



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.1



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm



Bild von Mike by Pexels

# H | Ein-, Ausgabe und Geräteverwaltung

(Grundlagen der) Betriebssysteme

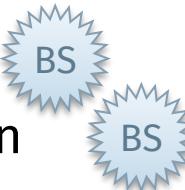


Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Überblick

## Überblick der Themenabschnitte

- A – Organatorisches
- B – Zahlendarstellung und Rechnerarithmetik
- C – Aufbau eines Rechnersystems
- D – Einführung in Betriebssysteme
- E – Prozessverwaltung und Nebenläufigkeit
- F – Dateiverwaltung
- G – Speicherverwaltung
- H – Ein-, Ausgabe und Geräteverwaltung
- I – Virtualisierung
- J – Verklemmungen
- K – Rechteverwaltung



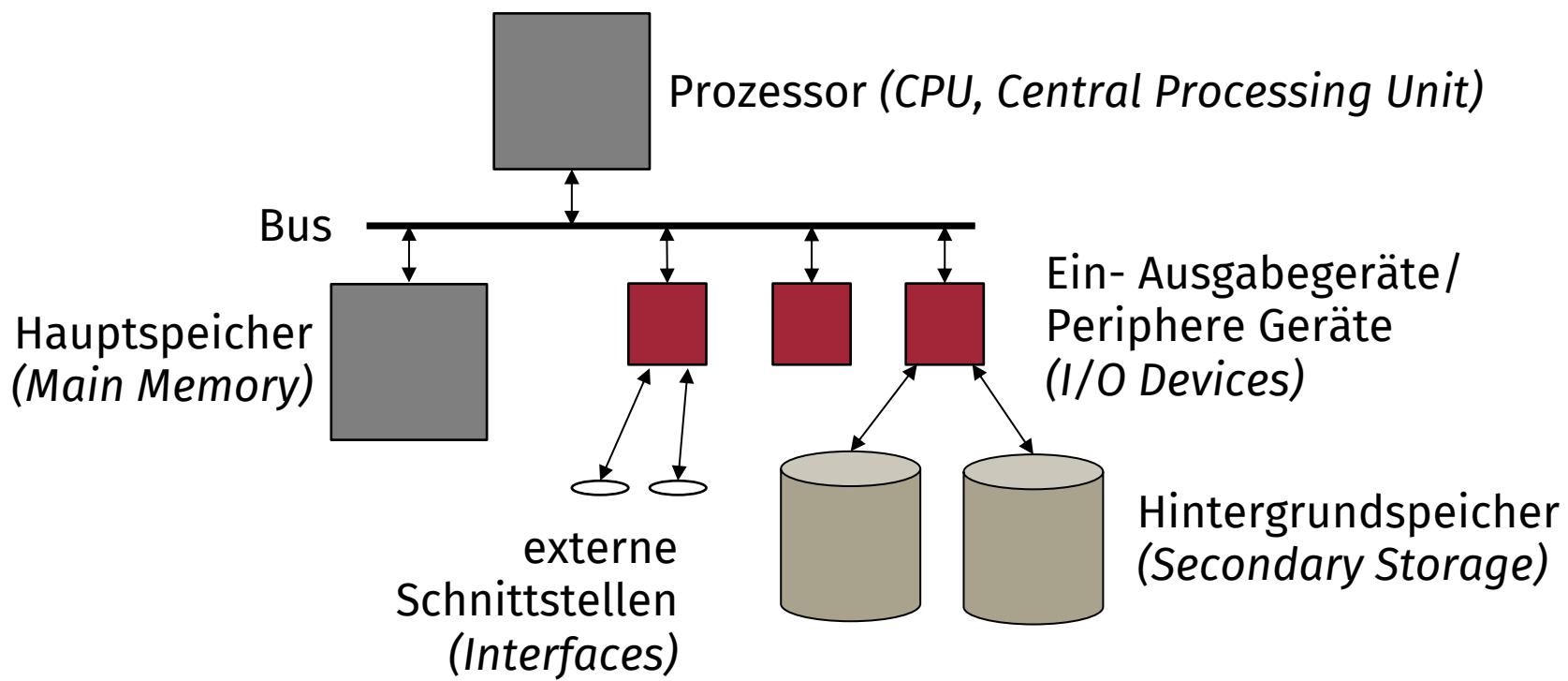
# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Einordnung

## Betroffene physikalische Ressourcen



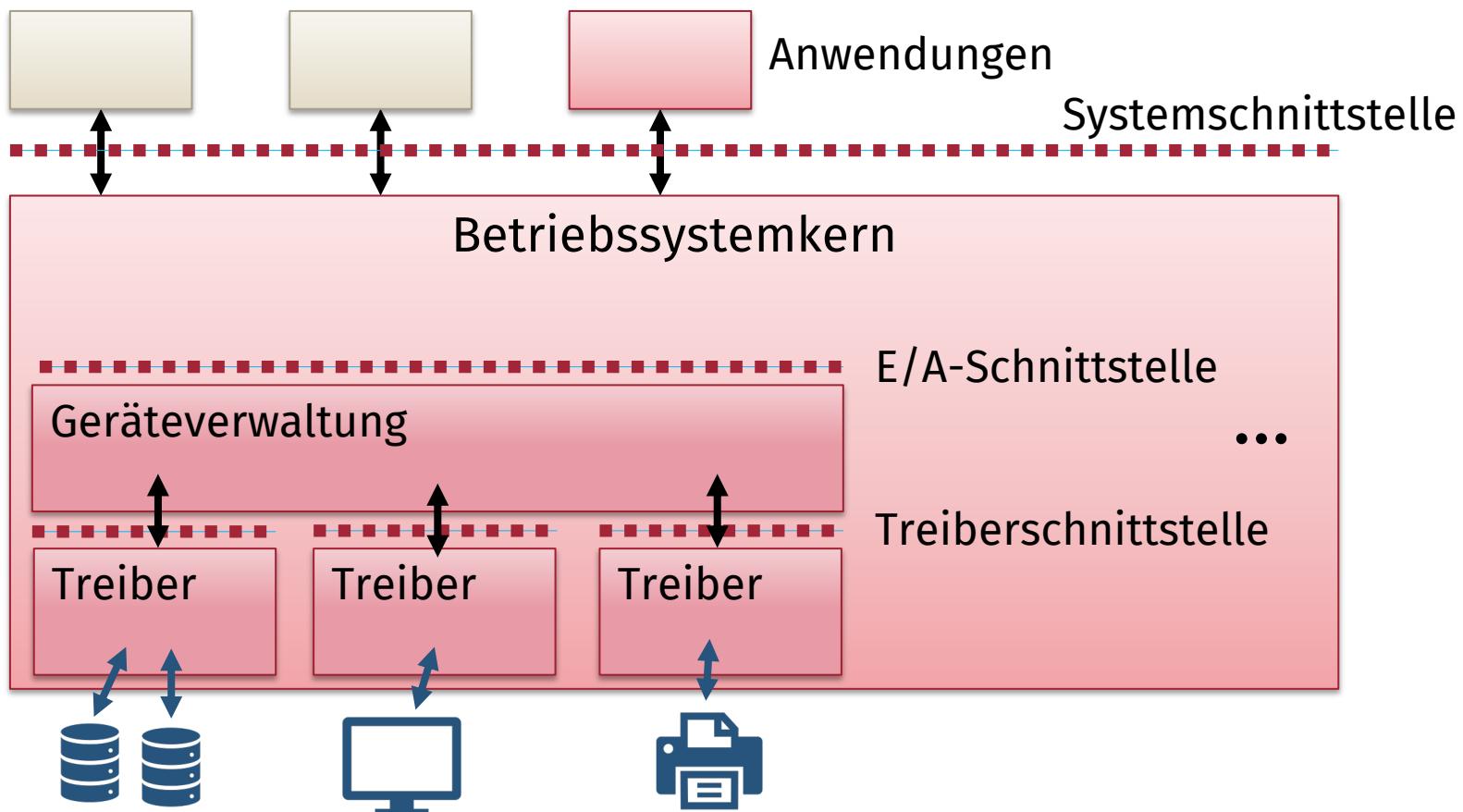
# Geräte

## Übliche Ein-/Ausgabegeräte

- Speichermedien (z.B. Festplatten, CD-ROM, DVD-ROM)
  - schon im Kapitel „Dateiverwaltung“ angesprochen
- Netzwerkkarten
  - WiFi, Bluetooth, LTE, Ethernet etc.
- Human Interface Devices (HIDs)
  - Tastatur, Maus etc.
- externe I/O-Schnittstellen (z.B. USB, RS232, Firewire)
  - zum Ansprechen verschiedenster Geräte
  - z.B. Scanner, Drucker, Fax, Web-Cams, Mikrofone, MIDI-Geräte etc.

# Geräteverwaltung

## Schichtung der Systemsoftware bis zum Gerät



# Geräteverwaltung (2)

## Schnittstellen

- einheitliche Operationen zum Zugriff
  - identisch für unterschiedlichste Geräte
- Ein-/Ausgabeschnittstelle
  - Nutzung innerhalb des Kerns oder durch die Anwendung
- Treiberschnittstelle
  - Nutzung innerhalb der Geräteverwaltung
  - geräteunabhängig
  - häufig ähnlich zur E/A-Schnittstelle

# Geräteverwaltung (3)

## Treiber

- Software-Modul für dedizierte Geräte
  - z.B. für einen bestimmten Festplattenkontroller
- Entlastung des Kerns von notwendigen Spezialbehandlungen
- implementiert Treiberschnittstelle für dedizierte Geräte
  - Programmierung der E/A-Register
  - Behandlung von gerätespezifischen Unterbrechungen

## Geräteverwaltung

- verwaltet Treiber und zugehörige Geräte

# Treiber

## Aufgaben des Treibers

- Verwaltung von Zugriffsrechten
  - Wer darf was?
- optimale Nutzung der Geräte
  - strategische Entscheidungen innerhalb des Treibers
  - z.B. optimaler Plattendurchsatz
- Zuteilung von Ressourcen an Benutzerprozesse und Betriebssystem
- Auflösung von Zugriffskonflikten
- Datentransport zum und vom Gerät



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.2



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Treiberimplementierung

## Problem

- große Latenzzeiten des Geräts
- Alternativen an der Treiberschnittstelle
  - Aufruf muss im Treiber warten bis Operation abgeschlossen
    - synchrone Ein-/Ausgabe
  - Aufruf kehrt vorzeitig zurück
    - asynchrone Ein-/Ausgabe
    - Operation wird (nur) angestoßen
    - evtl. nachträgliche Synchronisation mit Abschluss der Operation nötig
- Alternativen im Treiber zur Implementierung des Wartens
  - Polling-Betrieb
  - Unterbrechungsbetrieb

# Treiberimplementierung (2)

## Polling-Betrieb

- Prozessor ist einzige aktive Instanz bei einer E/A-Operation
- Prozessor fragt E/A-Baustein ab, ob Teilaktion beendet
  - z.B. ob Daten vorliegen
- aktives Warten, falls Aktion nicht abgeschlossen

## Problem

- Prozessor ständig aktiv
  - **Vergeudung** von Rechenzeit
- **implizites Blockieren** anderer Aktivitätsträger
- heute nur sinnvoll bei **äußerst kurzen** Wartezeiten

# Treiberimplementierung (3)

## Unterbrechungsbetrieb

- Gerät kann Unterbrechung auslösen
  - z.B. Gerät meldet Beendigung eines Teilauftrags
- Aktivitätsträger kann blockiert werden
  - Prozessor wird frei für andere Aktivitätsträger
- Unterbrechungsbehandlung führt E/A-Operation fort
  - weckt schließlich Aktivitätsträger wieder auf (deblockiert)

## Problem

- **komplexes** Programmiermodell
- **Zusatzaufwand** durch Unterbrechungsbehandlung
  - Registersicherung etc.

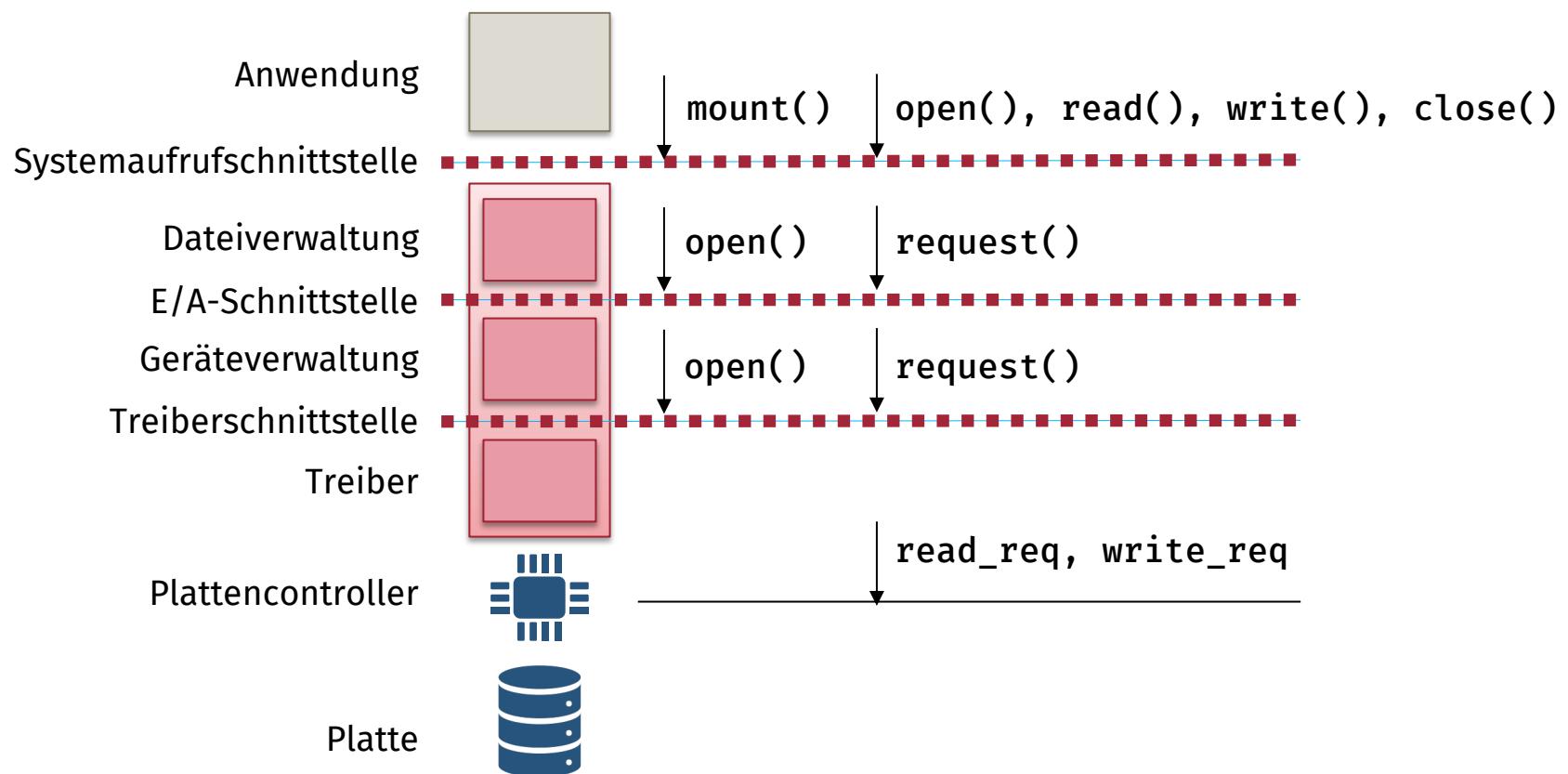
# Treiberimplementierung (4)

## Unterbrechungsbetrieb (fortges.)

- typische Abwicklung mehrerer E/A-Aufträge
  - interne Warteschlange für Aufträge
  - Blockierung des Aktivitätsträgers (synchrone E/A)
  - strategische Abarbeitung der Aufträge
  - Freigabe blockierter Aktivitätsträger mit beendeten E/A-Aufträgen
    - in der Unterbrechungsbehandlung
  - Anstoßen des nächsten Auftrags
    - aus der Warteschlange
    - in der Unterbrechungsbehandlung
- Synchronisierung bei asynchronen E/A-Operationen
  - wie synchrone Abarbeitung

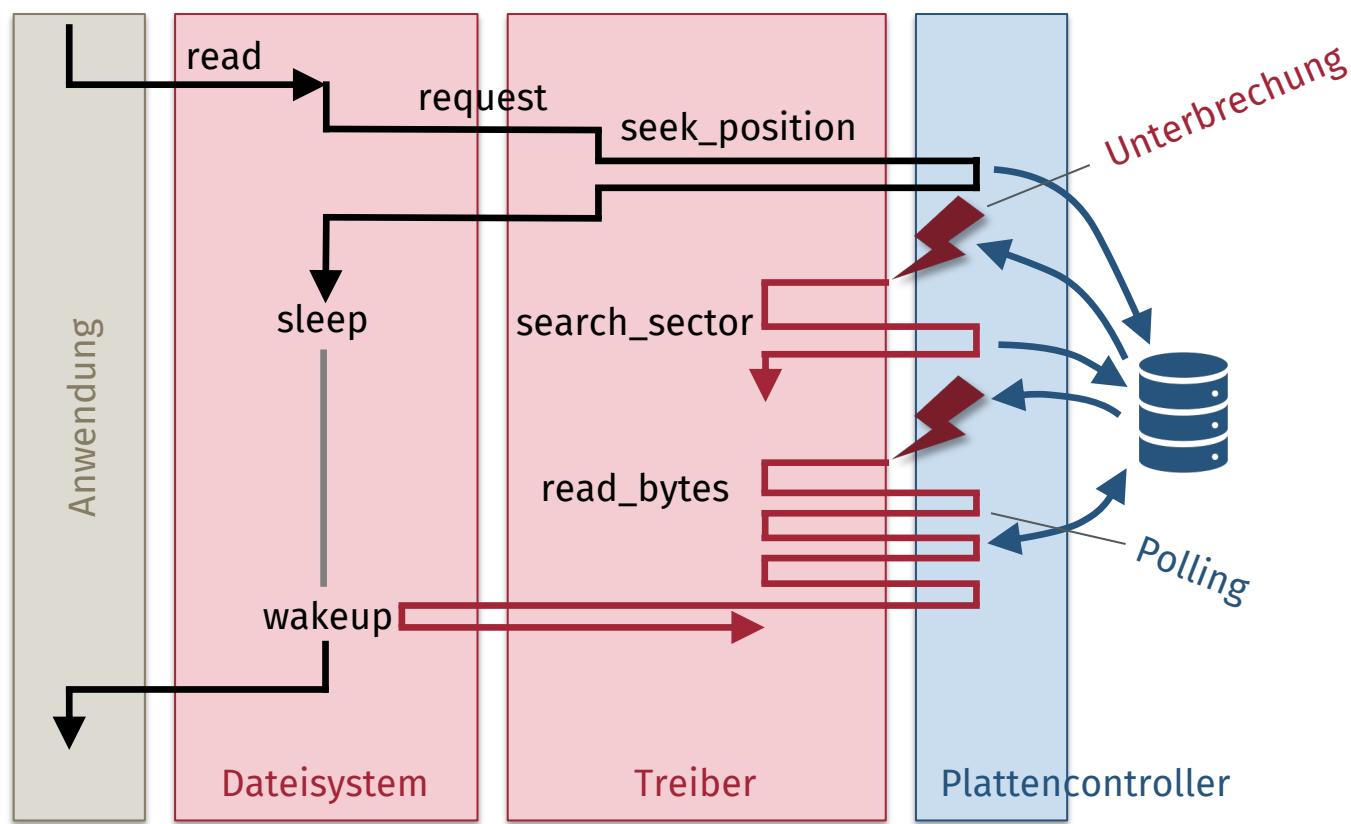
# Plattentreiber

## Software und Hardware zwischen Anwender und Platte



# Einfacher Plattentreiber

## Ablauf eines synchronen Leseaufrufs



## Einfacher Plattentreiber (2)

Anwendung führt lesenden Systemaufruf aus

- Dateisystem prüft, ob Block im Speicher vorhanden (Cache)
- falls Block nicht vorhanden
  - Speicherplatz wird bereitgestellt
  - `request()` im entsprechenden Treiber aufgerufen
- Ausführung von `request()` stößt Plattenpositionierung an
- Prozess blockiert sich im Kernel
  - Scheduler lässt andere Prozesse laufen
- Plattencontroller meldet sich nach erfolgter Positionierung
  - Unterbrechungsbehandlung führt Auftrag fort
    - stößt Sektorsuche an

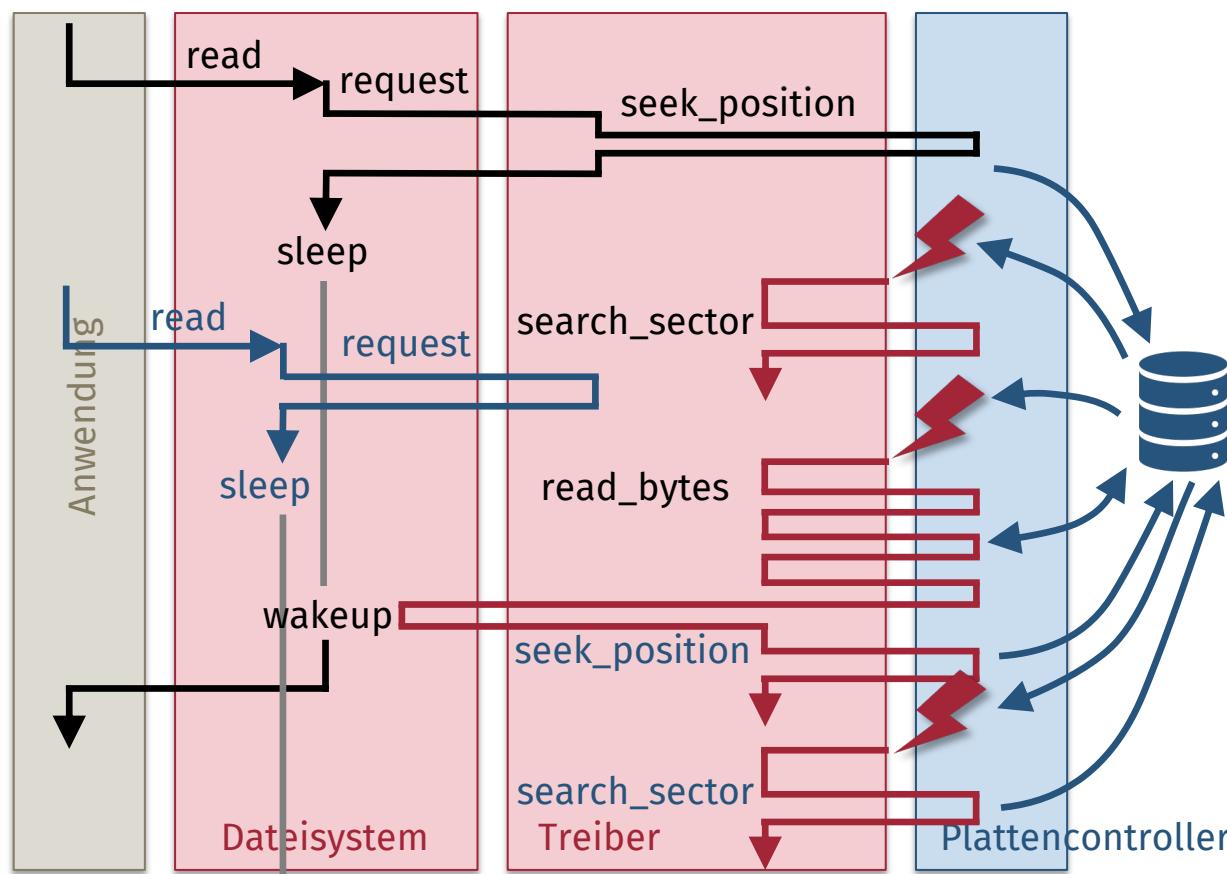
## Einfacher Plattentreiber (3)

Anwendung führt lesenden Systemaufruf aus (fortges.)

- Plattencontroller meldet sich sobald Sektor durchläuft
  - Unterbrechungsbehandlung liest Daten im Pollingbetrieb
- Prozess wird aufgeweckt bzw. deblockiert
  - d.h. in den Zustand „bereit“ überführt

## Einfacher Plattentreiber (4)

### Ablauf mehrerer synchroner Leseaufrufe



## Einfacher Plattentreiber (5)

### Unterbrechungsbehandlung für weitere Aufträge zuständig

- zugehörige Prozesse sind alle blockiert
- Unterbrechungsbehandlung am Schluss eines Auftrags
  - beendet den Auftrags
  - deblockiert zugehörigen Prozess
  - Auswählen und Aufsetzen eines Folgeauftrags
    - falls vorhanden



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.3



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Direct Memory Access (DMA)

## Datentransport zwischen E/A-Gerät und Hauptspeicher

- bislang: nur durch Prozessor
  - einziger Nutzer des Speicherbusses
    - Bus-Master
    - legt Adressen an, erzeugt Kontrollsignale, ...
  - Prozessor liest aus Speicher und übergibt Daten an E/A-Baustein
  - Prozessor liest Daten vom I/O-Baustein und schreibt in Speicher
- evtl. große Datenmengen
  - z.B. bei Festplatten
- ständige Wartezeiten durch relativ langsame I/O-Geräte

## Direct Memory Access (2)

### Idee

- Entlastung des Prozessors durch eine eigene Transfereinheit
- spezielle **Hardware** tritt als Aktor auf dem Speicherbus auf
  - liest Daten und übergibt sie an I/O-Baustein, oder umgekehrt
  - CPU ist währenddessen **frei**
    - lediglich Speicherbus teilw. durch DMA belegt

### Programmierung des DMA-Systems

- Länge, Speicheradresse, E/A-Adresse von Ein-Ausgaberegister
- **Unterbrechung** am Ende des Transfers
  - oder bei Fehlern

# Direct Memory Access (3)

## Arbeitsmodi

### ■ Burst-Modus

- DMA-Baustein übernimmt Systembus für die Dauer des vollständigen Transfers
- **Vorteil:** hohe Transferrate möglich
- **Nachteil:** CPU für lange Zeit am Speicher-/Buszugriff gehindert

# Direct Memory Access (4)

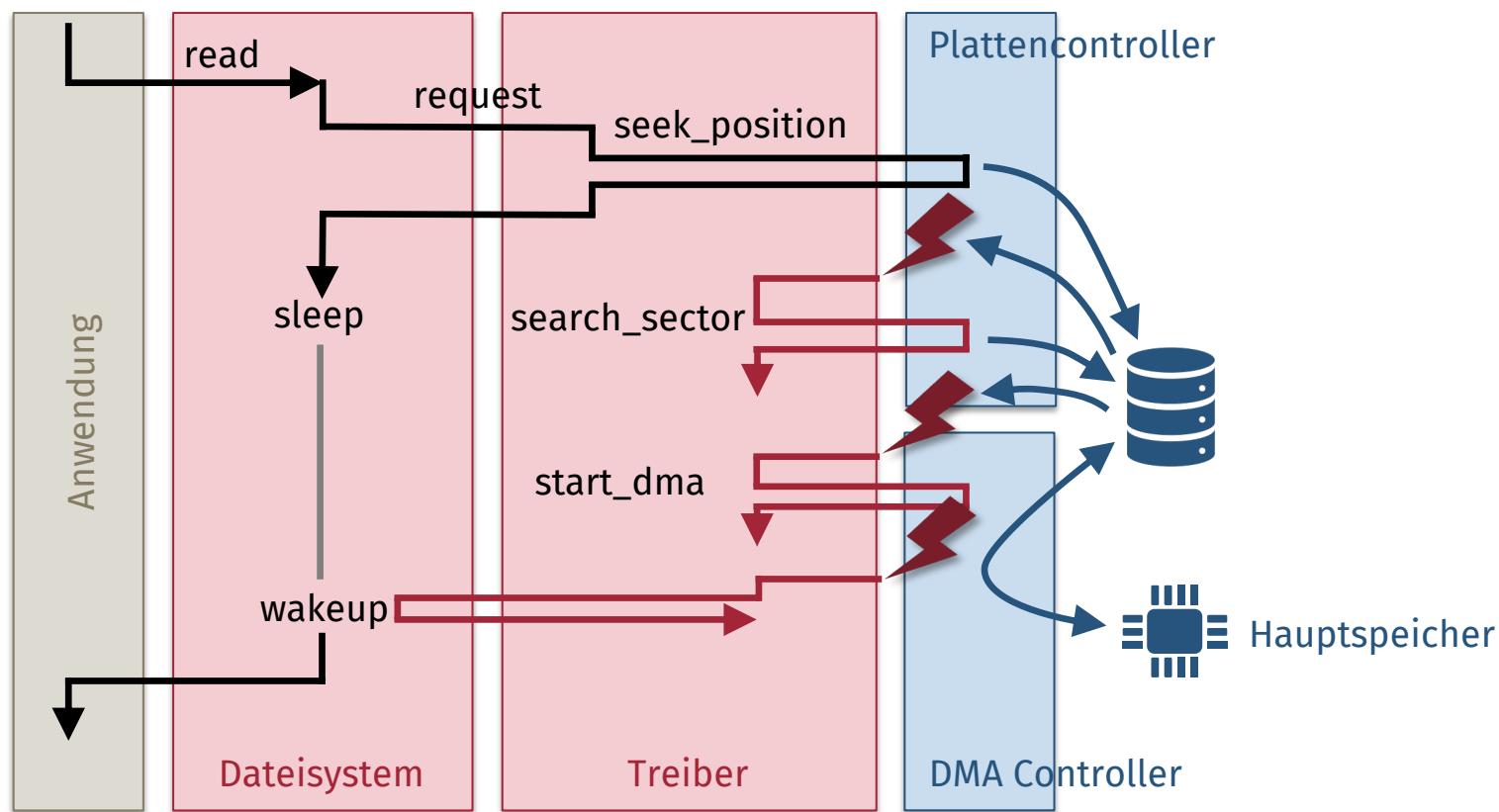
## Arbeitsmodi (fortges.)

### ■ Cycle Stealing

- Mischen von DMA- und CPU-Buszyklen
- z.B. fester Anteil an Buszyklen (z.B. jeder zweite) wird vom DMA-Baustein der CPU „gestohlen“
- DMA nutzt zusätzlich alle von der CPU nicht benötigten Buszyklen
  - z.B. während der Bearbeitung von Instruktionen, die nur CPU-interne Ressourcen benötigen
- **Vorteil:** CPU nur geringfügig behindert
- **Nachteil:** geringere Transferrate

# Plattentreiber mit DMA

## Ablauf eines synchronen Leseaufrufs



# Treiber mit DMA

## Große Systeme mit mehreren DMA-Kanälen und vielen Platten

- Suche nach freiem DMA-Kanal
  - evtl. Blockierung bis einer frei
- Anforderung des DMA-Kanals kann evtl. parallel zur Plattenpositionierung erfolgen

## Mainframe-Systeme

- Steuereinheit fasst mehrere Platten zu einem Gerät zusammen
- mehrere Steuereinheiten hängen an einem Kanal zum Hauptspeicher
  - zum Zugriff auf die eigentliche Platte muss erst die Steuereinheit und dann der Kanal belegt werden (Teilwegbelegung)

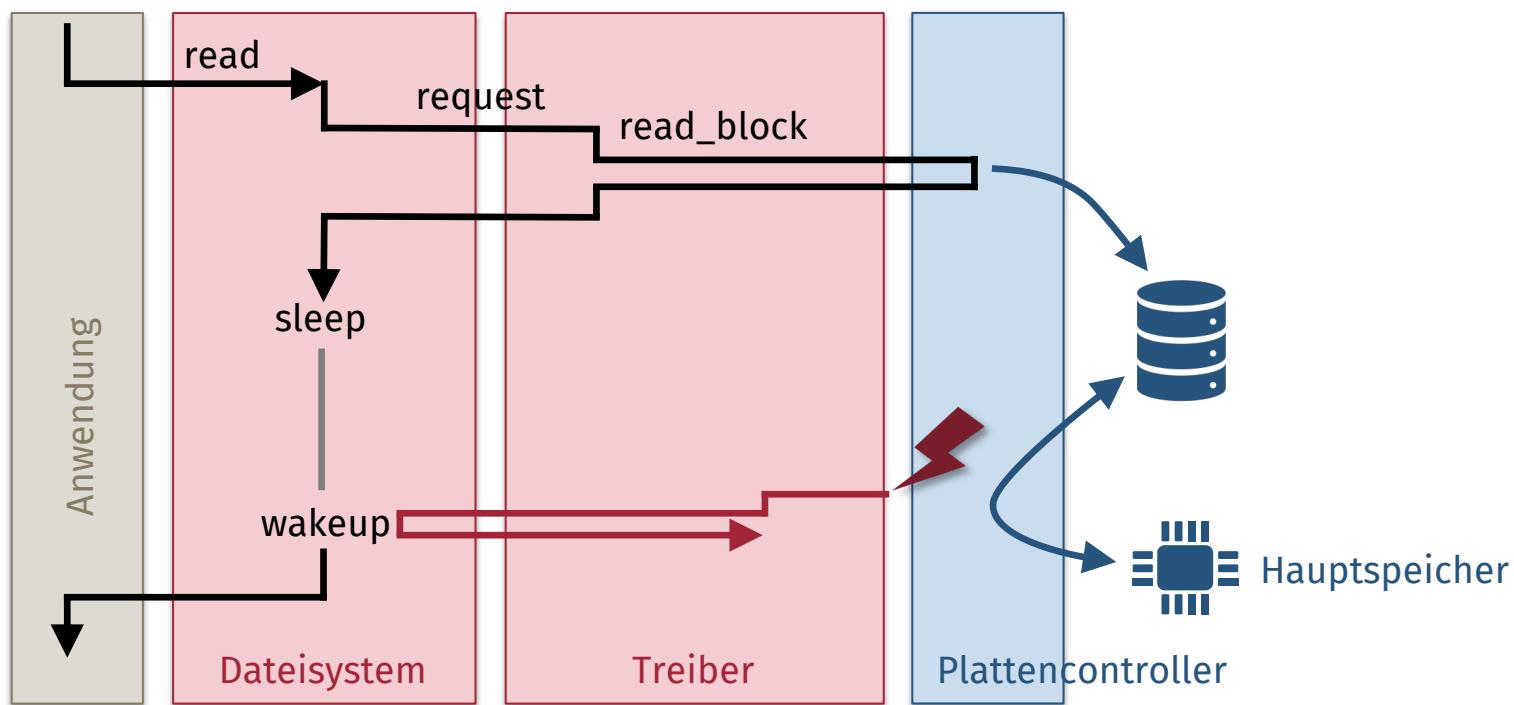
## Treiber mit DMA (2)

### DMA und Caching

- heutige Prozessoren arbeiten mit Datencaches
- DMA läuft am Cache vorbei
  - Betriebssystem muss vor dem Aufsetzen von DMA-Transfers Caches zurückschreiben und invalidieren

# Plattentreiber für Bus-Master-Controller

Heutige Festplattencontroller können Bus-Master sein



- Plattentreiber transferiert selbständig

## Andere Treiber

### Abläufe prinzipiell identisch

- DMA oft nur bei großen Datenmengen
- Bus-Mastering auch für kleine Datenmengen möglich
  - z.B. USB Tastaturen etc.
- Netzwerkkarten
  - vorgegebene Puffer im Speicher
  - Bus-Master füllt Puffer mit Netzwerkpaketen
  - Bus-Master sendet vorbereitete Nachrichten im Puffer über Netzwerk



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.4



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Anwendersicht unter UNIX/Linux

## Repräsentation von Ein-/Ausgabegeräten

- Spezialdatei im Dateisystem
  - blockorientierte Geräte
    - z.B. Platten, CD-ROM etc.
  - zeichenorientierte Geräte
    - z.B. USB-Geräte, serielle Schnittstellen, Audiokanäle etc.
    - fast immer auch zeichenorientierte Repräsentation von blockorientierten Geräten vorhanden
      - wird intern auf blockorientierte E/A umgesetzt
- können wie Dateien geöffnet werden
  - normale Systemaufrufe zum Lesen und Schreiben

# Anwendersicht unter UNIX/Linux (2)

## Repräsentation von Ein-/Ausgabegeräten (fortges.)

### ■ Spezialdatei im Dateisystem

- nur eine Art Verweis auf den Treiber
  - vergleichbar mit Symbolic Link (aber anders)
- besteht nur aus Inode, keine Daten
  - wird also in irgendeinem Dateisystem gespeichert
  - enthält eine **Major** und **Minor Number**
  - enthält Typ: **blockorientiert** oder **zeichenorientiert**

### ■ Major Number

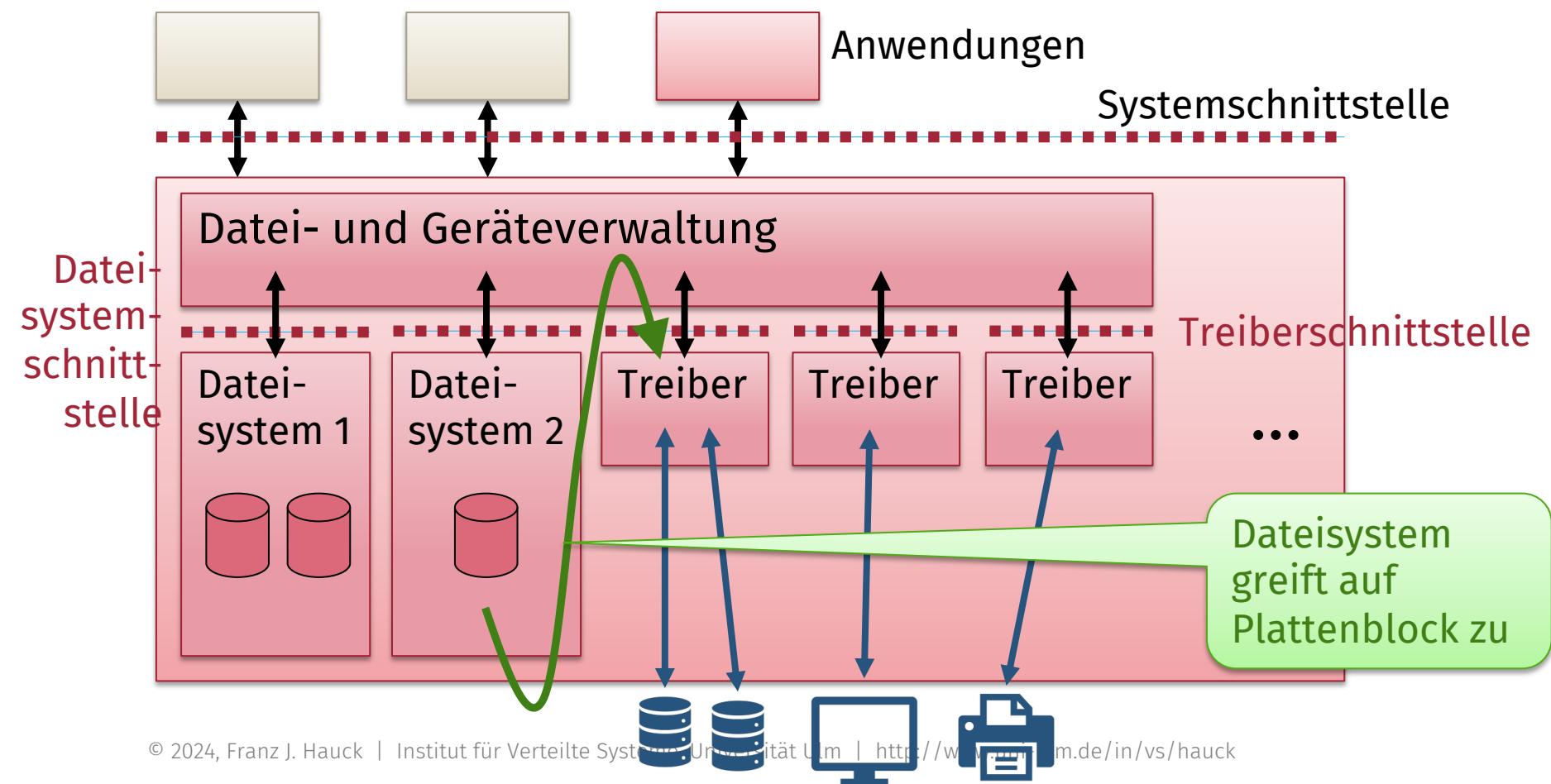
- verweist auf den Treiber

### ■ Minor Number

- verweist auf eines der vom Treiber verwalteten Geräte

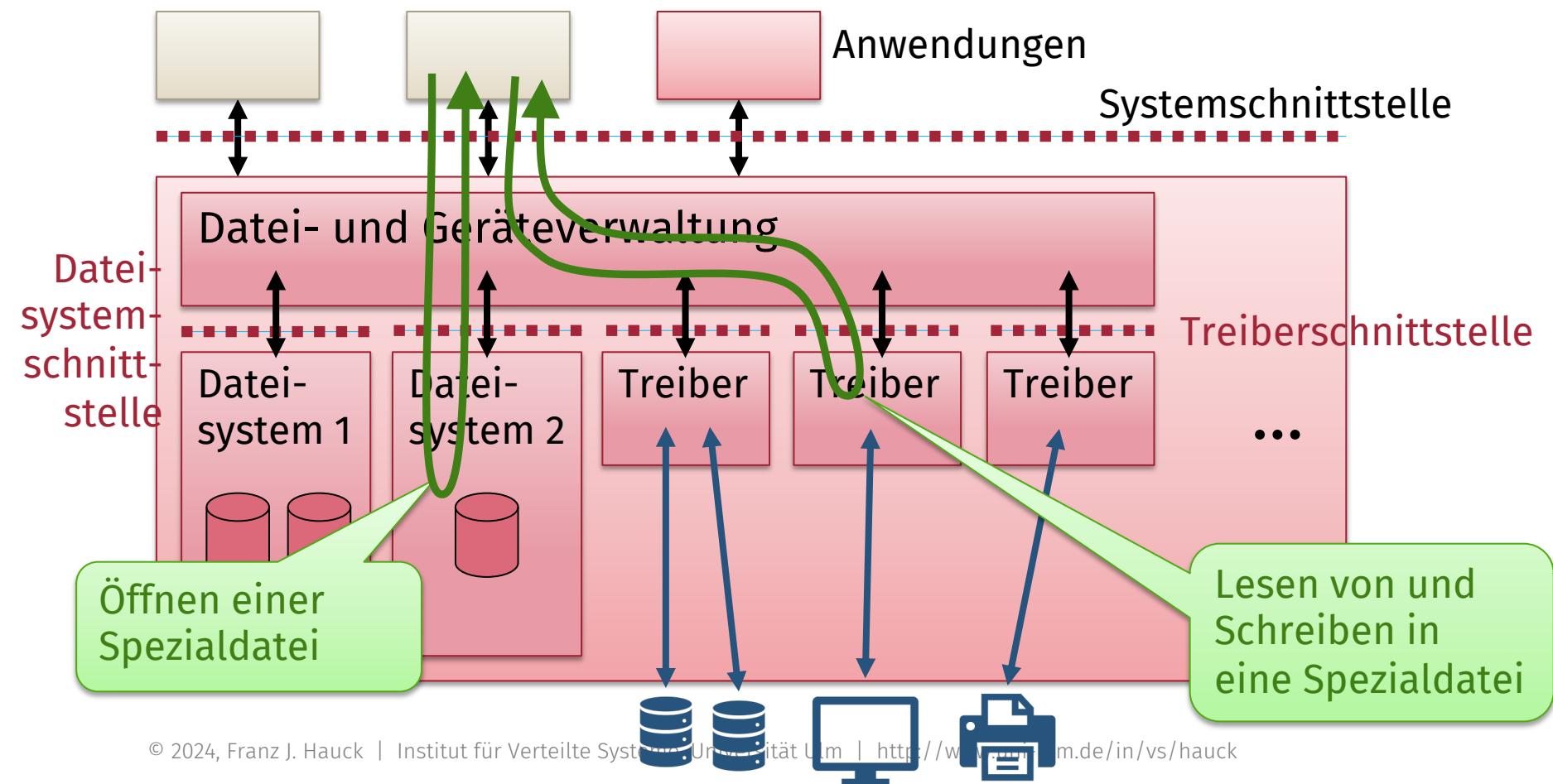
## Anwendersicht unter UNIX/Linux (3)

### Ein-/Ausgabeverwaltung und Dateisysteme



## Anwendersicht unter UNIX/Linux (4)

### Ein-/Ausgabeverwaltung und Dateisysteme



## Anwendersicht unter UNIX/Linux (5)

### Beispiel eines Verzeichnislistings von /dev (Ausschnitt)

lrwxrwxrwx	root	root	3	Jan	11	14:09	dvd1	-> sr0
crw-----	root	root	5,	1	Jan	11	08:05	console
lrwxrwxrwx	root	root	3	Jan	11	14:09	dvdrw1	-> sr0
crw-rw----	root	video	29,	0	Jan	11	08:04	fb0
brw-rw----	root	disk	8,	0	Jan	11	08:05	sda
brw-rw----	root	disk	8,	1	Jan	11	08:05	sda1
brw-rw----	root	disk	8,	2	Jan	11	08:05	sda2
brw-rw----	root	disk	8,	3	Jan	11	08:05	sda3
brw-rw----	root	disk	8,	4	Jan	11	08:05	sda4
brw-rw----+	root	cdrom	11,	0	Jan	11	14:09	sr0
crw-rw-rw-	root	tty	5,	0	Jan	11	08:04	tty

↑                   ↑                   ↑                   ↑                   ↑  
Zugriffsrechte      Eigentümer      Major +      Name  
c: Character Device                   Minor Number      letzte      Modifikation  
b: Block Device                        Number  
l: Symbolic Link

## Treibergerüst für Linux (zeichenorientiert)

```
static struct file_operations fops;

static int __init mod_init( void ) {
    if ( register_chrdev( 240, "DummyDriver", &fops ) == 0 )
        return 0;          // Treiber erfolgreich angemeldet
    return -EIO;         // Anmeldung beim Kernel fehlgeschlagen
}

static void __exit mod_exit( void ) {
    unregister_chrdev( 240, "DummyDriver" );
}

module_init( mod_init );
module_exit( mod_exit );
```

Funktion  
zum Start

Funktion  
zum Stop

## Treibergerüst für Linux (2)

```
static struct file_operations fops;

static int __init mod_init( void ) {
    if (register_chrdev( 240, "DummyDriver", &fops ) == 0 )
        return 0;          // Treiber erfolgreich angemeldet
    return -EIO;         // Anmeldung beim Kernel fehlgeschlagen
}

static void __exit mod_exit( void ) {
    unregister_chrdev( 240, "DummyDriver" );
}

module_init( mod_init );
module_exit( mod_exit );
```

Anmeldung mit Major-Number 240

## Treibergerüst für Linux (3)

```
static int dummy_open( struct inode *geraete_datei,
                      struct file *instanz ) {
    printk( "driver_open called\n" ); return 0;
}

static int dummy_close( struct inode *geraete_datei,
                       struct file *instanz ) {
    printk( "driver_close called\n" ); return 0;
}
```

Funktion zum  
Schließen der  
Spezialdatei

Funktion zum  
Öffnen der  
Spezialdatei

## Treibergerüst für Linux (4)

```
static char hello_world[] = "Hello World\n";  
  
static ssize_t dummy_read( struct file *instanz, char *user,  
                           size_t count, loff_t *offset ) {  
    int not_copied, to_copy;  
  
    to_copy = strlen( hello_world ) + 1;  
    if ( to_copy > count ) to_copy = count;  
    not_copied = copy_to_user( user, hello_world, to_copy );  
    return to_copy - not_copied;  
}  
  
static struct file_operations fops = {  
    .open = dummy_open,  
    .release = dummy_close,  
    .read = dummy_read  
};
```

Funktion zum Lesen aus der Spezialdatei

Funktion zum Datentransfer in den Prozess

# Blockorientierte Geräte unter UNIX/Linux

## Globaler Cache für Blöcke

- Block Buffer Cache
- im Hauptspeicher
  - Bezug zu Gerät wird verwaltet
    - Major Number, Minor Number, logische Blocknummer
- Lese- und Schreiboperationen automatisch über Cache
- Zusammenhang mit Speicherverwaltung
  - Seitenein-, auslagerung vs. Block Lesen und Schreiben
  - Einblenden von Dateien in den logischen Adressraum
    - z.B. ausführbare Datei



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.5



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Treiber für serielle Schnittstellen

## Treiber

- zeichenorientiertes Gerät
  - z.B. `/dev/tty` unter Linux
- vom Prinzip her ähnlich dem Plattentreiber
  - zeichenorientierte statt blockorientierte Übertragung

## Parametereinstellung

- neben `read()` und `write()` weiterer Systemaufruf:
  - `ioctl()` - I/O Control
- treiberabhängige Parameter zum Konfigurieren des Geräts
  - für serielle Schnittstellen festgelegte Einstellungen
  - Treiber übernimmt Editierfunktionen sowie Softwareflusskontrolle

# Gerätekonfiguration

Spezielle Geräteeigenschaften werden über ioctl angesprochen

IOCTL(2)

Linux Programmer's Manual

IOCTL(2)

## NAME

ioctl - control device

## SYNOPSIS

```
#include <sys/ioctl.h>

int ioctl(int fd, int request, ...);
```

■ Schnittstelle generisch, Semantik gerätespezifisch

## CONFORMING TO

No single standard. Arguments, returns, and semantics of **ioctl()** vary according to the device driver in question (the call is used as a catch-all for operations that don't cleanly fit the UNIX stream I/O model). The **ioctl()** function call appeared in Version 7 of AT&T UNIX.

# Disk-Scheduling

## Strategische Entscheidung innerhalb eines Plattentreibers

- mehrere Aufträge in der Warteschlange
- bestimmte Ordnung der Aufträge kann Effizienz steigern

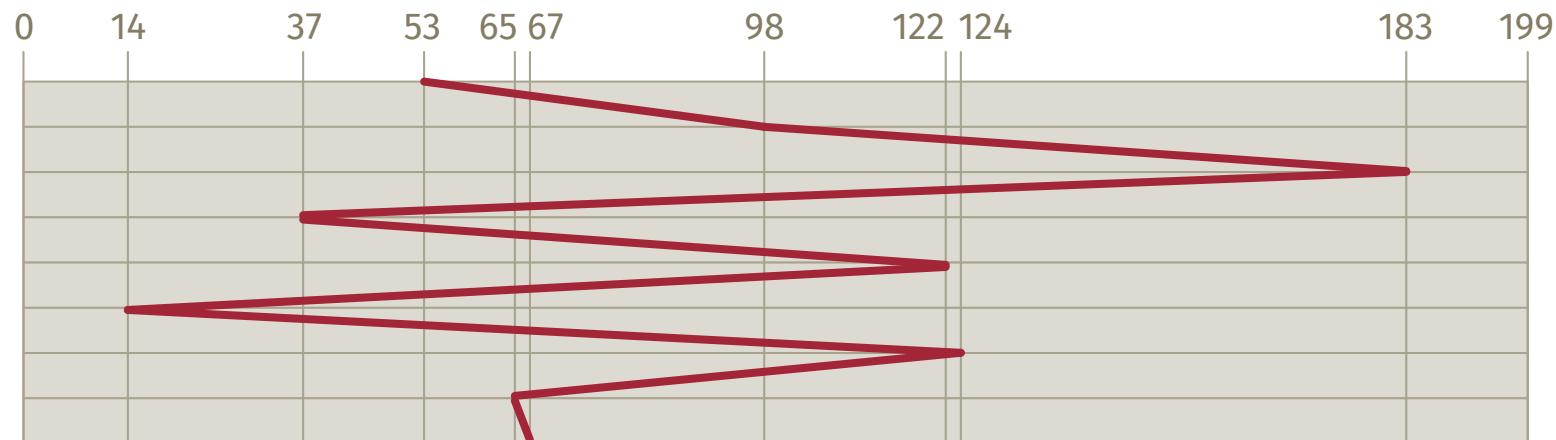
## Zusammensetzung der Bearbeitungszeit eines Plattenauftrags

- **Positionierzeit:** abhängig von der aktuellen Stellung des Plattenarms
  - **Latenzzeit:** Zeit, bis der Magnetkopf den Sektor bestreicht
  - **Übertragungszeit:** Zeit zur Übertragung der eigentlichen Daten
- ◆ **Ansatzpunkt:** Positionierzeit

# First-Come, First-Served Scheduling (FCFS)

Bearbeitung gemäß Ankunft des Auftrags

- Referenzfolge (Folge von Zylindernummern)
  - z.B. 98, 183, 37, 122, 14, 124, 65, 67
- aktueller Zylinder: 53

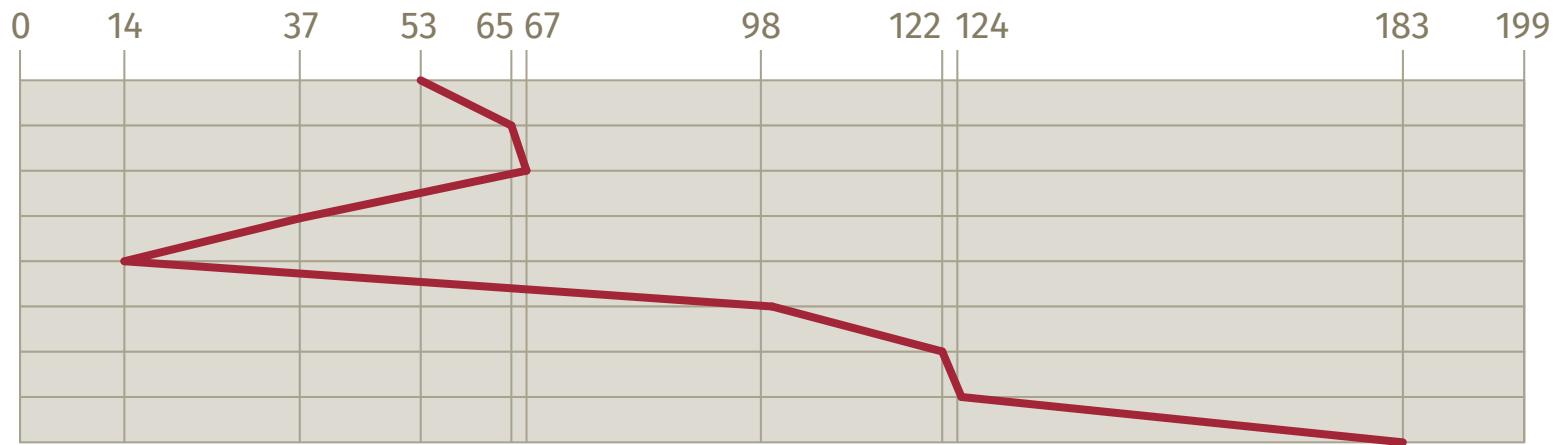


- Gesamtzahl der Spurwechsel: 640
  - Wegstrecke des Schreib-Lese-Kopfs in Anzahl Spurbreiten

# Shortest Seek Time First Scheduling (SSTF)

Auftrag mit der kürzesten Positionierzeit vorgezogen

- Tie-Break Heuristik bei gleich großer Positionierzeit
- Gleiche Referenzfolge
  - Annahme: Positionierzeit proportional zum Zylinderabstand

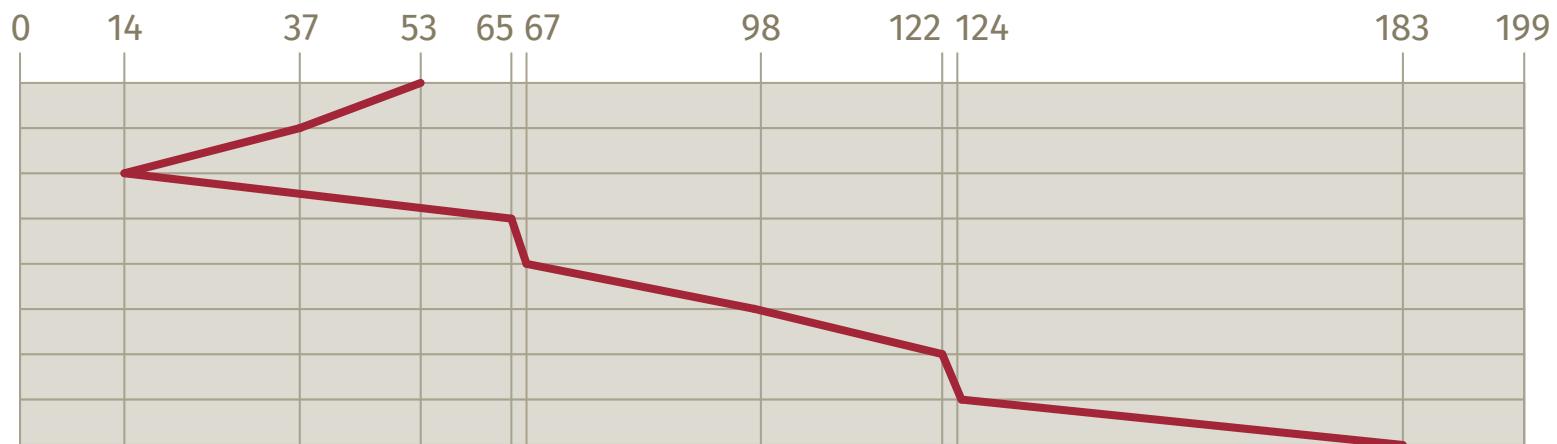


- Gesamtzahl der Spurwechsel: 236
  - SSTF kann zu Aushungerung führen: nicht optimal

## SCAN Scheduling

**Bewegung des Plattenarms immer in eine Richtung**

- bis keine Aufträge in dieser Richtung mehr vorhanden sind
  - Fahrstuhlstrategie
- gleiche Referenzfolge
  - Annahme: bisherige Kopfbewegung in Richtung 0



- Gesamtzahl der Spurwechsel: 208

## SCAN Scheduling (2)

### Beobachtung

- neue Aufträge werden miterledigt ohne zusätzliche Positionierzeit und ohne Aushungerung

### Variante C-SCAN (Circular SCAN)

- Bewegung immer nur in eine Richtung
- am Ende Rückkehr zum Anfang



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.6



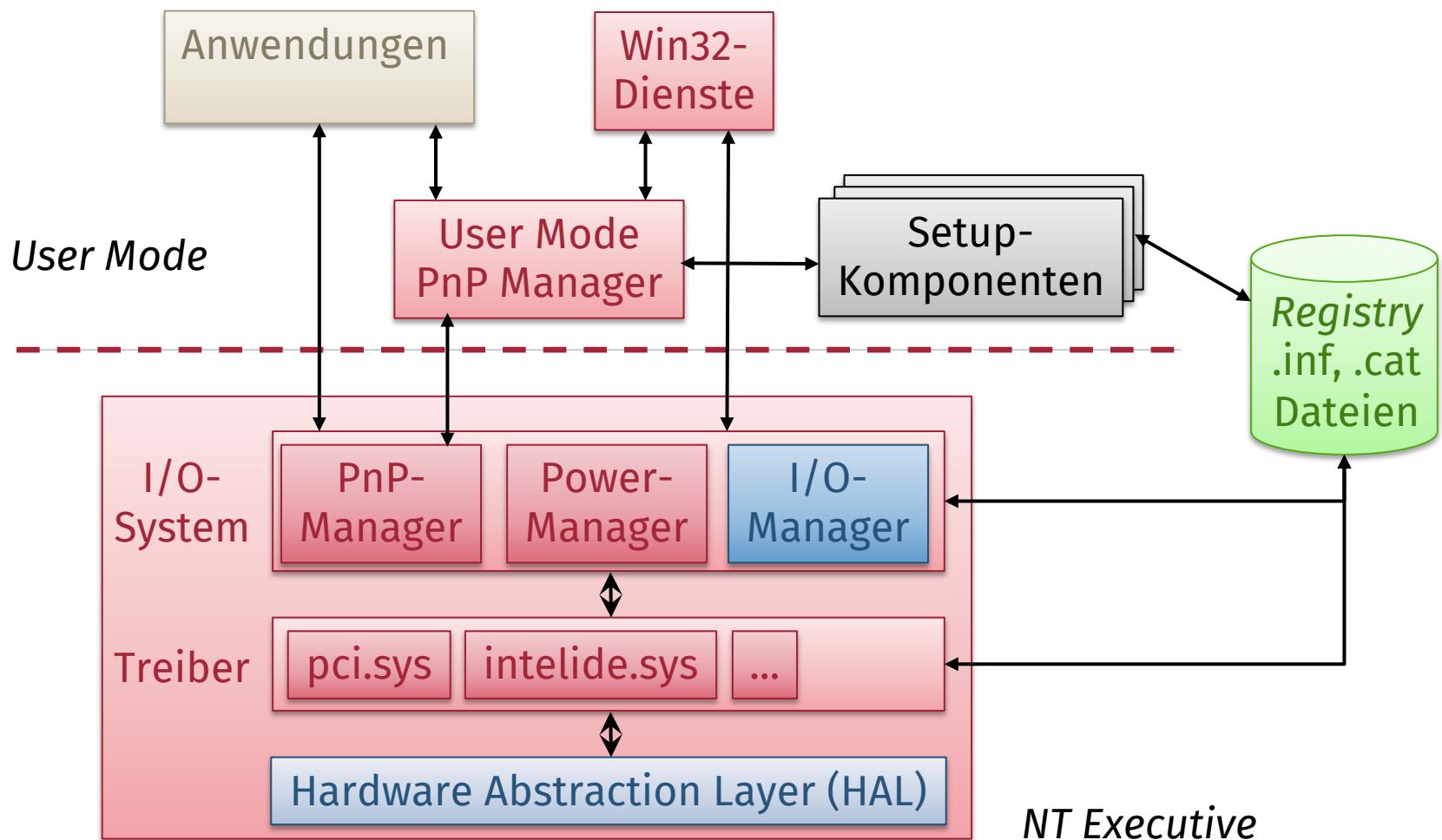
Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Fallstudie: Windows I/O-System



## Fallstudie: Windows I/O-System (2)

### I/O-Manager

- zentrale Komponente des Windows I/O-Systems
- Ein-, Ausgabeverwaltung

### Plug-and-Play (PnP)

- PnP-Manager erkennt selbständig neue Geräte
- fragt mit Hilfe seiner Systemanwendung nach einem Treiber

### Registry

- zentrale hierarchische Konfigurationsdatenbank von Windows

# Fallstudie: Windows I/O-System (3)

## Treiber

### ■ Kollektion von Dateien

- Binärkode des Treibers in .sys-Datei(en)
- Meta-Information über den Treiber in .inf-Datei
  - z.B. Gerätekasse, welche Dateien sind wohin zu kopieren
- kryptographische Signatur über alle Dateien in .cat-Datei

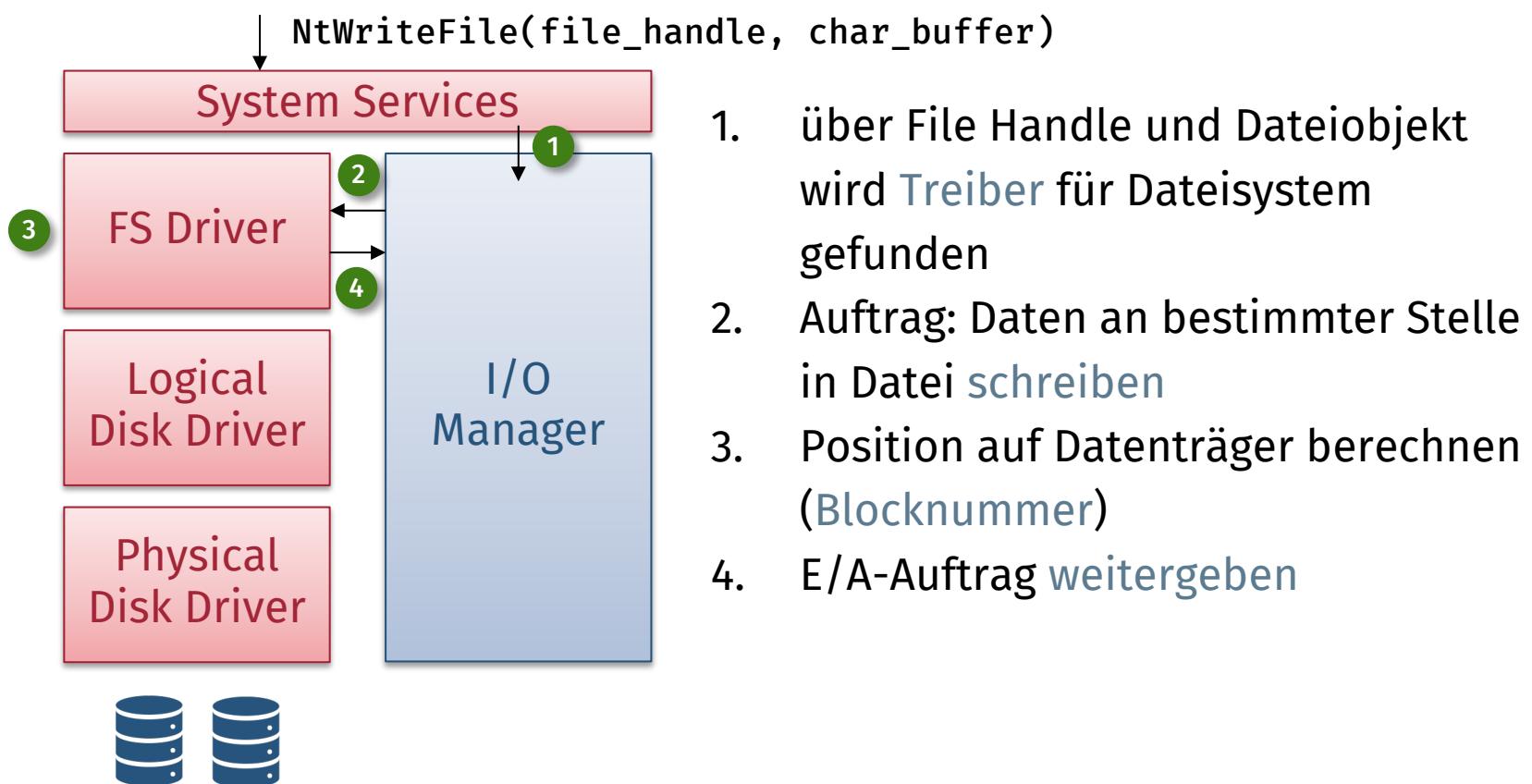
# Fallstudie: Windows I/O-System (4)

## Treiberfunktionen

- Initialisierungsroutine / Entladeroutine
  - wird nach/vor dem Laden/Entladen des Treibers ausgeführt
- Routine zum Hinzufügen von Geräten
  - PnP-Manager hat ein neues Gerät für den Treiber
- „Verteilerroutinen“
  - Öffnen, Schließen, Lesen, Schreiben
  - gerätespezifische Operationen
- Interrupt Service Routine (ISR)
  - wird von zentraler Interrupt-Verteilungsroutine aufgerufen
- E/A-Komplettierungs- und Abbruchroutine
  - liefert Informationen über den Ausgang von E/A-Aufträgen

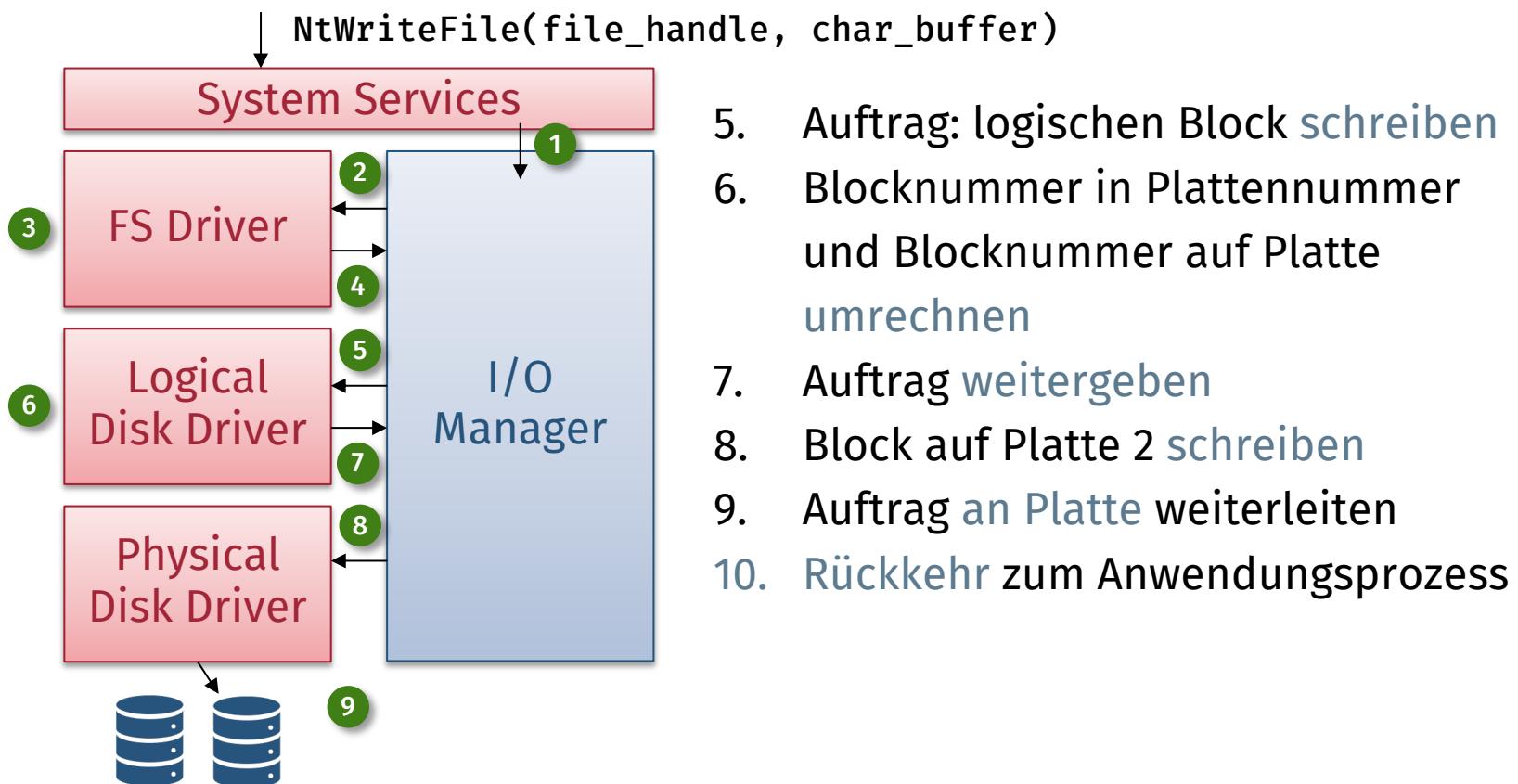
# Fallstudie: Windows I/O-System (5)

## Typischer Schreibaufgruf



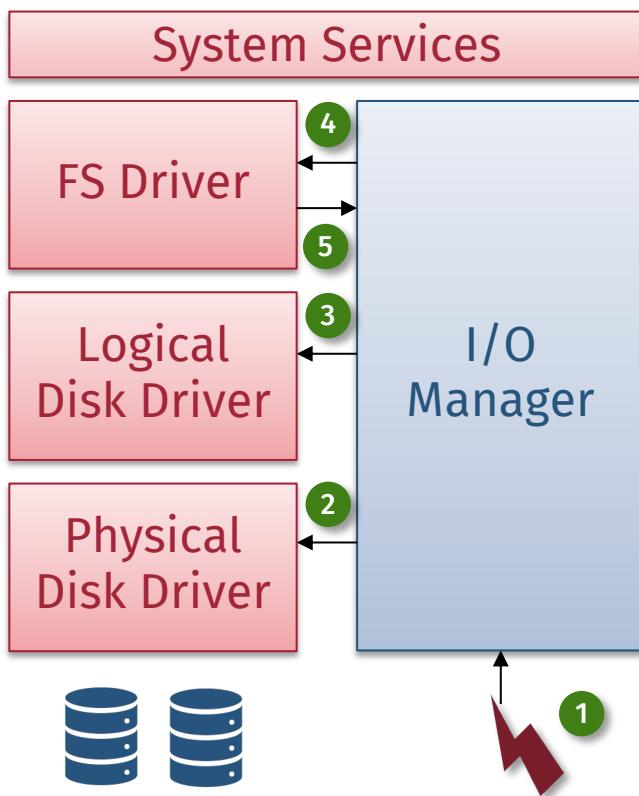
## Fallstudie: Windows I/O-System (6)

### Typischer Schreibauftrag (fortges.)



# Fallstudie: Windows I/O-System (7)

## Typische Komplettierung eines Schreibauftrags



1. Plattencontroller signalisiert per Unterbrechung den Abschluss der Operation
2. Aufruf der ISR
3. Aufruf der Komplettierungsroutine
4. Aufruf der Komplettierungsroutine weiterer (Teil-)Auftrag an den Datenträgertreiber

# Windows Plug-and-Play

## Hardware-Erkennung

- PnP-kompatible Geräte vom Hersteller eindeutig gekennzeichnet (Hardware IDs, Device Identification Strings)
  - Device ID
    - identifiziert exakt die aktuelle Hardware
    - identifiziert einen kompatiblen Treiber
  - Hardware IDs
    - alternative Hardwareidentifikation
    - optional
  - Compatible IDs
    - zeigen an, dass Gerät kompatibel mit anderen ist
    - anderer Treiber kann **notfalls** verwendet werden
    - optional

## Windows Plug-and-Play (2)

### Beispiel: neues USB-Gerät eingesteckt

- PnP-Manager erkennt neues USB-Gerät
  - Abfrage der Hardware IDs
  - standardisierte USB-Nachrichten
- Windows sucht nach passendem Treiber
  - lokal, evtl. auch über Windows Update
- in der Registry hinterlegte .inf-Dateien
  - werden nach IDs durchsucht, die am besten passen
    - Device ID passt: passender Treiber
    - Hardware ID, Compatibility ID passt: „brauchbarer“ Treiber
      - je weiter hinten in der Liste desto weniger brauchbar

# Windows Power Management

## Unterstützung des Advanced Configuration and Power Interface (ACPI)

- ACPI versetzt komplettes Rechensystem und Geräte in energiesparende Zustände
- System (Mainboard und BIOS) und Geräte müssen ACPI unterstützen
- sechs **Systemzustände** S0 bis S5
- vier **Gerätezustände** D0 bis D3
- Windows Power-Manager verwaltet Systemzustände
  - je nach Systemzustand werden Geräte in Gerätezustände versetzt
  - Power-Manager benachrichtigt Treiber

# Windows Power Management (2)

## ACPI-Systemzustände

- **S0 (Working):** Maschine ist voll verfügbar
- **S1 (Standby):** reduzierte Energieaufnahme
  - CPU angehalten, Kontext von CPU und Speichern bleibt erhalten
    - CPU und Speicher stehen noch unter Strom
  - Wiederanlauf in  $\approx 2\text{s}$
- **S2 (Standby):** weniger Stromaufnahme als S1
  - Kontext von CPU und Caches geht verloren und muss vorher gesichert werden
  - Wiederanlauf in  $> 2\text{s}$

# Windows Power Management (3)

## ACPI-Systemzustände (fortges.)

- **S3 (Suspend-to-RAM, Standby, Sleep):**
  - nur noch **Hauptspeicher** erhält Strom
  - CPU, Caches und (Teile des) Mainboard sind **ausgeschaltet**
- **S4 (Suspend-to-Disk, Hibernation):** System ist vollständig abgeschaltet
  - Speicherinhalte und Kontexte sind auf Platte **gesichert**
- **S5 (Off):** System ausgeschaltet, kein Strombedarf

# Windows Power Management (4)

## ACPI-Gerätezustände

- **D0 (Fully-On):** Gerät ist vollständig aktiv und einsatzbereit
- **D1:** reduzierte Stromaufnahme, schneller Übergang nach D0
  - Verhalten des Geräts herstellerabhängig
  - Treiber kann auf Gerät nicht mehr zugreifen
- **D2:** weiter reduzierte Stromaufnahme, langsamer Übergang nach D0
  - ansonsten wie D1
- **D3 (Off):** Gerät ist komplett von Stromversorgung getrennt
  - Betriebssystem muss Gerät neu initialisieren, wenn es wieder hochgefahren werden soll
- für viele Geräte sind D1 und D2 nicht definiert



Bild von Mike by Pexels

# (Grundlagen der) Betriebssysteme | H.7



Franz J. Hauck | Institut für Verteilte Systeme, Univ. Ulm

# Inhaltsüberblick

## Ein-, Ausgabe- und Geräteverwaltung

- Geräteverwaltung
  - Treiber und Treiberschnittstelle
- Treiberimplementierung
  - Betriebsarten, Scheduling
  - Plattentreiber, Treiber für serielle Schnittstellen, Netzwerktreiber
- Fallstudie UNIX/Linux
- Fallstudie Windows I/O-System
- Zeichensätze

# Zeichensätze

## Repräsentation von Texten

### ■ Naiver Ansatz

- Codierung A → 0, B → 1, ... und dann Binärcodierung

### ■ Probleme

- Welche Zeichen sollen codiert werden?
- Wie kann man Daten/Texte mit anderen austauschen?

### ■ Lösung

- Standardisierte Zeichensätze

# ASCII

## American Standard Code for Information Interchange

- 1963 von der American Standards Organization
- 7-Bit Code
- Für die USA gedacht
- Neben Zeichen auch Steuercodes
  - z.B. CR (Carriage Return) für Wagenrücklauf bei Druckern/Terminals
  - z.B. NL (New Line) für neue Zeile bei Druckern/Terminals
  - z.B. BEL für Glocke

# ASCII (2)

## Zeichensatz-Tabelle

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

## ASCII (3)

### Problem

- Viele wichtige Zeichen fehlen
- Keine Unterstützung für andere Sprachen als Englisch

# ISO 8859

## Standard der International Organization for Standardization (ISO)

### ■ 8-Bit Code

- zur Zeit 15 Zeichensätze
- Codes  $00_{16}$  bis  $7F_{16}$  jeweils identisch mit ASCII
- Codes  $80_{16}$  bis  $9F_{16}$  nicht definiert (andere Standards)

### ■ ISO-8859-1 / Latin 1

- westeuropäischer Zeichensatz

### ■ ISO-8859-15 / Latin 9

- erneuerter westeuropäischer Zeichensatz z.B. mit €-Symbol

### ■ Deutsche Buchstaben in allen Latin-Zeichensätzen enthalten

- ISO-8859-1/-2/-3/-4/-9/-10/-13/-14/-15/-16
- mit der gleichen Codierung

# ISO 8859 (2)

## Zeichensatz-Tabelle für Latin-1 / ISO 8859-1

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
8...	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9...	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
A...	NBSP	ı	ç	£	¤	¥	ı	§	..	©	¤	«	¬	SHY	®	-
B...	º	±	²	³	‘	µ	¶	·	,	¹	º	»	¼	½	¾	¿
C...	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D...	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E...	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F...	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

- NBSP = Non-Breakable Space, SHY = Soft Hyphen

## ISO 8859 (2)

### Zeichensatz-Tabelle für Latin-9 / ISO 8859-15

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
8...	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9...	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
A...	NBSP	ı	ç	£	€	¥	Š	§	š	©	¤	«	¬	SHY	®	-
B...	º	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
C...	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D...	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E...	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F...	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

- NBSP = Non-Breakable Space, SHY = Soft Hyphen

# Unicode

## System zur Erfassung und Darstellung aller Schriftzeichen

- Verwaltet vom Unicode-Konsortium <http://www.unicode.org>
- 17 Planes à 65.536 Zeichen (**U+0000 bis U+10FFFF**)
  - insgesamt 1.114.112 Zeichen
    - durch Kombinationen sogar noch mehr
  - die meisten Planes noch undefiniert
- Basic Multilingual Plane (BMP, 0)
  - grundlegender Codebereich mit allen aktuell gebräuchlichen Schriftsysteme, Satzzeichen und Symbole
  - **U+0000 bis U+007F** identisch zu ASCII
  - **U+0080 bis U+00FF** identisch zu Latin-1

## Unicode (2)

### System zur Erfassung und Darstellung aller Schriftzeichen

- Unicode wird kontinuierlich um neue Zeichen ergänzt
- Unicode vereinbart auch weitere Informationen
  - z.B. Schreibrichtung, Normalisierung mehrdeutiger Darstellungen

## Unicode (3)

### Codierungsformate

- UTF = Unicode Transformation Format
- UTF-8
  - 8-Bit Code, aber eventuell mehrere Codes für ein Zeichen nötig
  - U+0000 bis U+007F identische Codierung wie ASCII: 1 Byte lang
  - U+0080 bis U+00FF selben Zeichen wie Latin-1 aber 2 Byte lang
  - bis zu 4 Byte lange Codes für Unicode-Zeichen
  - Einsatz
    - z.B. Linux/Windows Standardcodierung

# Unicode (4)

## Codierungsformate

### ■ UTF-8 (fortges.)

- Bildung der Codes aus dem Unicode-Wort

Unicode	UTF-8
U+000000 – U+00007F	0xxxxxxx
U+000080 – U+0007FF	110xxxxx, 10xxxxxx
U+000800 – U+00FFFF	1110xxxx, 10xxxxxx, 10xxxxxx
U+010000 – U+10FFFF	11110xxx, 10xxxxxx, 10xxxxxx, 10xxxxxx

- Beispiele

- A = U+000041 = 41<sub>16</sub>
- ä = U+0000E4 = C3 A4<sub>16</sub>
- € = U+0020AC = E2 82 AC<sub>16</sub>

# Unicode (5)

## Codierungsformate

### ■ UTF-16

- 16-Bit Code, aber eventuell mehrere Codes für ein Zeichen
- U+0000 bis U+00FF selben Zeichen wie Latin-1 **aber 2 Byte lang**
- bis zu 4 Byte lange Codes für Unicode-Zeichen
- Einsatz
  - z.B. Java, interne Stringcodierung

# Unicode (6)

## Codierungsformate

### ■ UTF-16 (fortges.)

- Bildung der Codes aus dem Unicode-Wort

Unicode	UTF-16
U+000000 – U+00FFFF	xxxxxxxx xxxxxxxx
U+010000 – U+10FFFF	110110xx xxxxxxxx, 110111xx xxxxxxxx mit Codewert vorher um <b>10000<sub>16</sub></b> reduziert

- ◆ Hinweis: Zeichen U+D800 bis U+DBFF (High Surrogates) und U+DC00 bis U+DFFF (Low-Surrogates) sind freigehalten

- Beispiele

- A = U+000041 = 00 41<sub>16</sub>
- ä = U+0000E4 = 00 E4<sub>16</sub>
- 眇 = U+024F5C = D8 53 DF 5C<sub>16</sub>

# Unicode (7)

## Codierungsformate

### ■ UTF-32

- 32-Bit Code
- U+0000 bis U+00FF selben Zeichen wie Latin-1 aber 4 Byte lang
- Nachteil: hoher Speicherbedarf
- Beispiele
  - A = 00 00 00 41<sub>16</sub>
  - ä = 00 00 00 E4<sub>16</sub>
  - 眇 = 00 02 4F 5C<sub>16</sub>

## Unicode (8)

### Besondere Eigenschaft der Codierungsformate

- Wahlfreier Zugriff z.B. in einer Textdatei
  - immer erkennbar, ob dort ein Zeichen anfängt oder nicht
- UTF-32
  - trivial: jedes 32-Bit Wort enthält ein Zeichen
- UTF-16
  - 16-Bit Wort mit Muster **110111xxxxxxxxxx** zeigt zweite Hälfte an, d.h. vorheriges 16-Bit Wort ist Beginn des codierten Zeichens
- UTF-8
  - Byte mit Muster **10xxxxxx** zeigt zweites, drittes oder viertes Byte eines codierten Zeichens an, d.h. erste vorherige Byte, das dieses Muster nicht zeigt, ist Beginn des codierten Zeichens

# Was ist hier passiert?



# ISO 646 Variante für Deutschland

## Zeichensatztabelle (Differenz zu ASCII)

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	§	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ü	^	-
6...	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	ä	ö	ü	ß	DEL

◆ Es gibt noch viele ziemlich alte Zeichensätze (Legacy)!