

---

# Betriebssysteme (BS)

## 10. Eingabe und Ausgabe

<https://sys.cs.tu-dortmund.de/de/lehre/ss25/bs>

---

23.06.2025

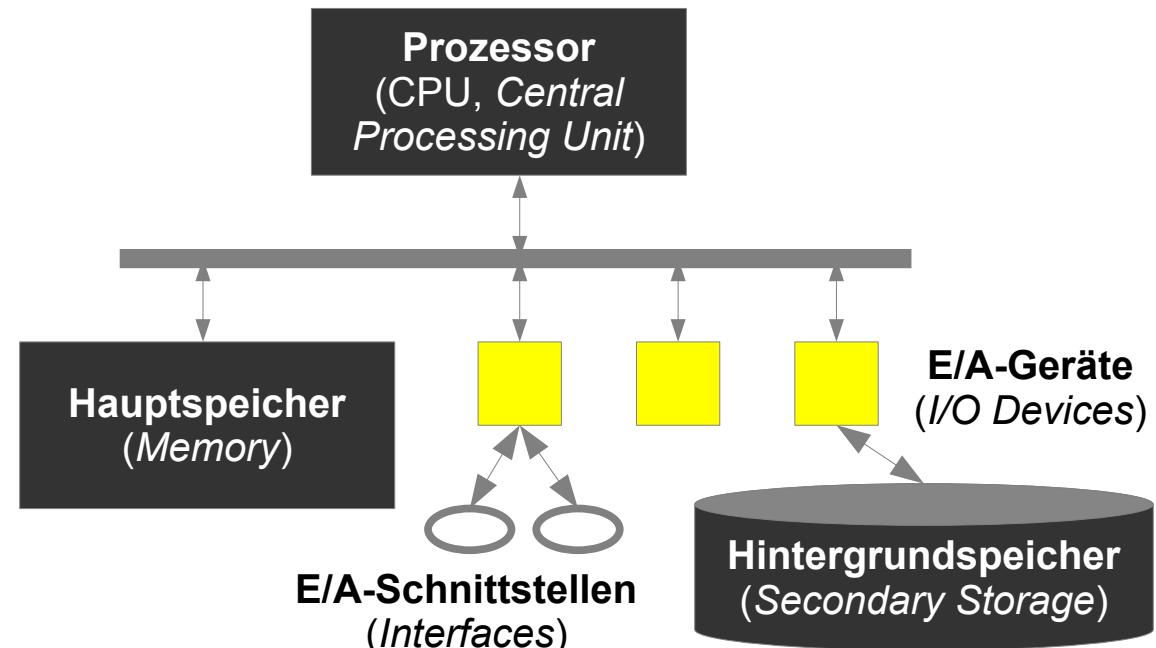
**Peter Ulbrich**

[peter.ulbrich@tu-dortmund.de](mailto:peter.ulbrich@tu-dortmund.de)  
[bs-problems@ls12.cs.tu-dortmund.de](mailto:bs-problems@ls12.cs.tu-dortmund.de)  
<https://sys.cs.tu-dortmund.de/de/lehre/kummerkasten>

In Teilen basierend auf *Betriebssysteme* von Olaf Spinczyk, Universität Osnabrück

# Wiederholung: Betriebsmittel

- Das Betriebssystem hat folgende Aufgaben:
  - Verwaltung der Betriebsmittel des Rechners
  - Schaffung von Abstraktionen, die Anwendungen einen einfachen und effizienten Umgang mit Betriebsmitteln erlauben
- Bisher:
  - Prozesse
  - Arbeitsspeicher
- Heute: E/A-Geräte
  - Verwaltung von Peripheriegeräten



# Inhalt

- Ein-/Ausgabe-Hardware
- Gerätaprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

# Inhalt

- Ein-/Ausgabe-Hardware
- Gerätaprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

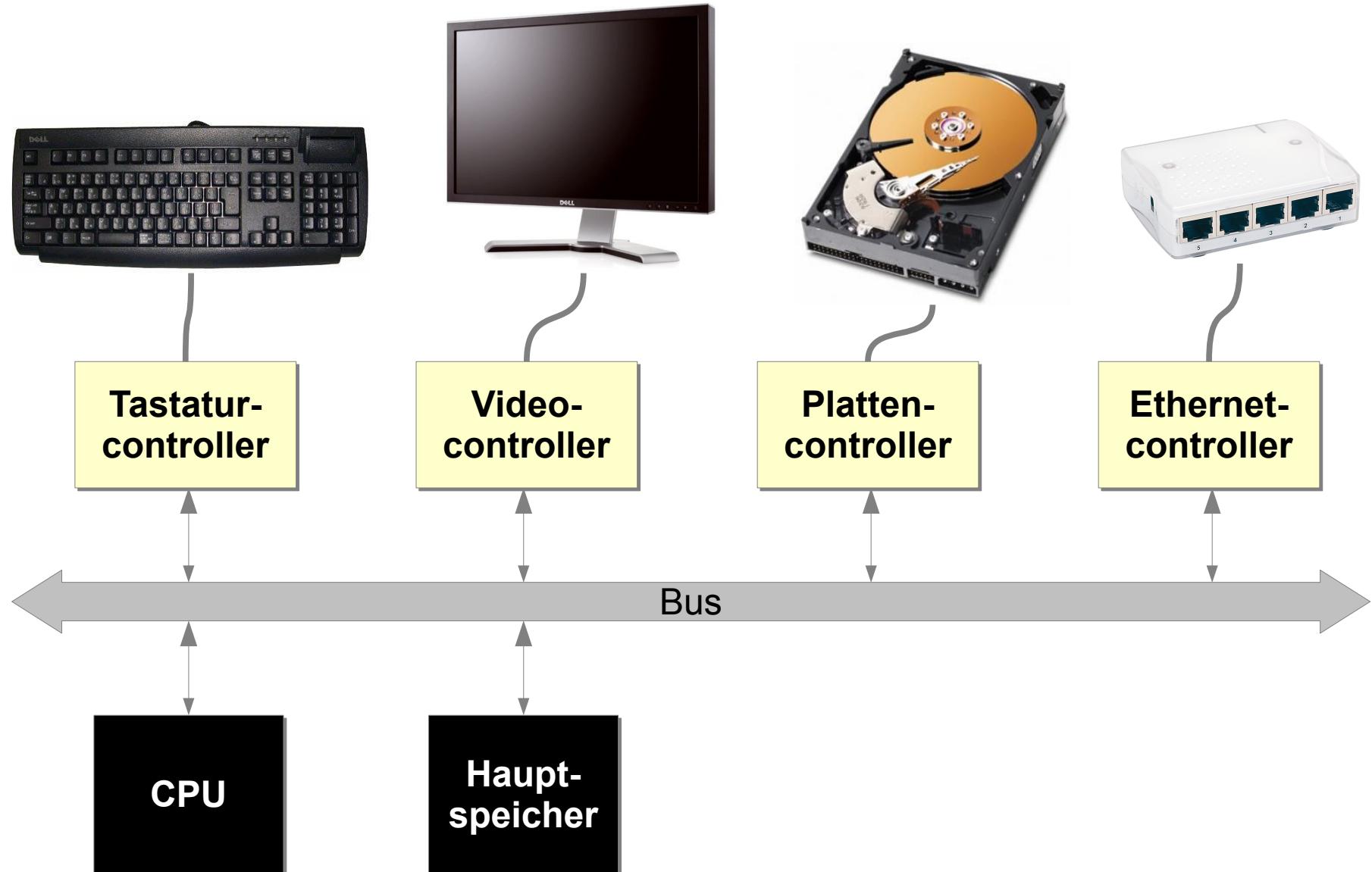
*Tanenbaum*  
5: Ein-Ausgabe  
*Silberschatz*  
13: I/O-Systems

# Inhalt

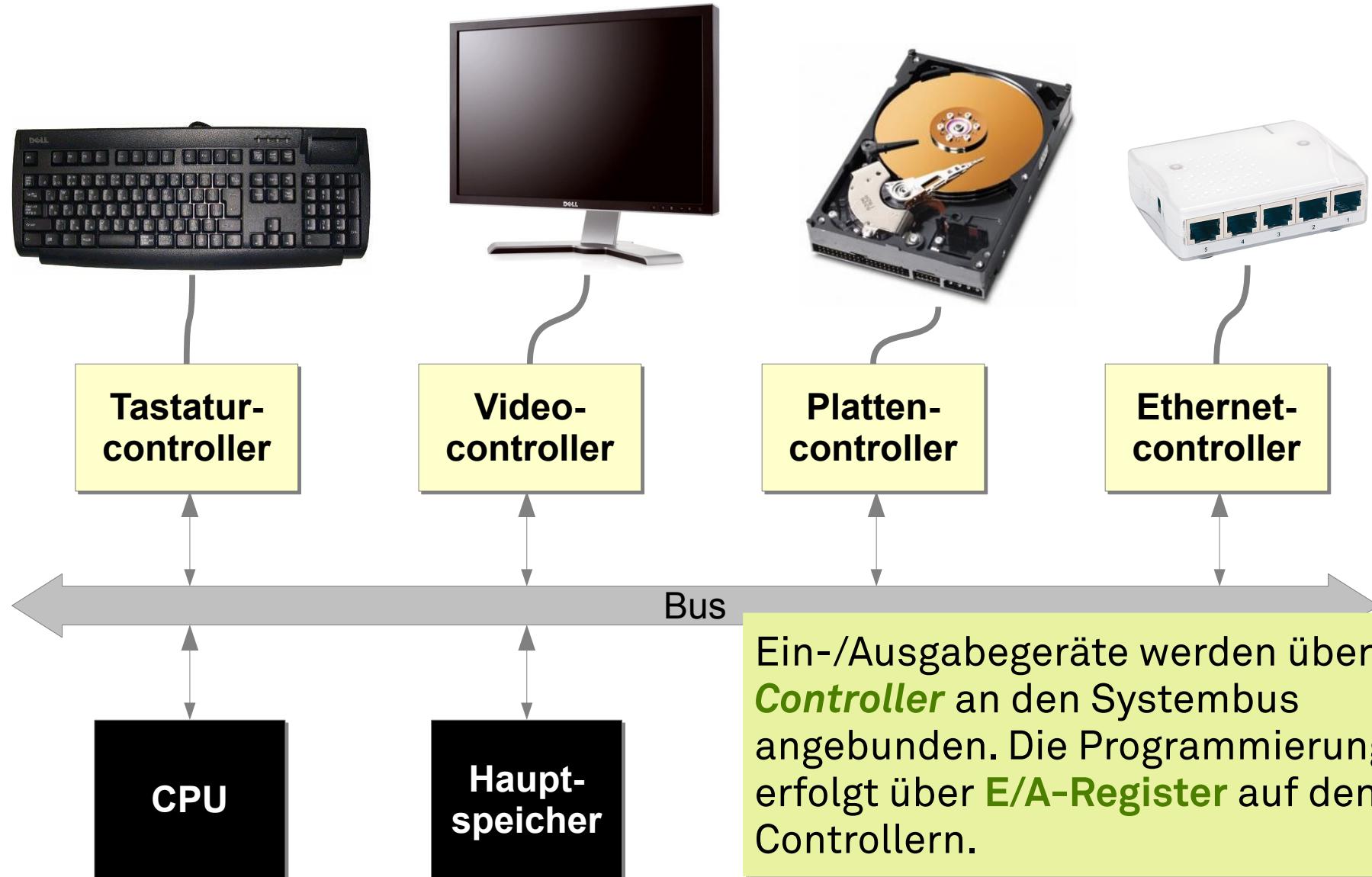
- Ein-/Ausgabe-Hardware
- Gerätaprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

*Tanenbaum*  
5: Ein-Ausgabe  
*Silberschatz*  
13: I/O-Systems

# Anbindung von E/A-Geräten

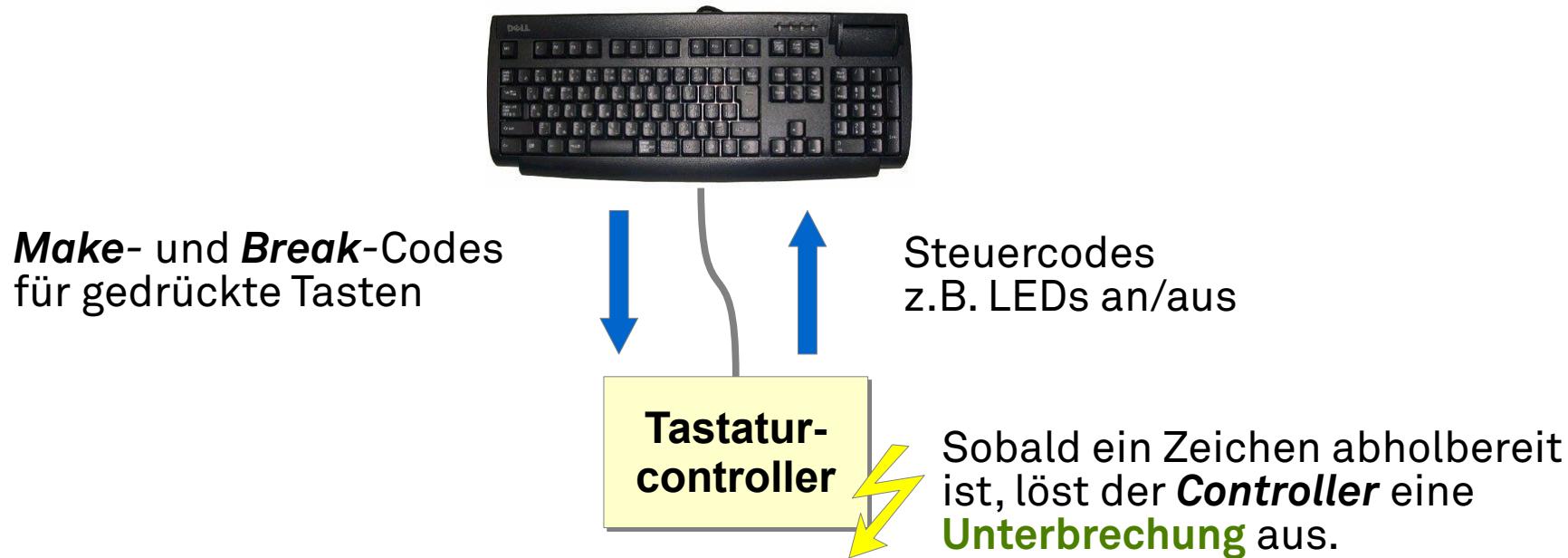


# Anbindung von E/A-Geräten



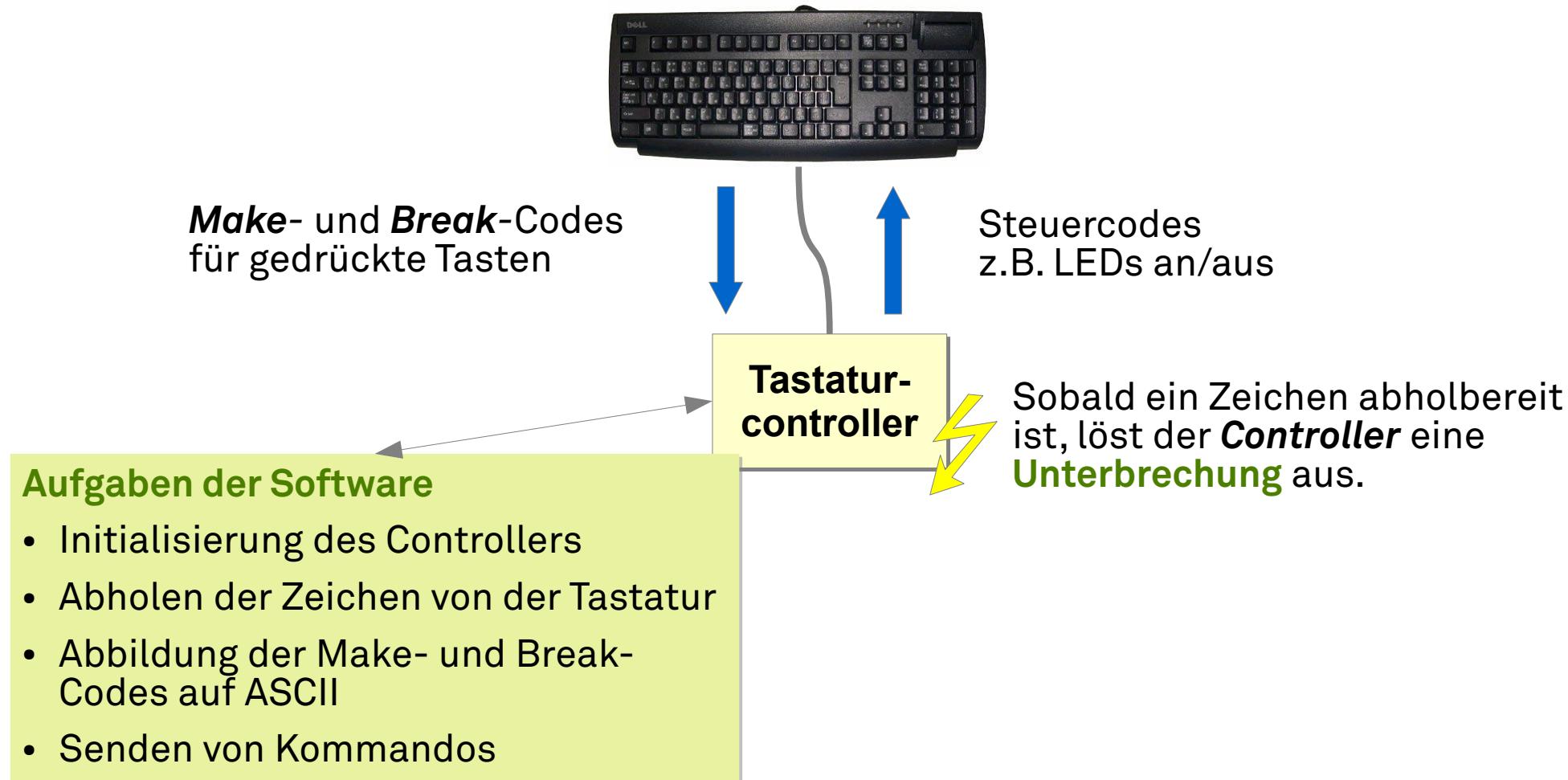
## Beispiel: PC-Tastatur

- serielle zeichenweise Kommunikation
  - Tastatur ist „intelligent“ (besitzt eigenen Prozessor)



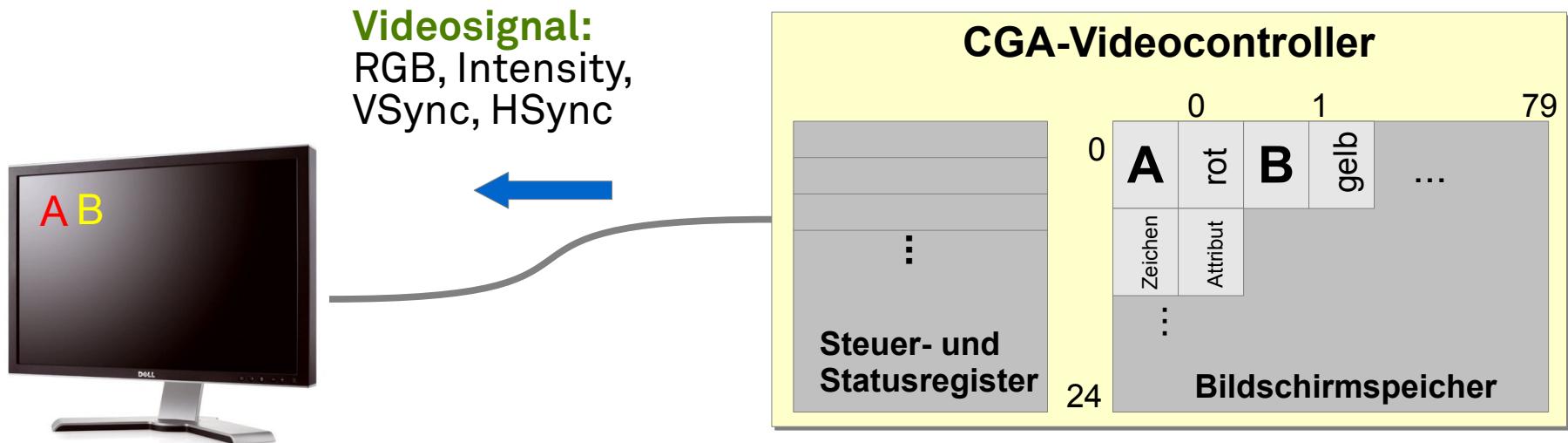
# Beispiel: PC-Tastatur

- serielle zeichenweise Kommunikation
  - Tastatur ist „intelligent“ (besitzt eigenen Prozessor)



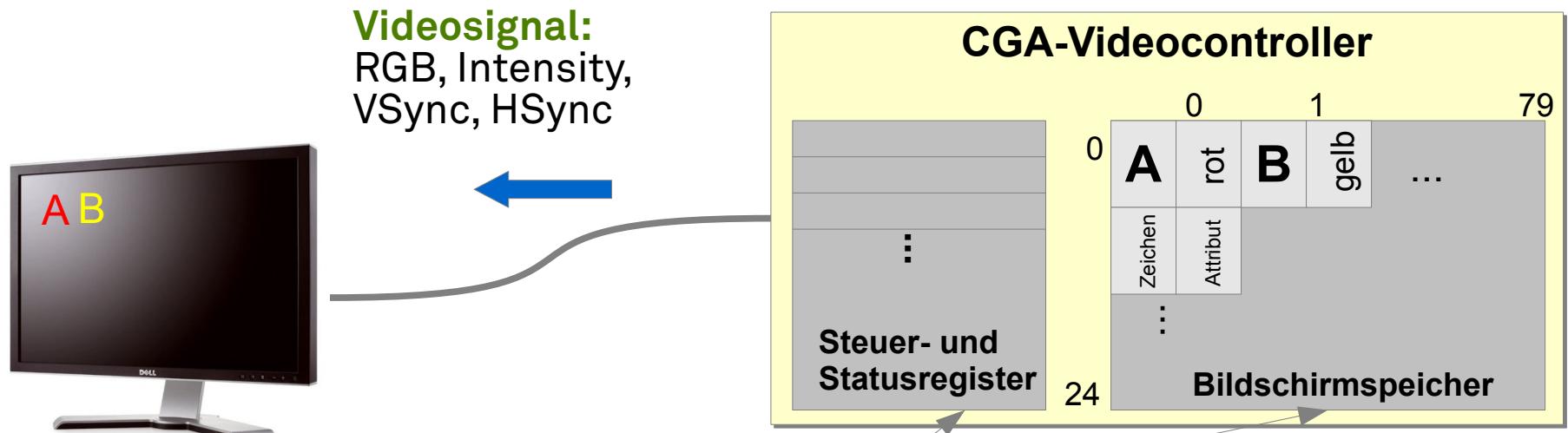
# Beispiel: CGA-Videocontroller

- Kommunikation über Videosignal
  - Umwandlung des Bildschirmspeicherinhalts in ein Bild (80x25 Z.)



# Beispiel: CGA-Videocontroller

- Kommunikation über Videosignal
  - Umwandlung des Bildschirmspeicherinhalts in ein Bild (80x25 Z.)

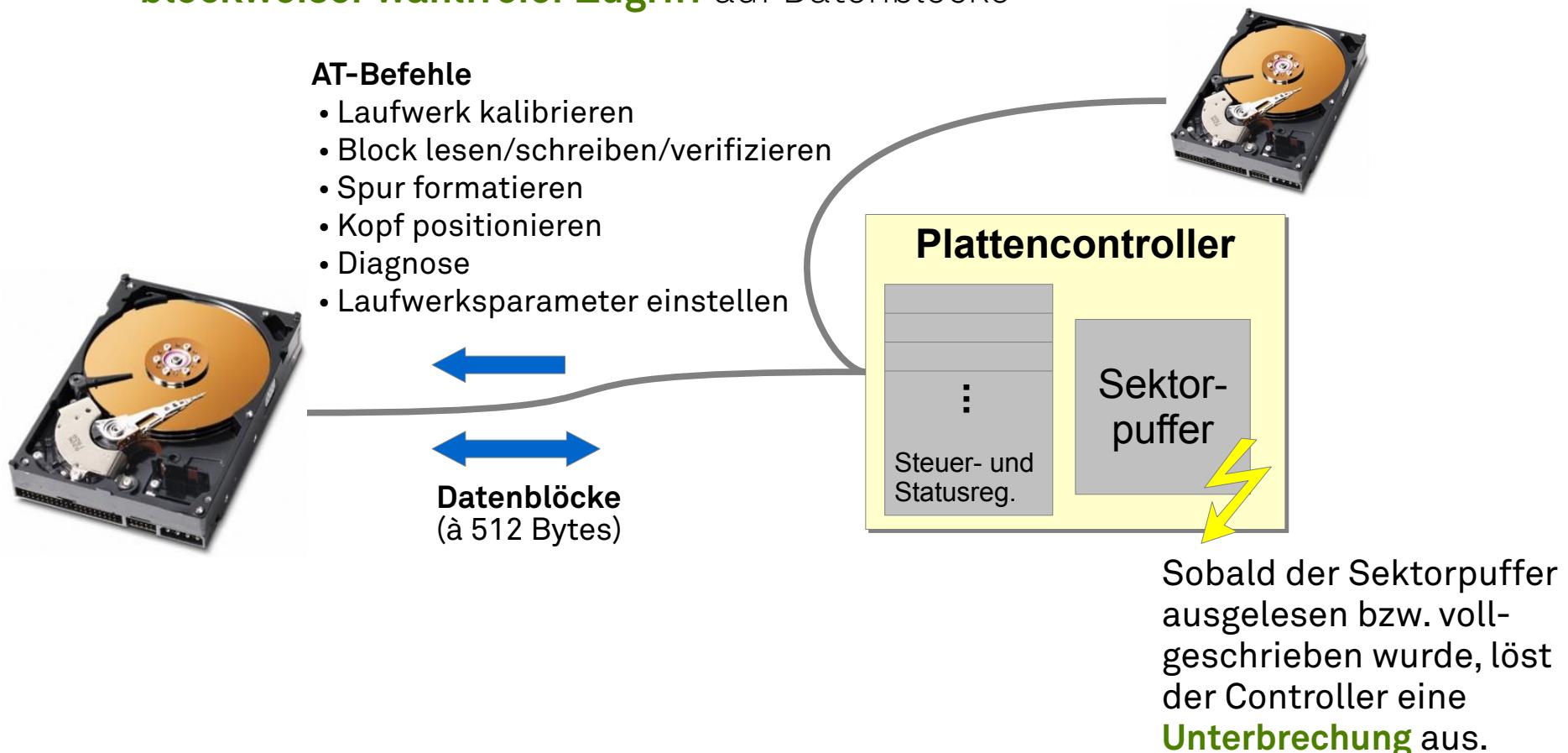


## Aufgaben der Software

- Initialisierung des Controllers
- Bildschirmspeicher mit den gewünschten Zeichencodes füllen
- Steuerung der Position des Cursors
- Cursor an- und abschalten

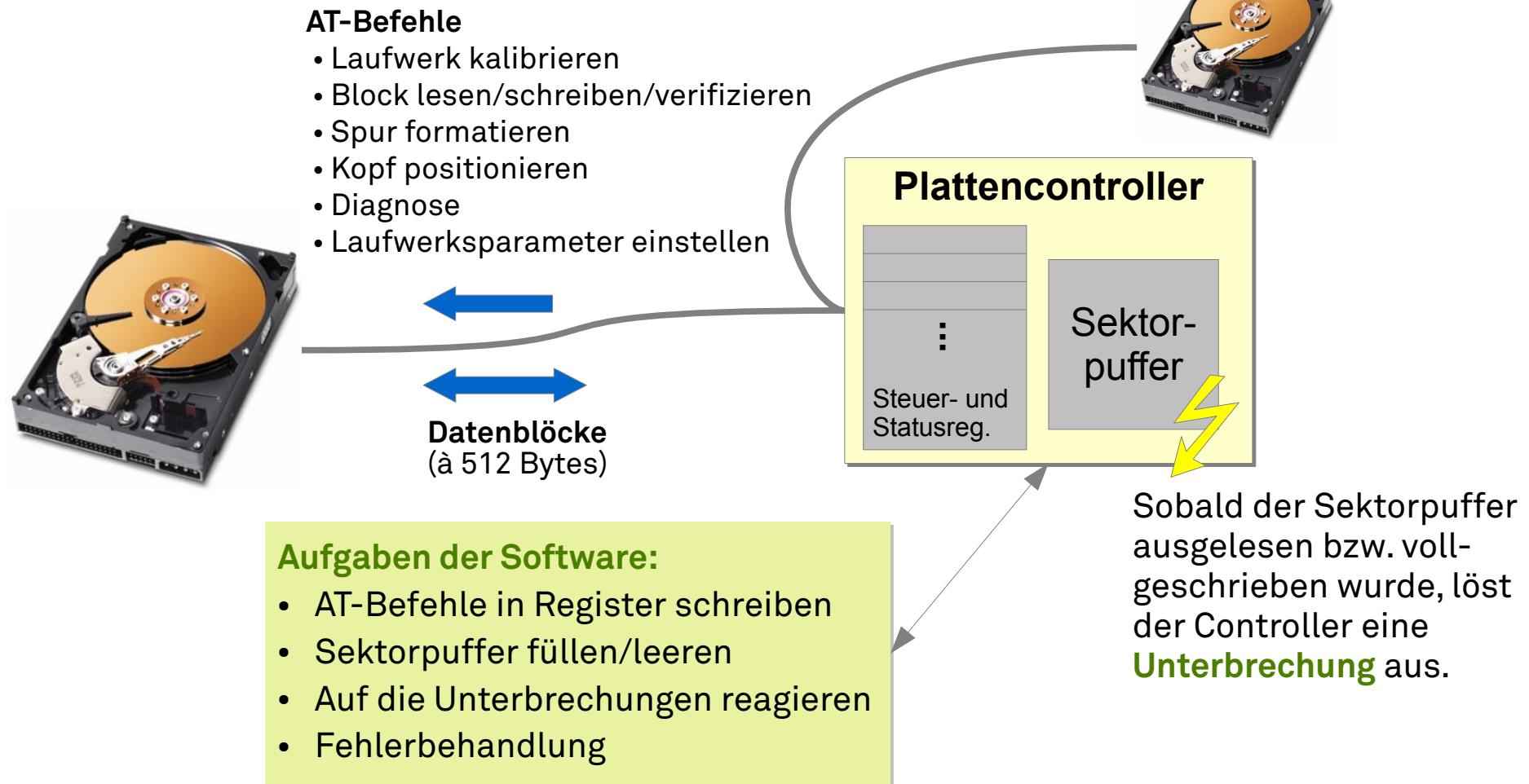
# Beispiel: IDE-Plattencontroller

- Kommunikation über AT-Befehle
  - **blockweiser wahlfreier Zugriff** auf Datenblöcke



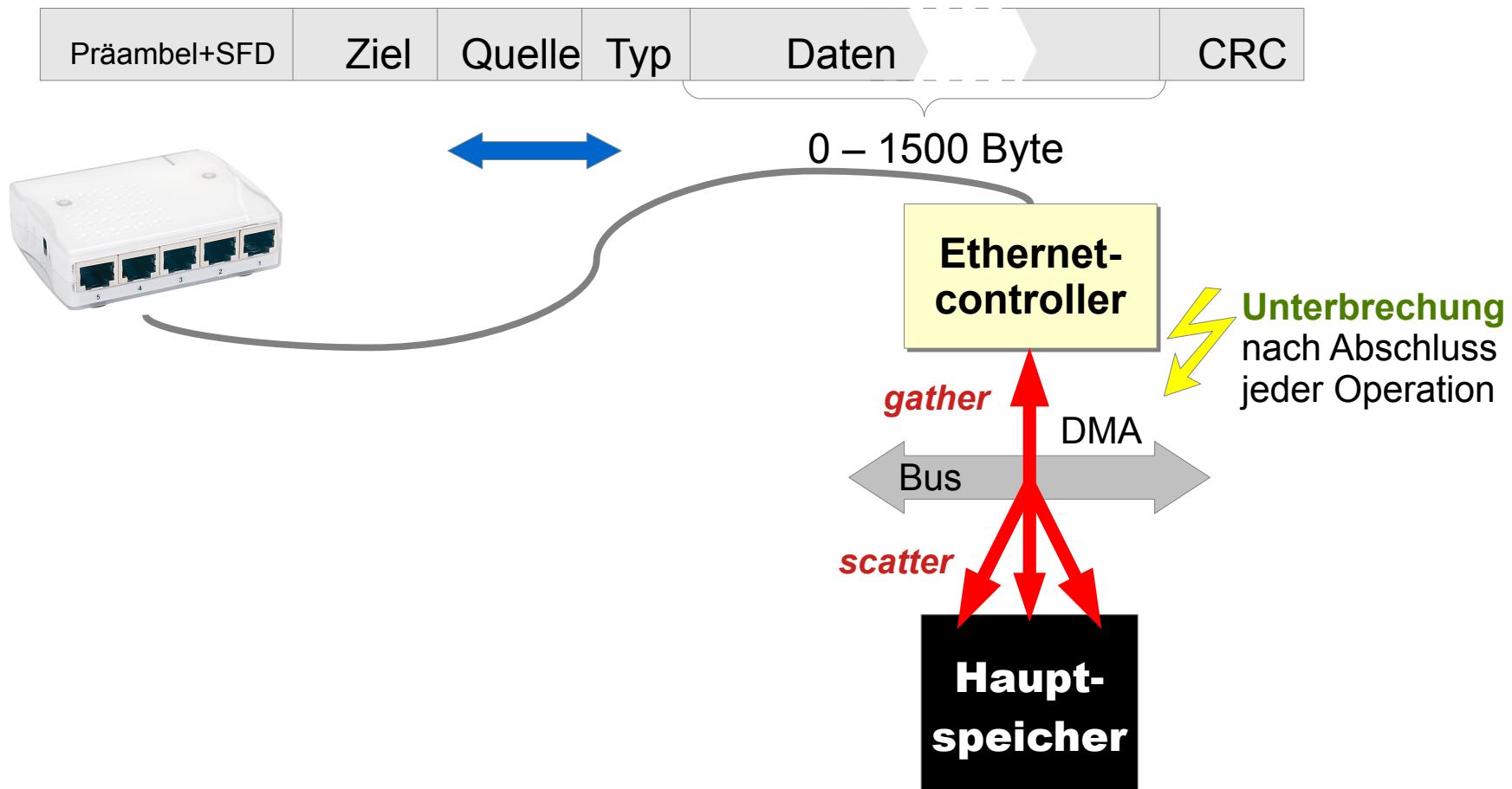
# Beispiel: IDE-Plattencontroller

- Kommunikation über AT-Befehle
  - **blockweiser wahlfreier Zugriff** auf Datenblöcke



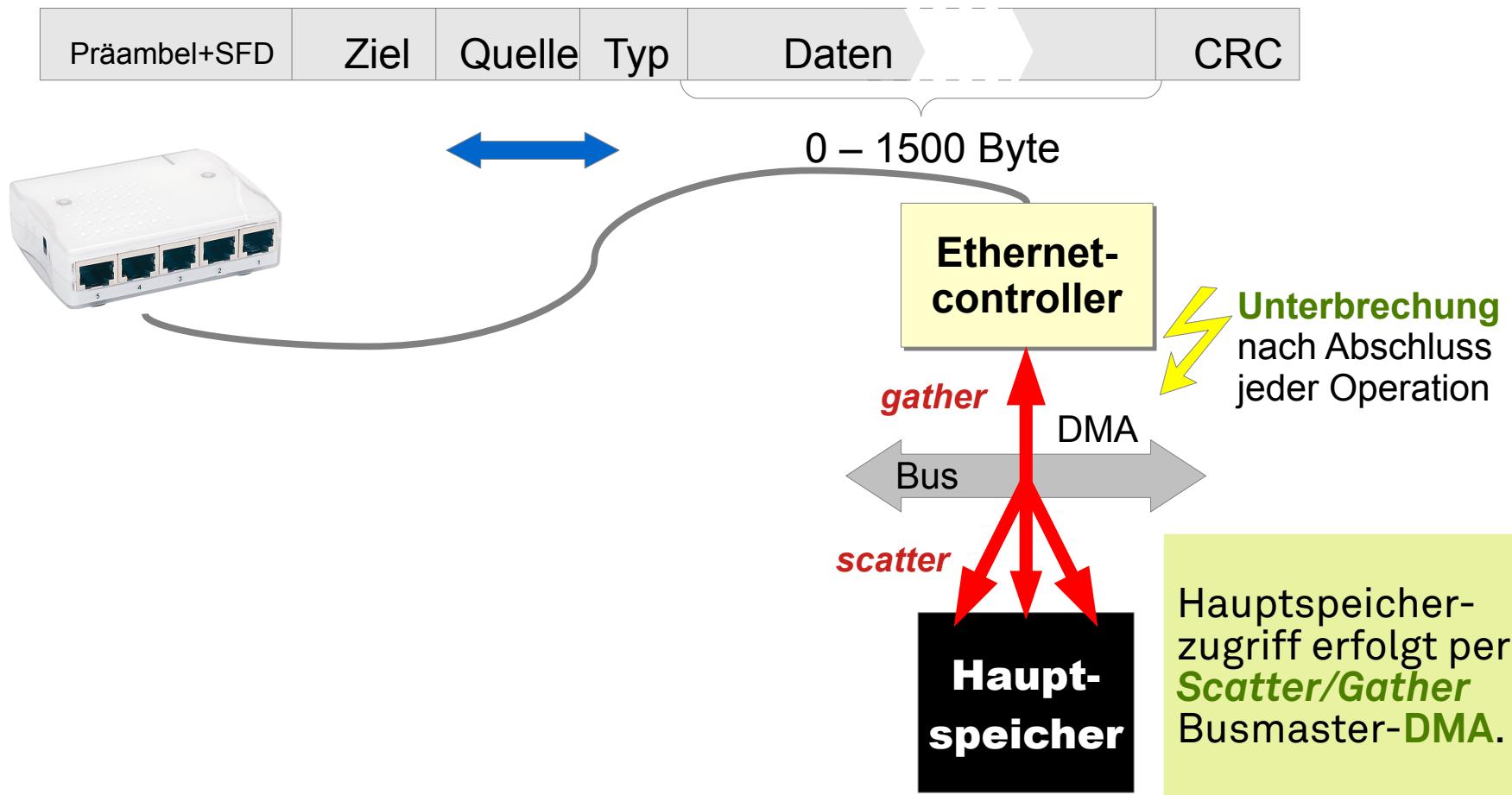
# Beispiel: Ethernet-Controller

- serielle paketbasierte Buskommunikation
  - Pakete haben eine variable Größe und enthalten Adressen



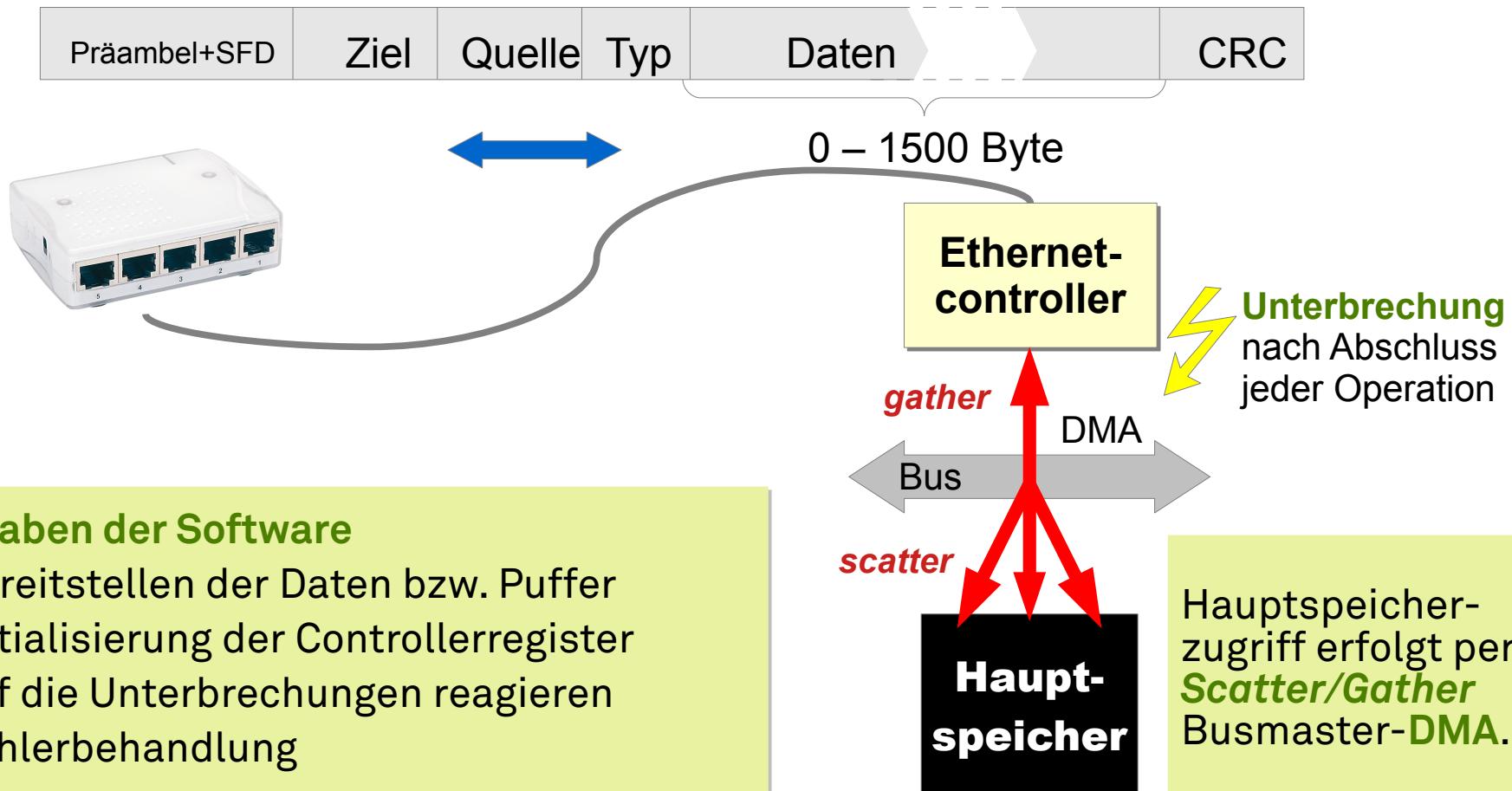
# Beispiel: Ethernet-Controller

- serielle paketbasierte Buskommunikation
  - Pakete haben eine variable Größe und enthalten Adressen



# Beispiel: Ethernet-Controller

- serielle paketbasierte Buskommunikation
  - Pakete haben eine variable Größe und enthalten Adressen



# Geräteklassen

## ■ **zeichenorientierte Geräte**

- Tastatur, Drucker, Modem, Maus, ...
- meist rein **sequentieller Zugriff**, selten wahlfreie Positionierung

## ■ **blockorientierte Geräte**

- Festplatte, Diskette, CD-ROM, DVD, Bandlaufwerke, ...
- meist **wahlfreier blockweiser Zugriff** (random access)

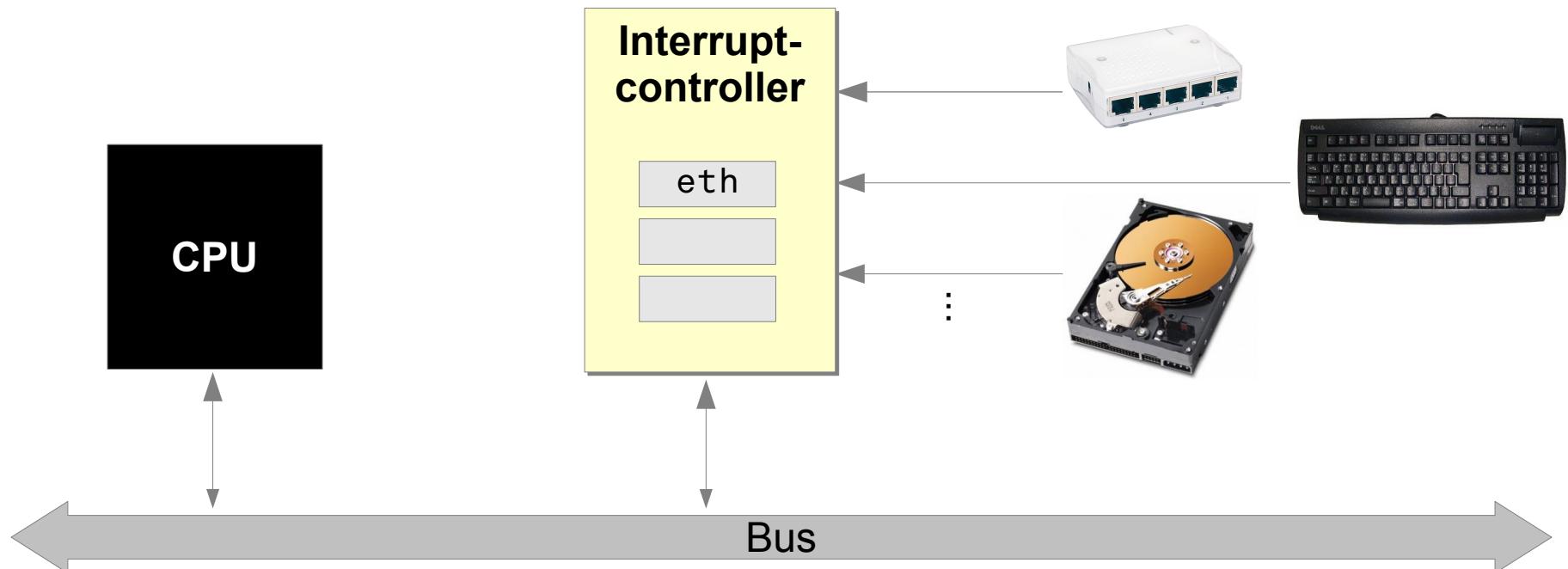
## ■ **Andere Geräte passen weniger gut in dieses Schema:**

- Grafikkarten (insbesondere 3D-Beschleunigung)
- Netzwerkkarten (Protokolle, Adressierung, Broadcast/Multicast, Nachrichtenfilterung, ...)
- Zeitgeberbaustein (einmalige oder periodische Unterbrechungen)
- ...

# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

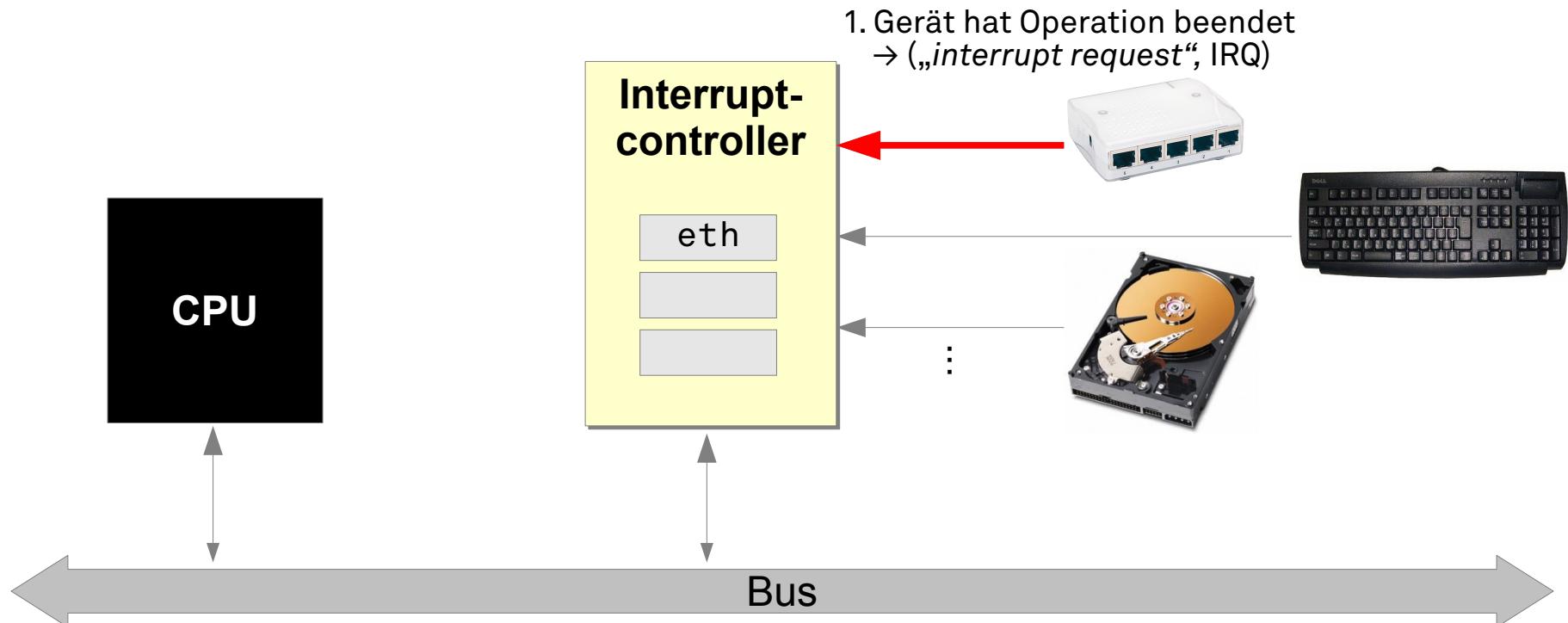
## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene



# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

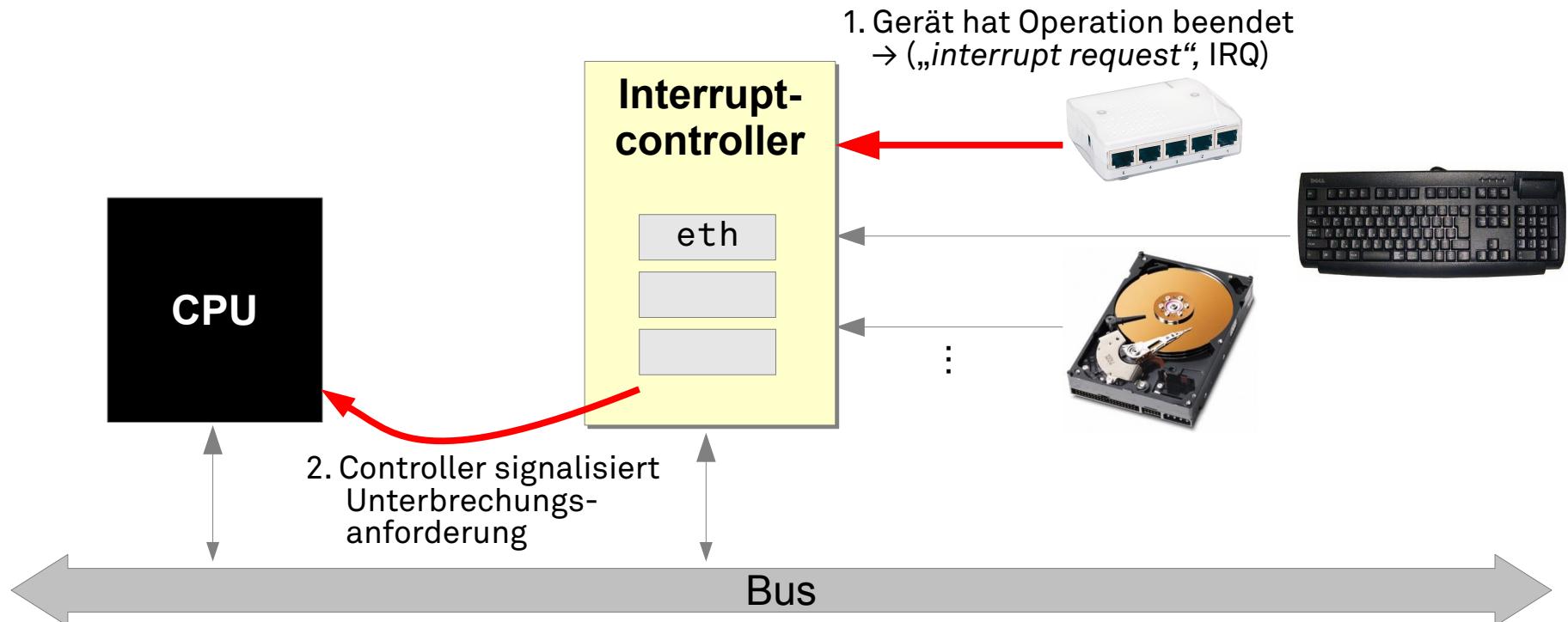
## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene



# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

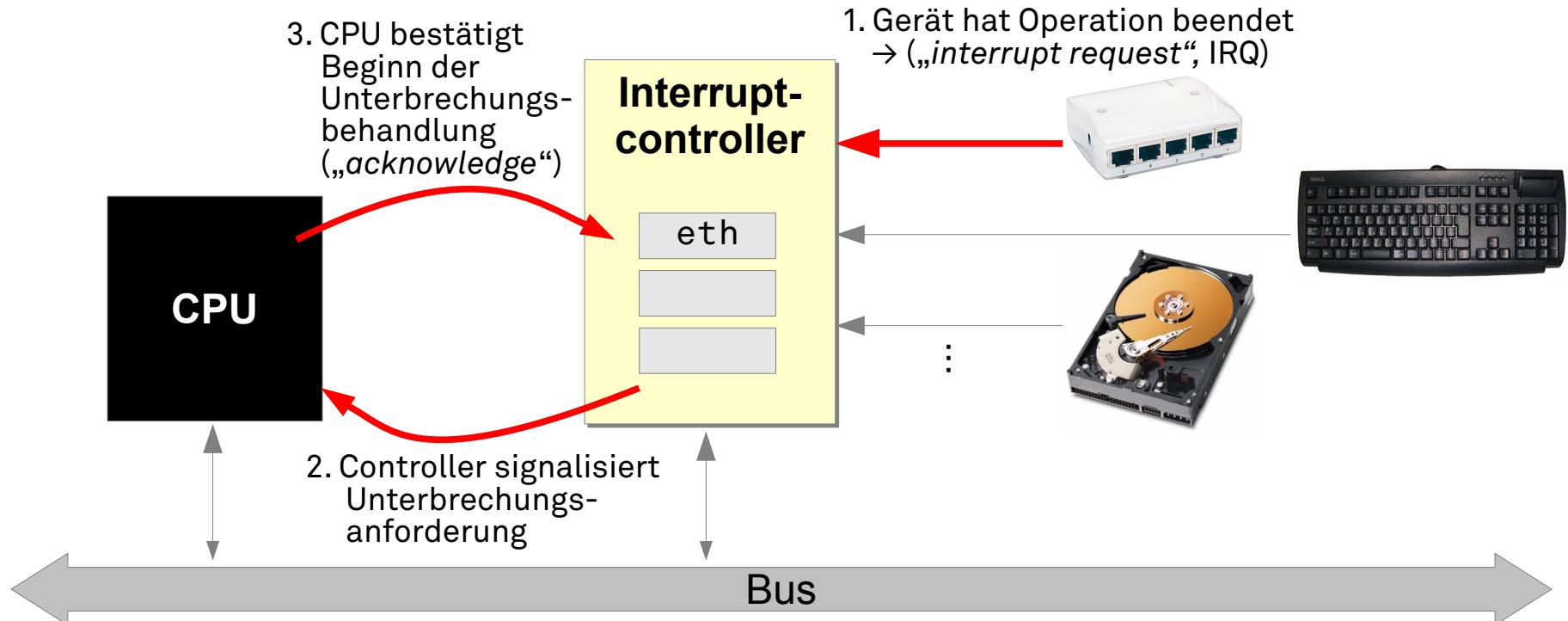
## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene



# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

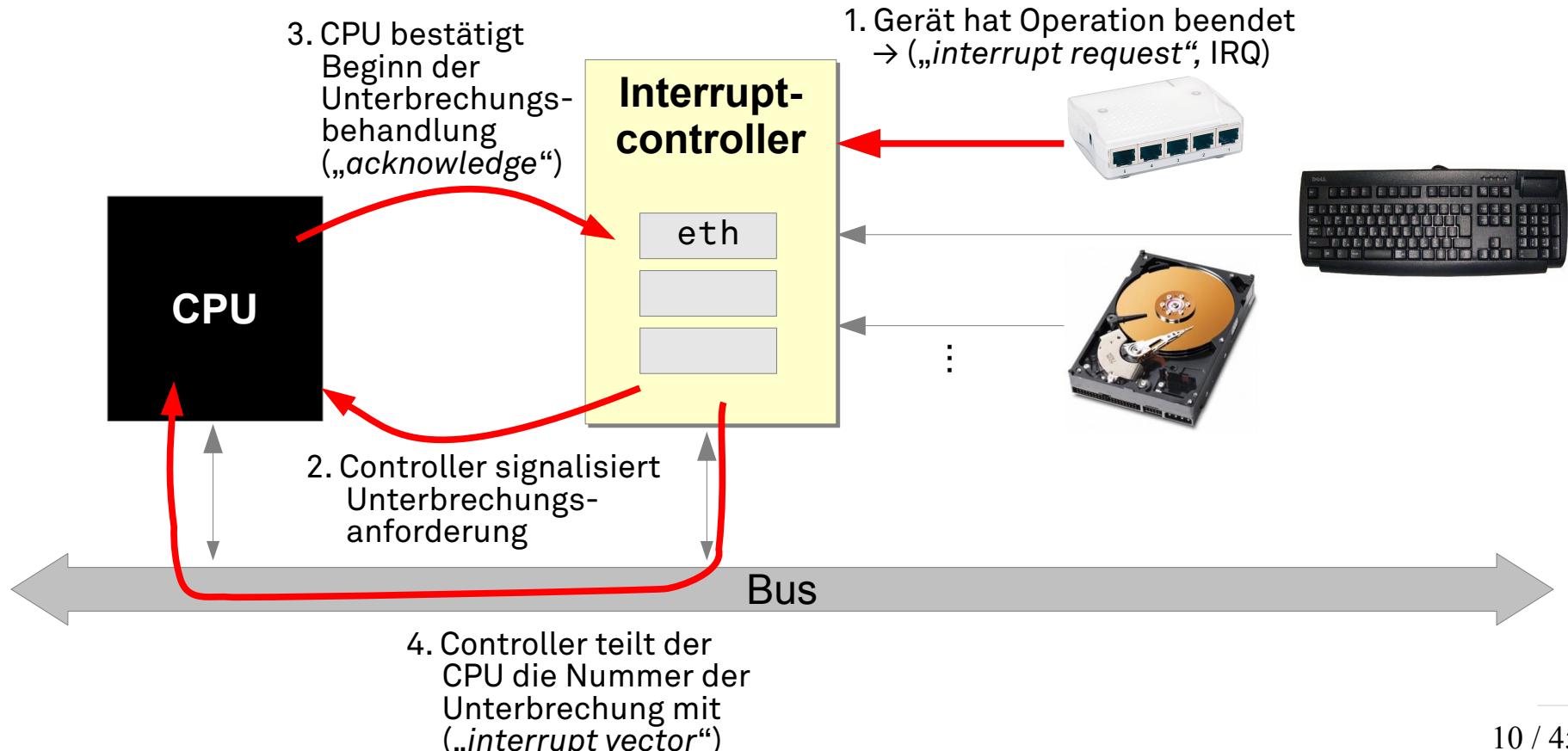
## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene



# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene

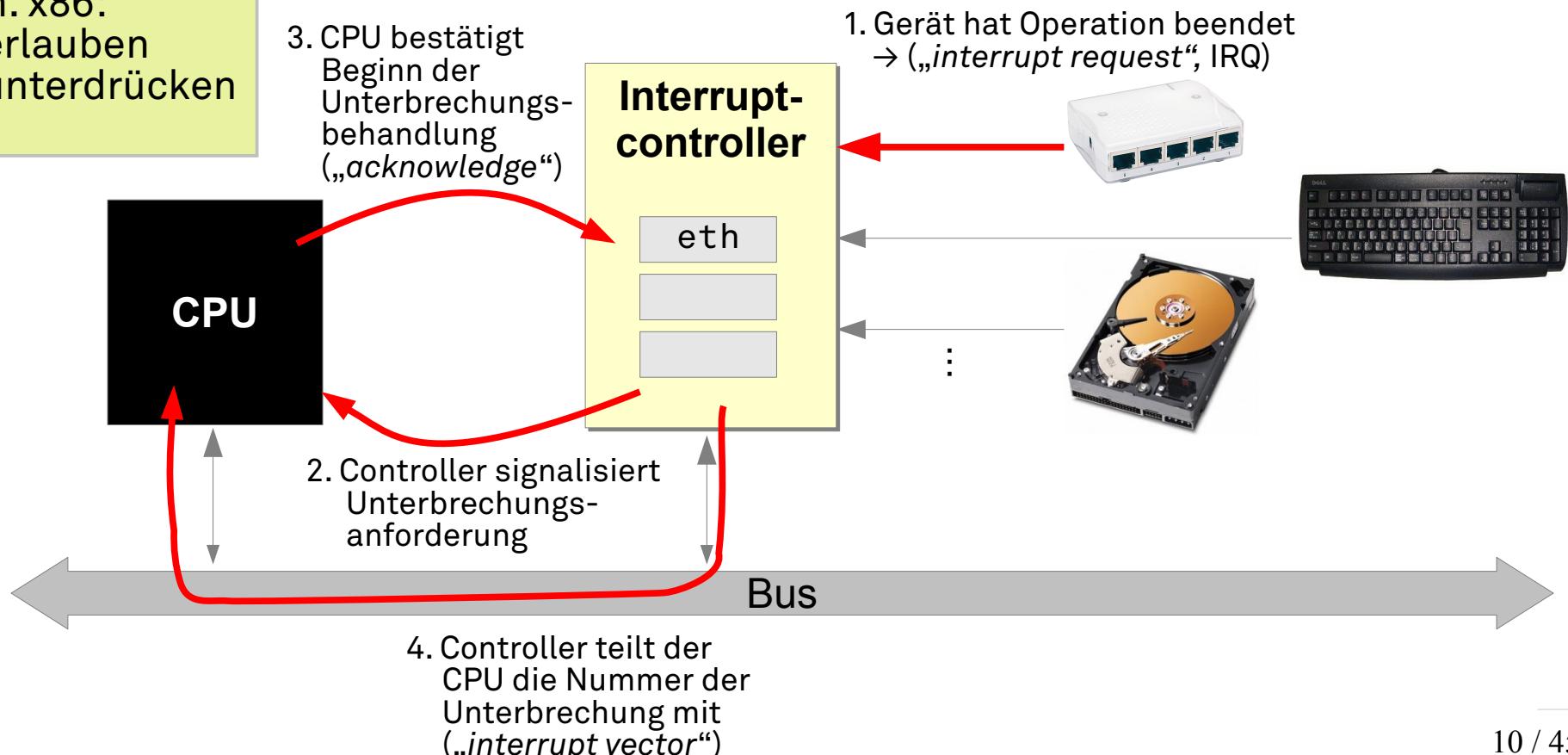


# Unterbrechungen ...

- signalisieren, dass die Software aktiv werden muss

Software kann IRQ-  
Behandlung unter-  
drücken. x86:  
**sti** → erlauben  
**cli** → unterdrücken

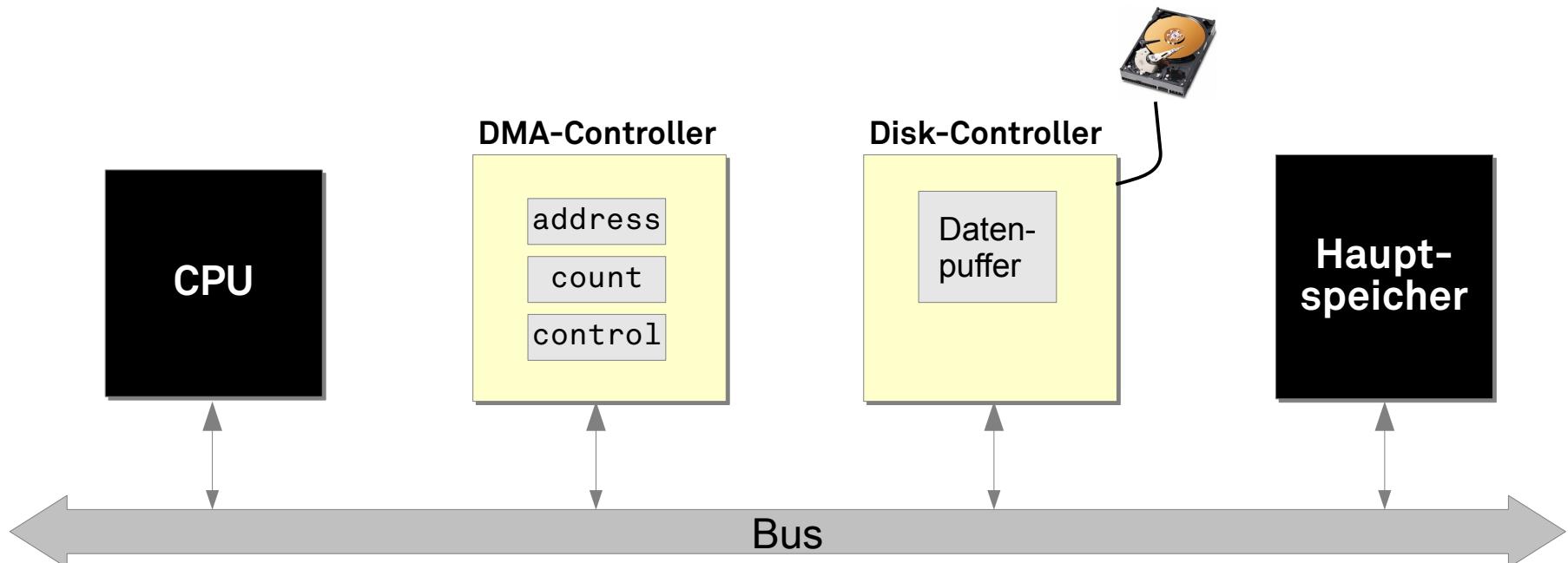
## Ablauf einer Unterbrechungsbehandlung auf der Hardwareebene



## **Direct Memory Access (DMA) ...**

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

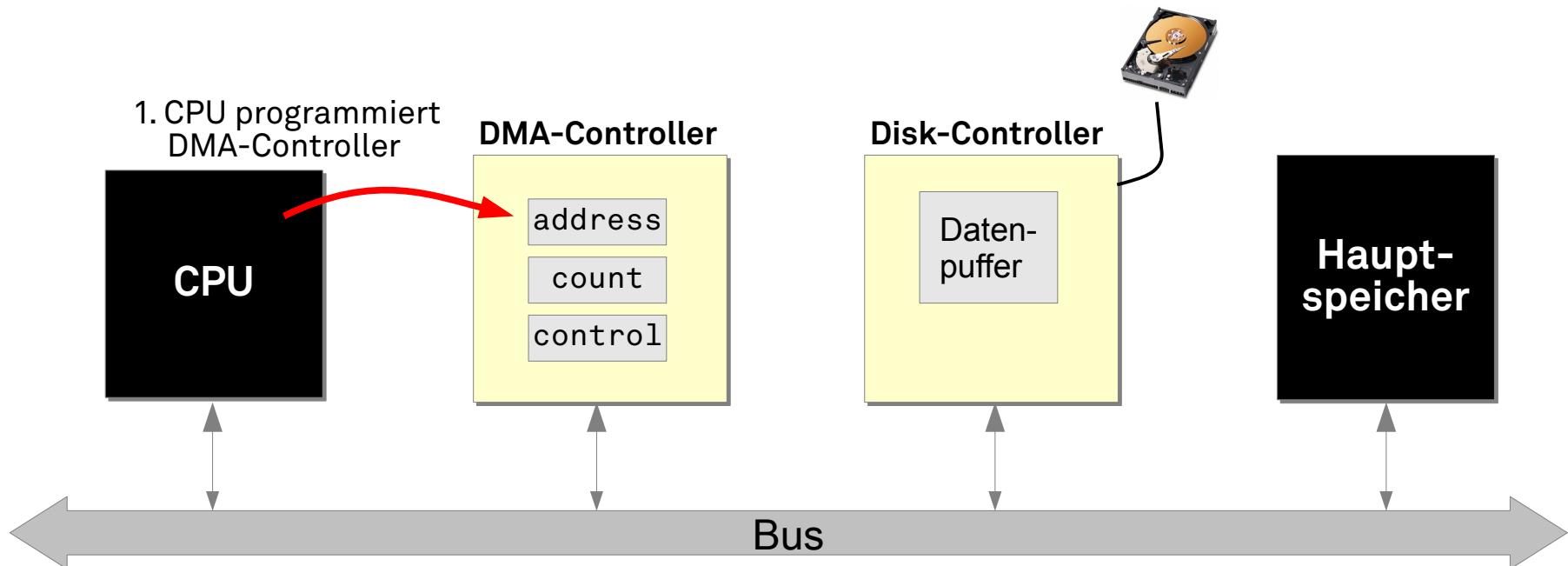
### **Durchführung eines DMA-Transfers**



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

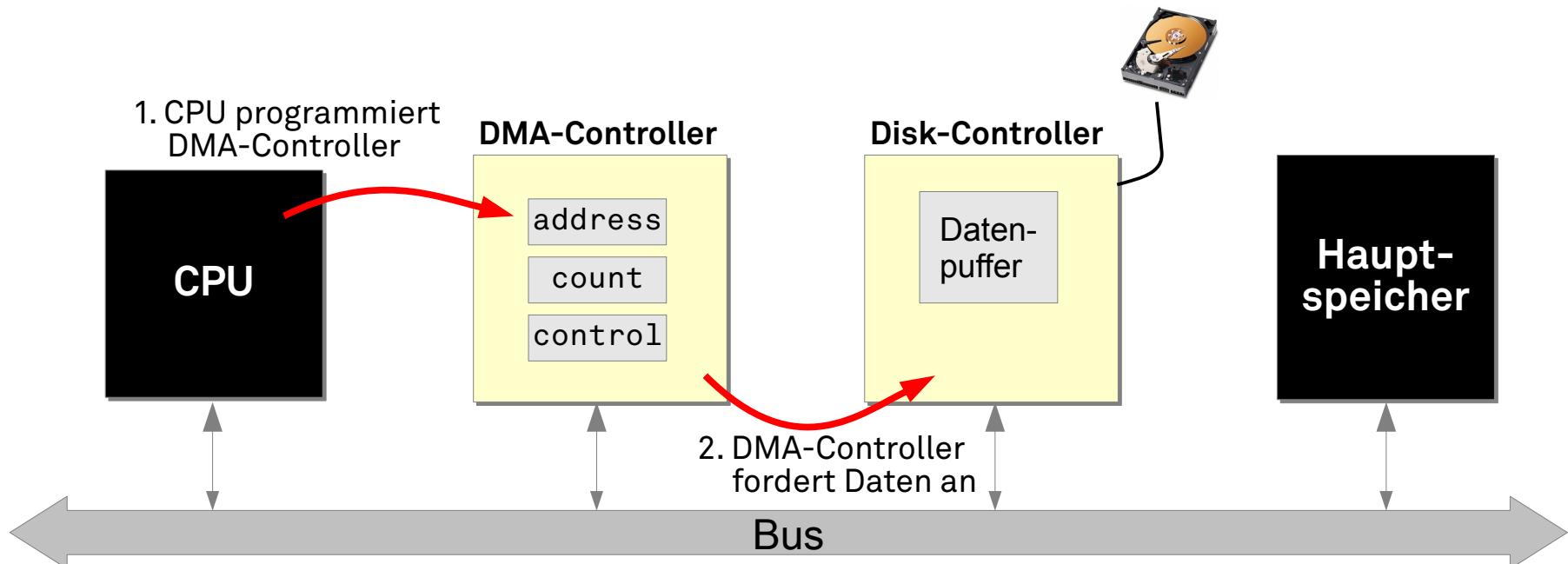
### Durchführung eines DMA-Transfers



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

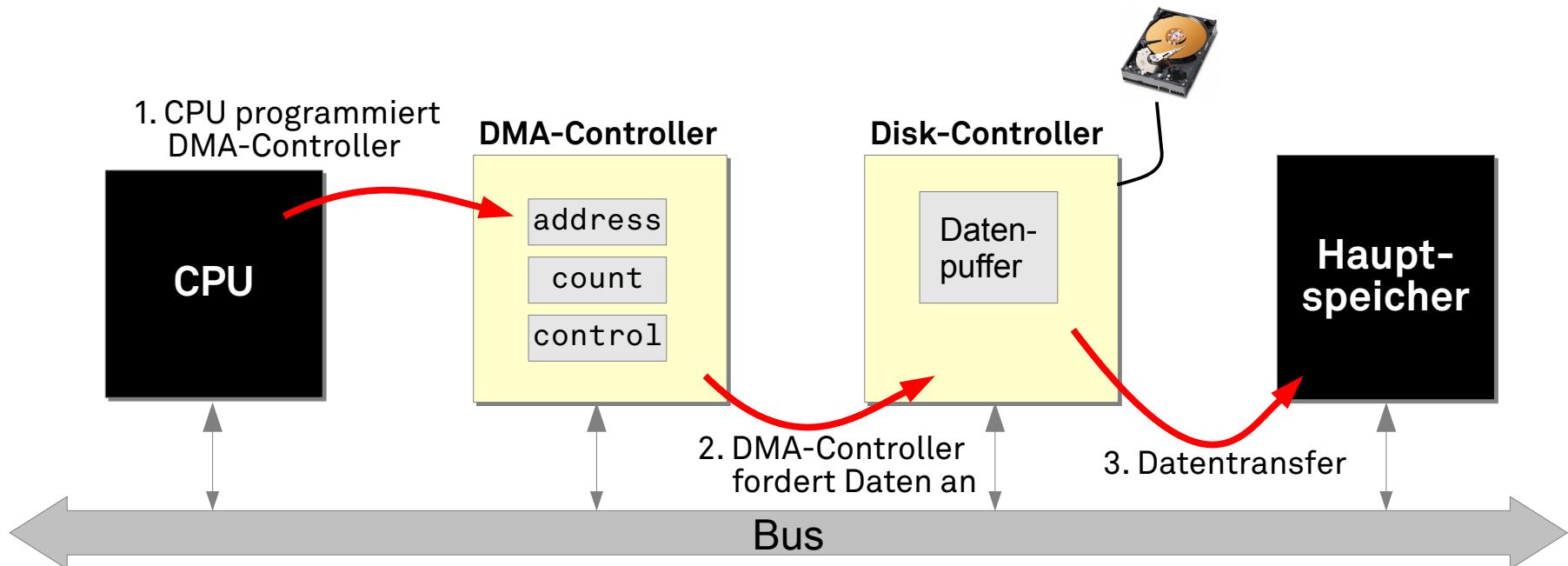
### Durchführung eines DMA-Transfers



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

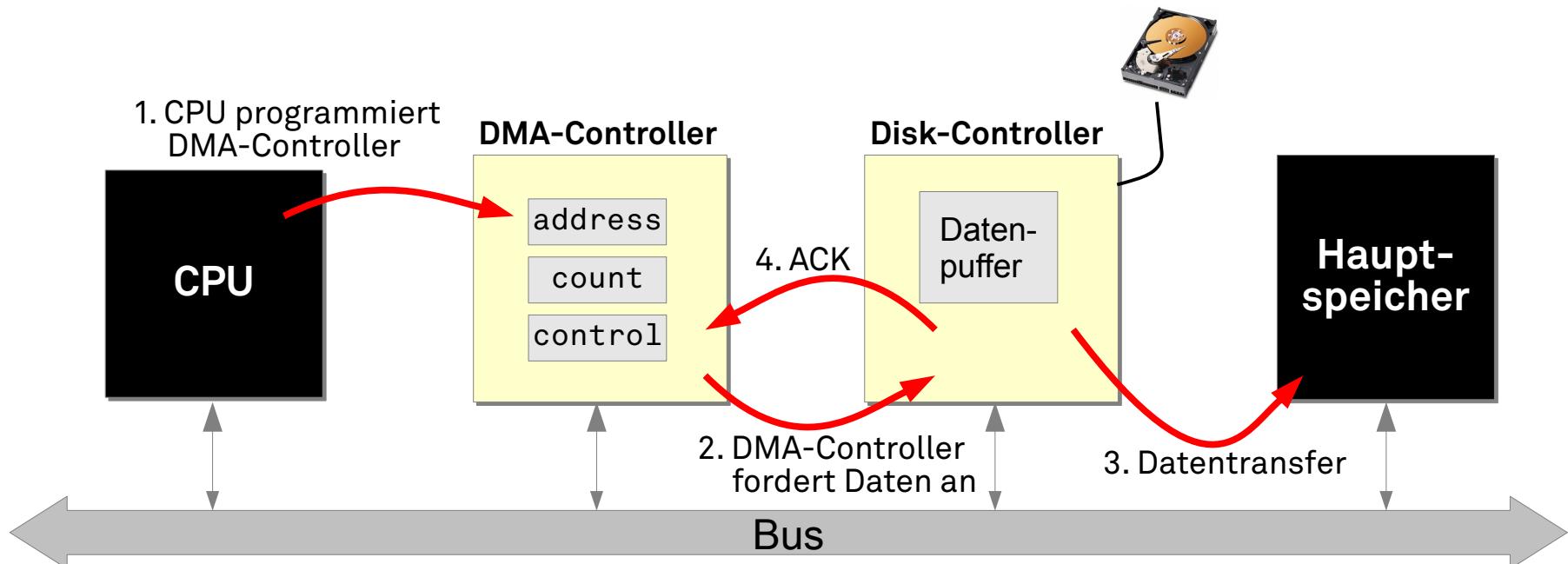
### Durchführung eines DMA-Transfers



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

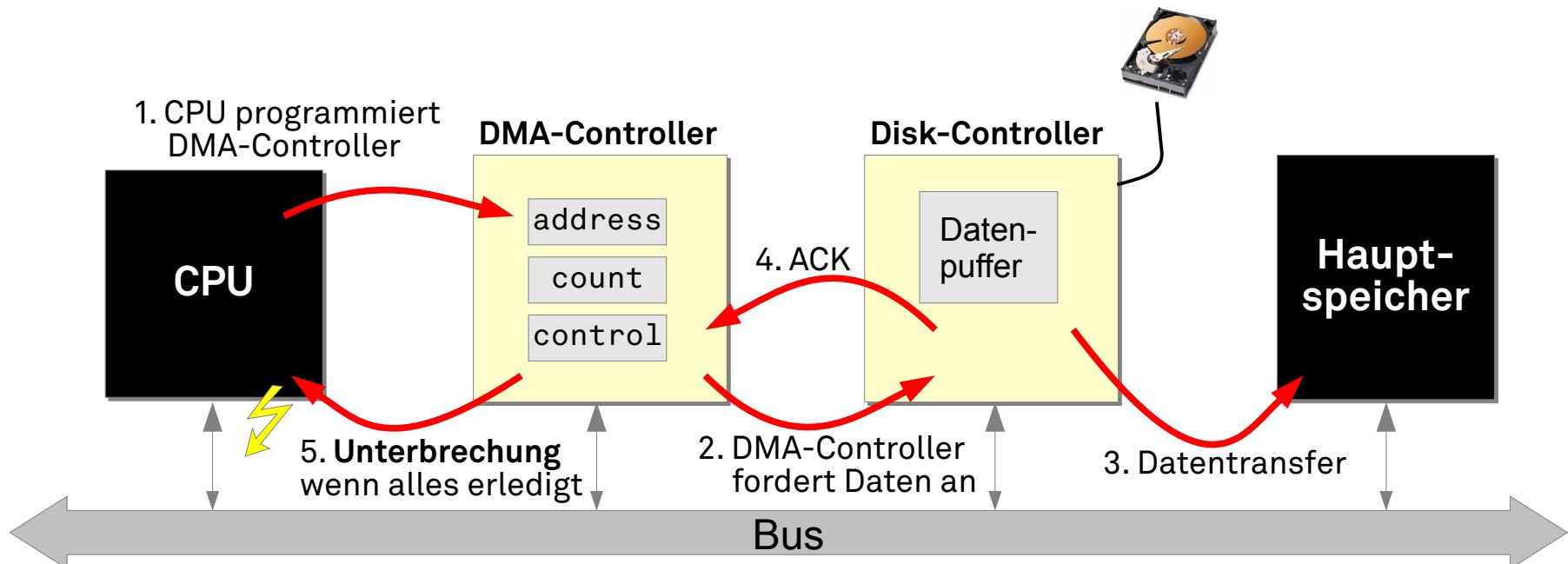
### Durchführung eines DMA-Transfers



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

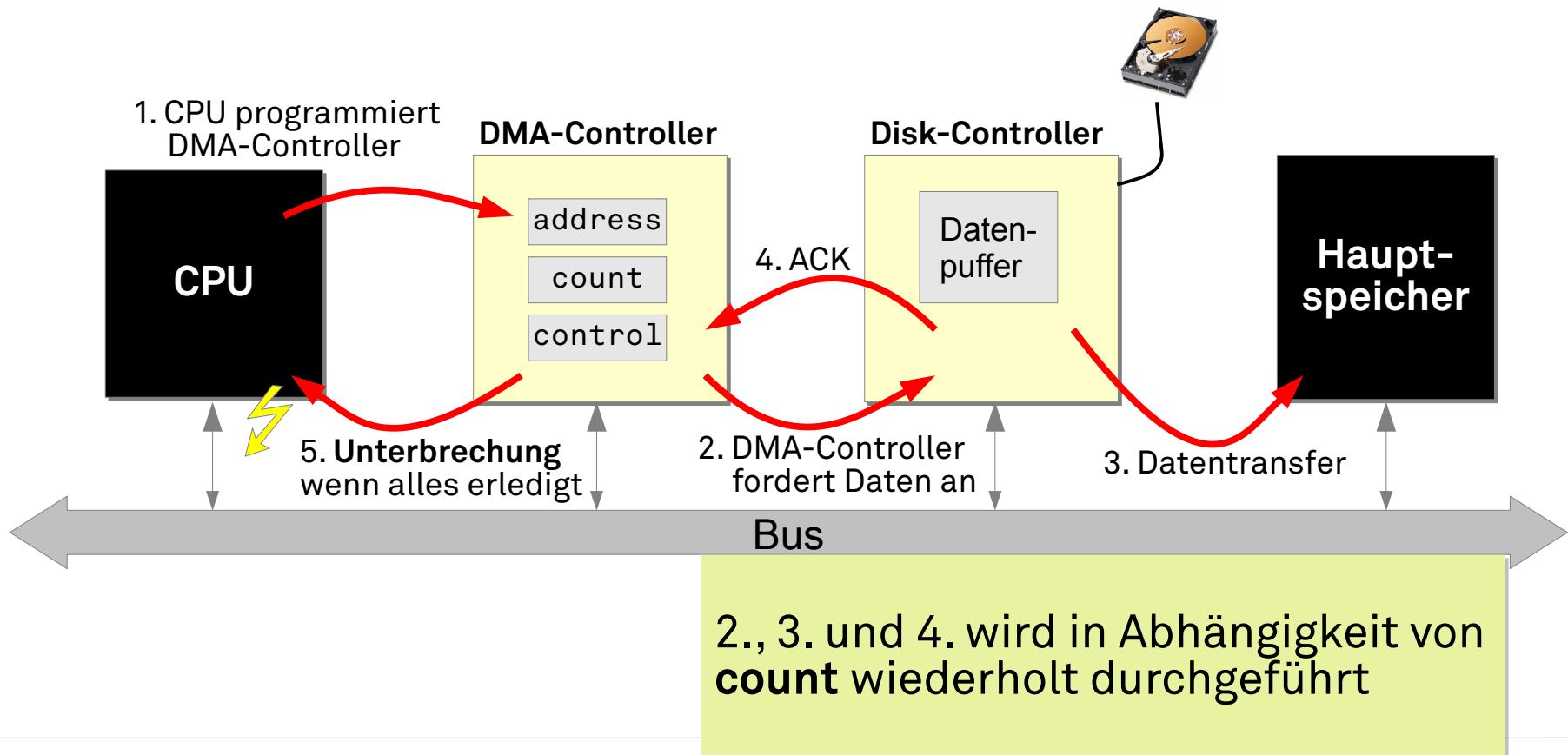
### Durchführung eines DMA-Transfers



## Direct Memory Access (DMA) ...

- ... wird von komplexen Controllern benutzt, um Daten unabhängig von der CPU in den bzw. aus dem Hauptspeicher zu transferieren.

### Durchführung eines DMA-Transfers

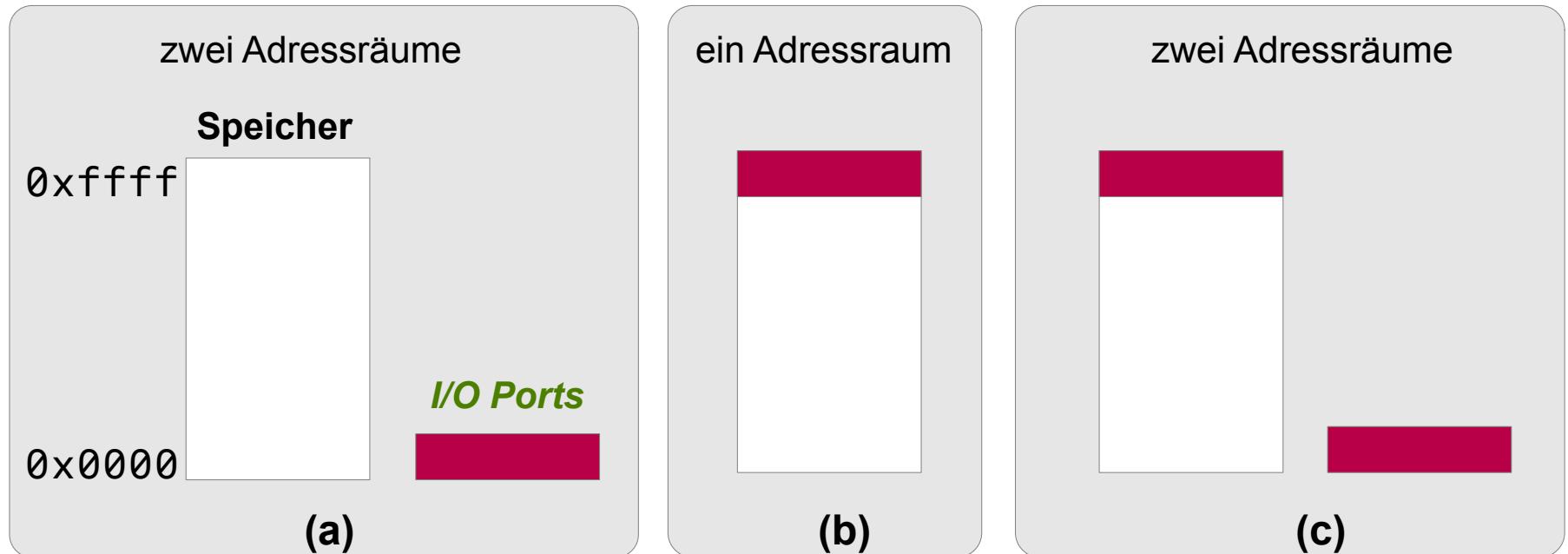


# Inhalt

- Ein-/Ausgabe-Hardware
- Gerätaprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

# Ein-/Ausgabeadressraum

- Zugriff auf *Controller*-Register und *Controller*-Speicher erfolgt je nach Systemarchitektur ...



## (a) separater E/A-Adressraum

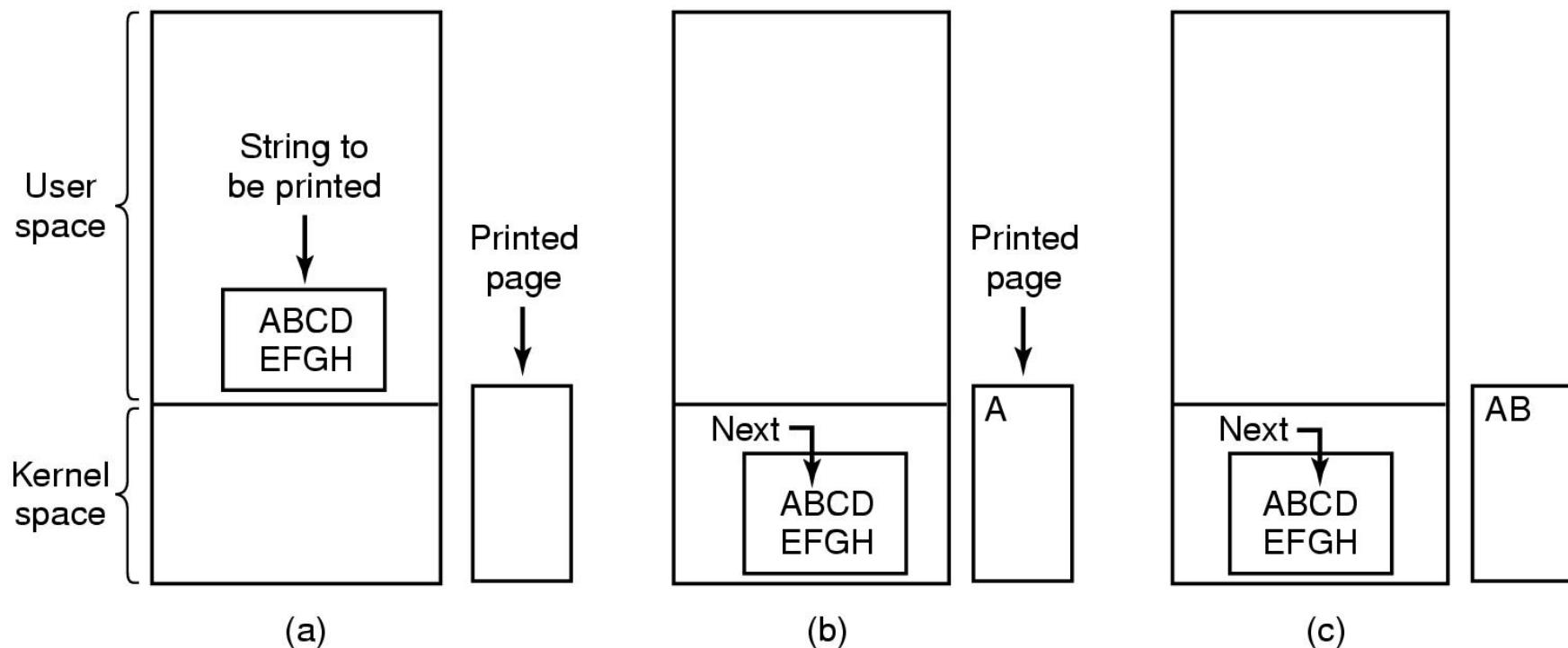
- anzusprechen über spezielle Maschineninstruktionen

## (b) gemeinsamer Adressraum (*Memory-Mapped I/O*)

## (c) hybride Architektur

# Arbeitsweise von Gerätetreibern

- Je nach Fähigkeiten des Geräts erfolgt E/A mittels ...
  - **Polling** (oder „Programmierte E/A“),
  - **Unterbrechungen** oder
  - **DMA**
- **Beispiel:** Drucken einer Textzeile



Quelle: Tanenbaum, „Modern Operating Systems“

# Polling (oder „Programmierte E/A“)

... bedeutet **aktives Warten** auf ein Ein-/Ausgabegerät.

```
/* Zeichen in Kern-Puffer p kopieren */
copy_from_user (buffer, p, count);

/* Schleife über alle Zeichen */
for (i = 0; i < count; i++) {

    /* Warte "aktiv" bis Drucker bereit */
    while (*printer_status_reg != READY)
        ;

    /* Ein Zeichen ausgeben */
    *printer_data_reg = p[i];
}

return_to_user ();
```

Pseudo-Code einer Betriebssystemfunktion zum Drucken von Text im **Polling-Betrieb**

# Unterbrechungsgetriebene E/A

... bedeutet, dass die CPU während der Wartezeit einem anderen Prozess zugewiesen werden kann.

## Code, der die E/A-Operation initiiert

```
copy_from_user (buffer, p, count);

/* Druckerunterbrechungen erlauben */
enable_interrupts ();

/* Warte bis Drucker bereit */
while (*printer_status_reg != READY);

/* Erstes Zeichen ausgeben */
*printer_data_reg = p[i++];

scheduler ();
return_to_user ();
```

## Unterbrechungsbehandlungsroutine

```
if (count > 0) {
    *printer_data_reg = p[i];
    count--;
    i++;
}
else
    unblock_user ();

acknowledge_interrupt ();
return_from_interrupt ();
```

# DMA-getriebene E/A

... bedeutet, dass die Software nicht mehr für den Datentransfer zwischen Controller und Hauptspeicher zuständig ist.

- Die CPU wird weiter entlastet.

```
copy_from_user (buffer, p, count);  
set_up_DMA_controller (p, count);  
scheduler ();  
return_to_user ();
```

Code, der die E/A-Operation initiiert

```
acknowledge_interrupt ();  
unblock_user ();  
return_from_interrupt ();
```

Unterbrechungsbehandlungsroutine

# Diskussion: Unterbrechungen

## ■ Kontextsicherung

- Wird teilweise von der CPU selbst erledigt.
  - z.B. Statusregister und Rücksprungadresse, aber nur das Minimum.
- **Alle veränderten Register** müssen gesichert und am Ende der Behandlung wiederhergestellt werden.

## ■ Behandlungsroutine **möglichst kurz**

- Während der Unterbrechungsbehandlung werden i.d.R. weitere Unterbrechungen unterdrückt.
  - Es droht der **Verlust von Unterbrechungen**
- Möglichst nur den Prozess wecken, der auf E/A-Beendigung wartet.

# Diskussion: Unterbrechungen (2)

- Unterbrechungen sind die Quelle der Asynchronität
  - Ursache für Race Conditions im Betriebssystemkern
- Unterbrechungssynchronisation:
  - einfachste Möglichkeit: **Unterbrechungsbehandlung** durch die CPU zeitweise *hart verbieten*, während kritische Abschnitte durchlaufen werden.
    - x86: sti, cli
    - wieder Gefahr des **Unterbrechungsverlusts**
  - BS gängig: **mehrstufige Behandlungen**, durch die das harte Sperren von Unterbrechungen minimiert wird
    - Abstrakt: **Prolog** (asynchron) und **Epilog** (synchron zum BS)
    - UNIX: *Top Half, Bottom Half*
    - Linux: *Tasklets*
    - Windows: *Deferred Procedures*

# Diskussion: *Direct Memory Access*

## ■ Caches

- Heutige Prozessoren arbeiten mit Daten-Caches;  
**DMA läuft am Cache vorbei!**
- Vor dem Aufsetzen eines DMA-Vorgangs muss der Cache-Inhalt in den Hauptspeicher zurückgeschrieben und invalidiert werden bzw. der Cache darf für die entsprechende Speicherregion nicht eingesetzt werden.

## ■ Speicherschutz

- Heutige Prozessoren verwenden eine MMU zur Isolation von Prozessen und zum Schutz des Betriebssystems;  
**DMA läuft am Speicherschutz vorbei!**
- Fehler beim Aufsetzen von DMA-Vorgängen sind extrem kritisch.
- Anwendungsprozesse dürfen DMA-Controller nie direkt programmieren!

# Inhalt

- Ein-/Ausgabe-Hardware
- Geräteprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

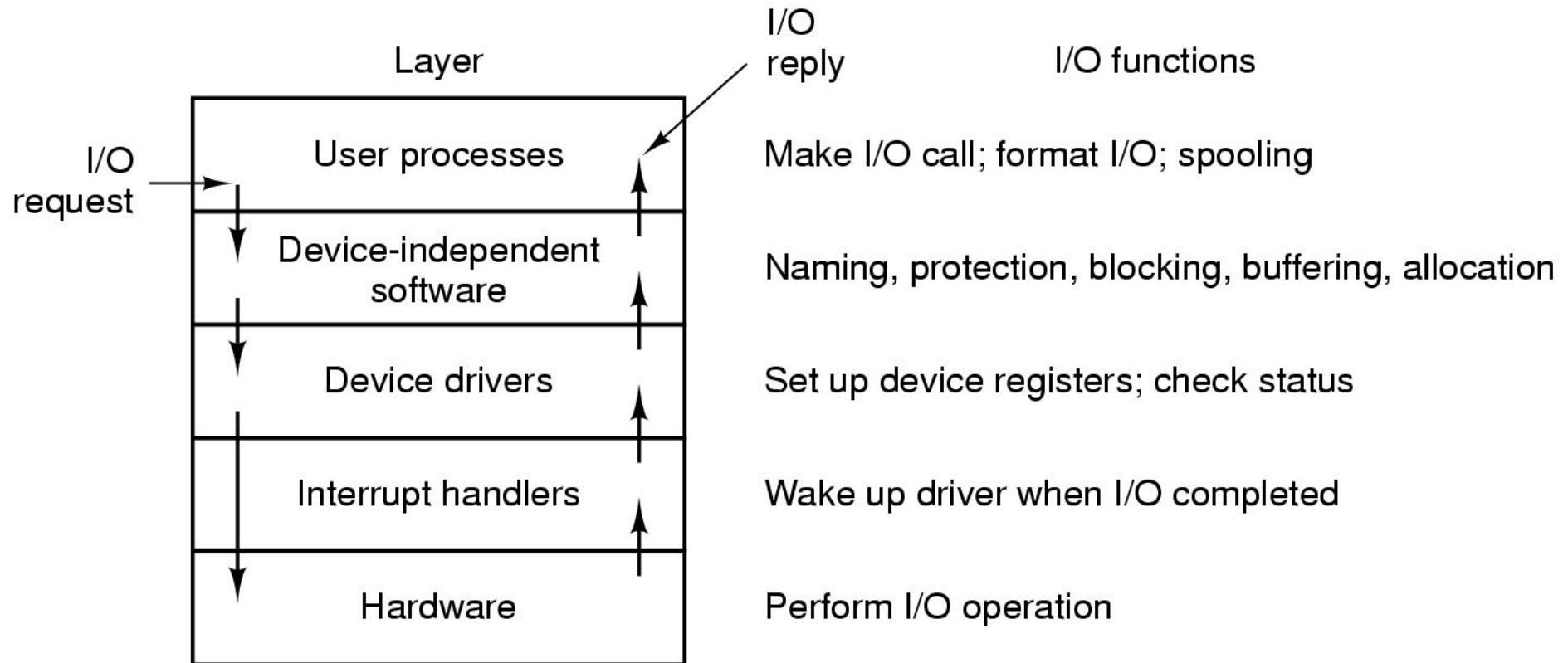
# Aufgaben des Betriebssystems

- **Geräteabstraktionen** schaffen
  - einheitlich, einfach, aber vielseitig
- **Ein-/Ausgabenprimitiven** bereitstellen
  - synchron und/oder asynchron
- **Pufferung**
  - falls das Gerät bzw. der Empfängerprozess noch nicht bereit ist
- **Geräteansteuerung**
  - möglichst effizient unter Beachtung mechanischer Eigenschaften
- **Ressourcenzuordnung** verwalten
  - bei **teilbaren** Geräten: Welcher Prozess darf wo lesen/schreiben?
  - bei **unteilbaren** Geräten: zeitweise Reservierungen
- **Stromsparzustände** verwalten
- **Plug&Play** unterstützen
- ...

# Aufgaben des Betriebssystems

- **Geräteabstraktionen** schaffen
  - einheitlich, einfach, aber vielseitig
- **Ein-/Ausgabenprimitiven** bereitstellen
  - synchron und/oder asynchron
- **Pufferung**
  - falls das Gerät bzw. der Empfängerprozess noch nicht bereit ist
- **Geräteansteuerung**
  - möglichst effizient unter Beachtung mechanischer Eigenschaften
- **Ressourcenzuordnung** verwalten
  - bei teilbaren Geräten: Welcher Prozess darf wo lesen/schreiben?
  - bei unteilbaren Geräten: zeitweise Reservierungen
- **Stromsparzustände** verwalten
- **Plug&Play** unterstützen
- ...

# Schichten des E/A-Subsystems



Quelle: Tanenbaum, „Modern Operating Systems“

# UNIX: Geräteabstraktionen

- Periphere Geräte werden als **Spezialdateien** repräsentiert:
  - Geräte können **wie Dateien** mit Lese- und Schreiboperationen angesprochen werden.
  - Öffnen der Spezialdateien schafft eine Verbindung zum Gerät, die durch einen Treiber hergestellt wird.
  - direkter Durchgriff vom Anwender auf den Treiber
- **blockorientierte Spezialdateien** (block devices)
  - Plattenlaufwerke, Bandlaufwerke, Floppy Disks, CD-ROMs
- **zeichenorientierte Spezialdateien** (character devices)
  - serielle Schnittstellen, Drucker, Audiokanäle etc.

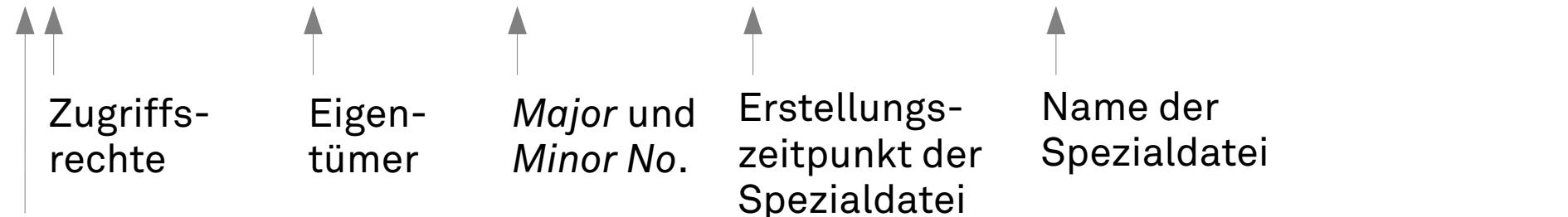
## UNIX: Geräteabstraktionen (2)

- Eindeutige Beschreibung der Geräte durch ein 3-Tupel:  
(Gerätetyp, *Major Number*, *Minor Number*)
- **Gerätetyp:** *Block Device*, *Character Device*
- **Major Number:**  
Auswahlnummer für einen Treiber
- **Minor Number:**  
Auswahl eines Gerätes innerhalb eines Treibers

# UNIX: Geräteabstraktionen (3)

- Auszug aus dem *Listing* des **/dev**-Verzeichnisses

```
brw-rw---- ulbr disk 3,  0 2008-06-15 14:14 /dev/hda
brw-rw---- ulbr disk 3, 64 2008-06-15 14:14 /dev/hdb
brw-r----- root  disk 8,  0 2008-06-15 14:13 /dev/sda
brw-r----- root  disk 8,  1 2008-06-15 14:13 /dev/sda1
crw-rw---- root  uucp 4, 64 2006-05-02 08:45 /dev/ttyS0
crw-rw---- root  lp   6,  0 2008-06-15 14:13 /dev/lp0
crw-rw-rw- root  root  1,  3 2006-05-02 08:45 /dev/null
lrwxrwxrwx root  root   3 2008-06-15 14:14 /dev/cdrecorder -> hdb
lrwxrwxrwx root  root   3 2008-06-15 14:14 /dev/cdrom -> hda
```



Zugriffs-  
rechte      Eigen-  
tümer      Major und  
Minor No.      Erstellungs-  
zeitpunkt der  
Spezialdatei      Name der  
Spezialdatei

c: *character device*

b: *block device*

l: *link*

# UNIX: Zugriffsprimitive

Das Wichtigste in Kürze ... (siehe man 2 ...)

- **int open(const char \*devname, int flags)**
  - Öffnen eines Geräts. Liefert Dateideskriptor als Rückgabewert.
- **off\_t lseek(int fd, off\_t offset, int whence)**
  - Positioniert den Schreib-/Lesezeiger
  - nur bei Geräten mit **wahlfreiem Zugriff**
- **ssize\_t read(int fd, void \*buf, size\_t count)**
  - Einlesen von max. **count** Bytes in Puffer **buf** von Deskriptor **fd**
- **ssize\_t write(int fd, const void \*buf, size\_t count)**
  - Schreiben von **count** Bytes aus Puffer **buf** auf Deskriptor **fd**
- **int close(int fd)**
  - Schließen eines Geräts: Dateideskriptor **fd** kann danach nicht mehr benutzt werden.

# UNIX: Gerätespezifische Funktionen

- 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 d, int request, ...);
```

- Schnittstelle generisch, Semantik gerätespezifisch:

**CONFORMING TO**

No single standard. Arguments, returns, and semantics of `ioctl(2)` 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 AT&T Unix.

# UNIX: Warten auf mehrere Geräte

- bisher: **Lese- oder Schreibafrufe blockieren**
  - Was tun beim Lesen von mehreren Quellen?
- **Alternative 1:** nichtblockierende Ein-/Ausgabe
  - O\_NDELAY beim `open()`
  - **Polling-Betrieb:** Prozess muss immer wieder `read()` aufrufen
  - unbefriedigend, da **Verschwendungen von CPU-Zeit** bis etwas vorliegt

# UNIX: Warten auf mehrere Geräte (2)

## ■ Alternative 2: Blockieren an mehreren Dateideskriptoren

- Systemaufruf:

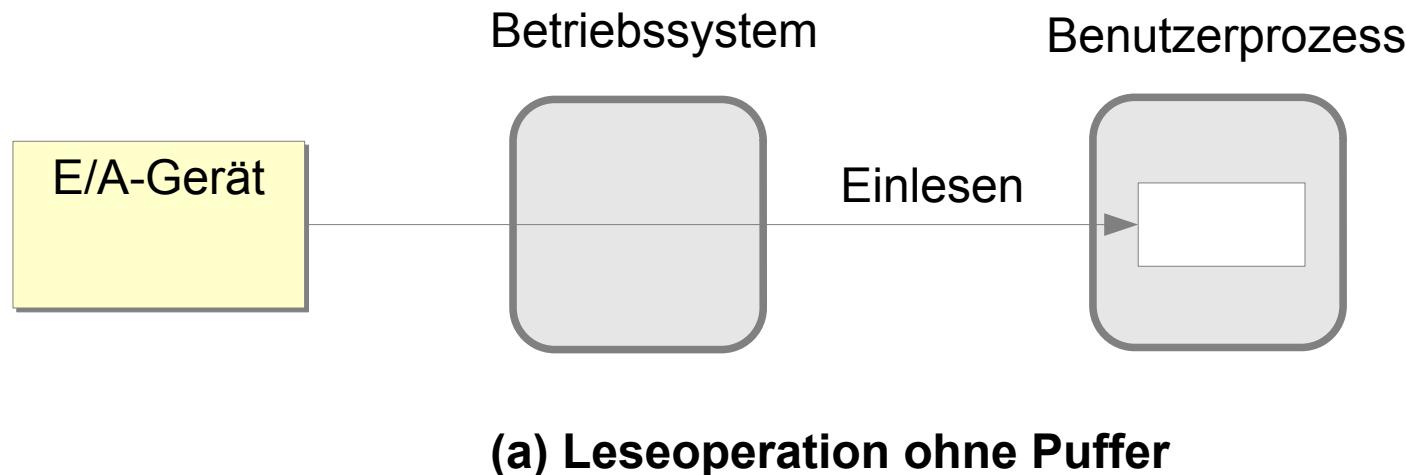
```
int select (int nfds, fd_set *readfds, fd_set *writefds,  
            fd_set *errorfds, struct timeval *timeout);
```

- **nfds** legt fest, bis zu welchem Dateideskriptor **select** wirken soll.
- ...**fds** sind Dateideskriptoren, auf die gewartet werden soll:
  - **readfds** — bis etwas zum Lesen vorhanden ist
  - **writefds** — bis man schreiben kann
  - **errorfds** — bis ein Fehler aufgetreten ist
- **Timeout** legt fest, wann der Aufruf spätestens deblockiert.
- Makros zum Erzeugen der Dateideskriptormengen
- **Ergebnis:** In den Dateideskriptormengen sind nur noch die Dateideskriptoren vorhanden, die zur Deblockade führten.

# Pufferung bei E/A-Operationen

## ■ **Probleme** ohne Datenpuffer im Betriebssystem:

- Daten, die eintreffen, bevor `read` ausgeführt wurde (z.B. von der Tastatur), müssten verloren gehen.
- Wenn ein Ausgabegerät beschäftigt ist, müsste `write` scheitern oder den Prozess blockieren, bis das Gerät wieder bereit ist.
- Ein Prozess, der eine E/A-Operation durchführt, kann nicht ausgelagert werden.



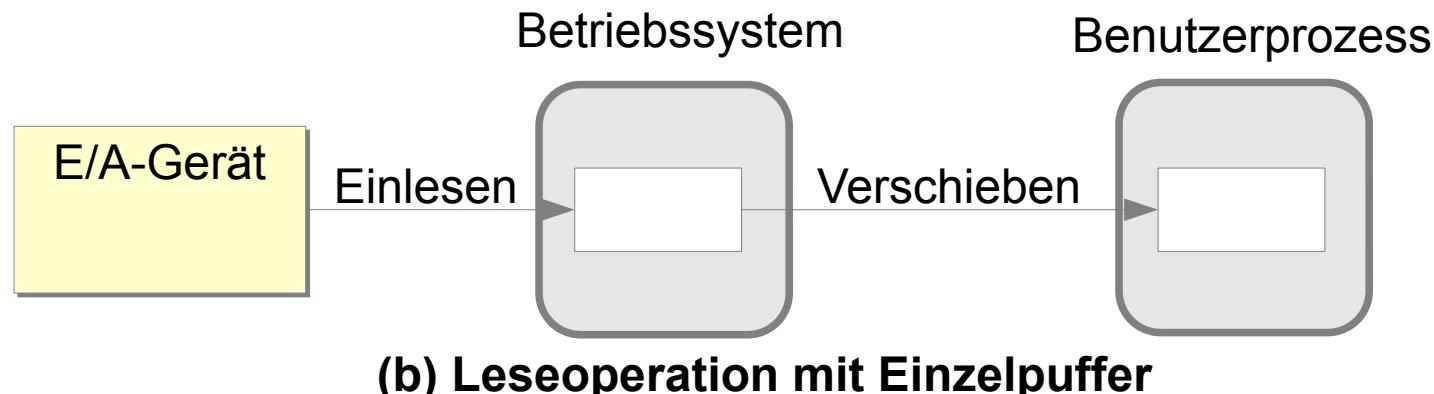
# E/A-Einzelpuffer

## ■ Einlesen:

- Daten können vom System entgegengenommen werden, auch wenn der Leserprozess noch nicht `read` aufgerufen hat.
- Bei Blockgeräten kann der nächste Block vorausschauend gelesen werden, während der vorherige verarbeitet wird.
- Prozess kann problemlos ausgelagert werden. DMA erfolgt in Puffer.

## ■ Schreiben:

- Daten werden kopiert. Aufrufer blockiert nicht. Datenpuffer im Benutzeradressraum kann sofort wiederverwendet werden.



# E/A-Einzelpuffer



## Leistungsabschätzung

Eine einfache Rechnung zeigt den Leistungsgewinn beim wiederholten blockweisen Lesen mit anschließender Verarbeitung:

- T: Dauer der Leseoperation
- C: Rechenzeit für die Verarbeitung
- M: Dauer des Kopiervorgang (Systempuffer→Benutzerprozess)

G: Gesamtdauer für Lesen und Verarbeiten eines Blocks

**ohne Puffer:**  $G_0 = T + C$

**mit Puffer:**  $G_E = \max(T, C) + M$

**Mit  $T \approx C$  und  $M \approx 0$  wäre  $G_0 \approx 2 \cdot G_E$ . Leider ist  $M > 0$ .**

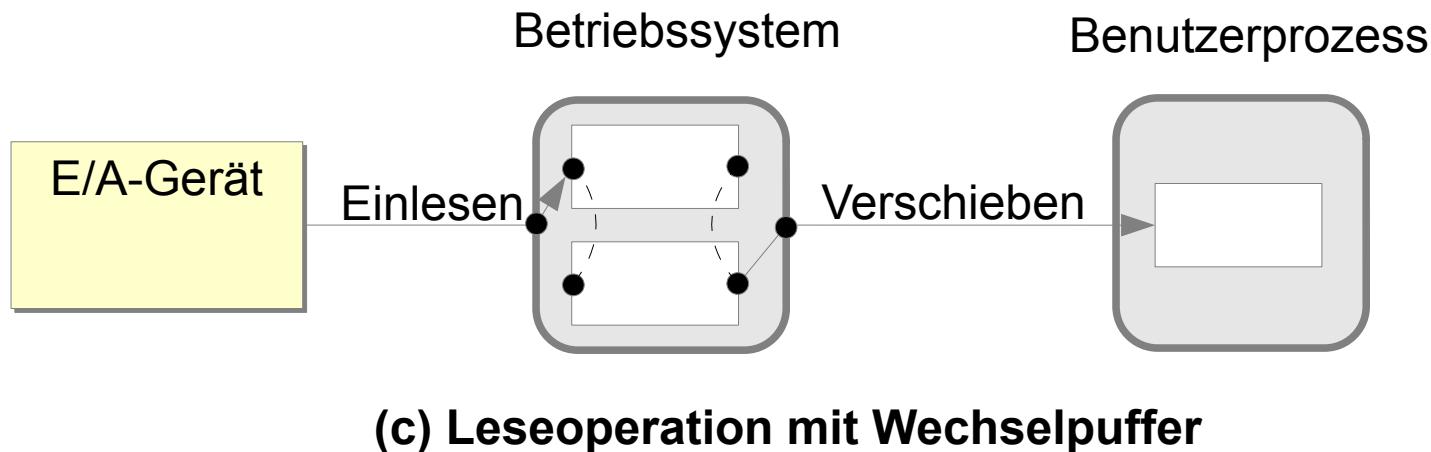
# E/A-Wechselpuffer

## ■ Einlesen:

- Während Daten vom E/A-Gerät in den einen Puffer transferiert werden, kann der andere Pufferinhalt in den Empfängeradressraum kopiert werden.

## ■ Schreiben:

- Während Daten aus einem Puffer zum E/A-Gerät transferiert werden, kann der andere Puffer bereits mit neuen Daten aus dem Senderadressraum gefüllt werden.



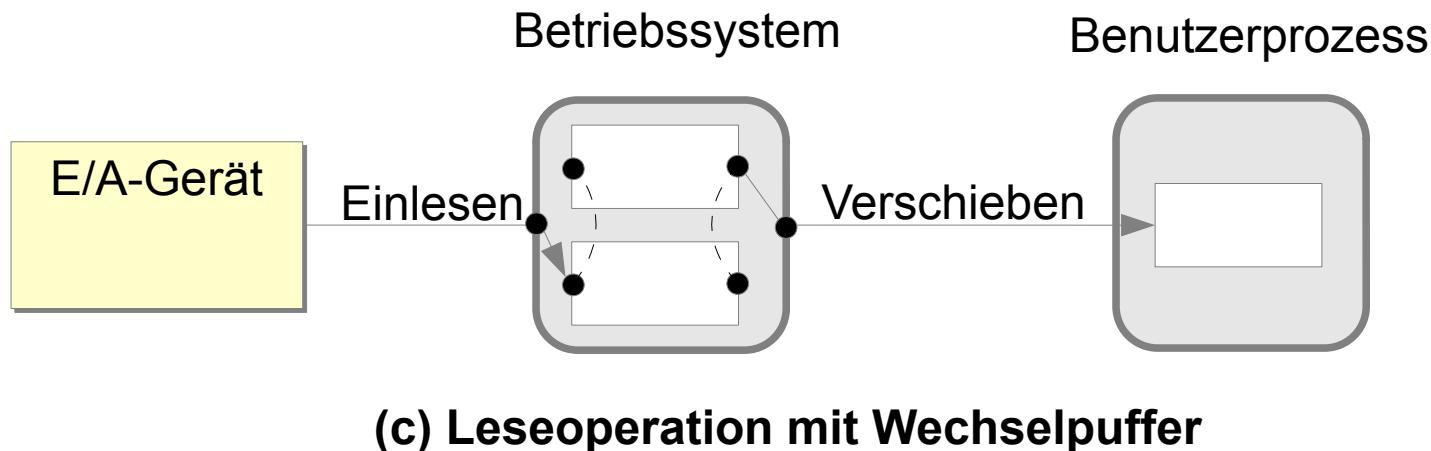
# E/A-Wechselpuffer

## ■ Einlesen:

- Während Daten vom E/A-Gerät in den einen Puffer transferiert werden, kann der andere Pufferinhalt in den Empfängeradressraum kopiert werden.

## ■ Schreiben:

- Während Daten aus einem Puffer zum E/A-Gerät transferiert werden, kann der andere Puffer bereits mit neuen Daten aus dem Senderadressraum gefüllt werden.



# E/A-Wechselpuffer

## ■ Einlesen:

### Leistungsabschätzung

Mit einem Wechselpuffer kann ein Leseoperation parallel zur Kopieroperation und Verarbeitung erfolgen.

**ohne Puffer:**  $G_0 = T + C$

**mit Puffer:**  $G_E = \max(T, C) + M$

**mit Wechselpuffer:**  $G_W = \max(T, C + M)$

Mit  $C + M \leq T$  könnte das Gerät  
zu 100% ausgelastet werden.

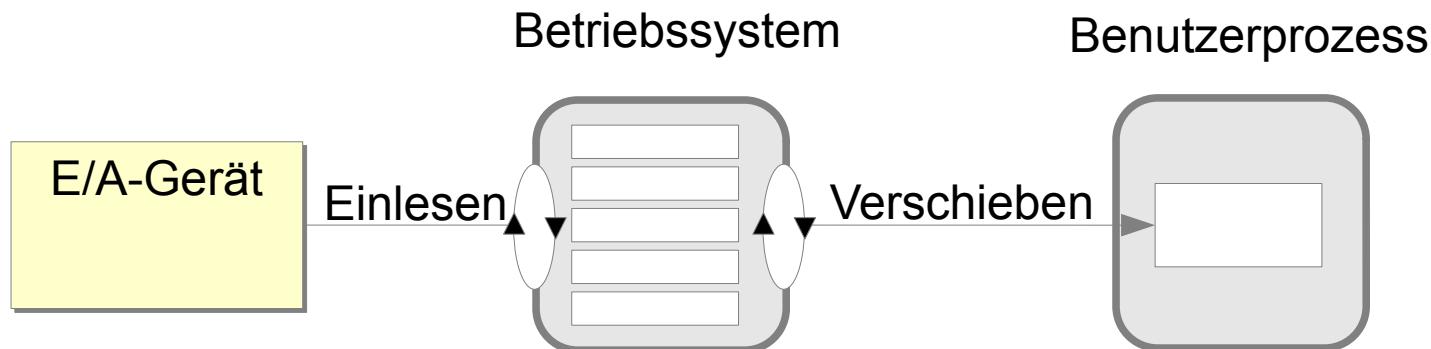
# E/A-Ringpuffer

## ■ Einlesen:

- Viele Daten können gepuffert werden, auch wenn der Leserprozess nicht schnell genug `read`-Aufrufe tätigt.

## ■ Schreiben:

- Ein Schreiberprozess kann mehrfach `write`-Aufrufe tätigen, ohne blockiert werden zu müssen.



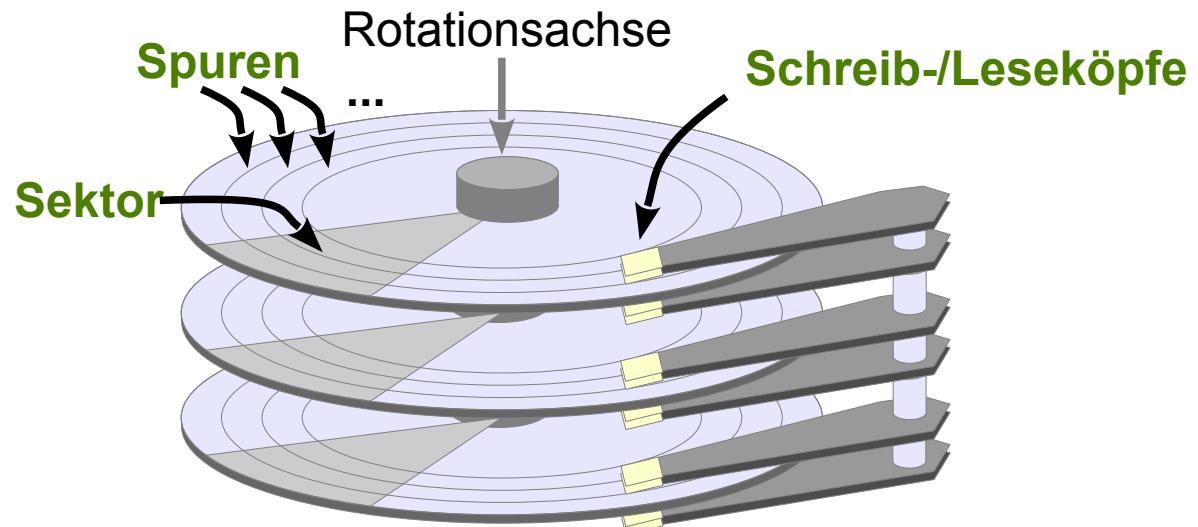
**(d) Leseoperation mit Ringpuffer**

# Diskussion: E/A-Puffer

- **E/A-Puffer entkoppeln** die E/A-Operationen der Nutzerprozesse vom Gerätetreiber
  - **Kurzfristig** lässt sich eine erhöhte Ankunftsrate an E/A-Aufträgen bewältigen.
  - **Langfristig** bleibt auch bei noch so vielen Puffern ein Blockieren von Prozessen (oder Verlust von Daten) nicht aus.
- **Puffer haben ihren Preis:**
  - Verwaltung der Pufferstruktur
  - Speicherplatz
  - Zeit für das Kopieren
- **In komplexen Systemen wird teilweise mehrfach gepuffert.**
  - **Beispiel:** Schichten von Netzwerkprotokollen
  - Nach Möglichkeit vermeiden!

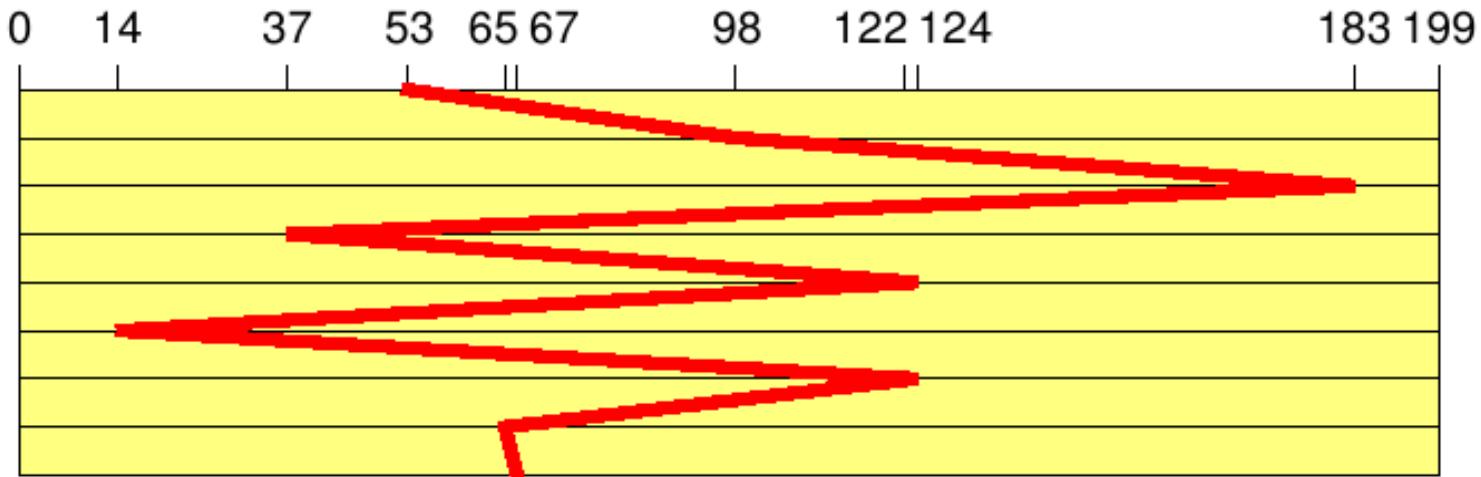
# Geräteansteuerung: Beispiel Festplatte

- Treiber muss **mechanische Eigenschaften** beachten!
- Plattentreiber hat in der Regel mehrere Aufträge in Warteschlange
  - Eine bestimmte Ordnung der Ausführung kann Effizienz steigern.
  - Zusammensetzung der Bearbeitungszeit eines Auftrags:
    - **Positionierungszeit:** abhängig von akt. Stellung des Plattenarms
    - **Rotationsverzögerung:** Zeit bis der Magnetkopf den Sektor bestreicht
    - **Übertragungszeit:** Zeit zur Übertragung der eigentlichen Daten
- Ansatzpunkt:  
**Positionierungszeit**



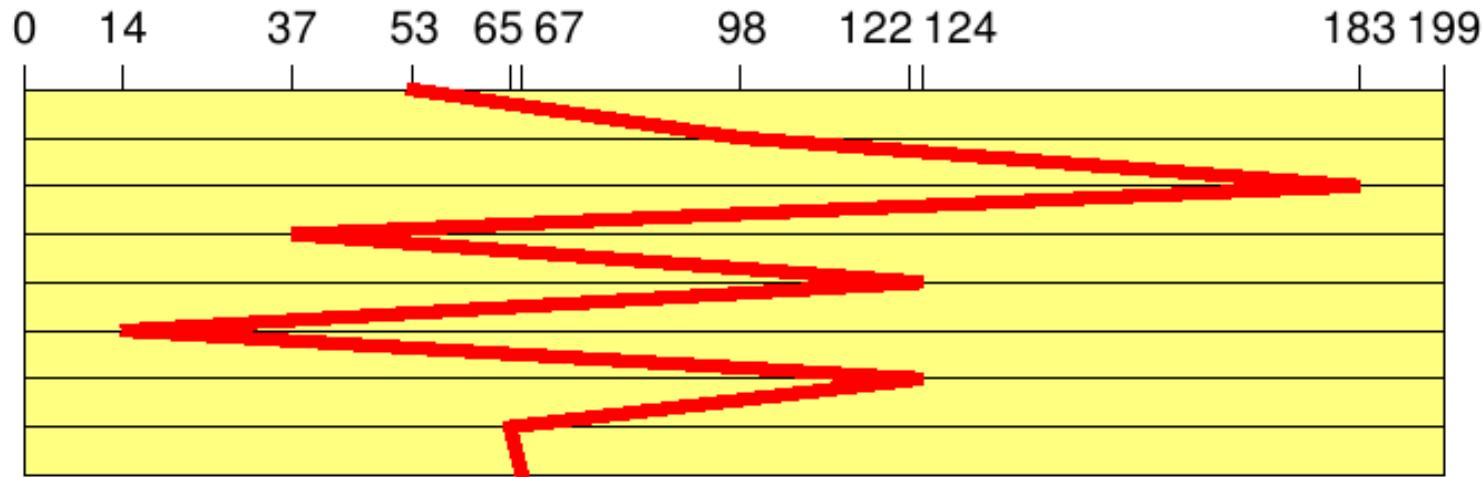
## E/A-Scheduling: FIFO

- Bearbeitung gemäß Ankunft des Auftrags  
**(First In First Out)**
  - Referenzfolge (Folge von Spurnummern):  
98, 183, 37, 122, 14, 124, 65, 67
  - Aktuelle Spur: 53



## E/A-Scheduling: FIFO

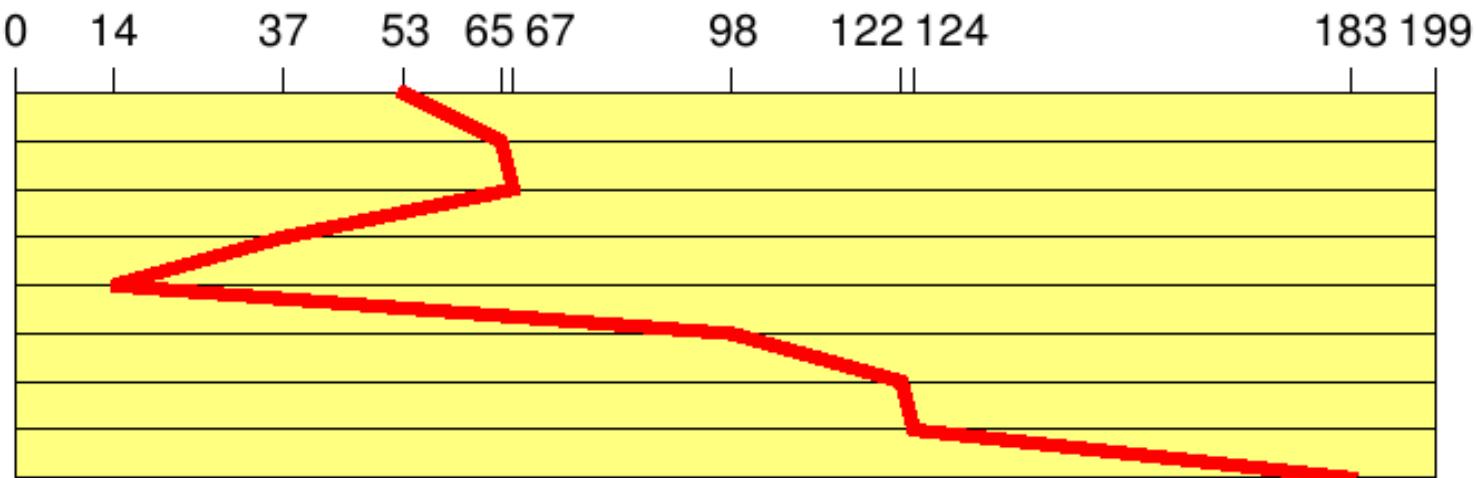
- Bearbeitung gemäß Ankunft des Auftrags  
**(First In First Out)**
  - Referenzfolge (Folge von Spurnummern):  
98, 183, 37, 122, 14, 124, 65, 67
  - Aktuelle Spur: 53



- Gesamtzahl der **Spurwechsel: 640**
- Weite Bewegungen des Schwenkarms:  
mittlere Bearbeitungsdauer lang!

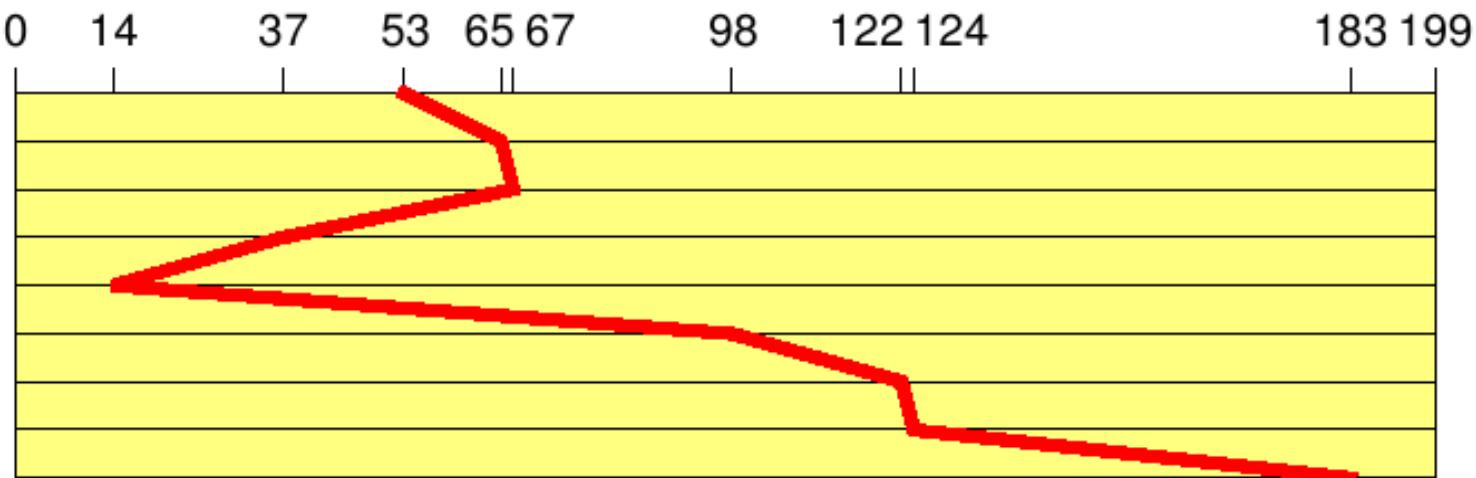
## E/A-Scheduling: SSTF

- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (**Shortest Seek Time First**)
  - dieselbe Referenzfolge: 98, 183, 37, 122, 14, 124, 65, 67
  - (Annahme: Positionierungszeit proportional zum Spurabstand)



## E/A-Scheduling: SSTF

- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (**Shortest Seek Time First**)
  - dieselbe Referenzfolge: 98, 183, 37, 122, 14, 124, 65, 67
  - (Annahme: Positionierungszeit proportional zum Spurabstand)



- Gesamtzahl der **Spurwechsel: 236**

## E/A-Scheduling: SSTF

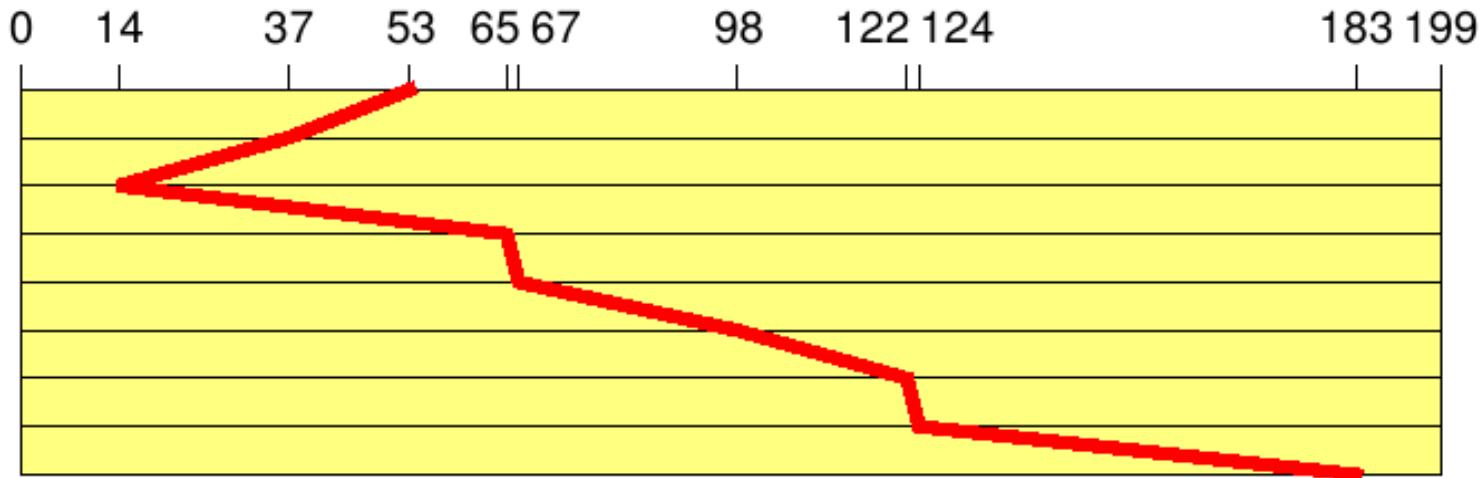
- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (**Shortest Seek Time First**)
  - dieselbe Referenzfolge: 98, 183, 37, 122, 14, 124, 65, 67
  - (Annahme: Positionierungszeit proportional zum Spurabstand)



- Gesamtzahl der **Spurwechsel: 236**
- ähnlich wie SJF kann auch SSTF zur **Aushungerung** führen!
- noch nicht optimal

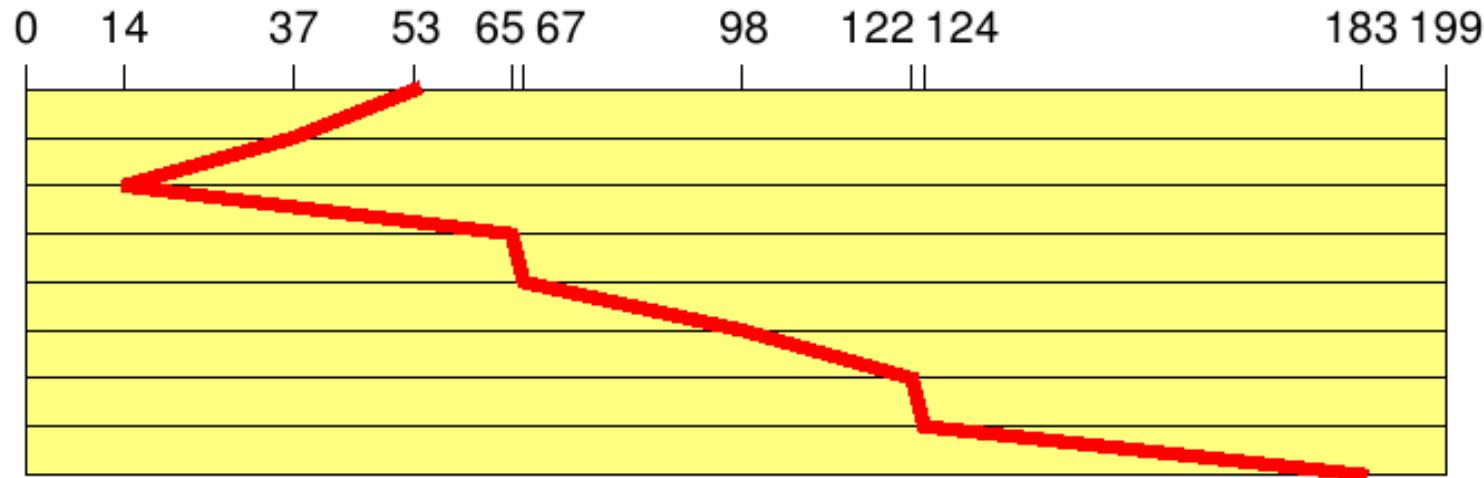
## E/A-Scheduling: Elevator

- Bewegung des Plattenarms in eine Richtung bis keine Aufträge mehr vorhanden sind (**Fahrstuhlstrategie**)
  - Gleiche Referenzfolge (Annahme: bisherige Kopfbewegung Richtung 0)



## E/A-Scheduling: Elevator

- Bewegung des Plattenarms in eine Richtung bis keine Aufträge mehr vorhanden sind (**Fahrstuhlstrategie**)
  - Gleiche Referenzfolge (Annahme: bisherige Kopfbewegung Richtung 0)



- Gesamtzahl der **Spurwechsel: 208**
- Neue Aufträge werden miterledigt ohne zusätzliche Positionierungszeit
- **Keine Aushungerung**, lange Wartezeiten aber nicht ausgeschlossen

# Diskussion: E/A-Scheduling heute

- Platten sind intelligente Geräte
  - Physikalische Eigenschaften werden verborgen (Logische Blöcke)
  - Platten weisen riesige Caches auf
  - *Solid State Disks* enthalten keine Mechanik mehr
- ➔ E/A-Scheduling verliert langsam an Bedeutung
- ➔ Erfolg einer Strategie ist schwerer vorherzusagen
  
- Trotzdem ist E/A-Scheduling noch immer sehr wichtig:
  - CPUs werden immer schneller, Platten kaum
  - Linux implementiert zur Zeit zwei verschiedene Varianten der **Fahrstuhlstrategie** (+ FIFO für „Platten“ ohne Positionierungszeit):
    - **DEADLINE**: Bevorzugung von Leseanforderungen (kürzere Deadlines)
    - **COMPLETELY FAIR**: Prozesse erhalten gleichen Anteil an E/A-Bandbreite

# Inhalt

- Ein-/Ausgabe-Hardware
- Gerätaprogrammierung
- Aufgaben des Betriebssystems
- Zusammenfassung

# Zusammenfassung

- E/A-Hardware ist sehr unterschiedlich
  - teilweise auch „hässlich“ zu programmieren
- Die Kunst des Betriebssystembaus besteht darin, ...
  - trotzdem **einheitliche und einfache Schnittstellen** zu definieren
  - **effizient** mit der Hardware umzugehen
  - CPU und E/A-**Geräteauslastung zu maximieren.**
- Gerätetreibervielfalt ist für den Erfolg eines Betriebssystems extrem wichtig.
  - Bei Systemen wie Linux und Windows sind die Gerätetreiber das weitaus größte Subsystem.