

Fachgebiet System- und Rechnerarchitektur

Lehre und Forschung in der systemnahen Informatik

Prof. Dr.-Ing. habil. Daniel Lohmann

Prof. Dr.-Ing. habil. Jürgen Brehm

05.12.2017



Prof. Dr.-Ing. habil. Daniel Lohmann
lohmann@sra.uni-hannover.de

Betriebssysteme



Prof. Dr.-Ing. habil. Jürgen Brehm
brehm@sra.uni-hannover.de

Rechnerarchitektur



Prof. Dr.-Ing. Christian Müller-Schloer
cms@sra.uni-hannover.de

Organic Computing





[Problem]

Hochsprache



semantische Lücke



CPU-Steuerinformationen

[Ausführung]



- Informatisches Urproblem:
Schließen der Semantischen Lücke
- Komplexitätsreduktion durch Hierarchie
 - *Divide et impera* – Teile und herrsche
 - Hierarchisch angeordnete **virtuelle Maschinen**
 - Problem schrittweise in einfachere Darstellungen bis auf die reale Maschine herunterbrechen.
- Top-Down vs. Bottom-Up
 - Top-Down: Vom Problem zur Lösung
 - Bottom-Up: Erschaffen von Lösungsräumen

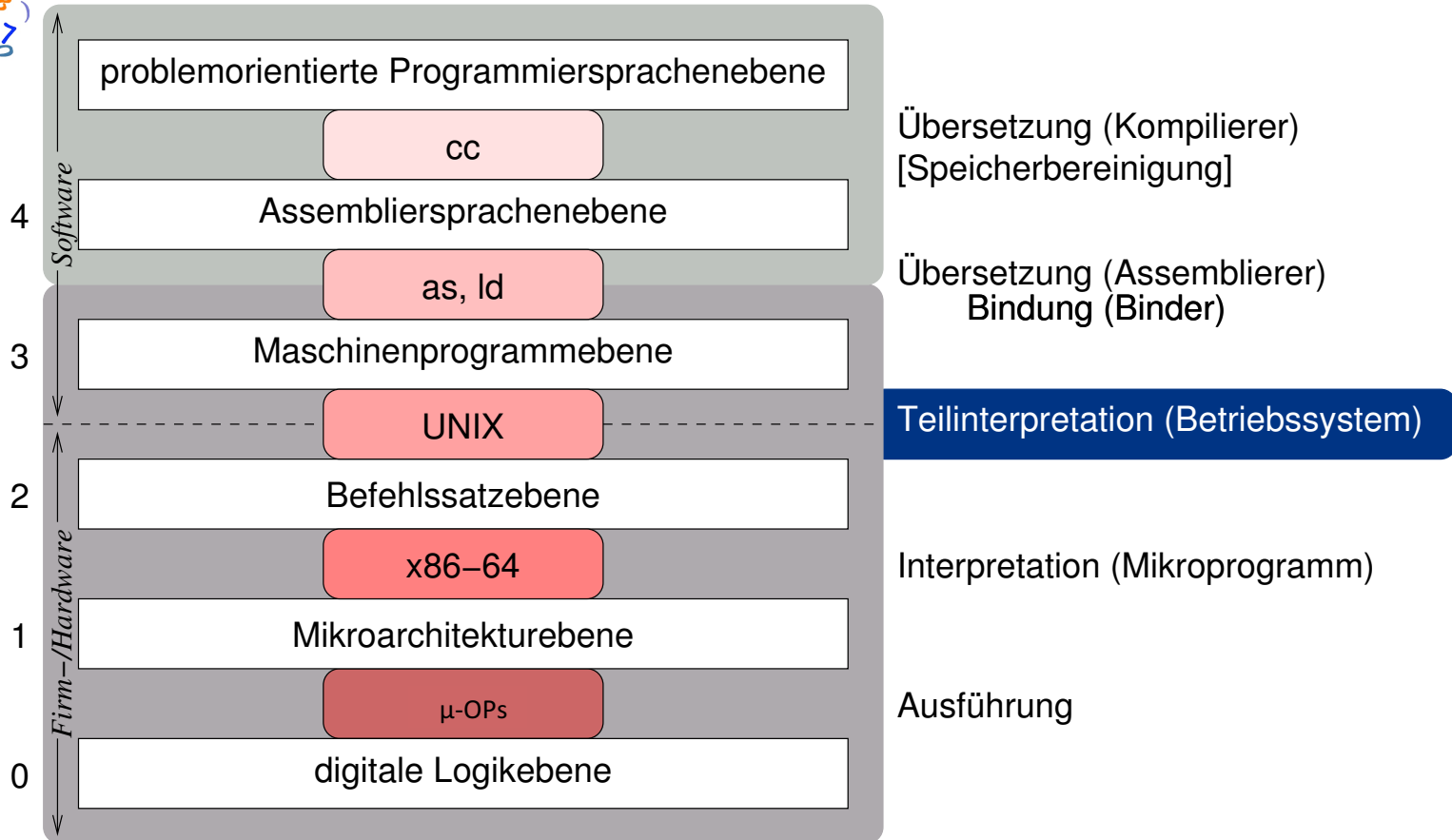
→ Systeminfrastruktur



Ebene		
n	virtuelle Maschine M_n mit Maschinsprache 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 Maschinsprache 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 Maschinsprache 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 Maschinsprache S_0	Programme in S_0 werden direkt von der Hardware ausgeführt



Angelehnt an: Tanenbaum, „Structured Computer Organization“



Angelehnt an: Tanenbaum, „Structured Computer Organization“

„Muss ich das wirklich im Detail wissen?“





Mathematik

$$\mathbf{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \quad \text{sum}(\mathbf{A}) = \sum_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} a_{ij}$$

Informatik (Hochsprache)

```
int sum = 0;
for (int j = 0; j < N; ++j) {           // Erst Spalten, dann Zeilen
    for (int i = 0; i < M; ++i) {
        sum += matrix[i][j];
    }
}
```

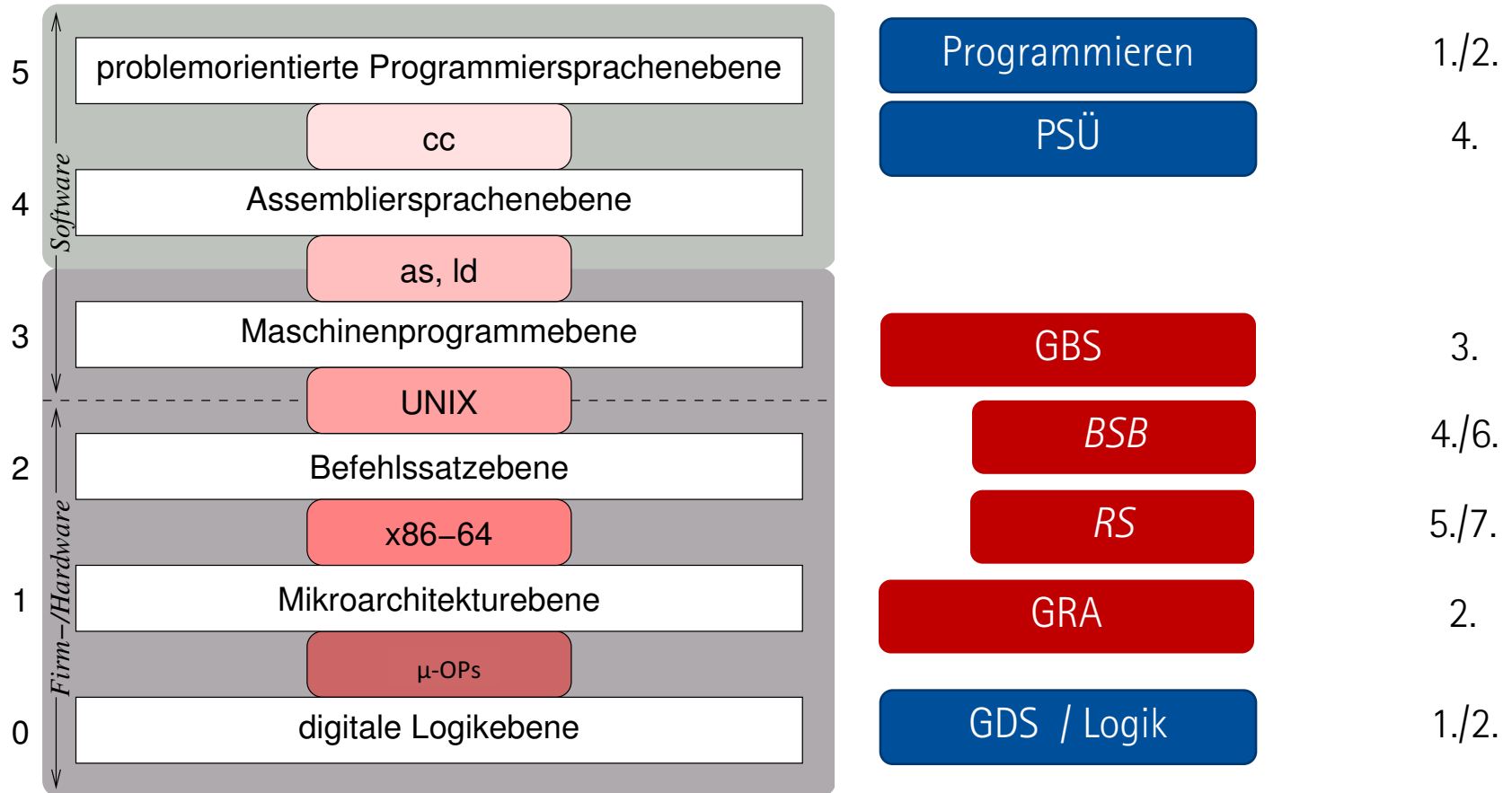
Welche Lösung ist besser?

Gibt es überhaupt einen Unterschied?

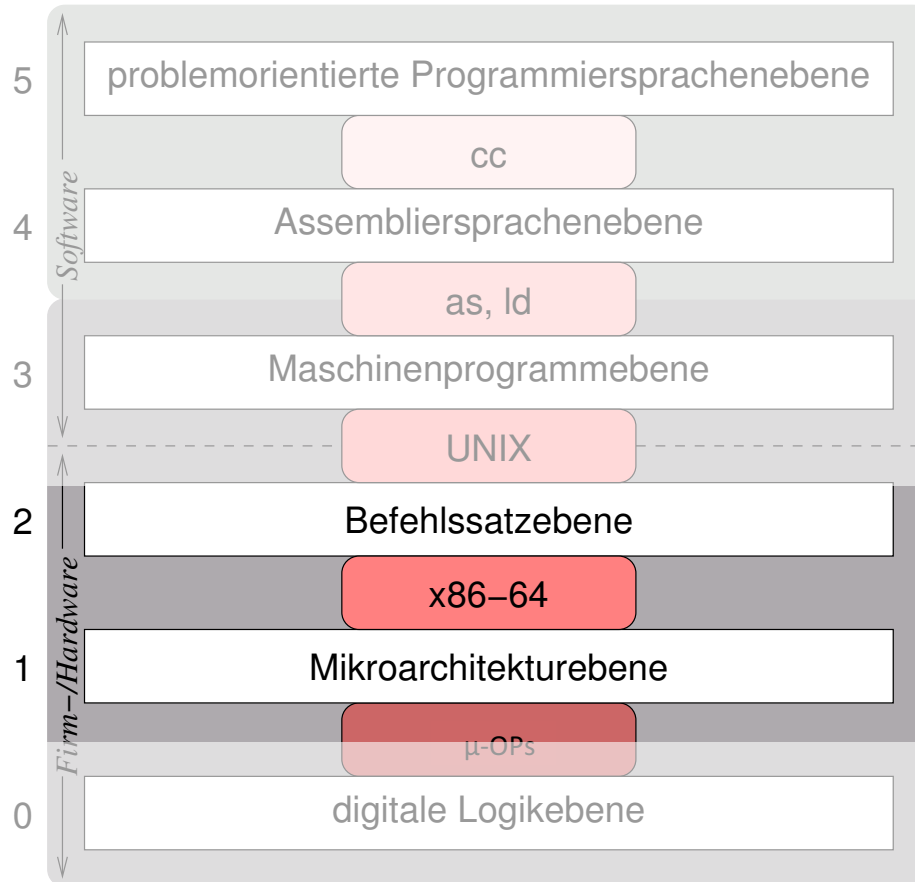
Informatik (Hochsprache, alternative Implementierung)

```
int sum = 0;
for (int i = 0; i < M; ++i) {           // Erst Zeilen, dann Spalten
    for (int j = 0; j < N; ++j) {
        sum += matrix[i][j];
    }
}
```

- **Funktional** sind beide Lösungen identisch
 - Wegen der Kommutativität der Addition sind beide Verfahren korrekt
- **Nichtfunktional** zeigen sich erhebliche Unterschiede
 - Spaltenweise Bearbeitung bis zu **Faktor 10** langsamer
- Die Ursache findet sich in den tieferen Schichten!
 - Wie organisiert der Compiler das Feld im Speicher? Wie den Zugriff?
 - Wie stellt das Betriebssystem Speicher bereit?
 - Wie läuft der Speicherzugriff auf der Hardware ab?
- Wissen, was dahinter steckt!



Bereich Rechnerarchitektur



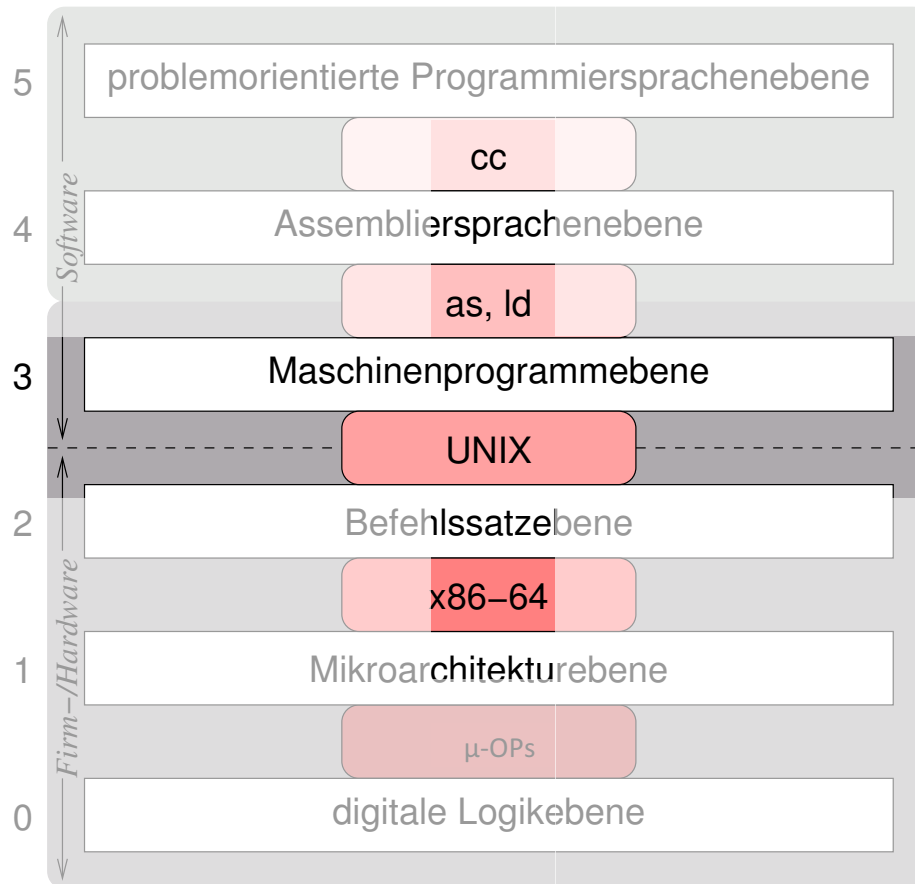
5./7.

2.

- Pflichtveranstaltung im 2. Semester, 2V+2Ü, 5 LP
- Thema: Grundlagen der Rechnerarchitektur
 - Grundsätzliches: Wie ist ein Prozessor aufgebaut und wie funktioniert er eigentlich?
 - Konkretes: Am Beispiel der minimalistischen Architektur Minimax
 - Kapitel: Historie, Systematik, Informationstheoretische Grundbegriffe, Codierung, Automaten, HW/SW-Interface, Maschinensprache, von-Neumann-Rechner, Speichersystem, Ausführungseinheit, Steuereinheit, Pipeline (Grundlagen), Performance, Ein- und Ausgabe
- Vorlesung: Verstehen der Konzepte und Prinzipien
- Übung: Vertiefen der Vorlesungsthemen anhand von Übungsaufgaben mit Bezug zur Vorlesung

- Wahlpflichtveranstaltung im 5. oder 7. Semester, 2V+2Ü, 5 LP
- Thema: Rechnerarchitektur und Multiprozessoren
(Advanced Computer Architecture)
 - Grundsätzliches: Wie ist ein moderner Mikroprozessor aufgebaut und welche Möglichkeiten der Leistungssteigerung gibt es?
 - Konkretes: Fallstudien
 - Kapitel: Ziele der Rechnerarchitektur, Performance und Kosten, Befehlssatzdesign, ALU-Entwurf, Datenpfad, Cache, Superskalarität, parallele Rechnerarchitekturen, Multicore-Architekturen, Hyperthreading, Synchronisation
- Vorlesung: Verstehen der Konzepte und Prinzipien
- Übung: Vertiefen der Vorlesungsthemen anhand von Übungsaufgaben mit Bezug zur Vorlesung

Bereich Betriebssysteme



BS-Konzepte
verstehen und
verwenden

GBS

3.

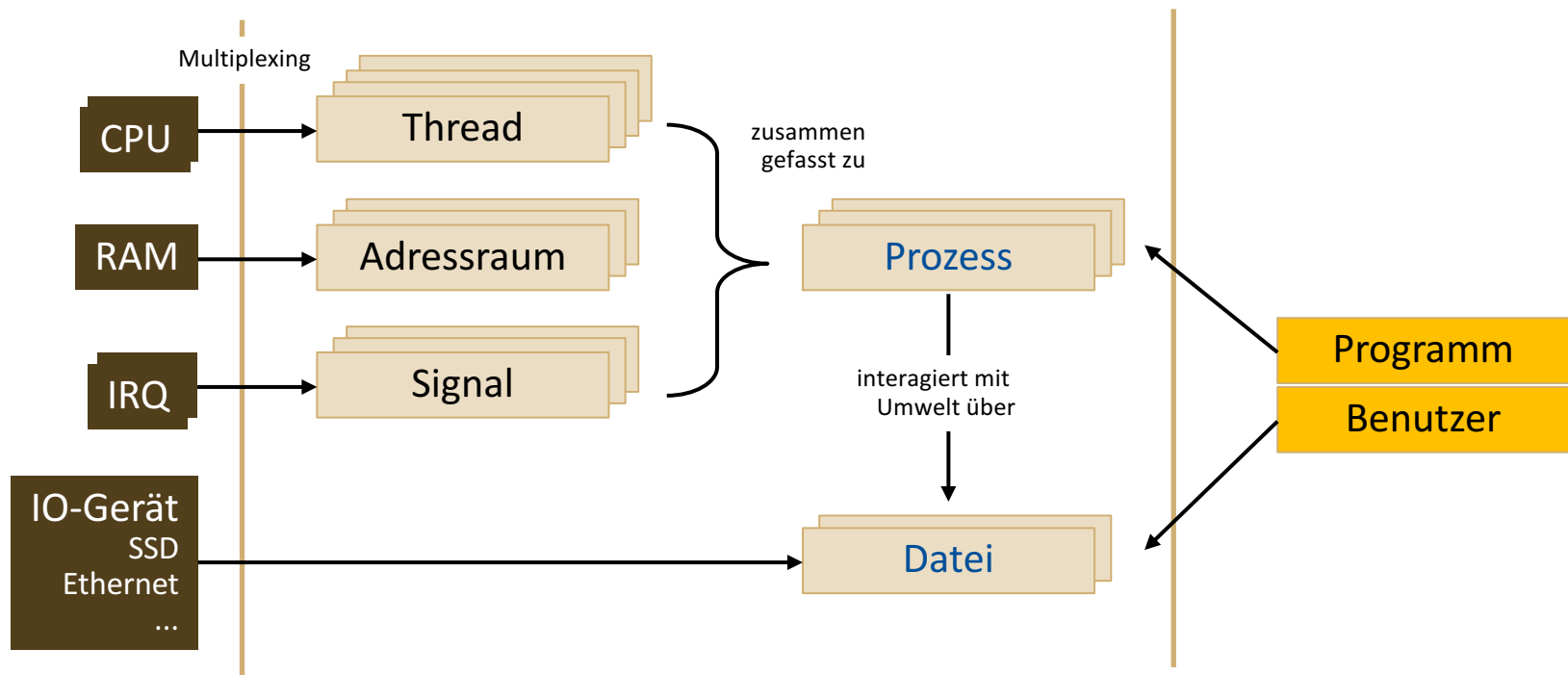
BSB

4./6.

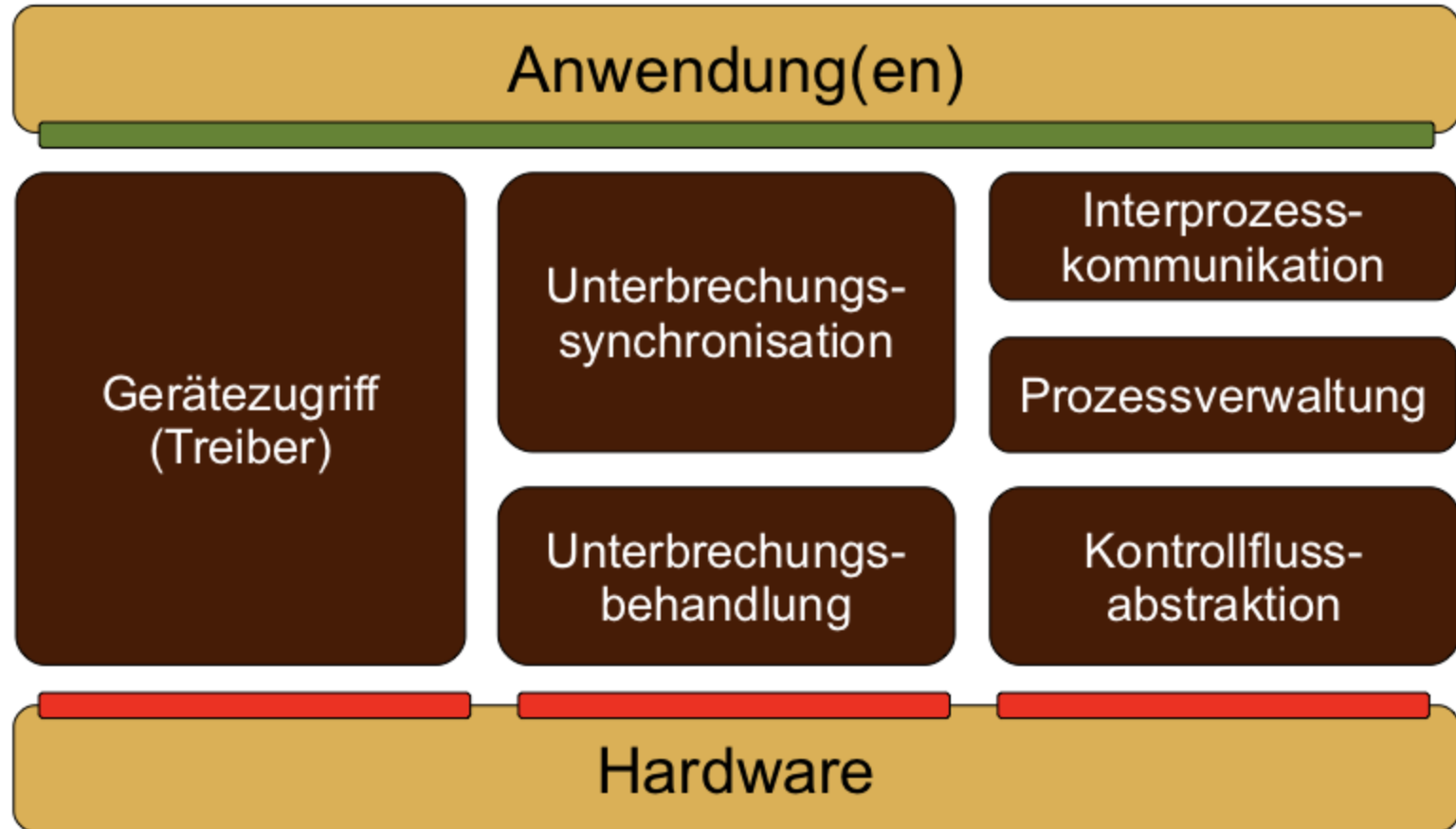
BS-Konzepte
entwerfen und
implementieren

- Pflichtveranstaltung im 3. Semester, 2V+2Ü, 5 LP
 - Neue Veranstaltung: Wird erstmalig gelesen im WS 2018
- Thema: Grundlagen der Betriebssysteme :-)
 - Grundsätzliches: Was ist eigentlich ein Betriebssystem?
 - Konkretes: Am Beispiel UNIX (POSIX) und Linux
 - Prozesse und Threads, Speicher und Adressräume, Dateien und IO, Zugriffsschutz und Sicherheit, Parallelität und Synchronisierung, ...
- Vorlesung: Verstehen der Konzepte und Prinzipien
- Übung: Systemnahe Softwareentwicklung an der BS-Schnittstelle (Programmierübungen in C unter Linux)

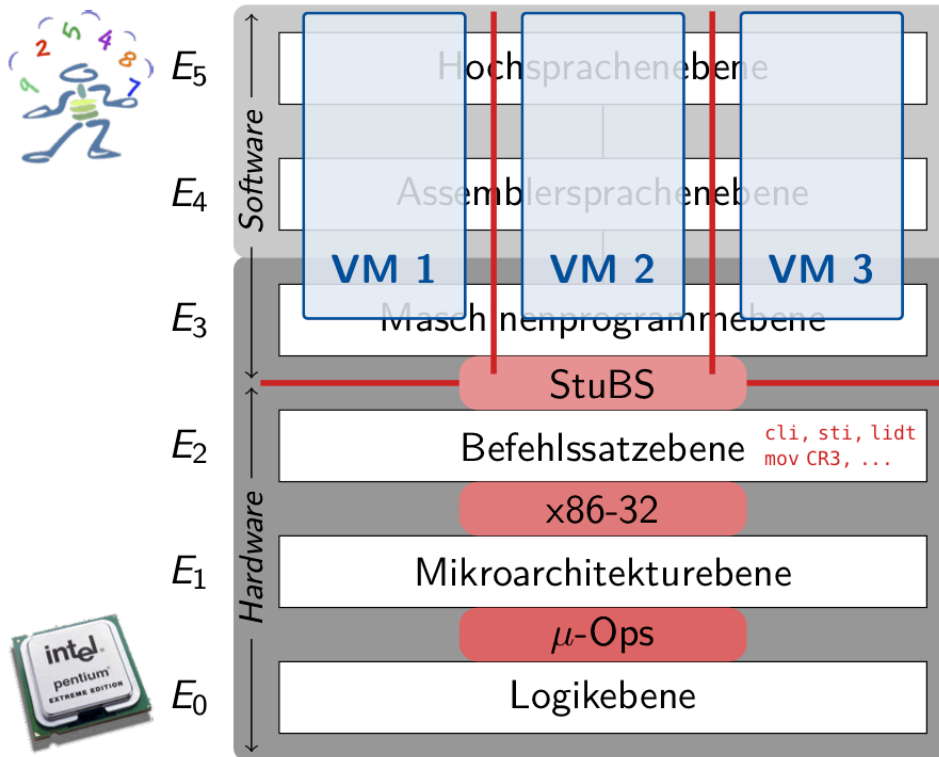
- Virtuelle Hardwareressourcen werden repräsentiert durch die grundlegenden Konzepte (Abstraktionen) eines BS:
 - Prozess (Adressraum + Thread(s) + Signal(e)) → virtueller „Computer“
 - Datei → virtuelles „Gerät“ (ggfs. persistent)



- WP im „Fachmodul Betriebssysteme“ im 4./6. Semester, 2V+2Ü, 5 LP
Alternative Variante (Master): BSB für Mehrkernsysteme, 2V+4Ü, 8LP
- Thema: Grundlagen des Betriebssystem**baus** für Mehrkernsysteme
 - **Vertiefen** des Wissens über die interne Funktionsweise von Betriebssystemen
 - Ausgangspunkt: Grundlagen der Betriebssysteme
 - Schwerpunkt: Nebenläufigkeit und Synchronisation
 - **Entwickeln** eines Betriebssystems „*from scratch*“
 - OOSTuBS / MPStuBS Lehrbetriebssysteme in Kleingruppen entwickeln
 - **Praktische** Erfahrungen im BS-Bau und hardwarenaher Softwareentwicklung machen
- Vorlesung: Verstehen der Konzepte und Prinzipien, inklusiver ihrer Implementierungsaspekte auf realer Hardware
- Übung: Entwurf und Implementierung des eigenen Betriebssystems



- Labor im Master, 4L, 6 LP
- Thema: Schutz und Isolation in Betriebssystemen



Horizontale Isolation

(zeitlich/räumlich)

unabhängiger virtueller
Maschinen (Prozesse) durch
IRQs, MPU/MMU (auf E₂)

Teilinterpretation (Betriebssystem)

Vertikale Isolation

(Benutzer-/Systemmodus)

durch Abschirmung der
E₂-Instruktionen für die
horizontale Isolation

- AKSI – Ausgewählte Kapitel der Systemnahen Informatik
 - Seminar im Master (WS), wechselnde Themen
 - Aktuell: Die Architektur des Linux-Kerns

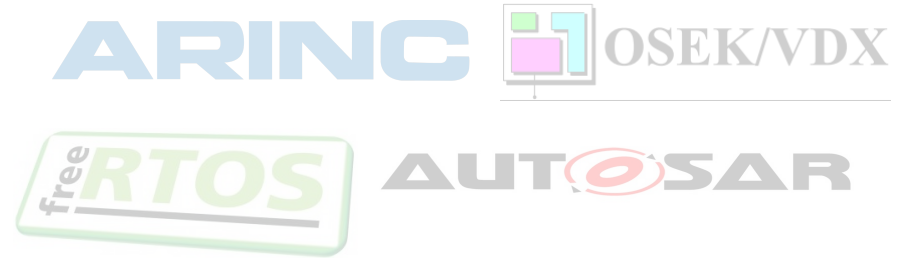
- PSRA – Projekt System- und Rechnerarchitekturen
 - Projekt im Bachelor (SS), wechselnde Themen
 - Geplant: Systemnahe Softwareentwicklung im OpenSource-Umfeld





Vielzweckbetriebssysteme
(general-purpose operating systems)

2%

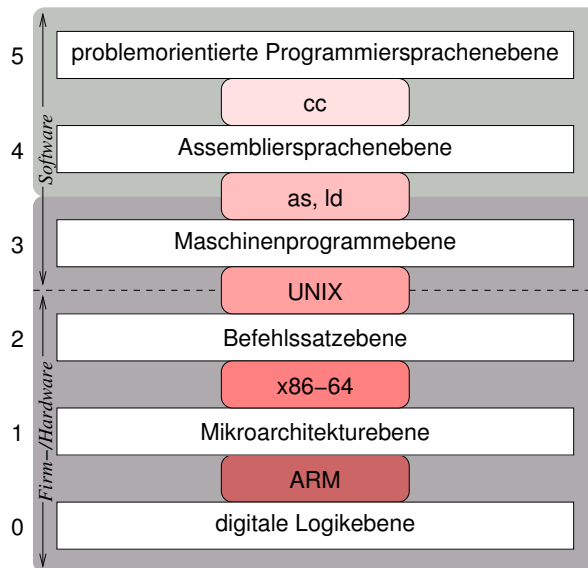


98%

Spezialzweckbetriebssysteme
(special-purpose operating systems)



Die Funktionsweise von Hardware und Betriebssystemen zu verstehen ist unabdingbar, um Phänomene eines Rechensystems begreifen, verstehen und letztlich beherrschen zu können!



Wissen, was dahinter steckt!