

Betriebssysteme (BS)

VL 2.1 – Betriebssystementwicklung – Übersetzung

Volkmar Sieh / Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

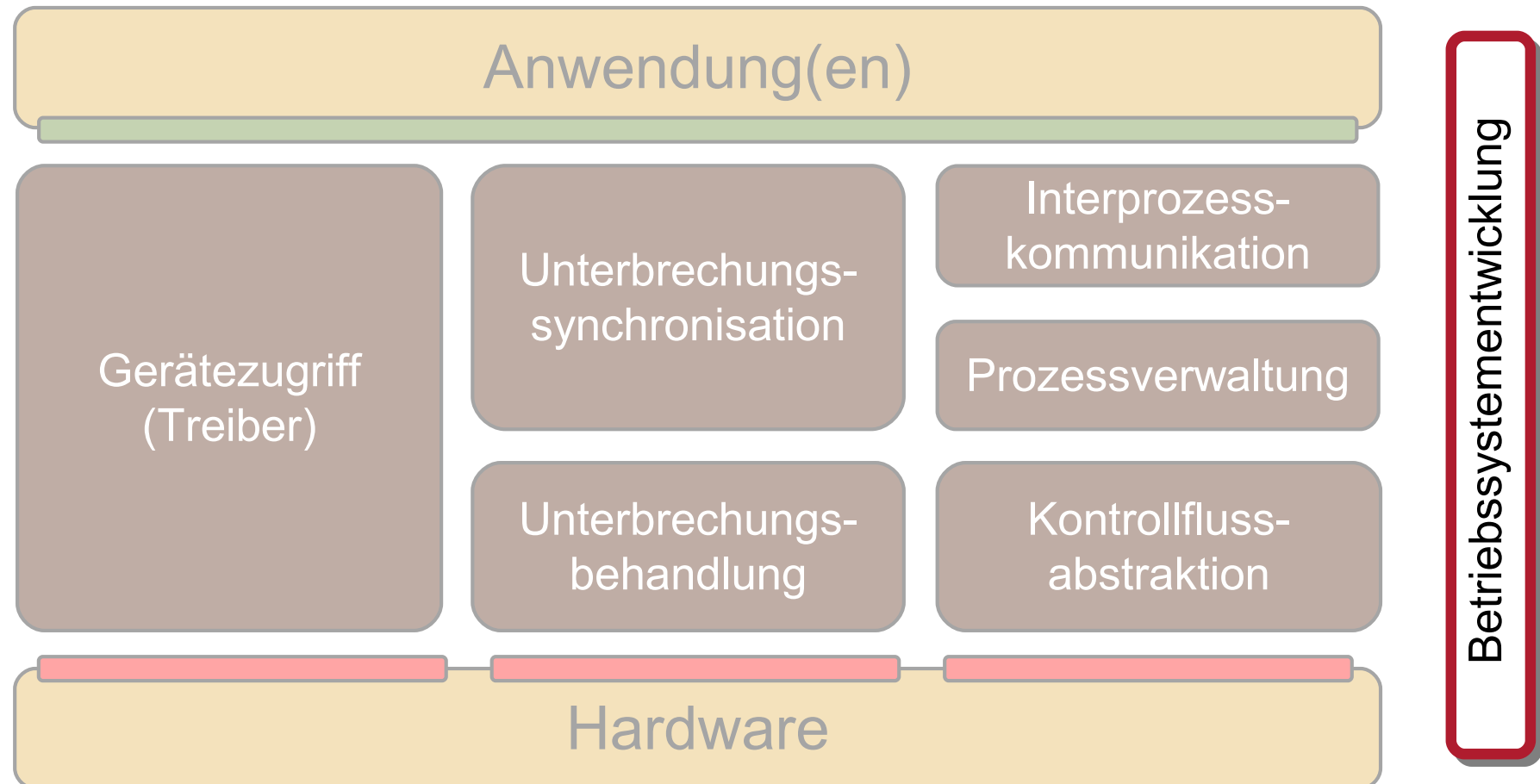
Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 20 – 2. November 2020



https://www4.cs.fau.de/Lehre/WS20/V_BS

Überblick: Einordnung dieser VL



Agenda

Einordnung
Übersetzen und Linken
Booten
Debugging
Zusammenfassung



Agenda

Einordnung

Übersetzen und Linken

Booten

Debugging

Zusammenfassung



BS-Entwicklung (oft ein harter Kampf)

- Erste Schritte
wie bringt man sein System auf die Zielhardware?
 - Übersetzung
 - Bootvorgang
- Testen und *Debugging*
was tun, wenn das System nicht reagiert?
 - „printf“ *debugging*
 - Emulatoren
 - *Debugger*
 - *Remote-Debugger*
 - Hardwareunterstützung



Agenda

Einordnung
Übersetzen und Linken
Booten
Debugging
Zusammenfassung



Übersetzung – *Hello, World?*

```
#include <iostream>

int main () {
    std::cout << "Hello, World" << std::endl;
}
```

```
> g++ -o hello hello.cc
```

- Annahme:
 - das Entwicklungssystem läuft unter Linux/x86
 - das Zielsystem ist ebenfalls ein PC
- Läuft dieses Programm auch auf der „nackten“ Hardware?
- Kann man Betriebssysteme überhaupt in einer Hochsprache entwickeln?



Übersetzung – Probleme u. Lösungen

- kein dynamischer Binder vorhanden
 - alle nötigen **Bibliotheken statisch einbinden**.
- libstdc++ und libc benutzen Linux Systemaufrufe (insbesondere write)
 - die normalen C/C++ **Laufzeitbibliotheken können nicht benutzt werden**. Andere haben wir (meistens) nicht.
- generierte Adressen beziehen sich auf virtuellen Speicher! ("nm hello | grep main" liefert "0804846c T main")
 - die Standardeinstellungen des Binders können nicht benutzt werden. **Man benötigt eine eigene Binderkonfiguration**.
- der Hochsprachencode stellt Anforderungen (Registerbelegung, Adressabbildung, Laufzeitumgebung, Stapel, ...)
 - ein eigener **Startup-Code** (in Assembler erstellt) muss die Ausführung des Hochsprachencodes vorbereiten

