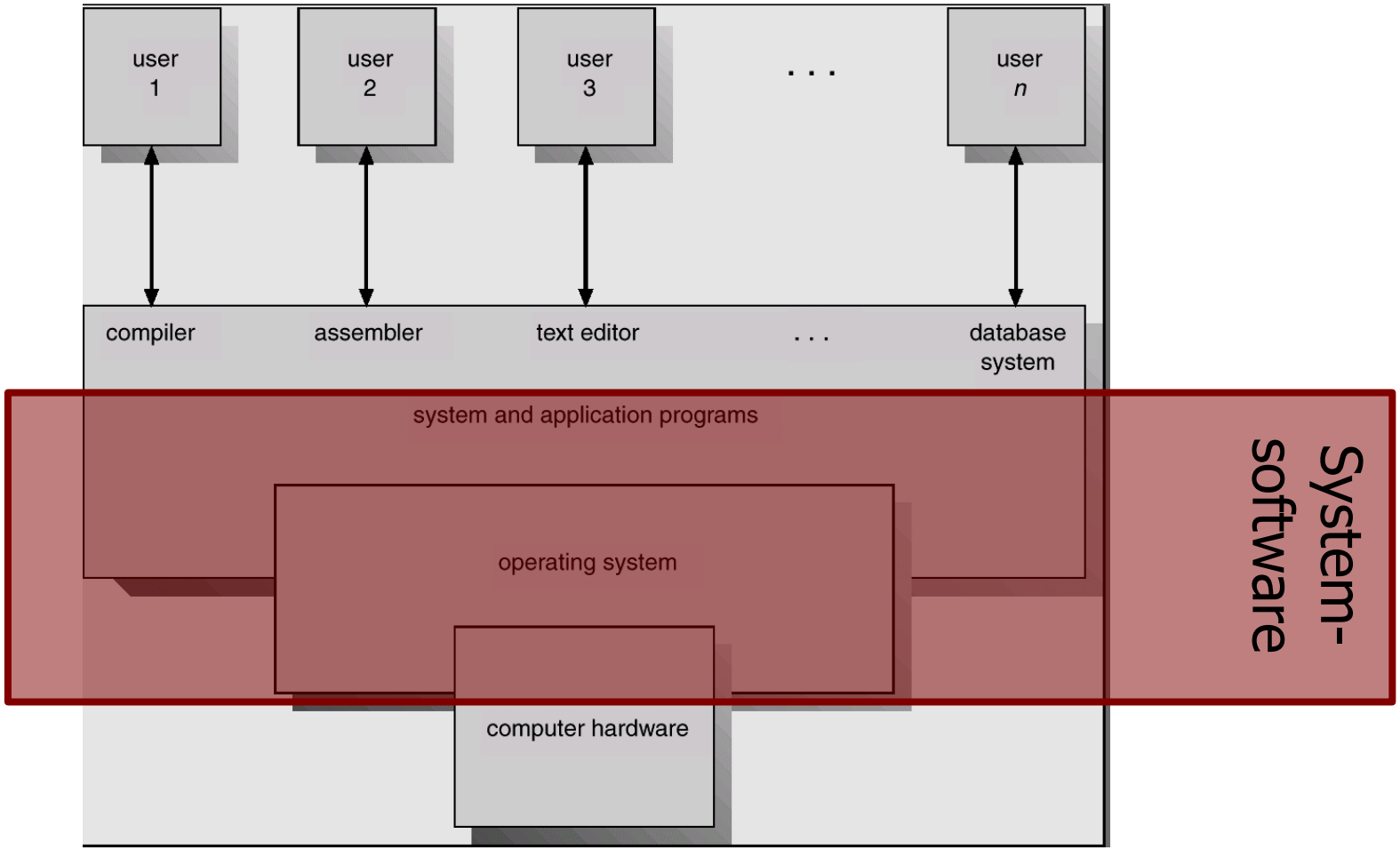
1. Rechnerarchitektur und Betriebssysteme

- Überblick
  - 1.1 Rechnerarchitektur
  - 1.2 Struktur von Betriebssystemen
  - 1.3 Fallstudien: Windows, Unix, Android, iOS
  - 1.4 Geschichtliche Betrachtung

1.1 Computersysteme und Rechnerarchitektur

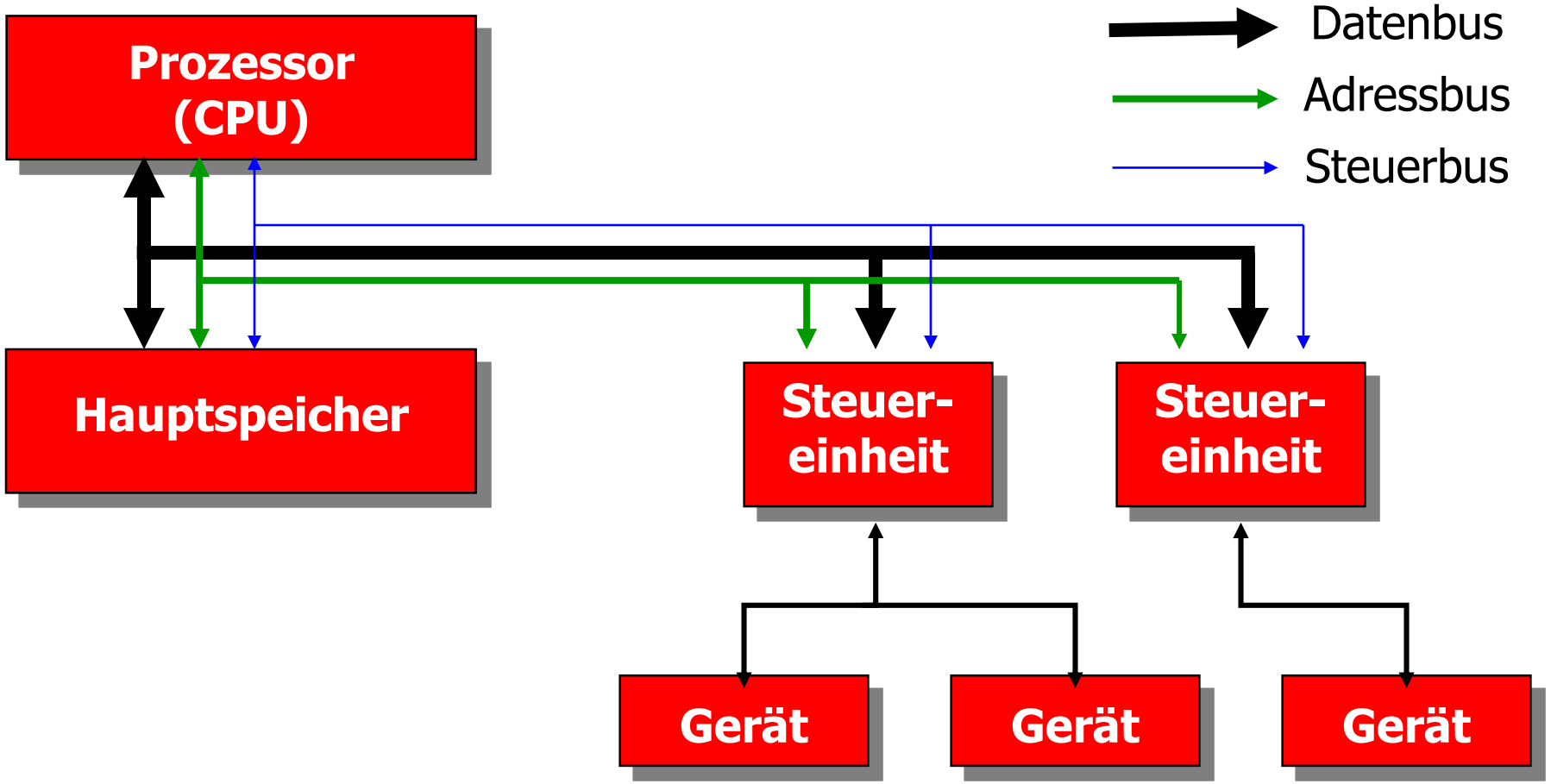


Rechnerarchitektur

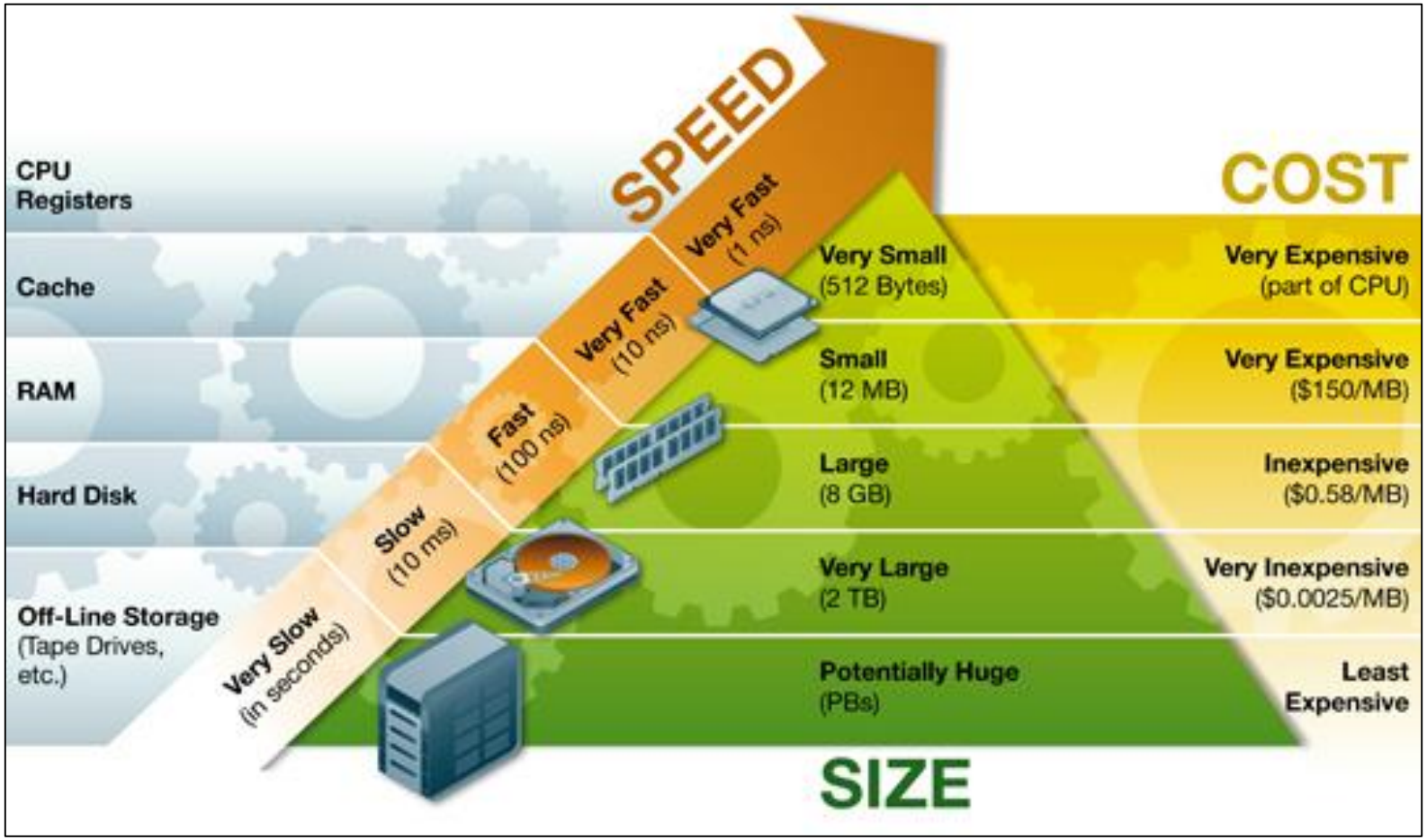
- Systemsoftware und Systemprogrammierung eng gekoppelt an Rechnerarchitektur

- Architektur eines Rechners wird definiert durch
  - **Operationsprinzip** (funktionales Verhalten)
    - Informationsstrukturen und die darauf anwendbaren Operationen
    - Kontrollstrukturen: beschreiben den zeitlichen Ablauf der Operationen
  - **Struktur** (die Art der Realisierung des Operationsprinzips)
    - Welche Komponenten sind für den Aufbau einer Architektur notwendig?

Von-Neumann Rechnerarchitektur



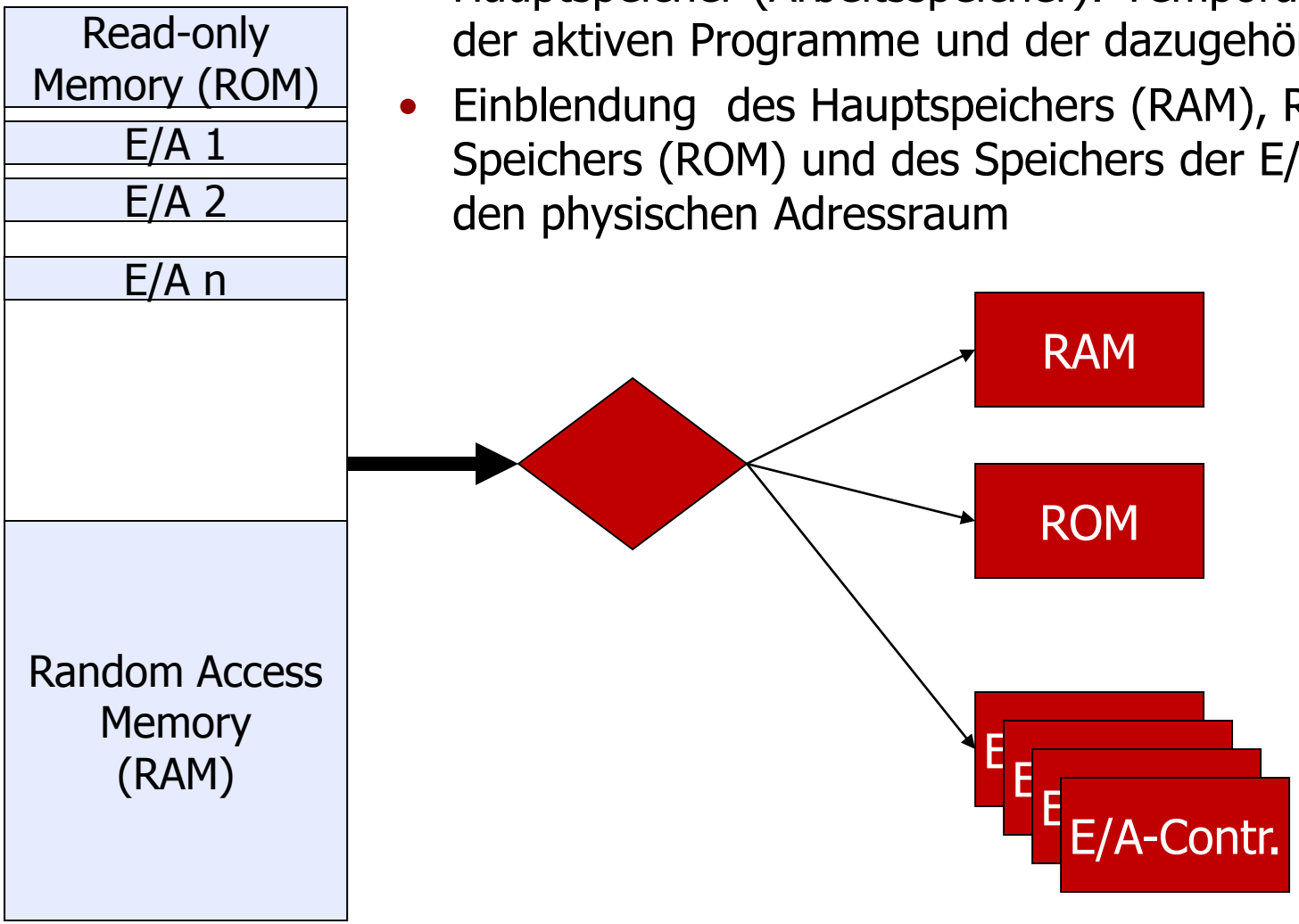
Speicherhierarchie



ts.avnet.com

Hauptspeicher

- Hauptspeicher (Arbeitsspeicher): Temporäre Speicherung der aktiven Programme und der dazugehörigen Daten
- Einblendung des Hauptspeichers (RAM), Read-Only Speichers (ROM) und des Speichers der E/A-Geräte in den physischen Adressraum



Cache-Speicher

- Zwischenspeicher zur Verkleinerung der Lücke zwischen Prozessor- und Speichergeschwindigkeit
  - Inhalt einzelner Zellen samt Adresse wird zwischen gespeichert
  - Beim Datenzugriff wird zunächst der Cache überprüft:
    - Falls Datum vorhanden ⇒ kurze Ladeoperation (Cache-Hit)
    - Sonst wird ein Arbeitsspeicherzugriff initiiert (Cache-Miss)

- Moderne Caches erreichen Trefferraten bis zu 90%
  - Hauptgrund ist die Referenzlokalität der meisten Programme: Sequentielle Ausführung, Variablen in Schleifen usw.

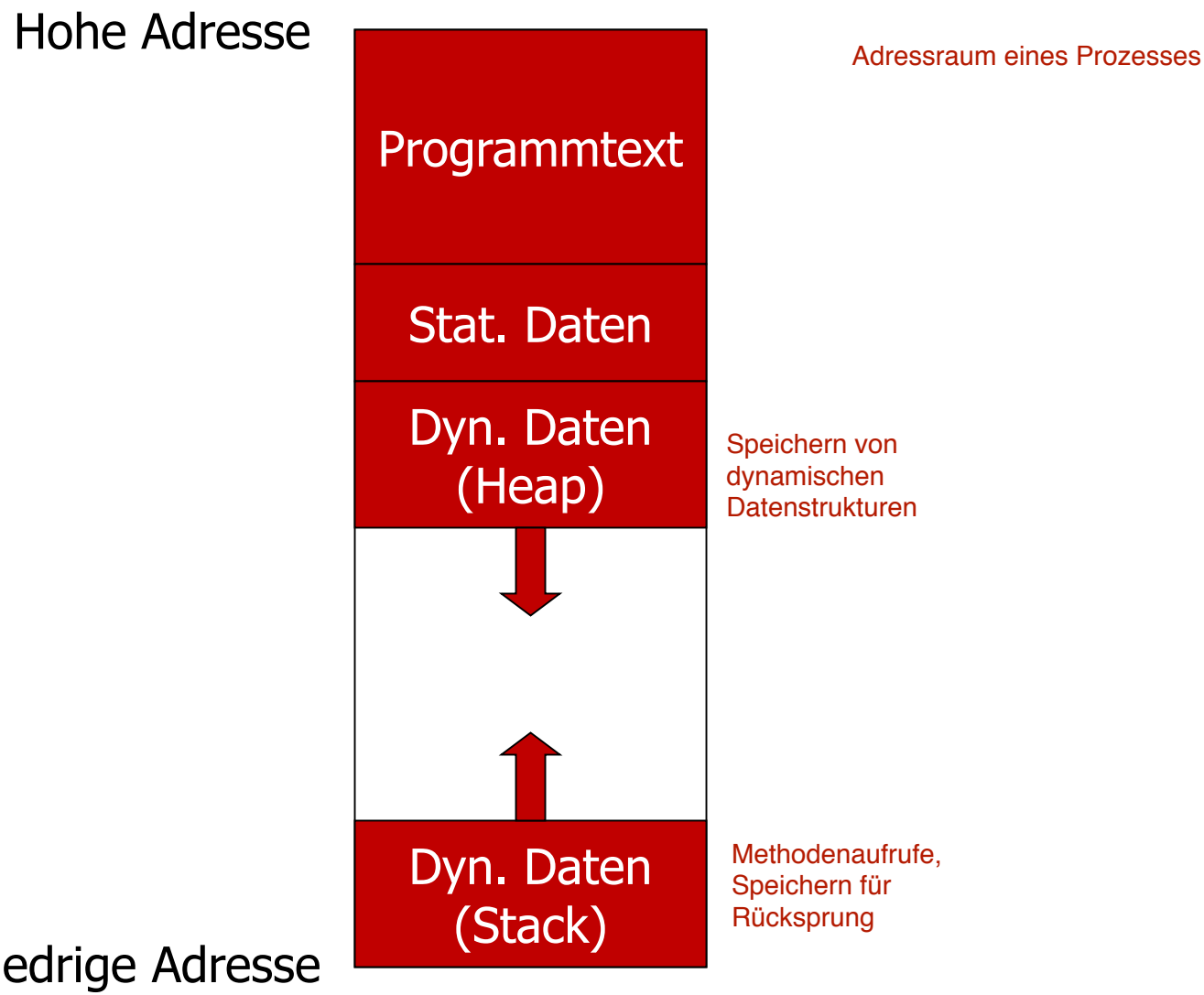
- Kalter / Heißer Cache
  - Gerade geladenes Programm ⇒ Cacheinhalte entsprechen nicht den vom Programm referenzierten Zellen
    - Geringe Trefferrate ⇒ Kalter (ineffizienter) Cache
  - Nach Vorlaufzeit: Cache passt sich an das aktuelle Programm
    - Trefferwahrscheinlichkeit steigt an ⇒ Heißer (effizienter) Cache

Adressräume

wichtig!

- Adressraum
  - zusammenhängende Menge von Adressen
  - dient der Aufnahme aller zur Ausführung eines Programms notwendigen Instruktionen und Datenstrukturen
- Teile des Adressraum können undefiniert sein
  - ⇒ Zugriff darauf führt zu einem Fehler
- Unterscheidung
  - Physischer Adressraum (definiert durch Breite des Adressbusses)
  - Logischer Adressraum, Programmadressraum (aus der Sicht des Programms)
  - Virtueller Adressraum (zur effizienten Nutzung des Hauptspeichers)

Layout eines logischen Adressraums



Statische und dynamische Variablen

- Statische Variablen
  - Benötigter Speicher wird im Quelltext festgelegt
  - Lässt sich während der Laufzeit nicht mehr verändern
- Problem: Anzahl der Einträge abhängig von Nutzung und daher zur Erstellungszeit meist unbekannt!

- Lösung
  - Obere Grenze z.B. für Arrays festlegen (*unflexibel, verschwenderisch*) oder
  - Dynamische Speicherverwaltung: Reservierung von Speicherplatz während der Laufzeit

Zeiger

- Grundlage der dynamischen Programmierung
- Beispiel Adressenliste
  - Statische Lösung: Array von Strukturen
    - Einfügen bzw. Löschen von Elementen aufwendig
    - Anzahl der Adressen ist unbekannt (Array über- oder unterdimensioniert)
  - Dynamische Lösung: einfach verkettete Liste
- Ein Zeiger muss definiert werden. Beispiel in C:
  `&ptr` liefert **die Adresse** auf der sich `ptr` befindet
  `*ptr` liefert **den Wert**, der auf dieser Adresse gespeichert ist
- Ein Zeiger ist immer typgebunden: `Datentyp *Name;`
- Der Inhalt eines Zeigers ist immer eine Adresse
  - Durch Definition der Zeigervariable wird nur so viel Speicherplatz reserviert, wie für die Darstellung einer Adresse notwendig ist (*2 Byte bzw. 4 Byte*)

Speicherzuweisung für die eigentlichen Daten

- Reservierung des nötigen Speichers im Heap durch Verwendung von Funktionen wie `malloc(unsigned size)`
  1. Aufruf der Funktion `malloc(size)` mit der genauen Angabe, wie viel Speicherplatz benötigt wird
  2. Steht genug Speicher zur Verfügung
     ⇒ Rückgabe des reservierten Speicherblocks, sonst der vordefinierte NULL-Zeiger
  3. Der Speicherblock wird mit den Daten gefüllt
- Der zugewiesene Speicher wird mit der Funktion `free()` wieder freigegeben
  ⇒ Anwendung nur sinnvoll, wenn `free()` nicht am Programmende steht

Größenangabe bei `malloc()`

- Bei Aufruf von `malloc()` muss die Anzahl der zu reservierenden Bytes übergeben werden, d.h. die Größe des Objektes, das durch den Zeiger referenziert wird
- Konstanten als Größenangabe bei `malloc()` führt zu schlecht portierbaren Programmen
- Richtiger Aufruf

```
Datentyp *ptr;
ptr = malloc(sizeof(*ptr));
```

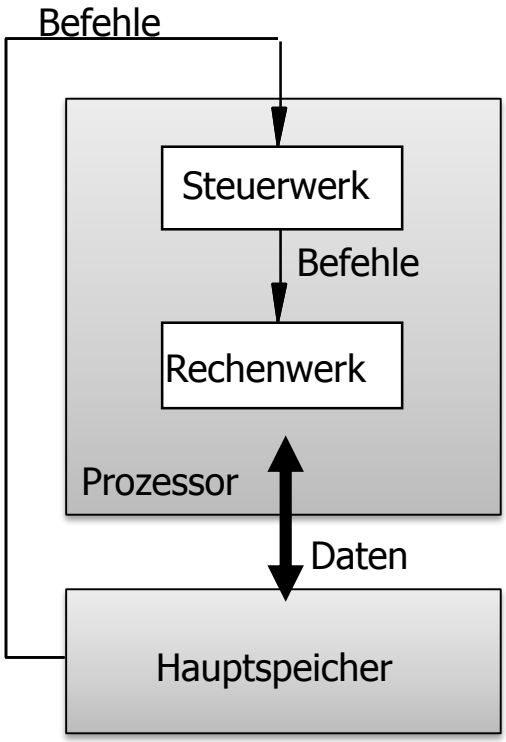
- NULL-Zeiger: Vordefinierter Zeiger, dessen Wert sich von allen regulären Zeigern unterscheidet
  - ⇒Nutzung zur Anzeige von Fehlern
  - ⇒Bei jedem Aufruf einer Funktion mit Rückgabe Zeiger muss auf NULL-getestet und ggf. Fehler abgefangen werden

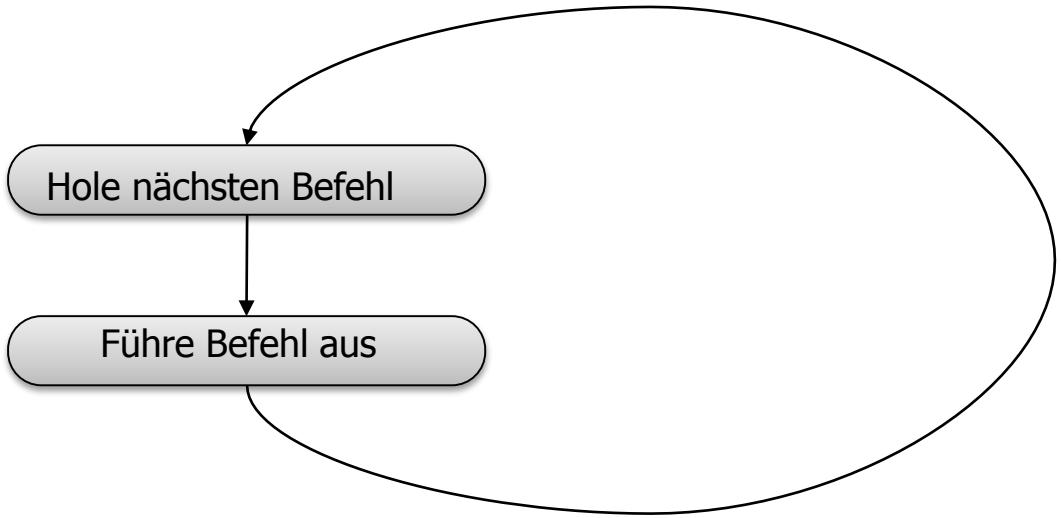
Datenstrukturen mit Zeigern

- Entwurf dynamischer Strukturen
  - Zeiger auf Strukturen konstruieren
  - Zeiger in der Struktur selbst einbetten
- Wichtige Datenstrukturen
  - Listen: Jedes Element kennt seinen Nachfolger und evtl. seinen Vorgänger
  - Bäume: Vater-Sohn-Relation, d.h. jeder Knoten hat ein, zwei oder mehrere Nachfolger
  - Stack: Art von Liste. Der Zugriff erfolgt immer über das oberste Element (LIFO: Last In First Out)
  - Queues: Die Elemente werden am Listenende eingefügt und am Listenanfang gelesen (FIFO: First In First Out)

Wie arbeitet der Prozessor?

- In jedem Zyklus wird durch das Steuerwerk der nächste auszuführende Befehl aus dem Hauptspeicher beschafft





Prozessorzyklus

Sicherheit der CPU

- Unterscheidung aus Sicherheitsgründen zwischen zwei Zuständen oder Modi (Bit im Prozessorstatuswort)
  - Benutzermodus (*user mode*)
    - Einige Befehle gesperrt und einige Register nicht zugreifbar, in der Regel für Benutzerprogramme

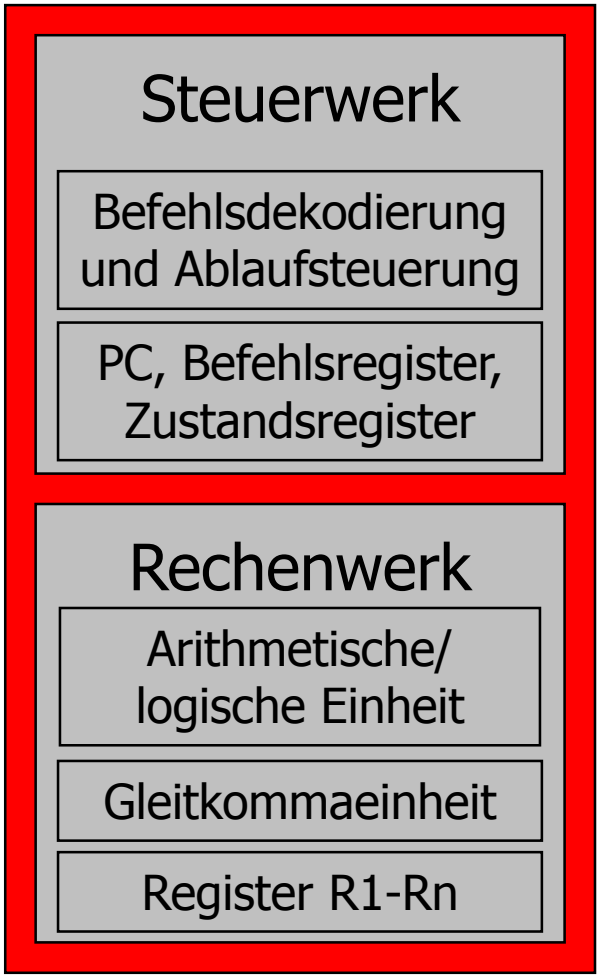
    "Sandkasten"-Modus
  - Systemmodus/privilegierter Zustand (*system/supervisor mode*, ...)
    - alle Befehle zulässig, alle Register benutzbar, in der Regel für das Betriebssystem
- Modusänderung mit einem privilegierten Befehl
  - Benutzerprogramm kann das Betriebssystem aufrufen (Befehl SVC, trap)

    ⇒CPU wechselt in privilegierten Zustand

    ⇒Aufgabe des Betriebssystems, vor der Rückkehr in das Benutzerprogramm den Zustand wieder zurückzusetzen

Prozessor

- Grundelemente eines Prozessors
  - Rechenwerk
  - Steuerwerk: Stellt Daten für das Rechenwerk zur Verfügung
    - ⇒ Holt Befehle aus dem Speicher
    - ⇒ Koordiniert den internen Ablauf
  - Register: Speicher mit Informationen über die aktuelle Programmbearbeitung, z.B.
    - Rechenregister, Indexregister
    - Stapelzeiger (*stack pointer*)
    - Basisregister (*base pointer*)
    - Befehlszähler (*program counter*, PC)
    - Unterbrechungsregister,...



Geschwindigkeit hängt starkt von Kommunikation mit externen Geräten ab (hier kann man viel gewinnen, aber auch verlieren!)

Vielfalt der Geräte

Device	Purpose	Partner	Data Rate
Keyboard	input	human	10 B/s
Mouse	input	human	200 B/s
Microphone	input	human	1-8 KB/s
Voice output	output	human	1-8 KB/s
Line printer	output	human	1 KB/s
Laser printer	output	human	0.1-100 MB/s
Graphic display	output	human	30-1000 MB/s
CPU to frame buffer	output	machine	133-8000 MB/s
Network-LAN	in-/output	machine	10-100 MB/s
Infiniband	in-/output	machine	250-6000 MB/s
Optical disk	storage	machine	0.15-54 MB/s
Magnetic tape	storage	machine	2-120 MB/s
Hard disk	storage	machine	100-150 MB/s
Solid state disk	storage	machine	100-700 MB/s

- Die Vielfalt der Geräte erfordert unterschiedliche Vorgehensweisen bei der Gestaltung der E/A-Vorgänge

Ein- und Ausgabearchitekturen

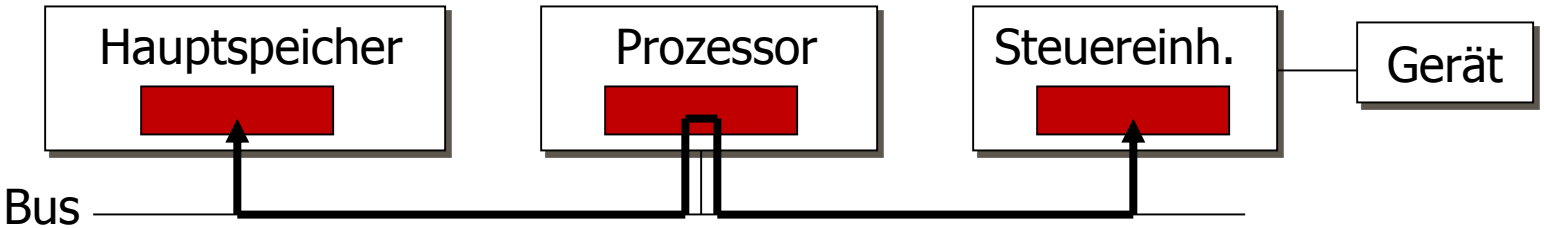
- Zwei wesentliche Ansätze
  - Speicherbasierte E/A (Memory-mapped I/O, Programmed I/O): einfach, aber langsam
  - Direkter Speicherzugriff (DMA, Direct Memory Access): zusätzlicher Hardware, komplexer, schnell, Standard

durch die CPU gesteuert -> langsam! deswegen nur für Übertragungen mit wenigen Daten geeignet

Zugriff an der CPU vorbei -> CPU hilft nur beim Aufbau, danach direkte Kommunikation; heute Standard

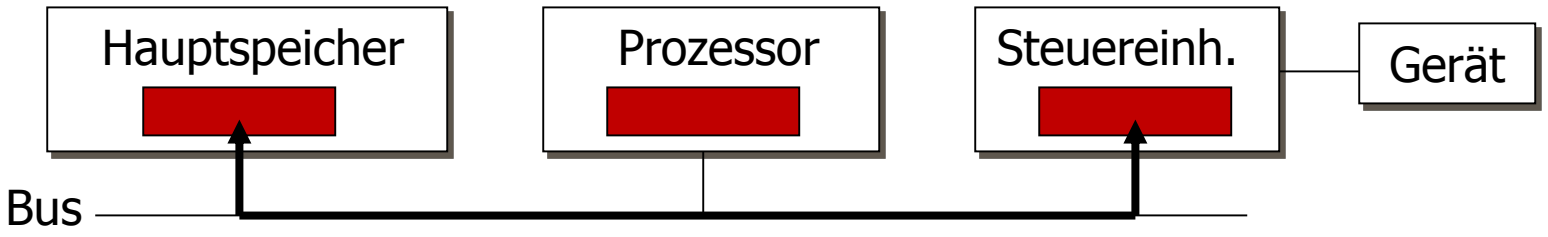
Speicherbasierte E/A

- CPU liest/schreibt byteweise Daten in Register der Steuereinheit
- Auslösung: Wie erhält die Steuereinheit ihre Aufträge?
  - CPU füllt die entsprechenden Register der Steuereinheit mit
    - Art der Operation (z.B. Lesen, Schreiben) im Befehlsregister: Was ist zu tun
    - Quelle/Ziel im Datenregister: auszutauschende Daten
    - Statusregister: Zustand der Steuereinheit und des Geräts



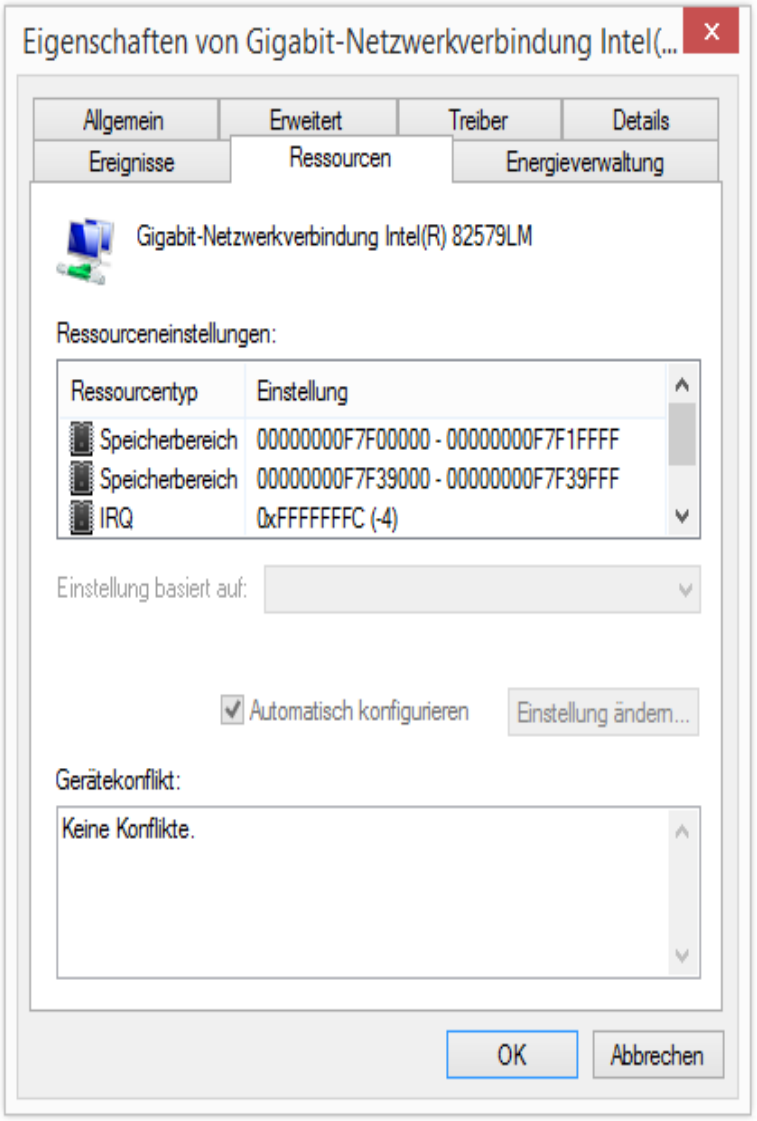
Ein- und Ausgabearchitekturen (2)

- Direkter Speicherzugriff (DMA): Steuereinheit kann über den Bus selbständig auf den Hauptspeicher zugreifen
  - Übertragung von Datenblöcken zwischen Speicher und E/A-Geräten
  - E/A-Bussteuereinheiten (PCI, SCSI, ...) können Datentransfers zwischen angeschlossenen Geräten autonom initiieren und durchführen



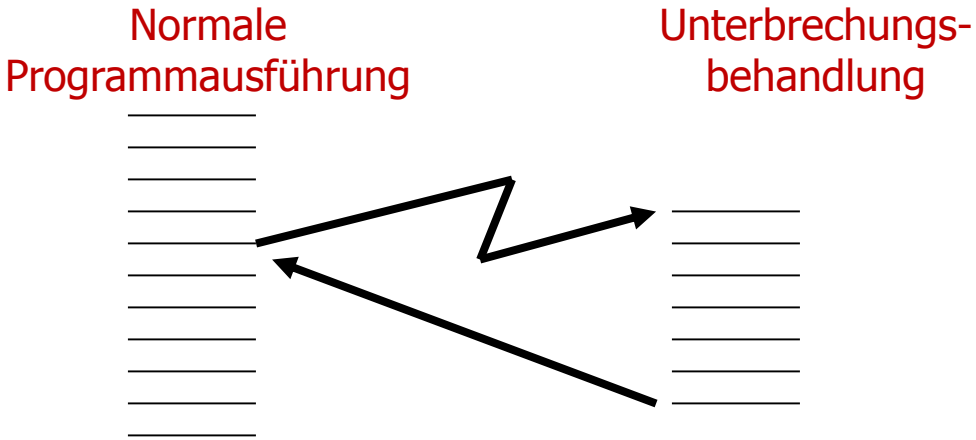
Reaktion

- Information der CPU nach Ende der E/A-Operation
  1. Polling: CPU fragt gelegentlich das Statusregister der Steuereinheit ab (In den meisten Fällen zu ineffizient.)
  2. Unterbrechung / Interrupt: Spezielles Signal informiert die CPU über das Ende der Übertragung Standardmethode



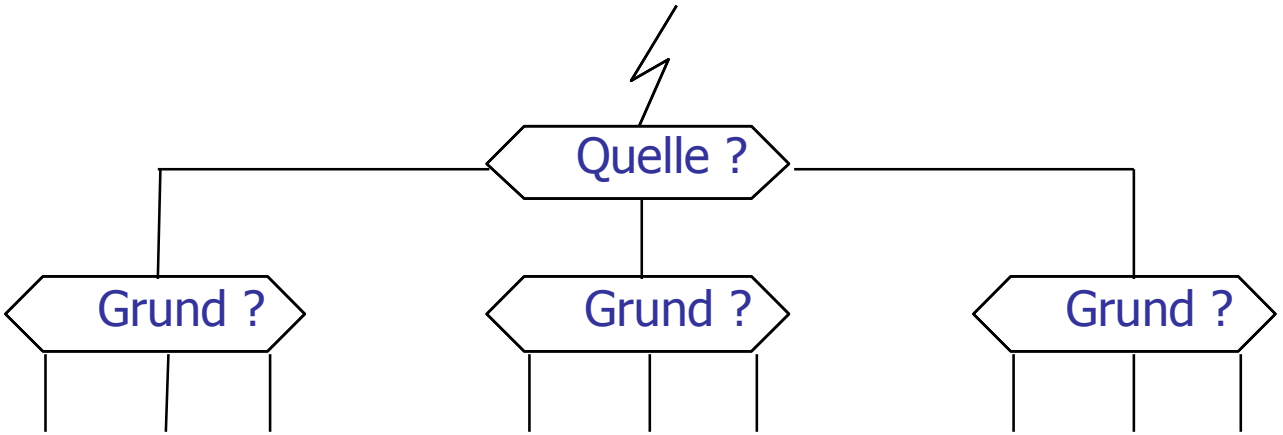
Unterbrechungen (Interrupts)

- Der Bus verfügt über (mindestens) eine Unterbrechungsleitung
  - Prüfung nach jedem Befehl der CPU, ob an dieser Leitung ein Signal (Spannung) anliegt
  - Falls ja
    - Sofortiger Sprung in eine Prozedur zur Auswertung der Unterbrechung
    - Abhängig von Auswertung werden die erforderlichen Aktionen durchführt oder veranlasst
  - Falls nein $\Rightarrow$ nächster Befehl wird bearbeitet



Unterbrechungsanalyse

- Unterbrechungssignal liegt vor
- Analyse mit Ziel herauszufinden
  - wer (welches Gerät) die Unterbrechung verursacht hat (Quelle),
  - warum die Unterbrechung ausgelöst wurde (z.B. Ende der Übertragung, Fehler).
- Struktur der Unterbrechungsbehandlung



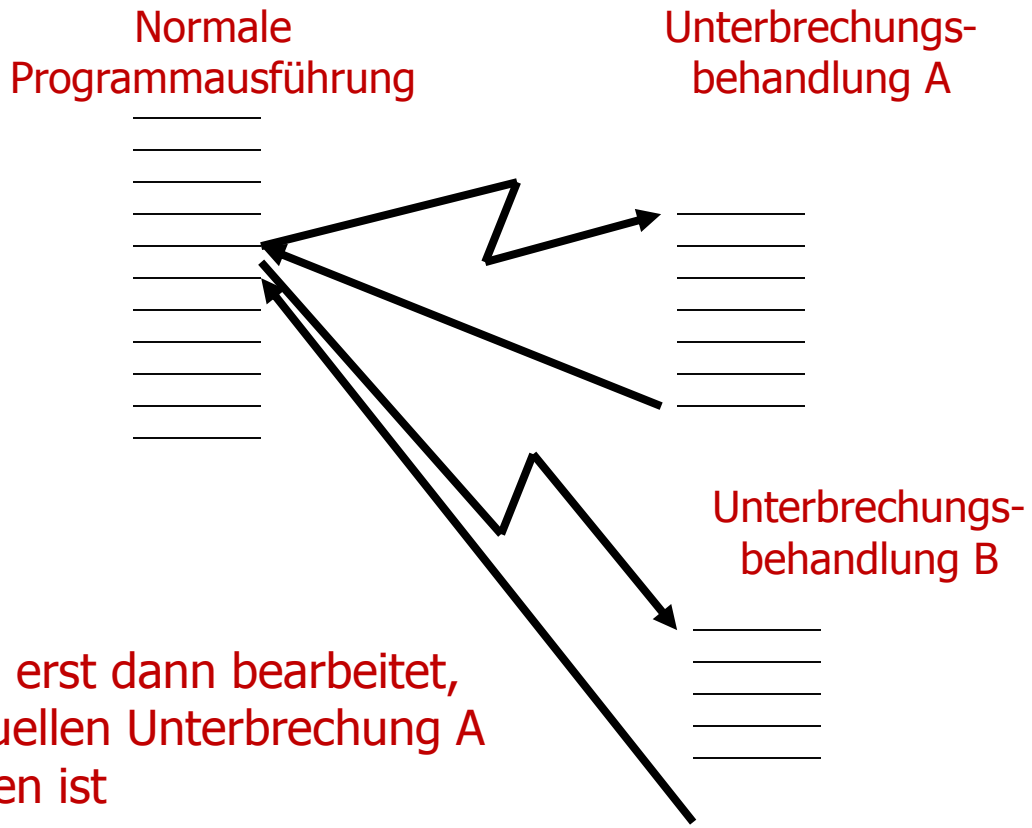
Unterbrechungsbehandlung

- Eine Unterbrechung kann zu jedem Zeitpunkt und in jeder Situation auftreten
  - Knifflig: Unterbrechung während einer Unterbrechungsbehandlung!

- Abarbeitung der Unterbrechungen
  1. Sequentielle Bearbeitung (in Auftrittsreihenfolge)
  2. Geschachtelte Bearbeitung (nested interrupt processing)

Unterbrechungen sequentiell

- Verbieten weiterer Unterbrechungen während der Unterbrechungs-behandlung (Unterbrechungssperre setzen, disable interrupt).
- Das Verbot kann auf bestimmte Unterbrechungstypen beschränkt werden (Maskierung)

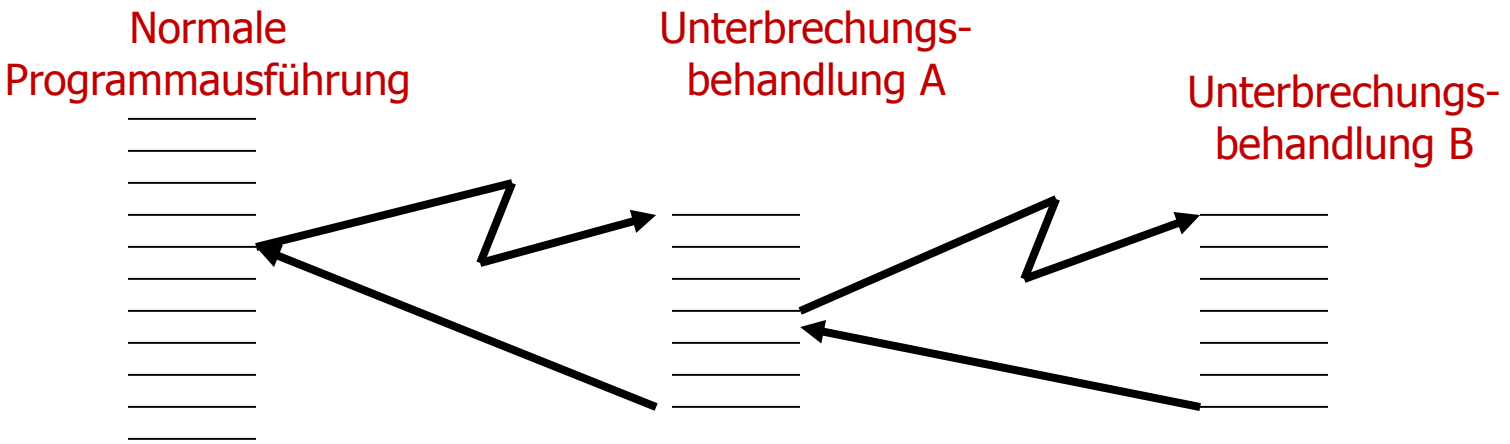


Zweite Unterbrechung B wird erst dann bearbeitet, wenn die Bearbeitung der aktuellen Unterbrechung A abgeschlossen ist

Geschachtelte Interrupts

Unterbrechungen geschachtelt

- Klassifikation von Unterbrechungen in Prioritätsklassen (statisch)
  - ⇒ Unterbrechungen höherer Priorität dürfen die Bearbeitung von Unterbrechungen geringerer Priorität unterbrechen



1.4 Parallele Architekturen

Problem: schwere (Berechnungs-)Aufgabe
-> Lösung: Hochleistungsrechner (aber sehr teuer!)

Hochleistungsrechner: Bis zu Millionen Kerne
-> Alternative Lösung: Parallelisierung

- Operationsprinzip
  - Gleichzeitige Ausführung von Befehlen
  - Sequentielle Verarbeitung lediglich durch Beschränkungen des Algorithmus bedingt

- Arten des Parallelismus

Bsp. Threads

  - Implizit: die Möglichkeit der Parallelverarbeitung ist nicht a priori bekannt
    - ⇒Datenabhängigkeitsanalyse ermittelt die parallelen und sequentiellen Teilschritte des Algorithmus zur Laufzeit

Compiler / Betriebssystem erkennt aus dem Code, welche Datenstrukturen / Bereiche unabhängig sind und parallelisiert diese

  - Explizit: die Möglichkeit der Parallelverarbeitung wird a priori festgelegt
    - ⇒Einsatz von geeigneten Datentypen bzw. Datenstrukturen wie z.B. Vektoren bei Programmerstellung

Programm wird für parallele Maschine geschrieben, bzw. so, dass möglichst große Teile parallel ausgeführt werden können: Aufgaben sind nicht voneinandere abhängig und greifen nicht auf die gleichen Daten zu

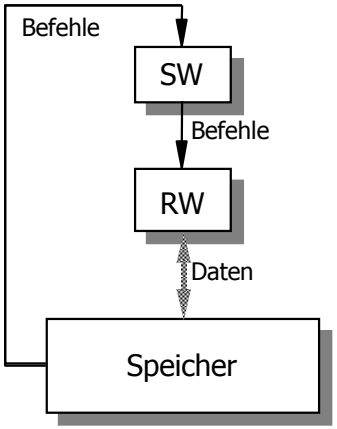
Klassifikation von Rechnerarchitekturen

- Grobklassifikation nach Flynn: Unterscheidung nach der Anzahl von Befehls- und Datenströmen

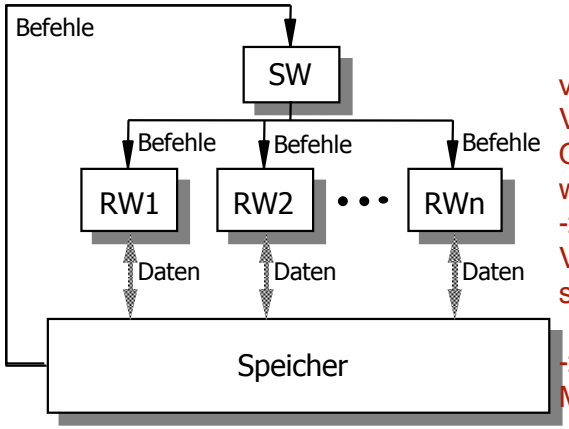
	SD (Single Data)	MD (Multiple Data)
SI (Single Instruction)	ein Befehl auf einen Datensatz angewendet SISD konventionelle von-Neumann-Rechner	SIMD Vektorrechner, Feldrechner
MI (Multiple Instruction)	MISD Datenflussmaschinen Pipelining: die gleichen Daten werden durch verschiedene Prozesse gereicht	MIMD Multiprozessorsysteme, Parallelrechner Verteilte Systeme

"echte Parallelität" - viele Prozesse und Datenquellen gleichzeitig -> schnellste Variante

Flynn‘sches Klassifikationsschema



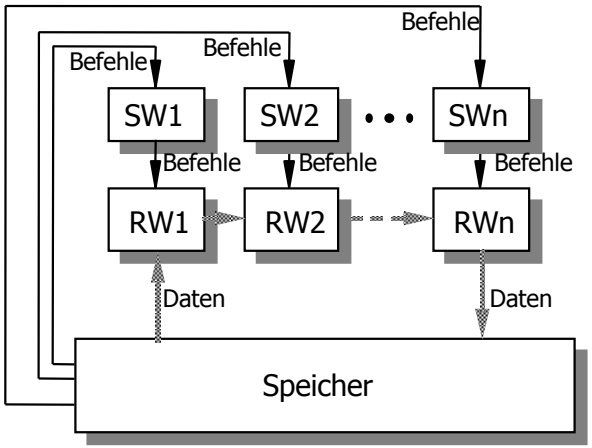
SISD



SIMD

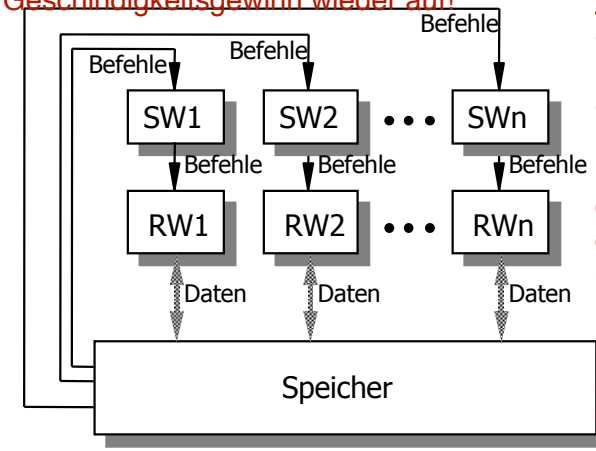
viele Datensätze, z.B. Vektoren, die von der selben Operation verarbeitet werden (z.B. Addition -> schnell für die Verarbeitung gleicher, spezieller Daten

-> Bsp: Vektorenrechner in MatLab für Simulationen



MISD

größtes Problem: Übergänge zwischen Bereichen! falls benachbarte Daten nötig sind -> viele Kommunikations-vorgänge, frisst Geschindigkeitsgewinn wieder auf!



MIMD

Bsp. Bild rendern mit 16 CPUs: Aufteilen in 16 Quadranten wobei jeder von einem anderen Kern bearbeitet wird -> schnellere Bearbeitung (allerdings bestimmt die langsamste CPU die Gesamtgeschwindigkeit -> manche Bereiche sind schwerer; zusätzlich Overhead durch Zusammensetzen (lohnt nur für große Bilder!)

Pipeline-Rechner: Daten werden von verschiedenen Befehlen sequentiell bearbeitet und am Ende wieder gespeichert -> Beschleunigung z.B. einzelner Maschinenbefehle

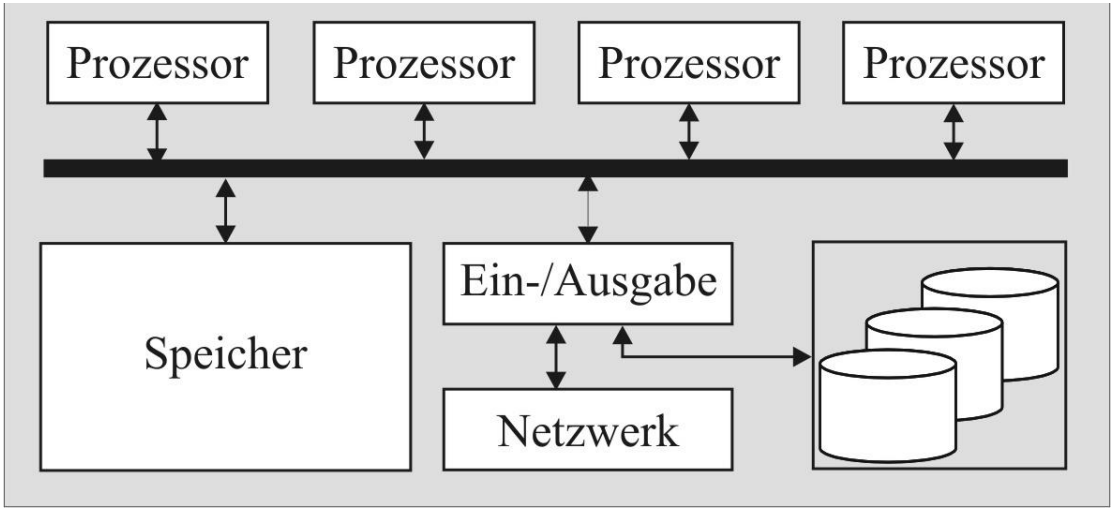
Klassifikation von MIMD Architekturen

- Wichtigstes Merkmal: physikalische Speicheranordnung
  - Gemeinsamer Speicher (*shared memory*) einfach da kein Umdenken nötig
  - Verteilter Speicher (*distributed memory*)

- Die Speicheranordnung beeinflusst weitere Merkmale
  - Programmiermodell: globaler Adressraum oder nachrichtenorientiert (*message passing*) Kommunikation indem Prozesse Nachrichten austauschen (vgl. Email)
  - Kommunikationsstruktur: Speicherkopplung oder Austausch von Nachrichten
  - Synchronisation: gemeinsame Variablen oder synchronisierende Nachrichten
  - Adressraum: global (*gemeinsam*) oder lokal (*privat*)

Architekturen mit gemeinsamen Speicher

- Gleichförmiger Speicherzugriff (uniform memory access, UMA):
  - Die Zugriffsweise ist für jede Kombination (Prozessor, Speichermodul) identisch ⇒ gleichförmige Latenz
- Beispiel: Symmetrische Multiprozessoren (SMP)
  - Mehrere baugleiche und gleichberechtigte Prozessoren
    ⇒Aktuelle Multicore-Prozessoren fallen auch in diese Kategorie
  - Alle anderen Elemente sind aus Sicht des BS einmal vorhanden
  - Physikalisch können die Komponenten aus mehreren Einheiten bestehen (Festplattenarrays)



Nur ein Zugang zu externen Geräten -> dabei können sich die Prozesse in die Quere kommen -> Flaschenhals
Lösung: Cross-Bus-Switches erlauben parallele Zugriffe (teuer!)

Architekturen mit verteiltem Speicher

- Architekturen mit verteiltem Speicher bestehen aus vernetzten Knoten mit jeweils
  - Einem oder mehreren Prozessoren
  - Lokalen Speichermodulen
  - Verbindungsschnittstellen
- Kommunikation und Synchronisation zwischen den Prozessen auf verschiedenen Prozessoren erfolgt durch Austausch von Nachrichten
- Dieses Prinzip kann sowohl für gleichartige als auch für verschiedene Prozessoren realisiert werden

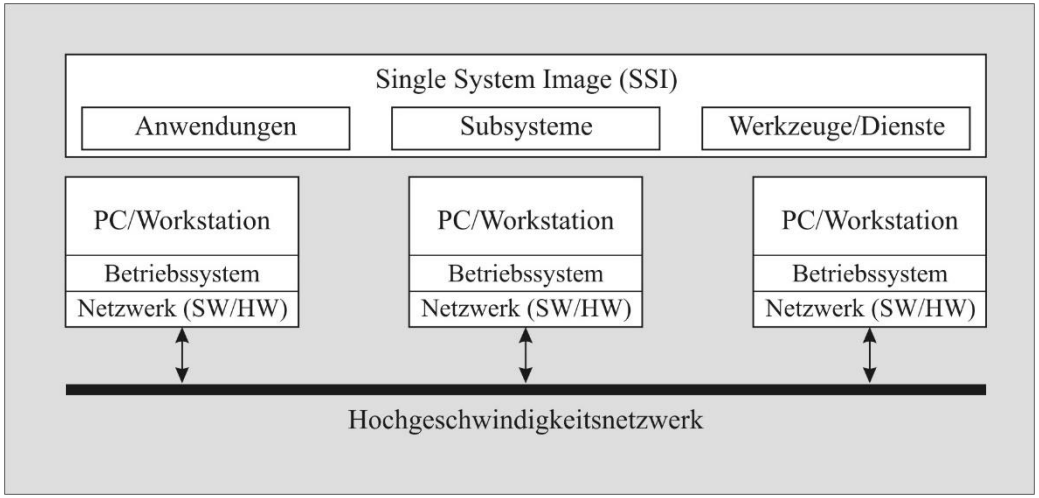
Massiv-parallele Prozessorsysteme (Massively Parallel Processors, MPP)

- Höchstleistungsrechner für Einsatzgebieten wie Wettervorhersage, Medikamentenentwicklung, Simulation usw.
- Typische Merkmale
  - Große Anzahl von Knoten O(10000) bis O(100000) (siehe top500.org)
  - Standard CPUs
  - Lokaler, privater Speicher sowie ein Kommunikationsprozessor
  - Leistungsfähiges, herstellerspezifisches Netzwerk mit großer Bandbreite und niedriger Latenz für die interne Kommunikation
  - Spezielle Knoten für Kontrolle der Ein-/Ausgabe, Administration, Anmeldung, für den Zugriff auf die externen Netzwerke
  - Zentrale Jobverteilung
- Anwendungen werden hauptsächlich mit dem nachrichten-basierten Programmiermodell entwickelt

Cluster

Knoten sind eigene kleine Rechner mit eigenem OS -> einzelne Knoten sind relativ günstig!
Verbindung erfolgt über Software

- Paralleles System, das aus einem Netzwerk von Rechenknoten besteht und als eine einheitliche Computerressource genutzt werden kann
- Rechenknoten
  - Computersystem, das alle Elemente einer Rechnerarchitektur und ein Betriebssystem besitzt und
  - außerhalb des Rechnerverbunds als einzelne Einheit funktionsfähig ist



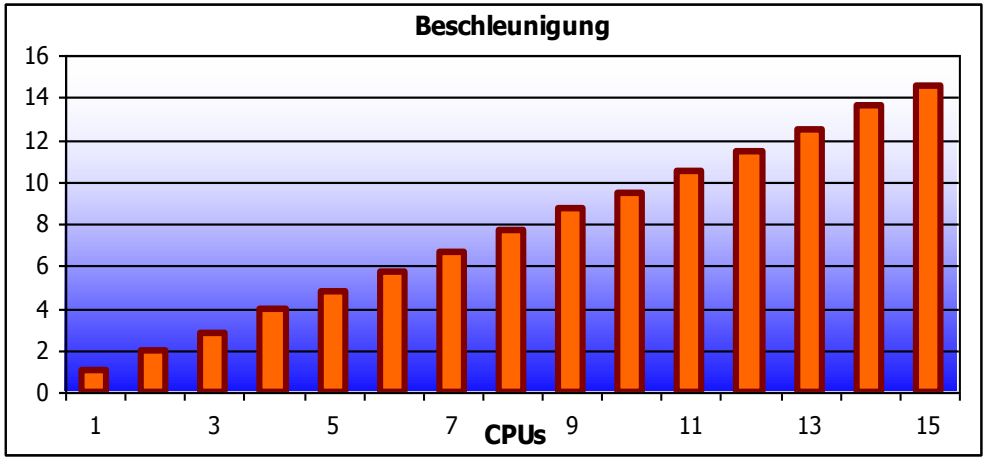
Bewertung paralleler Programme

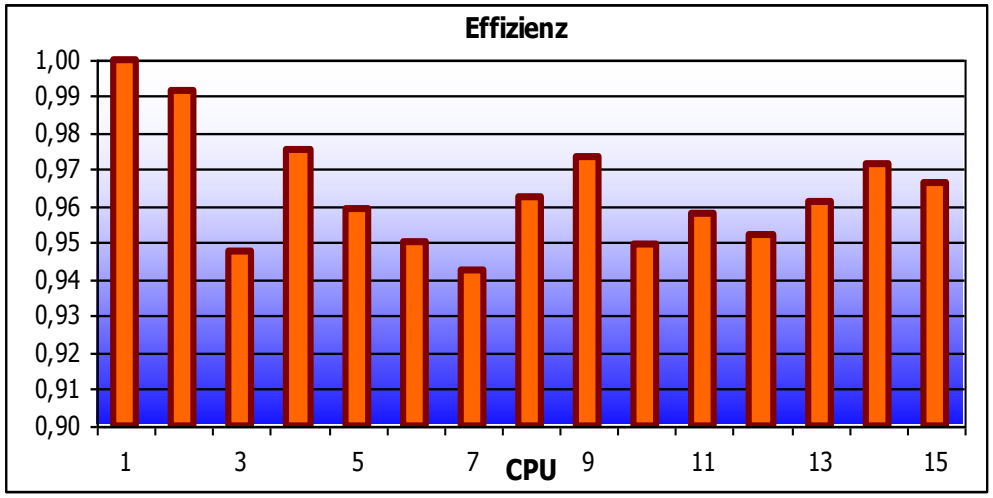
Beschleunigung durch Parallelität (*Speedup*)

$$S_P = \frac{\text{Rechenzeit 1 CPU}}{\text{Rechenzeit p CPUs}} = \frac{T_1}{T_p}$$

$$S_P \in (0, p]$$

Im besten Fall: S = p
-> 4 Kerne bedeuten 4-fache Geschwindigkeit
real nicht zu erwarten





Auslastung (*Effizienz*, *Efficiency*)

$$E_P = \frac{\text{Speedup bei p CPUs}}{\text{p}} = \frac{S_p}{p}$$

$$E_P \in (0,1]$$

1.2 Definition Betriebssystem (BS)

- Betriebssystem (Definition nach DIN 44300)

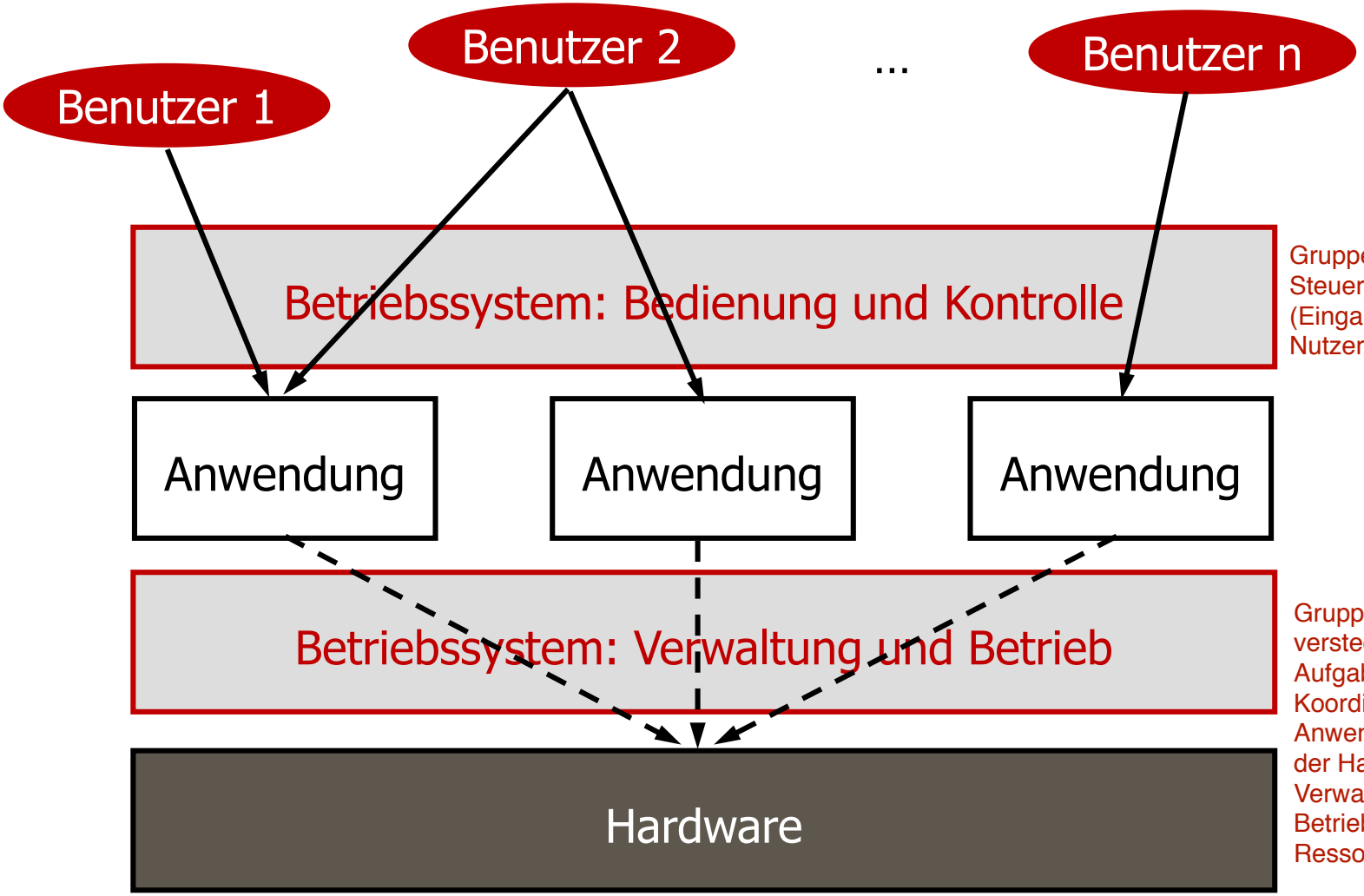
  Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Ausführung von Programmen steuern und überwachen

OS ist Software, die zwischen Anwendungsprogrammen der Nutzer und der Hardware vermittelt (ausreichende Definition)

- BS als Mittler zwischen den Anwendungsprogrammen und der Computerhardware
- Basiskatalog von Funktionen in der Regel für verschiedene BS identisch, Unterschiede in Umfang und Art der Implementierung

Betriebssysteme für Universalrechner

2 Gruppen von Aufgaben des OS



Gruppe 1: GUI, Steuerung (Eingaben der Nutzer)

Gruppe 2: versteckte Aufgaben: Koordination der Anwendungen und der Hardware -> Verwaltung und Betrieb von Ressourcen

Aufgabenbereiche eines Betriebssystems

- Grobe Aufteilung in drei Aufgabenbereiche
  - Bereitstellung von Hilfsmitteln für Benutzerprogramme
  - Vernachlässigung der genauen Benutzerkenntnis von HW-Eigenschaften und spezieller SW-Komponenten, wie z.B. Gerätetreiber
  - Koordination und Vergabe der zur Verfügung stehenden Betriebsmittel an mehrere, gleichzeitig arbeitende Benutzer
- Einzelfunktionen eines Betriebssystems
  - Unterbrechungsverarbeitung (*interrupt handling*)
  - Verteilung (*dispatching*): Prozessumschaltung
  - Betriebsmittelverwaltung (*resource management*): Belegen, Freigeben und Betreiben von Betriebsmitteln, Werkzeuge zur Prozesssynchronisation
  - Programmallokation (*program allocation*): Linken von Teilprogrammen, Laden und Verdrängen von Programmen in/aus dem Hauptspeicher

Einzelfunktionen eines Betriebssystems

- Grundlegende Betriebssystemfunktionen (... Fortsetzung)
  - Dateiverwaltung (*file management*)
    - Organisation des Speicherplatzes in Form von Dateien auf Datenträgern
    - Bereitstellung von Funktionen zur Speicherung, Modifikation und Wiedergewinnung der gespeicherten Informationen
  - Auftragsteuerung (*job control*)
    - Festlegung der Reihenfolge, in der die eingegangenen Aufträge und deren Bestandteile bearbeitet werden sollen
  - Zuverlässigkeit (*reliability*)
    - Funktionen zur Reaktion auf Störungen und Ausfälle der Rechnerhardware sowie auf Fehler in der Software
    - Korrektheit, Robustheit und Toleranz (ständig betriebsbereit unter der Aufrechterhaltung einer Mindestfunktionsfähigkeit)

Mechanismen und Methoden (Policies)

wichtig

- Wichtige Unterscheidung zwischen Mechanismen und Policies
  - Mechanismus: Wie wird eine Aufgabe prinzipiell gelöst? Hier in der Vorlesung behandelt
  - Policy: Welche Vorgaben/Parameter werden im konkreten Fall eingesetzt?

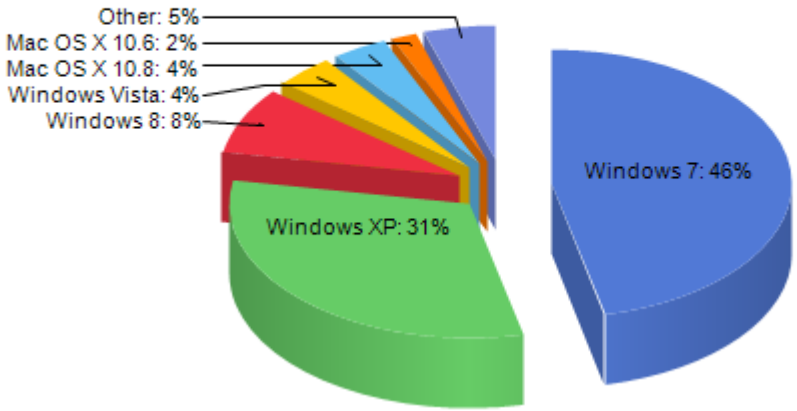
Mechanismen sind für fast alle Betriebbsysteme gleich, unterschiedlich sind die Policies!

- Beispiel: Zeitscheibenprinzip = Round Robin
  - Existenz eines Timers zur Bereitstellung von Unterbrechungen ⇒ Mechanismus
  - Entscheidung, wie lange die entsprechende Zeit für einzelne Anwendungen / Anwendungsgruppen eingestellt wird ⇒ Policy
- Trennung wichtig für Flexibilität
  - Policies ändern sich im Laufe der Zeit oder bei unterschiedlichen Plattformen ⇒ Falls keine Trennung vorhanden, muss jedes Mal auch der grundlegende Mechanismus geändert werden
  - Wünschenswert: Genereller Mechanismus, so dass eine Policiesveränderung durch Anpassung von Parametern umgesetzt werden kann

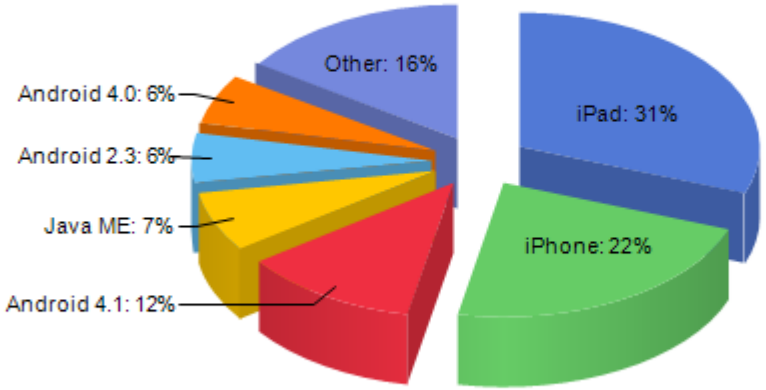
Markt für Betriebssysteme

2014

Desktop



Mobile



©netmarketshare.com

Verwaltung ect:
Nutzung als
“Schreibmaschine”
-> wenig Interesse an
Neuerungen

Zahlen nicht realistisch?

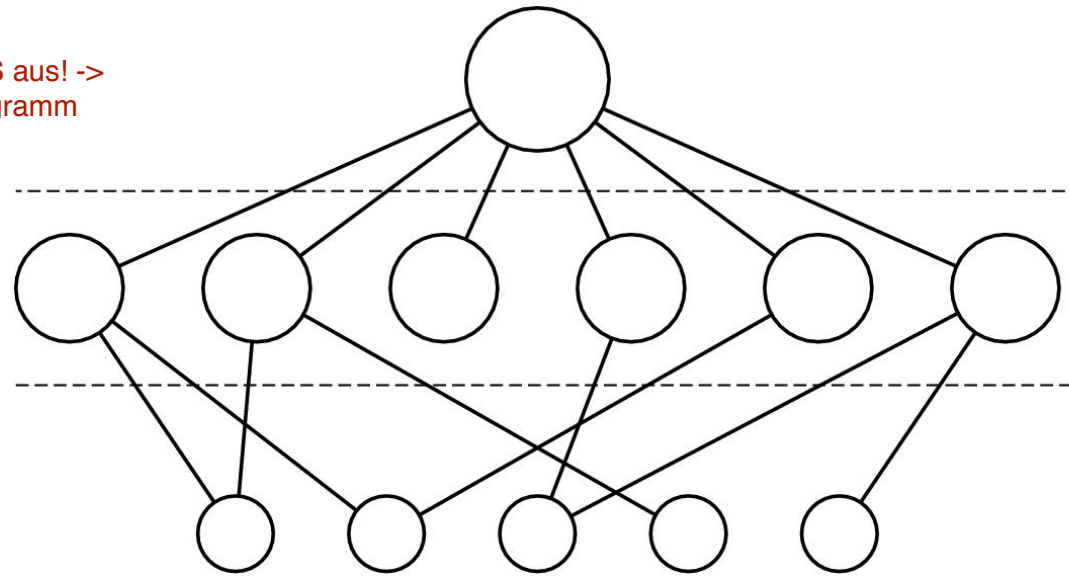
Strukturen der Betriebssysteme

- Häufige Designstrukturen für Betriebssysteme
  - Monolithische Systeme
  - Geschichtete Systeme
  - Virtuelle Maschinen
  - Exokern und
  - Client-Server-Systeme mit Mikrokern

Einfaches Strukturmodell für monolithische Betriebssysteme

- Monolithische Systeme: Häufigste Realisierungsform "Chaos"
  - Große Menge von Funktionen mit wohl-definierten Schnittstellen für Parameter und Ergebnisse ⇒ Alle Funktionen bilden den Objektcode
  - Hauptprogramm: Ruft die Dienstfunktionen auf
  - Dienstfunktionen: Führen die Systemaufrufe durch
  - Hilfsfunktionen: stellen Mechanismen bereit, die von diesen benötigt werden, z.B. Kopieren von Daten aus einem Programm
- Häufiger Wechsel aus Benutzer- in Systemmodus und umgekehrt

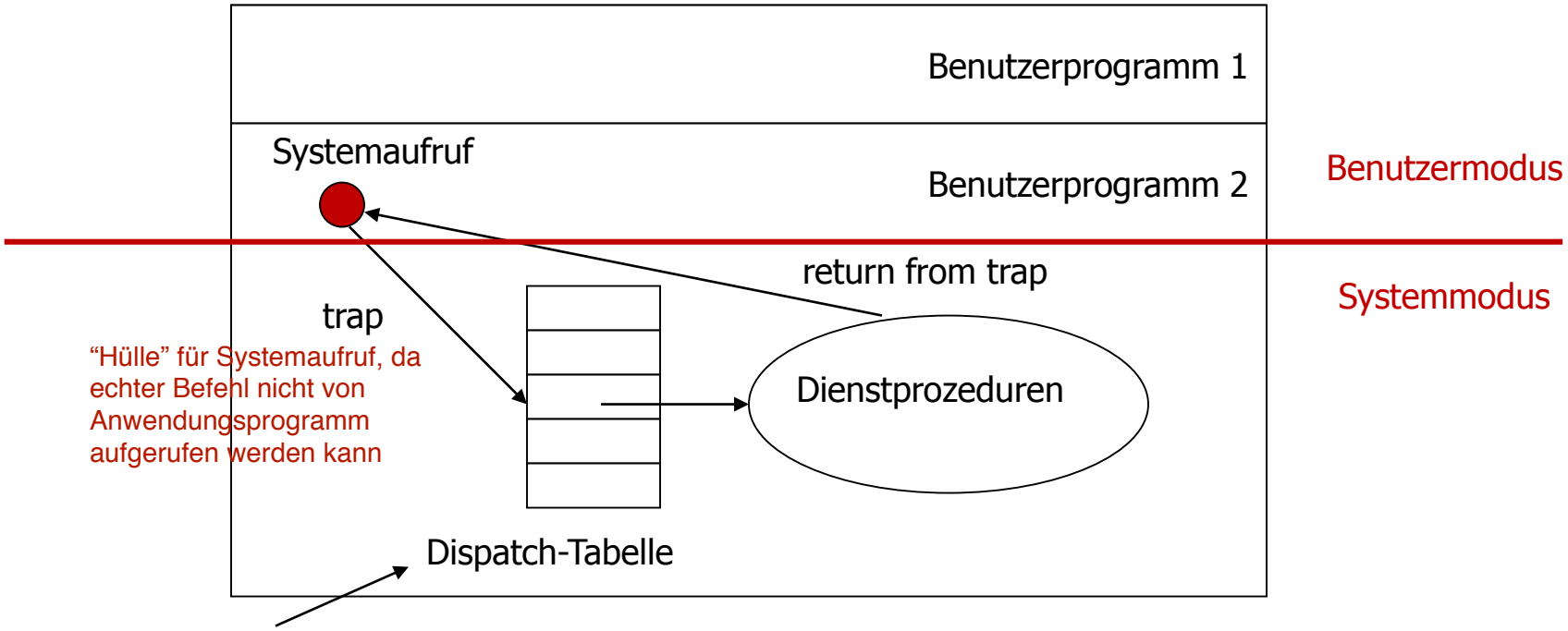
So sehen reale OS aus! -> riesiges Hauptprogramm



Unterschiede: Wie viele Funktionen implementiert Grundaufbau gleich

Abarbeitung eines Systemaufrufs

- Anwenderprogramme werden in Benutzermodus ausgeführt
  - ➢ Bei Kernzugriffen wird der trap-Befehl mit der Kennziffer des auszuführenden Befehls als Parameter aufgerufen
  - ⇒ Wechsel vom Benutzer- in Systemmodus und Befehlsausführung
- Nach Abarbeitung Rückkehr in Benutzermodus



Meist zeigt ein spezielles Register im Prozessor auf den Beginn der Tabelle

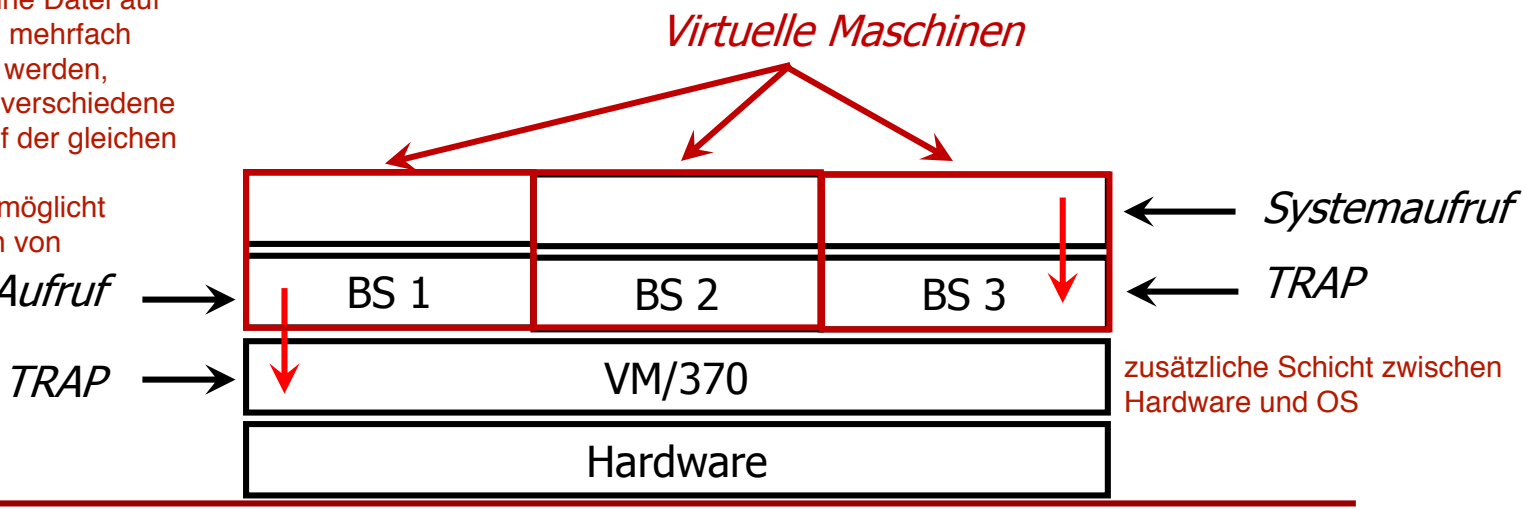
Virtuelle Maschinen

Bsp: Web Server: mehrere Server laufen auf der selben Hardware

-> LSF hat an den ersten Semestertagen viele Kerne reserviert, später dann weniger nötig

- Nachbildung der zugrunde liegenden Hardware
- Erstes System IBM VM/370 mit zwei Komponenten
  1. Monitor der virtuellen Maschine: Ausführung auf der realen Hardware, Realisierung von Mehrprogrammbetrieb
  2. Mehrere virtuelle Maschinen mit exakter Kopien der HW mit Kern- und Benutzermodus und der Ein- und Ausgabe und Unterbrechungen
- Systemaufrufe werden von der virtuellen Maschine abgefangen
- Systemaufrufausführung auf der realen HW durch Monitor

Vorteil: VM ist nur eine Datei auf der Festplatte, kann mehrfach zugleich ausgeführt werden, außerdem mehrere verschiedene Betriebbsysteme auf der gleichen Hardware nötig
-> Virtualisierung ermöglicht besseres Ausnutzen von Ressourcen



zusätzliche Schicht zwischen Hardware und OS

Client-Server-Modell und Mikrokern

je größer der Kernel, desto mehr potentielle Sicherheitslücken

- Idee eines minimalen Kerns (Mikrokern) durch
  - Auslagerung von BS-Funktionen als Server-Dienste
  - Kommunikation zwischen Funktionen im Kern und Benutzerraum nach Client/Server-Muster durch Austausch von Nachrichten

nur absolut notwenige Funktionen im Kernel (allerdings nicht klar, welche das sind)

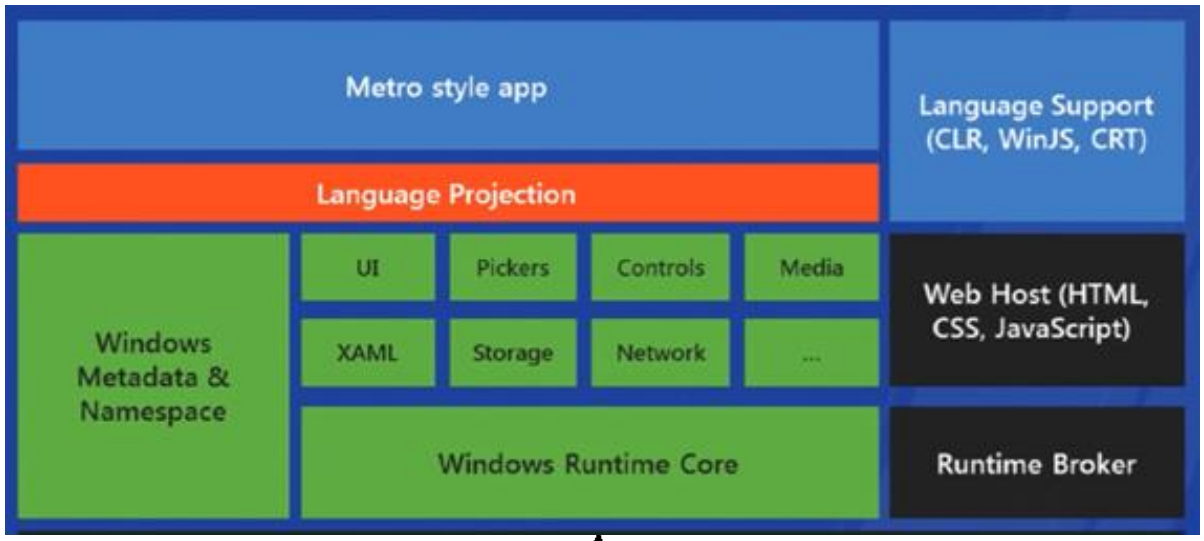
- Durch Aufteilung des BS entstehende Einheiten wie Dateiserver, Prozessserver, Terminalserver, …
  - Effiziente Implementierung des Kerns einfacher zu realisieren
  - Serverdienste sind Prozesse im Benutzermodus ohne direkten Hardwarezugriff $\Rightarrow$ bei Fehlfunktionen stürzen einzelne Dienste ab, das Gesamtsystem ist – eingeschränkt – funktionsfähig
- Probleme
  - Einige Dienste lassen sich nur im Kernmodus realisieren
  - Geeignete Trennung von Mechanismen (Kern) und Strategien oder Policies (Server im Benutzermodus) notwendig

1.3 Fallstudie: Windows

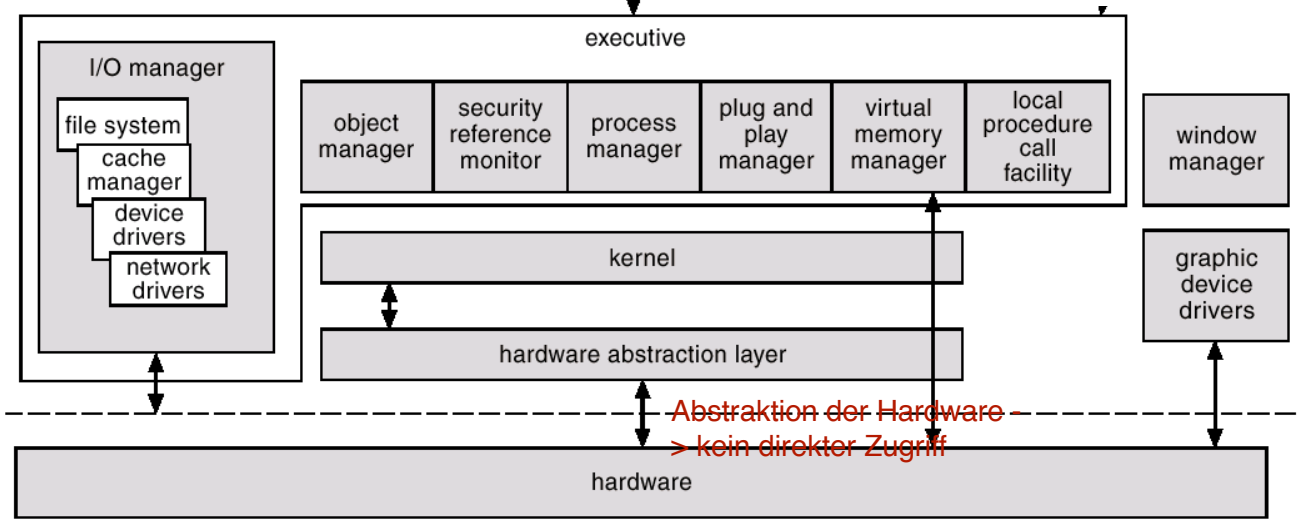
- Ursprünglich auf Betrieb von mehreren Teilsystemen ausgelegt (z.B. Unix, OS/2, Windows)
- Betriebssystemkern oft als Mikrokern bezeichnet, enthält allerdings Systemkomponenten, die nach Mikrokerndefinition nicht notwendigerweise integriert werden müssen
- Aufteilung der Systemkomponenten in Schichten
  - Hardwareabstraktionsschicht (HAL)
  - Kern mit zentralen Aufgaben
  - Executive
  - Gerätetreiber

Windows Architektur

www.buildwindows.com



Benutzermodus: geschützte Subsysteme



Kernmodus: privilegierte Subsysteme

Treiber außerhalb des Kerneis -> Portabilität, Stabilität

Früher Treiber im Kernel -> Problem, da Treiber von Geräteherstellern veröffentlicht werden

Fallstudie: Unix

- Unix/Linux: Schichten-basiertes System mit monolithischem Kern
- Kernmodus
  - Alle Befehle mit Zugriff auf Hardware
  - Kritische Dienste wie Scheduler, Module-Loader, Prozessmanagement, Semaphore, Tabelle mit Systemaufrufen, …
- Struktur eines typischen UNIX-Kerns am Beispiel 4.4BSD-Kern

<table>
<tr><td colspan="3">Systemaufrufe</td><td colspan="4">Unterbrechungen</td></tr>
<tr><td colspan="2">Terminal-Behandlung</td><td>Sockets</td><td>Datei-benennung</td><td>Map-ping</td><td>Seiten-fehler</td><td rowspan="2">Signal-Behand-lung</td><td rowspan="2">Prozess-erzeugung und been-digung</td></tr>
<tr><td rowspan="2">Rohes Terminal</td><td>Cooked Term.</td><td>Netzwerkprotokolle</td><td>Datei-systeme</td><td colspan="2">Virtueller Speicher</td></tr>
<tr><td>Line-Verwalt.</td><td>Routing</td><td>Puffer-Cache</td><td>Seiten-Cache</td><td colspan="2">Prozess-Scheduling</td></tr>
<tr><td colspan="2">Zeichengeräte</td><td>Netzwerk-Gerätetreiber</td><td colspan="2">Festplatten-Gerätetreiber</td><td colspan="2">Prozess-Kernzuteilung</td></tr>
<tr><td colspan="7">Hardware</td></tr>
</table>

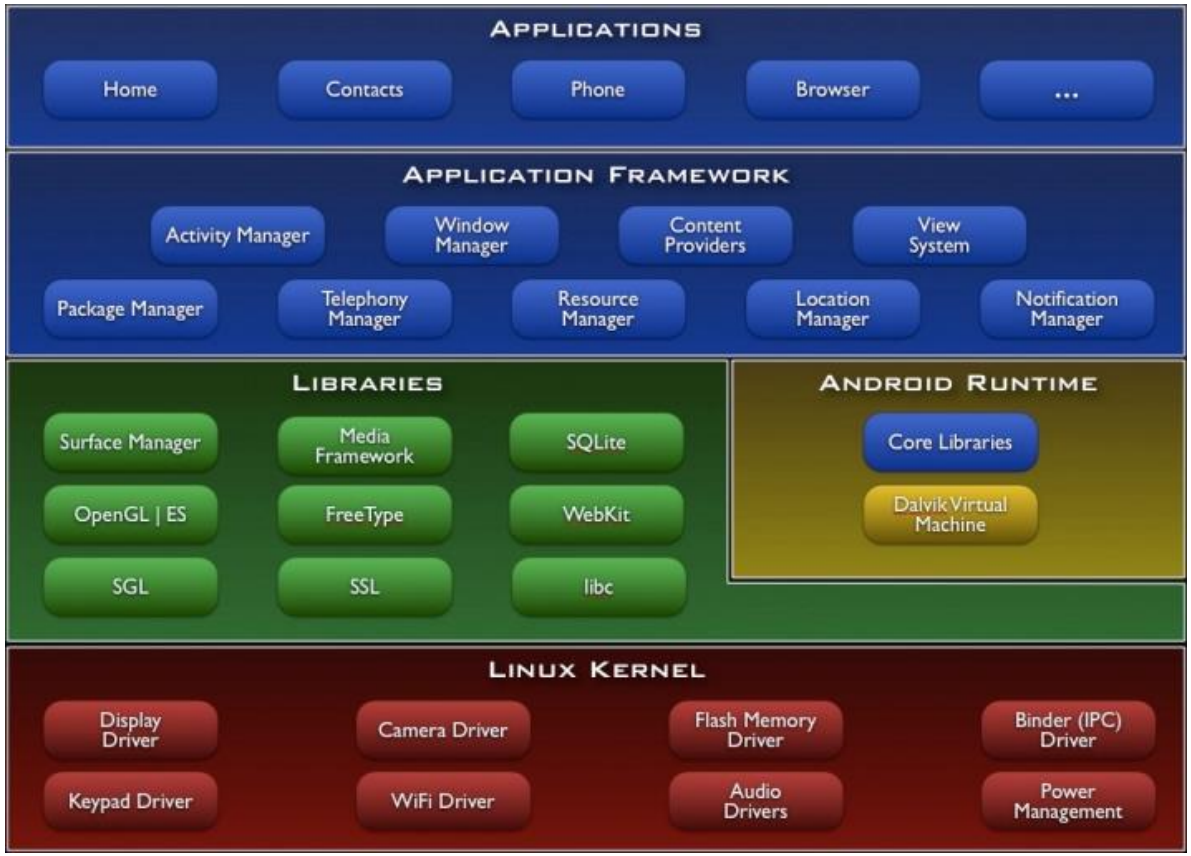
Fallstudie: Android

- Betriebssystem und Middleware für mobile Geräte wie Smart Phones und Netbooks entwickelt von Open Handset Alliance
  - Entstanden auf Basis des Linux-Kernel 2.6
  - Freie und quelloffene Software
  - SDK verfügbar zur Entwicklung von Anwendungen für Android-Plattformen in Java
- Historie
  - Android = Unternehmen zur Entwicklung von standortbezogenen Diensten für mobile Geräte, gegründet 2003
  - Aufkauf durch Google im Sommer 2005
  - Gründung der Open Handset Alliance ab Ende 2007 u.a. mit China Mobile, NTT DoCoMo, T-Mobile, Telecom Italia, Telefónica, eBay, Google, Broadcom, Intel, Nvidia, Qualcomm, HTC, LG, Motorola, Samsung, Vodafone, Acer, Garmin, Huawei, Sony Ericsson, Toshiba, … (www.openhandsetalliance.com/)

Android Basis

Mobile OS basieren auf Desktop OS in abgespeckter Version (nicht benötigte Funktionen weggelassen)

- Android bietet Komponenten für
  - Sicherheit, Speicher/Prozessmanagement, Netzwerk, Gerätetreiber für GSM, Bluetooth, EDGE, 3G, Wlan, Camera, GPS, Kompass, und Beschleunigungssensoren
  - Laufzeitumgebung = Dalvik Virtual Machine
  - ⇒ Keine direkte Verwendung der Java-Bytecodes, aber Verwendung vieler Java-Werkzeuge

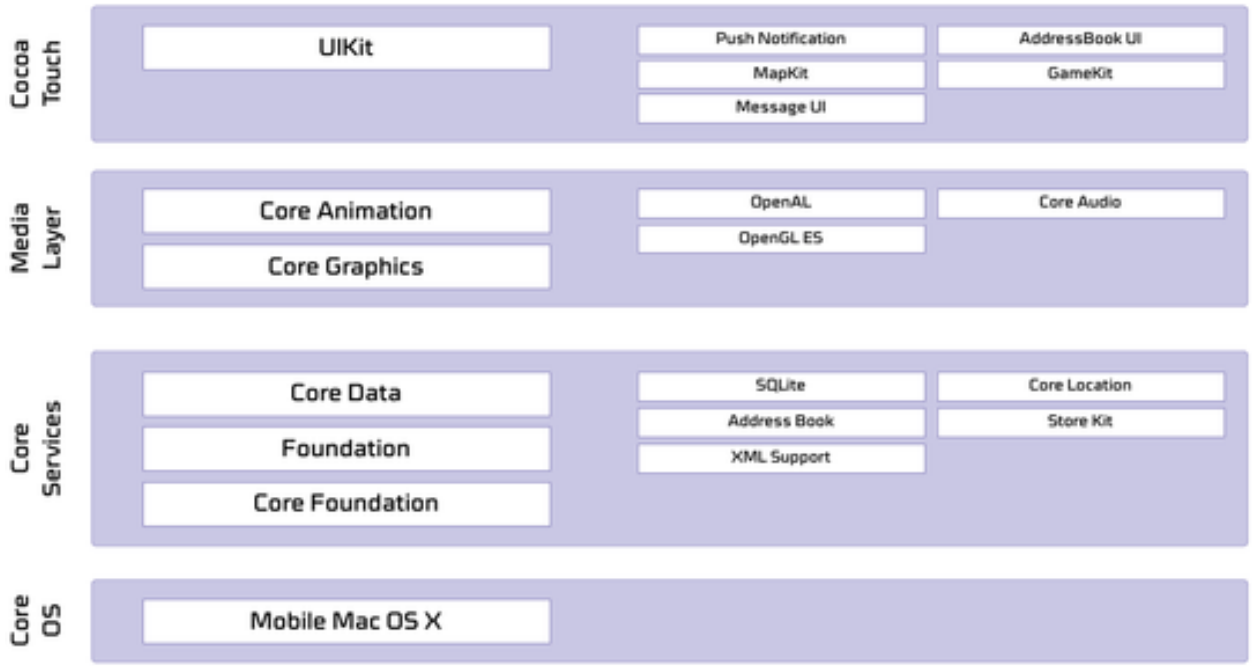


http://developer.android.com/guide/

Schichtenmodell

Fallstudie: iOS

- Komponenten
  - Cocoa Touch: Multi-Touch, Core Motion, Localization, Controls, Alerts, Web View, Map Kit, Image Picker, Camera
  - Media: Audio, Video Playback, JPEG, PNG, TIFF, PDF, Animation, OpenGL
  - Core Services: Collections, Address Book, Networking, File Access, SQLite, Core Location, Net Services, Threading, …
  - Core OS: OSX Kernel Mach 3.0, BSD Sockets, Security, Power Management, Certificates, File System



1.4 Geschichte der Betriebssysteme

nicht klausurrelevant

- Entwicklung der Betriebssysteme ist eng mit der Entwicklung der Rechnerarchitekturen verbunden
- Erste Generation (1945-1955) keine Betriebssysteme
  - Programmierung in Maschinensprache
  - Betriebssysteme und Programmiersprachen noch unbekannt
- Zweite Generation (1955-1965)
  - Transistoren führen zur erhöhten Zuverlässigkeit
  - Unterscheidung: Entwickler, Hersteller, Operateur und Programmierer
  - Programme auf Lochkarten
  - Stapelverarbeitung als Hauptbetriebsart
    - Manuell: Operateur lädt neuen Job, wenn der alte abgearbeitet wurde
    - Automatisch: Kleiner, billiger Rechner liest neue Jobs und speichert diese auf Band. Band dient als Eingabe für den Mainframe

Dritte Generation (1965-1980)

- Einführung von integrierten Schaltungen
- Existenz zweier, zueinander inkompatiblen Produktlinien
  - Wortorientierte Rechner (numerische Berechnungen in Wissenschaft und Industrie)
  - Zeichenorientierte Rechner (Sortieren und Ausdrucken von Bändern in Banken und Versicherungen)

  ⇒ Portabilitätsprobleme bei Erweiterung des Rechnerpools
- IBM System/360

  IBM wollte die beiden Linien in einem Produkt vereinen -> dauerte ewig und wurde Chaos
  - Serie von Software-kompatiblen Rechnern mit derselben Architektur und identischem Instruktionssatz
- Widersprüchliche Anforderungen führen zu großem, komplexem und fehlerbehaftetem Betriebssystem

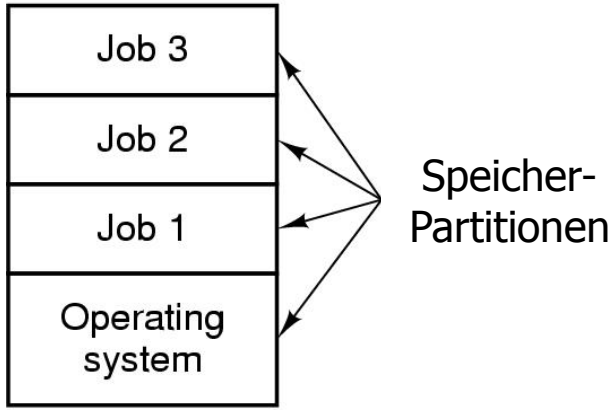
  ⇒ Einsatz hoher Programmiersprachen zur Implementierung von BS

  ⇒ Geburtsstunde von Software-Engineering

  "erst nachdenken, dann programmieren"

Dritte Generation: Einführung wichtiger Schlüsseltechniken

- Multiprogramming
  - Mehrere Jobs komplett im Speicher
  - Auf E/A wartende Jobs werden blockiert
  - Hardwareschutz der Jobpartitionen im Speicher



- Time Sharing: Einführung des Zeitscheibenkonzepts
- Echte Parallelität durch E/A-Prozessoren
- Spooling (*Simultaneous Peripheral Operation On Line*)
  - Jobs wurden von Lochkarten eingelesen und auf Platten zur Ausführung – sobald möglich – bereit gehalten
- Virtualisierung des Prozessors in Form eines *Prozesses*
- Einführung des Virtueller Speichers
- Prozesse werden als interne Strukturierungsmittel auch für Betriebssysteme eingeführt

Einführung von UNIX

- UNIX entsteht nach dem Prinzip „simple is beautiful“ <-> IBM
- MULTICS (*MULTI-plexed Information and Computing System*)
  - System entwickelt von Bell Labs, MIT und General Electric
  - Grundidee entstammt der Stromversorgung: bei Bedarf steckt man einen Stecker in die Dose und bezieht Elektrizität
  - Analog mit Rechenleistung $\Rightarrow$ MULTICS sollte ausreichend Rechenleistung für alle Einwohner von Boston bieten
  - Bell Labs und General Electrics verlassen das Projekt, das von MIT zum Teilerfolg weitergeführt wurde
- MULTICS-Entwickler Ken Thompson suchte sich eine neue Beschäftigung und schrieb UNICS für PDP-7 Kleinrechner
- Bald unterstützen ganze Abteilungen von Bell Labs Thompsons Projekt und entwickelten Portierungen für diverse PDP-Geräte

Real programmers ☺

- Dennis Richie (stehend) und Ken Thompson vor einem PDP-11 System
  Quelle: http://www.psych.usyd.edu.au/pdp-11/real_programmers.html

Einführung von UNIX (2)

- Portierung von UNIX auf neue Maschinen ist wegen der Assemblersprache sehr aufwendig
  - ⇒ Entwicklung in einer hohen Programmiersprache notwendig
- Vorhandene Sprachen wie B (vereinfacht von BCPL) reichten nicht aus
  - ➢ Ritchie entwickelt C und schreibt mit Thompson UNIX in C neu
  - ⇒ C als Standard für Systemprogrammierung entsteht
- AT&T durfte nicht ins Computergeschäft einsteigen
  - ⇒ UNIX wurde an Universitäten inklusive Quellcodes ausgeliefert
  - ➢ Stand: 8200 Zeilen C-Code und 900 Zeilen Assembler-Code
  - ➢ Unternehmen lizenzierten den Quellcode, um eigene Versionen von UNIX zu entwickeln (z.B. XENIX vom Startup Microsoft ☺)
- Berkeley UNIX: Berkeley Software Distribution (BSD)
  - ➢ SUN, DEC usw. bauten ihre UNIX-Versionen auf Grundlage von BSD und nicht der „offiziellen" Version UNIX V von AT&T auf

Einführung von UNIX (3)

- Standardisierungsprojekt POSIX
  - Systemaufrufe aus der Schnittmenge von System V und BSD
  - Verabschiedet als IEEE Standard 1003.1
  - Aber erneute Spaltung in OSF (IBM, DEC, HP, …) und UI (AT&T, …) (siehe auch http://www.levenez.com/unix/)

- Tanenbaum entwickelt MINIX als „UNIX für Ausbildungszwecke"
  - Mikrokern umfasst minimale Funktionalität
  - 1600 Zeilen C-Code und 800 Zeilen Assembler
  - Schnell wachsende Entwicklergemeinschaft

- Linus Torvalds stellt 1991 Linux 0.0.1 zur Verfügung aus Basis von MINIX
  - Monolithischer Ansatz, zunächst auf Intel 386 beschränkt
  - Version 1.0 erscheint 1994 mit 165000 Zeilen Code
  - Kompatibilität zum UNIX-Standard
  - Version 2.0 erscheint 1996 mit 470000 Zeilen Code
  - Linux ist frei verfügbar (GNU Public License)

Alles beginnt mit einem ersten Schritt ☺

> Summary: small poll for my new operating system
> Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.Fi>
> Date: 25 Aug 91 20:57:08 GMT
> Organization: University of Helsinki
>
> Hello everybody out there using minix –
> I'm doing a (free) operating system (just a hobby, won't be big and
> professional like gnu) for 386(486) AT clones. This have been brewing since
> april, and is starting to get ready. I'd like any feedback on things people
> like/dislike in minix, as my OS resembles it somewhat (same physical layout of
> the file system (due to practical reasons) among other things).
>
> I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
> This implies that I'll get something practical within a few months, and I'd like
> to know what features most people would want. Any suggestions are welcome,
> but I won't promise I'll implement them :-)
> Linus (torvalds@kruuna.helsinki.fi)
> P.S. It is NOT portable (uses 386 task switching etc.), and it probably never will
> support anything other than AT-harddisks, as that's all I have :-(.

Vierte Generation (1980-1990)

- Entwicklung von Mikrocomputern und großes Aufkommen von Arbeitsrechnern und PCs
- Massenverbreitung von Betriebssystemen
  - CP/M (*Control Program for Microcomputers*)
    - Plattenbasiertes Betriebssystem für Intel 8080
    - Dominiert den Mikrocomputermarkt ca. 5 Jahre lang
  - MS-DOS (*MicroSoft Disk Operating System*)
    - Basiert auf DOS der Firma Seattle Computer Products
    - Wird mit IBM PC im Paket ausgeliefert ⇒ Vermarktungsstrategie entscheidender Erfolgsfaktor (CP/M wird an Endkunden verkauft)
  - Mac OS: Einführung von GUI und benutzerfreundlichen Interfaces
  - UNIX wird ebenfalls um eine grafische Oberfläche erweitert, die auf dem am MIT entwickelten X-Windows-Konzept basiert

quasi Monopol bei Mikrocomputern

Mac OS: große Verbreitung, wegen günstiger Abgabe an Unis

- Schlüsseltechnologien betreffen Netzwerkbetriebssysteme und verteilte Betriebssysteme

Vierte Generation (2)

- Leistungsfähige Kommunikationsmedien erlauben die Einführung und Nutzung von verteilten Systemen
  - ⇒ Betriebssysteme überwinden Rechnergrenzen: Von der Rechnerkommunikation zum Verbundsystem

- Neue Technologien
  - Einführung von Lightweight Processes (Threads)
  - Aufnahme von Parallelitätskonzepten in BS und Programmiersprachen wie massiv-parallele Verarbeitung, Lastenausgleich, …
  - Multimedia: Unterstützung von Bildern, Audio- und Videoströmen
  - Eingebettete Systeme: Maßgeschneiderte, effiziente, zuverlässige BS, oft Echtzeitanforderungen
  - Interoperabilität: Verteilte Systeme in heterogenen Umgebungen

Bsp. VPNs, Cloud Computing

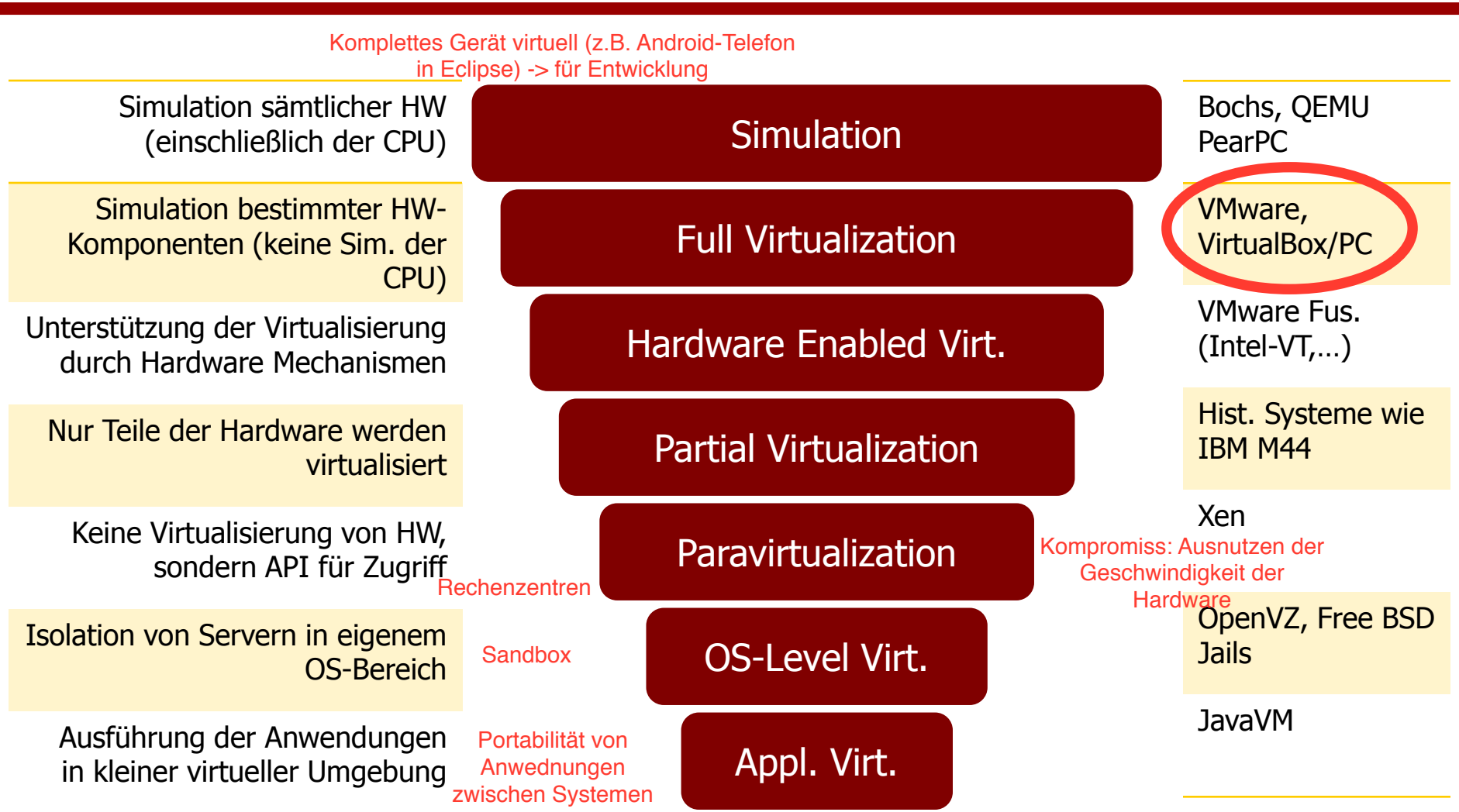
Aktuell: Virtualisierung

- Ziel von Mobile Codes, Grid Computing, Hosting, ...: Ausführung eigener Anwendungen auf gemieteten Rechnerplattformen

keine/wenig eigene Hardware nötig, On-Demand-Nutzung

- Probleme
  - Komplexer Zugang
  - Konfigurationsprobleme: ist alles da, damit die Anwendung tatsächlich laufen kann?
  - Dienstgütegarantien
- Weiterentwicklung: Virtualisierung
  - Nicht einzelne Anwendungen und Daten werden übertragen, sondern die gesamte Laufzeitumgebung
  - Abbildung der Laufzeitumgebung auf realer Hardware
  - Ressourcenteilung zwischen mehreren virtuellen Maschinen
- Virtuelle Server = Systeme die von einander komplett abgekoppelt sind und "nach außen" als eingeständige Server wirken

Virtualisierung



Cloud Computing

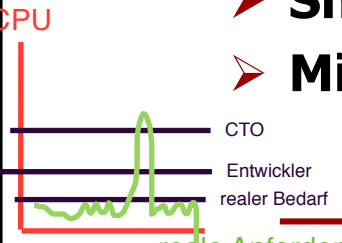
- NIST-Definition
  - Cloud computing is a model for enabling **convenient**, **on-demand** network access to a **shared pool** of **configurable computing resources** (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned** and released with **minimal management effort** or service provider interaction.

(National Institute for Standards and Technology, NIST)

Idee von Amazon: Optimale Nutzung der Server -> nicht benötigte Rechenleistung an andere verkaufen

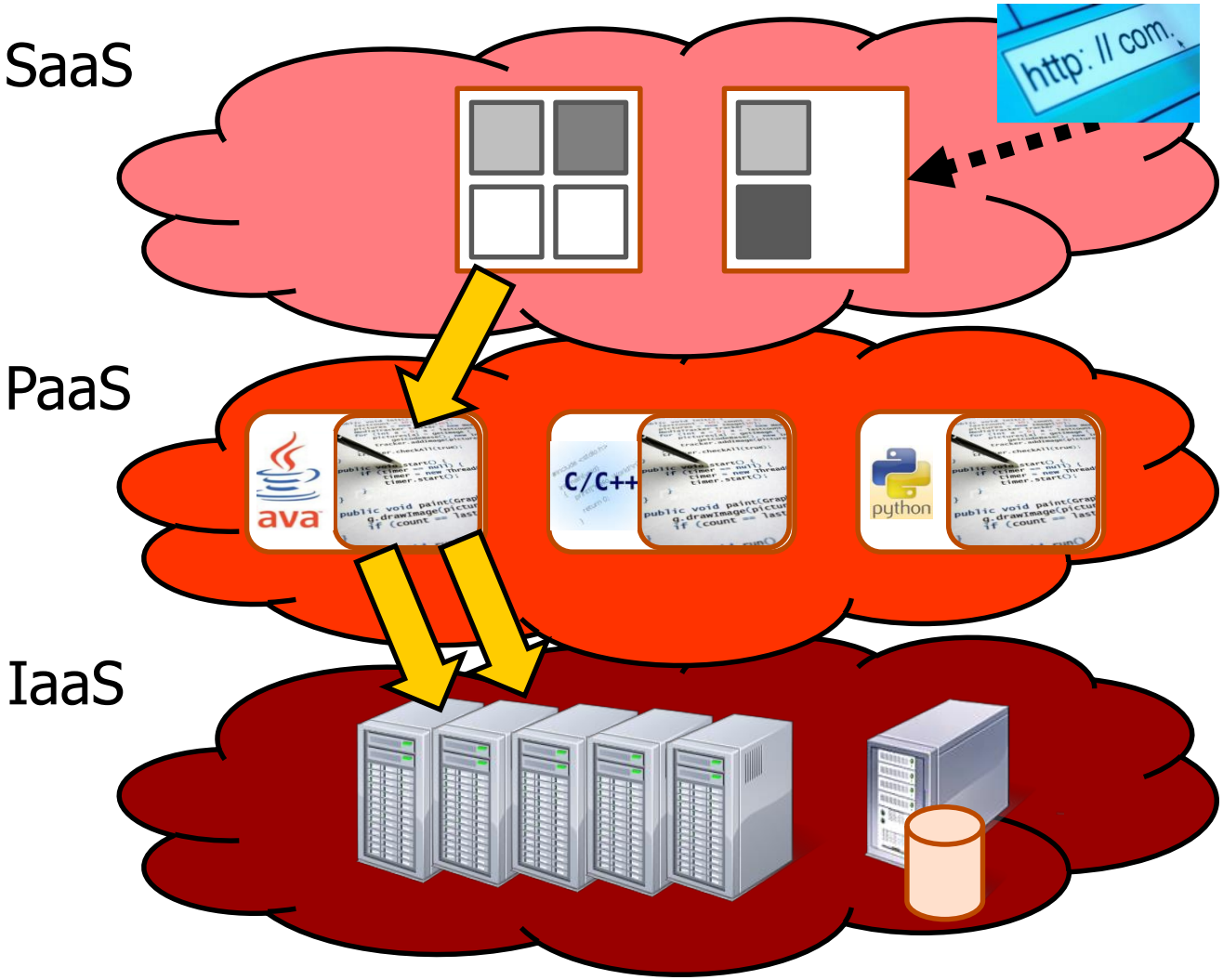
- Zentrale Elemente in dieser Definition
  - **Convenient** = einfache Schnittstelle
  - **On demand, rapidly provisioned** = kurzfristige Bereitstellung
  - **Shared Pool** = Keine dedizierten Ressourcen, Multi-Tenant
  - **Minimal management effort** = Standard-Service des Providers

ohne dass menschliche Beteiligung nötig ist



Problem: Cloud Computing garantiert keine Verfügbarkeit, d.h. laut Service Level Agreement sind auch Ausfälle möglich -> 100% Uptime wird nie angeboten —> nicht für alle Anwednungsfälle geeignet

Schichten des Cloud Computing



Platform as a Service

Infrastructure as a Service

Dreischichtenmodell für Cloud-Dienste

- Aufteilung von Clouddiensten in drei Bereiche
  - Software as a Service (SaaS)
    - Geschäftsanwendungen werden als standardisierte Services von einem Dienstleister bereitgestellt (z.B. MS Windows Live Services)
  - Plattform as a Service (PaaS)
    - Infrastruktur für Anwendungen, die auf Basis von technischen Frameworks angeboten werden
    - Ziel: Entwicklung und Integration von Anwendungskomponenten, wie z.B. Zugriffskontrolle, Workflow-Steuerung, Billing (Google App Engine)
  - Infrastructure as a Service (IaaS)
    - Basisinfrastruktur für Rechen- und Speicheraktivitäten auf virtualisierten Servern sowie Netzwerkinfrastruktur mit hohem Standardisierungsgrad und intelligentem Systemmanagement
    - Einzelne Funktionen können eng verbunden (Orcherstriert) und als integrierter Service angeboten werden (z.B. Amazon EC2).