B Betriebssysteme

B1 Einführung und Motivation

(a) Tanenbaum+wl

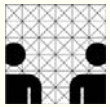
(b) Wolfinger

B2 Prozesse: Scheduling und Betriebsmittelzuteilung

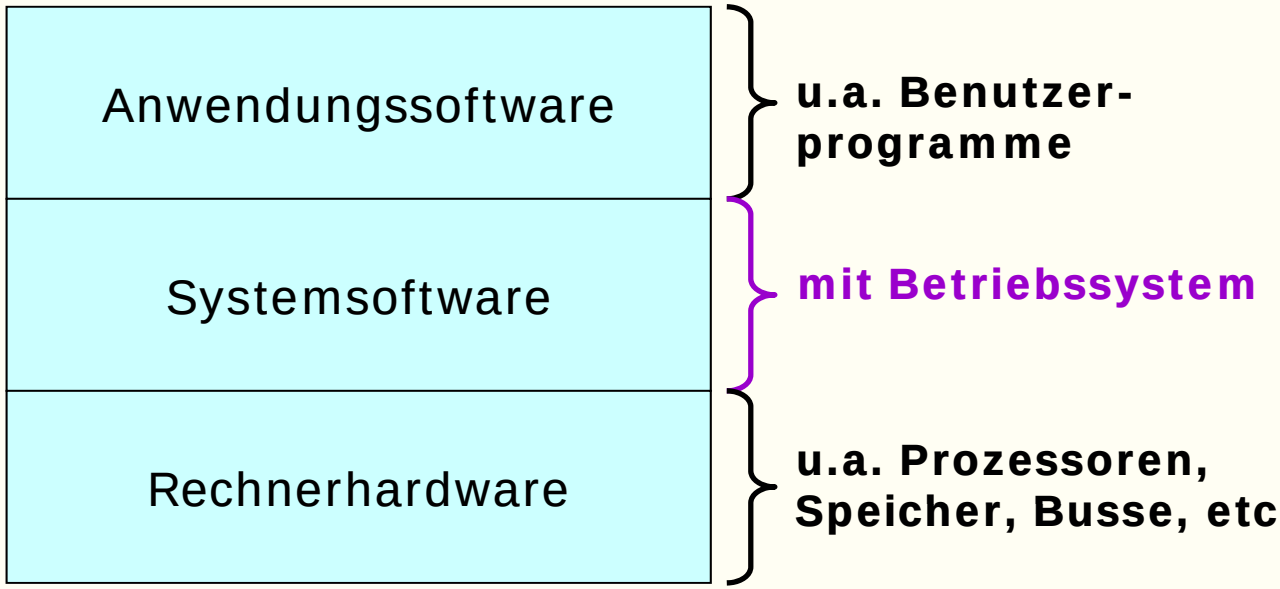
B3 Speicherverwaltung

B4 Dateisysteme

B5 Ein-/Ausgabe



B Betriebssysteme
B1 Einführung und Motivation (b)



Elementare Hardware-/Software-Schichtenstruktur

Zur Erinnerung (z.B. aus „Rechnerstrukturen“ o.ä.):
B1.1 Aufbau & Architektur konventioneller Rechensysteme

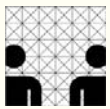
Der von Neumann-Rechner

Def. **Prozessor** (DIN 44300):

Eine Funktionseinheit innerhalb eines digitalen Rechensystems, die *Rechenwerk* und *Leitwerk* umfasst.

wobei:

- **Rechenwerk** (*arithmetic logic unit*; ALU) :
  Ausführung von Verarbeitungsbefehlen
  auch: *Operationswerk*, *Verarbeitungswerk*, *Datenprozessor* -
  W.K. Giloi: Rechnerarchitektur, Springer-Verlag, 1993 [Giloi 93]
- **Leitwerk**: (*control unit*):
  - Steuerung der Befehlsabfolge
  - Funktions- (Op-Code-) Entschlüsselung
  - Operationssteuerung (Signale an Rechen- bzw. E/A-Werk)
    auch : *Steuerwerk*, *Befehlsprozessor* [Giloi 93]
- **E/A-Werk**: (*input/output unit*, *I/0-unit*):
  - Ausführung von Ein-/Ausgabebefehlen



Basiskomponenten des von Neumann-Rechners

CPU (**c**entral **p**rocessing **u**nit) ≡ Datenprozessor ∪ Befehlsprozessor
auch: *Zentralprozessor*, *zentrale Recheneinheit* [Giloi 93]
(→ bitte *nicht*: Zentraleinheit!)

Def. **Speicher**:

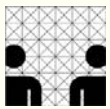
Eine Funktionseinheit innerhalb eines digitalen Rechensystems, die digitale Daten aufnimmt, aufbewahrt und abgibt.

Def. **Eingabeeinheit**:

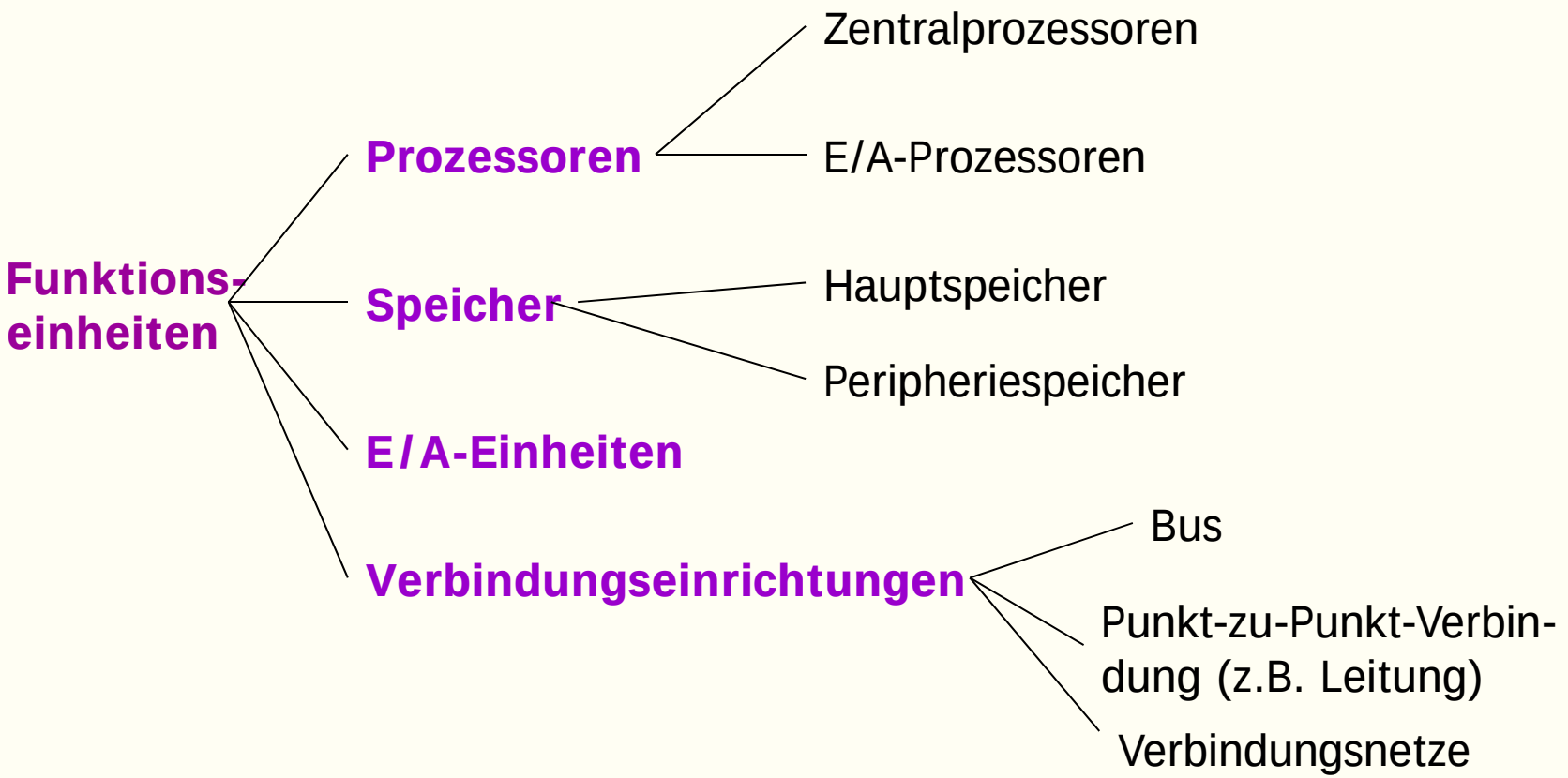
Eine Funktionseinheit innerhalb eines digitalen Rechensystems, mit der das System Daten von außen her aufnimmt.

Def. **Ausgabeeinheit**:

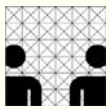
Eine Funktionseinheit innerhalb eines digitalen Rechensystems, mit der das System Daten, z.B. Rechenergebnisse, nach außen hin abgibt.



Hauptkomponenten von Rechensystemen

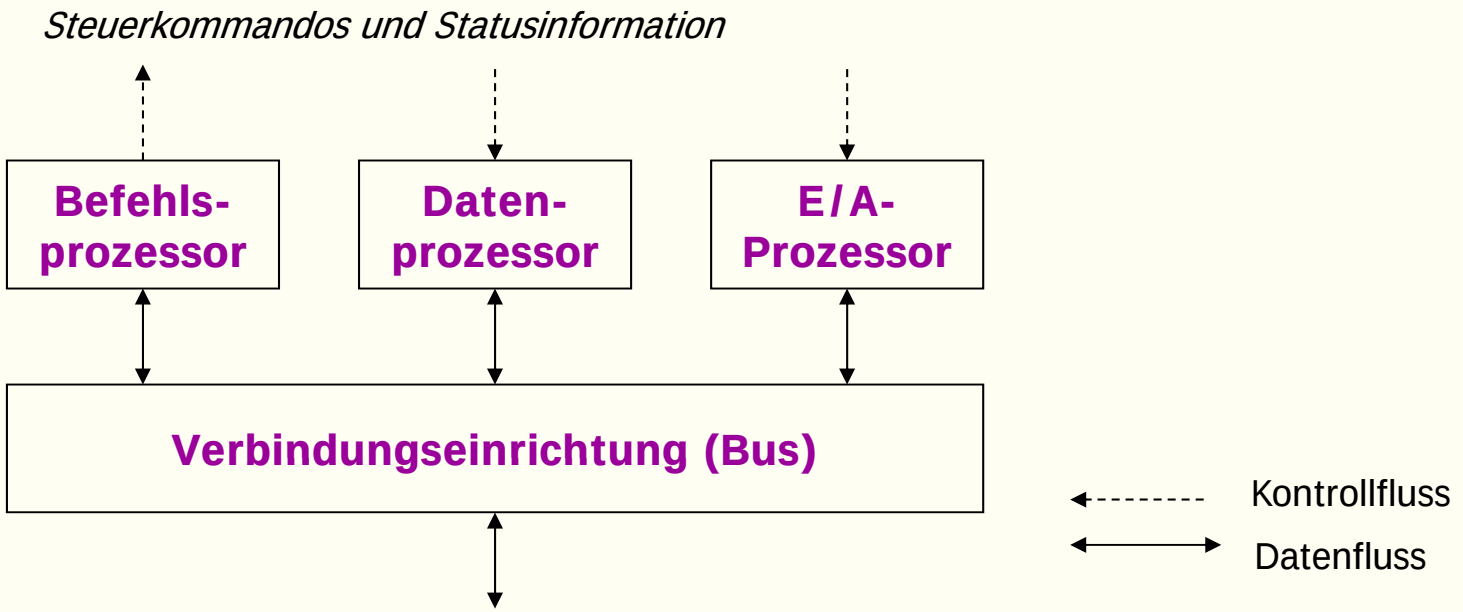


Funtionseinheiten: z.T. auch ***Hardware-Betriebsmittel*** genannt



Die Hauptkomponenten des von Neumann-Rechners und ihre Interaktionen

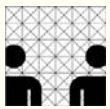
Zusammenspiel der Hauptkomponenten im **von Neumann-Rechner**
(auch: "von Neumann-Maschine", "Princeton Computer" [Giloi 93])



→ vgl. *Burks, Goldstine, von Neumann* : "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument". Report to the U.S. Army Ordonance Dept., 1946

Wesentliche Eigenschaften der von Neumann-Maschine

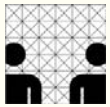
- Speicher für Programm und Daten inkl. Operanden sind Einheit ($\rightarrow$ Schreib-/Lesespeicher benötigt; Anweisungen und Operanden bzgl. Speicherung gleich behandelt $\Rightarrow$ *fehleranfälliges Konzept*, u.a. wg. zustandsabhängiger Interpretation gelesener Bitmusters)
- Spezieller Speicheraufbau: Einteilung in fortlaufend nummerierte Zellen (Nummern = Adressen)
- verarbeitet nur Maschinenprogramme (= Folgen elementarer Anweisungen)
- Befehlsaufbau: Operator + ggf. Adresse(n) des bzw. der Operanden
- Abweichungen von vorgegebener Befehlsreihenfolge durch *Sprungbefehle* (bedingt, unbedingt) $\rightarrow$ Sprungziel anstelle von Operandenangabe
- Verwendung von Binärzeichen/-signalen zur Darstellung aller Daten



Wesentliche Erweiterungen des von Neumann-Konzepts (in ca. 60 Jahren !)

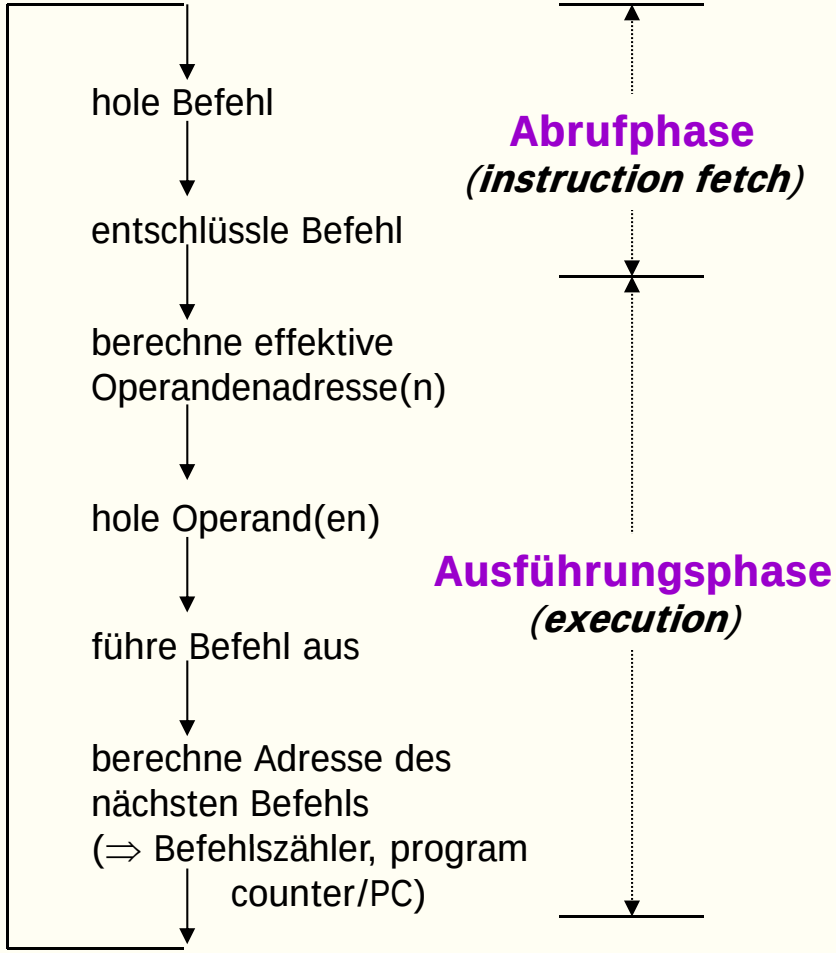
- Software unterstützt auch wesentlich andere Benutzungsschnittstellen und Betriebsformen
- gleichzeitig *mehrere* Prozesse in Funktionseinheiten (Füllung > 1)
- Mikroprogrammierung (z.B. von Rechenwerk, E/A-Werk)
- vielfältige Strukturierung von Befehlen (z.B. ≥ 2 Adressen möglich); viele verschiedene Befehlstypen mit unterschiedlichen Formaten [jedoch auch gegenläufige Tendenz → *RISC*, ***r****educed* ***i****nstruction* ***s****et* ***c****omputers* mit bewusst simplem Befehlssatz]
- Adressierung über Folge von Transformationen (z.B. Adressierung relativ zu Programmbeginn)
- Speicherhierarchie → Entschärfung des *„von Neumann Bottleneck"* (Flaschenhals)
- auch externe Signale (Unterbrechungen) werden durch Leitwerk mit berücksichtigt
- mehrfache Anordnung der von Neumann'schen Funktionseinheiten

⇒ *Feldrechner*, *Multiprozessorsysteme*, VS/*Rechnernetze* (s.u.) etc.



Zur Abarbeitung von Maschinenbefehlen

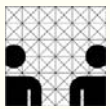
Der *(Maschinen-) Befehlszyklus:*



evtl. auch:

- Berücksichtigung von Unterbrechungswünschen
- triviale Ausführungsphasen (z.B. bei Sprungbefehlen)
- kein Holen von Operanden
- Holen von ≥ 2 Operanden

nota bene: hier kontrollflussorientierte Verarbeitung – andersartige Abläufe z.B. bei *daten*flussorientierter Verarbeitung möglich!



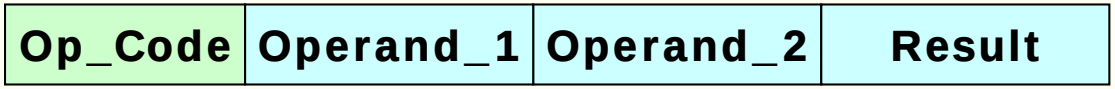
n-Adressmaschinen

Befehlsformate :

Op_Code: Befehlscode = Art/Typ des Befehls, z.B. **ADD(iere)**

- *n*-Adressmaschine :

- 3-Adressmaschine :

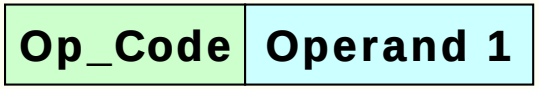


z.B.: **Result := ADD (Operand_1, Operand_2)**

- 2-Adressmaschine :

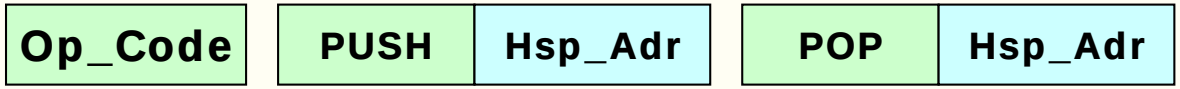
z.B.: **Operand_2 := ADD (Operand_1, Operand_2);** sog. Überdeckung (von Operand_2)

- 1-Adressmaschine :



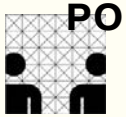
z.B.: **ACCUM := ADD (Operand_1, ACCUM);** sog. Überdeckung (von ACCUM als spez. Register)

- 0-Adressmaschine :



(„Stack-Rechner“)

z.B.: **Stack_1 := ADD (Stack_1, Stack_2);** Stack_i = i-ter Stack-Eintrag (von oben)
PUSH (Hsp_Adr) bringt Inhalt (Hsp_Adr) auf Stack;
POP (Hsp_Adr) bringt Inhalt des obersten Stack-Eintrags in den Hsp nach Adr.: Hsp_Adr.



Beispiel für Arbeitsweise einer 2-Adressmaschine

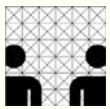
Aufgabe: Berechnung von

Z := (A – B * C) / (B + D / E)

Annahme: Überdeckung des 2. Operanden; H1 / H2 : Hilfsspeicherzellen;
>X< bezeichne Adresse der Zelle, die X beinhaltet

	Befehlsfolge		***Wirkung***
(1) →	>B<	>H1<	H1 := B
(2) *	>C<	>H1<	H1 := C * H1
(3) –	>A<	>H1<	H1 := A – H1
(4) →	>E<	>H2<	H2 := E
(5) /	>D<	>H2<	H2 := D / H2
(6) +	>B<	>H2<	H2 := B + H2
(7) /	>H1<	>H2<	H2 := H1 / H2
(8) →	>H2<	>Z<	Z := H2

Ergo: ***Gesamtaufwand*** für Lösung der Aufgabe:
8 Befehle mit
13 Leseaufträgen an Hsp./Cache
(jedoch nur **6 Leseaufträge**, sofern H1 / H2 : Register)



Beispiel für Arbeitsweise einer 1-Adressmaschine

Aufgabe: Berechnung von

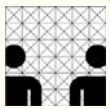
Z := (A – B * C) / (B + D / E)

Annahme : Überdeckung des Akkumulator-Inhaltes; AC: Akkumulator;
>X< bezeichne Adresse der Zelle, die X beinhaltet

	Befehlsfolge	***Wirkung***
(1) **lade**	>D<	AC := D
(2) **/**	>E<	AC := AC / E
(3) **+**	>B<	AC := AC **+** B
(4) **speichere**	>H1<	H1 := AC
(5) **lade**	>B<	AC := B
(6) *****	>C<	AC := AC ***** C
(7) **speichere**	>H2<	H2 := AC
(8) **lade**	>A<	AC := A
(9) –	>H2<	AC := AC – H2
(10) **/**	>H1<	AC := AC / H1
(11) **speichere**	>Z<	Z := AC

evtl. ersetzbar durch Befehl mit Wirkung : AC := A – AC

Ergo: ***Gesamtaufwand*** für Lösung der Aufgabe:
11 Befehle mit
8 Leseaufträgen an Hsp./Cache



Beispiel für Arbeitsweise einer 0-Adressmaschine

Aufgabe: Berechnung von

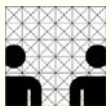
Z := (A – B * C) / (B + D / E)

Ausgangspunkt:
sog. Postfix-Notation (s.u.)
A B C * – B D E / + /

Annahme : In 0-Adressmaschine (auch: Keller-Rechner bzw. “Stack Computer“) exist. Befehle **PUSH**, **POP** (für Transfers Hsp ↔ Stack), **ADD**, **SUB**, **MUL**, **DIV**, … (arithmet. Befehle)

	Befehlsfolge		***Kellerinhalt***
(1)	**PUSH**	>A<	A
(2)	**PUSH**	>B<	A B
(3)	**PUSH**	>C<	A B C
(4)	**MUL**		A B*C
(5)	**SUB**		A-B*C
(6)	**PUSH**	>B<	A-B*C B
(7)	**PUSH**	>D<	A-B*C B D
(8)	**PUSH**	>E<	A-B*C B D E
(9)	**DIV**		A-B*C B D/E
(10)	**ADD**		A-B*C (B+D/E)
(11)	**DIV**		(A-B*C) / (B+D/E)
(12)	**POP**	>Z<	

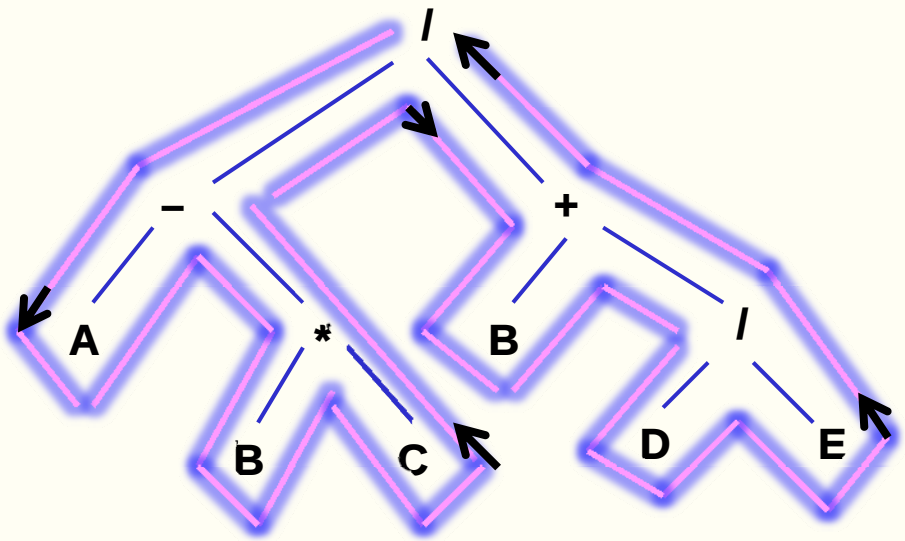
Ergo: ***Gesamtaufwand*** für Lösung der Aufgabe: **12 Befehle** mit **6 Leseaufträgen** an Hsp./Cache



Herleiten der Postfix-Notation zu gegebenem Ausdruck

Aufgabe: Berechnung von

Z := (A – B * C) / (B + D / E)



Resultat:
Ausdruck in Postfix-Notation

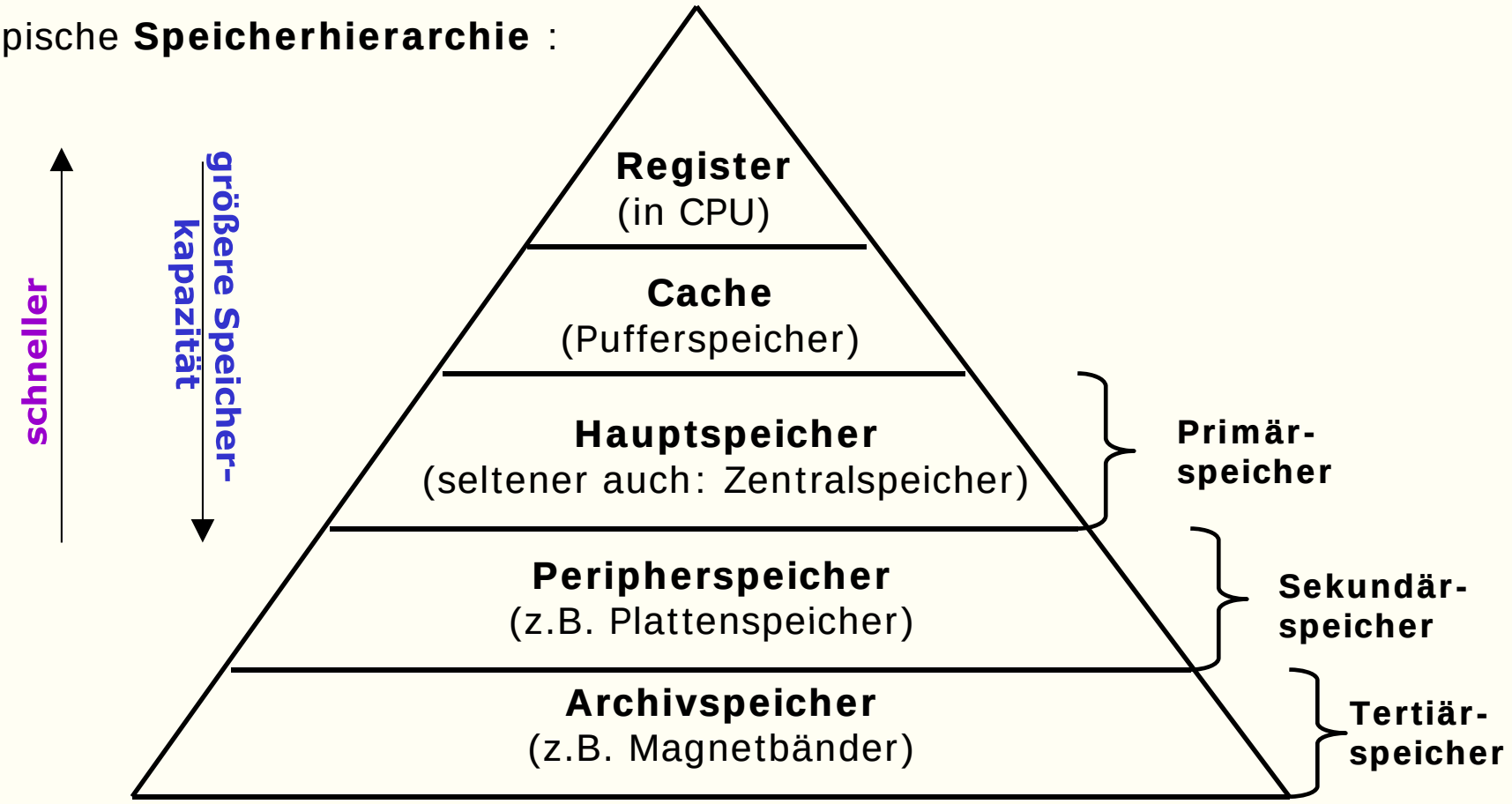
A B C * – B D E / + /

Speicherhierarchie

Gründe für Speicherhierarchien:

- schneller Speicher ist relativ teuer
- großer Speicher ist relativ langsam (große Zugriffszeiten)

⇒ Typische **Speicherhierarchie** :



B 1.2 Betriebssysteme: Historische Entwicklung und einige Grundkonzepte

Systemsoftware als “Bindeglied” zwischen Programmen und Hardware

Das Doppelgesicht der Systemsoftware:

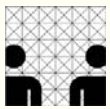
PROGRAMM

Barriere
der Systemsoftware

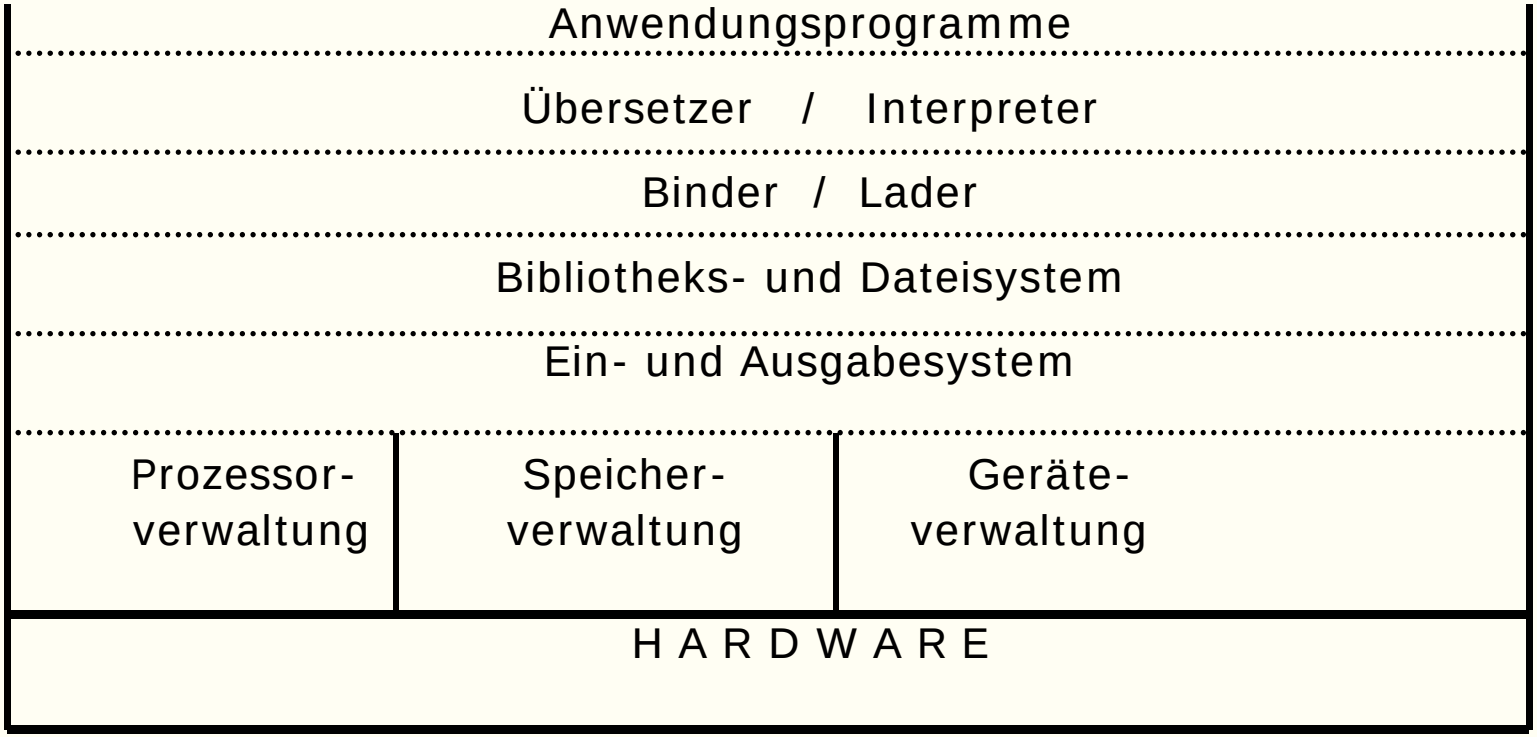
HARDWARE

Negativ:
- Die Systemsoftware verbraucht wertvolle Betriebsmittel.
- Die Systemsoftware zwingt den Programmierer zur Beachtung sehr vieler Vorschriften.

Positiv:
+ Entlastung des Programmierers von den Idiosynkrasien der technischen Gerätereal isation.
+ Unterstützung des Programmierers bei der Erstellung und Verwaltung von Programmen.



Grobes Modell eines Rechensystems

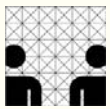


Fragen:

(i) Welche der obigen Tätigkeiten gehören in ein Betriebssystem?

(ii) Wohin legt man eine „Sicherheitsschicht“?

(iii) Nach welchen allgemeinen Konstruktionsprinzipien werden Systeme erstellt?



Aufgaben eines Betriebssystems

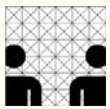
Die zwei **Grundaufgaben** eines Betriebssystems (BS):

I. Ein Betriebssystem verbirgt die Hardwaredetails und stellt dem Nutzer eine „angenehme“ Arbeitsumgebung zur Verfügung. Dies führt zur Bildung plattformunabhängiger Betriebssysteme, z.B. Solaris, Windows NT, Windows Vista/7, OS400, Symbian, Android, …
→ **BS als „virtuelle Maschine“**

II. Ein Betriebssystem verwaltet Betriebsmittel im Auftrag von Nutzern.
→ **BS als „Betriebsmittelverwalter“**

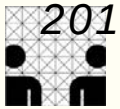
Eine Listung der **Einzelaufgaben** eines Betriebssystems führt zu:

- Buchhaltung,
- Verwaltung von Betriebsmitteln,
- Erhöhung der Benutzbarkeit,
- Erhöhung der Zuverlässigkeit,
- Minderung der Maschinenabhängigkeit,
- Automatischer Operateur,
- Erleichterung menschlicher Kommunikation.



Einige bekannte Betriebssysteme

1955: GM OS (Das erste Betriebssystem ?)
1961: CTSS
1964: **OS/360** (Ankündigung, erste Einführung 1967)
1965: Multics
1967: 'THE'- multiprogramming system
1969: TENEX
1969: **Unix**
1973: RT-11
1975: TOPS-20
1978: **Apple DOS 3.1**
1981: **MS-DOS**
1984: **Macintosh OS**
1993: **Windows NT 3.1**
1995: **OS/390**
2000: **Windows 2000**
2002: **MAC OS X**
2003: Fedora Core Linux
2006/7: **Windows Vista**
2009/10: **Android**, Windows 7 *(mobile/phone)* / *MeeGo* / ...
2012/13 Windows 8...



Einige Daten zur Entwicklung von UNIX

1961 CTSS ("Compatible Timesharing System")

1965 Start von MULTICS

1969 Ken Thompson erstellt erste Version von **Unix** auf einer PDP-7.

1970 Ken Thompson überträgt Unix von der PDP-7 auf eine PDP-11, die Größe des Systems beträgt etwa 12.000 Byte.

1973 Unix wird weitgehend in C neu programmiert, etwa 10.000 Codezeilen in C und etwa 1000 Codezeilen in Assembler.

1976 Unix Version 6

1978 BSD-Unix; Unix Version 7

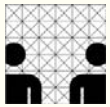
1983 Unix System V

1985 System V Interface Definition

1991 **Solaris** 2, Linux

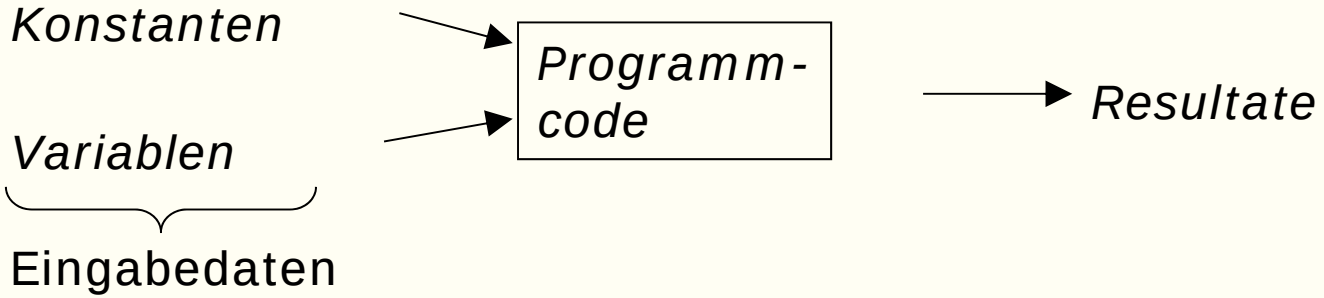
1998 Sun Solaris 7

2001 **Linux** 2.4 –--



Prozesse

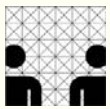
- Programmablauf:



Definitionen "**Prozess**":

Def. I: (Rechen-) Prozess ≅ Ablauf eines Programms, sofern dieser durch das Betriebssystem verwaltet wird.

Def. II: Prozess ≅ Folge von Verarbeitungsschritten, deren erster begonnen, deren letzter aber noch nicht abgeschlossen ist.

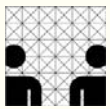


Prozesse

Bemerkungen:

- Prozess benötigt **Prozessor** zur Ausführung
- Jedem Prozess ist ein **Prozessadressraum** zugeordnet, u.a. mit
  - Programmcode
  - Konstanten
  - prozessspezifischen Variablen

  nota bene: Momentanzustand der Prozessausführung i.d.R. in dedizierten Betriebssystem-Tabellen, als sog. Prozesstabelle geführt (beinhaltet u.a. aktuelle Registerinhalte für Prozess, insbesondere Befehlszähler).
- sequentieller Prozess: Prozess mit *linearer* Folge von Verarbeitungsschritten
- mobile Prozesse??

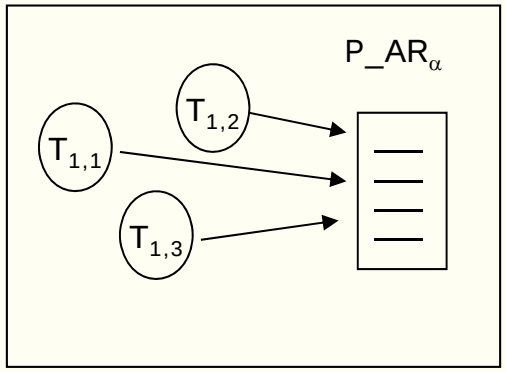


Leichtgewichtsprozesse / “Threads“

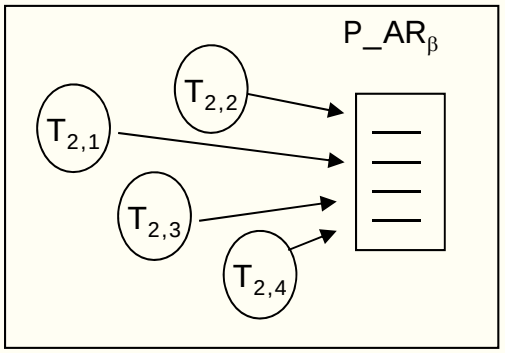
Def. „**Thread**“ (auch „*Leichtgewichtsprozess*“, engl. „*light weight process*“):

Programmabläufe („Prozesse“), die

- durch das Betriebssystem verwaltet werden,
- einen gemeinsamen Prozessadressraum verwenden.



Prozess 1
(mit Threads $T_{1,i}$)



Prozess 2
(mit Threads $T_{2,j}$)

P_AR: Prozess-Adressraum

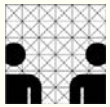
Gründe für „Threads“

- effizienter Zugriff auf Daten des gemeinsamen Prozess-Adressraumes (P_AR)
- deutlich *aufwandsärmere* Erzeugung von Threads als von Prozessen

  → Threads auch für kurzfristige Aufgaben geeignet
- bei Prozesswechsel (CPU-Vergabe an neuen Prozess) ist im allg. P_AR nicht Hauptspeicher-resident

  → Zeitaufwand (z.B. für „Paging“);

  bei Thread-Wechsel (zwischen Threads eines gemeinsamen Prozesses) befindet sich hingegen i.d.R. zumindest ein Teil des benötigten P_AR bereits im Hauptspeicher



Benutzerprozesse vs. Systemprozesse

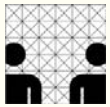
Def.

- **Benutzerprozesse** (BP): Prozesse, die Aufträge von Benutzern abwickeln (insbes. aus der Ausführung benutzerspezifischer Programme oder von Anwendungsprogrammen resultierend).
- **Systemprozesse** (SP): Prozesse, die ausgewählte Dienstleistungen des Betriebssystems (BS) erbringen.

⇨

Merkmale von *Systemprozessen*:

- häufig im Besitz spezieller Rechte
- im Systemmodus lauffähig
- sind wesentliche Betriebssystemkomponenten (neben dem ***Betriebssystemkern*** ≅ nicht als Prozesse realisierte BS-Komponenten)



Dateien

Datei:

längerfristig zu speichernde geordnete Datenmenge bestehend aus logisch zusammenhängender Menge von (Daten-) Sätzen.

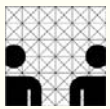
... oder *allgemeiner*: Einheit der Speicherung beliebiger Daten auf einem persistenten (dauerhaften) Speicher [NeS98].

Eigenschaften von Dateien:

- i.d.R. benutzerdefiniert
- sehr unterschiedliche Arten von Inhalten (z.B. Programmdaten, Eingabedaten, Festbilder, Videosequenzen, u.v.a.m.)
- Abstraktion von der physikalischen Speicherung, Konzept zur Realisierung von Geräte-/Speicherunabhängigkeit
- Organisation von Dateien in *Katalogen* bzw. *Verzeichnissen* (engl.: *directories*) → u.a. Gruppierung von Dateien in Verzeichnissen und Zugriff auf Dateien über Verzeichnisse
- benutzerspezifische Zugriffsrechte

Dateiverwaltung als wesentliche Aufgabe von Betriebssystemen

(⇨ zu Details vgl. B4)



Kommandointerpretierer / “Shell“

Benutzungsschnittstellen von Betriebssystemen:

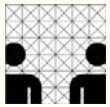
- graphische Oberfläche (GUI), oder
- Kommandoschnittstelle

Bei Kommandoschnittstelle kann ein Kommandointerpretierer (auch: „Shell“ bei Betriebssystemen der UNIX-Familie)

- beim Anmelden eines Benutzers gestartet werden
- auf Eingabe eines Kommandos warten
- die kommandospezifischen Betriebssystemoperationen initiieren, z.B. mittels geeigneter Betriebssystemaufrufe.

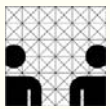
Typische **Klassen von Betriebssystemaufrufen**:

- Prozessverwaltung
- Signale
- Datei(- und Geräte-)verwaltung
- Katalog- und Dateisystemverwaltung
- Schutzmechanismen
- Zeitverwaltung



Beispiele für Betriebssystemaufrufe

Aufrufklasse	Beispiele
Prozess-verwaltung	• Ersetzen des Prozessadressraums • Beendigung der Prozessausführung mit Statusrückmeldun
Signale	• Senden eines Signals an einen Prozess • Suspendierung des Anrufers bis nächstes Signal eintrifft
Datei-verwaltung	• Erzeugen einer Datei • Schließen einer geöffneten Datei • Lesen von Daten aus einer Datei in einen Puffer
Katalog- und Dateisystem-verwaltung	• Entfernen eines Katalogeintrags • Wechsel des aktuellen Arbeitskatalogs
Schutz-mechanismen	• Veränderung der (Zugriffs-)Schutz-Bits einer Datei • Wechsel des Dateieigentümers (Benutzer und/oder Gruppe)
Zeitverwaltung	• Setzen der Zeit des letzten Zugriffs auf eine Datei • Lesen der angefallenen Benutzer- bzw. Systemzeiten



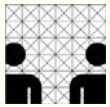
Sicht I auf Betriebssysteme:
Betriebssystem als „Betriebsmittelverwalter“

Aufgaben des Betriebssystems bei Sicht I:

- Ermittlung des Betriebsmittel (BM)-Bedarfs von Prozessen → evtl. Problem: Bedarf a priori nicht präzise bekannt
- Allokation der BM zu Prozessen (auf Anforderung hin) → u.a. Fairness und Effizienz bei BM-Vergabe
- Ermittlung und ggf. Abrechnung der beanspruchten BM („accounting“)
- Behandlung von Problemen und Konflikten bei Inanspruchnahme von BM → u.a. Behandlung von Verklemmungs- und Engpass-Situationen.

Betriebsmittelverwaltung u.a. erschwert bei:

- großer Auswahl von um BM konkurrierenden Prozessen
- unpräzise spezifiziertem BM-Bedarf
- hoher Auslastung eines BM (zahlreiche zeitlich benachbarte Zugriffe und/ oder geringe „Kapazität“ des BM, wie Speicher oder Verarbeitungsleistung)
- BM-Allokation in verteilten Systemen! (→ vgl. Vert. Syst. & Rechnernetze)



Sicht II auf Betriebssysteme:
Betriebssystem als „virtuelle Maschine“

Aufgaben des Betriebssystems bei Sicht II:

- Details der zugrunde liegenden Rechnerhardware den Benutzern verbergen (z.B. Unsichtbarkeit der Speicherhierarchie, der angeschlossenen E/A-Geräte und Netze sowie der benutzten Peripherspeicher etc.)
- Details der gleichzeitigen Mehrfachbenutzung des Rechners und seiner Komponenten dem einzelnen Benutzer verbergen
- Realisierung von (weitestgehender) Hardwareunabhängigkeit für die zu erstellenden Programme

➡ durch Abstraktion von Hardware-Details:

Nutzung des Rechners deutlich komfortabler (Betriebssystem übernimmt die Abbildung*) der benutzernahen Dienste – z.B. Zugriff auf Dateien – auf die elementaren, systemnahen Basisfunktionen der Hardwarekomponenten).

---

*) *nota bene*: Diese Abbildung erfolgt aus Komplexitätsgründen im Allgemeinen nicht direkt, sondern durch Abbildung auf sukzessive elementarere Dienste (→ „Diensthierarchien in Betriebssystemen“)