Nachtrag: Anmeldung zur Übung

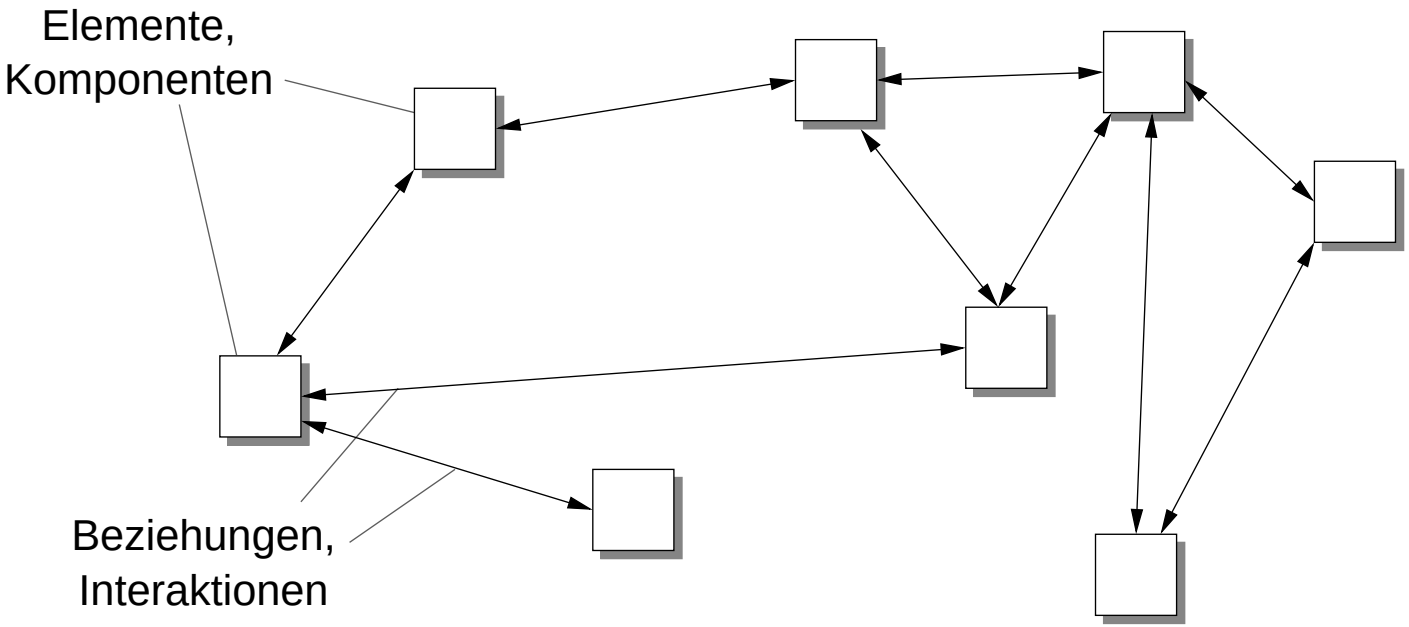
- Anmeldung zu einem *konkreten* Übungstermin ist nicht notwendig.
- **Aber:** Anmeldung im QISPOS meist notwendig für Klausur **und Übung**, damit erfolgreiches Bestehen der Übung angerechnet werden kann.
- Im Zweifelsfall gibt die Prüfungsordnung oder das Prüfungsamt Auskunft welche Anmeldung notwendig ist.

Kapitel 2
BS Architektur: Prozesse und Kern

BS: Grobstruktur

Ein System (und auch ein Betriebssystem) besteht i.d.R. aus:

- Elementen
- Beziehungen zwischen den Elementen



Zwischen Elementen existieren Interaktionen unterschiedlicher Art:
Datenfluss, Auftragsfluss, Synchronisation, Aufruf, Kommunikation, ...

Betriebssystem als Ressourcenmanager

- Das Betriebssystem steuert/verwaltet die Computer-Ressourcen, die auch Betriebsmittel genannt werden
- Dieser Steuerungsmechanismus ist jedoch von gesteuerten Objekten selbst nicht ganz getrennt:
  - BS funktioniert wie normale Software: es ist ein Programm, das vom Prozessor (CPU) des Computers ausgeführt wird
  - BS ist ein besonderes Programm: es lenkt den Prozessor bei der Verwendung anderer Systemressourcen
  - BS gibt oft die Kontrolle ab, und ist dann auf den Prozessor angewiesen, die Kontrolle zurück zu erlangen
- Der Prozessor ist selbst ein Betriebsmittel, seine Zeit wird vom BS zwischen verschiedenen Aufgaben/Programmen eingeteilt
- Die Zusammenarbeit unterstrichener Teile:

  Prozessor – Betriebsmittel – BS – Programme

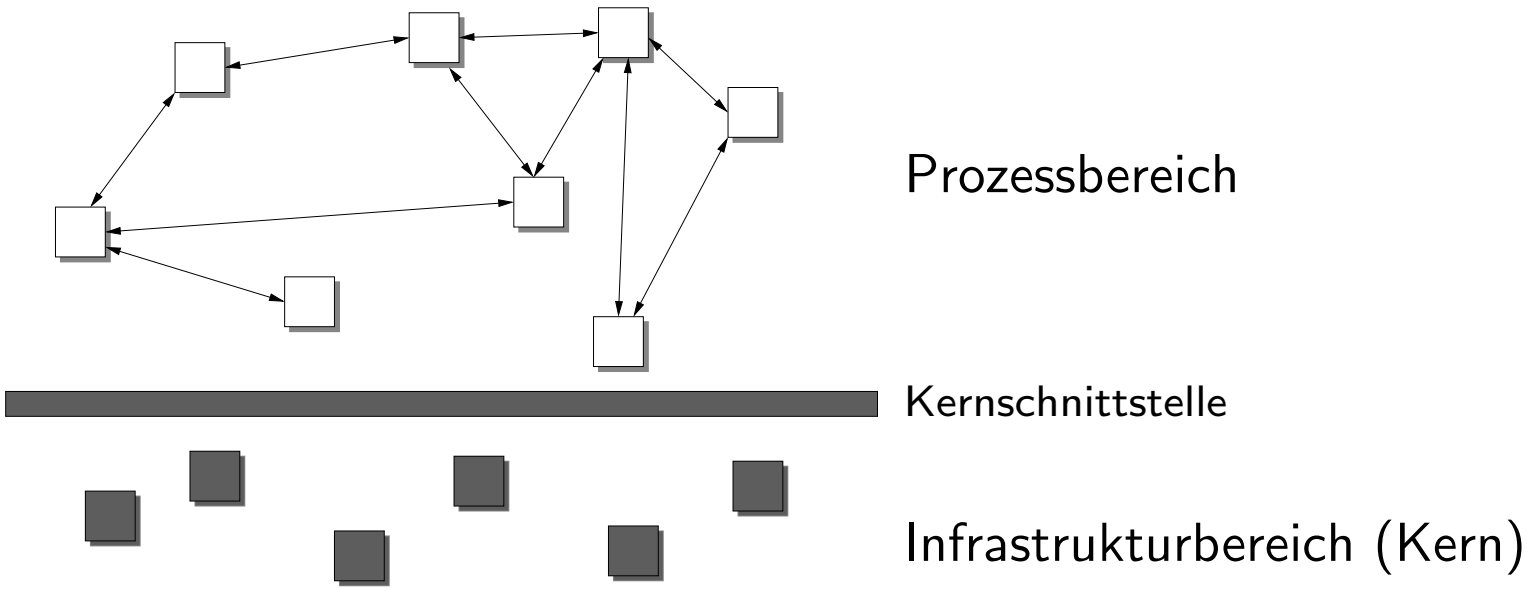
  wird mithilfe des *Prozess*-Begriffs behandelt

Zweiteilung des BS: Prozesse und Kern

- Intuitiv: **Prozess** ist ein Programm in Ausführung
- In einem BS als System werden die Elemente von Prozessen gebildet, d. h. ein Betriebssystem ist eine *Menge interagierender Prozesse*
- Da Prozesse nicht in der Hardware (ursprünglich) vorgesehen sind, muss es etwas geben, das Prozesse und ihre Interaktion ermöglicht und unterstützt.
- Dieser Bereich heißt **Kern** (*kernel*) des Betriebssystems.
  Er stellt die grundlegende Infrastruktur für Prozesse bereit.

Zweiteilung des BS: Prozesse und Kern (Forts.)

- In einer ersten groben Gliederung eines Betriebssystems unterscheiden wir daher zwei Bereiche:
  - **Prozessbereich**, in dem die eigentlichen Funktionen von BS erbracht werden
  - **Kern(bereich)**, der für Prozesse die erforderliche Infrastruktur zur Verfügung stellt



Die Einordnung des Betriebssystems

- Die unterste Ebene der Hierarchie ist die Hardware: Chips, Platinen, Platten, Tastatur, Monitor, etc.
- Über der Hardware liegt die Software
  - **Prozessbereich**, in dem die eigentlichen Funktionen von BS erbracht werden
  - **Kern(bereich)**, der für Prozesse die erforderliche Infrastruktur zur Verfügung stellt

Einordnung des Betriebssystems

(BS-)Software:

- **Prozessbereich**, in dem die eigentlichen Funktionen von BS erbracht werden
- **Kern(bereich)**, der für Prozesse die erforderliche Infrastruktur zur Verfügung stellt

liegt über der Hardware

Hardware: Chips, Platinen, Platten, Tastatur, Monitor, etc.

Unterste Ebene

Benutzer- vs. Systemmodus

- Die meisten modernen CPUs haben mind. zwei Arbeits-Modi:
  - **Benutzermodus**: für „normale“ Programme/Anwendungen
  - „**Priviligierter**“ Modus (auch System-, Steuer-, Kernel-Modus)
- Bestimmte Befehle werden nur im priviligierten Modus ausgeführt:
  - Lesen/Schreiben bestimmter Register
  - primitive E/A-Befehle
  - Speicherverwaltung
- Der Grund für zwei Modi: BS und BS-interne Daten müssen vor Benutzer-Eingriff geschützt werden – das darf nur der Kern!
- Zwei Fragen:
  - **Wie weiss der Prozessor in welchem Modus er gerade arbeitet?** Durch ein Bit im Programmstatuswort-Register (PSW), s. später
  - **Wie wird der Modus geändert?** Das Bit wird als Reaktion auf bestimmte Ereignisse, z. B. einen Aufruf an einen BS-Dienst geändert: z. B. durch CHM (Change-Mode)-Befehl
- Wenn ein nicht-priviligiertes Benutzerprogramm versucht, einen CHM-Befehl auszuführen, führt das zu einem Fehler

Benutzer- vs. Systemmodus (Forts.)

Die Trennung zwischen Benutzer- und Systemmodus ist oft unscharf:

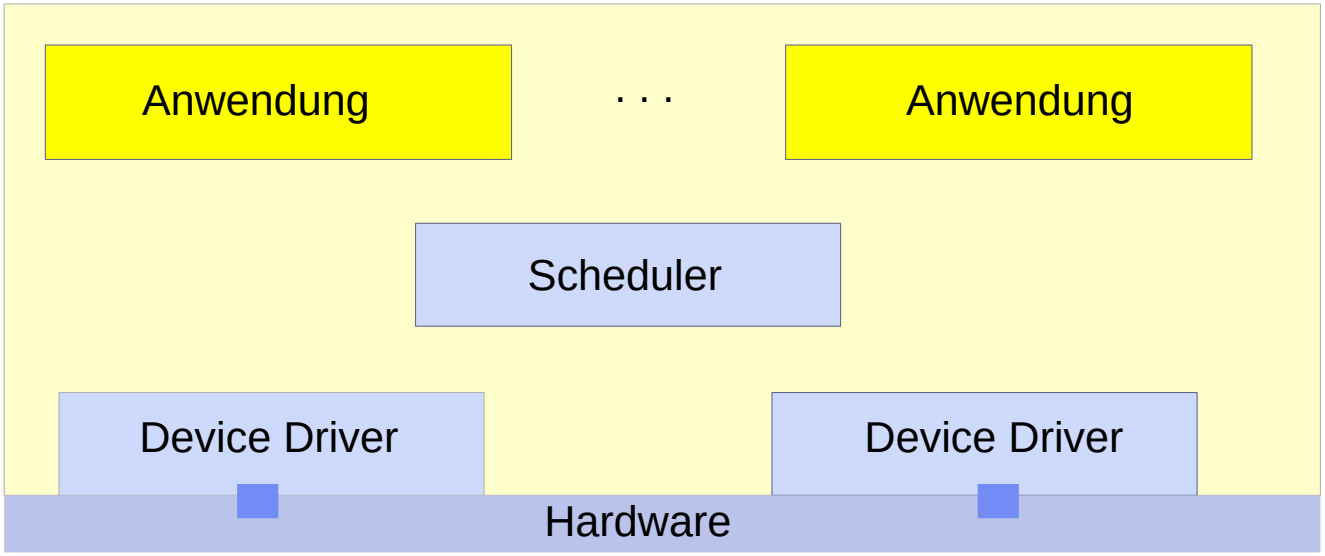
- Nicht alle BS bzw. nicht jede Hardware verwenden mehrere Modi:
  - In eingebetteten Systemen (z. B. in Waschmaschinen, Autos, . . .) wird aus Effizienzgründen darauf verzichtet
  - Interpretierte BS (z. B. auf Java basierend) verwenden andere Schutzmechanismen (Interpreter)
- Alle Benutzerprogramme laufen im Benutzermodus, aber nicht alle Teile des BS müssen im Kern laufen
  - Bsp.: Programm zum Ändern von Passwörtern ist nicht Bestandteil des BS und läuft im Benutzermodus, muss jedoch geschützt werden
  - Unterschiedliche BS führen unterschiedlich viele Teile im Benutzermodus aus
- Im Allgemeinen gilt: Alles was im Kernmodus läuft gehört zum BS (aber nicht umgekehrt).

Mikro- vs. Makrokernarchitektur

- Die Kerne moderner Betriebssysteme unterscheiden sich in ihrer Größe erheblich: von einigen MByte Hauptspeicher bis zu wenigen 100 KByte (Nanokern oder Picokern)
- Es herrscht keine Übereinstimmung darüber, was in einen Kern hinein gehört (Forschungsgegenstand)
- Prozessverwaltung und Prozesskommunikation werden i. d. R. im Kern platziert, s. Kap. 3.
- Sind nur essentielle BS-Funktionen im Kern enthalten, so spricht man von einer **Mikrokern-Architektur**
- Eine Mikrokern-Architektur unterscheidet sich von vielen gängigen BS, wie UNIX oder Windows, wo z. B. auch das Dateisystem im Kern realisiert ist (**Makrokern-Architektur**)
- Im Folgenden besprechen wir kurz einige konkrete Klassen von BS

BS-Architektur I: Monolithische Systeme

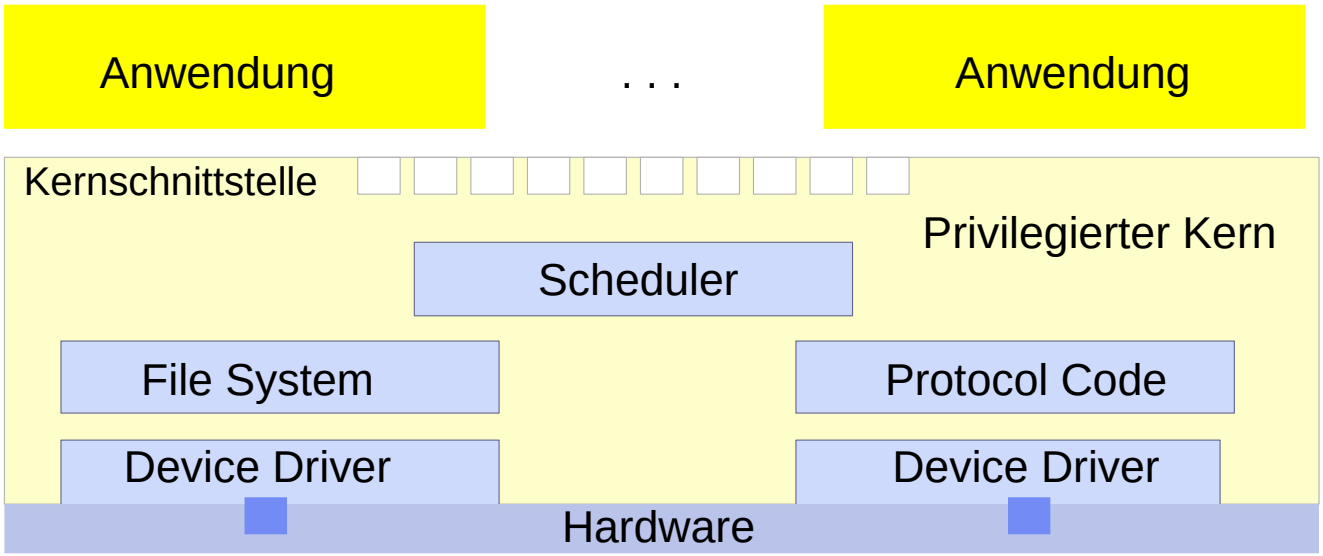
- Erste Klasse: sog. Monolithische Betriebssysteme



- Keine strenge Trennung zwischen Applikation und BS: eine Menge von Prozeduren, die sich gegenseitig aufrufen
- Geeignet für kleine, statische Betriebssysteme, da bei großen BS die Prozeduren fehleranfällig sind
- Beispiel: MS-DOS

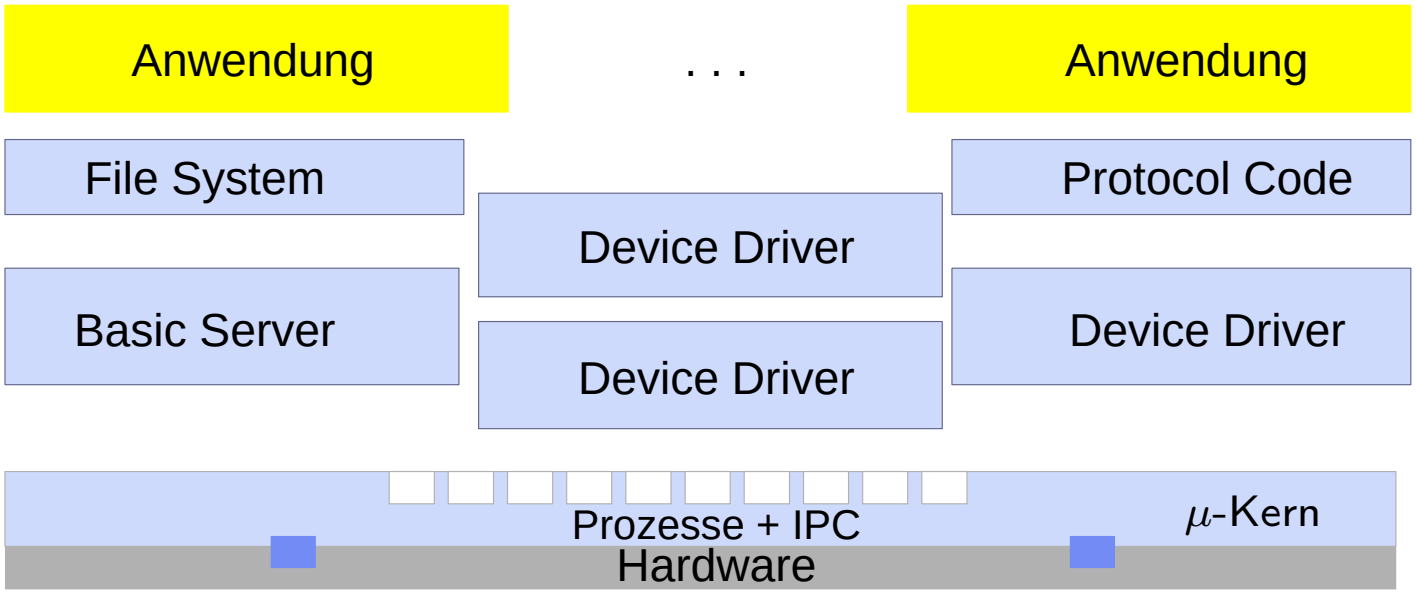
BS-Architektur II: Monolithischer BS-Kern

- Zweite Architekturklasse: Monolithischer BS-Kern (Makrokern)



- Trennung Anwendung – BS, aber keine unter Kernkomponenten (in [Tanenbaum 2003] wird als monolithisches System bezeichnet)
- Geschichtetes BS: einzelne Funktionen hierarchisch angeordnet, mit Kommunikation zwischen benachbarten Schichten
- Immer noch problematisch: Schichtänderungen haben auf benachbarte Schichten große Auswirkungen, schwer zu verfolgen

BS-Architektur III: Mikrokern-BS



- Der Kern umfasst nur Prozessmanagement, z. B. *Scheduling* und *Dispatching*, sowie die Interprozesskommunikation (IPC)
- Externe Teilsysteme sind nunmehr: Treiber, Dateisysteme, etc.

Vorzüge der Mikrokernarchitektur

- Klare Kernschnittstelle begünstigt modulare Struktur.
- Realisierung der Dienste liegt außerhalb des Kerns, dadurch:
  - Sicherheit und Stabilität: der Kern wird durch fehlerhafte Dienste nicht in Mitleidenschaft gezogen
  - Flexibilität und Erweiterbarkeit: ein Dienst kann hinzugefügt oder weggenommen werden, selbst im laufenden Betrieb
  - Portierbarkeit: BS kann dank Mikrokern schnell auf neue Plattformen portiert werden
- Der sicherheitskritische Teil des Systems (Kern) ist relativ klein und kann daher besser verifiziert oder ausgetestet werden.

Nachteil der Mikrokernarchitektur

- Ein Problem der Mikrokernarchitektur ist die i.d.R. schlechtere Performance

  Warum?

  - Zusammenspiel der Komponenten außerhalb des Kerns erfordert mehr Interprozesskommunikation und daher mehr Systemaufrufe.
  - Performance-Problematik ist Gegenstand moderner BS-Forschung

  What is the performance of MINIX 3 like?[1]

  We made measurements of the performance of MINIX 3 (user-mode drivers) versus MINIX 2 (kernel-mode drivers) and MINIX 3 is 5-10% slower. We have not compared it to other systems because there are so many other differences. The biggest difference is that MINIX 3 represents about a handful man-year of work so far and other systems represent thousands of man-years of work and our priority has been reliability, not performance.

[1] aus: https://wiki.minix3.org/doku.php?id=faq

Anwendungen, Programme, Prozesse

- Wir haben bislang die drei Begriffe recht synonym benutzt, ohne sie genauer zu erklären.
- Insbesondere haben wir ihre Existenz vorausgesetzt.
- Anwendungsprogramme (als Daten auf einem Datenträger) gibt es natürlich, sie sind aber nicht einfach so durch eine CPU ausführbar.
- Dagegen ist der Kernel (der Betriebssytemkern) bootbar:
  - Die CPU führt den Code des BS nach dem Booten aus
  - Das BS kennt den "ganzen" Rechner, den physikalischen Speicher, die abslouten Adressen etc.
- Damit ein Anwendungsprogramm ausgeführt werden kann, schafft der BS-Kern dafür eine Umgebung, die einen für die Anwendung eigenen Rechner (virtuelle CPU) simuliert: den Prozess.
  Darin wird dann das Nutzerprogramm durch die reale CPU ausgeführt.

Prozesse und Adressräume

- **Prozess:** Ein Programm in Ausführung, inklusive
  - dem aktuellem Wert des Programmzählers
  - den aktuellen Registerinhalten
  - der Belegung der Variablen
- Konzeptionell besitzt jeder Prozess eine virtuelle CPU, d.h. alle Prozesse können ständig und gleichzeitig 'laufen'
- Die reale CPU schaltet jedoch zwischen den Prozessen um!
- Jedem Prozess wird ein **Adressraum** („Speicher“) zugeordnet
- Es können (beliebig) viele *logische Adressräume* gebildet werden, ggf. gestreut auf den physikalischen Speicher abgebildet.
- Zwar benötigt jeder Prozess zu jedem Zeitpunkt einen Adressraum, es sind jedoch mehrere Relationen möglich:
  - Ein Prozess besitzt genau einen Adressraum (Unix-Prozess).
  - Mehrere “leichte” Prozesse teilen sich einen Adressraum (Threads).

Terminologie

Bezüglich der Terminologie in der Literatur ist Vorsicht geboten:

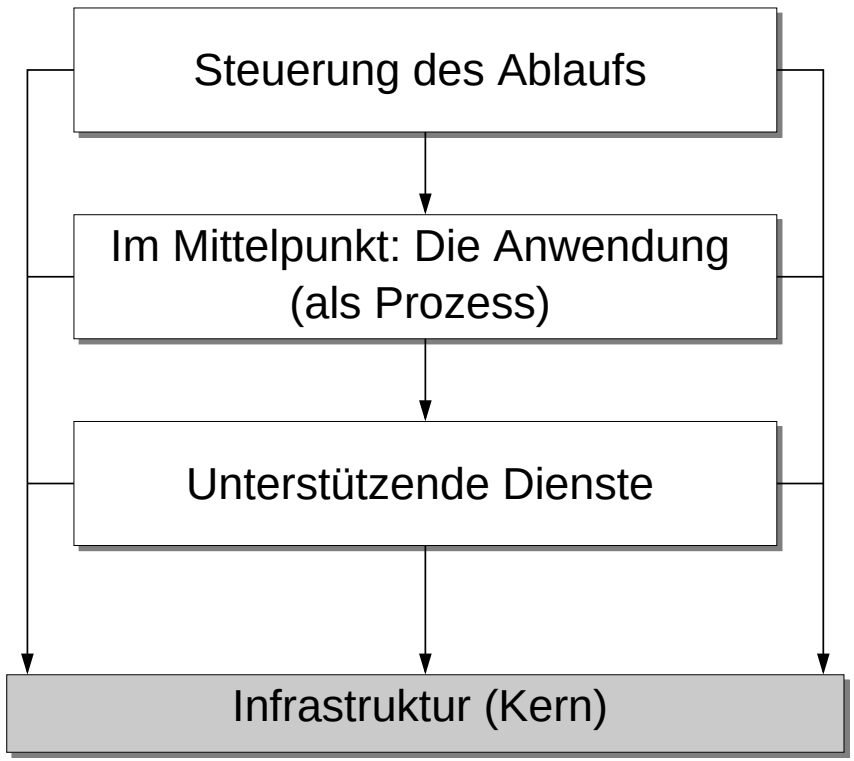
- Ein *Prozess* (*process, task*) wird oft im Sinne von Unix verstanden als ein Prozess mit einem eigenen Adressraum.
- Die meisten neueren Betriebssysteme (auch neuere UNIX-Varianten) bieten dagegen die Möglichkeit an, mehrere Prozesse in einem gemeinsamen Adressraum ablaufen zu lassen.
- Man nennt sie *leichtgewichtige Prozesse* oder *Threads*.
- In modernen Unix-Varianten (z. B. Solaris) gibt es ursprüngliche Unix-Prozesse (tasks), die aus vielen Threads bestehen können.
- Ein Unix-Prozess ist daher ein Adressraum, der mindestens einen Thread enthält (gleiche Sprechweise gilt auch für WindowsNT)
- In dieser Vorlesung verwenden wir den Begriff Prozess im Sinne von Thread, d. h. wir unterscheiden i.d.R. nicht zwischen den beiden.

2.2 Der Prozessbereich

Die erste, sehr grobe Gliederung (Folie 5) werden wir nun schrittweise auflösen, d. h. mit mehr und mehr Details beschreiben, wobei wir uns auf eine **Mikrokernarchitektur** beziehen

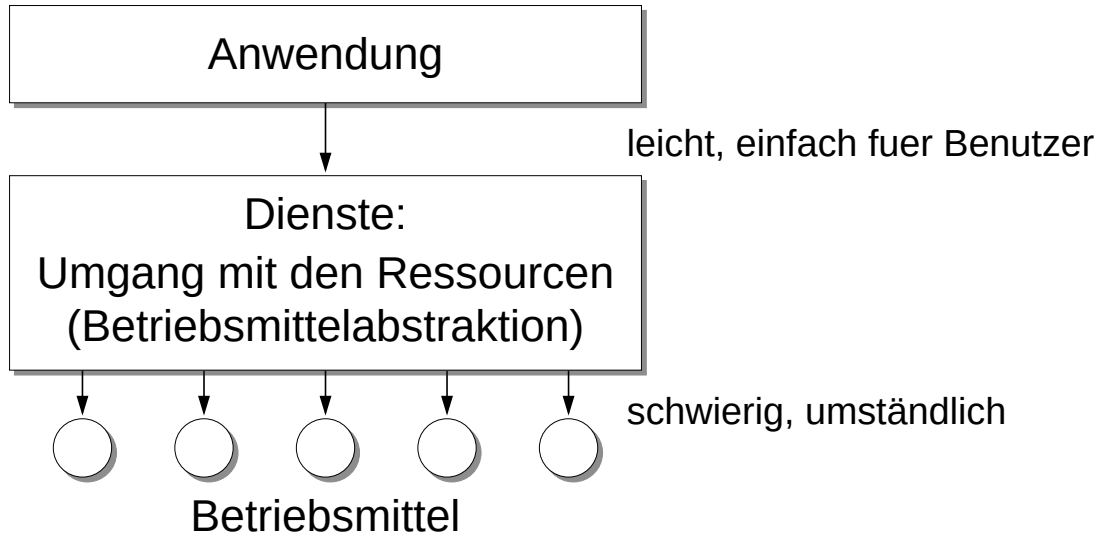
Weitere Auflösung: Prozessbereich

(Pfeile stehen für „steuert/benutzt“)



Dienste und Betriebsmittel

Weitere Auflösung: Unterstützende Dienste



Unterscheidung Betriebsmittel:

Logische BM: Aus organisatorischen Gründen „ausgedacht“, werden durch reale, physikalische Betriebsmittel realisiert
Beispiel: Datei, Fenster.

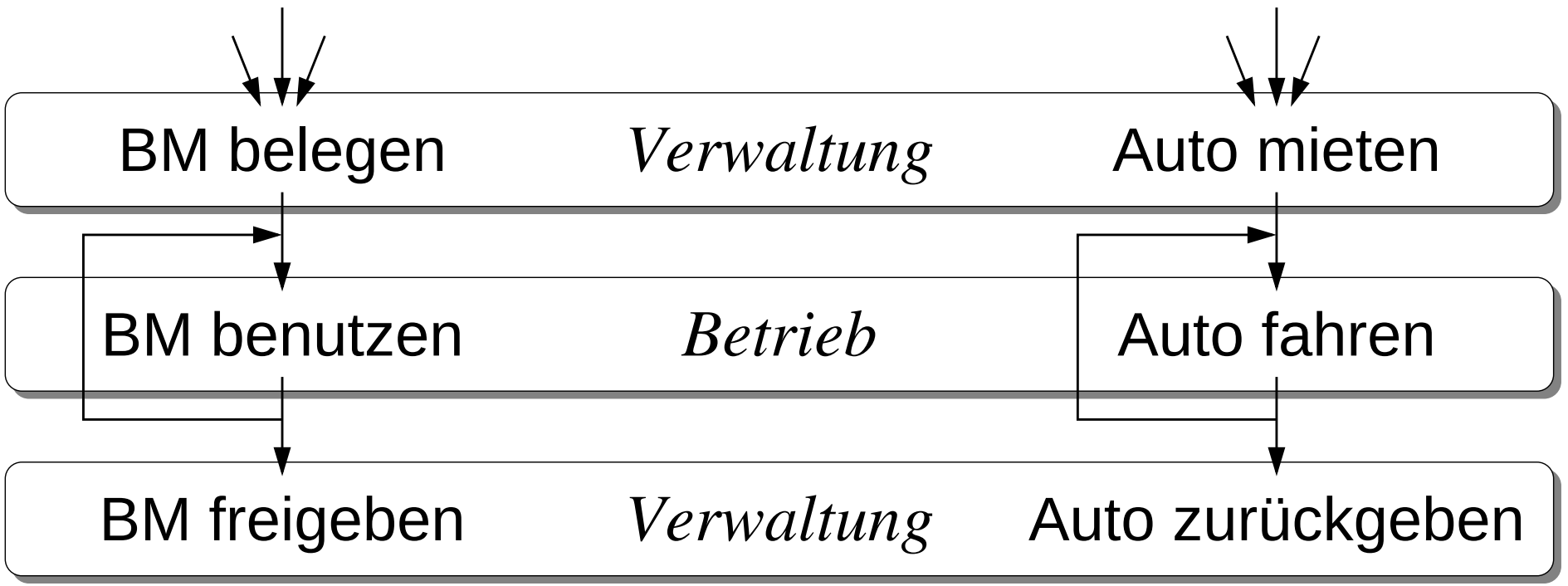
Physikalische BM: Real vorhanden, “zum Anfassen”.
Beispiel: Platte, Bildschirm

Umgang mit Betriebsmitteln: Beispiele

Der Umgang mit Betriebsmitteln hat zwei Aspekte:

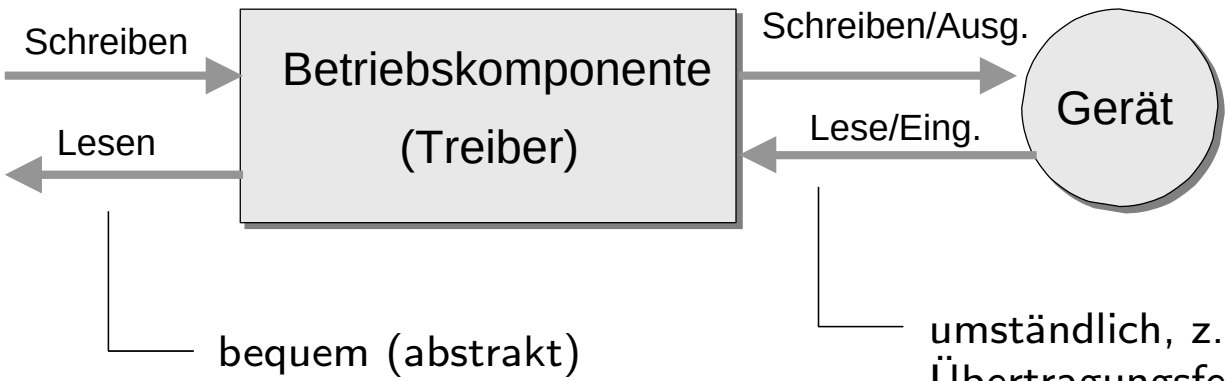
- **BM-Betrieb:** Tatsächliche Nutzung, z. B. Datentransport
- **BM-Verwaltung:** Wer darf was wann benutzen? (ggf. Wettbewerb)

Beispiel: (Analogie zur Autovermietung)

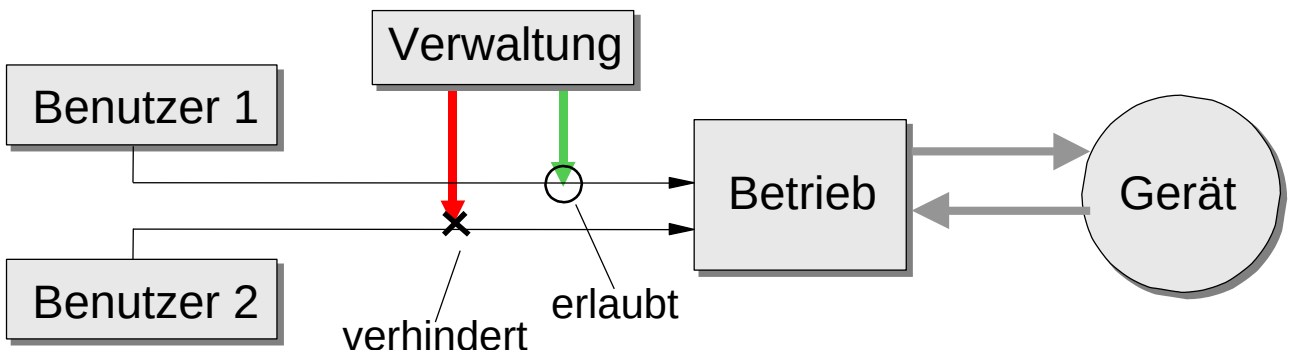


Umgang mit Betriebsmitteln

- Zum Begriff **Betrieb**

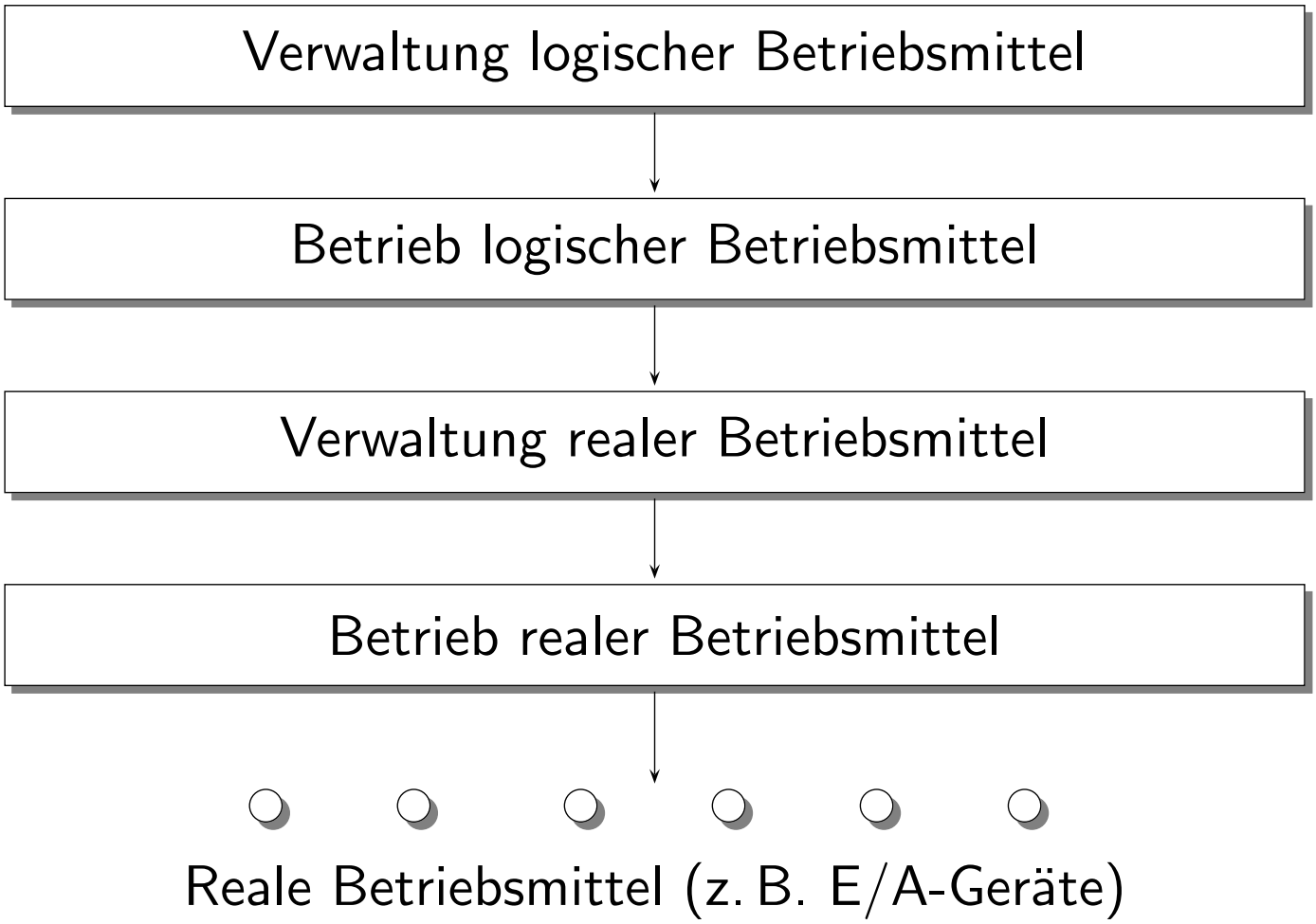


- Zum Begriff **Verwaltung**



Betriebssystem- Dienste

Gliederung der „Dienste“-Schicht:

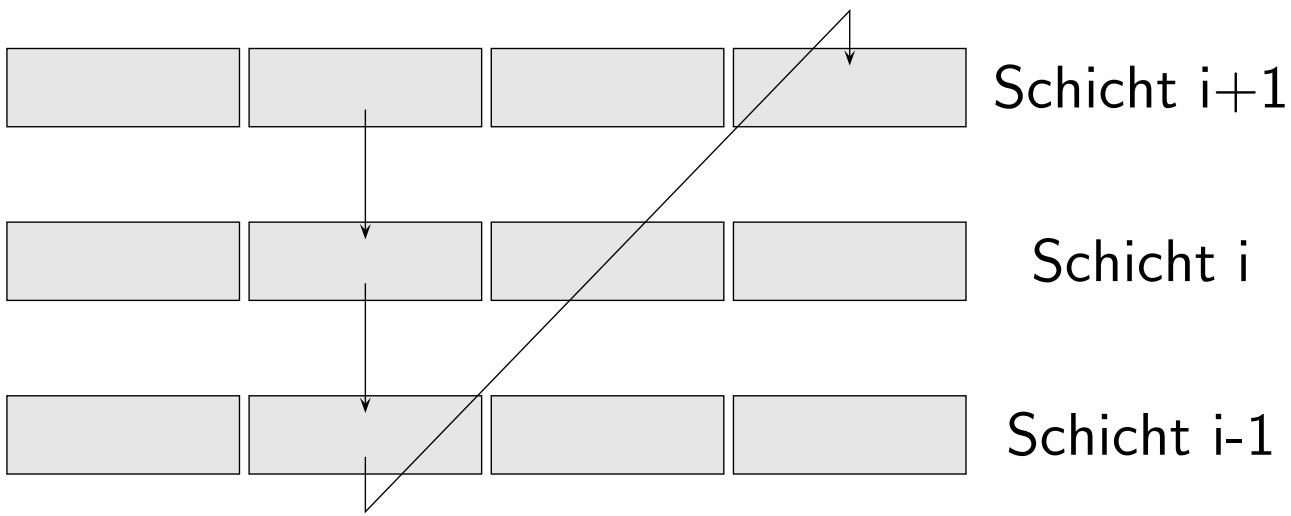


Hinweise

- Jede Schicht (z. B. Betrieb) kann partitioniert sein.

Betrieb Gerät A	Betrieb Gerät B	Betrieb Gerät C	Betrieb Gerät D

- Aufwärtsaufrufe (z. B. von Betrieb zu Verwaltung) sind auch erlaubt, solange keine Zyklen entstehen:



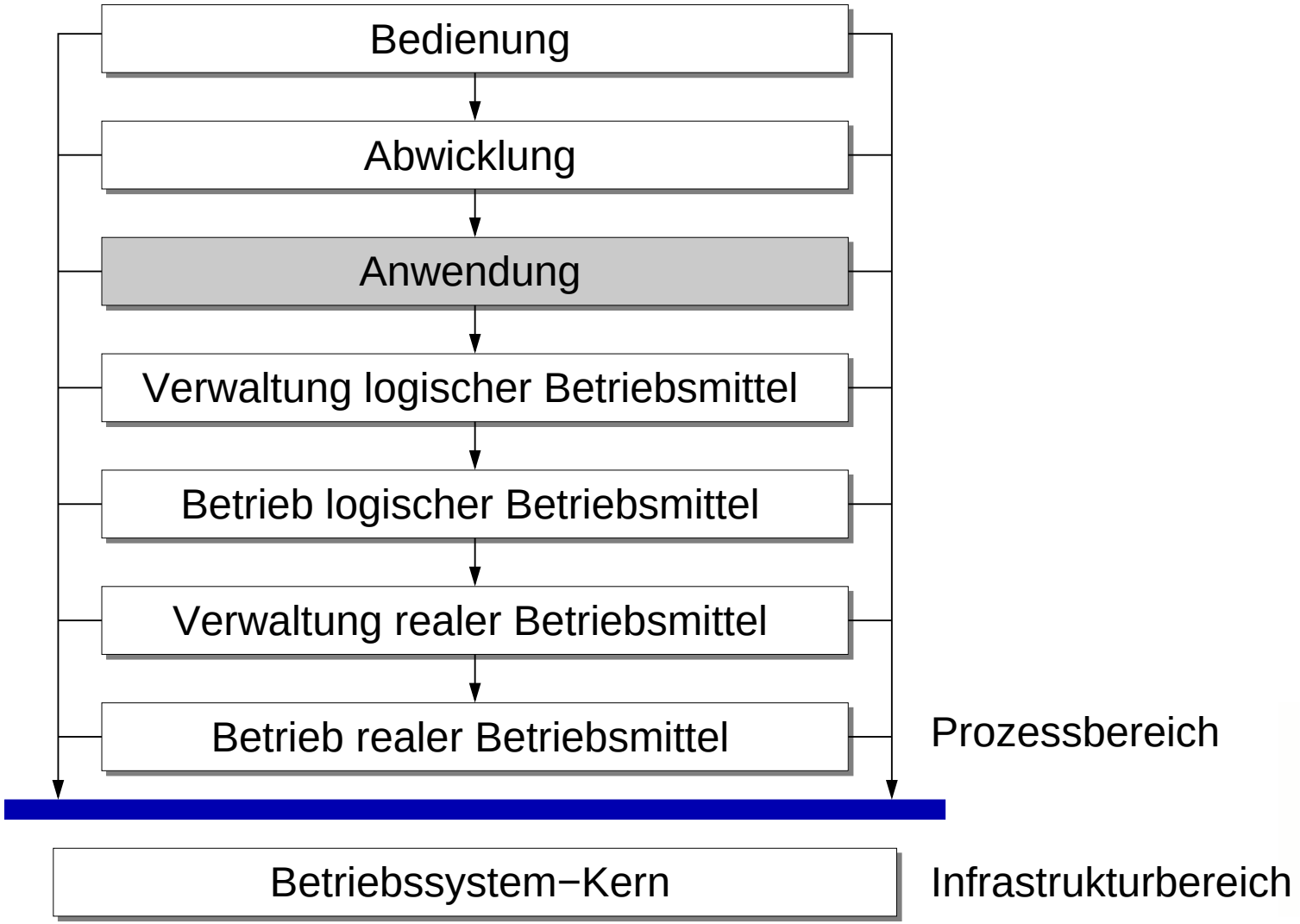
Steuerung

Bei der Steuerung des Ablaufs (Folie 17 , oben) unterscheidet man oft zwischen zwei Arten:

- **Bedienung**: Interaktion zwischen Mensch und System
  - Benutzerschnittstelle (graphisch), Fenstersysteme
  - BS-Kommandos sowie komplexe Aufträge ans Betriebssystem
- **Abwicklung**:
  - z. B. durch eine Programmiersprachliche Notation mit eingebetteten BS-Kommandos zum Steuern komplexer Aufträge (Shell).

Die nächste Folie zeigt die nun aktuelle Struktur

Übersicht


Bedienung
Abwicklung
Anwendung
Verwaltung logischer Betriebsmittel
Betrieb logischer Betriebsmittel
Verwaltung realer Betriebsmittel
Betrieb realer Betriebsmittel
Prozessbereich
Betriebssystem−Kern
Infrastrukturbereich

2.3 Die Kernschnittstelle: Systemaufrufe

- Systemaufrufe sind Betriebssystemfunktionen, die von Benutzerprogrammen aus aufgerufen werden können.
- Beispiele in Unix:
  - Prozessmanagement:
    `fork`: neuen Prozess starten
    `exit`: Prozess beenden
  - Dateimanagement:
    `open`: Datei öffnen
    `read`/`write`: aus Datei lesen/schreiben
  - Verzeichnismanagement
    `mkdir`: Verzeichnis anlegen
    `unlink`: Dateinamen entfernen
  - Verschiedenes:
    `chmod`: Zugriffsrechte für Datei ändern
    `kill`: Signal über das Beenden an Prozess schicken

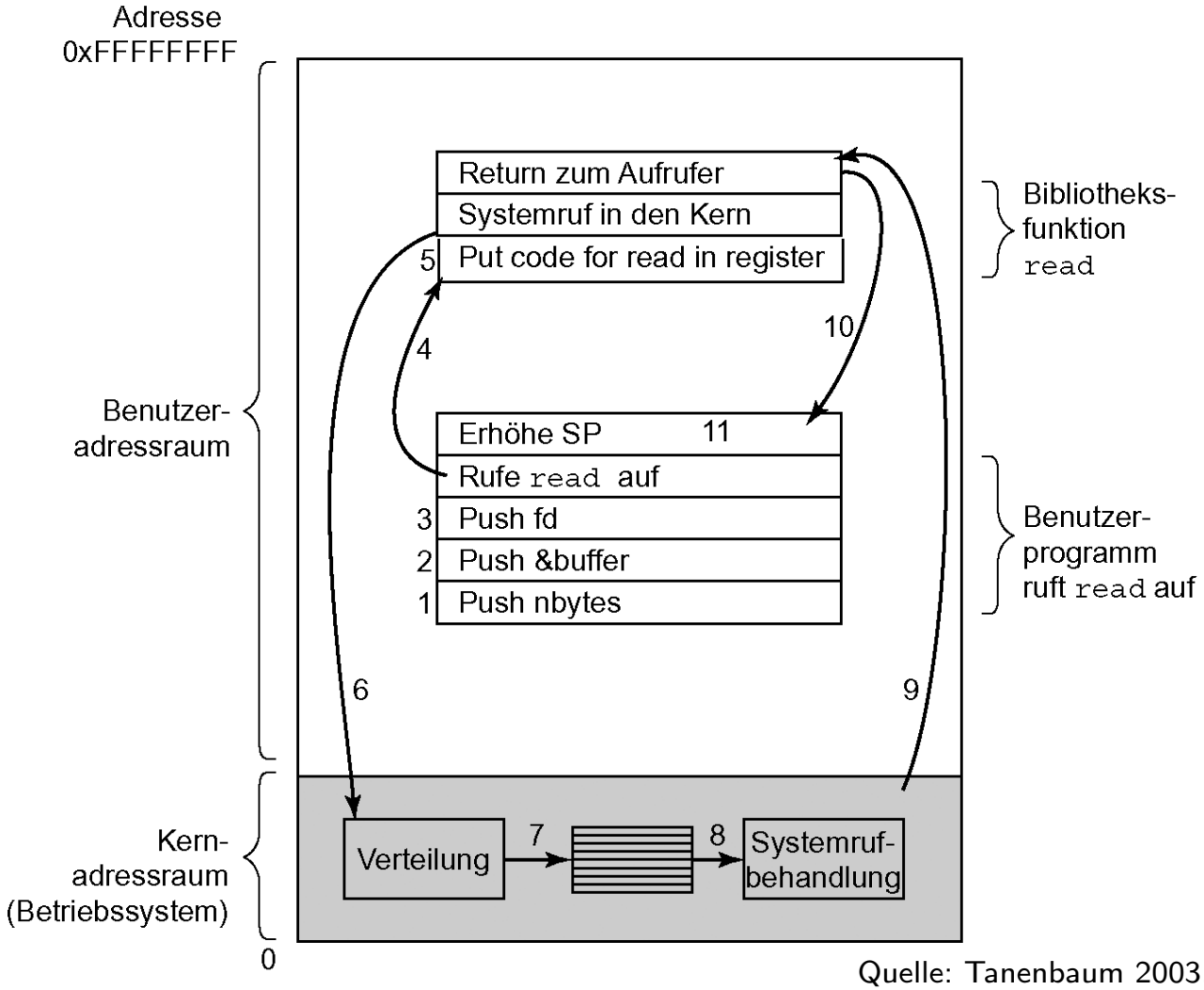
Systemaufrufe: Implementierung

- Konkrete Implementierung von Systemaufrufen ist abhängig von der Hardware und dem Betriebssystem
- Allgemeine Vorgehensweise:
  - Benutzerprogramm hinterlegt Parameter an einer vorher vereinbarten Stelle (Speicheradresse, Register) und signalisiert dem BS durch spezielle Befehle, dass ein Systemaufruf ausgeführt werden soll.
  - Das Benutzerprogramm wird unterbrochen (sog. TRAP-Befehl) und die Kontrolle dem BS übergeben (Eintritt in den Kern).
  - Das BS wertet die Parameter des Benutzerprogramms aus und führt den angeforderten Dienst aus.
  - Ergebnisse werden für das Benutzerprogramm hinterlegt und das Benutzerprogramm fortgesetzt.
- Systemaufruf ist somit meist mit einem Moduswechsel (Benutzer- $\rightarrow$ Systemmodus und zurück) verbunden

Kernschnittstelle: Systemaufrufe – Beispiel

- Wir betrachten einen konkreten Systemaufruf in Unix
- Systemaufruf `read` zum Lesen aus einer Datei besitzt drei Parameter: Dateideskriptor, Datenpuffer, Zeichenanzahl
- Aufruf in C:
  ```
  count = read(fd, buffer, nbytes);
  ```
- Es wird durch `count` die Anzahl tatsächlich gelesener Zeichen zurückgeliefert, sie kann evtl. kleiner als `nbytes` sein
- Wurde der Systemaufruf nicht erfolgreich ausgeführt, dann wird `count` auf -1 gesetzt, die Fehlernummer wird hierbei in die globale Variable `errno` gelegt
- Jeder Systemaufruf wird in mehreren Schritten ausgeführt, siehe nächste Folie für `read`, der 11 Schritte braucht

Systemaufruf `read` – Beispiel



Quelle: Tanenbaum 2003.

- 1-3: Die Parameter werden auf den Stack gelegt
- 4-5: Sprung in die Bibliotheksfunktion (i.d.R. Assembler), Ablage der Systemaufruf-Nummer und der Adresse des künft. Resultats in einem Register

Systemaufruf `read` – Fortsetzung

- 6: Die TRAP-Funktion ausführen zum Wechseln in den Kernmodus, Sprung in den Kern
- 7-8: Finden (in einer Tabelle aus Funktionszeigern) und Ausführen des Systemaufrufs `read`
- 9-10: Kontrolle zurück an die Bibliotheksfunktion (die TRAP ausgeführt hat) geben und von dort zurück ans Benutzerprogramm
- 11: Stack aufräumen: Stackpointer erhöhen

Beachte: Im Schritt 9 kann der Systemaufruf den Aufrufer blockieren (z.B. Warten auf die Tastatur-Eingabe)

Beispiel: Linux

- Ganz ähnlich macht das Linux:
  `https://de.wikipedia.org/wiki/Systemaufruf#Linux`
- Auch andere BS mit grösseren Kernen verfahren vergleichbar.
- Linux bietet ziemlich viele Systemaufrufe . . . :
  `https://de.wikipedia.org/wiki/Liste_der_Linux-Systemaufruf`

Mikrokernel und Systemaufrufe

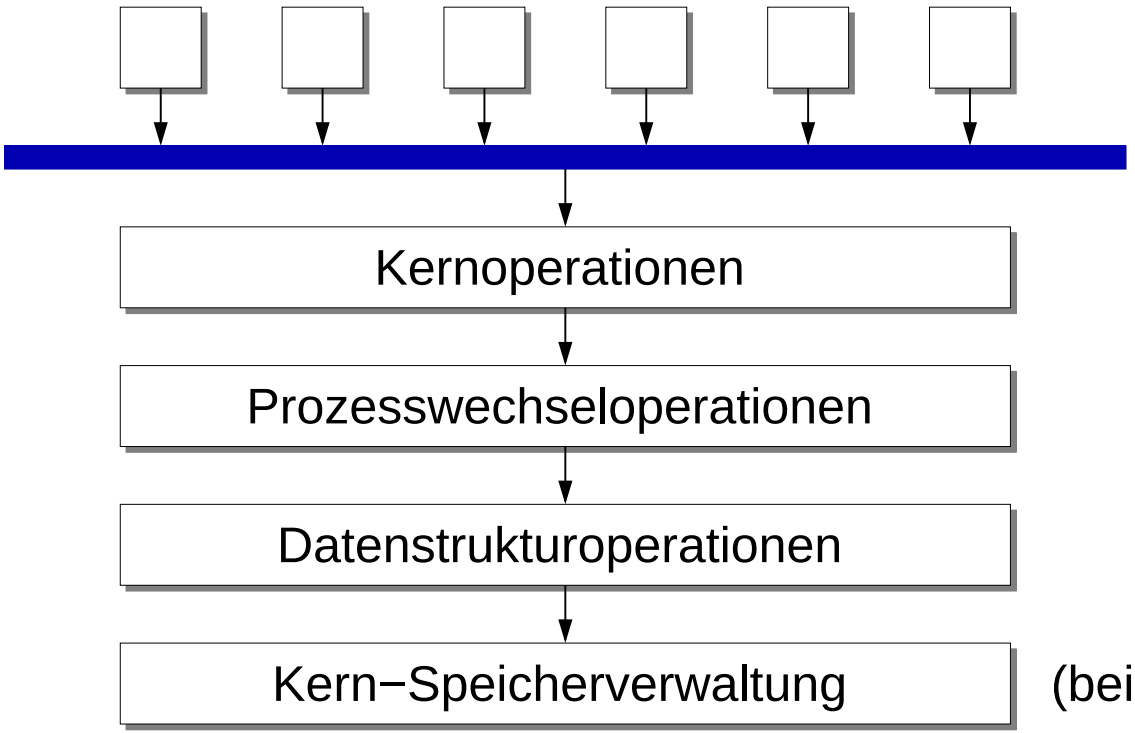
aus `https://www.gnu.org/software/hurd/system_call.html`:

> *In an UNIX-like system, a system call (syscall) is used to request all kinds of functionality from the operating system kernel.*
>
> *A microkernel-based system typically won't offer a lot of system calls – apart from one central one, and that is send message – but instead RPCs will be used instead. See GNU Mach's system calls.*
>
> *In the GNU Hurd , a lot of what is traditionlly considered to be a UNIX system call is implemented (primarily by means of RPC) inside glibc.*

RPC = Remote Procedure Call, wird später erklärt.

2.4 Der Kern

weitere Auflösung **Kern**,
Details dazu – im nächsten Kapitel



Prozesse

Kernschnittstelle

(bei dynamischen Systemen)