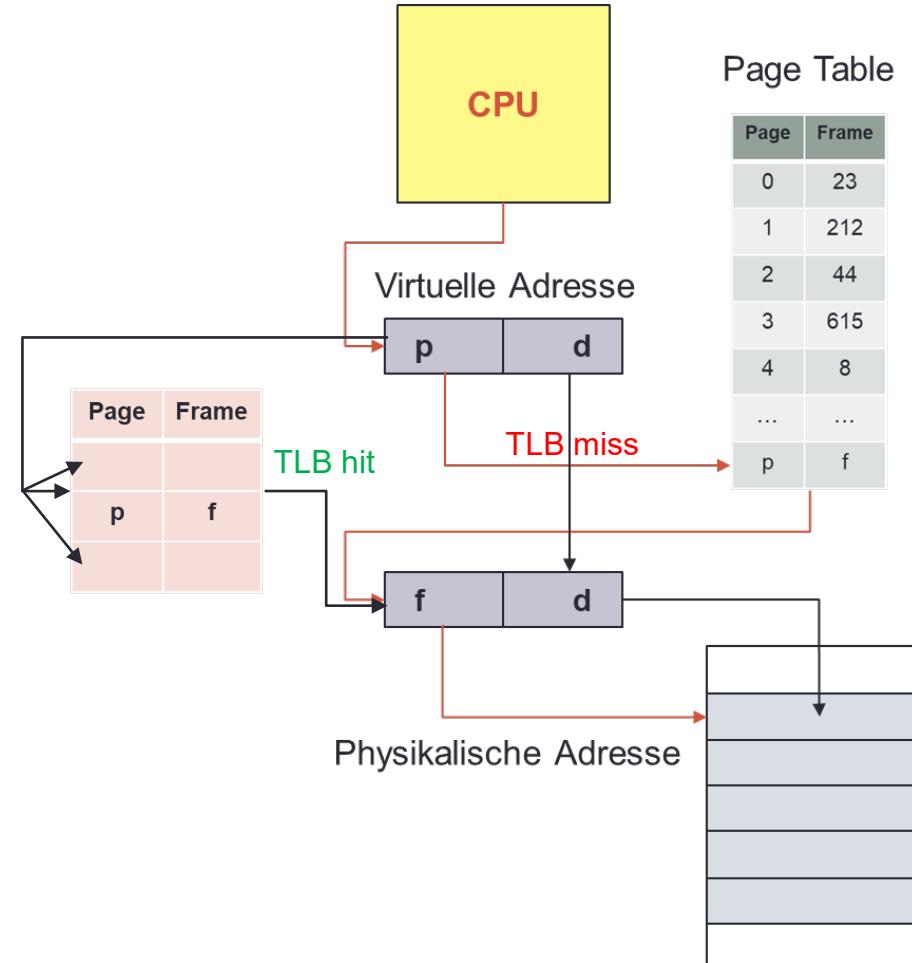
- Dennoch benötigen das Ausführen oder das Auslesen einer Adressübersetzung vier Schritte, je eine pro Tabelle
- Zur Beschleunigung dieser Vorgänge kommt daher der „übliche Verdächtige“ zum Einsatz: **Caching**
- Die MMU besitzt einen zusätzlichen Speicher, den **Translation Lookaside Buffer** oder **TLB** mit meist weniger als 256 Einträgen

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Der TLB kann **alle seine Einträge parallel** vergleichen und ist daher viel schneller als eine Seitentabelle
- Wird im TLB eine Abbildung gefunden (**TLB hit**), kann sie sofort an die MMU übergeben werden
- Andernfalls (**TLB miss**) sucht die MMU in der Page Table



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Virtueller Speicher mit Paging ermöglicht eine weitere Abstraktion des physikalischen Speichers
- Nicht ganze Prozesse werden ausgelagert, sondern nur ungenutzte Seiten als Teilbereiche des virtuellen Speichers eines Prozesses
- Paging wird durch eine MMU und TLB-Caching beschleunigt
- Das Auslagern ist so schneller und vermeidet externe Fragmentierung
- Was aber passiert, wenn eine Seite benötigt wird, deren zugeordneter Seitenrahmen ausgelagert ist und sich nicht im RAM befindet?

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

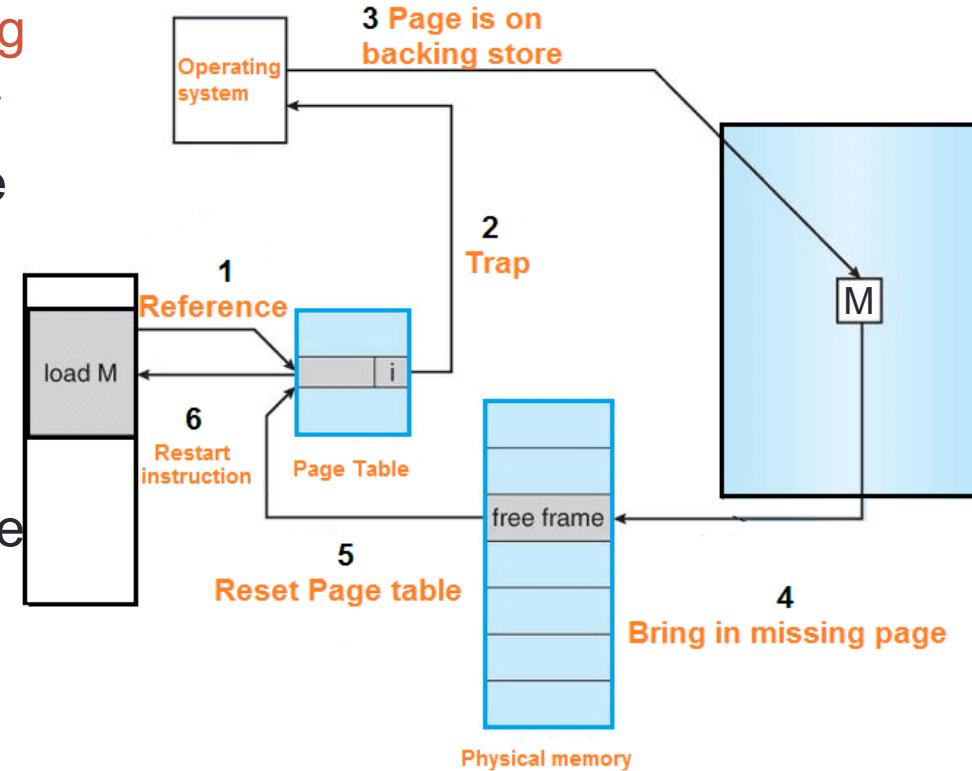
- Fordert die CPU eine Seite an, die nicht im RAM sondern auf die Festplatte ausgelagert ist, generiert die MMU einen **Page Fault** oder **Seitenfehler** und der Prozess wird angehalten
- Das Betriebssystem findet einen freien Rahmen oder es muss nun zunächst Platz schaffen, indem ein anderer Rahmen aus dem RAM auf die Festplatte kopiert wird
- Anschließend muss das OS den Inhalt des angeforderten Rahmens auf der Festplatte suchen, auslesen und als Seite in den RAM laden
- Danach wird der angehaltene Prozess fortgesetzt und der letzte Befehl erneut ausgeführt

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher

- Die MMU erkennt einen Page Fault und gibt dies als Trap aus (1-2)
- Das Betriebssystem führt Schritte 3 und 4 aus
- Die MMU aktualisiert die Page Table und übergibt den Inhalt der CPU (5-6)



# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsstrategien

- Beim Auftreten eines Seitenfehlers und voll ausgelastetem RAM muss das Betriebssystem entscheiden, welche Seite im RAM es durch eine neu angeforderte Seite ersetzen soll
- Diese Entscheidung zur **Seitenersetzung** übernehmen Algorithmen
- Ein optimaler Seitenersetzungsalgorithmus würde eine Seite auf die Festplatte auslagern, die erst in ferner Zukunft wieder genutzt wird
- Dann ist die Zeit bis zum nächsten Seitenfehler so lang wie möglich und damit die Anzahl der Seitenfehler minimal

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsstrategien

- Die **zukünftige Seitennutzung** ist aber **nicht vorhersehbar**, der optimale Seitenersetzungsalgorithmus ist **daher nicht realisierbar**
- Folglich muss ein Seitenersetzungsalgorithmus eine **Annahme** treffen, **welche Seite vermutlich länger nicht benutzt wird**
- Für diese Annahmen wertet er verschiedene Angaben zu den Seiten aus, die er selbst pflegen oder der Seitentabelle entnehmen kann
- Der Algorithmus sollte außerdem leicht verständlich, einfach zu implementieren und möglichst effizient sein

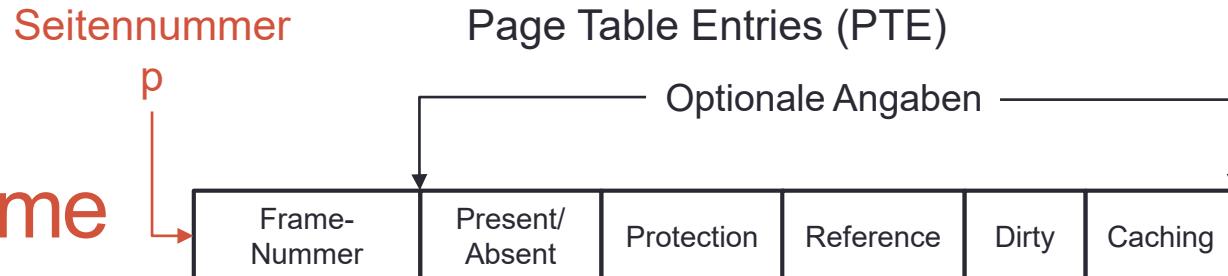
# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsstrategien

- Einige mögliche Realisierungen für die Seitenersetzung:
  - **First In First Out:** Ersetze die älteste Seite
  - **Second Chance** und **Clock:** Prüfe vor dem Ersetzen ein zweites mal
  - **Not recently Used:** Ersetze, was in letzter Zeit nicht benutzt wurde
  - **Least Recently Used:** Ersetze, was am längsten nicht benutzt wurde
  - **Not Frequently Used:** Software-Variante von LRU
  - **WSClock:** Ersetze Seiten, die nicht zum aktuellen Arbeitssatz des Prozesses gehören

# Betriebssysteme

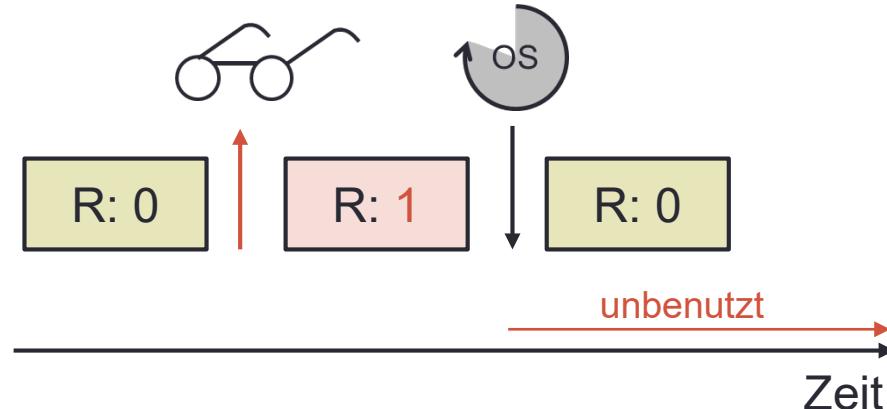


Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsstrategien

- Einige Algorithmen bewerten Seiten anhand ihrer Einträge in der Seitentabelle
- Eine typische Seitentabelle besitzt u.a. folgende Einträge:
  - **Rahmennummer f**
  - **Present/Absent Bit (Valid)**, kennzeichnet, ob die Page im RAM ist (1)
  - **Protection Bits**, Schutz vor Lesen/Schreiben/Ausführen RWX
  - **R-Bit für reference**, kennzeichnet einen **Seitenzugriff**
  - **M-Bit für modified**, kennzeichnet eine **Seitenänderung** (dirty Bit)
  - **Caching-Bit**, erlaubt oder verbietet das Caching der Seite

# Betriebssysteme



Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsstrategien

- Das **R-Bit** jeder Seite ist beim Start eines Prozesses 0 und wird bei dessen erstem Zugriff auf diese Seite auf **1** gesetzt
- In regelmäßigen Abständen (z.B. Timer Interrupt) setzt das Betriebssystem alle R-Bits wieder zurück
- Ein **R-Bit = 0** zeigt also, dass eine Seite seit dem letzten Zurücksetzen nicht wieder benutzt wurde (und so entbehrlich ist)
- Das **M-Bit** wird vor allem zur Kennzeichnung von geänderten Seiten verwendet, damit die Änderung noch in die Pagefile auf der Festplatte geschrieben werden kann (passiert in der Regel später)

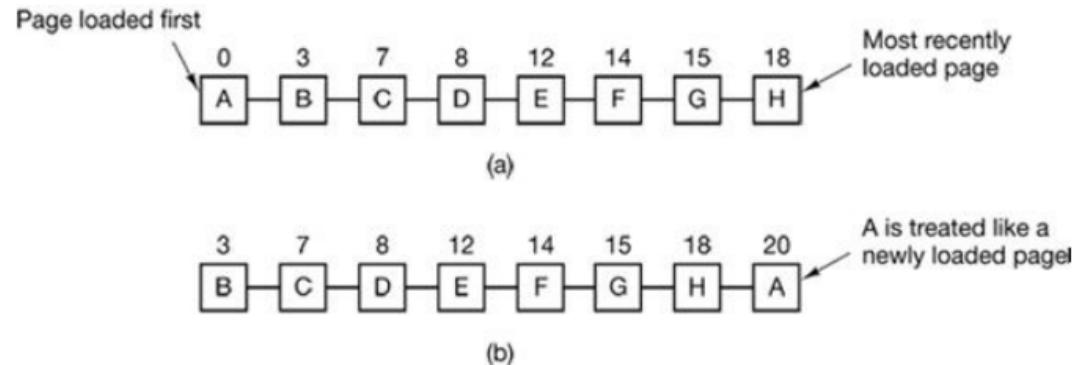
# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der **First-In-First-Out**-Algorithmus kann zur Entscheidung über die nächste zu ersetzende Seite das **Alter der Seite** heranziehen
- Eine **verkettete Liste** mit den Startzeiten aller Seiten enthält die älteste Seite am Anfang
- **FIFO** ersetzt diese älteste Seite
- **Problem: Das Alter einer Seite sagt nichts darüber aus, ob sie nicht sofort wieder benötigt wird**
- **FIFO** ist also einfach, kann aber wichtige Seiten entfernen, die dann sofort wieder gebraucht und nachgeladen werden müssten

# Betriebssysteme



Quelle: Tanenbaum und Bos (2016)

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

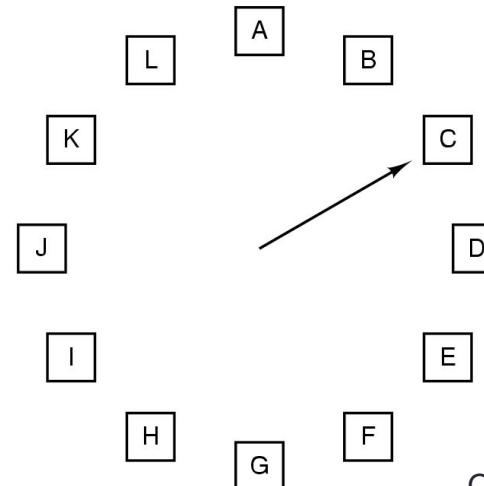
- Um festzustellen, ob eine schon alte Seite noch gebraucht wird, kann **FIFO** zu **Second Chance** modifiziert werden
- Die älteste Seite wird gesucht und dann geprüft, ob das R-Bit 0 ist
- Wenn ja, kann die Seite ersetzt werden
- Wenn nein, setze das R-Bit auf 0, hänge diese Seite mit der aktuellen Zeit ans Ende der Liste und suche weiter
- Wenn beim ersten Durchlauf keine geeignete Seite gefunden wird, taucht beim zweiten Durchlauf die ehemals älteste Seite wieder auf
- Jetzt ist das R-Bit 0 und die Seite wird ersetzt

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Eine effizientere Variante des **Second-Chance-Algorithmus** ist der **Clock-Algorithmus**
- Anstelle der verketteten Liste wird eine ringförmige Liste geführt
- Das weitere Vorgehen entspricht dem Second-Chance-Algorithmus
- Der Vorteil ist, dass die Reihenfolge der Einträge in der Liste nicht verändert werden muss



When a page fault occurs,  
the page the hand is  
pointing to is inspected.  
The action taken depends  
on the R bit:

R = 0: Evict the page  
R = 1: Clear R and advance hand

Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der **Not Recently Used**-Algorithmus wertet alle vier möglichen Kombinationen von R- und M-Bit aus und bildet vier Klassen:
  1. 00, nicht gelesen und nicht modifiziert
  2. 01, nicht gelesen, aber modifiziert
  3. 10, gelesen, aber nicht modifiziert
  4. 11, gelesen und modifiziert
- Eine Seite aus der **niedrigsten belegten Klasse** wird bei Bedarf ersetzt, soweit möglich eine, die seit dem letzten Reset der R-Bits **weder gelesen noch modifiziert** wurde

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der **Least-Recently-Used**-Algorithmus LRU versucht, aus der Nutzung einer Seite in der Vergangenheit auf die Zukunft zu schließen
- **Annahme:** Eine Seite, die bisher lange nicht genutzt wurde, wird wohl auch zukünftig nicht so schnell gebraucht
- **Also:** Es wird die Seite entfernt, die am längsten nicht gebraucht wurde
- **Herausforderung:** Wie die Nutzungshäufigkeit ermitteln?

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Lösung: Einen **Zähler** einrichten, der bei jedem Zugriff auf eine Seite inkrementiert wird
- Dabei muss bei **jedem Seitenzugriff** die ganze **Liste durchsucht** und der betreffende Eintrag aktualisiert werden
- => **ineffizient und praktisch nicht durchführbar**
  
- Es existieren zwei machbare Alternativen:
- **Zähler in der Hardware**, entweder CPU oder MMU: selten realisiert
- **Zähler in der Software (NFU)**

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der Zähler kann softwareseitig an LRU angenähert werden: **Not Frequently Used**
- Jede Seite bekommt in der Seitentabelle einen Zähler
- Das R-Bit wird periodisch zurückgesetzt wie bei Not Recently Used
- Vor dem Zurücksetzen des R-Bit wird aber dessen Wert ausgelesen und dem Zähler hinzugefügt
- Bei einer Seitenersetzung wird die Seite gewählt, die den kleinsten Zählerstand hat

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der Zähler kann softwareseitig an LRU angenähert werden **Not Frequently Used**
- **NFU** hat jedoch folgende Nachteile:
  - Ganz neue Seiten können gelöscht werden, weil sie einen niedrigen Zählerstand besitzen
  - Sehr alte Seiten, die nur am Anfang eines Prozesses viel genutzt wurden und danach nicht mehr, werden nie gelöscht (hoher Zähler)
  - Diese Probleme werden durch **Aging** umgangen: Das Alter der Zugriffe wird berücksichtigt, indem es den Zähler reduziert

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Aging verändert die Bewertung des Zählerstandes:
- Dividiere den Zählerstand durch 2 (Verschiebung um ein Bit nach rechts)
- Ersetze das höchstwertige Bit durch den aktuellen Stand des R-Bits

Konsequenz:

- Hohe alte Zählerstände werden zügig kleiner
- Aktuelle Nutzung (R-Bit =1) erhöht Zählerstand deutlich

Zyklus	R-Bit	Zähler
1	1	100000 = 32
2	0	010000 = 16
3	0	001000 = 8
4	0	000100 = 4
5	1	100010 = 34

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- **WSClock** ersetzt Seiten eines Prozesses, die zwar im RAM stehen, aber nicht zum aktuellen **Working Set** des Prozesse gehören

Definition **Working Set** oder **Arbeitssatz**:

- Prozesse nutzen meist nur einen kleinen Anteil ihrer Seiten im Arbeitsspeicher
- Dieser Anteil ist der **Working Set** oder **Arbeitssatz**, also das Werkzeug aus der Werkzeugkiste, das der Prozess tatsächlich regelmäßig braucht

# Betriebssysteme

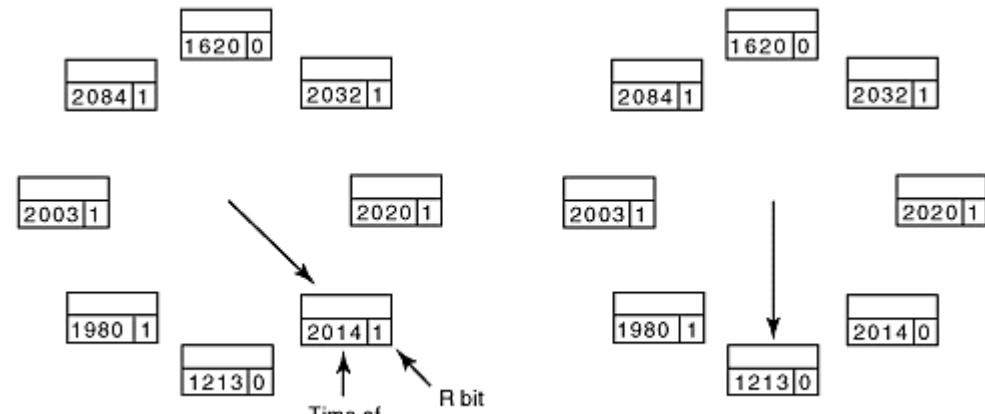
Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Demand Paging wird eingesetzt: Seiten werden erst geladen, wenn der Prozess sie braucht
- Anfangs erzeugt fast jede neue Instruktion Seitenfehler; nach einiger Zeit hat der Prozess das Meiste beisammen, irgendwann alles
- Um den aktuellen Arbeitssatz eines Prozesses zu ermitteln, können alle in einem Zeitintervall T referenzierten Seiten gesucht werden
- Nur solche Seiten gehören zum Arbeitssatz
- Zeit meint hier **virtuelle Zeit**, also nur die CPU-Zeit, die dem Prozess tatsächlich zur Verfügung stand

# Betriebssysteme

Current virtual time = 2204    T = 800



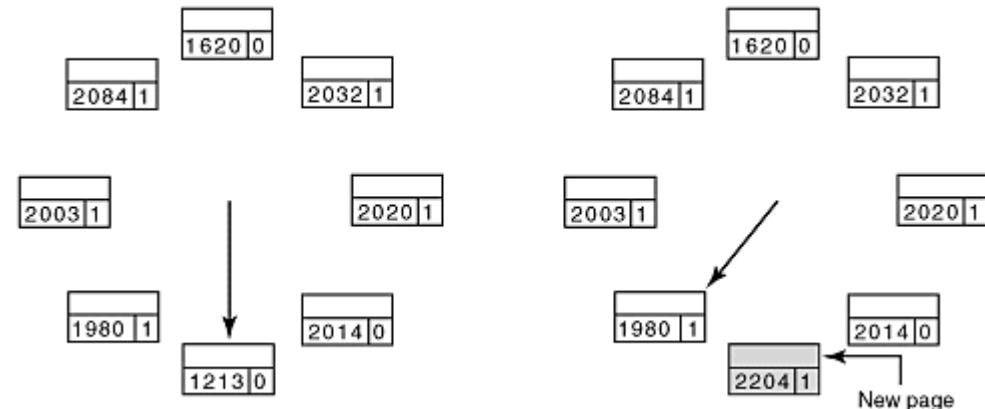
Quelle: Tanenbaum und Bos (2016)

## Aufgaben – Speicherverwaltung

## Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Der WSClock-Algorithmus erstellt eine ringförmige Seitenliste
- Die Liste enthält alle genutzten Seiten mit einem Zeitstempel und dem R-Bit, neue Seiten kommen mit ihrem Zeitstempel hinzu
- Bei einem Seitenfehler prüft der Algorithmus zuerst den Status des R-Bits für die Seite, auf die der Zeiger gerade zeigt
- Ist das R-Bit=1, befindet sich die Seite im Arbeitssatz, weil sie innerhalb des letzten Intervalls genutzt wurde
- Die Seite wird nicht entfernt, aber das R-Bit zurückgesetzt und die Suche fortgesetzt

Current virtual time = 2204     $T = 800$



# Betriebssysteme

Aufgaben – Speicherverwaltung

Quelle: Tanenbaum und Bos (2016)

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Ist das R-Bit=0, wird die Zeit geprüft
- Bei  $< T$  wird die Seite noch einmal verschont und weitergesucht
- Bei  $> T$  wird angenommen, dass sie sich nicht mehr im Arbeitssatz befindet
- Ist das M-Bit in der Seitentabelle 0, ist die Seite sauber
- Sie kann gelöscht und durch die neue aktuelle Seite ersetzt werden
- Ist das M-Bit in der Seitentabelle 1, muss die offenbar veränderte Seite zuerst auf die Festplatte geschrieben werden
- In diesem Fall muss der Algorithmus weitersuchen

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Kann WSClock keine Seite finden, die nicht zum Arbeitssatz gehört, wählt er eine beliebige Seite aus und ersetzt diese

Welcher Seitenersetzungsalgorithmus bietet beste Leistung?

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher, Seitenersetzungsalgorithmen

- Laut Lehrbuch sind **Aging** und **WSClock** aktuell die leistungsfähigsten Seitenersetzungsalgorithmen, die beide auf LRU beruhen

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher: Weitere Aspekte

- Die Seitenersetzung kann sich für neue Frames entweder beim eigenen Prozess bedienen oder alle Prozesse in die Suche einbeziehen
- **Lokale Suche** im eigenen Prozess kann bei knappem RAM dazu führen, dass der Prozess ständig Seitenfehler erzeugt (Thrashing) und nicht mehr vorwärts kommt
- Dafür ist die **globale Suche** bei allen laufenden Prozessen besser geeignet

# Betriebssysteme

Aufgaben – Speicherverwaltung

Methoden – Virtueller Speicher: Weitere Aspekte

- Wenn der RAM eher unterdimensioniert ist, kann Paging mit **Swapping** kombiniert werden, um den RAM von größeren nicht genutzten Seitenmengen zu befreien und bei den laufenden Prozessen die Seitenfehler zu reduzieren
- Ebenso ist es möglich, Paging mit dem **Segmentieren** zu verknüpfen, um anwachsenden Prozessen neue Adressbereiche zuzuweisen
- Allerdings wurde dies selten eingesetzt und daher in den aktuellen 64-Bit-Prozessoren von Intel auch nicht mehr unterstützt

# Betriebssysteme

## Aufgaben – Speicherverwaltung – Zusammenfassung

- Die Speicherverwaltung versucht, die **Ressource Arbeitsspeicher** möglichst effizient allen konkurrierenden Prozessen zuzuteilen
- Dabei wird das OS inzwischen von der **MMU als Hardware** unterstützt
- Neben der Auslagerung ganzer Prozesse per Swapping auf die Festplatte ist vor allem **virtueller Speicher** das wichtigste Konzept
- Virtueller Speicher arbeitet mit logischen Speichereinheiten, den **Pages** oder **Seiten**, die Prozesse und CPU vom physikalischen Speicher entkoppeln
- **Paging** und die Ersetzung von Seiten wird mit komplexen Algorithmen verwaltet, die einen erheblichen Aufwand bedeuten

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - Speicherverwaltung
  - **Dateisysteme**
  - Ein- und Ausgabe
  - Multiprozessorsysteme
  - Virtualisierung

Praxis: Umgang mit Linux (Prof. Brunner) und Windows Server 2019

# Betriebssysteme

Dateisysteme – Definitionen

Aus dem Einführungsteil:

- Dateien sind die Abstraktion des Betriebssystems für Speicherplatz auf einem nichtflüchtigen Speichermedium wie etwa einer Festplatte
- ?

# Betriebssysteme

## Dateisysteme – Definitionen

- **Dateien** sind die Abstraktion des Betriebssystems für Speicherplatz auf einem nichtflüchtigen Speichermedium wie etwa einer Festplatte
- Formal ist eine Datei eine **benannte, geordnete Sammlung von Informationen**, die auf einem sekundären Datenträger gespeichert ist
- Betriebssysteme verwenden spezifische Modelle dafür: Verschiedene **Dateisysteme**
- Ein Dateisystem stellt ein **Verzeichnis** dar, in dem Dateien erzeugt, verschoben, gespeichert, verändert und gelöscht werden können
- Spezielle Dateisysteme können sich auch über Netzwerke ausdehnen, **Netzwerkdateisysteme**

# Betriebssysteme

## Dateisysteme

### Dateien – Themen

- Warum Dateien und Dateisysteme?
- Wie sind Dateien aufgebaut und wie werden sie bezeichnet?
- Welche Eigenschaften besitzen Dateien?
- Welche wichtigen Dateisysteme gibt es und wie funktionieren sie?
- Wie funktioniert die Abbildung auf den physikalischen Datenträger?
- Welche Aufgaben hat das Betriebssystem im Zusammenhang mit der Dateiverwaltung?

# Betriebssysteme

## Dateisysteme

### Dateien – Warum?

- **Abstraktion:** Einfache Systemaufrufe für alle Dateioperationen, verfügbar für Applikationen, Dienste, GUI, Shell...
- Dateien speichern **große Mengen an Informationen**
- Informationen müssen **permanent gespeichert** werden, also auch das Abschalten der Energieversorgung überstehen
- Informationen müssen bei Bedarf **gemeinsam nutzbar** sein
- Informationen müssen **vor unbefugtem Zugriff geschützt** sein
- Im RAM nur teilweise möglich (Menge beschränkt, keine permanente Speicherung...)

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Dateien besitzen einen **strukturierten** oder **unstrukturierten Inhalt**
  - **Byte-Sequenz**, unstrukturierte Byte-Folge (Unix, Windows...)
  - **Baumstruktur**, strukturierte Einträge mit Schlüssel (IBM Mainframes)
- Inhalte können unterschiedlich sein, u.a.:
  - **Daten**, z.B. **ASCII-Zeichenfolgen**, die als Text angezeigt werden können, Zeilen oft durch Return getrennt
  - **Ausführbare Dateien**, also Source- oder Binärdaten, evtl. mit definierter Dateiendung bestimmtem Format und besonderen Metadaten
- Unter Unix gilt: Everything is a file, also auch Geräte...

CbeU/KfIW58q6L&#xA; cXGi6+NNNp9rJJJJaws7u0Mi  
m2f/JiPFU6xV2KuxV2KuxV2KuxV2KuxV2KuxV2Ksa8y  
xV2KuxV2KuxV2KuxV2KuxV2KuxV2KuxV2KuxV2KuxV.  
W4+tem0LIA0QCsa9MVY/&#xA; PY6pD5a1/wAnSaddXl  
Ir995gTWJ5JtSRI5reQKYAsw0XALbq441P7Xc1riqc-  
A; fJf5Tw+bv+Vo+Wv5Lr/kWv8A1Ux/k3L5L/KeHzd/)  
ZmeQRCKR9SnqMVAJ40Hvhz/wB2&#xA; UaevEDIL5bXl  
YTTS01z8QcrIFi+C6dzSR03ZF+yd2rsx0QmuXw/Ysh:  
+91&#xA; 7yr5a10xV2KtFFJqRgoJ4i5lqpAxI2UHdQj  
+Tvr9j/y0xf8jf/rj+Yxfzh81/L5f5p+&#xA; Tvr9j,  
jM0coLRQwc1k1Pw4FFjczzUw0KDjSTVGRFwqN0Nhfc!  
/t9/wDekpPhdBw8&#xA; DIbk0vvc9oIAstc9uojgpK  
S1530AsmSNPznFJTa/5wdE/7nUF9uD+9JSv+cHRP+5:  
kpXT/APk/G/4mv/qQkpsJKUkpSSml1Pql+nVsezFvy!

Nas ist Anwendungsadministration?

Umfangreiche Software-Applikationen für Fach-  
dass auch sehr gut qualifizierte IT-Administrat-  
volumfähiglich administrieren können. Die mit  
Aufgaben erfordern vertiefte Kenntnisse der A  
aber auch ein gutes Verständnis für die Platt-  
nutzen Funktionen des jeweiligen Betriebs-sys  
und Datenbankser-vern zusammen oder machen Ge  
Authentifizierung von Benutzern. Die Aufgaben  
Erstkonfiguration bei der Installation über U  
Schulung und Support sämtlicher Anwender. Si  
Erledigung nicht mehr ohne spezifische Schulu  
verteilt werden sollte.

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Für Dateien gibt es verschiedene Zugriffsmuster, aktuell vor allem:
- **Sequenziell:** Daten werden von Anfang bis Ende der Reihe nach gelesen
  - Schreiben neuer Dateien
  - Parsen vom HTML-Seiten
- **Random Access oder wahlfreier Zugriff:** Daten werden gezielt gesucht und direkt angesteuert
  - Datenbankzugriffe
  - Seitenersetzung beim Paging

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Dateien werden im Dateisystem oft mit eindeutigen Nummern identifiziert
- Für den Benutzer sind aber Dateinamen einfacher zu handhaben
- Dateien haben daher stets einen Namen
- Für die Bezeichnung von Dateien gibt es dateisystemabhängig verschiedene Konzepte, die teilweise nach heutigen Maßstäben unter erheblichen Beschränkungen leiden

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Ein altes und in seinen Möglichkeiten sehr beschränktes Konzept für Dateinamen bietet das für MS-DOS entwickelte Dateisystem FAT (File Allocation Table)
- Dateinamen können 11 Zeichen umfassen
- Der Name ist in zwei Abschnitte geteilt, den eigentlichen Dateibezeichner mit **maximal 8 Zeichen** und einer durch „.“ abgetrennten **Dateinamenserweiterung** (file extension) mit ursprünglich **3 Zeichen**:

Beispiel: **PROGRAMM.EXE**

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Relevant ist hier die Funktion der Dateinamenserweiterung oder Dateiendung: Mit ihr werden Dateiformat und Verknüpfung mit einem Programm definiert
- Das ist unter Windows noch heute so
- UNIX erlaubt zwar Dateiendungen, nutzt sie aber nicht
- Es gibt allerdings Konventionen: siehe Tabelle

Dateiendung	Bedeutung
.bak	Sicherungsdatei
.html	WWW-Seite
.jpg	JPEG-Bilddatei
.mp3	MPEG Musikdatei
.pdf	PDF-Datei
.zip	Komprimiertes Archiv
Windows	
.docx	MS Word Dokument
.exe, .bat, .com	Ausführbare Dateien
.txt	Simples Textdokument

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

- Manche Zeichen sind im Dateinamen möglicherweise unzulässig, weil sie Steuerfunktionen im Dateisystem haben
- Unter Windows insbesondere Sonderzeichen wie: \ : \* ? " < > |
- Auch weitere Sonderzeichen können entweder nicht von allen Programmen oder von Betriebssystemen nicht sauber verarbeitet werden: = [ ] " ; , \$ # & @ ! / ( ) - { } ' \_ ~ ^
- Leerzeichen dürfen in Dateinamen zwar vorkommen, aber nicht alle Programme und Schnittstellen können damit umgehen
- **Sonderzeichen und Leerzeichen also besser vermeiden**

# Betriebssysteme

## Dateisysteme

### Dateien – Aufbau und Namen

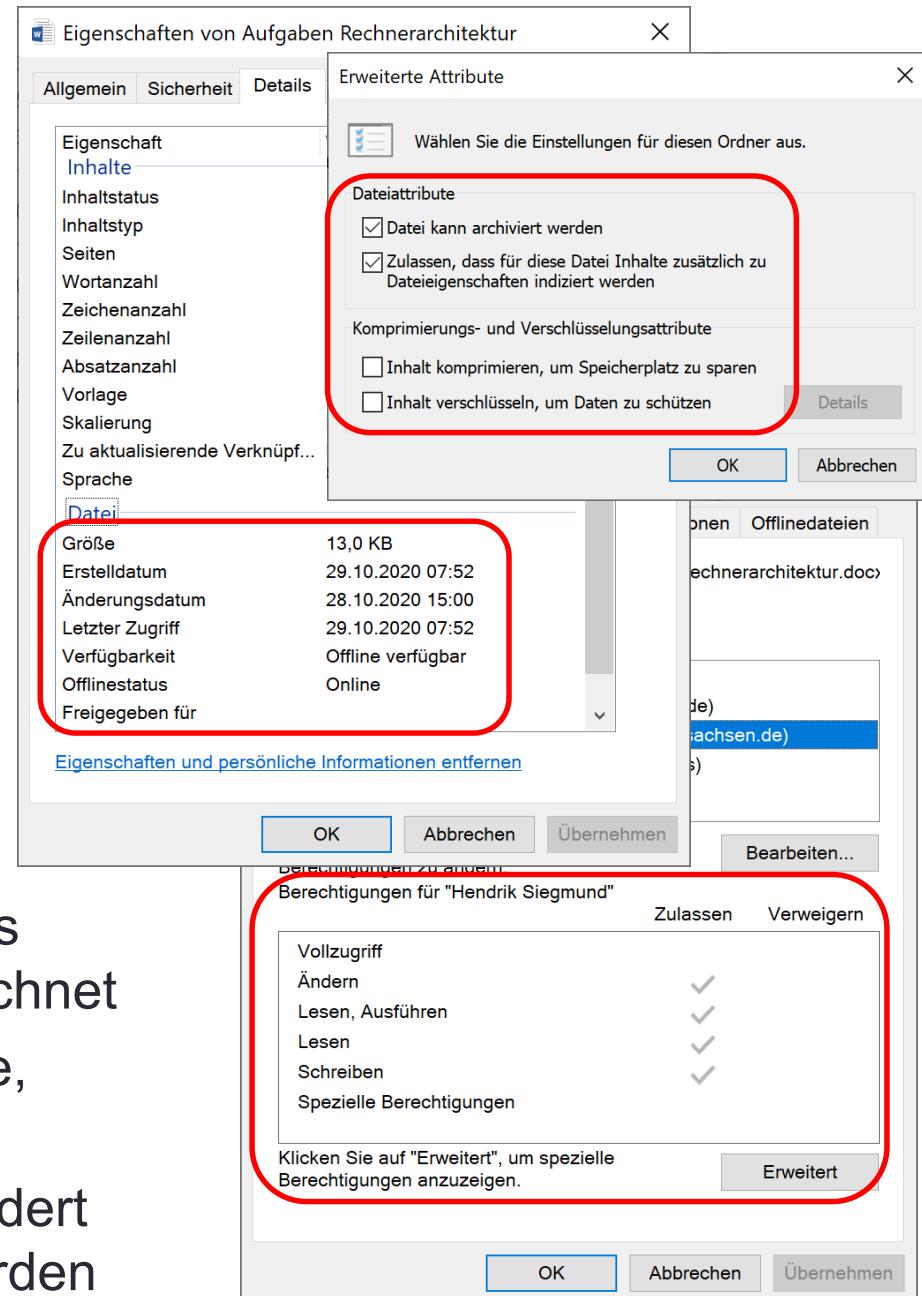
- Schließlich gibt es Regeln zur **maximalen Länge von Dateinamen** und zur **Pfadtiefe**, die zwischen den Betriebssystemen variieren können
- Unix mit ext2, ext3 und ext4: **255 Byte, Pfadtiefe 4096 Zeichen**
- Windows NTFS: **255 Zeichen, Pfadtiefe 65535 Zeichen, aber:**
- **Systemvariable MAX\_PATH = 255**, kann manuell angepasst werden
- FAT, FAT 16 und FAT32: **8.3**, mit aktuellen Erweiterungen auch mehr
  - Kameras schreiben Daten auf SD-Karten, die FAT-Varianten einsetzen
  - Die Dateinamen für Bilder folgen daher der 8.3-Regel

# Betriebssysteme

## Dateisysteme

### Dateien – Dateieigenschaften

- Zusätzlich zum Inhalt und dem Dateinamen verarbeiten alle Betriebssysteme weitere Dateieigenschaften
- Diese Eigenschaften werden als **Attribute** oder **Metadaten** bezeichnet
- Jedes OS nutzt andere Attribute, einige Beispiele aus Windows
- Attribute können gelesen, geändert und unterschiedlich genutzt werden



# Betriebssysteme

## Dateisysteme

### Dateien – Dateioperationen

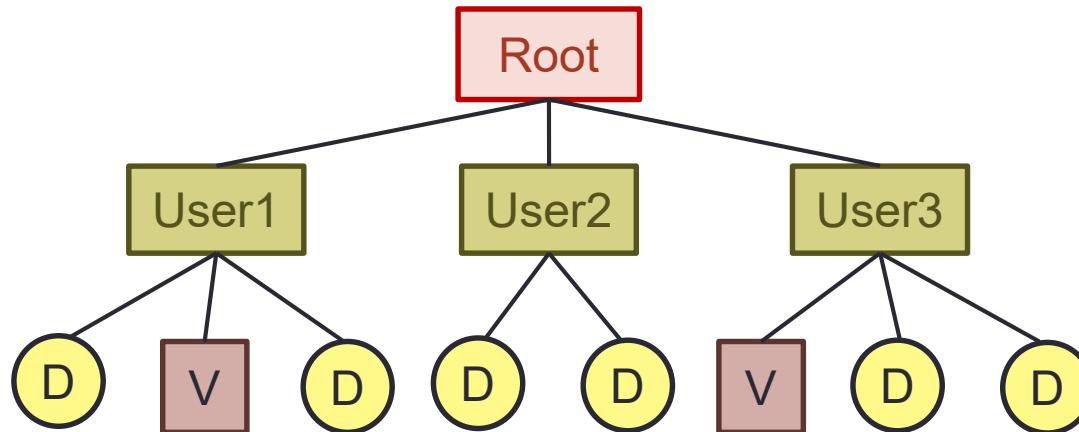
- Dateien werden von Anwendern genutzt, aber nur indirekt
- Nur das Betriebssystem arbeitet direkt mit Dateien
- Dazu bietet es Systemaufrufe, die alle wesentlichen Funktionen abdecken
- Systemaufrufe für Dateioperationen sind für jedes OS meist gut dokumentiert

Funktion	Beschreibung
Create	Erstellt leere Datei und Attribute
Open	Öffnet Datei zur Bearbeitung
Close	Beendet Arbeit an Datei
Delete	Löscht Datei vom Datenträger
Seek	Positioniert Zeiger an definierter Stelle in Datei
Read	Liest Daten von aktueller Position
Write	Schreibt Daten an aktueller Position
Get Attributes	Liest Attribute zur Prüfung und Verwendung der Datei
Set Attributes	Legt Attributwerte einer Datei fest

# Betriebssysteme

## Dateisysteme – Typen und Funktion

- Dateisysteme sind normalerweise hierarchisch aufgebaute Verzeichnisse, sie teilen sich in mehrere Ebenen auf
- Dateien werden im Verzeichnis mit **Pfaden** lokalisiert

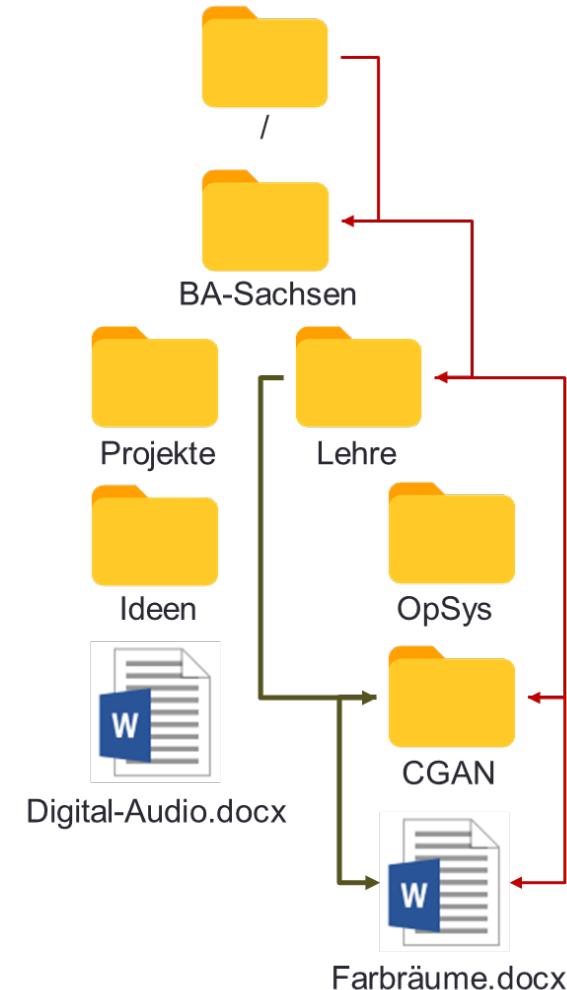


# Betriebssysteme

## Dateisysteme – Typen und Funktion

- Der **absolute Pfad** gibt den **Weg vom Stammverzeichnis bis zur Datei** an
- Der **relative Pfad** geht von einem **Verzeichnis unterhalb der Root** aus und gibt den **Weg von dort bis zur Datei** an
- Er kann nur genutzt werden, wenn der Weg bis zur gewünschten Datei nicht über eine höhere Verzeichnisebene führt :

Absolut: /BA-Sachsen/Lehre/CGAN/Farbräume.docx  
Relativ von Lehre aus: CGAN/Farbräume.docx



# Betriebssysteme

## Dateisysteme – Typen und Funktion

- Verzeichnisse im Dateisystem können wie Dateien bearbeitet werden
- Operationen auf Verzeichnissen werden mit ähnlichen Systemaufrufen durchgeführt wie bei Dateien

Funktion	Beschreibung
Create	Legt ein leeres Verzeichnis an
Opendir	Öffnet Verzeichnis z.B. zum Lesen von Dateien
Closedir	Schließt Verzeichnis und gibt Tabellenplatz im Speicher frei
Delete	Löscht (leeres) Verzeichnis
Link	Verknüpft Datei aus einem anderen Verzeichnis mit dem aktuellen Verzeichnis
Unlink	Hebt Verknüpfung wieder auf

# Betriebssysteme



Quelle: Intenso

## Dateisysteme – Abbildung auf Datenträger

- Festplatten als wichtigste sekundäre Speicher arbeiten mit einer **kleinsten Zuordnungseinheit** für Daten namens **Sektor**
- Sektoren besaßen lange eine Standardgröße von 512 Byte, inzwischen sind es jedoch 4096 Byte
- Auf Festplatte oder SSD kann immer nur **ein ganzer Sektor angesprochen** werden
- Sektoren sind damit die kleinsten physischen Dateneinheiten auf Festplatten
- Sektoren werden bei der heute üblichen logischen Blockadressierung LBA mit einer individuellen 48 Bit langen Nummer adressiert

# Betriebssysteme

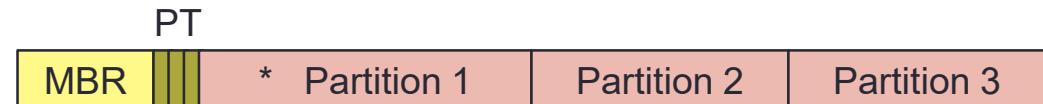
## Dateisysteme – Abbildung auf Datenträger

- Dateisysteme arbeiten ebenfalls kleinste Zuordnungseinheiten, die logischen **Blöcke**
- Die Größe der Blöcke unterscheidet sich zwischen Dateisystemen und ist nicht notwendigerweise identisch mit der Größe physischer Sektoren
- Allerdings haben logische Blöcke ebenfalls typische Größen zwischen 512 und 4096 Byte, um die Zuordnung zu vereinfachen
- Dateien in Dateisystemen werden in ganzen Blöcken betrachtet

# Betriebssysteme

## Dateisysteme – Abbildung auf Datenträger

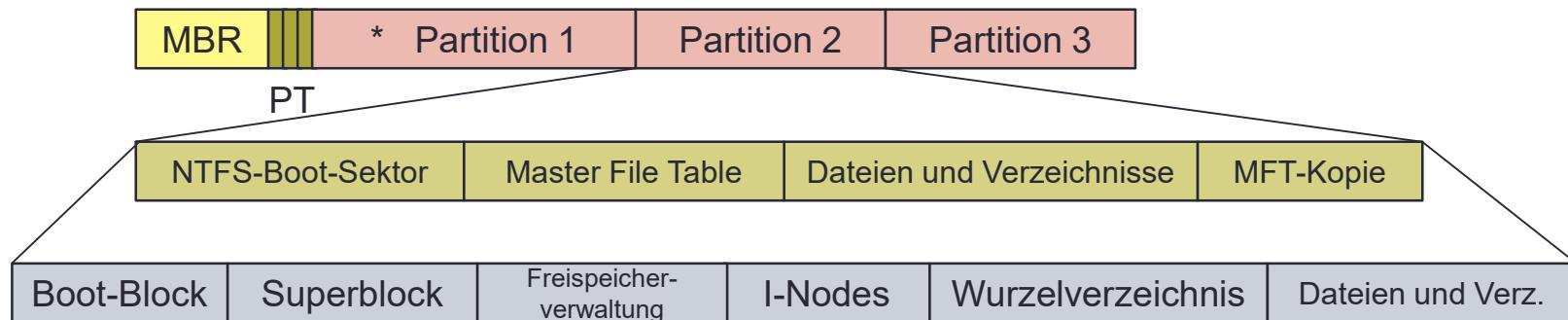
- Dateisysteme werden auf Festplatten klassisch in **Partitionen** gespeichert
- Partitionen sind logische Unterteilungen des physischen durch Sektoren gebildeten Speicherplatzes
- Der typische logische Aufbau einer Festplatte umfasst:
  - Master Boot Record MBR
  - Partitionstabelle PT
  - Partitionen



# Betriebssysteme

## Dateisysteme – Abbildung auf Datenträger

- Der **Master Boot Record** umfasst Sektor 0 und enthält Angaben zur Partition für den Start eines Betriebssystems
- Eine Partition ist als **aktiv markiert** \* und enthält in ihrem **Boot-Block** bzw. **Boot-Sektor** ausführbaren Code zum Laden eines Betriebssystems
- Aufbau und Inhalt einer Partition sind abhängig vom Dateisystem



# Betriebssysteme

## Dateisysteme – Abbildung auf Datenträger

- Die **Master File Table** ist das Inhaltsverzeichnis des Windows-Dateisystems NTFS und enthält mindestens einen Eintrag je Datei
- Darin sind u.a. auch die Attributwerte der Dateien gespeichert
- Die MFT umfasst standardmäßig 12,5% der Partition, die MFT-Kopie umfasst nur die ersten 16 Einträge zur MFT selbst
- Die MFT entspricht funktional dem I-Node unter Unix

### MFT-Inhalte

Speicherort
Dateigröße
Erstelldatum
Änderungsdatum
Schreibschutz
Versteckt
Zugriffsrechte
Freigaben
Dateityp
...

# Betriebssysteme

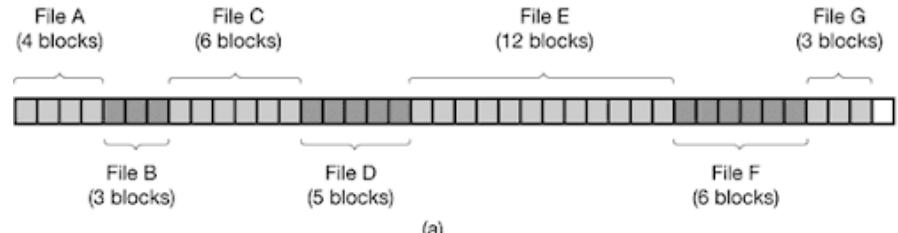
## Dateisysteme – Abbildung auf Datenträger

- Dateien werden im NTFS standardmäßig in **Blöcken** oder **Clustern** zu gespeichert, ab einer Partitionsgröße von 2 GB in **4096 Byte** Größe
- Bei älteren Festplatten nutzt ein solcher Block 8 Sektoren, auf aktuellen Platten und SSDs besteht eine 1:1 Beziehung zwischen diesen Blöcken und den Physikalischen Sektoren
- Das Betriebssystem **schreibt Dateien zusammenhängend** auf den Datenträger
- Das ergibt die schnellsten Schreib- und Leseraten, weil der Kopf nicht neu positioniert werden muss und in SSDs der Cache damit rechnet, das sequenziell ausgelesen wird

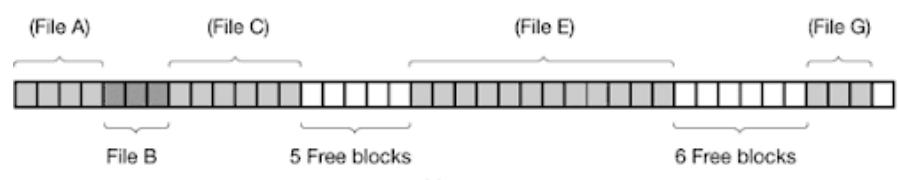
# Betriebssysteme

## Dateisysteme – Abbildung auf Datenträger

- Werden Dateien gelöscht oder verändern sie ihre Größe, können Lücken in der Folge der belegten Blöcke entstehen
- Diese **Fragmentierung** nimmt mit der Zeit zu, auch wenn neue Dateien immer wieder in passende Lücken geschrieben werden



(a)



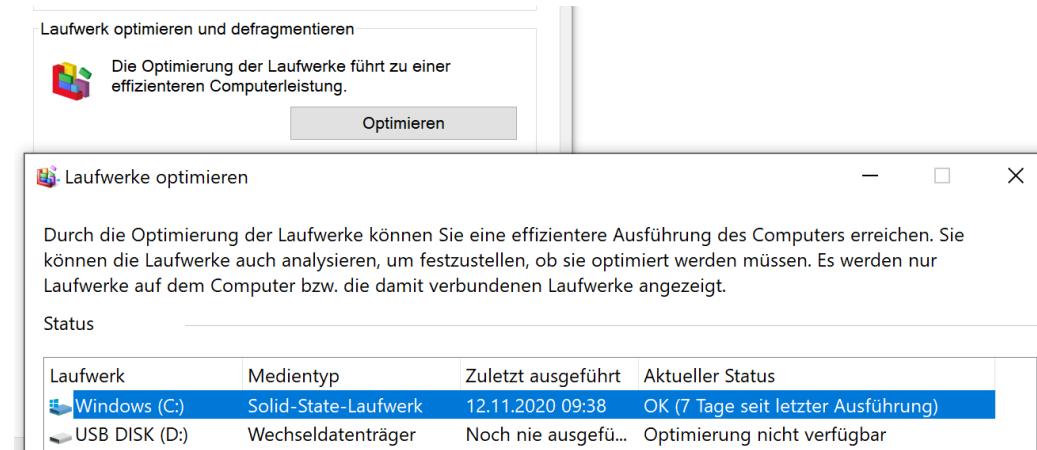
(b)

Quelle: Tanenbaum und Bos (2016)

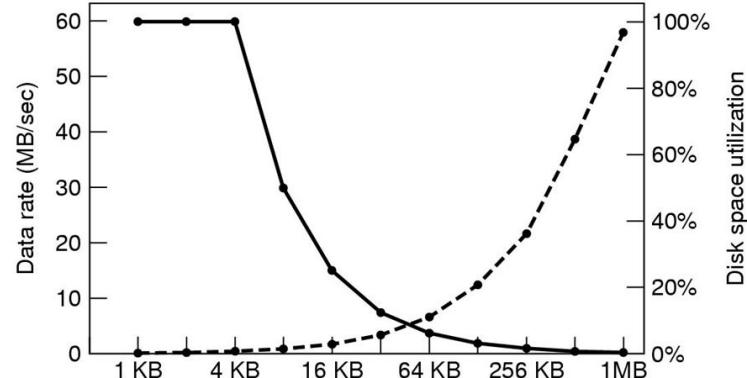
# Betriebssysteme

## Dateisysteme – Abbildung auf Datenträger

- Das Betriebssystem sollte dieser Fragmentierung bei Verwendung von konventionellen Festplatten entgegenwirken, um Leistungseinbußen durch ständige Neupositionierung des Kopfes zu vermeiden
- Windows nutzt dafür ein integriertes Tool, das automatisch z.B. einmal wöchentlich im Hintergrund eine mechanische Festplatte(n) defragmentiert
- Bei SSDs ist die Defragmentierung nicht erforderlich und sogar schädlich (unnötige Schreib/Lesebefehle)
- Die Optimierung unter Windows nutzt daher andere Optionen, etwa den TRIM-Befehl, der gelöschte Dateien wirklich entfernt



# Betriebssysteme



Quelle: Tanenbaum und Bos (2016)

## Dateisysteme – Anmerkungen zur Blockgröße

- Die Wahl der Blockgröße hat Einfluss auf die Datenübertragungs-rate und die Effizienz der Speichernutzung
- Kleine Blöcke ermöglichen hohe Speicherausnutzung, weil nur ein kleiner Verschnitt im letzten Block jeder Datei auftritt
- Große Blöcke reduzieren den Lese- und Schreiboverhead, weil je Datei weniger Blöcke benötigt werden
- Die Anforderungen stehen im Widerspruch, Kompromisse sind nötig
- Ist die Häufigkeitsverteilung der Dateigrößen bekannt, kann die Blockgröße optimiert werden, aber:
- Die physische Sektorgröße ist 4KB, kleinere Blöcke sind also nutzlos

# Betriebssysteme

## Dateisysteme – Löschen von Dateien

- Das Löschen von Dateien entfernt in der Regel nur den Eintrag für die Datei im Verzeichnis, etwa der MFT, und die Freigabe der betreffenden Blöcke für die erneute Nutzung
- Die Dateidaten selbst werden auf dem Datenträger zunächst nicht gelöscht
- Dateien können evtl. solange wiederhergestellt werden, bis sie ganz oder teilweise überschrieben wurden
- Wiederherstellungsprogramme können die Partition auf Datenträgerebene durchsuchen und so auch gelöschte Dateien lesen und z.B. auf einen anderen Datenträger sichern

# Betriebssysteme

## Dateisysteme – Gemeinsame Nutzung von Dateien

- Zur gemeinsamen Dateinutzung im gleichen Dateisystem gehören mehrere Aspekte, insbesondere
  - Speicherort und Anzahl der Instanzen
  - Zugriffsberechtigungen
  - Optionen für gleichzeitiges Arbeiten an derselben Datei
  - Kontrolle des von Benutzern verwendeten Speicherplatzes

# Betriebssysteme

## Dateisysteme – Gemeinsame Nutzung von Dateien Speicherort und Instanzen

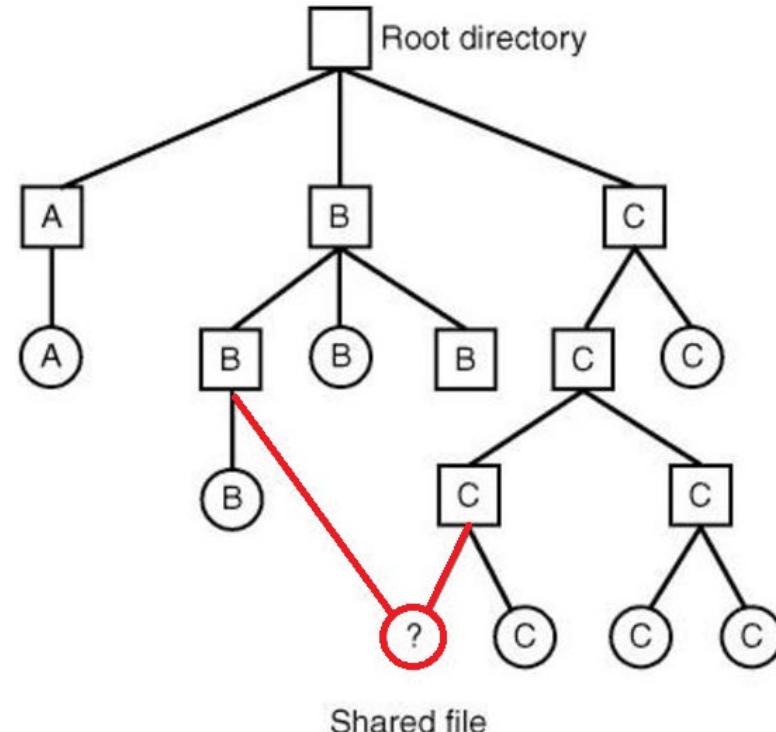
- Sofern nicht ausdrücklich eine Versionskontrolle erwünscht ist, sollen Dateien nach Möglichkeit nur einmal im Dateisystem gespeichert werden, **je Datei nur eine Instanz**
- Zur gemeinsamen Nutzung gibt es dann folgende Optionen:
  - **Verknüpfung** mit Hard- oder Softlink
  - **Gemeinsames lokales Verzeichnis**
  - **Gemeinsames Netzlaufwerk (Server)**

# Betriebssysteme

Dateisysteme – Gemeinsame Nutzung von Dateien

Speicherort und Instanzen

- Wenn mehrere Benutzer mit getrennten Verzeichnissen auf dieselbe Datei eines Benutzers zugreifen sollen, kann die Datei den anderen Benutzern verknüpft werden

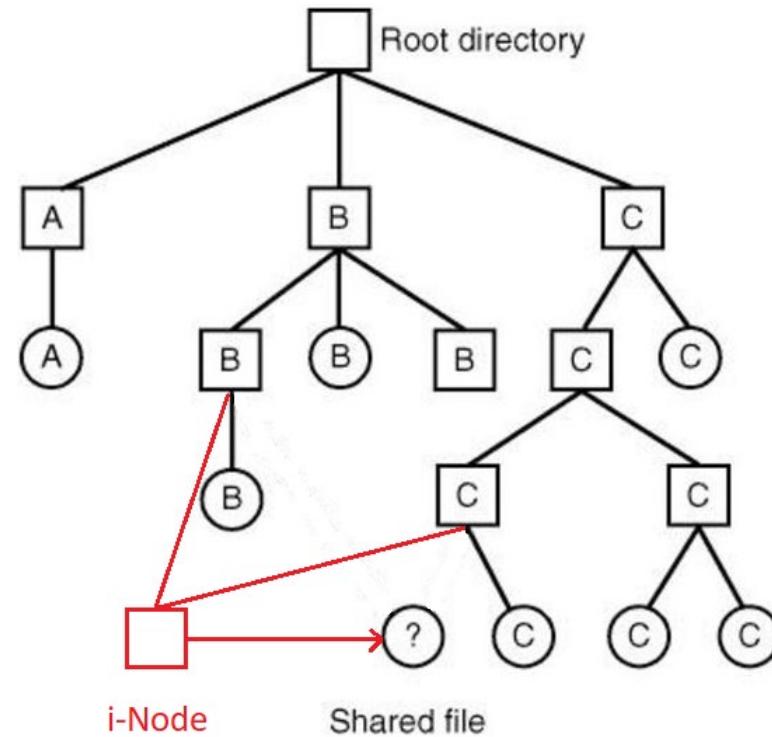


# Betriebssysteme

Dateisysteme – Gemeinsame Nutzung von Dateien

## Hard Link

- Eine Lösung ist ein **Hard Link**
- Dies ist **Verweis** auf Angaben zur Dateien im **Inhaltsverzeichnis des Dateisystems** (z.B. i-Node in Unix), nicht auf die Datei selbst
- Problem: löscht der Eigentümer C die Datei, entstehen **Inkonsistenzen**, weil im Inhaltsverzeichnis Einträge für nicht existierende Dateien stehen

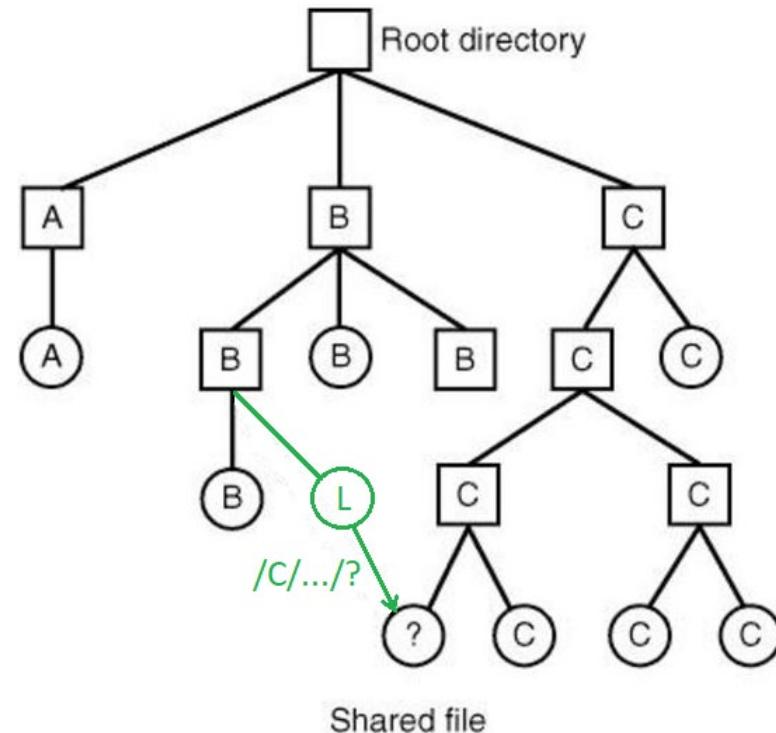


# Betriebssysteme

Dateisysteme – Gemeinsame Nutzung von Dateien

## Soft Link

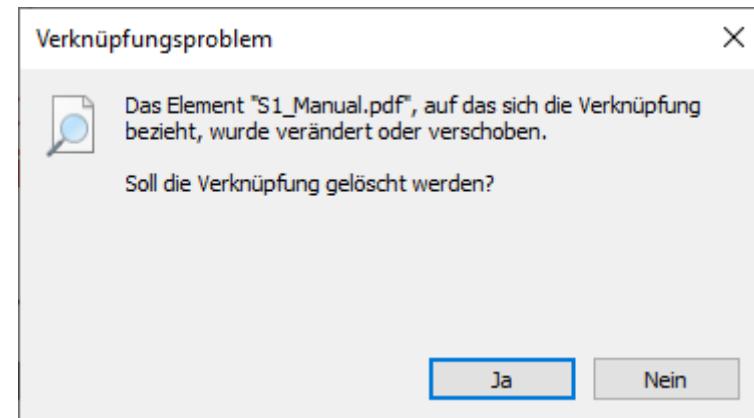
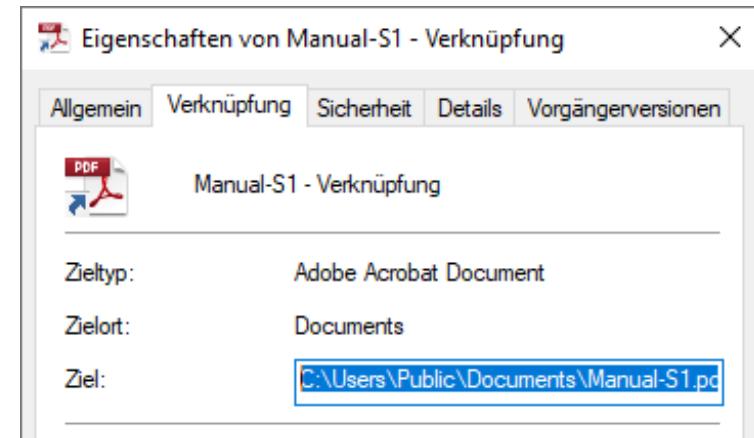
- Eine Lösung ist ein **Soft Link** oder **Symbolic Link**
- Es entsteht **eine neue Datei mit einer Pfadangabe zur gemeinsam genutzten Datei**
- Der Verwaltungsaufwand ist höher, aber beim Löschen einer Datei wird keine Inkonsistenz erzeugt, sondern lediglich eine Fehlermeldung



# Betriebssysteme

## Dateisysteme – Gemeinsame Nutzung von Dateien: Verknüpfung

- Das NTFS von Windows kennt Soft Links, die hier als **Verknüpfung** oder **Link** bezeichnet werden
- Verknüpfungen erhalten ein eigenes Symbol, der Dateityp ist **.LNK**
- Allerdings können Benutzer mit der Fehlermeldung nach dem Löschen der Zielfile oft nichts anfangen, weil sie Verknüpfungen nicht kennen...
- Hard Links sind seit NTFS v3 möglich



# Betriebssysteme

## Dateisysteme – Gemeinsame Nutzung von Dateien

### Zugriffsberechtigungen

- Auch wenn Benutzer eigene Dateien anderen zur Verfügung stellen, werden Zugriffsberechtigungen angewendet
- Es ist Aufgabe des Dateisystems, die Berechtigungen entsprechend anzupassen
- NTFS hat hier eine nachteilige Eigenschaft: Werden Dateien nur von einem Verzeichnis in ein anderes auf derselben Partition **verschoben**, bleiben die bestehenden Berechtigungen erhalten
- NTFS-Berechtigungen werden innerhalb derselben Partition nur neu gesetzt, wenn Dateien in einem anderen Verzeichnis **neu erstellt** werden

# Betriebssysteme

## Dateisysteme – Gemeinsame Nutzung von Dateien

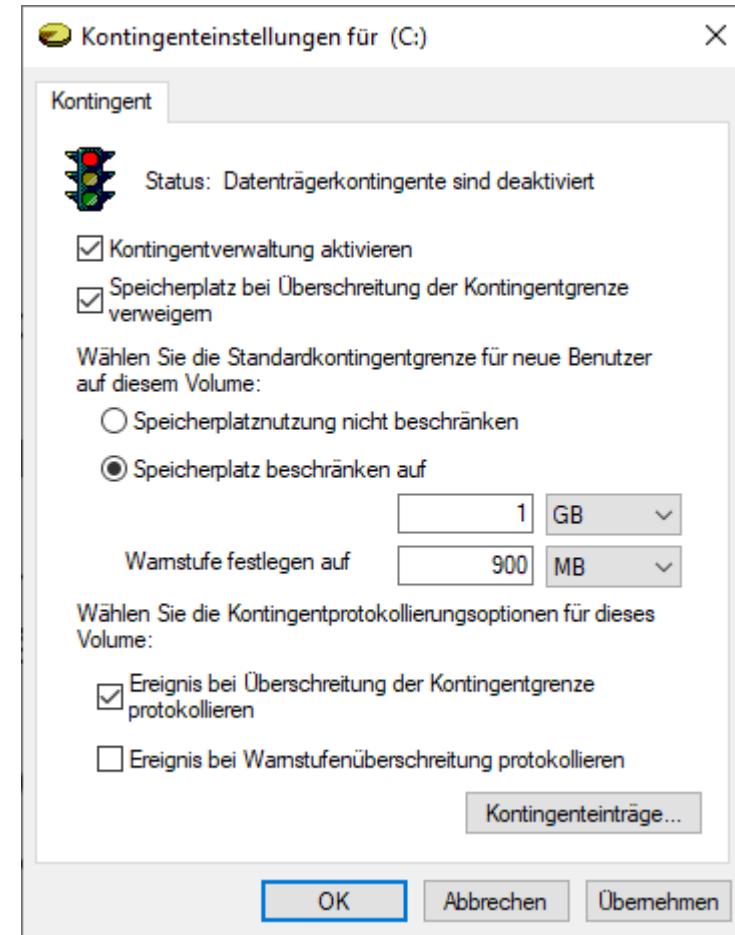
### Zugriffsberechtigungen

- Das kann bei der Übernahme von neuen Aufgaben kritisch sein:
- Alle Mitarbeiter einer Abteilung haben ein persönliches Home-Verzeichnis, auf das nur sie selbst zugreifen können
- Benutzer A gibt Aufgaben an den Nachfolger Benutzer B ab und verschiebt seine Dateien in ein gemeinsames Projektverzeichnis auf denselben Serverpartition
- Ergebnis: Obwohl Benutzer B die erforderlichen Zugriffsrechte für das Projektverzeichnis besitzt, wird der Dateizugriff verweigert
- **Lösung:** Die Dateien müssen kopiert und nicht verschoben werden

# Betriebssysteme

## Dateisysteme – Datenkontingente

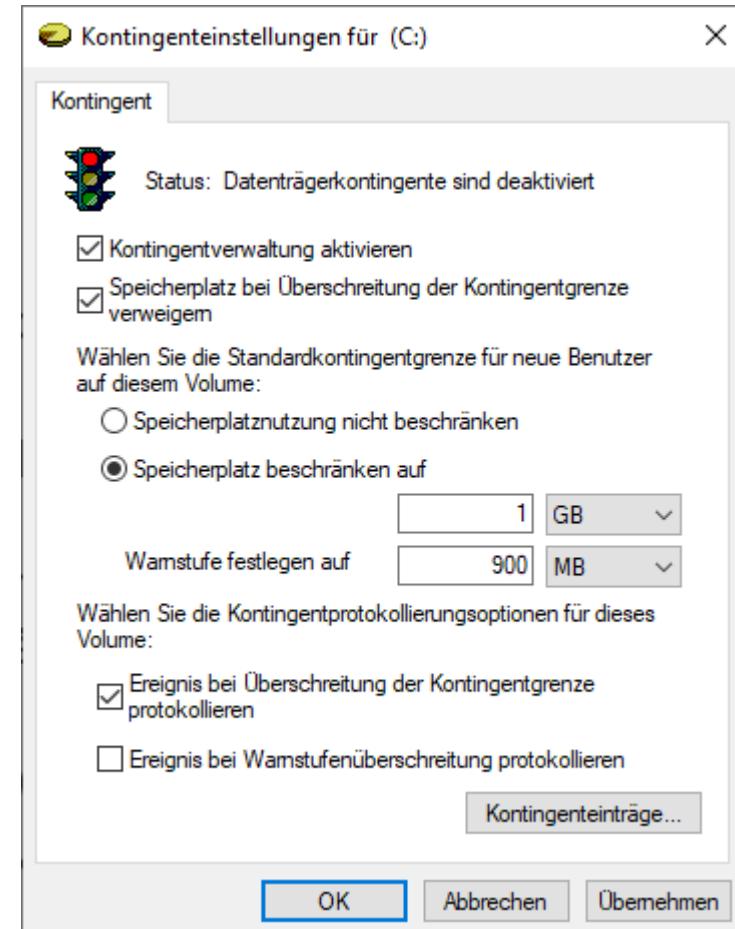
- Dateien haben die unangenehme Eigenschaft, ständig mehr zu werden und an Größe zuzunehmen
- In Unternehmensnetzwerken werden daher oft Funktionen benötigt, die den verfügbaren Dateispeicherplatz für Benutzer überwachen und beschränken
- Diese Funktion bieten moderne Dateisysteme für Unix und Windows
- Kontingente (Quotas) werden im NTFS auf Partitionsebene aktiviert



# Betriebssysteme

## Dateisysteme – Datenkontingente

- Die Kontingentverwaltung zählt in der Regel die von einzelnen Benutzern verwendeten Blöcke
- Es gibt je Benutzer ein **weiches Limit** und ein **hartes Limit**
- Das **weiche Limit** ist eine Warnstufe, es hat nur Informationscharakter
- Erst ab Erreichen des harten Limits ist kein weiteres Speichern mehr möglich
- Ein ähnliches System verwendet der Mailserver Microsoft Exchange



# Betriebssysteme

## Dateisysteme – Dateisystemkonsistenz und Journaling

- Das Schreiben von Dateien ist eine Änderung am Dateisystem und damit fehleranfällig
- Kann z.B. ein Kopiervorgang aufgrund eines Systemabsturzes nicht abgeschlossen werden, besteht die Gefahr, dass Dateispeicherort und zugehörige Angaben im Inhaltsverzeichnis nicht übereinstimmen
- Das Dateisystem wäre in diesem Fall **inkonsistent**,
- Die Konsistenz muss dann nach einem Neustart geprüft werden, was bei aktuellen Partitionsgrößen Stunden dauern und zu Datenverlust führen kann

# Betriebssysteme

## Dateisysteme – Dateisystemkonsistenz und Journaling

- Um zumindest Inkonsistenzen zu vermeiden und daher nach einem Systemabsturz schneller starten zu können, führen Dateisysteme mit Journaling ein Protokoll, in das alle geplanten Änderungen vorab eingetragen werden
- Erst wenn die Änderungen erfolgreich durchgeführt werden konnten, wird der entsprechende Eintrag im Journal als erledigt markiert
- Es gibt mehrere Arten der Protokollierung, u.a.:
  - **Vollständiges Journaling**
  - **Metadaten-Journaling**

# Betriebssysteme

## Dateisysteme – Dateisystemkonsistenz und Journaling

### Vollständiges Journaling

- Hierbei werden alle physischen Änderungen an **Blöcken, Daten und Metadaten** zuerst ins Protokoll aufgenommen und nachträglich im Dateisystem ausgeführt
- Damit sind sowohl Inhalte als auch Metadaten geschützt, das System verhält sich wie eine **logbasierte Transaktion** in einer Datenbank
- Nachteilig ist, dass **jeder Vorgang zweimal** ausgeführt werden muss, und dass **aus den Logs Dateiinhalte gelesen werden können** (Sicherheit, beim Löschen von Dateien werden diese Inhalte nicht gelöscht)

# Betriebssysteme

## Dateisysteme – Dateisystemkonsistenz und Journaling

### Metadaten-Journaling

- Hierbei werden nur **Metadaten** vorab ins Protokoll aufgenommen, bei einem Kopiervorgang also die Änderungen an den Verzeichnissen
- Zu kopierende Daten werden zuerst in den **Page Cache im RAM** eingelesen und erst später durch den Systemaufruf `sync` auf die Festplatte geschrieben (Write Back); Beispiel: NTFS
- Stürzt das System zwischen Journal-Änderung und Abschluss des Write-Back ab, kann es zu Datenverlusten kommen
- Allerdings können aus dem Journal dann keine Dateninhalte gelöschter Dateien mehr ausgelesen werden

# Betriebssysteme

## Dateisysteme – Dateisystemkonsistenz und Journaling

### Copy-on-Write

- Eine Alternative zum Journaling bezüglich Datensicherheit bieten aktuelle Dateisysteme mit Copy-on-Write
- Grundidee: Bei Änderungen werden nicht bestehende Dateien überschrieben, sondern zuerst vollständig neue Dateien in freien Blöcken angelegt
- Anschließend wird das Inhaltsverzeichnis so angepasst, dass es auf die neue Datei zeigt
- Die alte Datei bleibt mindestens solange erhalten, bis die Anpassung erledigt und verifiziert ist

# Betriebssysteme

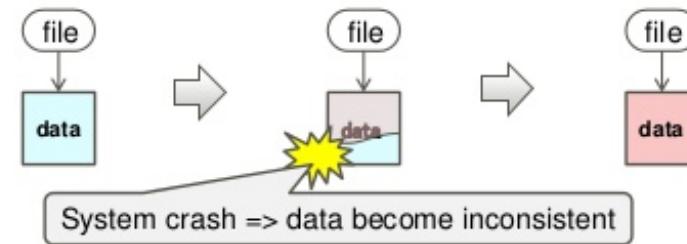
## Dateisysteme – Dateisystemkonsistenz und Journaling

### Copy-on-Write

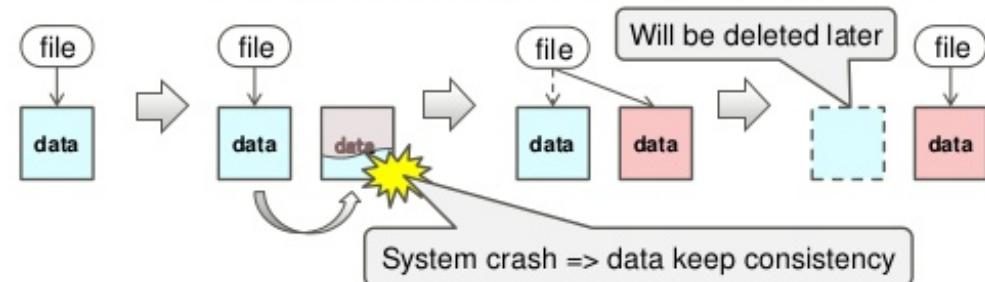
#### Dateisystembeispiele

- ZFS von SUN
- Resilient File System (ReFS) von Microsoft

### ■ Overwrite style: Update the data in place



### ■ CoW style: Copy, update, and replace pointer



# Betriebssysteme

## Dateisysteme – Optionen für die Datensicherung

- Zur Sicherung von Dateisystemen können verschiedene Methoden herangezogen werden
- Das **Image Backup** erstellt bei Verwendung mit Festplatten oder Partitionen ein **exaktes Abbild der physischen Sektoren**, ggf. einschließlich fehlerhafter Sektoren und kommt ohne Support durch das Dateisystem aus
- Eine **logische Datensicherung** oder **File-Level-Backup** arbeitet mit dem Dateisystem zusammen
- Einige Dateisysteme bieten **Snapshots**, mit denen sich der jeweils aktuelle Zustand eines Dateisystems festhalten und später wiederherstellen lässt

# Betriebssysteme

## Dateisysteme – Optionen für die Datensicherung

- Das **Image Backup** kann auf physische oder logische Datenträger angewendet werden, ohne dass dafür eine Unterstützung des Dateisystems erforderlich wäre
- Das Image Backup sichert jeden Sektor eines Datenträgers und schreibt alle Sektoren in eine Image-Datei
- Die Image-Datei kann meist gemounted und dann wie ein normaler Datenträger nach Dateien durchsucht werden
- Alternativ können Datenträger komplett wiederhergestellt werden
- **Hersteller von Sicherungssoftware definieren den Begriff Image Backup möglicherweise anders!**

# Betriebssysteme

## Dateisysteme – Optionen für die Datensicherung

- Die **logische Datensicherung** erlaubt die gezielte Auswahl von Dateien oder Verzeichnissen schon vor der Sicherung und gestattet deren Automatisierung
- Dateisysteme führen dazu Buch über Änderungen an Dateien und markieren geänderte Dateien z.B. mit einem **Archiv-Bit** als **zu sichern**
- Ein Sicherungsprogramm kann dann angewiesen werden, nur die Dateien zu sichern, deren Archiv-Bit gesetzt ist
- Nach erfolgreicher Sicherung werden die Archiv-Bits vom Sicherungsprogramm wieder zurückgesetzt
- Die Logische Datensicherung ist das klassische Backup-Verfahren

# Betriebssysteme

## Dateisysteme – Optionen für die Datensicherung

- Ein **Snapshot** wird vom Betriebssystem als Momentaufnahme des Dateisystems zu einem bestimmten Zeitpunkt erstellt
- Der Snapshot enthält eine **Übersicht über die Blockverteilung und die Metadaten des Dateisystems**, jedoch **keine vollständigen Daten**
- Jeder Snapshot umfasst nur die Differenz zu einem Vorgänger
- Snapshots lassen sich sehr schnell erstellen und benötigen nur wenig Speicherplatz
- Snapshots ermöglichen die Rückkehr zu einem früheren Datenstand, **ersetzen aber nicht die Sicherung**
- Auch hier gilt, dass **Hersteller den Begriff anders definieren können**

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - Speicherverwaltung
  - Dateisysteme
  - **Ein- und Ausgabe**
  - Multiprozessorsysteme
  - Virtualisierung

Praxis: Umgang mit Linux (Prof. Brunner) und Windows Server 2019

# Betriebssysteme

## Ein- und Ausgabe – Übersicht

- Gerätekategorien
- Gerätekommunikation
- Software
  - Schichten
  - Gerätetreiber
  - Treiberschnittstellen
- Einige Geräte und Gerätetreiber
  - Zeitgeber
  - Tastatur
  - Maus
  - Bildschirm

# Betriebssysteme

## Ein- und Ausgabe – Gerätekategorien

- Den Kontakt zur Außenwelt stellt ein Computer über Ein- und Ausgabegeräte her
- Grundsätzlich ist es wünschenswert, im Betriebssystem alle I/O-Geräte gleich zu behandeln
- Je nach Gerätetyp und Art der Ansteuerung bzw. Datenverarbeitung bestehen jedoch Unterschiede zwischen den Geräten:
- **Blockorientierte Geräte** wie Festplatte, USB-Stick oder Band verarbeiten nur Datenblöcke fester Länge
- **Zeichenorientierten Geräten** wie Tastatur, Netzwerkkarte, Maus, Kamera, Mikrofon usw. verarbeiten Folgen einzelner Bytes

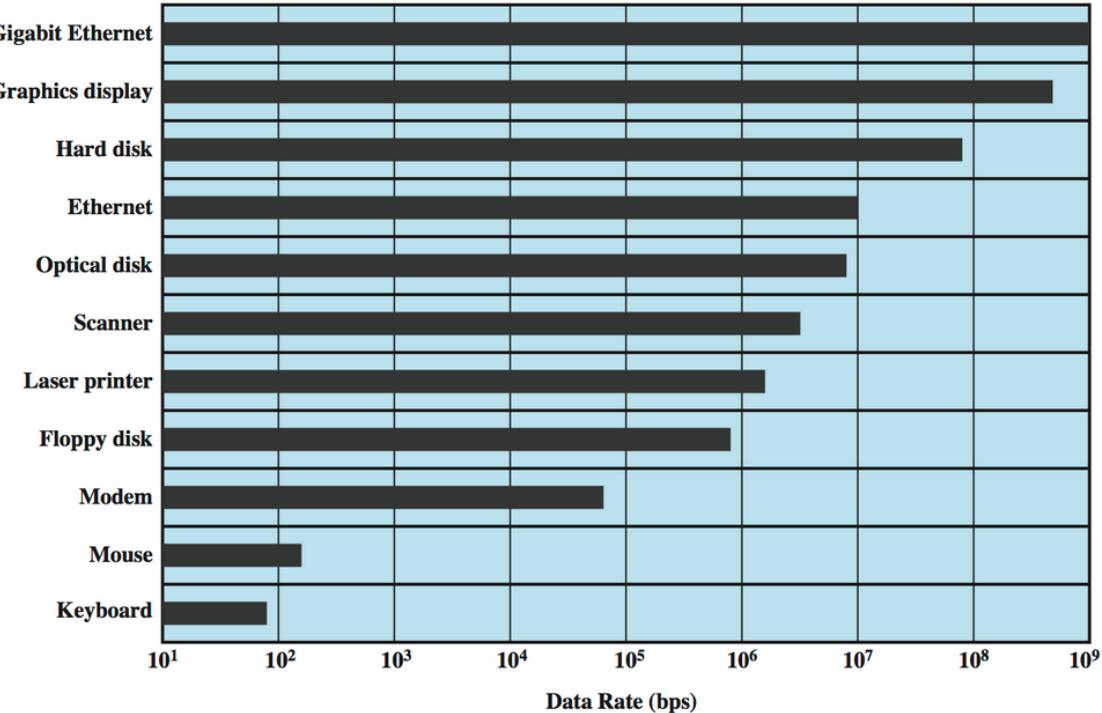
# Betriebssysteme

## Ein- und Ausgabe – Gerätekategorien

- Nicht alle Ein- und Ausgabegeräte lassen sich jedoch problemlos in diese Kategorien eingruppieren
- **Zeitgeber** (Timer) und **Bildschirme** liefern und verarbeiten andere Signalarten
- Diese Geräte werden separat betrachtet
- Verschiedene Geräte arbeiten außerdem mit stark unterschiedlichen Datenraten

# Betriebssysteme

## Ein- und Ausgabe – Gerätekategorien



Quelle: teaching.csse.uwa.edu.au

- Die meisten dieser Datenraten liegen erheblich unter den internen Datenraten der CPU und würden diese ohne weitere Maßnahmen mit viel Wartezeit auf Daten ausbremsen
- Deshalb gibt es den jeweiligen Datenraten angepasste Wege zur Kommunikation mit der CPU und dem Hauptspeicher

# Betriebssysteme

## Ein- und Ausgabe – Gerätekommunikation

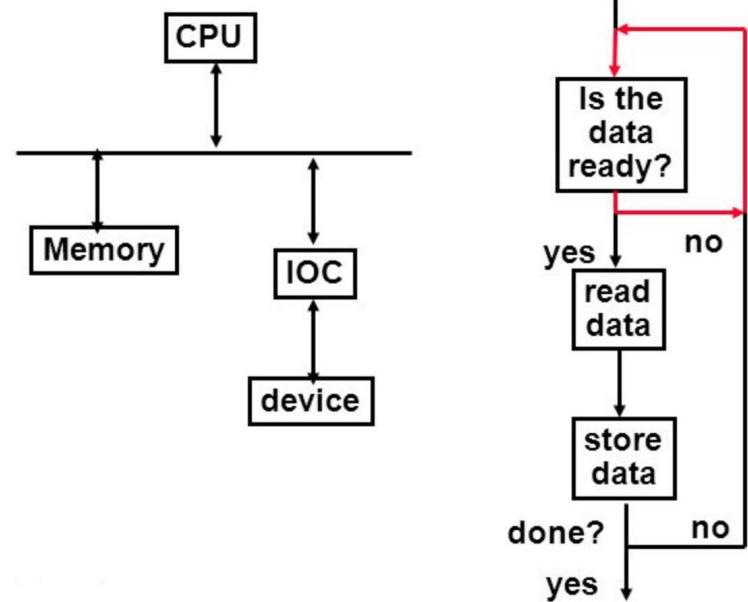
- Je nach Gerätetyp gibt es passende Verfahren, mit denen Geräte kommunizieren und ihre Daten übergeben oder abholen
- **Programmierte Ein/Ausgabe, Polling:** Regelmäßiges Abfragen der Geräte untereinander, schrittweises Abarbeiten der Ein/Ausgabeoperation durch Betriebssystem und CPU
- **Interrupt:** Geräte lösen einen Interrupt aus, die CPU unterbricht laufende Arbeit und prüft die Geräteanforderung
- **Direct Memory Access (DMA):** Die übertragene Datenmenge zwischen Interrupts wird auf Blöcke oder Pufferinhalte vergrößert, was die Anzahl der Interrupts verringert

# Betriebssysteme

## Ein- und Ausgabe

### Programmierte Ein/Ausgabe, Polling

- Synchrone, schrittweises Abarbeiten der Ein/Ausgabeoperation durch Betriebssystem und CPU
- CPU fragt Gerät zyklisch ab, ob Daten zu liefern sind
- Nachteil: Viel CPU-Zeit wird mit Warteschleife verbracht
- OK in Systemen, die nichts anderes zu tun haben (eingebettete Systeme)

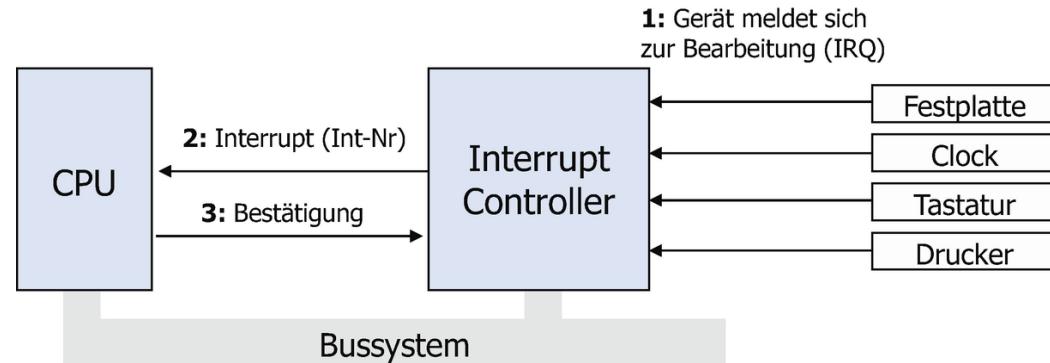


# Betriebssysteme

## Ein- und Ausgabe

### Interrupt

- Beispielsweise durch Fehler während der Arbeit der CPU oder **Ein/Ausgabeanforderungen von Geräten** wie Timer usw. können **Interrupts** (Unterbrechungen) ausgelöst werden
- Interrupts sind **Hardwareereignisse**
- Für Interrupts besitzen CPUs eine spezielle Signalleitung oder das System enthält einen separaten **Interrupt-Controller**
- Interrupts sollen Unterbrechungen der aktuellen Prozesse einer CPU auslösen

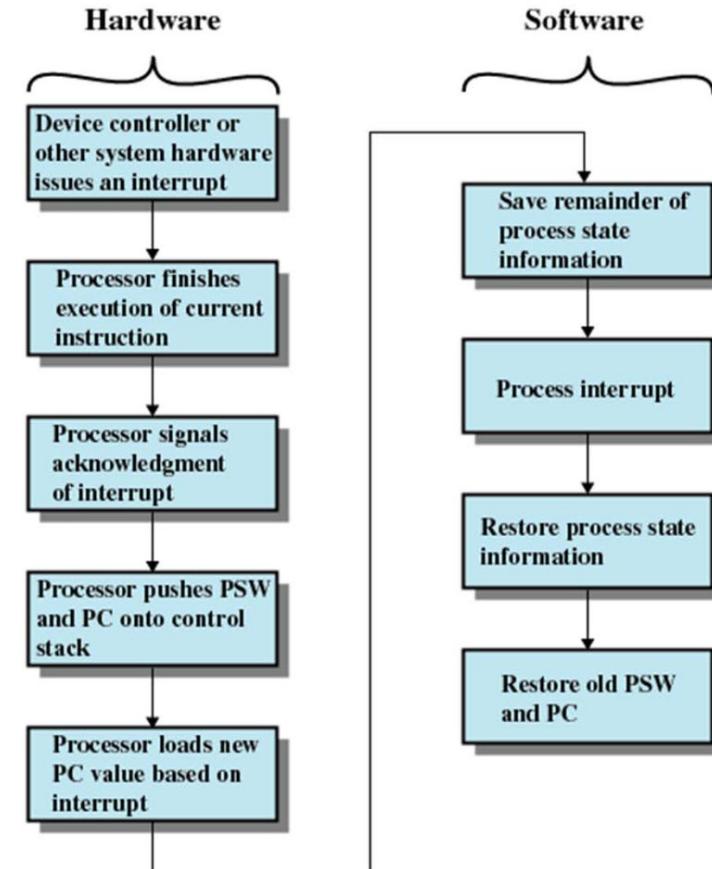


Quelle: link.springer.com

# Betriebssysteme

## Ein- und Ausgabe Interrupt

- Bei Auftreten eines Gerät-Interrupts beendet die CPU die Arbeit am aktuellen Prozess und startet den Ein/Ausgabeprozess für das Gerät
- CPU und OS führen den Kontextwechsel in den neuen Prozess durch, arbeiten die Geräteanforderung ab und wechseln wieder zurück in den alten Kontext



# Betriebssysteme

## Ein- und Ausgabe

### Interrupt

- Interrupts besitzen eine Identifikationsnummer, den **Interrupt-Vektor**
- Für Geräte nutzbar sind Interrupts **32-255**
- Was genau die CPU zu tun hat, wenn ein bestimmter Interrupt auftritt, wird mit einer vom Interrupt-Vektor abhängigen **Interrupt-Routine** definiert

Vector No.	Mnemonic	Description
0	#DE	Divide Error
1	#DB	Debug
2		NMI Interrupt
3	#BP	Breakpoint
4	#OF	Overflow
5	#BR	BOUND Range Exceeded
6	#UD	Invalid Opcode (UnDefined Opcode)
7	#NM	Device Not Available (No Math Coprocessor)
8	#DF	Double Fault
9	#MF	CoProcessor Segment Overrun (reserved)
10	#TS	Invalid TSS
11	#NP	Segment Not Present
12	#SS	Stack Segment Fault
13	#GP	General Protection
14	#PF	Page Fault
15		Reserved
16	#MF	Floating-Point Error (Math Fault)
17	#AC	Alignment Check
18	#MC	Machine Check
19	#XM	SIMD Floating-Point Exception
20-31		Reserved
32-255		Maskable Interrupts

# Betriebssysteme

## Ein- und Ausgabe

### Interrupt

- Interrupt-gesteuerte Ein/Ausgabe entlastet die CPU von regelmäßigen Abfragen aller Geräte
- Sie ist nur noch während der Datenübertragung am Vorgang beteiligt
- Interrupts werden **priorisiert** und können so abgelehnt oder angenommen werden, auch wenn gerade ein anderer Interrupt bearbeitet wird
- Die Effizienz ist höher als bei programmierte Ein/Ausgabe, auch wenn Kontextwechsel durchgeführt werden müssen, **solange nicht zu viele Interrupts auftreten**

# Betriebssysteme

## Ein- und Ausgabe

### Direct Memory Access (DMA)

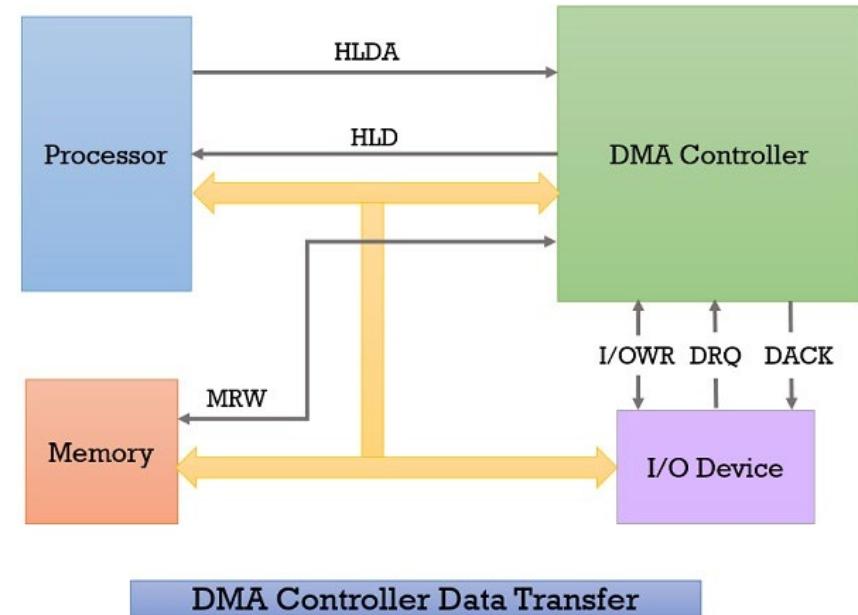
- Die Effizienz kann erhöht werden, wenn die Anzahl der Interrupts so gering wie möglich ist und die Datenübertragung von ihr entkoppelt wird
- Beides ermöglicht DMA
- Bei DMA übernimmt eine separate Hardware die Kommunikation mit dem Gerät und die Steuerung der Datenübertragung zum RAM
- Diese Hardware ist der **DMA-Controller**

# Betriebssysteme

## Ein- und Ausgabe

### Direct Memory Access (DMA)

- Der DMA-Controller erhält vom Gerät einen DMA-Request (DRQ)
- Der DMA-Controller einigt sich mit der CPU über eine Freigabe des Bussystems und steuert dann die Datenübergabe (HLD, HLDA)
- Abschließend gibt er die Kontrolle über das Bussystem an die CPU zurück



Quelle: [binaryterms.com/direct-memory-access-dma.html](http://binaryterms.com/direct-memory-access-dma.html)

# Betriebssysteme

## Ein- und Ausgabe

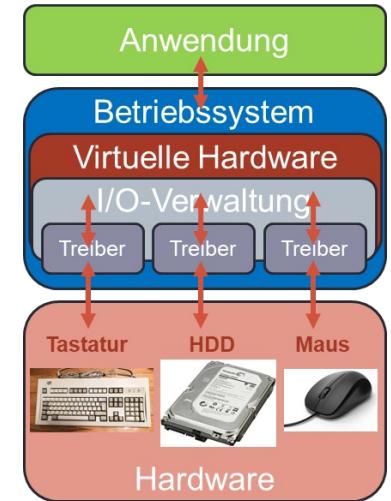
### Direct Memory Access (DMA)

- CPU und DMA-Controller müssen sich den Daten- und Adressbus teilen, damit das Gerät Daten in den RAM übertragen kann
- Für das Teilen gibt es drei Modi:
- **Burst-Modus**: Der DMA-Controller hat den Bus solange bis ein Datenblock übertragen ist (maximale Datentaten)
- **Cycle-Stealing-Modus**: Der DMA-Controller nimmt der CPU den Bus für einzelne Taktzyklen ab, bis jeweils ein Byte übertragen ist
- **Transparenter Modus**: Der DMA-Controller erhält den Bus nur, wenn die CPU ihn nicht benötigt (optimale Performanz des Gesamtsystems)

# Betriebssysteme

## Ein- und Ausgabe – Software

- Trotz der Verschiedenheit der Geräte ist es wünschenswert, den Applikationen für alle Geräteoperationen möglichst die gleichen einfachen Schnittstellen zu bieten => Virtuelle Hardware
- Das Stichwort ist **Geräteunabhängigkeit** oder **Device Independence**
- Um diese Aufgabe zu lösen, wird Software zur Steuerung der Geräte in mehrere **funktionale Ebenen** oder **Schichten** unterteilt
- Ein Teil der Operationen ist für alle Geräte ähnlich und kann in einer **systemnahen Softwarekomponente** erledigt werden
- Ein anderer Teil der Operationen ist gerätespezifisch und fordert für verschiedene Gerätetypen **hardwarenahe Softwarekomponenten**

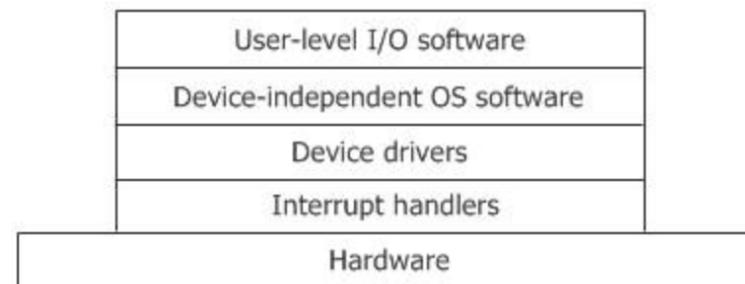


# Betriebssysteme

## Ein- und Ausgabe – Softwareschichten

### Physische Schichten

- Anwendungsprogramme arbeiten mit den APIs des Betriebssystems für Geräteaufrufe zusammen
- Die geräteunabhängige, systemnahe Software ist **Teil des Betriebssystems**
- Die gerätespezifische, hardwarenahe Komponente ist ein **Gerätetreiber**
- Gerätetreiber verarbeiten die Interrupt-Anforderungen der Hardware



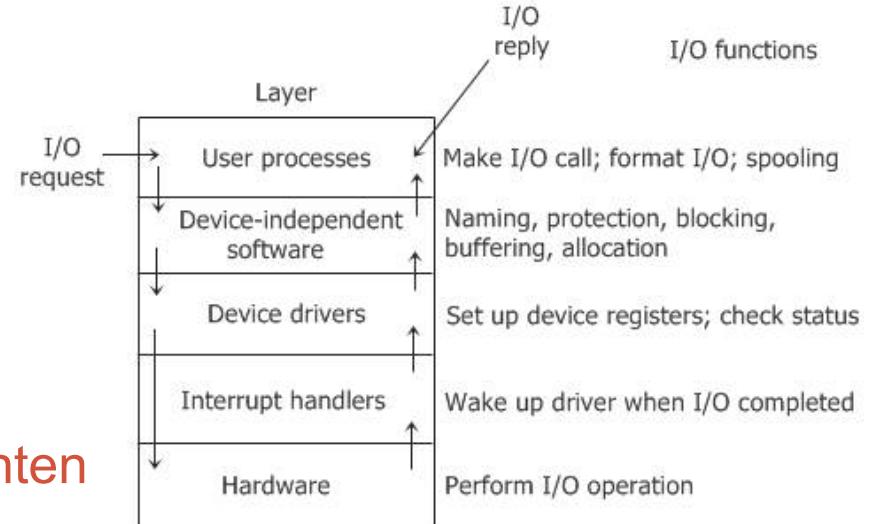
Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Ein- und Ausgabe – Softwareschichten

### Arbeitsweise 1

- Bei einer I/O-Anforderung kommuniziert die Applikation mit dem OS
- Das OS fordert den Gerätetreiber zur entsprechenden Funktion auf
- Der Gerätetreiber übersetzt die Anforderung in Hardwarebefehle und blockiert bis zu deren Erledigung
- Die Hardware führt die Operation aus und erzeugt nach deren Abschluss einen Interrupt



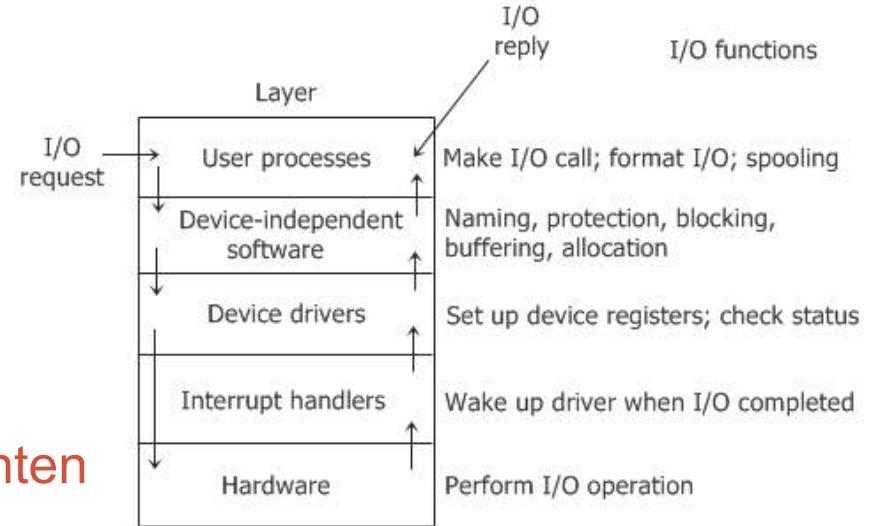
Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Ein- und Ausgabe – Softwareschichten

### Arbeitsweise 2

- Der Gerätetreiber wird nun fortgesetzt und bereitet die Datenübergabe an das OS vor
- Die geräteunabhängige Softwareschicht erledigt die Datenübergabe vom Gerät an die Applikation und Berücksichtigt dabei Gerätenamen, Pufferung, Schutz (durch Zugriffsrechte) usw.



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Ein- und Ausgabe – Softwareschichten

### Arbeitsweise 3

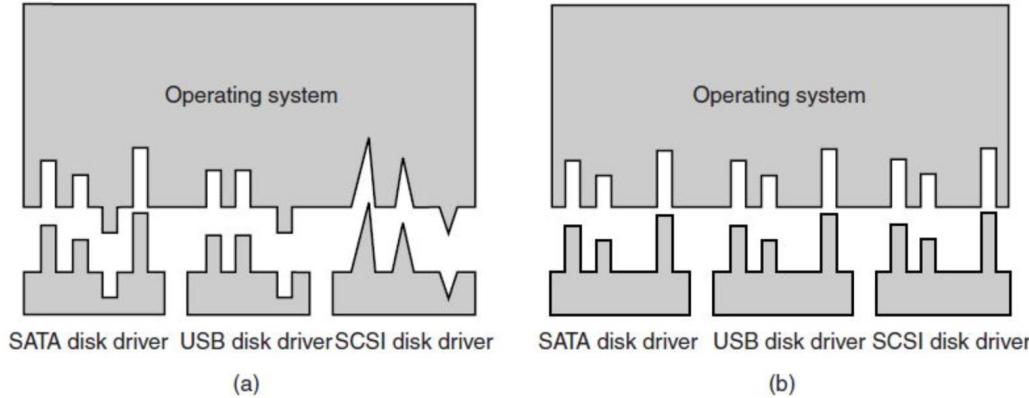
- Die geräteunabhängige Softwareschicht des Betriebssystems ist für weitere Aufgaben zuständig
- **Pufferung** eingehender und ausgehender Daten
- **Schutz** durch Zugriffskontrolle
- Bei manchen Geräten, die nur von einem Prozess gleichzeitig verwendet werden können wie z.B. Drucker, **Reservierung und Freigabe** des Gerätes für andere Prozesse

# Betriebssysteme

## Ein- und Ausgabe – Software

### Einheitliche Schnittstelle

- Die **Schnittstelle** zwischen Gerätetreibern und geräteunabhängiger Softwareschicht des Betriebssystems sollte **einheitlich** sein, damit der Programmieraufwand möglichst gering bleibt
- Zwar können nicht für alle Gerätetypen identische Treiber eingesetzt werden, aber es können wenige **Geräteklassen** definiert werden
- Für jede Klasse definiert das Betriebssystem einen Satz von Funktionen als Standard, den ein Treiber unterstützen muss
- Alle Gerätehersteller müssen sich an diesen Standard halten



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

Ein- und Ausgabe

Einige Geräte und Gerätetreiber

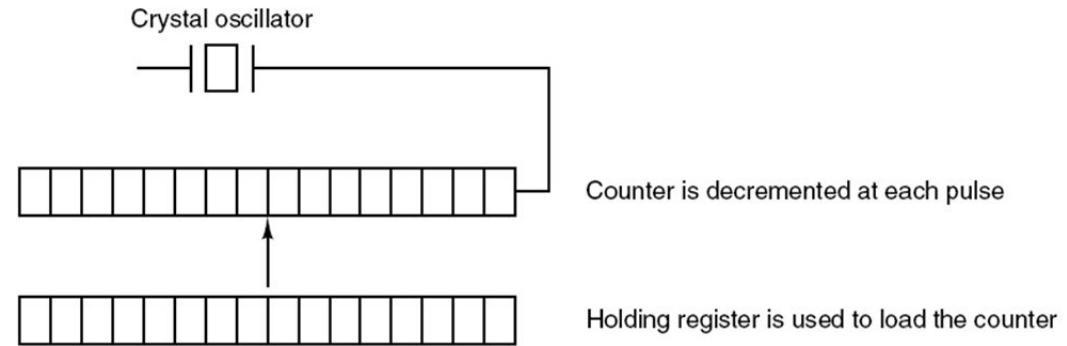
- Zeitgeber
- Tastatur
- Maus
- Bildschirm

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Zeitgeber

- Zeitgeber bzw. Timer steuern als **programmierbare Uhren** viele Abläufe im Computer
- Dazu gehört u.a. die Generierung von **Timer-Interrupts**, die auch für die Bemessung und Steuerung der CPU-Zeit für Prozesse und deren Kontextwechsel benutzt werden
- Programmierbare Uhren leiten ihre Taktsignale von einem Quarzoszillator ab, dessen Signal einen Zähler dekrementiert
- Der Startwert des Zählers – und damit das Timerintervall – wird durch ein Softwaregesteuertes Halteregister festgelegt



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Zeitgeber

- Die von der Hardware des Zeitgebers gelieferten Timer-Interrupts setzt der **Uhrentreiber** als Software-Uhr zur Erledigung seiner Aufgaben ein, z.B....
- Uhrzeitverwaltung und **Synchronisierung mit der aktuellen Zeit**
- Steuerung der Prozesslaufzeiten über den Scheduler
- Bereitstellung von Uhren für **verschiedene Zwecke im System**, z.B. **Watchdog-Timer** als „Totmannschalter“ für das Betriebssystem:
- Im lauffähigen Zustand setzt das OS den Timer selbst zurück, falls es stehengeblieben ist, erkennt dies der Timer und startet es z.B. neu

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Tastatur

- Die Tastatur wandelt über Schalter mechanische Bewegungen in elektrische Zustände um: offener oder geschlossener Kontakt
- Sowohl das Schließen des Kontaktes beim Drücken einer Taste als auch das Öffnen beim Loslassen entspricht einer Information
- Der Ruhezustand sind geöffnete Kontakte bei nicht gedrückten Tasten
- Der Tastaturtreiber muss davon abweichende Zustände an das System übergeben
- Welche Tasten werden in welcher Reihenfolge gedrückt?
- Welche Tasten sind gleichzeitig noch gedrückt?

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Tastatur

- Der Tastaturtreiber erzeugt bei jedem Drücken einer Taste einen Interrupt
- Dann muss er aus den gedrückten Tasten einen Wert ermitteln und diesen an das System übergeben
- Meist übergibt der Treiber jedes Zeichen einzeln an das System (zeichenorientierter Raw-Modus)
- Alternativ kann im kanonischen Modus die Übergabe so lange verzögert werden, bis eine Zeile zusammengekommen ist: erst dann wird die Übergabe vorgenommen

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Maus

- Mäuse steuern einen Cursor über die Anzeigefläche einer grafischen Benutzeroberfläche (GUI)
- Dazu besitzen sie mechanische oder optische Bewegungssensoren, Tasten und ein Scrollrad
- Bewegungssensoren arbeiten in Inkrementen von z.B. 0,1 mm und übertragen die Differenz zum letzten Status
- Die vom Bewegungssensor aufgenommene Bewegungen der Maus, des Scrollrades und der Tastenzustand werden in wenigen Bytes z.B. 40 mal pro Sekunde an das System übertragen

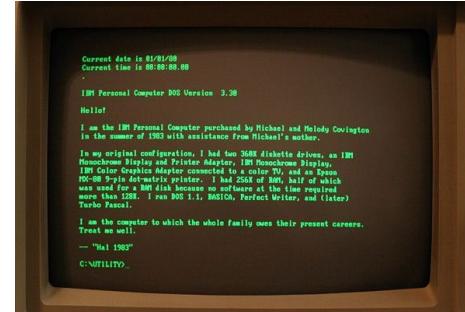
# Betriebssysteme

## Einige Geräte und Gerätetreiber

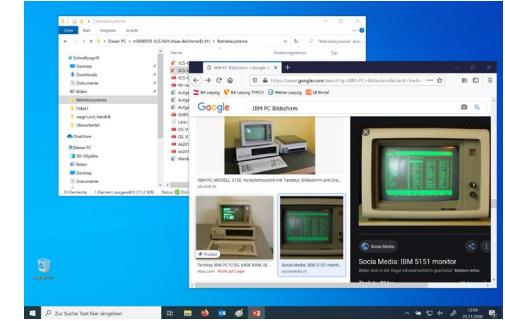
### Maus

- Der Maustreiber wertet die Informationen aus
- Er entscheidet auch, ob zwei nacheinander ausgeführte Klicks räumlich und zeitlich so nahe beieinander lagen, dass diese als Doppelklick zu interpretieren sind, und löst die gewünschte Aktion im System aus

# Betriebssysteme



Quelle: www.pinterest.de



## Einige Geräte und Gerätetreiber Bildschirm

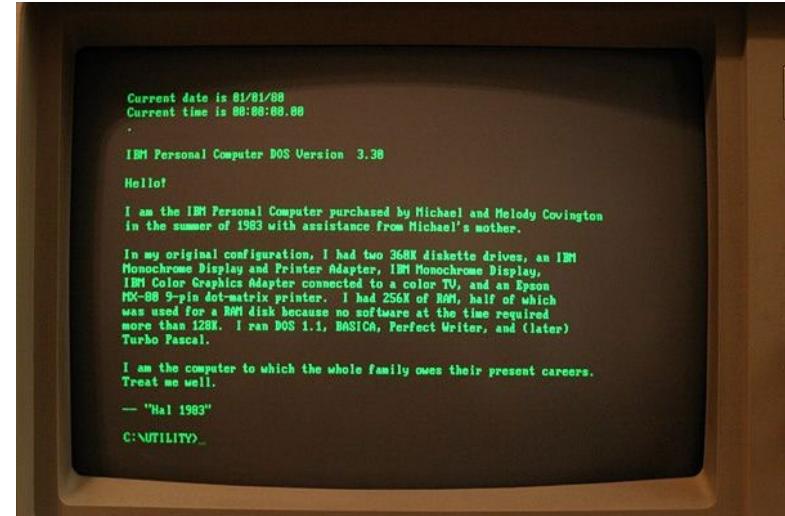
- Wichtigstes Ausgabemedium für Computer ist der Bildschirm
- Die benötigten Ausgabefunktionen unterscheiden sich grundlegend danach, ob das Betriebssystem eine **Shell** oder eine **grafische Benutzeroberfläche** besitzt
- Die Anzeige einer Shell umfasst die Darstellung und Löschung von Zeichen und die Positionierung eines Cursors und benötigt wenig Ressourcen
- Die Darstellung einer grafischen Benutzeroberfläche ist dagegen sehr ressourcenintensiv und es wird eine umfangreiche Grafik-Hardware benötigt

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Bildschirm

- Die Ausgabesoftware für die Anzeige einer Shell hat die Aufgabe, vom Rechner gesendete Zeichen anzuzeigen und Kommandos zu interpretieren, die die Anzeige der Zeichen betreffen: welches Zeichen, welche Position, Hinzufügen, Löschen usw.
- Der Bildschirm bzw. das **Terminal** akzeptiert die Zeichen direkt und zeigt sie auf einer Fläche von z.B. 25x80 Zeichen an
- Kommandos werden von normalen Zeichen durch Voranstellen des **ESC-Zeichens 0x1B** getrennt
- Nach dem ESC folgen die Argumente des Kommandos



Quelle: [www.pinterest.de](http://www.pinterest.de)

# Betriebssysteme

## Einige Geräte und Gerätetreiber Bildschirm

- Erkennt der Terminaltreiber ein ESC, unterbricht er die laufende Verarbeitung, analysiert das folgende Kommando und führt es aus
- Diese **Escape-Sequenzen** sind seit 1978 ANSI-Standard und wurden so erstmals im Terminal **DEC-V100** realisiert (1979-1983)



Quelle: computermuseum.informatik.uni-stuttgart.de

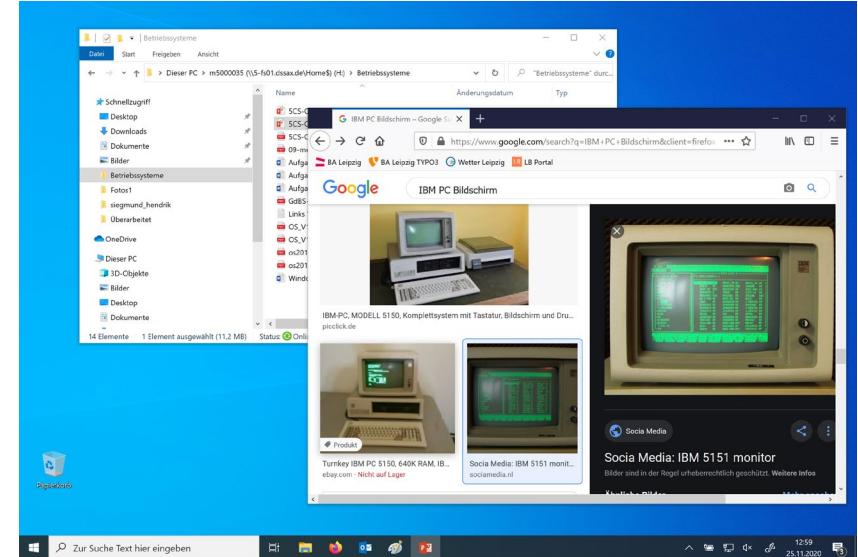
Escape sequence	Meaning
ESC [nA	Move up <i>n</i> lines
ESC [nB	Move down <i>n</i> lines
ESC [nC	Move right <i>n</i> spaces
ESC [nD	Move left <i>n</i> spaces
ESC [ <i>m</i> ; <i>n</i> H	Move cursor to ( <i>m</i> , <i>n</i> )
ESC [ <i>s</i> J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [ <i>s</i> K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [ <i>n</i> L	Insert <i>n</i> lines at cursor
ESC [ <i>n</i> M	Delete <i>n</i> lines at cursor
ESC [ <i>n</i> P	Delete <i>n</i> chars at cursor
ESC [ <i>n</i> @	Insert <i>n</i> chars at cursor
ESC [ <i>n</i> m	Enable rendition <i>n</i> (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Einige Geräte und Gerätetreiber Bildschirm

- Monochrome Terminalbildschirme müssen nur eine Matrix aus rund 1000 Zeichen auf einer Seite darstellen können
- Davon weicht die Darstellung der Elemente einer grafischen Benutzeroberfläche ab: Zeichen können nicht mehr direkt als ASCII-Code ausgewertet und angezeigt werden, sondern müssen als Bilder berechnet und ausgegeben werden
- Tatsächlich ist der gesamte Bildschirminhalt ein Bitmap-Bild
- Grafische Benutzeroberflächen benötigen deshalb schon in der kleinsten Auflösung des IBM-PCs 640x480 Pixel, also 307200 Bit für eine monochrome Darstellung



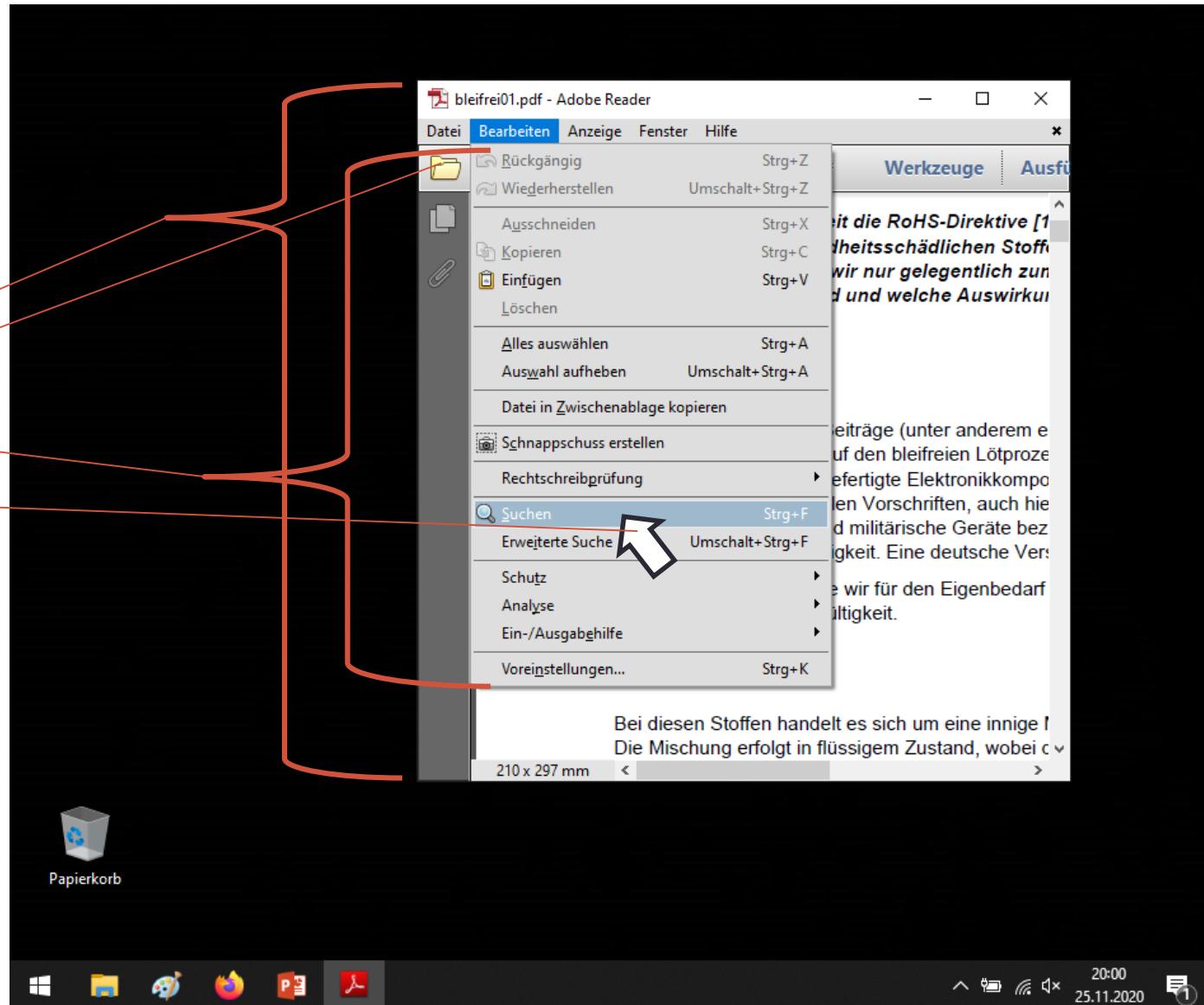
# Betriebssysteme

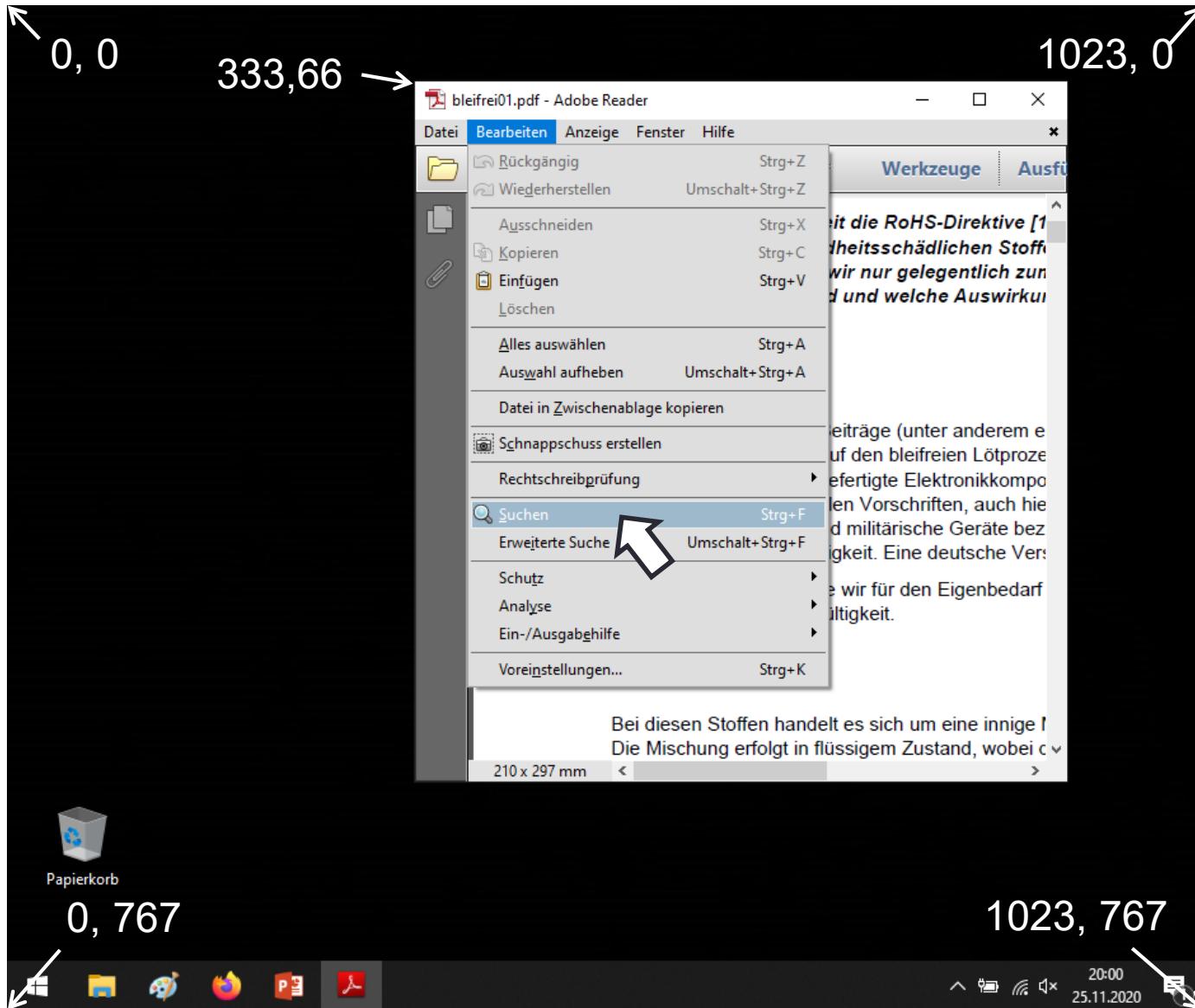
## Einige Geräte und Gerätetreiber

### Bildschirm

- Grafische Benutzeroberflächen wurden Mitte der 1970er Jahre entwickelt und setzten sich etwa 10 Jahre später durch, kommerziell zuerst im Apple Macintosh und dann mit Microsoft Windows
- Die mittlerweile klassische grafische Benutzeroberfläche am PC ohne Touchscreen umfasst vier Grundelemente:
- **Window**
- **Icon**
- **Menu**
- **Pointer**

- Window
  - Icon
  - Menu
  - Pointer
- => WIMP





# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Bildschirm

- Jede Windows-GUI-Applikation öffnet beim Starten ein oder mehrere Fenster
- Das Design der Funktionselemente des Fensters wird vom Betriebssystem bestimmt
- Programmierer legen jedoch fest, welche Funktionselemente und Menüs es geben soll
- Jedes Fenster umfasst den Arbeitsbereich, in dem das Programm Inhalte darstellt und verändert, beispielsweise Text oder Grafiken

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Bildschirm

- Windows-Applikationen greifen nicht direkt auf die Grafik-Hardware zu, sondern statt dessen auf ein spezielles API, das **Graphic Device Interface GDI**
- Das GDI stellt alle Funktionen zur Darstellung von Farben, Grafik, Text und Bitmaps bereit
- Es kann zur Bildschirm- und Druckerausgabe verwendet werden

# Betriebssysteme



## Einige Geräte und Gerätetreiber

### Bildschirm

- Schriften wurden ursprünglich als reine Bitmaps, also Abbilder der Buchstaben dargestellt
- Für jede Buchstabengröße und jeden Buchstaben wurden mangels Skalierbarkeit eigene Bitmaps benötigt
- Später wurden TrueType-Fonts entwickelt, die alle Buchstaben als frei skalierbare Vektorgrafiken beschreiben
- Die Umwandlung in Bitmaps erfolgt erst im finalen Maßstab
- Drehen und Skalieren von Schrift hat keinen Einfluss mehr auf die Darstellungsqualität

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Bildschirm

- Da alle Bildschirminhalte letztlich als komplette Bitmap des gesamten Bildschirms an das Display weitergegeben werden, ergeben sich erhebliche Leistungsanforderungen an die Grafik-Hardware
- Bei Full-HD-Auflösung (1920x1080 Pixel) und 24 Bit Farbtiefe muss der Grafikspeicher  $2.073.600 \text{ Pixel} \times 3 \text{ Byte}$  oder rund 6,2 MB Daten für ein Bild speichern
- Bei 60 Frames pro Sekunde sind theoretisch 60 vollständige Änderungen des gesamten Bildinhalts möglich, also rund 370 MB pro Sekunde an kontinuierliche Videodaten erforderlich



Quelle: AMD

# Betriebssysteme

## Einige Geräte und Gerätetreiber Bildschirm

- Aktuelle Bildschirme mit Touchscreen gehen über das klassische WIMP-Modell hinaus
- Neben dem Zeigen per Pointer werden Gesten unterstützt
- Bewegungen mit mehreren Fingern, die weitere Funktionen auslösen und die Bedienung noch intuitiver machen



Quelle: [www.elektronikpraxis.vogel.de](http://www.elektronikpraxis.vogel.de)

# Betriebssysteme

Einige Geräte und Gerätetreiber

Bildschirm

- Auch 3D-Funktionen werden in Benutzeroberflächen integriert
- Geräte registrieren ihre Orientierung und drehen den Bildschirminhalt
- Sensoren an den Extremitäten nehmen deren Position im Raum auf und bewegen Objekte in einer zweidimensionalen Abbildung des Raumes auf einem Bildschirm
- Astronomie-Apps registrieren Position und Blickrichtung der Kamera eines Smartphones und erläutern die sichtbaren Himmelskörper
- In Virtual Reality-Systemen wird die Anzeige in eine Maske integriert und folgt den Bewegungen des Kopfes



Quelle: GettyImages/Deaglerez

# Betriebssysteme

## Einige Geräte und Gerätetreiber

### Bildschirm

- Allerdings ist die Displaytechnik selbst noch nicht tatsächlich dreidimensional
- Das Betriebssystem steuert nach wie vor nur ein einzelnes oder mehrere 2D-Displays an, die Berechnung der verschiedenen Bildinhalte obliegt den Applikationen

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - Speicherverwaltung
  - Dateisysteme
  - Ein- und Ausgabe
  - **Multiprozessorsysteme**
  - Virtualisierung

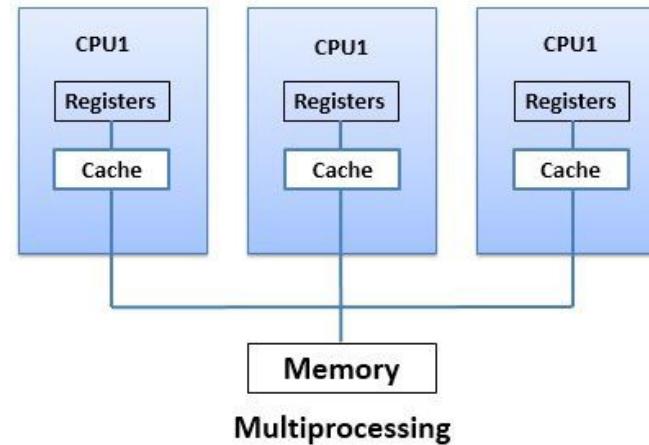
Praxis: Umgang mit Linux (Prof. Brunner) und Windows Server 2019

# Betriebssysteme

## Multiprozessorsysteme

### Definition

- Multiprozessorsysteme sind **Rechner mit mehr als einer CPU, die sich einen gemeinsamen Speicher teilen**

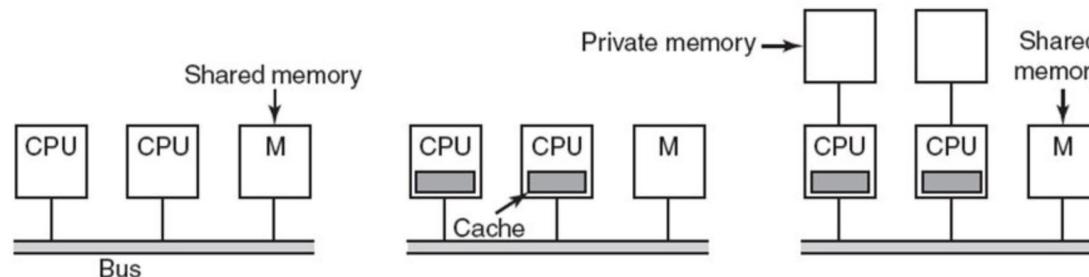


# Betriebssysteme

## Multiprozessorsysteme

### Architekturen

- Zu unterscheiden ist die Zugriffsart auf diesen Speicher, sie ist entweder gleichberechtigt (uniform) oder nicht gleichberechtigt (non uniform)
- Uniformer Zugriff (Uniform Memory Access UMA) kann mit lokalem Cache oder privatem Speicher kombiniert werden



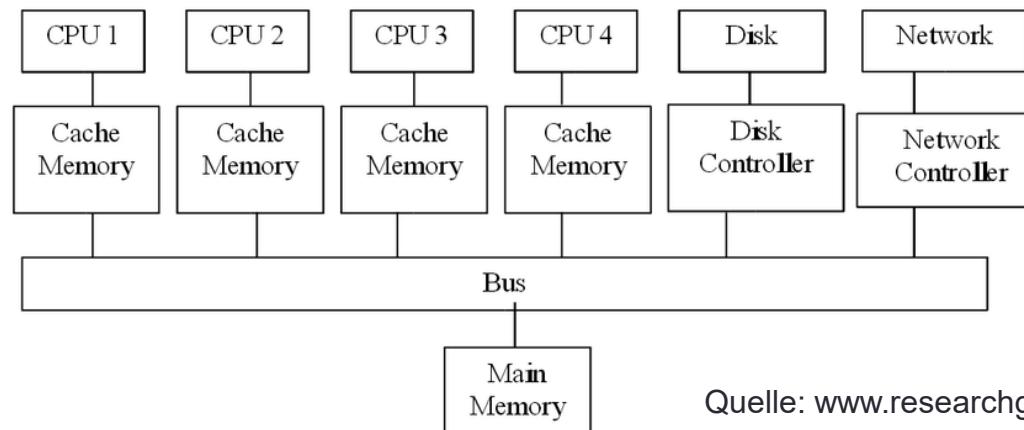
Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Architekturen

- Im UMA-Multiprocessing können die Prozessoren **gleichrangig** oder ungleichrangig behandelt werden:
- **Symmetrisches Multiprocessing...**



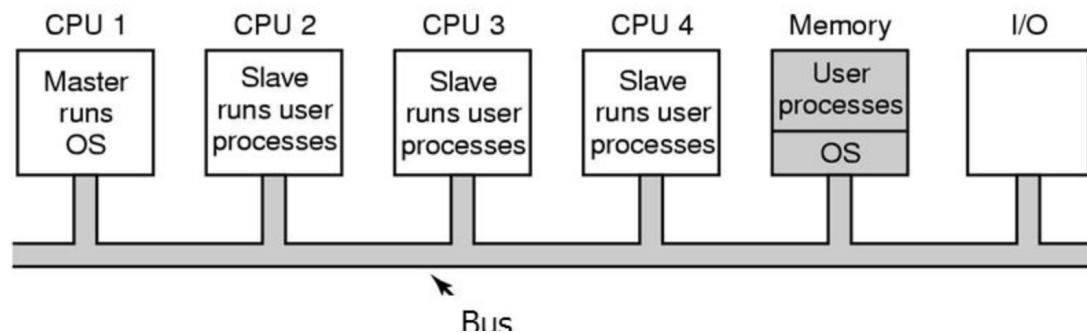
Quelle: [www.researchgate.net](http://www.researchgate.net)

# Betriebssysteme

## Multiprozessorsysteme

### Architekturen

- Im UMA-Multiprocessing können die Prozessoren gleichrangig oder **ungleichrangig** behandelt werden:
- ...und **asymmetrisches** Multiprocessing



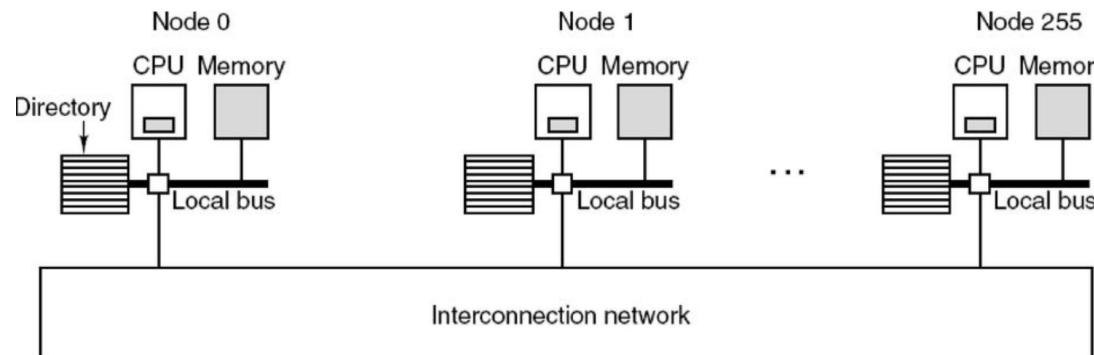
Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Architekturen

- Im **NUMA-Multiprocessing** besitzen die Prozessoren dagegen keinen gleichrangigen Speicherzugriff, sondern sind netzwerkgekoppelt
- Der Übergang zu Multicore Computing ist fließend

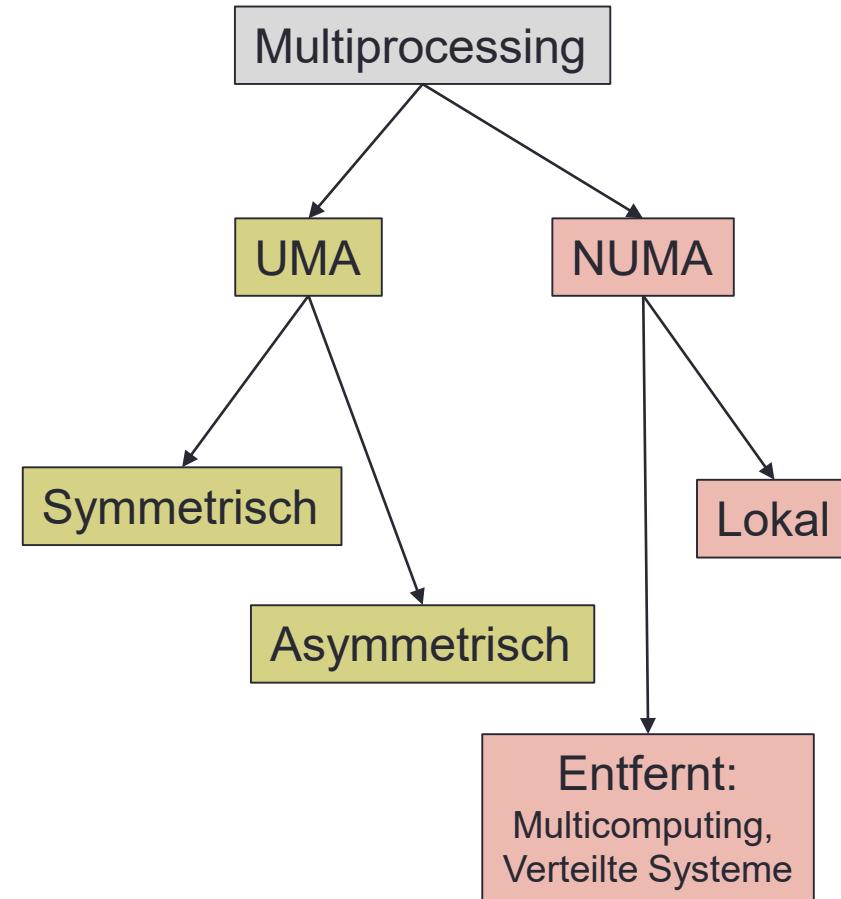


Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme Architekturen - Übersicht

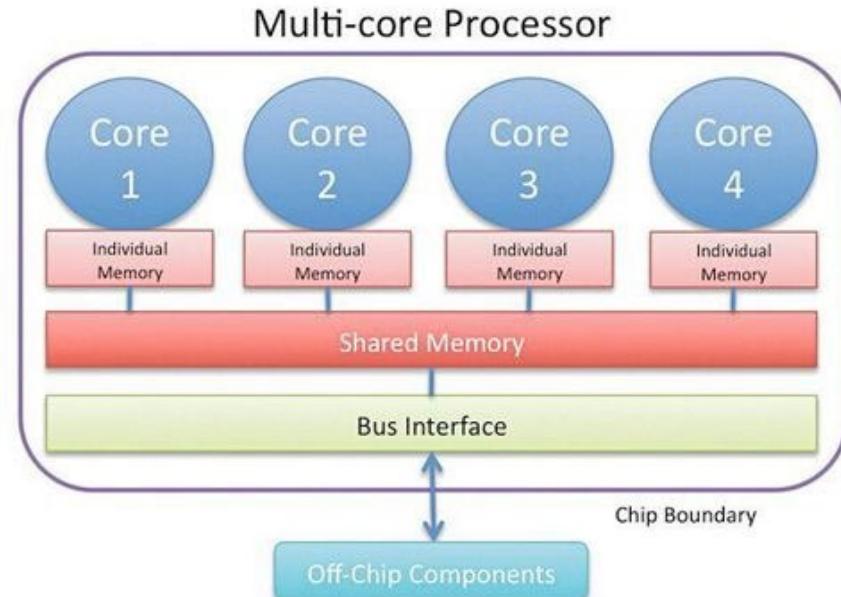
- Uniform Multiprocessing UMA
  - => gleichrangiger Speicherzugriff
    - Symmetrisches Multiprocessing
    - Asymmetrisches Multiprocessing
- Non Uniform Multiprocessing NUMA
  - => nicht gleichr. Speicherzugriff
    - Verbindung über Netzwerk
    - Lokal
    - Entfernt => verteilte Systeme, lose gekoppelt, Multic computing



# Betriebssysteme

## Multiprozessorsysteme Architekturen - Übersicht

- In der Praxis ist für einzelne Computer das Symmetrische Multiprocessing SMP übrig geblieben: **Multicore-CPU**s
- Jeder CPU-Kern besitzt eigene L1- und L2-Caches
- Alle Kerne teilen sich den L3-Cache
- Gemeinsamer RAM außerhalb des Chips



Quelle: [www.embedded-software-engineering.de](http://www.embedded-software-engineering.de)

# Betriebssysteme

Multiprozessorsysteme

Symmetrisches Multiprocessing

Funktionsweise (meistens):

- Ein Betriebssystem verwaltet alle Prozessoren
- Prozesse werden dynamisch auf die Prozessoren verteilt
- Es besteht gemeinsame Datennutzung in RAM und Dateisystem
- Es gibt echte Parallelität für Prozesse und damit höhere Leistung

Limitierung:

- Das System skaliert schlecht, nicht mehr als 64 CPUs – **Warum?**
- Hohe Skalierung nur mit Multicore und Supercomputern

# Betriebssysteme

Multiprozessorsysteme

Symmetrisches Multiprocessing

Limitierung:

- Der gemeinsame **Datenbus** ist wieder einmal der **Flaschenhals**
- **Race-Conditions** nicht nur zwischen Prozessen, sondern auch zwischen Prozessoren
- Der **Cache** ist als Ganzes **nicht kohärent**, zwischen den individuellen Caches können **Unterschiede** auftreten

# Betriebssysteme

Multiprozessorsysteme

Symmetrisches Multiprocessing

Anforderungen:

- Das Betriebssystem muss in der Lage sein, mehrere laufende Prozesse effizient auf die Prozessoren zu verteilen
- Keine CPU soll leer laufen, keine überlastet sein
- Wie in Einzelprozessorsystemen müssen gemeinsam genutzte Betriebsmittel verwaltet werden:
  - Mutexe, Semaphoren usw. zur Synchronisation und Scheduling auf Prozessorebene

# Betriebssysteme

Multiprozessorsysteme

Symmetrisches Multiprocessing

Anforderungen:

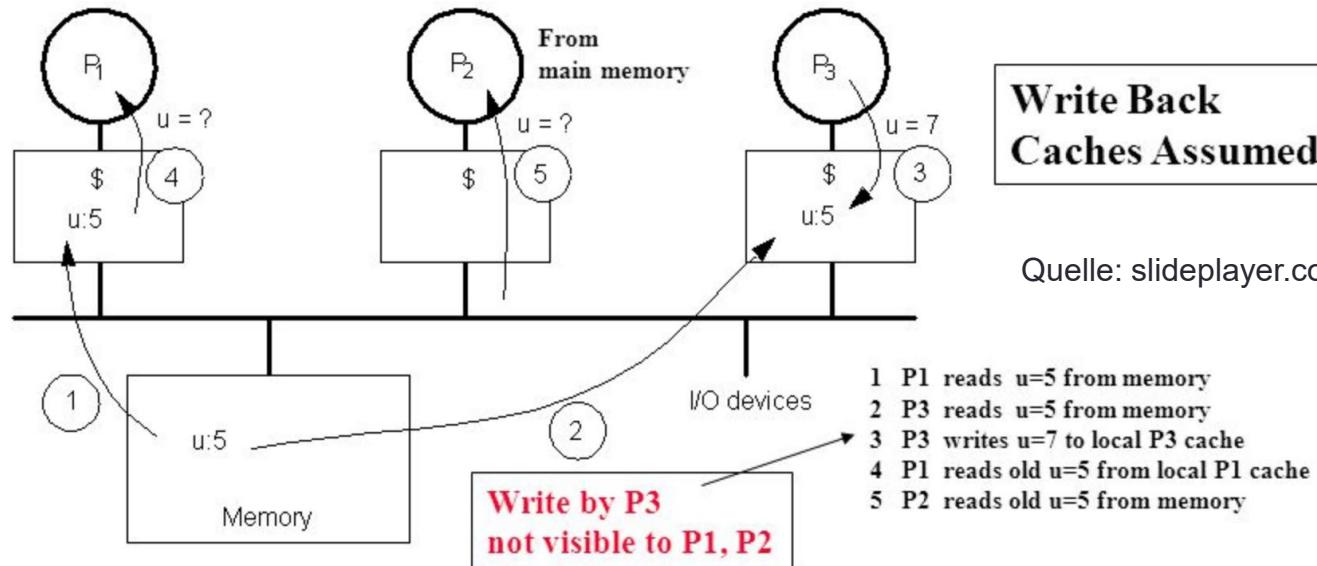
- Neu: Das System muss die Cache-Kohärenz überwachen und sicherstellen
- Neu: Erweiteres Interrupt Handling, Verteilung der Interrupts auf alle CPUs und Interrupts zwischen CPUs

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Cache-Kohärenz

- Das System muss die Cache-Kohärenz überwachen und sicherstellen



# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Cache-Kohärenz

- Das System muss die Cache-Kohärenz überwachen und sicherstellen
- Das passiert über das MESI-Protokoll, das für jede Cache-Zeile zwei zusätzliche Bits und damit vier mögliche Zustände definiert
  - **Modified**: Daten im lokalen Cache wurden geändert, Hauptspeicherkopie ungültig
  - **Exclusive**: Daten liegen nur in diesem Cache vor und sind aktuell, die Hauptspeicherkopie ist gültig
  - **Shared**: Daten liegen identisch in mehreren Caches vor, es gab keine lokalen Änderungen, die Hauptspeicherkopie ist gültig
  - **Invalid**: Der Inhalt der Cache-Zeile ist ungültig oder es existiert kein Eintrag

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Cache-Kohärenz

- Das MESI-Protokoll gibt außerdem vor, dass alle Prozessoren ihre eigenen Cache-Zugriffe in diese Bits schreiben und auf die Veränderungen durch andere Prozesse achten
- Zur Überwachung der Veränderungen lesen und setzen die Prozessoren definierte Signale auf dem Bus (Bus-Snooping)
- Alle Prozessoren achten so gemeinsam auf die Cache-Kohärenz
- Nicht machbar bei vielen Prozessoren, Skalierungsproblem!

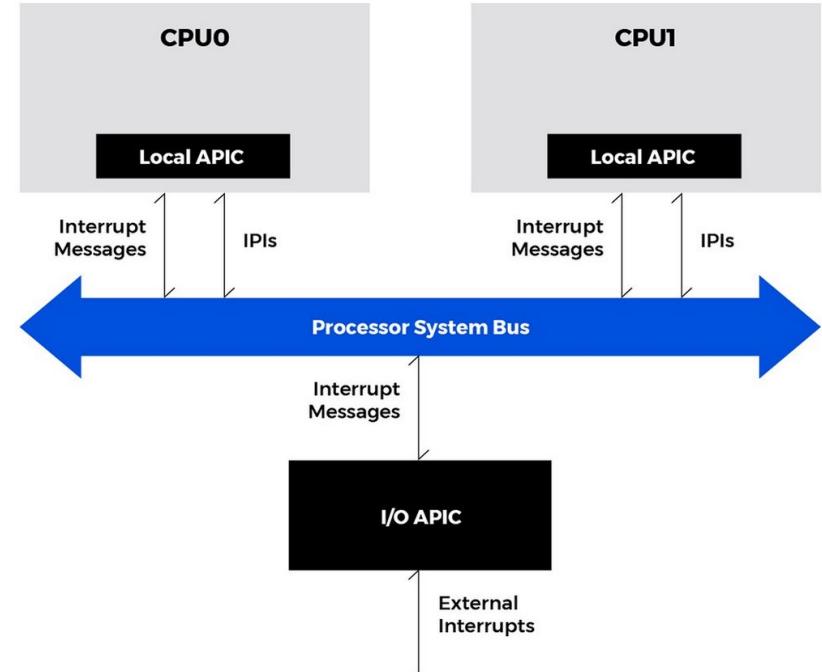
# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Interrupt Handling

#### Anforderungen:

- Das Interrupt-Handling kann an die Hardware delegiert bleiben
- Spezielle Applikationsabhängig programmierbare Bausteine **Application Programmable Integrated Circuits APIC** im Intel-Chipsatz und lokale APICs in jeder CPU kommunizieren miteinander



Quelle: miro.medium.com

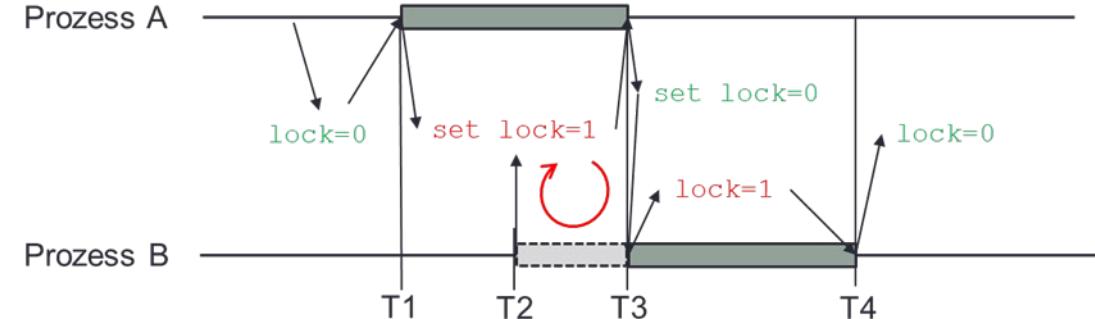
# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Interrupt Handling

- CPUs können sich gegenseitig aus dem Suspend (Schlafzustand mit niedrigem Energieverbrauch) wecken
- Aber: Maschinenbefehle zum Sperren und Erlauben wirken je Prozessor
- Dies muss bei der Synchronisation berücksichtigt werden

# Betriebssysteme



## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Synchronisation

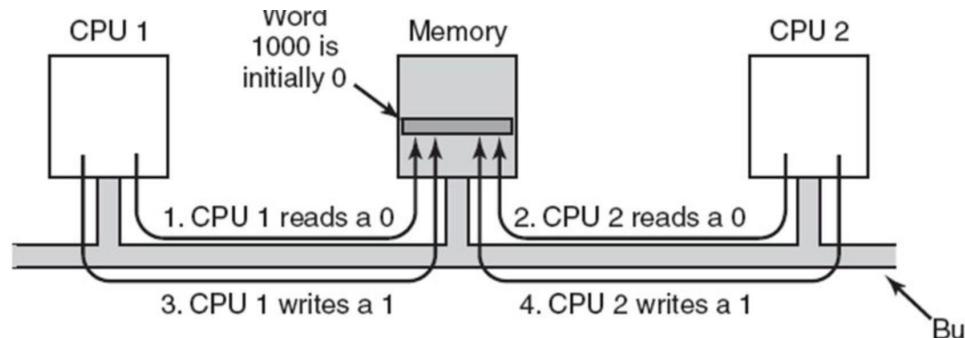
- Synchronisation beschreibt hier die Vermeidung von Race Conditions zwischen Prozessoren
- Wie bei Einzelprozessorsystemen kann **Test, Set & Lock** für das Sperren einer gemeinsam genutzten Ressource eingesetzt werden, am besten auf Hardwareebene, TSL-Befehl
- **Problem:** Prozesse laufen jetzt nicht mehr sequenziell in Zeitscheiben sondern tatsächlich parallel

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Synchronisation

- Ein Prozessor muss beim Eintritt eines seiner Prozesse in die kritische Region sicherstellen, dass er dabei der einzige bleibt...



Quelle: Tanenbaum und Bos (2016)

- Dies gelingt z.B. durch **Sperren** des gemeinsamen Busses

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Synchronisation

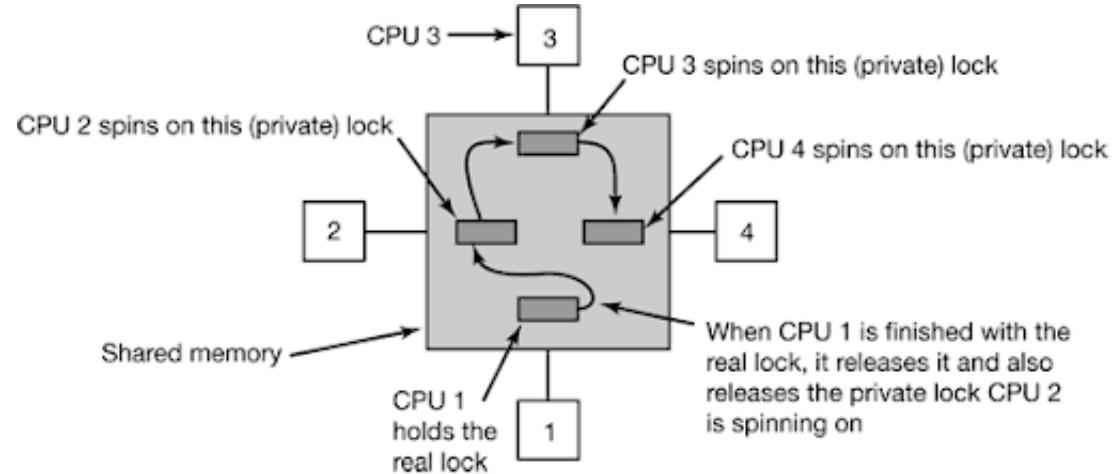
- Allerdings muss der Bus dann bei jedem TSL gesperrt und wieder freigegeben werden, was die Performanz reduziert
- Eine wirksame Alternative ist das **Individualisieren der Sperrvariable**: Jeder Prozessor erhält eine eigene Variable im gemeinsam zugänglichen Cache-Bereich (schneller Zugriff)

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Synchronisation

- Prozessoren erhalten beim Fehlschlag des Eintritts in eine kritische Region eine individuelle Sperre und reihen sich in eine Schlange wartender Prozessoren ein
- Wird die kritische Region vom Prozessor, der sie gerade genutzt hat, wieder freigegeben, schließt dies auch die individuelle Sperre des nächsten wartenden Prozessors ein
- Was bleibt, ist das Problem des Wartens: Jeder Prozessor, der auf eine Ressource wartet, befindet sich im **Spinlock** und fragt immer wieder die Variable ab



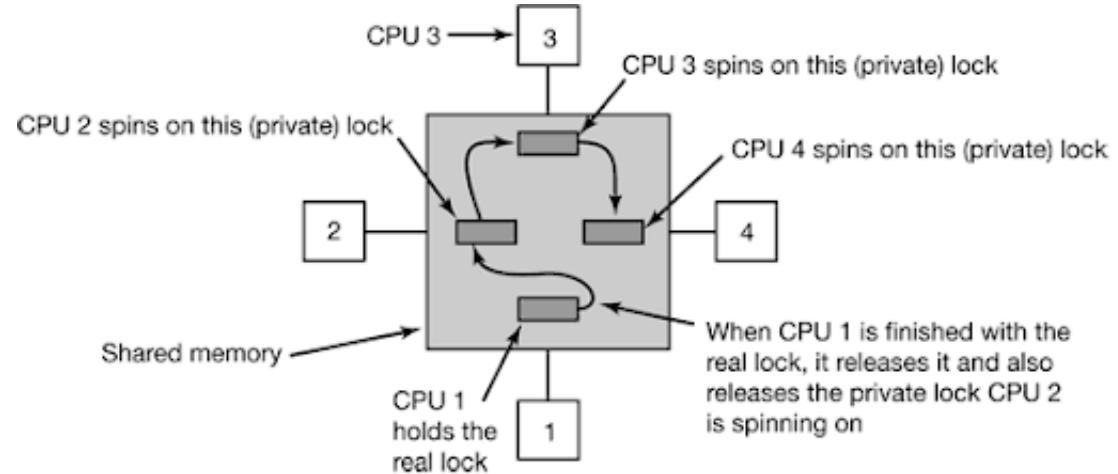
Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Synchronisation

- Die Alternative zum aktiven Warten im Spinlock wäre blockieren des betreffenden Prozesses und ein Kontextwechsel
- Aber: auch ein Kontextwechsel erzeugt Overhead und Wartezeiten
- Es kann folglich effizienter sein, einfach nur zu warten und zu hoffen, dass dies kürzer und weniger aufwendig ist als Kontextwechsel...



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Andere, komplexere Scheduling-Algorithmen als bisher
- Mehrere Ebenen: Scheduler wählt **Prozess und Prozessor**
- Prozesse und Threads auf einzelnen Prozessoren
- Prozessorübergreifendes Scheduling
- Die Lösungen sollen möglichst transparent für Applikationen und Programmierer sein
  - Time Sharing
  - Space Sharing
  - Gang Sharing

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Time Sharing basiert auf einer einzigen Bereit-Warteschlange für alle CPUs und verteilt die Arbeitslast
- Alternative Bezeichnungen sind **Load Sharing** und **Single Queue Multiprocessor Scheduling**

## Funktionsweise

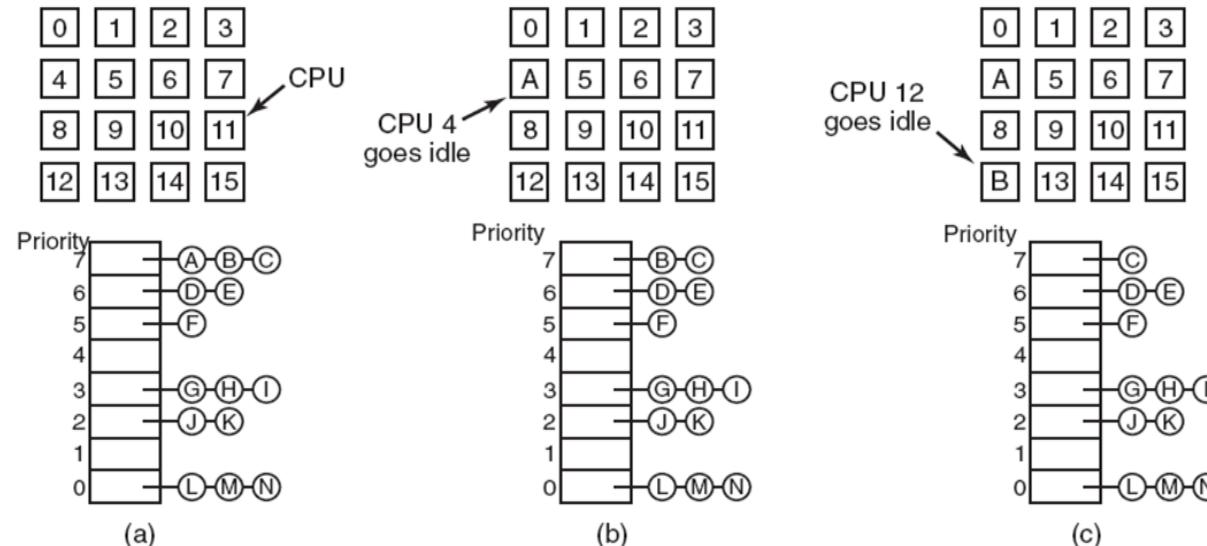
- Alle CPUs sind gleichberechtigt
- Der Scheduler weist jeder CPU den nächsten Thread oder Prozess zu, der an der Reihe ist

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Time Sharing



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Time Sharing ist einfach und transparent für Programme
- Liefert Lastenausgleich (load balancing) zwischen den CPUs
- Sorgt dafür, dass alle CPUs ausgelastet sind

### Probleme

- **Keine Prozessoraffinität:** Prozesse wechseln zwischen CPUs, die einen eigenen Cache besitzen; Cache-Inhalte gehen bei jedem Kontextwechsel verloren
- **Zusammengehörige Threads** werden **nicht koordiniert** ausgeführt und können **nicht miteinander kommunizieren**

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Space Sharing löst diese Probleme
- Zusammengehörige Threads werden gruppiert (maximal so viele wie es CPUs gibt, Limit des Space Sharing)
- Threadgruppen werden Prozessorgruppen zugeordnet und laufen darin parallel ab, bis alle abgeschlossen sind:

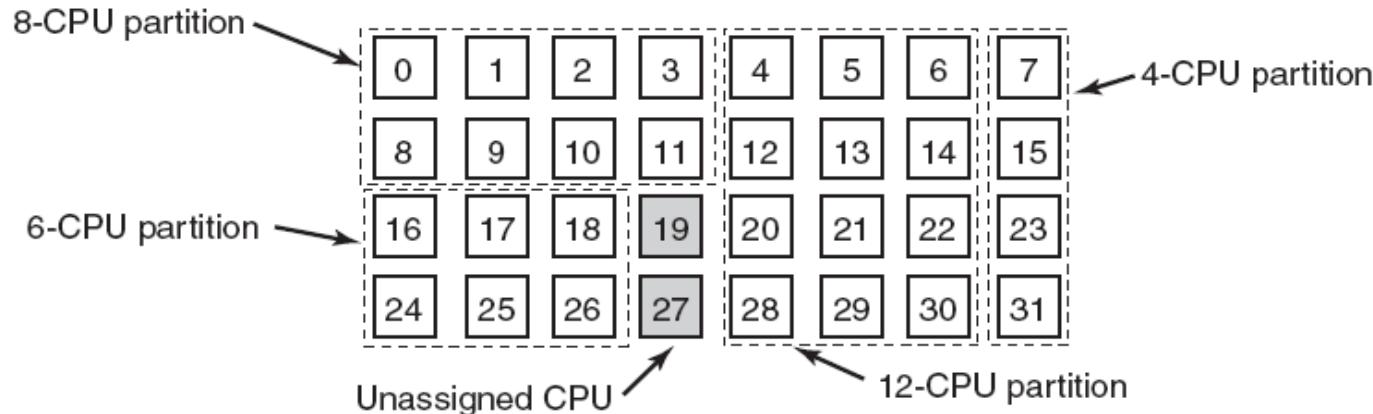
### Affinity Scheduling

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Space Sharing



# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Space Sharing
- Zusammengehörige Threads laufen gleichzeitig
- Keine Multiprogrammierung, kein Kontext-Switching, kein Cache-Verlust

### Probleme

- Möglicherweise Leerlauf einzelner CPUs
- Keine Kommunikation für Threads, die nicht gleichzeitig laufen

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Gang Scheduling
- Alle zusammengehörigen Threads laufen gleichzeitig als Einheit (Gang)
- Wie beim Multitasking in Zeitscheiben auf mehreren CPUs
- Gemeinsamer Start, gemeinsame Kontextwechsel, gemeinsamer Abschluss aller Threads der Einheit

	CPU					
	0	1	2	3	4	5
0	A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
1	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>
2	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	E <sub>0</sub>
3	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>5</sub>	E <sub>6</sub>
4	A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
5	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>
6	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	E <sub>0</sub>
7	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>5</sub>	E <sub>6</sub>

Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- Gang Scheduling
- Ermöglicht verzögerungsfreie Interprozesskommunikation
- Die Leerlaufzeit wird reduziert, weil ein blockierter Prozess nur in seiner Zeitscheibe nichts tut, die CPU kann andere Zeitscheiben nutzen

# Betriebssysteme

## Multiprozessorsysteme

### Symmetrisches Multiprocessing – Scheduling

- An Algorithmen für Multiprozessor-Scheduling wird aktuell noch intensiv geforscht
- Insbesondere für RTOS ist noch Luft für die Optimierung

# Themenübersicht

Inhalte, Umfang und Ablauf

## 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- **Aufgaben des Betriebssystem im Detail**
  - Prozesse und Threads
  - Speicherverwaltung
  - Dateisysteme
  - Ein- und Ausgabe
  - Multiprozessorsysteme
  - **Virtualisierung**

Praxis: Umgang mit Windows Server 2019

# Betriebssysteme

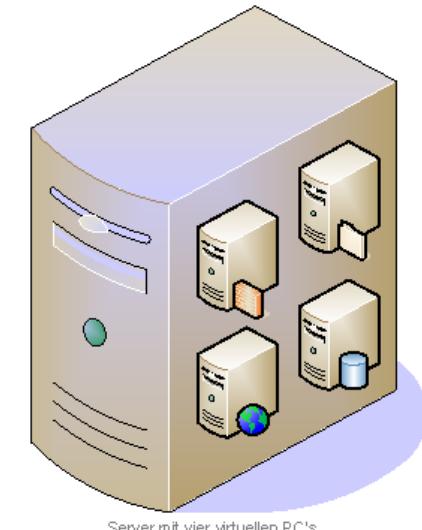
## Virtualisierung – Begriffe

- Virtualisierung bezeichnet im Zusammenhang mit Betriebssystemen allgemein eine **Abstraktion der Hardware** zur einfacheren und flexibleren Nutzung durch Software und Anwender
- Hardware kann **innerhalb eines Betriebssystems** für dessen eigene Zwecke und die laufenden Applikationen abstrahiert werden, wie im Konzept der Gerätetreiber für Betriebssysteme erläutert
- Wird das Konzept der Hardware-Abstraktion vollständig umgesetzt, steht anstelle einer physischen Hardware schließlich eine rein **virtuelle Maschine** (VM)

# Betriebssysteme

## Virtualisierung – Begriffe

- Virtuelle Maschinen laufen als vollwertige Rechner nicht direkt auf einer physischen Hardware, sondern sie arbeiten als **virtuelle Gastsysteme** in einem Wirtssystem aus physischer Hardware und einer besonderen **Virtualisierungsschicht**, dem **Virtual Machine Monitor** oder **Hypervisor**
- **Hardware-Virtualisierung**
- Vergleichbare Konzepte bestehen für Storage, Programmierung (Java-VM), Applikationen....



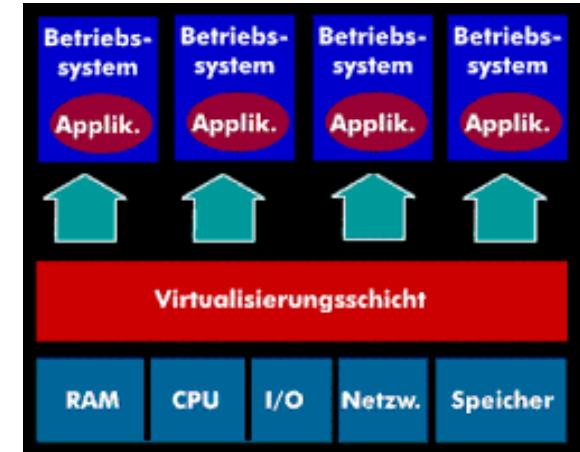
Server mit vier virtuellen PC's

Quelle: <http://www.endler-web.de>

# Betriebssysteme

## Virtualisierung – Konzept

- Auf einem geeigneten Rechner wird die Software zur Virtualisierung installiert
- Die Software etabliert einen **Hypervisor**
- In der Software werden eine oder mehrere Virtuelle Maschinen (VMs) angelegt, i.e. die virtuelle Hardware
- Auf den VMs werden Betriebssysteme und die benötigten Serverdienste oder Applikationen installiert



Quelle: [www.itwissen.info](http://www.itwissen.info)

# Betriebssysteme

## Virtualisierung – Motivation

- Hardware-Virtualisierung bietet eine Reihe von Vorteilen
- Ein physischer Rechner kann mehrere virtuelle Maschinen unterstützen und so die vorhandene Hardware besser ausnutzen (Auslastung optimieren)
- Auslastungsschwankungen verschiedener Serverdienste können durch geschickte Kombination virtueller Server auf physischen Servern besser ausgeglichen werden

# Betriebssysteme

## Virtualisierung – Motivation

- **Hochverfügbarkeit** durch Online-Migration der VMs zwischen physischen Servern
- Für Test- und Schulungsumgebungen können komplexe Rechnersysteme mit deutlich weniger Hardware und damit **kostengünstiger** realisiert werden
- Bei Bedarf können **historische Applikationen und Betriebssysteme** auf einer VM installiert und betrieben werden, obwohl die aktuelle Hardware diese nicht mehr unterstützt

# Betriebssysteme

## Virtualisierung – Motivation

Nachteilig sind

- Der **Overhead durch das Wirtssystem** und die notwendige Koordination der Gastsysteme: 5-10%
- Die **Konkurrenz** der Gastsysteme um die verfügbare Hardware
- Probleme bei besonderer Hardware, z.B. Dongles
- Umfangreiche Auswirkung des Ausfalls eines physischen Servers, erfordert zusätzliche Redundanzen und komplexes Management der VMs
- Hohe Komplexität der Datensicherungen von VMs

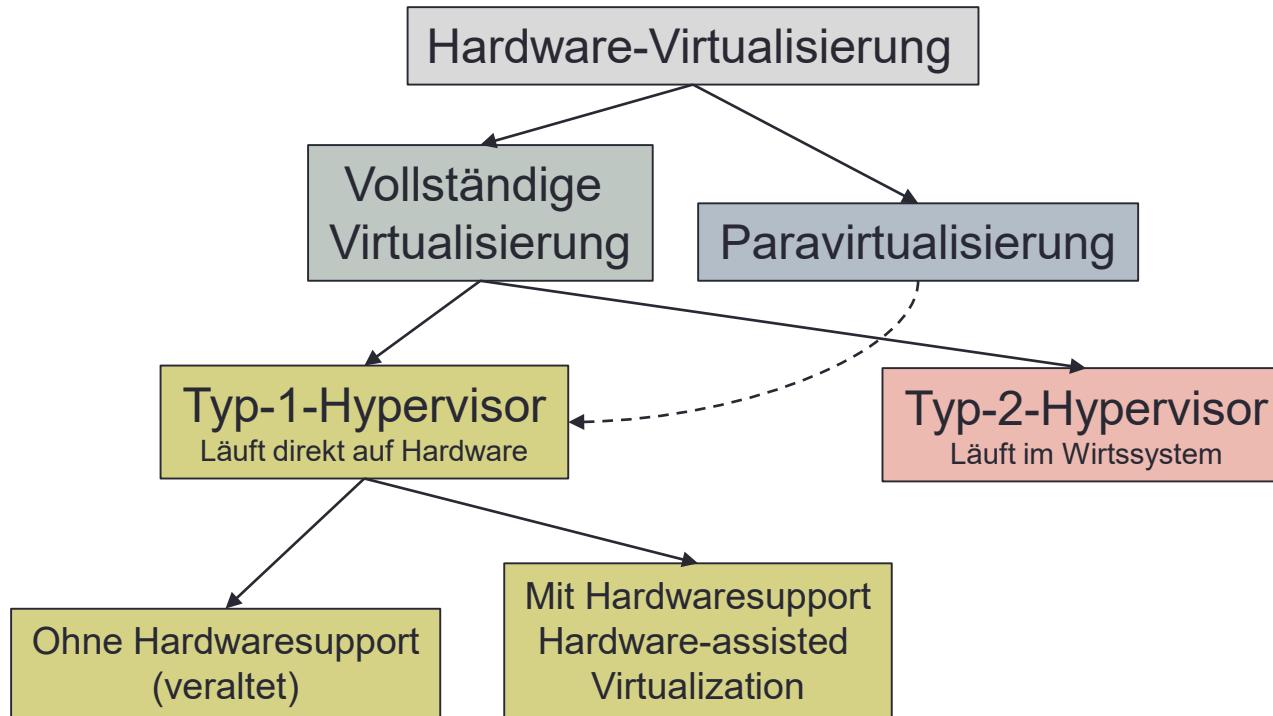
# Betriebssysteme

## Virtualisierung – Architekturen

- Hardware-Virtualisierung wird nach der Funktion des Hypervisors in verschiedene Kategorien eingeteilt:
- Nach der **Rolle des Hypervisors bei der Interaktion des Gastsystems mit der Hardware** wird zwischen **vollständiger Virtualisierung** und **Paravirtualisierung** unterschieden
- Nach der **Position des Hypervisors im Schichtenmodell** werden **Typ-1-** und **Typ-2-Virtualisierung** unterschieden

# Betriebssysteme

## Virtualisierung – Architekturen Übersicht



# Betriebssysteme

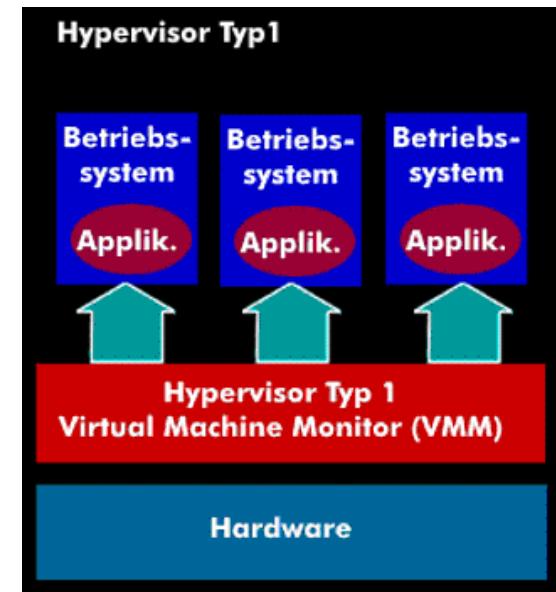
## Virtualisierung – Architekturen

### Typ-1-Virtualisierung

- Der Hypervisor wird direkt auf der Hardware installiert
- Bare-Metal-Hypervisor
- Der Hypervisor ist nur ein Mini-Betriebssystem
- Die Hardware wird vollständig virtualisiert (Vollvirtualisierung)

Beispiele:

- XenServer von Citrix Systems, vSphere ESX von VMware, Hyper-V von Microsoft



Quelle: [www.itwissen.info](http://www.itwissen.info)

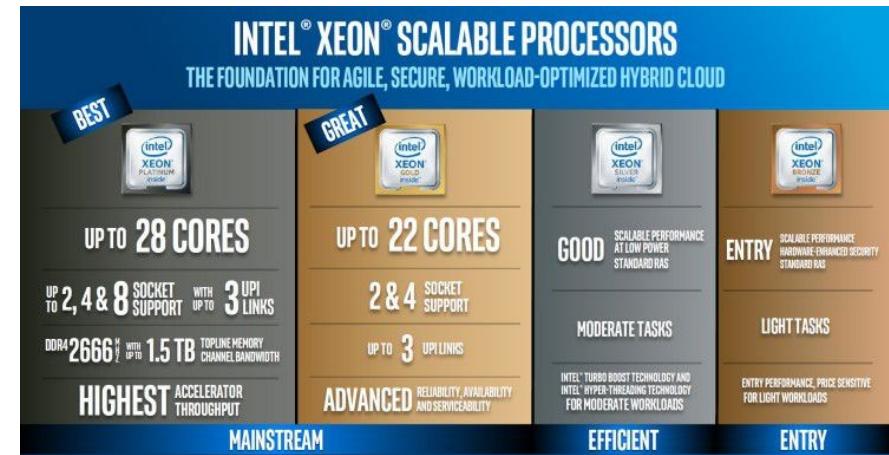
# Betriebssysteme

## Virtualisierung – Architekturen

- Geeignete Rechnerinfrastruktur
  - Ausreichend Rechenleistung, CPU-Cores, RAM, Storage....
- Hardware-Support für Typ-1-Virtualisierung erforderlich
  - vollkommende Isolierung gleichzeitig bearbeiteter Aufgaben und genutzter Ressourcen

Beispiele:

- Intels Virtualisierungstechnik VT-x und Multicore-Prozessoren
- AMD-V oder Pacifica



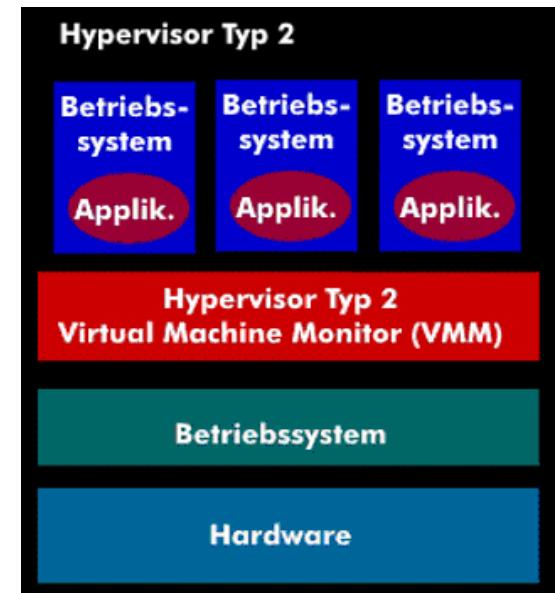
Quelle: intel

# Betriebssysteme

## Virtualisierung – Architekturen

### Typ-2-Virtualisierung

- Hypervisor läuft als Anwendung auf einem Betriebssystem
- Nutzt Ressourcen des OS und nicht die Hardware
- Der Hypervisor emuliert die Hardware für das Gastsystem
- Vollständige Virtualisierung
- Beispiele: Microsoft Virtual PC, VMware Workstation



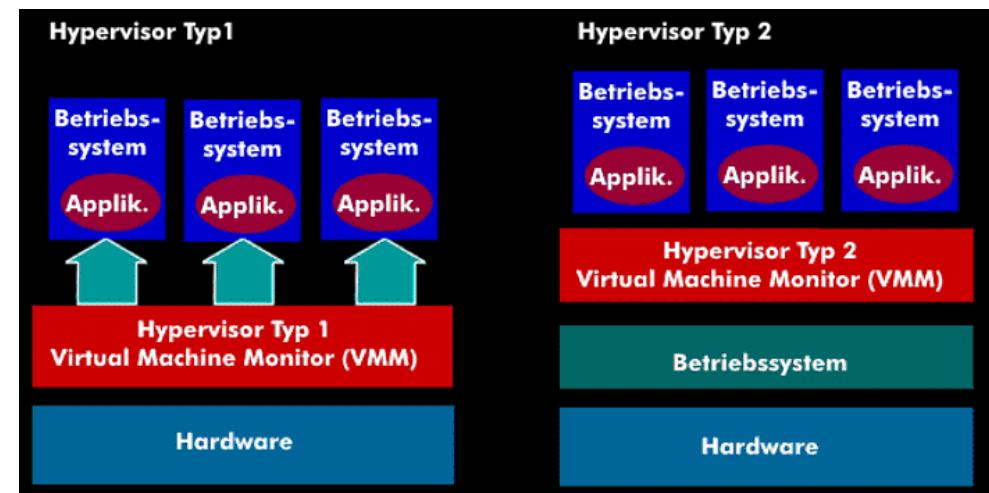
Quelle: [www.itwissen.info](http://www.itwissen.info)

# Betriebssysteme

## Virtualisierung – Architekturen

### Vergleich Typ 1 und Typ 2

- Typ-1-Virtualisierung
  - Geringer Overhead
  - Produktiveinsatz
- Typ-2-Virtualisierung
  - Hardwareunabhängig für Gastsystem, keine Treiber nötig
  - Mehr Overhead
  - Für Testumgebungen usw.
- Beide Typen **virtualisieren die Hardware vollständig und transparent für Gastbetriebssysteme**



Quelle: [www.itwissen.info](http://www.itwissen.info)

# Betriebssysteme

## Virtualisierung – Architekturen

### Aufgaben des Hypervisors

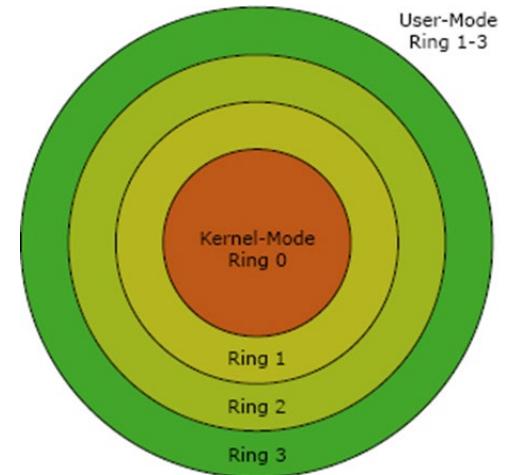
- In der Typ-1- und Typ-2-Virtualisierung gehen Gastbetriebssysteme davon aus, selbst direkt auf der Hardware zu laufen
- Das fordert vom Hypervisor einige aufwändige Leistungen:
- Trennung in Benutzermodus und Kernmodus aufheben bzw. verschieben
- Speicherseiten der Gastbetriebssysteme verwalten
- I/O-Anforderungen und Interrupts kontrollieren und korrekt zuweisen

# Betriebssysteme

Virtualisierung – Architekturen

Benutzermodus und Kernmodus, Typ-1-Hypervisor

- Der Kern eines Betriebssystem arbeitet normalerweise im privilegierten **Kernel-Modus** der Prozessoren (Ring 0)
- Für das Gastbetriebssystem in einer VM trifft das aber nicht zu, es arbeitet im **Benutzermodus** (Ring 1)
- Applikationen im Gastsystem arbeiten jedenfalls im Benutzermodus
- Der Hypervisor arbeitet selbst im Kernel-Modus und muss es dem Gastsystem ermöglichen, ebenfalls Befehle im Kernel-Modus auszuführen



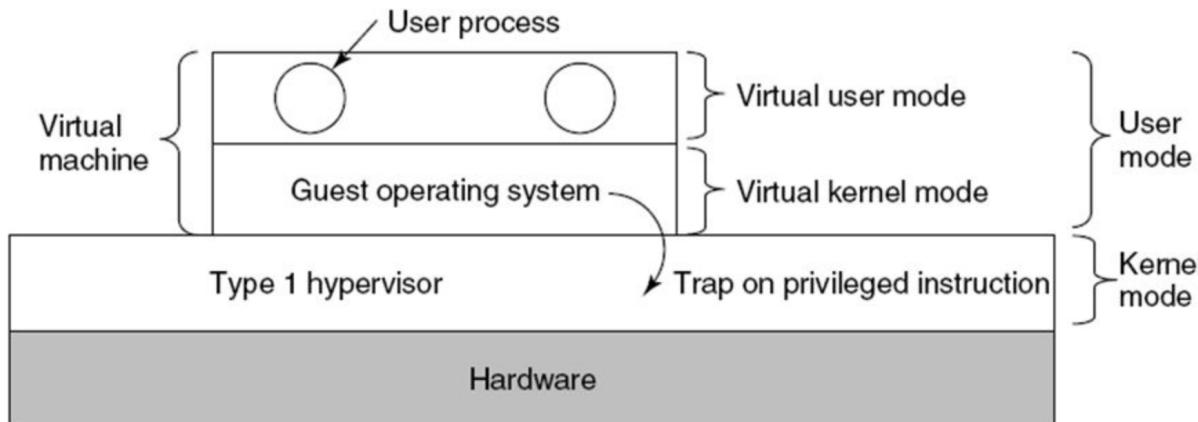
Quelle: [www.knausware.de](http://www.knausware.de)

# Betriebssysteme

## Virtualisierung – Architekturen

### Benutzermodus und Kernmodus, Typ-1-Hypervisor

- Wird Virtualisierung durch die **Hardware** unterstützt (VT oder AMD-V), führt ein privilegierter Befehl des Gast-OS zu einem **Sprung in den Hypervisor**



Quelle: Tanenbaum und Bos (2016)

# Betriebssysteme

## Virtualisierung – Architekturen

### Benutzermodus und Kernmodus, Typ-1-Hypervisor

- Der Hypervisor prüft jetzt, woher der privilegierte Befehl kommt
- Stammt er vom Gastbetriebssystem, veranlasst er die Ausführung auf der Hardware im Namen des Gastsystems
- Stammt er von einer Applikation im Gastsystem, emuliert der Hypervisor die Aktivitäten der Hardware
- Diese Form der Vollvirtualisierung wird auch **Hardware-assisted Virtualization** genannt
- Sie erzeugt weniger Overhead als herkömmliche Vollvirtualisierung

# Betriebssysteme

## Virtualisierung – Architekturen

### Benutzermodus und Kernmodus, Typ-2-Hypervisor

- Ein Typ-2-Hypervisor läuft selbst als Prozess im Benutzermodus des Wirtsbetriebssystems und kann keine Kernel-Modus-Befehle ausführen
- Er muss daher alle Kernel-Modus-Befehle des Gastsystems durch Prozeduraufrufe ersetzen und vom Wirtssystem ausführen lassen
- Zum Gastsystem hin emuliert der Typ-2-Hypervisor damit die Hardware

# Betriebssysteme

Virtualisierung – Architekturen

Speicherseitenverwaltung

- Laufen mehrere Gastsysteme auf einem Hypervisor, ist nicht auszuschließen, dass diese zufällig die gleichen physischen Seitenrahmen aufrufen möchten
- Der Hypervisor verhindert dies, in dem er **Schattenseitentabellen** führt
- Darin weist er die Seitenrahmen jedes Gastbetriebssystems einem Seitenrahmen im echten Speicher

# Betriebssysteme

## Virtualisierung – Architekturen

### Paravirtualisierung

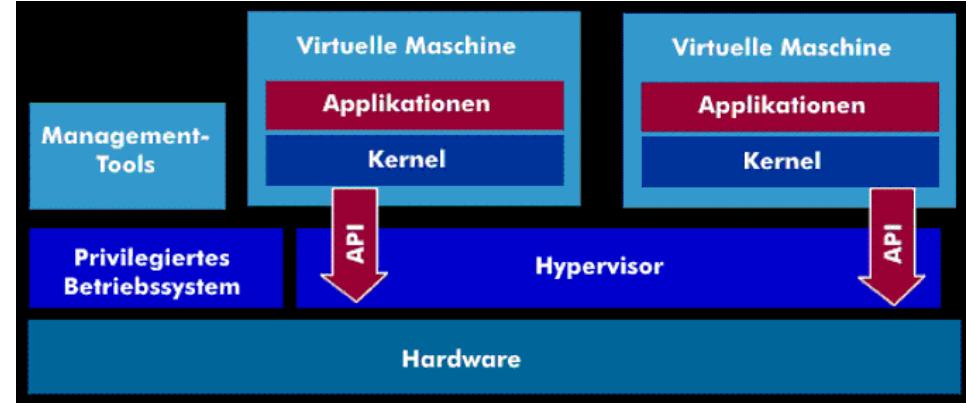
- Typ-1- und Typ-2-Virtualisierung bieten den Gastbetriebssystemen eine vollständig virtualisierte Hardware, die Virtualisierung ist für das Gastsystem transparent
- Die vollständige Virtualisierung ist aufwändig und vorteilhaft, aber keineswegs immer erforderlich
- Wenn Gastbetriebssysteme bekannt und Quelloffen sind, können **Anpassungen zur Optimierung der Leistung in einer VM** vorgenommen werden

# Betriebssysteme

## Virtualisierung – Architekturen

### Paravirtualisierung

- Bei der Paravirtualisierung werden Gastsysteme zunächst **portiert**, d.h. ihr Kernel wird an das Laufen auf einem Hypervisor angepasst
- Anstelle von Hardwareaufrufen kommuniziert das Gastsystem mit dem Hypervisor über **spezielle APIs**
- Der Hypervisor wird zum Microkernel



Quelle: www.itwissen.info

Vorteil: Bessere Leistung der VMs

Beispiele: XenServer, Linux KVM

# Themenübersicht

## Inhalte, Umfang und Ablauf

### 2 Betriebssysteme

- Einführung:
  - Definition Betriebssysteme, Geschichte, Klassen, Aufgaben und Konzepte
- Aufgaben des Betriebssystem im Detail
  - Prozesse und Threads
  - Speicherverwaltung
  - Dateisysteme
  - Ein- und Ausgabe
  - Multiprozessorsysteme
  - Virtualisierung

### Praxis: Umgang mit Windows Server 2019