Universität Bamberg


# GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

## Vorlesung 2: Struktur von Computersystemen und HW/SW-Schnittstellen

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap



# Worum geht's in GRABS eigentlich?

Universität Bamberg

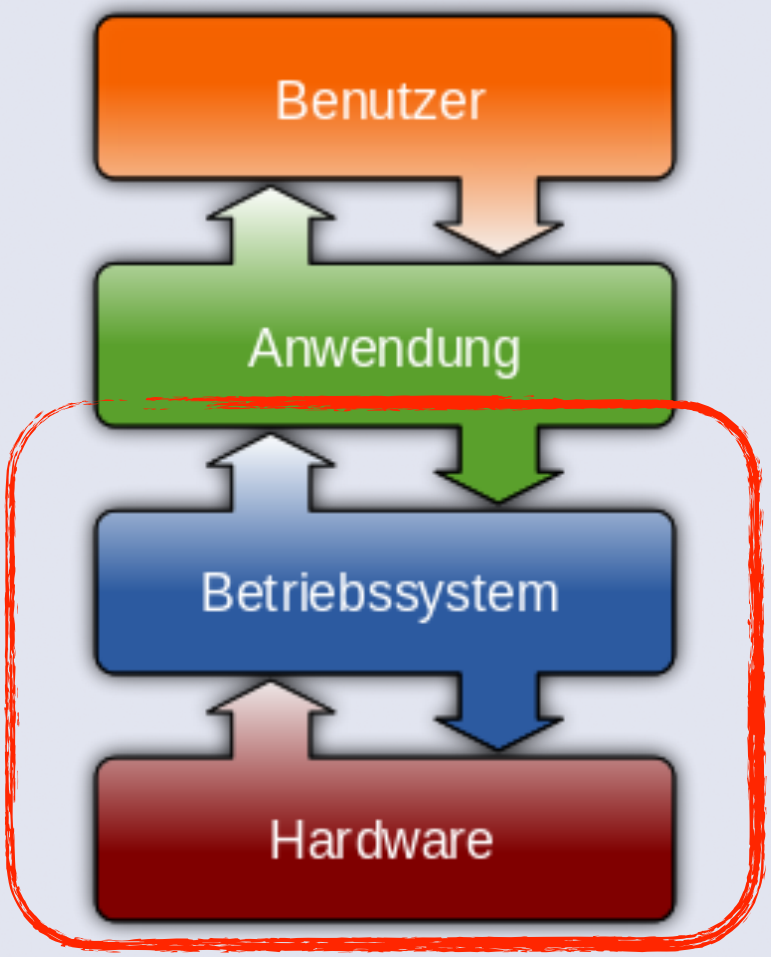
- **Nicht alles in der Informatik ist eine (Web-)Anwendung**
  - Hardware und Systemsoftware sind essentiell für die Funktionalität, aber auch für die Leistung, die Energieaufnahme oder die Sicherheit und Zuverlässigkeit eines Computersystems

- Wir betrachten die untersten Informatik-relevanten Schichten von Computersystemen: **Rechnerarchitektur** und **Betriebssystem**
  - Wir lassen den größten Teil der elektrotechnischen und physikalischen Grundlagen aus (auch aus Zeitgründen)
  - Ein besonderer Schwerpunkt liegt auf auf den **Schnittstellen** und der **Interaktion** zwischen Hardware und Software

- ***Warum soll ich mir das antun, ich will Python programmieren?!?***
  - Besseres Verständnis von Computerverhalten, z.B. beim Debugging
  - Nicht-funktionale Eigenschaften wie z.B. Energieaufnahme und Performance sind auch für Anwendungsentwickler hochrelevant!
  - **…außerdem macht es (mir zumindest) Spaß!**

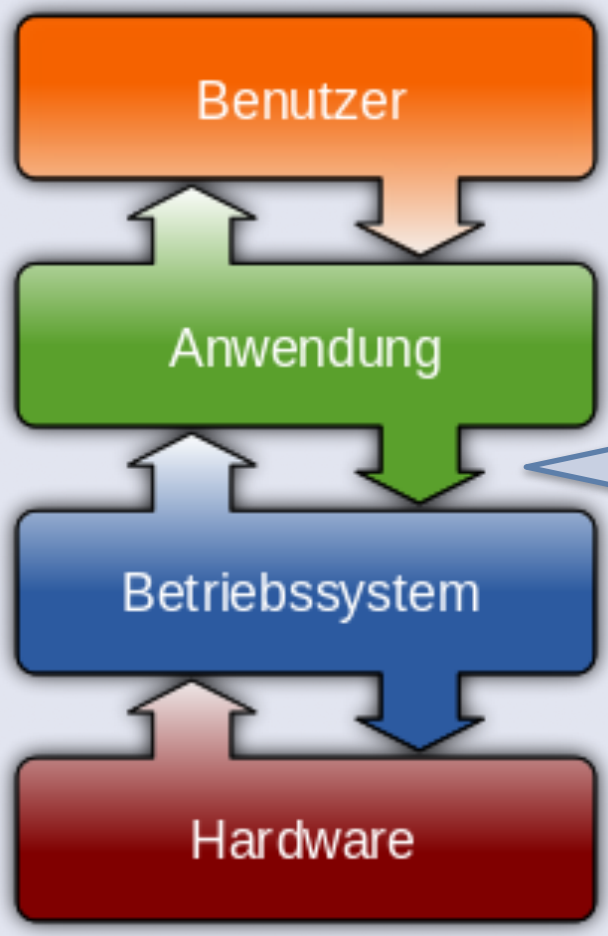
# Hardware/Software-Ebenen





# Inhalte von GRABS: Schnittstelle Anw./BS

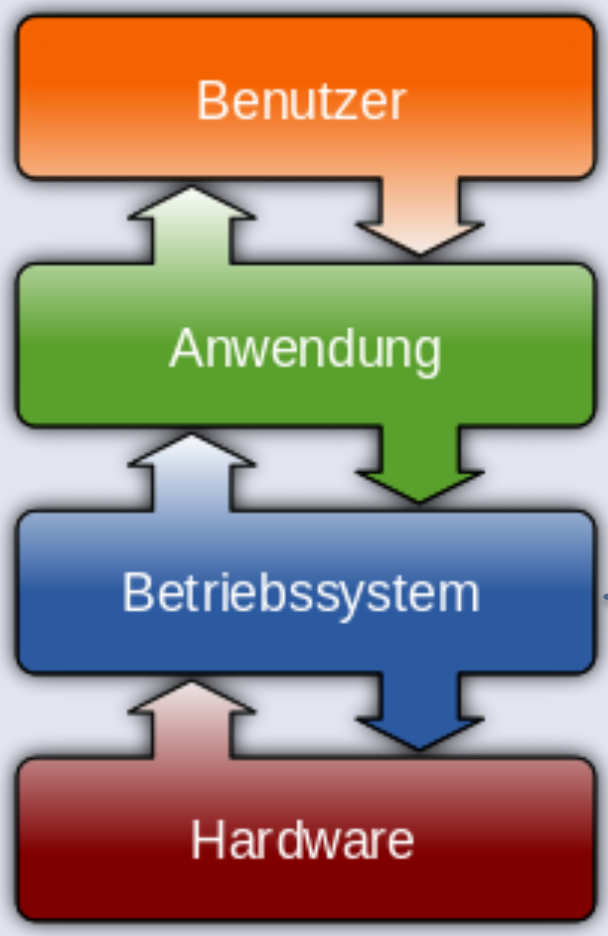





- Übersetzungsvorgang
  - Compiler, Assembler, Linker
- Programme und Prozesse
  - Kontroll- und Datenfluss
  - Speichersegmente
  - Prozesserzeugung
- Darstellung von Daten
  - Zahlendarstellungen
  - Zeichenketten
- Betriebssystemschnittstellen
  - Systemaufrufe
  - Darstellung von Ressourcen
  - Zugriffskontrolle

# Inhalte von GRABS: Betriebssystem

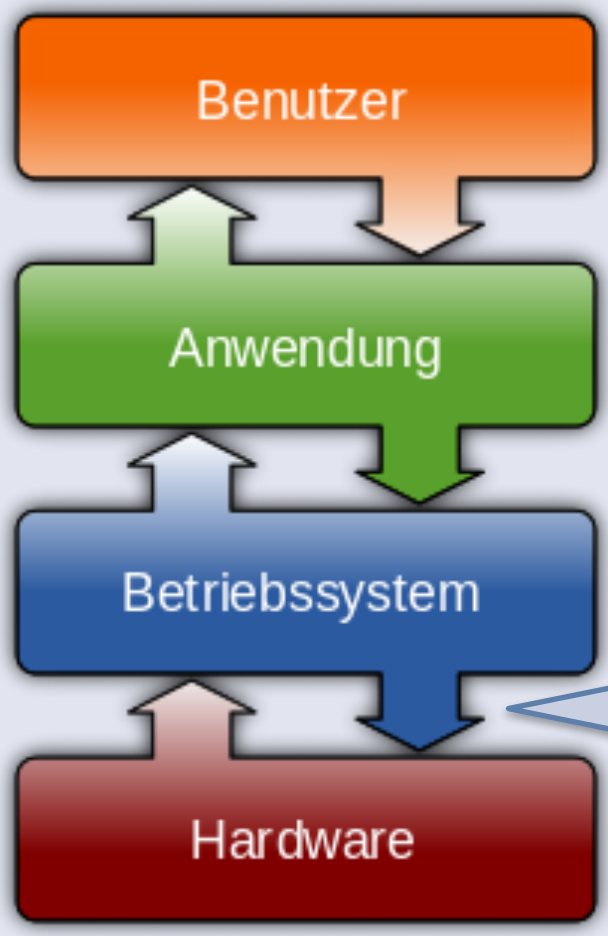





- Programme und Prozesse
  - Prozessdefinition & -zustände
  - Isolation und Synchronisation
- Ressourcen
  - Konflikte und Teilen
- Nebenläufigkeit und Parallelität
  - Scheduling
  - Kritische Abschnitte
  - Semaphore und Deadlocks
- Speicherverwaltung
  - Virtueller Speicher
  - Persistenter Speicher
- Kommunikation
  - Shared memory, IPC

# Inhalte von GRABS: Schnittstelle BS/HW

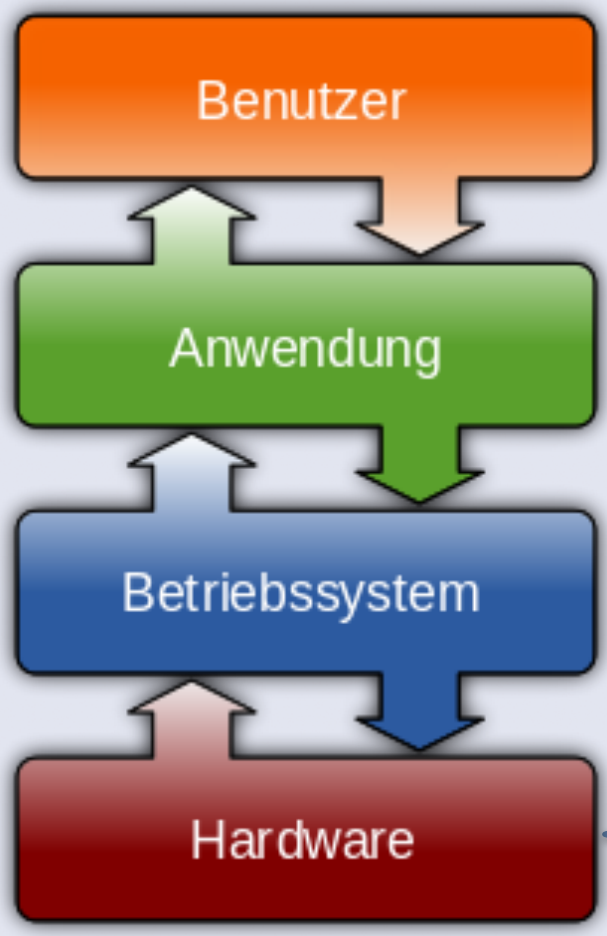





- Zugriffskontrolle
  - Prozessormodi und Privilegien
  - Zugriffsrechte
- Speicher
  - Virtuelle Speicherverwaltung
  - Dateisysteme
- Gerätebehandlung
  - Polling, Interrupts, DMA
  - Adressierung
- Parallele Systeme
  - Vektorrechner
  - Multicore-Systeme
  - GPUs

# Inhalte von GRABS: Hardware

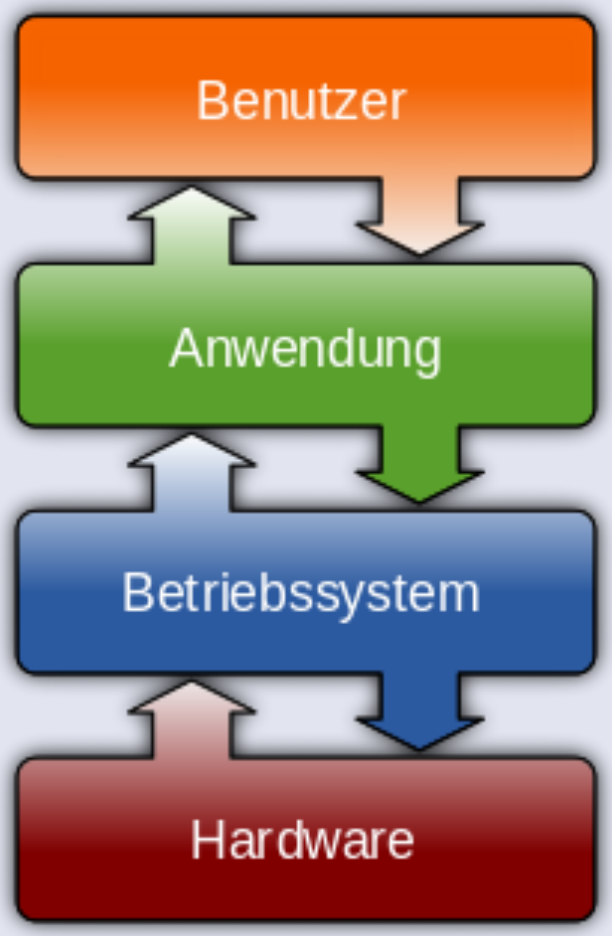


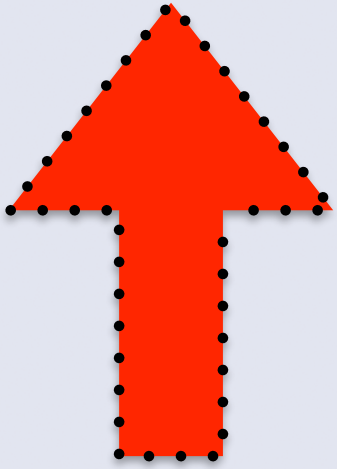



- Digitale Systeme
  - Logik und Arithmetik
  - Endliche Automaten
- Prozessor (CPU)
  - Struktur und Mikroarchitektur
  - Befehlssatz und -darstellung
  - Assemblerprogrammierung
- Speicher
  - Register, Caches
  - DRAM, SRAM, ROM/Flash
- Peripherie
  - Schnittstellen
  - Massenspeicher

# Inhalte von GRABS: Ablauf





- Wir verfolgen generell einen "bottom-to-top"-Ansatz
  - Start bei der Hardware
- Die Vorlesung setzt einen Schwerpunkt auf die **Schnittstellen** zwischen Hardware und Software und deren **Interaktion**
- Diese Vorlesung gibt einen Überblick über das Themengebiet der **Technischen Informatik** und **Betriebssysteme**

# Literatur



Frank Slomka, Michael Glaß
**Grundlagen der Rechnerarchitektur**
**Von der Schaltung zum Prozessor**

PDF kostenlos über Bibliothek ladbar (im Uninetz),
10 gedruckte Exemplare vorhanden

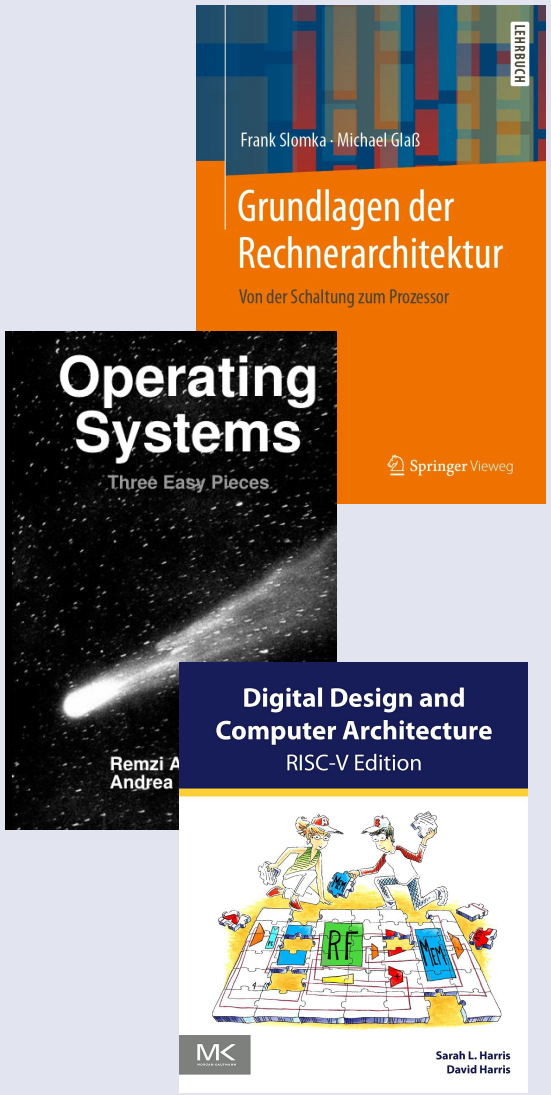
Remzi Arpaci-Dusseau, Andrea Arpaci-Dusseau
**Operating Systems: Three Easy Pieces**

Kostenloser PDF-Download
https://pages.cs.wisc.edu/~remzi/OSTEP/

Ein gedrucktes Exemplar in ERBA-Bibliothek

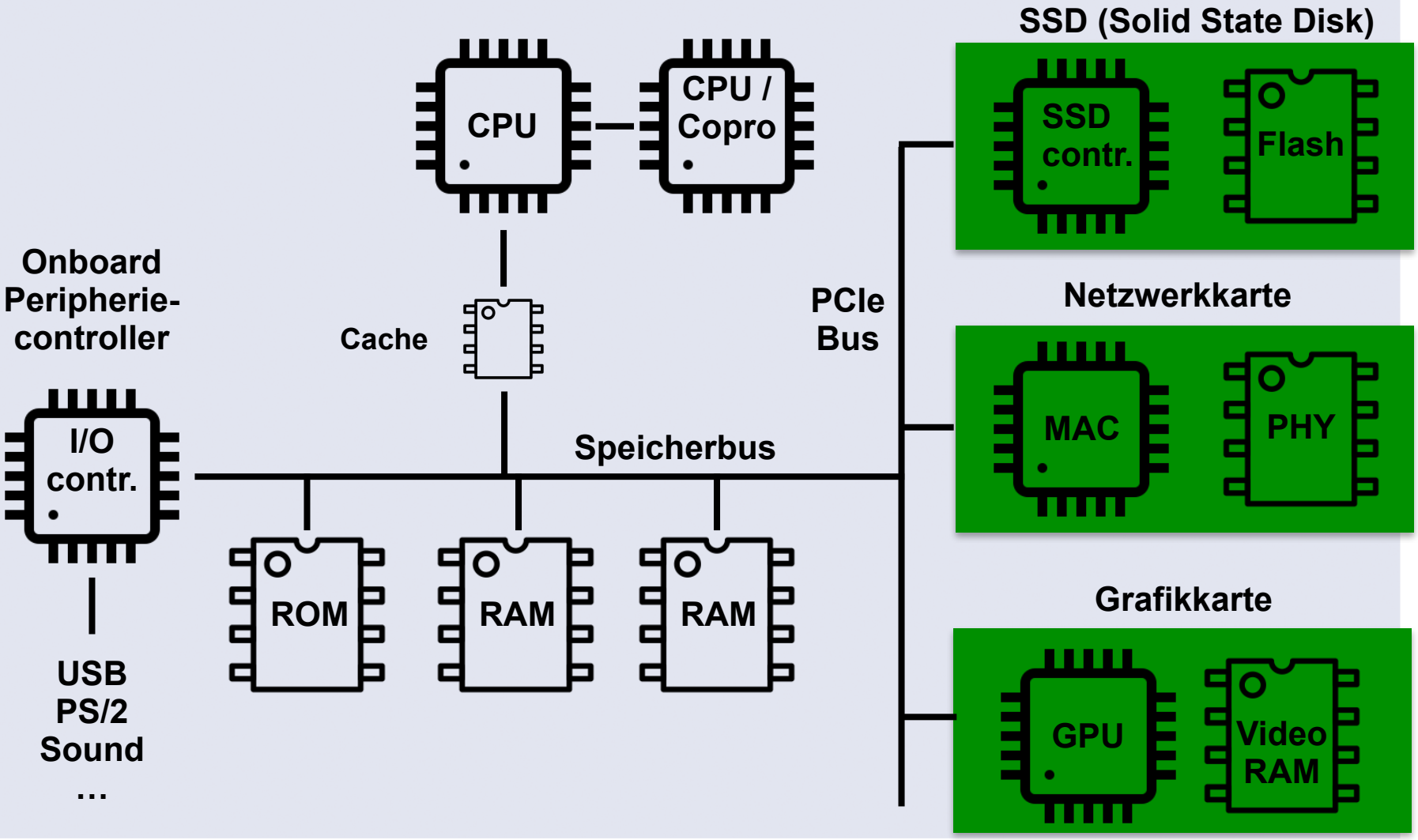
Ergänzend: David Harris, Sarah Harris
**Digital Design and Computer Architecture,**
**RISC-V Edition**

Morgan-Kaufman 2021, ISBN-13: 978-0128200643



# Aufbau eines Computersystems

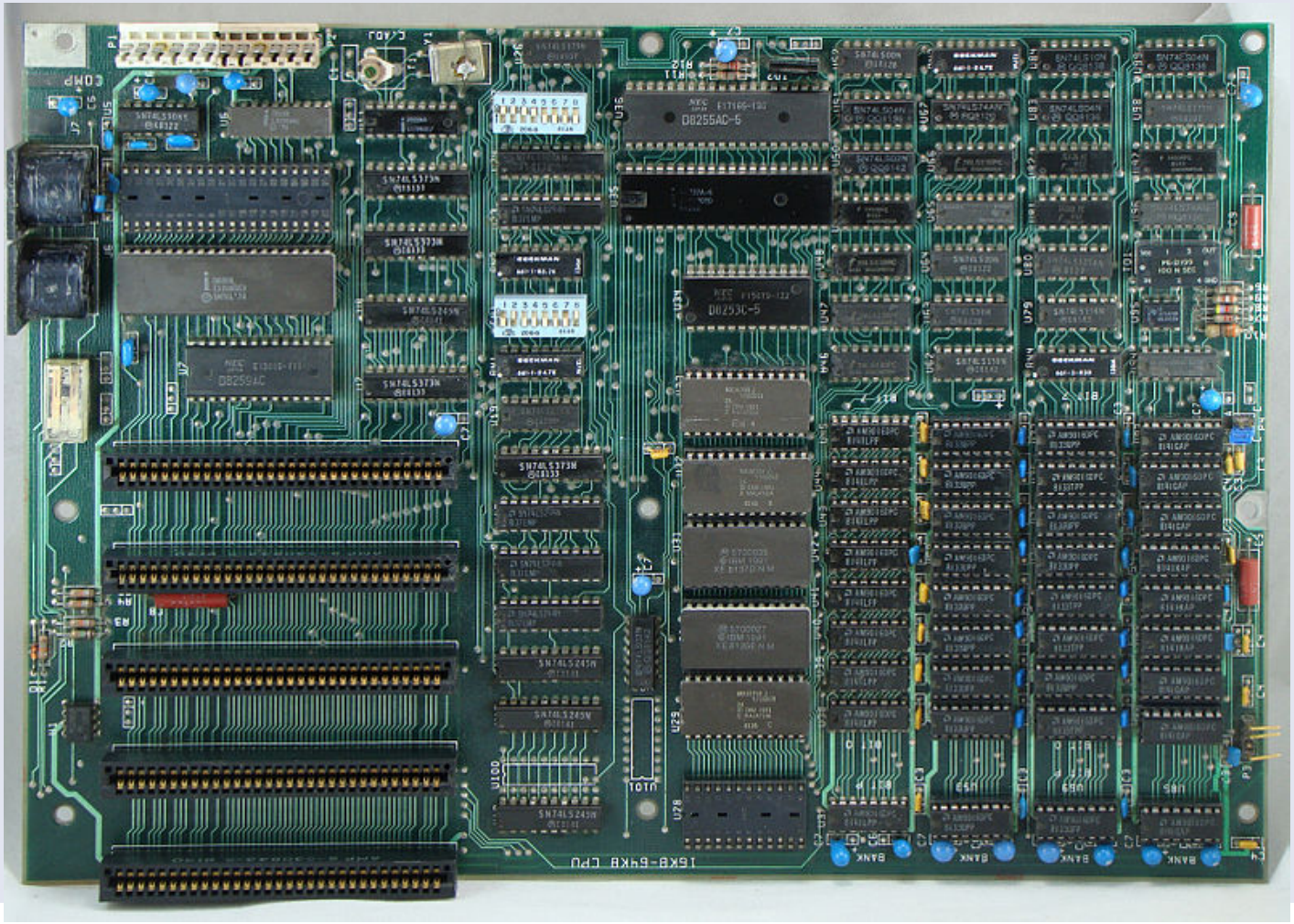
Universität Bamberg



# Aufbau eines Computersystems

Universität Bamberg

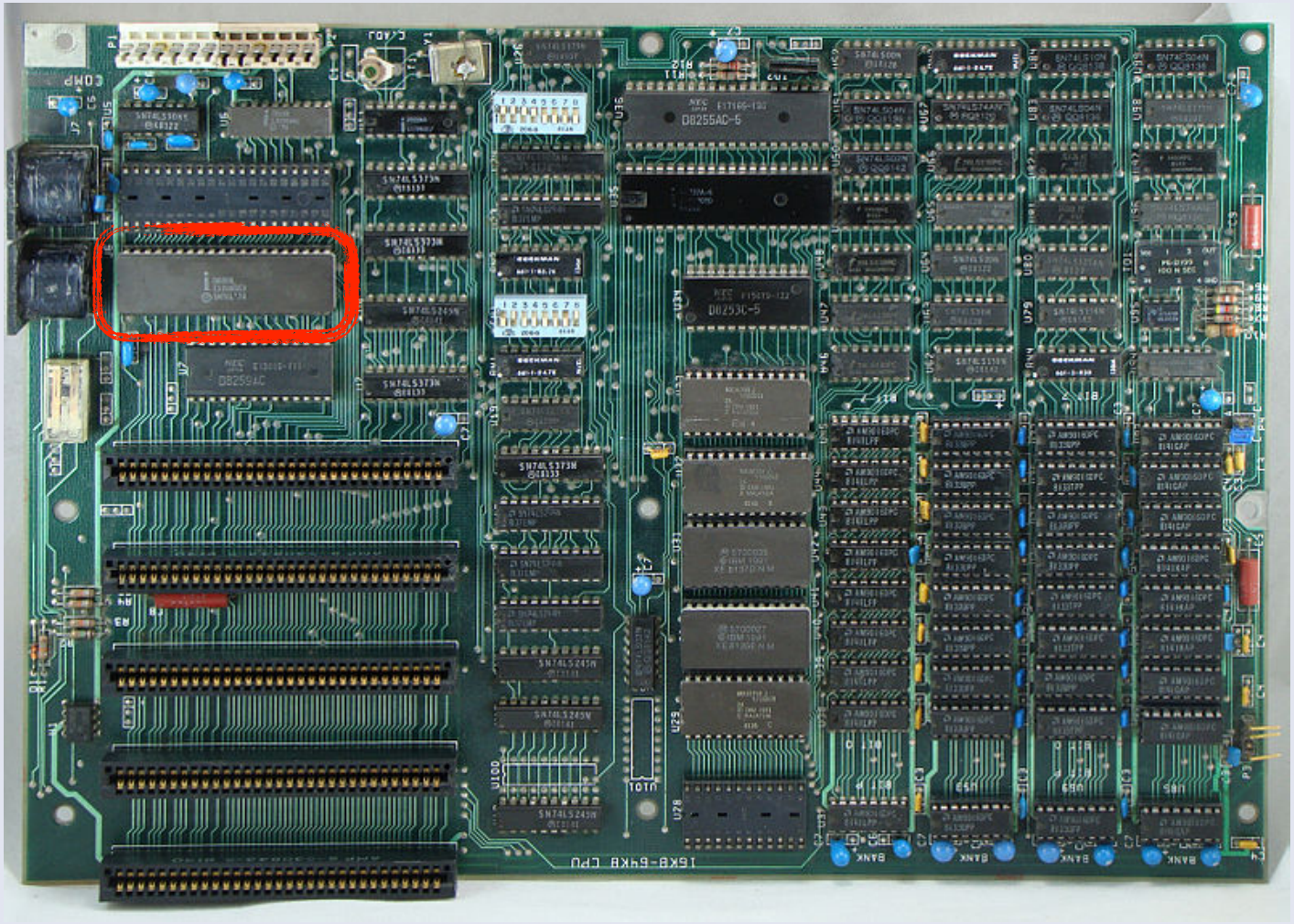
## Das Mainboard des allerersten IBM PC von 1981



# Aufbau eines Computersystems



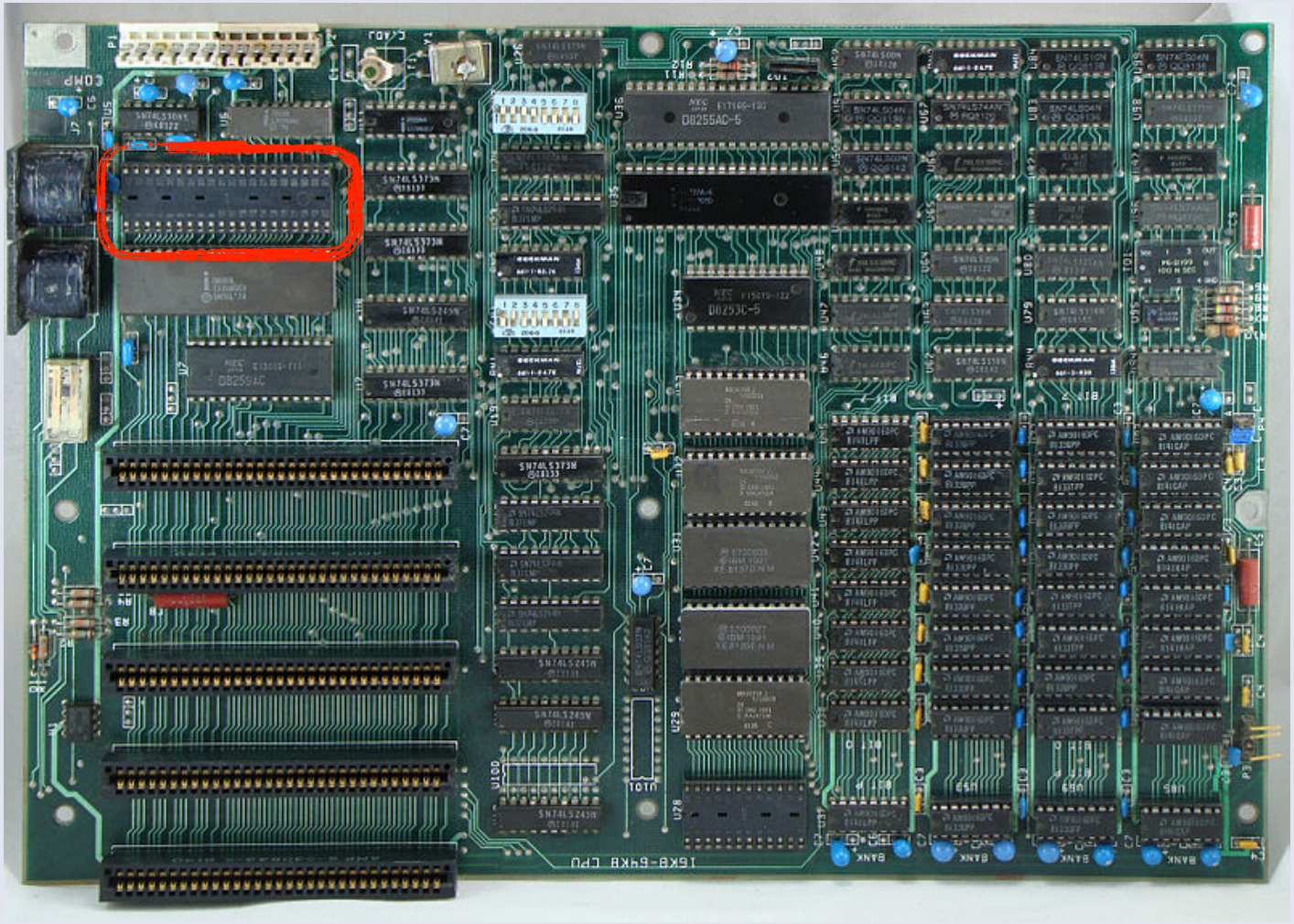
Prozessor (CPU): Intel 8088 4,77 MHz



# Aufbau eines Computersystems

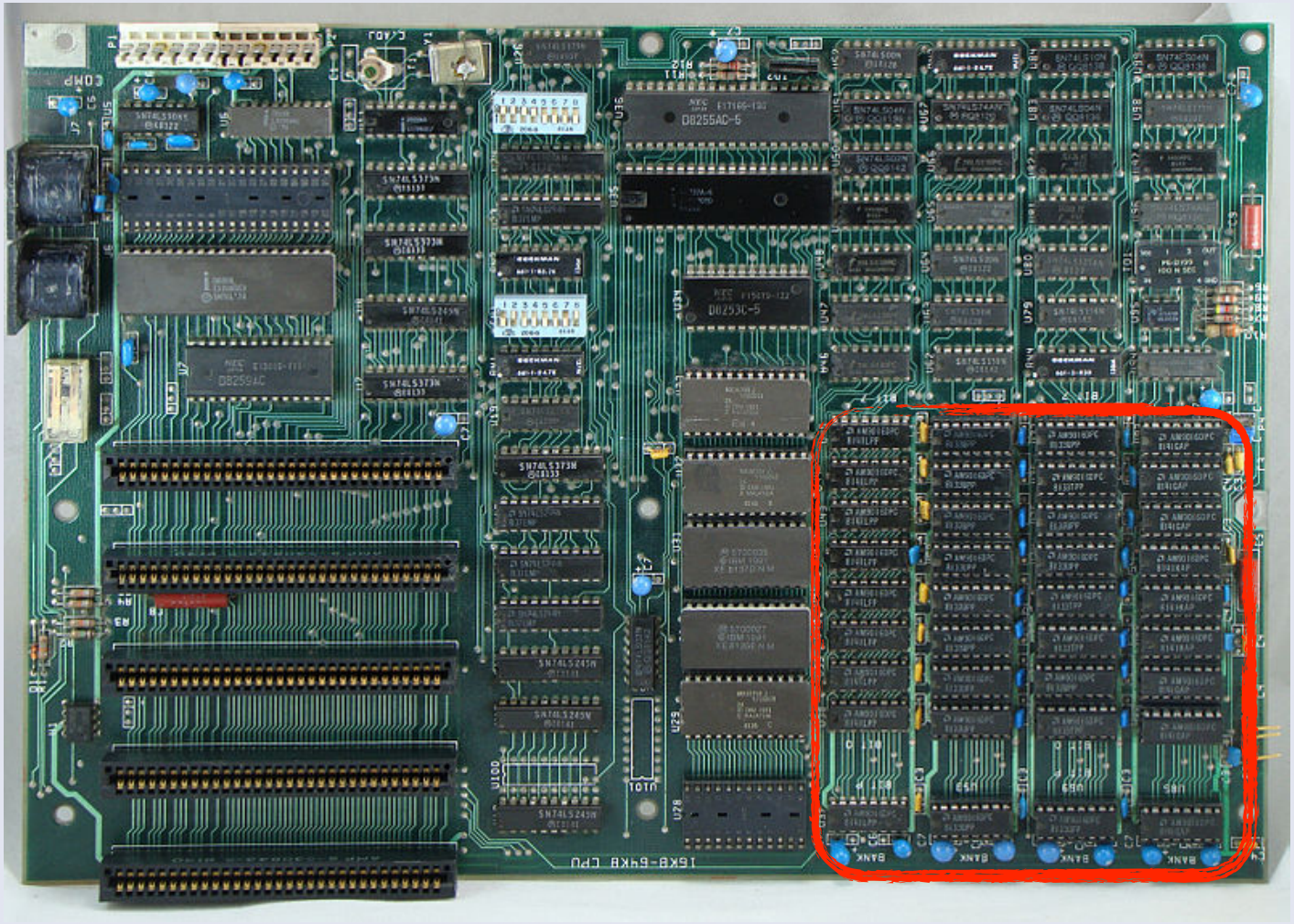


Sockel für Gleitkomma-koprozessor (FPU): Intel 8087



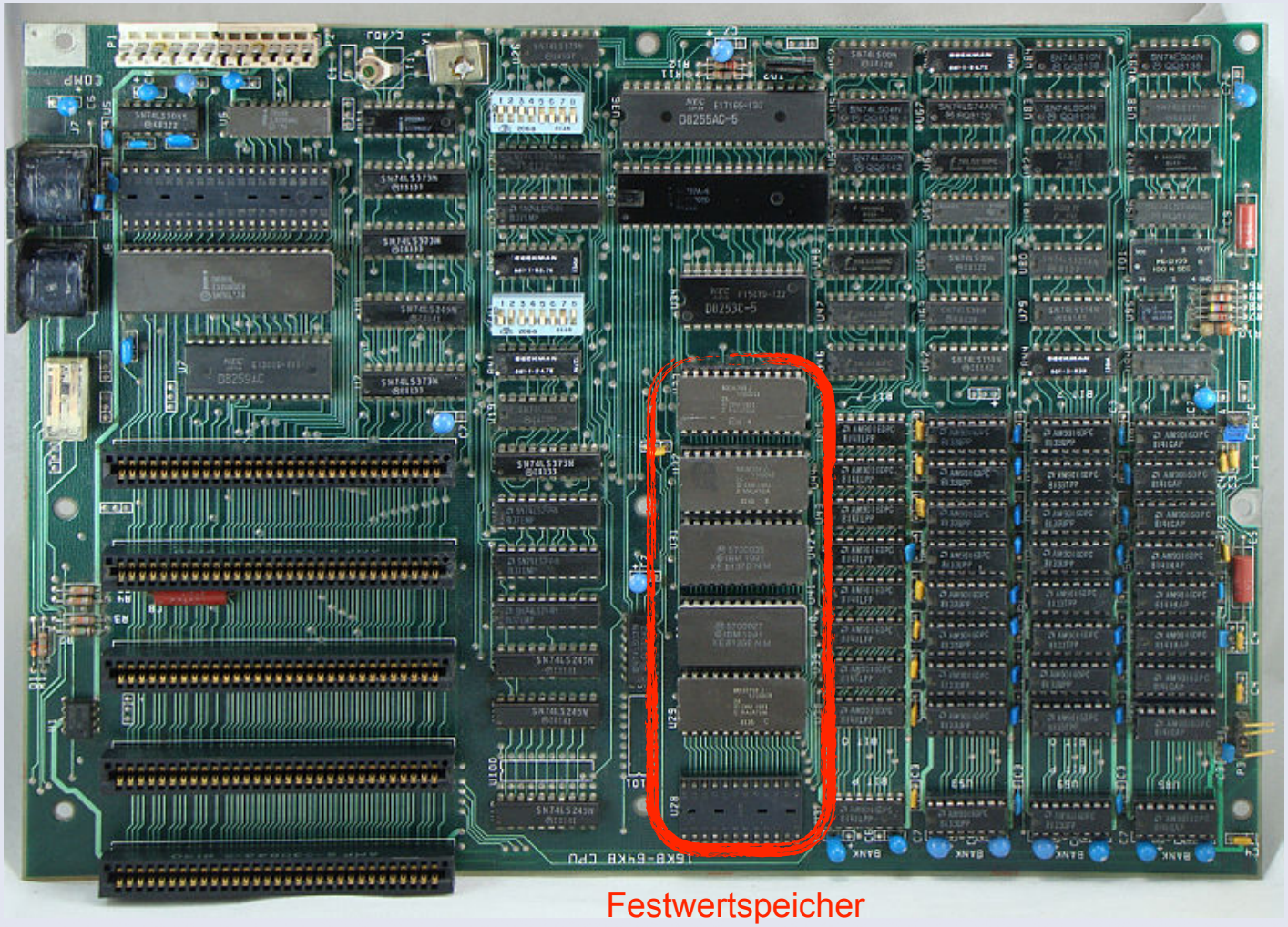
# Aufbau eines Computersystems





# Aufbau eines Computersystems

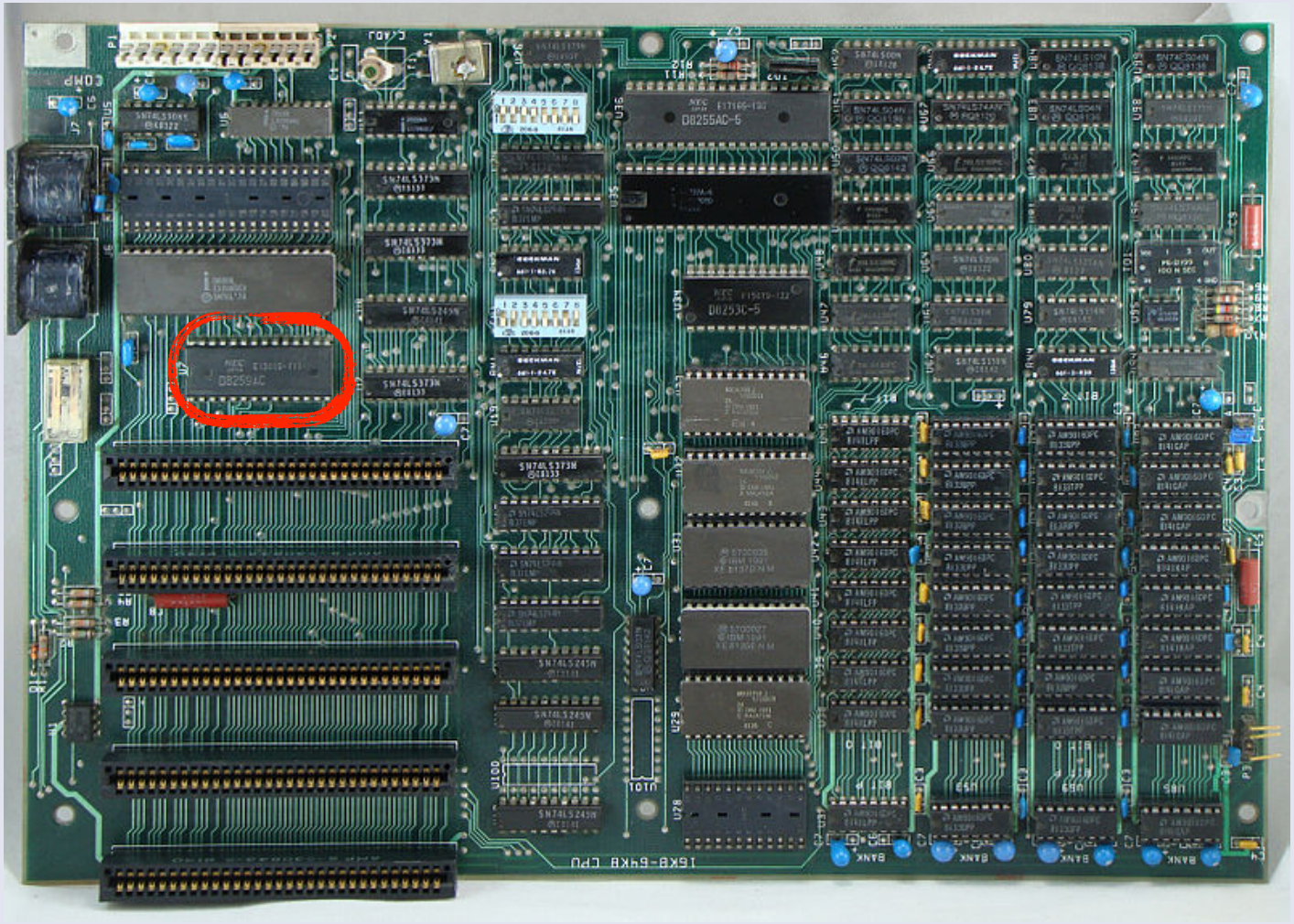




# Aufbau eines Computersystems

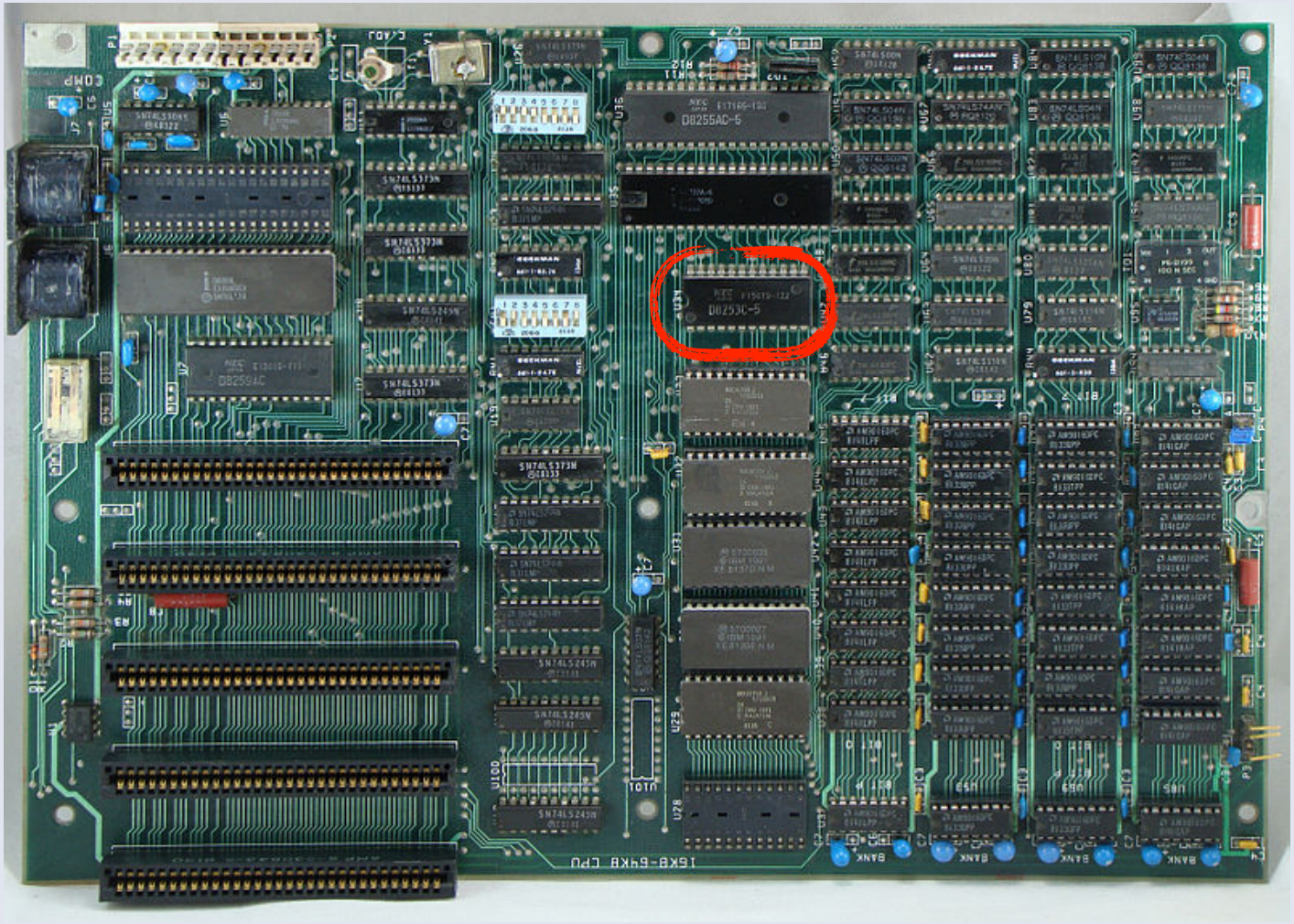


Interrupt-controller Intel 8259



# Aufbau eines Computersystems

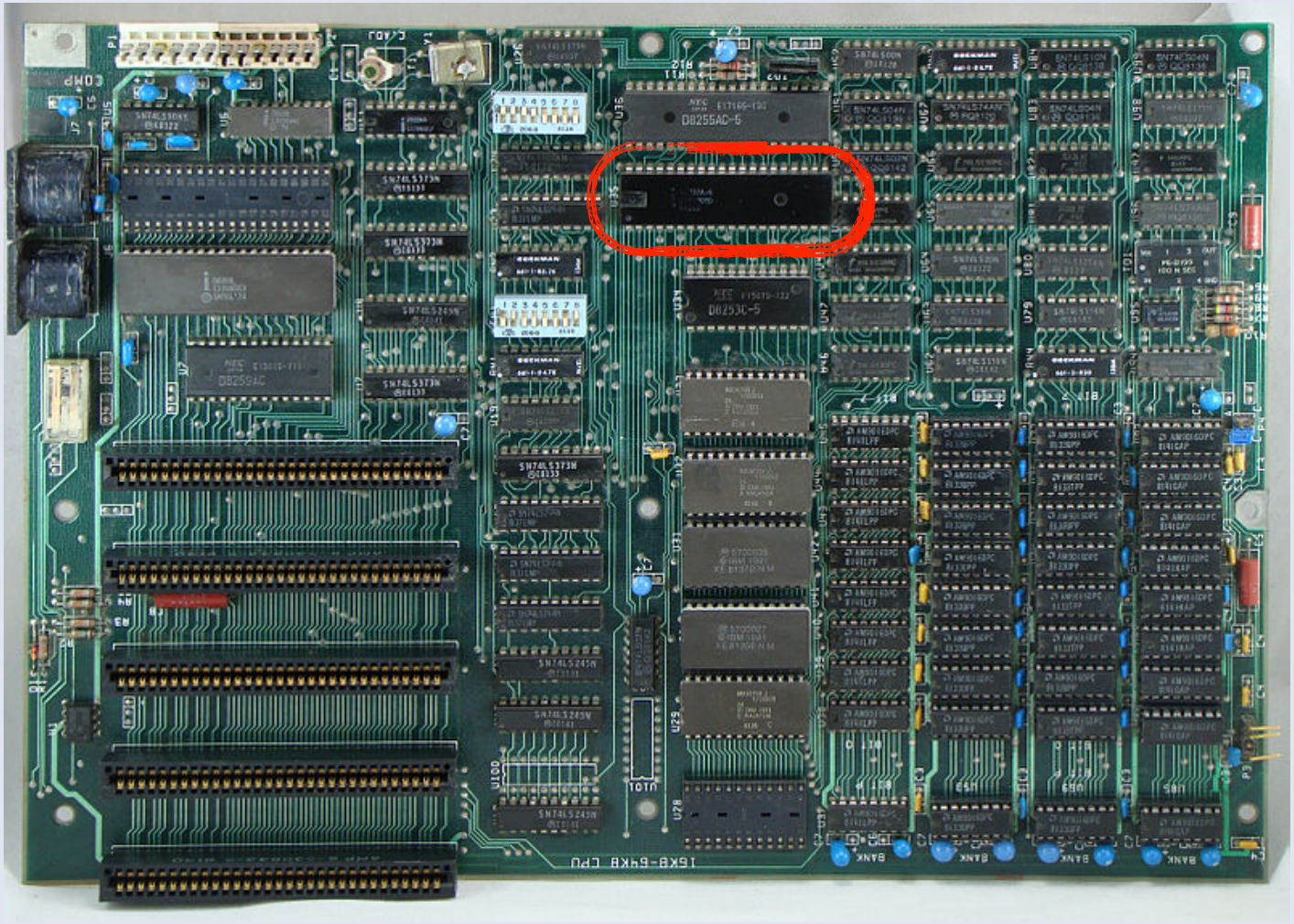





Zeitgeber
Intel 8253

# Aufbau eines Computersystems

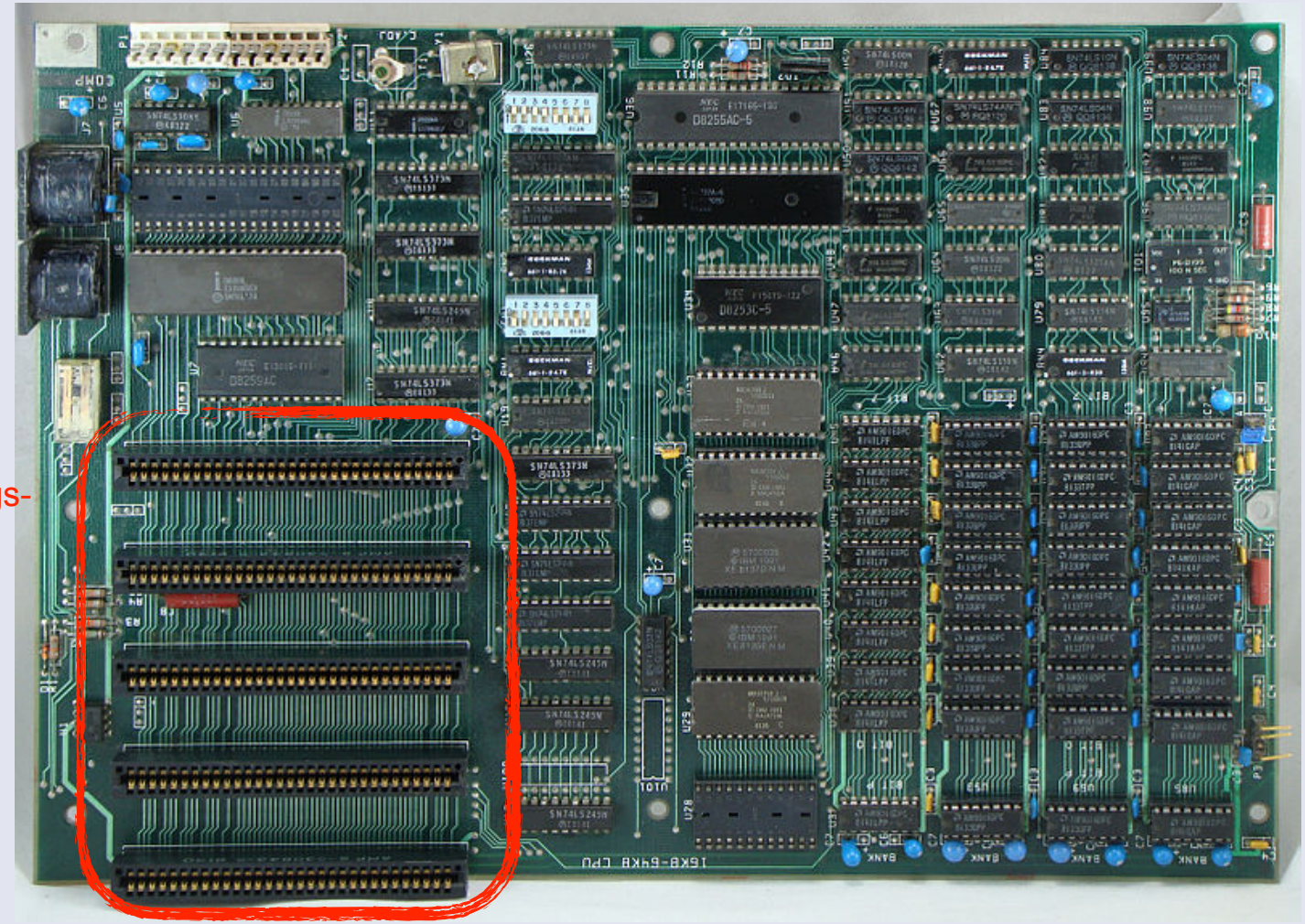





DMA-controller Intel 8237

# Aufbau eines Computersystems



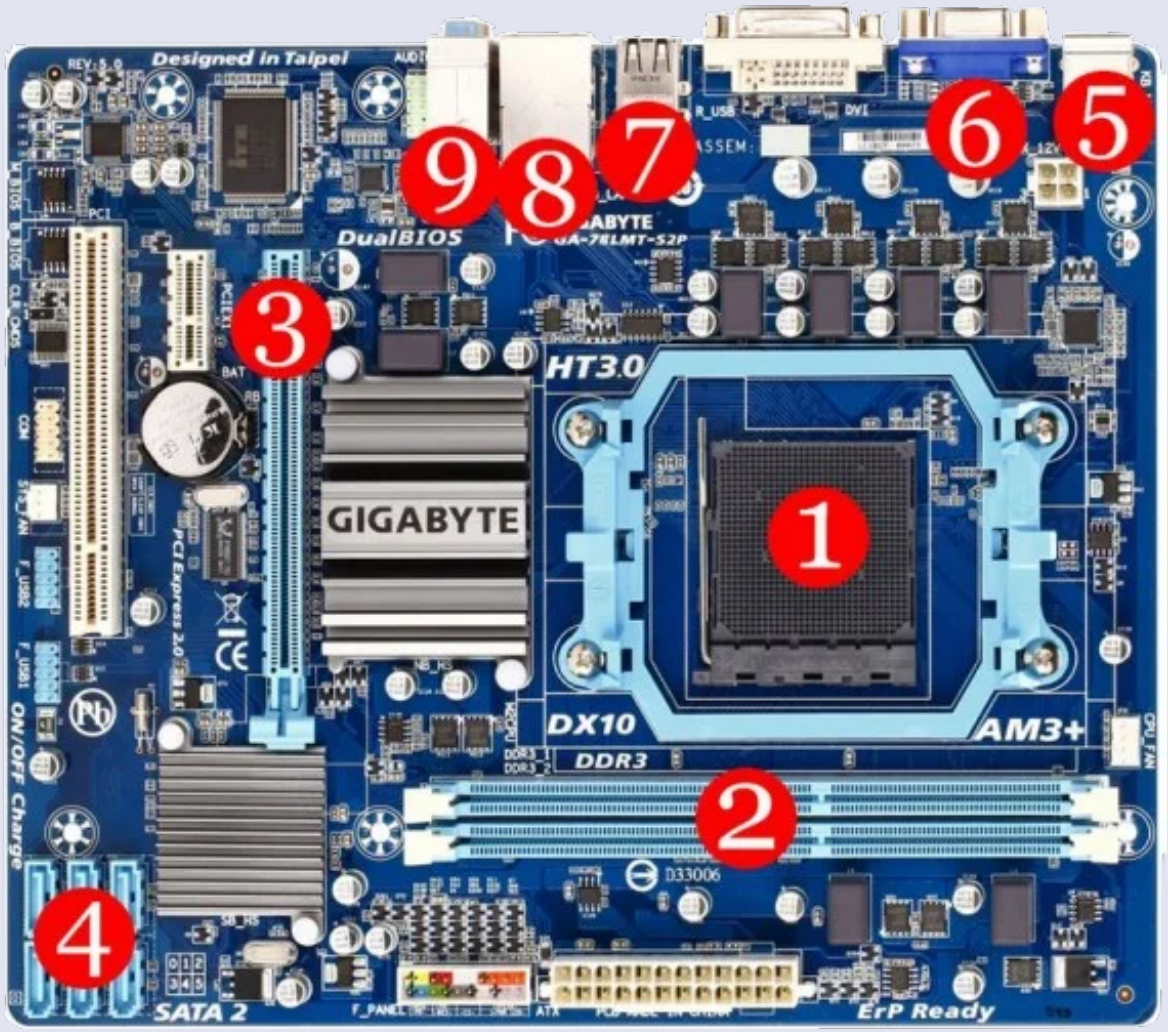



Erweiterungs-slots (ISA) für Grafik, Speicher, Disketten- und Fest-platten-controller, Netzwerk, Sound usw.

# Ein moderne(re)s PC-Mainboard

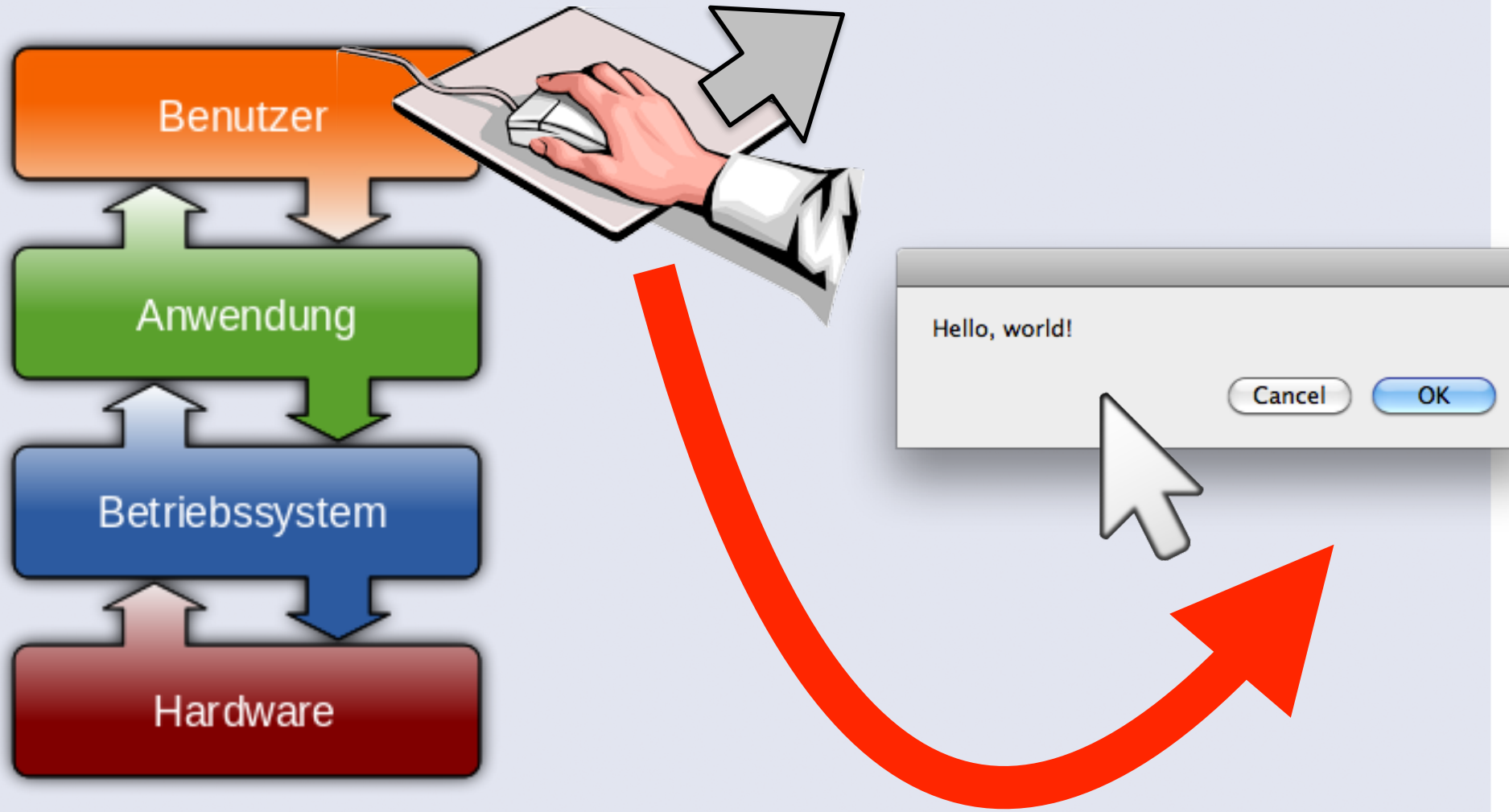
Universität Bamberg

1. CPU im Sockel
2. Slots für RAM-DIMMs
3. PCIe-Slots (Grafikkarte und Erweiterungen)
4. SATA-Steckverbinder für Festplatten usw.
5. Tastaturanschluß (PS/2)
6. Videoanschluß (VGA)
7. USB-Anschlüsse
8. Ethernet-Anschluß
9. Audio-Anschlüsse



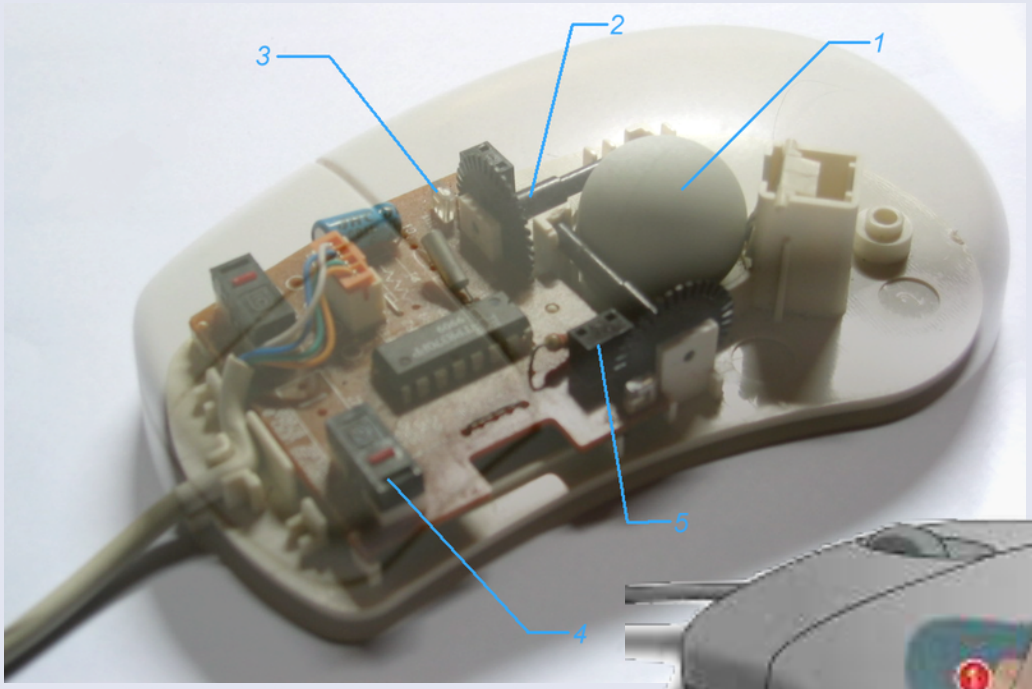
# Motivation: Wie funktioniert eine Maus?





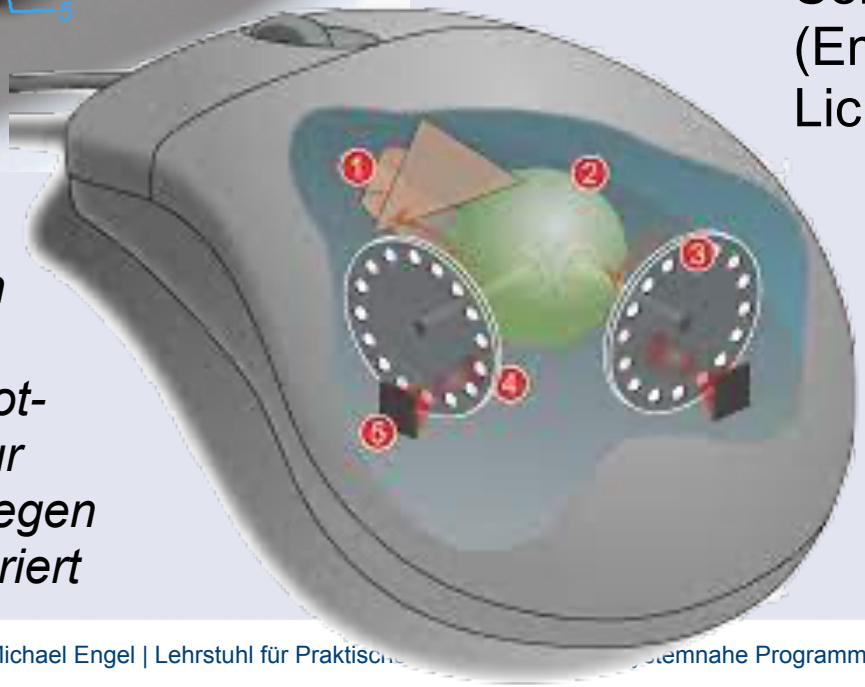
# Wir sezieren eine Maus…






1. Maus-Kugel
2. Schlitzscheiben
3. LED (Lichtquelle der Lichtschranke)
4. Taster
5. Optischer Sensor (Empfänger der Lichtschranke)

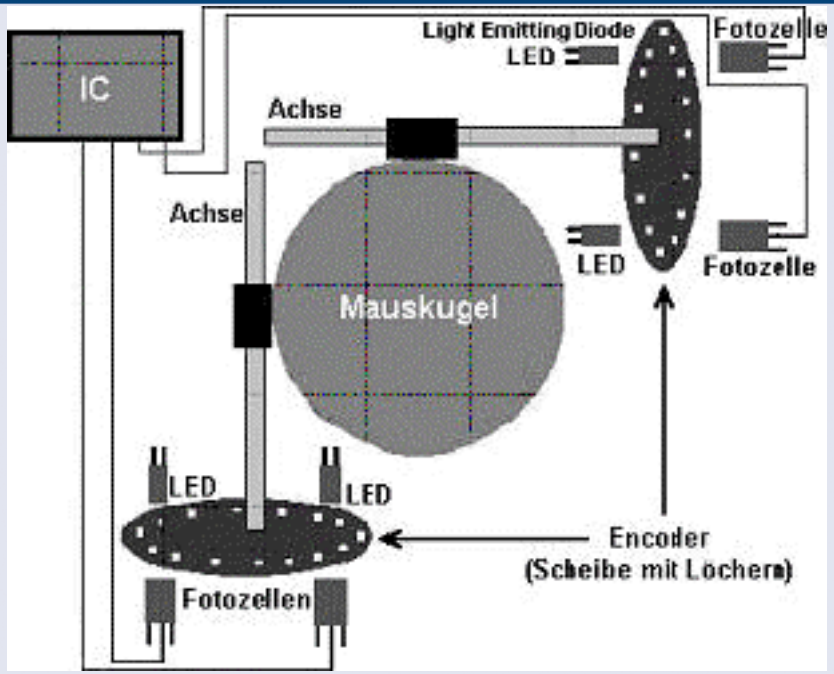





*Moderne Mäuse verwenden anstelle von Kugel und Lichtschranken einen Infrarot-Kamerachip, der die Struktur des Untergrunds beim Bewegen filmt und Änderungen registriert*

# Eine Maus sezieren…

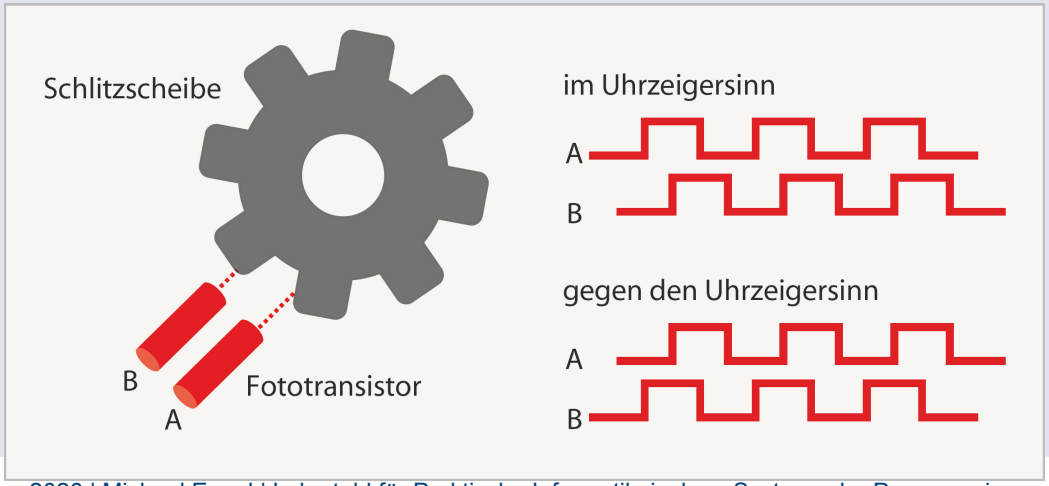


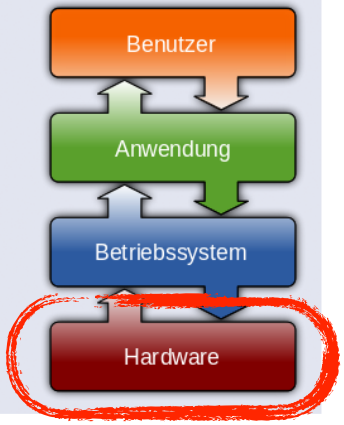



Durch die Bewegung der Maus wird die Kugel bewegt

- Horizontale und vertikale Teilvektoren der Bewegungs-richtung werden durch die Lichtschranken erfasst
- Zwei Lichtschranken pro Dimension (H/V) ermöglichen Unterscheidung der Richtung (links/rechts bzw. oben/unten)

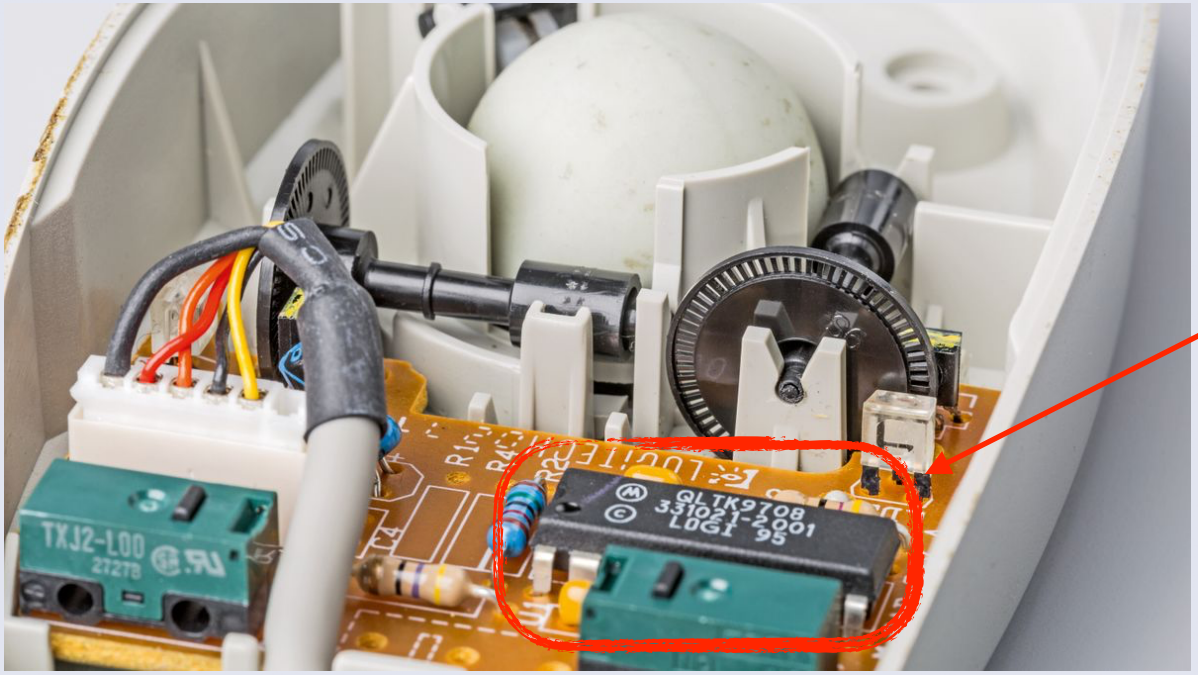
*Die ersten PCs mit grafischer Benutzer-schnittstelle (Apple Mac, Atari ST, Amiga) haben die "Rechtecksignale" der Maus direkt ausgewertet*



# Moderne Mäuse sind intelligent!

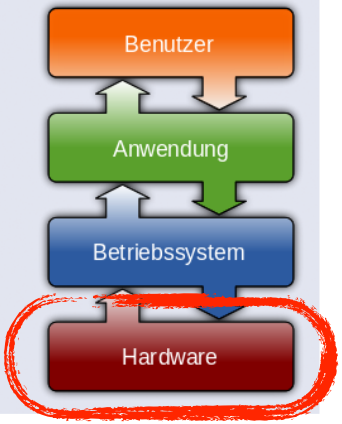





Modernere Mäuse verwenden einen eigenen kleinen Computer auf einem Chip (***Mikrocontroller***), der die Rechtecksignale der ***Sensoren*** in ***Signale*** und ***Protokolle*** einer Standard-***Schnittstelle*** übersetzt

*Es gab im Laufe der Zeit eine Reihe von Mausschnittstellen:*

- *Busmaus – die "rohen" Rechtecksignale der vorigen Folie*
- *Seriell – Mausbewegung als 3-5 Bytes über RS232-Schnittstelle*
- *PS/2 – synchrone serielle Schnittstelle (statt RS232)*
- *Apple Desktop Bus (ADB) – Mausschnittstelle an alten Macs*
- ***heute:** meist **USB** oder drahtlos (z.B. über Bluetooth)*



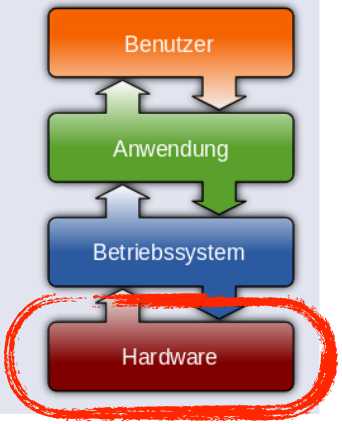
# Die Maus redet mit dem Computer



Welche Information muss die Maus dem Computer mitteilen?

- **Position der Maus** in horizontaler (x) und vertikaler (y) Richtung
  - *Mäuse haben ein schlechtes Gedächtnis:* Positionsangaben der Maus werden ***relativ*** zu der vorher gesendeten Position gesendet (kein Speicher notwendig)
  - Gesendeter Wert ist ***Differenz*** in x- und y-Richtung: positiver Wert = "rechts/runter", negativer Wert = "links/hoch"

- **Zustand der Maustasten**:
  - Gedrückt oder nicht gedrückt für 1–3 Tasten




*Die genauen Protokolle über die verschiedenen Schnittstellen unterscheiden sich in Details und haben Erweiterungen, z.B. für weitere Tasten, Scrollräder, usw.*

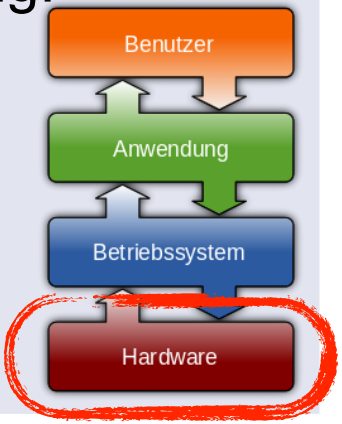
# Die Maus redet mit dem Computer



- **Position der Maus** in horizontaler (x) und vertikaler (y) Richtung
  - Gesendeter Wert ist ***Differenz*** in x- und y-Richtung (dx/dy): positiver Wert = "rechts/runter", negativer Wert = "links/hoch"

***Wie groß kann die Differenz werden?***

- Abhängig von der ***Auflösung*** der Maus, in Werte oder Punkte pro Zoll (*dots per inch,* dpi) – 1 Zoll = 2,54 cm
  - "Büro"mäuse haben z.B. 400 dpi, also ändert sich die Mausposition um ca. den Wert 16 für 1 mm Bewegung!
  - Gamingmäuse haben deutlich höhere Auflösungen, z.B. 2000-4000 dpi!
- Mäuse verwenden typischerweise Werte von -256 bis +255 für die x- und y-Differenzen dx und dy



# Die Maus redet mit dem Computer

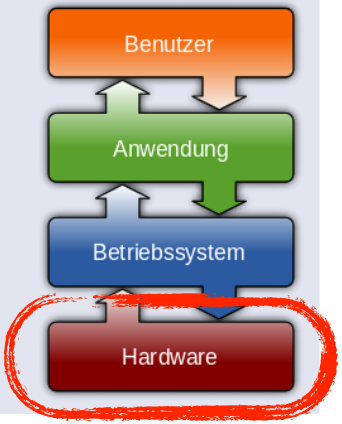
Universität Bamberg

- **Zustand der Maustasten**:
  - Gedrückt oder nicht gedrückt für 1–3 Tasten
- Für die Information, ob eine Maustaste gerade gedrückt wird, reicht ***1 Bit*** an Information aus
  - Der Wert des Bits ist z.B.
    - = 0 für "nicht gedrückt"
    - = 1 für "gedrückt"
  - Insgesamt also z.B. 3 Bits für drei Maustasten

Wir sehen in Vorlesung 3, wie **positive** und **negative** Zahlen mit Bits ***codiert*** werden können

Ein mögliches ***Protokoll*** für die Mausdaten (hier: PS/2):

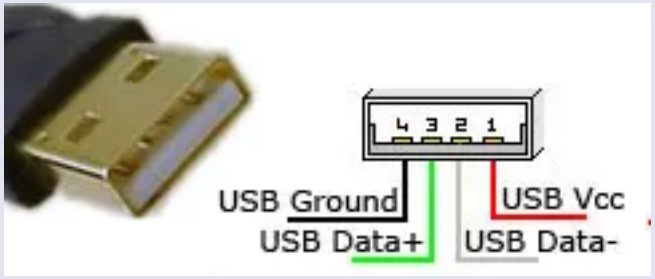
| Byte/ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **1** | Yovfl | Xovfl | dy8 | dx8 | 1 | Mittlere Taste | Rechte Taste | Linke Taste |
| **2** | dx7 | dx6 | dx5 | dx4 | dx3 | dx2 | dx1 | dx0 |
| **3** | dy7 | dy6 | dy5 | dy4 | dy3 | dy2 | dy1 | dy0 |



# Die Maus redet mit dem Computer

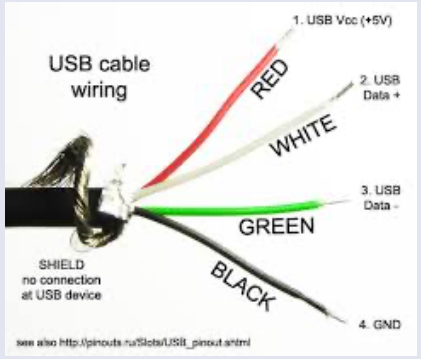
Universität Bamberg

PS/2



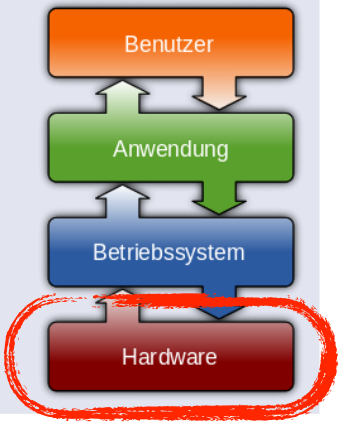

USB

Seriell/RS232




**Problem:**
Wir senden drei **Bytes** an Information (3 x 8 Bit), aber typische Mauskabel haben deutlich weniger als 8 Leitungen…

**Lösung:**
Die Bits jedes Bytes werden ***seriell***, also ein Bit nach dem anderen, an den Computer gesendet!



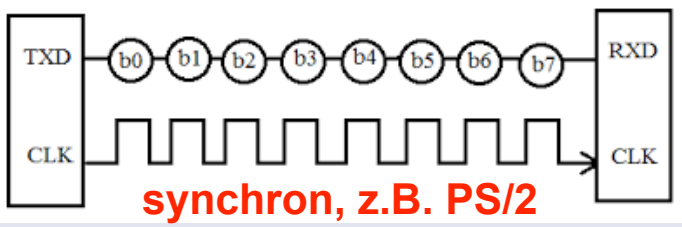

| Byte/Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **1** | Yovfl | Xovfl | dy8 | dx8 | 1 | Mittlere Taste | Rechte Taste | Linke Taste |
| **2** | dx7 | dx6 | dx5 | dx4 | dx3 | dx2 | dx1 | dx0 |
| **3** | dy7 | dy6 | dy5 | dy4 | dy3 | dy2 | dy1 | dy0 |

# Serielle Datenübertragung

Universität Bamberg

Bei serieller Datenübertragung werden ***Bits zeitlich nacheinander*** gesendet

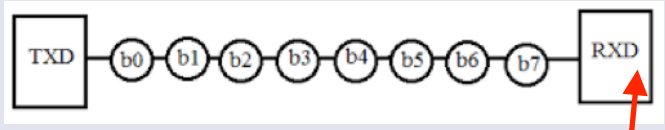
- entweder Bit 0 zuerst
  *("Least significant bit first", LSB first)*
- oder das "höchste" Bit zuerst
  *("Most significant bit first", MSB first)*
- Sender und Empfänger müssen sich über die Reihenfolge einig sein!




**synchron, z.B. PS/2**

Es muss auch bekannt sein, **wann das nächste Bit beginnt**

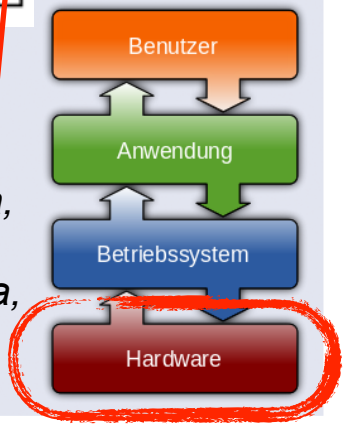
- entweder durch ein zusätzliches Signal ("**CLK**" = clock oder Takt): ***synchrone Datenübertragung***
- oder Sender und Empfänger müssen sich auf eine einheitliche Dauer jedes Bits einigen: ***asynchrone Datenübertragung***




**asynchron, z.B. RS232**
**Dauer von b0…b7 konstant**

| Byte/Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **1** | Yovfl | Xovfl | dy8 | dx8 | 1 | Mittlere Taste | Rechte Taste | Linke Taste |
| **2** | dx7 | dx6 | dx5 | dx4 | dx3 | dx2 | dx1 | dx0 |
| **3** | dy7 | dy6 | dy5 | dy4 | dy3 | dy2 | dy1 | dy0 |

***TXD*** *= Transmit Data, = Sender*
***RXD*** *= Receive Data, = Empfänger*



# Der PC hört zu



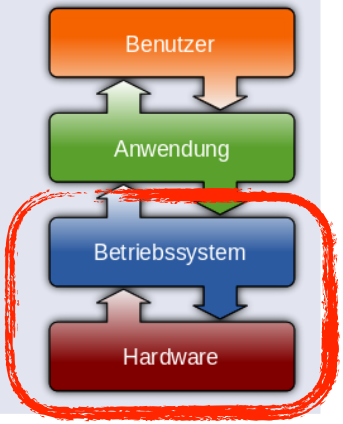
Der Stecker der Maus wird im PC mit einem ***Interface-Chip*** verbunden

- ein eigener Mikrocontroller (Intel 8042) für PS/2
- ein serieller Schnittstellenbaustein (z.B. Intel 16550) für RS232: "**UART**" – "Universal Asynchronous Receiver and Transmitter"

Daten der jeweiligen Maus lassen sich von Software aus ***Registern*** des zugehörigen Interface-Chips auslesen – jedes Register hat eine ***Adresse***, über die es angesprochen wird, jeder Chip hat mehrere Registeradressen:

- Mikrocontroller 8042 für PS/2: Adressen 96–100 (=0x0060–0x0064)
- UART 16550 für Maus: üblicherweise 1016–1023 (=0x03F8-0x03FF)

Zahlen, die mit **0x** beginnen, sind im **hexadezimalen Zahlensystem** notiert. Wir lernen dies in Kürze kennen!



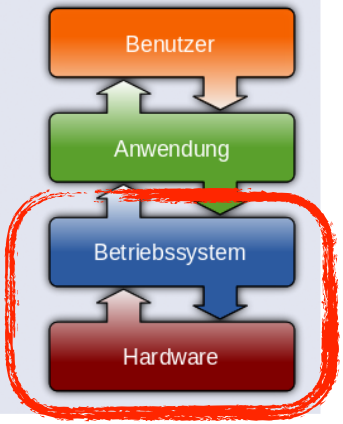
# Maus an PC!



Der UART-Baustein 16550 verfügt über 8 Register:

| Adresse | Funktion | Wichtiger Inhalt |
|---|---|---|
| **1016** | **Data Register** | **Daten von der Maus** |
| 1017 | Interrupt Enable Register | |
| 1018 | Interrupt Identification Reg. | |
| 1019 | Line Control Register | |
| 1020 | Modem Control Register | |
| **1021** | **Line Status Register** | **Bit 0 gibt an, ob neue Daten da sind** |
| 1022 | Modem Status Register | |
| 1023 | Scratch Register | |

*Wir haben hier einige Details weggelassen, der echte 16550 hat noch weitere Funktionen…*



# Betriebssystem fragt Maus (ab)…



```
Setze DataRegister = 1016                    // Adresse des Datenregister
Setze LineStatusRegister = 1021              // Adresse des Line Status Reg.
Setze xpos = 0; ypos = 0                     // Variablen für x- und y-Position
Setze taste_l = 0; taste_r = 0; taste_m = 0  // Maustasten

Funktion hierKommtDieMaus() {
  b1 = 0; b2 = 0; b3 = 0          // Drei Bytes zwischenspeichern

  Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
  Kopiere Wert aus DataRegister in b1          // Byte 1
  Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
  Kopiere Wert aus DataRegister in b2          // Byte 2
  Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
  Kopiere Wert aus DataRegister in b3          // Byte 3

  Wenn Bit 0 von b1 gesetzt, dann setze taste_l = 1
  Wenn Bit 1 von b1 gesetzt, dann setze taste_r = 1
  Wenn Bit 2 von b1 gesetzt, dann setze taste_m = 1

  Setze xpos = xpos+b2; Setze ypos = ypos+b3
}
```

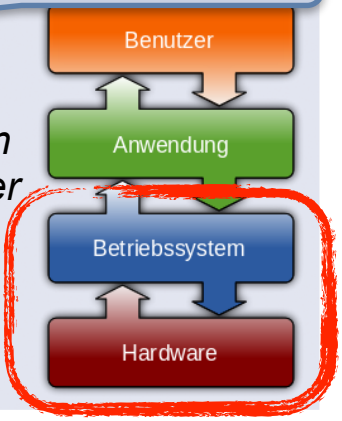
Was macht diese Zeile Code genau – **warum wird hier addiert**?

Gibt es einen **Fehler** in dieser Zeile?

Und: fehlt hier nicht noch etwas, wenn wir uns das Protokoll anschauen?

| Byte/Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 1 | Yovfl | Xovfl | dy8 | dx8 | 1 | Mittlere Taste | Rechte Taste | Linke Taste |
| 2 | dx7 | dx6 | dx5 | dx4 | dx3 | dx2 | dx1 | dx0 |
| 3 | dy7 | dy6 | dy5 | dy4 | dy3 | dy2 | dy1 | dy0 |

*In Betriebssystemen wird Code wie dieser **Gerätetreiber** genannt – dieser vermittelt zwischen Gerät und Betr.syst.*



# Däumchen drehn im Computer



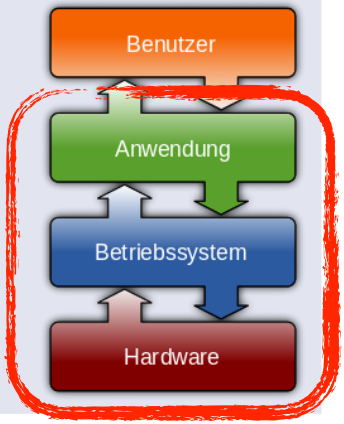
```
Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
Kopiere Wert aus DataRegister in b1         // Byte 1
Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
Kopiere Wert aus DataRegister in b2         // Byte 2
Warte, bis Bit 0 von LineStatusRegister = 1 // Auf Daten warten
Kopiere Wert aus DataRegister in b3         // Byte 3
```

Wir betreiben hier "**aktives Warten**" – ist das sinnvoll?

Unser einfacher Beispielcode wartet, bis das LineStatusRegister anzeigt, dass neue Daten von der Maus angekommen ist…

**Problem:**

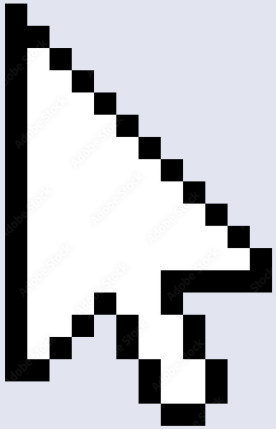
- Computer (also der Prozessor) ist **permanent mit Warten beschäftigt**!
  - Wir nennen dies "**polling**"
- Damit können (auf einem Einprozessorsystem) **nicht gleichzeitig andere Befehle** – z.B. von Anwendungsprogrammen – **ausgeführt werden**!
  - Wenn die Maus aber nicht abgefragt wird, könnten Daten von der Maus verloren gehen!
- ***Wir zeigen in Vorlesung 7, wie das besser geht!***



# Bits auf dem Bildschirm



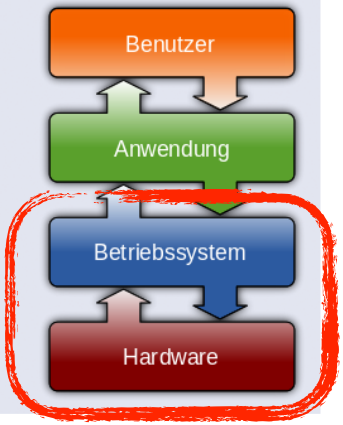
Der Mauszeiger ist eine ***bitmap***-Grafik, die auf dem Bildschirm vor anderen Elementen (Fenster, Icons, Hintergrund) an der jeweils aktuellen absoluten x/y-Position gezeichnet wird



- Die Bildschirmanzeige besteht aus einer Matrix von ***Pixeln***, die im einfachsten Fall schwarz oder weiß sind
  - Die ***Auflösung*** beträgt z.B. 1024 * 768 Pixel (x * y)
- Die einfachsten Mauszeiger sind ***monochrom*** (schwarz/weiß), benötigen also ein Bit pro Pixel

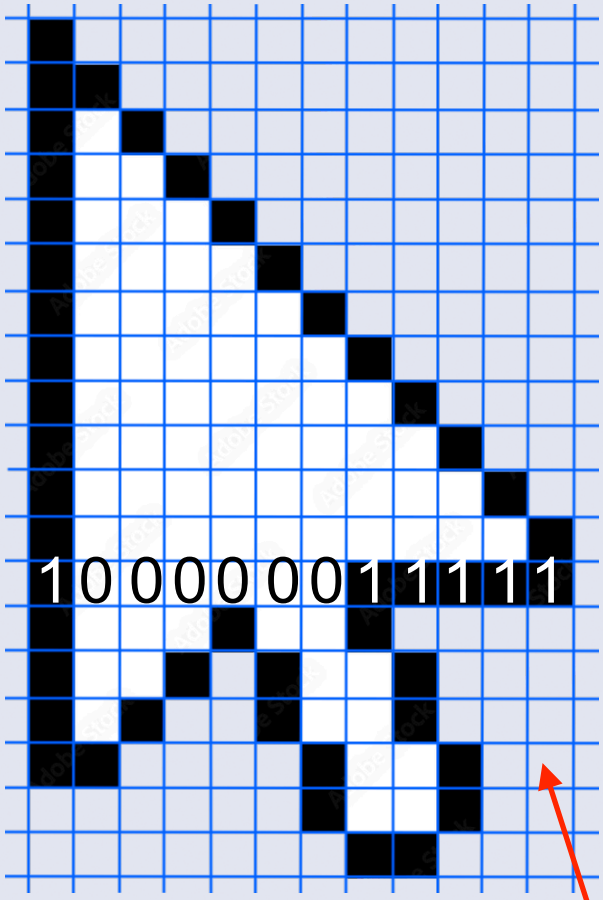
- Der Hintergrund – die Pixel "hinter" dem Mauszeiger – wird (meist) gespeichert, damit er beim Bewegen des Mauszeigers schnell wiederhergestellt werden kann

*Moderne Grafikkarten haben die Möglichkeit, den Mauszeiger durch ein Overlay (also eine weitere Pixelebene) darzustellen. Damit muss der Zeiger nicht dauernd "von Hand" neu gezeichnet werden.*



# Maus-Bitmap






Die dargestellte Zeile des Mauszeigers wird (mit 1 = "schwarz"/0 = "weiß") dargestellt durch die Bits (von links nach rechts):

**1 0 0 0 0 0 0 1 1 1 1 1**

**Bits im Bildschirmspeicher werden durch die Bits des Mauszeigers überschrieben**


***Frage zum Nachdenken:***
*Was ist mit den "transparenten" Bits, bei denen der Hintergrund durchscheint, die also nicht vom Mauszeiger überdeckt sind?*

# Fragen?