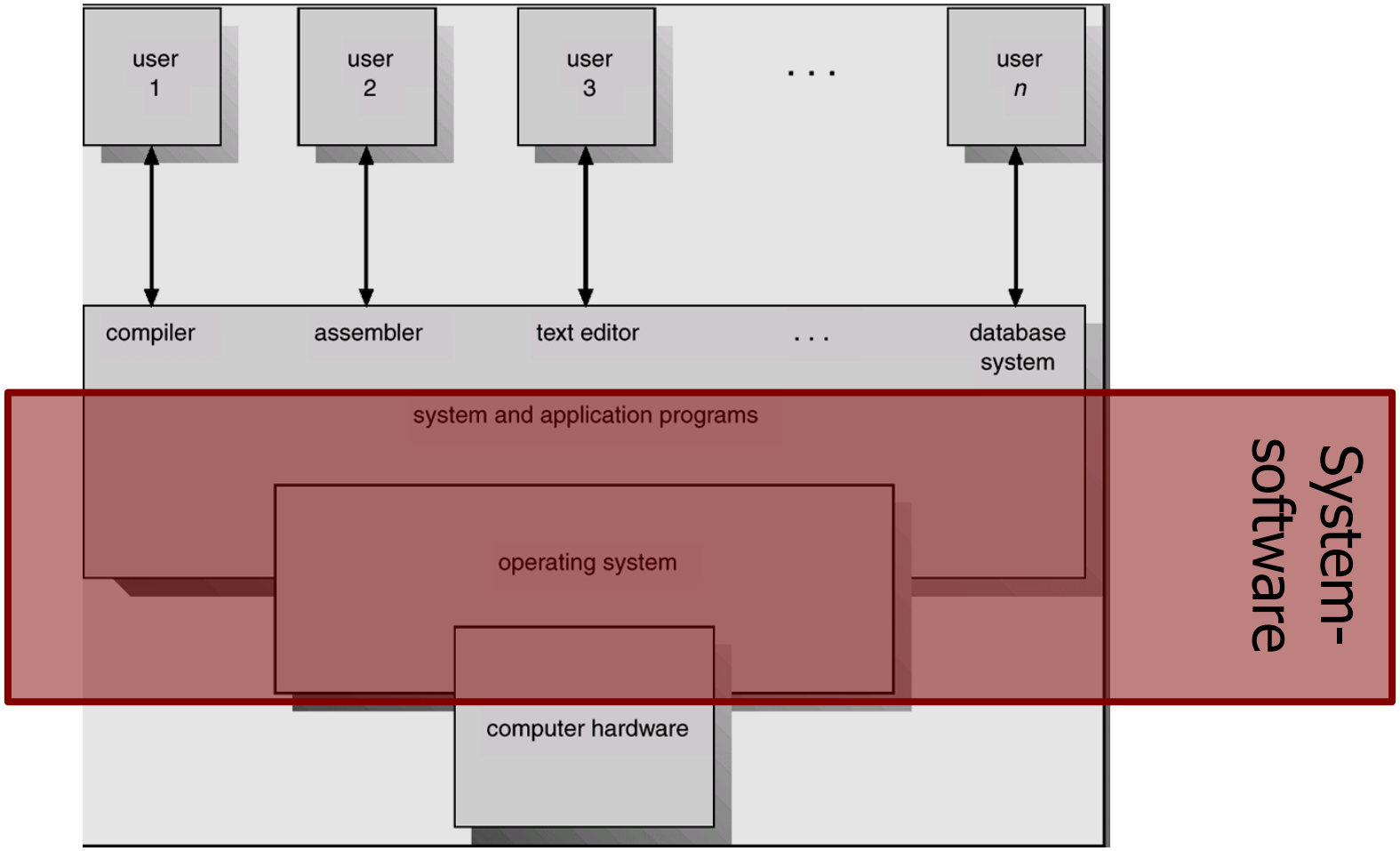
1. Rechnerarchitektur und Betriebssysteme

- Überblick
  - 1.1 Rechnerarchitektur
  - 1.2 Betriebssysteme: Grundlegende Funktionen, Konstruktionsformen
  - 1.3 Fallstudien: Windows, Unix, Android
  - 1.4 Parallele Architekturen

1.1 Computersysteme und Rechnerarchitektur

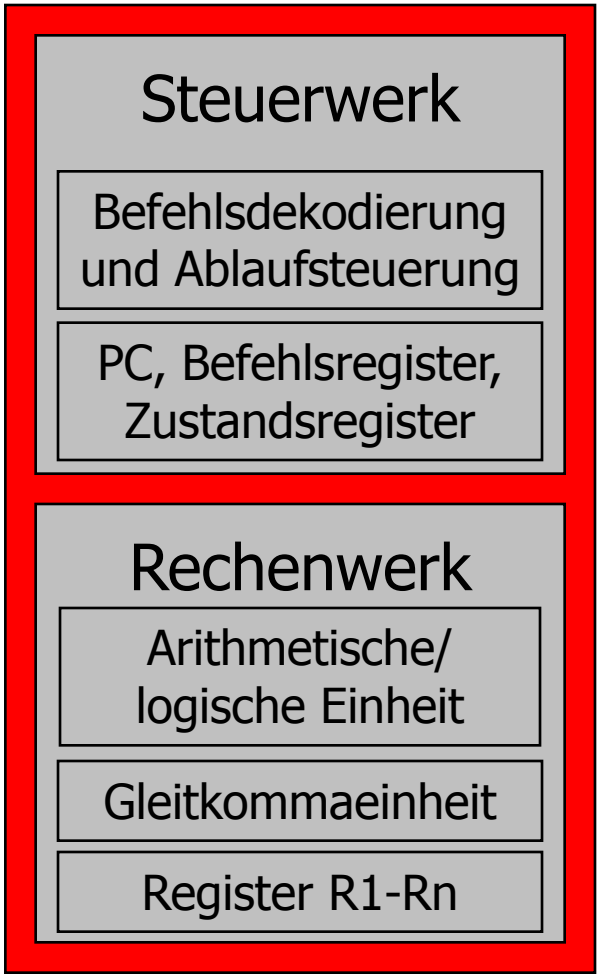


Definitionen der Grundbegriffe

- Systemsoftware und Systemprogrammierung eng gekoppelt an Rechnerarchitektur
- Hardware
  - Ermöglicht Grundrechenleistung (CPU, Speicher, E/A)
- Betriebssystem (operating system)
  - Kontrolliert und koordiniert die Nutzung der Hardware durch Anwendungsprogramme und Benutzer
- Anwendungsprogramme (application programs)
  - Definieren die Art der Nutzung von Systemressourcen zur Lösung der Benutzerprobleme
- Benutzer (users)
  - Menschen, Maschinen, Computer

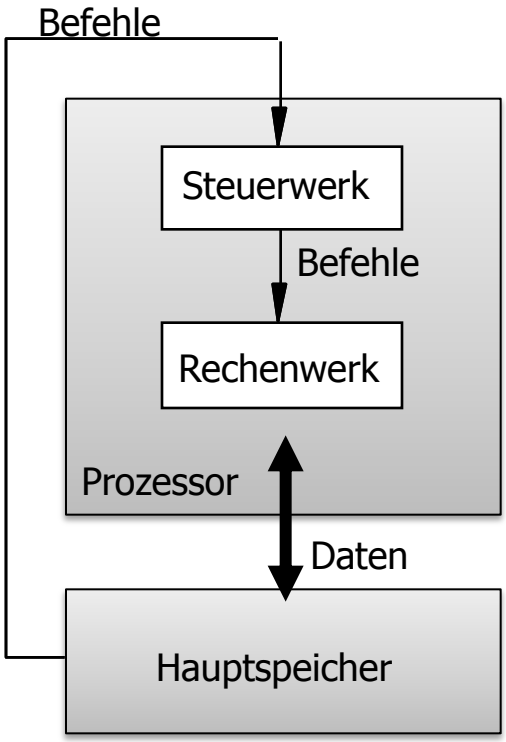
Prozessor

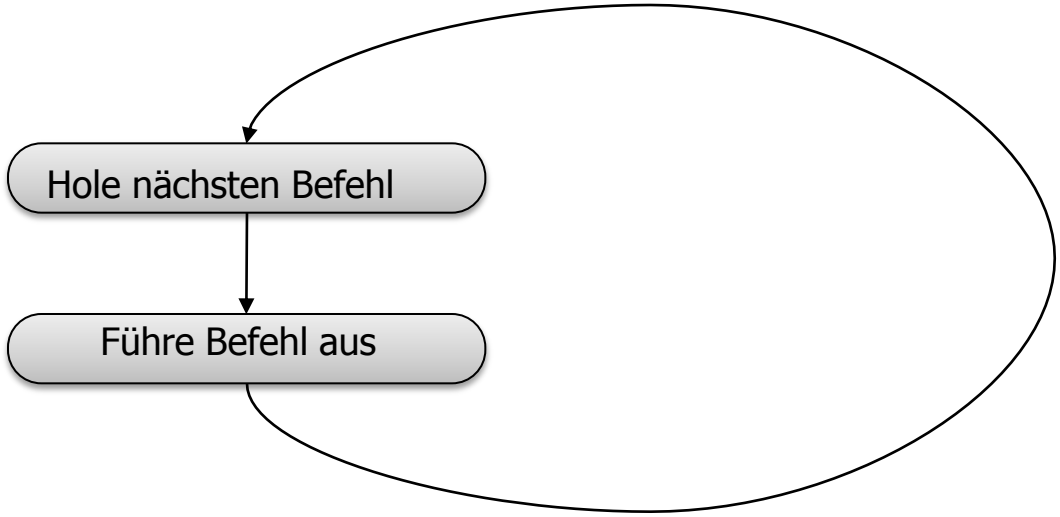
- Grundelemente eines Prozessors
  - Rechenwerk
  - Steuerwerk: Stellt Daten für das Rechenwerk zur Verfügung
    - → Holt Befehle aus dem Speicher
    - → Koordiniert den internen Ablauf
  - Register: Speicher mit Informationen über die aktuelle Programmbearbeitung, z.B.
    - Rechenregister, Indexregister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



Arbeitsweise des Prozessors (vereinfacht)

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





Prozessorzyklus

Arbeitsweise des Prozessors (vereinfacht)

- Befehlsverarbeitung nach starrem Zweitakt-Zyklus

  Takt 1 (*Befehlszustand*):

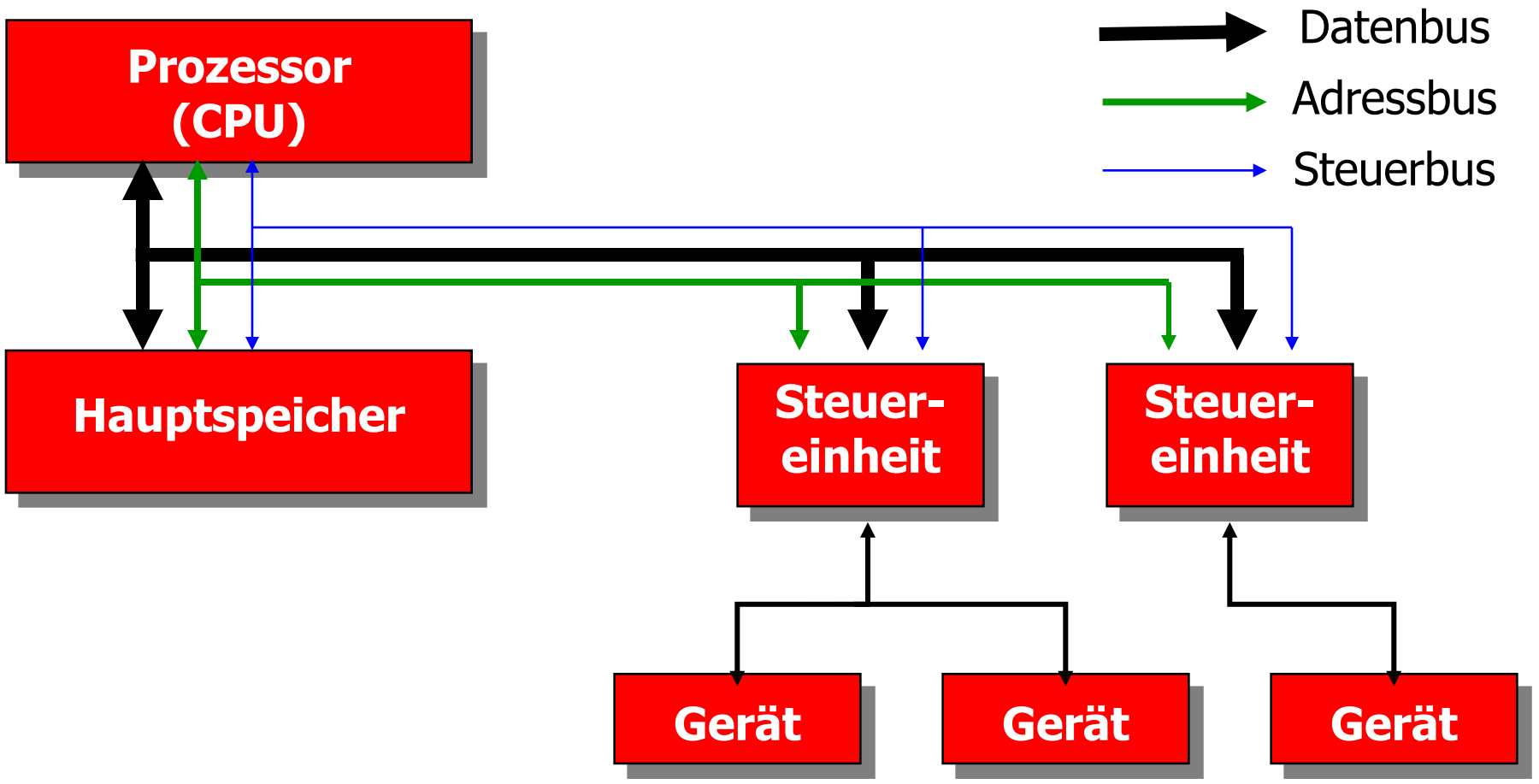
  - Befehl holen und interpretieren

    ⇒Inhalt der Speicherzelle, auf die der Befehlszähler zeigt, wird geholt und als Befehl interpretiert
  - Befehlszähler erhöhen
  - Adresse berechnen: Die physikalische Adresse des Datums oder des Sprungziels wird in Abhängigkeit von der Adressierungsart (indirekt, relativ, absolut…) berechnet

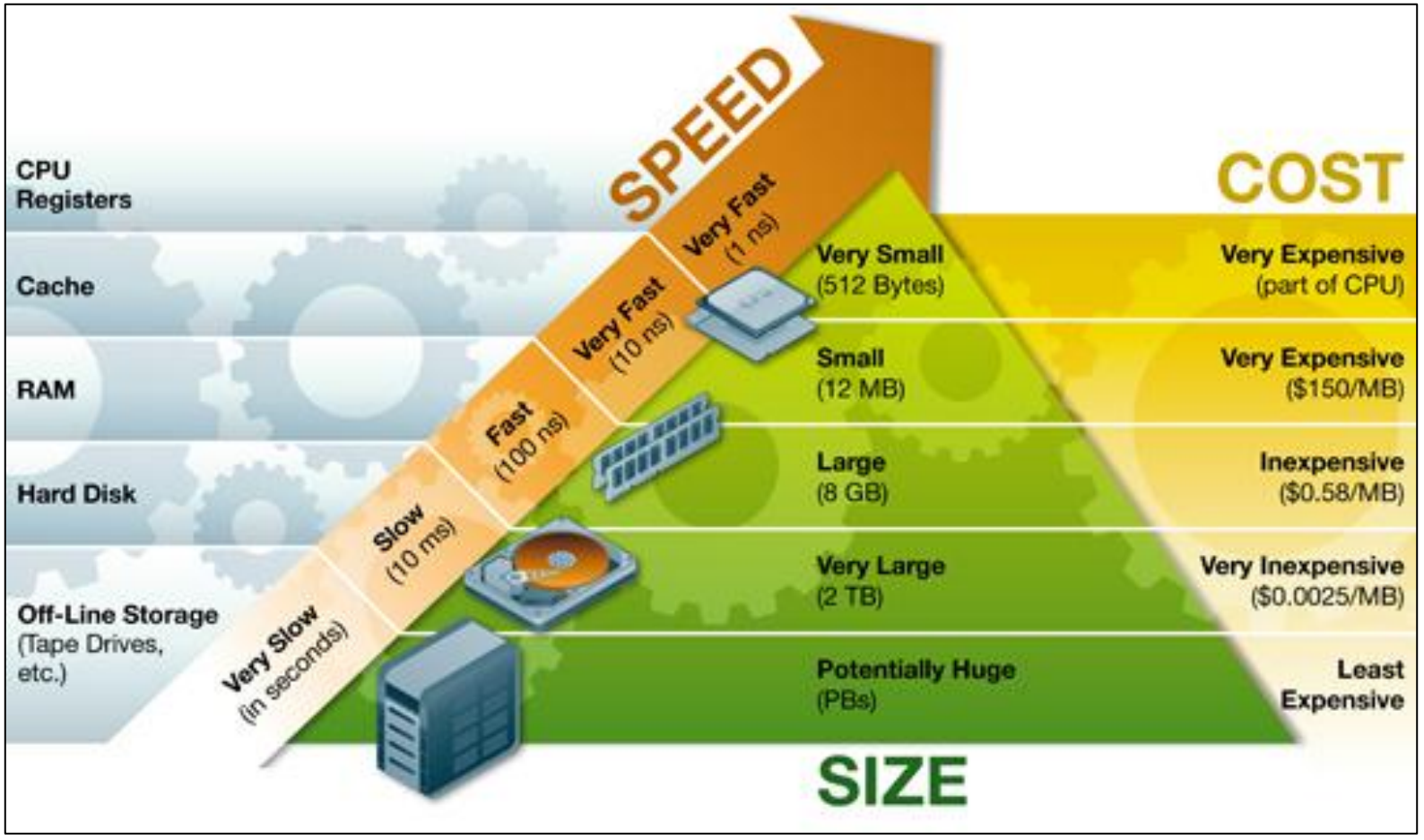
  Takt 2 (*Datumszustand*):

  - Datum holen: Die Bitkette, die zur berechneten Adresse gehört, wird geholt
  - Befehl ausführen: Die Bitkette wird instruktions-spezifisch (als Int, Float, Zeiger…) interpretiert und verarbeitet

Rechnerarchitektur nach von Neumann



Speicherhierarchie



ts.avnet.com

Cache

- Zwischenspeicher zur Verkleinerung der Lücke zwischen Prozessor- und Speichergeschwindigkeit
  - Inhalt einzelner Zellen samt Adresse wird zwischengespeichert
  - Beim Datenzugriff wird zunächst der Cache überprüft:
    - Falls Datum vorhanden $\Rightarrow$ 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.

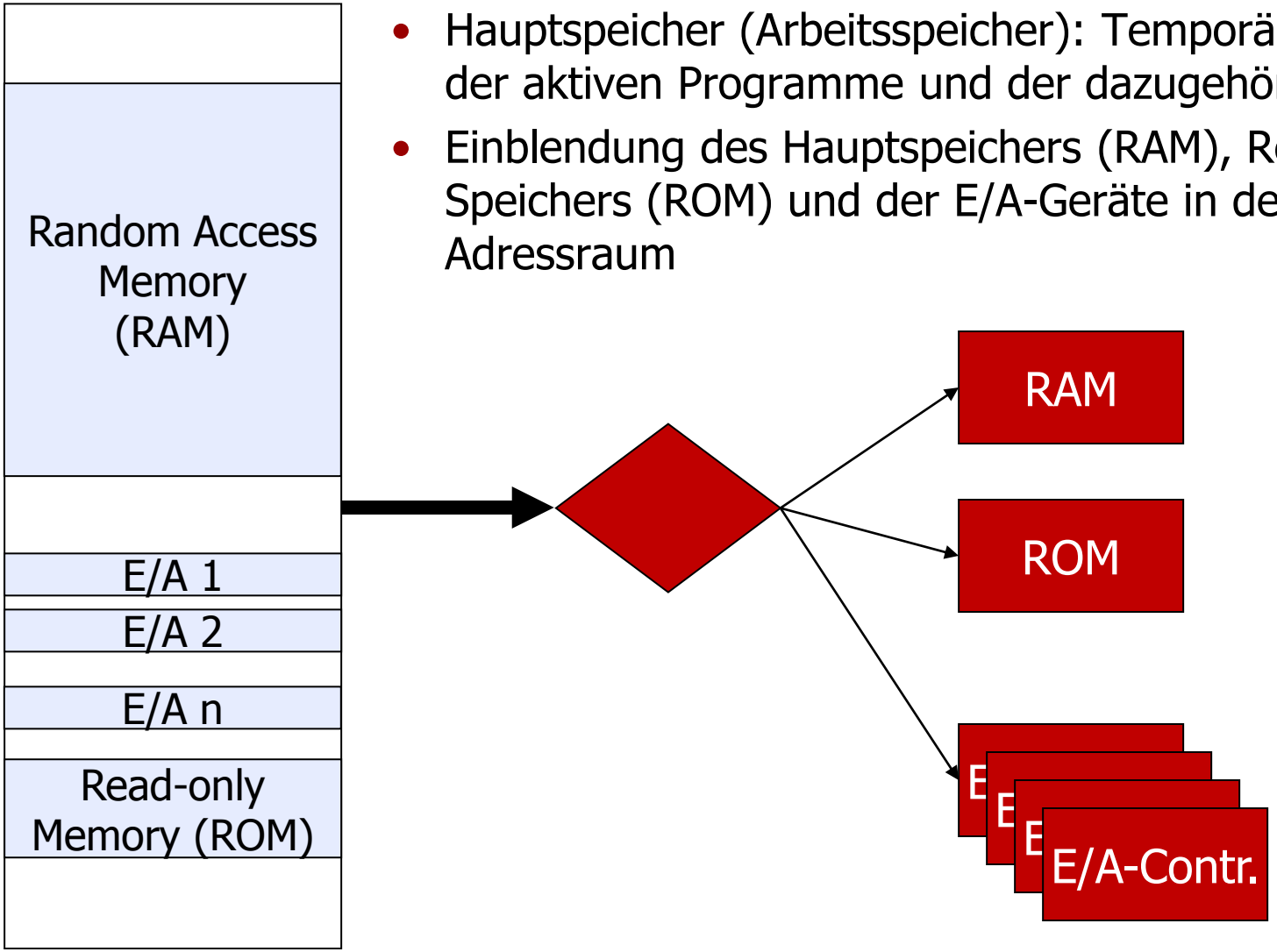
- Kalter / Heißer Cache
  - Gerade geladenes Programm $\Rightarrow$ Cacheinhalte entsprechen nicht den vom Programm referenzierten Zellen
    - Geringe Trefferrate $\Rightarrow$ Kalter (ineffizienter) Cache
  - Nach Vorlaufzeit: Cache passt sich an das aktuelle Programm an
    - Trefferwahrscheinlichkeit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

Adressräume

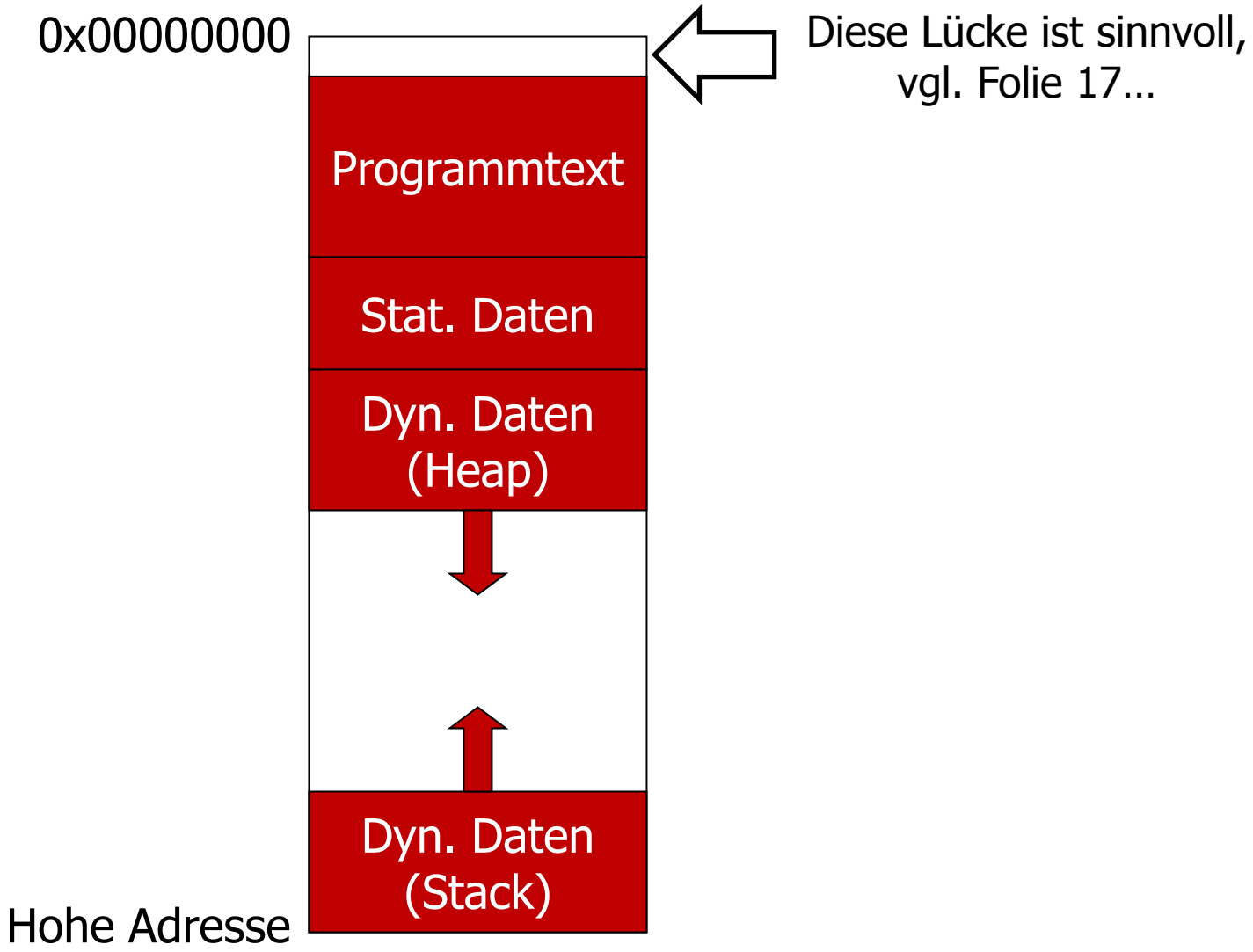
- Adressraum = Speicherkonfiguration
  - Physischer Adressraum:
    - existiert genau einmal
    - enthält alle Systemkomponenten (RAM, E/A-Geräte…)
    - Positionen der Komponenten nicht veränderbar
  - Virtueller Adressraum („Programmadressraum"):
    - vom Betriebssystem erzeugt und konfiguriert
    - i.d.R. gilt: je Prozess ein virtueller Adressraum
    - enthält die für das Programm nötigen Instruktionen und Daten
- Betriebssystem schaltet CPU beim Startvorgang in virtuellen Adressierungsmodus (Paging; Details erst in Kap. 6)
- Teile des Adressraums können undefiniert sein
  - Zugriff darauf führt zu einem Fehler

Physischer Adressraum

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



Virtueller Adressraum



Einschub: Speicherbereiche in C-Programmen

- Programmtext
  - Instruktionen des Programms
- Statische Daten
  - globale Variablen, lokale Variablen mit `static`-Modifier
- Dynamische Daten (Heap)
  - zur Laufzeit explizit reservierbarer Speicherbereich
  - wächst und schrumpft nach Bedarf
- Dynamische Daten (Stack)
  - je aufgerufene Funktion: lokale Variablen, Aufrufparameter
  - wächst, je tiefer die Aufrufkette ist (Rekursion!)
  - im Gegensatz zum Heap Benutzung „automatisch“

Statische Speicherverwaltung

- 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!

- Lösungsmöglichkeiten:
  - Obere Grenze z.B. für Arrays festlegen (unflexibel: entweder einschränkend oder verschwenderisch)
    ODER
  - Speicher dynamisch (d. h. zur Laufzeit) verwalten: belegen und freigeben gemäß tatsächlichem Bedarf

Zeiger

- Grundlage der dynamischen Speicherverwaltung
- Zeiger *verweist* auf Speicherbereich:
  - auf eine (einzelne) Variable ODER
  - auf ein Array (an den Anfang oder hinein)
- Operatoren: „Adresse-von“ (`&`) und „Wert-an“ (`*`)

```
int *a;
int x, y;

x = 42;
a = &x;
y = *a;

// y == 42
```

```
int *b;
int z[5];

b = &(z[0]);
b++;
z[1] = 42;

// *b == 42
```

Dynamische Speicherverwaltung

- Reservierung des nötigen Speichers im Heap durch Verwendung von Funktionen der C-Standardbibliothek wie `void *malloc(unsigned int size)`
  1. Aufruf der Funktion `malloc(size)` mit der genauen Angabe, wie viel Speicherplatz benötigt wird
  2. Steht genug Speicher zur Verfügung → Rückgabe eines Zeigers auf den reservierten Speicherbereich, sonst NULL (Zeiger auf die Adresse 0)
  3. Der Speicherblock kann mit Daten gefüllt werden

- Freigeben des reservierten Speichers mit:
  `void free(void *ptr)`
  → nur sinnvoll, wenn nicht am Programmende

Portables Allozieren mit `malloc()`; der NULL-Zeiger

- Verwendung von Konstanten als Größenangabe bei `malloc()` führt zu schlecht portierbaren Programmen:

```
int *ptr;
ptr = malloc(4); // Größe von „int“ nicht def.
```

- Stattdessen so:

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

- NULL-Zeiger: Vordefinierter Zeiger, dessen Wert sich von allen regulären (gültigen) Zeigern unterscheidet

  → Nutzung zur Anzeige von Fehlern

  → Bei jedem Aufruf einer Funktion, die einen Zeiger zurückgibt, muss auf NULL getestet und ggf. Fehler abgefangen werden!

  → Verwendung von NULL führt i.d.R. zum Programmabsturz

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 (spezielle Liste): Zugriff erfolgt immer über das oberste Element (LIFO: Last In First Out)
  - Queues (spezielle Liste): Elemente werden am Listenende eingefügt und am Listenanfang gelesen (FIFO: First In First Out)

Sicherheit der CPU

- Unterscheidung aus Sicherheitsgründen zwischen zwei Zuständen oder Modi (Bit im Prozessorstatusregister)

  - Benutzermodus/unprivilegierter Zustand (*user mode*)
    - einige Instruktionen gesperrt
    - einige Register nicht zugreifbar
    - in der Regel für Benutzerprogramme

  - Systemmodus/privilegierter Zustand (*system/supervisor mode*, ...)
    - alle Instruktionen zulässig
    - alle Register benutzbar
    - in der Regel für das Betriebssystem

Sicherheit der CPU (2)

- Wechsel zwischen den Modi:
  - unprivilegiert → privilegiert:
    - beim Auftreten einer Unterbrechung (s. Folie 28 ff.)
    - beim Auslösen eines Fehlers (Division durch Null, Zugriffsversuch auf ein „Loch" im Adressraum, verbotene Instruktion …)
    - durch explizite Instruktion (z. B. x86: `sysenter`, ARM: `svc`)
    - Ausführung wird an vom BS definierten Einsprungpunkten fortgesetzt
    - ursprünglicher Prozessorzustand (Register etc.) wird gesichert
  - privilegiert → unprivilegiert:
    - jederzeit erlaubt
    - vom BS durchgeführt, um das unterbrochene Programm fortzusetzen

Sicherheit der CPU (3)

- Terminologie der Wechselereignisse nicht ganz trennscharf:
  - Auftreten einer Unterbrechung: *Interrupt*
  - Instruktionen, die explizit das BS aufrufen (`sysenter`, `svc`) oder die im unprivilegierten Modus verboten sind → *Trap*
  - Instruktionen, die einen Fehler auslösen → *Exception*
  - Instruktionen, die einen Speicherfehler („Loch" im Adressraum) auslösen → *Fault*

- „Trap": Falle, die zuschnappt, wenn eine verbotene Instruktion ausgeführt wird; manche Traps sind auch konfigurierbar (z. B. x86 `rdtsc` zum Auslesen des Prozessortaktzählers kann trappen, muss aber nicht – Wahl des BS)

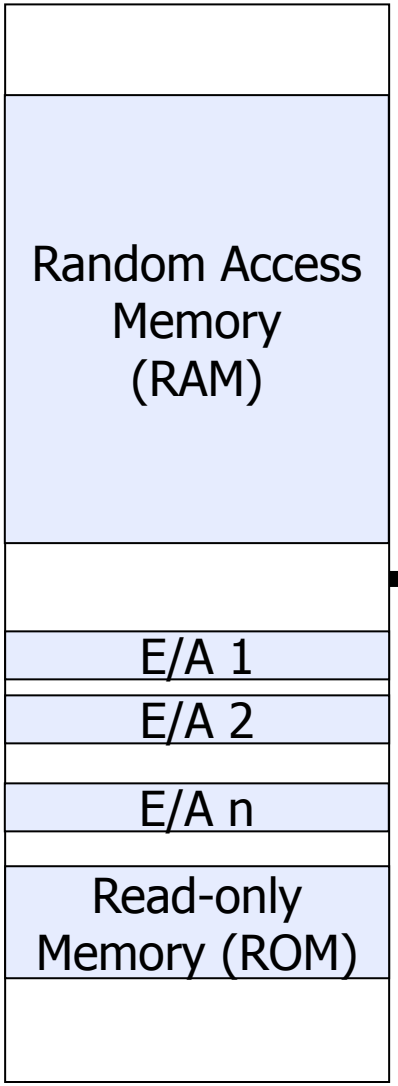
Ein- und Ausgabearchitekturen

- Vielfältige Geräte erfordern verschiedene Herangehensweisen

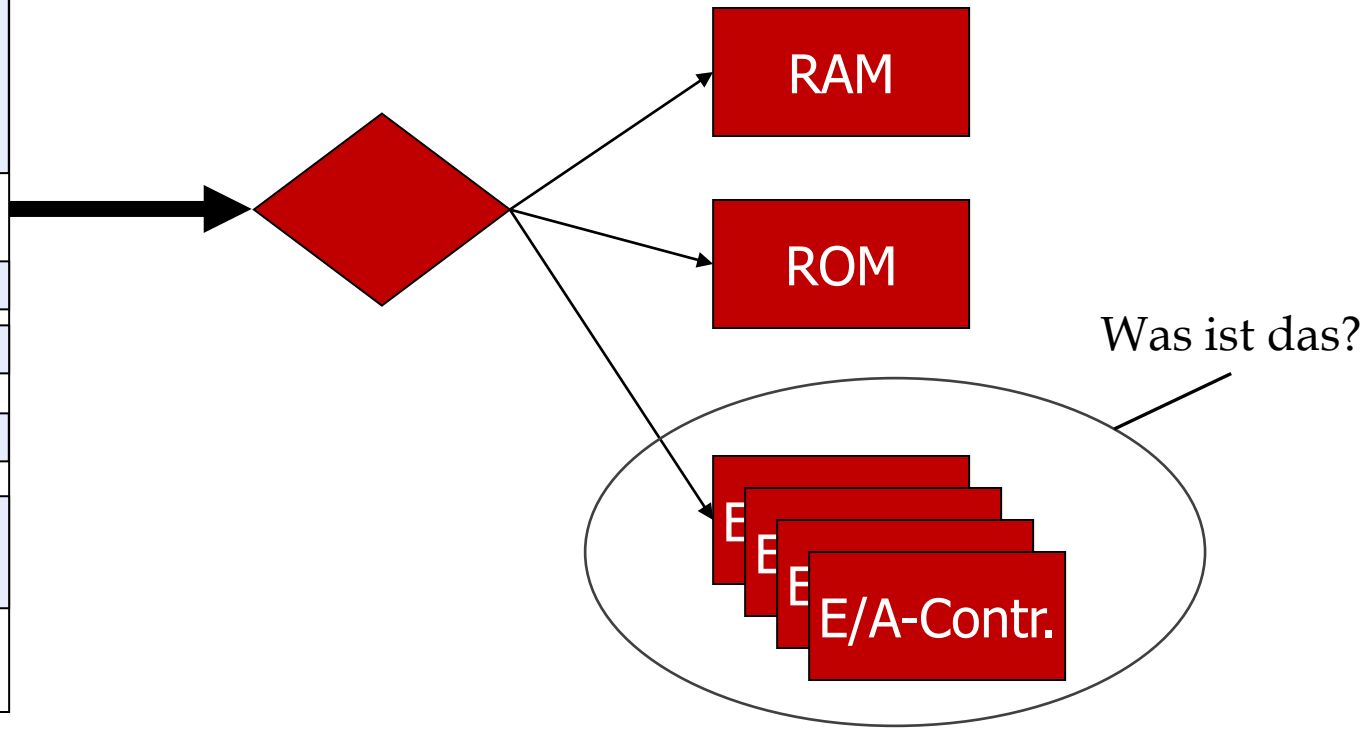
Device	Purpose	Partner	Data Rate
Keyboard	input	human	10 B/s
Mouse	input	human	200 B/s
Microphone	input	human	1-8 KB/s
Voice output	output	human	1-8 KB/s
Laser printer	output	human	0.1-100 MB/s
Graphic display	output	human	30-1000 MB/s
CPU to frame buffer	output	machine	133-8000 MB/s
Network-LAN	in-/output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

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

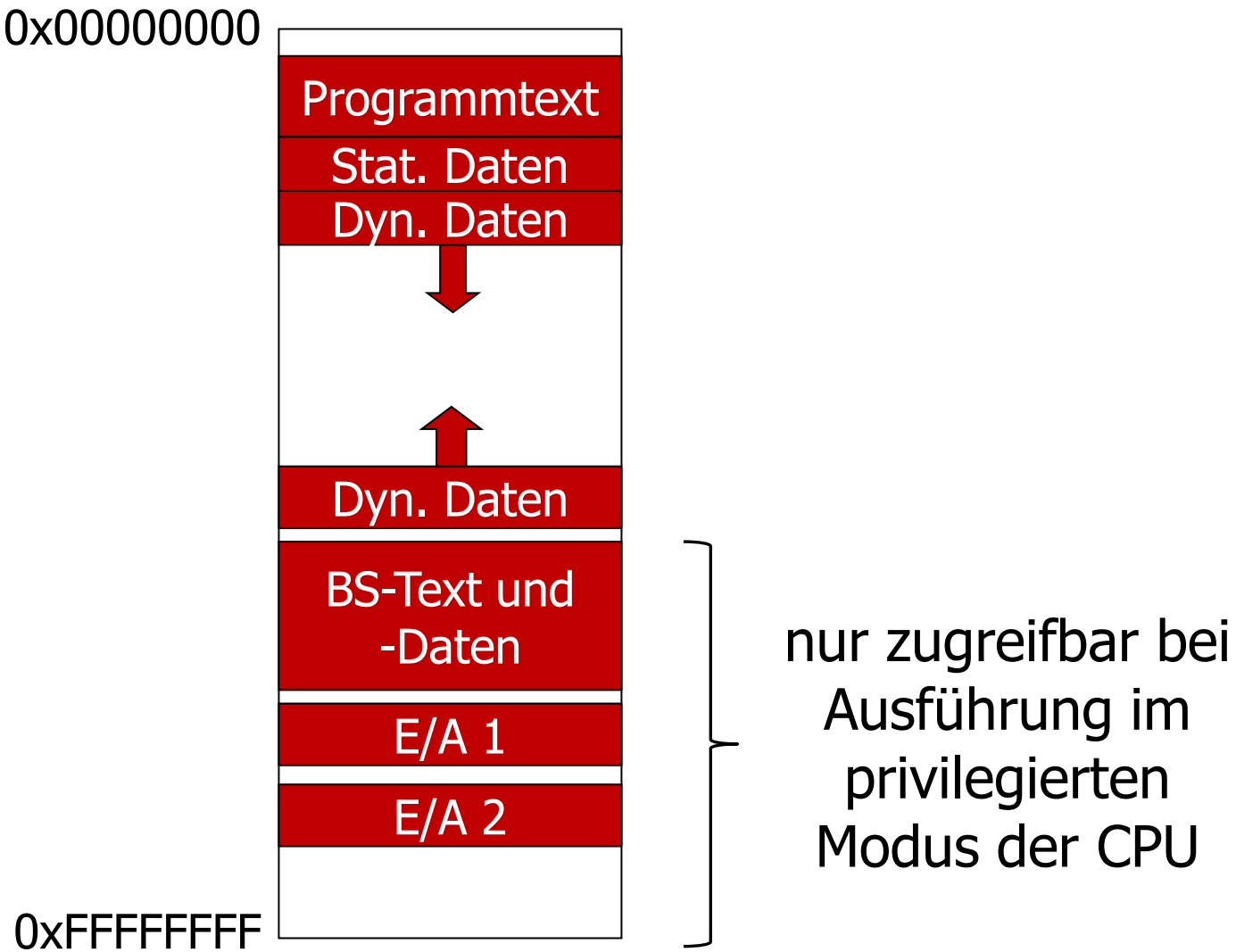
Physischer Adressraum



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

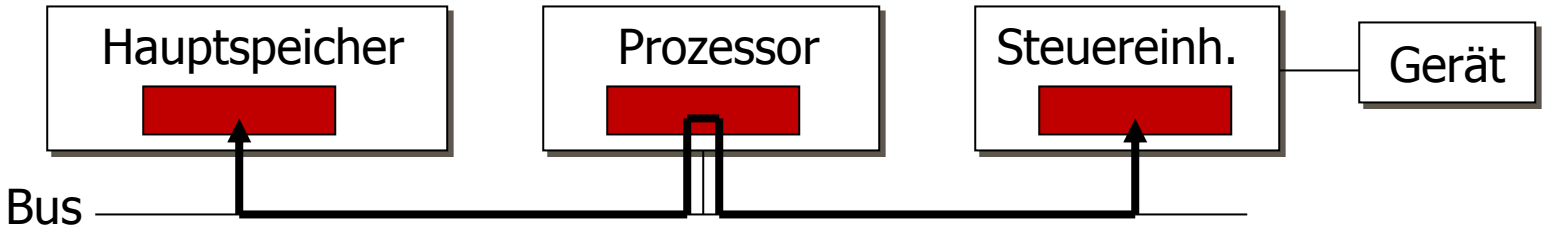


Vorgriff: Virtueller Adressraum (Sicht des Betriebssystems)



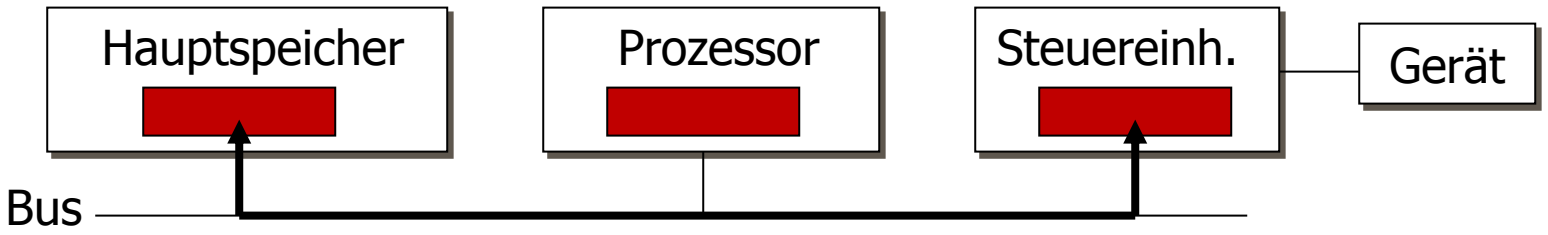
Ein- und Ausgabearchitekturen

- *Speicherbasierte E/A*: CPU liest wortweise Daten aus dem Hauptspeicher und schreibt diese in Register der Steuereinheit
- Kommunikationsmuster gerätespezifisch; Beispiel:
  - CPU schreibt in Befehlsregister der Steuereinheit: „Daten-transfer zum Gerät beginnt, Nachricht X Worte lang"
  - CPU kopiert Speicherinhalt Wort für Wort in das Datenregister der Steuereinheit
  - CPU liest ggf. Statusregister der Steuereinheit (Überprüfung, ob Gerät alle Daten akzeptiert hat)



Ein- und Ausgabearchitekturen (2)

- *Direkter Speicherzugriff (DMA)*: Steuereinheit kann über den Bus selbständig auf den Hauptspeicher zugreifen
- CPU initiiert nur den Transfer; Beispiel:
  - CPU schreibt (physische) Startadresse und Länge in Adressregister der Steuereinheit
  - CPU schreibt „DMA-Lese-Transfer starten" in Befehlsregister
  - Steuereinheit liest eigenständig gewünschte Menge Daten aus dem Speicher
- CPU kann währenddessen andere Dinge tun
- Wie erfährt CPU vom Abschluss des Transfers?

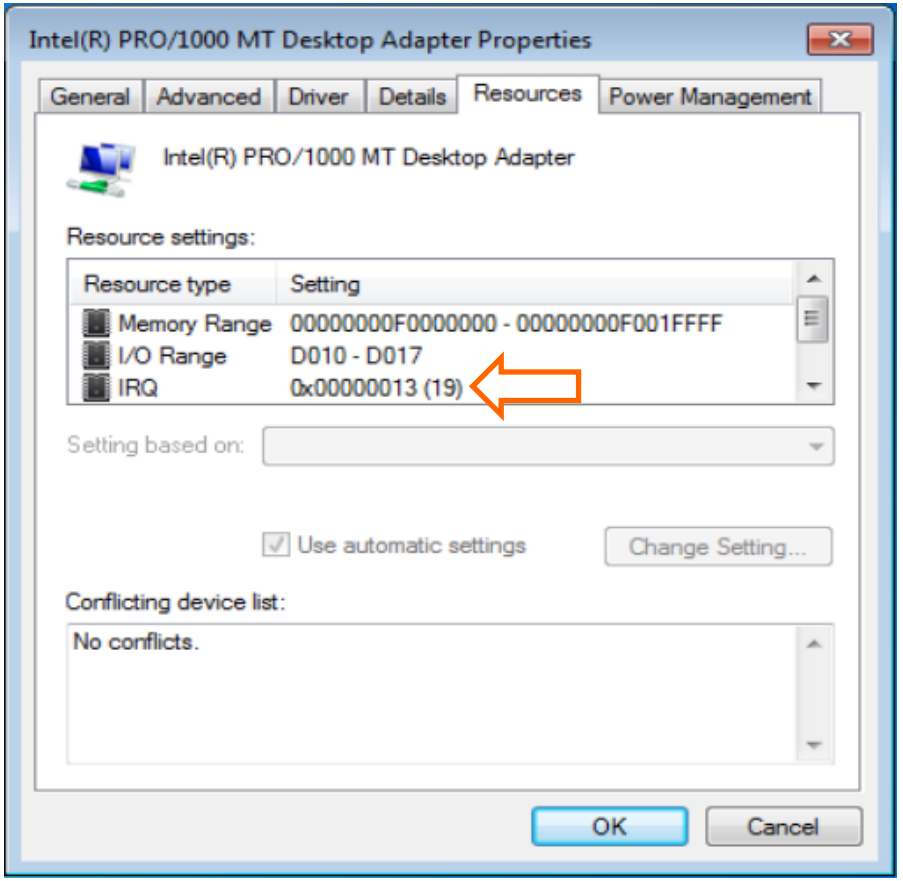


Reaktion

- Information der CPU nach Ende der E/A-Operation

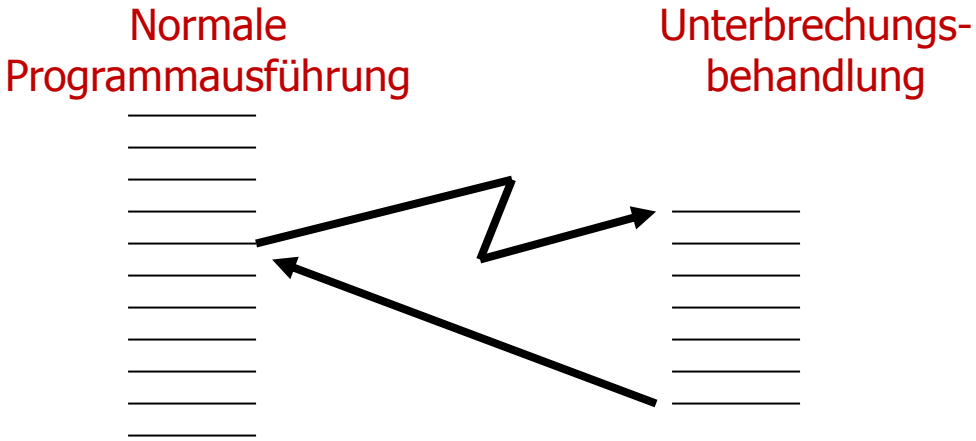
1. Polling: CPU fragt gelegentlich das Statusregister der Steuereinheit ab (Ineffizient!)

2. Unterbrechung / Interrupt: Spezielles Signal informiert die CPU über das Ende der Übertragung



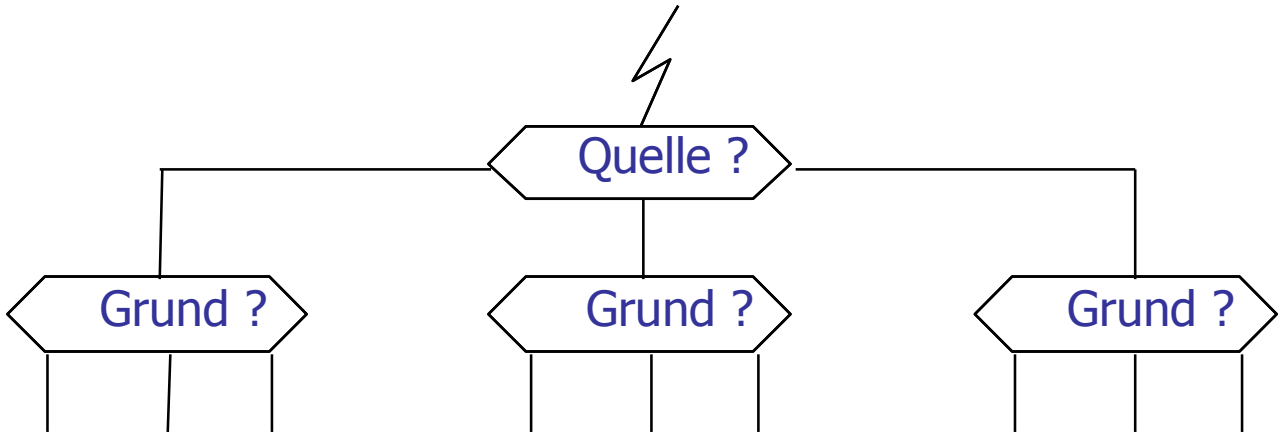
Unterbrechungen (Interrupts)

- 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 durchgeführt oder veranlasst
  - Falls nein → nächster Befehl wird bearbeitet



Unterbrechungsanalyse

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



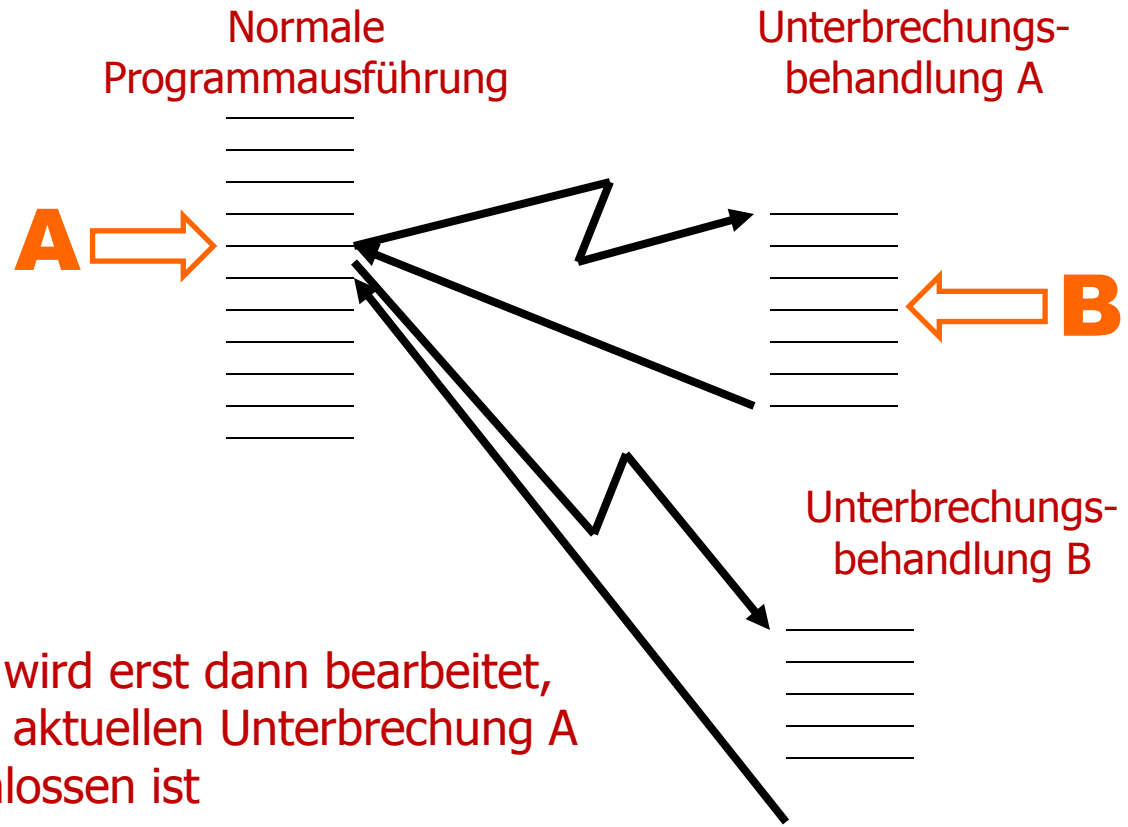
Unterbrechungsbehandlung

- 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)

Sequentielle Unterbrechungsbehandlung

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

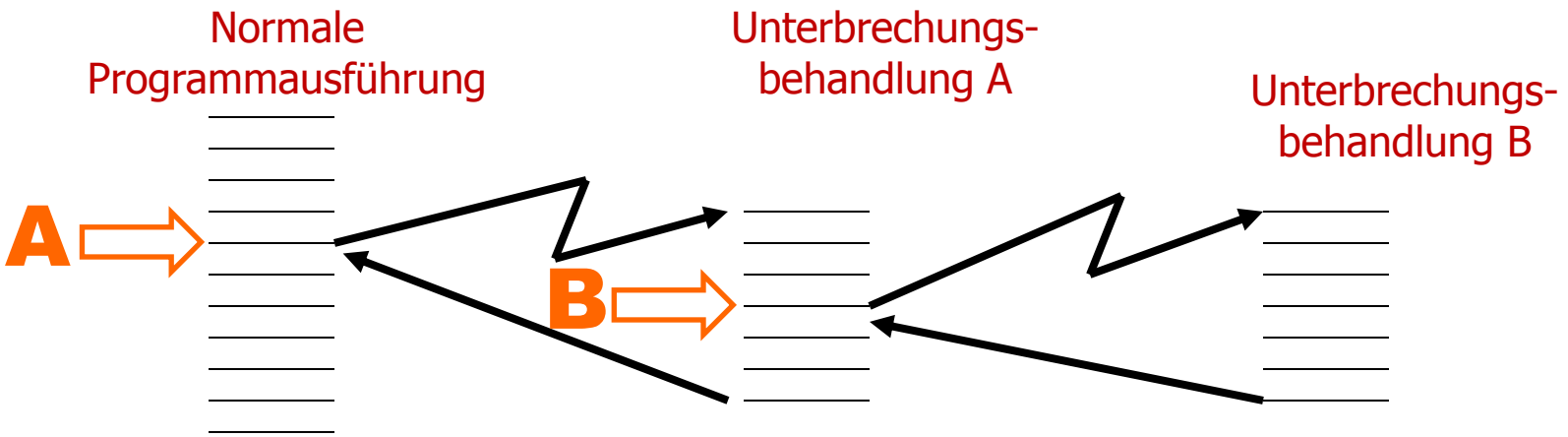


Zweite Unterbrechung B wird erst dann bearbeitet, wenn die Bearbeitung der aktuellen Unterbrechung A abgeschlossen ist

Geschachtelte Unterbrechungsbehandlung

Geschachtelte Unterbrechungsbehandlung

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



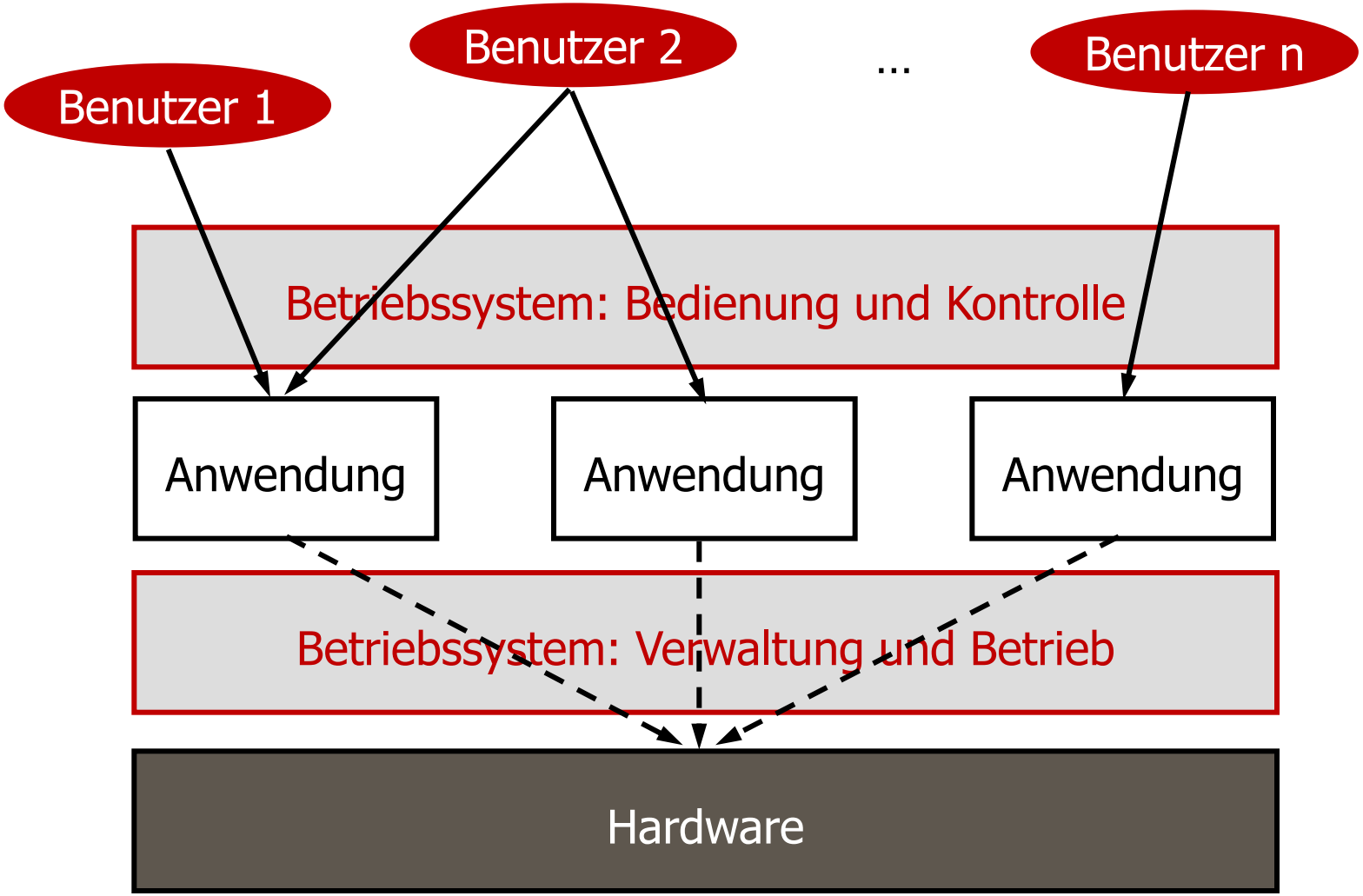
1.2 Definition Betriebssystem

- 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

- 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



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
  1. Unterbrechungsverarbeitung (*interrupt handling*)
  2. Verteilung (*dispatching*): Prozessumschaltung
  3. Betriebsmittelverwaltung (*resource management*): Belegen, Freigeben und Betreiben von Betriebsmitteln, Werkzeuge zur Prozesssynchronisation
  4. Programmallokation (*program allocation*): Linken von Teilprogrammen, Laden und Verdrängen von Programmen in/aus dem Hauptspeicher

Einzelfunktionen eines Betriebssystems

- Grundlegende Betriebssystemfunktionen (… Fortsetzung)
  5. Dateiverwaltung (*file management*)
     - Organisation des Speicherplatzes in Form von Dateien auf Datenträgern
     - Bereitstellung von Funktionen zur Speicherung, Modifikation und Wiedergewinnung der gespeicherten Informationen
  6. Auftragsteuerung (*job control*)
     - Festlegung der Reihenfolge, in der die eingegangenen Aufträge und deren Bestandteile bearbeitet werden sollen
  7. 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)

Kommunikation mit dem BS: der Systemaufruf

- BS bietet Funktionalität über „Systemaufruf"-Interface an
- Ablauf:
  1. Anwendung bereitet Systemaufruf vor (Register mit Parametern belegen, architekturspezifisch)
  2. Anwendung führt spezielle Instruktion aus (`svc`/...) ⟶ *Trap*
  3. Ausführung springt zu BS-Behandlungsroutine (⟶ privilegierter Modus!) für Systemaufrufe
  4. BS analysiert Parameter, identifiziert gewünschte Funktionalität
  5. BS prüft Berechtigung, Ressourcen, … führt ggf. gewünschte Funktion durch
  6. BS setzt Anwendung fort (Rückkehr in unprivilegierten Modus zur Instruktion, die der aus Schritt 2. folgt)

Mechanismen und Methoden (Policies)

- Wichtige Unterscheidung zwischen Mechanismen und Policies
  - Mechanismus: Wie wird eine Aufgabe prinzipiell gelöst?
  - Policy: Welche Vorgaben/Parameter werden im konkreten Fall eingesetzt?
- Beispiel: Zeitscheibenprinzip
  - Existenz eines Zeitgebers 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 Veränderung der Policy durch Anpassung von Parametern umgesetzt werden kann

Strukturen der Betriebssysteme

- Häufige Designstrukturen für Betriebssysteme
  - Monolithisches System
  - (historisch: Geschichtetes System)
  - Hypervisor mit Virtuellen Maschinen
  - Mikrokern
  - Exokern

Monolithische Systeme

- Gesamte Funktionalität in einem großen Programm vereint
  - Unterbrechungsbehandlung, Systemaufrufbehandlung
  - Treiber für E/A-Geräte
  - Scheduler
  - Abstraktionen: Dateisysteme, Netzwerkprotokolle...
- Vorteil: einfach zu konstruieren
- Nachteile:
  - Menge an Quellcode sehr groß
  - Keine Trennung zwischen Komponenten → anfällig!

Monolithische Systeme mit Modulen

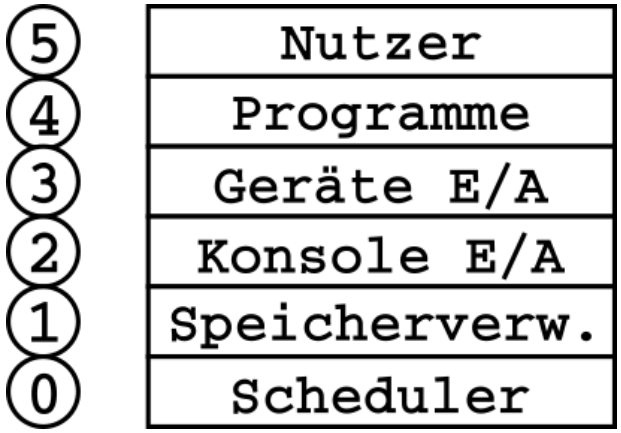
- Linux enthält Treiber für Tausende von Geräten
- ein PC enthält eher nur ein paar Dutzend Geräte
- Problem: Linux-Kern unnötig groß

- Idee: Treiber nicht in Linux-Kern integrieren, sondern separat kompilieren & bereithalten
- Treiber werden bei Bedarf von BS in Speicher geladen, wenn BS entsprechendes Gerät vorfindet
- Übertragbar auf Dateisysteme, Netzwerkprotokolle, …

- Aber: löst nicht das Sicherheitsproblem!

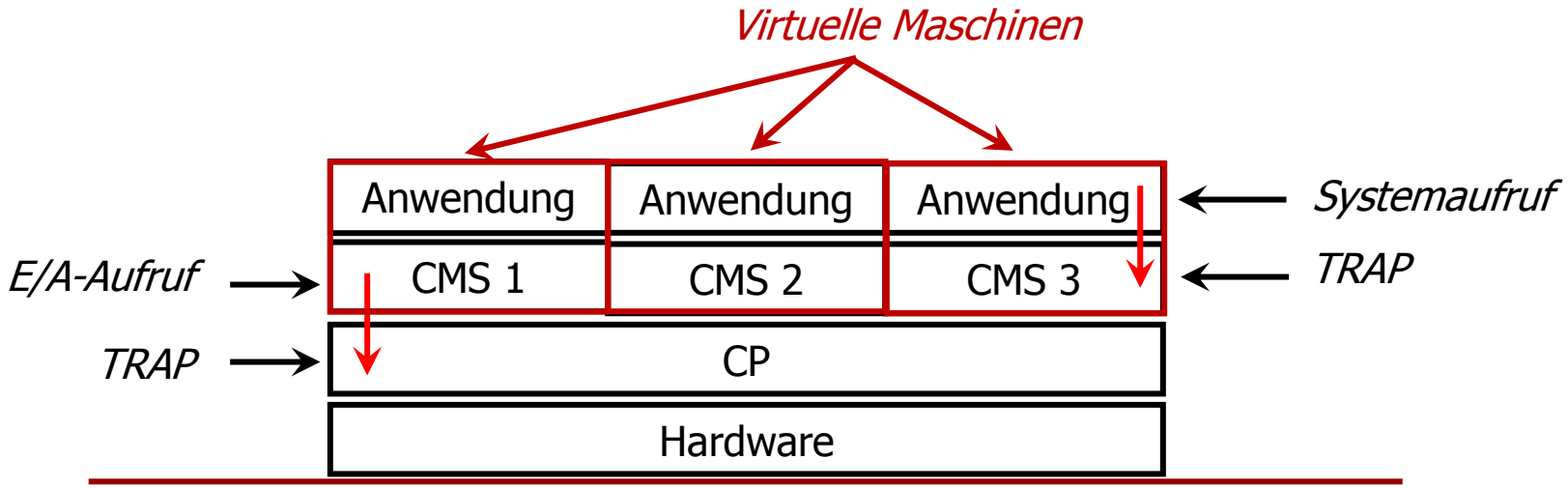
Geschichtete Systeme

- Historisch, vgl. „THE Multiprogramming System“ von E. Dijkstra
- Monolithisches Design, aber mit interner Struktur
- Abhängigkeit nur von höherer zu niederer Schicht
- Konstruktion sollte Entwicklung und formale Beschreibung vereinfachen



Hypervisor und Virtuelle Maschinen

- IBM: CP/CMS, der erste Hypervisor (später reimplementiert als VM/370)
- Zwei Komponenten:
  1. Control Program (CP): Ausführung auf der realen Hardware, bildet für darauf laufende Software die echte Systemhardware nach
  2. Cambridge Monitor System (CMS): häufig genutztes (Single-User-) Betriebssystem in den von CP bereitgestellten virtuellen Maschinen
- Systemaufrufe von Anwendungen landen in CMS
- Zugriffe auf E/A-Geräte durch CMS von CP überwacht



Mikrokerne

- Idee eines minimalen Kerns durch Auslagerung von BS-Funktionen als normale Prozesse: Server-Dienste
- Durch Aufteilung des BS entstehen Dienste wie Dateiserver, Grafikanzeigeserver, Druckserver, …
- Paradigma Mikrokern: nur Funktionalität, die *Systemmodus unbedingt benötigt*, verbleibt im Kern
- Vorteile:
  - Sehr schlanker und effizienter Kern
  - Fehlerbegrenzung: wenn ein Dienst abstürzt,
    - kann er neugestartet werden
    - hat das kaum Einfluss auf den Rest des Systems
- Nachteile:
  - Komplexe Implementierung
  - Langsamere Systemaufrufe, da Prozessumschaltung

Exokerne

- Klassische BS verbinden Abstraktion (Dateien in einem Dateisystem statt Blöcke auf einer Festplatte) mit Kontrolle von Ressourcen („darf Prozess X Datei Y lesen?“)
- Idee Exokern:
  - Abstraktion ggf. für bestimmte Prozesse kontraproduktiv (z. B. Datenbankserver)
  - Im Kern nur unabstrahierte Ressourcenkontrolle (Prozess X: Festplattenblöcke 20-40)
  - Abstraktion kann bei Bedarf über Bibliotheken vom Prozess eingebunden werden
- Bisher rein akademisches Konzept (z. B. ExOS vom MIT); keine kommerziellen Produkte

1.3 Fallstudie: Windows

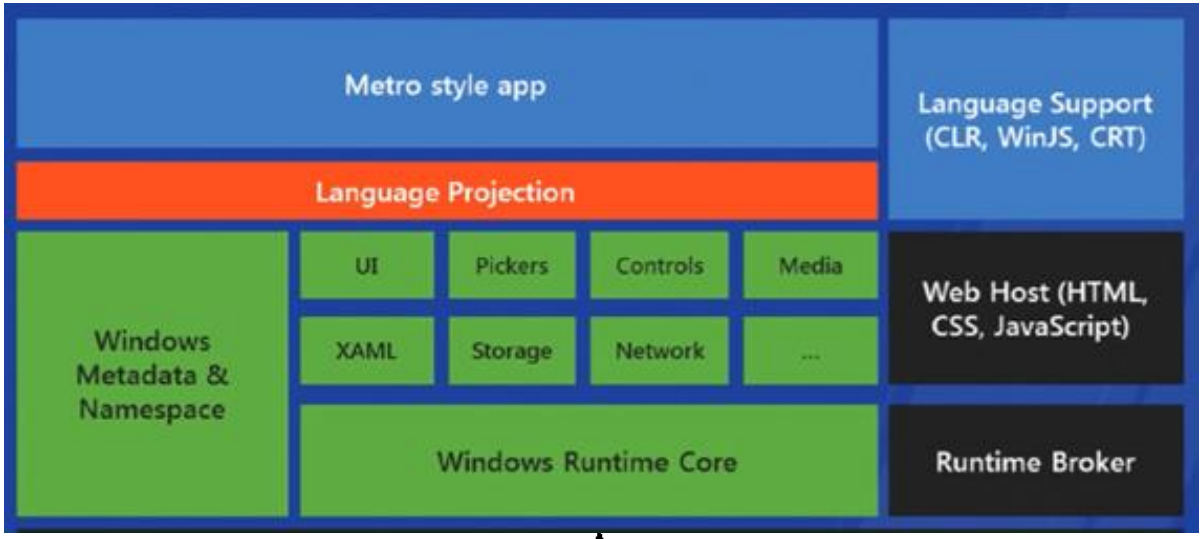
- Ursprünglich auf Betrieb von mehreren Teilsystemen ausgelegt (z.B. Unix, OS/2, Windows)

- Betriebssystemkern oft als Mikrokern bezeichnet, enthält allerdings Systemkomponenten, die nach Mikrokerndefinition nicht notwendigerweise integriert werden müssen

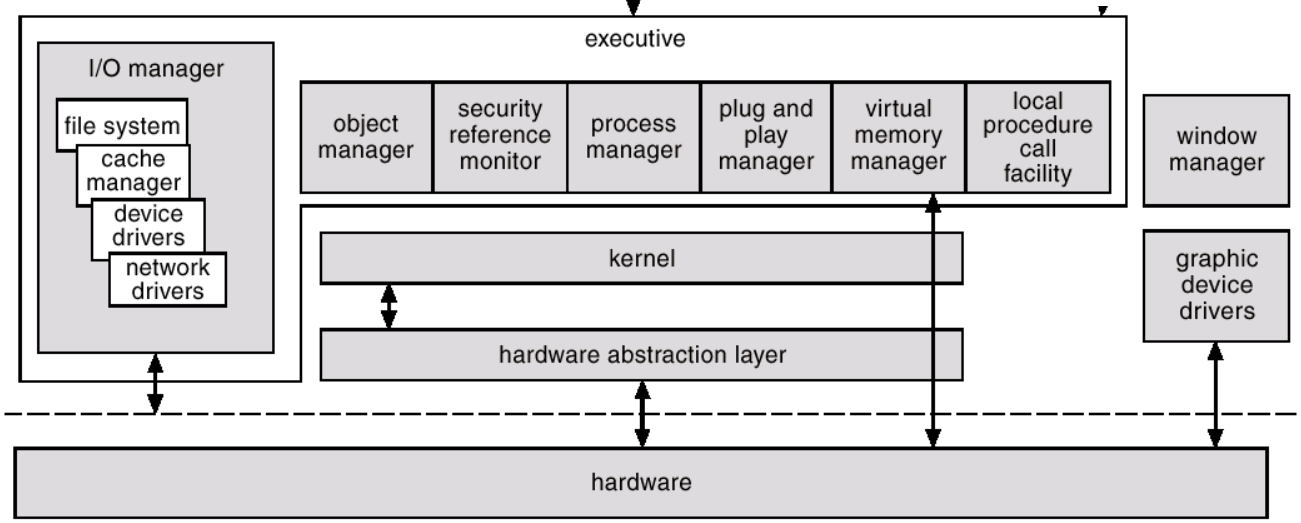
- Aufteilung der Systemkomponenten in Schichten
  - Hardwareabstraktionsschicht (HAL)
  - Kern mit zentralen Aufgaben
  - Executive
  - Gerätetreiber

Windows Architektur

www.buildwindows.com



Benutzermodus: geschützte Subsysteme



Kernmodus: privilegierte Subsysteme

Fallstudie: Unix

- Unix: 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="4">Systemaufrufe</td><td colspan="3">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>Signal-Behand-lung</td><td>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><td rowspan="2" colspan="2">Prozess-Scheduling</td></tr>
<tr><td>Line-Verwalt.</td><td>Routing</td><td>Puffer-Cache</td><td colspan="2">Seiten-Cache</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>

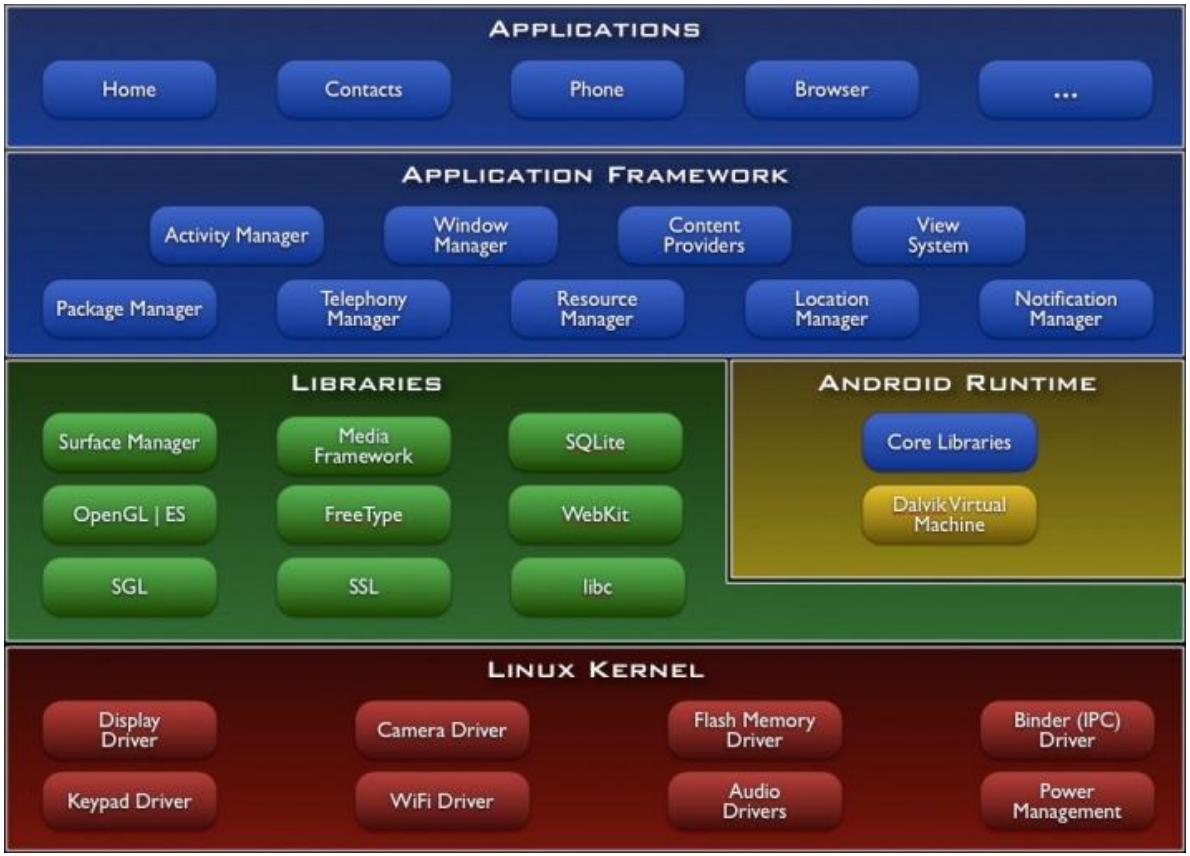
Fallstudie: Android

- 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 u.a. (www.openhandsetalliance.com)

Android Basis

- Android bietet Komponenten für
  - Sicherheit, Speicher/Prozessmanagement, Netzwerk, Gerätetreiber für GSM, Bluetooth, EDGE, 4G, Wlan, Camera, GPS, Kompass, und Beschleunigungssensoren
  - Laufzeitumgebung = Dalvik Virtual Machine (mittlerweile Android Virtual Machine)

  → Keine direkte Verwendung der Java-Bytecodes, aber Verwendung vieler Java-Werkzeuge



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

1.4 Parallele Architekturen

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

- Arten des Parallelismus
  - Implizit: die Möglichkeit der Parallelverarbeitung ist nicht a priori bekannt

    →Datenabhängigkeitsanalyse ermittelt die parallelen und sequentiellen Teilschritte des Algorithmus zur Laufzeit

  - Explizit: die Möglichkeit der Parallelverarbeitung wird a priori festgelegt

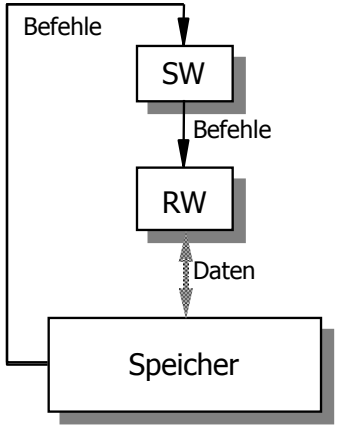
    → Einsatz von geeigneten Datentypen bzw. Datenstrukturen wie z.B. Vektoren bei Programmerstellung

Klassifikation von Rechnerarchitekturen

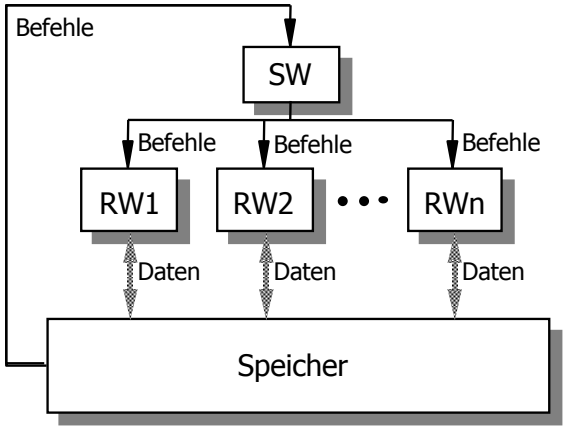
- Grobklassifikation nach Flynn: Unterscheidung nach der Anzahl von Befehls- und Datenströmen

	SD (Single Data)	MD (Multiple Data)
SI (Single Instruction)	SISD konventionelle von-Neumann-Rechner	SIMD Vektorrechner, Feldrechner
MI (Multiple Instruction)	MISD Datenflussmaschinen	MIMD Multiprozessorsysteme, Parallelrechner Verteilte Systeme

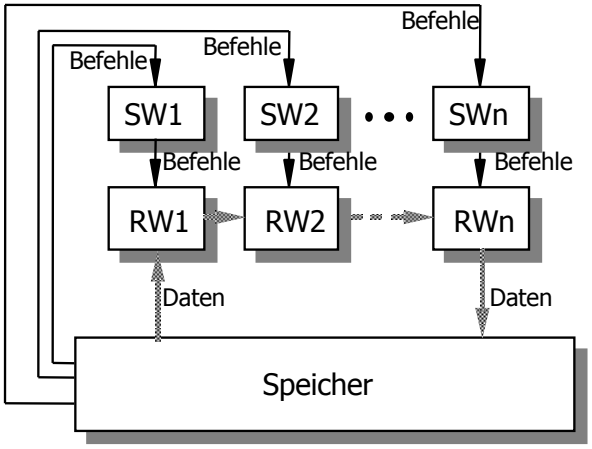
Flynn‘sches Klassifikationsschema



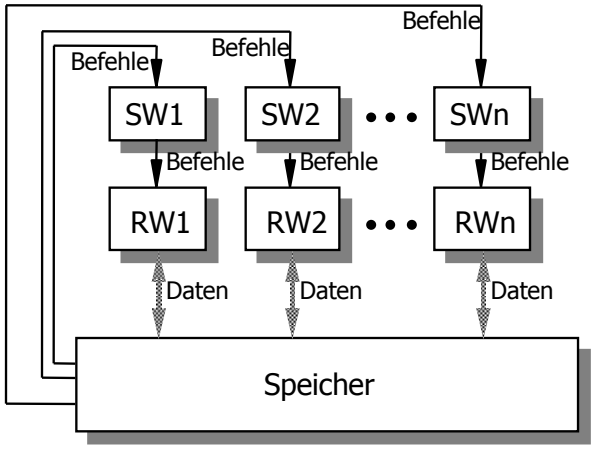
SISD



SIMD



MISD



MIMD

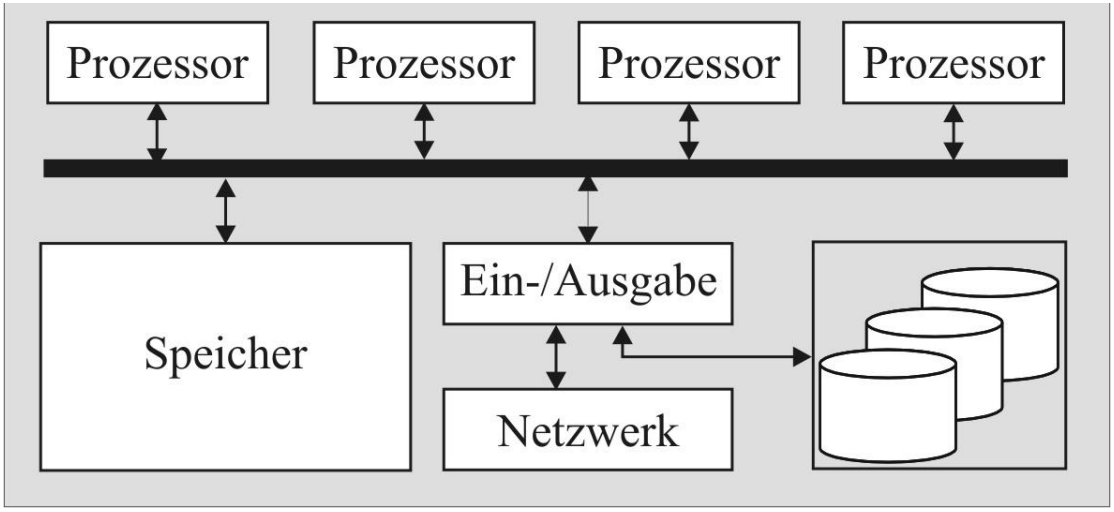
Klassifikation von MIMD Architekturen

- Wichtigstes Merkmal: physikalische Speicheranordnung
  - Gemeinsamer Speicher (*shared memory*)
  - Verteilter Speicher (*distributed memory*)

- Die Speicheranordnung beeinflusst weitere Merkmale
  - Programmiermodell: globaler Adressraum oder nachrichtenorientiert (*message passing*)
  - 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)



Architekturen mit verteiltem Speicher

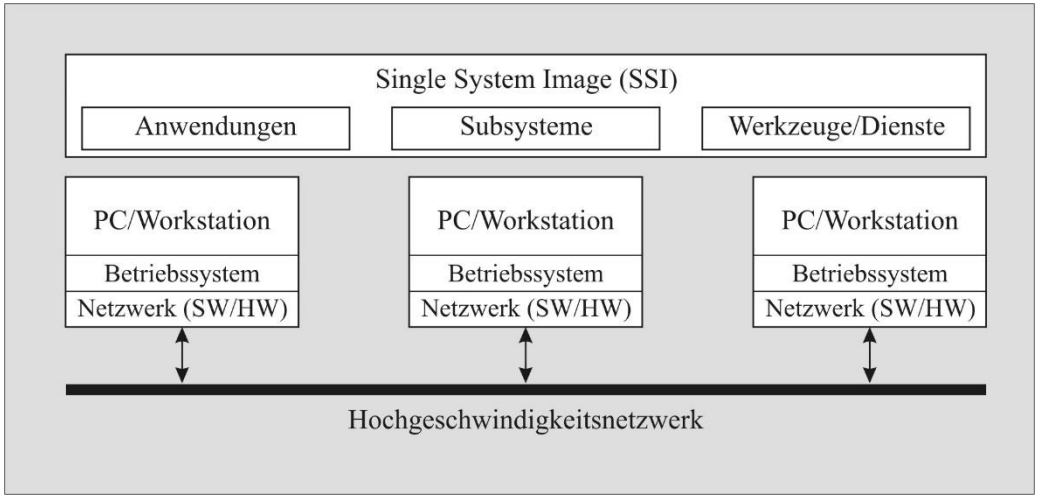
- Architekturen mit verteiltem Speicher bestehen aus vernetzten Knoten mit jeweils
  - Einem oder mehreren Prozessoren
  - Lokalen Speichermodulen
  - Verbindungsschnittstellen
- Kommunikation und Synchronisation zwischen den Prozessen auf verschiedenen Prozessoren erfolgt durch Austausch von Nachrichten
- 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(100000) bis O(1000000) (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

- 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

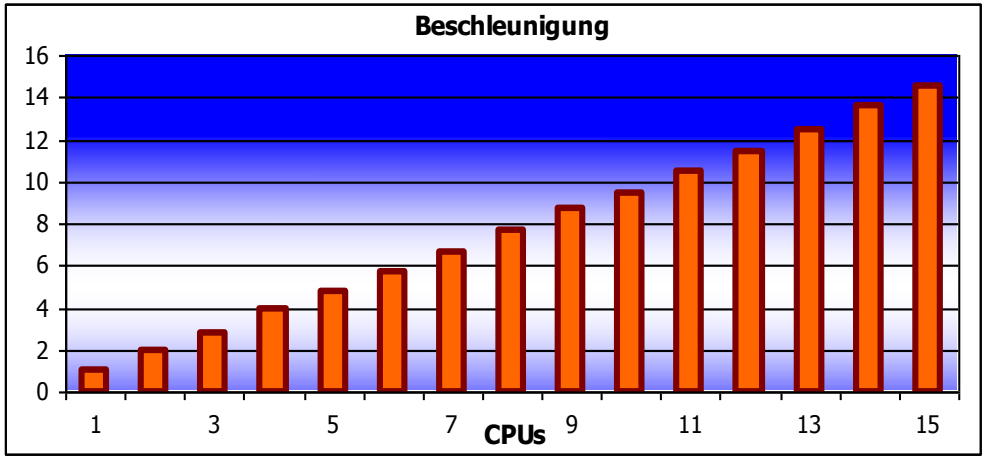


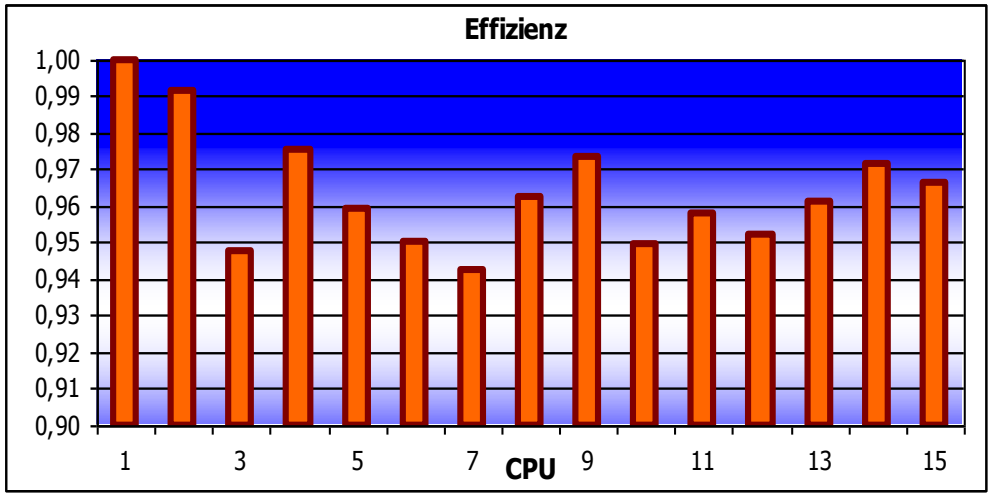
Bewertung paralleler Programme

Beschleunigung durch Parallelität (*Speedup*)

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

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





Auslastung (*Effizienz, Efficiency*)

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

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