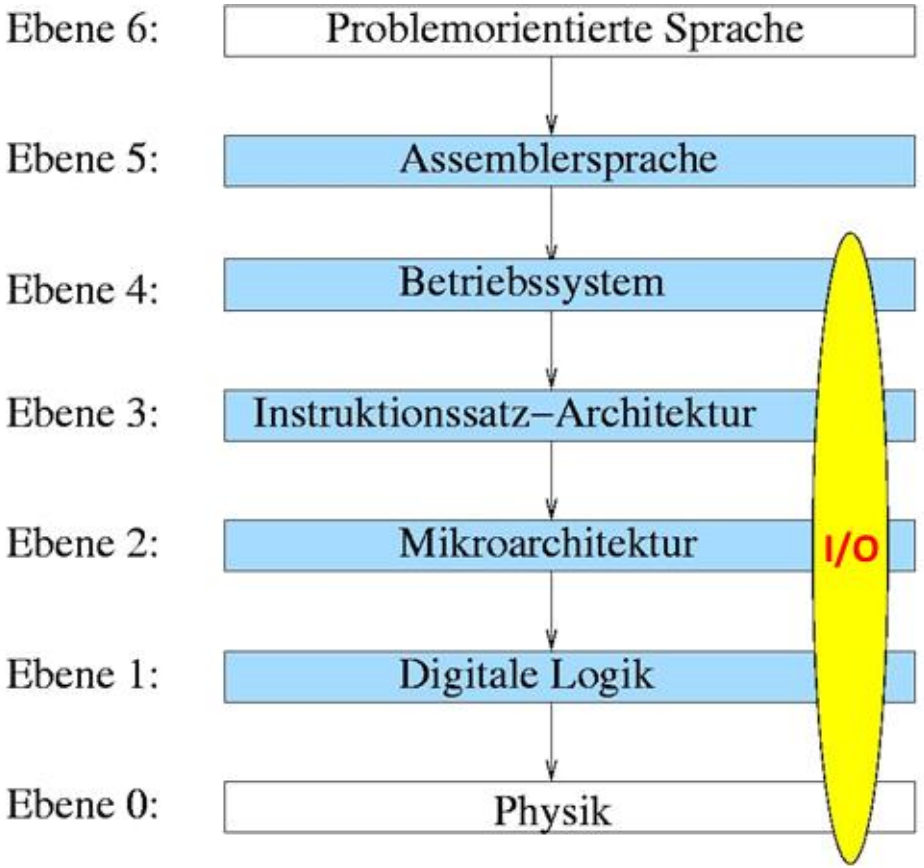
Ein-Ausgabekonzepte und Bussysteme

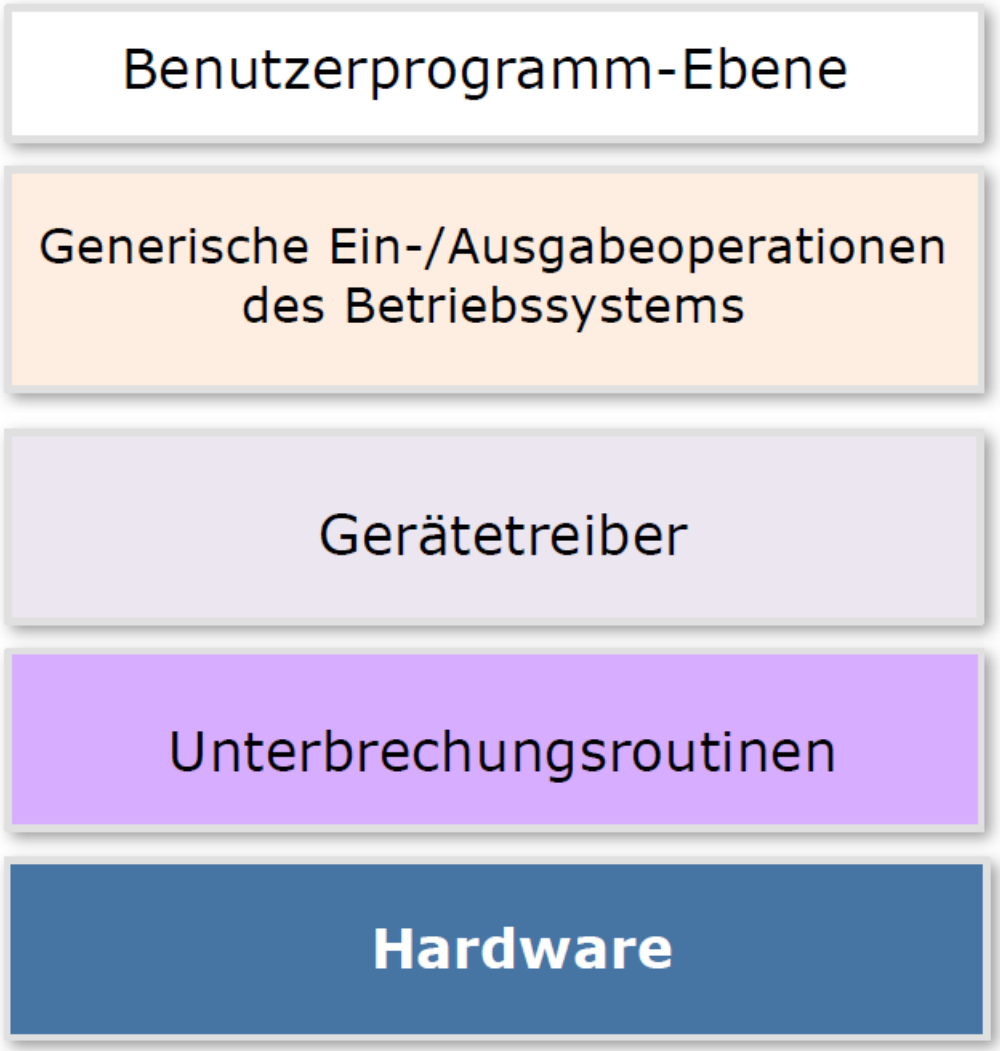
1. Ein-/Ausgabe
2. Strategien der Ein-/Ausgabe
3. Techniken der Datenübertragung
4. Punkt-zu-Punkt Schnittstellen
5. Direct Memory Access (DMA)
6. Bussysteme
7. Gerätetreiber
8. Pufferung
9. E/A-Scheduling

Ein-Ausgabekonzepte und Bussysteme

- Einordnung in das Schichtenmodell:



Ein-/Ausgabe-Schnittstelle



Systemaufrufe kapseln
Geräteverhalten in generische Klassen

Gerätetreiber kapseln die Unterschiede zwischen den verschiedenen Geräte-Controllern

Jede Schicht soll eine wohldefinierte Schnittstelle definieren mit einer wohldefinierten Funktionalität

Ein-/Ausgabe Realisierung

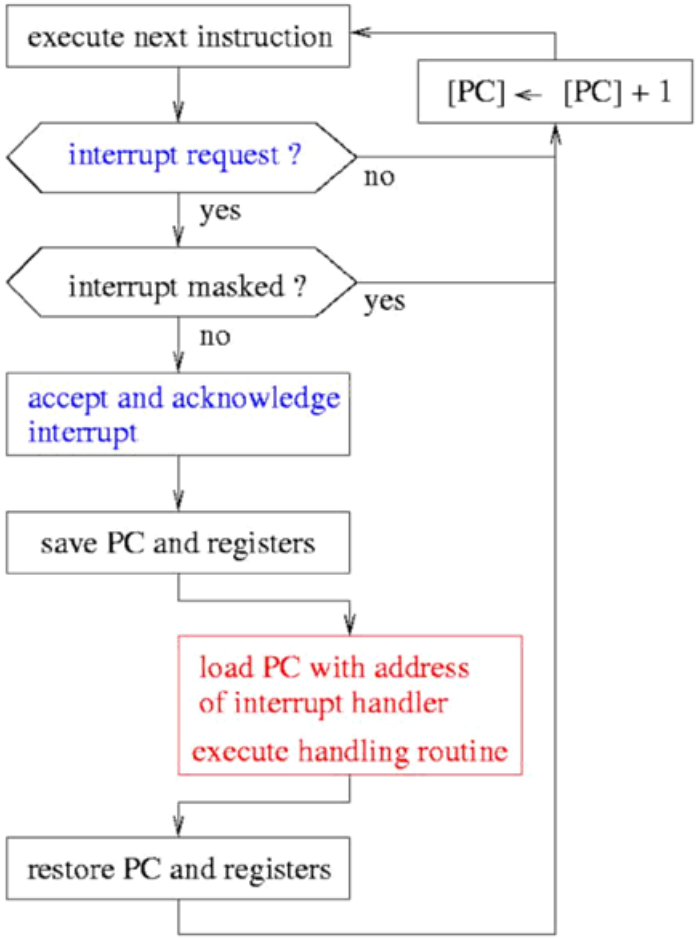
- Zur Realisierung einer Ein-/Ausgabe werden benötigt:
  - eine **Strategie** der Ein-/Ausgabe
  - eine nach Möglichkeit **standardisierte Schnittstelle** (*Interface*) zur Ein-/Ausgabe, z.B. seriell (RS-232) oder parallel (EPP):
    - Definition eines *Übertragungsprotokoll*
    - Definition von Steckern/Buchsen und Kabeln
    - Hardwareunterstützung durch entsprechenden E/A-Baustein (*I/O Controller*), ggf. separate *Schnittstellenkarte*
  - **E/A-Geräte** bzw. **Peripheriegeräte**
    - zur Umwandlung von elektrischen Signalen in eine verwertbare physikalische Form ...
      (z.B. Bildschirm, Drucker)
    - ... und umgekehrt
      (z.B. Maus, Tastatur, Scanner)

Strategien der Ein-/Ausgabe

- **Programmierte Ein-/Ausgabe**:
  - ein laufendes Programm (z.B. ein Anwendungsprogramm) legt explizit fest, *wann* eine Ein-/Ausgabe erfolgt
  - Falls der Ausgabebaustein noch nicht bereit ist, kann CPU in einer Warteschleife auf dessen Bereitschaft warten (*Busy Waiting*)
  - Eingabebausteine können zu definierten Zeitpunkten abgefragt werden, ob neue Eingabedaten anliegen (*Polling*)
- **Unterbrechungen** (*Interrupts*):
  - bei Eintreffen neuer Daten kann ein Eingabebaustein eine Unterbrechung anfordern (*Interrupt Request*)
  - Ausgabebaustein kann durch eine Unterbrechungsanforderung der CPU die erneute Bereitschaft signalisieren
  - sobald möglich, bestätigt CPU den Interrupt (*Interrupt Acknowledge*) und startet eine zugehörige Behandlungsroutine

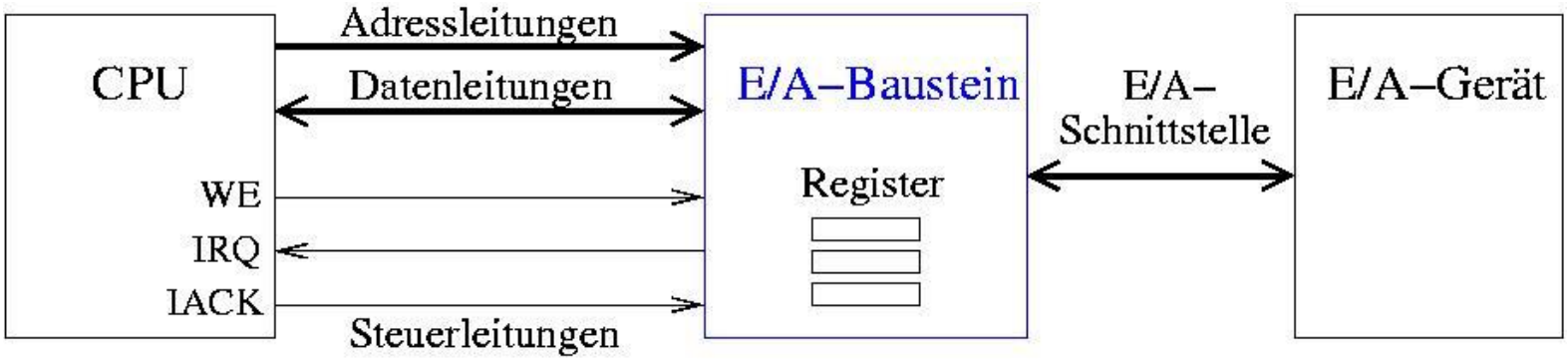
Ein-/Ausgabe Interrupt-Zyklus

- Unterbrechungen erlauben eine schnelle Reaktion der CPU auf E/A-Ereignisse
  - allgemeiner **Ablauf** bei einer Unterbrechung:
  - Unterbrechung nur **nach** der Ausführung einer Instruktion möglich
  - fast alle Prozessoren bieten die Möglichkeit, Unterbrechungen zu verbieten (d.h. zu **maskieren**) bzw. wieder zu gestatten
  - verschiedene **Prioritätsstufen** bei mehreren E/A-Geräten sinnvoll



Kopplung der E/A-Geräten

- Kopplung von CPU und E/A-Geräten erfolgt über spezielle **E/A-Bausteine** am Systembus:
  - Auswahl eines E/A-Bausteins über Adressleitungen
  - Datentransfer von/zu E/A-Baustein über Datenleitungen
  - Steuerleitungen z.B. für Richtungsauswahl, Unterbrechungsanforderung und Bestätigung einer Unterbrechung
  - zur Kommunikation mit CPU bietet E/A-Bausteine einige interne Register an



Register im E/A Baustein

- ein E/A-Baustein verfügt über drei Arten interner Register:
  - **Kontrollregister**
    - zur Initialisierung und Parameterwahl durch CPU
  - **Datenregister**
    - zur Zwischenpufferung von einzulesenden oder auszugebenden Daten (nötig, da E/A-Geräte zumeist langsamer als CPU sind und zudem asynchron zur CPU arbeiten)
    - einige Bausteine mit getrennten *Data-In* und *Data-Out* Register
  - **Statusregister**
    - zum Austausch von Statusinformationen zwischen E/A-Baustein und CPU (z.B. Verfügbarkeit eines neuen Eingabewertes, oder Ausgabegerät hat Zeichen aus Datenregister gelesen)
    - E/A-Baustein setzt/löscht entsprechende Bits im Statusregister selbständig
    - bei Verwendung der Strategie *Polling* wird Statusregister in einer Schleife abgefragt

Kommunikation zwischen CPU und IO-Geräten

- zwei Möglichkeiten für den Zugriff der CPU auf die internen Register der E/A-Bausteine:
  - **speicherbezogene Adressierung** (*Memory-Mapped I/O*):
    - Register sind an bestimme Speicheradressen in den physikalischen Adressraum der CPU eingeblendet
    - E/A-Adressen müssen vom Caching ausgenommen werden!
    - Zugriff mit normalen `load`- und `store`-Befehlen, Zugriffsschutz nur über Speicherverwaltung
  - **separate Ein-/Ausgabeadressen (IO-Ports)**
    - separater, oft kleiner E/A-Adressraum
    - spezielle, i.a. privilegierte Befehle (z.B. `in`, `out`) für Lesen und Schreiben im E/A-Adressraum
    - zusätzliche Steuerleitung IO/Memory zur Selektion von Speicher-oder E/A-Adressraum

Techniken der Datenübertragung

- zwischen Sender (z.B. Rechner) und Empfänger (z.B. Peripherie-gerät) existieren Leitungen für die Übertragung von **Daten** und **Steuersignalen**
- drei verschiedene Übertragungsprotokolle:

1) **Open-Loop** Datenübertagung
   - keine Rückmeldung bei der Datenannahme

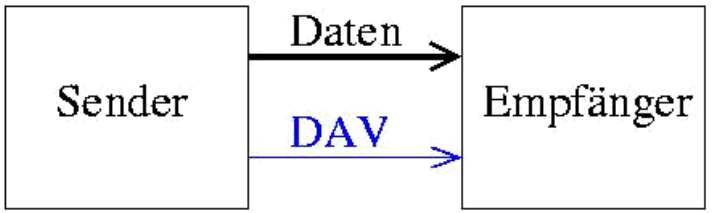
2) **Closed-Loop Datenübertagung**
   - Quittierung der Datenannahme über Steuersignal
   - auch als *Handshaking* bezeichnet

3) **Fully-Interlocked** Datenübertragung
   - Quittierung sowohl der Datenannahme als auch aller Steuersignale
   - auch als *Fully-Interlocked Handshaking* bezeichnet

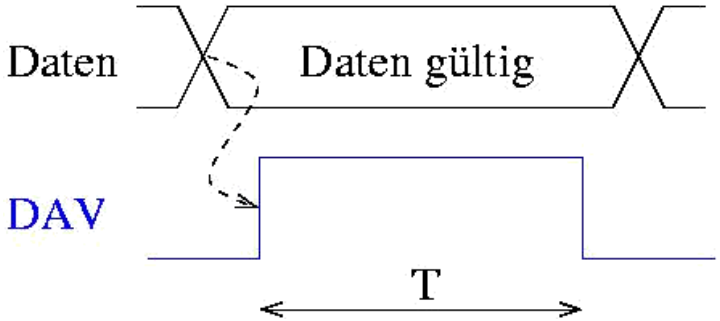
Open-Loop Datenübertragung

- eine Steuerleitung:
  DAV = *Data Valid*
- Sender legt ein Datum auf Datenleitungen und setzt für eine Zeitdauer T das Signal **DAV = 1**
- Empfänger muss die Daten übernehmen, solange **DAV = 1** ist
- Sender und Empfänger müssen Zeitdauer T vereinbart haben

Signale:



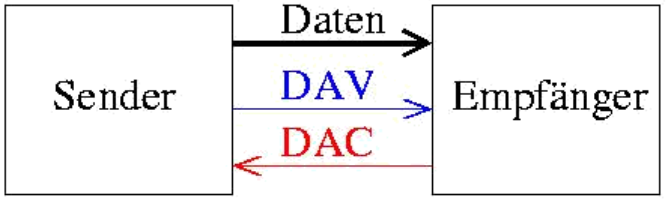
Zeitdiagramm:



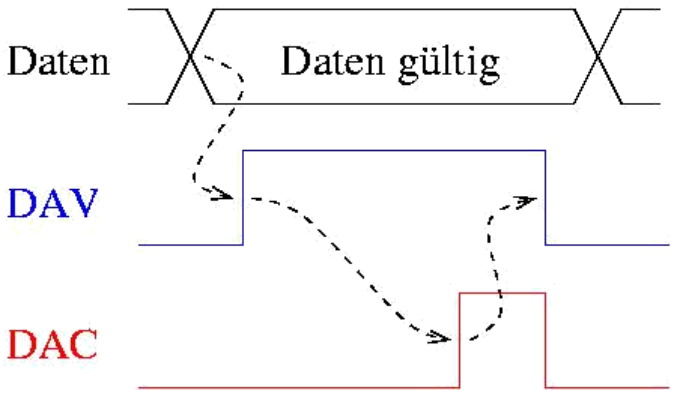
Closed-Loop Datenübertragung

- zwei Steuerleitungen:
  DAV = *Data Valid*
  DAC = *Data Accepted*
- nach Empfang eines Datums bestätigt Empfänger dies mit **DAC = 1**
- Sender nimmt Daten von Datenleitungen zurück, sobald er **DAC = 1** empfangen hat
- **einfache Flusskontrolle** (langsamer Empfänger kann Sender anhalten)
- ggf. Abbruch nach Wartezeit (**Timeout**)

Signale:



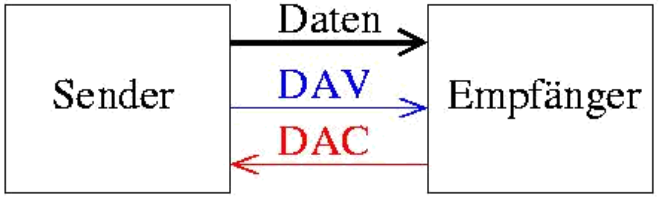
Zeitdiagramm:



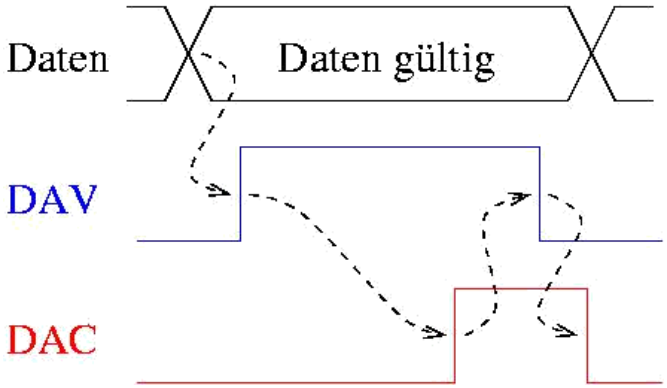
Fully-Interlocked Datenübertragung

- zwei Steuerleitungen, wie bei Closed-Loop
- Unterschiede zu Closed-Loop:
  - Empfänger hält **DAC = 1**, bis der Sender **DAV = 0** setzt
  - Empfänger bestätigt somit, die Deaktivierung des Signals DAV gesehen zu haben
  - Sender gibt erst neue Daten aus, wenn **DAC = 0** vorliegt
- **verbesserte Flußkontrolle**
  (Empfänger kann auch nach Empfang von Daten den Sender aufhalten)

Signale:



Zeitdiagramm:



Punkt-zu-Punkt Schnittstellen

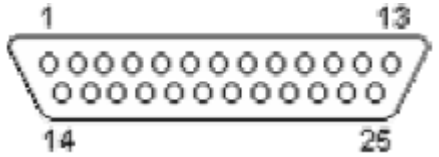
- Bei Verwendung einer **Punkt-zu-Punkt Schnittstelle** lässt sich nur ein einzelnes Peripheriegerät an eine Schnittstelle anschließen
- Gegenteil: **Bussysteme**
- Punkt-zu-Punkt Schnittstelle
  - **unidirektional**
    (es gibt einen Sender und einen Empfänger)
  - **bidirektional**
    (beide Seiten können als Sender oder Empfänger arbeiten)
- Beispiele für standardisierte Punkt-zu-Punkt Schnittstellen
  1) parallele Schnittstelle (IEEE 1284)
  2) serielle Schnittstelle (RS-232)

Parallele Schnittstelle

- **8-Bit** Datenworte werden **parallel** vom Sender zum Empfänger übertragen
- früher: *Centronics* Schnittstelle
  – nur unidirektionaler Betrieb: Ausgabe von Daten an ein Peripheriegerät
  – Transferrate bis zu 150 KByte/s
- heute: IEEE 1284 Schnittstelle, unterstützt u.a. 3 Modi:
  – *Enhanced Parallel Port* (**EPP**) gestattet einen bidirektionalen Betrieb: bei Anschluss eines Druckers kann dieser z.B. auch Statusmeldungen (kein Papier, Toner leer, ...) übertragen
  – *Extended Capability Port* (**ECP**) leistet zusätzlich Datenkomprimierung und Auswahl mehrerer logischer Geräte
  – *Standard Parallel Port* (**SPP**) definiert einen unidirektionalen Betrieb (kompatibel zur *Centronics* Schnittstelle)
  – mit EPP ist eine Transferrate bis zu ca. 2 MByte/s möglich
- Stellt heute eine einfache, preiswerte Schnittstelle für Geräte mit relativ hoher Datenübertragungsrate dar:
  – Drucker und Scanner
  – Zip-Laufwerk, …
- wird jedoch von anderen Schnittstellen (z.B. **USB**) verdrängt

Parallele Schnittstelle: Steckerverbindung

- **25-polige** Steckerverbindung (Sub-D):



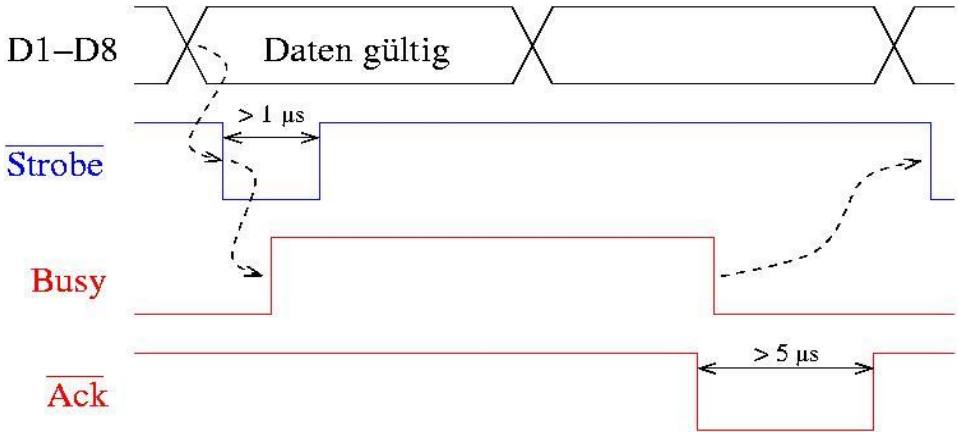
- Bedeutung der Steckerpins bei Einsatz im **SPP-Modus** :

1	/Strobe		A	Daten gültig	10	/Ack	E	Bestätigung
2	D1	(LSB)	A	Datenbit 0	11	Busy	E	Endgerät beschäftigt
3	D2		A	Datenbit 1	12	PE	E	Papierende
4	D3		A	...	13	Select	E	Endgerät offline/online
5	D4		A		14	/Auto Feed	A	autom. Zeilenvorschub
6	D5		A		15	/Error	E	Fehler
7	D6		A		16	/Reset	A	Endgerät zurücksetzen
8	D7		A	Datenbit 6	17	/Select In	A	Auswahlleitung
9	D8	(MSB)	A	Datenbit 7	18-25	GND		Abschirmung

  - einige Steuersignale (mit / markiert) mit sind „*low active*“
  - z.T. auch 36-polige Steckerverbindung üblich
- **EPP-Modus** :
  - Belegung teilweise geändert
  - Datenleitungen nun **bidirektional**, zusätzliche Steuerleitung zur **Richtungsangabe** (/Write = 0 : Ausgabe, /Write = 1: Eingabe)
  - EPP stellt eine Multifunktionsschnittstelle dar (Adressübertragung, benutzerdefinierbare statt druckerspezifische Signale)

Parallele Schnittstelle: Zeitdiagramm

- Zeitdiagramm für die Ausgabe eines Bytes zum Peripheriegerät:



  - TTL-Pegel, d.h. +5V für logisch 1, 0V für logisch 0
  - Closed-Loop Datenübertragung mit Steuersignalen /Strobe und Busy; das Signal /ACK liefert eine zusätzliche Bestätigung und kann zur Unterbrechungsanforderung verwendet werden

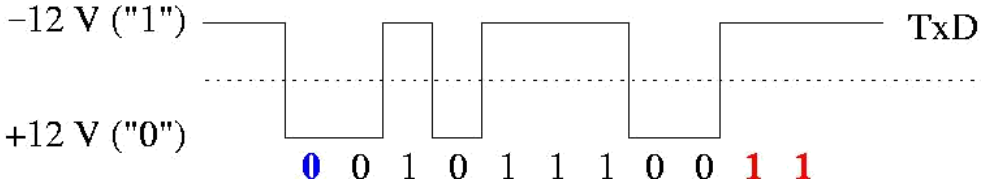
- Übertragungsprotokoll bei **SPP** in Software:
  - CPU muss Status des Empfängers (Busy bzw. Ack) mittels Statusregister handeln
- Übertragungsprotokoll bei **EPP** in Hardware:
  - CPU schreibt/liest lediglich ein Byte in/aus Datenregister

Serielle Schnittstelle

- Serielle, **asynchrone** Punkt-zu-Punkt Verbindung
  - jedes zu übertragene Zeichen wird mit Start-, Stop- und ggf. Paritätsbit in ein Paket verpackt, das asynchron (ungetaktet) und bitseriell über eine Leitung übertragen wird
  - mit minimal 3 Leitungen bereits funktionsfähig (⇨ geringe Kabelkosten)
  - verbindet **DTE** (*Data Terminal Equipment*, z.B. PC) mit **DCE** (*Data Communication Equipment*, z.B. Modem),
  - kann auch **DTE** mit **DTE** verbinden
  - Sender und Empfänger müssen sich für die Übertragung eines jeden Pakets erneut synchronisieren
- Standard: **RS-232** (*Electronic Industries Association*, 1969)
  - auch als V.24 bezeichnet (CCITT)
- Enge Verwandte: **RS-422** und **RS-485** **(Bus!)**
  - Signalleitung mit symmetrisch um null liegenden Spannungen
  - Wesentlich größere Leitungslängen möglich

Serielle Schnittstelle (2)

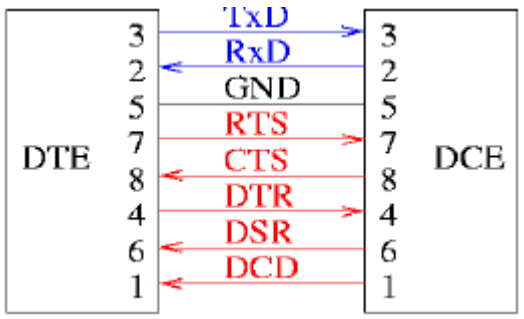
- **Spannungspegel** auf den Leitungen:
  - Logisch 1: < 3V (typisch 12V, maximal 15V)
  - Logisch 0: > +3V (typisch +12V, maximal +15V)
  - hoher Störabstand ⇨ maximale Leitungslänge: ca. 20m
- serielle Datenübertragung
  - Ruhezustand: TxD = 1
  - **1 Startbit** (TxD = 0)
  - 5,6,7 oder 8 Datenbits (niedrigstwertiges Datenbit zuerst)
  - **1** oder **2 Stopbits** (TxD = 1)
- **Paritätsbit:**
  - **gerade Parität** (*Even Parity*): Summe der 1-Datenbits ist gerade
  - **ungerade Parität (***Odd Parity*): Summe der 1-Datenbits ist ungerade
- Beispiel: serielle Übertragung des Byte 0x3A = 001110102
  - Auf der Leitung: **0**01011100**1**$_2$ (ohne Parität)
  - 1 Bit dauert ca. 1 ms bei 9600 Bps (**Baud**)



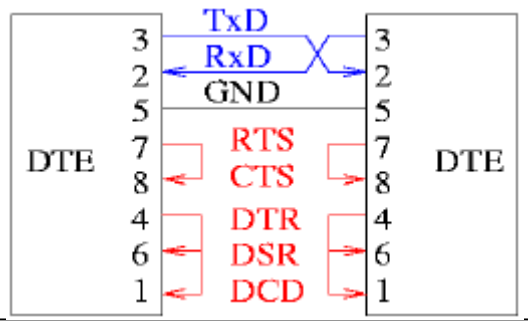
Serielle Schnittstelle: Flußkontrolle

- keine Empfangsbestätigung ⇨ Open-Loop Datenübertragung
- Anpassung unterschiedlicher Geschwindigkeiten von DTE und DCE mittels **Flußkontrolle**:
  - entweder mittels **Hardware** (*RTS/CTS protocol*):
    - wenn **DCE** keine Daten mehr aufnehmen kann, erhält DTE auf der Eingangsleitung **CTS** den Pegel 0
    - wenn **DTE** keine Daten mehr aufnehmen kann, wechselt DTE den Pegel auf der Ausgangsleitung **RTS** von 1 auf 0
  - oder mittels **Software** (*Xon/Xoff protocol*):
    - Senden des Steuerzeichens *Xoff* (Ctrl-S, ASCII 19h) stoppt den Datentransport in die entgegengesetzte Richtung
    - Senden des Steuerzeichens *Xon* (Ctrl-Q, ASCII 17h) hebt den Stop wieder auf
    - beide Steuerzeichen dürfen nicht im Datenstrom vorkommen!

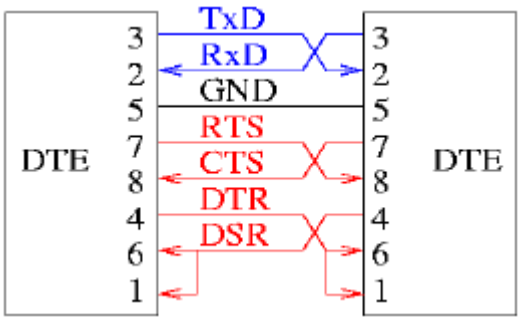
Modem straight (RTS/CTS)



Null Modem (Xon/Xoff)



Null Modem (RTS/CTS)



Direct Memory Access

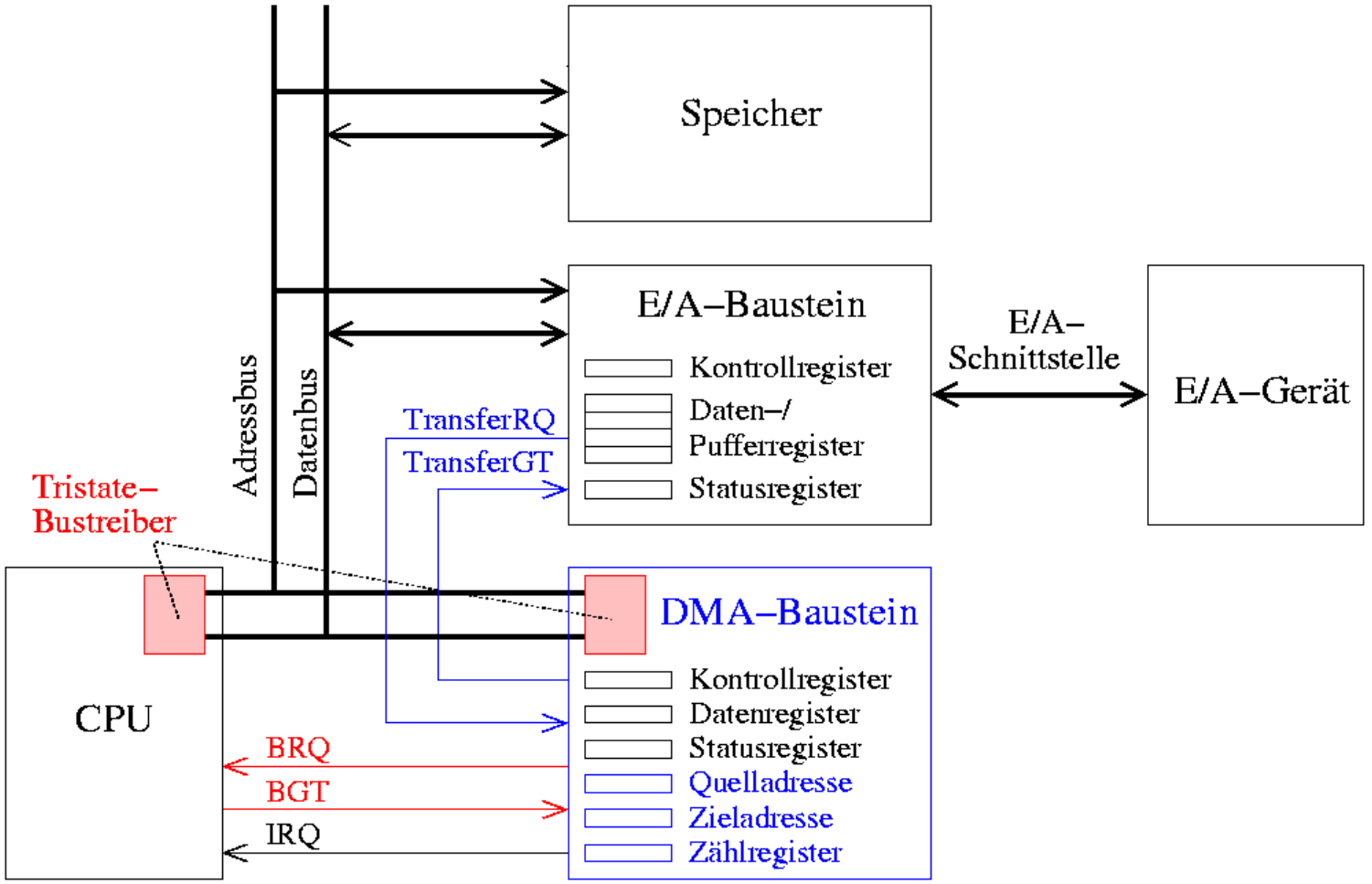
- oft werden lange Datenströme aus dem Speicher zur Peripherie ausgegeben, bzw. von der Peripherie in den Speicher eingelesen
  (⇨ Ÿunnötige Belastung der CPU mit trivialen Aufgaben: Inkrementieren der Adresse, Zählen der Datenworte, Statusabfrage des E/A-Bausteins)

- Idee: ein zusätzlicher **DMA-Baustein** (*Direct Memory Access*) führt nach Initialisierung durch die CPU den Speichertransfer selbständig durch
  (⇨ ŸCPU kann sich anspruchsvolleren Tätigkeiten widmen!)

- ein DMA-Baustein enthält
  - ein **Quelladressregister** und ein **Zieladressregister**, in dem die Start-und Zieladresse des zu transferierenden Datenblocks eingetragen werden
  - ein **Zählregister**, das mit der Anzahl der zu transferierenden Bytes bzw. Datenworten initialisiert werden muss
  - ein **Kontrollregister**, um z.B. Richtung oder Arbeitsmodus festzulegen

Direct Memory Access: Aufbau

- prinzipieller Aufbau eines Systems mit DMA-Bausteins:



Direct Memory Access: Ablauf

- prinzipieller Ablauf eines DMA-Transfers vom E/A-Baustein in den Speicher:
  1) CPU initialisiert E/A-Baustein (Kontrollregister) und DMA-Baustein (mit Startadresse, Zieladresse und Anzahl an Datenworten)
  2) E/A-Baustein setzt das Signal **TransferRQ** (*Transfer Request*), sobald Daten vorhanden sind
  3) DMA-Baustein fordert mit dem Signal **BRQ** (*Bus Request*) den Bus von der CPU an
  4) CPU deaktiviert eigene Bustreiber und setzt Signal **BGT** (*Bus Grant*)
  5) DMA-Baustein zeigt dem E/A-Baustein den Beginn des Datentransfers durch das Signal **TransferGT** (*Transfer Grant*) an
  6) DMA-Baustein transferiert Daten vom E/A-Baustein in den Speicher
  7) DMA-Baustein gibt durch Rücknahme von **BRQ** den Bus wieder frei
  8) DMA-Baustein kann der CPU durch Interrupt das Ende des Transfers signalisieren

Direct Memory Access: Arbeitsmodi

- Arbeitsmodi eines DMA-Bausteins:
  - **Burst Modus**:
    DMA-Baustein erhält den Systembus für den kompletten Transfer eines Blocks aus vielen Datenworten
    - E/A-Baustein benötigt internen Pufferspeicher
    - Vorteil: sehr hohe Transferrate
    - Nachteil: CPU wird für lange Zeit am Speicher-/Buszugriff gehindert
  - **Cycle Stealing**:
    DMA-Buszyklen und CPU-Buszyklen werden gemischt
    - ein fester Anteil an Buszyklen (z.B. jeder zweite Buszyklus) wird vom DMA-Baustein der CPU „gestohlen“
    - DMA kann zusätzlich alle von der CPU nicht benötigten Buszyklen nutzen (z.B. während der Ausführungsphase von Register/Register-Befehlen)
    - **Vorteil**: CPU wird nur geringfügig behindert
    - **Nachteil**: geringere Transferrate

Direct Memory Access: Diskussion

- Anmerkungen:
  - im Kontrollregister des DMA-Bausteins kann die **Übertragungs-art** gewählt werden:
    - beim **direkten** DMA-Transfer erfolgt die Datenübertragung direkt zwischen E/A-Baustein und Speicher (Ÿ1 Buszyklus je Wort)
    - beim **indirekten** DMA-Transfer wird jedes Wort im Datenregister des DMA-Bausteins zwischengespeichert (Ÿ2 Buszyklen je Wort)
  - Statusregister des DMA-Bausteins enthält Informationen über Zustand (frei/belegt), Blockende und Fehler
  - auch der Datentransfer zwischen zwei E/A-Bausteinen oder von Speicher zu Speicher kann über DMA realisiert werden
  - oft mehrere **DMA-Kanäle** je DMA-Baustein (Ÿmehrere Transfers gleichzeitig in Bearbeitung)
  - DMA-Baustein ist bei heutigen PCs im **Chipset** integriert

Bussysteme

- Verbindung von **mehreren** Komponenten über identische Transport- und Steuerleitungen
  (Vorteil: bedeutend kleinerer Verdrahtungsaufwand als bei Punkt-zu-Punkt Verbindungen)
- Bussysteme waren früher hersteller- und systemspezifisch, sind heute jedoch weitgehend **standardisiert**
  - E/A-Geräte bzw. E/A-Karten können unabhängig von Hersteller und Rechnersystem entwickelt werden:
    - größerer Absatzmarkt, geringere Kosten
    - Beispiele: Steckkarten für PCI-Bus, Festplatten
  - Standardisierung umfasst
    - Signale und Spannungspegel
    - zeitliches uns elektrisches Verhalten der Bussignale
    - Steckverbinder und Pinbelegungen

Arten von Bussystemen

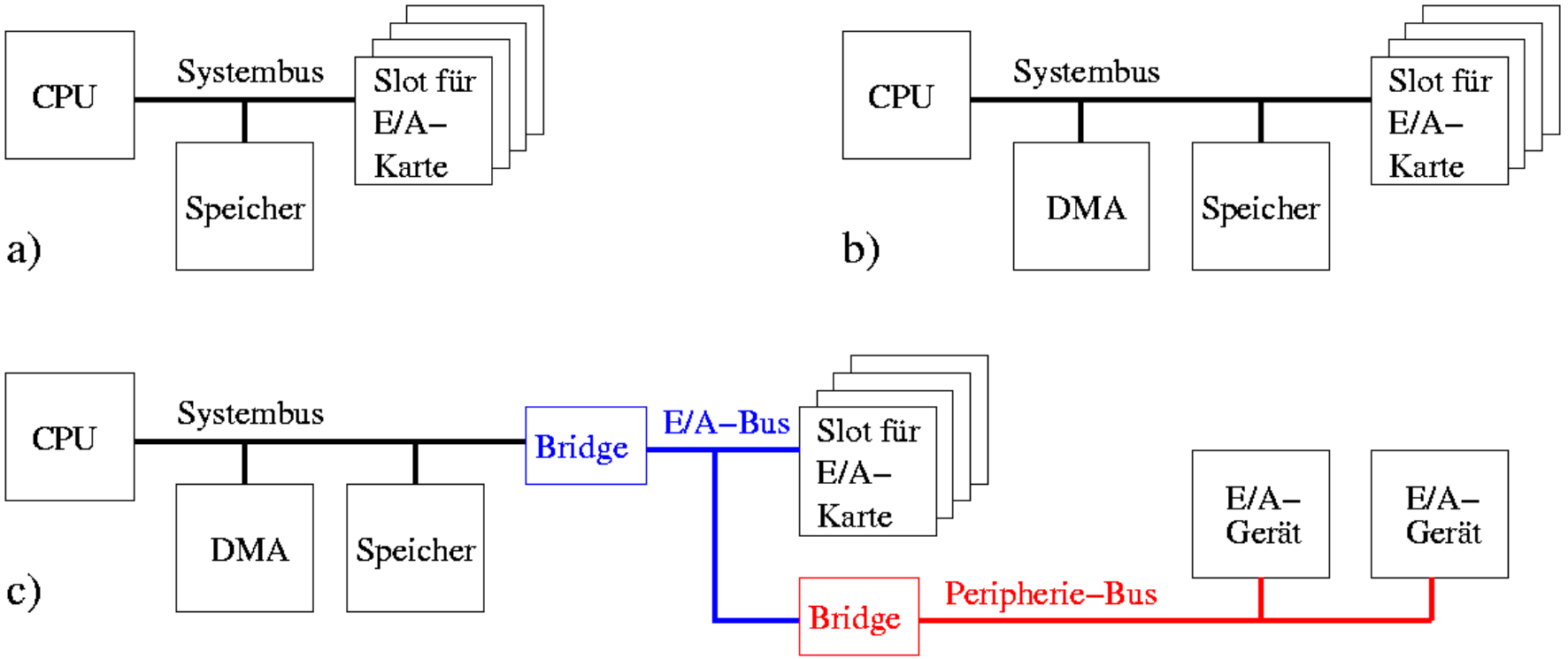
- verschiedene Arten von Bussystemen in einem Rechner:
  - **prozessorinterne Busse**
    - arbeiten i.a. mit CPU-Taktfrequenz
    - verbinden z.B. Registersatz, Arithmetik-Einheiten und L1-Datencache
  - **Systembus**
    - verbindet CPU mit schnellen Systemkomponenten
    - auch als *Front-Side-Bus* bezeichnet
    - Taktfrequenz typisch 100 bis 200 MHz
  - **Ein-Ausgabebus** (intern, auch *Local Bus*)
    - Bus für E/A-Erweiterungen der Hauptplatine, z.B. PCI-Bus
  - **Peripherie-Bus** (extern)
    - zum Anschluss mehrerer Peripheriegeräte an eine Busschnittstelle, z.B. USB, SCSI-Bus

Merkmale von Bussystemen

- unterscheidende Merkmale von Bussystemen:
  - **Breite** von **Datenbus** und **Adressbus**
  - **Multiplexing** von Adress- und Datenleitungen
  - maximale **Datentransferrate** (in MByte/s)
  - **Taktung** (synchron/asynchron) und ggf. **Bustaktfrequenz**
  - **maximale Anzahl** von möglichen Buskomponenten bzw. Bussteckplätzen (*Slots*)
  - Art der **Busarbitrierung**
  - Art der möglichen **Buszyklen**
  - Realisierung von **Interrupts**
  - Unterstützung von **DMA**
  - **physikalische Größen**: Spannungspegel, Steckverbinder
- zwei verschiedene Bussysteme können mittels einer **Bridge** gekoppelt werden:
  - Anpassung z.B. von Datenbusbreite (z.B. 32 ⇨ 16 Bit), Taktfrequenz, Buszyklen

Systembus-Architektur

- einige mögliche Systembus-Architekturen:



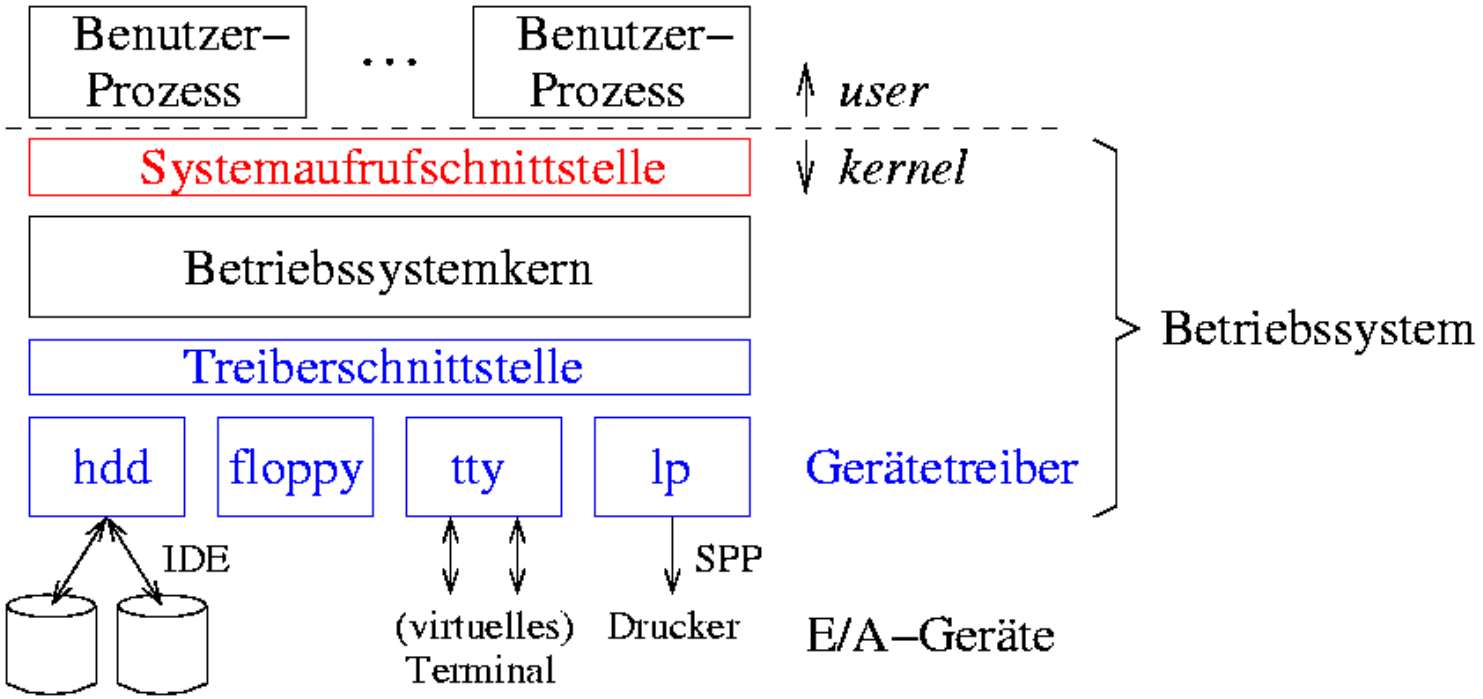
Gerätetreiber

- ein **Gerätetreiber** stellt eine Softwareschicht zwischen dem Betriebssystemkern und dem E/A-Gerät dar:
  - ein Gerätetreiber ist ein **Softwaremodul**, das geräteabhängigen Code zur Steuerung von E/A-Geräten eines Typs enthält

    (Betriebssystemkern bleibt unabhängig von E/A-Geräten!)
  - es sind mehrere Gerätetreiber erforderlich (z.B. für Festplatten, RS232, EPP, USB), die zum Kern hinzu gebunden werden
  - einheitliche **Treiberschnittstelle** zum Betriebssystem (Implementierungs-details bleiben verborgen, z.B. Adressen und Inhalt der E/A-Register)
  - Betriebssystem bietet eine einheitliche (d.h. geräteunabhängige) **System-aufrufschnittstelle** zum Benutzerprozess
  - Beispiel:
    Der Unix-Systemaufruf `read(fd,buf,n)` liest `n` Bytes von einem beliebigen E/A-Gerät `fd` in einen Puffer `buf`, wobei das ausführende Programm keine Kenntnis von der Art des E/A-Gerätes haben muss.

Gerätetreiber: Schichten

- Schichten eines Betriebssystems zwischen Benutzerprozess und E/A-Gerät (vereinfacht):



- Benutzerprozess und Gerätetreiber arbeiten in unterschiedlichen Speicherbereichen (*user* / *kernel space*)

Gerätetreiber: Aufgaben

- Aufgaben eines Gerätetreibers (Auswahl):
  - **Initialisierung** und **Überwachung** der E/A-Gerätes durch geeignete Programmierung der E/A-Register
  - Bereitstellen einer Schnittstelle zur Annahme **abstrakter** Anfragen an ein E/A-Gerät und Umsetzen der abstrakten Anfrage in eine **konkrete** geräteabhängige Form
  - Übernahme/Übergabe und Pufferung von Daten
  - **Zuteilung** von E/A-Geräten an Benutzerprozesse (zur exklusiven oder gemeinsamen Nutzung)
  - Implementierung von Warteschlangen für E/A-Geräte
  - Verwaltung von **Zugriffsrechten**
  - Behandlung von **Unterbrechungsanforderungen**
  - Behandlung von **Fehlermeldungen** des E/A-Gerätes
  - Wahl von Parametern/Strategien zur optimalen Nutzung eines Gerätes

Gerätetreiber: Betriebsarten

- Betriebsarten eines Gerätetreibers:

  1) Polling:

  – CPU wartet aktiv, bis E/A-Gerät bereit ist

  – nach Ausführung jedes Teiltransfers vom/zum E/A-Gerät wartet CPU aktiv, bis der Transfer beendet ist

  2) Interrupt:

  – wenn E/A-Gerät noch nicht bereit ist, schläft Prozess (⇨ Prozesswechsel durch Betriebssystem)

  – E/A-Gerät kann bei Eintritt der Bereitschaft durch Senden einer Unterbrechungsanforderung den schlafenden Prozess aktivieren

  – nach Initiierung jedes Teiltransfers vom/zum E/A-Gerät schläft Prozess (⇨ Prozesswechsel durch Betriebssystem), bis der Transfer beendet ist.

  – Vorteil: keine Blockierung anderer Prozesse, höhere Auslastung der CPU

  – Nachteil: höherer Aufwand (Interruptroutinen, Sicherung der Register, ...)

Gerätetreiber: Ein-/Ausgabearten

- Arten der Ein-/Ausgabe:

1) synchron:

- Systemaufruf zur Ein-/Ausgabe terminiert erst, wenn die E/A-Operation vollständig abgeschlossen ist
- bei Interrupt-Betrieb kann ggf. zwischenzeitlicher Prozesswechsel durch Betriebssystem erfolgen

1) asynchron:

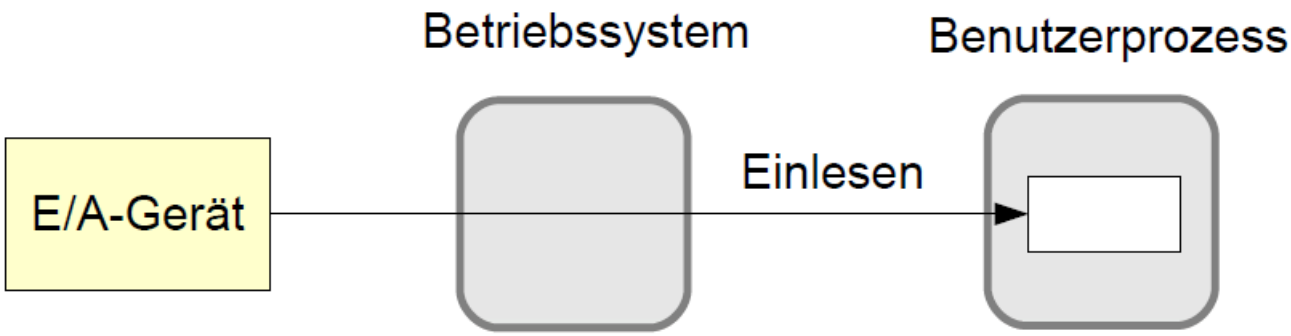
- Systemaufruf initiiert lediglich die Ein-/Ausgabe und gibt Kontrolle an den aufrufenden Prozess zurück
- sinnvoll vor allem bei Ausgabeoperationen!
- durch zusätzlichen Systemaufruf kann sich Benutzerprozess nachträglich mit Ende der E/A-Operation synchronisieren
- Vorteil: Benutzerprozess kann CPU während der E/A-Operation nutzen

Auftragspufferung

- Es macht Sinn, für die eintreffenden E/A-Aufträge beim Treiber eine Warteschlange zu führen: er kann damit bei Beendigung einer Ein/Ausgabeoperation als Teil der Unterbrechungsbehandlung sehr schnell die nächsten Operation starten – was für eine gute Ausnutzung des Gerätes sorgt.
- Ein Puffer wird auch verwendet, um Daten zu speichern, die zwischen Programm und Gerät gerade transferiert werden;
  Ziel:
  - Geschwindigkeitsunterschiede der Datenströme zwischen Erzeuger und Verbraucher zu korrigieren
  - verschiedene Datentransfer-Größen anzupassen
  - Nebenläufigkeit zwischen Verbraucher und Erzeuger

Pufferung bei E/A-Operationen

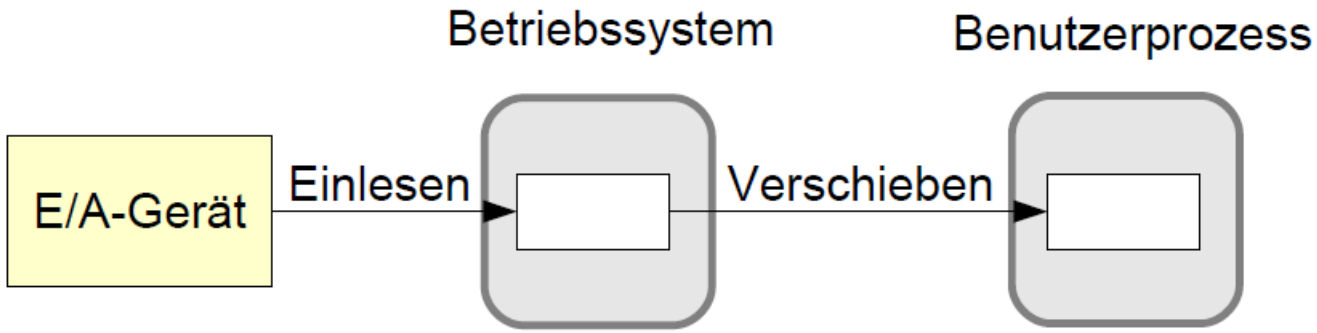
- Probleme ohne Datenpuffer im Betriebssystem:
  - Daten, die eintreffen bevor ***read*** ausgeführt wurde (z.B. von der Tastatur), müssten verloren gehen.
  - Wenn ein Ausgabegerät beschäftigt ist, müsste ***write*** scheitern oder den Prozess blockieren, bis das Gerät wieder bereit ist.
  - Ein Prozess,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



(a) Leseoperation ohne Puffer

E/A-Einzelpuffer

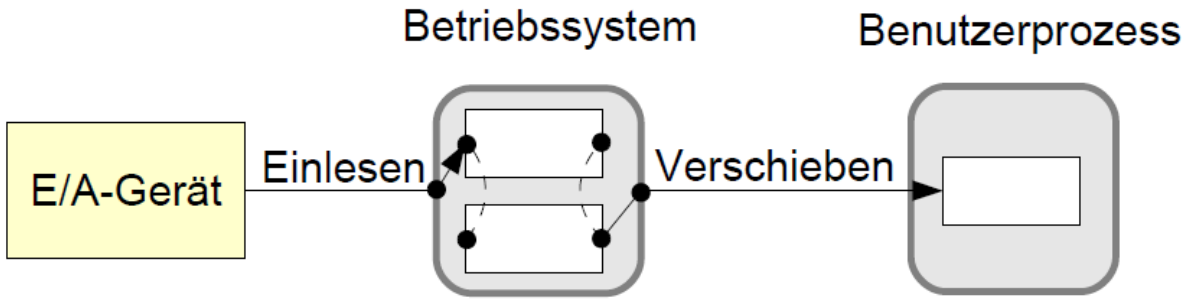
- Einlesen
  - Daten können vom System entgegengenommen werden, auch wenn der Leserprozess noch nicht ***read*** aufgerufen hat.
  - Bei Blockgeräten kann der nächste Block vorausschauend gelesen werden, während der vorherige verarbeitet wird.
  - Prozess kann problemlos ausgelagert werden. DMA erfolgt in Puffer.
- Schreiben
  - Daten werden kopiert. Aufrufer blockiert nicht. Datenpuffer im Benutzeradressraum kann sofort wiederverwendet werden.



(b) Leseoperation mit Einzelpuffer

E/A-Wechselpuffer

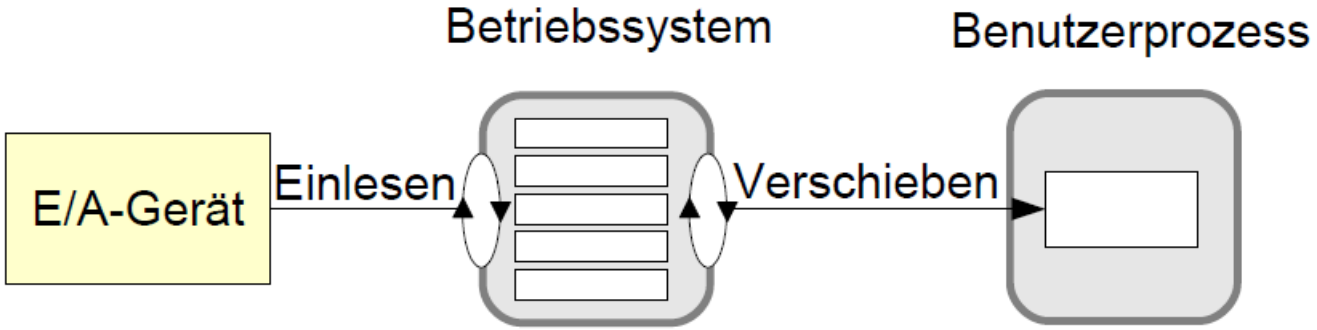
- Einlesen
  - Während Daten vom E/A-Gerät in den einen Puffer transferiert werden, kann der andere Pufferinhalt in den Empfängeradressraum kopiert werden.
- Schreiben
  - Während Daten aus einem Puffer zum E/A-Gerät transferiert werden, kann der andere Puffer bereit mit neuen Daten aus dem Senderadressraum gefüllt werden.



(b) Leseoperation mit Wechselpuffer

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.



(b) Leseoperation mit Ringpuffer

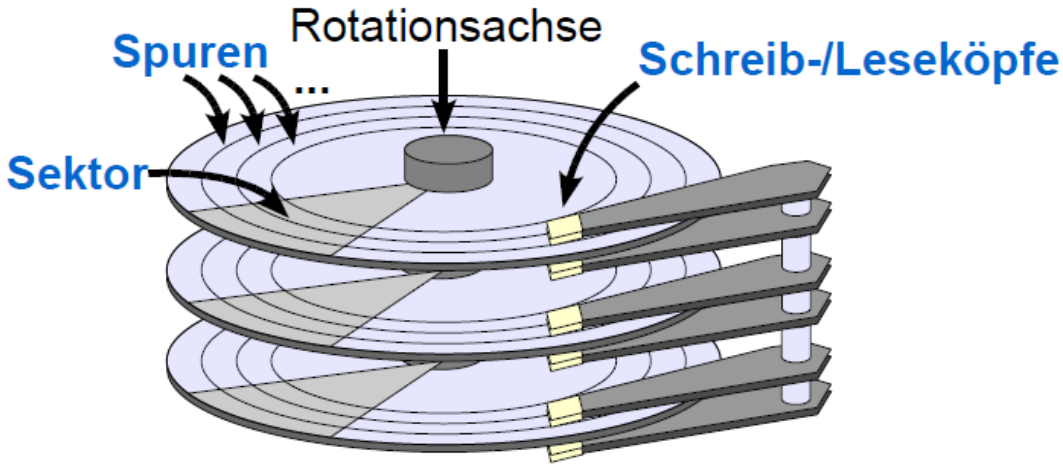
- Anpassung zwischen verschiedenen Datentransfergrößen
  - Beispiel: Fragmentierung / Defragmentierung von Daten

Diskussion der E/A-Pufferung

- **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 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!

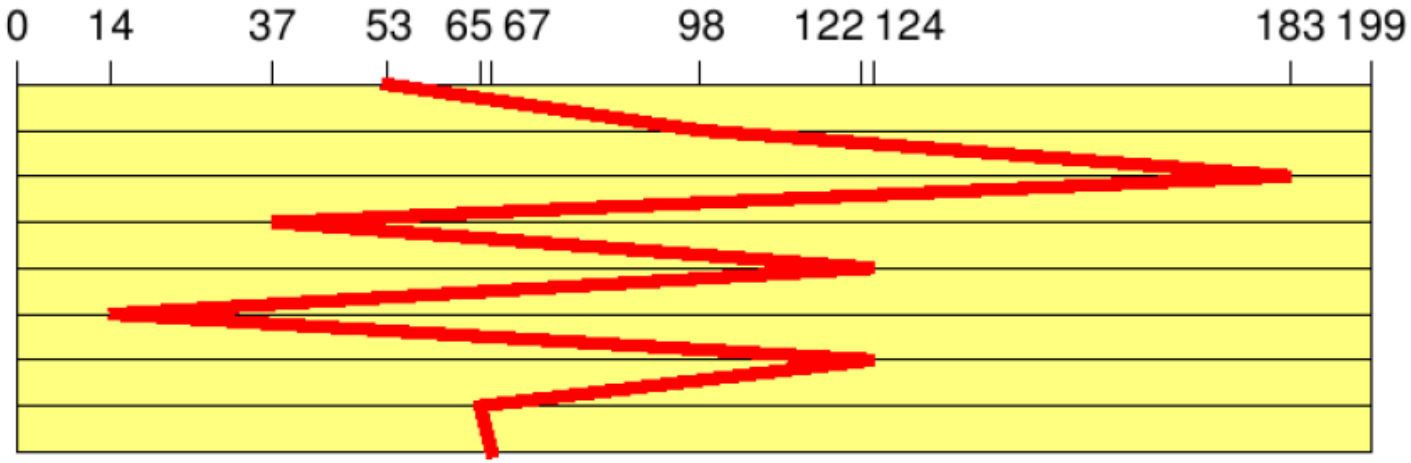
Geräteansteuerung: Bsp. Platte

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



E/A-Scheduling: FIFO

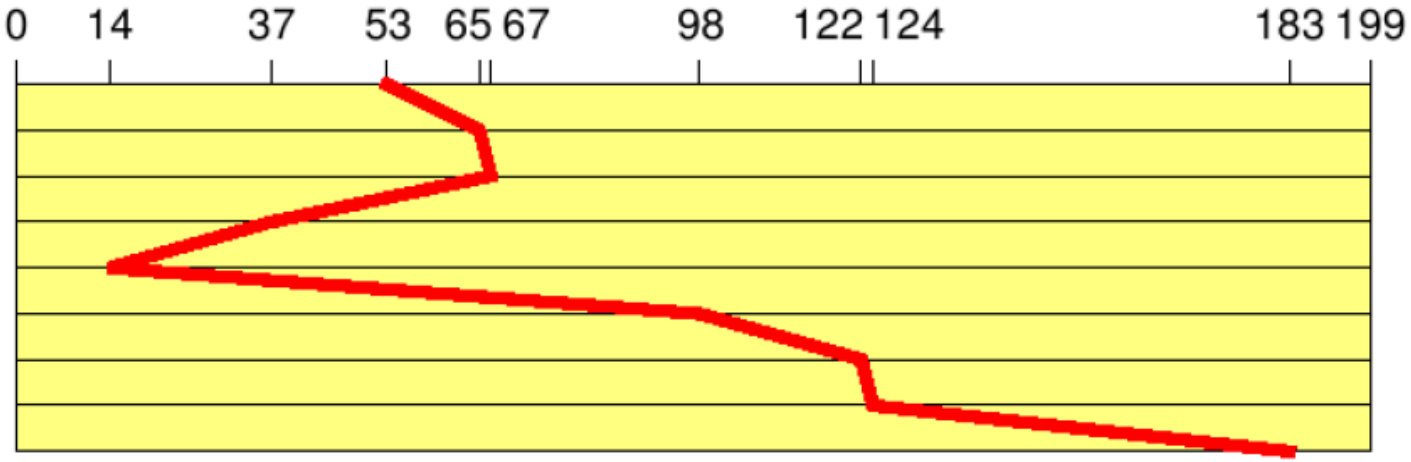
- Bearbeitung gemäß Ankunft des Auftrags (**First In First Out**)
  - Referenzfolge (Folge von Spurnummern):
    98, 183, 37, 122, 14, 124, 65, 67
  - Aktuelle Spur: 53



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

E/A-Scheduling: SSTF

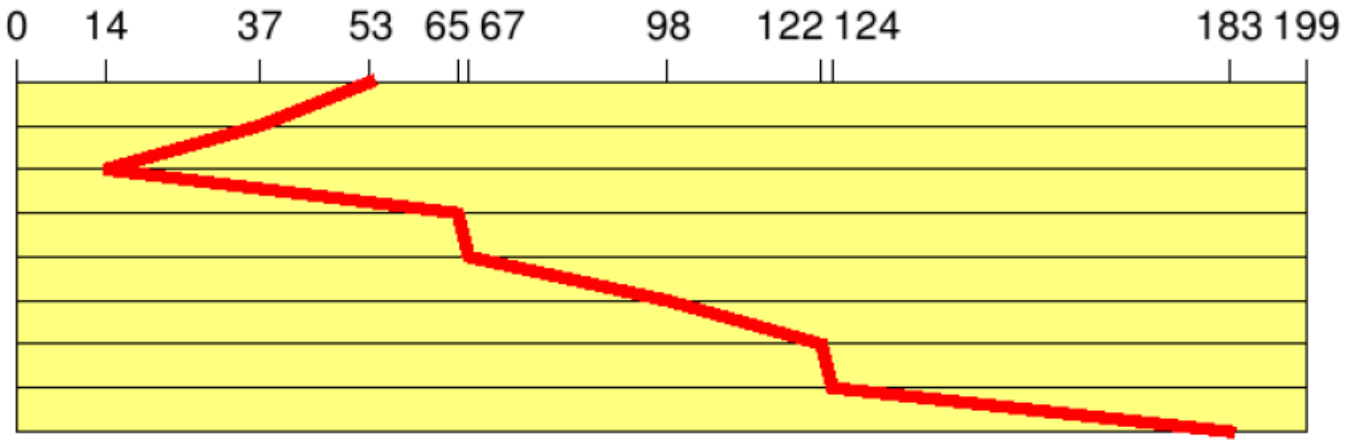
- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (**Shortest Seek Time First**)
  - Gleiche Referenzfolge
  - Annahme: Positionierungszeit proportional zum Spurabstand



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

E/A-Scheduling: Elevator (SCAN)

- 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 und ohne mögliche Aushungerung
- Optimierungen: Circular-SCAN (C-SCAN), C-LOOK

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 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 vier verschiedene Varianten