

# Systemprogrammierung

Grundlagen von Betriebssystemen

## Teil B – V.1 Rechnerorganisation: Virtuelle Maschinen

16. Mai 2024

Rüdiger Kapitza

(© Wolfgang Schröder-Preikschat, Rüdiger Kapitza)



Lehrstuhl für Informatik 4  
Systemsoftware



Friedrich-Alexander-Universität  
Technische Fakultät

## Gliederung

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

## Agenda

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

## Lehrstoff

- Rechensysteme als eine **Schichtenfolge** von Maschinen
  - die eine **funktionale Hierarchie** [7] von spezifischen Maschinen zur Ausführung von Programmen darstellt
  - wobei manche dieser Maschinen nicht in Wirklichkeit vorhanden sind, sein müssen oder sein können
  - die somit jeweils als eine **virtuelle Maschine** [11] in Erscheinung treten
- **Abstraktionshierarchie** als Basis Rechnerkonstruktionen
  - in der die einzelnen Schichten durch **Prozessoren** implementiert werden, die vor (*off-line*) oder zur (*on-line*) Programmausführungszeit wirken
  - wobei ein Prozessor als **Übersetzer** oder **Interpreter** ausgelegt ist
- Platz für das **Betriebssystem** in dieser Hierarchie ausmachen
  - erkennen, dass ein Betriebssystem ein spezieller Interpreter ist und den Befehlssatz wie auch die Funktionalität einer CPU erweitert
  - die **Symbiose** insbesondere von Betriebssystem und CPU verinnerlichen
- Grundlagen eines „Weltbilds“ legen, das zentral für SP sein wird

## Gliederung

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

## Verschiedenheit zwischen Quell- und Zielsprache

Faustregel:  $\left\{ \begin{array}{l} \text{Quellsprache} \rightarrow \text{höheres} \\ \text{Zielsprache} \rightarrow \text{niedrigeres} \end{array} \right\}$  Abstraktionsniveau

### Semantische Lücke (*semantic gap*, [14])

*The difference between the complex operations performed by high-level constructs and the simple ones provided by computer instruction sets.*

*It was in an attempt to try to close this gap that computer architects designed increasingly complex instruction set computers.*

- Kluft zwischen gedanklich Gemeintem und sprachlich Geäußertem

## Schichtenstruktur

### Semantische Lücke

## Schichtenstruktur

### Fallstudie

## Beispiel: Matrizenmultiplikation

Problemraum



(Mathematik)



Lösungsraum



(Informatik)

- „gedanklich gemeint“ ist ein Verfahren aus der linearen Algebra
- „sprachlich geäußert“ auf verschiedenen Ebenen der **Abstraktion**

## Ebene mathematischer Sprache: Lineare Algebra

- Multiplikation von zwei  $2 \times 2$  Matrizen:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Zwei Matrizen werden multipliziert, indem die Produktsummenformel auf Paare aus einem Zeilenvektor der ersten und einem Spaltenvektor der zweiten Matrix angewandt wird.

- Produktsummenformel für  $C = A \times B$ :  $C_{i,j} = \sum_k A_{ik} \cdot B_{kj}$

SP

Schichtenstruktur

B - V1 / 7

SP

Schichtenstruktur

B - V1 / 8

## Ebene informatischer Sprache: C

- Skalarprodukt oder „inneres Produkt“, Quellmodul (multiply.c):

```
1 typedef int Matrix [N][N];
2
3 void multiply(in Matrix a, in Matrix b, out Matrix c) {
4     unsigned int i, j, k;
5     for (i = 0; i < N; i++)
6         for (j = 0; j < N; j++) {
7             c[i][j] = 0;
8             for (k = 0; k < N; k++)
9                 c[i][j] += a[i][k] * b[k][j];
10        }
11 }
```

- **Konkretisierung** für zwei  $N \times N$  Matrizen:  $c = a \times b$ 
  - ausgelegt als Unterprogramm: Prozedur  $\mapsto$  C function
  - insgesamt sechs Varianten (d.h., Schleifenanordnungen)
    - $\{ijk, jik, ikj, jki, kij, kji\}$ : funktional gleich, nichtfunktional ggf. ungleich

SP

Schichtenstruktur

B - V1 / 9

## Ebene informatischer Sprache: ASM [8, 4]

```
1 .file "multiply.c"
2 .text
3 .p2align 4,,15
4 .globl multiply
5 .type multiply,@function
6 multiply:
7     pushl %ebp
8     movl %esp,%ebp
9     pushl %edi
10    pushl %esi
11    pushl %ebx
12    subl $4,%esp
13    movl 16(%ebp),%esi
14    movl $0,-16(%ebp)
15    .L2:
16    movl 8(%ebp),%edi
17    xorl %ebx,%ebx
18    addl -16(%ebp),%edi
19    .p2align 4,,7
20    .p2align 3
21    .L4:
22    movl 12(%ebp),%eax
23    xorl %edx,%edx
24    movl $0,(%esi,%ebx,4)
25    leal (%eax,%ebx,4),%ecx
26    .p2align 4,,7
27    .p2align 3
28    .L3:
29    movl (%ecx),%eax
30    addl $400,%ecx
31    imull (%edi,%edx,4),%eax
32    addl $1,%edx
33    addl %eax,(%esi,%ebx,4)
34    cmpl $100,%edx
35    jne .L3
36    addl $1,%ebx
37    cmpl $100,%ebx
38    jne .L4
39    addl $400,-16(%ebp)
40    addl $400,%esi
41    cmpl $40000,-16(%ebp)
42    jne .L2
43    addl $4,%esp
44    popl %ebx
45    popl %esi
46    popl %edi
47    popl %ebp
48    ret
49    .size multiply, .-multiply
50    .ident "GCC: (Debian 4.3.2-1.1) 4.3.2"
51    .section .note.GNU-stack,"",@progbits
```

- **Kompilation** der Quelle in ein semantisch äquivalentes Programm

- Trick: Übersetzung der Quelle vor dem **Assemblieren** beenden
  - Übersetzung<sup>1</sup> von multiply.c mit `-DN=100: C function`  $\mapsto$  ASM/x86

<sup>1</sup> gcc -O -m32 -fomit-frame-pointer -fno-pic -static -S

SP

Schichtenstruktur

B - V1 / 10



# Mehrebenenmaschinen

## Maschinenhierarchie

### Hierarchie virtueller Maschinen [13, S. 3]

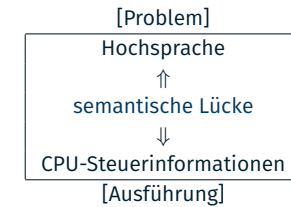
- **Interpretation und Übersetzung** (Kompilation, Assemblieren):

Ebene		
$n$	virtuelle Maschine $M_n$ mit Maschinensprache $S_n$	Programme in $S_n$ werden von einem auf einer tieferen Maschine laufenden Interpreter gedeutet oder in Programme tieferer Maschinen übersetzt
$\vdots$	$\vdots$	$\vdots$
2	virtuelle Maschine $M_2$ mit Maschinensprache $S_2$	Programme in $S_2$ werden von einem auf $M_1$ bzw. $M_0$ laufenden Interpreter gedeutet oder nach $S_1$ bzw. $S_0$ übersetzt
1	virtuelle Maschine $M_1$ mit Maschinensprache $S_1$	Programme in $S_1$ werden von einem auf $M_0$ laufenden Interpreter gedeutet oder nach $S_0$ übersetzt
0	reale Maschine $M_0$ mit Maschinensprache $S_0$	Programme in $S_0$ werden direkt von der Hardware ausgeführt

- Techniken, die einander unterstützend – teils sogar „symbiotisch“ – Verwendung finden, um Programme zur Ausführung zu bringen

### Aufgabenstellung $\mapsto$ Programmlösung

- das Ausmaß der semantischen Lücke gestaltet sich fallabhängig:
  - bei gleich bleibendem Problem mit der Plattform (dem System)
  - bei gleich bleibender Plattform mit dem Problem (der Anwendung)



- der Lückenschluss ist ganzheitlich zu sehen und auch anzugehen
  - Schicht für Schicht die innere (logische) Struktur des Systems herleiten
  - das System, das die Lücke schließen soll, als Ganzes als „Bild“ erfassen
    - hinsichtlich benötigter funktionalen und nicht-funktionalen Eigenschaften
- Kunst der kleinen Schritte: seman. Lücke schrittweise schließen
  - durch hierarchisch angeordnete virtuelle Maschinen Programmlösungen auf die reale Maschine herunterbrechen [12]
  - Prinzip *divide et impera* („teile und herrsche“)
    - einen „Gegner“ in leichter besiegbare „Untergruppen“ aufspalten

### Abstrakter Prozessor

Kompilierer (*compiler*) und Interpreter

- jede einzelne Ebene (d.h., Schicht) in der Hierarchie wird durch einen spezifischen Prozessor implementiert:

#### Kompilator *lat.* (Zusamenträger)

- ein **Softwareprozessor**, transformiert in einer *Quellsprache* vorliegende Programme in eine semantisch äquivalente Form einer *Zielsprache*
  - {Ada, C, C++, Eiffel, Modula, Fortran, Pascal, ...}  $\mapsto$  Assembler
  - aber ebenso: C++  $\mapsto$  C  $\mapsto$  Assembler

#### Interpreter *lat.* (Ausleger, Erklärer, Deuter)

- ein **Hard-, Firm- oder Softwareprozessor**, der die Programme direkt ausführt  $\leadsto$  ausführbares Programm (*executable*)
  - z.B. Basic, Perl, C, `sh(1)`, x86
- ggf. **Vorübersetzung** durch einen Kompilierer, um die Programme in eine für die Interpretation günstigere Repräsentation zu bringen
  - z.B. Pascal P-Code, Java Bytecode, x86-Befehle

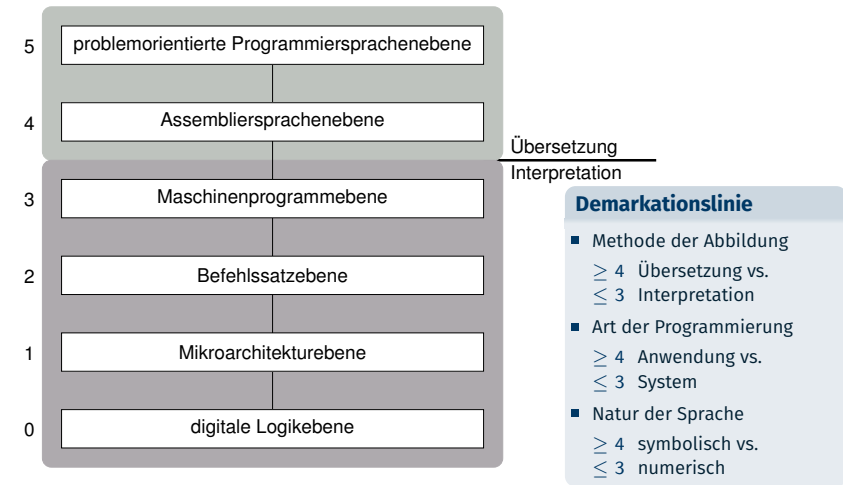
- also abstrakte oder reale Prozessoren, die vor beziehungsweise zur Ausführungszeit des Programms wirken, das sie verarbeiten

# Mehrebenenmaschinen

## Maschinen und Prozessoren

# Schichtenfolge in Rechensystemen I

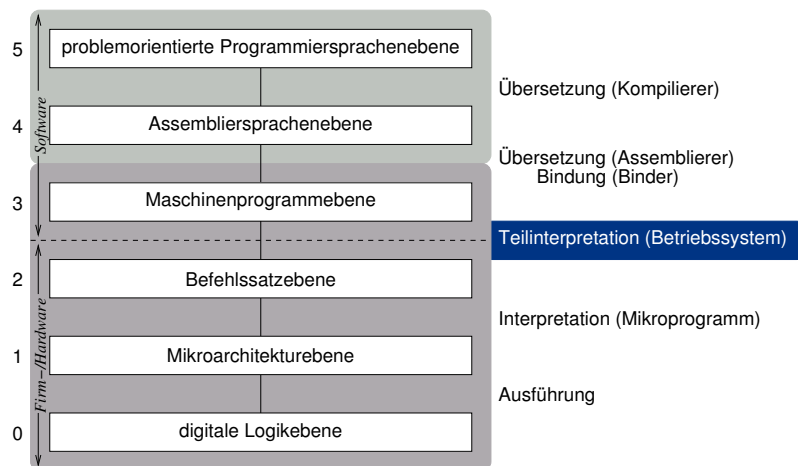
in Anlehnung an [12]



- Schichten der Ebene<sub>[4,5]</sub> sind nicht wirklich existent sondern
  - werden durch Übersetzung aufgelöst und auf tiefere Ebenen abgebildet
  - so dass am Ende nur ein Maschinenprogramm (Ebene<sub>3</sub>) übrig bleibt

# Schichtenfolge in Rechensystemen II

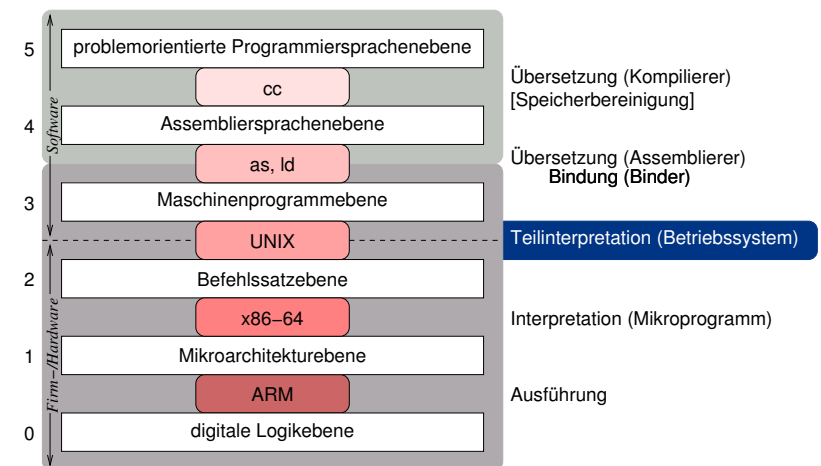
in Anlehnung an [12]



- Schichten der Ebene<sub>[0,2]</sub> liegen normalerweise nicht in SW vor
  - sie können aber in Software simuliert/emuliert oder virtualisiert werden
  - dadurch lassen sich Rechensysteme grundsätzlich **rekursiv** organisieren

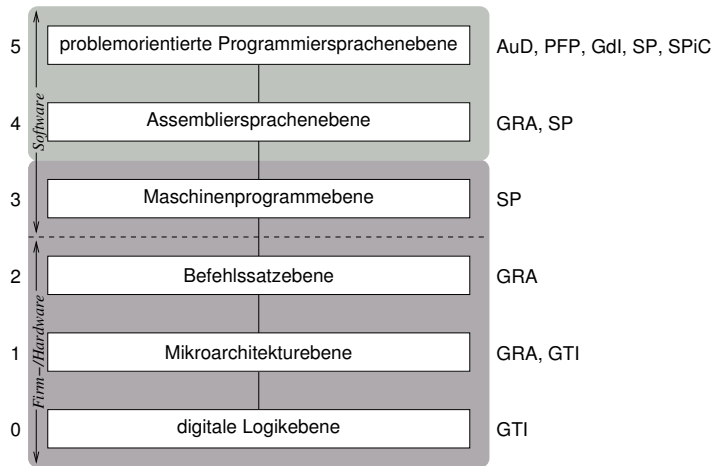
# Schichtenfolge in Rechensystemen III

Entitäten



- RISC auf Ebene<sub>1</sub> und gegebenenfalls (hier) CISC auf Ebene<sub>2</sub>
  - nach außen „complex“, innen aber „reduced instruction set computer“
  - Intel Core oder Haswell ↔ AMD Bulldozer oder Zen (ARM)

## Schichtenfolge eingebettet im Informatikstudiengang



- die Schicht auf Ebene<sub>4</sub> ist auch hier eher nur logisch existent ☺
  - Programmierung in Assemblersprache hat (leider) an Bedeutung verloren
  - Prinzipien werden in GRA vermittelt [5], in SP nur bei Bedarf behandelt

SP

Mehrebenenmaschinen

B - V.1 / 21

## Mehrebenenmaschinen

### Entvirtualisierung

## Abbildung der Schichten

### Auflösung der Abstraktionshierarchie

- Schichten der Ebene<sub>[3,5]</sub> repräsentieren **virtuelle Maschinen**, die auf die eine **reale Maschine** (Ebene<sub>[0,2]</sub>) abzubilden sind
  - dabei werden diese Schichten „entvirtualisiert“, aufgelöst und zu einem **Maschinenprogramm** „verschmolzen“
  - dieser Vorgang hängt stark ab von der Art einer virtuellen Maschine<sup>2</sup>
- **Übersetzung**
  - aller Befehle des Programms, das der Ebene<sub>i</sub> zugeordnet ist
  - in eine semantisch äquivalente Folge von Befehlen der Ebene<sub>j</sub>, mit  $j \leq i$
  - dadurch **Generierung** eines Programms, das der Ebene<sub>j</sub> zugeordnet ist
- **Interpretation**
  - total**
    - aller Befehle des Programms, das der Ebene<sub>i</sub> zugeordnet ist
  - partiell**
    - nur der Befehle des Programms, die der Ebene<sub>i</sub> zugeordnet sind
    - wobei das Programm der Ebene<sub>k</sub>,  $k \geq i$ , zugeordnet sein kann
- durch **Ausführung** eines Programms der Ebene<sub>j</sub>, mit  $j \leq i$

<sup>2</sup>vgl. insb. [10]: die Folien sind Teil des ergänzenden Materials zu SP.

SP

Mehrebenenmaschinen

B - V.1 / 22

## Abbildung durch Übersetzung

Ebene<sub>5</sub>  $\mapsto$  Ebene<sub>4</sub> (Kompilation)

- Ebene<sub>5</sub>-Befehle „1:N“,  $N \geq 1$ , in Ebene<sub>4</sub>-Befehle übersetzen
  - einen Hochsprachenbefehl als mögliche Sequenz von Befehlen einer Assemblersprache implementieren
    - eine **semantisch äquivalente Befehlsfolge** generieren
- im Zuge der Transformation ggf. Optimierungsstufen durchlaufen

Ebene<sub>4</sub>  $\mapsto$  Ebene<sub>3</sub> (Assemblieren)

- Ebene<sub>4</sub>-Befehle „1:1“ in Ebene<sub>3</sub>-Befehle übersetzen
  - ein **Quellmodul** in ein **Objektmodul** umwandeln
  - mit **Bibliotheken** zum Maschinenprogramm zusammenbinden
    - ein **Lademodul** erzeugen
- dabei den symbolischen Maschinenkode (d.h., die Mnemone) auflösen
  - in binären Maschinenkode umwandeln
    - ADD EAX (Mnemon)  $\mapsto$  05<sub>16</sub> (Hexadezimalcode)  $\mapsto$  00000101<sub>2</sub> (Binärkode)
    - hier: Beispiel für den Befehlssatz x86-kompatibler Prozessoren

SP

Mehrebenenmaschinen

B - V.1 / 23

## Abbildung durch Interpretation

Ebene<sub>3</sub> ↦ Ebene<sub>2</sub> (*partielle* Interpretation, Teilinterpretation)

- Ebene<sub>3</sub>-Befehle typ- und zustandsabhängig verarbeiten:
  - i als Folgen von Ebene<sub>2</sub>-Befehlen ausführen
    - **Systemaufrufe** annehmen und befolgen, sensitive Ebene<sub>2</sub>-Befehle emulieren
    - synchrone/asynchrone **Unterbrechungen** behandeln
  - ii „1:1“ auf Ebene<sub>2</sub>-Befehle abbilden (nach unten „durchreichen“)
- ein Ebene<sub>3</sub>-Befehl aktiviert im Fall von i ein Ebene<sub>2</sub>-Programm
  - verursacht durch eine **Ausnahmesituation**, die durch Ebene<sub>2</sub> erkannt und zur Behandlung an ein Programm der Ebene<sub>2</sub> „hochgereicht“ wird
  - Ebene<sub>2</sub> stellt eine Falle (*trap*), bedient von einem Ebene<sub>2</sub>-Programm

Ebene<sub>2</sub> ↦ Ebene<sub>1</sub> (Interpretation)

- Ebene<sub>2</sub>-Befehle als Folgen von Ebene<sub>1</sub>-Aktionen ausführen
  - **Abruf- und Ausführungszyklus** (*fetch-execute-cycle*) der CPU
- ein Ebene<sub>2</sub>-Befehl löst Ebene<sub>1</sub>-Steueranweisungen aus

## Mehrebenenmaschinen

### Ausnahmesituation

## Zeitpunkte der Abbildungsvorgänge

Bezugspunkt ist das jeweils zu „prozessierende“ Programm:

- **vor Laufzeit** (Ebene<sub>5</sub> ↦ Ebene<sub>3</sub>) ∼ **statisch**
  - Vorverarbeitung (*preprocessing*)
  - Vorübersetzung (*precompilation*)
  - Übersetzung: Kompilation, Assemblieren
  - Binden (*static linking*)
- **zur Laufzeit** (Ebene<sub>5</sub> ↦ Ebene<sub>1</sub>) ∼ **dynamisch**
  - bedarfsorientierte Übersetzung (*just in time compilation*)
  - Binden (*dynamic linking*)
  - bindendes Laden (*linking loading, dynamic loading*)
  - Teilinterpretation
  - Interpretation

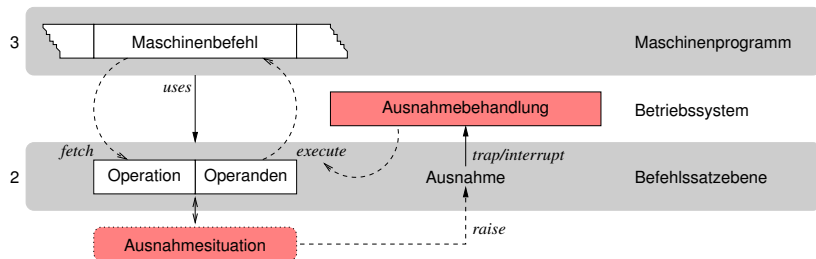
### Betriebssysteme entvirtualisieren zur Laufzeit

↔ dynamisches Binden, bindendes Laden, Teilinterpretation

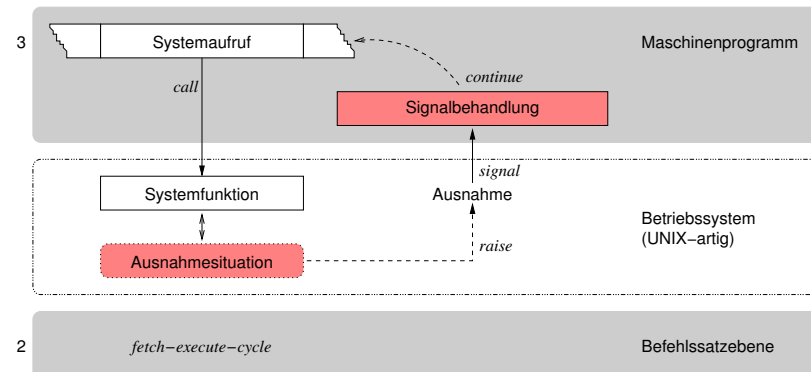
## Abweichung vom normalen Programmablauf

- **Ausnahme** (*exception*), Sonderfall, der die **Unterbrechung** oder den **Abbruch** der Ausführung des Maschinenprogramms bedeutet
  - Feststellung einer **Ausnahmesituation** beim Abruf-/Ausführungszyklus
    - ungültiger Maschinenbefehl oder Systemaufruf
    - Schutz-/Zugriffsverletzung, Seitenfehler, Unterbrechungsanforderung
  - zieht die Reaktion in Form einer **Ausnahmebehandlung** nach sich
    - realisiert durch ein spezielles Programm, einem Unterprogramm ähnlich
    - das durch Erheben (*raise*) einer Ausnahme implizit aufgerufen wird
- die Behandlung eines solchen Sonderfalls verläuft je nach Art und Schwere der Ausnahme nach verschiedenen Modellen:
  - Wiederaufnahme**
    - Ausführungsfortsetzung nach erfolgter Behandlung
    - ↔ Seitenfehler, Unterbrechungsanforderung
  - Termination**
    - Ausführungsabbruch, schwerwiegender Fehler
    - ↔ ungültiger Befehl, Schutz-/Zugriffsverletzung
- manche Programmiersprachen (z.B. Java, C++) bieten Konstrukte zum Umgang mit solchen Ausnahmen





- im Abruf- und Ausführungszyklus interpretiert die CPU den nächsten Maschinenbefehl, führt so das Programm weiter aus
  - ein solcher Befehl hat einen Operations- und ggf. Operandenteil
- bei der Interpretation dieses Befehls tritt eine Ausnahmesituation auf, die CPU erhebt (*raise*) eine Ausnahme
  - die Operation wird abgefangen (*trap*) bzw. unterbrochen (*interrupt*)
- die Ausnahmebehandlung erfolgt durch das Betriebssystem, das dazu durch die CPU aktiviert wird
  - ggf. wird die CPU instruiert, die Operation wieder aufzunehmen



- bei Ausführung der Systemfunktion tritt eine Ausnahmesituation auf, das Betriebssystem erhebt (*raise*) eine Ausnahme
  - die auf ein Signal abgebildet und zur Behandlung hoch gereicht wird
- die Signalbehandlung erfolgt im Kontext des Maschinenprogramms, sie setzt am Ende die Ausführung des Maschinenprogramms fort

## Gliederung

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

## Resümee

...virtuelle Maschinen existieren vor oder zur Programmlaufzeit

- Rechensysteme zeigen eine bestimmte innere **Schichtenstruktur**
    - die **semantische Lücke** zwischen Anwendungsprogramm und Hardware
    - die Kluft zwischen gedanklich Gemeintem und sprachlich Geäußertem
  - jedes Rechensystem ist als **Mehrebenenmaschine** ausgeprägt
    - eine Hierarchie virtueller Maschinen: **Interpretation** und **Übersetzung**
    - Demarkationslinie** bzw. ein grundlegender Bruch zwischen Ebene<sub>3,4</sub>
      - Methode der Abbildung, Art der Programmierung, Natur der Sprache
    - Abbildung der Schichten und Zeitpunkte der Abbildungsvorgänge
      - Betriebssysteme entvirtualisieren zur Laufzeit
    - Kunst der kleinen Schritte: semantische Lücke schrittweise schließen
  - Ausnahmesituationen** bilden Ebenenübergänge „von unten nach“
    - im Sonderfall bei der Programmausführung kooperieren die Maschinen
    - Analogie zwischen Betriebssystem und CPU: abstrakter/realer Prozessor
- ↔ ergänzend dazu zeigt der Anhang weitere **Interpretersysteme**
- Virtualisierungssystem** realisiert als VMM (*virtual machine monitor*)

## Zusammenfassung

### Bibliographie

## Literaturverzeichnis (2)

- [5] FEY, D. :  
**Hardwarenahe Programmierung in Assembler.**  
In: LEHRSTUHL INFORMATIK 3 (Hrsg.): *Grundlagen der Rechnerarchitektur und -organisation.*  
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien), Kapitel 2
- [6] GOLDBERG, R. P.:  
**Architectural Principles for Virtual Computer Systems / Harvard University, Electronic Systems Division.**  
Cambridge, MA, USA, Febr. 1973 (ESD-TR-73-105). –  
PhD Thesis
- [7] HABERMANN, A. N. ; FLON, L. ; COOPRIDER, L. W.:  
**Modularization and Hierarchy in a Family of Operating Systems.**  
In: *Communications of the ACM* 19 (1976), Mai, Nr. 5, S. 266–272

## Literaturverzeichnis (1)

- [1] APPLE COMPUTER, INC.:  
**Rosetta.**  
In: *Universal Binary Programming Guidelines.*  
Apple Computer, Inc., Jun. 2006 (Appendix A), S. 65–74
- [2] CHAMBERLAIN, S. ; TAYLOR, I. L.:  
**Using 1d: The GNU Linker.**  
Boston, MA, USA: Free Software Foundation, Inc., 2003
- [3] CONNECTIX CORP.:  
**Connectix Virtual PC.**  
Press Release, Apr. 1997
- [4] ELSNER, D. ; FENLASON, J. :  
**Using as: The GNU Assembler.**  
Boston, MA, USA: Free Software Foundation, Inc., Jan. 1994

## Literaturverzeichnis (3)

- [8] RITCHIE, D. M.:  
**/\* You are not expected to understand this. \*/.**  
<http://cm.bell-labs.com/cm/cs/who/dmr/odd.html>, 1975
- [9] ROBIN, J. S. ; IRVINE, C. E.:  
**Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor.**  
In: *Proceedings of 9th USENIX Security Symposium (SSYM'00)*,  
USENIX Association, 2000, S. 1–16
- [10] SCHRÖDER-PREIKSCHAT, W. :  
**Virtuelle Maschinen.**  
Sept. 2013. –  
Eingeladener Vortrag, INFORMATIK 2013, Workshop „Virtualisierung: gestern, heute und morgen“, Koblenz

- [11] SMITH, J. E. ; NAIR, R. :  
**Virtual Machines: Versatile Platforms for Systems and Processes.**  
 Morgan Kaufmann Publishers Inc., 2005. –  
 656 S. –  
 ISBN 9781558609105
- [12] TANENBAUM, A. S.:  
**Multilevel Machines.**  
 In: Structured Computer Organization[13], Kapitel 7, S. 344–386
- [13] TANENBAUM, A. S.:  
**Structured Computer Organization.**  
 Prentice-Hall, Inc., 1979. –  
 443 S. –  
 ISBN 0-130-95990-1
- [14] <http://www.hyperdictionary.com/computing/semantic+gap>

## Architektonische Prinzipien virtueller Rechnersysteme

- Schichten der Ebene<sub>[2,3]</sub> werden durch reale oder virtuelle Maschinen implementiert, die normalerweise als **Interpreter** fungieren
  - real**
    - beschränkt auf Ebene<sub>2</sub>, nämlich die **physische CPU** (z.B. x86)
  - virtuell**
    - für beide jeweils durch ein spezifisches Programm in **Software**
    - im Falle von Ebene<sub>3</sub> das **Betriebssystem** (nur partiell)
    - bezüglich Ebene<sub>2</sub> ein **Virtualisierungssystem** (total/partiell)
- gelegentlich ist aber auch **Binärübersetzung** anzufinden (z.B. [1])
- dabei interpretiert das Virtualisierungssystem alle oder nur einen Teil der Befehle der Programme der virtuellen Maschine
  - total**
    - als **Emulator** der eigenen oder einer fremden realen Maschine [3]
    - „complete software interpreter machine“ (CSIM, [6, S. 21])
  - partiell**
    - als **virtual machine monitor** (VMM, [6, S. 21]), Typ I oder II
    - der nur „sensitive Befehle“ abfängt und (in SW) emuliert
- je nach VMM ist der Übereinstimmungsgrad von virtueller und realer Maschine (Wirt) möglicherweise unterschiedlich [6, S. 17]
  - bei **Selbstvirtualisierung** besteht 100% funktionale Übereinstimmung
  - im Gegensatz zur **Familienvirtualisierung**, bei der die virtuelle Maschine lediglich Mitglied der Rechnerfamilie der Wirtsmaschine ist

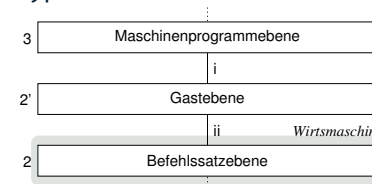
## Anhang

### Interpretersysteme

## Wächter virtueller Maschinen

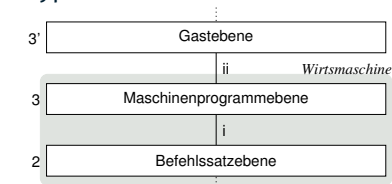
VMM bzw. Hypervisor

### Typ I VMM



- läuft auf einer „nackten“ Wirtsmaschine
- unter keinem Betriebssystem

### Typ II VMM



- läuft auf einer erweiterten Wirtsmaschine
- unter dem Wirtsbetriebssystem

- beiden gemeinsames Operationsprinzip ist die **Teilinterpretation**:
  - i durch das Betriebssystem (Typ I) bzw. Wirtsbetriebssystem (Typ II)
  - ii durch den VMM
- Gegenstand der Teilinterpretation sind **sensitive Befehle**
  - jeder Befehl, dessen direkte Ausführung durch die VM nicht tolerierbar ist
    - privilegierte Befehle ausgeführt im unprivilegierten Modus  $\rightsquigarrow$  *Trap*
    - aber leider auch unprivilegierte Befehle mit kritischen Seiteneffekten

- typische Anforderungen an die Befehlssatzebene [6, S.47–53]:
  1. annähernd äquivalente Ausführung der meisten unprivilegierten Befehle im System- und Anwendungsmodus des Rechnersystems
  2. Schutz von Programmen, die im Systemmodus ausgeführt werden
  3. Abfangvorrichtung („Falle“, *trap*) für **sensitive Befehle**:
    - a Änderung/Abfrage des Systemzustands (z.B. Arbeitsmodus des Rechners)
    - b Änderung/Abfrage des Zustands reservierter Register oder Speicherstellen
    - c Referenzierung des (für 2. erforderlichen) Schutzsystems
    - d Ein-/Ausgabe
- **unprivilegierte sensitive Befehle** sind kritisch, **Intel Pentium** [9]:
  - verletzt 3.b** ■ SGDT, SIDT, SLDT; [SMSW;] POPF, PUSHF
  - verletzt 3.c** ■ LAR, LSL, VERR, VERW; POP, PUSH; STR, MOVE
  - CALL, INT n, JMP, RET
- bei Vollvirtualisierung (VMware), ist **partielle Binärübersetzung** eine Lösung, oder eben **Paravirtualisierung** (VM/370, Denali, Xen)
- in beiden Fällen sind aber Softwareänderungen unvermeidbar, entweder am Maschinenprogramm oder am Betriebssystem

- **Vollvirtualisierung** (Selbstvirtualisierung) ist funktional transparent
  - bis auf Zeitmessung hat das Betriebssystem sonst keine Möglichkeit, in Erfahrung zu bringen, ob es eine virtuelle oder reale Maschine betreibt
  - vorausgesetzt der Abwesenheit (unprivilegierter) sensibler Befehle und damit der Nichterfordernis von Binärübersetzung
- ↳ Betriebssystem und VMM wissen nicht voneinander
- anders verhält es sich mit **Paravirtualisierung** → intransparent
  - Grundidee dabei ist, dass das Betriebssystem gezielt mit dem VMM in Interaktion tritt und bewusst auf Transparenz verzichtet
  - Hintergrund ist die **Deduplikation** von Funktionen aber auch Daten, die sowohl im Betriebssystem als auch im VMM vorhanden sein müssen
    - Betriebsmittelverwaltung, Gerätetreiber, Prozessorsteuerung, ...
  - weiterer Aspekt ist die damit einhergehende Reduktion von Gemeinkosten (*overhead*) durch Wegfall der Teilinterpretation des Betriebssystems
  - in dem Zusammenhang werden im Betriebssystem ursprünglich enthaltene sensitive Befehle als Elementaroperationen des VMM repräsentiert
- ↳ Betriebssystem und VMM gehen eine Art **Symbiose** ein

