Betriebssysteme

# Rechnerarchitektur

## Universelle Rechner und Alan Turing

* Alan Turing definiert das mathematische Modell der Turing-Maschine
* diese beschreibt einen universell einsetzbaren Computer
* Turing-Maschine kann alle mathematischen Funktionen berechnen, für die es einen Algorithmus gibt und welche endlich sind
* bisher ist kein Computermodell bekannt, was mehr leisten kann
* heutige Computer sind turingmächtig, jedoch mit der Einschränkung, dass der Speicher nicht unendlich groß ist

## EVA-Prinzip

* zur Verarbeitung gehört auch das temporäre oder permanente Speichern von Daten
* früher war die Abfolge der Schritt rein sequenziell, heute schon teilweise parallel und in Echtzeit

## Von Neumann Architektur

* Befehle und Daten werden binär dargestellt
* Befehle und Daten sind im selben Speicher
* Struktur des Rechners ist unabhängig von der zu bearbeitenden Aufgabe
  + man kann Fotos bearbeiten, Texte schreiben…

**Eingabe-/Ausgabewerk**

* dient zum Anschluss von Peripheriegeräten
* Schnittstelle nimmt eine mechanische, elektronische und logische Signalanpassung zwischen beiden Systemen vor
* Kommunikation zwischen Rechner und Umwelt

**Steuerwerk**

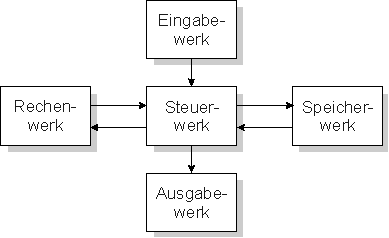
* auch als CU (control unit) oder Leitwerk bezeichnet
* holt Befehle aus RAM
* decodiert Befehle
* steuert die Befehlsausführung
* erhöht Befehlszähler

**Rechenwerk (ALU = arithmetic logic unit)**

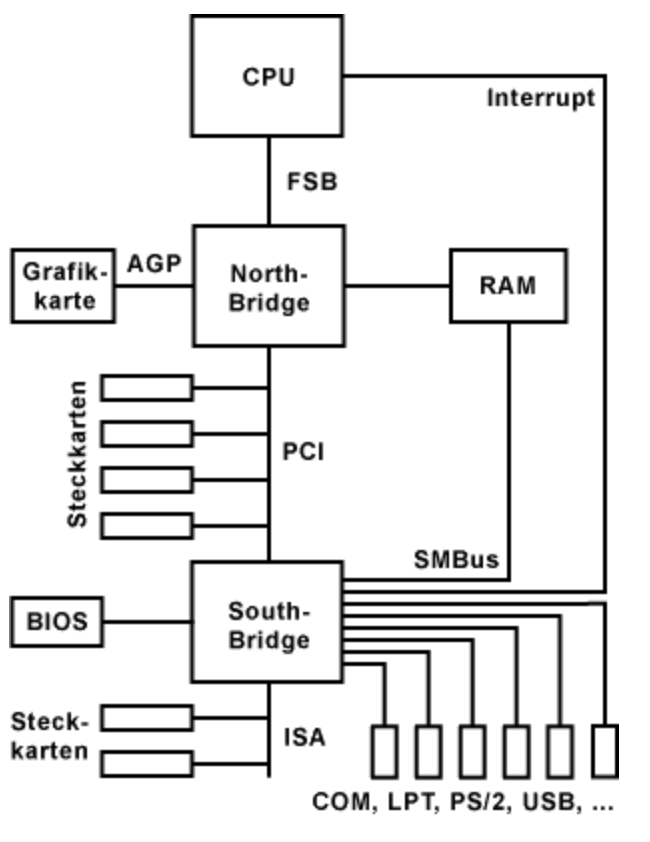
* führt arithmetische und logische Befehle aus
* Operanden werden vom Steuerwerk zur Verfügung gestellt

**Speicherwerk**

* Register: Speicher mit sehr kurzer Zugriffszeit, z.B. zur Speicherung von Zwischenergebnissen
* Akkumulator: Register (AX), in dem vor Ausführung einer Operation die entsprechenden Operanden gespeichert werden



* Vorteile der Von-Neumann-Architektur
  + sequenzielle Abarbeitung vermeidet Inkonsistenzen
  + Anweisungen und Daten können im laufenden Betrieb verändert werden
  + Modell und Programmierung sind gut beherrschbar
  + in der Abarbeitung sind Programmschleifen möglich
* Nachteil der Von-Neumann-Architektur (Flaschenhals)
  + das Bussystem muss Daten und Befehle nacheinander transportieren und wird bei leistungsfähigen Rechenwerken überfordert, es entstehen Wartezeiten
* Lösung des Von-Neumann-Flaschenhals
  + Unterteilung in North- und Southbridge
    - schnelle Northbridge mit Grafik-, Speicher- und Steckkartenzugriff
    - langsame Southbridge übrigen IO-Ports und BIOS



## weitere Verbesserung (Memory Wall)

* CPUs entwickeln sich deutlich schneller als RAM -> Memory Wall bildet den nächsten Flaschenhals
* Lösung: Trennung von Daten und Befehlen und Einsatz von Zwischenspeicher (Cache) in der CPU
* Cache in der CPU ist als SRAM verbaut, ist schneller, aber auch teurer
* im RAM ist DRAM verbaut, ist langsamer, aber auch günstiger
* häufig benutze Daten werden im Cache gespeichert, somit muss nicht mehr auf den RAM zugegriffen werden -> Rechner läuft schneller
* es gibt verschiedene Cache-Level, werden aufsteigend immer langsamer, heute bis Level 3

## weitere Verbesserung (Hub-Architektur)

* Northbridge wird in die CPU verlagert
* RAM und Grafik können somit schneller angesprochen werden
* Southbridge wird zum Hub für alles andere Langsamere

## weitere Verbesserung (Multicore)

* wenn eine einzelne CPU nicht weiter beschleunigt werden kann (Wärme), dann werden mehrere eingebaut

## Harvard-Architektur

* aktuelle Rechner sind stark an die Harvard-Architektur angelehnt
* Daten- und Befehlsspeicher sind getrennt
* es gibt parallele Rechenwerke
* Daten und Befehle können gleichzeitig gelesen werden -> Flaschenhals verschwindet
* Zugriffs- und Speicherschutz durch Rechtevergabe umsetzbar

## weitere Verbesserung (von 16 Bit zu 64 Bit)

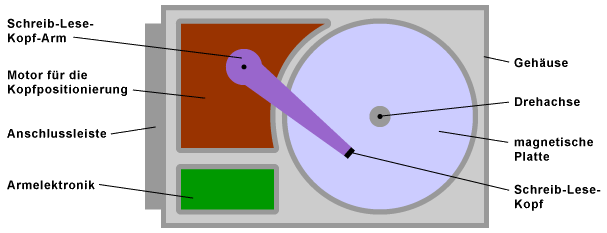
* mit der Weiterentwicklung der Prozessoren hat sich auch die Anzahl der gleichzeitig zu verarbeitenden Bits erhöht (Busbreite)
* anfangs waren es 8 bzw. 16 Bit
* um 2000 waren es 32 Bit, jedoch konnten damit nur 4 GB RAM adressiert werden, damit stieß man an die Grenzen
* heute sind es 64 Bit
* theoretischer Adressraum liegt im ExaByte Bereich, wird aber aktuell noch nicht unterstützt (max. 48 Bit Adressen)

## weitere CPU-Architekturen

* RISC-Prozessoren von PowerPC oder ARM
* RISC-Prozessoren haben einen reduzierten, sehr kleinen Befehlssatz von ca. 100 kurzen Befehlen
* bei gleicher Taktrate erzielt man so höhere Leistung und geringeren Energiebedarf
* Intel CPUs waren damals noch CISC-Prozessoren
* diese besaßen ca. 300 teilweise komplexe Befehle
* damit konnten mit einem Befehl ach schwierige Aufgaben ausgeführt werden
* dafür wurde aber auch separater Microcode und mehrere Taktzyklen für die Ausführung eines Befehls benötigt
* CISC-Prozessoren benötigen weniger RAM, weniger Register, jedoch eine höhere Taktrate
* heute nutzen x86 CPUs intern auch eine RISC-Architektur, bieten nach außen hin aber trotzdem einfache Schnittstellen

## weitere Hardware

### mechanische Festplatten



* Schreib/Lesekopf zeichnet Daten als magnetische Informationen auf
* Daten auf der Festplatte sind in Spuren und Sektoren organisiert
* diese Spuren und Sektoren werden bei der Herstellung durch eine Low-Level-Formatierung festgelegt
* lange Zeit besaßen Sektoren eine Standardgröße (Blöcke) von 512 Byte
* wegen der immer höheren Speicherdichte, wurde ab 2011 **intern** auf 4096 Byte (4K) umgestellt
* nach außen wurden weiterhin 512 Byte große Blöcke ausgegeben (512E)
* diese Emulation kostete Performance
* seit 2014 geben alle Festplatten 4K-Blöcke nativ aus, Betriebssysteme unterstützen trotzdem noch 512 Byte
* Lebenszeit
  + entweder fallen Festplatten sehr früh aus (Produktionsfehler)
  + oder erst später durch ihren Verschleiß
  + Mean Time Between Failures (MTBF) zwischen 600.000 und 2.000.000 Stunden
* SMART
  + Überwachungssystem für Festplatten, welche viele verschiedene Sensoren an der Festplatte ausliest und vorhersagen soll, wann die Festplatte ausfallen wird, ist aber nicht wirklich zuverlässig

### Schnittstellen für mechanische Festplatten

**SATA**

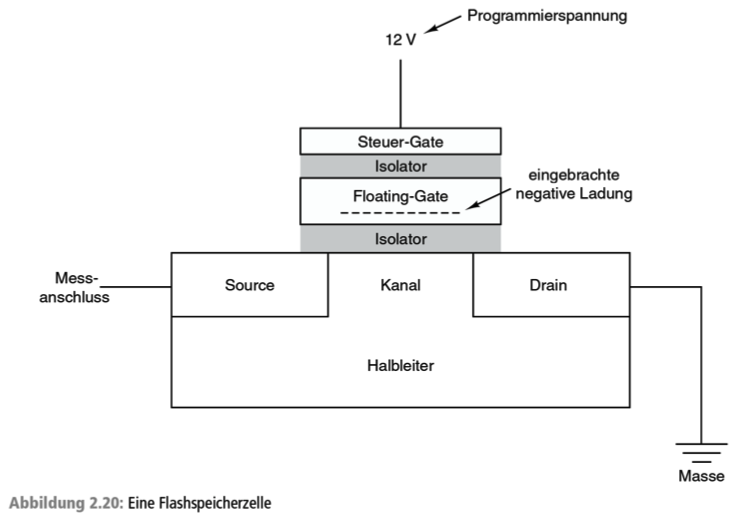
* max. 600MB/s

**SAS**

* bis zu 3GB/s
* Festplatten rotieren mit bis zu 15.000 U/min

### SSD-Festplatten

* besteht aus vielen Flachtransistoren
* es kann nur blockweise gelesen und geschrieben werden
* um Flashzelle zu programmieren legt man eine hohe Spannung an das Steuer-Gate an
* Injektion heißer Ladungsträger in das Floating-Gate wird beschleunigt
* Elektronen werden in das Floating-Gate eingebettet und erzeugen im Inneren eine negative Spannung
* die eingebettete negative Spannung erhöht sich die Spannung um den Transistor einzuschalten
* durch das Testen, ob der Transistor mit einer hohen oder niedrigen Spannung einschaltet ist (= ob das Floating-Gate geladen ist -> Flashzelle hat 0- oder 1-Wert)
* die eingebettete Ladung verbleibt auch, wenn keine Betriebsspannung mehr anliegt



* Vorteile
  + viel schneller als Festplatten
  + besitzen keine bewegbaren Teile

**Lebensdauer**

* Flashzelle lässt sich ca. 100.000 beschreiben
* durch Lesen und Schreiben der Speicherzellen, wird die Isolationsschicht des Floating-Gates zerstört -> Speicherzelle kann keine Information mehr speichern
* um Lebensdauer zu erhöhen wird Wear-Levelling betrieben
  + Schreibvorgänge werdend dabei auf alle Flashzellen im Laufwerk verteilt
  + dafür wird Speicher in logische Blöcke unterteilt
* S.M.A.R.T.-Technologie
  + ersetzt kaputte Speicherzellen durch intakte
  + ca. 10% der Kapazität sind Reserve

**Speicherzellen**

* SLC (Single Level Cell)
  + kann nur 1 Bit speichern
* MLC (Multi Level Cells)
  + kann 2/3/4 Bits speichern
  + das geschieht dadurch, weil das Floating-Gate verschiedene Ladungszustände annehmen kann
  + Speicherdichte erhöht sich, Geschwindigkeit sinkt

**Architekturen (Bauarten)**

unterscheiden sich in Speicherdichte und Zugriffsgeschwindigkeit

* NAND-Flash
  + Zellen werden in Reihe angesprochen
  + entspricht Blöcken bei Festplatten
  + wenige Datenleitungen und höhere Geschwindigkeiten
* NOR-Flash
  + jede Zelle kann einzelnen angesprochen werden
  + viele Datenleitungen und geringe Geschwindigkeit

### Tastatur

* enthält heute bis zu 108 Tasten
* PS/2 Anschluss für Tastatur und Maus ab 1987
* Tastatur und Maus



* konnten nicht im Betrieb umgesteckt werden, sonst stürzt der PC ab -> Neustart erforderlich
* heute werden USB-A Stecker verwendet, es ist im laufenden Betrieb möglich umzustecken

### Maus

* Idee dafür entstand in den 1960er Jahren
* früher gab es mechanische Mäuse mit Gummikugel -> mussten oft gereinigt werden
* heute gibt es optische Mäuse
* anfangs nur für spezielle Unterlagen, heute kann man sie auf fast jede Unterlage verwenden
* heute sind optische Funkmäuse ohne Kabelanschluss Standard

### Monitore

* früher Röhrenmonitore, diese waren sehr schwer und erzeugten Röntgenstrahlung als Abfallprodukt

# Betriebssysteme

* ein Betriebssystem (BS) umfasst verschiedene Softwarebestandteile,
* die alle physischen Komponenten eines Computers (Hardware) steuern
* und für Anwendungsprogramme und Nutzer verfügbar machen
* 2 hauptsächliche Aufgaben
  + Abstraktion der Hardware für die Anwendungen und die Interaktion mit dem Benutzer
  + die Verwaltung und Zuteilung von Hardware-Ressourcen

## Entwicklungsgeschichte

* BS gab es nicht von Anfang an, entweder wurden sie direkt in Maschinensprache oder mit Steckkarten programmiert oder sogar hart verdrahtet
* später folgten FORTRAN- und Assembler-Programme
* Mitte der 1950er Jahre wurde die Stapelverarbeitung eingeführt
  + Daten und Programme wurden zu Stapeln von Lochkarten gesammelt und dann nacheinander ausgeführt -> Zeitersparnis
  + die Stapelverarbeitung wurde von einem permanent im RAM gehaltenen Programm gesteuert -> Vorläufer des BS
* erstes BS wurde 1964 eingeführt: IBM OS/360
* es konnte mehrere Jobs parallel ausführen
* es wurde später auch eine GUI entwickelt

## Klassifizierung von Betriebssystemen

### Nach Computertyp

* Großrechnerbetriebssysteme
  + z/OS, zLinux
* Serverbetriebssysteme
  + Windows Server, Linux
* PC-Betriebssysteme
  + Windows 10, MacOS
* Tablet- und Smartphone-BS
  + iOS, Android
* Embedded Systems
  + Linux, QNX
* IoT-geräte/Sensoren
  + TinyOS, RIOT, Amazon FreeRTOS

### Nach Betriebsart

* Batch/Stapelverarbeitung
  + DOS, erste Windows Versionen
* Dialog
  + Ab Windows XP, Linux
* Echtzeit
  + VXWorks, Windows CE, QNX

### nach Anzahl gleichzeitiger Aufträge

* Single-Task
  + DOS
* Multi-Task
  + alle modernen BS

### Nach Anzahl gleichzeitiger Nutzer

* Single User
  + DOS, Android, iOS
* Multi User
  + Unix, Linux, Windows Server

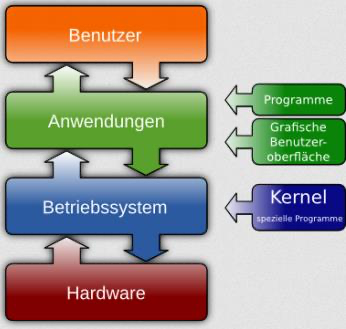
## Firmware

* manche BS greifen nicht direkt auf die Hardware zu, sondern auf die Firmware des Rechners
* was ist eine Firmware
  + interagiert selbst mit der Hardware
  + sorgt für die Umsetzung der Hardwarefunktionen
  + ist hardwarespezifisch
  + liefert bei bestimmten gerätetypen alle Funktionen
  + kann vom Nutzer aktualisiert werden
* auf Server-Mainboards oder Storage-Systemen übernimmt die Firmware auch Managementfunktionen, ist auch ohne laufendes BS über das Netzwerk erreichbar
* auf Intel-PCs ist die Firmware das BIOS
* das BIOS stellt dem BS alle Komponenten (z.B. Maus, Tastatur…) zur Verfügung
* beim Start werden alle Komponenten geprüft und ggf. initialisiert
* durch BIOS hat der Boot-Loader Zugriff auf den Massenspeicher und kann die Boot-Signatur suchen
* im BIOS werden Grundeinstellungen zur Hardware des Computers vorgenommen
* diese Einstellungen werden auch ohne Stromanschluss mittel Pufferbatterie gespeichert
* BIOS unterstützt keine 64 Bit Prozessoren
* heute werden 64 Bit Prozessoren vom UEFI unterstützt
* UEFI bietet eine grafische Benutzeroberfläche
* kann auch vom Netzwerk booten

## Aufgaben des BS

### Abstraktion der Hardware

* die Hardware bietet komplizierte und uneinheitliche Schnittstellen die nicht wirklich nutzbar sind
* Aufgabe des BS ist es, diese Schnittstellen so zu abstrahieren, dass sie konsistent, verständlich und nutzbar sind
* Beispiel
  + Anwendungen greifen zum Speichern von Daten nicht direkt auf die Hardware der Festplatte zu, sondern verwenden Systemaufrufe um Daten zu speichern
* die Abstraktion wird von mehreren Ebenen übernommen
  + Hardware
    - unterste Ebene
  + Betriebssystem
    - kommuniziert direkt oder indirekt (über BIOS und Treiber) mit der Hardware
  + Anwendungen
    - Benutzeroberfläche und Anwendungsprogramme
  + Benutzer
    - oberste Ebene



### Verwaltung der Hardware

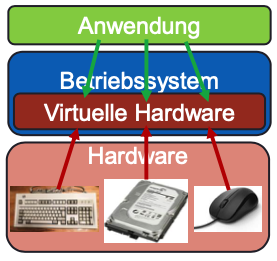
* Kernel
  + steuert und verwaltet Hardware, laufende Prozesse, Ressourcen, Anwendungen und Daten
  + lädt alle Gerätetreiber, steuert Speicherzugriffe, Systemsicherheit, I/O-Aufgaben
  + startet Systemdienste, die im Hintergrund laufen
    - z.B. DHCP, DNS, Druckwarteschlange…
  + hat volle Nutzungsrechte für die Hardware -> läuft im Kernel-Modus
* Benutzeroberfläche
  + grafisch oder als Kommandozeile (Shell) zur Interaktion mit dem Benutzer
  + Benutzer-Modus hat deutlich weniger Rechte um Computer und Daten zu schützen
* Dateiverwaltung
  + erstellt und verwaltet Massenspeicher
  + bietet Zugriff auf Ordner und Dateien
* Hilfsprogramme
  + Bieten viele Grundfunktionen wie Kopieren, Organisieren, Suchen…
* Boot-Loader
  + startet den Computer
  + lädt das BS
  + analysiert angeschlossenen Massenspeicher und sucht nach einer Boot-Signatur (diese steht im MBR (Master-Boot-Record) oder einer speziellen Partition)
  + liest vom Massenspeicher die Kernel-Dateien und lädt sie in den RAM
  + übergibt Systemverantwortung an Kernel

## Konzepte

* Virtualisierung der konkurrierend genutzten Systemressourcen
* Adressräume im RAM
* Prozesse als Modell für gleichzeitig laufende Verarbeitungsvorgänge
* Dateien
* Treiber
* Maßnahmen für Datenschutz und Datensicherheit

### Virtualisierung

* ist die Entkoppelung einer nutzbaren Systemressource von der physisch existierenden Hardware
* Anwendungen nutzen nicht direkt die Hardware, sondern deren logische Repräsentation im BS



* Anwendung der Virtualisierung zum Beispiel beim RAM
  + um Anwendungen genügend RAM zur Verfügung stellen zu können, wird Virtual Memory Management (VMM) betrieben
  + dieser setzt sich aus dem physikalischen RAM und einer auf dem Datenträger ausgelagerten Datei zusammen (pagefile.sys unter Windows)
  + Vorgang nennt sich Paging

### Prozesse

* ein Prozess ist eine Ausführung eines Programms, ein Programm zur Laufzeit
* das Modell des Prozesses ermöglicht quasi-paralleles Bearbeiten mehrerer Programme in nur einem Prozessor
* beim Anhalten eines Prozesses, werden alle nötigen Informationen zur Wiederaufnahme gespeichert
* das BS führt eine Liste über alle Prozesse und deren Status
* Prozess hat folgende Bestandteile
  + Adressraum im RAM
    - Bereich im RAM, den der Prozess nutzen darf und der gegenüber anderen Prozessen abgeschirmt wird
  + einen Process Identifier (PID)
  + Angaben zum Benutzer, der ihn ausführt (UID)
  + evtl. Kinderprozesse
    - von einem Prozess erzeugte weitere Prozesse
  + evtl. Threads
    - parallele Verarbeitungsstränge innerhalb eines Prozesses

### Dateien

* ist eine Abstraktion des BS für Speicherplatz auf einem nichtflüchtigen Speicher wie der Festplatte
* BS verwenden verschiedene Dateisysteme
* Dateisysteme stellen ein Verzeichnis dar, in dem die Dateien verschoben, gespeichert und geändert werden können
* Verzeichnisse beginnen mit einem Root
* es gibt 2 Pfadtypen
  + Absoluter Pfad
    - gibt den Weg vom Stammverzeichnis (Root) bis zur Datei an
  + Relativer Pfad
    - geht von einem Verzeichnis unterhalb der Root aus und gibt den Weg von dort bis zur Datei an
    - kann nur genutzt werden, wenn der Weg zur Datei nicht über eine höhere Verzeichnisebene führt

### Treiber für Ein- und Ausgabegeräte

* ist eine Software, die gerätespezifische Funktionen und Daten so zur Verfügung stellt, dass sie von BS genutzt werden können
* sind speziell auf ein Gerät oder eine Gerätegruppe zugeschnitten
* arbeiten als Teil des Kernels und besitzen hohe Rechte

### Maßnahmen für Datenschutz und Datensicherheit

* verschiedene Rechteebenen (Kernelmodus und Benutzermodus)
  + um Funktionen im Kernelmodus auszuführen führen Anwendungen Systemaufrufe durch (Syscall)
  + BS prüft, ob die Berechtigung dafür vorliegt, führt den Auftrag aus und schaltet wieder zurück in den Benutzermodus
* Benutzer müssen sich am Gerät authentifizieren
* Anwendungen und Dateisysteme werden mit dedizierten Zugriffsrechten geschützt
  + es gibt verschiedene Zugriffsrechte, wie Lese, Schreiben, Ausführen (r, w, x)
  + die Rechte werden an einer Datei gesetzt
  + Benutzer oder Gruppen können verschiedene Rechte zugeteilt werden
  + Zugriffsrechte werden an untergeordnete Objekte vererbt
* Verschlüsselung
* auf zweiter Ebene
  + Anti-Malware-Schutz
  + Firewall
    - erlaubt oder verweigert ein- und ausgehenden Netzwerkverkehr
    - man kann manuell Regeln für den Zugriff konfigurieren
  + Patch-System
    - regelmäßige Updates

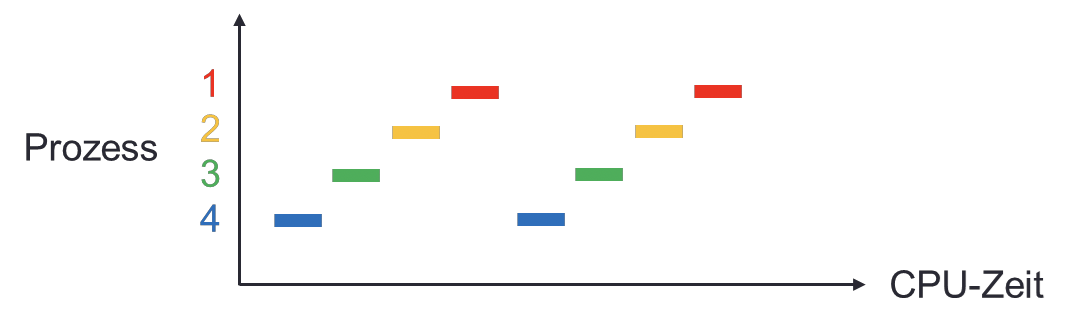
# Aufgaben des BS im Detail

## Prozesse und Threads

* ein Prozess (Task) ist eine laufende Instanz eines Programmes mit allen zur Ausführung benötigten Daten, Befehlen und sonstigen Informationen
* ein Prozess kann als eine Umgebung für eine Aktivität verstanden werden, der vom BS CPU-Zeit zur Ausführung zugeteilt wird

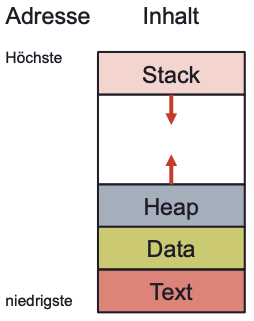
### Prozessmodell

* Ziel ist das „gefühlte“ gleichzeitige Ausführen von Programmen (Multitasking)
* ausführbare Software wird als Menge von sequenziellen Prozessen verstanden
* Scheduler verteilt die CPU-Zeit auf die Prozesse
* Scheduler nutzt dafür eine Prozesstabelle mit allen Metadaten zu den Prozessen
* Prozessen wird CPU-Zeit zugeteilt -> Prozesse werden abwechseln ausgeführt
* beim Wechsel zwischen Prozessen werden die Prozessbestandteile gesichert



### Prozessbestandteile

* Prozesse besitzen ihren privaten Adressraum im RAM mit folgenden Inhalten
  + Programmcode
  + Daten
  + Heap
  + Stack

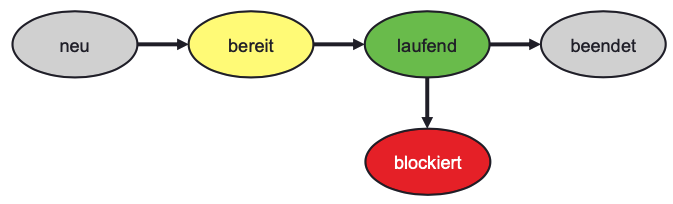


* weitere Prozessbestandteile
  + Wert des Befehlszählers
  + Inhalt des Befehlsregisters
  + Inhalte der Prozesstabelle

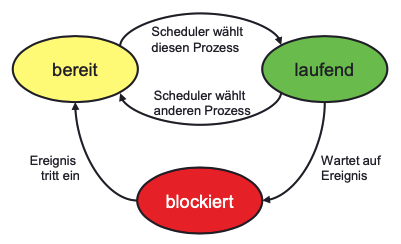
### Prozesserzeugung

* Prozesse werden ausfolgenden Anlässen erzeugt
  + Systemstart
  + Anforderung durch einen anderen Prozess
  + Anforderung durch einen Benutzer
* wird technisch durch einen Systemaufruf erzeugt
  + z.B. unter Windows mit CreateProcess
* Prozesse ohne Benutzerinteraktion
  + werden automatisch gestartet und laufen ohne, dass es der Benutzer merkt (Dienste)

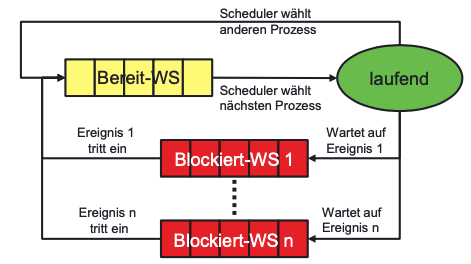
### Prozesszustände



* neu
  + ein Prozess ist fertig vorbereitet und wartet auf Zulassung
* bereit
  + ein Prozess ist zur Ausführung bereit
* laufend
  + ein Prozess wird von der CPU ausgeführt
* blockiert
  + ein Prozess wartet auf ein Ergebnis
* beendet
  + ein Prozess ist beendet
* Prozessübergänge werden vom Scheduler gesteuert



* normalerweise sind mehrere Prozesse gleichzeitig aktiv, der Scheduler arbeitet deshalb mit Warteschlangen für jeden Zustand



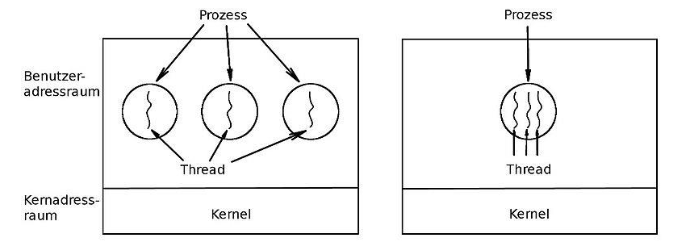
* Zustand eines Prozesses kann sich ausfolgenden Gründen ändern
  + Scheduler aktiviert nach Ablauf der CPU-Zeit einen anderen Prozess
  + das Programm ruft explizit einen anderen Prozess auf (Supervisor Call)
  + ein Fehler beendet den Prozess (Trap)
  + Interrupt geht ein, Prozess muss beendet werden, um dem Interrupt Handler die CPU zur Verfügung zustellen

### Prozesse beenden

* Gründe für das Beenden eines Prozesses
  + normales Beenden (freiwillig)
  + Beenden nach einem Fehler (freiwillig)
  + Beenden nach einem schwerwiegenden Fehler (unfreiwillig)
  + Beenden durch einen anderen Prozess (unfreiwillig)
* unter Windows mit exitProcess
* Anhalten durch den Scheduler ist **kein** beenden des Prozesses

### Threads

* Threads bieten die Möglichkeit, mehrere Programme auf einer CPU quasi gleichzeitig auszuführen
* jeder Prozess hat einen privaten RAM-Bereich, andere Prozesse können auf diesen nicht zugreifen
* über eine Pipe können Prozesse miteinander kommunizieren
* Threads können gleichzeitig mehrere Operationen auf denselben Daten ausführen
* mehrere Threads eines Prozesses teilen sich den privaten Adressraum, globale Variablen und Dateien
* Threads besitzen auch eigene Informationen
  + Stack
  + Befehlszähler
  + Register
  + Zustand



### Threadebenen

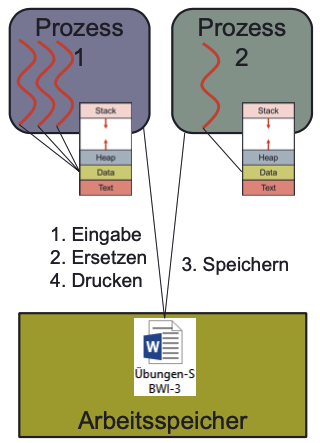
* User-Level Threads (ULT)
  + werden direkt vom Prozess verwaltet, der eine Thread Library besitzt
  + sind für den Kernel unsichtbar
  + sind einfacher zu handhaben
  + Nachteil: blockiert ein Thread -> blockiert auch der gesamte Prozess
* Kernel-Level Threads (KLT)
  + können nicht mit Thread Library verwaltet werden, Kernel muss diese Aufgabe übernehmen
  + Verwaltungsaufwand deutlich „teurer“

### Vergleich mit Prozessen

* Threads werden auch als leichtgewichtige Prozesse bezeichnet
* es können gleichzeitig mehre Threads ausgeführt werden (Multithreading)
* Threads besitzen die Zustände: bereit, blockiert, laufend
* Threads müssen untereinander synchronisiert werden, damit Fehler vermieden werden
* wird ein Prozess beendet, werden auch alle seine Threads beendet

### Interprozesskommunikation und Scheduling

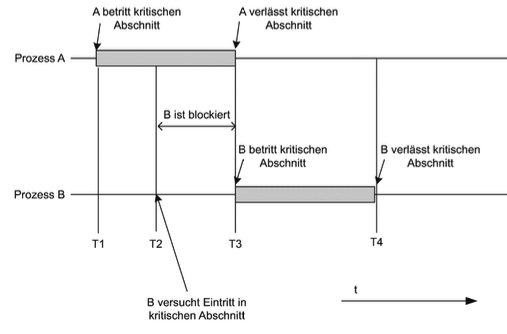
* Threads greifen auf den gemeinsamen Speicher ihres Prozesses zu
* Prozesse können sich Daten mit anderen Prozessen über einen externen Speicherbereich austauschen
* bei gemeinsam genutzten Daten kann die Abarbeitung der Prozesse das Ergebnis beeinflussen (siehe Beispiel)
* solche Zustände werden als Race Conditions (zeitkritische Abläufe) bezeichnet -> diese müssen vermieden werden
* Race Conditions treten unter folgenden Bedingungen auf
  + Threads werden ausgeführt
  + diese Threads nutzen gemeinsame Daten



* Race Conditions können verhindert werden, indem sie sich gegenseitig ausschließen (mutual exclusion)
  + wenn ein Prozess auf Daten zugreift, das kein anderer Prozess auf diese Daten zugreifen, solange der Prozess noch auf die Daten zugreift -> das reicht noch nicht aus für eine gute Prozessorsteuerung
* Bedingungen für effiziente Zusammenarbeit
  + keine zwei Prozesse dürfen sich gleichzeitig in einem kritischen Abschnitt befinden
  + es darf keine Annahme über Geschwindigkeit und Anzahl der CPUs gemacht werden
  + kein Prozess, der selbst nicht in einem kritischen Abschnitt ist, darf einen anderen Prozess blockieren
  + kein Prozess soll unendlich lange warten müssen, bis er in seinen kritischen Abschnitt eintreten darf
* diese Bedingungen können durch verschiedene Methoden erfüllt werden
  + Interprozesskommunikation (IPC) mit den Optionen
    - Synchronisierung von Prozessen
    - Kommunikation zwischen Prozessen
  + verschiedene Strategien für das Scheduling

#### Synchronisation von Prozessen

* während sich Prozesse in einem kritischen Abschnitt befinden, müssen sie sich gegenseitig ausschließen



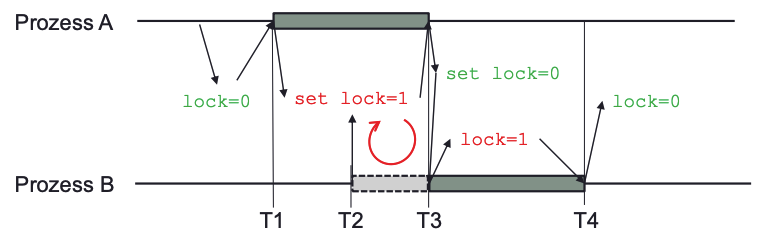
* damit die Prozesse sich gegenseitig ausschließen können, gibt es verschiedene Ansätze
  + vorübergehend alle Interrupts abschalten
  + Prozesse versuchen selbst ihren zeitlichen Ablauf abzustimmen, signalisieren z.B. mit Sperrvariablen und aktivem Warten oder TSL-Befehl
  + aktive Einbeziehung des BS: Semaphore

**Alle Interrupts abschalten**

* wenn ein Prozess fehlerhaft ist, und er seine Arbeit nicht beenden konnte, wird der Interrupts evtl. nie wieder ausgeschaltet und der Scheduler wäre blockiert -> kein Multitasking mehr möglich

**Sperrvariablen verwenden**

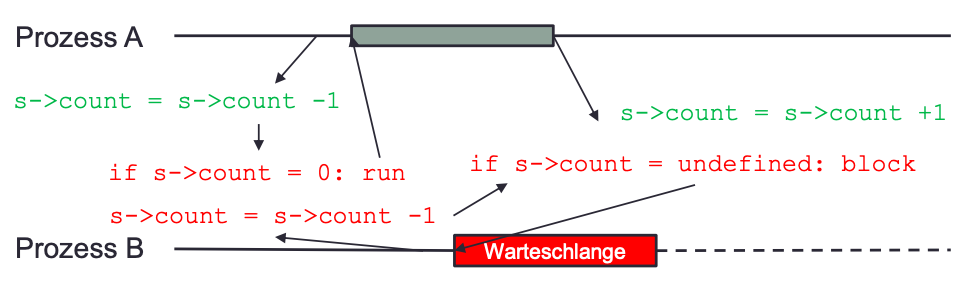
* gemeinsame Variable für alle Prozesse: int lock;
* Prozesse führen Test, Set und Lock aus



* Probleme
  + auch während die Operationen Test, Set und Lock ausgeführt werden, kann ein Prozess vom Scheduler abgeschalten und die Werte der Variablen überschrieben werden
  + ein Prozess in der Warteschlange verschwendet CPU-Zeit

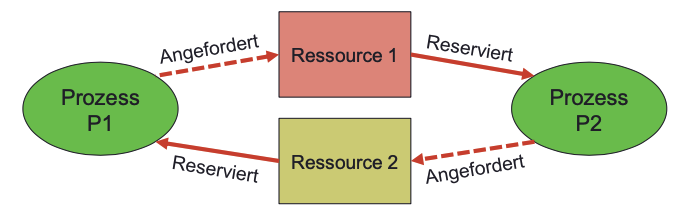
**Semaphore**

* Semaphore bedeutet Signalgeber
* Prozesse müssen sich nicht mehr selbst prüfen und benachrichtigen -> dies übernimmt z.B. der Scheduler
* Prozesse müssen nicht mehr aktiv warten, sondern blockieren in dieser Zeit
* Semaphore ist eine Datenstruktur, die einen Zähler und eine Warteschlange enthält
* der Zähler kann von allen Prozessen gelesen werden
* Datenstruktur darf nur durch P(down) und V(up) atomar (ununterbrechbar) verändert werden
* für den gegenseitigen Ausschluss von Prozessen wird ein binärer Semaphor verwendet -> es werden nur die Zustände 0 und 1 ausgewertet
* Binäre Semaphore oder Mutex-Locks (mutual exclusion) funktionieren wie folgt
  + ein Prozess, der in seinen kritischen Abschnitt eintreten möchte, dekrementiert den Zähler count um 1 -> P(down)
  + ist der Zähler jetzt bei >= 0, setzt der Prozess seine Arbeit fort
  + durch reduzieren auf 0, wird anderen Prozessen signalisiert, dass die gemeinsam genutzte Ressource gesperrt ist (Mutex Lock)
  + ist der Zähler bereits 0, wird die Prozess blockiert und in die Warteschlange eingereiht
  + wenn der Prozess den kritischen Abschnitt verlässt, wird die Zähler wieder um 1 erhöht -> V(up) und damit die gemeinsame Ressource wieder freigegeben



* Worin ist der Unterschied zu einer Sperrvariable?
  + down und up wird nicht von den Prozessen selbst ausgeführt, sondern als Systemaufruf vom BS
  + BS stellt sicher, dass diese Aufrufe atomar sind (nicht unterbrochen werden können)
* es gibt mehrere Arten von Semaphoren
  + Zählsemaphoren (Initialwert = N)
    - wenn mehrere Ressourcen zur Verfügung stehen
  + Binärsemaphoren (Initialwert = 1)
    - Lösung des Wechselseitigen Ausschlusses
  + Schlafen/Wecken (Initialwert = 0)
    - Aufrufender Prozess wird blockiert, bis er durch einen anderen Prozess wieder geweckt wird

### Deadlock



* ein Deadlock entsteht unter folgenden Bedingungen
  + Mutual Exclusion Condition: eine Ressource steht einem Prozess nur exklusiv zur Verfügung
  + Wait for Condition (Hold and Wait): Prozesse warten bis sie alle nötigen Ressourcen haben und behalten dabei die Kontrolle über Ressourcen, die ihnen bereits zugewiesen sind
  + No Preemption Condition: keinem Prozess kann eine Ressource gewaltsam entzogen werden
  + Circular Wait Condition: es existiert eine zirkulare Abhängigkeit zwischen den Prozessen (P1 fordert das an, was P2 hat -> P2 fordert das an, was P1 hat) siehe Beispiel oben

### Behandlung von Deadlocks

* Deadlocks verhindern (Prevention)
  + indirekt durch Verhindern mindestens einer der Bedingungen 1-3
  + direkt durch Verhindern des Circular Wait
* Deadlocks vermeiden (Avoidance)
  + das BS wird so konstruiert, das Deadlocks nicht auftreten können
* Deadlocks erkennen und beheben
  + wenn Deadlocks auftreten, werden Maßnahmen getroffen um sie zu beseitigen
* Deadlocks werden ignoriert
  + verhindern oder vermeiden von Deadlocks ist sehr aufwendig, treten sie nur sehr selten auf, nimmt man dies hin, durch einen Reboot kann der Deadlock beseitigt werden

#### Deadlocks verhindern (Prevention)

**Mutual Exclusion Condition auflösen**

* es gibt exklusiv nutzbare Ressourcen, die nur von einem Prozess zu Zeit genutzt werden können, z.B. CPU, Drucker, Festplatte…
* gegenseitiger Ausschluss wird durch Semaphore bewirkt
* -> diese Bedingung kann nicht aufgehoben werden

**Wait for Condition (Hold and Wait) auflösen**

* das BS könnte von allen Prozessen verlangen, dass sie alle benötigten Ressourcen nur auf einmal reservieren dürfen
* -> ist aber problematisch, weil Ressourcen somit sehr lange ungenutzt brach liegen würden
* anderen Prozesse müssten dann ewig auf häufig genutzte Ressourcen warten oder würden verhungern
* -> kann praktisch nicht umgesetzt werden

**No Preemption Condition auflösen**

* Prozesse müssen bereits reservierte Ressourcen wieder freigeben, wenn sie eine weitere Ressource benötigen, diese aber nicht bekommen, später dürfen sie die Ressourcen erneut anfordern
* -> kann nur teilweise umgesetzt werden für Ressourcen deren Zustand sich leicht sichern lässt, z.B. Register, für Drucker aber nicht

**Circular Wait Condition auflösen**

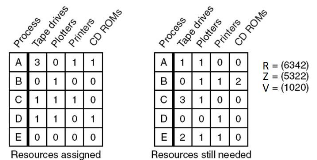
* strikte lineare Ordnung festlegen
* alle Ressourcen durchnummerieren und eine Liste erstellen: R1, R2, R3, Rn
* dabei gilt R1<R2<R3<Rn
* ein Prozess, der Ri besitzt, darf nur Rj zugeteilt bekommen, wenn Rj>Ri ist
* besitzt ein Prozess bereits eine Ressource Rk mit Rk>Rj, bekommt er Rj nur, nachdem er Rk abgegeben hat
* nach Rj muss dann Rk erneut reserviert werden
* -> funktioniert und ist umsetzbar, führt aber möglicherweise zu schlechter Systemperformance -> ein Prozess bekommt eine Ressource nur der Ordnung halber nicht und muss warten

#### Deadlocks vermeiden (Avoidance)

* System prüft, ob das Starten weiterer paralleler Prozesse zu einen Deadlock führen würde
* nur wenn das nicht der Fall ist, werden weitere Ressourcen vergeben
* Voraussetzung
  + Anzahl der Prozesse sowie verfügbarer und maximal benötigter Ressourcen muss bekannt sein
* es werden 3 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 P1-Pn, die jedem Prozess auch mit maximal benötigen Ressourcen den erfolgreichen Abschluss ermöglicht
* wenn es mehrere Ressourcentypen gibt, muss diese Bedingung für jeden Ressourcentyp erfüllt sein
* Prozesse werden nur gestartet, wenn der Unsafe State nicht erreicht wird
* alle Deadlocks sind Unsafe, aber Unsafe ist nicht automatisch ein Deadlock



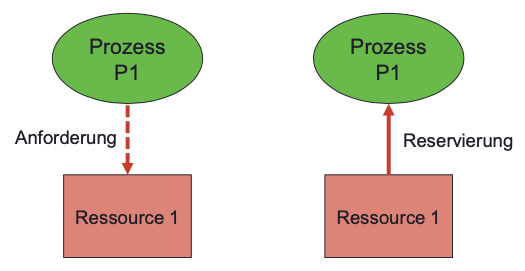
* für die Prüfung wird der Bankier-Algorithmus verwendet
* dieser benötigt folgende Informationen
  + R als Vektor der insgesamt vorhandenen Ressourcen jedes Typs
  + Z als Vektor der aktuell zugewiesenen Ressourcen jedes Typs
  + V als Vektor der noch verfügbaren Ressourcen jedes Typs

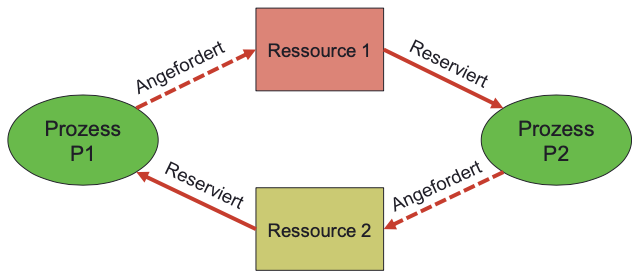


* Vorgehensweise des Algorithmus
  + suche einen Prozess, dessen maximaler Ressourcenbedarf <= V ist
  + weise diesem Prozess alle benötigten Ressourcen zu und lasse ihn seine Arbeit beenden, markiere den Prozess als Beendet und füge die freien Ressourcen zu V hinzu
  + wiederhole Schritt 1 und 2 für alle noch verbliebenen Prozesse, bis alle Prozesse als beendet markiert sind
  + ist das möglich, dann bleibt das System im Zustand Safe, andernfalls geht es in Unsafe
* der Bankier-Algorithmus ist theoretisch funktionsfähig, wird aber trotzdem nur selten genutzt
* Gründe
  + kaum ein Prozess weiß im Vorhinein, welche Ressource 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

#### Deadlocks erkennen und beheben

* BS könne Deadlocks zulassen, müssen dann aber Maßnahmen zur Erkennung und Behebung treffen
* Erkennung funktioniert unterschiedlich, je nachdem, ob es eine Ressource nur einmal oder mehrmals gibt
  + wenn es die Ressource nur 1x gibt, z.B. CPU
    - ein Ressourcenbelegungsplan kann Deadlocks anzeigen
    - wenn es einen Zyklus gibt, dann liegt ein Deadlock vor





* + wenn es die Ressource mehrmals gibt, z.B. Speicher
    - Bankier-Algorithmus kann verwendet werden -> suche einen Prozess, der mit den aktuell **nicht** zugewiesenen Ressourcen beendet werden kann
    - beende diesen Prozess und gebe seine Ressourcen wieder frei
    - ist dies für alle Prozesse möglich, gibt es keinen Deadlock
    - bleiben mehr ein oder mehrere Prozesse übrig, besteht ein Deadlock
* wurde ein Deadlock erkennt, gibt es verschiedene Methoden diesen aufzulösen
  + abbrechen aller beteiligten Prozesse
  + abbrechen einzelner beteiligter Prozesse, wenn nötig schrittweise
  + Rollback (sofern es eine Form des Protokolls gibt)
  + zeitweiliges Entziehen einer Ressource eines Prozesses (preemtion)
* Frage ist: Welcher Prozess wird das Opfer?
* meist Prozesse, die ohne gravierende Nebenwirkungen neugestartet werden können, z.B. Programmstarts, Websitenaufrufe
* keine komplexen Prozesse, wie z.B. Printjobs

### Scheduling

* für effiziente Zusammenarbeit von Prozesse gibt es folgende Bedingungen
  + keine 2 Prozesse dürfen gleichzeitig in ihrem kritischen Abschnitt sein
  + es darf keine Annahme über Geschwindigkeit und Anzahl der CPUs gemacht werden
  + kein Prozess, der selbst nicht in einem kritischen Abschnitt ist, darf einen anderen Prozess blockieren
* -> es ist Aufgabe des Schedulers, diese Bedingungen umzusetzen
* kann auf verschiedene Weisen umgesetzt werden
  + per Interprozesskommunikation (IPC) -> wurde vorher schon beschrieben
    - Synchronisierung
    - Kommunikation zwischen Prozessen
    - Deadlock-Behandlung
  + verschiedene Instanzen, Regeln und Strategien für das Scheduling

#### Scheduling-Instanzen im System

* Long-term Scheduler
  + entscheiden, welche neuen Prozesse sie zum Eintritt in die Bereit-Warteschlange zulassen
  + dabei wird zwischen CPU- und IO-lastigen Prozessen unterschiegen
  + Aufgabe ist es, eine gute Balance zwischen beiden Prozesstypen zu finden und die Prozesse entsprechen zuzulassen
* Medium-term Scheduler
  + kümmern sich darum, welche Prozesse sich vorübergehend aus dem Ram entfernen lassen und in die langsamen Teile des virtuellen Speichers übertragen werden können
  + sie übernehmen das Swapping und Paging
* Short-term Scheduler
  + weisen Prozesse aus der Bereit-Warteschlange die Prozessorzeit zu

#### Scheduling-Strategien

* 2 Möglichkeiten
  + nicht unterbrechend
    - ein einmal gestarteter Prozess läuft so lange bis er beendet oder blockiert ist
    - Scheduler greift nicht ein
    - blockierte oder beendete Prozesse werden sofort durch neue ersetzt
  + unterbrechend
    - Scheduler unterbricht laufende Prozesse nach bestimmten Regeln, um die Durchlaufzeit beispielsweise zu erhöhen
* Scheduling hat je nach Art des BS verschiedene Schwerpunkte
  + alle BS
    - Fairness: jeder Prozess ist mal dran
    - Strategien werden durchgesetzt
    - Balance: alle Systemkomponenten sind ausgelastet
  + Stapelverarbeitungssysteme
    - Durchsatz: so viele Jobs wie möglich pro Stunde (z.B. Gehaltsabrechnungen)
    - minimale Durchlaufzeit für jeden Job
    - maximale CPU-Auslastung
  + Interaktive Systeme
    - schnelle Antwort auf Eingaben
    - Proportionalität: Erwartungen des Benutzers erfüllen
  + Echtzeitsysteme
    - Deadlines einhalten: Datenverlust vermeiden
    - Vorhersagbarkeit: keine Qualitätseinbußen beim Streaming
* Strategien in Stapelverarbeitungssystemen
  + First Come First Served (FCFS)
  + Shortest Job First
  + Shortest Remaining Time Next
* Strategien in interaktiven BS
  + Round-Robin-Scheduling
  + Priority-Scheduling
  + Fair-Share-Scheduling

**First Come First Served (FCFS)**

* ist **nicht** unterbrechbar
* 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 andauernde Prozesse bremsen Kurze aus

**Shortest Job First (SJF)**

* ist **nicht** unterbrechbar
* sortiert 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



* Durchlaufzeit kann optimiert werden
* Voraussetzungen
  + alle Prozesse sind gleichzeitig verfügbar
  + die Durchlaufzeit jedes Prozesses ist vorher bekannt

**Shortest Remaining Time Next**

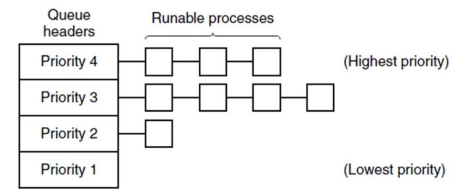
* ist ein unterbrechbarere SJF
* jeder neue Prozess wird an der verbleibenden Laufzeit des aktuellen Prozesses gemessen
* ist der neue Prozess kürzer, als die Restlaufzeit des aktuellen Prozesses, wird dieser unterbrochen und der neue zuerst abgearbeitet
* bevorzugt neue kurze Prozesse

**Round-Robin-Scheduling**

* teilt jedem Prozess ein festes Zeitfenster zu
* ausgewogenes Verhältnis bezüglich Durchsatzes und Durchlaufzeit
* erzeugt relativ hohen Overhead
* berücksichtigt nicht die Bedeutung von Prozessen

**Priority-Scheduling**

* vergibt Prioritäten an Prozesse
* weist den Prozessen entsprechend ihrer Priorität mehr oder weniger Ressourcen zu
* Problem: Prozesse mit niedrigen Prioritäten würden verhungern
* Lösung
  + dynamische Prioritäten
    - interaktive Prozesse bekommen eine hohe Priorität
    - Hintergrundprozesse bekommen eine niedrigere Priorität
    - bekommt ein Prozess erstmals eine Ressource zugewiesen, wird er bearbeitet und anschließend seine Priorität herabgesetzt
    - letztlich „überholen“ Prozesse mit niedrigeren Prioritäten irgendwann Prozesse, die ursprünglich höher priorisiert waren
  + Kombination von Prioritätsklassen und Round-Robin in einer Klasse
    - es können mehrere Warteschlangen verwaltet werden (Multilevel Queue Scheduling)
    - Scheduling findet dann nur zwischen den Klassen statt, innerhalb einer Klasse wird z.B. Round-Robin eingesetzt
    - Vorgehensweise: lasse erst alle Prozesse der höchsten Priorität per Round-Robin laufen, bis die Warteschlange leer ist, arbeite danach die anderen Warteschlangen ab
    - Prozesse wechseln nicht zwischen Klassen
    - die Prioritäten müssen dynamisch angepasst werden, damit kein Prozess verhungert



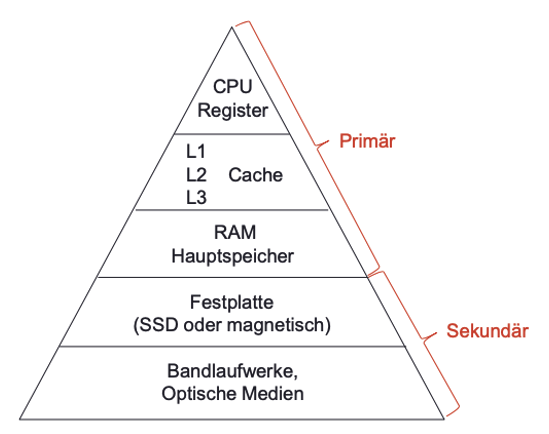
* + um noch flexibler und fairer zu sein, kann das Multilevel Queue Scheduling um eine Bewertung des vorherigen Verhaltens eines Prozesses erweitert werden
    - Scheduling verschiebt Prozesse, nach z.B. CPU-Zeit oder IO-Aktivität zwischen den Prioritätsklassen
    - Prozesse, die zu viel CPU-Zeit nutzen, werden herabgestuft, lang wartende Prozesse werden heraufgestuft
    - -> diese Strategie heißt Multilevel Feedback Queue Scheduling
    - Windows und MacOS arbeiten damit

**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 ihn nur dann zu, wenn dieser Besitzer seinen Anteil noch nicht mit anderen Prozessen ausgeschöpft hat
* Linux ab 2.6.3 nutzt Completely Fair Scheduling

## Speicherverwaltung

* Computer besitzen mehrere Speicherebenen mit verschiedenen Eigenschaften
  + Primärer Speicher
    - schnell (SRAM), klein bis mittel, preiswert bis teuer
    - kann direkt von CPU angesprochen werden
    - flüchtig
  + Sekundärer Speicher
    - langsam (DRAM), groß bis sehr groß, preiswert
    - kann nur über einen Controller angesprochen werden
    - nicht flüchtig



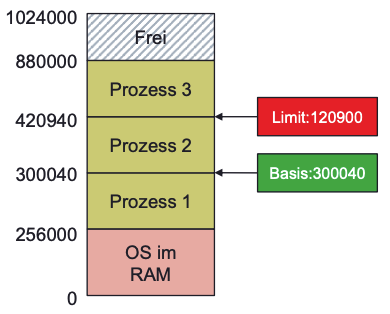
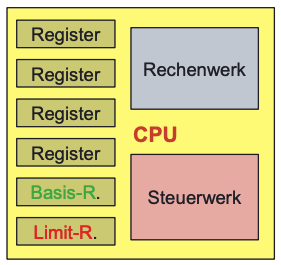
* Ziele der Speicherverwaltung
  + Verhinderung, dass Prozesse Speicherbereiche des Kernels verwenden
  + Verhinderung, dass Prozesse auf Speicherbereiche von anderen Prozessen zugreifen
  + alle laufenden Prozesse sollen genügend Speicher zur Verfügung haben
    - ohne Verschwendung von Speicher
    - unabhängig vom tatsächlich physikalisch verfügbaren RAM
* Methoden der Speicherverwaltung
  + Basis- und Limit-Register (nur bei 8/16 Bit Prozessoren)
  + Segmentierung
  + Swapping
  + Virtueller Speicher mit Paging
  + -> alle diese Methoden abstrahieren den physikalischen Speicher

### Monoprogrammierung und physikalischer Speicher

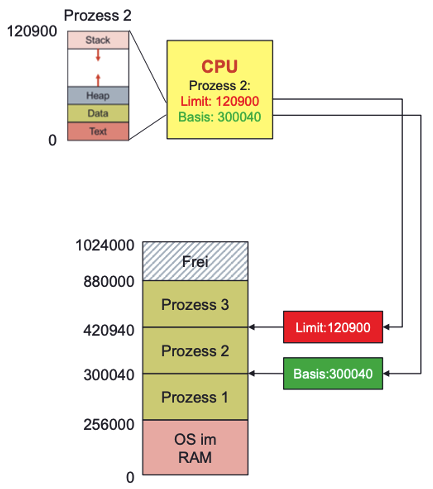
* manche Handhelds, eingebettete Systeme und MS-DOS führten nur einen Prozess gleichzeitig aus und abstrahierten den Speicher nicht
* Speicherabstraktion
  + jeder Prozess bekommt seinen eigenen privaten Adressraum
  + beginnt bei 0, ist zusammenhängend und endet bei einer höchsten Adresse
* Vorteil
  + der Prozess arbeitet intern mit logischen und innerhalb des Prozesses eindeutigen Adressen, egal auf welcher Maschine der Prozess läuft
* Nachteil
  + mehrere parallele Prozesse verwenden für sich die gleichen logischen Adressen
* die Speicherabstraktion muss dafür sorgen, dass die gleichen logischen Adressen auf verschiedene physikalische Adressen im Ram abgebildet werden

### Basis- und Limit-Register

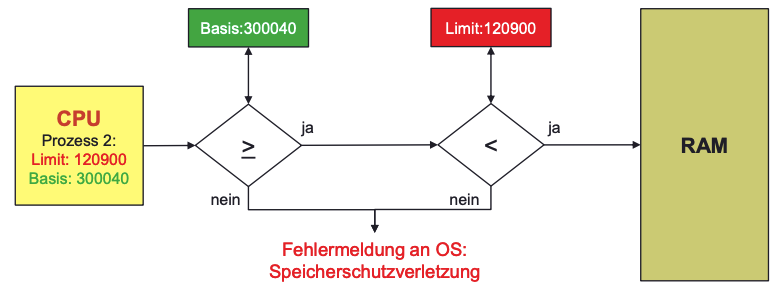
* es wurden 2 zusätzliche Register eingeführt
  + Basis-Register und Limit-Register
* dies ermöglicht die Nebenläufigkeit im physikalischen Speicher
* das Basis-Register enthält einen individuellen Adress-Offset für den Prozess, der zu jeder Speicheranforderung hinzuaddiert wird
* das Limit-Register enthält einen relativen maximalen Adresswert für jeden Prozess oberhalb von dessen Basisadresse



* bei jeder Anforderung eines Prozesses hebt die CPU die geforderte Adresse in das richtige Speichersegment für den angeforderten Prozess



* diese Segmentierung des physikalischen Speichers wurde zusätzlich mit einem Speicherschutz versehen
* vor jedem Speicherzugriff eines Prozesses prüft die CPU vorher
* nur wenn die vom Prozess angeforderten Adressen sich nach Addition des Basisregisterwertes noch im Speicherbereich des Prozesses befindet (Limit), wird die Anforderung ausgeführt
* sonst wird eine Fehlermeldung ausgegeben

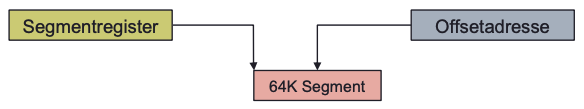


### Speicherabstraktion in x86-Prozessoren

* diese Prozessoren verwendeten erst nur physikalischen Speicher und adressierten diesen im Real Mode
* der Real Mode adressiert eine Segmentierung mit festen Abständen
* mit 20 Bit Adressbusbreiten konnte 1 MB adressiert werden
* die einzelne Adresse wurde in 2 Teilen angegeben
  + Segmentadresse
  + Offsetadresse
* Segmentadresse wird in einem 16 Bit-Register gehalten und zeigt auf den Beginn eines Speichersegments
* von den 16 Bit im Register haben aber nur die 12-höchstwertigen Bits eine Bedeutung, die anderen sind 0 und werden ignoriert (entspricht einer Multiplikation mit 16)



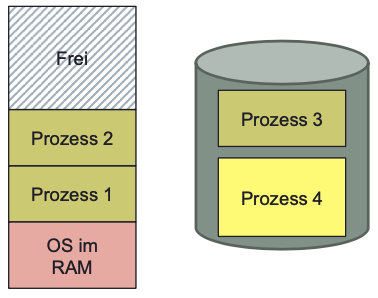
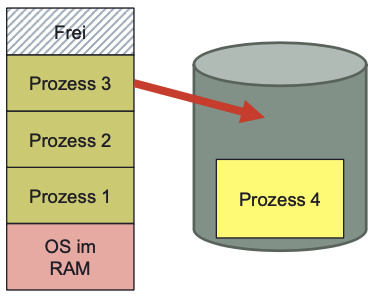
* die Offsetadresse zeigt auf eine Adresse innerhalb des jeweiligen Segments



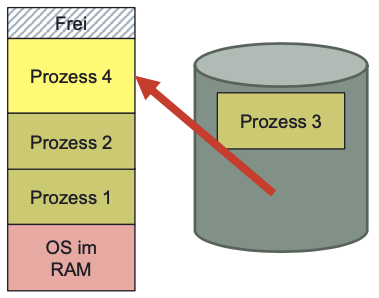
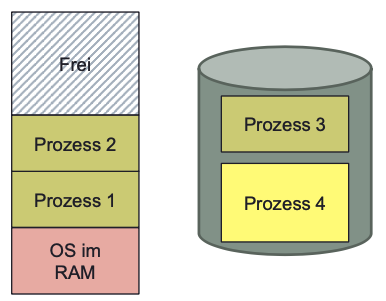
* die gesamte Speicheradresse wird aus den ersten 12 Bits des Segmentzeigers und der 8 Bit langen Offsetadresse gebildet
* damit ergeben sich Segmente mit je 64K Größe und 20 Bit Adressen
* jede 32-Bit x86 CPU startet im Real Mode
* im Real Mode bestand kein Speicherschutz
* später kam der Protected Mode, dieser bot Speicherschutz, konnte 4 GB RAM unterstützen und virtuellen Speicher verwalten

### Swapping

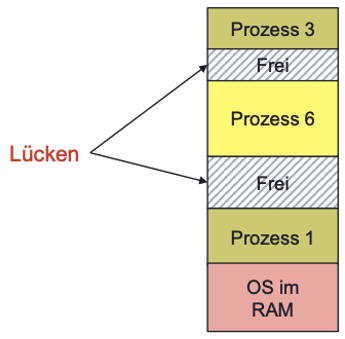
* Basis- und Limit-Register erlaubt kein Auslagern von Prozessen -> ist der physikalische Ram voll, können keine weiteren Programme ausgeführt werden
* Lösungsansatz
  + die nächste Ebene der Speicherhierarchie in den verfügbaren logischen Adressraum einbinden, auch wenn diese bedeutend langsamer ist -> Swapping (Auslagern aktuell nicht aktiver Prozesse auf die Festplatte)
* der Scheduler erkennt den Status aller Prozesse
* er entscheidet, ob und wie lange ein Prozess benötigt wird
* wird ein Prozess eine Zeit lang nicht benötigt, kann er zeitweise komplett auf die Festplatte ausgelagert werden
* der frei gewordene Platz im RAM kann dann von einem anderen Prozess belegt werden
* Vorgehensweise
  + Swap Out
    - Prozess 3 aus dem Ram auf die Festplatte auslagern, da er aktuell nicht benötigt wird



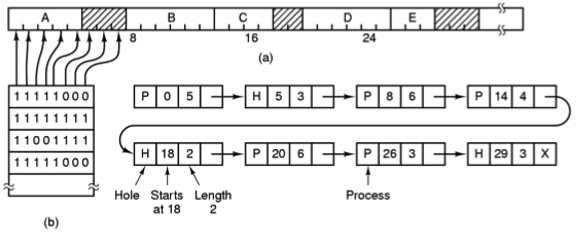
* Swap In
  + Prozess 4 von der Festplatte in den RAM einlagern
  + anschließend ist Prozess 4 zur Ausführung fertig



* Nachteile des Swappings
  + Fragmentierung des RAMs
    - es entstehen kleine Lücken
    - externe Fragmentierung zwischen den Prozessen
    - interne Fragmentierung innerhalb der Prozesse (Heap/Stack-Zwischenraum) -> hier aber nicht
  + Ineffizienz durch hohe Datenmengen und lange Wartezeiten bei Ein- und Auslagerung (Festplatten sind viel langsamer als RAM)



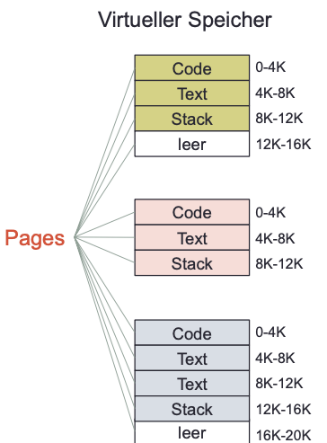
* Lücken zwischen den Prozessen werden von BS verwaltet und versucht bestmöglich wieder aufzufüllen
* Verwaltung durch
  + Bitmaps
  + Listen



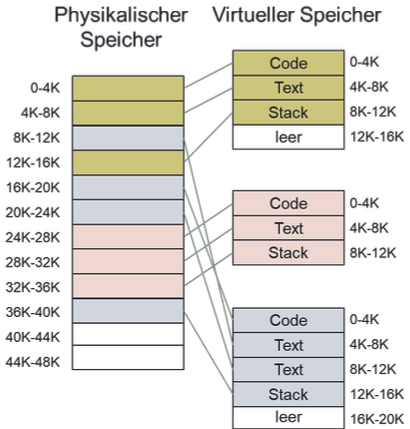
* verschiedene Algorithmen, um eine passende Lücke zu finden
  + First Fit
    - sucht die erste ausreichend große Lücke
    - sehr schnell
  + Best Fit
    - sucht die beste passende Lücke
    - dabei muss die gesamte Liste durchsucht werden -> langsam
  + Worst Fit
    - sucht die größte Lücke und lässt den größten Restbereich
  + Quick Fit
    - findet häufige Größen (4K, 8K, 16K, 20K…) sehr schnell
  + beste Leistung haben First Fit und Quick Fit
  + Fragmentierung kann aber nicht vollständig verhindert werden
  + nach einiger Zeit sind bis zu 1/3 des RAMs in so kleine Fragmente aufgeteilt, dass kein ganzer Prozess mehr hineinpasst

### Paging

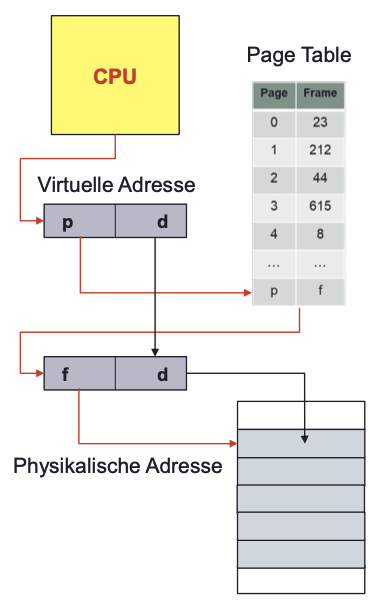
* Prozesse erwarten einen zusammenhängenden Adressraum
* Swapping hat deshalb ganze Prozesse ausgelagert
* jedoch ist bei einem laufenden Prozess nicht dessen gesamter Arbeitsspeicher in Benutzung
* Paging kann einzelne Teilbereiche eines Prozesses auslagern
* gesamter virtueller Adressraum wird in Pages (Seiten) eingeteilt
* Pages sind zusammenhängende Adressblöcke von fester Größe
* jeder Prozess belegt mit seinem Speicherbedarf eine entsprechende Anzahl an Pages



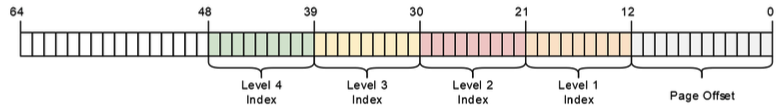
* Pages sind z.B. 4K groß
* aktuelle CPUs unterstützen mehrere Größen zwischen 4K und 16M
* Pages werden auf Page Frames (Seitenrahmen) im physikalischen Speicher abgebildet
* Page Frame und Pages müssen gleich groß sein



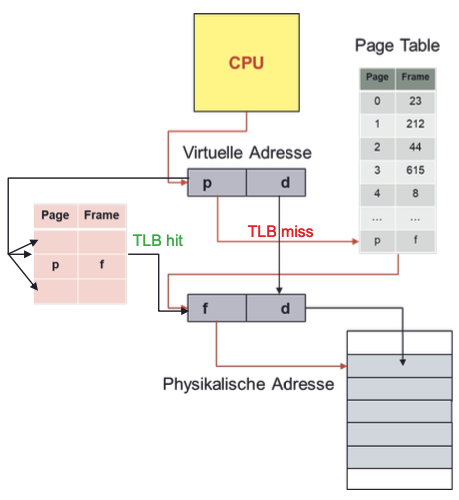
* das Auslagern von nicht benötigten Pages wird als Paging bezeichnet
* es muss nicht der gesamte Prozess ausgelagert werden, es können auch nur Prozessteile, die aktuell nicht benötigt werden ausgelagert werden
* Ein- und Auslagern einzelner Pages ist schneller als bei ganzen Prozessen
* dadurch, dass Pages und Frames gleich groß sind, gibt es keine externe Fragmentierung
* jedoch entsteht eine interne Fragmentierung
  + wenn der benötigte Speicherplatz eines Prozesses kein ganzzahliges Vielfaches der Pagegröße ist
  + somit ist eine Page dann nur zum Teil gefüllt
* die MMU hat eine Page Table (Seitentabelle)
  + diese zeigt die Übersicht zu ungenutzten Pages und die Beziehung zwischen virtuellem und physikalischem Speicher
  + es werden die Pagenummern und die Framenummern gespeichert
  + die genaue Adresse innerhalb der Seite braucht bei Abbildung nicht festgehalten werden, sie kann unverändert als Offset weitergegeben werden
* Vorgehensweise
  + von der CPU kommt die virtuelle Adresse einer Seite im Format Pagenummer (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 Page Table werden auch weitere Informationen gespeichert, z.B. Berechtigungen



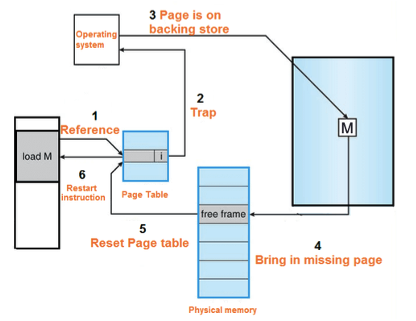
* Page Tables können aufgrund ihrer Größe kaum in Registern gespeichert werden und liegen deshalb im RAM
* bei großen Speicherbereichen wird die Page Table sehr groß -> ist dann schwer durchsuchbar
* aktuelle CPUs verwenden deshalb Multi Level Page Tables (kaskadierte Seitentabellen in mehreren Ebenen)
* AMD führte Page Map Level 4 ein
  + diese Technologie besitzt 4 Page Tables mit je 512 (29) Einträgen
  + die Seitennummer p umfasst alle 4x9=36 Bit der 4 Tabellenebenen
  + mit dem Offset d von 12 Bit ergibt sich ein 48 Bit Adressraum und damit 265 TB adressierbarer Speicher



* dennoch benötigt das Ausführen oder Auslesen einer Adressübersetzung vier Schritte
* dies wird durch den Einsatz eines Cache beschleunigt
* die MMU besitzt einen zusätzlichen Speicher (Translation Lookaside Buffer (TLB)) mit meist weniger als 256 Einträgen
* der TLB kann alle seine Einträge parallel vergleichen und ist deshalb schneller als eine Page Table
* 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

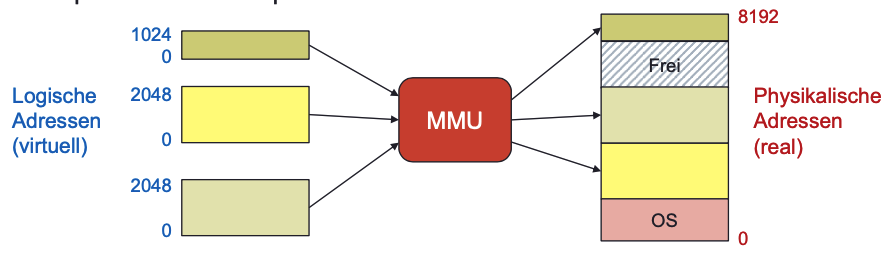


* was passiert, wenn eine Page benötigt wird, deren zugeordneter Frame aber ausgelagert ist (sich nicht im RAM befindet)?
  + die MMU generiert einen Page Fault (Seitenfehler) und der Prozess wird angehalten
  + das BS findet einen freien Frame oder muss zunächst Platz schaffen, indem ein Frame ausgelagert wird
  + anschließend muss das BS den Inhalt des angeforderten Frames auf der Festplatte suchen, auslesen und als Page in den RAM laden
  + danach wird der angehaltene Prozess fortgesetzt und der letzte Befehl erneut ausgeführt
  + Vorgehensweise
    - MMU erkennt einen Page Fault und gibt dies als Fehler aus (1-2)
    - BS führt Schritte 3 und 4 aus
    - die MMU aktualisiert die Page Table und übergibt den Inhalt der CPU (5-6)

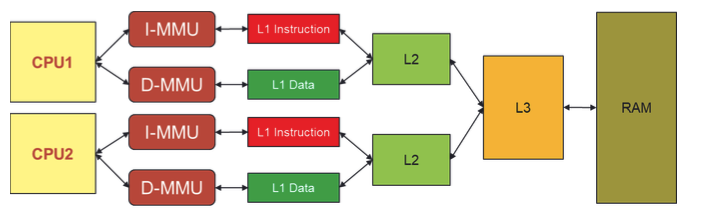


### Memory Management Unit (MMU)

* war früher eine externe Komponente zwischen CPU und Speicher
* heute sind mehrere MMUs direkt in der CPU integriert
* MMU übernimmt die gesamte Adressübersetzung aus den logischen Adressen der Prozesse in die physikalischen Adressen des RAMs

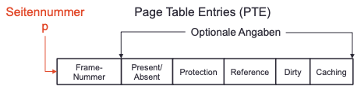


* MMU befindet sich zwischen CPU und L1-Cache
* in der CPU werden nur noch logische (virtuelle) Adressen verarbeitet



### Seitenersetzungsstrategien

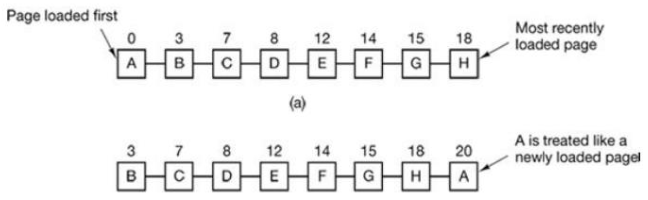
* wenn es zu einem Page Fault kommt und der RAM voll ausgelastet ist, muss das BS entscheiden, welche Page aus dem RAM ausgelagert werden muss
* diese Entscheidung zur Seitenersetzung übernehmen Algorithmen
* ein optimaler Seitenersetzungsalgorithmus würde eine Seite auslagern, die erst in ferner Zukunft wieder genutzt wird
* somit ist die Zeit bis zum nächsten Seitenfehler so lang wie möglich und damit die Anzahl der Seitenfehler minimal
* die zukünftige Seitennutzung ist aber nicht vorhersehbar -> optimaler Algorithmus ist also nicht realisierbar
* es muss also eine Annahme getroffen werden, welche Seite vermutlich länger nicht benutzt wird
* dafür wertet der Algorithmus verschiedene Informationen zu den Pages aus
* Algorithmen
  + First In First Out
    - ersetze die älteste Seite
  + Second Chance oder 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 gehört
* Angaben in einer Page Table
  + Rahmennummer f
  + Present/Absent Bit (Valid)
    - kennzeichnet, ob Page im RAM ist
  + Protection Bits
    - Schutz vor Lesen, Schreiben, Ausführen (RWX)
  + R-Bit für reference
    - kennzeichnet einen Pagezugriff
  + M-Bit für modified
    - kennzeichnet eine Seitenänderung (dirty Bit)
  + Caching-Bit
    - erlaub oder verbietet das Caching der Seite



* das R-Bit jeder Seite ist beim Start eines Prozesses 0 und wird beim ersten Zugriff der Seite auf 1 gesetzt
* in regelmäßigen Abständen setzt das BS alle R-Bits wieder zurück
* R-Bit = 0 zeigt also, dass die Seite nach dem letzten Zurücksetzen nicht wieder benutzt wurde (und somit entbehrlich ist)
* das M-Bit wird zur Kennzeichnung von geänderten Seiten verwendet, damit die Änderungen noch in die Pagefile auf der Festplatte geschrieben werden können

**First-In-First-Out-Algorithmus**

* kann zur Entscheidung über die nächste zu ersetzende Seite das Alter der Seite heranzeihen
* eine verkettete Liste mit den Startzeiten enthält am Anfang die älteste Seite
* FIFO ersetzt diese älteste Seite
* Problem: das Alter der Seite sagt nichts darüber aus, ob sie nicht sofort wieder benötigt wird
* FIFO ist also einfach, kann aber wichtige Seiten entfernen, die eigentlich bald wieder gebraucht und dann nachgeladen werden müssen
* um festzustellen, ob eine alte Seite noch gebraucht wird, kann FIFO zu Second Chance modifiziert werden

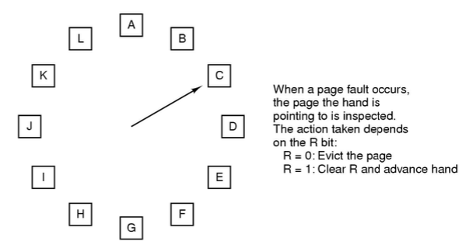


**Second-Chance-Algorithmus**

* 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 an das Ende der Liste und suche weiter
* wenn beim ersten Durchlauf keine geeignete Seite gefunden wird, taucht beim 2. Durchlauf die ehemals älteste Seite wieder auf
* jetzt ist das R-Bit = 0 und die Seite kann ersetzt werden
* effizientere Variante des Second-Chance-Algorithmus ist der Clock-Algorithmus

**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

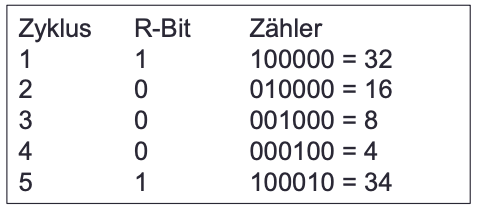


**Not Recently Used-Algorithmus**

* wertet alle vier möglichen Kombinationen von R- und M-Bit aus und bildet vier Klassen
  + 00 -> nicht gelesen und nicht modifiziert
  + 01 -> nicht gelesen, aber modifiziert
  + 10 -> gelesen, aber nicht modifiziert
  + 11 -> gelesen und modifiziert
* eine Page aus der niedrigsten belegten Klasse wird bei Bedarf ersetzt (im besten Fall 00)

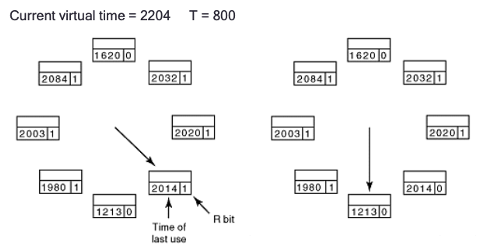
**Least-Recently-Used-Algorithmus (LRU)**

* 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 in Zukunft nicht so schnell gebraucht werden
* Herausforderung: wie wir die Nutzungshäufigkeit ermittelt?
* Lösung
  + einen Zähler einrichten, der bei jedem Zugriff auf eine Seite inkrementiert wird
  + -> ineffizient und praktisch nicht durchführbar
  + es existieren 2 machbare Alternativen
    - Zähler in der Hardware, entweder CPU oder MMU (selten realisiert)
    - Zähler in der Software
* der Zähler kann softwareseitig an LRU angenähert werden (Not Frequently Used)
* jede Page bekommt in der Page Table einen Zähler
* das R-Bit wird periodisch zurückgesetzt wie bei Not Recently Used
* vor dem Zurücksetzen des R-Bits wird aber dessen Wert ausgelesen und dem Zähler hinzugefügt
* bei einer Seitenersetzung wird die Seite gewählt, die den kleinsten Zählerstand hat
* NFU hat folgende Nachteile
  + ganze neue Seiten könnten gelöscht werden, weil sie einen niedrigen Zählerstand haben
  + 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 Altern der Zugriffe wird berücksichtigt, indem es den Zähler reduziert
* 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

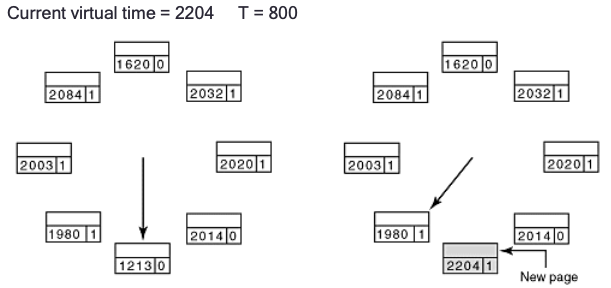


**WSClock**

* ersetzt Seiten eines Prozesses, die zwar im RAM stehen, aber nicht zum aktuellen Working Set des Prozesses gehören
* was ist ein Working Set?
  + Prozesse nutzen meist nur einen sehr kleinen Anteil ihrer Pages im RAM
  + dieser Anteil ist der Working Set
* 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 das aktuelle Working Set eines Prozesses zu ermitteln, können alle in einem Zeitintervall T referenzierten Pages gesucht werden
* nur solche Pages gehören zum Working Set
* Zeit meint hier virtuelle Zeit, also nur die CPU-Zeit, die dem Prozess tatsächlich zur Verfügung stand
* der WSClock-Algorithmus erstellt eine ringförmige Seitenliste
* diese Liste enthält alle genutzten Pages mit einem Zeitstempel und dem R-Bit, neue Pages 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 = 11, befindet sich die Seite im Working Set, weil sie innerhalb des letzten Intervalls genutzt wurde
* die Seite wird nicht entfernt, aber das R-Bit zurückgesetzt und die Suche fortgesetzt



* ist das R-Bit = 0, wird die Zeit geprüft
  + bei < T wird die Page noch einmal verschont und weitergesucht
  + bei >T wird angenommen, dass sie sich nicht mehr im Working Set befindet
* ist das M-Bit in der Page Table = 0, ist die Page sauber
* sie kann gelöscht und durch eine neue aktuelle Page ersetzt werden
* ist das M-Bit in der Page Table = 1, muss die offenbar veränderte Page zuerst auf die Festplatte geschrieben werden
* in diesem Fall muss der Algorithmus weitersuchen
* kann WSClock keine Page finden, die nicht zum Arbeitssatz gehört, wählt er eine beliebige aus und ersetzt diese



* Welcher Seitenersetzungsalgorithmus bietet die beste Leistung?
  + Aging und WSClock (beruhen beide auf LRU)

**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 knappen RAM dazu führen, dass der Prozess ständig Seitenfehler erzeugt (Thrashing) und nicht mehr vorwärtskommt
* dafür ist die globale Suche bei allen laufenden Prozessen besser geeignet
* wenn der RAM unterdimensioniert ist, kann Paging mit Swapping kombiniert werden
* Paging kann auch mit dem Segmentieren verknüpft werden, um anwachsenden Prozessen neue Adressbereiche zuzuweisen
* allerdings wurde dies selten eingesetzt und daher in den aktuellen 64-Bit-CPUs von Intel auch nicht mehr unterstützt

## Dateisysteme

* Dateien sind die Abstraktion des BS für Speicherplatz auf einem nichtflüchtigen Speichermedium wie etwa einer Festplatte
* Datei ist eine benannte, geordnete Sammlung von Informationen, die auf einem sekundären Datenträger gespeichert ist
* Dateisystem stellt ein Verzeichnis dar, in dem Dateien erzeugt, verschoben, gespeichert, verändert und gelöscht werden können

### Warum Dateien und Dateisysteme

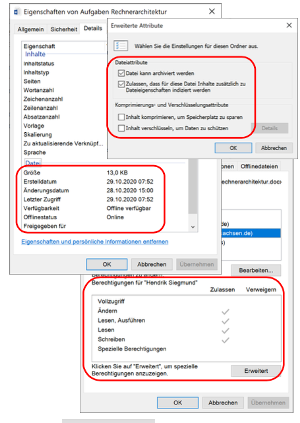
* Abstraktion
  + einfache Systemaufrufe für alle Dateioperationen
* Dateien speichern große Mengen an Informationen
* Informationen müssen permanent gespeichert werden, also auch, wenn keine Versorgungsspannung anliegt
* Informationen müssen bei Bedarf gemeinsam nutzbar sein
* Informationen müssen vor unbefugten Zugriff geschützt werden

#### Aufbau und Namen

* Dateien besitzen einen strukturierten oder unstrukturierten Inhalt
  + Byte-Sequenz: unstrukturierte Bytefolge
  + Baumstruktur: strukturierte Einträge mit Schlüssel
* Inhalte können unterschiedlich sein
  + Daten, z.B. ASCII-Zeichenfolgen, die als Text angezeigt werden
  + Ausführbare Dateien
* unter Unix sind auch Geräte Dateien
* für Dateien gibt es verschiedene Zugriffsmuster
  + Sequenziell
    - Daten werden von Anfang bis Ende der Reihen nachgelesen
      * Schreiben neuer Dateien
      * Parsen von HTML-Seiten
  + Random Access oder wahlfreier Zugriff
    - Daten werden gezielt gesucht und direkt angesteuert
      * Datenbankzugriffe
      * Seitenersetzung beim Paging
* Dateien werden oft mit eindeutigen Nummern identifiziert
* Namen sind aber für Benutzer einfacher zu handhaben
* es gibt Einschränkungen für die Vergabe von Dateinamen
  + unter DOS maximal 8 Zeichen langer Dateiname und eine Endung von 3 Zeichen
  + z.B. Programm.exe
* Pfadtiefe ist auch begrenzt

#### Dateieigenschaften

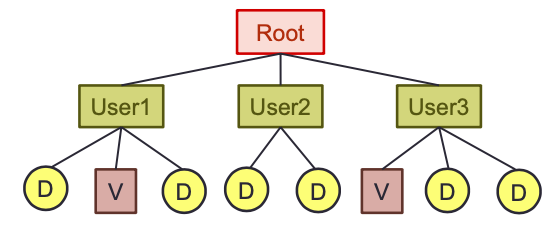
* zur Datei werden Metadaten gespeichert



* Dateien werden vom Anwender nur indirekt genutzt, nur das BS arbeitet direkt mit den Dateien
* dazu bietet das BS Systemaufrufe an, Create, Open, Close…

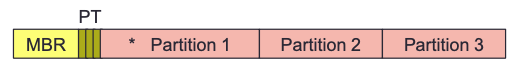
#### Typen und Funktionen

* Dateisysteme sind hierarchisch aufgebaut
* Dateien werden mit Pfaden lokalisiert

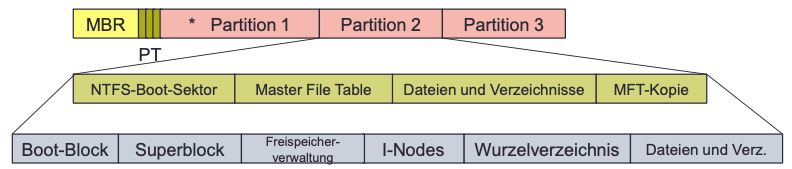


#### Abbildung auf Datenträger

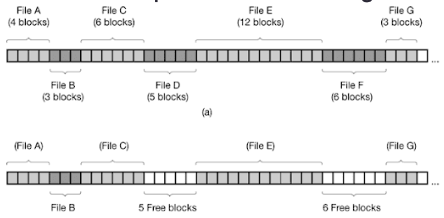
* Dateisysteme werden auf Festplatten klassisch in Partitionen gespeichert
* Partitionen sind logische Unterteilungen des physischen durch Sektoren gebildeten Speicherplatz
* typischer logische Aufbau einer Festplatte
  + Master Boot Record (MBR)
  + Partitionstabelle (PT)
  + Partitionen



* MBR umfasst Sektor 0 und enthält Angaben zur Partition für den Start eines BS
* eine Partition ist als aktiv markiert und enthält in ihrem Bott-Sektor ausführbaren Code zum Laden des BS
* Aufbau und Inhalt einer Partition sind abhängig von Dateisystem



* die Master File Table (MFT) ist das Inhaltsverzeichnis des Windows Dateisystems NTFS
* es enthält mindestens einen Eintrag pro Datei
* die MFT umfasst 12,5% der Partition
* Inhalte der MFT
  + Speicherort
  + Dateigröße
  + Erstelldatum
  + …
* in NTFS werden Dateien in Blöcken zu 4096 Byte gespeichert
* das BS schreibt Daten zusammenhängend auf die Platte
* dass ergibt die beste Performance, weil der Kopf nicht neu positioniert werden muss und bei SSDs der Cache damit rechnet, das sequenziell ausgelesen wird
* werden Dateien gelöscht oder ändern ihre Größe, entstehen Lücken (Fragmentierung)



* das BS defragmentiert die Festplatten in bestimmten Zeitabständen
* bei SSDs ist das nicht erforderlich (dort sogar schädlich)
* Löschen einer Datei entfernt in der Regel nur den Eintrag aus der MFT
* Dateien können somit evtl. wiederhergestellt werden
* mit dem TRIM-Befehl unter Windows werden Daten wirklich gelöscht

### Gemeinsame Nutzung von Dateien

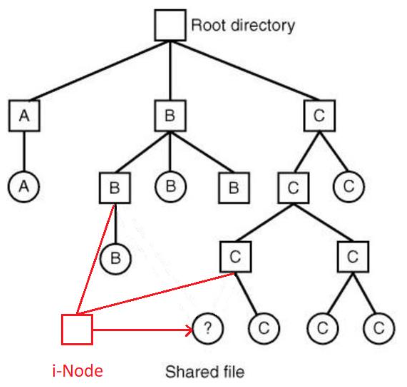
* wichtig sind
  + Speicherort und Anzahl der Instanzen
  + Zugriffsberechtigungen
  + Option für gleichzeitiges Arbeiten an derselben Datei
  + Kontrolle des von Benutzern verwendeten Speicherplatzes

#### Speicherort und Instanzen

* normalerweise sollte es pro Datei nur eine Instanz geben
* zur gemeinsamen Nutzung gibt es dann folgende Optionen
  + Verknüpfung mit Hard- und Softlink
  + gemeinsames lokales Verzeichnis
  + gemeinsames Netzlaufwerk (Server)

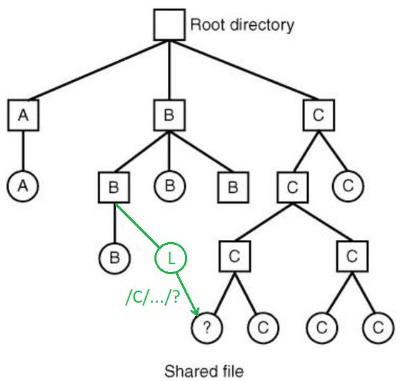
**Hard Link**

* Verweis auf Angabe zur Datei im Inhaltsverzeichnis des Dateisystems (z.B. i-Node)
* Problem
  + löscht der Eigentümer C die Datei, entsteht eine Inkonsistenz, weil im Inhaltsverzeichnis ein Eintrag steht, zu einer Datei die nicht mehr existiert



**Soft Link**

* es entsteht eine neue Datei mit einer Pfadangabe zur gemeinsam genutzten Datei
* der Verwaltungsaufwand ist höher, aber beim Löschen der Datei entsteht keine Inkonsistenz, sondern nur eine Fehlermeldung



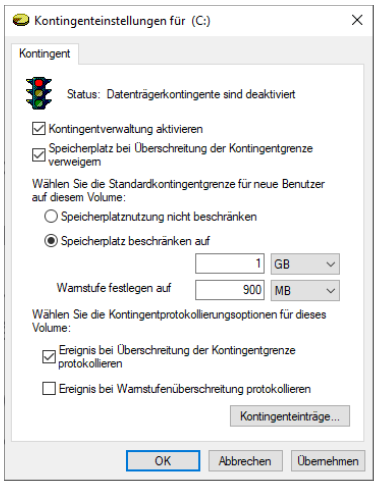
* im BS werden diese Dateien Verknüpfungen genannt

#### Zugriffsberechtigungen

* man sollte Dateien kopieren und nicht verschieben, da es sonst zu Zugriffsverweigerungen kommen kann

#### Dateikontingente

* man kann Kontingente für Partitionen setzen



### Dateikonsistenz und Journaling

* z.B. ein Kopiervorgang einer Datei kann aufgrund eines Systemabsturzes nicht abgeschlossen werden
* es besteht die Gefahr, dass Dateispeicherort und zugehörige Angaben im Inhaltsverzeichnis nicht übereinstimmen
* das Dateisystem wäre in diesem Fall inkonsistent
* die Konsistenz müsste dann nach einem Neustart geprüft werden, was sehr lange dauern kann
* Journaling soll dies verhindern
* erst wenn die Änderungen erfolgreich durchgeführt wurden, wird der entsprechende Eintrag im Journal als erledigt markiert
* man unterscheidet 2 Arten
  + vollständiges Journaling
  + Metadaten-Journaling

#### Vollständiges Journaling

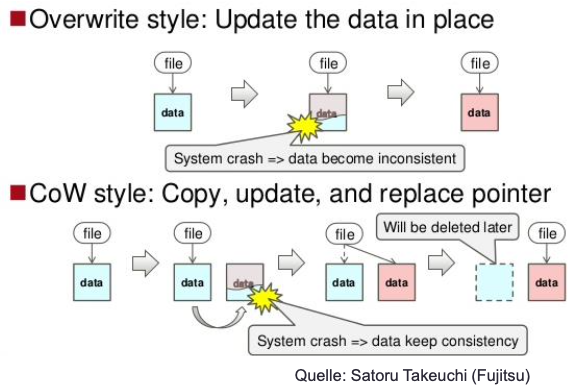
* hierbei werden alle physischen Änderungen an Blöcken, Daten und Metadaten zuerst ins Protokoll aufgenommen und erst danach im Dateisystem ausgeführt
* somit werden Inhalte und Metadaten gesichert
* nachteilig ist aber, dass jeder Vorgang 2mal ausgeführt werden muss und dass sich die Inhalte der Dateien noch in den Logs befinden (Sicherheitsrisiko)

#### 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 die Page Cache im RAM eingelesen und erst später auf die Festplatte geschrieben
* stürzt das System zwischen Journal-Änderung und Abschluss des Write-Back ab, kann es zu Datenverlusten kommen

#### Copy-on-Write

* ist eine Alternative zum Journaling
* Grundidee
  + bei Änderungen werden bestehende Dateien nicht überschrieben, sondern zuerst vollständig neue Dateien in freien Blöcken angelegt
* anschließen wird das Inhaltsverzeichnis so angepasst, dass es auf die neue Datei zeigt
* die alte Datei bleibt solange erhalte, bis die Anpassung erledigt und verifiziert ist



### Optionen für die Dateisicherung

* es gibt verschiedene Optionen
  + Image Backup
    - erstellt Abbild aller Sektoren
  + logische Dateisicherung (FIle-Level-Backup)
    - arbeitet mit Dateisystem zusammen
  + Snapshots
    - aktueller Zustand des Dateisystems lässt sicher festhalten und später wiederherstellen

#### Image Backup

* schreibt alle Sektoren in eine Image-Datei
* ein Datenträger kann somit wieder komplett hergestellt werden

#### Logische Dateisicherung

* es können bestimmte Dateien oder Verzeichnisse ausgewählt werden
* Dateisystem überprüft jeweils, ob es Änderungen an einer Datei gab und setzt dafür das Archiv-Bit
* ein Sicherungsprogramm sichert dann nur die Dateien, bei welchen das Archiv-Bit gesetzt ist
* danach werden die Archiv-Bits von Sicherungsprogramm wieder zurückgesetzt
* -> klassisches Backup-Verfahren

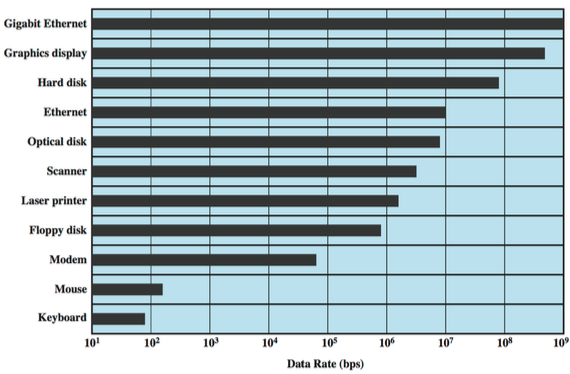
#### Snapshot

* enthält nur Informationen über Metadaten und Blockverteilung, nicht aber über vollständige Dateien
* umfasst nur die Differenz zum vorherigen Snapshot

## Ein- und Ausgabe

### Gerätekategorien

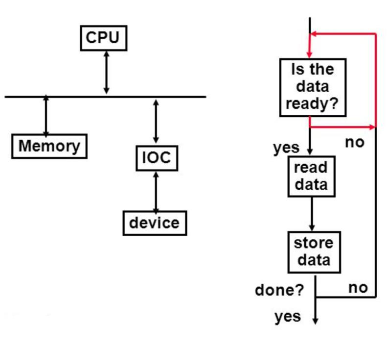
* Blockorientierte Geräte
  + verarbeiten nur Datenblöcke fester Länge
  + Festplatte, USB-Sick…
* Zeichenorientierte Geräte
  + verarbeiten Folgen einzelner Bytes
  + Kamera, Mikrofon, Tastatur, Maus…
* nicht alle Eingabe- und Ausgabegeräte lassen sich diesen beiden Gruppen zuordnen
  + Zeitgeber (Timer), Bildschirm…
* Geräte arbeiten mit sehr unterschiedlichen Datenraten



* Datenraten liegen teilweise stark und den Datenraten einer CPU, somit würde die CPU ohne weitere Maßnahmen sehr stark ausgebremst werden
* deshalb müssen die Kommunikationswege angepasst werden
* dafür gibt es verschiedene Optionen
  + Programmierte Ein- und Ausgabe (Polling)
    - regelmäßiges Abfragen der Geräte untereinander
  + Interrupt
    - Geräte löschen ein Interrupt aus
    - CPU unterbricht die laufende Arbeit und prüft die Geräteanforderung
  + Direct Memory Access (DMA)
    - die übertragene Datenmenge zwischen Interrupts wird auf Blöcke vergrößert, was die Anzahl der Interrupts verringert

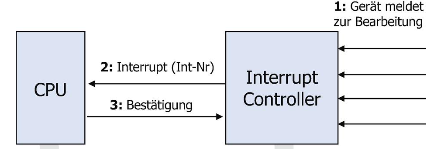
#### Programmierte Ein- und Ausgabe (Polling)

* schrittweises Abarbeiten der Ein-Ausgabeoperationen durch BS und CPU
* CPU fragt zyklisch ab, ob Daten zu liefern sind
* Nachteil
  + viel CPU-Zeit wird mit warten verbracht
* in eingebetteten Systemen ist das okay, weil sie nichts anderes zu tun haben

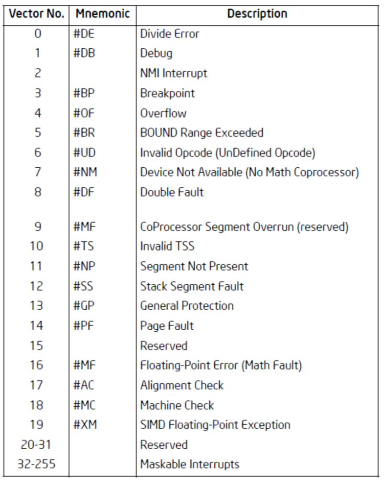


#### Interrupt

* Interrupts sind Hardwareereignisse
* für Interrupt besitzen CPUs eine spezielle Signalleitung oder einen separaten Interrupt-Controller



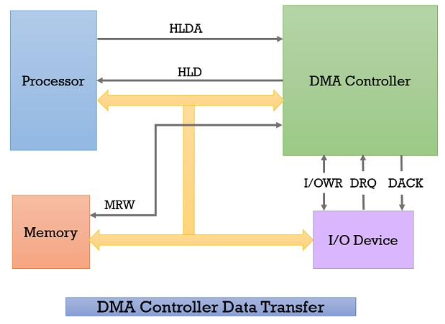
* wenn ein Interrupt auftritt, beendet die CPU die Arbeit und startet den Ein-Aufgabeprozess für das Gerät
* danach wechselt die CPU wieder zum alten Prozess und arbeitet diesen weiter ab
* Interrupt besitzen eine ID (Interrupt-Vektor)
* für Geräte nutzbar sind 32-255
* 0-32 sind für BS reserviert



* Interrupt-gesteuerte Ein- und Ausgaben entlasten die CPU vom regelmäßigen Abfragen alle Geräte
* CPU ist nur noch während der Datenübertragung am Vorgang beteiligt
* Interrupts werden priorisiert und können somit abgelehnt oder angenommen werden
* Effizienz ist höher als bei programmierter Ein- Ausgabe, solange nicht zu viele Interrupts auftreten

#### Direct Memory Access (DMA)

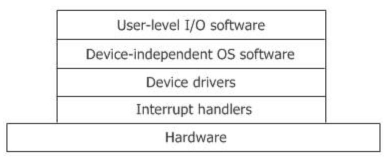
* separate Hardware (der DMA-Controller) übernimmt die Kommunikation mit dem Gerät und die Steuerung der Datenübertragung zum RAM
* der DMA-Controller erhält von Gerät einen DMA-Request (DQR)
* der DMA-Controller einigt sich mit der CPU über die Freigabe des Bussystems und steuert dann die Datenübertragung (HLD, HLDA)
* abschließend gibt er die Kontrolle über das Bussystem an die CPU zurück



* CPU und DMA-Controller müssen sich Adress- und Datenbus teilen, dafür gibt es verschiedene Modi
  + Brust-Modus
    - der DMA-Controller hat den Bus solange, bis ein Datenblock übertragen ist (maximale Datenraten)
  + Cycle-Stealing-Modus
    - der DMA-Controller mit der CPU den Bus für einzelne Taktzyklen ab, bis jeweils 1 Byte übertragen ist
  + Transparenter Modus
    - der DMA-Controller erhält den Bus nur, wenn die CPU ihn nicht benötigt (optimale Performanz des Gesamtsystems)

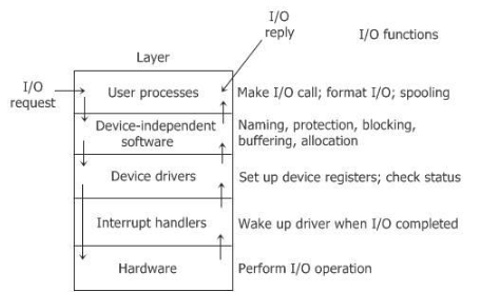
### Software

* trotz Unterschiedlichkeit der Geräte, sollen möglichst gleiche Schnittstellen geboten werden
* um das zu garantieren, wird Software zur Steuerung der Geräte in mehrere funktionalen Ebenen unterteilt
* ein Teil der Operationen ist für alle Geräte ähnlich und kann in einer systemnahmen Softwarekomponente erledigt werden
* ein anderer Teil der Operationen ist gerätespezifisch und fordert für verschiedene Gerätetypen hardwarenahe Softwarekomponenten
* Anwendungsprogramme arbeiten mit den APIs des BS für Geräteaufrufe zusammen
* die geräteunabhängige, systemnahe Software ist Teil des BS
* die gerätespezifische, hardwarenahe Komponente ist ein Gerätetreiber
* Gerätetreiber verarbeiten di Interrupt-Anforderungen der Hardware



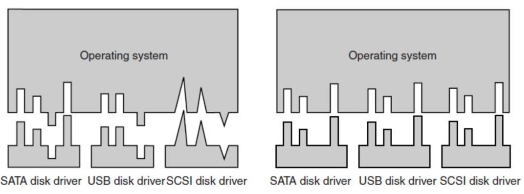
#### Arbeitsweise

* bei einer IO-Anforderung kommuniziert die Applikation mit dem BS
* das BS fordert die Gerätetreiber zur entsprechenden Funktion auf
* der Gerätetreiber übersetzt die Anforderung in Hardwarebefehle und blockiert bis zu deren Erledigung
* die Hardware führt die Operationen aus und erzeugt nach deren Abschluss ein Interrupt
* der Gerätetreiber wird nun fortgesetzt und bereitet die Datenübergabe an das BS vor
* die geräteunabhängige Softwareschicht erledigt die Datenübergabe vom Gerät an die Applikation und berücksichtigt dabei Gerätenamen, Pufferung, Schutz usw.



#### Einheitliche Schnittstelle

* für einzelne Geräteklassen gelten Standards, an die sich die Hersteller halten müssen



#### Einige Geräte und Gerätetreiber

**Zeitgeber (Timer)**

* steuern viele Abläufe im Computer
  + z.B. Timer-Interrupts beim Paging, Uhrzeitverwaltung
* Timer leiten Taktsignale von einem Quarzoszillator ab
* Startwert des Zählers wird durch ein softwaregesteuertes Halteregister festgelegt

**Tastatur**

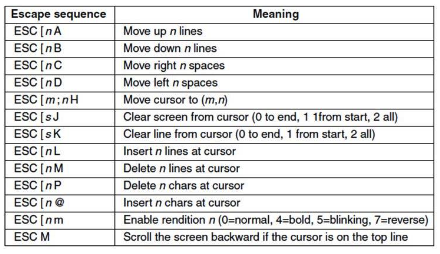
…

**Maus**

…

**Bildschirm**

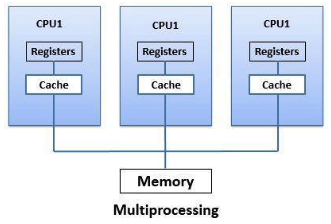
* man unterscheidet, ob das BS eine Shell oder eine grafische Benutzeroberfläche besitzt
  + die Anzeige einer Shell umfasst die Darstellung und Löschung von Zeichen und die Positionierung eines Cursors und benötigt wenig Ressourcen
  + die Anzeige einer grafischen Benutzeroberfläche ist dagegen sehr ressourcenintensiv und es wird eine umfangreiche Grafik-Hardware benötigt
* Shell
  + Ausgabe der vom Rechner gesendeten Zeichen
  + Interpretation der vom Rechner gesendeten Zeichen, z.B. Hinzufügen, Löschen…
  + ca. 25x80 Zeichen groß
  + Kommandos werden durch das Voranstellen des ESC-Zeichens 0x1B gekennzeichnet



* grafische Bildschirme benötigen eine monochrome Darstellung
* Bildschirminhalt wird als Bitmap-Bild angezeigt
* bei 640x480 Pixel also 307.200 Bit
* grafische Benutzeroberfläche umfasst folgende Grundelemente
  + Window
  + Icon
  + Menu
  + Pointer
* links oben ist Koordinate (0;0) -> geht dann nach rechts unten
* Windows-Applikationen greifen nicht direkt auf die Grafik-Hardware zu, sondern auf das Graphic Device Interface (GDI)
* Schriften waren früher Bitmaps, vergrößerte man sie aber, wurden sie unscharf
* heute sind Schriften True Type-Fonts (Vektorgrafiken)

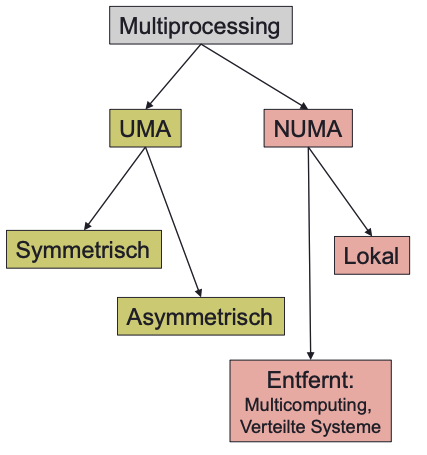
## Multiprozessorsysteme

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



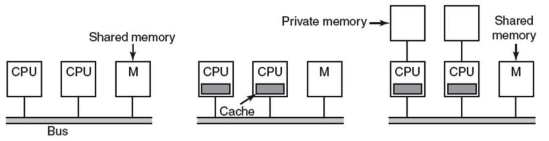
**Architekturen**

* zu unterscheiden ist die Zugriffsart auf den Speicher
  + gleichberechtigt (uniform)
  + nicht gleichberechtigt (non uniform)



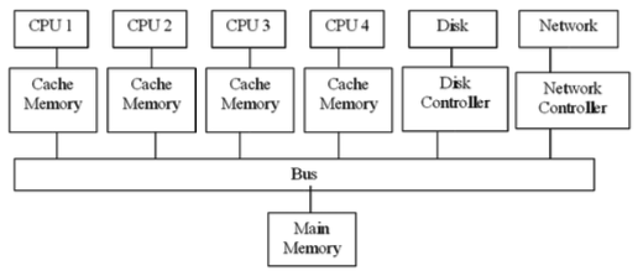
### Uniformer Zugriff

* kann mit lokalem Cache oder privatem Speicher kombiniert werden



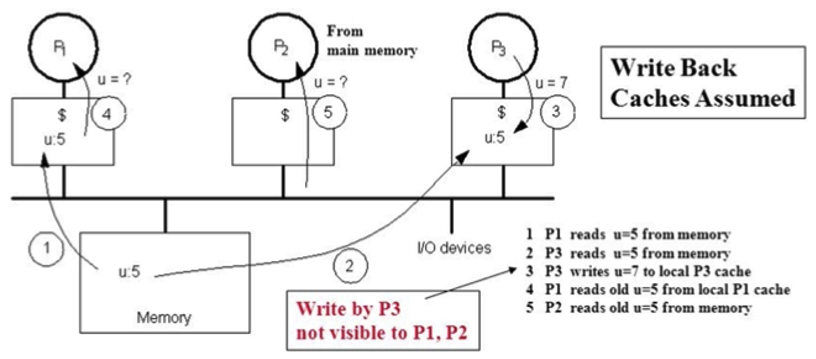
* CPUs können gleichrangig (Symmetrisches Multiprocessing) oder ungleichrangig (Asymmetrisches Multiprocessing)behandelt werden

#### Symmetrische Multiprocessing



* jeder CPU-Kern besitzt eigenen L1- und L2-Caches
* alle Kerne teilen sich den L3-Cache
* gemeinsamer RAM außerhalb des Chips
* Funktionsweise
  + ein BS verwaltet alle CPUs
  + Prozesse werden dynamisch auf CPUs verteilt
  + es besteht eine gemeinsame Datennutzung im RAM
  + es gibt echt Parallelität für Prozesse und daher höhere Leistung
* Limitierung
  + es sind nicht mehr als 64 CPUs möglich (nur in Supercomputern)
  + gemeinsamer Datenbus ist der Flaschenhals
  + Race-Conditions nicht nur zwischen Prozessen sondern auch CPUs
  + der Cache ist nicht kohärent, zwischen den individuellen Caches können Unterschiede auftreten

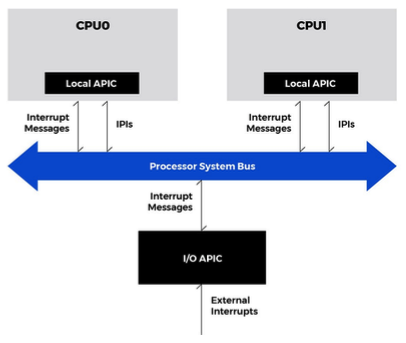
**Cache-Kohärenzen**



* System muss Kohärenzen überwachen
* wird durch das MESI-Protokoll realisiert
* für jede Cache-Zeile werden 2 zusätzliche Bits und damit 4 mögliche Zustände definiert
  + Modified
    - Daten im lokalen Cache wurden geändert
    - RAM-Kopie ungültig
  + Exclusive
    - Daten liegen nur in diesem Cache vor und sind aktuell
    - die RAM-Kopie ist gültig
  + Shared
    - Daten liegen identisch in mehreren Caches vor
    - es gab keine lokalen Änderungen
    - die RAM-Kopie ist gültig
  + Invalid
    - der Inhalt der Cache-Zeile ist ungültig oder es existiert kein Eintrag
* MESI-Protokoll gibt auch vor, dass alle CPUs ihre eigenen Cache-Zugriffe in diese Bits schreiben und auf die Veränderungen durch andere CPUs achten
* zur Überwachung der Veränderungen lesen und setzen die CPUs definierte Signale auf dem Bus (Bus-Snooping)
* dies ist bei sehr vielen CPUs (mehr als 64) nicht machbar

**Interrupt Handling**

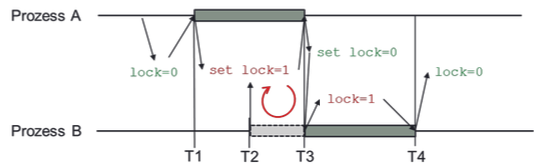
* Application Programmable Integrated Circuits (APIC) in jeder CPU kommunizieren miteinander



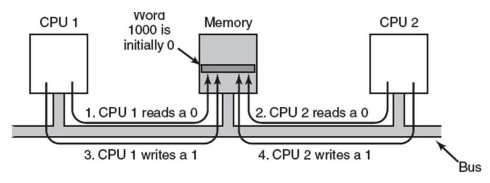
* CPUs könne sich gegenseitig aus dem Suspend (Schlafzustand mit niedrigem Energieverbrauch) wecken
* Aber: Maschinenbefehle zum Sperren und Erlauben wirken je CPU
* dies muss bei der Synchronisation berücksichtigt werden

**Synchronisation**

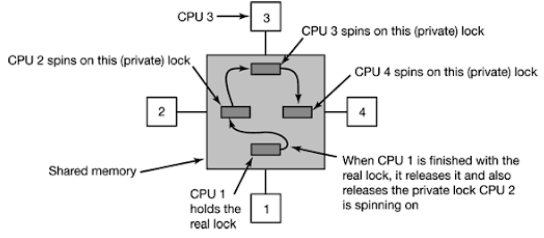
* beschreibt hier die Vermeidung von Race-Conditions bei CPUs
* wie bei Einzelprozessorsystemen kann hier auch Test, Set & Lock (TSL) verwenden werden
* Problem
  + Prozesse laufen nicht mehr sequenziell in zeitscheiden, sondern tatsächlich parallel



* wenn ein Prozess eines CPUs in einen Kritischen Abschnitt eintritt, muss sichergestellt werden, dass er der einzige bleibt
* dies kann z.B. durch sperren des gemeinsamen Busses realisiert werden

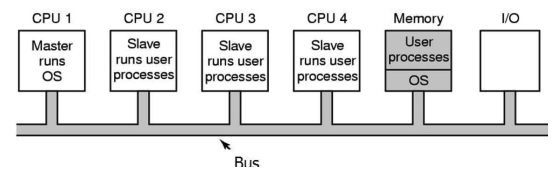


* jedoch müsste der Bus dann bei jedem TSL gesperrt und wieder freigegeben werden -> nicht performant
* besser ist: Individualisieren der Sperrvariablen
* jeder CPU erhält eine eigene Variable im gemeinsam zugänglichen Cache-Bereich
* CPUs erhalten beim Fehlschlag des Eintritts in eine kritische Region eine individuelle Sperre und reihen sich in eine Schlange wartender CPUs ein
* wir die kritische Region von der CPU, die sie gerade nutzt, wieder freigegeben, schließt dies auch die Individuelle Sperre der nächsten wartenden CPU
* was bleibt, ist das Problem des Wartens:
  + jede CPU, die auf eine Ressource zugreift, befindet sich im Spinlock und fragt immer wieder die Variable ab
  + Alternative wäre, blockieren des betreffenden Prozesses und ein Kontextwechsel
  + aber auch Kontextwechsel erzeugen einen Overhead



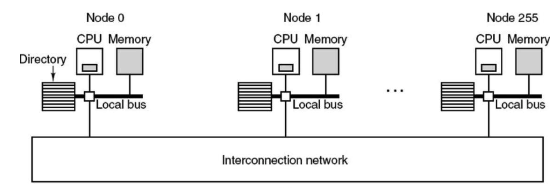
**Scheduling**

#### Asymmetrisches Multiprocessing



#### Non-uniform Zugriff

* CPUs besitzen keinen gleichrangigen Speicherzugriff, sondern sind netzwerkgekoppelt



# Virtualisierung

* Abstraktion der Hardware zur einfacheren und flexibleren Nutzung durch Software und Anwender
* bei vollständiger Abstraktion steht anstelle einer physischen Hardware eine rein virtuelle Maschine
* virtuelles Gastsystem in einem Wirtssystem aus physischer Hardware und einer besonderen Virtualisierungsschicht, dem Virtual Machine Monitor oder Hypervisor

### Konzept

* Software zur Virtualisierung auf Wirtssystem installiert (Hypervisor)
* auf VMs werden BS und benötigte Serverdienste oder Applikationen installiert

### Motivation

* Vorteile:
  + physischer Rechner kann mehrere VMs unterstützen 🡪 vorhandene Hardware besser ausnutzen
  + Auslastungsschwankungen können besser ausgeglichen werden
  + Hochverfügbarkeit
  + kostengünstige Test- und Schulungsumgebungen
  + historische Applikationen und Betriebssysteme können auf der VM laufen, obwohl sie von der eigentlichen Hardware nicht mehr unterstützt werden
* Nachteile:
  + Overhead durch das Wirtssystem und die notwendige Koordination der Gastsysteme
  + Konkurrenz um verfügbare Hardware
  + größere Probleme bei Ausfall eines physischen Servers 🡪 zusätzliche Redundanzen und komplexes Management der VM

### Architekturen

* Kategorisierung:
  + nach 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-Virtualisierungen unterschieden

### Typ-1

* Hypervisor direkt auf der Hardware installiert
* Hypervisor hat nur Mini-BS
* Vollvirtualisierung der Hardware
* Bsp: XenServer (Citrix Systems), vSphere ESX (VMware), Hyper-V (Microsoft)
* Rechnerinfrastruktur: ausreichend Rechenleistung, CPU-Cores, RAM, Storage, …
* vollkommene Isolierung gleichzeitig bearbeiteter Aufgaben und genutzter Ressourcen erforderlich

### Typ-2

* Hypervisor als Anwendung auf einem BS
* nutzt Ressourcen des OS und nicht der Hardware
* Hypervisor emuliert die Hardware für das Gastsystem
* vollständige Virtualisierung
* Bsp: Microsoft Virtual PC, VMware Workstation

### Vergleich Typ-1 & Typ-2

* Typ-1:
  + geringer Overhead
  + Produktiveinsatz
* Typ-2
  + Hardwareunabhängig für Gastsystem, keine Treiber nötig
  + mehr Overhead
  + für Testumgebungen
* beide Typen virtualisieren die Hardware vollständig und transparent für Gastsysteme

### Aufgaben Hypervisor

* Trennung in Benutzermodus und Kernmodus aufheben bzw. verschieben
* Speicherseiten der GastBS verwalten
* I/O-Anforderungen und Interrupts kontrollieren und korrekt zuweisen

### Paravirtualisierung

* Gastsysteme werden portiert 🡪 Kernel wird an das Laufen auf einem Hypervisor angepasst
* Gastsystem kommuniziert mit dem Hypervisor über spezielle APIs
* Hypervisor wird zum Microkernel
* Vorteil: bessere Leistung der VMs
* Bsps: XenServer, Linux KVM

Table of Contents

[Rechnerarchitektur 1](#_Toc58246671)

[Universelle Rechner und Alan Turing 1](#_Toc58246672)

[EVA-Prinzip 1](#_Toc58246673)

[Von Neumann Architektur 1](#_Toc58246674)

[weitere Verbesserung (Memory Wall) 3](#_Toc58246675)

[weitere Verbesserung (Hub-Architektur) 3](#_Toc58246676)

[weitere Verbesserung (Multicore) 3](#_Toc58246677)

[Harvard-Architektur 3](#_Toc58246678)

[weitere Verbesserung (von 16 Bit zu 64 Bit) 3](#_Toc58246679)

[weitere CPU-Architekturen 3](#_Toc58246680)

[weitere Hardware 4](#_Toc58246681)

[mechanische Festplatten 4](#_Toc58246682)

[Schnittstellen für mechanische Festplatten 4](#_Toc58246683)

[SSD-Festplatten 4](#_Toc58246684)

[Tastatur 6](#_Toc58246685)

[Maus 6](#_Toc58246686)

[Monitore 6](#_Toc58246687)

[Betriebssysteme 7](#_Toc58246688)

[Entwicklungsgeschichte 7](#_Toc58246689)

[Klassifizierung von Betriebssystemen 7](#_Toc58246690)

[Nach Computertyp 7](#_Toc58246691)

[Nach Betriebsart 7](#_Toc58246692)

[nach Anzahl gleichzeitiger Aufträge 7](#_Toc58246693)

[Nach Anzahl gleichzeitiger Nutzer 8](#_Toc58246694)

[Firmware 8](#_Toc58246695)

[Aufgaben des BS 8](#_Toc58246696)

[Abstraktion der Hardware 8](#_Toc58246697)

[Verwaltung der Hardware 9](#_Toc58246698)

[Konzepte 10](#_Toc58246699)

[Virtualisierung 10](#_Toc58246700)

[Prozesse 10](#_Toc58246701)

[Dateien 11](#_Toc58246702)

[Treiber für Ein- und Ausgabegeräte 11](#_Toc58246703)

[Maßnahmen für Datenschutz und Datensicherheit 11](#_Toc58246704)

[Aufgaben des BS im Detail 12](#_Toc58246705)

[Prozesse und Threads 12](#_Toc58246706)

[Prozessmodell 12](#_Toc58246707)

[Prozessbestandteile 12](#_Toc58246708)

[Prozesserzeugung 13](#_Toc58246709)

[Prozesszustände 13](#_Toc58246710)

[Prozesse beenden 14](#_Toc58246711)

[Threads 14](#_Toc58246712)

[Threadebenen 15](#_Toc58246713)

[Vergleich mit Prozessen 15](#_Toc58246714)

[Interprozesskommunikation und Scheduling 15](#_Toc58246715)

[Deadlock 18](#_Toc58246716)

[Behandlung von Deadlocks 18](#_Toc58246717)

[Scheduling 21](#_Toc58246718)

[Speicherverwaltung 25](#_Toc58246719)

[Monoprogrammierung und physikalischer Speicher 25](#_Toc58246720)

[Basis- und Limit-Register 26](#_Toc58246721)

[Speicherabstraktion in x86-Prozessoren 27](#_Toc58246722)

[Swapping 27](#_Toc58246723)

[Paging 29](#_Toc58246724)

[Memory Management Unit (MMU) 33](#_Toc58246725)

[Seitenersetzungsstrategien 33](#_Toc58246726)

[Dateisysteme 38](#_Toc58246727)

[Warum Dateien und Dateisysteme 38](#_Toc58246728)

[Gemeinsame Nutzung von Dateien 40](#_Toc58246729)

[Dateikonsistenz und Journaling 42](#_Toc58246730)

[Optionen für die Dateisicherung 43](#_Toc58246731)

[Ein- und Ausgabe 44](#_Toc58246732)

[Gerätekategorien 44](#_Toc58246733)

[Software 46](#_Toc58246734)

[Multiprozessorsysteme 49](#_Toc58246735)

[Uniformer Zugriff 49](#_Toc58246736)

[Virtualisierung 53](#_Toc58246737)

[Konzept 53](#_Toc58246738)

[Motivation 53](#_Toc58246739)

[Architekturen 53](#_Toc58246740)

[Typ-1 53](#_Toc58246741)

[Typ-2 54](#_Toc58246742)

[Vergleich Typ-1 & Typ-2 54](#_Toc58246743)

[Aufgaben Hypervisor 54](#_Toc58246744)

[Paravirtualisierung 54](#_Toc58246745)